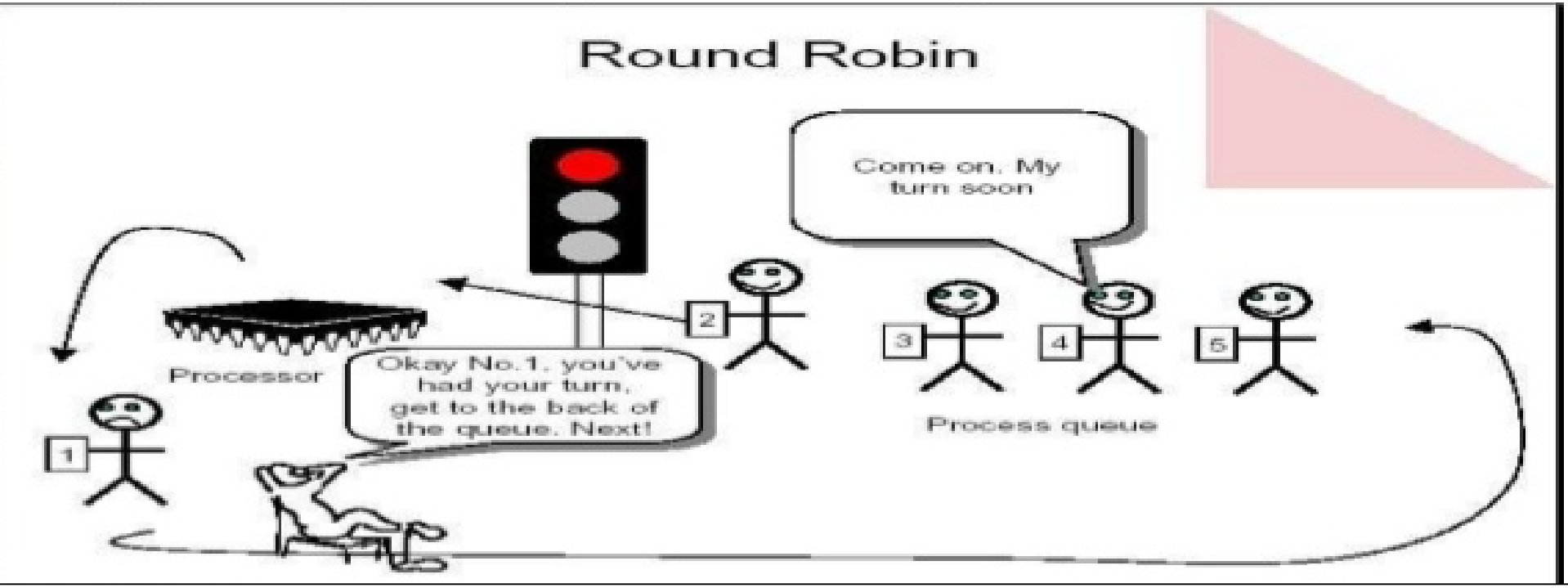
RR

- Disadvantages:

Setting the quantum too short, increases the overhead and lowers the CPU efficiency, but setting it too long may cause poor response to short processes.

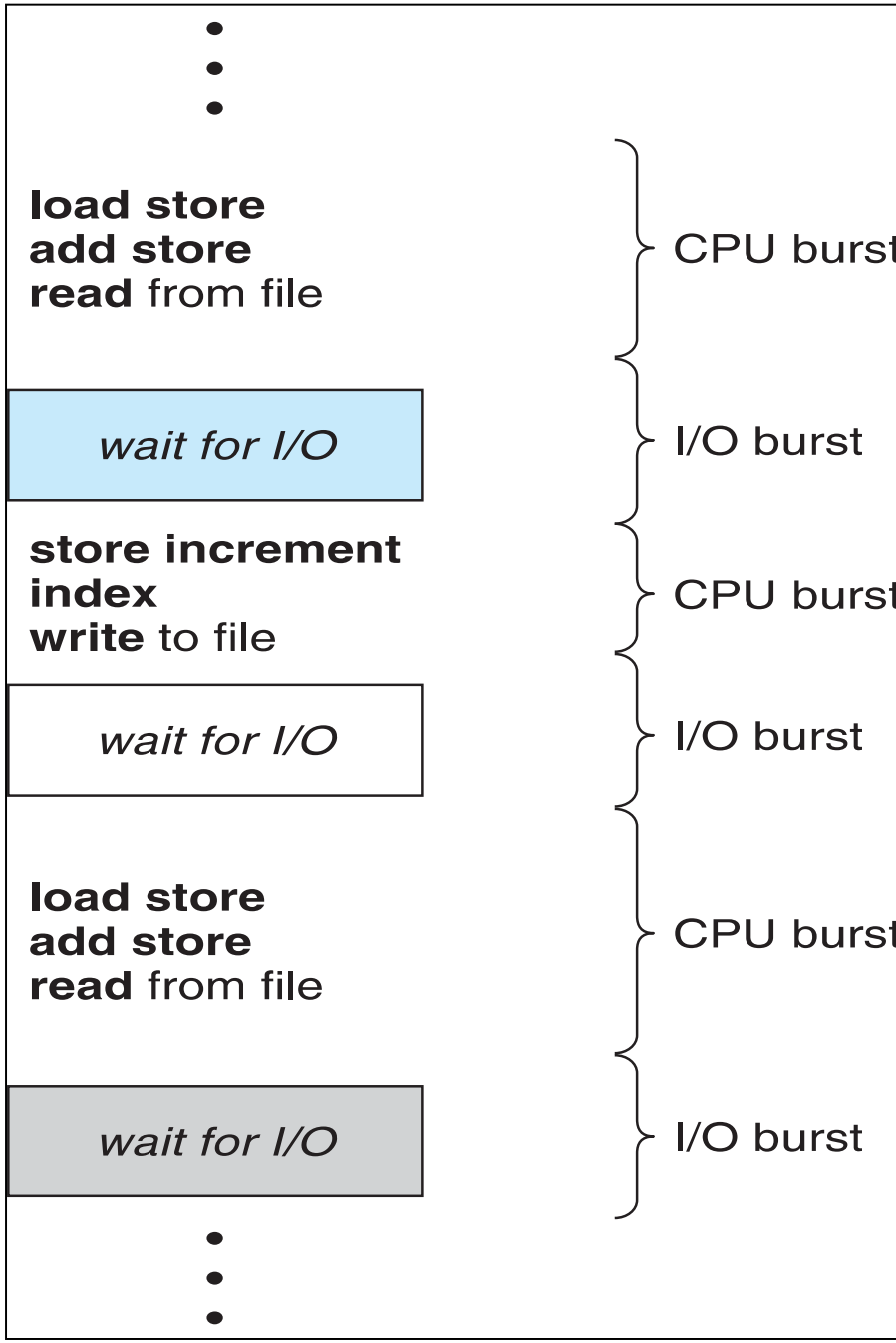


Objectives

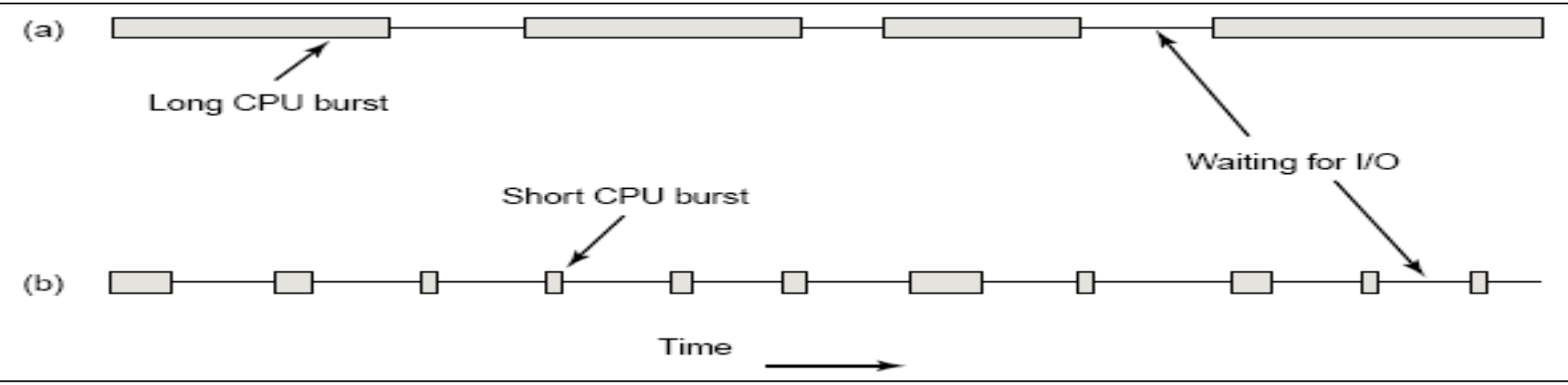
- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
- To examine the scheduling algorithms of several operating systems

Basic Concepts

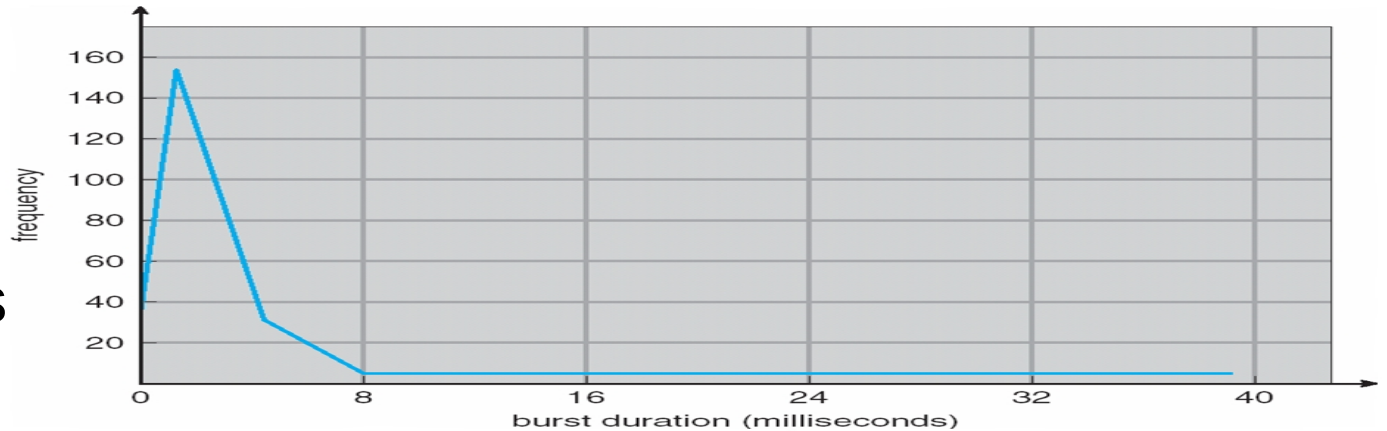
- Maximum CPU utilization obtained with multiprogramming
- Most processes exhibit the following behavior:
- **CPU burst** followed by **I/O burst**
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- CPU burst distribution is of main concern



- Maximum CPU utilization obtained with multiprogramming
- **CPU–I/O Burst Cycle**: Process execution consists of a *cycle* of CPU execution and I/O wait
  - Alternating Sequence of CPU and I/O Bursts



Histogram of CPU-burst Times

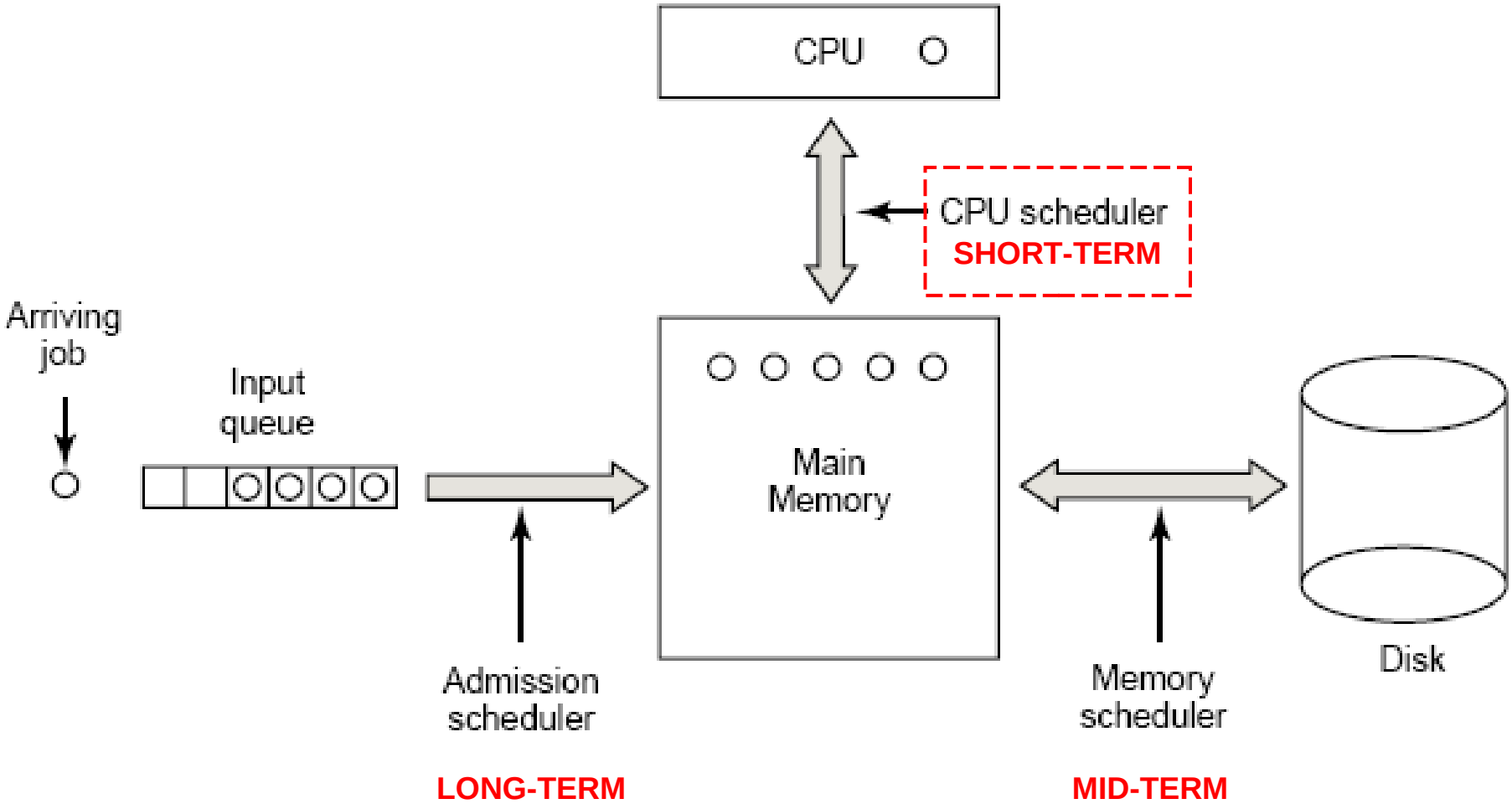


- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready state
  4. Terminates

CPU Scheduler

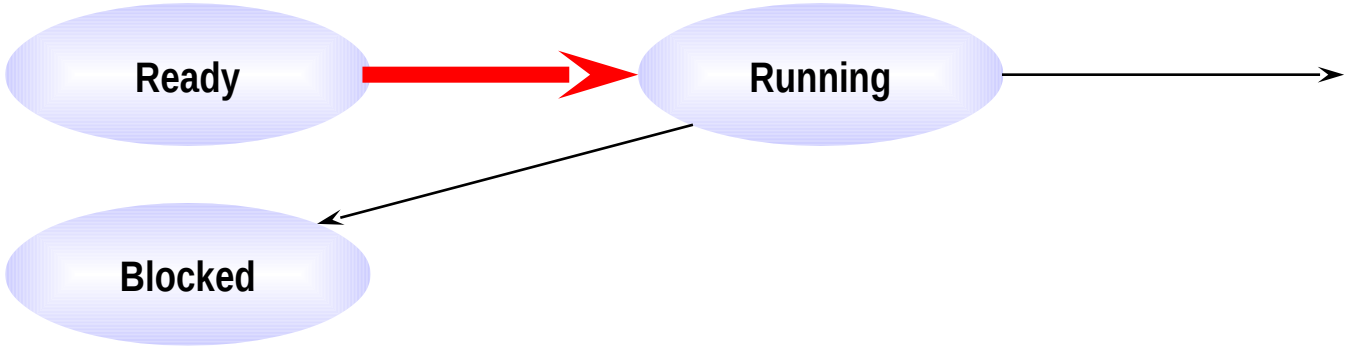
- Scheduling under 1 and 4 is **nonpreemptive**
  - Processes keep CPU until it releases either by terminating or I/O wait.
  - All other scheduling is **preemptive**
  - Interrupts

Three-level Scheduling

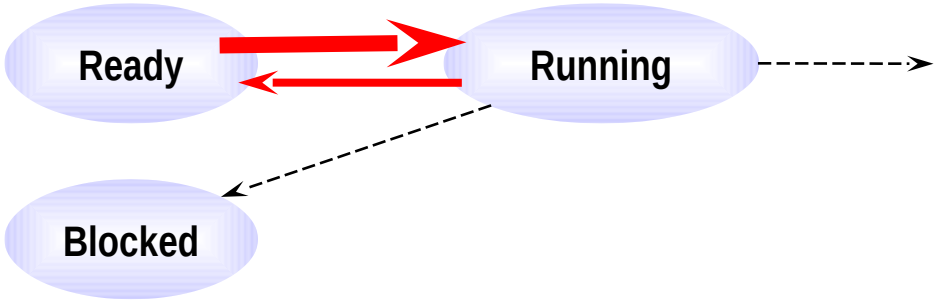


Two kinds of CPU-scheduling algorithms

- **cooperative** scheduling
  - let a process run until it blocks on I/O, terminates or voluntarily releases the CPU (system call)



- **preemptive** scheduling
  - follow clock interrupts (ex: 50Hz) to forcibly switch processes (demote the “Running” to “Ready”)



Nonpreemptive Scheduling

- Once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU:
  - either by terminating
  - or by switching to the waiting state.

Preemptive scheduling

- Preemptive scheduling can result in race conditions when data are shared among several processes.
  - **Consider the case of two processes that share data**. While one process is updating the data, it is preempted so that the second process can run. The second process then tries to read the data, which are in an inconsistent state.
  - **Consider process preemption while in kernel mode (next slides…..)**
  - **Consider interrupts occurring during crucial OS activities (next slides…..)**
- Virtually all modern operating systems including Windows, Mac OS X, Linux, and UNIX use preemptive scheduling algorithms.

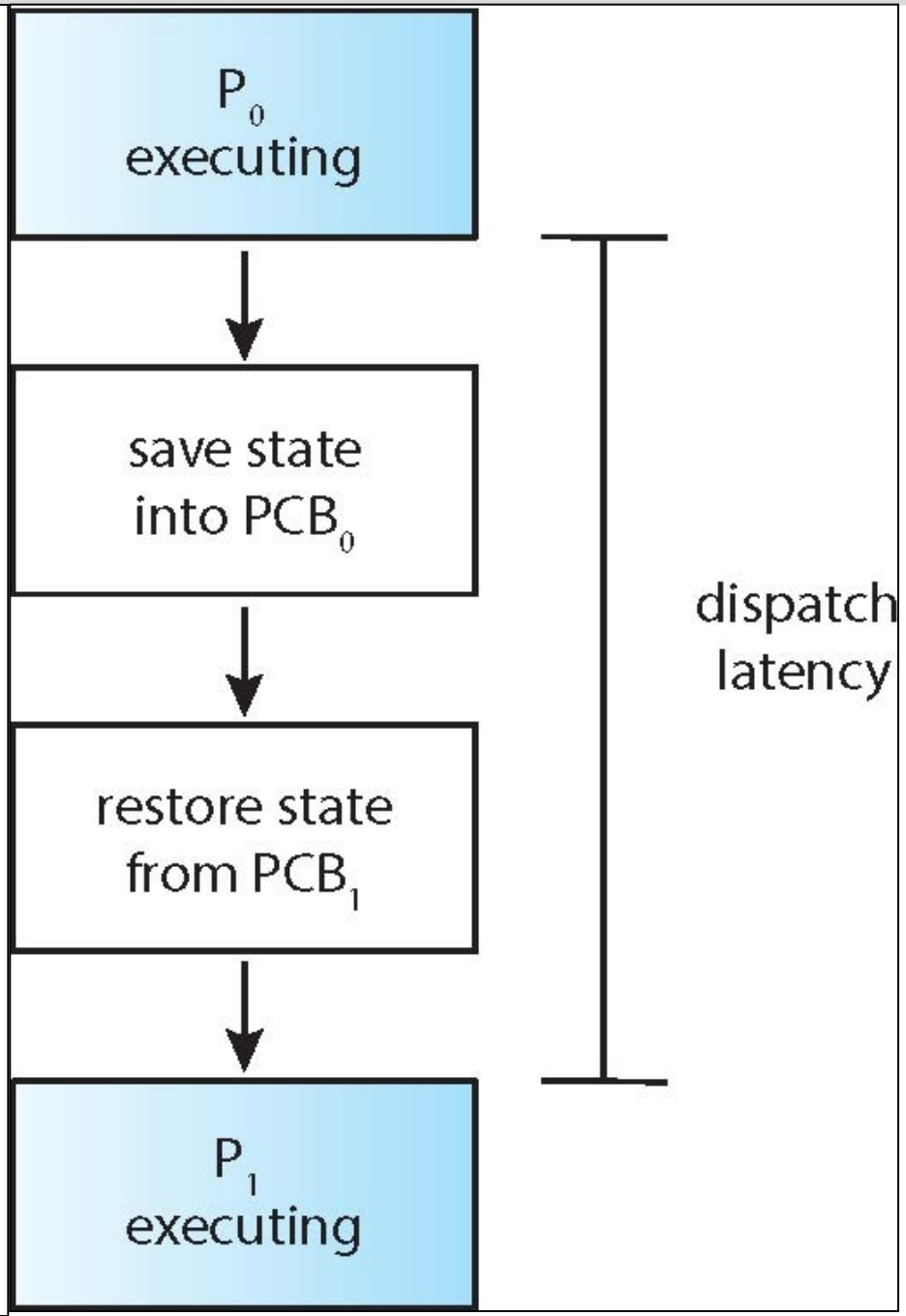
Why process running in kernel mode cannot be preempted ?

What is the difference between Non-preemptive, Preemptive and Selective Preemptive Kernel?

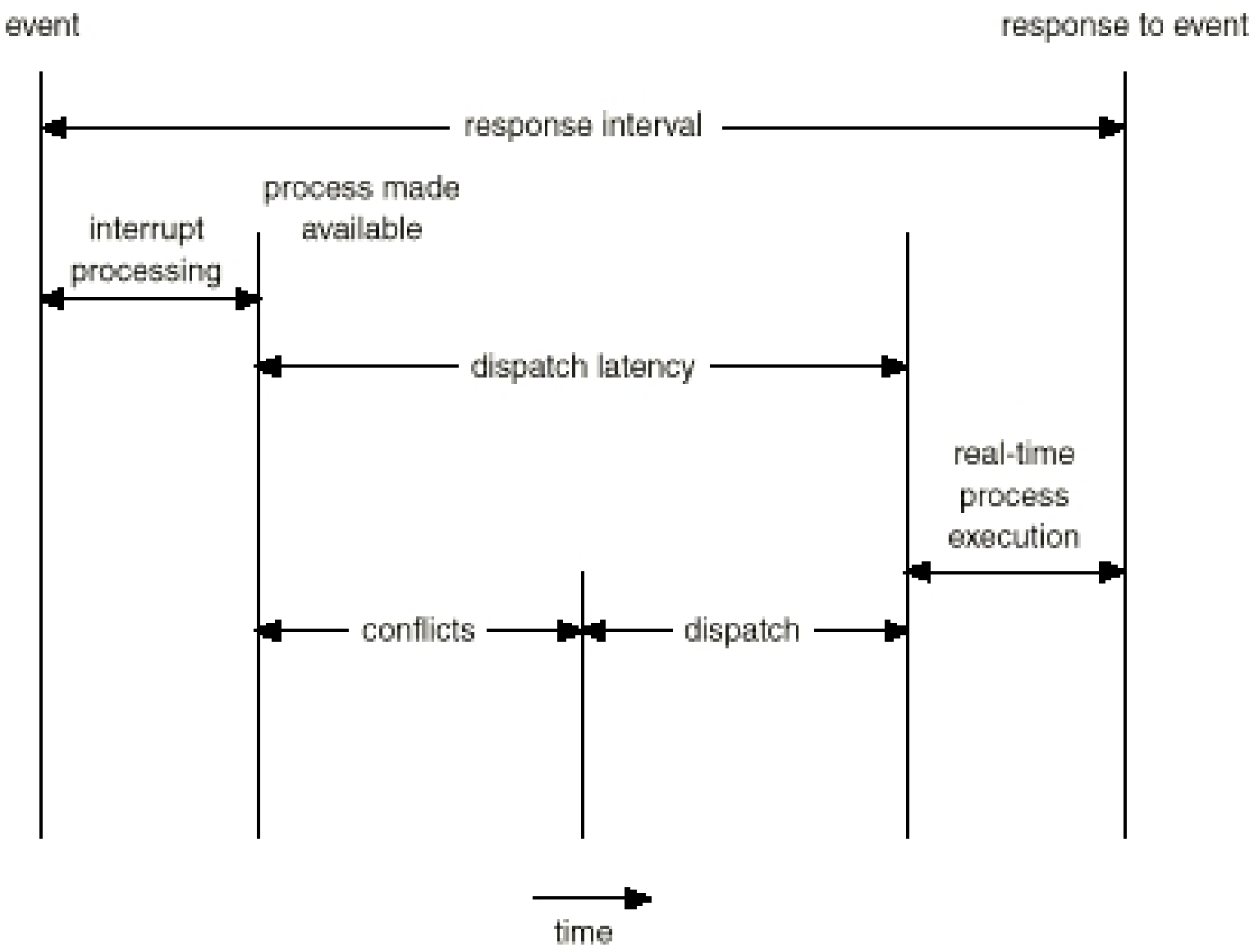
- Process A executes an exception handler, Process B gets awaken by an IRQ request, the kernel replaces process A with B (a forced process switch).
- Process A is left unfinished. The scheduler decides afterwards if process A gets CPU time or not.
- On a non-preemptive kernel, process A would just have used all the processor time until he is finished or voluntarily decides to allow other processes to interrupt him (a planned process switch).
- Today's Linux based operating systems generally do not include a fully preemptive kernel, there are still critical functions which have to run without interruption.
- So I think you could call this a "selective preemptive kernel".

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the CPU 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



Dispatcher

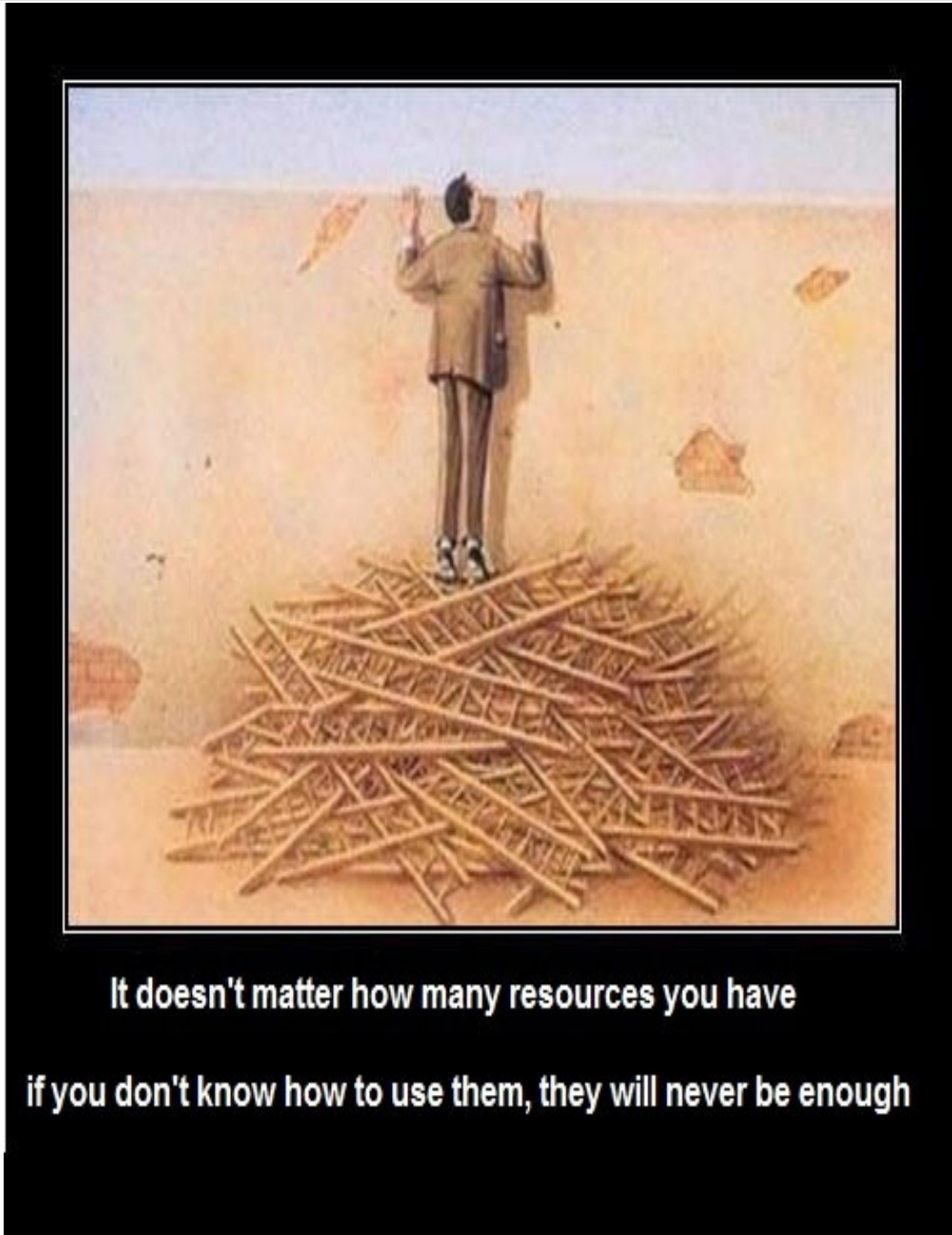

event
response to event
response interval
process made available
interrupt processing
dispatch latency
real-time process execution
conflicts
dispatch
time

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
  - Typically between 40% to 90%
- **Throughput** – # of processes that complete their execution per time unit
  - Depends on the length of process
- **Turnaround time** – amount of time to execute a particular process (Min turnaround time)
  - Sum of wait for memory, ready queue, execution, and I/O.
- **Waiting time** – amount of time a process has been waiting in the ready queue
  - Sum of wait in 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

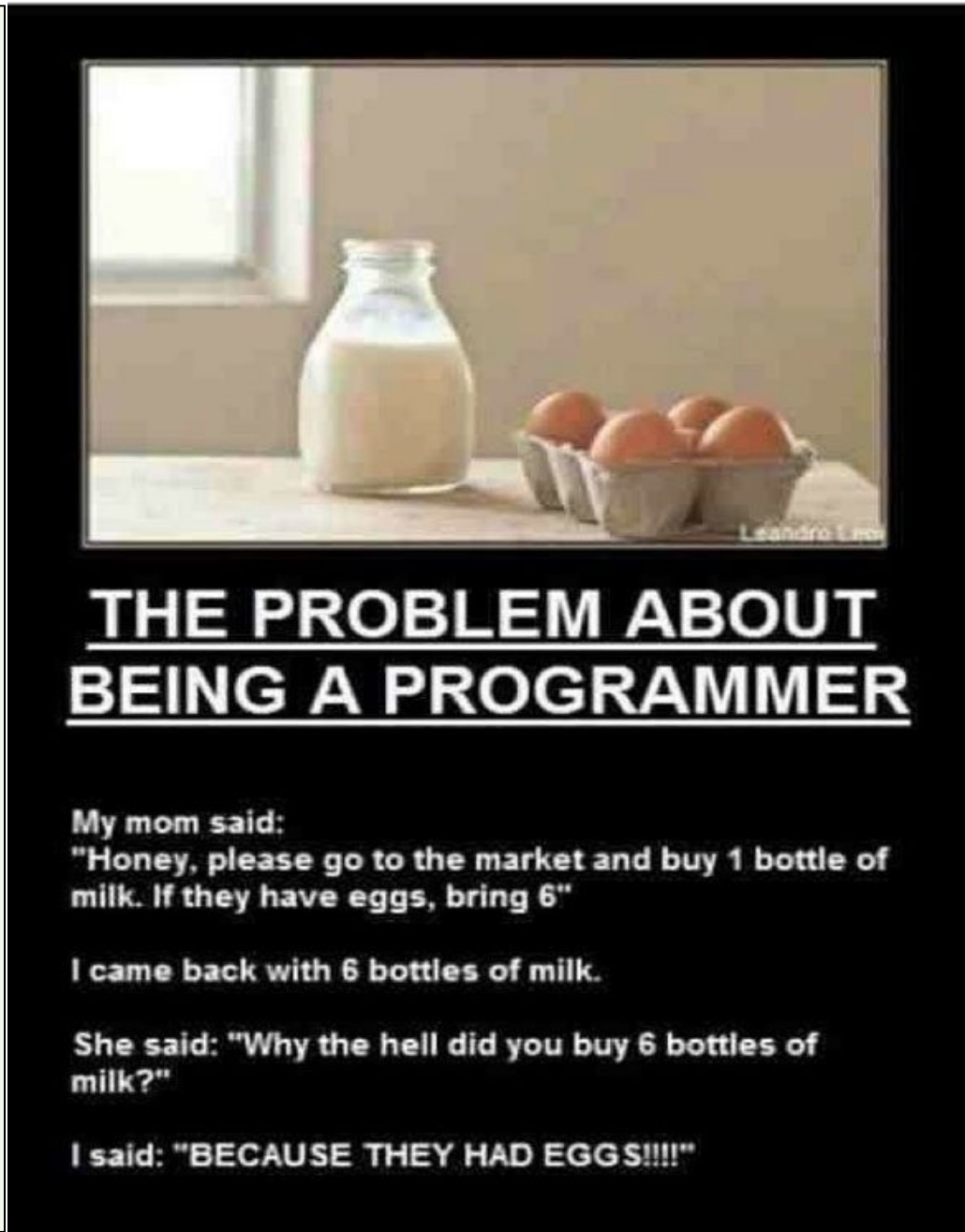
Optimization Criteria for Scheduling

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



Scheduling Algorithms

- First –come, First-serve (FCFS)
- Shortest-Job-First Scheduling (SJF)
- Round-Robin Scheduling (RR)
- Priority Scheduling
- Multilevel Queue Scheduling



CPU SCHEDULING

Scheduling Algorithms

FIRST-COME, FIRST SERVED:

- (FCFS) same as FIFO
- Simple, fair, but poor performance. Average queueing time may be long.
- What are the average queueing and residence times for this scenario?
- How do average queueing and residence times depend on ordering of these processes in the queue?

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

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)

- 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
  - How do we know what is the length of the next CPU request
  - Could ask the user
    - What if the user lies?

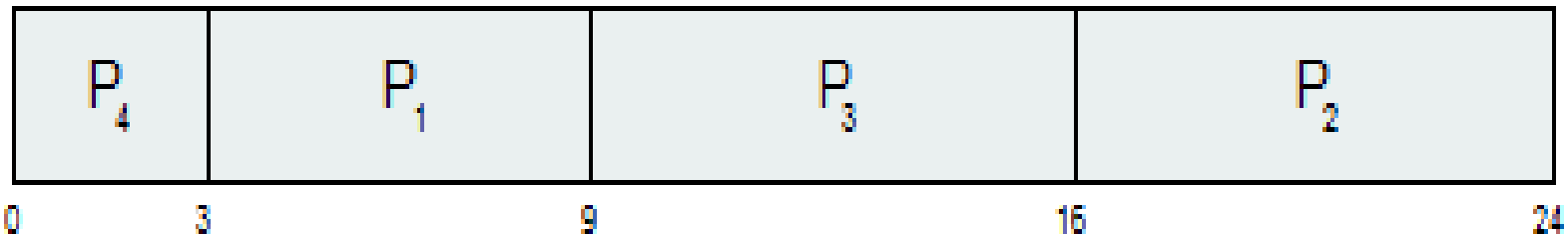
Example of SJF

- Consider the following four processes and their burst time

Assume all four processes arriving at the same time...

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



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

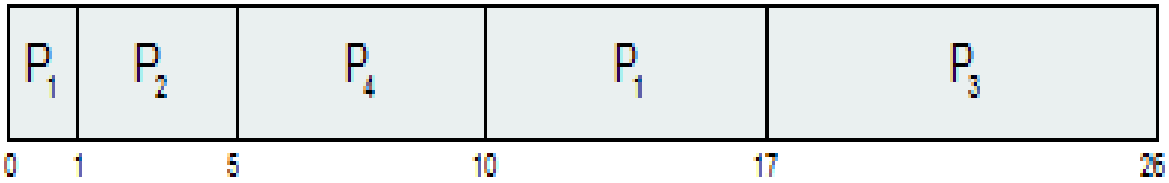
P1, P2, P3, P4

Shortest-remaining-time-first

- Preemptive version of SJF is called shortest-remaining-time-first
- Example illustrating the concepts of varying arrival times and preemption.

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



P3 starts at 17 ms with CPU burst = 9 ms, arrives at 2 ms, i.e., wait time = 17 – 2 = 15 ms

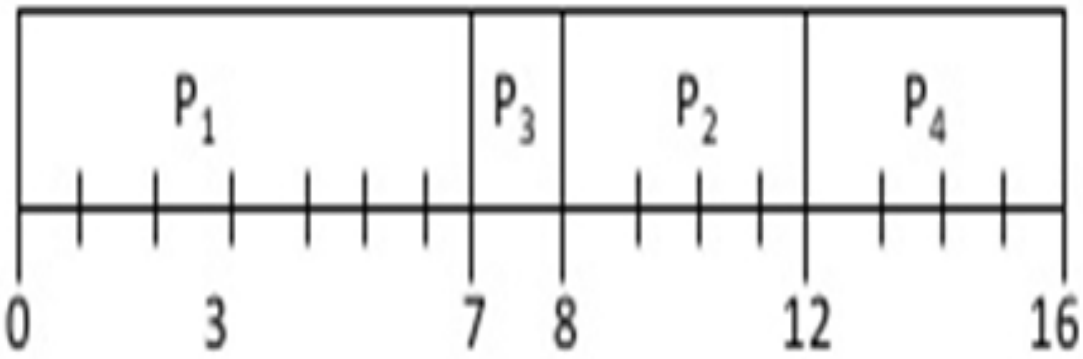
- Average waiting time = [(10-1)+(1-1)+(17-2)+(5-3)]/4 = 26/4 = 6.5 msec

Waiting time for four processes → P1: 10-1; P2: 1-1, P3: 17-2, P4: 5-3; Considering (Job start time & CPU burst – arrival time)

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)

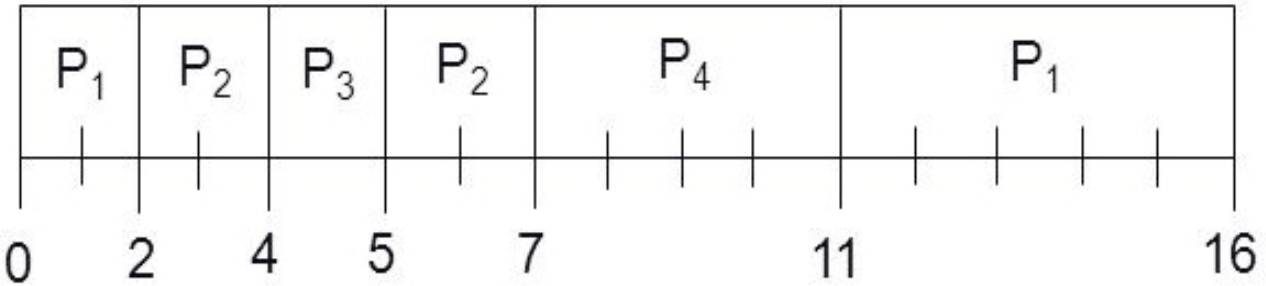


- Average waiting time = (0 + 6 + 3 + 7)/4 ≈ 4

Example of Preemptive SJF (SRTF)

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 = ((11-2) + (5-4) + (4-4) +(7-5))/4 = 12/3 = 3

Round Robin (RR)

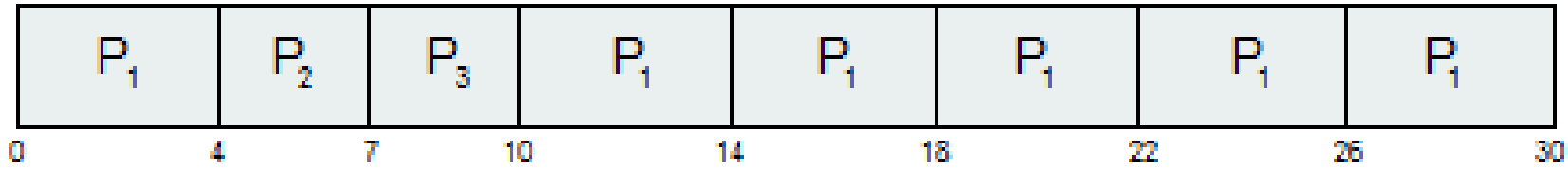
- Each process gets a small unit of CPU time (**time quantum** *q*). After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *N* p*q* is usually 10-100 milliseconds ue and the time quantum is *q*, then each process gets 1/*N* of the CPU time in chunks 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 ⇒ FIFO
  - *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

- Consider the following three processes and their burst time:

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

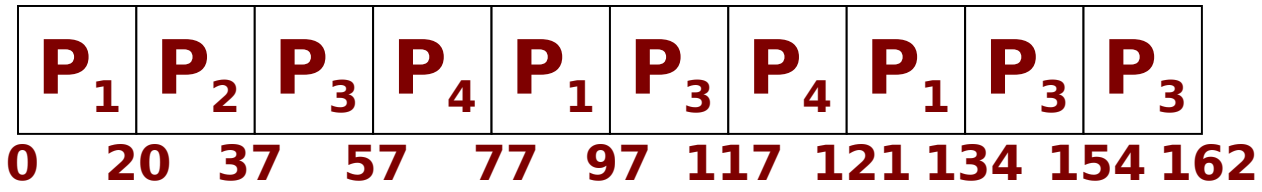


- The average waiting time under the RR policy is often longer
- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q is usually 10ms to 100ms, context switch < 10 usec

Example: RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

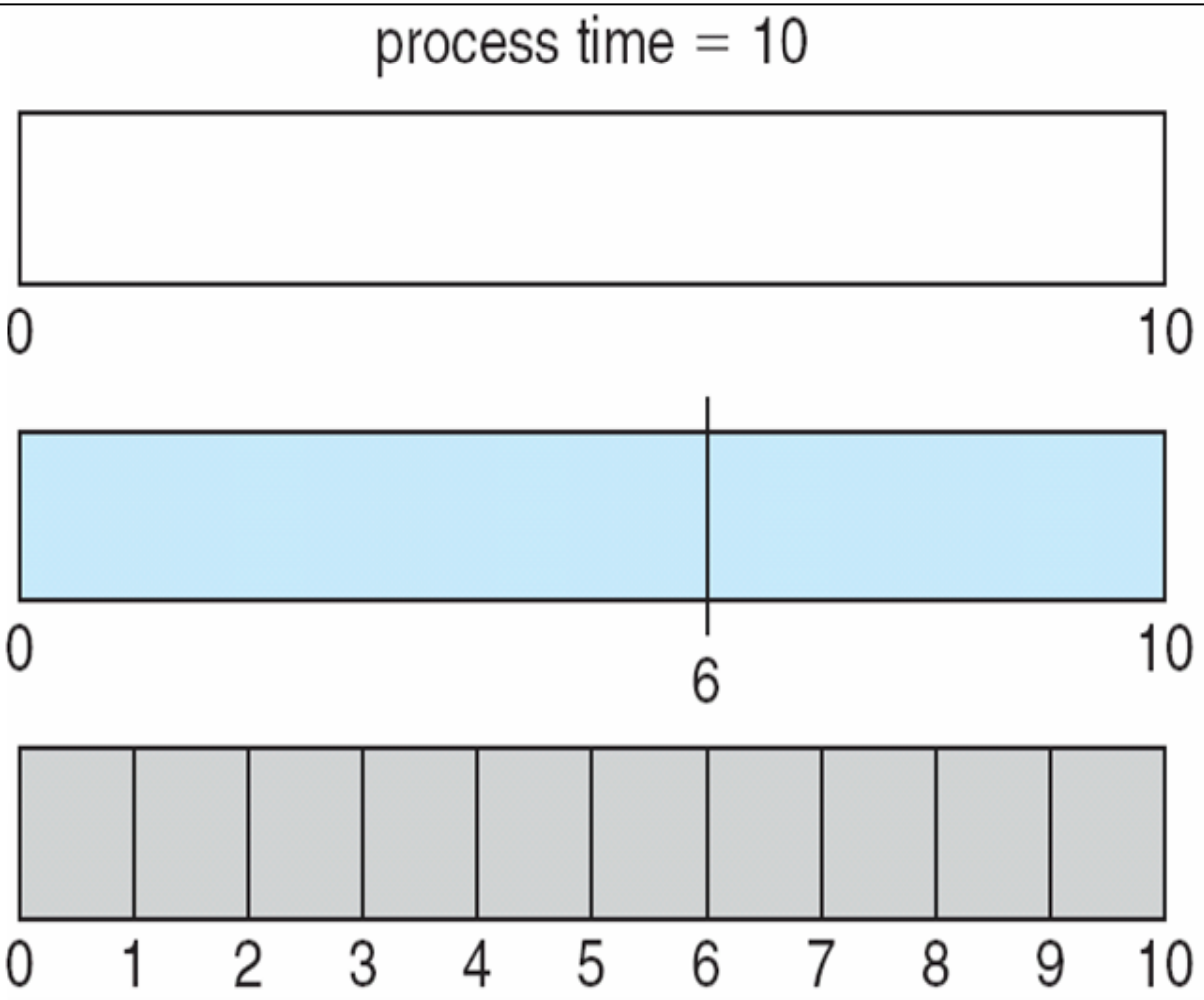
- The Gantt chart is:



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

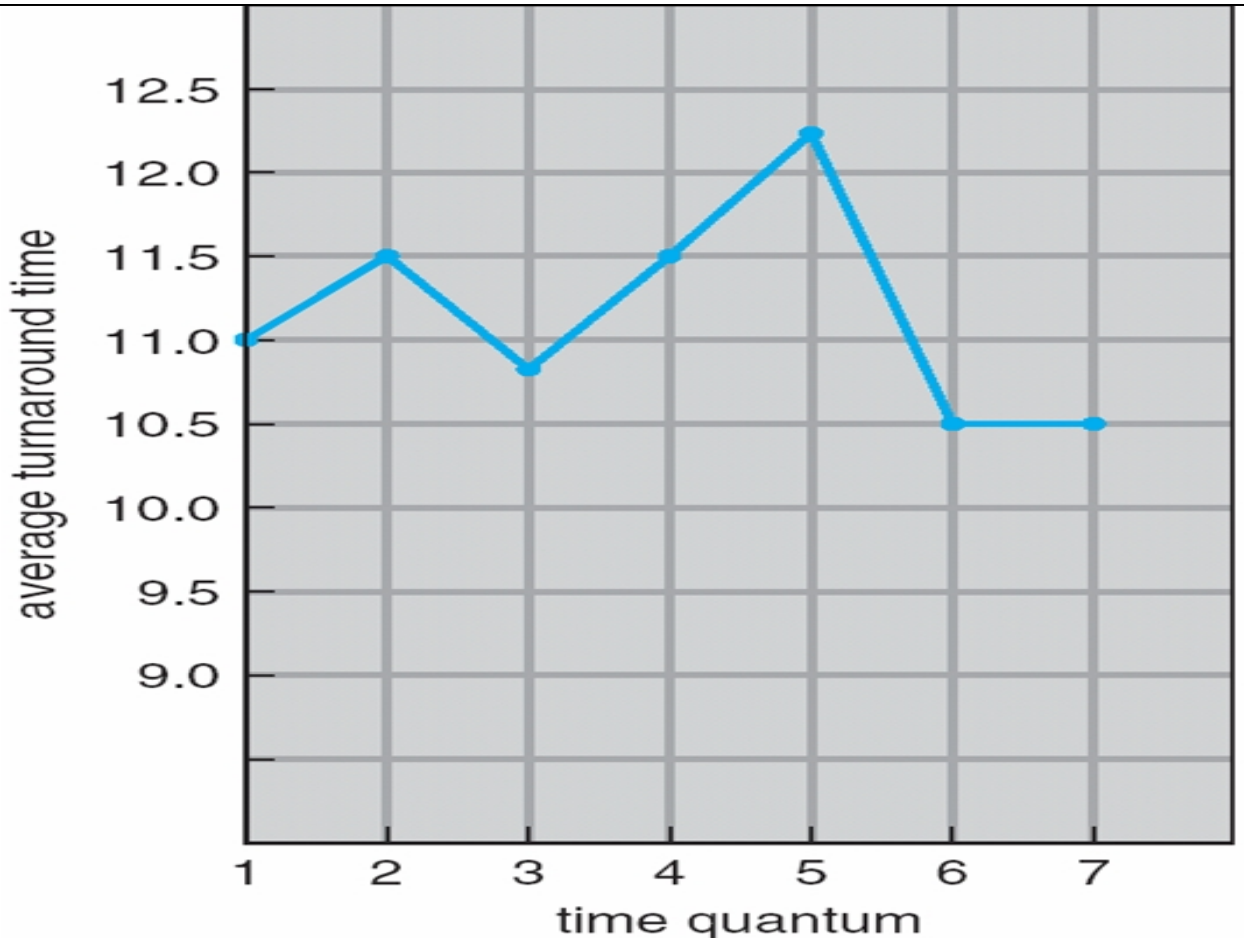
Time Quantum and Context Switch Time

- The performance of the RR algorithm depends on the size of the time quantum. If the time quantum is extremely small (say, 1 millisecond), RR can result in a large number of context switches.



Turnaround Time Varies with the Time Quantum

- The average turnaround time of a set of processes does not necessarily improve as the time-quantum size increases. In general, the average turnaround time can be improved if most processes finish their next CPU burst in a single time quantum.



process	time
P_1	6
P_2	3
P_3	1
P_4	7

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 priority scheduling where priority is the inverse of predicted next CPU burst time***
- **Problem ≡ Starvation – low priority processes may never execute**
- **Solution ≡ Aging – as time progresses increase the priority of the process**

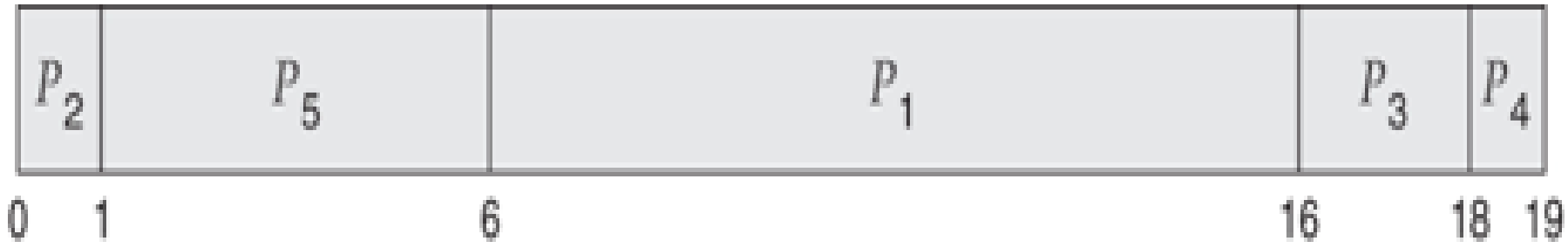
Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Wait Time (WT)
P2: 0
P5: 1
P1: 6
P3: 16
P4: 18

Average Wait Time (AWT)
(P2 + P5 + P1 + P3 + P4)/5 = 8.2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

Combining Priority Scheduling and RR

- System executes the highest priority process; processes with the same priority will be run using round-robin.
- Consider the following five processes and their burst time

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

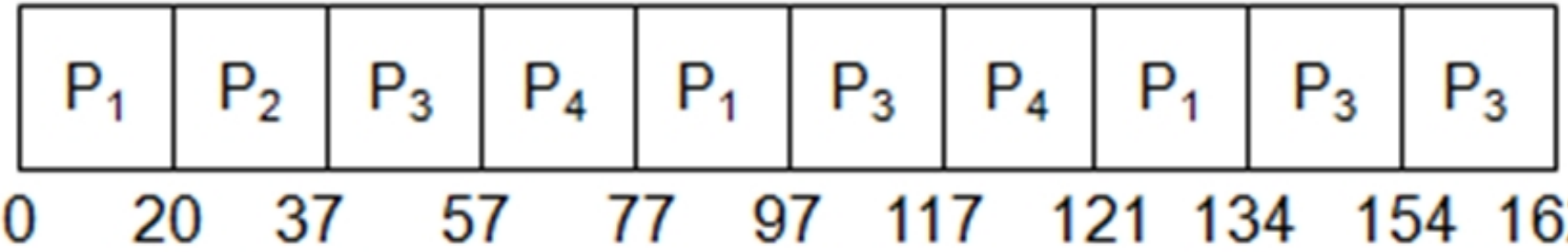
P4: No RR; Applying RR to rest, time quantum = 2-4 ms.
Based on the Gantt chart, WT:
P4: 0; P2: 7+2+2=11;
P3:9+2+1=12; P1: 20+2=22;
P5: 22+2=24;
AWT = (P4+P2+P3+P1+P5)/5
= 13.8 ms

- Priority scheduling Gantt Chart

Which scheduling the system is following?

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*

- **Turnaround time** – amount of time to execute a particular process (**Min turnaround time**); Sum of wait for memory, ready queue, execution, and I/O.
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output

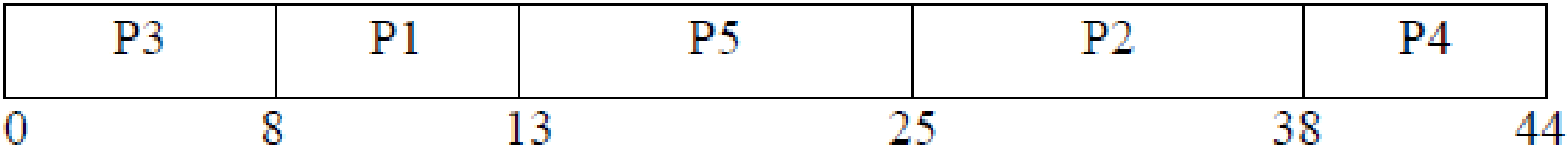
Indefinite blocking (or starvation)

- 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

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

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

Static Priority Scheduling

- Associate an (integer) priority with each process
- For example:

Priority	Type	Priority	Type
0	system internal processes	**2**	interactive processes (students)
1	interactive processes (staff)	**3**	batch processes.

- Then allocate CPU to the highest priority process:
  - 'highest priority' typically means smallest integer
  - get preemptive and non-preemptive variants.
- e.g. SJF is priority scheduling where priority is the predicted next CPU burst time.
- **Problem**: how to resolve ties?
  - round robin with time-slicing
  - allocate quantum to each process in turn.
  - Problem: biased towards CPU intensive jobs.
    - per-process quantum based on usage?
    - ignore?
- **Problem**: starvation. . .

Dynamic Priority Scheduling

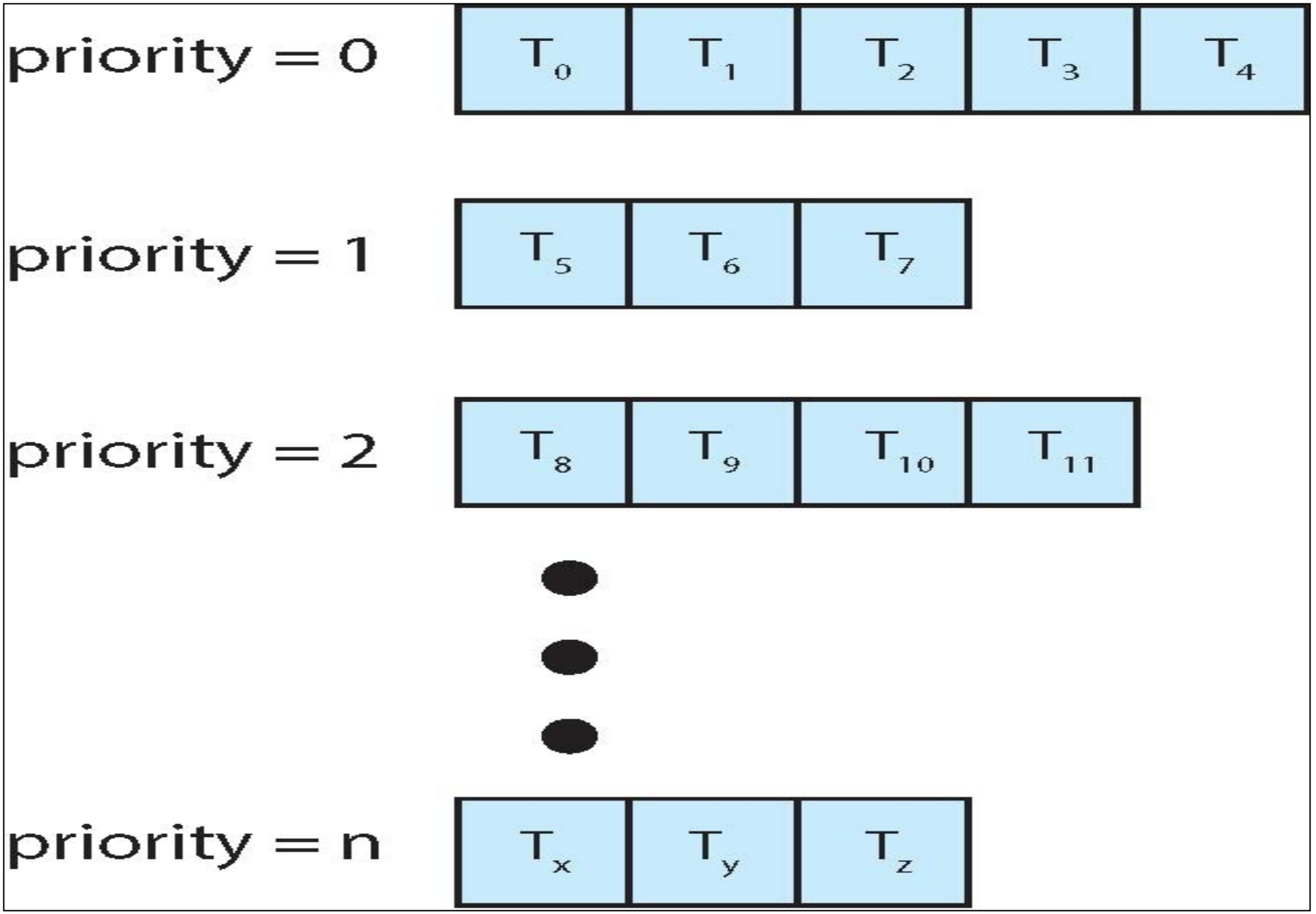
- Use same scheduling algorithm, but allow priorities to change over time.
- e.g. simple aging:
  - processes have a (static) *base priority* and a dynamic *effective priority*.
  - if process starved for k seconds, increment effective priority.
  - once process runs, reset effective priority.
- e.g. computed priority:
  - first used in Dijkstra's THE
  - time slots: $\ldots, t, t+1, \ldots$
  - in each time slot t, measure the CPU usage of process j: u^j
  - priority for process j in slot $t+1$:
    $p^j_{t+1} = f(u^j_t, p^j_t, u^j_{t-1}, p^j_{t-1}, \ldots)$
  - e.g. $p^j_{t+1} = p^j_t/2 + ku^j_t$
  - penalises CPU bound $\rightarrow$ supports I/O bound.
- today such computation considered acceptable. . .


START

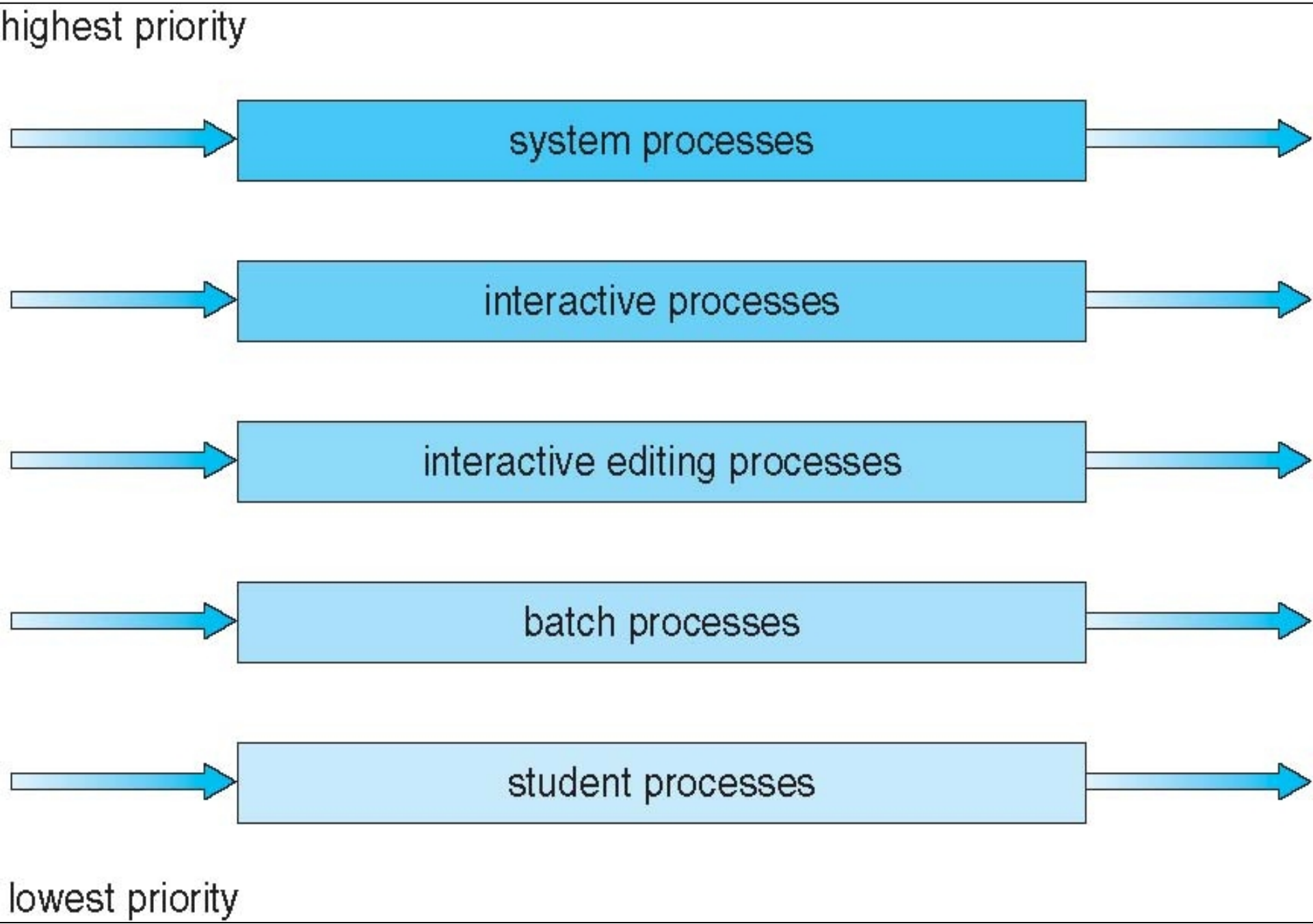
Multilevel Queue

- 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

Separate Queue For Each Priority



Multilevel Queue Scheduling

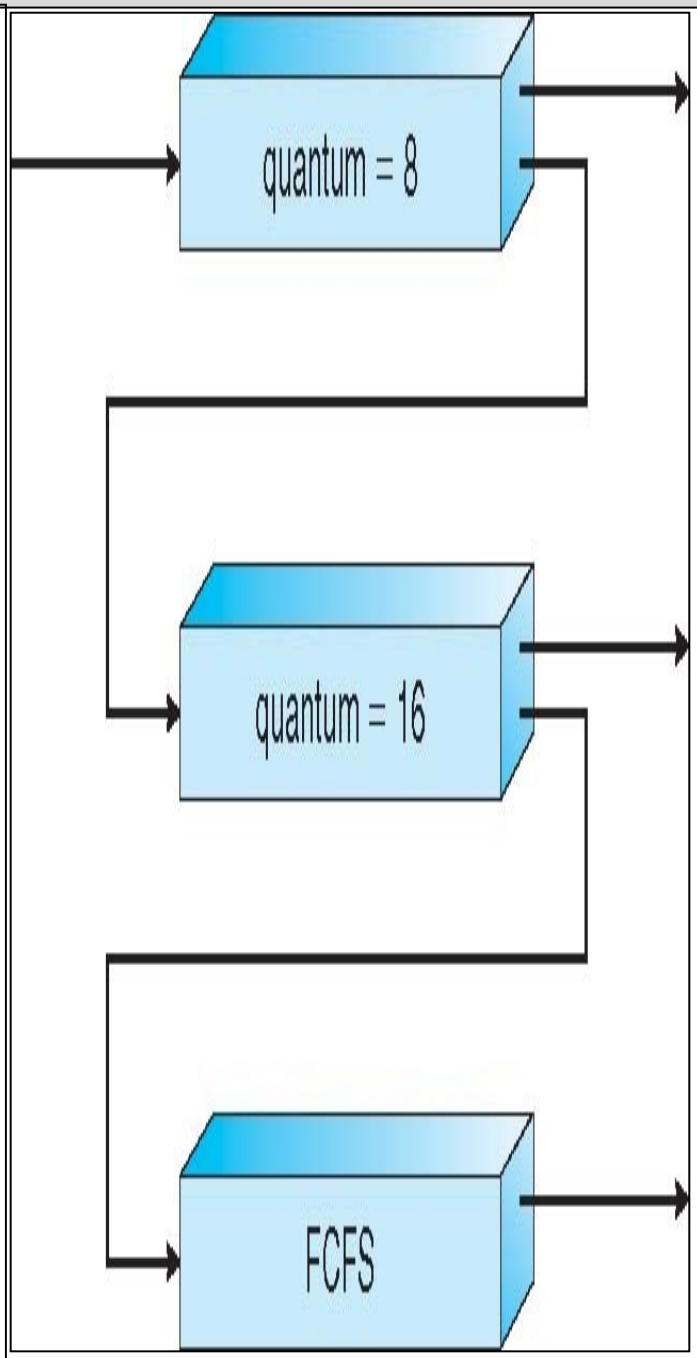


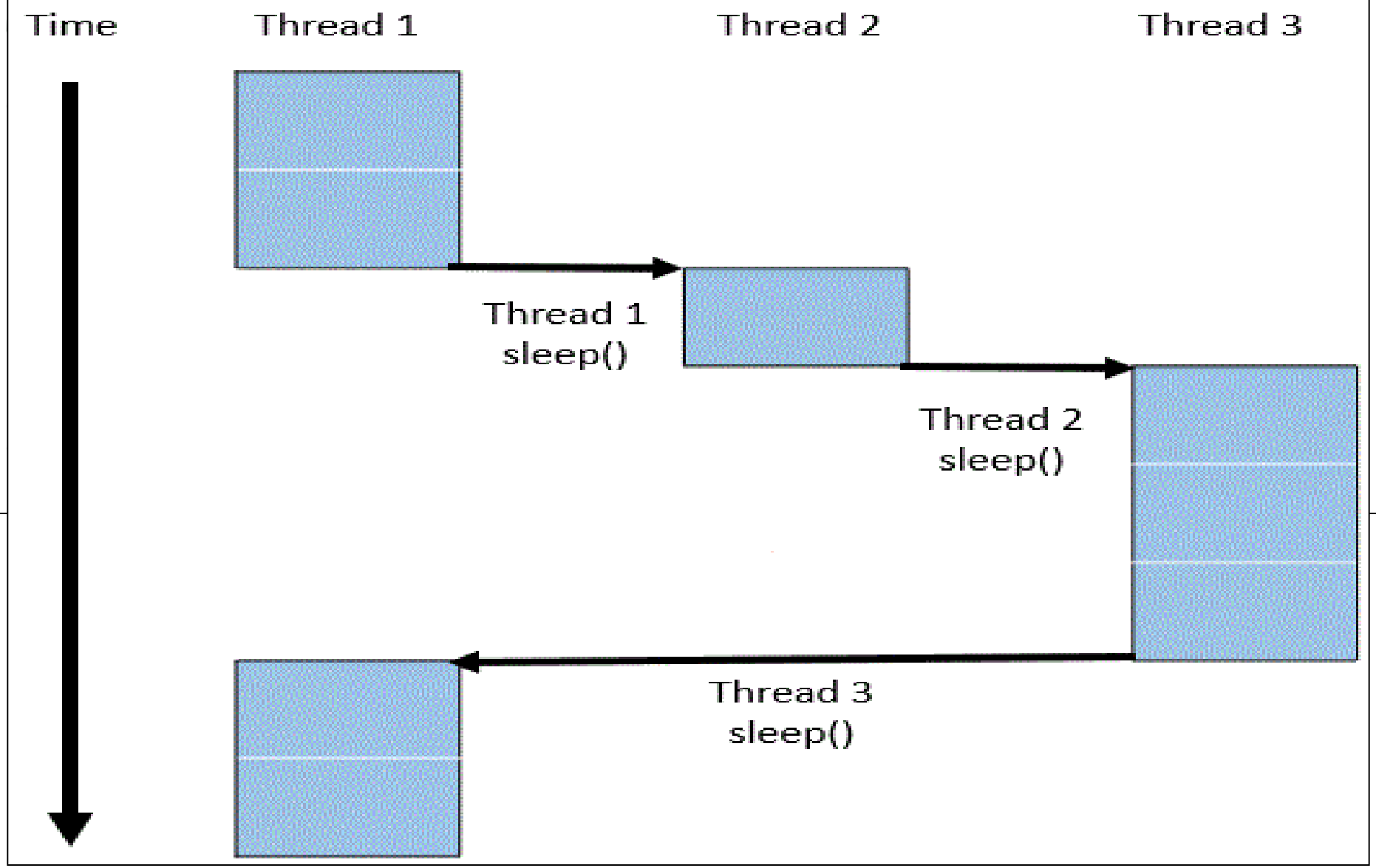
Multilevel Feedback Queue

- 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





Thread scheduling

Questions…

Which of the following process scheduling algorithm may lead to starvation

(A) FIFO

(B) Round Robin

(C) Shortest Job Next

(D) None of the above

Answer: (C)

Explanation: Shortest job next may lead to process starvation for processes which will require a long time to complete if short processes are continually added.

Consider three CPU-intensive processes, which require 10, 20 and 30 time units and arrive at times 0, 2 and 6, respectively. How many context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end.

(A) 1

(B) 2

(C) 3

(D) 4

Answer: (B)

Explanation: Let three process be P0, P1 and P2 with arrival times 0, 2 and 6 respectively and CPU burst times 10, 20 and 30 respectively. At time 0, P0 is the only available process so it runs. At time 2, P1 arrives, but P0 has the shortest remaining time, so it continues. At time 6, P2 arrives, but P0 has the shortest remaining time, so it continues. At time 10, P1 is scheduled as it is the shortest remaining time process. At time 30, P2 is scheduled. Only two context switches are needed. P0 to P1 and P1 to P2.

Questions…

Which module gives control of the CPU to the process selected by the short-term scheduler?

A. dispatcher
B. interrupt
C. scheduler
D. none of the mentioned

Answer: Option A

The interval from the time of submission of a process to the time of completion is termed as:

A. waiting time
B. turnaround time
C. response time
D. throughput

Answer: Option B

In priority scheduling algorithm, when a process arrives at the ready queue, its priority is compared with the priority of:

A. all process
B. currently running process
C. parent process
D. init process

Answer: Option B

Processes are classified into different groups in:

A. shortest job scheduling algorithm
B. round robin scheduling algorithm
C. priority scheduling algorithm
D. multilevel queue scheduling algorithm

Answer: Option D

Questions…

Which one of the following can not be scheduled by the kernel?

A. kernel level thread **B.** user level thread
C. process **D.** none of the mentioned

Answer: Option B

The two steps of a process execution are : (choose two)

A. I/O Burst, CPU Burst **B.** CPU Burst
C. Memory Burst **D.** OS Burst

Answer: Option A

An I/O bound program will typically have :

Answer: Option C

A. a few very short CPU bursts **B.** many very short I/O bursts
C. many very short CPU bursts **D.** a few very short I/O bursts

The switching of the CPU from one process or thread to another is called :**A.** process switch **B.** task switch
C. context switch **D.** All of these

Answer: Option D

Waiting time is :

A. the total time in the blocked and waiting queues
B. the total time spent in the ready queue
C. the total time spent in the running queue
D. the total time from the completion till the submission of a process

Answer: Option B

Questions…

Response time is :
A. the total time taken from the submission time till the completion time
B. the total time taken from the submission time till the first response is produced
C. the total time taken from submission time till the response is output
D. None of these

Answer: Option B

The portion of the process scheduler in an operating system that dispatches processes is concerned with :
A. assigning ready processes to CPU
B. assigning ready processes to waiting queue
C. assigning running processes to blocked queue
D. All of these

Answer: Option A

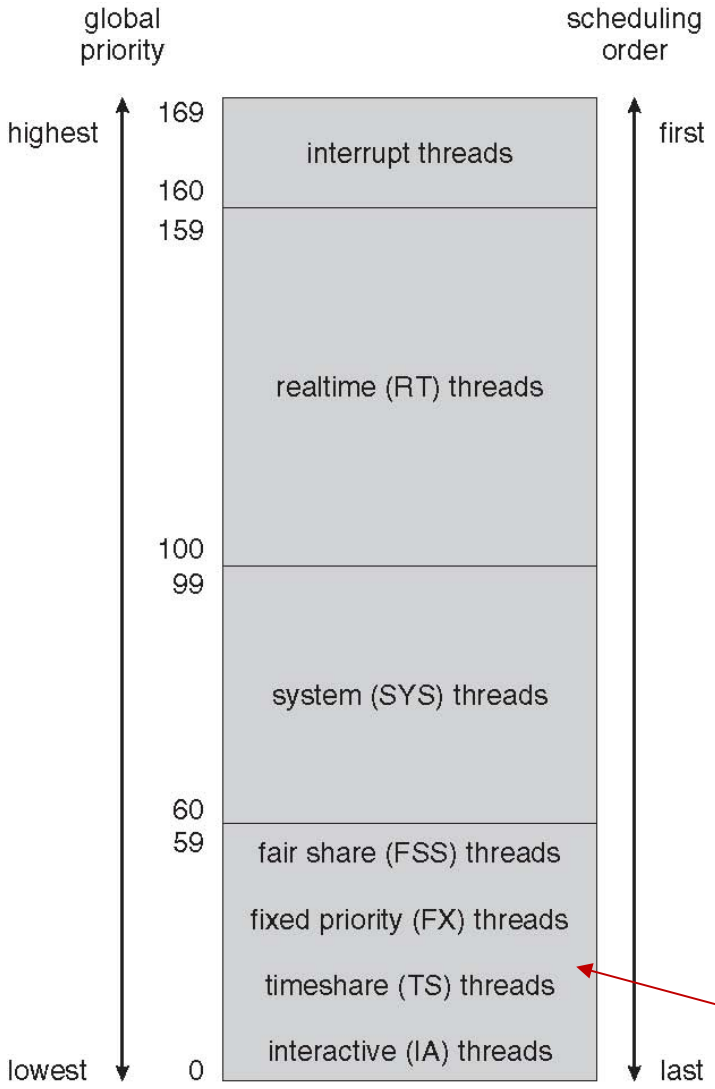
Under multiprogramming, turnaround time for short jobs is usually _____ and that for long jobs is slightly _____.
A. Lengthened; Shortened
B. Shortened; Lengthened
C. Shortened; Shortened
D. Shortened; Unchanged

Answer: Option B

Operating System Examples

- Solaris scheduling
- Windows XP scheduling
- Linux scheduling

Solaris Scheduling



priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

Dispatch Table

Multilevel feedback queue

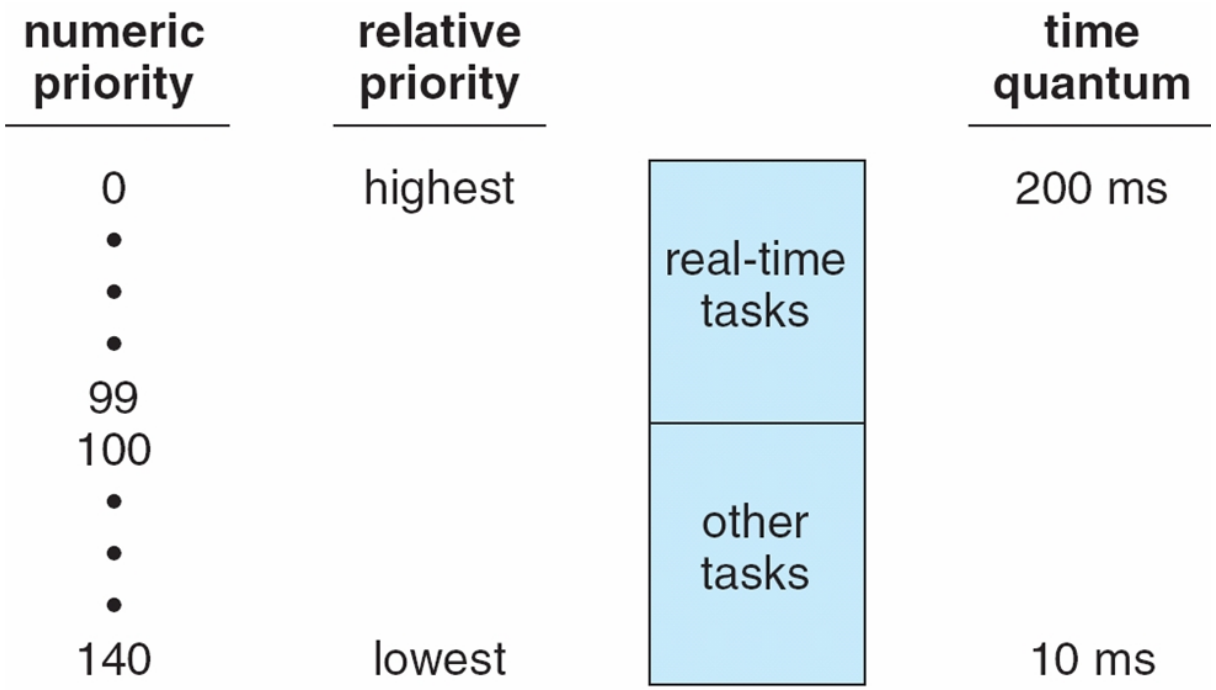
Windows XP Priorities

- Priority based
  - preemptive

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Linux Scheduling

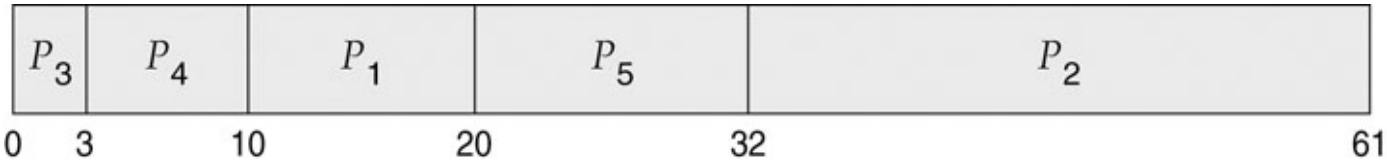
- Constant order *O*(1) scheduling time
- Priority based
  - Preemptive



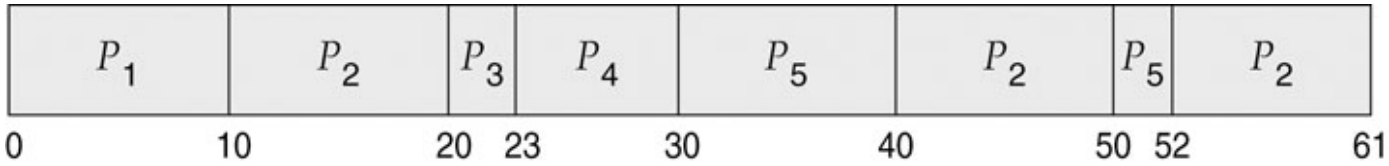
Algorithm Evaluation

- **Analytical modeling**
  - Several projections become possible by just formulas
- **Deterministic modeling**
  - takes a particular predetermined workload and defines the performance of each algorithm for that workload
- FCFS

  - Average wait = 28
- SJF



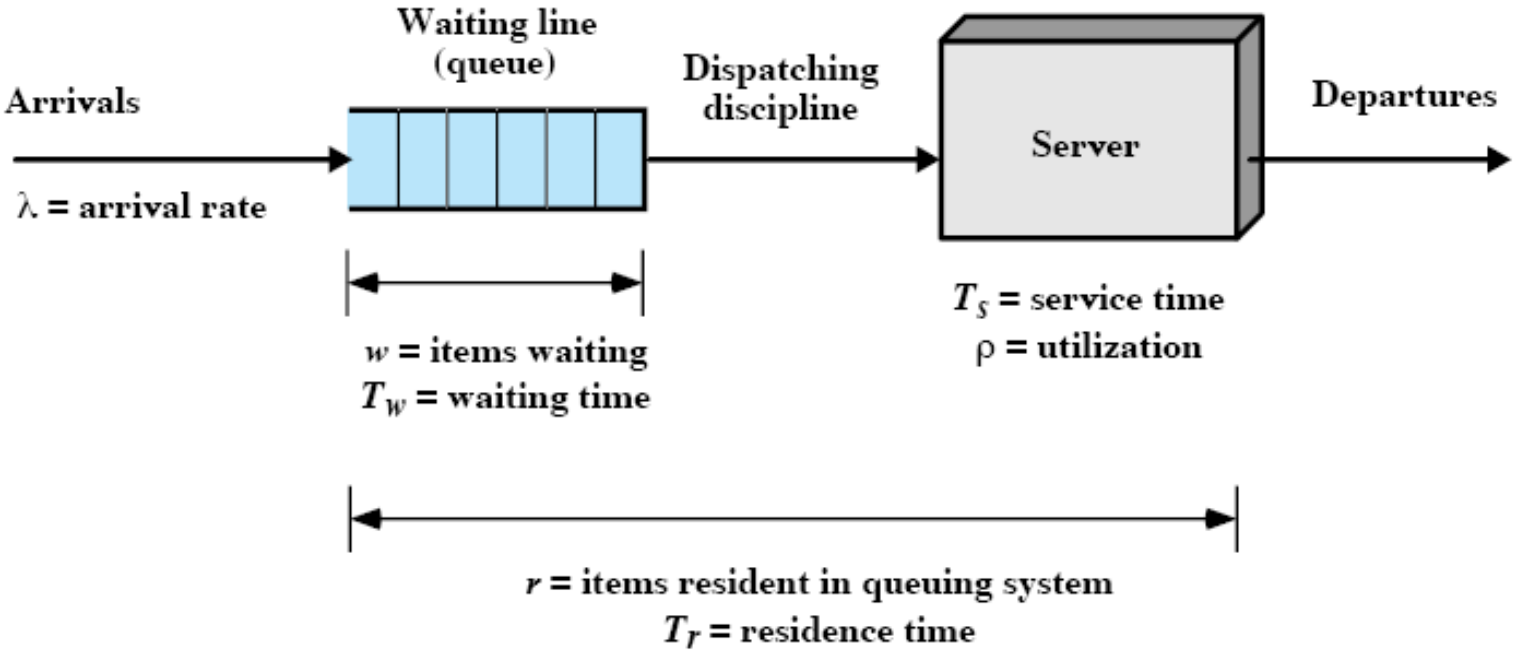
  - Average wait = 13
- RR



  - Average wait = 23

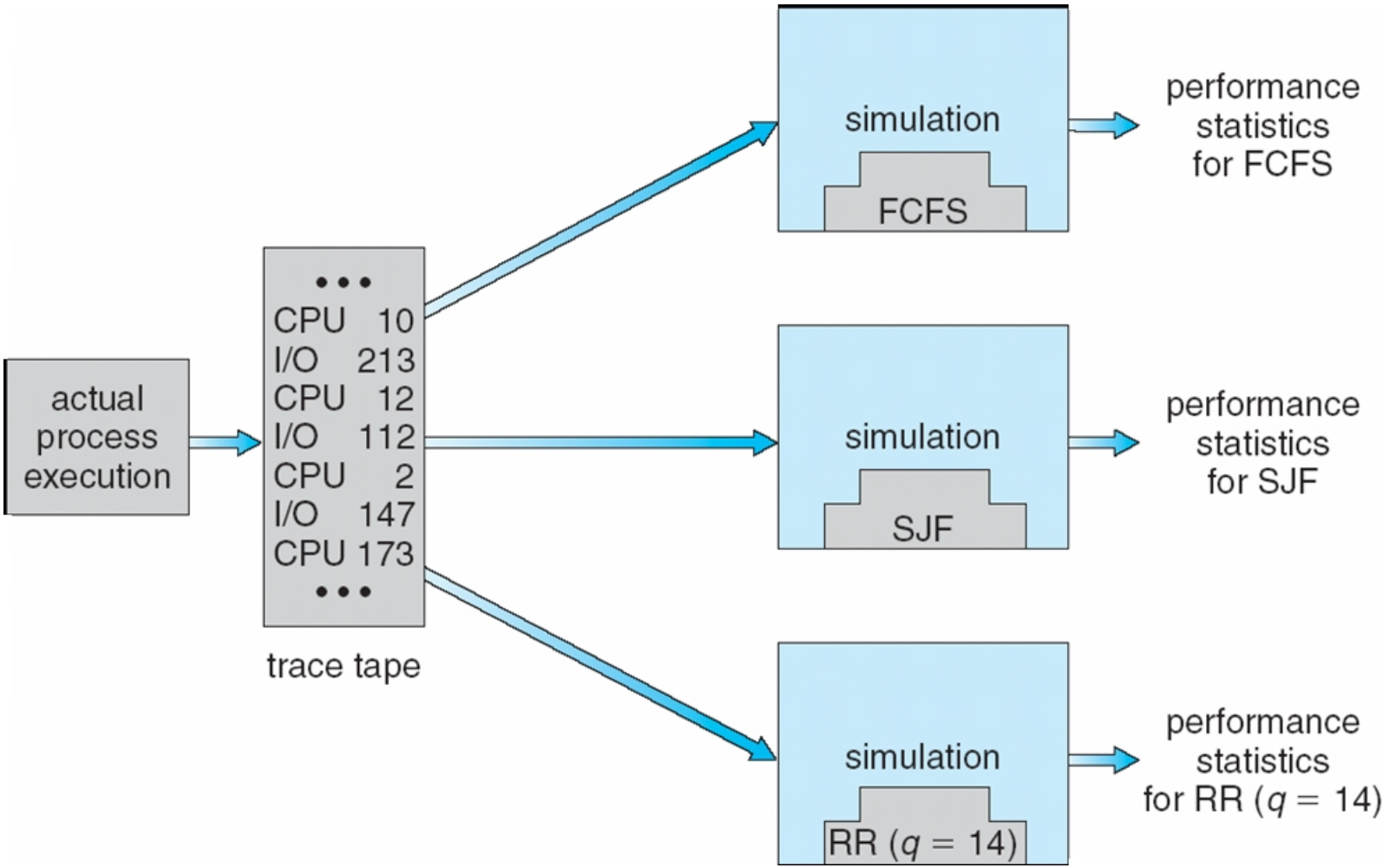
Algorithm Evaluation

- **Queuing models**
  - Distribution of CPU and I/O bursts
  - Distribution of arrival times
- Little's formula: n = λ x W
  - In steady-state, number of departures must be equal to the number of arrivals



Algorithm Evaluation

- **Simulation**



Scheduling algorithm goals

All systems

- Fairness - giving each process a fair share of the CPU
- Policy enforcement - seeing that stated policy is carried out
- Balance - keeping all parts of the system busy

Batch systems

- Throughput - maximize jobs per hour
- Turnaround time - minimize time between submission and termination
- CPU utilization - keep the CPU busy all the time

Interactive systems

- Response time - respond to requests quickly
- Proportionality - meet users' expectations

Real-time systems

- Meeting deadlines - avoid losing data
- Predictability - avoid quality degradation in multimedia systems

the end of one chapter
is just
the beginning of another