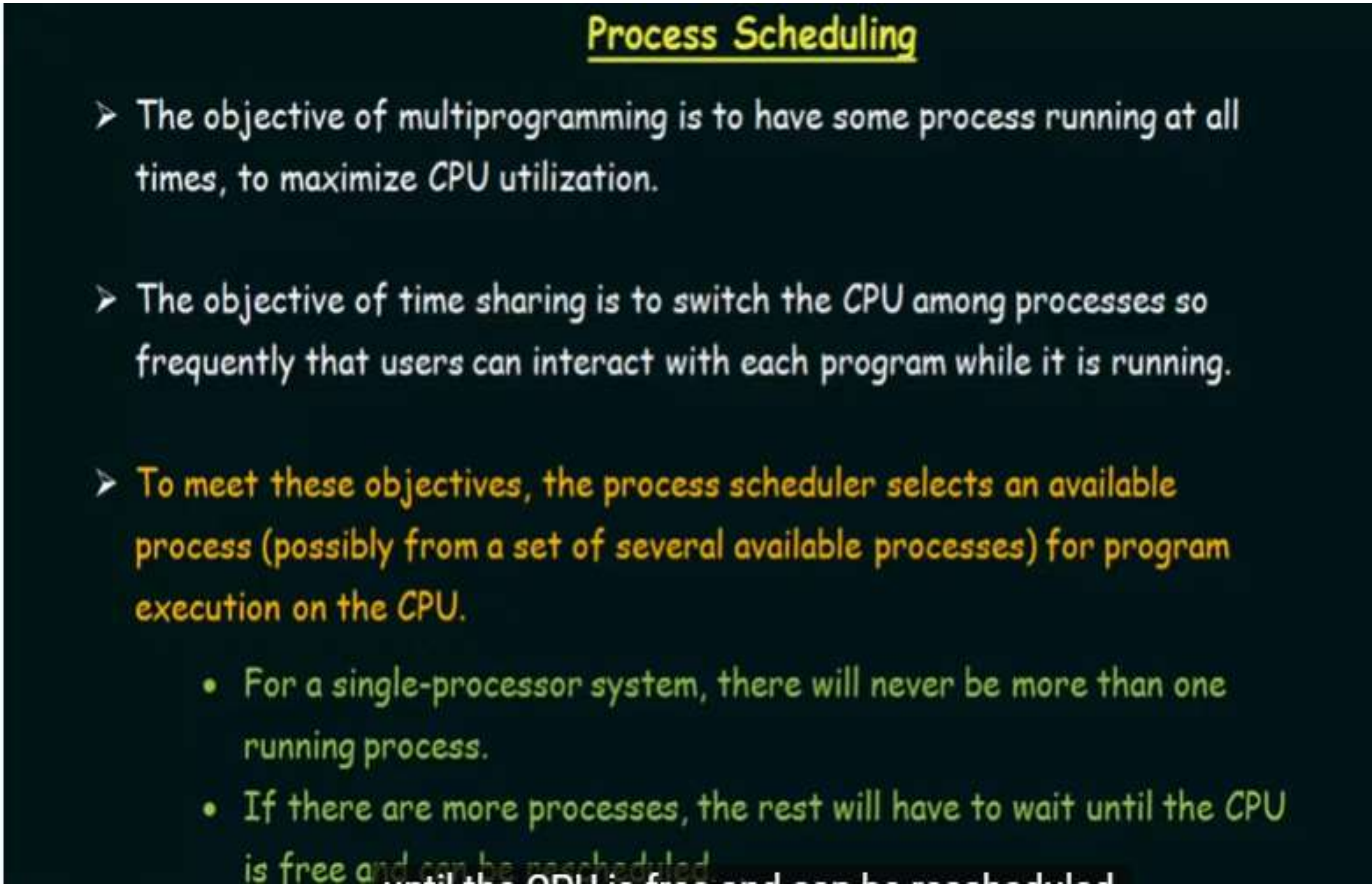
UNIT-1 (PART-2) Process Scheduling Criteria And Algorithms

PROCESS SCHEDULING



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.

until the CPU is free and can be rescheduled

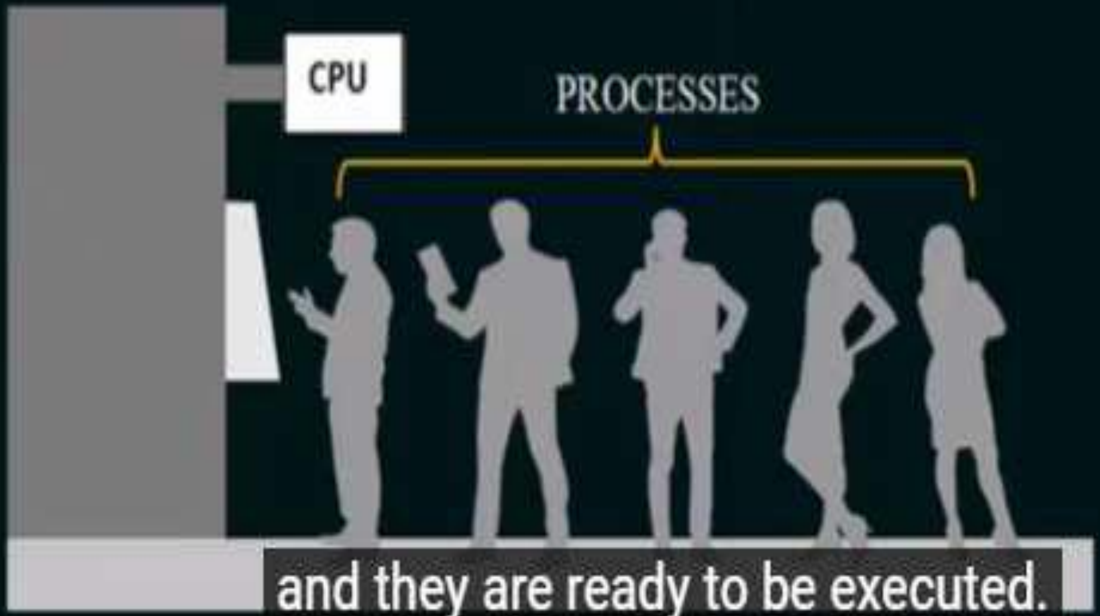
Scheduling Queues

JOB QUEUE

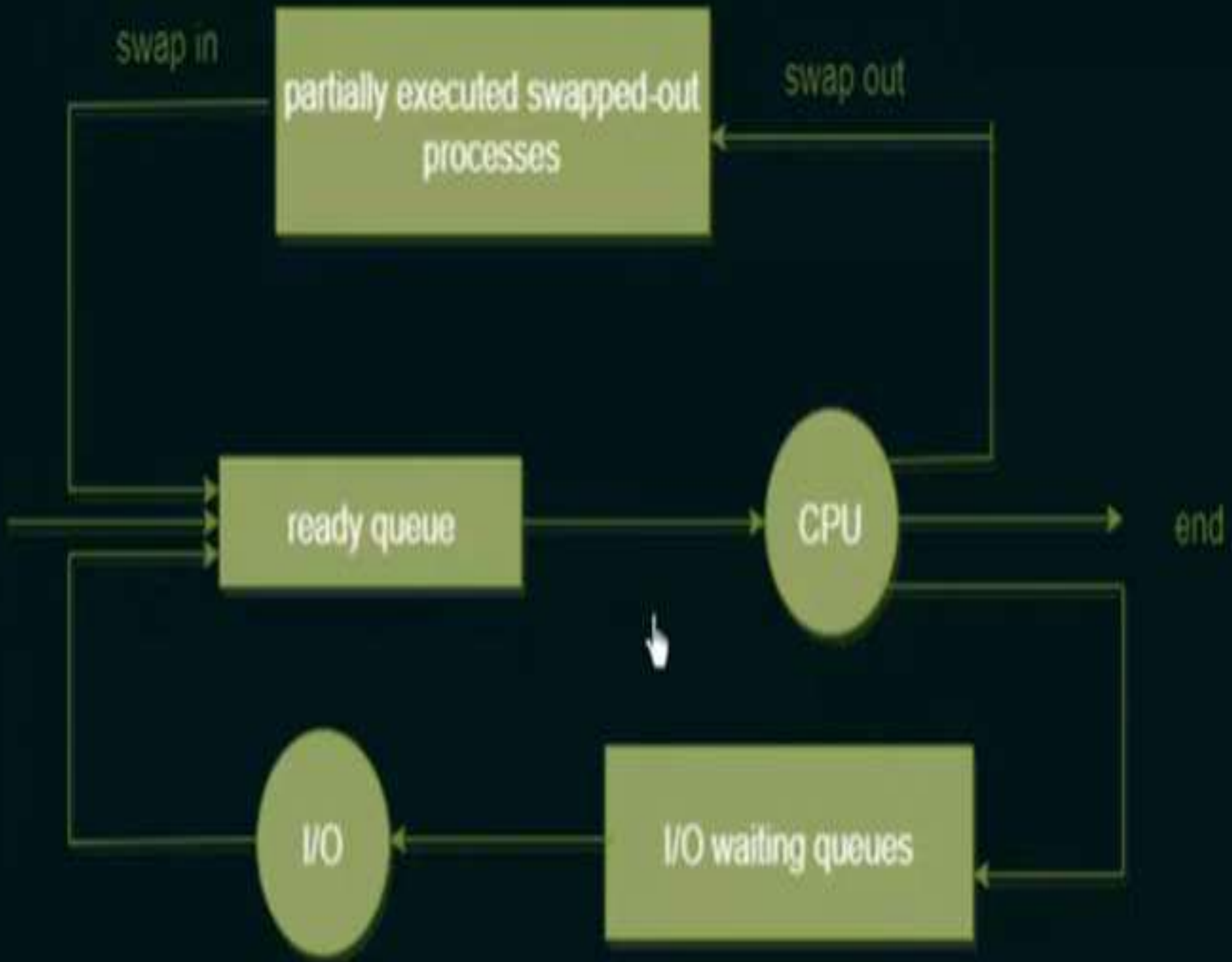
As processes enter the system, they are put into a **job queue**, which consists of all processes in the system.

READY QUEUE

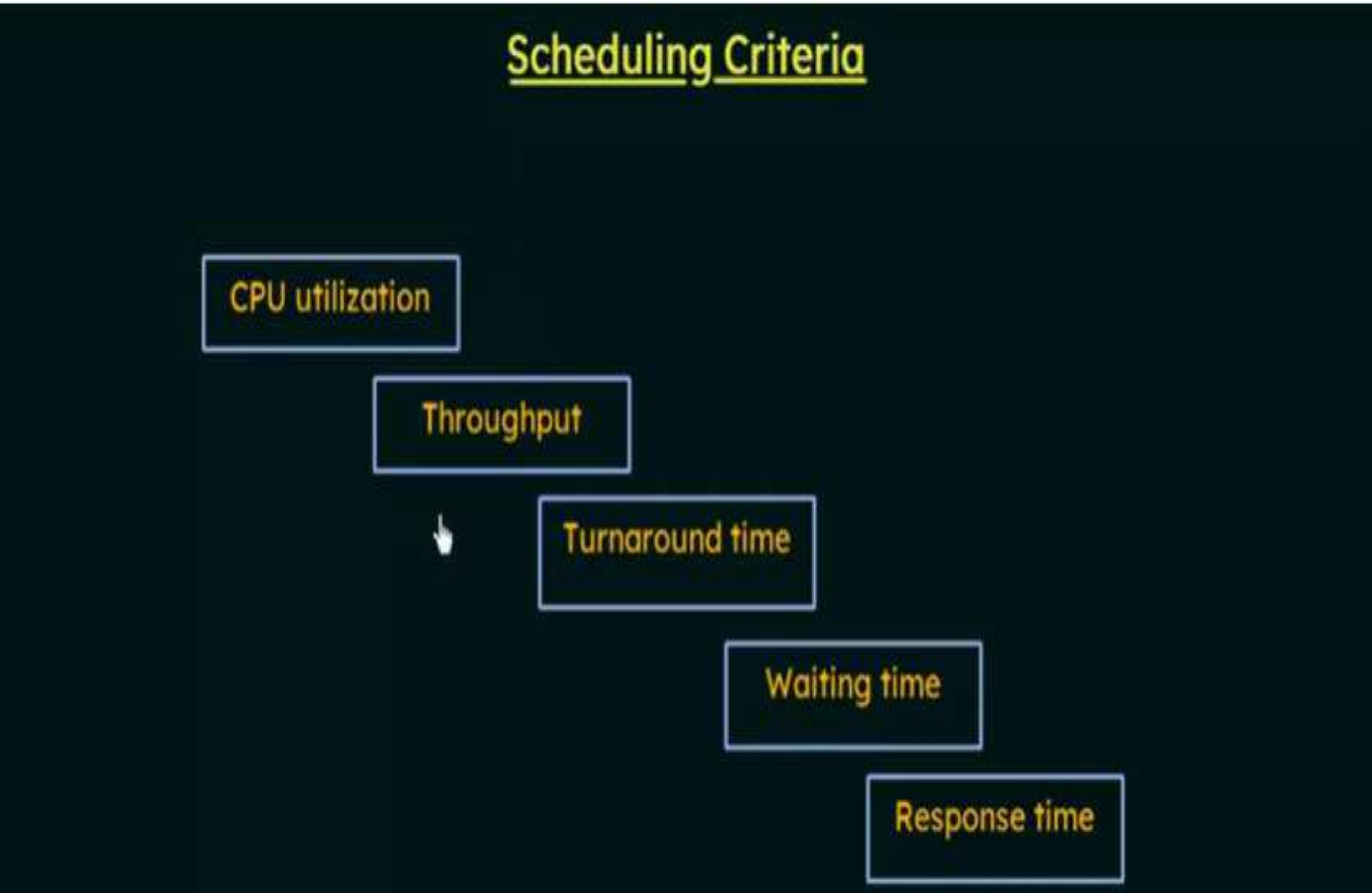
The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue.**



and they are ready to be executed.


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

SCHEDULING CRITERIA



Scheduling Criteria

CPU utilization

Throughput

Turnaround time

Waiting time

Response time

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.

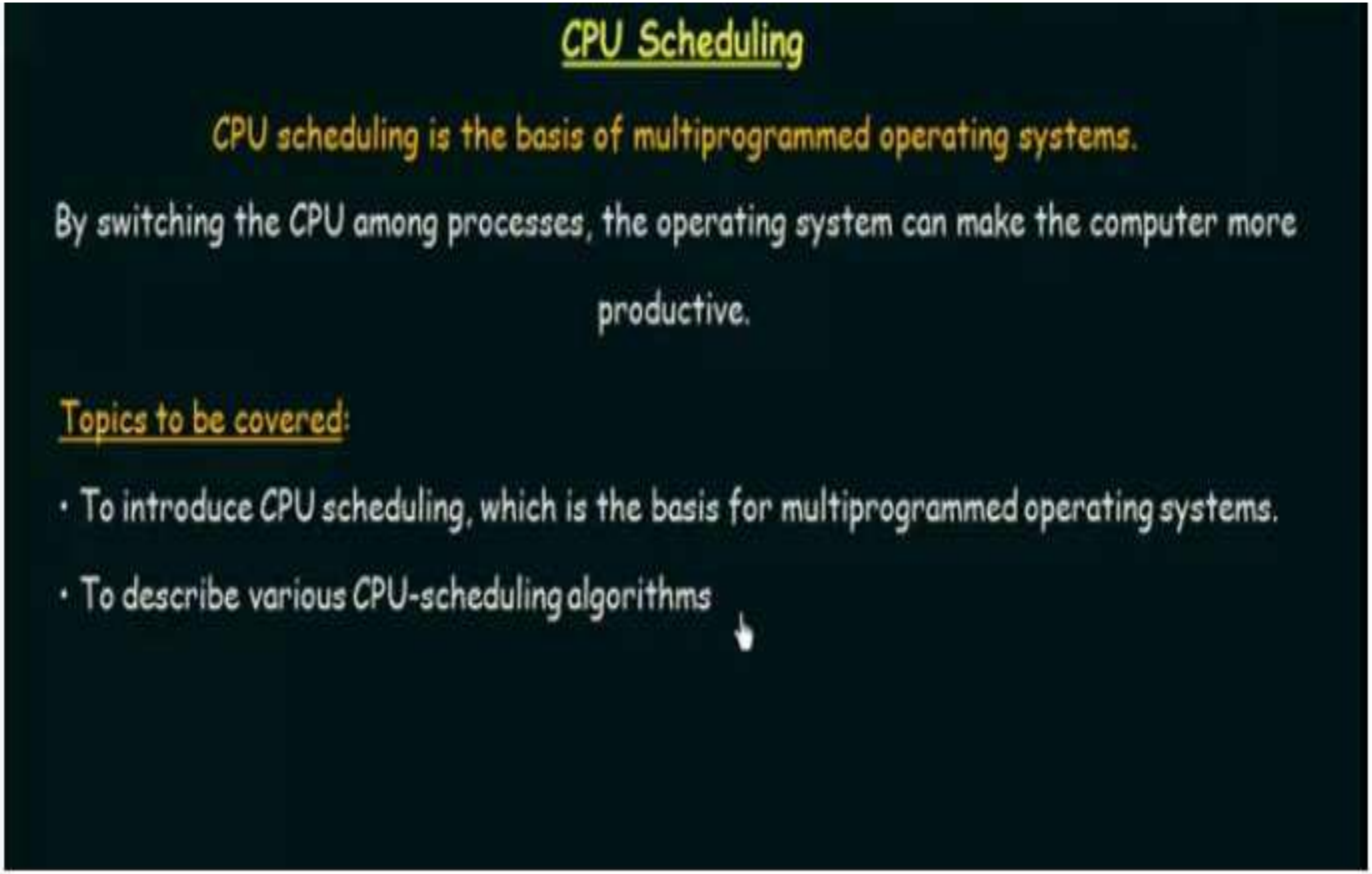
Waiting time

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.

CPU SCHEDULING



CPU Scheduling

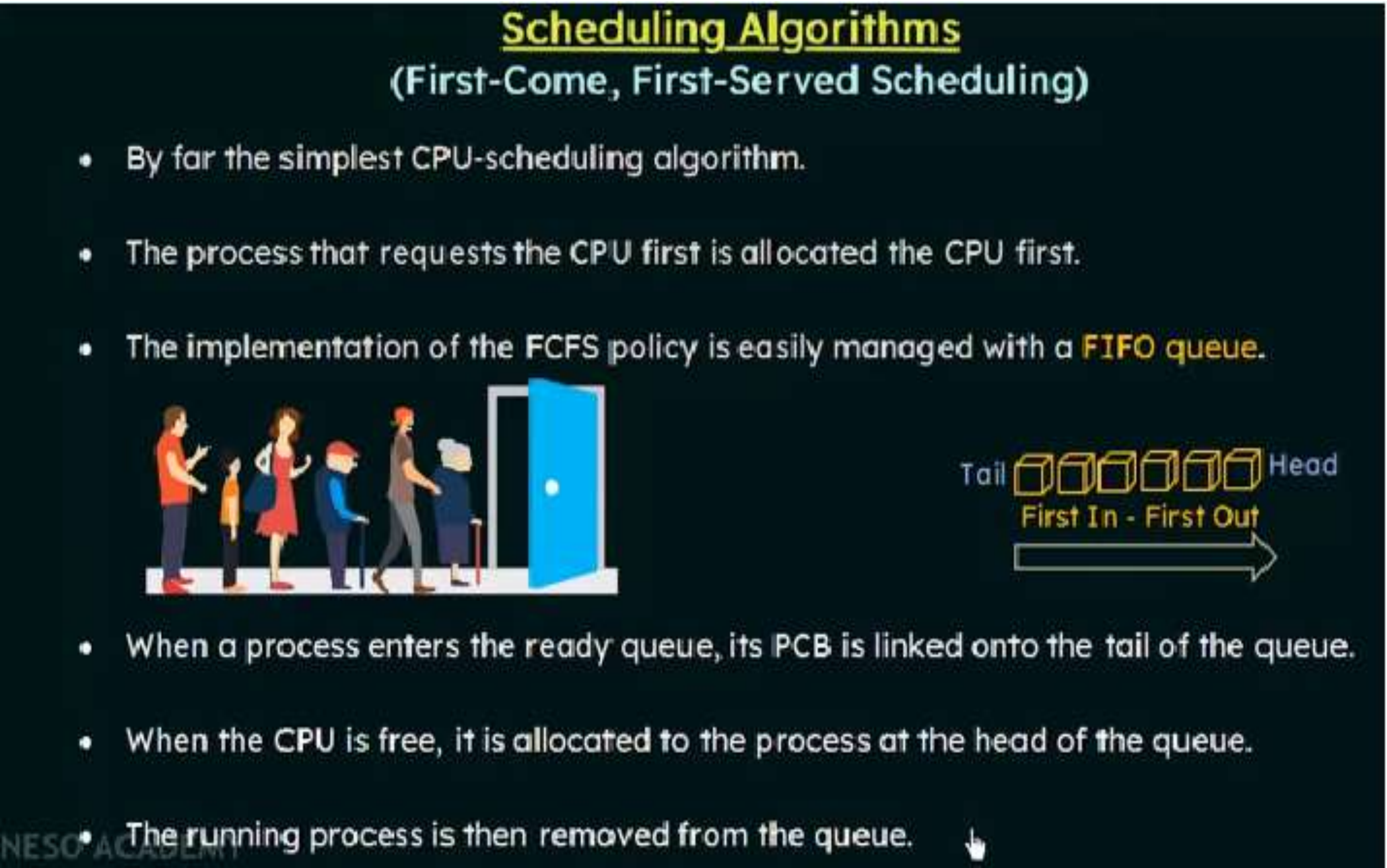
CPU scheduling is the basis of multiprogrammed operating systems.

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

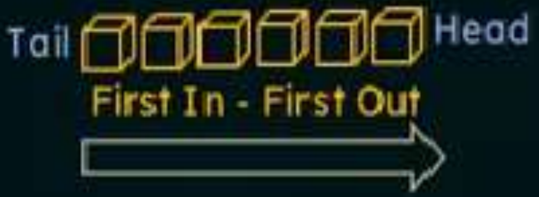
FCFS ALGORITHM



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.

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.

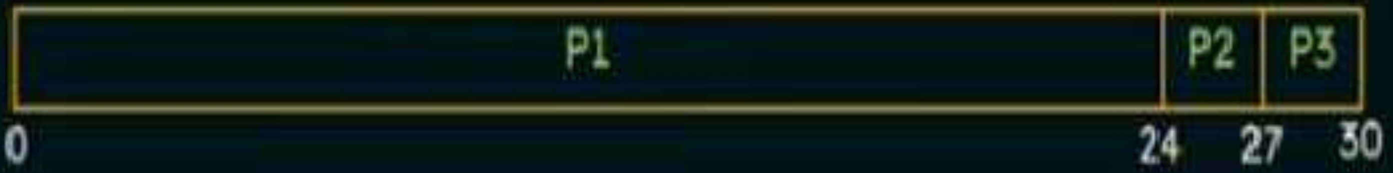
- The FCFS algorithm is thus particularly troublesome for time-sharing systems, where it is important that each user get a share of the CPU at regular intervals.

- It would be disastrous to allow one process to keep the CPU for an extended period.

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:



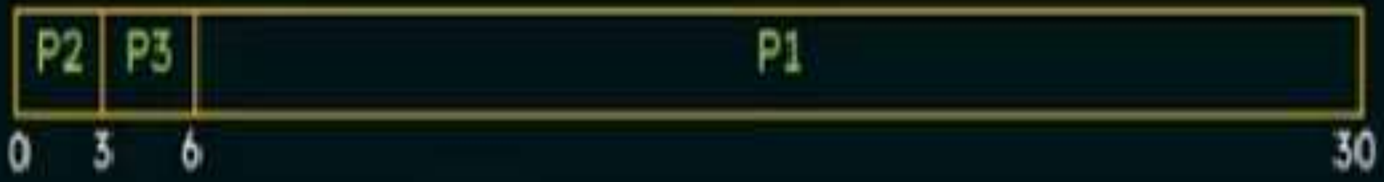
Waiting Time for P1 = 0 ms

Waiting Time for P2 = 24 ms

Waiting Time for P3 = 27 ms

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

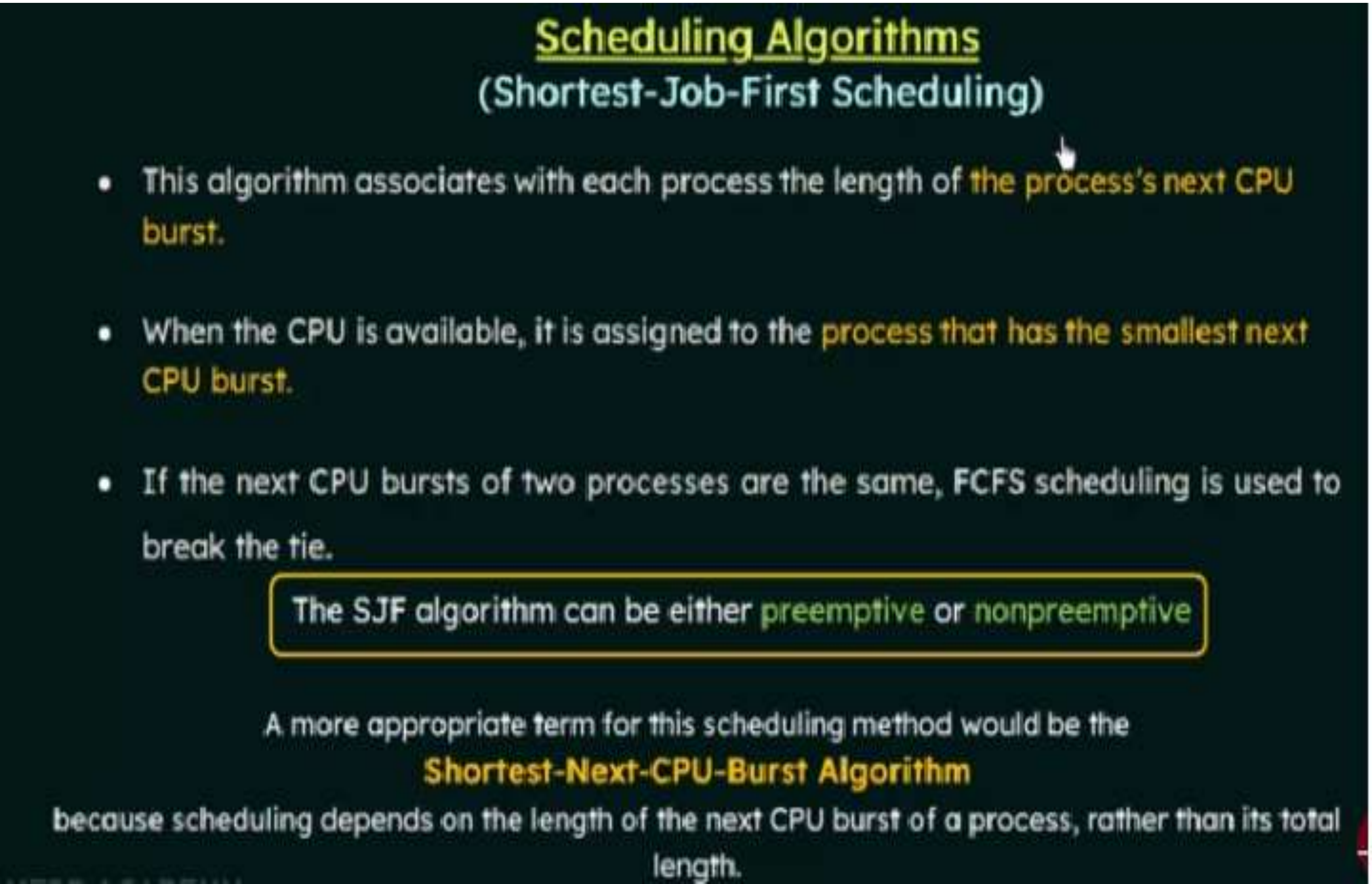
If the processes arrive in the order P2, P3, P1, however the result will be shown in the following Gantt chart:



$$\left.\begin{array}{l}\text{Waiting Time for P1} = 6\text{ ms}\\ \text{Waiting Time for P2} = 0\text{ ms}\\ \text{Waiting Time for P3} = 3\text{ ms}\end{array}\right\}\ \text{Average Waiting Time} = (6 + 0 + 3)/3 = 3\text{ ms}$$

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.

SJF ALGORITHM



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.

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

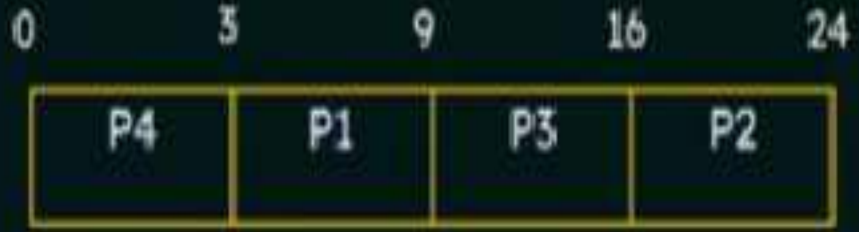
Waiting Time for P3 = 9 ms

Waiting Time for P4 = 0 ms

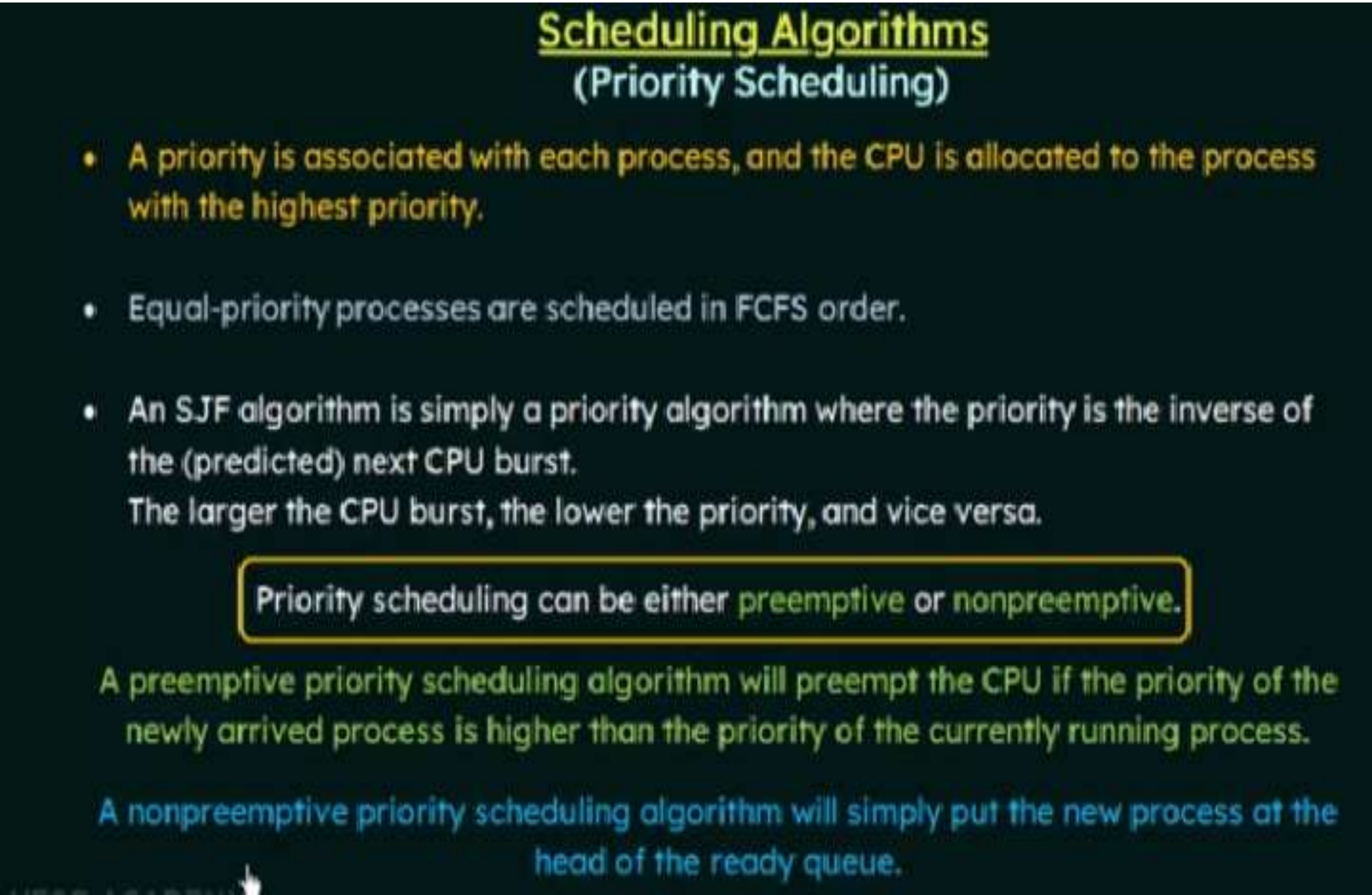
Average Waiting Time

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

Gantt Chart:



PRIORITY ALGORITHM



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

Waiting Time for P1 = 6 ms

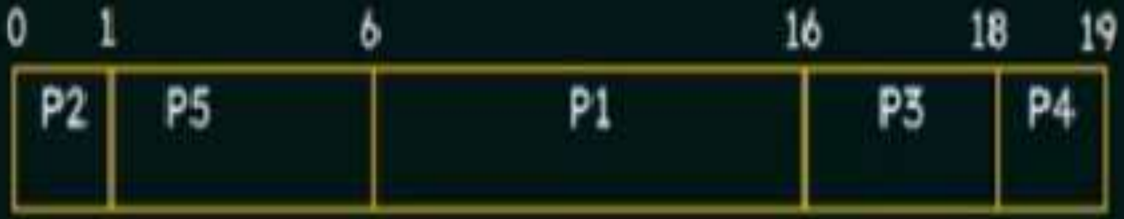
Waiting Time for P2 = 0 ms

Waiting Time for P3 = 16 ms

Waiting Time for P4 = 18 ms

Waiting Time for P5 = 1 ms

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



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

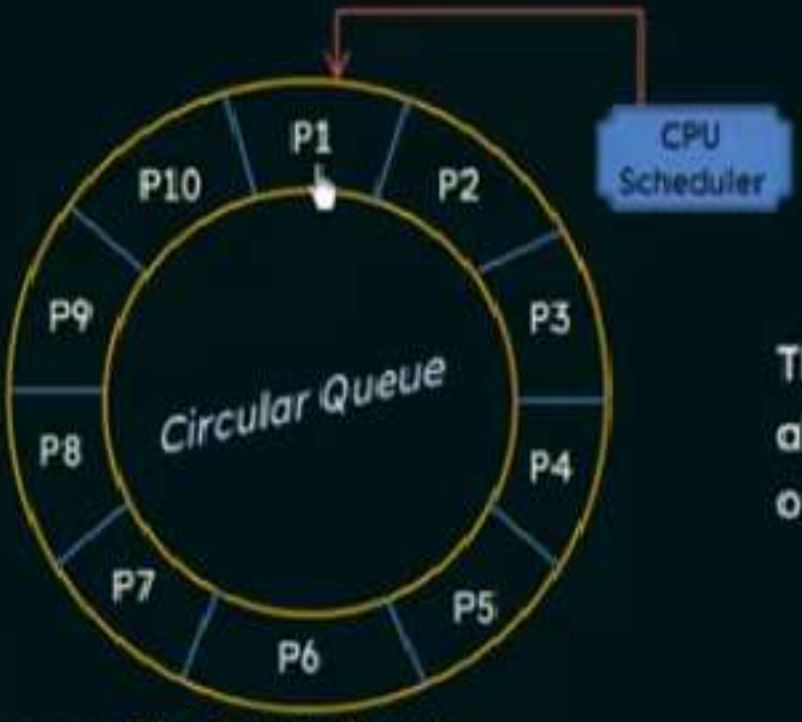
- A solution to the problem of indefinite blockage of low-priority processes is aging.
- Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.
- For example,
  If priorities range from 127 (low) to 0 (high), we could increase the priority of a 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.

ROUND ROBBIN ALGORITHM

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)



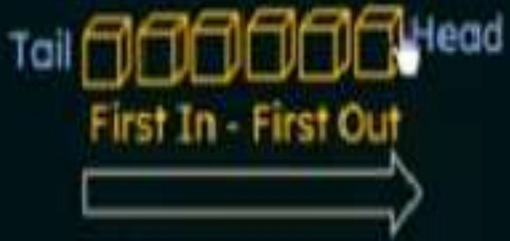
The Ready Queue

- 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

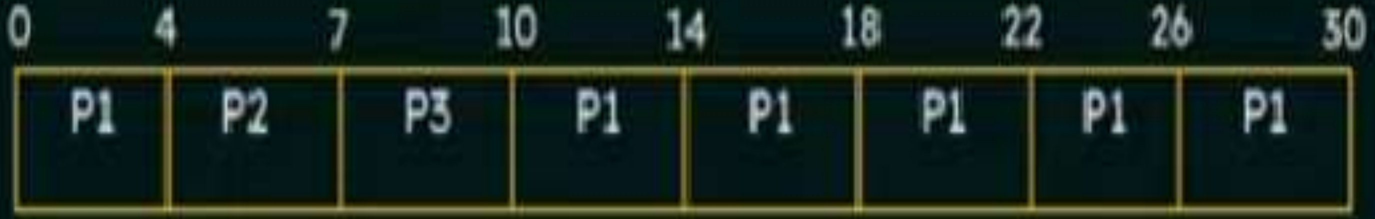
Round-Robin Scheduling
(Turnaround Time and Waiting Time)

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:



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

Average Turn Around time

= (30 + 7 + 10) / 3

= 47 / 3 = 15.66 ms

Average waiting time

= (6 + 4 + 7) / 3

= 17 / 3 = 5.66 ms

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

Average waiting time

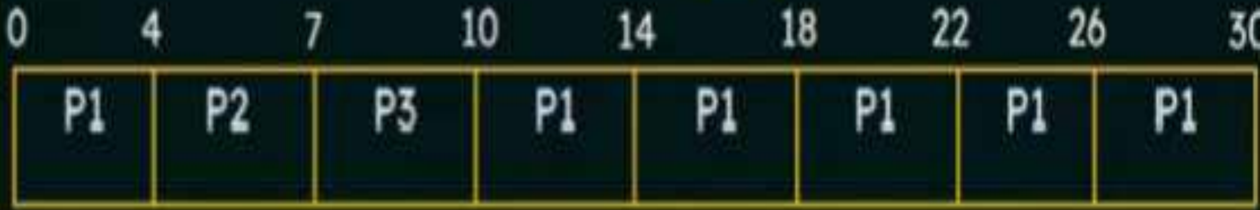
= (6 + 4 + 7) / 3

= 17 / 3 = 5.66 ms

Process ID	Burst Time
P1	24
P2	3
P3	3

Gantt Chart:

Time Quantum 4 ms



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