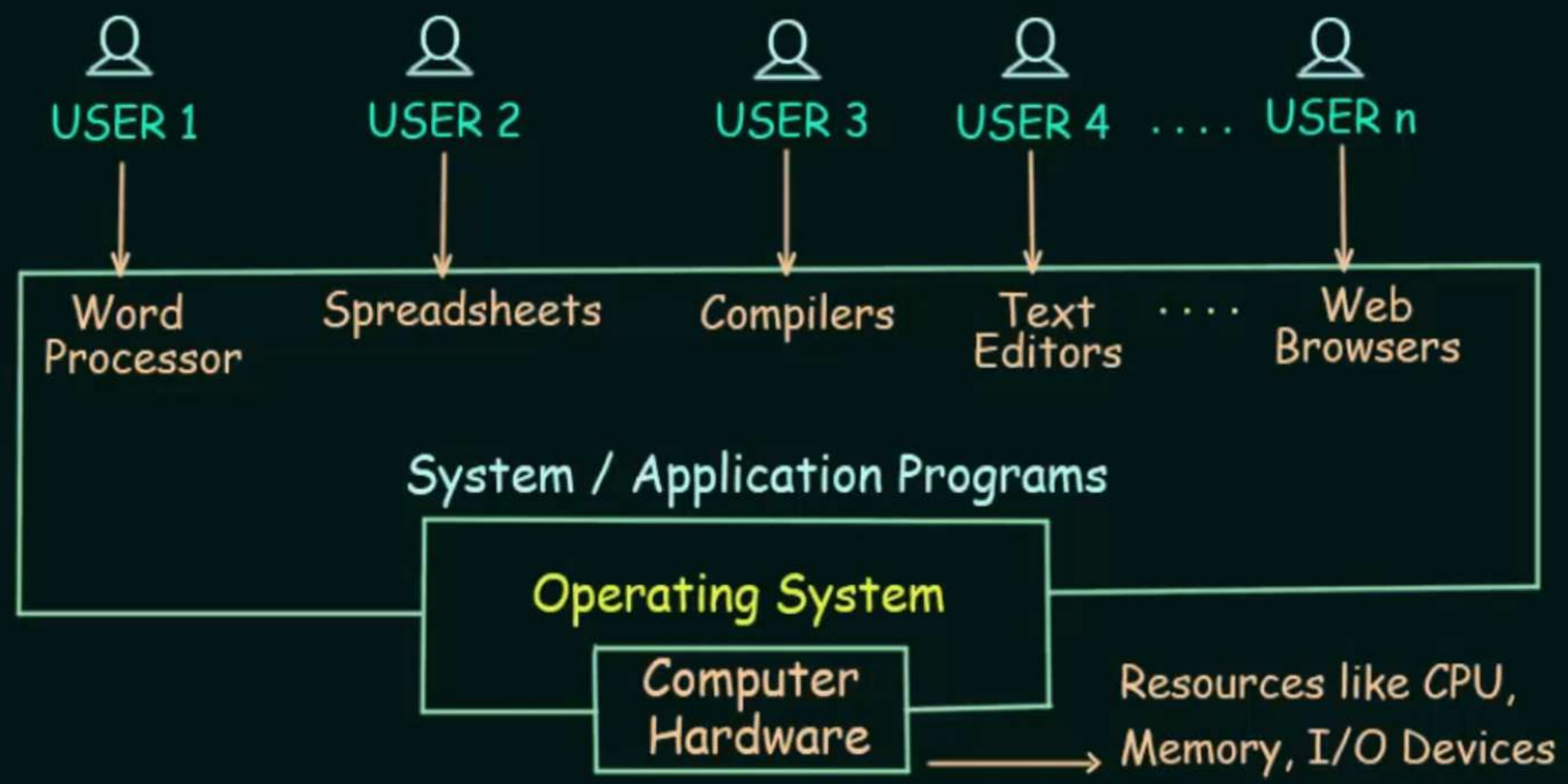
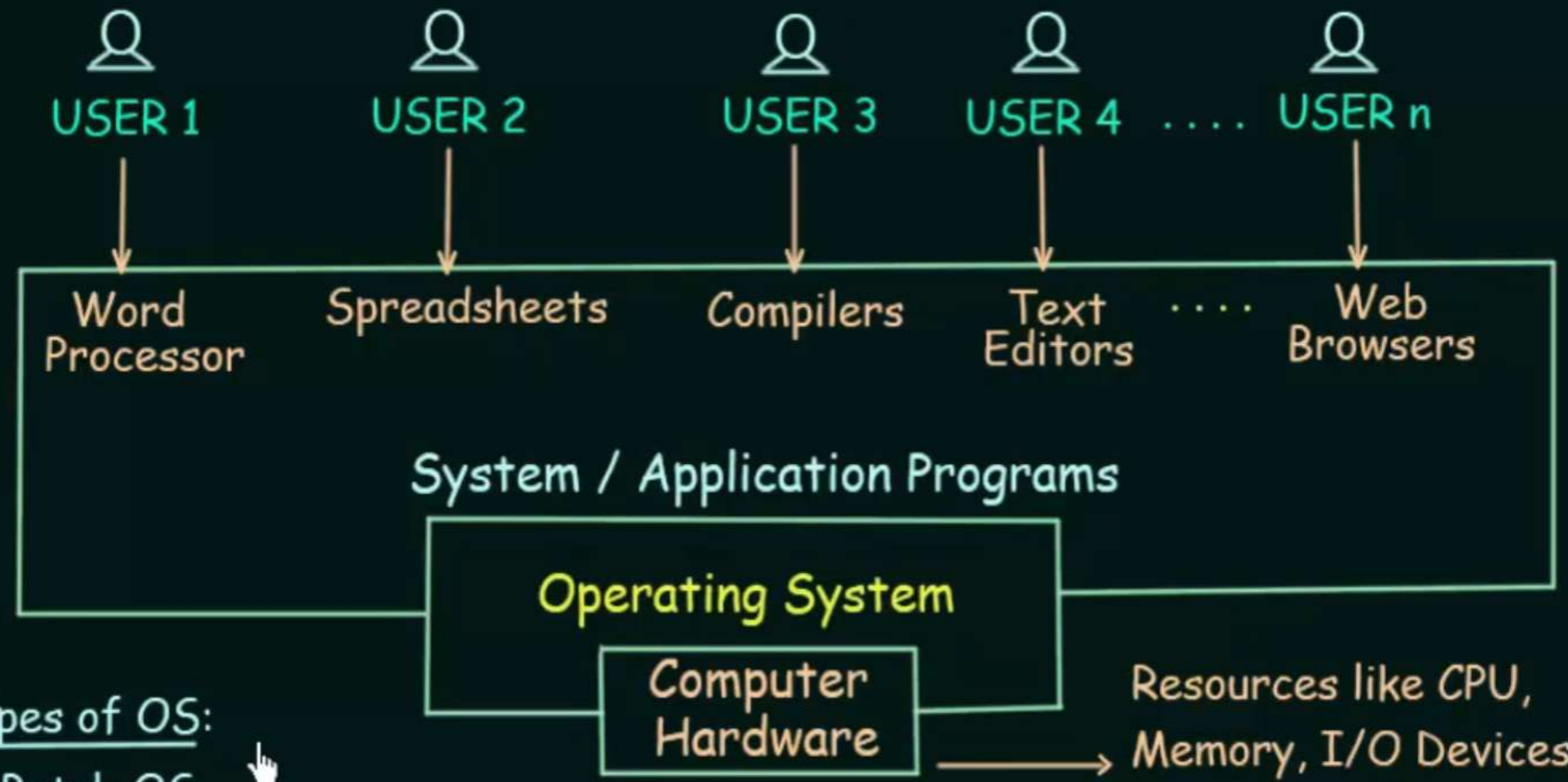
Windows Linux Ubuntu Mac OS X iOS Android

→ An Operating System (OS) is a program that manages the computer hardware.

→ It also provides a basis for Application Programs and acts as an intermediary between computer User and compter Hardware.





Resources like CPU, Memory, I/O Devices

Types of OS:

- → Batch OS
- → Time sharing OS
- → Distributed OS
- → Network OS
- → Real Time OS
- → Multi Programming/ Processing/ Tasking OS

Goals of OS:

i) Convenience
ii) Efficiency
iii) Both

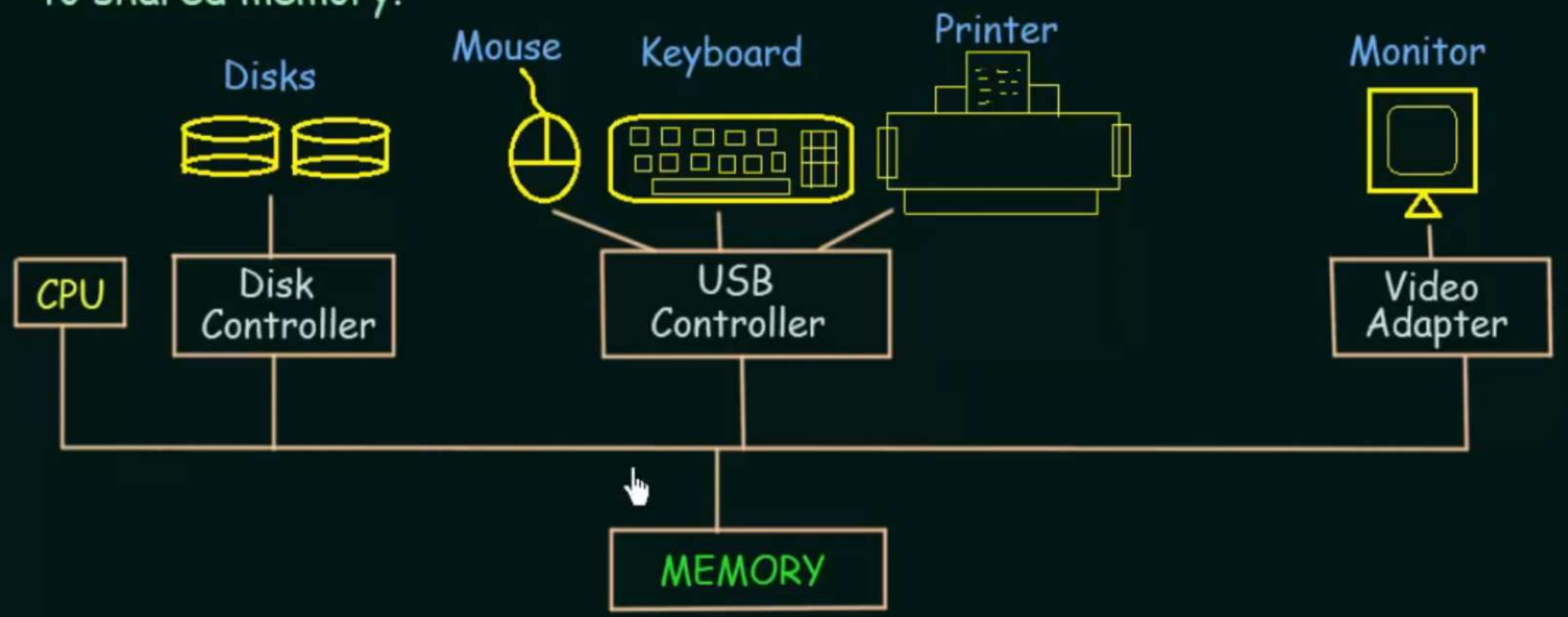
Functions of OS:

- → It is an interface between User & Hardware
- → Allocation of Resources
- → Management of Memory, Security, etc.

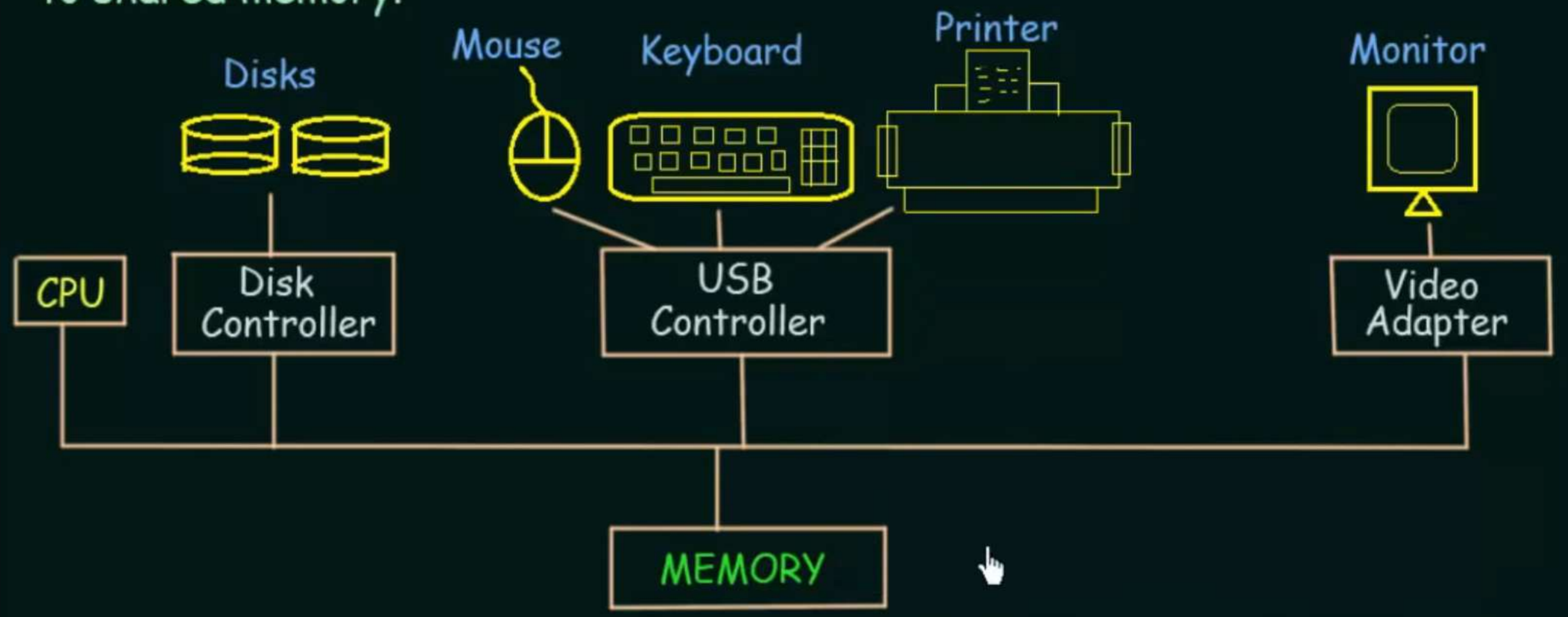
Basics of Operating System (Computer System Operation)

Some basic knowledge of the structure of Computer System is required to understand how Operating Systems work.

→ A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory.



a number of device controllers connected through a common bus that provides access to shared memory.



→ Each device controller is in charge of a specific type of device

→ The CPU and the device controllers can execute concurrently, competing for memory cycles

→ To ensure orderly access to the shared memory, a memory controller is provided whose function is to synchronize access to the memory

→To ensure orderly access to the shared memory, a memory controller is provided whose function is to synchronize access to the memory

Some important terms:

1) Bootstrap Program:→The initial program that runs when a computer is powered up or rebooted.
→It is stored in the ROM.
→It must know how to load the OS and start executing that system.
→It must locate and load into memory the OS Kernel.

2) Interrupt: → The occurence of an event is usually signalled by an Interrupt from Hardware or Software.
→ Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by the way of the system bus.

3) System Call (Monitor call): → Software may trigger an interrupt by executing a special operation called System Call.

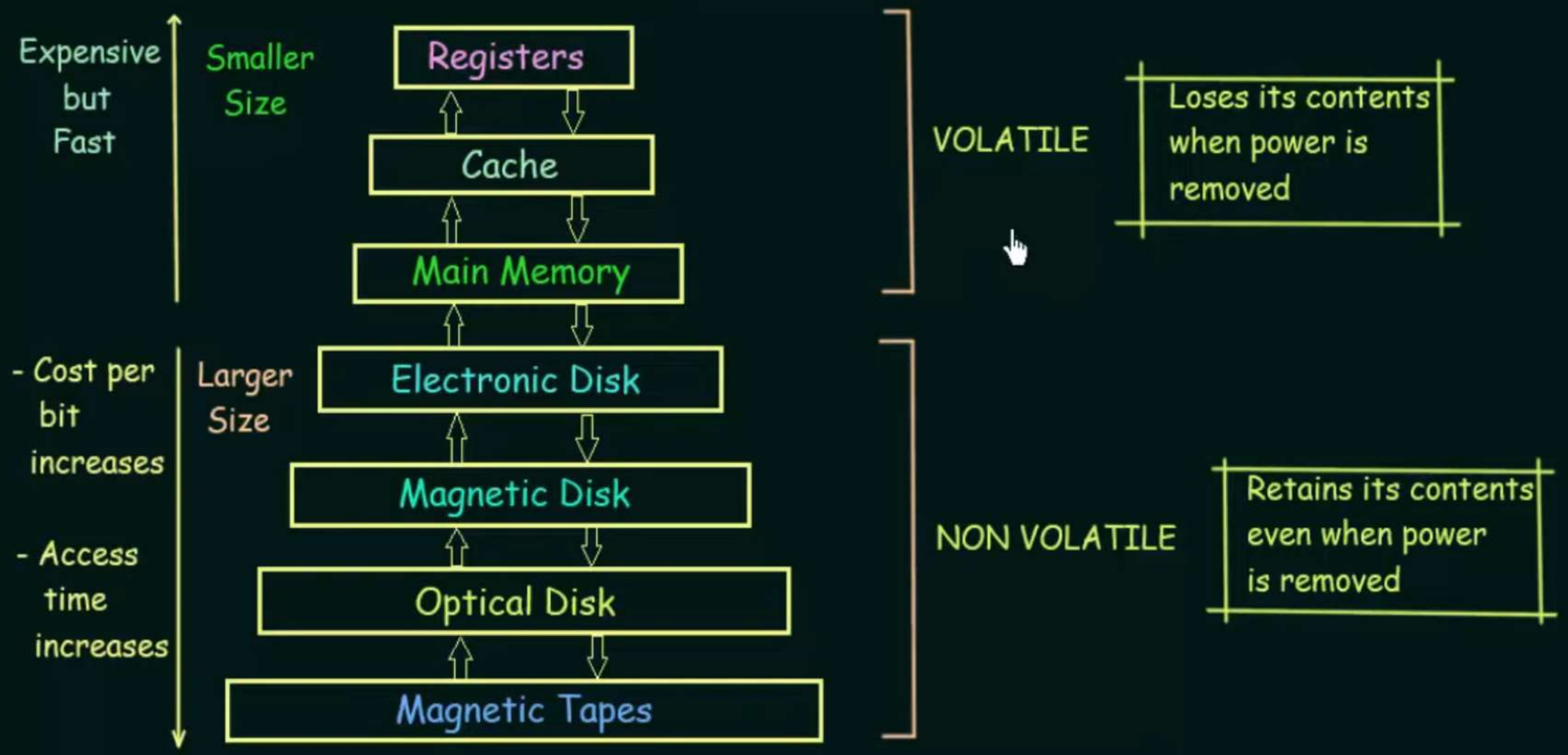
When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location.

→ The fixed location usually contains the starting address where the Service Routine of the interrupt is located.

The Interrupt Service Routine executes.

On completion, the CPU resumes the interrupted computation.

Basics of Operating System (Storage Structure)



Basics of Operating System (I/O Structure)

-> Storage is only one of many types of I/O devices within a computer

-> A large portion of operating system code is dedicated to managing I/O, both because of its importance to the reliability and performance of a system and because of the varying nature of the devices

-> A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus

-> Each device controller is in charge of a specific type of device

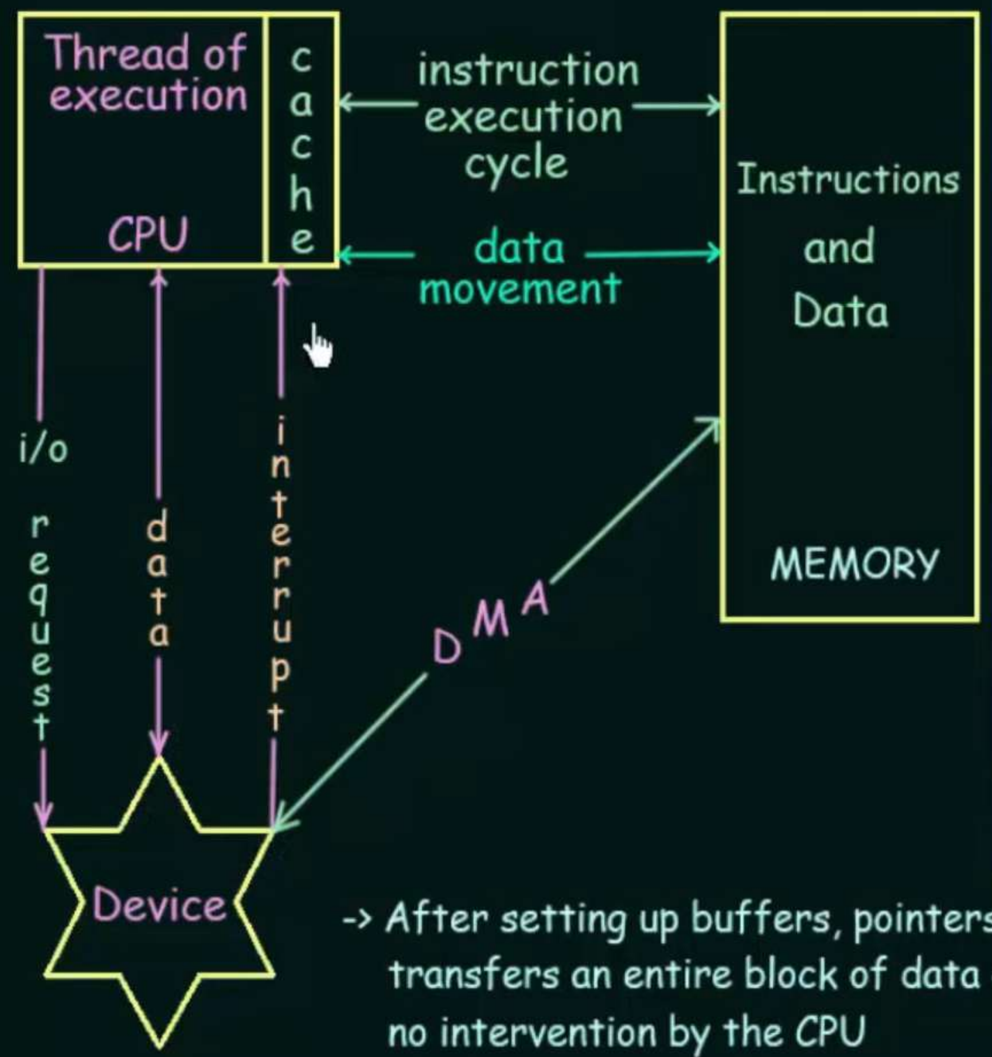
maintains

- Local Buffer Storage
- Set of Special Purpose Registers

-> Typically, operating systems have a device driver for each device controller

-> This device driver understands the device controller and presents a uniform interface to the device to the rest of the operating system

Working of an I/O Operation:



-> To start an I/O operation, the device driver loads the appropriate registers within the device controller

-> The device controller, in turn, examines the contents of these registers to determine what action to take

-> The controller starts the transfer of data from the device to its local buffer

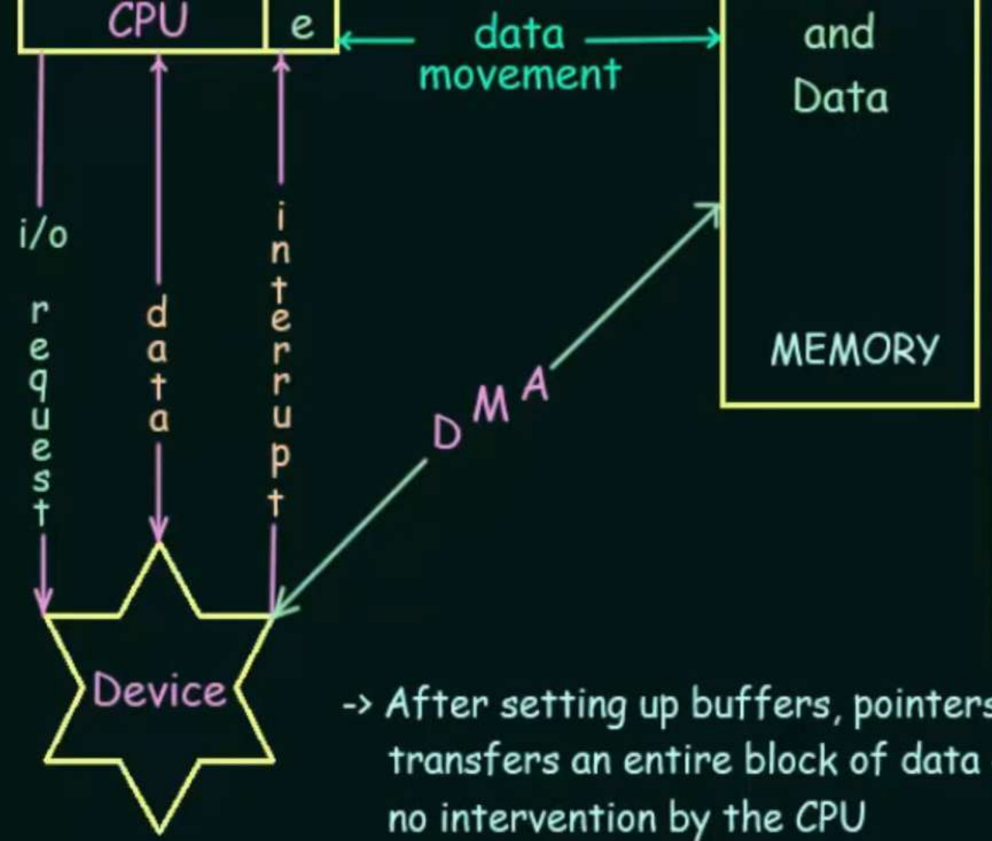
-> Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation

-> The device driver then returns control to the operating system

This form of interrupt-driven I/O is fine for moving small amounts of data but can produce high overhead when used for bulk data movement

To solve this problem, Direct Memory Access (DMA) is used

-> After setting up buffers, pointers, and counters for the I/O device, the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU



-> The controller starts the transfer of data from the device to its local buffer

-> Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation

-> The device driver then returns control to the operating system

This form of interrupt-driven I/O is fine for moving small amounts of data but can produce high overhead when used for bulk data movement

To solve this problem, Direct Memory Access (DMA) is used

-> After setting up buffers, pointers, and counters for the I/O device, the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU

-> Only one interrupt is generated per block, to tell the device driver that the operation has completed

-> While the device controller is performing these operations, the CPU is available to accomplish other works

Computer System Architecture

Types of Computer Systems based on number of General Purpose Processors:

1. Single Processor Systems

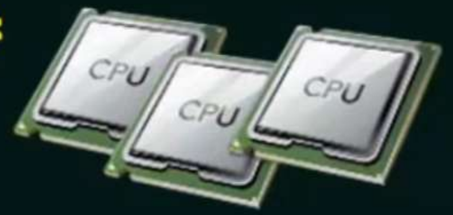
2. Multiprocessor Systems

3. Clustered Systems

1. Single Processor Systems:

- One main CPU capable of executing a general purpose instruction set including instructions from user processes.
- Other special purpose processors are also present which perform device specific tasks

2. Multiprocessor Systems:



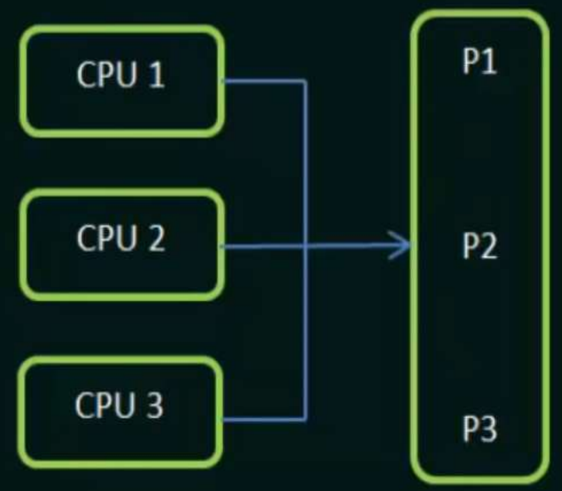
- Also known as parallel systems or tightly coupled systems.
- Has two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices

Advantages:

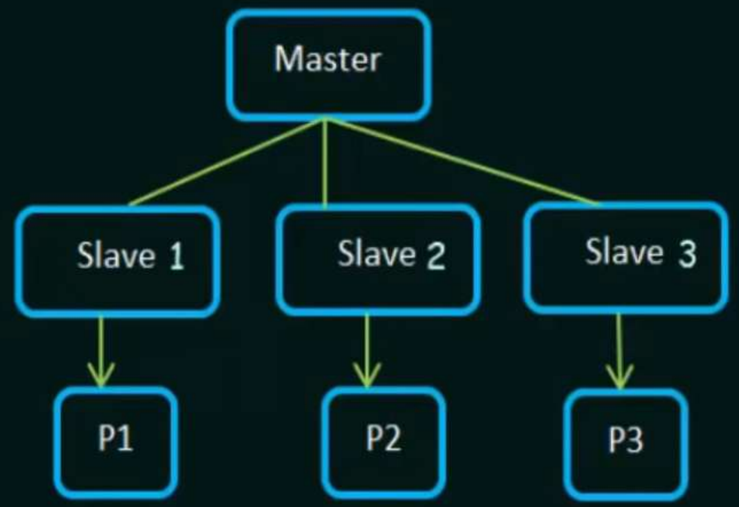
- 👍 Increased throughput
- 👍👍 Economy of scale
- 👍👍👍 Increased reliability

Types of Multiprocessor Systems:

Symmetric Multiprocessing



Asymmetric Multiprocessing



3. Clustered Systems

- Like multiprocessor systems, clustered systems gather together multiple CPUs to accomplish computational work.
- They are composed of two or more individual systems coupled together.
- Provides high availability
- Can be structured asymmetrically or symmetrically

asymmetrically

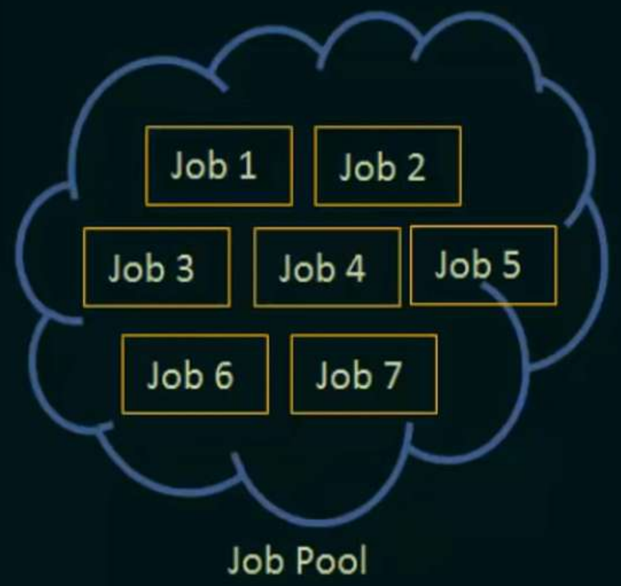
- One machine in Hot-Standby mode
- Others run applications

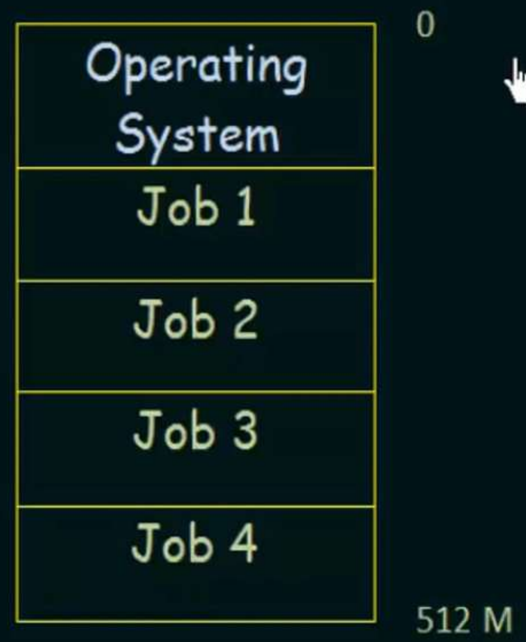
symmetrically

- Two or more hosts run applications
- Monitors each other

(i) Multiprogramming

- A single user cannot, in general, keep either the CPU or the I/O devices busy at all times
- Multiprogramming increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute.





Memory layout for a multiprogramming system

Multiprogrammed systems provide an environment in which the various system resources (for example, CPU, memory, and peripheral devices) are utilized effectively, but they do not provide for user interaction with the Computer system.

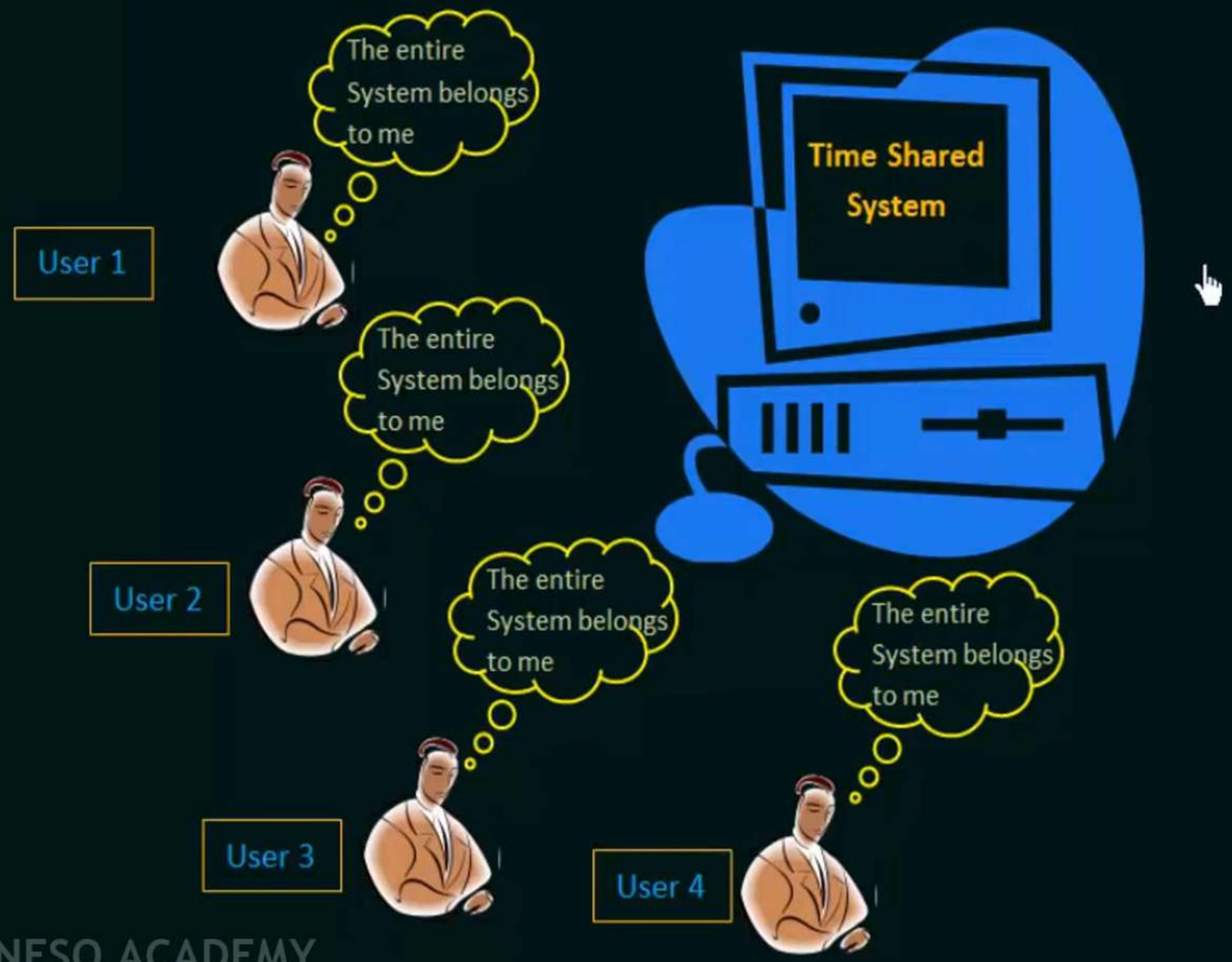
NESO ACADEMY

(ii) Time Sharing (Multitasking)

- CPU executes multiple jobs by switching among them
- Switches occur so frequently that the users can interact with each program while it is running
- Time sharing requires an interactive (or hands-on) computer system, which provides direct communication between the user and the system.
- A time-shared operating system allows many users to share the computer simultaneously.

NESO ACADEMY

- A time-shared operating system allows many users to share the computer simultaneously.



- Uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer.
- Each user has at least one separate program in memory
- A program loaded into memory and executing is called a "PROCESS"

NESO ACADEMY

Operating System Services

- An OS provides an environment for the execution of programs
- It provides certain services to programs and to users of those programs

NESO ACADEMY

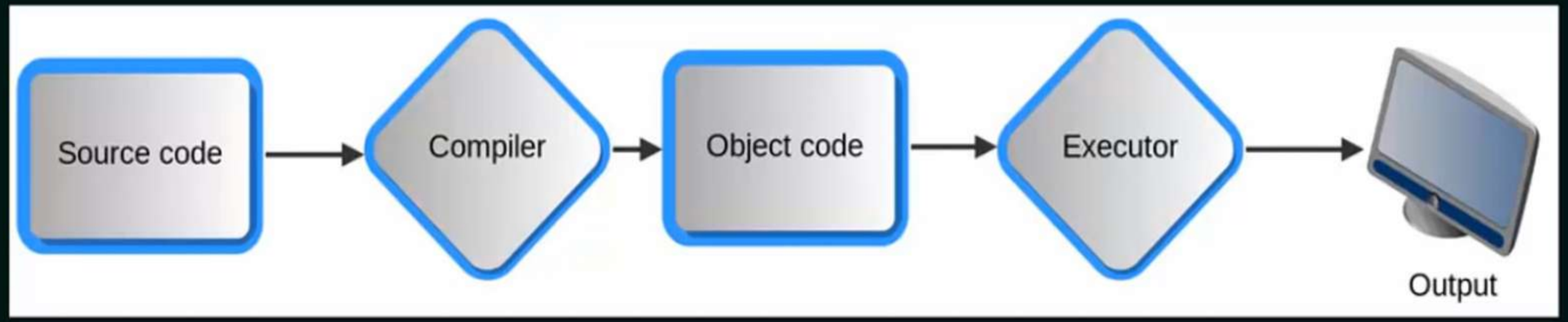
1) User Interface

Command Line Interface (CLI)

Graphical User Interface (GUI)

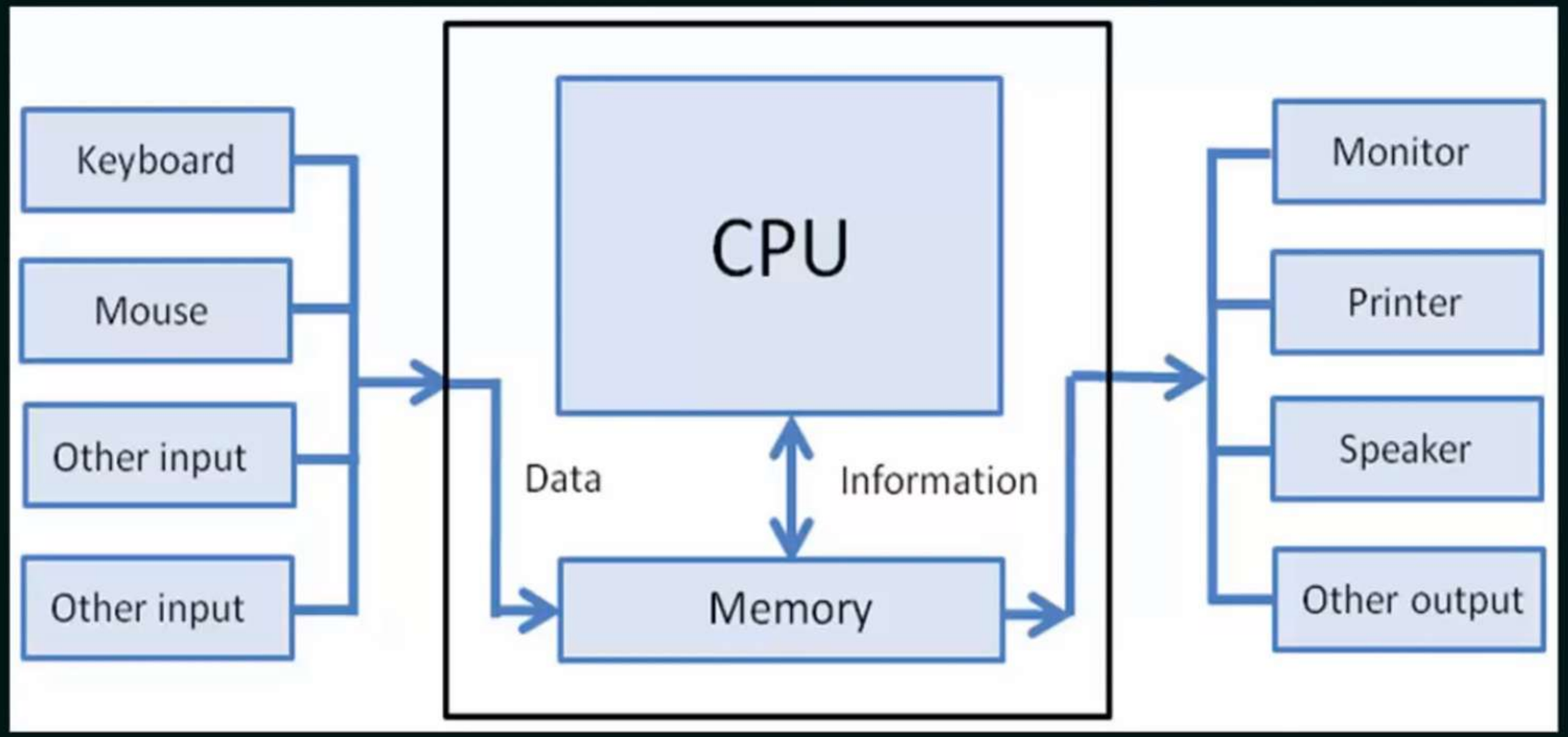
NESO ACADEMY

2) Program Execution



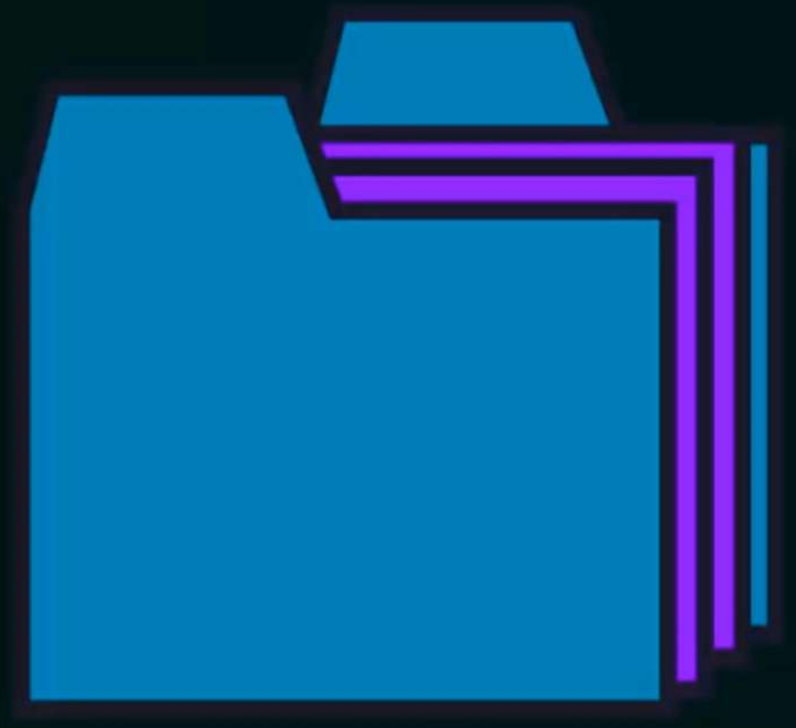
NESO ACADEMY

3) I/O Operations



NESO ACADEMY

4) File System Manipulation

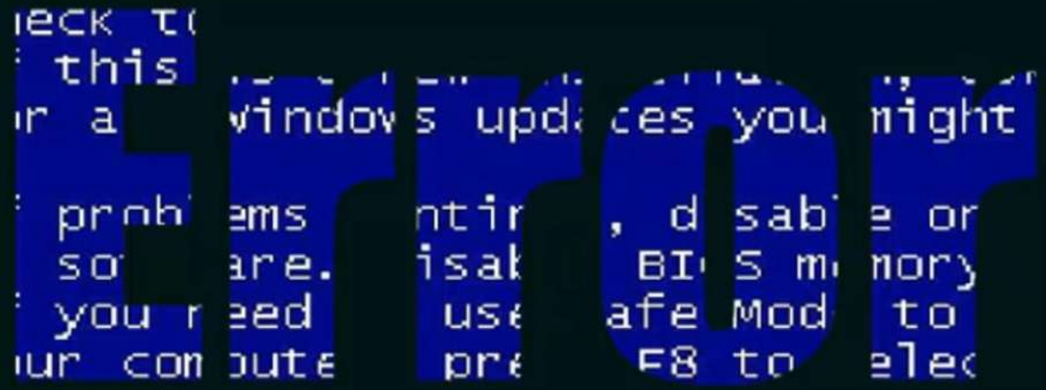


NESO ACADEMY

5) Communications

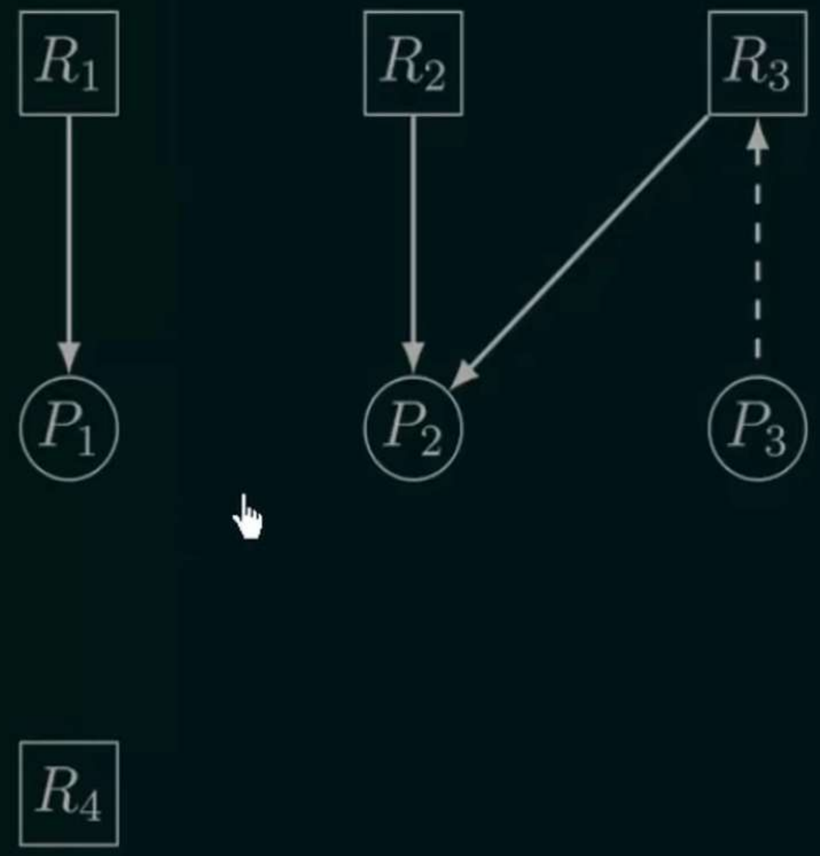
NESO ACADEMY

6) Error detection



NESO ACADEMY

7) Resource Allocation



NESO ACADEMY

8) Accounting

NESO ACADEMY

9) Protection and Security

NESO ACADEMY

1. User Interface
2. Program Execution
3. I/O Operations
4. File System manipulation
5. Communications
6. Error Detection
7. Resource Allocation
8. Accounting
9. Protection and Security

Operating System Services

NESO ACADEMY

User Operating System Interface

There are two fundamental approaches for users to interface with the operating system:

1) Provide a Command-Line Interface (CLI) or Command Interpreter that allows users to directly enter commands that are to be performed by the operating system.

2) Allows the user to interface with the operating system via a Graphical User Interface or GUI.

NESO ACADEMY

Command Interpreter

- Some operating systems include the command interpreter in the kernel.
- Others, such as Windows XP and UNIX, treat the command interpreter as a special program.
- On systems with multiple command interpreters to choose from, the interpreters are known as shells.
- E.g.
  - Bourne shell,
  - C shell
  - Bourne-Again shell (BASH)
  - Korn shell, etc.

NESO ACADEMY

neso@neso-academy ~/Desktop

File Edit View Search Terminal Help

```
neso@neso-academy ~ $ pwd
/home/neso
neso@neso-academy ~ $ cd Desktop/
neso@neso-academy ~/Desktop $ pwd
/home/neso/Desktop
neso@neso-academy ~/Desktop $ mkdir Myfile1
neso@neso-academy ~/Desktop $ ls
Myfile1
neso@neso-academy ~/Desktop $ mkdir Myfile2
neso@neso-academy ~/Desktop $ ls
Myfile1  Myfile2
neso@neso-academy ~/Desktop $ rm -rf Myfile1
neso@neso-academy ~/Desktop $ ls
Myfile2
neso@neso-academy ~/Desktop $ rm -rf Myfile2
neso@neso-academy ~/Desktop $ ls
neso@neso-academy ~/Desktop $
```

Menu neso@neso-academ... 17:39

NESO ACADEMY

Recycle Bin

Computer

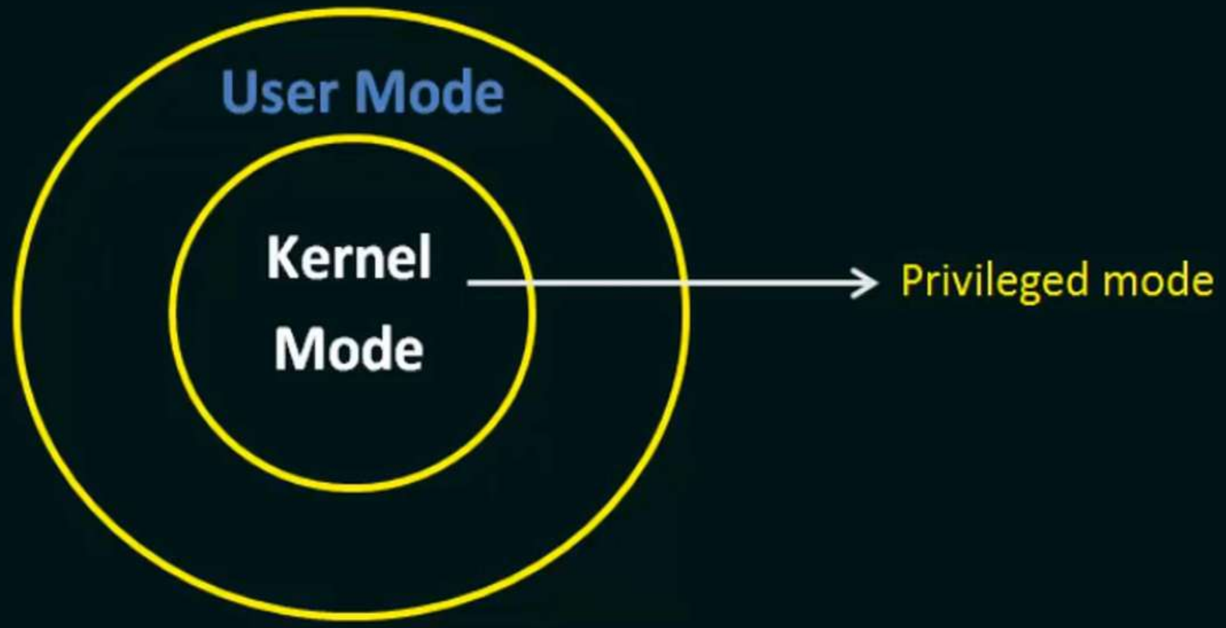
New folder

C:\Windows\system32\cmd.exe

```
C:\Users\Neso>cd Desktop

C:\Users\Neso\Desktop>mkdir Myfile1

C:\Users\Neso\Desktop>mkdir Myfile2

C:\Users\Neso\Desktop>dir
 Volume in drive C is Windows7_OS
 Volume Serial Number is A444-53AD

 Directory of C:\Users\Neso\Desktop

06-03-2018  12:55    <DIR>          .
06-03-2018  12:55    <DIR>          ..
06-03-2018  12:55    <DIR>          Myfile1
06-03-2018  12:55    <DIR>          Myfile2
06-03-2018  12:40    <DIR>          New folder
               0 File(s)              0 bytes
               5 Dir(s)  31,948,390,400 bytes free

C:\Users\Neso\Desktop>rmdir Myfile1

C:\Users\Neso\Desktop>rmdir Myfile2

C:\Users\Neso\Desktop>
```

NESO ACADEMY

System calls provide an interface to the services made available by an Operating System.



- System call is the programmatic way in which a computer program requests a service from the kernel of the operating system.
- These calls are generally available as routines written in C and C++.

NESO ACADEMY

Example of a System Call sequence for writing a simple program to read data from one file and copy them to another file:

Acquire Input Filename

Write prompt to screen

Accept Input

Acquire Output Filename

Write prompt to screen

Accept Input

NESO ACADEMY

NESO ACADEMY

Types of System Calls

System calls can be grouped roughly into five major categories:

1. Process Control
2. File Manipulation
3. Device Management
4. Information Maintenance
5. Communications

NESO ACADEMY

1. Process Control

- end, abort
- load, execute
- create process, terminate process
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory

NESO ACADEMY

2. File Manipulation

- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

NESO ACADEMY

3. Device Manipulation

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

NESO ACADEMY

4. Information Maintenance

- get time or date, set time or date
- get system data, set system data
- get process, file, or device attributes
- set process, file, or device attributes

NESO ACADEMY

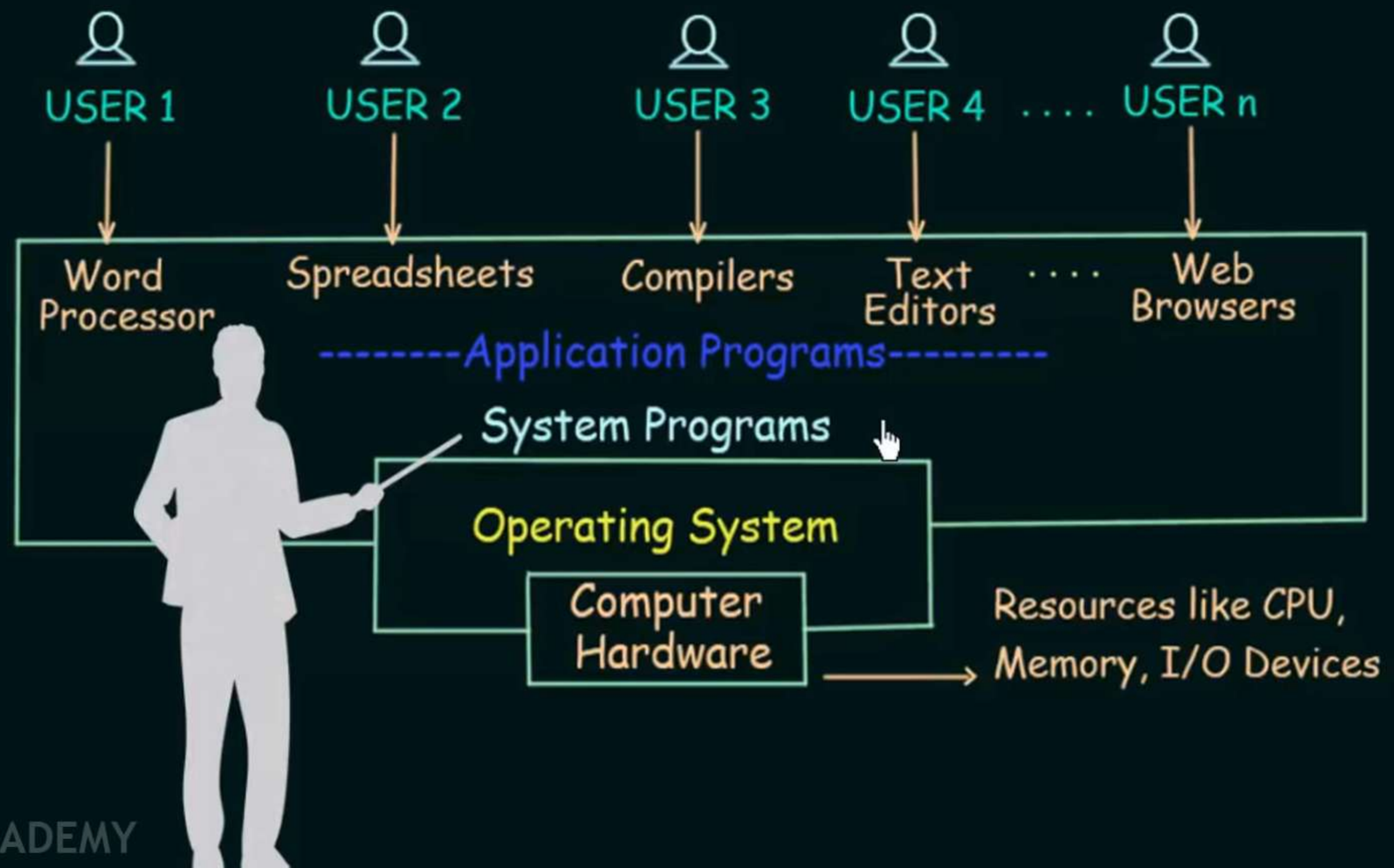
5. Communications

- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

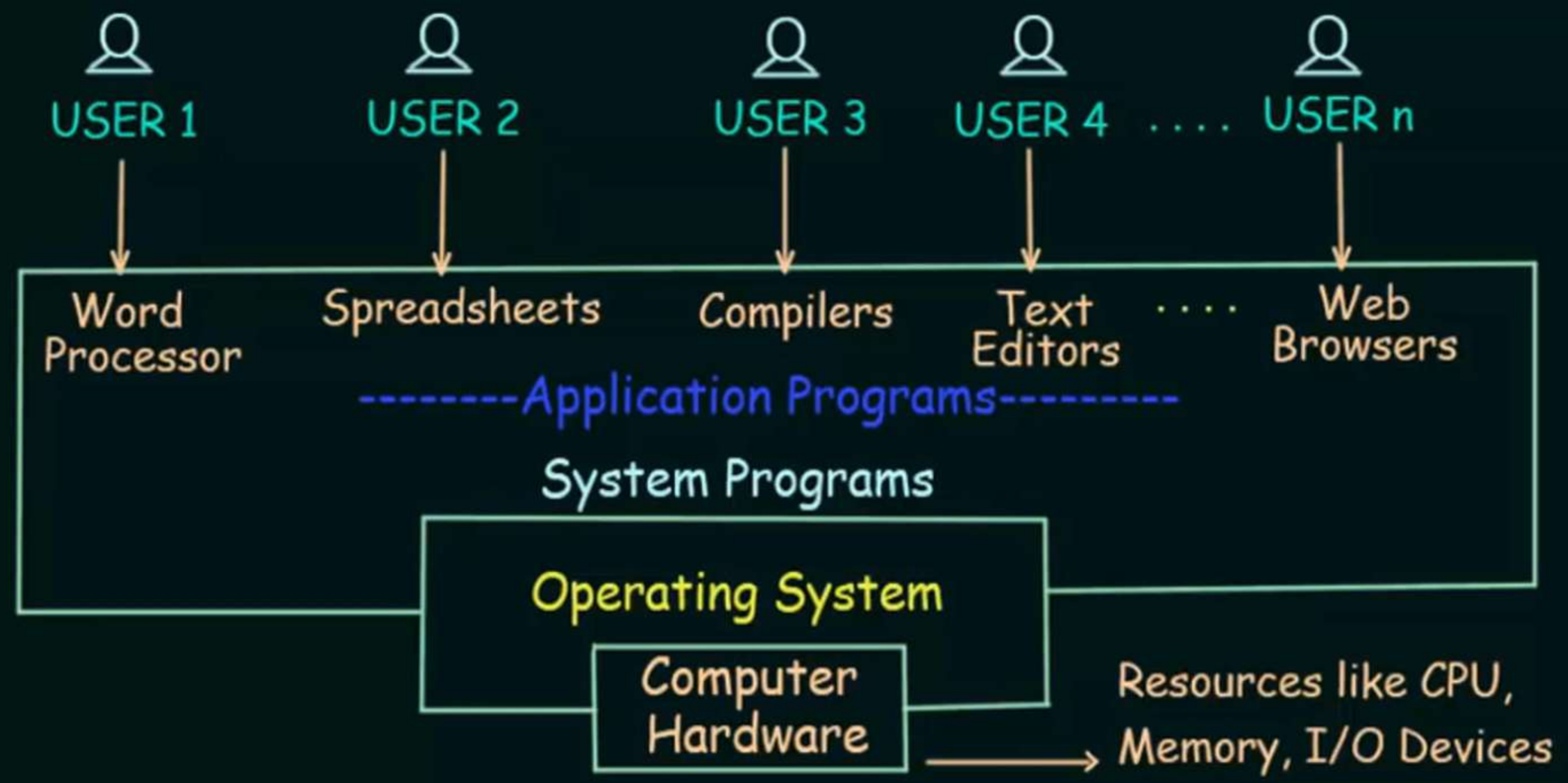
NESO ACADEMY

System Programs

An important aspect of a modern system is the collection of system programs.



NESO ACADEMY



- System programs provide a convenient environment for program development and execution.
- Some of them are simply user interfaces to system calls.
- Others are considerably more complex.

NESO ACADEMY

System Programs can be divided into the following categories:

File Management

- Create
- Delete
- Copy
- Rename
- Print
- Dump
- List, and generally manipulate files and directories.

NESO ACADEMY

Status Information

Ask the system for:

- Date, Time
- Amount of available memory or disk space
- Number of users
- Detailed performance
- Logging, and debugging information etc.

NESO ACADEMY

File modification

- Several text editors may be available to create and modify the content of files stored on disk or other storage devices.
- There may also be special commands to search contents of files or perform transformations of the text.

NESO ACADEMY

Programming-language support

- Compilers
- Assemblers
- Debuggers and
- Interpreters

for common programming languages

(such as C, C++, Java,Visual Basic, and PERL)

are often provided to the user with the operating system.

NESO ACADEMY

Program loading and execution

Once a program is assembled or compiled, it must be loaded into memory to be executed.

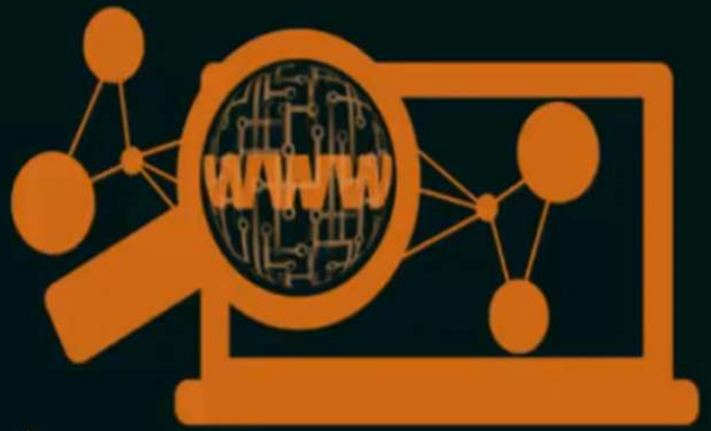
The system may provide:

- Absolute loaders
- Relocatable loaders
- Linkage editors and
- Overlay loaders

Debugging systems for either higher-level languages or machine language are needed as well.

NESO ACADEMY

Communications



These programs provide the mechanism for:

- Creating virtual connections among processes, users, and computer systems.
- Allowing users to send messages to one another's screens
- To browse webpages
- To send electronic-mail messages
- To log in remotely or to transfer files from one machine to another.

NESO ACADEMY

In addition to systems programs, most operating systems are supplied with programs that are useful in solving common problems or performing common operations.

NESO ACADEMY

Examples:

Web Browsers

Word Processors

Spreadsheets

Database Systems

Games

etc.

Application Programs

NESO ACADEMY

Categories of System Programs:

- File Management
- Status Information
- File Modification
- Programming-language support
- Program Loading and Execution
- Communications

NESO ACADEMY

Operating System Design & Implementation

Design Goals:

1st Problem: - Defining Goals and specification

- Choice of Hardware
- Type of System

Beyond this highest design level, the requirements may be much harder to specify.

Requirements:

- User Goals
- System Goals

NESO ACADEMY

Requirements:

- User Goals
- System Goals

User Requirements:
(User Goals)

The system should be:
Convenient to use,
Easy to learn & use,
Reliable, safe & fast

The system should be:
Easy to design, implement,
maintain, operate.
It should be flexible, reliable,
error free & efficient

Designer,
Engineer Requirements:
(System goals)

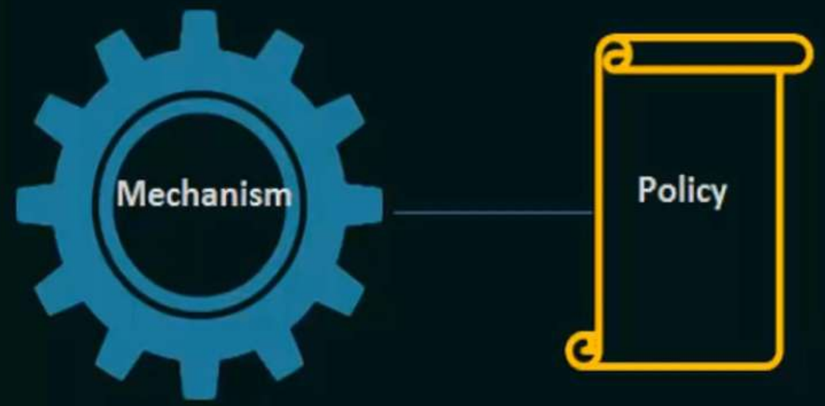
NESO ACADEMY

Mechanisms and Policies:

Mechanisms determine how to do something.

Policies determine what will be done.

One important principle is the separation of policy from mechanism.



Good and Flexible

Not Good

NESO ACADEMY

Implementation:

- Once an operating system is designed, it must be implemented.
- Traditionally, operating systems have been written in assembly language.
- Now, however, they are most commonly written in higher-level languages such as C or C++

NESO ACADEMY

Advantages of writing in high level languages:

- The code can be written faster
- It is more compact
- It is easier to understand and debug
- It is easier to port

E.g.

MS-DOS was written in Intel 8088 assembly language. Consequently, it is available on only the Intel family of CPUs.

The Linux operating system, in contrast, is written mostly in C and is available on a number of different CPUs, including Intel 80X86, Motorola 680X0, SPARC, and MIPS RXOOO.

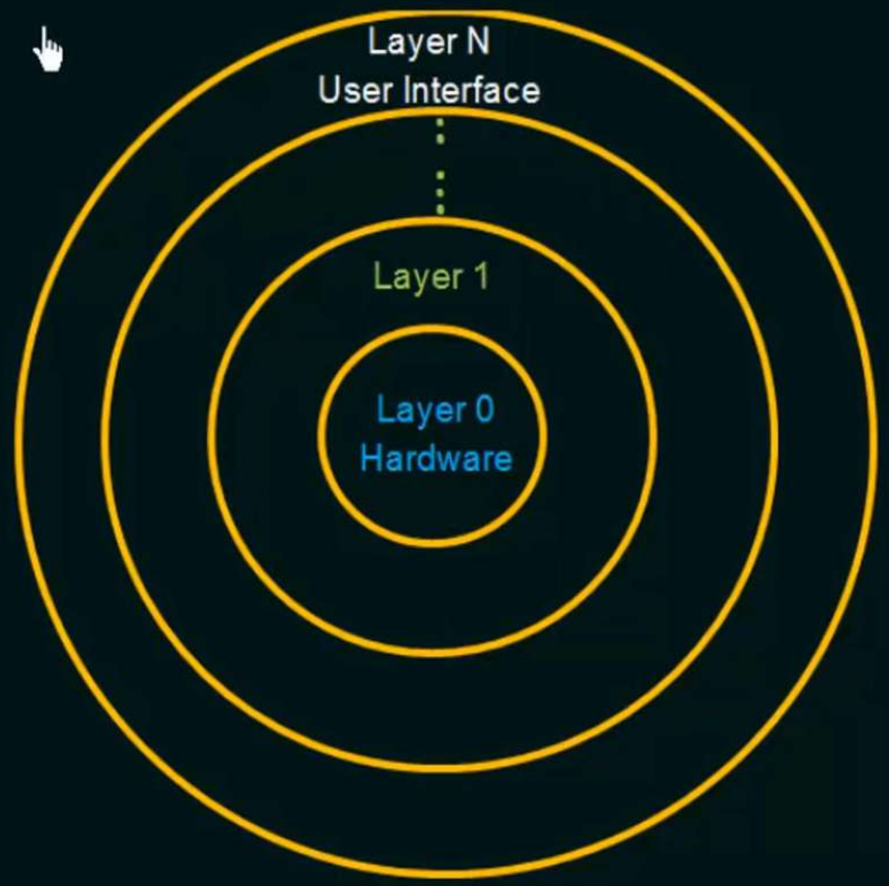
NESO ACADEMY

Monolithic Structure

	(The Users)		
	Shells and Commands Compilers and Interpreters System Libraries		
Kernel	System-Call interface to the Kernel		
	Signal, terminal handling character I/O system terminal drivers	file system swapping block I/O system disk and tape drivers	CPU scheduling page replacement demand paging virtual memory
	kernel interface to the hardware		
	terminal controllers terminals	device controllers disk and tapes	memory controllers physical memory

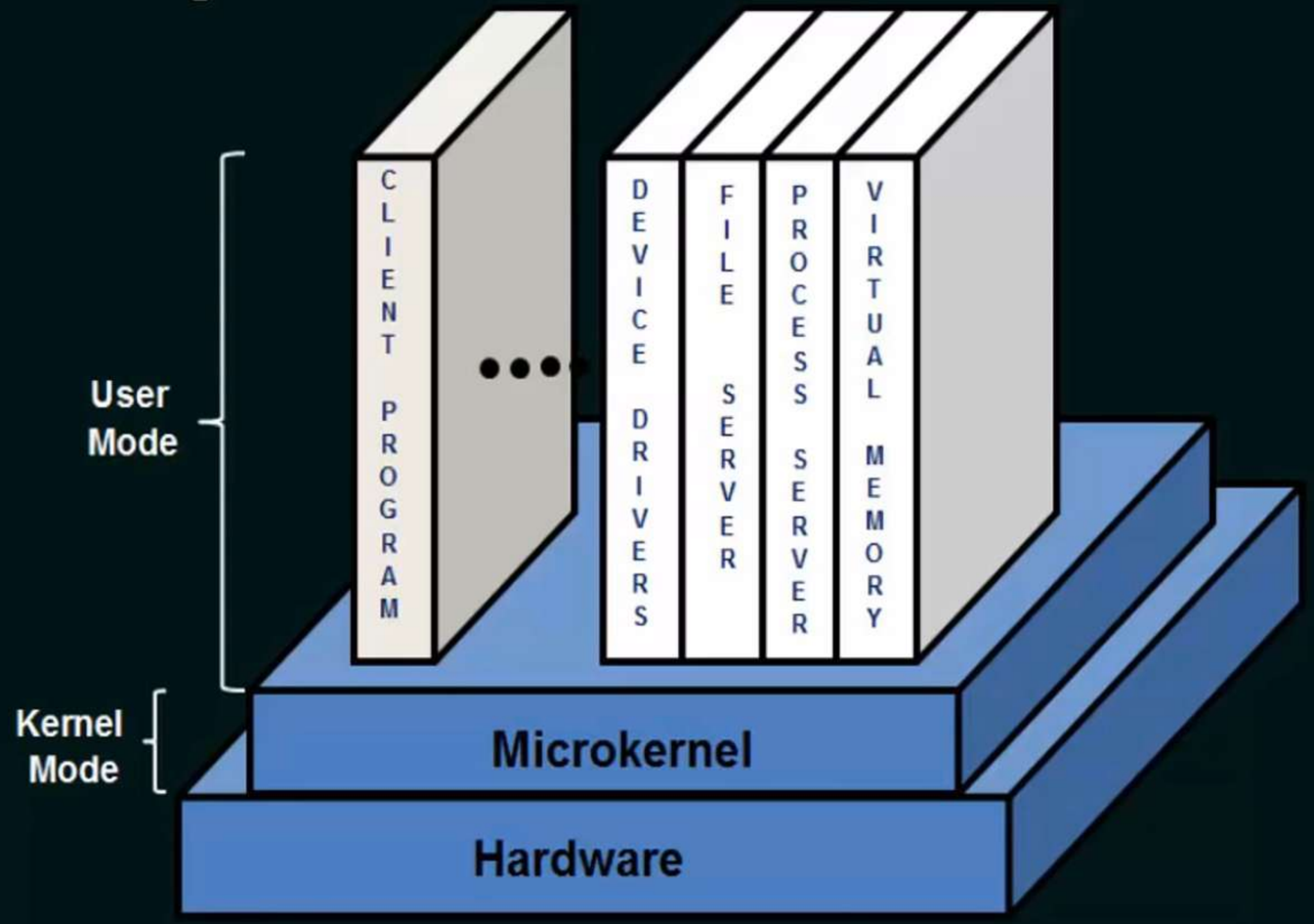
NESO ACADEMY

Layered Structure



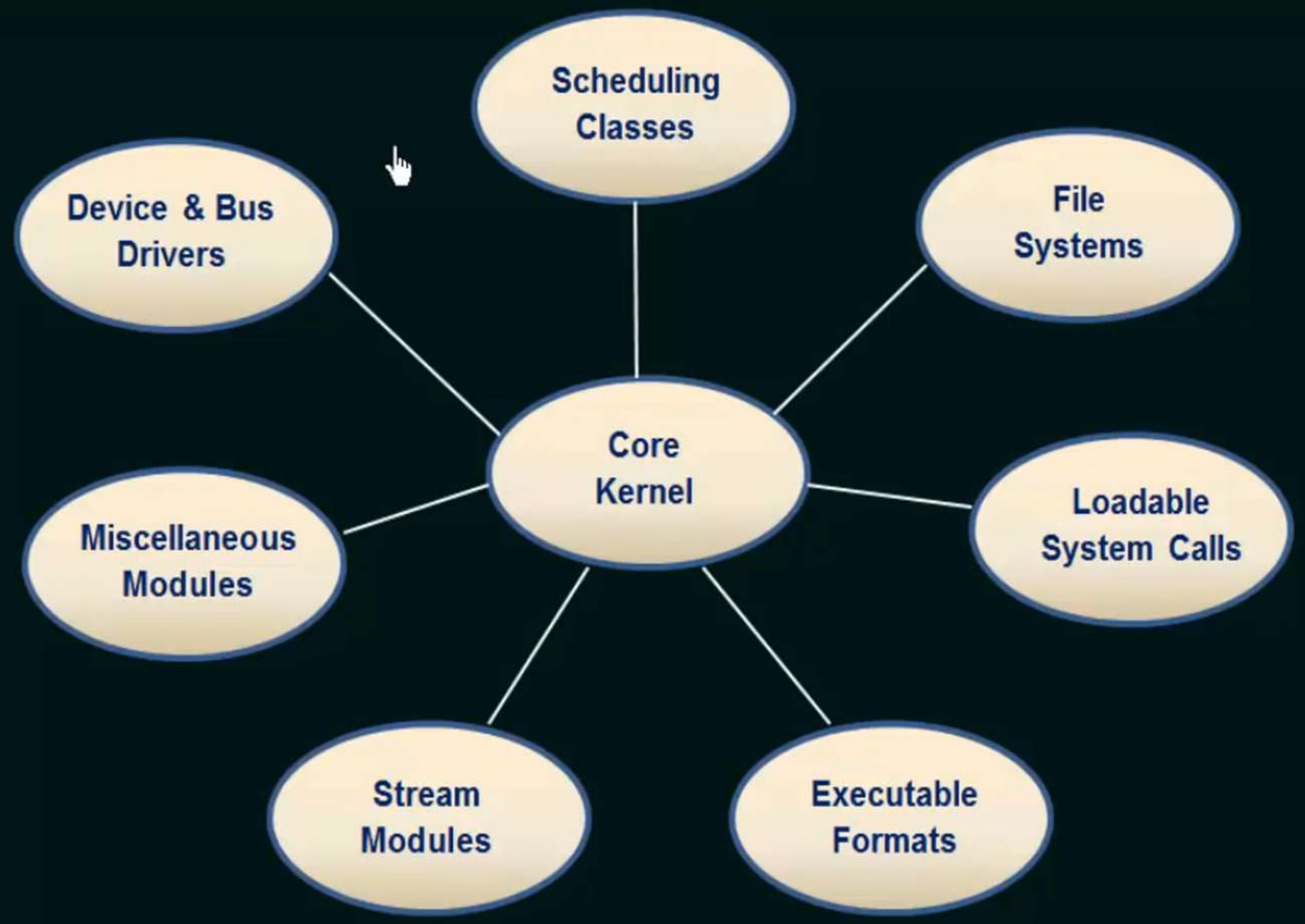
NESO ACADEMY

Microkernels



NESO ACADEMY

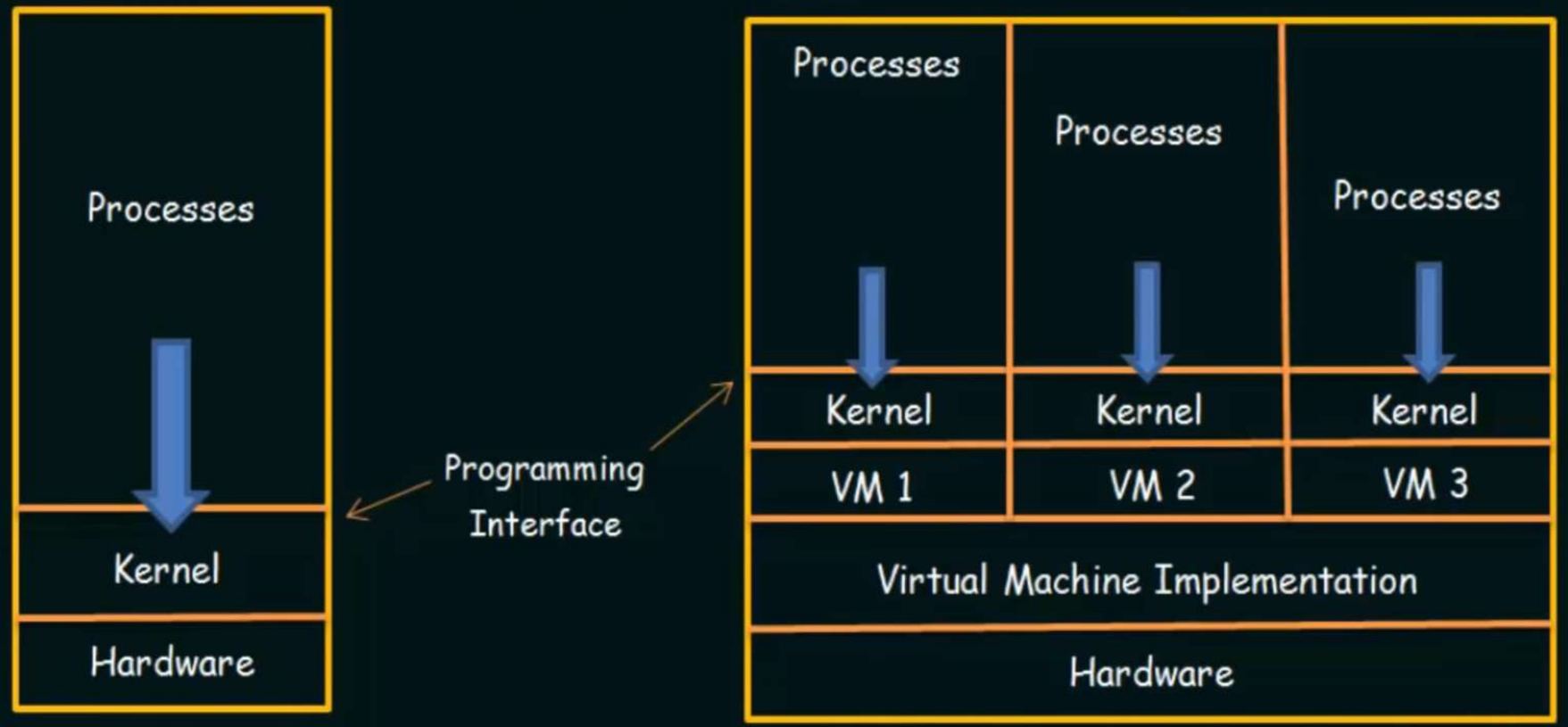
Modules



NESO ACADEMY

Virtual Machines

The fundamental idea behind a virtual machine is to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate execution environment is running its own private computer.



(a) (b)

System Models:

(a) Nonvirtual Machine

(b) Virtual Machine

NESO ACADEMY

IMPLEMENTATION

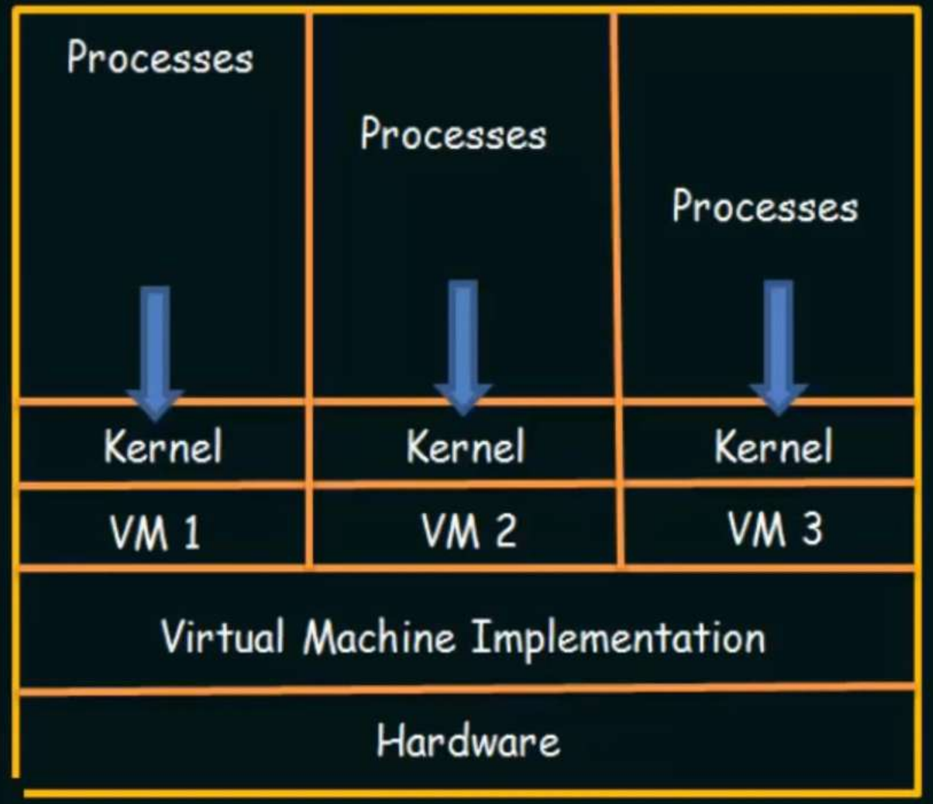
Virtual Machine Software – Runs in Kernel mode
Virtual Machine itself – Runs in User Mode

Just as the physical machine has two modes, however, so must the virtual machine.

Consequently, we must have:

- A virtual user mode and
- A virtual kernel mode

BOTH OF WHICH RUN IN A PHYSICAL USER MODE



NESO ACADEMY

Operating System Generation & System Boot

- Design, code, and implement an operating system specifically for one machine at one site

- Operating systems are designed to run on any of a class of machines at a variety of sites with a variety of peripheral configurations

- The system must then be configured or generated for each specific computer site, a process sometimes known as system generation (SYSGEN) is used for this

The following kinds of information must be determined by the SYSGEN Program:

- What CPU is to be used?
- How much memory is available?
- What devices are available?
- What operating-system options are desired?

NESO ACADEMY

System Boot

- The procedure of starting a computer by loading the kernel is known as booting the system.
- On most computer systems, a small piece of code known as the bootstrap program or bootstrap loader locates the kernel
- This program is in the form of read-only memory (ROM), because the RAM is in an unknown state at system startup. ROM is convenient because it needs no initialization and cannot be infected by a computer virus.

EPROM
Erasable Programable Read Only Memory

NESO ACADEMY

System Boot

- The procedure of starting a computer by loading the kernel is known as booting the system.
- On most computer systems, a small piece of code known as the bootstrap program or bootstrap loader locates the kernel
- This program is in the form of read-only memory (ROM), because the RAM is in an unknown state at system startup. ROM is convenient because it needs no initialization and cannot be infected by a computer virus.

When the full bootstrap program has been loaded, it can traverse the file system to find the operating system kernel, load it into memory, and start its execution.
It is only at this point that the system is said to be

RUNNING

NESO ACADEMY

Process Management

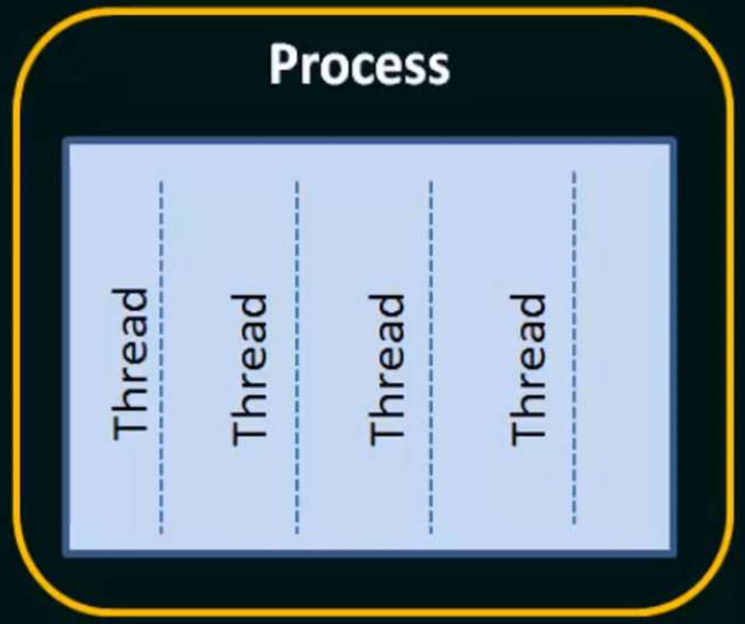
(Processes and Threads)

Process:

A process can be thought of as a program in execution.

Thread:

A thread is the unit of execution within a process. A process can have anywhere from just one thread to many threads.



NESO ACADEMY

Process State

- As a process executes, it changes state.
- The state of a process is defined in part by the current activity of that process.

Each process may be in one of the following states:

NEW	The process is being created.
RUNNING	Instructions are being executed.
WAITING	The process is waiting for some event to occur (Such as an I/O completion or reception of a signal).
READY	The process is waiting to be assigned to a processor.
TERMINATED	The process has finished execution.

NESO ACADEMY

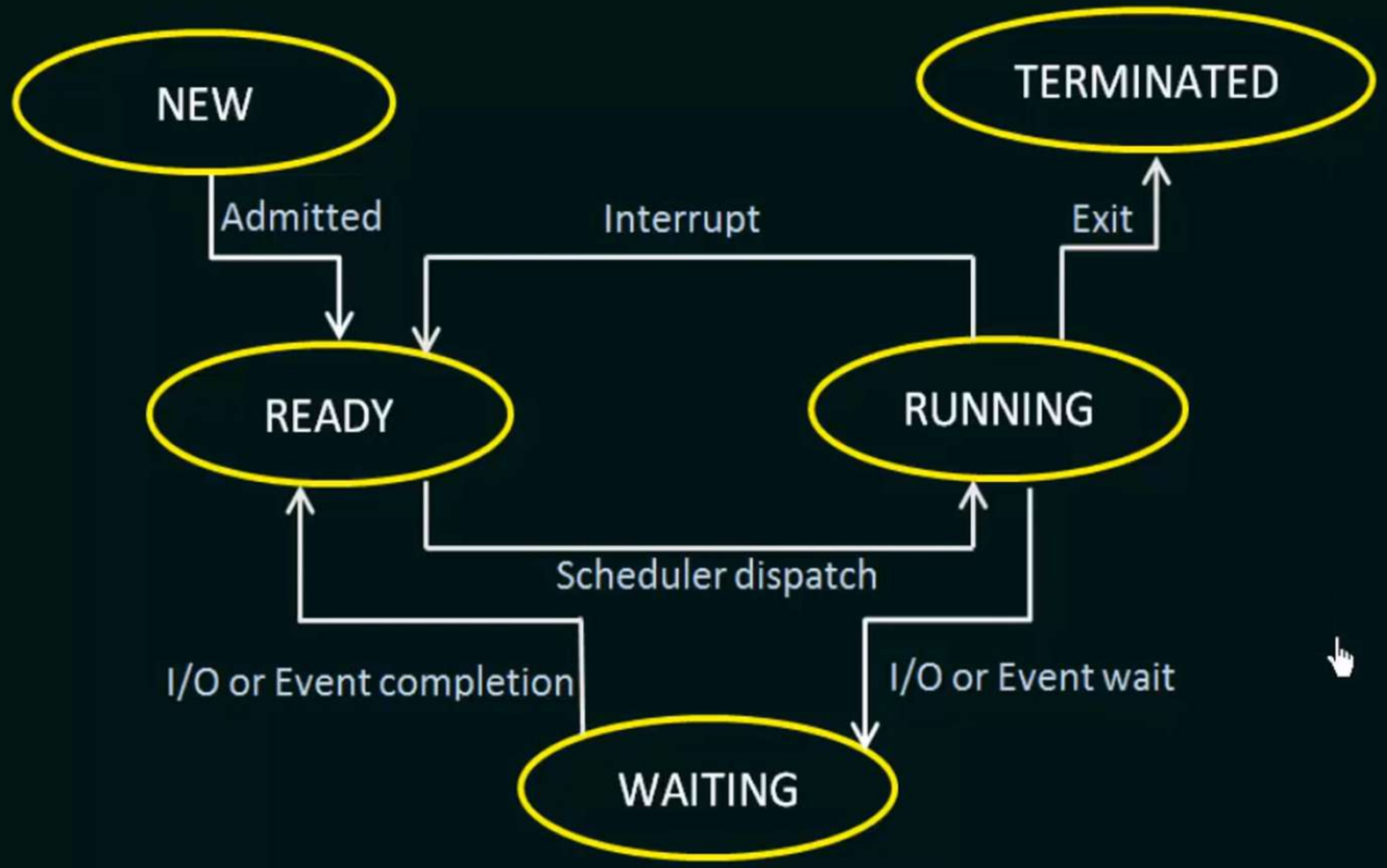


Fig: Diagram of process state

NESO ACADEMY

Process Control Block

Each process is represented in the operating system by a Process Control Block (PCB) — also called a task control block.

NESO ACADEMY

Process State

Program Counter

CPU Registers

Process ID

Process state
Process number
Program counter
Registers
Memory limits
List of open files
.....

CPU Scheduling information

Memory management information

Accounting information

I/O Status Information

NESO ACADEMY

Process Scheduling

- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.

- The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.

- To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU.
    - For a single-processor system, there will never be more than one running process.
    - If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.

NESO ACADEMY

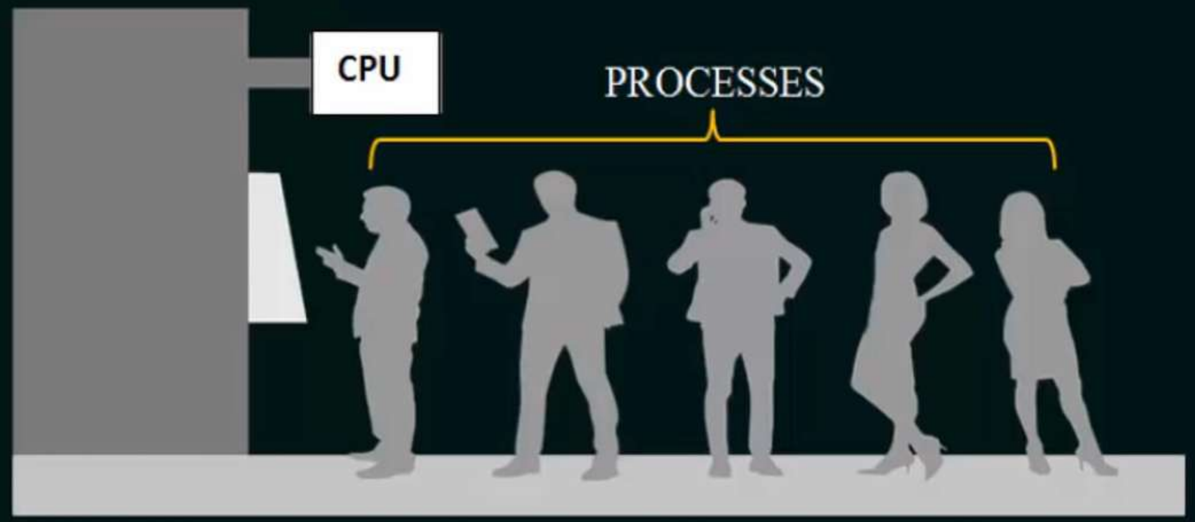
Scheduling Queues

JOB QUEUE

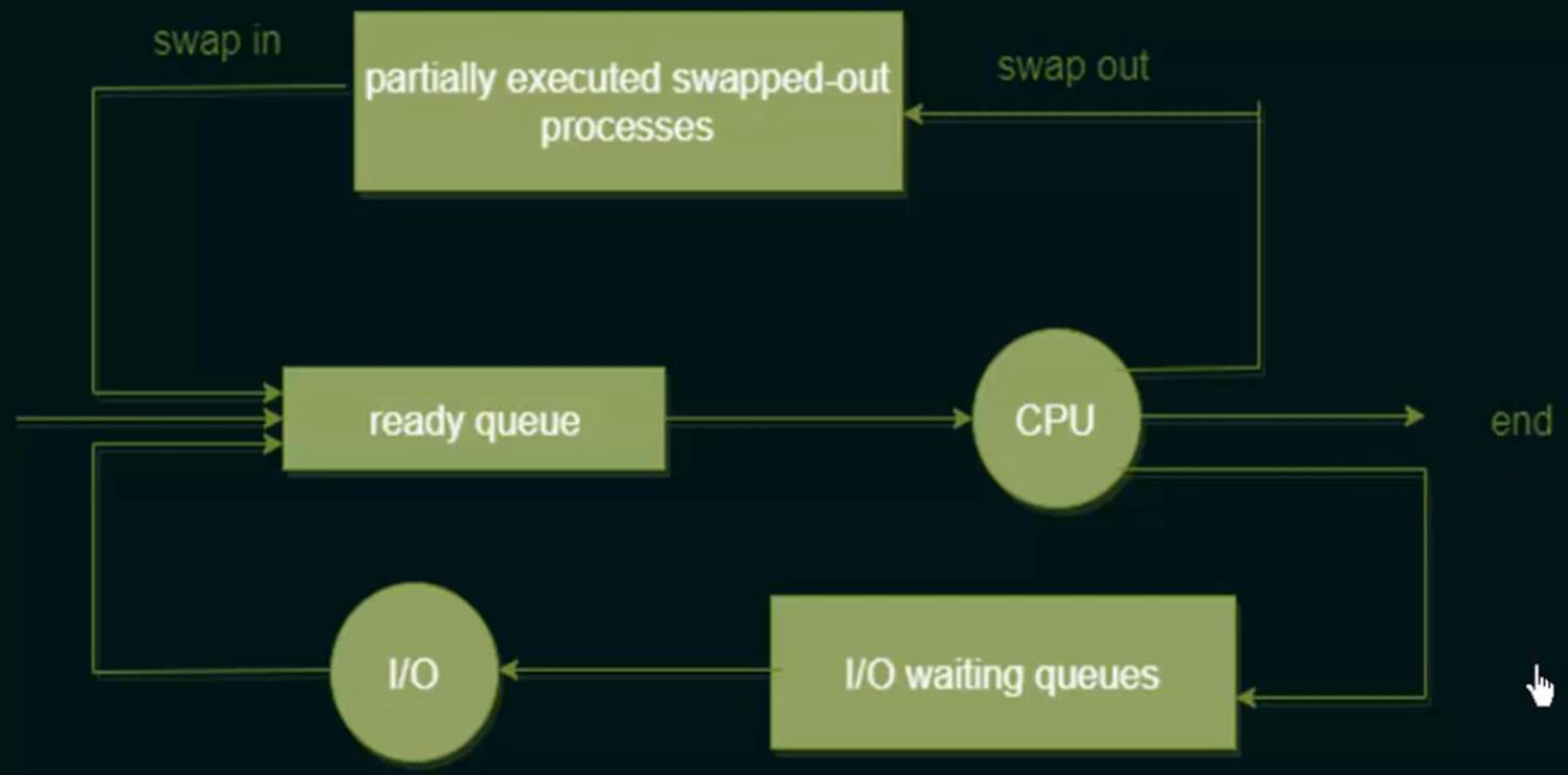
As processes enter the system, they are put into a **job queue,** which consists of all processes in the system.

READY QUEUE

The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue.**



NESO ACADEMY



NESO ACADEMY

Context Switch

- Interrupts cause the operating system to change a CPU from its current task and to run a kernel routine.

- Such operations happen frequently on general-purpose systems.

When an interrupt occurs, the system needs to save the current **context** of the process currently running on the CPU so that it can restore that context when its processing is done, essentially suspending the process and then resuming it.

- The context is represented in the PCB of the process

Process Control Block

NESO ACADEMY

Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process.

State Save

State Restore

This task is known as a **context switch**.

- Context-switch time is pure overhead, because the system does no useful work while switching.
- Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions (such as a single instruction to load or store all registers).
- Typical speeds are a few milliseconds.

NESO ACADEMY

Operations on Processes

(Process Creation)

- A process may create several new processes, via a create-process system call, during the course of execution.
- The creating process is called a parent process, and the new processes are called the children of that process.
- Each of these new processes may in turn create other processes, forming a tree of processes.

NESO ACADEMY

When a process creates a new process, two possibilities exist in terms of execution:

1. The parent continues to execute concurrently with its children.

2. The parent waits until some or all of its children have terminated.

There are also two possibilities in terms of the address space of the new process:

1. The child process is a duplicate of the parent process (it has the same program and data as the parent).

2. The child process has a new program loaded into it.

NESO ACADEMY

Operations on Processes

(Process Termination)

- A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit () system call.
- At that point, the process may return a status value (typically an integer) to its parent process (via the wait() system call).
- All the resources of the process—including physical and virtual memory, open files, and I/O buffers—are deallocated by the operating system.

NESO ACADEMY

Termination can occur in other circumstances as well:

- A process can cause the termination of another process via an appropriate system call.
- Usually, such a system call can be invoked only by the parent of the process that is to be terminated.
- Otherwise, users could arbitrarily kill each other's jobs.

NESO ACADEMY

A parent may terminate the execution of one of its children for a variety of reasons, such as these:

- The child has exceeded its usage of some of the resources that it has been allocated. *(To determine whether this has occurred, the parent must have a mechanism to inspect the state of its children.)*
- The task assigned to the child is no longer required.
- The parent is exiting, and the operating system does not allow a child to continue if its parent terminates.

NESO ACADEMY

Interprocess Communication

Processes executing concurrently in the operating system may be either independent processes or cooperating processes.

Independent processes -	They cannot affect or be affected by the other processes executing in the system.
Cooperating processes -	They can affect or be affected by the other processes executing in the system.

Any process that shares data with other processes is a cooperating process.

NESO ACADEMY

There are several reasons for providing an environment that allows process cooperation:

Information sharing

Computation speedup

Modularity

Convenience

NESO ACADEMY

Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data and information.

There are two fundamental models of interprocess communication:

(1) Shared memory

(2) Message passing

- In the shared-memory model, a region of memory that is shared by cooperating processes is established.
  Processes can then exchange information by reading and writing data to the shared region.

- In the message passing model, communication takes place by means of messages exchanged between the cooperating processes.

NESO ACADEMY

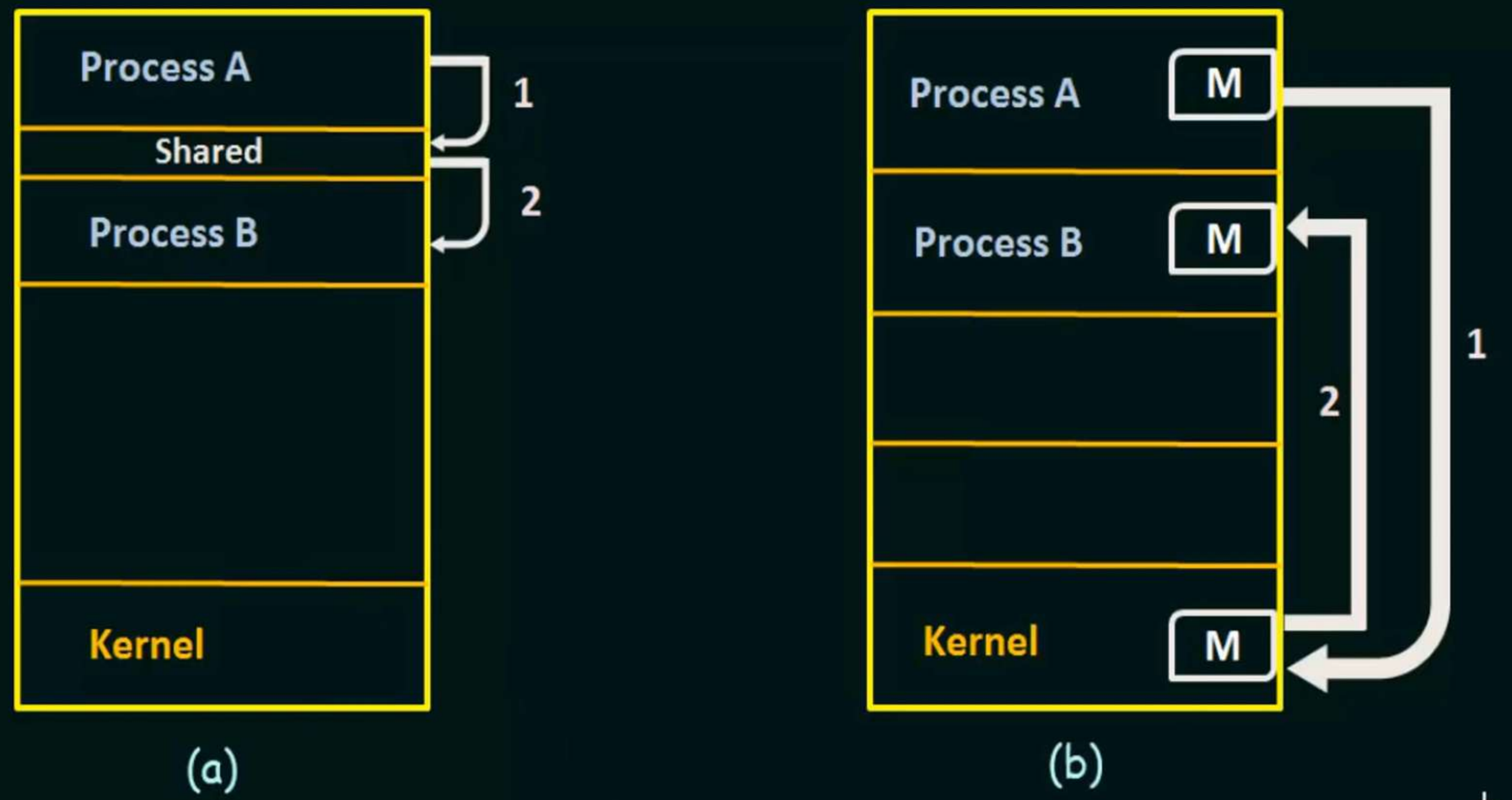
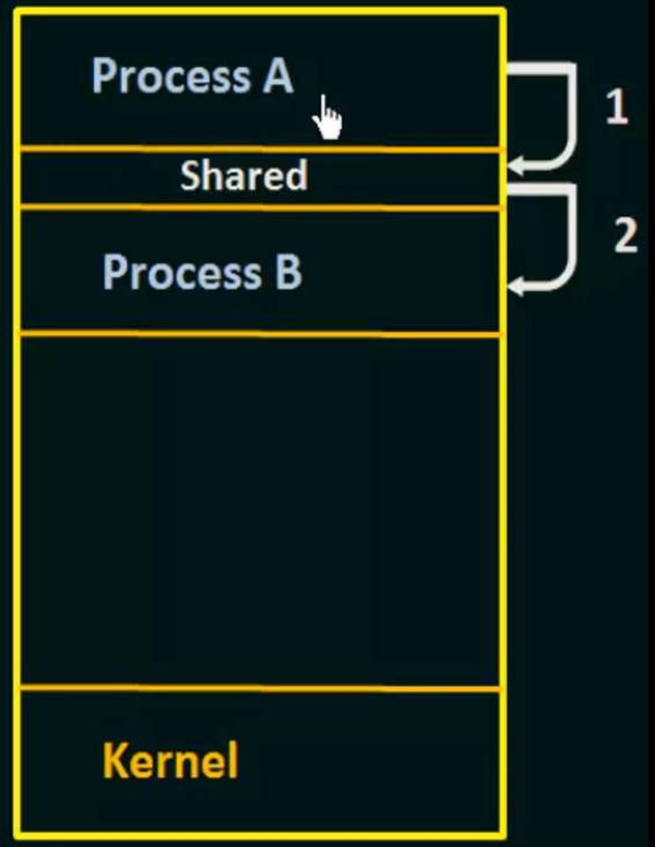


Fig: Communications models, (a) Shared memory, (b) Message Passing.

NESO ACADEMY

Shared Memory Systems

- Interprocess communication using shared memory requires communicating processes to establish a region of shared memory.
- Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment.
- Other processes that wish to communicate using this shared-memory segment must attach it to their address space.
- Normally, the operating system tries to prevent one process from accessing another process's memory.
- Shared memory requires that two or more processes agree to remove this restriction.



NESO ACADEMY

Producer Consumer Problem

A producer process produces information that is consumed by a consumer process.

For example, a compiler may produce assembly code, which is consumed by an assembler. The assembler, in turn, may produce object modules, which are consumed by the loader.

- One solution to the producer-consumer problem uses shared memory.
- To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer.

NESO ACADEMY

- One solution to the producer-consumer problem uses shared memory.
- To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer.
- This buffer will reside in a region of memory that is shared by the producer and consumer processes.
- A producer can produce one item while the consumer is consuming another item.
- The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

NESO ACADEMY

Two kinds of buffers:

Unbounded buffer

Places no practical limit on the size of the buffer. The consumer may have to wait for new items, but the producer can always produce new items.

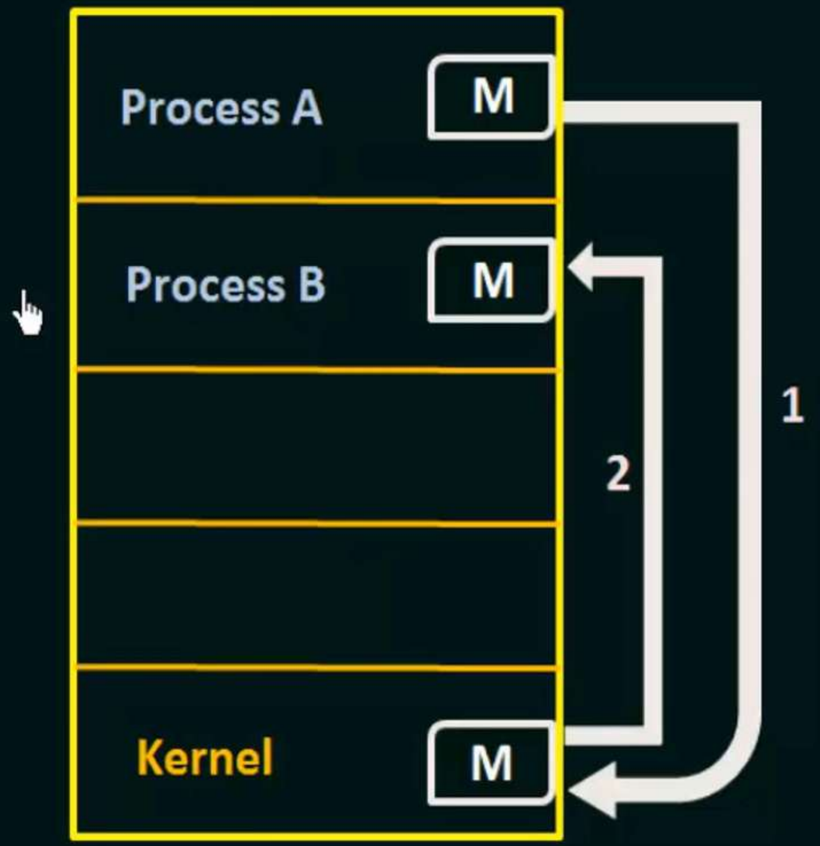
Bounded buffer

Assumes a fixed buffer size. In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.

NESO ACADEMY

Message-Passing Systems (Part-1)

Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space and is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network.



NESO ACADEMY

A message-passing facility provides at least two operations:

- send (message)

and

- receive (message)

Messages sent by a process can be of either fixed or variable size.

FIXED SIZE: The system-level implementation is straightforward.
But makes the task of programming more difficult.

VARIABLE SIZE: Requires a more complex system-level implementation.
But the programming task becomes simpler.

NESO ACADEMY

If processes P and Q want to communicate, they must send messages to and receive messages from each other.

A communication link must exist between them.

This link can be implemented in a variety of ways. There are several methods for logically implementing a link and the **send( ) /receive( )** operations, like:

- Direct or indirect communication
- Synchronous or asynchronous communication
- Automatic or explicit buffering

There are several issues related with features like:

- **Naming**
- **Synchronization**
- **Buffering**

NESO ACADEMY

Naming

Processes that want to communicate must have a way to refer to each other. They can use either **direct** or **indirect** communication.

Under direct communication- Each process that wants to communicate must explicitly name the recipient or sender of the communication.

- send (P, message) - Send a message to process P.
- receive (Q, message) - Receive a message from process Q.

A communication link in this scheme has the following properties:

- A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.
- A link is associated with exactly two processes.
- Between each pair of processes, there exists exactly one link.

This scheme exhibits **symmetry in addressing**; that is, both the sender process and the receiver process must name the other to communicate

NESO ACADEMY

Another variant of Direct Communication- Here, only the sender names the recipient; the recipient is not required to name the sender.

- send (P, message) - Send a message to process P.
- receive (id, message) - Receive a message from any process; *the variable id is set to the name of the process with which communication has taken place.*

This scheme employs **asymmetry in addressing**.

The disadvantage in both of these schemes (symmetric and asymmetric) is the limited modularity of the resulting process definitions. Changing the identifier of a process may necessitate examining all other process definitions.

NESO ACADEMY

With indirect communication:

The messages are sent to and received from **mailboxes**, or ports.

- A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed.
- Each mailbox has a unique identification.
- Two processes can communicate only if the processes have a shared mailbox

  - send (A, message) — Send a message to mailbox A.
  - receive (A, message) — Receive a message from mailbox A.

NESO ACADEMY

A communication link in this scheme has the following properties:

- A link is established between a pair of processes only if both members of the pair have a shared mailbox.
- A link may be associated with more than two processes.
- Between each pair of communicating processes, there may be a number of different links, with each link corresponding to one mailbox.

NESO ACADEMY

Now suppose that processes P1, P2, and P3 all share mailbox A

Process P1 sends a message to A, while both P2 and P3 execute a receive() from A. **Which process will receive the message sent by P1?**

NESO ACADEMY

Process P1 sends a message to A, while both P2 and P3 execute a receive() from A. **Which process will receive the message sent by P1?**

The answer depends on which of the following methods we choose:

- Allow a link to be associated with two processes at most.
- Allow at most one process at a time to execute a receive () operation.
- Allow the system to select arbitrarily which process will receive the message (that is, either P2 or P3, but not both, will receive the message). The system also may define an algorithm for selecting which process will receive the message (that is, round robin where processes take turns receiving messages). The system may identify the receiver to the sender.

A **mailbox** may be **owned** either by a **process** or by the **operating system.**

NESO ACADEMY

Synchronization

Communication between processes takes place through calls to send() and receive () primitives. There are different design options for implementing each primitive.

Message passing may be either **blocking** or **nonblocking**— also known as **synchronous** and **asynchronous**.

NESO ACADEMY

Message passing may be either **blocking** or **nonblocking**— also known as **synchronous** and **asynchronous**.

Blocking send: The sending process is blocked until the message is received by the receiving process or by the mailbox.

Nonblocking send: The sending process sends the message and resumes operation.

Blocking receive: The receiver blocks until a message is available.

Nonblocking receive: The receiver retrieves either a valid message or a null.

NESO ACADEMY

Buffering

Whether communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue.

Basically, such queues can be implemented in three ways:

NESO ACADEMY

Zero capacity: The queue has a maximum length of zero; thus, the link cannot have any messages waiting in it. In this case, the sender must block until the recipient receives the message.

Bounded capacity: The queue has finite length n; thus, at most n messages can reside in it. If the queue is not full when a new message is sent, the message is placed in the queue and the sender can continue execution without waiting. The links capacity is finite, however. If the link is full, the sender must block until space is available in the queue.

Unbounded capacity: The queues length is potentially infinite; thus, any number of messages can wait in it. The sender never blocks.

NESO ACADEMY

Sockets

Used for communication in Client-Server Systems

- A socket is defined as an endpoint for communication.
- A pair of processes communicating over a network employ a pair of sockets—one for each process.
- A socket is identified by an IP address concatenated with a port number.
- The server waits for incoming client requests by listening to a specified port. Once a request is received, the server accepts a connection from the client socket to complete the connection.

NESO ACADEMY

- A socket is identified by an IP address concatenated with a port number.

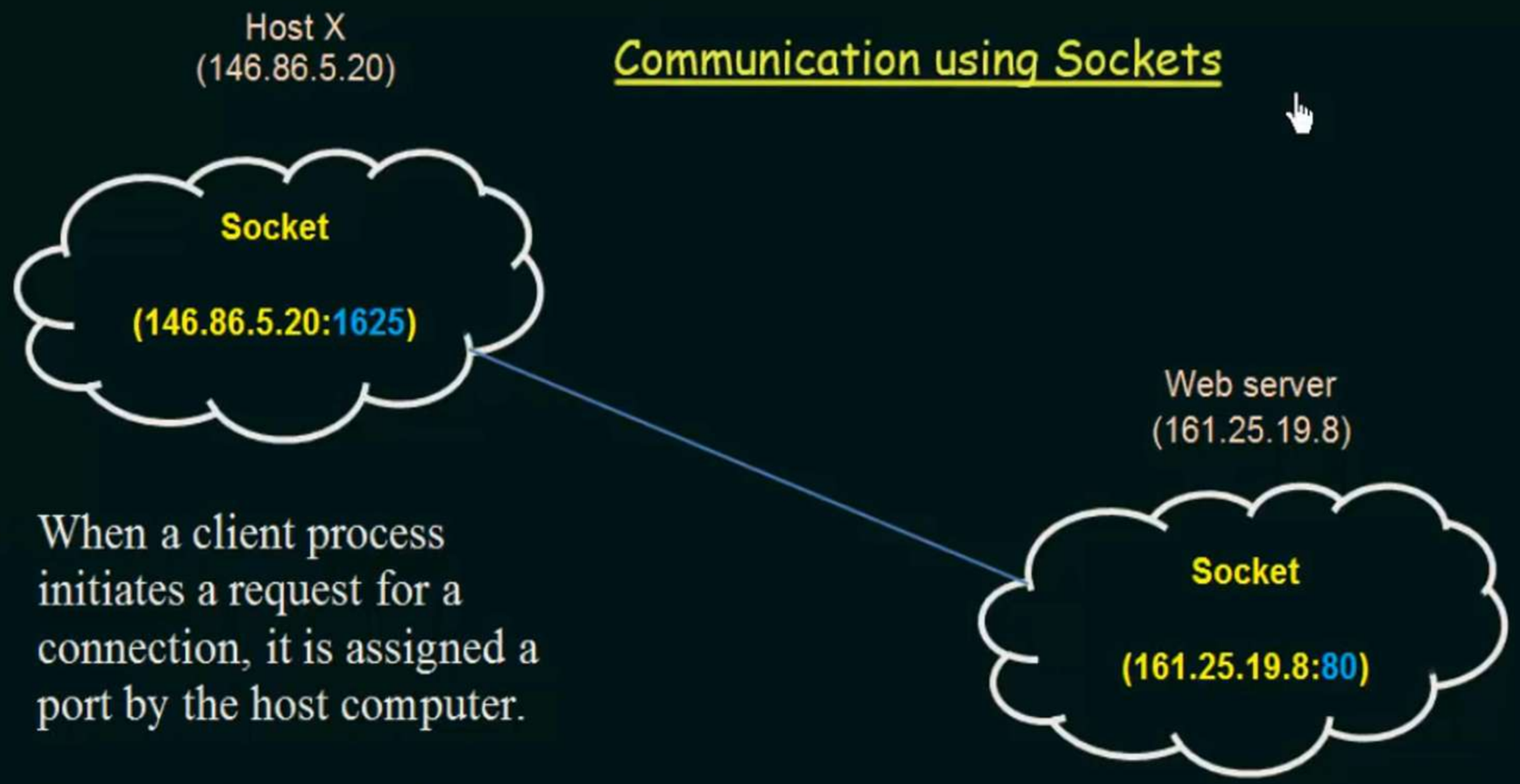
- The server waits for incoming client requests by listening to a specified port. Once a request is received, the server accepts a connection from the client socket to complete the connection.

- Servers implementing specific services (such as telnet, ftp, and http) listen to well-known ports
  (a telnet server listens to port 23, an ftp server listens to port 21, and a web, or http, server listens to port 80).

- All ports below 1024 are considered well known; we can use them to implement standard services

NESO ACADEMY

Communication using Sockets



- When a client process initiates a request for a connection, it is assigned a port by the host computer.
- This port is some arbitrary number greater than 1024.

The packets traveling between the hosts are delivered to the appropriate process based on the destination port number.

NESO ACADEMY

Remote Procedure Calls (RPC)

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

- It is similar in many respects to the IPC mechanism.
- However,because we are dealing with an environment in which the processes are executing on separate systems, we must use a message based communication scheme to provide remote service.
- In contrast to the IPC facility, the messages exchanged in RPC communication are well structured and are thus no longerjust packets of data.
- Each message is addressed to an RPC daemon listening to a port on the remote system, and each contains an identifier of the function to execute and the parameters to pass to that function.
- The function is then executed as requested, and any output is sent back to the requester in a separate message.

NESO ACADEMY

The semantics of RPCs allow a client to invoke a procedure on a remote host as it would invoke a procedure locally

- The RPC system hides the details that allow communication to take place by providing a stub on the client side.
- Typically, a separate stub exists for each separate remote procedure.
- When the client invokes a remote procedure, the RPC system calls the appropriate stub, passing it the parameters provided to the remote procedure. This stub locates the port on the server and marshals the parameters.
- Parameter marshalling involves packaging the parameters into a form that can be transmitted over a network.
- The stub then transmits a message to the server using message passing.
- A similar stub on the server side receives this message and invokes the procedure on the server.
- If necessary, return values are passed back to the client using the same technique.

NESO ACADEMY

Issues in RPC and how they are resolved

Issues	How they are resolved
Differences in data representation on the client and server machines. Eg. Representation of 32-bit integers: Some systems (known as big-endian) use the high memory address to store the most significant byte, while other systems (known as little-endian) store the least significant byte at the high memory address.	RPC systems define a machine-independent representation of data. One such representation is known as external data representation (XDR). On the client side, parameter marshalling involves converting the machine dependent data into XDR before they are sent to the server. On the server side, the XDR data are unmarshalled and converted to the machine-dependent representation for the server.

NESO ACADEMY

Whereas local procedure calls fail only under extreme circumstances, RPCs can fail, or be duplicated and executed more than once, as a result of common network errors.	The operating system must ensure that messages are acted on exactly once, rather than at most once. Most local procedure calls have the "exactly once" functionality, but it is more difficult to implement.

NESO ACADEMY

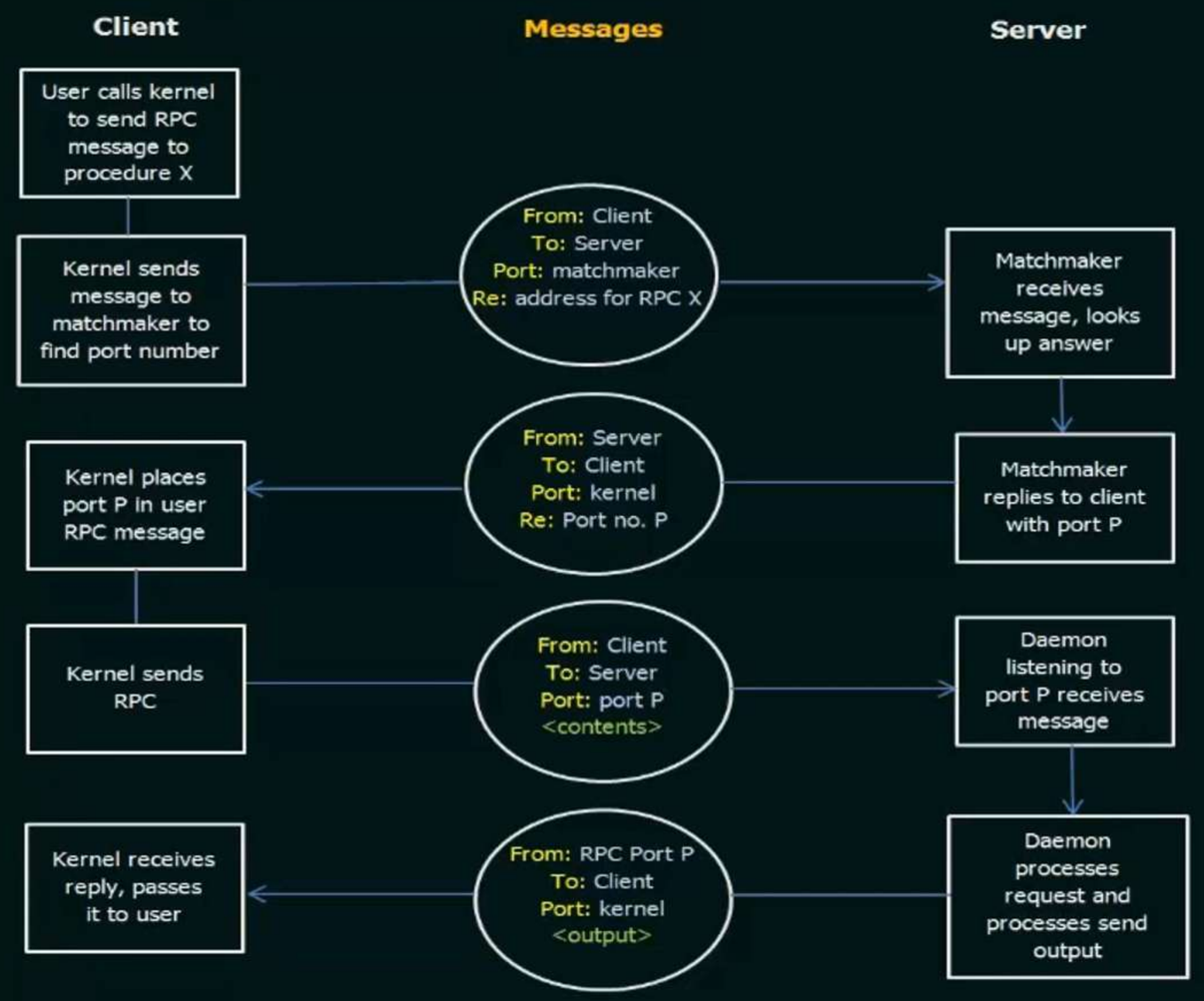
With standard procedure calls, some form of binding takes place during link, load, or execution time so that a procedure call's name is replaced by the memory address of the procedure call. The RPC scheme requires a similar binding of the client and the server port, but how does a client know the port numbers on the server? Neither system has full information about the other because they do not share memory.

1) The binding information may be predetermined, in the form of fixed port addresses. At compile time, an RPC call has a fixed port number associated with it. Once a program is compiled, the server cannot change the port number of the requested service.

2) Binding can be done dynamically by a rendezvous mechanism. Typically, an operating system provides a rendezvous (also called a matchmaker) daemon on a fixed RPC port. A client then sends a message containing the name of the RPC to the rendezvous daemon requesting the port address of the RPC it needs to execute. The port number is returned, and the RPC calls can be sent to that port until the process terminates (or the server crashes).

NESO ACADEMY

Execution of a remote procedure call (RPC)



NESO ACADEMY

Threads

A thread is a basic unit of CPU utilization.

It comprises

- A thread ID
- A program counter
- A register set and
- A stack

It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.

A traditional / heavyweight process has a single thread of control.

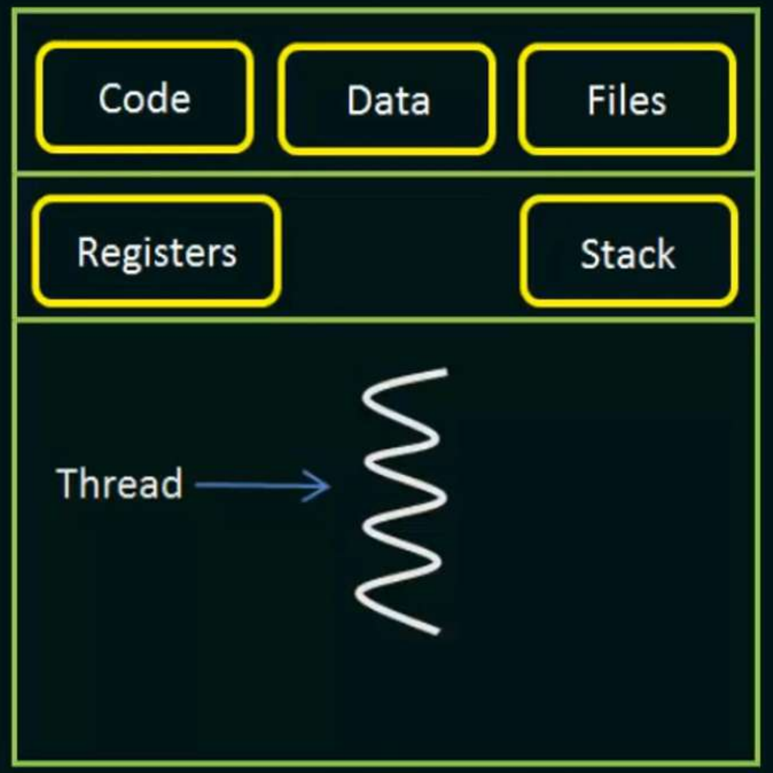
If a process has multiple threads of control, it can perform more than one task at a time.

NESO ACADEMY

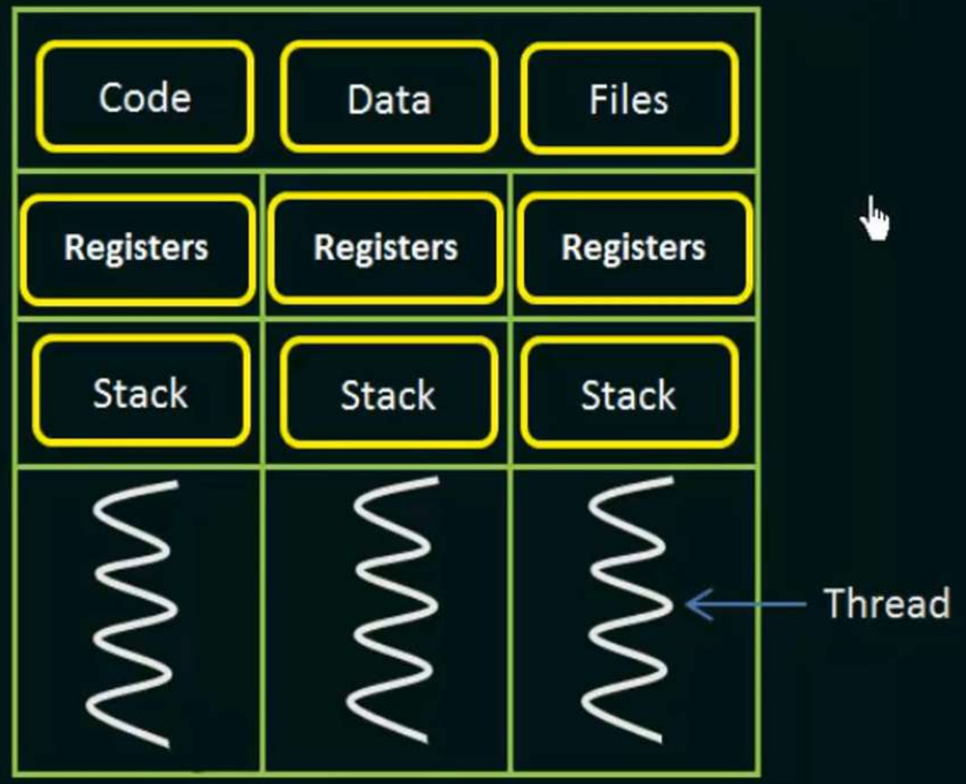
It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.

A traditional / heavyweight process has a single thread of control.

If a process has multiple threads of control, it can perform more than one task at a time.



Single-threaded process



Multi-threaded process

NESO ACADEMY

The benefits of multithreaded programming can be broken down into four major categories:

Responsiveness	Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.
Resource sharing	By default, threads share the memory and the resources of the process to which they belong. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.
Economy	Allocating memory and resources for process creation is costly. Because threads share resources of the process to which they belong, it is more economical to create and context-switch threads.
Utilization of multiprocessor architectures	The benefits of multithreading can be greatly increased in a multiprocessor architecture, where threads may be running in parallel on different processors. A single-threaded process can only run on one CPU, no matter how many are available. Multithreading on a multi-CPU machine increases concurrency.

NESO ACADEMY

Multithreading Models and Hyperthreading

Types of Threads:

1) User Threads - Supported above the kernel and are managed without kernel support.

2) Kernel Threads - Supported and managed directly by the operating system.

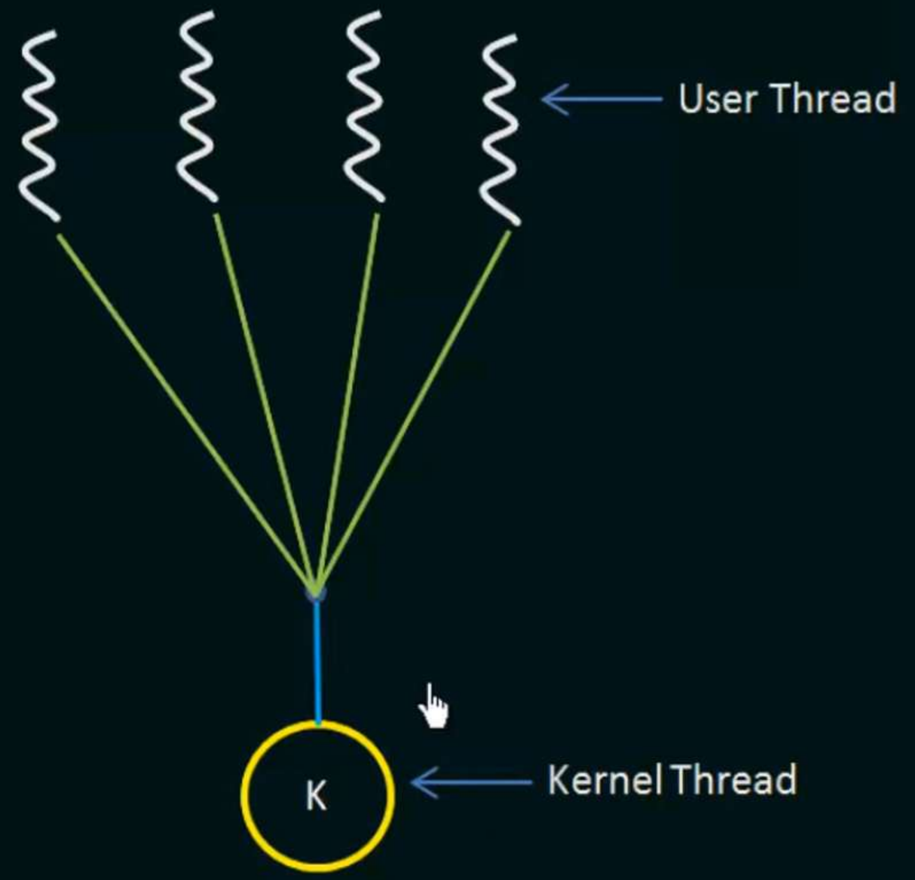
Ultimately, there must exist a relationship between user threads and kernel threads.

There are three common ways of establishing this relationship:

1. Many-to-One Model
2. One-to-One Model
3. Many-to-Many Model

NESO ACADEMY

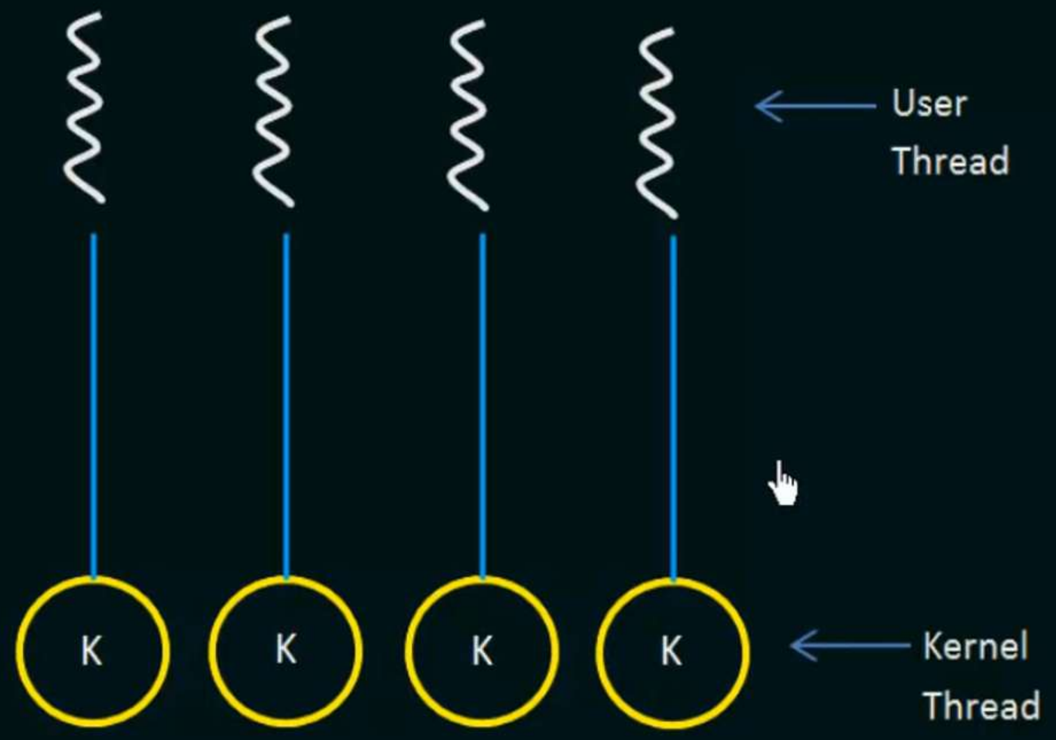
Many-to-One Model



- Maps many user-level threads to one kernel thread.
- Thread management is done by the thread library in user space, so it is efficient.
- The entire process will block if a thread makes a blocking system call.
- Because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.

NESO ACADEMY

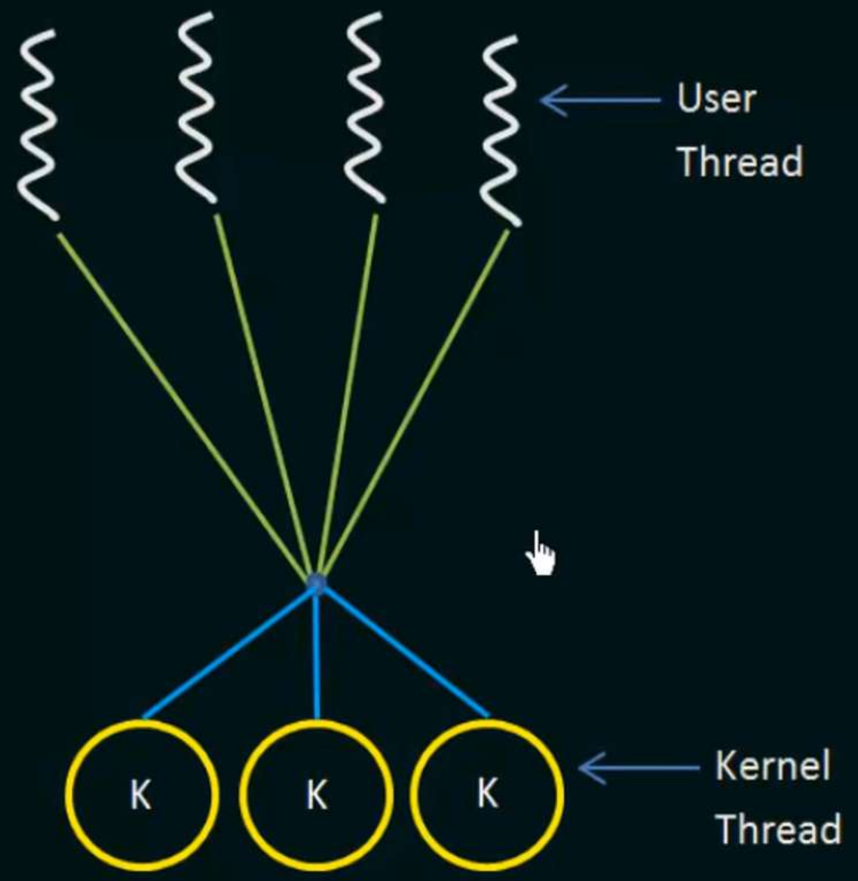
One-to-One Model



- Maps each user thread to a kernel thread.
- Provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call;
- Also allows multiple threads to run in parallel on multiprocessors.
- Creating a user thread requires creating the corresponding kernel thread.
- Because the overhead of creating kernel threads can burden the performance of an application, most implementations of this model restrict the number of threads supported by the system.

NESO ACADEMY

Many-to-Many Model



- Multiplexes many user-level threads to a smaller or equal number of kernel threads.
- The number of kernel threads may be specific to either a particular application or a particular machine.
- Developers can create as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor.
- Also, when a thread performs a blocking system call, the kernel can schedule another thread for execution.

NESO ACADEMY

Hyperthreading

or

Simultaneous Multithreading (SMT)

Hyperthreaded systems allow their processor cores' resources to become multiple logical processors for performance.

It enables the processor to execute two threads, or sets of instructions, at the same time. Since hyper-threading allows two streams to be executed in parallel, it is almost like having two separate processors working together.

NESO ACADEMY

The fork() and exec() System Calls

fork() : The **fork()** system call is used to create a separate, duplicate process.

exec() : When an **exec()** system call is invoked, the program specified in the parameter to exec () will replace the entire process — including all threads.

NESO ACADEMY

Total number of Processes = 2^n

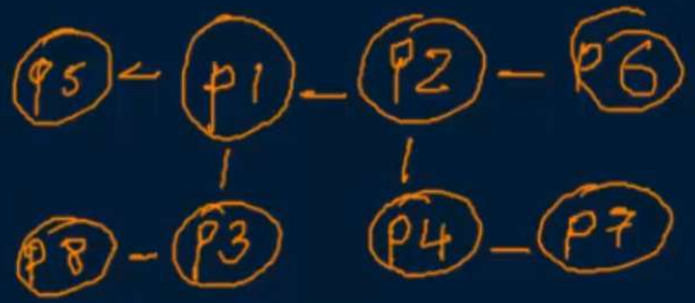
where n is the number of fork() system calls

File Edit View Search Tools Documents Help

Open Save Undo

fex1.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    fork();
    fork();
    fork();
    printf("Hello Neso Academy!\n PID = 

    return 0;
}
```




```
jaison@neso-academy ~/Desktop/Fork  $ ./a.out
Hello Neso Academy!
 PID = 5837
Hello Neso Academy!
 PID = 5838
jaison@neso-academy ~/Desktop/Fork  $ gcc fex1.c
jaison@neso-academy ~/Desktop/Fork  $ ./a.out
Hello Neso Academy!
 PID = 5850
Hello Neso Academy!
 PID = 5852
Hello Neso Academy!
Hello Neso Academy!
 PID = 5855
 PID = 5851
Hello Neso Academy!
 PID = 5853
Hello Neso Academy!
 PID = 5857
jaison@neso-academy ~/Desktop/Fork  $ Hello Neso Academy!
 PID = 5856
Hello Neso Academy!
 PID = 5854
```

C Tab Width: 4 Ln 8, Col 12

NESO ACADEMY

Open Save Undo

ex1.c ex2.c

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    printf("PID of ex1.c = %d\n", getpid());
    char *args[] = {"Hello", "Neso", "Academy", NULL};
    execv("./ex2", args);
    printf("Back to ex1.c");
    return 0;
}
```

jaison@neso-academy ~/Desktop/Exec

File Edit View Search Terminal Help

```
                    ||      ||
jaison@neso-academy ~/Desktop/Exec $ gcc ex1.c -o ex1
jaison@neso-academy ~/Desktop/Exec $ gcc ex2.c -o ex2
jaison@neso-academy ~/Desktop/Exec $ ./ex1
PID of ex1.c = 5962
We are in ex2.c
PID of ex2.c = 5962
jaison@neso-academy ~/Desktop/Exec $
```

NESO ACADEMY

Open Save Undo

ex1.c ex2.c

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    printf("We are in ex2.c\n");
    printf("PID of ex2.c = %d\n", getpid());
    return 0;
}
```

jaison@neso-academy ~/Desktop/Exec

File Edit View Search Terminal Help

```
                    ||      ||
jaison@neso-academy ~/Desktop/Exec $ gcc ex1.c -o ex1
jaison@neso-academy ~/Desktop/Exec $ gcc ex2.c -o ex2
jaison@neso-academy ~/Desktop/Exec $ ./ex1
PID of ex1.c = 5962
We are in ex2.c
PID of ex2.c = 5962
jaison@neso-academy ~/Desktop/Exec $
```

NESO ACADEMY

Threading Issues (Part-1)

The fork() and exec() System Calls

The semantics of the fork() and exec() system calls change in a multithreaded program.

Issue

If one thread in a program calls fork(), does the new process duplicate all threads, or is the new process single-threaded?

Solution

Some UNIX systems have chosen to have two versions of fork (), one that duplicates all threads and another that duplicates only the thread that invoked the fork () system call.

But which version of fork () to use and when ?

Also, if a thread invokes the exec () system call, the program specified in the parameter to exec () will replace the entire process —including all threads.

NESO ACADEMY

But which version of fork () to use and when ?

Also, if a thread invokes the exec () system call, the program specified in the parameter to exec () will replace the entire process —including all threads.

Which of the two versions of fork () to use depends on the application.

If exec() is called immediately after forking

Then duplicating all threads is unnecessary, as the program specified in the parameters to exec () will replace the process.
In this instance, duplicating only the calling thread is appropriate.

If the separate process does not call exec () after forking

Then the separate process should duplicate all threads.

NESO ACADEMY

Threading Issues (Part-2)

Thread Cancellation

Thread cancellation is the task of terminating a thread before it has completed.

If multiple threads are concurrently searching through a database and one thread returns the result, the remaining threads might be canceled.

When a user presses a button on a web browser that stops a web page from loading any further, all threads loading the page are canceled.

A thread that is to be canceled is often referred to as the target thread.

NESO ACADEMY

Cancellation of a target thread may occur in two different scenarios:

1. Asynchronous cancellation: One thread immediately terminates the target thread.

2. Deferred cancellation: The target thread periodically checks whether it should terminate, allowing it an opportunity to terminate itself in an orderly fashion.

Where the difficulty with cancellation lies:

NESO ACADEMY

Where the difficulty with cancellation lies:

In situations where:

- Resources have been allocated to a canceled thread
- A thread is canceled while in the midst of updating data it is sharing with other threads.

Often, the OS will reclaim system resources from a canceled thread but will not reclaim all resources.

Therefore, canceling a thread asynchronously may not free a necessary system-wide resource.

NESO ACADEMY

Often, the OS will reclaim system resources from a canceled thread but will not reclaim all resources.

Therefore, canceling a thread asynchronously may not free a necessary system-wide resource.

With deferred cancellation:

One thread indicates that a target thread is to be canceled.

But cancellation occurs only after the target thread has checked a flag to determine if it should be canceled or not.

This allows a thread to check whether it should be canceled at a point when it can be canceled safely.

NESO ACADEMY

CPU Scheduling

CPU scheduling is the basis of multiprogrammed operating systems.

By switching the CPU among processes, the operating system can make the computer more productive.

Topics to be covered:

- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems.
- To describe various CPU-scheduling algorithms

Let's get the basics right

NEO ACADEMY

Let's get the basics right

- In a single-processor system, only one process can run at a time.
- Any others must wait until the CPU is free and can be rescheduled.
    - The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
    - A process is executed until it must wait, typically for the completion of some I/O request.

In a simple computer system, the CPU then just sits idle.

All this waiting time is wasted; no useful work is accomplished.

NESO ACADEMY

In a simple computer system, the CPU then just sits idle.

All this waiting time is wasted; no useful work is accomplished.

- With multiprogramming, we try to use this time productively.
  - Several processes are kept in memory at one time.

- When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process and this pattern continues.

NESO ACADEMY

CPU and I/O Burst Cycles

Process execution consists of a cycle of CPU execution and I/O wait.

Processes alternate between these two states.

Process execution begins with a

CPU burst That is followed by an

I/O burst , which is followed by another

CPU burst , then another

I/O burst ,

and so on...

CPU burst is when the process is being executed in the CPU

I/O burst is when the CPU is waiting for I/O for further execution

Eventually, the final CPU burst ends with a system request to terminate execution.

NESO ACADEMY

load store
add store
read from file

CPU Burst

wait for I/O

I/O Burst

store increment
index
write to file

CPU Burst

wait for I/O

I/O Burst

load store
add store
read from file

CPU Burst

wait for I/O

I/O Burst

Alternating sequence of
CPU
&
I/O bursts

NESO ACADEMY

Preemptive and Non-Preemptive Scheduling

CPU Scheduler

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.

Dispatcher

The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler.

The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency.

NESO ACADEMY

CPU-scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state.
2. When a process switches from the running state to the ready state (for example, when an interrupt occurs).
3. When a process switches from the waiting state to the ready state (for example, at completion of I/O).
4. When a process terminates.

For situations 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution.
However, there is a choice for situations 2 and 3.

When scheduling takes place only under circumstances 1 and 4, we say that the scheduling scheme is nonpreemptive or cooperative; otherwise, it is preemptive.

NESO ACADEMY

Scheduling Criteria

CPU utilization

We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).

Throughput

If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called throughput.

Turnaround time

From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

NESO ACADEMY

Waiting time

The CPU scheduling algorithm does not affect the amount of time during which a process executes or does I/O; it affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.

Response time

In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. This measure, called response time, is the time it takes to start responding, not the time it takes to output the response. The turnaround time is generally limited by the speed of the output device.

NESO ACADEMY

Scheduling Algorithms
(First-Come, First-Served Scheduling)

- By far the simplest CPU-scheduling algorithm.
- The process that requests the CPU first is allocated the CPU first.
- The implementation of the FCFS policy is easily managed with a FIFO queue.

- When a process enters the ready queue, its PCB is linked onto the tail of the queue.
- When the CPU is free, it is allocated to the process at the head of the queue.
- The running process is then removed from the queue.

NESO ACADEMY

Consider the following set of processes that arrive at time 0

Process	Burst Time (ms)
P1	24
P2	3
P3	3

If the processes arrive in the order P1, P2, P3, and are served in FCFS order, we get the result shown in the following Gantt chart:

Waiting Time for P1 = 0 ms

Waiting Time for P2 = 24 ms

Waiting Time for P3 = 27 ms

Average Waiting Time = (0 + 24 + 27)/3 = 17 ms

NESO ACADEMY

If the processes arrive in the order P2, P3, P1, however the result will be shown in the following Gantt chart:

Waiting Time for P1 = 6 ms

Waiting Time for P2 = 0 ms

Waiting Time for P3 = 3 ms

Average Waiting Time = (6 + 0 + 3)/3 = 3 ms

This reduction is substantial. Thus, the average waiting time under an FCFS policy is generally not minimal and may vary substantially if the process's CPU burst times vary greatly.

NESO ACADEMY

This reduction is substantial. Thus, the average waiting time under an FCFS policy is generally not minimal and may vary substantially if the process's CPU burst times vary greatly.

The FCFS scheduling algorithm is nonpreemptive

- Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O.
- The FCFS algorithm is thus particularly troublesome for time-sharing systems, where it is important that each user get a share of the CPU at regular intervals.
- It would be disastrous to allow one process to keep the CPU for an extended period.

NESO ACADEMY

First Come First Served Scheduling (Solved Problem 1)

First-Come, First-Served Scheduling

Solved Problem -1

Convoy Effect

If processes with higher burst time arrived before the processes with smaller burst time, then, smaller processes have to wait for a long time for longer processes to release the CPU.

1:18 / 18:49

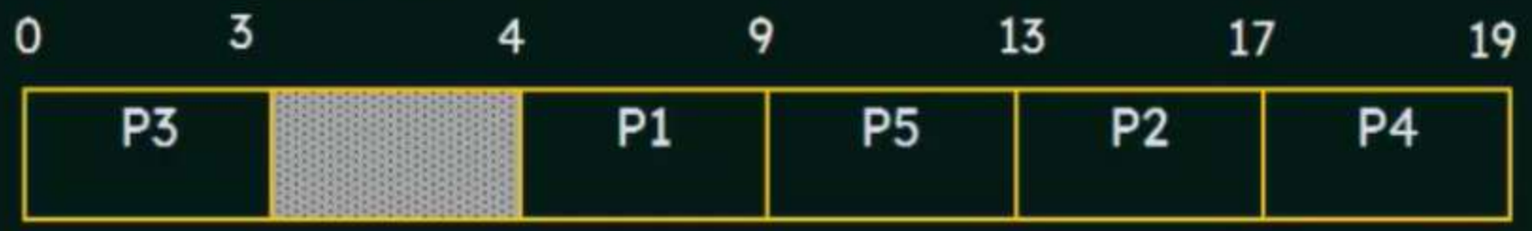
Consider the set of 5 processes whose arrival time and burst time are given below:

Process ID	Arrival Time	Burst Time
P1	4	5
P2	6	4
P3	0	3
P4	6	2
P5	5	4

Calculate the average waiting time and average turnaround time, if FCFS Scheduling Algorithm is followed.

Solution:

Gantt Chart:



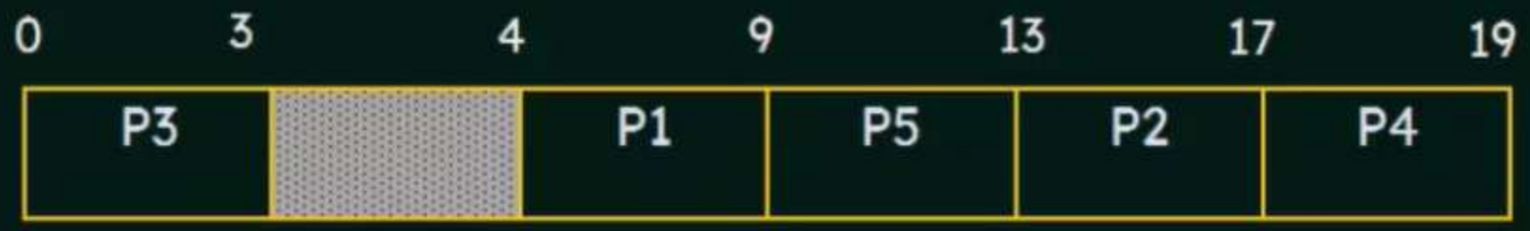
The shaded box represents the idle time of CPU

NESO ACADEMY

Solution:

Process ID	Arrival Time	Burst Time
P1	4	5
P2	6	4
P3	0	3
P4	6	2
P5	5	4

Gantt Chart:



The shaded box represents the idle time of CPU

Turn Around time = Completion time – Arrival time

Waiting time = Turn Around time – Burst time

Process ID	Completion Time	Turnaround Time	Waiting Time
P1	9	9 – 4 = 5	5 – 5 = 0
P2	17	17 – 6 = 11	11 – 4 = 7
P3	3	3 – 0 = 3	3 – 3 = 0
P4	19	19 – 6 = 13	13 – 2 = 11
P5	13	13 – 5 = 8	8 – 4 = 4

NESO ACADEMY

First-Come, First-Served Scheduling

Solved Problem - 2

The arrival times and burst times for a set of 6 processes are given in the table below:

Process ID	Arrival Time	Burst Time
P1	0	3
P2	1	2
P3	2	1
P4	3	4
P5	4	5
P6	5	2

If FCFS Scheduling Algorithm is followed and there is 1 unit of overhead in scheduling the processes, find the efficiency of the algorithm.

NESO ACADEMY

Process ID	Arrival Time	Burst Time
P1	0	3
P2	1	2
P3	2	1
P4	3	4
P5	4	5
P6	5	2

If FCFS Scheduling Algorithm is followed and there is 1 unit of overhead in scheduling the processes, find the efficiency of the algorithm.

Solution:

Gantt Chart:

0–1	1–4	4–5	5–7	7–8	8–9	9–10	10–14	14–15	15–20	20–21	21–23
δ	P1	δ	P2	δ	P3	δ	P4	δ	P5	δ	P6

Here, δ denotes the unit overhead in scheduling the processes

NESO ACADEMY

Solution:

Gantt Chart:

0–1	1–4	4–5	5–7	7–8	8–9	9–10	10–14	14–15	15–20	20–21	21–23
δ	P1	δ	P2	δ	P3	δ	P4	δ	P5	δ	P6

Here, δ denotes the unit overhead in scheduling the processes

Now,

Useless time or Wasted time = 6 x δ = 6 x 1

= 6 units

Total time = 23 units

Useful time = 23 units – 6 units

= 17 units

Efficiency (η) = Useful time / Total Time

= 17 units / 23 units

= 0.7391

= 73.91%

NESO ACADEMY

Scheduling Algorithms
(Shortest-Job-First Scheduling)

- This algorithm associates with each process the length of the process's next CPU burst.
- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
- If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.

The SJF algorithm can be either preemptive or nonpreemptive

A more appropriate term for this scheduling method would be the

Shortest-Next-CPU-Burst Algorithm

because scheduling depends on the length of the next CPU burst of a process, rather than its total length.

NESO ACADEMY

Example of SJF Scheduling (Non-Premptive)

Consider the following set of processes,with the length of the CPU burst given in milliseconds:

Process ID	Burst Time
P1	6
P2	8
P3	7
P4	3

Waiting Time for P1 = 3 ms

Waiting Time for P2 = 16 ms

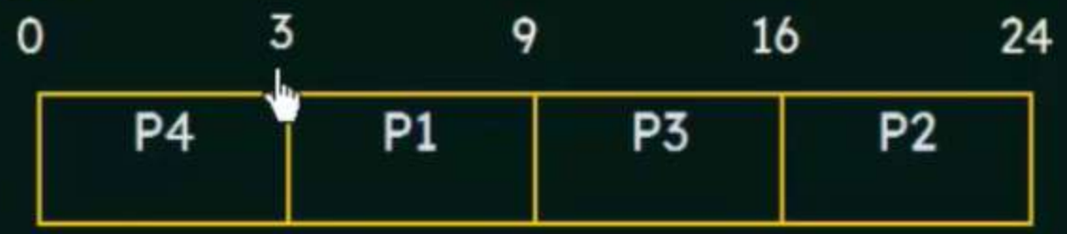
Waiting Time for P3 = 9 ms

Waiting Time for P4 = 0 ms

Average Waiting Time

= (3 + 16 + 9 + 0)/4 = 7 ms

Gantt Chart:



By comparison, if we were using the FCFS scheduling scheme, the average waiting time would be 10.25 milliseconds.

NESO ACADEMY

Example of SJF Scheduling (Premptive)

Consider the following four processes, with the length of the CPU burst given in milliseconds and the processes arrive at the ready queue at the times shown:

Process ID	Arrival Time	Burst Time
P1	0	~~8~~ 7
P2	1	4
P3	2	9
P4	3	5

Waiting Time for P1 = (10 – 1 - 0) = 9 ms

Waiting Time for P2 = (1 – 0 - 1) = 0 ms

Waiting Time for P3 = (17 – 0 - 2) = 15 ms

Waiting Time for P4 = (5 – 0 - 3) = 2 ms

Gantt Chart:

0	1	5	10	17	26
P1	P2	P4	P1	P3	

Average Waiting Time

= (9 + 0 + 15 + 2)/4 = 6.5 ms

Waiting Time = Total waiting Time – No. of milliseconds Process executed – Arrival Time

NESO ACADEMY

Problems with SJF Scheduling:

- The real difficulty with the SJF algorithm is knowing the length of the next CPU request.
- Although the SJF algorithm is optimal, it cannot be implemented at the level of short-term CPU scheduling.
- There is no way to know the length of the next CPU burst.

One approach is:

- To try to approximate SJF scheduling.
- We may not know the length of the next CPU burst, but we may be able to predict its value.
- We expect that the next CPU burst will be similar in length to the previous ones.
- Thus, by computing an approximation of the length of the next CPU burst, we can pick the process with the shortest predicted CPU burst.

NESO ACADEMY

Problems with SJF Scheduling:

- The real difficulty with the SJF algorithm is knowing the length of the next CPU request.
- Although the SJF algorithm is optimal, it cannot be implemented at the level of short-term CPU scheduling.
- There is no way to know the length of the next CPU burst.

One approach is:

- To try to approximate SJF scheduling.
- We may not know the length of the next CPU burst, but we may be able to predict its value.
- We expect that the next CPU burst will be similar in length to the previous ones.
- Thus, by computing an approximation of the length of the next CPU burst, we can pick the process with the shortest predicted CPU burst.

NESO ACADEMY

Shortest-Job-First Scheduling

Solved Problem - 2

GATE 2016

Consider the following processes, with the arrival time and the length of the CPU burst given in milliseconds. The scheduling algorithm used is preemptive shortest remaining-time first.

Process ID	Arrival Time	Burst Time
P1	0	10
P2	3	6
P3	7	1
P4	8	3

The average turn around time of these processes is _____ milliseconds.

(A) 8.25 (B) 10.25 (C) 6.35 (D) 4.25

NESO ACADEMY

Process ID	Arrival Time	Burst Time
P1	0	~~10~~ 7
P2	3	~~6~~ 2
P3	7	1
P4	8	3

Turnaround Time for P1 = (20 – 0) = 20 ms

Turnaround Time for P2 = (10 – 3) = 7 ms

Turnaround Time for P3 = (8 – 7) = 1 ms

Turnaround Time for P4 = (13– 8) = 5 ms

Average Turnaround Time

= (20 + 7 + 1 + 5)/4

= 33/4

= 8.25 ms

Solution:

Gantt Chart:

0 – 3	3 – 7	7 – 8	8 – 10	10 – 13	13 – 20
P1	P2	P3	P2	P4	P1

Turn Around time = Completion time – Arrival time

NESO ACADEMY

Scheduling Algorithms
(Priority Scheduling)

- A priority is associated with each process, and the CPU is allocated to the process with the highest priority.

- Equal-priority processes are scheduled in FCFS order.

- An SJF algorithm is simply a priority algorithm where the priority is the inverse of the (predicted) next CPU burst.
  The larger the CPU burst, the lower the priority, and vice versa.

Priority scheduling can be either preemptive or nonpreemptive.

A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.

NESO ACADEMY

Problem with Priority Scheduling

- A major problem with priority scheduling algorithms is indefinite blocking, or starvation.
- A process that is ready to run but waiting for the CPU can be considered blocked.
- A priority scheduling algorithm can leave some low priority processes waiting indefinitely.
- In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

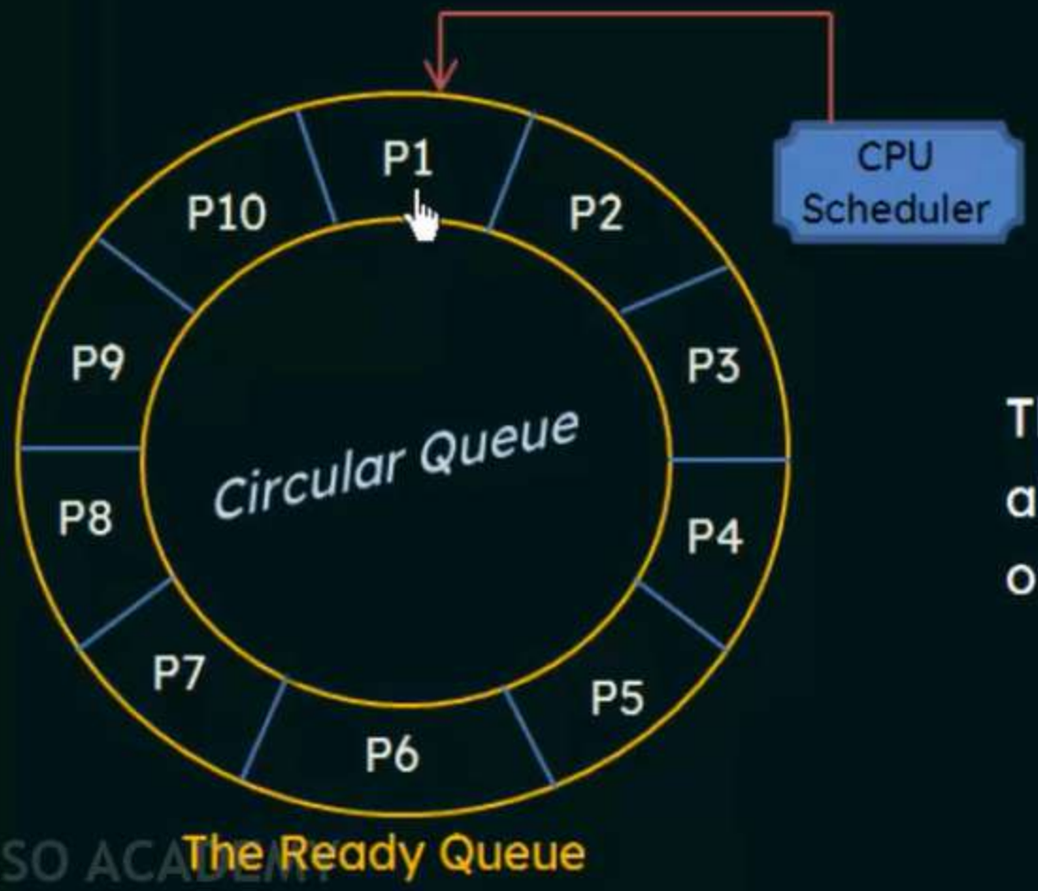
Solution to the Problem

- A solution to the problem of indefinite blockage of low-priority processes is **aging**.
- Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.
- For example,
  If priorities range from 127 (low) to 0 (high), we could increase the priority of a waiting process by 1 every 15 minutes.
- Eventually, even a process with an initial priority of 127 would have the highest priority in the system and would be executed.

NESO ACADEMY

Scheduling Algorithms
(Round-Robin Scheduling)

- The round-robin (RR) scheduling algorithm is designed especially for timesharing systems.
- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- A small unit of time, called a time quantum or time slice, is defined (generally from 10 to 100 milliseconds)



- The ready queue is treated as a circular queue.

The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.

NESO ACADEMY

Implementation of Round Robin scheduling:

- We keep the ready queue as a FIFO queue of processes.
- New processes are added to the tail of the ready queue.
- The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.

One of two things will then happen

The process may have a CPU burst of less than 1 time quantum

↓

The process itself will release the CPU voluntarily

↓

The CPU scheduler will then proceed to the next process in the ready queue

The CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the O.S.

↓

A context switch will be executed, and the process will be put at the tail of the ready queue

↓

The CPU scheduler will then select the next process in the ready queue

NESO ACADEMY

Round-Robin Scheduling
(Turnaround Time and Waiting Time)

Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds and time quantum taken as 4 milliseconds for RR Scheduling:

Process ID	Burst Time
P1	24
P2	3
P3	3

Time Quantum

Gantt Chart:

0	4	7	10	14	18	22	26	30
P1	P2	P3	P1	P1	P1	P1	P1	

NESO ACADEMY

Process ID	Burst Time
P1	24
P2	3
P3	3

Gantt Chart:

Time Quantum 4 ms

0 – 4	4 – 7	7 – 10	10 – 14	14 – 18	18 – 22	22 – 26	26 – 30
P1	P2	P3	P1	P1	P1	P1	P1

Method 1

Turn Around time = Completion time – Arrival time

Waiting time = Turn Around time – Burst time

Process ID	Completion Time	Turnaround Time	Waiting Time
P1	30	30 – 0 = 30	30 – 24 = 6
P2	7	7 – 0 = 7	7 – 3 = 4
P3	10	10 – 0 = 10	10 – 3 = 7

Method 2

Waiting time = Last Start Time - Arrival Time – (Preemption x Time Quantum)

Process ID	Waiting Time
P1	26 – 0 – (5x4) = 6
P2	4 – 0 – (0x4) = 4
P3	7 – 0 – (0x4) = 7

NESO ACADEMY

Round Robin Scheduling

Solved Problem

(Part - 1)

Consider the set of 5 processes whose arrival time and burst time are given below:

Process ID	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

If the CPU scheduling policy is Round Robin with time quantum = 2 units, calculate the average waiting time and average turn around time.

NESO ACADEMY

Process ID	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

Time Quantum
2 Units

Clock

1

Gantt Chart:

0–2	2–4	4–5	5–7	7–9	9–11	11–12	12–13	13–14
P1	P2	P3	P1	P4	P5	P2	P1	P5

Ready Queue

NESO ACADEMY

Turn Around time = Completion time – Arrival time

Waiting time = Turn Around time – Burst time

Process ID	Completion Time	Turnaround Time	Waiting Time
P1	13	13 – 0 = 13	13 – 5 = 8
P2	12	12 – 1 = 11	11 – 3 = 8
P3	5	5 – 2 = 3	3 – 1 = 2
P4	9	9 – 3 = 6	6 – 2 = 4
P5	14	14 – 4 = 10	10 – 3 = 7

Process ID	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

Average Turn Around time

= (13 + 11 + 3 + 6 + 10) / 5

= 43 / 5 = **8.6 units**

Average waiting time

= (8 + 8 + 2 + 4 + 7) / 5

= 29 / 5 = **5.8 units**

NESO ACADEMY

Scheduling Algorithms
(Multilevel Queue Scheduling)

A class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.

Example:

Foreground Processes (Interactive)

They have:

- Different response-time requirements
- Different scheduling needs

Background Processes (Batch)

In addition, foreground processes may have priority (externally defined) over background processes.

A multilevel queue scheduling algorithm partitions the ready queue into several separate queues.

NESO ACADEMY

A multilevel queue scheduling algorithm partitions the ready queue into several separate queues.

- The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.
- Each queue has its own scheduling algorithm.

Example:

Separate queues might be used for foreground and background processes.

The foreground queue might be scheduled by an RR algorithm, while the background queue is scheduled by an FCFS algorithm.

In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling.

For example, the foreground queue may have absolute priority over the background queue.

NESO ACADEMY

An example of a multilevel queue scheduling algorithm with five queues, listed below in order of priority:

highest priority

System processes

Interactive processes

Interactive editing processes

Batch processes

Student processes

lowest priority

NESO ACADEMY

Scheduling Algorithms
(Multilevel Feedback-Queue Scheduling)

The multilevel feedback-queue scheduling algorithm allows a process to move between queues.

- The idea is to separate processes according to the characteristics of their CPU bursts.
- If a process uses too much CPU time, it will be moved to a lower-priority queue.
- This scheme leaves I/O-bound and interactive processes in the higher-priority queues.
- In addition, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue.

This form of aging prevents starvation.

NESO ACADEMY

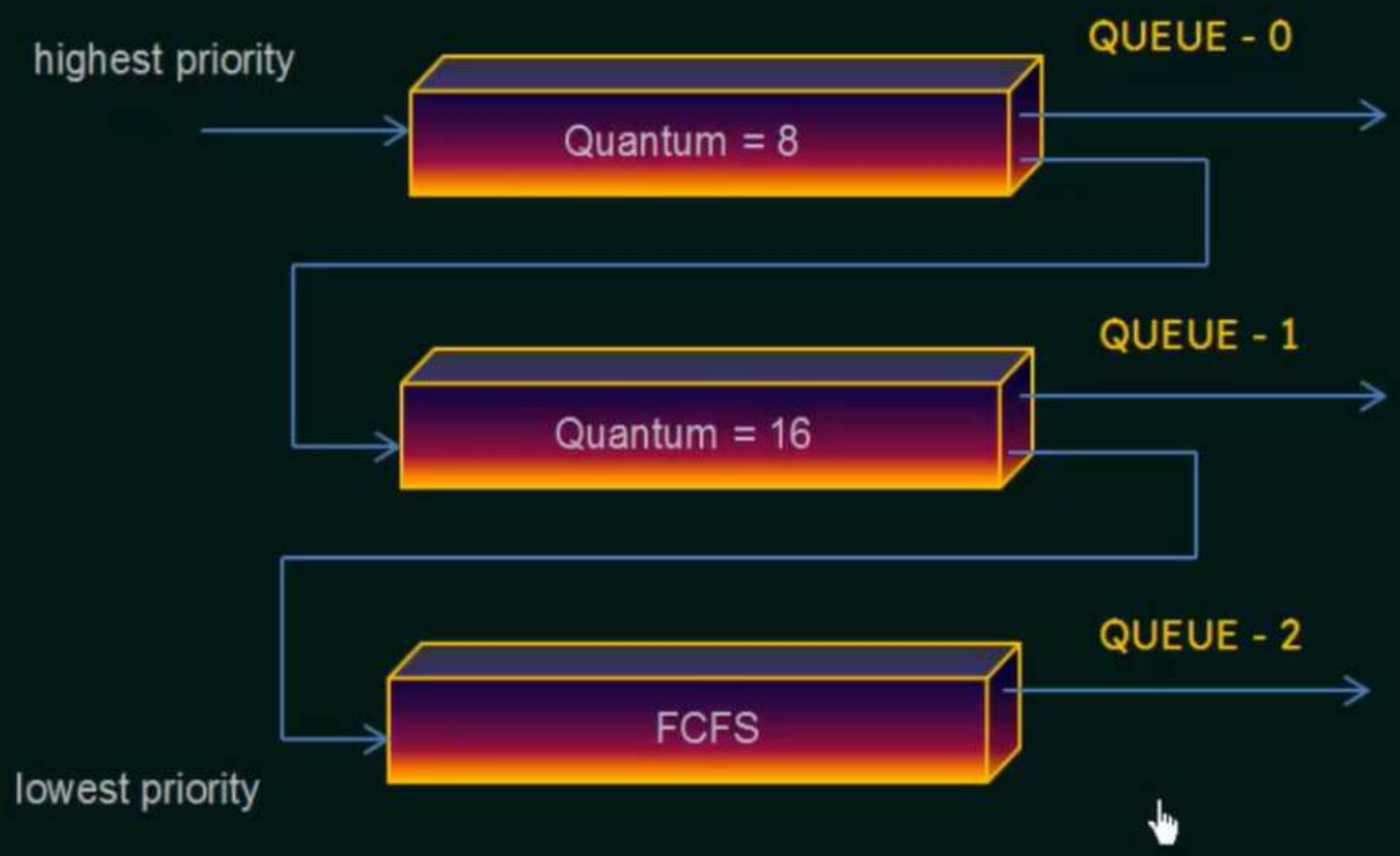


Figure: Multilevel feedback queues

NESO ACADEMY

In general, a multilevel feedback-queue scheduler is defined by the following parameters:

- The number of queues
- The scheduling algorithm for each queue
- The method used to determine when to upgrade a process to a higher priority queue
- The method used to determine when to demote a process to a lower priority queue
- The method used to determine which queue a process will enter when that process needs service

NESO ACADEMY