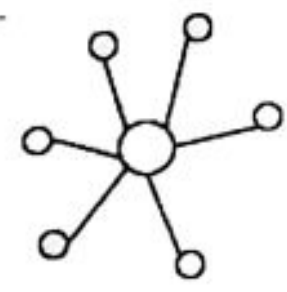


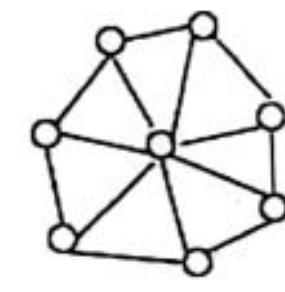



2079Baishakh

Q 1) Compare centralized and distributed system in terms of their advantages and disadvantages. Explain the importance of transparency in distributed system.

Ans:

Centralized System	Distributed System
 <p>  — Server/master  — Computer/slave </p>	 <p>  — Node/Computer </p>
Centralized systems are systems that use client/server architecture where one or more client nodes are directly connected to a central server.	In decentralized systems, every node makes its own decision.
This is the most used type of system in many organizations where a client sends a request to a company server and receives the response.	Google search system. Each request is worked upon by hundreds of computers which crawl the web and return the relevant results.
Limitations of Centralized System – <ul style="list-style-type: none"> • Can't scale up vertically after a certain limit – After a limit, even if you increase the hardware and software capabilities of the server node, the performance will not increase appreciably leading to a cost/benefit ratio < 1. • Bottlenecks can appear when the traffic spikes – as the server can only have a finite number of open ports to which can listen to connections from client nodes. So, when high traffic 	Limitations of Distributed System – <ul style="list-style-type: none"> • Difficult to design and debug algorithms for the system. These algorithms are difficult because of the absence of a common clock; so no temporal ordering of commands/logs can take place. Nodes can have different latencies which have to be kept in mind while designing such algorithms. The complexity increases with the increase in the number of nodes. Visit this link for more information • No common clock causes difficulty in the temporal ordering of events/transactions • Difficult for a node to get the global view of the

occurs like a shopping sale, the server can essentially suffer a Denial-of-Service attack or Distributed Denial-of-Service attack.

system and hence take informed decisions based on the state of other nodes in the system

Advantages of Centralized System –

- Easy to physically secure. It is easy to secure and service the server and client nodes by virtue of their location
- Smooth and elegant personal experience – A client has a dedicated system which he uses (for example, a personal computer) and the company has a similar system which can be modified to suit custom needs
- Dedicated resources (memory, CPU cores, etc)
- More cost-efficient for small systems up to a certain limit – As the central systems take fewer funds to set up, they have an edge when small systems must be built
- Quick updates are possible – Only one machine to update.
- Easy detachment of a node from the system. Just remove the connection of the client node from the server and voila! Node detached.

Advantages of Distributed System –

- Low latency than a centralized system – Distributed systems have low latency because of high geographical spread, hence leading to less time to get a response

Second part:

Transparency

Transparency "is the concealment from the user of the separation of components of a distributed system so that the system is perceived as a whole". Transparency in distributed systems is applied at several aspects such as :

Access Transparency – Local and Remote access to the resources should be done with same efforts and operations. It enables local and remote objects to be accessed using identical operations.

Location transparency – User should not be aware of the location of resources. Wherever is the location of resource it should be made available to him as and when required.

Migration transparency – It is the ability to move resources without changing their names.

Replication Transparency – In distributed systems to achieve fault tolerance, replicas of resources are maintained. The Replication transparency ensures that users cannot tell how many copies exist.

Concurrency Transparency – As in distributed systems, many users work concurrently, the resource sharing should happen automatically without the awareness of the users, except as by making the users.

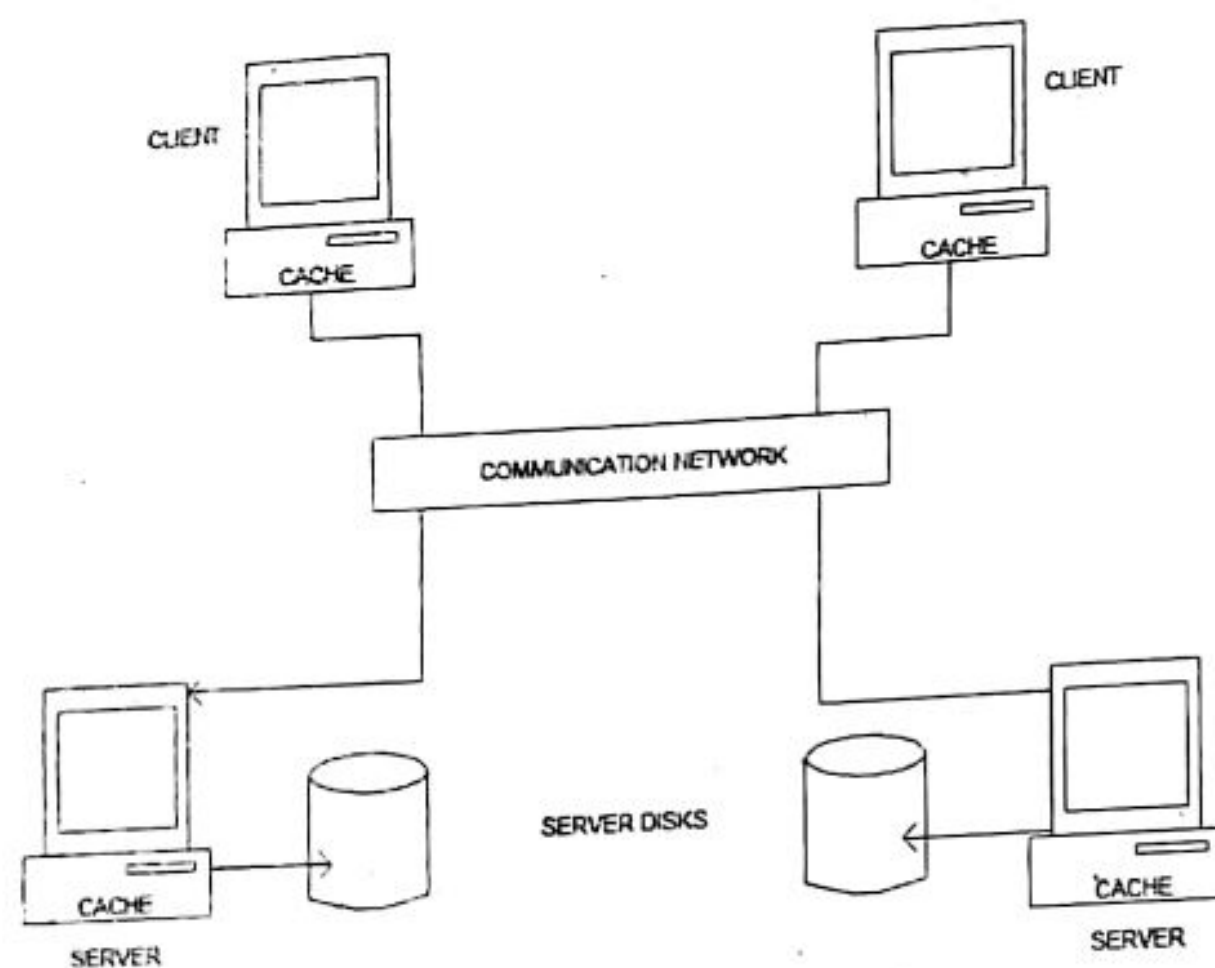
Failure Transparency – If users should be unaffected from partial failures. The system should cope up with partial failures without the users awareness.

Parallelism transparency – Activities can happen in parallel without users knowing.

Q2) What is distributed file system? Explain the architecture and operations of Sun NFS.

Ans: A Distributed File System (DFS) as the name suggests is a file system that is distributed on multiple servers or nodes in a network. The users program can access and store files as they do with the local nodes, allowing programmers to access files from any networked computer.

Architecture of a Distributed File System

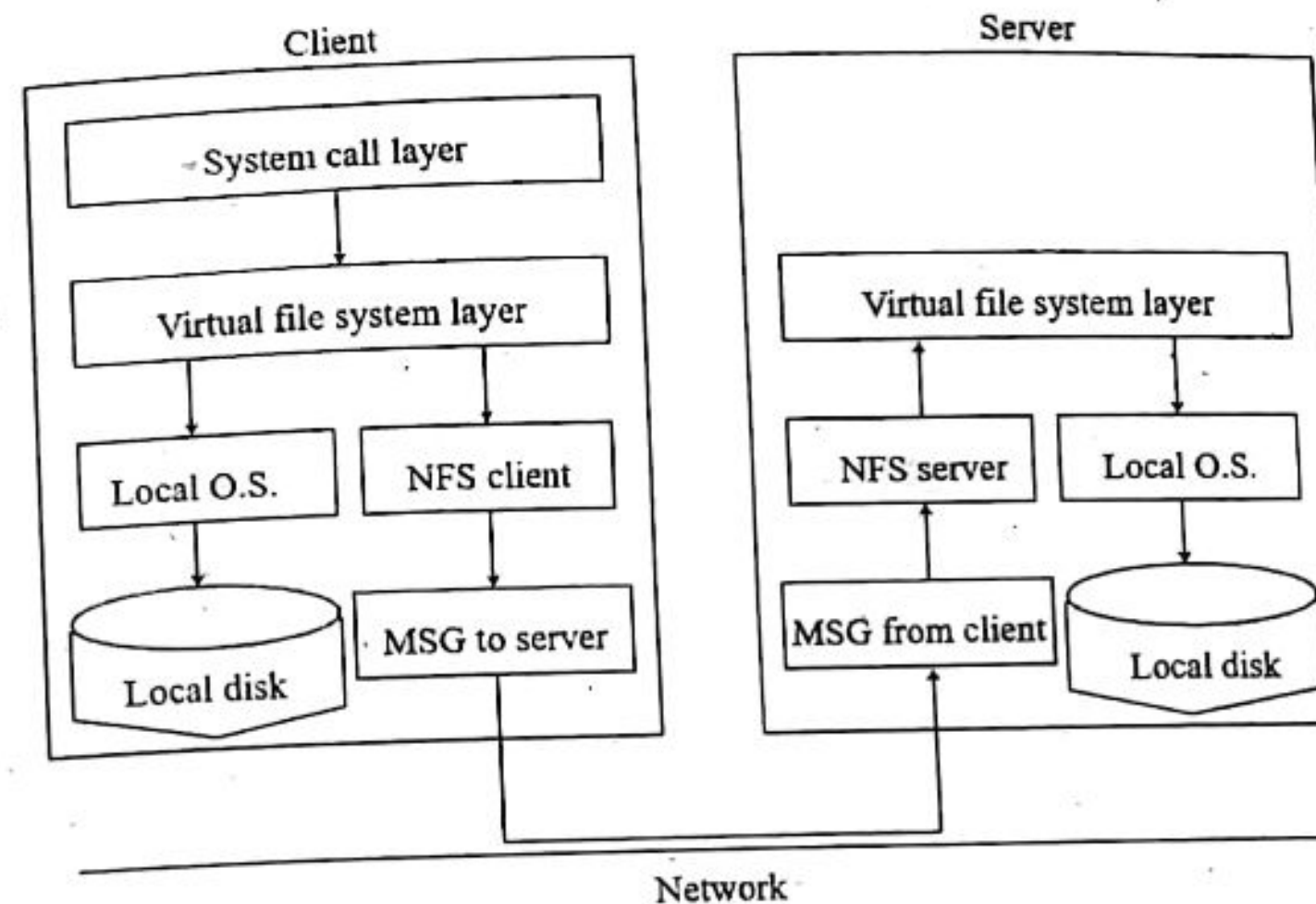


Second part:

Ans: The earliest successful distributed system could be attributed to Sun Microsystems, which developed the Network File System (NFS). NFSv2 was the standard protocol followed for many years, designed with the goal of simple and fast server crash recovery. This goal is of utmost importance in multi-client and single server-based network architectures because a single instant of server crash means that all clients are un-served. The entire system goes down.

The major features of SUN NFS are:

- Machine and Operating System Independence:** The protocols used should be independent of UNIX so that an NFS server can supply files to many different types of clients. The protocols should also be simple enough that they can be implemented on low-end machines like the PC.
- Crash Recovery:** When clients can mount remote filesystems from many different servers it is very important that clients and servers be able to recover easily from machine crashes and network problems.
- Transparent Access:** We want to provide a system which allows programs to access remote files in the same way as local files, without special pathname parsing, libraries, or recompiling. Programs should not need or be able to tell whether a file is remote or local.
- UNIX Semantics Maintained on UNIX Client:** For transparent access to work on UNIX machines, UNIX filesystem semantics have to be maintained for remote files.
- Reasonable Performance:** People will not use a remote filesystem if it is no faster than the existing networking utilities, such as rcp, even if it is easier to use. Our design goal was to make NFS as fast as a small local disk on a SCSI interface.



Basic Design

The NFS design consists of three major pieces: the protocol, the server side and the client side.

NFS Protocol:

NFS uses a stateless protocol. The NFS protocol is defined in terms of a set of procedures, their arguments and results, and their effects. Remote procedure calls are synchronous, that is, the client application blocks until the server has completed the call and returned the results. This makes RPC very easy to use and understand because it behaves like a local procedure call.

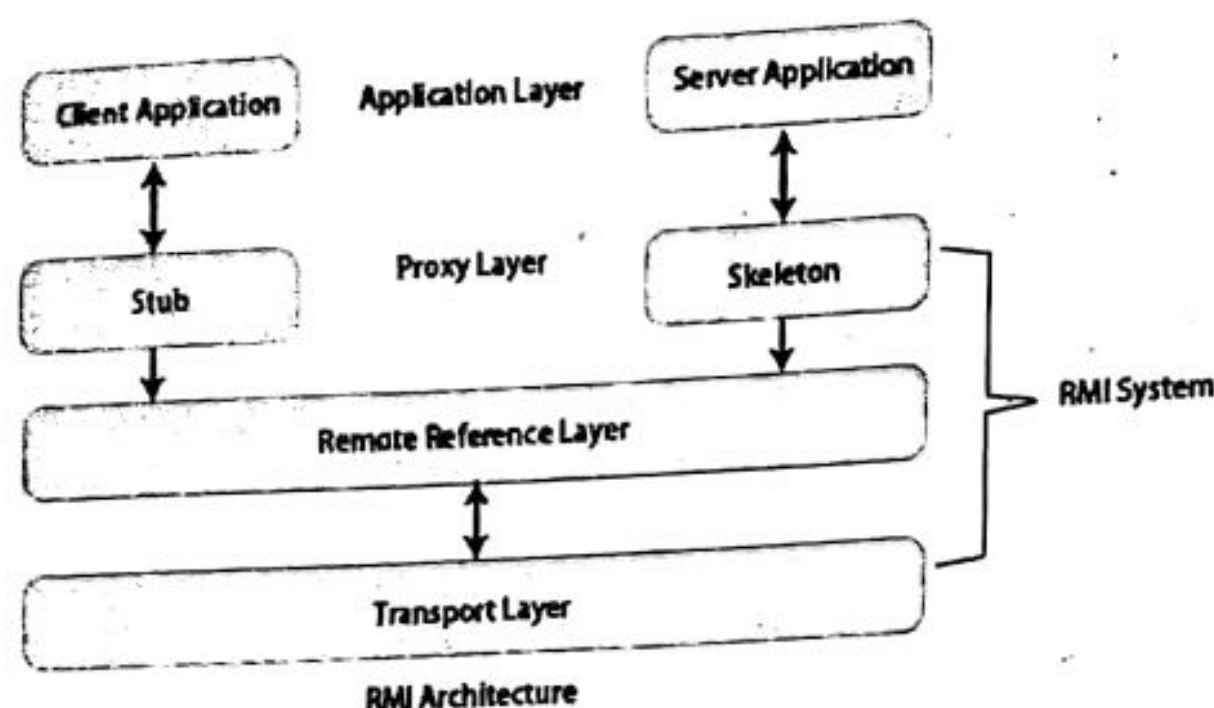
The Server side:

Because the NFS server is stateless, when servicing an NFS request it must commit any modified data to stable storage before returning results. The implication for UNIX based servers is that requests which modify the filesystem must flush all modified data to disk before returning from the call. For example, on a write request, not only the data block, but also any modified indirect blocks and the block containing the inode must be flushed if they have been modified.

The Client Side:

The Sun implementation of the client side provides an interface to NFS which is transparent to applications. To make transparent access to remote files work we had to use a method of locating remote files that does not change the structure of path names. Some UNIX based remote file access methods use pathnames like host: path or /.../host/path to name remote files. This does not allow real transparent access since existing programs that parse pathnames have to be modified.

Q3) How RMI helps in distributed programming model? Describe the strengths and weakness of RPC.



RMI stands for Remote Method Invocation. It is an API provided by java that allows an object residing in one JVM (Java Virtual Machine) to access or invoke an object running on another JVM. The other JVM could be on the same machine or remote machine.

Let us now discuss the components of this architecture.

- Transport Layer – This layer connects the client and the server. It manages the existing connection and sets up new connections.
- Stub – A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- Skeleton – This is the object which resides on the server side. stub communicates with this skeleton to pass request to the remote object.
- RRL (Remote Reference Layer) – It is the layer which manages the references made by the client to the remote object.

Following are the features of RMI –

- To minimize the complexity of the application.
- To preserve type safety.
- Distributed garbage collection.
- Minimize the difference between working with local and remote objects.

Second part:

Advantages of RPC:

- Server independent.
- Process-oriented and thread oriented models supported by RPC.

- The development of distributed systems is simple because it uses straightforward semantics and easier.
- Like the common communications between the portions of an application, the development of the procedures for the remote calls is quite general.
- The procedure calls preserves the business logics which is apt for the application.
- The code re-writing / re-developing effort is minimized.
- Enables the usage of the applications used in the distributed environment, not only in the local environment.

- RPC provides interoperability between CORBA ORB implementations.
- A lightweight RPC protocol permits efficient implementations.

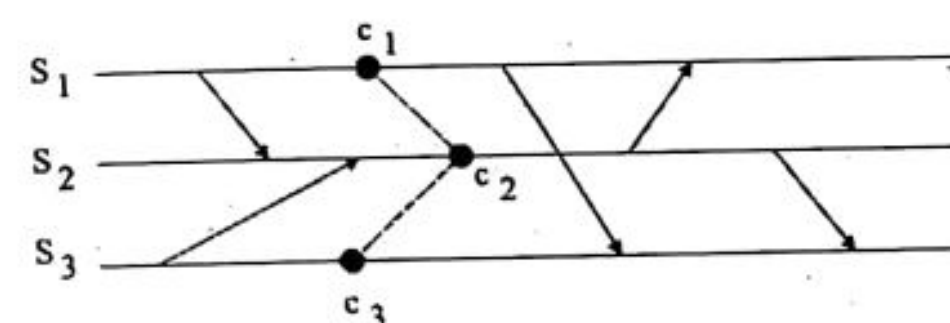
Disadvantages of RPC:

- Context switching increases scheduling costs
- RPC is not a standard – it is an idea that can be implemented in many ways
- RPC does not solve most of the distribution creation problems
- RPC is only interaction based. This does not offer any flexibility in terms of hardware architecture
- RPC implementations are not yet mature.
- It requires the TCP/IP protocol. Other transport protocols are not supported yet.
- Not yet proven over wide-area networks.

Q 4) Define the term distributed cut along with its types. Explain how clock synchronization can be done using vector clock method.

Ans

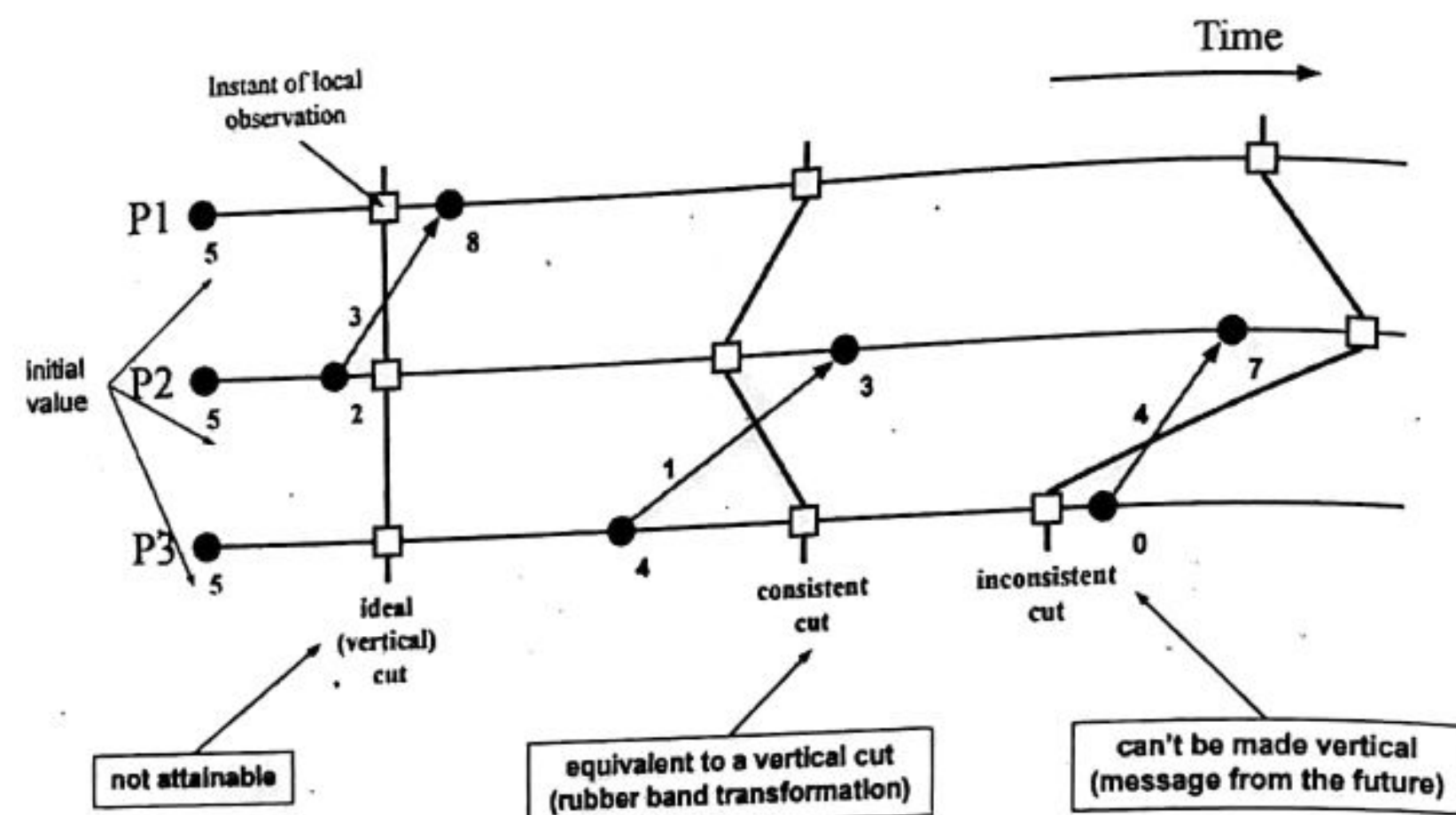
A cut is a set of cut events, one per node, each of which captures the state of the node on which it occurs.



$$C = \{c1, c2, c3\}$$

Consistent Cut:

A cut $C = \{c1, c2, c3, \dots\}$ is consistent if for all sites there are no events e_i and e_j such that: $(e_i \rightarrow e_j)$ and $(e_j \rightarrow c_j)$ and $(e_i \rightarrow c_i)$.



Inconsistent Cut:

A cut $C = \{c_1, c_2, c_3, \dots\}$ is inconsistent if for all sites there are events e_i and e_j such that: $(e_i \rightarrow e_j)$ and $(e_j \rightarrow c_j)$ and $(e_i \not\rightarrow c_i)$.

Second part:

Vector Clock is an algorithm that generates partial ordering of events and detects causality violations in a distributed system. These clocks expand on Scalar time to facilitate a causally consistent view of the distributed system, they detect whether a contributed event has caused another event in the distributed system. It essentially captures all the causal relationships. This algorithm helps us label every process with a vector (a list of integers) with an integer for each local clock of every process within the system. So for N given processes, there will be vector/ array of size N .

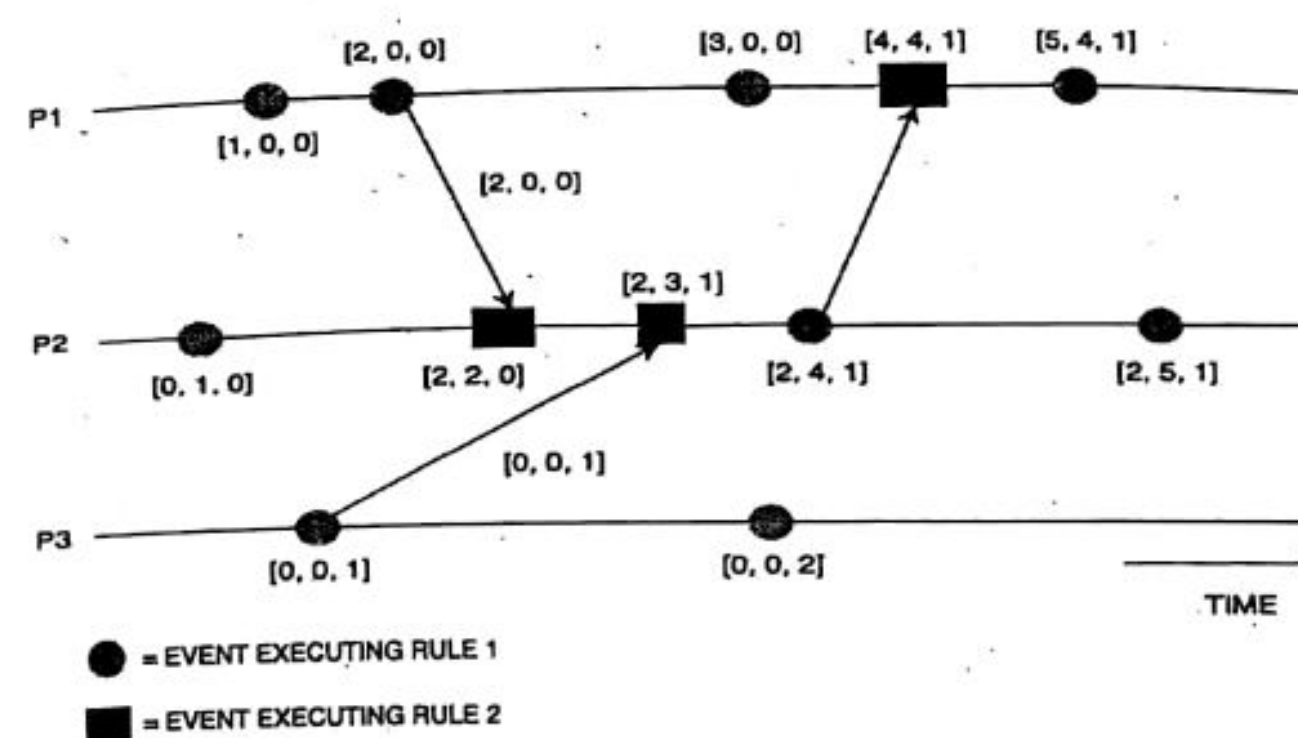
How does the vector clock algorithm work:

- Initially, all the clocks are set to zero.
- Every time, an Internal event occurs in a process, the value of the processes' logical clock in the vector is incremented by 1
- Also, every time a process sends a message, the value of the processes's logical clock in the vector is incremented by 1.

Every time, a process receives a message, the value of the processes's logical clock in the vector is incremented by 1, and moreover, each element is updated by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

Example:

Consider a process (P) with a vector size N for each process: the above set of rules mentioned are to be executed by the vector clock:



The above example depicts the vector clocks mechanism in which the vector clocks are updated after execution of internal events, the arrows indicate how the values of vectors are sent in between the processes (P1, P2, P3).

To sum up, Vector clocks algorithms are used in distributed systems to provide a **causally consistent** ordering of events but the entire Vector is sent to each process for every message sent, in order to keep the vector clocks in sync.

Q5) Why is election algorithm needed in distributed system? Explain with example how coordinator is selected distributed system.

Ans: Importance of election algorithm:

- Many distributed algorithms require one process to act as coordinator, initiator, sequencer, or otherwise perform some special role.

[Example: the coordinator in the centralized mutual exclusion algorithm.

- Problem: look at algorithms for electing a coordinator. The algorithms differ in the way they do the location.

[Remark: If all processes are exactly the same, with no distinguishing characteristics, there is no way to select one of them to be special.

Second part:

We have two election algorithms for two different configurations of distributed system.

1. The Bully Algorithm –

This algorithm applies to system where every process can send a message to every other process in the system.

Algorithm – Suppose process P sends a message to the coordinator.

- If coordinator does not respond to it within a time interval T , then it is assumed that coordinator has failed.
- Now process P sends election message to every process with high priority number.

3. It waits for responses, if no one responds for time interval T then process P elects itself as a coordinator.
4. Then it sends a message to all lower priority number processes that it is elected as their new coordinator.
5. However, if an answer is received within time T from any other process Q,
 - (I) Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.
 - (II) If Q doesn't respond within time interval T' then it is assumed to have failed and algorithm is restarted.

2. The Ring Algorithm –

This algorithm applies to systems organized as a ring (logically or physically). In this algorithm we assume that the link between the processes are unidirectional and every process can message to the process on its right only. Data structure that this algorithm uses is **active list**, a list that has priority number of all active processes in the system.

Algorithm –

1. If process P1 detects a coordinator failure, it creates new active list which is empty initially. It sends election message to its neighbour on right and adds number 1 to its active list.
2. If process P2 receives message elect from processes on left, it responds in 3 ways:
 - (I) If message received does not contain 1 in active list then P1 adds 2 to its active list and forwards the message.
 - (II) If this is the first election message it has received or sent, P1 creates new active list with numbers 1 and 2. It then sends election message 1 followed by 2.
 - (III) If Process P1 receives its own election message 1 then active list for P1 now contains numbers of all the active processes in the system. Now Process P1 detects highest priority number from list and elects it as the new coordinator.

Q 6) How do you ensure high available services in distributed system. Describe various consistency models applicable in distributed system.

Ans:

High availability (HA) can be achieved when systems are equipped to operate continuously without failure for a long duration of time. The phrase suggests that parts of a system have been thoroughly tested and accommodated with redundant components to remain functional when a failure occurs.

1. Middleware HA — load balancer

A few high-availability solutions exist such as load balancing and basic clustering. Load balancing also known as horizontal scaling can be achieved by adding new nodes with identical functionality to existing ones, redistributing the load amongst them all. By exclusively redirecting requests to available servers, reliability and availability can be maintained.

2. Scaling up and down

In companies, high availability is achieved by scaling the servers up or down depending on the application server's load and availability. It is done mostly outside the application at the server level.

3. Implementing multiple application servers

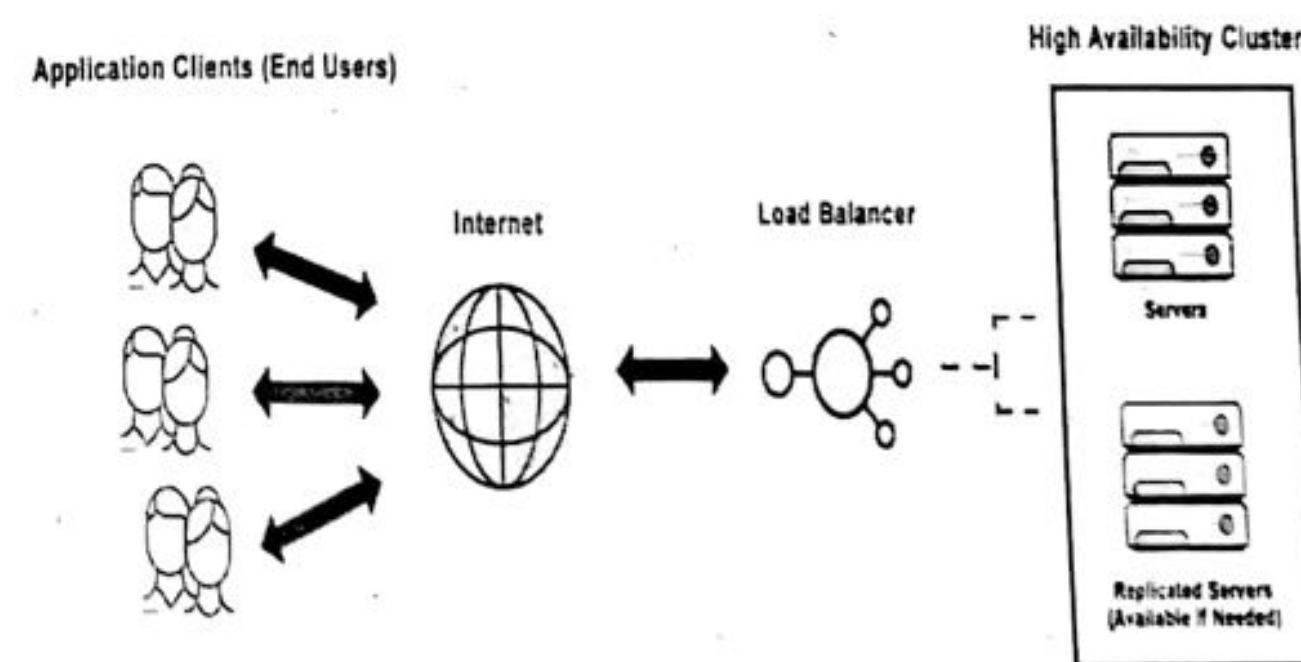
Overburdened servers may crash and cause an outage; it is advisable to deploy applications over multiple servers to keep the applications running all the time. It creates a sense of being always operational.

4. Multi-region deployments

When it comes to cloud environments, systems are deployed in units and are referred to as regions. A region can be defined as a data center, or it may be consisting of a set of data centers located somewhat close to each other. Then there comes a more granular unit inside of the regions and is known as availability zone. So, each availability zone is a single data center within one region.

5. Clustering techniques

Clustering techniques are generally used to improve and increase the performance and availability of a complex system. A cluster is usually designed as a redundant set of services rendering the same set of functionalities and capabilities.



Q 7) Describe how a non-recoverable situation could arise if write locks are released after the last operation of a transaction but before its commitment.

Ans:

Locks

When one thread of control wants to obtain access to an object, it requests a *lock* for that object. This lock is what allows DB to provide your application with its transactional isolation guarantees by ensuring that:

no other thread of control can read that object (in the case of an exclusive lock), and

no other thread of control can modify that object (in the case of an exclusive or non-exclusive lock).

Lock Resources

When locking occurs, there are conceptually three resources in use:

The locker.

This is the thing that holds the lock. In a transactional application, the locker is a transaction handle. For non-transactional operations, the locker is a cursor or a Db handle.

The lock.

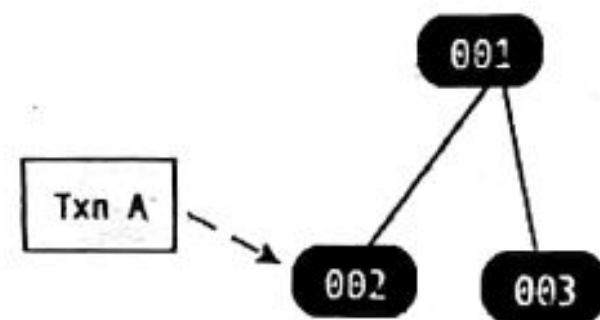
This is the actual data structure that locks the object. In DB, a locked object structure in the lock manager is representative of the object that is locked.

The locked object.

The thing that your application actually wants to lock. In a DB application, the locked object is usually a database page, which in turn contains multiple database entries (key and data). However, for Queue databases, individual database records are locked.

You can configure how many total lockers, locks, and locked objects your application is allowed to support. See Configuring the Locking Subsystem for details.

The following figure shows a transaction handle, Txn A, that is holding a lock on database page 002. In this graphic, Txn A is the locker, and the locked object is page 002. Only a single lock is in use in this operation.



Types of Locks

DB applications support both exclusive and non-exclusive locks. *Exclusive locks* are granted when a locker wants to write to an object. For this reason, exclusive locks are also sometimes called *write locks*.

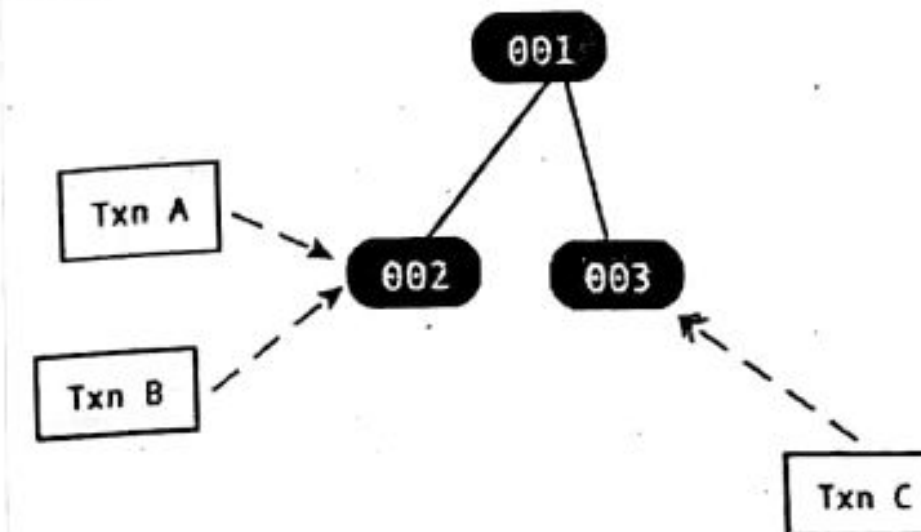
An exclusive lock prevents any other locker from obtaining any sort of a lock on the object. This provides isolation by ensuring that no other locker can observe or modify an exclusively locked object until the locker is done writing to that object.

Non-exclusive locks are granted for read-only access. For this reason, non-exclusive locks are also sometimes called *read locks*. Since multiple lockers can simultaneously hold read locks on the same object, read locks are also sometimes called *shared locks*.

A non-exclusive lock prevents any other locker from modifying the locked object while the locker is still reading the object. This is how transactional cursors are able to achieve repeatable reads; by default, the cursor's transaction holds a read lock on any object that the cursor has examined until such a time as the

transaction is committed or aborted. You can avoid these read locks by using snapshot isolation. See Using Snapshot Isolation for details.

In the following figure, Txn A and Txn B are both holding read locks on page 002, while Txn C is holding a write lock on page 003:



Lock Lifetime

A locker holds its locks until such a time as it does not need the lock any more. What this means is:

A transaction holds any locks that it obtains until the transaction is committed or aborted.

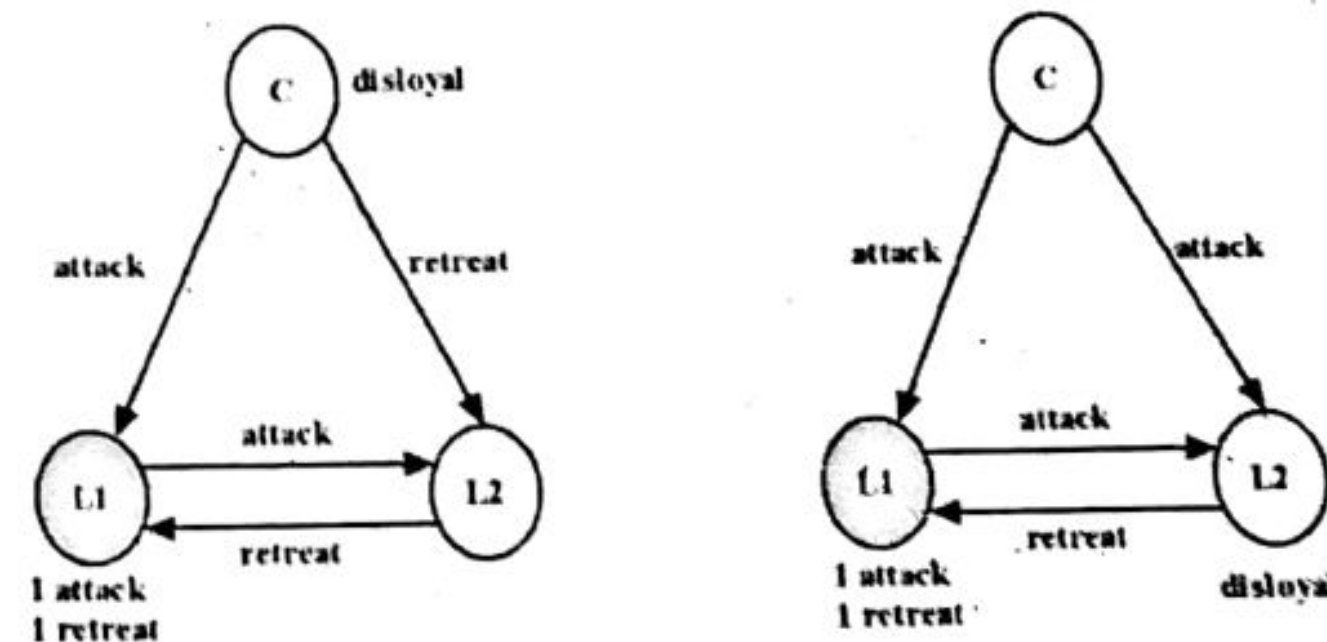
All non-transaction operations hold locks until such a time as the operation is completed. For cursor operations, the lock is held until the cursor is moved to a new position or closed.

Q8) What is agreement protocols? What is agreement and validity objectives of Byzantine agreement problems?

Agreement protocol is used to ensure that DS can achieve the common goal even after occurrence of various failures in Distributed system. • There are some standard agreement problems in DC. We will see each problem and try to find some protocol or algorithm to solve the agreement problem. Three well known problems ① Byzantine agreement problem ② Consensus problem ③ Interactive consistency problem

Second part

The Byzantine Agreement Problem Three generals cannot reach Byzantine agreement.



An arbitrarily chosen processor, called the source processor, broadcasts its initial value to all other processors. A solution to the Byzantine agreement problem should meet the following two objectives: Agreement: All nonfaulty processors agree on the same value. Validity: If the source processor is nonfaulty, then the common agreed upon value by all nonfaulty processors should be initial value of the source.

Two points should be noted :

- ① If the source processor is faulty, then all nonfaulty processors can agree on any common value.
- ② It is irrelevant what value faulty processors agree on or whether they agree on a value at all.

Q9) What are the needs and roles of atomic commitment protocols (ACP) in Distributed transaction? Explain how optimistic concurrency control mechanism works.

Ans:

Atomic Commitment Protocol

- All the DM that reach a decision, reach the same decision.
- Decisions are not reversible.
- A Commit decision can only be reached if all the DMs vote to commit.
- If there are no failures and all the DMs vote to commit, the decision will be Commit.
- At any point, if all failures are repaired, and no new failures are introduced, then all the DMs eventually reach a decision

Types:

Distributed One-phase Commit

Distributed one-phase commit is the simplest commit protocol. Let us consider that there is a controlling site and a number of slave sites where the transaction is being executed. The steps in distributed commit are –

- After each slave has locally completed its transaction, it sends a “DONE” message to the controlling site.
- The slaves wait for “Commit” or “Abort” message from the controlling site. This waiting time is called window of vulnerability.
- When the controlling site receives “DONE” message from each slave, it makes a decision to commit or abort. This is called the commit point. Then, it sends this message to all the slaves.
- On receiving this message, a slave either commits or aborts and then sends an acknowledgement message to the controlling site.

Distributed Two-phase Commit

Distributed two-phase commit reduces the vulnerability of one-phase commit protocols. The steps performed in the two phases are as follows –

Phase 1: Prepare Phase

- After each slave has locally completed its transaction, it sends a “DONE” message to the controlling site. When the controlling site has received “DONE” message from all slaves, it sends a “Prepare” message to the slaves.
- The slaves vote on whether they still want to commit or not. If a slave wants to commit, it sends a “Ready” message.
- A slave that does not want to commit sends a “Not Ready” message. This may happen when the slave has conflicting concurrent transactions or there is a timeout.

Phase 2: Commit/Abort Phase

- After the controlling site has received “Ready” message from all the slaves –
 - The controlling site sends a “Global Commit” message to the slaves.
 - The slaves apply the transaction and send a “Commit ACK” message to the controlling site.
 - When the controlling site receives “Commit ACK” message from all the slaves, it considers the transaction as committed.
- After the controlling site has received the first “Not Ready” message from any slave –
 - The controlling site sends a “Global Abort” message to the slaves.
 - The slaves abort the transaction and send a “Abort ACK” message to the controlling site.
 - When the controlling site receives “Abort ACK” message from all the slaves, it considers the transaction as aborted.

Distributed Three-phase Commit

The steps in distributed three-phase commit are as follows –

Phase 1: Prepare Phase

The steps are same as in distributed two-phase commit.

Phase 2: Prepare to Commit Phase

- The controlling site issues an “Enter Prepared State” broadcast message.
- The slave sites vote “OK” in response.

Phase 3: Commit / Abort Phase

The steps are same as two-phase commit except that "Commit ACK"/"Abort ACK" message is not required.

Q10) Write short notes on:

- MACH
- RMI software
- Process and Threads in DS

a. MACH

- First Generation micro-kernel
- Builds operating system above minimal kernel
- Kernel provides only fundamental services
- These services basic but powerful enough to be used on, wide range of architectures
- Aids distributed computing and multiprocessing

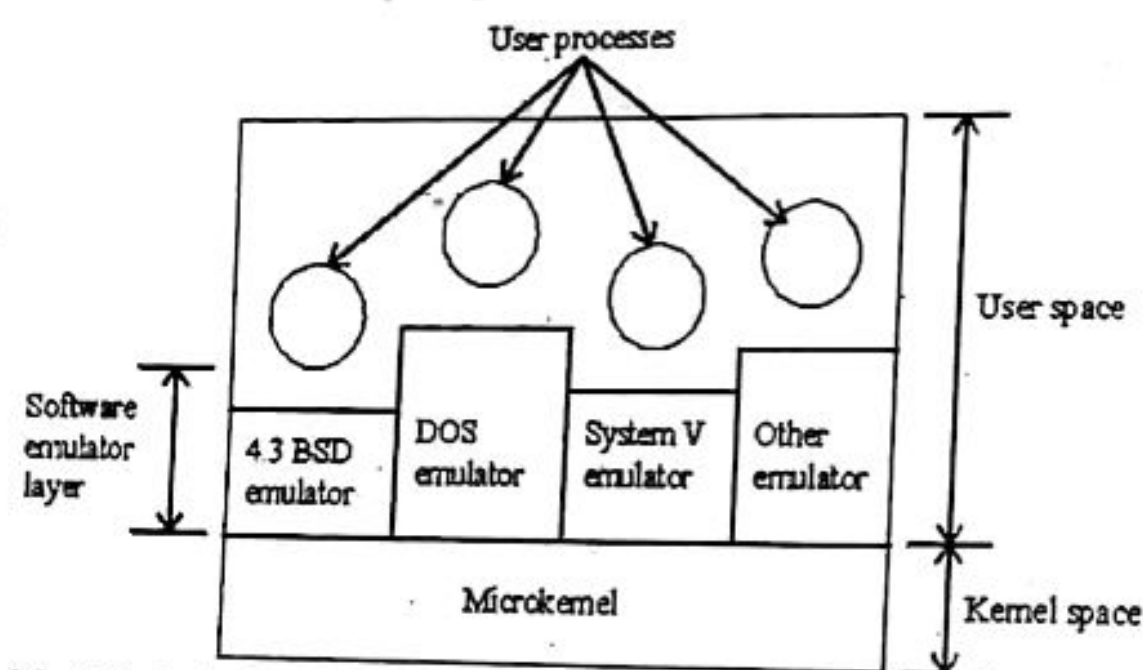
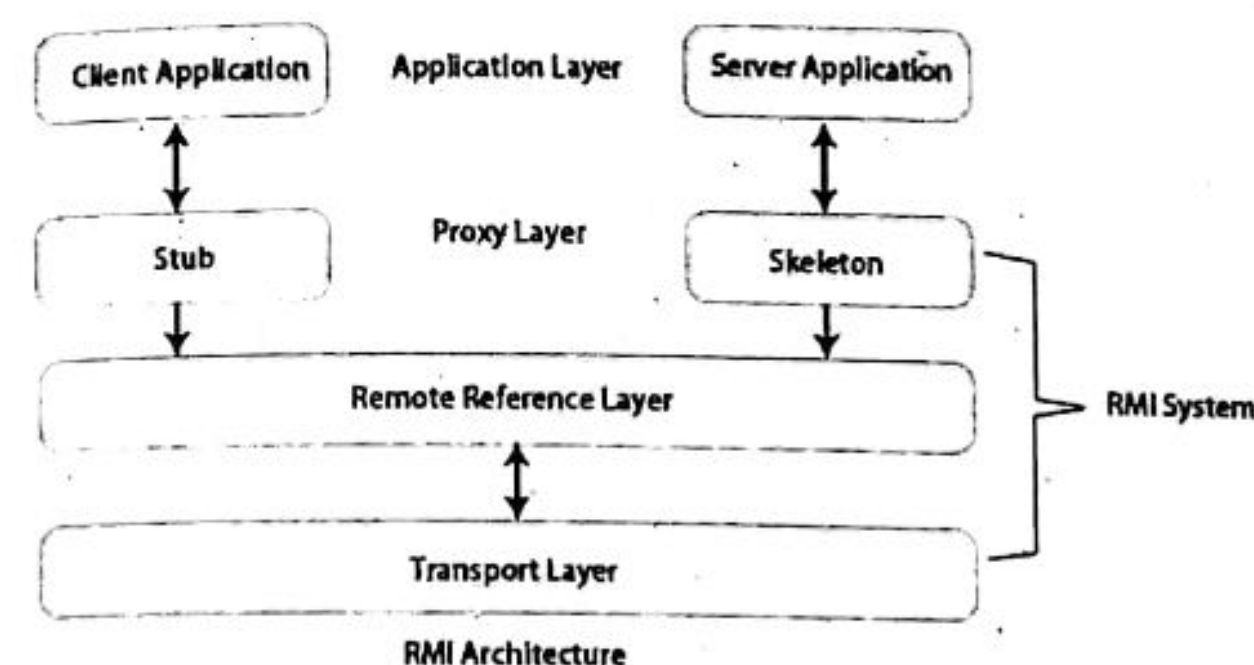


Fig: Mach Architecture

b. RMI software



RMI stands for Remote Method Invocation. It is an API provided by java that allows an object residing in one JVM (Java Virtual Machine) to access or invoke an object running on another JVM. The other JVM could be on the same machine or remote machine.

Let us now discuss the components of this architecture.

- Transport Layer – This layer connects the client and the server. It manages the existing connection and sets up new connections.
- Stub – A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- Skeleton – This is the object which resides on the server side. stub communicates with this skeleton to pass request to the remote object.
- RRL (Remote Reference Layer) – It is the layer which manages the references made by the client to the remote object.

Following are the features of RMI –

- To minimize the complexity of the application.
- To preserve type safety.
- Distributed garbage collection.
- Minimize the difference between working with local and remote objects.

c. Process and Threads

Process:

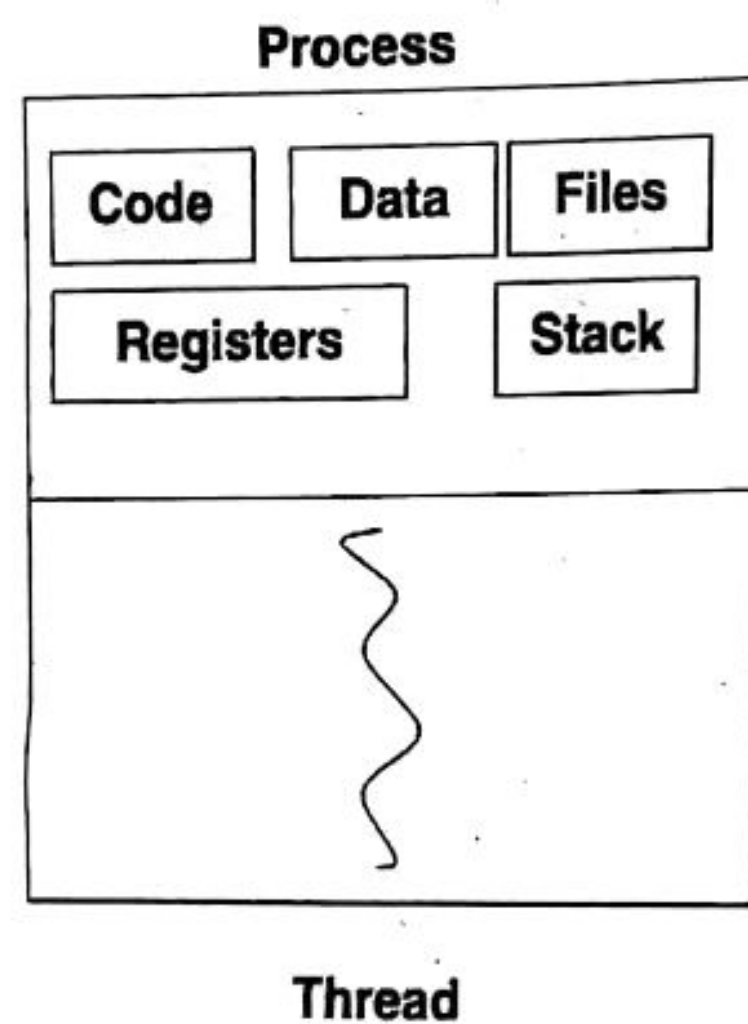
Processes are basically the programs that are dispatched from the ready state and are scheduled in the CPU for execution. PCB(Process Control Block) holds the concept of process. A process can create other processes which are known as Child Processes. The process takes more time to terminate and it is isolated means it does not share the memory with any other process.

The process can have the following states new, ready, running, waiting, terminated, and suspended.

Thread:

Thread is the segment of a process means a process can have multiple threads and these multiple threads are contained within a process. A thread has three states: Running, Ready, and Blocked.

The thread takes less time to terminate as compared to the process but unlike the process, threads do not isolate.



2078 Bhadra

Q 1) What do you mean by Distributed System? Explain various model of Distributed computing systems.

A distributed system is one in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing. A distributed system is a collection of independent computers that appear to the users of the system as a single computer.

Second part,

The various model of Distributed computing systems are:

Minicomputer model

Workstation model

Workstation – server model

Processor – pool model

Hybrid model

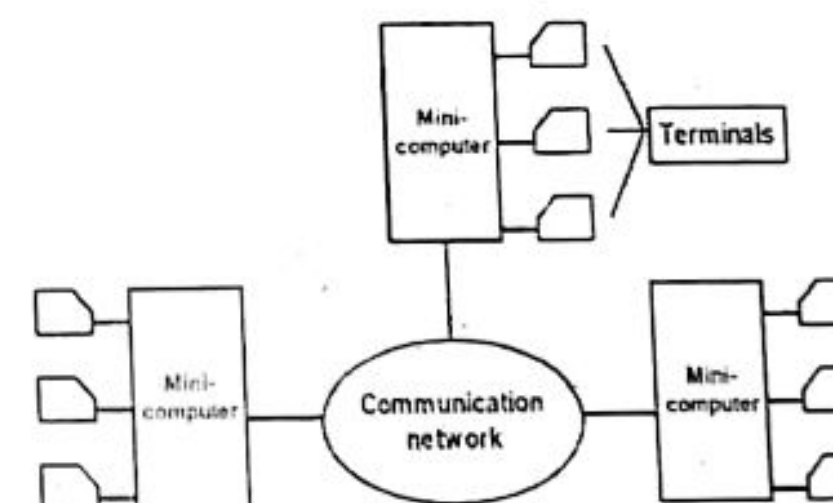
Minicomputer Model

The minicomputer model is a simple extension of the centralized time-sharing system.

A distributed computing system based on this model consists of a few minicomputers (they may be large supercomputers as well) interconnected by a communication network.

Each minicomputer usually has multiple users simultaneously logged on to it. For this, several interactive terminals are connected to each minicomputer. Each user is logged on to one specific minicomputer, with remote access to other minicomputers..

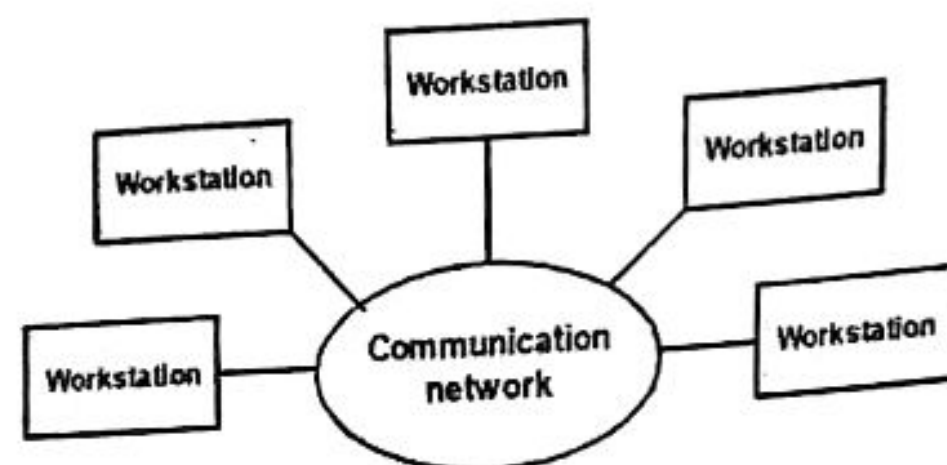
The early ARPAnet is an example of a distributed computing system based on the minicomputer model.



Workstation Model

A distributed computing system based on the workstation model consists of several workstations interconnected by a communication network.

An organization may have several workstations located throughout a building or campus, each workstation equipped with its own disk and serving as a single-user computer.



Workstation – Server Model

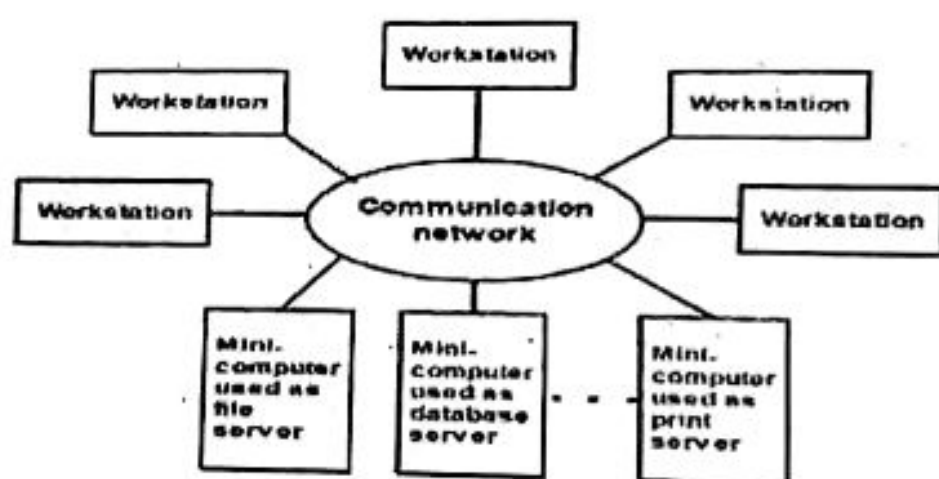
The workstation model is a network of personal workstations, each with its own disk and a local file system.

A workstation with its own local disk is usually called a diskful workstation and a workstation without a local disk is called a diskless workstation.

With the proliferation of high-speed networks, diskless workstations have become more popular in network environments than diskful workstations, making the workstation-server model more popular than the workstation model for building distributed computing systems.

A distributed computing system based on the workstation-server model consists of a few minicomputers and several workstations (most of which are diskless, but a few of which may be diskful) interconnected by a communication network.

Processor – pool model



Hybrid Model

Out of the four models described above, the workstation-server model, is the most widely used model for building distributed computing systems. This is because a large number of computer users only perform simple interactive tasks such as editing jobs, sending electronic mails, and executing small programs. The workstation-server model is ideal for such simple usage. However, in a working environment that has groups of users who often perform jobs needing massive computation, the processor-pool model is more attractive and suitable.

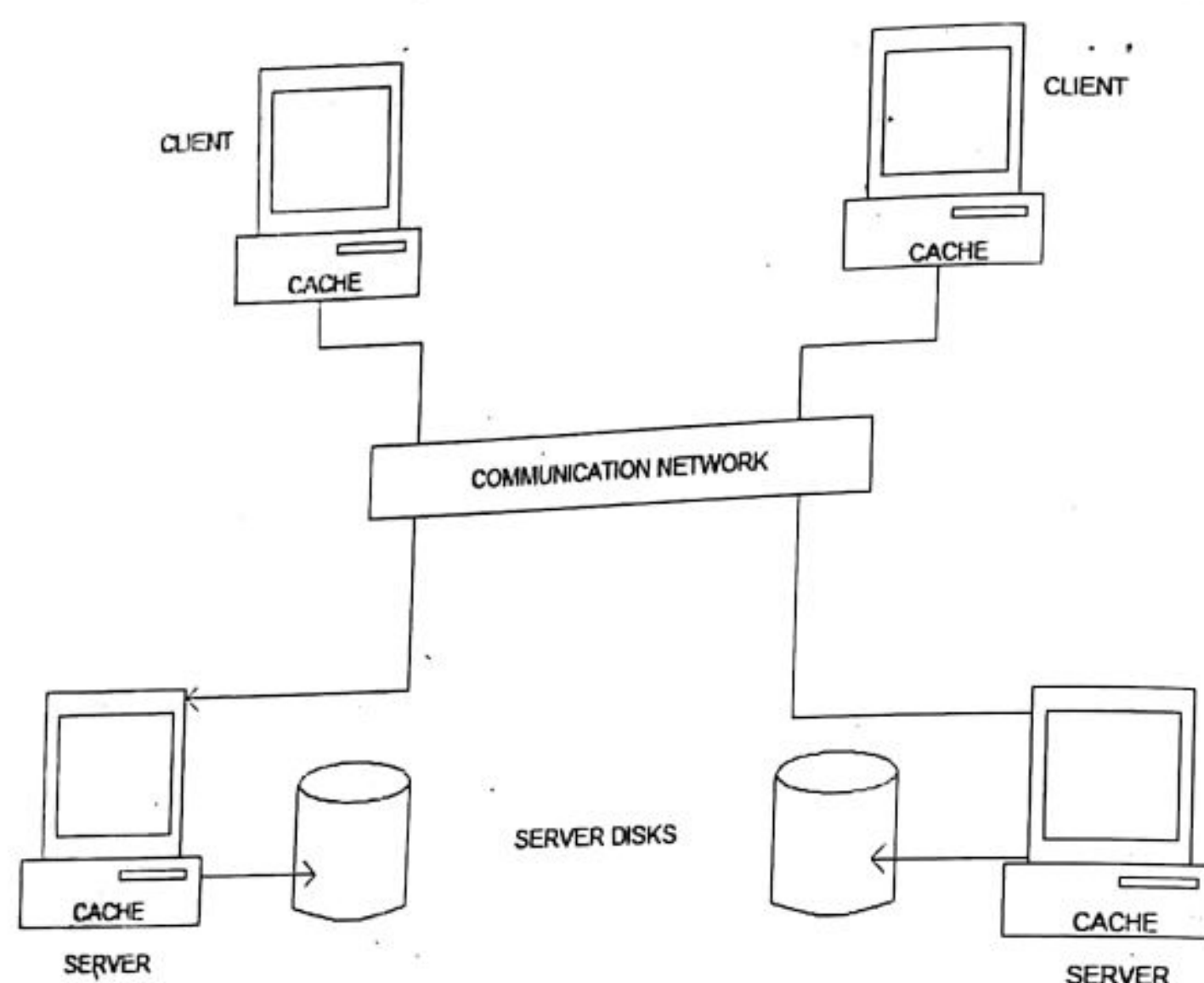
Q 2) Elaborate difference between RPC and RMI . Describe the File Service Architecture of DFS.

S.NO	RPC	RMI
1.	RPC is a library and OS dependent platform.	Whereas it is a java platform.
2.	RPC supports procedural programming.	RMI supports object oriented programming.
3.	RPC is less efficient in comparison of RMI.	While RMI is more efficient than RPC.
4.	RPC creates more overhead.	While it creates less overhead than RPC.
5.	The parameters which are passed in RPC are ordinary or normal data.	While in RMI, objects are passed as parameter.
6.	RPC is the older version of RMI.	While it is the successor version of RPC.
7.	There is high Provision of ease of programming in RPC.	While there is low Provision of ease of programming in RMI.
8.	RPC does not provide any security.	While it provides client level security.
9.	It's development cost is huge.	While it's development cost is fair or reasonable.
10.	There is a huge problem of versioning in RPC.	While there is possible versioning using RMI.
11.	There is multiple codes are needed for simple application in RPC.	While there is multiple codes are not needed for simple application in RMI.

Second part,

A Distributed File System (DFS) as the name suggests, is a **file system that is distributed on multiple file servers or multiple locations**. It allows programs to access or store isolated files as they do with the local ones, allowing programmers to access files from any network or computer.

Architecture of a Distributed File System



Flat File Service:

Concerned with implementing operations on the concepts of files.

Unique File Identifiers (UFIDs) are used to refer to files in all requests for flat file service operations.

Responsibilities of file and directory service is based upon UFID (long sequence of bits so each file has UFID which is unique in DS).

Directory Service:

It provides a mapping between text names for files and their UFIDs

Client Obtain UFID by quoting text name to the directory service.

Client Module:

Run on each client computer

Q 3) a what are the key differences between Network OS and Distributed OS.

S.NO	Network Operating System	Distributed Operating System
1.	Network Operating System's main objective is to provide the local services to remote client.	Distributed Operating System's main objective is to manage the hardware resources.
2.	In Network Operating System, Communication takes place on the basis	In Distributed Operating System, Communication takes place on the basis of

	of files.	messages and shared memory.
3.	Network Operating System is more scalable than Distributed Operating System.	Distributed Operating System is less scalable than Network Operating System.
4.	In Network Operating System, fault tolerance is less.	While in Distributed Operating System, fault tolerance is high.
5.	Rate of autonomy in Network Operating System is high.	While The rate of autonomy in Distributed Operating System is less.
6.	Ease of implementation in Network Operating System is also high.	While in Distributed Operating System Ease of implementation is less.
7.	In Network Operating System, All nodes can have different operating system.	While in Distributed Operating System, All nodes have same operating system.

b) Explain the various role of middleware in DS.

A distributed software support layer which abstracts over the complexity and heterogeneity of the underlying distributed environment with its multitude of network technologies, operating systems, and implementation languages. The primary role of middleware is to ease the task of developing, deploying, and managing distributed applications by providing a simple, consistent, and integrated distributed programming environment.

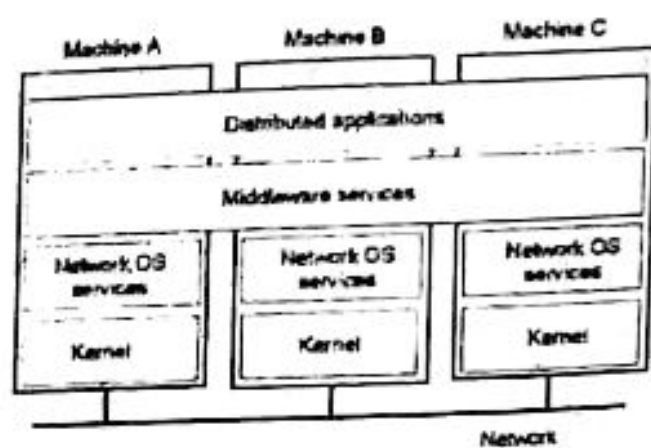
A key component of a heterogeneous distributed client-server environment is middleware.

•Middleware is a set of services that enable applications and end users to interact with each other across a heterogeneous distributed system.

Middleware software resides above the network and below the application software.

Middleware should make the network transparent to the applications and end users \Rightarrow users and applications should be able to perform the same operations across the network that they can perform locally. Middleware should hide the details of computing hardware, OS, software components across networks.

• Different kind of software qualifies, to certain extent, as middleware: - File-transfer packages (FTP) and email; - Web browsers; - CORBA.



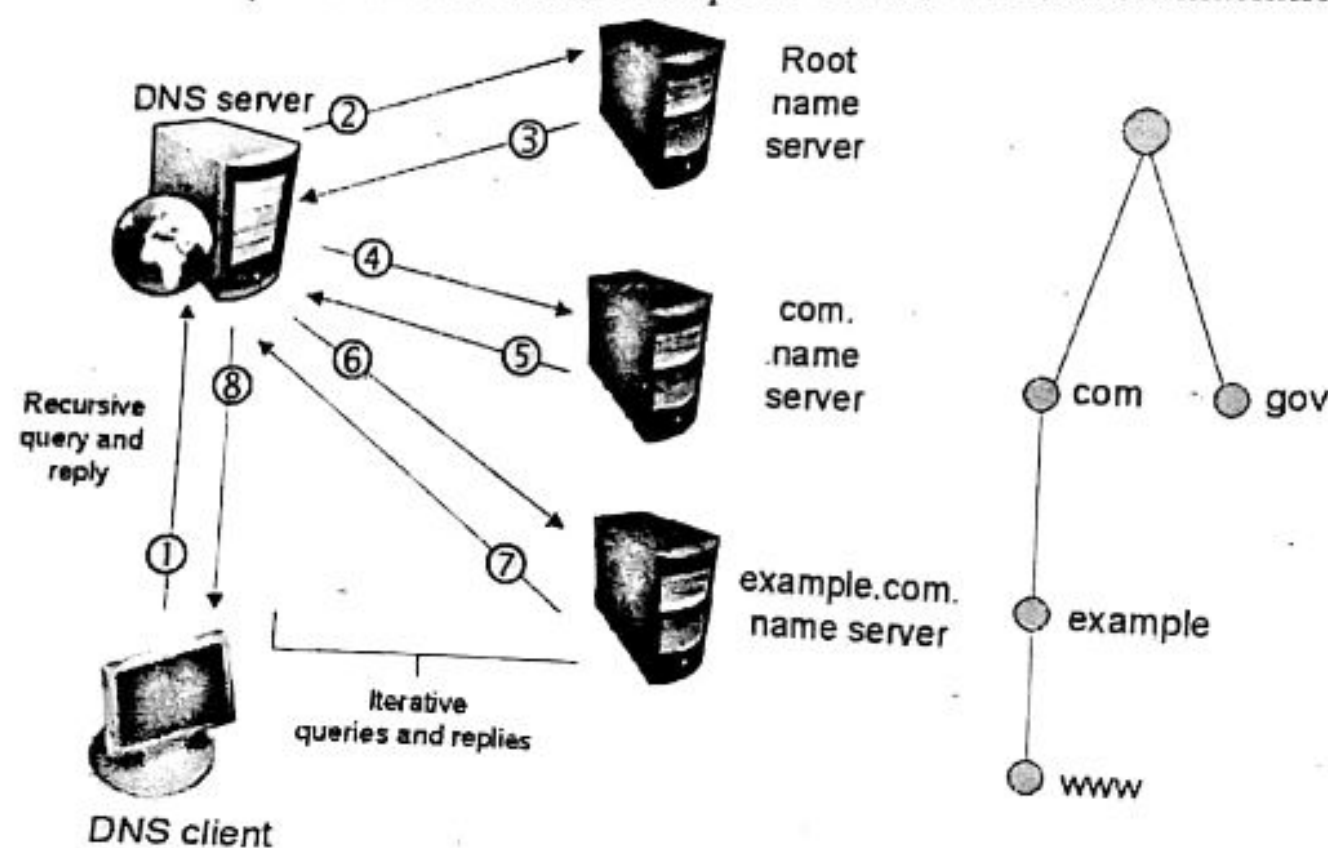
Q 4) What are the design issues of NFS? Show the complete working of DNS.

The NFS was designed for simplicity, but had several limitations, mainly with regards to its performance, consistency, and semantic difficulties. Performance. The NFS has slow write performance and this is given a more thorough treatment in the next section.

Consistency. Servers in NFS do not keep track of which clients are using which files. Because of this, it is the responsibility of the clients to check if their cached files have been modified. Clients do this by polling the server every time they would like to use their cached data by issuing a GETATTR request. However, this request itself is cached by the client for a few seconds to avoid overloading the server with GETATTR requests from multiple clients. This works since the common case is that a client is usually the only one using a file¹. As a result, the system is eventually consistent and there are windows of inconsistency where stale data might be used by clients.

Semantic Difficulties. There are certain features that are difficult to implement and still maintain an NFS server's stateless and idempotent properties. Furthermore, there are also instances in which there is a semantic conflict, since not all NFS operations are idempotent, such as the case of the MKDIR operation.

The Domain Name System (DNS) is the phonebook of the Internet. Humans access information online through domain names, like nytimes.com or espn.com. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses so browsers can load Internet resources. Each device connected to the Internet has a unique IP address which other machines use to find the device.



The 8 steps in a DNS lookup:

1. A user types 'example.com' into a web browser and the query travels into the Internet and is received by a DNS recursive resolver.
2. The resolver then queries a DNS root nameserver (.).
3. The root server then responds to the resolver with the address of a Top-Level Domain (TLD) DNS server (such as .com or .net), which stores the information for its domains. When searching for example.com, our request is pointed toward the .com TLD.
4. The resolver then makes a request to the .com TLD.
5. The TLD server then responds with the IP address of the domain's nameserver, example.com.
6. Lastly, the recursive resolver sends a query to the domain's nameserver.
7. The IP address for example.com is then returned to the resolver from the nameserver.
8. The DNS resolver then responds to the web browser with the IP address of the domain requested initially. Once the 8 steps of the DNS lookup have returned the IP address for example.com, the browser is able to make the request for the web page:
9. The browser makes a HTTP request to the IP address.
10. The server at that IP returns the webpage to be rendered in the browser (step 10).

Q 5) Write the implementation rule of Lamport clock. State the limitations of Lamport logical clock.

Lamport's Logical Clock was created by Leslie Lamport. It is a procedure to determine the order of events occurring. It provides a basis for the more advanced Vector Clock Algorithm. Due to the absence of a Global Clock in a Distributed Operating System Lamport Logical Clock is needed.

Algorithm:

Happened before relation (\rightarrow): $a \rightarrow b$, means 'a' happened before 'b'.

Logical Clock: The criteria for the logical clocks are:

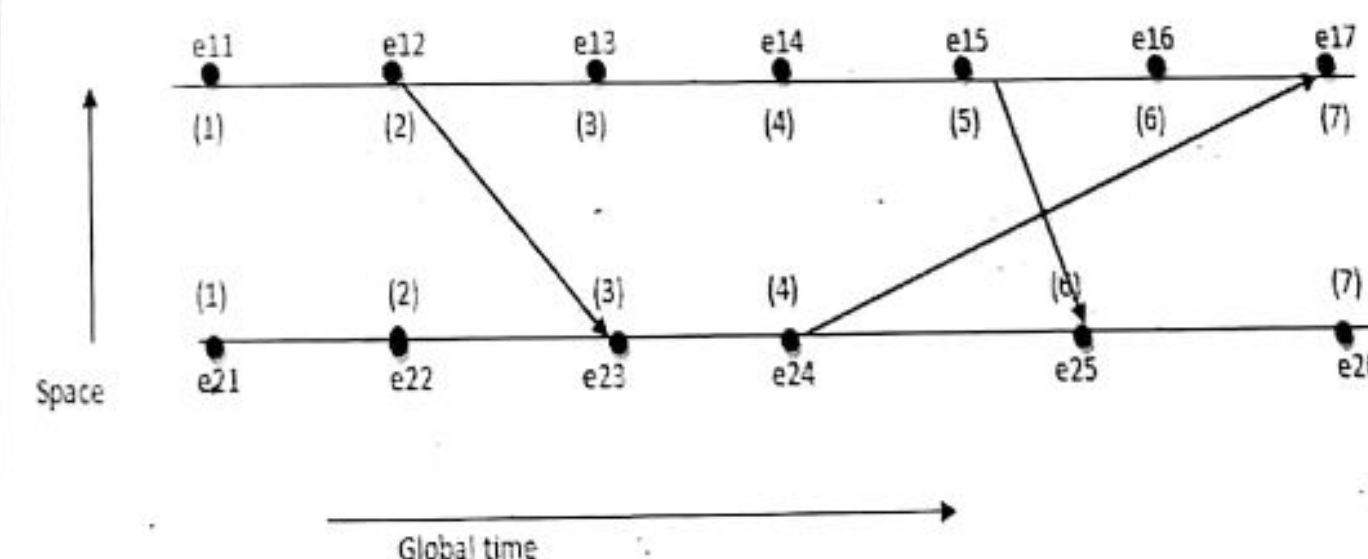
[C1]: $C_i(a) < C_i(b)$, [$C_i \rightarrow$ Logical Clock, If 'a' happened before 'b', then time of 'a' will be less than 'b' in a particular process.]

[C2]: $C_i(a) < C_j(b)$, [Clock value of $C_i(a)$ is less than $C_j(b)$]

Implementation Rules [IR]:

[IR1]: If $a \rightarrow b$ ['a' happened before 'b' within the same process] then, $C_i(b) = C_i(a) + d$

[IR2]: $C_j = \max(C_j, t_m + d)$ [If there's more number of processes, then t_m = value of $C_i(a)$, C_j = max value between C_i and $t_m + d$]



Take the starting value as 1, since it is the 1st event and there is no incoming value at the starting point:

$e11 = 1$

$e21 = 1$

The value of the next point will go on increasing by d ($d = 1$), if there is no incoming value i.e., to follow [IR1].

$e12 = e11 + d = 1 + 1 = 2$

$$\begin{aligned}
 e13 &= e12 + d = 2 + 1 = 3 \\
 e14 &= e13 + d = 3 + 1 = 4 \\
 e15 &= e14 + d = 4 + 1 = 5 \\
 e16 &= e15 + d = 5 + 1 = 6 \\
 e22 &= e21 + d = 1 + 1 = 2 \\
 e24 &= e23 + d = 3 + 1 = 4 \\
 e26 &= e25 + d = 6 + 1 = 7
 \end{aligned}$$

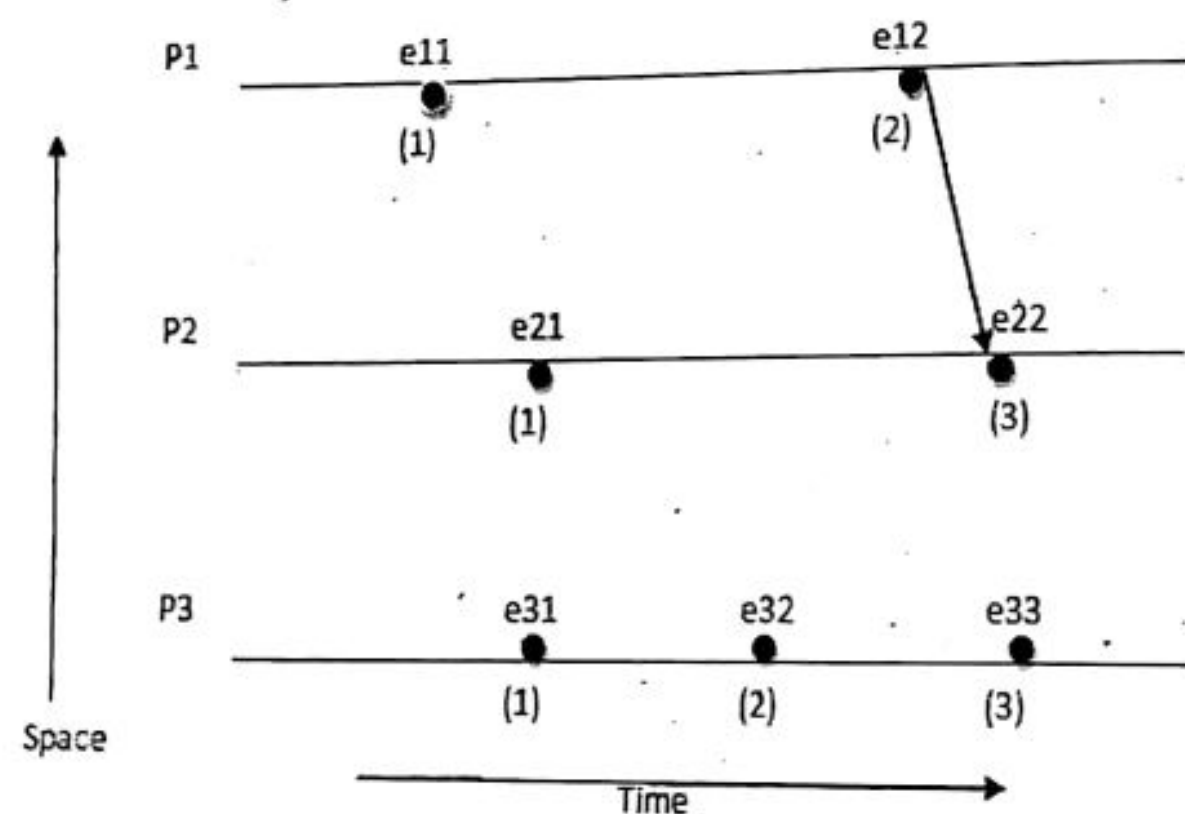
When there will be incoming value, then follow [IR2] i.e., take the maximum value

between C_j and $T_m + d$.
 $e17 = \max(7, 5) = 7$, [$e16 + d = 6 + 1 = 7$, $e24 + d = 4 + 1 = 5$, maximum among 7 and 5 is 7]
 $e23 = \max(3, 3) = 3$, [$e22 + d = 2 + 1 = 3$, $e12 + d = 2 + 1 = 3$, maximum among 3 and 3 is 3]
 $e25 = \max(5, 6) = 6$, [$e24 + 1 = 4 + 1 = 5$, $e15 + d = 5 + 1 = 6$, maximum among 5 and 6 is 6]

Limitation:

In case of [IR1], if $a \rightarrow b$, then $C(a) < C(b) \rightarrow \text{true}$.

In case of [IR2], if $a \rightarrow b$, then $C(a) < C(b) \rightarrow \text{May be true or may not be true.}$



Q 6) Compare and contrast token mutual exclusion algorithm and non-token based mutual exclusion algorithm. Explain the various state of Ricart Agrawala token based mutual exclusion.

	Token based algorithm	Non token based algorithm
1.	In the Token-based algorithm, a unique token is shared among all the sites in Distributed Computing Systems.	In Non-Token based algorithm, there is no token even not any concept of sharing token for access.
2.	Here, a site is allowed to enter the Computer System if it possesses the token.	Here, two or more successive rounds of messages are exchanged between sites to determine which

		site is to enter the Computer System next.
3.	The token-based algorithm uses the sequences to order the request for the Computer Systems and to resolve the conflict for the simultaneous requests for the System.	Non-Token based algorithm uses the timestamp (another concept) to order the request for the Computer Systems and to resolve the conflict for the simultaneous requests for the System.
4.	The token-based algorithm produces less message traffic as compared to Non-Token based Algorithm.	Non-Token based Algorithm produces more message traffic as compared to the Token-based Algorithm.
5.	They are free from deadlock (i.e. here there are no two or more processes are in the queue in order to wait for messages that will actually can't come) because of the existence of unique token in the distributed system.	They are not free from the deadlock problem as they are based on timestamp only.
6.	Here, it was ensured that requests are executed exactly in the order as they are made in.	Here there is no surety of execution order.
7.	Token-based algorithms are more scalable as they can free your server from having to store session state and also they contain all the necessary information which they need for authentication.	Non-Token based algorithms are less scalable than the Token-based algorithms because here server is not free from its tasks.
8.	Here the access control is quite Fine-grained because here inside the token roles, permissions and resources can be easily specifying for the user.	Here the access control is not so fine as there is no token which can specify roles, permission, and resources for the user.
9.	Token-based algorithms make authentication quite easy.	Non-Token based algorithms can't make authentication easy.

10.	<p>Examples of Token-Based Algorithms are:</p> <p>(i) Singhal's Heuristic Algorithm</p> <p>(ii) Raymonds Tree Based Algorithm</p> <p>(iii) Suzuki-Kasami Algorithm</p>	<p>Examples of Non-Token Based Algorithms are:</p> <p>(i) Lamport's Algorithm</p> <p>(ii) Ricart-Agarwala Algorithm</p> <p>(iii) Maekawa's Algorithm</p>
-----	--	--

Second part

Ricart-Agrawala algorithm is an algorithm to for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala. This algorithm is an extension and optimization of Lamport's Distributed Mutual Exclusion Algorithm. Like Lamport's Algorithm, it also follows permission based approach to ensure mutual exclusion.

In this algorithm:

Two type of messages (**REQUEST** and **REPLY**) are used and communication channels are assumed to follow FIFO order.

A site send a **REQUEST** message to all other site to get their permission to enter critical section.

A site send a **REPLY** message to other site to give its permission to enter the critical section.

A timestamp is given to each critical section request using Lamport's logical clock.

Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section request is always in the order of their timestamp.

Algorithm:

To enter Critical section:

When a site S_i wants to enter the critical section, it send a timestamped **REQUEST** message to all other sites.

When a site S_j receives a **REQUEST** message from site S_i , It sends a **REPLY** message to site S_i if and only if

Site S_j is neither requesting nor currently executing the critical section.

In case Site S_j is requesting, the timestamp of Site S_i 's request is smaller than its own request.

Otherwise the request is deferred by site S_j .

To execute the critical section:

Site S_i enters the critical section if it has received the **REPLY** message from all other sites.

To release the critical section:

Upon exiting site S_i sends **REPLY** message to all the deferred requests.

Message Complexity:

Ricart-Agrawala algorithm requires invocation of $2(N-1)$ messages per critical section execution. These

$2(N-1)$ messages involves

$(N-1)$ request messages

$(N-1)$ reply messages

Drawbacks of Ricart-Agrawala algorithm:

Unreliable approach: failure of any one of node in the system can halt the progress of the system. In this situation, the process will starve forever.

The problem of failure of node can be solved by detecting failure after some timeout.

Performance:

Synchronization delay is equal to maximum message transmission time

It requires $2(N-1)$ messages per Critical section execution

Q 7) Mention the requirement and challenges of replication. Explain the active replication model in fault tolerance .How it differ than passive model.

Data Replication is the process of storing data in more than one site or node. It is useful in **improving the availability of data**. It is simply copying data from a database from one server to another server so that all the users can share the same data without any inconsistency. The result is a **distributed database** in which users can access data relevant to their tasks without interfering with the work of others.

Requirement of replication – Data Replication is generally performed to:

To provide a consistent copy of data across all the database nodes.

To increase the availability of data.

The reliability of data is increased through data replication.

Data Replication supports multiple users and gives high performance.

To remove any data redundancy, the databases are merged and slave databases are updated with outdated or incomplete data.

Since replicas are created there are chances that the data is found itself where the transaction is executing which reduces the data movement.

To perform faster execution of queries.

Challenges of replication:

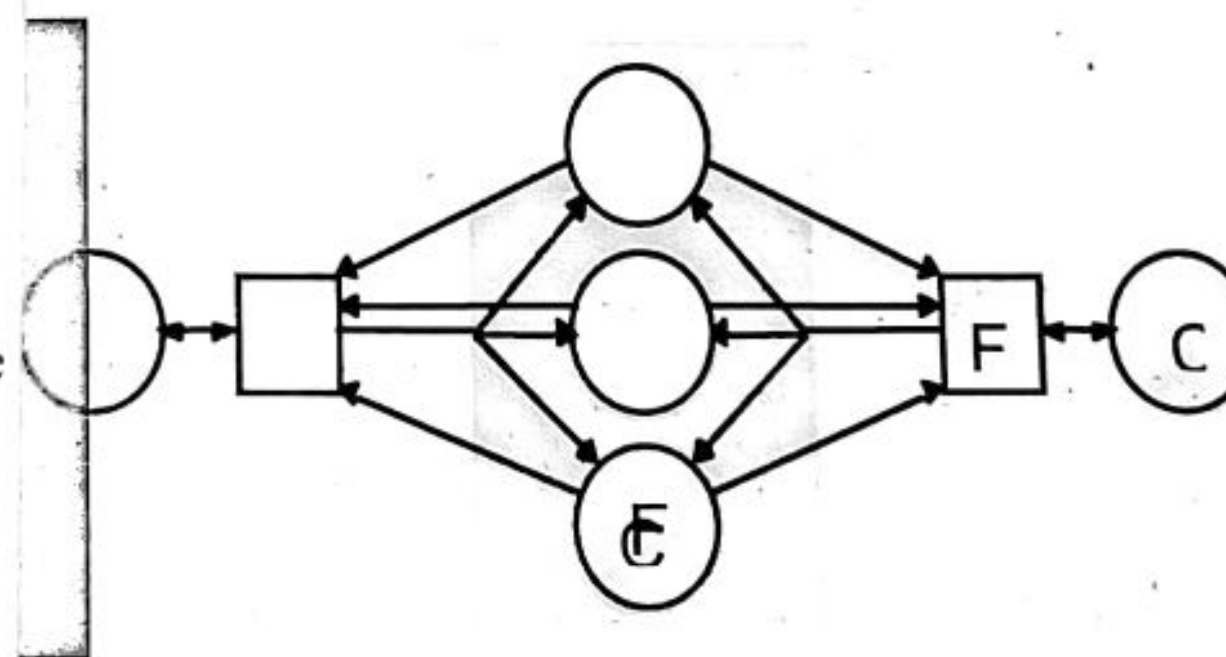
More storage space is needed as storing the replicas of same data at different sites consumes more space.

Data Replication becomes expensive when the replicas at all different sites need to be updated.

Maintaining Data consistency at all different sites involves complex measures.

Second part,

Where update volume or other factors such as partitioned networks demand updates to more than one replica at the same time, passive replication may be unsuitable. So , active replication is used.



In **active replication** each client request is processed by all the servers. This requires that the process hosted by the servers is **deterministic**. **Deterministic** means that, given the same initial state and a request sequence, all processes will produce the same response sequence and end up in the same final state. In

order to make all the servers receive the same sequence of operations, an atomic broadcast protocol must be used. An atomic broadcast protocol guarantees that either all the servers receive a message or none, plus that they all receive messages in the same order. The big disadvantage for active replication is that in practice most of the real world servers are non-deterministic. Still active replication is the preferable choice when dealing with real time systems that require quick response even under the presence of faults or with systems that must handle byzantine faults.

In **passive replication** there is only one server (called primary) that processes client requests. After processing a request, the primary server updates the state on the other (backup) servers and sends back the response to the client. If the primary server fails, one of the backup servers takes its place. Passive replication may be used even for non-deterministic processes. The disadvantage of passive replication compared to active is that in case of failure the response is delayed.

8) Discuss how consensus can be achieved in Distributed System. Explain the check pointing approach for distributed recovery.

Consensus means agreeing on things (leader, sequence numbers, time for action, action to be taken etc).

Basic Consensus

- Set of processes
- Each starts with state = undecided
- Each has a single value
- Have to set their decision variable to the same value and enter decided state
- Termination: each process sets its decision variable and enters decided state
- Agreement: If 2 processes have entered decided state, then their decision variables are equal
- Integrity: If all working processes proposed the same value v , then all of them in decided state has decision= v

Second part,

In the distributed computing environment, checkpointing is a technique that helps tolerate failures that otherwise would force long-running application to restart from the beginning. Checkpointing is an important feature in distributed computing systems. It gives fault tolerance without requiring additional efforts from the programmer. A checkpoint is a snapshot of the current state of a process.

There are two main approaches for checkpointing in the distributed computing systems: **coordinated checkpointing and uncoordinated checkpointing**. In the coordinated checkpointing approach, processes must ensure that their checkpoints are consistent.

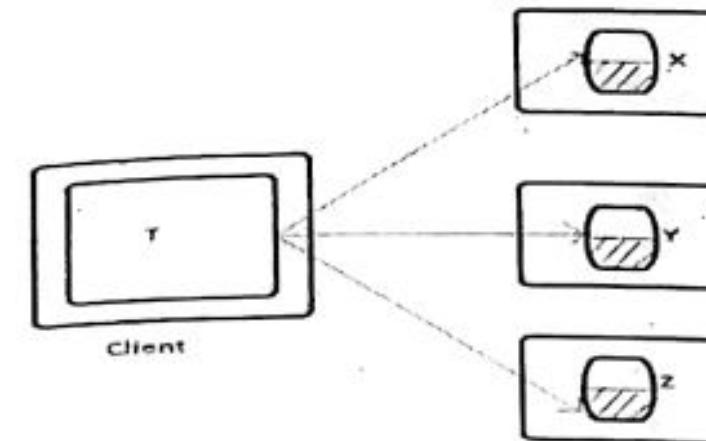
In uncoordinated checkpoint, each process takes its checkpoints checkpoints independently.

Q 9) What are flat and nested transactions? Explain how the problems of 2PC protocols are solved by 3PC.

A flat transaction has a single initiating point (Begin) and a single end point (Commit or abort). They are usually very simple and are generally used for short activities rather than larger ones.

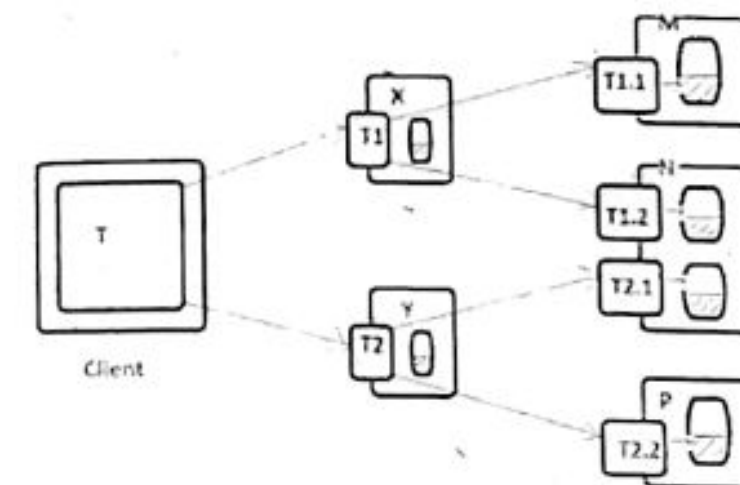
A client makes requests to multiple servers in a flat transaction. Transaction T, for example, is a flat transaction that performs operations on objects in servers X, Y, and Z.

Before moving on to the next request, a flat client transaction completes the previous one. As a result, each transaction visits the server object in order. A transaction can only wait for one object at a time when servers utilize locking.



Nested transaction:

A transaction that includes other transactions within its initiating point and an end point are known as nested transactions. So the nesting of the transactions is done in a transaction. The nested transactions here are called sub-transactions.



Second part,

Disadvantages of 2PC:

a) The major disadvantage of the Two-phase commit protocol is faced when the Coordinator site failure may result in blocking, so a decision either to commit or abort **Transaction(T)** may have to be postponed until coordinator recovers.

b) **Blocking Problem:** Consider a scenario, if a **Transaction(T)** holds locks on data-items of active sites, but amid the execution, if the coordinator fails and the active sites keep no additional log-record except **<read T>** like **<abort>** or **<commit>**. So, it becomes impossible to determine what decision has been made (whether to **<commit>** / **<abort>**). So, In that case, the final decision is delayed until the **Coordinator** is restored or fixed. In some cases, this may take a day or long hours to restore and during this time period, the locked data items remain inaccessible for other **transactions(T_i)**. This problem is known as **Blocking Problem**.

Three-Phase Commit (3PC) Protocol is an extension of the Two-Phase Commit (2PC) Protocol that avoids blocking problem under certain assumptions. In particular, it is assumed that no network partition occurs, and not more than k sites fail, where we assume ' k ' is predetermined number. With the mentioned assumptions, protocol avoids blocking by introducing an extra third phase where multiple sites are involved in the decision to commit.

Instead of directly noting the commit decision in its persistent storage, the **coordinator** first ensures that at least ' k ' other sites know that it intended to commit transaction.

In a situation where coordinator fails, remaining sites are bound to first select new coordinator. This new coordinator checks status of the protocol from the remaining sites. If the coordinator had decided to

commit, at least one of other 'k' sites that it informed will be up and will ensure that commit decision is respected. The new coordinator restarts third phase of protocol if any of rest sites knew that old coordinator intended to commit transaction. Otherwise, new coordinator aborts the transaction.

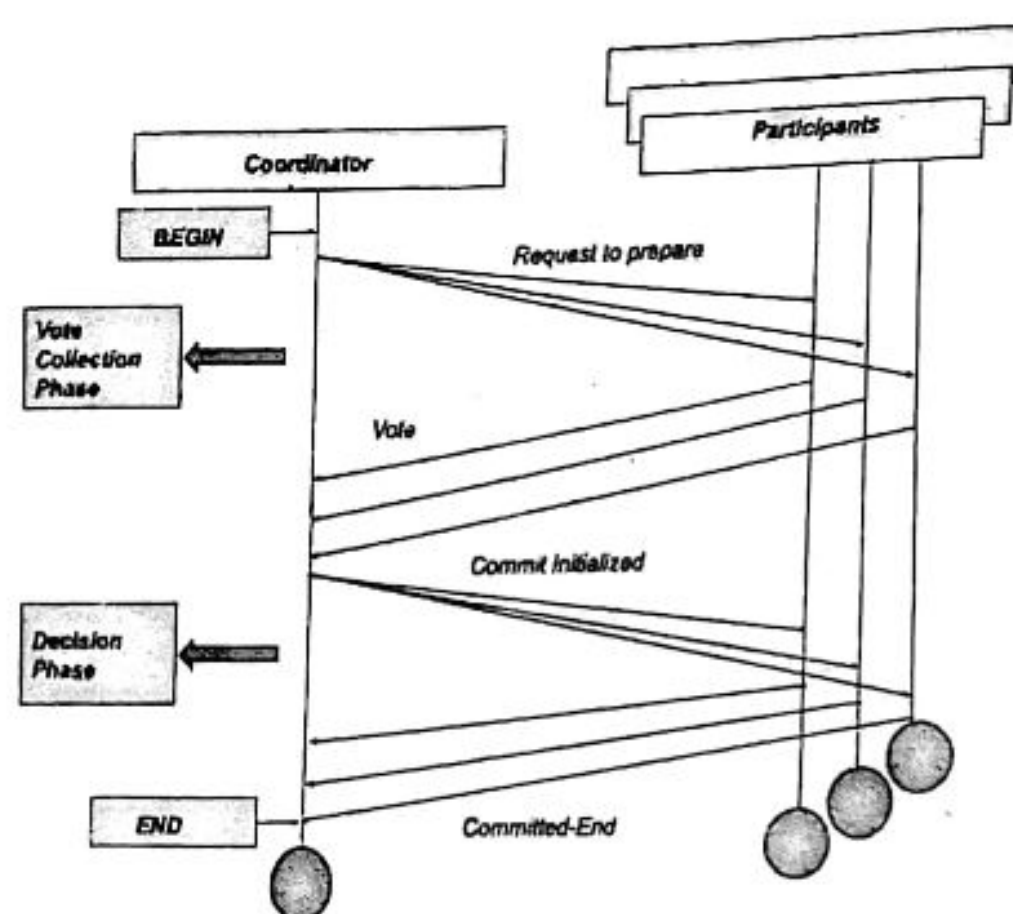


Fig: 2 phase commit

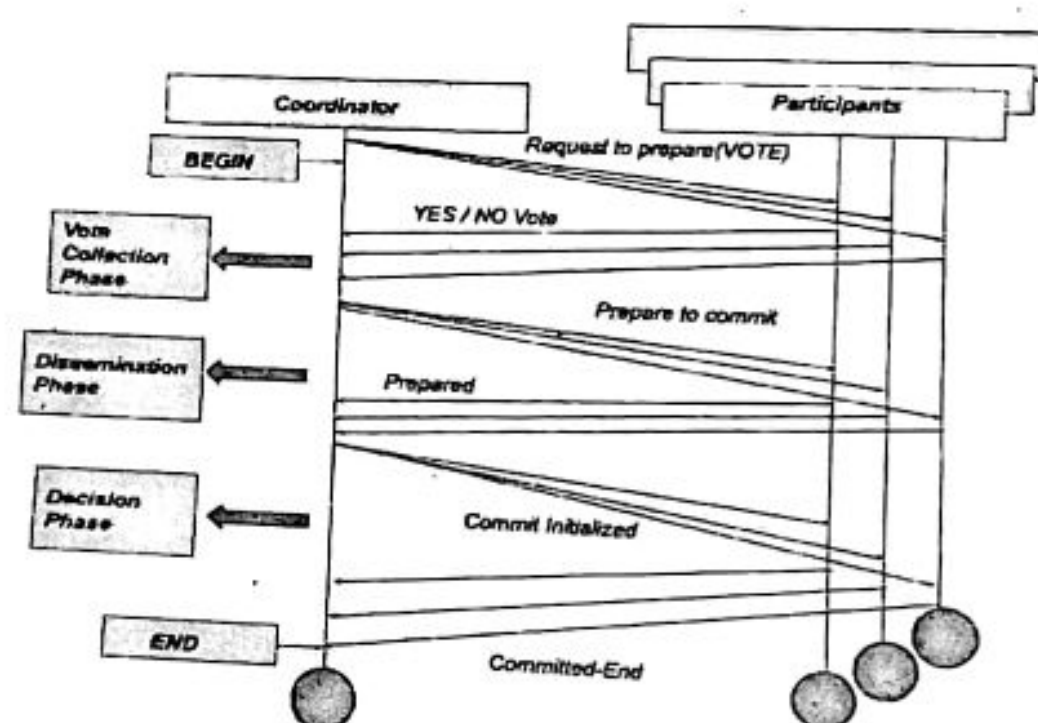


Fig: 3 phase commit

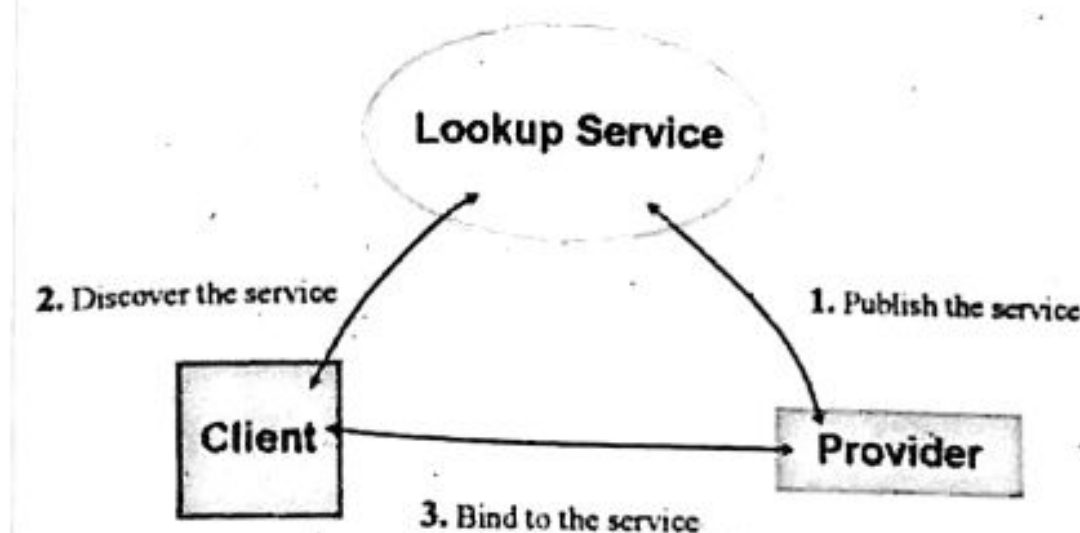
10) Write short notes on:

a) JINI

Jini is actually a Java program that serves as a translator for communication between a computer and other devices on the network. It enables all types of devices to simply connect into impromptu networks, making access to and delivery of new network services as simple as plugging. It enables all types of devices to work together in a community put together without extensive planning, installation, or human intervention.

Jini is a distributed computing network environment that offers, "Network plug and play". Jini is connection technology is based on a simple concept that "device should work together." no driver to find, no operating system issues, no wired cables and connectors.

JINI architecture:



JINI PROCESS:

Discover: find a Lookup service.

Join: send a copy of the service proxy to the Lookup service.

Discover: find a Lookup service.

Lookup: request a service.

Receive a copy of the service proxy.

Advantages

Jini is open source, meaning that the program code is freely available on the Internet and there are no fees for using it.

Jini supports an extremely flexible network as Services and Clients can move code to where it is needed. New services are easy to program in Java and can be easily added to the network.

b) Distributed debugging:

Distributed debugging is the ability to debug associated ABL processes within a distributed application environment. That is, one debugger session can control both the client process and AppServer process, giving the client session the ability to step into code running on the AppServer.

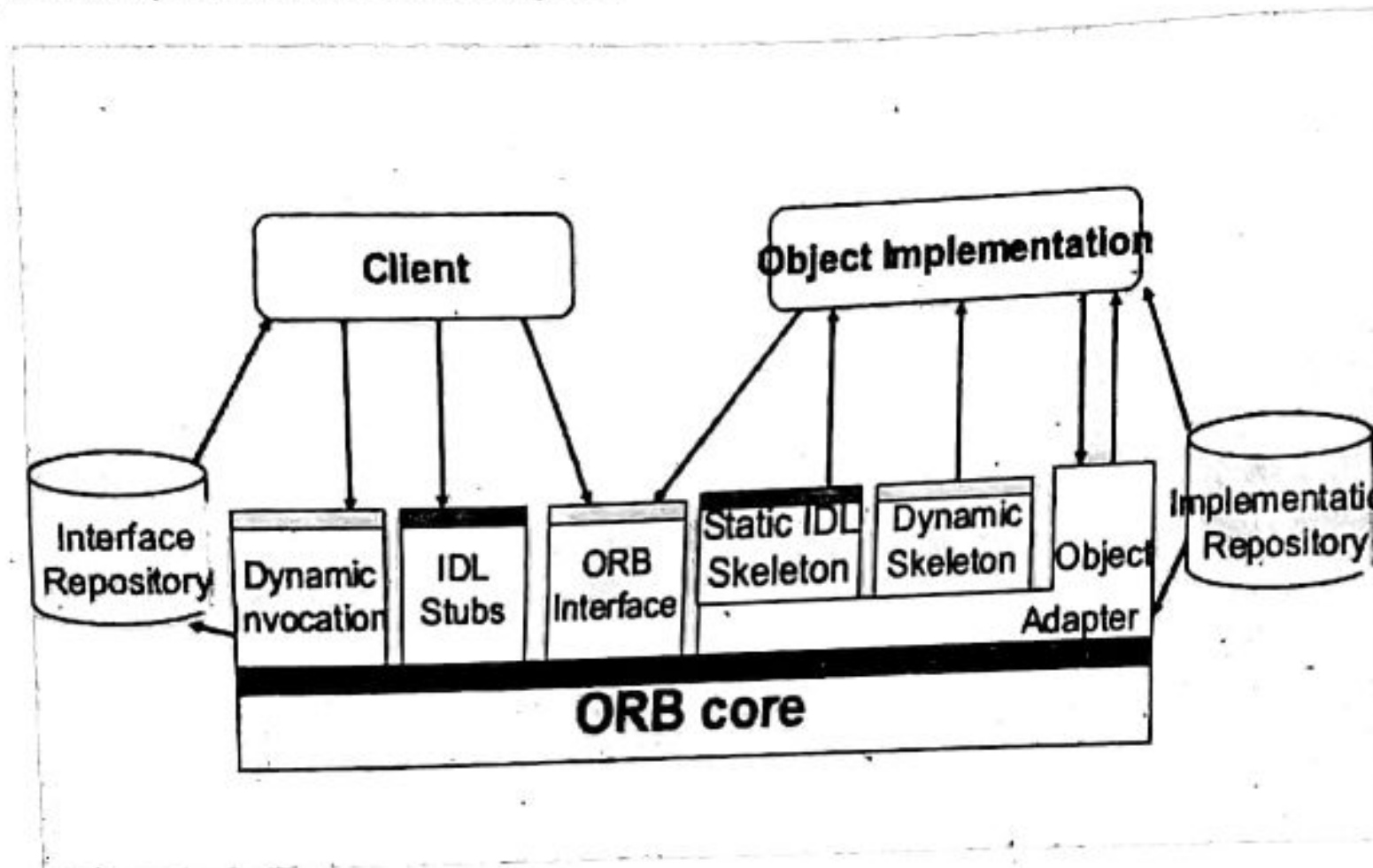
Distributed debugging example:

- Dropping large chunks of messages occasionally
- Using default OS stack parameters
- Packet trace shows many bursts or ARP messages ! ARP table being flushed ! Occasionally get garbled data
- Occurs when changing packet size and rate ! QNX IP stack building malformed packets ! Can't scale the system past 50 nodes - New nodes added dynamically - Nodes share their config data with each other ! But in this case, nodes resent all config data to everyone.

c) CORBA architecture:

CORBA stands for Common Object Request Broker Architecture. It is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

CORBA provides a platform-independent, language-independent way to write applications that can invoke objects that live across the room or across the planet. More specifically, CORBA is a mechanism in software for normalizing the method-call semantics between application objects residing either in the same address space or remote address space.



The Basic Architecture of CORBA consists of following components:

Interface Repository:

It provides representation of available object interfaces of all objects. It is used for dynamic invocation. It contains interface definitions for registered CORBA server components. Interfaces are represented in a "metadata" format so that they can be discovered dynamically at run time.

Implementation Repository:

It stores implementation details for each object's interfaces. (Mapping from server object name to filename to implement service) The information stored may be OS specific. It is used by object adapter to solve incoming call and activate right object method. It Provides information about registered IDL interfaces to the clients and servers that require it. It is optional for static invocation; required for dynamic invocation only.

Object Request Broker (ORB) core:

It provides mechanisms by which objects can interact with each other transparently. It carries out the request-reply protocol between client and server. It provides operations that enable process to be started and stopped and operations to convert between remote object references and strings. Its functions are:

- Find the object implementation for the request
- Prepare the object implementation to receive the request
- Communicate the data making up the request
- Retrieve results of request

Invocation:

It allows a client to invoke requests on an object whose compile time knowledge of server's interface specification is known. For client, object invocation is similar to local method of invocation, which automatically forwarded to object implementation through ORB, object adapter and skeleton. It has low overhead and is efficient at run time.

Dynamic Invocation:

It allows a client to invoke requests on object without having compile time knowledge of object's interface. The object and interface are detected at run time by inspecting the interface repository. The request is then constructed and invocation is performed as it static invocation. It has high overhead.

Object Adapter:

It is the interface between server object implementation and ORB. Its function are:

- Bridges the gap between CORBA objects and the programming language interfaces of the servant classes.
- Creates remoter object references for the CORBA objects
- Dispatches each RMI to the appropriate servant class via a skeleton and activates objects.
- Assigns a unique name to itself and each object

Skeletons (server) :

An IDL compiler generates skeleton classes in the server's language. It receives request from stubs and pass to the real remote object It dispatch RMI's to the appropriate servant class. It is responsible for marshaling and unmarshalling arguments, results and exceptions.

Client Proxies / Stubs:

It is generated by an IDL compiler in the client language. It lives in the client machine and pretends to be remote object .A proxy class is created for object oriented languages but Stub procedures are created for procedural languages. It is responsible for marshaling and unmarshalling arguments, results and exceptions.

d)Process Resilience:

Resilience is the use of strategies for improving a distributed system's availability. One of the primary goals of resilience is to prevent situations where an issue with one microservice instance causes more issues, which escalate and eventually lead to distributed system failure. This is known as a cascading failure.

2076 Chaitra

Q1) Define Distributed System. Explain the requirements to successfully implement DS to suit to modern computing.

Ans: A distributed system is a computing environment in which various components are spread across multiple computers (or other computing devices) on a network. These devices split up the work, coordinating their efforts to complete the job more efficiently than if a single device had been responsible for the task.

Advantages of Distributed Systems

Some advantages of Distributed Systems are as follows –

- All the nodes in the distributed system are connected to each other. So nodes can easily share data with other nodes.
- More nodes can easily be added to the distributed system i.e. it can be scaled as required.
- Failure of one node does not lead to the failure of the entire distributed system. Other nodes can still communicate with each other.

Resources like printers can be shared with multiple nodes rather than being restricted to just one.

Disadvantages of Distributed Systems

Some disadvantages of Distributed Systems are as follows –

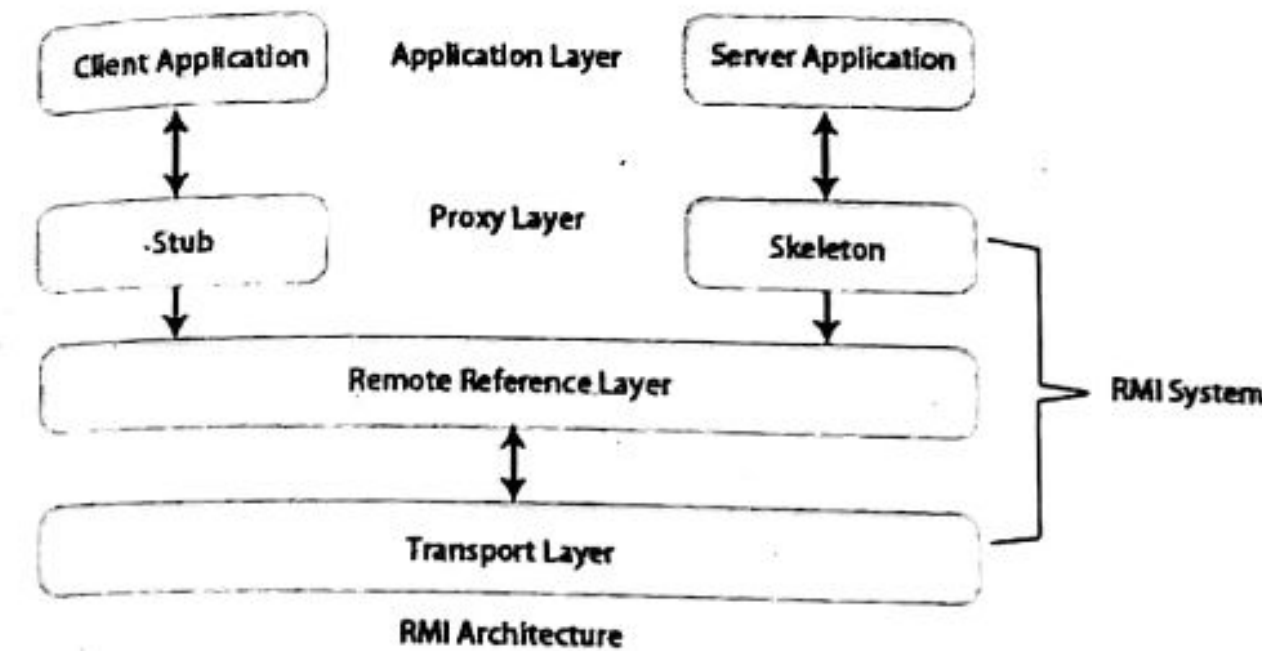
- It is difficult to provide adequate security in distributed systems because the nodes as well as the connections need to be secured.
- Some messages and data can be lost in the network while moving from one node to another.
- The database connected to the distributed systems is quite complicated and difficult to handle as compared to a single user system.
- Overloading may occur in the network if all the nodes of the distributed system try to send data at once.

While implementation Distributed System:

1. Heterogeneity: Heterogeneity is applied to the network, computer hardware, operating system, and implementation of different developers. A key component of the heterogeneous distributed system client-server environment is middleware. Middleware is a set of services that enables application and end-user to interact with each other across a heterogeneous distributed system.

2. **Openness:** The openness of the distributed system is determined primarily by the degree to which new resource-sharing services can be made available to the users. Open systems are characterized by the fact that their key interfaces are published. It is based on a uniform communication mechanism and published interface for access to shared resources. It can be constructed from heterogeneous hardware and software.
3. **Scalability:** Scalability of the system should remain efficient even with a significant increase in the number of users and resources connected.
4. **Security:** Security of information system has three components Confidentiality, integrity and availability. Encryption protects shared resources, keeps sensitive information secrets when transmitted.
5. **Failure Handling:** When some faults occur in hardware and the software program, it may produce incorrect results or they may stop before they have completed the intended computation so corrective measures should be implemented to handle this case. Failure handling is difficult in distributed systems because the failure is partial i, e, some components fail while others continue to function.
6. **Concurrency:** There is a possibility that several clients will attempt to access a shared resource at the same time. Multiple users make requests on the same resources, i.e read, write, and update. Each resource must be safe in a concurrent environment. Any object that represents a shared resource a distributed system must ensure that it operates correctly in a concurrent environment.
7. **Transparency:** Transparency ensures that the distributed system should be perceived as a single entity by the users or the application programmers rather than the collection of autonomous systems, which is cooperating. The user should be unaware of where the services are located and the transferring from a local machine to a remote one should be transparent.

Q 2) What the functionalities provided by RMI software. How is the event and notification system implemented in distributed object-based communication?



RMI stands for Remote Method Invocation. It is an API provided by java that allows an object residing in one JVM (Java Virtual Machine) to access or invoke an object running on another JVM. The other JVM could be on the same machine or remote machine.

Let us now discuss the components of this architecture.

- **Transport Layer** – This layer connects the client and the server. It manages the existing connection and sets up new connections.
- **Stub** – A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- **Skeleton** – This is the object which resides on the server side. stub communicates with this skeleton to pass request to the remote object.
- **RRL (Remote Reference Layer)** – It is the layer which manages the references made by the client to the remote object.

Following are the features of RMI –

- To minimize the complexity of the application.
- To preserve type safety.
- Distributed garbage collection.
- Minimize the difference between working with local and remote objects.

Second part:

Distributed system requires entities which reside in different address spaces potentially on different machines to communicate. Distributed object means that objects reside in separate address spaces and whose methods can be access on remote address space potentially on different machines. The remote

method call is issue in an address space separated from the address space where the target object resides. The code issuing the call is referred to as server object or remote object.

Event: an event occurs at an object of interest as the result of the completion of a method execution. The idea behind the use of events is that one object can react to exchange occurring in another object.

Notification: is an object that contains information about an event. Notifications of events are essentially asynchronous and determined by their receivers.

Distributed event-based systems extend the local event model by allowing multiple objects at different locations to be notified of events taking place at an object.

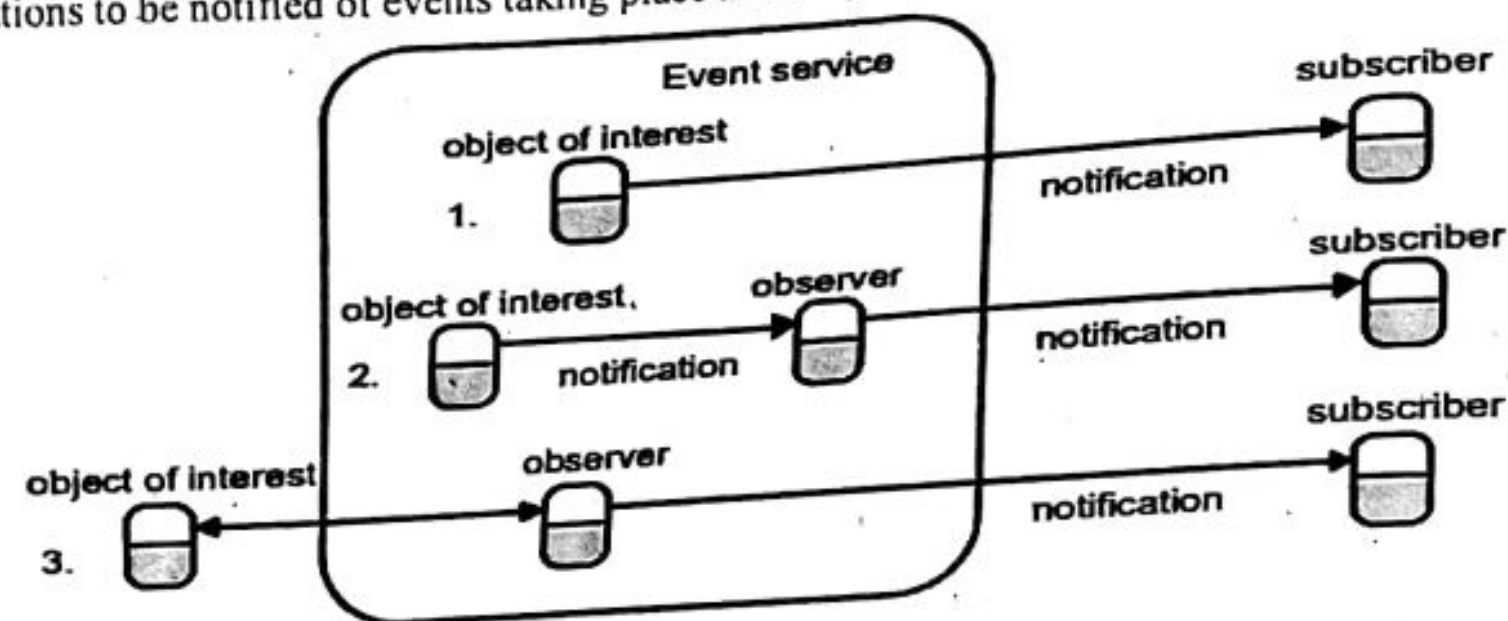


Figure shows an architecture that specifies the roles played by the objects that participate in distributed event-based systems. The main component is an event service that maintains a database of published events and of subscribers' interests.

The roles of the participating objects are as follows:

The object of interest: an object that experiences changes of state, because of its operations being invoked and is considered as part of the event service if it transmits notifications.

Event: an event occurs at an object of interest as the result of the completion of a method execution.

Notification is an object that contains information about an event. Typically, it contains the type of the event and its attributes.

Subscriber: is an object that has subscribed to some type of events in another object.

Observer objects: to decouple an object of interest from its subscribers.

Publisher: an object that declares that it will generate notifications of particular types of events. A publisher may be an object of interest or an observer.

Three cases: An object of interest inside the event service without an observer. It sends notifications directly to the subscribers. An object of interest inside the event service with an observer. The object of interest sends notifications via the observer to the subscribers. An object of interest outside the event service. In this case, an observer queries the object of interest to discover when events occur. The observer sends notifications to the subscribers.

Q 3) What is distributed file system? Explain the principle operations of any modern distributed file system.

Ans: A Distributed File System (DFS) as the name suggests, is a file system that is distributed on multiple file servers or multiple locations. It allows programs to access or store isolated files as they do with the local ones, allowing programmers to access files from any network or computer.

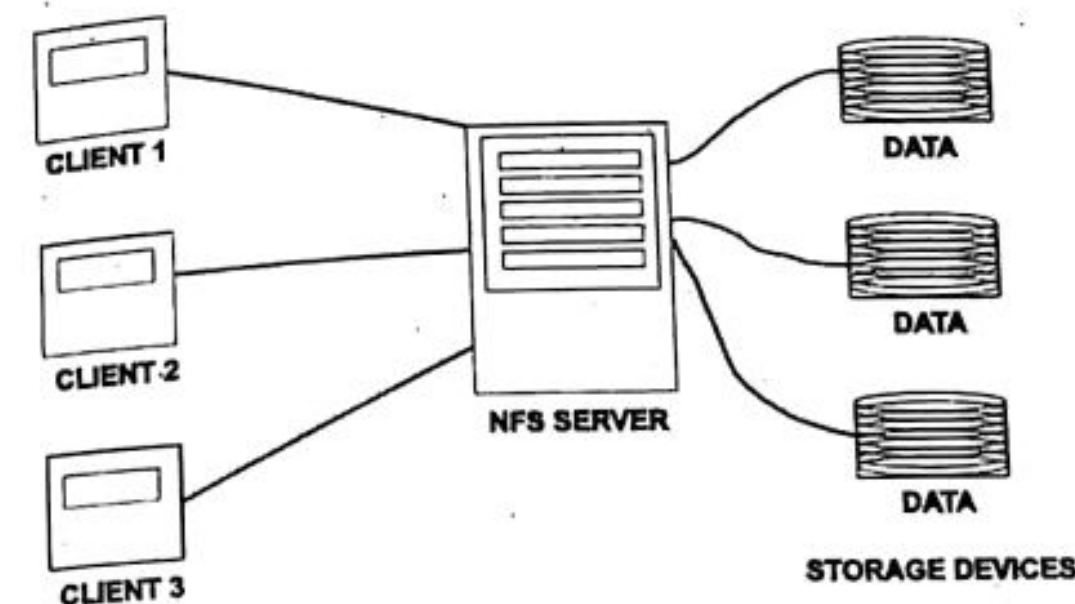


Fig: DFS

Second part:

Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware.

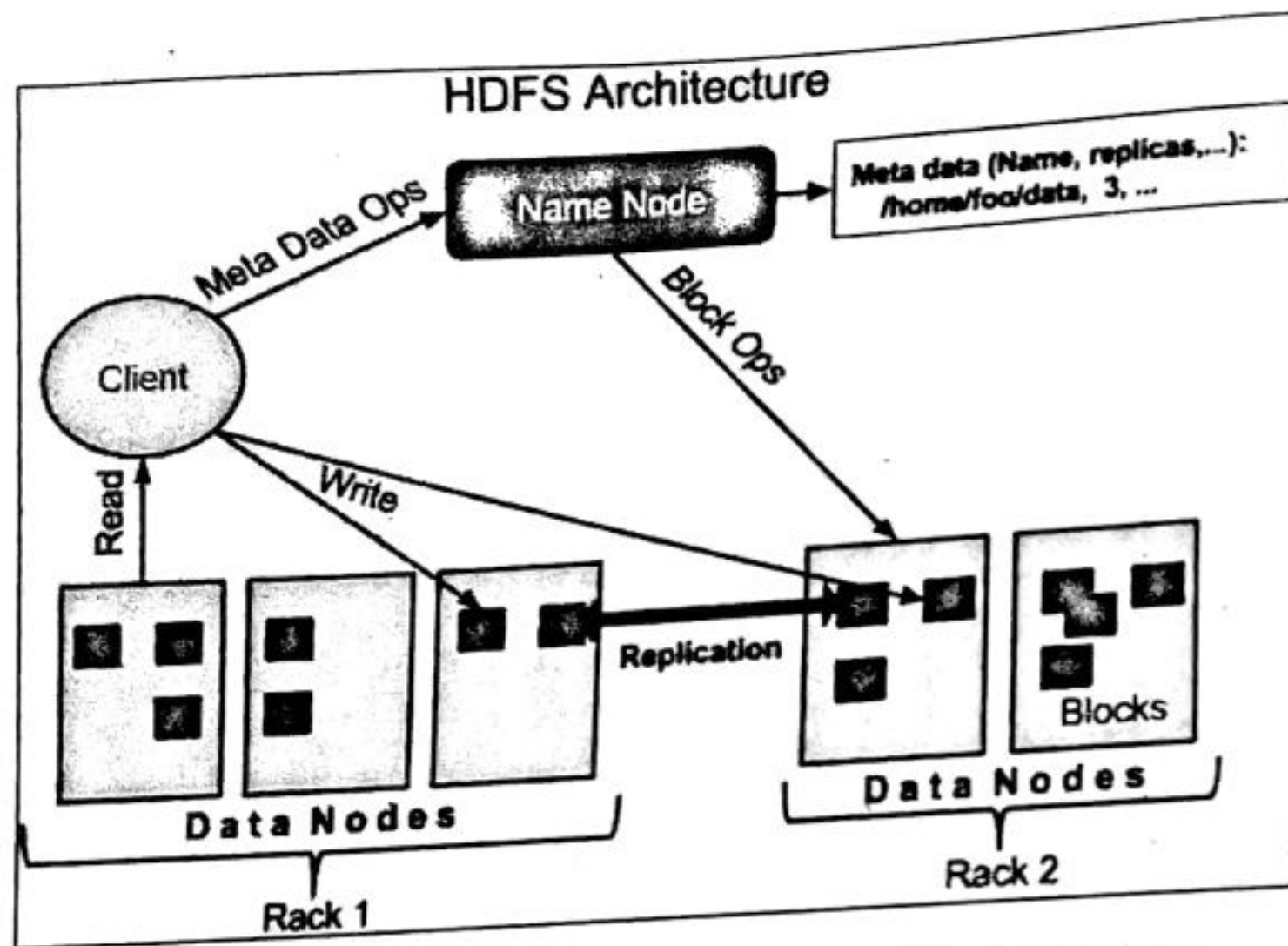
HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

Features of HDFS

- It is suitable for the distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of name node and data node help users to easily check the status of cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

HDFS Architecture

Given below is the architecture of a Hadoop File System.



HDFS follows the master-slave architecture, and it has the following elements.

Name node

The name node is the commodity hardware that contains the GNU Linux operating system and the name node software. It is a software that can be run on commodity hardware. The system having the name node acts as the master server and it does the following tasks –

- Manages the file system namespace.
- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

Data node

The data node is a commodity hardware having the GNU Linux operating system and data node software. For every node (Commodity hardware/System) in a cluster, there will be a data node. These nodes manage the data storage of their system.

- Data nodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the name node.

Block

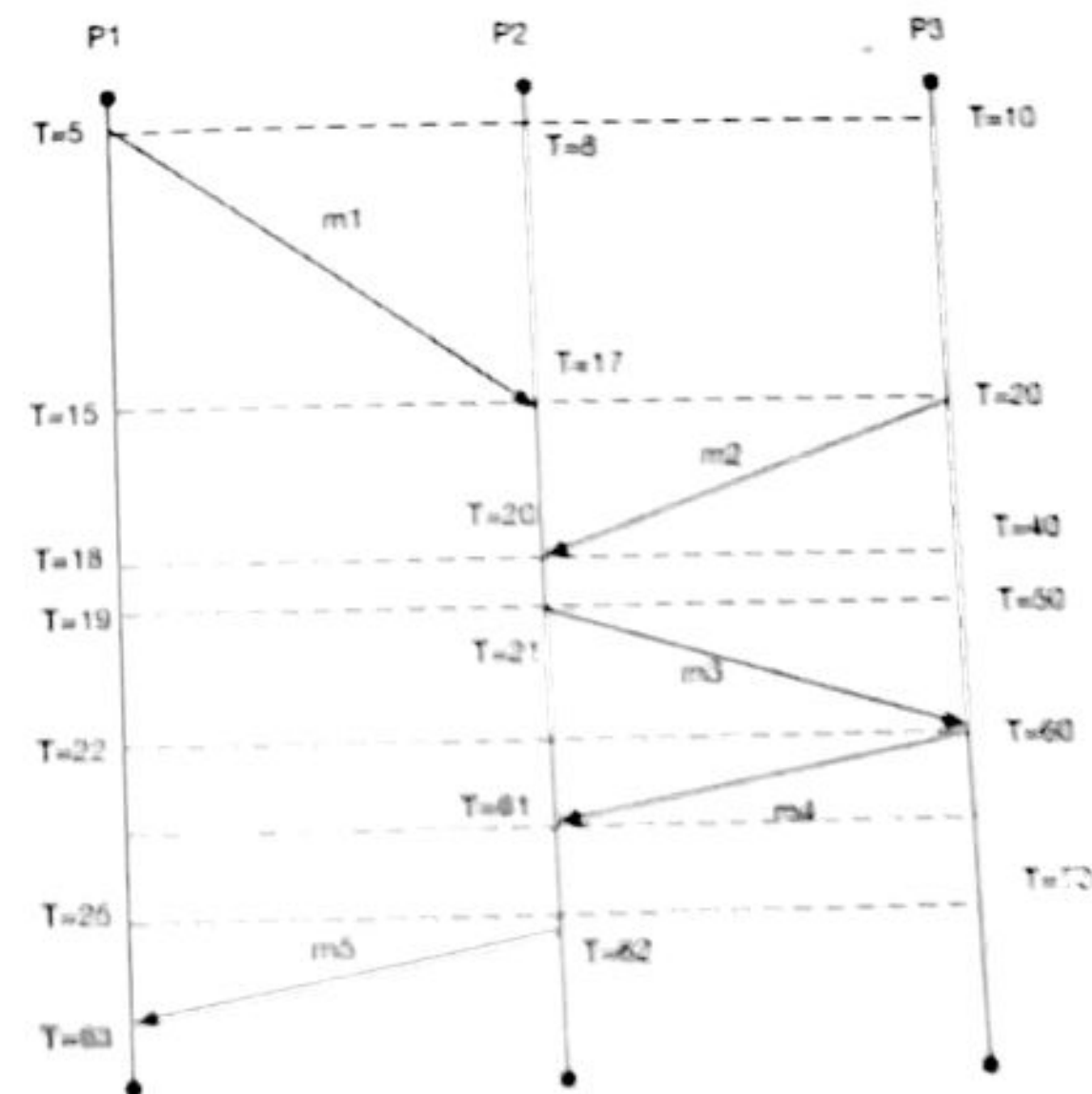
Generally, the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Q 4) What is the issue in Lamport's timestamp? How do you avoid this issue? Explain with alternative algorithm.

Ans: Limitations of Lamport's Logical Clocks

Lamport's logical clocks lead to a situation where all events in a distributed system are totally ordered. That is, if $a \rightarrow b$, then we can say $C(a) < C(b)$.

Unfortunately, with Lamport's clocks, nothing can be said about the actual time of a and b . If the logical clock says $a \rightarrow b$, that does not mean in reality that a actually happened before b in terms of real time.



From this diagram, we can see that $m^1 \rightarrow m^3$. We also know that $C(m^1) < C(m^3)$. We can see that $m^2 \rightarrow m^3$ and that $C(m^2) < C(m^3)$. What we cannot tell here is whether m^1 or m^2 occurred m^3 to be sent.

- The problem with Lamport clocks is that they do not capture causality.

- If we know that $a \rightarrow c$ and $b \rightarrow c$ we cannot say which action-initiated c .
- This kind of information can be important when trying to replay events in a distributed system (such as when trying to recover after a crash).
- The theory goes that if one node goes down, if we know the causal relationships between messages, then we can replay those messages and respect the causal relationship to get that node back up to the state it needs to be in.

Second part and Third part:

Vector Clocks

- Vector clocks allow causality to be captured
- Rules of Vector Clocks:
 - A vector clock $VC(a)$ is assigned to an event a
 - If $VC(a) < VC(b)$ for events a and b , then event a is known to causally precede b .
- Each Process P_i maintains a vector VC_i with the following properties:
 - $VC_i[i]$ is the number of events that have occurred so far at P_i . i.e. $VC_i[i]$ is the local logical clock at process P_i
 - If $VC_i[j] = k$ then P_i knows that k events have occurred at P_j . It is thus P_i 's knowledge of the local time at P_j

Implementing Vector Clocks

- The first property of the vector clock is accomplished by incrementing $VC_i[i]$ at each new event that happens at process P_i
- The second property is accomplished by the following steps:
 0. Before executing any event (sending a message or some internal event), P_i executes $VC_i[i] \leftarrow VC_i[i] + 1$
 1. When process P_i sends a message m to P_j , it sets m 's (vector) timestamp $ts(m) = VC_i$
 2. Upon receiving a message m , process P_j adjusts its own vector by setting $VC_j[k] \leftarrow \max(VC_j[k], ts(m)[k])$ for each k .

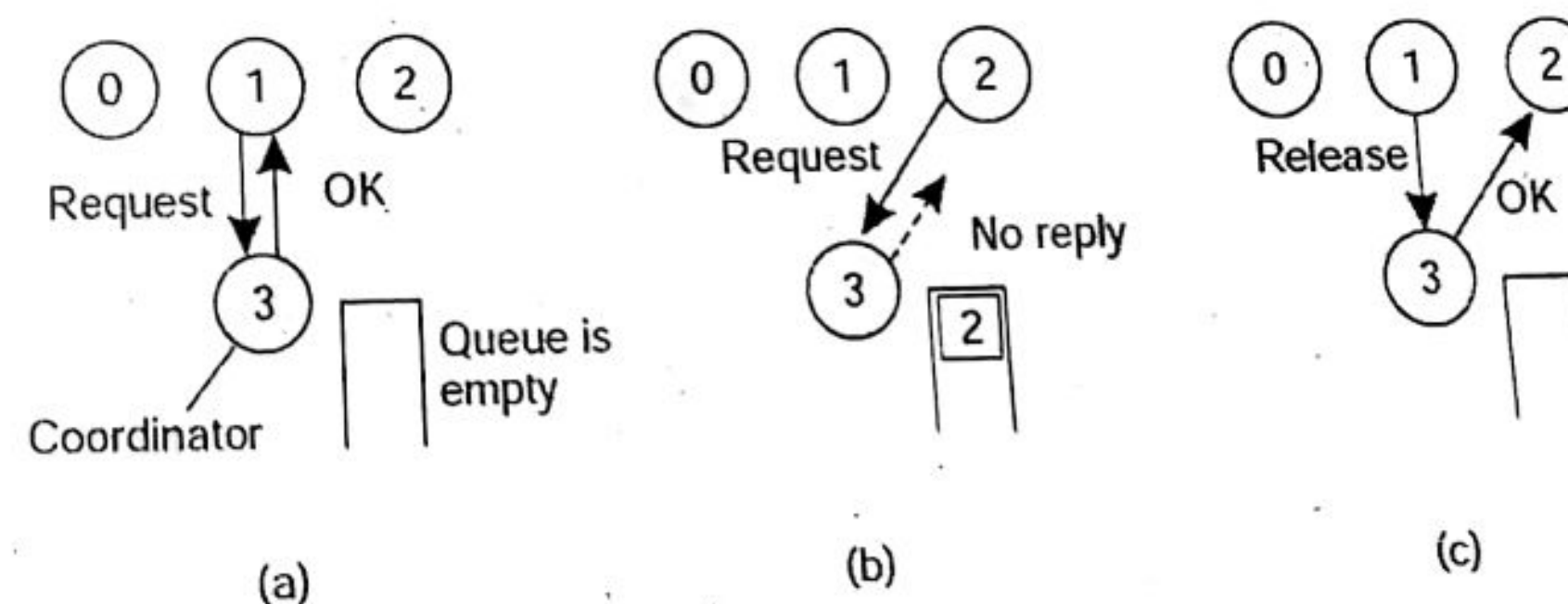
So... What Did We Get Out of All of This?

- We can say if an event a has a timestamp $ts(a)$, then $ts(a)[i] - 1$ denotes the number of events processed at P_i that causally precede a
- This means that when P_j receives a message from P_i with timestamp $ts(m)$, it knows about the number of events that occurred at P_i that causally preceded the sending of m
- Even more importantly, P_j has been told how many events in **other** processes have taken place before P_i sent message m .
- So, this means we could achieve a very important capability in a distributed system: we can ensure that a message is delivered only if all messages that causally precede it have also been received as well.
- We can use this capability to build a truly distributed dataflow graph with dependencies without having a centralized coordinating process.

Q 5) How does a new coordinator elect in executing central coordinator algorithm? How to come to consensus in DS? Explain.

Ans: Centralized coordinator Algorithm

- In centralized algorithm one process is elected as the coordinator which may be the machine with the highest network address.



- Whenever a process wants to enter a critical region, it sends a request message to the coordinator stating which critical region it wants to enter and asking for permission. If no other process is currently in that critical region, the coordinator sends back a reply granting permission, as shown in Fig (a). When the reply arrives, the requesting process enters the critical region.

- Suppose another process 2 shown in Fig (b), asks for permission to enter the same critical region. Now the coordinator knows that a different process is already in the critical region, so it cannot grant permission. The coordinator just refrains from replying, thus blocking process 2, which is waiting for a reply, or it could send a reply saying 'permission denied.'
- When process 1 exits the critical region, it sends a message to the coordinator releasing its exclusive access as shown in Fig (c).
- The coordinator takes the first item off the queue of deferred requests and sends that process a grant message. If the process was still blocked it unblocks and enters the critical region.
- If an explicit message has already been sent denying permission, the process will have to poll for incoming traffic or block later. When it sees the grant, it can enter the critical region.
- **Advantages:**
 - Algorithm guarantees mutual exclusion by letting one process at a time into each critical region.
 - It is also fair as requests are granted in the order in which they are received.
 - No process ever waits forever so no starvation.
 - Easy to implement so it requires only three messages per use of a critical region (request, grant, release).
 - Used for more general resource allocation rather than just managing critical regions.
- **Disadvantages:**
 - The coordinator is a single point of failure, the entire system may go down if it crashes.
 - If processes normally block after making a request, they cannot distinguish a dead coordinator from "permission denied" since no message comes back.
 - In a large system a single coordinator can become a performance bottleneck.

Second part:

Finding Consensus

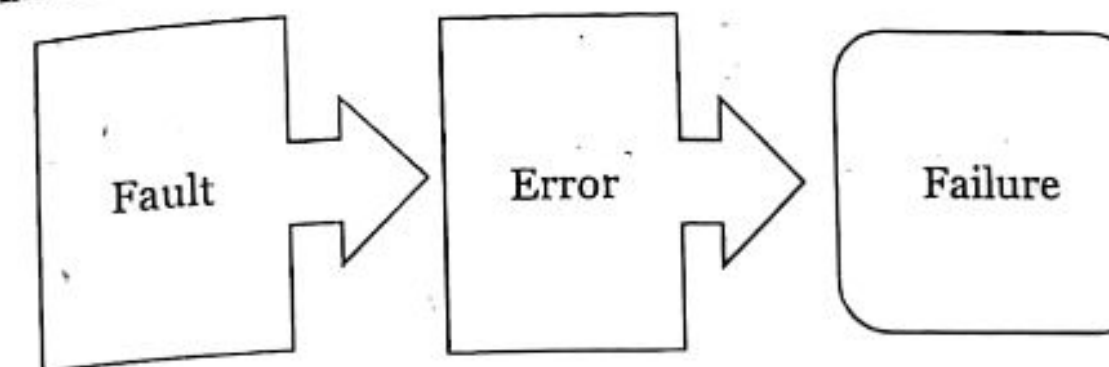
Many cases need agreement.

- Committing a transaction
- dividing tasks
- synchronization

Having the group find agreement protects against fault, but causes its own problems in communication

Q 6) What is fault? How do you implement primary-backup replica system? How is it differ from active replication?

Ans: Failure implies a fault.



Fault types are explained as following below.

1. Method failure:

In this type of failure, the distributed system is generally halted and unable to perform the execution. Sometimes it leads to ending up the execution resulting in an associate incorrect outcome. Method failure causes the system state to deviate from specifications, and also method might fail to progress.

2. System failure:

In system failure, the processor associated with the distributed system fails to perform the execution. This is caused by computer code errors and hardware issues. Hardware issues may involve CPU/memory/bus failure. This is assumed that whenever the system stops its execution due to some fault then the interior state is lost.

3. Secondary storage device failure:

A storage device failure is claimed to have occurred once the keep information can't be accessed. This failure is sometimes caused by parity error, head crash, or dirt particles settled on the medium.

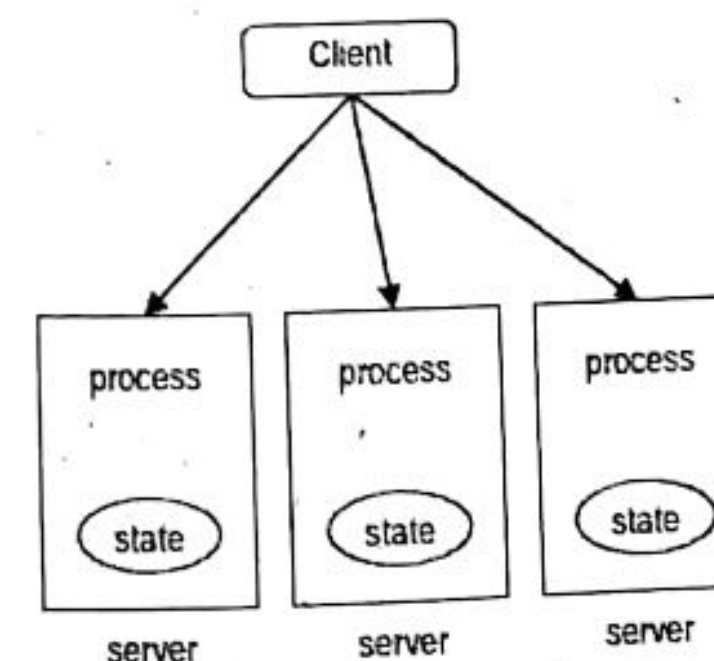
4. Communication medium failure:

A communication medium failure happens once a web site cannot communicate with another operational site within the network. it's typically caused by the failure of the shift nodes and/or the links of the human activity system.

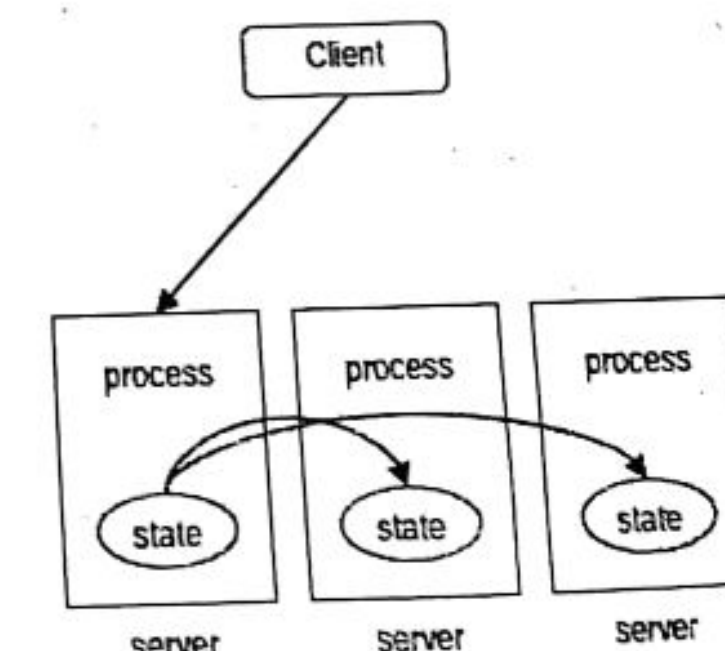
Second and Third part:

In the distributed systems research area replication is mainly used to provide fault tolerance. The entity being replicated is a process. Two replication strategies have been used in distributed systems: **Active** and **Passive** replication.

Active Replication



Passive Replication



In **active replication** each client request is processed by all the servers. Active Replication was first introduced by under the name **state machine replication**. This requires that the process hosted by the servers is **deterministic**. **Deterministic** means that, given the same initial state and a request sequence, all processes will produce the same response sequence and end up in the same final state. To make all the servers receive the same sequence of operations, an **atomic broadcast** protocol must be used. An atomic broadcast protocol guarantees that either all the servers receive a message or none, plus that they all receive messages in the same order. The big disadvantage for active replication is that in practice most of the real-world servers are non-deterministic. Still active replication is the preferable choice when dealing with real-time systems that require quick response even under the presence of faults or with systems that must handle **byzantine faults**.

In **passive replication** there is only one server (called primary) that processes client requests. After processing a request, the primary server updates the state on the other (backup) servers and sends back the response to the client. If the primary server fails, one of the backup servers takes its place. Passive replication may be used even for non-deterministic processes. The disadvantage of passive replication compared to active is that in case of failure the response is delayed.

Q 7) What do you mean by forward and backward recovery? How to implement coordinated checkpointing for recovery in DS?

Ans: Forward Recovery (REDO)

- Forward recovery is also called Roll Forward
- Used in case of physical damage. For example, disk crash, failures during writing of data to database buffers or during flushing buffers to secondary storage.
- Intermediate result of transaction is stored in database buffers. From buffers is transferred to secondary storage.
- Flushing operation of buffers could be triggered by COMMIT operation of transaction or automatically in the event of buffer becoming full.
- If COMMIT is already issued, recovery manager will redo (roll forward) so that updates are written to database.
- Forward recovery guarantees durability of transaction.

Backward Recovery (UNDO)

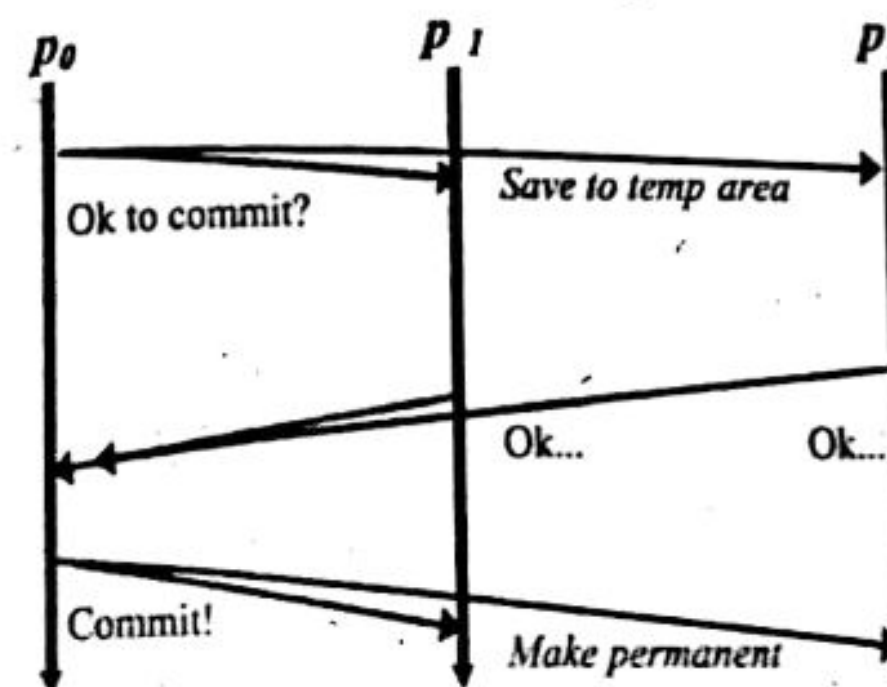
- Backward Recovery is also called Roll-Backward
- Used in case an error occurs during normal operation of database.
- If transaction had not committed at the time of failure, it causes inconsistency in the database.
- Recovery manager must undo (roll backward) any effects of the transaction to the database.
- Backward Recovery guarantees the atomicity property of the transaction.

Second part

Coordinated checkpointing: Process coordinate their checkpoints to save a system-wide consistent state. This consistent set of checkpoints can be used to bound the rollback.

There are two main approaches for checkpointing in the distributed computing systems: coordinated checkpointing and uncoordinated checkpointing. In the coordinated checkpointing approach, processes must ensure that their checkpoints are consistent.

This is usually achieved by two-phase commit protocol algorithm.



Q 8) What are the alternative approaches to avoid possibility of deadlock in distributed system? Explain.

Ans: Deadlock Characteristics

As discussed in the previous post, deadlock has following characteristics.

1. Mutual Exclusion
2. Hold and Wait
3. No preemption
4. Circular wait

Deadlock Prevention

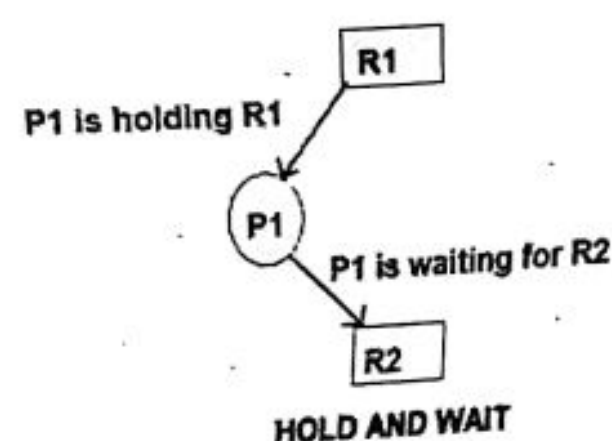
We can prevent Deadlock by eliminating any of the above four conditions.

Eliminate Mutual Exclusion

It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

Eliminate Hold and wait

1. Allocate all required resources to the process before the start of its execution, this way holds and wait condition is eliminated but it will lead to low device utilization. for example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remain blocked till it has completed its execution.
2. The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.



Eliminate No Preemption

Preempt resources from the process when resources required by other high priority processes.

Eliminate Circular Wait

Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing order of numbering.

For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.

Deadlock Avoidance

Deadlock avoidance can be done with Banker's Algorithm.

Banker's Algorithm

Banker's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn't allow the request made by the process.

Inputs to Banker's Algorithm:

1. Max need of resources by each process.
2. Currently, allocated resources by each process.
3. Max free available resources in the system.

The request will only be granted under the below condition:

1. If the request made by the process is less than equal to max need to that process.

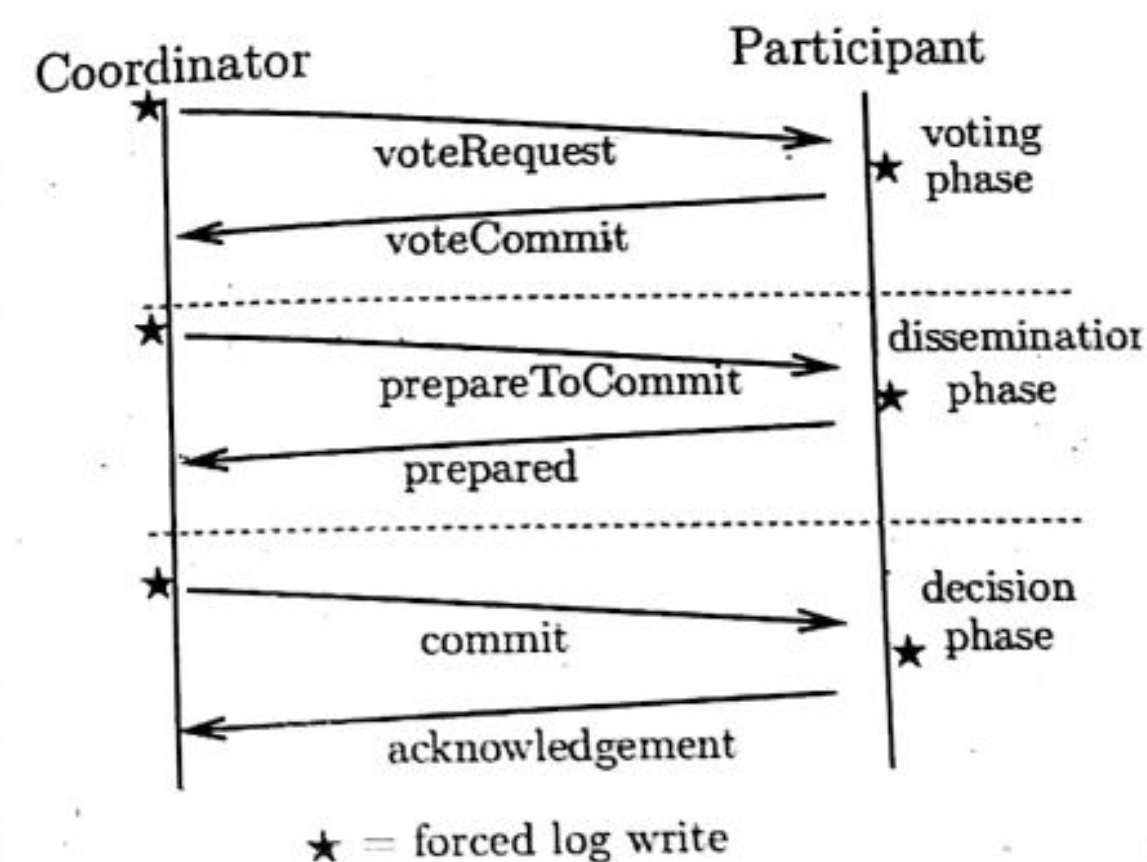
2. If the request made by the process is less than equal to the freely available resource in the system

Q 9)Shorts Notes:

1. Three Phase Commit
2. CORBA components for RMI
3. Cristain's Algorithm
4. Two Phase Commit Protocol
5. MACH

Ans:

Three Phase commit Protocol



In computer networking and databases, the three-phase commit protocol is a distributed algorithm which lets all nodes in a distributed system agree to commit a transaction. It is a more failure-resilient refinement of the two-phase commit protocol.

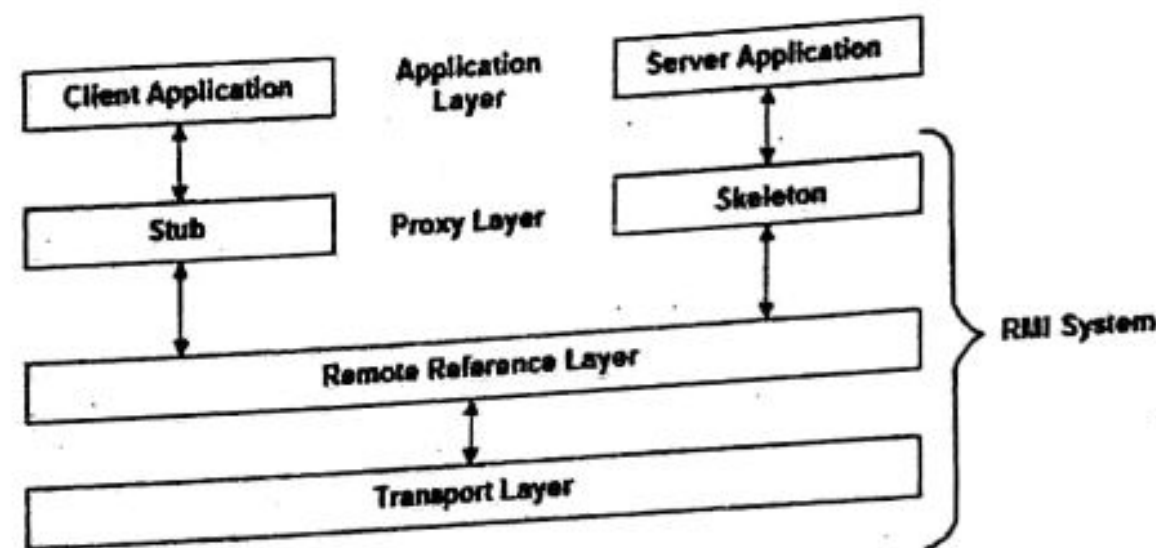
Three-phase commit (3PC) is a synchronization protocol that ensures global atomicity of distributed transactions while alleviating the blocking aspect of 2PC (Two-Phase Commit) in the events of site failures.

b. RMI

The RMI architecture, shown in Figure, describes how remote objects behave and how parameters are passed between remote methods. Remote method invocation allows the program to define separately the behavior and the code that implements the behavior and allows running them on separate JVMs. This

principle facilitates the clients to focus on the definition of a service while the servers can focus on providing the service.

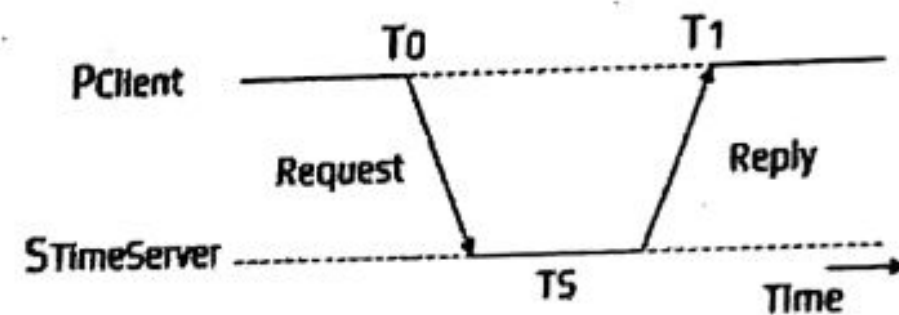
A remote method can be invoked by referring to the remote object. The object is exported through a server application and a remote client, by either looking up the object name in the registry or by checking the value returned from the remote method. This object must implement at least one interface that is extended by the java.rmi.Remote interface. All interactions with the remote object will be performed through this interface only. Basically, this interface describes the methods, which can be invoked remotely, and the remote object then implements it.



Architecture of RMI

c. Cristian's Algorithm

- It makes use of a time server to get the current time and helps in synchronization of computer externally.
- Upon request, the server process S provides the time according to its clock to the requesting process p.
- This method achieves synchronization only if the round trip times between client and time server are sufficiently short compared to the required accuracy.



$$T_{new} = T_{server} + T_1 - T_0 / 2$$

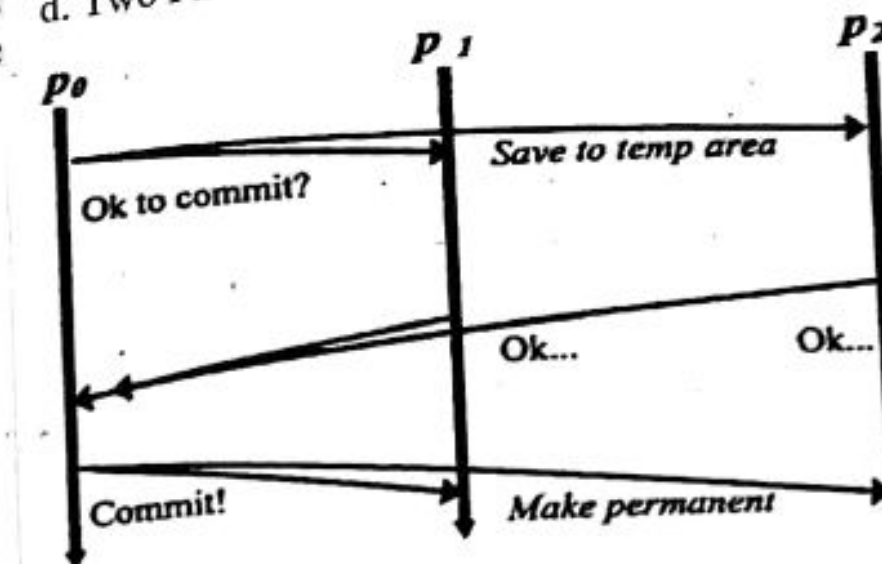
Algorithm:

- A process p requests time in a message mr and receives time value t in a message mt. Process p records total round trip time T(round) taken to send request mr and receive reply mt.
- Assuming elapsed time is splitted before and after S placed t in mt, the time estimate to which p should set its clock is $t + T(\text{round})/2$.
- Assuming min as the earliest point at which S could have placed time mt after p dispatches mr, then:
 - a) Time by S's clock when reply arrives is in range $[t + \text{min}, t + T(\text{round}) - \text{min}]$
 - b) Width of time range is $T(\text{round}) - 2 * \text{min}$
 - c) Accuracy is $\pm (T(\text{round}) / 2 - \text{min})$

Discussion:

- If a time server fails, the synchronization is impossible.
- To remove this drawback, time should be provided by a group of synchronized time servers.

d. Two Phase commit Protocol



Coordinator :

multicast: ok to commit?
collect replies
all ok => send commit
else => send abort

Participant:

ok to commit =>
save to temp area, reply ok
commit =>
make change permanent
abort =>
delete temp area

In transaction processing, databases, and computer networking, the two-phase commit protocol is a type of atomic commitment protocol. It is a distributed algorithm that coordinates all the processes that participate in a distributed atomic transaction on whether to commit or abort the transaction.

e. MACH

- First Generation micro-kernel
- Builds operating system above minimal kernel
- Kernel provides only fundamental services
- These services basic but powerful enough to be used on, wide range of architectures
- Aids distributed computing and multiprocessing

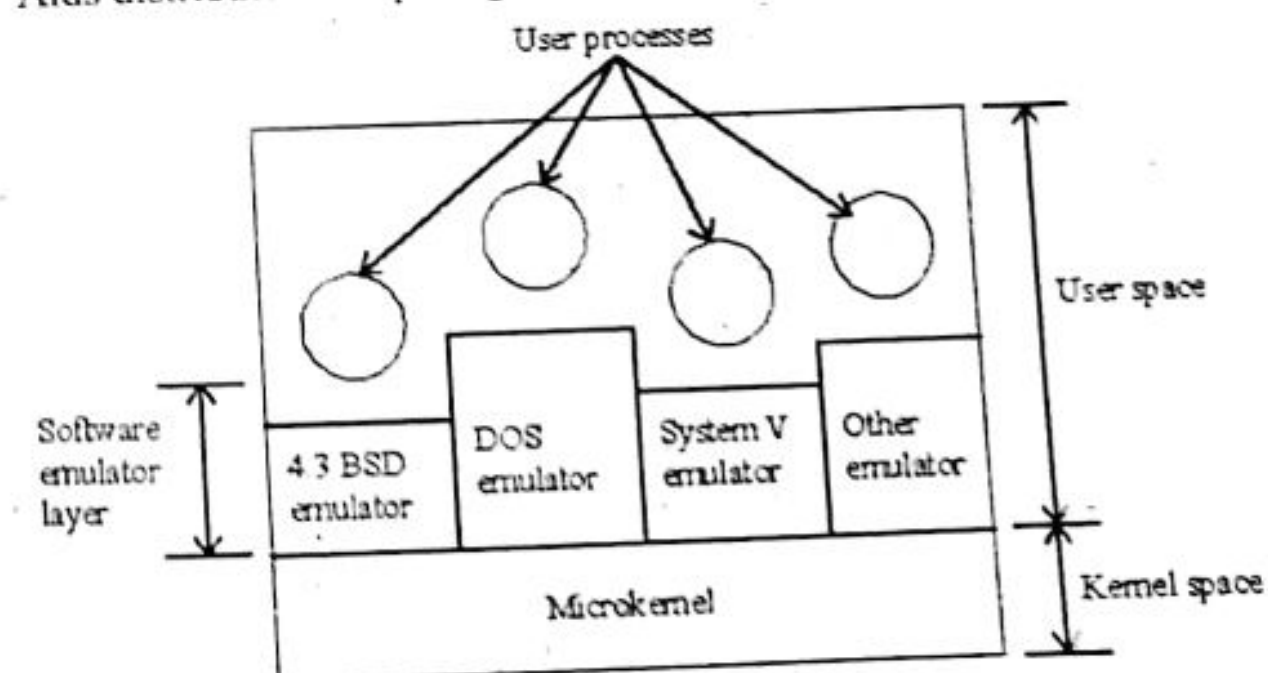


Fig: Mach Architecture

Q 1) What are the major goals of the distributed system and what are the challenges during the design of the distributed system?

Ans: A **distributed system** should easily connect users to resources, it should hide the fact that resources are distributed across a network, must be open, and must be scalable. The main goal of a distributed system is to make it easy for users to access remote resources, and to share them with other users in a controlled manner. Resources can be virtually anything, typical examples of resources are printers, storage facilities, data, files, web pages, and networks. Besides, the major goals are:

- 1) **Connecting Users and Resources**
- 2) **Transparency**
- 3) **Openness:**
- 4) **Scalability:**
- 5) **Reliability**
- 6) **Performance**

The challenges during the design of the distributed system are:

1. **Heterogeneity:** Heterogeneity is applied to the network, computer hardware, operating system and implementation of different developers. A key component of the heterogeneous distributed system client-server environment is middleware.
2. **Openness:** The openness of the distributed system is determined primarily by the degree to which new resource-sharing services can be made available to the users. Open systems are characterized by the fact that their key interfaces are published. It is based on a uniform communication mechanism and published interface for access to shared resources. It can be constructed from heterogeneous hardware and software.
3. **Scalability:** Scalability of the system should remain efficient even with a significant increase in the number of users and resources connected.
4. **Security:** Security of information system has three components Confidentiality, integrity and availability. Encryption protects shared resources, keeps sensitive information secrets when transmitted.

5. **Failure Handling:** When some faults occur in hardware and the software program, it may produce incorrect results or they may stop before they have completed the intended computation so corrective measures should be implemented to handle this case. Failure handling is difficult in distributed systems because the failure is partial i.e., some components fail while others continue to function.
6. **Concurrency:** There is a possibility that several clients will attempt to access a shared resource at the same time. Multiple users make requests on the same resources, i.e. read, write, and update. Each resource must be safe in a concurrent environment. Any object that represents a shared resource in a distributed system must ensure that it operates correctly in a concurrent environment.
7. **Transparency:** Transparency ensures that the distributed system should be perceived as a single entity by the users or the application programmers rather than the collection of autonomous systems, which is cooperating. The user should be unaware of where the services are located and the transferring from a local machine to a remote one should be transparent.

Q2) Define the distributed objects and explain communication between distributed systems.

Ans:

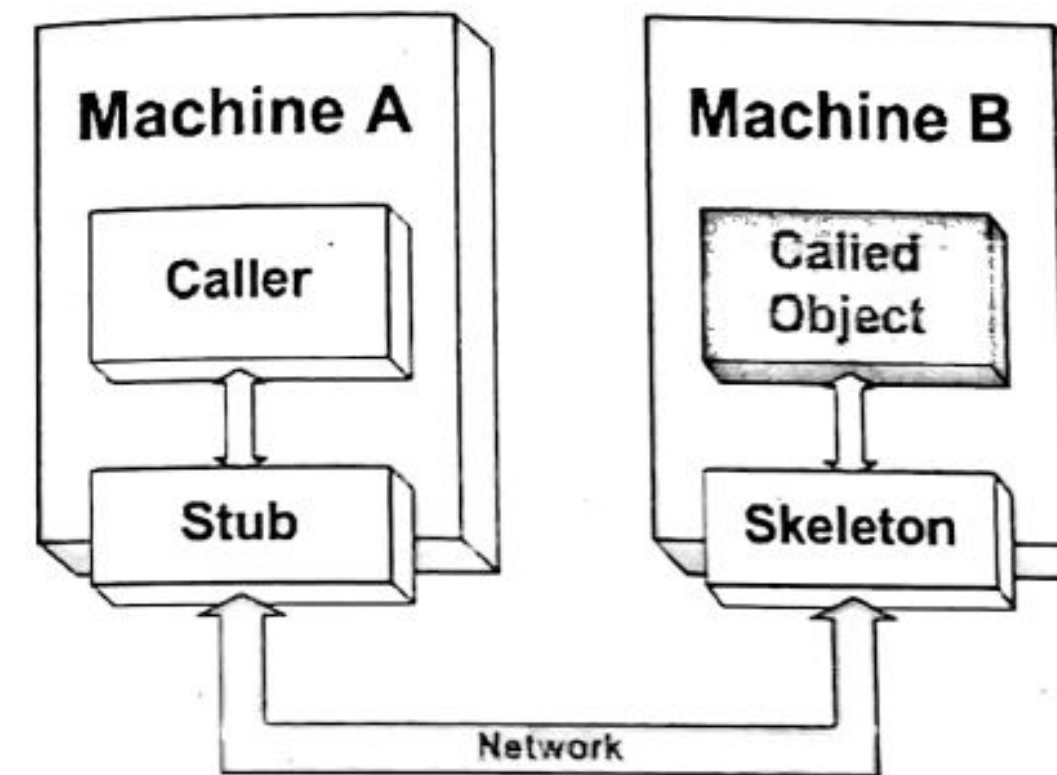
In distributed computing, distributed objects are objects (in the sense of object-oriented programming) that are distributed across different address spaces, either in different processes on the same computer, or even in multiple computers connected via a network, but which work together by sharing data and invoking.

Second part,

In details, the communication consists of several steps:

1. caller calls a local procedure implemented by the stub
2. stub marshalls call type and the input arguments into a request message
3. client stub sends the message over the network to the server and blocks the current execution thread
4. server skeleton receives the request message from the network
5. skeleton unpacks call type from the request message and looks up the procedure on the called object
6. skeleton unmarshalls procedure arguments
7. skeleton executes the procedure on the called object
8. called object performs a computation and returns the result

9. skeleton packs the output arguments into a response message
10. skeleton sends the message over the network back to the client
11. client stub receives the response message from the network
12. stub unpacks output arguments from the message
13. stub passes output arguments to the caller, releases execution thread and caller then continues in execution



Q 3) Define distributed file system. Point out difference between stateless and stateful services.

A **Distributed File System (DFS)** as the name suggests, is a file system that is distributed on multiple file servers or multiple locations. It allows programs to access or store isolated files as they do with the local ones, allowing programmers to access files from any network or computer.

The main purpose of the Distributed File System (DFS) is to allow users of physically distributed systems to share their data and resources by using a Common File System. A collection of workstations and mainframes connected by a Local Area Network (LAN) is a configuration on Distributed File System. A DFS is executed as a part of the operating system. In DFS, a namespace is created, and this process is transparent for the clients.

Second part,

Stateless

In stateless, the server is not required to retain the information about the state.

It is easy to scale the architecture.

The server and client are independent i.e., loosely coupled.

It handles crashes well, so it is easy to restart a server after a crash.

Server's design, architecture, and implementation is straightforward.

Stateful

In stateful, the server is required keep information about the current state and session.

It is not easy to scale the architecture.

In stateful, server and client are not independent i.e., tightly bound.

It does not handle crashes well, so management of crashes is difficult.

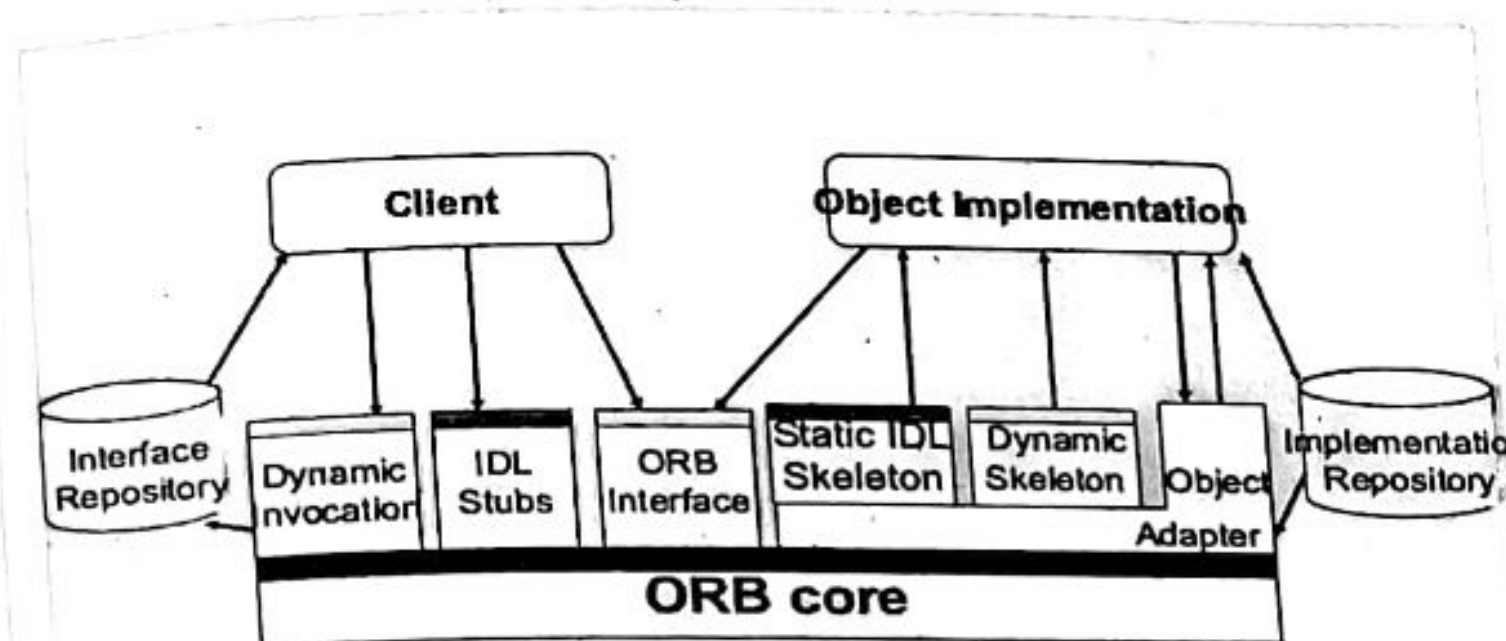
Server's design, architecture, and implementation is complex.

Q 4) Explain the components in CORBA architecture with a diagram.

Ans:

CORBA stands for Common Object Request Broker Architecture. It is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

CORBA architecture:



The Basic Architecture of CORBA consists of following components:

Interface Repository:

It provides representation of available object interfaces of all objects. It is used for dynamic invocation. It contains interface definitions for registered CORBA server components. Interfaces are represented in a "metadata" format so that they can be discovered dynamically at run time.

Implementation Repository:

It stores implementation details for each object's interfaces. (Mapping from server object name to filename to implement service) The information stored may be OS specific.

Object Request Broker (ORB) core:

It provides mechanisms by which objects can interact with each other transparently. It carries out the request-reply protocol between client and server. It provides operations that enable process to be started and stopped and operations to convert between remote object references and strings. Its functions are:

- Find the object implementation for the request
- Prepare the object implementation to receive the request
- Communicate the data making up the request
- Retrieve results of request

Static Invocation:

It allows a client to invoke requests on an object whose compile time knowledge of server's interface specification is known.

Dynamic Invocation:

It allows a client to invoke requests on object without having compile time knowledge of object's interface.

Object Adapter:

It is the interface between server object implementation and ORB. Its function are:

- Bridges the gap between CORBA objects and the programming language interfaces of the servant classes.
- Creates remoter object references for the CORBA objects
- Dispatches each RMI to the appropriate servant class via a skeleton, and activates objects.
- Assigns a unique name to itself and each object

Skeletons (server):

An IDL compiler generates skeleton classes in the server's language.

Client Proxies / Stubs:

It is generated by an IDL compiler in the client language. It lives in the client machine and pretends to be remote object.

Q 5) Explain lamport's logical clock with its pros and cons.

Ans

A Lamport logical clock is a **numerical software counter value maintained in each process**. Conceptually, this logical clock can be thought of as a clock that only has meaning in relation to messages moving between processes. When a process receives a message, it re-synchronizes its logical clock with that sender.

Algorithm:

- **Happened before relation(\rightarrow):** $a \rightarrow b$, means 'a' happened before 'b'.
- **Logical Clock:** The criteria for the logical clocks are:
 - [C1]: $C_i(a) < C_i(b)$, [$C_i \rightarrow$ Logical Clock, If 'a' happened before 'b', then time of 'a' will be less than 'b' in a particular process.]
 - [C2]: $C_i(a) < C_j(b)$, [Clock value of $C_i(a)$ is less than $C_j(b)$]

Implementation Rules[IR]:

- [IR1]: If $a \rightarrow b$ ['a' happened before 'b' within the same process] then, $C_i(b) = C_i(a) + d$
- [IR2]: $C_j = \max(C_j, t_m + d)$ [If there's more number of processes, then t_m = value of $C_i(a)$, $C_j = \max$ value between C_j and $t_m + d$]

Lamport's clock has the advantage of **requiring no changes in the behavior of the underlying protocol**.

The disadvantage that clocks are entirely under the control of the logical-clock protocol and may as a result make huge jumps when a message is received.

Q 6) Explain reliable multicast with its properties and an algorithm.

Reliable Multicast -

- Must have the following properties:
 - Integrity: A working process p in group g delivers m at most once, and m was multicast by some working process –
 - Agreement: If a working process delivers m then all other working processes in group g will deliver m
- What is the point of having reliable multicast?
 - We ensure that one process can communicate with all others
 - Application programmer does not have to worry about it

Reliable multicast is the one in which the sending process is in the multicast group.

- Nodes may fail (by crashing)
- We will use one to one communication between processes
 - The communication is reliable (may be using suitable ack based protocol)
 - If both processes are alive, the message gets delivered. i.e. the network does not fail
- Note that these assumptions are necessary.
 - If network and message delivery can fail, then there may be 2 sets of processes who never communicate with each other
 - Thus message from one set will never reach the other.

Reliable multicast use Bmulticast as function/procedure :

- Implement Rmulticast(g,m) and Rdeliver(m).

Algorithm:

- Initialization: Received={}
- p.Rmulticast(g,m):
 - p.Bmulticast(g, m)
- Q.Bdeliver(m):
 - If m is not in Received:
 - Received = Received \cup {m}
 - If $p \neq q$: q.Bmulticast(g,m)
 - q.Rdeliver(m)

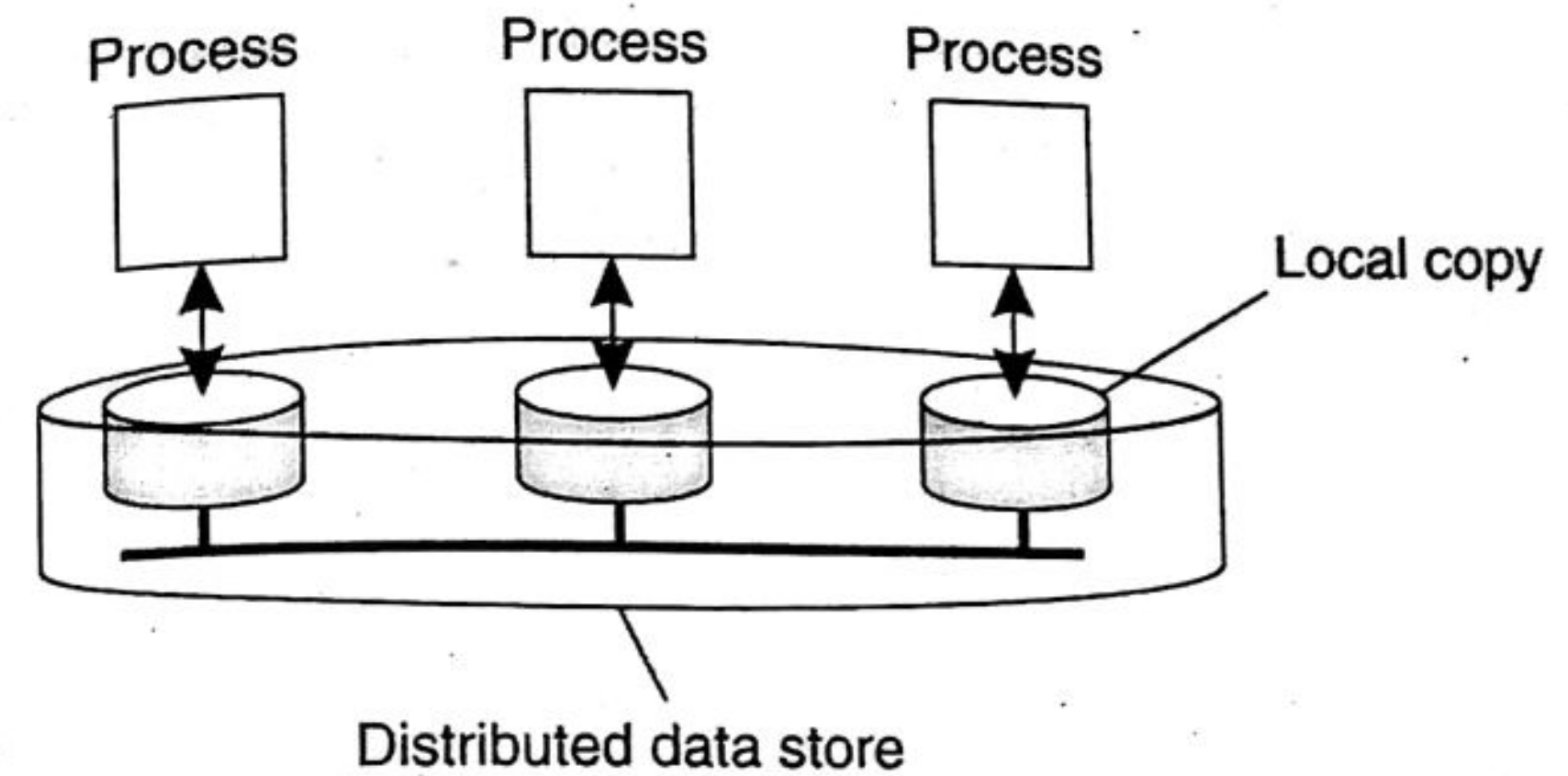
The key point is that q sends the message to other working nodes before it accepts the message and delivers to the interested application

Q 7)Specify data centric consistency models and explain any one of them in detail.

Consistency model:

- A consistency model is essentially a contract between the software and the memory.
- If the software agrees to obey certain rules, the memory promises to work correctly.
- If the software violates these rules, correctness of memory operation is no longer guaranteed.

The general organization of a logical data store, physically distributed and replicated across multiple processes which is shown in figure below:



The various models are:

- 1)Continuous Consistency
- 2)Strict Consistency
- 3) Sequential Consistency
- 4)Causal Consistency
- 5)FIFO Consistency
- 6)Weak Consistency
- 7)Release Consistency
- 8)Entry consistency

Second part

FIFO Consistency

Writes done by a single process are seen by all other processes in the order in which they were issued but writes from different processes may be seen in a different order by different processes.

Also called "PRAM Consistency" – Pipelined RAM.

Easy to implement - There are no guarantees about the order in which different processes see writes – except those two or more writes from a single process must be seen in order.

Easy to implement - There are no guarantees about the order in which writes are seen, except those two or more writes from a single process must be seen in order.

P1:	W(x)a			
P2:	R(x)a	W(x)b	W(x)c	
P3:			R(x)b	R(x)a R(x)c
P4:			R(x)a	R(x)b R(x)c

A valid sequence of FIFO consistency events.

Note that none of the consistency models given so far would allow this sequence of events.

Q 8) Why it is necessary to maintain transaction? What is deadlock and what are phantom deadlocks.

Distributed transactions are necessary when you need to quickly update related data that is spread across multiple databases.

- Transaction: specified by a client as a set of operations on objects to be performed as an indivisible unit by the servers managed those objects.
- Goal of transaction:
 - ensure all the objects managed by a server remain in a consistent state when accessed by multiple transactions and in the presence of server crashes.
 - Maximize concurrency: transactions are allowed to execute concurrently if they would have the same effect as a serial execution.
 - Transaction applies to recoverable objects and intended to be atomic (atomic transaction);

Note: Recoverable objects: objects can be recovered after their server crashes.

- Atomic operations: operations that are free from interference from concurrent operations being performed in the other threads.

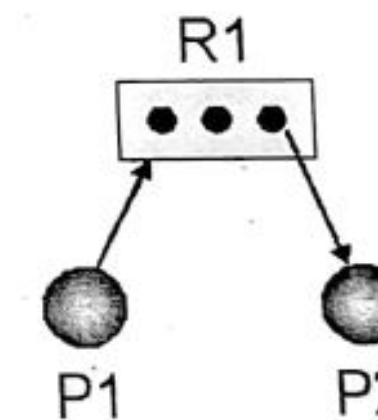
- **Atomicity**- Transaction must be all or none
- **Consistency**- Transaction of the system takes one consist to another consistent state.
- **Isolation** -one process could not hamper other
- **Durability** – storing the transaction's log report in non- volatile storage)

Second part, .

A Blocked Process which can never be resolved unless there is some outside Intervention is deadlock.

For e.g.

In the fig below resource (reusable) R1 is requested by Process P1 but is held by Process P2.



There are 2 situation that cause deadlocks –

- 1 – The lack of requested resource is one cause. Which is called the resource deadlock.
- 2 – The other type is caused due to communication, in which case the process waits for a certain message before it can proceed, which is called communication deadlock.

Phantom deadlock:

In distributed deadlock detection, the delay in propagating local information might cause the deadlock detection algorithms to identify deadlocks that do not really exist. Such situations are called phantom deadlocks and they lead to unnecessary aborts.

Phantom Deadlock is a deadlock that occurs in a Distributed DBMS due to communication delays between different processes and leads to unnecessary process abortions. In fact, it is not really a deadlock.

Now, let us understand how phantom deadlock occurs with the help of an example.

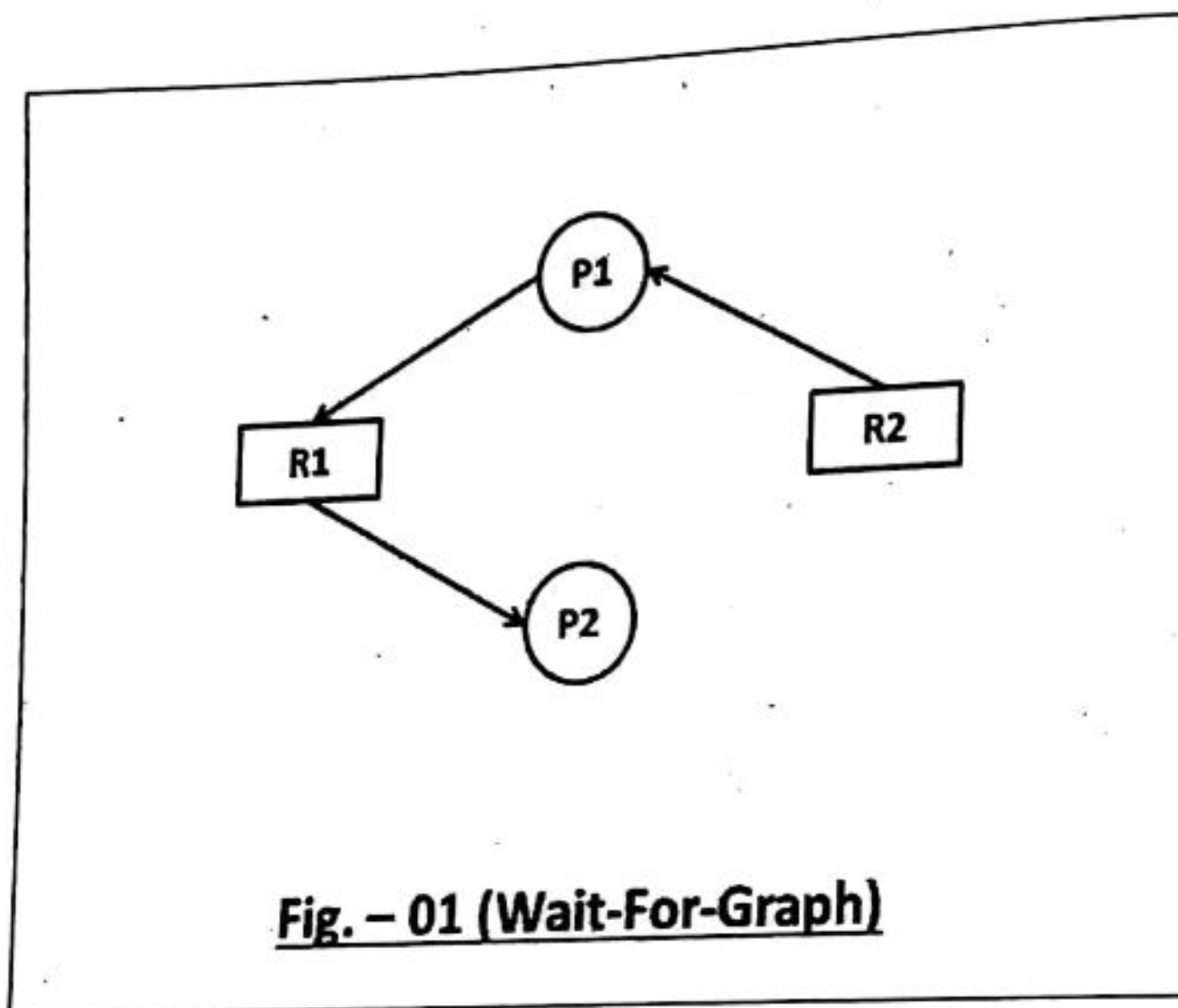


Fig. - 01 (Wait-For-Graph)

The above diagram is a wait-for-graph for two processes - P1 and P2 respectively. You can also see two resources R1 and R2. Process P1 holds the resource R2, and request for the resource R1. On the other hand, Process P2 holds the resource R1. Currently, there is no-deadlock in the system.

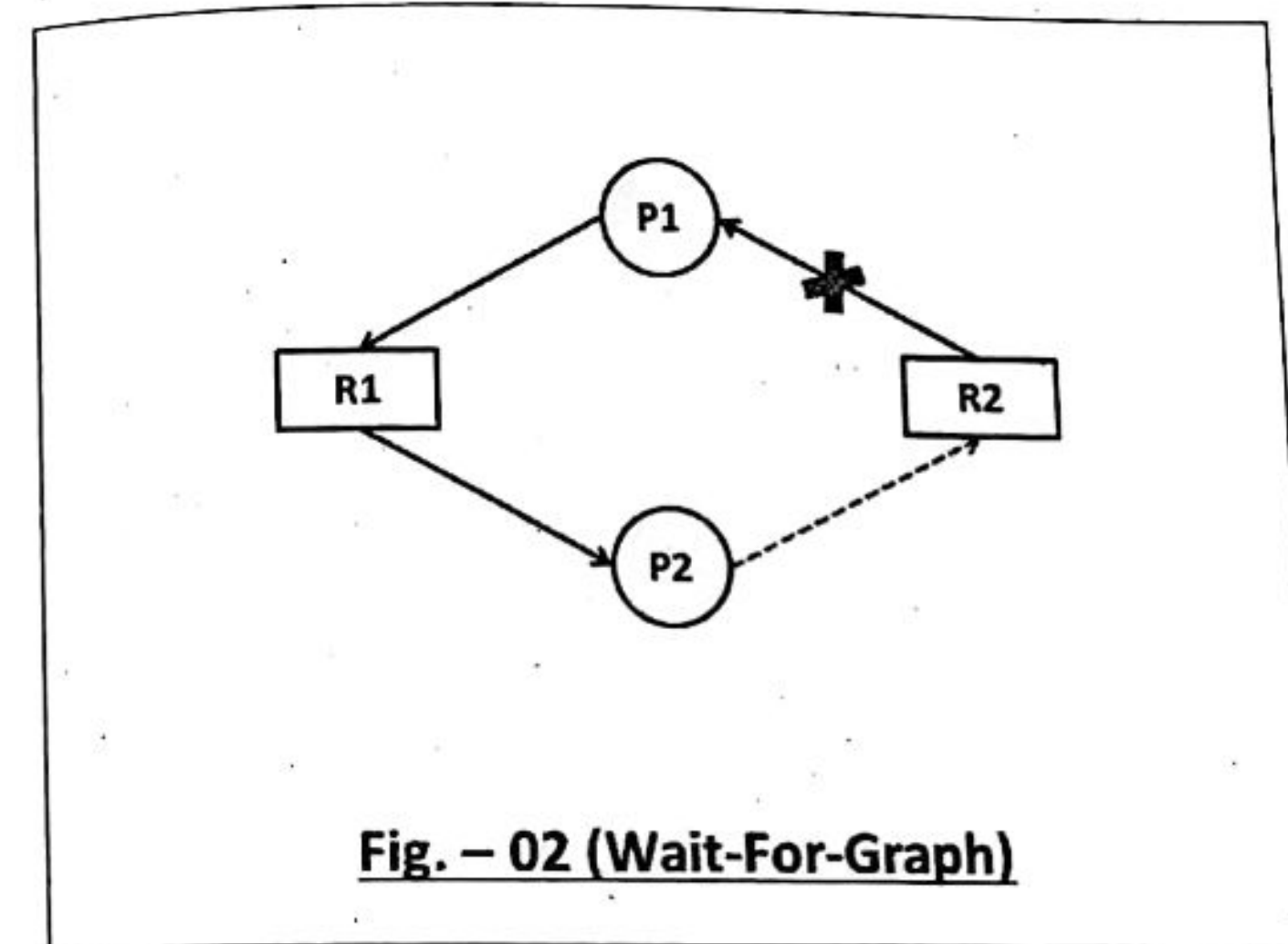


Fig. - 02 (Wait-For-Graph)

Now in the above diagram, you can see, process P1 releases resource R2 and process P2 has requested resource R2. In practical scenarios, P1 will send a release message, and P2 will send a request message for resource R2 to the centralized system.

Now, assume the case where the resource request message has arrived for resource R2 first. On the other hand, the release message has not yet arrived for the resource R2. In such a case, the deadlock detection algorithm will treat it as a deadlock and will unnecessarily abort some processes to break this deadlock. But in actual, there is no such deadlock.

Q 9) what is fault tolerance? Explain the different types of fault that may occur in a distributed system.

Fault Tolerance simply means a system's ability to continue operating uninterrupted despite the failure of one or more of its components. This is true whether it is a computer system, a cloud cluster, a network, or something else.

Fault tolerance is a process that enables an operating system to respond to a failure in hardware or software. This fault-tolerance definition refers to the system's ability to continue operating despite failures or malfunctions.

The different types of fault that may occur in distributed system are:

- Node (hardware) faults
- Program (software) faults
- Communication faults
- Timing faults

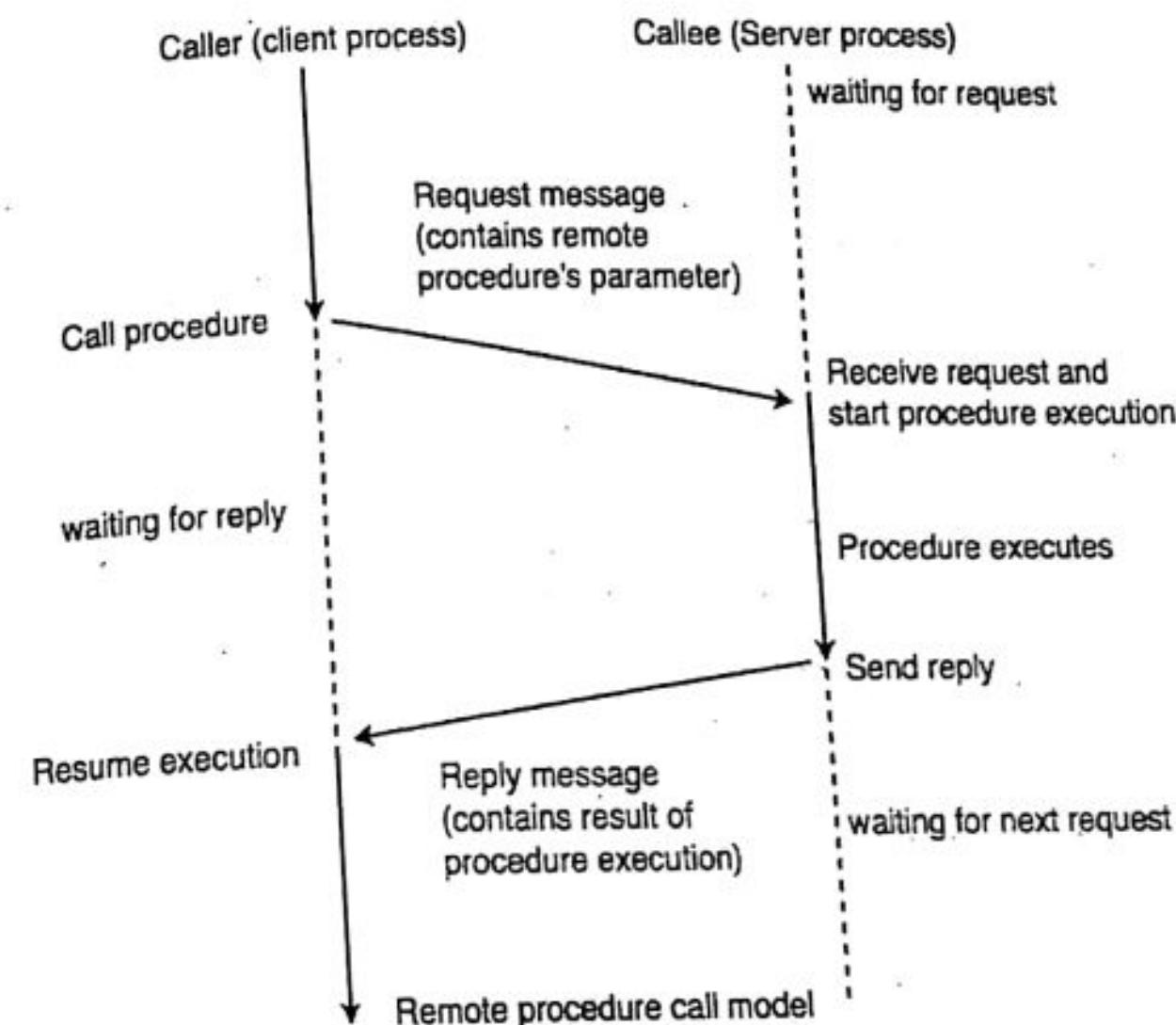
Q10) Write short notes on:

a) RPC:

(RPC) is a communication technology that is used by one program to make a request to another program for utilizing its service on a network without even knowing the network's details. A function call or a subroutine call are other terms for a procedure call.

Advantages of RPC:

1. RPC provides abstraction i.e message-passing nature of network communication is hidden from the user.
2. RPC often omits many of the protocol layers to improve performance. Even a small performance improvement is important because a program may invoke RPCs often.
3. RPC enables the usage of the applications in the distributed environment, not only in the local environment.
4. With RPC code re-writing / re-developing effort is minimized.
5. Process-oriented and thread oriented models supported by RPC.



b) Monolithic and micro kernel:

A monolithic kernel is an operating system architecture where the entire operating system is working in kernel space. The monolithic model differs from other operating system architectures, such as the microkernel architecture, in that it alone defines a high-level virtual interface over computer hardware.

A set of primitives or system calls implement all operating system services such as process management, concurrency, and memory management. Device drivers can be added to the kernel as modules.

Here are the following advantages of a monolithic kernel, such as:

- o The execution of the monolithic kernel is quite fast as the services such as memory management, file management, process scheduling, etc., are implemented under the same address space.
- o A process runs completely in single address space in the monolithic kernel.

The monolithic kernel is a static single binary file.

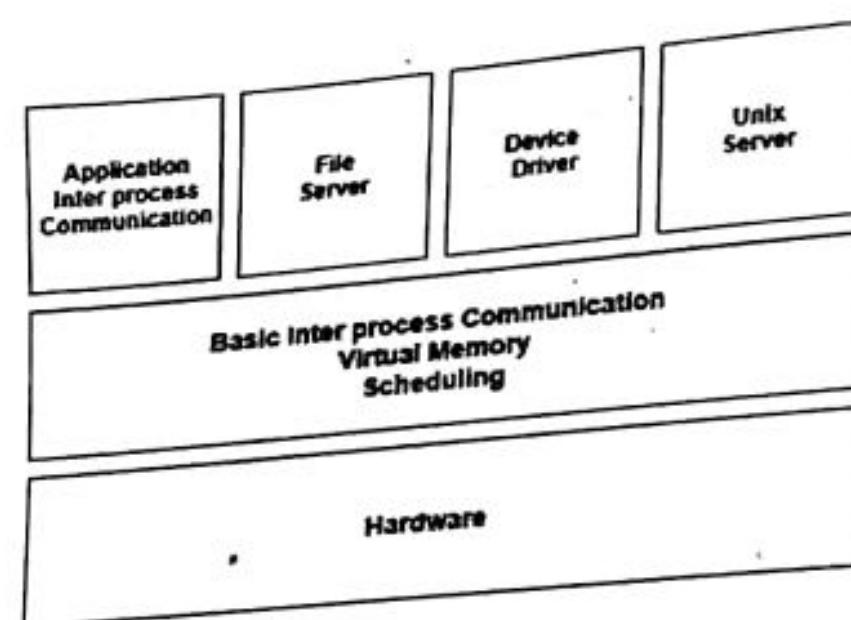
Here are some disadvantages of the monolithic kernel, such as:

- o If any service fails in the monolithic kernel, it leads to the failure of the entire system.
- o The entire operating system needs to be modified by the user to add any new service.

Microkernel

A microkernel is the minimum software that is required to correctly implement an operating system. This includes memory, process scheduling mechanisms and basic inter-process communication.

A diagram that demonstrates the architecture of a microkernel is as follows –



Microkernel Based Operating System

In the above diagram, the microkernel contains basic requirements such as memory, process scheduling mechanisms and basic interprocess communication. The only software executing at the privileged level i.e. kernel mode is the microkernel. The other functions of the operating system are removed from the kernel mode and run in the user mode. These functions may be device drivers, file servers, application interprocess communication etc.

Some of the benefits of microkernels are –

- Microkernels are modular and the different modules can be replaced, reloaded, modified, changed etc. as required. This can be done without even touching the kernel.
- Microkernels are quite secure as only those components are included that would disrupt the functionality of the system otherwise.
- Microkernels contain fewer system crashes as compared to monolithic systems. Also, the crashes that do occur can be handled quite easily due to the modular structure of microkernels.

c)Mach

Mach is a microkernel designed to be the core of a **distributed UNIX-like system**. It communicates via sending messages directly between different applications via ports and handles only the most basic

operating system functions on its own. More advanced functions are handled by servers that it interfaces with.

The Mach_msg system call provides for both asynchronous message passing and request-reply style interactions, which makes it extremely complicated. We shall give only an overview of its semantics.

Transparent multiprocessing – Avoiding issues in BSD.

Protected message passing –

Better than Unix message messaging.

• “extensible” Microkernel

• Multiple levels of operating system

Other O/S's implemented as “applications” • Basis for NeXT O/S, Mac X O/S, OSF/1

2075 Chaitra

Q 1) Why distributed system is preferred over centralized system? Explain the layers of transparency.

Distributed systems have an advantage over centralized systems in terms of network speed, since as the information is not stored in a central location, a bottleneck is less likely, in which the number of people Attempting to access a server is larger than it can support, causing waiting times and slowing.

Advantages of Distributed Systems over Centralized Systems:

Economics: Microprocessors offer a better price/performance than mainframes

Speed: A distributed system may have more total computing power than a mainframe

Inherent distribution: Some applications involve spatially separated machines

Reliability: Computing power can be added in small increments

- Transparency deals with how to achieve the single-system image, i.e how to make a collection of computers appear as a single computer.
- Hiding all the distribution from the users as well as the application programs can be achieved at two levels:
 - hide the distribution from users
 - At a lower level, make the system look transparent to programs.

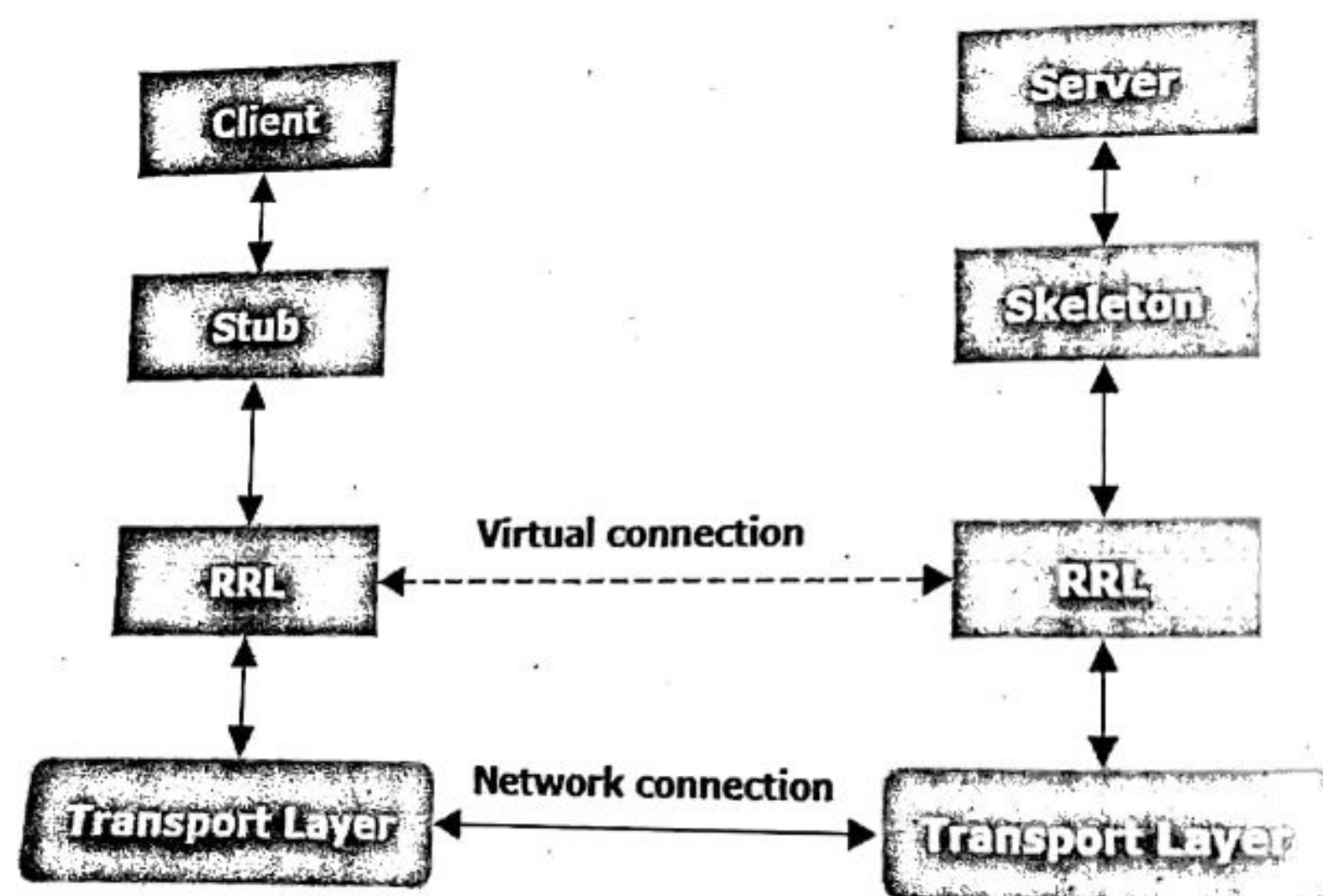
Forms of Transparency in a Distributed System are:

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location or is migrated to newer version
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be shared by several competitive users

Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Q 2) What do you mean by RMI software? Compare discuss RPC and RMI.

RMI (Remote Method Invocation) is a way that a programmer, using the Java programming language and development environment, can write object-oriented programming in which objects on different computers can interact in a distributed network.



Second part,

S.NO	RPC	RMI
1.	RPC is a library and OS dependent platform.	Whereas it is a java platform.
2.	RPC supports procedural programming.	RMI supports object-oriented programming.
3.	RPC is less efficient in comparison of RMI.	While RMI is more efficient than RPC.
4.	RPC creates more overhead.	While it creates less overhead than RPC.
5.	The parameters which are passed in RPC are ordinary or normal data.	While in RMI, objects are passed as parameter.
6.	RPC is the older version of RMI.	While it is the successor version of RPC.
7.	There is high Provision of ease of programming in RPC.	While there is low Provision of ease of programming in RMI.
8.	RPC does not provide any security.	While it provides client level security.
9.	It's development cost is huge.	While it's development cost is fair or reasonable.
10.	There is a huge problem of versioning in RPC.	While there is possible versioning using RMI.
11.	There is multiple codes are needed for simple application in RPC.	While there is multiple codes are not needed for simple application in RMI.

Q 3) Compare stateful and stateless service. Describe the architecture and operation of SUNNFS with its services.

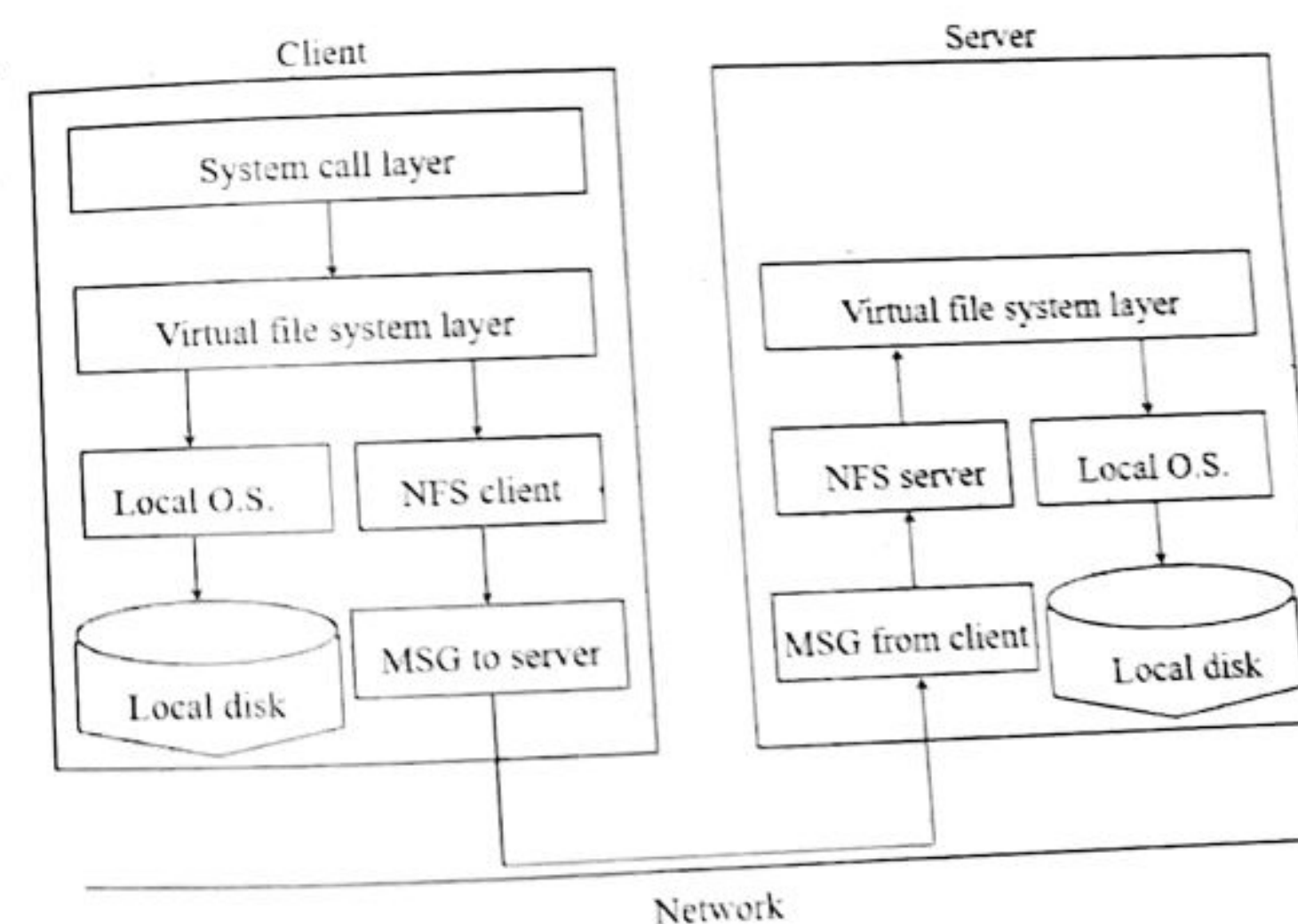
Ans:

Parameters	Stateful	Stateless
1. State	A Stateful server remember client data (state) from one request to the next.	A Stateless server keeps no state information
2. Programming	Stateful server is harder to code	Stateless server is straightforward to code
3. Efficiency	More Because clients do not have to provide full file information every time they perform an operation	Less because information needs to be provided
4. Crash recovery	Difficult due to loss of information	Can easily recover from failure. Because there is no state that must be restored
5. Information transfer	Using a Stateful, file server, the client can send less data with each request	Using a stateless file server, the client must, specify complete file names in each request specify location for reading or writing re-authenticate for each request.
6. Extra services	Stateful servers can also offer clients extra services such as file locking, and remember read and write positions	It does not have to implement the state accounting associated with opening, closing, and locking of files.

Parameters	Stateful	Stateless
7. Operations	Open, Read, Write, Seek, Close	Read, Write

Second part,

The earliest successful distributed system could be attributed to Sun Microsystems, which developed the Network File System (NFS). NFSv2 was the standard protocol followed for many years, designed with the goal of simple and fast server crash recovery. This goal is of utmost importance in multi-client and single server-based network architectures because a single instant of server crash means that all clients are unserved. The entire system goes down.



Architecture:

Basic Design

The NFS design consists of three major pieces: the protocol, the server side and the client side.

NFS Protocol

NFS uses a stateless protocol. The NFS protocol is defined in terms of a set of procedures, their arguments and results, and their effects. Remote procedure calls are synchronous, that is, the client application blocks until the server has completed the call and returned the results. This makes RPC very easy to use and understand because it behaves like a local procedure call.

The Server side

Because the NFS server is stateless, when servicing an NFS request it must commit any modified data to stable storage before returning results. The implication for UNIX based servers is that requests which modify the filesystem must flush all modified data to disk before returning from the call. For example, on a write request, not only the data block, but also any modified indirect blocks and the block containing the inode must be flushed if they have been modified.

The Client Side

The Sun implementation of the client side provides an interface to NFS which is transparent to applications. To make transparent access to remote files work we had to use a method of locating remote files that does not change the structure of path names. Some UNIX based remote file access methods use pathnames like host path or host path to name remote files. This does not allow real transparent access since existing programs that parse pathnames have to be modified.

Q 4) Compare heterogenous and homogenous distributed system. Explain the CORBA architecture and its services.

In a homogenous distributed system, each database is an Oracle database.

Homogeneous systems are much easier to design and manage.

This approach provides incremental growth, making the addition of a new site to the distributed system easy, and allows increased performance by exploiting the parallel processing capability of multiple sites.

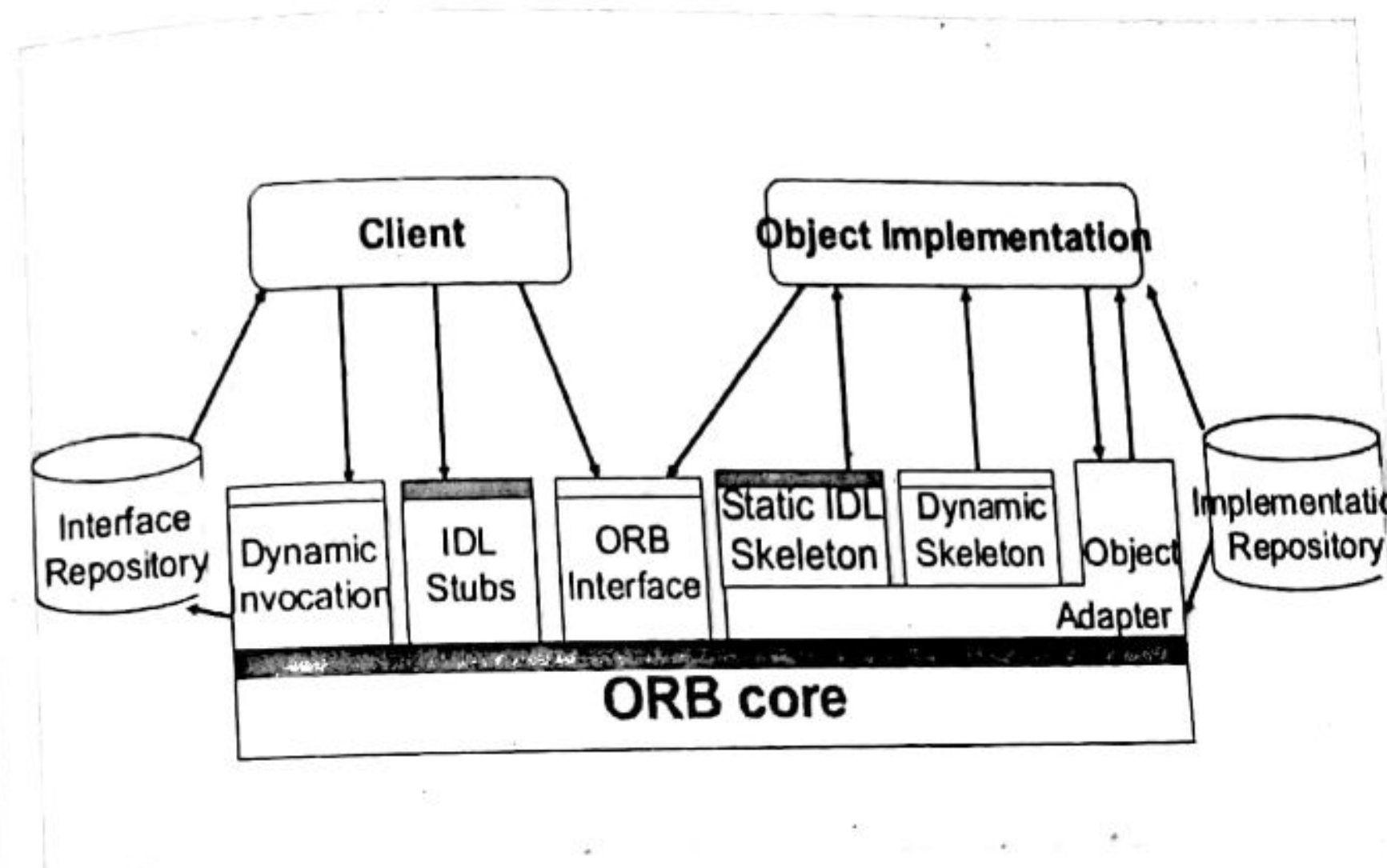
In a heterogeneous distributed system, at least one of the databases is a non-Oracle database

Heterogeneous system usually results when individual sites have implemented their own database and integration is considered at a later stage.

Second part,

CORBA stands for Common Object Request Broker Architecture. It is a standard defined by the Object Management Group(OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

CORBA architecture:



The Basic Architecture of CORBA consists of following components:

Interface Repository:

It provides representation of available object interfaces of all objects. It is used for dynamic invocation. It contains interface definitions for registered CORBA server components. Interfaces are represented in a "metadata" format so that they can be discovered dynamically at run time.

Implementation Repository:

It stores implementation details for each object's interfaces. (Mapping from server object name to filename to implement service) The information stored may be OS specific.

Object Request Broker (ORB) core:

It provides mechanisms by which objects can interact with each other transparently. It carries out the request-reply protocol between client and server. It provides operations that enable process to be started and stopped and operations to convert between remote object references and strings. Its functions are:

- Find the object implementation for the request
- Prepare the object implementation to receive the request
- Communicate the data making up the request
- Retrieve results of request

Static Invocation:

It allows a client to invoke requests on an object whose compile-time knowledge of server's interface specification is known.

Dynamic Invocation:

It allows a client to invoke requests on object without having compile time knowledge of object's interface.

Object Adapter:

It is the interface between server object implementation and ORB. Its function are:

- Bridges the gap between CORBA objects and the programming language interfaces of the servant classes.
- Creates remoter object references for the CORBA objects
- Dispatches each RMI to the appropriate servant class via a skeleton and activates objects.
- Assigns a unique name to itself and each object

Skeletons (server):

An IDL compiler generates skeleton classes in the server's language.

Client Proxies / Stubs:

It is generated by an IDL compiler in the client language. It lives in the client machine and pretends to be remote object.

CORBA SERVICES

Naming service

It allows clients to find and locate objects based on name.

Trading service

It allows clients to find and locate objects based on their properties.

Notification service

It allows objects to notify other objects that some event has occurred.

Transaction Service

It allows atomic transactions and rollback on failures.

Security Service

It protects components from unauthorized access or users.

Concurrency control service

It provides a lock manager that ca7. Life cycle servicen obtain and free locks for transactions to manage concurrent transactions.

Life cycle service

It defines conventions for creating, deleting, copying and moving CORBA objects.

Time service

It provides interfaces for synchronizing time.

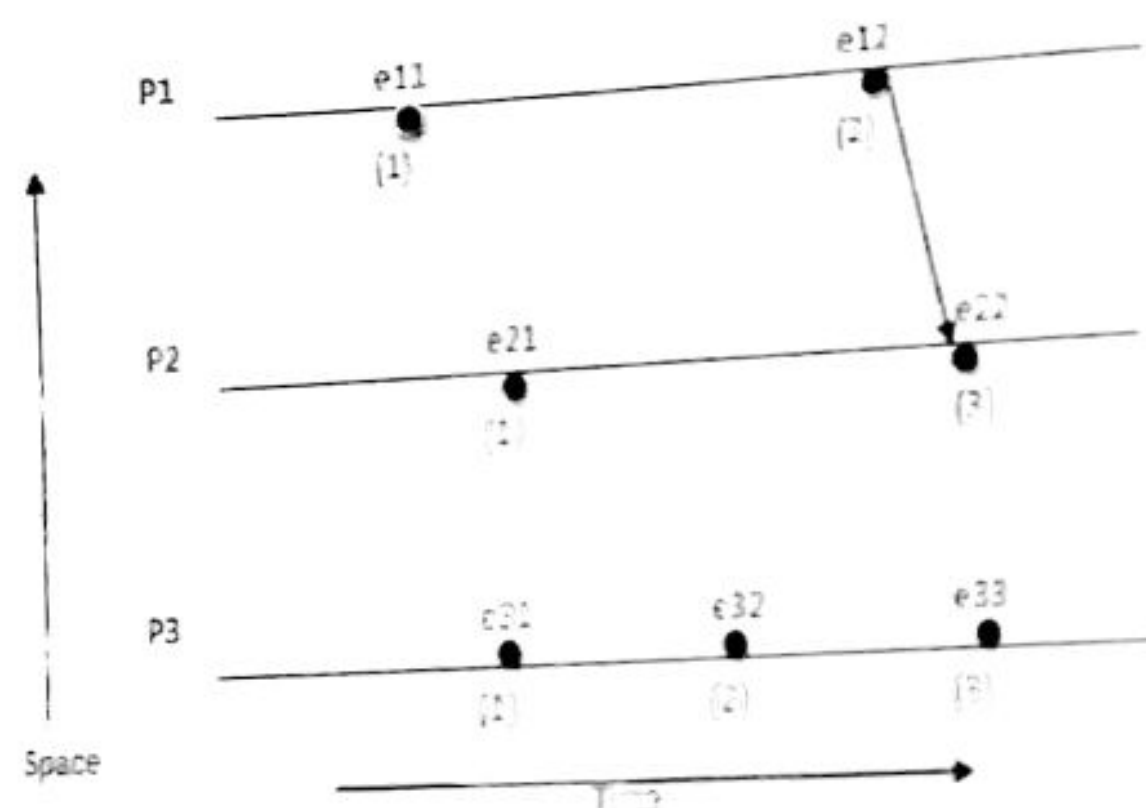
Q 5)List the problems with Lamport clock with example. How the vector clock is beneficial than Lamport's clock? Explain with implementation rules of vector clock.

Lamport's logical clocks lead to a situation where all events in a distributed system are totally ordered. That is, if \rightarrow , then we can say $C(a) < C(b)$. Unfortunately, with Lamport's clocks, nothing can be said about the actual time of a and b.

Let us demonstrate it through the example below:

Limitation:

- In case of [IR1], if $a \rightarrow b$, then $C(a) < C(b) \rightarrow \text{true}$.
- In case of [IR2], if $a \rightarrow b$, then $C(a) < C(b) \rightarrow \text{May be true or may not be true.}$



Vector Clocks represent an extension of Lamport Timestamps in that they guarantee the strong consistency condition which (additionally to the clock consistency condition) dictates that if one event's clock comes before another's, then that event comes before the other, i.e., it is a this-way condition.

Vector Clock is an algorithm that generates partial ordering of events and detects causality violations in a distributed system. These clocks expand on Scalar time to facilitate a causally consistent view of the distributed system, they detect whether a contributed event has occurred or later event in the distributed system.

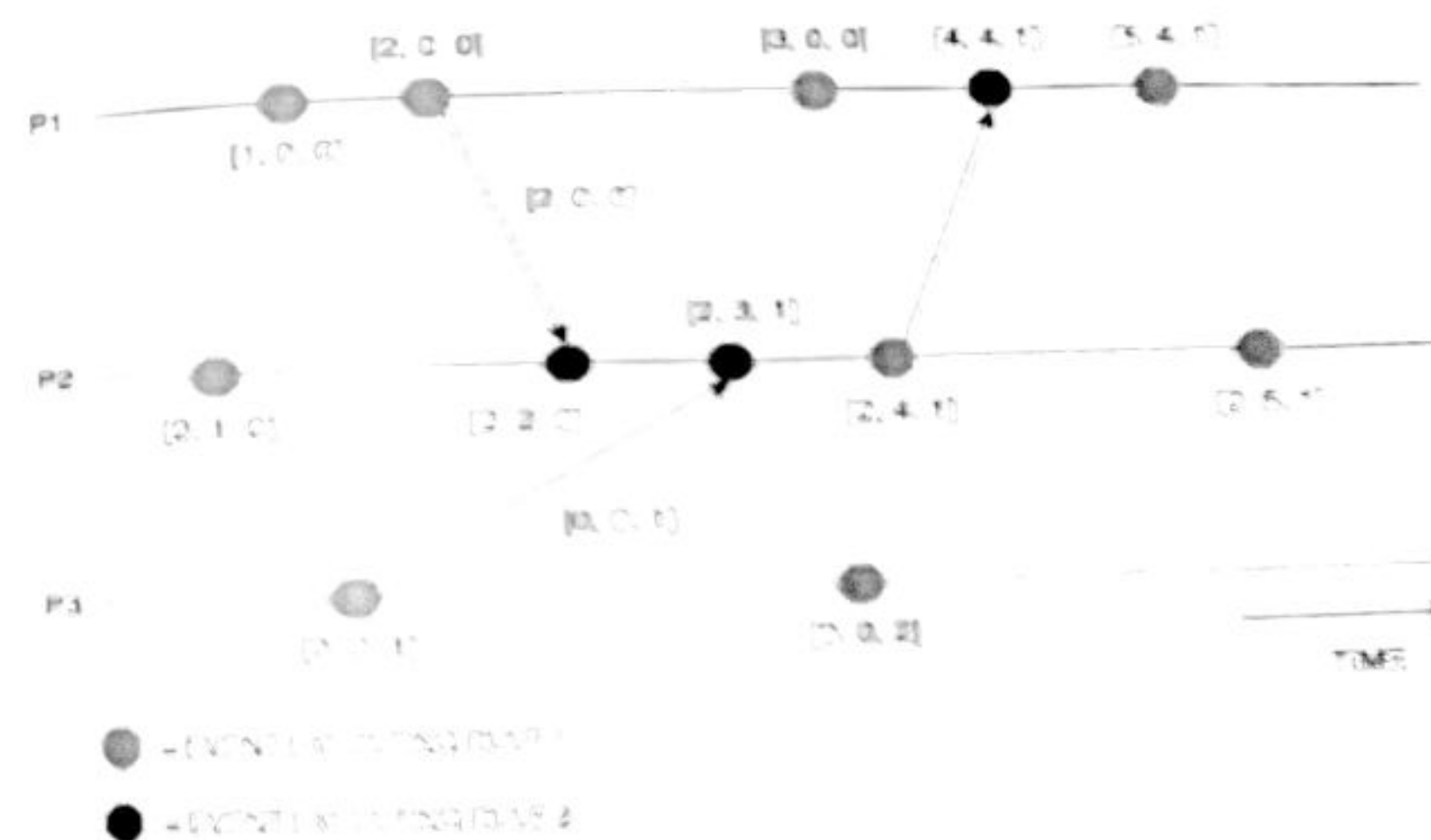
How does the vector clock algorithm work :

- Initially, all the clocks are set to zero
- Every time, an Internal event occurs in a process, the value of the processes' logical clock in the vector is incremented by 1
- Also, every time a process sends a message, the value of the processes' logical clock in the vector is incremented by 1.

Every time, a process receives a message, the value of the processes' logical clock in the vector is incremented by 1, and moreover, each element is updated by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

Example

Consider a process (P) with a vector size N for each process, the above set of rules mentioned are to be executed by the vector clock.



The above example depicts the vector clocks mechanism in which the vector clocks are updated after execution of internal events, the arrows indicate how the values of vectors are sent in between the processes (P1, P2, P3).

Q 6) How the token system works for mutual exclusion in distributed system? Explain with token based algorithm.

Token Based System works for mutual exclusion in distributed system :

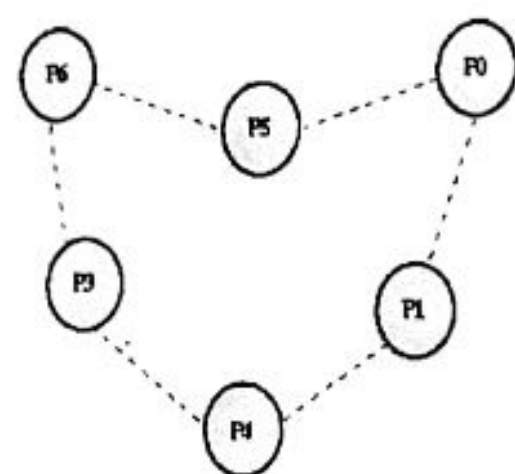
- A unique token is shared among all the sites.
- If a site possesses the unique token, it is allowed to enter its critical section
- This approach uses sequence number to order requests for the critical section.
- Each requests for critical section contains a sequence number. This sequence number is used to distinguish old and current requests.
- This approach insures Mutual exclusion as the token is unique.

Second part,

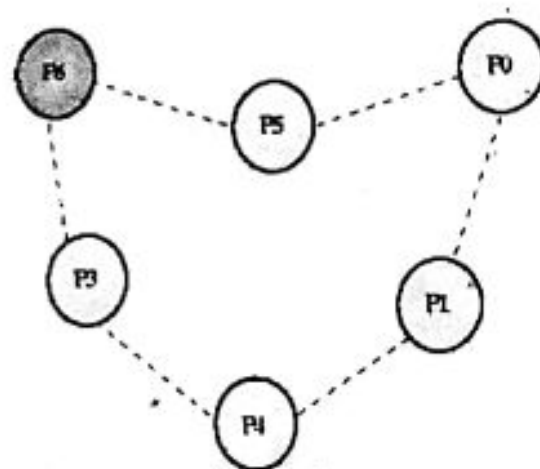
Token ring algorithm

When any process notice that the coordinator is not functioning, it builds an ELECTION MESSAGE containing its own process number and sends the message to its successor.

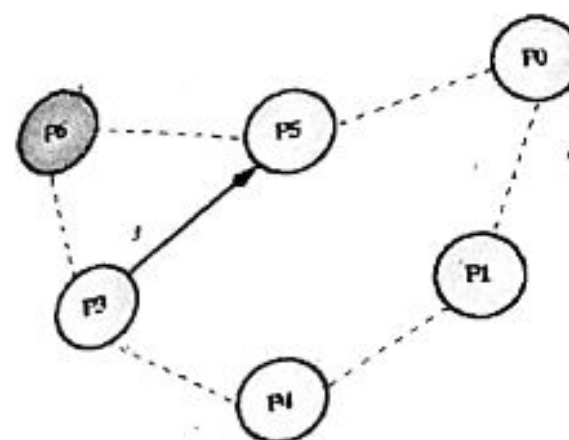
Example:



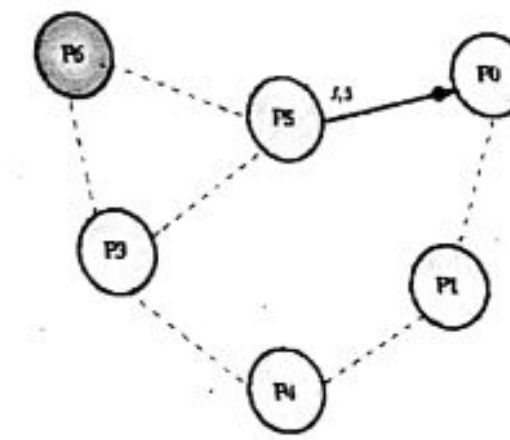
Token Ring Election Algorithm: Step 0



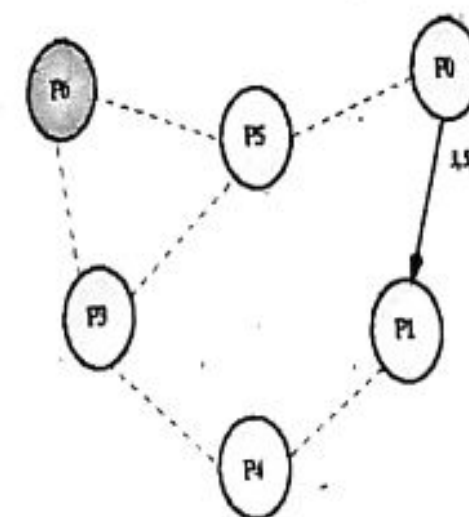
Token Ring Election Algorithm: Step 1



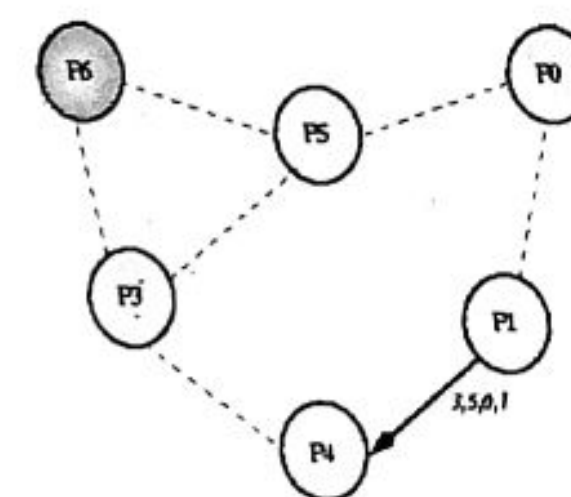
Token Ring Election Algorithm: Step 2



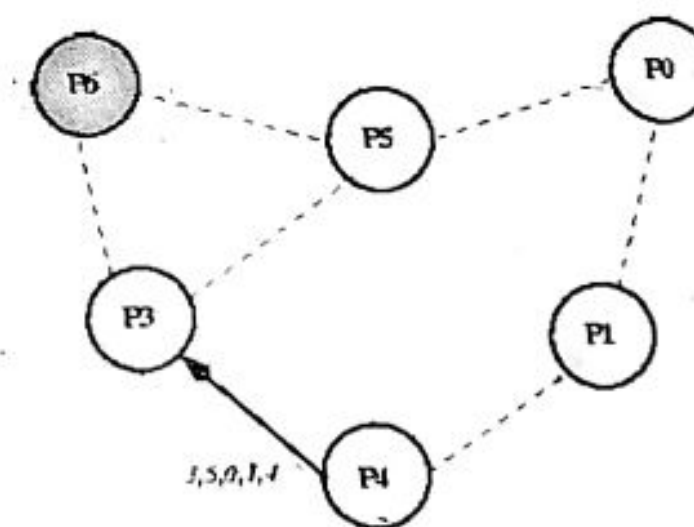
Token Ring Election Algorithm: Step 3



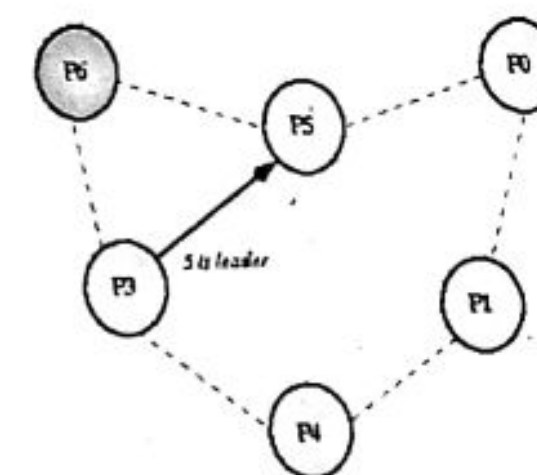
Token Ring Election Algorithm: Step 4



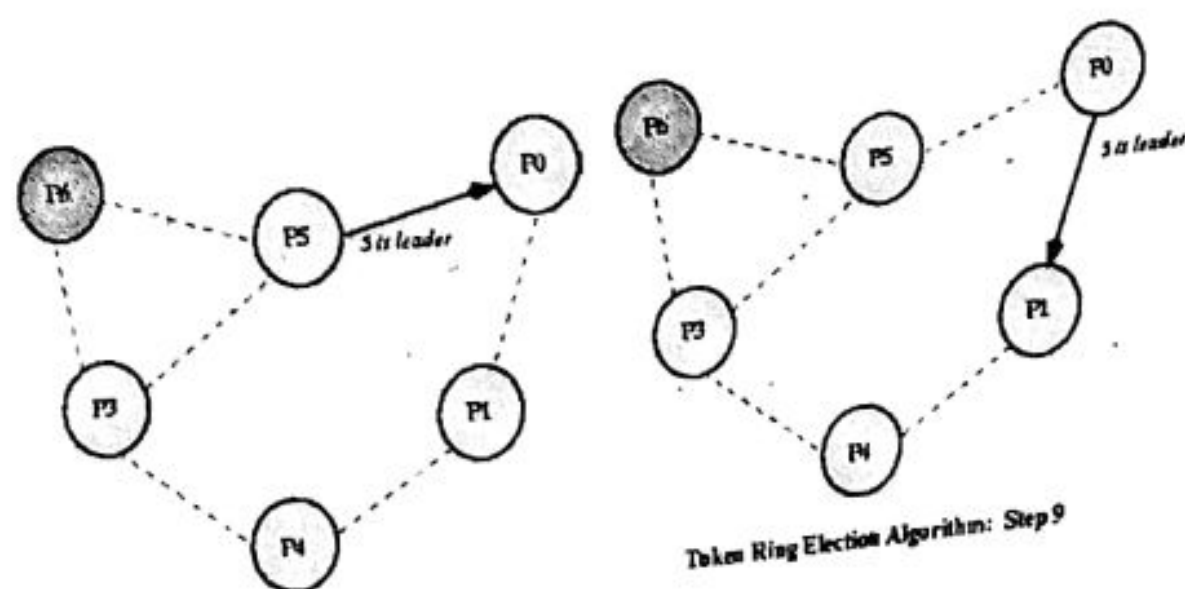
Token Ring Election Algorithm: Step 5



Token Ring Election Algorithm: Step 6



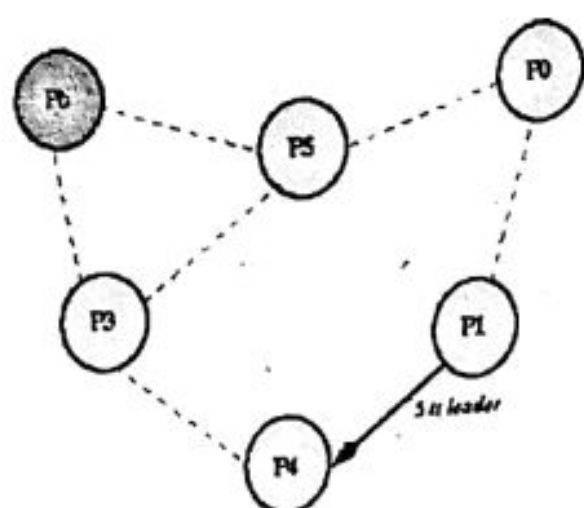
Token Ring Election Algorithm: Step 7



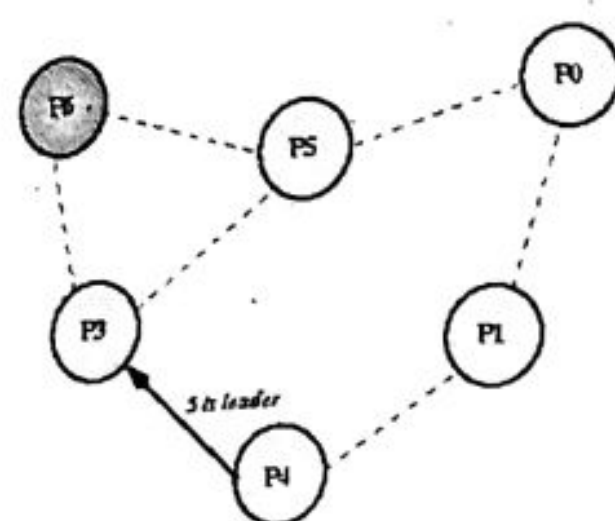
Token Ring Election Algorithm: Step 8

Token Ring Election Algorithm: Step 9

Step 10: Process 1 passes on the coordinator message. Process 4 passes on the coordinator message.
Step 11: Process 3 receives the coordinator message and stops it.



Token Ring Election Algorithm: Step 10



Token Ring Election Algorithm: Step 11

Step 0: We start with 6 processes, connected in a logical ring. Process 6 is the leader, as it has the highest number.

Step 1: Process 6 fails.

Step 2: Process 3 notices that Process 6 does not respond. So it starts an election, sending a message containing its id to the next node in the ring.

Step 3: Process 5 passes the message on, adding its own id to the message.

Step 4: Process 0 passes the message on, adding its own id to the message.

Step 5: Process 1 passes the message on, adding its own id to the message.

Step 6: Process 4 passes the message on, adding its own id to the message.

Step 7: When Process 3 receives the message back, it knows the message has gone around the ring, as its own id is in the list. Picking the highest id in the list, it starts the coordinator message "5 is the leader" around the ring.

Step 8: Process 5 passes on the coordinator message.

Step 9: Process 0 passes on the coordinator message.

Q 7) How replication is used as basic scaling technique in distributed system? Explain the active replication model with its advantages and disadvantages.

Replication in computing involves sharing information so as to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility.

It is used as it has following features:

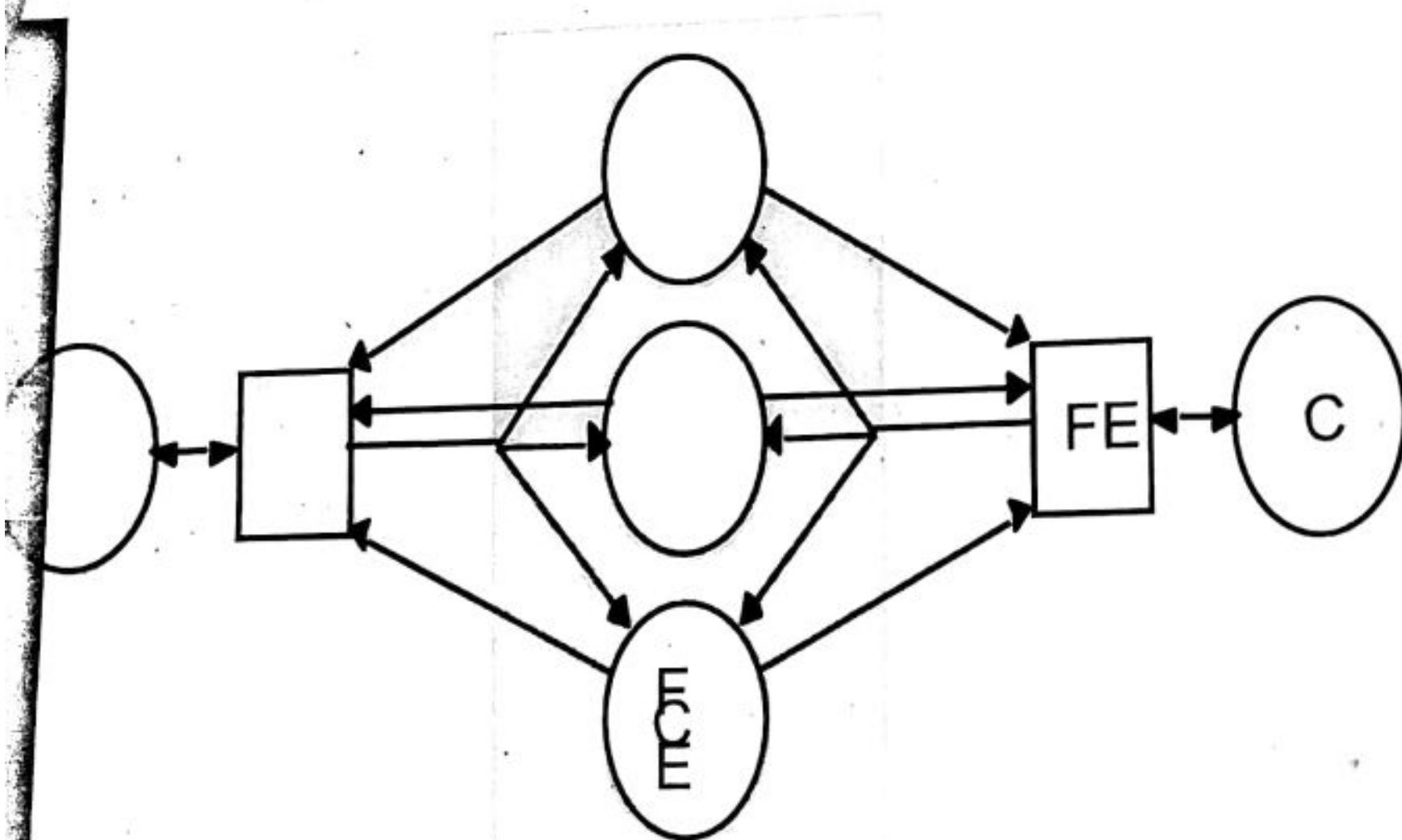
It creases the reliability of a system. If one replica is unavailable or crashes, use another Protect against corrupted data

It improves the performance of a system.

Second part,

Where update volume or other factors such as partitioned networks demand updates to more than one replica at the same time, passive replication may be unsuitable.

- In this case, we are still considering fault tolerance alone without performance enhancement, as this model also has a high overhead.



Active replication steps:

1. Request: front end attaches unique ID to request and multicasts (totally ordered, reliable) to Ms. Front end is assumed to fail only by crashing.
2. Coordination: every correct RM receives request in same total order.
3. Execution: every RM executes the request.
4. Coordination: (not required due to multicast)
5. Response: each RM sends response to front end, which manages responses depending on failure assumptions and multicast algorithm.

Advantage:

Active replication is the preferable choice when dealing with real time systems that require quick response even under the presence of faults or with systems that must handle byzantine faults.

Disadvantage:

The big disadvantage for active replication is that in practice most of the real world servers are non-deterministic.

Q 8) Compare nested and distributed transactions. Explain the two-phase commit protocol of handling distributed transactions.

In nested transactions, each transaction can have sub transactions. For example, the Pay Loan from Checking transaction can have two sub transactions Debit Checking and Pay Loan. Like ordinary "flat" (i.e., non-nested) transactions, sub transactions are bracketed by the Start, Commit, and Abort operations.

A **distributed transaction** includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database.

A simple example of a distributed transaction job includes a single source queue that supplies messages for a single database update. The DoTransaction3Links_IUD and DoTransaction3Links_DUI jobs demonstrate the effect of different ordering of Distributed Transaction input link processing.

Two-phase Commit

Two-phase commit reduces the vulnerability of one-phase commit protocols. The steps performed in the two phases are as follows –

Phase 1: Prepare Phase

- After each slave has locally completed its transaction, it sends a "DONE" message to the controlling site. When the controlling site has received "DONE" message from all slaves, it sends a "Prepare" message to the slaves.
- The slaves vote on whether they still want to commit or not. If a slave wants to commit, it sends a "Ready" message.
- A slave that does not want to commit sends a "Not Ready" message. This may happen when the slave has conflicting concurrent transactions or there is a timeout.

Phase 2: Commit/Abort Phase

- After the controlling site has received "Ready" message from all the slaves –
 - The controlling site sends a "Global Commit" message to the slaves.
 - The slaves apply the transaction and send a "Commit ACK" message to the controlling site.
 - When the controlling site receives "Commit ACK" message from all the slaves, it considers the transaction as committed.
- After the controlling site has received the first "Not Ready" message from any slave –
 - The controlling site sends a "Global Abort" message to the slaves.
 - The slaves abort the transaction and send a "Abort ACK" message to the controlling site.
 - When the controlling site receives "Abort ACK" message from all the slaves, it considers the transaction as aborted.

Q 9) what do you learn from Byzantine general problems? Explain the basic principle of k-fault tolerant.

The term takes its name from an allegory, the "Byzantine generals problem", developed to describe a situation in which, in order to avoid catastrophic failure of the system, the system's actors must agree on a concerted strategy, but some of these actors are unreliable.

The Byzantine Generals Problem is a game theory problem, which describes the difficulty decentralized parties have in arriving at consensus without relying on a trusted central party. In a network where no member can verify the identity of other members, how can members collectively agree on a certain truth?

A system is said to be k-fault tolerant if it can withstand k faults.

k fault tolerant A system is k fault tolerant if it can have k faulty components and still meet its specifications.

Silent Failure k + 1 components will provide k coverage, because only one component needs to be correct.

Byzantine Failure 2k + 1 components are needed, so that the k + 1 non-faulty components can outvote the k faulty components. It is unlikely that k components will fail, but k + 1 will not fail. A large-scale failure is not always predictable.

Four constituent principle of the k- fault tolerance approach have been identified:

- (i) error detection.
- (ii) damage assessment.
- (iii) error recovery.
- (iv) fault treatment and continued system service.

10) write short notes:

a) Reliable Group Communication:

- When a group is static and processes do not fail
- Reliable communication = deliver the message to all group members
 - Any order delivery
 - Ordered delivery

Communication between two processes in a distributed system is required to exchange various data, such as code or a file, between the processes. When one source process tries to communicate with multiple processes at once, it is called **Group Communication**.

The reliable group communication guarantees a form of atomicity in that the messages are received by all operational members of the group or by none of them. Since the overhead in enforcing the order of messages is nontrivial, the mechanism provides two types of message transmission: one guarantees delivery of the messages in the same order to all members of a group, and the other guarantees only atomicity with messages delivered in some arbitrary order. The message-ordering property can be used to simplify distributed database and distributed processing algorithms. The mechanism can survive despite process, host, and communication failures.

b) Distributed deadlock:

Distributed deadlocks can occur when distributed transactions or concurrency control are utilized in distributed systems. It may be identified via a distributed technique like edge chasing or by creating a global wait-for graph (WFG) from local wait-for graphs at a deadlock detector.

Approaches to detect deadlock in the distributed system

Various approaches to detect the deadlock in the distributed system are as follows:

1. Centralized Approach

Only one resource is responsible for detecting deadlock in the centralized method, and it is simple and easy to use. Still, the disadvantages include excessive workload on a single node and single-point failure (i.e., the entire system is dependent on one node, and if that node fails, the entire system crashes), making the system less reliable.

2. Hierarchical Approach

In a distributed system, it is the integration of both centralized and distributed approaches to deadlock detection. In this strategy, a single node handles a set of selected nodes or clusters of nodes that are in charge of deadlock detection.

3. Distributed Approach

In the distributed technique, various nodes work to detect deadlocks. There is no single point of failure as the workload is equally spread among all nodes. It also helps to increase the speed of deadlock detection.

Deadlock Handling Strategies

Various deadlock handling strategies in the distributed system are as follows:

1. There are mainly three approaches to handling deadlocks: deadlock prevention, deadlock avoidance, and deadlock detection.

2. Handling deadlock becomes more complex in distributed systems since no site has complete knowledge of the system's present state and every inter-site communication entails a limited and unpredictable latency.
3. The operating system uses the deadlock Avoidance method to determine whether the system is in a safe or unsafe state. The process must inform the operating system of the maximum number of resources, and a process may request to complete its execution.
4. Deadlocks preventions are commonly accomplished by implementing a process to acquire all of the essential resources at the same time before starting execution or by preempting a process that already has the resource.
5. In distributed systems, this method is highly inefficient and impractical.

c) Forward and backward recovery in distributed system:

Forward recovery - the continuation of the currently execute process from some further point with compensation for the corrupted and missed data. The assumptions: The precise error conditions that caused the detection and the resulting damage can be accurately assessed. The errors in the process (system) state can be removed. The process (system) can move forward. Example: exception handling and recovery

Backward recovery - the current process is rolled back to a certain, error-free, point and re-executes the corrupted part of the process thus continuing the same requested service.

The assumptions:

The nature of faults cannot be foreseen and errors in the process (system) state cannot be removed without re-executing.

The process (system) state can be restored to a previous error-free state of the process (system). terminology:

Checkpointing: periodically saving state of system

Logging: saving changes made to system state

Recovery point: the point to which we recover in case of

Difference between(extra)

2075 Ashwin

Q1) Define distributed system? Explain Transparency Properties of Distributed System.

Ans: A distributed system contains multiple nodes that are physically separate but linked together using the network. All the nodes in this system communicate with each other and handle processes in tandem. Each of these nodes contains a small part of the distributed operating system software.

Types of Distributed Systems

- Client/Server Systems
- Peer to Peer Systems

Some advantages of Distributed Systems are as follows –

- All the nodes in the distributed system are connected to each other. So nodes can easily share data with other nodes.
- More nodes can easily be added to the distributed system i.e. it can be scaled as required.
- Failure of one node does not lead to the failure of the entire distributed system. Other nodes can still communicate with each other.
- Resources like printers can be shared with multiple nodes rather than being restricted to just one.

Some disadvantages of Distributed Systems are as follows –

- It is difficult to provide adequate security in distributed systems because the nodes as well as the connections need to be secured.
- Some messages and data can be lost in the network while moving from one node to another.
- The database connected to the distributed systems is quite complicated and difficult to handle as compared to a single user system.
- Overloading may occur in the network if all the nodes of the distributed system try to send data at once.

Second part

Transparency “is the concealment from the user of the separation of components of a distributed system so that the system is perceived as a whole”. Transparency in distributed systems is applied at several aspects such as:

- Access Transparency – Local and Remote access to the resources should be done with same efforts and operations. It enables local and remote objects to be accessed using identical operations.
- Location transparency – User should not be aware of the location of resources. Wherever is the location of resource it should be made available to him as and when required.
- Migration transparency – It is the ability to move resources without changing their names.
- Replication Transparency – In distributed systems to achieve fault tolerance, replicas of resources are maintained. The Replication transparency ensures that users cannot tell how many copies exist.
- Concurrency Transparency – As in distributed system multiple users work concurrently, the resource sharing should happen automatically without the awareness of concurrent execution by multiple users.
- Failure Transparency – Users should be concealed from partial failures. The system should cope up with partial failures without the users awareness.
- Parallelism transparency - Activities can happen in parallel without users knowing.

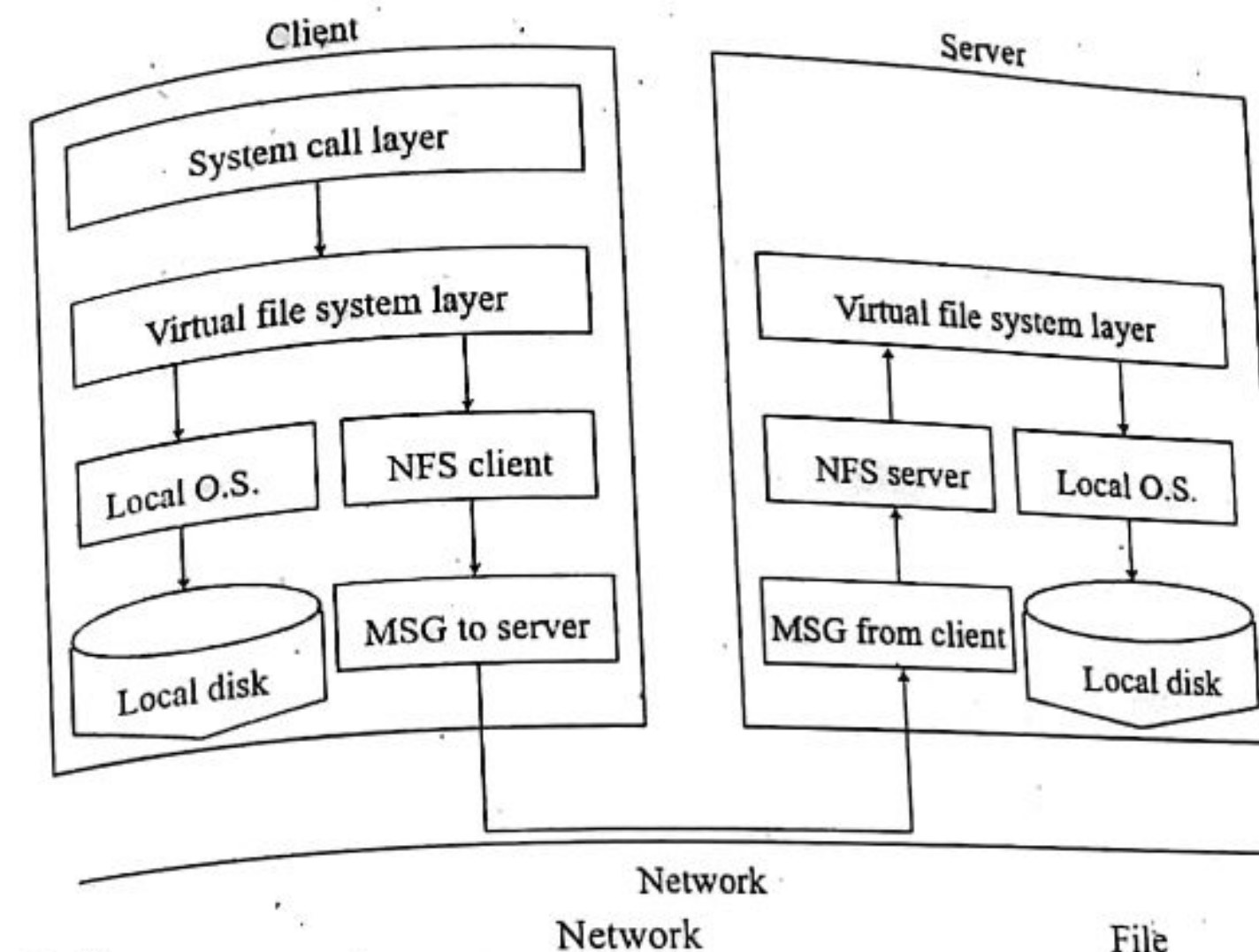
Q 2) Why naming is necessary in distributed system? Explain SUN NFS architecture with its features.

Ans: Names play a very important role in all computer systems. They are used to share resources, to uniquely identify entities, to refer to locations, and more. An important issue with naming is that a name can be resolved to the entity it refers to.

Naming services store information in a central place, which enables users, machines, and applications to communicate across the network.

Second part:

SUN NFS



Sun's
The earliest successful distributed system could be attributed to Sun Microsystems, which developed the Network File System (NFS). NFSv2 was the standard protocol followed for many years, designed with the goal of simple and fast server crash recovery. This goal is of utmost importance in multi-client and single server-based network architectures because a single instant of server crash means that all clients are unserved. The entire system goes down.

Stateful protocols make things complicated when it comes to crashes. Consider a client A trying to access some data from the server. However, just after the first read, the server crashed. Now, when the server is up and running, client A issues the second read request. However, the server does not know which file the client is referring to, since all that information was temporary and lost during the crash. Stateless protocols come to our rescue. Such protocols are designed so as to not store any state information in the server. The server is unaware of what the clients are doing — what blocks they are caching, which files are opened by them and where their current file pointers are. The server simply delivers all the information that is required to service a client request. If a server crash happens, the client would simply have to retry the request. Because of their simplicity, NFS implements a stateless protocol.

File Handles:

NFS uses file handles to uniquely identify a file or a directory that the current operation is being performed upon. This consists of the following components:

- Volume Identifier – An NFS server may have multiple file systems or partitions. The volume identifier tells the server which file system is being referred to.
- Inode Number – This number identifies the file within the partition.
- Generation Number – This number is used while reusing an inode number.

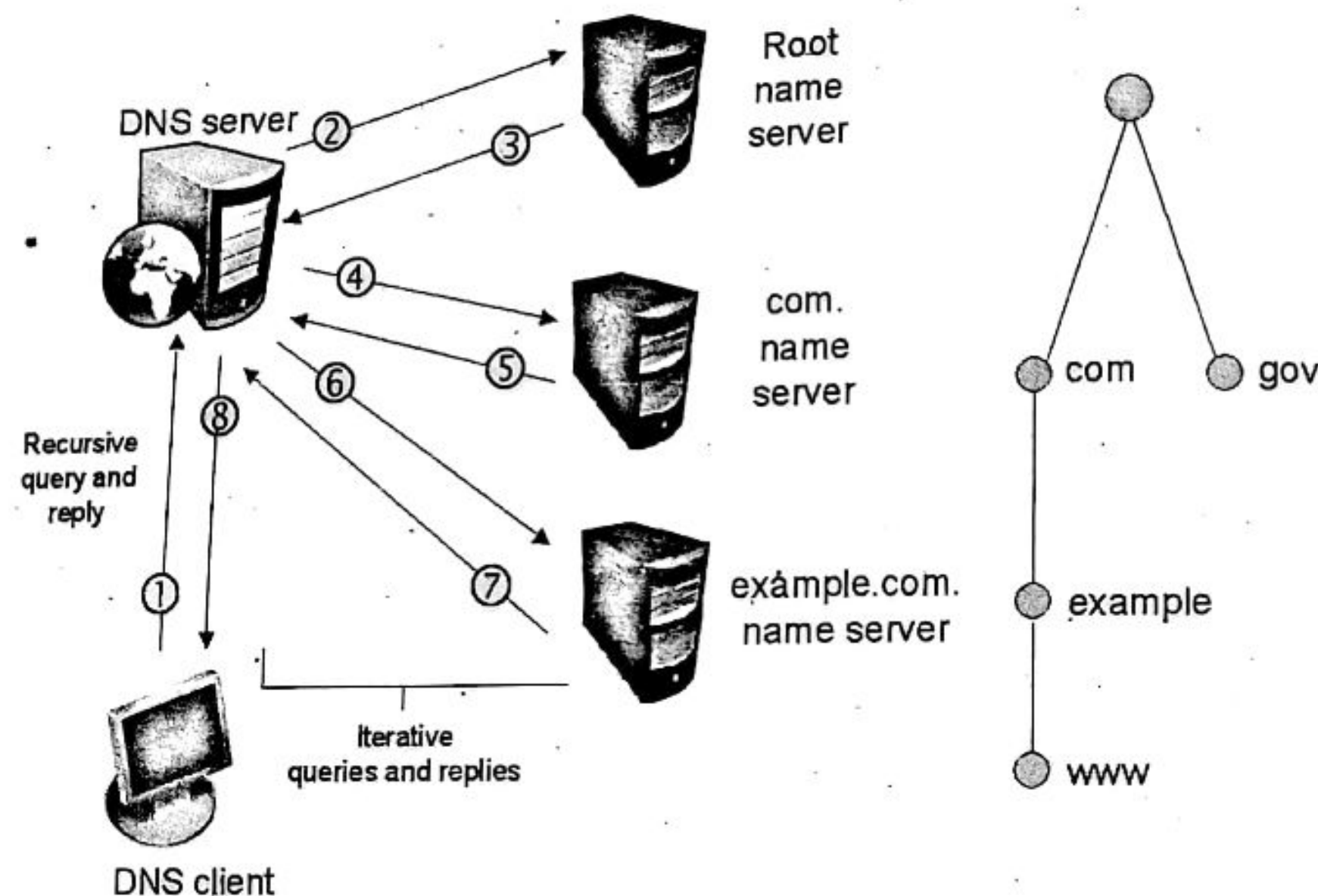
File Attributes:

"File attributes" is a term commonly used in NFS terminology. This is a collective term for the tracked

metadata of a file, including file creation time, last modified, size, ownership permissions etc. This can be accessed by calling `stat()` on the file.

Q3) What is DNS? Explain the DNS working mechanisms with suitable example.

Ans; The Domain Name System (DNS) is the phonebook of the Internet. Humans access information online through domain names, like `nytimes.com` or `espn.com`. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses so browsers can load Internet resources. Each device connected to the Internet has a unique IP address which other machines use to find the device.



The 8 steps in a DNS lookup:

1. A user types 'example.com' into a web browser and the query travels into the Internet and is received by a DNS recursive resolver.
2. The resolver then queries a DNS root nameserver (.).
3. The root server then responds to the resolver with the address of a Top-Level Domain (TLD) DNS server (such as .com or .net), which stores the information for its domains. When searching for example.com, our request is pointed toward the .com TLD.
4. The resolver then makes a request to the .com TLD.
5. The TLD server then responds with the IP address of the domain's nameserver, example.com.
6. Lastly, the recursive resolver sends a query to the domain's nameserver.
7. The IP address for example.com is then returned to the resolver from the nameserver.
8. The DNS resolver then responds to the web browser with the IP address of the domain requested initially.

Once the 8 steps of the DNS lookup have returned the IP address for example.com, the browser is able to make the request for the web page:

9. The browser makes a HTTP request to the IP address.
10. The server at that IP returns the webpage to be rendered in the browser (step 10).

Q 4) What do you mean by DOS (Distributed Operating System)? Briefly explain the Monolithic and microkernel architecture of operation system.

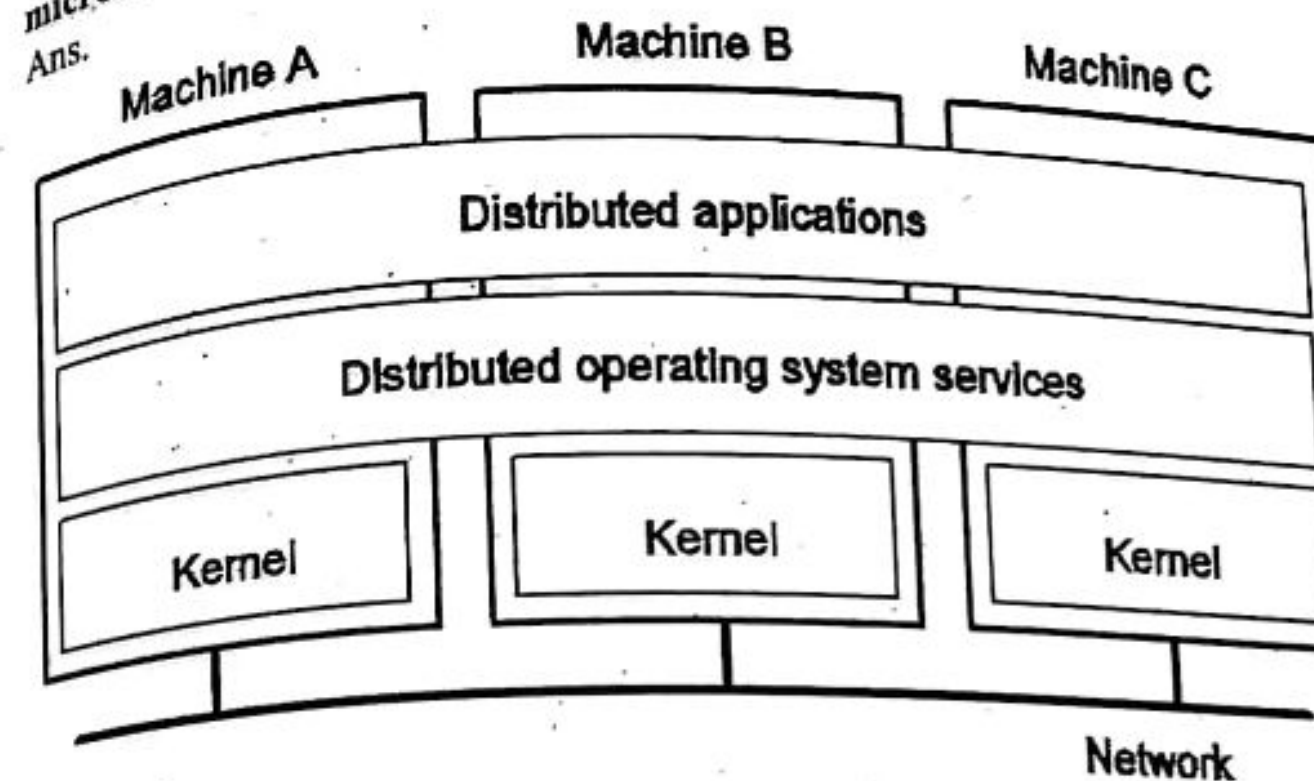


Fig: Distributed operating system

A distributed operating system (DOS) is an essential type of operating system. Distributed systems use many central processors to serve multiple real-time applications and users. As a result, data processing jobs are distributed between the processors.

It connects multiple computers via a single communication channel. Furthermore, each of these systems has its own processor and memory. Additionally, these CPUs communicate via high-speed buses or telephone lines. Individual systems that communicate via a single channel are regarded as a single entity. They're also known as loosely coupled systems.

Second part:

Monolithic kernel

A monolithic kernel is an operating system architecture where the entire operating system is working in kernel space. The monolithic model differs from other operating system architectures, such as the microkernel architecture, in that it alone defines a high-level virtual interface over computer hardware.

A set of primitives or system calls implement all operating system services such as process management, concurrency, and memory management. Device drivers can be added to the kernel as modules.

Here are the following advantages of a monolithic kernel, such as:

- The execution of the monolithic kernel is quite fast as the services such as memory management, file management, process scheduling, etc., are implemented under the same address space.
- A process runs completely in single address space in the monolithic kernel.

The monolithic kernel is a static single binary file.

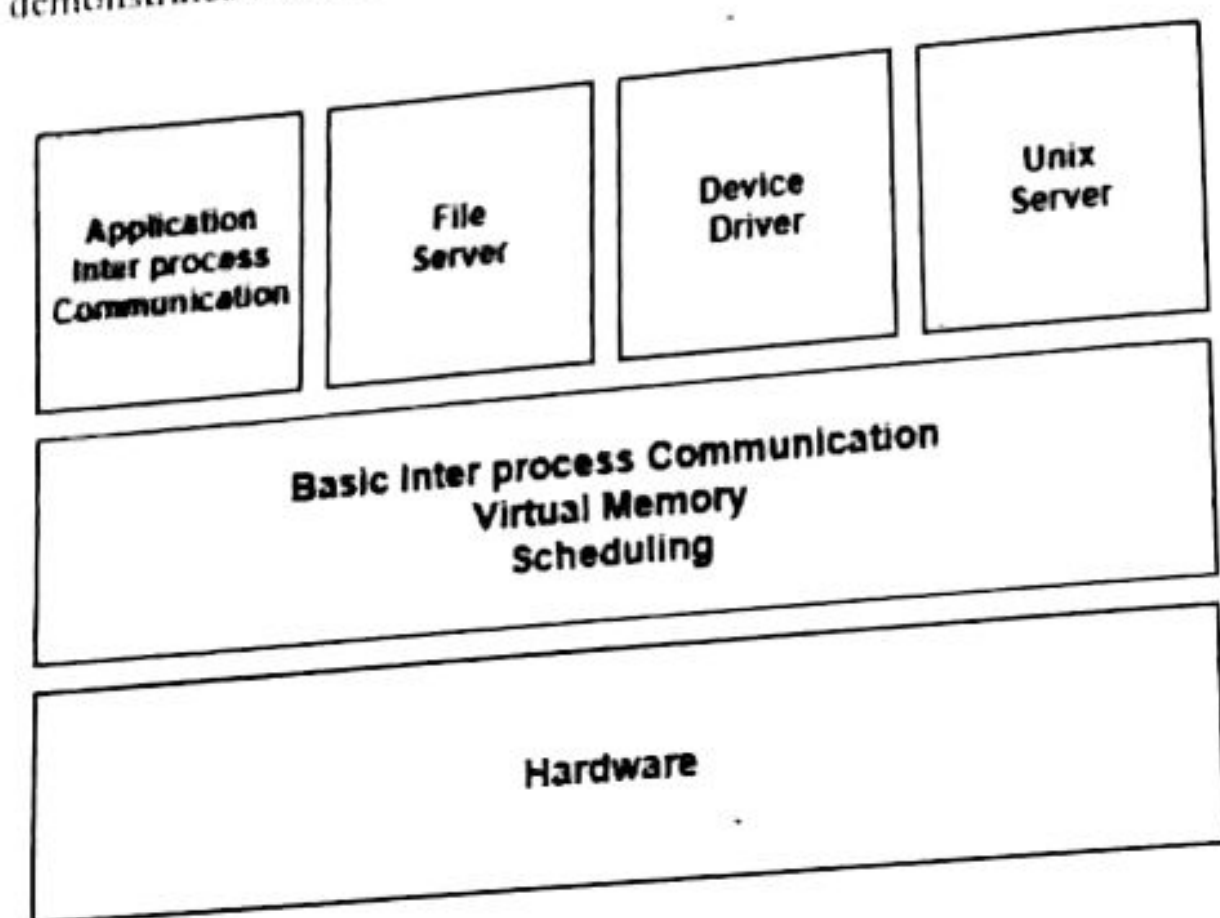
Here are some disadvantages of the monolithic kernel, such as:

- If any service fails in the monolithic kernel, it leads to the failure of the entire system.
- The entire operating system needs to be modified by the user to add any new service.

Microkernel

A microkernel is the minimum software that is required to correctly implement an operating system. It includes memory, process scheduling mechanisms and basic inter-process communication.

A diagram that demonstrates the architecture of a microkernel is as follows –



Microkernel Based Operating System

In the above diagram, the microkernel contains basic requirements such as memory, process scheduling mechanisms and basic interprocess communication. The only software executing at the privileged level i.e. kernel mode is the microkernel. The other functions of the operating system are removed from the kernel mode and run in the user mode. These functions may be device drivers, file servers, application interprocess communication etc.

The microkernel makes sure that the code can be easily managed because the services are divided in the user space. This means that there is less code running in the kernel mode which results in increased security and stability.

Essential Components in a Microkernel

A microkernel contains only the core functionalities of the system. A component is included in the microkernel only if putting it outside would disrupt the functionality of the system. All the other non-essential components are put in the user mode.

The minimum functionalities included in the microkernel are –

- Memory management mechanisms like address spaces are included in the microkernel. This also contains memory protection features.
- Processor scheduling mechanisms are also necessary in the microkernel. This contains process and thread schedulers.
- Interprocess communication is important as it is needed to manage the servers that run their own address spaces.

Performance of a Microkernel System

Providing services in a microkernel system are much more expensive than in a normal monolithic system. The service is obtained by sending an interprocess communication message to the server and getting one in return. This means a context switch or a function call if the drivers are implemented as processes or procedures respectively.

So performance can be complicated in microkernel systems and may lead to some problems. However, this issue is reducing in the modern microkernel systems created such as L4 microkernel systems.

Benefits of Microkernels

Some of the benefits of microkernels are –

- Microkernels are modular and the different modules can be replaced, reloaded, modified, changed etc. as required. This can be done without even touching the kernel.
- Microkernels are quite secure as only those components are included that would disrupt the functionality of the system otherwise.
- Microkernels contain fewer system crashes as compared to monolithic systems. Also, the crashes that do occur can be handled quite easily due to the modular structure of microkernels.

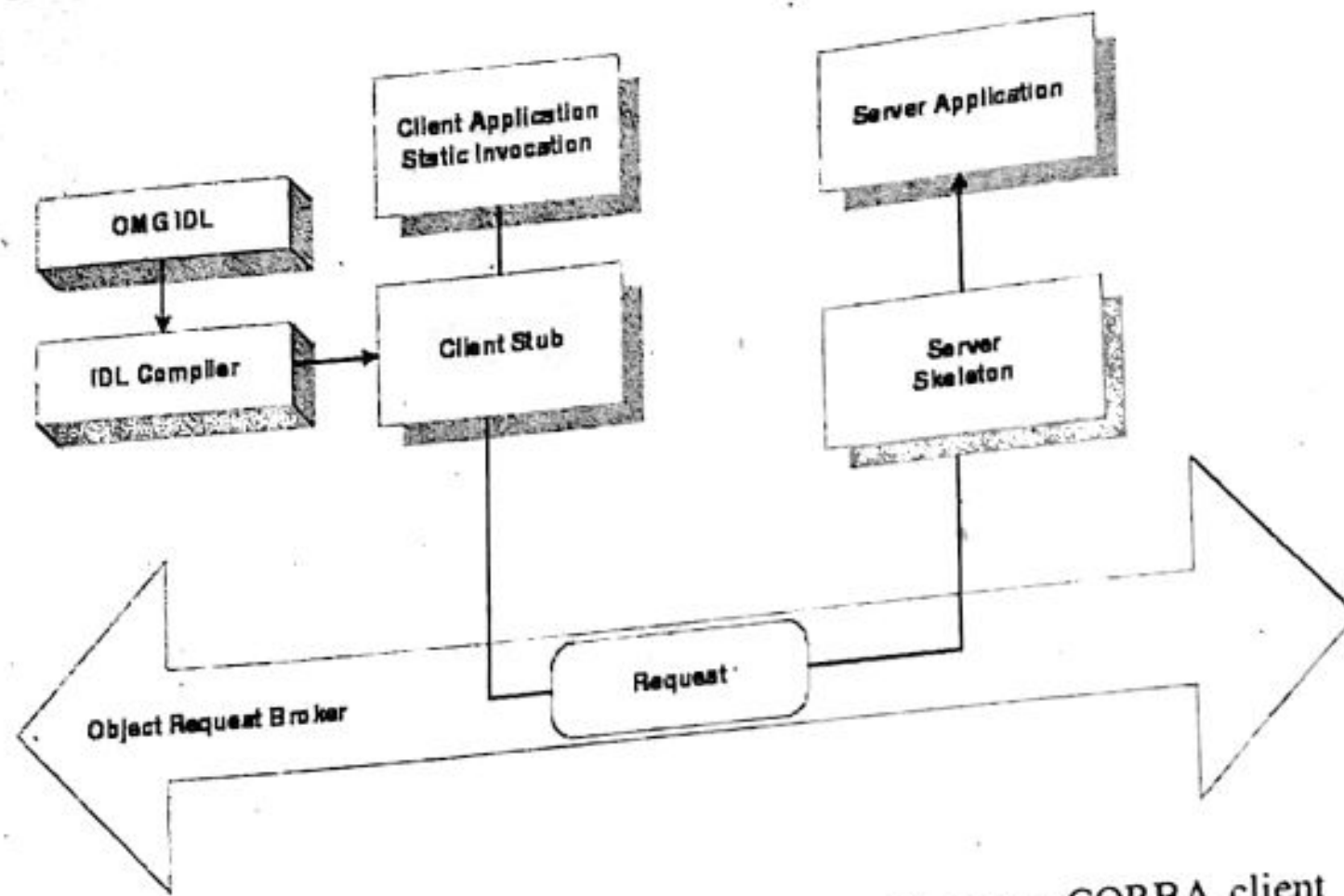
Q 5) Define Object Adapter. Explain the invocation methods in CORBA.

Ans: An object adapter is the primary way for an object to access ORB services such as object reference generation. A portable object adapter exports standard interfaces to the object. The main responsibilities of an object adapter are Generation and interpretation of object references.

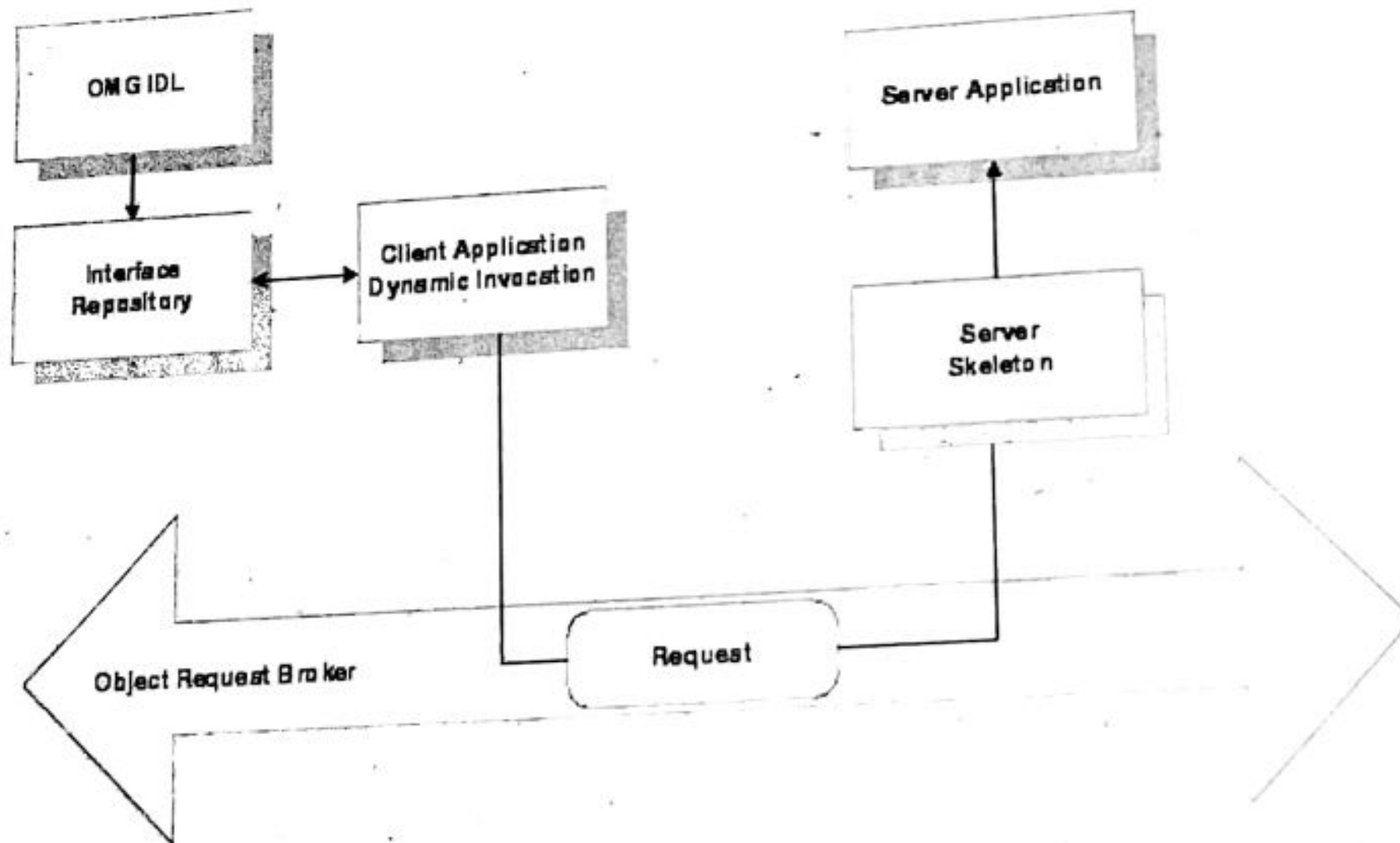
It is often used to make existing classes work with others without modifying their source code. An example is an adapter that converts the interface of a Document Object Model of an XML document into a tree structure that can be displayed.

Second part:

When using static invocation, the CORBA client application invokes operations directly on the client stubs. Static invocation is the easiest, most common type of invocation. The stubs are generated by the IDL compiler. Static invocation is recommended for applications that know at compile time the particulars of the operations they need to invoke and can process within the synchronous nature of the invocation.



While dynamic invocation is more complicated, it enables your CORBA client application to invoke operations on any CORBA object without having to know the CORBA object's interfaces at compile time.



When using dynamic invocation, the CORBA client application can dynamically build operation requests for a CORBA object interface that has been stored in the Interface Repository. CORBA server applications do not require any special design to be able to receive and handle dynamic invocation requests. Dynamic invocation is generally used when the CORBA client application requires deferred synchronous communication, or by dynamic client applications when the nature of the interaction is undefined.

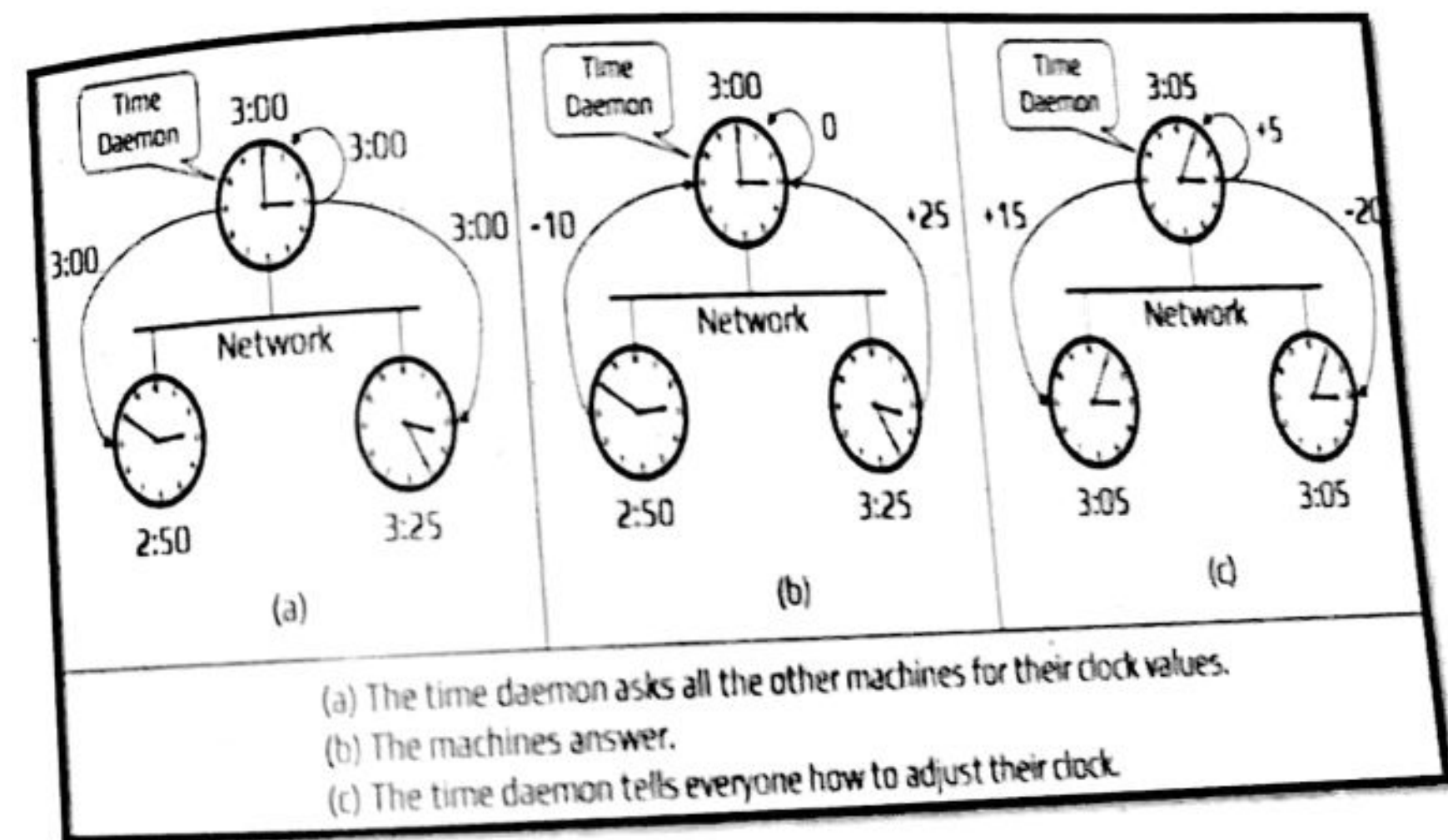
Q 6) What is NTP (Network Time Protocol)? How Berkeley minimizes the problems of single time failures of Cristian's algorithm?

Ans: Network Time Protocol (NTP) is a protocol that helps the computers clock times to be synchronized in a network. This protocol is an application protocol that is responsible for the synchronization of hosts on a TCP/IP network. NTP was developed by David Mills in 1981 at the University of Delaware. This is required in a communication mechanism so that a seamless connection is present between the computers.

Features of NTP:

- NTP servers have access to highly precise atomic clocks and GPU clocks
- It uses Coordinated Universal Time (UTC) to synchronize CPU clock time.
- Avoids even having a fraction of vulnerabilities in information exchange communication.
- Provides consistent timekeeping for file servers

Second part:



Berkeley's Algorithm is a clock synchronization technique used in distributed systems. The algorithm assumes that each machine node in the network either doesn't have an accurate time source or doesn't possess a UTC server.

Algorithm

- 1) An individual node is chosen as the master node from a pool node in the network. This node is the main node in the network which acts as a master and the rest of the nodes act as slaves. The master node is chosen using an election process leader election algorithm.
- 2) Master node periodically pings slaves nodes and fetches clock time at them using Cristian's algorithm.

3) Master node calculates the average time difference between all the clock times received and the clock time given by the master's system clock itself. This average time difference is added to the current time at the master's system clock and broadcasted over the network.

Q 7) What is the need of an election algorithm? Explain non token based Ricart- Agrawala mutual exclusion algorithm along with an example.

Ans:

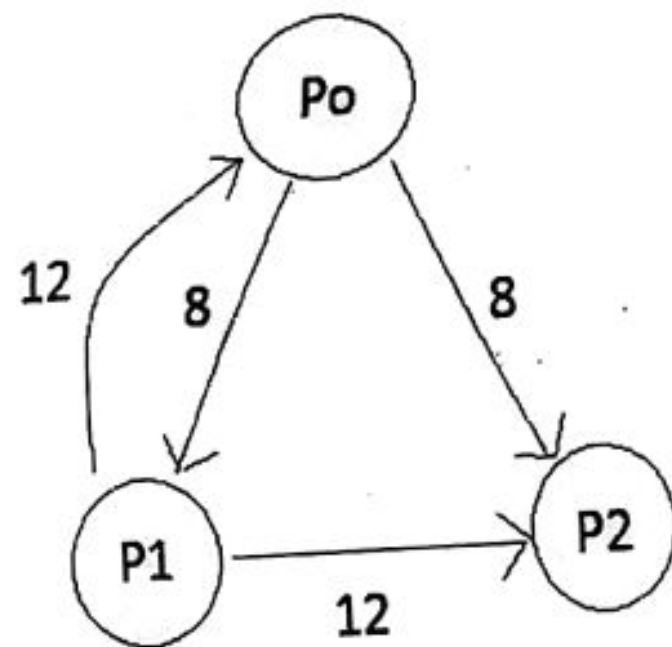
Needs of an election algorithm:

- To decide which process becomes the coordinator process from among the currently running
- Election algorithms are meant for electing a coordinator process from among the currently running processes in such a manner that at any instance of time there is a single coordinator for all processes in the system.

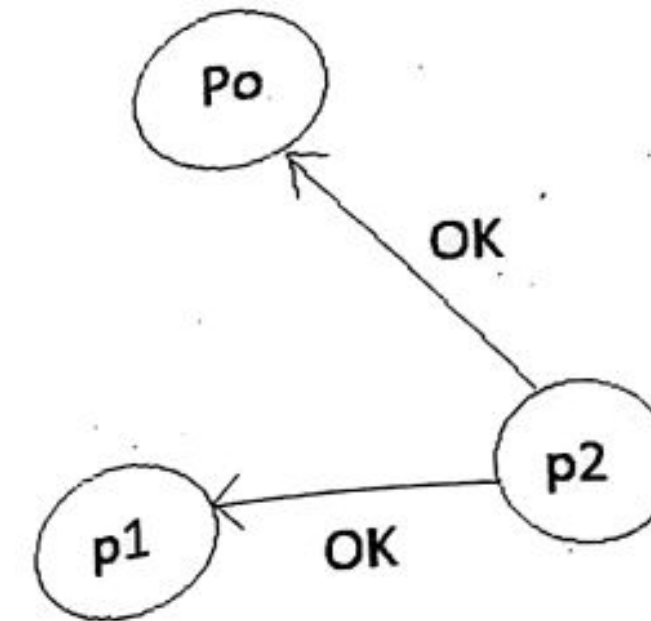
Second part:

Ricart - Agarwala's a non token based algorithm that uses broadcast technique for mutual calculation.

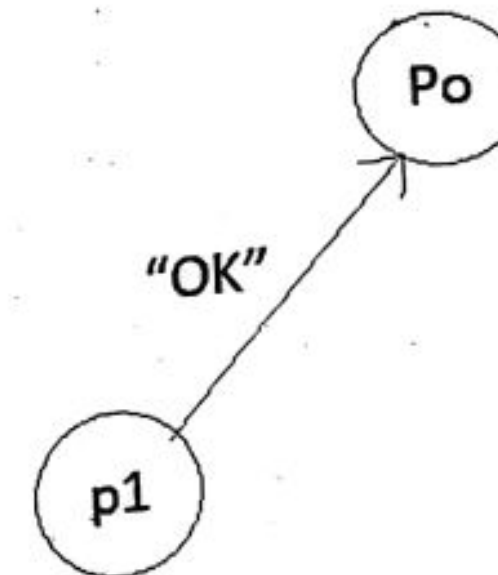
Consider a scenario when process P0 & P1 want to enter critical section. Both P0 & P1 send broadcast to all 3 processes in the system along with time stamp.



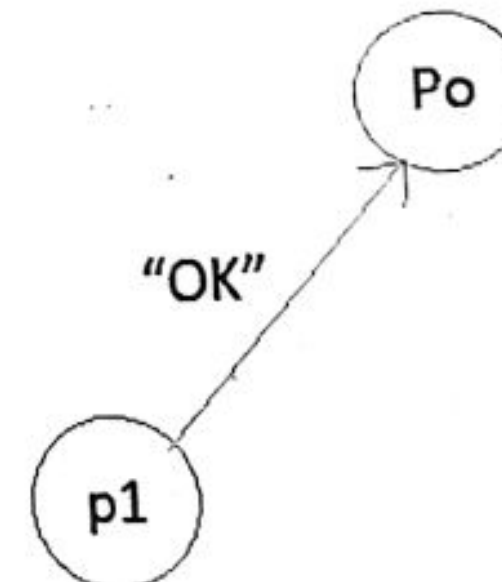
Since P2 does not want to enter critical section hence it sends "OK" to both the processes.



Since P1 finds Ts of P0 to be lesser than Ts of self therefore P1 sends to P0.



Now P0 can enter critical section, as it is received (n-1) OK messages. Once P0 is done with critical section, it sends "Ok" message to P1.



After completion of CS by P0

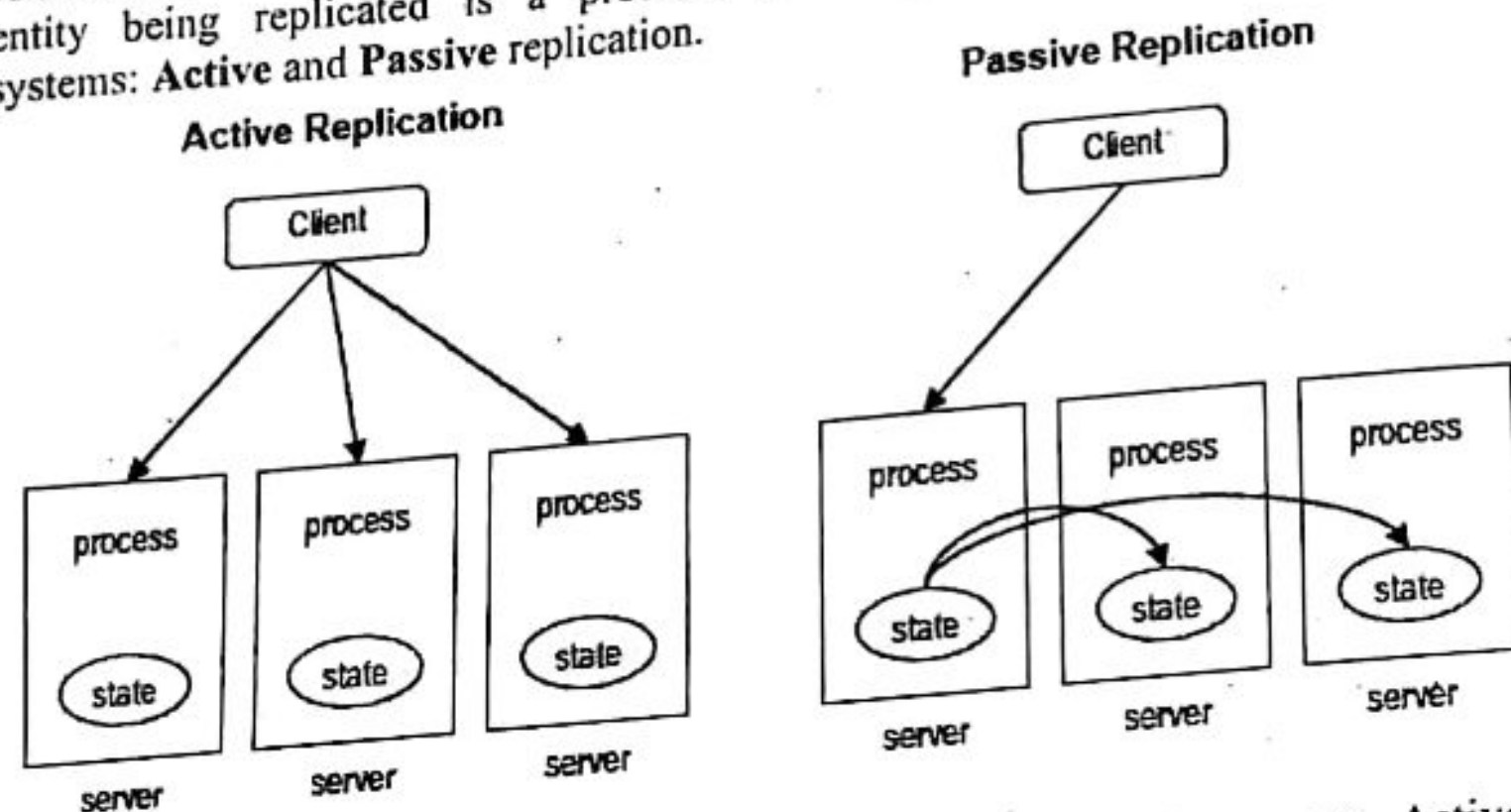
Now P1 can enter CS.

One of the problems with this algorithm is that once a process P1 receives "OK" from all (n-1) processes, it can enter CS subsequently without sending repeat message.

Secondly if the node having process fails then all other processes stare forever. this can be solved by detecting failure of nodes.

Q 8) Differentiate between passive and active replication approach. Discuss with a technique that make the distributed system high available.

Ans: In the distributed systems research area replication is mainly used to provide fault tolerance. The entity being replicated is a process. Two replication strategies have been used in distributed systems: **Active** and **Passive** replication.



In **active replication** each client request is processed by all the servers. Active Replication was first introduced by under the name **state machine replication**. This requires that the process hosted by the servers is **deterministic**. **Deterministic** means that, given the same initial state and a request sequence, all processes will produce the same response sequence and end up in the same final state. To make all the servers receive the same sequence of operations, an **atomic broadcast** protocol must be used. An atomic broadcast protocol guarantees that either all the servers receive a message or none, plus that they all receive messages in the same order. The big disadvantage for active replication is that in practice most of the real-world servers are non-deterministic. Still active replication is the preferable choice when dealing with real-time systems that require quick response even under the presence of faults or with systems that must handle **byzantine faults**.

In **passive replication** there is only one server (called primary) that processes client requests. After processing a request, the primary server updates the state on the other (backup) servers and sends back the response to the client. If the primary server fails, one of the backup servers takes its place. Passive replication may be used even for non-deterministic processes. The disadvantage of passive replication compared to active is that in case of failure the response is delayed.

Second part:

High availability (HA) can be achieved when systems are equipped to operate continuously without failure for a long duration of time. The phrase suggests that parts of a system have been thoroughly tested and accommodated with redundant components to remain functional when a failure occurs.

1. **Middleware HA** — load balancer

A few high-availability solutions exist such as load balancing and basic clustering. Load balancing also known as horizontal scaling can be achieved by adding new nodes with identical functionality to existing

ones, redistributing the load amongst them all. By exclusively redirecting requests to available servers, reliability and availability can be maintained.

2. **Scaling up and down**

In companies, high availability is achieved by scaling the servers up or down depending on the application server's load and availability. It is done mostly outside the application at the server level.

3. **Implementing multiple application servers**

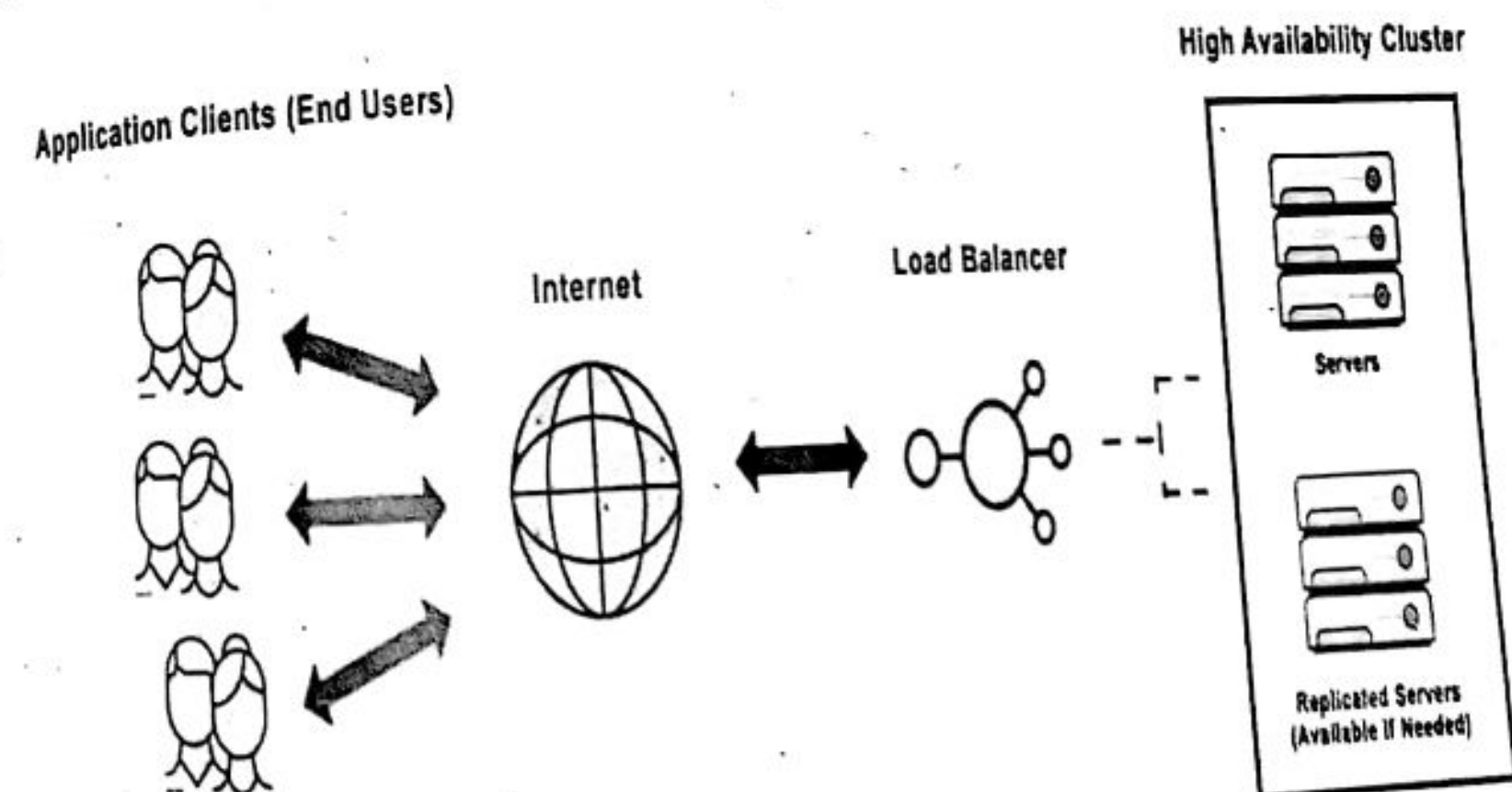
Overburdened servers may crash and cause an outage; it is advisable to deploy applications over multiple servers to keep the applications running all the time. It creates a sense of being always operational.

4. **Multi-region deployments**

When it comes to cloud environments, systems are deployed in units and are referred to as regions. A region can be defined as a data center, or it may be consisting of a set of data centers located somewhat close to each other. Then there comes a more granular unit inside of the regions and is known as availability zone. So, each availability zone is a single data center within one region.

5. **Clustering techniques**

Clustering techniques are generally used to improve and increase the performance and availability of a complex system. A cluster is usually designed as a redundant set of services rendering the same set of functionalities and capabilities.



Q 9) Write down the rules of two-version locking. Explain how optimistic concurrency control mechanism works.

Ans: Two-Phase Locking —

A transaction is said to follow the Two-Phase Locking protocol if Locking and Unlocking can be done in two phases.

Growing Phase: New locks on data items may be acquired but none can be released.

Shrinking Phase: Existing locks may be released but no new locks can be acquired.

Note – If lock conversion is allowed, then upgrading of lock (from S(a) to X(a)) is allowed in the Growing Phase, and downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

Let's see a transaction implementing 2-PL.

	T ₁	T ₂
1	lock-S(A)	
2		lock-S(A)
3	lock-X(B)	
4
5	Unlock(A)	
6		Lock-X(C)
7	Unlock(B)	
8		Unlock(A)
9		Unlock(C)
10

This is just a skeleton transaction that shows how unlocking and locking work with 2-PL. Note for:
Transaction T₁:

The growing Phase is from steps 1-3.

The shrinking Phase is from steps 5-7.

Lock Point at 3

Transaction T₂:

The growing Phase is from steps 2-6.

The shrinking Phase is from steps 8-9.

Lock Point at 6

Second part: Optimistic Concurrency Control Algorithm

In systems with low conflict rates, the task of validating every transaction for serializability may lower performance. In these cases, the test for serializability is postponed to just before commit. Since the conflict rate is low, the probability of aborting transactions which are not serializable is also low. This approach is called optimistic concurrency control technique.

In this approach, a transaction's life cycle is divided into the following three phases –

- **Execution Phase** – A transaction fetches data items to memory and performs operations upon them.
- **Validation Phase** – A transaction performs checks to ensure that committing its changes to the database passes serializability test.
- **Commit Phase** – A transaction writes back modified data item in memory to the disk.

This algorithm uses three rules to enforce serializability in validation phase –

Rule 1 – Given two transactions T_i and T_j, if T_i is reading the data item which T_j is writing, then T_i's execution phase cannot overlap with T_j's commit phase. T_j can commit only after T_i has finished execution.

Rule 2 – Given two transactions T_i and T_j, if T_i is writing the data item that T_j is reading, then T_i's commit phase cannot overlap with T_j's execution phase. T_j can start executing only after T_i has already committed.

Rule 3 – Given two transactions T_i and T_j, if T_i is writing the data item which T_j is also writing, then T_i's commit phase cannot overlap with T_j's commit phase. T_j can start to commit only after T_i has already committed.

Q 10) How does triple modular redundancy works? Explain how reliable client server communication can be achieved in distributed system.

Ans: In computing, triple modular redundancy, sometimes called triple-mode redundancy, (TMR) is a fault-tolerant form of N-modular redundancy, in which three systems perform a process and that result is processed by a majority-voting system to produce a single output.

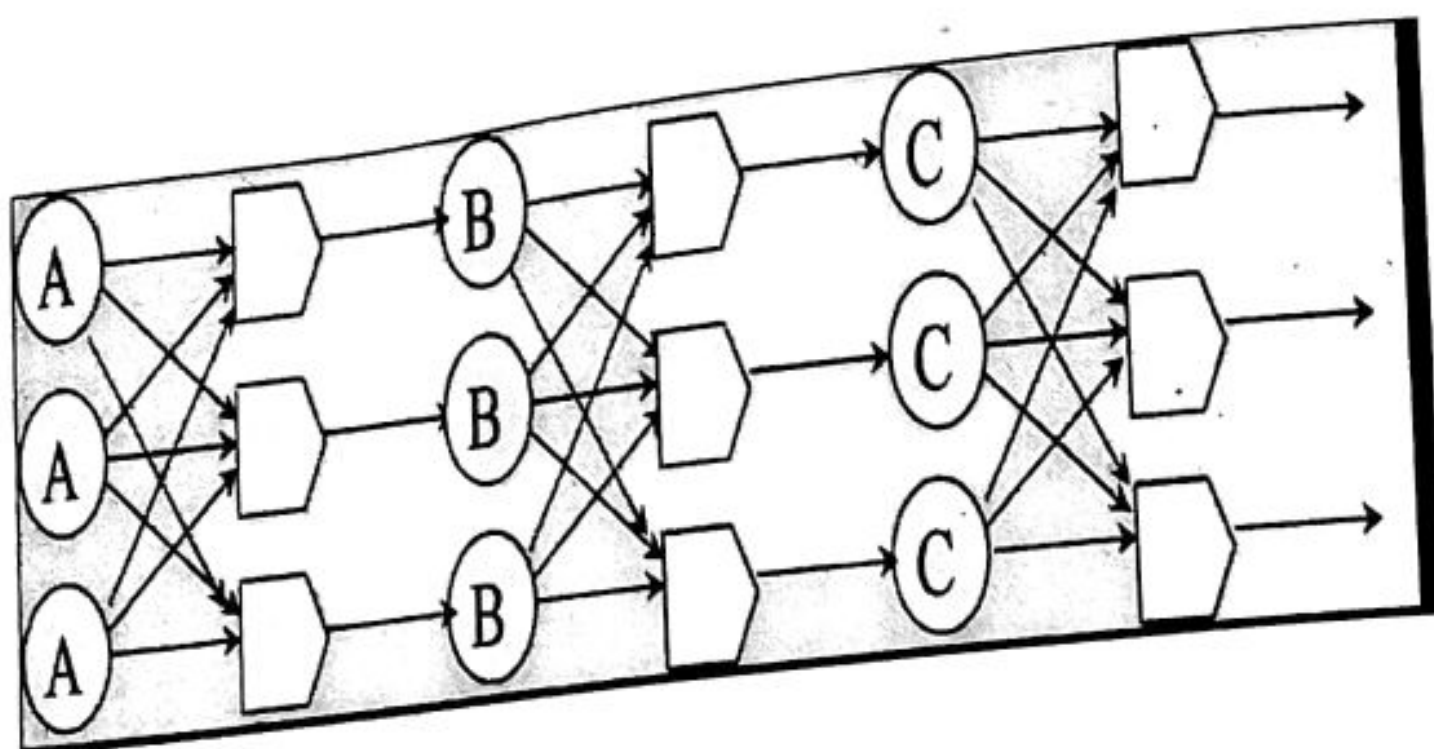


Fig :TMR

Second part:

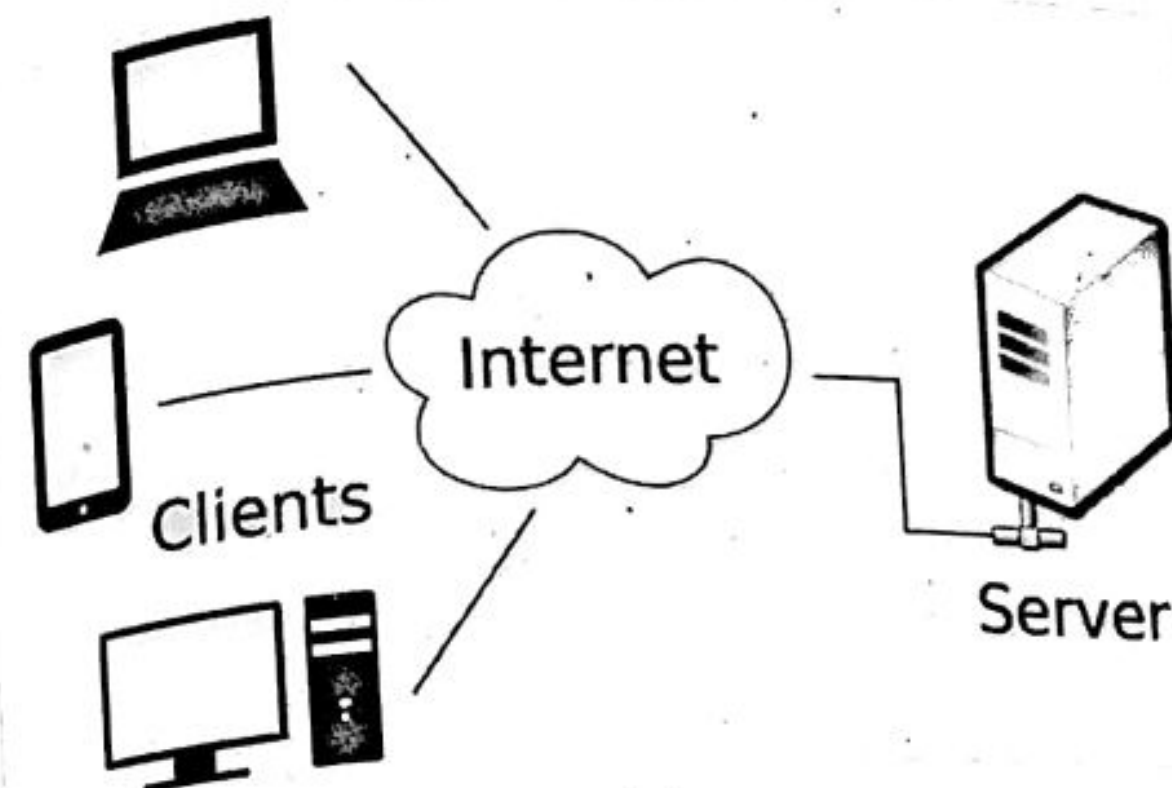
The Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client. Clients do not share any of their resources. Examples of Client-Server Model are Email, World Wide Web, etc.

How the Client-Server Model works ?

In this article we are going to take a dive into the Client-Server model and have a look at how the Internet works via, web browsers. This article will help us in having a solid foundation of the WEB and help in working with WEB technologies with ease.

- **Client:** When we talk the word **Client**, it mean to talk of a person or an organization using a particular service. Similarly in the digital world a **Client** is a computer (**Host**) i.e. capable of receiving information or using a particular service from the service providers (**Servers**).
- **Servers:** Similarly, when we talk the word **Servers**, It mean a person or medium that serves something. Similarly in this digital world a **Server** is a remote computer which provides information (data) or access to particular services.

So, its basically the **Client** requesting something and the **Server** serving it as long as its present in the database.



Advantages of Client-Server model:

- Centralized system with all data in a single place.
- Cost efficient requires-less maintenance cost and Data recovery is possible.
- The capacity of the Client and Servers can be changed separately.

Disadvantages of Client-Server model:

- Clients are prone to viruses, Trojans and worms if present in the Server or uploaded into the Server.
- Server are prone to Denial of Service (DOS) attacks.
- Data packets may be spoofed or modified during transmission.
- Phishing or capturing login credentials or other useful information of the user are common and MITM(Man in the Middle) attacks are common.

Q 11) Write Short notes :

a. Lamport's Clock

b. TIB/Rendezvous

c. feedback suppression mechanism in M-cast communication

a. Lamport's clock

- The ordering of events is based on two situations:

1. If two events within a same process occurred, they occurred in the order in which that process observes.

2. Whenever a message is sent between processes, the event of sending message occurred before the event of receiving the message.

- Lamport generalizes the two conditions to form happened-before relation, denoted by \rightarrow
i.e $a \rightarrow b$; meaning that event a happened before event b

- According to Lamport:

1. If for some process p_i : $a \rightarrow b$, then $a \rightarrow b$.
2. For any message m , $\text{send}(m) \rightarrow \text{receive}(m)$
3. If a , b and c are events such that $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$.
4. If $a \rightarrow b$, event a casually affects event b .
5. If $a \rightarrow e$ and $e \rightarrow a$ are false, then a and e are concurrent events, which can be written as $a \parallel e$.

Implementation Rules:

1. CP_i is incremented before each event is issued at process P_i .
 $CP_i := CP_i + 1$
2. a) When $\text{send}(m)$ is a event of process P_i , timestamp $tm = CP_i(a)$ is included in m .
b) On receiving message m by P_j , its clock CP_j is updated as:
 $CP_j := \max [CP_j, tm]$
- c) The new value of CP_j is used to timestamp event $\text{receive}(m)$ by P_j

Problems:

1. Lamport's logical clock impose only partial order on set of events but pairs of distinct events of different processes can have identical time stamp.
2. Total ordering can be enforced by global logical time stamp.

b. TIB/ Rendezvous

An agreement between two or more entity to meet at a certain time and place.

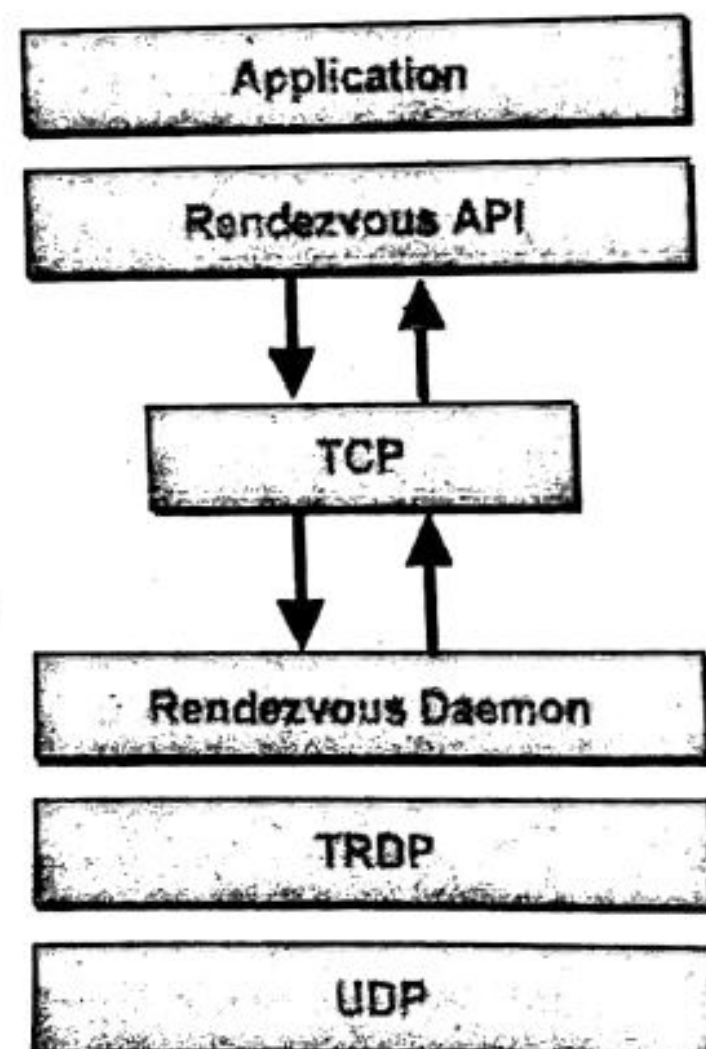


Fig : TIB/Rendezvous

c. Feedback suppression mechanism in M-cast communication

Feedback implosion is one of the crucial issues confronting the scalability of reliable multicast protocols. Generally, there are two ways of solving this feedback implosion problem-timer-based and structure-based approaches. In the timer-based approach, when a receiver needs to send feedback, it multicasts the feedback to all receivers earlier than the other receivers, and hence suppresses others from returning feedback in response to the same event. The structure-based approach depends on intermediate network elements to consolidate feedback arising from the same event. It hence significantly reduces the number of feedback messages arriving at the sender from a single event. In this paper, we present an effective end-to-end NACK suppression algorithm which could solve the problem caused by correlative packet loss and independent packet loss among receivers



2074 Ashwin

Q1) Why there are challenges in achieving some requirements of a distributed system? Explain the challenges associated with different requirement of a distributed system.

Ans: The distributed information system is defined as "a number of interdependent computers linked by a network for sharing information among them". A distributed information system consists of multiple autonomous computers that communicate or exchange information through a computer network.

Design issues of distributed system –

1. **Heterogeneity:** Heterogeneity is applied to the network, computer hardware, operating system and implementation of different developers. A key component of the heterogeneous distributed system client-server environment is middleware. Middleware is a set of services that enables application and end-user to interact with each other across a heterogeneous distributed system.
2. **Openness:** The openness of the distributed system is determined primarily by the degree to which new resource-sharing services can be made available to the users. Open systems are characterized by the fact that their key interfaces are published. It is based on a uniform communication mechanism and published interface for access to shared resources. It can be constructed from heterogeneous hardware and software.
3. **Scalability:** Scalability of the system should remain efficient even with a significant increase in the number of users and resources connected.
4. **Security:** Security of information system has three components Confidentiality, integrity and availability. Encryption protects shared resources, keeps sensitive information secrets when transmitted.
5. **Failure Handling:** When some faults occur in hardware and the software program, it may produce incorrect results or they may stop before they have completed the intended computation so corrective measures should be implemented to handle this case. Failure handling is difficult in distributed systems because the failure is partial i.e., some components fail while others continue to function.
6. **Concurrency:** There is a possibility that several clients will attempt to access a shared resource at the same time. Multiple users make requests on the same resources, i.e. read, write, and update. Each resource must be safe in a concurrent environment. Any object that

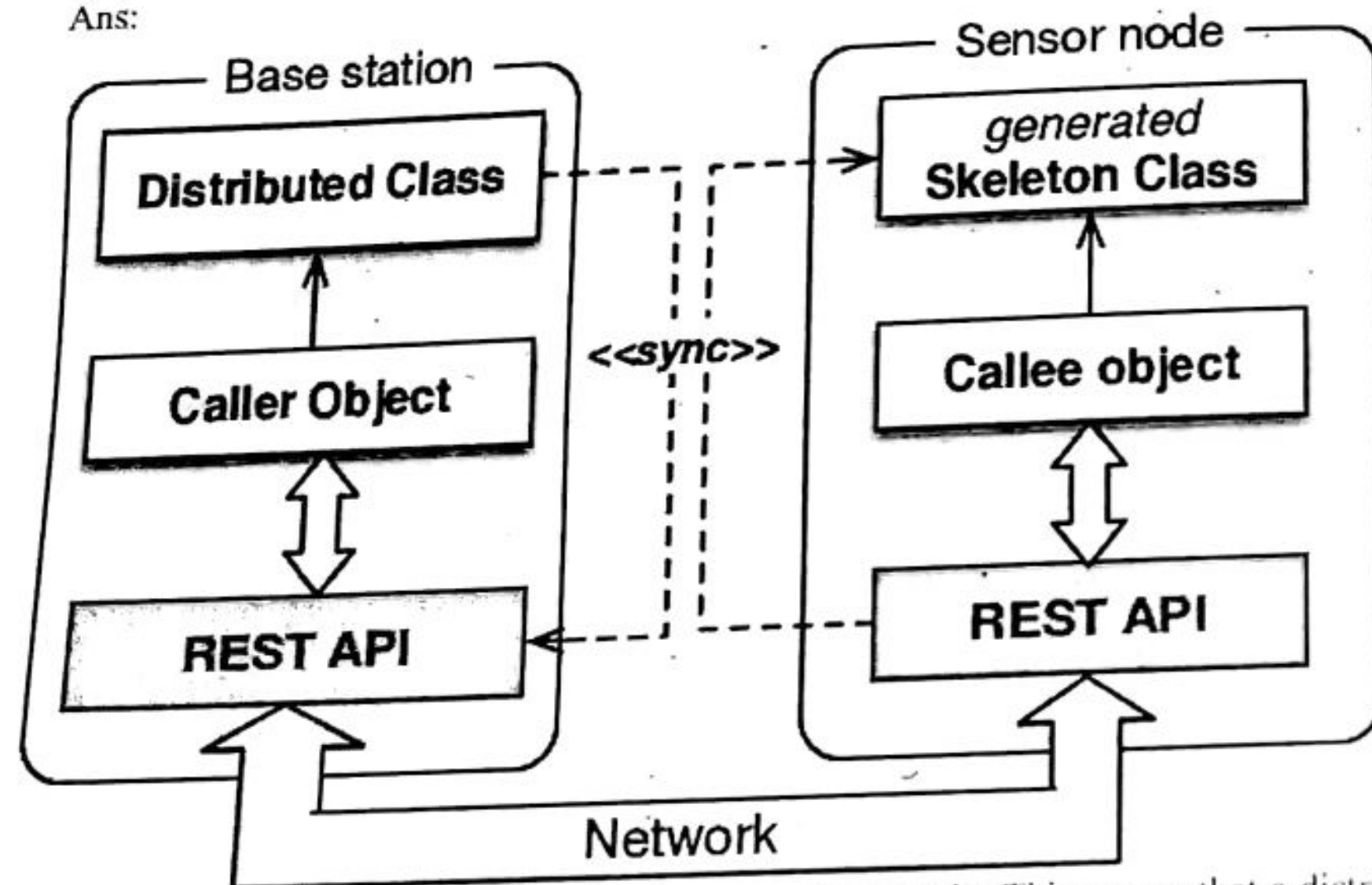
represents a shared resource a distributed system must ensure that it operates correctly in a concurrent environment.

7. **Transparency:** Transparency ensures that the distributed system should be perceived as a single entity by the users or the application programmers rather than the collection of autonomous systems, which is cooperating. The user should be unaware of where the services are located and the transferring from a local machine to a remote one should be transparent.

Types of Transparency One of the major objectives of Distributed database system is providing the appearance of centralized system to end user. The eight transparencies are believed to incorporate the desired functions of a distributed database system. Such an image is accomplished by using the following transparencies: • Access Transparency • Location Transparency • Concurrency Transparency • Replication Transparency • Failure Transparency • Migration Transparency • Performance Transparency • Scaling Transparency

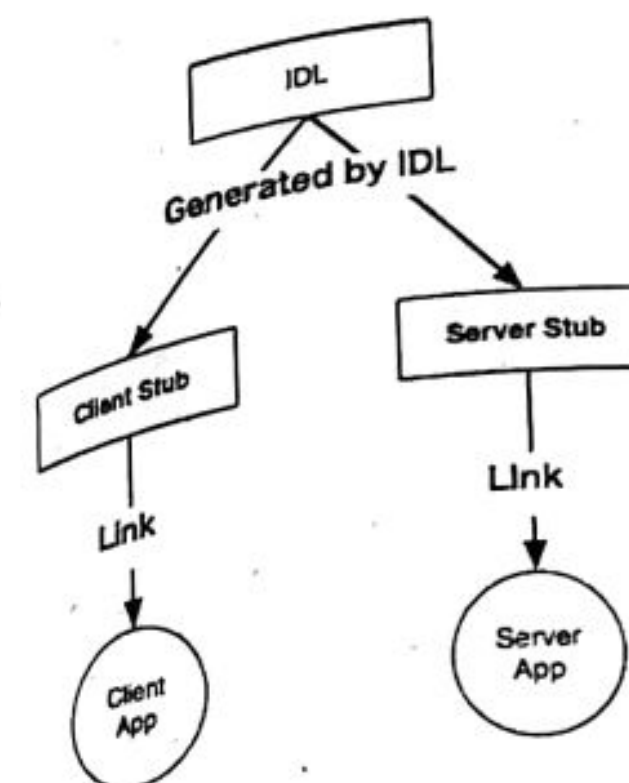
Q2) Define Distributed object and IDL. Compare RPC and RMI architecture.

Ans:



A distributed object is an object that can be accessed remotely. This means that a distributed object can be used like a regular object, but from anywhere on the network. An object is typically considered to encapsulate data and behavior. The location of the distributed object is not critical to the user of the object.

IDL



IDL stands for Interface Definition Language. Its purpose is to define the capabilities of a distributed service along with a common set of data types for interacting with those distributed services. IDL meets several objectives:

1. Language Independence
2. Distributed service specification
3. Definition of complex data types:
4. Hardware Independence:

Second part:

RPC	RMI
RPC is a library and OS dependent platform.	Whereas it is a java platform.
RPC supports procedural programming.	RMI supports object-oriented programming.
RPC is less efficient in comparison of RMI.	While RMI is more efficient than RPC.
There is multiple codes are needed for simple application in RPC.	While there is multiple codes are not needed for simple application in RMI.
RPC creates more overhead.	While it creates less overhead than RPC.
RPC does not provide any security.	While it provides client level security.

The parameters which are passed in RPC are ordinary or normal data.

While in RMI, objects are passed as parameter.

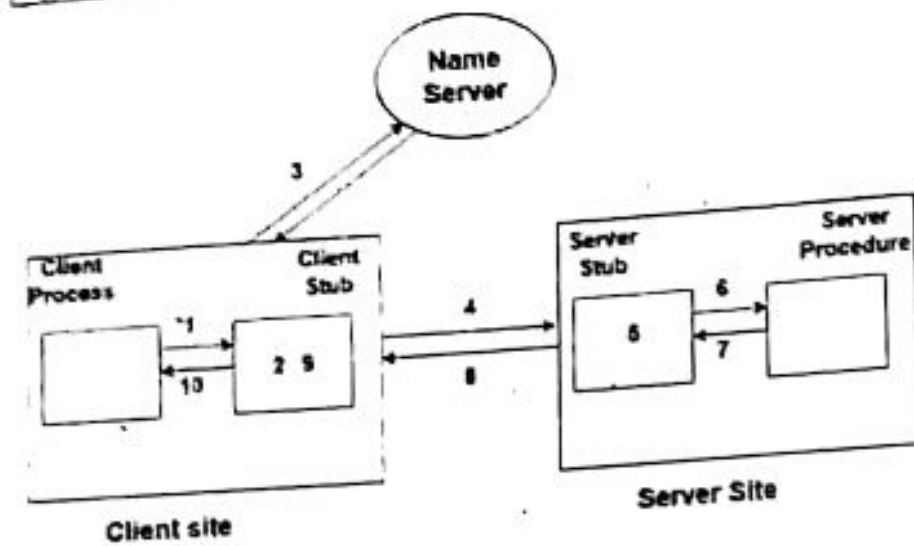


Fig :RPC

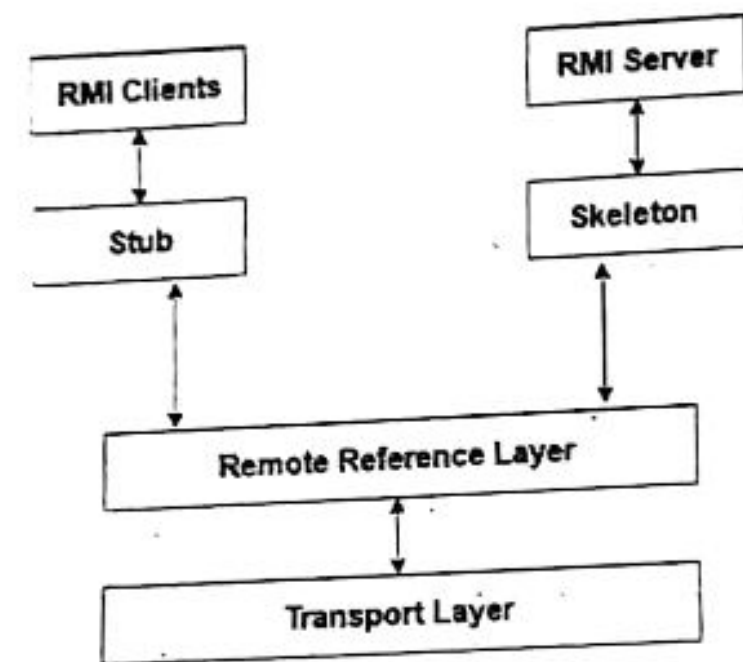


Fig: RMI

Q3) What is stateful and stateless service in file system? Explain the DNS working Mechanism with suitable practical example.

Ans:

1) State Information:

- A Stateful server remember client data (state) from one request to the next. Stateful servers, do store session state. They may, therefore, keep track of which clients have opened which files, current read and write pointers for files, which files have been locked by which clients, etc.
- A Stateless server keeps no state information. Stateless file servers do not store any session state. This means that every client request is treated independently, and not as a part of a new or existing session.

2) Programming :

- Stateful server is harder to code.
- Stateless server is straightforward to code.

3) Crash recovery :

- Stateful servers have difficult crash recovery due to loss of information.
- Stateless servers can easily recover from failure because there is no state that must be restored.

4) Information transfer :

- Using a Stateful server, file server, the client can send less data with each request.
- Using a stateless file server, the client must, specify complete file names in each request specify location for reading or writing re-authenticate for each request.

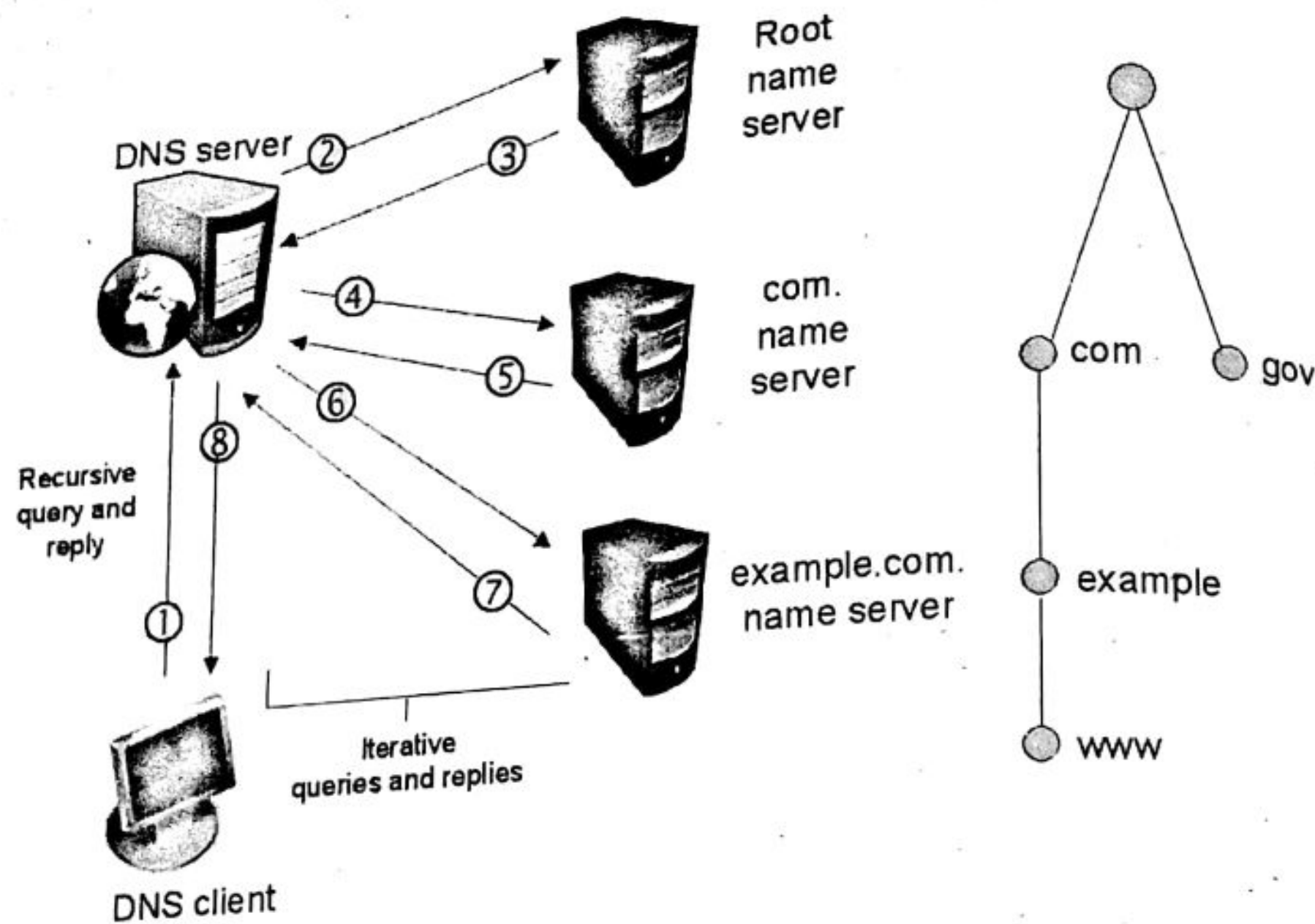
Advantages of both Stateful and Stateless servers:

The main advantage of Stateful servers, is that they can provide better performance for clients because clients do not have to provide full file information every time they perform an operation, the size of messages to and from the server can be significantly decreased. Likewise the server can make use of knowledge of access patterns to perform read-ahead and do other optimizations. Stateful servers can also offer clients extra services such as file locking, and remember read and write positions.

The main advantage of stateless servers is that they can easily recover from failure. Because there is no state that must be restored, a failed server can simply restart after a crash and immediately provide services to clients as though nothing happened. Furthermore, if clients crash the server is not stuck with abandoned opened or locked files. Another benefit is that the server implementation remains simple because it does not have to implement the state accounting associated with opening, closing, and locking of files.

Second part:

The Domain Name System (DNS) is the phonebook of the Internet. Humans access information online through domain names, like nytimes.com or espn.com. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses so browsers can load Internet resources. Each device connected to the Internet has a unique IP address which other machines use to find the device.



The 8 steps in a DNS lookup:

1. A user types 'example.com' into a web browser and the query travels into the Internet and is received by a DNS recursive resolver.
2. The resolver then queries a DNS root nameserver (.).
3. The root server then responds to the resolver with the address of a Top-Level Domain (TLD) DNS server (such as .com or .net), which stores the information for its domains. When searching for example.com, our request is pointed toward the .com TLD.
4. The resolver then makes a request to the .com TLD.
5. The TLD server then responds with the IP address of the domain's nameserver, example.com.
6. Lastly, the recursive resolver sends a query to the domain's nameserver.
7. The IP address for example.com is then returned to the resolver from the nameserver.
8. The DNS resolver then responds to the web browser with the IP address of the domain requested initially.

Once the 8 steps of the DNS lookup have returned the IP address for example.com, the browser is able to make the request for the web page:

9. The browser makes a HTTP request to the IP address.
10. The server at that IP returns the webpage to be rendered in the browser (step 10).

Q4) What are the characteristics of distributed operating system? Explain ORB and its interfaces.

Ans: There are various important goals that must be met to build a distributed system worth the effort. A **distributed system** should easily connect users to resources, it should hide the fact that resources are distributed across a network, must be open, and must be scalable.

1. Connecting Users and Resources :

The main goal of a distributed system is to make it easy for users to access remote resources and to share them with other users in a controlled manner. Resources can be virtually

anything, typical examples of resources are printers, storage facilities, data, files, web pages, and networks. There are many reasons for sharing resources. One reason is economics.

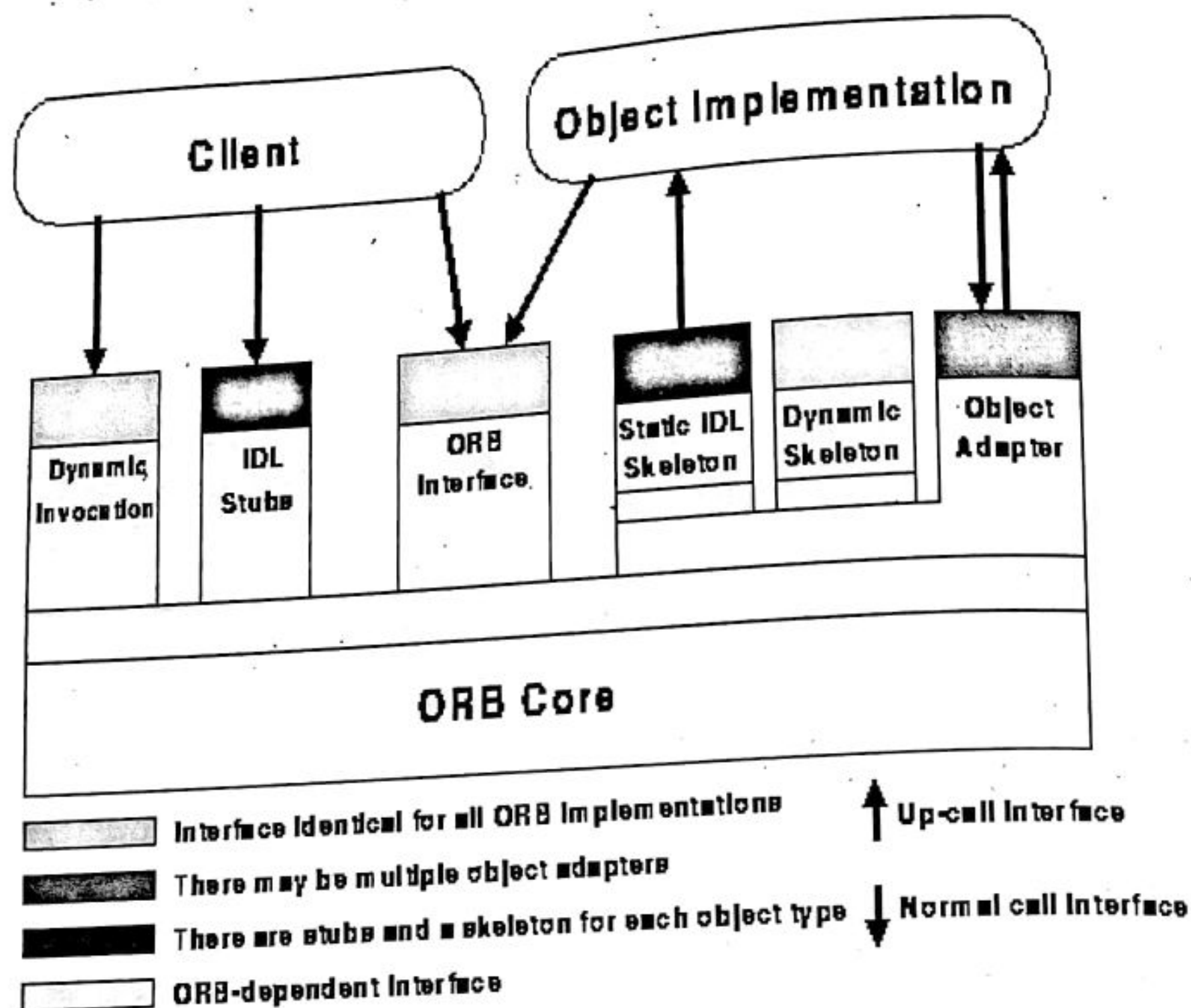
2. Transparency :

An important goal of a distributed system is to hide the fact that its process and resources are physically distributed across multiple computers. A distributed system that is capable of presenting itself to users and applications such that it is only a single computer system is called transparent.

The concept of transparency can be applied to many aspects of a distributed system as shown in table.

Different Forms of Transparency -

S.No.	Transparency	Description	Second part: In CORBA, the Object Request Broker or ORB takes care of all of the details involved in routing a request from client to object, and routing the response to its destination.
(1)	Access	Hide data representation.	
(2)	Location	Hide location	
(3)	Migration	Move place information.	
(4)	Relocation	Hide moved place relocation.	
(5)	Replication	Hide that a resource is replication.	
(6)	Concurrency	Shared data bases access	
(7)	Failure	Hide fact about resource failure.	
(8)	Persistence	Hide fact about memory location.	



The Object Request Broker (ORB) is middleware that uses the CORBA specification. The Object Request Broker or ORB takes care of all of the details involved in routing a request from client to object and routing the response to its destination. The ORB is also the custodian of the Interface Repository (abbreviated variously IR or IFR), an OMG-standardized distributed database containing OMG IDL interface definitions.

On the client side, then, the ORB provides interface definitions from the IFR, and constructs invocations for use with the Dynamic Invocation Interface (DII). It also converts Object References between session and stringified format, and (for CORBA 2.4 and later ORBs) converts URL-format corba loc and corbaname object references to session references.

On the server side, the ORB de-activates inactive objects, and re-activates them whenever a request comes in. CORBA supports a number of activation patterns, so that different object or component types can activate and de-activate in the way that uses resources best.

Q5) Why clock synchronization is necessary? Explain the clock synchronization algorithm using vector clock along with an example.

Ans: Clock synchronization is necessary for the ordering of events and to preserve the state of resources. As per algorithms, we can say that for clock synchronization there is need to consider propagation time of messages among each node in both types of algorithms centralized and distributed. In DCS, all the

computers share their own data. In order to access updated data from each computer, time stamp of all computers must be same. So we have to make sure that all the computers' clocks (which are part of DCS) should be synchronized.

Second part:

Vector Clock is an algorithm that generates partial ordering of events and detects causality violations in a distributed system. These clocks expand on Scalar time to facilitate a causally consistent view of the distributed system. They detect whether a contributed event has caused another event in the distributed system. It essentially captures all the causal relationships. This algorithm helps us label every process with a vector (a list of integers) with an integer for each local clock of every process within the system. So for N given processes, there will be vector/ array of size N.

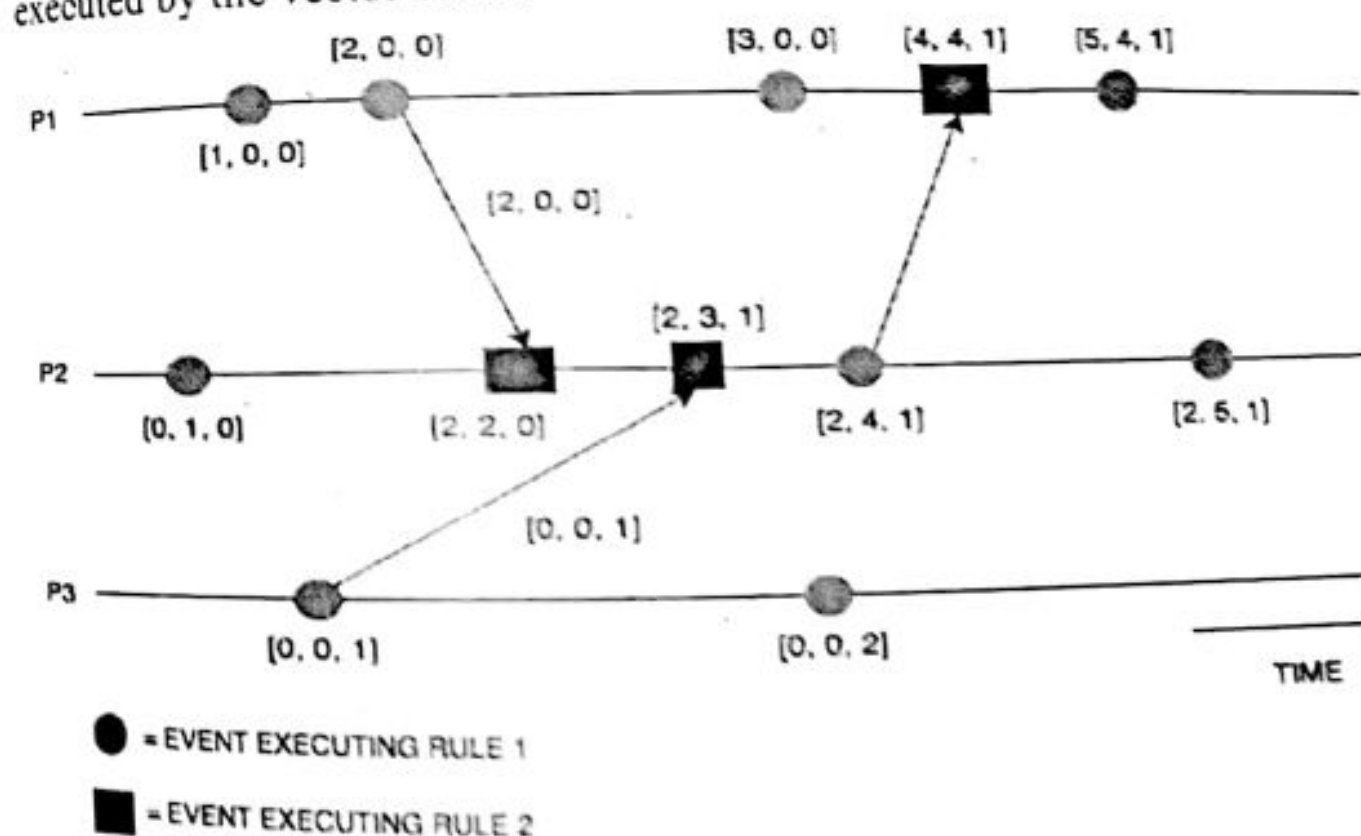
How does the vector clock algorithm work :

- Initially, all the clocks are set to zero.
- Every time, an Internal event occurs in a process, the value of the processes's logical clock in the vector is incremented by 1
- Also, every time a process sends a message, the value of the processes's logical clock in the vector is incremented by 1.

Every time, a process receives a message, the value of the processes's logical clock in the vector is incremented by 1, and moreover, each element is updated by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

Example

Consider a process (P) with a vector size N for each process: the above set of rules mentioned are to be executed by the vector clock:



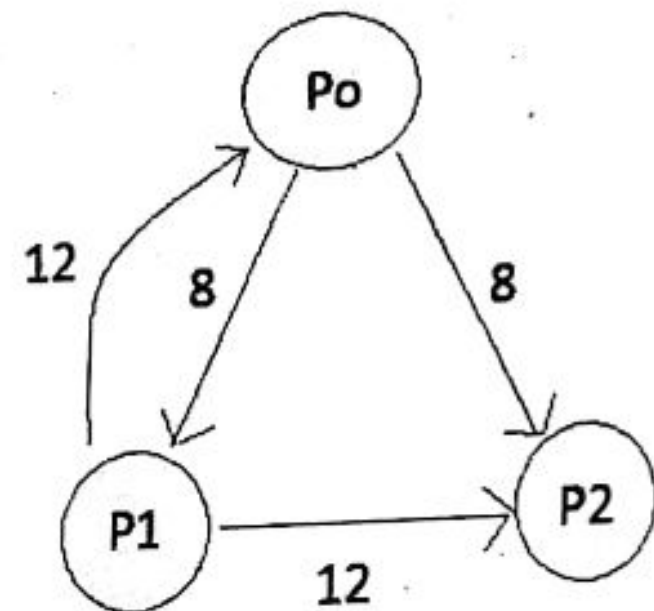
The above example depicts the vector clocks mechanism in which the vector clocks are updated after execution of internal events, the arrows indicate how the values of vectors are sent in between the processes (P1, P2, P3).

To sum up, Vector clocks algorithms are used in distributed systems to provide a causally consistent ordering of events but the entire Vector is sent to each process for every message sent, in order to keep the vector clocks in sync.

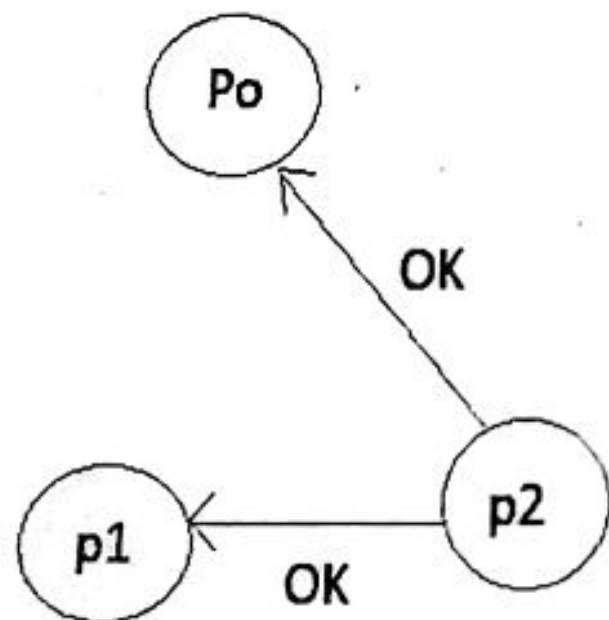
Q6) Describe non token based centralized and Ricart Agrawala algorithm with example and compare them.

Ricart - Agarwala's a non token based algorithm that uses broadcast technique for mutual calculation.

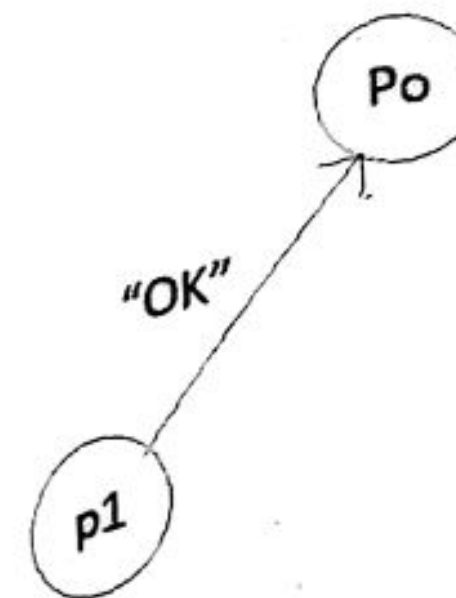
Consider a scenario when process P0 & P1 went to enter critical sector. Both P0 & P1 send broadcast to all 3 processes in the system along with time stamp.



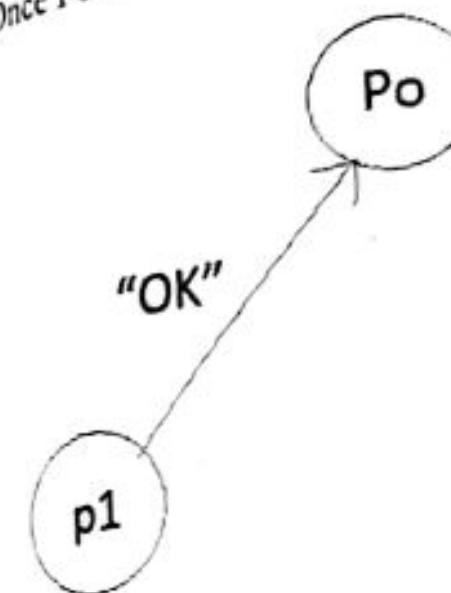
Since P2 does not want to enter critical section hence it sends "OK" to both the processes.



Since P1 finds Ts of P0 to be lesser than Ts of self therefore P1 sends to P0.



Now P0 can enter critical section, as it is received (n-1) OK messages.
Once P0 is done with critical section, it sends "Ok" message to P1.



After completion of CS by P0

Now P1 can enter CS.

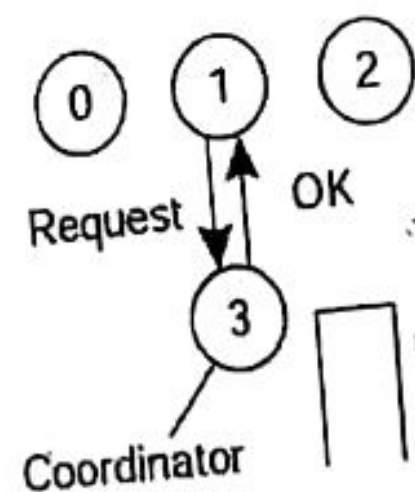
One of the problems with this algorithm is that once a process P1 receives "OK" from all (n-1) processes, it can enter CS subsequently without sending repeat message.

Secondly if the node having process fails then all other processes stare forever, this can be solved by detecting failure of nodes.

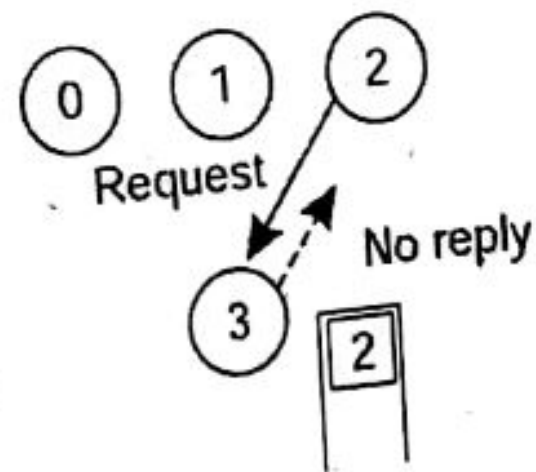
Non token based centralized algorithm

Centralized coordinator Algorithm

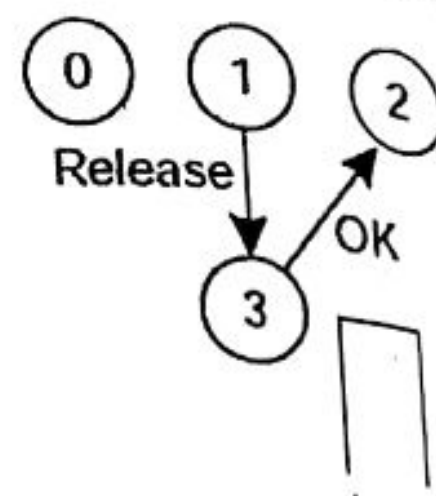
- In centralized algorithm one process is elected as the coordinator which may be the machine with the highest address.



(a)



(b)

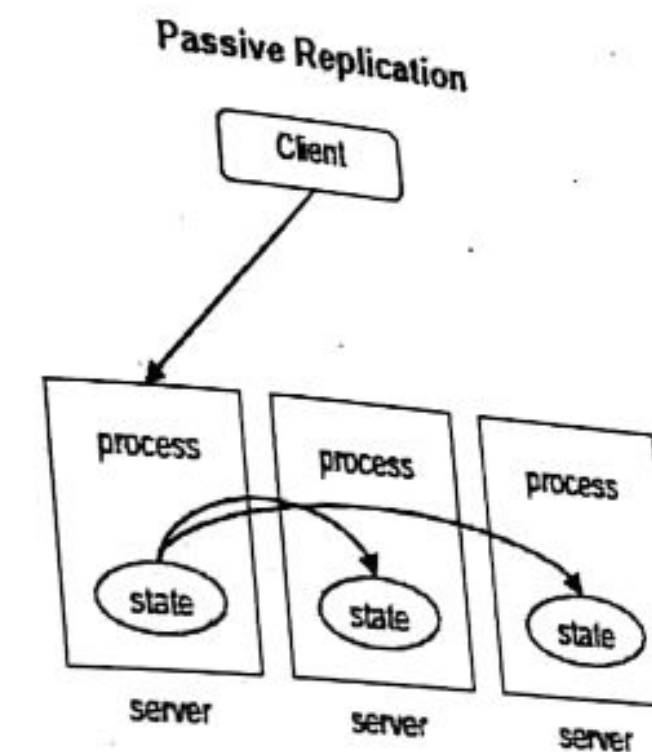
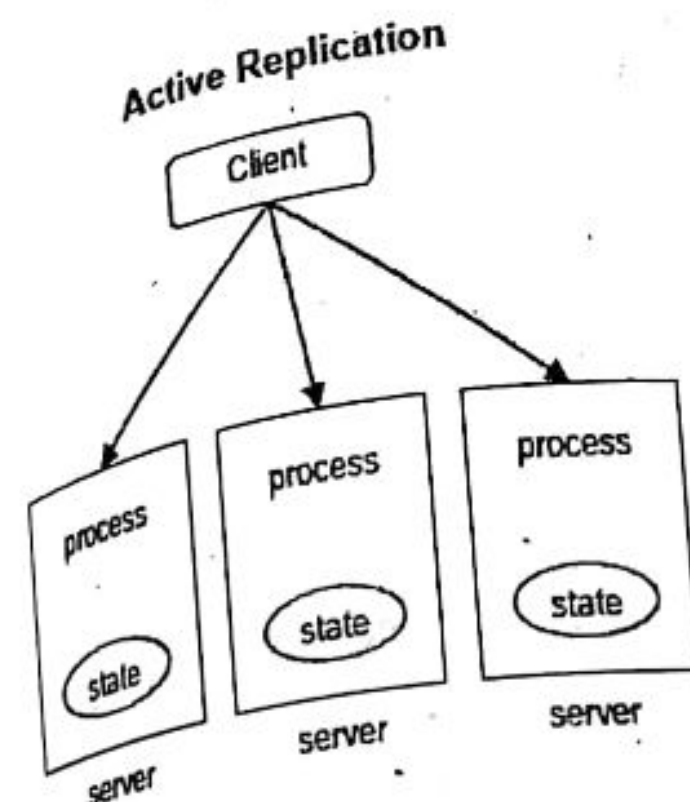


(c)

- Whenever a process wants to enter a critical region, it sends a request message to the coordinator stating which critical region it wants to enter and asking for permission. If no other process is currently in that critical region, the coordinator sends back a reply granting permission, as shown in Fig (a). When the reply arrives, the requesting process enters the critical region.
- Suppose another process 2 shown in Fig (b), asks for permission to enter the same critical region. Now the coordinator knows that a different process is already in the critical region, so it cannot grant permission. The coordinator just refrains from replying, thus blocking process 2, which is waiting for a reply, or it could send a reply saying 'permission denied.'
- When process 1 exits the critical region, it sends a message to the coordinator releasing its exclusive access as shown in Fig (c).
- The coordinator takes the first item off the queue of deferred requests and sends that process a grant message. If the process was still blocked it unblocks and enters the critical region.
- If an explicit message has already been sent denying permission, the process will have to poll for incoming traffic or block later. When it sees the grant, it can enter the critical region.

Q 7) Differentiate between active and passive replication. Explain working mechanism of active replication.

In the distributed systems research area replication is mainly used to provide fault tolerance. The entity being replicated is a process. Two replication strategies have been used in distributed systems: **Active** and **Passive** replication.



In **active replication** each client request is processed by all the servers. Active Replication was first introduced by under the name **state machine replication**. This requires that the process hosted by the servers is **deterministic**. **Deterministic** means that, given the same initial state and a request sequence, all processes will produce the same response sequence and end up in the same final state. To make all the servers receive the same sequence of operations, an **atomic broadcast** protocol must be used. An atomic broadcast protocol guarantees that either all the servers receive a message or none, plus that they all receive messages in the same order. The big disadvantage for active replication is that in practice most of the real-world servers are non-deterministic. Still active replication is the preferable choice when dealing with real time systems that require quick response even under the presence of faults or with systems that must handle **byzantine faults**.

In **passive replication** there is only one server (called primary) that processes client requests. After processing a request, the primary server updates the state on the other (backup) servers and sends back the response to the client. If the primary server fails, one of the backup servers takes its place. Passive replication may be used even for non-deterministic processes. The disadvantage of passive replication compared to active is that in case of failure the response is delayed.

Q8) How cascading aborts occurs and solved? Explain three phase commit protocol with state diagram.

Ans

Cascading Rollback:

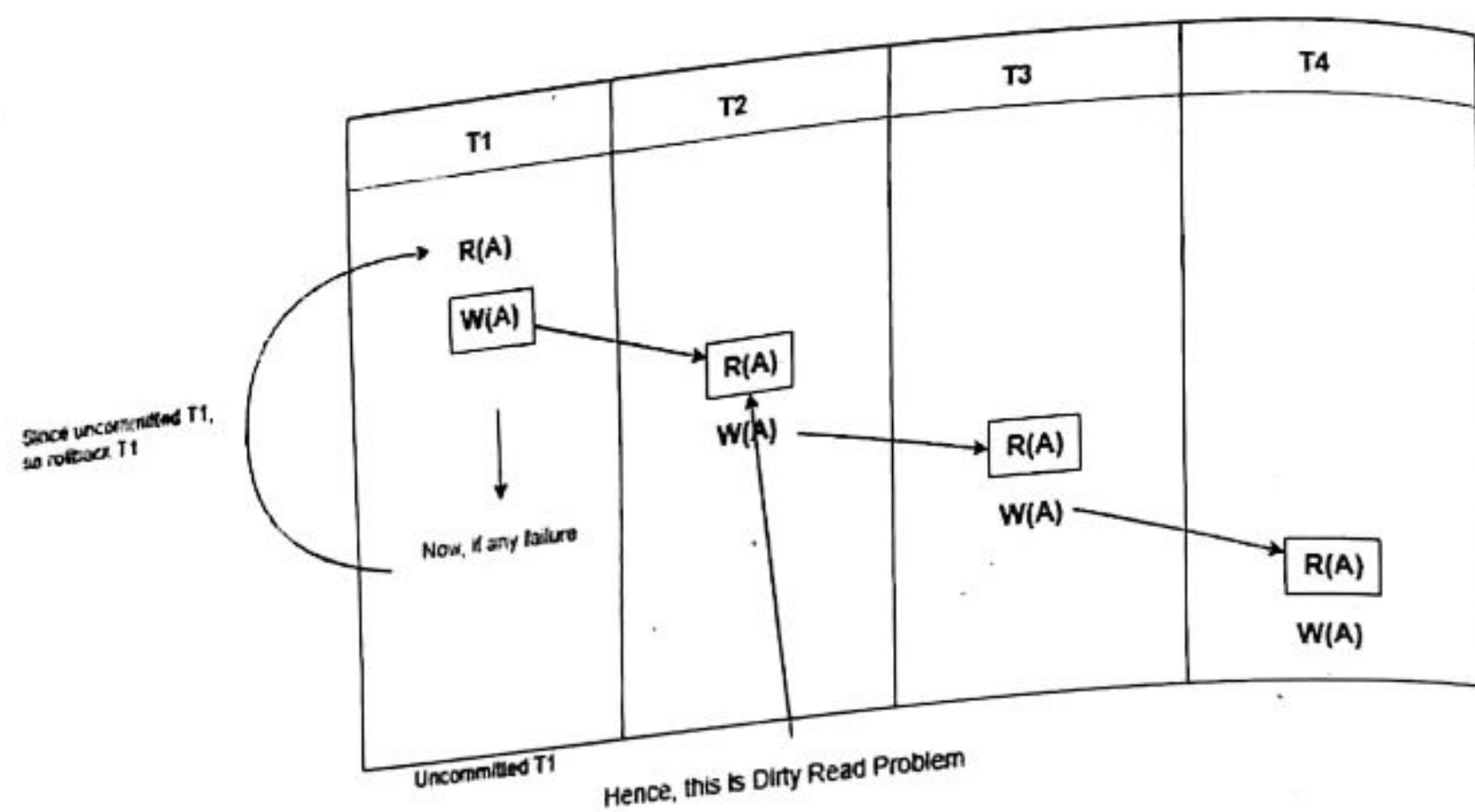
If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a Cascading Rollback or Cascading Abort or Cascading Schedule. It simply leads to the wastage of CPU time.

These Cascading Rollbacks occur because of **Dirty Read problems**.

For example, transaction T1 writes uncommitted x that is read by Transaction T2. Transaction T2 writes uncommitted x that is read by Transaction T3.

Suppose at this point T1 fails.

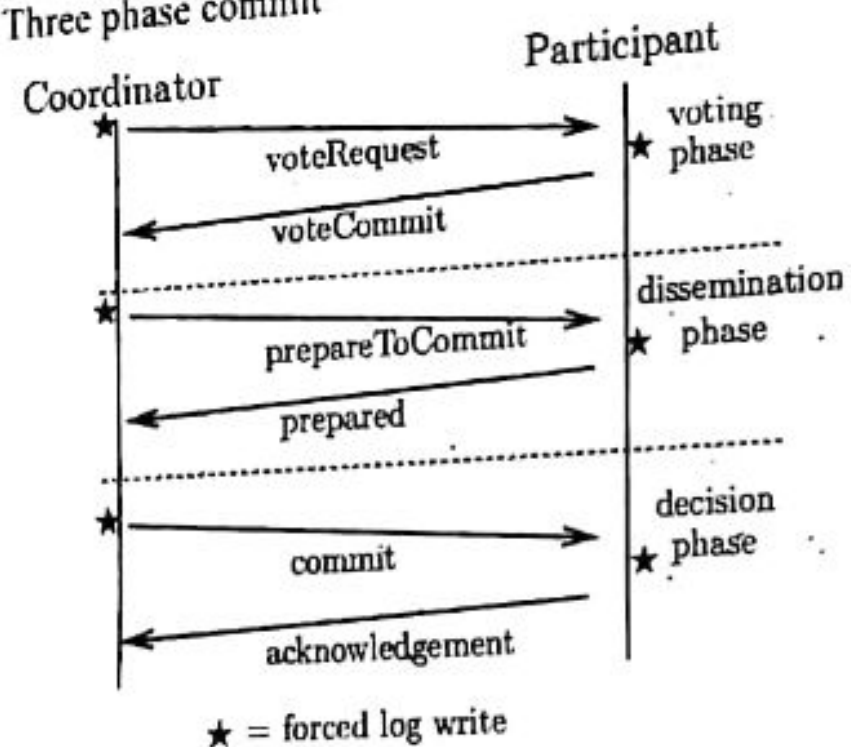
T1 must be rolled back, since T2 is dependent on T1, T2 must be rolled back, and since T3 is dependent on T2, T3 must be rolled back.



Because of T1 rollback, all T2, T3, and T4 should also be rollback (Cascading dirty read problem). This phenomenon, in which a single transaction failure leads to a series of transaction rollbacks is called **Cascading rollback**.

Second part:

Three phase commit



Three-Phase Commit (3PC) Protocol is an extension of the Two-Phase Commit (2PC) Protocol that avoids blocking problem under certain assumptions. In particular, it is assumed that no network partition occurs, and not more than k sites fail, where we assume ' k ' is predetermined number. With the mentioned assumptions, protocol avoids blocking by introducing an extra third phase where multiple sites are involved in the decision to commit. Instead of directly noting the commit decision in its persistent storage, the coordinator first ensures that at least ' k ' other sites know that it intended to commit transaction. In a situation where coordinator fails, remaining sites are bound to first select new coordinator. This new coordinator checks status of the protocol from the remaining sites. If the coordinator had decided to commit, at least one of other ' k ' sites that it informed will be up and will ensure that commit decision is respected. The new coordinator restarts third phase of protocol if any of

rest sites knew that old coordinator intended to commit transaction. Otherwise, new coordinator aborts the transaction.

Q9) What is k-fault tolerant system? Explain fault recovery technique.

Ans: A system is said to be k -fault tolerant if it can withstand k -faults.

If the components fail silently, then it is sufficient to have $k+1$ components to achieve k fault tolerance: k components can fail and one will still be working.

If the components exhibit Byzantine faults, then a minimum of $2k+1$ components are needed to achieve k fault tolerance. In the worst case, k components will fail (generating false results) and $k+1$ components will remain working properly, providing a majority vote that is correct.

Second part: **Fault-tolerance** is the process of working of a system in a proper way in spite of the occurrence of the failures in the system. Even after performing the so many testing processes there is possibility of failure in system. Practically a system can't be made entirely error free. Hence, systems are designed in such a way that in case of error availability and failure, system does the work properly and given correct result. Any system has two major components – Hardware and Software. Fault may occur in either of it. So there are separate techniques for fault-tolerance in both hardware and software.

Hardware Fault-tolerance Techniques:

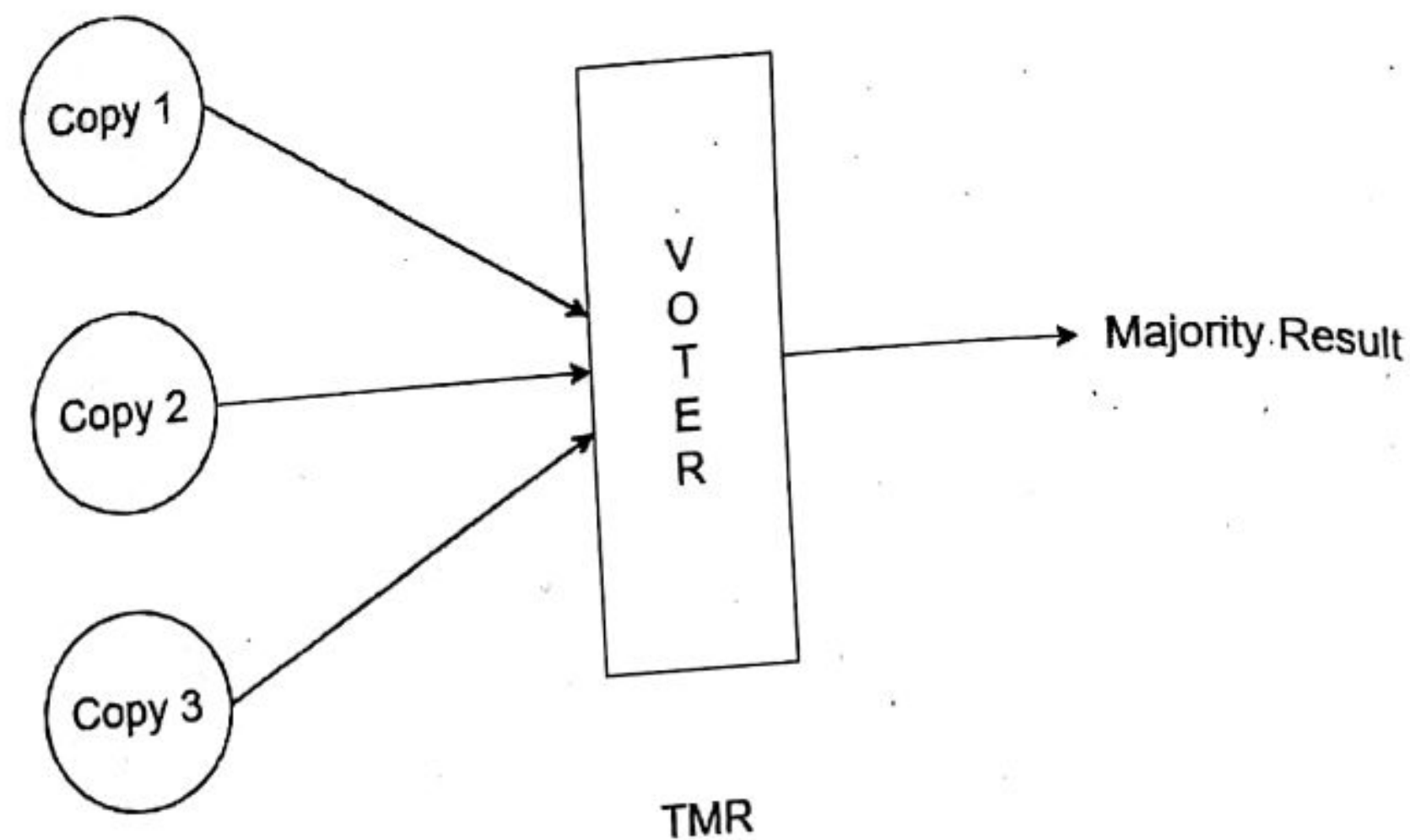
Making a hardware fault-tolerance is simple as compared to software. Fault-tolerance techniques make the hardware work proper and give correct result even some fault occurs in the hardware part of the system. There are basically two techniques used for hardware fault-tolerance:

1. BIST –

BIST stands for Build in Self Test. System carries out the test of itself after a certain period of time again and again, that is BIST technique for hardware fault-tolerance. When system detects a fault, it switches out the faulty component and switches in the redundant of it. System basically reconfigure itself in case of fault occurrence.

2. TMR –

TMR is Triple Modular Redundancy. Three redundant copies of critical components are generated and all these three copies are run concurrently. Voting of result of all redundant copies are done and majority result is selected. It can tolerate the occurrence of a single fault at a time.

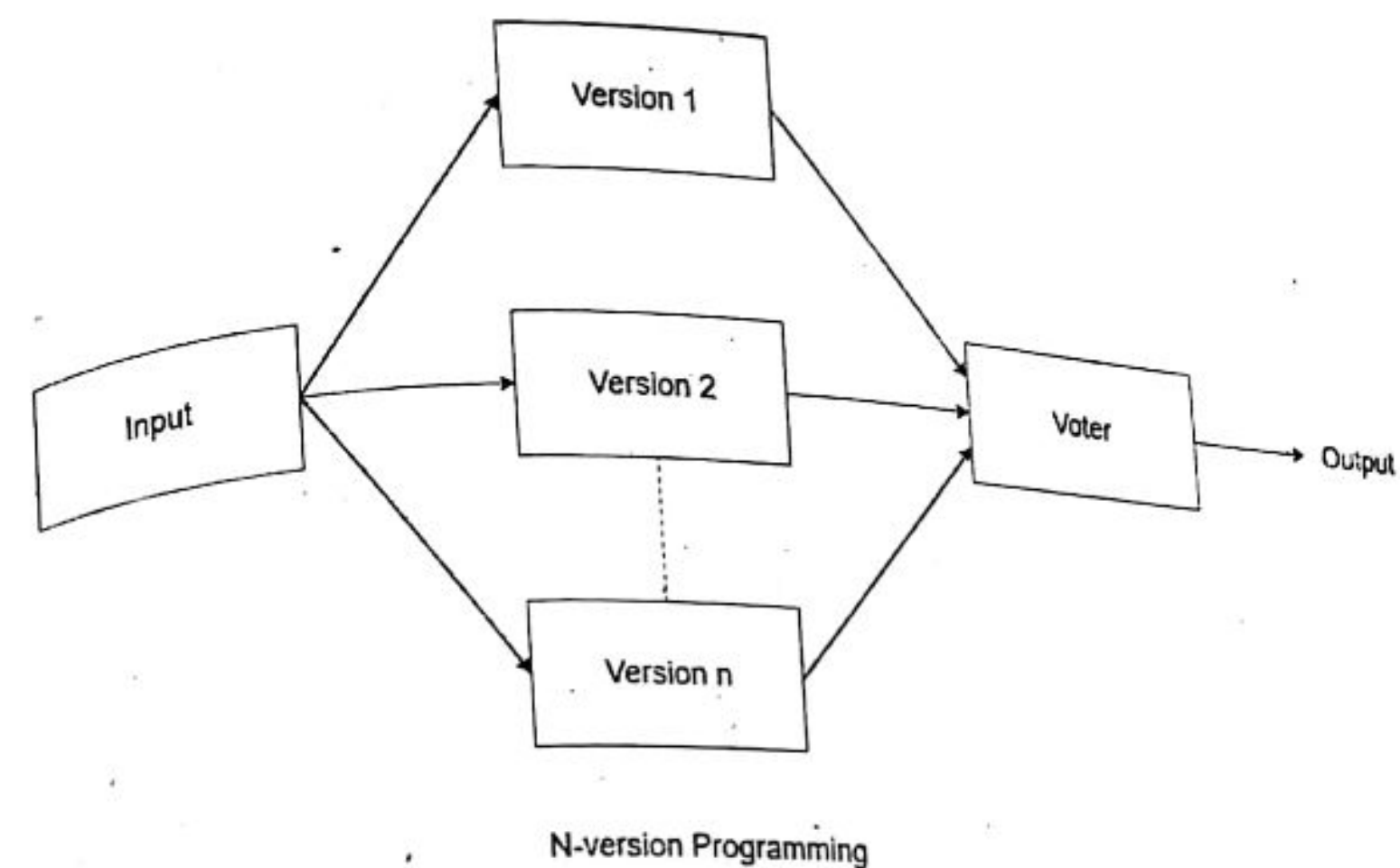


Software Fault-tolerance Techniques:

Software fault-tolerance techniques are used to make the software reliable in the condition of fault occurrence and failure. There are three techniques used in software fault-tolerance. First two techniques are common and are basically an adaptation of hardware fault-tolerance techniques.

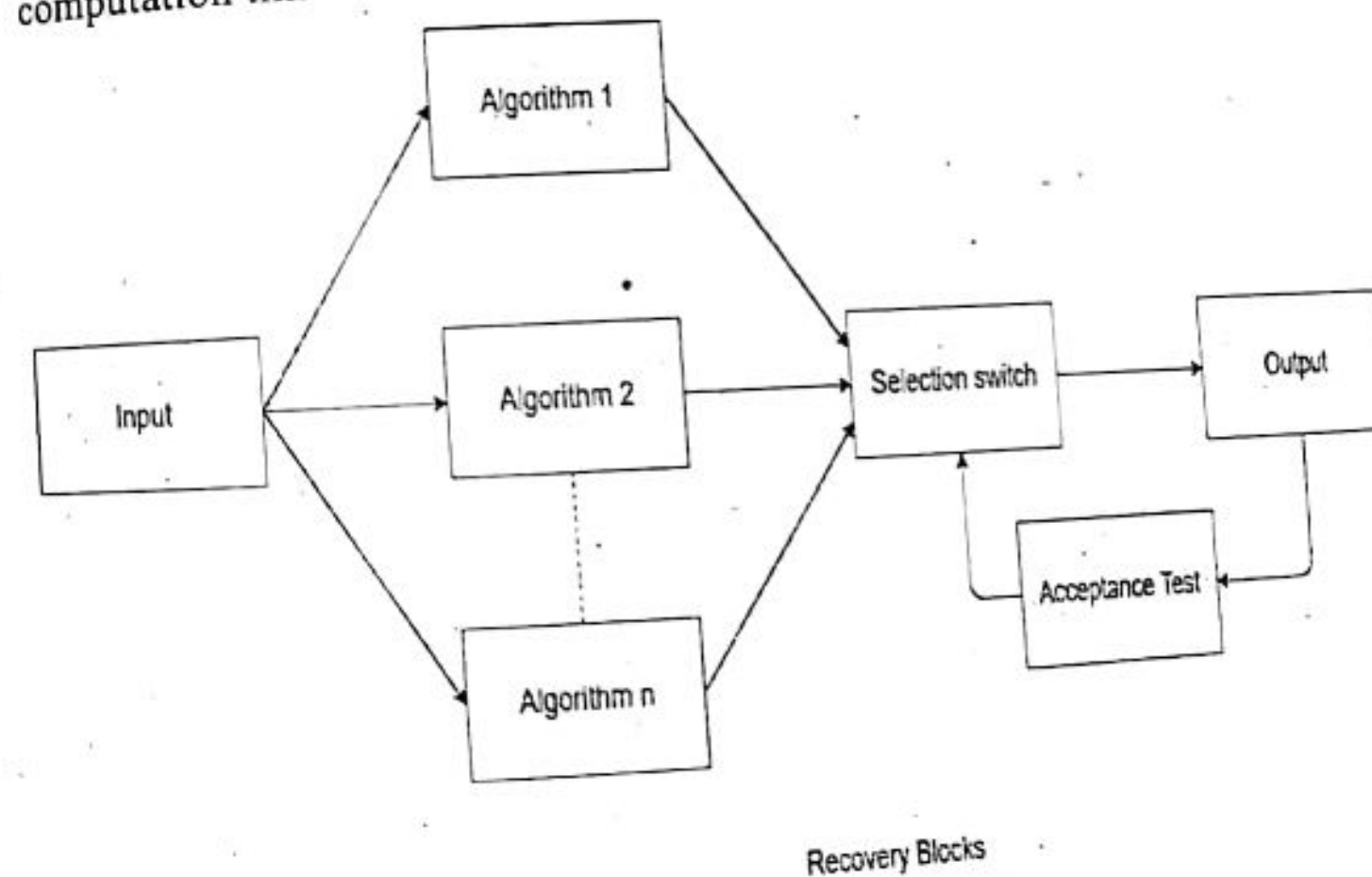
1. N-version Programming –

In N-version programming, N versions of software are developed by N individuals or groups of developers. N-version programming is just like TMR in hardware fault-tolerance technique. In N-version programming, all the redundant copies are run concurrently and result obtained is different from each processing. The idea of n-version programming is basically to get the all errors during development only.



2. Recovery Blocks –

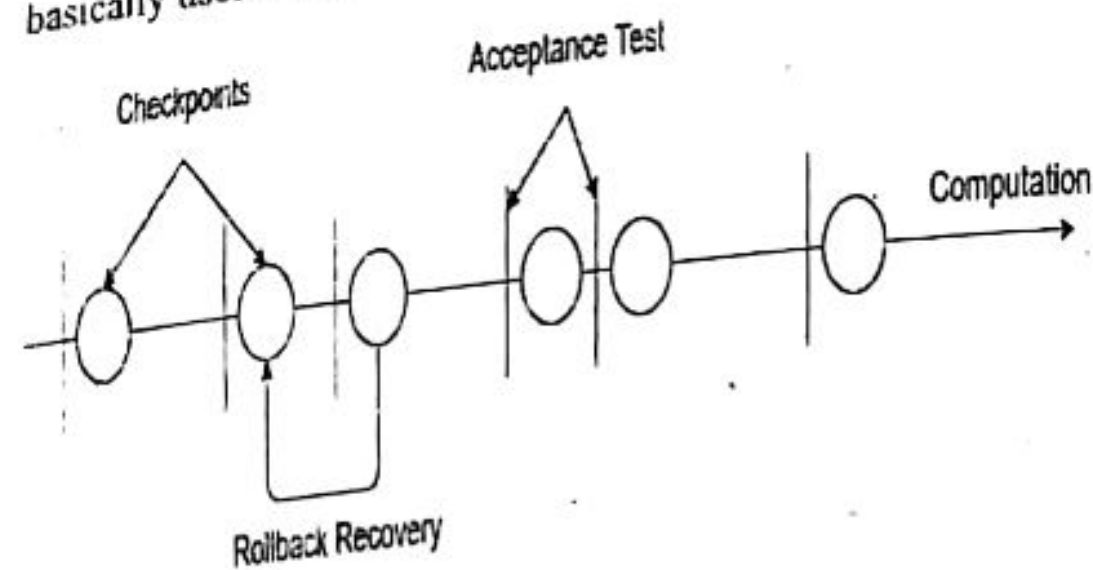
Recovery blocks technique is also like the n-version programming but in recovery blocks technique, redundant copies are generated using different algorithms only. In recovery block, all the redundant copies are not run concurrently and these copies are run one by one. Recovery block technique can only be used where the task deadlines are more than task computation time.



3. Check-pointing and Rollback Recovery –

This technique is different from above two techniques of software fault-tolerance. In this

technique, system is tested each time when we perform some computation. This technique is basically useful when there is processor failure or data corruption.



Deadlock recovery:

Process Termination: To eliminate the deadlock, we can simply kill one or more processes. For this, we use two methods:

Resource Preemption: To eliminate deadlocks using resource preemption, we preempt some resources from processes and give those resources to other processes.

b) MACH

- First Generation micro-kernel
- Builds operating system above minimal kernel
- Kernel provides only fundamental services
- These services basic but powerful enough to be used on, wide range of architectures
- Aids distributed computing and multiprocessing

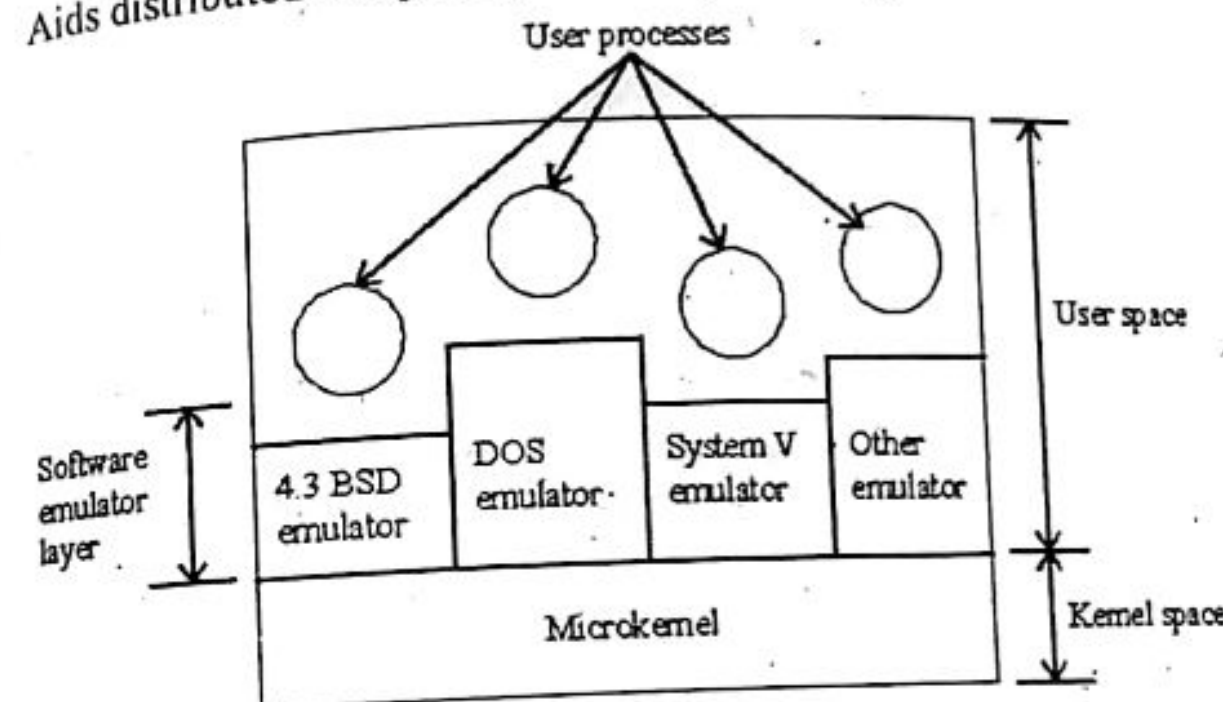


Fig: Mach Architecture

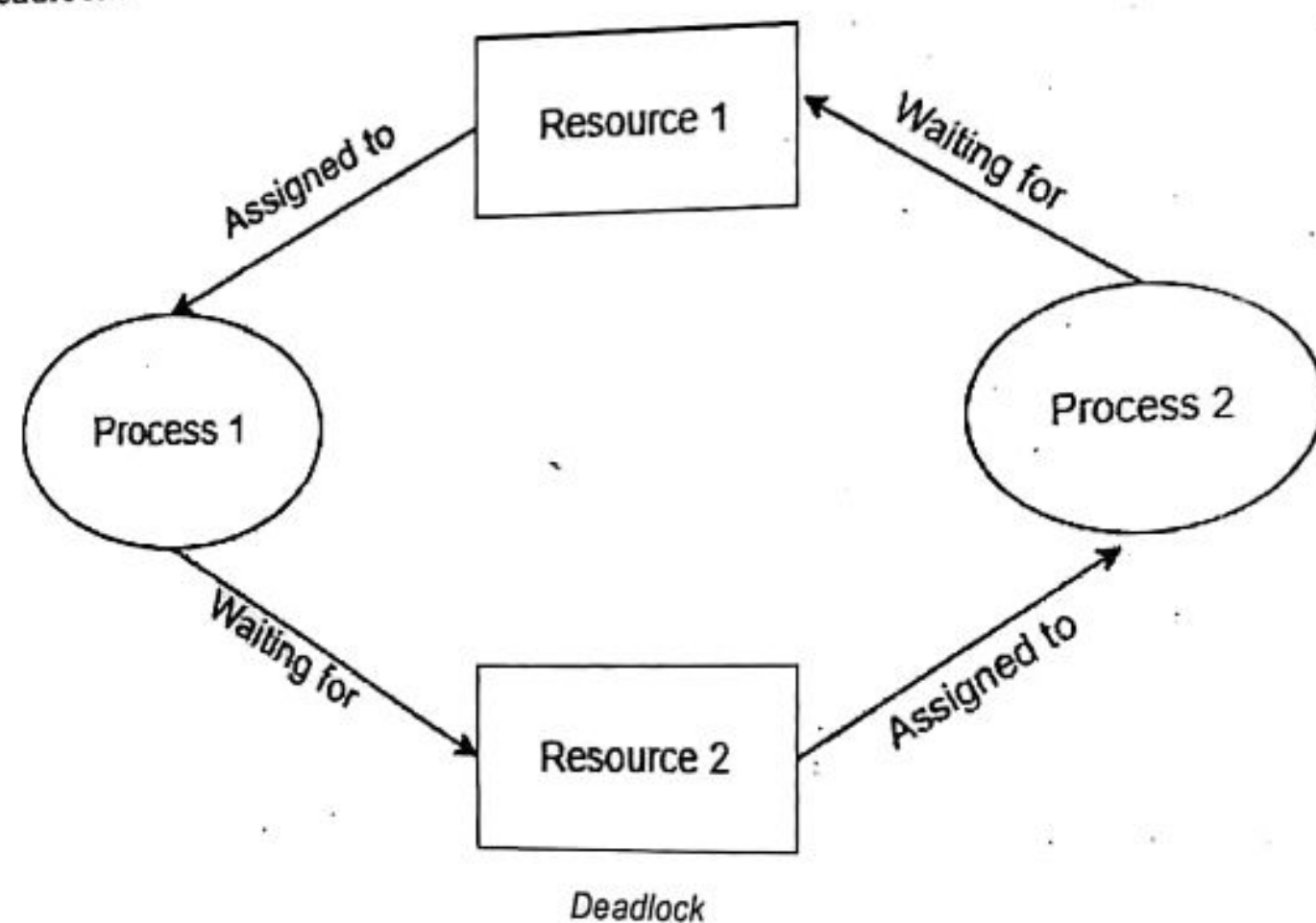
Q10) Write short notes on:

a) Distributed deadlocks and recovery

b) MACH

c) Process Resilience

a) A Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource occupied by some other process. When this situation arises, it is known as Deadlock.



c) Process Resilience



Resilience is the use of strategies for improving a distributed system's availability. One of the primary goals of resilience is to prevent situations where an issue with one microservice instance causes more issues, which escalate and eventually lead to distributed system failure. This is known as a cascading failure.

Process groups: Protect yourself against faulty processes by replicating and distributing computations in a group.

2074 Chaitra

Q1) What is True Distributed System (TDS)? How distributed system can be organized as middleware? Explain

A true distributed system is a piece of software that a collection of independent computer appears to its users as a single coherent system. A distributed system with all the challenges like transparency, scalability, Dependability, Performance, and Flexibility, being solved to its full extent, is called a True Distributed System (TDS).

Eg : internet, intranet, www, airline reservation system, etc

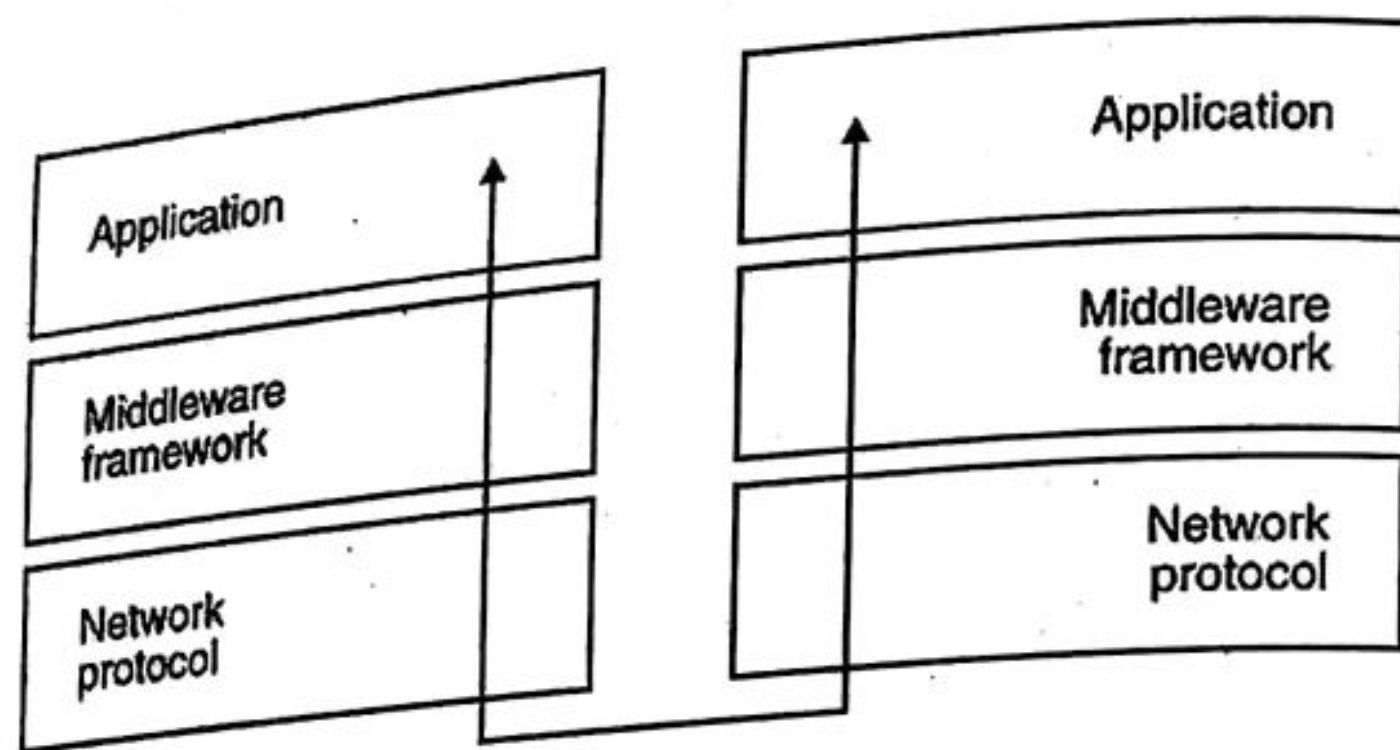
Middleware is a software that lies between an operating system and distributed applications like running applications on it. Distributed middleware helps to connect software with software.

There are three kind of middleware :

- Communication middleware
- Database middleware
- System middleware

Communication middleware

A communication middleware framework provides an environment that enables two applications to set up a conversation and exchange data. Typically, this exchange of data will involve the triggering of one or more transactions along the way. The figure below shows how this middleware framework acts as an intermediary between the application and the network protocol. A communication middleware framework isolates the application developers from the details of the network protocol.



Database middleware

Database middleware is any middleware that facilitates communications with a database, whether from an application or between databases. It supports functionality such as interpolating between software components. It improves database service such as performing scaling or fault tolerance.

System middleware

System middleware is any middleware that helps the communications with the system components and any other applications/components.

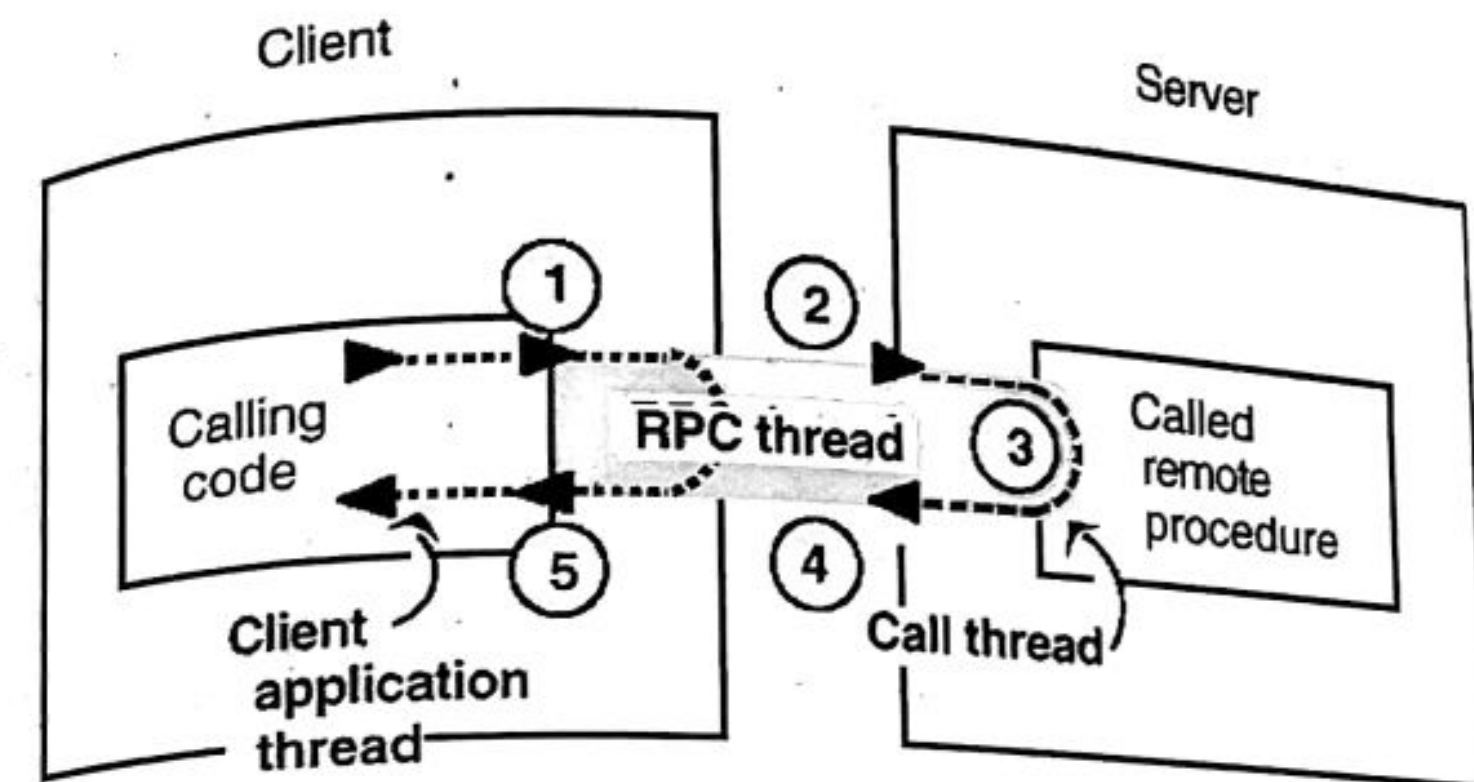
Q2) Define Remote Procedure Call (RPC)? Describe various RPC communications semantics of client server communication in distributed system.

(RPC) is a communication technology that is used by one program to make a request to another program for utilizing its service on a network without even knowing the network's details. A function call or a subroutine call are other terms for a procedure call.

Advantages of RPC:

- RPC provides abstraction i.e message-passing nature of network communication is hidden from the user.
- RPC often omits many of the protocol layers to improve performance. Even a small performance improvement is important because a program may invoke RPCs often.
- RPC enables the usage of the applications in the distributed environment, not only in the local environment.

- With RPC code re-writing / re-developing effort is minimized.
- Process-oriented and thread oriented models supported by RPC.



In distributed system, there must be a reliable client-server communication, but the communication channel may loose, corrupt message so, for handling the communication we should use a reliable transport protocol (eg TCP) or at a application layer.

For reliable communication here are some of technique:

- Use redundant bits to detect bit errors in packets.
- Use sequence no. to detect packet loss.
- Recover from corrupted lost package using ACK
- RPC semantics in case of a client-server communication

Types of failure:

- Client cannot locate server.
- For this RPC-system informs the client of failure.
- Server crashes after receiving a request.
- Client requests/server response lost.
- Client crashes after sending request. #) Server crashes after receiving a request:
- The client cannot tell if the crash occurred before or after request is carried out.

There are 3 possible semantics:

- At least once - keep trying until a reply is received.
- At most once - give up immediately and report back failure.
- Exactly once - desirable but not achievable.

#) lost request/reply message:

Client waits for reply message, resends the request upon time.

Client can safely resend the request for idempotent operations.

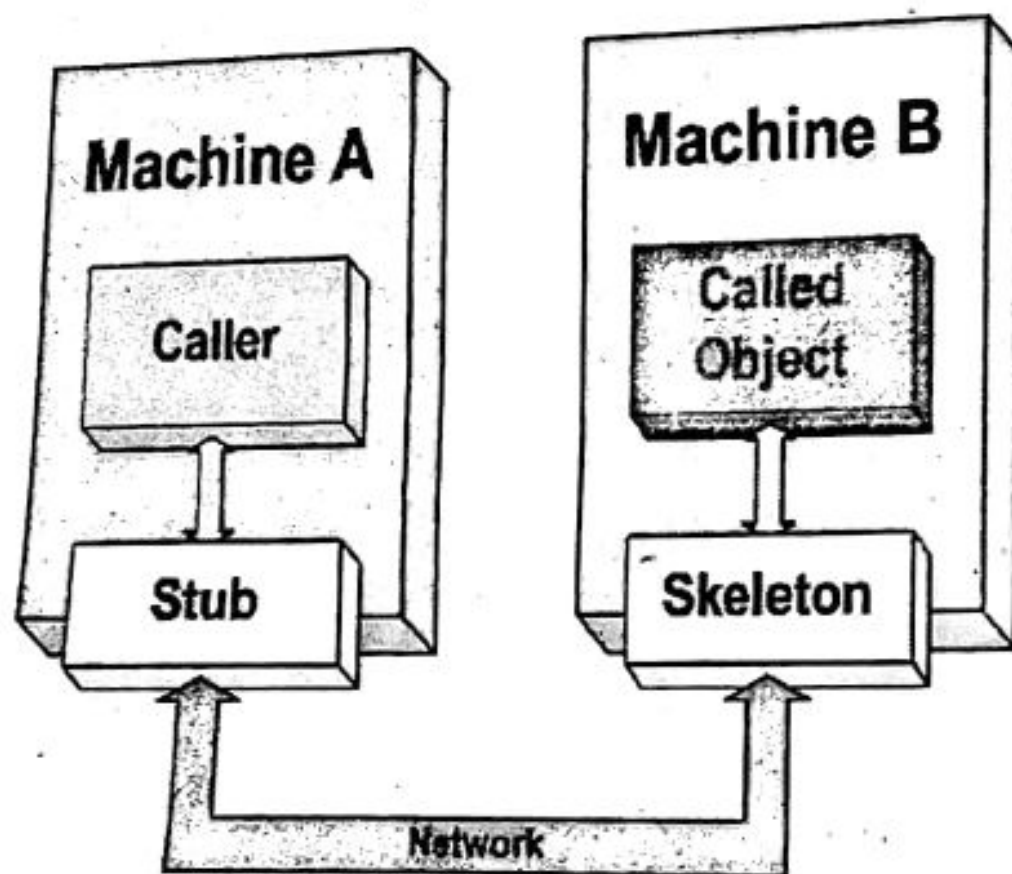
Q3). Mention the role of stub and skeleton in distributed system. Explain the architectural details of the Network File System (NFS).

A **stub** for a remote object acts as a client's local representative or proxy for the remote object. The caller invokes a method on the local stub which is responsible for carrying out the method call on the remote object. In **RMI**, a stub for a remote object implements the same set of remote interfaces that a remote object implements.

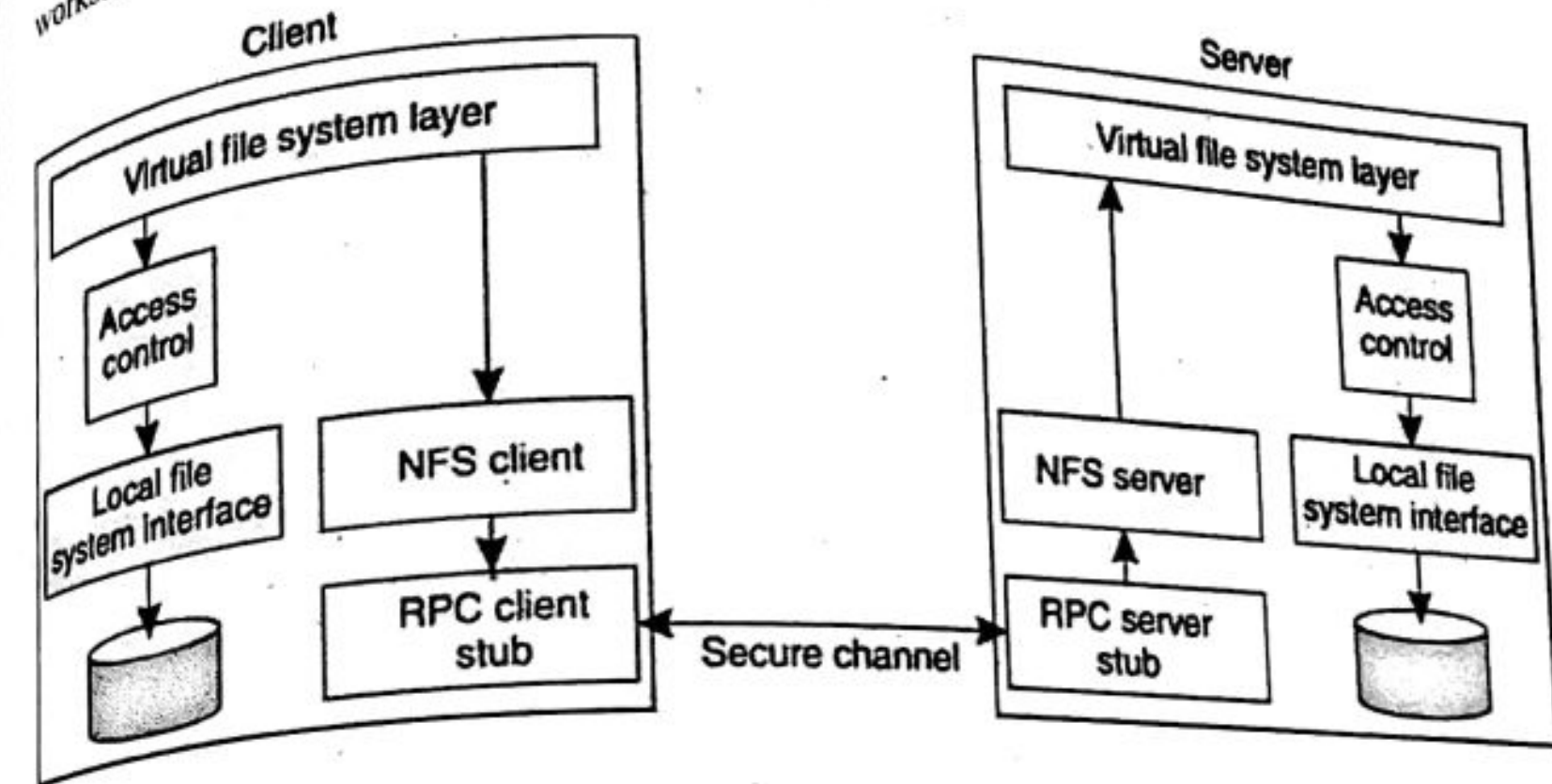
The skeleton is responsible for dispatching the call to the actual remote object implementation.

When a skeleton receives an incoming method invocation, it does the following:

1. reads the parameters for the remote method,
2. invokes the method on the actual remote object implementation.
3. writes and transmits the result (return value or exception) to the caller.



First commercially successful network file system developed by Sun Microsystems for their diskless workstations in 1984 which was designed for robustness and "adequate performance".

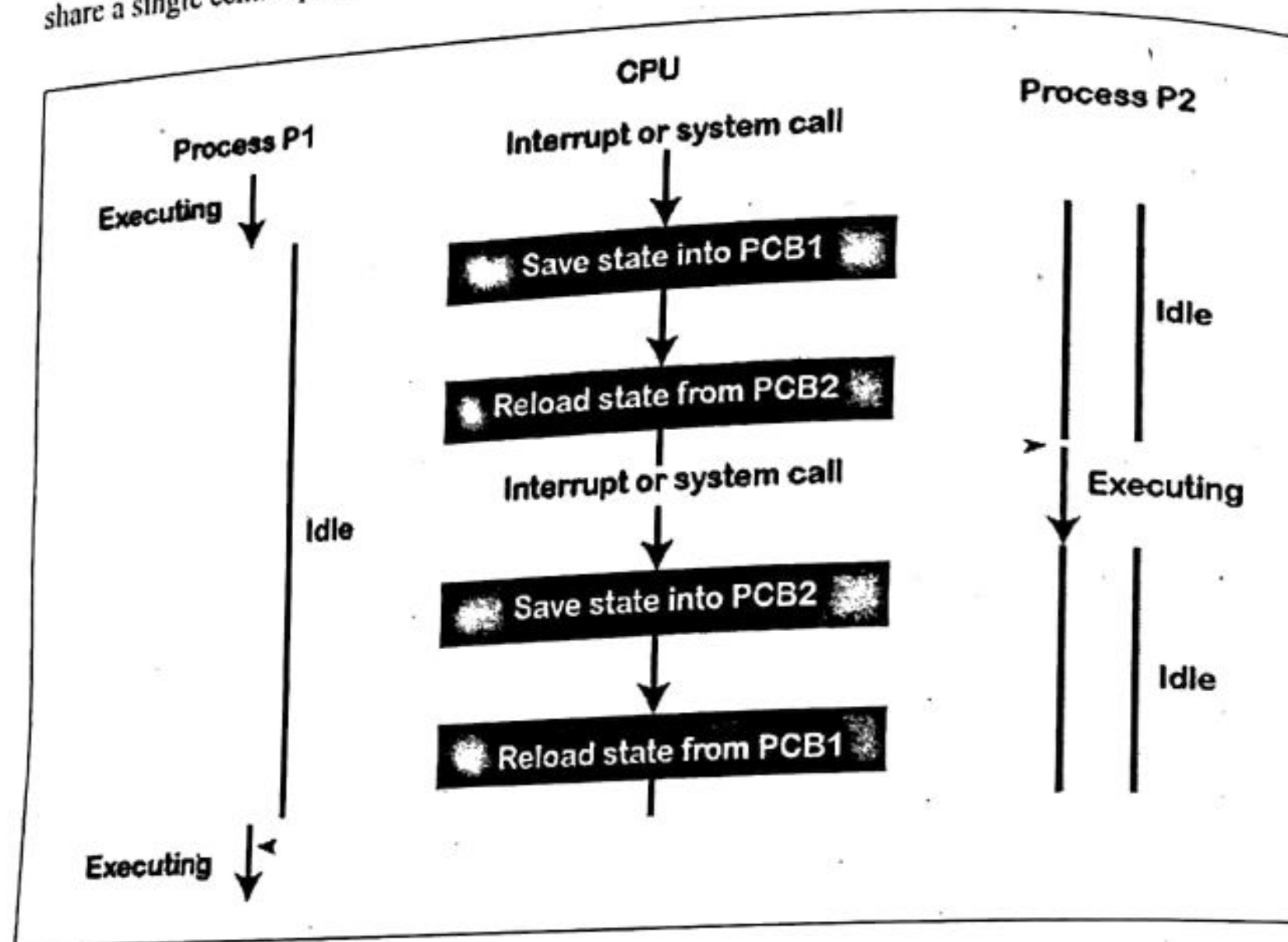


NFS ARCHITECTURE:

1. UNIX filesystem layer - does normal open / read / etc. commands.
2. Virtual file system (VFS) layer -
Gives clean layer between user and filesystem.
Acts as deflection point by using global vnodes.
Understands the difference between local and remote names.
Keeps in memory information about what should be deflected (mounted directories) and how to get to these remote directories.
3. System call interface layer -
Presents sanitized validated requests in a uniform way to the VFS.

Q4) What do you mean by Context Switching in distributed system? How distributed OS is different from network OS?

Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier. This allows multiple processes to share a single central processing unit (CPU), and is an essential feature of a multitasking operating system.



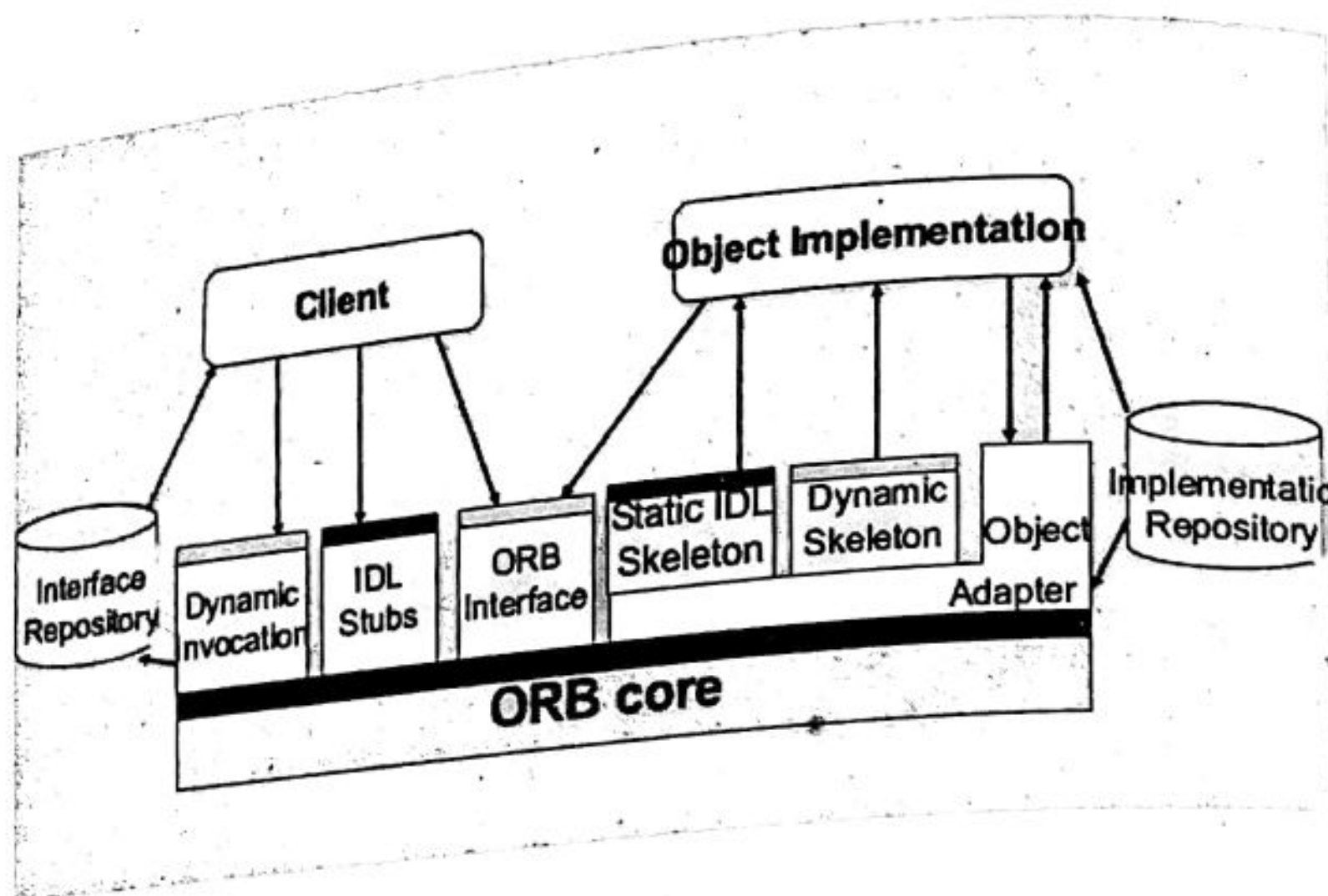
	of files.	messages and shared memory.
3.	Network Operating System is more scalable than Distributed Operating System.	Distributed Operating System is less scalable than Network Operating System.
4.	In Network Operating System, fault tolerance is less.	While in Distributed Operating System, fault tolerance is high.
5.	Rate of autonomy in Network Operating System is high.	While The rate of autonomy in Distributed Operating System is less.
6.	Ease of implementation in Network Operating System is also high.	While in Distributed Operating System Ease of implementation is less.
7.	In Network Operating System, All nodes can have different operating system.	While in Distributed Operating System, All nodes have same operating system.

S.NO	Network Operating System	Distributed Operating System
1.	Network Operating System's main objective is to provide the local services to remote client.	Distributed Operating System's main objective is to manage the hardware resources.
2.	In Network Operating System, Communication takes place on the basis	In Distributed Operating System, Communication takes place on the basis of

Q5). Explain CORBA Invocation methods with its service.

CORBA stands for Common Object Request Broker Architecture. It is a standard defined by the Object Management Group(OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

CORBA provides a platform-independent, language-independent way to write applications that can invoke objects that live across the room or across the planet. More specifically, CORBA is a mechanism in software for normalizing the method-call semantics between application objects residing either in the same address space or remote address space.



The CORBA invocations methods are:

Static Invocation:

It allows a client to invoke requests on an object whose compile time knowledge of server's interface specification is known. For client, object invocation is similar to local method of invocation, which automatically forwarded to object implementation through ORB, object adapter and skeleton. It has low overhead and is efficient at run time.

Dynamic Invocation:

It allows a client to invoke requests on object without having compile time knowledge of object's interface. The object and interface are detected at run time by inspecting the interface repository. The request is then constructed and invocation is performed as it static invocation. It has high overhead.

CORBA SERVICES

Naming service:

It allows clients to find and locate objects based on name.

Trading service:

It allows clients to find and locate objects based on their properties.

Notification service:

It allows objects to notify other objects that some event has occurred.

Transaction Service:

It allows atomic transactions and rollback on failures.

Security Service:

It protects components from unauthorized access or users.

Concurrency control service:

It provides a lock manager that can manage concurrent transactions.

Life cycle service:

It defines conventions for creating, deleting, copying and moving CORBA objects.

Time service:

It provides interfaces for synchronizing time.

6) Define clock synchronization. What is the need of clock synchronization? Explain Cristian's algorithm for physical clock synchronization along with necessary diagram.

Clock synchronization is a topic in computer science and engineering that aims to coordinate otherwise independent clocks.

- Distributed System is a collection of computers connected via the high speed communication network. In the distributed system, the hardware and software components communicate and coordinate their actions by message passing. Each node in distributed systems can share their resources with other nodes. So, there is need of proper allocation of resources to preserve the state of resources and help coordinate between the

several processes. To resolve such conflicts, synchronization is used. Synchronization in distributed systems is achieved via clocks.

The various clocks in the system even if set to a common time value at an instant, drift apart due to unavoidable reasons. Hence some kind of continuous mechanism for synchronization is needed so that they can coordinate and work together to achieve the objectives of the distributed system.

Cristian's Method :

Cristian's algorithm relies on the existence of a time server as a physical clock synchronization algorithm. The time server maintains its clock by using a radio clock or other accurate time source, then all other computers in the system stay synchronized with it. A time client will maintain its clock by making a procedure call to the time server. Variations of this algorithm make more precise time calculations by factoring in network radio propagation time. This method achieves synchronization only if the round-trip times between client and time server are sufficiently short compared to the required accuracy.

Algorithm:

- The process on the client machine sends the request for fetching clock time (time at the server) to the Clock Server at time T_0 .
- 2) The Clock Server listens to the request made by the client process and returns the response in form of clock server time.
- 3) The client process fetches the response from the Clock Server at time T_1 and calculates the synchronized client clock time using the formula given below.

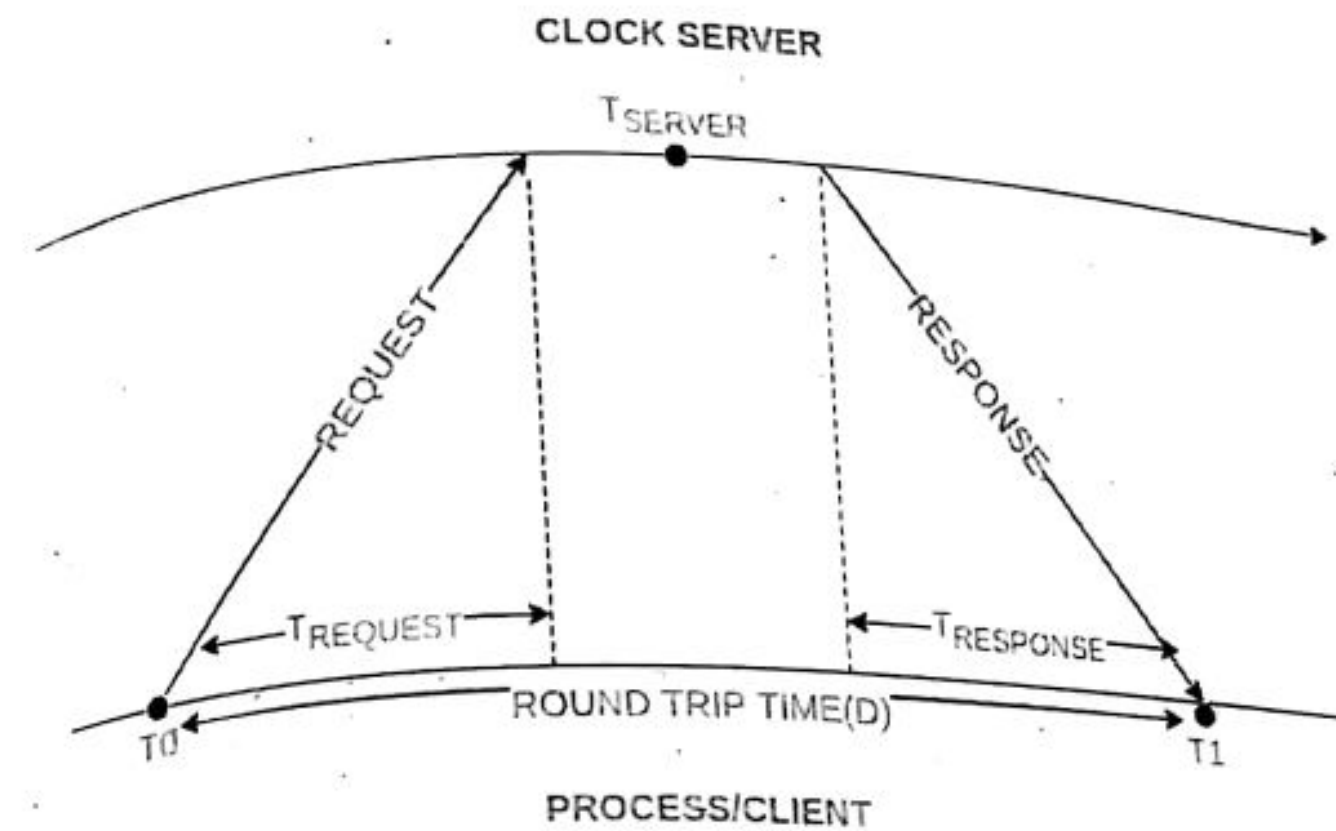
$$T_{CLIENT} = T_{SERVER} + (T_1 - T_0)/2$$

where T_{client} refers to the synchronized clock time,

T_{server} refers to the clock time returned by the server,

T_0 refers to the time at which request was sent by the client process,

T_1 refers to the time at which response was received by the client process.



7) Why Consensus is needed in DS? Explain Bully algorithm with suitable example.

A fundamental problem in distributed computing and multi-agent systems is to achieve overall system reliability in the presence of several faulty processes. This often requires coordinating processes to reach consensus or agree on some data value that is needed during computation.

So, consensus is a procedure to reach a common agreement in a distributed or decentralized multi-agent platform. It is important for the message passing system.

Features:

- It ensures reliability and fault tolerance in distributed systems.
- In the presence of faulty individuals, it ensures correct operations.

Bully Algorithm

This algorithm applies to system where every process can send a message to every other process in the system.

Algorithm – Suppose process P sends a message to the coordinator.

If coordinator does not respond to it within a time interval T , then it is assumed that coordinator has failed.

Now process P sends election message to every process with high priority number.

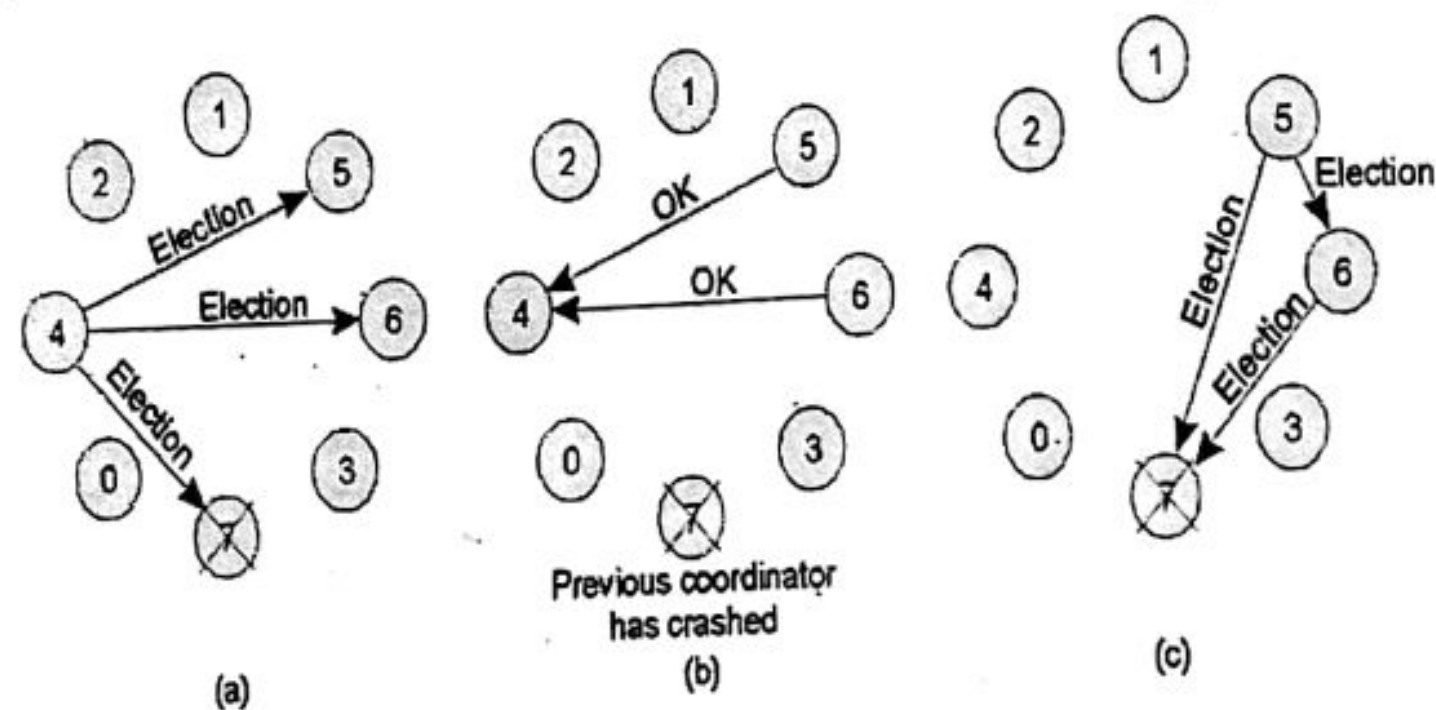
It waits for responses, if no one responds for time interval T then process P elects itself as a coordinator.

Then it sends a message to all lower priority number processes that it is elected as their new coordinator.

However, if an answer is received within time T from any other process Q ,

(I) Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.

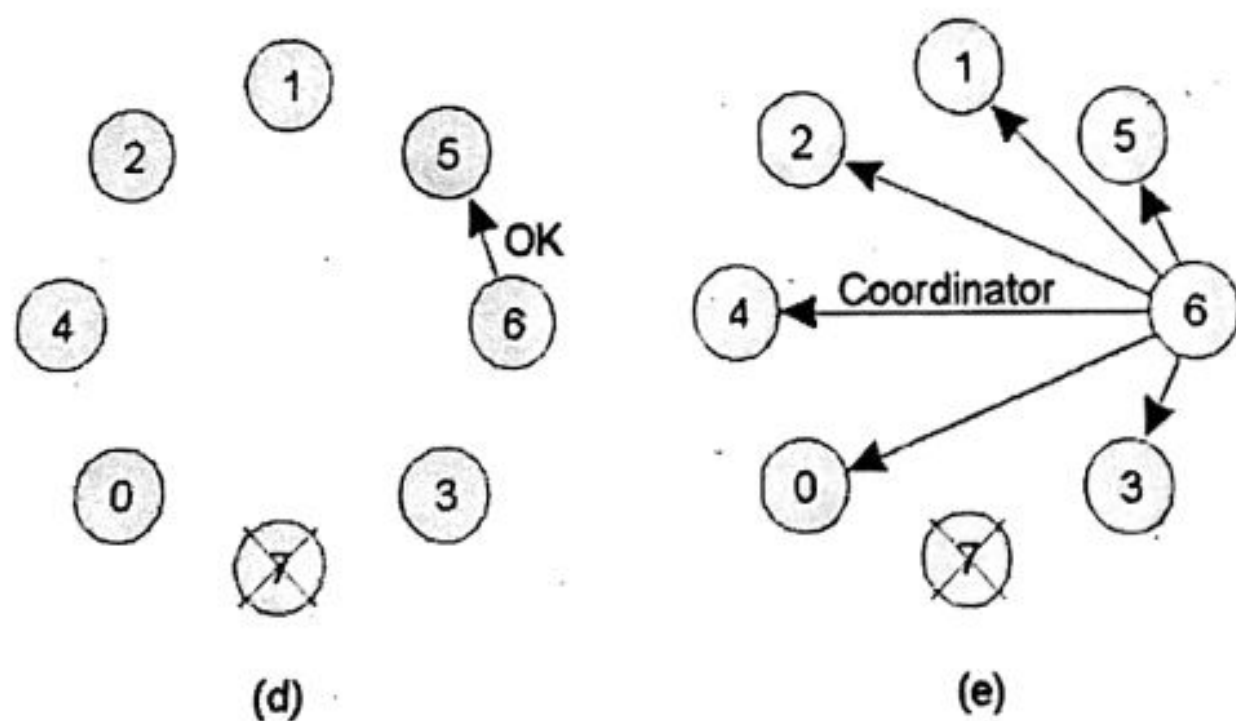
(II) If Q doesn't respond within time-interval T' then it is assumed to have failed and algorithm is restarted.



Process 4 holds an election

Process 5 and 6 respond, telling 4 to stop

Now 5 and 6 each hold an election



Process 6 tells 5 to stop

Process 6 wins and tells everyone

8) What do you mean by object replication? Explain process resilience approach.

Object replication is the process of taking the snapshots of remote objects for sharing purpose in distributed system.

Data in the distributed system contains collection of items. These items are called objects. An object can be file or object generated by programming paradigm. Object replication is the mechanism to form physical replicas of such objects, each stored at single computer and tied to some degree of consistency.

Reason for replication:

- reliability
- performance

Problem: If objects are shared, the concurrent accesses to the shared objects should be managed to guarantee state consistency.

** 2nd part repeated ***

9) What are the benefits and drawbacks of using lock in Transaction Processing? Explain the lost-update problems with suitable example.

Benefits:

Locking is a mechanism to ensure data integrity while allowing maximum concurrent access to data. It is used to implement concurrency control when multiple users access table to manipulate its data at the same time.

Drawback:

Locking has a poor degree of concurrency.

It in fact has no concurrency at all.

In the lost update problem, an update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction.

Occur when two transactions reads the old value of variable and then use it to calculate the new value.

TRANSACTION T	TRANSACTION U
<code>balance = b.getBalance();</code> <code>b.setBalance(balance*1.1);</code> <code>a.withdraw(balance/10);</code>	<code>balance = b.getBalance();</code> <code>b.setBalance(balance*1.1);</code> <code>c.withdraw(balance/10);</code>
<code>balance = b.getBalance();</code> \$200 <code>b.setBalance(balance*1.1);</code> \$220 <code>a.withdraw(balance/10);</code> \$80	<code>balance = b.getBalance();</code> \$200 <code>b.setBalance(balance*1.1);</code> \$220 <code>c.withdraw(balance/10);</code> \$280

Initially A,B,C has \$100,\$200,\$300 respectively. Transaction T transfer the amount from A to B and Transaction U transfer an amount from account C to an account B. In both cases, the amount transfer is calculated to increase the balance by 10%. The net effects on account B of executing the transactions T and U should be to increase the balance of account B by 10% twice, so its final value is \$242.

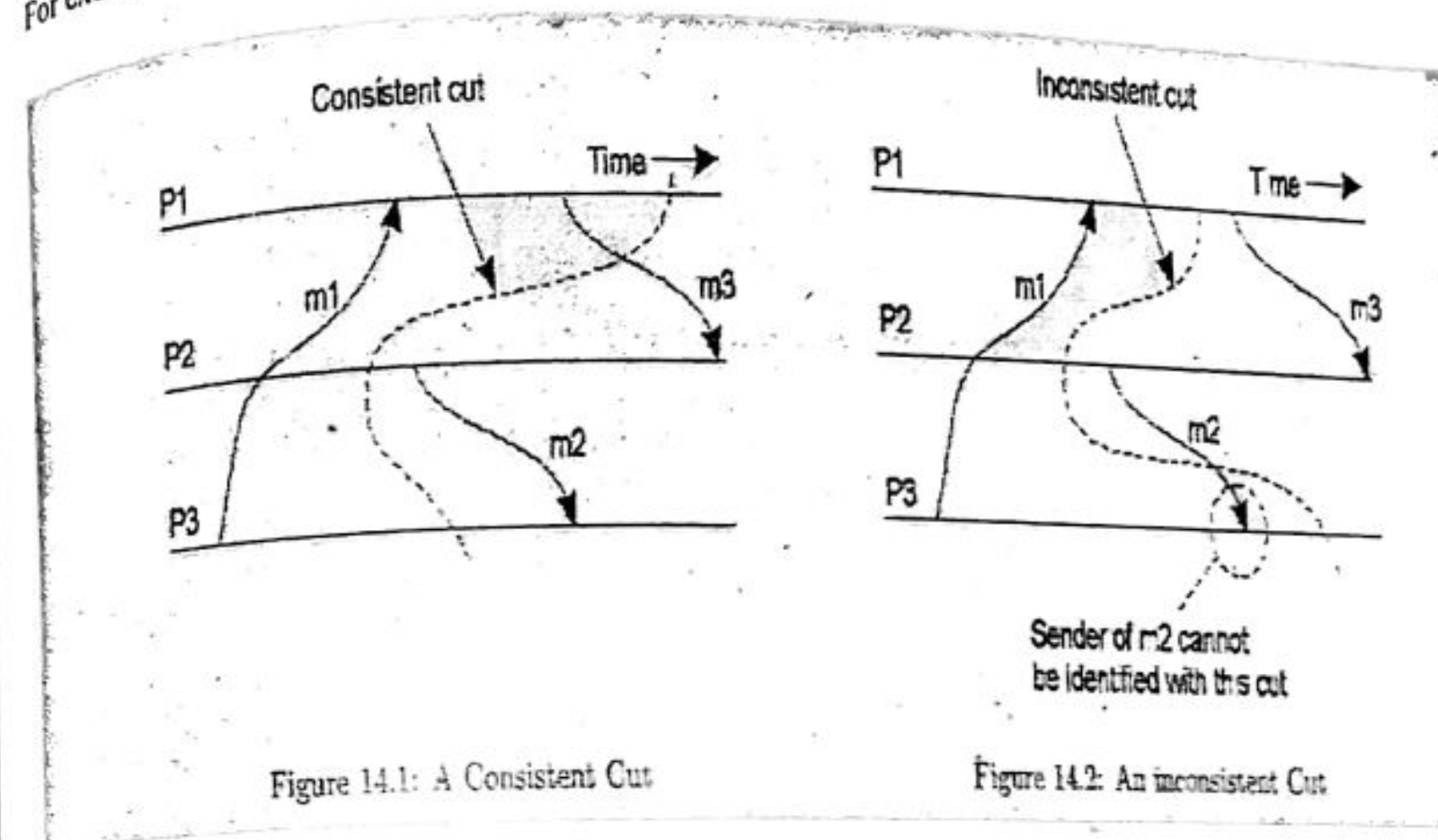
10)What are independent checkpointing and co-ordinated checkpointing? Explain snapshot algorithm used for backward recovery in distributed system.

independent checkpointing requires multiple local checkpoints of each node to be stored on stable storage and is affected by "domino effect". independent checkpoint is also called quasi-synchronous.

Coordinated checkpointing has been found better than independent checkpointing as it is domino-free and has minimum storage and performance overheads. Coordinated checkpointing is attractive due to simple recovery, domino-freeness and optimal stable storage requirement.

A distributed snapshot algorithm captures a consistent global state of a distributed system. A global state can be described by a cut that indicates the time at which each process "checkpoints" its local state and

messages. In the case of a consistent cut C (Fig 14.1), if a message crosses C, its "send" should be before C and its "receive" should be after C. When the system is recovered from a consistent cut, every message will be sent exactly once. If a message's "send" is after C while its "receive" is before C, C becomes inconsistent (Fig 14.2). It will cause problems when the processes are restarted from an inconsistent cut. For example, message m2 can be executed twice in Fig 14.2.



The algorithm assumes that:

- There are no network failures
- Each process communicates with another process using unidirectional point-to-point channels
- There is no message reordering on each channel
- There is a communication path between any two processes

Any process can initiate the algorithm by: 1) checkpointing its local state which contains all necessary information to restart itself, and 2) sending a marker on every outgoing channel. Then for every message from every incoming channel, the process writes a copy to disk, until a subsequent marker is received.

Now every other process may receive markers. If one process receives its first marker, it also checkpoints its local state, sends a marker on every outgoing channel and saves messages from all other incoming channels until a subsequent marker comes. A process finishes when it receives a marker on each incoming channel. It finally collects states of all channels and send them with its own local state to initiator. Multiple

snapshots may be in progress. Each of them is separate and distinguished by tagging the marker with the initiator ID (and sequence number). This algorithm will always stop because: 1) every process sends markers and saves incoming messages only when it receives the first marker; 2) every process stops saving messages from an incoming channel when it receives a subsequent marker from that channel, and such a marker always comes. Intuitively, this algorithm can capture a consistent global state because: 1) no message is recorded/sent twice; 2) no message is lost, since all messages in transit are always between the two markers on the channel and therefore recorded.

11. Write short notes on: (any two)

i) JINI Distributed Event Specification

ii) IDL

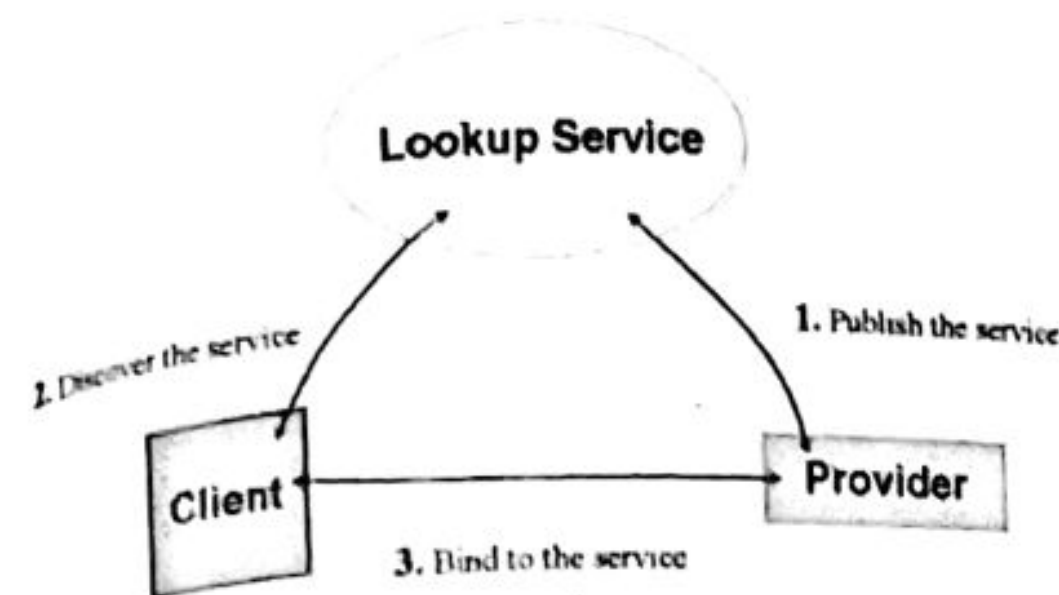
iii) Distributed cut

i) JINI Distributed Event Specification

Jini is actually a Java program that serves as a translator for communication between a computer and other devices on the network. It enables all types of devices to simply connect into impromptu networks, making access to and delivery of new network services as simple as plugging. It enables all types of devices to work together in a community put together without extensive planning, installation, or human intervention.

Jini is a distributed computing network environment that offers, "Network plug and play". Jini is connection technology is based on a simple concept that "device should work together," no driver to find, no operating system issues, no wired cables and connectors.

JINI Service-Oriented Architecture



JINI Process

Discover: find a Lookup service.

Join: send a copy of the service proxy to the Lookup service.

Discover: find a Lookup service.

Lookup: request a service.

Receive a copy of the service proxy.

Advantages

Jini is open-source, meaning that the program code is freely available on the Internet and there are no fees for using it.

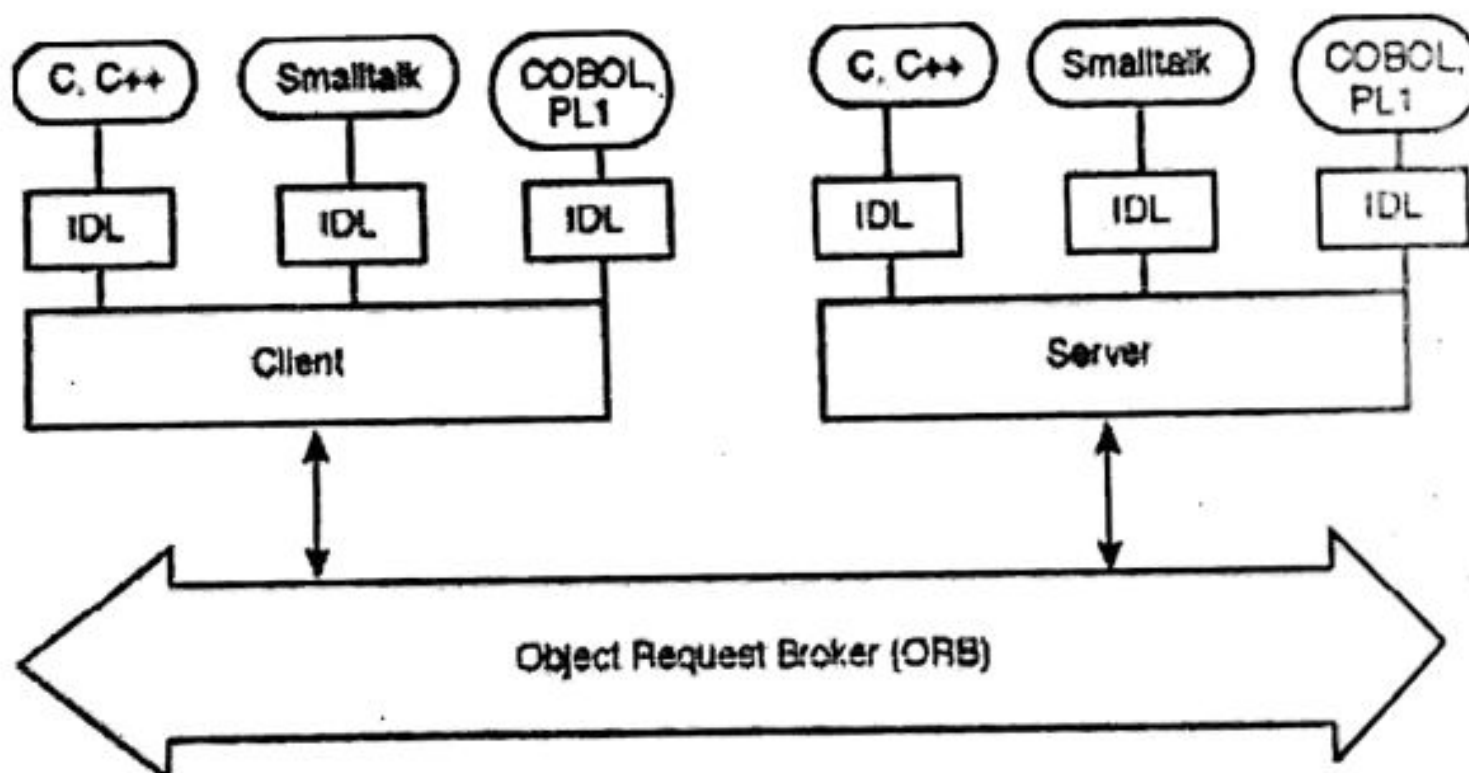
Jini supports an extremely flexible network as Services and Clients can move code to where it is needed.

New services are easy to program in Java, and can be easily added to the network.

ii) IDL

An Interface Definition Language (IDL) is a language that is used to define the interface between a client and server process in a distributed system. Each interface definition language also has a set of associated IDL compilers, one per supported target language. An IDL compiler compiles the interface specifications, listed in an IDL input file, into source code (e.g., C/C++, Java) that implements the low-level communication details required to support the defined interfaces. IDL can also be used to populate an implementation repository, which other programs can use to look up information on an interface at runtime. This is necessary when a program, such as a debugger or interface browser, does not have access to an application's IDL file. One advantage of an interface definition language is that it does not contain any mechanism for specifying computational details. The stubbed-out routines, generated by the IDL compiler, must be filled in with implementation specific details provided by the application developer. Thus, an IDL clearly enforces the separation of a distributed application's interface from its implementation.

IDL defines the modules, interfaces and operations for the applications and is not considered a programming language.



iii) Distributed cut

A cut is a set of cut events, one per node, each of which captures the state of the node on which it occurs.

2073 Chaitra

Q1) Explain the major challenges in designing the distributed system. How interaction model handles the issues in DS?

Ans: The distributed information system is defined as "a number of interdependent computers linked by a network for sharing information among them". A distributed information system consists of multiple autonomous computers that communicate or exchange information through a computer network.

The major challenges in designing the distributed system are:

1. Heterogeneity: Heterogeneity is applied to the network, computer hardware, operating system, and implementation of different developers. A key component of the heterogeneous distributed system client-server environment is middleware. Middleware is a set of services that enables application and end-user to interact with each other across a heterogeneous distributed system.

2. Openness: The openness of the distributed system is determined primarily by the degree to which new resource-sharing services can be made available to the users.

3. Security: Security of information system has three components Confidentiality, integrity and availability. Encryption protects shared resources, keeps sensitive information secrets when transmitted.

4. Scalability: Scalability of the system should remain efficient even with a significant increase in the number of users and resources connected.

5. Failure handling: Failure handling is difficult in distributed systems because the failure is partial i.e., some components fail while others continue to function.

6. Concurrency: There is a possibility that several clients will attempt to access a shared resource at the same time. Multiple users make requests on the same resources, i.e. read, write, and update.

7. Transparency: Transparency ensures that the distributed system should be perceived as a single entity by the users or the application programmers rather than the collection of autonomous systems, which is cooperating.

a) Location transparency: The users cannot tell where resources are located

b) Migration transparency: Resources can move at will without changing their names

c) Replication transparency: The users cannot tell how many copies exist.

d) Concurrency transparency: Multiple users can share resources automatically.

e) Parallelism transparency: Activities can happen in parallel without users knowing.

8. Quality of service

9. Reliability

10. Performance

Second part.

Interaction model are for handling time i.e., for process execution, message delivery, clock drifts etc.

Interaction model in Synchronous and Asynchronous distributed systems.

Synchronous distributed systems

Main features:

-Lower and upper bounds on execution time of processes can be set.

-Transmitted messages are received within a known bounded time.

-Drift rates between local clocks have a known bound.

Important consequences: In a synchronous distributed system there is a notion of global physical time (with a known relative precision depending on the drift rate).

Only synchronous distributed systems have a predictable behavior in terms of timing. Only such systems can be used for hard real-time applications.

In a synchronous distributed system, it is possible and safe to use timeouts in order to detect failures of a process or communication link.

Note: It is difficult and costly to implement synchronous distributed systems.

Asynchronous distributed systems

- Many distributed systems (including those on the Internet) are asynchronous. - No bound-on process execution time (nothing can be assumed about speed, load, and reliability of computers). - No bound-on message transmission delays (nothing can be assumed about speed, load, and reliability of interconnections) - No bounds on drift rates between local clocks.

Important consequences:

In an asynchronous distributed system, there is no global physical time. Reasoning can be only in terms of logical time (see lecture on time and state).

Asynchronous distributed systems are unpredictable in terms of timing.

No timeouts can be used.

Note: Asynchronous systems are widely and successfully used in practice.

In practice timeouts are used with asynchronous systems for failure detection.

However, additional measures must be applied to avoid duplicated messages, duplicated execution of operations, etc.

Q 2) What are the needs of event and notification system during the communication among distributed objects? Explain the distributed event notification process in detail.

Ans. Distributed system requires entities which reside in different address spaces potentially on different machines to communicate. Distributed object means that objects reside in separate address spaces and whose methods can be access on remote address space potentially on different machines. The remote method call is issue in an address space separated from the address space where the target object resides. The code issuing the call is refers to as client. The target object is referred to as server object or remote object.

Event: an event occurs at an object of interest as the result of the completion of a method execution. The idea behind the use of events is that one object can react to exchange occurring in another object.

Notification: is an object that contains information about an event. Notifications of events are essentially asynchronous and determined by their receivers.

Distributed event-based systems extend the local event model by allowing multiple objects at different locations to be notified of events taking place at an object.

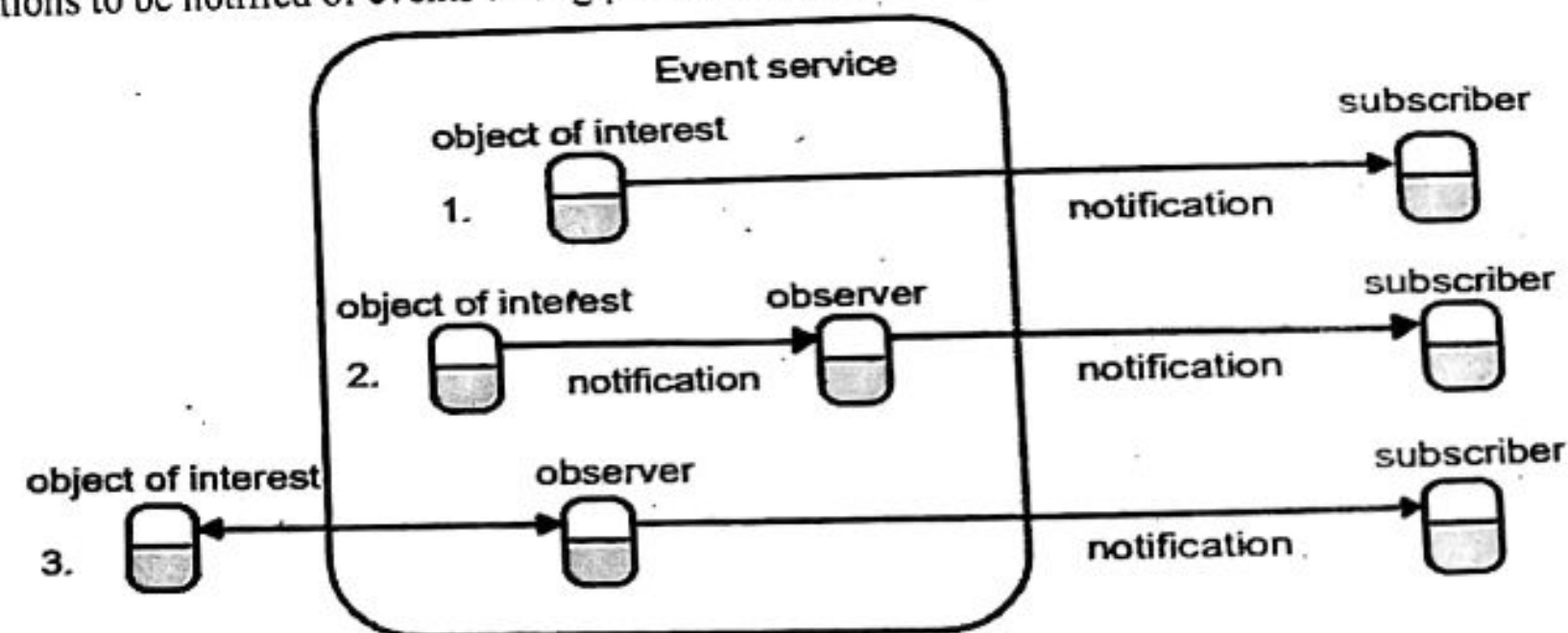


Figure shows an architecture that specifies the roles played by the objects that participate in distributed event-based systems. The main component is an event service that maintains a database of published events and of subscribers' interests.

The roles of the participating objects are as follows:

The object of interest: an object that experiences changes of state, because of its operations being invoked and is considered as part of the event service if it transmits notifications.

Event: an event occurs at an object of interest as the result of the completion of a method execution.

Notification is an object that contains information about an event. Typically, it contains the type of the event and its attributes.

Subscriber: is an object that has subscribed to some type of events in another object.

Observer objects: to decouple an object of interest from its subscribers.

Publisher: an object that declares that it will generate notifications of particular types of events. A publisher may be an object of interest or an observer.

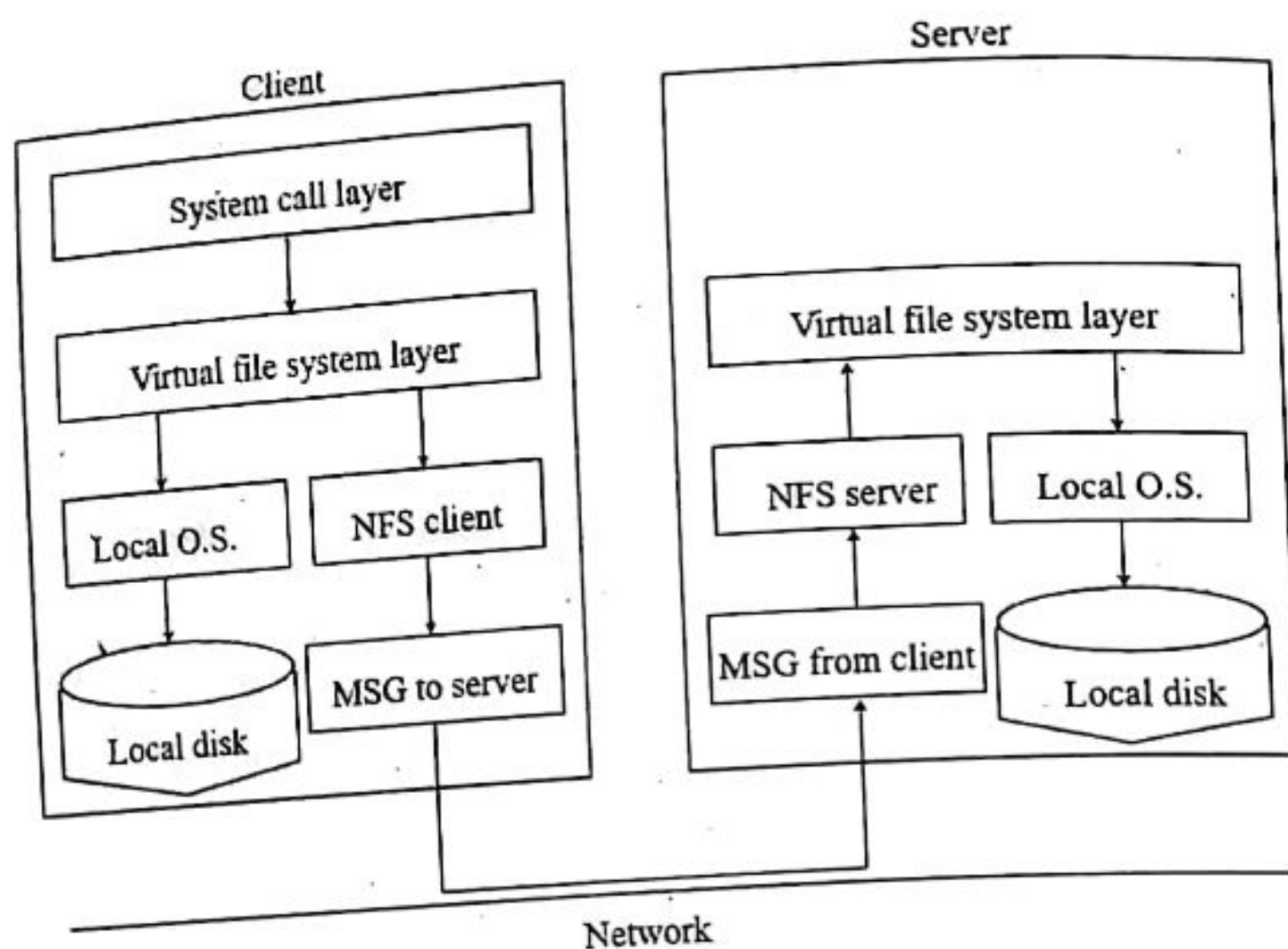
Three cases: An object of interest inside the event service without an observer. It sends notifications directly to the subscribers. An object of interest inside the event service with an observer. The object of interest sends notifications via the observer to the subscribers. An object of interest outside the event service. In this case, an observer queries the object of interest to discover when events occur. The observer sends notifications to the subscribers.

Q 3) What are the major features of SUN NFS? Explain the operation of SUN NFS with its architecture?

Ans: The earliest successful distributed system could be attributed to Sun Microsystems, which developed the Network File System (NFS). NFSv2 was the standard protocol followed for many years, designed with the goal of simple and fast server crash recovery. This goal is of utmost importance in multi-client and single server-based network architectures because a single instant of server crash means that all clients are un-served. The entire system goes down.

The major features of SUN NFS are:

- Machine and Operating System Independence:** The protocols used should be independent of UNIX so that an NFS server can supply files to many different types of clients. The protocols should also be simple enough that they can be implemented on low-end machines like the PC.
- Crash Recovery:** When clients can mount remote filesystems from many different servers it is very important that clients and servers be able to recover easily from machine crashes and network problems.
- Transparent Access:** We want to provide a system which allows programs to access remote files in the same way as local files, without special pathname parsing, libraries, or recompiling. Programs should not need or be able to tell whether a file is remote or local.
- UNIX Semantics Maintained on UNIX Client:** For transparent access to work on UNIX machines. UNIX filesystem semantics have to be maintained for remote files.
- Reasonable Performance:** People will not use a remote filesystem if it is no faster than the existing networking utilities, such as rcp, even if it is easier to use. Our design goal was to make NFS as fast as a small local disk on a SCSI interface.



Basic Design

The NFS design consists of three major pieces: the protocol, the server side and the client side.

NFS Protocol:

NFS uses a stateless protocol. The NFS protocol is defined in terms of a set of procedures, their arguments and results, and their effects. Remote procedure calls are synchronous, that is, the client application blocks until the server has completed the call and returned the results. This makes RPC very easy to use and understand because it behaves like a local procedure call.

The Server side:

Because the NFS server is stateless, when servicing an NFS request it must commit any modified data to stable storage before returning results. The implication for UNIX based servers is that requests which modify the filesystem must flush all modified data to disk before returning from the call. For example, on a write request, not only the data block, but also any modified indirect blocks and the block containing the inode must be flushed if they have been modified.

The Client Side:

The Sun implementation of the client side provides an interface to NFS which is transparent to applications. To make transparent access to remote files work we had to use a method of locating remote files that does not change the structure of path names. Some UNIX based remote file access methods use pathnames like host: path or /.../host/path to name remote files. This does not allow real transparent access since existing programs that parse pathnames have to be modified.

Q 4) What are the RPC communication semantics? Explain the components of CORBA environment.

Ans: In RPC the caller and callee processes can be situated on different nodes. The normal functioning of an RPC may get disrupted due to one or more reasons mentioned below:

- Call message is lost, or response message is lost
- The callee node crashes and is restarted
- The caller node crashes and is restarted.

In RPC system the call semantics determines how often the remote procedure may be executed under fault conditions. The different types of RPC call semantics are as follows:

a. May-Be Call Semantics

- This is the weakest semantics in which a timeout mechanism is used that prevents the caller from waiting indefinitely for a response from the callee.
- This means that the caller waits until a pre-determined timeout period and then continues to execute.
- Hence this semantics does not guarantee the receipt of call message nor the execution. This semantics is applicable where the response message is less important and applications that operate within a local network with successful transmission of messages.

b. Last-Once Call Semantics

- This call semantics uses the idea of retransmitting the call message based on timeouts until the caller receives a response.
- The call, execution and result of will keep repeating until the result of procedure execution is received by the caller.
- The results of the last executed call are used by the caller, hence it known as last-one semantics.
- Last one semantics can be easily achieved only when two nodes are involved in the RPC, but it is tricky to implement it for nested RPCs and cases by orphan calls.

c. Last-of-Many Call Semantics

- This semantics neglects orphan calls unlike last-once call semantics. Orphan call is one whose caller has expired due to node crash.
- To identify each call, unique call identifiers are used which to neglect orphan calls.
- When a call is repeated, it is assigned to a new call identifier and each response message has a corresponding call identifier.
- A response is accepted only if the call identifier associated with it matches the identifier of the most recent call else it is ignored.

d. At-Least-Once Call Semantics

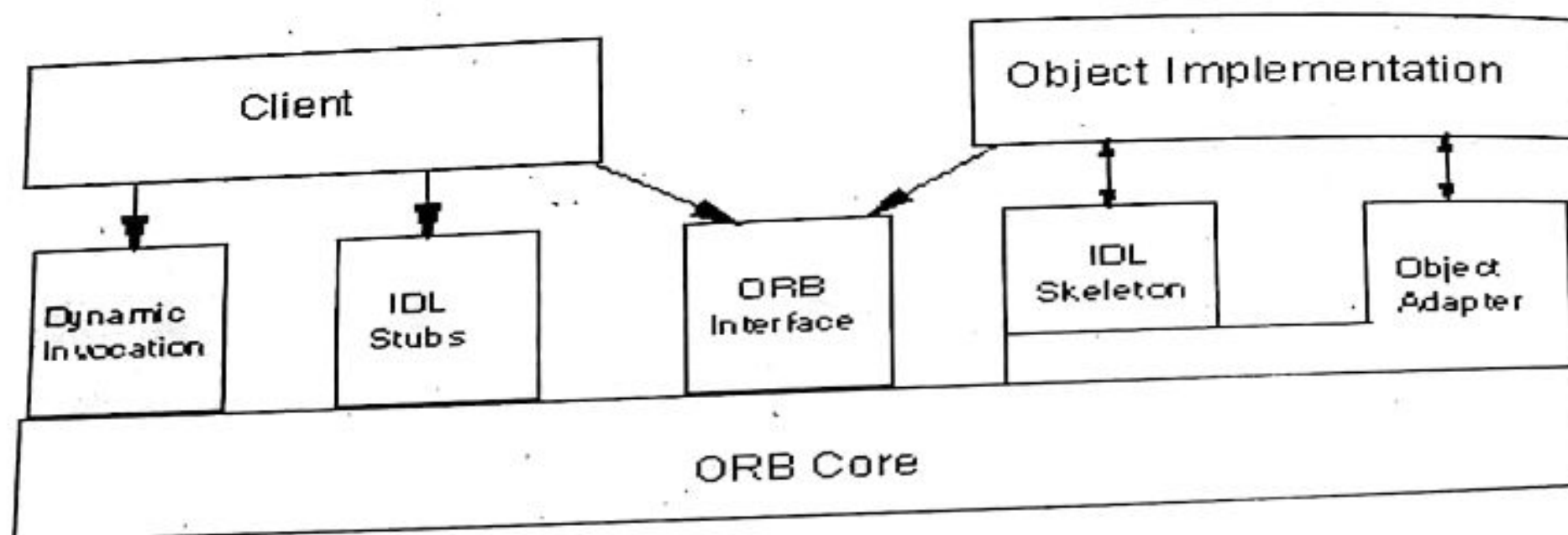
- This semantics guarantees that the call is executed one or more times but does not specify which results are returned to the caller.
- It can be implemented using timeout-based retransmission without considering the orphan calls.

e. Exactly-Once Call Semantics

- This is the strongest and the most desirable call semantics. It eliminates the possibility of a procedure being executed more than once irrespective of the number of retransmitted calls.
- The implementation of exactly-once call semantics is based on the use of timeouts, retransmission, call identifiers with the same identifier for repeated calls and a reply cache associated with the callee.

Second part.

CORBA Architecture



The Common Object Request Broker Architecture (CORBA) describes a messaging mechanism by which objects distributed over a network can communicate with each other irrespective of the platform and language used to develop those objects. CORBA enables collaboration between systems on different operating systems, programming languages, and computing hardware. CORBA uses an object-oriented model although the systems that use the CORBA do not have to be object-oriented.

Dynamic Invocation - This interface allows for the specification of requests at runtime. This is necessary when the object interface is not known at run-time. Dynamic Invocation works in conjunction with the interface repository.

IDL Stub - This component consists of functions generated by the IDL interface definitions and linked into the program. The functions are a mapping between the client and the ORB implementation. Therefore,

ORB capabilities can be made available for any client implementation for which there is a language mapping. Functions are called just as if it was a local object.

ORB Interface - The ORB interface may be called by either the client or the object implementation. The interface provides functions of the ORB which may be directly accessed by the client (such as retrieving a reference to an object.) or by the object implementations. This interface is mapped to the host programming language. The ORB interface must be supported by any ORB.

ORB core - Underlying mechanism used as the transport level. It provides basic communication of requests to other sub-components.

IDL Skeleton Interface - The ORB calls method skeletons to invoke the methods that were requested from clients.

Object Adapters (OA) - Provide how object implementations access most ORB services. This includes the generation and interpretation of object references, method invocation, security, and activation. Requests - The client requests a service from the object implementation. The ORB transports the request, which invokes the method using object adapters and the IDL skeleton.

Object Adapters (OA) are the primary ORB service providers to object implementations. OA has a public interface that is used by the object implementation and a private interface that is used by the IDL skeleton.

Q 5) Why vector clock is important? Explain the types of Distributed cut with examples. How do you perform state recording?

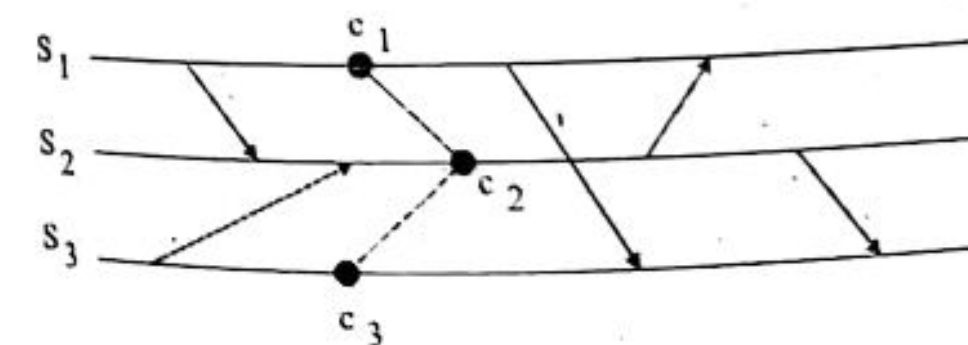
Ans. Vector Clock is an algorithm that generates partial ordering of events and detects causality violations in a distributed system. These clocks expand on Scalar time to facilitate a causally consistent view of the distributed system, they detect whether a contributed event has caused another event in the distributed system. It essentially captures all the causal relationships. This algorithm helps us label every process with a vector (a list of integers) with an integer for each local clock of every process within the system. So, for N given processes, there will be vector/ array of size N.

Advantages of vector clock:

- Vector Clocks are used in distributed systems to determine whether pairs of events are causally correlated.
- Using Vector Clocks, timestamps are created for each event in the system, and their fundamental relationship is determined by comparing those timestamps.

Second part.

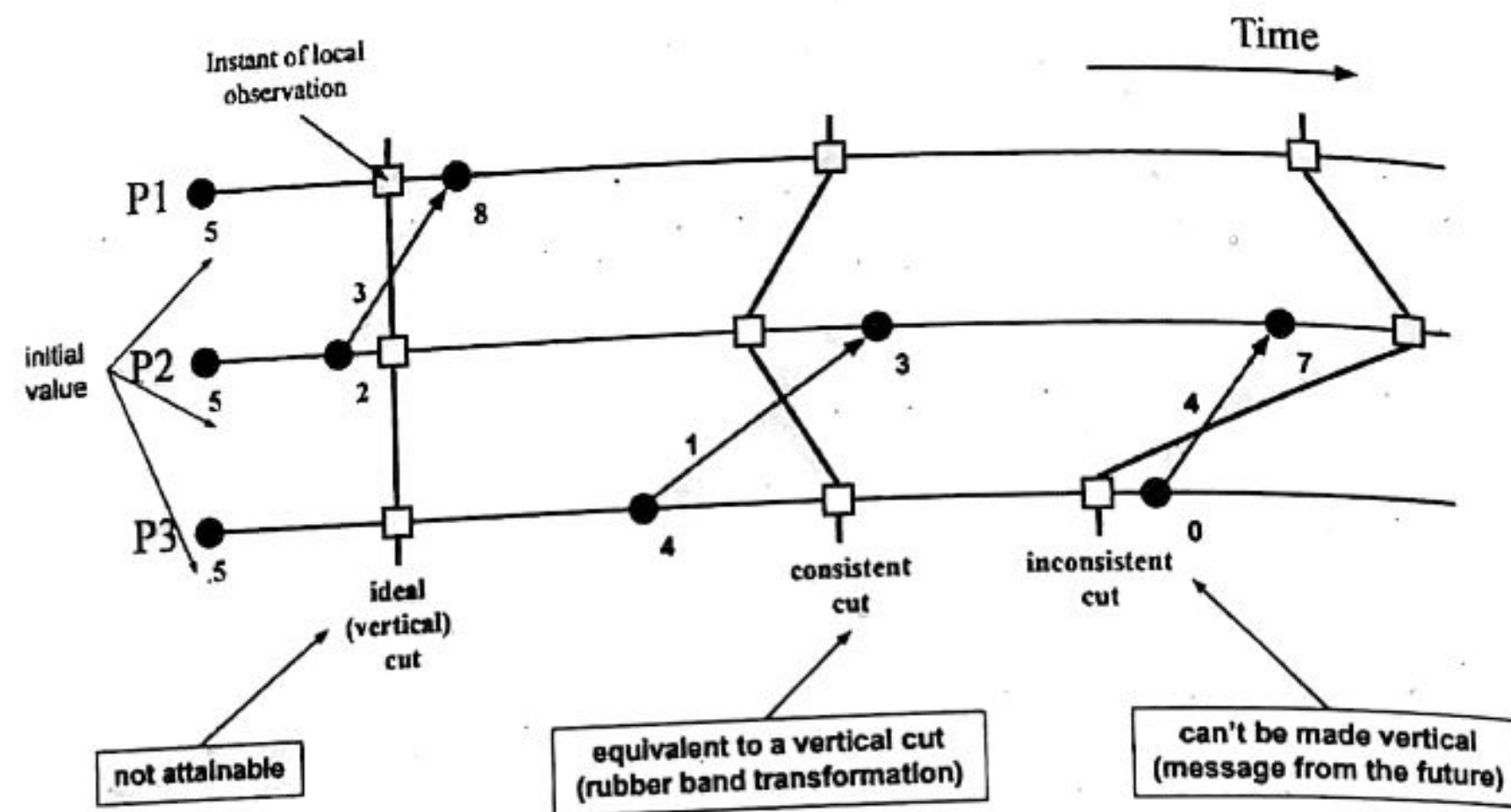
A cut is a set of cut events, one per node, each of which captures the state of the node on which it occurs.



$C = \{c_1, c_2, c_3\}$

Consistent Cut:

A cut $C = \{c_1, c_2, c_3, \dots\}$ is consistent if for all sites there are no events e_i and e_j such that: $(e_i \rightarrow e_j)$ and $(e_j \rightarrow c_j)$ and $(e_i \rightarrow c_i)$.



Inconsistent Cut:
A cut $C = \{c_1, c_2, c_3, \dots\}$ is inconsistent if for all sites there are events e_i and e_j such that:
($e_i \rightarrow e_j$) and ($e_j \rightarrow c_j$) and ($e_i \not\rightarrow c_i$).

Third part.

The global state of a distributed system is the set of local states of each individual processes involved in the system plus the state of the communication channels. Snapshot of the distributed application, i.e., a global picture is useful.

- Checkpointing: can restart distributed application on failure
 - Garbage collection of objects: objects at servers that don't have any other objects (at any servers) with pointers to them
 - Deadlock detection: Useful in database transaction systems
 - Termination of computation: Useful in batch computing systems like Folding@Home, SETI@Home
- The algorithm has three phases: Initiation, Propagation and Termination and works as follows:

1. Initiating a Snapshot

- Process P_i initiates the snapshot.
- The initiator, P_i records its own state and creates the marker message.
- P_i then forwards the marker message to all other processes using the using $N-1$ outbound channels $C_{ij}(j = 1 \text{ to } N \ \& \ j \neq i)$.
- P_i starts recording all incoming messages from channels $C_{ji}(j = 1 \text{ to } N \ \& \ j \neq i)$.

2. Propagating a Snapshot

If a process P_j receives a Marker message on an incoming channel C_{kj} , and if this is the first Marker that P_j is seeing:

- P_j records its own state and mark C_{kj} as empty.
 - P_j then forwards the marker message to all other processes using the $N-1$ outbound channels.
 - P_j starts recording all incoming messages from channels C_{lj} for l not equal to j or k .
- If the process P_j has already seen a Marker message:
- Mark the state of channel C_{kj} as all the messages that have arrived on it since the recording was turned on for C_{kj} .
- #### 3. Terminating a Snapshot
- The algorithm terminates when:
- All processes have received a marker (which implies that process state has been recorded)
 - All processes have received a marker on all the $N-1$ incoming channels (which implies that state of all channels has been recorded)
 - Later, if needed, the central server collects all these partial states to build the global snapshot.

Q 6) Why election algorithm is important? Explain bully algorithm with example.

Ans: Election algorithms choose a process from group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor. Election algorithm basically determines where a new copy of coordinator should be restarted. Election algorithm assumes that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has highest priority number. Then this number is sent to every active process in the distributed system.

We have two election algorithms for two different configurations of distributed system.

- a. Bully Algorithm
- b. Ring Algorithm

Bully Algorithm

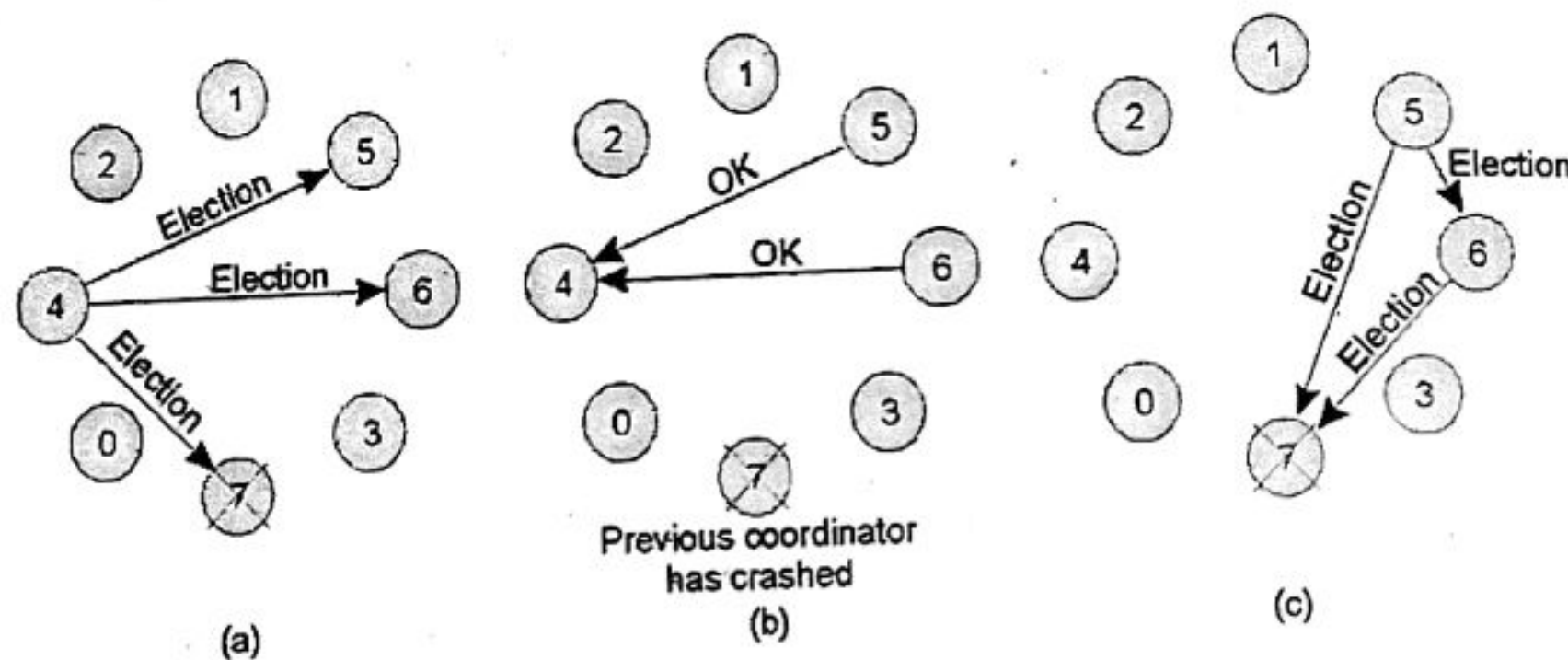
This algorithm applies to system where every process can send a message to every other process in the system.

Algorithm – Suppose process P sends a message to the coordinator.

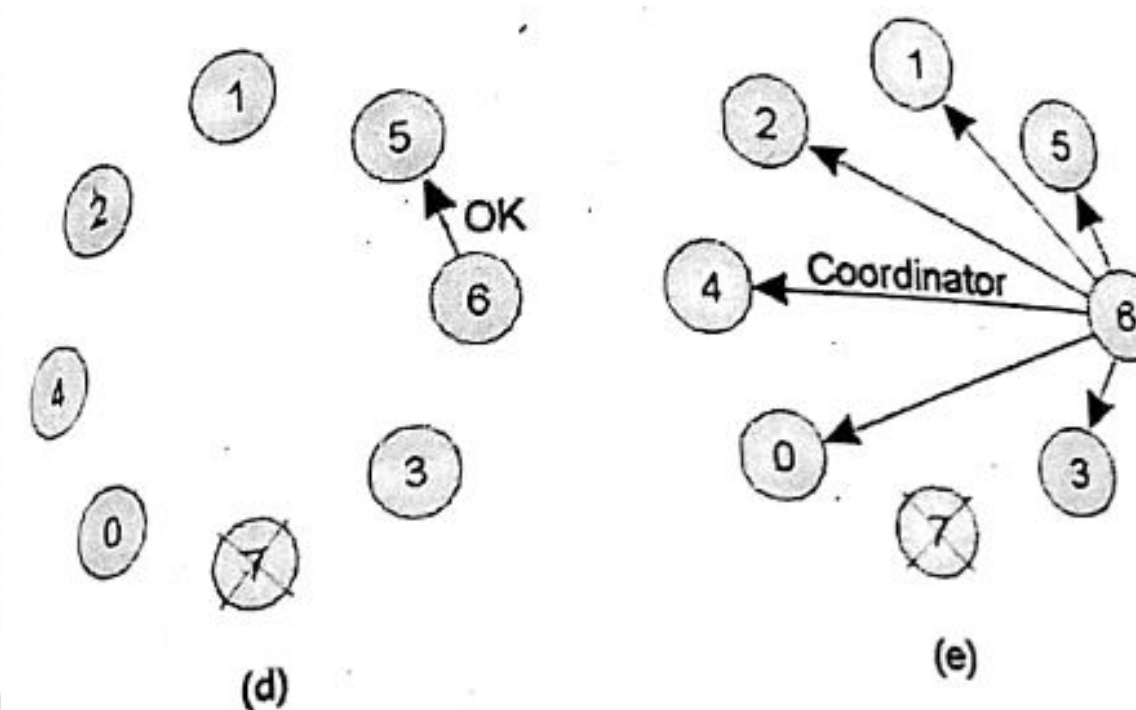
1. If coordinator does not respond to it within a time interval T , then it is assumed that coordinator has failed.

2. Now process P sends election message to every process with high priority number.
3. It waits for responses, if no one responds for time interval T then process P elects itself as a coordinator.
4. Then it sends a message to all lower priority number processes that it is elected as their new coordinator.
5. However, if an answer is received within time T from any other process Q,
 - (I) Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.
 - (II) If Q doesn't respond within time interval T' then it is assumed to have failed, and algorithm is restarted.

Example:



- Process 4 holds an election
- Process 5 and 6 respond, telling 4 to stop
- Now 5 and 6 each hold an election



- Process 6 tells 5 to stop
- Process 6 wins and tells everyone

Q 7) How can you claim that replication is one of the scaling techniques in distribute system? How do you ensure the high available services in DS? Explain with a suitable approach.

Ans:

Replication as Scaling

Keeping copies up to date may require more network bandwidth. If the access-to-update ratio is very low, no need to install a local replica.

- Keeping multiple copies consistent itself is subject to serious scalability problems. Implementing atomicity with a large number of replicas across a large-scale network is inherently difficult when operations are also required to complete quickly. Difficulties come from the need to synchronize all replicas.

In many cases, the real solution is to loosen the consistency constraints.

- To what extent consistency can be loosened depends on the access and update patterns of the replicated data, as well as on the purpose for which those data are used.

Second part.

High availability (HA) can be achieved when systems are equipped to operate continuously without failure for a long duration of time. The phrase suggests that parts of a system have been thoroughly tested and accommodated with redundant components to remain functional when a failure occurs.

1. Middleware HA — load balancer

A few high-availability solutions exist such as load balancing and basic clustering. Load balancing also known as horizontal scaling can be achieved by adding new nodes with identical functionality to existing ones, redistributing the load amongst them all. By exclusively redirecting requests to available servers, reliability and availability can be maintained.

2. Scaling up and down

In companies, high availability is achieved by scaling the servers up or down depending on the application server's load and availability. It is done mostly outside the application at the server level.

3. Implementing multiple application servers

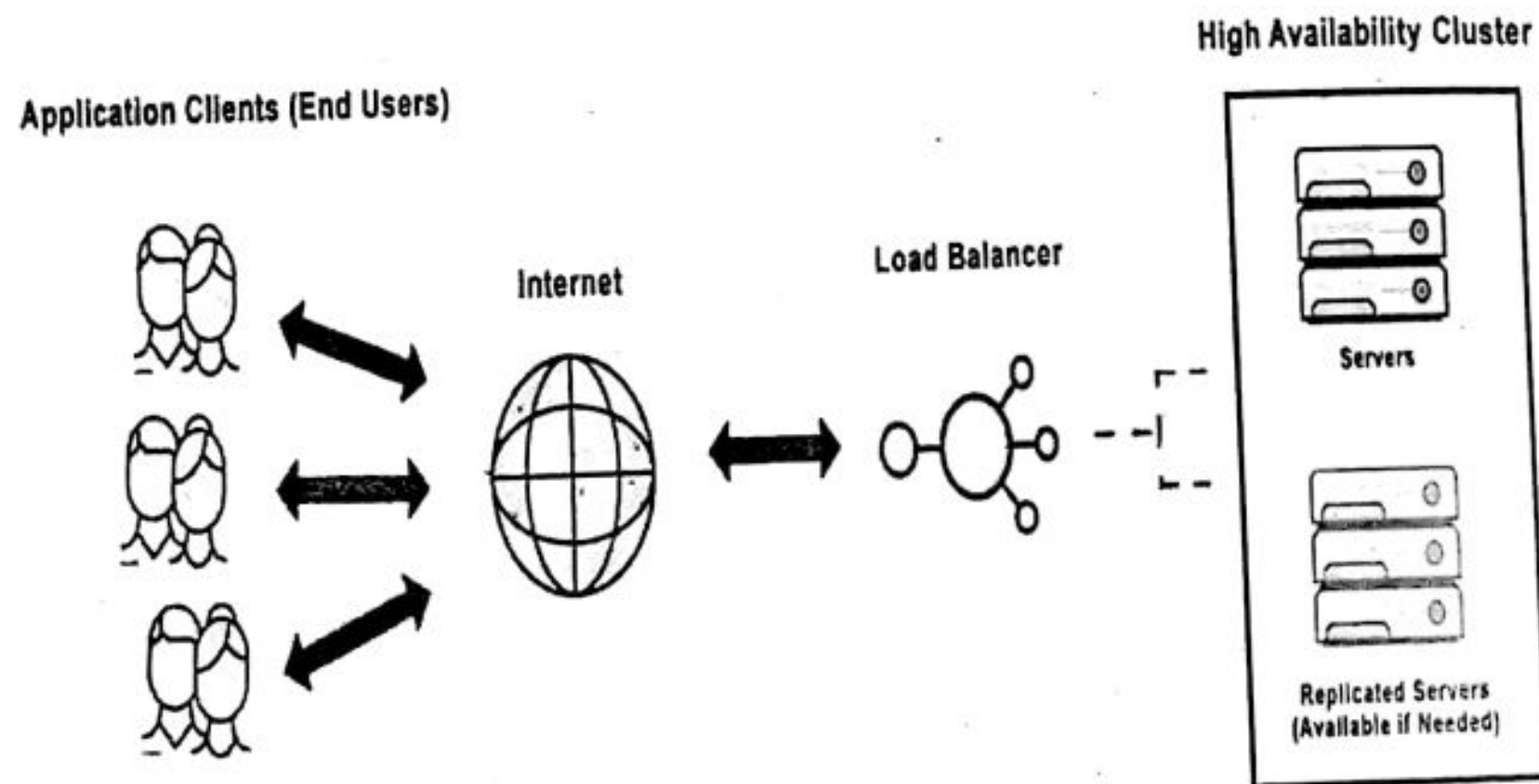
Overburdened servers may crash and cause an outage; it is advisable to deploy applications over multiple servers to keep the applications running all the time. It creates a sense of being always operational.

4. Multi-region deployments

When it comes to cloud environments, systems are deployed in units and are referred to as regions. A region can be defined as a data center, or it may be consisting of a set of data centers located somewhat close to each other. Then there comes a more granular unit inside of the regions and is known as availability zone. So, each availability zone is a single data center within one region.

5. Clustering techniques

Clustering techniques are generally used to improve and increase the performance and availability of a complex system. A cluster is usually designed as a redundant set of services rendering the same set of functionalities and capabilities.



Q 8) Write the Lock compatibility rules for two-phase locking. Describe the methods for concurrency control in distributed system.

Ans: Two-Phase Locking –

A transaction is said to follow the Two-Phase Locking protocol if Locking and Unlocking can be done in two phases.
Growing Phase: New locks on data items may be acquired but none can be released.
Shrinking Phase: Existing locks may be released but no new locks can be acquired.

Note – If lock conversion is allowed, then upgrading of lock (from S(a) to X(a)) is allowed in the Growing Phase, and downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.
 Let's see a transaction implementing 2-PL.

	T ₁	T ₂
1	lock-S(A)	lock-S(A)
2	lock-X(B)
3	Lock-X(C)
4	Unlock(A)
5	Unlock(A)
6	Unlock(B)	Unlock(C)
7
8
9
10

This is just a skeleton transaction that shows how unlocking and locking work with 2-PL. Note for:
 Transaction T₁:
 The growing Phase is from steps 1-3.
 The shrinking Phase is from steps 5-7.
 Lock Point at 3
 Transaction T₂:
 The growing Phase is from steps 2-6.
 The shrinking Phase is from steps 8-9.
 Lock Point at 6

Second part.

Various concurrency control techniques are:

1. Two-phase locking Protocol
2. Time stamp ordering Protocol
3. Multi version concurrency control
4. Validation concurrency control

Time Stamp Ordering Protocol

A timestamp is a tag that can be attached to any transaction or any data item, which denotes a specific time on which the transaction or the data item had been used in any way. A timestamp can be implemented in 2 ways. One is to directly assign the current value of the clock to the transaction or data item. The other is to attach the value of a logical counter that keeps increment as new timestamps are required.

The timestamp of a data item can be of 2 types:

- (i) W-timestamp(X):

This means the latest time when the data item X has been written into.

- (ii) R-timestamp(X):

This means the latest time when the data item X has been read from. These 2 timestamps are updated each time a successful read/write operation is performed on the data item X.

Multi version Concurrency Control:

Multi version schemes keep old versions of data item to increase concurrency.

Multi version 2 phase locking:

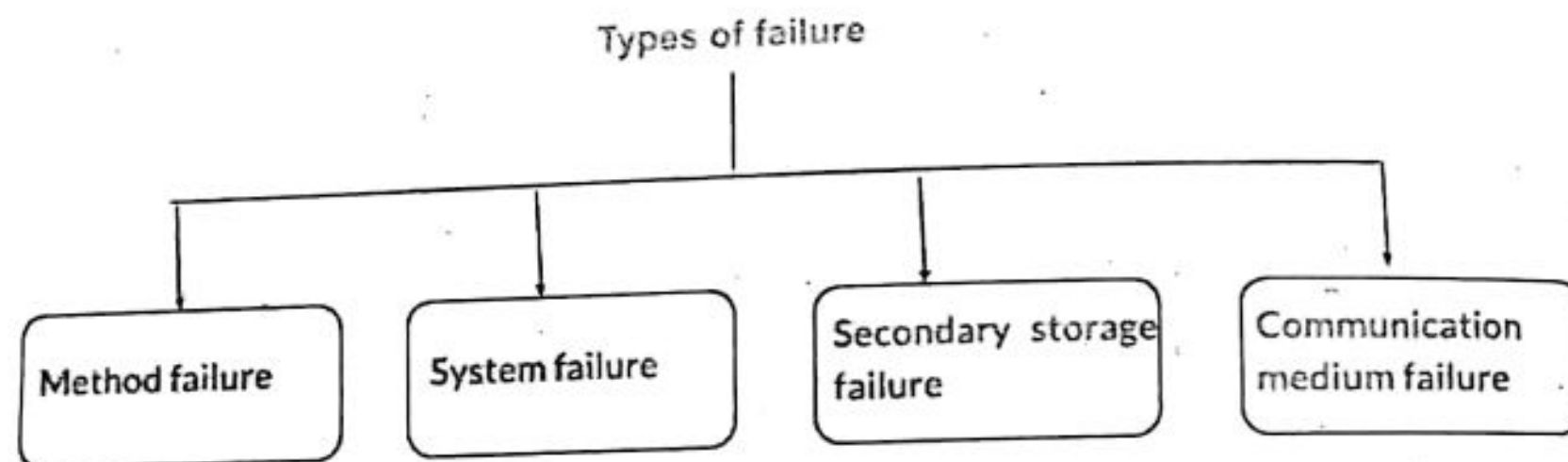
Each successful write results in the creation of a new version of the data item written. Timestamps are used to label the versions. When a read(X) operation is issued, select an appropriate version of X based on the timestamp of the transaction.

Validation Concurrency Control:

The optimistic approach assumes that most of the database operations do not conflict. The optimistic approach requires neither locking nor time stamping techniques. Instead, a transaction is executed without restrictions until it is committed. Using an optimistic approach, each transaction moves through 2 or 3 phases, referred to as read, validation and write.

- (i) During read phase, the transaction reads the database, executes the needed computations and makes the updates to a private copy of the database values. All update operations of the transactions are recorded in a temporary update file, which is not accessed by the remaining transactions.
- (ii) During the validation phase, the transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If the validation test is positive, the transaction goes to a write phase. If the validation test is negative, the transaction is restarted and the changes are discarded.
- (iii) During the write phase, the changes are permanently applied to the database.

Q 9) Explain the types of faults and failures. How do you detect arbitrary faults? Explain with respect to Byzantine failure.



Ans:

1. Method failure:

In this type of failure, the distributed system is generally halted and unable to perform the execution. Sometimes it leads to ending up the execution resulting in an associate incorrect outcome. Method failure causes the system state to deviate from specifications, and also method might fail to progress.

2. System failure:

In system failure, the processor associated with the distributed system fails to perform the execution. This is caused by computer code errors and hardware issues. Hardware issues may involve CPU/memory/bus failure. This is assumed that whenever the system stops its execution due to some fault then the interior state is lost.

3. Secondary storage device failure:

A storage device failure is claimed to have occurred once the keep information can't be accessed. This failure is sometimes caused by parity error, head crash, or dirt particles settled on the medium.

4. Communication medium failure:

A communication medium failure happens once a web site cannot communicate with another operational site within the network. It's typically caused by the failure of the shift nodes and/or the links of the human activity system.

Second part

Byzantine Generals Problem (Lamport's Algorithm)

In this case, the faulty node can also generate arbitrary data, pretending to be a correct one, but making fault tolerant difficult.

Step 1: Every non faulty process i sends v_i to every other process using reliable unicasting.

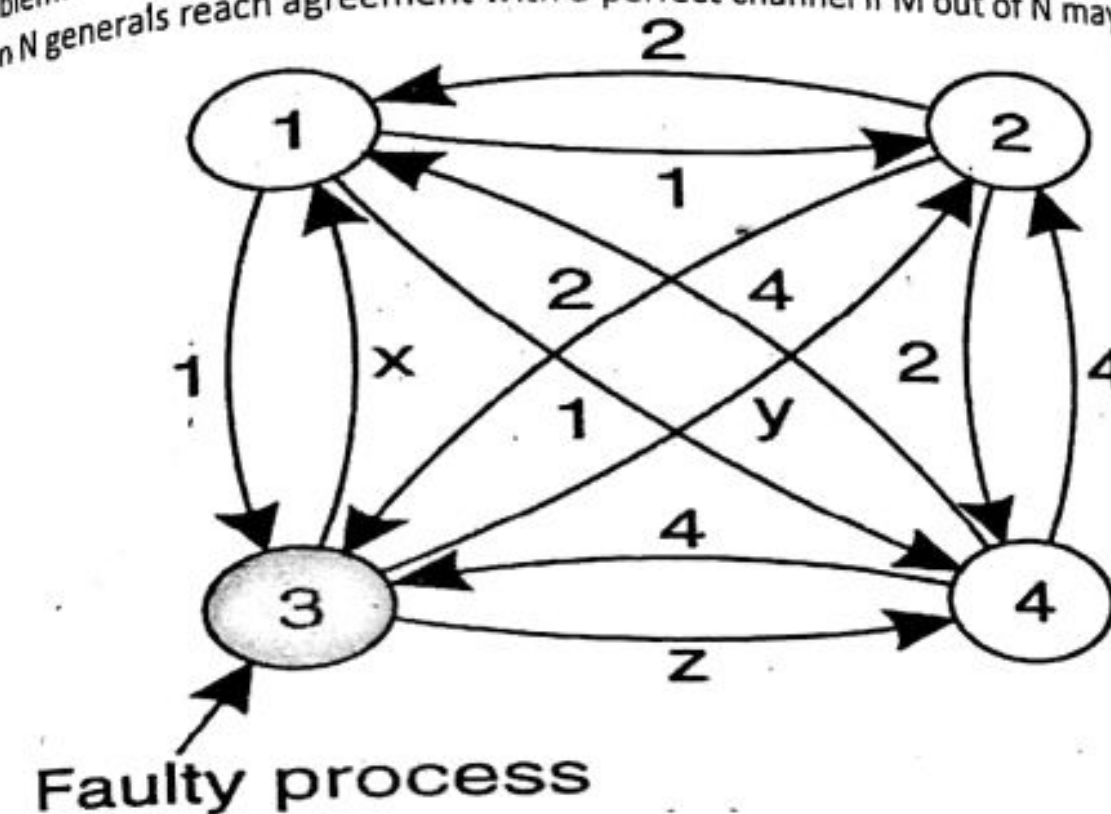
Step 2: The result of the announcements of step 1 are collected in the form of vectors.

Step 3: Every process passing its vector from to every other process. (Process may get $n-1$ vectors).

Step 4: Each process examines the i th element of each of the newly received vectors. If any value has a majority, that value is put into the result vector. Of no value has majority, the corresponding element of the result vector is marked UNKNOWN.

Problem:

Can N generals reach agreement with a perfect channel if M out of N may be traitors?



$N = 4$ (processor)

$M = 1$ (traitor)

The Byzantine agreement problem for three non-faulty and one faulty process.

(a) Each process sends their value to the others.

(b) The vectors that each process assembles based on (a).

(c) The vectors that each process receives in step 3.

1 Got(1, 2, x, 4)

2 Got(1, 2, y, 4)

3 Got(1, 2, 3, 4)

4 Got(1, 2, z, 4)

1 Got

(1, 2, y, 4)

(a, b, c, d)

(1, 2, z, 4)

2 Got

(1, 2, x, 4)

(e, f, g, h)

(1, 2, z, 4)

4 Got

(1, 2, x, 4)

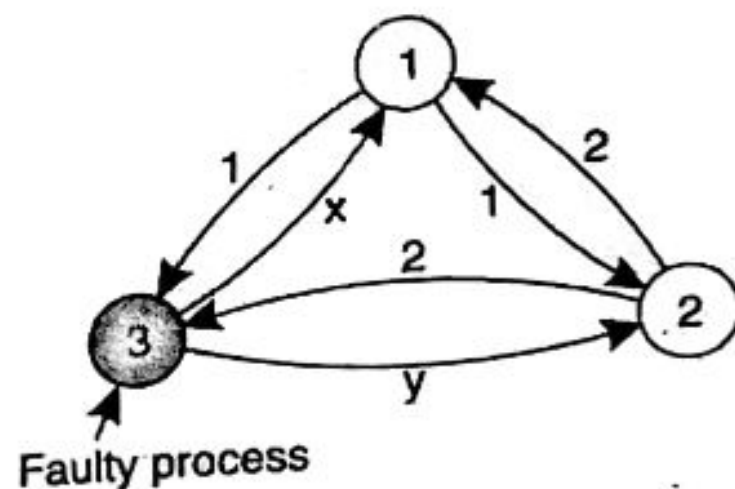
(1, 2, y, 4)

(i, j, k, l)

Conclusion:

With m faulty processes, agreement is possible only if $2m + 1$ processes function correctly.

Suppose $n=3$, $m=1$. Agreement cannot be reached.



1 Got(1, 2, x)
 2 Got(1, 2, y)
 3 Got(1, 2, 3)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

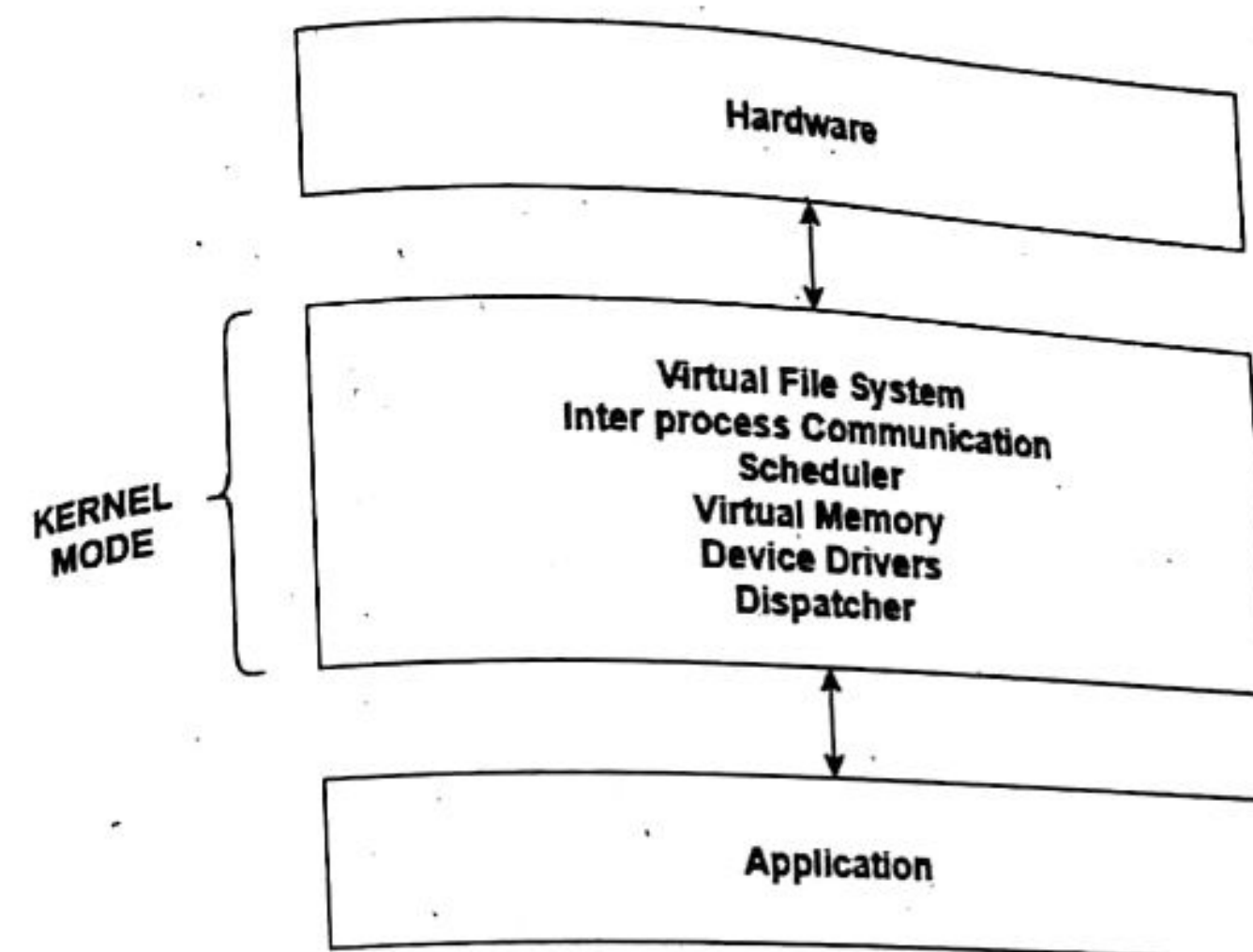
With two correct process and one faulty process. No majority. Cannot reach an agreement.

Q 10) Write short notes on: (Any two)

- a) Monolithic kernel
- b) IDL
- c) Check pointing approach for recovery in AS
- d) JINI

a) Monolithic kernel

A monolithic kernel is an operating system architecture where the entire operating system is working in kernel space. The monolithic model differs from other operating system architectures (such as the microkernel architecture) in that it alone defines a high-level virtual interface over computer hardware. A set of primitives or system calls implement all operating system services such as process management, concurrency, and memory management. Device drivers can be added to the kernel as modules.

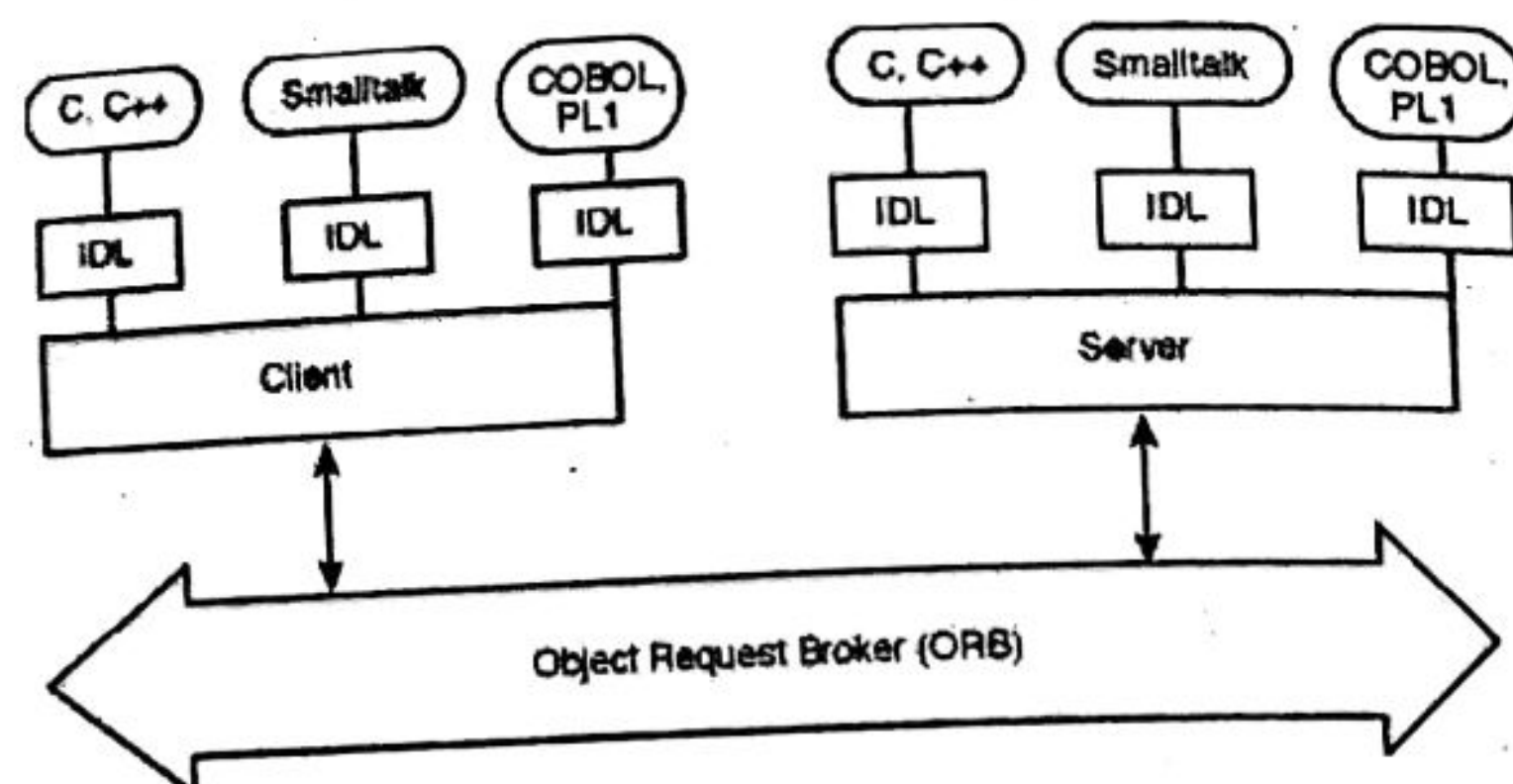


Monolithic Kernel Operating System

b) IDL

-An Interface Definition Language (IDL) is a language that is used to define the interface between a client and server process in a distributed system. Each interface definition language also has a set of associated IDL compilers, one per supported target language. An IDL compiler compiles the interface specifications, listed in an IDL input file, into source code (e.g., C/C++, Java) that implements the low-level communication details required to support the defined interfaces. IDL can also be used to populate an implementation repository, which other programs can use to look up information on an interface at runtime. This is necessary when a program, such as a debugger or interface browser, does not have access to an application's IDL file. One advantage of an interface definition language is that it does not contain any mechanism for specifying computational details. The stubbed-out routines, generated by the IDL compiler, must be filled in with implementation specific details provided by the application developer. Thus, an IDL clearly enforces the separation of a distributed application's interface from its implementation.

-IDL defines the modules, interfaces and operations for the applications and is not considered a programming language.



c) Check pointing approach for recovery in AS

Checkpointing and recovery are two techniques that must be developed hand in hand to enhance the availability of a cluster system. We will start with the basic concept of checkpointing. This is the process of periodically saving the state of an executing program to stable storage, from which the system can recover after a failure. Each program state saved is called a checkpoint. The disk file that contains the saved state is called the checkpoint file. Although all current checkpointing software saves program states in a disk, research is underway to use node memories in place of stable storage to improve performance.

Checkpointing techniques are useful not only for availability, but also for program debugging, process migration, and load balancing. Many job management systems and some operating systems support checkpointing to a certain degree. The Web Resource contains pointers to numerous check-point-related web sites, including some public domain software such as Condor and Libckpt. Here we will present the important issues for the designer and the user of checkpoint software. We will first consider the issues that are common to both sequential and parallel programs, and then we will discuss the issues pertaining to parallel programs.

d) JINI

Jini is a way to do distributed computing that helps you manage the dynamic nature of networks, Provides mechanisms to enable smooth adding removal, and finding of devices and services on the network

- * Provides a programming model for reliable, secure distributed services and makes it easier for programmers to get their devices talking to each other.

Jini technology promises to be a reality in the immediate future as an architecture to enable connections between devices anytime, anywhere.

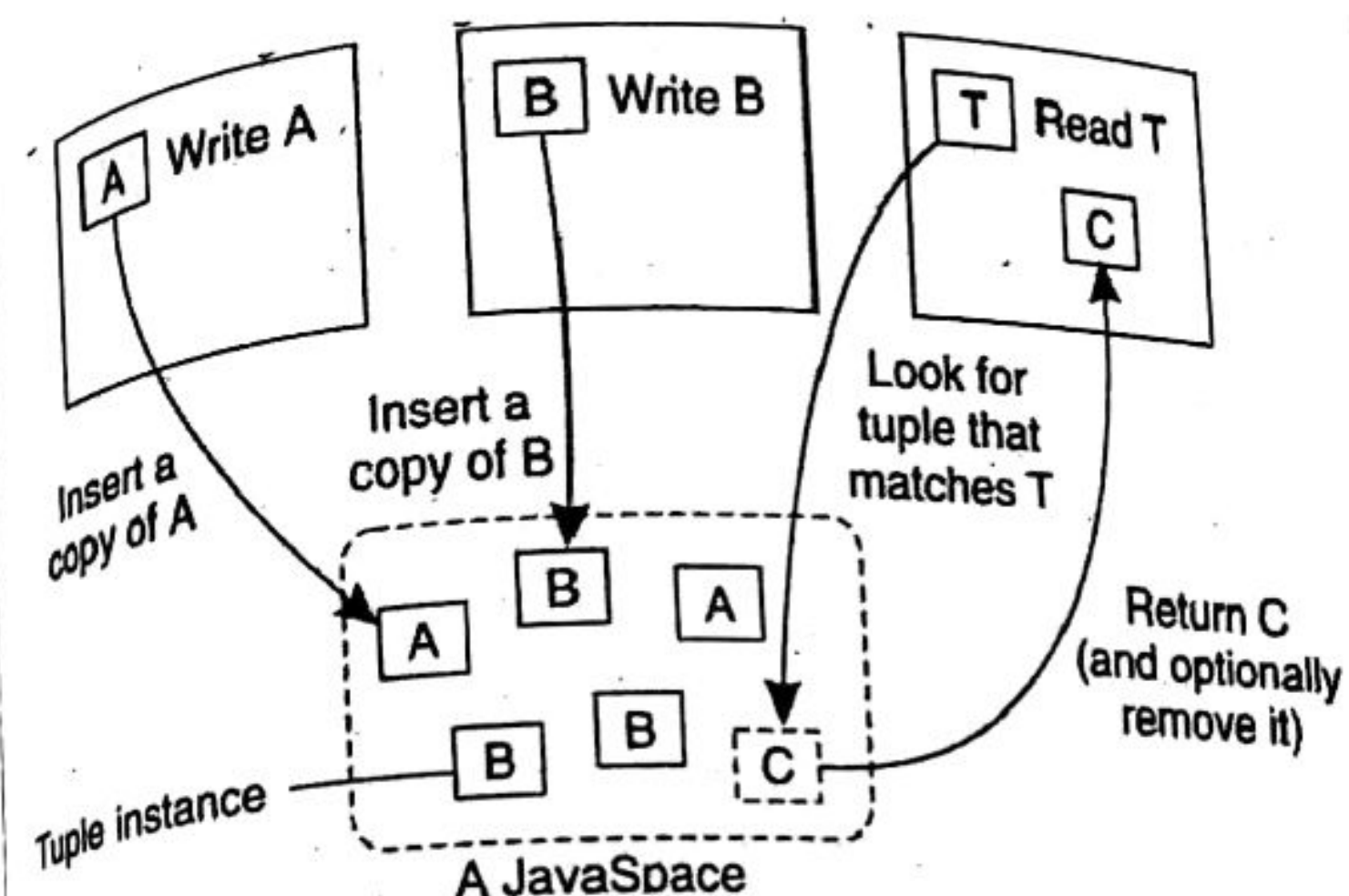


Fig: The general organization of a JavaSpace in Jini.

Q1) What are the principal applications of Distributed System (DS)? Discuss the advantages and disadvantages of DS.

The principal application of Distributed System are:

- Social networks
- mobile systems
- online banking
- online gaming (e.g., multiplayer systems)
- Additional areas of application for distributed computing include e-learning platforms, artificial intelligence, and e-commerce.

Advantages of Distributed System

1. Cost Effective

Although distributed systems consist of high implementation costs, they are cost effective in the long run.

2. Efficiency

Distributed systems are made to be efficient in every aspect since they possess multiple computers. Each of these computers could work independently to solve problems. This not only considered to be efficient, but it also significantly saves time of the user.

3. Scalability

Distributed systems are made on default to be scalable. Whenever there is an increase in workload, users can add more workstations. There is no need to upgrade a single system. Moreover, no any restrictions are placed on the number of machines.

4. Reliability

The distributed systems are far more reliable than single systems in terms of failures. Even in the case of a single node malfunctioning, it does not pose problems to the remaining servers. Other nodes can continue to function fine.

5. Latency

Distributed systems result on low latency. If a particular node is located closer to the user, the distributed system makes sure that the system receives traffic from that node. Thus, the user could notice much less time it takes to serve them.

Disadvantage of Distributed System

1. Startup Cost

Compared to a single system, the implementation cost of a distributed system is significantly higher. The infrastructure used in a distributed system makes it expensive. In addition to that, constant transmission of information and processing overhead further increases the cost.

2. Security

Distributed systems always come with security risks since it contains open system characteristics. The data of the user is stored in different workstations. Thus, the user needs to make sure that their data is secured in each of these computers. Moreover, unlike in a centralized computing system, it is not an easy task to manage data access in a distributed system.

3. Complexity

The difficulty involved in implementation, maintenance and troubleshooting makes distributed system a complex strategy. Besides hardware complexity, distributed system poses difficulty in software too. The software used in distributed system needs to be well attentive when handling communication and security.

4. Overheads

Over heading is a common problem faced by a distributed system. This happens when all the workstations try to operate at once. Even though this essentially brings desired results, eventually there will be an increase in computing time. This ultimately impacts the system's response time.

5. Network Errors

Distributed systems are prone to network errors which results in communication breakdown. The information may fail to be delivered or not in the correct sequence. And also, troubleshooting errors is a difficult task since the data is distributed across various nodes.

Q2) What are the requirements of Distributed File System? Describe file service architecture for Distributed File System.

The requirements of Distributed File System are:

- Transparency
- User mobility
- Performance
- Simplicity and ease of use
- High availability
- Scalability
- High reliability
- Data integrity

Second part, (done already)

Q3) Why naming is necessary in distributed system? Explain Domain Naming Service (DNS) with its features.

Ans:

Names play a very important role in all computer systems. They are used to share resources, to uniquely identify entities, to refer to locations, and more. An important issue with naming is that a name can be resolved to the entity it refers to. Name resolution thus allows a process to access the named entity.

Second part (done already)

Q4) What are the advantages of micro-Kernal over monolithic kernel? In your view, which kernel is preferable for distributed Operating system and why?

Ans:

The advantages of micro-kernel over monolithic kernel are:

Microkernel is more secure than monolithic kernel as if a service fails in microkernel the operating system remains unaffected. On the other hands, if a service fails in monolithic kernel entire system fails. Monolithic kernel designing requires less code, which further leads to fewer bugs.

I think microkernel is preferable for distributed operating system due to the following reasons.

A microkernel is one of the classifications of the kernel. Being a kernel, it manages all system resources. But in a microkernel, the user services and kernel services are implemented in different address spaces. The user services are kept in user address space, and kernel services are kept under kernel address space, thus also reduces the size of kernel and size of an operating system as well.

User mode	Applications			
	Libraries			
	Memory management	File system services	Drivers	...
Kernel mode	Microkernel			
	Hardware			

It provides minimal services of process and memory management. The communication between client program/application and services running in user address space is established through message passing, reducing the speed of execution microkernel.

The Operating System **remains unaffected** as user services and kernel services are isolated so if any user service fails it does not affect kernel service. Thus it adds to one of the advantages of a microkernel.

It is easily **extendible** i.e., if any new services are to be added they are added to user address space and hence require no modification in kernel space.

It is also portable, secure, and reliable.

- Advantages of Microkernel –
The architecture of this kernel is small and isolated hence it can function better.
- Expansion of the system is easier, it is simply added to the system application without disturbing the kernel.

Q5)a. What are the components of CORBA environment?

Done already

b) What do you mean by logical clock? Explain Lamport's Logical clock.

Done already

Q6) Define distributed coordination in DS? Explain how token ring algorithm works for mutual exclusion in DS.

Done already

Q7) Define replication and fault tolerance in DS and explain why are they necessary? Explain how replication enhanced scalability for DS.

Done already.

8) What is LOCK and DEADLOCK in DS? Discuss the methods of distributed deadlock avoidance.

A lock is a variable associated with a data item that determines whether read/write operations can be performed on that data item.

A Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource occupied by some other process.

Deadlock Avoidance

Deadlock avoidance can be done with Banker's Algorithm.

Banker's Algorithm

Banker's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn't allow the request made by the process.

Inputs to Banker's Algorithm:

Max need of resources by each process.

Currently, allocated resources by each process.

Max free available resources in the system.

The request will only be granted under the below condition:

- If the request made by the process is less than equal to max need to that process.
- If the request made by the process is less than equal to the freely available resource in the system.

Example:

Total resources in system:

A B C D

6 5 7 6

Available system resources are:

A B C D

3 1 1 2

Processes (currently allocated resources):

A B C D

P1 1 2 2 1

P2 1 0 3 3

P3 1 2 1 0

Processes (maximum resources):

A B C D

P1 3 3 2 2

P2 1 2 3 4

P3 1 3 5 0

Need = maximum resources - currently allocated resources.

Processes (need resources):

A B C D

P1 2 1 0 1

P2 0 2 0 1

P3 0 1 4 0

9) Write short notes on:

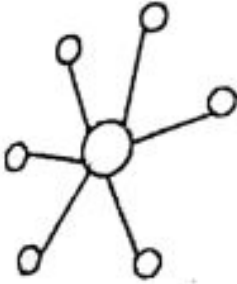


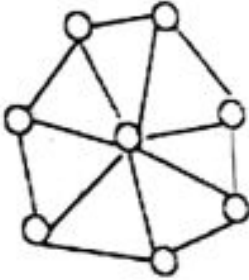

i) Process Resilience

ii) Mach

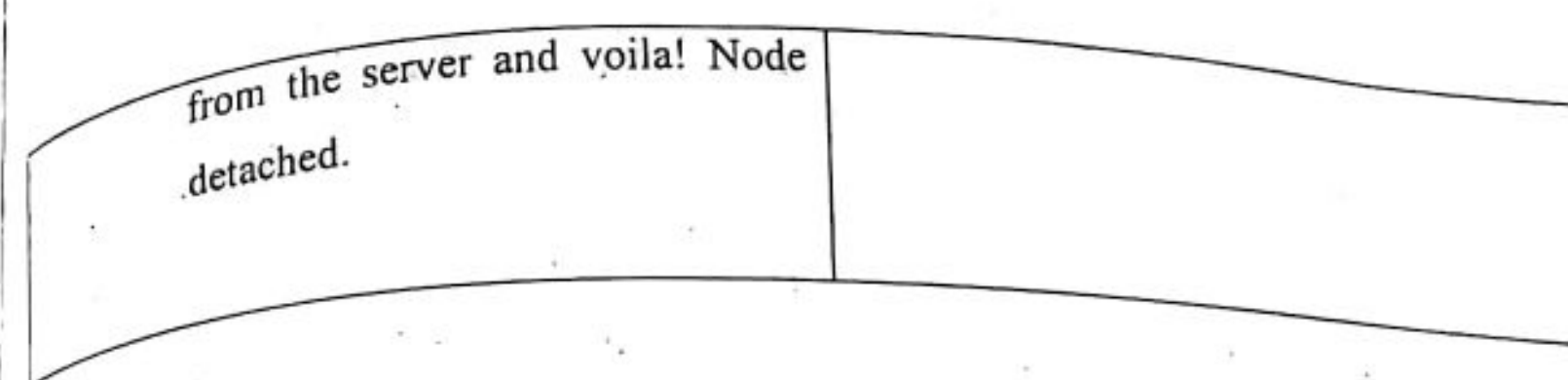
done already

Q1) Differentiate between centralized and distributed system? Explain the design issues related to distributed system?

Ans:

Centralized System	Distributed System
 <p>  — Server/master  — Computer/slave </p>	 <p>  — Node/Computer </p>
<p>Centralized systems are systems that use client/server architecture where one or more client nodes are directly connected to a central server.</p>	<p>In decentralized systems, every node makes its own decision.</p>
<p>This is the most used type of system in many organizations where a client sends a request to a company server and receives the response.</p>	<p>Google search system. Each request is worked upon by hundreds of computers which crawl the web and return the relevant results.</p>
<p>Limitations of Centralized System –</p> <ul style="list-style-type: none"> • Can't scale up vertically after a certain limit – After a limit, even if you increase the hardware and software capabilities of the server node, the performance will not increase appreciably leading to a cost/benefit ratio < 1. • Bottlenecks can appear when the traffic spikes – as the server can only have a finite number of open ports to which can listen to connections from client nodes. So, when high traffic 	<p>Limitations of Distributed System –</p> <ul style="list-style-type: none"> • Difficult to design and debug algorithms for the system. These algorithms are difficult because of the absence of a common clock; so no temporal ordering of commands logs can take place. Nodes can have different latencies which have to be kept in mind while designing such algorithms. The complexity increases with the increase in the number of nodes. Visit this link for more information

occurs like a shopping sale, the server can essentially suffer a Denial-of-Service attack or Distributed Denial-of-Service attack..	<ul style="list-style-type: none"> No common clock causes difficulty in the temporal ordering of events/transactions Difficult for a node to get the global view of the system and hence take informed decisions based on the state of other nodes in the system
Advantages of Centralized System – <ul style="list-style-type: none"> Easy to physically secure. It is easy to secure and service the server and client nodes by virtue of their location Smooth and elegant personal experience – A client has a dedicated system which he uses (for example, a personal computer) and the company has a similar system which can be modified to suit custom needs Dedicated resources (memory, CPU cores, etc) More cost-efficient for small systems up to a certain limit – As the central systems take fewer funds to set up, they have an edge when small systems must be built Quick updates are possible – Only one machine to update. Easy detachment of a node from the system. Just remove the connection of the client node 	Advantages of Distributed System – <ul style="list-style-type: none"> Low latency than a centralized system – Distributed systems have low latency because of high geographical spread, hence leading to less time to get a response



Second part:

8. **Heterogeneity:** Heterogeneity is applied to the network, computer hardware, operating system, and implementation of different developers. A key component of the heterogeneous distributed system client-server environment is middleware. Middleware is a set of services that enables application and end-user to interact with each other across a heterogeneous distributed system.
9. **Openness:** The openness of the distributed system is determined primarily by the degree to which new resource-sharing services can be made available to the users. Open systems are characterized by the fact that their key interfaces are published. It is based on a uniform communication mechanism and published interface for access to shared resources. It can be constructed from heterogeneous hardware and software.
10. **Scalability:** Scalability of the system should remain efficient even with a significant increase in the number of users and resources connected.
11. **Security:** Security of information system has three components Confidentiality, integrity and availability. Encryption protects shared resources, keeps sensitive information secrets when transmitted.
12. **Failure Handling:** When some faults occur in hardware and the software program, it may produce incorrect results or they may stop before they have completed the intended computation so corrective measures should be implemented to handle this case. Failure handling is difficult in distributed systems because the failure is partial i.e., some components fail while others continue to function.
13. **Concurrency:** There is a possibility that several clients will attempt to access a shared resource at the same time. Multiple users make requests on the same resources, i.e. read, write, and update. Each resource must be safe in a concurrent environment. Any object that represents a shared resource in a distributed system must ensure that it operates correctly in a concurrent environment.
14. **Transparency:** Transparency ensures that the distributed system should be perceived as a single entity by the users or the application programmers rather than the collection of autonomous systems, which is cooperating. The user should be unaware of where the services are located and the transferring from a local machine to a remote one should be transparent.

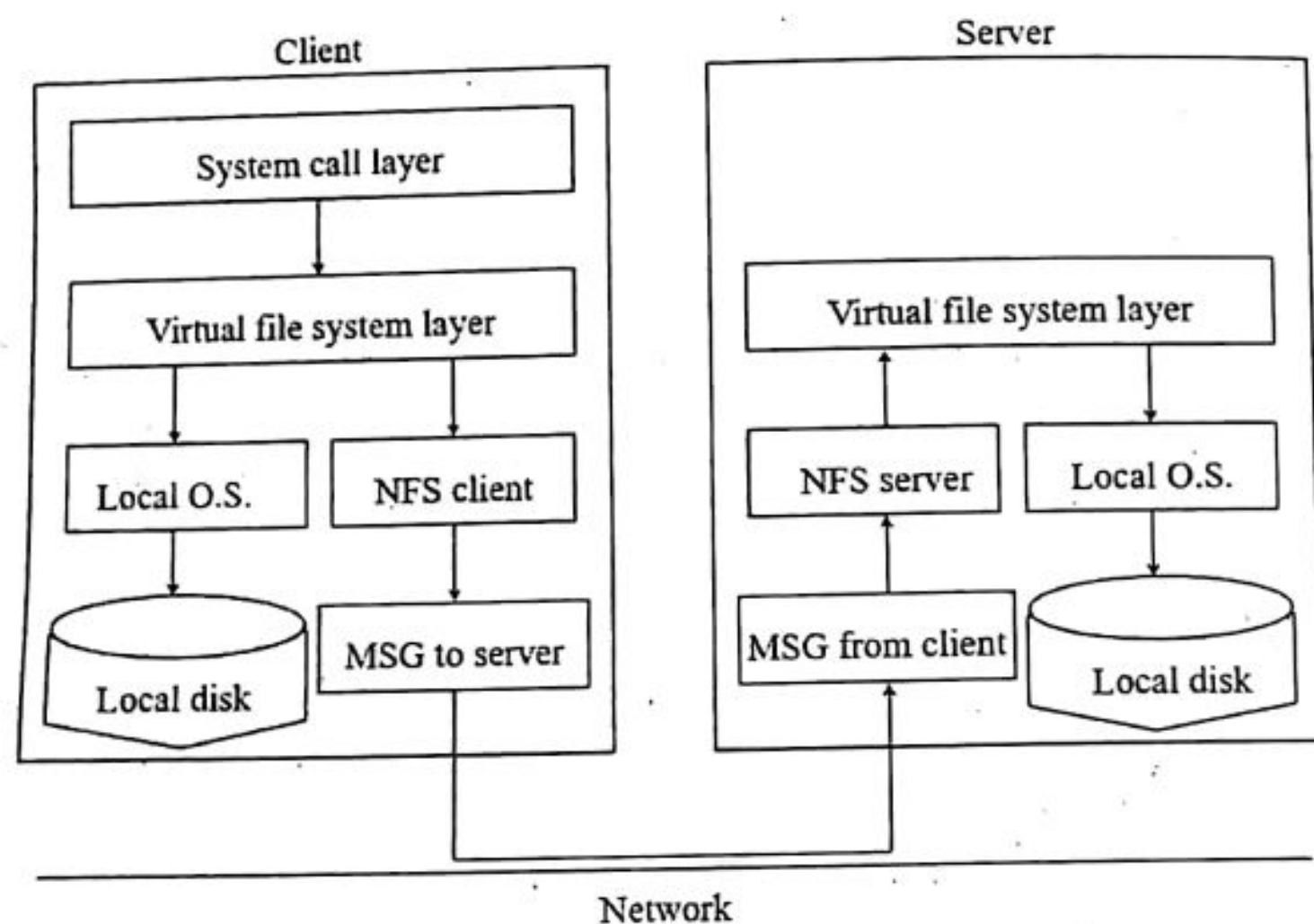
Q2) Discuss the importance of distributed file system (DFS). Describe the operations of SUN NFS with its properties.

Ans:

It allows the data to be share remotely. It improved the availability of file, access time, and network efficiency. Improved the capacity to change the size of the data and also improves the ability to exchange the data. Distributed File System provides transparency of data even if server or disk fails.

Second part:

SUN NFS



Sun's Network File System:

The earliest successful distributed system could be attributed to Sun Microsystems, which developed the Network File System (NFS). NFSv2 was the standard protocol followed for many years, designed with the goal of simple and fast server crash recovery. This goal is of utmost importance in multi-client and single server-based network architectures because a single instant of server crash means that all clients are unserved. The entire system goes down.

Stateful protocols make things complicated when it comes to crashes. Consider a client A trying to access some data from the server. However, just after the first read, the server crashed. Now, when the server is up and running, client A issues the second read request. However, the server does not know which file the client is referring to, since all that information was temporary and lost during the crash.

Stateless protocols come to our rescue. Such protocols are designed so as to not store any state information in the server. The server is unaware of what the clients are doing — what blocks they are caching, which files are opened by them and where their current file pointers are. The server simply delivers all the information that is required to service a client request. If a server crash happens, the client would simply have to retry the request. Because of their simplicity, NFS implements a stateless protocol.

File Handles:

NFS uses file handles to uniquely identify a file or a directory that the current operation is being performed upon. This consists of the following components:

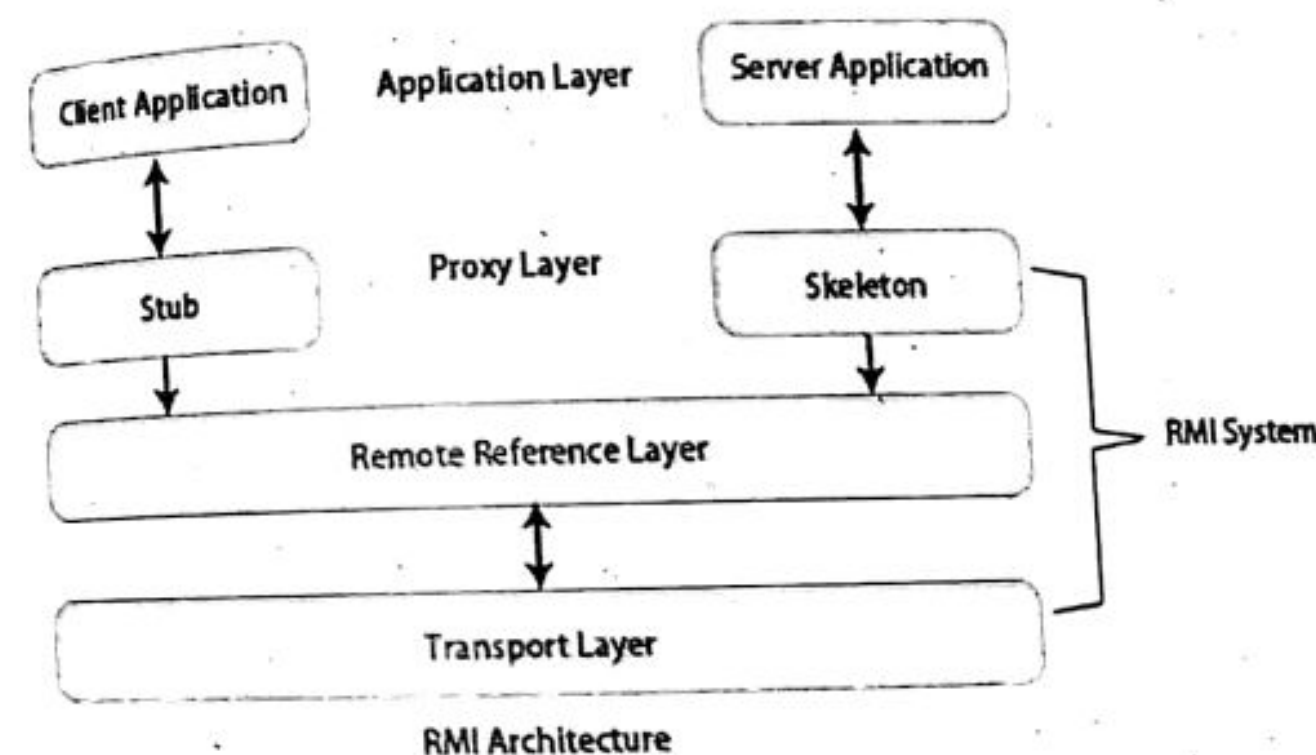
- **Volume Identifier** – An NFS server may have multiple file systems or partitions. The volume identifier tells the server which file system is being referred to.
- **Inode Number** – This number identifies the file within the partition.
- **Generation Number** – This number is used while reusing an inode number.

File Attributes:

"File attributes" is a term commonly used in NFS terminology. This is a collective term for the tracked metadata of a file, including file creation time, last modified, size, ownership permissions etc. This can be accessed by calling `stat()` on the file.

Q3) Explain RMI with suitable diagram. How RMI is superior to RPC.

Ans:



RMI stands for Remote Method Invocation. It is an API provided by java that allows an object residing in one JVM (Java Virtual Machine) to access or invoke an object running on another JVM. The other JVM could be on the same machine or remote machine.

Let us now discuss the components of this architecture.

- **Transport Layer** – This layer connects the client and the server. It manages the existing connection and sets up new connections.
- **Stub** – A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.

- **Skeleton** – This is the object which resides on the server side. stub communicates with this skeleton to pass request to the remote object.
- **RRL (Remote Reference Layer)** – It is the layer which manages the references made by the client to the remote object.

Following are the features of RMI –

- To minimize the complexity of the application.
- To preserve type safety.
- Distributed garbage collection.
- Minimize the difference between working with local and remote objects.

Second part:

RMI:

It supports object-oriented programming.

It is more efficient in comparison to RPC.

It creates less overhead in comparison to RPC. In this, objects are passed as parameters.

Q4) What is the role of middleware in DS? Explain about CORBA and its services.

Ans:

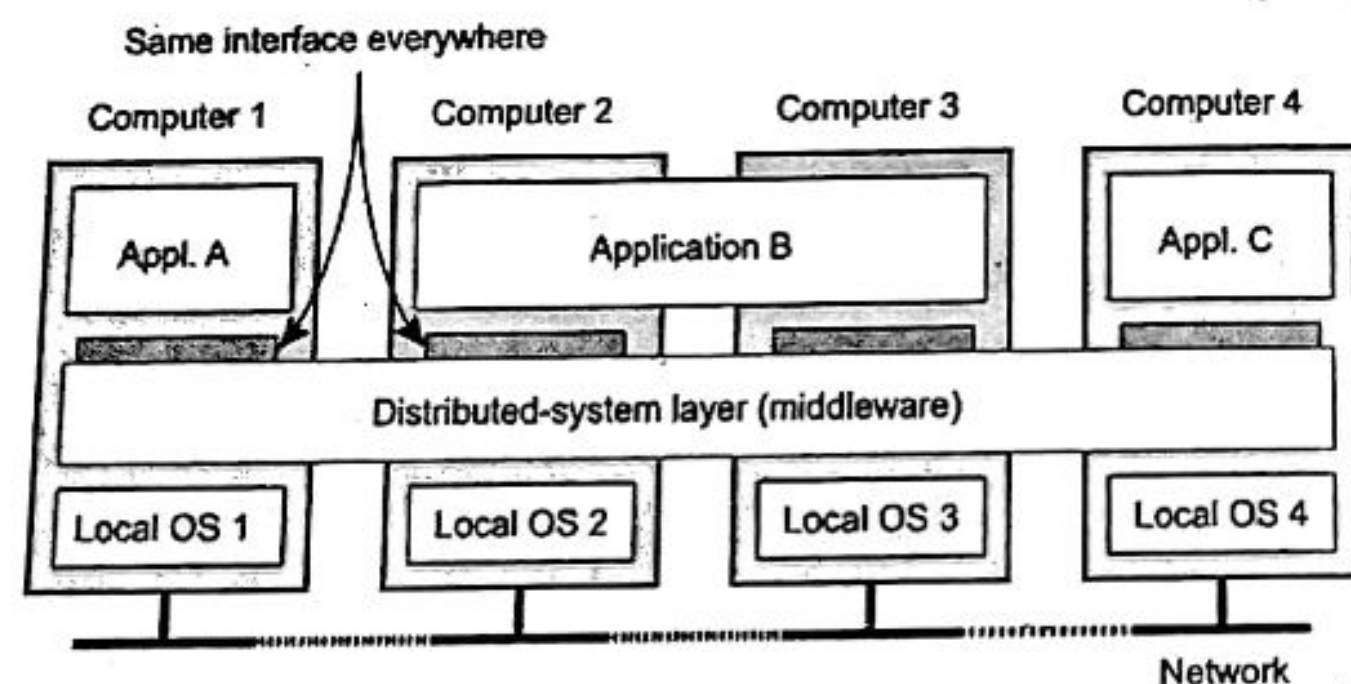
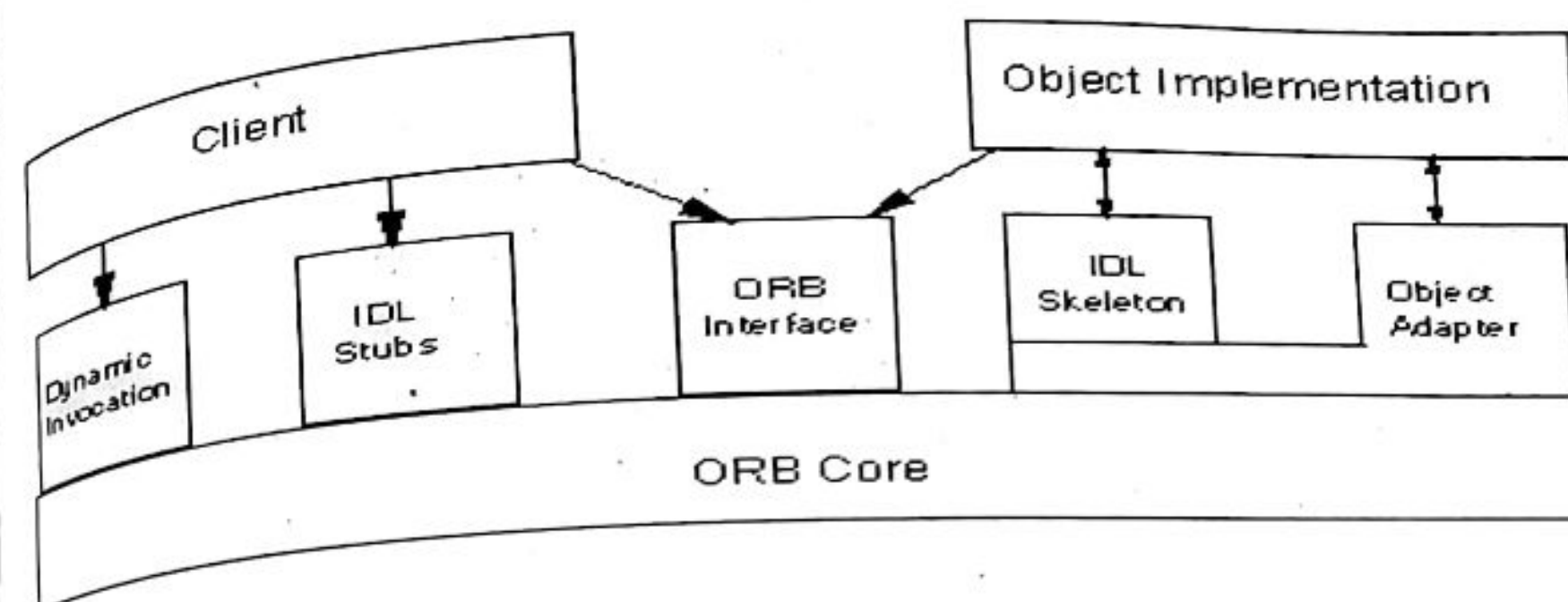


Fig: middleware in DS

Middleware in the context of distributed applications is software that provides services beyond those provided by the operating system to enable the various components of a distributed system to communicate and manage data. Middleware supports and simplifies complex distributed applications

Second part:

CORBA Architecture



The Common Object Request Broker Architecture (CORBA) describes a messaging mechanism by which objects distributed over a network can communicate with each other irrespective of the platform and language used to develop those objects. CORBA enables collaboration between systems on different operating systems, programming languages, and computing hardware. CORBA uses an object-oriented model although the systems that use the CORBA do not have to be object-oriented.

Dynamic Invocation - This interface allows for the specification of requests at runtime. This is necessary when the object interface is not known at run-time. Dynamic Invocation works in conjunction with the interface repository.

IDL Stub - This component consists of functions generated by the IDL interface definitions and linked into the program. The functions are a mapping between the client and the ORB implementation. Therefore, ORB capabilities can be made available for any client implementation for which there is a language mapping. Functions are called just as if it was a local object.

ORB Interface - The ORB interface may be called by either the client or the object implementation. The interface provides functions of the ORB which may be directly accessed by the client (such as retrieving a reference to an object.) or by the object implementations. This interface is mapped to the host programming language. The ORB interface must be supported by any ORB.

ORB core - Underlying mechanism used as the transport level. It provides basic communication of requests to other sub-components.

IDL Skeleton Interface - The ORB calls method skeletons to invoke the methods that were requested from clients.

Object Adapters (OA) - Provide how object implementations access most ORB services. This includes the generation and interpretation of object references, method invocation, security, and activation. Requests - The client requests a service from the object implementation. The ORB transports the request, which invokes the method using object adapters and the IDL skeleton.

Object Adapters - Object Adapters (OA) are the primary ORB service providers to object implementations. OA has a public interface that is used by the object implementation and a private interface that is used by the IDL skeleton.

Q5) Differentiate between physical and logical clock. Why it is difficult to synchronize physical clock? Describe a method for physical clock.

Ans:

Physical Clock	Logical Clock
Tied to the notion of real time	Not tied to the notion of real time
Perfect synchronization not possible because of inability to estimate network delays exactly	Perfect Synchronization possible.
The physical clocks are used to adjust the time of nodes. Each node in the system can share its local time with other nodes in the system.	A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system.

Second part:

Communication between processes in a distributed system can have unpredictable delays, processes can fail, messages may be lost •

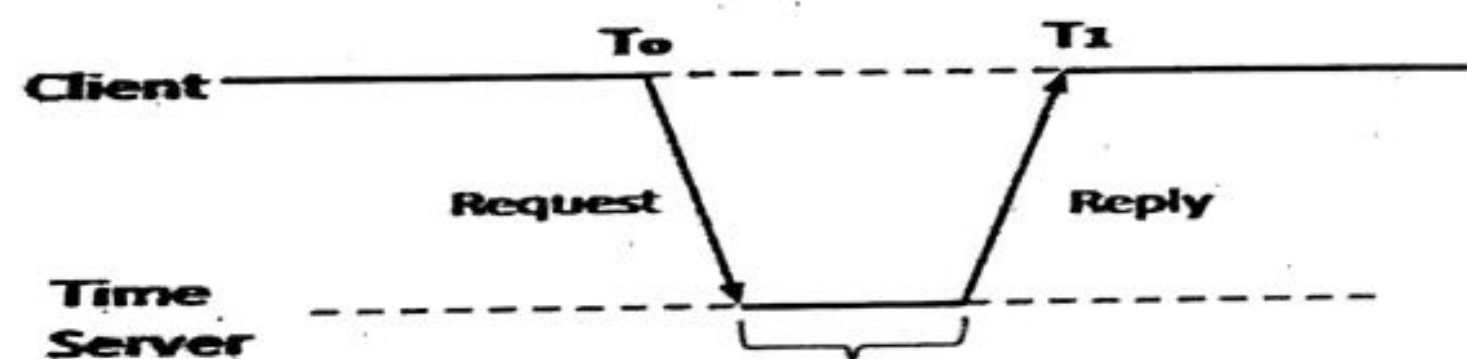
Synchronization in distributed systems is harder than in centralized systems because the need for distributed algorithms.

Third part:

Cristian's Method

- It makes use of a time server to get the current time and helps in synchronization of computer externally.
- Upon request, the server process S provides the time according to its clock to the requesting process p.
- This method achieve synchronization only if the round trip times between client and time server are sufficiently short compared to the required accuracy.

Algorithm:



- A process p requests time in a message m_r and receives time value t in a message m_t . Process p records total round trip time $T(\text{round})$ taken to send request m_r and receive reply m_t .
- Assuming elapsed time is splitted before and after S placed t in m_t , the time estimate to which p should set its clock is $t + T(\text{round})/2$.
- Assuming \min as the earliest point at which S could have placed time m_t after p dispatches m_r , then:
 - a) Time by S's clock when reply arrives is in range $[t + \min, t + T(\text{round}) - \min]$
 - b) Width of time range is $T(\text{round}) - 2 * \min$
 - c) Accuracy is $\pm (T(\text{round}) / 2 - \min)$

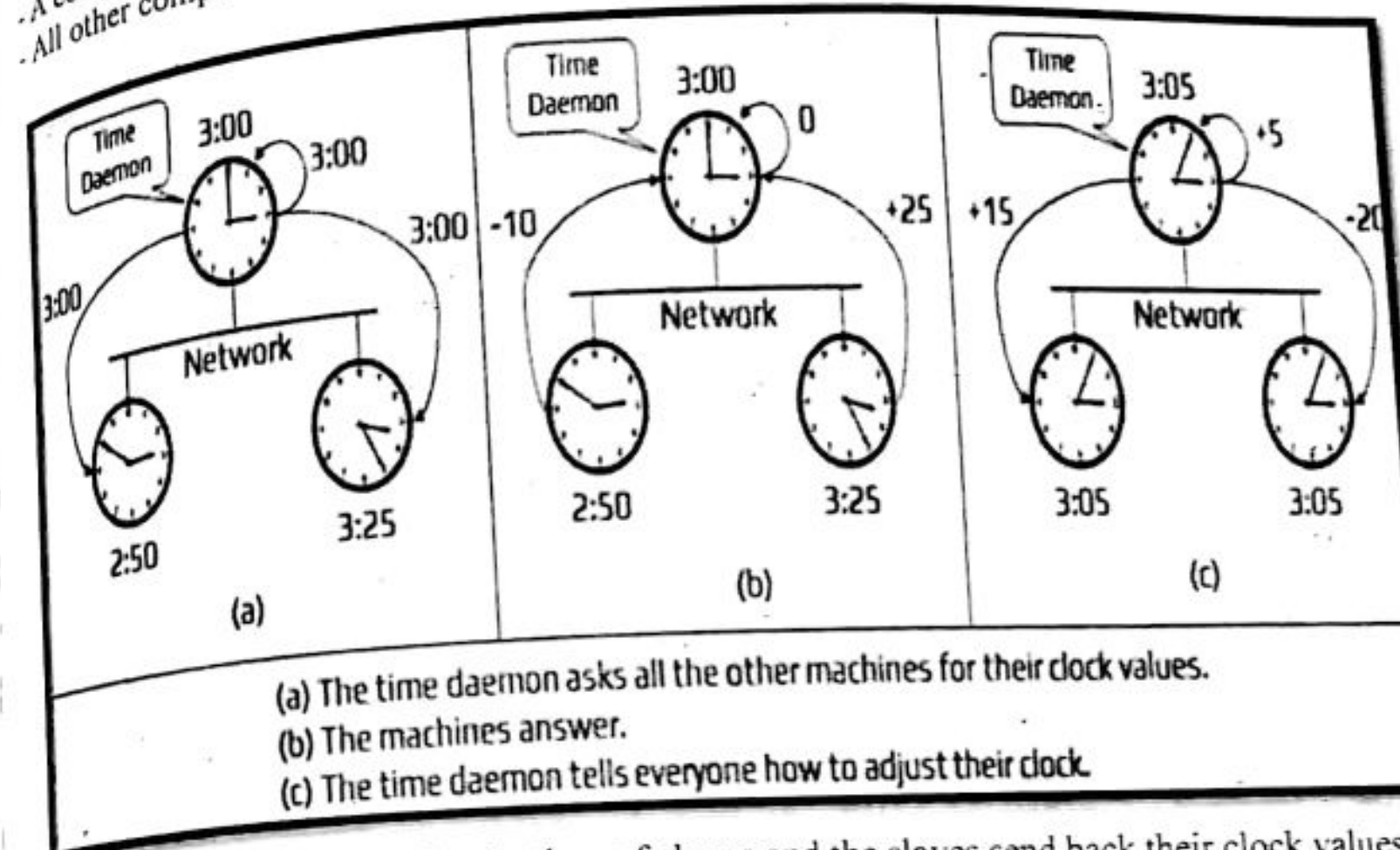
Discussion:

If a time server fails, the synchronization is impossible.

To remove this drawback, time should be provided by a group of synchronized time servers.

rikeley's Algorithm

- It is an algorithm for internal synchronization.
- A computer is chosen as a master.
- All other computers are slaves.



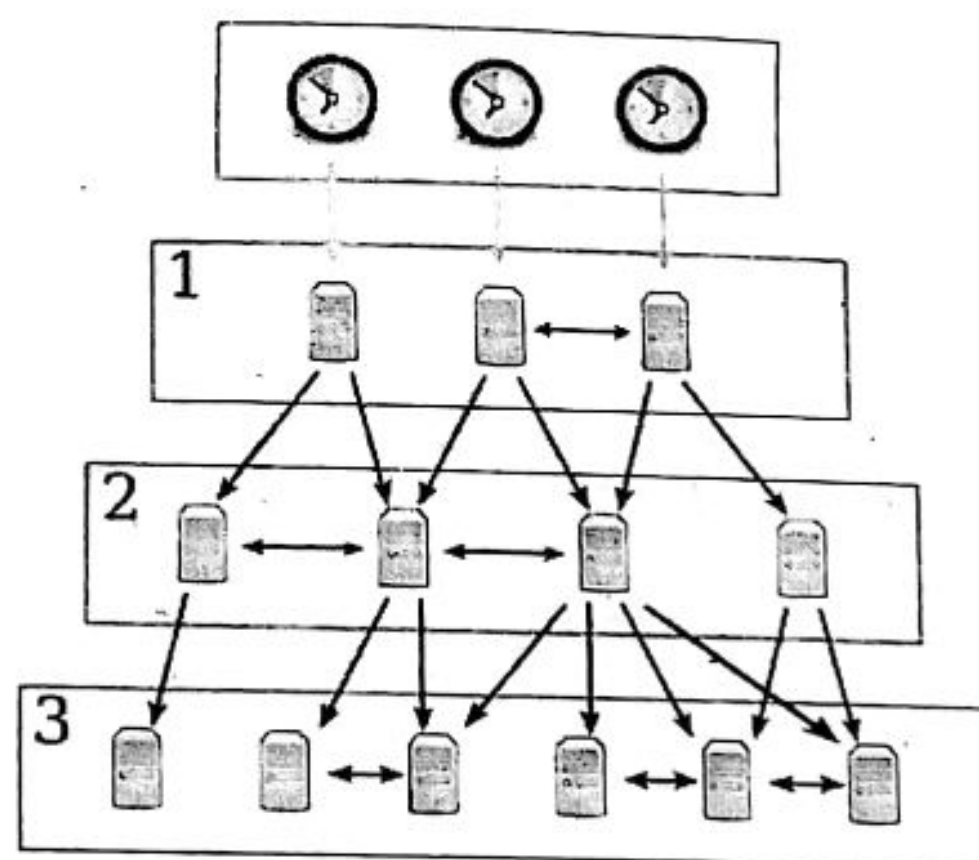
- Master periodically polls for the time of slaves and the slaves send back their clock values to master.
- The master estimates local time of each slave by observing the round-trip times.
- Master calculates average of obtained time including its own time.
- While calculating average, it eliminates faulty clocks by choosing a subset of clocks that do not differ from one another by more than a specified amount.
- The master then sends the amount by which each slave should adjust their clock which may be positive or negative.
- If the master fails, one of the slaves can be elected to take the place of master.

Network Time Protocol (NTP)

- It defines an architecture to enable clients, across the Internet to be synchronized accurately to UTC.
- It synchronizes against many time servers.

Design Aims:

- Adjust system clock close to UTC over Internet.
- Handle bad connectivity
- Enable frequent resynchronization
- Security



Hierarchical Structure of NTP:

- NTP is provided by a network of servers located across the Internet.
- Primary servers are connected directly to a time server.
- Secondary servers are synchronized with primary servers.
- The logical hierarchy of server connection is called synchronization subnet.
- Each level of synchronization subnet is called stratum.
- Lowest level executes in user's workstation.
- Server with high stratum numbers are liable to have less accurate clocks.

Server synchronization can be done in following ways:

1. Multicast mode:

- Servers periodically multicasts time to other servers in the network.
- Receivers set their clock assuming small delay.

2. Procedure Call mode:

- One server accepts requests from other computers.
- Server replies with its timestamp.

3. Symmetric mode:

- A pair of servers on higher subnet layers exchange messages to improve accuracy of synchronization over time.

Q6) What are the basic requirements for mutual exclusion in distributed system? Explain the non-token based distributed mutual exclusion algorithm and compare it with token-based algorithm.

Ans:

Requirements of Mutual exclusion Algorithm:

• No Deadlock:

Two or more sites should not endlessly wait for any message that will never arrive.

- **No Starvation:**
Every site who wants to execute critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other site is repeatedly executing critical section
- **Fairness:**
Each site should get a fair chance to execute critical section. Any request to execute critical section must be executed in the order they are made i.e. Critical section execution requests should be executed in the order of their arrival in the system.
- **Fault Tolerance:**
In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

Second part:

Token Based	Non Token Based
In the Token-based algorithm, a unique token is shared among all the sites in Distributed Computing Systems.	In Non-Token based algorithm, there is no token even not any concept of sharing token for access.
Here, a site is allowed to enter the Computer System if it possesses the token.	Here, two or more successive rounds of messages are exchanged between sites to determine which site is to enter the Computer System next.
The token-based algorithm uses the sequences to order the request for the Computer Systems and to resolve the conflict for the simultaneous requests for the System.	Non-Token based algorithm uses the timestamp (another concept) to order the request for the Computer Systems and to resolve the conflict for the simultaneous requests for the System.
The token-based algorithm produces less message traffic as compared to Non-Token based Algorithm.	Non-Token based Algorithm produces more message traffic as compared to the Token-based Algorithm.
They are free from deadlock (i.e. here there are no two or more processes are in the queue in order to wait for messages that will actually can't come) because of the existence of unique token in the distributed system.	They are not free from the deadlock problem as they are based on timestamp only.
Here, it was ensured that requests are executed exactly in the order as they are made in.	Here there is no surety of execution order.
Token-based algorithms are more scalable as they can free your server from having to store session state and also they contain all the necessary information which they need for authentication.	Non-Token based algorithms are less scalable than the Token-based algorithms because here server is not free from its tasks.
Here the access control is quite Fine-grained	Here the access control is not so fine as there

because here inside the token roles, permissions and resources can be easily specifying for the user.	is no token which can specify roles, permission, and resources for the user.
Token-based algorithms make authentication quite easy.	Non-Token based algorithms can't make authentication easy.
Examples of Token-Based Algorithms are: (i) Singhal's Heuristic Algorithm (ii) Raymonds Tree Based Algorithm (iii) Suzuki-Kasami algorithm	Examples of Non-Token Based Algorithms are: (i) Lamport's Algorithm (ii) Ricart-Agarwala Algorithm (iii) Maekawa's Algorithm

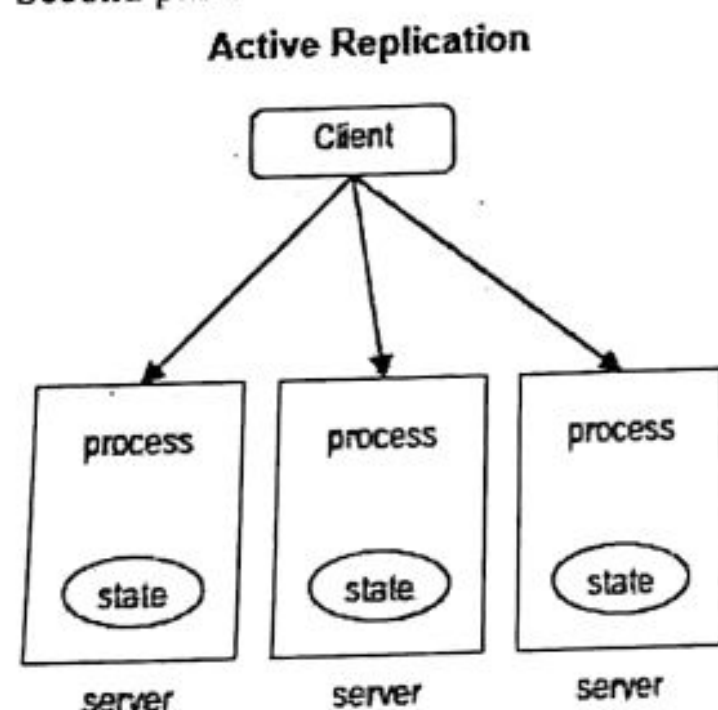
Q7) What are the reasons for replication? Explain active replication model with its advantages and disadvantages.

Ans:

Two primary reasons for replication: reliability and performance.

- Increasing reliability: – If a replica crashes, system can continue working by switching to other replicas. – Avoid corrupted data: • can protect against a single, failing write operation.
- Improving performance: – Important for distributed systems over large geographical areas. – Divide the work over a few servers. – Place data in the proximity of clients.

Second part:



In **active replication** each client request is processed by all the servers. Active Replication was first introduced by under the name **state machine replication**. This requires that the process hosted by the servers is **deterministic**. **Deterministic** means that, given the same initial state and a request sequence, all processes will produce the same response sequence and end up in the same final state. To make all the servers receive the same sequence of operations, an **atomic broadcast** protocol must be used. An atomic broadcast protocol guarantees that either all the servers receive a message or none, plus that they all receive messages in the same order. The big disadvantage for active replication is that in practice most of the real world systems that require quick response even under the presence of faults or with systems that must handle **byzantine faults**.

Advantages:

- It is really simple. The codes in active replication are the same throughout.

- It is transparent.
- Even if a node fails, it will be easily handled by replicas of that node.

Disadvantages:

- It increases resource consumption. The greater the number of replicas, the greater the memory needed.
- It increases the time complexity. If some change is done on one replica it should also be done in all others.

Q8) What do you mean by nested transaction? Explain optimistic concurrency control method with its advantages over the other concurrency control methods.

Ans:

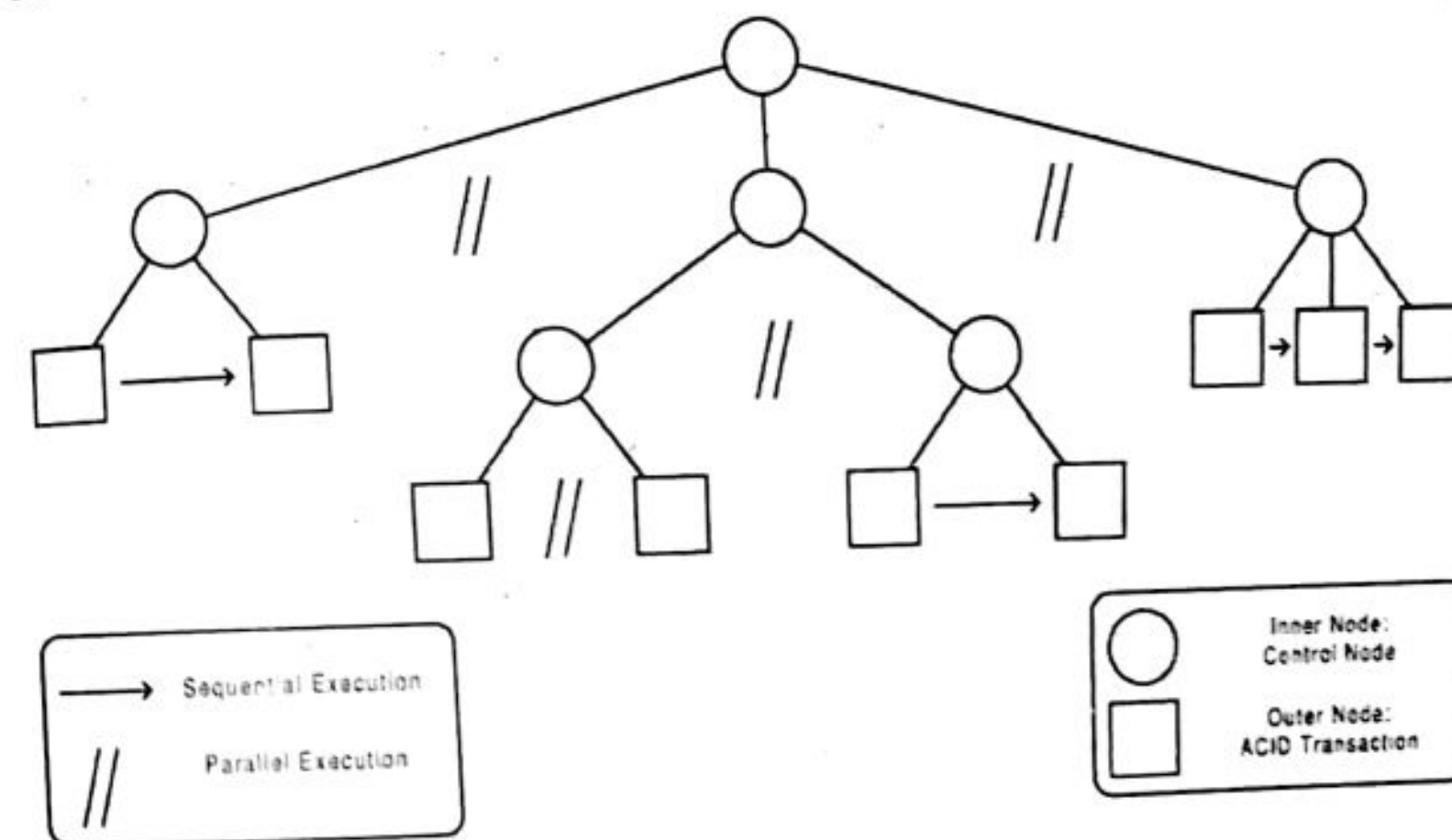


Fig: Nested Transaction

A transaction that includes other transactions within its initiating point and a end point are known as nested transactions. So, the nesting of the transactions is done in a transaction. The nested transactions here are called sub-transactions.

The top-level transaction in a nested transaction can open sub-transactions, and each sub-transaction can open more sub-transactions down to any depth of nesting.

Second part:

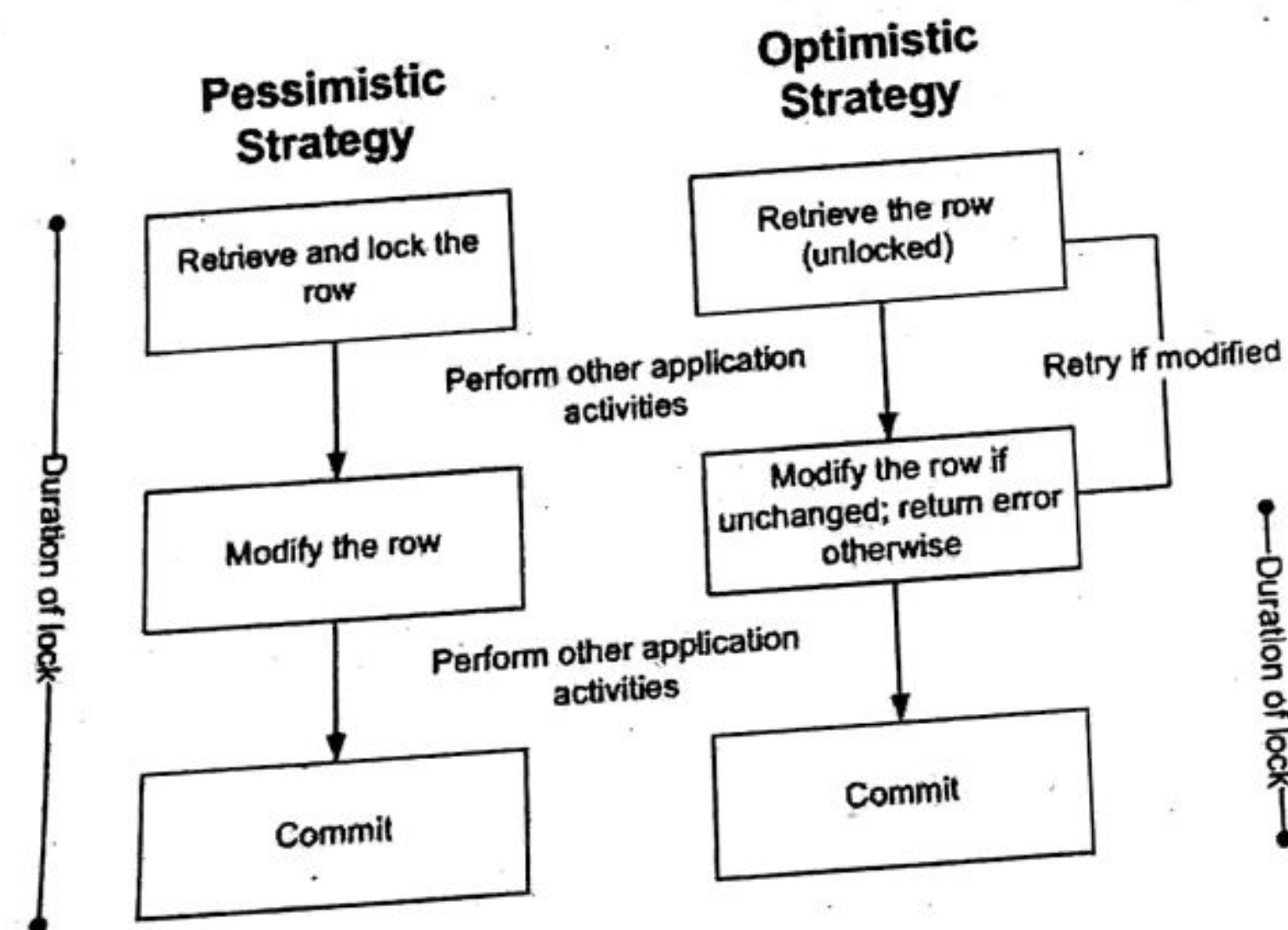
Concurrency can simply be said to be executing multiple transactions at a time. It is required to increase time efficiency. If many transactions try to access the same data, then inconsistency arises. Concurrency control required to maintain consistency data.

For example, if we take ATM machines and do not use concurrency, multiple persons cannot draw money at a time in different places. This is where we need concurrency.

Optimistic

It assumes that conflict is rare, and it is more efficient to allow transactions to proceed without imposing delays to ensure serializability. In a multiuser environment, there are two models for updating data in a database: optimistic concurrency and pessimistic concurrency. The Dataset object is designed to encourage the use of optimistic concurrency for long-running activities, such as removing data and interacting with data. Pessimistic concurrency involves locking rows at the data source to prevent other users from modifying data in a way that affects the current user. In a pessimistic model, when a user performs an action that causes a lock to be applied, other users cannot perform actions that would conflict with the lock until the lock owner releases it. This model is primarily used in environments where there is heavy contention for data, so that the cost of protecting data with locks is less than the cost of rolling back transactions if concurrency conflicts occur.

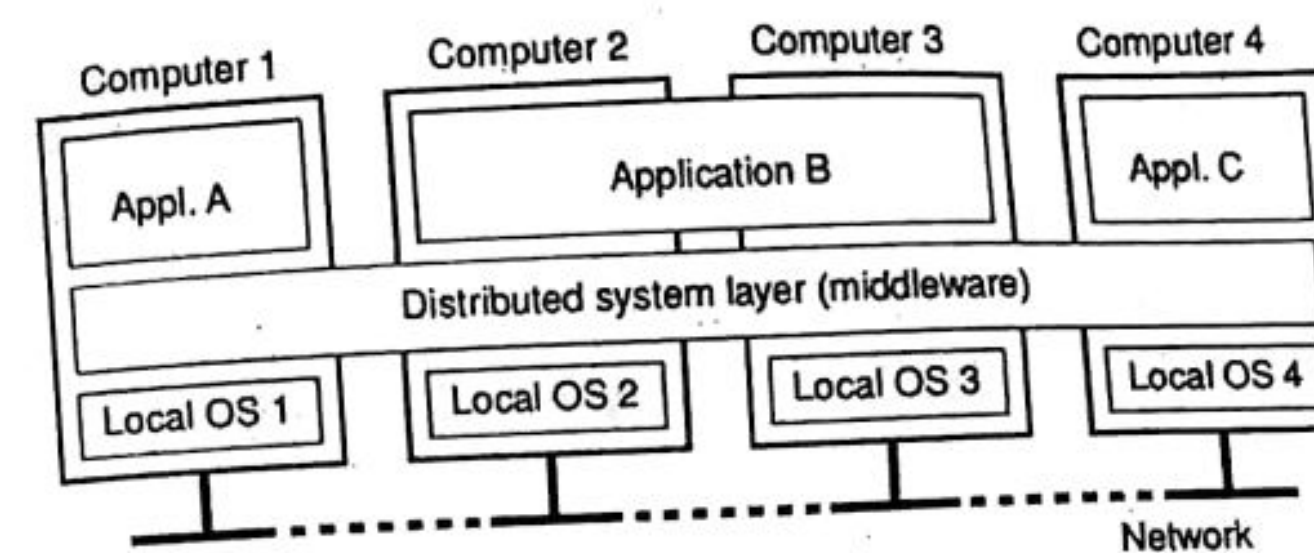
Therefore, in a pessimistic concurrency model, a user who updates a row establishes a lock. Until the user has finished the update and released the lock, no one else can change that row. For this reason, pessimistic concurrency is best implemented when lock times will be short, as in programmatic processing of records. Pessimistic concurrency is not a scalable option when users are interacting with data and causing records to be locked in relatively large periods of time.



Q9) Write short notes on:

- Distributed OS
- JINI

i) Distributed OS



A distributed operating system (DOS) is an essential type of operating system. Distributed systems use many central processors to serve multiple real-time applications and users. As a result, data processing jobs are distributed between the processors.

It connects multiple computers via a single communication channel. Furthermore, each of these systems has its own processor and memory. Additionally, these CPUs communicate via high-speed buses or telephone lines. Individual systems that communicate via a single channel are regarded as a single entity. They're also known as loosely coupled systems.

Types of Distributed Operating System

There are various types of Distributed Operating systems. Some of them are as follows:

- Client-Server Systems
- Peer-to-Peer Systems
- Middleware
- Three-tier
- N-tier

ii) JINI

Jini is a way to do distributed computing that helps you manage the dynamic nature of networks, Provides mechanisms to enable smooth adding removal, and finding of devices and services on the network

- * Provides a programming model for reliable, secure distributed services and makes it easier for programmers to get their devices talking to each other.

Jini technology promises to be a reality in the immediate future as an architecture to enable connections between devices anytime, anywhere.

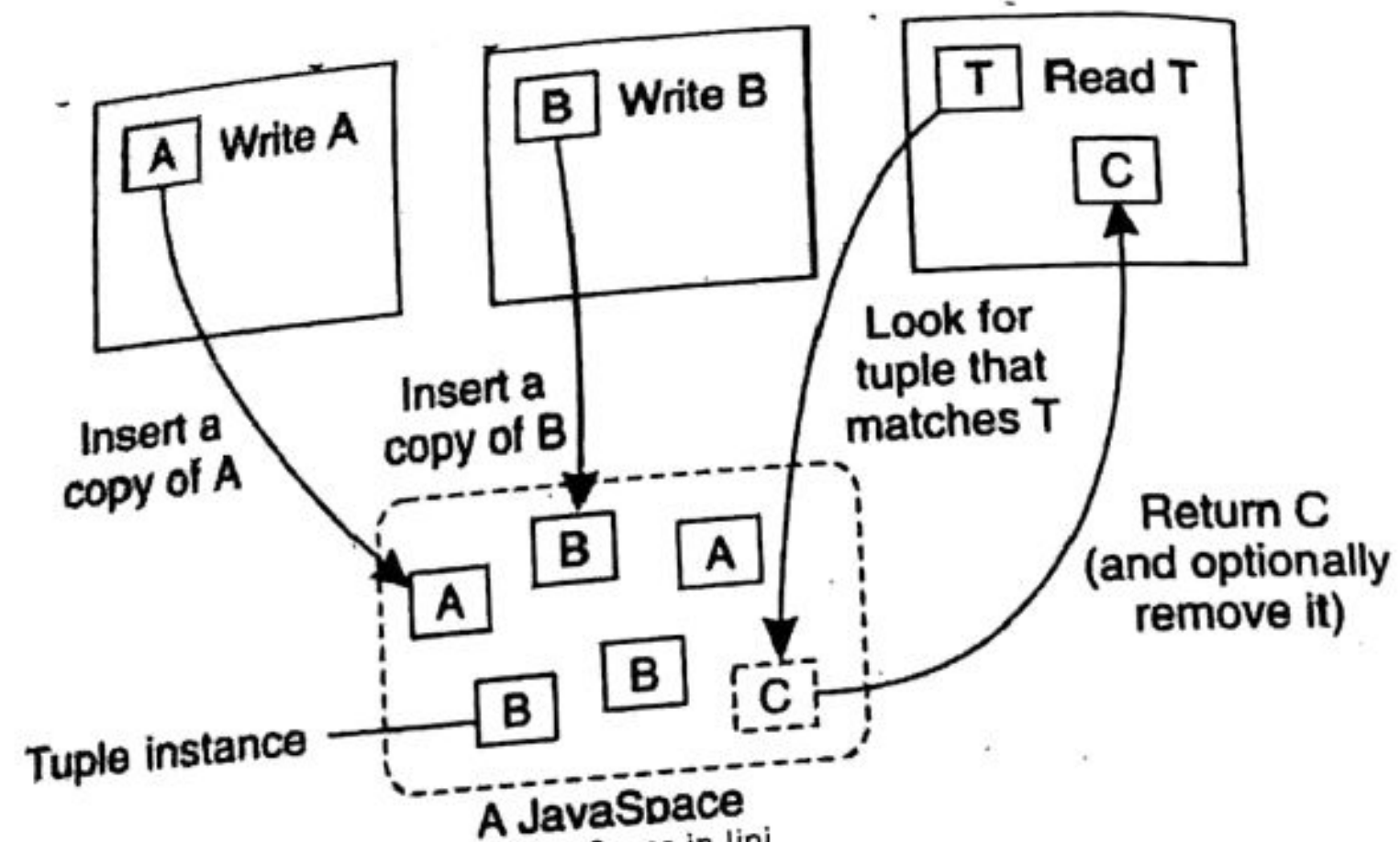


Fig : The general organization of a JavaSpace in Jini.

References

1. de Matos F, Rego P and Trinta F Secure Computational Offloading with gRPC: A Performance Evaluation in a Mobile Cloud Computing Environment Proceedings of the 11th ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications, (45-52)
2. Zimmerling M, Mottola L and Santini S 2020, Synchronous Transmissions in Low-Power Wireless, ACM Computing Surveys, 53:6, (1-39), Online publication date: 30-Nov-2021.
3. Fu X, Cai H, Li W and Li L 2020, SEADS, ACM Transactions on Software Engineering and Methodology, 30:1, (1-45), Online publication date: 31-Jan-2021.
4. Weiss W, Jiménez V and Zeiner H 2020, Dynamic Buffer Sizing for Out-of-order Event Compensation for Time-sensitive Applications, ACM Transactions on Sensor Networks, 17:1, (1-23), Online publication date: 28-Feb-2021.
5. Kokolis A, Psistakis A, Reidys B, Huang J and Torrellas J Distributed Data Persistency MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, (71-85)
6. Class Note provided by Subject Teacher