

CYB 510 OPERATING SYSTEMS

Prepared by Prof. Dr. Lamiaa Fattouh

Lecture 4

جامعة الجميع الذكية

تسعى لتكون الكيان الأول الذي يتبنى مبدأ المجتمع التعليمي
التعاوني ويقدم العلم مقابل العمل لتنمية وبناء المجتمعات.

CYB 510 CPU SCHEDULING



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

- The Real difficulty with the SJF algorithm is knowing the length of the next CPU request
- We may not know but we can predict
- Can be done by using the length of previous CPU bursts, using exponential averaging

SHORTEST JOB FIRST:

- Optimal for minimizing queueing time, but impossible to implement. Tries to predict the process to schedule based on previous history.
- Predicting the time, the process will use on its next schedule:

$$t(n+1) = w * t(n) + (1 - w) * T(n)$$

Here: $t(n+1)$ is time of next burst.

$t(n)$ is time of current burst.

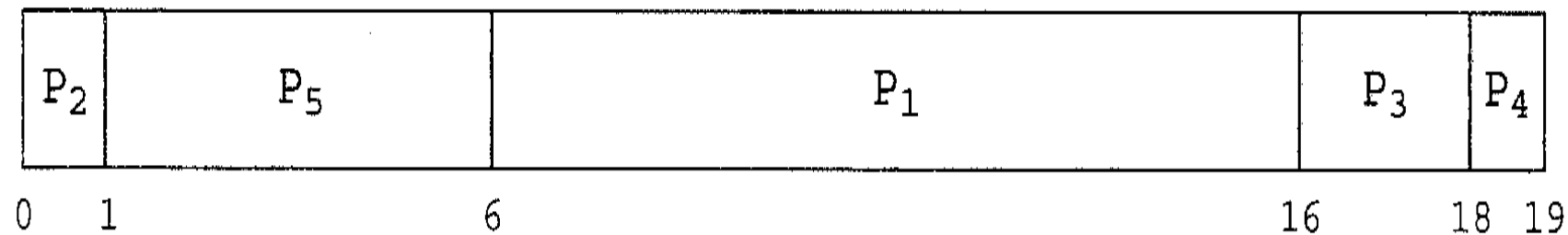
$T(n)$ is average of all previous bursts .

w is a weighting factor emphasizing current or previous bursts.

- SJF is priority scheduling where priority is the predicted next CPU burst T
- Priority can be defined internally (to OS) or externally.
 - Internal: CPU usage time, memory requirements, I/O usage
 - External: importance, type, political issues
- A priority number (integer) is associated with each process (small $\equiv$ high)
- The CPU is allocated to the process with the highest priority (smallest integer $\equiv$ highest priority).
- When a process arrives at ready state, its priority will be compared to priority of running process. **If newly process has higher priority:**
 - **Preemptive** – it preempts the running one and old running one will be put into ready.
 - **Nonpreemptive** – it does not preempt the running one and it will be put at the head of ready state.
- Problem $\equiv$ **Starvation (indefinite blocking)** – low priority processes may never execute
- Solution $\equiv$ **Aging** – As a process uses more execution time, dynamically decreases its priority or increase the priority of other lower processes (say increase priority by 1 every 15 min)

	<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>
	P_1	10	3
Highest	—————→ P_2	1	1
	P_3	2	4
lowest	—————→ P_4	1	5
	P_5	5	2

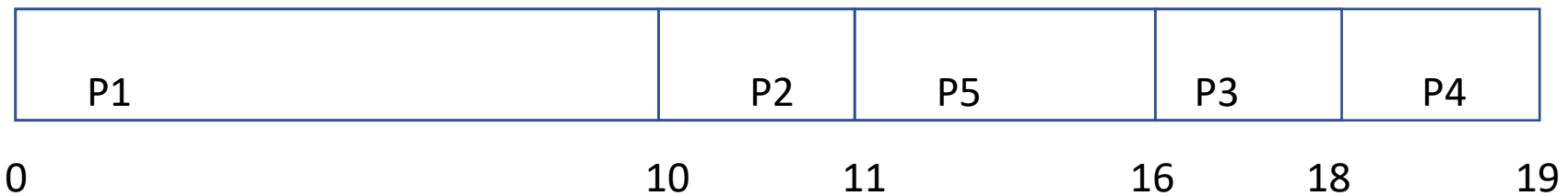
Using priority scheduling, we would schedule these processes according to the following Gantt chart:



$$\text{Average waiting time} = (0+1+6+16+18)/5 = 41/5 = 8.2\text{msec}$$

Non-Preemptive

<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>	<u>Arrival time</u>
P_1	10	3	0
P_2	1	1	3
P_3	2	4	2
P_4	1	5	1
P_5	5	2	4



Waiting time $P_1=0$

$$P_2 = 10 - 3 = 7$$

$$P_5 = 11 - 4 = 7$$

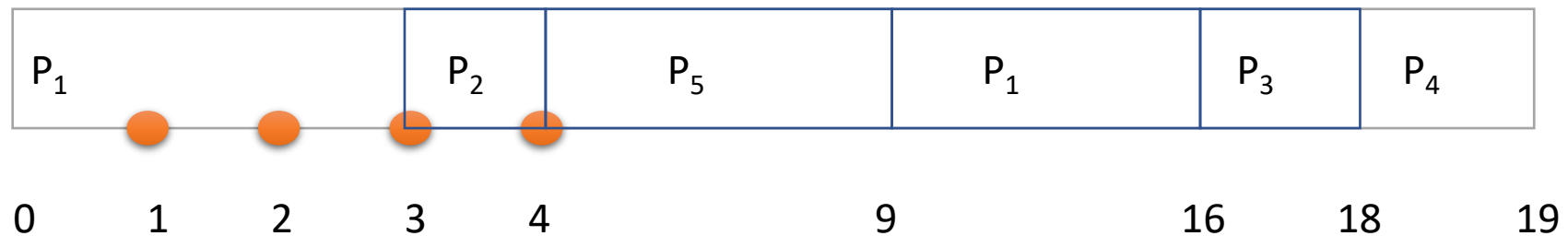
$$P_3 = 16 - 2 = 14$$

$$P_4 = 18 - 1 = 17$$



Preemptive

<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>	<u>Arrival time</u>
P_1	10	3	0
P_2	1	1	3
P_3	2	4	2
P_4	1	5	1
P_5	5	2	4



- In Timesharing systems that wish to provide “fair” distribution of CPU resources one process **can’t monopolize CPU**
- **RR** can be considered **FCFS with preemption**.
- Each process is assigned a single **time interval (time-slice)**, called **quantum**, in which it is allowed to run (use CPU) before it is **preempted**.



Each process gets a small unit of CPU time, usually 10-100 milliseconds.

If process **is still running** at the end of the quantum, CPU is given to another process from ready queue and that process is preempted $\Rightarrow$ **move to ready**

RR puts those processes in ready queue. Where?
will put them at the end

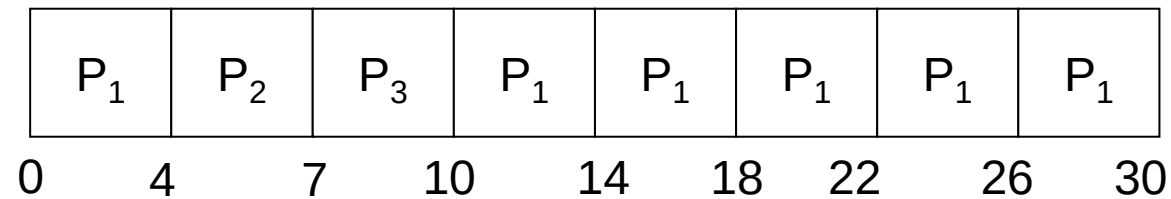
If there are **n processes** in the ready queue and the time **quantum is Q** , each process gets **$1/n$ of the CPU**, in chunks of at most size Q (**fairness**)

No process **waits** for more than **$(N-1)Q$ time units** before next quantum

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:



$$\text{Average waiting time} = (0+4+7+(10-4))/3 = 17/3 = 5.66\text{msec}$$

Chapter 5 Continue

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

Quantum=2

P ₁	P ₂	P ₃	P ₁	P ₂	P ₃	P ₁	P ₁	
0	2	4	6	8	9	10	12	30

Average = $[(6-2)+(10-8)]+(2+(8-4))+(4+(9-6)))/3=19/3= 6.33 \text{ msec}$. It gets better if the quantum is chosen smaller

Sure all above numbers ignore context switch time!



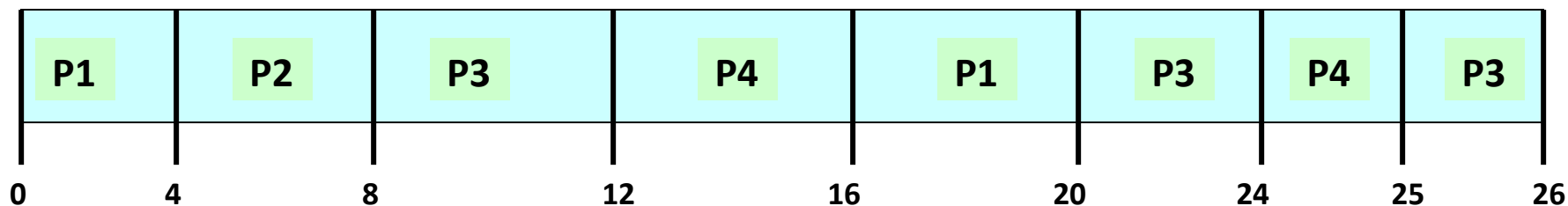
Chapter 5 Continue

Example of RR with Time Quantum= 4

EXAMPLE DATA:

Process	Arrival Time	Service Time
1	0	8
2	1	4
3	2	9
4	3	5

Round Robin, quantum = 4, no priority-based preemption



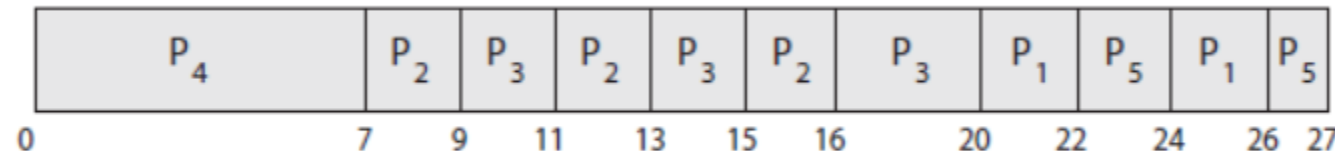
$$\begin{aligned} \text{Average wait} &= (12) + (4-1) + ((8-2)+(20-12)+(25-24)) + ((12-3)+(24-16)) / 4 = \\ &= 12+3+15+17= \\ &= 47/4= 11.75 \end{aligned}$$

Chapter 5 Continue

Round-robin and Priority scheduling

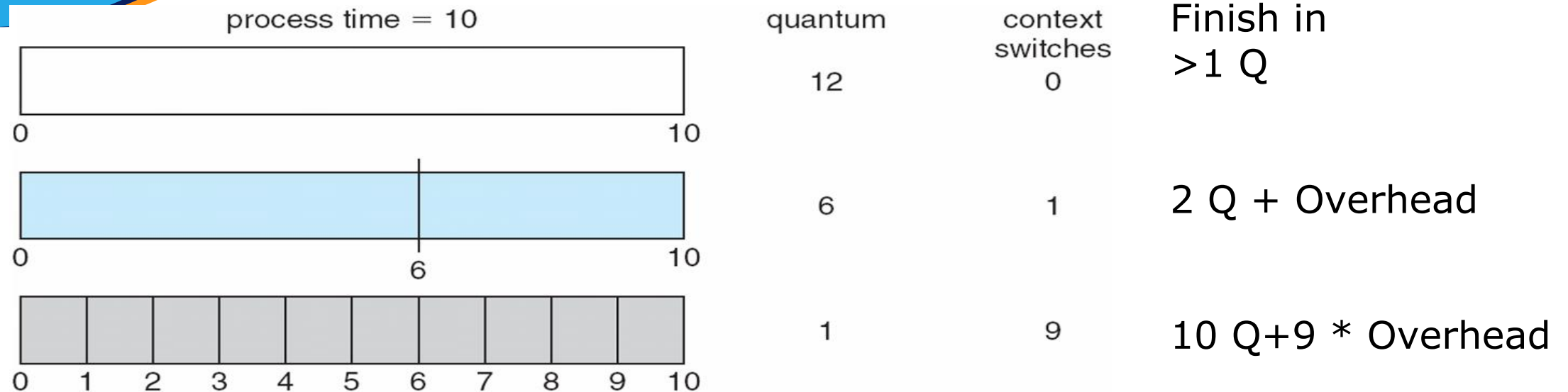
Another option is to combine round-robin and priority scheduling in such a way that the system executes the highest-priority process and runs processes with the same priority using round-robin scheduling. **Time Quantum= 2**

<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3



Chapter 5 Continue

Time Quantum and Context Switch Time

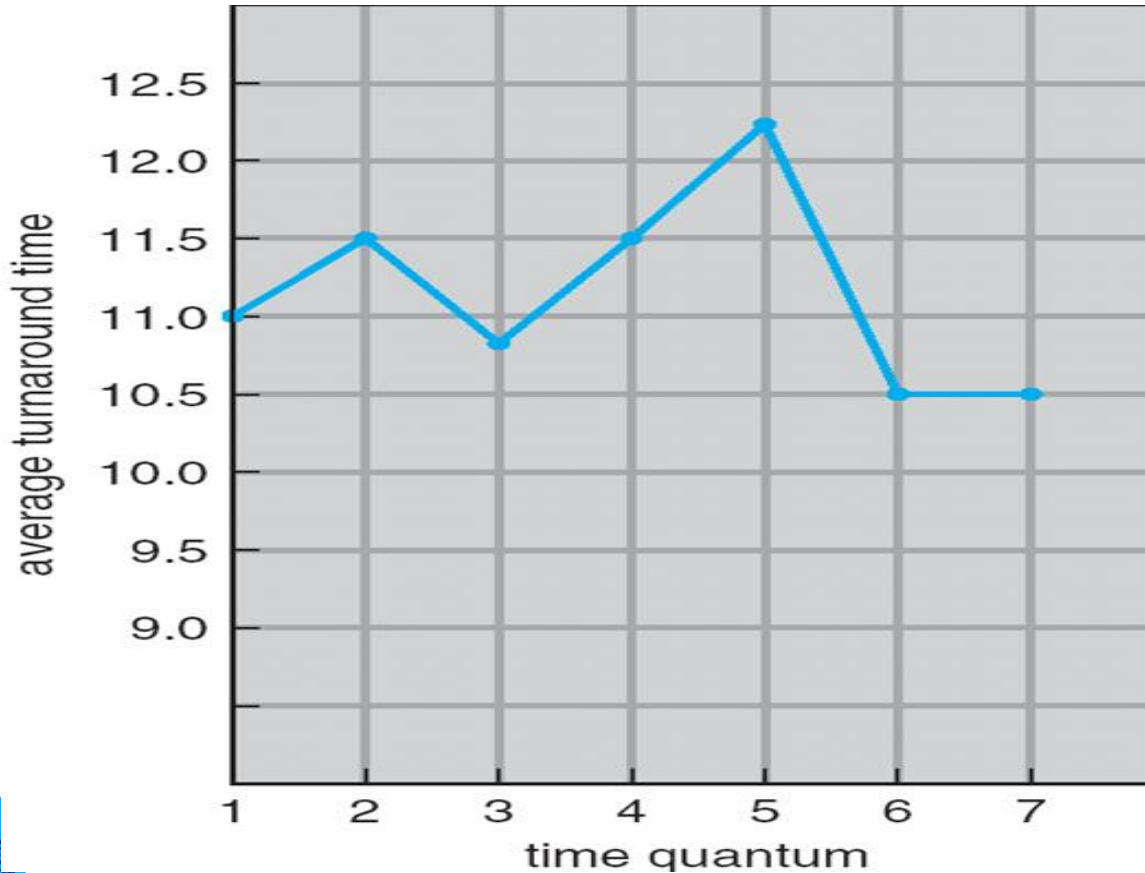


Thus, we want the time quantum to be large with respect to the context-switch time. If the context-switch time is approximately 10 percent of the time quantum, then about 10 percent of the CPU time will be spent in context switching. In practice, most modern systems have time quanta ranging from 10 to 100 milliseconds. The time required for a context switch is typically less than 10 microseconds; thus, the context-switch time is a small fraction of the time quantum.

- Setting **quantum too short** → many process switches → **lowers the CPU efficiency**. Suppose: quantum = 20 msec. process switch = 5 msec. This yields a 20% CPU waste time and 80% usage.
- Setting the **quantum too long** → may cause **poor response time** to short interactive requests (e.g. 500 msec. And 5 → CPU wasted time less than 1%)

Chapter 5 Continue

Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Turnaround time depends on size of quantum .

Average turnaround time does not necessarily improve as time quantum size increase.



- Suitable for the case where processes can be classified in different groups according to some process criteria (type, size, priority,...)
- Multilevel Queue Scheduling partitions the Ready queue into several separate queues according to response time requirement .
- Processes have different scheduling needs:
 - foreground (interactive) -high response time and have external priority
 - background (batch) – has less interaction and less priority

Each queue has its own scheduling algorithm

foreground – RR

background – FCFS

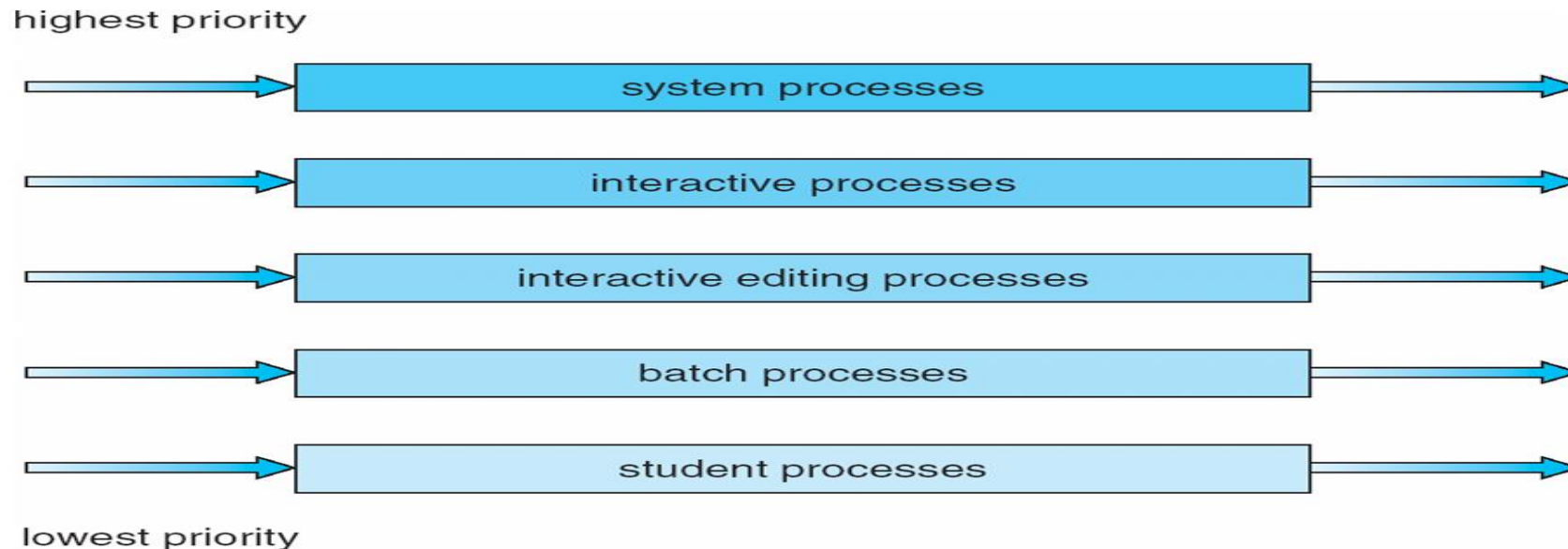
Scheduling must be done between the queues. Two possibilities

1) **Fixed priority scheduling**; (i.e., serve all from foreground then from background). Possibility of **starvation**.

If a student process is running and an interactive process enters the ready queue, the student process will be preempted.

Means that no process can run from batch if processes exist in the higher priority queues--→ Starvation is possible

2) **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 and 20% to background in FCFS



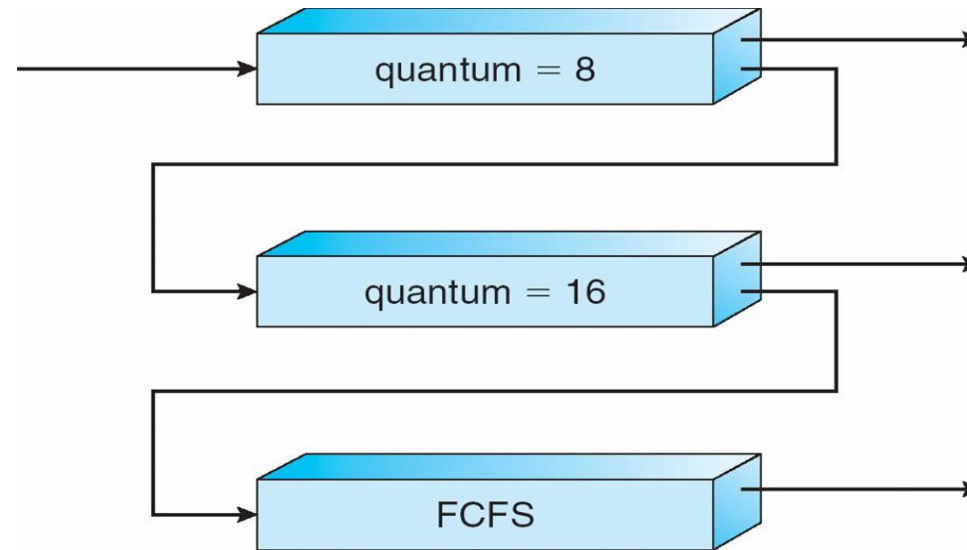
Problem: normally a process enters a queue will remain there. System does not move it from queue to queue!. **Permanent queue assignment**

- A **process** can move **between the various queues**:
 - Separate processes according to their **CPU bursts characteristics**
 - If a process uses too much CPU, move it to lower priority queue
 - Use aging to prevent starvation
 - If a process stay too long to get CPU, move it to higher priority queue
- 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



- **Three queues** with different Q and priority and scheduling algorithm:
 - Q_0 – RR with time quantum 8 milliseconds (Highest priority)
 - Q_1 – RR time quantum 16 milliseconds
 - Q_2 – FCFS (Lowest priority)
- Scheduling
 - A new job enters queue Q_0 which is served FCFS basis . 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 basis and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2 .

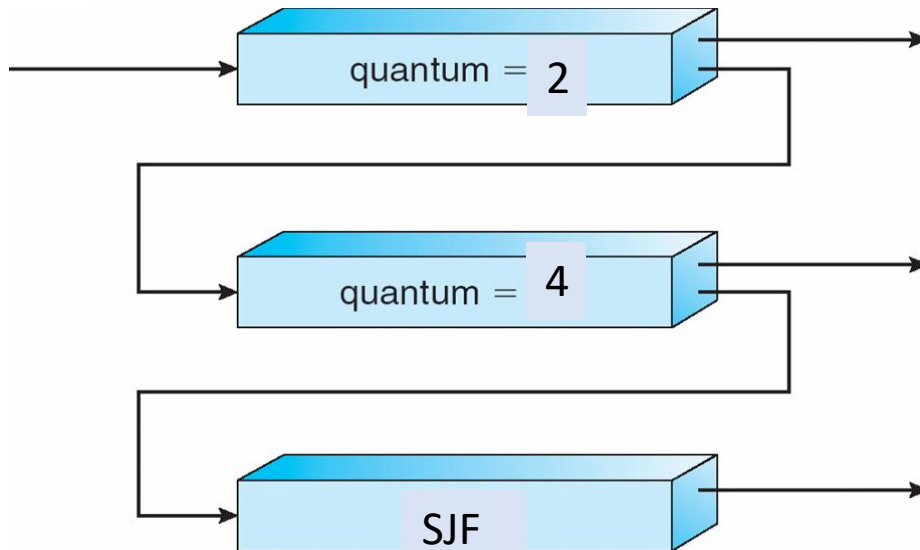
Chapter 5 Continue



Short Process has higher priority:

- Always run processes in Q_0
- Only when Q_0 is empty, processes in Q_1 or Q_2 (if Q_1 is empty) get CPU
- Preempt running process from Q_1 or Q_2 when a process enters Q_0

Process	Arrival time	Burst time
P1	3	8
P2	4	7
P3	8	7
P4	12	1



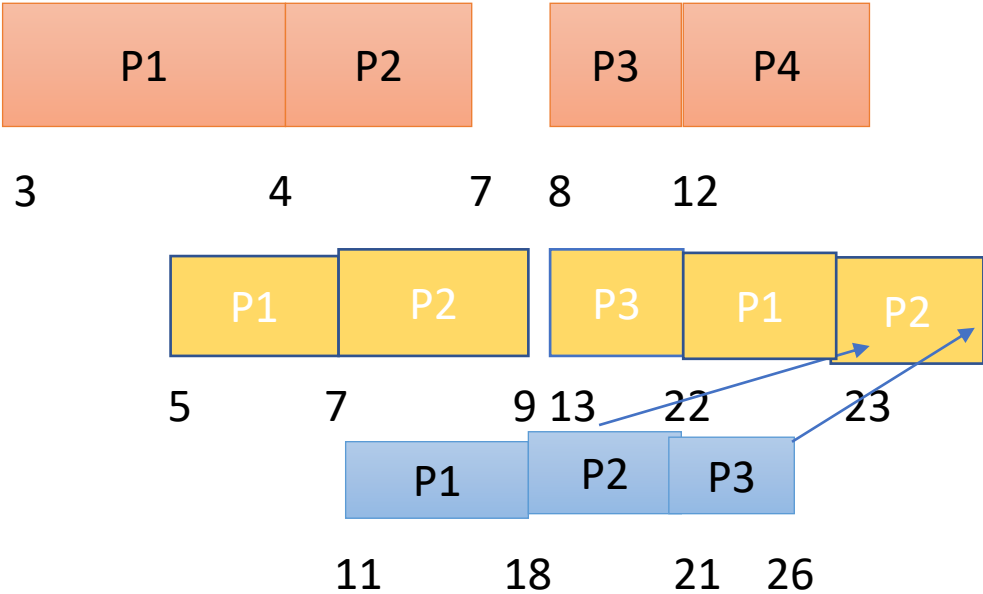
- **Q1**: RR, $q=2$ ms..... (higher priority)
- **Q2**: RR, $q=4$ ms
- **Q3**: Non preemptive-SJF.. (lower priority)
- Scheduling between queues is **Non-preemptive priority**

▪ In **Q1 & Q2** : If the process doesn't finish in its time slice, it will be moved to the next lower-priority queue.

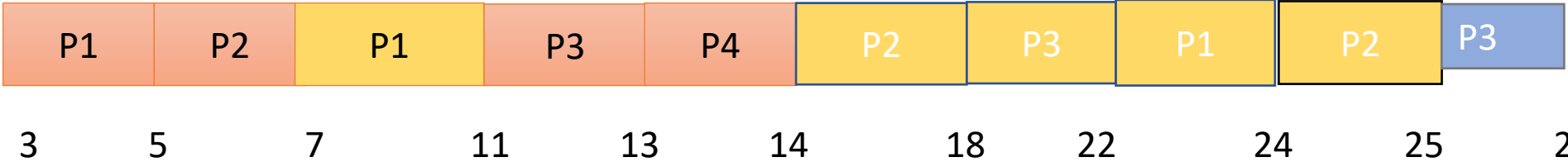
▪ If a process **waiting time ≥ 13 ms**, it must be **upgraded to the next higher queue**

Chapter 5 Continue

	TIME= 13	22	23	
P1	2	13		
P2	1	12	13	
P3	3	8	9	
P4	1	1		



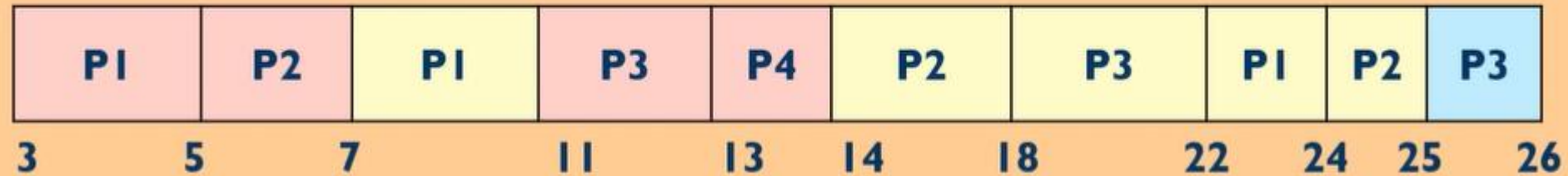
Process	Arrival time	Burst time
P1	3	8
P2	4	7
P3	8	7
P4	12	1



Chapter 5 Continue

Process	Arrival time	Burst time
P1	3	8
P2	4	7
P3	8	7
P4	12	1

(a) CPU Gantt chart:



$$P1 = 7 - 5 + 22 - 11 = 13$$

$$P2 = 5 - 4 + 14 - 7 + 24 - 18 = 14$$

$$P3 = 11 - 8 + 18 - 13 + 25 - 22 = 11$$

$$P4 = 13 - 12 = 1$$



(b) The process will be **demoted** according the following conditions:

Q1 → Q2 : If Process burst time >2

Q2 → Q3 : If Process burst time >6

(c) P1 has been upgraded at
time(22) from Q3 to Q2

**P2 has been upgraded at time 24
from Q3 to Q2**

Chapter 5 Continue

Test yourself

Process	Burst	Priority
P ₁	8	4
P ₂	6	1
P ₃	1	2
P ₄	9	2
P ₅	3	3

- Priority
- SJF
- FCFS

Chapter 5 Continue

Answer : priority

Non-Preemptive Priority

Process	Burst	Priority
P ₁	8	4
P ₂	6	1
P ₃	1	2
P ₄	9	2
P ₅	3	3

0	6	7	16	19	27
P ₂	P ₃	P ₄	P ₅	P ₁	

$$\text{Avg. Wait Time} = 0+6+7+16+19 = 48/5 = 9.6\text{ms} \quad \text{Avg TAT} = 6+7+16+19+27 = 75/5 = 15\text{ms}$$

Chapter 5 Continue

Answer : SJF

Process	Burst	Priority
P ₁	8	4
P ₂	6	1
P ₃	1	2
P ₄	9	2
P ₅	3	3

0	1	4	10	18	27
P ₃	P ₅	P ₂	P ₁	P ₄	

$$\text{Avg. Wait} = 0+1+4+10+18 = 33/5 = 6.6\text{ms} \quad \text{Avg. TAT} = 1+4+10+18+27 = 60/5 = 12\text{ms}$$



Chapter 5 Continue

Answer : FCFS

Process	Burst	Priority
P ₁	8	4
P ₂	6	1
P ₃	1	2
P ₄	9	2
P ₅	3	3

First Come First Served

0	8	14	15	24	27
P ₁	P ₂	P ₃	P ₄	P ₅	

$$\text{Avg. Wait} = 0+8+14+15+24 = 61/5 = 12.2\text{ms} \quad \text{Avg. TAT} = 8+14+15+24+27 = 17.6\text{ms}$$



End lecture 4
Thank You

