Process Scheduling

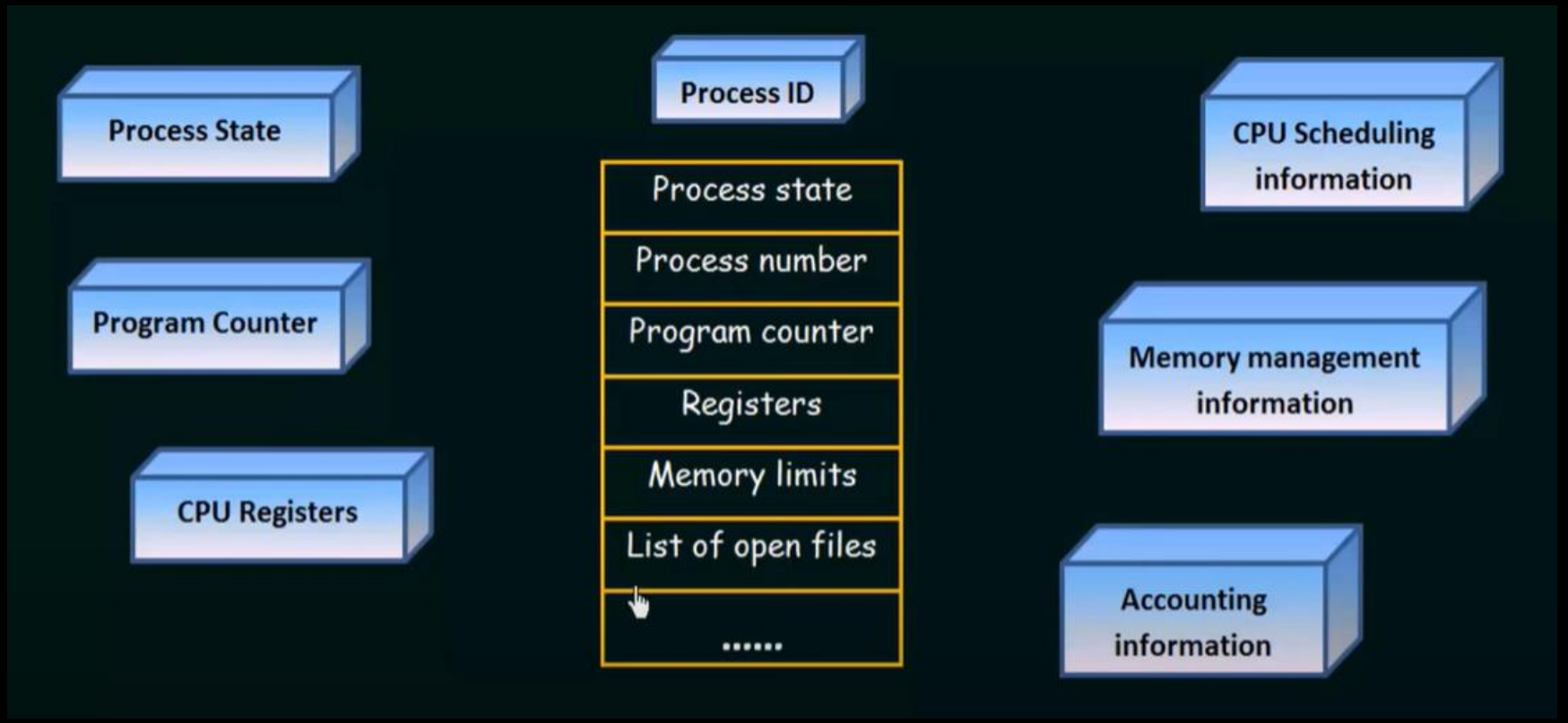
Asst.Prof.Rachna Amish Karnavat

PICT,IT

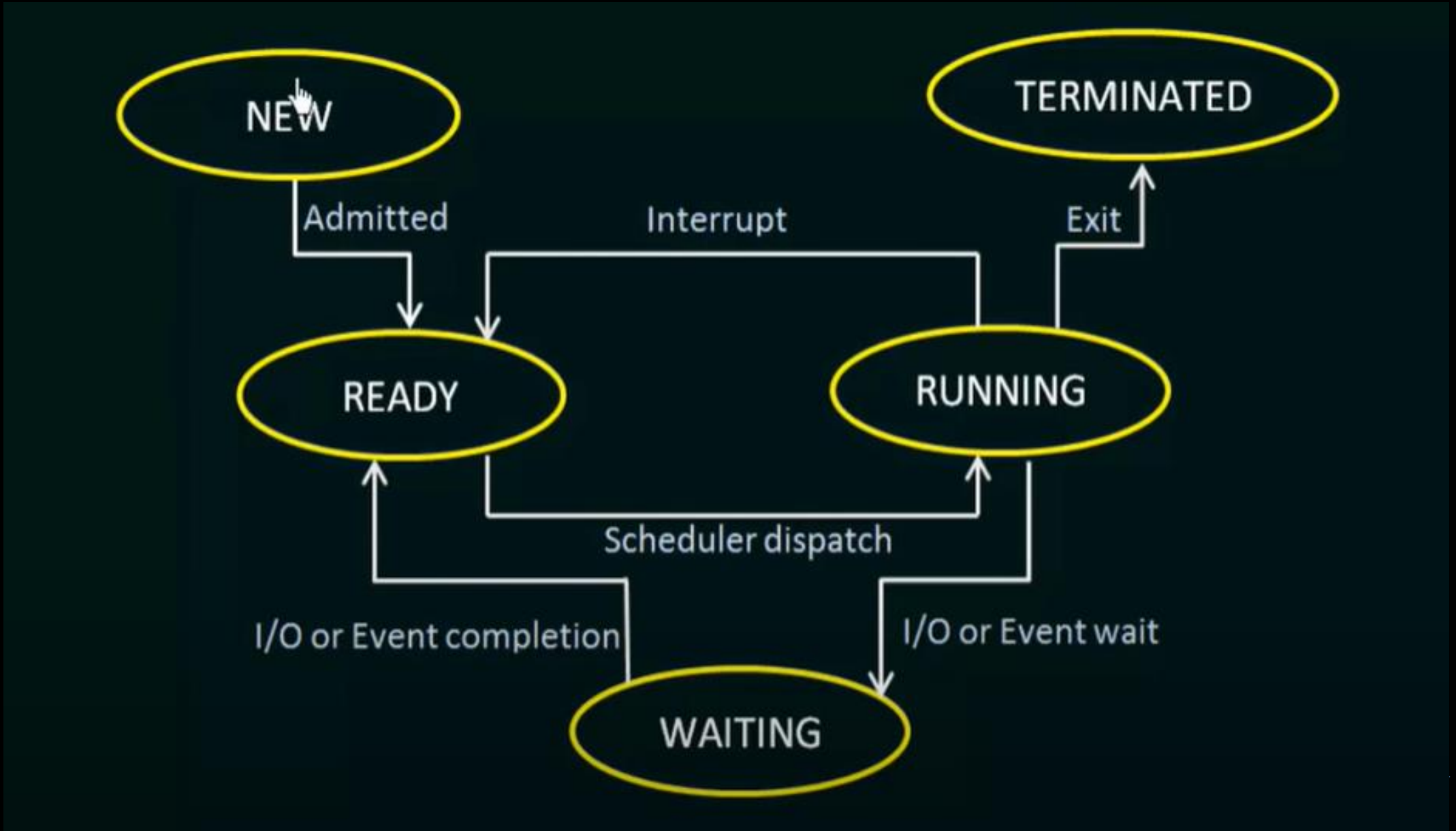
Points to cover

- Scheduling
- Types of Scheduling
- CPU Scheduling and Dispatcher
- Scheduling Criteria
- Non-preemptive & Preemptive Scheduling
- Scheduling Algorithms

Revision


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

Process States



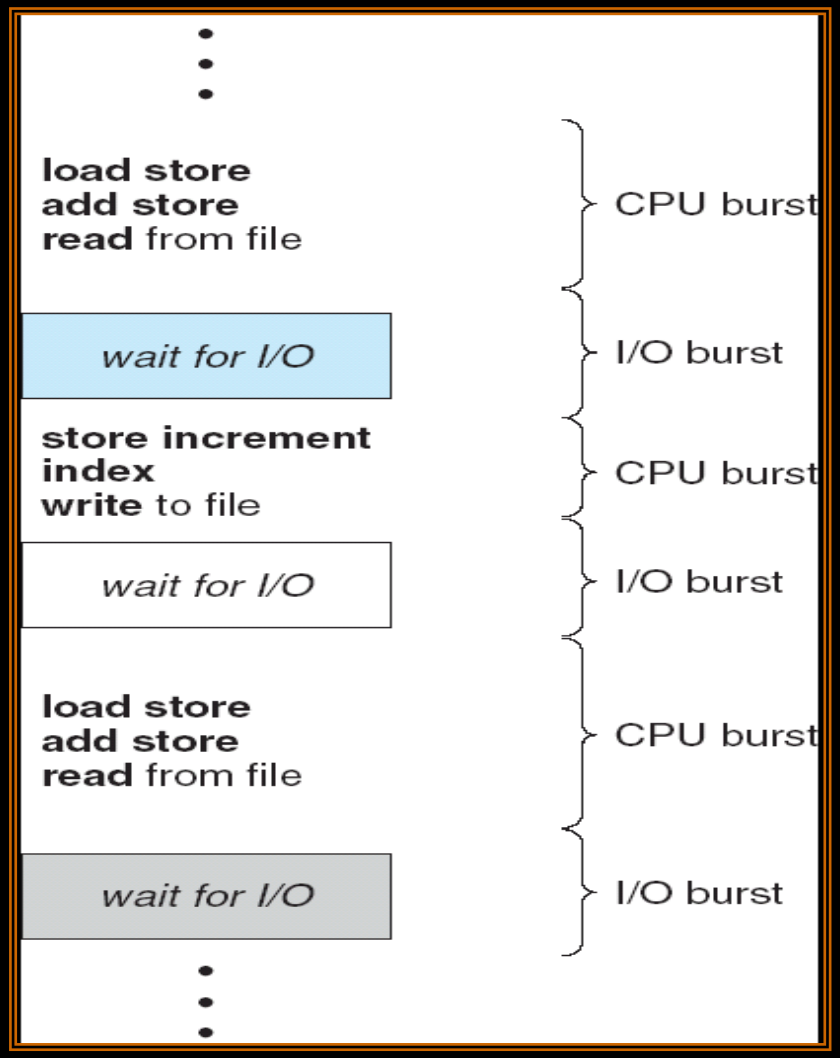
CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Scheduling in BSD UNIX

Basic Concepts

- 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.*

Alternating Sequence of CPU And I/O Bursts



Scheduling

- Select process(es) to run on processor(s).
- Process state is changed from “***ready***” to “***running***”.
- The component of the OS which does the scheduling is called the ***scheduler.***

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

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

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

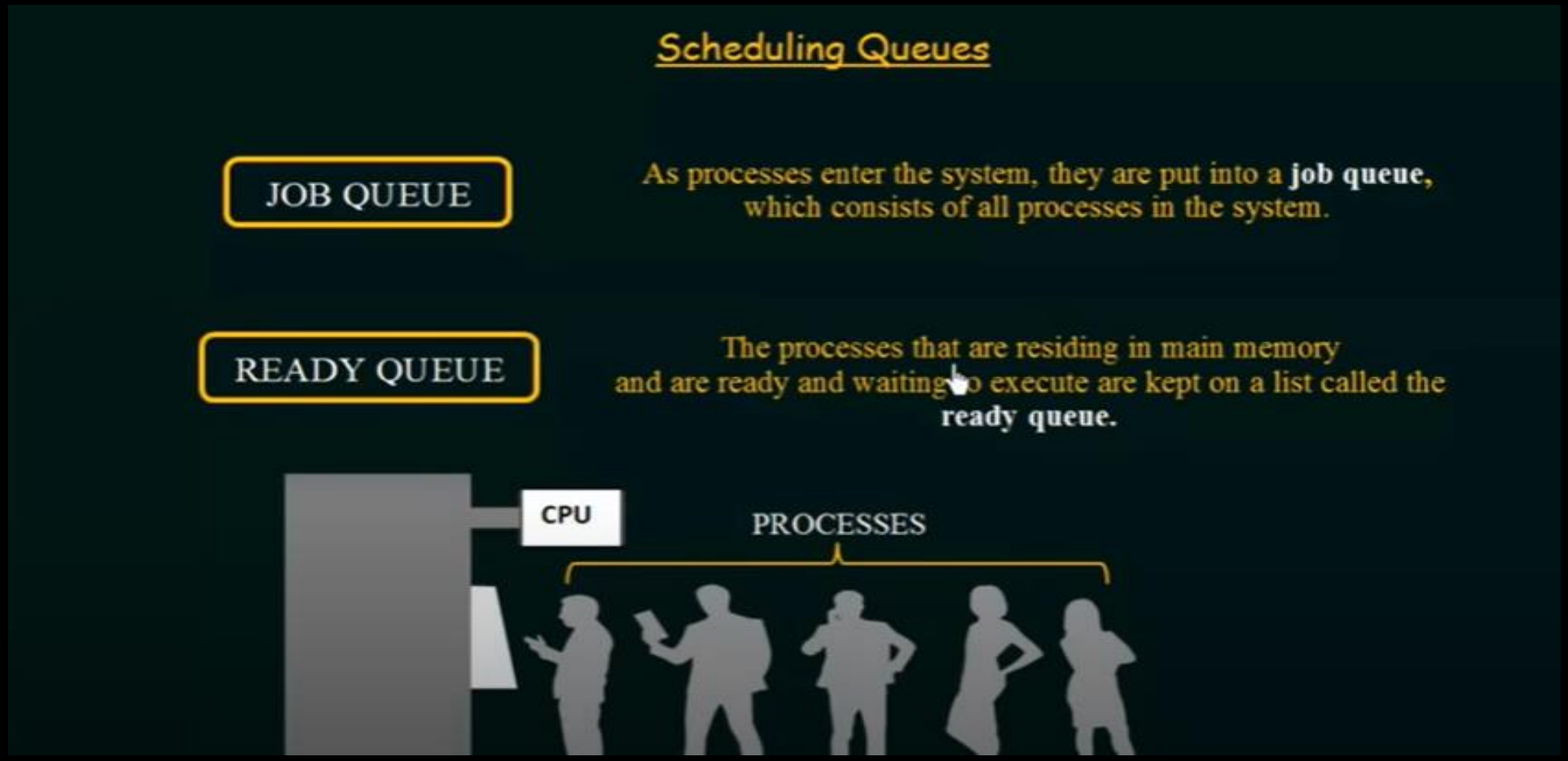
- With multiprogramming, we try to use this time productively.
  - Several processes are kept in memory at one time.

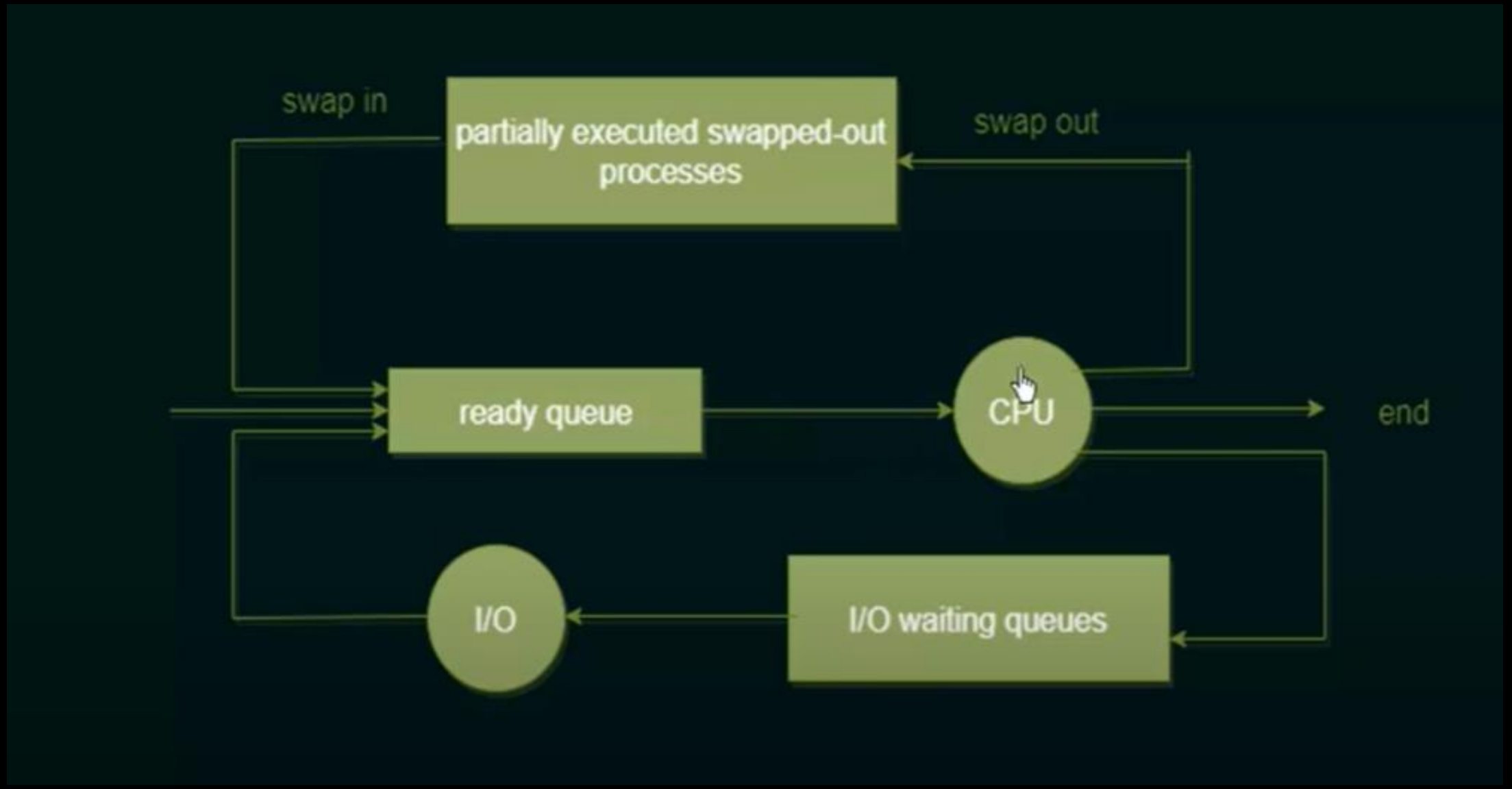
- When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process and this pattern continues.

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.

Scheduling Queues




swap in
partially executed swapped-out processes
swap out
ready queue
CPU
end
I/O
I/O waiting queues

Context Switching

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

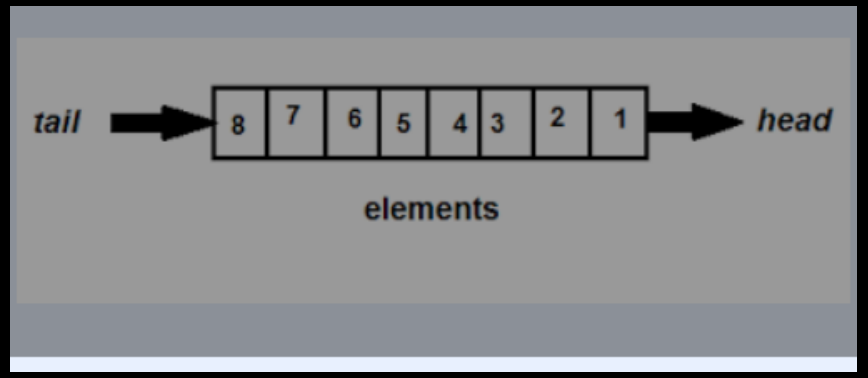
Context Switching

- 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

Scheduling Algorithms: FCFS

First-Come, First-Served (FCFS) Scheduling

- It is simplest CPU-Scheduling algorithm.
- The process that request 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..

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

Suppose that the processes arrive in the order: P_1 , P_2 , P_3
The Gantt Chart for the schedule is:

Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
Average waiting time: (0 + 24 + 27)/3 = 17

- 17Aug
- Absent Number
- 4,7,23,24,25,33,34,38,44,57,58,62,65,66,67,75,77,79,81,86,88,

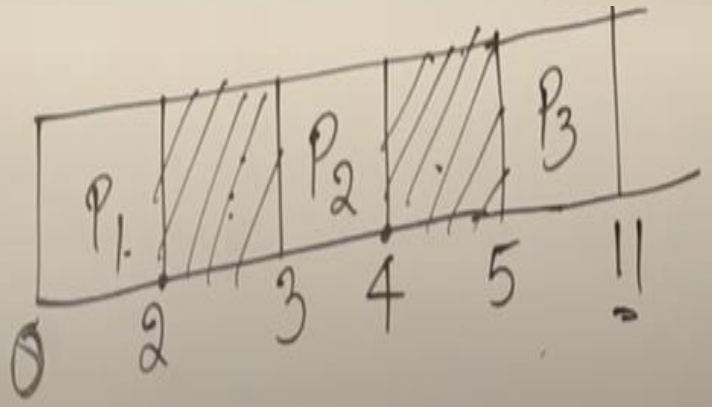
FCFS with different arrival times)

FCFS Scheduling of processes with same arrival time

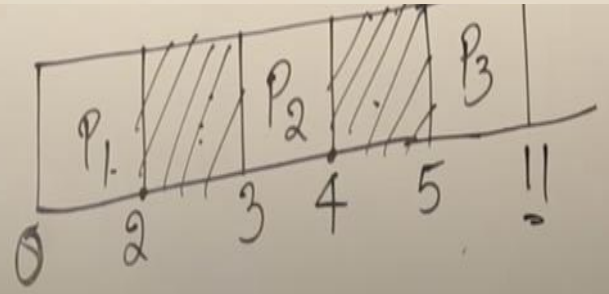
- Given n processes with their burst times and arrival times, the task is to find average waiting time and an average turn around time using FCFS scheduling algorithm
- Completion Time: Time at which the process completes its execution.
- Turn Around Time: Time Difference between completion time and arrival time.
- Turn Around Time = Completion Time – Arrival Time
- Waiting Time(W.T): Time Difference between turn around time and burst time.
  Waiting Time = Turn Around Time – Burst Time.

Example:Case 1

PNO	AT	BT
1	0	2
2	3	1
3	5	6

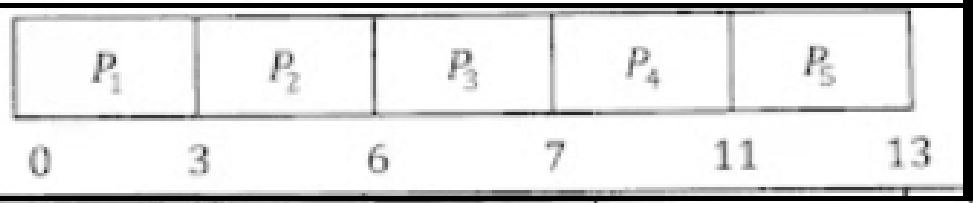


PNO	AT	BT	CT	TAT	WT
1	0	2	2	2	0
2	3	1	4	1	0
3	5	6	11	6	0



Example: Calculate the turn around time, waiting time, average turn around time, average waiting time, throughput, and processor utilization for the given set of processes that arrive at a given arrival time.

Process	Arrival Time	Processing Time
P_1	0	3
P_2	2	3
P_3	3	1
P_4	5	4
P_5	8	2



Time	Process Complete	Turn Around Time	Processing Time
0	–	–	–
3	P_1	3 – 0 = 3	3 – 3 = 0
6	P_2	6 – 2 = 4	4 – 3 = 1
7	P_3	7 – 1 = 6	6 – 1 = 5
11	P_4	11 – 4 = 7	7 – 4 = 3
13	P_5	13 – 2 = 11	11 – 2 = 9

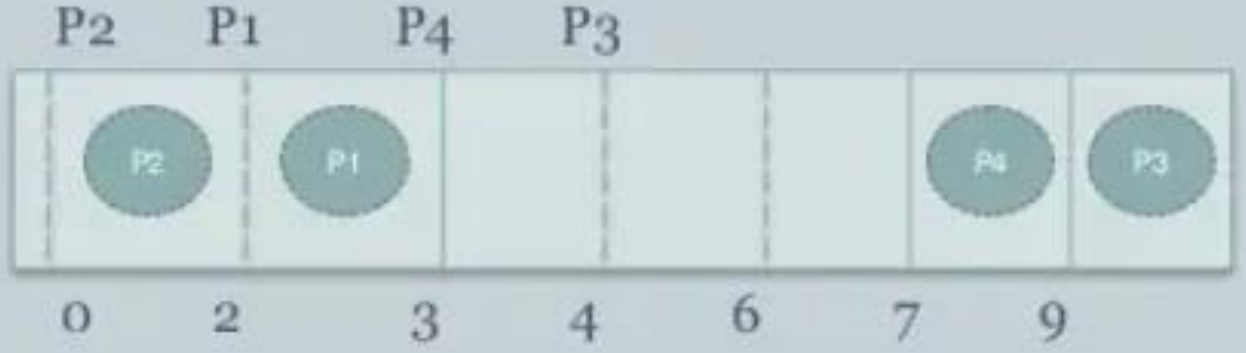
$$\text{Average turn around time} = \frac{(3+4+6+7+11)}{5} = 6.2$$

$$\text{Average waiting time} = \frac{(0+1+5+3+9)}{5} = 3.6$$

Example of FCFS

- Example.

Process	AT	BT	WT	RT	TT
P2	0	3	0	0	3
P4	4	2	3	3	5
P3	6	3	3	3	6
P1	2	4	1	1	5



AWT	1.75
ART	1.75
ATT	4.75

18Aug
Absent Number

4,,18,23,28,25,,57,62,
68,75,86,

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.

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

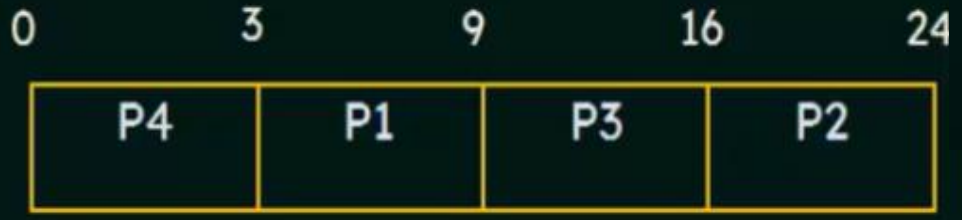
Waiting Time for P3 = 9 ms

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.

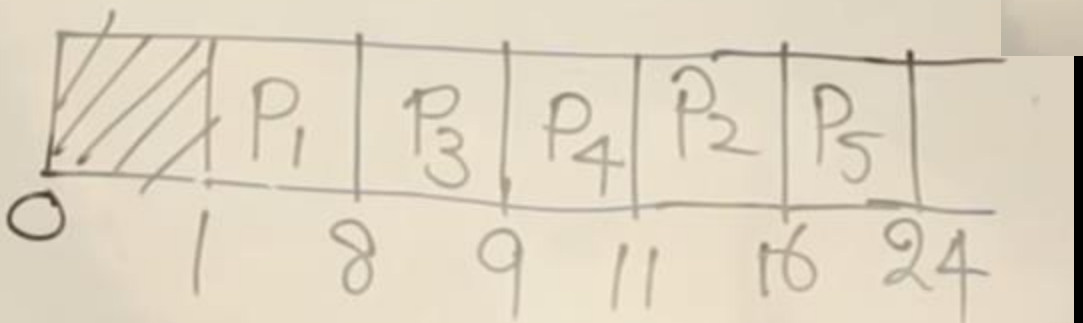
Shortest Job First

Shortest Job first

criteria – Burst time

mode – Non preemptive

PNO	AT	BT
1	1	7
2	2	5
3	3	1
4	4	2
5	5	8



PNO	AT	BT	CT	TAT
1	1	7	8	7
2	2	5	16	14
3	3	1	9	6
4	4	2	11	7
5	5	8	24	19

PNO	AT	BT	CT	TAT	WT
1	1	7	8	7	0
2	2	5	16	14	9
3	3	1	9	6	5
4	4	2	11	7	5
5	5	8	24	19	11

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
P2	1	4
P3	2	9
P4	3	5

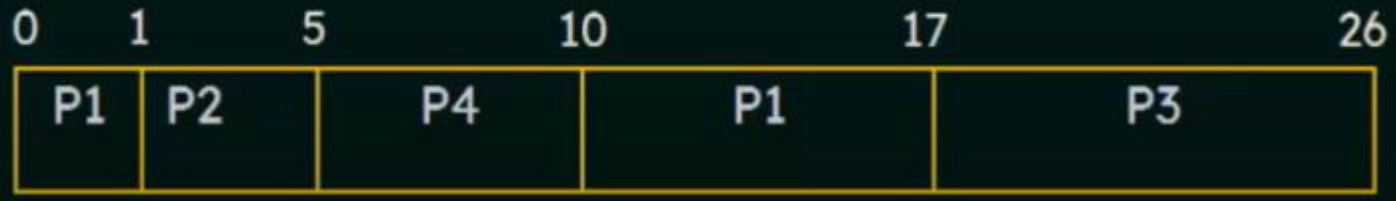
Waiting Time for P1 = (10 – 1 - 0) = 9 ms

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:



Average Waiting Time

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

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

Preemptive SJF scheduling is sometimes called Shortest-Remaining-Time-First Scheduling.

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF)
- SJF is optimal – gives minimum average waiting time for a given set of processes

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.

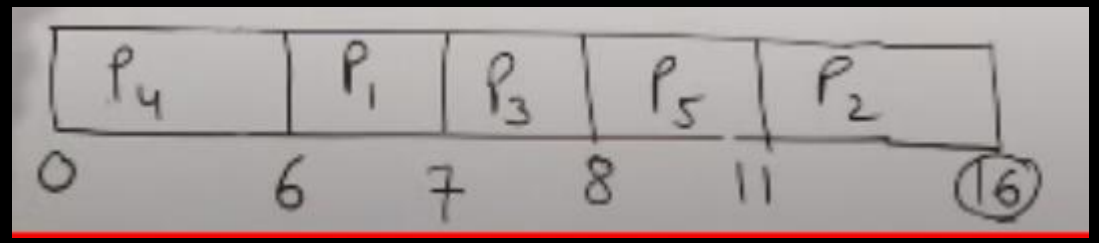
One approach is:

- 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.

	Arrival time	Burst time
P_1	2	1
P_2	1	5
P_3	4	1
P_4	0	6
P_5	2	3

20 Aug
Absent Number

4,7,8,,28,40,42,43,44,48,55,57,62,66,68,,72,74,82
,,85,86,,88,



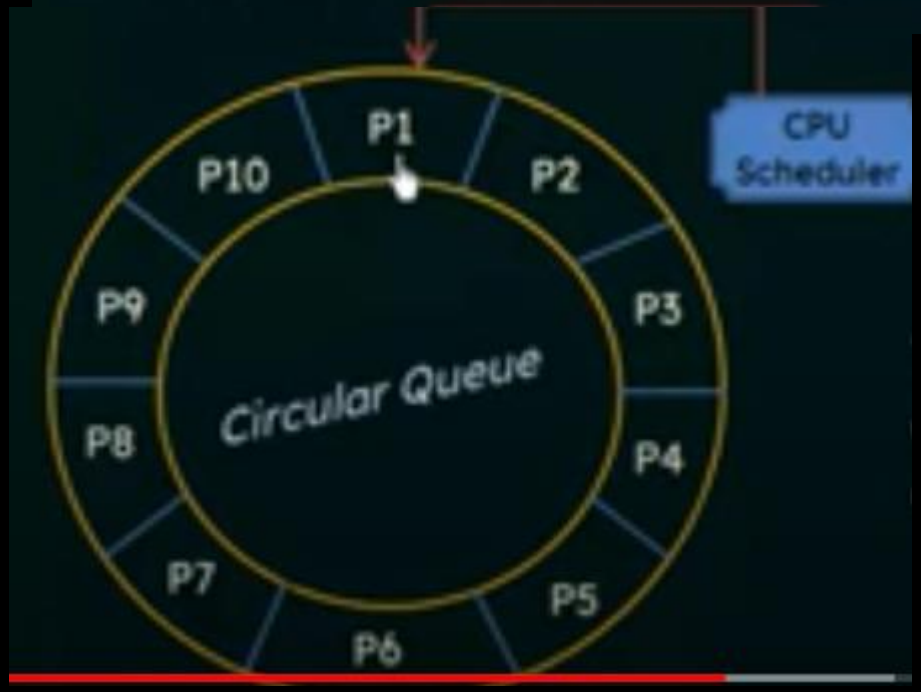
	Arrival time	Burst time	CT	TAT	WT	Response time
P_1	2	1	7	5	4	4
P_2	1	5	16	15	10	10
P_3	4	1	8	4	3	3
P_4	0	6	6	6	0	0
P_5	2	3	11	9	6	6

Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Round Robin (RR)

- 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.

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

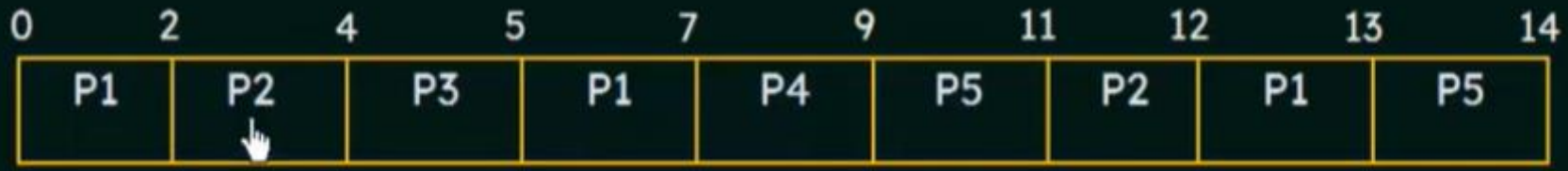
Process ID	Arrival Time	Burst Time
P1	0	5- 3
P2	1	3
P3	2	1
P4	3	2
P5	4	3

Time Quantum
2 Units

Clock

(((Alarm Rings)))

Gantt Chart:



P1 P3 Ready Queue

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

Time Quantum
2 Units

Clock

(((Alarm Rings)))

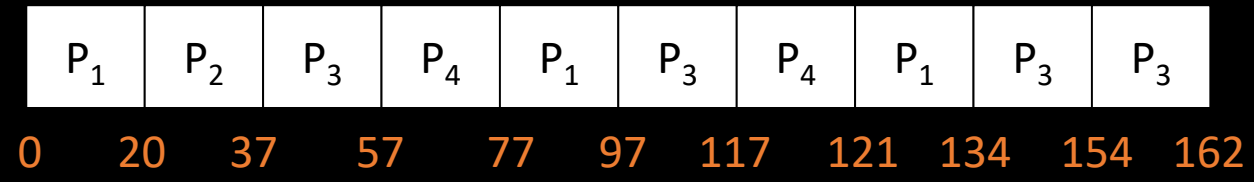
Gantt Chart:

Ready Queue

Example of RR with Time Quantum

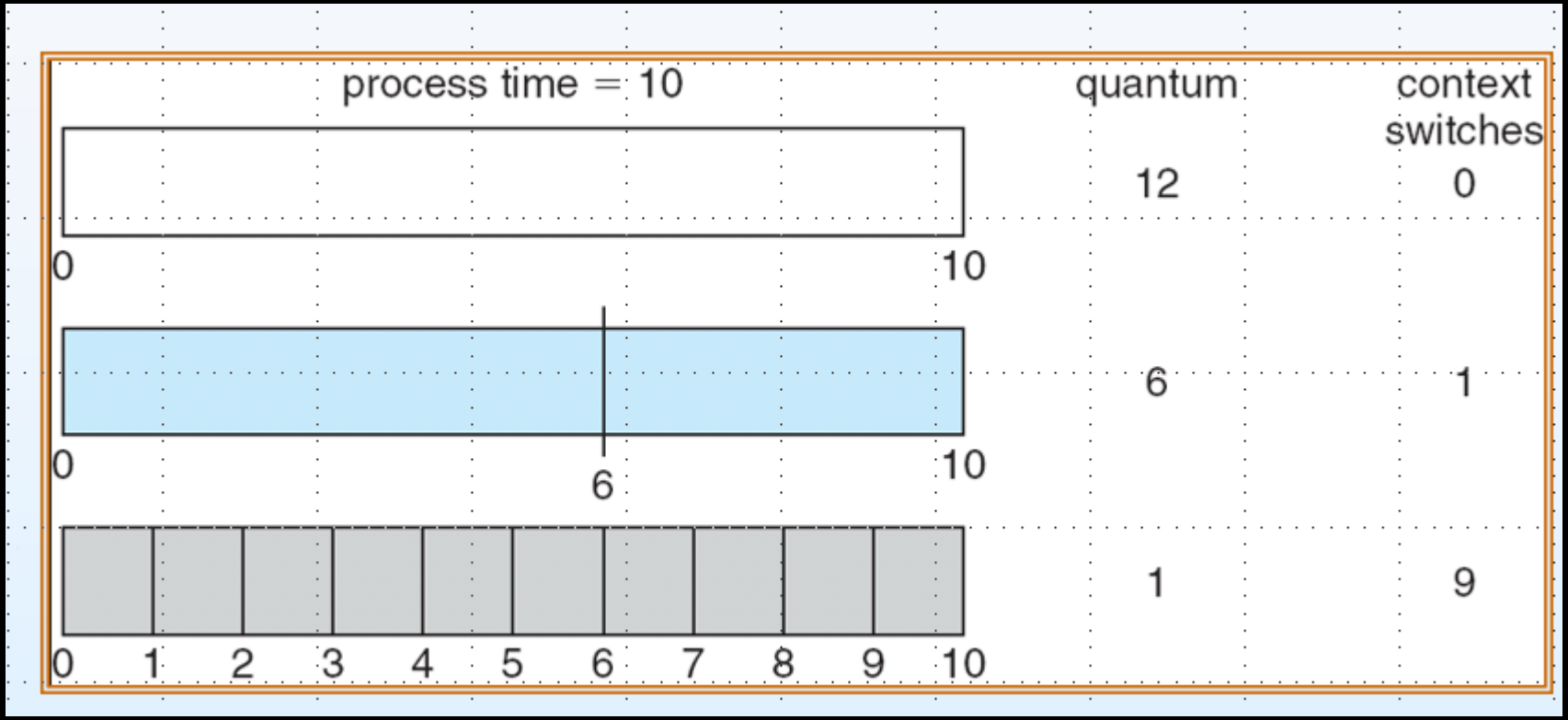
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better *response*

Time Quantum and Context Switch Time



Round Robin Example

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 unit, calculate the average waiting time and average turn around time.

Solution:

Gantt Chart

Process Id	Exit time	Turn Around 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

Now,

- Average Turn Around time = (13 + 11 + 3 + 6 + 10) / 5 = 43 / 5 = 8.6 unit
- Average waiting time = (8 + 8 + 2 + 4 + 7) / 5 = 29 / 5 = 5.8 unit

Round Robin

- **Advantage of Round-robin Scheduling**
  - It doesn't face the issues of starvation or convoy effect.
  - All the jobs get a fair allocation of CPU.
  - It deals with all process without any priority
  - If you know the total number of processes on the run queue, then you can also assume the worst-case response time for the same process.
  - This scheduling method does not depend upon burst time. That's why it is easily implementable on the system.
  - Once a process is executed for a specific set of the period, the process is preempted, and another process executes for that given time period.
  - Allows OS to use the Context switching method to save states of preempted processes.
  - It gives the best performance in terms of average response time.

Convoy Effect: one slow process slows down the performance of the entire set of processes, and leads to wastage of CPU time and other devices.

To avoid Convoy Effect, preemptive scheduling algorithms like Round Robin Scheduling can be used – as the smaller processes don’t have to wait much for CPU time – making their execution faster and leading to less resources sitting idle.

Round Robin

- **Disadvantage of Round-robin Scheduling**
  - If slicing time of OS is low, the processor output will be reduced.
  - This method spends more time on context switching
  - Its performance heavily depends on time quantum.
  - Priorities cannot be set for the processes.
  - Round-robin scheduling doesn't give special priority to more important tasks.
  - Decreases comprehension
  - Lower time quantum results in higher the context switching overhead in the system.
  - Finding a correct time quantum is a quite difficult task in this system.

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.

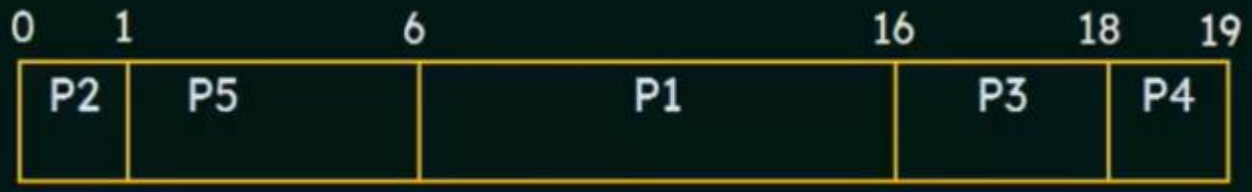
A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.

A nonpreemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.

Consider the following set of processes, assumed to have arrived at time 0, in the order P1, P2, P3, P4, P5, with the length of the CPU burst given in milliseconds:

Process ID	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Using Priority Scheduling, we would schedule these processes according to the following Gantt Chart:



Waiting Time for P1 = 6 ms

Waiting Time for P2 = 0 ms

Waiting Time for P3 = 16 ms

Waiting Time for P4 = 18 ms

Waiting Time for P5 = 1 ms

Average Waiting Time

= (6 + 0 + 16 + 18 + 1) / 5

= 41 / 5 ms

= 8.2 ms

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.

Priority Scheduling

Solved Problem - 2

Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds), and priority shown below: (Higher number represents higher priority)

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

If the CPU scheduling policy is priority non-preemptive, calculate the average waiting time and average turn around time.

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

Solution:

Gantt Chart:

Process ID	Completion Time	Turnaround Time	Waiting Time
P1	4	4 - 0 = 4	4 - 4 = 0
P2	15	15 - 1 = 14	14 - 3 = 11
P3	12	12 - 2 = 10	10 - 1 = 9
P4	9	9 - 3 = 6	6 - 5 = 1
P5	11	11 - 4 = 7	7 - 2 = 5

Average Turn Around time

= (4 + 14 + 10 + 6 + 7) / 5

= 41 / 5 = 8.2 ms

Average waiting time

= (0 + 11 + 9 + 1 + 5) / 5

= 26 / 5 = 5.2 ms

Find the Avg. TAT and Avg. WT wrt SJF(P) and priority (P). Also draw the gantt Chart.

Process Name	Arrival Time	Burst Time	Priority
A	9	4	1
B	3	4	2
C	0	8	3
D	1	6	4
E	12	6	2

Threads

A thread is a basic unit of CPU utilization.

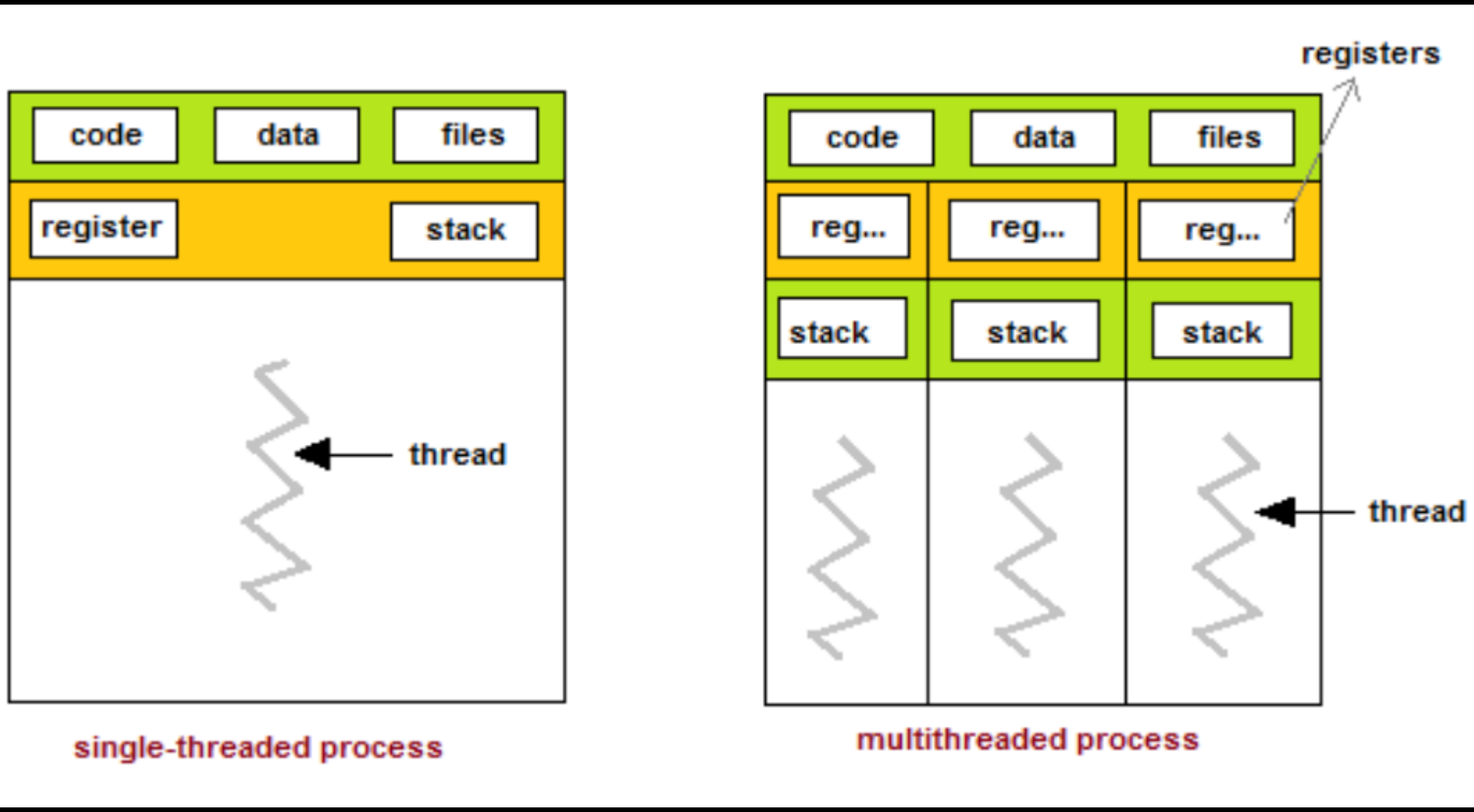
It comprises

- 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.


code
data
files
register
stack
thread
single-threaded process
registers
code
data
files
reg...
reg...
reg...
stack
stack
stack
thread
multithreaded process

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Advantages of Thread

- Responsiveness
- Resource sharing, hence allowing better utilization of resources.
- Economy. Creating and managing threads becomes easier.
- Scalability. One thread runs on one CPU. In Multithreaded processes, threads can be distributed over a series of processors to scale.
- Context Switching is smooth. Context switching refers to the procedure followed by the CPU to change from one task to another.
- Enhanced Throughput of the system.
- Let us take an example for this: suppose a process is divided into multiple threads, and the function of each thread is considered as one job, then the number of jobs completed per unit of time increases which then leads to an increase in the throughput of the system.

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.

Types of thread

1.User Threads

These are above the kernel and without kernel support. These are the threads that application programmers use in their programs

2.Kernel Thread

These are supported within the kernel of the OS itself. All modern OSs support kernel-level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

Difference between User-Level & Kernel-Level Thread

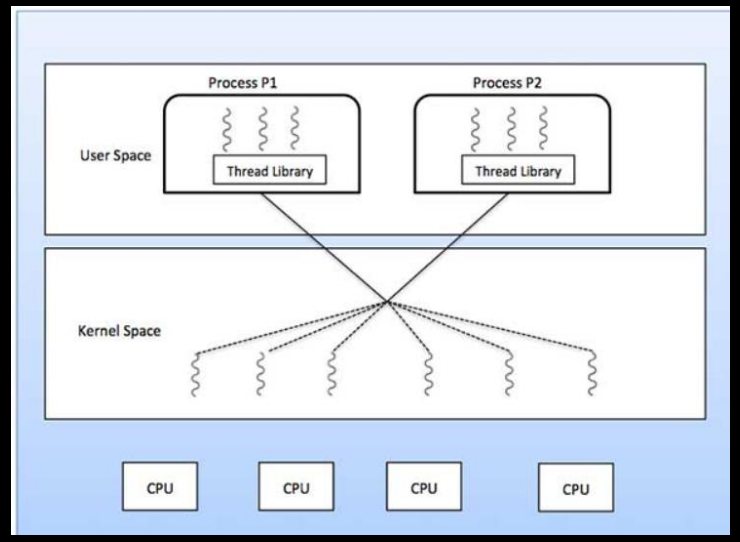
S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Multithreading Models

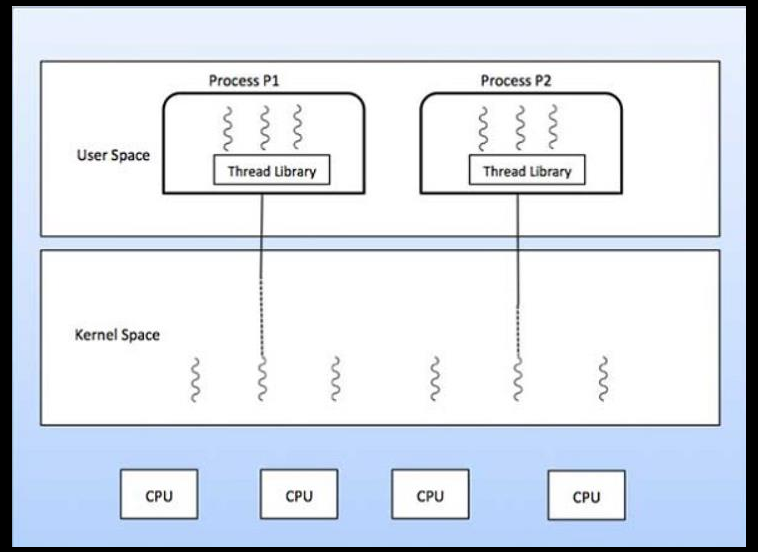
Multithreading models are three types

- Many to many relationship.
- Many to one relationship.
- One to one relationship.

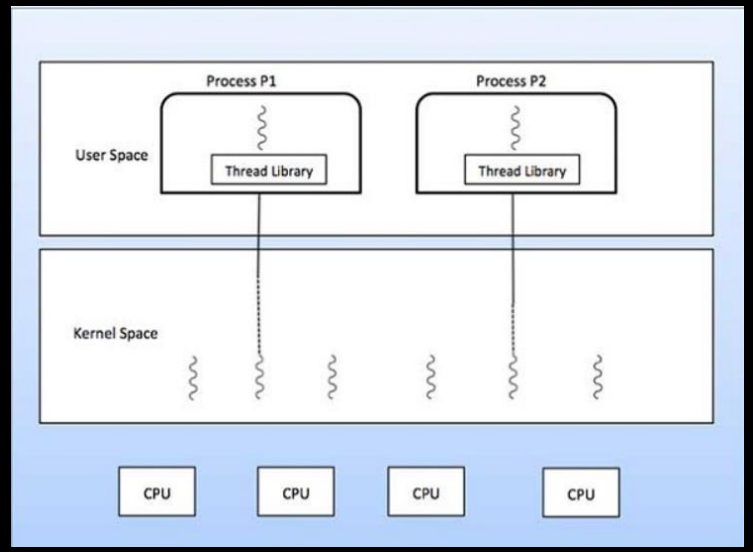
Many to many relationship.



Many to one relationship



One to one relationship



Thread Libraries

- Thread libraries provide programmers with API for the creation and management of threads.
- It may be implemented either in user space or in kernel space.
- The user space involves API functions implemented solely within the user space, with no kernel support.
- The kernel space involves system calls and requires a kernel with thread library support.

Three Types of Threads:

1.POSIX Thread: may be provided as either a user or kernel library, as an extension to the POSIX standard.

2. **Win32 threads:** are provided as a kernel-level library on Windows systems.

3. **Java threads**: Since Java generally runs on a Java Virtual Machine, the implementation of threads is based upon whatever OS and hardware the JVM is running on, i.e. either Pitheads

ABSENT NUMBER 3 sep21

- 1,4,7,8,,35,,48,51,57,60,62,68,69,71,72,75,77,82,86