Advanced Operating Systems

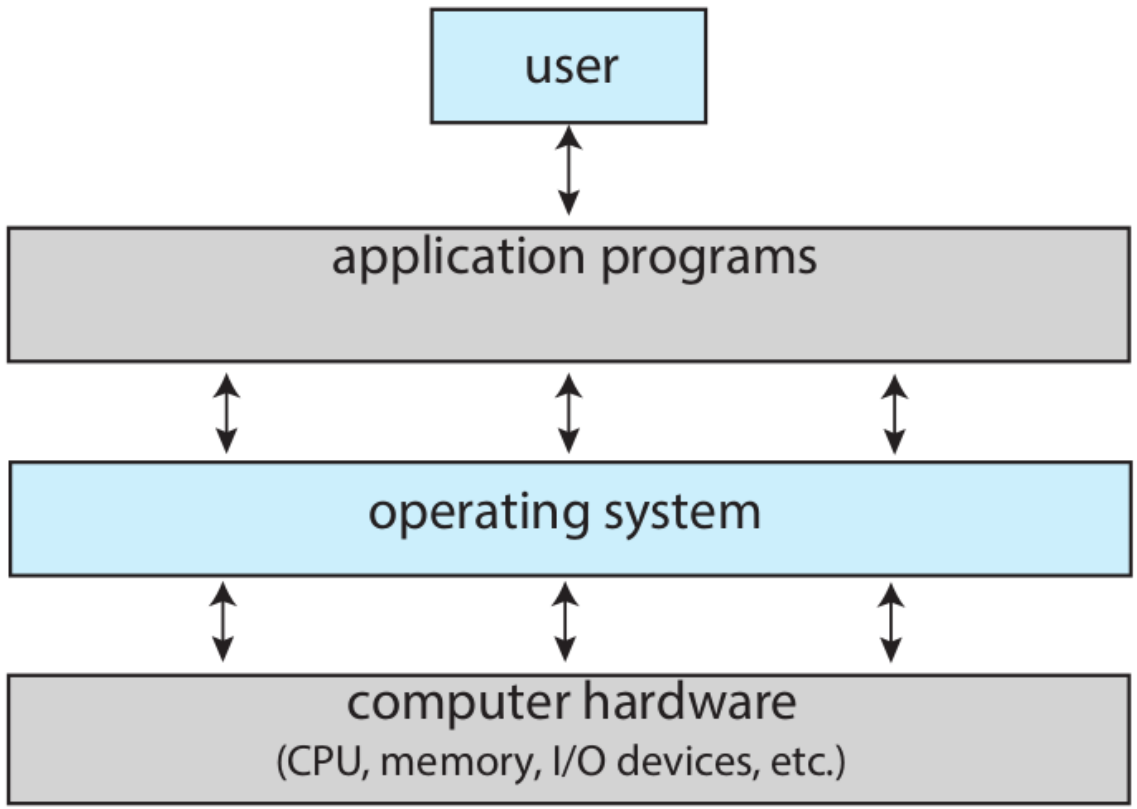
Vasudevan T V

Course Contents

- Module 1 - Overview, Synchronisation Mechanisms, Distributed Operating Systems
- Module 2 - Distributed Mutual Exclusion, Security
- Module 3 - Distributed Resource Management
- Module 4 - Multiprocessor Operating Systems
- Module 5 - Database Systems

Module 1
Overview

- An Operating System is a software that acts as an interface between the user of a computer and the computer hardware
- It provides an environment in which a user can execute programs in a convenient and efficient manner.



Functions of an Operating System

- Memory Management
- Process Management
- Device Management
- File System Management
- Security

Functions of an Operating System

- Memory Management
  - It refers to the management of main memory
  - To execute a program all (or part) of the instructions must be in memory
  - All (or part) of the data that is needed by the program must be in memory
- Memory Management Activities
  - Keeping track of which parts of memory are currently being used and by whom
  - In a multiprogramming environment, the OS decides which process will get memory when and how much
  - Allocating and deallocating memory space as needed

Functions of an Operating System

- Process Management
  - A process is a program in execution
- Process Management Activities
  - In multiprogramming environment, the OS decides which process gets the processor, when and for how much time. This is called process scheduling
  - Keeps track of the status of processor and processes.
  - Allocates the processor to a process
  - De-allocates processor when it is no longer required

Functions of an Operating System

- Device Management
  - It refers to the management of I/O devices
- Device Management Activities
  - Keeps tracks of all devices
  - Decides which process gets the device, when and for how much time
  - Allocates the device to a process
  - De-allocates the device when it is no longer required

Functions of an Operating System

- File System Management
  - Anything that is stored in the secondary storage is a file
  - The structure in which files are organised in a computer is called a file system
  - A directory is a location for storing files
- File System Management Activities
  - Creating and deleting files and directories
  - Gives the permission to the program for operation (read-only, read-write, denied) on file
  - Mapping files onto secondary storage
  - Backup files onto secondary storage

Functions of an Operating System

- Protection and Security
  - Protection is a mechanism for controlling access of processes or users to resources defined by the OS (Setting Passwords, Access Privileges)
  - Security is a mechanism for handling external threats to the system (Attack from hackers, viruses)

Design Approaches

- We can design an operating system in 3 different ways

1. Layered Approach
2. The Kernel Based Approach
3. The Virtual machine Approach

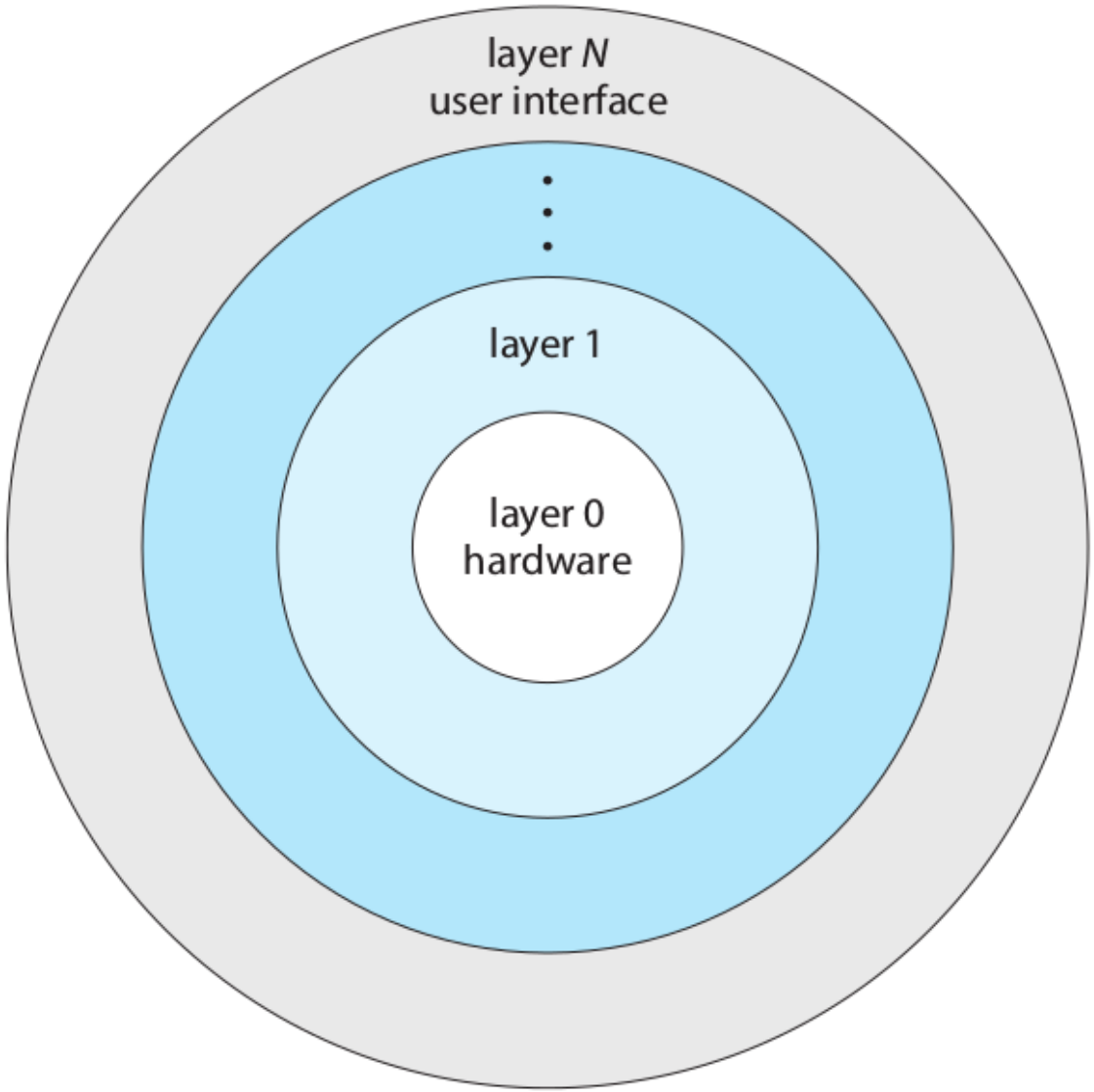
Design Approaches

1. Layered Approach

- The operating system is broken into a number of layers
- The bottom layer (layer 0) is the hardware
- The highest layer (layer N) is the user interface
- Each layer has a well defined functionality and input-output interface with the adjacent layers
- advantage - Each layer can be designed, coded and tested independently
- disadvantage - A user program need to traverse through multiple layers to obtain an operating-system service
- Example - THE operating system designed by a team led by E.W.Dijkstra in 1968

Design Approaches

1. Layered Approach



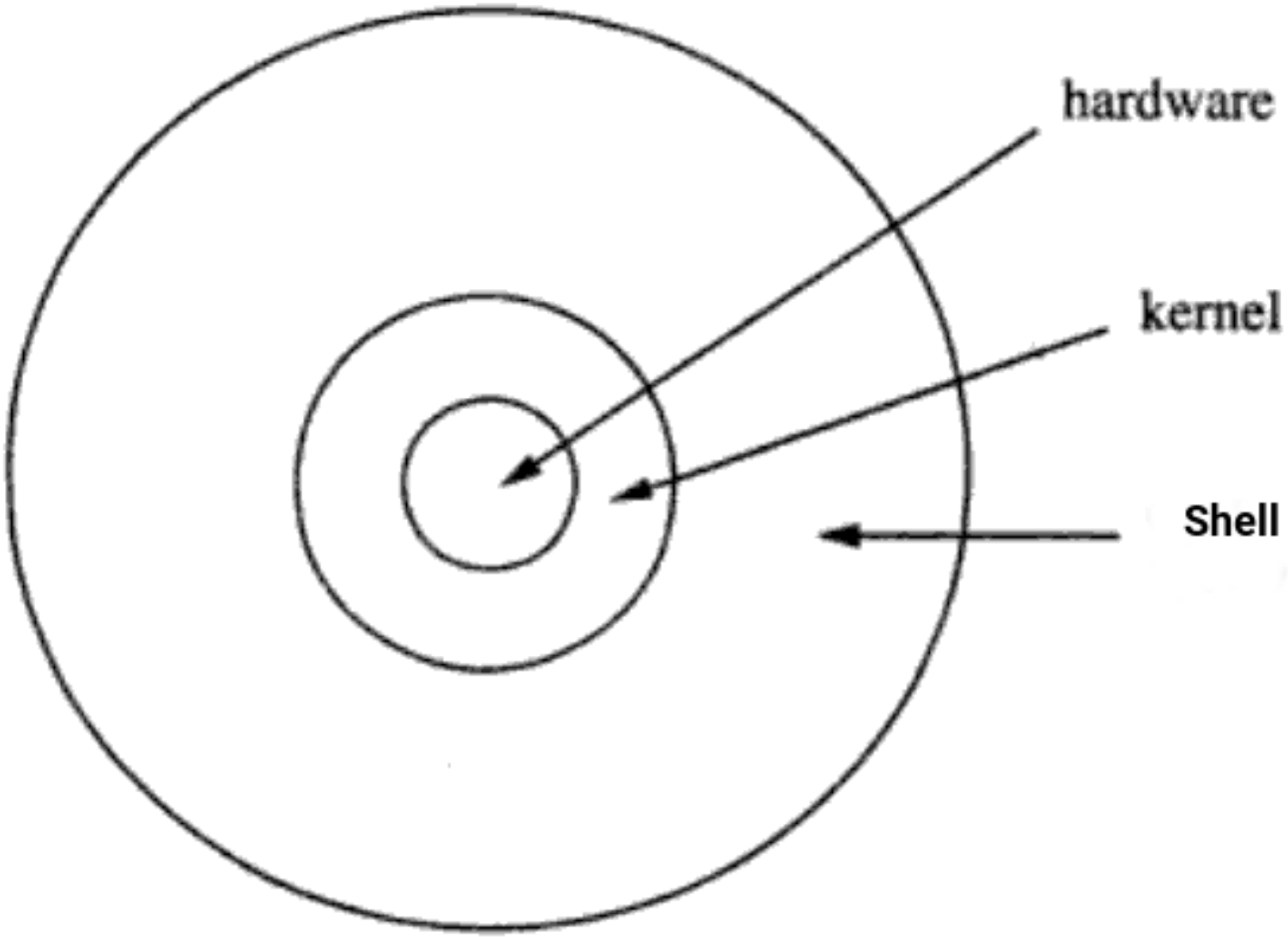
Design Approaches

2. The Kernel Based Approach

- Here there is a core component of the OS called kernel, which interacts directly with the hardware
- Another important component is shell, which is built on top of the kernel, takes commands from the user and provides services to the user
- It is considered as the best approach for an OS
- Examples - GNU Linux, Mac OS, Windows

Design Approaches

2. The Kernel Based Approach



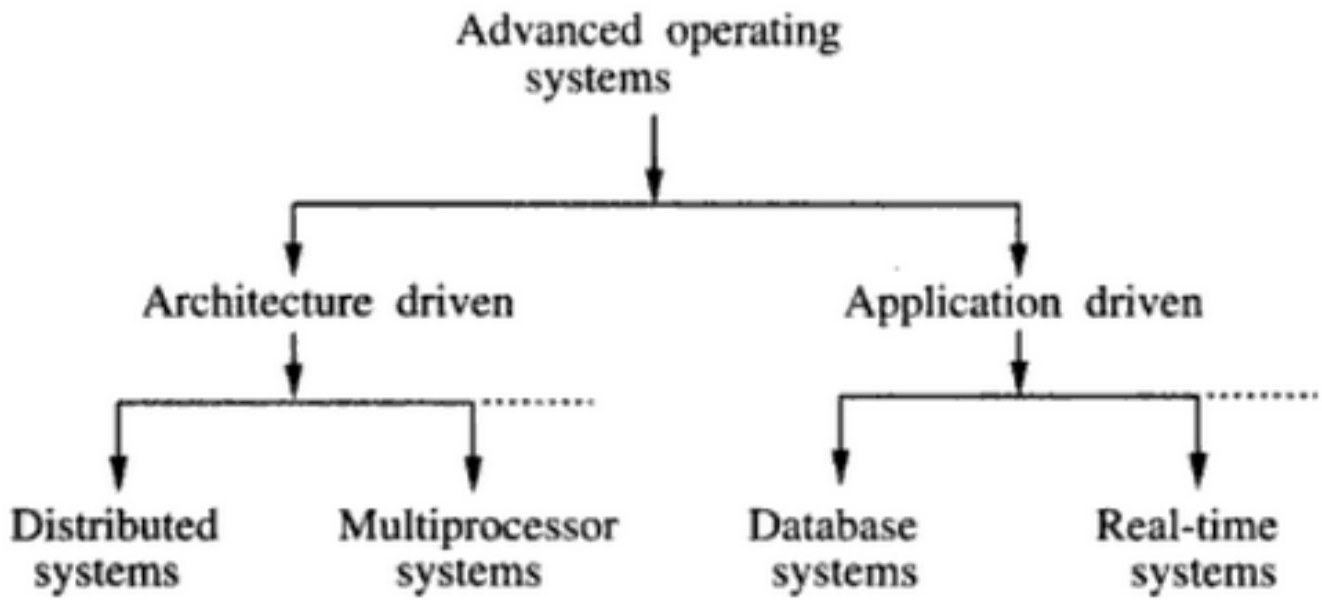
Design Approaches

3. The Virtual Machine Approach

- It is a virtual version of the operating system (also called guest OS) installed within a host computer - personal computer or remote server
- All the resources needed for this OS like CPU, Memory and Storage are borrowed from the host computer
- The virtual machine is partitioned from the rest of the system
- The software inside a virtual machine cannot interfere with the host computer's primary operating system (also called host OS).
- Example - Using the virtualisation software VirtualBox, we can load multiple guest OSes under a single host OS

Advanced Operating Systems

- Traditional operating systems ran on stand alone computers with a single processor
- Advancements in the fields of computer architecture and computer applications led to the development of advanced operating systems
- Classification of Advanced Operating Systems



Advanced Operating Systems

1. Architecture Driven Advanced Operating Systems

- They are used in multicomputer systems with high speed architectures

(a) Distributed Operating Systems

- They are operating systems used for a set of autonomous computers connected by a communication network

(b) Multiprocessor Operating Systems

- They are operating systems used for a set of processors that share memory, storage and peripheral devices over an interconnection network

Advanced Operating Systems

2. Application Driven Advanced Operating Systems

- They are used in advanced applications that require special OS support

(a) Database Operating Systems

- They are operating systems used for database applications

(b) Real Time Operating Systems

- They are operating systems used for real time systems that have job completion deadlines

Synchronisation Mechanisms : Concept of Processes

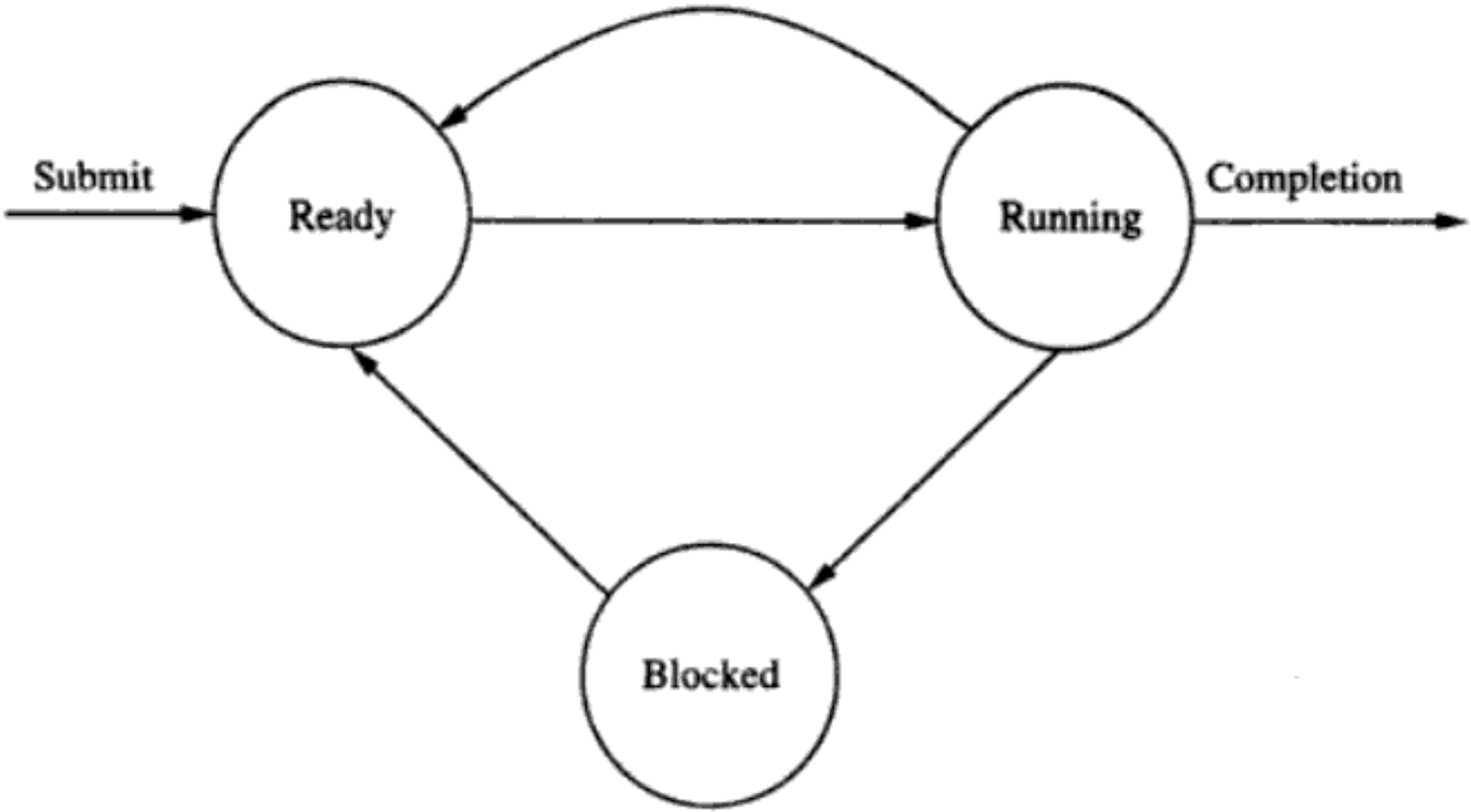
- Here we will discuss about various mechanisms for process synchronisation
- A process is a program in execution
- A process can be in any of the three basic states

1. Running - Instructions are being executed
2. Ready - The process is waiting for the availability of the processor
3. Blocked (Waiting) - The process is waiting for some event to occur such as I/O completion, Memory Availability etc.

- Process Synchronisation is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.
- A resource is any hardware or software associated with a computer.

Concept of Processes

- State Transition Diagram of a Process



Concept of Processes

- Two processes can be serial or concurrent
- Two processes are called serial if the execution of one completes before the execution of the other begins
- Two processes are called concurrent if the execution of one overlaps with the other
- concurrent processes interact with the other by using shared variables and message passing
- If concurrent processes do not interact with the other, then their execution is same as their serial execution

Concepts of Threads

- A thread is a basic unit of execution within a process
- A thread performs a single task associated with a process
- A thread is a lightweight process
- A process can have many threads within it
- A web browser might have one thread display images or text while another thread retrieves data from the network
- A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.

The Critical Section Problem

- Concurrent Processes(or Threads) access shared variables at the same time
- A Critical Section is a code segment in a process in which a shared variable is accessed
- In such a situation, the integrity of variables may be violated
- To solve this problem (Critical Section Problem) we have to ensure that processes need to synchronise in such a way that only one process access the shared variables at one time
- We have to ensure that no two processes are executing in their critical sections at the same time

The Critical Section Problem

- Each process must request permission to enter its critical section
- The section of code implementing this request is the entry section
- The critical section may be followed by an exit section
- The remaining code is the remainder section

```
while (true) {

    entry section

        critical section

    exit section

        remainder section

}
```

The Critical Section Problem

- A solution to the critical-section problem must satisfy the following three requirements:

1. Mutual Exclusion
   Only one process can execute in its critical section in a given period of time.
2. Progress
   When no process is executing in its critical section, then any process which wants to enter its critical section must be granted permission without delay.
   If there are two or more processes which want to enter their critical sections, then the selection cannot be postponed indefinitely.
3. Bounded Waiting
   No process can prevent any other from entering its critical section indefinitely. In other words, there exists a bound on the waiting time of a process before entering its critical section.

Semaphores

- A solution to this problem is the usage of semaphores
- A semaphore is an integer variable that, apart from initialisation, is accessed only through two standard atomic operations: wait() and signal() .
- Semaphores were introduced by the Dutch computer scientist E.W.Dijkstra

```
wait(S)
{
     while (S <= 0);  // If S is not positive no
                      // operation is performed
     S = S - 1;  // If S is positive, it is decremented
                 // by 1
}
signal(S)
{
     S = S + 1; // S is incremented by 1
}
```

Semaphores

- There are two types of semaphores

1. counting semaphore
   They have an unrestricted value domain. The semaphore count is the number of available resources. If the resources are added, semaphore count is incremented and if the resources are removed, the count is decremented.
2. binary semaphore
   Their values are restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal operation works only when semaphore is 0.

- Binary semaphores are easier to implement than counting semaphores

Semaphore

- Semaphore Usage
- Consider two concurrently running processes: P1 with a statement S1 and P2 with a statement S2.
- Suppose we require that S2 be executed only after S1 has completed
- We can implement this scheme by letting P1 and P2 share a common binary semaphore S, initialised to 0

```
Process P1          Process P2
----------          ----------

..........          ..........
S1 ;                wait(S);
signal(S);          S2;
..........          ..........
```

- The object used for achieving mutual exclusion is called mutex
- Here binary semaphore S is the mutex

Semaphore

- Advantages of Semaphores
- They provide a simple scheme for all kinds of synchronisation problems.
- Semaphores are implemented in the machine independent code of the micro kernel.
- Disadvantages of Semaphores
- A process that uses a semaphore has to know which other processes are using it.
- The wait and signal operations must be implemented in the correct order to prevent deadlocks.

Monitor

- A Monitor is a synchronisation tool that encapsulates data with a set of functions to operate on that data
- It was invented by Hansen and Hoare

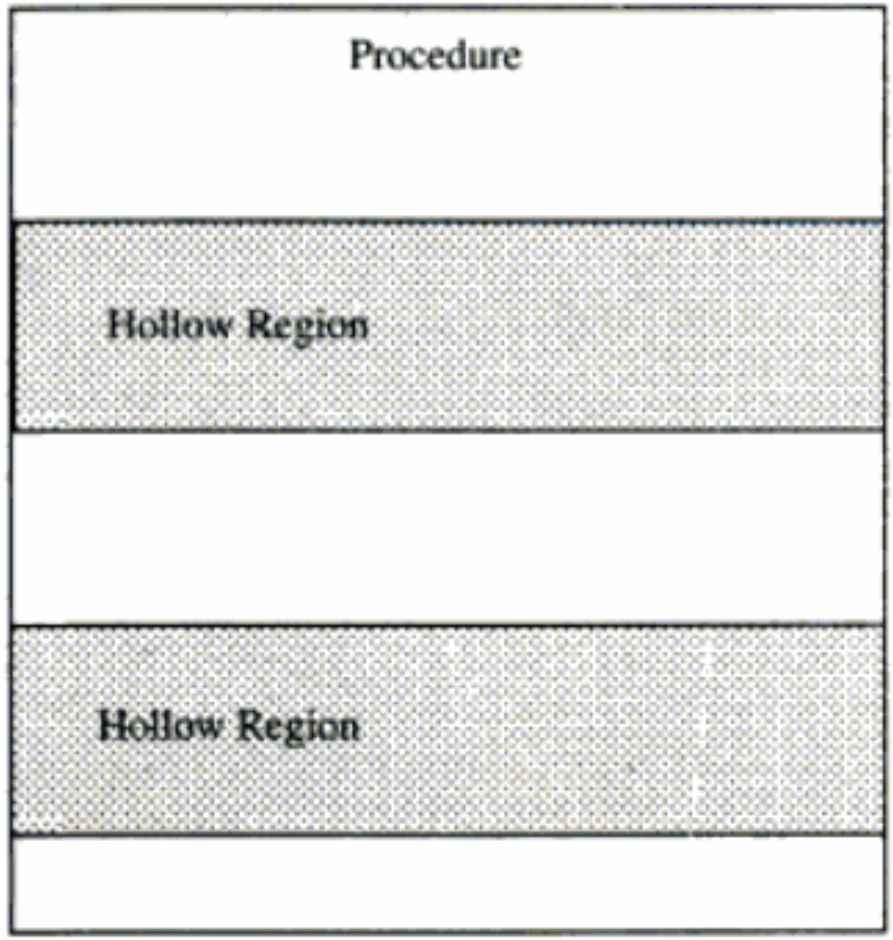
```
monitor monitor name
{
   /* shared variable declarations */

   function P1 ( . . . ) {
      . . .
   }

   function P2 ( . . . ) {
      . . .
   }

         .
         .
         .
   function Pn ( . . . ) {
      . . .
   }

   initialization_code ( . . . ) {
      . . .
   }
}
```

Monitor

- A function defined within a monitor can access only the local variables within the monitor
- Local variables within the monitor can be accessed by the local functions
- The synchronisation of processes is achieved by two special operations, wait and signal, that execute within monitor's functions
- The monitor construct ensures that only one process is active (can execute a function) at a time within it. Thus it ensures mutual exclusion
- Other processes that need to access the shared variables in a monitor will be waiting in a queue
- Monitors are implemented using high level programming languages (Java, C#)
- There is a possibility of deadlock here also, if wait and signal operations are not in proper order

Serialiser

- This tool was developed by Hewitt and Atkinson in 1976
- It is a construct similar to a monitor
- However, functions within a serialiser can have hollow regions where concurrent processes can be active



Serialiser

- When a process releases the serialiser, no explicit signal is sent to other waiting processes, as in a monitor.
- The signalling is done automatically in a serialiser
- advantages
- It promotes concurrent execution of processes in hollow regions
- It minimises the prospect of deadlocks, due to the presence of hollow regions
- disadvantages
- Its implementation is more complex
- Automatic signalling process increases overhead

Path Expressions

- It follows a different approach to process synchronisation
- It was proposed by Campbell and Habermann
- It is an expression whose variables are the operations on the shared resource and whose operators are the execution order of these operations
- A path expression has the following form
- **path** S **end**
- Here S contains variables and operators
- The variables are open,read, write and close
- The operators are
- **;** defines a sequencing order among operations
- + specifies that only one of the operations connected by it can be executed at the same time
- {} signifies the concurrent execution of the specified operations within it

Path Expressions

- Examples
- **path** open; read; close **end**
- An open operation is performed first, followed by read and then close operations
- **path** read + write **end**
- Either read or write operation can be performed at a given time; Order does not matter
- **path** {read} **end**
- Several read operations can be executed concurrently

Distributed Operating Systems:

- A distributed os appears to its users as a centralised operating system for a single machine, but it runs on multiple independent computers
- An identical copy of the operating system runs on every computer
- Here the user does not know, on what computers a job was executed, on what computers files are stored, how the communication and synchronisation between computers are done
- Examples - Amoeba, Inferno, MOSIX

Design Issues in Distributed Operating Systems

1. Global Knowledge

- We have to find techniques to capture the current global state of the system, as there is no shared memory in a distributed os

2. Naming

- Names are used to identify objects such as computers, printers, users, files etc.
- Since objects are distributed across various computers, how naming is done plays a key role in the efficiency of the system

3. Scalability

- It is the ability of the system to adapt with changes
- The performance of the system should not get affected by increase in users, hardware and software resources

Design Issues in Distributed Operating Systems

4. Compatibility

- The ability of one computer to work with another, without problems
- binary level compatibility - The executable code of a program should run correctly in all computers without recompilation
- execution level compatibility - The source code of a program can be compiled and run on all computers of the system
- protocol level compatibility - All computers support a common set of rules for file access, naming, authentication etc.

5. Process Synchronisation

- It is more difficult due to the absence of shared memory

Design Issues in Distributed Operating Systems

6. Resource Management

- The local and remote resources are made available in an effective manner
- They are made available in 3 ways.
- data migration - data is transferred from its source to the location of computation and if any modification is made in the data then it is also reflected at the source side.
- computation migration - In this approach, computation itself is migrated (message passing) to the location where required data is present; used when data migration is costly
- distributed scheduling - Here, processes are executed in computers different from where they are originated; used when the computer where a process is originated is overloaded

Design Issues in Distributed Operating Systems

7. Security

- The security of the computer system is the responsibility of OS
- There are two aspects of security
- authentication - the identity of users are checked for providing the access to the system (username, password)
- authorisation - the privileges of the users are checked for accessing resources (read privilege, write privilege)
- authentication is performed before authorisation

Design Issues in Distributed Operating Systems

8. Structuring

- It refers to how various parts of the OS are organised
- There are three ways of doing this
- Monolithic Kernel(Examples - GNU Linux, DOS, Windows 98)
- This kernel will contain all the services of the operating system
- It will be wasteful for a distributed system where every computer may not use every service provided by the OS
- The Collective Kernel Structure (Examples - Chorus OS, GNU Hurd)
- The operating system is structured as a collection of processes that are largely independent of each other
- The interaction between the processes is done through message passing, which is the responsibility of microkernel, the nucleus of the os
- Object Oriented OS (Examples - Amoeba,Eden)
- Here the system services are implemented using objects

Design Issues in Distributed Operating Systems

9. Client-Server Computing Model

- In this model, processes are categorised as clients and servers
- A client is a process that need a service (example - read data from a file)
- A server is a process that provide a service
- A client and a server interact with each other by sending messages

Communication Networks

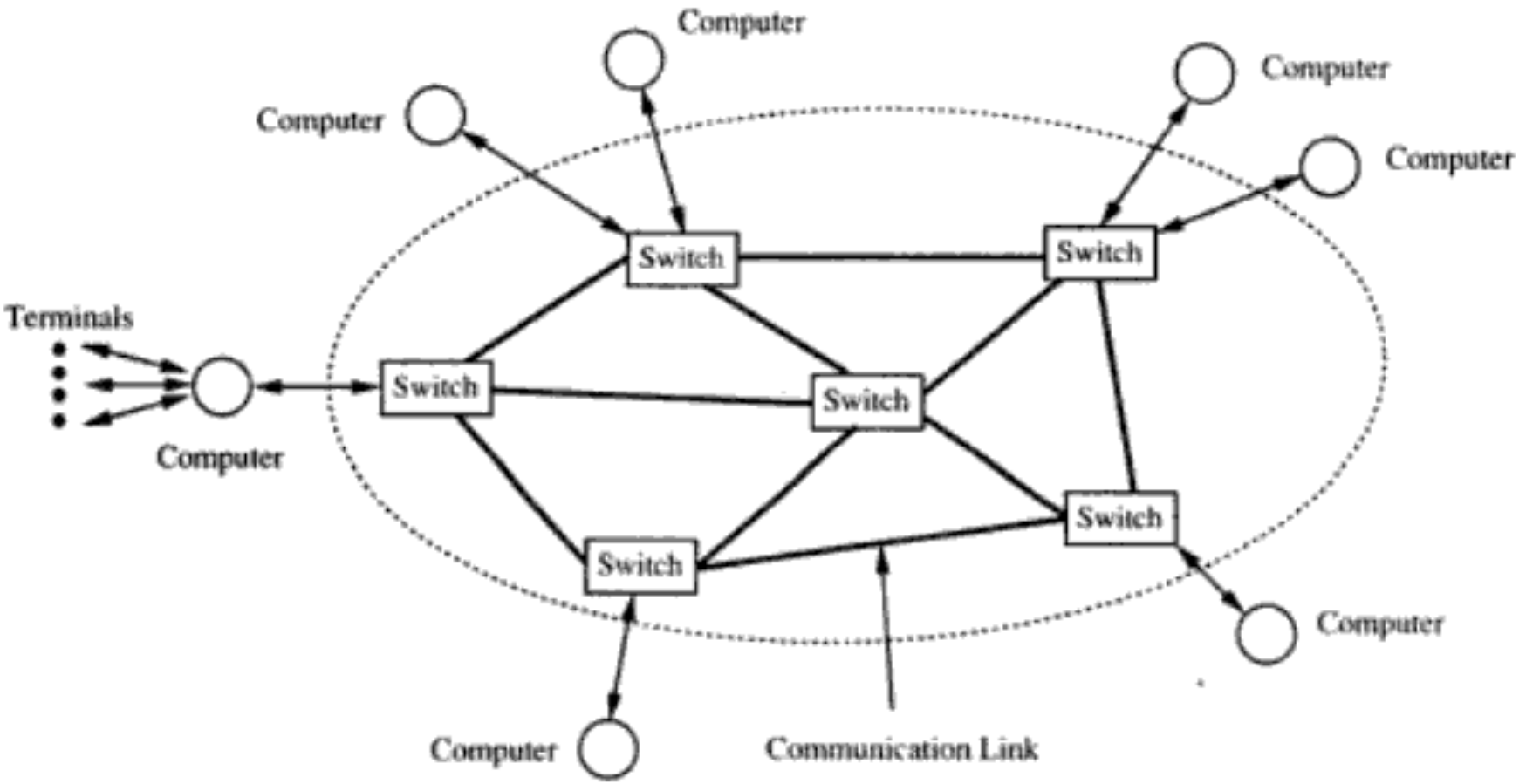
- In a distributed system, all the computers are connected through a communication network
- There are mainly two types of communication networks

1. Wide Area Network (WAN) (Long Haul Network)
2. Local Area Network (LAN)

Communication Networks

1. Wide Area Network (WAN)

- It connects computers spread over a wide geographic area that cover cities, states and countries



Communication Networks

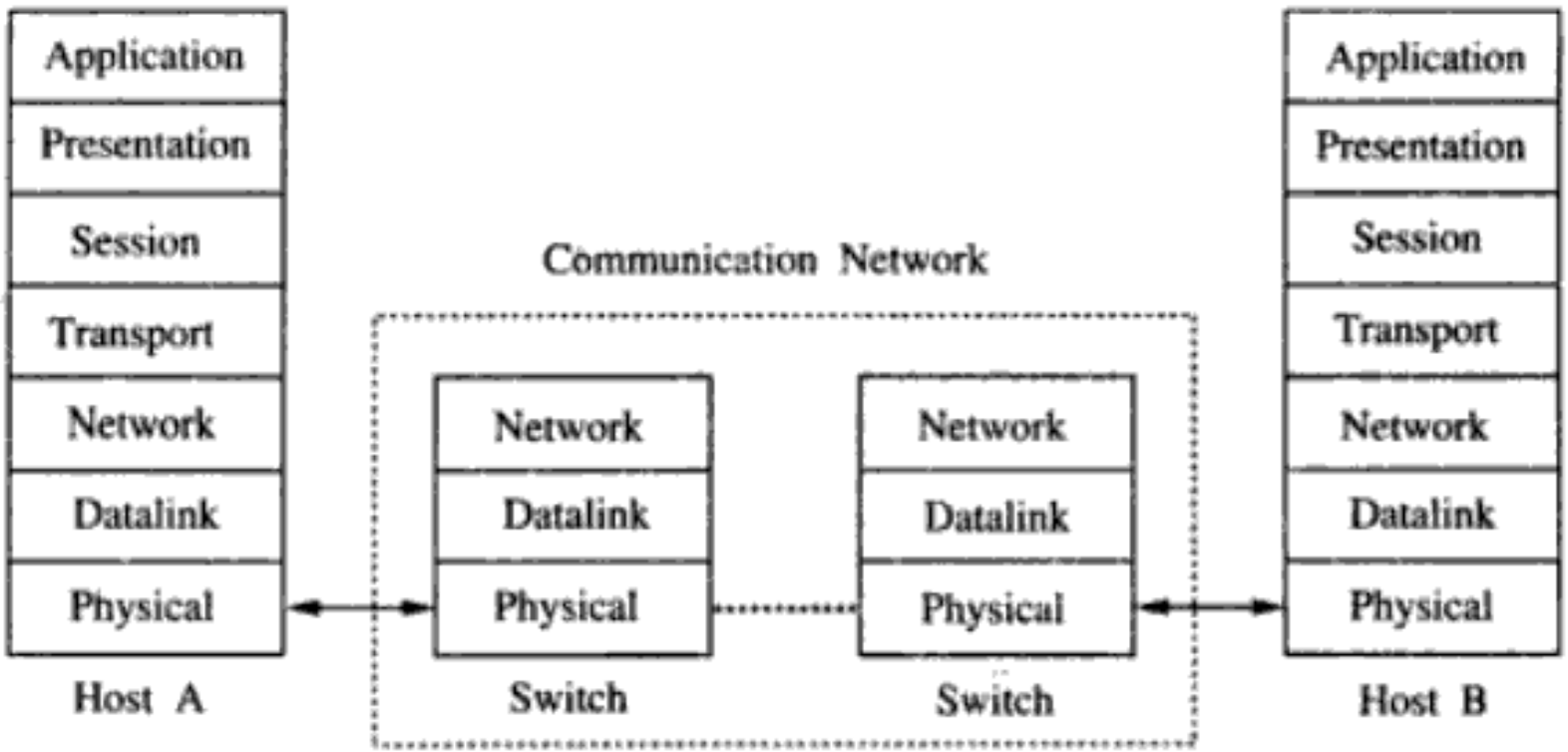
1. Wide Area Network (WAN)

- The communication facility in a WAN consists of a set of switches interconnected by communication links
- Switches are devices used for routing data through an appropriate path while avoiding network congestion
- Data can be transmitted using either Circuit Switching or Packet Switching methods
- In Circuit Switching, data is transmitted through a dedicated path between two devices
- In Packet Switching, data is divided into several packets, each packet is individually routed to the destination, and they are combined at the destination
- The communication links in a WAN can be telephone lines, optical fibre cables, satellites or microwave links

Communication Networks

1. Wide Area Network (WAN)

- A WAN consists of heterogeneous devices such as computers, terminals, printers etc.
- The ISO OSI Reference Model(shown below) provides a framework for communication in a heterogeneous environment



Communication Networks

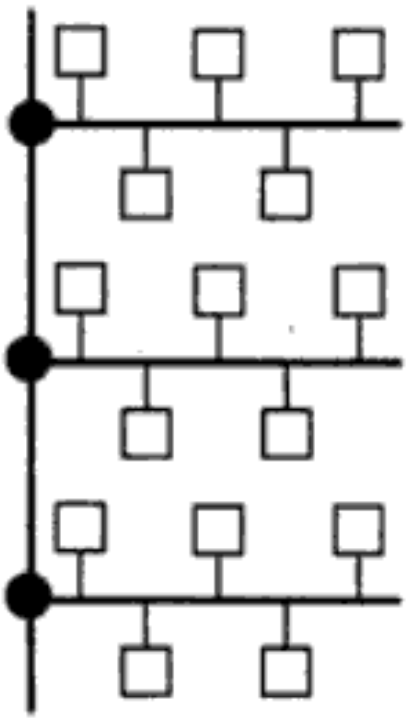
1. Wide Area Network (WAN)

- The ISO OSI Reference Model consists of 7 layers
- The Physical Layer is responsible for sending data in bit streams over a communication network
- The Data Link Layer is responsible for recovering from data transmission errors
- The Network layer is responsible for routing and congestion control
- The Transport Layer is responsible for reliable transfer of data between two devices
- The Session Layer is responsible for managing communication sessions between user processes
- The Presentation Layer is concerned with the syntax and semantics of the information exchanged between two systems
- The Application Layer provides application program interface to the users

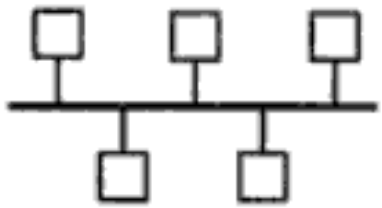
Communication Networks

2. Local Area Network (LAN)

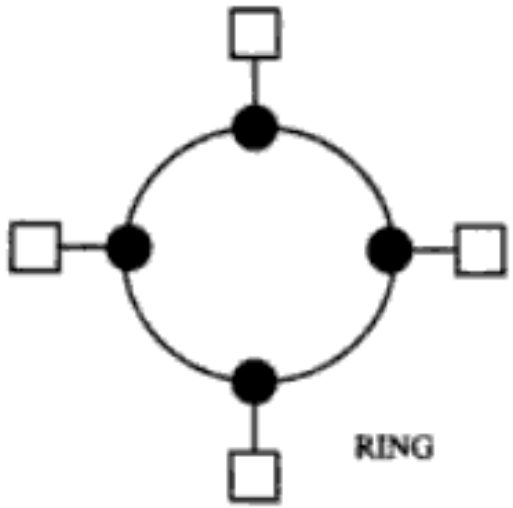
- It connects computers within a small geographic area that cover a single building or several buildings
- Network Topologies for LAN



TREE



BUS



RING

Communication Networks

2. Local Area Network (LAN)

- The communication media can be coaxial cable, twisted pair cable or optical fibre cable
- Bus/Tree Topology
- The bus topology can be viewed as branches of a tree topology
- In Bus/Tree Topology, data is directly send between two devices, through the common data path (bus)
- CSMA/CD Protocol and Token Bus Protocol are used to control access to the bus
- Carrier Sense Multiple Access / Collision Detection (CSMA/CD) Protocol is capable of detecting collision of multiple transmissions over the bus
- In Token Bus Protocol, only the device holding a token is allowed to transmit through the bus

Communication Networks

2. Local Area Network (LAN)

- Ring Topology
- In ring topology, there is a circular data path
- Here data is transmitted from one device to another, through intermediate devices, if there are any
- Token Ring Protocol and Slotted Ring Protocol are used to control access to the bus
- In Token Ring Protocol, a token circulates around the ring
- When no device is transmitting data, token is labelled free
- When a device wants to transmit data, the arriving token is labelled busy and data is transmitted; no other device can use the token now
- After data transmission, when the token again arrives at the device, it is marked as free

Communication Networks

2. Local Area Network (LAN)

- Ring Topology
- In Slotted Ring Protocol, a number of fixed length slots circulates around the ring
- When no device is transmitting data, slots are labelled free
- When a device wants to transmit data, the arriving slot is labelled busy and data is transmitted; no other device can use that slot now
- After data transmission, when the slot again arrives at the device, it is marked as free
- Here, a device can use only one slot at a time

Communication Primitives

- They are the high level constructs in programs for accessing communication networks
- We will discuss about two communication models - message passing and remote procedure call - that provide communication primitives
- The message passing model uses two communication primitives - SEND and RECEIVE
- The SEND primitive has two parameters - a message and its destination
- The RECEIVE primitive has two parameters - the source of the message and a buffer for storing the message

Communication Primitives

- These primitives can have four modes - blocking,non blocking, synchronous and asynchronous
- In blocking mode, SEND primitive does not return control to the user process until message has been sent, or an acknowledgement has been received
- In non blocking mode, SEND primitive returns control to the user process as soon as the message is copied to kernel buffer, an intermediate buffer used for storing message
- In synchronous mode, SEND primitive is blocked until the execution of the corresponding RECEIVE message (Another SEND cannot be performed before the completion of RECEIVE)
- In asynchronous mode, SEND primitive is not blocked until the execution of the corresponding RECEIVE message (Another SEND can be performed before the completion of RECEIVE)

Communication Primitives

- In the remote procedure call model, a local computer (client) can call a procedure on a remote computer (server)

1. Client invokes a procedure on Server
2. The invoked procedure is executed and the result is sent back to the client
3. The calling procedure is suspended until the arrival of the result

- Here also two primitives similar to SEND and RECEIVE are used

Lamport's Logical Clocks

- People use physical time to order events
- Example - An event at 8.00 AM occurs before an event at 8.01 AM
- In distributed systems physical clocks are not always precise
- Hence we can use logical clocks for ordering events
- Leslie Lamport proposed one such system in 1978

Lamport's Logical Clocks

- A logical clock is a mathematical function that assigns numbers to events
- The number assigned to an event is called timestamp associated with the event
- The following two conditions are satisfied by Lamport's Logical Clocks
- Condition 1 - For any two events a and b in Process P_i, If a happens before b then, $C_i(a) < C_i(b)$
- Condition 2 - If a is the event of sending a message in Process P_i and b is the event of receiving the same message in Process P_j then, $C_i(a) < C_j(b)$

Causal Ordering of Messages

- It was proposed by Birman and Joseph
- The condition for causal ordering of messages is given below
- If message M_1 is sent before message M_2, then every recipient of these messages should receive M_1 before M_2
- It is not automatically guaranteed in a distributed system
- The basic idea to implement this is to ensure that deliver a message only if the previous message has been delivered
- Until then we buffer the message

Module 2
Distributed Mutual Exclusion:

- Implementation of Mutual Exclusion is more complex in a distributed system owing to the following reasons

1. Lack of shared memory
2. Absence of a common physical clock
3. Unpredictable message delays
4. Difficult to know the current global state of the system

Classification of Mutual Exclusion Algorithms

- Token Based Algorithms
  A site (or process) is allowed to enter its critical section only if it possesses a token
- Non Token Based Algorithms
  A site (or process) can enter its critical section only after getting permission from other sites

Requirements of Mutual Exclusion Algorithms

1. Freedom from deadlocks
   Two or more sites should not endlessly wait to enter their critical sections
2. Freedom from starvation
   Every requesting site should get an opportunity to enter its critical section in a finite time
3. Fairness
   Requests from sites must be executed in the order in which they arrive
4. Fault Tolerance
   The system continues to operate even after failure of some of its components

Measuring Performance

- Metrics Used

1. Message Complexity
   The number of messages required per critical section execution by a site
2. Synchronisation Delay
   The time interval between one site leaving the critical section and another site entering the critical section
3. Response Time
   The time interval between sending request message to enter critical section and the completion of critical section execution
4. System Throughput
   The rate at which the system executes requests for the critical section.
   system throughput $= 1 / (sd + E)$
   where sd is the synchronisation delay and E is the average critical section execution time

Lamport's Algorithm

- Non Token Based Algorithm

1. A site sends a request message to all other sites to enter the critical section and places the request in its request queue (ordered by timestamps)
2. The site waits for reply message(permission) from all other sites
3. A site receiving a request message, places it in its request queue, gives a reply message to the sender
4. A site can enter the critical section, after getting reply message from all other sites and its own request is at the front of the request queue
5. Upon exiting the critical section, the site removes its request from the queue and sends a release message to every other site
6. A site after receiving a release message, removes the corresponding request from its request queue

Ricart–Agrawala Algorithm

- Non Token Based Algorithm
- Optimised Version of Lamport's Algorithm
- No need of release messages

1. A site sends a request message to all other sites to enter the critical section
2. The site waits for reply message(permission) from all other sites
3. A site receiving a request message, gives a reply message to the sender, if and only if it is not currently executing (or not interested) in critical section
4. A site can enter the critical section, after getting reply message from all other sites
5. Upon exiting the critical section, the site sends all deferred reply messages

Suzuki–Kasami Broadcast Algorithm

- Token Based Algorithm

1. A site who wants to enter the critical section and who does not have the token sends a request message for it to all other sites
2. A site holding the token and is not presently executing the critical section, sends the token to the requesting site upon receiving the request message
3. A site holding the token and is presently executing the critical section, sends the token to the requesting site only after completing the execution
4. A site holding the token can repeatedly enter the critical section until it sends the token to any other site

Security : Potential Security Violations

- Security deals with the control of unauthorised use and the access to hardware and software resources of a computer system
- Classifications of potential security violations

1. Unauthorised information release
   This occurs when an unauthorised person access information stored in the system
   Example - Unauthorised person reading balance in a bank account
2. Unauthorised information modification
   This occurs when an unauthorised person modify information stored in the system
   Example - Unauthorised person updating balance in a bank account
3. Unauthorised denial of service
   This occurs when an unauthorised person prevents an authorised person from accessing information stored in the system

Design Principles for Secure Systems

1. Economy
   One way to achieve economy is to keep the design as simple as possible
2. Complete Mediation
   Every request to access an object should be checked
3. Open Design
   A protection mechanism should work even if its underlying principles are known to an attacker
4. Separation of Privileges
   Give privileges as required for various users for performing read, write, edit, execute etc.

Design Principles for Secure Systems

5. Least Privilege
   A user should be given minimum access rights required to perform a task
6. Least Common Mechanism
   Minimise shared access to information
7. Acceptability
   The security mechanism should be simple to use
8. Fail-safe defaults
   If a system fails, it should give a safe outcome; default case should be lack of access

The Access Matrix Model

- This is the most fundamental model of protection in a computer system
- It was proposed by Lampson
- Components of this model

1. Current Objects
   They are entities whose access is to be controlled
   Example - A file
2. Current Subjects
   They are entities who access current objects
   Example - A process
3. Generic Rights
   Various access rights subjects can have on objects
   Examples - read, write, execute

The Access Matrix Model

- The Protection State of a system is represented by the triplet (S,O,P) where
- S is the set of current subjects
- O is the set of current objects
- P is a matrix called the access matrix where
- There is a row for every current subject
- There is a column for every current object
- Each entry P[s,o] denotes generic rights of subject s on object o

The Access Matrix Model

- An access matrix representing a protection state

	o_1	o_2	s_1	s_2	s_3
s_1	*read, write*	*own, delete*	*own*	*sendmail*	*recmail*
s_2	*execute*	*copy*	*recmail*	own	*block, wakeup*
s_3	*own*	*read, write*	*sendmail*	*block, wakeup*	*own*

- subjects - S_1, S_2, S_3
- objects - O_1, O_2, S_1, S_2, S_3
- Here subjects which are accessed by other subjects can also be treated as objects

Implementation - The Access Control List Method

- An access matrix is likely to be very sparse, since every subject is not having access right on every object
- It is better to implement this using an access control list, which is a column wise decomposition of the access matrix
- Here we represent access rights for each object by various subjects
- Example

Subjects	Access rights
Smith	read, write, execute
Jones	read
Lee	write
Grant	execute
⋮	⋮
White	read, write

Module 3
Distributed Resource Management: Introduction

- A distributed file system is a resource management component of a distributed operating system
- It implements a common file system that is shared by all the autonomous computers in the system
- Goals of a distributed file system

1. Network Transparency
   Users need not be aware of the location of files to access them
2. High Availability
   Users can easily access files irrespective of their physical location

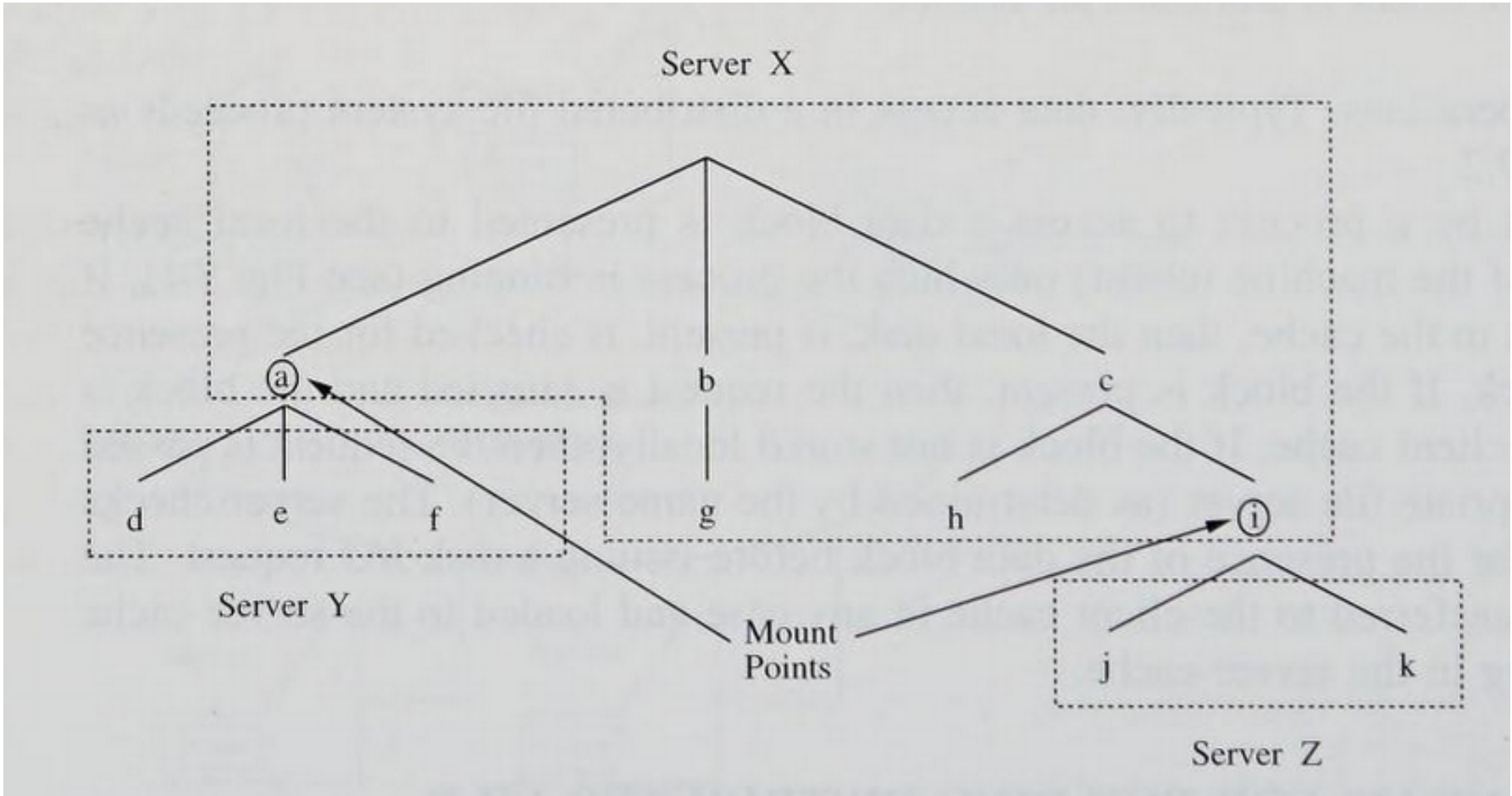
Mechanisms for Building Distributed File Systems

1. Mounting

- The process by which the operating system makes files and directories on a storage device (such as hard drive, pen drive, CD / DVD etc) available for users to access via the computer's file system
- A mount point is an empty directory in the currently accessible file system on which an additional file system is mounted
- The kernel maintains a data structure called mount table, which holds information about the currently mounted file systems
- Mounting is UNIX specific
- Most of the existing distributed file systems are based on UNIX

Mechanisms for Building Distributed File Systems

1. Mounting



Mechanisms for Building Distributed File Systems

2. Caching

- Here a copy of data stored on a remote server is brought to the client, when referenced by the client
- Subsequent access to the data is done locally, thereby reducing access delay
- Data can either be cached in main memory or on the local disk

3. Hints

- When multiple clients cache and modify data, the problem of cache inconsistency arises
- It is difficult to guarantee cache consistency in a distributed file system
- An alternative approach is to treat cached data as hints, which may not be fully accurate
- Valid cache entries improve performance of the system
- We need to recover from invalid cache entries

Mechanisms for Building Distributed File Systems

4. Bulk Data Transfer

- Here multiple consecutive data blocks are copied from servers to clients, instead of just the referenced data block
- This may be needed later, since most files are accessed entirely

5. Encryption

- It is used for enforcing security in distributed systems
- Here the original information to be shared is converted into an encrypted form, using an encryption algorithm
- This need to be converted back to the original form at the destination using a decryption algorithm
- Only authorised users can perform decryption

Design Issues

- Here we will see design issues associated with distributed file systems

1. Naming and Name Resolution

- Naming is the act of giving a name to an object (such as file or directory) in a file system
- Name Resolution is the process of mapping a name to an object
- Name space is a collection of names in a file system
- There are 3 approaches for naming in a distributed file system
- Approach 1 - A file is identified by a combination of host name and local name
  host:local-name
- When a file is moved from one host to another, its name is changed

Design Issues

1. Naming and Name Resolution

- Approach 2 - Attach remote directories to local directories; Now we can refer any file in the system
- Since we can attach any remote directory to any local directory, the resulting hierarchy is highly unstructured
- Approach 3 - Create a single global directory of all files in the distributed system
- The requirement of system wide unique file names may be impractical for large distributed systems

Design Issues

2. Caches on Disk or Main Memory

- The data from a remote server can be cached in the main memory or local disk of a client
- Data caching in main memory
- Advantage - Data can be accessed fast
- Disadvantage - Large files cannot be cached in main memory
- Data caching in disk
- Advantage - Large files can be cached in disk
- Disadvantage - Data access will be slower

Design Issues

3. Writing Policy

- It decides when a modified data at the cache memory in the client is sent to server
- Approach 1 - Write Through Policy
- Modified data is updated immediately in the server
- Advantage - Reliability
- Disadvantage - some writes are intermediate results which need not be updated in server
- Approach 2 - Delayed Writing Policy
- Modified data is updated after some delay in the server
- Advantage - Only final results are updated in server
- Disadvantage - Less Reliability

Design Issues

4. Cache Consistency

- We need to ensure consistency of cached items at multiple clients
- Approach 1 - Server Initiated Approach
- Server is responsible for validating data stored at cache memory at clients
- Approach 2 - Client Initiated Approach
- Client is responsible for validating data stored at cache memory

Design Issues

5. Availability

- We need to ensure availability of data, which can be affected by failure of servers or communication network
- Replication is the mechanism for ensuring availability
- Using replication many copies of files are maintained at different servers
- We have to ensure consistency of replicated data
- Depending upon the requirement, the unit of replication can be a single file or a group of files(volume)

Design Issues

6. Scalability

- The ability of the system to adapt with change in work load
- Change in work load can happen with changes in number of users, storage requirement, number of components
- Scalability can be enhanced by compression and deduplication
- Compression reduces the size of a file
- Deduplication eliminates redundant data
- Both of these reduce the amount of storage needed

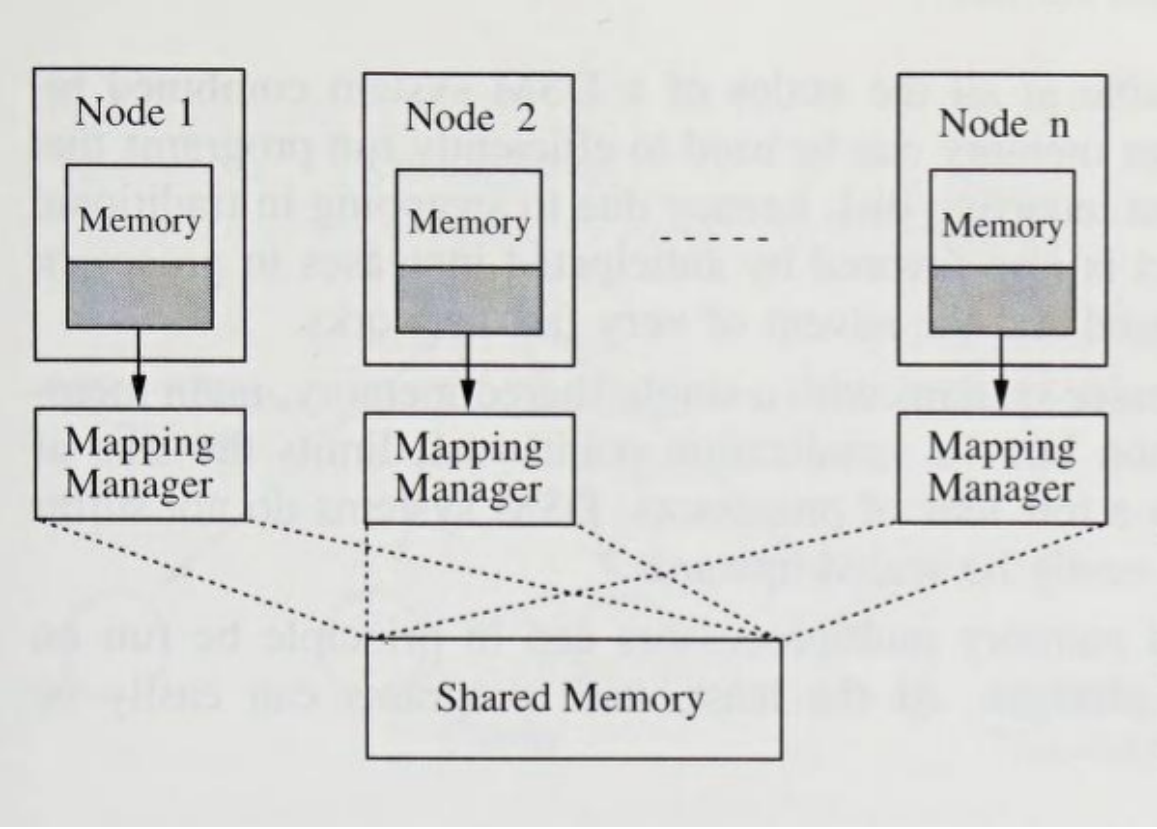
7. Semantics

- It deals with effects of accesses on files by various users
- We have to ensure that a read operation returns the data corresponding to the last write operation
- It is difficult to ensure this in a distributed system due to communication delays

Distributed Shared Memory(DSM)

- In a distributed system, there is no physically shared memory
- In other words, there is no centralised memory that is shared by different computers
- Distributed Shared Memory is a form of memory architecture in which separate physical memories are addressed as a single address space which is logically shared
- This logically shared address space is also called the virtual address space

Distributed Shared Memory(DSM)



- Here mapping manager is responsible for mapping the shared logical memory address to the physical memory (which can be local or remote)
- Mapping manager is a layer of software which is implemented either in the operating system kernel or as a run time library routine

Algorithms for Implementing Distributed Shared Memory

- The central issues in the implementation of distributed shared memory

1. How to keep track of the location of remote data
2. How to overcome the communication delays and high overhead while accessing remote data
3. How to make shared data concurrently accessible at different nodes

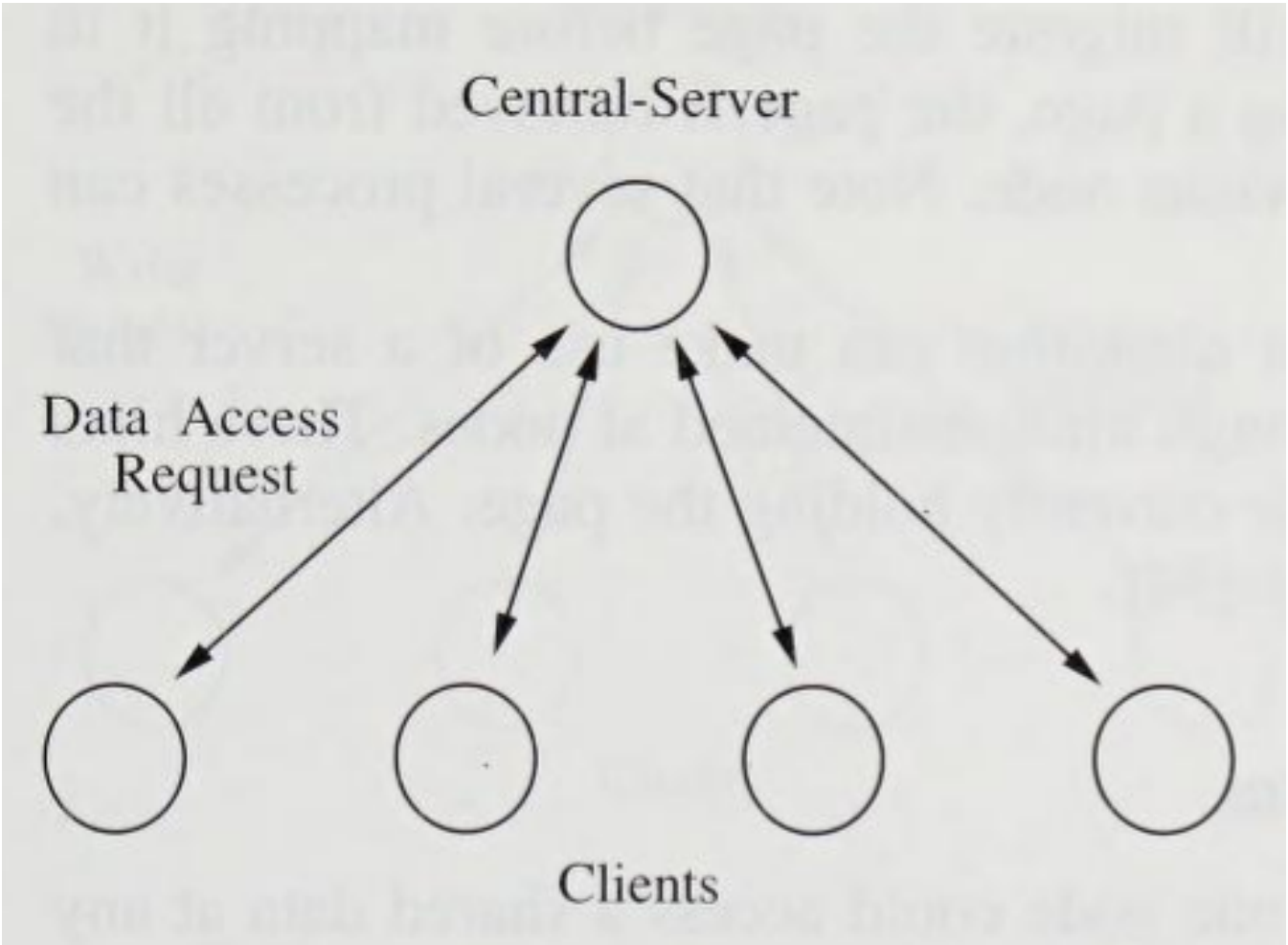
Algorithms for Implementing Distributed Shared Memory

1. The Central Server Algorithm
2. The Migration Algorithm
3. The Read Replication Algorithm
4. The Full Replication Algorithm

Algorithms for Implementing Distributed Shared Memory

1. The Central Server Algorithm

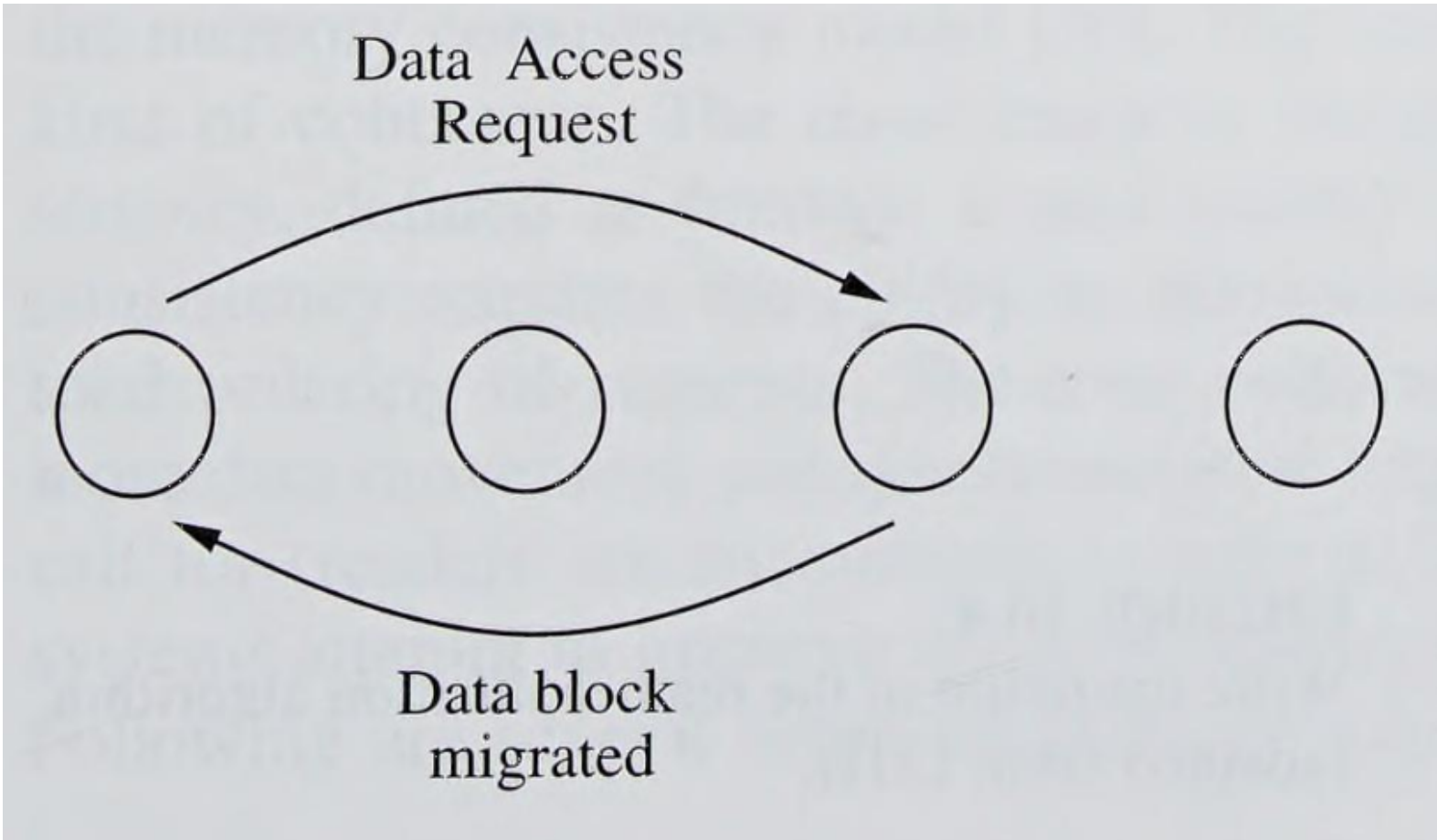
- Here a central server maintains all the shared data
- The central server services all read / write requests from clients



Algorithms for Implementing Distributed Shared Memory

2. The Migration Algorithm

- Here data is migrated from its original location to the node where it is needed, after the initial request
- After that data can be accessed locally
- Here only one node can access the shared data at a time



Algorithms for Implementing Distributed Shared Memory

3. The Read Replication Algorithm

- It replicates the data blocks that need to be accessed
- Here multiple nodes can read shared data at the same time
- However only one node can perform write operation at a time
- During write operation, all read operations are put on hold
- In the write operation, all copies of data blocks are updated to maintain consistency

Algorithms for Implementing Distributed Shared Memory

4. The Full Replication Algorithm

- It also replicates the data blocks that need to be accessed
- Here multiple nodes can read and write shared data at the same time
- Since several nodes perform write operation concurrently, the access to shared data must be controlled to maintain consistency

Issues in Load Distributing

- The workload of different computers in a distributed system may not be same
- This may be due to random arrival of tasks at various computers
- This can also occur due to the variation in processing capacity of different computers
- It is important to equally distribute the load for the timely completion of tasks

Issues in Load Distributing

1. Load

- CPU queue length and CPU Utilisation are good indicators of load in a computer
- CPU queue length is the number of threads that are ready, but unable to execute due to an active thread in the CPU
- CPU Utilisation is the percentage of time CPU is busy

Issues in Load Distributing

2. Classification of Load Distributing Algorithms

- There are three types of load distributing algorithms; static, dynamic and adaptive

(a) Static algorithms do not make use of current system state to take decision on load distribution; They perform load distribution based on some priori knowledge

(b) Dynamic algorithms make use of current system state to take decision on load distribution

(c) Adaptive algorithms are a special class of dynamic algorithms; They adapt their activities to suit changes in system state; Example - They may discontinue collecting system state information if the workload is too high

Issues in Load Distributing

3. Load Sharing Vs Load Balancing

- There are two types of load distributing algorithms based on their load distibuting principle

(a) Load Sharing Algorithms distribute load by transferring tasks from heavily loaded nodes to idle or lightly loaded nodes

(b) Load Balancing Algorithms attempt to equalise load at all nodes

Issues in Load Distributing

4. Preemptive Vs Non Preemptive Task Transfers

- Preemptive Task Transfers involve transfer of a task that is partially executed
- Here we have to transfer the task state also which includes process control block, file pointers, timers etc.
- Non Preemptive Task Transfers involve transfer of a task that has not begun its execution
- Here we need not transfer the task state

Components of a Load Distributing Algorithm

1. Transfer Policy
   This determines whether a node is in a suitable state to transfer a task
2. Selection Policy
   This determines which task is to be transferred
3. Location Policy
   This determines the node to which a task selected to be transferred
4. Information Policy
   This determines the information to be collected about system state

Components of a Load Distributing Algorithm

1. Transfer Policy

- This determines whether a node is in a suitable state to transfer a task
- Here we will be determining a threshold value T to the units of load that can be assigned to a node
- If load is greater than T, the node can be a sender of a task
- If load is less than T, the node can be a receiver of a task

Components of a Load Distributing Algorithm

2. Selection Policy

- This determines which task is to be transferred
- The simplest approach is to transfer the newly originated tasks that increases the load of a node beyond the threshold value
- We have to ensure that the overhead of task transfer is minimal
- Transferring tasks of smaller size cause less overhead

Components of a Load Distributing Algorithm

3. Location Policy

- This determines the node to which a task selected to be transferred
- One method is polling in which the sender node selects a suitable node for task transfer
- Another method is to broadcast a query for finding out if any node is available to share load

Components of a Load Distributing Algorithm

4. Information Policy

- This determines the information to be collected about system state
- It can be classified into 3 types

(a) Demand Driven - Here a node collects the state of other nodes only if it is a sender or receiver of tasks

(b) Periodic - Here a node collects information about the state of other nodes periodically

(c) State Change Driven - Here a node collects information about the state of other nodes, only if states change beyond a certain degree

Sender Initiated Algorithms

- Here the load distributing activity is initiated by the sender node which is overloaded
- The sender node transfers tasks to a receiver node which is underloaded
- The receiver node can be selected using 3 methods

1. Random - The receiver node is selected at random
- If the receiver node is heavily loaded, we have to again transfer the task to another node which is lightly loaded

2. Threshold - A node is selected at random; It is polled to find out whether it is a suitable one
- Otherwise polling is repeated for other nodes; The number of polls is limited by a parameter called poll limit

3. Shortest - A set of nodes are selected at random; They are polled to find out the node with the lightest load (shortest CPU queue length)
- The number of nodes selected for polling is limited by poll limit

Receiver Initiated Algorithms

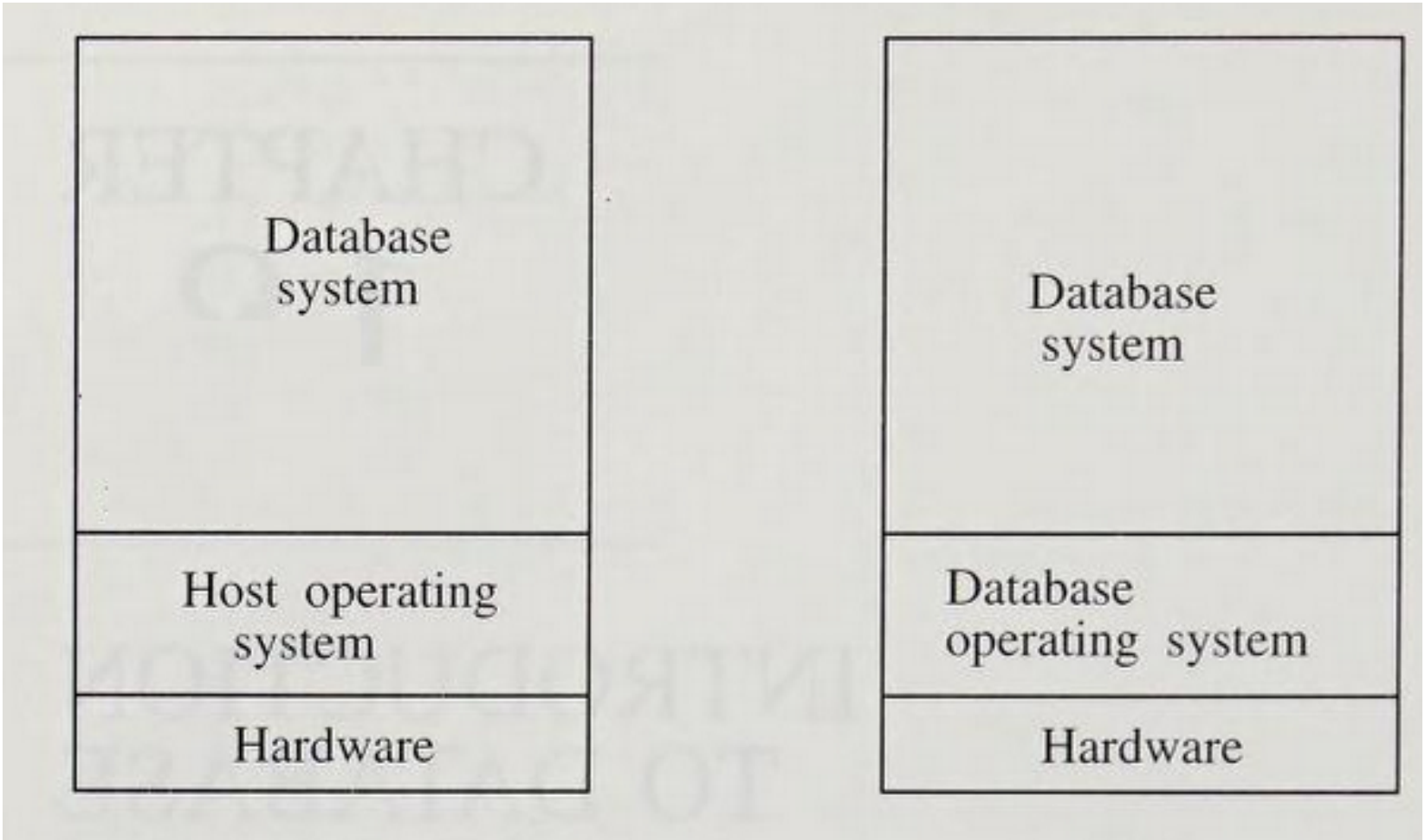
- Here the load distributing activity is initiated by the receiver node which is underloaded
- The receiver node retrieves tasks from a sender node which is overloaded
- The sender node can be selected using 3 methods

1. Random - The sender node is selected at random
   - If the sender node is lightly loaded, then task transfer will further reduce its load, which is not needed actually
2. Threshold - A node is selected at random; It is polled to find out whether it is a suitable one
   - Otherwise polling is repeated for other nodes; The number of polls is limited by a parameter called poll limit
3. Longest - A set of nodes are selected at random; They are polled to find out the node with the heaviest load (longest CPU queue length)
   - The number of nodes selected for polling is limited by poll limit

Module 5
Database Operating Systems

- There are two approaches for designing database systems
- Approach 1 - Implement a database system on top of general operating systems
- Its performance may not be best, since the requirements of a database system is different from a general purpose operating system
- Approach 2 - Build an operating system that efficiently supports only the functions needed by database systems
- It will have high performance
- The development of the entire OS from the scratch is expensive

Database System Design Approaches



Requirements of a Database OS

1. Transaction Management
   A database OS need to support properties of transactions such as atomicity, concurrency control and failure recovery
2. Support for Complex, Persistent Data
   Complex data types for storing image, video etc. should be supported
3. Buffer Management
   Buffers need to be maintained in main memory to cache the required data from disks

The Problem of Concurrency Control

- In a database system, several transactions execute simultaneously
- This leads to the following inconsistencies

1. Inconsistent Retrieval

- A transaction reading data items, before another transaction completes updating them
- Example- One Transaction reads balances of bank accounts before another one completes account transfer

2. Inconsistent Update

- Several transactions performing updation on the same set of data items concurrently
- Here the order of updation determines the final values of data items
- We need to control concurrent execution of transactions

Serialisability

- The execution sequence of read and write instructions of transactions is called a schedule(log)
- One way to control concurrent execution of transactions is to ensure the serialisability of concurrent schedules
- $T_1 = r_1[x], r_1[z], w_1[x]$
- $T_2 = r_2[y], r_2[z], w_2[y]$
- If a set of transactions are executed one after the other, then the corresponding execution sequence is called a serial schedule
- Example - $S_1 = r_1[x], r_1[z], w_1[x], r_2[y], r_2[z], w_2[y]$
- If a set of transactions are executed concurrently, then the corresponding execution sequence is called a concurrent schedule
- Example - $S_2 = r_1[x], r_1[z], r_2[y], r_2[z], w_1[x], w_2[y]$
- A concurrent schedule is said to be serialisable, if it is equivalent to a serial schedule

Serialisability

- For checking the serialisability, we will try to transform the concurrent schedule to a serial schedule by a series of swaps of nonconflicting instructions
- Two instructions conflict (i) if they are operations by different transactions on the same data item and (ii) at least one of these instructions is a write operation
- Now we will check whether S_2 is serialisable
- $S_2 = r_1[x], r_1[z], r_2[y], r_2[z], w_1[x], w_2[y]$ (Concurrent Schedule)
- swap $r_2[z]$ and $w_1[x]$
- $r_1[x], r_1[z], r_2[y], w_1[x], r_2[z], w_2[y]$
- $r_1[x], r_1[z], r_2[y], w_1[x], r_2[z], w_2[y]$
- swap $r_2[y]$ and $w_1[x]$
- $r_1[x], r_1[z], w_1[x], r_2[y], r_2[z], w_2[y]$ (Serial Schedule)
- Hence S_2 is serialisable

Basic Synchronisation Primitives for Concurrency Control

- The following synchronisation primitives are used for concurrency control in a database system

1. Locks

- In lock based algorithms each data item has a lock associated with it
- A transaction can perform an operation on a data item only after acquiring a lock on it
- A transaction can lock a data item in 2 modes

a. Shared Mode Lock

- Here the transaction can only perform read operation on the data item
- Multiple transactions can hold shared mode lock on the same data item at a time

b. Exclusive Mode Lock

- Here the transaction can perform both read and write operation on the data item
- No other transaction can hold a lock on the same data item at a time

Basic Synchronisation Primitives for Concurrency Control

2. Timestamps

- A timestamp is a unique number associated with a transaction or a data item
- The value of a timestamp increases with time
- We can use either the value of system clock or a logical counter as the timestamp
- For achieving concurrency control, transactions are ordered on their timestamps

Lock Based Algorithms

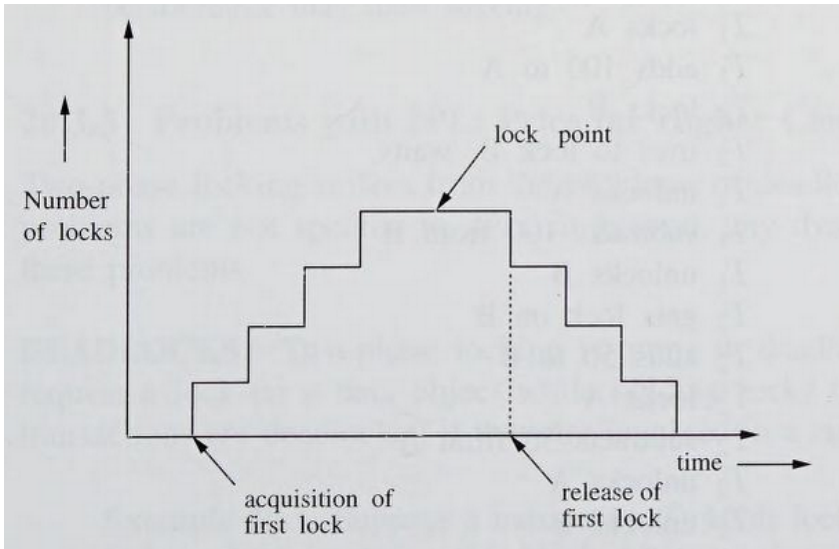
- Two Phase Locking Algorithm
- Here Each Transaction issues lock and unlock requests in two phases

1. Growing Phase

- A transaction obtains locks, but does not release any lock

2. Shrinking Phase

- A transaction releases locks, but does not obtain any new lock
- The growing phase is followed by shrinking phase
- The point where a transaction has obtained its final lock (the end of its growing phase) is called its lock point



Lock Based Algorithms

- In this algorithm, a transaction can request for lock on other data items, without releasing the locks which are currently held
- This can lead to deadlock, where multiple transactions circularly wait for accessing data items
- The deadlock is resolved by rolling back one of the transactions
- In this process all locks held by it are released and all the data items modified by it are restored to their original state
- Here we might have to also rollback other transactions which have already accessed the values of these data items
- This is called a cascading rollback

Lock Based Algorithms

- A cascading rollback is avoided by following the given below variations of Two Phase Locking Algorithm
- Strict Two Phase Locking Algorithm
- Here all the exclusive mode locks are held until a transaction completes
- Rigorous Two Phase Locking Algorithm
- Here all the locks are held until a transaction completes

Lock Based Algorithms

- Wait-Die Algorithm

  Suppose transaction T_i requests for a data item held by transaction T_j . If T_i is older it waits, otherwise T_i dies(is rolled back)
- Wound-Wait Algorithm

  Suppose transaction T_i requests for a data item held by transaction T_j . If T_i is older it wounds (rolls back) T_j , otherwise T_i waits.
- Comparison

1. Waiting Time

   In Wait-Die Algorithm, it is more for older requesting transactions

   In Wound-Wait Algorithm, it is more for younger requesting transactions
2. Number of Restarts

   In Wait-Die Algorithm, it is more for younger requesting transactions

   In Wound-Wait Algorithm, it is less for younger requesters

Timestamp Based Algorithms

- Here a unique fixed timestamp TS(T_i) is assigned to every transaction in the system
- This timestamp is assigned when the transaction begins its execution
- If T_i begins its execution before another Transaction T_j , then TS(T_i) $<$ TS(T_j)
- Associated with each data item Q there are 2 timestamp values
- R-timestamp(Q) denotes the largest timestamp of any transaction that executed read(Q) successfully
- W-timestamp(Q) denotes the largest timestamp of any transaction that executed write(Q) successfully

Timestamp Based Algorithms

- Timestamp Ordering Algorithm
- In this algorithm, any conflicting read and write operations are executed in timestamp order

1. Suppose that a transaction T_i issues read(Q)

(a) If TS(T_i) $<$ W-timestamp(Q), read operation is rejected and transaction is rolled back

(b) Otherwise, read operation is executed and R-timestamp(Q) is set to Max(R-timestamp(Q),TS(T_i))

2. Suppose that a transaction T_i issues write(Q)

(a) If TS(T_i) $<$ R-timestamp(Q), write operation is rejected and transaction is rolled back

(b) If TS(T_i) $<$ W-timestamp(Q), write operation is rejected and transaction is rolled back

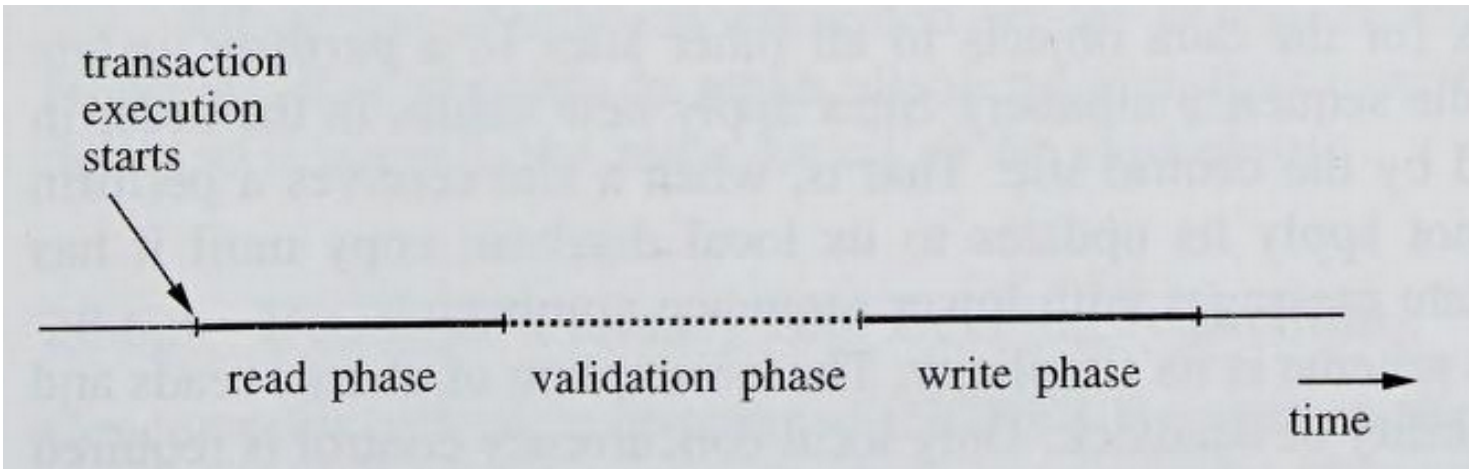
(c) Otherwise, write operation is executed and W-timestamp(Q) is set to TS(T_i)

Optimistic Algorithms

- They are based on the assumption that conflicts are rare in transactions
- Here conflicts are checked only at the end of transactions
- If any conflict is found, the corresponding transaction is rolled back
- These algorithms can be used if most of the operations are read only

Optimistic Algorithms

- Kung-Robinson Algorithm
- A transaction has 3 phases
- read phase - Here data items are read, computation is performed and results are written to temporary storage
- validation phase - Here we will check whether any of the writes violates the consistency of the database; if there is any, the transaction is rolled back
- write phase - If all writes are validated, the values are finally written to the database



Module 4
Multiprocessor Operating Systems

- A multiprocessor system consists of several processors that share common physical memory
- A multiprocessor operating system controls all the processors in a multiprocessor system
- Advantages of a multiprocessor system

1 Enhanced Performance

- Several tasks / subtasks can be executed concurrently in different processors, resulting in improved performance

2 Fault Tolerance

- The failure of a processor will not terminate the system, since the execution can be continued in other processors

Basic Multiprocessor System Architectures

- Based on the accessibility of main memory by processors, there are 2 architectures

1. Tightly Coupled Architecture

- All processors share the same main memory
- Example - Multimax of Encore Corporation

2. Loosely Coupled Architecture

- Main memory is partitioned into several modules and each module is assigned to a processor
- A processor cannot directly access memory module of another one
- To access other's memory, message passing is done
- Example - Intel's Hypercube

Basic Multiprocessor System Architectures

- Based on the vicinity and accessibility of main memory by processors, there are 3 architectures

1 Uniform Memory Architecture(UMA)

- All processors share the same main memory, which is equidistant from all
- Access time is same for all processors
- Example - Multimax of Encore Corporation

2 Non Uniform Memory Architecture(NUMA)

- Main memory is partitioned into several modules and each module is assigned to a processor
- A processor can access memory module of another one with a certain delay in access time
- Example - Butterfly Machine of BBN Laboratories

Basic Multiprocessor System Architectures

- Based on the vicinity and accessibility of main memory by processors, there are 3 architectures

3. NO Remote Memory Access(NORMA)

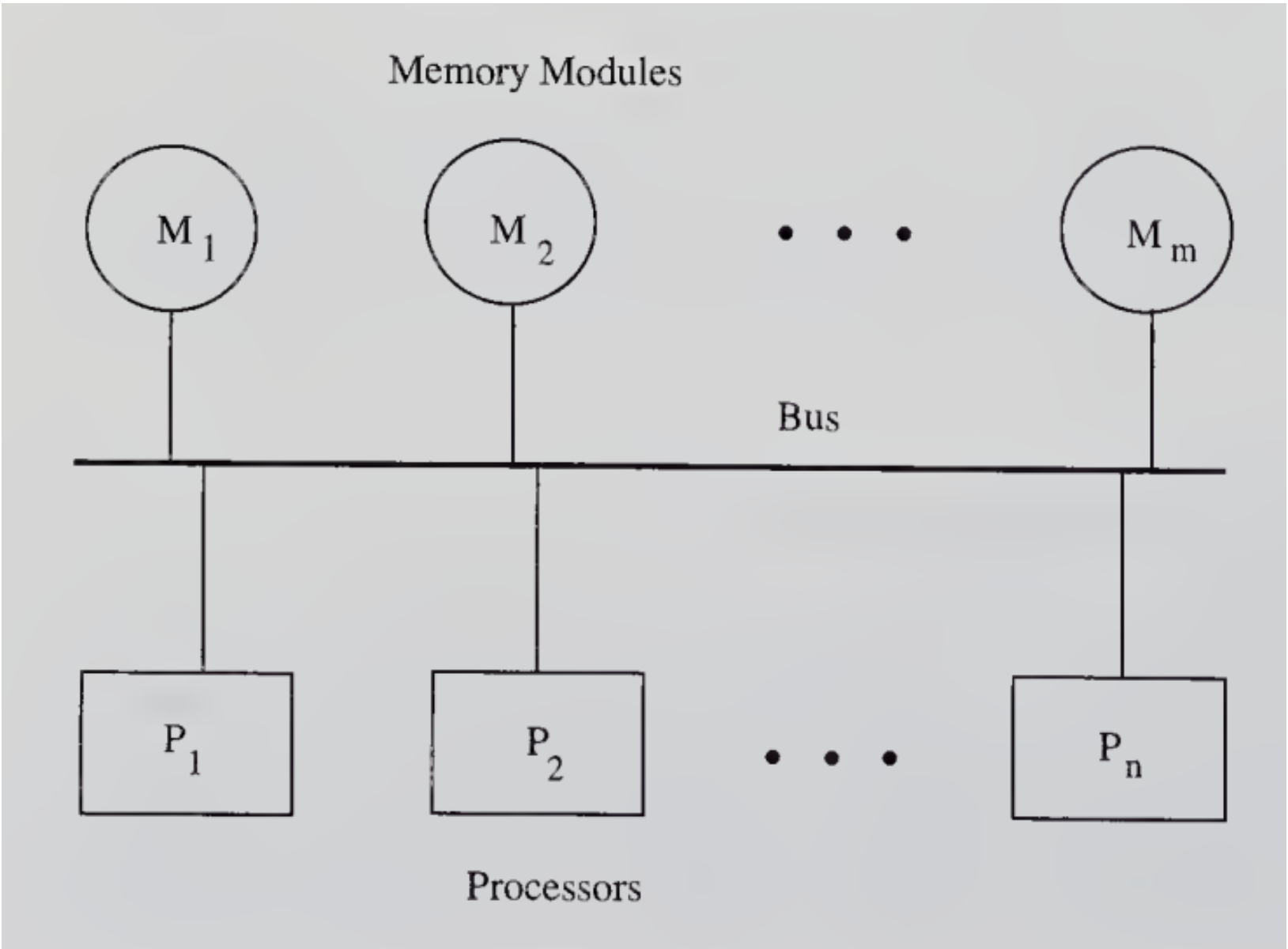
- Main memory is partitioned into several modules and each module is assigned to a processor
- A processor cannot directly access memory module of another one
- To access other's memory, message passing is done
- Example - Intel's Hypercube

Interconnection Networks for Multiprocessor Systems

1. Bus
2. Cross Bar Switch
3. Multistage Interconnection Network

Interconnection Networks for Multiprocessor Systems

1. Multiprocessor System with a Bus



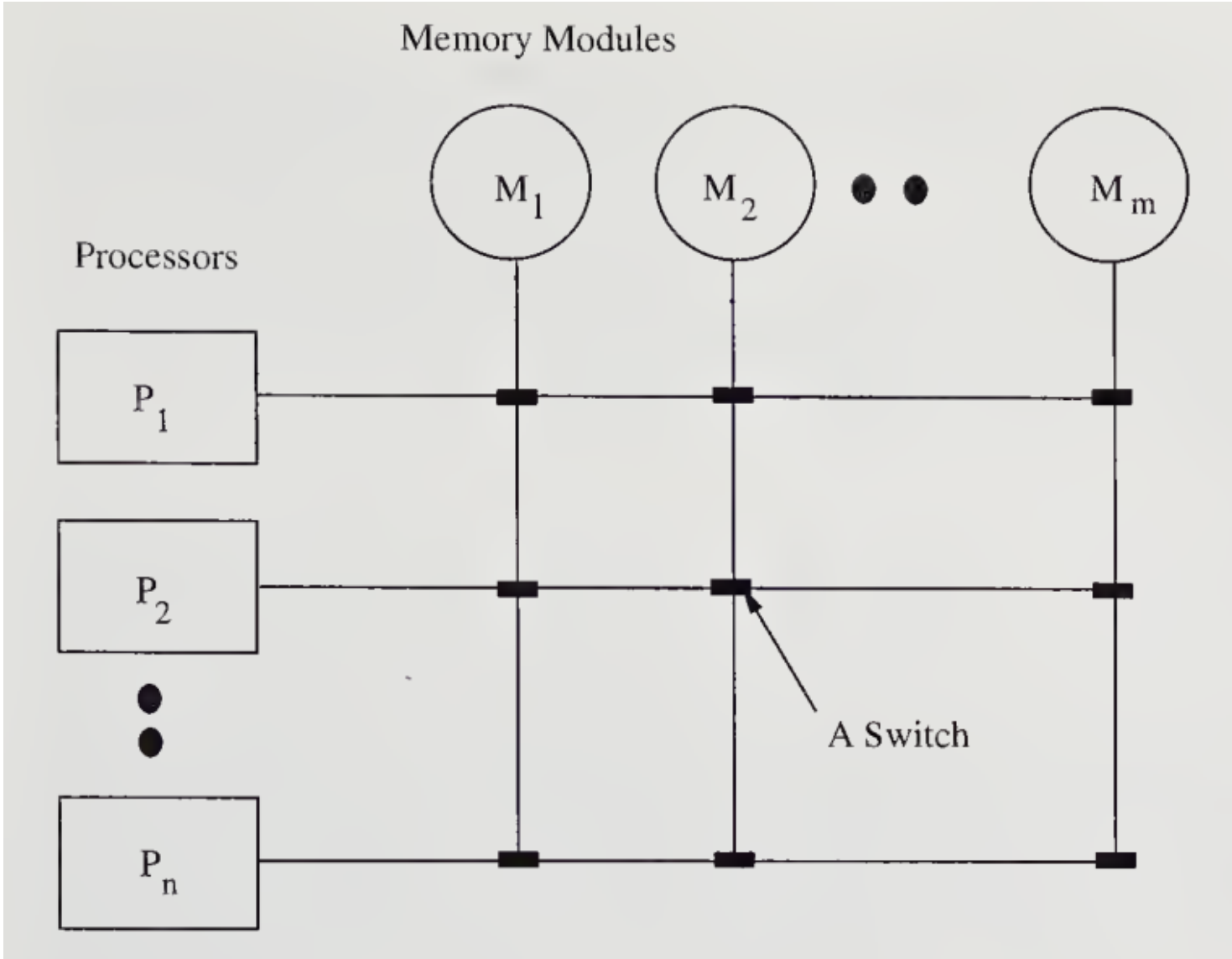
Interconnection Networks for Multiprocessor Systems

1. Multiprocessor System with a Bus

- This is the simplest multiprocessor system architecture
- It is relatively inexpensive
- However, the bus can support only one processor-memory communication at any time
- This architecture can support only a limited number of processors

Interconnection Networks for Multiprocessor Systems

2. Multiprocessor System with a Cross Bar



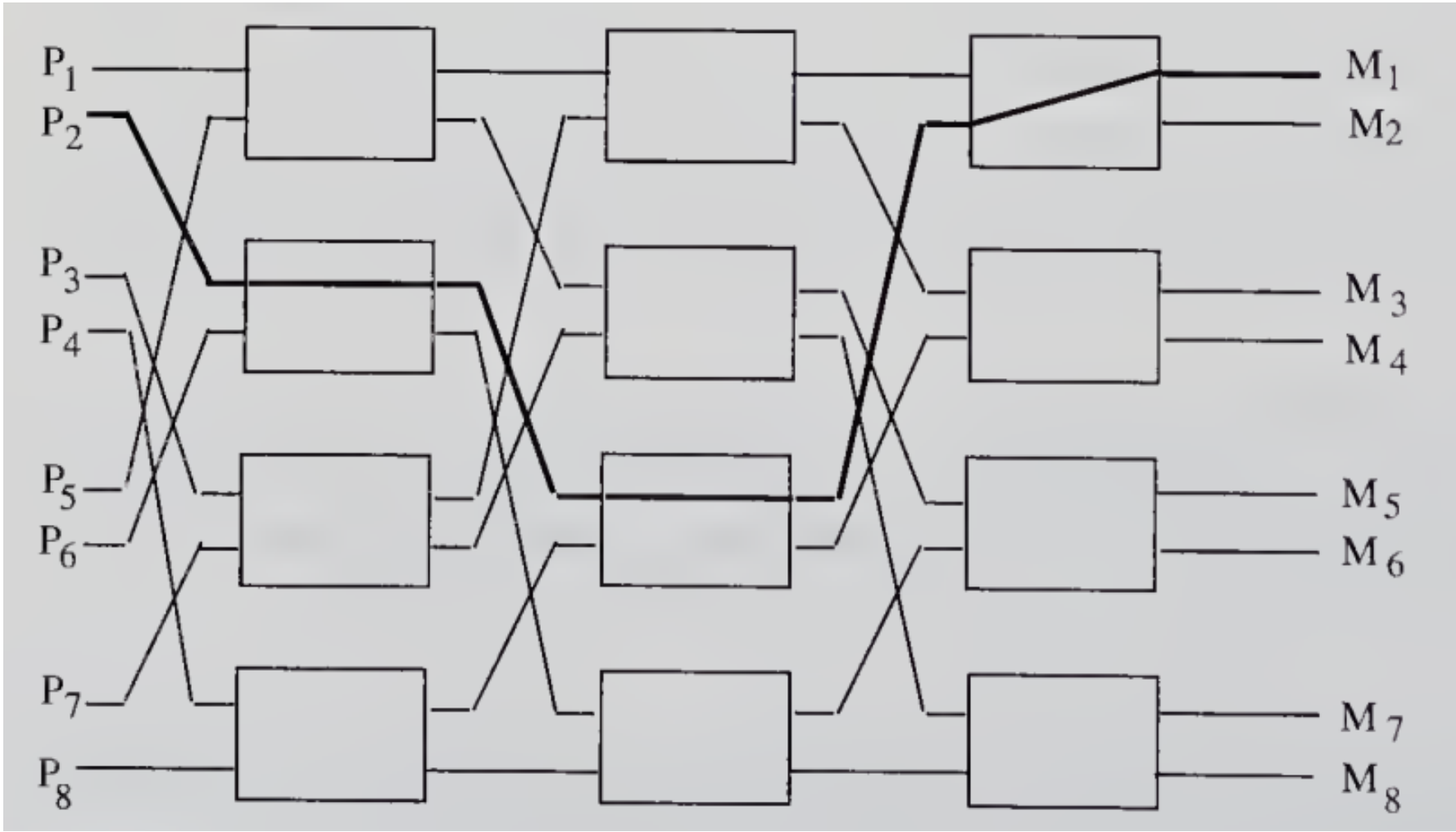
Interconnection Networks for Multiprocessor Systems

2. Multiprocessor System with a Cross Bar

- This is a matrix that has a switch at every cross-point
- This is capable of providing an exclusive connection between any processor -memory pair
- All processors can concurrently access memory modules provided that each processor is accessing a different memory module
- It is relatively expensive

Interconnection Networks for Multiprocessor Systems

3. Multistage Interconnection Network



Interconnection Networks for Multiprocessor Systems

3. Multistage Interconnection Network

- It is a compromise between a bus and a cross-bar switch
- It also permits simultaneous connections between several processor-memory pairs
- It consists of several stages of switches
- The outputs of the switches in a stage is connected to the inputs of the switches in the next stage
- There exists a unique path between a processor-memory pair
- It is more cost-effective than a cross-bar

Structures of Multiprocessor Operating Systems

1. The Separate Supervisor Configuration

- A supervisor is the most important program in the operating system
- It manages the entire operating system by loading other OS code into memory
- Here each processor will be having its own copy of the supervisor
- Processors are having greater autonomy
- However concurrent execution of a single task will be difficult

Structures of Multiprocessor Operating Systems

2. The Master - Slave Configuration

- Here one processor is designated as the master processor
- It is responsible for controlling and coordinating the execution of other processors, called slaves
- Here the master processor can only execute the supervisor program
- Here, concurrent execution of a single task can be achieved
- However, the failure of master processor leads to problems
- Example - Cyber 170

3. The Symmetric Configuration

- Here, a single copy of the supervisor is shared by all processors
- At a time, only one processor executes the supervisor program
- All processors are autonomous
- Here also, concurrent execution of a single task can be achieved
- We need to control concurrency due to the autonomy provided
- Example - Hydra

Multiprocessor Operating System Design Issues

1. Threads

- Multiple Threads provide concurrency in a single processor system
- Multiple Threads provide parallelism in a multiprocessor system
- User Level Threads - Implemented by users
- Kernel Level Threads - Implemented by OS
- Challenges in Design

a. Divide and balance the work in various threads
b. Divide the data between different threads
c. Identifying data dependencies between threads
d. Testing and Debugging is also challenging

Multiprocessor Operating System Design Issues

2. Process Synchronisation

- In a uniprocessor system only one thread will be executing at a time
- In a multiprocessor system several threads will be executing at several processors
- The critical section problem is more complex here
- We have to ensure that only one thread executes in its critical section at a time

Multiprocessor Operating System Design Issues

3. Processor Scheduling

- In a multiprocessor system several tasks are allocated to several processors
- Issues
- Preemption inside critical sections
- This occurs when a task is preempted in its critical section, by other requesting tasks
- Cache Corruption
- When a processor switches from one task to another, we need to make large number of changes in cache, if these tasks belong to different applications
- This will cause a very high cache miss ratio, which is also called cache corruption
- Context Switching Overheads
- Context switching involves storing the intermediate state of process / thread for retrieving later
- All the data structures associated with a process need to be stored; context switching is more; overhead is more

Multiprocessor Operating System Design Issues

4. Memory Management

- Here we will study about virtual memory management
- The design of virtual memory is complicated, since the main memory is shared by several processors
- Portability
- It is the ability of the OS to run on machines with different architectures
- The virtual memory system heavily relies on the system architecture
- Hence, achieving architecture independence is an important goal of virtual memory design
- Data Sharing
- A virtual memory system must provide facility for data sharing to support the execution of parallel programs

Multiprocessor Operating System Design Issues

4. Memory Management

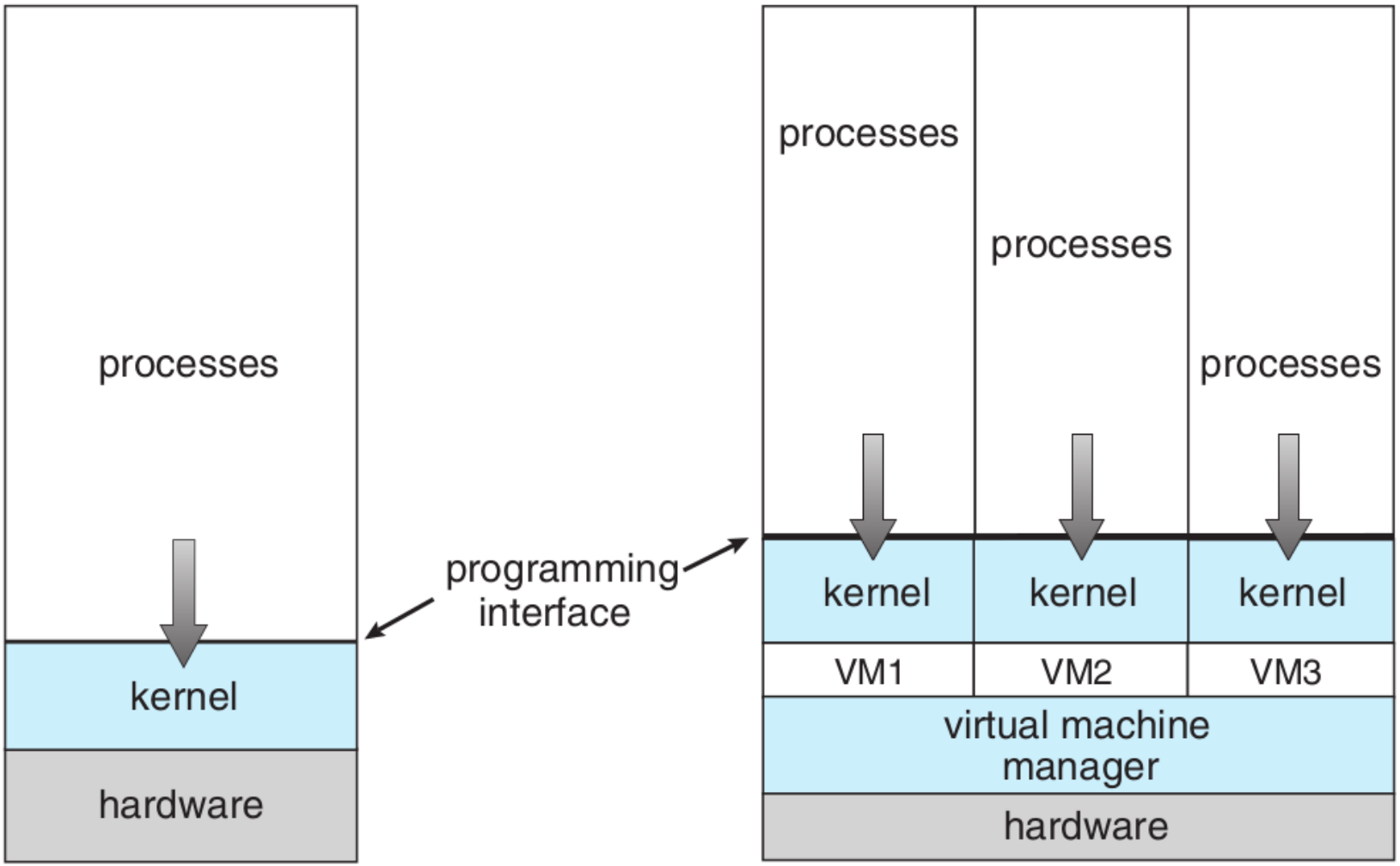
- Here we will study about virtual memory management
- The design of virtual memory is complicated, since the main memory is shared by several processors
- Protection
- Since memory is shared among various processes, memory protection is an important requirement
- The virtual memory system need to protect memory objects from unauthorised access
- Efficiency
- The virtual memory system must be efficient in address translations, page table look-up and page replacements

Virtualisation

- It is the process of building a virtual machine within a host computer - personal computer or remote server
- A virtual machine is a virtual version of the operating system (also called guest OS) installed within a host computer
- All the resources needed for this OS like CPU, Memory and Storage are borrowed from the host computer
- The virtual machine is partitioned from the rest of the system
- The software inside a virtual machine cannot interfere with the host computer's primary operating system (also called host OS).
- Example - Using the virtualisation software VirtualBox, we can load multiple guest OSes under a single host OS

Virtualisation

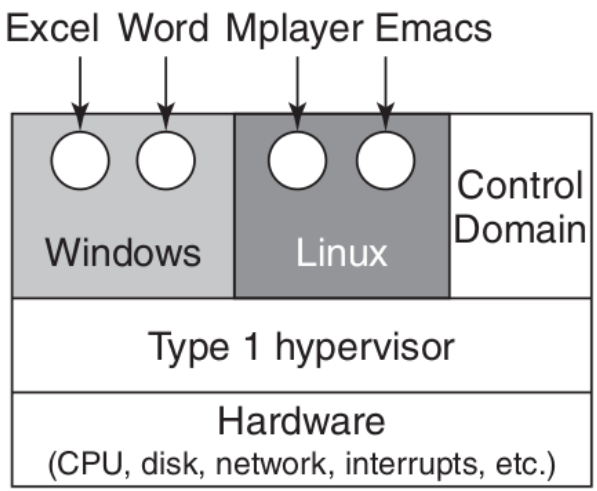
- Non Virtual Machine and Virtual Machine

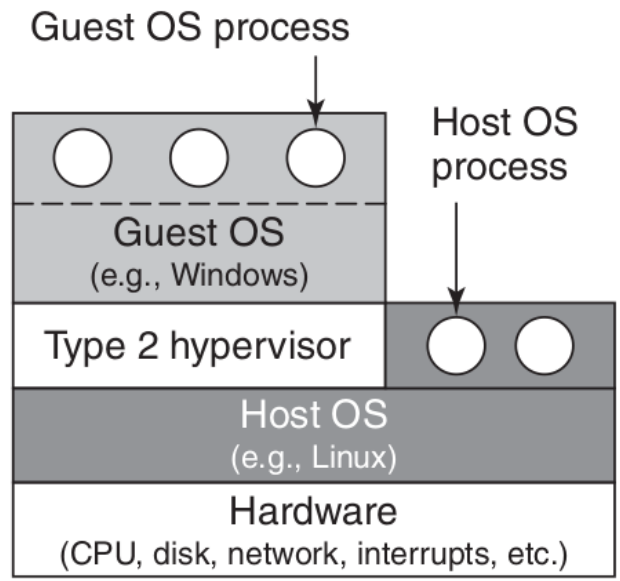


- The layer of software used for managing virtual machines is called Virtual Machine Manager(VMM) (Hypervisor)

Types of Hypervisors

- Type 1 Hypervisor
- A Hypervisor installed directly on top of the hardware
- There is no OS or software in between
- Example - VMware ESX
- Type 2 Hypervisor
- A Hypervisor installed on top of the guest OS
- Example - Oracle VirtualBox





ParaVirtualisation

- There are 2 ways of building a virtual machine - full virtualisation and paravirtualisation
- In the case of full virtualisation, the guest operating system runs independently
- In other words it cannot interfere with the hypervisor
- In contrast, paravirtualisation involves the interaction of the guest os with the hypervisor using hypercalls
- paravirtualisation is more secure compared to full virtualisation
- The performance of paravirtualisation is better compared to full virtualisation

Memory Virtualisation

- Here we will see how to implement virtual memory in virtual machines
- Virtual memory is basically a mapping of pages in virtual address space to pages in physical memory
- This mapping is defined by page tables
- Assume that a virtual machine has mapped its virtual pages 3, 4 and 7 onto physical pages 10, 11, and 12, respectively
- Now a second virtual machine in the system tries to map its virtual pages 4, 5 and 6 onto physical pages 10, 11, and 12, respectively
- The hypervisor cannot allow this, since these physical pages are already in use
- It has to find some other free pages
- In general, for each virtual machine the hypervisor needs to create a shadow page table that maps the virtual pages used by the virtual machine onto the actual pages the hypervisor gave it

I/O Virtualisation

- I/O Virtualisation is a technology that allows multiple virtual machines in a system to share storage devices and I/O devices.
- In this approach, physical devices are not directly allocated to virtual machines
- Instead, virtual versions of these devices are created, which can be allocated to virtual machines
- In this method, a single device can be allocated to multiple virtual machines
- A single device appears as multiple separate devices to each virtual machine

References

1. Mukesh Singhal and Niranjan G. Shivaratri, "Advanced Concepts in Operating Systems – Distributed, Database, and Multiprocessor Operating Systems", Tata McGraw-Hill, 2001
2. Silberschatz, Galvin and Gagne, Operating System Concepts, 10th Ed, Wiley, 2018
3. Andrew S. Tanenbaum, "Modern Operating Systems", 3rd Edition, Prentice Hall, 2012.
4. `https://www.tutorialspoint.com`
5. https://www.javatpoint.com/full-virtualization-vs-paravirtualization-in-operating-system