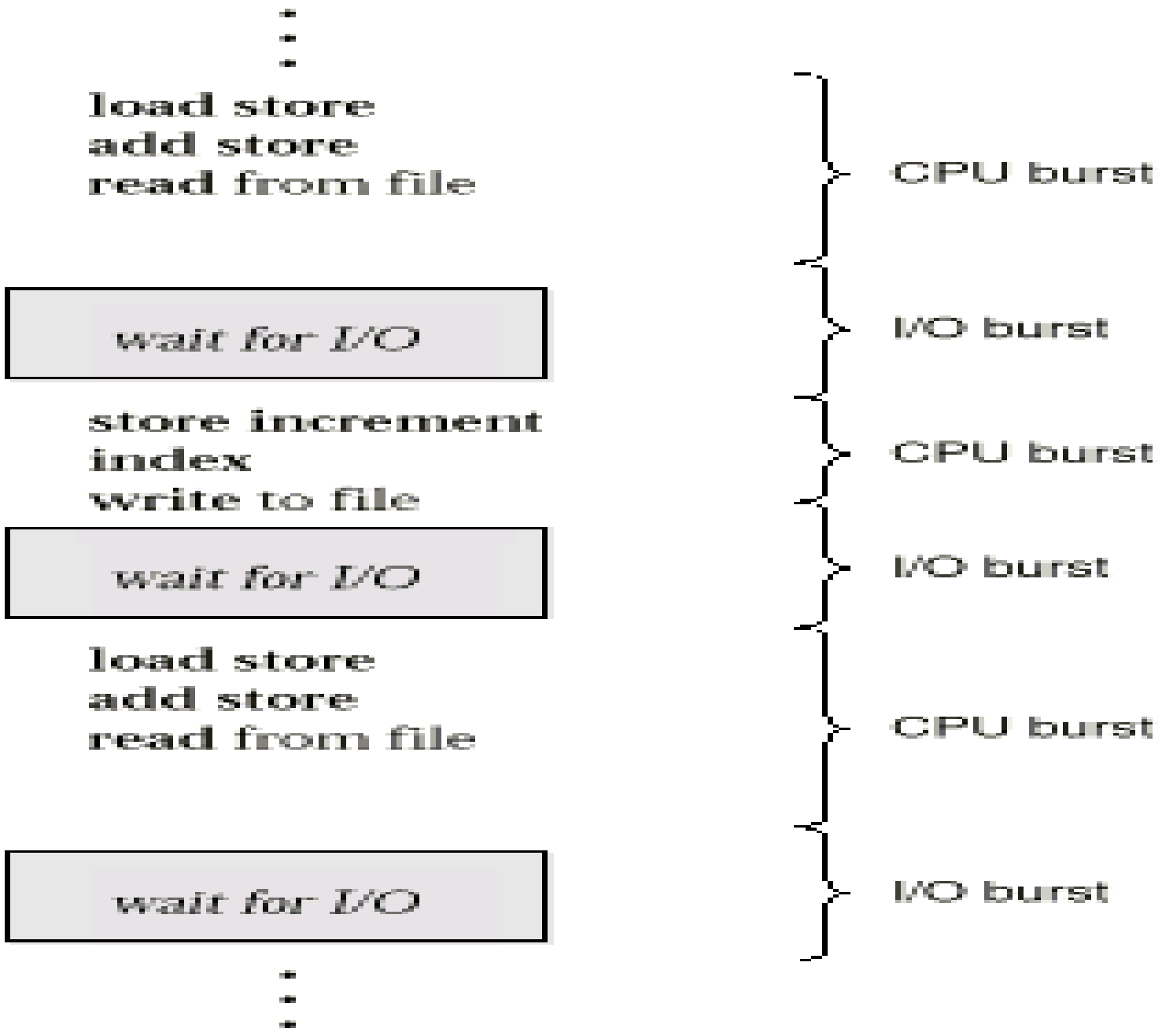
CS-3103 : Operating Systems : Sec-A (NB) :

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 distribution

Alternating Sequence of CPU And I/O Bursts

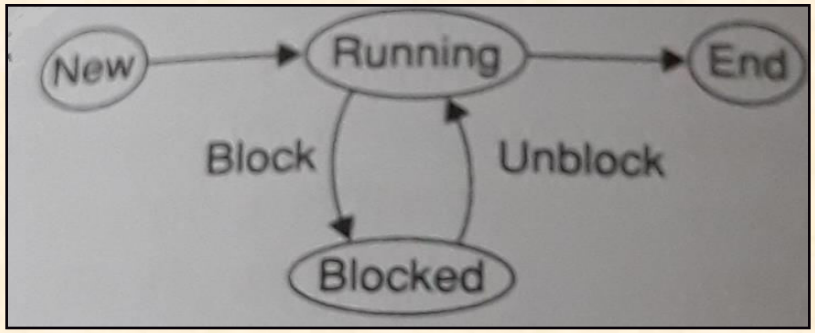


1. For each of the following transitions, between process states, which transition is not possible? **Ans: (d)**

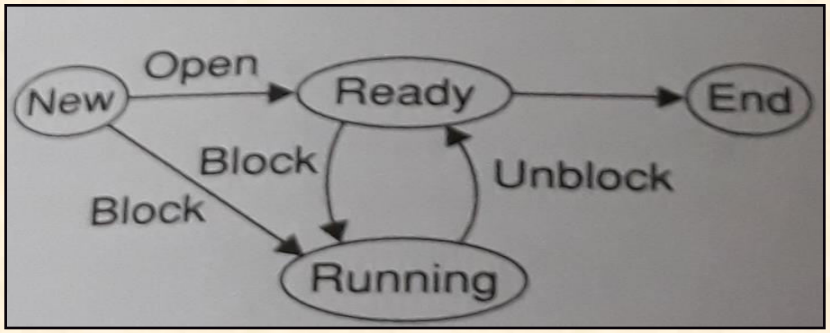
a) Running → Ready
b) Blocked → Suspend
c) Ready → Ready/Suspend
d) Blocked → Running

2. Which of the following is an appropriate four-state model for a process? **Ans: (a)**

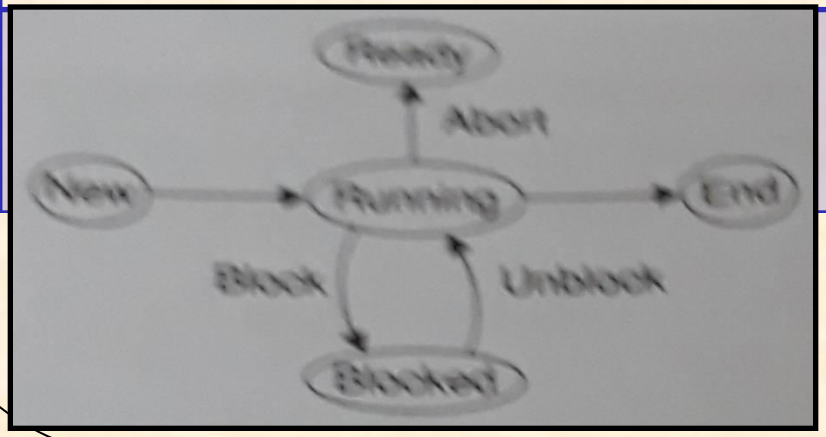
a)



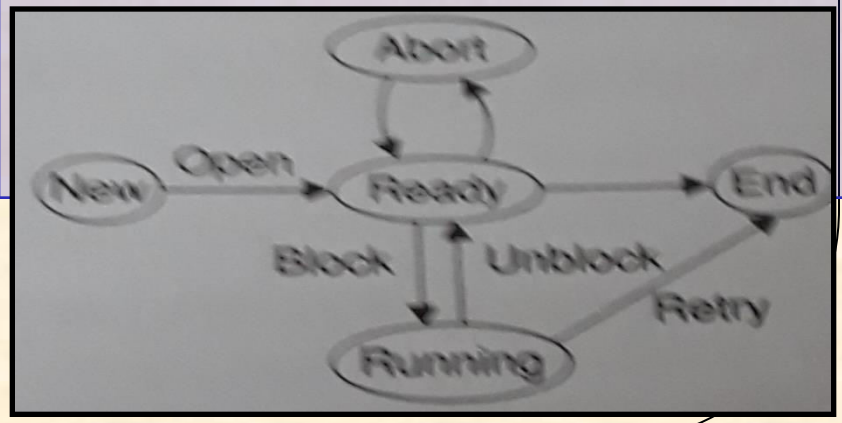
b)



c)

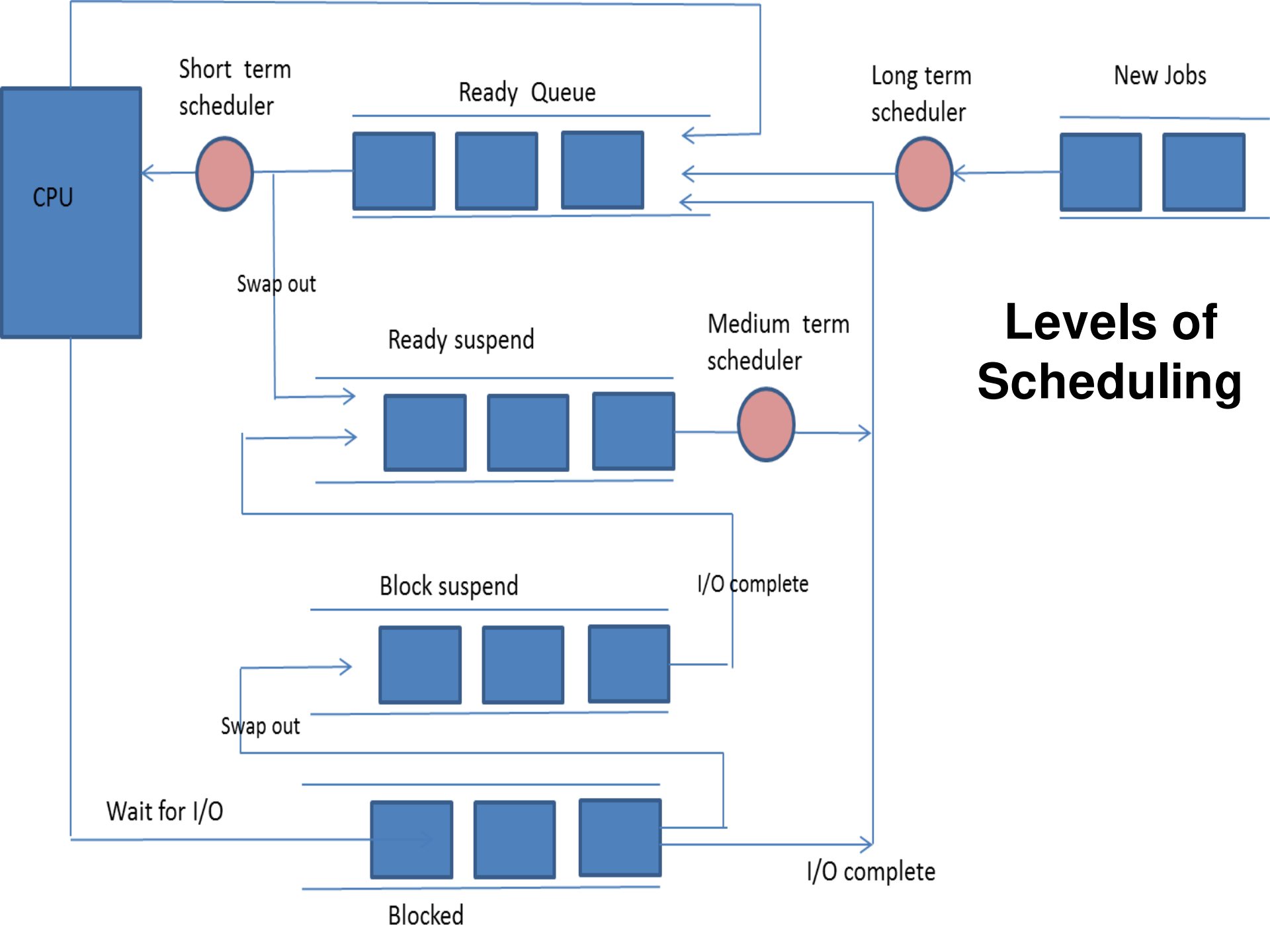


d)



CPU Scheduler

- Selects from among the processes in memory that are ready to execute, 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.
  3. Switches from waiting to ready.
  4. Terminates.
- Scheduling under 1 and 4 is *non preemptive*.
- All other scheduling is *preemptive*.



Levels of Scheduling

Dispatcher [*picture in previous slide*]

- **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.**

Performance Metrics

- During the execution of a process, both CPU and process need each other to do any useful work.
- The unit of work could be as small as one instruction or as large as one complete process.
- This measure is the *throughput* and is defined as the number of work units completed per unit time.
- Throughput is a metric for measuring system performance. Example: the number of application processes completed per second is a measure of the throughput of the system.

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. The time between the submission and completion of a job.**
- ***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)**

Optimization Criteria

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

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 to execute.

Device queues – set of processes waiting for an I/O device.

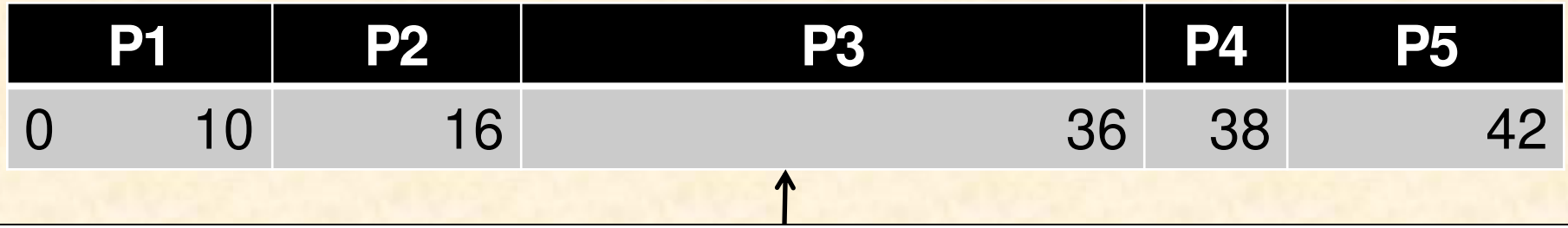
Process migration is performed between the various queues.

FCFS Scheduling

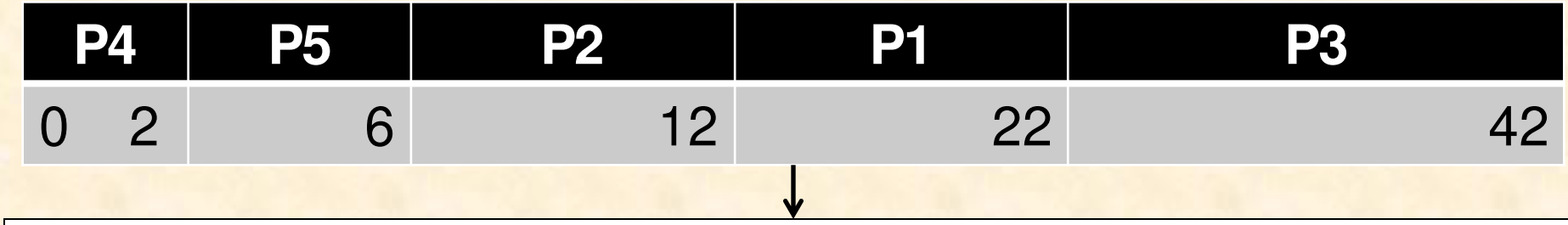
- **Jobs are executed on first come, first serve basis**.
- Easy to understand and implement.
- Poor in performance as average wait time is high.

- **Exercise**: Let P1, P2, P3, P4 and P5 be five processes that have become ready at the same time. They have joined the ready queue in the said order. Suppose they need 10, 6, 20, 2 and 4 units of time, respectively, to complete their executions. We assume that they are purely CPU-bound processes and they do not execute any waiting instructions. In addition, we ignore the context-switching time. The OS executes processes P1, P2, P3, P4 and P5, one after another in the said order.

FCFS Scheduling (Cont.)



Gantt chart of CPU allocation to processes with their turnaround/response times are 10,16, 36, 38 and 42 time units respectively. Thus, ***average response time = 28.4 time units***.



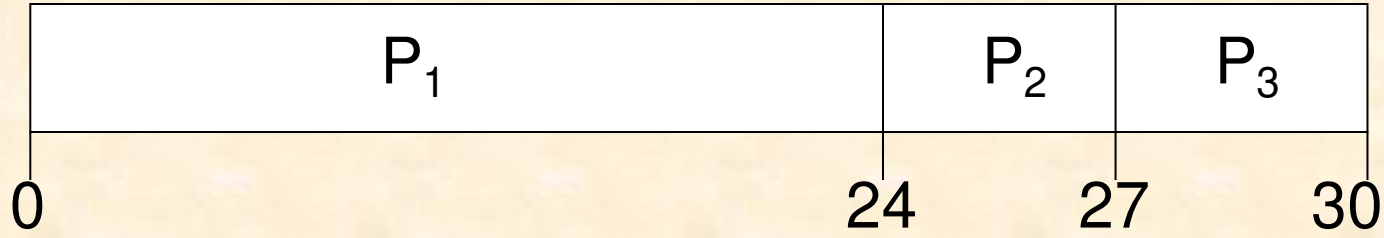
With a different order, P4<P5<P2<P1<P3, turnaround/response times would be 2,6,12,22 and 42 time units respectively. Thus, ***average response time = 16.80 time units***.

First-Come, First-Served (FCFS) Scheduling

- Example:

Process	CPU 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:



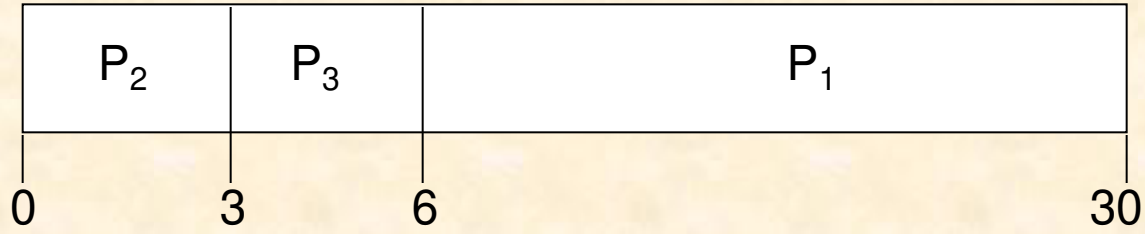
- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1 .$$

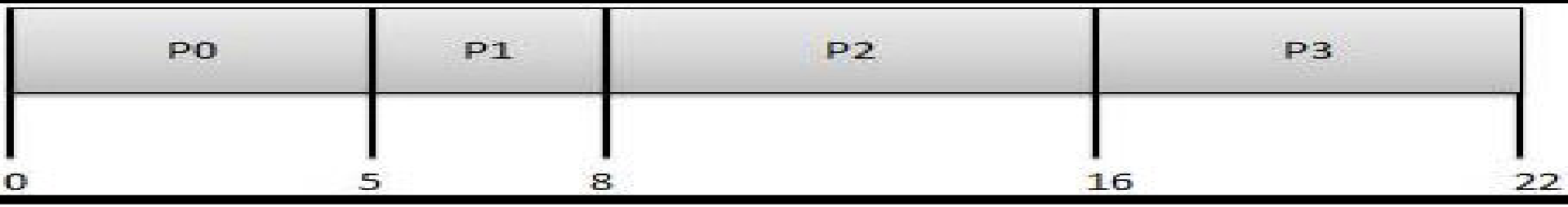
- The Gantt chart for the schedule is:



- 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.

FCFS Scheduling (Cont.)

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is following

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

SERVICE TIME: The time required by a device to handle a request.

Average Wait Time: (0+4+6+13) / 4 = 5.55

FCFS Scheduling (Cont.)

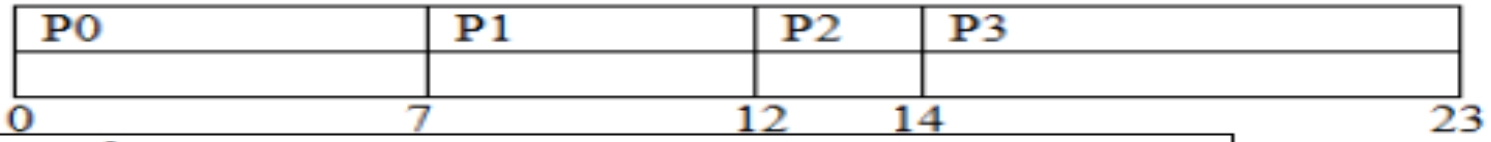
Find the average waiting time and average turnaround time for executing the following processes using FCFS (first-come first-service) scheduling?

Process	Burst time
P0	7
P1	5
P2	2
P3	9

Answer

The first step of the solution is founding the Gantt chart.

Gantt chart:



The waiting time of P0 = 0
The waiting time of P1 = 7
The waiting time of P2 = 12
The waiting time of P3 = 14
The average waiting time = (0 + 7 + 12 + 14) /4 = 33/4 = 8.25

The turnaround time of p0 = 7
The turnaround time of p1 = 12
The turnaround time of p2 = 14
The turnaround time of p3 = 23
The average turnaround time = (7+12+14+23)/4 = 56/4 = 14

Shortest-Job-First (SJF) Scheduling

- This algorithm associates with each process the length of the process's next CPU burst.
- *When the CPU is available, it is assigned to the process that has the smallest next CPU burst*.
- *If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie*.
- Most appropriate term → shortest-next-CPU-burst algorithm, because scheduling depends on the length of the next CPU burst of a process, rather than its total length.
- ***Best approach to minimize waiting time*.**
- ***Processer should know in advance how much time process will take*.**

Shortest-Job-First (SJF) Scheduling (Non-preemptive nature)…

Consider the following processes available in the ready queue for execution, with **arrival time** as 0 for all and the **burst times are given**.

In Shortest Job First Scheduling, the shortest Process is executed first.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

Hence the GANTT chart will be following :

Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

As you can see in the **GANTT chart** above, the process **P4** will be picked up first as it has the shortest burst time, then **P2**, followed by **P3** and at last **P1**.

Shortest-Job-First (SJF) Scheduling…

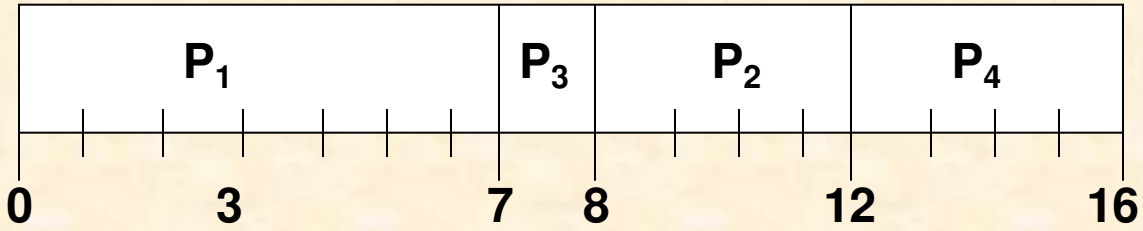
- ***Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time***.
- Two schemes:
  - ***Non-preemptive*** – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - ***Preemptive*** – if a new process 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.*

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

Non-Preemptive SJF

- SJF (non-preemptive)



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

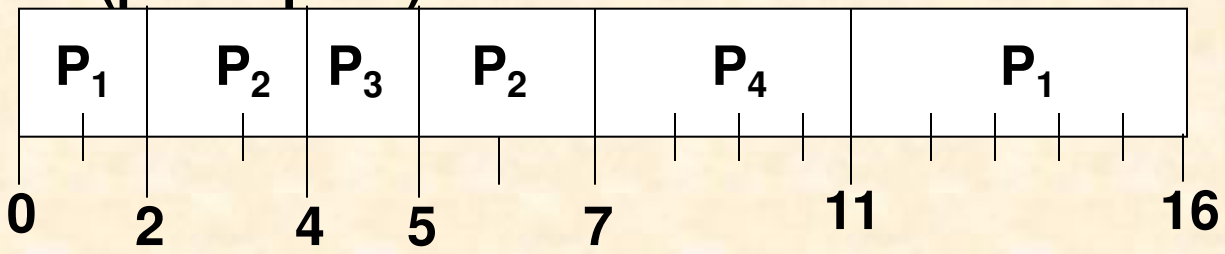
Wait Time → P1: 0, P2: 8-2=6, P3: 7-4=3, P4: 12-5=7

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

Preemptive SJF

- SJF (preemptive)



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

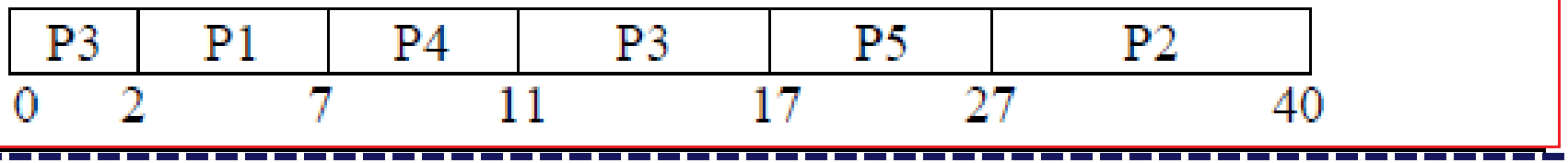
Wait Time → P1: 0+(11-2)=9; P2: (2-2)+(5-4)=1; P3: 4-4=0; P4: (7-5)=2;

Q: Find the average waiting time (A.W.T) and average turnaround time (A.T.A.T) for executing the following process using (1) Preemptive short-job first (2) Non-preemptive short-job first?

Process	P1	P2	P3	P4	P5
Burst time	5	13	8	4	10
Arrival time	2	3	0	5	1

Answer (1) Using preemptive short-job first

Gantt chart

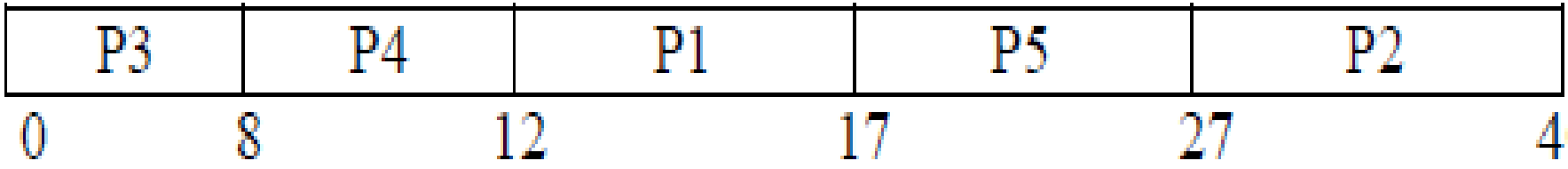


$$A.W.T. = (0+24+9+2+16)/5 = 51/5 = 10.2$$

$$A.T.A.T = (5+37+17+6+26)/5 = 91/5 = 18.2$$

Using non- preemptive short-job first

Gantt chart



W.T. of p1 = 12 -2 = 10

W.T. of p2 = 27 – 3 = 24

W.T. of p3 = 0

W.T. of p4 = 8 – 5 = 3

W.T. of p5 = 17 – 1 = 16

T.A.T. of P1= 17 – 2 = 15

T.A.T. of P2= 40 – 3 = 37

T.A.T. of P3= 8 – 0 = 8

T.A.T. of P4= 12 – 5 = 7

T.A.T. of P5= 27 -1 = 26

A.W.T. = (10+24+0+3+16)/5 = 53/5 = 10.6

A.T.A.T = (15+37+8+7+26)/5 = 93/5 = 18.6

Priority Scheduling

- *A priority number (integer) is associated with each process*
- *The CPU is allocated to the process with the highest priority.*
- *Equal-priority processes are scheduled in FCFS order.*
- *An SJF algo. is simply a priority algorithm where the priority (p) is the inverse of the CPU burst value. The larger the CPU burst, the lower the priority, and vice versa.*

Priority Scheduling

- *The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority).*
  - *Preemptive*
  - *nonpreemptive*
- *SJF is a priority scheduling where priority is the predicted next CPU burst time.*
- *Problem ≡ Starvation – low priority processes may never execute.*
- *Solution ≡ Aging – as time progresses increase the priority of the process.*

Priority Scheduling (Cont.)

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

P2	P5	P1	P3	P4
0 1	6	16	18	19

Using priority scheduling (***Non-preemptive***), we would schedule these processes according to the above table and Gantt chart. **The average waiting time is 8.2 milliseconds.**

Wait Time: P1: 6; P2: 0; P3: 16; P4: 18; P5: 1

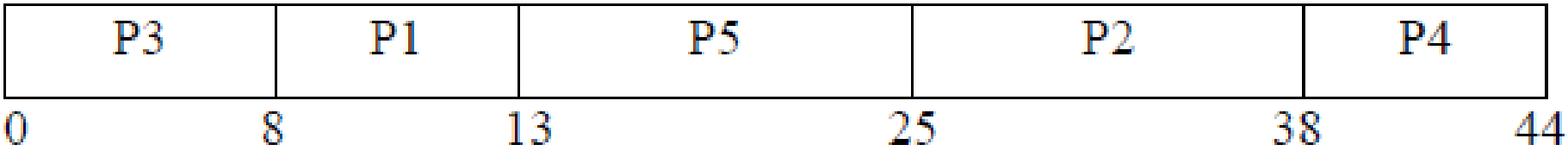
Indefinite blocking (or starvation)

- A major problem with priority-scheduling algorithms is ***indefinite blocking (or starvation).***
- ***A process that is ready to run but lacking the CPU can be considered blocked – waiting for the CPU***.
- ***A priority-scheduling algorithm can leave some low-priority processes waiting indefinitely for the CPU***.
- ***In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU***.
- Bad example: Rumor : When IBM 7094 at MIT was shut down in 1973, they found a low-priority process that had been submitted in 1967 and **had not yet been run**.

Q : Find the average waiting time and turnaround time for executing the following process using priority scheduling algorithm?

Process	P1	P2	P3	P4	P5
Burst time	5	13	8	6	12
Priority	1	3	0	4	2

Answer Gantt chart



W.T. of p1 = 8

W.T. of p2 = 25

W.T. of p3 = 0

W.T. of p4 = 38

W.T. of p5 = 13

T.A.T. of P1=13

T.A.T. of P2= 38

T.A.T. of P3= 8

T.A.T. of P4= 44

T.A.T. of P5= 25

$$A.W.T. = (8+25+0+38+13)/5 = 84/5 = 16.8$$

$$A.T.A.T = (13+38+8+44+25)/5 = 128/5 = 25.6$$

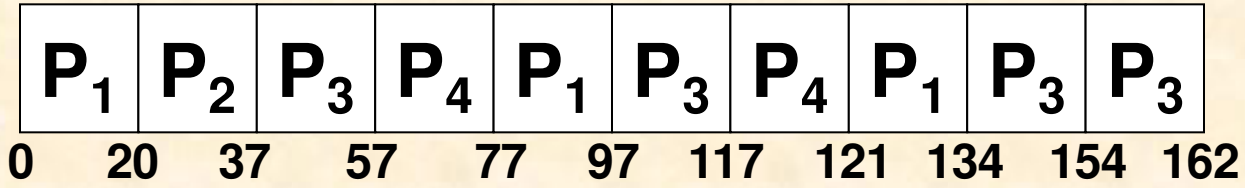
Round Robin (RR)

- **Each process gets a small unit of CPU time (*time quantum*), 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.
- Performance
  - *q* large $\Rightarrow$ FIFO
  - *q* small $\Rightarrow$ *q* must be large with respect to context switch, otherwise overhead is too high.

Example: RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



Typically, higher average turnaround than SJF, but better *response*.

Q : Find the average waiting time (A.W.T.) and the average turnaround time (A.T.A.T.) for executing the following processes using round-robin algorithm, where time quantum is 5?

Process	P1	P2	P3	P4	P5
Burst time	11	4	14	9	21
Arrival time	5	0	0	1	2

Answer

P2= ~~4~~ 0
P3= ~~14~~ ~~9~~ ~~4~~ 0
P4= ~~9~~ ~~4~~ 0
P5= ~~21~~ ~~16~~ ~~11~~ ~~6~~ ~~1~~ 0
P1= ~~11~~ ~~6~~ ~~1~~ 0

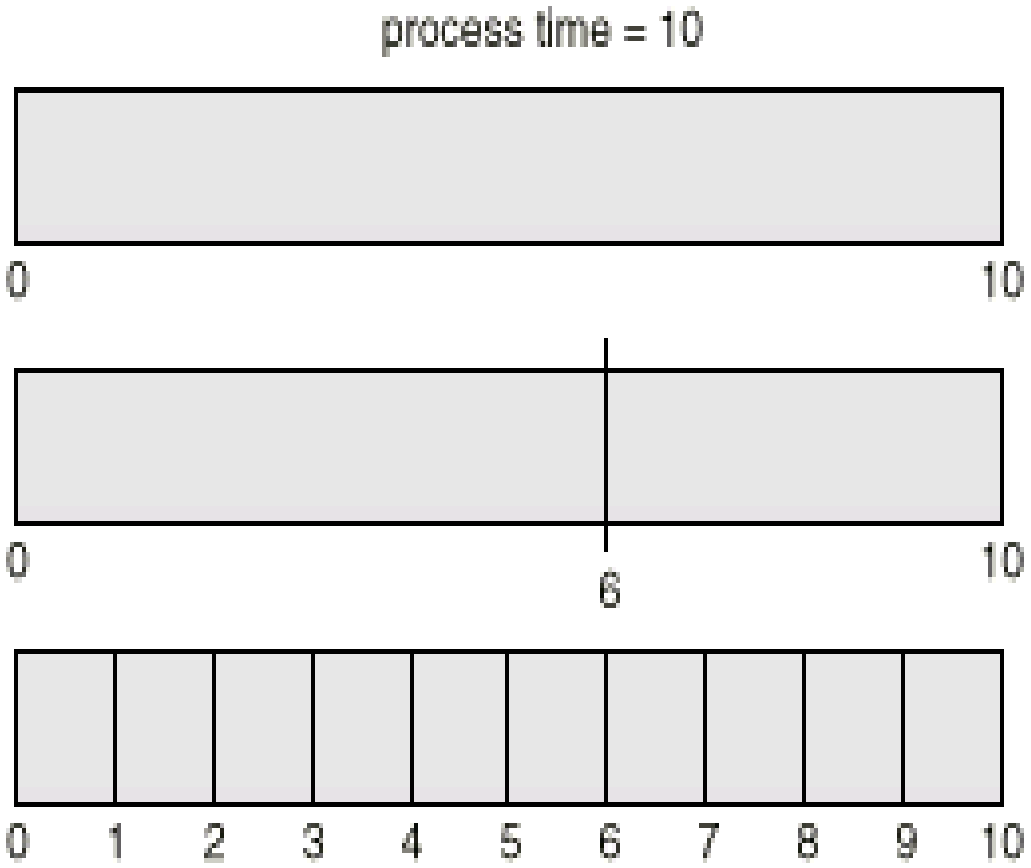
Gantt chart

W.T. of P1 =(19-5)+(38-24)+(52-43) = 14 + 14+9 = **37**
W.T. of P2 = **0**
W.T. of P3 = (4-0) + (24-9) + (43-29) = 4 + 15 + 14 = **33**
W.T. of P4 = (9-1) + (29-14) = 8 + 15 = **23**
W.T. of P5 = (14-2) + (33-19) + (47-38) + (53-52) = 12+14+9+1=**36**
A.W.T. = (37 + 0+ 33 + 23 + 36) /5 = 129 /5 = 25.8

T.A.T of P1 = (53-5)= **48**


START

How a Smaller Time Quantum Increases Context Switches

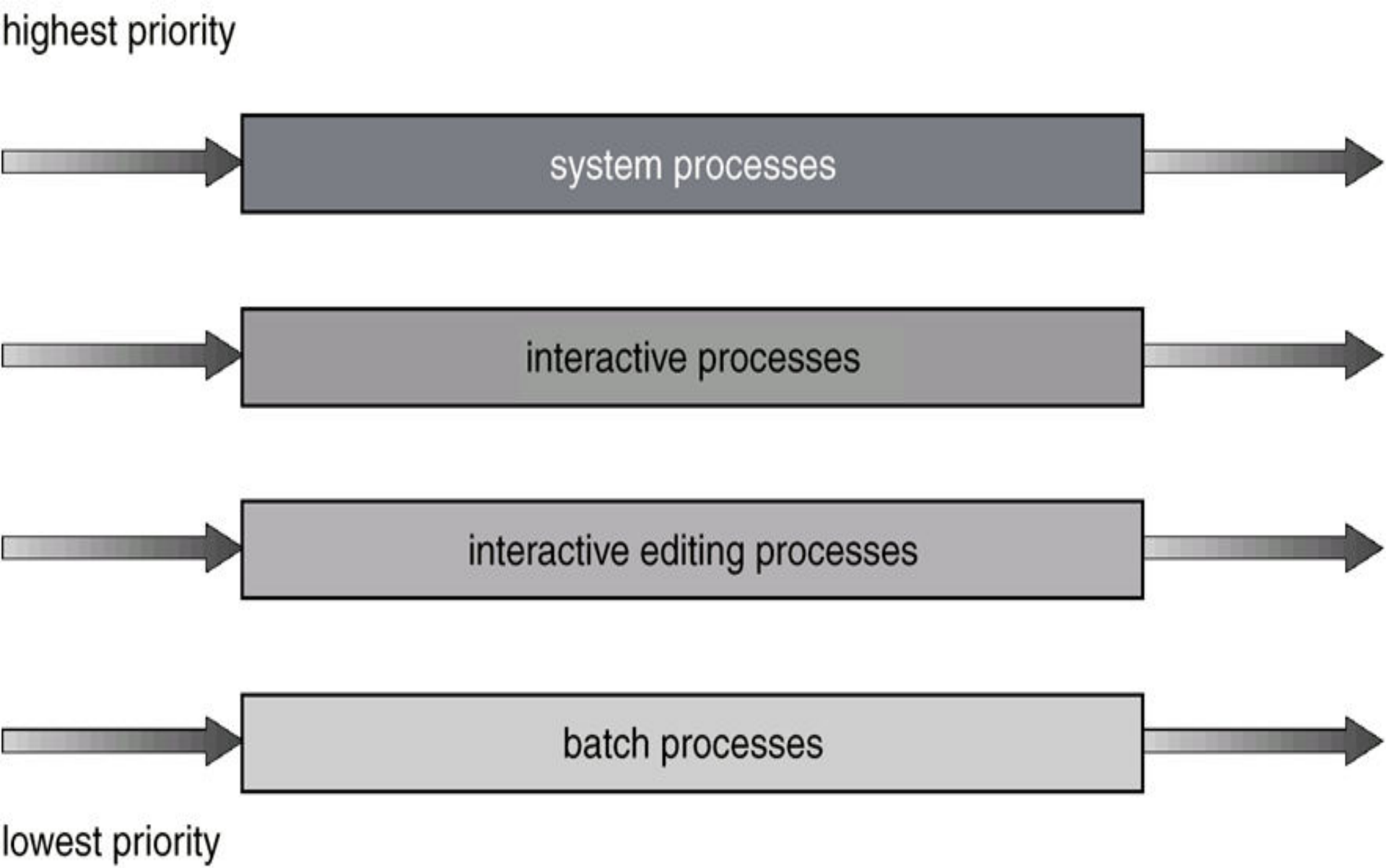


quantum	context switches
12	0
6	1
1	9

Multilevel Queue

- Ready queue is partitioned into separate queues:
  foreground (interactive)
  background (batch)
- 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

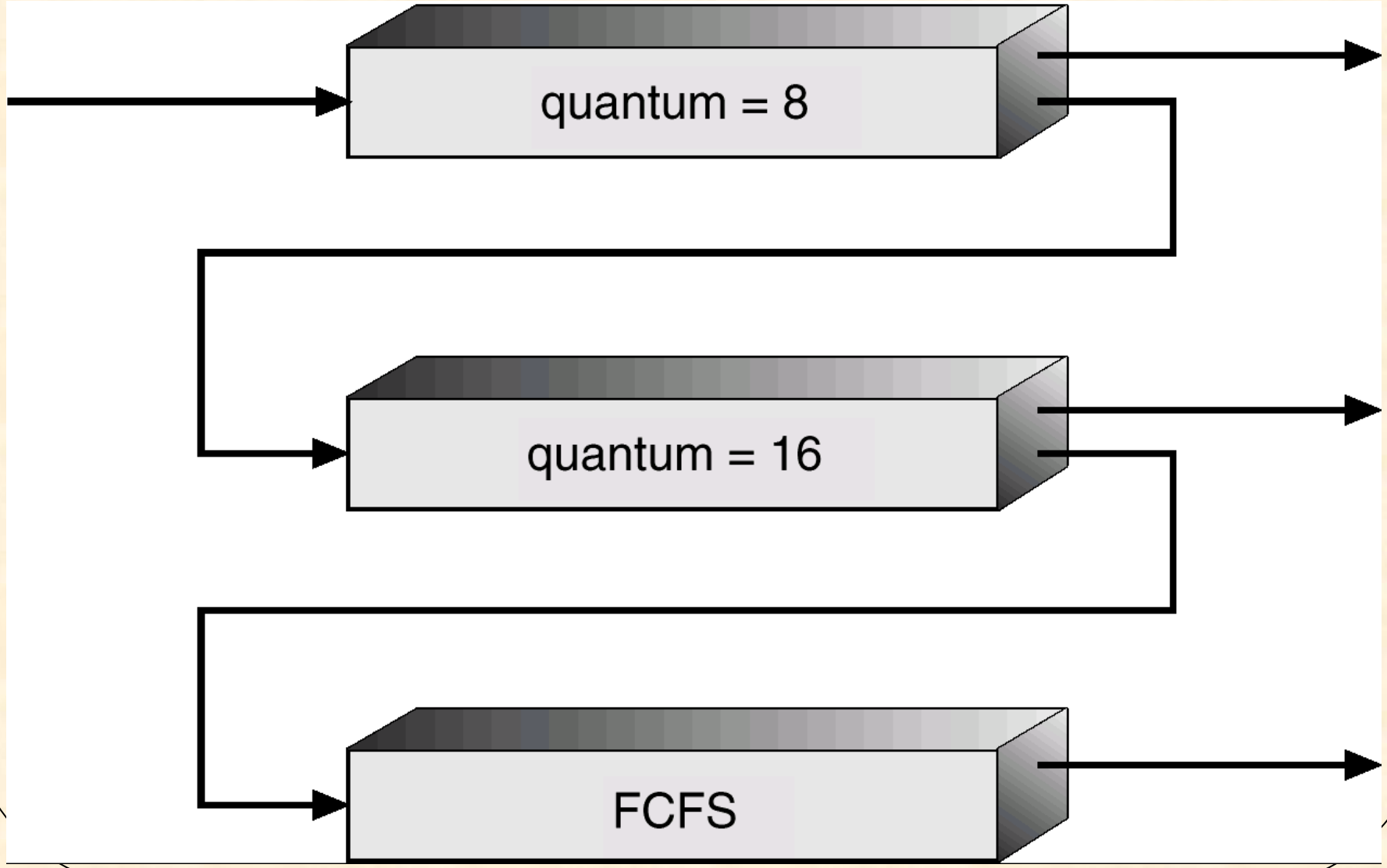
Multilevel Queue Scheduling



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

Multilevel Feedback Queues



Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – time quantum 8 milliseconds
  - Q_1 – 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.

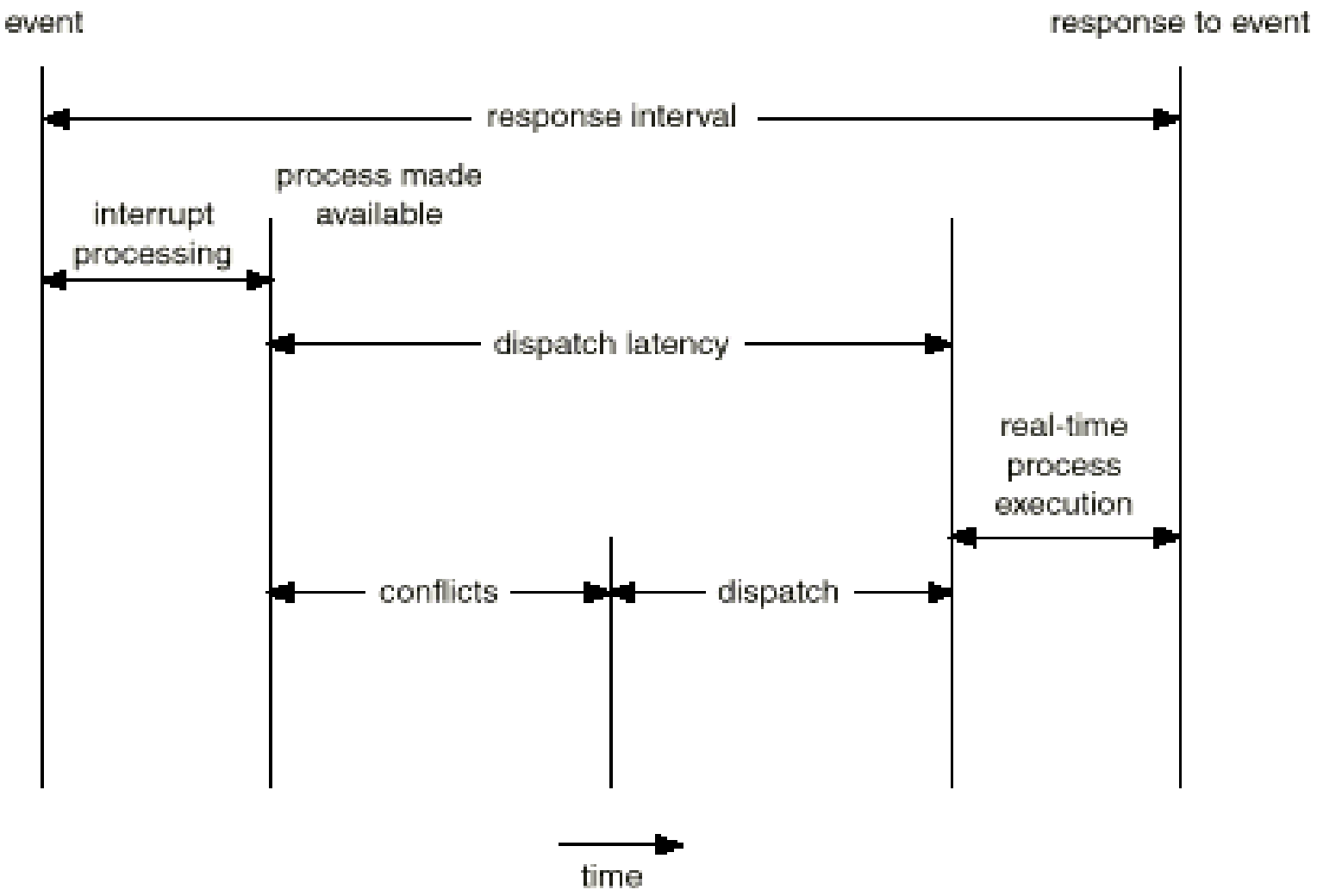
Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- *Homogeneous processors* within a multiprocessor.
- *Load sharing*
- *Asymmetric multiprocessing* – only one processor accesses the system data structures, alleviating the need for data sharing.

Real-Time Scheduling

- *Hard real-time* systems – required to complete a critical task within a guaranteed amount of time.
- *Soft real-time* computing – requires that critical processes receive priority over less fortunate ones.

Dispatch Latency



Algorithm Evaluation

- Deterministic modeling – takes a particular predetermined workload and defines the performance of each algorithm for that workload.
- Queuing models
- Implementation

Evaluation of CPU Schedulers by Simulation