Process Concept & Scheduling

Nirmala Shinde Baloorkar
Assistant Professor
Department of Computer Engineering

SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya
TRUST

Outline

- Basic Concept
- Scheduling Criteria
- Scheduling Algorithm

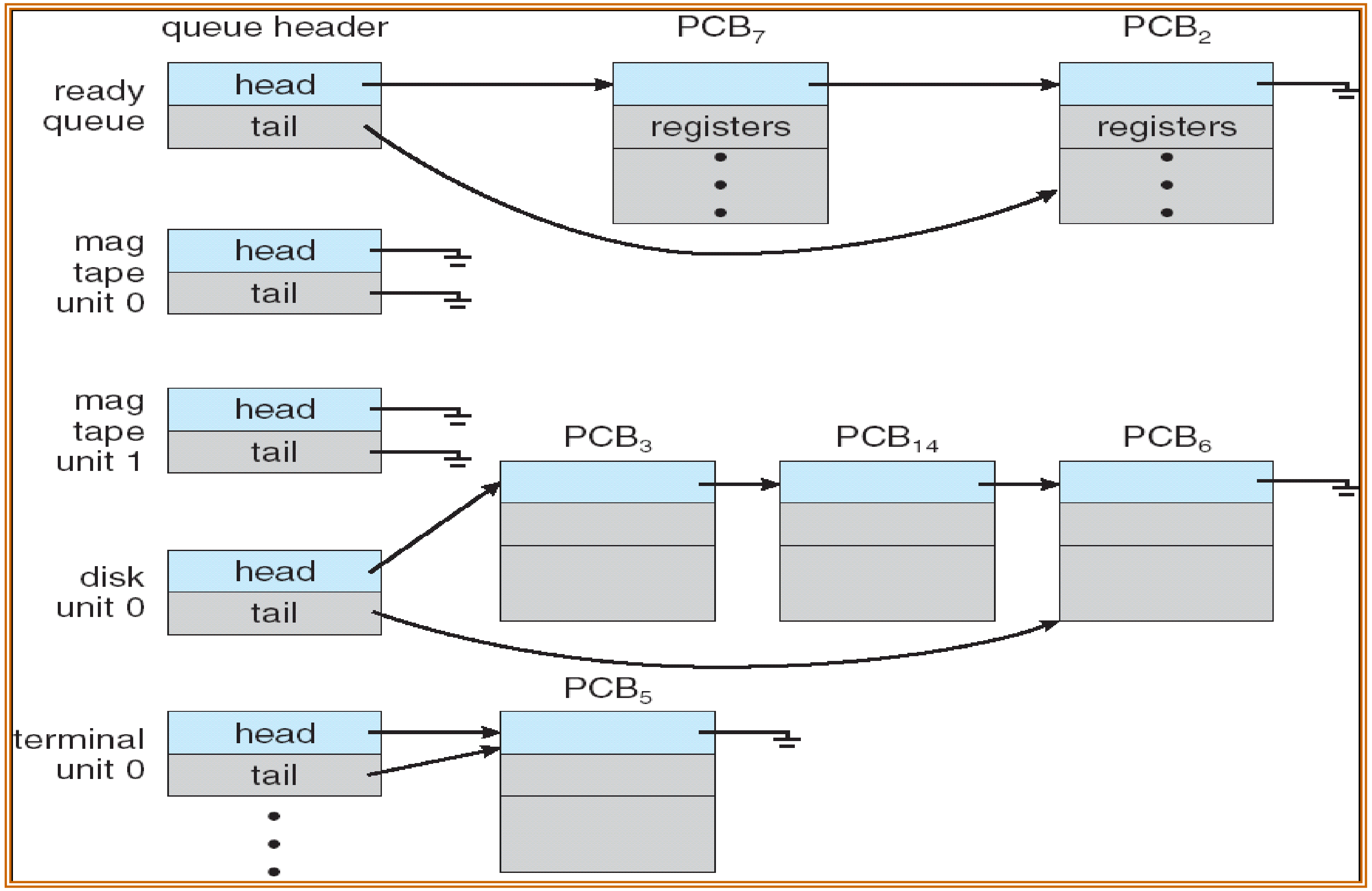
Process Scheduling

- The operating system is responsible for managing the *scheduling* activities.
  - A uniprocessor system can have only one running process at a time
  - The main memory cannot always accommodate all processes at run-time
  - The operating system will need to decide on which process to execute next (CPU scheduling), and which processes will be brought to the main memory (job scheduling)

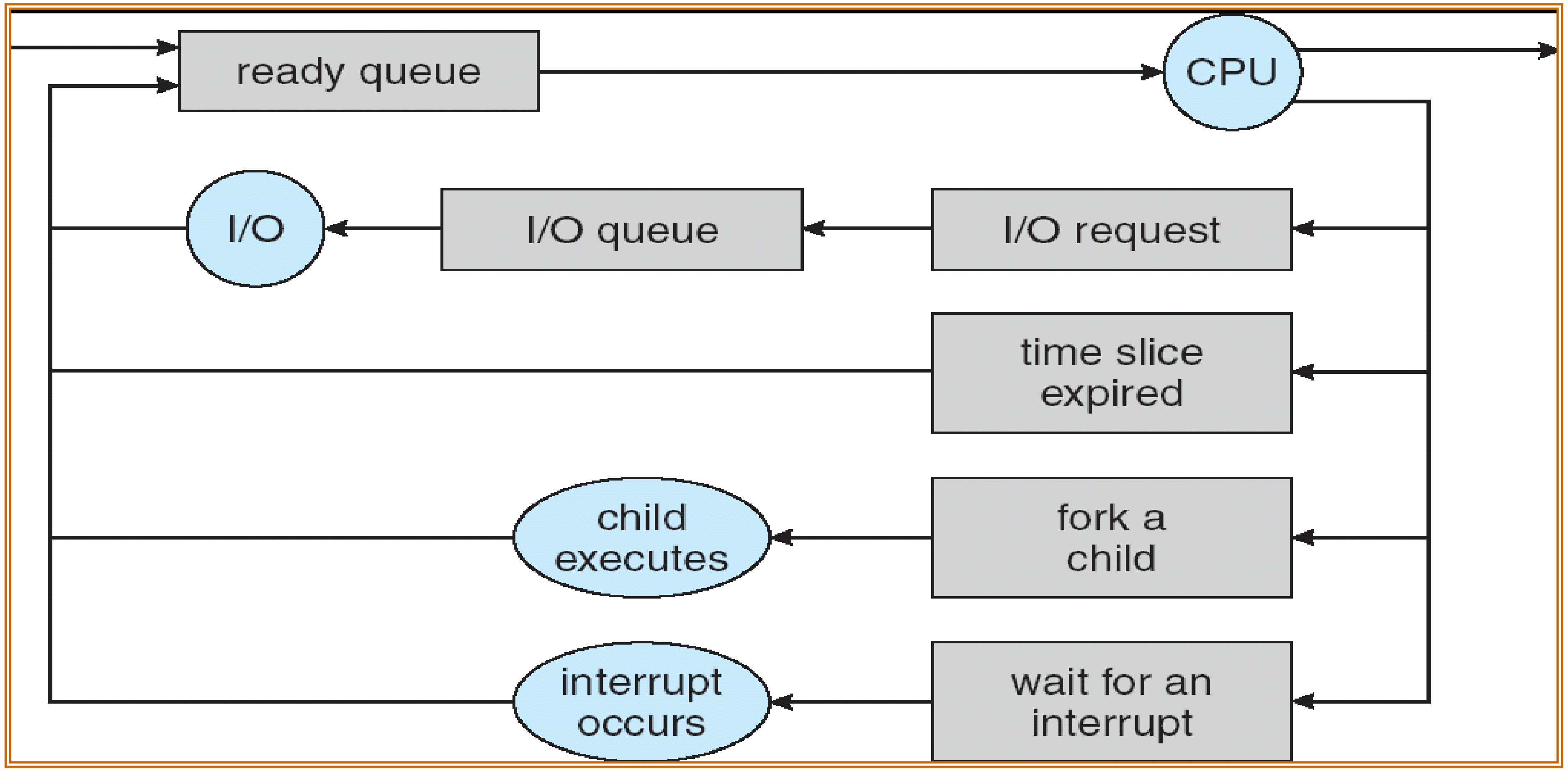
Process Scheduling Queues

- Job queue – set of all processes in the system.
- Ready queue – set of all processes residing in main memory, ready and waiting for CPU.
- Device queues – set of processes waiting for an I/O device.
- Process migration is possible between these queues.

Ready Queue and I/O Device Queues

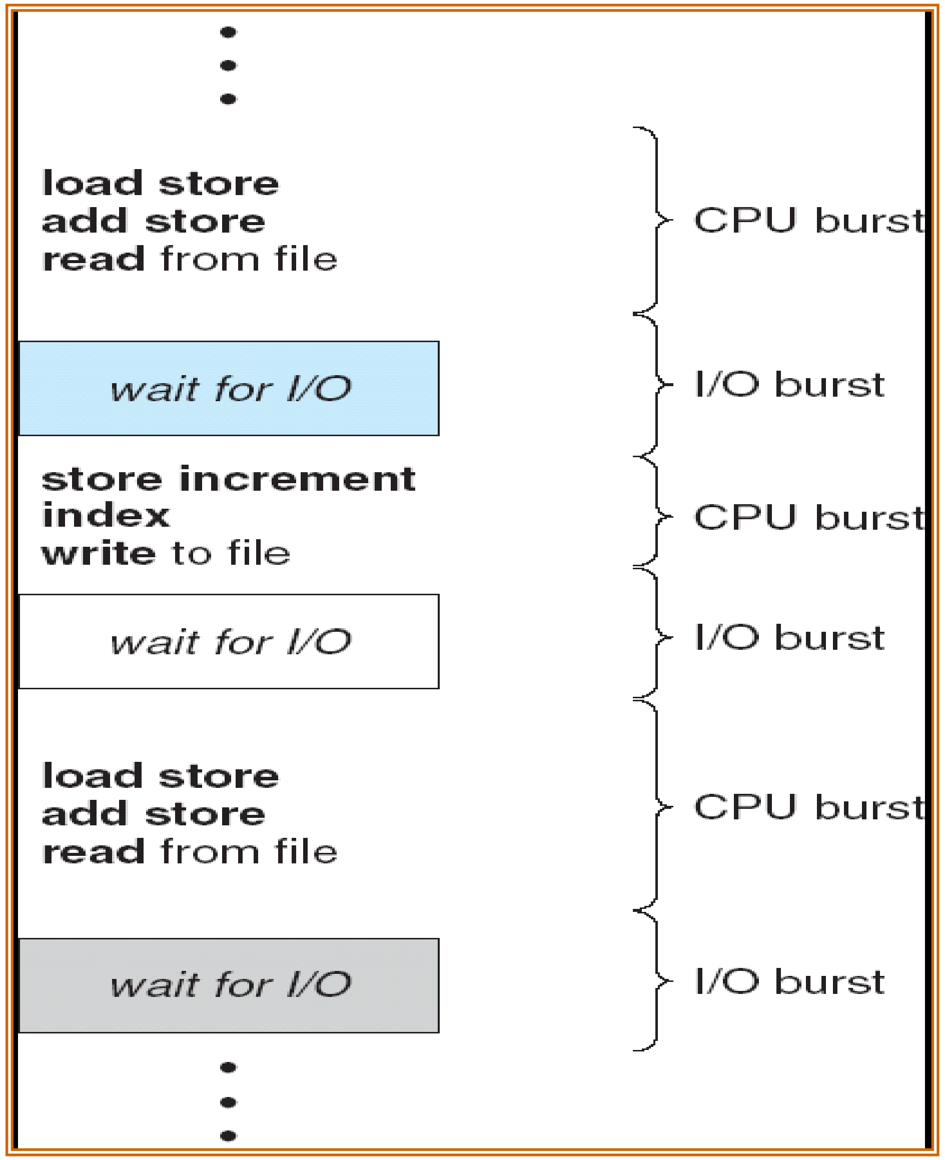


Process Lifecycle

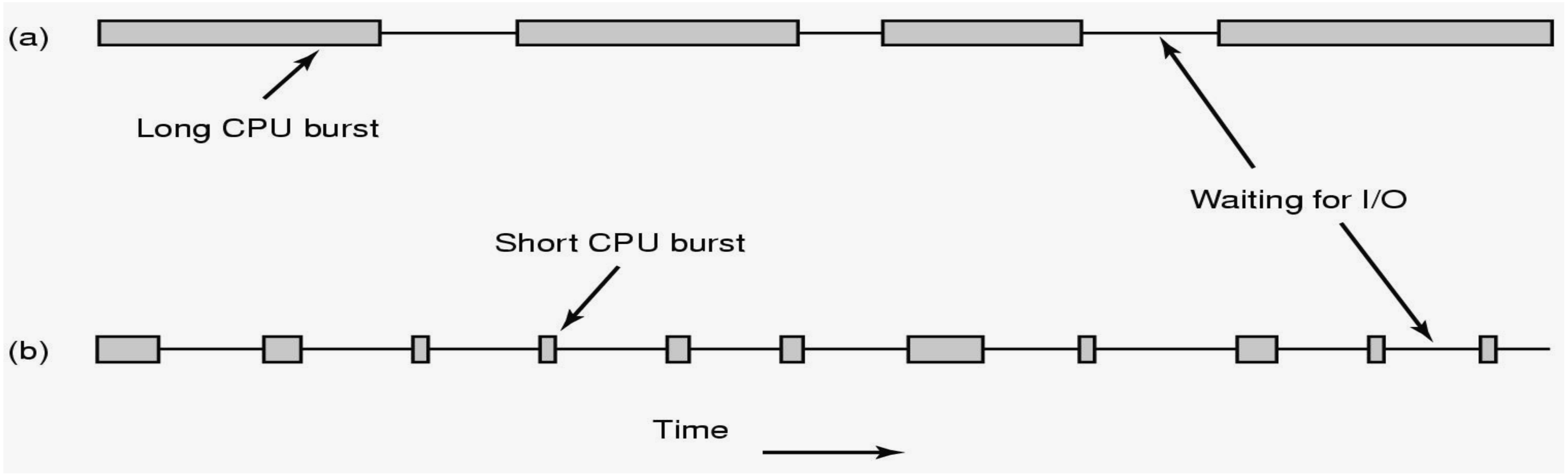


CPU and I/O Bursts

- CPU–I/O Burst Cycle –
  - Process execution consists of a *cycle* of CPU execution and I/O wait.
- I/O-*bound process* – spends more time doing I/O than computations, many short CPU bursts.
- *CPU-bound process* – spends more time doing computations; few very long CPU bursts.



CPU-bound and I/O-bound Processes

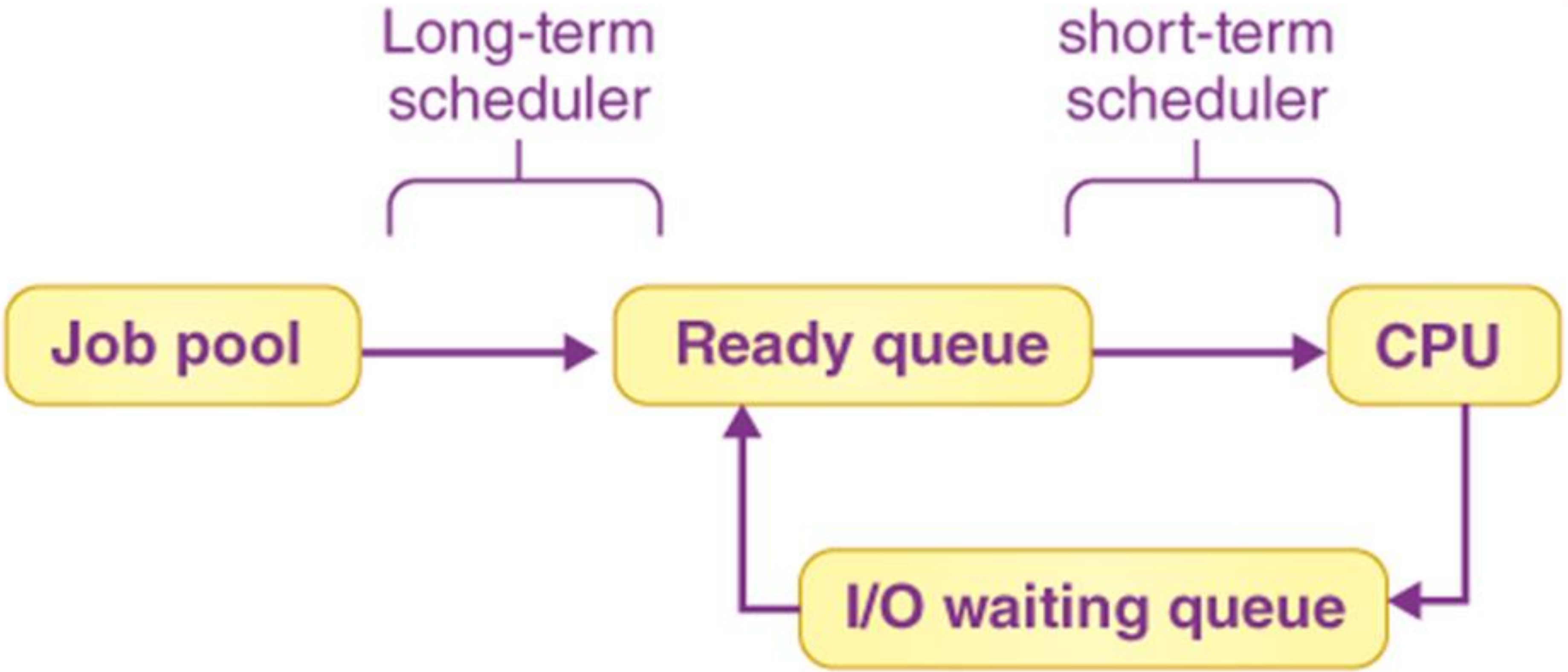


(a) A CPU-bound process   (b) An I/O-bound process

Schedulers

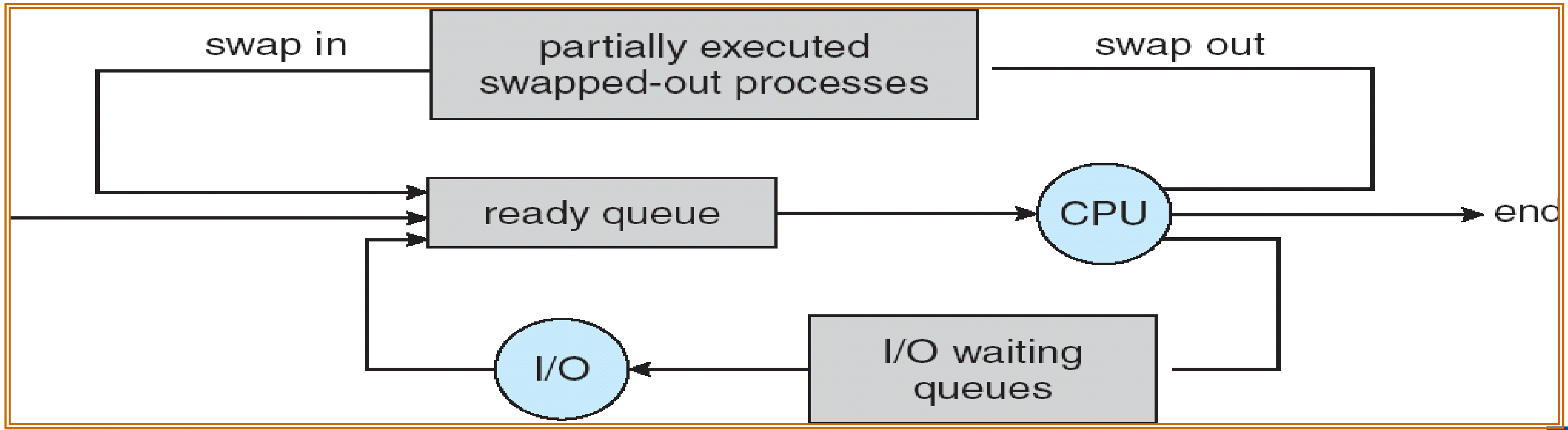
- The processes may be first spooled to a mass-storage system, where they are kept for later execution.
- The *long-term scheduler* *(or job scheduler)* – selects processes from this pool and loads them into memory for execution.
    - The long term scheduler, if it exists, will control the *degree of multiprogramming*
- The *short-term scheduler* *(or CPU scheduler)* – selects from among the *ready* processes, and allocates the CPU to one of them.
    - Unlike the long-term scheduler, the short-term scheduler is invoked very frequently.

Short Term Scheduler



Addition of Medium-Term Scheduler

- The medium-term scheduler can reduce the degree of multiprogramming by removing processes from memory.
- At some later time, the process can be re-introduced into memory (*swapping*).



Comparison of Schedulers

Parameters	Long-Term	Short-Term	Medium-Term
Type of Scheduler	It is a type of job scheduler.	It is a type of CPU scheduler.	It is a type of process swapping scheduler.
Speed	Its speed is comparatively less than that of the Short-Term scheduler.	It is the fastest among the other two.	Its speed is in between both Long and Short-Term schedulers.
Minimal time-sharing system	Almost absent	Minimal	Present

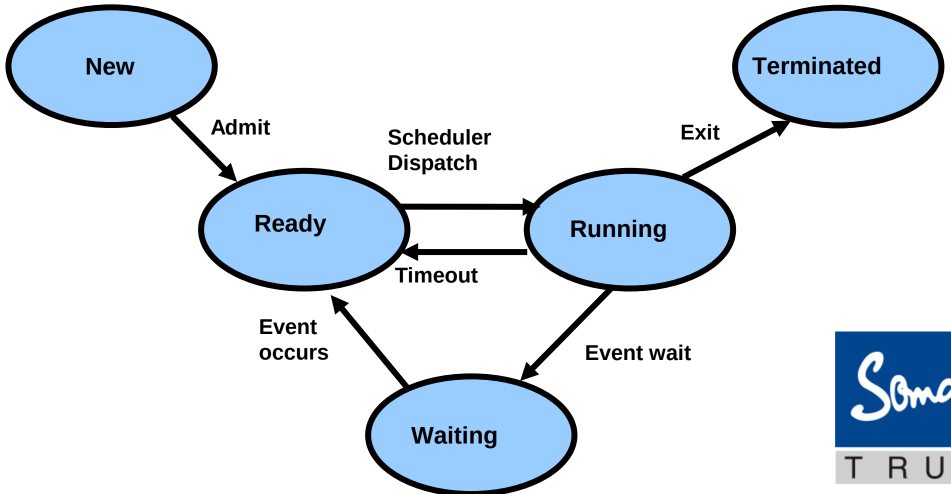
Comparison of Schedulers

Parameters	Long-Term	Short-Term	Medium-Term
Purpose	A Long-Term Scheduler helps in controlling the overall degree of multiprogramming.	The Short-Term Scheduler provides much less control over the degree of multiprogramming.	Medium-Term reduces the overall degree of multiprogramming.
Function	Selects processes from the pool and then loads them into the memory for execution.	Selects all those processes that are ready to be executed.	Can re-introduce the given process into memory. The execution can then be continued.

When to Schedule?

Process state transition model, CPU scheduler could be invoked at five different points:

1. When a process switches from the new state to the ready state.
2. When a process switches from the running state to the waiting state.
3. When a process switches from the running state to the ready state.
4. When a process switches from the waiting state to the ready state.
5. When a process terminates.



Non-preemptive vs. Preemptive Scheduling

- Under non-preemptive scheduling, each running process keeps the CPU until it completes or it switches to the waiting (blocked) state (points 2 and 5 from previous slides).

- Under preemptive scheduling, a running process may be also forced to release the CPU even though it is neither completed nor blocked.
  - In time-sharing systems, when the running process reaches the end of its time quantum (slice)
  - In general, whenever there is a change in the ready queue.

Scheduling Criteria

Several criteria can be used to compare the performance of scheduling algorithms

- 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 the complete output.
- Fairness - Ensuring that all processes receive an equitable share of CPU time, preventing any single process from monopolizing resources or starving.
- Meeting the deadlines (real-time systems)

Optimization Criteria

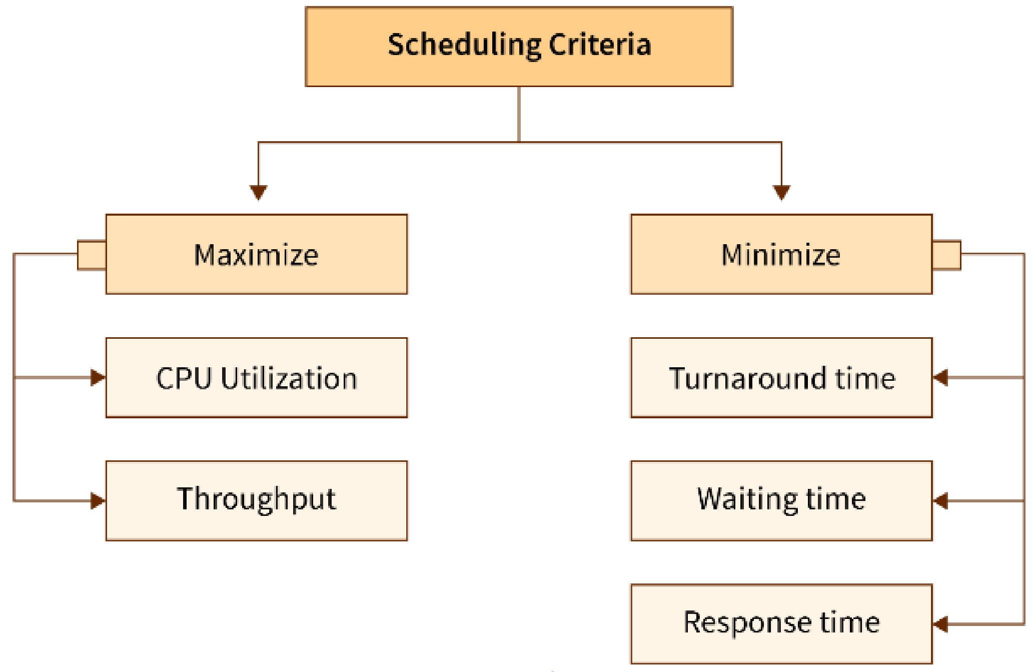


Image source : Scaler

- In the examples, we will assume
  - average waiting time is the performance measure
  - only one CPU burst (in milliseconds) per process

First-Come, First-Served (FCFS) Scheduling

- Single FIFO ready queue
- No-preemptive
  - Not suitable for timesharing systems
- Simple to implement and understand
- Average waiting time dependent on the order processes enter the system

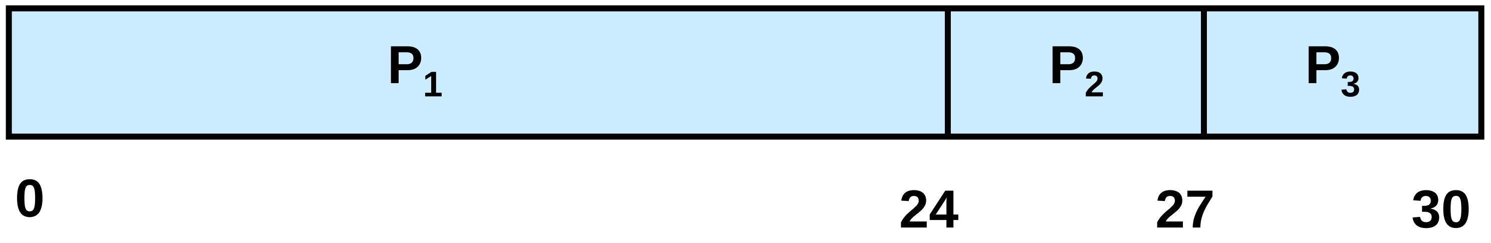
First-Come, First-Served (FCFS) Scheduling

- Consider processes arrive at time 0

Process	Burst Time
P_1	24
P_2	3
P_3	3

Turnaround Time = Completion Time – Arrival Time

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



- Turnaround Time P_1 = 24; P_2 = 27; P_3 = 30
- Average turnaround time: (24+27+30)/3 = 27ms

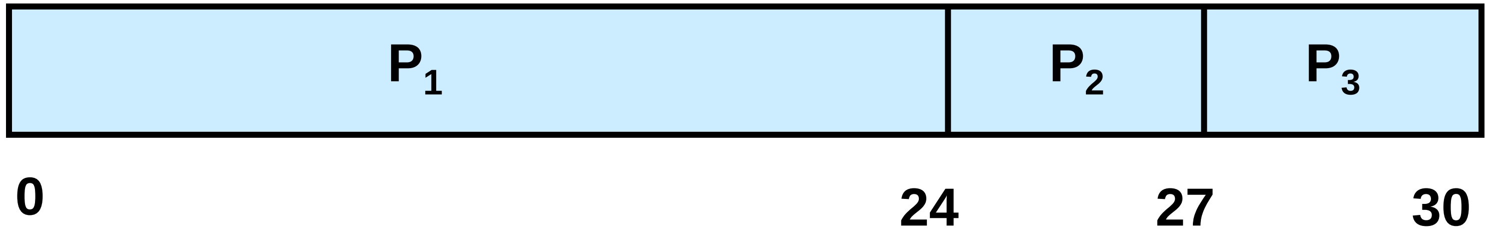
First-Come, First-Served (FCFS) Scheduling

- Consider processes arrive at time 0

Waiting Time = Turnaround Time – Burst Time

Process	Burst Time
P_1	24
P_2	3
P_3	3

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



- Turnaround Time P_1 = 24; P_2 = 27; P_3 = 30
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0+24+27)/3 = 17ms

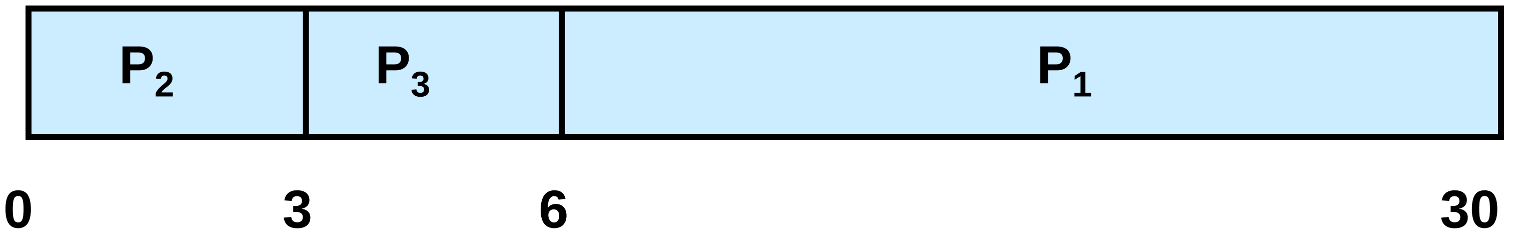
FCFS Scheduling (Cont.)

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

Turnaround Time = Completion Time – Arrival Time

- The Gantt chart for the schedule:



- Turnaround Time for $P_1 = 30$; $P_2 = 3$; $P_3 = 6$
- Average Turnaround time: (30+3+6)/3 = 13ms
- *Problems:*
  - *Convoy effect* (short processes behind long processes)
  - Non-preemptive -- not suitable for time-sharing systems

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya TRUST

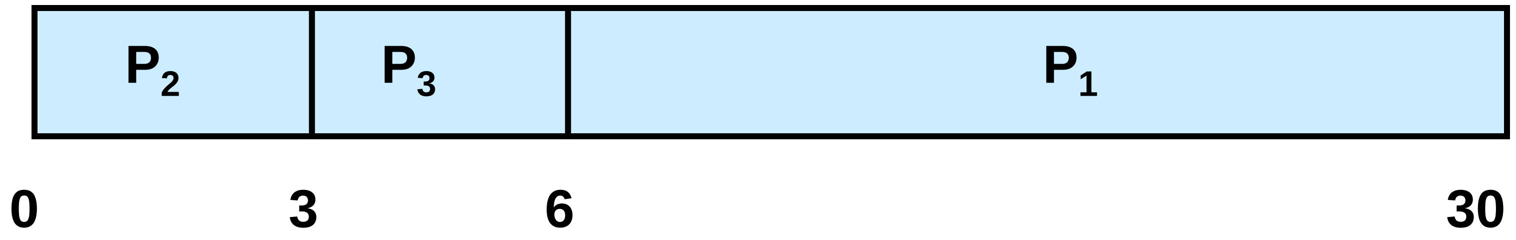
FCFS Scheduling (Cont.)

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order P_2 , P_3 ,

Waiting Time = Turnaround Time – Burst Time

- The Gantt chart for the schedule:



- Turnaround Time for $P_1 = 30$; $P_2 = 3$; $P_3 = 6$
- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: (30+3+6)/3 = 13ms

FCFS Scheduling (Cont.)

- *Problems:*
  - *Convoy effect* (short processes behind long processes)
  - Non-preemptive -- not suitable for time-sharing systems

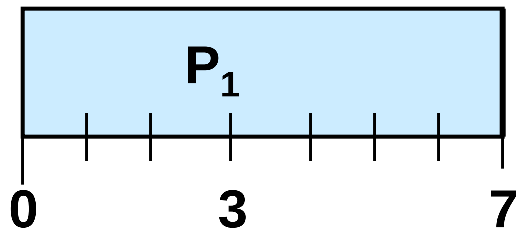
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. The CPU is assigned to the process with the smallest CPU burst (FCFS can be used to break ties).
- Two schemes:
  - nonpreemptive
  - preemptive – Also known as the Shortest-Remaining-Time-First (SRTF).
- Non-preemptive SJF is *optimal* if all the processes are ready simultaneously– gives minimum average waiting time for a given set of processes.

Example for 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

- At time 0, P_1 is the only process, so it gets the CPU and runs to completion

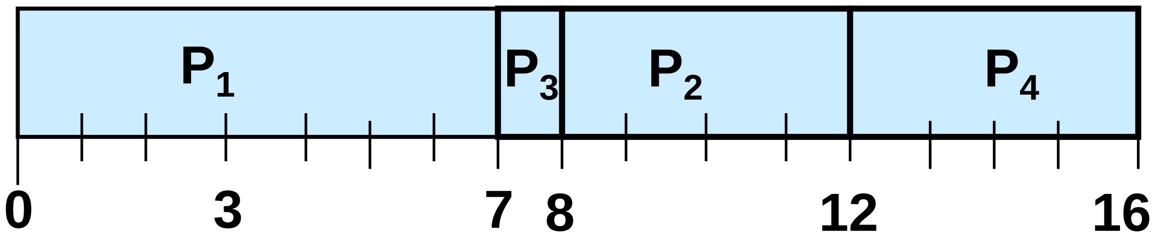


Example for Non-Preemptive SJF

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

Turnaround Time = Completion Time – Arrival Time

- Once P_1 has completed the queue now holds P_2, P_3 and P_4



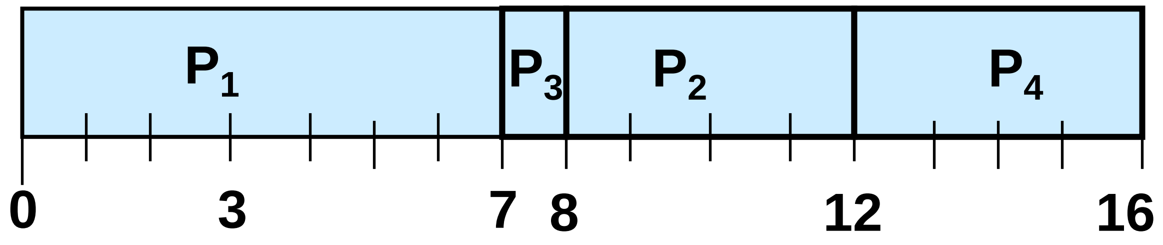
- P_3 gets the CPU first since it is the shortest. P_2 then P_4 get the CPU in turn (based on arrival time)
- Turnaround Time for process p1= 7, p2= 10. p3=4, p4=11
- Average Turnaround time : (7+10+4+11)/4 = 8ms

Example for 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

Waiting Time = Turnaround Time – Burst Time

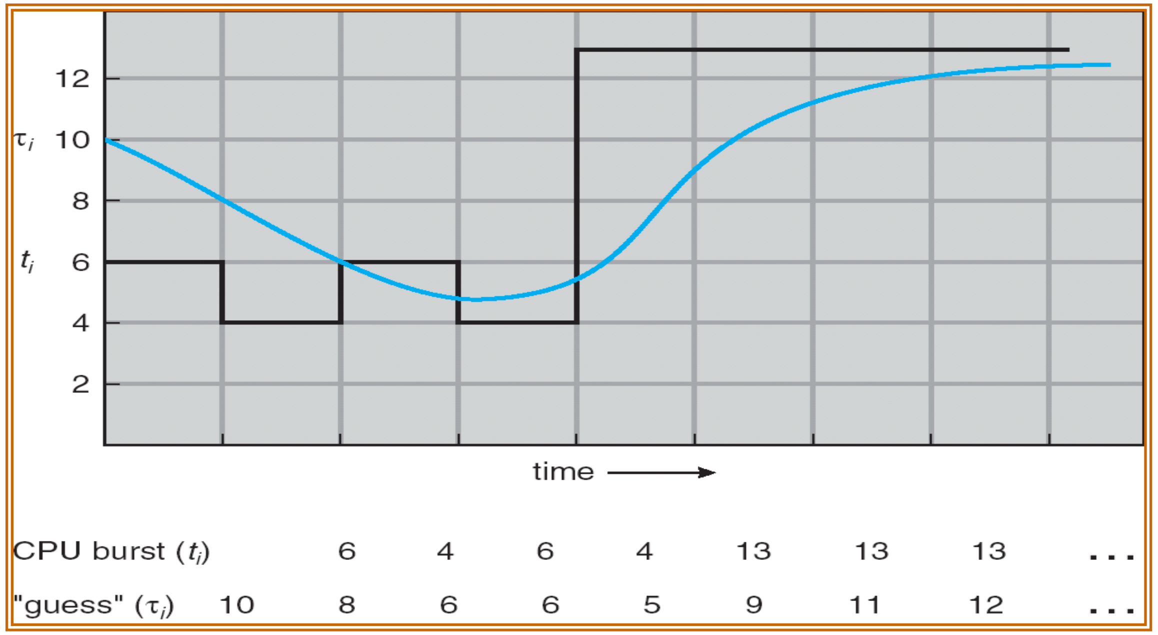
- Once P_1 has completed the queue now holds P_2, P_3 and P_4



- Turnaround Time for process p1= 7, p2= 10. p3=4, p4=11
- Waiting Time for process p1 = 0, p2=6,p3=3,p4=7

Estimating the Length of Next CPU Burst

- Problem with SJF: It is very difficult to know exactly the length of the next CPU burst.
- Idea: Based on the observations in the recent past, we can try to *predict*.
- *Exponential averaging: n*th CPU burst = t_n; the average of all past bursts τ_n, using a weighting factor $0 <= \alpha <= 1$, the next CPU burst is: $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \tau_n$.



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

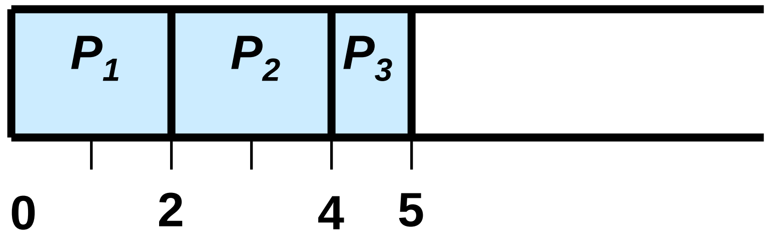
Shortest Remaining Time First (SRTF)

- Shortest Remaining Time First (SRTF) scheduling algorithm is basically a preemptive mode of the Shortest Job First (SJF) algorithm in which jobs are scheduled according to the shortest remaining time.
- In this scheduling technique, the process with the shortest burst time is executed first by the CPU, but the arrival time of all processes need not be the same.
- If another process with the shortest burst time arrives, then the current process will be preempted, and a newer ready job will be executed first.
- Also called as Shortest Remaining Time Next (SRTN)

Example for Preemptive SJF (SRTF)

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

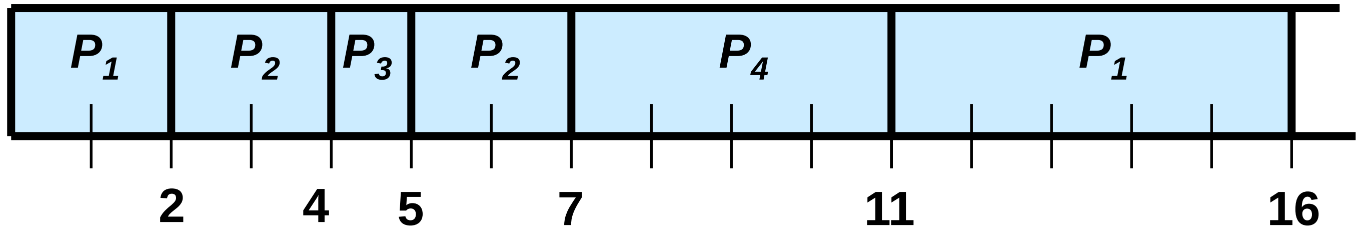
- Time 0 – P_1 gets the CPU   Ready = [(P_1,7)]
- Time 2 – P_2 arrives – CPU has P_1 with time=5, Ready = [(P_2,4)] – P_2 gets the CPU
- Time 4 – P_3 arrives – CPU has P_2 with time = 2, Ready = [(P_1,5),(P_3,1)] – P_3 gets the CPU



Example for Preemptive SJF (SRTF)

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

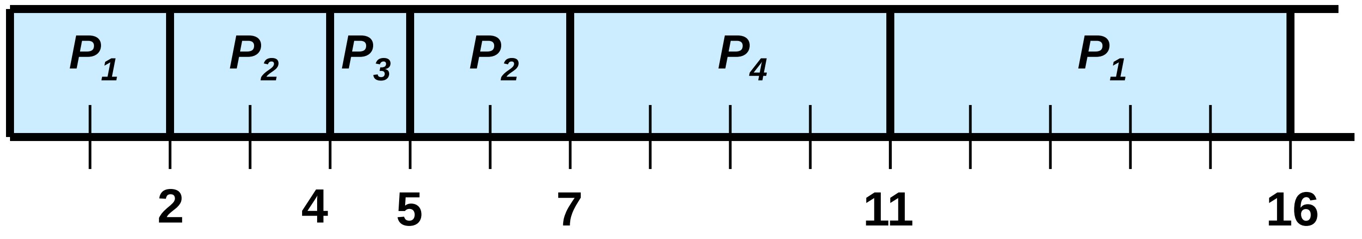
- Time 5 – P_3 completes and P_4 arrives - Ready = [(P_1,5),(P_2,2),(P_4,4)] – P_2 gets the CPU
- Time 7 – P_2 completes – Ready = [(P_1,5),(P_4,4)] – P_4 gets the CPU
- Time 11 – P_4 completes, P_1 gets the CPU



Example for Preemptive SJF (SRTF)

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

Turnaround Time = Completion Time – Arrival Time

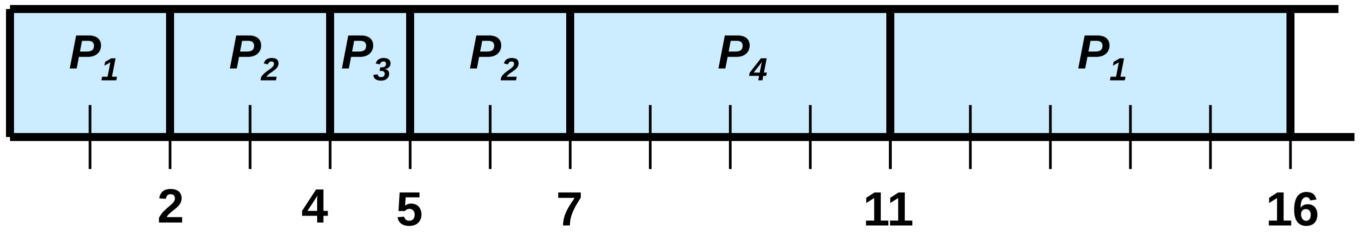


- Turnaround Time p1=16 ,p2=5,p3=1,p4=6
- Average Turnaround time = (16 + 5 + 1 +6)/4 = 7ms

Example for Preemptive SJF (SRTF)

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

Waiting Time = Turnaround Time – Burst Time



- Turnaround Time p1=16 ,p2=5,p3=1,p4=6
- Waiting Time p1=9 ,p2=1,p3=0,p4=2
- Average waiting time time = (9 + 1 + 0 +2)/4 = 3ms

Priority-Based 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).
- When a process arrives at the ready queue,
  - **the priority is compared with priority of the current running process.**
- It can be
  - **pre-emptive**
  - **non pre-emptive**

Priority-Based Scheduling (cont…)

Scenario:
If a newly arrived process has a higher priority than the currently running process.

Characteristics:

- **Preemptive Priority Scheduling Algorithm:**
  - The CPU is preempted, and the currently running process is moved to the ready queue.
  - The newly arrived process is then scheduled for execution.
- **Non-Preemptive Priority Scheduling Algorithm:**
  - The newly arrived process is placed at the tail of the ready queue.
  - The currently running process continues execution until it finishes, after which the scheduler picks the next process.

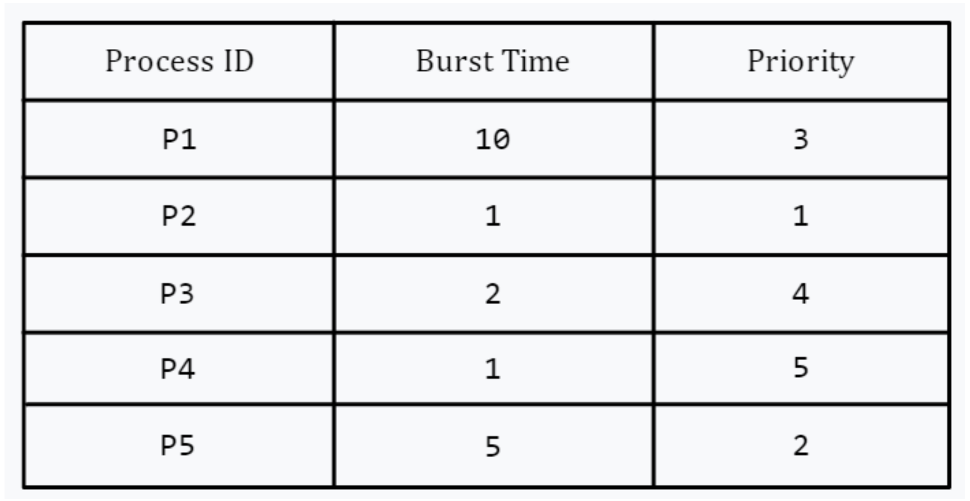
Priority-Based Scheduling (cont…)

- SJF is a special case of priority scheduling:
  - process priority = the *inverse of remaining CPU time*
  - **The larger the CPU burst, the lower the priority and vice versa**

- **Equal priority processes are scheduled in FCFS order**
  - FCFS can be used to break ties.

Example for Priority-based Scheduling

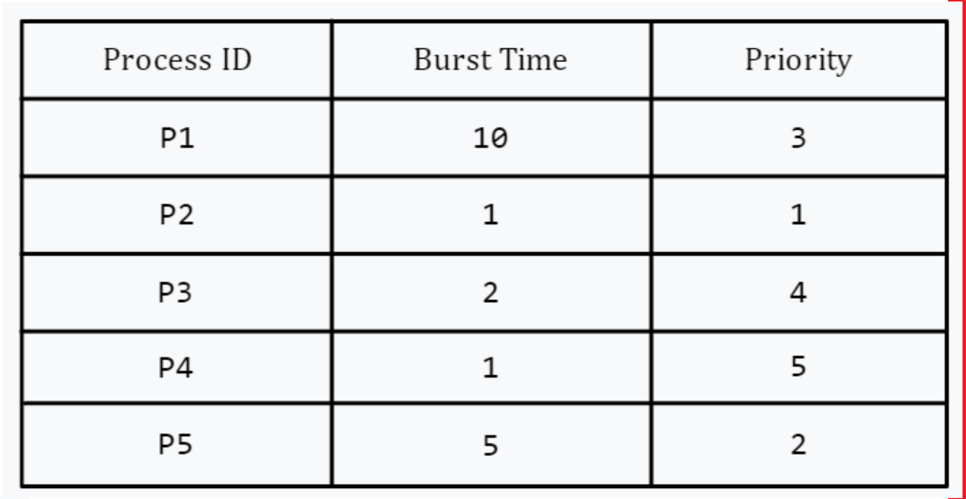
- Consider the following set of processes, assumed to have arrived at time 0, in the order P1, P2, P3, P4, P5, with the length of the CPU burst given in milliseconds.



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

- Low number represents the high priority.

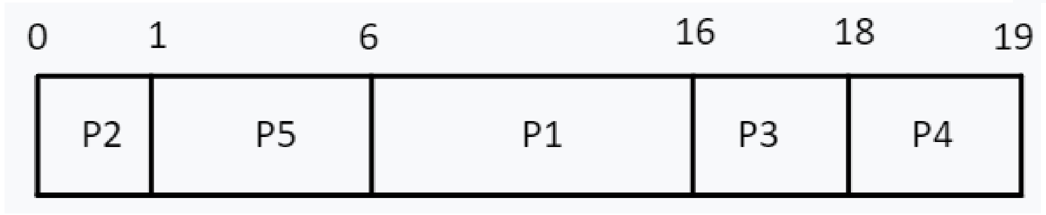
Example for Priority-based Scheduling



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

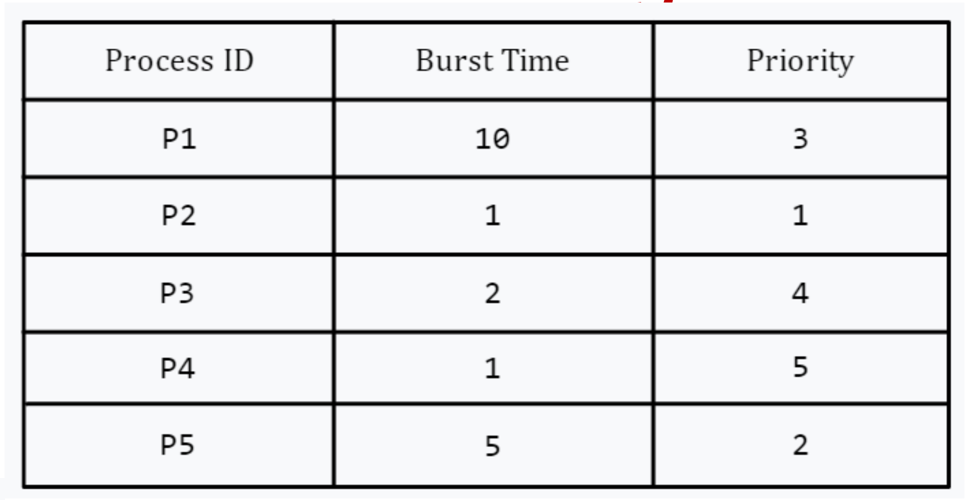
Turnaround Time = Completion Time – Arrival Time

- Gantt Chart



- Turnaround Time p1=16,p2=1,p3=18,p4=19,p5=6
- Average Turnaround Time = (16+1+18+19+6)/5=12ms

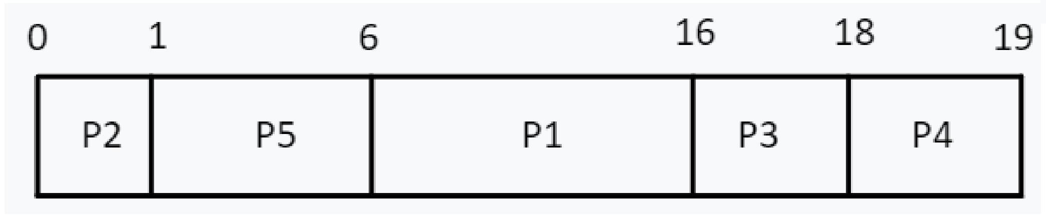
Example for Priority-based Scheduling



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

Waiting Time = Turnaround Time – Burst Time

- Gantt Chart



- Turnaround Time p1=16,p2=1,p3=18,p4=19,p5=6
- Waiting Time p1=6,p2=0,p3=16,p4=18,p5=1
- Average Turnaround Time = (6+0+16+18+1)/5=8.2ms

Priority-Based Scheduling (Cont.)

- Problem: Indefinite Blocking (or Starvation) –
  - low priority processes may never execute.
- One solution: *Aging* – as time progresses, increase the priority of the processes that wait in the system for a long time.
- Priority Assignment
  - Internal factors: timing constraints, memory requirements, the ratio of average I/O burst to average CPU burst….
  - External factors: Importance of the process, financial considerations, hierarchy among users…

Round Robin Scheduling

- It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes.
- A small unit of time, called a time quantum or time slice, is defined.
- A time quantum is generally from 10 to 100 milliseconds in length.
- Every process is assigned a time quantum for its execution, allowing it to execute only for that time ***quantum.***
- The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1-time quantum.

Round Robin (RR) Scheduling

- Each process gets a small unit of CPU time (*time quantum*). After this time has elapsed, the process is preempted and added to the end of the ready queue.
- Newly-arriving processes (and processes that complete their I/O bursts) are added to the end of the ready queue
- If there are n processes in the ready queue and the time quantum is q, then no process waits more than $(n-1)q$ time units.

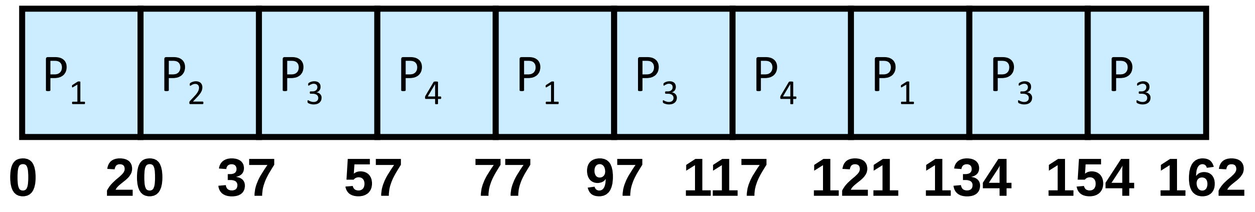
Example for Round-Robin

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- Time Quantum given as 20

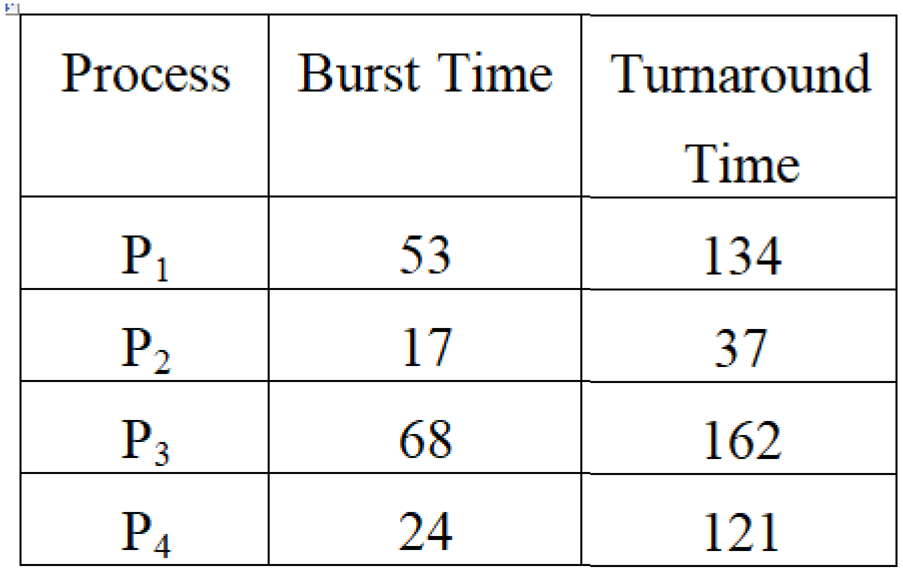
    Turnaround Time = Completion Time – Arrival Time

- The Gantt chart:



- Turn around time = 134+37+162+121=454/4=113.5

Example for Round-Robin

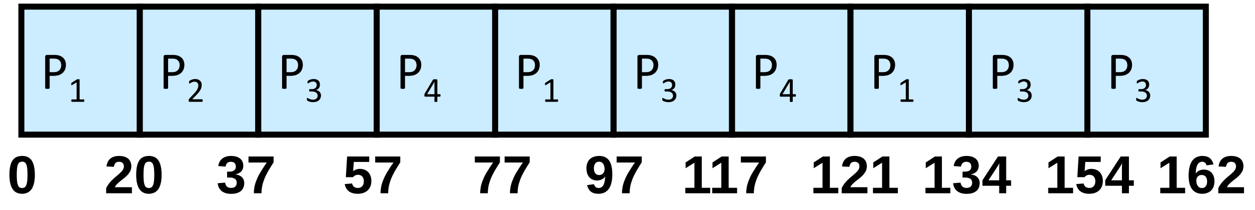


Process	Burst Time	Turnaround Time
P_1	53	134
P_2	17	37
P_3	68	162
P_4	24	121

- Time Quantum given as 20

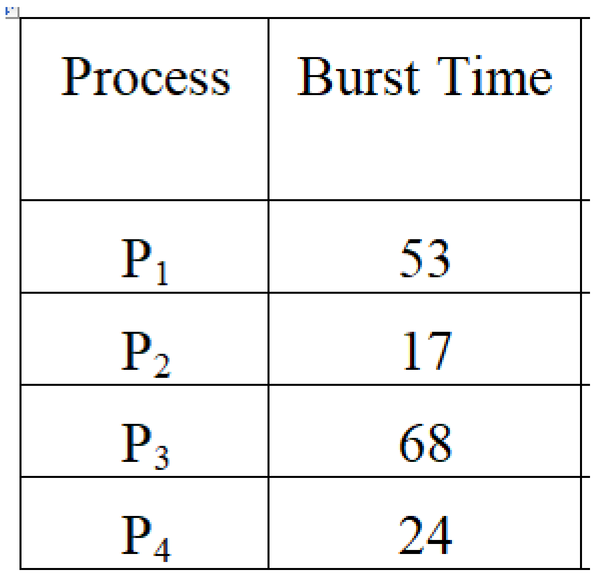
Waiting Time = Turnaround Time – Burst Time

- The Gantt chart:



- Average wait time = (81+20+94+97)/4 = 73
- Typically, higher average turnaround time (amount of time to execute a particular process) than SJF, but better *response time* (amount of time it takes from when a request was submitted until the first response is produced).

Example for Round-Robin

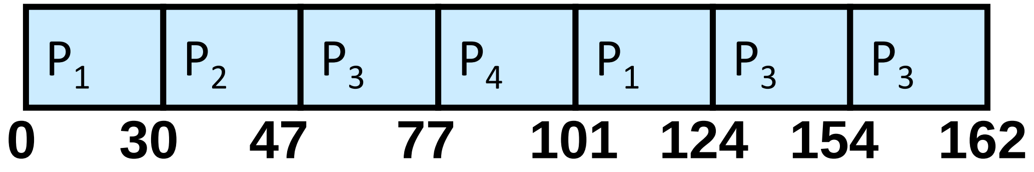


Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- Time Quantum = 30

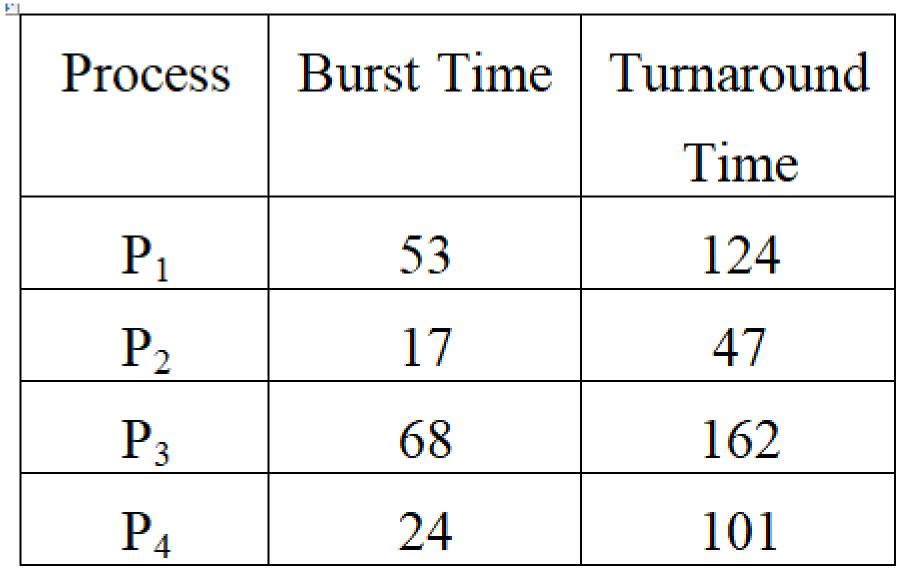
  Turnaround Time = Completion Time – Arrival Time

- The Gantt chart



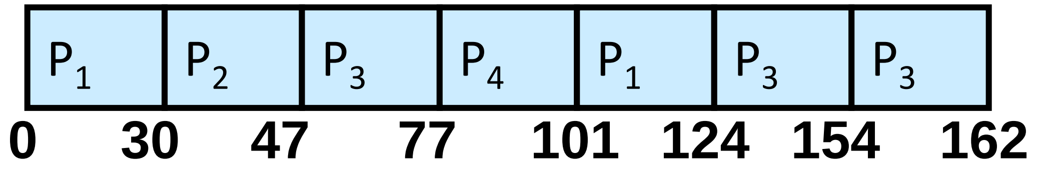
- Turn around Time = (124+47+162+101)=434/4 = 108.5

Example for Round-Robin



Process	Burst Time	Turnaround Time
P_1	53	124
P_2	17	47
P_3	68	162
P_4	24	101

The Gantt chart: (Time Quantum = 30)

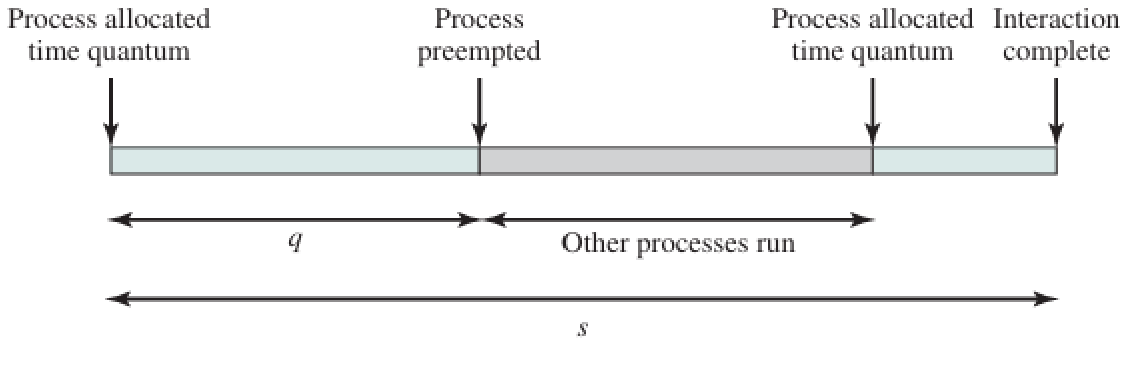


Waiting Time = Turnaround Time – Burst Time

- Average wait time = (71+30+94+77)/4 = 68

Effect of time quanta

- With round robin, the principal design issue is the length of the time quantum, or slice, to be used.

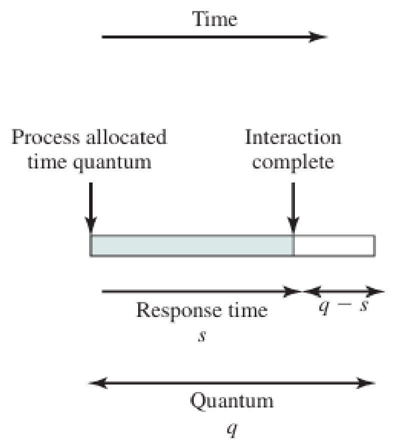


(b) Time quantum less than typical interaction

- If the quantum is very short, then short processes will move through the system relatively quickly.
- On the other hand, there is processing **over head** involved in handling the clock interrupt and performing the scheduling and dispatching function.
- Thus, very **short time quanta should be avoided**.

Effect of time quanta

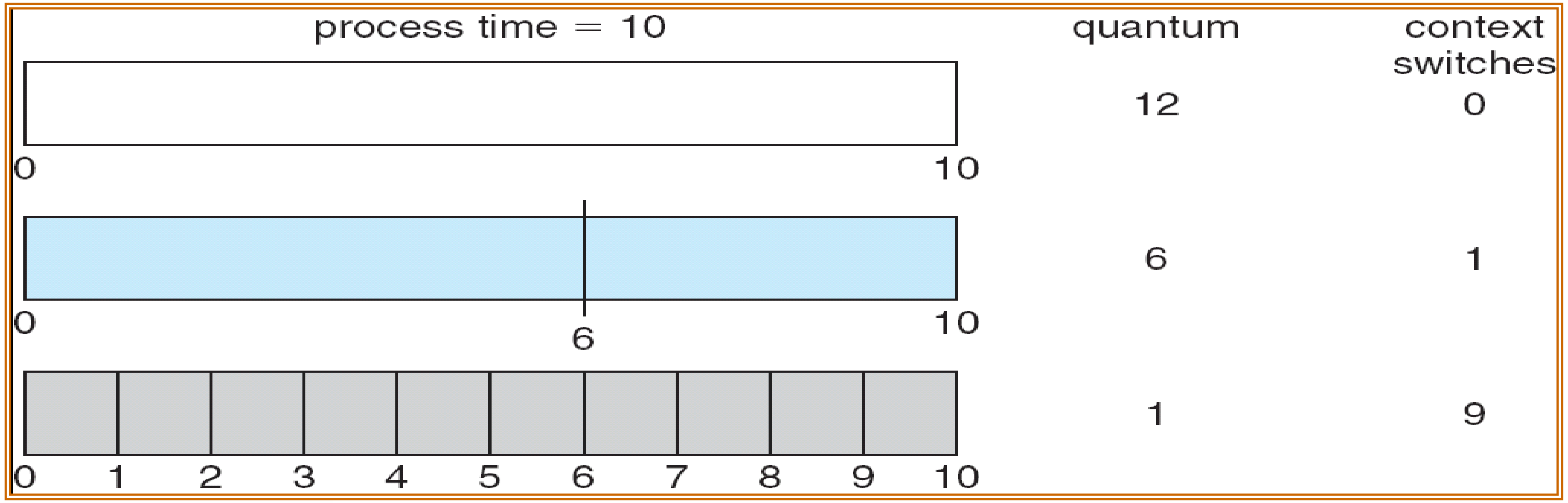
- When the time quantum is greater than the typical interaction time, it means that each process gets more time to execute before being switched out.
- This can lead to fewer context switches, which might improve efficiency for CPU-bound processes but could also increase response time for interactive processes.



(a) Time quantum greater than typical interaction

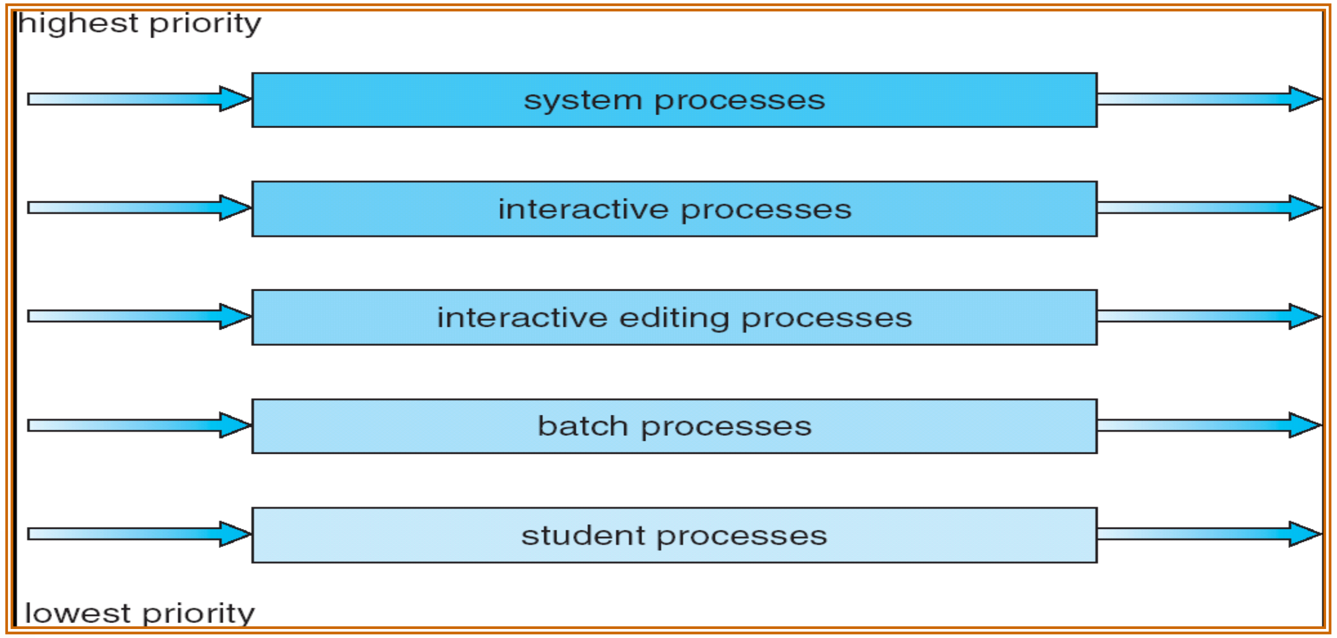
Choosing a Time Quantum

- The effect of quantum size on context-switching time must be carefully considered.
- The time quantum must be large with respect to the context-switch time
- Modern systems use quanta from 10 to 100 msec with context switch taking < 10 msec

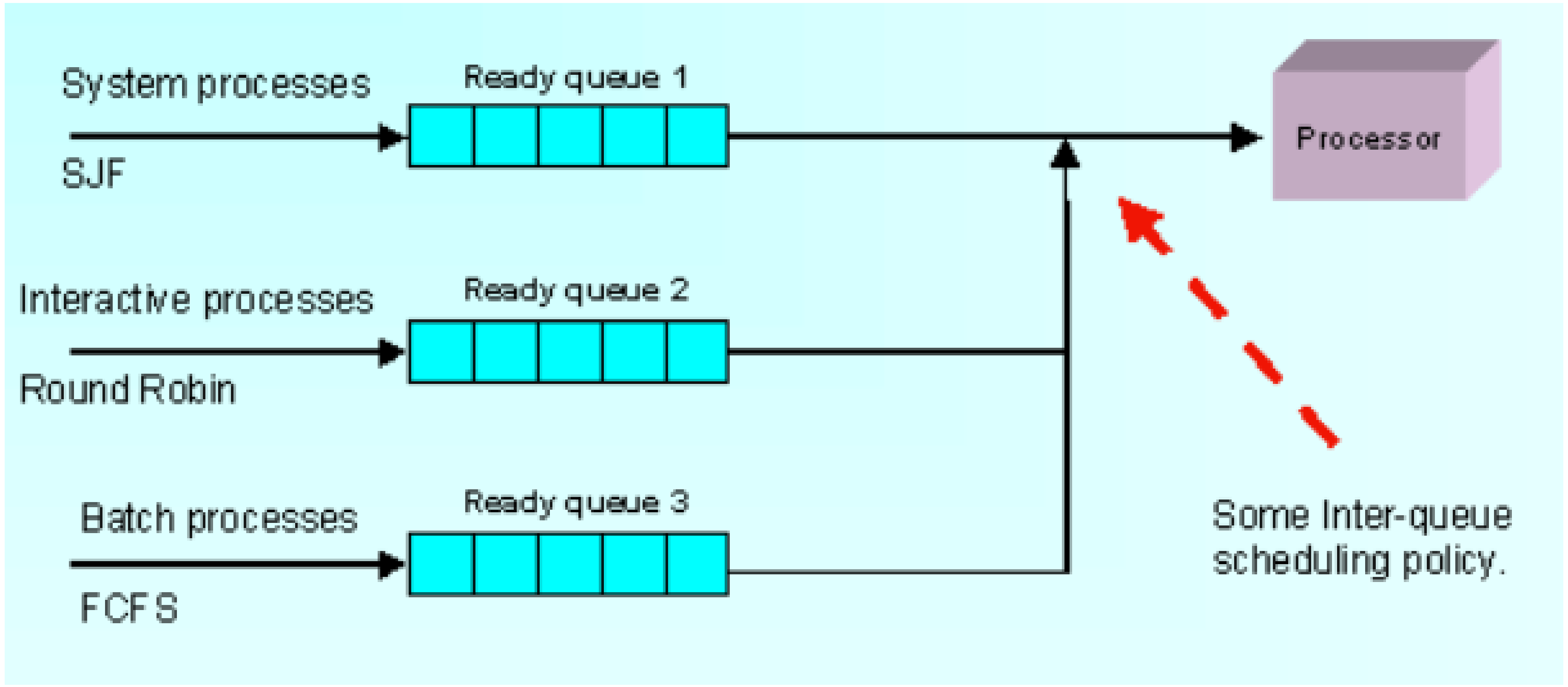


Multilevel Queue

- Ready queue is partitioned into separate queues:
  - Foreground (interactive)
  - Background (batch) processes;
- Each queue has its own scheduling policy



Multilevel Queue Scheduling

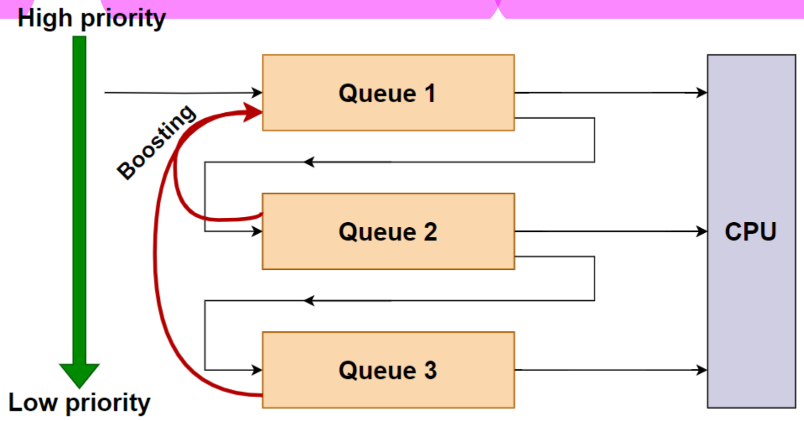


Multilevel Queue Scheduling

- Each queue may have has its own scheduling algorithm: Round Robin, FCFS, SJF…
- In addition, (meta-)scheduling must be done between the queues.
  - Fixed priority scheduling (i.e. serve first the queue with highest priority). Problems?
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; for example, 50% of CPU time is used by the highest priority queue, 20% of CPU time to the second queue, and so on..
  - Also, need to specify which queue a process will be put to when it arrives to the system and/or when it starts a new CPU burst.

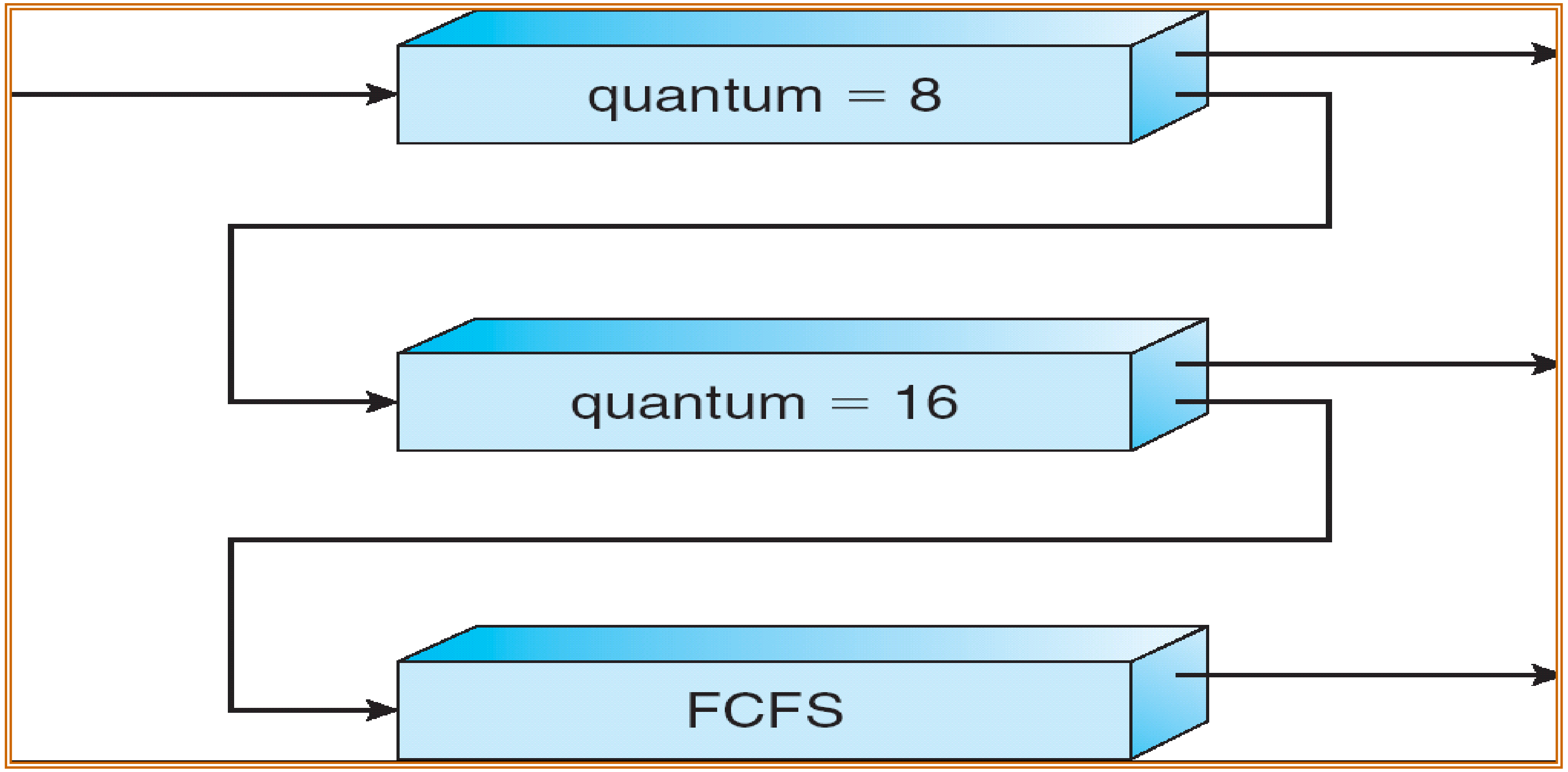
Multilevel Feedback Queue (MLFQ)

- In a multi-level queue-scheduling algorithm, processes are permanently assigned to a queue.
- Idea: Allow processes to move among various queues.



- Uses multiple queues with varying priorities to manage process execution.
- It dynamically adjusts priorities based on process behavior, promoting or demoting processes between queues.

Multilevel Feedback Queue



Multilevel Feedback Queue

- Multilevel feedback queue scheduler is 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

- The scheduler can be configured to match the requirements of a specific system.

Characteristics of Various Scheduling Policies

Scheduling Policy	Selection Function	Decision Mode	Throughput	Response Time	Overhead	Effect on Processes	Starvation
First-Come, First-Served (FCFS)	Arrival time	Non-preemptive	Low for long processes	High for long processes	Minimal	Simple, long wait times	No, but convoy effect
Shortest Job First (SJF)	Shortest next CPU burst	Non-preemptive	High	Low for short processes	Requires knowledge of process length	Efficient for short processes	Yes, long processes may starve
Shortest Remaining Time First (SRTF)	Shortest remaining CPU burst	Preemptive	High	Low for short processes	High, frequent context switching	Efficient for short processes	Yes, long processes may starve

Characteristics of Various Scheduling Policies

Scheduling Policy	Selection Function	Decision Mode	Throughput	Response Time	Overhead	Effect on Processes	Starvation
Priority Scheduling	Highest priority	Preemptive or Non-preemptive	Varies based on priority	Low for high-priority processes	Requires priority assignment	High-priority processes favored	Yes, low-priority processes may starve
Round Robin (RR)	Time quantum (fixed time slice)	Preemptive	Moderate	Moderate	High, frequent context switching	Fair time-sharing	No, each process gets CPU time

Characteristics of Various Scheduling Policies

Scheduling Policy	Selection Function	Decision Mode	Throughput	Response Time	Overhead	Effect on Processes	Starvation
Multilevel Queue	Based on queue priority	Preemptive or Non-preemptive	Varies based on queue configuration	Varies	High, managing multiple queues	Processes categorized into different queues	Yes, lower-priority queues may starve
Multilevel Feedback Queue	Based on aging and feedback	Preemptive	High	Low for interactive processes	Very high, complex management	Dynamic adjustment of process priority	Reduced, processes can move between queues

Question ?