INDEX

NAME: Pradeep Ku. Kobaro. CSE-51 SEC.: CSE-51 ROLL NO.: 2205432[illegible] SUB.: DOS

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		Unit-1		
1.	15/07/2024	Introduction to DOS	1.	
2.	17/07/2024	Goals of DOS	2-9	
3.	18/07/2024	Concepts	10-12	
4.	22/07/2024	Bus & switch Bus Architecture	14-27	
5.	[illegible]/07/2024	Software Concepts - Network OS,	28-32	
		True Distributed Systems, Multi-		
		processor Timesharing Systems.		
6.	29/07/2024	Design Issues	33-40	
7.	31/07/2024	Centralized & Decentralized Algorithms		
		Unit - 2		
8.	01/08/2024	Communication in Distributed Systems	41-43	
		ISO OSI Reference Model	44-47	
		ATM Networks	48-51	
9.	05/08/2024	The Client Server Model	52-53	
10.	07/08/2024	Blocking v/s Nonblocking Primitives	54-55	
		Buffered v/s Unbuffered Primitives	56-57	
11.	08/08/2024	Reliable & Unreliable primitives,	58	
		Message Passing		
12.	12/08/2024	RPC - Basic operation, Parameter	58-60	
		Passing, Dynamic Binding		
13.	14/08/2024	RPC Semantics - Server location,	60	
		Message Lost, Client Crashes	61	
14.	15/08/2024	RPC Performance Parameters - Protocol	61	
		Selection, Acks, Critical Path, Copying,		
		Time Management		
		Unit-3		
15.	19/08/2024	Clock Synchronization - Logical v/s Physical	62	
	19/08/2024	Clock Synchronization Algorithms -	63-65	
		Cristian's Algo, Berkeley Algo, Avg.		
		Algo, Uses.		
16.	21/08/2024	Mutual Exclusion Algo - Centralized,	66-67	

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		Distributed, Token Ring		
17.	22/08/2024	Election Algorithms - A Bully Algo, A Ring Algorithm	68-71	
18.	26/08/2024	Atomic Transaction & Modelling - Stable Storage, Primitives, Properties	72-74	
19.	28/08/2024	Concurrency Control - Locking System, Optimistic approach, Time stamps	75-76	
20.	29/08/2024	Implementation of Atomic Transaction - Private workspace, Writeahead log, Two Phase Commit Protocol	77-78	
21.	23/09/2024	Deadlock & detections	79-80	
22.	25/09/2024	Threads - implement in kernel & User.	80-82	
23.	26/09/2024	System Models	83-87	
24.	30/09/2024	Processor Allocation Algorithms	88-91	
25.	02/10/2024	Scheduling	92	
26.	02/10/2024	Consistency, Replication & Fault Tolerance	93-99	
27.	03/10/2024	Consistency Protocols	100-102	
28.	07/10/2024	Distributed Commit Protocols	105-106	
29.	09/10/2024	Shared Memory	107-108	
30.	09/10/2024	DASH Machine - Architecture, Protocols	109	
31.	10/10/2024	NUMA multiprocessors, & algorithms	110-111	
32.	14/10/2024	Consistency Models	111-114	
33.	16/10/2024	Page-based DSM	113	
34.	21/10/2024	Shared Variable DSM	113-114	
35.	23/10/2024	Object-based DSM	115-117	

DISTRIBUTED OPERATING SYSTEMS

(*) What is a distributed operating system?

=> A distributed operating system is a collection of independent computers that appear to the users of the system as a single computer.

It has two aspects:

1) Hardware :- The machines are autonomous.
2) Software :- The users think of the system as a single computer.

Real time example of DOS :-

* Netflix :

i) Resource sharing :- Netflix distributes its content access of multiple servers around the world. When you share a video, the system determines the best server to deliver the content based on your location ensuring minimal suffering & high quality playback.

ii) Load balancing :- Dos helps in balancing the load. If one system can redirect some of those requests to other less busy servers, maintaining smooth performance

iii) Fault Tolerance :- If one server fails, the DOS ensures that another server can take over its duty without interrupting viewing experience. This is possible because the data and process are spread accross multiple servers, not relying on a single period of failure.

(#) Why study DOS?

=>

1) Understanding modern systems :- Many todays technologies deals with applications, rely on Dos, Studying DOS helps us understand the underlying principles that make these technologies work.

2) Enhanced skills :- Konowledge on Dos ~~helps~~ enhances our skills through which the problem solving skills and handle complex computing environment.

3) Scalability & efficiency :- DOS helps in scaling by the help of that users can design systems that can efficiently handle increasing amounts of work, which is crucial for business as they grow.

Q1. How is DOS useful in everyday life?

=> In everyday life, DOS are essential for:

1) Internet Services :- They power platforms like social media, email, cloud storage, ~~etc~~ ensuring reliability & scalability.

2) Resource sharing :- Facilitating efficient sharing of computing power, storage, and devices across networks.

3) Fault Tolerance :- Ensuring services remain available despite component failures.

4) Security :- Implementing measures to protect data integrity and prevent unauthorized access.

5) Performance :- Enabling parallel processing and efficient handling of large-scale applications and data.

Micro-computer & Micro-processor

Micro-computer	Micro-processor
i) A complete computer system.	i) The CPU or brain of a computer system.
ii) Components : micro-processor + memory + I/P & O/P devices + peripherals	ii) ALU + CU + registers on the chip.
iii) Performs all functions of a computer, including processing, storage & I/O.	iii) Performs data processing & control functions.
iv) Examples :- Tablets, Laptops, etc.	iv) Intel core i7., Apple M1, etc.
v) The role of micro-computer is the entire system used by end-users to perform tasks.	v) The role is the main processing unit of a computer.

Note : The micro-processor is a crucial part of the micro-computer, but a micro-computer is the complete system used by people for various computing tasks.

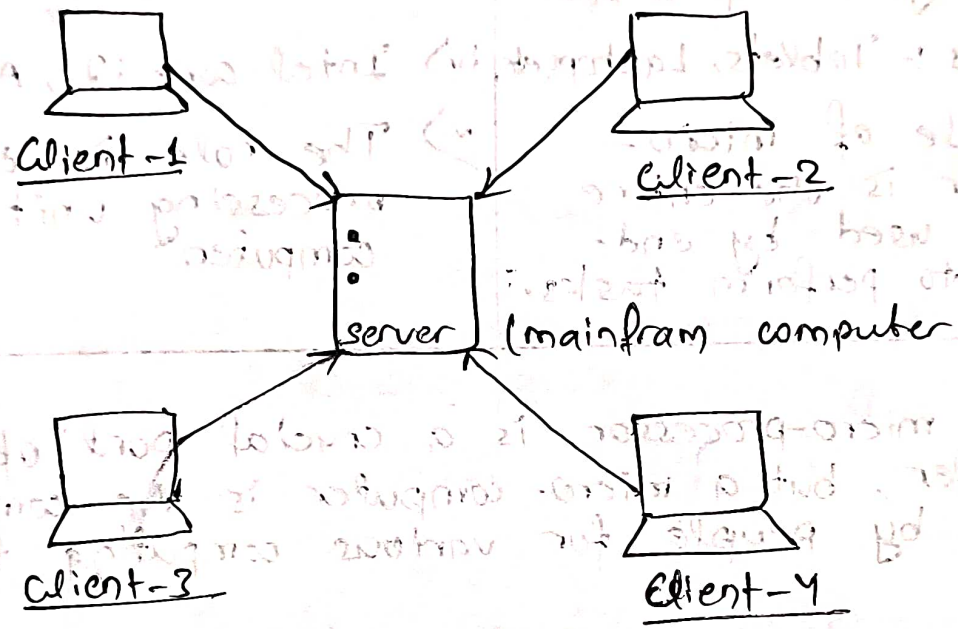
(Ans) What do you mean by centralized system? State the differences between centralized and distributed system.

⇒

Centralized System:

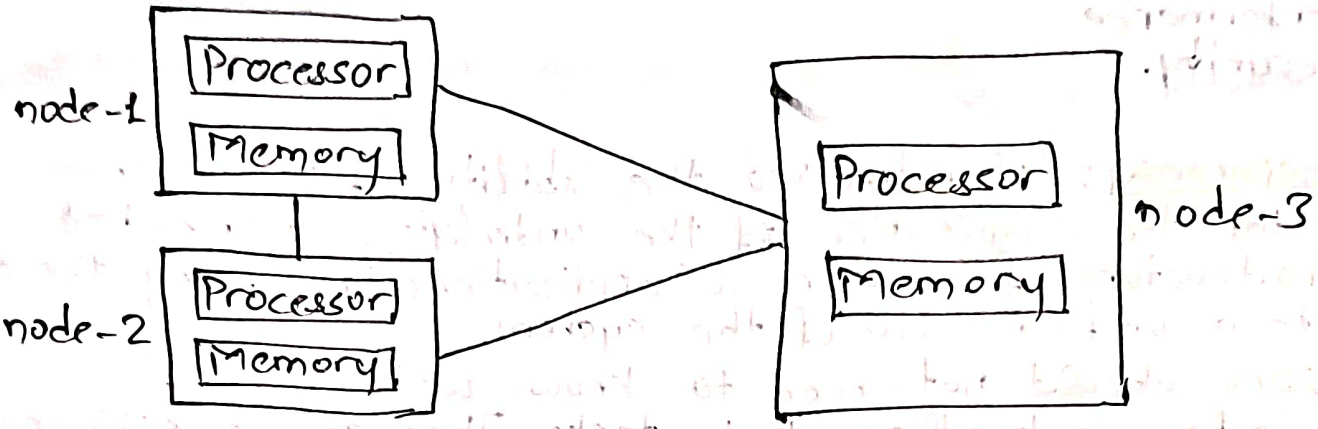
It is a computing system where single, central location.

All users and clients device connect to this central server to access data, perform task & run applications.



[#] Distributed System:

It is a computing system computer (nodes) work together to perform a task. These nodes share resources and communicate with each other, after appearing as a single coherent system to the user.



⊛ Difference between Centralized & Distributed System.

Centralized System	Distributed System
i) All processing is done in a single central server.	i) Processing is distribution across multiple interconnected nodes.
ii) Ex: A single mainframe computer serving multiple terminals.	ii) Ex: the internet, cloud services like google drive
iii) Less reliable	iii) More reliable.
iv) Limited scalability.	iv) Highly scalability.
v) Performance can degrade under heavy load.	vi) Better performance under load due to distributed processing.

Goals of distributed OS:

1) Transparency
2) Scalability
3) Reliability
4) Resource Sharing.
5) Flexibility
6) Performance
7) Security.

1) Transparency: It refers to the ability of the system to hide the complexities of the underlying distributed infrastructure from users & applications, providing them with a unified view of the system.
   - Users should not need to know which specific computer is handling their tasks. They should experience it as if they are using to single ~~peaceful~~ powerful machines.

2) Scalability:
   - allow the system to grow easily by adding more computers
   - the system should handle increasing loads without a drop in performance.
     Adding more computers should be simple & should improve performance.

3) Reliability:
   - Ensure the system is robust and can handle failures gracefully.
   - If one computer fails, the system should continue working without major disruptions. This ensures that users experience minimal downtime & data loss.

4) Resource Sharing:
   - Users should be able to access & use resources available on any computer in the network as if they were local resources, optimizing the use of h/w & s/w.

5) Flexibility:
- Support various types of h/w & s/w.
- The system should work with different types of computers & OS, allowing device to the part of the h/w.

6) Performance:
- The system should optimize task distribution & resource usage to ensure high performance, minimizing delays & maximizing throughped.
- provide fast & efficient processing.

7) Security:
- Protect data and resources from unauthorized access & threats.
- Implement measures to secure communication, authenticate users and safeguard data, ensuring privacy and data integrity.

Advantages of Distributed System over centralized:

=>
i) More reliable, failure of one node doesn't affect whole system.
ii) Easily scalable by adding more nodes.
iii) Better performance under heavy load due to distributed processing.
iv) Efficient sharing of resources across multiple nodes.
v) These can integrate different types of h/w & s/w.
vi) Less cost as compared to centralized system.
vii) Failure in one part does not stop the entire system, easier to isolate issues.
viii) Can support users & resources spread over large geographical areas.
ix) Can distribute workload across multiple nodes, optimizing efficiency.

Advantages of Distributed System over stand alone PC:

=>
i) Ability to handle larger workloads by adding more machines.
ii) Redundancy and resilience against failures.
iii) Efficient utilization of resources across multiple machines.
iv) Parallel processing capabilities for faster execution.
v) Access to resources & data from anywhere.
vi) Modular architecture for easier maintenance & upgrades.

Disadvantages of Distributed System compared to Centralized System :

=> i) Distributed systems are more complex to design, implement & maintain.
ii) More challenging to secure due to the distributed nature.
iii) Requires advanced algorithms for tasks and data management.
iv) Higher potential for delays due to h/w communication
v) Hard to ensure consistent and reliable data across node.
vi) Resource overhead is higher due to the need for redundancy and replication.
vii) Diagnosing and resolving issue is more difficult.
viii) Requires more sophisticated s/w for distributed o/p.
ix) Higher cost for infrastructure and h/w management.
x) Performance & availability depend on h/w quality.

Parallel System :

- A system where multiple processors work together on a single task simultaneously within one computer.
- All components are usually within the same physical location / computer.
- Processors are tightly coupled and depend on each other to complete tasks.
- Communication is done via shared memory or high speed interconnection which are faster & reliable.
- Lower fault tolerance, failure of a processor can affect the entire task.
- Limited reliability, adding more processors is restricted by h/w constraints.
- Example :- supercomputer, multi-core processors in modern computers.

Concepts

1) Grosch's law:

It is an old principle in computing economies. It states that, "Computing power of a computer increases as the square of its cost."

Example:- Let,

Computer A costs 1000 Rs.
Computer B costs 2000 Rs.

Acc to law:- if ~~the~~ we double the cost of a computer, we get '4' times the computing power.

Now,

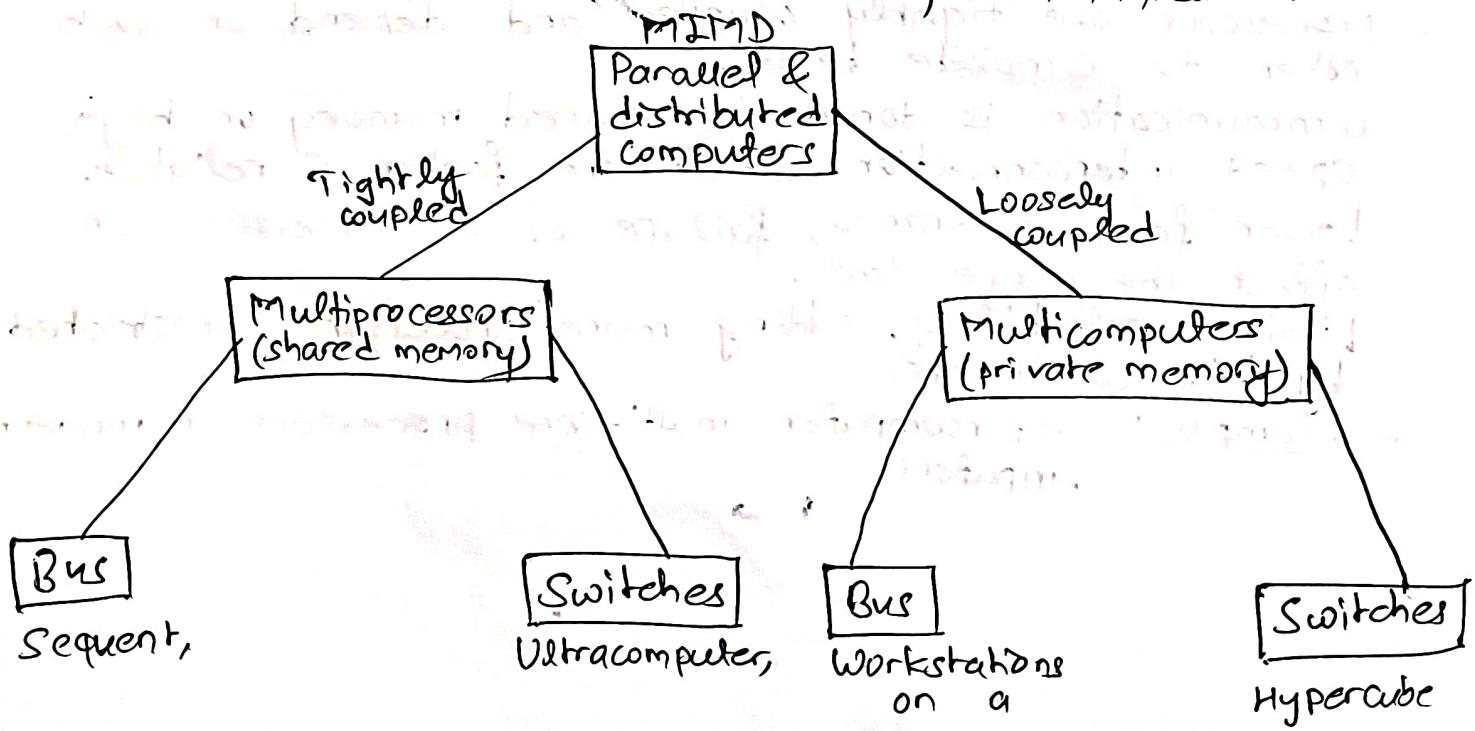
Computer 'A' has certain amount of computing power (let's say P)

then,

Computer 'B' which costs twice, should have '4P' computing power of computer A.

2) Hardware Concepts:

* The multiple CPUs in distributed System - How the CPUs are interconnected and how they communicate?



⊛ Flynn's Architecture (1972):

Characteristics:

i) The number of instruction streams
ii) The number of data streams

⊛ SIMD:

A Single Instruction is applied to Multiple Data points simultaneously.

Example:- GPUs (Graphics Processing Units)

- It is used as ideal for tasks that require the same operation to be performed on large data sets, such as image processing, matrix multiplication.

⊛ MIMD:

Different/Multiple Instructions are applied to Multiple Data points simultaneously.

Example:- Distributed Computing System, Multi-core CPU's.

- It is suitable for tasks where different processes need to run independently and simultaneously such as running multiple applications at once, complex simulations and real-time systems.

⊛ SISD: (Single Instruction on Single piece of Data)

All traditional uniprocessor computers (i.e. those having only one CPU).

⊛ MISD:

No known computers fit this model.

- Multiple Instructions on the Same Piece of Data at the same time.

Scenario based question:

Q) The amount of improvement that has happened in computer technology in machine costing 10 million dollars could execute 1 instruction/sec. New machine cost 100 dollars and can execute 10 million instructions/sec. A price performance gain of 10^10.
Specify another example.

=>

1990's : supercomputer = $10 million
performance = 1 instruction/second

2023 : modern smartphone = $1000
performance = 10 million (10^{10}) instruction/second.

Price performance gain =
- cost reduction, $10 million / 1000 = 10,000
performance improvement factor = 10 million / 1
= 10 million
(10^10)

Total performance gain =

1000 (cost reduction) x 10 million (performance improvement)
$= 10^{10}$

Data stream :- Sequence of data that is processed by the instructions in the instruction stream.

Instruction stream : sequence of commands / instructions that a computer's processor executes.

The list of steps that tell the computer what actions to perform.
Instructions can be like :-
"add these numbers".
"store this values in memory"
"Jump to this part of the program."

Data stream :- raw numbers that the computer works with, manipulates or processes based on the instructions.
- instruction is "add these numbers" the data stream should be the actual numbers being added.

Note :- Instruction stream tells the computer what to do, & Data stream provides the information needed to perform these actions.

Switch :

A switch in computing is a device that connects multiple devices together and controls the flow of data entered them.

It ensures that data goes from the source to the correct destination, without collision & efficiently manages the data traffic.

Bus :

A bus is a communication system that transfers data different components of a computer.

It allows various parts of the computer (like CPU, memory, peripherals) to communicate with each other by sending data back & forth.

Note :- Switch is like a traffic signal, it directs and controls the flow of data b/w multiple devices, insuring data reaches the correct destination.

& Bus is like a conveyor belt, it carries data b/w different parts of a computer, allowing them to communicate and work together.

Switch Architecture & Bus Architecture :

In the ~~main~~ Bus Architecture, the paths to different components of the network are shared & the response time is usually slow especially when a large number of users are there because of single path to a particular resource (shared memory).

But in case of switch networks there are concurrent paths to a resource & these paths are point to point. Due to this reason the throughput of switch based architecture is more than that of bus based architecture as multiple paths are available to a shared resource and if a user is just not getting off a resource, the other users are NOT stalled unlike bus architecture where they are stalled due to single path.

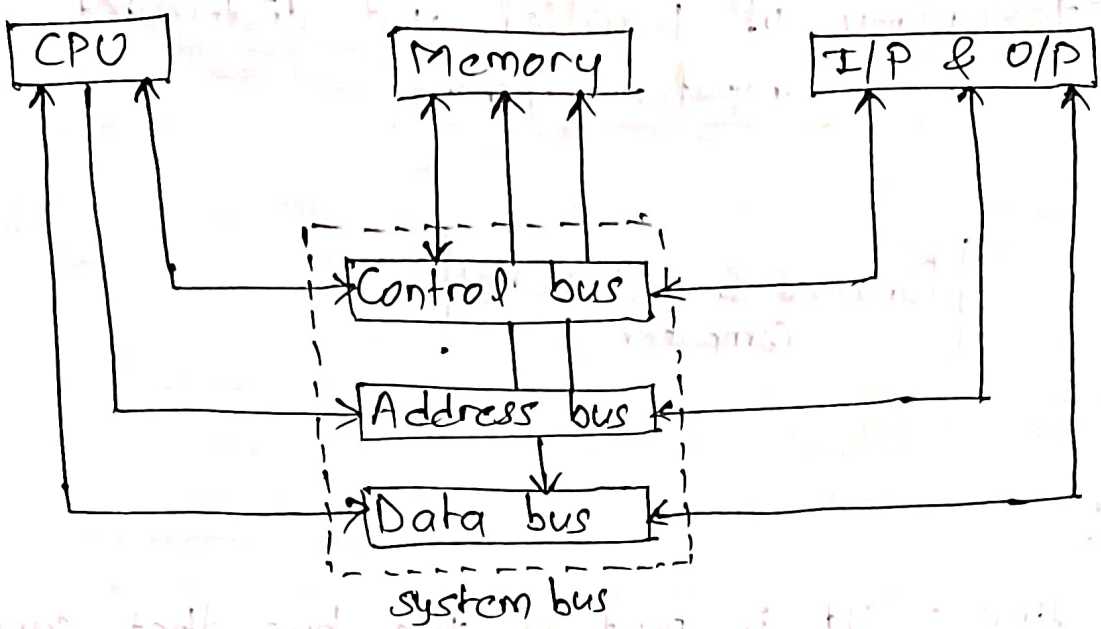


Fig :> System Bus Architecture

Delays :

ⓐ Bus : Simple, shared communication pathway (tightly coupled) prone to delays due to shared usage.

ⓑ Switch : Complex, independent communication pathway (loosely coupled), reduces delays with simultaneous communication.

Loosely & Tightly Coupled Systems

i) Loosely coupled : A system where each component operates independently with minimal dependancy on other components. Communication typically involves passing msgs over a network.

ii) Tightly coupled : A system where components are highly dependent on each other, often sharing the same memory and operating in close synchronization.
Communication b/w components is frequent & involves direct data sharing.

~~A taxonomy of parallel and distributed computer system~~

Parallel & distributed computer

Concepts :-

1) **Address line** :- It is part of the bus that carries the address of memory locations / devices. It tells the system where to find / place data.

2) **Data line** :- It is part of the bus that carries the actual data using transferred b/w components. It moves data to and from CPU, memory and other peripherals.

3) **Control line** :- It is part of the bus that carries control signals which co-ordinate and manage the actions of the system. It manages and directs the use of the address & data lines. It control signals include read/write signals, interrupt signals & clock signals.

[A] Bus Multiprocessor :

A bus multiprocessor is a type of computer architecture where multiple processors share a common communication pathway called a bus.

Note :- All processors share the same bus to communicate with memory & other components.

Working :

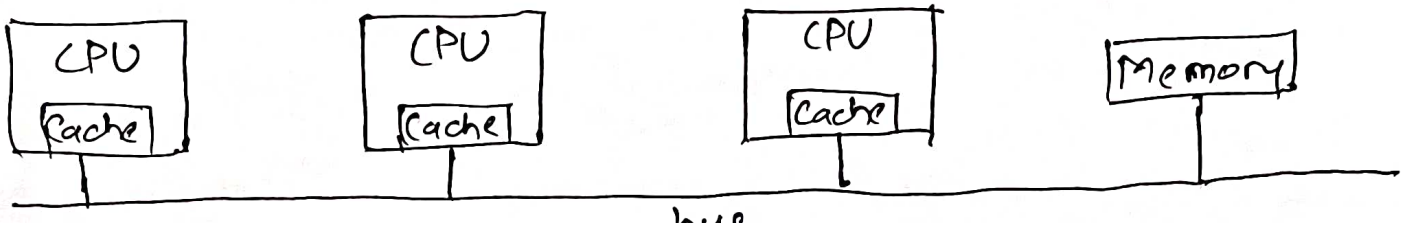
i) The bus acts as a shared highway of data.
ii) Processors take turns using the bus to access memory / other devices.
iii) Coordination is needed to manage the access to the bus, ensuring that only one processor uses it at a times to avoid conflicts.

Advantages :

i) Design is straight forward where all processors are connected to a single bus.
ii) Easier to implement and manage compared to complex architecture.
iii) Using a single bus reduces the need for multiple communication pathways, hardware cost lower.
iv) Processors can easily share the data as all the processors use the same bus to access memory.
v) It is relatively easy to add more processors by connecting them to the existing bus.

Disadvantages :

i) Since all processors share the same bus, they can only use it one at a time.
ii) As more processors are added they compute for bus access, leading to delays & reduced performance.
iii) Adding more processors does not improve performance and many even degrade it.
iv) High demand applications may suffer from insufficient bandwidth (data transfer capacity).
v) Heavy data traffic on the bus can slow down communication, leading to performance issues.



Coherence in bus based Multiprocessor:

⇒ Coherence ensures that all processors in a multiprocessor system see a consistent view of memory.
If one processor updates a memory location, other processors must see the updated value.

Example:- Imagine a shared whiteboard in a classroom. If one student writes a note on the whiteboard, all other students should see the same note. This consistency is what coherence ensures in a multiprocessor system.

High Speed Cache Memory:

It is needed to bridge the speed gap b/w the fast processors and the slower main memory.
It temporarily stores frequently accessed data to reduce the time processors spend waiting for data.
Example:- Notepad for frequently used information instead of looking it up in a big.

Hit rate in high speed cache:

It is the percentage of times the data requested by the processor is found in the cache
A high hit rate means the cache is effective in reducing access time to frequently used data.
Example:- If a student checks their notepad 100 times and finds the needed information 90 times, the hit rate is 90%.

Advantage of high speed cache in bus multiprocessor:

i) It reduces data access time, speeding up overall processing.
ii) Less frequent access than the slower main memory.

Disadvantage:

i) Managing cache coherence across multiple processors add complexity.
ii) High speed cache memory is more expensive than main memory.

Write through cache:

In write through ~~code~~ cache, every write to the cache is immediately written to the main memory as well.

Snoopy Cache:

It is a protocol used to maintain cache coherence. It involves each cache monitoring (snooping on) the bus to check if other caches are accessing data it has stored.

[B] Switched Multiprocessors

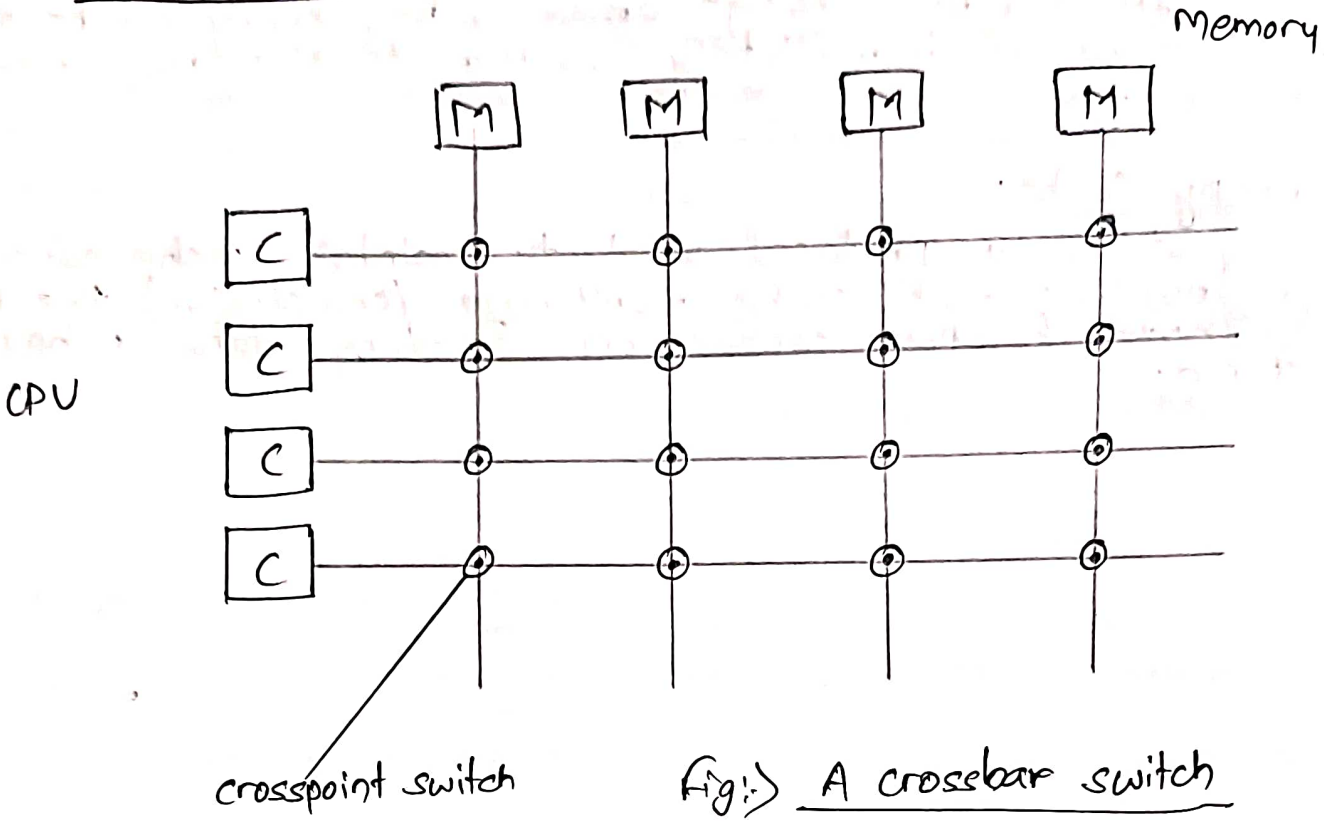


Fig:) A crossbar switch

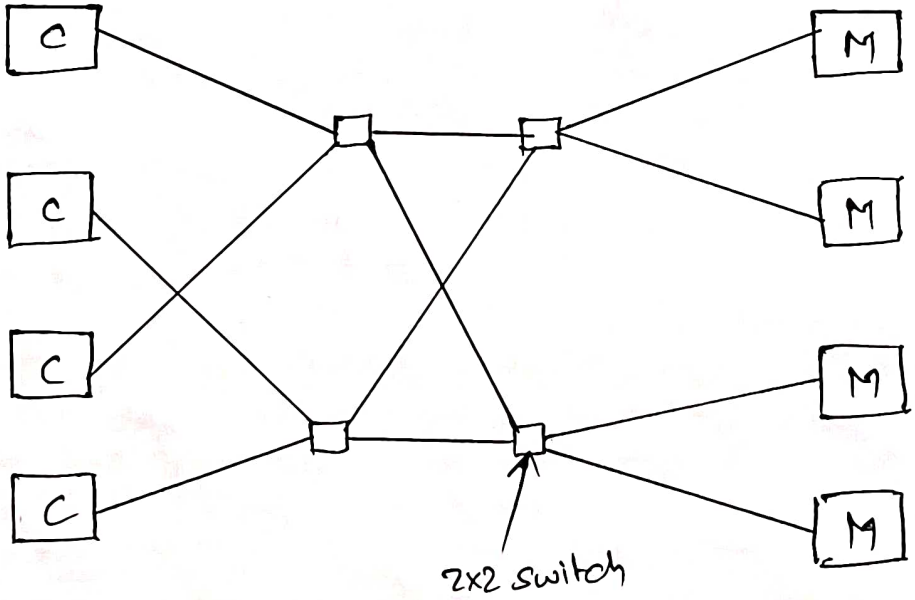


Fig:) An omega switching network

⊛ Switched multiprocessors :-

These are systems where multiple processors are connected through a network of switches rather than a single bus.
This allows for more efficient and scalable communication between processors, memory and other devices.

Types of switches :

1) **Crossbar switch :-**
It is a type of network switch where every processor is connected to every memory module through a grid of switches.
Each instruction in the grid is called as crosspoint, can be opened or closed to allow data to pass through.

2) **Crosspoint Switch :-**
Individual switches at the instructions in a crossbar switch.
Each crosspoint can be set to either connect or disconnect the pathways, allowing or blocking data flow at that instruction.

⊛ Omega Switching network :-

It is a multi stage network of switches providing indirect paths between processors & memory.

⊛ Diff. b/w :-

Features	Crossbar Switch	Omega Switching Network
Direct connections	– Yes	– No
Scalability	– limited by complexity & cost.	– better scalability with more stages.
Cost	– high due to many switches.	– lower due to fewer switches
Performance	– high with direct paths.	– variable depends on n/w load.
Complexity	– Using high with many switches.	– Lower, simple switch stages.

Uses of Crossbar Switch & Omega Switching network:

1) **Crossbar Switch**:- It is used when high performance & direct connections are critical, despite higher costs & complexity.

2) **Omega switching network**:- It is used when scalability and cost are more important, and indirect paths with multiple stages are acceptable.

Qn. What is a 2x2 switch?

=> A 2x2 switch has 2 inputs and 2 outputs.
It can route the inputs to the outputs in various ways:

Straight through:

Input '0' is connected to output '0'.
Input '1' is connected to output '1'.

Crossed:

Input '0' $\Longleftrightarrow$ output '1'
Input '1' $\Longleftrightarrow$ output '0'

Qn. What do you mean by NUMA ?

=> Non-Uniform Memory Access here CPU's have different access time to memory.

Memory access :- ① In NUMA system, each processer has its own local memory.

② Processors can access their local memory faster than remote memory.

Access time :- ① Accessing local memory is faster because it's physically closer to the processor.

② Accessing remote memory takes longer because its farther away, often requiring data to be transfered across a bus as network.

[C] Bus based multicomputers:

These are computer systems where multiple computers are connected using a shared communication pathway called as 'bus'.
Each computer has its own memory and communicates with allies through the shared bus.

Advantages:

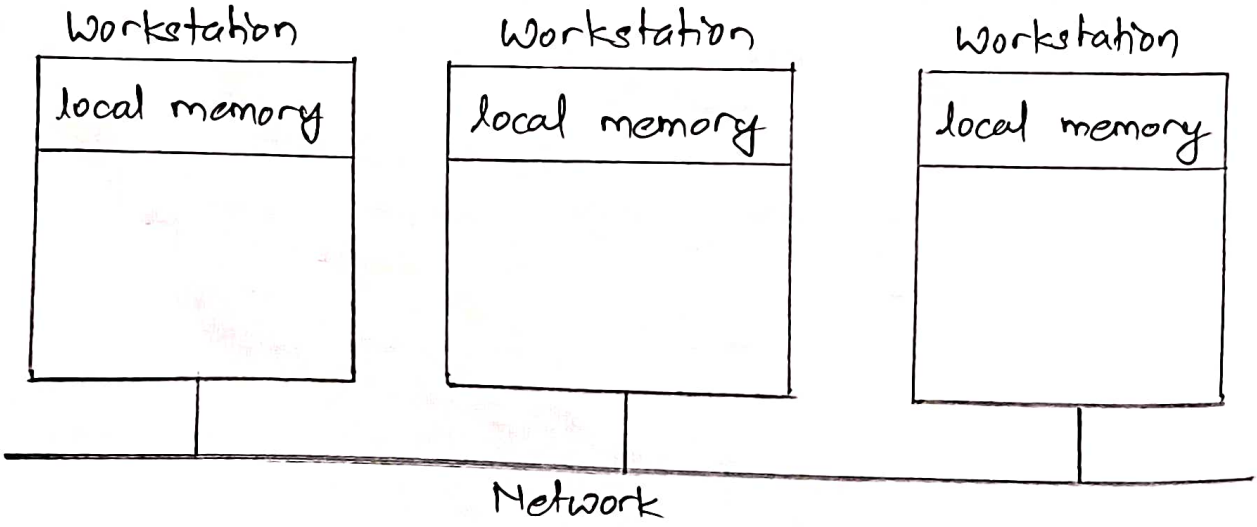
@ Simplicity: Easy to understand and implement compared to more complex n/w topologies.
ⓑ Cost-effectiveness: Less expensive to build.
© Scalability: Can be scaled by adding more computers to the bus.

Disadvantages:

@ Limited Scalability: Bus can become a bottleneck as the number of connected computers increases.
ⓑ Limited Performance: Performance can degrade as more computers share the same communic' pathway.
© Limited Flexibility: May not be suitable for all types of parallel computing tasks.

---

a multicomputer consisting of workstations on a LAN



Diff. b/w Bus Multiprocessor & Bus Multicomputer :

Aspect	Bus Multiprocessor	Bus Multicomputer
Communication	- direct communication between processors.	- Communication through a shared bus.
Scalability	- limited scalability due to bus contention.	- limited scalability as bus becomes a bottleneck.
Performance	- may degrade with more processors.	- may degrade with more computers on the bus.
Complexity	- Simple design with ~~should~~ shared memory.	- Simple design with shared communication pathway (bus).
Cost	- generally cost effective.	- cost effective but scalability can be an issue.

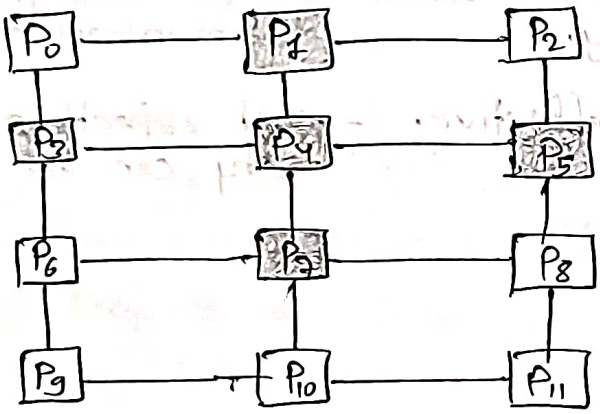
[D] Switched Multicomputer :-

These are computer systems where multiple processors are connected through a network of switches, allowing them to communicate with each other.

This setup is different from bus-based systems, where all processors share a single communication pathway (bus).

In switched multicomputers, processors can communicate directly with each other through the switches, enabling faster and more efficient data transfer.

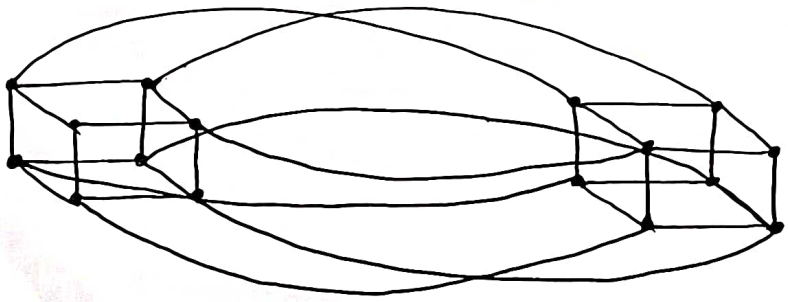
⊛ Grid architecture :



In a grid architecture, processors are arranged in a grid like fashion, similar to rows & columns.

Each processor is connected to its neighbouring processors through switches.

⊛ Hypercube architecture :



- Each processor is connected to other processors. [2 ordinary cubes with 8 entities & 12 edges]
- To communicate with another processor, data follows a path through the hypercubes, where each step represents a binary decision (ex: moving left/right/up/down). (follows a structured tree path)

Diff. b/w Grid architecture & hypercube architecture :

Aspect	Grid architecture	Hybrid architecture
Structure	- grid like arrangements with rows & columns.	- hypercubus like arrangements with binary connections.
Connections	- processors are connected to neighbouring processors.	- processors are connected through binary tree structure.
Scalability	- limited scalability as grid increases.	- better scalability with fewer connections/node.
Complexity	- ~~less~~ complex, straight forward connections.	- more complex, follows binary decision paths.
Routing	- data may need to hop through multiple processors.	- data follows a structured binary path.

Software Concepts

* **Virtual uniprocessor** :- It is a conceptual model used in multiprocessor systems to simplify the execution of tasks. In a virtual uniprocessor environment, the system makes multiple processors appear as a single processor to the s/w & the user.

* **Single run queue** :- It is a data structure used in the scheduling system of a virtual uniprocessor to manage the tasks that need to be executed. All tasks ready to run are placed in this single queue, regardless of how many processors are available.

Qn) What is meant by a Single System Image?

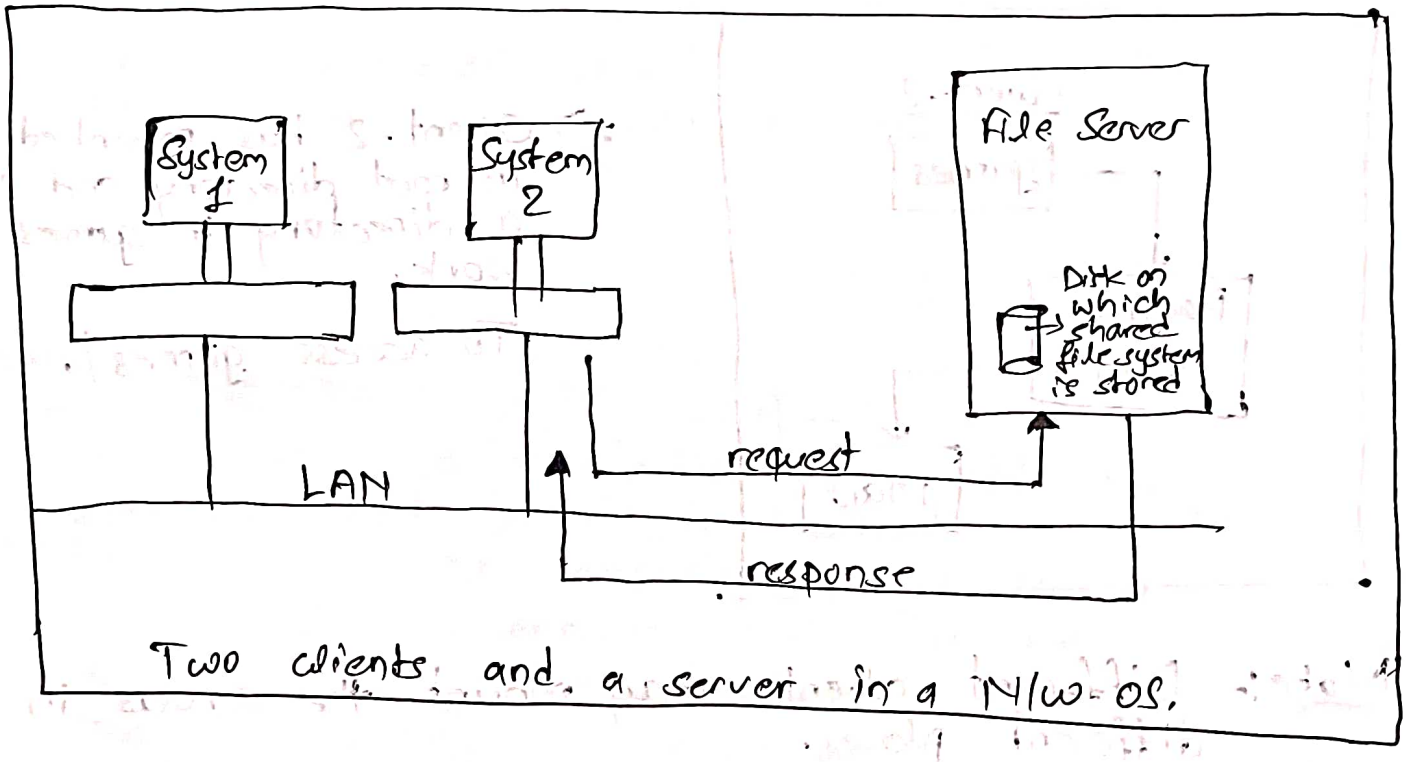
=> Single System Image (SSI) in a distributed OS means that the distributed system appears to the user as a single, unified computing system.
Ex: Google Search Engine.

Concepts:
1) Network OS
2) True Distributed System
3) Multiprocessor Timesharing System.

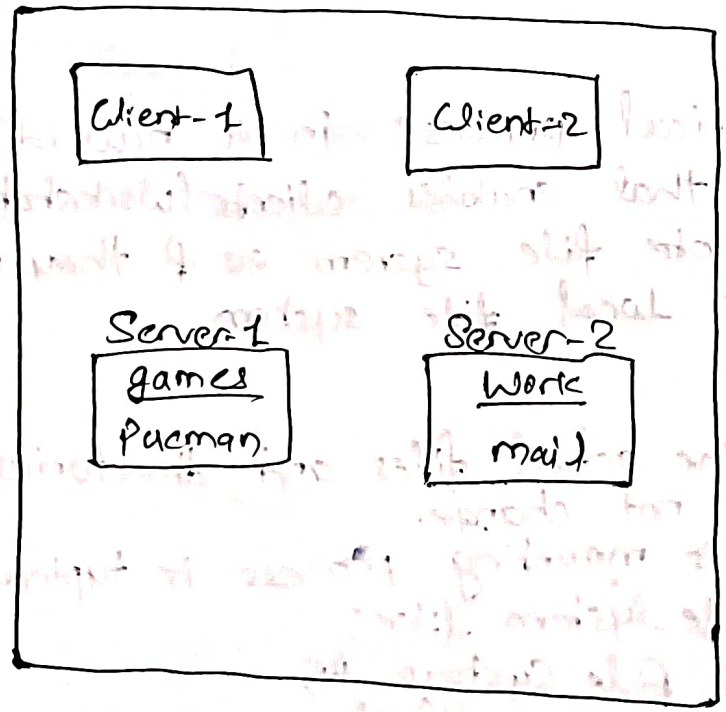
[1] Network OS :-

It is a software that connects multiple computers and devices on a network, so they can communicate and share resources.
Each computer in the network runs its own OS, but uses N/w OS to interact with other devices and share things like files, printers and internet connection.

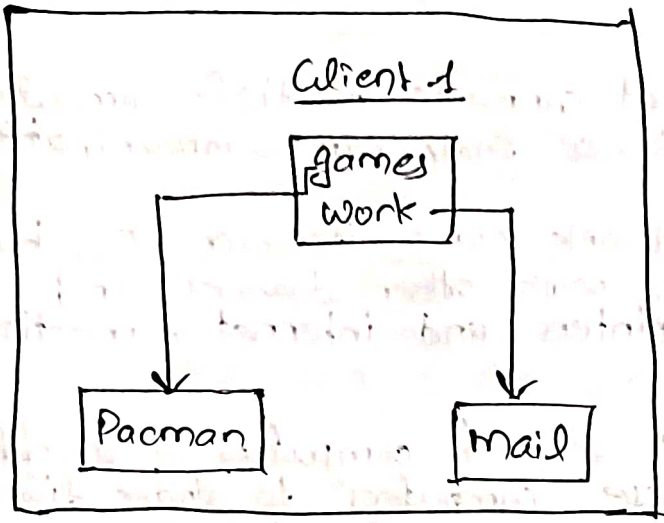
Example :- Imagine you have several computers in an office. A network OS allows these computers to share files and printers, even though each computer has its own separate OS. (like Windows or macOS)



Two clients and a server in a N/w-OS.

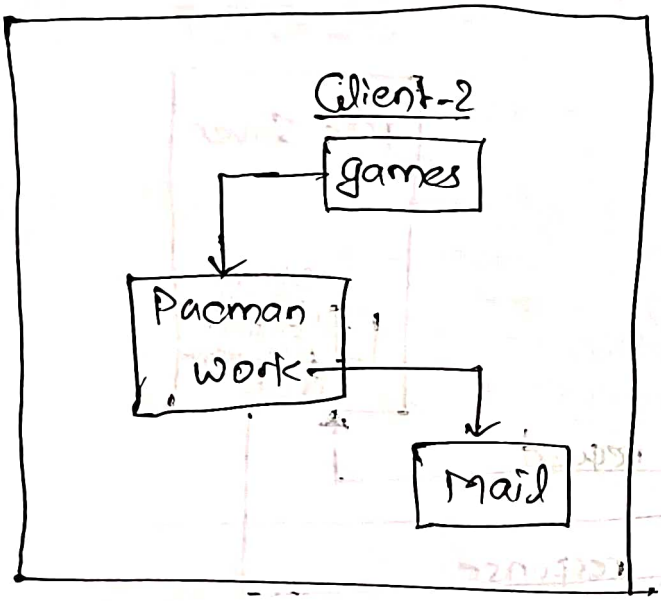


⇒ 2 file servers are used one directory called games, other called work.
(When we mount the server's file system, the file/directories on that server appear in your workstation file system, just like your own local files).



⇒ Client-1 has mounted games & work in the root directory

access :- /games
/work.



⇒ Client-2 has mounted games in root directory and created a directory in games as work.

To access games/work.

Note :- Different clients may mount the servers in different places.

Mounting: It is a critical process in a n/w id environment that enables clients (workstations) to access remote file system as if they were part of their local file system.

Note :- When mounting, the actual files and directories on the servers do not change.
This mounting process is typically done using n/w file system like:
NFS → Ntw File System or,
SMB → Servers Msg Block.

[2] True Distributed System :

It is a group of independent computers that work together so closely they appear to the user as a single system.

These systems share resources & coordinate tasks to achieve a common goal, often leading to improved performance & reliability.

The main idea is that the system hides the complexity of the underlying h/w & n/w from the user.

Example :

A distributed online multiplayer game : Players from around the world connect to different servers, but the game appears as a seamless experience.

[3] Multiprocessor Timesharing System :

It is a single computer with multiple processors (CPU) that can run multiple tasks at the same time, sharing the computing power among them.

The OS ensures that each task gets a fair share of the CPU time, making it appear as though they are running simultaneously.

Example :

Imagine a powerful servers with several CPU's. It can run a web server, a database and a file server all at once.

Differences betn Network OS, True Distributed System & Timesharing Multiprocessor Timesharing System:

Aspect	Network OS	True Distd Sys.	Multiprocessor TS Sys.
Virtual Uniprocessor	- does not appear as a single computer.	- appear as a single, unified system.	- appears as a single sys with multiple processors
Run the same OS	- each computer may run different OS (win, macos)	- all parts run the same OS.	- all processors run unde one OS.
OS copies	- each machine has its own copy of os.	- single copy spread across the system.	- single copy on a mul processor machine.
Commun	- Uses networking & message passing.	- uses transparent message mechanisms.	- uses shared memory.
Network Protocols required	- required to facilitate communication.	- required for internal communication.	- not required, comm is internal.
Single run queue	- No, each system has its own run queue.	- No, tasks managed across multiple nodes.	- Yes, a single run que for all processors.
file Sharing semantics	- basic file sharing	- well defined, consis-tent file sharing semantics.	- well defined, consisten file sharing within the system.

System Architecture:

It is a fundamental concept in technology & engineering that refers to the high-level design & structure of complex systems.

V. Imp

Key Components:

1) Hardware infrastructure
2) Software stack
3) Network configuration
4) Data storage
5) Web User Interface (UI)
6) Security measures
7) Scalability solutions
8) Integration protocols
9) Monitoring & logging
10) Backup & recovery

Design Issues

(8) What do you mean by transparency in DOS? Explain different types of transparency.

⇒ Transparency means hiding the complexity of the system from users & applications.
It makes the system appear as a single, unified system even though it may be composed of multiple, interconnected computers.

Types:

1) Location Transparency :- Users and applications physical location of resources (like files, printers, etc) in the n/w. They can access these resources without knowing where they are actually located.

Ex: accessing a file, you do not need to know at which server the file is stored. You can open the file on your local machine, & the system will handle & retrieve the file.

2) Migration Transparency :- resources & processes can move from one location to another within the n/w without affecting the operation of applications/users.
The system manages the relocation seamlessly.

Ex: Suppose an application is running on "Secure A". If the system decides to move it to "Secure B" for load balancing/maintainance, the application continues to run without the user noticing any change.

3) Replication Transparency :- multiple copies of resources can exist to increase reliability & performance.
Users and applications are unaware of the replication and interact with the resource as if there is only one copy.

4) Parallelism Transparency :- the system can execute multiple operations in parallel to improve performances, but this parallelism is hidden from users & applications.
They perceive the operations as happening independently, without worrying about the underlying parallel processes.

5) Concurrency Transparency :- multiple users/applications concurrently without interference.
The system manage the concurrent access to ensure consistency & correctness.

These types of transparency helps simplify the use of distributed systems, making them more efficient & user friendly by abstracting the underlying complexity.

Features	Microkernal	Monolithic Kernal
Design approach	- minimalist core with user services.	- all-in comparising kernal with integrated services.
Kernal size	- small	- large
Performance	- generally slower due to IPC overhead.	- generally faster due to direct calls.
Stability/ Security	- higher (user-spaces bugs has likely to crash.)	- lower (kernal space bugs can crash the system).
modulatity	- high (easy to update/replace services).	- low (integrated, hard to modify parts)
development complexity	- higher (separate services).	- lower (integrated services).
example	- Minix	- Linux, windows

Design Issues :

1) Transparency :- It refers to the ability of the system to make its operations and behaviour understable and visible to users or administrators. It also includes how well the system hides complexity from users.

⊛ Strategies for Achieving Transparency :
- Provide clear documentation and user interfaces.
- Use logging and monitoring tools to offer visibility into system operations.
- Design APIs and interfaces that abstract away the complexity from end users.

2) Flexibility :- It is the system's ability to adapt to changes and new requirements without requiring major rework.

⊛ Strategies :
- Employ modular design principles, such as microservices or plug-in architectures.
- Use configuration files or management interfaces that allow for dynamic changes.
- Implement extensibility points where new functionality can be added without altering existing code.

3) Reliability :- It refers to the system's ability to consistently perform its intended functions without failure.

⊛ Strategies :
- Incorporate redundancy and failover mechanisms to handle component failures.
- Implement comprehensive error handling and logging to detect and address issues promptly.
- Use rigorous testing and validation processes to identify & fix potential reliability issues.

4) Performance :- Performance pertains to how efficiently the system performs its tasks, including speed, responsiveness, and resource usage.

⊛ Strategies :
- Optimized algorithms and data structures to improve processing speed.
- Use catching and load balancing to manage high traffic and reduce latency.
- Regularly profile and benchmark the system to identify and address performance bottlenecks.

5) Scalability :- It is the system's ability to handle increased load or demand by adding resources or expanding capacity.

* Strategies :
  - Design the system with distributed architecture principles, allowing for easy addition of nodes.
  - Use scalable databases and storage solutions that can handle growing amounts of data.
  - Implement load balancing to distribute traffic and workloads efficiently across resources.

Reliability & Availability in DOS :

Reliability	Availability
- focuses on correctness & accuracy over time.	- focuses on uptime & accessibility.
- It ensures the system continous to work correctly despite faults.	- It ensures the system is up & running where users need it.

Fault Tolerence :- It is the ability of a DOS to continues functioning correctly even when some of ~~the~~ its components fail.

* Key Concepts :
  # Redundancy :- Having multiple copies of critical components so that if one fails, other can take over. multiple servers hosting the same application.
  # Failure : The process of automatically switching to a standby component where a failure is detected.
  # Recovery : The ability to restore the system to a normal state after a failure.

Concepts :

Centralized components :- These are single points within a system that perform specific function. All other parts of the system depend on these centralized components for certain operations.

Centralized tables :- There are single databases that stores critical information needed by various parts of the system. All nodes in the system refer to this centralized tabble to access/update data.

Centralized algorithm : It relies on a single decision making to perform all its functions. This central controller gathers information from various parts of the system, processes it and makes decision.

Decentralized algorithm :- Distributes the decision making process among multiple nodes/components within the system. Each node operates independently and makes decisions based on local information and possibly information from other nodes.

Aspects	Centralized algorithm	Decentralized algorithm
Control Point	- easier to manage & co-ordinate because there is only one decision making point.	- multiple independent nodes.
Decision making	- centralized & consistent.	- distributed & potentially inconsistent.
Failure impact	- single point of failure.	- no single point of failure, system can continues to operate even if some nodes fail.
Scalability	- The central point can become a bottleneck as the system grows, leading to performance issues.	- better scalability as the system can handle growth more efficiently as decision making is spread out.
Co-ordinations	- Simpler co-ordination, easier to manage & coordinate because there is only one decision making point.	- more challenging to manage and co-ordinate since there are multiple decision points.

Qn.) In an experiment file server is up 3/4th of the time & down 1/4th of the time, due to bugs. How many times the file server have to be replicated to give an availability of atleast 99% ?

⇒

To achieve atleast 99% availability for a file server that is 75% (3/4) of the time up & (25%) 1/4 down.

need to determine how many replicas of the server are necessary.

Aim :- availability of 1 server(A) = 0.75
unavailability of 1 server(U) = 0.25
desired continued availability = 0.99

Formula : When we have multiple replicas of the server, the system is unavailable only if all replicas are down simultaneously.

for 'n' replicas, continued unavailability is,

$$U_{continued} = U^n$$

(continued unavailability to be less than or equal to 0.01 (1-0.99 = 0.01)).

$$\text{unavailable} = (0.25)^n \leq 0.01$$

$$\log(0.25)^n \leq \log(0.01)$$

$$n \log 0.25 \leq \log(0.01)$$

$\log(0.25) \approx -0.602$
$\log(0.01) = -2$ } (finding log values)

$$n \geq \frac{-2}{-0.602}$$

$$\geq 3.32$$

$$n \geq 4 \text{ (round off)}$$

∴ we need atleast '4' copies of the file server to achieve availability of 99%.

Qn) Suppose that you have a large source program consisting of 'm' files to compile. The compilation is to take place on a system with 'n' processors, where n>>m. The best we can hope for is an M-fold speed up over a single processor. What factors may cause the speed up to be less than the maximum? Explain

⇒ M-fold speedup :- It refers to the idea that if we split a task into 'm' parts and run each part in parallel the task should ideally take $\frac{1}{m}$ of the time it would take to do the whole task sequentially.

if we can perfectly parallize a task, having 'm' processors should make the task 'm' times faster.

Scenario :- Lets say we trace to paint a large wall, and it would take 1 person 4 hours to paint alone.

1 person (processor) takes = 4 hours

M-folds speedup with 4 people :- $\frac{\text{4 hours (1 person)}}{\text{4 people}}$

= 1 hour (with 4 people).

When compiling a large source program with 'm' files on a system with 'n' processors (n>>m) theoritically speedup we can achieve is an 'm' fold speedup.

Several factors are in reality that cause speedup to be less than this maximum:

① Dependency b/w files :- some files may depend on others, meaning they cannot be compiled until the files they depend on are compiled first.

② Overhead of parallelism :- dividing the work among multiple processors and managing them involves some overhead this includes the time taken to distribute tasks & gather results.

③ Uneven workload distribution : If the file take diff. amounts of time to compile, some processors may finish their tasks earlier and sit idle, waiting for others to finish.

④ Communication delays :- If the processors need to communicate frequently, the time spent on communication can reduce the effective speedup.

⑤ Resource contention :- Multiple processors competing for shared resources like memory, disk access, on n/w bandwidth can slow down the process.

Example :- Consider 4 files to compile & 4 processors.

(a) Ideal condition :- no overhead, perfect distribution each file takes 10 mins to compile.

$$\text{Ideal speedup} = \frac{10 \text{ mins/file} \times 4 \text{ files}}{4 \text{ processors}} = 10 \text{ mins in total.}$$

(b) Reliable condition :- (considering overhead + uneven distribution)

(a) Overhead for managing parallelization = 2 mins.

(b) Compiling time = File A (10 mins)
File B (10 mins)
File C (10 mins)
File D (12 mins) - due to complexity

Total Compilation Time = longest file compilation + overhead
= 12 mins (File D) + 2 mins (overhead)
= 14 mins total.

Qn. Why are network conditions and bandwidth important?

=> Network conditions affect how quickly and reliably data can be sent and received.

Bandwidth determines the capacity of the n/w to handle data. Higher bandwidth means more data can be transmitted simultaneously, leading to faster & more efficient communication.

Unit-2

Communication in Distributed System

Q) Why do we require OSI & ATM ? Why do we have two different systems ?

=> OSI (Open Systems Interconnection)
ATM (Asynchronous Transfer Mode)

1) OSI Model :-

It is a conceptual framework used to understand and design network communication.
It provides a standard approach for different network protocols to work together.

Significance :
i) OSI model helps standardize n/w communication, making it easier for different systems & technologies to interoperate.
ii) Breaks down communication into server layers, allowing for modular design. Each layer can be developed & improved independently.
iii) The layered approach helps in troubleshooting network issues by isolating problems to specific layers.

2) ATM (Asynchronous Transfer Mode) :

It is a networking technology used for transmitting data, voice & video over a single network.
It uses small, fixed-size cells to ensure efficient and predictable data transfer.

Significance :
i) It allow for efficient data transfer and reduce latency, making it suitable for real-time applications like video conferencing.
ii) ATM provides mechanisms to ensure different types of data (voice, video, data) are delivered with the required quality of service, such as guaranteed bandwidth & low latency.
iii) ATM can handle a large number of simultaneous connections, making it suitable for large n/w's like the ones used by telecom operators.

Qn) Why two different systems?

⇒ - The OSI model is a theoretical framework for understanding and designing n/w protocols.
- ATM is a specific technology used for transmitting data efficiently.

Frames :- It is a container that holds a chunk of data, along with control information.

Packet :- It is a small piece of data that is sent over a network. It includes both the data & control information like the destination address.

Error checking :- It is a method to ensure that data is transmitted correctly and without errors.

Qn) What is meant by an Open System? Why some systems are not open?

⇒ An Open System uses, standardized protocols and interfaces to communicate and interact with other systems. This allows different computers and devices to connect, share resources and work together seamlessly.

Some systems are not open because they use proprietary protocols / interfaces that are not compatable with other systems. They are designed to work in isolation or only with specific, compatible devices.

Open System :- promote interoperability and flexibility, allowing different technologies to work together.

Closed System :- are more secure and controlled but less flexible in terms of compatability and communication with other systems.

Connection-oriented & Connection-less protocols

[1] Connection-Oriented Protocols:

A connection is established ~~due to~~ between the communicating devices before any data is sent.
This connection is maintained throughout the communication session and is terminated only after all data has been transferred.
Ex:-

TCP (Transmission Control Protocol):

- TCP establishes a connection using a 3-way handshake before data transmission begins.
- It ensures reliable delivery of data by acknowledging received packets and retransmitting lost packets.
  Ex; Email, file transfer where reliability is crucial.

[2] Connectionless Protocols:

In a connectionless protocol, data is sent as individual packets without establishing a dedicated connections.
Each packet is treated independently and may take different paths to reach the destination.
Ex:

UDP (User Datagram Protocol):

- UDP sends packet called datagrams without establishing a prior connections.
- There is no acknowledgement of received packets, and lost packets are not retransmitted.
  Ex:- Online gaming, live video streaming.

Notes:

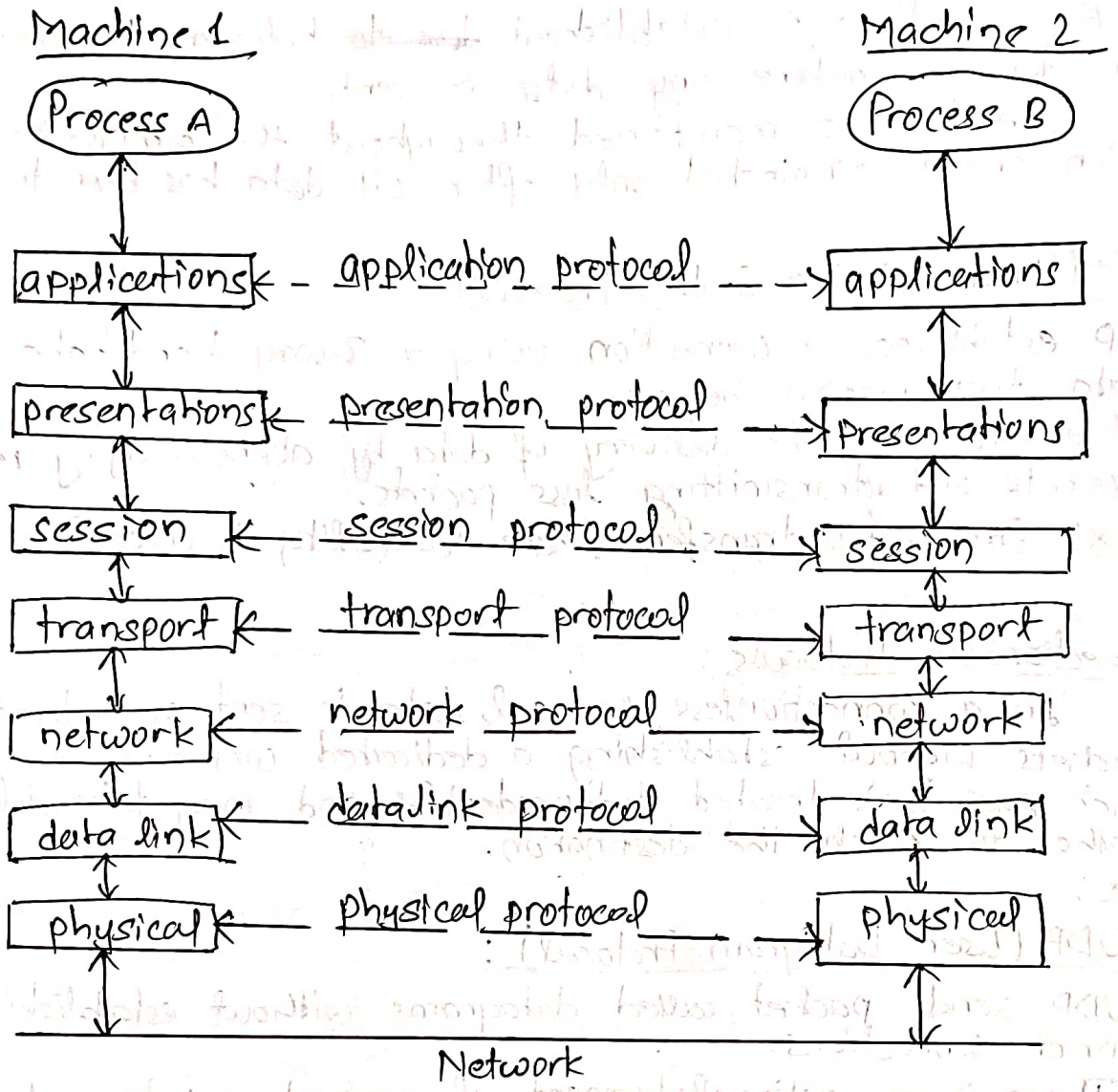
a) Header: a block of information attached to the beginning of a data packet.

b) Layers: individual levels in OSI model each with specific function.

c) Interface: the boundary between two layers in the OSI model. It defines how the layers interact & exchange information.

d) Protocols: a set of rules and standards that define how data is transmitted and received accross a network.

OSI Model Layers



① Physical layer (track carrying the package)	transmits raw bits (ethernet, wifi)
② Data link layer (shipping label with the sender and receiver's address)	frames and error checking (ethernet)
③ Network layer (the postal route and sorting centers determining the path)	routes packets (IP, X.25)
④ Transport layer (the confirmation that the package was delivered safely)	ensures reliable data transfer (TCP, UDP)
⑤ Session layer (ensuring the communication b/w sender & receiver continues smoothly)	manage sessions (video calls)
⑥ Presentation layer (packing the item securely and possibly adding a gift wrap)	translates data formats (encryption, data concession)
⑦ Application layer (the content of the package) (can be a gift, a letter)	provides n/w services (web browsers, email clients)

Example Process :-

1) You write and send an email (application layer)
2) The email is encrypted (presentation layer)
3) A session is established with the email server. (session layer)
4) The email is broken into segments and sent reliably. (transport layer)
5) Each segment is given an IP address & routed (network layer)
6) Segments are framed with MAC addressess and error-checking. (data-link layer)
7) Frames are converted into signals and transmitted. (Physical layer)

1) Physical layer :- It is the first layer of the OSI model, responsible for the physical connection b/w devices.
It handles the transmission of raw binary data (0's & 1's) over physical mediums such as coreless radio waves or fiber optics.

2) Data link layer :- It is the second layer of the OSI model. It manages the node to node transfer of data and ensures error-fail transmission by using frames.

3) Network layer :- It handles the routing of data packets from the source to the destinations across multiple network

4) Transport layer :- Ensures complete data transfer & error recovery between end systems. It manges data flow control, error checking, and retransmission of lost data. Key protocols here include TCP & UDP.

5) Session layer :- Manages & controls the connections between computers. It establishes, maintains, & terminates sessions or connections between applications.

6) Presentation layer :- Translates data between the application layer & the network. It handles data encryption, compression, & translates between different data format

7) Application layer :- Provides network services directly to end-user applications. It interfaces with software applicatio to provide various network services such as email file transfer, & web browsing.

Protocols of transport layer:

Aspects	TCP	UDP
Connections	- connection-oriented	- connectionless
Reliability	- reliable (error-checking, retransmission)	- unreliable (no error-checkin
Speed	- slower	- faster
Use cases	- web browsing	- online games

Protocol Stack :

It refers to the set of protocols used in network where each layer of the stack is responsible for a different aspect of communication.

Internet Protocols Stack (IP/TCP stack) :

1.) Application layer : (writing the letter)
- http for web browsing
- FTP for file transfer

2) Transport layer : (manages end to end communication & reliability)
- TCP for reliable connection
- UDP for faster, less reliable transmission

3) Network layer : (taking the envelope to the post office & routing it)
- IP for addressing & routing

4) Data link layer :- (the postal worker ensuring the letter is sorted correctly)
- Ethernet for wired connections.

5) Physical layer :- (the truck or plane carrying the letter to its destn)
- deals with the physical connections & transmission of raw data
- (cables, radio waves)

Asynchronous Transfer Mode (ATM)

ATM :- It is a networking technology designed for the high speed transmission of various types of data, such as voice, video & computer data, over a network.

Key features :

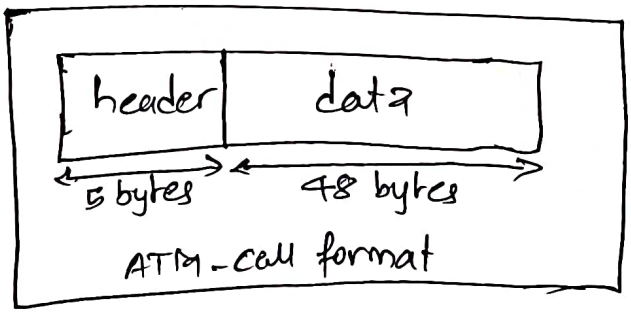
1) Fixed Size Packets (cells) :-
   -> ATM uses small, fixed size packets called cells. Each cell is 53 bytes long, with 5 bytes for the header & 48 bytes for the data.
   -> This uniform size helps maintain high speeds and consistent performance.

2) Asynchronous transmission :
   -> Unlike traditional n/w's that might send data in a synchronized or regular timing ~~pleatform~~ pattern, ATM sends cells asynchronously.
   -> This means cells are sent as soon as they are ready, allowing for efficient use of available bandwidth.

3) High speed & efficiency :
   -> ATM is designed to handle making it suitable for applications that require quick and effecient communication, such as video conferencing or real-time data streaming.

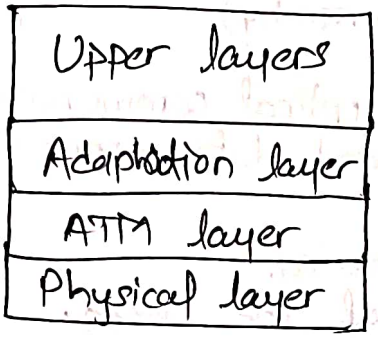
4) Quality Of Service :
   -> ATM supports different levels of service quality, ensuring that critical applications (like voice or video) get the necessary bandwidth and priority over less critical data.

5) Cell-switching technology :
   -> ATM uses a cell-switching technique where each cell is routed independently through the n/w.
   -> Switches in the n/w direct each cell to its destination based on information in the cell's header.



ATM-cell format

ATM Reference Model :



- **Physical layer** :- deals with transmission of cells over physical media.

  This includes : ⓐ standards for transmission speeds &
  ⓑ formats of the frame used for sending ATM cells.

- **ATM Layer** :- This layer is comparable to data link layer of OSI model. It accepts the 48 byte segments from the upper layer, adds a 5 byte header to each segment and converts into 53 byte cells. This layer is responsible for routing of each cell, traffic management, multiplexing & switching.

- **Adaptation layer** :- This layer corresponds to network layer of OSI model. It provides facilities to the existing packet switched networks to connect to ATM network and use its services. It accepts the data & converts them into fixed sized segments. The transmission can be of fixed or variable data rate. This layer has two sub layers - Convergence sub layer & Segmentation and Reassembly sub layer.

Notes on Physical layer:

SONET (Synchronous Optical Network) &
SDH (Synchronous Digital Hierarchy)

These are the standards for optical communication that provides the framework for high speed data transmission over fibre optic networks.

- SONET is used in North America.
- SDN is used in the rest of the world.

They are similar but have different naming connections & slightly different specifications.

Key Concepts :

⊛ Optical Communication : Is a way of sending information using light.

⊛ Fibre optic network : uses these special cables to connect different places, like homes, offices & data centre. Inside a fibre optic cable, there are thin strands of glass or plastic fibres. Light signals are sent through thus fibres to transmit data over long distance at every high speeds.

Optical Carrier (OC) levels :

OC-1 :- the base level, providing a data rate of 51.84 Mbps.

OC-3 :- Provides a data rate of 155.52 Mbps.

OC-3c :- "Concatanated" OC-3, which treats the bandwidth as a signal unit rather than 3 separate OC-1.

OC-12 :- provides a data rate of 692.08 Mbps.

OC-12c :- Concatanated session of OC-12.

OC-48 :- provides a data rate of 2.488 Gbps.

OC-192 :- Provides a data rate of 9.953 Gbps.

these levels indicate the transmission capacity and are multiples of the base OC-1 rate.

Cell Switching :- It is a method used to transfer data in a fixed size units called cells.

Multiplexing :- It is a technique used to combine multiple data streams into one signal over a shared medium.

ATM cell structure :

i) GFC => Genric flow Control => 4 bit field
ii) VPI => Virtual Path Identifier => 8 bit field
iii) VCI => Virtual Channel Identifier => 16 bit field
iv) CLP => Cell Less Priority => 1 bit field
v) CRC => Cyclic Redundancy Check => 8 bit field
vi) Payload type => 3 bit field.

i) GFC :- a 4 bit field used for local flow control between devices and the ATM network.

ii) VPI :- an 8 bit field in the user network interface (UNI) or a 12 bit field in the network-network-interface (NNI) that identifies the virtual paths.

iii) VCI :- a 16 bit field that identifies the virtual channel within a virtual path.

iv) CLP :- indicating the priority of the cell.
if the cell is set to 1 = discorded during congestion.

v) CRC :- used for error detections in the header.

vi) Payload type :- indicating the type of payload in the cell.

Client-Server Model

It is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and delivers the data packets requested back to the client. Clients do not share any of their resources. Example of Client-Server Model are :- Email, WWW, etc.

Client and Server :

* **Client** :- When we say the word 'Client', it means to talk of a person or an organization using a particular service. Similarly, in the digital world, a Client is a computer (Host) i.e. capable of receiving information or using a particular service from the service providers (Servers).

* **Servers** :- Similarly, when we talk about the word Servers, It means a person or medium that serves something. Similarly in this digital world, a Server is a remote computer that provides information (data) or access to particular services.

So, it is the Client requesting something and the Server serving it as long as it is in the database.

Addressing Process via Machine :-

The addressing process ensures that communication b/w clients and servers is correctly routed through a network. It involves several key steps:

* **IP Addressing** :
  - → Purpose : IP addresses uniquely identify devices on a network.
  - → IPv4 : Uses a 32-bit address format (eg, '192.168.1.1').
  - → IPv6 : Uses a 128-bit address format (eg, '2001:0db8:85a3:...')

* **Port Numbering** :
  - → Purpose : Ports allow multiple services to run on a single IP address by specifying different communication channels.
  - → Well-known Ports : Ranging from 0 – 1023 (eg. port 80 for http)
  - → Dynamic/Private Ports :- Ranging from 49152 – 65535, used in ephemeral

⊛ DNS Resolution :

Process : Converts human-readable domain names (e.g. 'www.example.com') into IP address.

Broadcasting :

It is a method of sending messages to all devices within a specific network segment.

- Local broadcasting :

Usage :- Common in LANs for tasks like discovering devices or services.

Ex :- ARP (Address Resolution Protocol) requests are broadcasted to find a device's MAC address associated with a given IP address.

Limitations :

Network Congestion :- Excessive broadcasting can lead to high traffic and network congestion.

Scope : Typically limited to local network segments ; global broadcasting is less common & usually avoided in favour of multicast or unicast.

ASCII Names Lookup :-

It is about translating human-readable names into machine-readable addresses, primarily through DNS.

- DNS (Domain Name System) :

Function :- Resolves domain names into IP Addresses to facilitate the communication between clients & servers.

Process :- As described above, involves querying ~~about~~ various DNS servers to obtain the IP address for a given domain name.

- ASCII Representation :

Domain Names : Domain names use ASCII characters, which are standard & easily readable.

Internationalized Domain Names (IDNs) : For non-ASCII characters, domain names are encoded using Punycode.
Ex : ('xn--fsq' for "中国 in Punycode).

Blocking versus Nonblocking Primitives :

In a client-server model, blocking and non-blocking primitives refer to how operations on resources (like network sockets, files, or other I/O operations) handle waiting for these operations to complete.

[A] Blocking Primitives :

In a blocking operation, the client (or server) makes a request and then waits (blocks) until the operation is complete before proceeding to the next task. During this time, the thread executing the operation is idle and cannot perform other tasks.

Example :- A blocking 'recv()' call on a socket waits for data to arrive. The calling thread is paused until data is received.

Use Cases :

-> Blocking operations are simple & easy to implement. They are useful when the operation is expected to complete quickly, or when the client/server can afford to wait.

Disadvantages :

- Inefficient for I/O operations that may take a long time, as it ties up the thread and resources.
- Can lead to performance bottlenecks, especially in systems with highly concurrency demands.

[B] Non-Blocking Primitives :

In a non-blocking operation, the client (or server) makes a request and continues executing without waiting for the operation to complete. The operation returns immediately, possibly with a status indicating that it is still in progress.

Example :- A non-blocking "recv()" call on a socket immediately returns with either the received data or an indication that no data is currently available. The calling thread can continue to perform other tasks or check back later.

Use Cases :

- Non-blocking operations are useful in scenarios where high concurrency is needed, such as in event-driven architectures or when implementing asynchronous I/O.

Disadvantages :

- More complex to implement, as the application needs to manage the completion of tasks and handle scenarios where the operation is not yet complete.
- Requires careful management of resources and state, especially when dealing with multiple non-blocking operations.

Parameters	Blocking Primitives	Non-Blocking Primitives
Execution	- The operation waits until it is fully completed before returning control to the program.	- The operation returns immediately, without waiting for completion, often with a status indicating progress
Thread behaviour	- The thread is paused (or blocked) during operation & cannot perform other tasks.	- The thread continues executing other tasks and can check the operation's status later.
Resource Utilization	- Less efficient as the thread is idle, waiting for the operation to complete.	- More efficient, allowing the CPU to handle multiple tasks concurrently.
Implementation	- Simpler to implement.	- Complex.
Use case	- Suitable for quickly operations or where waiting is acceptable.	- Ideal for high-concurrency scenarios, event-driven architectures, or when waiting is not acceptable.
Example	- A blocking 'recv()' call on a socket waits for data to be received before continuing.	- A non-blocking 'recv()' call on a socket returns immediately, possibly with no data, allowing the thread to perform other tasks.

Buffered versus Unbuffered Primitives:

Buffered & unbuffered primitives refer to different approaches to handling data I/O operations in computing. These concepts are particularly relevant in scenarios where data is read from or written to a resource, such as files, network sockets, or other I/O devices.

[A] Buffered Primitives:

Buffered primitives use an intermediary memory area, known as a buffer, to temporarily store data before it is processed or transmitted. The buffer collects data until it reaches a certain size or a specific condition is met (like a flush command), at which point the data is sent or processed in bulk.

⊛ How it works?:

- When data is written using buffered primitives, it is first stored in the buffer.
- The data is then processed or transmitted once the buffer is full or when an explicit flush operation is called.
- Similarly, when reading data, the buffer may pre-fetch data from the source to reduce the number of direct I/O operations.

[B] Unbuffered Primitives:

Unbuffered primitives directly process or transmit data without using an immediately buffer. Each operation is handled immediately, without any delay or aggregation of data.

⊛ How it works?:

- Data is read from or written to the resource as soon as the I/O operation is called.
- There is no intermediate storage; each operation corresponds to a directly interaction with the resource.

Difference between Buffered & Unbuffered Primitives :

Parameters	Buffered Primitives	Unbuffered Primitives
Data handling	- Data is temporarily stored in a buffer before being processed or transmitted. This allows for larger chunks of data to be handled at once, improving efficiency.	- Data is processed or transmitted immediately, without being temporarily stored in a buffer. Each operation directly interacts with the underlying resource.
Efficiency	- Can be more efficient because they reduce the no. of I/O operations.	- Less efficient for large amount of data.
Latency	- Buffered operations may introduce latency, as data is not immediately processed or transmitted; it waits in the buffer until the buffer is full or a flush operation occurs.	- Unbuffered operations generally have lower latency because data is processed or transmitted as soon as the operation is initiated.
Use case	- Ideal for bulk data transfers where efficiency is prioritized.	- Suitable for real-time applications requiring immediate data processing
Example	- A buffered 'send()' operation on a socket may store data in a buffer and only transmit it once the buffer is full or explicitly flushed by the application.	- An buffered 'send()' operation on a socket transmits data immediately to the server, ensuring low-latency communication.

Reliable and Unreliable Primitives :

Parameters	Reliable Primitives	Unreliable Primitives
Guranteee	- Guarantee data is delivered accurately and in the correct order, often using acknowledgment & retransmission mechanisms.	- Does not guarantee the delivery, order, or integrity of the data. Data may be lost, duplicated, or received out of order.
Error handling	- Includes error detection & correction, with retransmissions if data is lost or corrupted.	- Minimal or no error detection and correction; the application must handle any issues.
Overhead	- Typically higher, due to additional processing for acknowledgments, error checking, & retransmissions.	- Lower, since there are fewer mechanisms in place for ensuri data delivery.
Use case	- Suitable for applications where data integrity and order are critical, such as file transfers, email, & financial transactions.	- Suitable for applications where speed is more important than reliability, or where occasional data loss is acceptable, such a live streaming or online gaming
Example	- TCP is a reliable protocol that ensures data is delivered correctly.	- UDP is an unreliable protocol that does not ensure data delivery.

Message Passing :

It is a method used in the client-server model for commn between the client and server. It involves sending & receiving messages (data packets) between processes, which can be on the same machine or across a network. This approach is widely used in distributed systems to implement the client-server architecture.

Implementing the Client-Server Model with Message Passing :-

1) Message Structure :- Header (metadata) and payload (data).
2) Client :- Sends request messages.
3) Server :- Processes requests and sends responses.
4) Communication Methods :-
   - (a) Synchronous :- Client waits for a response.
   - (b) Asynchronous :- Client continues without waiting.
5) Protocols :-
   - (a) TCP :- Reliable, ensures delivery.
   - (b) UDP :- Unreliable, faster

6) Concurrency :- Handled using threads or asynchronous I/O.
7) Security :- Encryption and authentication for secure communication.
8) Use cases :- RPC, RESTful services, and distributed systems.

Remote Procedure Call (RPC) :-

It is a method used in client-server architectures to allow a client to execute procedures (or functions) on a remote server as if they were local.

Basic Operation :

1) Client Request :- The client application calls a local stub procedure (proxy) instead of the actual remote procedure.
2) Stub :- The client stub encodes the procedure call and its parameters into a message & sends it over the network to the server.
3) Server Stub :- The server receives the message, decodes the procedure call & parameters, & invokes the corresponding remote procedure.
4) Execution :- The server executes the procedure & sends the result back to the client stub.
5) Client Stub :- The client stub receives the response, decodes it, and returns the result to the client application.

Parameter Passing :

1) Marshalling :- Converting parameters into a format suitable for transmission over the network (serialization).
2) Sending :- The marshalled data is sent from the client to the server.
3) Unmarshalling :- Converting the received data back into the original parameter types on the server side.
4) Execution & Response :- After executing the procedure, the server marshals the result and sends it back to the client, where it is unmarshalled and used by the client application.

⊛ Dynamic Binding :

1) **Definition** :- Dynamic binding (or late binding) allows the client to call procedures without knowing the exact server implementation at compile time. The actual procedure to be called is determined at runtime.

2) **Service Discovery** :- Clients can discover available procedures & their signatures dynamically, often using a service registry or directory.

3) **Interface Definition** :- Typically, procedures are defined in an Interface Description Language (IDL), & stubs are generated based on this definition.

4) **Flexibility** :- Allows for updates & changes to the server-side implementation without requiring changes to the client application, as long as the interface remains consistent.

RPC Semantics during different failures :-

1) **Server Location Changes** :

When the server's location changes, such as due to an IP address update or server reallocation, RPC systems use service discovery mechanisms like DNS or service registries to locate the new server address. Load balancers also play a role by masking server location changes from the client. If the client encounters an outdated server address, it will retry the connection or be redirected to the updated server location.

2) **Message Loss** :-

To manage message loss, RPC systems employ acknowledgment mechanisms where the server sends a confirmation of received requests. Clients use timeouts to detect when acknowledgments are not received within a specified period, triggering automatic retransmissions of the message. Procedures are designed to be idempotent, meaning repeated requests will have the same effect as a single request, preventing unintended side effects from message loss and retries.

3) **Client Crashes** :

When a client crashes, the server may use timeouts to detect when a client becomes unresponsive and clean up any resources associated with that client. To handle client crashes gracefully, RPC procedures are often designed to be ~~session~~ stateless, reducing reliance on the client's ongoing presence. Additionally, session management systems can maintain session info. on the server, allowing clients to recover & resume their session upon reconnection after a crash.

RPC Performance Parameters :-

RPC (Remote Procedure Call) performance is influenced by several key parameters that can either optimize or degrade the system's efficiency.

Parameters :-

1) Protocol Selection :-
The choice of communication protocol (eg. TCP for reliability, UDP for speed) affects latency, throughput, and reliability.

2) Acknowledgements :-
Acknowledgement strategies (synchronous vs. asynchronous) influence the trade-off between reliability & performance, affecting response times & throughput.

3) Critical Path :
The sequence of operations that determine the total execution time. Optimizing the critical path reduces latency and improves RPC responsiveness.

4) Copying :-
Data copying between different system layers can introduce overhead. Reducing or eliminating copying (e.g. using zero-copy techniques) enhances performance.

5) Time Management :-
Efficient management of timers for timeouts and retransmissions balances system responsiveness and resource usage, preventing delays and improving reliability.

Clock Synchronization:

Clock synchronization is crucial in distributed systems to ensure that the different nodes or machines in the network maintain a consistent notion of time. Without synchronization, processes on different machines could experience inconsistencies, such as incorrect event ordering or failure to coordinate tasks correctly.

It is vital for maintaining consistency, coordination, and correct event ordering in distributed systems. It involves both logical clocks, which manage event order, and physical clocks, which maintain real-world time, with various protocols & algorith used to address synchronization challenges.

Logical Clocks versus Physical Clocks :-

Parameters	Logical Clocks	Physical Clocks
Purpose	- Order events based on casuality, not actual time.	- Track and synchronize real-world time across nodes.
Time Representation	- Abstract counters or vectors (e.g. Lamport timestamps).	- Actual wall-clock time (e.g. hours, minutes).
Synchron-ization	- No real-world time sync; focuses on event order.	- Requires protocols like NTP to ensure consistent real-time.
Application	- Used where event order matters, like in distributed databases.	- Used where real-time accuracy is critical, like in time-sensitive operations.

Clock Synchronization Algorithms :

1) Cristian's Algorithm :

⊛ **Principle:-** Cristian's Algorithm is used for synchronizing a client's clock with a server's clock in a network.

⊛ **Process :-**

1) **Request :** The client sends a request message to the time server, at time T_1.

2) **Reply :** Upon receiving the request, the server notes the time T_2 and sends a response back to the client, including T_2 and the server's current time T_s.

3) **Receive :-** The client receives the response at time T_3.

4) **Calculate :-**

i) **Round-Trip Time :** The round-trip time is $(T_3 - T_1)$.

ii) **Propagation Delay :** Assume the network delay is symmetric. The one-way delay is $\frac{(T_3 - T_1)(T_s - T_2)}{2}$.

iii) **Adjusted Time :-** The client adjusts its clock by adding this one-way delay to the received server time T_s to estimate the correct time.

⊛ **Challenges :**

Cristian's Algorithm assumes that network delays are symmetrical, which might not always be the case. It's also sensitive to network jitter & delays.

[2] The Bakeley Algorithm :

⊛ Principle :- The Bakeley algorithm is used for synchronizing clocks in a distributed system with a central coordinator usually in a more robust manner compared to simpler algorithms.

⊛ Process :

1) Polling : The master node (coordinator) periodically polls each node in the system to collect their current clock times.

2) Average Calculation : The master node computes the average of these times.

3) Adjustment : The master node sends out an adjustment message to each node, informing them of the difference between their current time and the computed average time.

4) Clock Update : Each node adjusts its clock based on the adjustment message received from the master node

⊛ Challenges :

The Berkeley Algorithm assumes that the master node is reliable and that all nodes have approximately synchronized clocks to begin with. It can also be affected by network delays, and a single point of failure exists if the master node fails.

[3] Averaging Algorithms :

Averaging algorithms can be used within clock synchronization methods to aggregate multiple time readings & reduce discrepancies.

In the context of clock synchronization, averaging algorithms might include :

1) Simple Average :- Calculating the mean of all node's clock times as used in the Berkeley Algorithm. This is a basic form of averaging that helps in smoothing out inconsistencies.

2) Weighted Average :- Nodes might be assigned different weights based on their reliability or network latency, and the average is computed accordingly. This can help in situations where some nodes are more reliable than others.

3) Trimmed Mean :- Removing outliers or extreme values from the set of clock times before averaging can improve accuracy. This is useful in scenarios where some nodes might have significantly skewed clocks.

4) Median :- In cases where the data is skewed or contains outliers, the median might be used instead of the mean to provide a robust estimate of the average time.

Summary :
Cristian's Algorithm and the Berkeley Algorithm both aim to synchronize clocks but use different methods & assumptions. Averaging algorithms are often used to aggregate multiple time readings to improve synchronization accuracy & robustness.

Use of Clock Synchronization :

i) Distributed Systems :- Ensures consistent data and coordination across nodes.
ii) Financial Transactions :- Maintains transaction order & audit trails.
iii) Networking :- Synchronizes devices for accurate protocol operations and security.
iv) Real-Time Systems :- Manages task scheduling and event ordering.
v) Data Acquisition :- Provides accurate timestamps for data consistency.
vi) Telecommunications :- Coordinates communication and manages timing.
vii) Cloud Computing :- Aids in resource management and consistency.
viii) Gaming :- Ensures fair play and synchronized game events.
ix) Media :- Aligns content delivery and editing.

Mutual Exclusion Algorithms

These algorithms are designed to manage access to shared resources in concurrent computing environments, ensuring that only one process or thread can access a critical section (a shared resource) at any given time. This prevents issues like race conditions, where multiple processes simultaneously modify shared data, leading to inconsistent or incorrect results.

[1] Centralized Algorithm :

⊛ Principle :- A single coordinator (central authority) manages access to the critical section ~~sends a request to the coordinator~~ for all processes.

⊛ Process :

1) Request : A process that wants to enter the critical section sends a request to the coordinator.

2) Grant : If no other process is in the critical section, the coordinator grants permission by sending a reply.

3) Release : After finishing in the critical section, the process sends a release message to the coordinator, allowing another waiting process to enter.

[2] Distributed Algorithm :

⊛ Principle :- There is no central coordinator; instead, processes coordinate among themselves to ensure mutual exclusion.

⊛ Process :

1) Request :- A process that wants to enter the critical section sends a request message with a timestamp to all other processes.

2) Reply :- Each process receiving the request responds if it is not in the critical section & has no earlier requests. Otherwise it queues the request and replies later.

3) Access :- The requesting process enters the critical section, the process sends a release message to all processes that were waiting for it.

[3] Token Ring Algorithm :

⊛ Principle :- Processes are arranged in a logical ring, and a token (a special message) circulates among them. Only the process holding the token can enter the critical section.

⊛ Process :-

1) Token circulation :- The token continuously moves from one process to the next in the ring.

2) Access :- A process can only enter the critical section if it possesses the token. After using the critical section, the process passes the token to the next process in the ring.

3) Release :- If a process does not need the token, it simply forwards it to the next process.

Comparison :

i) Centralized Algorithm :- Efficient in small systems but vulnerable to a single point of failure.

ii) Distributed Algorithm :- Scalable and resilient but involves high communication overhead.

iii) Token Ring Algorithm :- Fair and simple but can be inefficient and requires careful handling of token loss or process failures.

Election Algorithms

- Election algorithms choose a process from a 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.

Two algorithms :

1) The Bully Algorithm
2) The Ring Algorithm

1) Bully Algorithm :

- The Bully Algorithm works by letting the "biggest" process take charge.

* Working :-

i) Election Trigger :- If a process (let's call it P) notices that the current coordinator is not responding (perhaps it has crashed), P decides to start an election to find a new coordinator.

ii) Election Process :- P sends an "ELECTION" message to all processes that have higher numbers (IDs). If no one with a higher number responds, then P declares itself the new coordinator.

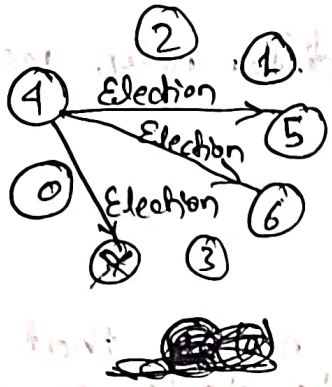
iii) Response from higher Process :- If a process with a higher number does respond, it sends back an "OK" message to P, telling it to stop because the higher-numbered process will take over the electi

iv) Higher Process Election :- The higher-numbered process that responded now starts its own election, following the same steps, by sending "ELECTION" messages.

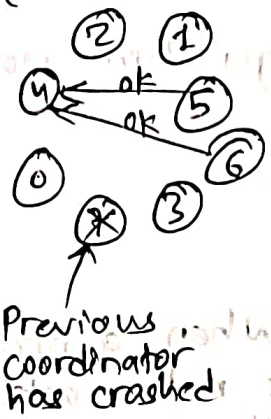
v) New Coordinator Announcement :- This continues until the process with the highest number wins. This proces then sends a "COORDINATOR" message to all other processes, announcing itself as the new coordinator.

* Special Case : Recovery :- If a process that was previously down comes back up, it will automatically start an electio If it has the highest no. among all running processes, it will win & become new coordinator, "bullying" its way to the top.

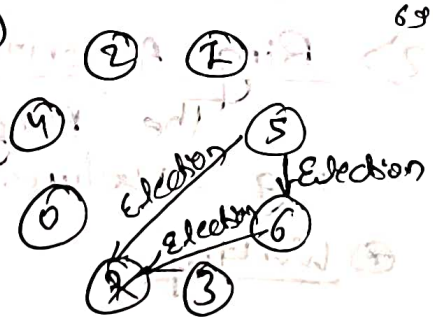
(a)



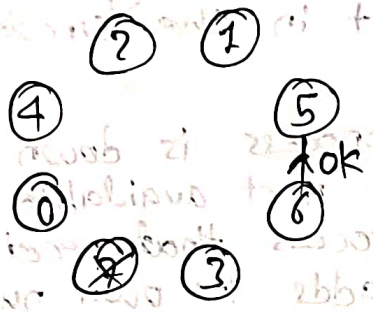
(b)



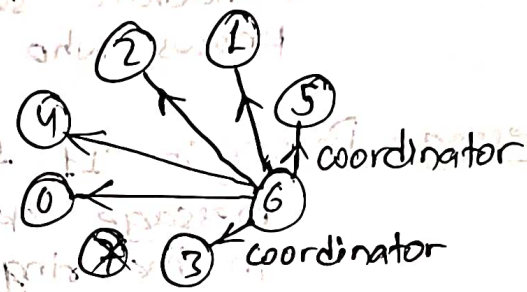
(c)



(d)



(e)



Summary:

Imagine you have 8 processes, numbered from 0 to 7. Process 7 was the coordinator but has crashed. Process 4 notices this and starts an election by sending "ELECTION" messages to processes 5, 6 & 7. Processes 5&6 respond, so process 4 stops its election & waits. Then, process 6 starts its election. Since process 7 is down, process 6 wins and announces itself as the new coordinator.

2) Ring Algorithm :

The Ring Algorithm works differently, using a ring structure.

* Working :

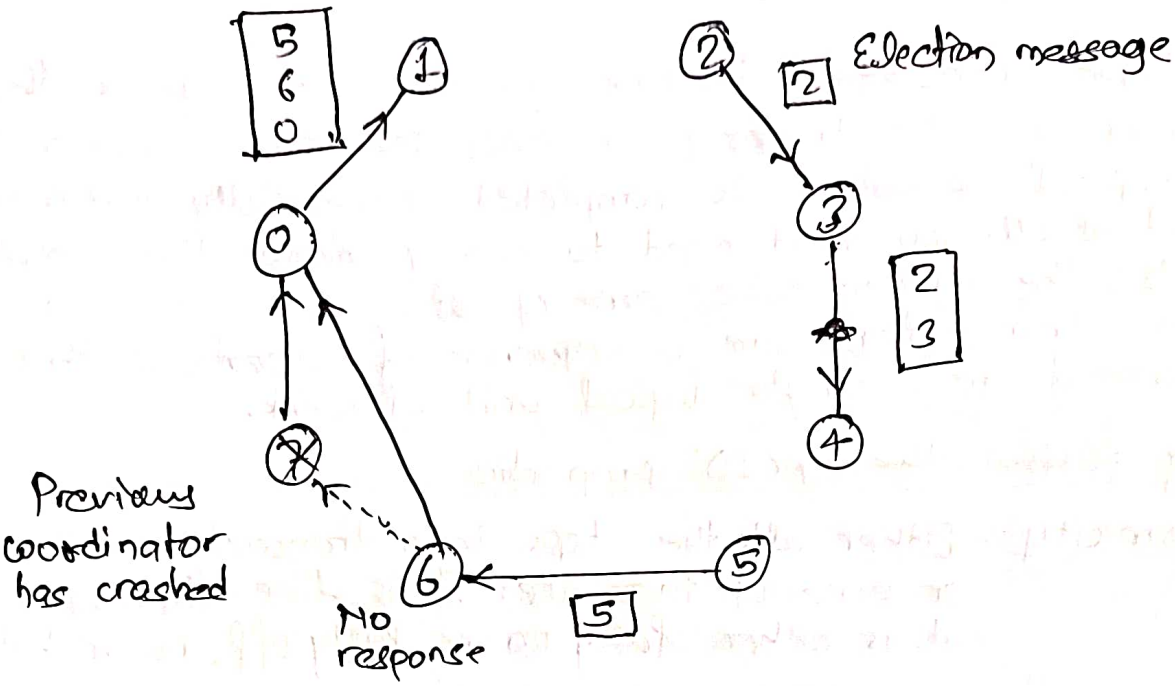
i) Election Trigger :- When a process notices that the coordinator is not working, it starts an election by sending an "ELECTION" message to its immediate successors in the ring (each process knows who comes next in the circle).

ii) Message Passing :- If the next process is down, the message skips to the next available process in the ring. Each process that receives the "ELECTION" message adds its own number to the list inside the message.

iii) Completing the Ring :- The message eventually comes back to the original sender (the process that started the election). When this happens, the sender knows that it has gone around the whole ring.

iv) New Coordinator Selection :- The process then changes the "ELECTION" message to a "COORDINATOR" message and sends it around the ring again, this time to inform everyone who the new coordinator is (the process with the highest number in the list).

v) Normal Operation Resumes :- Once the "COORDINATOR" message has gone around the ring, everyone knows who the new,



(Aspects) Parameters	Ring Algorithm	Bully Algorithm
Election process	- Each process sends an election message around a logical ring, including its own process number.	- A process that detect a coordinator failure sends election messages to all higher numbered process.
Selection Criteria	- High number process in the ring wins & becomes coordinator.	- Higher number process that responds to the election message wins and becomes the coordinator.
Message complexity	- O(N)	- In worst case, $O(N^2)$
Coordinator Announcement	- After winning, the coordinator sends a message around the ring to announce itself.	- The winning coordinator sends a message to all processes to announce itself.
Handling failures	- If a process or its successor is down, the election message skips to the next available process	- The algorithm can handle multiple process failures and still elect the process with the highest no.
Simultaneous election	- Multiple simultaneous elections can happen but will converge, resulting in the same coordinator	- Simultaneous elections are possible, but the process with the highest no. always wins.
Topology dependency	- Requires logical or physical topology	- Does not require any specific topology.
Efficiency	- More efficient	- Can be less efficient.
Scalability	- Scales well in systems where the no. of processes is fixed & known	- Can scale to larger systems but may incur higher communication overhead.
Simplicity	- Simpler to implement.	- Slightly more complex.

#Atomic Transactions#

- An atomic transaction is like an automatic system that manages all the traffic (operations) for you. It ensures that a group of operations is completed successfully and consistently, or not at all. You don't need to worry about the complex details; the system takes care of it.
- Atomic transactions are a sequence of operations that are performed as a single logical unit of work.

- They follow the "ACID" properties:

1) Atomicity:- Either all the steps in a transaction are completed, or none of them are. It is like flipping a switch: it is either fully on or fully off, no in-between.

2) Consistency:- The transaction ensures that the system remains in a consistent state before and after the transaction. Think of it like balancing your checkbox: after every transaction, the numbers have to add up correctly.

3) Isolation:- Even if multiple transactions are happening at the same time, each one appears to be the only one occuring. It's like each transaction has its private workspace.

4) Durability:- Once a transaction is completed, the results are permanent, even if the system crashes right afterward. It's like writing something in pen instead of pencil - once it is done, it stays that way.

Q.) Why Use Atomic Transactions?

=> Atomic transactions are like a high-level toolkit that simplifies the programmer's job. Instead of worrying about all the low-level details (like avoiding crashes or deadlocks), the programmer can focus on what the program needs to do.

For example, if you're writing software that manages a bank's transactions:

- With low level synchronization, you'd have to manually ensure that each transaction is processed correctly, even if multiple transactions are happening at the same time, and that the system can recover if something goes wrong.
- With atomic transactions, you can simply define what each transaction should do (like transferring money between accounts), and the system will automatically handle the synchronization, ensuring everything works

Stable Storage:

It refers to a type of non-volatile storage that is robust against failures, ensuring that once data is written, it will not be lost, even in the event of system crashes or power failures. This is crucial for maintaining the durability property of transactions.

* Mechanisms involved:

i) Write-Ahead Logging (WAL): Before any changes are made to the database, the changes are first written to a log. If a crash occurs during the transaction, the system can use the log to recover to a consistent state.

ii) Redundant Storage: Data is stored redundantly across multiple devices or locations to prevent data loss in case one of the storage mediums fails.

Transaction Primitives:

- Real-Time Example: Imagine you're writing a book, and you have a special pen that can "commit" or "rollback" your changes. You start writing (BEGIN_TRANSACTION), make some edits (WRITE), and then decide whether to keep the changes (END_TRANSACTION) or erase them (ABORT_TRANSACTION). If you commit, the edits become permanent. If you abort, it's like you never wrote anything.

- In Computers: In programming, transaction primitives are special commands that let you manage these operations. For example:
  - BEGIN_TRANSACTION: Marks the start of a transaction.
  - END_TRANSACTION: Tries to commit (make permanent) the transaction.
  - ABORT_TRANSACTION: Rolls back (undoes) all changes if something goes wrong.
  - READ/WRITE: Read from or write to files or databases.

Properties of Atomic Transactions:

1) Isolation levels :- Controls how & when changes by one transaction are visible to others. Ranges from minimal isolation (fast but risky) to full isolation (safe but slow)
2) Cohension : Ensures all steps in a transaction are logically related, maintaining system consistency.
3) Idempotence : Repeating a transaction multiple times yields the same result, preventing unintended side effects.
4) Durability : Once a transaction is committed, its changes are permanent, even after a system crash.
5) Concurrency Control : Manages transactions so they don't interfere with each other, avoiding conflicts like lost updates.
6) Atomicity : The transaction is all-or nothing; if any part fails, the entire transaction is rolled back.
7) Consistency : Transactions must move the system from one valid state to another, preserving rules & constraints.
8) Rollback & Recovery : Allows the system to undo failed transactions, restoring the system to the previous state.

Nested Transactions:

- Real-Time Example: Imagine you're managing a big event with multiple tasks. You might delegate some tasks to others (like setting up chairs, preparing food, etc), but you still oversee everything. Each person's task is a subtask of the main event. If something goes wrong with one subtask, it might be possible to fix it without affecting the entire event.

- In Computers: In computer systems, a transaction can obtain sub-transactions (nested transactions). The main transaction might split into smaller tasks, running in parallel on different processors for better performance. If a sub-transaction fails, it can be rolled back without affecting the others, making the whole process more efficient & resilient.

Concurrency Control:

It ensures that when multiple processes (or transactions) are happening simultaneously in a distributed system, they don't interfere with each other. Think of it as managing traffic at a busy intersection so that cars (transactions) can move smoothly without crashing (interfere with each other).

i) Locking:

Locking is one of the oldest methods to manage concurrency. Imagine youre working on a group project, and there's one document everyone needs to edit. To avoid confusion, one person (process) locks the document (gains exclusive access) while they are editing it, so no one else can make changes at the same time.

Types:

i) Read lock: Multiple people can read the document at the same time, but no one can make changes while it's being read.

ii) Write Lock: Only one person can make changes, and no one else can even read it while it's being edited.

⊛ Granularity of Locking:

This refers to how much of the document is locked. It could be the whole document, a section, or just a paragraph. If you lock only a small part (fine granularity), others can work on different parts simultaneously, but it might lead to more complex management.

⊛ Two-phase Locking:

Imagine you need several tools to complete a task. In first phase (growing phase), you gather all the tools you need. In the second phase (shrinking phase), you return them all at once. This method prevents inconsistencies or deadlocks (getting stuck waiting for tools) because you either have everything you need to proceed or you don't start at all.

⊛ Strict Two-Phase Locking:

Here, you hold onto the tools (locks) until you've completely finished your task, ensuring everything you did is based on the right information and no one else changed anything while you were working.

ii) Optimistic Concurrency Control :

This approach is like a teacher telling all students to write their essays at the same time without worrying about what others are doing. Once they finish, the teacher checks if anyone wrote about the same topic. If two students wrote on the same topic, one of them will to start over. This method is "optimistic" because it assumes conflicts will be rare, & most of the time, things will go smoothly.

Advantages :- It allows everyone to work simultaneously without waiting for each other, maximizing productivity.

Disadvantages :- Sometimes, a student might have to start over if their topic was already covered, which can be frustrating, especially if it happens often.

iii) Timestamping :

This method assigns each transaction a timestamp, like giving each student a number when they start working. The rule is that students with lower numbers (started earlier) get priority in submitting their work.

* Read & Write Timestamps :- Every file (or piece of information) has a record of when it was last read or written. If a student tries to submit their essay but another student with a lower number (started earlier) already changed the topic, they'll have to start over.

Why these methods matter ?

=> In distributed systems, where multiple processes are running on different machines, these methods help ensure that data remains consistent and reliable, even when many processes are accessing or modifying it simultaneously.

Each method has its strengths and weakness :

-> Locking is straightforward but can lead to deadlocks.

-> Optimistic Concurrency Control is flexible and deadlock-free but can result in some transactions being restarted.

-> Timestamping is simple & deadlock-free but may cause some transactions to abort unnecessarily if they're out of order.

Implementation of Atomic Transactions :

Two ways to restore old state :

1) Private Workspace
2) Write-ahead log

1) Private Workspace :

- When a process starts a transaction, it is given a private workspace containing all the objects including data & files.
- I/O operations are performed here.
- The data within the workspace is written back when the transaction commits.

Problem :- cost of copying every object is high

Solution :-

i) do not copy reads, only copy objects that are to be updated.
ii) Use indices to objects.
iii) When an object needs to be updated, a copy of the index and the objects are made in.
iv) The private index is updated.
v) On commit, the private index is copied to the parent's index (i.e. replacing pointers).

2) Write-ahead log :

- Before any changes are made, a record (log) is written to a write-ahead log on stable storage.
- The log contains the following information
  - (a) Which transaction is making the change
  - (b) Which object is being changes
  - (c) What the old & new values are
- Only after the log has been written successfully, the change is made to the object.
- If the transaction committed, a commit record is written to the log.
- If the transaction aborts, the log can be used to rollback to the original state (or even, recover from crashes).

Distributed Commit

Two-Phase Commit Protocol:

→ The action of commiting a transaction must be done instantaneously and indivisibly.
→ In a distributed system, the commit requires cooperation of multiple processes (which may be) on different machines.
→ Two-phase commit protocol is the most widely-used protocol to implement atomic commit.

Process:

- One process acts as the coordinator
- The other participating processes are subordinates ("cohorts").
- The coordinator writes an entry "prepare" in the log on a stable storage and sends the other processes involved (the subordinates) a message telling them to prepare.
- When a subordinate receives the message, it checks to see if it is ready to commit, makes a log entry, and sends back its decision.
- After collecting all the responses, if all the processes are ready to commit, the transaction is committed (otherwise, aborted).
- The coordinator writes a log entry & sends a message to each subordinate informing it of the decision.
- The coordinator's writing commit log actually means committing the transaction and no rolling back occurs no matter what happens afterward.

Deadlocks :

It occurs when a set of processes or resources are unable to proceed because each process is waiting for another to release a resource. This situation can lead to a standstill where no process can complete its execution. In distributed systems, deadlocks are ~~some~~ more complex due to factors such as lack of global state information and communication delays.

Deadlock occurs when the following 4 necessary conditions are simultaneously present :

i) Mutual Exclusion :- At least one resource must be held in a non-shareable mode, meaning only one process can use it at a time.

ii) ~~Hold~~ & Wait :- Processes already holding resources can request new resources while keeping the ones they already hold.

iii) No Preemption :- A resource cannot be forcibly taken away from a process, the process must release it voluntarily.

iv) Circular Wait :- A set of processes must exist in a circular ~~path~~ chain, where each process is waiting for a resource held by the next process in the chain.

Detection of deadlock :

1) Centralized approach :- A coordinator tracks resources and detects cycles, signaling deadlock.

2) Distributed approach :- Each node tracks resources and communicates to detect cycles collectively.

3) Hierarchical approach :- Nodes send data to higher levels for deadlock detection, reducing overhead.

Deadlock Prevention :

1) Avoid hold & wait :- Processes request all resources at once.
2) Preemption :- Resources can be taken from a process if needed.
3) Avoid Circular Wait :- Impose an ordering on resource requests.
4) Avoid mutual Exclusion :- Share resources when possible.

#Threads#

→ A thread is a lightweight unit of execution within a process. Each thread runs independently but shares the same memory space with other threads in the same process.

Real-time example :

- Imagine a restaurant (process) with three chefs (threads).
- The chefs work in the same kitchen (shared memory), but each focuses on preparing different parts of a meal. (separate execution).
- They can use the same ingredients and tools (memory & resources) but may perform their tasks at the same time.

Three processes with one thread :-

Each process has its own memory space and resources. These processes don't share memory, so they must communicate through mechanisms like message passing.

One process with three thread :-

Multiple threads within one process share the same memory space but run independently. They can easily communicate and share data.

Aspect	Process	Thread
Memory	- Separate memory space	- Shared memory space within a process.
Overhead	- Higher (context switching, memory)	- Lower (faster context switching)
Isolation	- Stronger isolation	- Less isolation
Creation	- Slower	- Faster
Communication	- Inter-Process Communication (IPC)	- Direct (shared memory),

Thread Usage :-

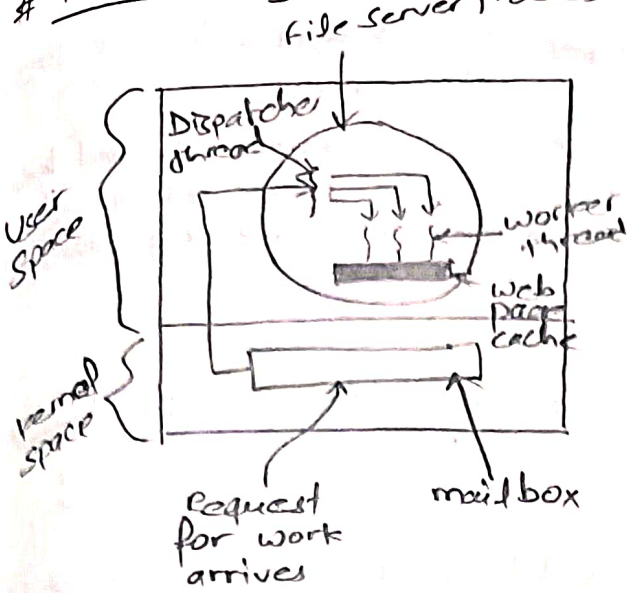


Fig:- Dispatcher Model

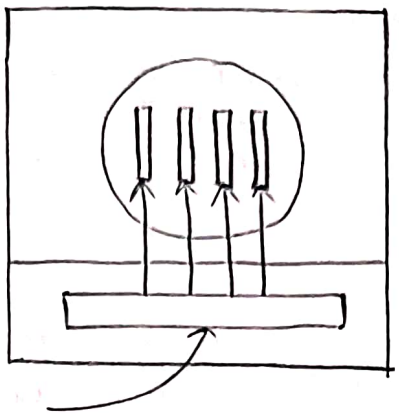


Fig:- Team model

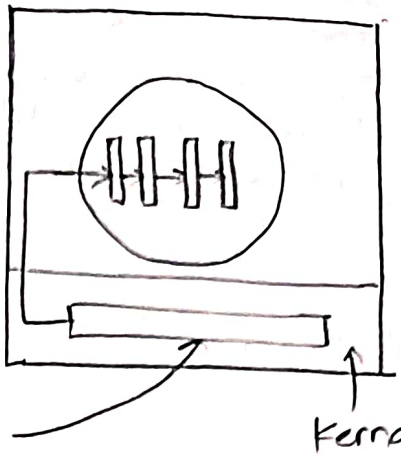


Fig:- Pipeline Model

1) **Dispatcher Model** :- A single dispatcher assigns tasks to worker threads.

2) **Team Model** :- A group of threads work together on the same task.

3) **Pipeline Model** :- Different stages of a task are handled by different threads in a sequence.

Implementing threads in User Space :-

- User space means that the OS is not directly involved in managing the threads. Instead, the threads are controlled by a user-level library. The OS only sees one process, & it doesn't know how many threads are running inside that process.
- Thread switching is very fast because the OS is not involved. The thread library manages everything.

Pros :

1) **Fast switching** :- Because the OS is not involved, threads can quickly switch between tasks.
2) **Custom scheduling** :- The user-level library can decide how to schedule threads based on specific needs.

Cons :

1) **No OS help** : If one thread is blocked, the entire process is blocked bcz the OS doesn't know that there are multiple threads inside.

#Implementing threads in Kernel Space :-

- In kernal space, the OS directly manages the threads. The OS knows about all the threads and controls how they run.
- Thread switching is slower because the OS is involved and has to manage each switch, but the OS can handle more advanced scheduling, like prioritizing important tasks or handling blocked threads.

Pros :

1) Better control :- The OS can manage all threads & make decisions like prioritizing important threads or helping blocked threads.
2) Handles blocking :- If one thread is blocked, the OS can still run other threads.

Cons :

1) Slower switching :- Involving the OS makes switching between threads slower than in user space.

#Difference between Thread & RPC :

Feature	Thread	Remote Procedure Call (RPC)
Execution	- Runs within the same process.	- Sends a request to another process (even on another machine)
Communication	- Shares memory with other threads.	- Sends messages over a network to communicate.
Overhead	- Low overhead (fast switching)	- Higher overhead because of message passing.

Notes :- Creating threads with RPC :

In an RPC system, when a message arrives, you can create a new thread to handle that message. This thread will take care of the task without affecting other running threads.

Implicit Receive :- An implicit receive means that the system waits for a message to arrive and automatically triggers an action when it does.

Pop-up Threads :- It is a new thread that is automatically created when a message arrives. It handles the task & disappears after finishing

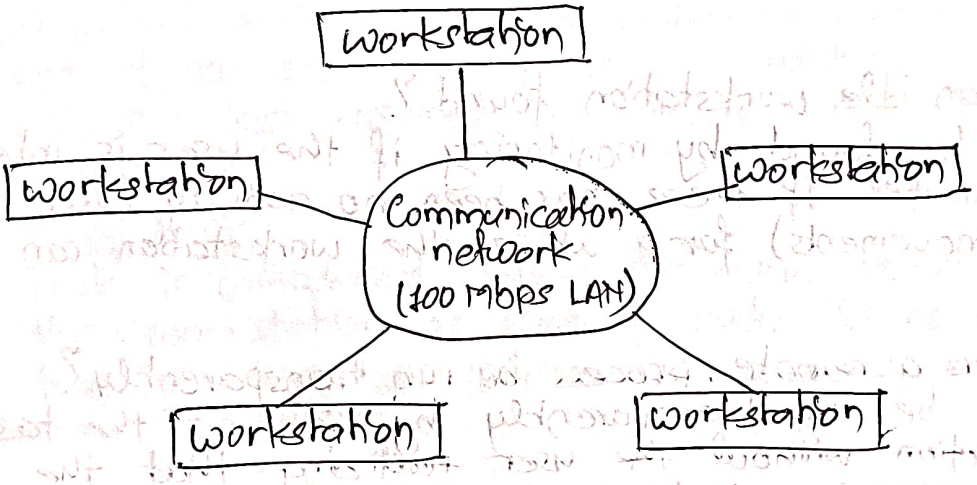
System Model:

It is a like a plan or design that shows how different parts of a computer system work together. It helps us understand the overall structure & function of the system, including how computers communicate, share resources, and handle tasks.

1) Workstation Model:

It refers to the way computers (workstations) are set up & used in a network. Each workstation is usually a personal computer connected to a network that allows users to perform tasks like editing documents or running programs.

In simple terms, think of a workstation as your own personal desk in an office. You can do your work on your own, but you are connected to the company's network to access files and resources shared by others.



Disk usage:

It refers how much data or what kind of files are stored locally (on the workstation's hard drive) or accessed from a central server.

Types:

i) Diskless workstation:- The workstation has no disk of its own. It relies completely on the network to access programs and files.

Adv: Easier to manage because all data is on the server.
Disadv: It can't do much without a network connection.

ii) Paging & Scratch files only:- The workstation uses some disk space, but only for temporary data (paging or scratch files). All other files are stored on the server.

Adv: It saves network bandwidth for temporary data.
Disadv: Permanent files are still dependent on the network.

iii) Paging, Scratch files, & Binaries:
The workstation stores some programs (binaries) on the disk but accesses user files from the network.

iv) Full Local file System:
The workstation has its own complete file system. It stores both programs and user data locally.

2) ~~Workstation Server Model~~ / An Idle Workstation:
It is a computer that is not being used by its owner. When the owner is away, the computer is considered "idle". Using an idle workstation means using the unused processing power of that workstation to run tasks for other users or systems.

Qn) How is an idle workstation found?
⇒ It can be found by monitoring if the user is interacting with the computer. If there has been no activity (like typing or mouse movements) for a while, the workstation can be considered idle.

Qn) How can a remote process be run transparently?
⇒ It can be run transparently by sending the task to an idle workstation without the user realizing that the process is not running on their own machines. This is usually managed by the system, so users don't need to worry about where the process is running.

Qn) What happens if the machine's owner comes back?
⇒ If the owner returns and starts using the workstation, the system will stop using the workstation for remote tasks. The remote tasks might either be moved to another idle workstation or paused.

* Algorithms for finding & using idle workstations:

Steps:

1) Idle detection:
- The system monitors workstation activity (keyboard, mouse, CPU, etc)
- A timer starts once no activity is detected, and if the workstation remains inactive beyond the threshold (e.g. 10 minutes), it is flagged as idle.

2) Task assignment:
- The scheduler looks for idle workstations & assigns pending tasks to them.
- The system checks the resource needs of each task & matches them to the capabilities of the idle workstation.

3) Task execution:
- The idle workstation starts executing the task in the background.
- The system periodically checks the workstation to ensure that the task is running smoothly.

4) Owner detection (Resumption of Activity):
- If the workstation's owner returns and begins using the machine, the system detects this through input events (eg, mouse movement, etc).
- The system immediately preempts the background task, freeing up resources for the owner.

5) Task migration or Pausing:
- The background task is either migrated to another idle workstation or paused if no other workstation is available.
- If paused, the task resumes as soon as a suitable idle workstation becomes available.

6) Completion of task:
- If the task is completed before the owner returns, the system marks the workstation as free & ready for normal use.
- If the task is paused or migrated, it continues on another machine until finished.

Advantages:
1) Better resource utilization :- Makes use of otherwise unused machine
2) Cost effective :- Reduces the need for additional hardware as idle machines are used for heavy tasks.
3) Scalable :- The system can grow with more workstations being added to the network.

Disadvantages:
1) Complexity in the migration :- Moving tasks from one machine to another can be ~~tricky~~ tricky & time-consuming.
2) Owner disruption :- If the system isn't efficient in stopping background tasks quickly, it can cause minor disruptions to the workstation's owner.

Note :- (Imp)
- The remote based algorithm for using idle workstations allows distributed systems to make use of unused computers by running background tasks on them.

Q) What do you mean by home workstation?

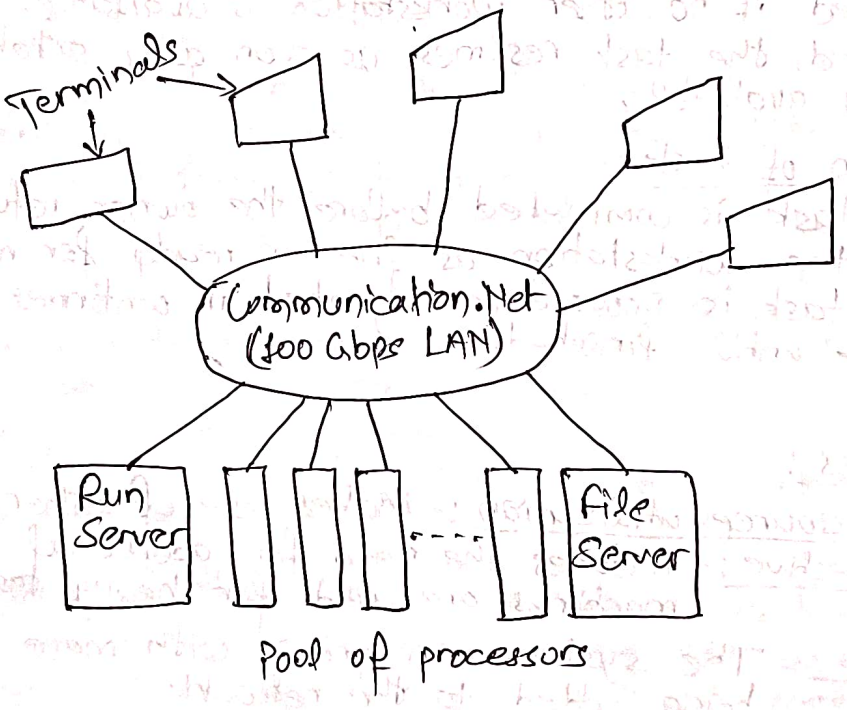
⇒ A home workstation is the primary computer a user works on in a distributed system.

Example:- Your desktop computer at home that you are daily is your "home workstation".

3) The Processor Pool Model:

In the processor pool model, instead of giving each user their own workstation, a central pool of powerful processors (or computers) is shared among all users. When a user needs to run a task, they can borrow processing power from the pool.

Example: Imagine a classroom with a pool of computers that all students can use when needed, instead of each student having their own dedicated computer.



Key concepts:

1) Central Pool of processors:
- In this model, instead of assigning each user or task a dedicated processor, the system maintains a central pool of processors.
- These processors are not tied to a specific workstation or user; instead, they are shared resources that can be allocated to tasks dynamically based on demand.

2) Dynamic Allocation:
- As tasks come in, the system dynamically allocates processors from the pool to handle them. Once the task is completed, the processors are returned to the pool to be reused by other tasks.

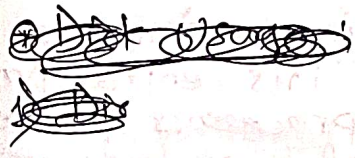
3) Scalability:
- The processor pool model is highly scalable. As more tasks are introduced or more users require processing power, additional processors can be added to the pool to meet demand.
- This model is especially beneficial in environments where workloads are unpredictable or bursty, as it allows for flexibility in handling varying levels of demand.

4) Load balancing:
- The system is responsible for distributing tasks among the available processors in the pool in a balanced way. This ensures that no single processors is overloaded while others remain idle.
- Load balancing techniques help maximize system performance and minimize task completion time.

5) Cost efficiency:
- By pooling processors, organizations can reduce costs because they do not need to allocate a dedicated processor to each task or user.
- Instead, resources are shared, which leads to more efficient use of hardware & potentially lower power consumption & cooling requirements.

Advantages:
1) Efficient Resource Utilization
2) Scalability
3) Cost-effective

Disadvantages:
1) Complex Scheduling
2) Resource contention
3) Overhead.



Processor Allocation Algorithms

Processor allocation involves deciding which computer should run a specific task or process. The goal is to balance the workload so that no single computer is overloaded, and all computers are utilized efficiently.

Allocation models determine how processes are assigned to processors. There are 2 main types:

1) Migratory Allocation Algorithm:-
In this model, a process can move from one processor to another after it has started. This is useful when one processor becomes overloaded, and another is free.

2) Non-Migratory Allocation Algorithm:
In this model, once a process is assigned to a processor, it stays there until it is completed, regardless of other factors.

Notes:

1) Response time :- This is the time it takes for the system to start responding to a request after it is made.

2) CPU utilization :- This measures how much of the processor's capacity is being used.

3) Response ratio :- This is the ratio of the actual time taken for a process to complete to the expected time.

Imp

Algorithms :

1) Centralized Algorithms :

How it works?

⇒ In this model, a single central controller (processor) makes all the decisions about where tasks should go. This central processor monitors the workload of all other processors & assigns tasks to them. No other processor has control over task assignment.

Adv:
- Easy to manage because there is one central authority making all decisions.
- Ensures balance across processors because the central controller can assign tasks evenly.

Disadv:
- The central controller can become a bottleneck if too many tasks come in at once, slowing down the whole system.
- If the central controller fails, the entire system collapses.

A single processor or system controls all task assignments.

How it works?

→ In this model, a single ce

2) Graph Deterministic Algorithm:

In this algorithm, all processors are connected in a fixed, structured graph. Each processor (or node in the graph) knows exactly which other processor to pass tasks to, based on predetermined rules. The graph could be a ring, a tree, or any other structured shape. The task movement is based on the graph's design, and there is no randomness involved.

Advantages:
- It is predictable because the path for each task is pre-decided.
- The system doesn't require communication between processors to decide who gets the task.

Disadvantages:
- It is inflexible. If one processor is overloaded, it can't pass tasks to another processor outside of its fixed path.
- If a processor fails, the entire system could break down unless backups are in place.

3) Hierarchical Algorithm:

This algorithm is organized in levels, like a hierarchy. At the top of the hierarchy, one or more processors are responsible for managing the others. These top-level processors can delegate tasks to lower-level processors. As tasks move down the hierarchy, each level of processors can delegate work to the level below them.

Advantages:
- It creates a clear chain of command, which can reduce confusion.
- Higher-level processors can delegate tasks, reducing their workload.

Disadvantages:
- If a high-level processor fails, the processors below it may not receive tasks.
- It can create delays as tasks must go through multiple levels before they are assigned.

4) Sender-Initiated Distributed Algorithm:

In this algorithm, the processor that is overloaded (the sender) takes the initiative to transfer tasks. When a processor has too many tasks to handle, it looks for other processors with less work and sends some tasks to them.

Advantages:
- It helps to reduce the load on overloaded processors by distributing tasks.
- It makes efficient use of all processors in the system.

Disadvantages:
- If all processors are busy, it might take time to find someone to offload tasks to.
- Sending tasks between processors requires communication, which can add overhead (extra work).

5) Receiver-Initiated Distributed Heuristic Algorithm:

In this algorithm, the idle processors (receivers) take the initiative. When a processor is idle and has no tasks to work on, it reaches out to other busy processors to request work. The idle processor tries to "steal" tasks from overloaded processors.

Advantages:
- It ensures that idle processors are quickly put to work, improving overall system efficiency.
- It balances the workload effectively by pulling tasks from overloaded processors.

Disadvantages:
- If there are too many idle processors, they might all compete for tasks, leading to inefficiently.
- Like sender-initiated algorithms, this also requires communication, which can increase system overhead.

6) Bidding Algorithm:

In this algorithm, processors complete for tasks by placing bids. A task is announced, and each processor bids based on its ability to complete the task efficiently. The processor that places the best bid gets the task. The bid can be based on factors like current load, expected response time, or available resources.

Advantages:
- It promotes competition, leading to better task allocation.
- It ensures that tasks go to the processors that can complete them most efficiently.

Disadvantages:
- The binding process takes time, which can delay task assignments.
- If too many processors are bidding, it can create confusion or slow down the system.

* Key differences between the Algorithms:

1) Graph Deterministic :- It is rigid & follows a fixed structure, whil others are more flexible.

2) Centralized :- It has one authority assigning tasks, while distributed algorithms (sender-initiated, receiver-initiated, bidding) allow processors to share tasks based on their load.

3) Hierarchical :- Adds multiple layers of delegation, which can improve management but may slow down task assignment.

4) Bidding :- Allows processors to complete for tasks, adding a layer of decision-making.

* Each of these algorithms has different strengths & weakness depending on the needs of the system, like how dynamic the workload is and how much communication overhead is acceptable.

Scheduling:

It is the process of deciding which task or process should run on which processor, and when it should run. The goal is to manage the workload efficiently across multiple processors or machines, so no one processor is overloaded while others remain idle.

In distributed systems, scheduling becomes more complex because
- There are multiple processors spread across different locations
- Communication between processors takes time, so coordination is required.
- Tasks or processes may need to move between processors, adding to the complexity.

Types:

1) **Local Scheduling:-**
Each processor schedules its own tasks independently, without considering what other processors are doing.

2) **Global Scheduling:-**
A central system is responsible for scheduling tasks across all processors. It has an overview of the entire system & can decide which processor should handle which task.

3) **Distributed Scheduling:-**
Processors work together to make scheduling decisions. They share information about their workload & cooperate to balance the tasks.

Key Challenges in Scheduling:

1) **Load balancing:-** Ensuring that no processor is overloaded while others are idle.

2) **Communication Overhead:-** Processors need to share information about their workloads, but too much communication can slow down the system.

3) **Fault tolerance:-** If one processor fails, the system needs to reassign its tasks without disrupting overall performance.

⊛ Coscheduling:

It is a scheduling strategy where a group of related processes that communicate with each other are scheduling to run at the same time on different processors. This is important because these processes need to exchange data frequently, and if they are not running at the same time, one process may be left waiting for the others, leading to inefficiencies.

Replication:

Replication in distributed systems involves making redundant copies of resources, such as data or process, while ensuring that all the copies are identical, to improve reliability, fault-tolerance & performance of the system.

Why replicate?

=> Assuming a simple model in which we make a copy of a specific part of a system (meaning code & data).

i) **Increase reliability:-**
If one copy does not live up to specifications, switch over to the other copy while repairing the failing one.

ii) **Performance:-**
Simply spread requests between different replicated parts to keep load balanced, or to ensure quick responses by taking proximity into account.

Problem:
Having multiple copies, means that when any copy changes, that change should be made at all copies: replicas need to be kept the same, that is, be kept consistent.

⊛ Performance & Scalability:

Main issue:- To keep replicas consistent, we generally need to ensure that all conflicting operations are done in the same order everywhere.

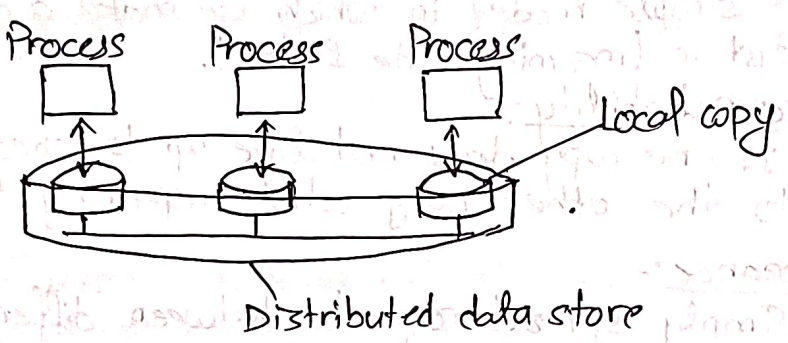
Conflicting operations:
i) **read-write conflict:** a read operation & a write operation act concurrently.
ii) **write-write conflict:** two concurrent write operations.

Data-Centric Consistency Models:

The data-centric consistency model focuses on the system's perspective, ensuring that all replicas of data across the distributed system remain consistent. It guarantees that every user, regardless of which node they access, sees the same data. This model includes several levels of consistency, such as strict, sequential, and eventual consistency.

Essential:

A data store is a distributed collection of storages:



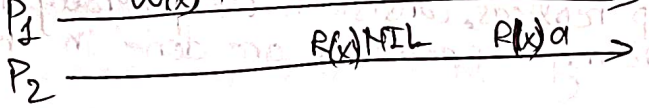
Some notations

Read and write operations:

- $W_i(x)a$: Process P_i write value a to x.
- $R_i(x)b$: Process P_i reads value b from x.
- All data items initially have value NIL.

Possible behavior:

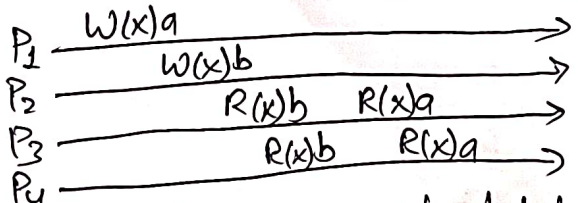
We omit the index when passible & draw according to time(x-axis)



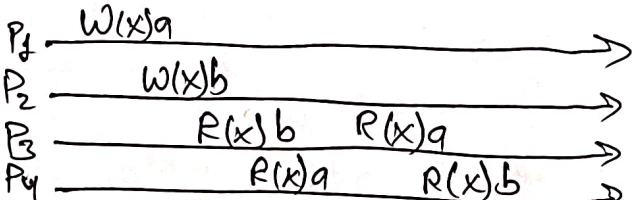
Types:

1) Sequential Consistency:

The result of any execution is the same as if the operations of all processes were executed in some sequential order, and the operations of each individual process appear in this sequence in the order specified by its program.



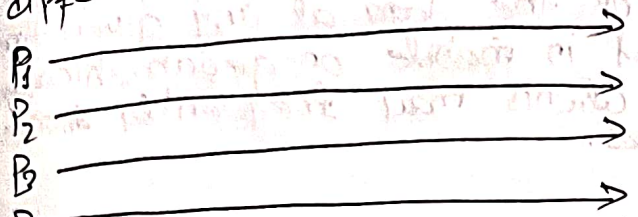
A sequentially consistent data store



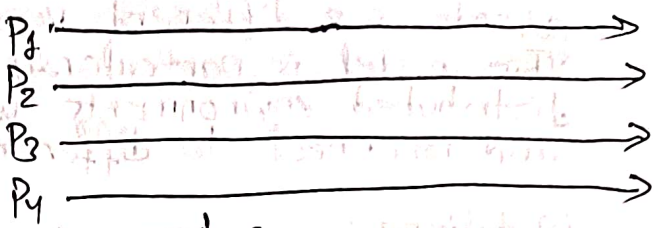
A data store that is not sequentially consistent

ii) Casual Consistency:

Writes that are potentially casually related must be seen by all processes in the same order. Concurrent writes may be seen in a different order by different processes.



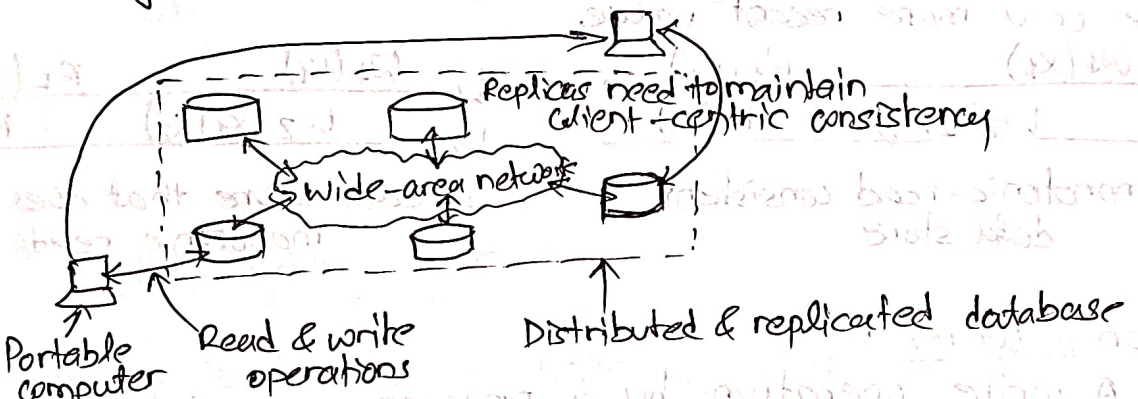
A violation of a casually-consistent store



A correct s

iii) Eventual Consistency:

It is the most relaxed form, where updates to replicas are not immediately visible. Instead, the system ensures that all nodes eventually converge to the same data state. This model is commonly used in large-scale, distributed systems like cloud services, where availability & partition tolerance are prioritized.



Client-centric consistency:

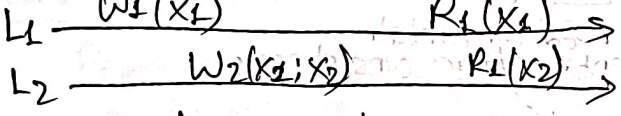
The client-centric consistency model, on the other hand, focuses on the user or client's experience. It aims to provide a consistent view of data for individual clients, even if different clients see different versions of the data at any given time. This model is particularly useful in mobile or geographically distributed environments where clients may frequently disconnect and reconnect to different nodes.

Notations:

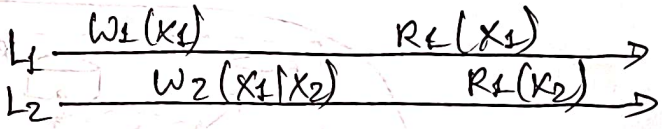
- $W_i(x_2)$ is the write operation by process P_i that leads to version x_2 of
- $W_i(x_i:x_j)$ indicates P_i produces version x_j based on a previous version x
- $W_i(x_i|x_j)$ indicates P_i produces version x_j concurrently to version x_i.

i) Monotonic reads:

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 or a more recent value.



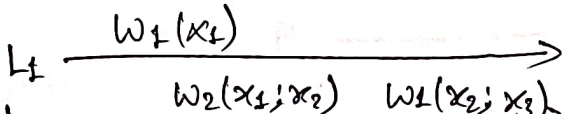
A monotonic-read consistent data store



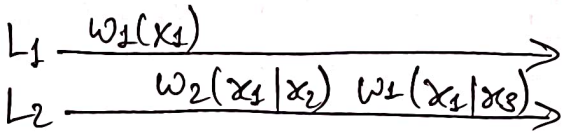
A data store that does not provide monotonic reads

ii) Monotonic writes:

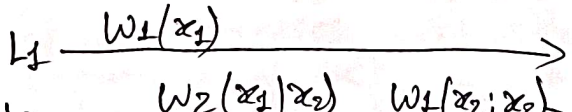
A write operation by a process on a data item x is completed before any successive write operation on x by the same process.



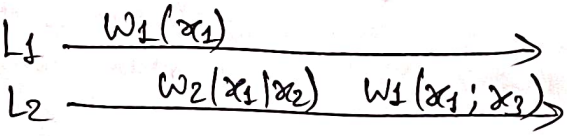
OK



Not OK

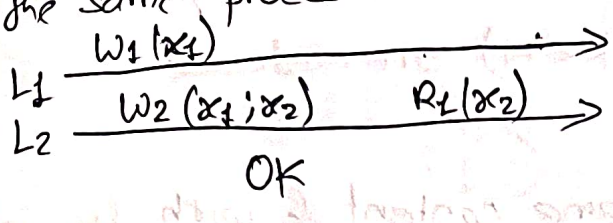


Not OK

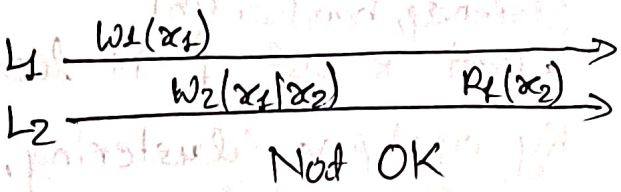


OK

Read your writes:
The effect of a write operation by a process on a data item x, will always be seen by a successive read operation on x by the same process.

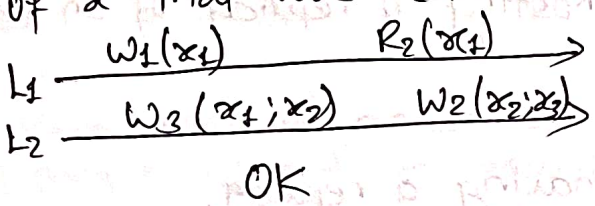


OK

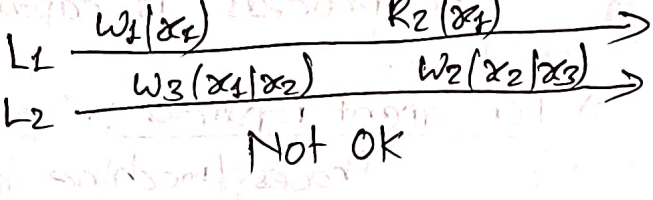


Not OK

Writes follow reads:
A write operation by a process on a data item x following a previous read operation on x by the same process, is guaranteed to take place on the same or a more recent value of x that was read.



OK



Not OK

Replica Management

Replication:- is having multiple copies of data & services in a distributed system.

Reasons:-
-> Reliability of the system
-> Better protection against corrupted data
-> Improved performance & faster response time
-> Facilitates scaling in numbers & geographical area.

Key issues:
-> Where, when & by whom replicas should be placed.
-> Mechanisms to keep them consistent.
-> Two main sub-problems:
1) Replica-server Placement
   -> Finding best location or placed where a server can beplaced
2) Content Placement
   -> Finding out which server is best for storing a particular content.

1) Replica-Server Placement :

- Based on distance between clients & locations as starting point. (latency, bandwidth)
  - Best K out of N locations (K<N) are selected.
- By applying clustering,
  - Group nodes accessing the same content & with low inter-nodal latencies into groups or clusters & place a replica on the k largest clusters.

2) Content Replication & Placement :

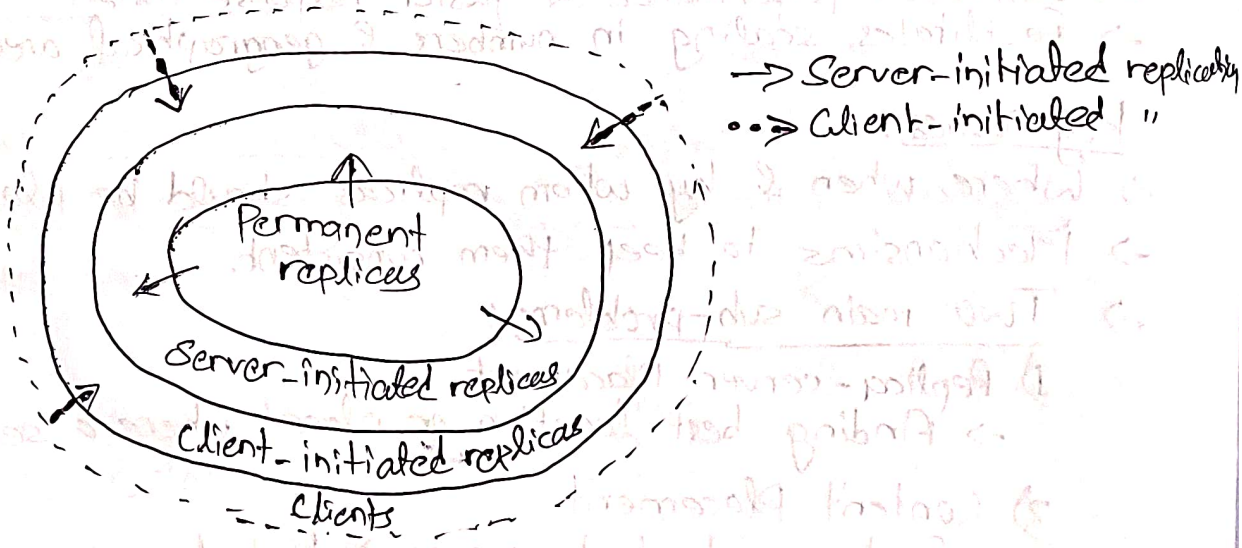
A process is capable of hosting a replica of an object or data.

i) Permanent replicas :-
Process/machine always having a replica.

ii) Server-initiated replica :-
Process that can dynamically host a replica on request of another server in the data store.

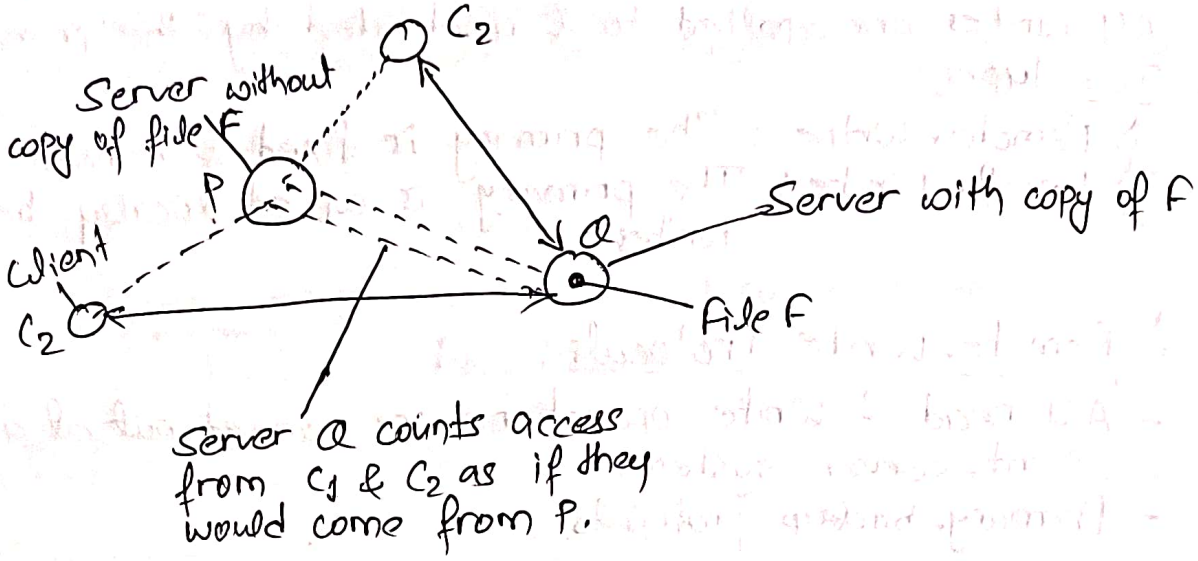
iii) Client-initiated replica :-
Process that can dynamically host a replica on request of a client (client cache).

The logical organization of different kinds of copies of a data store into three concentric rings :



ii) Server-initiated replicas:

Counting access requests from different clients



- Keep track of access counts per file, aggregated by considering server closest to requesting clients.
- Number of accesses drops below threshold $D \Rightarrow$ drop file.
- Number of accesses exceed threshold $R \Rightarrow$ replicate file.
- Number of access between D & $R \Rightarrow$ migrate file.

iii) Client Initiated Replicas:

- client caches
- Managing is entirely by client
- Improve access time
- Placement
  - same machine
  - LAN
  - WAN

Content Distribution:

Consider only a client-server combination

i) Propagate only notification/invalidation of an update (often used for cache)

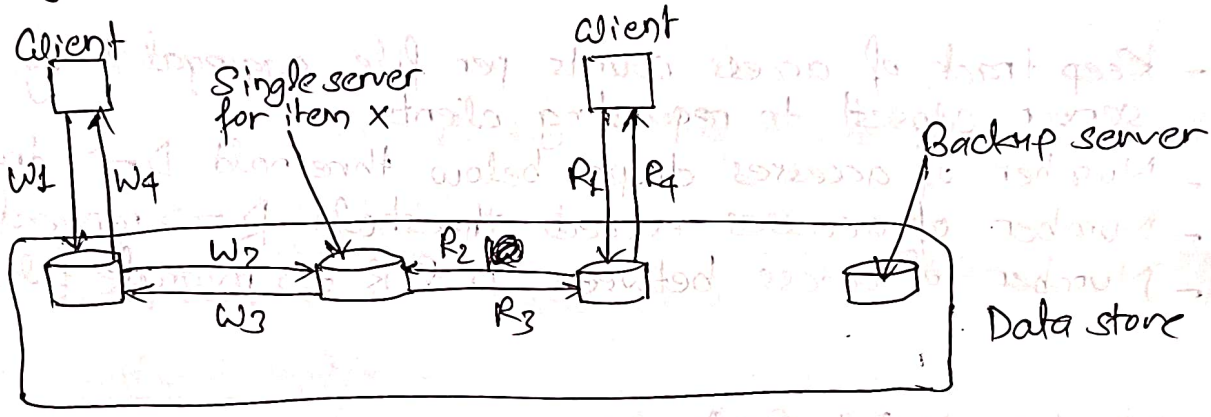
ii) Transfer data from one copy to another (distributed databases): passive replication

iii) Propagate the update operation to other copies: active replication

Consistency Protocols

1) Primary-based protocols :- (each data item has a primary replica)
- All writes are applied to & coordinated by the primary.
- Two types :
  i) Remote-Write : The primary is fixed & remote
  ii) Local-Write : The primary is copied locally before applying writes.

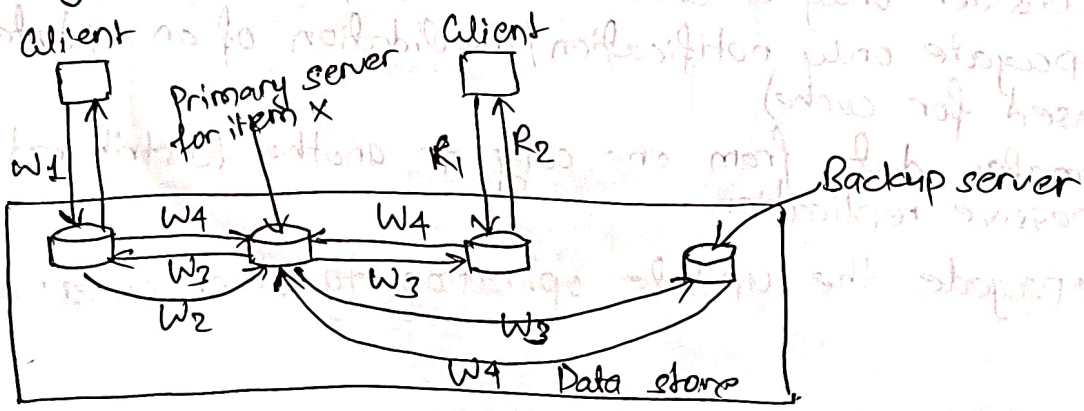
i) Remote-Write Protocols :
- All read & write operations are carried out at a remote server.
- Client-server systems
- Primary-backup protocols



W1 : Write request
W2 : Forward request to server for x
W3 : Acknowledge write completed
W4 : Acknowledge write completed

R1 : Read request
R2 : Forward request to server for x
R3 : Return response
R4 : Return response

Imp Primary-backup protocol :

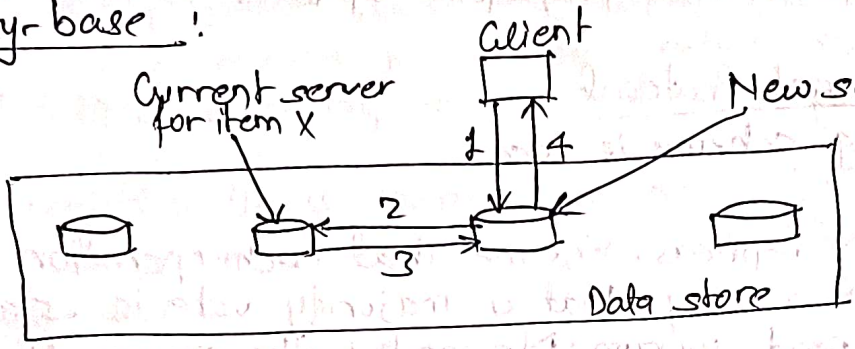


Example :- Traditionally applied in distributed databases & file syst that require a high degree of fault tolerance. Replicas are often placed on the same LAN.

Local-write Protocols :

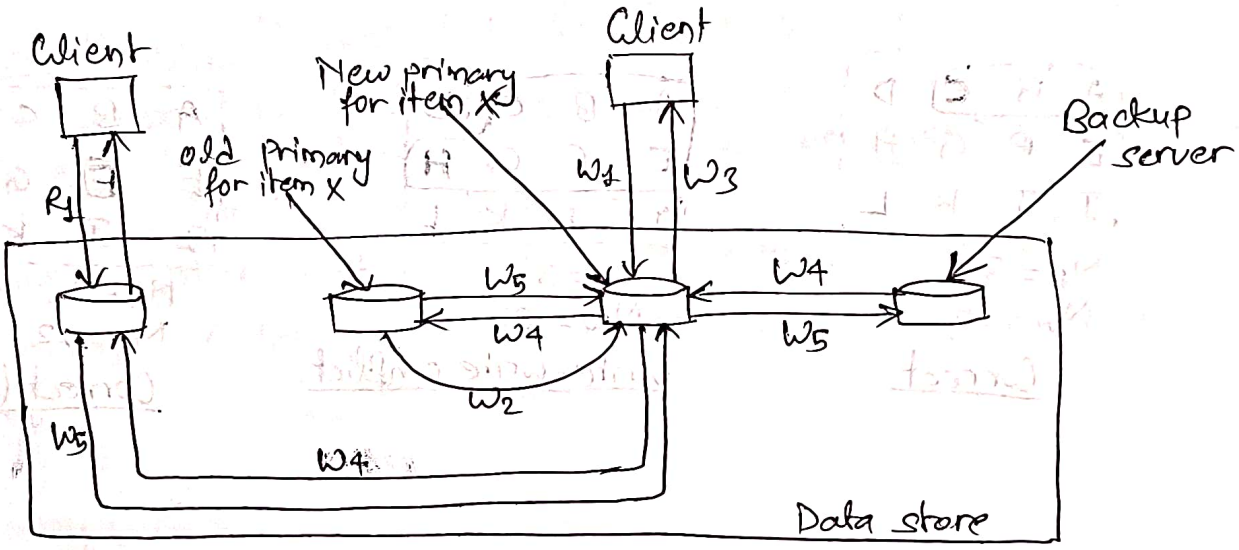
- Primary-base local-write protocols :- only a single copy of data item X.
- Primary-backup local-write protocols :-
  - multiple copies of each data item x exist, but one of them are primary.
  - can disconnected mode.

a) Primary-base :



IMP

b) Primary-backup with local-write :



Example :- Mobile computing in disconnected mode (ship all relevant files to user before disconnecting, & update later on).

2) Replication-based protocols:

- Operations can be carried out at multiple replicas.
- Two types:

i) Active Replication Protocol:

→ Clients write at any replica (no primary replicas)
→ The replica will propagate updates to other replicas.

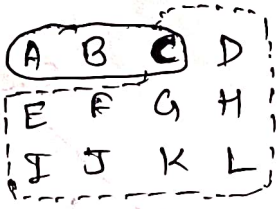
IMP

ii) Quorum-Based Protocol:

→ A voting scheme is used.

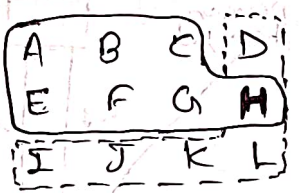
Assume N replicas. Ensure that each operation is carried out in such a way that a majority vote is established; distinguish read quorum N_R and write quorum N_W. Ensure;

1) $N_R + N_W > N$ (prevent read-write conflicts)
2) $N_W > N/2$ (prevent write-write conflicts)



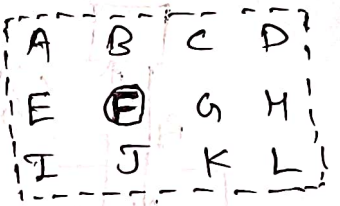
$N_R = 3$
$N_W = 10$

Correct



$N_R = 3$
$N_W = 10$

Write-write conflict



$N_R = 1$
$N_W = 12$

Correct (ROWA)

Fault Tolerence:

It is the capability of a system to deliver uninterrupted service despite one or more of its components failing.

System failures:

1) Fail-Silent Fault :- When a system experiences a fail-silent fault, it stops working silently but does not produce incorrect results or mislead other systems. It either functions correctly or stops functioning altogether.

2) Byzantine faults :- It is more complicated because the system or component doesn't just fail silently. Instead, it behaves unpredictably. It could give wrong answers, provide partial data, or cause confusion for the rest of the system. Byzantine faults are difficult to detect because the faulty system might still appear to be running normally to others.

Component Faults : (Types of fault tolerence):

The components that can fail include:

i) Hardware :- Identical or equivalent backup systems can replace failed hardware.

ii) Software :- Other software instances can backup software systems.

iii) Power sources :- Alternative sources can support power sources.

Redundancy :

It involves having multiple copies of the same data or processes in case of failure. By duplicating critical parts of a system, it ensures that even if one part fails, the other can take over.

It can be implemented in no. of ways, including:

i) Hardware redundancy
ii) Software redundancy
iii) Data redundancy
iv) Alternative computer system :- having an entire alternate computer system in place in case a failure occurs.

⊛ Active replication:

In active replication, more than one replica of the server executes an RPC. If any of the replicas fails, the surviving replicas adopt the orphan.

Fault tolerance using active replication:

1) Triple Modular Redundancy (TMR):

TMR is a fault-tolerance mechanism where three identical systems process the same tasks, and a majority vote is taken. If one system fails, the other two can outvote the faulty one & provide the correct result.

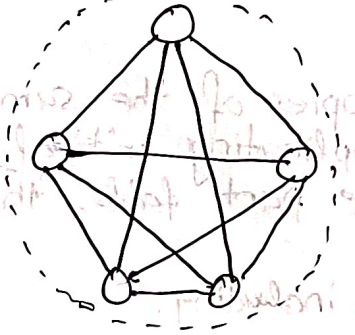
2) State Machine Approach:

The state machine approach assumes that every process in a system behaves like a finite state machine, meaning it follows a set of rules to move between states. Each replica of a process starts from the same state & processes the same sequence of events.

⊛ Process Resilience:

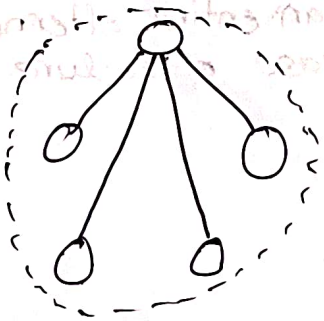
Protect against malfunctioning process through process replication, organizing multiple processes into a process group.

i) Flat groups:



Communication in a flat group:-
All the processes are equal, decisions are made collectively.
No single point-of-failure, however: decision making is complicated as consensus is required.

ii) Hierarchical groups:



Communication :- One of the processes is elected to be the coordinator, which selects another process (a worker) to perform the operation.
Single point-of-failure, however: decision making can be easily happen by coordinator without first having to get consensus.

Distributed Commit Protocols :

Goal :
- We want an operation to be performed by all group members or none at all.
- In the case of atomic multicasting, the operation is the delivery of the message.

Three types :

1) Single-phase Commit Protocol :-

- An elected co-ordinator tells all the other processes to perform the operation in question.
- But, what if a process cannot perform the operation? There's no way to tell the coordinator!

Solution :
- Two-Phase & Three-Phase Commit.

2) Two-phase Commit Protocol :-

Summarized : (GET READY, OK, GO AHEAD)

i) The coordinator sends a VOTE_REQUEST message to all group members.

ii) The group member returns VOTE_COMMIT if it can commit locally, otherwise VOTE_ABORT.

iii) All votes are collected by the coordinator. A GLOBAL_COMMIT is sent if all the group members voted to commit. If one group member voted to abort, a GLOBAL_ABORT is sent.

iv) The group members then COMMIT or ABORT based on the last message received from the coordinator.

Finite State Machines :

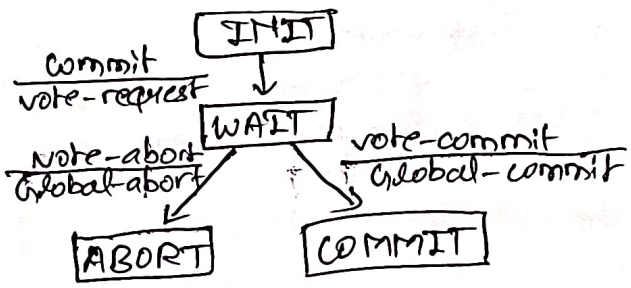


Fig :ⓐ Finite state machine for the coordinator.

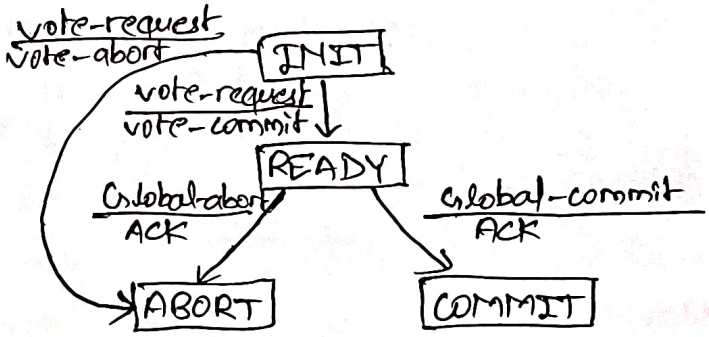


Fig :ⓑ Finite state machine for a participant (group member)

Big problem with 2PC :

- It can lead to both the coordinator and the group members blocking, which may lead to the dreaded deadlock.
- If the coordinator crashes, the group members may not be able to reach a final decision, and they may, therefore, block until the coordinator recovers.

Therefore, it is known as blocking-commit protocol.

Solution: - Three-Phase Commit Protocol

Recovery Strategies:

Once a failure has occured, it is essential that the process where the failure happened recovers to a correct state.

- Recovery from an error is fundamental to fault tolerance.
- Two main forms of recovery:
  i) Backward recovery
  ii) Forward recovery

⊛ Reliable Client/Server Communication:

- In addition to process failures, a communication channel may exhibit crash, omission, timing, and/or arbitrary failures.
- In practice, the focus is on making crash and omission failures.

Shared Memory

#) **Distributed Shared Memory** :- It is a concept where physically separate computers (nodes) share memory in a way that it looks like a single shared memory to the programs running on those nodes.

Shared Memory :- It is a memory space that multiple processors can access simultaneously. In a shared memory system, all processors in a multiprocessor system can read and write to a common memory space, which allows them to communicate or share data easily.

Architectures :

1) **On-chip memory** :
It is also known as fast memory, on-chip memory is located on a chip and has a small capacity of several megabytes. It is temporarily store commonly used code and data that is copied from the main memory.

2) **Hypothetical shared memory multiprocessor** :
It is an architecture that allows any processor to access data created or used by any other processor.

3) **Bus-based multiprocessors** :
It is a type of system where multiple processors are connected to the same memory through a single communication pathway called a bus. The bus allows the processors to send and receive data to and from the shared memory. Since there is only one bus, only one processor can use it at a time, which can cause delays if multiple processors want to access memory simultaneously.

4) **Ring-based multiprocessors** :
In ring-based multiprocessors, processors are connected in a ring-like structure. Data is passed around the ring until it reaches the intended processor.

Concepts :

i) **Data flow** :- Each processor is connected to two —

neighbours, and messages are passed from one processor to another around the ring.

ii) Communications :- If a processor wants to communicate with another, the message travels around the ring until it reaches the destination.

5) Switched multiprocessors :

In switched multiprocessors, processors are connected using a switch, allowing them to communicate with each other directly through the switch rather than passing messages through a ring.

Directories :

In the context of multiprocessor systems, a directory is a table that keeps track of which processors have cached copies of memory blocks. It helps manage data consistency across caches by knowing who has what data.

Caching :

Caching is a technique where frequently used data is stored in a small, fast memory (cache) so it can be accessed quickly instead of fetching it from the main memory every time.

1) Uncached :- Data is fetched directly from memory everytime.

2) Clean :- The cached data is the same as in the main memory.

3) Dirty :- The cached data has been modified but not yet written back to the main memory.

What is DASH Machine ?

=> A DASH (Directory Architecture for Shared Memory) machine is a multiprocessor system that uses directories to maintain cache coherence in a distributed shared memory system. It allows multiple processors to share memory effecientfly while keeping track of data ownership and consistency through a directory.

⊛ DASH Architecture :

The DASH architecture connects multiple processors using directories to manage cache coherence. Each processor has its local cache, and the directory ensures that data stays consistent across all caches.

How it works ?

Processor :- A processing unit that works on tasks.

Cache :- Stores frequently accessed data to reduce the need to fetch it from memory.

Directory :- keeps track of which cache holds the most recent copy of a memory block.

When one processor changes data, the directory ensures all other caches with copies of that data are updated or invalidated.

⊛ Protocols of DASH :

The DASH machine uses cache coherence protocols to ensure that the data in different caches remains consistent. These protocols handle operations like :

1) Read Miss :-
When a processor requests data that isn't in the cache, it fetches it from the main memory or another cache.

2) Write Miss :
When a processor wants to write to data that isn't in its cache, it fetches the data, modifies it, and then writes it back.

3) Write Hit :-
If the data is already in the cache, the processor can update it directly.

NUMA Multiprocessors :-

NUMA (Non-Uniform Memory Access) multiprocessors are systems where the memory access time depends on the memory location relative to the processor. In simpler terms, processors can access some parts of the memory faster than others.

In contrast to Uniform Memory Access (UMA), where all processors have the same speed of access to memory, in a NUMA system:

- Local memory (memory attached to a processor) is faster to access.
- Remote memory (memory attached to another processor) is slower to access.

This architecture is designed to improve scalability by reducing the bottleneck that occurs in shared-memory systems.

Properties :-

i) Distributed Memory :- Each processor has its own local memory, but can still access other processor's memory.

ii) Faster Local Access :- Accessing local memory is faster compared to accessing memory on another processor (remote memory).

iii) Scalability :- NUMA systems are scalable, meaning as you add more processors, the system can handle larger workloads efficiently.

iv) Non-Uniform Latency :- Since access to remote memory is slower than local memory, latency is non-uniform.

v) Memory Coherence :- NUMA systems typically maintain memory coherence through specialized protocols.

NUMA algorithms:

NUMA algorithm optimize how data is placed and accessed in memory to reduce the cost of accessing remote memory.

i) **Data Affinity Algorithms:-**
These ensures that data is stored close to the processor that uses it most frequently, reducing remote memory access.

ii) **Load Balancing Algorithms:-**
These balance workloads between processors to avoid overloading one processor and increasing remote memory accesses.

Consistency Models:

Consistency models describe the rules that a distributed system follows to ensure that data reads and writes behave as expected.

1) **Strict Consistency:**
Strict consistency is the strongest consistency model. It guarantees that:
- Any read operation on a memory location will return the most recent write to that location, regardless of which process performed the write.
- In other words, every process sees updates to the memory immediately.

2) **Sequential Consistency:-**
Sequential consistency guarantees that:
- The result of executing operations (reads and writes) is as if all processes executed their operations in some sequential order.
- Each process's operations are executed in the order they were issued.

Q7) Why is Sequential Consistency weaker than Strict Consistency?

=> Strict consistency requires immediate visibility of writes across all processes, while sequential consistency only requires that all operations appear to be performed in some order, but not necessarily immediately.

3) Casual Consistency:-

Casual consistency ensures that causally related operations are seen by all processes in the same order, but unrelated operations (those without a casual relationship) may be seen in different orders.

4) PRAM Consistency:-

Pipelined Random Access Memory (PRAM) consistency (or processor consistency) ensures that writes made by a single process are seen in the order they were issued, but writes from different processes may be seen in different orders.

5) Processor Consistency:

It is slightly weaker version of PRAM consistency where the order of writes from each processor is maintained, but processors do not have to agree on the order of writes from other processors.

6) Weak Consistency:

It allows processes to see updates only at specific synchronization points, instead of immediately after every write.

It requires that:

i) All previous writes must complete before any synchronization event.

ii) Synchronization ensures visibility of updates to all process

7) Release Consistency :

It is a form of weak consistency where synchronization occurs through acquire and release operations.

Acquire :- Happens before accessing shared data, ensuring all previous writes are visible.

Release :- Happens after updating shared data, ensuring that updates are visible to other processes.

Protected :- Ensures that the acquire and release protect access to critical regions of memory.

8) Entry Consistency :

Synchronization is done on a per-variable basis.

Page-Based Distributed Shared Memory (DSM) :-

It is a system where the memory is shared across multiple machines (nodes) in a distributed system, and the memory is managed in terms of pages (fixed-sized blocks, usually 4 kB). The idea is to simulate shared memory across nodes, but instead of direct access, pages of memory are transferred between nodes based on access patterns.

Shared Variable Distributed Shared Memory :

In shared variable DSM, memory is shared among multiple machines (nodes) in a distributed system, and specific variables are identified as shared across these machines. Unlike page-based DSM, which handles memory in large chunks (pages), shared variable DSM focuses on specific variables that are shared between nodes. This allows for more precise communication, as only relevant variables are shared and synchronized between machines.

Examples of Shared Variable Distributed Shared Memory:

MUNIN DSM:

It is a DSM system designed to support multiple consistency protocols based on how data is used in distributed shared memory. It optimizes the performance of DSM by using different protocols for different types of shared variables.
MUNIN also implements release consistency, which allows greater flexibility in memory updates compared to strict or sequential consistency models.

Midway:

It is a distributed shared memory programming system that supports multiple memory consistency models in a single parallel program. Midway was selected for use in a DSM system developed for RHODOS, along with TreadMarks.

Object-based Distributed Shared Memory:

Object-based distributed shared memory (DSM) is a model that allows processes in a distributed system to communicate and share data by manipulating objects. Instead of sharing raw memory, processes interact with shared objects, invoking methods on them to perform operations.

Communication in Object-based DSM:

In an object-based DSM, processes communicate by invoking methods on shared objects. for example, if two processes need to share data about a bank account, they might interact with a BankAccount object. One process can call a method like deposit(amount) on the shared object, which updates the account's state.

Advantages:

1) Encapsulation :- Objects bundle data and methods, improving code organization and reducing errors.

2) Information Hiding :- Only relevant methods are exposed, preventing unintended interactions with the object's internal state.

3) Reuse :- Objects can be reused across different applications, promoting code efficiency.

4) Flexibility :- Objects can be easily modified or extended without affecting other parts of the system.

5) Ease of use :- Developers can use familiar object-oriented programming principles, making it easier to design and implement distributed systems.

Examples:

1) Linda:

Linda is a coordination language that provides a model for concurrent programming, primarily using a tuple space concept.

* Tuple Space:

A tuple space is a shared memory space where data is stored as tuples (ordered lists of elements). It allows processes to communicate by reading, writing, and matching tuples.

* Operations on Tuples:

- In: Removes a tuple from the space.
- Out: Adds a tuple to the space
- Read: Retrieves a tuple without removing it.
- Tuple Broadcast: Sending a tuple to all processes interested in that data.

* Template:

A template is a pattern used to match tuples in the tuple space.

* Replicated Worker Model:

This model allows multiple workers to access shared tasks, improving performance and fault tolerance.

* TaskBag:

A taskbag is a collection of tasks that can be dynamically allocated to workers for processing.

* Implementing Linda:

1) Type Signature: Define the structure of tuples.
2) Tuple Broadcast: Use templates to find matching tuples & distribute them among processes.
3) Unreplicated Tuple Space: A single instance of the tuple space where all tuples reside.

2) ORCA :

Object-RPC Coordination Architecture is a coordination language designed for building distributed applications using shared objects.

In ORCA, shared objects are managed using :

i) Synchronization :- Ensuring that multiple processes can safely access shared objects without conflicts.

ii) Deadlock Prevention :- ORCA's design prevents deadlocks by using techniques like resource allocation graphs or locks that avoid circular wait conditions.

Q. Why Deadlocks are impossible in ORCA ?

=> Deadlocks are avoided in ORCA through :

i) Locking Mechanisms :- Properly designed locks that prevent processes from holding onto resources while waiting for others.

ii) Resource Ordering :- A fixed order for acquiring resources, ensuring that processes always ~~resources~~ request resources in the same sequence, eliminating circular wait conditions.

This comprehensive overview of object-based DSM, Linda, and ORCA covers their core concepts, functionalities and advantages.