EECS 111:

System Software

Lecture : CPU Scheduling

Prof. Mohammad Al Faruque

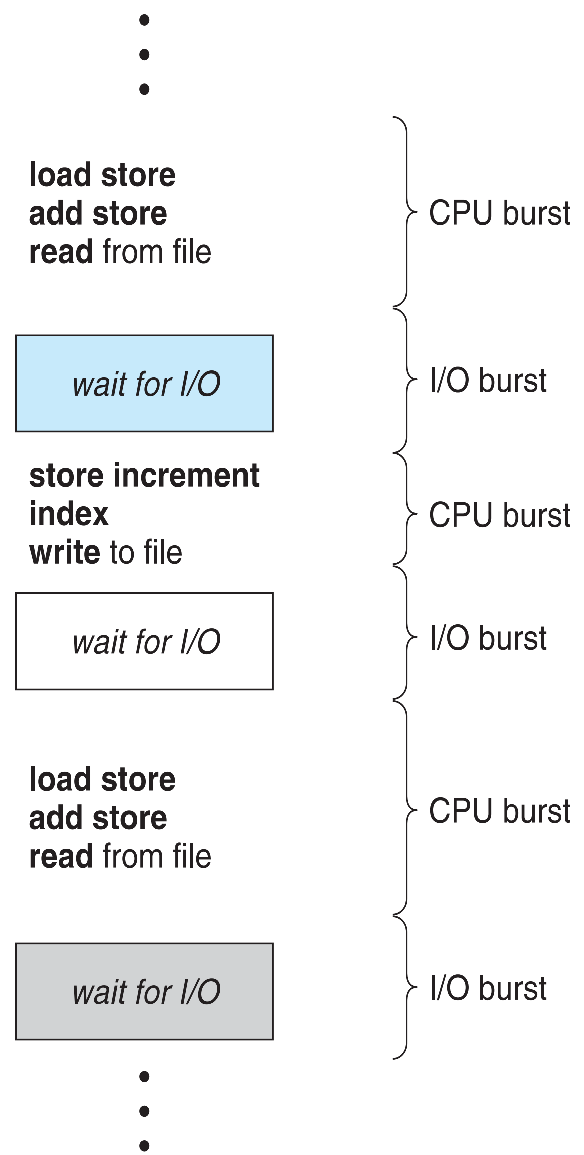
The Henry Samueli School of Engineering
Electrical Engineering & Computer Science
University of California Irvine (UCI)

Chapter 6: CPU Scheduling

- **Basic Concepts**
- **Scheduling Criteria**
- **Scheduling Algorithms**
- **Real-Time CPU Scheduling**

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

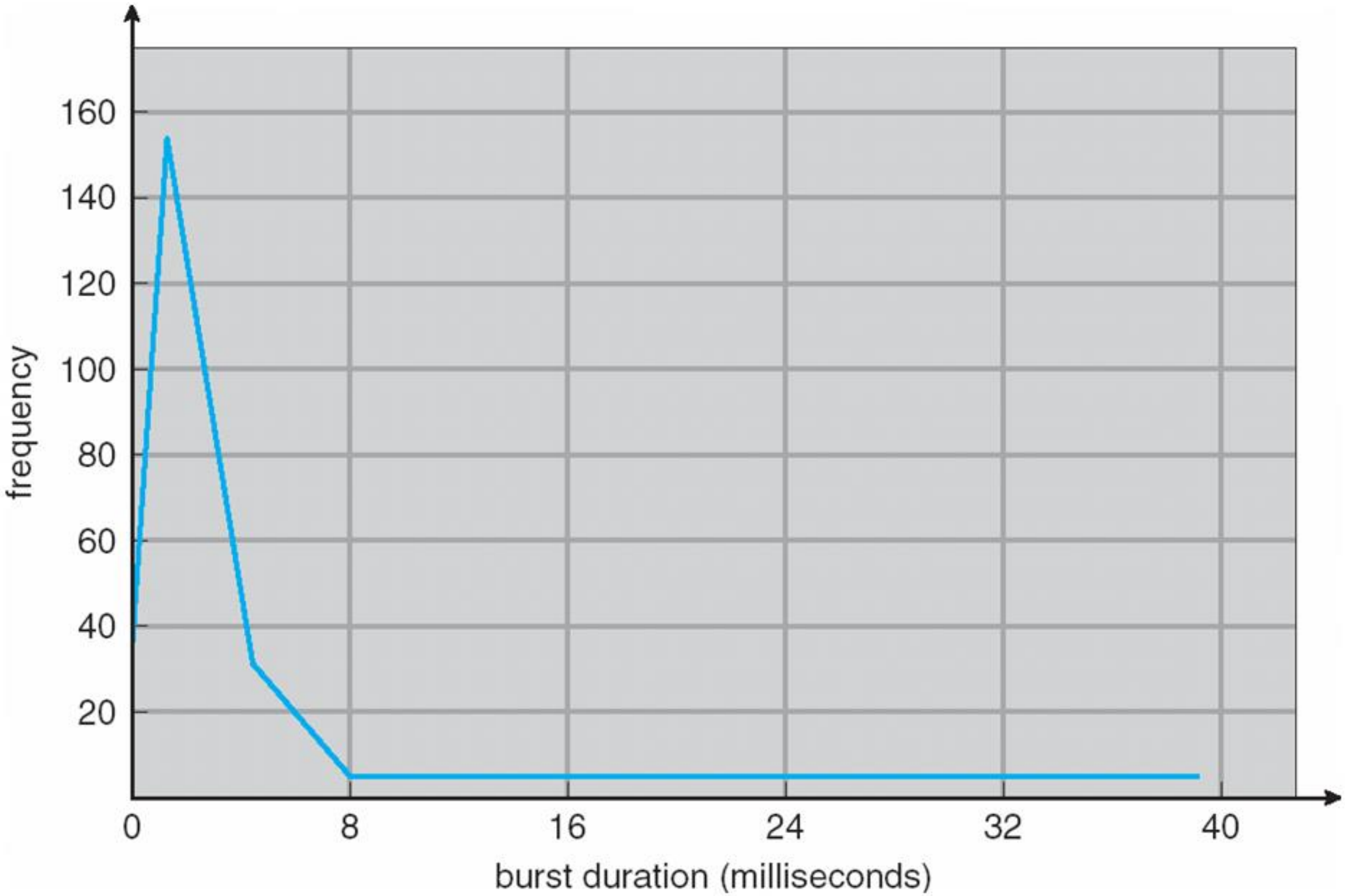
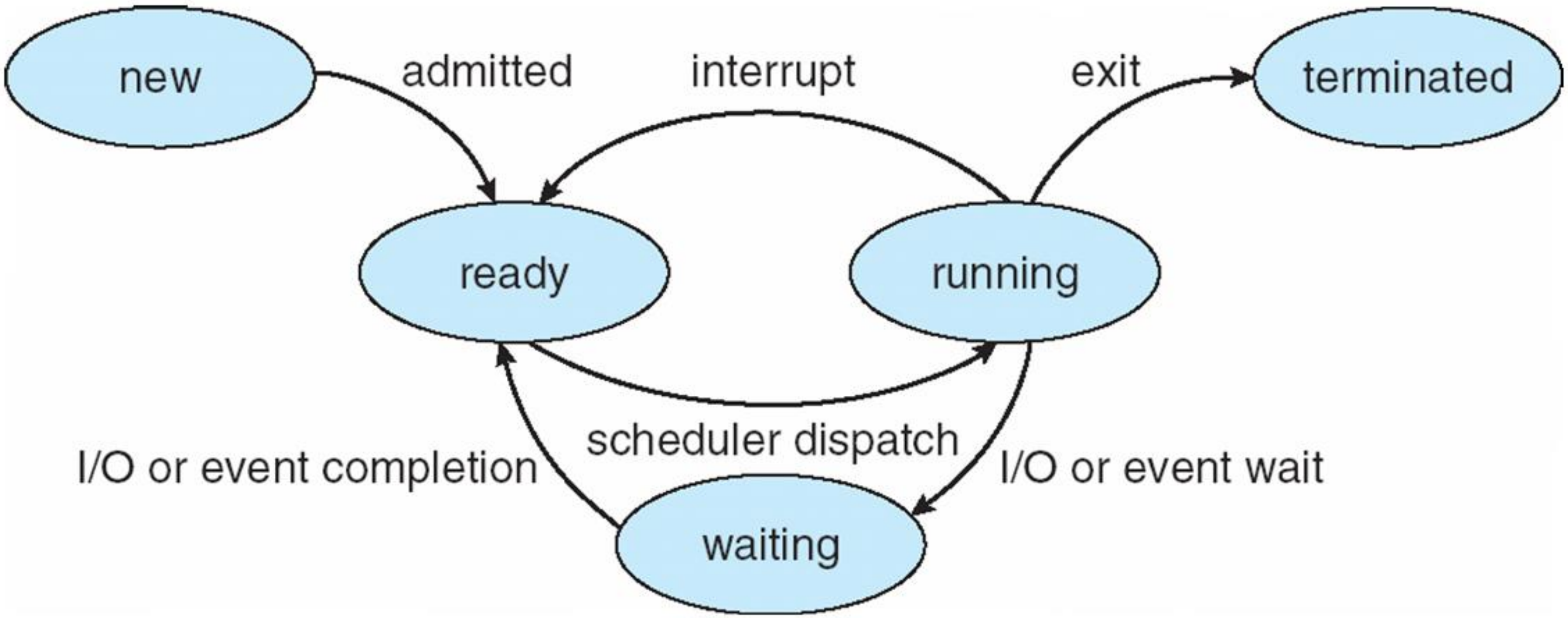


Diagram of Process State



Schedulers

- **Long-term scheduler (or job scheduler) -**
  - selects which processes should be brought into the ready queue.
  - invoked very infrequently (seconds, minutes); may be slow.
  - controls the degree of multiprogramming
- **Short term scheduler (or CPU scheduler) -**
  - selects which process should execute next and allocates CPU.
  - invoked very frequently (milliseconds) - must be very fast
- **Medium Term Scheduler**
  - swaps out process temporarily → swapping
  - balances load for better throughput

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

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)

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.)

- **Suspose 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**

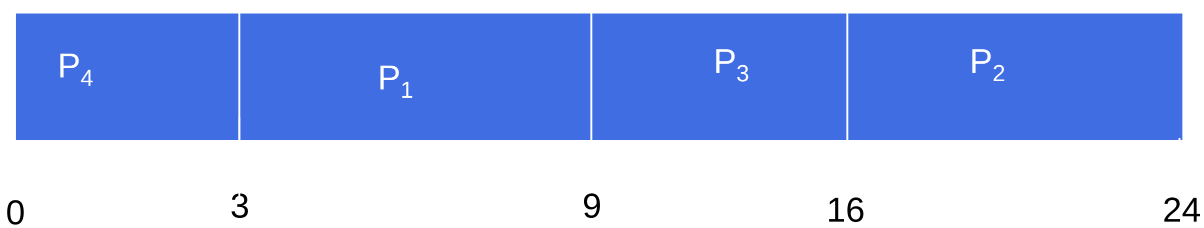
Shortest-Job-First (SJF) Scheduling

- 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

Determining Length of Next CPU Burst

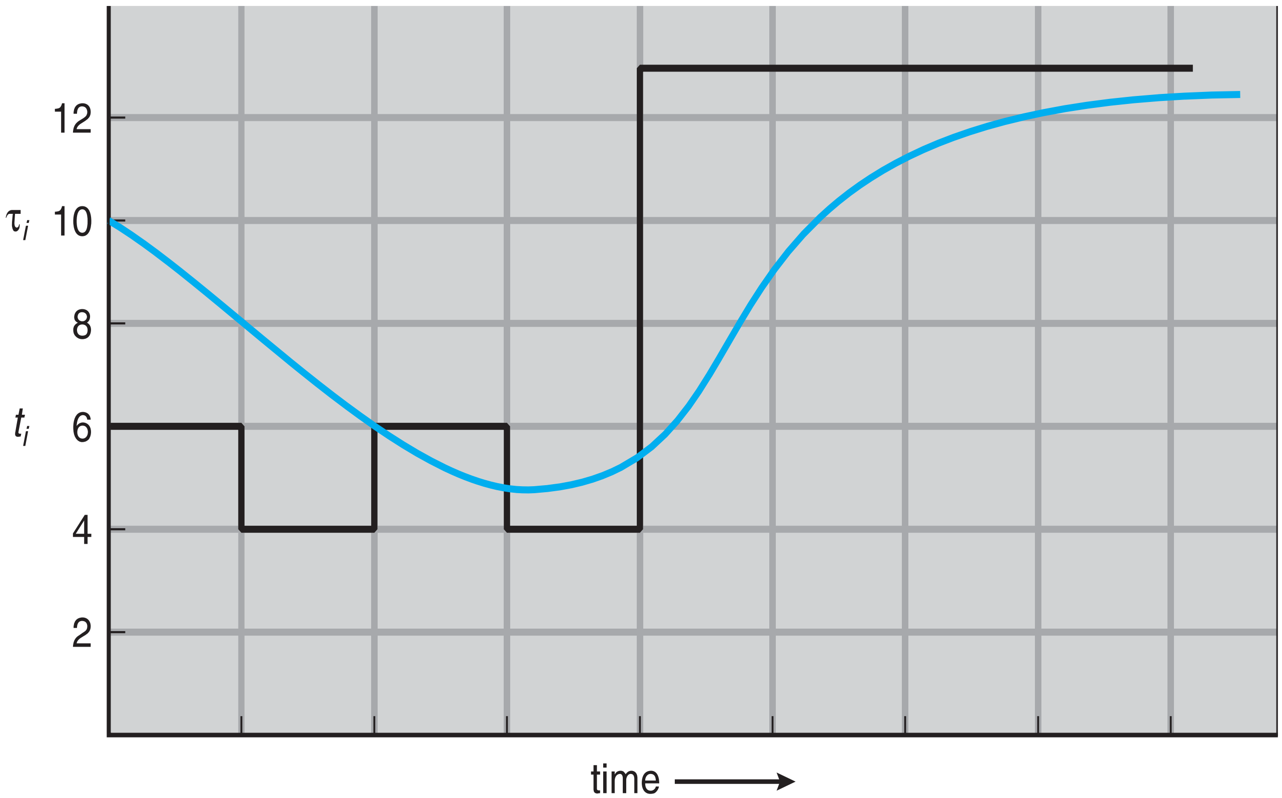
- **Can only estimate the length – should be similar to the previous one**
  - **Then pick process with shortest predicted next CPU burst**
- **Can be done by using the length of previous CPU bursts, using exponential averaging**

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

$$\tau_{n=1} = \alpha\, t_n + (1 - \alpha)\tau_n .$$

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

Prediction of the Length of the Next CPU Burst



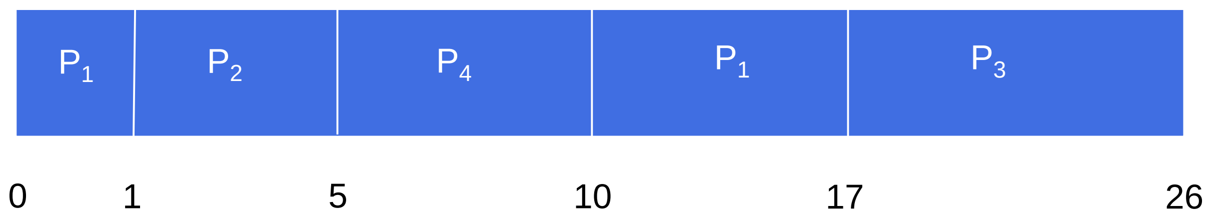
CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

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**

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

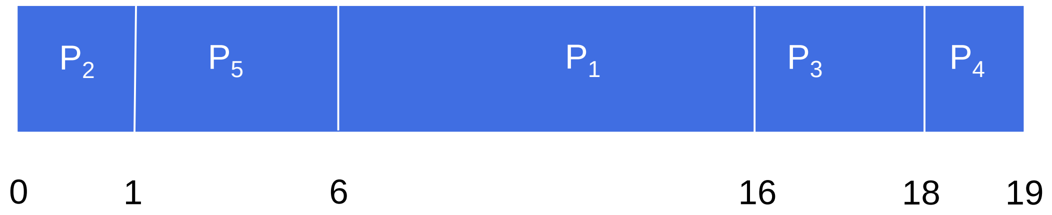
- Problem ≡ Starvation – low priority processes may never execute

- Solution ≡ Aging – as time progresses increase the priority of the process

Example of Priority Scheduling

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**

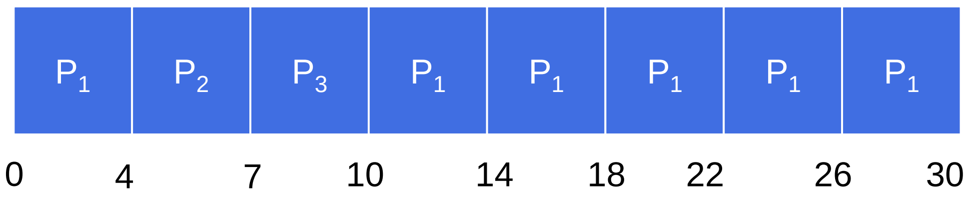
Round Robin (RR)

- 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

Example of RR with Time Quantum = 4

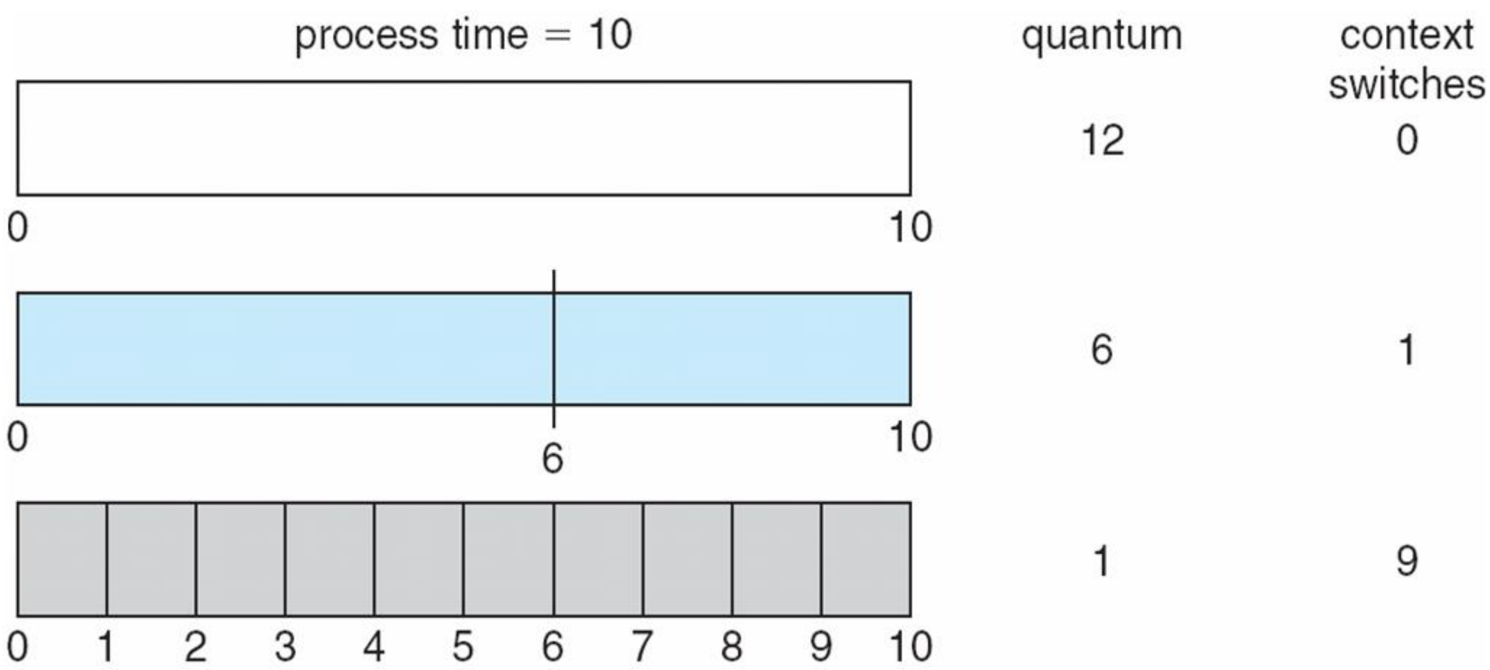
Process	Burst Time
P_1	24
P_2	3
P_3	3

- 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

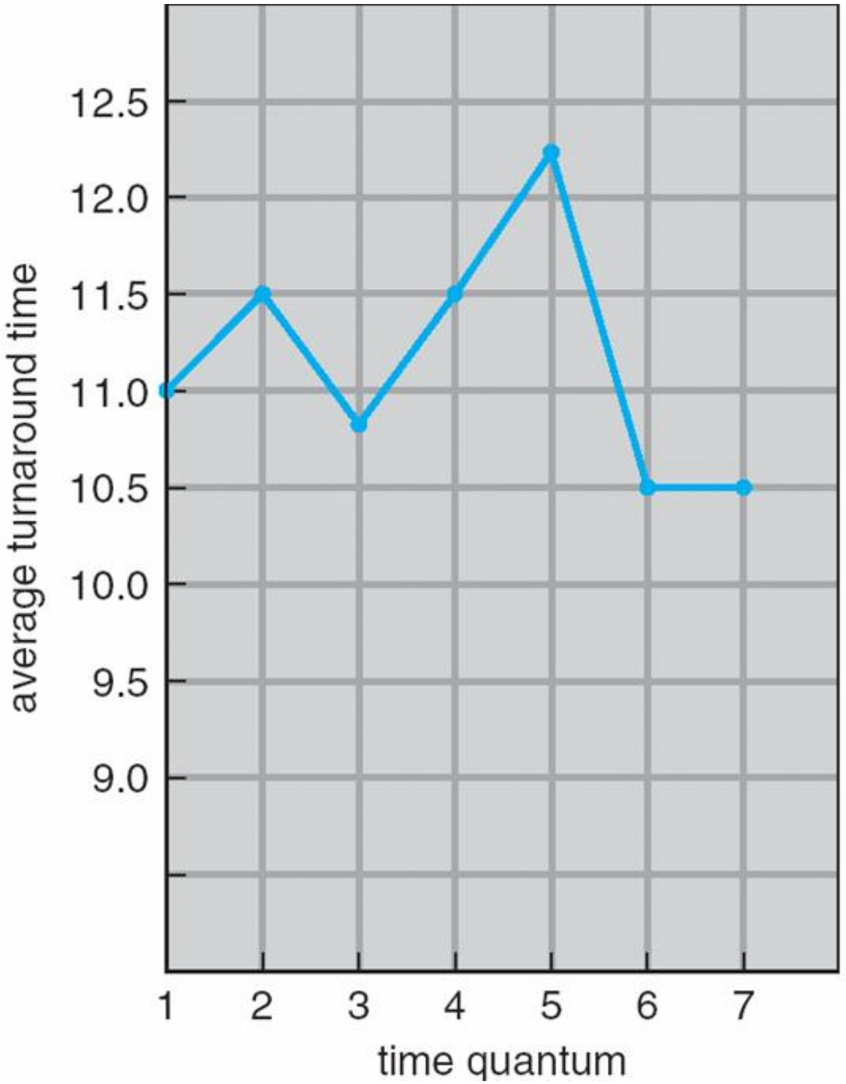
Time Quantum and Context Switch Time



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)**

Turnaround Time Varies With The Time Quantum

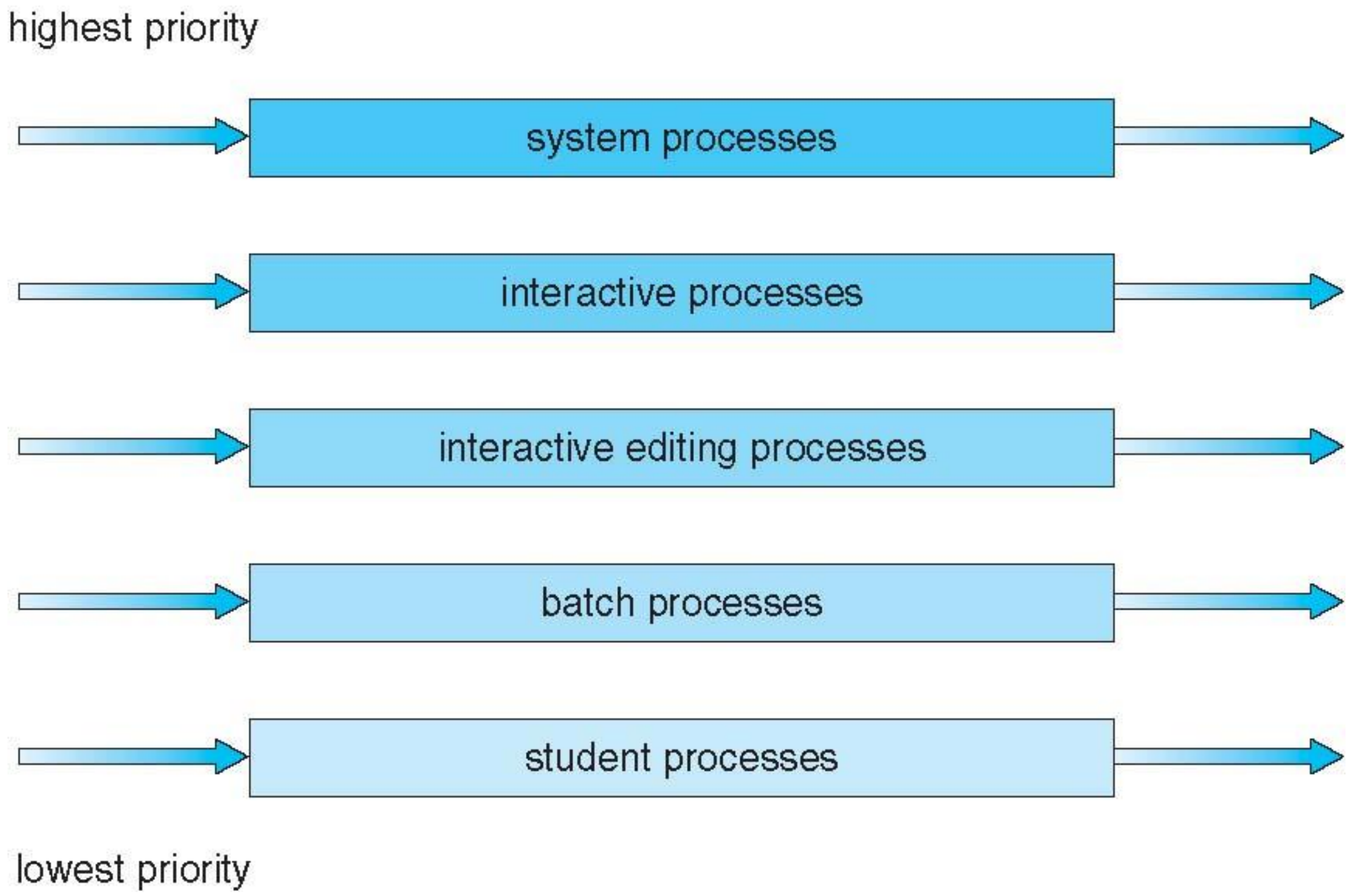


process	time
P_1	6
P_2	3
P_3	1
P_4	7

Multilevel Queue

- **Ready queue is partitioned into separate queues, e.g.:**
  - **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

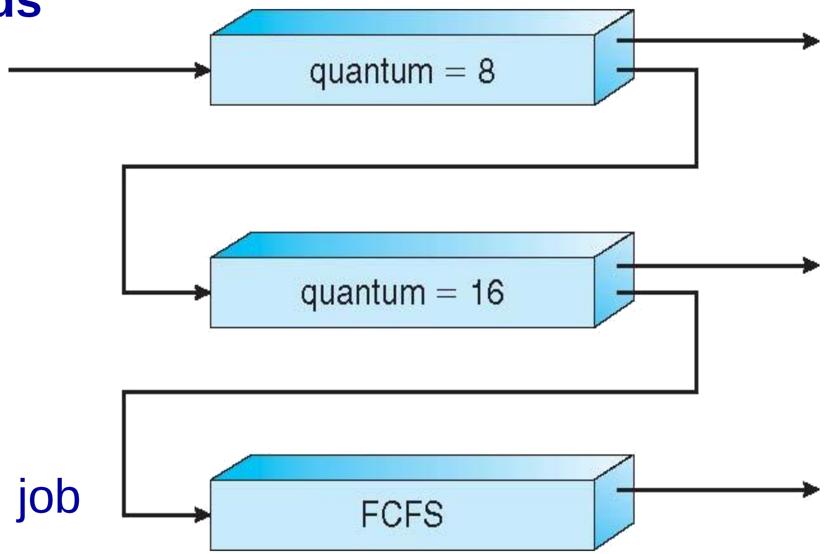


Multilevel Feedback Queue

- 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

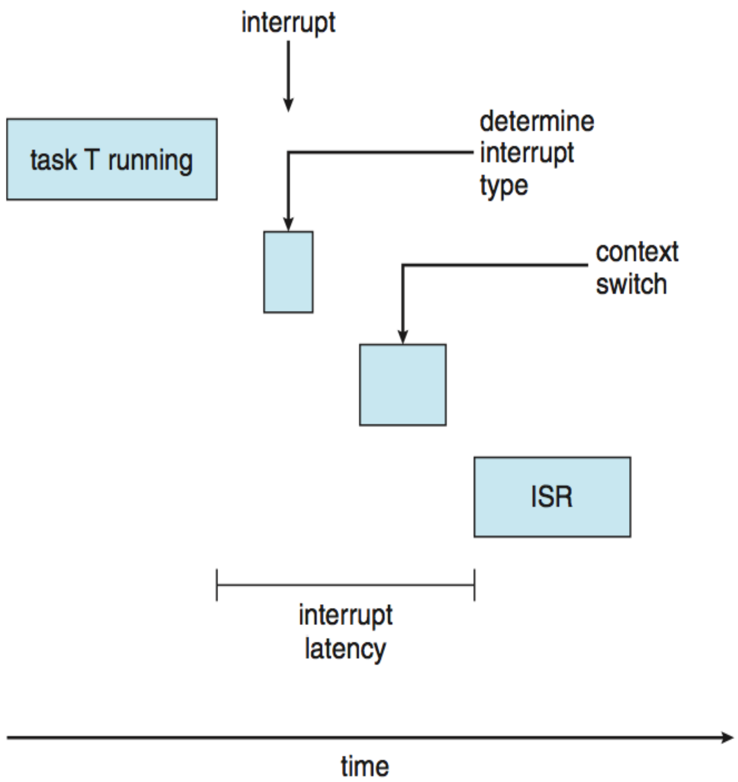
- **Three queues:**
  - ***Q_0* – RR with time quantum 8 milliseconds**
  - ***Q_1* – RR time quantum 16 milliseconds**
  - ***Q_2* – FCFS**



- **Scheduling**
  - **A new job enters queue *Q_0* which is served RR**
    - 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 RR and receives 16 additional milliseconds**
    - If it still does not complete, it is preempted and moved to queue Q_2
  - **At *Q_2* job is served FCFS only if Q0 and Q1 are empty**

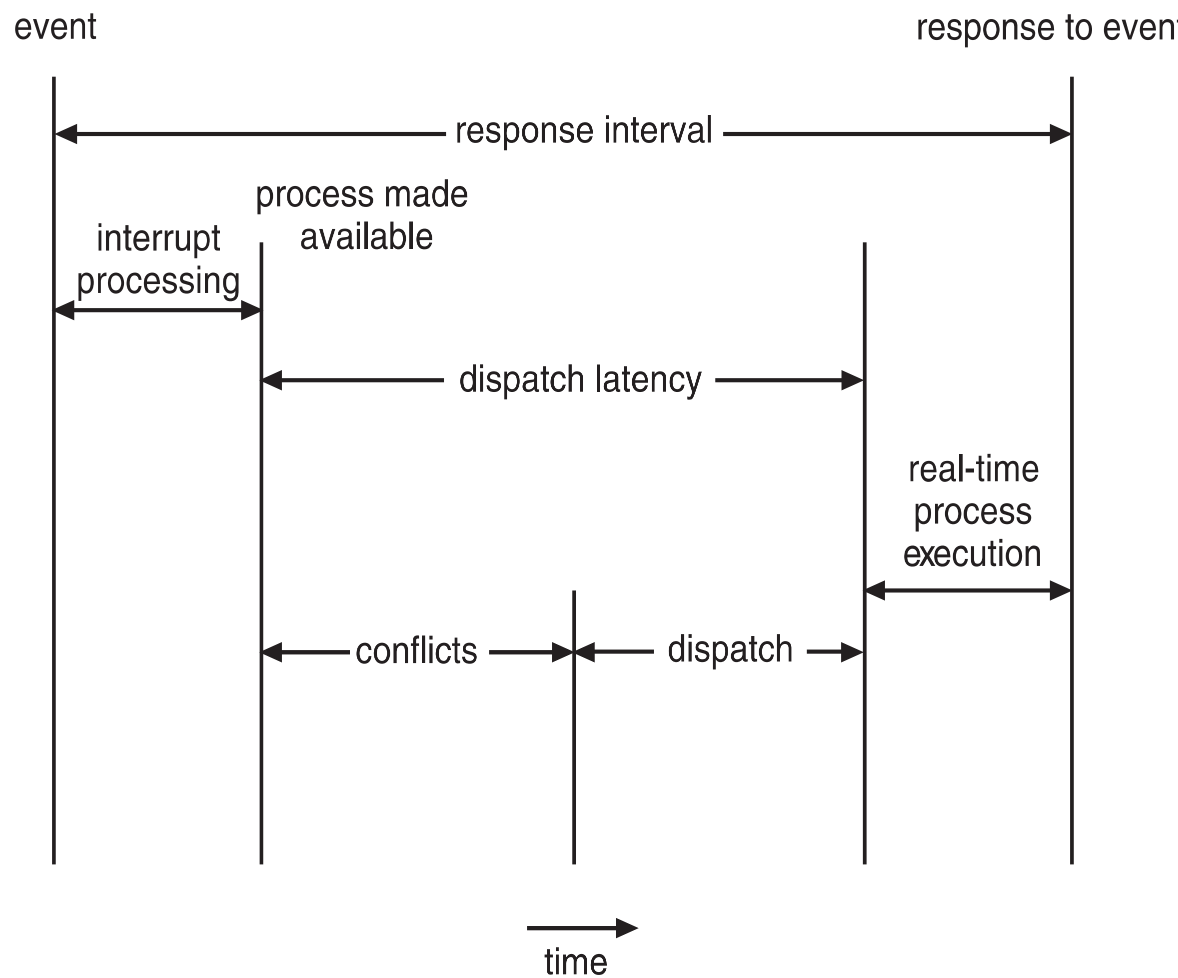
Real-Time CPU Scheduling

- 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



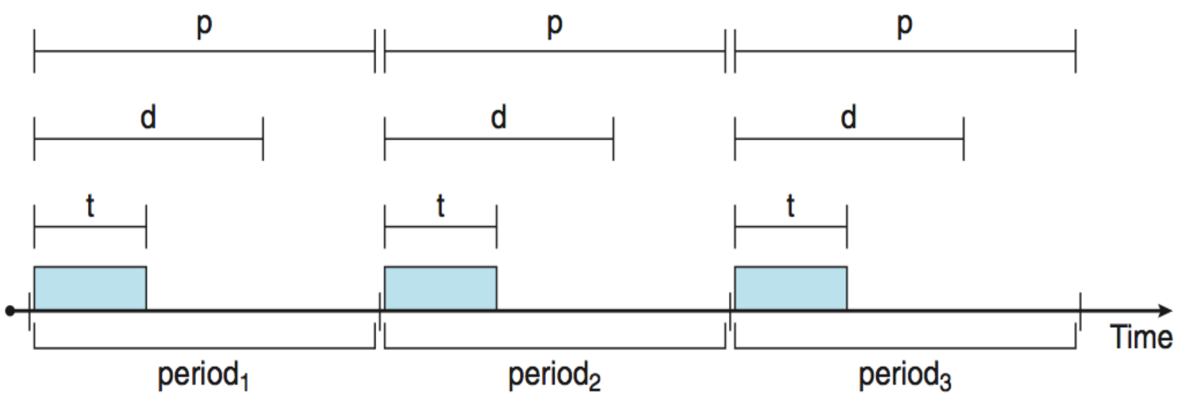
Real-Time CPU Scheduling (Cont.)

- **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**



Priority-based Scheduling

- **For real-time scheduling, scheduler must support preemptive, priority-based scheduling**
  - **But only guarantees soft real-time**
- **For hard real-time must also provide ability to meet deadlines**
- **Processes have new characteristics: periodic ones require CPU at constant intervals**
  - **Has processing time *t*, deadline *d*, period *p***
  - $0 \le t \le d \le p$
  - **Rate of periodic task is 1/*p***

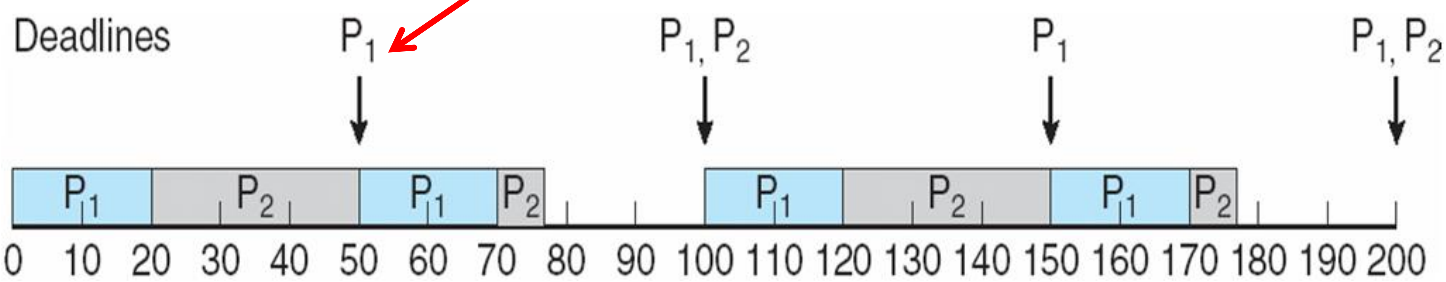


Rate Montonic Scheduling

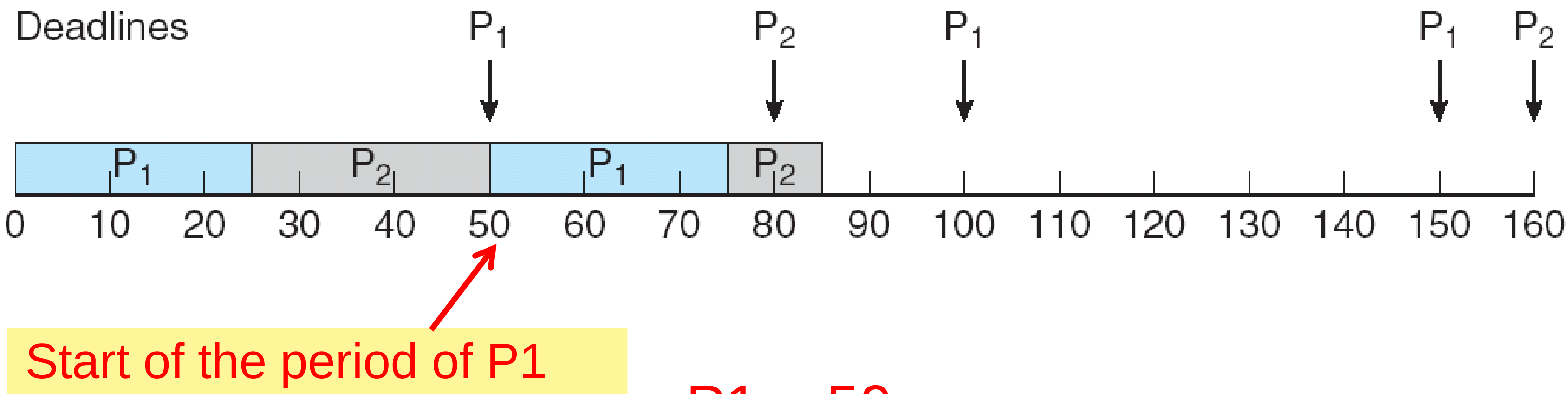
- **A priority is assigned based on the inverse of its period**
- **Shorter periods = higher priority;**
- **Longer periods = lower priority**
- **P_1 is assigned a higher priority than P_2.**

P1 = 50
t1 = 20
P2 = 100
t2 = 35
Deadline for each process requires that it complete its CPU burst by the start of its next period.

Start of the period of P1



Missed Deadlines with Rate Monotonic Scheduling



P1 = 50

t1 = 25

P2 = 80

t2 = 35

Earliest Deadline First Scheduling (EDF)

- **Priorities are assigned according to deadlines:**

the earlier the deadline, the higher the priority;

the later the deadline, the lower the priority

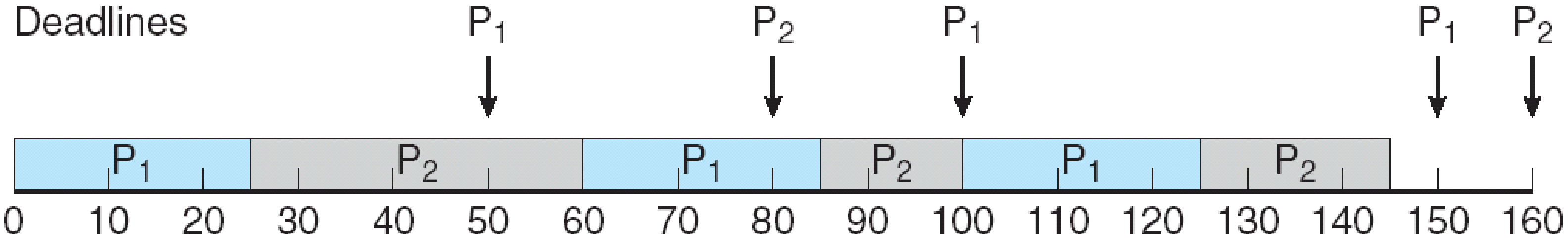
P1 = 50

t1 = 25

P2 = 80

t2 = 35

Dt2 = 80 Dt1= 100	Dt1 = 100 Dt2= 160	Dt1 = 150 → start of its period Dt2= 160



References

Part of the contents of this lecture has been adapted from the book Abraham Silberschatz, Peter B. Galvin, Greg Gagne: "Operating System Concept ", Publisher : Wiley; 9 edition (December 17, 2012), ISBN-13: 978-1118063330

Slides also contain lecture materials from John Kubiatowicz (Berkeley), John Ousterhout (Stanford), Nalini (UCI), Rainer (UCI), and others

Some slides adapted from http://www-inst.eecs.berkeley.edu/~cs162/ Copyright © 2010 UCB

Thank you for your attention