Scheduling for uniprocessor systems
Introduction

Lecturer:
William Fornaciari
Politecnico di Milano
fornacia@elet.polimi.it
www.elet.polimi.it/~fornacia

SUMMARY

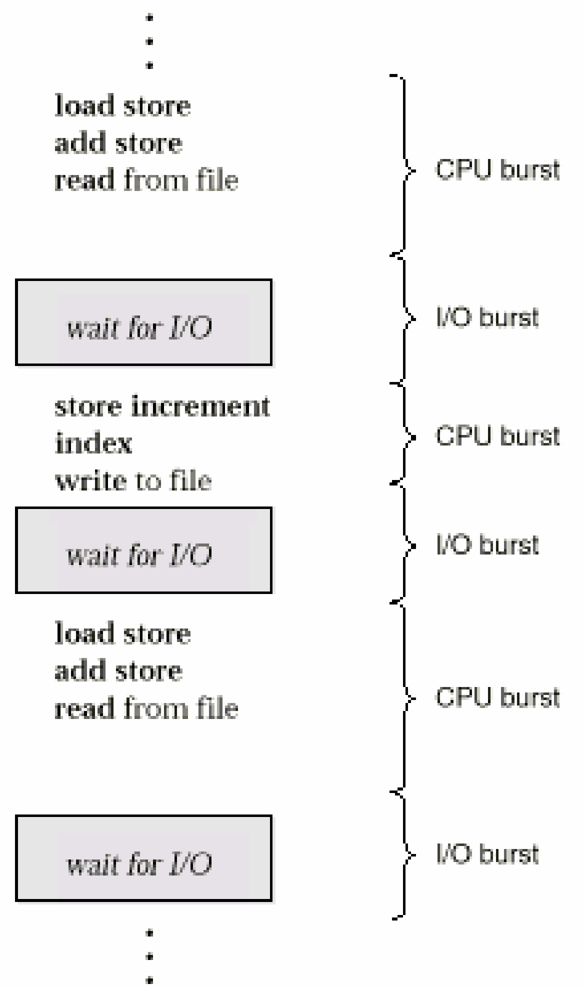
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms

...more to come

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



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 *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
  - 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.

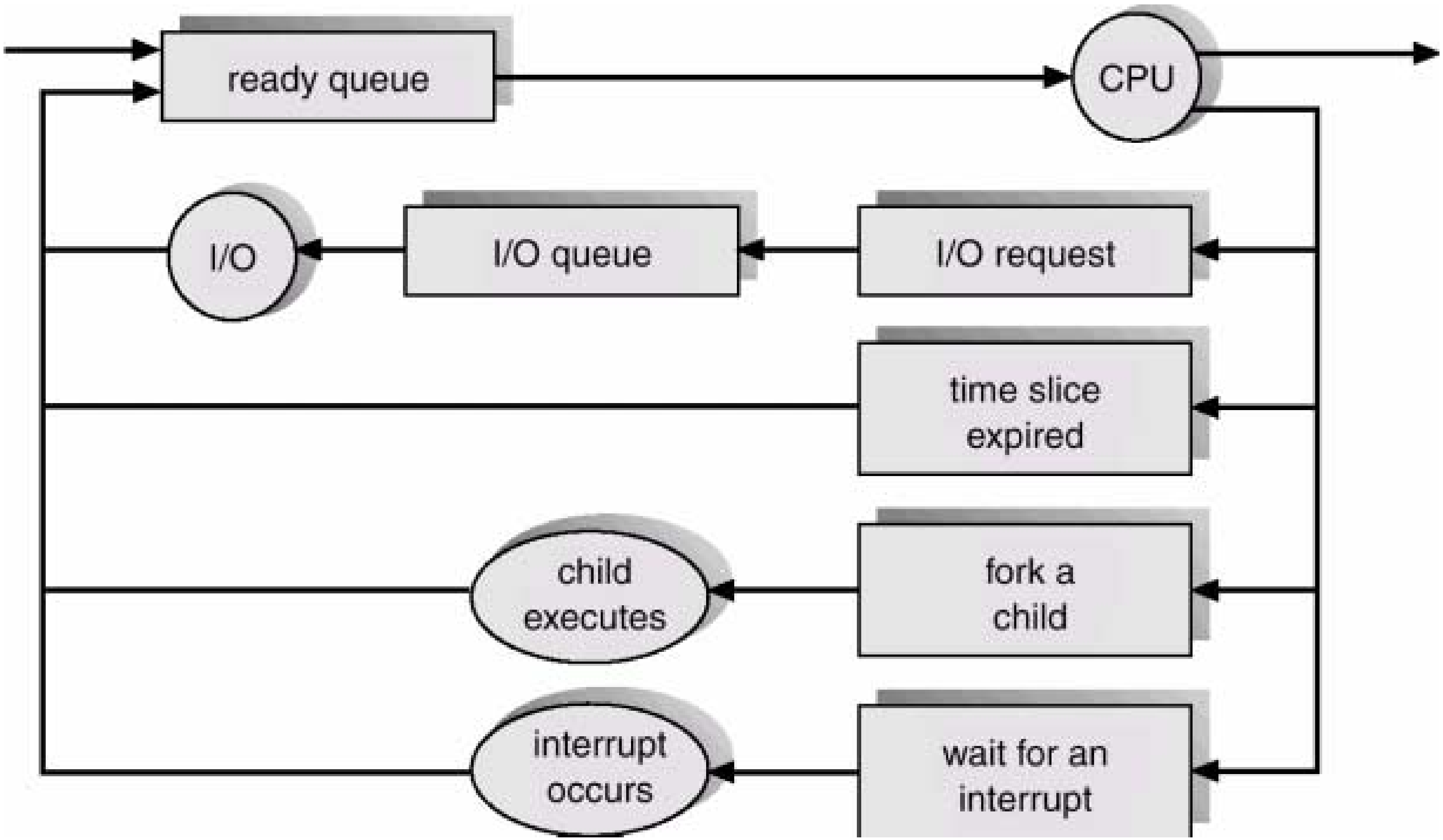
Scheduling Criteria

- 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 (for time-sharing environment)

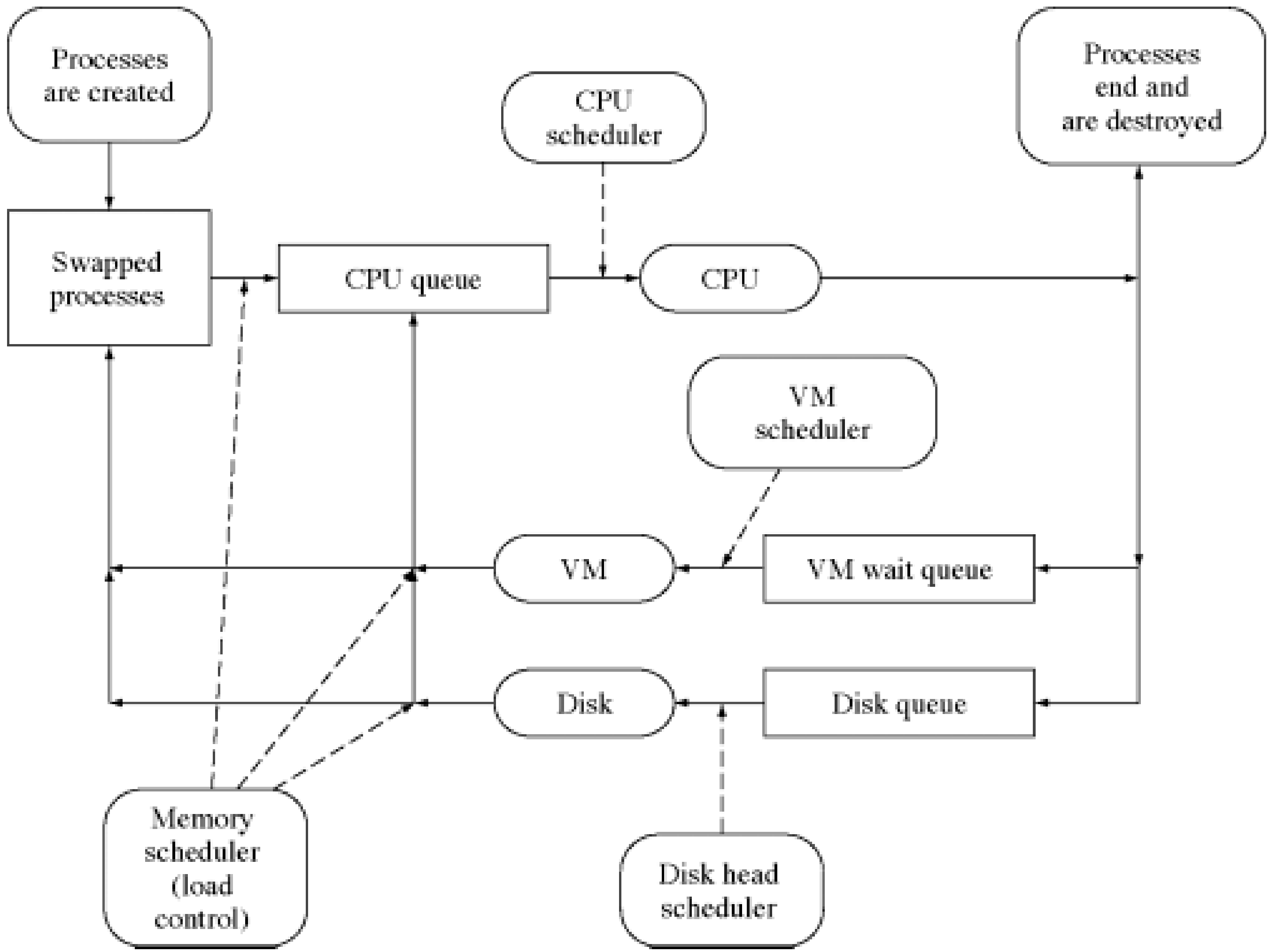
Optimization Criteria

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

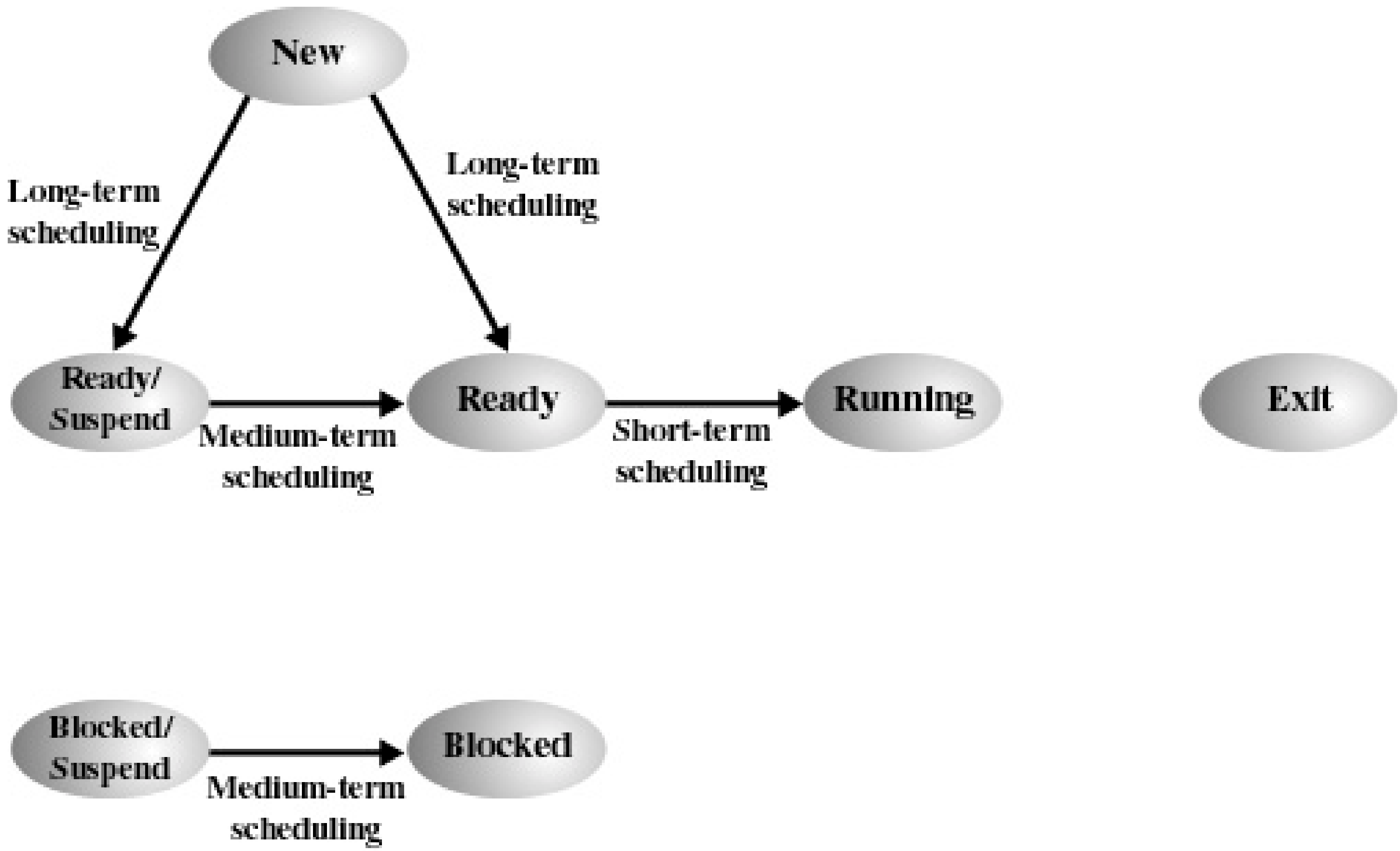
Process Scheduling: Examples



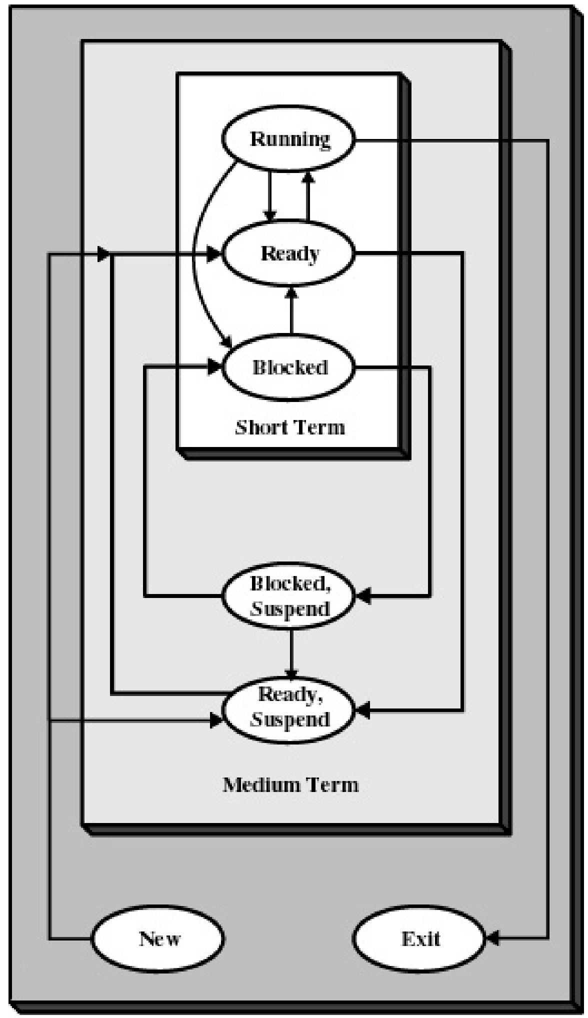
OS Schedulers



Scheduling and Process State Transition



Levels of Scheduling



Types of Scheduling

- Long-term
  - The decision to add to the pool of processes to be executed
- Medium-term
  - The decision to add to the number of processes that are partially or fully in main memory
- Short-term
  - The decision as to which available process will be executed by the processor
- I/O Scheduling
  - The decision as to which process's pending I/O request shall be handled by an available I/O device

Long-Term Scheduling

- Determines which programs are admitted to the system for processing
- Controls the degree of multiprogramming
- More processes, smaller percentage of time each process is executed

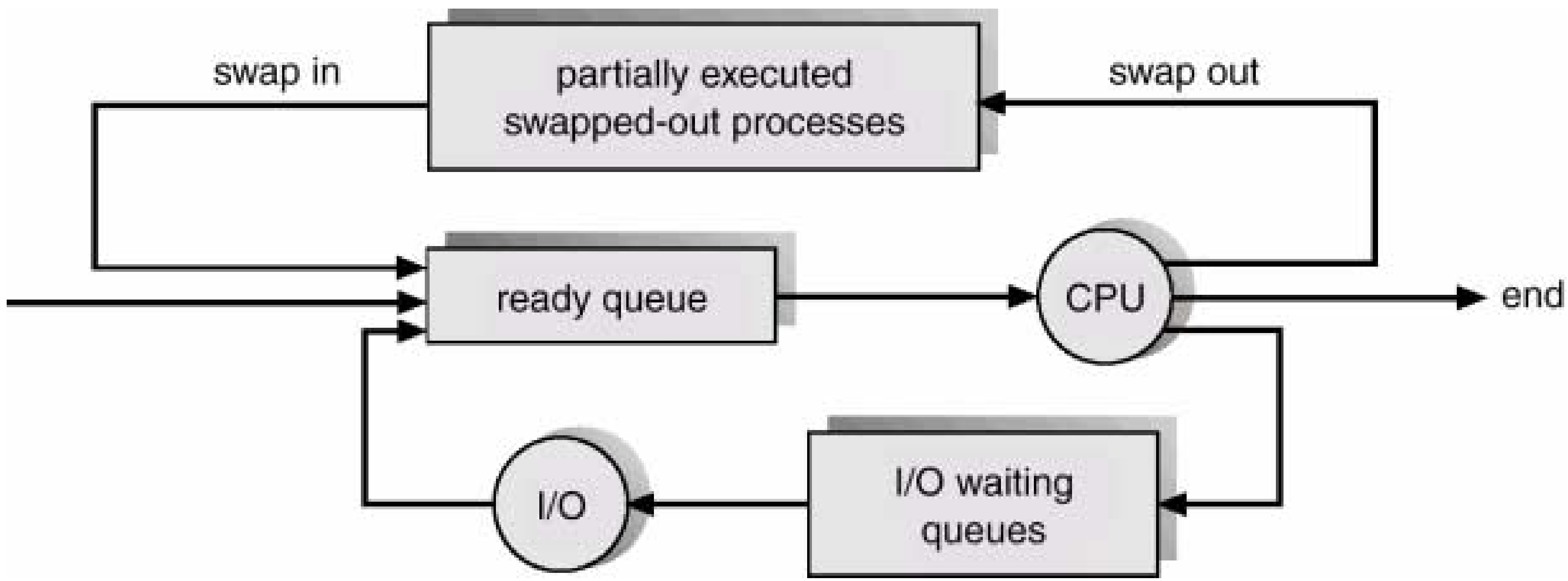
Medium-Term Scheduling

- Part of the swapping function
- Based on the need to manage the degree of multiprogramming

Short-Term Scheduling

- Known as the dispatcher
- Executes most frequently
- Invoked when an event occurs
  - Clock interrupts
  - I/O interrupts
  - Operating system calls
  - Signals

Addition of Medium Term Scheduling



Short-Term Scheduling Criteria

- User-oriented
  - Response Time
    - Elapsed time between the submission of a request until there is output
- System-oriented
  - Effective and efficient utilization of the processor

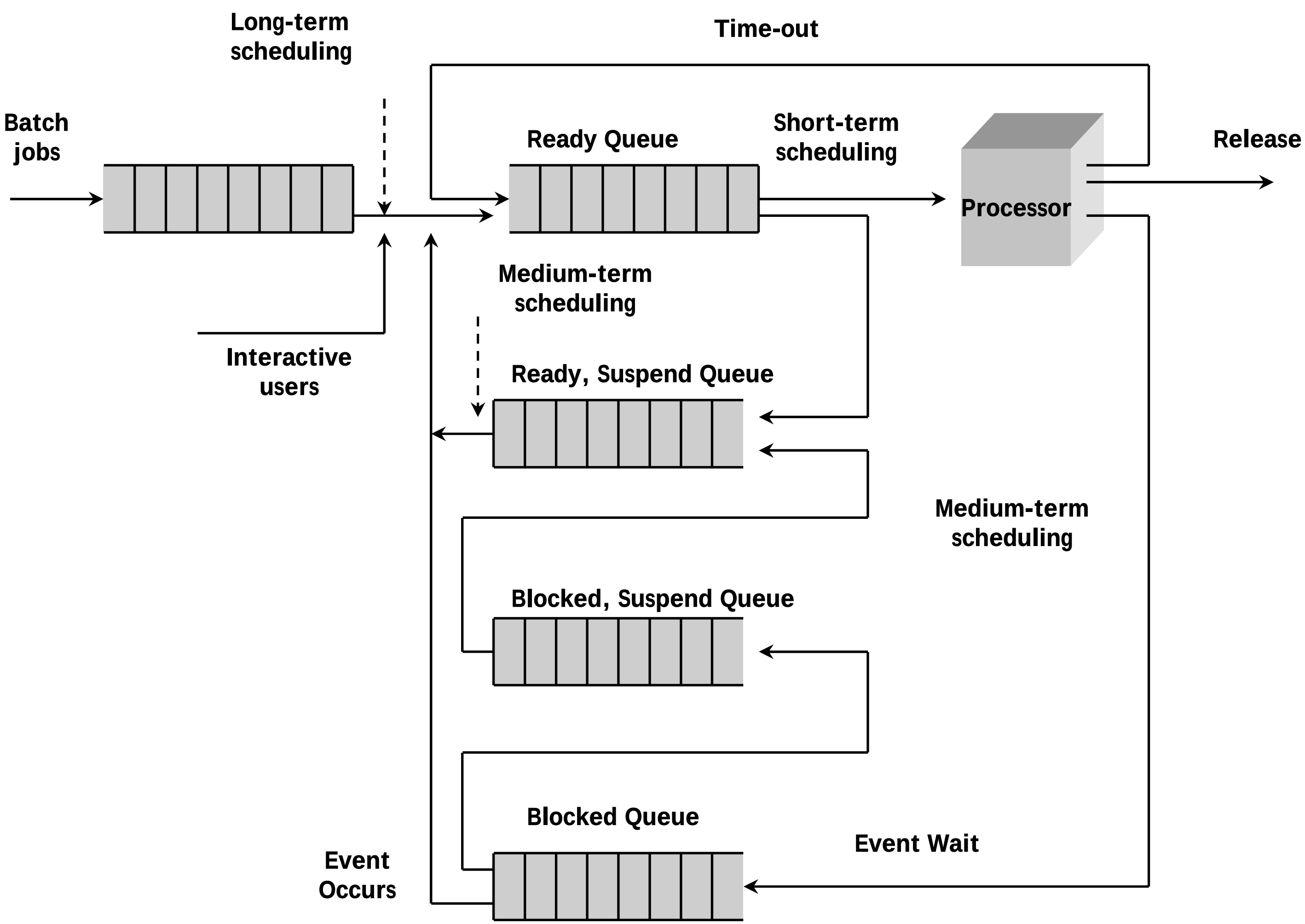
Short-Term Scheduling Criteria

- Performance-related
  - Quantitative
  - Measurable such as response time and throughput
- Not performance related
  - Qualitative
  - Predictability

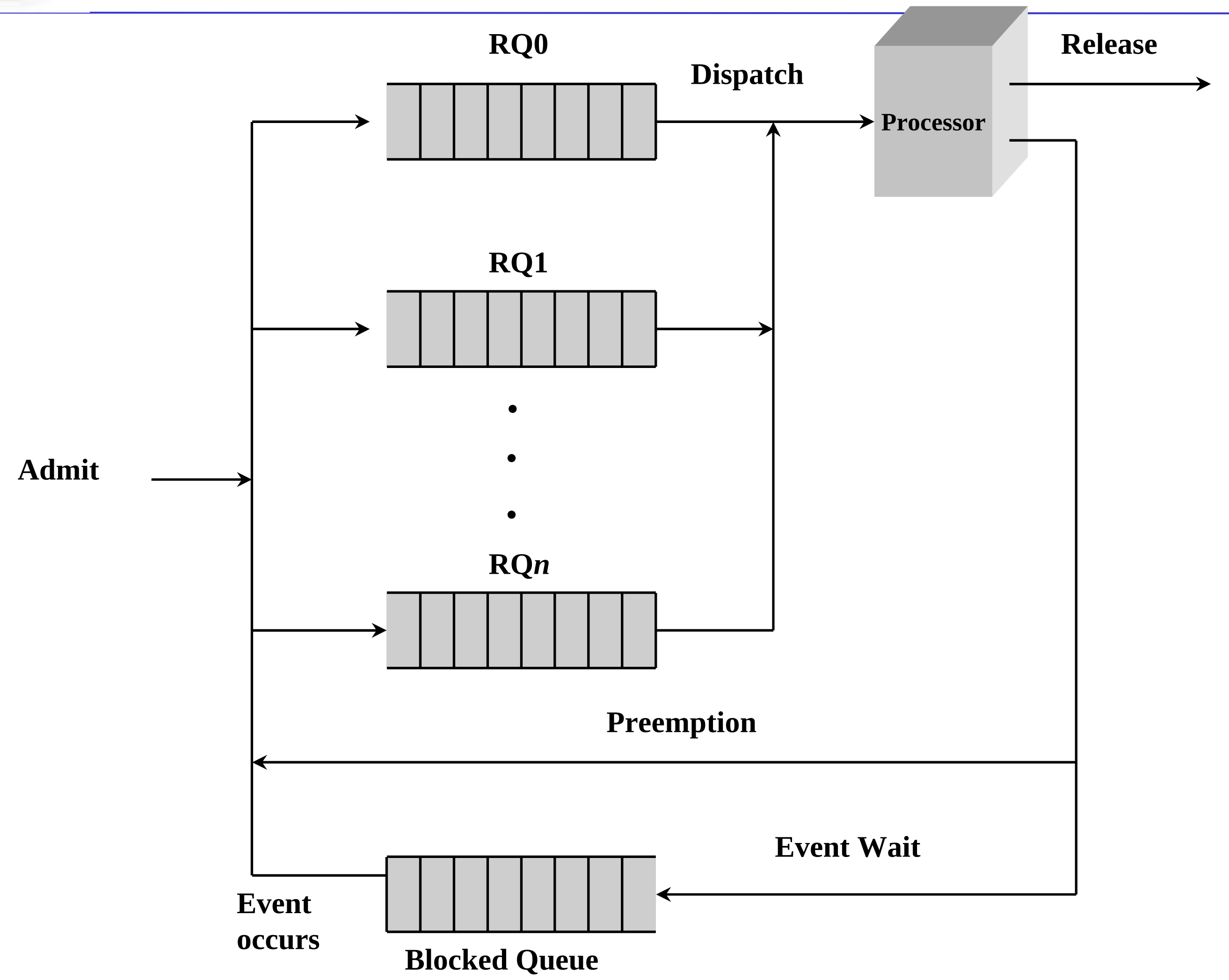
Priorities

- Scheduler will always choose a process of higher priority over one of lower priority
- Have multiple ready queues to represent each level of priority
- Lower-priority may suffer starvation
  - allow a process to change its priority based on its age or execution history

Queuing Diagram for Scheduling



Priority Queuing



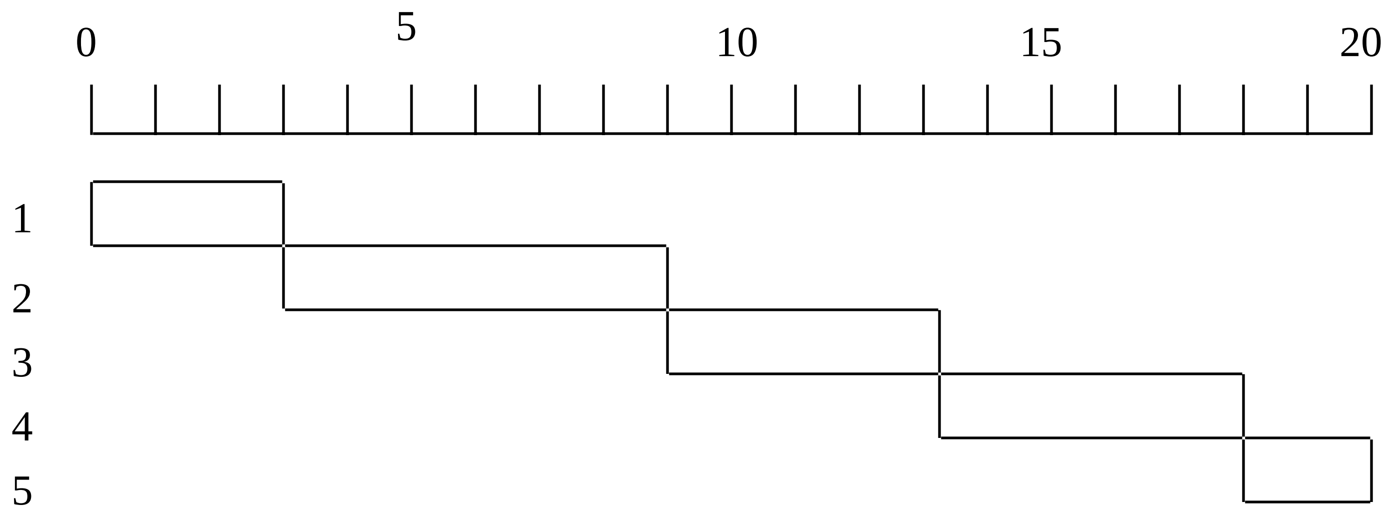
Decision Mode

- Nonpreemptive
  - Once a process is in the running state, it will continue until it terminates or blocks itself for I/O
- Preemptive
  - Currently running process may be interrupted and moved to the Ready state by the operating system
  - Allows for better service since any one process cannot monopolize the processor for very long

Process Scheduling: Example

Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

First-Come, First-Served (FCFS) Scheduling



- Each process joins the Ready queue
- When the current process ceases to execute, the oldest process in the Ready queue is selected

First-Come, First-Served (FCFS) Scheduling

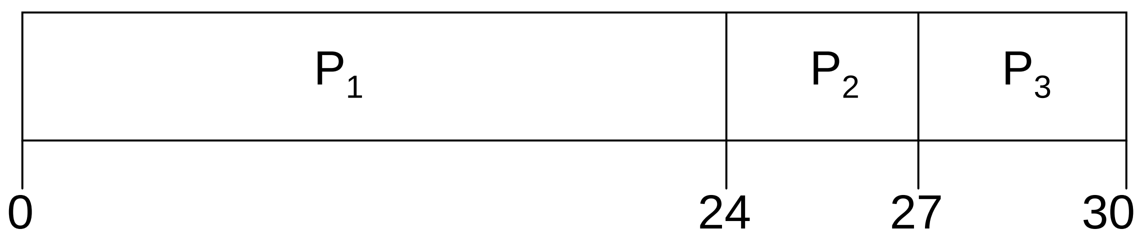
- A short process may have to wait a very long time before it can execute
- Favours CPU-bound processes
  - I/O processes have to wait until CPU-bound process completes

First-Come, First-Served (FCFS) Scheduling

- Example:

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:



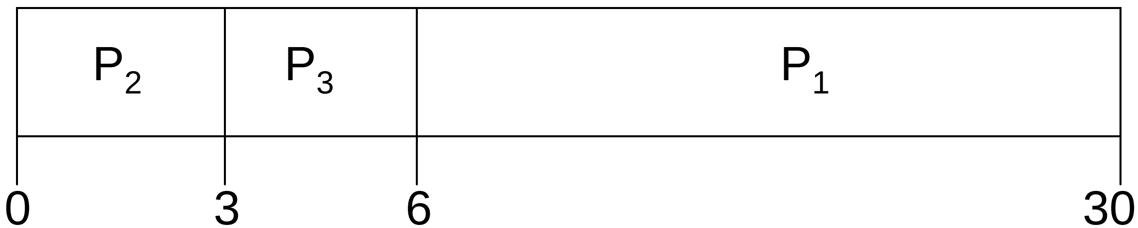
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

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.
- *Convoy effect* short process behind long process

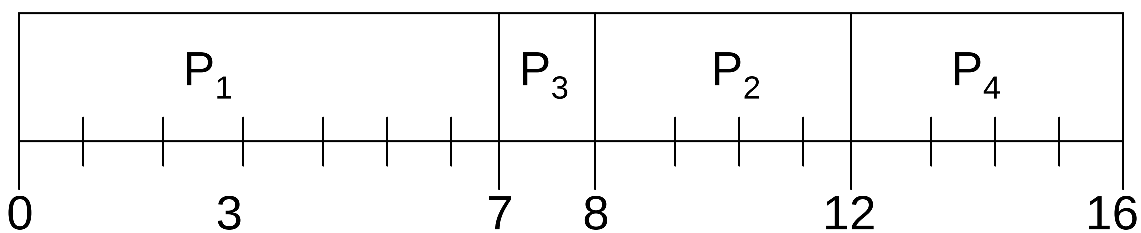
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - nonpreemptive – 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

- SJF (non-preemptive)

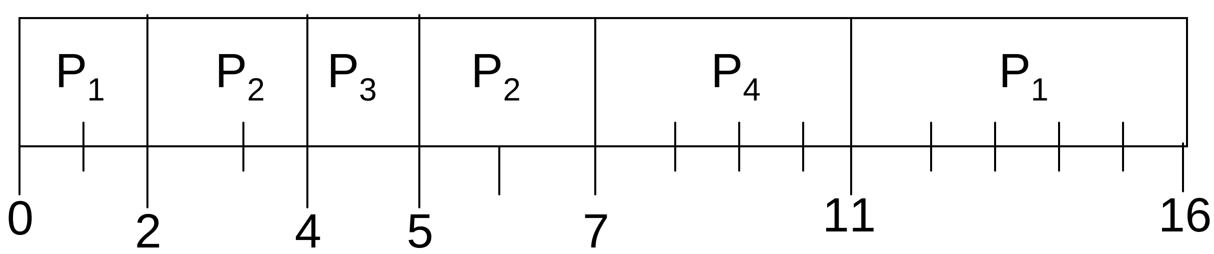


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

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

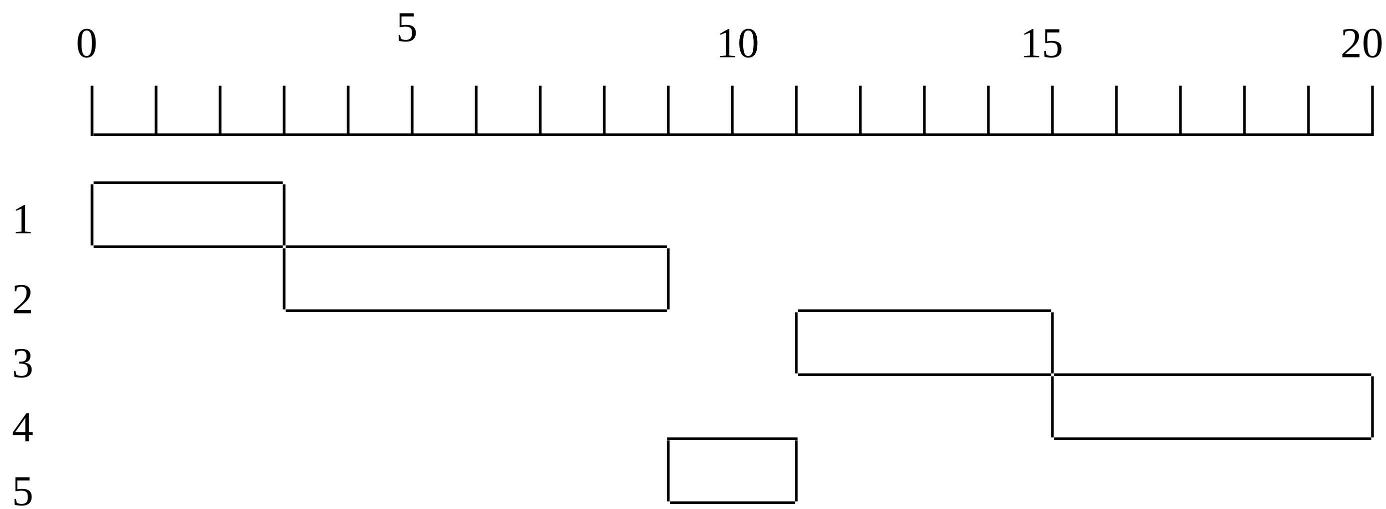
- SJF (preemptive)



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

Shortest Process Next

- Nonpreemptive policy
- Process with shortest expected processing time is selected next
- Short process jumps ahead of longer processes

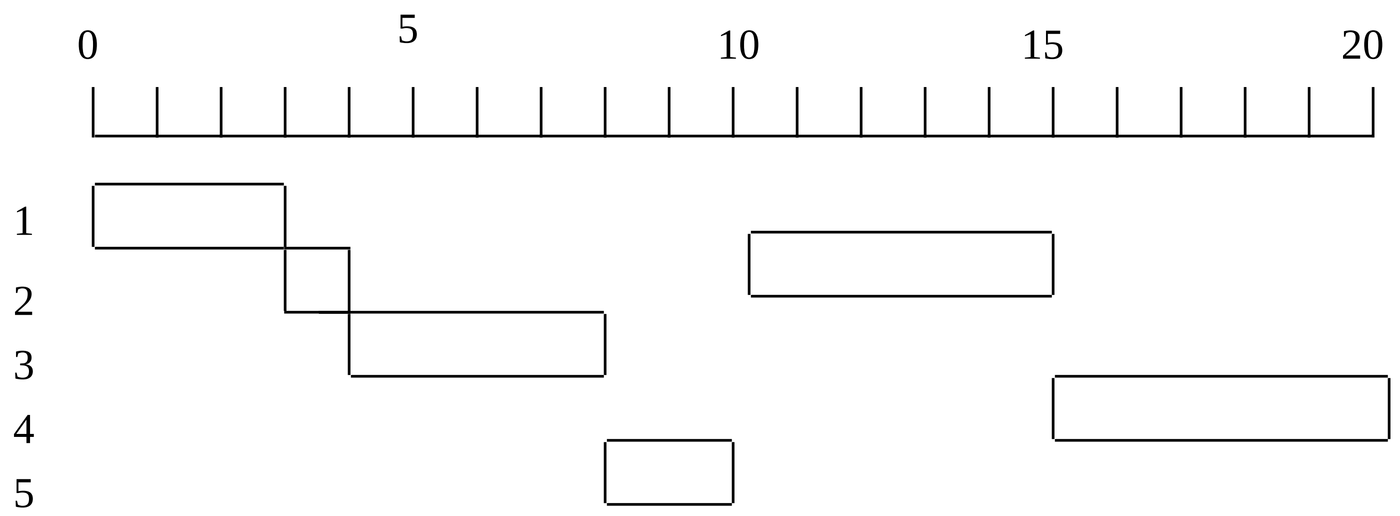


Shortest Process Next

- Predictability of longer processes is reduced
- If estimated time for process not correct, the operating system may abort it
- Possibility of starvation for longer processes

Shortest Remaining Time

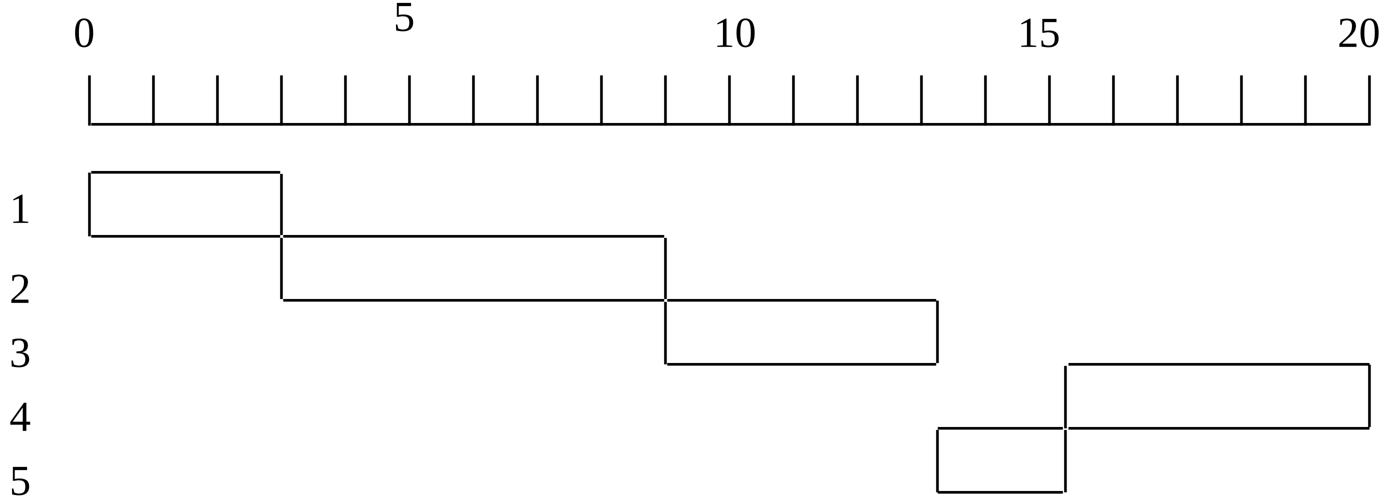
- Preemptive version of shortest process next policy
- Must estimate processing time



Highest Response Ratio Next (HRRN)

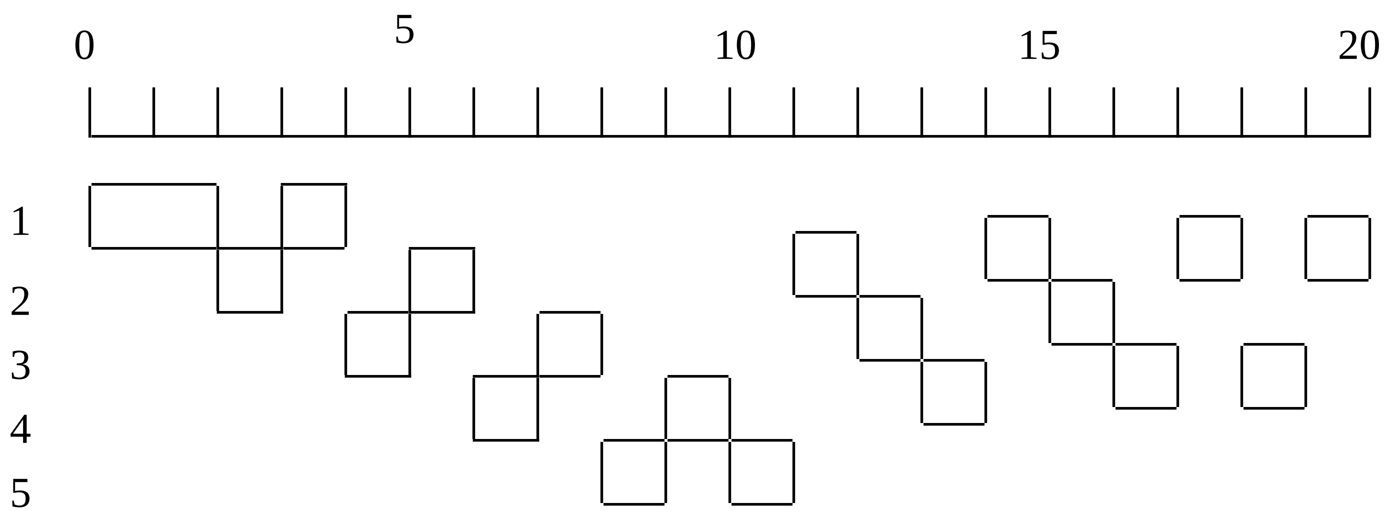
- Choose next process with the highest ratio

$$\frac{\textbf{time spent waiting + expected service time}}{\textbf{expected service time}}$$

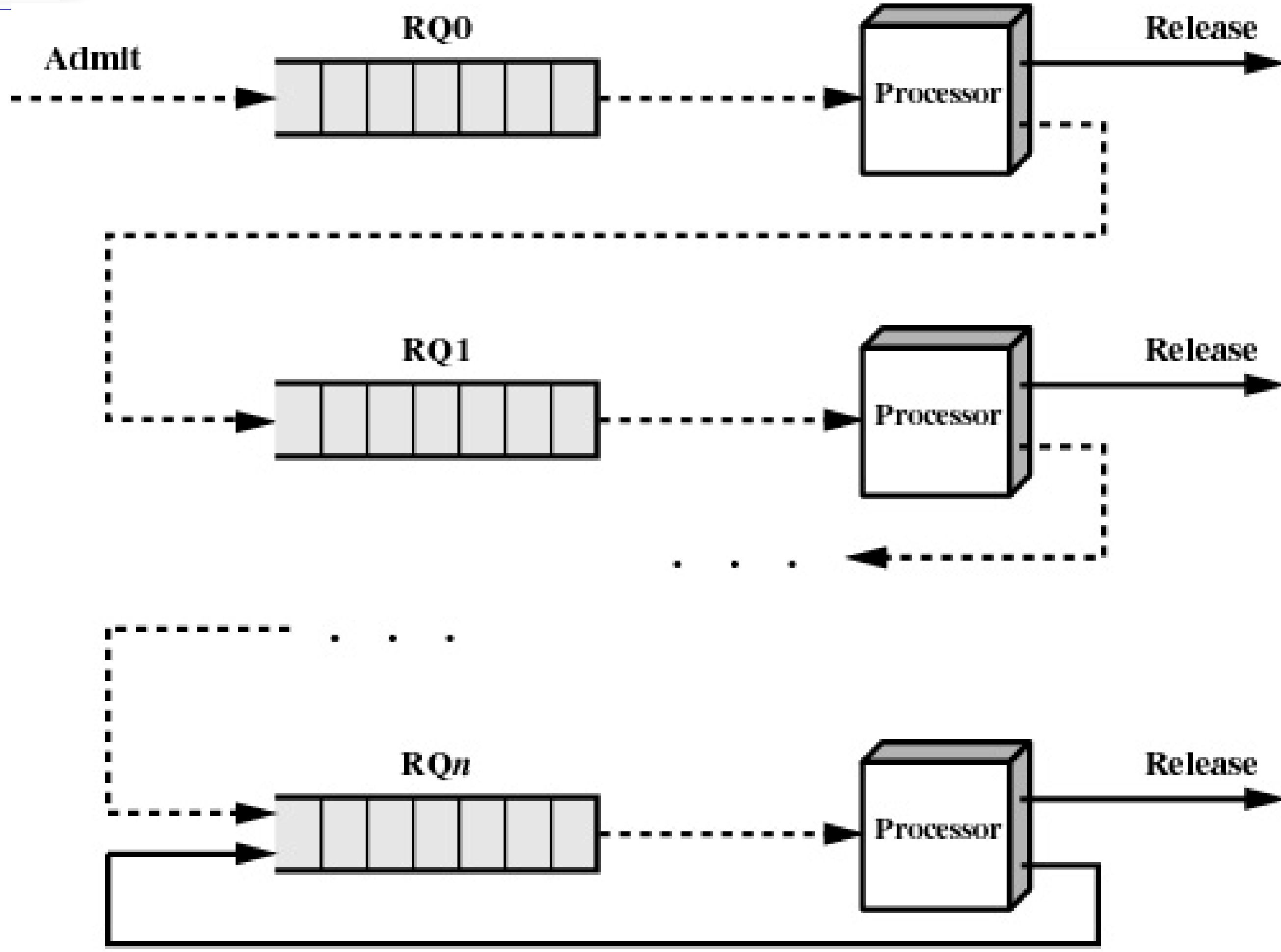


Feedback

- Penalize jobs that have been running longer
- Don’t know remaining time process needs to execute



Feedback Scheduling



Fair-Share Scheduling

- User's application runs as a collection of processes (threads)
- User is concerned about the performance of the application
- Need to make scheduling decisions based on process sets

Example of Fair-Share Scheduling three Processes two Groups

Time	Process A Priority	Process A Process	Process A Group	Process B Priority	Process B Process	Process B Group	Process C Priority	Process C Process	Process C Group
0	60	0 1 2 . . 60	0 1 2 . . 60	60	0	0	60	0	0
1	90	30	30	60	0 1 2 . . 60	0 1 2 . . 60	60	0	0 1 2 . . 60
2	74	15 16 17 . . 75	15 16 17 . . 75	90	30	30	75	0	30
3	96	37	37	74	15	15 16 17 . . 75	67	0 1 2 . . 60	15 16 17 . . 75
4	78	18 19 20 . . 78	18 19 20 . . 78	81	7	37	93	30	37
5	98	39	39	70	3	18	76	15	18

Group 1 (Process A)

Group 2 (Process B, Process C)

Shaded rectangle rapresent executing process

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 lenght of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n$$

Examples of Exponential Averaging

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

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

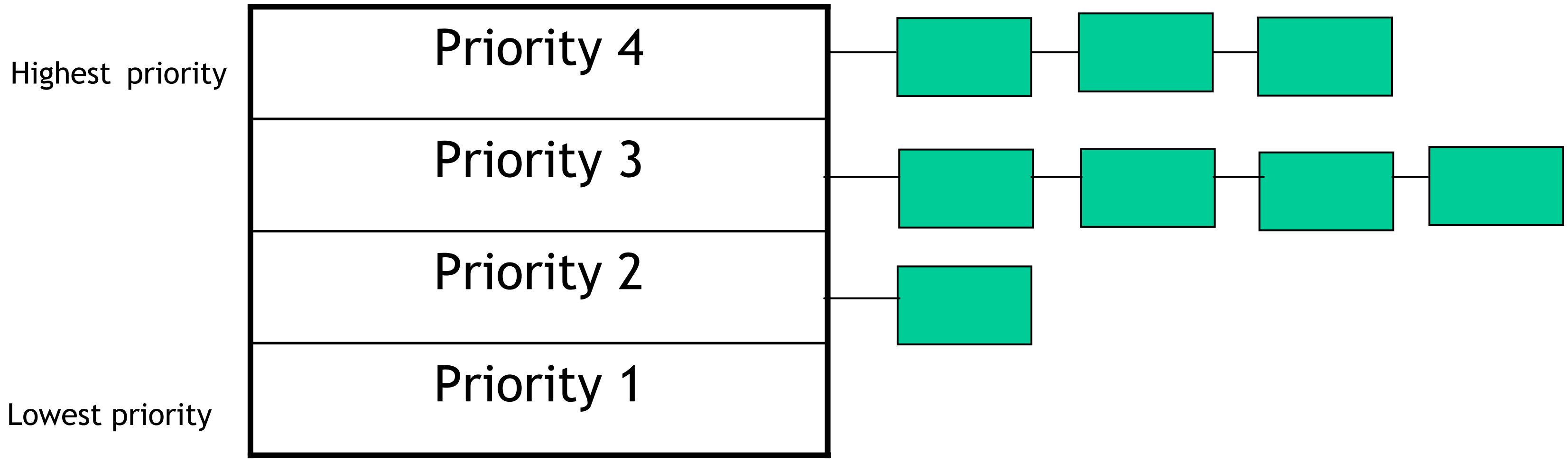
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

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
  - nonpreemptive
- SJN 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

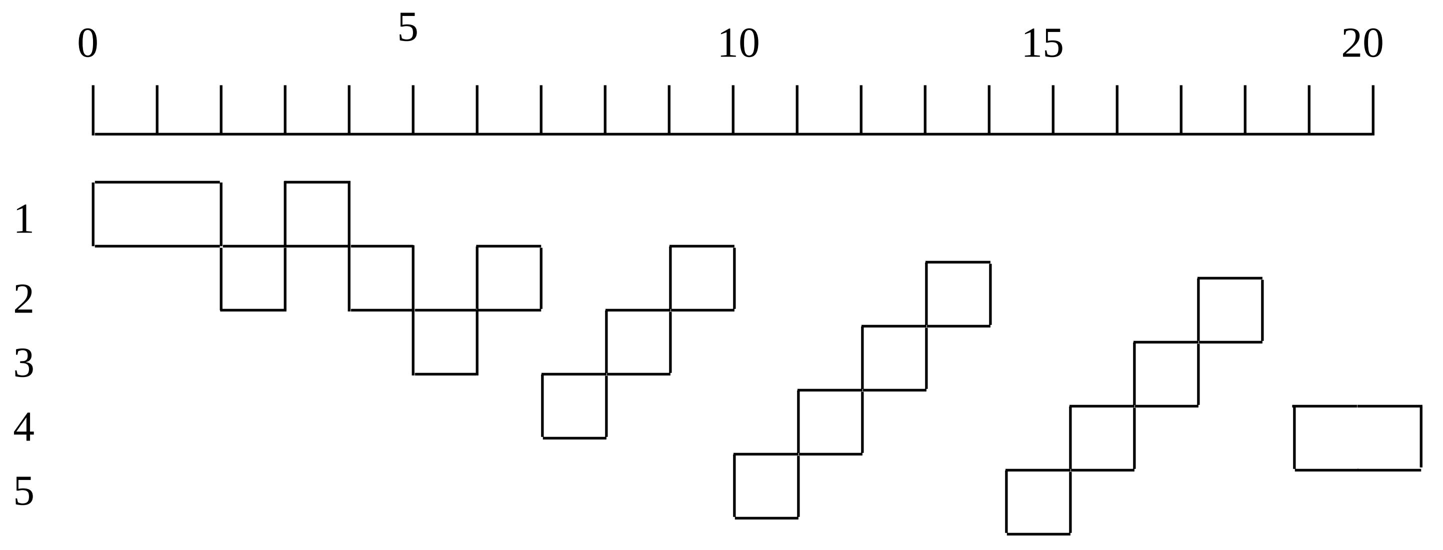
Priority Scheduling

A scheduling algorithm with four priority classes



Round-Robin

- Uses preemption based on a clock
- An amount of time is determined that allows each process to use the processor for that length of time



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

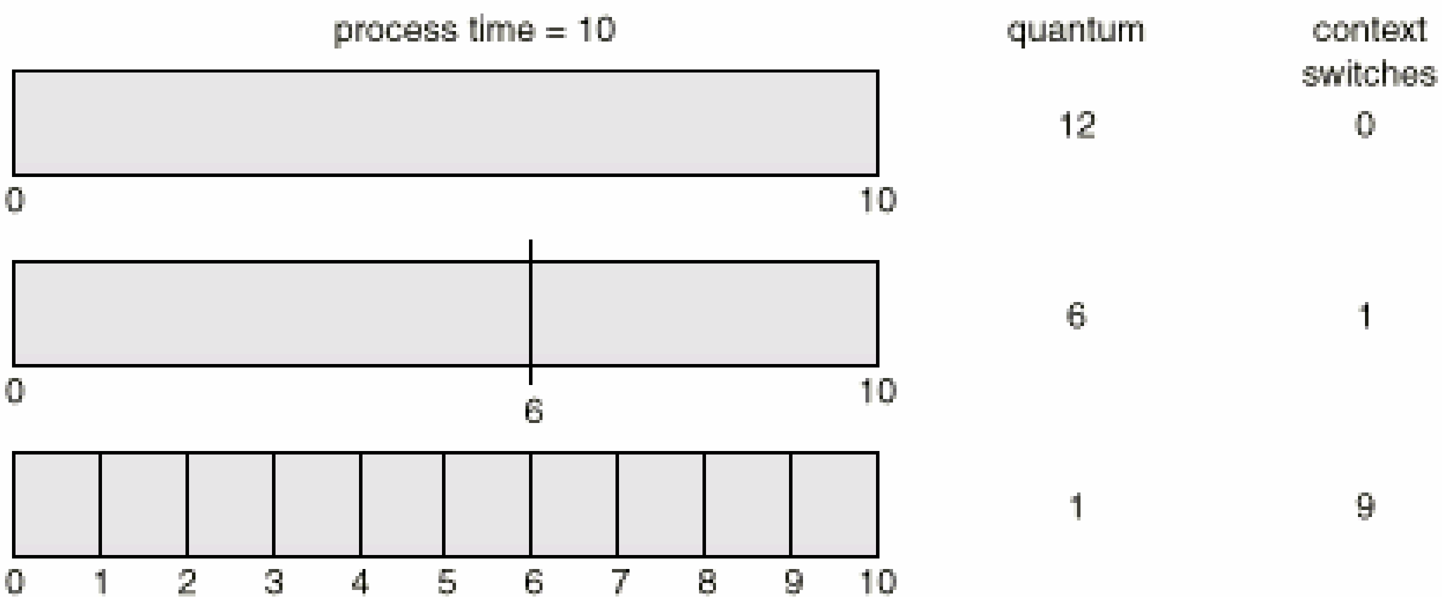
- The Gantt chart is:

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

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

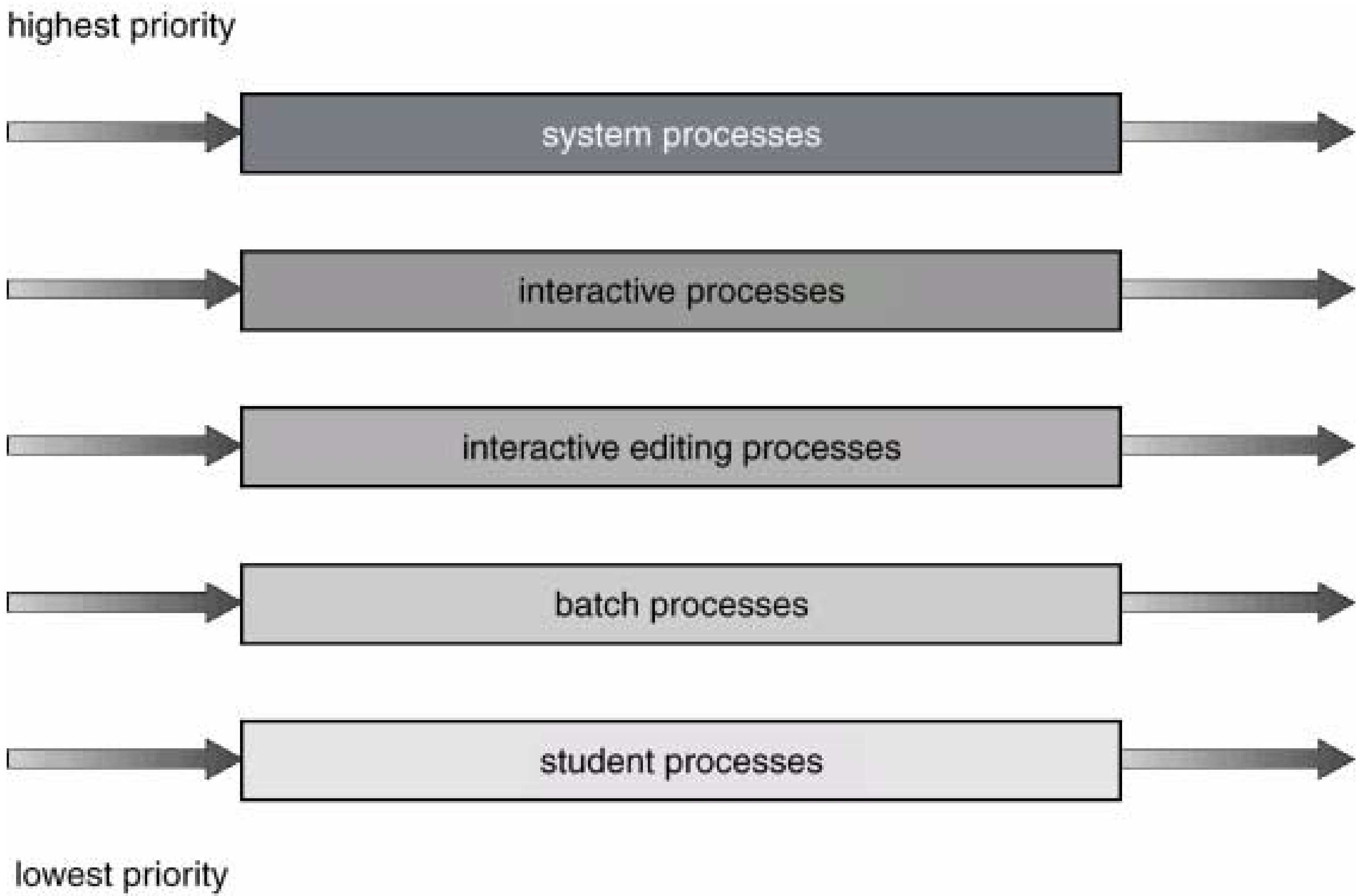
How a Smaller Time Quantum Increases Context Switches



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

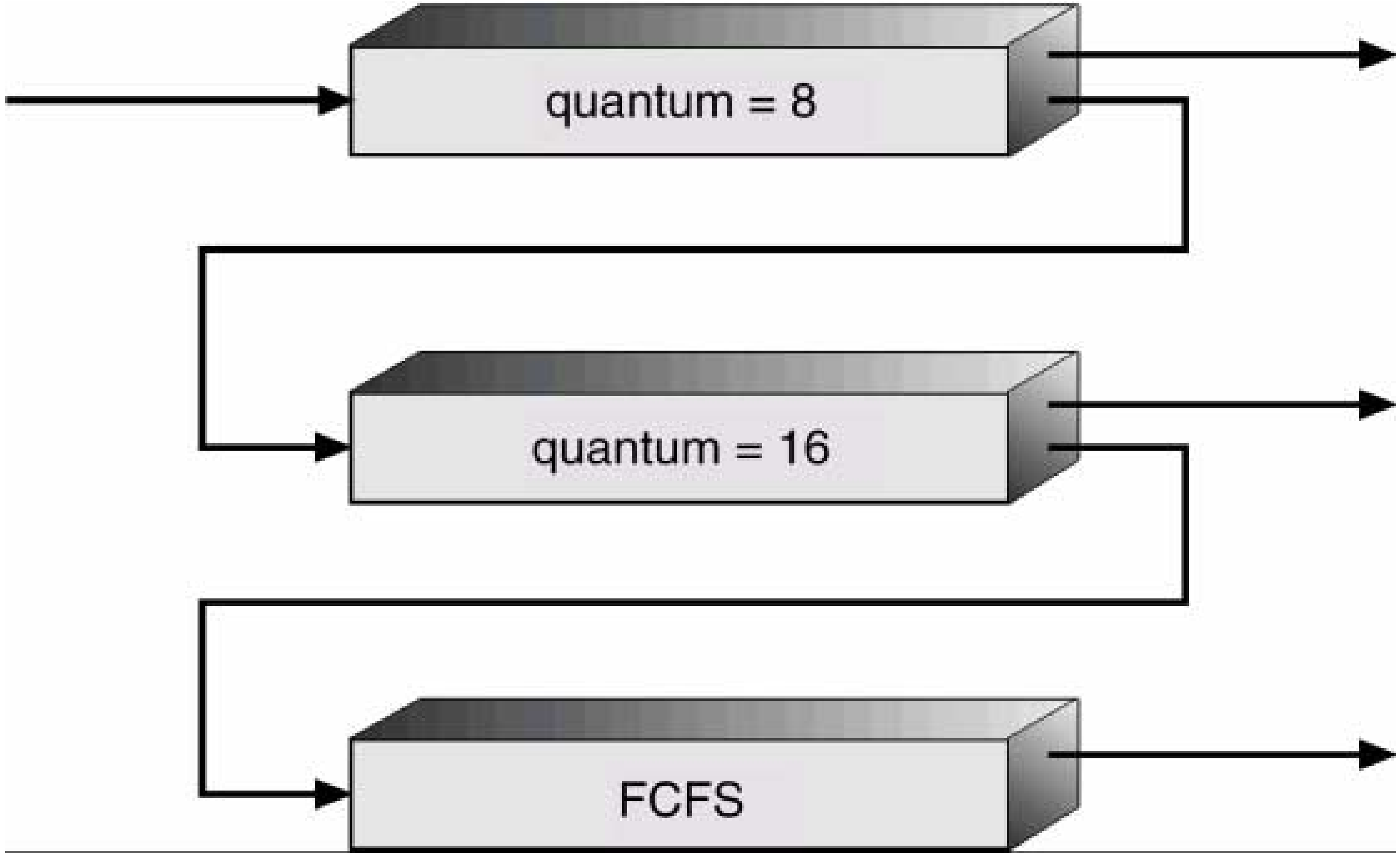
Multilevel Queue Scheduling



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