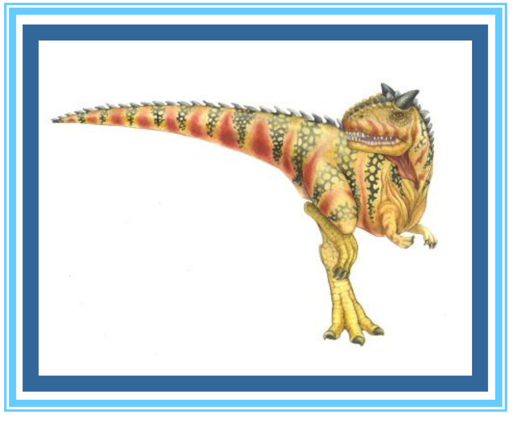
Module 3.2:
CPU Scheduling



Silberschatz, Galvin and Gagne ©2013

CPU Scheduling

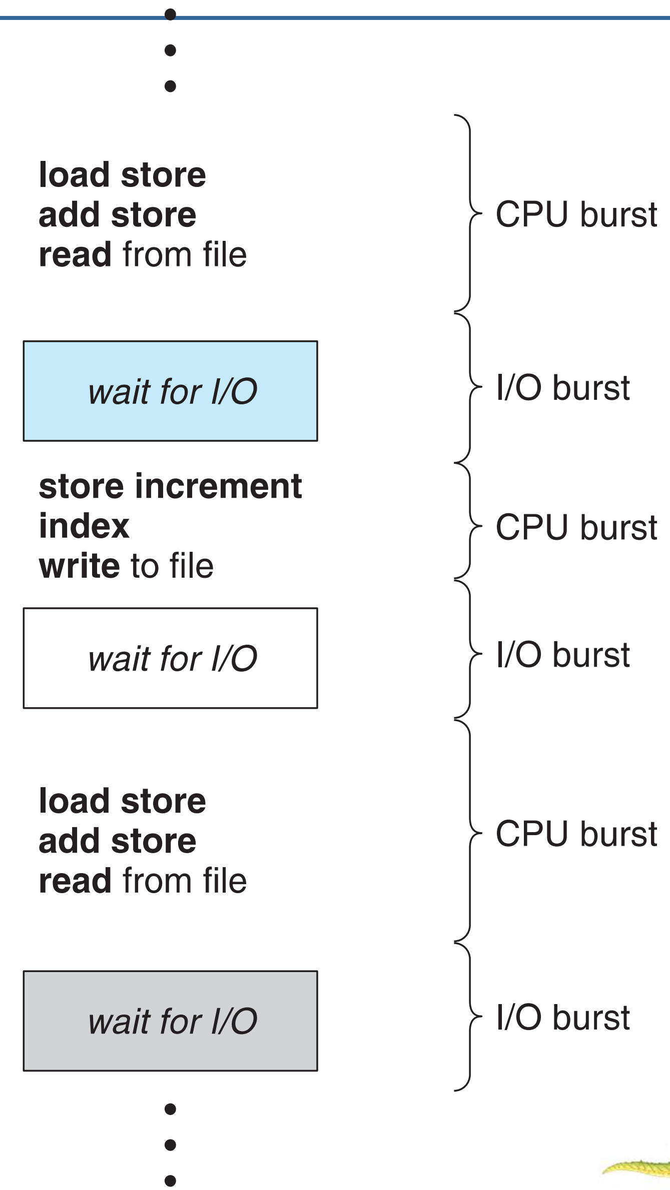
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Real-Time CPU Scheduling
- Operating Systems Examples

Objectives

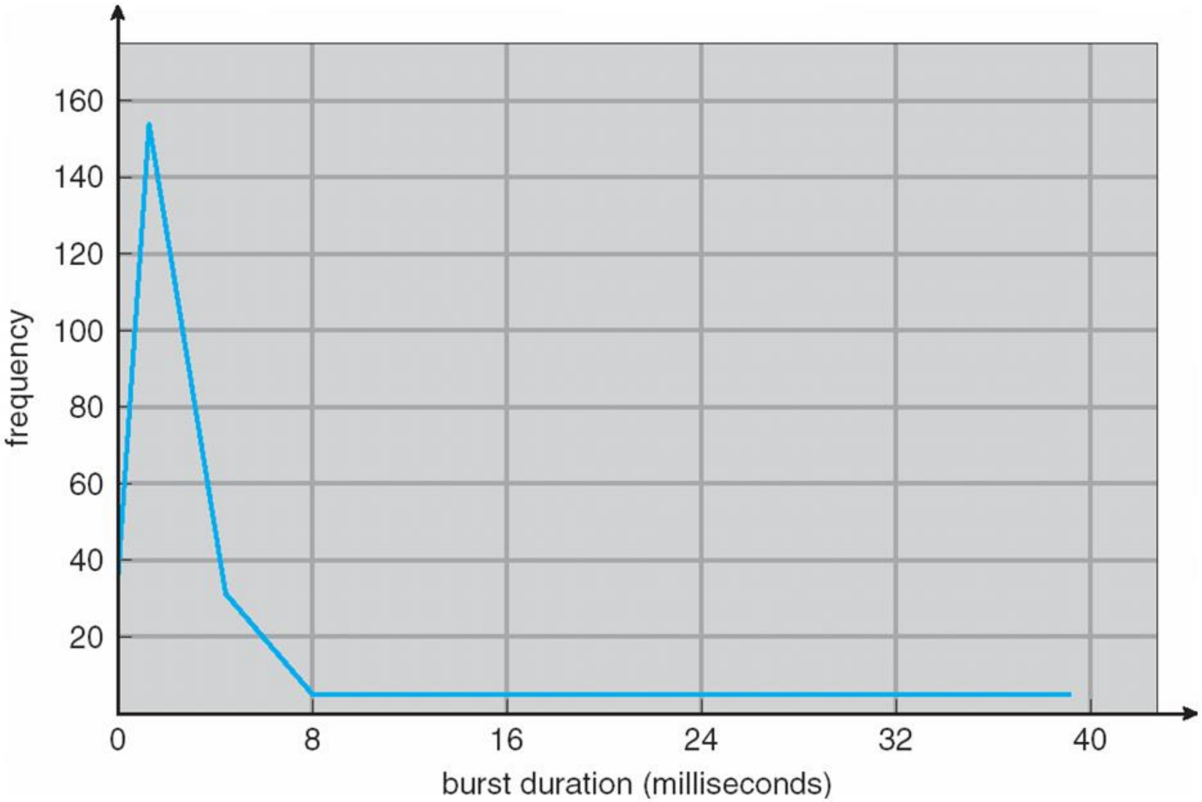
- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
- To examine the scheduling algorithms of several operating systems

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** followed by **I/O burst**
- CPU burst distribution is of main concern



Histogram of CPU-burst Times

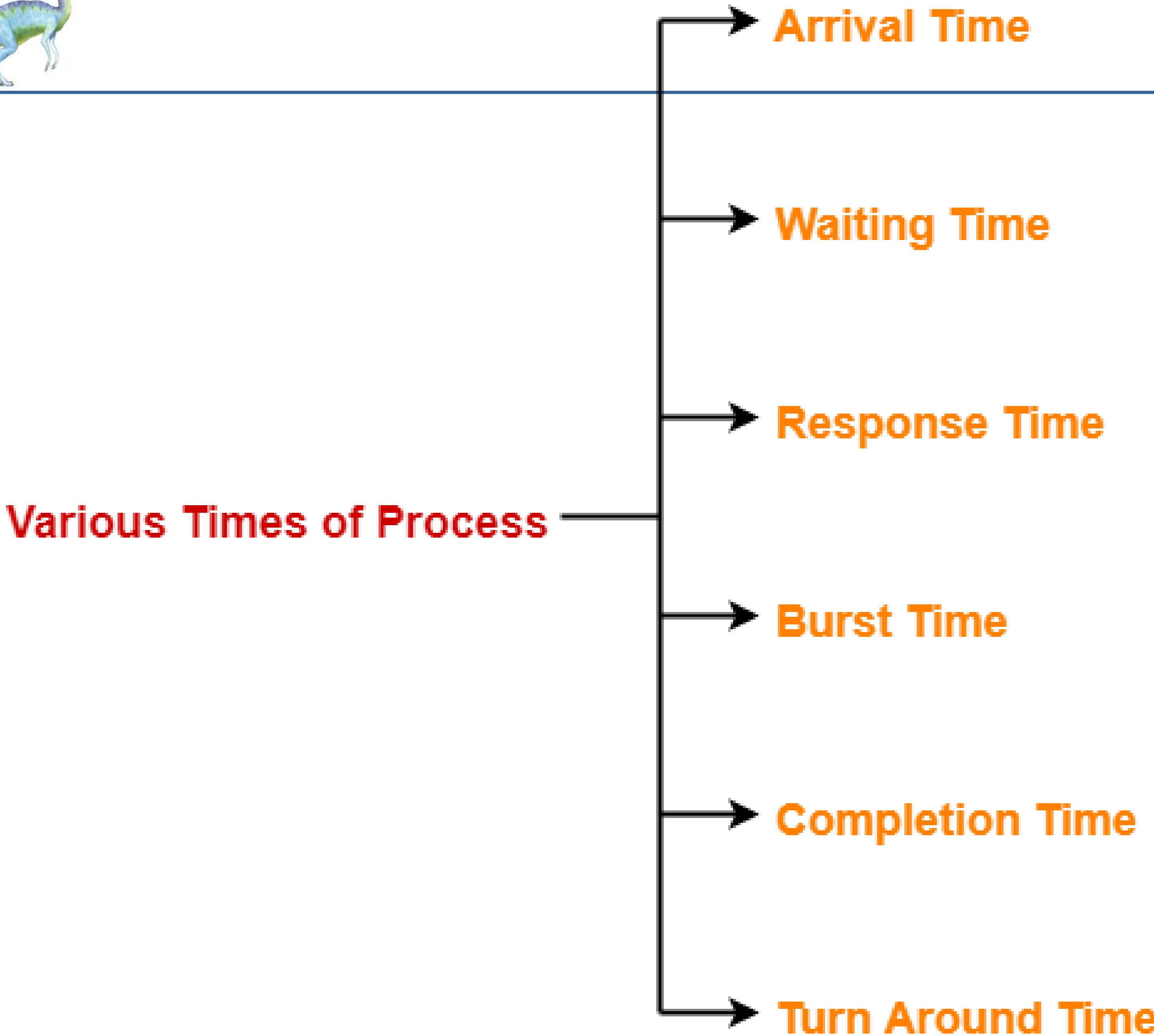


CPU Scheduler

- **Short-term 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


Various Times of Process
Arrival Time
Waiting Time
Response Time
Burst Time
Completion Time
Turn Around Time

Important CPU scheduling Terminologies

- **Burst Time/Execution Time**: It is a time required by the process to complete execution. It is also called running time.
- **Arrival Time**: when a process enters in a ready state
- **Finish/Completion Time**: when process complete and exit from a system
- **Multiprogramming**: A number of programs which can be present in memory at the same time.
- **Jobs**: It is a type of program without any kind of user interaction.
- **Process:** It is the reference that is used for both job and user.
- **CPU/IO burst cycle**: Characterizes process execution, which alternates between CPU and I/O activity. CPU times are usually shorter than the time of I/O.

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

Turn Around time = Completion time – Arrival time

Turn Around time = Burst time + Waiting time

- **Waiting time** – amount of time a process has been waiting in the ready queue

Waiting time = Turn Around time – Burst time

- **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)

Response Time = Time at which process first gets the CPU – Arrival time

Scheduling Algorithm Optimization Criteria

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

First- Come, First-Served (FCFS) Scheduling

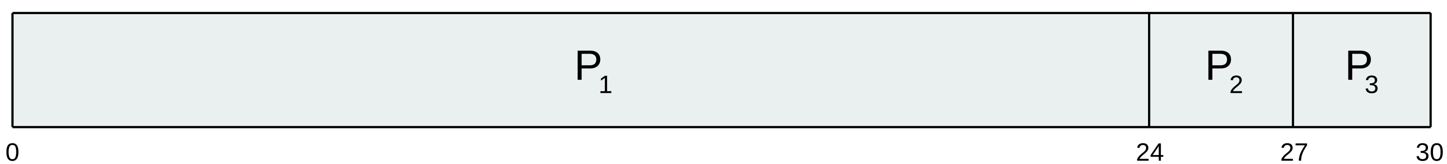
- The process which arrives first in the ready queue is firstly assigned the CPU.
- In case of a tie, process with smaller process id is executed first.
- It is always non-preemptive in nature.

- **Disadvantages-**
- It does not consider the priority or burst time of the processes.
- It suffers from **convoy effect**

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

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:

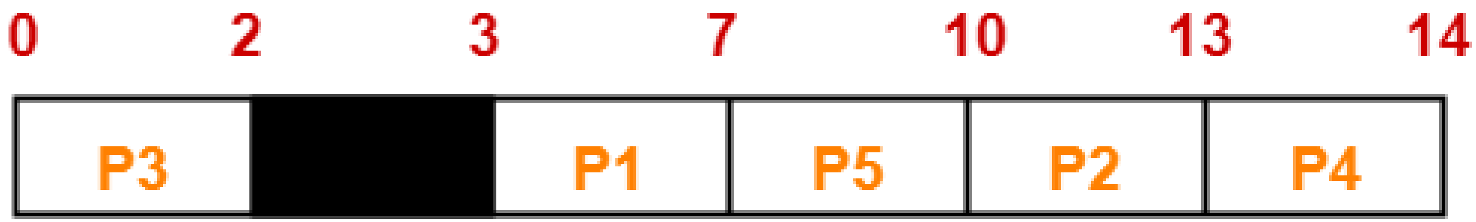
- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

FCFS SCHEDULING

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

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

If the CPU scheduling policy is FCFS, calculate the average waiting time and average turn around time.



Gantt Chart

FCFS Solution

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

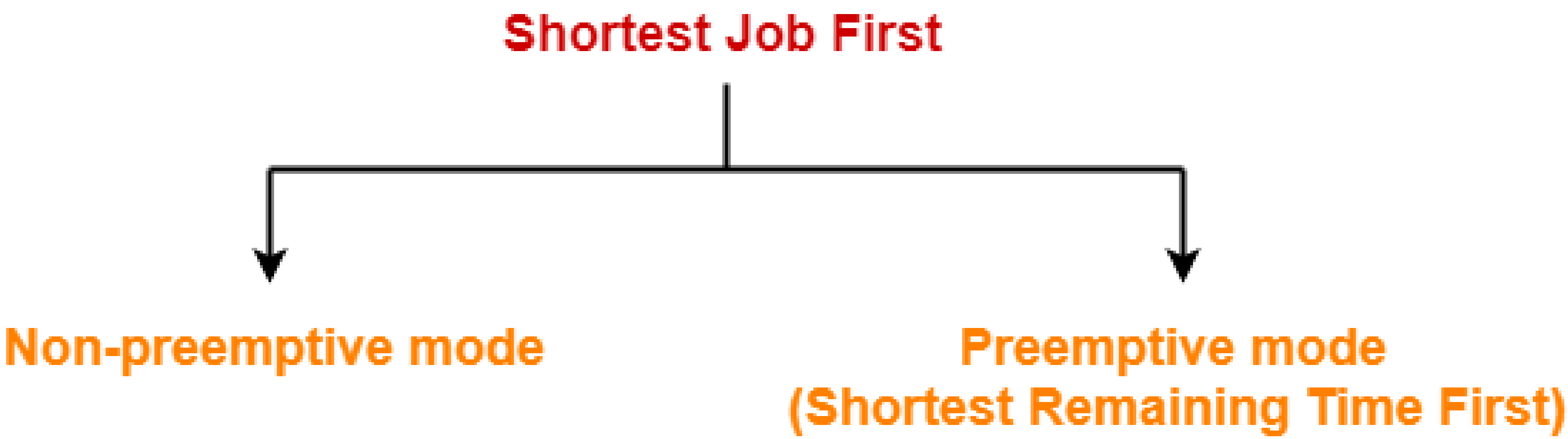
Process Id	Exit time	Turn Around time	Waiting time
P1	7	7 – 3 = 4	4 – 4 = 0
P2	13	13 – 5 = 8	8 – 3 = 5
P3	2	2 – 0 = 2	2 – 2 = 0
P4	14	14 – 5 = 9	9 – 1 = 8
P5	10	10 – 4 = 6	6 – 3 = 3

- Average Turn Around time = (4 + 8 + 2 + 9 + 6) / 5 = 29 / 5 = 5.8 unit
- Average waiting time = (0 + 5 + 0 + 8 + 3) / 5 = 16 / 5 = 3.2 unit

SJF

In SJF Scheduling,

- Out of all the available processes, CPU is assigned to the process having smallest burst time.
- In case of a tie, it is broken by FCFS Scheduling.



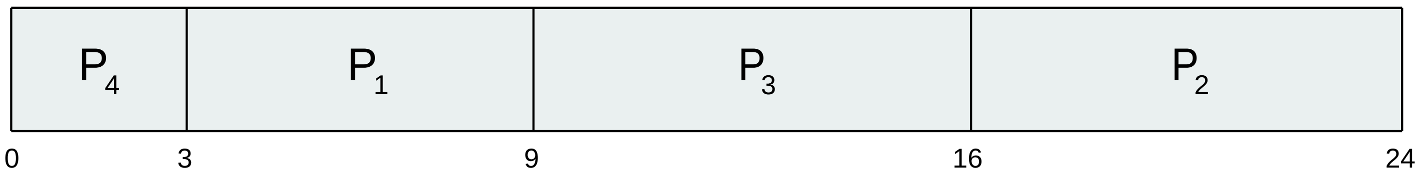
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- 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
  - Could ask the user

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



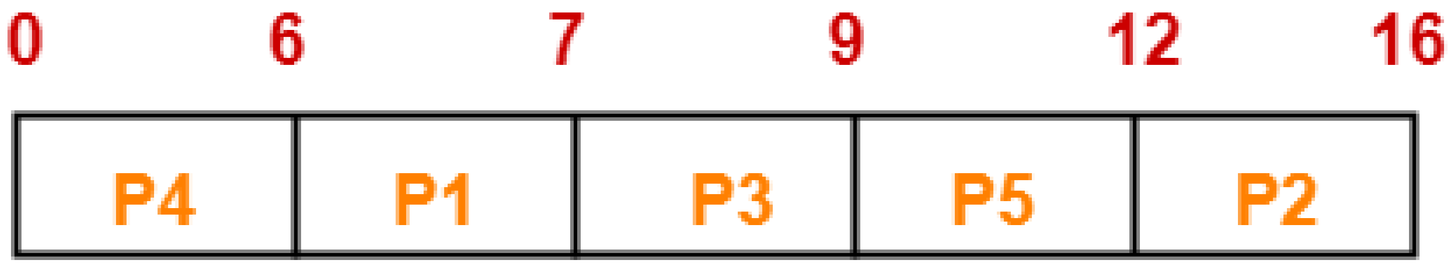
- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

SJF Problem

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

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

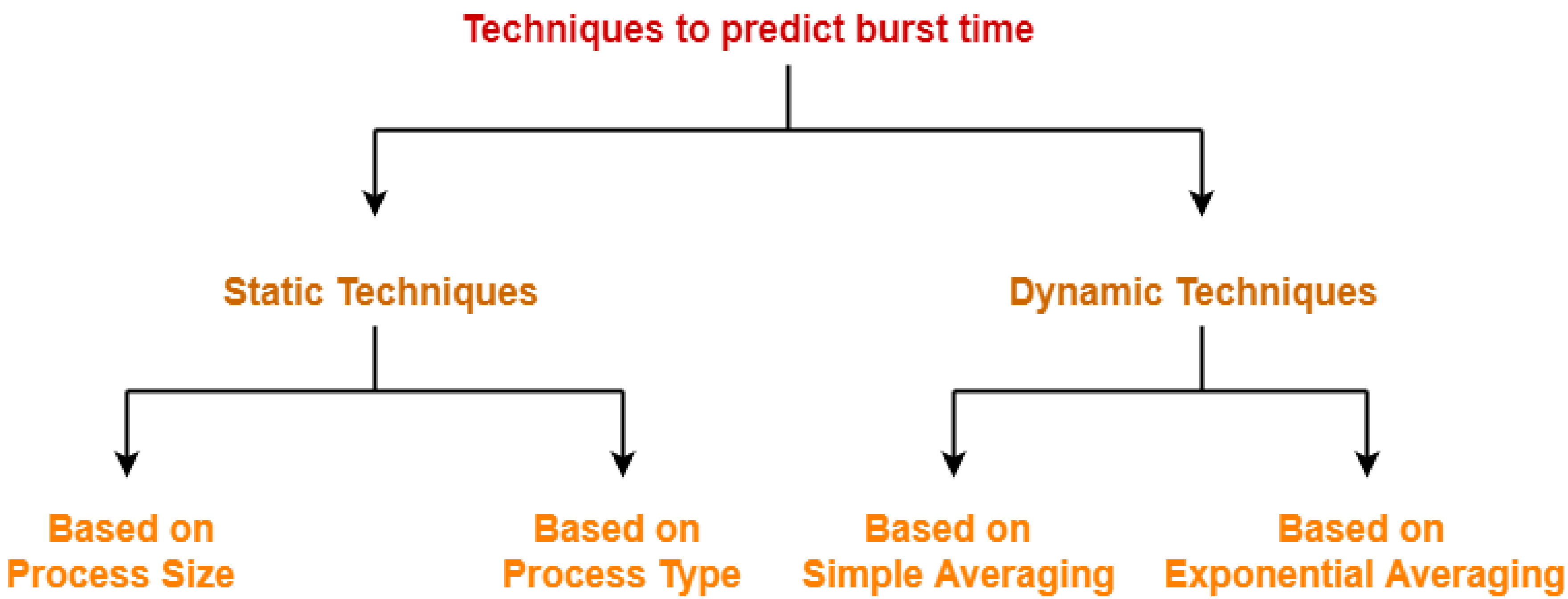
Gantt Chart



Gantt Chart

Process Id	Exit time	Turn Around time	Waiting time
P1	7	7 – 3 = 4	4 – 1 = 3
P2	16	16 – 1 = 15	15 – 4 = 11
P3	9	9 – 4 = 5	5 – 2 = 3
P4	6	6 – 0 = 6	6 – 6 = 0
P5	12	12 – 2 = 10	10 – 3 = 7

- **Average Turn Around time = (4 + 15 + 5 + 6 + 10) / 5 = 40 / 5 = 8 unit**
- **Average waiting time = (3 + 11 + 3 + 0 + 7) / 5 = 24 / 5 = 4.8 unit**


Techniques to predict burst time
Static Techniques
Dynamic Techniques
Based on Process Size
Based on Process Type
Based on Simple Averaging
Based on Exponential Averaging

Static Technique

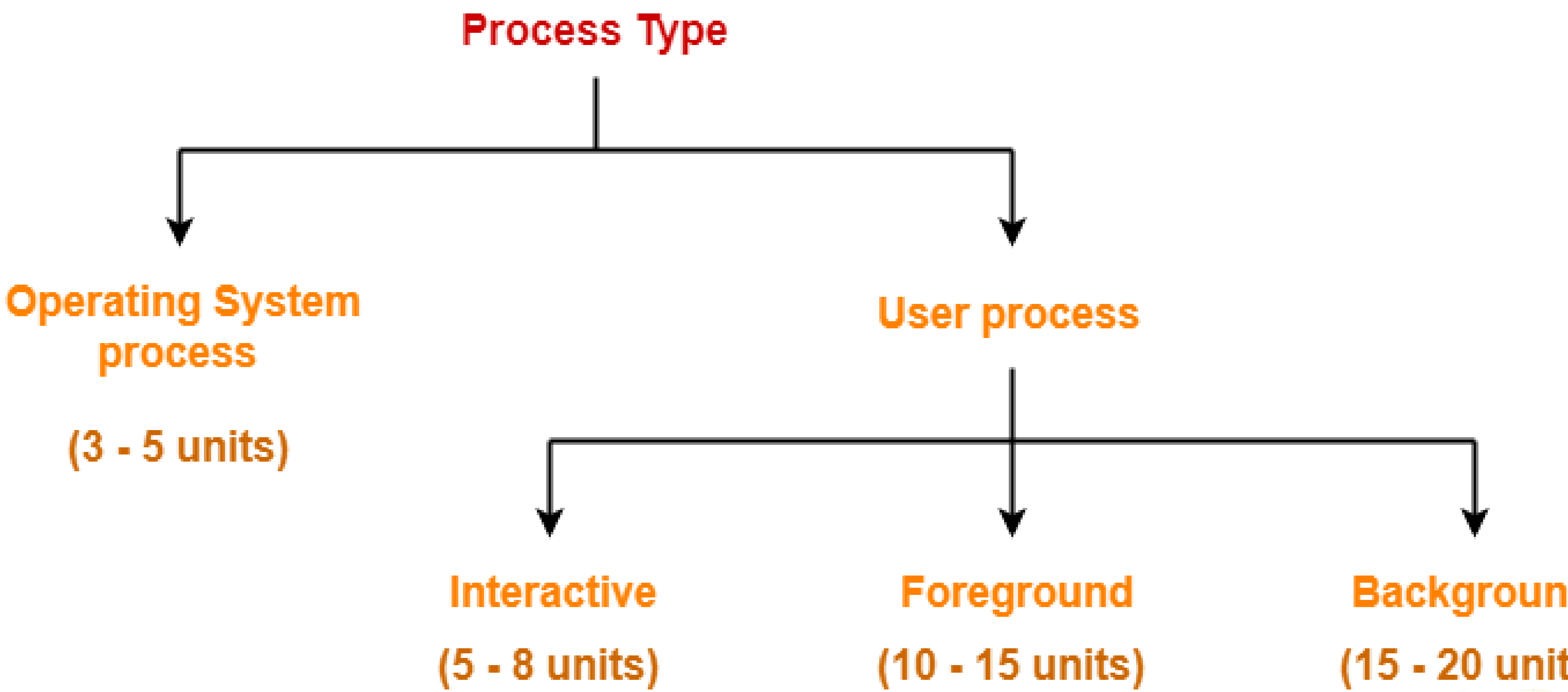
1. Based on Process Size-

- This technique predicts the burst time for a process based on its size.
- Burst time of the already executed process of similar size is taken as the burst time for the process to be executed.
- Example-
- Consider a process of size 200 KB took 20 units of time to complete its execution.
- Then, burst time for any future process having size around 200 KB can be taken as 20 units.
- *NOTE*
  - *The predicted burst time may not always be right.*
  - *This is because the burst time of a process also depends on what kind of a process it is.*

Static Technique

2. Based on Process Type-

- This technique predicts the burst time for a process based on its type.



Dynamic Technique

- There are two dynamic techniques-

1. Based on simple averaging

2. Based on exponential averaging

Dynamic Technique

1. Based on Simple Averaging-

- Burst time for the process to be executed is taken as the average of all the processes that are executed till now.

- Given n processes $P_1, P_2, \ldots, P_n$ and burst time of each process P_i as t_i, then predicted burst time for process P_{n+1} is given as-

$$\boxed{T_{n+1} = \frac{1}{n} \sum_{i=1}^{n} t_i}$$

Determining Length of Next CPU Burst

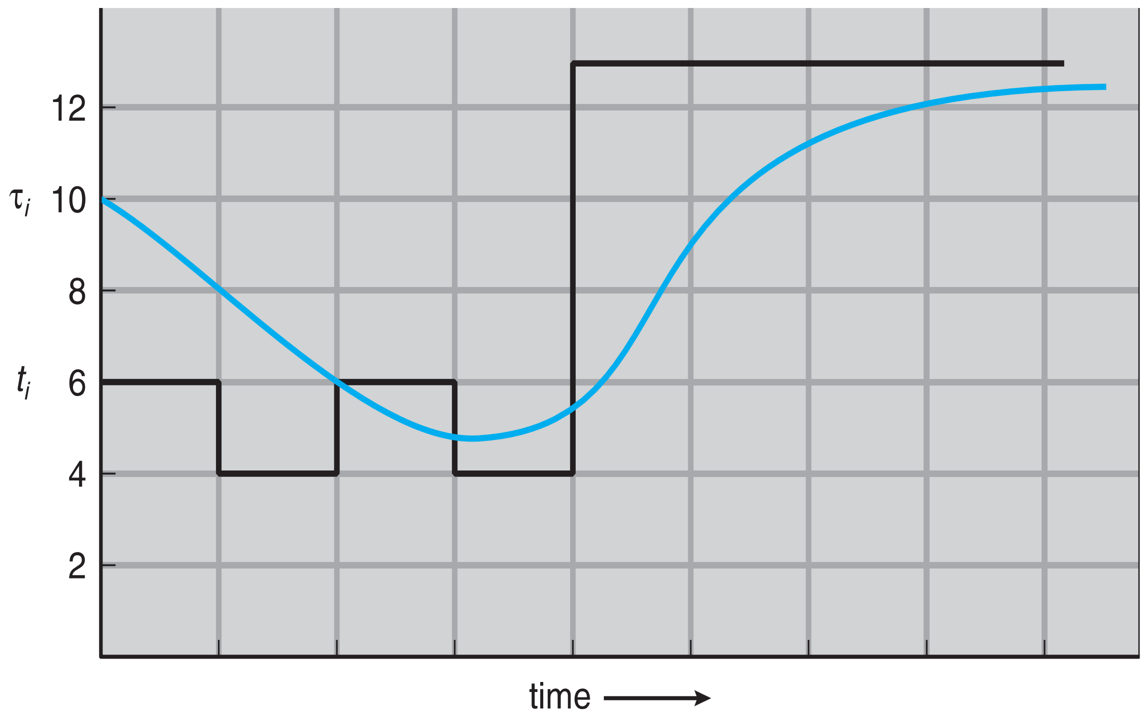
- 2. Based on exponential averaging

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

$$\boxed{T_{n+1} = \alpha\, t_n + (1 - \alpha)\, T_n}$$

- where-
  - α is called smoothening factor (0<= α <=1)
  - t_n = actual burst time of process P_n
  - T_n = Predicted burst time for process P_n
- Commonly, α set to ½

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	…

Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

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

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

PRACTICE PROBLEM BASED ON PREDICTING BURST TIME-

- Calculate the predicted burst time using exponential averaging for the fifth process if the predicted burst time for the first process is 10 units and actual burst time of the first four processes is 4, 8, 6 and 7 units respectively. Given $\alpha = 0.5$.

Solution

- Given-Predicted burst time for 1st process = 10 units
- Actual burst time of the first four processes = 4, 8, 6, 7
- α = 0.5
- Predicted Burst Time for 2nd Process-
- = α x Actual burst time of 1st process + (1-α) x Predicted burst time for 1st process
- = 0.5 x 4 + 0.5 x 10
- = 2 + 5
- = 7 units

Solution

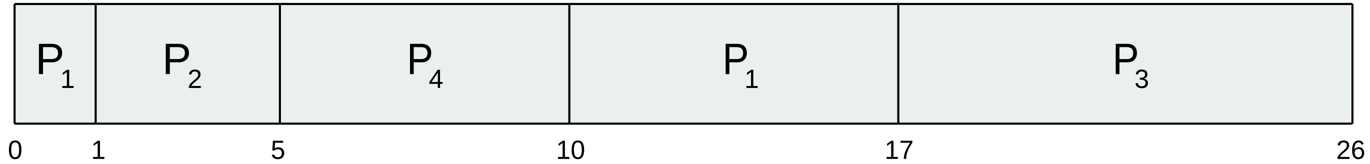
- Predicted burst time for 3rd process
- = α x Actual burst time of 2nd process + (1-α) x Predicted burst time for 2nd process
- = 0.5 x 8 + 0.5 x 7
- = 4 + 3.5
- = 7.5 units
- Predicted burst time for 4th process = 6.75 units
- 5th process = 6.875 units

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 Gantt Chart



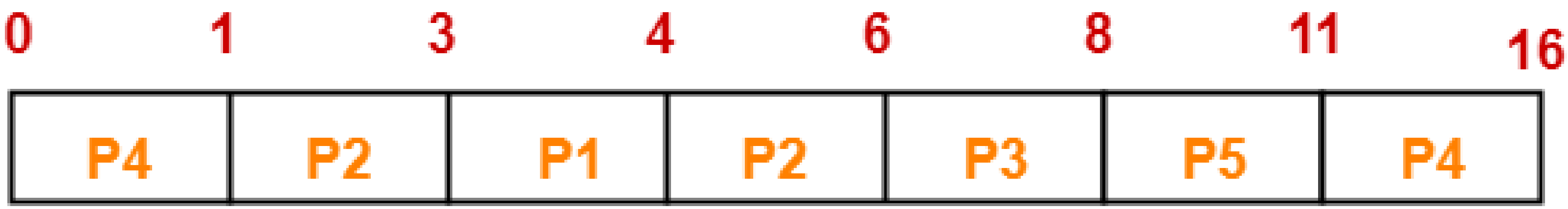
- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

SRTF Problem

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

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

SRTF



Gantt Chart

SRTF

Process Id	Exit time	Turn Around time	Waiting time
P1	4	4 – 3 = 1	1 – 1 = 0
P2	6	6 – 1 = 5	5 – 4 = 1
P3	8	8 – 4 = 4	4 – 2 = 2
P4	16	16 – 0 = 16	16 – 6 = 10
P5	11	11 – 2 = 9	9 – 3 = 6

Now,

- Average Turn Around time = (1 + 5 + 4 + 16 + 9) / 5 = 35 / 5 = 7 unit
- Average waiting time = (0 + 1 + 2 + 10 + 6) / 5 = 19 / 5 = 3.8 unit

SRTF

- Advantages-
- SRTF is optimal and guarantees the minimum average waiting time.
- It provides a standard for other algorithms since no other algorithm performs better than it.

- Disadvantages-
- It can not be implemented practically since burst time of the processes can not be known in advance.
- It leads to starvation for processes with larger burst time.
- Priorities can not be set for the processes.
- Processes with larger burst time have poor response time.

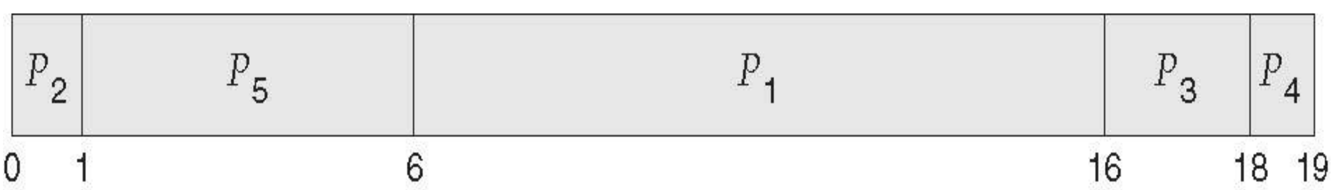
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation/Indefinite Blocking** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

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



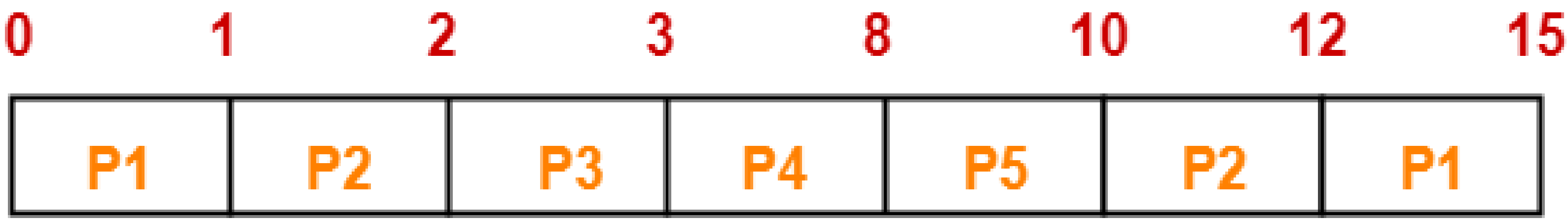
- Average waiting time = 8.2 msec

Priority (Preemptive)

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 preemptive, calculate the average waiting time and average turn around time. (Higher number represents higher priority)

Solution



Gantt Chart

Solution

Process Id	Exit time	Turn Around time	Waiting time
P1	15	15 – 0 = 15	15 – 4 = 11
P2	12	12 – 1 = 11	11 – 3 = 8
P3	3	3 – 2 = 1	1 – 1 = 0
P4	8	8 – 3 = 5	5 – 5 = 0
P5	10	10 – 4 = 6	6 – 2 = 4

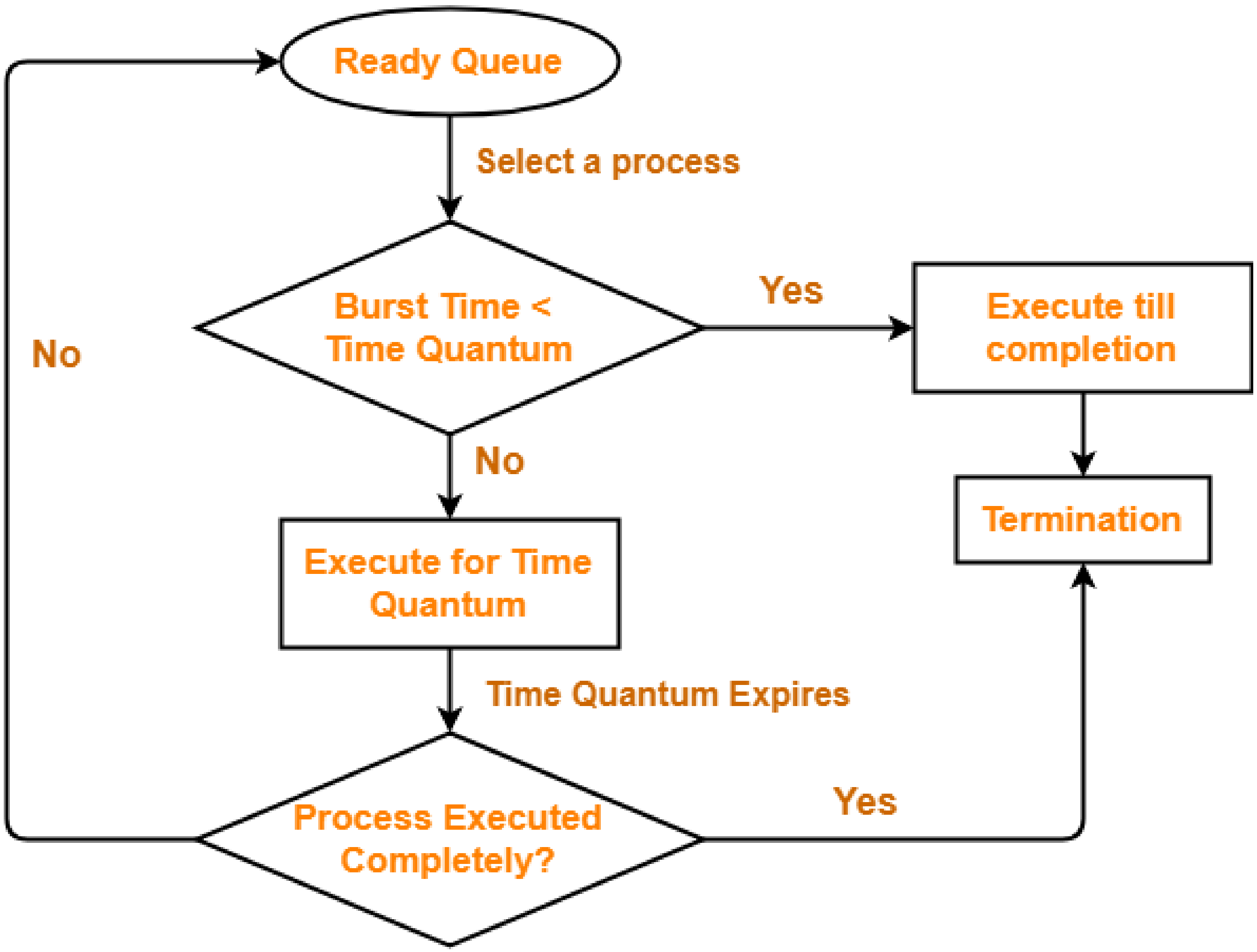
Now,

- **Average Turn Around time = (15 + 11 + 1 + 5 + 6) / 5 = 38 / 5 = 7.6 unit**
- **Average waiting time = (11 + 8 + 0 + 0 + 4) / 5 = 23 / 5 = 4.6 unit**

Round Robin (RR)

- 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.
- 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.
- Timer interrupts every quantum to schedule next process
- 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)



Round Robin Scheduling

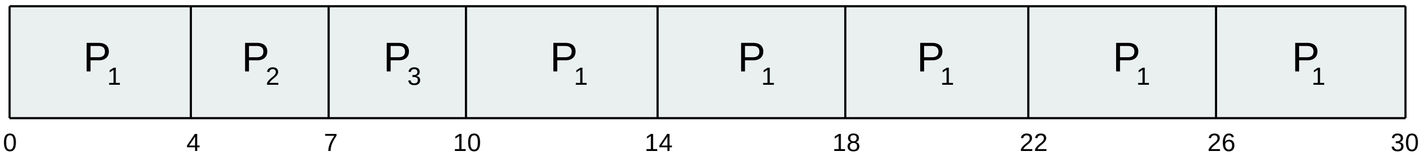
Round Robin (RR)

- **Advantages-**
- It gives the best performance in terms of average response time.
- It is best suited for time sharing system, client server architecture and interactive system.
- **Disadvantages-**
- It leads to starvation for processes with larger burst time as they have to repeat the cycle many times.
- Its performance heavily depends on time quantum.
- Priorities can not be set for the processes.

Example of RR with Time Quantum = 4

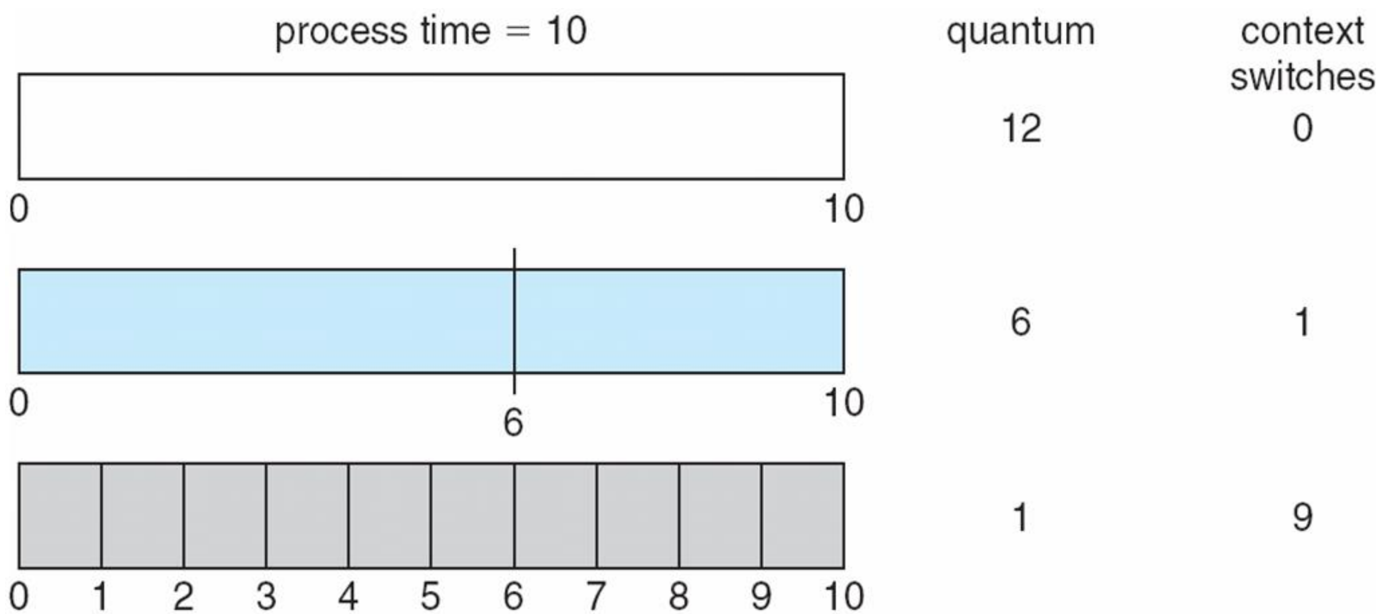
<u>Process</u>	<u>Burst Time</u>
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

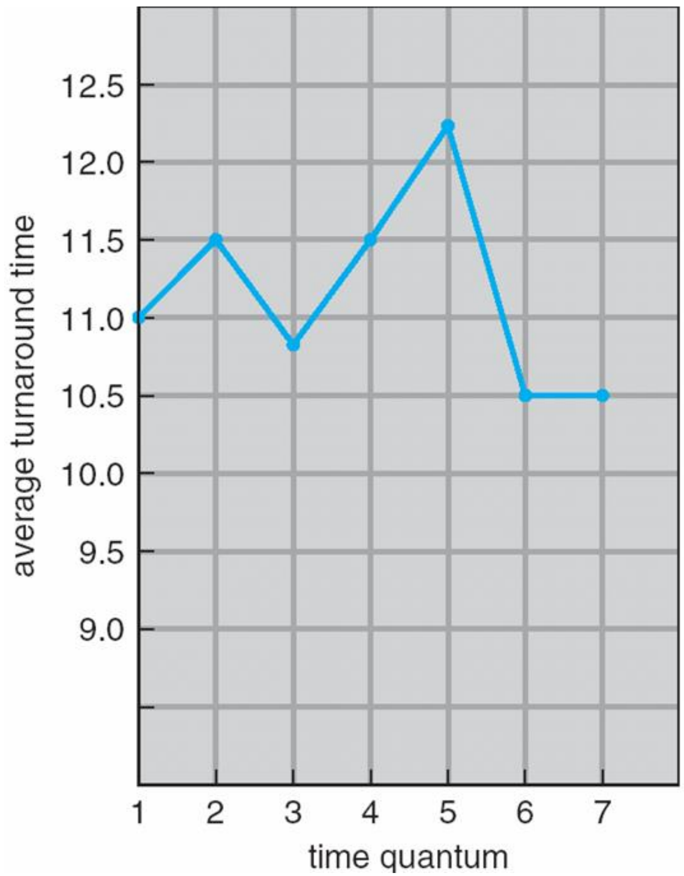


- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts
should be shorter than q

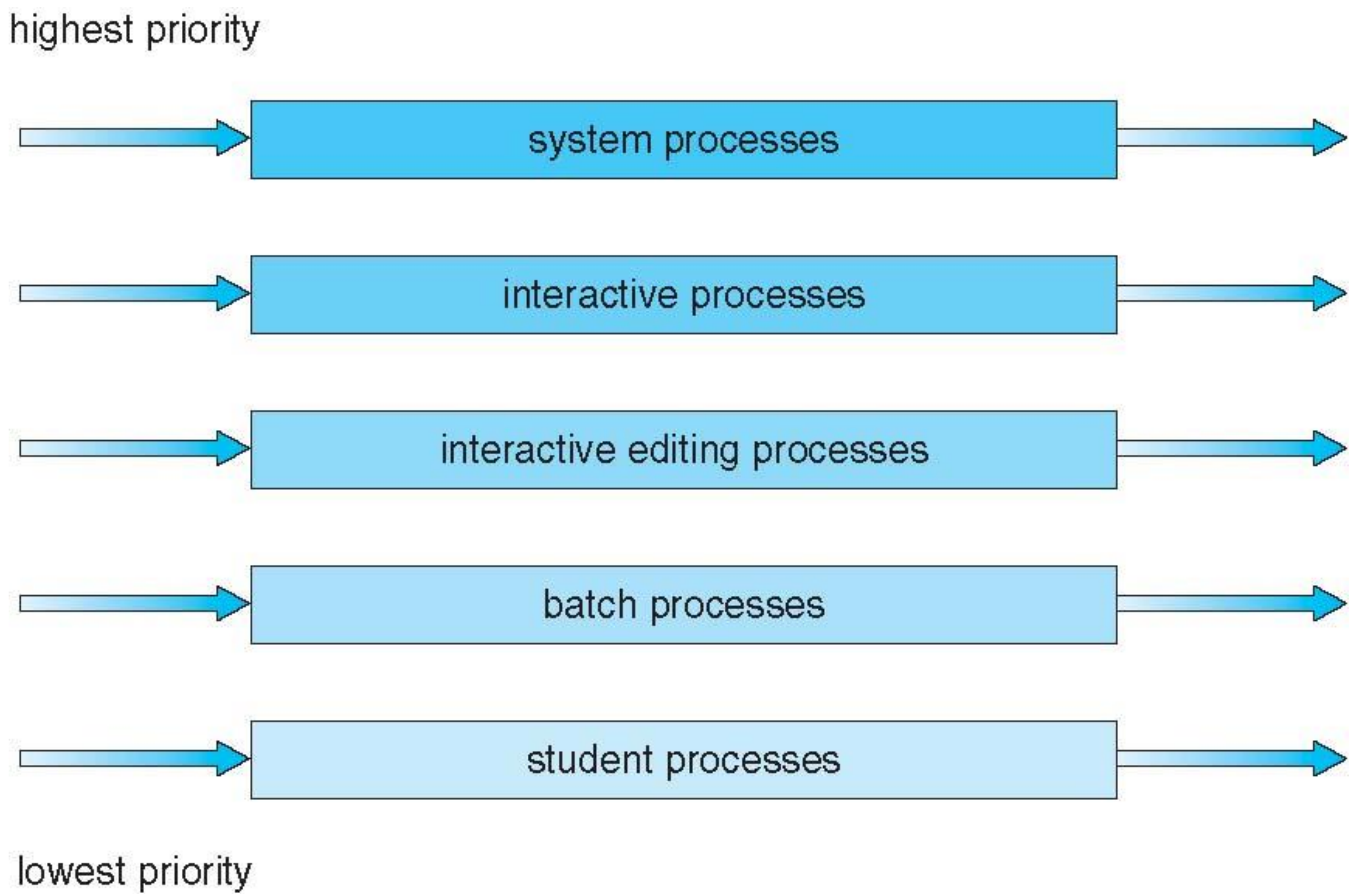
MIXED BURST TIME EXAMPLE

- Refer the example taken in lecture

Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
  - **foreground** (interactive)
  - **background** (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS

Multilevel Queue Scheduling

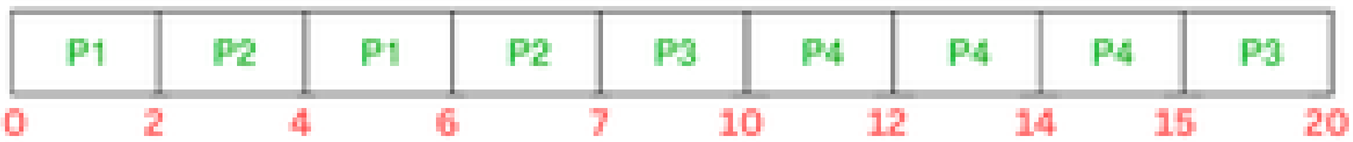


MLQ Example

Process	Arrival Time	CPU Burst Time	Queue Number
P1	0	4	1
P2	0	3	1
P3	0	8	2
P4	10	5	1

Priority of queue 1 is greater than queue 2.
queue 1 uses Round Robin (Time Quantum = 2) and queue 2 uses FCFS.

Gantt Chart



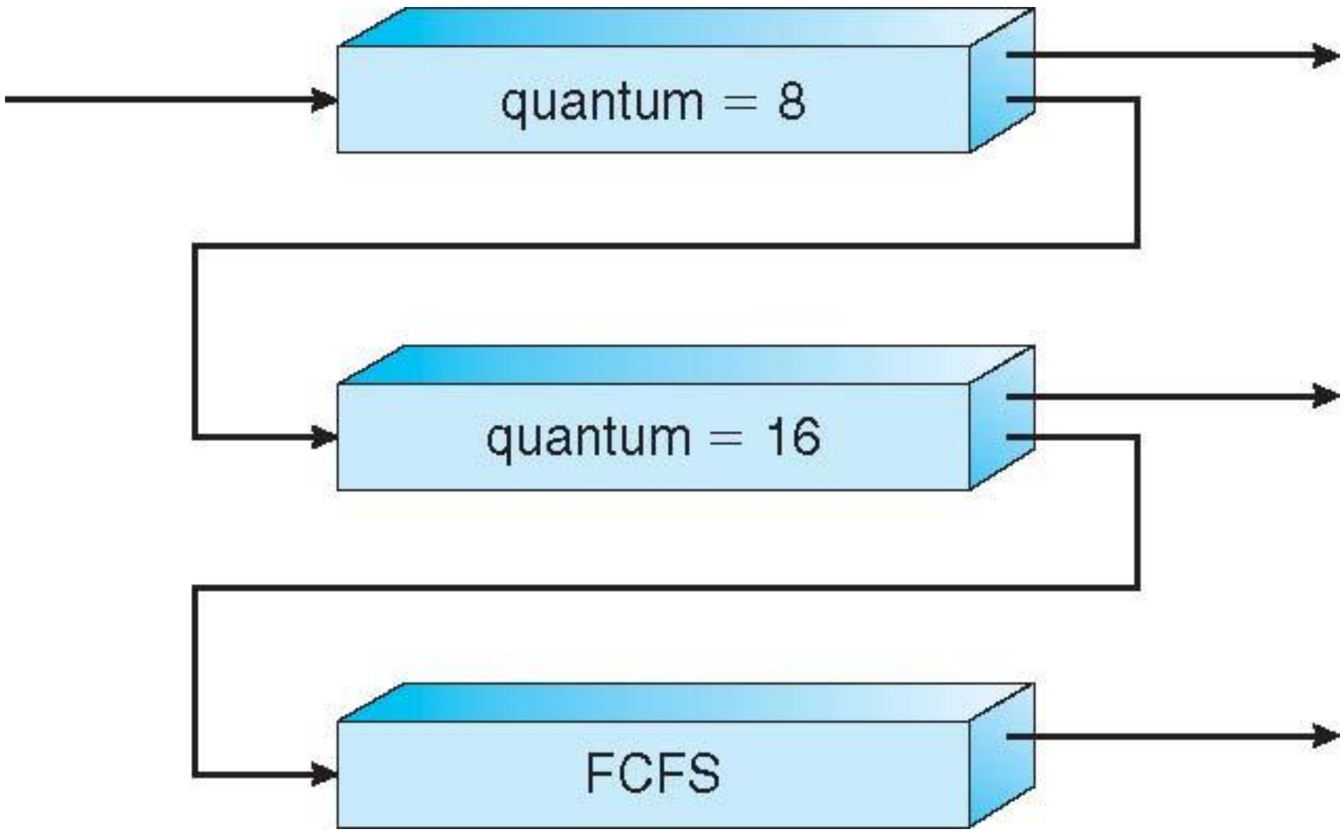
Working:

- At starting, both queues have process so process in queue 1 (P1, P2) runs first (because of higher priority) in the round robin fashion and completes after 7 units
- Then process in queue 2 (P3) starts running (as there is no process in queue 1) but while it is running P4 comes in queue 1 and interrupts P3 and start running for 5 second and
- After its completion P3 takes the CPU and completes its execution.

Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- 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



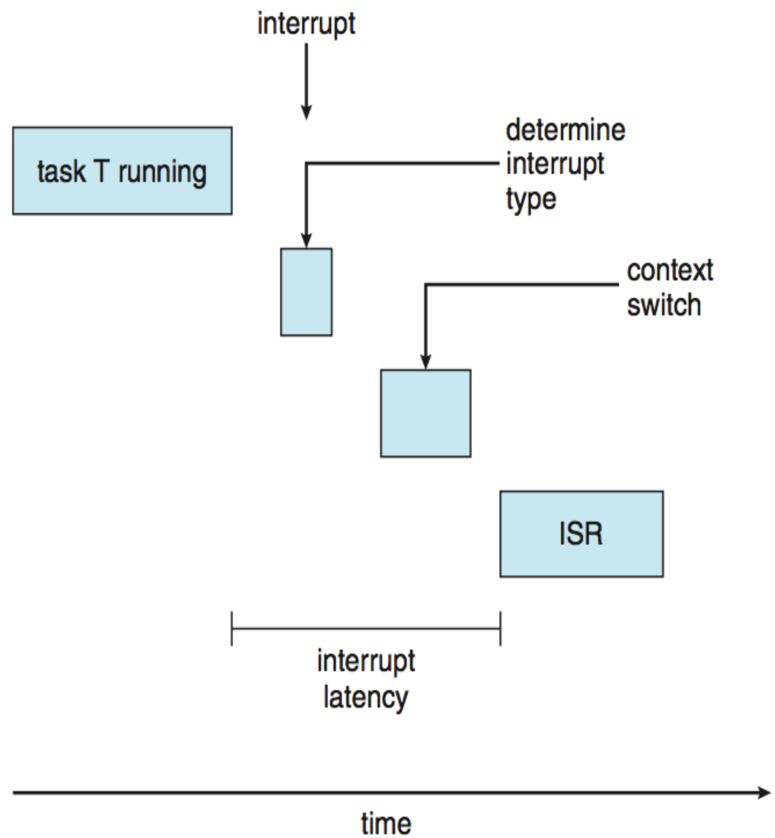
MLFQ

- Three queues:
  - 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

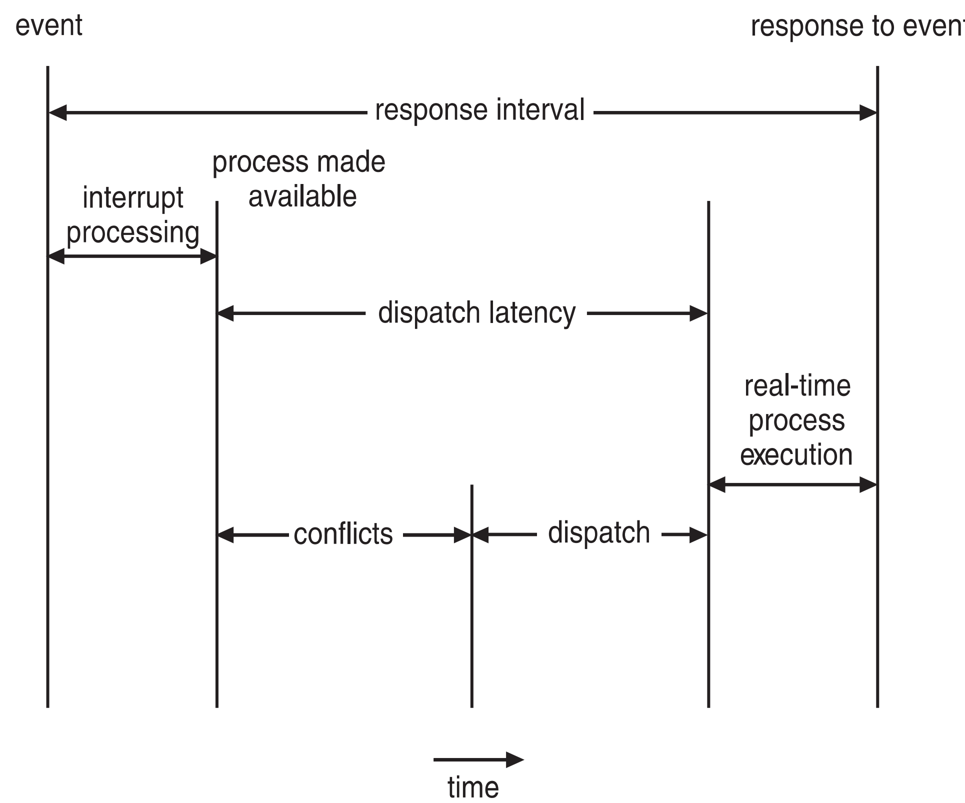
Real-Time CPU Scheduling

- Can present obvious challenges
- **Soft real-time systems** – no guarantee as to when critical real-time process will be scheduled
- **Hard real-time systems** – task must be serviced by its deadline
- Two types of latencies affect performance
  1. Interrupt latency – time from arrival of interrupt to start of routine that services interrupt
  2. Dispatch latency – time for schedule to take current process off CPU and switch to another



Real-Time CPU Scheduling (Cont.)

- Conflict phase of dispatch latency:
  1. Preemption of any process running in kernel mode
  2. Release by low-priority process of resources needed by high-priority processes



GATE QUESTIONS (1)

- Consider three CPU-intensive processes, which require 10, 20 and 30 time units and arrive at times 0, 2 and 6, respectively. How many context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end.

- **(A)** 1
  (B) 2
  (C) 3
  (D) 4

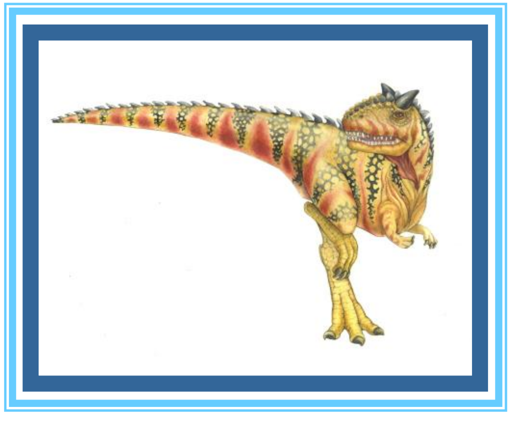
GATE QUESTIONS (2)

- An operating system uses shortest remaining time first scheduling algorithm for pre-emptive scheduling of processes. Consider the following set of processes with their arrival times and CPU burst times (in milliseconds):

Process	Arrival Time	Burst Time
P1	0	12
P2	2	4
P3	3	6
P4	8	5

The average waiting time (in milliseconds) of the processes is __________.

End of Chapter



Silberschatz, Galvin and Gagne ©2013