Operating Systems

Lecture – 8-10

CPU Scheduling Algorithms

Dr. Shamim Akhter

Problem

- You are the cook at the State St. Diner
  - Customers continually enter and place their orders
  - Dishes take varying amounts of time to prepare
- What is your goal?
- Which strategy achieves this goal?

Correspondence **DINER** - Multitasking Operating System

- Ordering Manager ↔ CPU Scheduler
- Cook ↔ CPU
- Client ↔ User
- Order ↔ Process
- Dish category ↔ CPU/I-O Burst

CPU/I-O Burst

- Process alternates between CPU and I/O bursts
  - **CPU-bound** jobs: Long CPU bursts

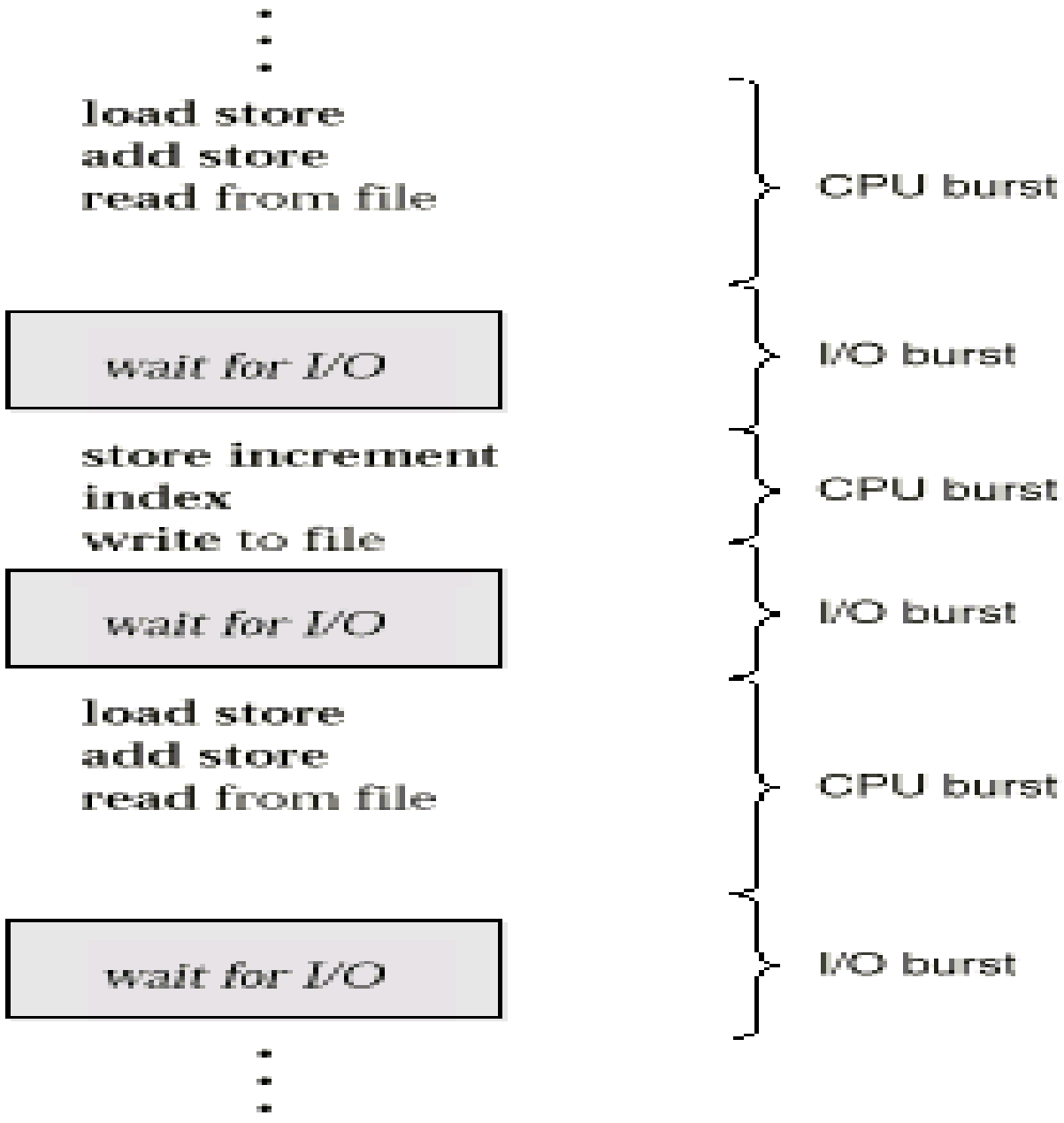
  - **I/O-bound**: Short CPU bursts

  - **I/O burst** = process idle, switch to another "for free"

CPU – I/O Burst Cycle

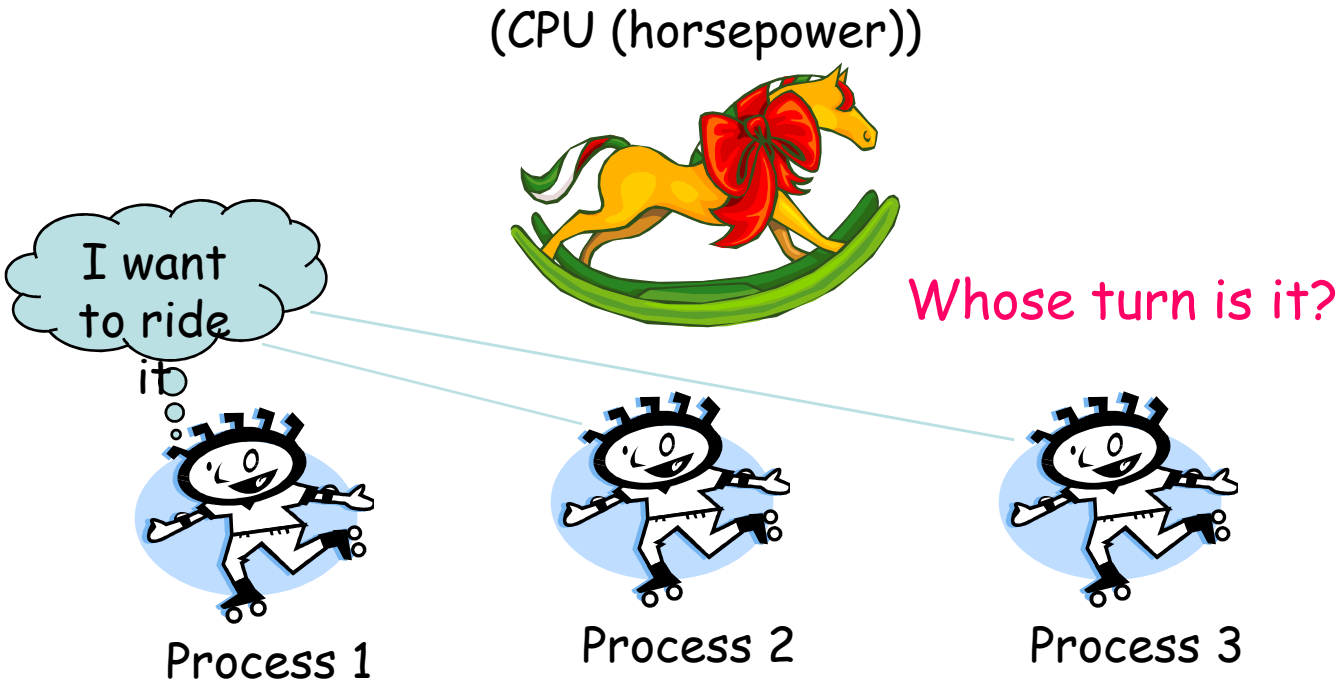
- Process execution consists of a cycle of CPU execution and I/O wait
  - Processes move back & forth between these two states
- A process execution begins with a CPU burst, followed by an I/O burst and then another CPU burst and so on
- An alternating sequence of CPU & I/O bursts is shown on next slide

Alternating Sequence of CPU and I/O Bursts



Scheduling

- Deciding which process/thread should occupy a resource (CPU, disk, etc.)



CPU Scheduler

- When CPU becomes idle
  - OS must select one of the processes in the Ready Queue to be executed.
- CPU scheduler
  - Is the OS code that implements the CPU scheduling algorithm .
  - Selects a process from the processes in memory that are ready to execute (from Ready Queue), and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state (e.g. when time slice of a process expires or an interrupt occurs)
  3. Switches from waiting to ready state. (e.g. on completion of I/O)
  4. On arrival of a new process
  5. Terminates

Dispatcher

- An important component involved in CPU scheduling
- The OS code that
  - takes the CPU away from the current process and hands it over to the newly scheduled process .
- Dispatcher gives control of the CPU to the process selected by the short-term scheduler
- Dispatcher performs following functions:
  - 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 executing another

Preemptive vs Non Preemptive Process

Non-preemptive

✓Process runs until voluntarily relinquish CPU

- process blocks on an event (e.g., I/O or synchronization)
- process terminates
- process periodically calls the yield() system call to give up the CPU

✓Only suitable for domains where processes can be trusted to relinquish the CPU

Preemptive

- The scheduler actively interrupts and reschedules an executing process
- Required when applications cannot be trusted to yield
- Incurs some overhead

Scheduling Criteria

- CPU utilization – percentage of time the CPU is not idle.
- Throughput – completed processes per unit of time
- Waiting time – time spent in the ready queue
  - For Non preemptive Algos = Starting.Time – Arrival.Time
  - For Preemptive Algos = Finish.Time – Arrival.Time – Burst.Time
- Turnaround time – amount of time to execute a particular process.
  - Finish.Time – Arrival.Time
- Response time – response latency,
  - amount of time it takes from when a request was submitted until the first response is produced, **not** output (for time-sharing environment)

The Perfect Scheduler

- Max CPU utilization= keep all devices busy
- Max throughput =Maximize jobs/time
- Min turnaround time
- Min waiting time
- Min response time =latency, first response
- Fairness: everyone makes progress, no one starves

Single CPU–Scheduling Algorithms

- First Come First Serve (FCFS)
- Smallest Next CPU Burst
  - Shortest Job First (SJF)
  - Shortest Remaining Time First (SRTF)
- Round Robin
- Priority Scheduling

First Come First Serve

- Simplest CPU scheduling algorithm
- Non preemptive
- The process that requests the CPU first is allocated the CPU first
- Implemented with a FIFO queue.
  - when a process enters the Ready Queue, its PCB is linked on to the tail of the Queue. When the CPU is free it is allocated to the process at the head of the Queue.

- **Limitations**
  - FCFS favor long processes as compared to short ones. (Convoy effect)
  - Average waiting time depends on arrival order
  - Average waiting time is often quite long
  - FCFS is non-preemptive, so it is trouble some for time sharing systems

Convoy Effect

"A convoy effect happens when a set of processes need to use a resource for a short time, and one process holds the resource for a long time, blocking all of the other processes. Causes poor utilization of the other resources in the system"

FCFS – Example

The process that enters the ready queue first is scheduled first, regardless of the size of its next CPU burst

Example:	Process	Burst Time
	P1	24
	P2	3
	P3	3

Suppose that processes arrive into the system in the order:
P1, P2 , P3

FCFS – Example

Processes are served in the order: P1, P2, P3

The **Gantt Chart** for the schedule is:

Waiting times P1 = 0; P2 = 24; P3 = 27
Average waiting time: (0+24+27)/3 = 17

FCFS – Example

Suppose that processes arrive in the order: P2 , P3 , P1 .
The Gantt chart for the schedule is:

Waiting time for P1 = 6; P2 = 0; P3 = 3
Average waiting time: (6 + 0 + 3)/3 = 3

FCFS – Example

Process	Duration/B.T	Order	Arrival Time
P1	24	1	0
P2	3	2	3
P3	4	3	4

The final schedule:

P1 waiting time: 0-0
P2 waiting time: 24-3
P3 waiting time: 27-4

The average waiting time:
(0+21+23)/3 = 14.667

FCFS – Example

- Draw the graph (Gantt chart) and compute average waiting time for the following processes using FCFS Scheduling algorithm.

Process	Arrival time	Burst Time
P1	1	16
P2	5	3
P3	6	4
P4	9	2

SJF & SRTF Scheduling…

- When the CPU is available it is assigned to the process that has the smallest next CPU burst.
- If two processes have the same length next CPU bursts, FCFS scheduling is used to break the tie.

Comes in two flavors

- **Shortest Job First (SJF)**
  - It's a non preemptive algorithm.
- **Shortest Remaining Time First (SRTF)**
  - It's a Preemptive algorithm.

SJF Example

Process	Duration/B.T	Order	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0

P4 waiting time: 0-0
P1 waiting time: 3-0
P3 waiting time: 9-0
P2 waiting time: 16-0

The total running time is: 28
The average waiting time (AWT):
(0+3+9+16)/4 = 7 time units

SRTF Example

Process	Duration	Order	Arrival Time
P1	10	1	0
P2	2	2	2

P1 waiting time: 4-2 =2
P2 waiting time: 0

The average waiting time (AWT):
(0+2)/2 = 1

Now run this using SJF!

SJF & SRTF – Example

Draw the graph (Gantt chart) and compute waiting time and turn around time for the following processes using **SJF** & **SRTF** Scheduling algorithm.

For SJF consider all processes arrive at time 0 in sequence P1, P2, P3, P4.

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

SJF & SRTF – Example

Draw the graph (Gantt chart) and compute waiting time and turn around time for the following processes using **SJF** & **SRTF** Scheduling algorithm.

Process	Arrival Time	Burst Time
P1	0	5
P2	1	2
P3	2	3
P4	3	1

SJF & SRTF – Example

Draw the graph (Gantt chart) and compute waiting time and turn around time for the following processes using **SJF** & **SRTF** Scheduling algorithm.

Process	Arrival Time	Burst Time
P1	0	9
P2	3	6
P3	6	2
P4	9	1

SJF & SRTF Scheduling

$100 QUESTION

How to compute the next CPU burst?

- This algorithm cannot be implemented at the level of short-term CPU scheduling. There is no way to know the length of the next CPU burst.
- One approach is to try to approximate.
- We may not *know* the length of the next CPU burst, but we may be able to **predict its value**.
- We expect that the next CPU burst will be similar in length to the previous ones.
  - Thus, by computing an approximation of the length of the next CPU burst, we can pick the process with the shortest predicted CPU burst.

SJF & SRTF Scheduling…

Exponential Averaging

Estimation based on historical data

t_n = Actual length of n^{th} CPU burst

τ_n = Estimate for n^{th} CPU burst

τ_{n+1} = Estimate for $n+1^{st}$ CPU burst

α, $0 \leq \alpha \leq 1$

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

Exponential Averaging…

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

- Plugging in value for τ_n, we get

$$\tau_{n+1} = \alpha t_n + (1- \alpha)[\alpha t_{n-1} + (1- \alpha)\tau_{n-1}]$$

$$\tau_{n+1} = \alpha t_n + (1- \alpha)\alpha t_{n-1} + (1- \alpha)^2 \tau_{n-1}$$

- Again plugging in value for τ_{n-1}, we get

$$\tau_{n+1} = \alpha t_n + (1- \alpha)\alpha t_{n-1} + (1- \alpha)^2 [\alpha t_{n-2} + (1- \alpha)\tau_{n-2}]$$

$$\tau_{n+1} = \alpha t_n + (1- \alpha)\alpha t_{n-1} + (1- \alpha)^2\, \alpha t_{n-2}\, (1- \alpha)^3 \tau_{n-2}$$

- Continuing like this results in

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

Exponential Averaging…

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

Lets take two extreme values of α

If $\alpha = 0$

$$\tau_{n+1} = \tau_n$$

Next CPU burst estimate will be exactly equal to previous CPU burst estimate.

If $\alpha = 1$

$$\tau_{n+1} = t_n$$

Next CPU burst estimate will be equal to previous actual CPU burst.

Exponential Averaging…

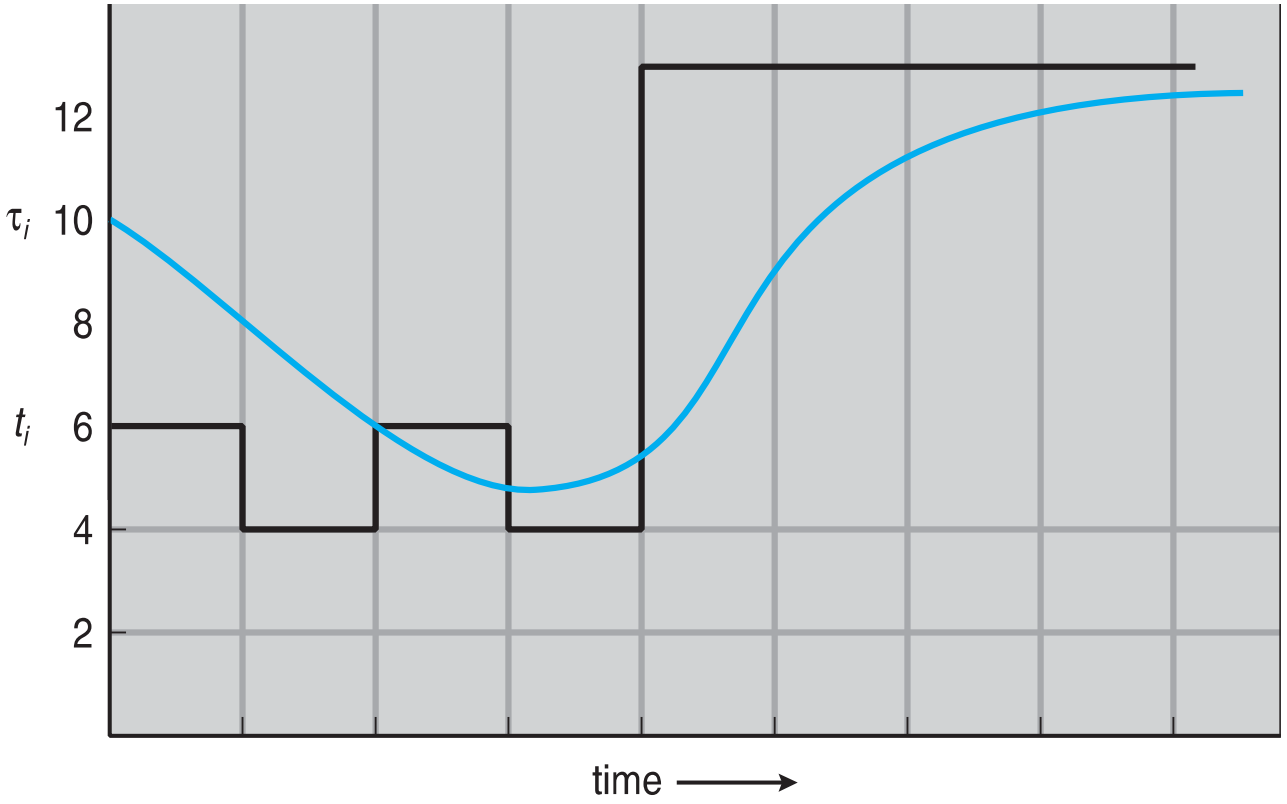
$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \alpha\, t_{n-1} + \ldots + (1 - \alpha)^j\, \alpha\, t_{n-j} + \ldots \qquad + (1 - \alpha)^{n+1}\, \tau_0$$

Typical value used for α is $\frac{1}{2}$. With this value, our $(n+1)^{st}$ estimate is

$$\tau_{n+1} = t_n/2 + t_{n-1}/2^2 + t_{n-2}/2^3 + t_{n-3}/2^4 + \ldots$$

Note that as we move back in time we are giving less weight-age to the previous CPU bursts. Older histories are given exponentially less weight-age

Prediction of the Length of the Next CPU Burst



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

$\tau_0 = 10$
$\tau_1 = .5*6 + .5 * 10 = 8$
$\tau_2 = .5*4 + .5 * 8 = 6$
$\tau_3 = .5*6 + .5 * 6 = 6$
$\tau_4 = .5*4 + .5 * 6 = 5$
$\tau_5 = .5*13 + .5 * 5 = 9$
…
…

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

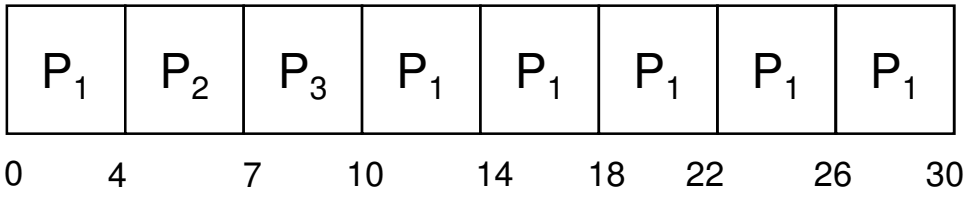
Round Robin (RR)

- 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

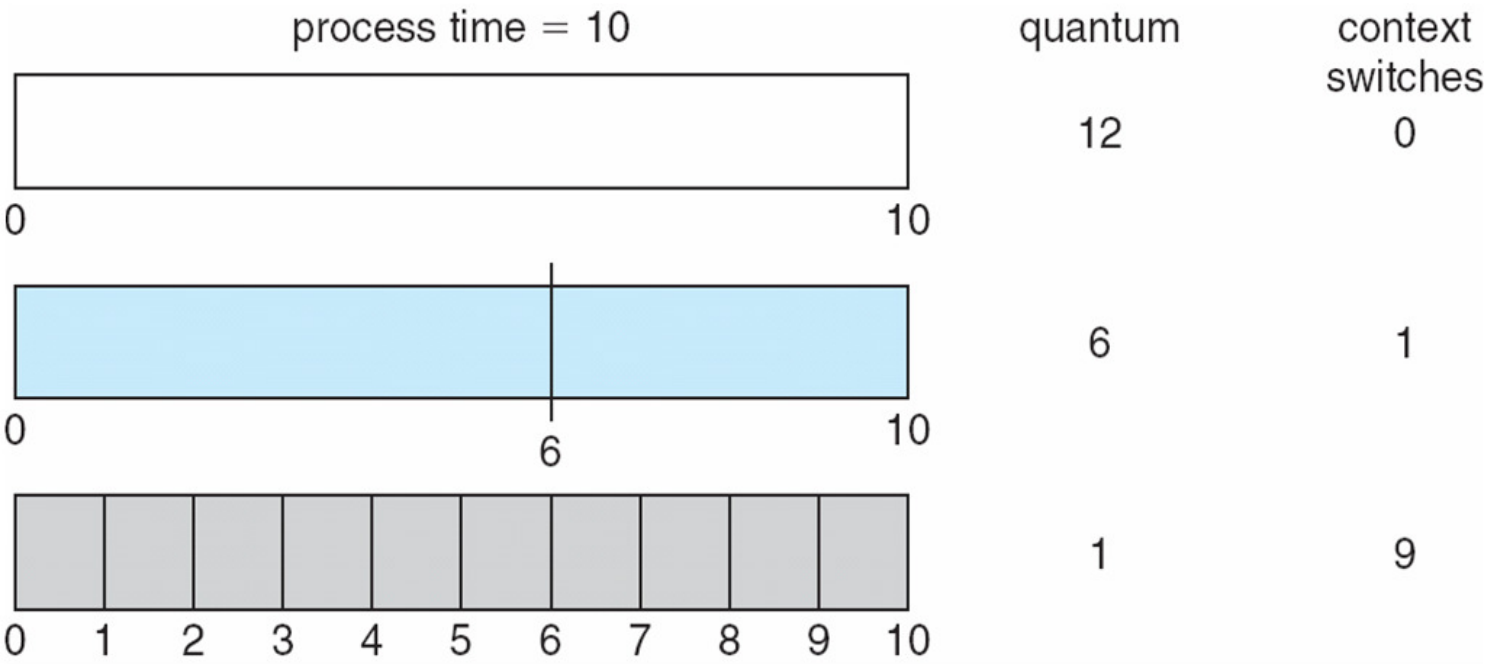
Process	Burst Time
P_1	24
P_2	3
P_3	3

- 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

Time Quantum and Context Switch Time



- One process of 10 time units
  - If q is 12, finish in less than 1 q
  - If q is 6, requires 2 q, one context swith
  - If q is 1, requires 10 q and 9 context switches

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

  -> Shortest Job, Highest Priority
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

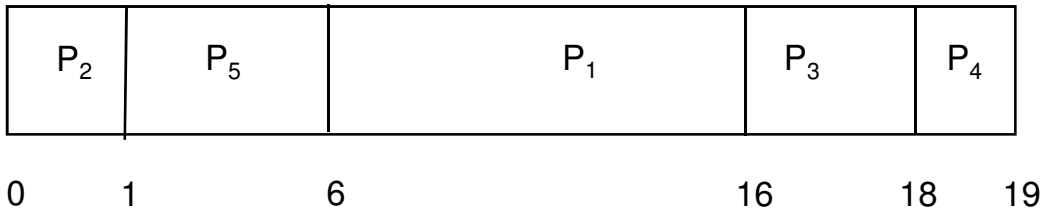
Priority Scheduling

- **Priority Scheduling**
  - Choose next job based on priority
  - For SJF, priority = expected CPU burst
  - Can be either preemptive or non-preemptive
- **Priority schedulers can approximate other scheduling algorithms**
  - **P = arrival time => FIFO**
  - **P = now - arrival time =>LIFO**
  - **P = job length => SJF**
- **Problem:**
  - Starvation: jobs can wait indefinitely
- **Solution to starvation**
  - Age processes: increase priority as a function of waiting time

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

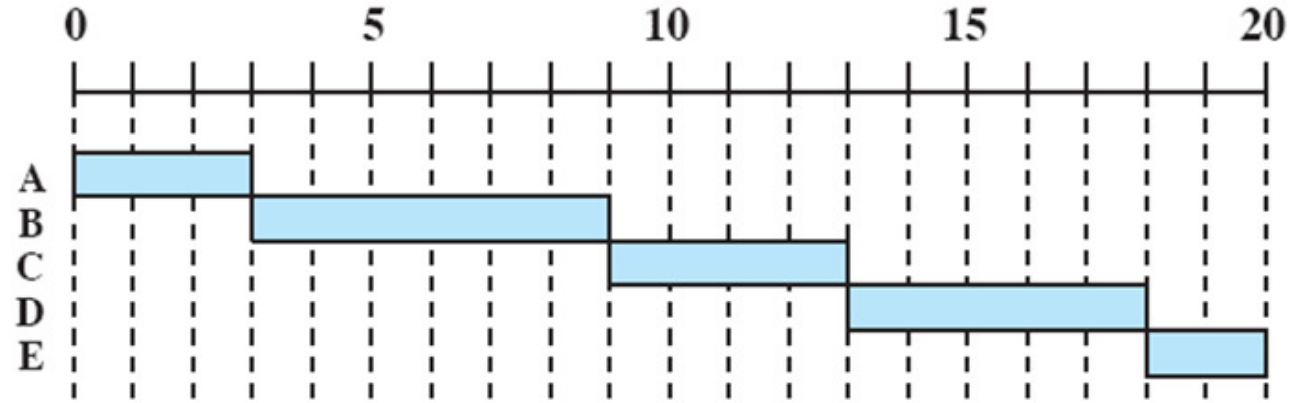
- Priority scheduling Gantt Chart



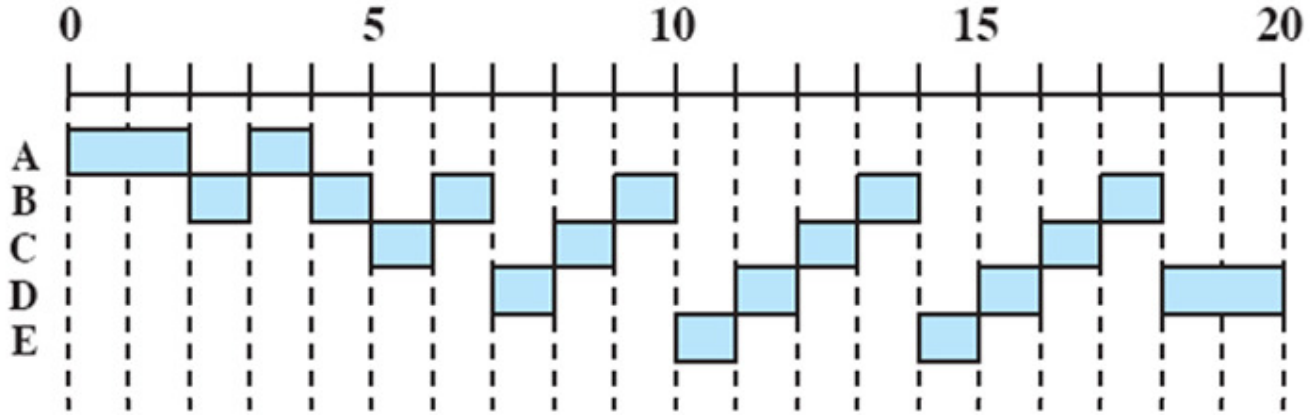
- Non-Preemptive Waiting time= 0+1+6+16+18=41
- Average waiting time = 8.2 msec

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

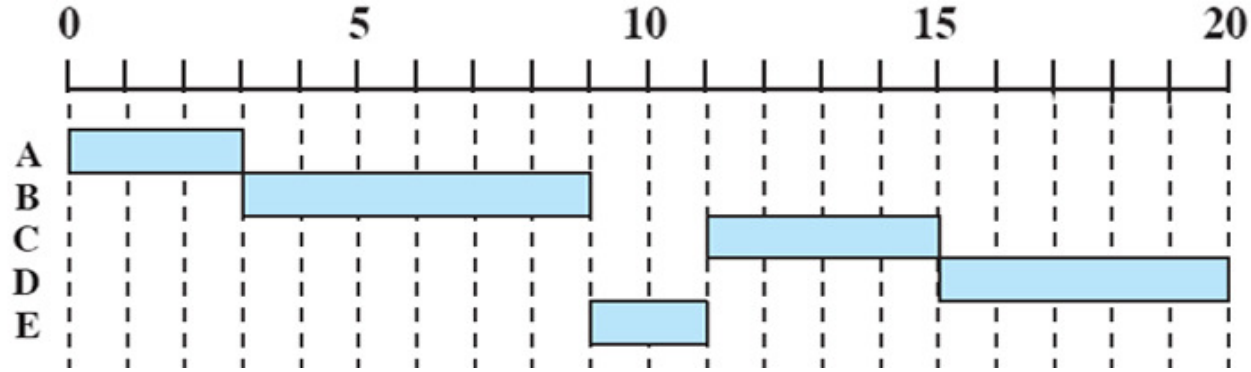
First-Come-First Served (FCFS)



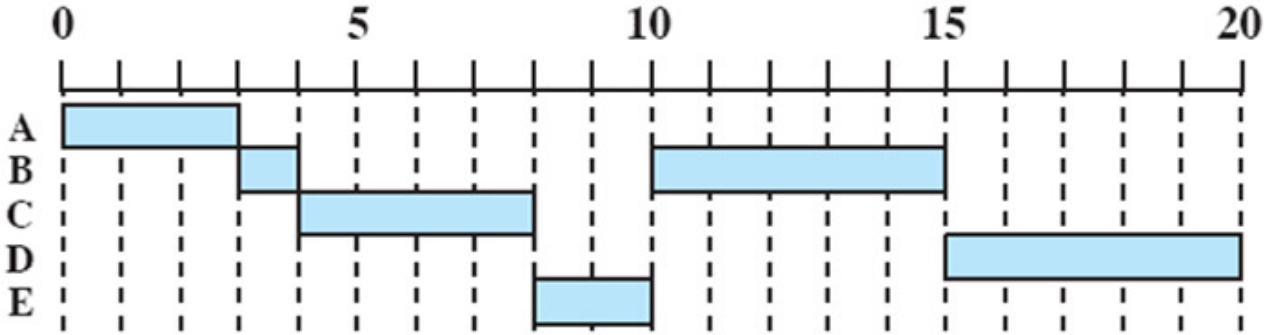
Round-Robin (RR), $q = 1$



Shortest Process Next (SPN)



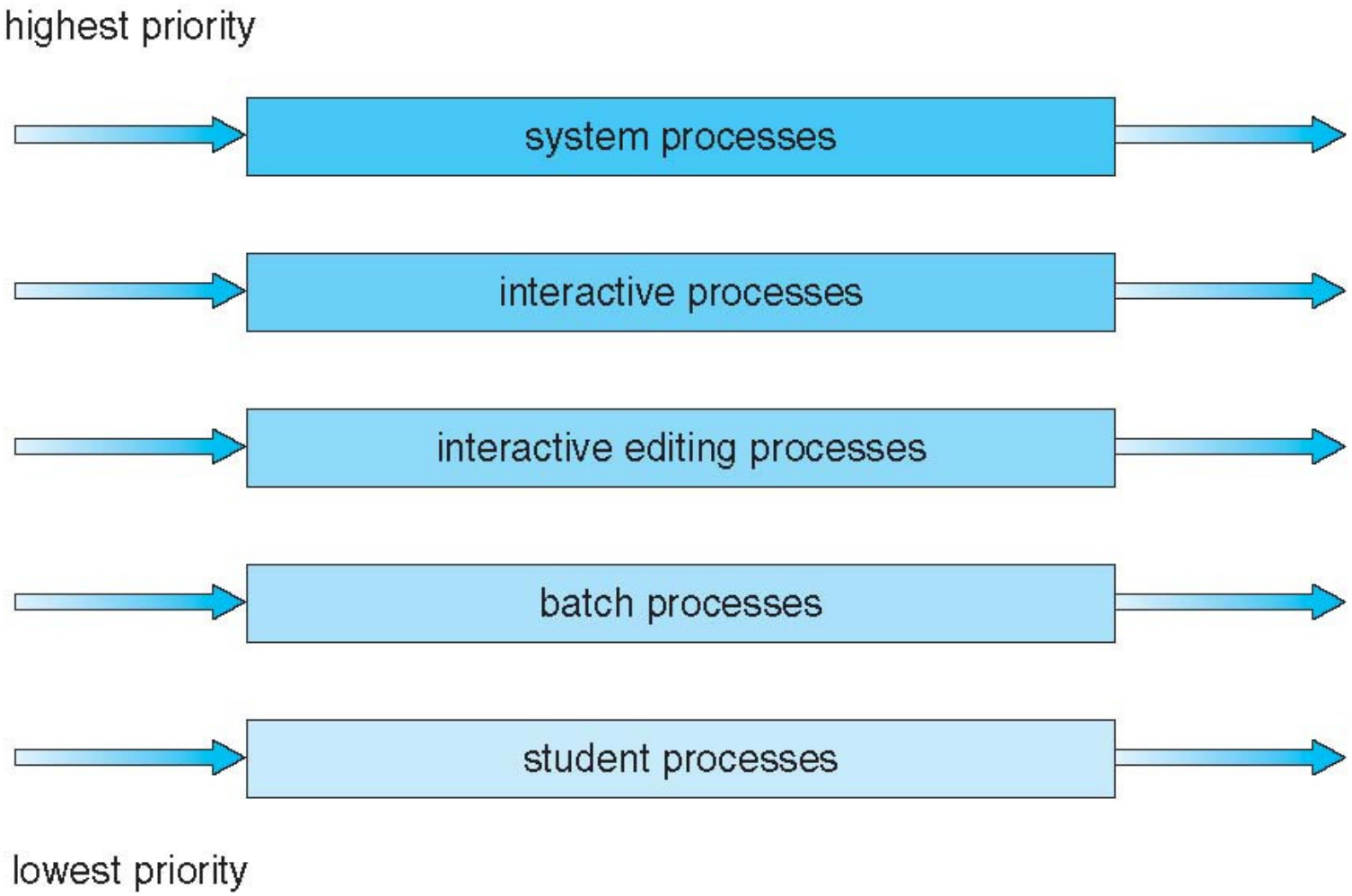
Shortest Remaining Time (SRT)



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



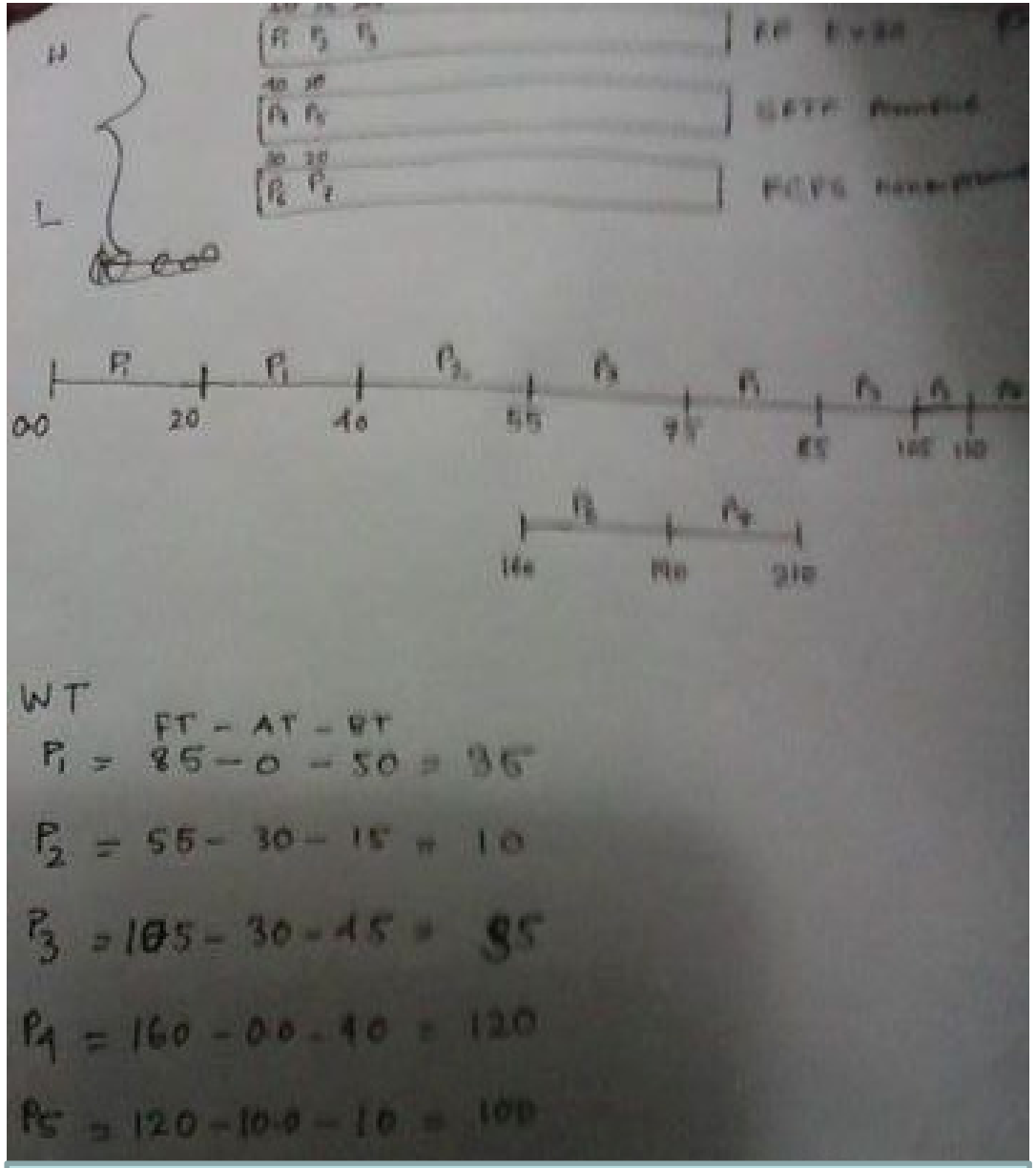
4. By "**Multilevel Queue Fixed priority**" scheduling algorithm, draw the CPU scheduling Gantt chart and complete the table for the following given processes information. (10)

Higher Priority

Lower Priority

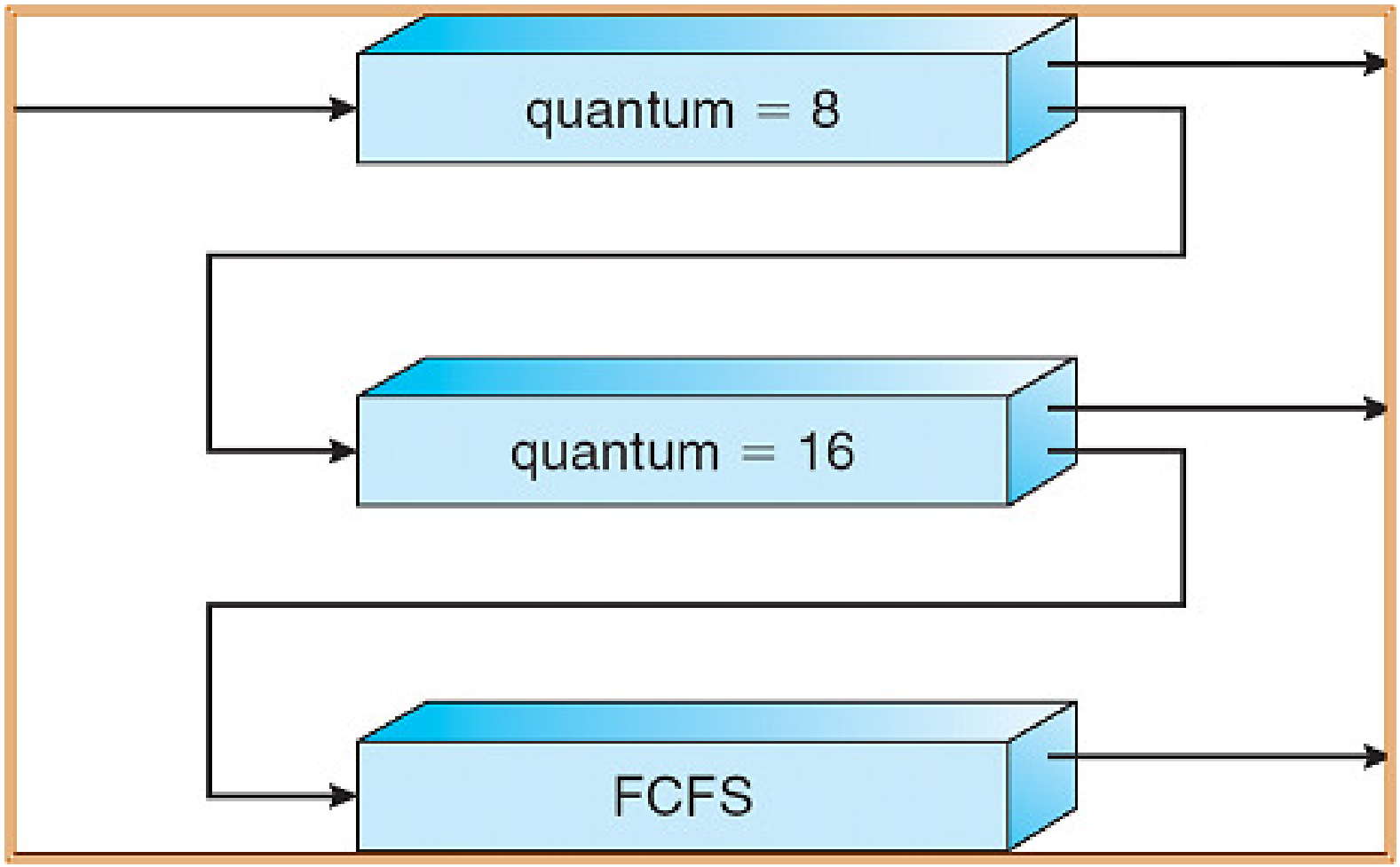
Queue Name	Process	Burst Time	Arrival Time	Algorithm
1st Foreground	P1 P2 P3	50 15 45	0.0 30.0 30.0	RR interval: 20 Preemptive
2nd Foreground	P4 P5	40 10	0.0 10.0	SRTF Preemptive
Background	P6 P7	30 20	60 130	FCFS Non-preemptive

	P1	P2	P3	P4	P5	P6	P7
Waiting time	**35**	**10**	**35**	**120**	**100**	**100**	**60**
Turnaround time	**85**	**25**	**80**	**160**	**110**	**130**	**80**



P6=190-60-30=100
P7=210-130-20=60

Multilevel Feedback Queue



Course Materials

@Galvin-6.1-6.3