# Solution DC Viva Questions

1. **Issues in DC**

Ans:

Issues in designing distributed systems:

**1. Heterogeneity**

The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks.Internet consists of many different sorts of network their differences are masked by the fact that all of the computers attached to them use the Internet protocols to communicate with one another.For eg., a computer attached to an Ethernet has an implementation of the Internet protocols over the Ethernet, whereas a computer on a different sort of network will need an implementation of the Internet protocols for that network.

**2. Openness**

The openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways.The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.

**3. Security**

Many of the information resources that are made available and maintained in distributed systems have a high intrinsic value to their users.Their security is therefore of considerable importance. Security for information resources has three components: confidentiality, integrity, and availability.

**4. Scalability**

Distributed systems operate effectively and efficiently at many different scales, ranging from a small intranet to the Internet. A system is described as scalable if it will remain effective when there is a significant increase in the number of resources and the number of users.

**5. Failure handling**

Computer systems sometimes fail. When faults occur in hardware or software, programs may produce incorrect results or may stop before they have completed the intended computation. Failures in a distributed system are partial – that is, some components fail while others continue to function. Therefore the handling of failures is particularly difficult.

**6. Concurrency**

Both services and applications provide resources that can be shared by clients in a distributed system. There is therefore a possibility that several clients will attempt to access a shared resource at the same time. Object that represents a shared resource in a distributed system must be responsible for ensuring that it operates correctly in a concurrent environment. This applies not only to servers but also to objects in applications. Therefore any programmer who takes an implementation of an object that was not intended for use in a distributed system must do whatever is necessary to make it safe in a concurrent environment.

**7. Transparency**

Transparency can be achieved at two different levels. Easiest to do is to hide the distribution from the users. The concept of transparency can be applied to several aspects of a distributed system.

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

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

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

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

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

**8. Quality of service**

Once users are provided with the functionality that they require of a service, such as the file service in a distributed system, we can go on to ask about the quality of the service provided. The main nonfunctional properties of systems that affect the quality of the service experienced by clients and users are reliability, security and performance. Adaptability to meet changing system configurations and resource availability has been recognized as a further important aspect of service quality.

**9. Reliability**

One of the original goals of building distributed systems was to make them more reliable than single-processor systems. The idea is that if a machine goes down, some other machine takes over the job. A highly reliable system must be highly available, but that is not enough. Data entrusted to the system must not be lost or garbled in any way, and if files are stored redundantly on multiple servers, all the copies must be kept consistent. In general, the more copies that are kept, the better the availability, but the greater the chance that they will be inconsistent, especially if updates are frequent.

**10. Performance**

Always the hidden data in the background is the issue of performance. Building a transparent, flexible, reliable distributed system, more important lies in its performance. In particular, when running a particular application on a distributed system, it should not be appreciably worse than running the same application on a single processor. Unfortunately, achieving this is easier said than done.

2. **Distributed Computing goals**

Ans:

The four important goals that should be met for an efficient distributed system are as follows:

**1. Connecting Users and Resources:**

- The main goal of a distributed system is to make it easy for users to acces remote resourses and to share them with others in a controlled way.
- It is cheaper to le a printer be shared by several users than buying and maintaining printers for each user.
- Collaborating and exchanging information can be made easier by connecting users and resource.

**2. Transparency:**

- It is important for a distributed system to hide the location of its process and resource. A distributed system that can portray itself as a single system is said to be transparent.
- The various transparencies need to be considered are access, location, migration, relocation, replication, concurrency, failure and persistence.
- Aiming for distributed transparency should be considered along with performance issues.

**3. Openness:**

- Openness is an important goal of distributed system in which it offers services according to standard rules that describe the syntax and semantics of those services.
- Open distributed system must be flexible making it easy to configure and add new components without affecting existing components.
- An open distributed system must also be extensible.

**4. Scalable:**

- Scalability is one of the most important goals which are measured along three different dimensions.
- First, a system can be scalable with respect to its size which can add more user and resources to a system.
- Second, users and resources can be geographically apart.
- Third, it is possible to manage even if many administrative organizations are spanned.

3. **Types of distributed systems**
Ans:

The nodes in the distributed systems can be arranged in the form of client/server systems or peer to peer systems. Details about these are as follows −

### Client/Server Systems

In client server systems, the client requests a resource and the server provides that resource. A server may serve multiple clients at the same time while a client is in contact with only one server. Both the client and server usually communicate via a computer network and so they are a part of distributed systems.

### Peer to Peer Systems

The peer to peer systems contains nodes that are equal participants in data sharing. All the tasks are equally divided between all the nodes. The nodes interact with each other as required as share resources. This is done with the help of a network.

4. **Distributed System Models**
Ans:
Distributed System Models is as follows:

1. Architectural Models
2. Interaction Models
3. Fault Models

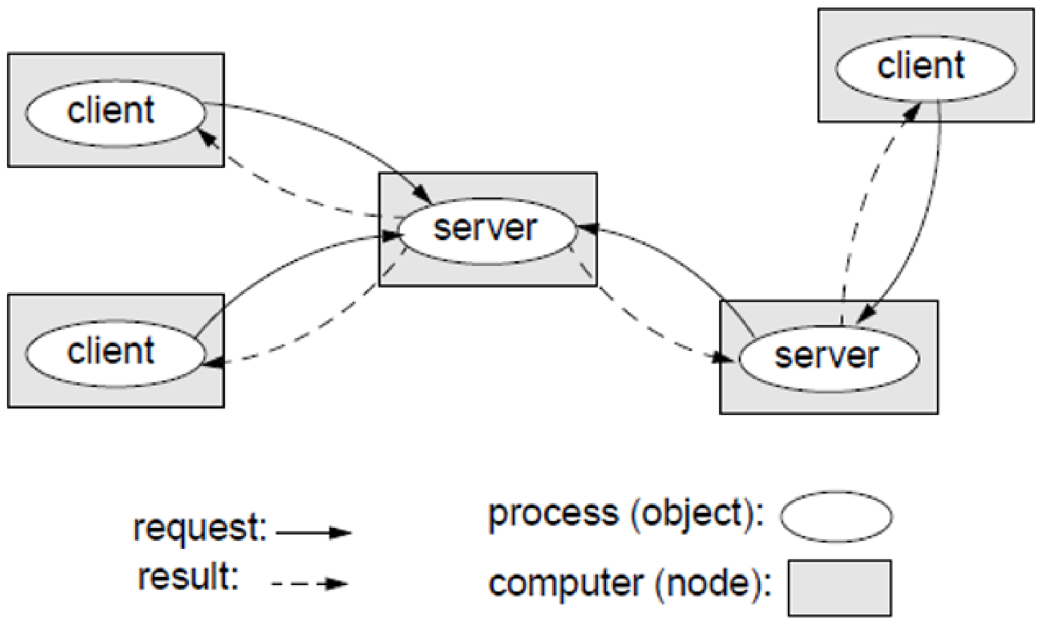
### 1. Architectural Models

Architectural model describes responsibilities distributed between system components and how are these components placed.

a)Client-server model

☞ The system is structured as a set of processes, called servers, that offer services to the users, called clients.

- The client-server model is usually based on a simple request/reply protocol, implemented with send/receive primitives or using remote procedure calls (RPC) or remote method invocation (RMI):
- The client sends a request (invocation) message to the server asking for some service;
- The server does the work and returns a result (e.g. the data requested) or an error code if the work could not be performed.

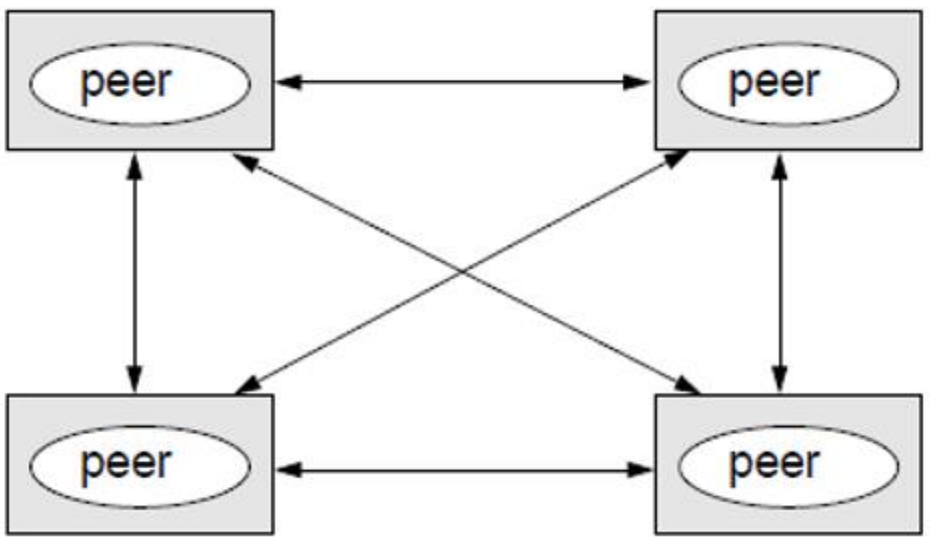


A server can itself request services from other servers; thus, in this new relation, the server itself acts like a client.

b)Peer-to-peer

☞ All processes (objects) play similar role.

- Processes (objects) interact without particular distinction between clients and servers.
- The pattern of communication depends on the particular application.
- A large number of data objects are shared; any individual computer holds only a small part of the application database.
- Processing and communication loads for access to objects are distributed across many computers and access links.
- This is the most general and flexible model.



- Peer-to-Peer tries to solve some of the above
- It distributes shared resources widely -> share computing and communication loads.

☞ Problems with peer-to-peer:

- High complexity due to
    - Cleverly place individual objects
    - retrieve the objects
    - maintain potentially large number of replicas.

**2.Interaction Model**

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

- Synchronous distributed systems

**Main features:**

- Lower and upper bounds on execution time of processes can be set.
- Transmitted messages are received within a known bounded time.
- Drift rates between local clocks have a known bound.

**Important consequences:**

1. In a synchronous distributed system there is a notion of global physical time (with a known relative precision depending on the drift rate).
2. Only synchronous distributed systems have a predictable behavior in terms of timing. Only such systems can be used for hard real-time applications.
3. In a synchronous distributed system it is possible and safe to use timeouts in order to detect failures of a process or communication link.

☞ It is difficult and costly to implement synchronous distributed systems.

- Asynchronous distributed systems

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

**Important consequences:**

1. In an asynchronous distributed system there is no global physical time. Reasoning can be only in terms of logical time (see lecture on time and state).
2. Asynchronous distributed systems are unpredictable in terms of timing.
3. No timeouts can be used.

☞ Asynchronous systems are widely and successfully used in practice.

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

However, additional measures have to be applied in order to avoid duplicated messages, duplicated execution of operations, etc.

**3. Fault Models**

☞ Failures can occur both in processes and communication channels. The reason can be both software and hardware faults.

☞ Fault models are needed in order to build systems with predictable behavior in case of faults (systems which are fault tolerant).

☞ such a system will function according to the predictions, only as long as the real faults behave as defined by the "fault model".

5. **Hardware concepts**

Ans:

All distributed systems consist of multiple CPUs. There are several different ways the hardware can be arranged. The important thing related to hardware is that how they are interconnected and how they communicate with each other. It is important to take a deep look at distributed system hardware, in particular, how the machines are connected together and how they interact.

Many classification schemes for multiple CPU computer systems have been proposed over the years, but none of them have really implemented. Still, the most commonly used taxonomy is Flynn's (1972), but it was in basic stage. In this scheme, Flynn took only two things to consider i.e. the **number of instruction streams** and the **number of data streams**.

**Single Instruction, Single Data Stream (SISD)**

A computer with a single instruction stream and a single data stream is called **SISD**. All traditional uni-processor computers (i.e., those having only one CPU) fall under this category, from personal computers to large mainframes. SISD flow concept is given in the figure below.

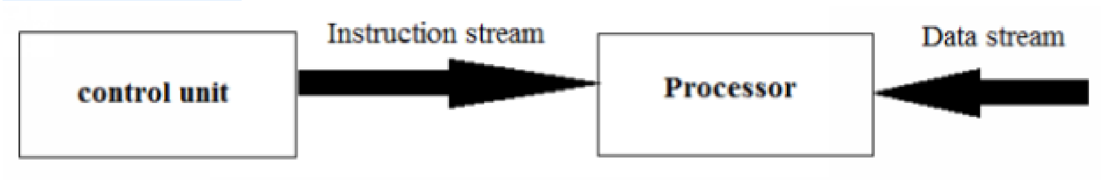


Fig: SISD Flow structure

**Single Instruction, Multiple Data Stream (SIMD)**

The next category is **SIMD**, i.e. single instruction stream, multiple data stream. This type uses an array of processors with only one instruction unit that fetches an instruction, and multiple data units which work in parallel. These machines are used where there need to apply the same instruction for multiple data example, adding up all the elements of 64 independent vectors. Some supercomputers are SIMD. Figure below shows the SIMD flow structure.

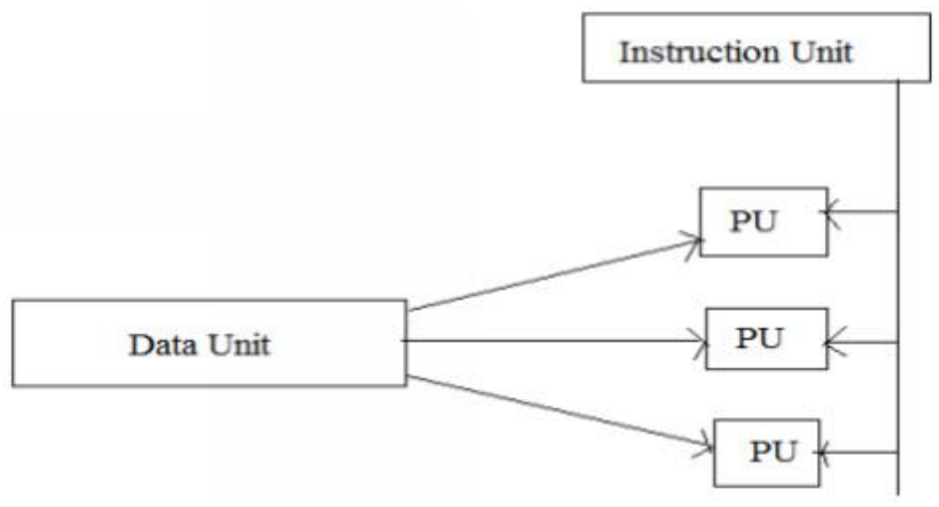


Fig: SIMD Flow structure

**Multiple Instruction, Single Data Stream (MISD)**

The next category is **MISD** i.e. multiple instruction streams, single data stream. This structure was worked when there are multiple different instructions to operate on the same type of data. In general MISD architecture is not use more in practical.

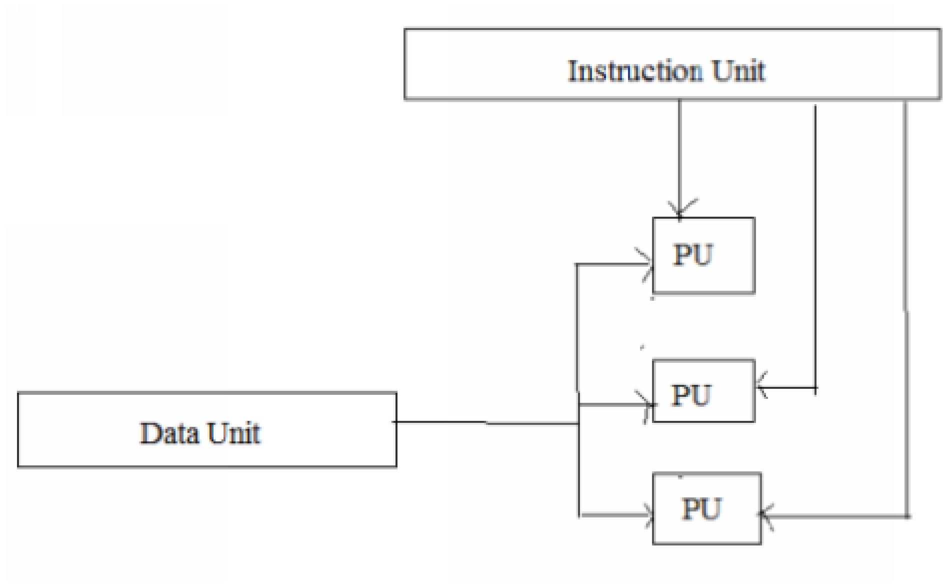


Fig: MISD Flow structure

**Multiple Instruction, Multiple Data Stream (MIMD)**

The next category is MIMD, which has multiple instructions performances on multiple data units. This means a group of independent computers; each has its own program counter, program, and data. All distributed systems are MIMD, so this classification system is not more useful for simple purposes.

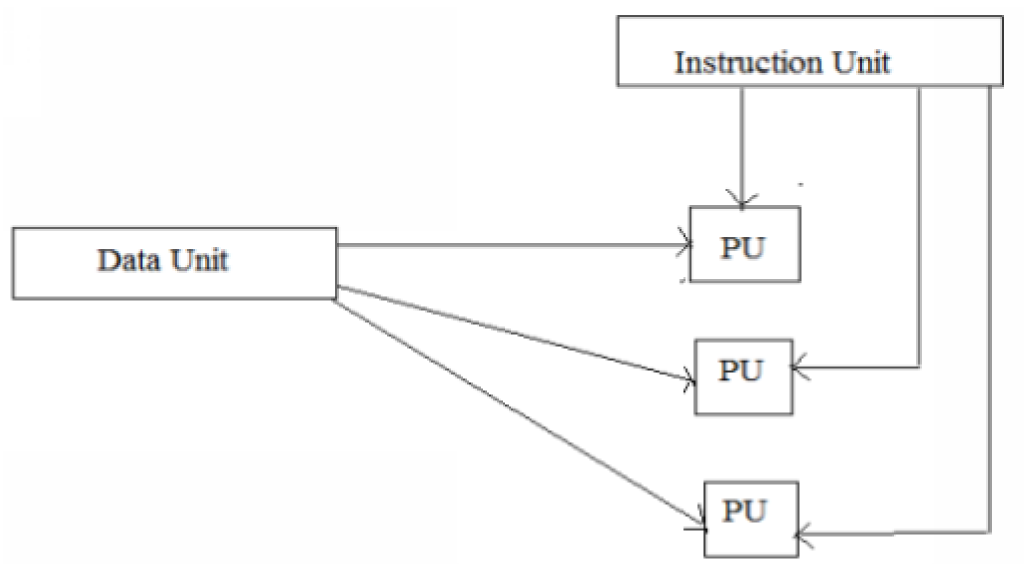


Fig: MIMD Flow structure

This was Flynn's classification. For advance study, let's divide all MIMD computers into two groups: The computers that have a shared memory, usually called **multi-processors**, and those that have their own local memory, sometimes called **multi-computer**.

The main concept of a multiprocessor is, there is a single virtual address space that is shared by all CPUs. If any CPU writes, for example, if one CPU writes value 44 to address 1000, then any other CPU subsequently reading from its location 1000 will get the value 44(but they can't write on same address location simultaneously). All the machines share the same memory.

On the other hand in a multi-computer, every machine has its own private memory i.e. distributed memory. If one CPU writes the value 44 to address 1000, when another CPU reads address 1000 it will get whatever value was there before. After writing 44 will not affect its memory at all. A simple example of a multi-computer is a collection of personal computers connected by a network.

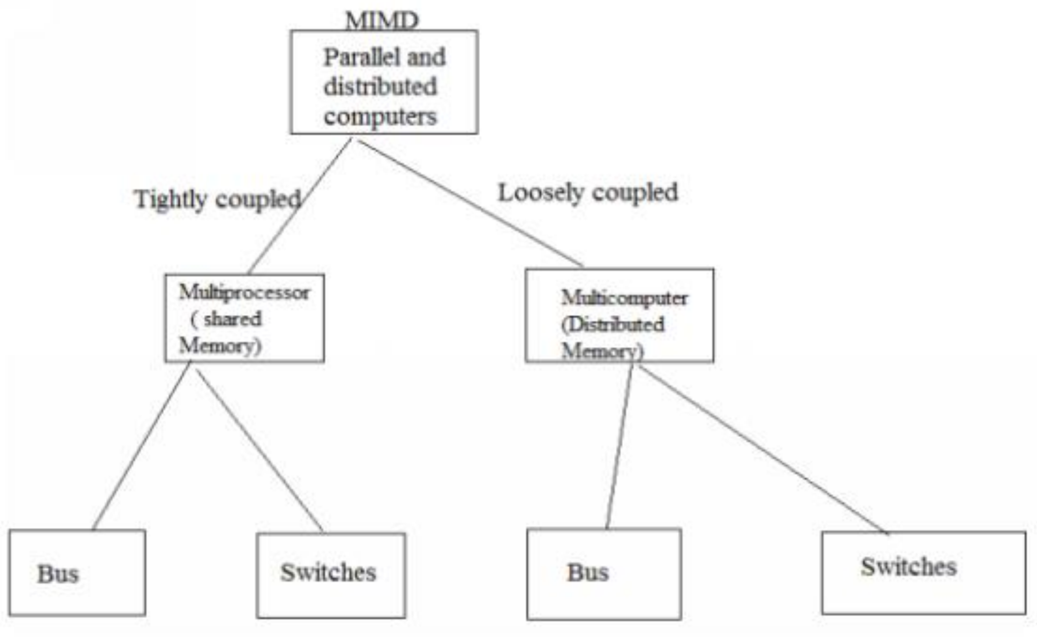

MIMD
Parallel and distributed computers
Tightly coupled
Loosely coupled
Multiprocessor ( shared Memory)
Multicomputer (Distributed Memory)
Bus
Switches
Bus
Switches

Fig: Taxonomy of parallel and distributed computer systems

In the above figure, the two categories described can be based on the architecture of the interconnection network. There are two categories at the bottom of the figure, **bus and switched**.

In the bus, there is a single network, backplane, bus, cable, or another medium that connects all the machines. In a cable television system, the cable company runs a wire down the street, and other subscribers have taps running to it from their television sets.

Switched systems do not have a single backbone like cable television. But, there are individual wires from machine to machine, having different wiring patterns in use. Messages move through the wires, and the final switching decision is made at each routing stage to route the message to an outgoing destination. The public telephone system for the whole world is organized in this way.

The next uppermost dimensions in our taxonomy are that in some systems the machines are **tightly coupled**, and in others, they are **loosely coupled**. In a tightly-coupled system, delay is short for message passing, and also the data rate is high; that means, the number of bits per second that can be transferred is large. In the loosely-coupled system, it is just the opposite of it. The inter-machine message delay is large and the data rate is low.

For example, two CPU chips on the same printed circuit board and connected by wires etched onto the board are likely to be tightly coupled, whereas two computers connected by a 2400 bit/sec modem over the telephone system are certain to be loosely coupled. Tightly-coupled systems used in more applications as parallel systems (working on a single problem) and loosely-coupled ones tend to be used as distributed systems (working on many unrelated problems), But this is not always true. One popular example is a project in which hundreds of computers all over the world worked together for factor a huge number (about 100 digits). Each computer was assigned a different range of divisors to try, and they all worked on the problem in their given time and also reporting the results back by email when they finished the task.

On the whole, multiprocessors tend to be more tightly coupled than multi-computers, because they can exchange data at memory speeds, but some fiber optic based multi-computer can also work at memory speeds.

## 6. Software Concept.

Ans:

The software of the distributed system is nothing but selection of different operating system platforms.

The operating system is the interaction between user and the hardware.

There are three largely used operating system types:

a) Distributed operating system

b) Network operating system

c) Middleware operating system

**Distributed operating system:**

It is different from multiprocessor and multicomputer hardware.

Multiprocessor- uses different system services to manage resources connected in a system and use system calls to communicate with the processor.

Multicomputer- the distributed Operating system uses a separate uniprocessor OS on each computer for communicating between different computers.

In distributed OS, a common set of services is shared among multiple processors in such a way that they are meant to execute a distributed application effectively and also provide services to separate independent computers connected in a network as shown in fig below

It communicates with all the computer using message passing interface(MPI).

It follows the tightly coupled architecture pattern.

It uses Data structure like queue to manages the messages and avoid message loss between sender and receiver computer.

Eg Automated banking system, railway reservation system etc.

**Disadvantages:**

- It has a problem of scalability as it supports only limited number of independent computers with shared resources.
- There is need to define message passing semantics prior to the execution of messages.

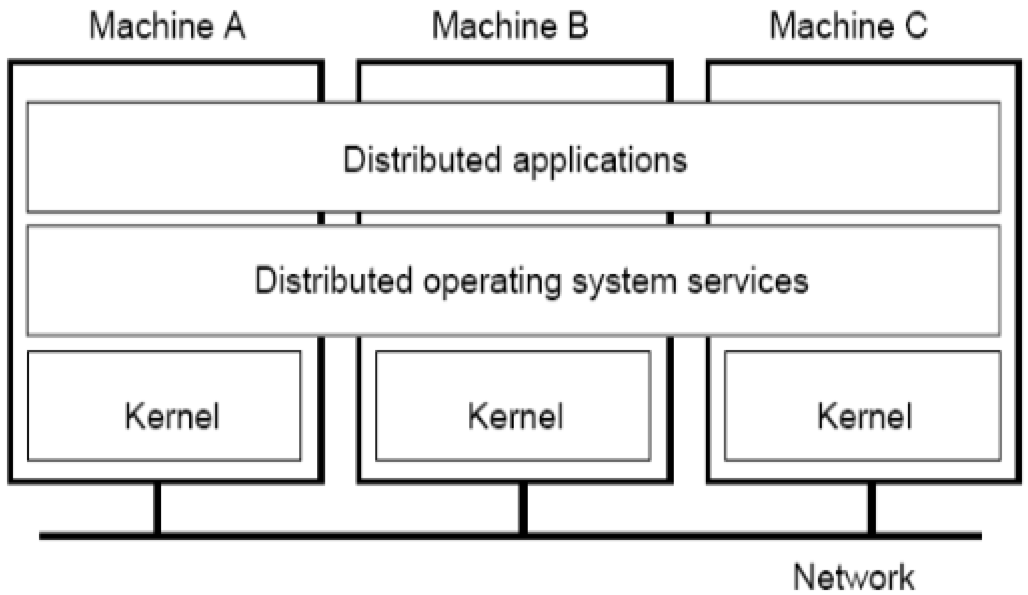


Fig. General structure of a multicomputer operating system

**Network operating system:**

It is specifically designed for hetrogeneous multicomputer system, where multiple hardware and network platforms are supported.

It has multiple operating system running on different hardware platforms connected in network.

It provides to each computer connected in network.

It follows the loosely coupled architecture pattern which allow user to use services provided by the local machine itself as shown in fig below.

Eg Remote login where user workstation is used to log in to the remoter server and execute remote commands over the network.

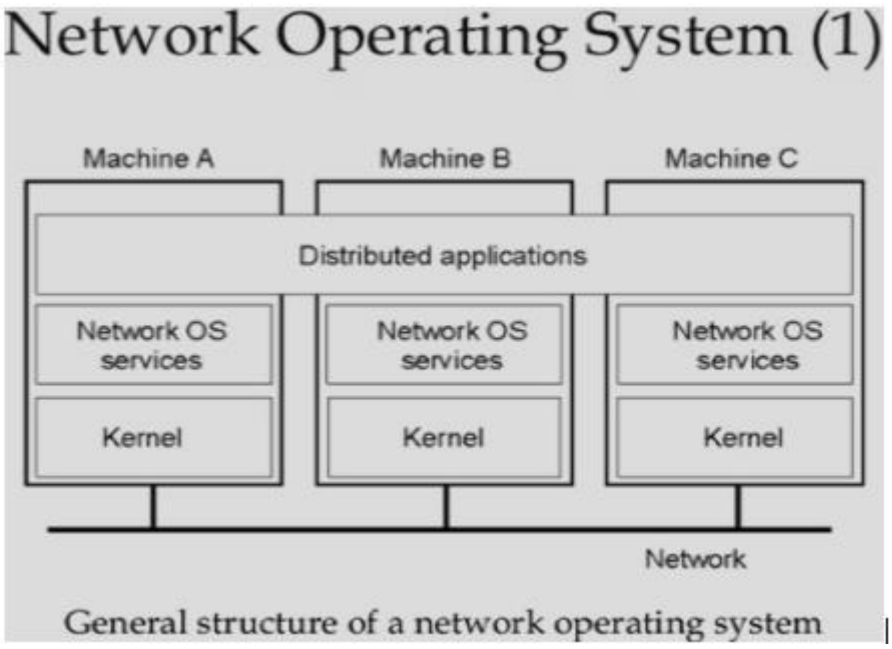
Eg Centralized file storage system.

**Advantage:**

It has scalability feature, where large number of resources and users are supported.

**Disadvantage:**

It fails to provide a single coherent view.



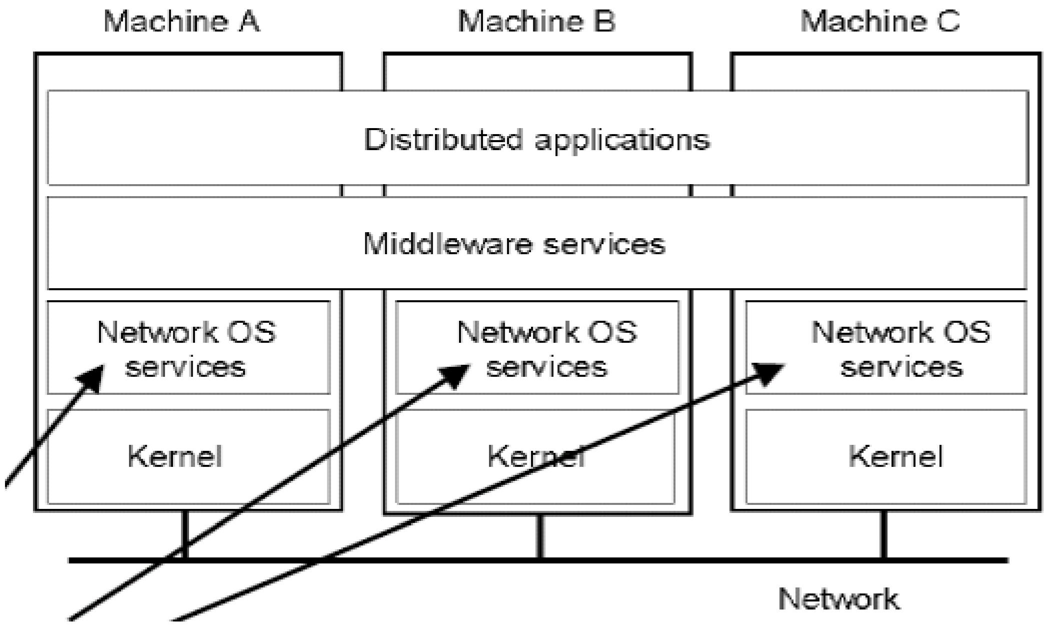
**Middle ware operating system:**

As distributed operating system has lack of scalability and network operating system fails to provide a single coherent view, therefore a new layer is formed between the distributed and network operating system is called the middleware operating system.

It has a common set of services is provided for the local applications and independent set of services for the remote applications.

It support heterogeneity that is it supports multiple languages and operating system where user gets freedom to write the application using the any of the supported language under any platform.

It provide the services such as locating the objects or interfaces by their names, finding the location of objects, maintaining the quality of services, handling the protocol information, synchronization, concurrency and security of the objects etc.



Fig(a) Middleware operating system

7. **Models of Middleware**

Ans:

The first middle ware model has started with the distributed file system, where the files were stored and distributed over the network.

Models of middle ware are as follows:

**Remote Procedure call:**

It is one of the succesful middle ware models used in modern distributed applications for communication.

It uses local call to call a procedure resides on the remote machine to get the result.

Hidden communication is done between client and server.

For eg If a user wants to get the sum of two numbers stored on remote server by using local method call, the user calls method with parameters, in turn server receives a RPC call from the client and returns the appropriate result to the client.

Therefore, though the method was executed remotely it appears like a local to the called machine.

This is a synchronous technology where both the client and server should be present during the communication.

**Message oriented Middleware(MOM):**

It is another model used to send and receive the communication messages between clients and servers.

It uses data structures like queue to store and retrieve messages.

When the client is sending the messages faster than the receiver receiving it or the client is sending the message when the receiver is not available. So it uses queuing mechanism between the client and server to avoid the message been misplaced.

It is asynchronous mechanism where messages can be sent even though the receiver is not available

For eg Email system

**Distributed Object Technology**

The distributed object technology has changed the scope of middleware technologies to one step up where objects are distributed to the remote server to facilitate the client.

Eg RMI and CORBA

The distributed object mechanism hides the communication interfaces and their details to provide access to the remote object efficiently.

**Remote Method Invocation**

In this objects are distributed and located by using the RMI registry.

The client can access remote objects by using the interfaces.

**Disadvantage**

It didnt support the concept of hetrogeneity and is compatible with java platform only. Common object request broker architecture(CORBA)-

It is one of the most popular distributed object technologies where objects can be accessible from remote location through ORB.

Server and client communicate with each other through object request broker bus.

To map the semantics of objects and fetch the appropriate object an interface definition language is used.

It is evolved with service based middle ware where service models are used.

In service model, the services are published by the service providers and consumed by the service consumer.

eg Service oriented Architecture (SOA)

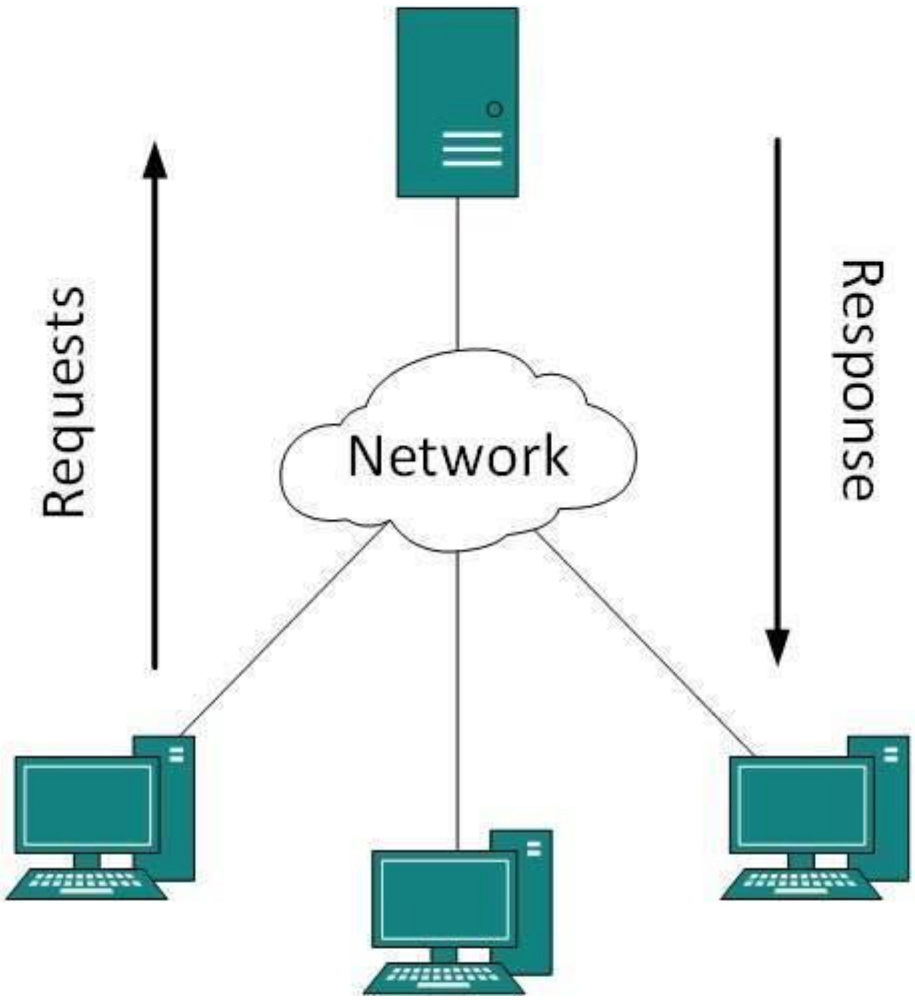
8. **Services offered by middleware**
Ans: https://youtu.be/8y-XNoaB790

9. **Client Server model.**

Ans:

- **Client-Server:** One remote process acts as a Client and requests some resource from another application process acting as Server.

In client-server model, any process can act as Server or Client. It is not the type of machine, size of the machine, or its computing power which makes it server; it is the ability of serving request that makes a machine a server.



A system can act as Server and Client simultaneously. That is, one process is acting as Server and another is acting as a client. This may also happen that both client and server processes reside on the same machine.

## Communication

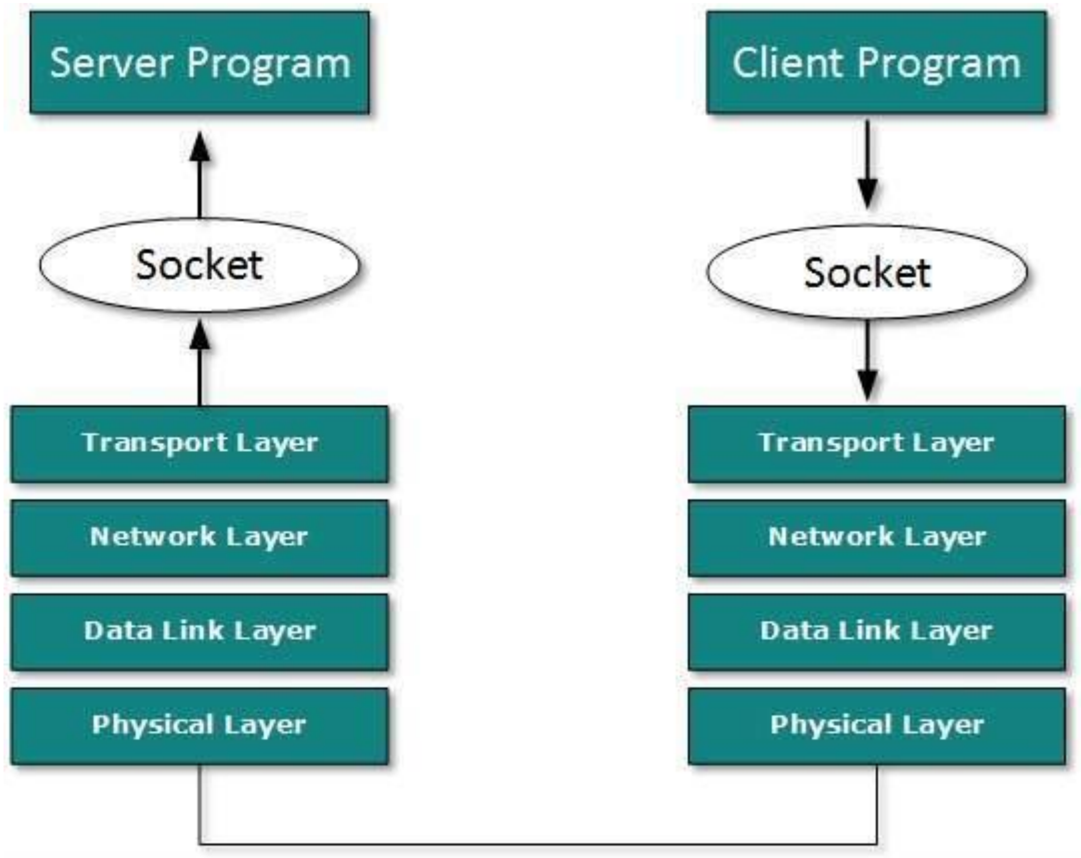
Two processes in client-server model can interact in various ways:

- Sockets
- Remote Procedure Calls (RPC)

### Sockets

In this paradigm, the process acting as Server opens a socket using a well-known (or known by client) port and waits until some client request comes.

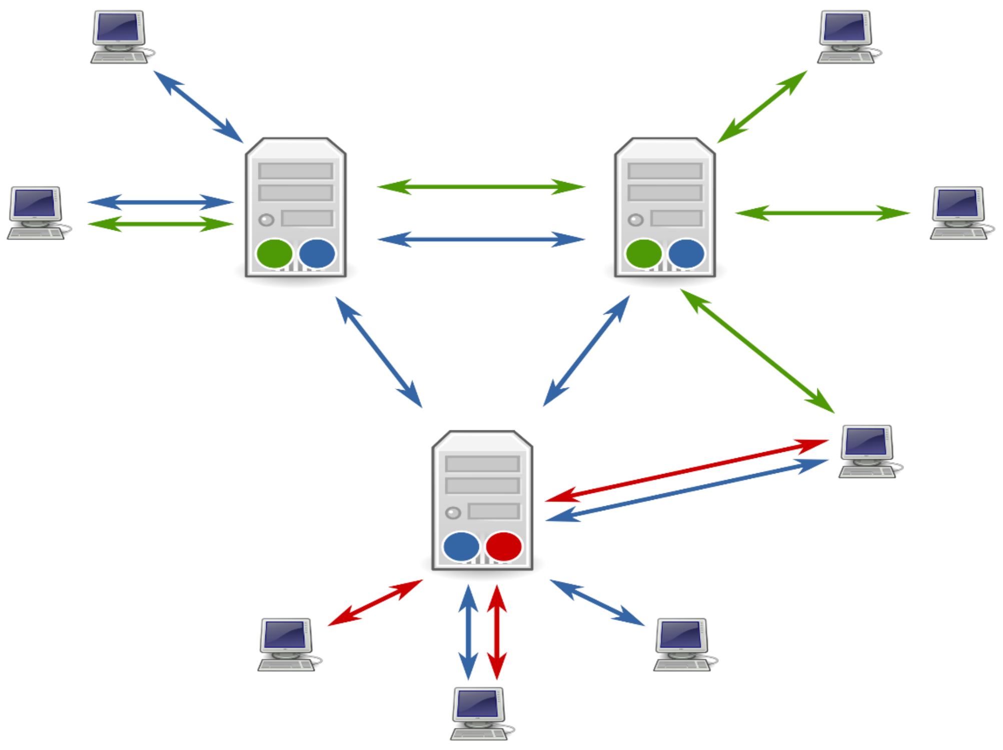
The second process acting as a Client also opens a socket but instead of waiting for an incoming request, the client processes ‘requests first’.



When the request is reached to server, it is served. It can either be an information sharing or resource request.

10. **OSI Layered Protocols**
Ans:

11. **What is IPC**
Ans:
**Interprocess Communication** is a process of exchanging the data between two or more independent process in a distributed environment is called as Interprocess communication. Interprocess communication on the internet provides both Datagram and stream communication.
**Examples Of Interprocess Communication:**
1. N number of applications can communicate with the X server through network protocols.
2. Servers like Apache spawn child processes to handle requests.
3. Pipes are a form of IPC: grep foo file | sort



It has two functions:

1. **Synchronization:**
   Exchange of data is done synchronously which means it has a single clock pulse.
2. **Message Passing:**
   When processes wish to exchange information. Message passing takes several forms such as: pipes, FIFO, Shared Memory, and Message Queues.

**Characteristics Of Inter-process Communication:**
There are mainly seven characteristics of inter-process communication in a distributed environment/system.

1. **Synchronous System Calls:**
   In the synchronous system calls both sender and receiver use blocking system calls to transmit the data which means the sender will wait until the acknowledgment is received from the receiver and receiver waits until the message arrives.
2. **Asynchronous System Calls:**
   In the asynchronous system calls, both sender and receiver use non-blocking system calls to transmit the data which means the sender doesn’t wait from the receiver acknowledgment.
3. **Message Destination:**
   A local port is a message destination within a computer, specified as an integer. Aport has exactly one receiver but many senders. Processes may use multiple ports from which to receive messages. Any process that knows the number of a port can send the message to it.

4. **Reliability:**
   It is defined as validity and integrity.
5. **Integrity:**
   Messages must arrive without corruption and duplication to the destination.
6. **Validity:**
   Point to point message services are defined as reliable, If the messages are guaranteed to be delivered without being lost is called validity.
7. **Ordering:**
   It is the process of delivering messages to the receiver in a particular order. Some applications require messages to be delivered in the sender order i.e the order in which they were transmitted by the sender.

12. **Explain concept of MPI**
Ans:
Message Passing Interface (MPI) is a standardized and portable **message-passing** system developed for distributed and parallel computing. MPI provides parallel hardware vendors with a clearly defined base set of routines that can be efficiently implemented. As a result, hardware vendors can build upon this collection of standard low-level routines to create higher-level routines for the distributed-memory communication environment supplied with their parallel machines.
MPI gives user the flexibility of calling set of routines from **C, C++, Fortran, C#, Java or Python.** The advantages of MPI over older message passing libraries are portability (because MPI has been implemented for almost every distributed memory architecture) and speed (because each implementation is in principle optimized for the hardware on which it runs)
The advantages of MPI over other message passing framework is portability and speed. It has been implemented for almost every distributed memory architecture and each implementation is in principle optimized for the hardware on which it runs.

13. **Process of Remote Procedure Call (RPC)**
Ans:

A remote procedure call is an interprocess communication technique that is used for client-server based applications. It is also known as a subroutine call or a function call.

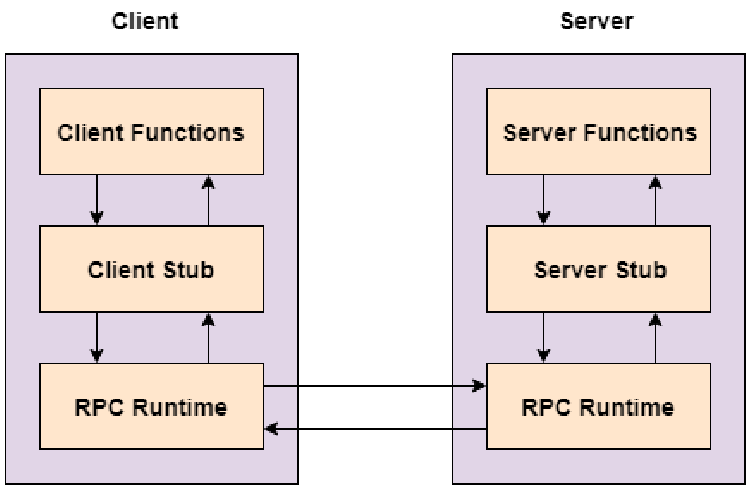
A client has a request message that the RPC translates and sends to the server. This request may be a procedure or a function call to a remote server. When the server receives the request, it sends the required response back to the client. The client is blocked while the server is processing the call and only resumed execution after the server is finished.

The sequence of events in a remote procedure call are given as follows −

- The client stub is called by the client.
- The client stub makes a system call to send the message to the server and puts the parameters in the message.

- The message is sent from the client to the server by the client's operating system.
- The message is passed to the server stub by the server operating system.
- The parameters are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.

A diagram that demonstrates this is as follows −



14. **Process of RMI**
Ans:

## RMI (Remote Method Invocation)

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

### Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

#### stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

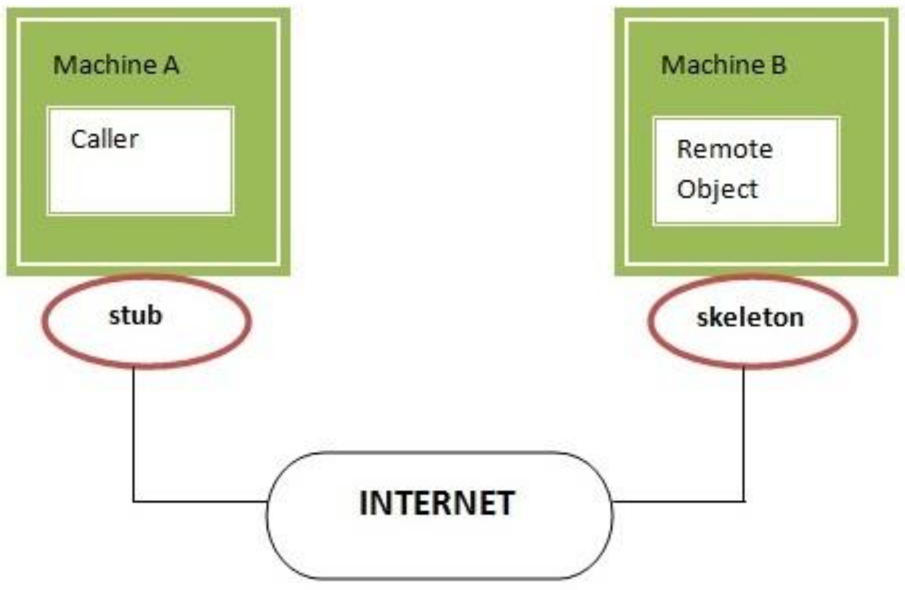
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

## skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.



15. **Message Oriented Communication**
Ans: https://youtu.be/aOtdqb3YrF4

16. **Stream Oriented Communication**
Ans:

## Stream-oriented communication

1. Support for continuous media.
2. Streams in distributed systems
3. Stream management

## Continuous media

**Observation**

All communication facilities discussed so far are essentially based on a discrete, that is time-independent exchange of information

**Continuous media**

Characterized by the fact that values are time dependent:

a. Audio

b. Video

c. Animations

d. Sensor data (temperature, pressure, etc.)

**Transmission modes**

Different timing guarantees with respect to data transfer:

**Asynchronous:** no restrictions with respect to when data is to be delivered

**Synchronous:** define a maximum end-to-end delay for individual data packets

**Isochronous:** define a maximum end-to-end delay and maximum delay variance (jitter is bounded)

## Stream

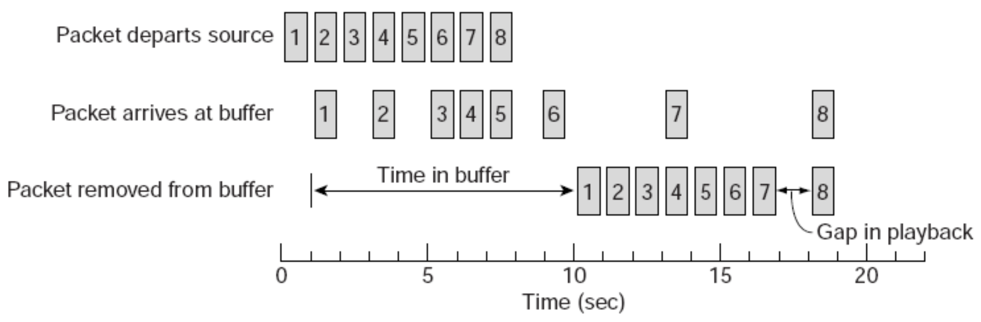
**Definition**

A (continuous) data stream is a connection-oriented communication facility that supports isochronous data transmission.

Some common stream characteristics

1. Streams are unidirectional
2. There is generally a single source, and one or more sinks
3. Often, either the sink and/or source is a wrapper around hardware (e.g., camera, CD device, TV monitor)

4. Simple stream: a single flow of data, e.g., audio or video
5. Complex stream: multiple data flows, e.g., stereo audio or combination audio/video



17. **Group Communication**
Ans:
• A group is a collection of processes that act together in some system or user specified way.

• The key property that all groups have that when a message is sent to the group; itself all the members of the group receive it.

• It is a form of one to many communication.

**• One – to - many**

a. Group management

b. Group Addressing

c. Message delivery to receiving process

d. Buffered and unbuffered multicast

e. Flexible reliability in multicast communication

f. Atomic multicast

g. Group communication primitives

• Single sender and multiple receiver is also known as multicast communication

• Broadcast is a special case of multicast communication.

**a. Group Management**

• In case of group communication the communicating processes forms a group.

• Such a group may of either of 2 types.

• Closed group is the one, in which only member can send message, outside process cannot sned message to the group as whole but can send to single member of a group.

• Open group is one in which any process in the system can send the message to group as a whole.

• Message passing provides flexibility and create groups dynamically and to allow process join or leave group dynamically.

• Centralized group server is used for this purpose but suffers from poor reliability and scalability.

**b. Group Addressing**

• Two level naming scheme is used for group addressing.

• Multicast address is a special n/w address, packet is delivered automatically here to all m/cs listening to address.

• It broadcast address the all m/cs has to aheck if packet is intended for it or else simply discard so broadcast is less efficient.

• If network doesn't support any addressing among two then one to one communication is used.

• First two methods send single packet but one to one sends many, creating much traffic.

**c. Message delivery to receiving process**

• User application uses high level group names in program.

• Centralized group server maintains a mapping of high level group names to their low level names, when sender sends message to the group. With high level name, kernel of server m/c asks the group servers low level name and list of process identifiers.

• When packet reaches m/c kernel, that m/c extract list of process IDs and forwards message to those processes belonging to its own m/c.

• If none of ID's is matching packet is discarded.

**d. Buffered and unbuffered multicast**

• Send to all semantics:- A copy of message is sent to each process of multicast group and message is buffered until it is accepted by process.

• Bulletin board semantics:- Message to be multicast is addressed to channel instead of each process.

```
♣     Channel plays as a bulletin board.
```

**e. Flexible Reliability in Multicast communication.**

• In multicast mechanism there is flexibility for user definable reliability.

• Degree of reliability is expressed in 4 forms.

```
a) O – Reliable: - No response is expected at all.


b) I – Reliable: - Sender expects response from any one of the receiver.


c) mount of n reliable :- Sender expects response from m of n receiver.


d) All reliable : - Sender expects response from all.
```

**f. Atomic Multicast**

• All or nothing property all reliable form requires atomic multicast facility.

• Message passing system should support both atomic and non- atomic multicast facility.

• It should provide flexibility to sender to specify whether atomicity is required or not.

• It is difficult to implement atomic multicast if sender or receiver fails.

• Solution to this is fault tolerated atomic multicast protocol.

• In this protocol, each message has message identifier field to distinguish message and one field to indicate data message in atomic multicast message.

**g. Group communication primitives**

• In one to one and one to many send or has to specify destination address and pointer to data so some send primitives can be used for both.

• But some systems use send group primitives because

```
1) It simplifies design and implementation


2) It provides greater flexibilities.
```

**Many – to – one**

• Multiple sender single receiver

• Single receiver may be selective or non-selective

• Selective receiver specifies unique sender message exchange takes place only if the sender sends the message.

• Non-selective receiver specifies set of sender if any from that sends message then message exchange takes place.

• No determinism issue need to be handled here.

• Rest all factors are same as for one – to – many communication.

**Many –to – many**

• One to many, many to one, all issues are induced in this.

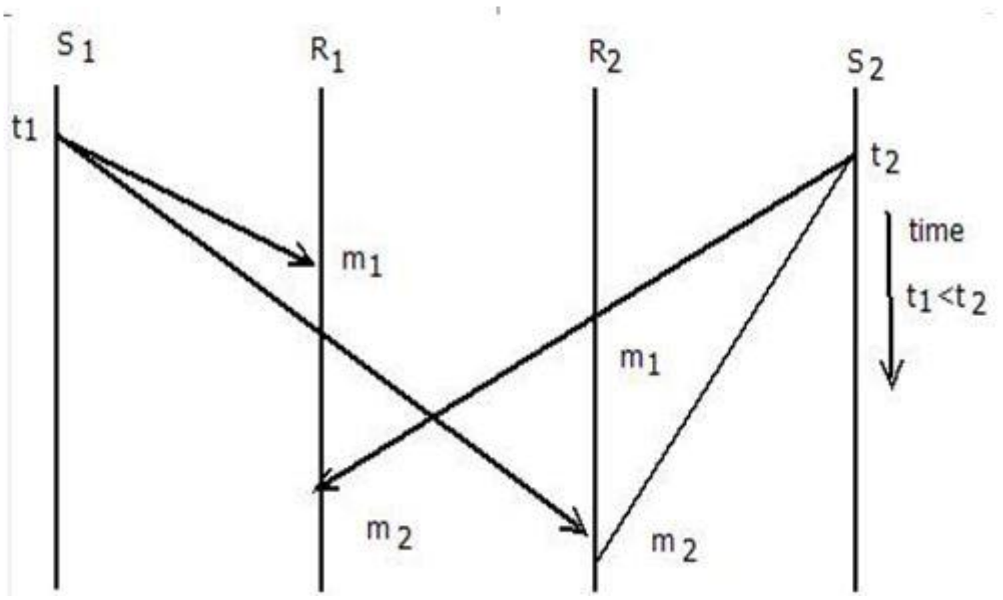
• Ordered message delivery/ event ordering.

```
1) Absolute ordering


2) Consistent ordering


3) Casual
```

**1) Absolute Ordering**



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

• Used to implement is to use global timestamps as message id.

• System is assumed to have clock synchronization and when sender message timestamps is taken as id of message and embedded in message.

• Kernel of receiver m/c, saves all incoming message in separate queue.

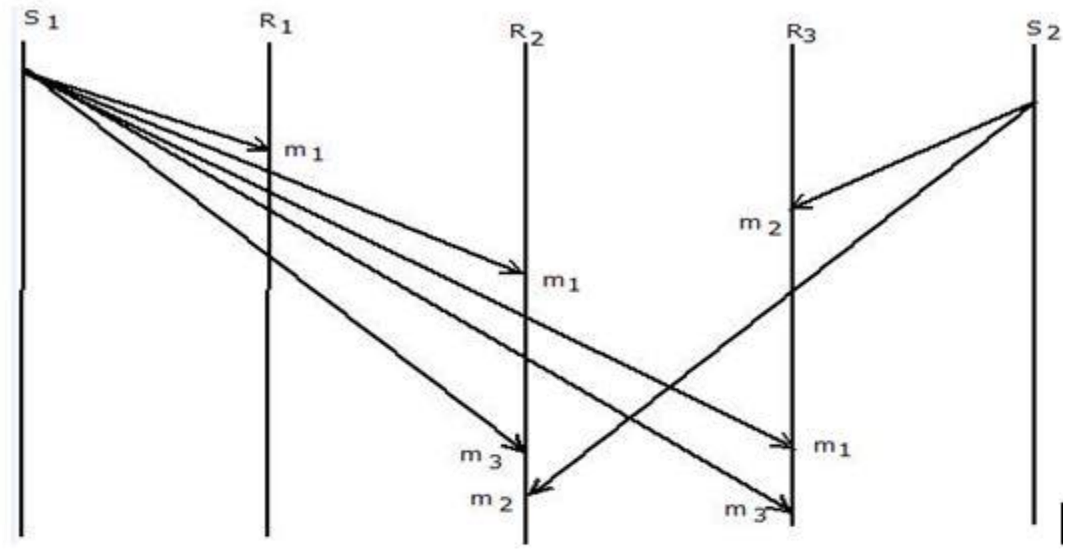
• A sliding window is used to periodically deliver message from receiver i.e fixed time interval is selected as window size.

• Window size and properly chosen by considering maximum time for message to kernel from one m/c to other in n/w.

```
 2) Casual Ordering
```

• For some application weaker semantics acceptable.

• Such semantic is called casual ordering.



• This semantics ensures that is the sending one message is casually related to event of sending another message, two messages are delivered to all receiver in correct order, otherwise these may be delivered in any order.

• Two message sending events are said to be casually related if they are correlated by happened before relation.

• Two message sending events are casually related if there is any possibilities of second one being influenced in a way by first one.

• For 8 implementing casual ordering semantics is the CACAST protocol:

```
1.  Each member process of group maintaining a vector of n components, whe
re n is the total
```

no. of members in group.

```
      a.    Each member is assigned a sequence no. from 0 to n, and the it
h component of vector corresponds to member with sequence no. i


2.  To send message, a process increments he value of its own component ve
ctor and sends vector as a part of message.


3.  When message arrives at receiver size, it is buffered by runtime syste
m.


   a.   Runtime system, test two conditions given below to user process or
 must be delayed to ensure casual ordering.


  b.    The two conditions are:
       S [i] = R [i] +1     and         s [i] ≤ R [i] for all j ≠ i.


 c. 1st condition ensures that receiver has not missed any message from se
nder and 2nd condition   ensures that sender has not get received any mess
age that receiver has not get received.


  d.    This test is needed to make sure that sender’s message is not casu
ally related to message   missed by receiver.


  e.    If message passes these tests runtime system delivers it to user p
rocess otherwise message is buffered.
```

18. **List Token based algorithms. Explain any one**
Ans:
**(i)** Singhal’s Heuristic Algorithm
**(ii)** Raymonds Tree Based Algorithm
**(iii)** Suzuki-Kasami Algorithm

**Suzuki–Kasami algorithm** is a token-based algorithm for achieving mutual exclusion in distributed systems.This is modification of Ricart–Agrawala algorithm, a permission based (Non-token based) algorithm which uses **REQUEST** and **REPLY** messages to ensure mutual exclusion.
In token-based algorithms, A site is allowed to enter its critical section if it possesses the unique token. Non-token based algorithms uses timestamp to order requests for the critical section where as sequence number is used in token based algorithms.

Each requests for critical section contains a sequence number. This sequence number is used to distinguish old and current requests.

**Data structure and Notations:**

- An array of integers **RN[1…N]**
  A site $S_i$ keeps **RN$_i$[1…N]**, where **RN$_i$[j]** is the largest sequence number received so far through **REQUEST** message from site $S_i$.
- An array of integer **LN[1…N]**
  This array is used by the token.**LN[J]** is the sequence number of the request that is recently executed by site $S_j$.
- A queue **Q**
  This data structure is used by the token to keep record of ID of sites waiting for the token

**Algorithm:**
- **To enter Critical section:**
  - When a site $S_i$ wants to enter the critical section and it does not have the token then it increments its sequence number **RN$_i$[i]** and

sends a request message **REQUEST(i, sn)** to all other sites in order to request the token.
Here **sn** is update value of **$RN_i$[i]**

- When a site $S_j$ receives the request message **REQUEST(i, sn)** from site $S_i$, it sets **$RN_j$[i]** to maximum of **$RN_j$[i]** and **sn** i.e **$RN_j$[i]** = max(**$RN_j$[i]**, **sn**).
- After updating **$RN_j$[i]**, Site $S_j$ sends the token to site $S_i$ if it has token and **$RN_j$[i]** = **LN[i]** + 1

- **To execute the critical section:**
    - Site $S_i$ executes the critical section if it has acquired the token.
- **To release the critical section:**
  After finishing the execution Site $S_i$ exits the critical section and does following:
    - sets **LN[i]** = **$RN_i$[i]** to indicate that its critical section request **$RN_i$[i]** has been executed
    - For every site $S_j$, whose ID is not prsent in the token queue **Q**, it appends its ID to **Q** if **$RN_i$[j]** = **LN[j]** + 1 to indicate that site $S_j$ has an outstanding request.
    - After above updation, if the Queue **Q** is non-empty, it pops a site ID from the **Q** and sends the token to site indicated by popped ID.
    - If the queue **Q** is empty, it keeps the token

**Message Complexity:**
The algorithm requires 0 message invocation if the site already holds the idle token at the time of critical section request or maximum of N message per critical section execution. This N messages involves

- (N – 1) request messages
- 1 reply message

**Drawbacks of Suzuki–Kasami Algorithm:**

- **Non-symmetric Algorithm:** A site retains the token even if it does not have requested for critical section. According to definition of symmetric algorithm
  "No site possesses the right to access its critical section when it has not been requested."

**Performance:**

- Synchronization delay is 0 and no message is needed if the site holds the idle token at the time of its request.
- In case site does not holds the idle token, the maximum synchronization delay is equal to maximum message transmission time and a maximum of N message is required per critical section invocation.

19. **List non token based algorithms. Explain anyone**
Ans:
**(i)** Lamport's Algorithm
**(ii)** Ricart-Agarwala Algorithm
**(iii)** Maekawa's Algorithm

**Lamport's Distributed Mutual Exclusion Algorithm** is a permission based algorithm proposed by Lamport as an illustration of his synchronization scheme for distributed systems.

In permission based timestamp is used to order critical section requests and to resolve any conflict between requests.
In Lamport's Algorithm critical section requests are executed in the increasing order of timestamps i.e a request with smaller timestamp will be given permission to execute critical section first than a request with larger timestamp.

In this algorithm:

- Three type of messages ( **REQUEST**, **REPLY** and **RELEASE**) are used and communication channels are assumed to follow FIFO order.
- A site send a **REQUEST** message to all other site to get their permission to enter critical section.
- A site send a **REPLY** message to requesting site to give its permission to enter the critical section.
- A site send a **RELEASE** message to all other site upon exiting the critical section.
- Every site $S_i$, keeps a queue to store critical section requests ordered by their timestamps.
  **request_queue**$_i$ denotes the queue of site $S_i$
- A timestamp is given to each critical section request using Lamport's logical clock.
- Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section request is always in the order of their timestamp.

**Algorithm:**

- **To enter Critical section:**
  - When a site $S_i$ wants to enter the critical section, it sends a request message **Request(ts$_i$, i)** to all other sites and places the request on **request_queue**$_i$. Here, $Ts_i$ denotes the timestamp of Site $S_i$
  - When a site $S_j$ receives the request message **REQUEST(ts$_i$, i)** from site $S_i$, it returns a timestamped REPLY message to site $S_i$ and places the request of site $S_i$ on **request_queue**$_j$

  .
- **To execute the critical section:**
  - A site $S_i$ can enter the critical section if it has received the message with timestamp larger than **(ts$_i$, i)** from all other sites and its own request is at the top of **request_queue**$_i$
- **To release the critical section:**
  - When a site $S_i$ exits the critical section, it removes its own request from the top of its request queue and sends a timestamped **RELEASE** message to all other sites
  - When a site $S_j$ receives the timestamped **RELEASE** message from site $S_i$, it removes the request of $S_i$ from its request queue

**Message Complexity:**
Lamport's Algorithm requires invocation of 3(N – 1) messages per critical section execution. These 3(N – 1) messages involves

- (N – 1) request messages
- (N – 1) reply messages
- (N – 1) release messages

**Drawbacks of Lamport's Algorithm:**

- **Unreliable approach:** failure of any one of the processes will halt the progress of entire system.
- **High message complexity:** Algorithm requires 3(N-1) messages per critical section invocation.

**Performance:**

- Synchronization delay is equal to maximum message transmission time
- It requires 3(N – 1) messages per CS execution.
- Algorithm can be optimized to 2(N – 1) messages by omitting the **REPLY** message in some situations.

20. **Need of Clock Synchronization**

Ans: Distributed System is a collection of computers connected via the high speed communication network. In the distributed system, the hardware and software components communicate and coordinate their actions by message passing. Each node in distributed systems can share their resources with other nodes. So, there is need of proper allocation of resources to preserve the state of resources and help coordinate between the several processes. To resolve such conflicts, synchronization is used. Synchronization in distributed systems is achieved via clocks.
The physical clocks are used to adjust the time of nodes.Each node in the system can share its local time with other nodes in the system. The time is set based on UTC (Universal Time Coordination). UTC is used as a reference time clock for the nodes in the system.

21. **What are  Logical Clocks**

Ans:

**Logical Clocks** refer to implementing a protocol on all machines within your distributed system, so that the machines are able to maintain consistent ordering of events within some virtual timespan. A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system. Distributed systems may have no physically synchronous global clock, so a logical clock allows global ordering on events from different processes in such systems.

**Example :**

If we go outside then we have made a full plan that at which place we have to go first, second and so on. We don't go to second place at first and then the first place. We always maintain the procedure or an organization that is planned before. In a similar way, we should do the operations on our PCs one by one in an organized way.
Suppose, we have more than 10 PCs in a distributed system and every PC is doing it's own work but then how we make them work together. There comes a solution to this i.e. LOGICAL CLOCK.

**Method-1:**
To order events across process, try to sync clocks in one approach.
This means that if one PC has a time 2:00 pm then every PC should have the same time which is quite not possible. Not every clock can sync at one time. Then we can't follow this method.

**Method-2:**
Another approach is to assign Timestamps to events.
Taking the example into consideration, this means if we assign the first place as 1, second place as 2, third place as 3 and so on. Then we always know that the first place will always come first and then so on. Similarly, If we give each PC their individual number than it will be organized in a way that 1st PC will complete its process first and then second and so on.

BUT, Timestamps will only work as long as they obey causality.

**What is causality ?**
Causality is fully based on HAPPEN BEFORE RELATIONSHIP.

- Taking single PC only if 2 events A and B are occurring one by one then TS(A) < TS(B). If A has timestamp of 1, then B should have timestamp more than 1, then only happen before relationship occurs.
- Taking 2 PCs and event A in P1 (PC.1) and event B in P2 (PC.2) then also the condition will be TS(A) < TS(B). Taking example- suppose you are sending message to someone at 2:00:00 pm, and the other person is receiving it at 2:00:02 pm.Then it's obvious that TS(sender) < TS(receiver).

**Properties Derived from Happen Before Relationship –**

- **Transitive Relation –**
  If, TS(A) <TS(B) and TS(B) <TS(C), then TS(A) < TS(C)
- **Causally Ordered Relation –**
  a->b, this means that a is occurring before b and if there is any changes in a it will surely reflect on b.
- **Concurrent Event –**
  This means that not every process occurs one by one, some processes are made to happen simultaneously i.e., A || B.

22. **Explain Election Algorithms**
Ans:

**Election Algorithms:**
Election algorithms choose a process from group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor. Election algorithm basically determines where a new copy of coordinator should be restarted.
Election algorithm assumes that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has highest priority number.Then this number is send to every active process in the distributed system.

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

**1. The Bully Algorithm –**
This algorithm applies to system where every process can send a message to every other process in the system.

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

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

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

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

23. **Advantage of Mutual Exclusion**
Ans:
Mutual exclusion is guaranteed as a process can enter its critical section only after getting permission from all processes, freedom from starvation since entry to critical section is scheduled according to the timestamp.

24. **Distributed Mutual Exclusion**
Ans:
**Mutual exclusion** is a concurrency control property which is introduced to prevent race conditions. It is the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time.
**Mutual exclusion in single computer system Vs. distributed system:**
In single computer system, memory and other resources are shared between different processes. The status of shared resources and the status of users is easily available in the shared memory so with the help of shared variable (For example: Semaphores) mutual exclusion problem can be easily solved.
In Distributed systems, we neither have shared memory nor a common physical clock and there for we can not solve mutual exclusion problem using shared variables. To eliminate the mutual exclusion problem in distributed system approach based on message passing is used.

A site in distributed system do not have complete information of state of the system due to lack of shared memory and a common physical clock.

25. **Requirements of Mutual Exclusion Algorithms & Performance measure**.
Ans:

**Requirements:**

- **No Deadlock:**
  Two or more site should not endlessly wait for any message that will never arrive.
- **No Starvation:**
  Every site who wants to execute critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other site are repeatedly executing critical section
- **Fairness:**
  Each site should get a fair chance to execute critical section. Any request to execute critical section must be executed in the order they are made i.e Critical section execution requests should be executed in the order of their arrival in the system.
- **Fault Tolerance:**
  In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

The performance of mutual exclusion algorithms is generally measured by the following four metrics:

1. Message complexity. It is number of messages that are required per critical section execution by a site.
2. Synchronization delay. After a site leaves the critical section, it is the time required and before the next site enters the critical section. Note that normally one or more sequential message exchanges may be required after a site exits the critical section and before next site can enter the critical section.
3. Response time. It is the time interval a request waits for its critical section execution to be over after its request messages have been sent out. Thus, response time does not include the time a request waits at a site before its request messages have been sent out.
4. System throughput. It is the rate at which the system executes requests for the critical section. If SD is the synchronization delay and E is the average critical section execution time, then the throughput is given by the following equation: system throughput=1/(SD+E)[2].