# Distributed computing notes compressed

Distributed Computing (University of Mumbai)

# Distributed Computing

## * Introduction to Distributed System.



## * Communication

Page :
Date :

* Resource and Process Management



* Consistency Replication Fault Tauleranca

Page :
Date :

# * Distributed File System and Name Service

Page
Date

# Module 1: Introduction to Distributed System

Q. What is Distributed System?

=> Distributed System is a collection of autonomous computers linked with a Communication network and equipped with a Distributed System Software.


Examples: Internet, intranet etc.

Characteristics of Distributed System:

1. Heterogenity: In Distributed Systems components can have variety and differences in Networks, Computer Hardware, Operating Systems, Programming languages and implementations by different developers.

2. Concurrency: It is a property of a system representing the fact that multiple activities are executed at the same time. The concurrent execution of activities take place in different components.

Camlin

running on multiple machines as part of a distributed system. In addition, these activities may perform some kind of interactions among them. Concurrency reduces the latency and increases the throughput of distributed system.

3. Openness: It is concerned with extensions and improvements of distributed systems. The distributed system must be open in terms of Hardwares and Softwares.

4. Scalability: It will remain effective when there is a significant increase in the number of users and the number of resources.

5. Fault Tolerance: In a Distributed System hardware, software, network anything can fail. The system must be designed in such a way that it is available all the time even after something has failed.

6. Concurrency: Concurrency refers to the Systems ability to handle the access and use of shared resources. This is important because if there is no measure implemented it is possible for the data to get corrupted or lost by two nodes making different changes

to the same resourse such that the system can carry this error through different process causing an incorrect result.

7. Resource Sharing: This means that the existing resources in a distributed system can be accessed or remotely accessed across multiple computers in the system. Example hardware, software or data.
Benefits: Reduction of cost and convenience.

8. Transparency:
Distributed systems should be perceived by users and application programmers as a whole rather than as a collection of ~~tow~~ cooperating components. Transparency can be of various types like ~~access~~, ~~location, concurrency~~, repli access, migration, failure etc.

| Transparency | Description |
|---|---|
| Access | Hide difference in data representation |
| Location | Hide where resource is located |
| Migration | Hide that a resource may move to other location |
| Relocation | Hide that a resource may move to other location while in use |
| Replication | Hide that a resource is Replicated |
| 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. |

Note: If this question is asked for 10 M you can write entire answer and if it is asked for 5 M you can write till 5th character.

Q2 Issues of Distributed System?

- Support of Heterogeneity: Some distributed system do not support heterogeniety as they are either platform independent or programming language dependent.

- Lack of failure handling mechanism: Most of the distributed system lack of failure handling mechanism where failure detection and recovery is difficult in real time.

- Scalability: The distributed System has flaws in scalability when the number of users or resources exceeds limit. Also affect the performance of system.

- Openness: Most of Distributed Systems have issues with openness, which does not allow the developers to extend or reimplement the System openly.

- Security: Denial of service is most common Security attack in Distributed System.

Q3 Goals of Distributed System?

=> Support of heterogeneous hardware and Software

- Resources are easily accessible across the network.

- It should hide the fact about the resources in terms of transparencies

- Distributed System should be scalable

- The system follows open standards so that they use Standard Syntax and semantics

- The system should be independent of geographical location of servers and easily manageable by the administrators

- The System should be capable to detect and recover from failure, it should be fault tolerant and robust.

Q4 Advantages and Disadvantages of Distributed System.

=> Advantages:

- All the nodes in the distributed system are connected to each other. So nodes can easily share data with other nodes.

- Failure of one node does not lead to failure of the entire distributed system. Other nodes can still communicate with each other.

- More nodes can easily be added to the distributed system i.e it can be scaled as required.

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

Disadvantages:

- Excess or overloading may happen in the network if all the nodes of the distributed system try to send data at once.

- Some content, messages and data can be lost in the network while moving from one node to another.

- It is very hard and difficult to provide adequate security in distributed systems because the nodes as well as the connections need to be secured.

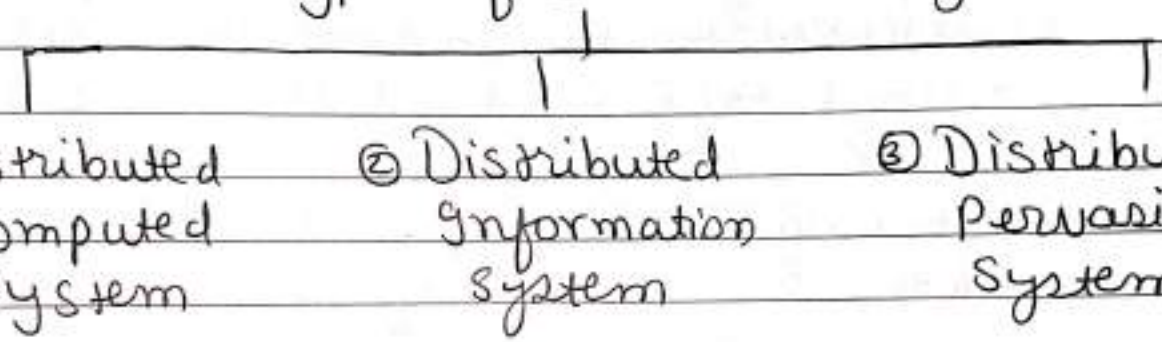
- The database connected to the distributed systems is quite complicated and complexor difficult to handle as compared to a single user system

Q5 What are the Types of Distributed System?

or

Explain different types of distributed System.

=) Based on functionality the distributed System is broadly classified into three types:




① Distributed Computing System:

The distributed system uses a group of computers that share a common computation problem among them so as to generate an efficient results in short time span

The time is an important entity in distributed computing, which can be reduced by increasing the number of computing nodes.

- The system uses special kind of software application to manage different task performed on multiple computer simultaneously.

- It helps in the interaction of different computer through the ~~exchange~~ exchange of message

- It has two subclasses cluster computing and grid computing.

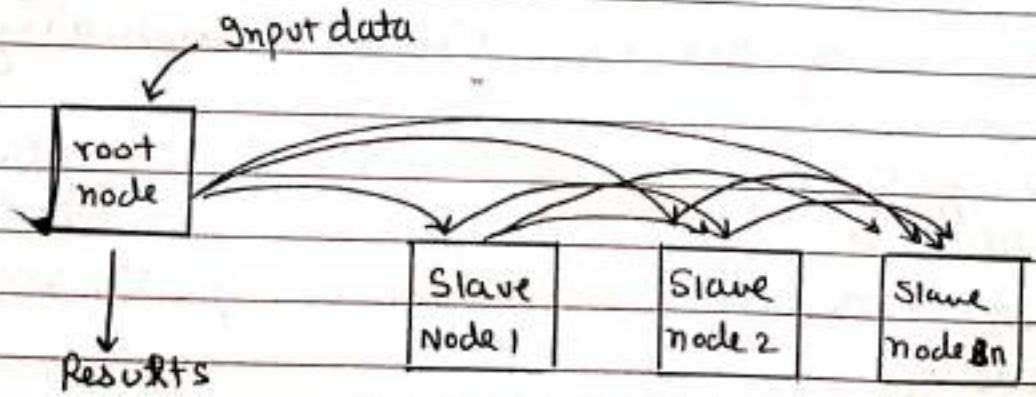
Cluster Computing:

- Cluster computing is a form of distributed computing where a group of computers are linked together in a network to perform a single task and act like single entity.

- High availability and load balancing are the key features of cluster computing.

- They ensure that the computing system will always be available by creating redundancy in the network so that the system never fails.

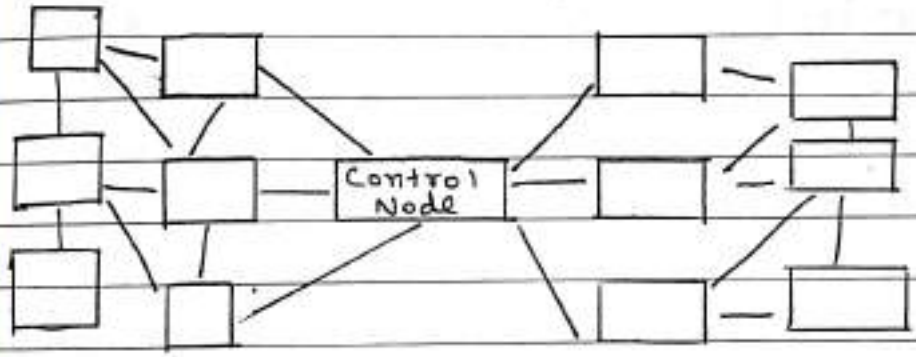
Disadvantage: It only supports Homogeneous infrastructure only. (same OS, Hardware, & Network)




Cluster Computing

## Grid Computing :

- Grid Computing is another computing form where heterogeneity is the key advantage
- It allows multiple computers with diverse hardware, Operating System and network to solve computation problem.
- All the computer in a grid network have a common goal and work on single task.




(2) Distributed Information System

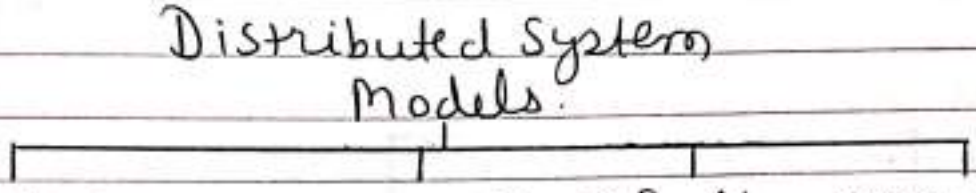
- It is used to keep track of the information about application running over the network which are distributed among various entities
- This system is primarily used for management and integration of business functions.
- TPS (Transaction processing system) and EAI (Enterprise Application Integration) are two examples of distributed information system.

## ③ Distributed Pervasive System

- It is basically used with mobile and embedded devices
- It is characterized by small battery powered device having only wireless connection. It is related to the system that has lack of human control.
- For example Home System (Connect all home device like laptop, mobile, PDA to network), sensor network (Process sensor information)

Q6 Explain Different Distributed System Models.

=> The distributed system models are used to express functionalities between the components have been categorized into four sub models.




Architecture Model:

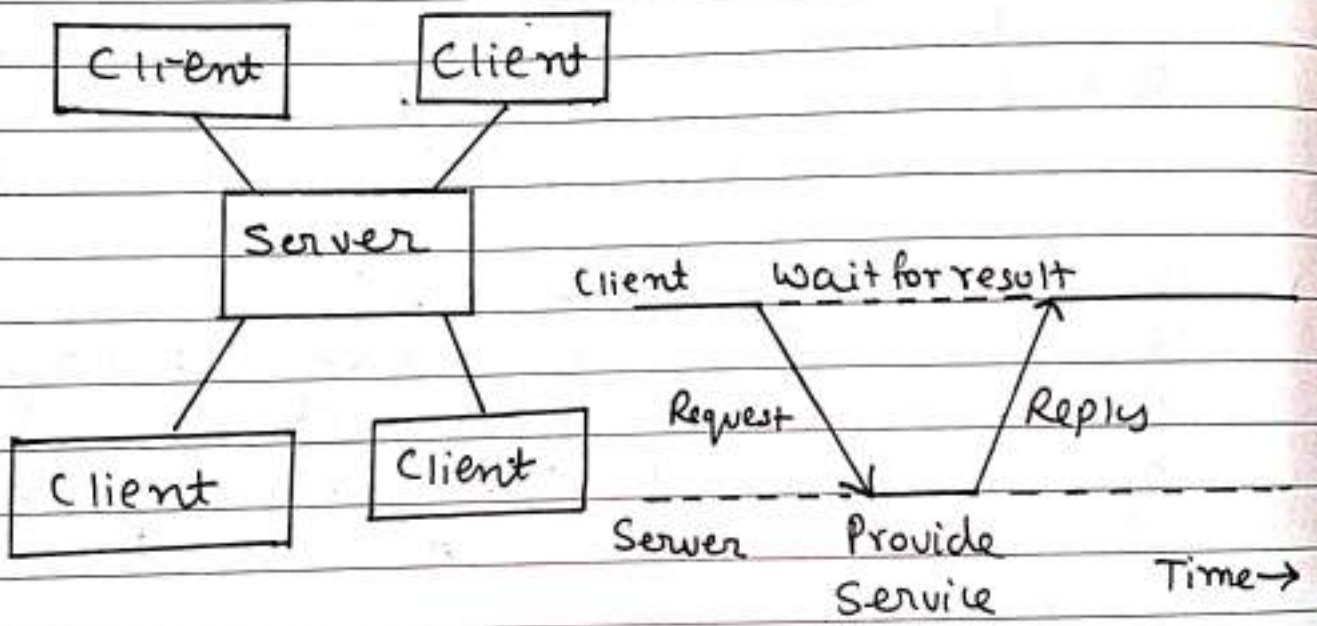
- An architecture model of distributed system defines the way in which the components of the system interact with each other and mapped onto an underlying network of computer.

- The client-server and peer to peer model are the parts of an architecture model

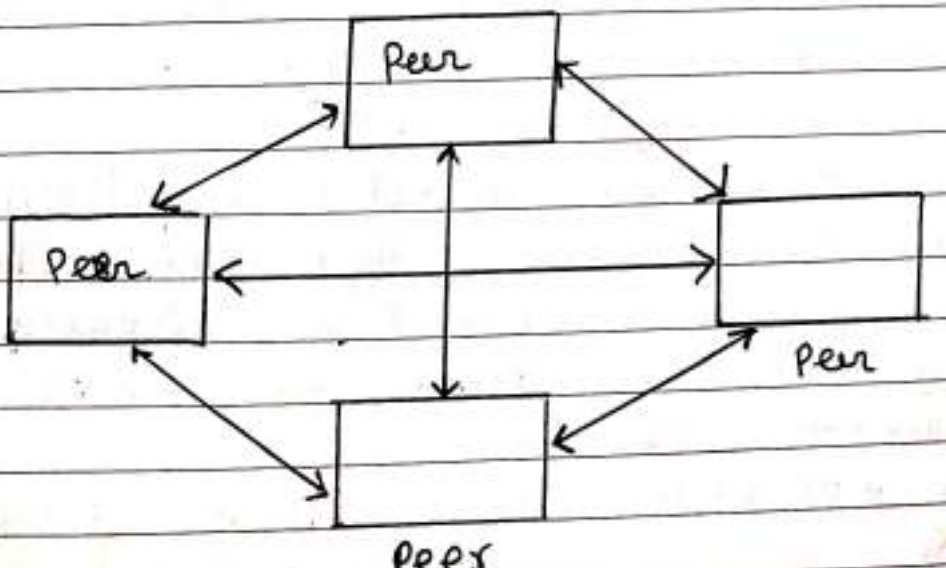
- The client-server model is widely used architecture in most of the distributed technologies, where client process interact with server process to get access the shared resources
  - use RPC / RMI for communication
  - GtS is based on request / reply protocol implemented with send / recieve primitives.

- In the peer to peer model all computers in the network get same previleges and run same program with the same set of interfaces.

- Disadvantages: P2P have lack of scalability and high complexity.




Client-Server Model



## 2. Interaction Model

- The interaction model deals with type of interaction pattern used in the communication. The two variants of the interaction model are synchronous distributed system and asynchronous distributed system.

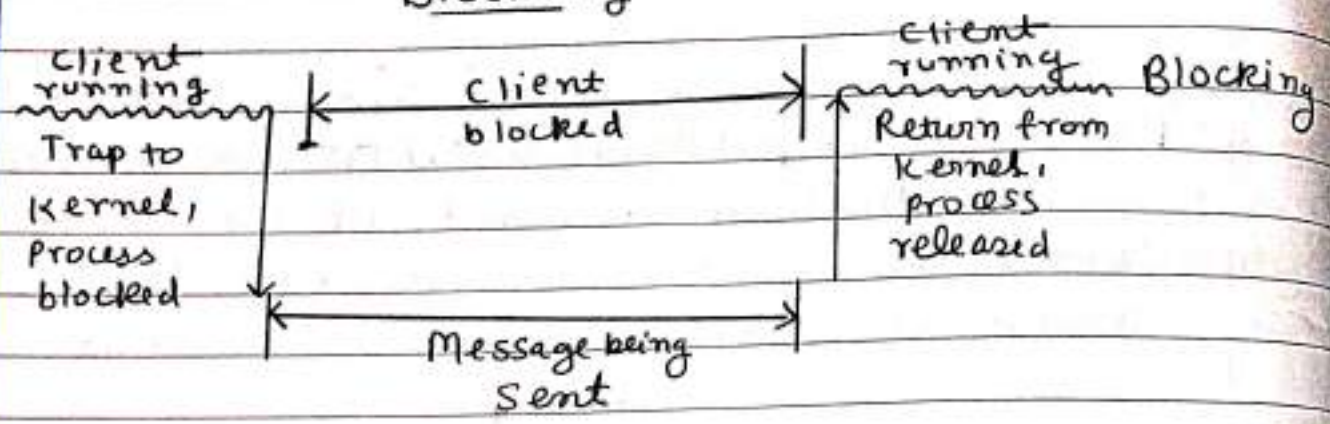
- In synchronous distributed system, both client and server should actively participate in the communication process, where the time to execute each step of a process records the lower and upper bounds.

Blocking communication (synchronous)
- Send blocks until message is actually sent
- Reciever blocks until message is actually recieved

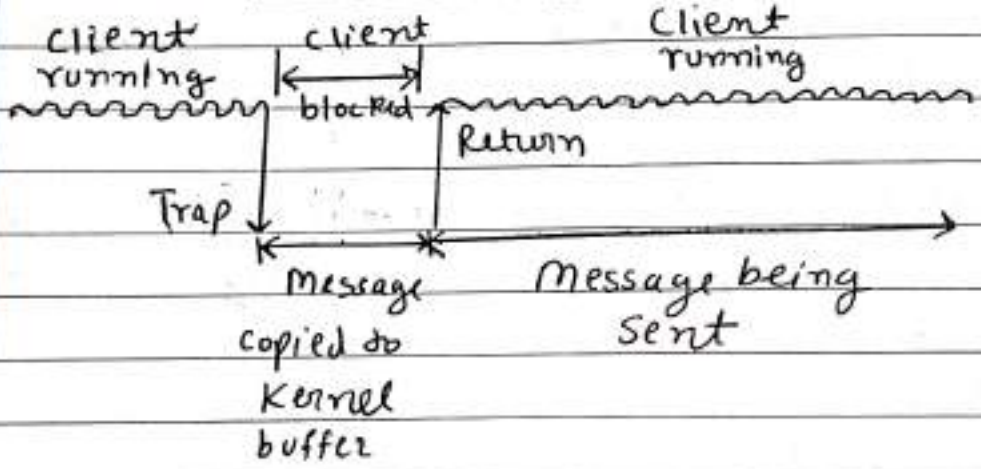
- In asynchronous distributed system the sender and reciever both need not to be activated at a time for communication because no time bound is recorded here.

Non-blocking communication (Asynchronous)
- Send return immediately
- Return does not block either.

## Blocking




## Non-Blocking.




3. fault Model & ~~Some classifications~~

- In distributed system, failure may occure with both process and communication channels

- There are three types of fault - omission fault, arbitrary fault and timing fault.
  1. Omission fault : the action is performed or omitted if fault occurs.

Arbitrary fault: Describes the worst possible fault semantics where any type of error may occur.

Timing fault: Time limit is exceeded.

4. Security Model

- The Security model is intended to provide Security to the shared resource, process and channel used for their interaction.

- It provide Security to the shared Object by means of encryption and decryption

- The various cryptographic algorithms are used for encrypting and decrypting objects and processes by using different key algorithm.

- Combination of authentication and encryptions is widely used to secure the communication Channel

Q7. Write a short Note on Hardware Concepts.

=> Hardware concepts illustrate the organization of hardware, their interconnection and the manner in which it communicate with each other. Multiple CPUs exist in distributed system

The machines are basically divided into two groups

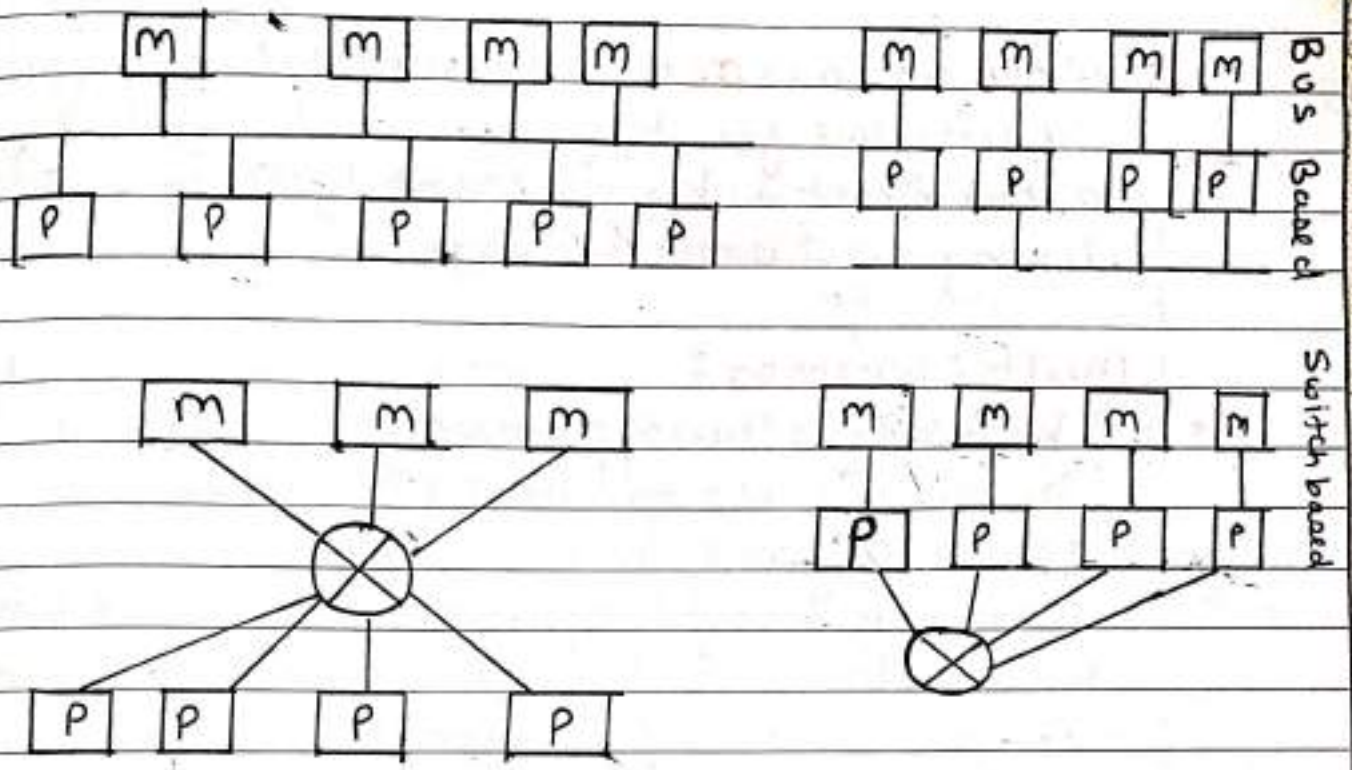
- Multiprocessor: Processors share the memory
- Multicomputer: Each processor has its own private memory

Both Multiprocessor and multicomputers further are divided into two categories on the basis of architecture of interconnection network

multicomputer

Multiprocessor
Shared Memory

Multicomputer
Private Memory




P - Processor M → Memory.

Bus-Based: Machines are connected by single cable or processors are connected by bus and share the memory by communicating over the bus.

Switch based: Machines are connected with different wiring patterns and messages are routed along outgoing line by switching the switch.

Homogeneous

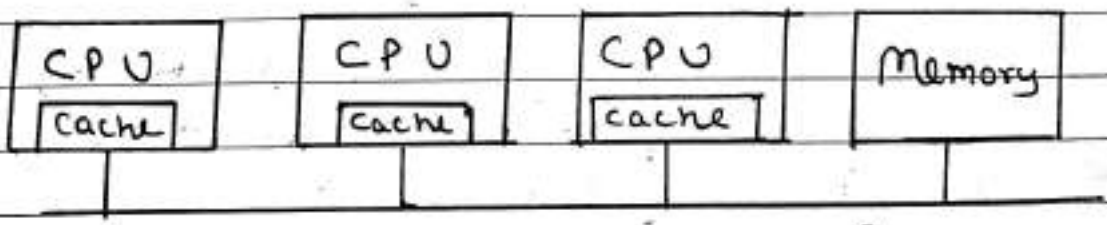
Multicomputers can be homogeneous or even heterogeneous. Homogeneous multicomputers use same technology for interconnection network

and all processors are similar, accessing the same amount of private memory

- Heterogeneous multicomputers have different architecture for different machines. Machines are connected by different types of networks using different technology

Multiprocessor:

- A bus based Multiprocessor
  - Direct access to Shared Memory
  - Connected to a common bus
  - Memory should maintain coherent property
  - With increase of CPU performance drop
  - ~~The~~ To increase performance, High-speed cache memory is used between CPU and bus
  - It store recent data and can lead to incoherent memory and need replication mechanisms.



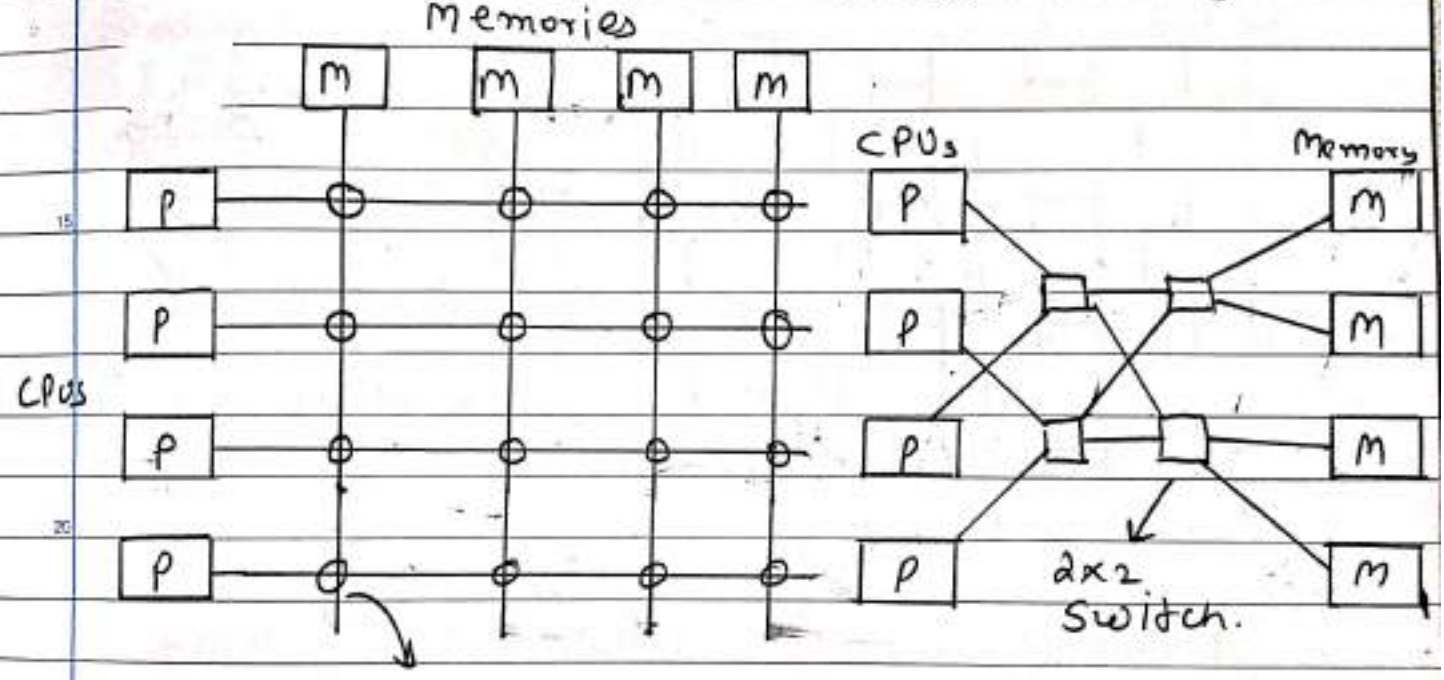

- Switch based multiprocessor
  a) A crossbar switch - Each CPU and Each memory are connected.
  - Switch closed when one CPU access the memory. Rest CPU waits for switch to open
  - For n CPU and n memory, $n^2$ switches are needed (network is limited to less value of n)

b) An omega switching network

- Network contains 2x2 switch, each having 2 input and 2 output.

- Disadvantage - There may be several switching stages between the CPU

- To reduce cost some use hierarchical system in which CPU have its own local memory.




A crossbar switch

Omega switching Network.

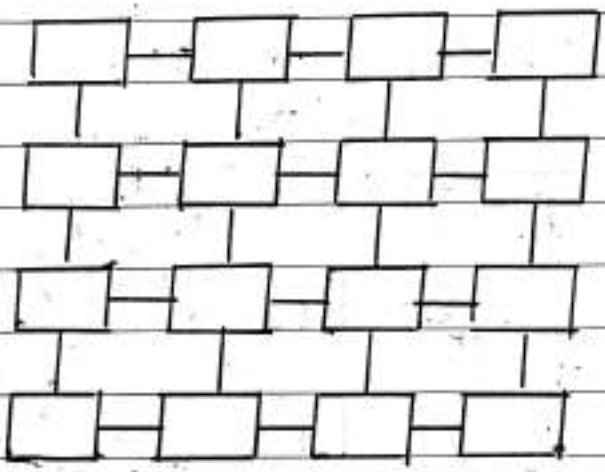
## Homogeneous Multicomputer Systems:

In CPU-CPU Communication, the volume of traffic is of lower magnitude than for CPU memory traffic.

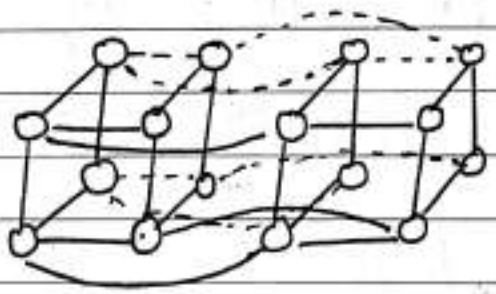
- System Area Network (SANs) - The nodes are mounted in a big rack and connected through single, high performance network.

Bus Based Multicomputer (20-100 nodes) -
Connected through shared multiaccess network such as Fast Ethernet.
(message is Broadcast)

Switch based Multicomputer - (message is routed)
a) Grid : 2D network
b) Hypercube : n Dimensional cube, each vertex is CPU



a)



b) Hypercube.

Heterogeneous Multicomputer Systems.

- Today most of the distributed systems are built on top of multicomputers having different processors, memory size etc. Interconnection network also have a different technology. These are called as Heterogeneous multicomputers.

Q8 Explain Software concepts.

=> To make Distributed Hardware work distributed based operating System is required

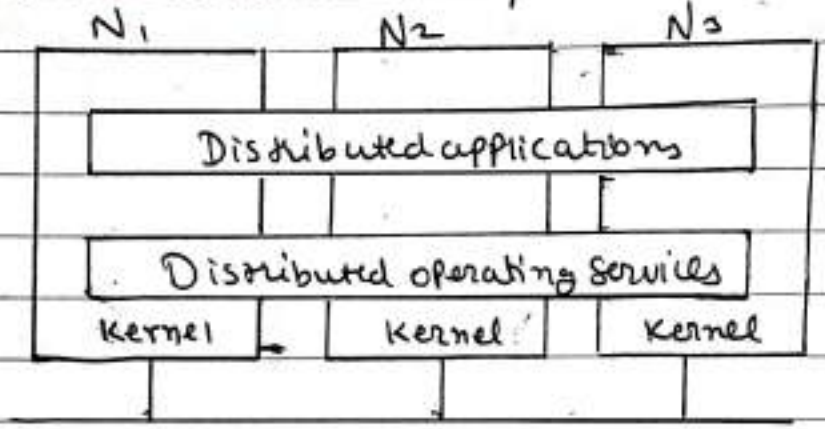
| parameters | Distributed operating System | Network operating System |
|---|---|---|
| System Image | Gives view of virtual uni process | Gives view of multiprocessor |
| | High Transparency | No Transparency |
| | Good System Image | Bad System Image |
| Autonomy | Same OS on all the nodes | Different OS on all the nodes |
| | Homogeneous env is created | Heterogenous env is created |
| | Bad | ⇒ Good |
| fault Tolerance | Migration of jobs easier due to Same OS | Migration of job difficult due to different OS |

can also add this...

## Distributed operating system:

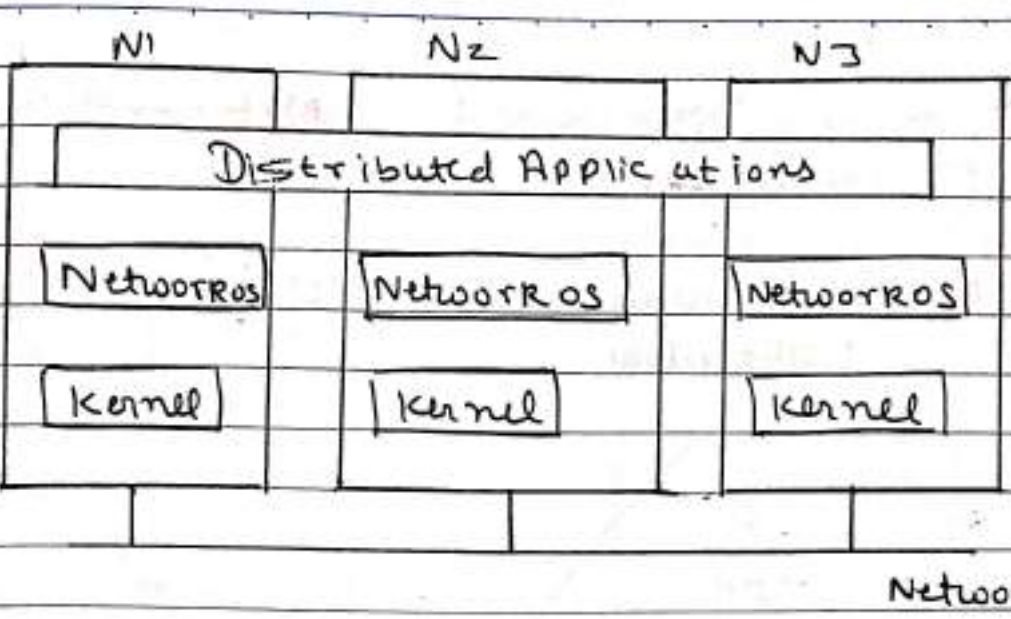
Tightly coupled OS for multiprocessors and homogeneous multicomputers.
Goal: To hide and manage services.

Characteristics:
- OS knows about other computers
- OS are same generally
- Services are transparent accross



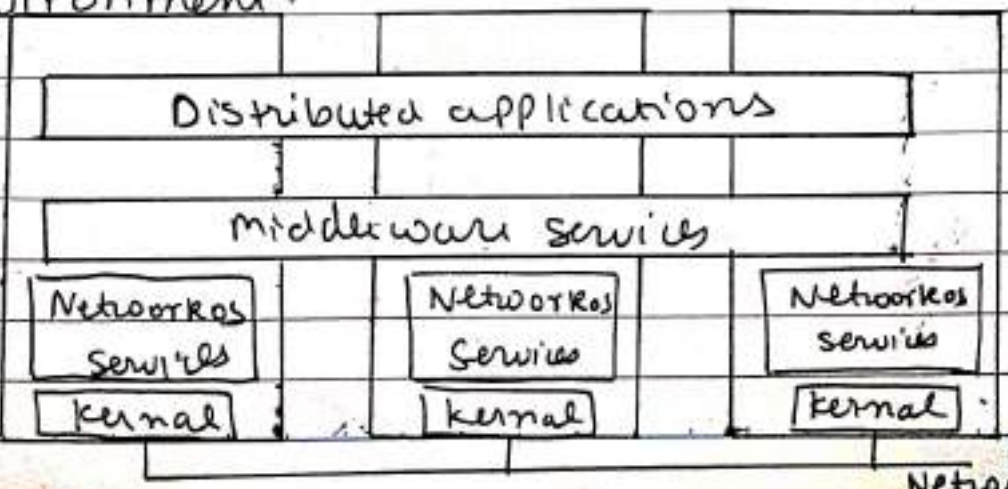

## Network OS:
- Simply losely coupled OS for heterogeneous multicomputers (LAN & WAN)
- Goal: To offer local services to remote clients.

Characteristics
- Each computer has its own OS and network Services.
- Each work independently
- Services tied on nodes (ftp, telnet)
- file-oriented.




Middleware Operating System

- Additional layer at top of networkOS
- Goal: To provide distribution transparency
- Services
  - communication services
  - information system services
  - control services
  - Security services.
- Neither DOS nor NOS qualifies as a good OS for Distributed Environment
- Hence a concept of middleware is developed to combine features of both DOS & NOS
- The middleware is called Distributed computing environment.



Q8 Compare Distributed OS, Network OS and middleware OS

| | Item | Distributed OS | | Network OS | Middleware OS |
|---|---|---|---|---|---|
| | | Multiprocessor | Multicomputer | | |
| 1. | Degree of Transparency | Very High | High | Low | High |
| 2. | Same os on all nodes | Yes | Yes | No | No |
| 3. | Number of copies of OS | 1 | N (multiple) | N (Multiple) | N (multiple) |
| 4 | Basis of Communication | Shared memory | Messages | File | Model specific |
| 5. | Resource Management | Global, Central | Global, distributed | Per node | Per node |
| 6. | Scalability | No | Moderately | Yes | Varies |
| 7. | Openness | Closed | Closed | open | open |

Q9. What are the Different ~~Distributed system~~ middleware Models?

=> 1] Remote Procedure Call

- Allows a process executing on one computer to invoke a procedure in a process executing on another computer

- Goal of RPC - TO simplify the process of writing distributed applications by preserving the syntax of a local procedure call while transparently initiating network communication

2) Remote method Invocation

- Enables a java process executing on one computer to invoke a method of an object on a remote computer using the same syntax as a local method call.

- Similar to RPC, the details of parameter marshaling and message transport in RMI are transparent to the calling program

- The stub/skeleton layer of RMI contains parameter-marshaling structures also analogous to the client and server stubs of RPC.

3) Common object request broker architecture (CORBA)

- Open Standard designed to enable interoperation among programs in heterogeneous as well as homogeneous systems

- supports Objects as parameters or return values in remote procedures during interprocess communication.

4) Distributed component object modeling (DCOM)

- Designed to allow software components residing on remote computers to interact with one another.

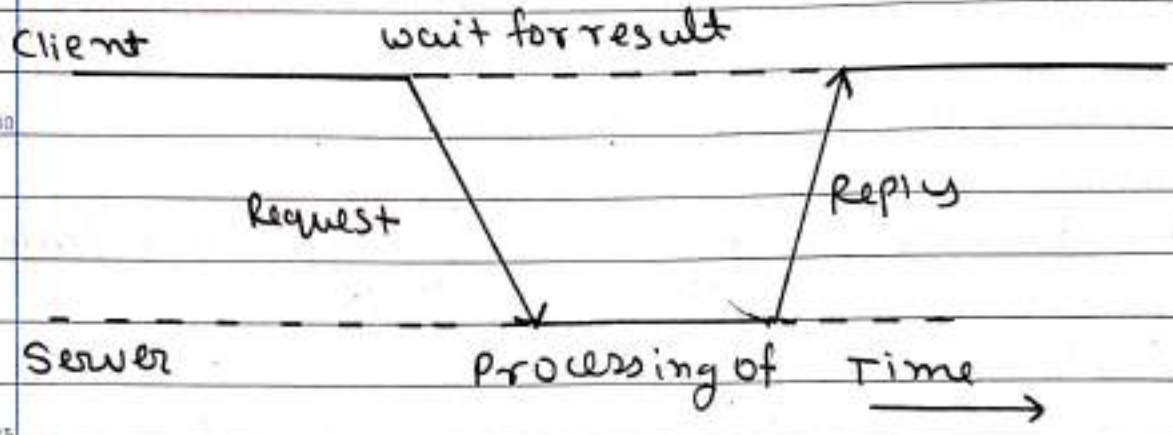
- As in CORBA, objects in DCOM are accessed via interfaces

- Unlike CORBA, however, DCOM objects may have multiple interfaces.

- When a client request a DCOM object from a server the client must also request a specific interface of the object.

Q10 Write a Short Note on client Server Model

=> The system is structured as a set of processes called servers, that offers Services to the Users called client.




Request Reply based interaction between client & server

- It is based on simple request/reply protocol
- The client sends a request message to the Server asking for service.
- The Server does the work and returns a result or an error code or message if the code could not be performed
- The Server can itself request Services from other server thus in this new relation, the Server acts like a client.

o Application layering.

Client-server application is partitioned between following three layers:

1. User Interface level: This level is implemented in clients. This level comprises the code that permits the end user to interact with application. The simple user interface program is character based screen.

- Graphical displays with pop-up and pull down menus can be used at client side. X windows interfaces is one of the example

2. The processing level:

- The core part of the application which operates on database or file system is considered at processing level.

- In search engines example, user type keyword to be searched in one user interface

- At backend there is huge database of web pages. The core logic here is to convert the typed keywords by user to database queries. Then ranking the results into list and transforming the list in html pages.

3 Data level

- In client server model, this level has programs which maintain the data onto which applications perform the operations. This data is very persistent.

- Although Application is not running, still the data remains stored at somewhere.

- The above discussed three levels suggest the different ways to distribute client server application across different machines in the distributed system.

- The simple organization is to keep interface level on client machine other two level on server machine

Multitiered Architecture :

~~Terminal dependent part of the browser~~

Following are the Different possibilities of two tier architecture:

- Terminal dependent part of the User interface on client machine and User interface application and database on server machine.

- Place entire User interface on client machine and Application and Database on server machine

- Place entire UI ~~or~~ and some part of App$^n$ (frontend) on client machine and other part of App$^n$ and database on server machine. Validation of filled forms is done at client side

- Place entire user interface and entire application on client machine and database on server machine. In this case client contact only for operations of data to server

- Place entire UI and entire App$^n$ and some part of DB ~~on client machine and~~ + (for example cached web pages on local disk) on client machine and Data base on server machine.

Modern Architecture:

- One way is vertical distribution where logically different components are placed on different machines

- In Horizontal distribution, client or server physically splits up in logical parts.

- for example web pages are distributed among many servers and client request is forwarded to those server in roundrobin fashion.

## Module 2: Communication

Q1 Write a short note on layered protocols: (ISO OSI model)

or

Explain ISO-OSI reference model with diagram?

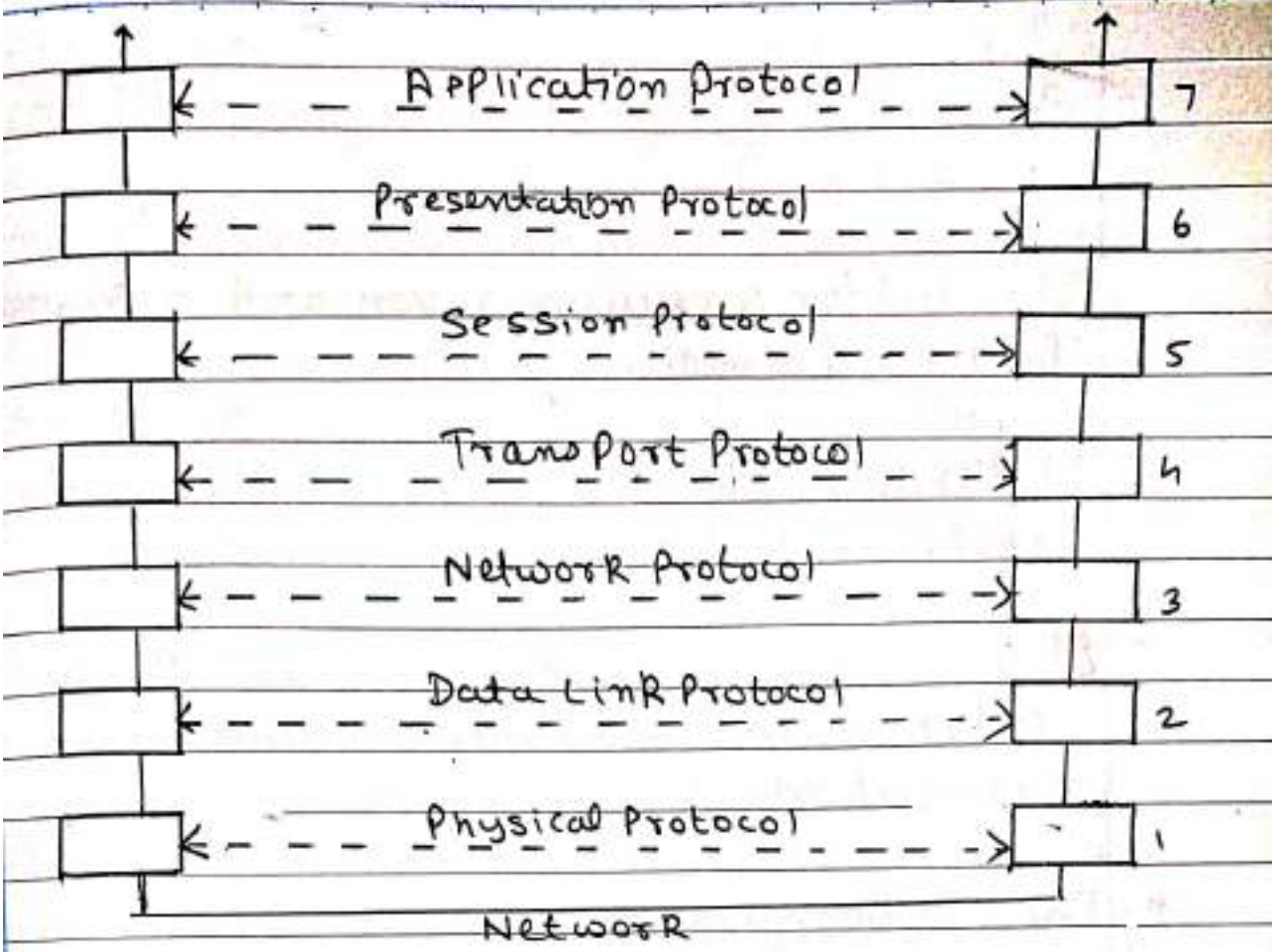
=> The users of a computer network are located over a wide physical range ie all over the world (world wide)

- Therefore to ensure that nationwide and worldwide data communication systems can be developed and are compatible to each other an international group of standards has been developed totally.

- These standards will fit into framework which has been developed by the "International organization of Standardization" (ISO).

- This framework is called OSI Model (Open system interconnection model)

- OSI model has 7 layers.




**Physical Layer :**

- It is responsible to activate, maintain and deactivate the physical connection.
- It defines voltages and data rate needed for transmission.
- It converts digital data bits into electrical signal
- It decides whether the transmission is simplex, half duplex or full duplex.
- It does not detect or correct errors

**Data Link Layer :**

- It is responsible for Node-to-node delivery of data.
- It does receives the data from network layer and creates FRAMES, add physical address the these frames & pass them to physical layer.

- It consists of Logical Link Layer & Medium access control
- It is used for synchronization and error control for the information.
- It divides the bits recieved from Network layer into frames and it provides Flow control
- To enable the error detection, it adds error detection bits to the data which is to be transmitted.

o The Network Layer

- It route the signs through various channels to the other end
- It act as the network controller by deciding which route data should take,
- It divide the outgoing messages into packets and assemble incomming packets into messages for the higher levels.

o Transport Layer

- It decides if the data transmission should take place on parallel paths or single path.

- It does the functions such as multiplexing, Splitting or Segmenting on data.

- It guarentees transmission of data from one end to the other.

- It breaks the data groups into smaller units so that they are handled more efficiently by the network layer.

o The Session Layer:

- The layer manages and synchronizes conversations between two different applications. This is the level at which the user will establish System to System connection.

- It controls logging on and off, user identification, billing and session management.

- In the transmission of data from one system to the other, at Session layer streams of data are marked and resynchronized properly so that the end of messages are not cut prematurely and data loss is avoided

o The presentation Layer:

- It is concerned with the syntax and Semantics of the information exchanged between the two devices. It also works as translating layer.

- It is used for data encryption, decryption and compression.
- It ensures that the information delivered at reciever end is understandable and usable
- It is used for data presentation or translation of form and syntax.
  Example: ASCII and to EBCDIC and vice versa also.

## Application Layer

- It allows the user to access the network.
- It provides user interface, and services like manipulation of information in various ways, retransferring the file of information, distributing the results etc. to the user who is sitting above this layer

Q2 Write a short note on Interprocess Communication

=> Types of communication :

1. Synchronous Communication : Here client application waits after sending request until reply & sent request coms from server application. Example is Remote Procedure Calls (RPC), Remote method Invocation (RMI).

2. Asynchronous communication : Here, Client application continues other work after sending request until reply of sent request comes from server application. Remote Procedure call (RPC) can be implemented with this communication type. Other examples can be transferring amount from one account to other, updating database entries etc.

3. Transiant Communication : Here sender application and reciever application both should be running to deliver the messages sent between them. Example is Remote Procedure call (RPC), Remote Method Invocation (RMI).

4. Persistent communication : Here, either sender application or reciever application both need not be running to deliver the messages sent between them. Example: Email.

# Message Passing Interface:

Message Passing Interface is specification for users and programmers for message passing libraries. MPI basically is developed for supporting the parallel applications and adapted to transient communication where sender and reciever should be active during communication.

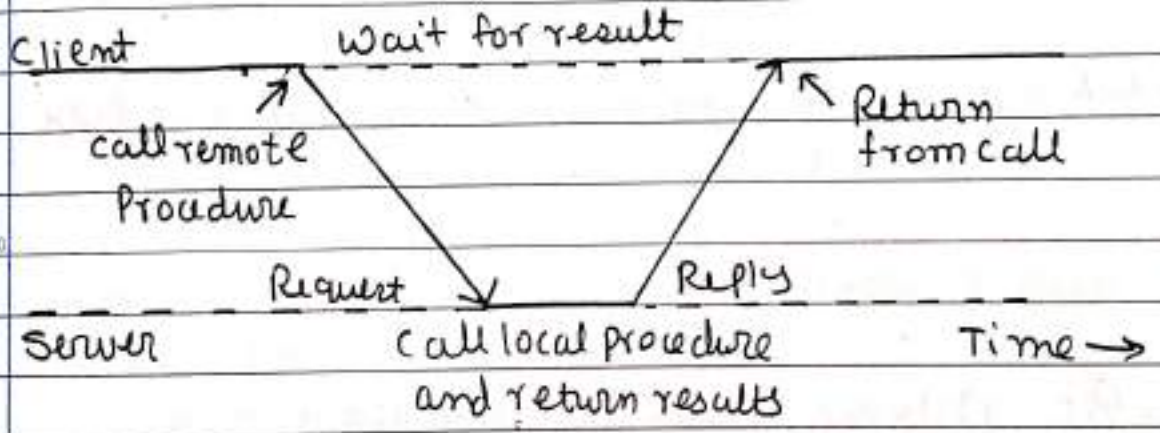
Message Passing Interface assumes process crash or network failure as incurable and these failures do not requir recovery action. MPI also assumes that known group of the processes establishes communi-cation between them. To recognize process group, GroupID is used and for process, processID is used. These are identifiers and the pair exclusively identifies the sender and reciever of the message. This pair of identifiers excludes using the transport level of address.

Q3 Write a short note on Remote Procedure Call.

=> Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network details.

- A procedure call is also called a function call or a subroutine call.

- RPC Uses client-server model:




- It includes mainly 5 elements:
  - The client
  - The client stub (Stub: Piece of code used for converting the parameters)
  - The RPC Runtime (RPC communication package)
  - The server Stub
  - The server.

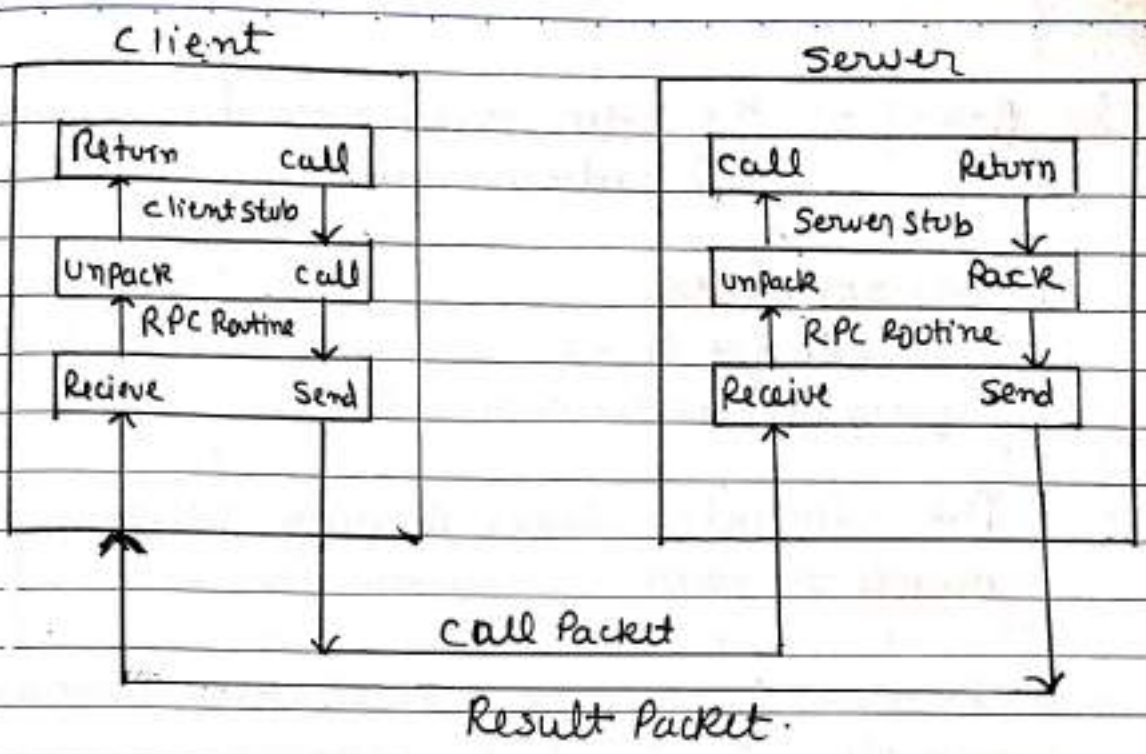
- The client:
  - It is used to process which initiates a RPC
  - The client make a perfectly normal call that invokes a corresponding procedure in the client stub.

- The client stub.
  - On receipt of a request it packs a requirment into a message and ask to RPC Runtime to Send.
  - On receipt of a result it unpacks the result and passes it to client.

- The RPC ~~Routi~~ Runtime
  - It handle transmission of message between client and server

- The Server stub
  - It unpack a call request and make a perfectly normal call to invoke the appropriate procedure in the server.
  - On receipt of a result a ~~proc~~ procedure ~~execut~~ it pack the result and asks to RPC Runtime to send.

- The server
  - It executes an appropriate procedure and returns the result from a server stub.




## Remote Procedure call

### Steps of a Remote Procedure call

1. Client procedure calls client stub in normal way.
2. Client stub builds message, calls local OS
3. Client's OS sends message do remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result do the stub
7. Server stub packs it in message, calls local OS
8. Server's OS send message do client's OS
9. Client's OS gives message do client stub
10. Stub unpacks result, returns to client.

Q4 What are the Advantages and Disadvantages of Remote Procedure Call.

=> Advantages:

- The effort to re-write and re-develop the code is minimum in RPC.
- The internal message passing mechanism of RPC is hidden from the user.
- RPC can be used in distributed environment as well as the local environment.
- RPC support process oriented and thread oriented models.

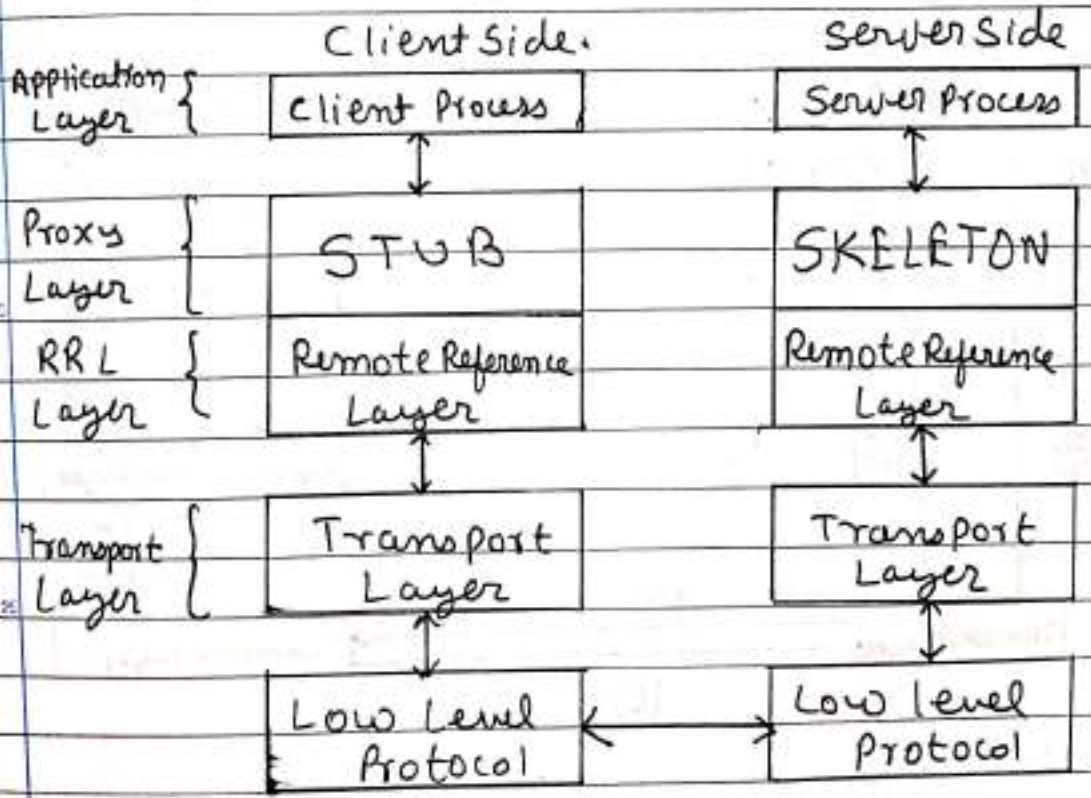
Disadvantages:

- There is an increase in Price because of the remote procedure call
- There is no flexibility in RPC for hardware architecture. It is only and Only Interaction based.
- RPC is a concept that can be totally implemented in different ways. It is not at all a standard.

Q5 Write a short note on Remote Method Invocation

=> It is a set of protocols being developed by Sun's Javasoft division that enables Java objects to communicate remotely with other Java objects

- RMI is relatively simple protocol, but unlike more protocols such as CORBA and DCOM, it works only with java objects. CORBA and DCOM are designed to support objects created in any language.

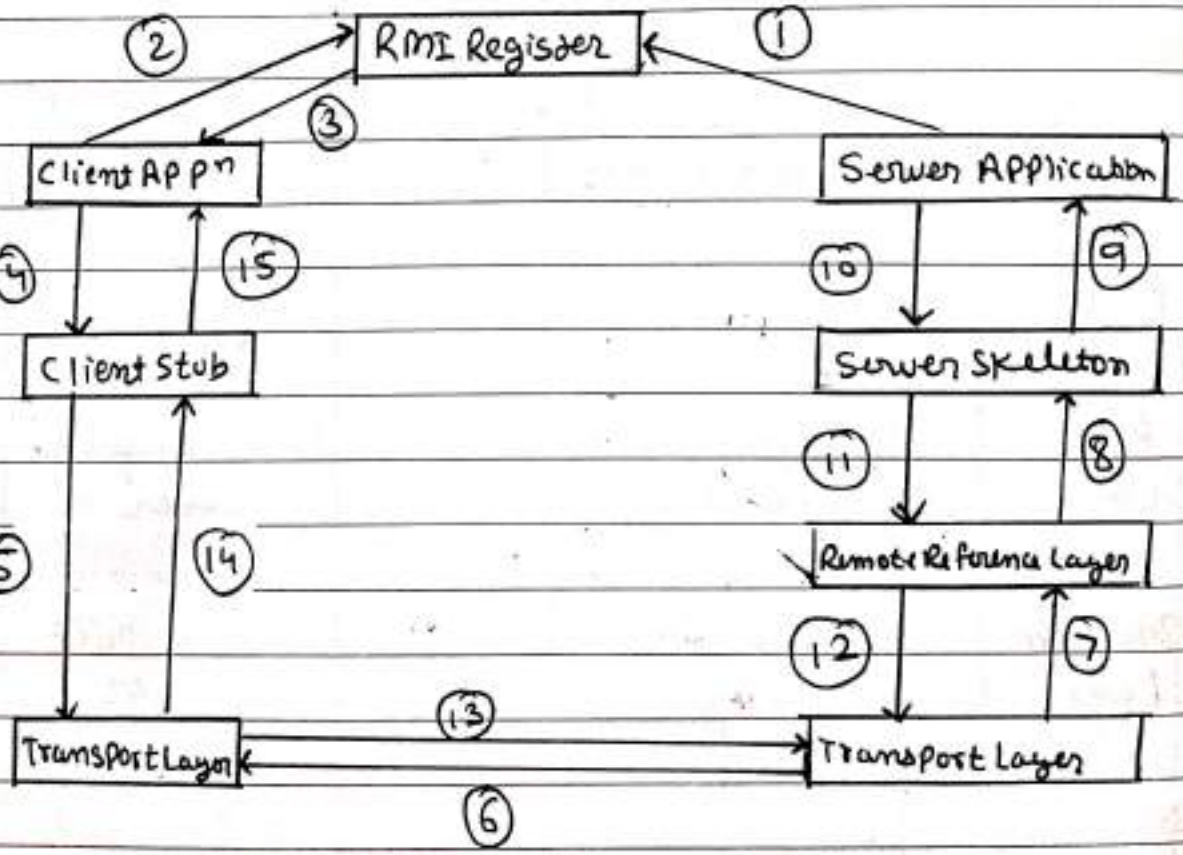



Remote Procedure call.

# Remote Method Invocation Goals

- Seamless object remote invocations
- Callbacks from server to client
- Distributed garbage collection
- NOTE : in RMI, all objects must be written in JAVA.

## Remote Method Invocation Architecture



1. Server registers a remote object bound to a name
2. Client lookup for object reference of called method in registry
3. The registry returns an instance of remote object.
4. creates clients stub of requested method
5. Forword client stub to transport layer.
6. Parameters
7. Forward client stub to remote reference layer
8. RRL checks the semantics and forward client stub to skeleton
9. Server unpacks stub and forwards data to called method.
10. Server pack result into skeleton
11. Skeleton forwarded to transport layer
12. Forward Skeleton to transport Layer
13. Result.
14. Skelton passes to client Stub
15. Unpacks Skeleton and sends results to client application

Q6 Write a Short Note on Message Oriented Communication.

=> Message oriented communication is the way of communication between the processes on the sender's and recievers machine, in terms of Sending and recieving explicit message between them.

Classification of Communication

1. Synchronous Communication:
- The Sender is blocked until its message is stored in local buffer at the reciever end or the message has been delivered
- The Strongest form of synchronous communication is when the sender can only continue executing after the reciever process the message.

2. ASynchronous Communication:
- Sender continues Immediately after message sent
- The message is either stored in local buffer or at the first communication Server
- Example: asynchronous - The answering machine

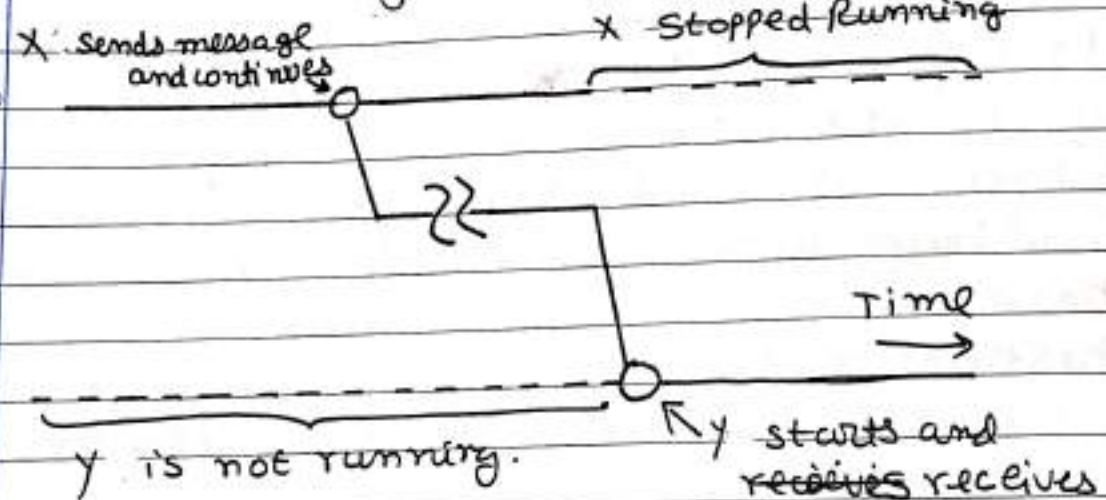
3. Persistent Communication:
- Message that has been submitted is stored by the communication System as long as it takes to deliver it to reciver.
- Example: Mail.

4. Transient Communication:
- Messages are stored as long as sending and recieving applications are running.

- The message will be discarded if the communication server cannot deliver the message to destination server.
- Example: Router.

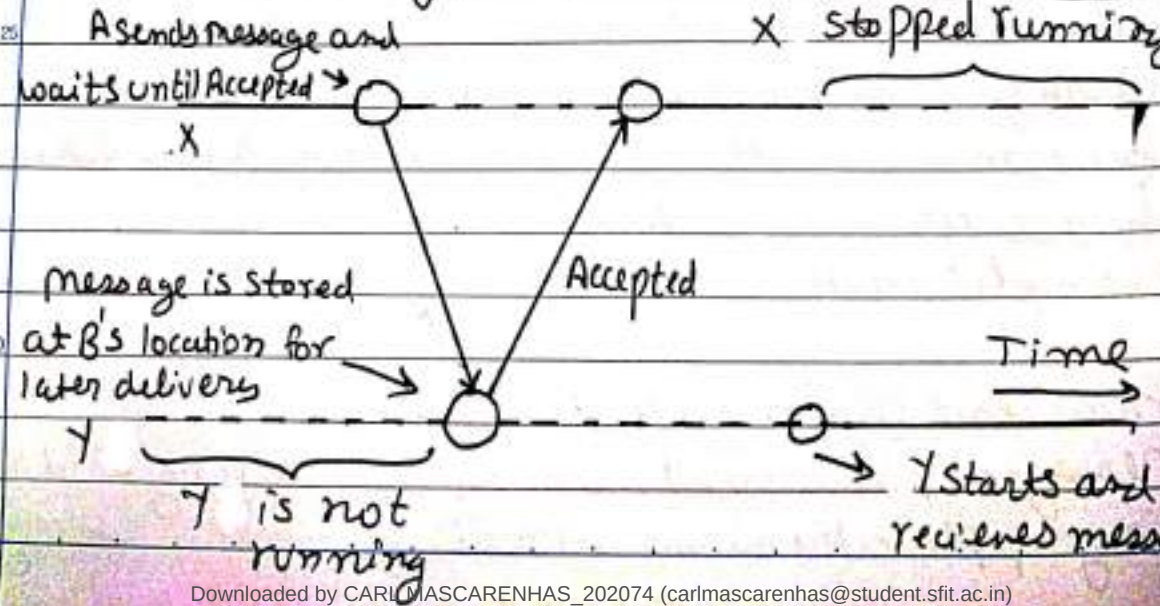
o Types of communication

a) Persistent asynchronous communication



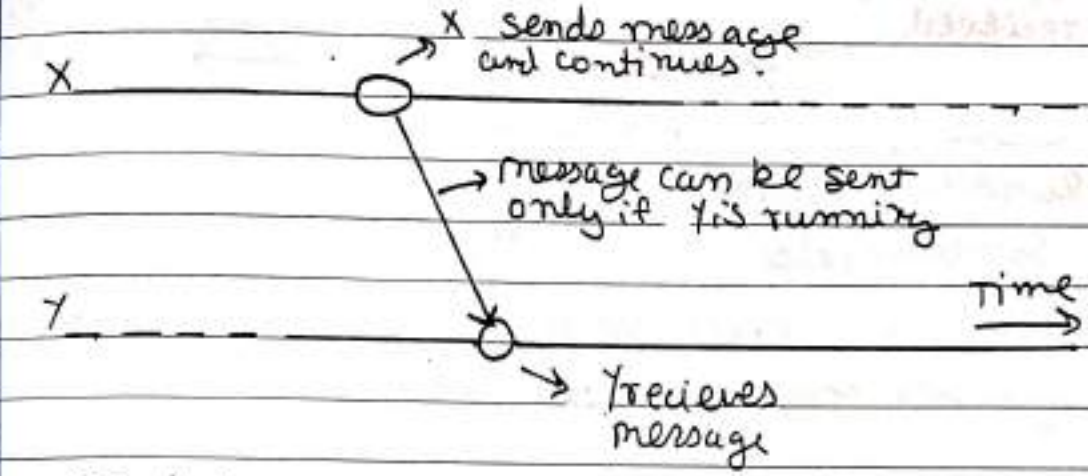

→ Here Message is either stored in the buffer at the local host or the communication server.
Example: Email

b) Persistent Synchronous communication



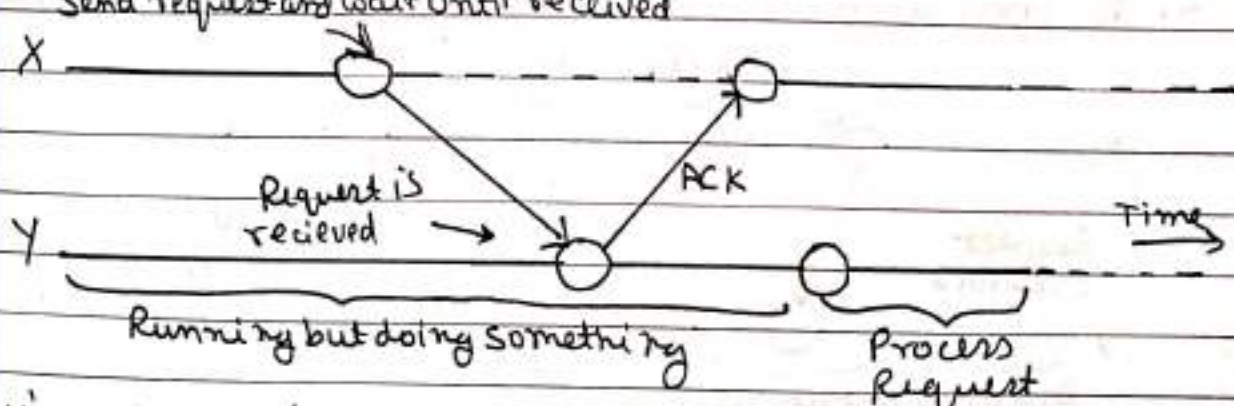
- Here messages are stored at the recieving host.
- X sender is blocked totally until its message is stored in receiver's host
- It is not necessary that receiving application should be running.

C. Transient Asynchronous communication.



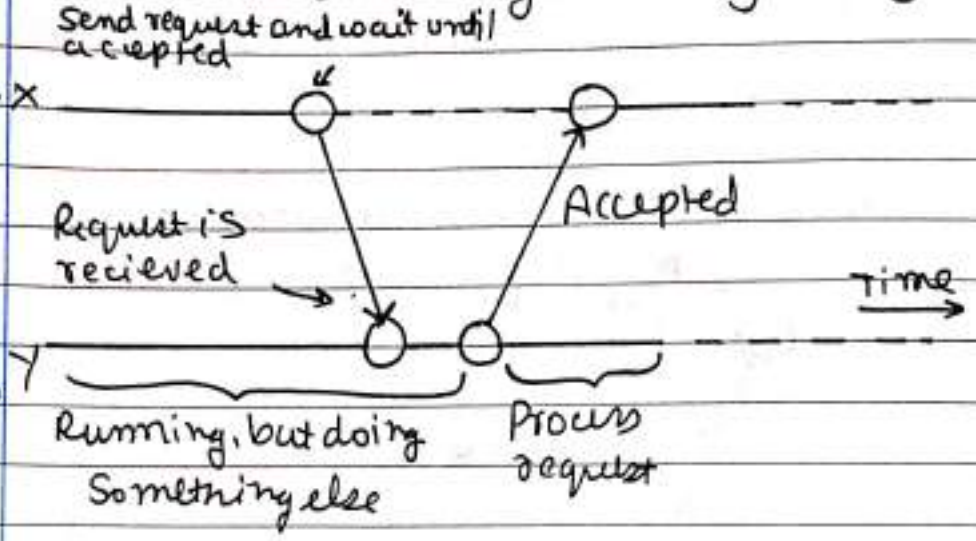

- X sends the message and continues execution.
- Y has to be running, because if it is not running the message will be discarded
- Even if any router along the way is down, the message will be discarded. Example: UDP communication

D. Reciept - based Transient Synchronous communication



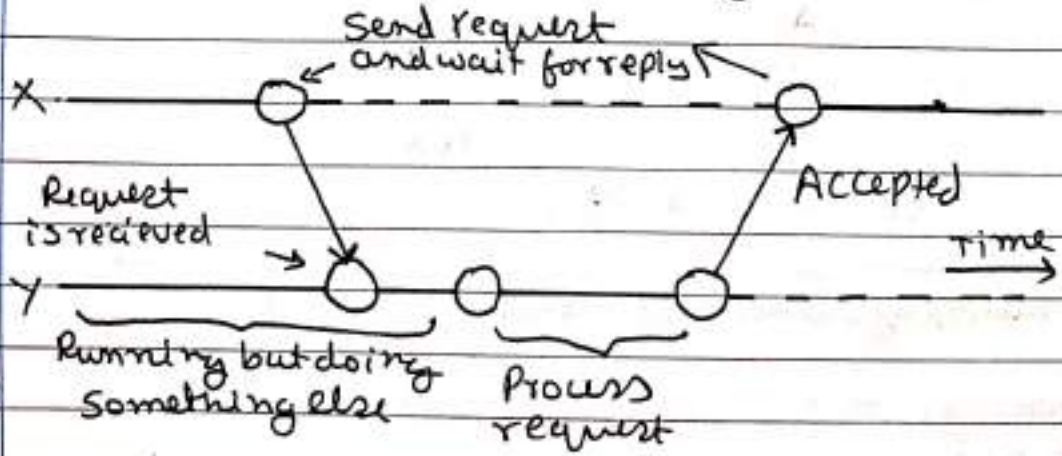

- X's message to Y is blocking (Synchronous) until an ack is recieved.
- This act simply tells us that the message was recieved at the other end. It does not tells us anything about wether the process has started.

e. Delivery-based transient synchronous communication at message delivery (asynchronous RPC)




- This is an extension of reciept-based transient synchronous communication
- X will resume when Y takes the delivery of the message
- The ack comes a little bit later than previous method
- This is essentially asynchronous RPC because from the perspective of an RPC, we are not blocking for the reply.

f. Response-based transient synchronous communication




- X resumes execution upon recliving a response

- That is, not only has the message been delivered, but it has been processed and the ack comes back in the form of a reply
- This is traditional RPC. The client is blocked for the entire duration the reply comes back.

Q7. Write a short Note on Stream oriented communication?

=> Stream oriented communication is a type of communication where Timing plays a very crucial role. Example: Audio Stream

(Few Points you can write in exam are covered in the next question called difference between message oriented communication and Stream oriented communication)

- Different Transmission modes:

1. Asynchronous transmission mode:
The data items in a stream are transmitted one after the other, but there are no further timing constraints on when transmission of items should take place.
- for example: a file transfered as a data stream.

2. Synchronous transmission mode:
There is a maximum end to end delay ~~behind~~ defined for each unit in a data stream. wether a data unit is transfered much faster than the maximum tolerated delay is not important.

- for example a sensor may sample temperature at a certain rate and pass it through a network to an operator. Here it may be important

that the end to end propagation time through the network is gen guaranteed to be lower than the time interval between taking samples, but it cannot do any harm if samples are propagated much faster than necessary.

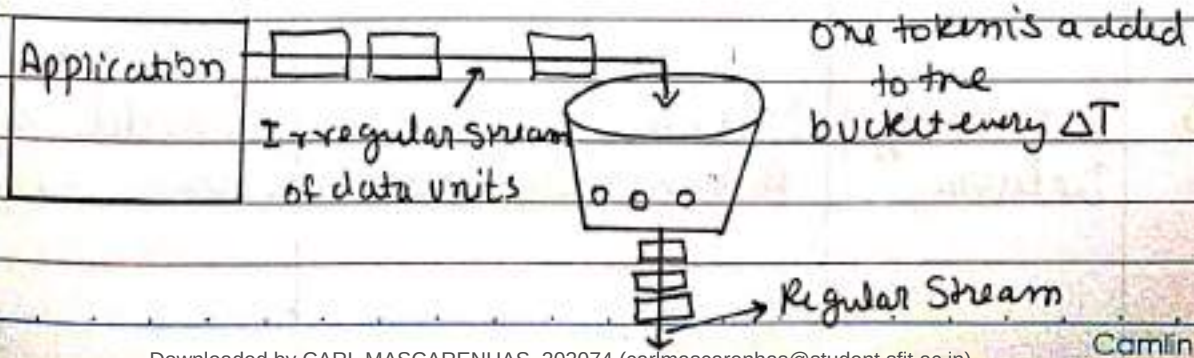
3. Isochronous Transmission mode:

It is necessary that data units are transferred on time. This means that data transfer is subject to a maximum and minimum end to end delay.

- Plays a crucial role in representing audio and video

Example:
- Token Bucket Algorithm

- A token Bucket can be easily implemented with a counter.
- The token is initialized to 0.
- Each time a token is added the counter is increased or incremented by 1
- Each time a unit of data is dispatched, the counter is decremented by one
- If the counter contains 0, the host cannot send any data.



Q8 Difference between message oriented communication and stream oriented communication.

=>

| parameters | Message oriented Communication | Stream oriented Communication |
|---|---|---|
| 1. Suitability | Suitable for app like audio, video where speed is critical than loss of message | Suitable for apps like email system where data must be persistent though delivered late |
| 2. Speed | Transmission speed is very high as compared to Stream oriented communication | Transmission speed is very low as compared to message oriented communication |
| 3. Retransmission | It is not performed | It is performed automatically in case of frameloss |
| 4. Communication | It is connectionless, data is sent without any setup | It is connection oriented, connection is established before communication |
| 5. ACKNOledged Delivery | Data delivery without Acknowledgement | Data delivery with acknowledgement |

| parameters | Message oriented communication | Stream oriented Communication |
|---|---|---|
| 6. Relibility | Unreliable | Reliable |
| 7. Overhead | Low overhead | High overhead |
| 8. Flow control | No flow control | Flow control is performed |
| 9. Protocol | It is used by user datagram protocol (UDP) | It is used by Transmission control protocol (TCP) |
| 10 Data. | Data is sent by applications in discrete packages called messages | Data is sent with no particular structure. |

Q9 : Write a short Note on Group Communication

=> A Group is the collection of users sharing some common interest.
- In group communication, messages sent to a group of processes will deliver to all the members of the group.

o Modes of communication

1. Unicast
- one to one communication or one to one message is sent

2. Anycast
- one to nearest 1 of several identical nodes the message is sent.

3. Netcast
- One to many but one at a time the message is sent.

4. Multicast
- one to many message is sent

5. Broadcast
- One to all the message is sent

# Types of group communications

1. One to Many (single sender and multiple recievers) (Multicast)

Example: Multicast and broadcast one to many communication includes:

a) Group Management
- Reciever Processes of a message from group. Such group are of two type ?
- closed group vs open group
- closed group is one in which only members of the group can send message to the group
- open group is one in which any process in the system can send a message to the group as a whole
- peer vs Hierarchical
peer: each member communicate with the group. advantage: less failure and disadvantage: complex process
- Hierarchical: Coordinator is the decision-maker and this is the reason why its decision making ability is simplified.

b) Buffered and Un buffered Multicast
- Multicast is asynchronous communication
  - Unrealistic to wait
  - Unaware of processes in multicast group
- In an unbuffered multicast, the message is not buffered for receiving process and it lost if the receiving process is not ready
- Buffered multicast; the message is buffered for the receiving processes.

c) Send-to-all and Bulletin-board Semantics
- Send-to-all Semantics: A copy of the message is sent to each process of the multicast group and the message is buffered until it is accepted by the process.

- Bulletin-board semantics: A message to be multicast is address to a channel instead of being sent to every individual process. Receiver can access to the message until it is available on bulletin.

d) Flexible reliability in multicast communication
- 0 reliable: No response is expected by the sender
- 1 reliable: The sender expects atleast one acknowledgement.
- M out of N reliable: The sender decides as to how many acknowledgements it expects out of N number of acknowledgements.

e) Atomic Multicast:
- It is either received to all or not received by any
- Degree of reliability should be all reliable (used retransmission technique)

2. Many to one (Multiple sender and one receiver)
- Many sender sends message to selective receiver
- Receiver may be selective or non selective
- Selective receiver specifies a unique sender, a message exchange takes place only if that sender sends a message.

- Nonselective receiver specifies a set of senders, and if any one sender in the set sends a message to this receiver a message exchange take place.

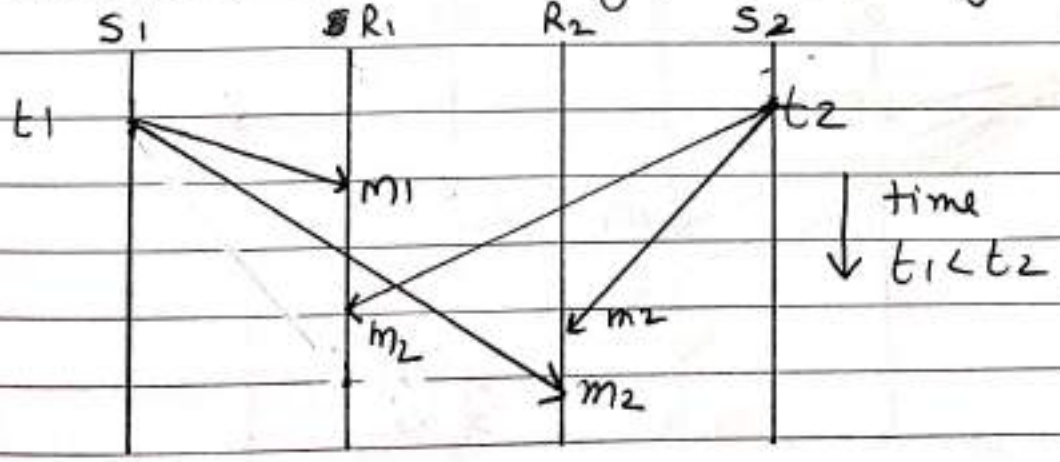
3. Many to Many ( Multiple sender and Multiple receiver)

- Many sender sends message to many receiver
- Important issue is ordered message delivery which ensures that all message are delivered to all receiver in an order acceptable to the application.

Semantics Ordered message delivery are
- Absolute ordering
- Consistent ordering
- ~~Ca~~ causal ordering.

- Absolute Ordering :

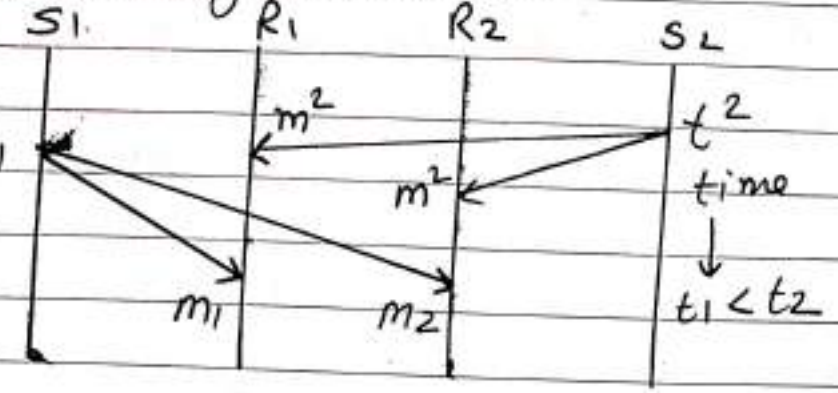
It ensures that all message delivered to all receiver in exact way in which they are sent.




- Consistent ordering

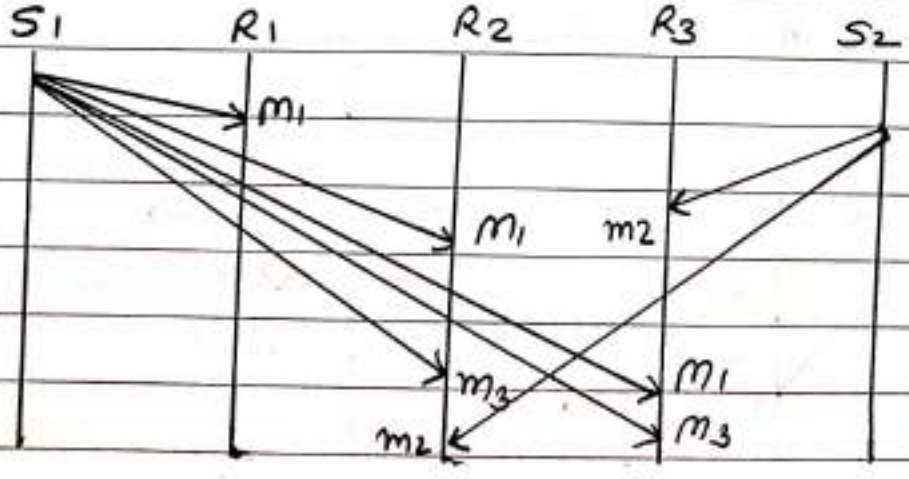
- Global Synchronization is not easy to implement

- All messages are delivered to all receiver process in the same order.
- This order may be different from the order in which message were sent.




- Causal Ordering

- This semantics ensures that if the event of the sending one message is casually related to the event of sending another message the two messages are delivered to all receivers in the correct order.



# Module 3 : Synchronization

## # Synchronization

Logical clock
- Lamport's Logical Clock
- Vector Clock.

physical clock
- Cristian Algorithm
- Berkeley Algorithm
- Network Time Protocol

Election Algorithm
- Bully Algorithm
- Ring Algorithm

Mutual Exclusion

Non-Token based Algorithm
- Centralized algorithm
- Lamport's Algorithm
- Ricart Agrawala Algorithm
- Maekawa's Algorithm

Token based Algorithm
- Suzuki-Kasami Algorithm
- Singhal-Heuristic Algorithm
- Raymond Algorithm

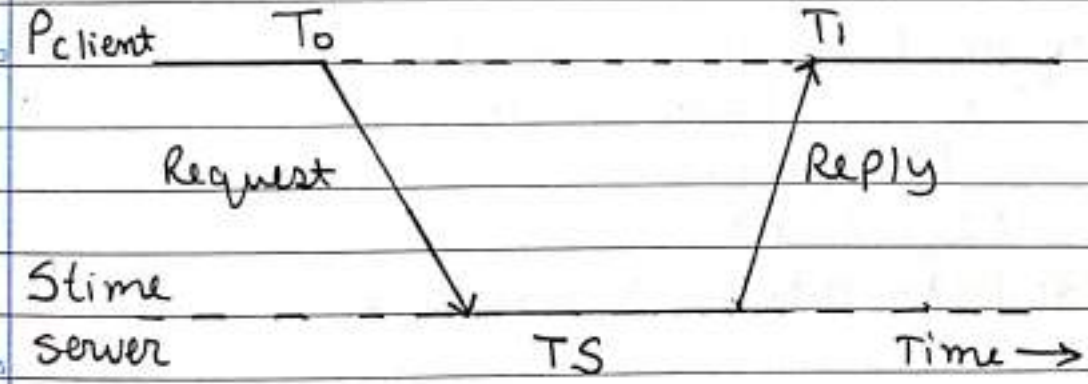
Q.1 Write a short Note on Christian's Algorithm

=> Network maintains a Time Server (TS) which is responsible for synchronising time of all clients.

- Whenever client Sends request, server Sends its own time and client sets the time according to the reply.

Advantage: Easy to implement

Disadvantage:
- Single point of failure
- No Successful clock synchronisation as propogation delay is not considered

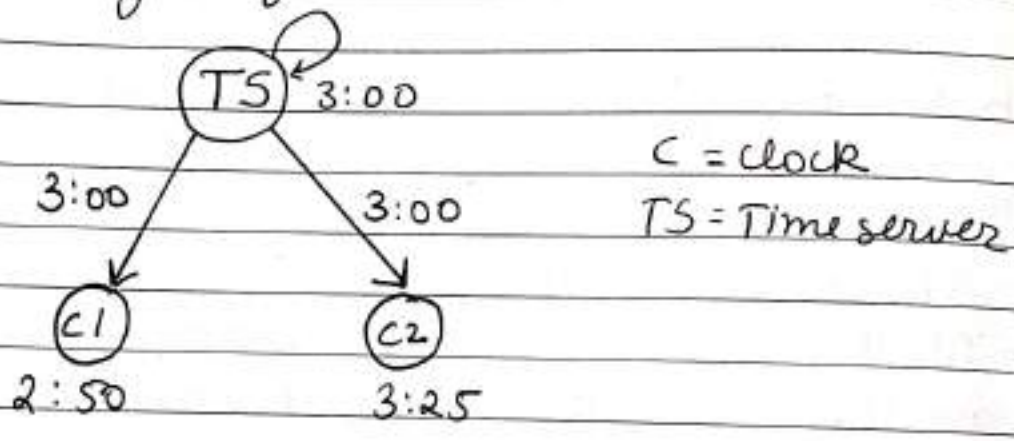



Algorithm:
- Let S be the timeServer and TS be its Time
- Process P request the time from S
- After receiving the request from P, S prepare response and append the time TS from its own clock and then Send it back to P
- $T_{new} = T_{server} + \frac{T_1 - T_0}{2}$

Q2 Write a short Note on Berkeley's Algorithm

=> Berkeley's Algorithm is a method of clock Synchronization in distributed computing which assumes no machine has an accurate time source.

- Time server periodically broadcast its own time to all the machines in the network including itself.




- Every machine now calculates clock value (CV)

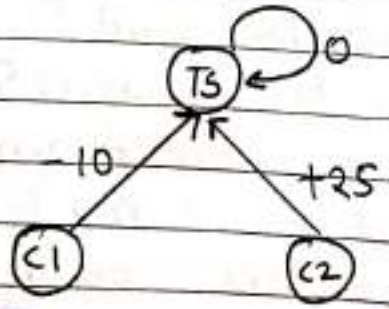
CV = own time - server time

CV(TS) = 0

CV(C1) = -10

CV(C2) = +25

- Every machine will send its calculated CV to time server

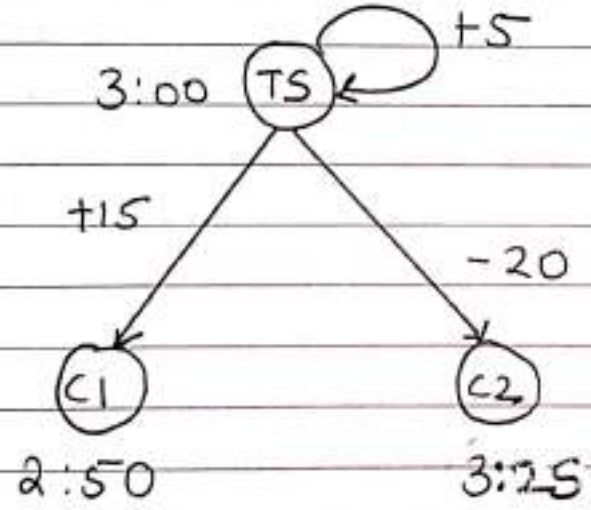


- Time server will now calculate average of all clock value (CV)
  Average (TS) = (0-10+25)/3 = 15/3 = 5

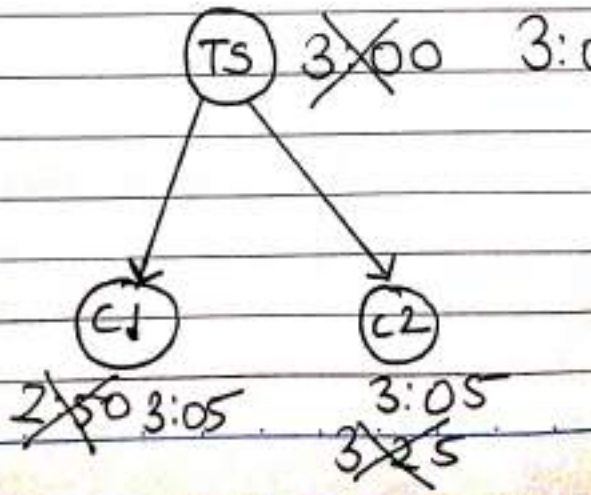
- Time server will now calculate clock adjustment factor (CAV) ie new CV

| process | old CV | New CV |
|---|---|---|
| TS | 0 | 5 |
| C1 | -10 | 15 |
| C2 | 25 | -20 |

- Time server (TS) will send this new CV to all machines




- Every machine will calculate time accordingly to the new CV



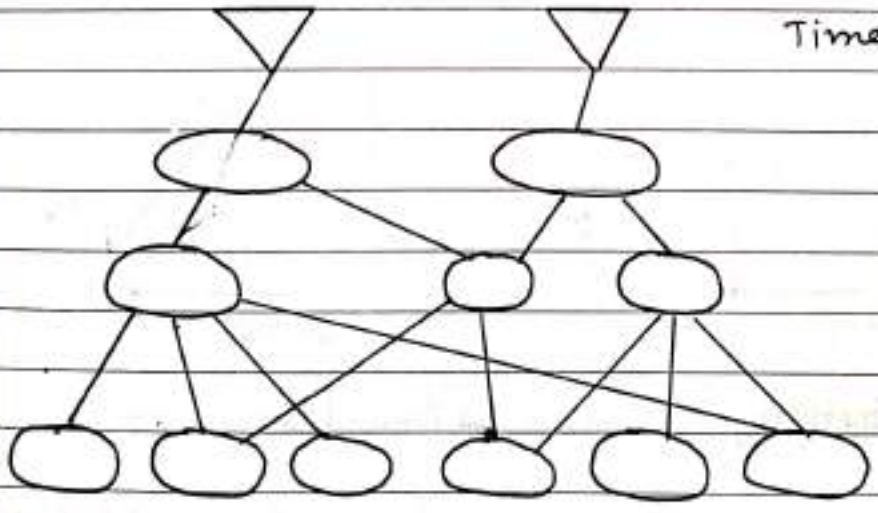
Advantage:
- Successful synchronization due to CV

DisAdvantages:
- Single point of failure
- complex to implement and time consuming to process.

- It works in Master Slave Configuration
Algo
- let Tm be the time estimate of master's clock
- let t[i] contains the time at each Slave at master where i=1 ... N
- If master
  - Send its Tm along with query for t[i]
  - Adjust = Sum(t[i])/N  // (25-10+0)/3=5
  - Send offset[i] = Adjust - t[i] to each Slave;
    // 5-(-10)=+15
    5-25=-20
    5-0=5
- If slave
  - send query response as t[i] = T[i] - Tm
- Time Set to
  - t[i] = t[i] + offset[i]

Q3 Write a Short Note on Network Time protocol

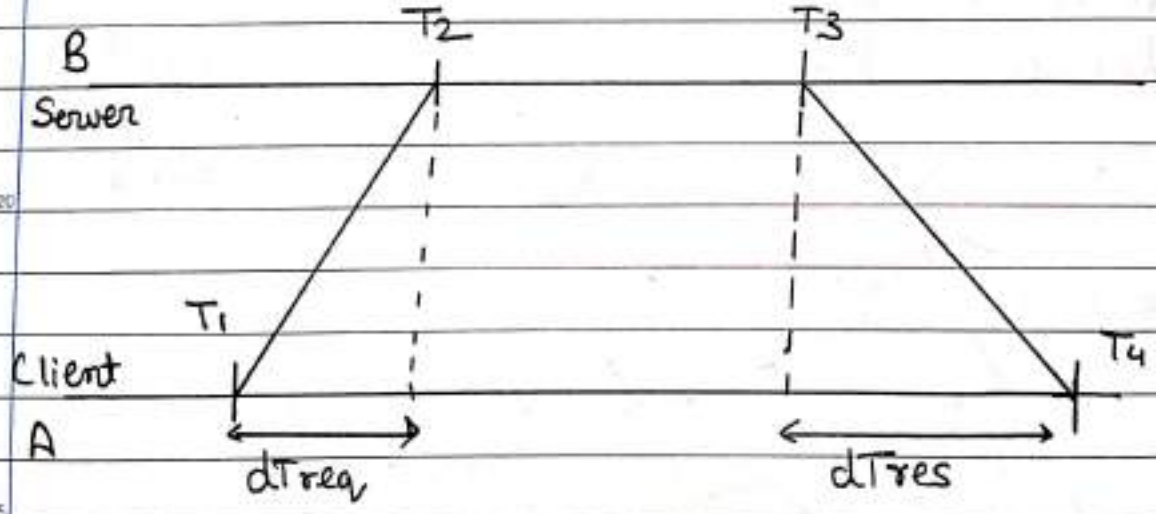
=> NTP is a standard followed to synchronization clocks on the internet.
- NTP synchronizes all participating computers within a few milliseconds of Coordinated Universal time clock.
- There is hierarchical tree structure where each node requires synchronizing with its parent node
- UDP port 123 is reserved for NTP server
- NTP client polls one server to synchronize its clock with a remote server
- It calculates its RTT and Offset O
- Accuracy of Time may depend on buffering delay, TCP time to queue and CPU delay.




1st Stratum: machines connected directly to accurate time source
2nd Stratum: machines synchronized from 1st Stratum machines
...

Network Time Protocol goals

- Enable clients across internet to be accurately synchronized to UTC despite message delays.
- Use statistical techniques to filter data and improve quality of results.
- Provide reliable service
  - Survive lengthy losses of connectivity
    - Redundant paths
    - Redundant Servers
- Enable clients to synchronize frequently
  - Adjustment of clocks by using offset (for symmetric mode)
- Provide protection against interference
  - Authenticate source of data.




Getting current time from Time Server

- $t_4 - t_1$ = the time elapsed at the client side between request sent and response received
- $t_3 - t_2$ = The time server waited before sending the response
- $RTT = (t_4 - t_1) - (t_3 - t_2)$
- $offset = o = ((t_2 - t_1) + (t_3 - t_4))/2$
- Once the RTT and offset is calculated, client clock is adjusted to reduce the offset

# Logical Clocks

- A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system

- ~~So~~ Distributed systems may have no physically synchronous global clock so A logical clock is used.

- It works on the principle of happened before relationship ($\rightarrow$)
  for eg:- $a \rightarrow b$
  This indicates event A happened before event B and hence C(A) < C(B)

- In other words we can say that Logical clock refers to implementing a protocol on all machines within your distributed Systems, So that the machines are able to maintain consistent ordering of events within some virtual timespan

Q4 write a short Note On Lamport's Algorithm or Lamports Logical clock Algorithm in Logical clock

=> In Distributed System clocks are not necessorily synchronized absolutely.

- If two process-es do not interact, It is not necessory that their clocks be synchronized because the lack of Synchronization would not be observable and thus it does not cause problem.
- It is not important that all the process agree on what the exact time is, but that they agree on the order in which events occur.
- Lamports clocks are a very simple and easy technique used for determining the order of events in a distributed system

- It works on the principle of happened before Relationship ($\rightarrow$)

  Defining 'Happens Before': "$\rightarrow$"

  - If a and b are events in the same process and a comes in time before b, then $a \rightarrow b$ and the time measured is $C(a) < C(b)$. always forward, no backward. corrections w.r.t can be made only in positive directions.

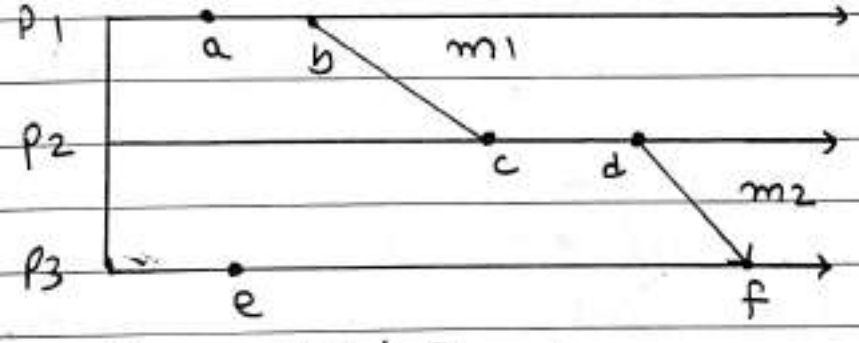
- If a is the sending of a message and b is the reciept of the same message by another process, then $a \rightarrow b$
- If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$ (transitivity)

- If x and y are events happening at different processes which do not exchange any messages then $x \rightarrow y$ is not true nor $y \rightarrow x$

Happened before relation

- $a \rightarrow b$ : Event a occurred before event b.
  Events in the same process $p_1$

- $b \rightarrow c$ : If b is the event of sending a message $m_1$ in a process $p_1$ and c is the event of receipt of the same message $m_1$ by another process $p_2$.
- $a \rightarrow b$, $b \rightarrow c$, then $a \rightarrow c$; "$\rightarrow$" is transitive

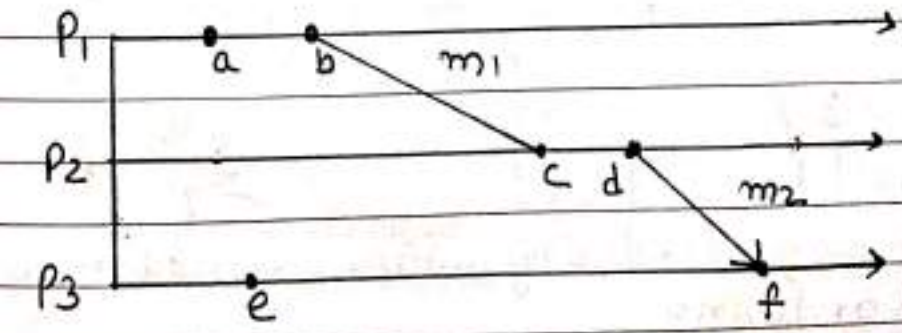



Causally ordered Events

- $a \rightarrow b$: Event a "causally" affects event b

Concurrent Events

$a \| e$: if $a \not\rightarrow e$ and $e \not\rightarrow a$



- A process increments is counter before each event in that process
- When a process sends a message, it includes its counter value with the message
- On receiving a message, one counter of the recipient is updated, if necessary, to the greater of its current counter and the timestamp in the recieved message. The counter is then incremented by 1 before the message is considered received. The algorithm for sending:

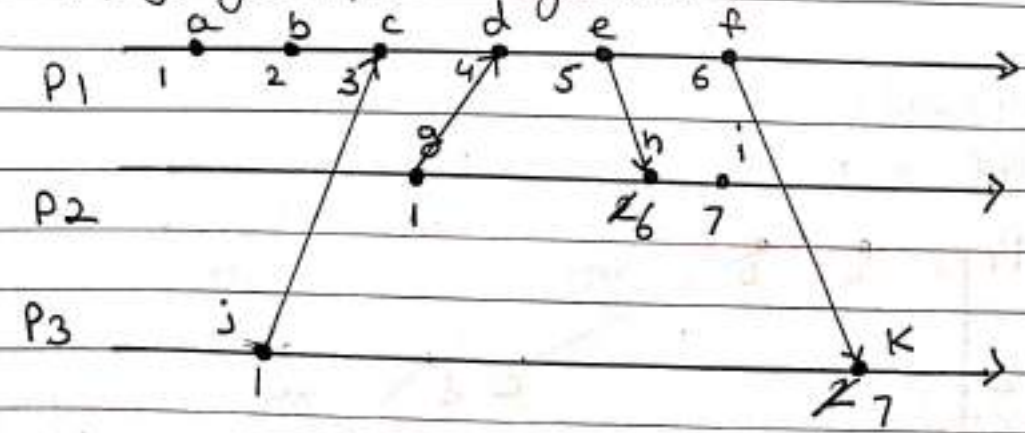
Sending end

```
time = time+1;
time_stamp = time;
Send (message, time_stamp);
```

Receiving end

```
(message, time_stamp) = receive();
time = max(time_stamp, time)+1;
```

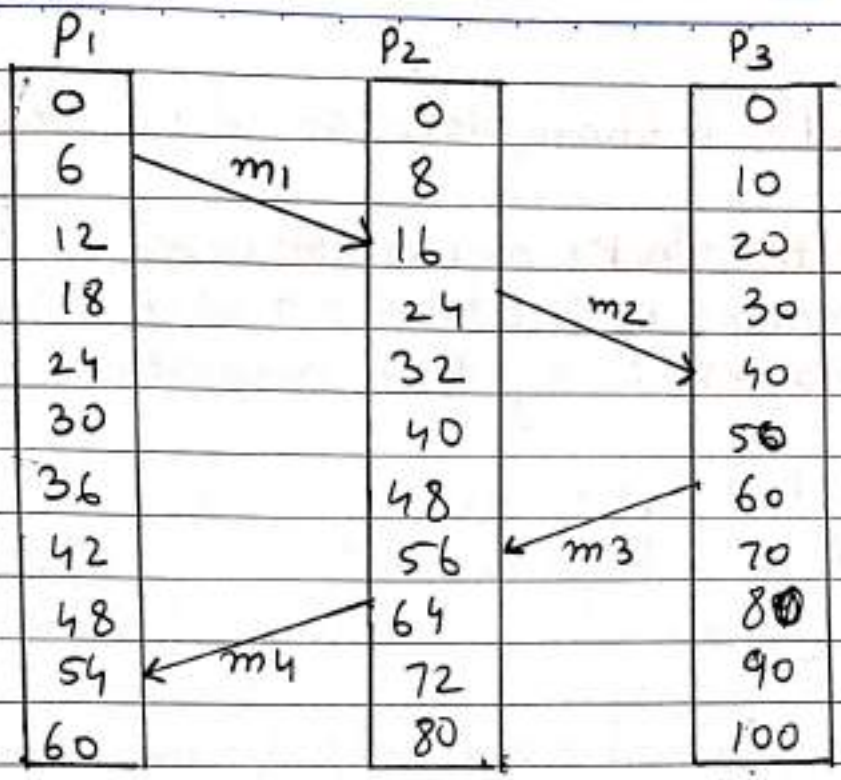
Applying lamport's algorithm




We have good ordering where we used to have bad ordering

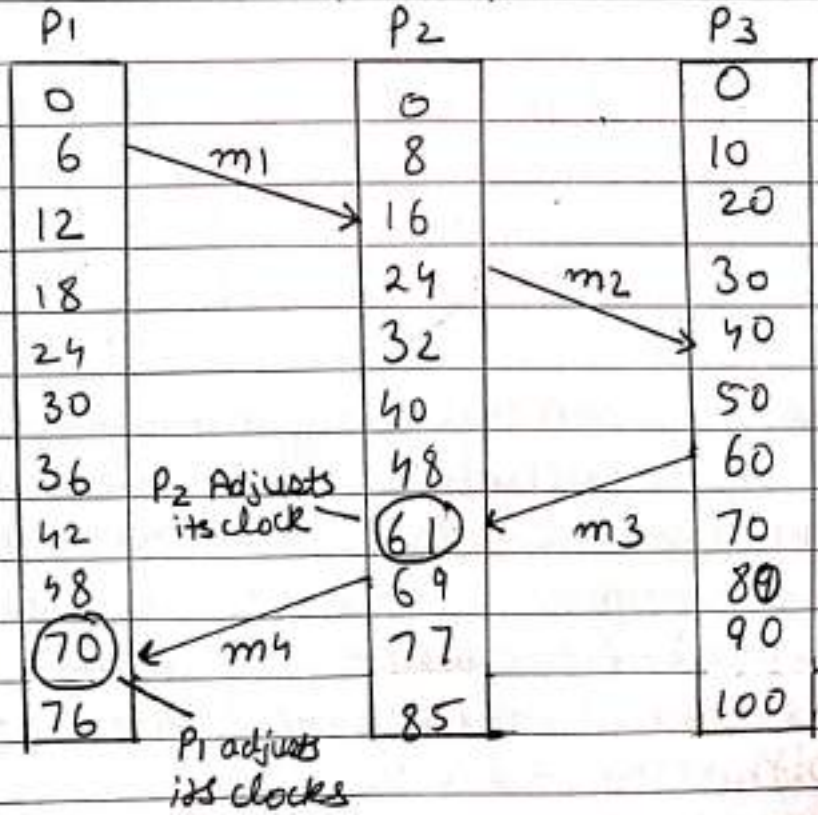
$e \rightarrow h$ and $5 < 6$

$f \rightarrow K$ and $6 < 7$




Here three processes; each with its own clock, The clocks run at different rate.

↓ Lamport's algorithm corrects the clocks



Q5 Write a short Note on vector clock

Vector clocks are constructed by letting each process $P_i$ maintain a vector $VC_i$ with the following two properties:

1. $VC_i[i]$ is the number of events that have occurred so far at $P_i$. In order words, $VC_i[i]$ is the local logical clock at process $P_i$.

2. 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 $P_j$.

Rule 1: Before executing an event (excluding the event of receiving a message) process $P_i$ increments the value $V_i[i]$ within its local vector by 1. This is the element in the vector that refers to processor (i)'s local clock.

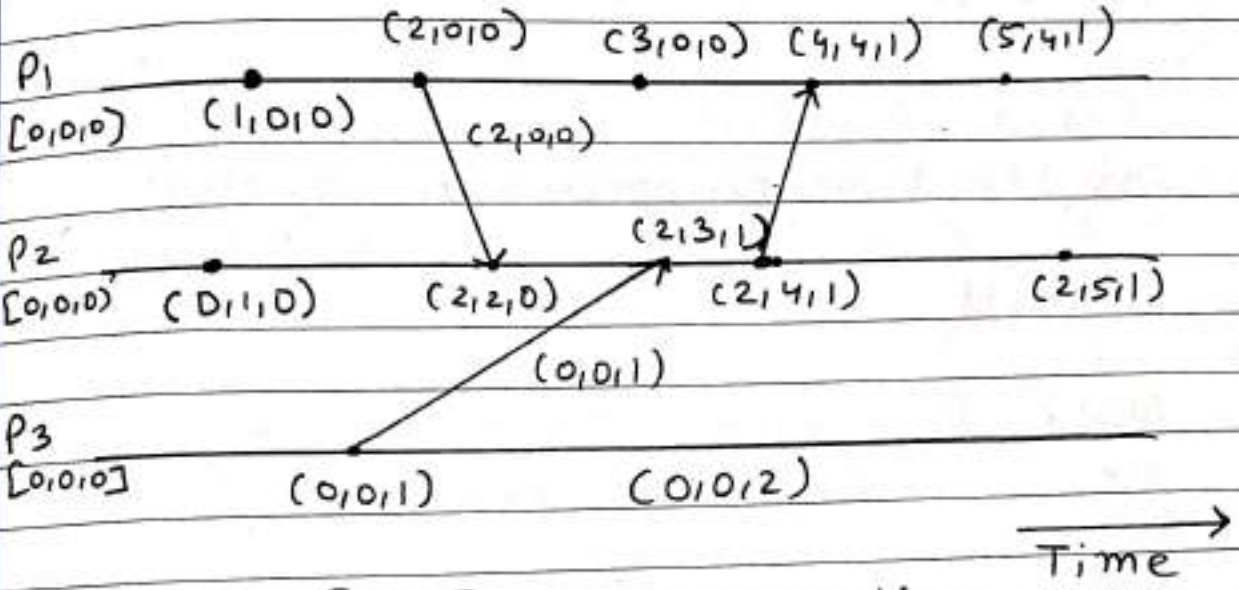
$$V_i[i] = V_i[i] + 1$$

Rule 2: When receiving a message (the message must include the senders vector) loop through each element in the vector sent and compare it to the local vector, updating the local vector to be the maximum of each element. Then increment your local clock within the vector by 1.

for every message at $P_i$ from $P_j$

① $V_i[i] = V_i[i] + 1$

② $V_i[j] = \max(V_i[j], V_j[j])$ for $j != i$




Note: For Better understanding
please refer the vector clock video on my
YouTube Channel Perfect Computer Engineer.

# Election Algorithm

- Many algorithms used in distributed systems require a coordinator
- In general, all processes in the distributed system are equally suitable for the role
- Election algorithms are designed to choose a coordinator.
- Any process can serve as co-ordinator
- Any process can 'call an election' (initiate the algorithm to choose a new co-ordinator).
- There is no harm (other than extra message traffic) in having multiple concurrent elections.
- Elections may be needed when the system is initialized, or if the coordinator crashes or retires.

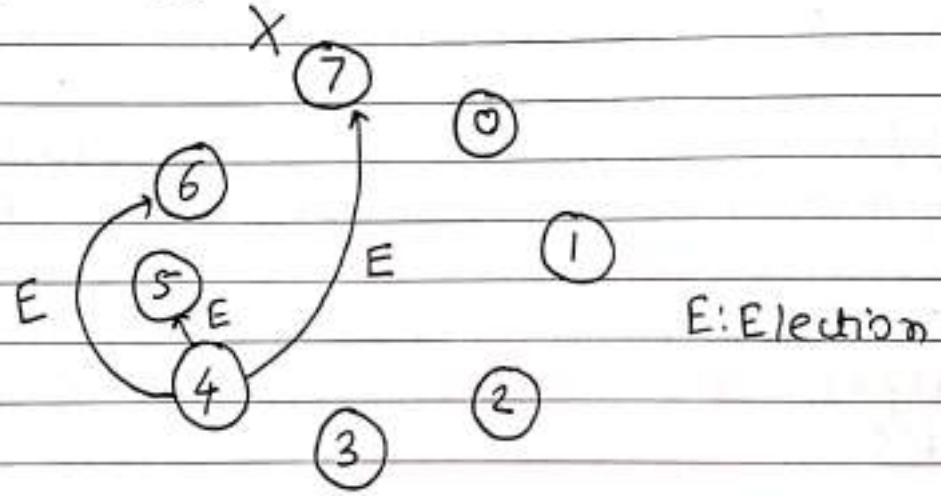
Assumptions:

- Every process/site has a unique ID; eg
  - the network address
  - a process number
- Every process in the system should know the values in the set of ID numbers, although not which processors are up or down.
- The process with highest I.D no. will be new co-ordinator
- Process groups satisfy these requirments.

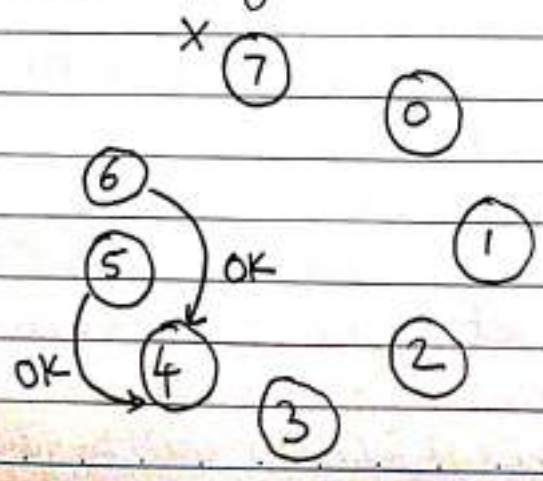
Q 6 Write a short Note on Bully Algorithm?

=> Note: On the beginning of this answer Start with the election Algorithm point discussed in previous page because this is a subset of the election Algorithm topic.

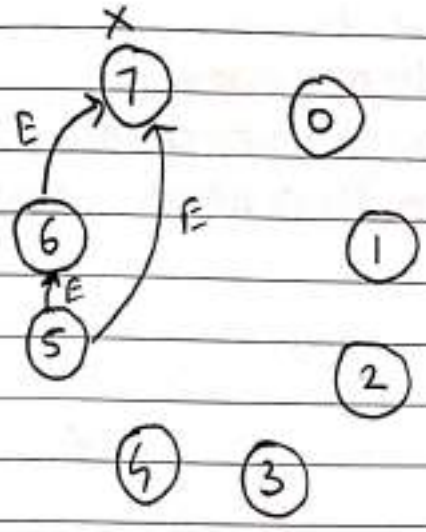
1. Assuming 7 is the co-ordinator and has failed and this is realised by 4 which with the hopes that it can become the co-ordinator Starts the election.




2. 5 and 6 realises that the co-ordinator has failed and process lower than them has started election. Hence 5 and 6 bully 4 by sending OK.

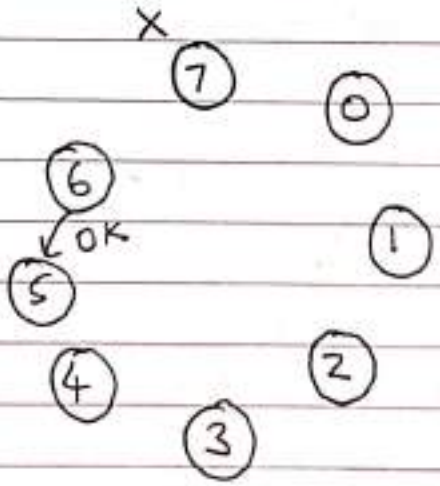


3. 5 and 6 start the election

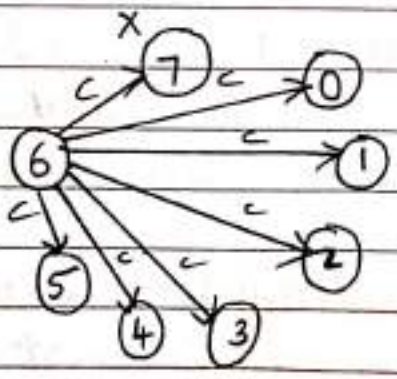



E: Election

4. 6 will bully 5




5. There is no one to bully 6 and hence 6 broadcast everyone to inform that it is the co ordinator




C: Co-ordinator

When the old co-ordinator (7) reboots it will perform election

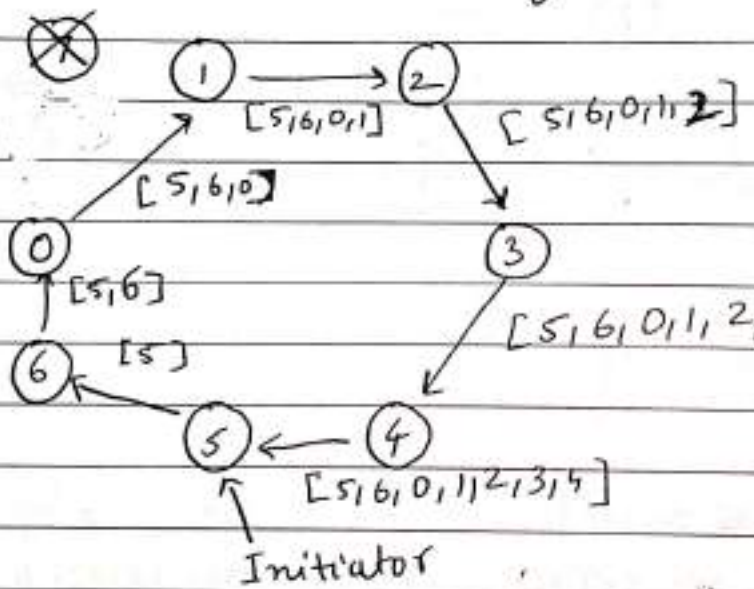
Q7. Write a short Note on Ring Algorithm

=> The ring algorithm assumes that the processes are arranged in a logical ring and each process knows the order of the ring of processes.

- Processes are able to "skip" faulty systems: instead of sending to process i, send to i+1.

- faulty systems are those that don't respond in a fixed amount of time.

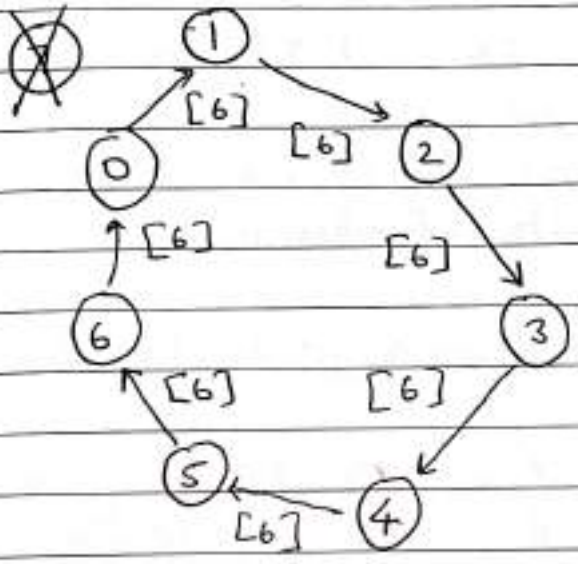
-




1. Process 5 Sense that the co-ordinator is dead
2. Process 5 sends an ELECTION message to its successors (or nodes) (next alive process) with its ID
3. ~~Message~~ At every step nodes keep on adding its own id at the end of the list
4. The process stops when the initiator receives the message it sent.

5. After this the node with highest ID is declared to be a co-ordinator.

6. Initiator Announces the co-ordinator by sending messages to nodes




Advantages:

1. Uses less messages than Bully Algorithm
2. Efficient and takes less space

Disadvantages:

1. Difficult to implement.

# Mutual Exclusion

Critical section: It is a section which is accessible only to one process at a time.

Mutual Exclusion: It makes sure that processes access shared resources or data in serialized way.

Requirments of Mutual Exclusion

Basic requirments for a mutual exclusion mechanism:

- Safety: at most one process may execute in the critical section at a time.

- Liveness: a process requesting entry to the CS is eventually granted it (as long as any process executing the CS eventually leaves it). Liveness implies freedom of deadlock and starvation

- Fairness: Each process should get a fair chance to execute the CR.

Performance Metrices

- Synchronization delay: Time interval between critical region exit and new entry by any process.

- System throughput : Rate at which requests for CR get executed
  - If synchronization delay is Sd, E the average CS execution time:
    System throughput = 1/(Sd+E)

- Message complexity: Number of message that are required per CS execution by a process.

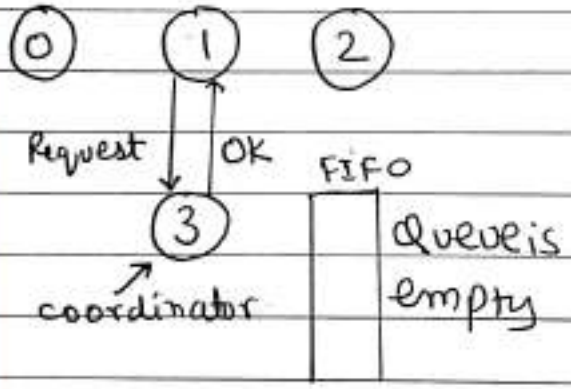
- Responce time : Time interval from a request send to its CR or CS execution completed.

Q8 Write a short Note on Centralized Algorithm

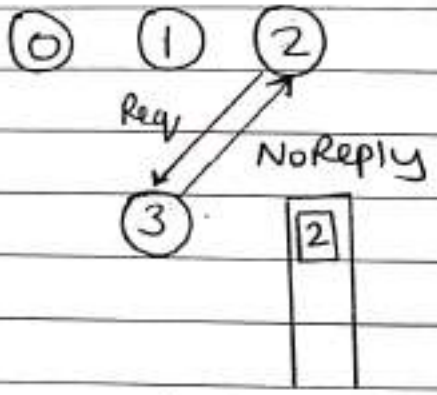
=> Critical section : It is a section which is accesible only to one process at a time.

Mutual Exclusion : It makes sure that process access shared reesources or data in Searalized way

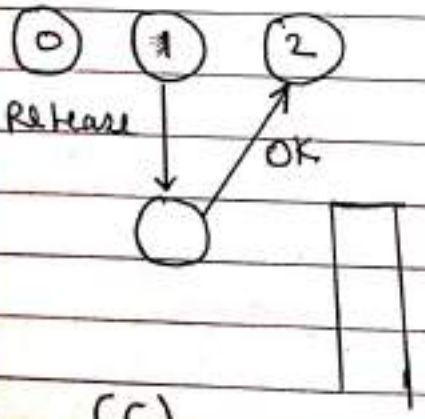
Centralized Algorithm is an Algorithm to implement mutual exclusion in a distributed System.




(a)




(b)




(c)

a) Process1 asks the coordinator for permission to enter a critical section. Permission is granted

b) Process 2 then asks permission to Enter the same critical region. The co-ordinator does not reply.

c) When process1 exits the critical region, it tells the coordinator, when then replies to 2.

Limitations: Coordinator is the single point of failure. Of processes normally block after making a request they cannot distinguish a dead co-ordinator. Single co-ordinator becomes a performance bottleneck in large systems.

Advantages:
1. No Starvation
2. Easy to Implement
3. Requires only 3 messages / use of critical section (request, Reply, release)

Disadvantages:
1. In a large system, a single coordinator can be a performance bottleneck
2. Single point of failure
3. Cannot distinguish between a dead co-ordinator and permission denied.

## Performance Parameters

- The algorithm is simple and fair as it handles the request in the sequential order.
- It guarentees No Starvation
- It uses three messages : Request, Reply, Release
- It has a Single point of failure
- Coordinatorselection could increase synchronization delay especially at time of frequent failure.

Q9 Write a Short Note on Lamport's Algorithm for Mutual Exclusion

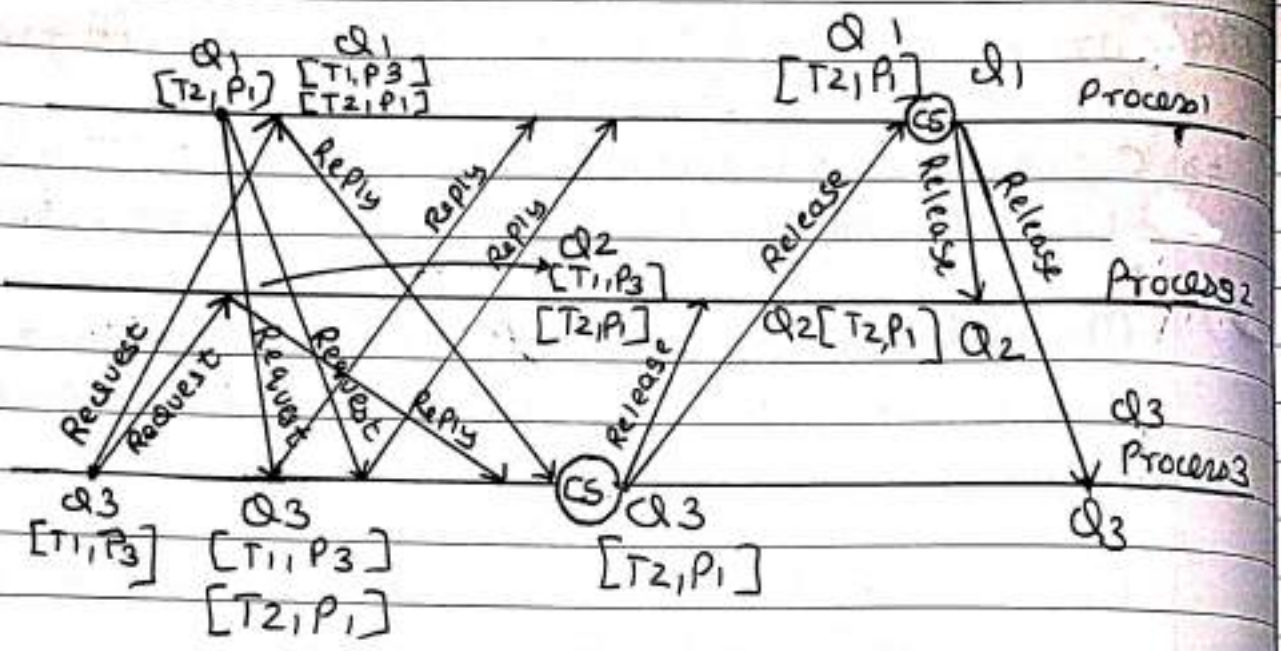
=> Centralized Section :- It is a section which is accessible only to one process at a time.

Mutual Exclusion :- It makes sure that processes access shared resources or data in serialized way.

Lamport's Algorithm is an algorithm to achieve mutual Exclusion.

Algorithm :-

1. Pi sends request to all the Pj's
2. Pi makes entry in its own queue
3. Pj release Request message
4. Pj makes entry in its own queue & sends reply to Pi
5. conditions for critical section
   - Pi has received Reply from all those Pj's to whome he has send request.
   - In queue his request is the topmost.
6. Pi exits the critical section and removes itself from queue
7. Pi broadcasts Release message
8. on receiving message Pj removes Pi's entry.




Note : for a better Explaination please Refer Perfect Computer Engineers Video Playlist on this Subject.

Performance :

- 3(n-1) messages Per Critical Section invocation
- (n-1) Reply
- (n-1) Request
- (n-1) Release

## Advantages :

1. No Singly Point Of failure
2. Mutual exclusion successifully achieved
3. Sophisticated as compared to Richart Agrawala Algorithm

## Disadvantages:

1. N/w bandwidth utilisation very poor due to broadcasting request, n-1 consents and broadcasting release.

2. Nodes getting confused when there is no reply with the fact whatever machine has failed and resource is busy.

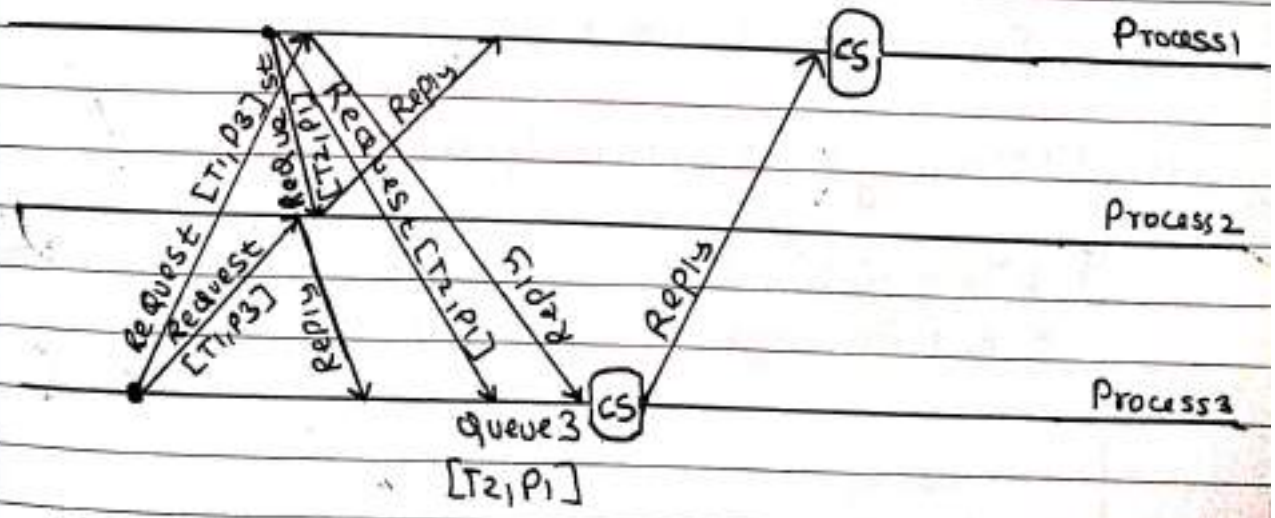
Q10 Write a Short Note on Ricart-Agrawala Algorithm

=> Critical Section: It is a section which is accessible only to one process at a time.

- Mutual Exclusion: It makes sure that processes access shared resources or data in serialized way.

- Ricart Agrawala is a way or an algorithm to achieve Mutual Exclusion.

The Ricart Agrawala Algorithm is an optimization of Lamport's algorithm that dispenses with "Release" Message by cleverly merging them with "Reply" Messages.




Note: for better understanding Refer Perfect Computer Engineer YouTube Channel - - - Video Lectures.

Algorithm:

- Requesting the critical section

1. When a Site Si wants to enter the critical Section, it sends a timestamped REQUEST message to all the Sites in its request set

2. When site Sj receives a REQUEST message from site Si, it sends a REPLY message to Site Si if Site Sj is niether requesting nor executing the critical Section or if site Sj is requesting and Si's requests timestamp is smaller than Sj's own requests timestamp. The Request is Defered otherwise

- Executing the Critical Section

1. Site Si Enters the critical Section after it has received Reply messages from all the Sites in its request set.

- Releasing the Critical section

1. When Site Si exits the critical section, it sends REPLY messages to all the deferred requests.

Performance:

- 2(N-1) messages per CS execution. (N-1) REQUEST + (N-1) REPLY.
- Synchronization delay: T.

Advantage:

1. No single Point of failure
2. Mutual Exclusion succesfully achieved

Disadvantages:

1. Nodes getting confused when there is no reply with the fact wether machine has failed and resources is busy.

Q.11 Write a short note on Maekawa's Algorithm.

=> In Maekawa's algorithm, a process request permission only from a subset of sites instead of all sites.

- To prevent two process to obtain all necessary permissions for CS, these subsets of process must have overlaps

- Each site receiving request message serves as a mediator that ensures only one process under its watch can be given permission for CS at a time.

- The construction of request sets. The request sets for process in Maekawa's algorithm are constructed to satisfy the following conditions:

M1: $(\forall i \forall j : i \neq j, 1 \leq i, j \leq N :: R_i \cap R_j \neq \phi)$

M2: $((\forall i : 1 \leq i \leq N :: P_i \in R_i)$

M3: $(((\forall i : 1 \leq i \leq N :: |R_i| = K)$

M4: Any process $P_j$ is contained in k number of $R_i$s, $1 \leq i, j \leq N$.

Note: you can Replace these 4 conditions with those conditions taught in my video.

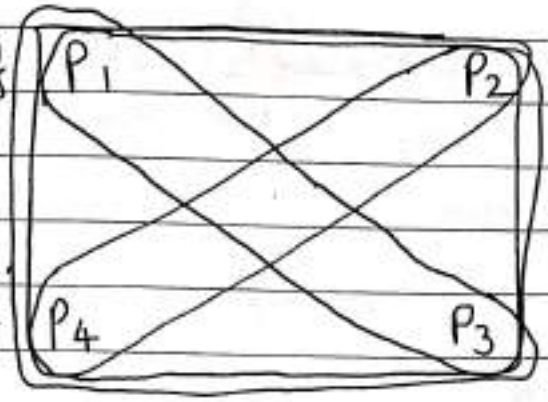
The data structure used by each process $P_i$

1. The request-deferred queue, $RD_i$
2. A variable called 'Voted' = False is set initially

Voted is set to true when a reply is sent indicating that it has already granted permission to a process in its Quorum.

Example:

Step:1




Step:2

$R_1 = \{P_1, P_4, P_2\}$ $(P_1)$ Voted = F

$(P_2)$ Voted = F $R_2 = \{P_2, P_3, P_1\}$

$R_4 = \{P_4, P_3, P_1\}$ $(P_4)$ Voted = F

$(P_3)$ Voted = F $R_3 = \{P_3, P_2, P_4\}$

Step.3: $P_1$ wants to enter the Critical Section

$R_1 = \{P_1, P_4, P_2\}$ $(P_1)$ —Request→ $(P_2)$ $R_2 = \{P_2, P_3, P_1\}$

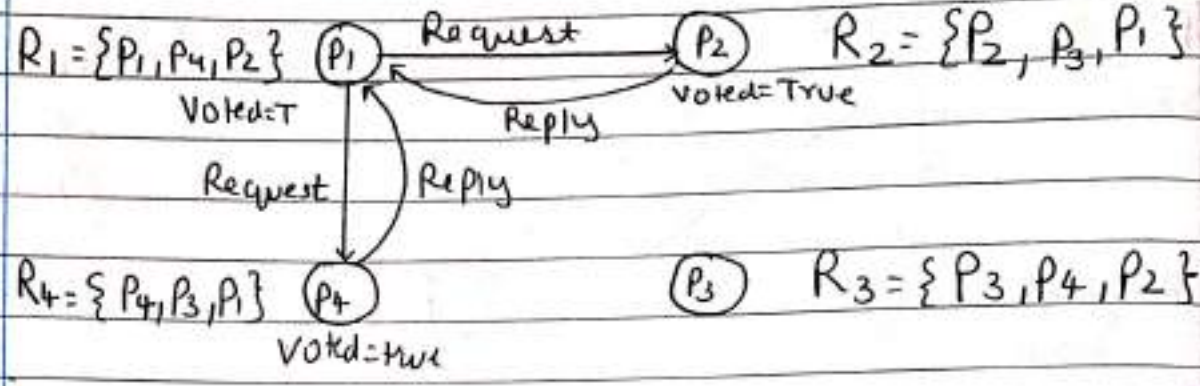
$P_1$ Voted = F; $P_2$ Voted: f

Request

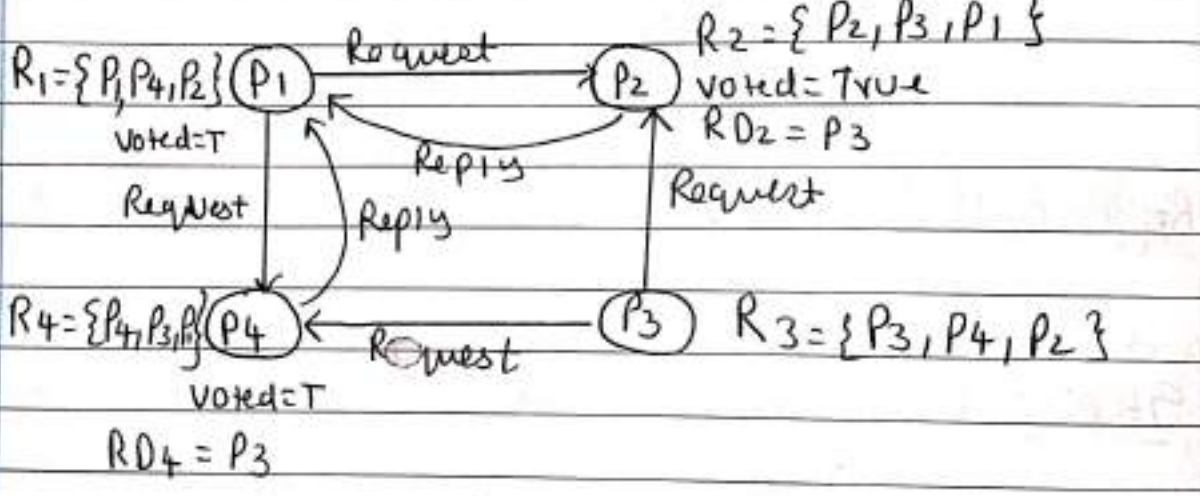
$R_4 = \{P_4, P_3, P_1\}$ $(P_4)$ Voted = F

$(P_3)$ Voted: F $R_3 = \{P_3, P_2, P_4\}$

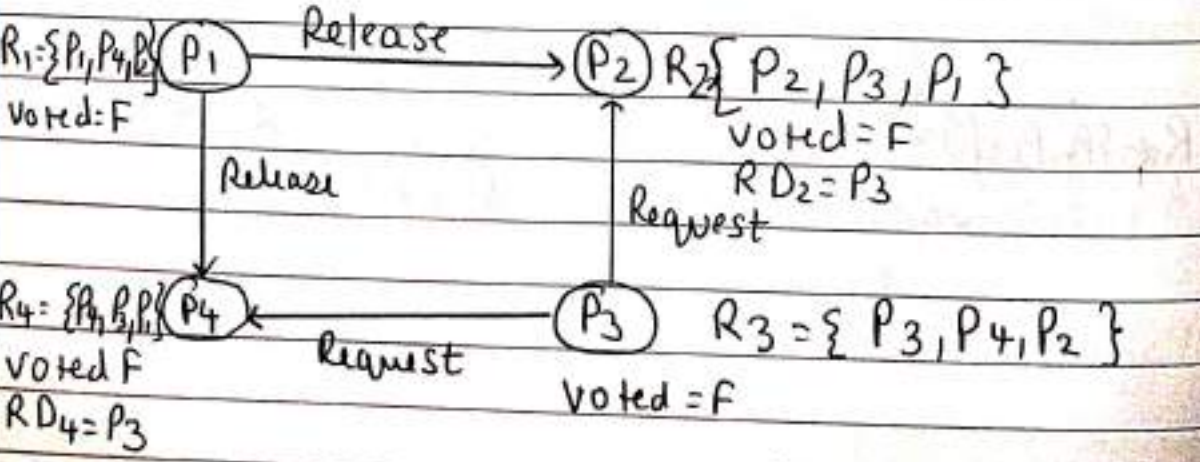
## Step 4:



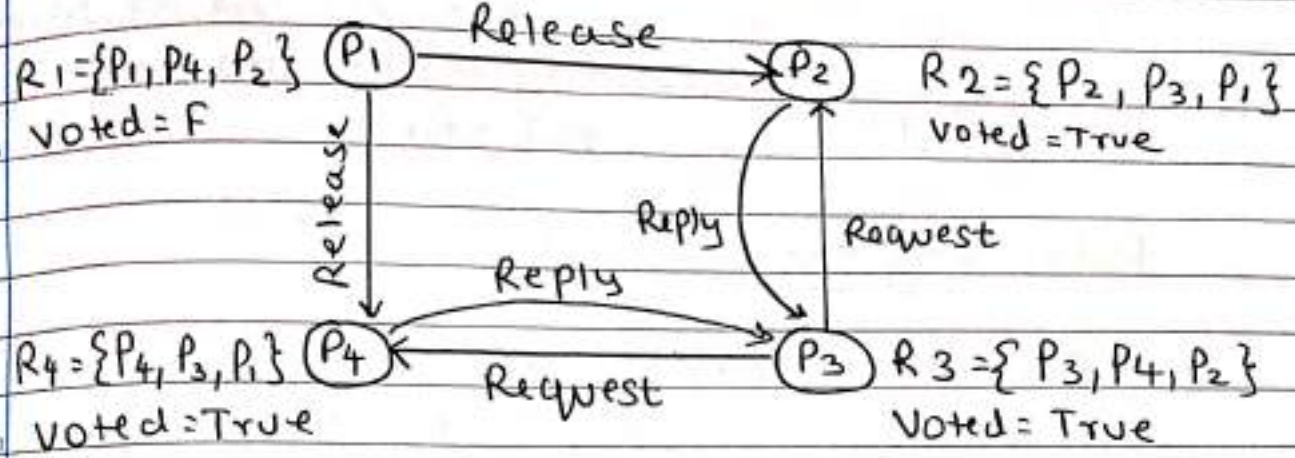

## Step 5: P3 wants to enter into the critical Section.



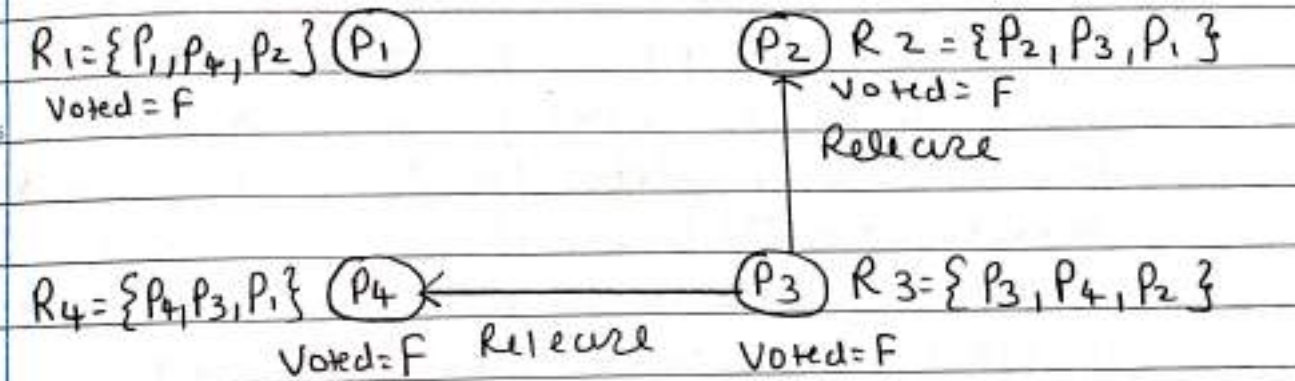

## Step 6: P1 wants to exit critical Section



Step: 7




Step: 8




Algorithm:

Requesting the critical section:

1. A process Pi requests access to the critical Section by sending REQUEST messages to all the process in its request set Ri.
2. When a process Pj receives the Request message, it sends a reply message to Pi, provided it hasent sent a reply message to a process from the time it received the last Release message. Otherwise it queues up the REQUEST for later consideration.

Executing the critical section.

1. Process Pi accesses the Critical Section only after receiving REPLY message from all the process in Ri.

Releasing the critical section:

1. After the execution of the critical section is over, site Pi sends RELEASE(i) message to all the process in Ri

2. When a process Pj receaves a RELEASEi message from process Pi, it sends a Reply message to the next process waiting in the queue and deletes that entry from the queue. If the queue is empty, then process updates its state to reflect that the process has not sent out any Reply message.

Advantages:

1. No Single point of failure
2. Mutual Exclusion Successfully achieved
3. Better bandwidth Utilisation

Disadvantage:

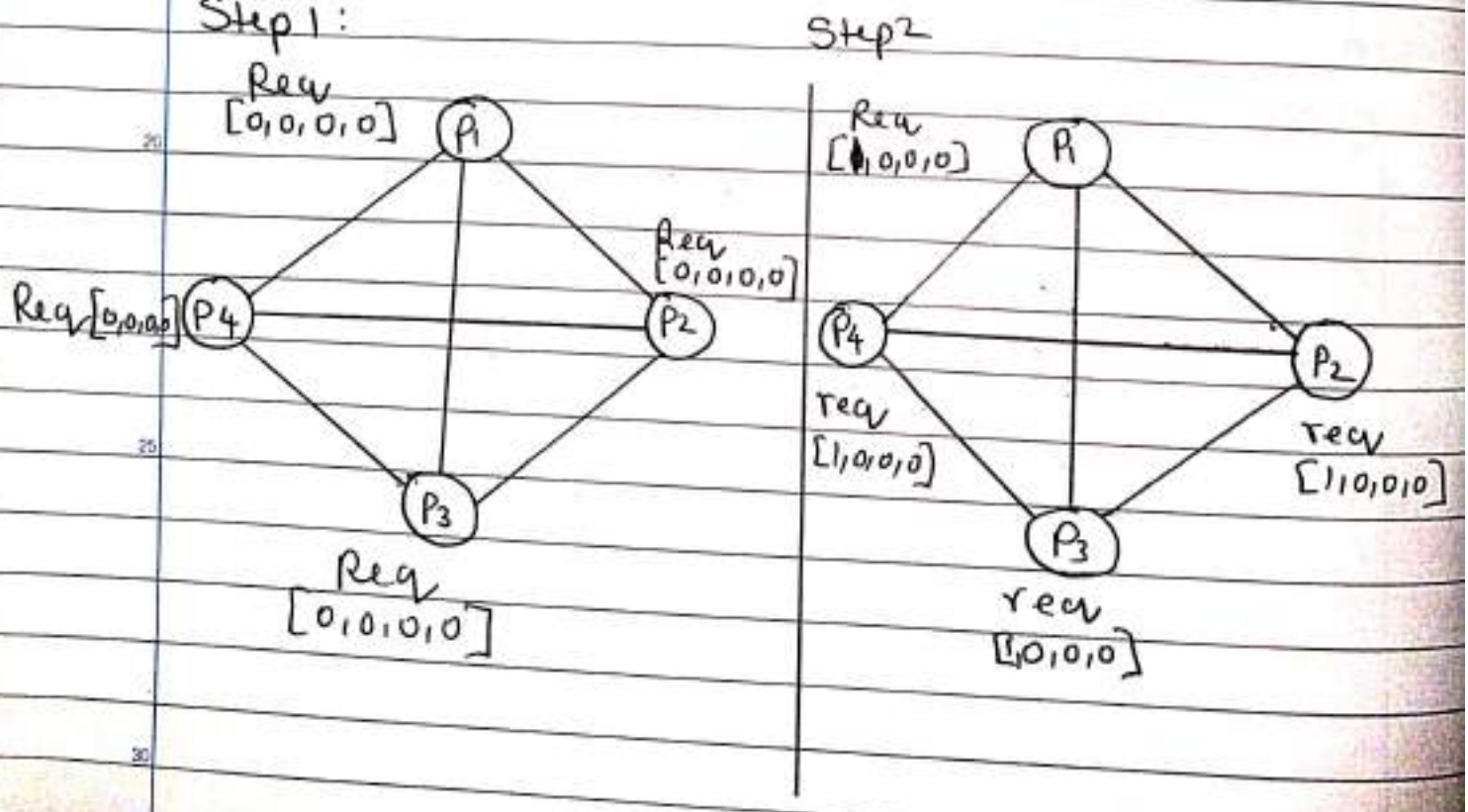
1. It might lead to deadlock
   Solution: Use timestamp.

Performance:
- Synchronization delay: 2T
- Messages: $3\sqrt{n}$ (one each for Request, Reply, Release messages)

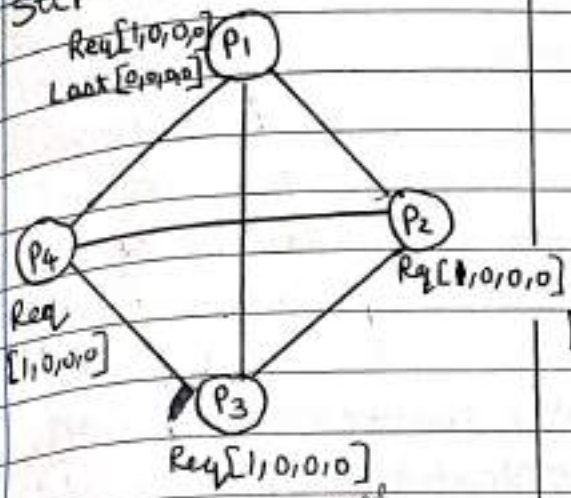
Q12 Write a short Note on Suzuki-Kasami Algorithm.

=> Suzuki Kasami's Algorithm is a token based algorithm which is used to achieve mutual exclusion in distributed systems.

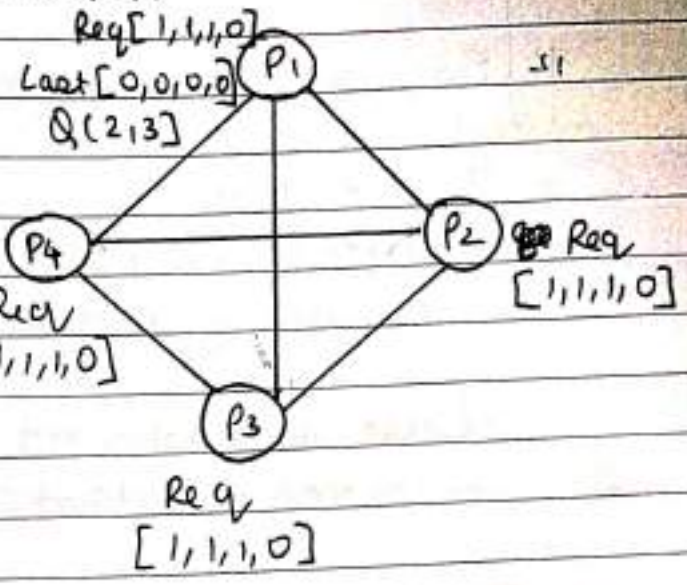
- It is a modified version of Ricart Agrawala algorithm.
- In this algorithm there are two requests used for attaining the Critical Section Request and Reply.
- The proces which holds the token is the only one who can attain the critical section



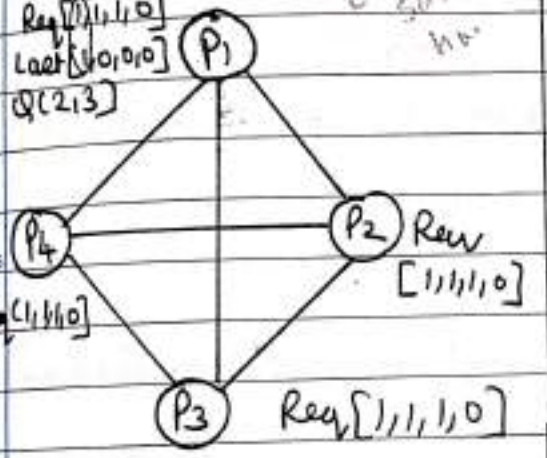
Step3:



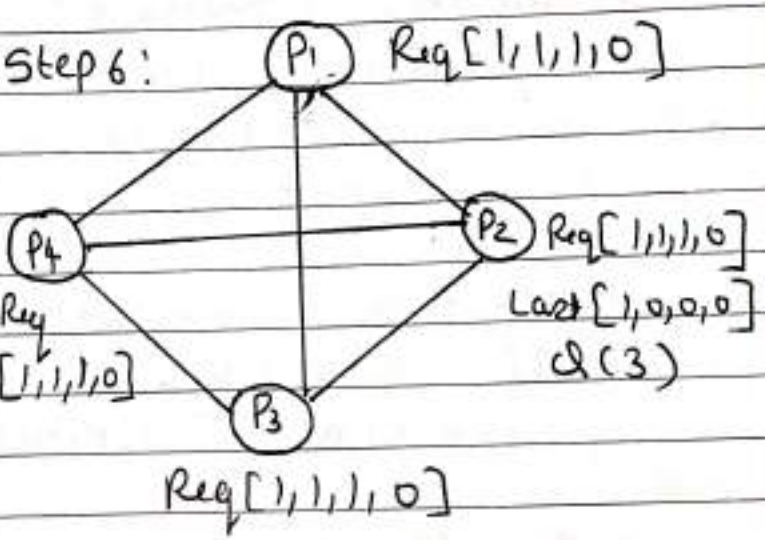

Step4:



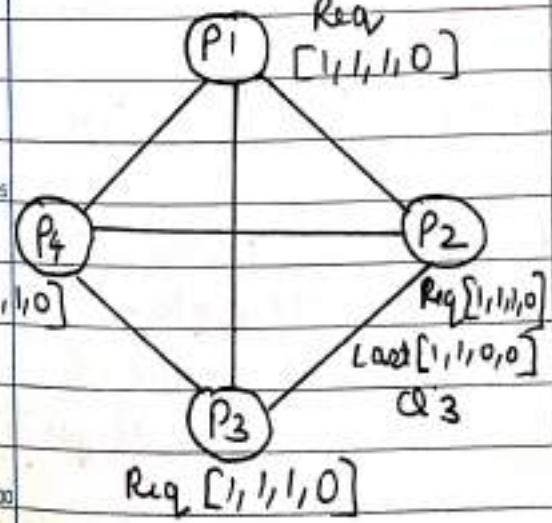

Step5:



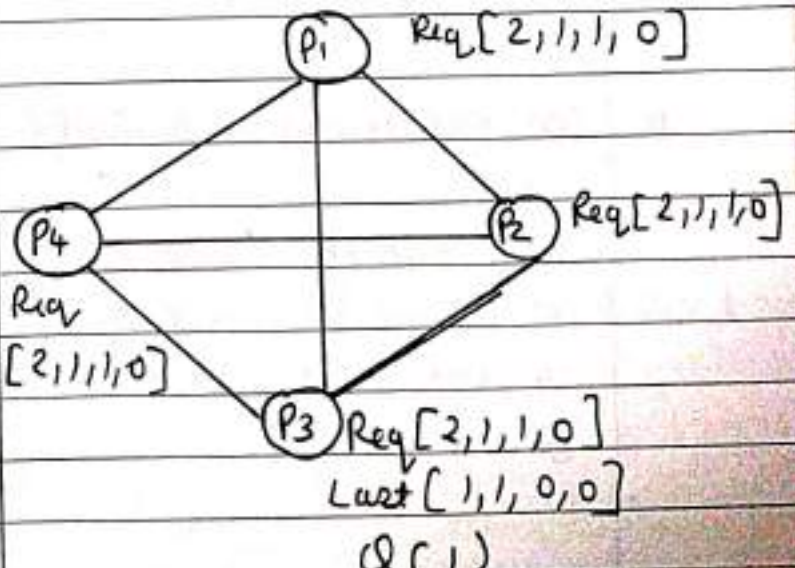

Step6:




Step7:




Step8:



# Suzuki Kasami Algorithm

o The token:
- Queue is a FIFO and it is a queue of the requesting process.

- Last node [1...n]: Sequence number of request that j executed most recently

o The request message:
- REQUEST(i, k): request message from node: i for its kth critical section execution.

o How to enter critical section:
- If i do not have the token, increment RNi[i] and send REQUEST(i, RNi[i]) to all nodes.

- And if i has token already, enter critical section if the token is idle. Else follow rules to release critical section.

o on receiving REQUEST(i, sn) at j:

- if j has the token and the token is idle, then send it to i if RNj[i] = LN[i] + 1. If token is not idle, follow rule to release critical section.

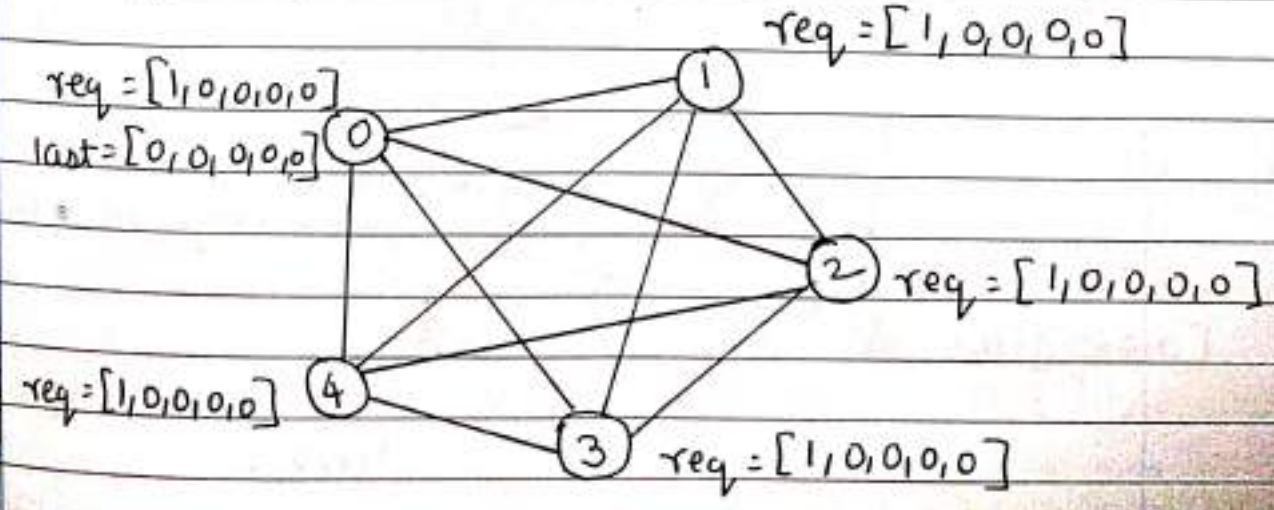
- How to enter critical Section
  - process can enter critical Section if the token is present.

- How to release critical section
  - set LN[i] = RNi[i]
  - if Q is not empty after the above, delete first node from Q and send the token to that node.

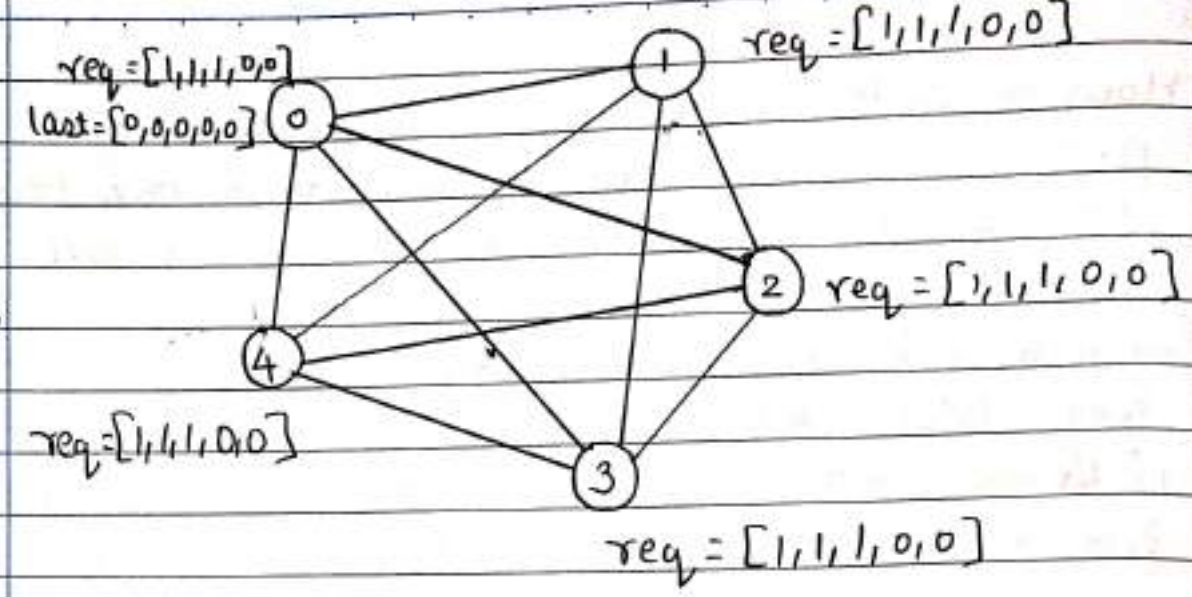
Performance:

- The beauty of the Suzuki Kasami algorithm lies in its simplicity and efficiency
- The algorithm requires 0 or N messages per CS invocation.
- Synchronization delay in this algorithm is 0 or T

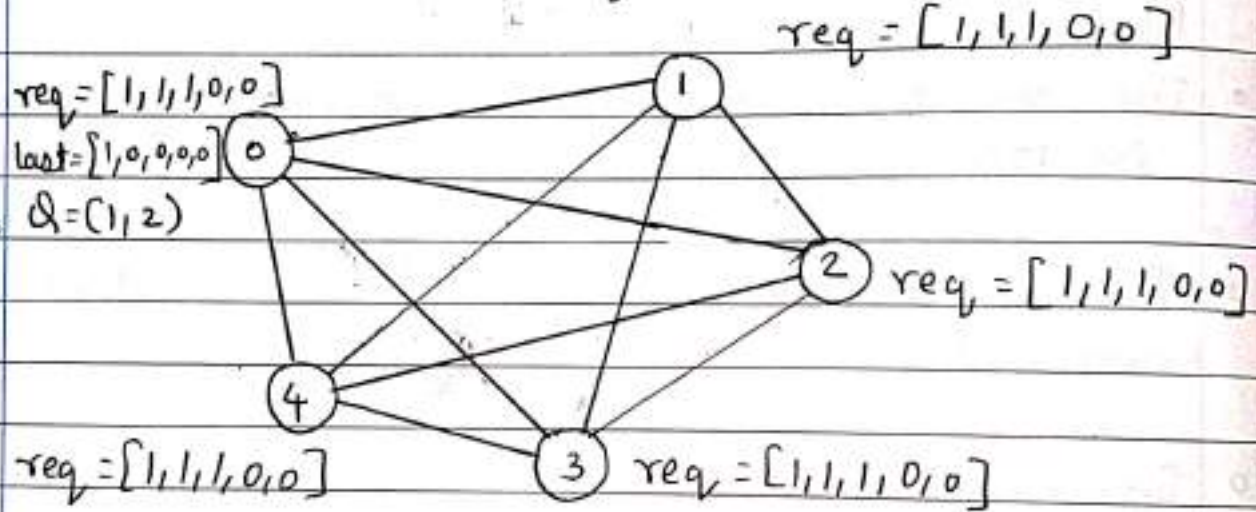
=> Example 2 :



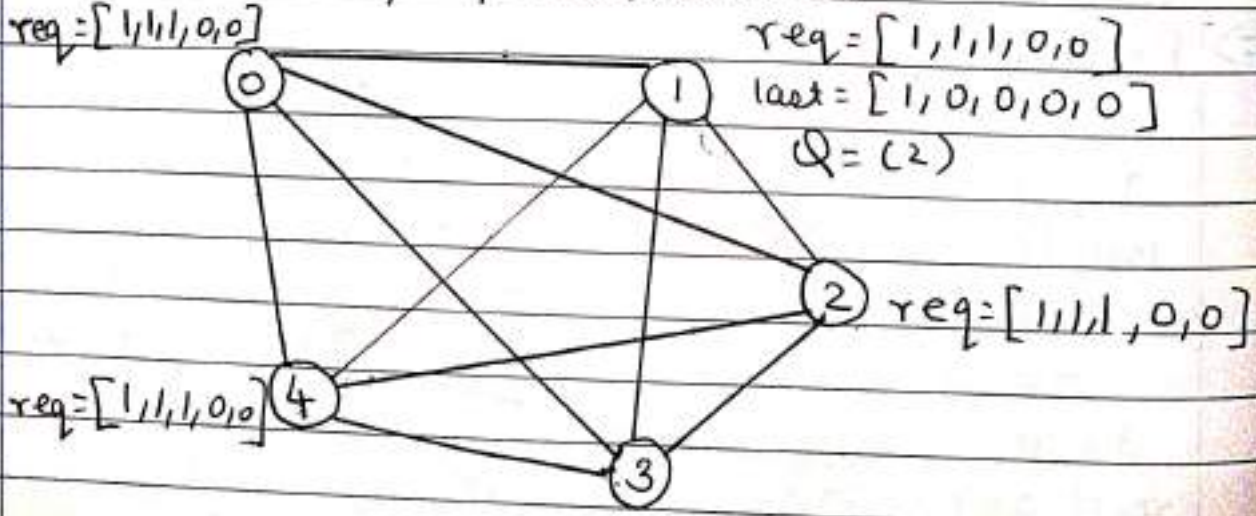

initial State




1 & 2 sends request

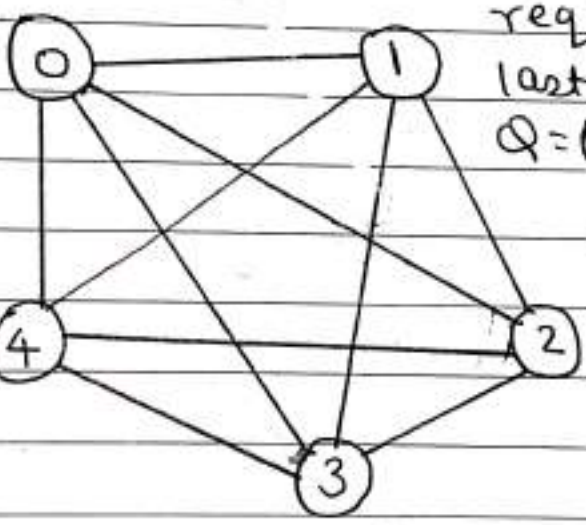



0 prepares to exit CS




0 passes token (Q and last) to 1

req=[2,1,1,1,0]




req=[2,1,1,1,0]
last=[1,0,0,0,0]
Q=[2,0,3)

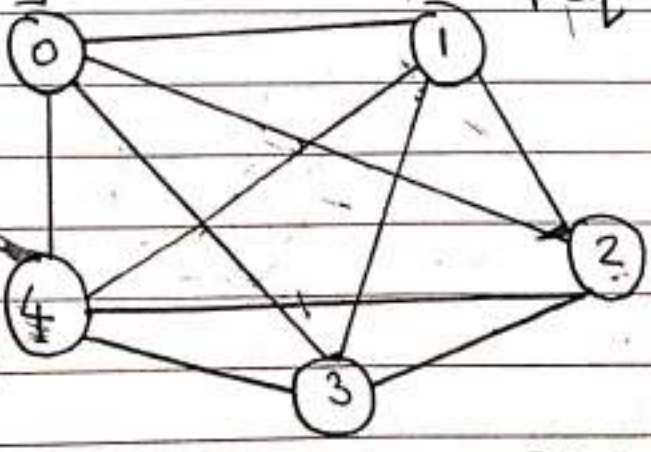
req=[2,1,1,1,0]

req=[2,1,1,1,0]

req=[2,1,1,1,0]

0 and 3 send requests.

req=[2,1,1,1,0]




req=[2,1,1,1,0]

req=[2,1,1,1,0]
last=[1,1,0,0,0]
Q=(0,3)

req=[2,1,1,1,0]

req=[2,1,1,1,0]

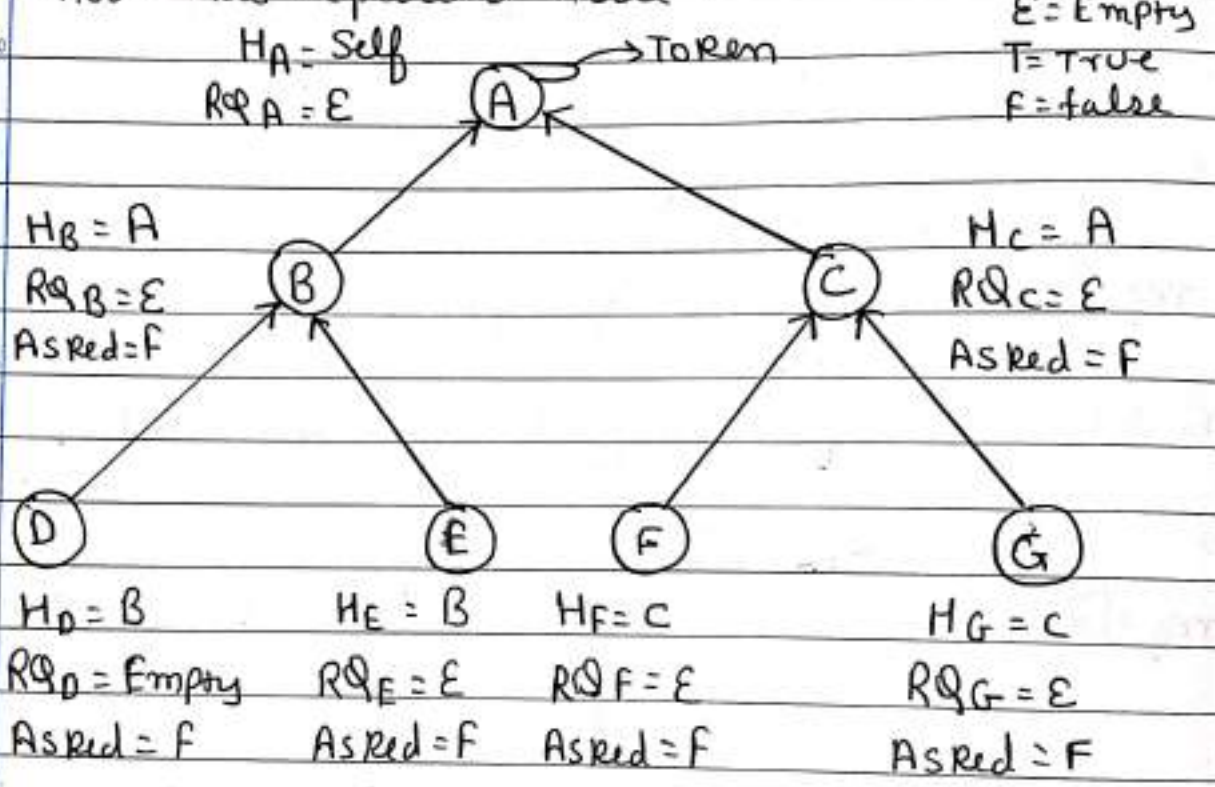
1 sends token to 2

Q=>13 Write a short note on Raymonds Algorithm

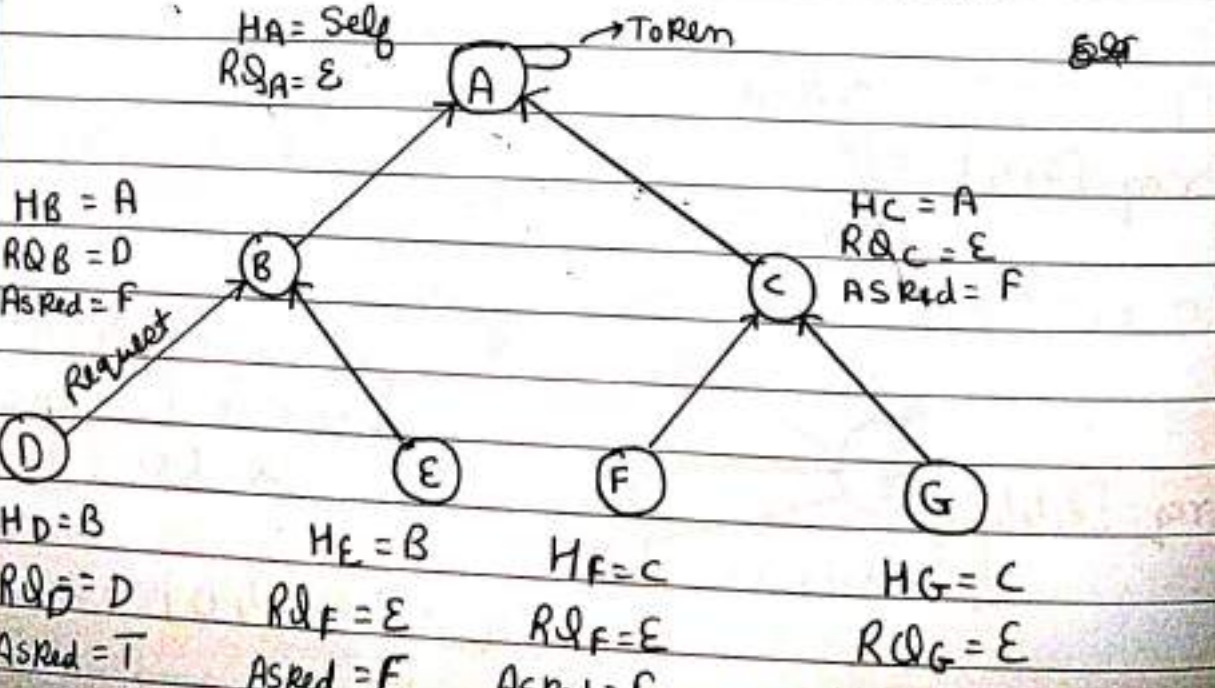
=> Raymond's Tree Based Algorithm is a token based algorithm and a directed tree is formed.

- Here nodes are arranged as a tree and every node has a parent node.

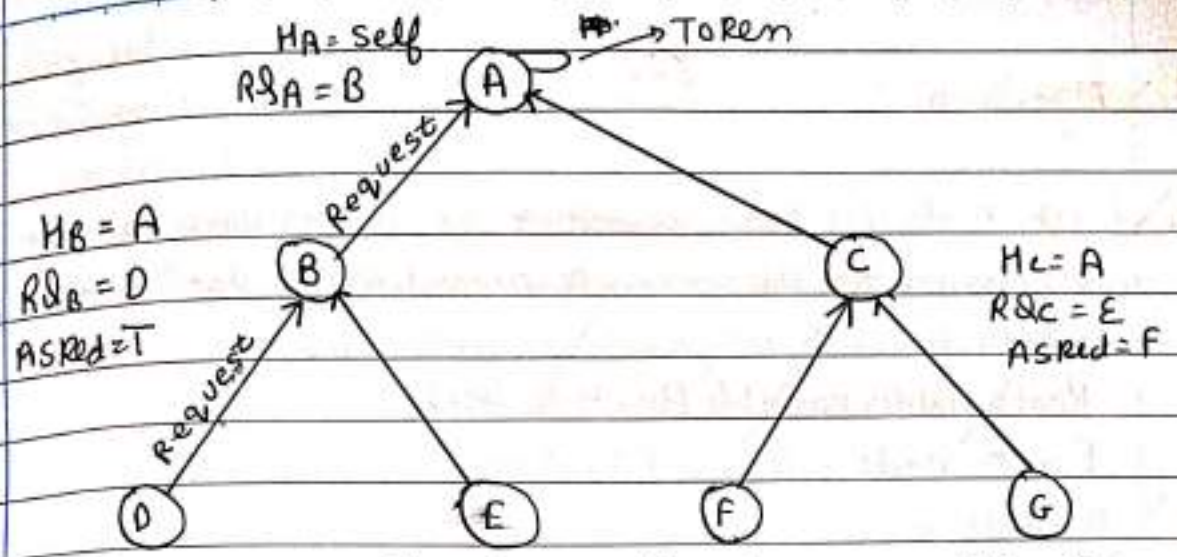
(1)




(2)

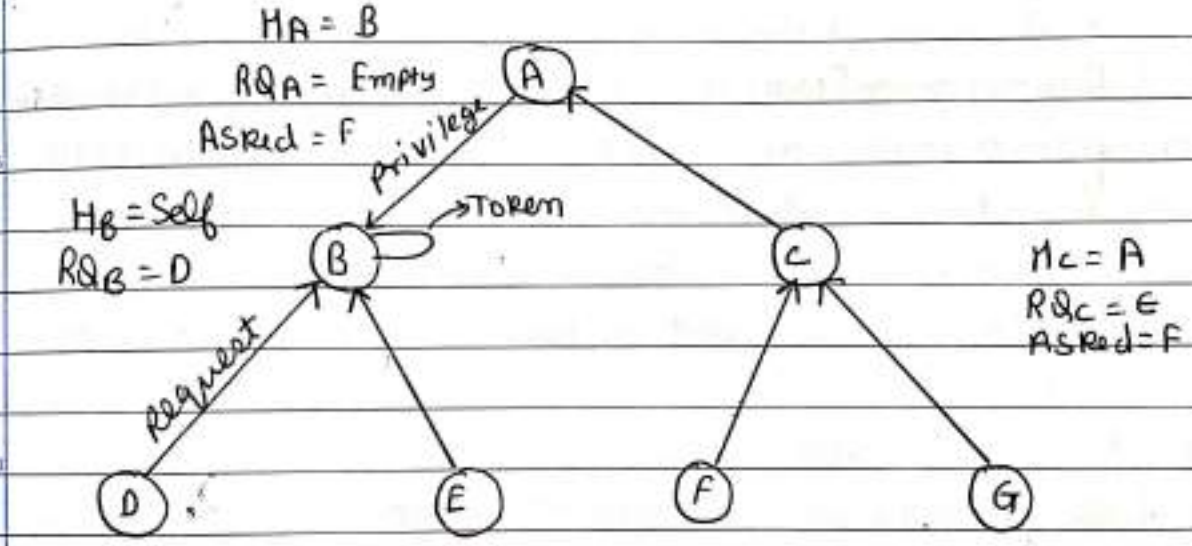


③



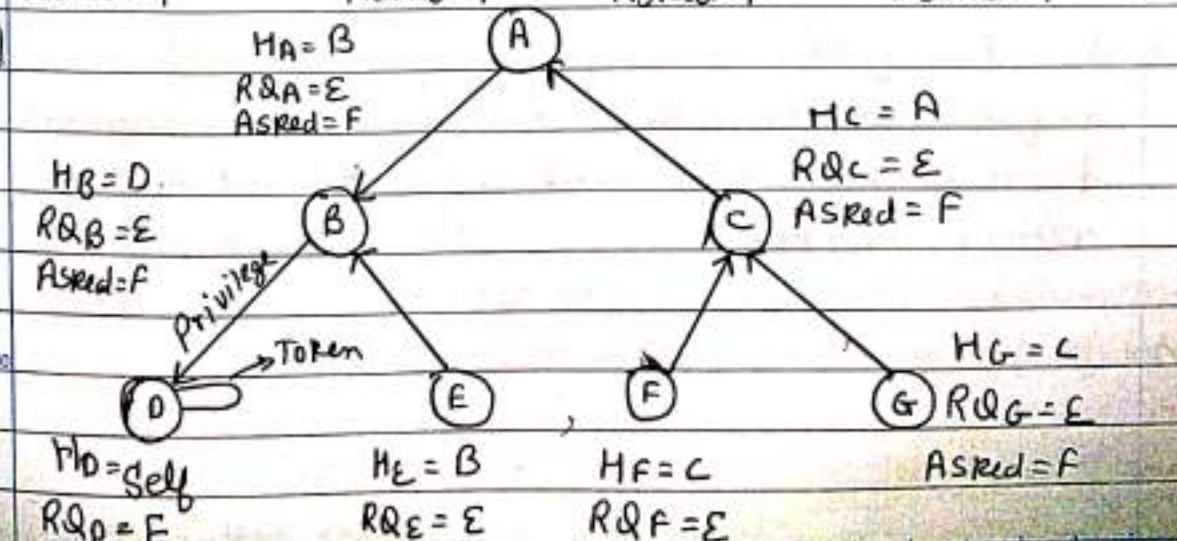

| | | | |
|---|---|---|---|
| $H_D = B$ | $H_E = B$ | $H_F = C$ | $H_G = C$ |
| $RQ_D = D$ | $RQ_E = E$ | $RQ_F = E$ | $RQ_G = E$ |
| ASRed = T | ASRed = F | ASRed = F | ASRed = F |

④




| | | | |
|---|---|---|---|
| $H_D = B$ | $H_E = B$ | $H_F = C$ | $H_G = C$ |
| $RQ_D = D$ | $RQ_E = E$ | $RQ_F = E$ | $RQ_G = E$ |
| ASRed = T | ASRed = F | ASRed = F | ASRed = F |

⑤




| | | |
|---|---|---|
| $H_E = B$ | $H_F = C$ | |
| $RQ_E = E$ | $RQ_F = E$ | |
| ASRed = F | ASRed = F | |

# Algorithm [Note: If you find this Algo long refer my YouTube video]

- processors are arranged as a logical tree
  - Edges are directed towards the processor that holds the token (called the "holder", initially the root of tree)

- Each processor has:
  - a variable holder that points to its neighbor on the directed path towards the holder or the token

- a FIFO queue called request_q that holds its requests for the token, as well as any requests from neighbors that have requested but haven't recieved the token.

e requesting the CS:

- When a process wants to enter the critical section (CS) but it does not have the token, it:
  - adds its request to its request_q
  - if its request_q was empty before the addition, it sends a request message along the directed path toward the holder.

- When a process is the path between the requesting process and the holder recieves the request message

- When the holder receives a request message, it
  - Sends the token towards the requesting process
  - Sets its holder variable to point toward that process

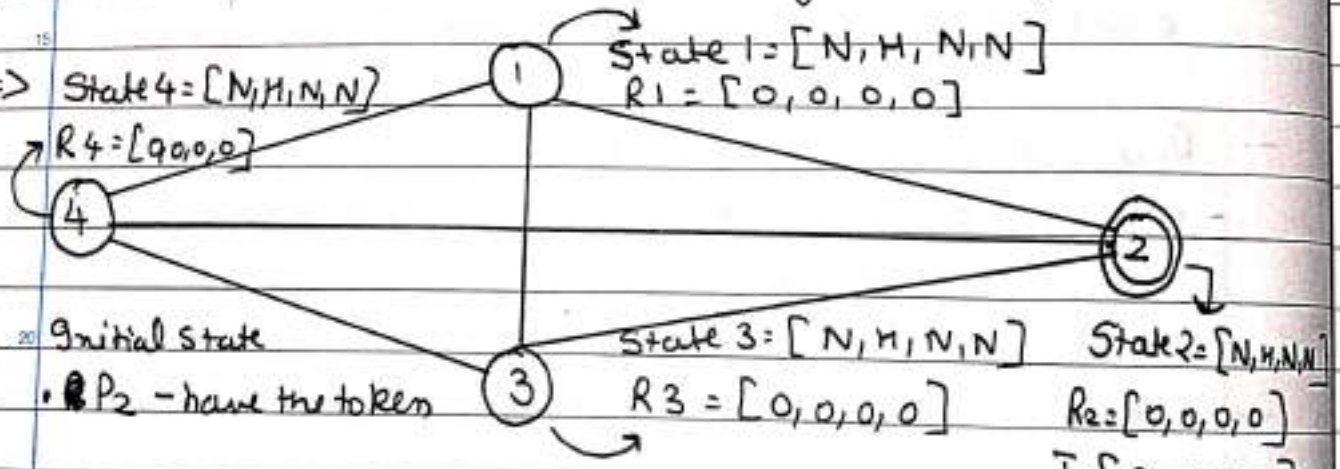
- Requesting the critical section
- When a process in the path between the holder and the requesting process receives the token, it
  - deletes the top entry from its request_q
  - Sends the token towards the process referenced by the deleted entry, and sets its holder variable to point towards that process
  - If its request_q is not empty after this deletion, it sends a request message along the directed path toward the new holder.

- Executing the CS:
- a process can enter the CS when it receives the token and its own entry is at the top of its request_q
  - it deletes the top entry from the request_q and enters the CS

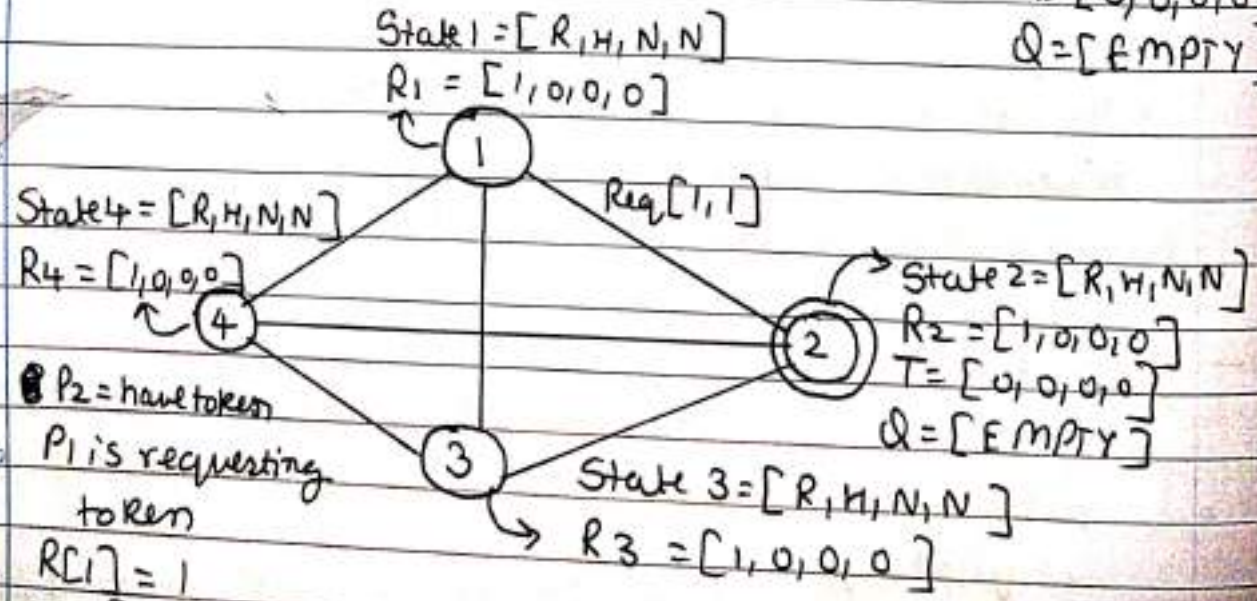
o Releasing the CS:
- When a process leaves the CS
  - if its request q is not empty
    - Deletes the top entry from its request q
    - Sends the token towards the process referenced by the deleted entry, and sets its holder variable to point towards that process.

  - if its request q is not empty after this deletion, it sends a request message along the directed path toward the new holder.

Q14 Write a short note on Singhal's Algorithm.

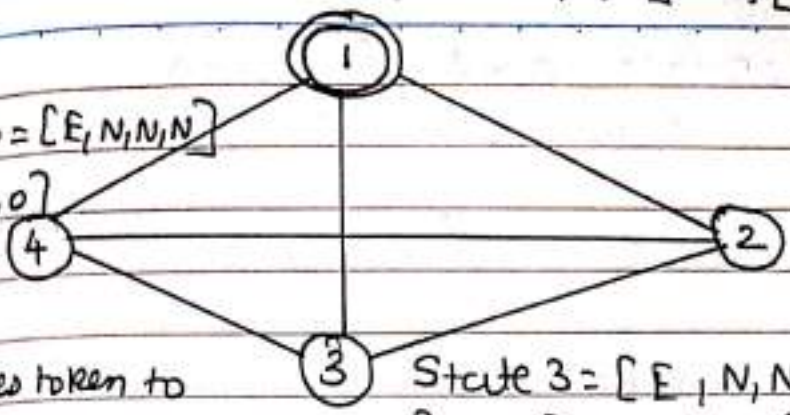



2.



State 1 = [E, N, N, N] T = [0, 0, 0, 0]
R1 = [1, 0, 0, 0] Q = [Empty]

3.




State 4 = [E, N, N, N]
R4 = [1, 0, 0, 0]

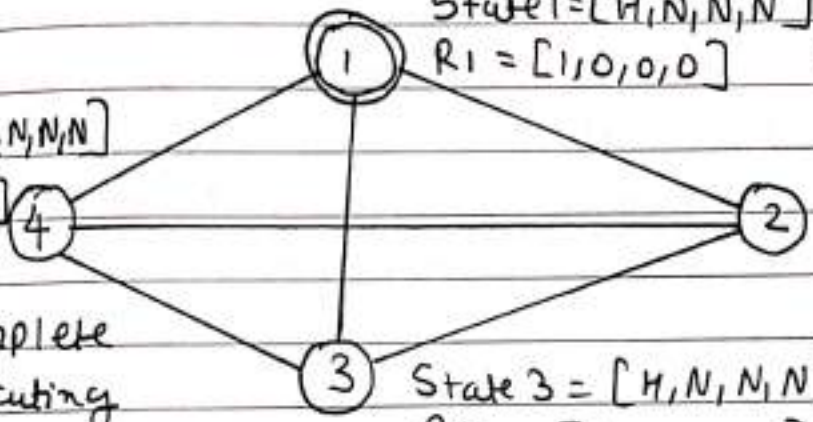
State 2 = [E, N, N, N]
R2 = [1, 0, 0, 0]

State 3 = [E, N, N, N]
R3 = [1, 0, 0, 0]

- P2 gives token to P1
- S[2] = N
- P1 - have the token S[1] = E

State 1 = [H, N, N, N] T = [1, 0, 0, 0]
R1 = [1, 0, 0, 0] Q = [EMPTY]

4.




State 4 = [H, N, N, N]
R4 = [1, 0, 0, 0]

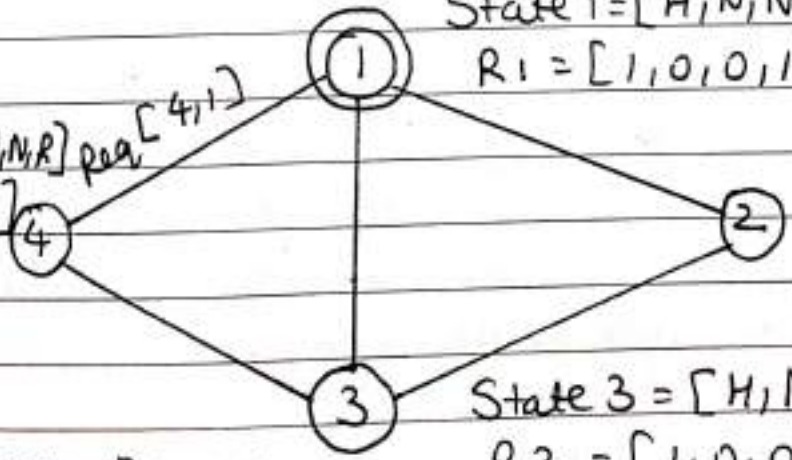
State 2 = [H, N, N, N]
R2 = [1, 0, 0, 0]

State 3 = [H, N, N, N]
R3 = [1, 0, 0, 0]

- P1 = complete executing
- S[1] = H
- T[1] = 1

State 1 = [H, N, N, R] T = [1, 0, 0, 0]
R1 = [1, 0, 0, 1] Q = [EMPTY]

5.



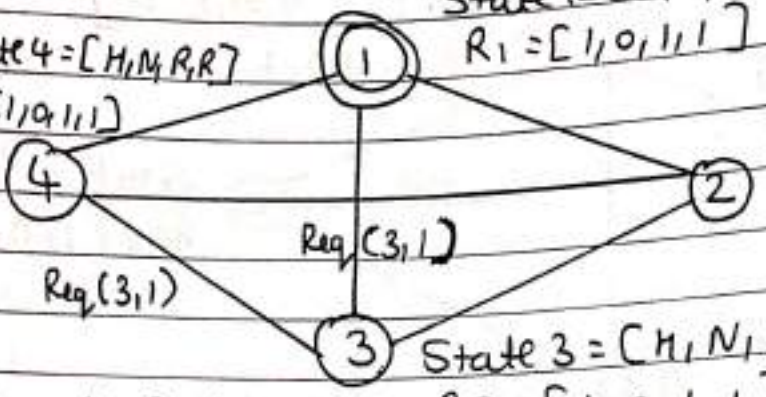

State 4 = [H, N, N, R] req [4, 1]
R4 = [1, 0, 0, 1]

State 2 = [H, N, N, R]
R2 = [1, 0, 0, 1]

State 3 = [H, N, N, R]
R3 = [1, 0, 0, 1]

- P4 requist token from P1
- S[4] = R
- R[4] = 1

6. State 4 = [H, N, R, R]
$R_4 = [1, 0, 1, 1]$
Q = [3]



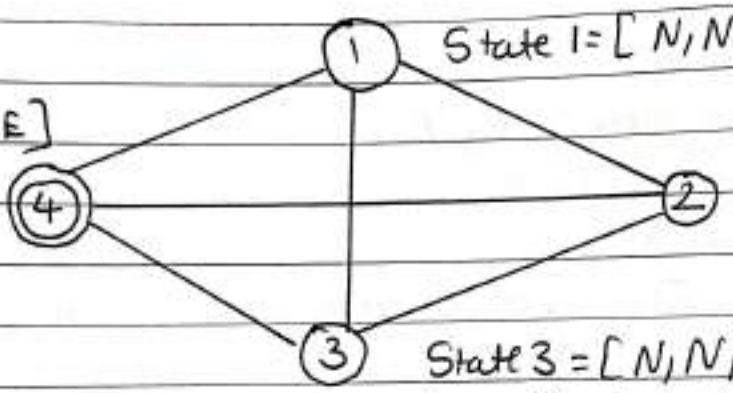

State 1 = [H, N, R, R] T = [1, 0, 0, 0]
$R_1 = [1, 0, 1, 1]$ Q = [3]

State 2 = [H, N, R, R]
$R_2 = [1, 0, 1, 1]$

State 3 = [H, N, R, R]
$R_3 = [1, 0, 1, 1]$

- Meanwhile P3 also request token from P1 & P4
- S[3] = R
- R[3] = 1

7.




State 4 = [N, N, R, E]
$R_4 = [1, 0, 1, 1]$
Q = [3]
T = [1, 0, 0, 0]

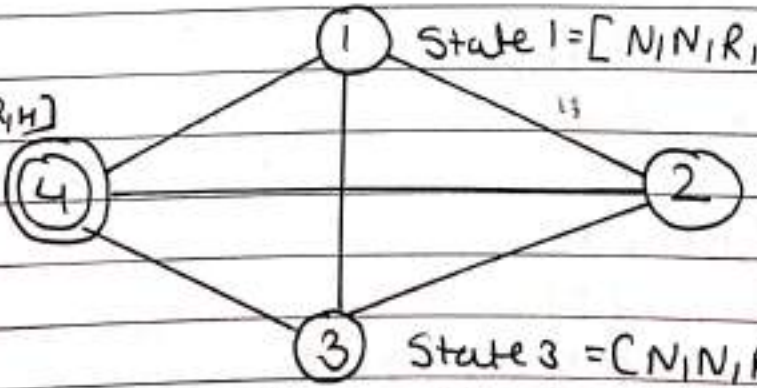
State 1 = [N, N, R, E], $R_1 = [1, 0, 1, 1]$

State 2 = [N, N, R, E]
$R_2 = [1, 0, 1, 1]$

State 3 = [N, N, R, E]
$R_3 = [1, 0, 1, 1]$

- P1 pass token to P4
- S[1] = N
- P4 start executing
- S[4] = E

8.



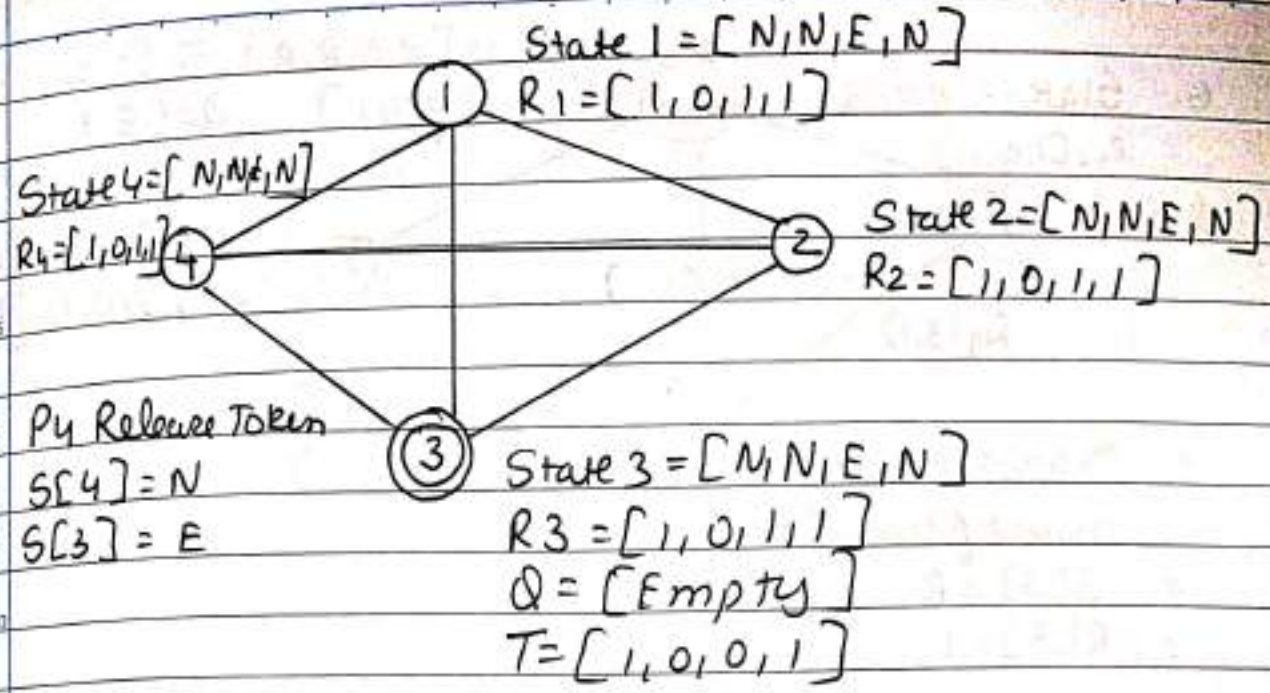

State 4 = [N, N, R, H]
$R_4 = [1, 0, 1, 1]$
Q = [3]
T = [1, 0, 0, 1]

State 1 = [N, N, R, H], $R_1 = [1, 0, 1, 1]$

State 2 = [N, N, R, H]
$R_2 = [1, 0, 1, 1]$

State 3 = [N, N, R, H]
$R_3 = [1, 0, 1, 1]$

- P4 complete execution
- S[4] = H
- T[4] = 1
- P4 check the queue And pass Token to P3




Algorithm:

1] CR request message by Pi
- Set State$_i$[i] = R
- n = R$_i$[i] + 1
- Send Req[i, n) do processes with State = H, E, R

2] Executing CR
- Once token is received set State$_i$[i] = E
- Procss Pi executes CR once it receiveus token

3] Release of CR
- Token[i] = R$_i$[i]
- Set State$_i$[i] = H
- Check queue
- If Non Empty then
  - If State$_i$[j] = R
  - Send token to the process Pj
- Set State$_j$[j] = N

4. $P_j$ on receiving Request

- $P_j$ sets $R_j[i] = \max(R_j[i], n)$
- if state $j[j] = N$ then set state $j[i] = R$
- if state $j[j] = R$ then send Req $(j, R_j[j])$ to $P_i$
- if state $j[j] = E$ then set state $j[i] = R$
- if state $j[j] = H$ then set state $j[i] = R$

if $R_j[i] = \text{TOKEN}[i] + 1$

then send TOKEN to $P_i$

change state $j[j] = N$

# Resource and Process Management

Q.1 What is Resource Management

=> A resource can be a logical such as a shared file, or physical, such as CPU (a node of the distributed system)

One of the functions of a distributed operating system is to assign processes to the nodes (resources) of the distributed system such that the resource usage, response time, network congestion, and scheduling overhead are optimized.

There are three techniques for scheduling processes of a distributed system:

1. Task Assignment Approach: In this, each process submitted by a user for processing is viewed as a collection of related tasks and these tasks are scheduled to suitable nodes so as to improve performance.

2. Load Balancing approach, in which all the processes submitted by the users are distributed among the nodes of the system so as to equalize the workload among the nodes.

Page : 12
Date :

3. Load-Sharing approach, which simply attempts to conserve the ability of the system to perform work by assuring that no node is idle while processes wait for being processed.

- The task assignment approach has limited applicability to practical situations because it works on the assumption that the characteristics (eg. Execution time, IPC costs etc.) of all the processes to be scheduled are known in advance.

Q2. Desirable features of Global Scheduling Algorithm.

=> Global Scheduling Algorithm is used for assigning the tasks and balancing the nodes.

1. No priori Knowledge about the process : Many of times the prior knowledge of any process is not available.
A good scheduling algorithm should be capable of working without any knowledge of the characteristics of process.

2. Dynamic in Nature : In distributed System the load assigned to any of the node may change dynamically. Thus the scheduling algorithm must be dynamic in nature.

3. Balanced system performance and scheduling overhead : Many of the global Scheduling algorithms use the global state information to make process assignment decisions.
The scheduling algorithm should provide an optimal or near optimal system performance with a minimum of global state information gathering overhead.

4. Quick decision making capability :
In distributed system, the situation always keeps on changing, so the scheduling algorithm should be capable enough for making quick decisions about the allocation by analyzing the situation.

5. Stability : Unstable process migration might lead to process thrashing so the scheduling algorithm should have stable process migration.

6. Scalability : The scheduling algorithm should always be capable enough of handling a network with any size. So it should be scalable to adopt any change.

7. Fault Tolerance : Algorithm should continue its working although one or more nodes in system crashes. It should consider available nodes in decision making

Page: 112
Date:

Q3 Write a short note on Task Assignment Approach.

=> In this approach, every process is viewed as a collection of tasks which are scheduled to the suitable processor to improve the performance.

A process has already been split up into pieces called tasks. This split occurs along natural boundaries, so that each task will have integrity in itself and data transfers among the tasks are minimized.

- The amount of computation required by each tasks and the speed of each CPU are known.
- The cost processing each task on every node is known. The IPC cost is for tasks assigned to the same node. This is usually estimated by an analysis of the static program. If two tasks communicate n times and the average time for each inter-task communications is t, then IPC costs for the two tasks is n*t.
- Precedence relationship among the tasks are known
- Reassignment of tasks is not possible.
- If these things are known then ~~is~~ its a static allocation for which Task assignment Approach is used.

- Goal is to assign the tasks of a process to the nodes of a distributed system in such a manner as to achieve goals such as the following goals :

1. Minimization of IPC costs
2. Quick turnaround time for the complete process
3. A high degree of parallelism
4. Efficient utilization of system resources in general

There goals often conflict Example: while minimi-zing IPC costs tends to assign all tasks of a process to a single node, efficient utilization of system resources tries to distribute the tasks evenly among the nodes. So also quick, turnaround time and a high degree of parallelism encourage parallel execution of the tasks, the precedence relationship among the tasks limits there parallel Execution.

Also note that in case of m tasks and q nodes, there are $m^q$ possible assignments of tasks to nodes. In practice, however, the actual number of possible assignments of tasks to nodes may be less than $m^q$ due to the restriction that certain tasks cannot be assigned to certain nodes due to their specific requirments.
(eg. need a curtain amount of memory or a certain data file).

This is not a widely used approach because

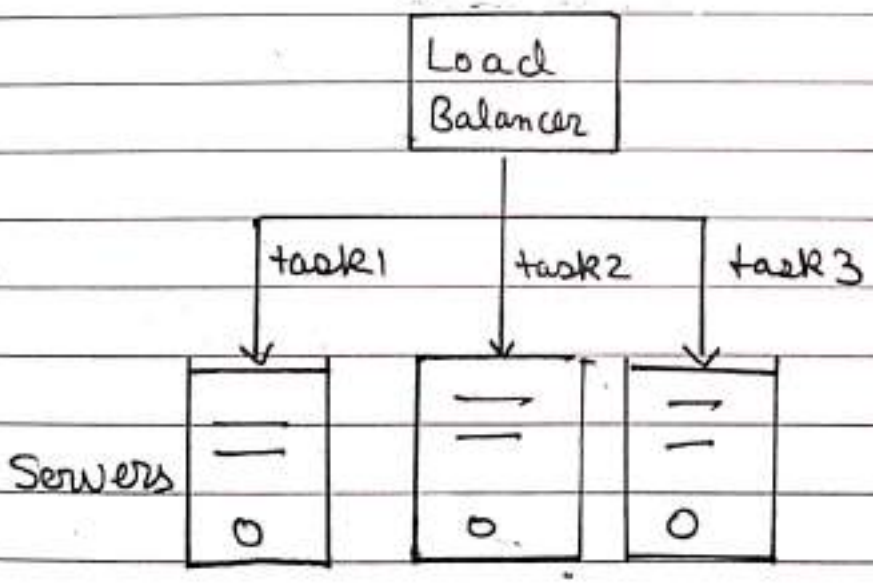
1. It requires characteristics of all the processes to be known in advance

2. This approach does not take into consideration the dynamically changing state of the system.

Q4 Write a short note on Load Balancing Approach.

=> Load balancing is a computer networking method to distribute workload across multiple computers or a computer cluster, network links, central processing rules, disk drives, or other resources

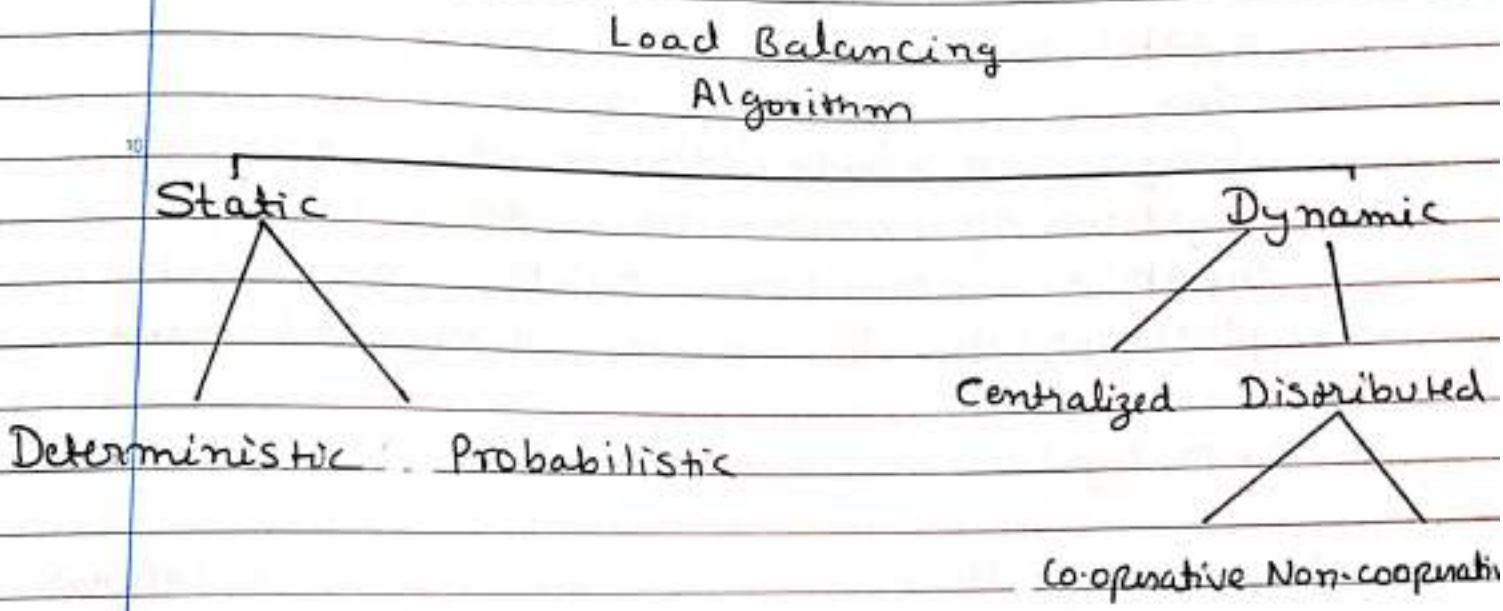
Goals of Load Balancing
- Achieve optimal resource utilization
- Maximize throughput
- Minimize response time
- Avoid overload
- Avoid crashing




Why we Load Balance
- Ease of administration / Maintainance
  Easily and transparently remove physical servers from rotation in order to perform any type of maintainance on the server.

Resource Sharing: can run multiple instances of an application/Service on a Server, can load balance to different part based on data analyzed.

- Types of Load Balancing




Static vs Dynamic

- Static Algorithm uses only information about the average behavior of the System
- They (Static Algo) ignore the current State or load of the nodes in the system. and also they are much Simpler
- Dynamic Algorithms collect State information and react to System state if it changed.
- Dynamic Algorithms are able to give significantly better performance.

- Deterministic vs Probabilistic

- Deterministic :

They use the information about the properties of the nodes and the characteristic of processes to be scheduled and this approach is difficult to optimize.

- probabilistic Algorithms :

They use information of static attributes of the system (eg. number of nodes, processing capability, topology) to formulate simple process placement rules. This approach has poor performance.

- Centralized VS Distributed.

- Centralized Algorithm : This approach collects information to server node and makes assignment decision.

- They can make efficient decisions, have lower fault-tolerance.

Distributed Algorithm :

- This approach contains entities to make decisions on a predefined set of nodes.

- They avoid the bottleneck of collecting state information and react faster.

o Co-operative versus Non-co-operative

Co-operative : Here, dissributed entities co-operate with each other. They are more complex and involve large overhead. Stability is better.

Non-co operative : Here, entities act as autonomous ones and make scheduling decisions independently from other entities.

Q5 write a short note on load sharing approach

=> Drawbacks of load Balancing approach

Load balancing technique with attempting equaliz-ing the workload on all the nodes is not an appropriate object since big overhead is generated by gathering exact state information.

Load Balancing is not achievable since number of processes in a node is always fluctuating and temporal unbalance among the nodes exists every moment.

→ Basic ideas for load-Sharing Approach

- It is necessary and sufficient to prevent nodes from being idle while some other nodes have more than two processes.

- Load-Sharing is much simpler than load balancing since it only attempts to ensure that no node is idle when heavily node exists.

Priority assignment policy and migration limiting policy are the same as that for the load balan-cing algorithms.

Since load-Sharing algorithms simply attempts to avoid idle nodes, it is sufficient to know whether a node is busy or idle.

- Thus these algorithm normally employ the simplest load estimation policy of counting the total number of processes.

- In modern systems where ~~parallel~~ permanent existance of several processes of an idle node is possible, algorithms measures CPU Utilization to estimate the load of a node.

Load sharing Approach - Process transfer policies

- Algorithms normally use all or nothing strategy
- This strategy uses the threshold value of all the nodes fixed to 1
- Nodes become receiver node when it has no process, and become Sender node when it has more than 1 process.
- To avoid processing power on nodes having zero process load sharing, algorithms use a threshold value of 2 instead of 1
- When CPU Utilization is used as the load estimation policy, the double threshold policy should be used as the process transfer policy.

- Location policy decides wether the sender node or the receiver node of the process takes the initiative to search for suitable node in the system, and this policy can be the following:
  - Sender-initiated location policy
    - Sender node decides where to send the process
    - Heavily loaded nodes search for lightly loaded nodes
  - Receiver-initiated location policy
    - Receiver node decides from where to get the process
    - Lightly loaded nodes search for heavily loaded nodes
  - In load-Sharing algorithms it is not necessary for the nodes to periodically exchange state information, but needs to know the state of other nodes when it is either underloaded or overloaded.
- Broadcast when State changes
  - In sender-initiated / receiver-initiated location policy a node broadcasts State Information Request when it becomes overloaded / underloaded
  - It is called broadcast-when-idle policy when receiver-initiated policy is used with fixed threshold value of 1.
- Poll when State changes
  - In large networks polling mechanism is used.

- polling mechanism randomly asks different nodes for state information until find an appropriate one or probe limit is reached.

- It is called poll-when-idle policy when receiver-initiated policy is used with fixed threshold value of 1.

Q6. Write a Short Note on Designing Issues In Load-Balancing Algorithms

1. Lode Estimation Policy

To balance the workload on all the nodes of the system, it is necessary to decide how to measure the workload of a particular node

- Some measurable parameters (with time and node dependent factor) can be the following:
  1. Total number of processes on the node
  2. Resource demands of these process
  3. Instruction mixes of these process
  4. Architecture and speed of the nodes processor

- Several lode balancing algorithms use the total number of processes to achieve big efficiency

- In some cases the true load could vary widely depending on the remaining service time, which can be measured in several way:
  - Memoryless method assumes that all processes have the same expected remaining service time, independent of the time used so far.
  - Pastrepeats' assumes that the remaining service time is equal to the time used so far
  - Distributed method states that if the distribution service times is known, the

associated processes remaining service time is the expected remaining time conditioned by the time already used.

- None of the previous methods can be used in modern systems because of periodically running processes and daemons.
- An acceptable method for use as the load estimation policy in these systems would be to measure the CPU utilization of the nodes.
- Central Processing Unit utilization is defined as the number of CPU cycles actually executed per unit of real time. It can be measured by setting up a timer to periodically check the CPU state (idle or busy)

2. Process transfer Policy

- Most of the algorithms use the threshold policy to decide on whether the node is lightly loaded or heavily loaded
- Threshold value is a limiting value of the workload of node which can be determined by

Static policy: predefined threshold value for each node depending on processing capability

Dynamic policy: threshold value is calculated from average workload and a predefined constant.

- Below threshold value node accepts processes to execute, above threshold value node tries to transfer processes to a lightly loaded node.
- Single threshold policy may lead to unstable algorithm because underloaded node could turn to be overloaded right after a process migration.
- TO reduce instability double-threshold policy has been proposed which is also known as high low policy.

Double threshold policy:

- When node is in overloaded region new local processes are sent to run remotely, request to accept remote processes are rejected.
- When node is in normal region new local processes run locally, requests to accept remote processes are rejected.
- When node is in underloaded region new local processes run locally, requests to accept remote processes are accepted.

3. Location Policy

- Threshold method
  - Policy selects a random node, checks wether the node is able to recieve the process then transfers the process. If node rejects another node is selected randomly. This continues until probe limit is reached.

- Shortest method
  - L distinct nodes are chosen at random, each is polled to determine its load. The process is transferred to the node having the minimum value unless its workload value prohibits to accept the process
  - Simple improvement is to discontinue probing whenever a node with zero load is encountered.
- Bidding method
  - Node contain managers (to send processes) and Contractors (to receive processes)
  - Managers broadcast a request for bid; contractors respond with bids (prices based on capacity of ~~good~~ contractor node) and manager selects the best offer
  - Winning contractor is notified and asked whether it accepts the process for execution or not
  - Full autonomy for the nodes regarding scheduling
  - Big communication overhead
  - Difficult to decide a good pricing policy.
- pairing
  - contrary to the former methods the pairing policy is to reduce the variance of load between pairs
  - Each node asks some randomly chosen node to form a pair with it.
  - If it recieves rejection it randomly selects another node and tries to pair again
  - Two nodes that differ greately in load are temporarily paired with each other and migration starts.

- The pair is broken as soon as the migration is over
- A node only tries to find a partner if it has atleast two processes.

4. State Information Policy
- Dynamic policies requir frequent exchange of state information, but these extra messages arise two opposite impacts:
  - Increasing the number of messages gives more accurate scheduling decision
  - Increasing the number of messages raises the quelling time of messages

- State Information policies can be the following:

o Periodic broadcast
- each node broadcast its state information after the elapse of every T units of time

- Problem: heavy traffic, fruitless messages, poor scalability since information exchange is too large for networks having many nodes

o Broadcast when state changes
- Avoid fruitless messages by broadcasting the state only when a process arrives or departures
- Further improvement is to broadcast only when state switches to another region (double threshold policy)

o On-demand exchange
- In this method a node broadcast a state-information-request message when its state switches from normal to either underloaded or overloaded region.

- on receiving this message other nodes reply with their own state information to the requesting node.

- Further improvement can be that only those nodes reply which are useful to the requesting node.

o Exchange by polling
- To avoide poor scalability (coming from broadcast message) the partner node is searched by polling the other nodes one by one, until poll limid is reached

5. Migration limiting Policy
This policy determines the total number of times a process can migrate.
o uncontrolled: A remote process arriving at a node is treated just as a process originating at a node, so a process may be migrated any number of times
o controlled.
- Avoides the instability of the uncontrolled policy
- Use a migration count parameter to fix

a limit on the number of times a process can migrate.

- Irrevocable migration policy : migration count is fixed to 1

- for long execution processes migration count must be greater than 1 to adapt for dynamically changing states.

6. Priority Assignment Policy

o Selfish: Local processes are given higher priority than remote processes. Worst response time performance of the three policies.

o Altruistic:
Remote processes are given higher priority than local processes. Best response time ~~performance~~ performance of the three policies

o Intermediate
When the number of local processes is greater or equal to the number of remote processes, local processes are given higher priority than remote processes.
Otherwise remote processes are given higher priority than local process.

Q Write a Short note on Load Sharing Approach.

=> Drawbacks of load balancing Approach

- Load balancing technique with attempting equilizing the workload on all the nodes is not an appropriate object since big ~~overhead~~ overhead is generated by gathering exact state information

- Load balancing is not achieveable since ~~exact~~ number of processes in a node is always fluctuating and temporal unbalance among the nodes exists every moment.

Basic ideas for load Sharing approach

- It is necessary and sufficient to prevent nodes from being idle while some other nodes have more than two processes

- Load Sharing is much simpler than load balancing Since it ~~always~~ only attempts to ensure that no node is idle when heavily node exists

- Priority assignment policy and migration limiting policy are the same as that for the load balancing algorithms

## Load Estimation policies

- Since load sharing algorithms simply attempt to avoid idle nodes, it is sufficient to know whether a node is busy or idle

- Thus these algorithms normally employ the simplest load estimation policy of counting the total number of processes

- In modern systems where permanent existence of several processes on an idle node is possible, algorithms measure CPU utilization to estimate the load of a node.

## Process Transfer policy

- Algorithms normally use all or nothing strategy

- This strategy uses the threshold value of all the nodes fixed to 1

- Nodes become reciever node when it has no process and become sender node when it has more than one process.

- TO avoid processing power on nodes having zero process load sharing algorithms use a threshold value of 2 instead of 1

- When CPU Utilization is used as the load estimation policy, the double-threshold policy should be used as the process transfer policy

## Location Transfer policy

- Location policy decides wether the sender node or the reciever node of the process takes the initiative to search for suitable node in the system, and this policy can be following:

1. Sender-initiated location policy
   - Sender node decides where to send the process
   - Heavily loaded nodes search for lightly loaded nodes.

2. Reciever-initiated location policy
   - Receiver node decides from where to get the process
   - Lightly loaded nodes search for heavily loaded nodes.

## State information exchange policies

- In load-sharing algorithms it is not necessary for the nodes to periodically exchange state information but needs to know the state of other nodes when it is either overloaded or under loaded.

Broadcast when State Changes
- In senders initiated / reciever-initiated location Policy a node broadcasts state information Request when it becomes overloaded / underloaded

- It is called broadcast when idle policy when receiver-initiated policy is used with fixed threshold value of 1

Poll when State Changes
- In large Networks polling mechanism is used
- Polling mechanism randomly asks different nodes for state Information until find and appropriate one or probe limit is reached
- It is called poll when idle policy when reciever initiated policy is used with fixed threshold value of 1

Note: Priority assignment policy and migration limiting Policy are same as load balancing algorithm so you can refer from previous question

Q. What is process Management.
This consists of 2 Parts:

- Conventional OS: Deals with the mechanisms and policies for sharing the processors of the system among all processes.

- Distributed Operating system: To make best possible use of the processing resources of the entire system by sharing them among all process

Three concepts to achieve this goal
- processor allocation: Deals with the process of deciding which process should be assigned to which processor

- process migration: Deals with the movement of a process from its current location to the processor to which it has been assigned.

- Threads: Deals with fine grained parallelism for better utilization of the processing capabilities of the system.

Q. Write a short note on process Migration

- The act of transferring a process between two machines during its execution is process Migration.
- Relocation of a process from the source node to destination node

~~process control process migration.~~

Process Migration is of two types

- Non-Preemptive process migration:
  In this approach, process is migrated from its current node to destination node before it starts executing on current node
- preemptive process migration: In this approach process is migrated from its current to destination node during its course of execution on current node. It is costlier than non-preemptive process migration as process environment also needs to transfer along with process to destination node.

Migration policy includes selection of process to be migrated and selection of destination node to which process should be migrated.

## Goals of process migration

1. Dynamic load distribution
2. Fault resilience
3. Improved system administration
4. Data access locality.

Involves three steps
- Selection of a process that should be migrated
- Selection of the destination node to which the selected process should be migrated
- Actual transfer of the selected process to the destination node.

## Process Migration Mechanism.

1. Mechanism for freezing and restarting the process

In preemptive process migration, usually snapshot of the process state is taken at its source node and then this snapshot is reinstated at its destination node. First process execution is suspended at its source node. Its state information is then transferred at destination node. After this, its execution is restarted at its destination node.

Freezing the processes at source node means its execution is suspended and all external interaction with the process is postponed.

2. Address Space Transfer Mechanism

- Following information is transferred when process is migrated from source node to destination node.
  - o process state: It includes execution status ie content of registers, scheduling information, memory used by process, I/O states, and its rights to access the objects which is capability list, processes identifiers, user and group identifiers of the process, information about file opened by process.
  - o process Address space: It includes code, data and stack of the program.
- Size of process state information is in few kilobytes. Whereas, process address space is of large size. Process address space can be transferred without stopping its execution on source node. However, while transferring state information, it is necessary to stop execution of the process.

3. Message-Forwarding Mechanism

It should be ensured that, the migrated process should receive all its pending, messages in transmission and future messages at destination node.

Following types of messages needs to be forwarded at destination node of migrated process.

Type 1 : Messages which are received at source node after process execution has been stopped and its execution has not until now been commenced at destination node.

type 2 : Message received at source node after process's execution started at destination node

type 3 : Message expected by already migrated process after it starts its execution on destination node.

4. Mechanism for Handling coprocesses :

# There should be efficient communication between parent and child processes if they are migrated and placed at different nodes.

-- Not allowing Separation of Process :

* process that wait for one or more of their children to complete are not allowed to migrate. The drawback of this approach is that it does not permit use of parallism with job. It is due restricting the various tasks of job to distribute on different nodes.

* If parent process migrates then its children processes will be migrated along with it. It is used by V System. Large overhead is the drawback.

Q. Features of Process migration

1. Minimal Interference:
   - can be done by minimizing freezing time
   - Freezing time: a time for which the execution of the process is stopped for transferring its information to the destination node.

2. Minimal residual dependencies
   - Migrated process should not continue to depend on its previous node once it has started executing on new node.

3. Efficiency
   - Time required of migrating a process
   - The cost of locationg an object
   - The cost of supporting remote execution once the process is migrated.

4. Robustness
   - The failure of a node other than the one on which a process is currently running should not affect the execution of that process.

5. - communication between coprocesses of a job: of processes of the single job are distributed over several nodes for parallel execution then these processes should be able to directly interact with each other

irrespective of their location.

Q. Advantages of Process migration

1. Improves System security: this can be achieved by migrating sensitive process to more secure node.

2. Improving system reliability: Critical process can be migrated to node with higher reliability in order to improve reliability.

3. Gaining Higher Throughput:
As all the CPUs are better utilized with proper load balancing policy, higher throughput can be achieved. By properly mixing CPU and I/O bound jobs globally, throughput can ~~increased~~ be increased.

4. Reducing Network traffic
- Migrate the process closer to the resources it is using most heavily
- To migrate and cluster two or more processes which frequently communicate with each other, on the same node.

4. Reducing Network traffic: Migrating processes closer to the resources or to the node having their required resources, reduces network traffic as resources are accessed locally. Instead of accessing huge database remotely, it is better to move the process to the node where such database is present.

5. Speeding up individual job:
The tasks of the job can be migrated to different node. Also job can be migrated to node having faster CPU. On both cases, execution of the job speeds up.

Q. Write a Short Note on Threads.

1. Program in execution is called as process and Thread is a part of process

a. Each thread belongs to exactly one process

b. Threads can share common data so they do not need to use interprocess communication

c. Like processes, threads also have states like ready, executing, blocked etc. priority can be assigned to the threads just like process and highest priority thread is scheduled first.

Advantages of threads.

a. Thread is a light weight process

b. New thread creation takes less time. (as compared to process creation)

c. New thread Termination takes less time (as compared to new process creation).

d. In Multi threaded Server implementation, if one thread is blocked and waiting, Second thread in the same process could execute.

e. Thread context Switch takes less time as compared to process context switch.

Q compare between Process and Thread

| | Process | Thread |
|---|---|---|
| 1. | Program in execution is called as process. It is heavy weight process. | Thread is part of process. It is also called a light weight process |
| 2. | Process context switch takes more time as compared to thread context switch because it needs interface of operating System | Thread context switch takes less time as compared to process context switch because it needs interrupt to Kernel only. |
| 3. | New process creation takes more time as compared to new thread creation | New thread creation takes less time as compared to new process creation. |

| | | |
|---|---|---|
| 4. | New process termination takes more time as compared to new thread termination | New thread termination takes more time as compared to new process termination |
| 5. | In process based implementation, if one process is blocked no other server process can execute until the first process is unblocked | In multithreaded server implementation if one thread is blocked and waiting, second thread in the same process could execute. |

8. Write a short note on Threads in Distributed System (you can write this information for any question on threads)

- An important property of threads is that they can provide a convenient means of allowing "blocking systems calls without blocking the entire process in which the thread is running."

- Threads used to express communication in the form of multiple logical connection at the same time

- A main contribution of thread in DS is that they allow client and server to be constructed such that communication and local processing can overlap, resulting in high level of performance.

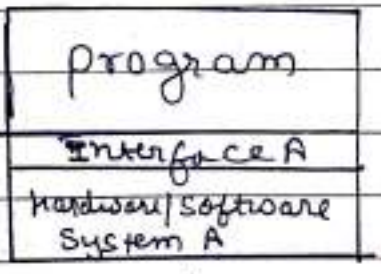
Q write a short note on Virtualization

=> From availability and security point of view using separate computer to put each service is more preferable by organizations. If one server fails other will not be affected.

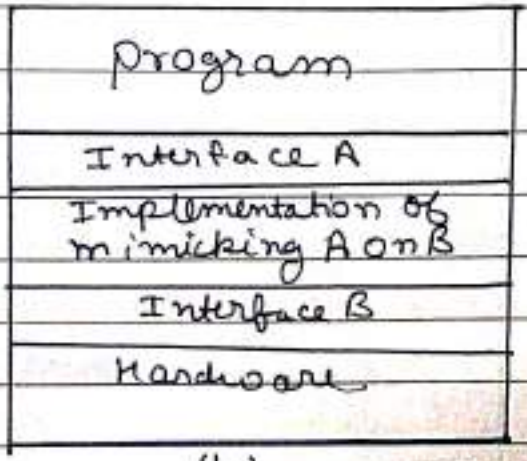
- It is also beneficial in case if organizations need to use different types of operating systems. However keeping separate machine for each service is costly. Virtual machine technology can solve this problem. Although this technology seems to be modern but idea behind it is old.

- Multiprogrammed operating systems provide the illusion of simultaneous execution through resource virtualization ie use software to make it look like concurrent processes are executing simultaneously.

- Virtualization machine technology creates separate virtual machines, capable of supporting multiple instances of different operating systems.




(a)




(b)

figure (a) General organization between a program, interface, and System.

(b) General Organization of virtualizing System A on top of system B.

benifits of Virtualization

1. Manageability: The ability to move, copy and isolate VMs

2. Suetainability: Energy Savings by the way of less hardware and electricity

3. Availability: the ability to snapshot, clone, and run redundant VMs

4. security: isolation of VMs and applications

Q. Write a short Note on Code Migration

=> There can be some situations in a distributed System where some program (may be being executed) needs to be transfered from one node to another. This is called code migration

- In process migration an entire process has to be moved from one machine to another.

- Through moving a running process to another machine is a Crucial task, but it has a good over all performance.

- But in some cases code needs to be migrated rather than the process. For example consider a client-server System in which the server hasto manage a big database.

- The client application has to perform many database operations

- In this situation it is better to move part of the client application to the server and the result is sent across the network.
- Thus ~~mig~~ code migration is a better Solution

## Reasons of code Migration

1. Improve computing performance by moving process from Heavily loaded machine to lightly loaded Machine

2. Improve communication time by Shipping code to System where large dataset reside.

- If code migration is supported in a System it improves the performance of the Over all System by exploiting parallelism. It becomes possible to dynamically configure the distributed system.

- Assume that a Server implements a Standardized Interface to a file System. If remote clients ~~and~~ are to be allowed to acess the file, the Server should use a proprietory protocol.

- In such case the client side implementation of file system interface must be linked with the application.

- It should also be based on the protocol used by the server. In this approach the software must be readily available to the client, during the development of the clients application. This is not possible in all cases.

- The other approach could be making the server responsible to provide the clients implementation when the client binds to the server.

- Thus the client can download the implementation dynamically.

- It then goes through the necessary initiation steps, and involves the server when required

- The advantage of this approach is that the clients need not have to install all the required software.

- The software can be moved in as and when necessary and discarded when it is not needed.

# Consistency Replication Fault Tolerance

## What is consistency and Replication

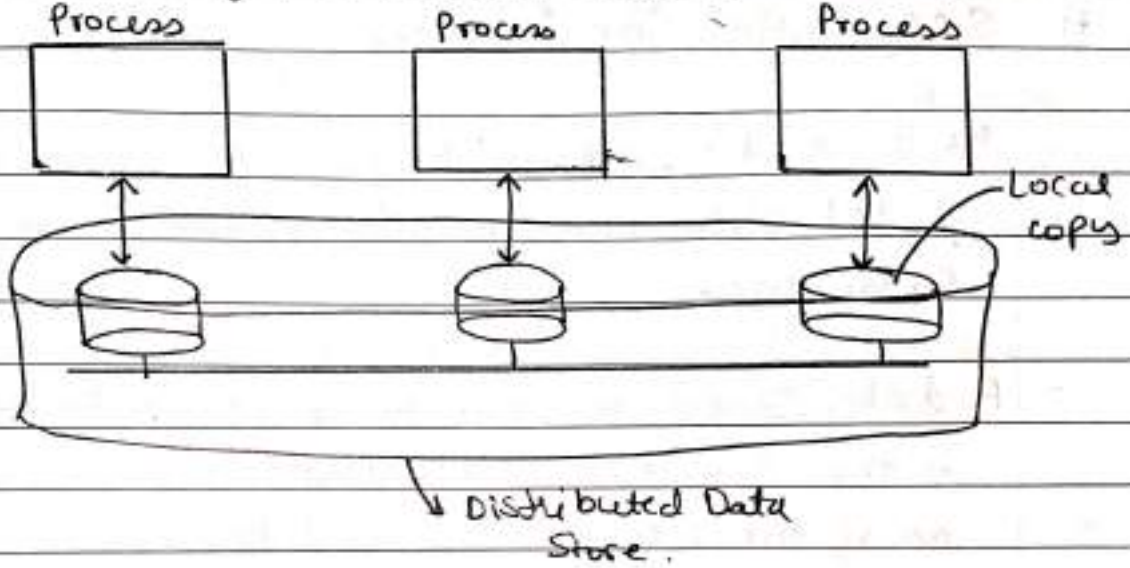
The Replication of data is carried out in Distributed system in order to enhance the reliability and to improve performance. Although reliability and performance can be achieved through replication, it should be ensured to have all the copies of data consistent. If one copy updates then this update should be propagated to other copies of data also. Otherwise Replica will not be same.

The five most important criteria which bring different aspects of consistency application requirments are as follows:

- concurrency - The degree at which the conflicting read/write access is tolerated
- consistency - The degree of tolerance of the update dependency and state read.
- Availability: The method and time to access data and replica.
- Visibility: Presenting a singular, global view, even while local changes are applied to the replicated data.
- Isolation: The time in which the remote updating is reflected locally

Q Write a short Note on Data centric consistency Model

=> A data Store may be physically distributed across multible machines. Each process that can access data from the store is assumed to have a local or nearby copy available of the entire store.




# Strict consistency Model

- Any read on a data item x returns a value corresponding to the result of the most recent write on x

- This is the strongest form of memory coherance which has the most stringent consistency requirment

- Strict consistency is the ideal model but it is impossible to implement in a distributed system. It is based on absolute global time or a global agreement on commitment of changes.

P1 W(x)a ✓
P2 R(x)a

P1 W(x)a X
P2 R(x)NIL R(x)a

## Sequential Consistency

- It is an important data centric consistency model which is a slightly weaker than strict consistency.

- A data store is said to be sequentially consistent if the result of any execution is the same as if the (read and write) operations by all process on the data store were executed in some sequential order and the operations of each individual process should appear in this sequence in a specified order.

- Example: Assume 3 operations read(R1), write(W1), read(R2) performed in an order on a memory address. Then (R1, W1, R2), (R1, R2, W1), (W1, R1, R2), (R2, W1, R1) are acceptable provided all processes see the same ordering.

PRP

# Causal consistency

- It is a weaker model than sequential consistency
- In Causal consistency all processes see only those memory reference operations in the correct order that are potentially causally related
- Memory reference operations which are not related may be seen by different processes in different order.
- A memory reference operation is said to be causally related to another memory reference operation if the first operation is influenced by the second operation.
- If a write (W2) operation is causally related to another write (W1) the acceptable order is (W1, W2)

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

| | | | | |
|---|---|---|---|---|
| P1 | W(x)a | | | |
| P2 | | R(x)a | W(x)b | |
| P3 | | | R(x)b | R(x)a |
| P4 | | | R(x)a | R(x)b |

(a) ✗

| | | | |
|---|---|---|---|
| P1 | W(x)a | | |
| P2 | | W(x)b | |
| P3 | | R(x)b | R(x)a |
| P4 | | R(x)a | R(x)b |

✓

# FIFO / PRAM consistency model

- It is weaker than causal consistency
- This model ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed like a single process in a pipeline.
- This model is simple and easy to implement having good performance because processes are ready in one pipeline.
- Implementation is done by sequencing write operations performed at each node independently of the operations performed on other nodes.

- P1 W(x) a
- P2 R(x)a W(x)b W(x)(
- P3 R(x)b R(x)a R(x)(
- P4 R(x)a R(x)b R(x)(

# Weak consistency model

- The basic idea behind the weak consistency model is enforcing consistency on a group of memory reference operations rather than individual operations
- A distributed shared memory system that supports the weak consistency model uses a special variable called a synchronization variable which is used to synchronize memory

- When a process accesses a Synchronization variable, the entire memory is synchronized by making visible the changes made to the memory to all other process

# Entry consistency Model

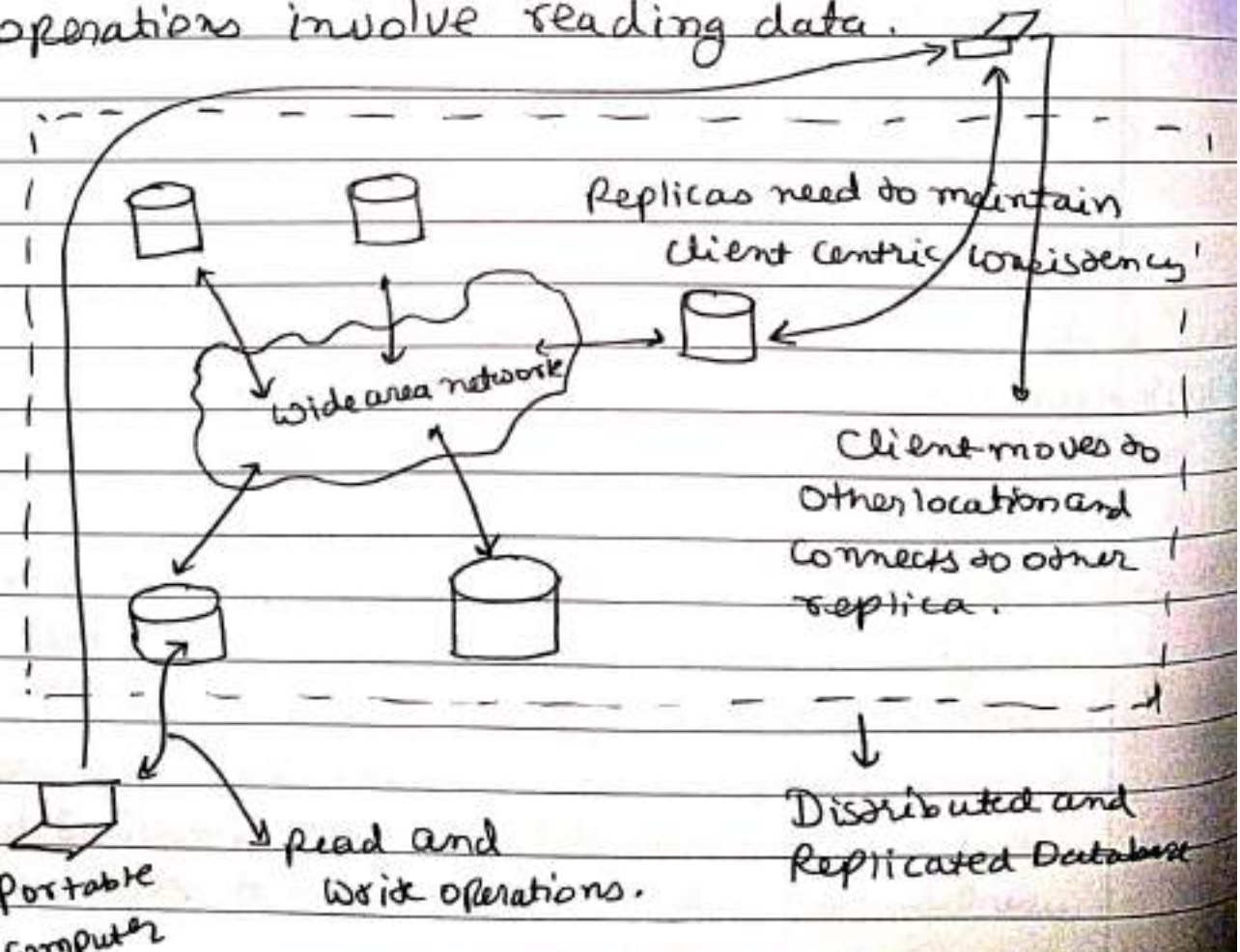
- In this model every shared data item is associated with a Synchronization variable.

- In order to access consistent, each Synchronization variable must be explicitly acquired.
- Release consistency affects all shared data but entry consistency affects only those shared data associated with a Synchronization Variable

Q Write short note on client centric consistency model.

=> Client centric consistency models aim at providing a system wide view on a data store.

- This model concentrates on consistency from the perspective of a single mobile client.
- Client-centric consistency models are generally used for applications that lack simultaneous updates where most operations involve reading data.



# Eventual Consistency Model

- In systems that tolerate high degree of inconsistency, if no updates take place for a too long time all replicas will gradually and eventually become consistent. This form of consistency is called Eventual Consistency.

- Eventual consistency only requires those updates that guarantee propagation to all replicas.

- Eventual consistent data stores work fine as long as clients always access the same replica.

- Write conflicts are often relatively easy to solve when assuming that only a small group of processes can perform updates. Eventual consistency is therefore often cheap to implement.

# Monotonic Reads Consistency Model

- A data store is said to provide monotonic-read consistency if a process reads the value of a data item X, any successive read operation on X by that process will always return that same value or a more recent value

- A process has seen a value of X at time t, it will never see an older version of X at a later time.

Example: A user can read incomming mail while moving. Each time the user connects to a different email server, that server fetches all the updates from the server that the user previously visited. Monotonic Reads guarantees that the user sees all updates, no matter from which server the automatic reading takes place.

# Monotonic writes consistency model

- ~~A day's~~ A data store is said to be monotonic write consistent if a write operation by a process on a data item x is completed before any succesive write operation on x by the same process.

- A write operation on a copy of data item x is performed only if that copy has been brought up to date by means of any preceding write operations, which may have taken place on other copies of x

- Example: Monotonic - write consistency guarentees that if an update is performed on a copy of server S, all preceding updates will be performed first. The resulting server will then indeed become the most recent version and will include all updates that have led to previous versions of the server.

# Read your writes consistency model

- A data store is said to provide read your writes consistency if the effect of a write operation by a process on data item X will always be a successive read operation on X by the same process.

- A write operation is always completed before a successive read operation by the same process no matter where that read operation takes place.

- Example: Updating a web page and guaranteeing that the web browser shows the newest version instead of its cached copy.

# Writes follow Read consistency model

- A data store is said to provide write-follow-reads consistency if a process has write operation on a data item x following a previous read operation on x than it is guaranteed to take place on the same or a more recent value of x that was read.

- Any successive write operation by a process on a data item x will be performed on a copy of x that is up to date with the value most recently read by the process.
- Example: Suppose a user first reads an article A then posts a response B. By requiring writes-follow reads consistency, B will be written to any copy only after A has been written.

Q. Replica Management.

- The number of replicas and their location decide the availability of a replica.
- These factors also effect the access latency and load balancing that is achieved due to replication
- An availability of one replica should not have any impact on the availability of the rest of the replicas
- This implies that the replication management is a location independent activity.
- We need to decide where, when and by whom copies of the data store are to be placed.
  Three types of copies
  - permanent ~~co~~ replicas
  - Server-initiated replicas
  - Client initiated replicas.

Permanent Replicas: Number of permanent replicas is and so be small ie often the initial set of replicas.
It is the main Replica created by the originator If a website is developed by Son then the Owner decides how many replica he wants to create.

Server Initiated Replica : Suppose the owner decides that the website would be placed on five server then those five servers would be deciding how many replica further to be created so that the client can get access to those replica easily

Client Initiated Replica : If sometimes we hit on a particular website on a regular bases so the cookies are developed of that website in the system so that we can access that website more easily, frequently and in less time.

Replication models

- Master - Slave - one copy is the master replica and all other copies are slaves.
- Client - Server : Just like master slave model. one replica is designed as server replica, but modification can happen at any of the clients too.
- peer to peer : All replicas are of equal importance and are peer.

Replication Consistency

Managing consistency for replicas may reduce system performance, so there is compromise called read only replicas

optimistic : These schemes assumes that faults are rare and implement recovery schemes when inconsistency occurs

pessimistic: These schemes says that the faults are common, and take necessary actions to ensure consistency of every accss

Q write short Note on Fault Tauleranse

- Hardware, Software and networks cannot be totally free from failures.

- Fault tolerance is a non-functional requirment that requires a system to continue to operate, even in the presence of faults.

- Fault tolerance should be achieved with minimal involvement of users or system administrators

- Distributed Systems can be more fault tolerant than Centralized Systems, but with more processor hosts generally the occurence of individual faults is likely to be more frequent.

- Notion of a partial failure is a distributed system.

=> System attributes:

- Availability: System always ready for use, or probability that System is ready or available at a given time.

- Reliability: property that a system can run without failures for a given time

- Safety - indicates the safety issues in the case the system fails

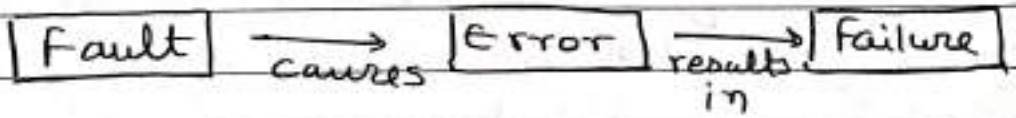
- Maintainability: refers to the ease of repair to a failed system.

- Failure in distributed system = when a service cannot be fully provided

- System failure may be partial

- A single failure may affect other parts of a System.

Terminology of Fault Tolerance




Fault - is a defect within the System

Error - is observed by a deviation from the expected behaviour of the System.

- Failure occures when the system can no longer perform as required ( does not meet spec )

- Fault Tolerance - is ability of system to provide a servia / even in the presence of errors.

Failure Model

1. Omissionfailure: A server fails to respond to incoming request

   Recive omaission : A server fails to receive incomming messages

   Send Omission : A server fails to send messages.

2. Timming failure : A Server's response lies outside the Specified time interval

3. Arbisrary failure : A server may produce arbisrary responses at arbisrary times.

# Distributed File Systems and Name Service

Q Write a Short Note on Distributed File System.

=> File System Describes how files are structured, name, accessed, used, protected and implemented

In distributed System, files are available on several computers. Computers in distributed System can share these physically dispersed files by using distributed File System.

DFS Supports the following:

- Remote Information sharing: Any node in the system can transparently access file irrespective of its location.
- User mobility: DFS allows the user to work on different nodes at different times without physically relocating the secondary storage devices
- Availability: DFS keeps multiple copies of file on different nodes. Failure of any node or copy does not affect the operation.
- Diskless workstation: DFS provides transparent file accessing capability, hence, economical diskless workstations can be used.

DFS Provides following three types of services.

1. Storage Service: DFS Deals with storage and management of space on secondary storage device that is used to store files in file system. File can be stored in blocks.

2. True file service: True file service supports operations on individual file. These are creating and deleting files, modifying and accessing data from files.

3. Name Service: Name service offers mapping between file names to file IDs. It is difficult for human to remember file ID's. Most file system uses directories to carry out mapping called as directory service. The service supports creation, deletion of directories etc.

Q Write Down Features of Distributed File System:

1. Transparency:

Transparency refers to hiding details from a user. The following types of transparency are desirable.

- ~~Server~~ Structure Transparency:
  There is no need for the client to know about the number or locations of the servers and the storage device. Multible file servers should be provided for performance, adaptability and dependability.

- Access Transparency:
  Both local and remote files should be accessible in the same manner. The file system should be automatically located on the accessed file and send it to the client's side.

- Naming Transparency: Name of file should remain same when it is moved from one node to other. Its name should be location dependent.

- Replication Transparency: Existance of multiple replicated copies and their location should remain hidden from clients.

2. User mobility: It will automatically bring the user's home directory to the node where the user logs in. This is because DFS should permit the user to work on different nodes at different times.

3. Performance: DFS must give performance same as centralized file system. User should not feel the need to place file explicitly to improve performance.

4. Simplicity and ease to use: The user interface of a FileSystem should be simple and the number of commands in one file should be small.

5. Scalability: DFS should continue to function if partial failure occurs in one

5. Scalability: DFS should also support for growth of nodes and users in network. Such growth should not lead to service loss or performance degradation of the system.

6. High Availability: A distributed file system should continue to function even in partial failures such as a link failure, a node failure, or a storage divice crash. Replicating files at multiple servers can help achieve availability.

7. High Reliability: Probability of loss of stored data should be minimized. System should automatically generate backup copies of critical files in event of loss.

8. Data integrity: Simultaneous accesses to shared file by many users should guarantee the integrity of data stored in it. These access requests to file from many users should be properly synchronized by using proper concurrency control mechanism.

9. Security: A distributed file system must secure data so that its users are confident of their privacy. File system should implement mechanisms to protect data that is stored within

10. Heterogeinity: Distributed file System should allow various types of workstations to participate in sharing files via distributed file system. Integration of a new type of workstation or storage media should be designed by a DFS.

Q. Write Short Note on File Models

=> There are two types of File models
(1) Structured and Unstructured files
(2) Mutable and Immutable files

(1) Structured files: In this file is presented to file server as ordered sequence of record. Record is smallest unit of data that can be accessed.

(2) Unstructured Files: File is unstructured sequence of data. Structure of the file appears to file server as uninterpreted sequence of bytes.

(3) Mutable:
- an Update performed on a file overwrites on its old contents

- A file is represented as a single stored sequence that is changed by each update operation

(4) Immutable Files

- A file cannot be modified once it has been created

- ~~Reverse~~ In other words you can say that in this file model, each update operation creates new version of the file. Changes are made in new version and old version is retained. Hence more Storage Space is required

Q. Write a short note on file accessing Models

=> Accessing Remote Files
* Remote Service Model
* Data ~~ter~~ Caching Model

Unit of Data Transfer
* File-level Transfer Model
* Block-level Transfer Model
* Byte-level Transfer Model
* Record-level Transfer Model

Remote Service Model: Processing of a client's request is performed at the servers node.

- Thus, the clients request for file access is delivered across the network as a message to the server, the server machine performs the access request, and the result is sent to the client.

Data Caching Model:

- This model attempts to reduce the network traffic of the previous model by caching the data obtained from the server node. This takes advantage of the locality feature of the found in file accesses.

- A replacement policy such as LRU is used to keep the cache size bounded.

- While this model reduces network traffic it has to deal with the cache coherency problem during writes, because the local cached copy of the data needs to be updated, the original file at the server node needs to be updated and copies in any other caches need to be updated.

## Unit of Data Transfer:

- In file systems that use the data-caching model, an important design issue is to decide the unit of data transfer.
- This refers to the fraction of a data that is transferred between clients and servers due to single read or write operation.

### File level transfer model:

On this model when file data is to be transferred, the entire file is moved.

### Block level transfer model:

File transfer takes place in file blocks. A file block is a contiguous portion of a file and is of fixed length.

## Byte level Transfer Model:

In this model, transfer of file data between client and server takes place in units of bytes. It offers flexibility as any range of data bytes within file can be requested and hence, it is difficult to manage cached data due to its variable length. Cambridge File server uses this model.

## Record level Transfer Model:

This model is applicable to structured files only. Hence, records are transferred between client and server. Research storage system uses this model.

Q. Write a short note on File Replication

=> Replicating the file on many machines improves availability which is one desirable feature of good distributed file system and File Replication is the primary mechanism for improving file availability.

Replication and caching

- Replica is created at server while cached copy is associated with clients.

- A replica is more persistant as compared to cached copy and Replica is widely known, secure, accurate, available and complete too.

- The existance of cached copy depends on locality in access patterns. Existence of replica depends on availability and performance.

- cache copy is dependent on replica. It needs to be periodically verified with replica then it becomes useful.

Advantages of Replication

1. Improved Response Time: Replication allows data to be accessed locally or from the node whose access time is less than that of primary copy access time.

2. Reduced Network Traffic : If Replica of file is available with file server that resides on client machine then client access request is serviced locally. Hence it results in reduced network traffic.

3. Improved System Throughput: As different clients requests are serviced by different servers in parallel, it results in improved throughput.

Q. Write a Short Note on Network File System

- In Network File System there are multiple client machines and one or a few servers. The server stores data on its disks and the clients may request data through some protocol messages.

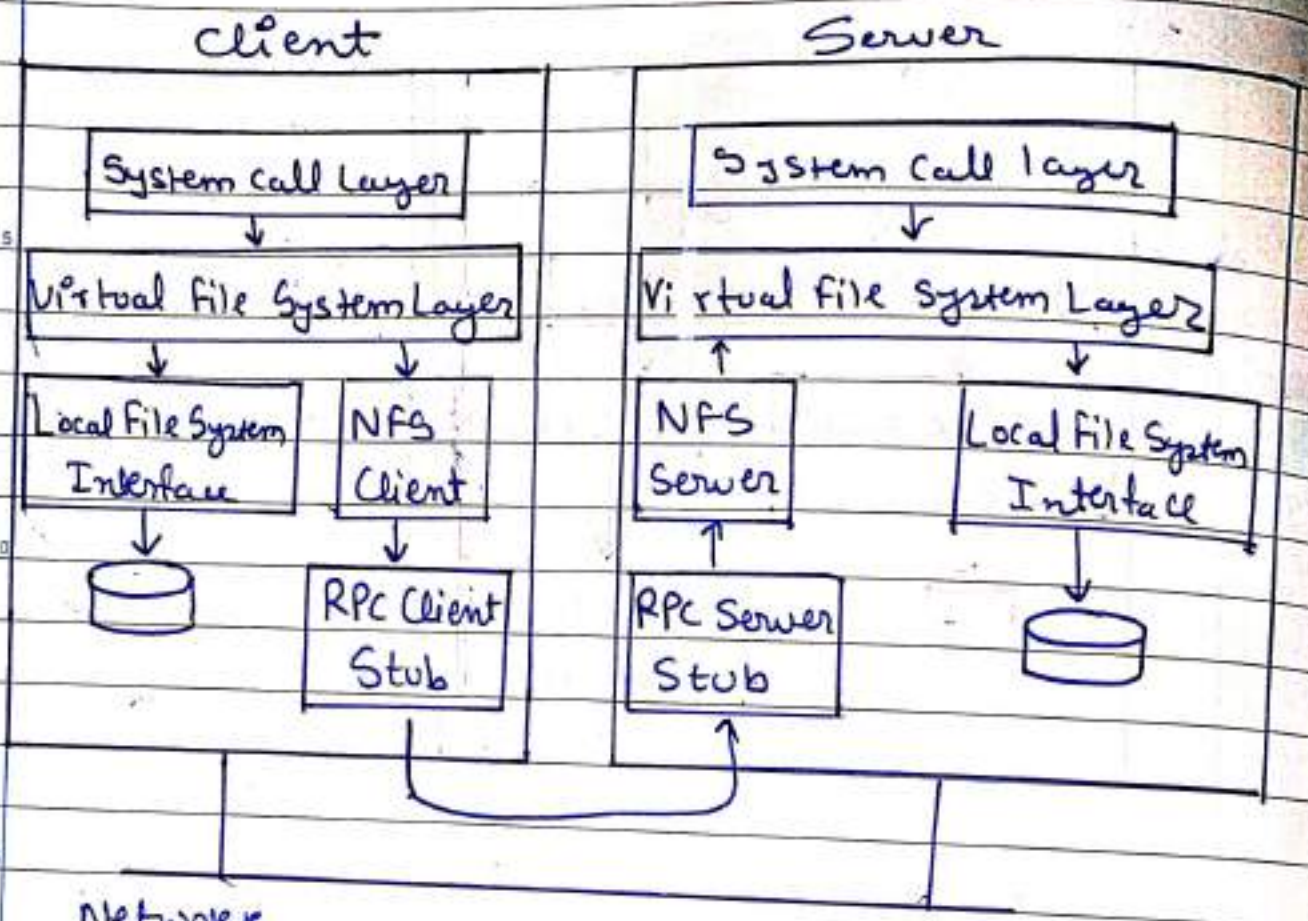
- NFS allows the user or system administrator to mount all or portion of file system on the server.

- The portion of file system can be accessed by client with whatever privilege are assign to each file

Advantage of a Distributed file system: (Sun Micro System's NFS is an example of DFS)

- Provides centralized administration.
- Provides security, ie one must only secure the Servers to secure data.

- It is machine Independent.




Network File System Architecture

- A client application issues a system call (eg write(), read(), Open(), Close() etc.) to access files on the client side file system, which in turn retrieves files from the server

- It is interesting to note that to a client application, the process seems no different from requesting data from a physical disk, since there is no special API required to do so. This phenomenon is known as transparency in terms of file access.

- It is the client-side File System that executes commands to service these System Calls.

## Sun's Network File System

- Network File System was originally developed by Sun Microsystems

- NFS V2 was the standard protocol followed for many years, designed with the goal of simple and fast Server crash recovery.

- Stateful protocols make things complicated when it comes to crashes. Consider a client x 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 x issues the second read request. However the server does not know which file the client is referring to, since all the information was temporary and lost during the crash.

- Stateless protocols came 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

- When the server receives a mount request, it returns a file handle to the client.

- File handle is basically a data-structure of length 32 bytes. It serves as the key for further accesses to files within the mounted system

- 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 are metadata associated with computer files including file creation time, last modified, size, ownership permission etc
- In NFS ~~fore~~ four attributes are classified as mandatory attributes that every implementation must support and set of recommendation attributes
- ~~Fo~~ Following are mandatory attributes
  - TYPE: type of file
  - SIZE: length of File is bytes
  - CHANGE: Indicator for client to check if when one file has Changes
  - FSID: Server unique identifier of files file System.
- Recommendation File Attributes
  - OWNER: Owner of the File
  - TIME-CREATE: time when file was created
  - TIME-ACCESS: Last time of accessing the data of file etc.

## Synchronization

- DFS allows to share its files among multiple clients. These Shared files between multiple Users should remain consistent, for this purpose Synchronization is needed.

- In NFSv4 file locking is integrated in NFS file access control protocol

- Following operation are provided for locking for write operation

  Lock: Create a lock for range of bytes
  Lockt: Test whether conflicting lock has been granted.
  Locku: Remove a lock from range of bytes
  Renew: Renew a lease on specific lock.

Client - side caching

- To improve performance of NFS, DFS's cache the data as well as the metadata read from the server onto the clients. This is known as client-side caching.

- This reduce the time taken for subsequent client accesses.

- The cache is also used as a temporary buffer for writing

- This helps improve efficiency even more since all writes are written onto the server at once

- Modification to the cache is immediately forwarded to the server

# Fault tolerance and Security

- RPC mechanism in NFS does not ensure guarantee regarding reliability.
- In case of loss of server replay, server will process retransmitted request by client.
- This problem is solved by means of duplicate request cache implemented by server.
- Each client request carries transition id that is cached by server when request arrives at server, after processing the request server also caches reply.
- After timer at client expires before reply come back then client retransmits same request with same XID
- Three cases occur
  1. Server not completed original request, it ignore retransmitted request.
  2. Server may get retransmitted request after reply sent to client, if arrival time of reply and retransmitted request is same Server ignores it.
  3. If reply really got lost cache reply is sent to retransmitted request.

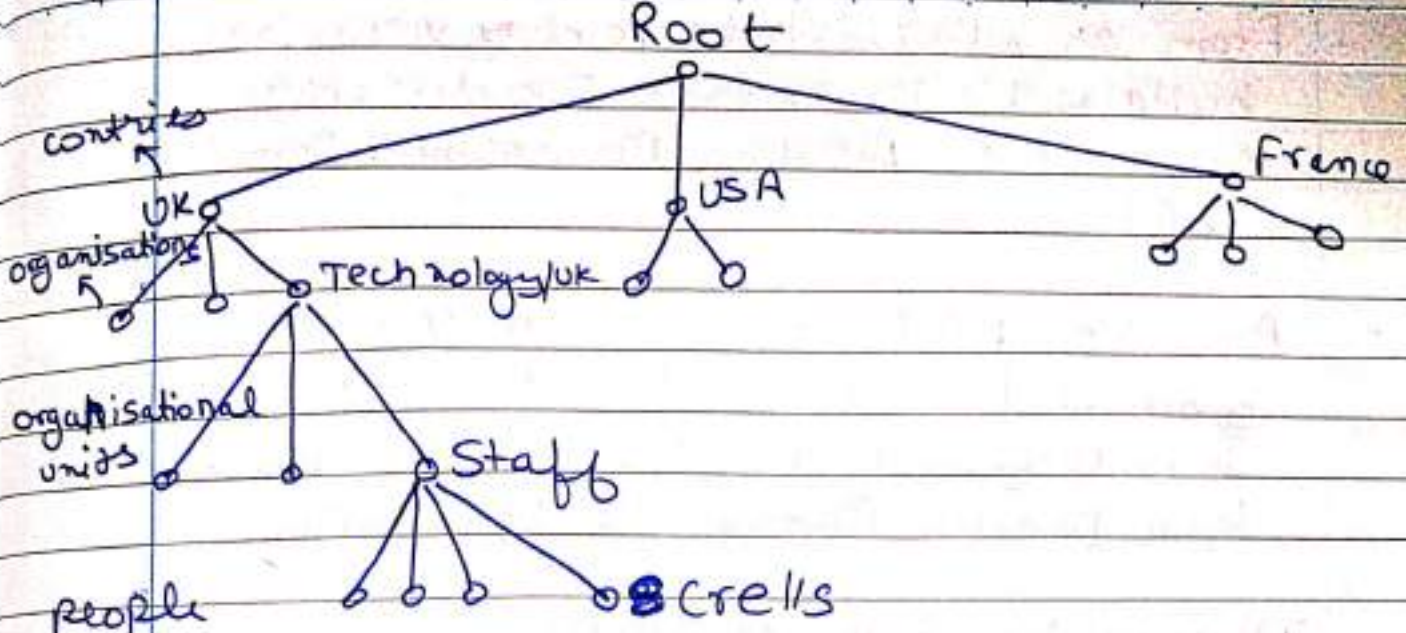
In case of lock crash lease is given to all locks and for establishing a session NFS use numbers

Q Explain or write Short Note on X.500 directory service.

- X.500 is a series of computer networking standards covering electronic directory services.

- The X.500 series was developed by the Telecommunication standardization sector of the international Telecommunications union

- It specifies a global hierarchical directory service.




## Features of X.500

1. A standards-based directory service for those applications that require directory information.
2. A single global hierarchical namespace of objects and their attributes.
3. Data management functions for viewing, adding, modifying and deleting directory objects.
4. Search capabilities for customizing complete data queries.

## Working of X.500

- X.500 defines a global directory service that consists of several components.

- From an administrative point of view, the building blocks of the X.500 directory service are Directory Management Domain (DMDs)

- An X.500 DMD is a collection of X.500 components that includes at least one Directory System Agent and is managed by a Domain Management Organization.

There are two types of DMDs

i) Administrative Directory Management Domains (ADDMDs): Directory Services managed by a registered private agency that provide public directory services. Examples of ADDMDs are Four11 and Big Foot, which provide public X.500 directly services over the Internet.

ii) Private Directory Management Domains: Directory services that provide private directory access. An example is a domain controller hosting Active Directory on a network running Microsoft Windows Server

- X.500 has 3 major components which are - Directory Information Base (DIB), Directory System Agent (DSA), (Directory User Agents (DUAs)