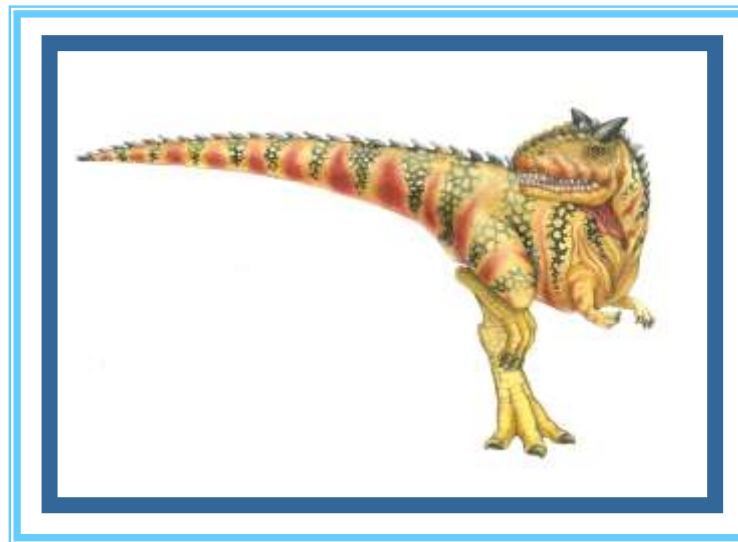


Chapter 5: CPU Scheduling





Chapter 5: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Operating Systems Examples





Objectives

- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system





CPU Scheduling

- Operating System manages a collection of processes.
- If a system has more processes and one CPU or More processes with fewer CPU's has to divide CPU time among different processes.
- This is called CPU scheduling.





Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
 - OS can make the computer more productive by switching the CPU among processes.
 - Main objective is some process running at all times

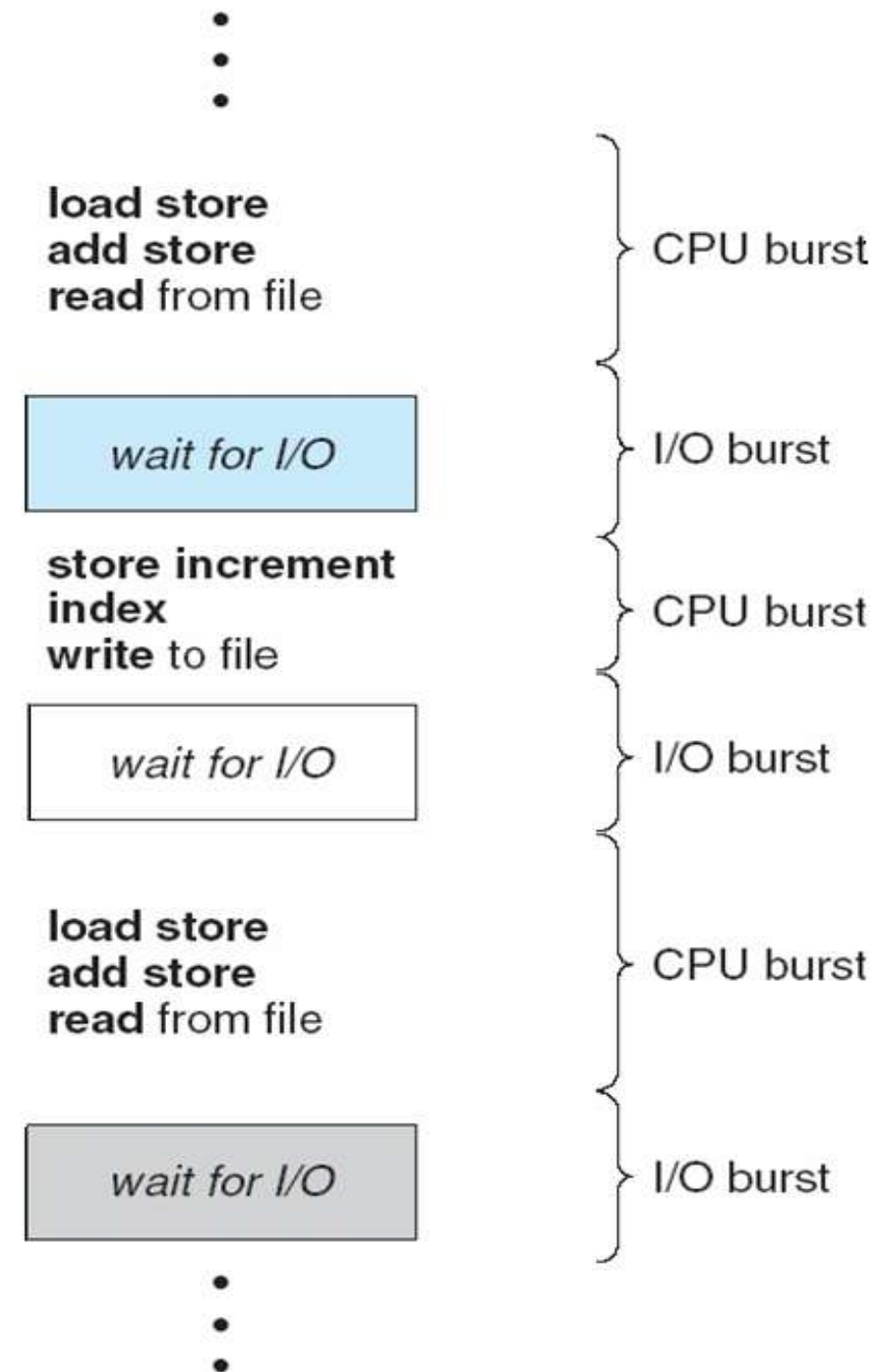
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait

- **CPU burst** distribution





Alternating Sequence of CPU and I/O Bursts





CPU 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**
 - Preemption means that a process may be forcibly removed from the CPU even if it does not want to release the CPU.





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

Different CPU-scheduling algorithms have different properties, and the choice of a particular algorithm may favor one class of processes over another. In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms.

Many criteria have been suggested for comparing CPU-scheduling algorithms.





Scheduling Criteria

The criteria include the following:

- **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.





Scheduling Algorithm Optimization Criteria

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

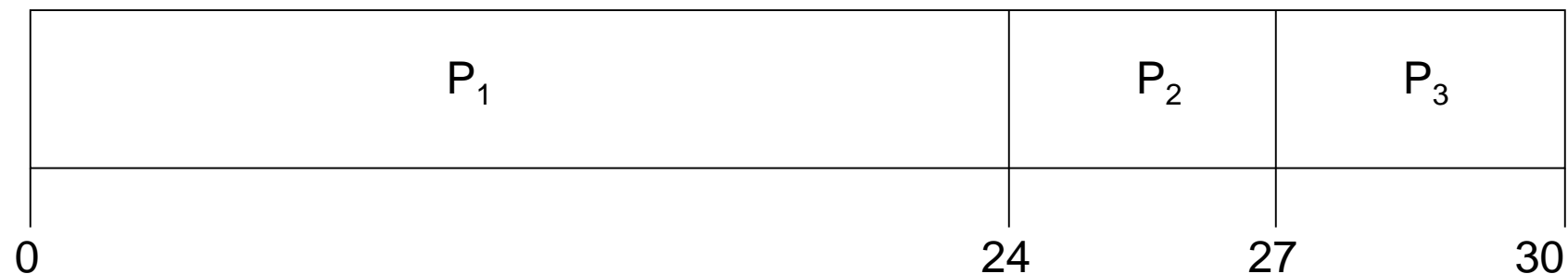




First-Come, First-Served (FCFS) Scheduling

<u>Process</u>	<u>Burst Time</u>
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: $(0 + 24 + 27)/3 = 17$



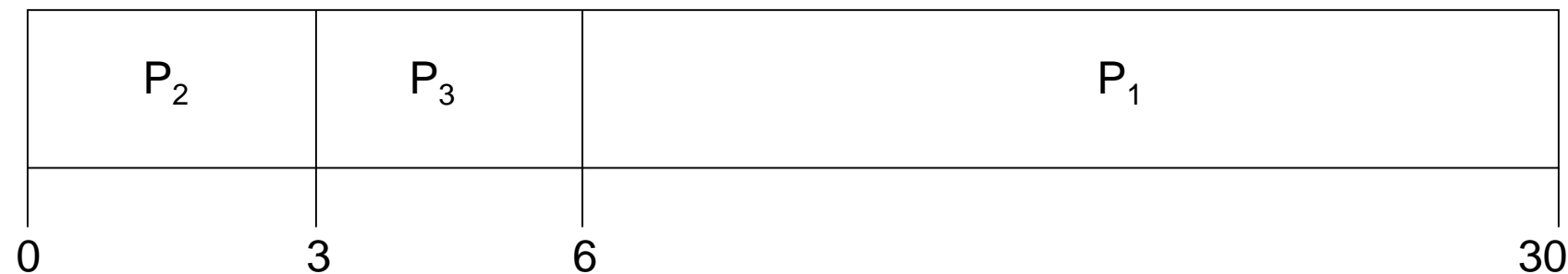


FCFS Scheduling (Cont.)

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
- **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

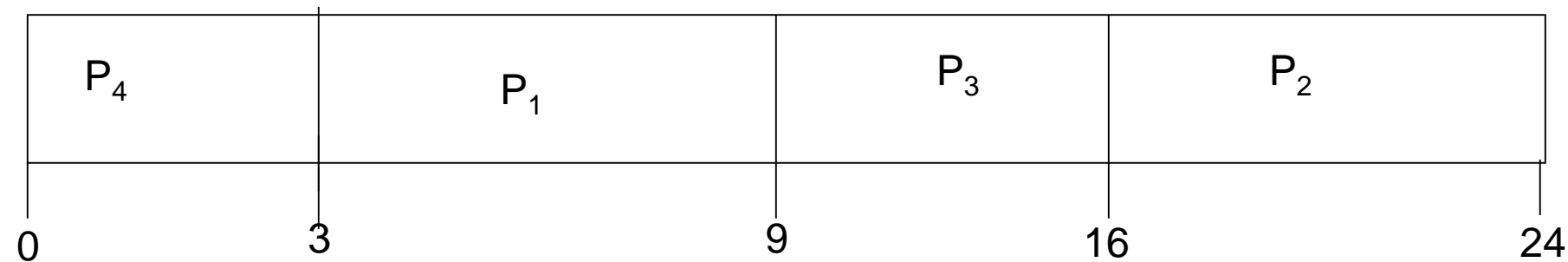




Example of SJF

<u>Process</u>	<u>Burst Time</u>
P_1	6
P_2	8
P_3	7
P_4	3

■ SJF scheduling chart



■ Average waiting time = $(3 + 16 + 9 + 0) / 4 = 7$



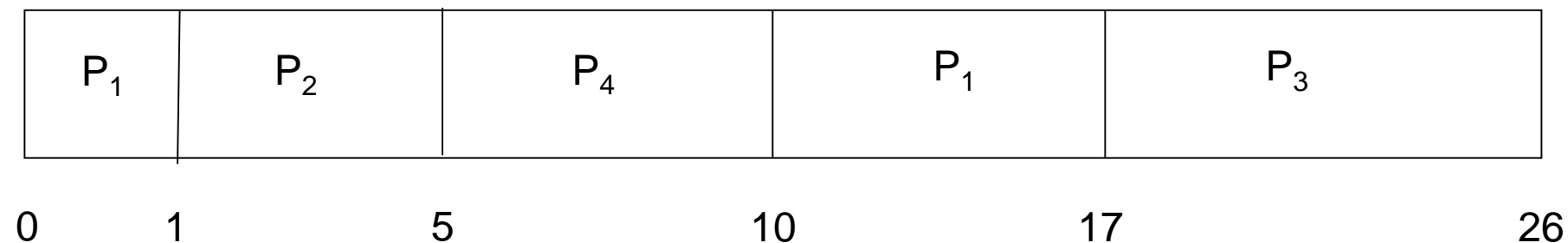


Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
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 \equiv highest priority)
 - Preemptive
 - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem \equiv **Starvation** – low priority processes may never execute
- Solution \equiv **Aging** – as time progresses increase the priority of the process

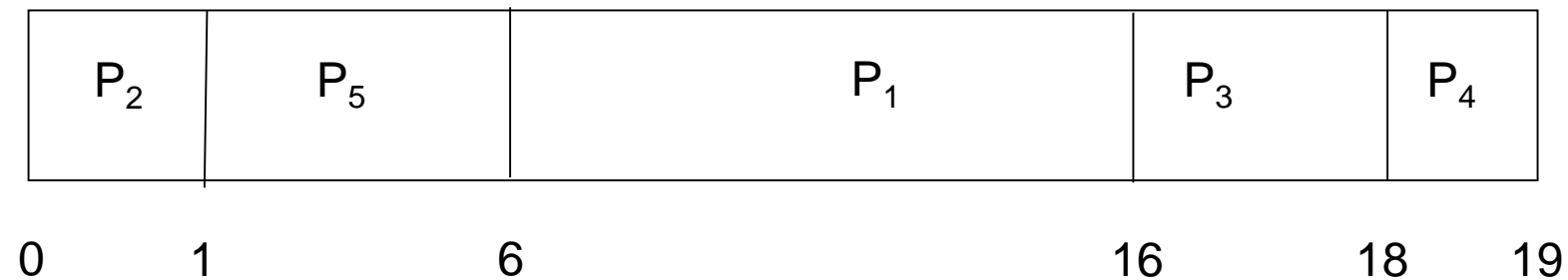




Example of Priority Scheduling

<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>
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



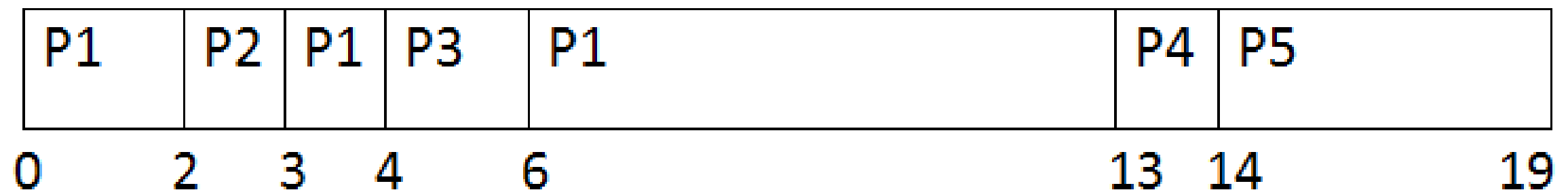


Example of Priority Scheduling

Use Preemption

Process name	Arrival time	Burst Time	Priority
P1	0	10	3
P2	2	1	1
P3	4	2	2
P4	6	1	4
P5	8	5	5

Gant Chart





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

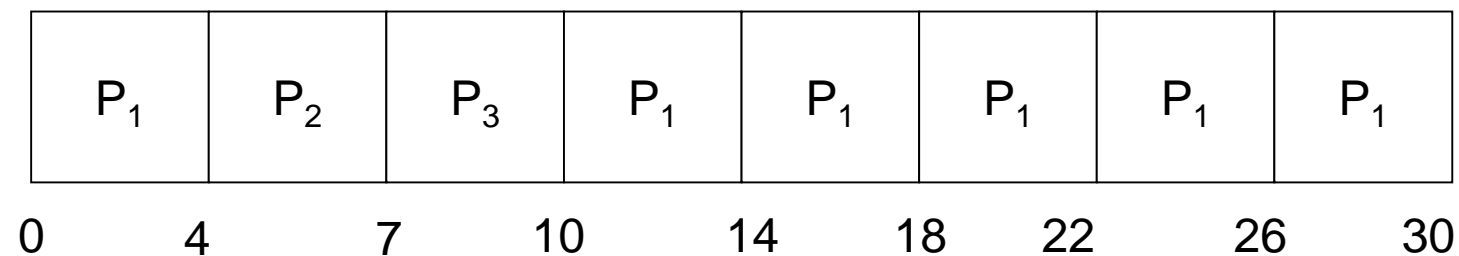




Example of RR with Time Quantum = 4

<u>Process</u>	<u>Burst Time</u>
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

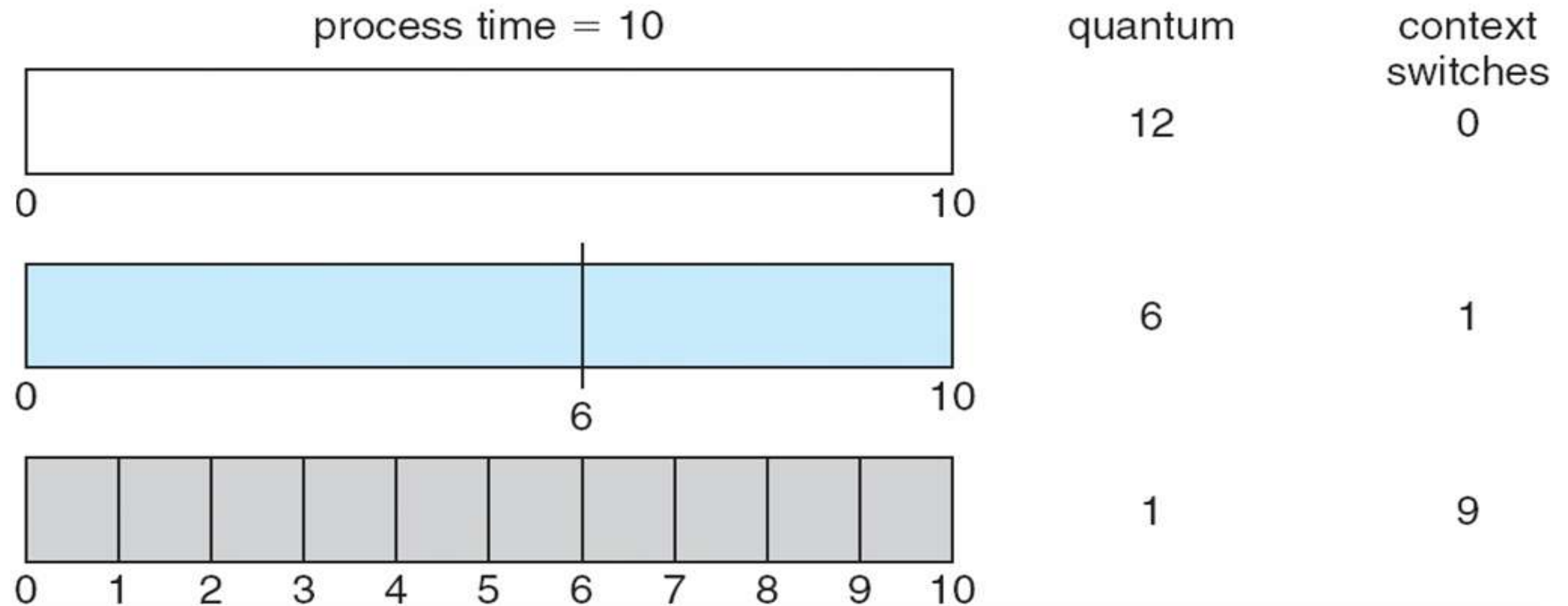


- Typically, higher average turnaround than SJF, but better *response*
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 us sec



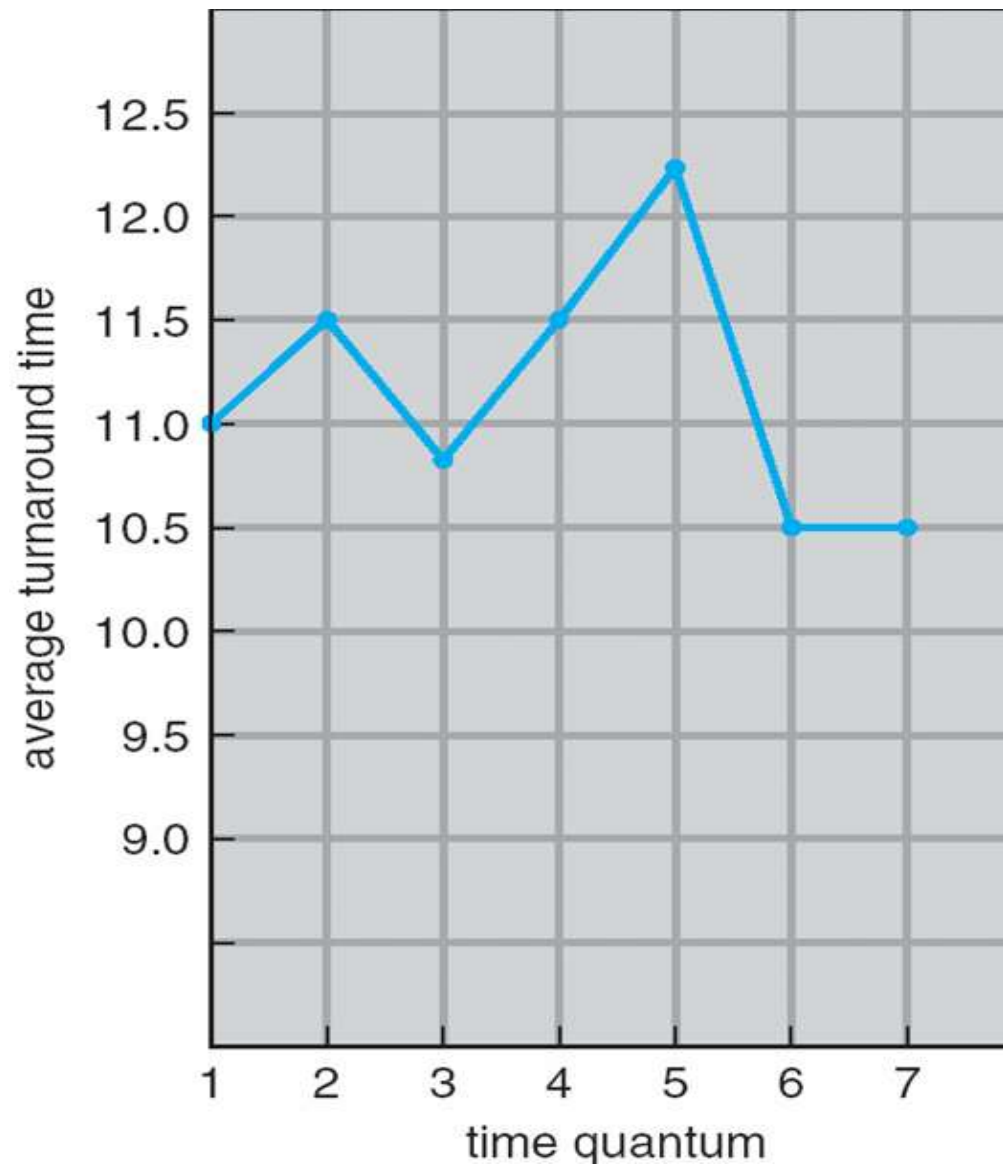


Time Quantum and Context Switch Time





Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts should be shorter than q





Multilevel Queue

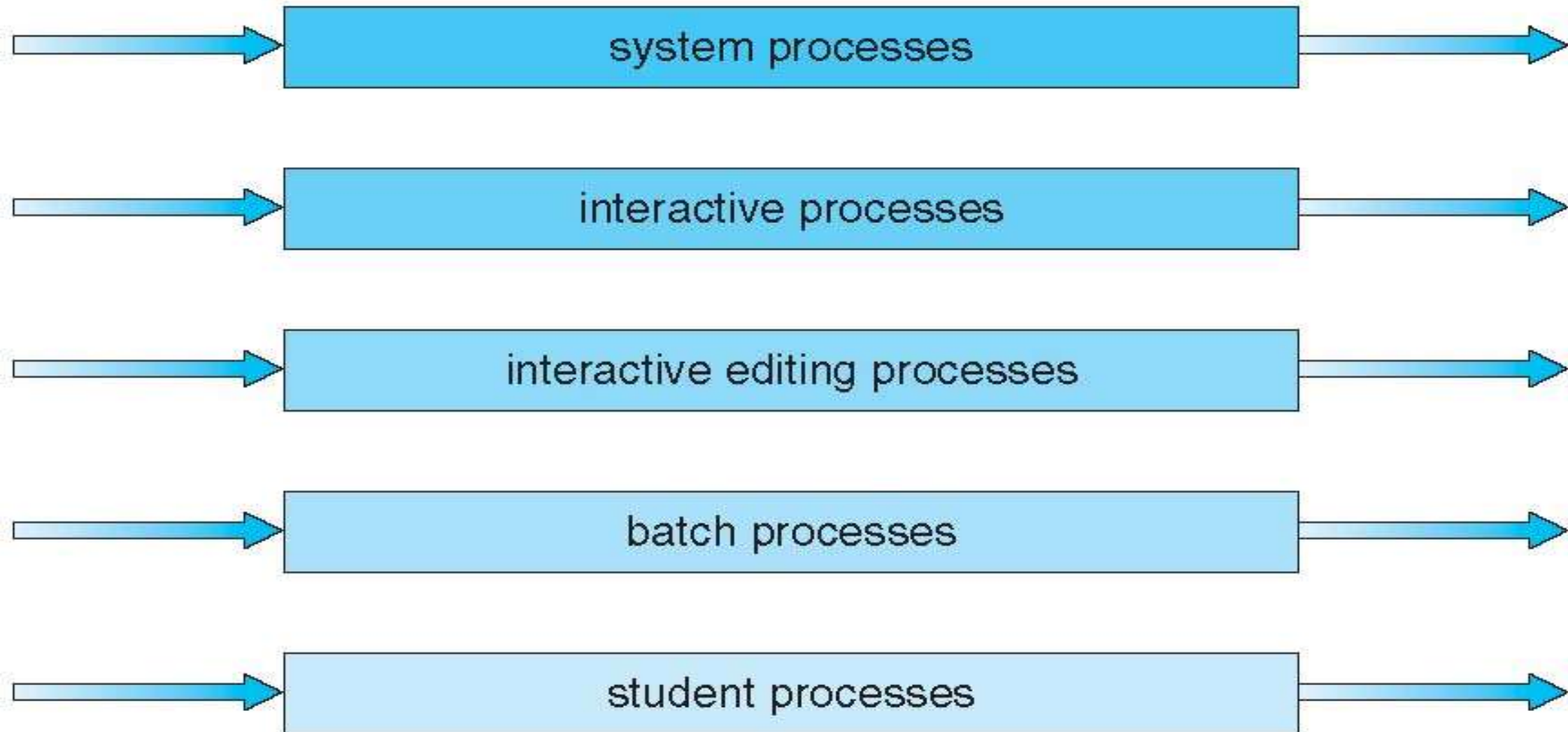
- Ready queue is partitioned into separate queues, eg:
 - foreground (interactive)
 - background (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm:
 - foreground – RR
 - background – FCFS
- Scheduling must be done between the queues:
 - Fixed priority scheduling; (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

highest priority



lowest priority





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





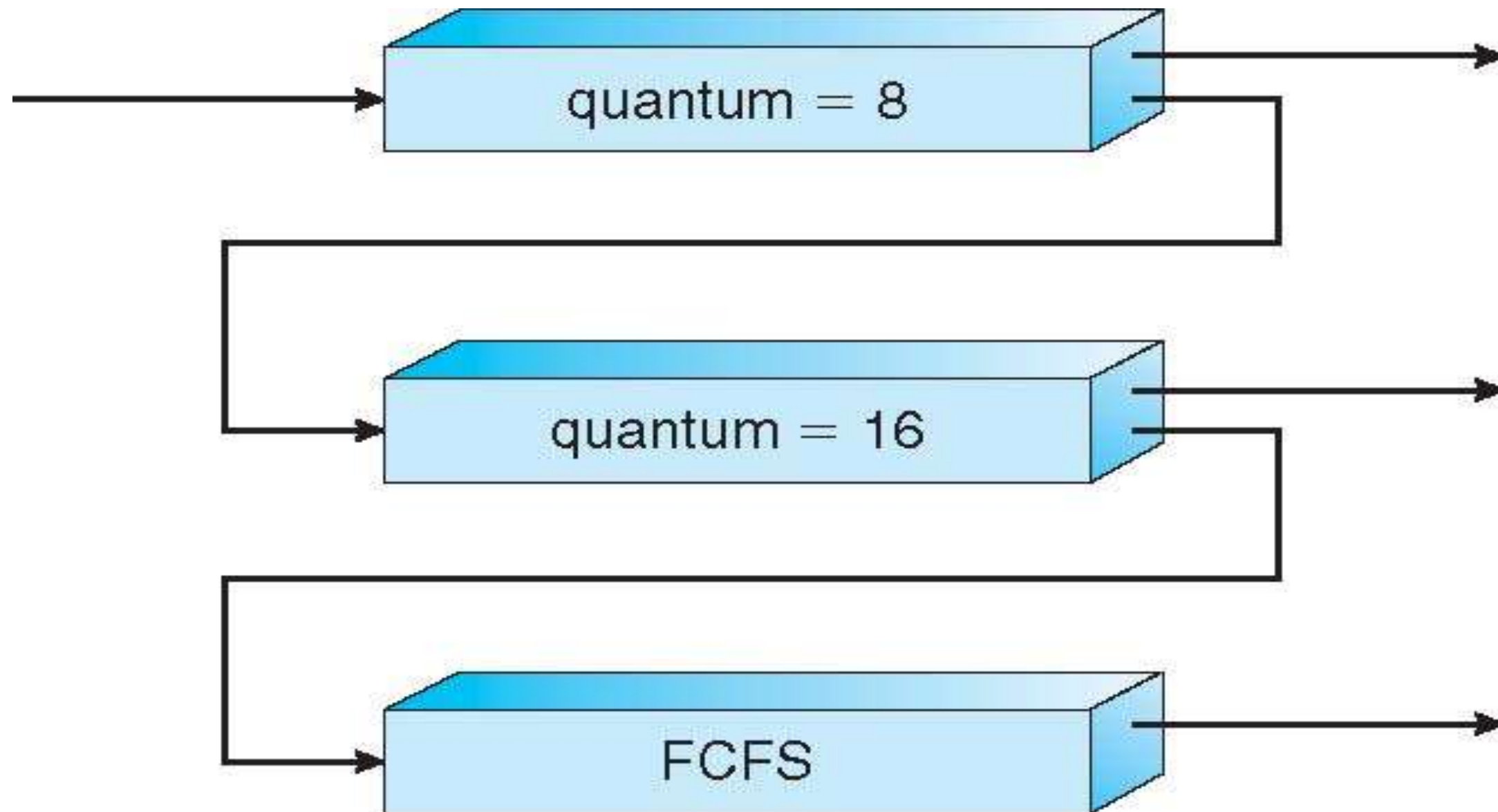
Example of Multilevel Feedback Queue

- 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 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





Multilevel Feedback Queues





Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- **Homogeneous processors** within a multiprocessor
- **Asymmetric multiprocessing** – only one processor accesses the system data structures, alleviating the need for data sharing. Other executing user code.
- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
 - Currently, most common
- **Processor affinity** – process has affinity for processor on which it is currently running. Most systems try to avoid migration of processes from one processor to another and instead attempt to keep a process running on the same processor.
- **soft affinity**
 - **hard affinity** it is possible for a process to migrate between processors.





NUMA and CPU Scheduling

non-uniform memory access

