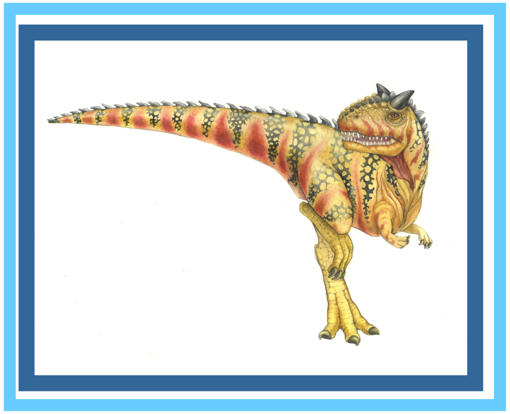
# Chapter 6:  CPU Scheduling



# Copyright

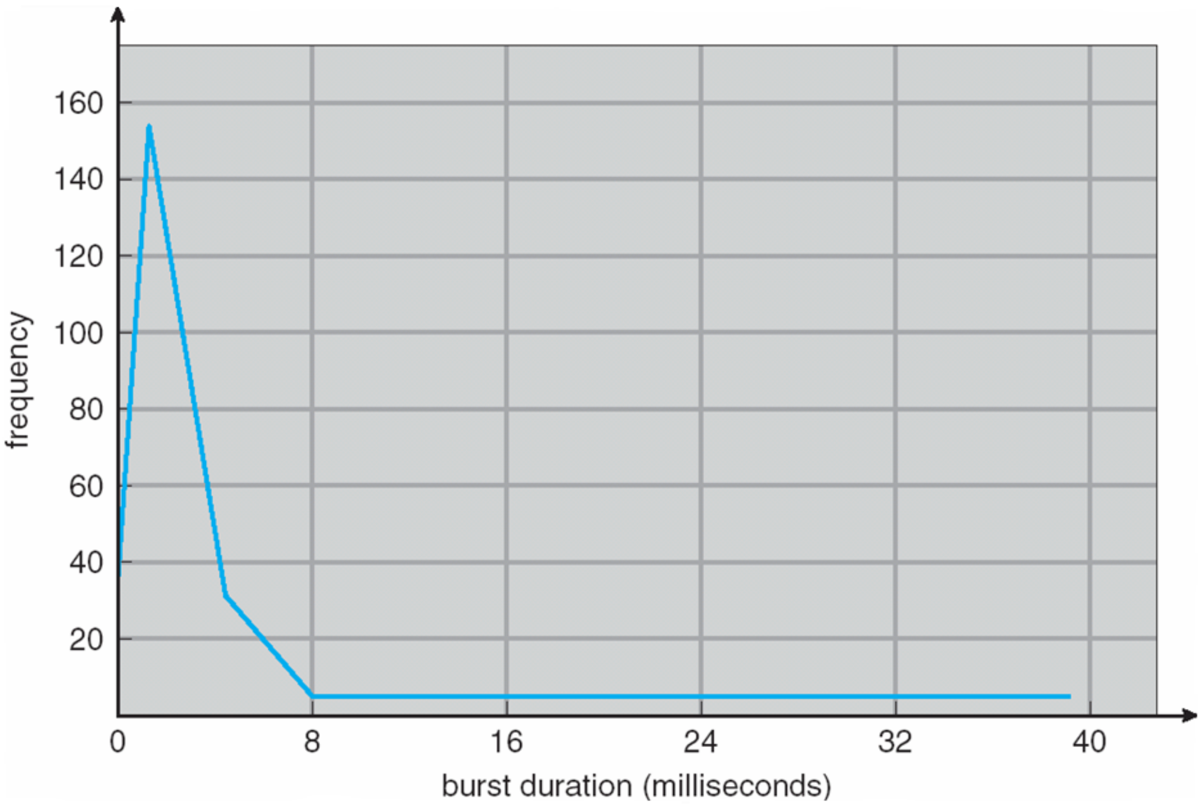
# Chapter 6:  CPU Scheduling

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

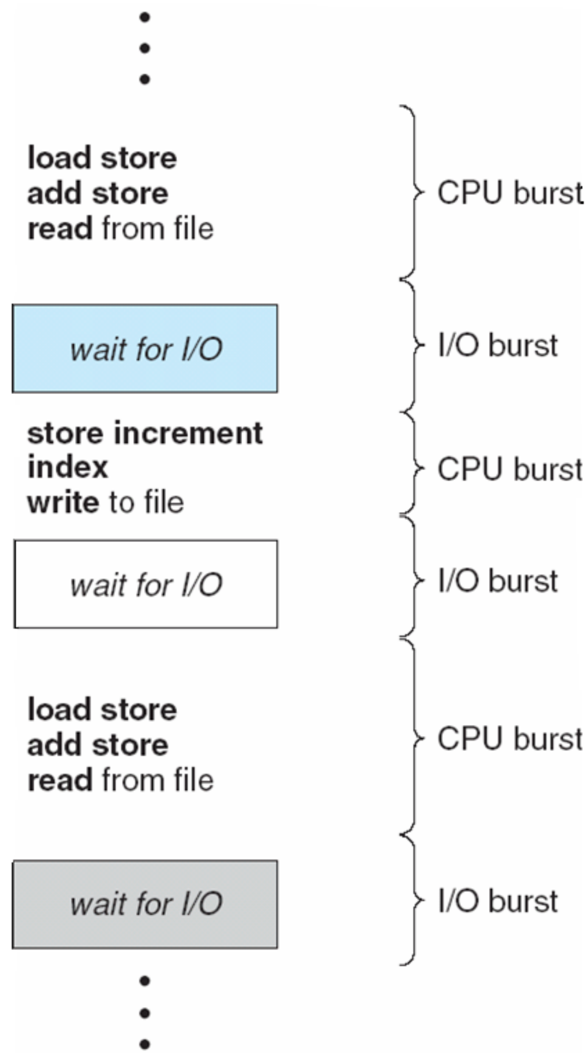
# Basic Concepts

- 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

# Histogram of CPU-burst Times



# Alternating Sequence of CPU And I/O Bursts



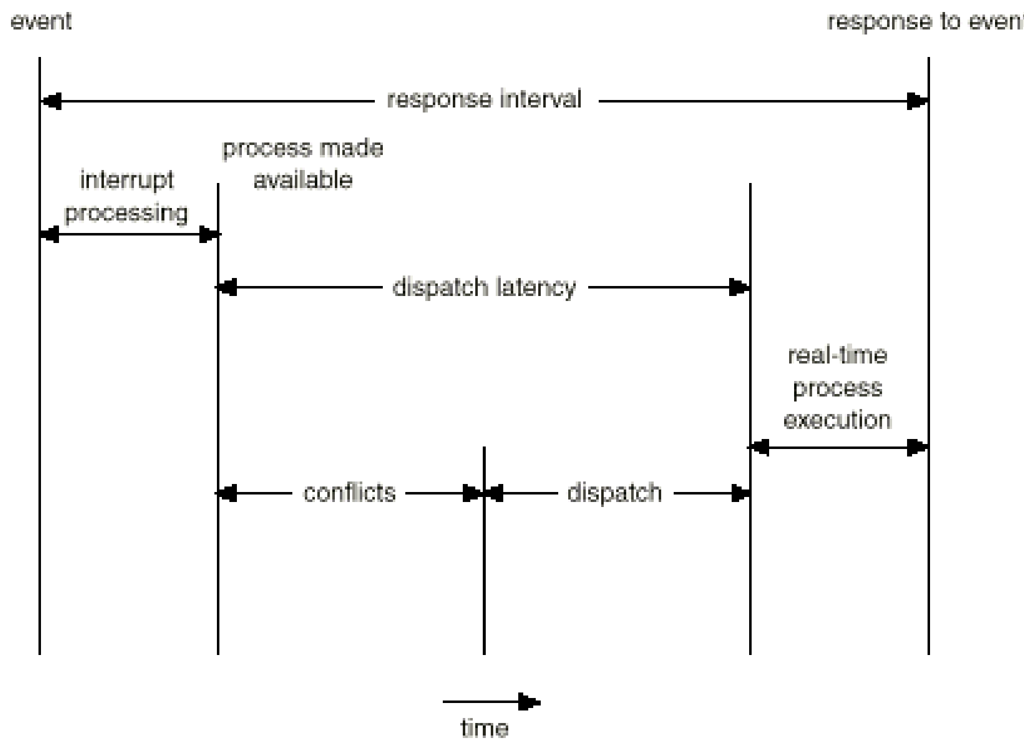
# CPU Scheduler

- 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 that is only allowed under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive**

# Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - RTN, which includes (simultaneously)
    - 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

# Dispatch Latency



# Scheduling Criteria

- **CPU utilization (maximize)** – keep the CPU as busy as possible
- **Throughput (maximize)** – # of processes that complete their execution per time unit
- **Turnaround time (minimize)** – amount of time to execute a particular process
- **Waiting time (minimize)** – amount of time a process has been waiting in the **ready queue**
- **Response time (minimize)** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

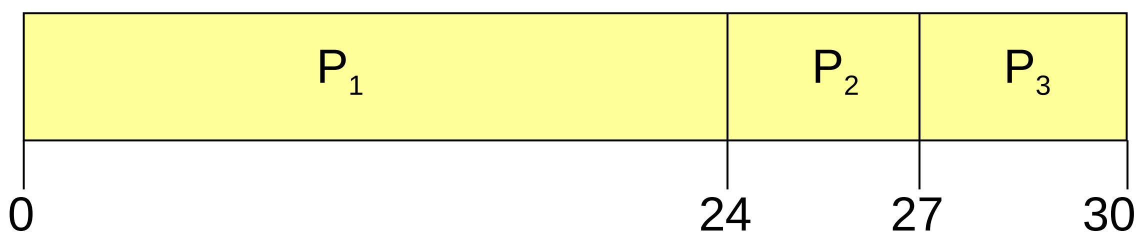
# Scheduling Algorithm Optimization Criteria

- Maximize CPU utilization
- Maximize throughput
- Minimize turnaround time
- Minimize waiting time
- Minimize response time

# First-Come, First-Served (FCFS) Scheduling

| Process | 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:



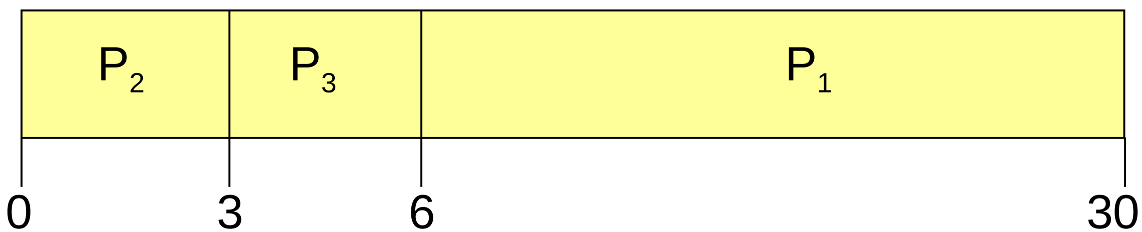

- Waiting time for $P_1$ = 0; $P_2$ = 24; $P_3$ = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

# 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
- Avoids the *Convoy effect:* short process behind long process

# 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

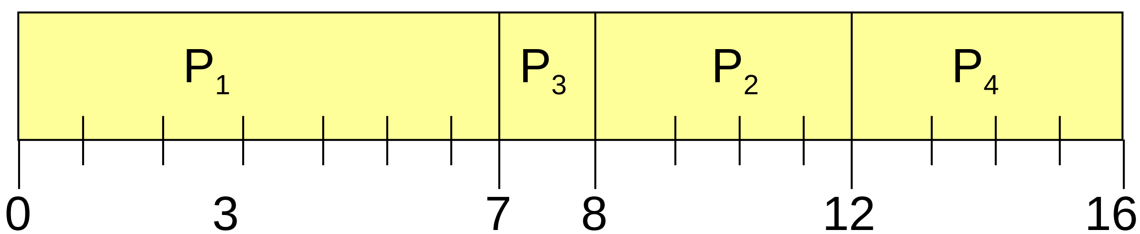
# Shortest-Job-First (SJR) Scheduling

- 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 (withing pre-emptive and non-pre-emptive scheduling algorithms, respectively) – gives minimum average waiting time for a given set of processes
- The difficulty is knowing the length of the next CPU request

# 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 |

- SJF (non-preemptive)

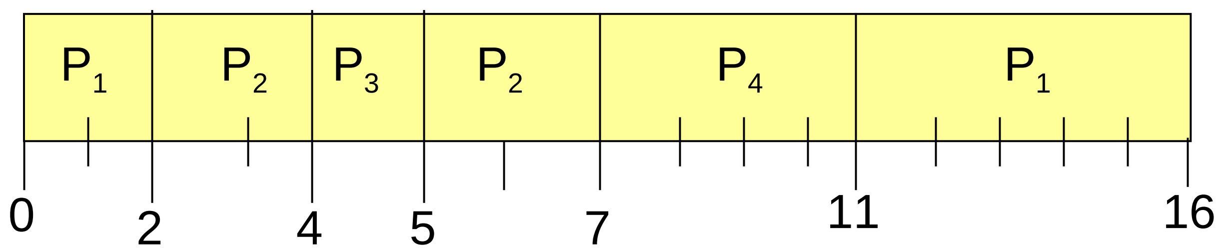



- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

# 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 |

- SJF (preemptive)



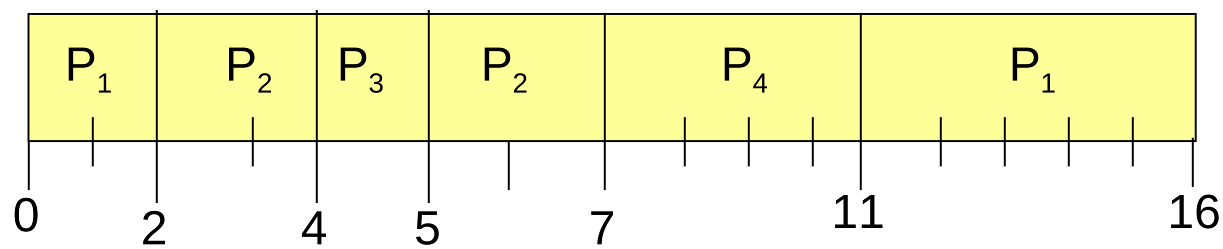

Average waiting time = (9 + 1 + 0 +2)/4 = 3 =

= ((11 + 7 + 5) – (0 + 2 + 4 + 5))/4

# 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 |

- SJF (preemptive)




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

= ((11 + 7 + 5) – (0 + 2 + 4 + 5))/4

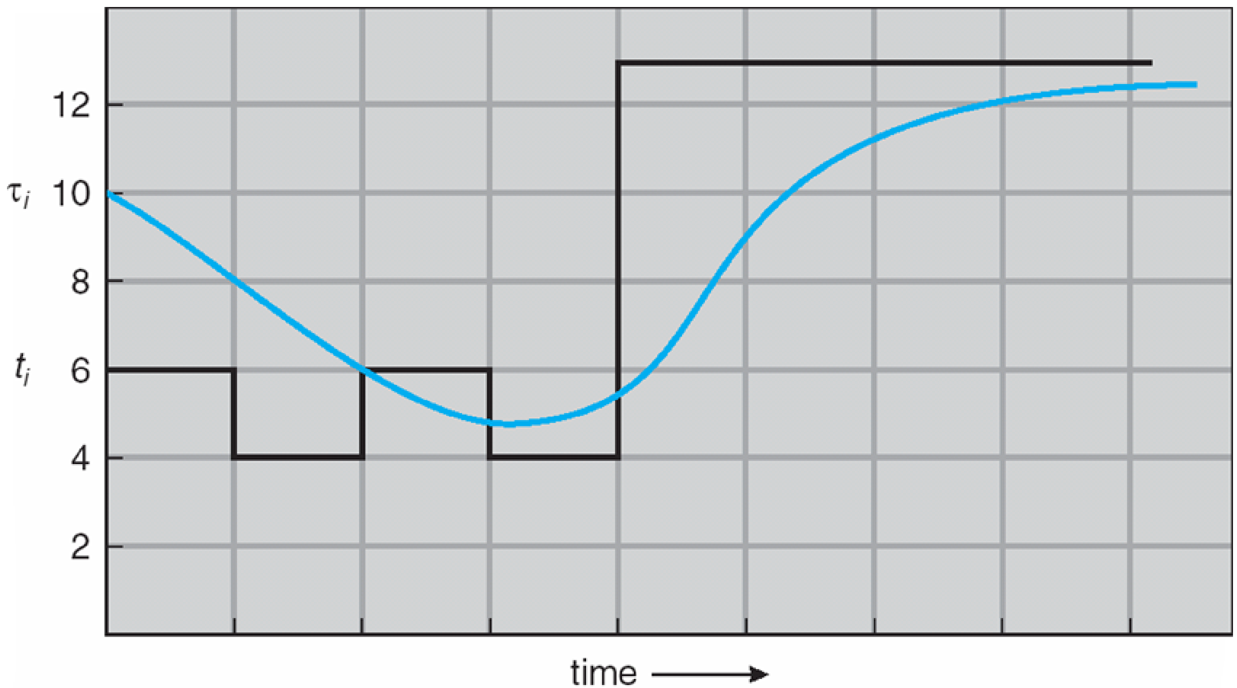
$$W_{avg} = \frac{1}{n}\left(\sum_{i=1}^{n-1} T_{depart}(P_i) - \sum_{i=1}^{n} T_{arrive}(P_i)\right)$$

# Determining Length of Next CPU Burst

- Can only estimate the length
- Can be done by using the length of previous CPU bursts, using exponential averaging

1. $t_n =$ actual length of $n^{th}$ CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, \ 0 \leq \alpha \leq 1$
4. Define: $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n .$

# Prediction of the Length of the Next CPU Burst




| CPU burst ($t_i$) | | 6 | 4 | 6 | 4 | 13 | 13 | 13 | … |
|---|---|---|---|---|---|---|---|---|---|
| "guess" ($\tau_i$) | 10 | 8 | 6 | 6 | 5 | 9 | 11 | 12 | … |

# Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

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

- Since both $\alpha$ and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

# 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
  - non-preemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem: How to avoid **starvation**? (Starvation in this case occurs when a low priority ready processes is never selected for running.)
- Solution: **Aging** – as time progresses increase the priority of the processes wating in the ready queue.

# 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 in ready queue more than $(n-1)q$ time units.
- Performance
  - $q$ large $\Rightarrow$ FIFO
  - $q$ small $\Rightarrow$ thrashing ($q$ must be large with respect to context switch, otherwise overhead is too high)

# Round Robin (RR)

**The purpose of RR is to approximate SJF**

**by classifying jobs as short (CPU burst not larger than Q) and long (otherwise)**

# Example of RR with Time Quantum = 20

| Process | Burst Time |
|---|---|
| $P_1$ | 53 |
| $P_2$ | 17 |
| $P_3$ | 68 |
| $P_4$ | 24 |

- The Gantt chart is:

| $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_1$ | $P_3$ | $P_4$ | $P_1$ | $P_3$ | $P_3$ |
|---|---|---|---|---|---|---|---|---|---|

0 20 37 57 77 97 117 121 134 154 162

- W = (134 + 121 + 37)/4 = 292/4 = 73
- Higher average turnaround than SJF
- Favors processes with CPU bursts <= Q

# Example of RR with Time Quantum = 20

The Gantt chart is:

| $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_1$ | $P_3$ | $P_4$ | $P_1$ | $P_3$ | $P_3$ |
|---|---|---|---|---|---|---|---|---|---|

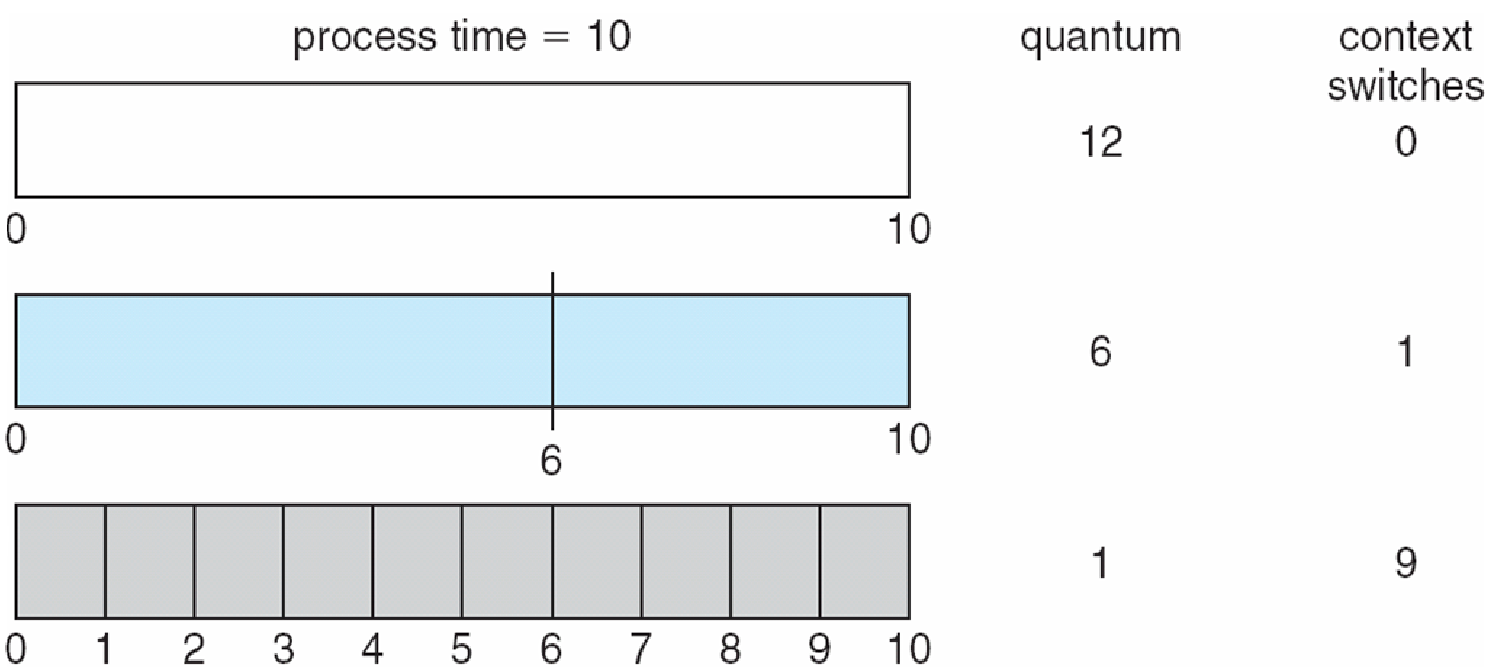
0 20 37 57 77 97 117 121 134 154 162

W = (134 + 121 + 37)/4 = 292/4 = 73

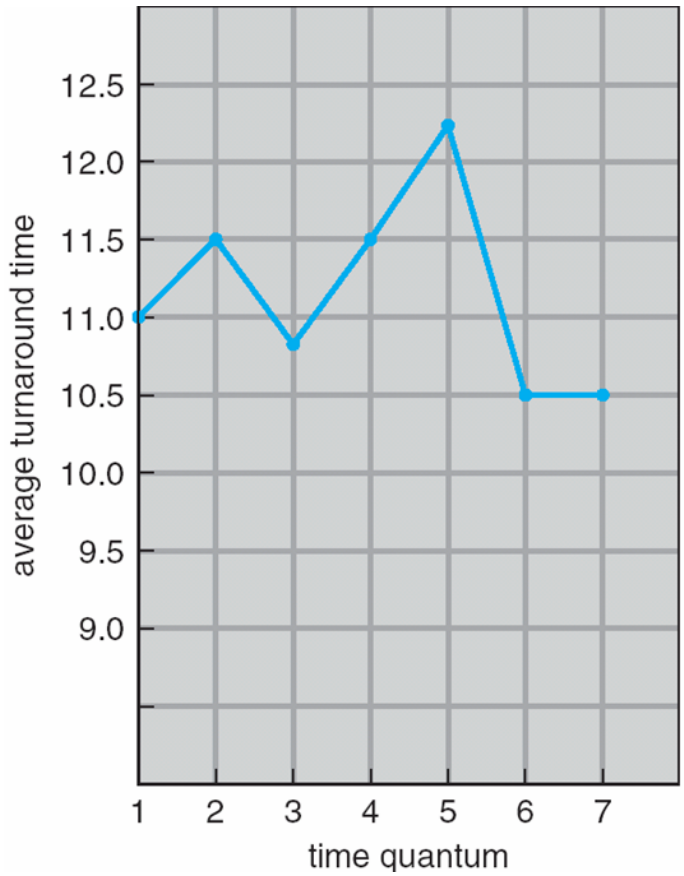
Remember:

$$W_{avg} = \frac{1}{n}\left(\sum_{i=1}^{n-1} T_{depart}(P_i) - \sum_{i=1}^{n} T_{arrive}(P_i)\right)$$

# 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 |

# Turnaround Time Varies With The Time Quantum

**Same scenario, but …**

CPU bursts: 6 3 7 1

Q = 3 T = 11.5 W = 7.25
Q = 6 T = 12   W = 7.75
Q = 7 T = 12   W = 7.75

# Turnaround Time Varies With The Time Quantum

**Same scenario, but ...**

CPU bursts: 6 7 3 1

Q = 3 T = 12.25 W = 8
Q = 6 T = 13.25 W = 9
Q = 7 T = 13 W = 8.75

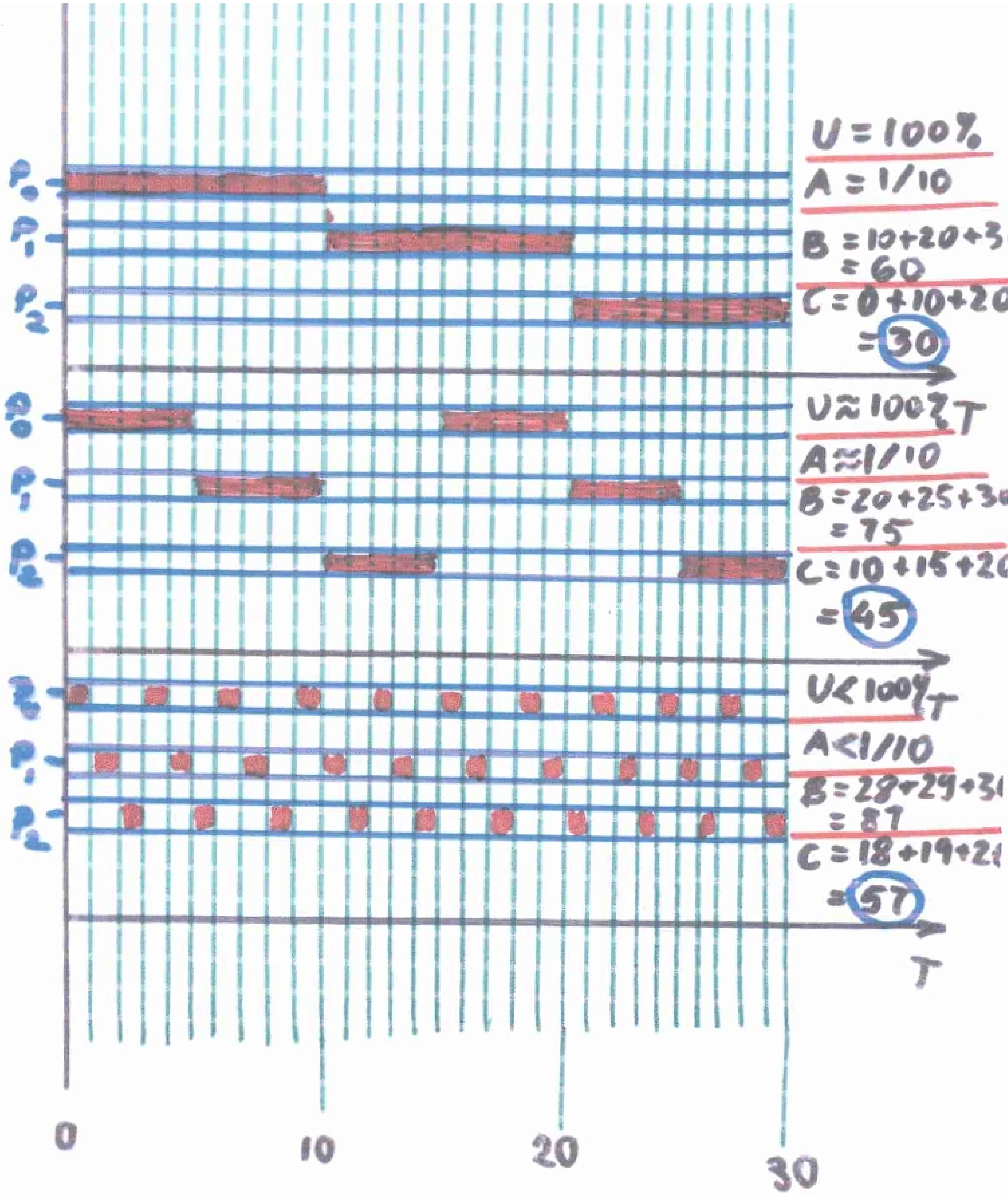
# Turnaround Time Varies With The Time Quantum

CPU bursts: 7 6 3 1

Q = 3 T = 13 W = 8.75
Q = 6 T = 15 W = 10.75
Q = 7 T = 13.25 W = 9

CPU bursts: 60 3 70 1

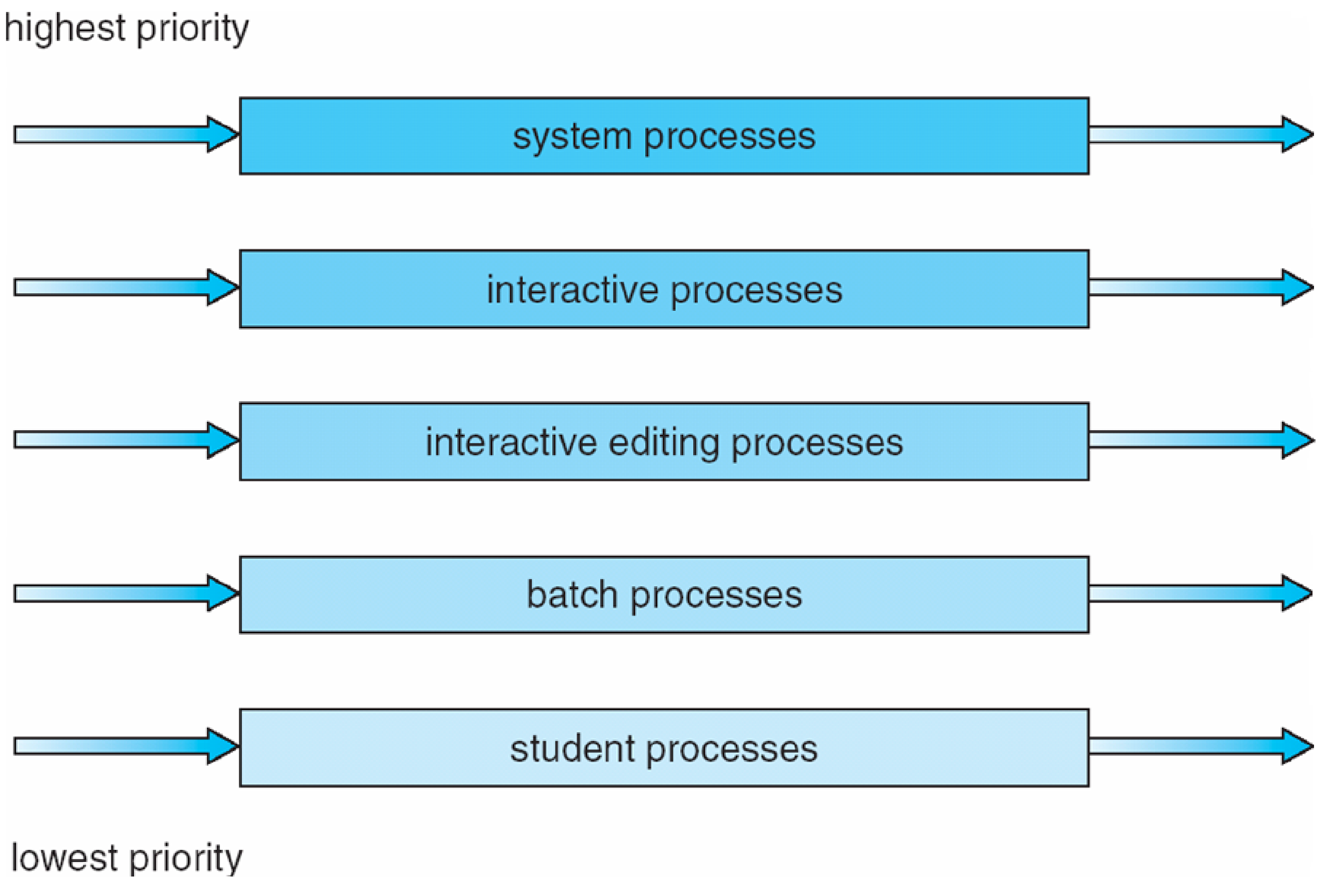
Q = 3 T = 66.25 W = 32.75
Q = 6 T = 95.25 W = 62.25
Q = 7 T = 97.4 W = 64.5


P0
P1
P2
U = 100%
A = 1/10
B = 10+20+3
= 60
C = 0+10+20
= 30
U ≈ 100%
T
A ≈ 1/10
B = 20+25+3
= 75
C = 10+15+2
= 45
U < 100%
T
A < 1/10
B = 28+29+3
= 87
C = 18+19+2
= 57
T
0
10
20
30

# 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



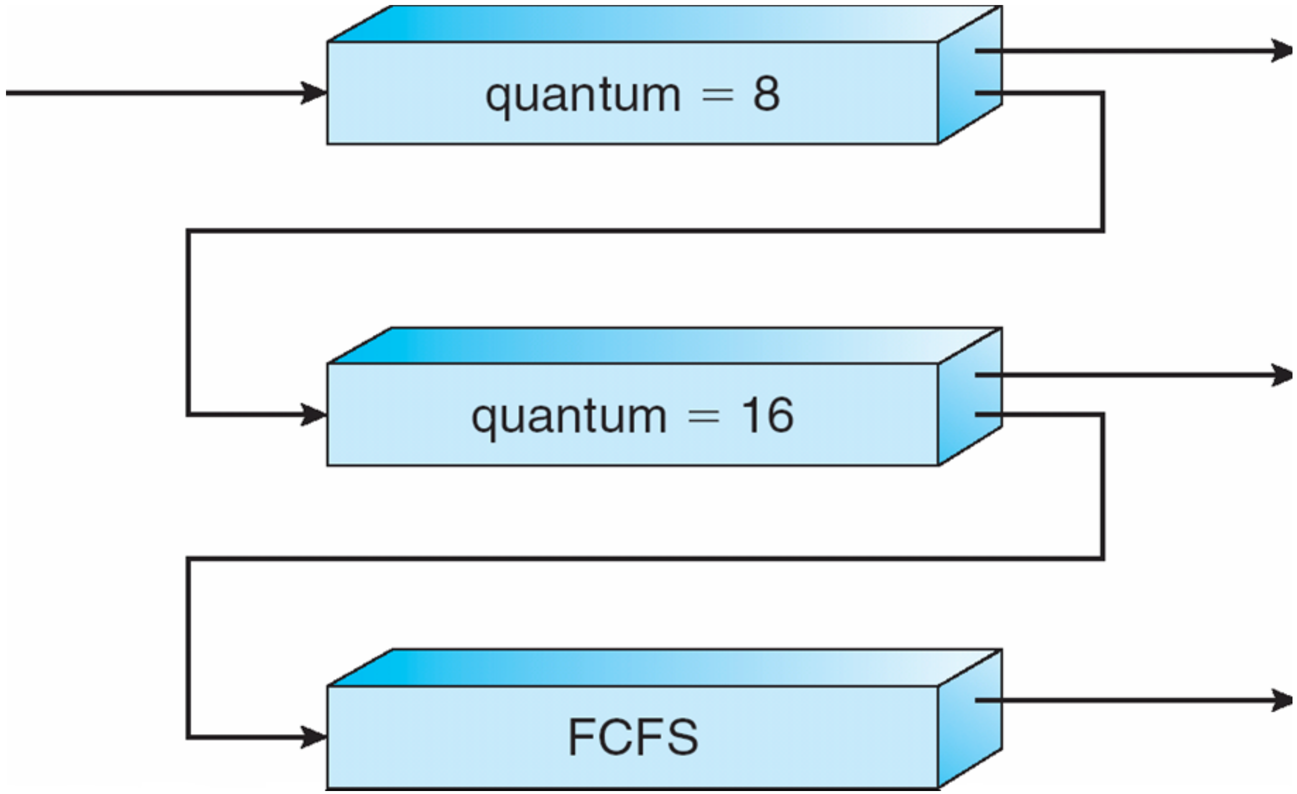
# Multilevel Feedback Queue

- 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

# 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



# Multilevel Feedback Queues

Provide more flexible approximations of the SJF scheduling algorithm.

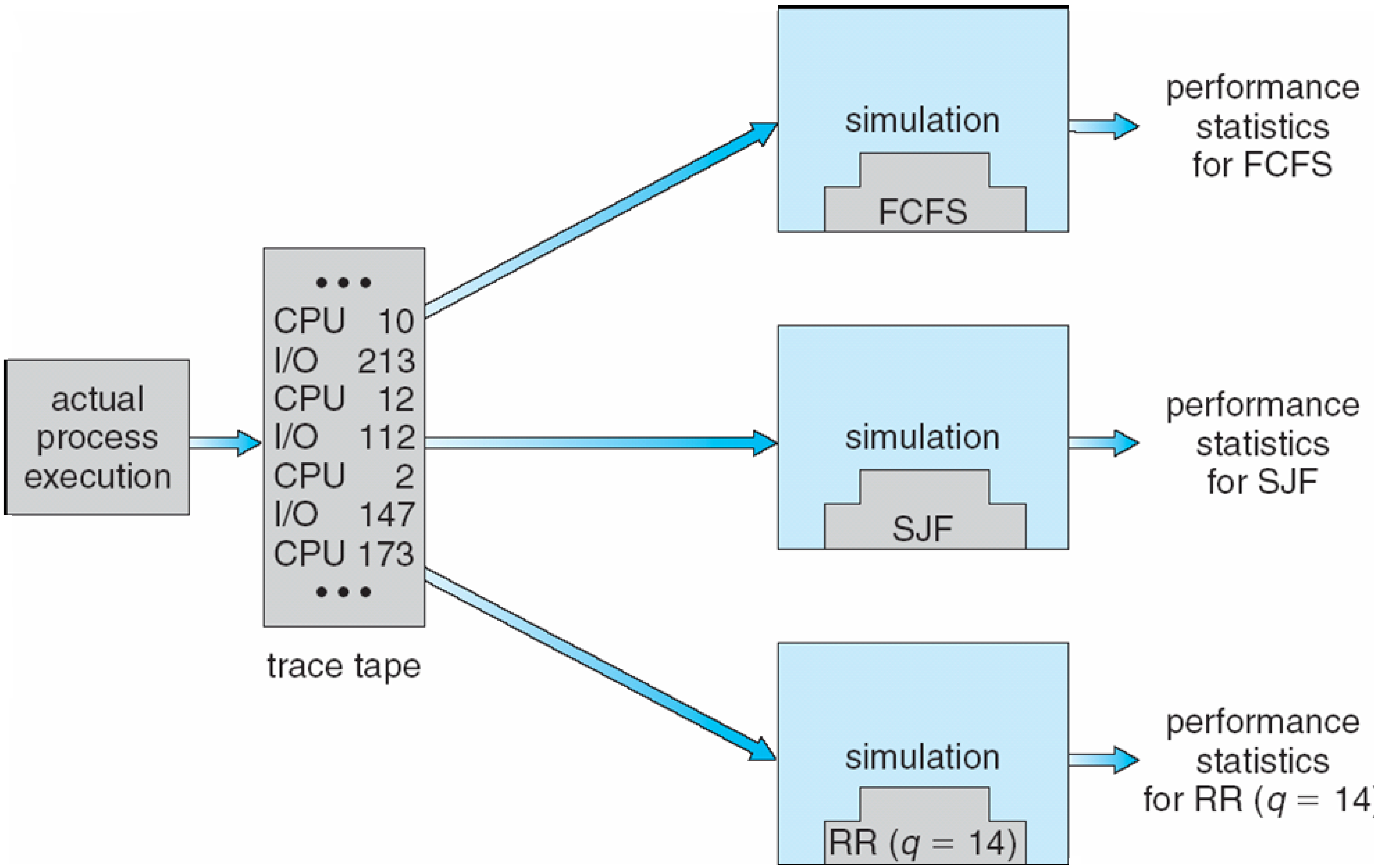
# Multiple-Processor Scheduling

- 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
- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
- **Processor affinity** – process has affinity for processor on which it is currently running
  - **soft affinity**
  - **hard affinity**

# Algorithm Evaluation

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

# Evaluation of CPU schedulers by Simulation



# End of Chapter 6