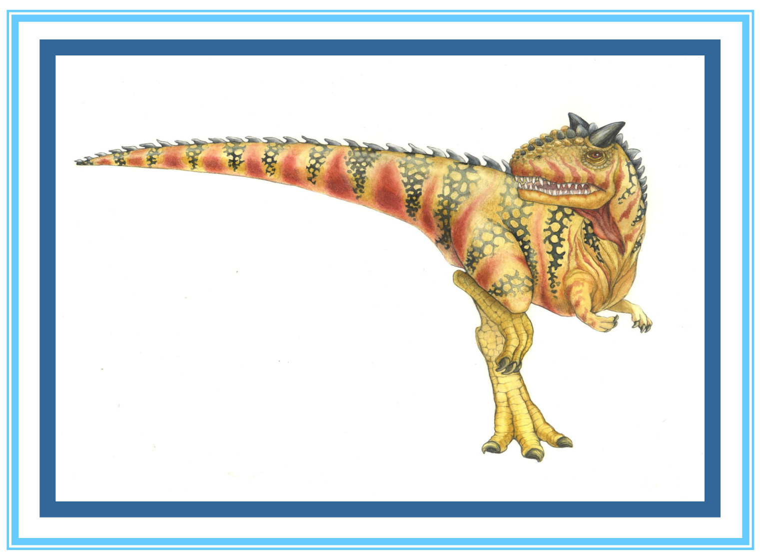
Chapter 6: CPU Scheduling

NARZU TARANNUM(NAT)



Chapter 6: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Examples

Basic Concepts behind CPU scheduling

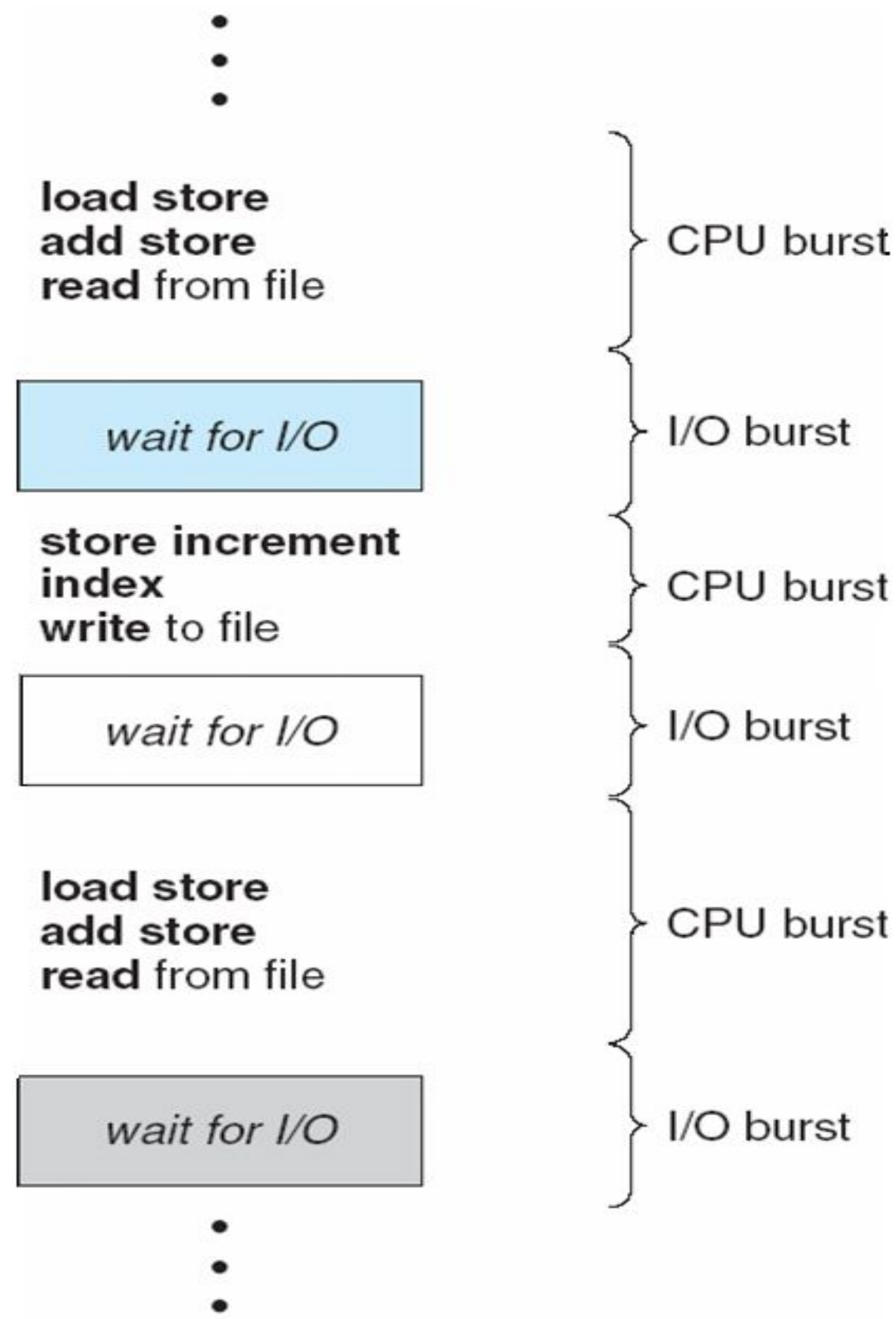
- CPU scheduling is the basis for multi-programmed operating systems.
- In Multi-programmed system, a process is executed until it must wait, typically for the completion of some I/O request.
- The objective of multi-programming is to have some process running at all times, to maximize CPU utilization.
- With multi-programming, several processes are kept in the 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
- To introduce CPU scheduling, here we describe various CPU-scheduling algorithms

Basic Concepts

CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait

- Almost all processes alternate between two states in a continuing ***cycle***, as shown in Figure below ***:***
  - A CPU burst of performing calculations, and
  - An I/O burst, waiting for data transfer in or out of the system.

- Processes alternate back and forth between this two states.
- CPU burst distribution is of main concern

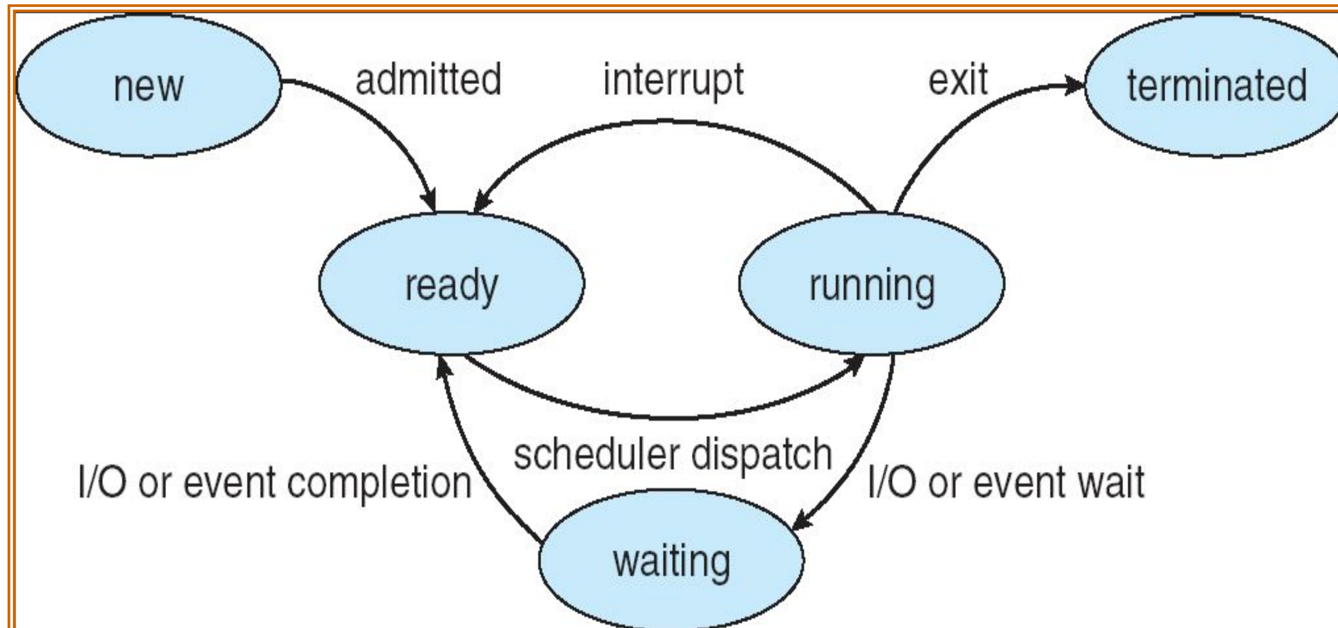


Preemptive and Non preemptive scheduling

- **Non-preemptive** – Once CPU given to a process it cannot be preempted until completes its CPU burst or finish its work.
  - The process is not interrupted until its life cycle is complete.
- **preemptive** – This term comes from the ability to remove a running process from CPU and allow another process to run in CPU.
  - The process can be interrupted, even before the completion.
  - Preempted process moves to ready queue/
  - **Preemptive Scheduling** is a CPU scheduling technique that works by dividing time slots of CPU to a given process. The time slot given might be able to complete the whole process or might not be able to it.
  - Algorithms that are backed by preemptive Scheduling are round-robin (RR), priority, SRTF (shortest remaining time first).

CPU Scheduler

- Short-term schedular selects from among the processes in ready queue, and allocates the CPU to one of them for execution.
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running state to waiting state
  2. Switches from running state to ready state
  3. Switches from waiting state to ready state
  4. Terminates
- Scheduling under 1 and 4 is **non-preemptive.**
- All other scheduling is **preemptive.**



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)
- **Fairness**-Give each process a fair share of CPU.

Scheduling Algorithm Optimization Criteria

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

First-Come, First-Served (FCFS) Scheduling Algorithm

- The process that requests the CPU first is allocated CPU first
- Implementation is easily managed with a FIFO queue
- The FCFS scheduling algorithm is non-preemptive
  - Once the CPU has been allocated to a process, that process keeps the CPU until it release the CPU, either by terminating or by requesting I/O.

- BT=Burst Time
- CT=Completion Time
- AT= Arrival Time
- RS=Response Time
- WT= Waiting Time
- TA= Turn Around time
- WT= RS-AT / TA-BT
- TA= CT-AT
- **Response time = Time at which the process gets the CPU for the first time - Arrival time**

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: $(0 + 24 + 27)/3 = 17$

In this case average waiting time is quite long

FCFS Scheduling (Cont.)

Process	Burst Time
P_1	24
P_2	3
P_3	3

Suppose that the processes arrive in the order: P_2 , P_3 , P_1

- The Gantt chart for the schedule is:

P_2	P_3	P_1

0 3 6 30

- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- The average waiting time is much better than the previous case; this reduction is substantial.
- Thus the average waiting time under an FCFS policy is generally not minimal and may vary substantially if the processes CPU burst time vary greatly.
- **Convoy effect** - short process behind long process
- It would be disadvantageous to allow one process to keep the CPU for an extended period.

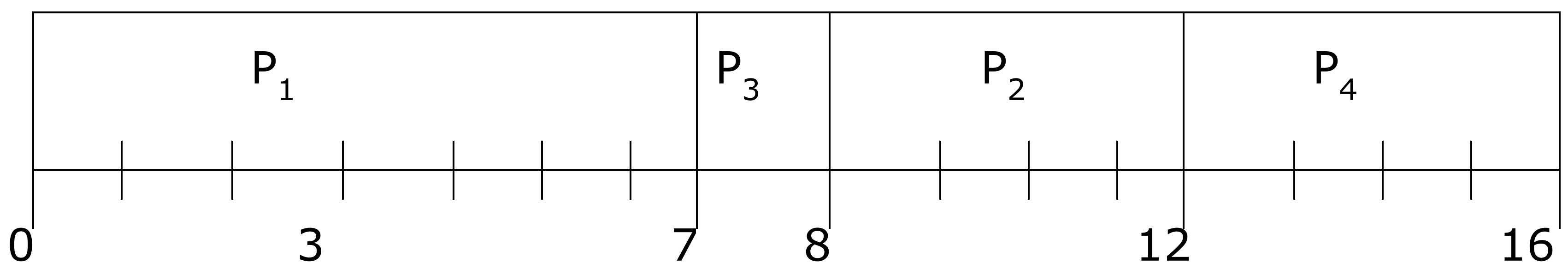
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst.
  - Use these lengths to schedule the process with the shortest time
  - **preemptive** – This term comes from the ability to remove a running process from CPU and allow another process to run in CPU.
  - 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
- The difficulty is knowing the length of the next CPU request

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

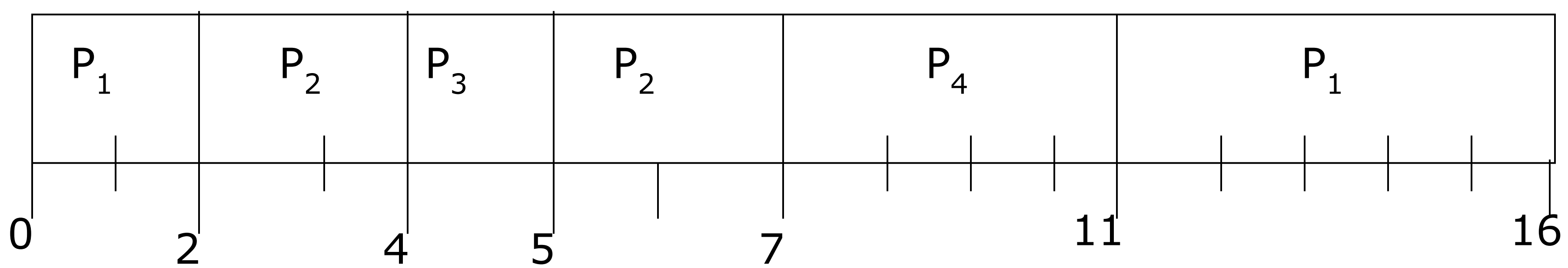


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)

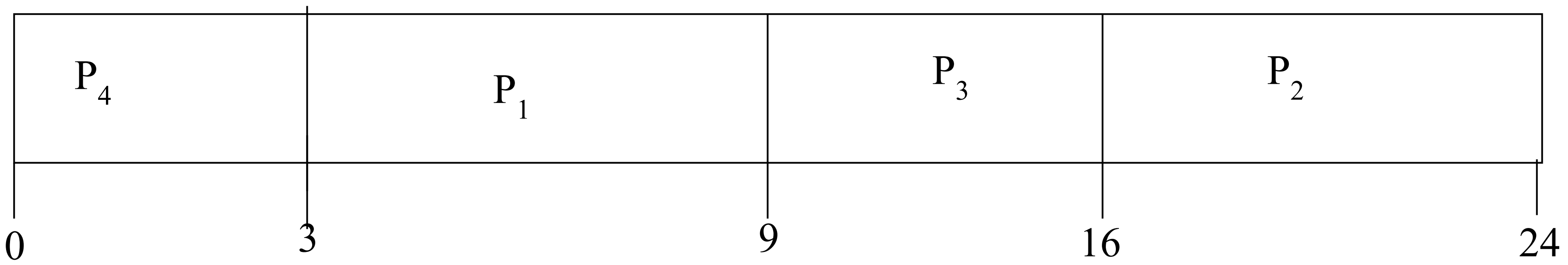


- Average waiting time = (9 + 1 + 0 +2)/4 = 3

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF/SRTF Gantt Chart:

P_1	P_2	P_4	P_1	P_3

0 1 5 10 17 26

- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

Example of SJF(Preemptive)

Processes	Arrival time	CPU burst
P1	2	10
P2	4	6
P3	6	23
P4	8	8
P5	10	30
P6	12	3
P7	14	18

Homework

- Draw a Gantt chart and find thruput, average waiting time, average turn around time, number of context switch using SRTF

Processes	Arrival time	CPU burst
P1	18	40
P2	29	17
P3	0	28
P4	21	37
P5	12	31

Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst

- Can be done by using the length of previous CPU bursts, using exponential average

$$1.\ t_n = \mathit{Most}\ \text{recent CPU burst}$$

$$2.\ \tau_n = \mathit{Past}\ \text{CPU burst of a process}$$

$$2.\ \tau_{n+1} = \text{predicted value for the next CPU burst}$$

$$3.\ \alpha,\ 0 \le \alpha \le 1$$

$$4.\ \text{Define}: \tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$$

- Commonly, α set to ½
- Preemptive version called **shortest-remaining-time-first**

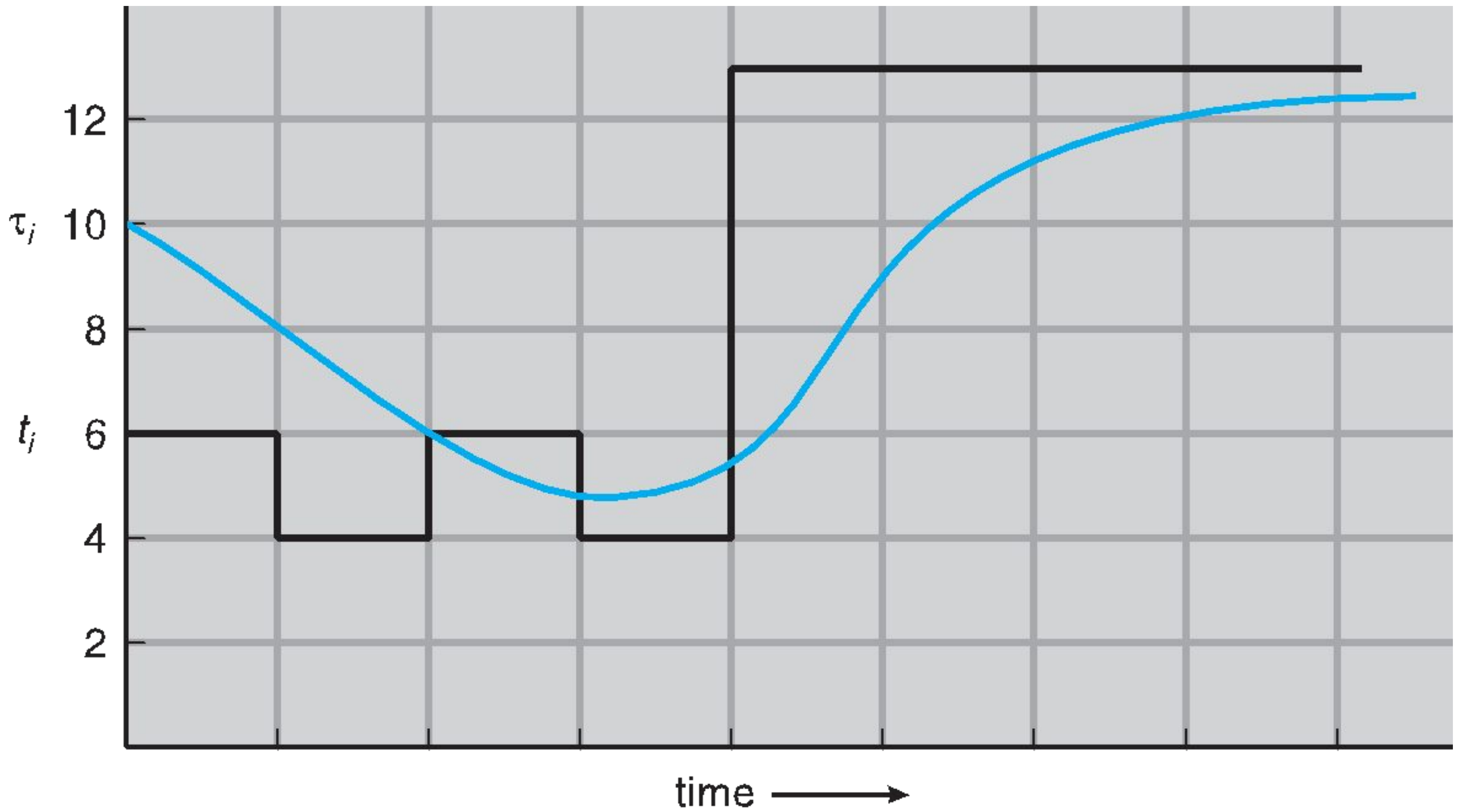
Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does no effect
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the most recent CPU burst matters
- If we expand the formula, we get:

$$\begin{aligned}\tau_{n+1} = {} & \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots \\ & + (1 - \alpha)^j \alpha\, t_{n-j} + \ldots \\ & + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor
- Commonly, if α is set to ½, so recent history and past history are equally weighted.
- Next slide figure shows an exponential average with $\alpha = .5$ and $\tau_o = 10$

Prediction of the Length of the Next CPU Burst



CPU burst (t_i)		6	4	6	4	13	13	13 …
"guess" (τ_i)	10	8	6	6	5	9	11	12 …

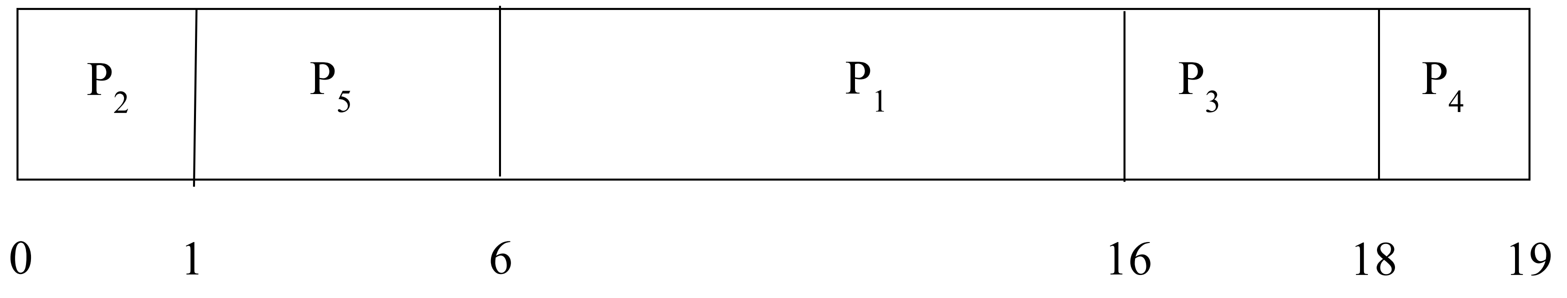
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
- Priority scheduling can be either Preemptive or Non-preemptive
- Equal-priority processes are schedule in FCFS order.
- 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.
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process that wait in the system for a long time.

Example of Priority Scheduling(Non-preemptive)

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

Example of Priority Scheduling(preemptive)

- Consider the set of processes with arrival time (in milli second), CPU burst time, and priority (0 is the height priority) shown bellow. None of the process have I/O burst time. The average waiting time of all process using preemptive priority scheduling algorithm is ___________.

Processes	Arrival time	Priority	CPU burst
P1	0	2	11
P2	5	0	28
P3	12	3	2
P4	2	1	10
P5	9	4	16

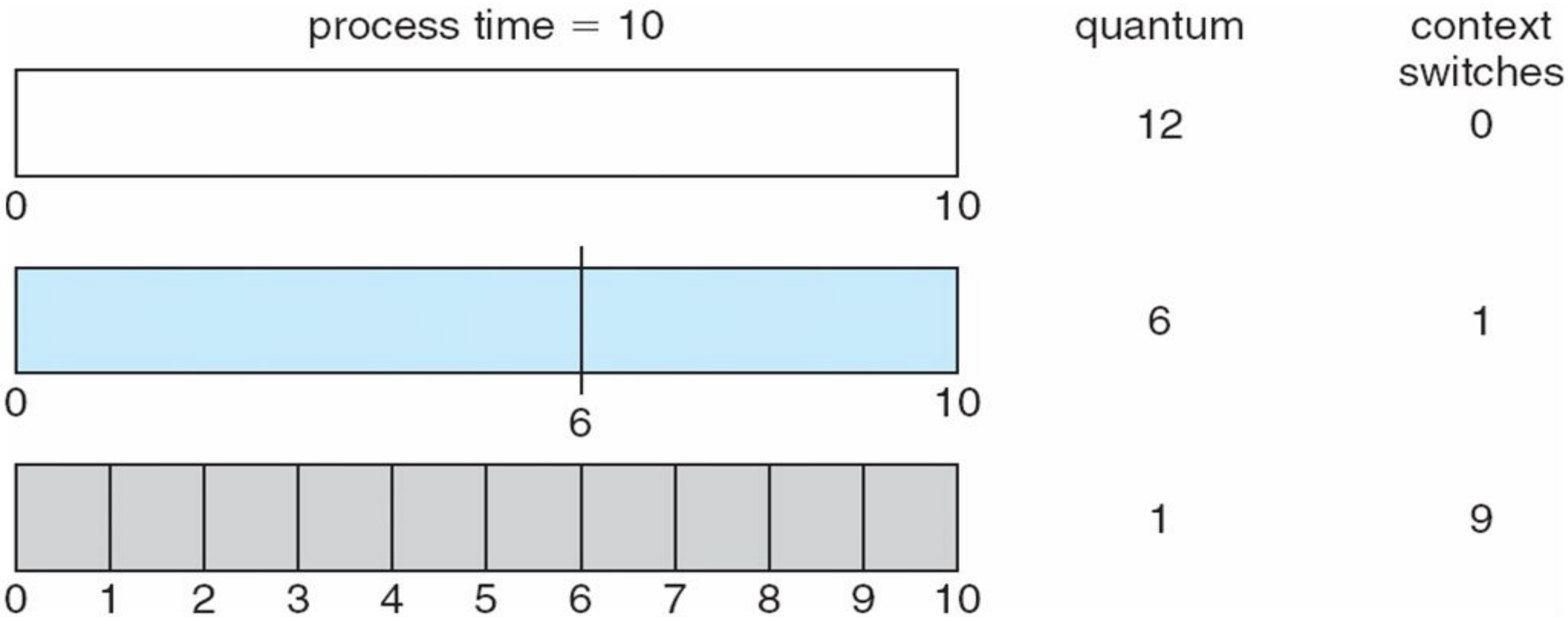
- (A) 29, (B) 30, (C) 31, (D) 32

Processes	Arrival time	Priority	CPU burst
P1	0	2	11
P2	5	0	28
P3	12	3	2
P4	2	1	10
P5	9	4	16

Round Robin (RR)

- The round-robin (RR) scheduling algorithm is designed especially for time sharing systems.
- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- Each process gets a small unit of CPU time (**time quantum** q), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- Timer interrupts every quantum to schedule next process.
- **Performance**
  - The average waiting time under the round-robin policy is often long, it depends on the size of time quantum
  - If the time quantum is large, RR policy will become same as FCFS
  - If the time quantum is extremely small, RR approach can result in a large number of context switches. We can see it in next slide
  - Typically, higher average turnaround than SJF, but better *response time.*
  - q should be large compared to context switch time.

Time Quantum and Context Switch Time



Round Robin (RR)

- Advantages – Every process gets an equal share of the CPU. RR is cyclic in nature, so there is no starvation.
- Disadvantages – Setting the quantum too short, increases the overhead and lowers the CPU efficiency, but setting it too long may cause poor response to short processes.

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

Example of RR with Time Quantum =2ms

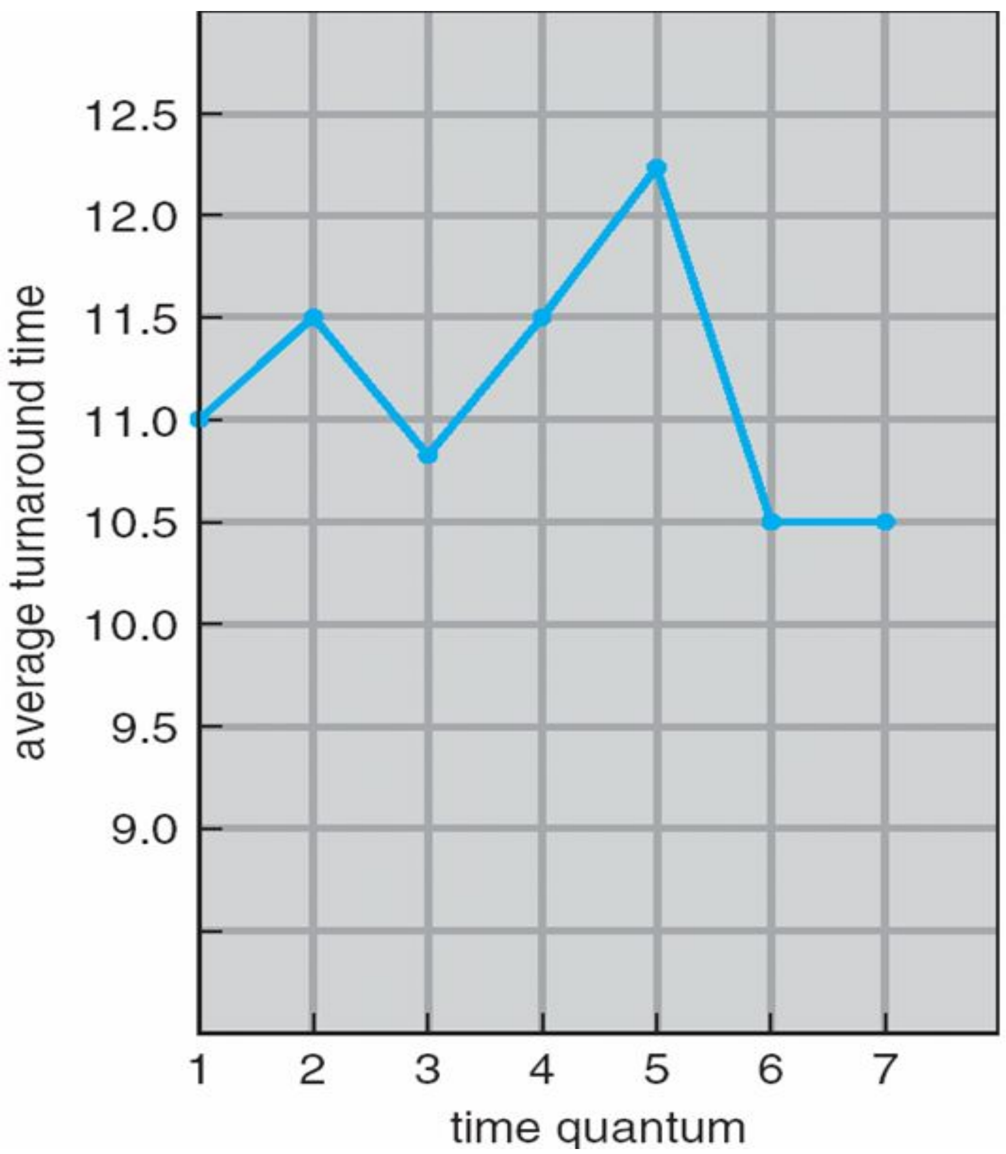
Process ID	Arrival Time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

RR Homework with time quantum=7ms

- Consider the set of processes with arrival time (in milli second) and CPU burst time shown bellow. Calculate average waiting time, average turn around time, throughput and number of context switch.

Process ID	Arrival Time	Burst time
P1	12	20
P2	9	17
P3	1	28
P4	7	23
P5	21	13

Turnaround Time Varies With The Time Quantum



- Turnaround time also depends on the size of the time quantum.
- We can see here in this figure the average turnaround time of a set of processes does not necessarily improve as the time quantum size increases.

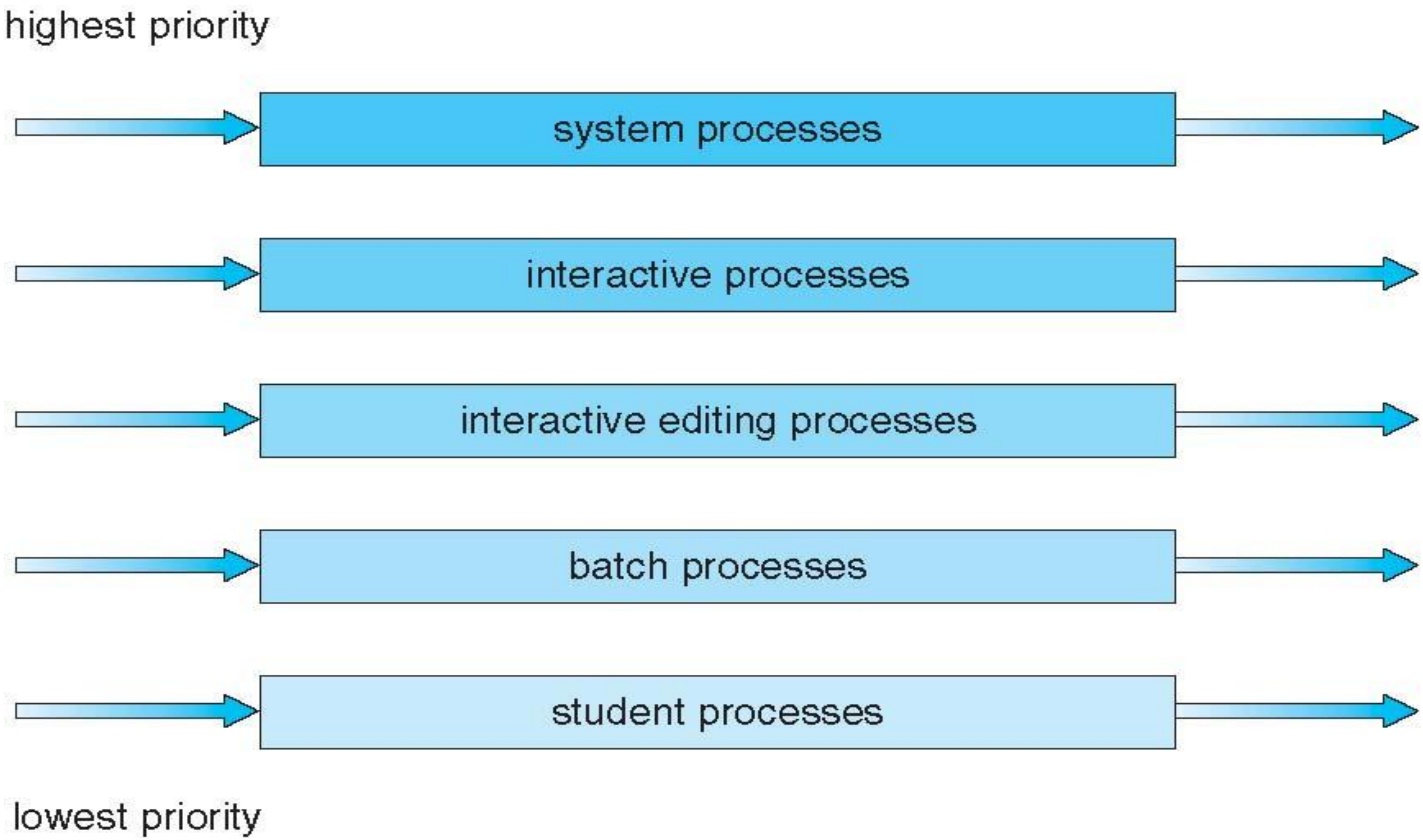
Multilevel Queue

- Another class of scheduling algorithm needs- in which processes are classified into different groups-e.x.:
  - foreground (interactive) processes
  - background (batch) processes
- They have different response time requirements-so different scheduling needs.
- Foreground processes may have priority over background processes.

- **A multilevel queue-scheduling algorithm partitions the ready queue into several separate queues**-
- we can see it in the figure of next slide:-
- The processes are permanently assigned to one queue.
- Each queue has its own scheduling algorithm:
  - Foreground queue scheduled by – RR algorithm
  - Background queue scheduled by – FCFS algorithm

- Scheduling must be done between the queues or there must be scheduling among the queue.
  - Which is commonly implemented as fixed priority preemptive scheduling; (i.e., serve all from foreground then from background).
  - Possibility of starvation.

Multilevel Queue Scheduling



Multilevel Feedback Queue scheduling

- In multi-level queue processes do not move from one queue to the other----But in
- Multilevel Feedback Queue scheduling, allows a process to move between queues.
- If a process uses too much CPU time, it will be moved to a lower priority queue.
- Similarly, if 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.

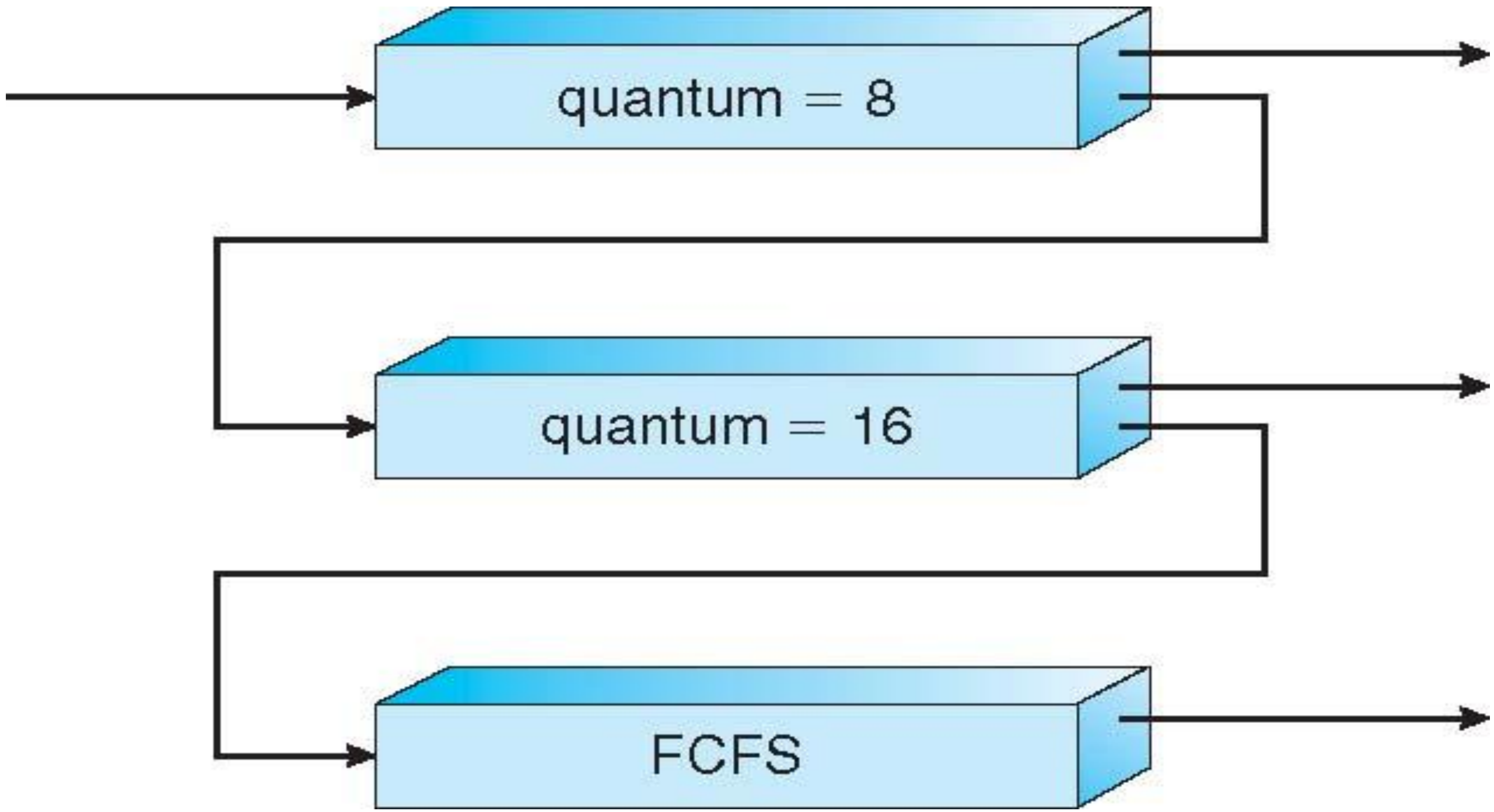
- Multilevel-feedback-queue scheduler defined by the following parameters:
    - number of queues
    - scheduling algorithms for each queue
    - method used to determine when to upgrade a process
    - method used to determine when to demote a process
    - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues: (can see the figure in next slide)
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS

- Scheduling
  - A new job enters queue Q_0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2

Multilevel Feedback Queues



End of Chapter 6