23MCAC103-
ADVANCED OPERATING SYSTEMS

by
Dr.Gobi Natesan

Course Outcome –Unit 1

- **Compare** the different kinds of CPU Scheduling algorithms in an Operating System based on processes arrival time, burst time and Turnaround time.

Syllabus- Unit 1

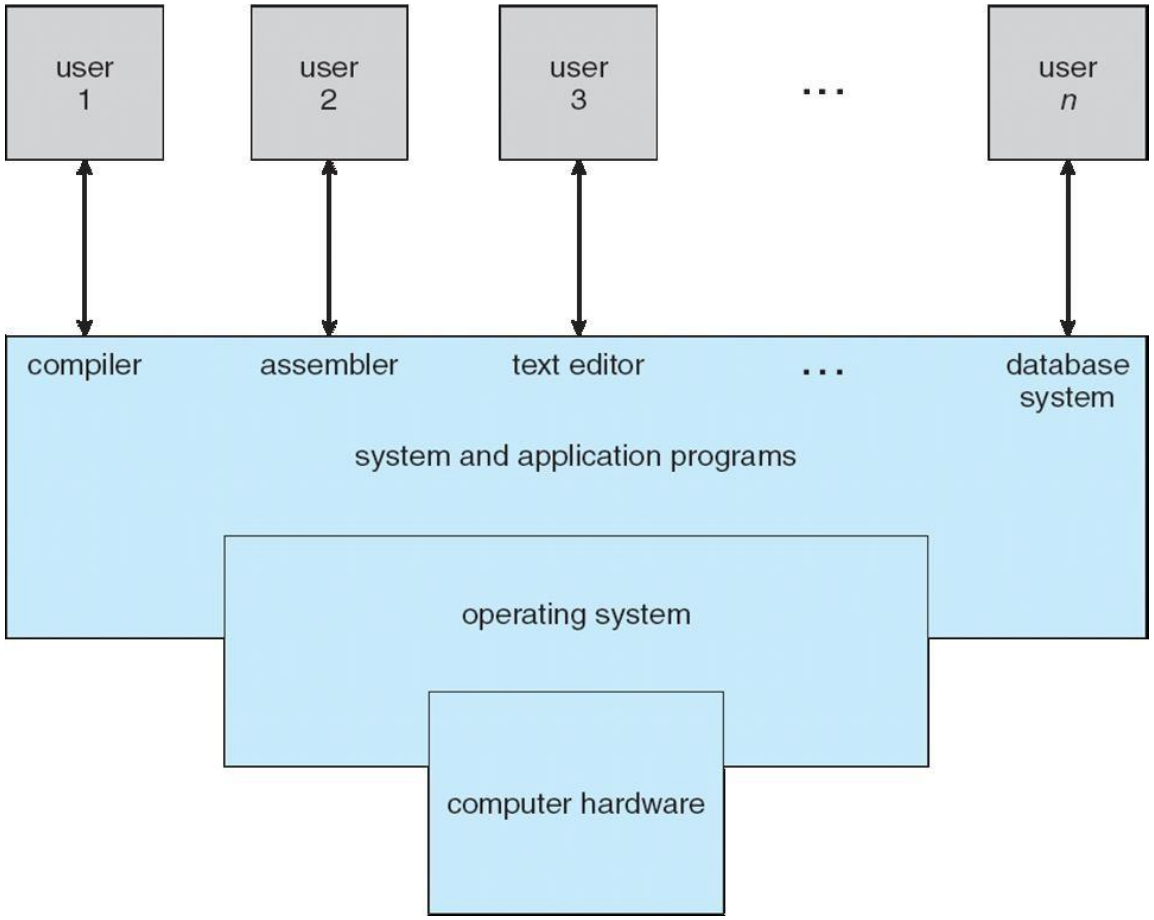
Fundamentals of Operating Systems:

Computer System organization, Operating System structure, Operating System operations, Operating System Services, System calls, Processes- CPU Scheduling - Case Study

Computer System Structure

- Computer system can be divided into four components:
  - **Hardware** – provides basic computing resources
    - CPU, memory, I/O devices
  - **Operating system**
    - Controls and coordinates use of hardware among various applications and users
  - **Application programs** – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - **Users**
    - People, machines, other computers

Four Components of a Computer System



Operating System-Definition

- A program that acts as an intermediary between a user of a computer and the computer hardware

- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner

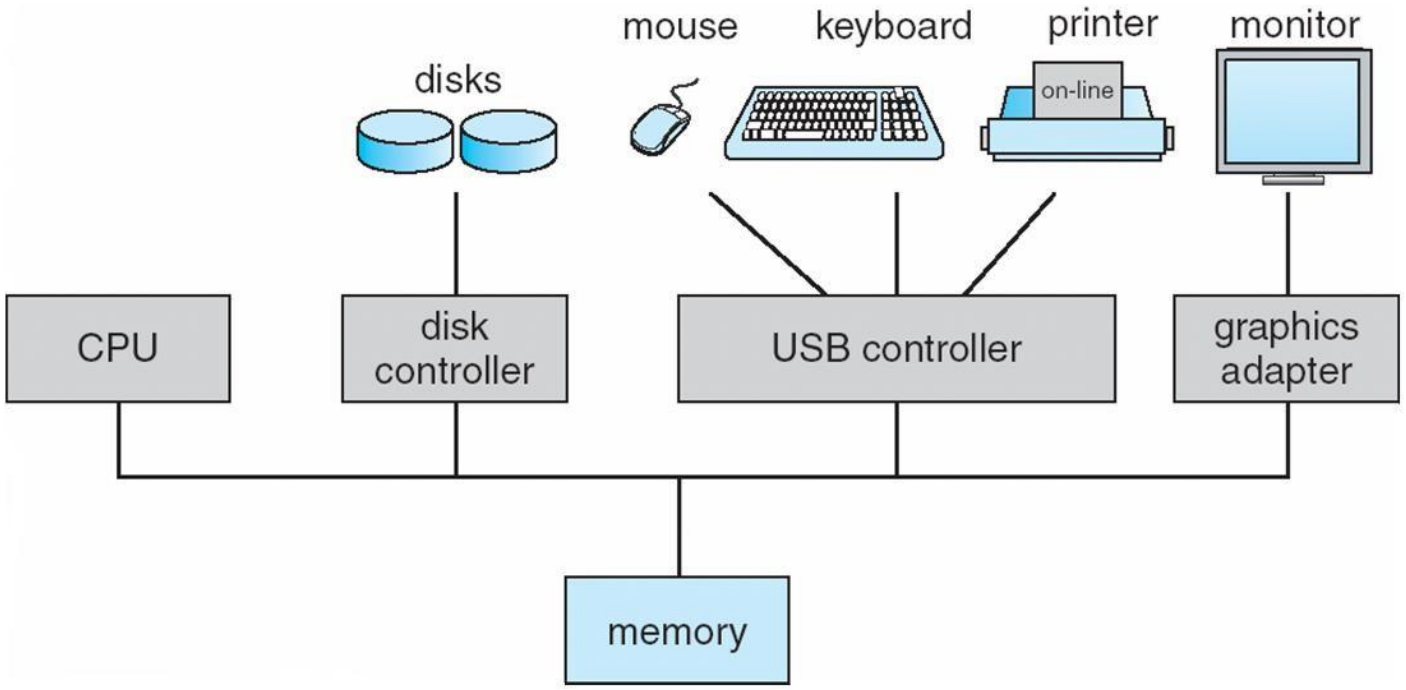
Operating System Definition

- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use

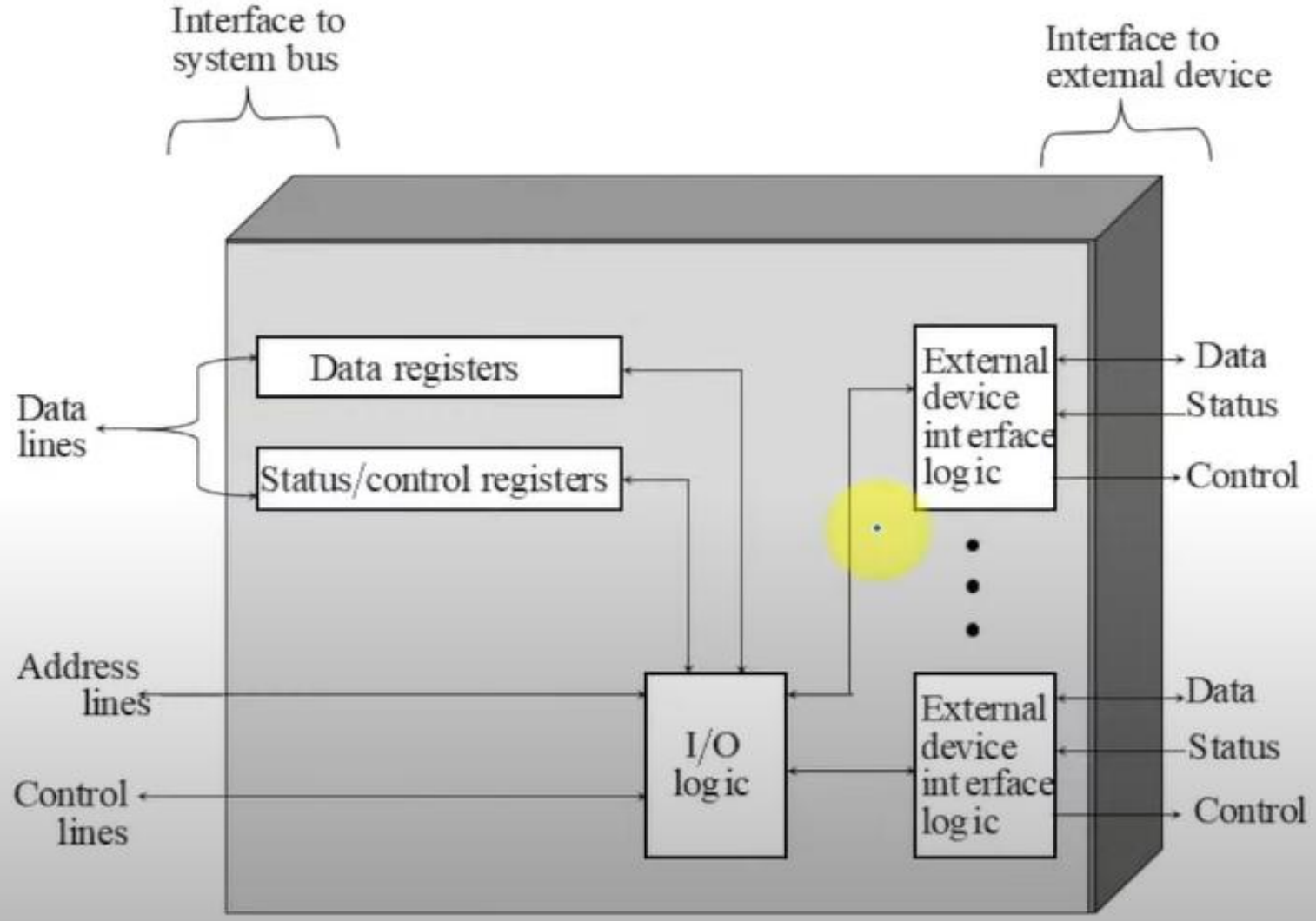
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles



I/O Devices and Interrupt



Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an interrupt

Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request.
- An operating system is **interrupt driven.**

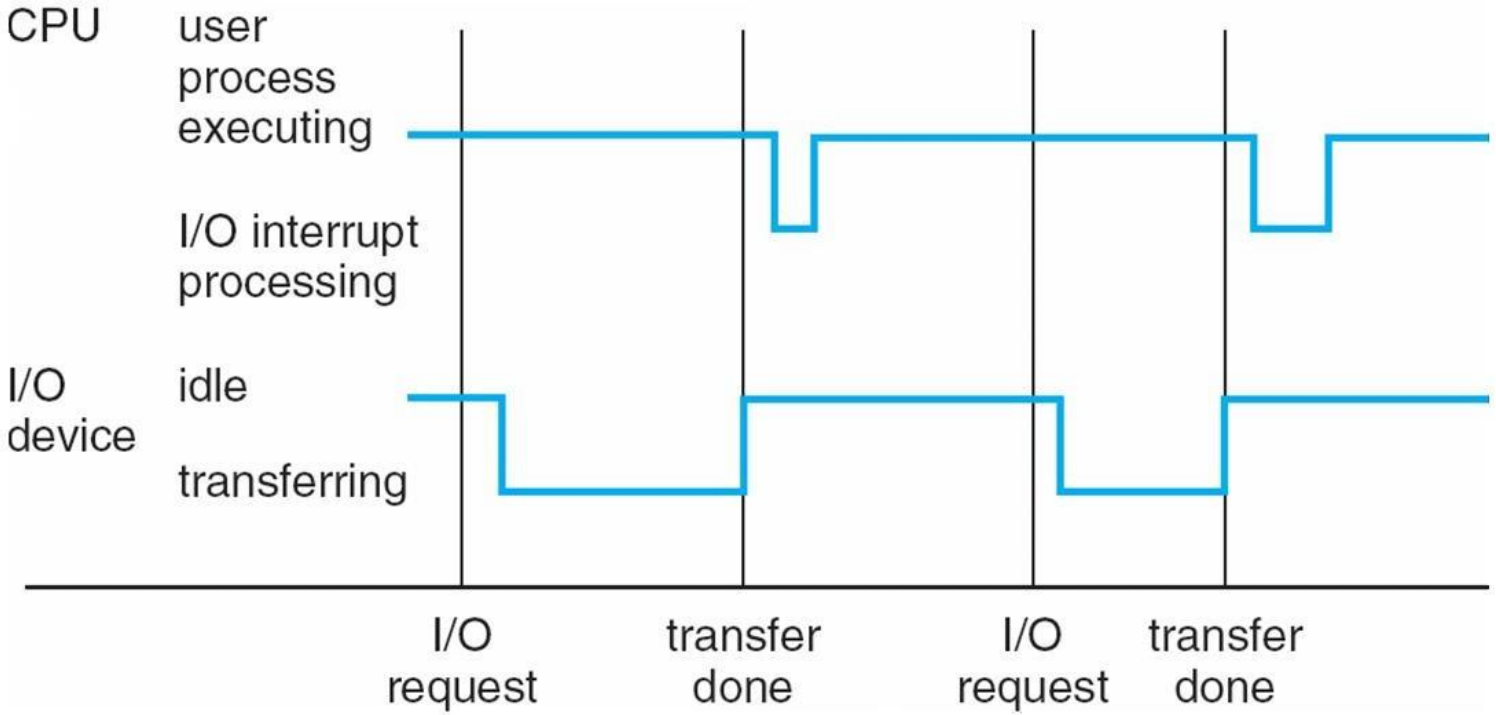
Interrupt Handling

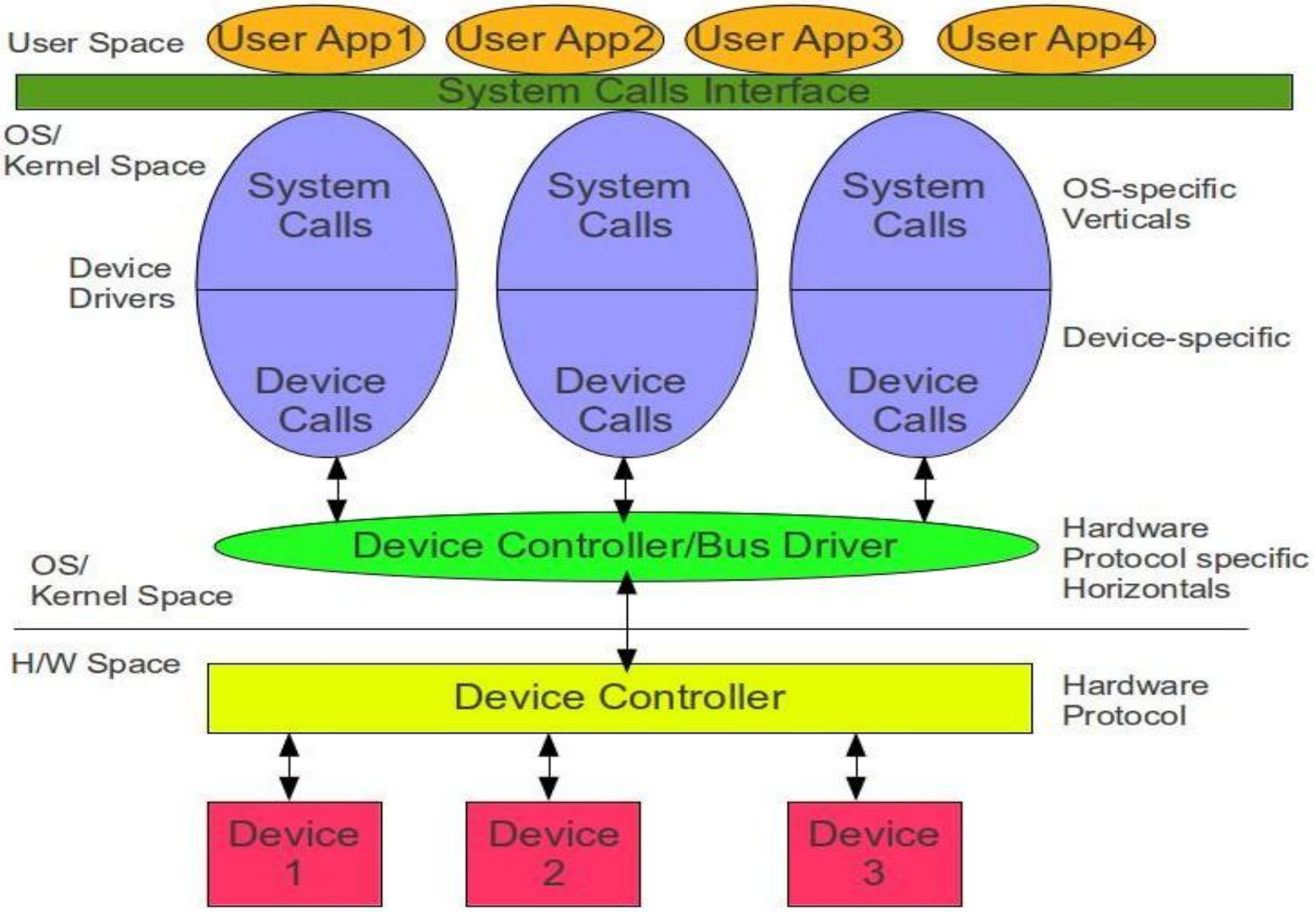
- The operating system preserves the state of the CPU by storing registers and the program counter

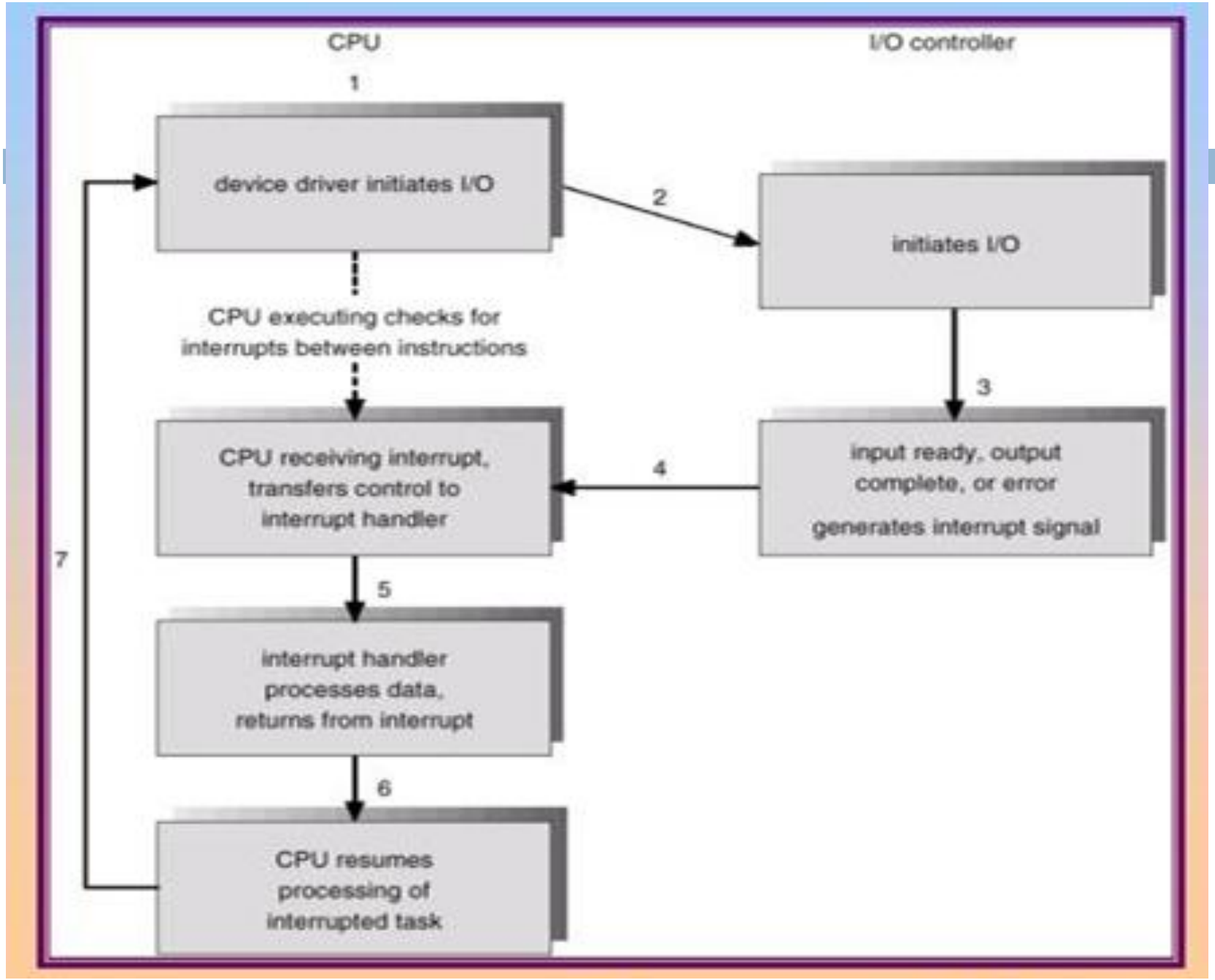
- Determines which type of interrupt has occurred:
  - **polling**
  - **vectored** interrupt system

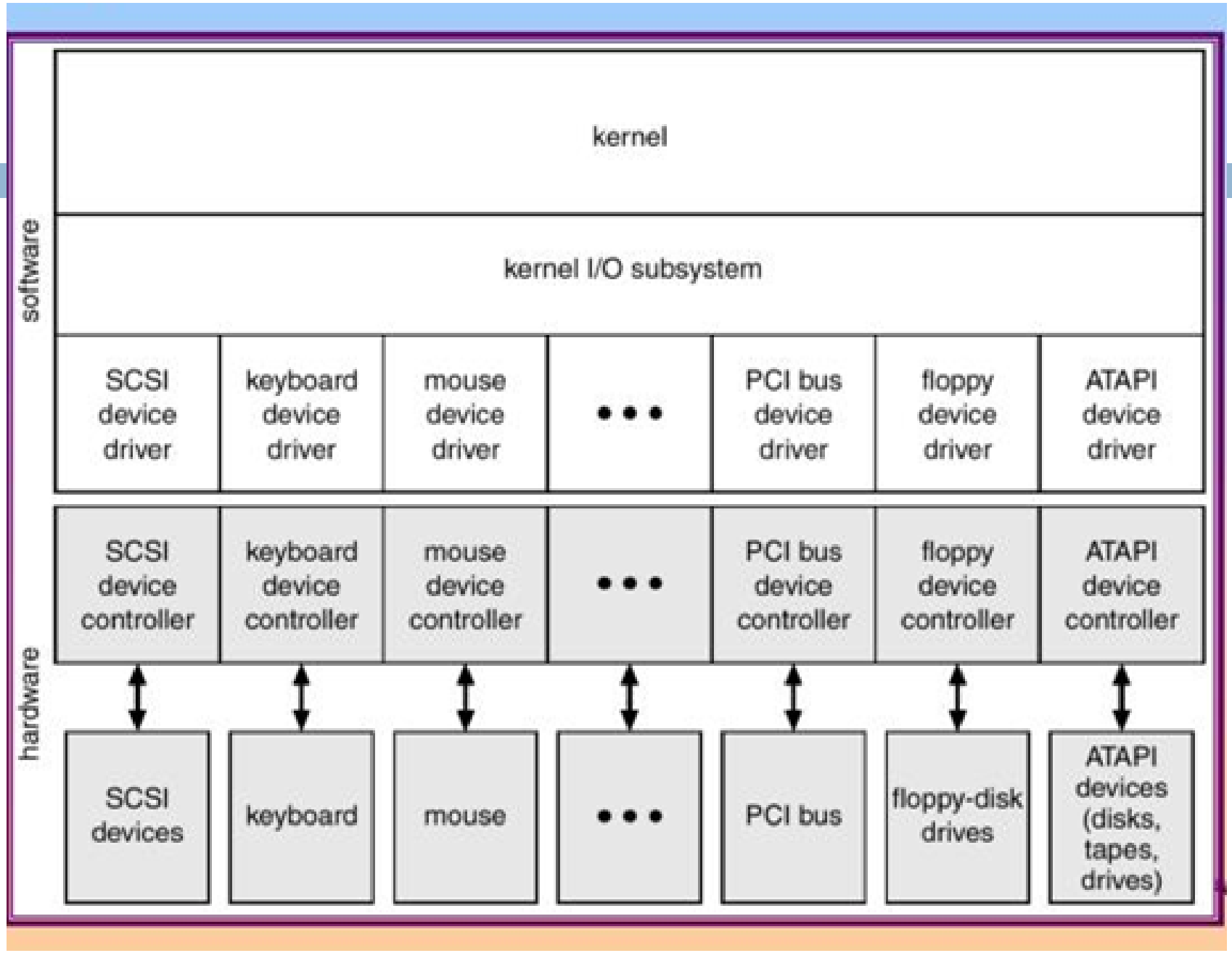
- Separate segments of code determine what action should be taken for each type of interrupt

Interrupt Timeline




User Space
User App1
User App2
User App3
User App4
System Calls Interface
OS/
Kernel Space
System Calls
System Calls
System Calls
OS-specific
Verticals
Device
Drivers
Device Calls
Device Calls
Device Calls
Device-specific
Device Controller/Bus Driver
Hardware
Protocol specific
Horizontals
OS/
Kernel Space
H/W Space
Device Controller
Hardware
Protocol
Device
1
Device
2
Device
3


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for
interrupts between instructions
3
CPU receiving interrupt,
transfers control to
interrupt handler
4
input ready, output
complete, or error
generates interrupt signal
7
5
interrupt handler
processes data,
returns from interrupt
6
CPU resumes
processing of
interrupted task


software
kernel
kernel I/O subsystem
SCSI device driver
keyboard device driver
mouse device driver
• • •
PCI bus device driver
floppy device driver
ATAPI device driver
hardware
SCSI device controller
keyboard device controller
mouse device controller
• • •
PCI bus device controller
floppy device controller
ATAPI device controller
SCSI devices
keyboard
mouse
• • •
PCI bus
floppy-disk drives
ATAPI devices (disks, tapes, drives)

→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.

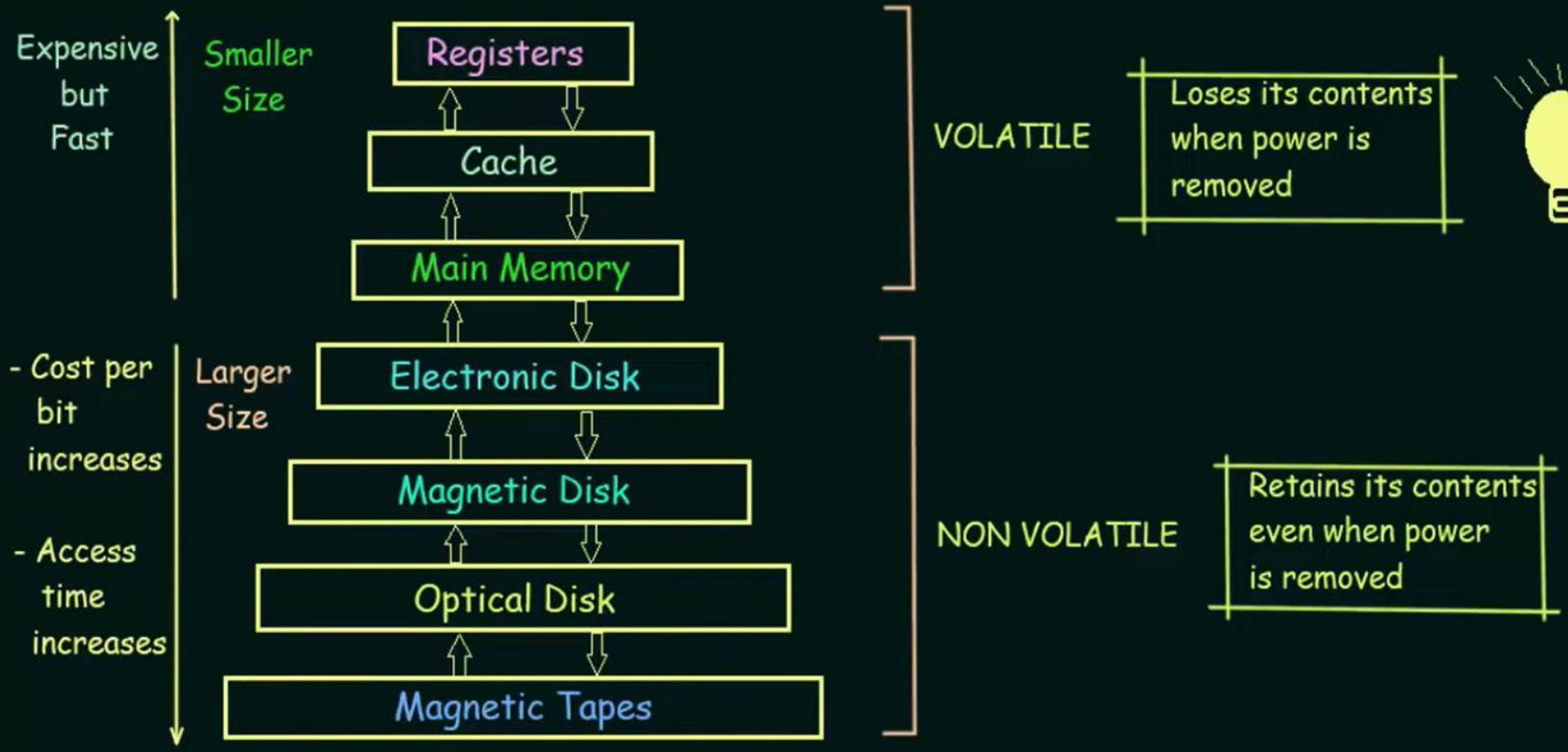
Byte Storage Notations

Metric	Value	Bytes
Byte (B)	1	1
Kilobyte (KB)	$1{,}024^1$	1,024
Megabyte (MB)	$1{,}024^2$	1,048,576
Gigabyte (GB)	$1{,}024^3$	1,073,741,824
Terabyte (TB)	$1{,}024^4$	1,099,511,627,776
Petabyte (PB)	$1{,}024^5$	1,125,899,906,842,624
Exabyte (EB)	$1{,}024^6$	1,152,921,504,606,846,976
Zettabyte (ZB)	$1{,}024^7$	1,180,591,620,717,411,303, 424
Yottabyte (YB)	$1{,}024^8$	1,208,925,819,614,629,174,706,176

Storage Structure

- Main memory – only large storage media that the CPU can access directly
  - **Random access**
  - Typically **volatile**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**
  - The **disk controller** determines the logical interaction between the device and the computer

Basics of Operating System (Storage Structure)



Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility

- **Caching** – copying information into faster storage system; main memory can be viewed as a cache for secondary storage

- **Device Driver** for each device controller to manage I/O
  - Provides uniform interface between controller and kernel

Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

Computer System Architecture

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

1. Single Processor Systems 

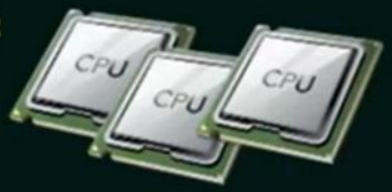
2. Multiprocessor Systems 

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

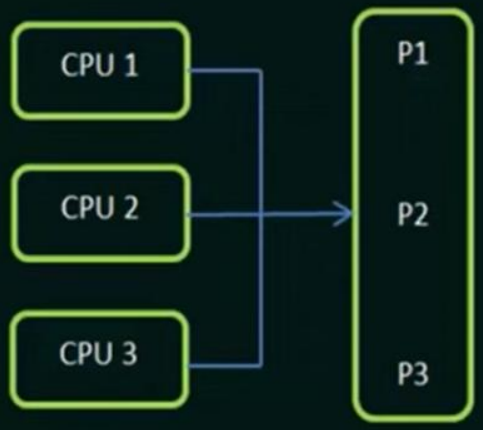
2. Multiprocessor Systems:



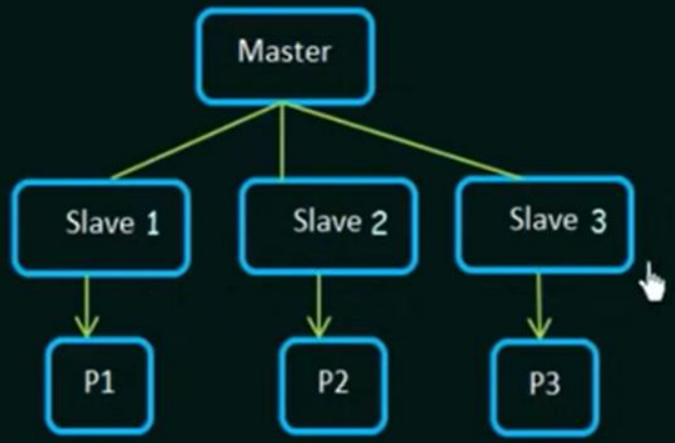
- Also known as parallel systems or tightly coupled systems.
- Has two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices

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

Operating System Structure

(Multiprogramming & Multitasking)

- Operating Systems vary greatly in their makeup internally
- COMMONALITIES:

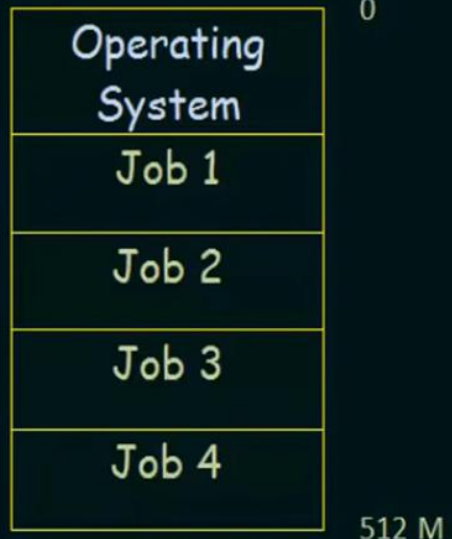
  (i) Multiprogramming

  (ii) Time Sharing (Multitasking)

(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.

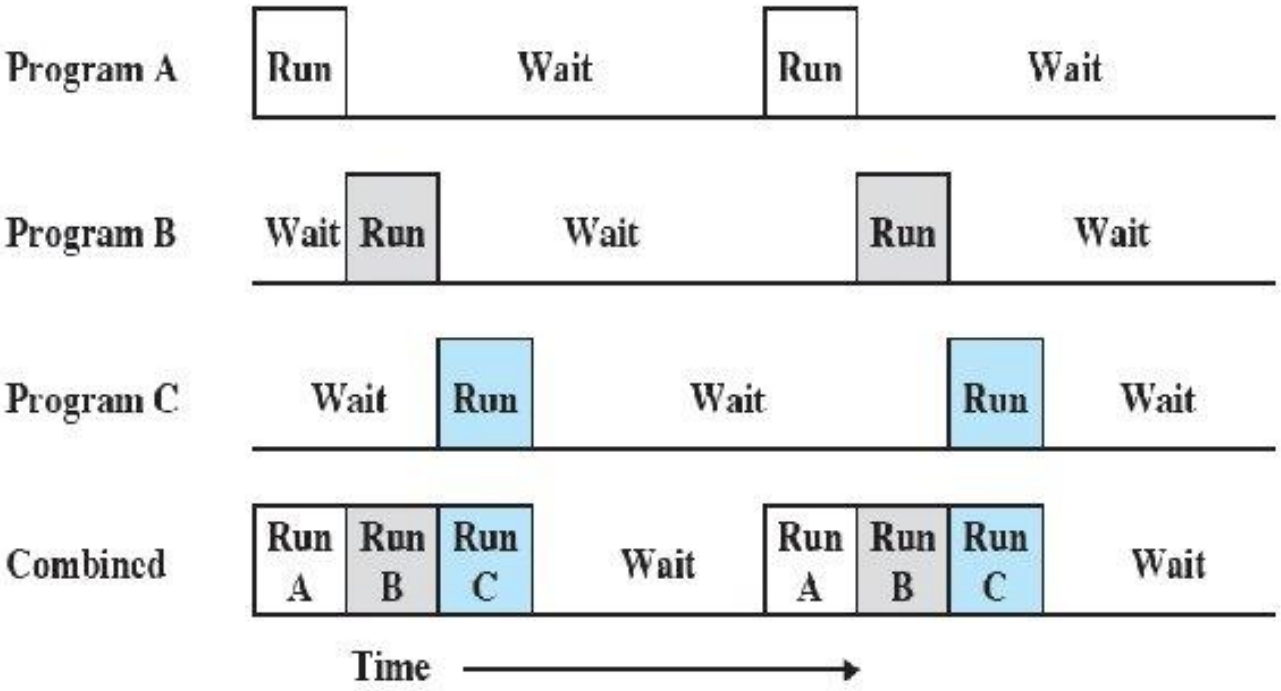
Job Pool



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.

Multiprogramming



(c) Multiprogramming with three programs

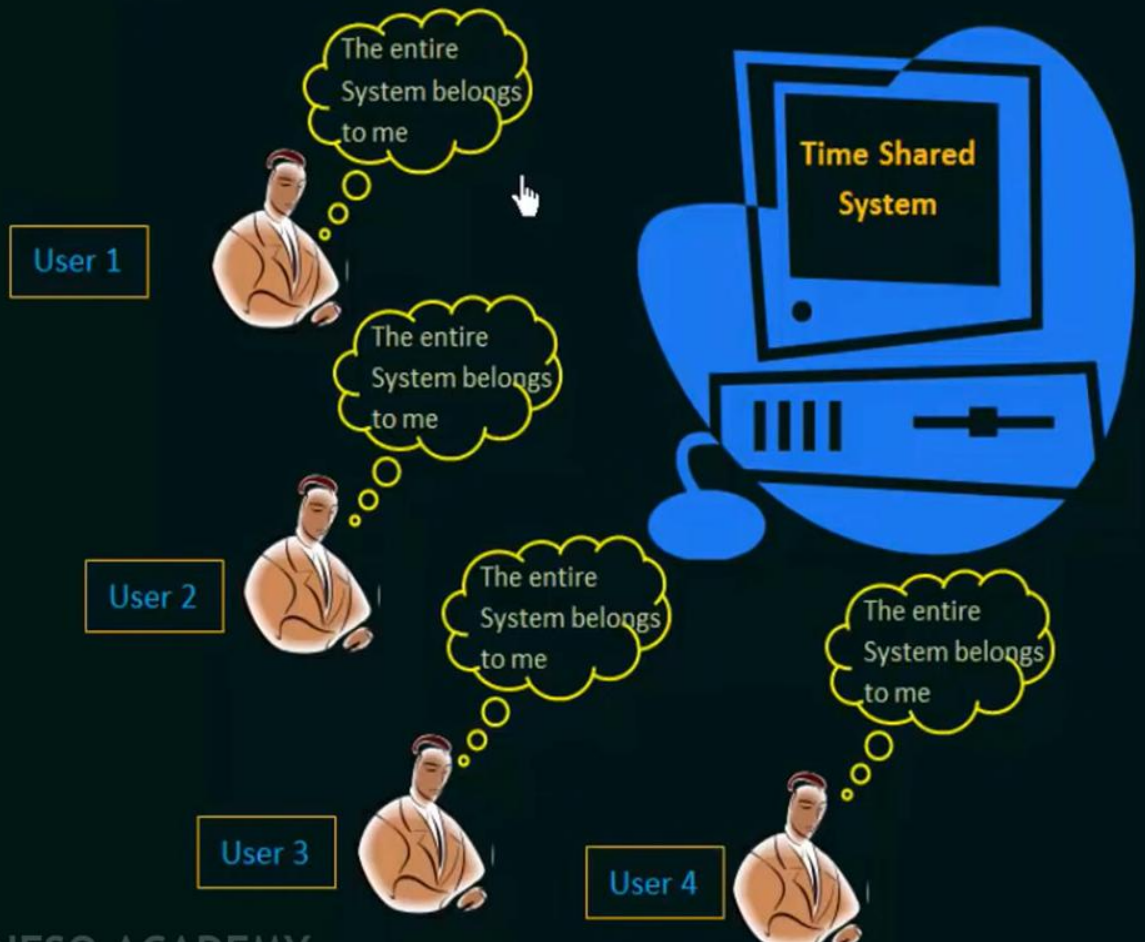
Multiprogramming

- **Multiprogramming** needed for efficiency
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job

(ii) Time Sharing (Multitasking)

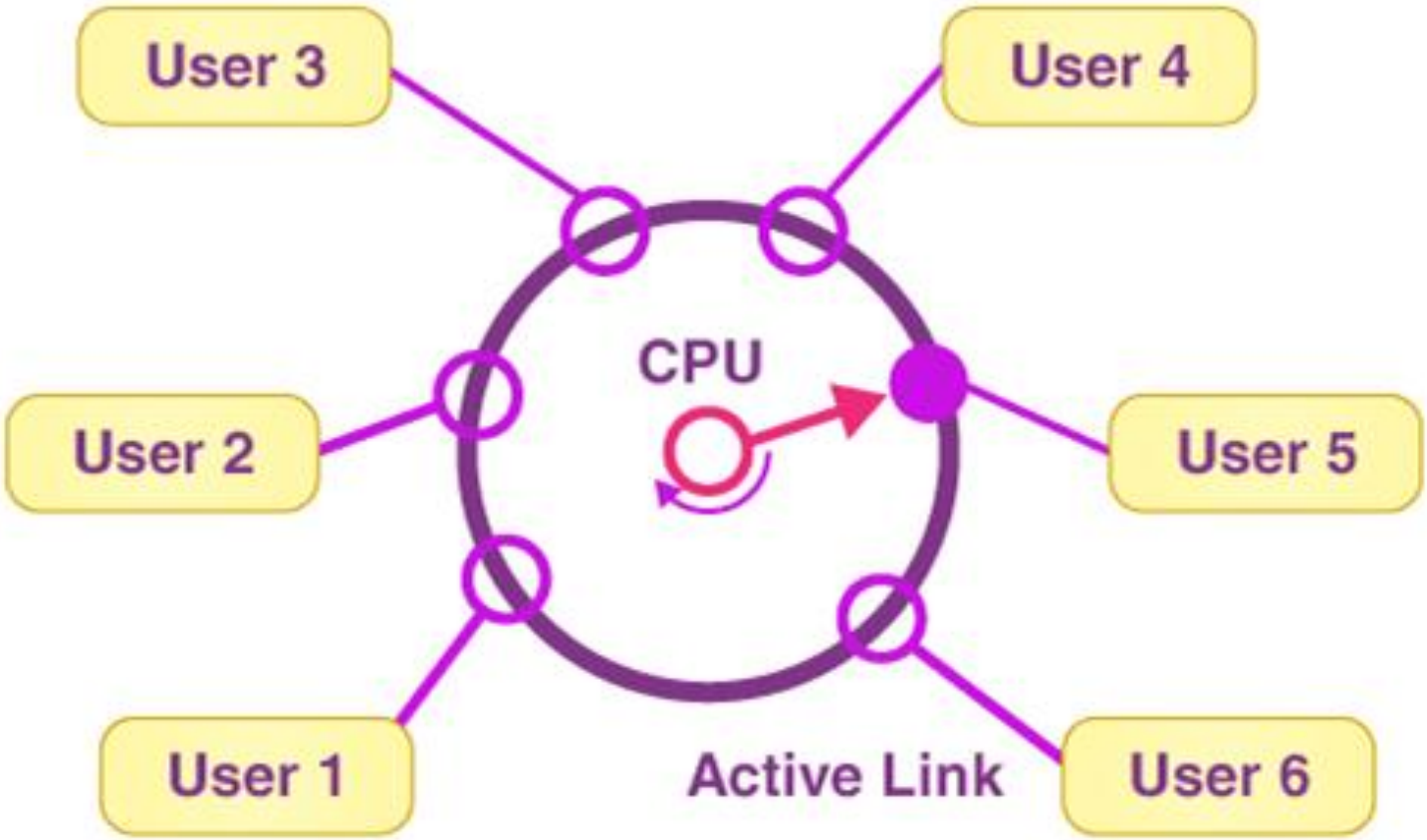
- CPU executes multiple jobs by switching among them
- Switches occur so frequently that the users can interact with each program while it is running
- Time sharing requires an interactive (or hands-on) computer system, which provides direct communication between the user and the system.
- A time-shared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

- 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"

Multitasking or Time-Sharing Operating System



Multitasking

- **Timesharing** (**multitasking**) is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory -**process**
  - If several jobs ready to run at the same time - **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

Operating-System Operations

- Modern operating systems are interrupt driven.
- When there is no process, no I/O routine, no user → CPU Utilization is Zero ie. System is in Idle State.
- A trap (or an exception) is a software-generated interrupt caused either by an error

- The interrupt-driven nature of an operating system defines that system's general structure.
- For each type of interrupt, separate segments of code in the operating system determine what action should be taken.
- An interrupt service routine is provided to deal with the interrupt.

- User request the OS to perform some services/Operations.
- Two Modes of Operations :
  - Dual Mode Operations
  - Timer based Operations

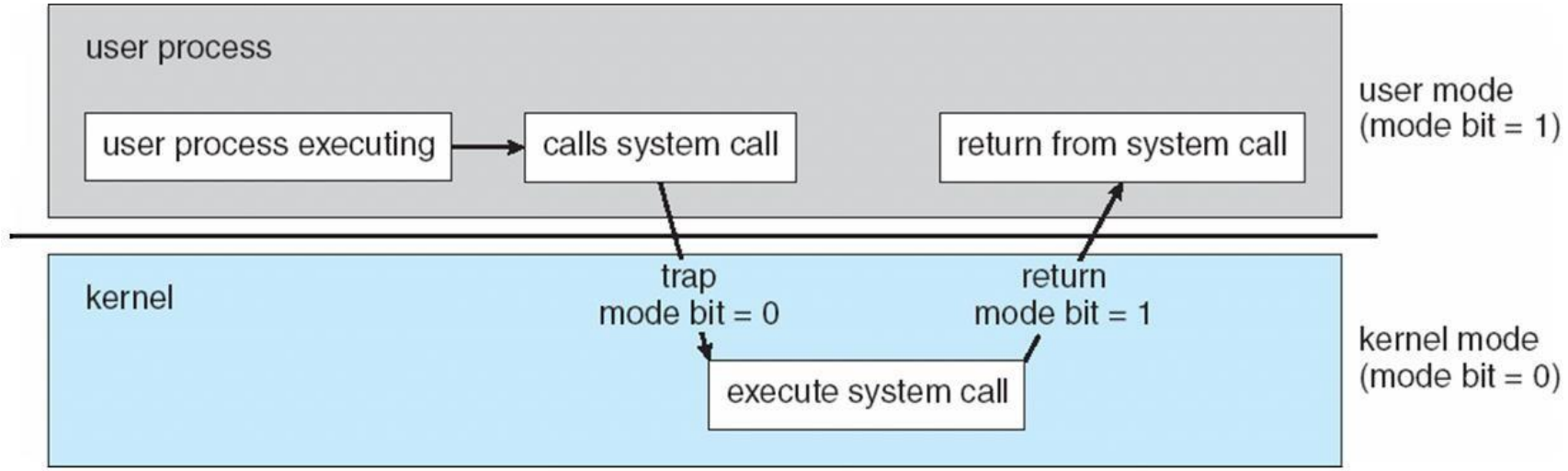
DUAL-MODE AND MULTIMODE OPERATION

- In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user defined code.
- Approach taken by most computer systems is to provide hardware support that allows us to differentiate among various modes of execution

- Two separate modes of operation: user mode and kernel mode (also called supervisor mode, system mode, or privileged mode).
- A bit, called the mode bit, is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1).
- With the mode bit, we can distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user.

Transition from User to Kernel Mode

- When user switched on the system, it is in Kernel mode later user mode.

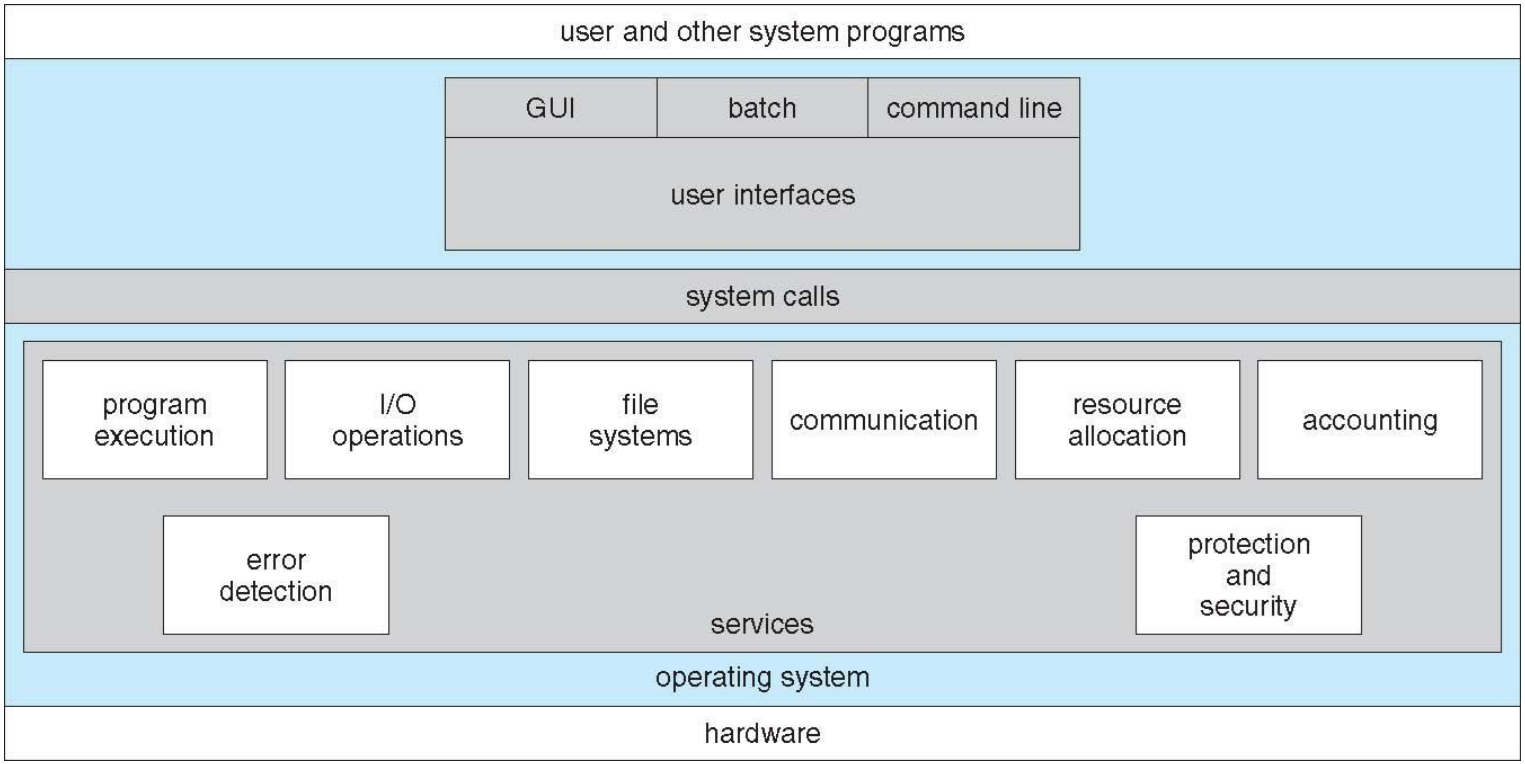


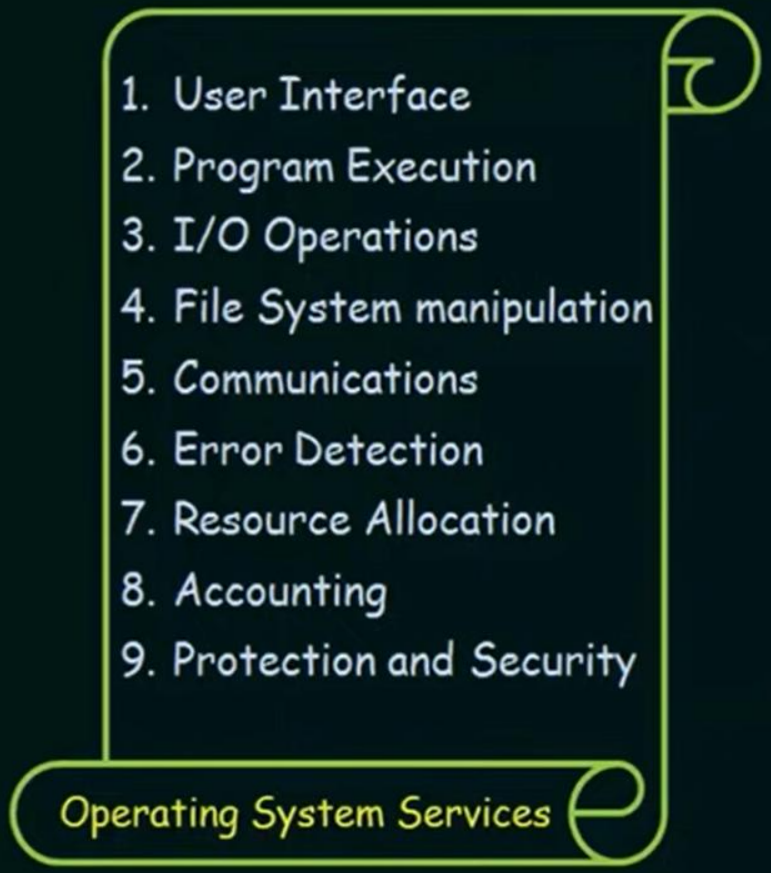
Timer

- Timer to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time

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


user and other system programs
GUI
batch
command line
user interfaces
system calls
program execution
I/O operations
file systems
communication
resource allocation
accounting
error detection
protection and security
services
operating system
hardware



1) User Interface

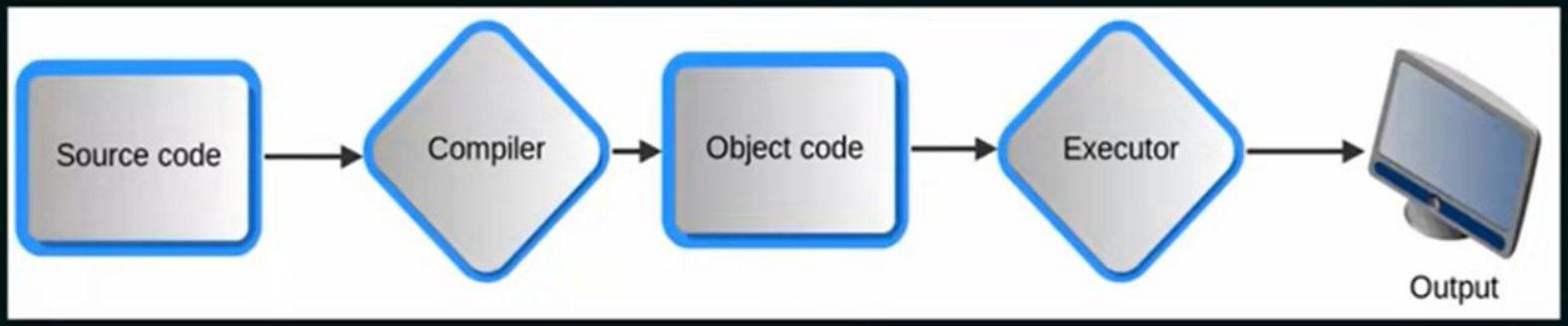
Command Line Interface (CLI)

Graphical User Interface (GUI)

User Interface

- Most operating systems have a user interface (UI).
- Command-line (CLI), Graphics User Interface (GUI), or batch

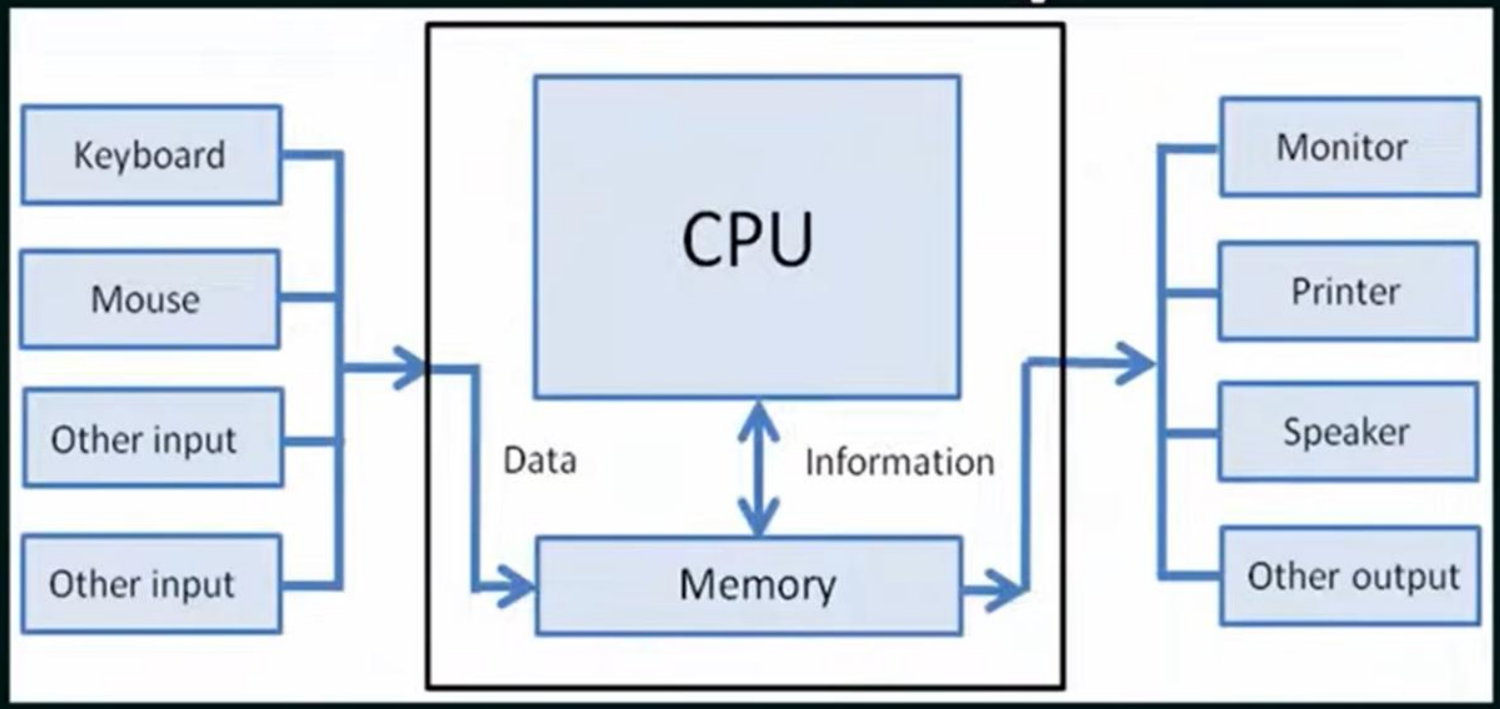
2) Program Execution



Program Execution

- Load and execute an program in the memory
- End execution, either normally or abnormally

3) I/O Operations



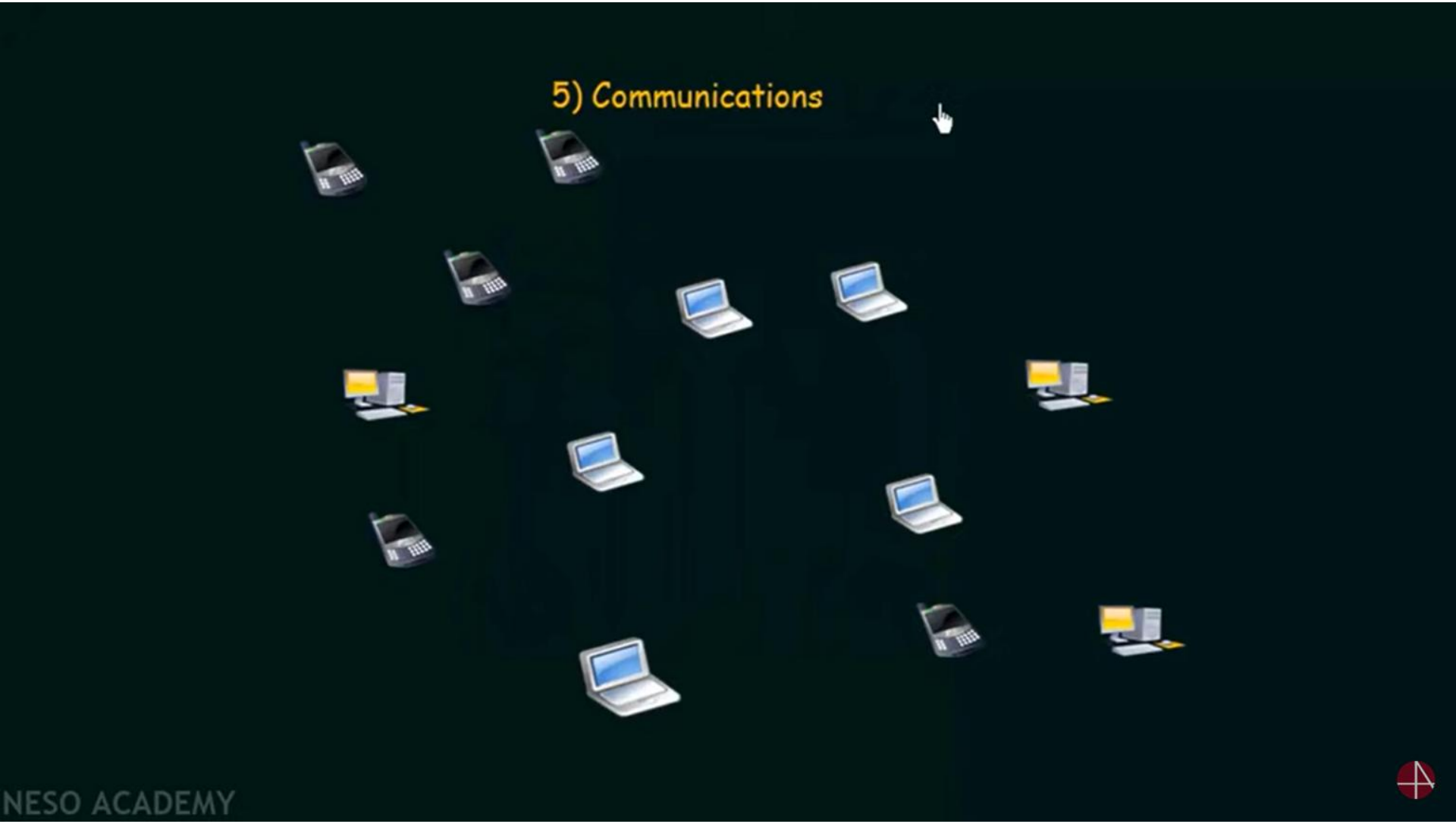
I/O Operations

- Running program may require I/O such as file or I/O device

4) File System Manipulation

File System Manipulation

- Read, write, create and delete files and directories
- Search or list files and directories
- Permission management


5) Communications
NESO ACADEMY

Communication

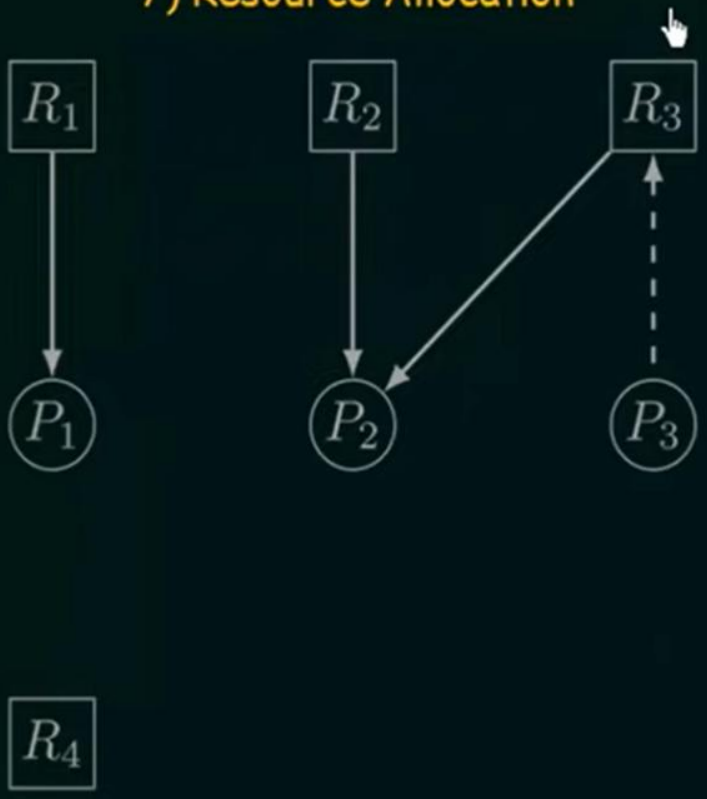
- Processes exchange information, on the same system or over a network
- via shared memory or through message passing

6) Error detection

Error Detection

- OS needs to be constantly aware of possible errors
- Errors in CPU, memory, I/O devices, programs
- It should take appropriate actions to ensure correctness and consistency

7) Resource Allocation



Resource Allocation

- Allocate resources for multiple users or multiple jobs running concurrently
- Many types of resources: CPU, memory, file, I/O devices

8) Accounting

Accounting

- To keep track of which users use how much and what kinds of resources

9) Protection and Security

Protection and Security

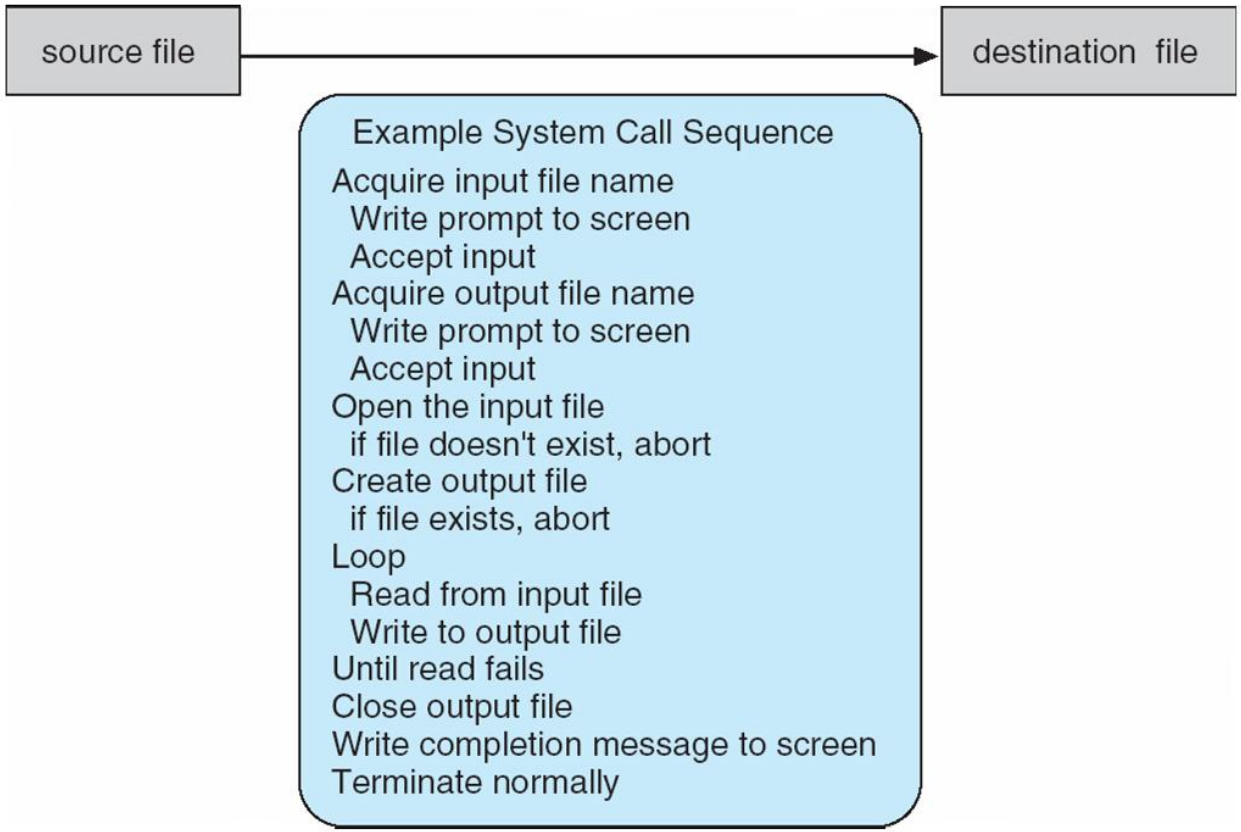
- Protection provides a mechanism to control access to system resources
  - access control: control access to resources
  - isolation: processes should not interfere with each other
- Security authenticates users and prevent invalid access to I/O devices
- Protection is the mechanism, security towards the policy

System Calls

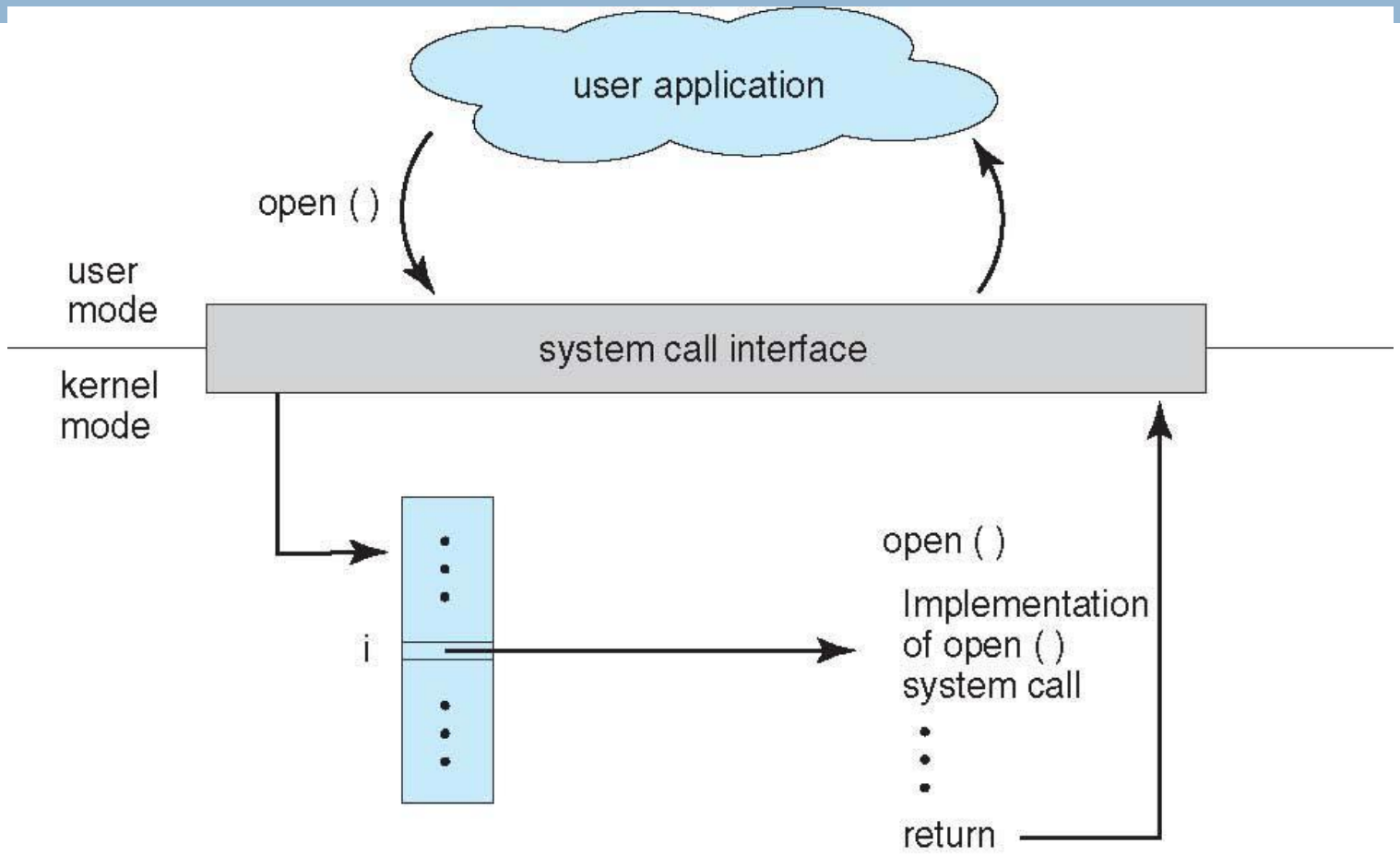
- Acts as an interface between the Kernel and User programs.
- It provides services to the user program through OS via API.
- Typically written in a high-level language (C or C++)
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

Example of System Calls

- System call sequence to copy the contents of one file to another file



API – System Call – OS Relationship



Types of System Calls

- Process control
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - **Debugger** for determining **bugs, single step** execution
  - **Locks** for managing access to shared data between processes

Types of System Calls

- File management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

Types of System Calls (Cont.)

- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages if **message passing model** to **host name** or **process name**
    - From **client** to **server**
  - **Shared-memory model** create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices

Types of System Calls (Cont.)

- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

Process Concept

- An operating system executes a variety of programs:
  - Batch system – **jobs**
  - Time-shared systems – **user programs** or **tasks**
- Terms ***job*** and ***process*** almost interchangeably

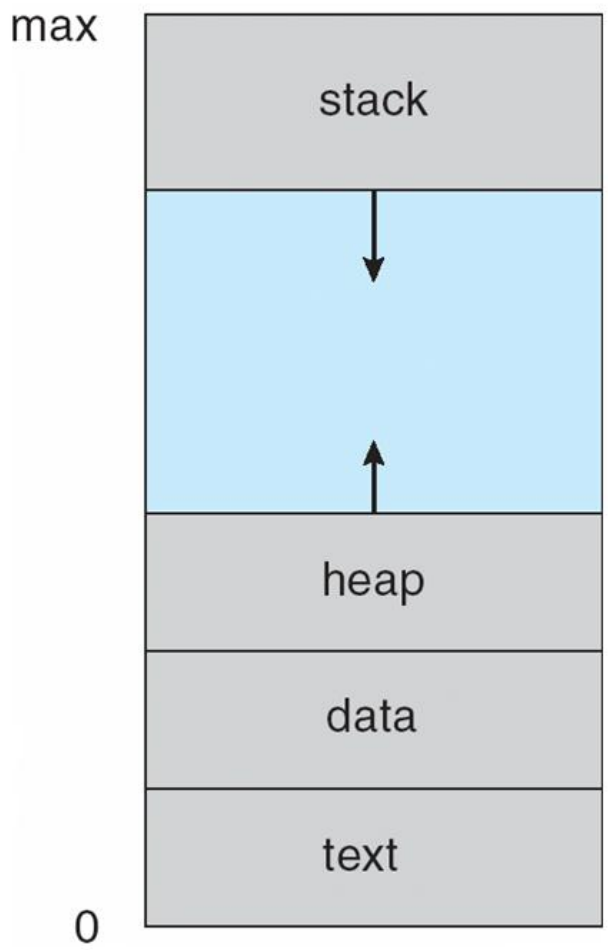
Process Concept

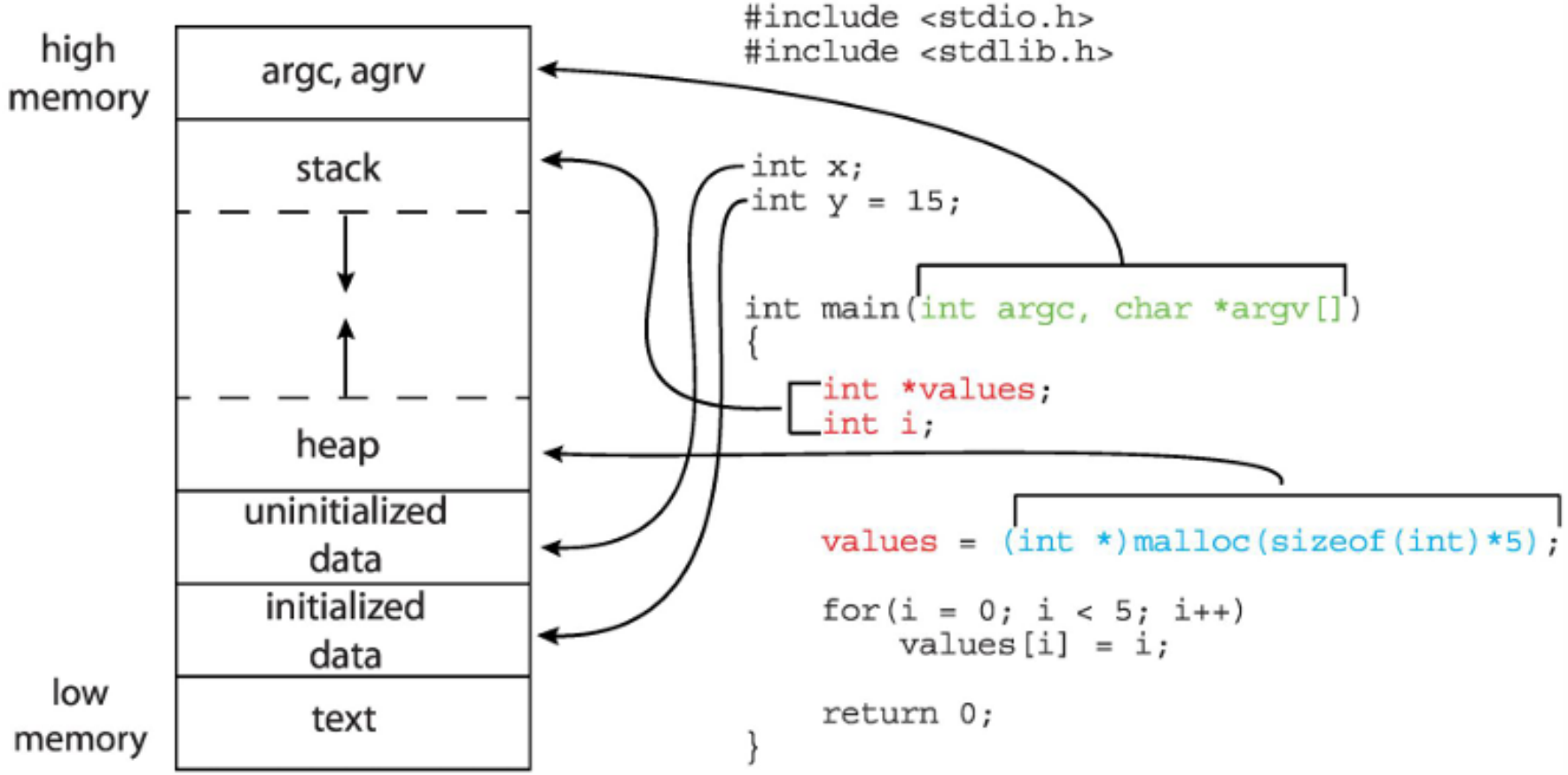
- **Process** – a program in execution; process execution must progress in sequential fashion
- Multiple parts
  - The program code, also called **text section**
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

Process Concept (Cont.)

- Program is ***passive/Static*** entity stored on disk (**executable file**), process is ***active /Dynamic***
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
  - Consider multiple users executing the same program

Process in Memory

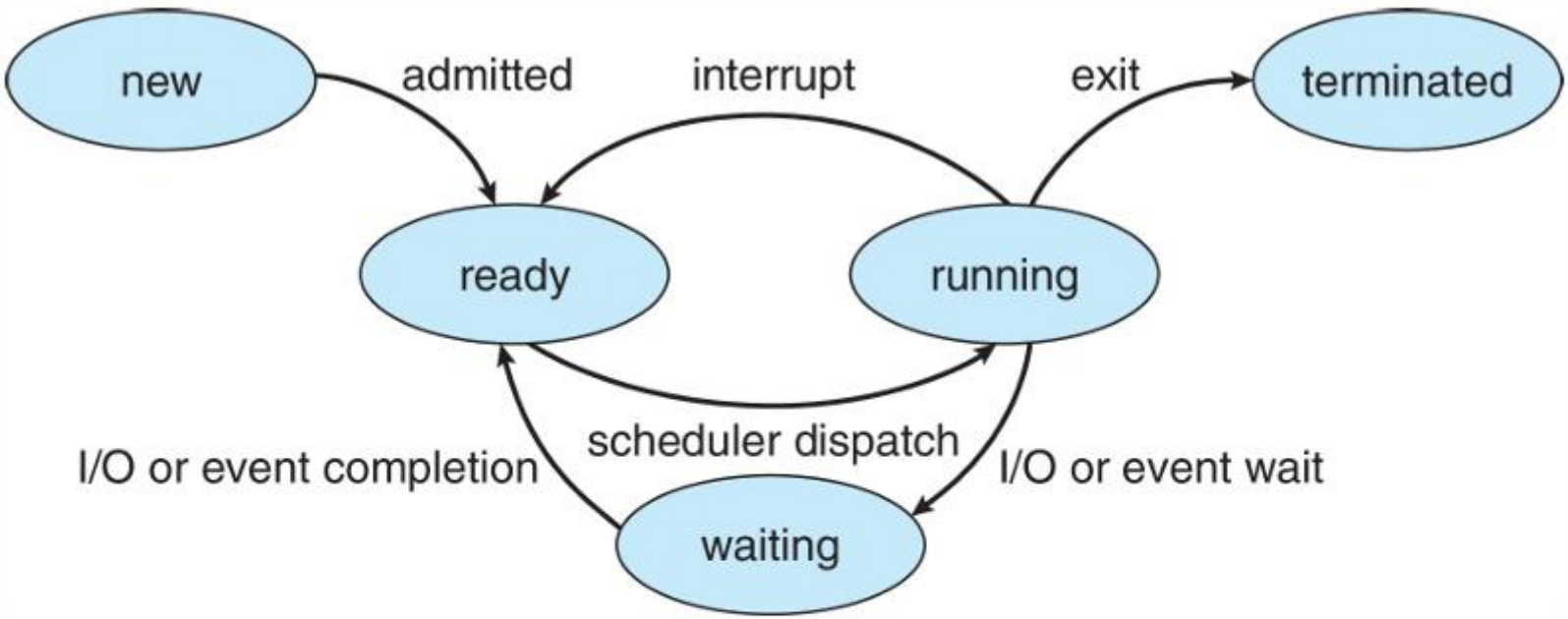



high memory
argc, agrv
stack
heap
uninitialized data
initialized data
text
low memory
#include <stdio.h>
#include <stdlib.h>
int x;
int y = 15;
int main(int argc, char *argv[])
{
int *values;
int i;
values = (int *)malloc(sizeof(int)*5);
for(i = 0; i < 5; i++)
values[i] = i;
return 0;
}

Process State

- As a process executes, it changes **state**
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

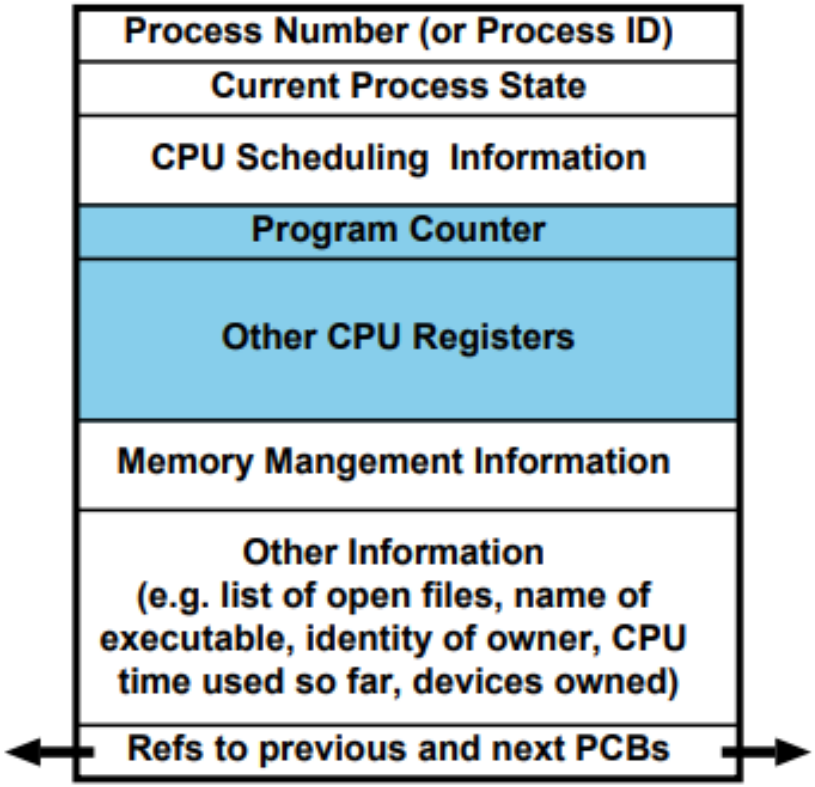
Diagram of Process State



New: The process is being created
Running: Instructions are being executed
Waiting: The process is waiting for some event to occur

Ready: The process is waiting to be assigned to a processor
Terminated: The process has finished execution

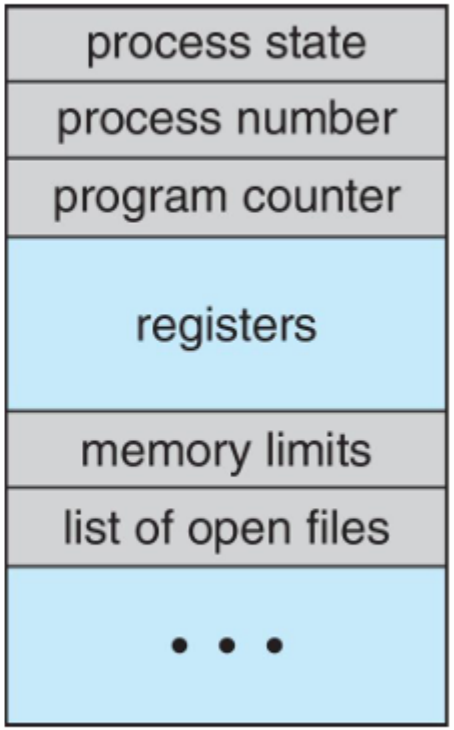
- OS maintains information about every process in a data structure called a Process Control Block (PCB).



Process Control Block (PCB)

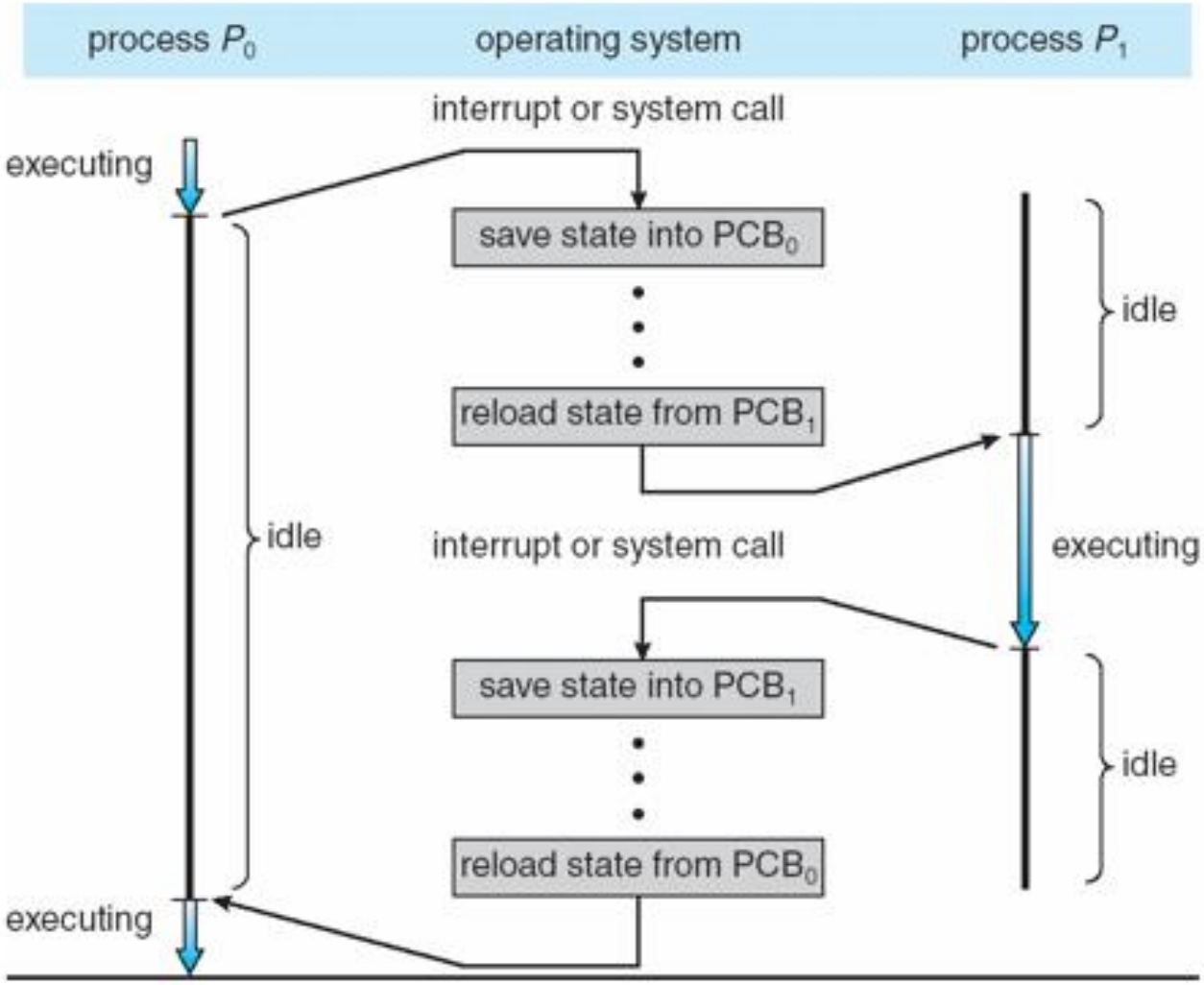
Information associated with each process(also called **task control block**)

- **Process state** – running, waiting, etc.
- **Program counter** – location of instruction to next execute
- **CPU registers** – contents of all process-centric registers
- **CPU scheduling information-** priorities, scheduling queue pointers
- **Memory-management information** – memory allocated to the process
- **Accounting information** – CPU used, clock time elapsed since start, time limits
- **I/O status information** – I/O devices allocated to process, list of open files

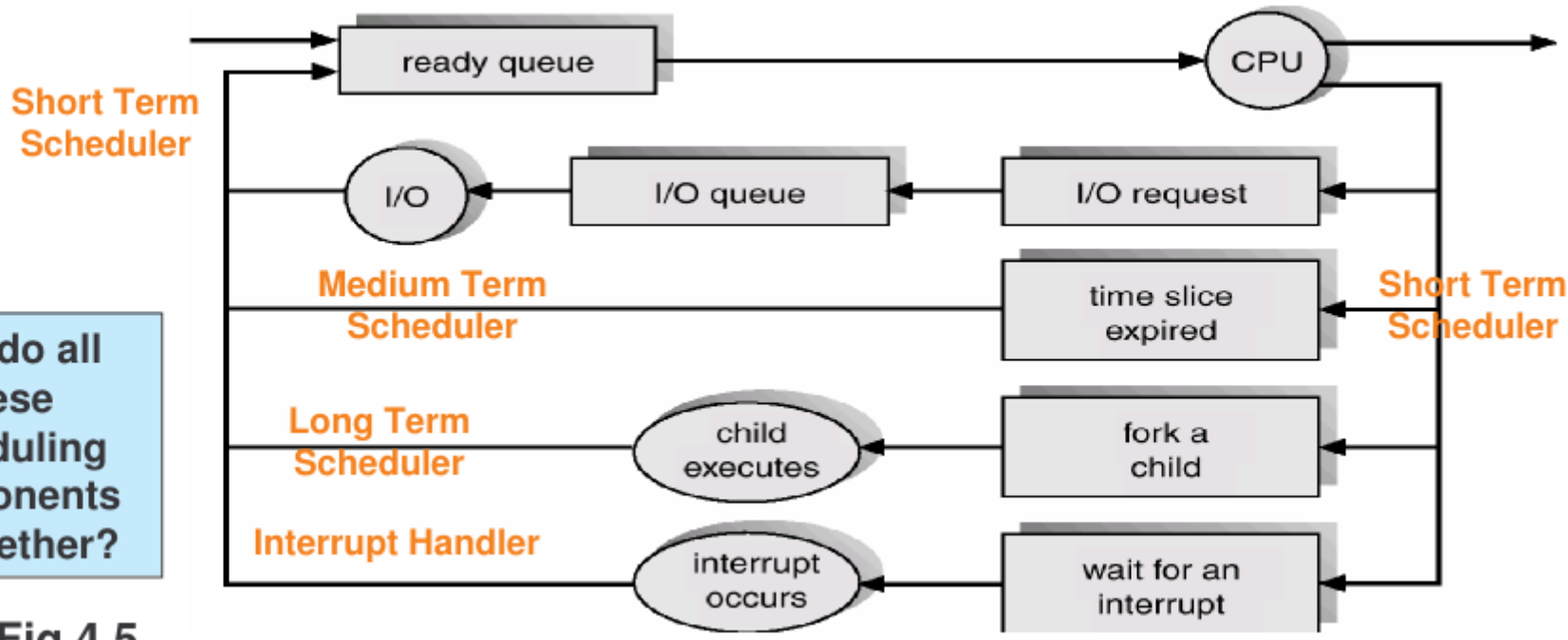


Context Switching

- Process Context = machine environment during the time the process is actively using the CPU.
- Context includes program counter, general purpose registers, processor status register.
- To switch between processes, the OS must:
  a) save the context of the currently executing process (if any),
  b) restore the context of that being resumed
- Time taken depends on h/w support.


process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
idle
reload state from PCB_1
idle
interrupt or system call
executing
save state into PCB_1
idle
reload state from PCB_0
executing

INTERRUPT HANDLER



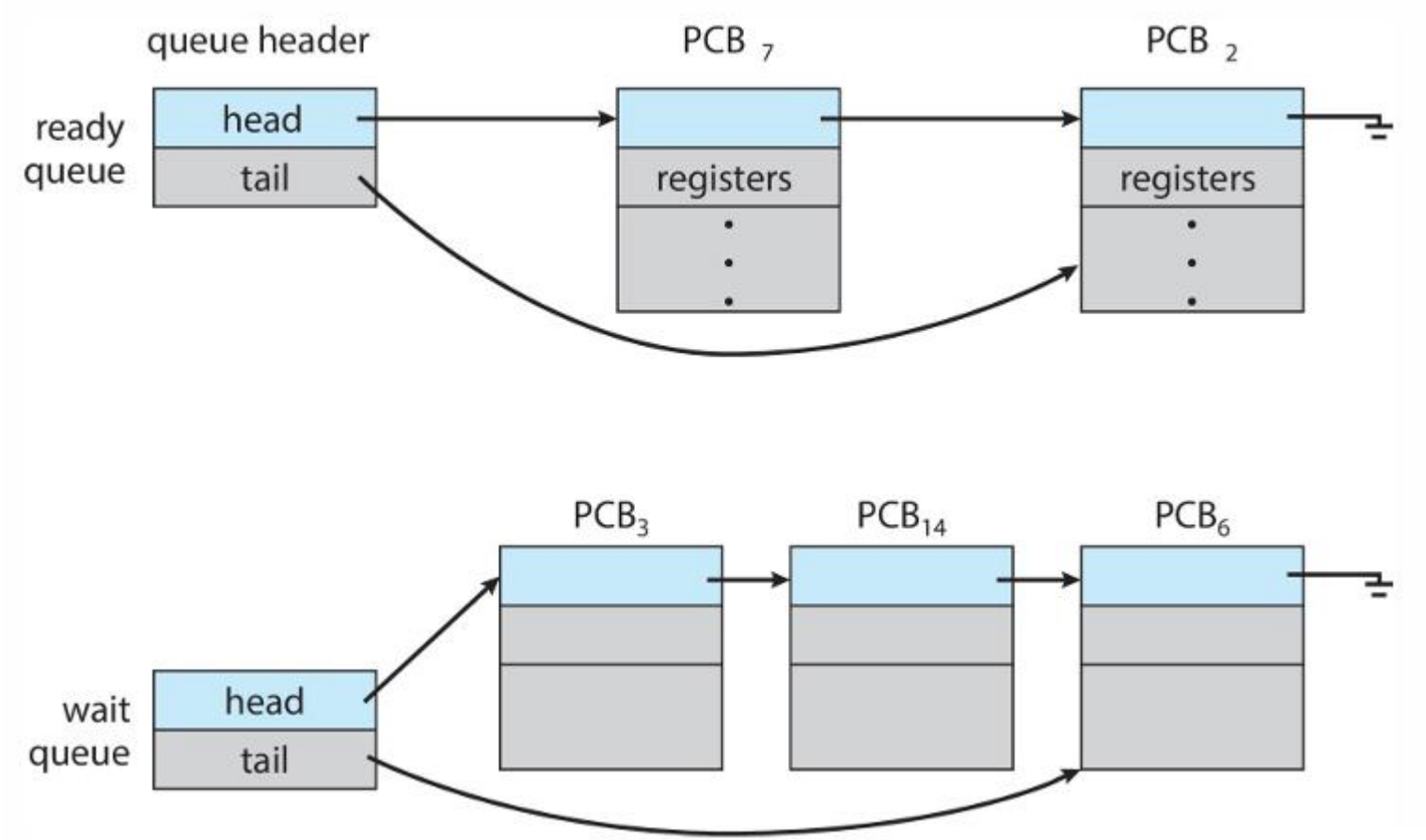
How do all these scheduling components fit together?

Fig 4.5

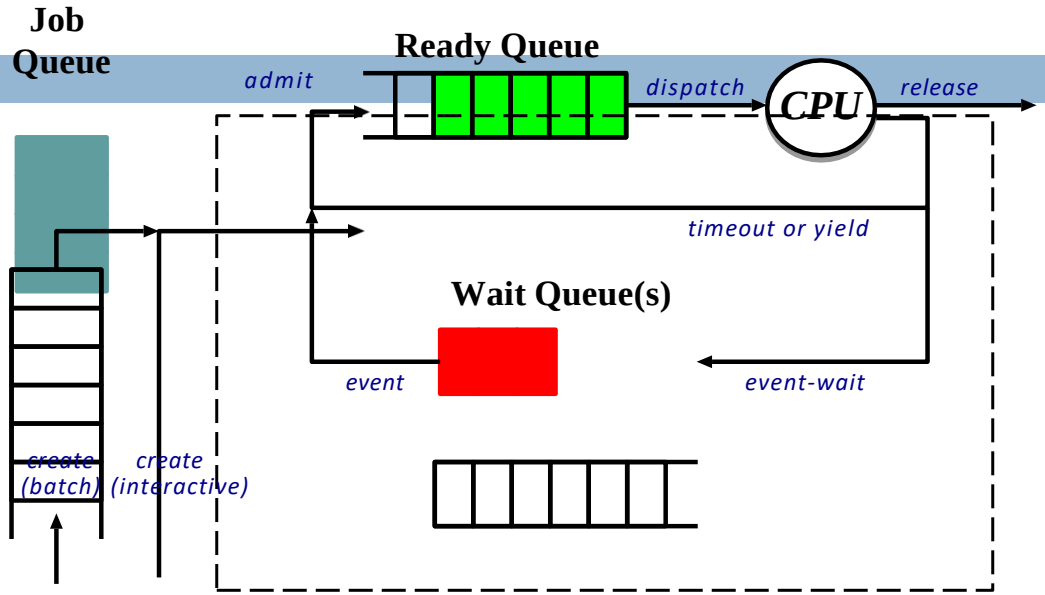
Process Scheduling

- **Process scheduler** selects among available processes for next execution on CPU core
- Goal -- Maximize CPU use, quickly switch processes onto CPU core
- Maintains **scheduling queues** of processes
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Wait queues** – set of processes waiting for an event (i.e., I/O)
  - Processes migrate among the various queues

Ready Queue and Wait Queue



Scheduling Queues



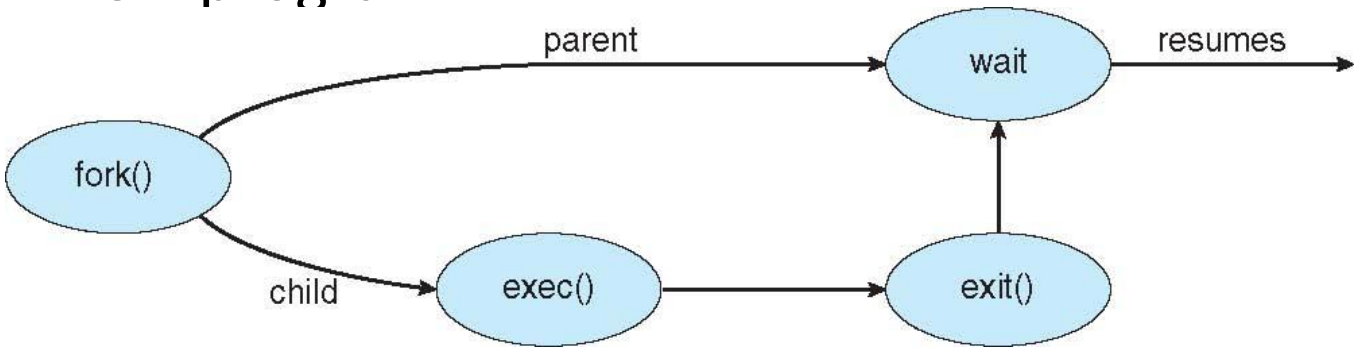
- Job Queue: batch processes awaiting admission.
- Ready Queue: set of all processes residing in main memory, ready to execute.
- Wait Queue(s): set of processes waiting for an I/O device (or for other processes)
- Long-term & short-term schedulers:
  - Job scheduler selects which processes should be brought into the ready queue.
  - CPU scheduler decides which process should be executed next and allocates the CPU to it.

Process Creation

- Nearly all systems are *hierarchical*: parent processes create children processes.
- Resource sharing:
  - parent and children share all resources, or
  - children share subset of parent's resources, or
  - parent and child share no resources.
- Execution:
  - parent and children execute concurrently, or
  - parent waits until children terminate.
- Address space:
  - child is duplicate of parent or
  - child has a program loaded into it.
- e.g. on Unix: `fork()` system call creates a new process
  - all resources shared (i.e. child is a clone).
  - `execve()` system call used to replace process' memory with a new program.

Process Creation (Cont.)

- UNIX examples
  - **`fork()`** system call creates new process
  - **`exec()`** system call used after a **`fork()`** to replace the process' memory space with a new program



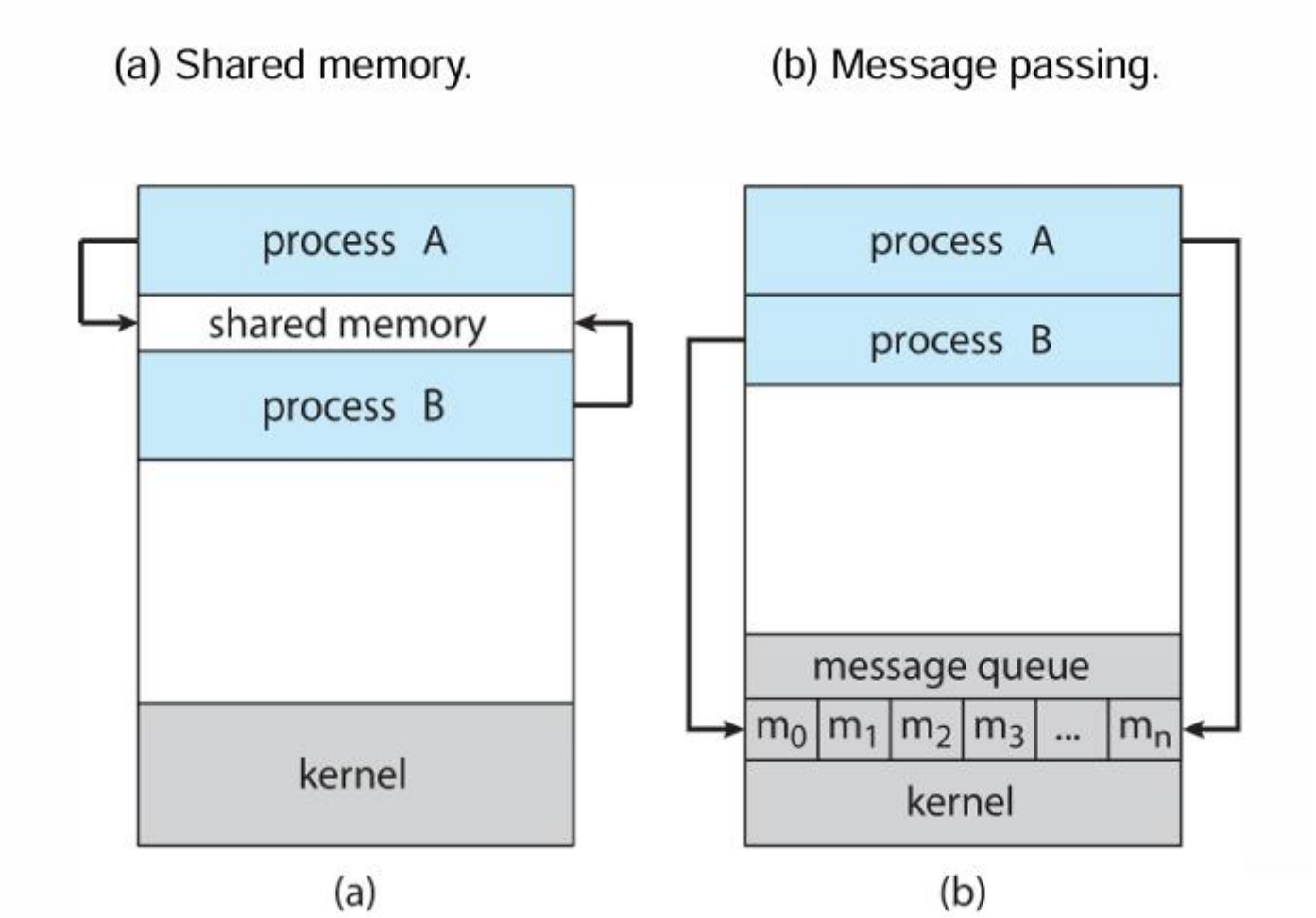
Process Termination

- Process executes last statement and asks the operating system to delete it (exit):
  - output data from child to parent (wait)
  - process' resources are deallocated by the OS.
- Process performs an illegal operation, e.g.
  - makes an attempt to access memory to which it is not authorised,
  - attempts to execute a privileged instruction
- Parent may terminate execution of child processes (abort, kill), e.g. because
  - child has exceeded allocated resources
  - task assigned to child is no longer required
  - parent is exiting ("cascading termination")
  - (many operating systems do not allow a child to continue if its parent terminates)
- e.g. Unix has `wait()`, `exit()` and `kill()`

Inter process Communication

- Processes within a system may be *independent* or *cooperating*
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **interprocess communication** (**IPC**)
- Two models of IPC
  - **Shared memory**
  - **Message passing**

Communications Models



Example of Cooperative processes:

- Producer-Consumer problem.
- Unbounded buffer or bounded buffer
- Buffer can be provided by OS (i.e., message-passing) e.g., pipe()
- Buffer can be programmed in the cooperative processes (i.e., shared memory)

Direct Communication

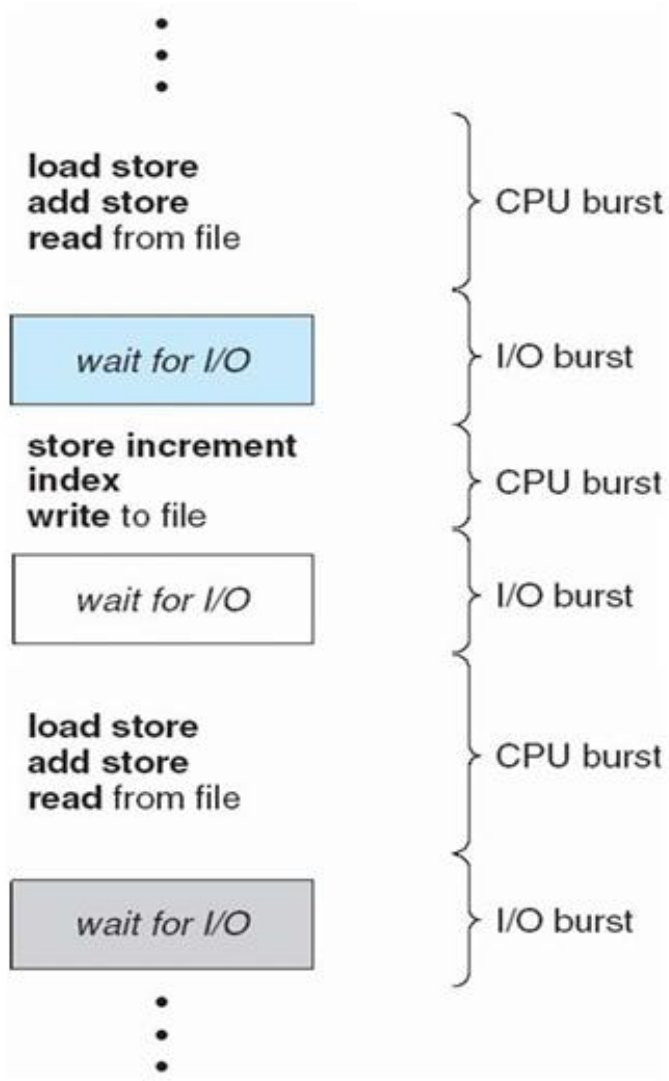
- Both sender and receiver must name each other. (Symmetric addressing)
  - Process Q: send (P, message)
  - Process P: receive (Q, message)
- Only the sender names the recipient. (Asymmetric addressing)
  - Process Q: send (P, message)
  - Process P: receive (id, message)
- Pros: provide some degree of protection
- Cons: Limited modularity (changing name of P)

Indirect Communication

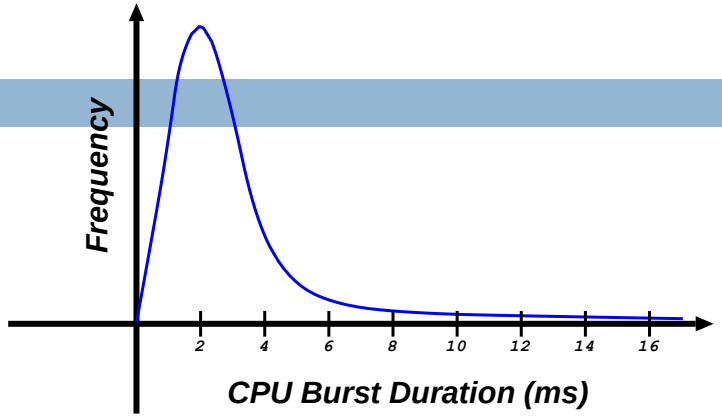
- Messages are sent and received to/from ports (or mailboxes)
- Each port has a unique ID
- Process P: send (Port ID, message)
- Process Q: receive (Port ID, message)
- Mailbox can belongs to OS or programmed in cooperative processes

CPU Scheduling

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait
- **CPU burst** distribution


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

CPU-I/O Burst Cycle



- CPU-I/O Burst Cycle: process execution consists of an on-going *cycle* of CPU execution, I/O wait, CPU execution, . . .
- Processes can be described as either:
  1. I/O-bound: spends more time doing I/O than computation; has many short CPU bursts.
  2. CPU-bound: spends more time doing computations; has few very long CPU bursts.
- Observe most processes execute for at most a few milliseconds before blocking

⇒ need multiprogramming to obtain decent overall CPU utilization.

CPU Scheduler

- Selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive**
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Types of CPU Scheduling

- First Come First Serve
- Shortest Job First-Non-Preemptive
- Shortest Job First-Preemptive
- Round Robin
- Priority Scheduling

https://www.youtube.com/watch?v=AiVKIdGheEU&list=PLIY8eNdw5tW_lHyageTADFKBt9weJXndE&index=1

CPU SCHEDULING ALGORITHM: FIRST COME FIRST SERVE (FCFS)

Q

PROCESS	Arrival Time (AT) (mSec)	Burst Time (BT)	Completion Time (CT)	Turnaround Time (TAT)	Waiting Time (WT)
P_0	0	5			
P_1	1	3			
P_2	2	8			
P_3	3	6			

Gnatt Chart:

P_0

CPU SCHEDULING ALGORITHM: FIRST COME FIRST SERVE (FCFS)

CT - AT TAT - BT

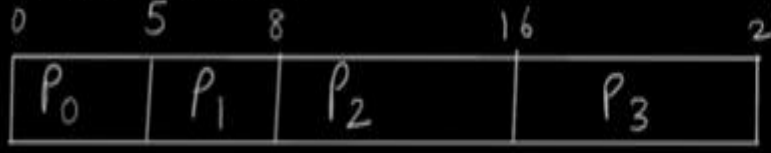
Q

PROCESS	Arrival Time (AT) (mSec)	Burst Time (BT)	Completion Time (CT)	Turnaround Time (TAT)	Waiting Time (WT)
P_0	0	5	5	5	0
P_1	1	3	8	7	4
P_2	2	8	16	14	6
P_3	3	6	22	19	

$t=0$ P_0

$t=5$ P_1, P_2, P_3

Gnatt Chart:



Process	Burst Time(ms)
P_1	5
P_2	24
P_3	16
P_4	10
P_5	3

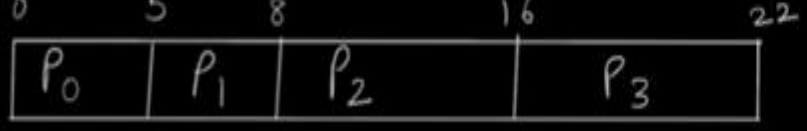
CPU SCHEDULING ALGORITHM: FIRST COME FIRST SERVE (FCFS)

CT - AT TAT - BT

Q

Process	Arrival Time (AT) (mSec)	Burst Time (BT)	Completion Time (CT)	Turnaround Time (TAT)	Waiting Time (WT)
P_0	0	5	5	5	0
P_1	1	3	8	7	4
P_2	2	8	16	14	6
P_3	3	6	22	19	13

Gnatt Chart:



$t=0$ P_0

$t=5$ P_1, P_2, P_3

Tot (TAT) = 5 + 7 + 14 + 19
= 45 mSec

Avg (TAT) = Tot (TAT) / 4
= 45/4 = 11.25

Tot (WT) = 0 + 4 + 6 + 13
= 23 mSec

Avg (WT) = 23/4 = 5.75 mSec

CPU SCHEDULING ALGORITHM : SHORTEST JOB FIRST [SJF] NON-PREEMPTIVE

Q

PROCESS	Arrival Time (AT) (mSec)	Burst Time (BT)	Completion Time (CT)	Turnaround Time (TAT)	Waiting Time (WT)
P_0	0	7			
P_1	2	4			
P_2	4	1			
P_3	5	4			

To Find :

Avg (TAT) =

Avg (WT) =

Gnatt Chart :

0

$t_0 = P_0$

Process Queue	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

CPU SCHEDULING ALGORITHM: SHORTEST JOB FIRST [SJF] NON-PREEMPTIVE

Q

PROCESS	Arrival Time (AT) (mSec)	Burst Time (BT)	Completion Time (CT)	Turnaround Time (TAT) (CT − AT)	Waiting Time (WT) (TAT − BT)
P_0	0	7	7	7	0
P_1	2	4	12	10	6
P_2	4	1	8	4	3
P_3	5	4	16	11	7
				32	16

To Find:

Avg (TAT) = Tot (TAT) / 4

= 32 / 4

= 8 m Sec

Avg (WT) = 16 / 4

= 4 m Sec

Gnatt Chart:

CPU SCHEDULING ALGORITHM: SHORTEST JOB FIRST [SJF] PREEMPTIVE [SRT]

CT - AT TAT - BT

Q

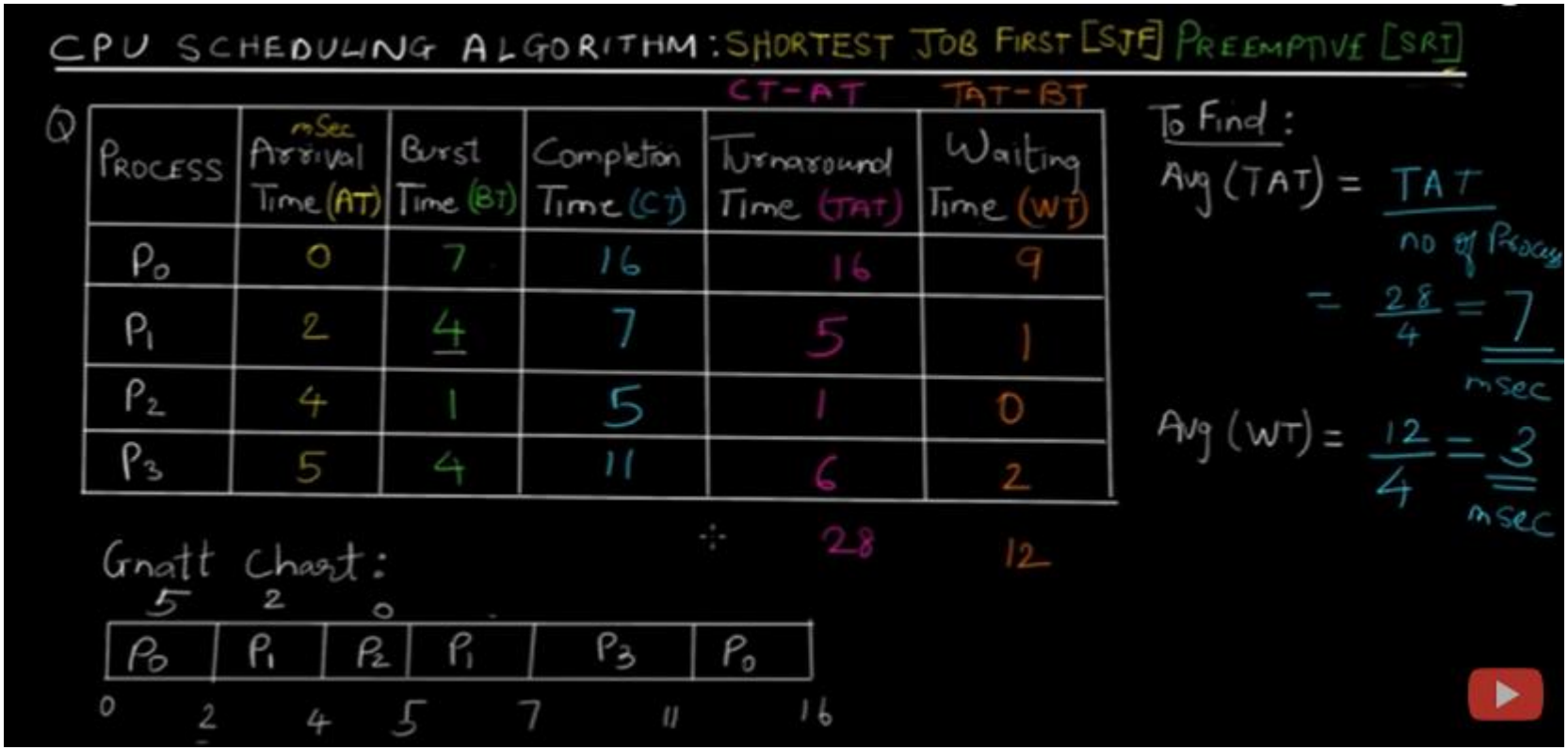
PROCESS	Arrival Time (AT) (mSec)	Burst Time (BT)	Completion Time (CT)	Turnaround Time (TAT)	Waiting Time (WT)
P_0	0	7			
P_1	2	4			
P_2	4	1			
P_3	5	4			

To Find:

Avg (TAT) =

Avg (WT) =

Gnatt Chart:


CPU SCHEDULING ALGORITHM : SHORTEST JOB FIRST [SJF] PREEMPTIVE [SRT]
CT-AT
TAT-BT
Q
Process
mSec
Arrival Time (AT)
Burst Time (BT)
Completion Time (CT)
Turnaround Time (TAT)
Waiting Time (WT)
P0 0 7 16 16 9
P1 2 4 7 5 1
P2 4 1 5 1 0
P3 5 4 11 6 2
∴ 28 12
To Find :
Avg (TAT) = TAT / no of Process = 28/4 = 7 msec
Avg (WT) = 12/4 = 3 msec
Gnatt Chart :
5 2 0
P0 P1 P2 P1 P3 P0
0 2 4 5 7 11 16

CPU SCHEDULING ALGORITHM: ROUND ROBIN [q = 3]

Q

PROCESS	Arrival Time (AT) (mSec)	Burst Time (BT)	Completion Time (CT)	Turnaround Time (TAT) (CT-AT)	Waiting Time (WT) (TAT-BT)
P_0	0	7			
P_1	2	4			
P_2	4	1			

To Find:

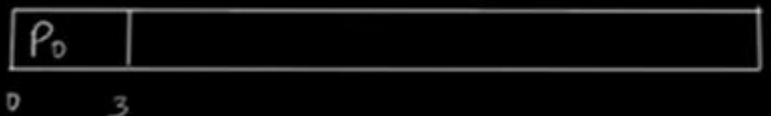
Avg (TAT) =

Avg (WT) =

quantum = 3msec

P_0 P_1 P_0

Gnatt Chart:



CPU SCHEDULING ALGORITHM: ROUND ROBIN [q = 3]

Q

Process	Arrival Time (AT) (mSec)	Burst Time (BT)	Completion Time (CT)	Turnaround Time (TAT) (CT−AT)	Waiting Time (WT) (TAT−BT)
P_0	0	7	12	12	5
P_1	2	4	11	9	5
P_2	4	1	10	6	5

quantum = 3msec

P_0 P_1 P_0 P_2 P_1 P_0 27 15

Gnatt Chart:

4 1 1 0 0 0

P_0	P_1	P_0	P_2	P_1	P_0

0 3 6 9 10 11 12

To Find:

Avg (TAT) = Tot TAT / no of Process

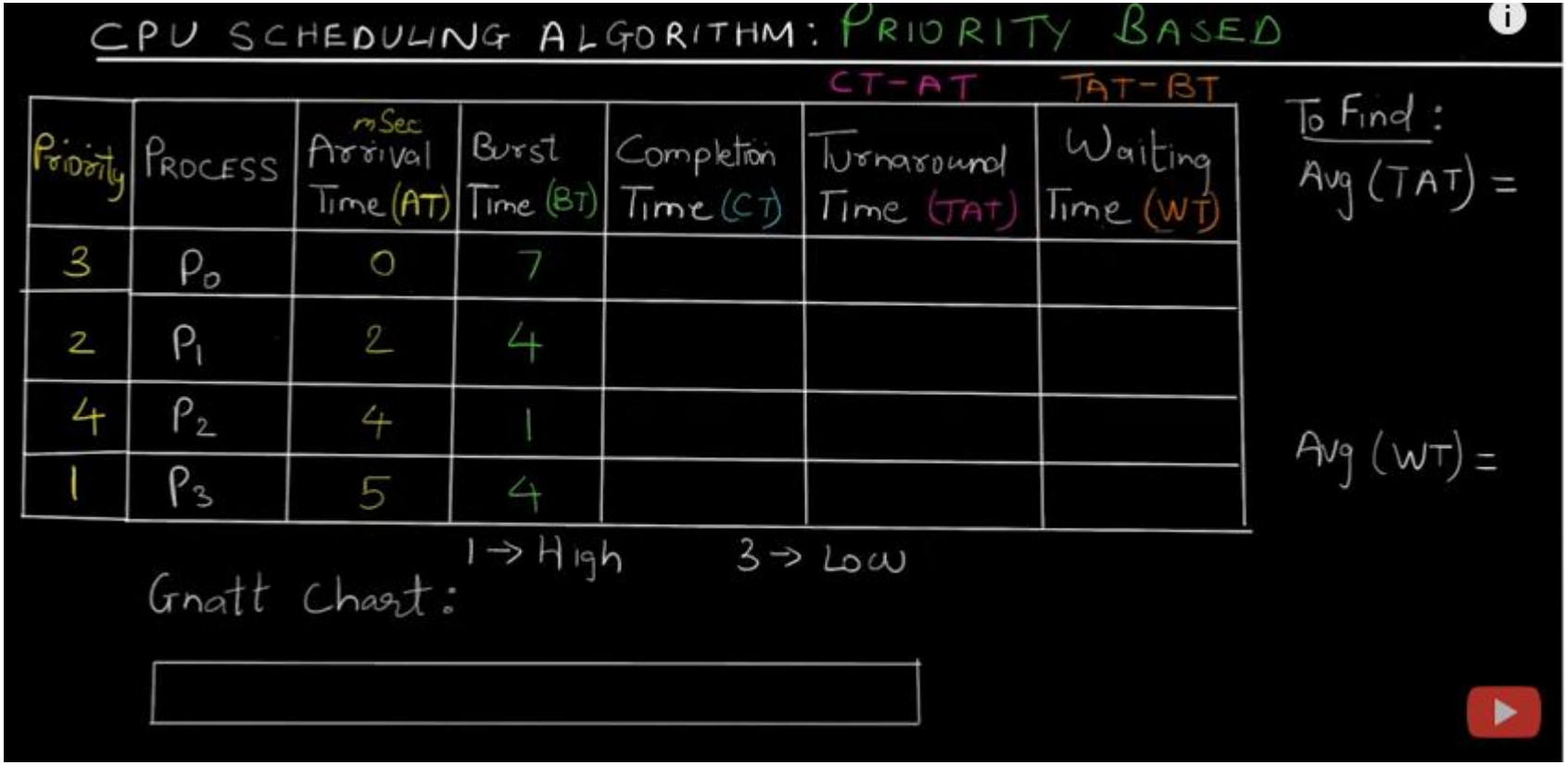
= 27/3 = 9 msec

Avg (WT) = Tot WT / no Process

= 15/3

= 5 msec

Priority Scheduling



CPU SCHEDULING ALGORITHM: PRIORITY BASED

Priority	Process	Arrival Time (AT) (mSec)	Burst Time (BT)	Completion Time (CT)	Turnaround Time (TAT) (CT-AT)	Waiting Time (WT) (TAT-BT)
3	P_0	0	7	7	7	0
2	P_1	2	4	15	13	9
4	P_2	4	1	16	12	11
1	P_3	5	4	11	6	2

1 → High 3 → Low

38

22

To Find:

Avg (TAT) = Tot TAT / no of Process

38/4 = 9.5 msec

Avg (WT) = 22/4 = 5.5 msec

Gnatt Chart: