Scheduling for uniprocessor systems
Introduction

Lecturer:
William Fornaciari
Politecnico di Milano

william.fornaciari@polimi.it
Home.dei.polimi.it/fornacia

SUMMARY

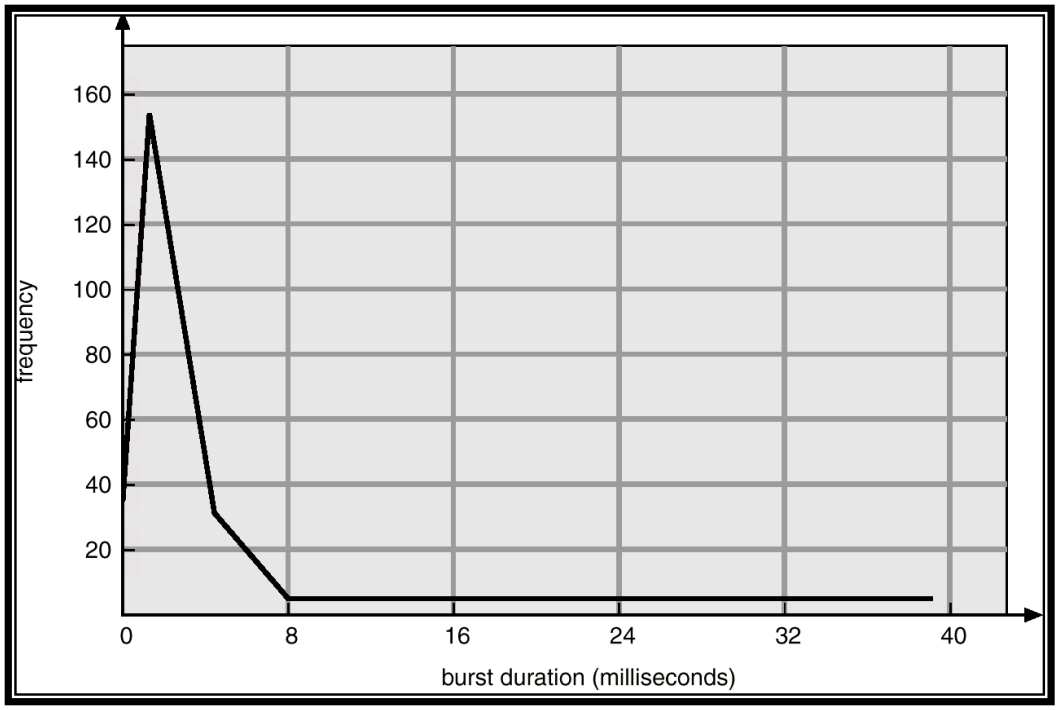
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms

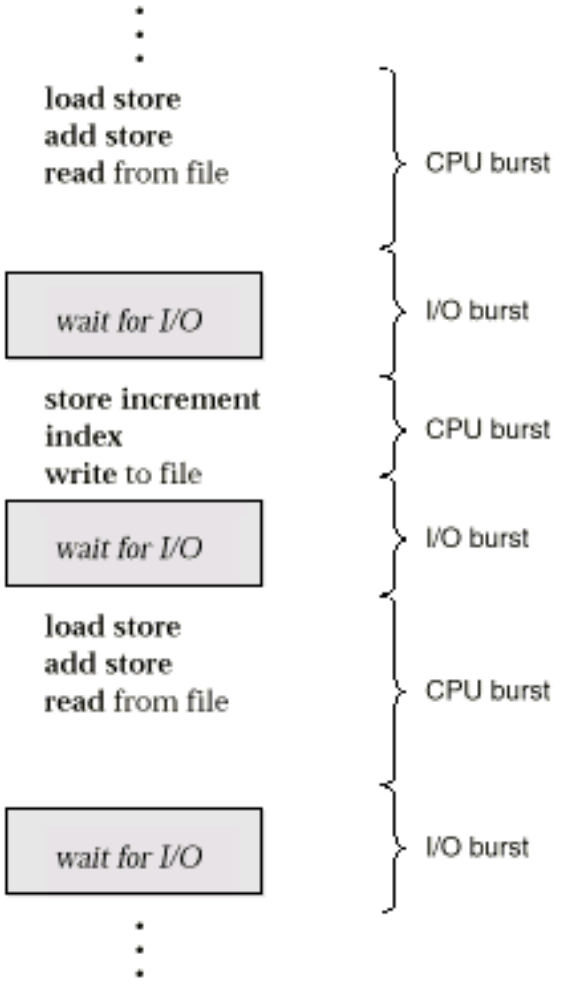
...more to come

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*.

Preemtive vs not preemptive

- Non-preemptive or cooperative scheduling
  - Once the CPU has been allocated to a process, the process keeps the CUP until it releases the CPU either by terminating or by switching to the waiting state
- Preemptive
  - A process having obtained the CPU may be forced to release the CPU…
    - Windows, UNIX

Preemptive

- Incur a cost
  - 2 processes share data: coordinating processes
- Effect on kernel design
  - during a system call: chaos if kernel data update!
  - so, wait for sys. call or I/O to complete (but it's bad for real-time computing/multiprocessing)
- Interrupt comes any time, can't be ignored by the kernel
  - Code sections (of interrupts) are not accessed concurrently by several processes
    - disable at entry/enable when leave (but time-consuming!)

Dispatcher

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

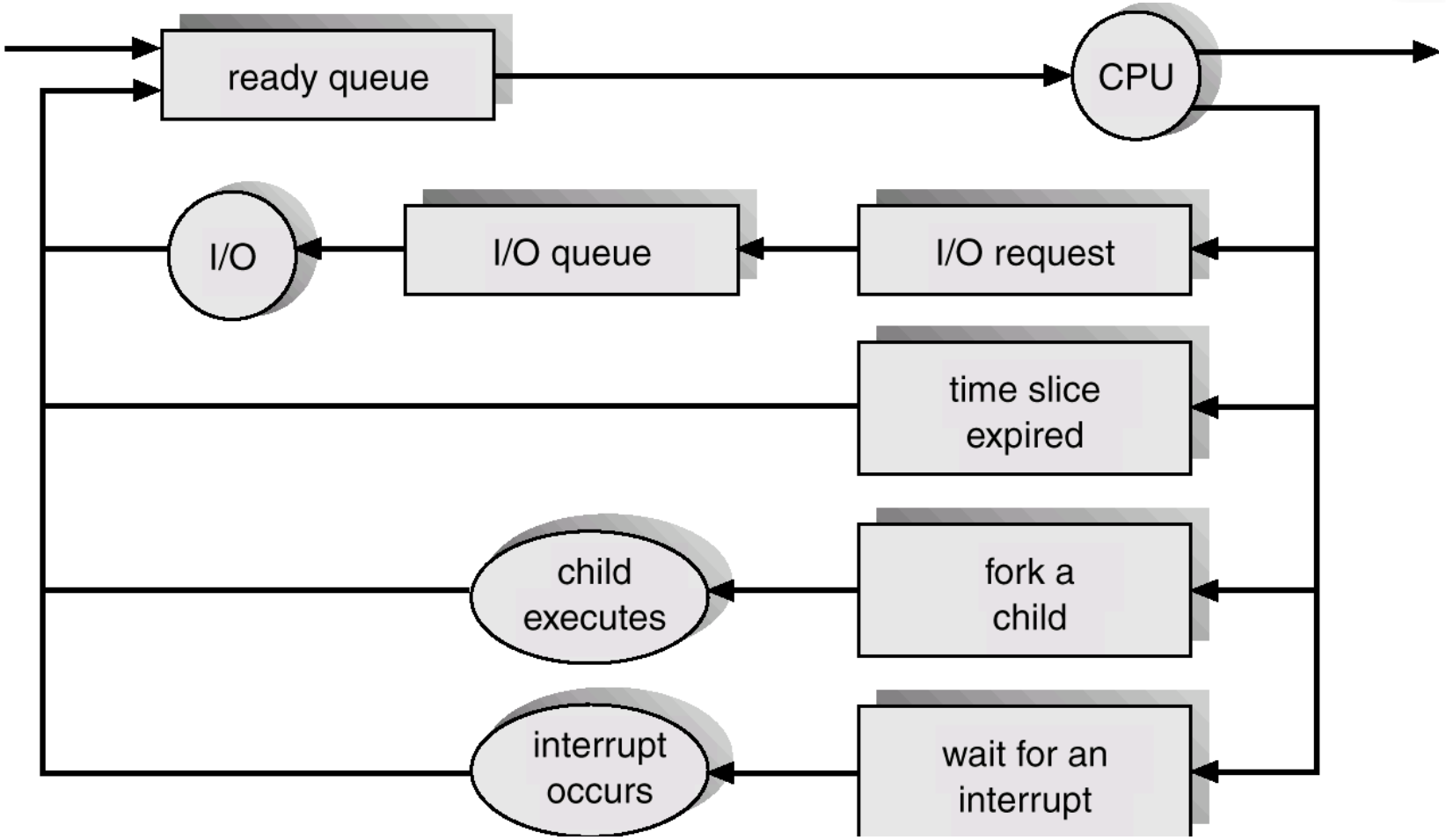
Scheduling Criteria

- 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 timesharing environment)
- Predictability, fairness, balance resources, priority…

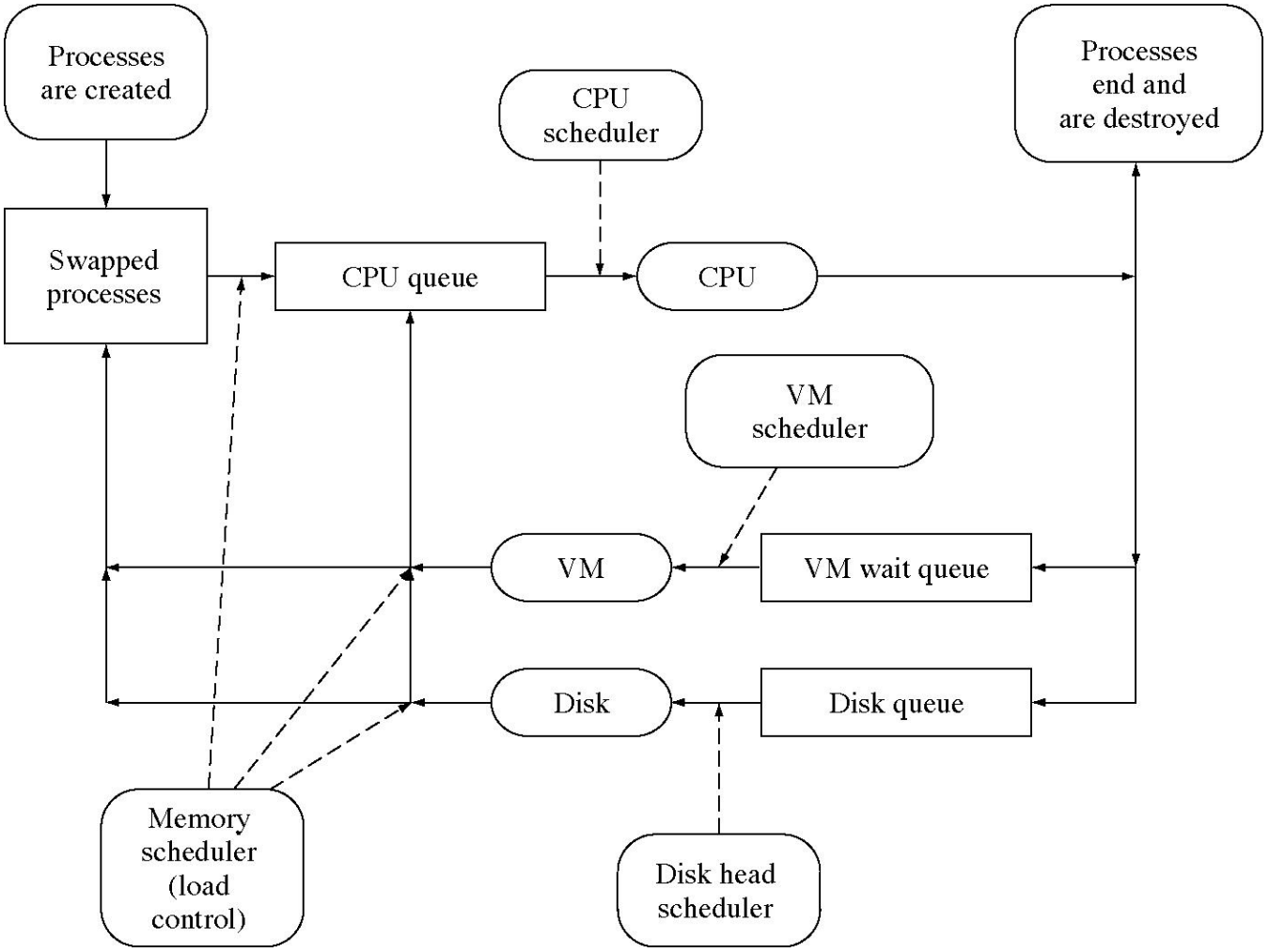
Scheduling Criteria (cont.)

- Optimization Criteria -- may be conflict
  - Max. CPU utilization
    - real: 40%~90%
  - Max. **throughput**
    - #proc. completed per time unit (eg.)
  - Min. turnaround time
  - Min. waiting time (where? ready queue!)
  - Min. response time (for interactive system)
- In real cases…
  - Minimize the variance in the response time (desire: reasonable, predictable)
  - Minimize the **average waiting time**

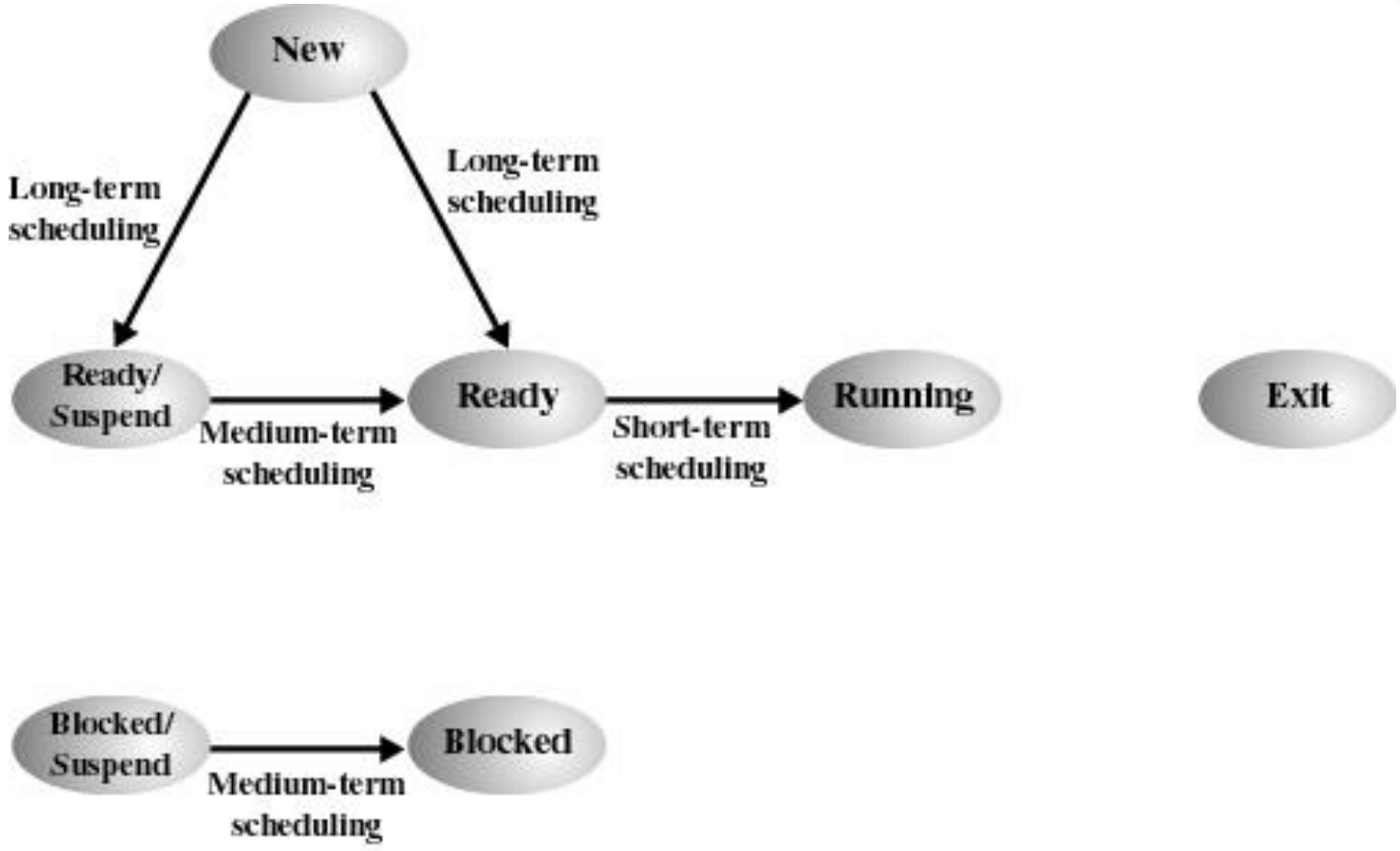
Process Scheduling: Examples



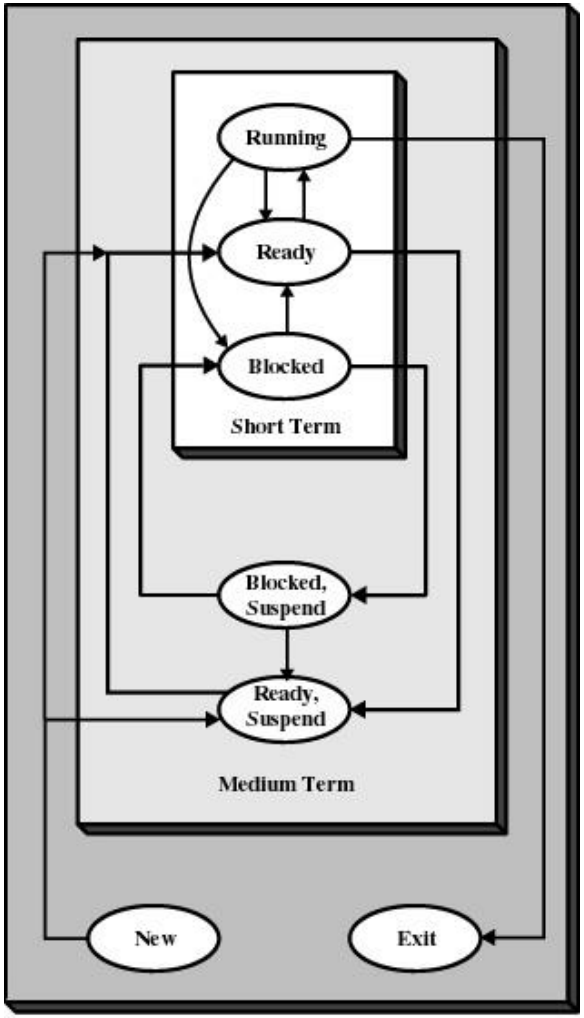
OS Schedulers



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

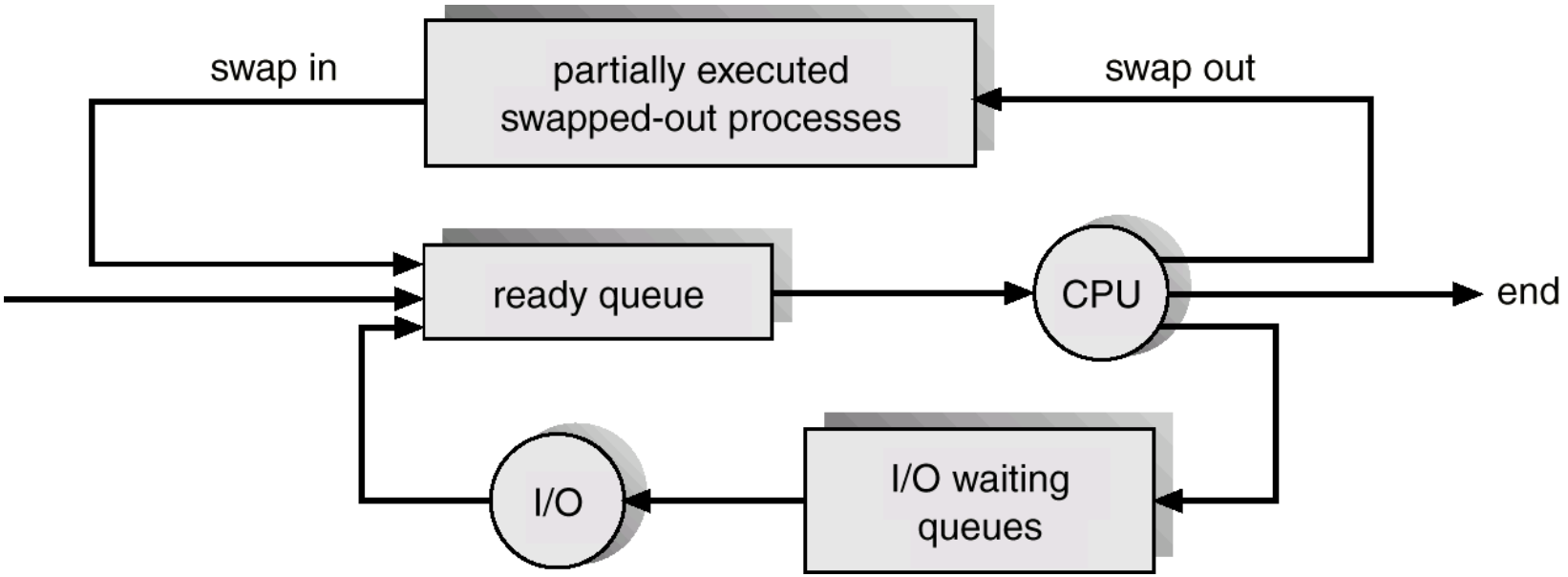
Medium-Term Scheduling

- 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

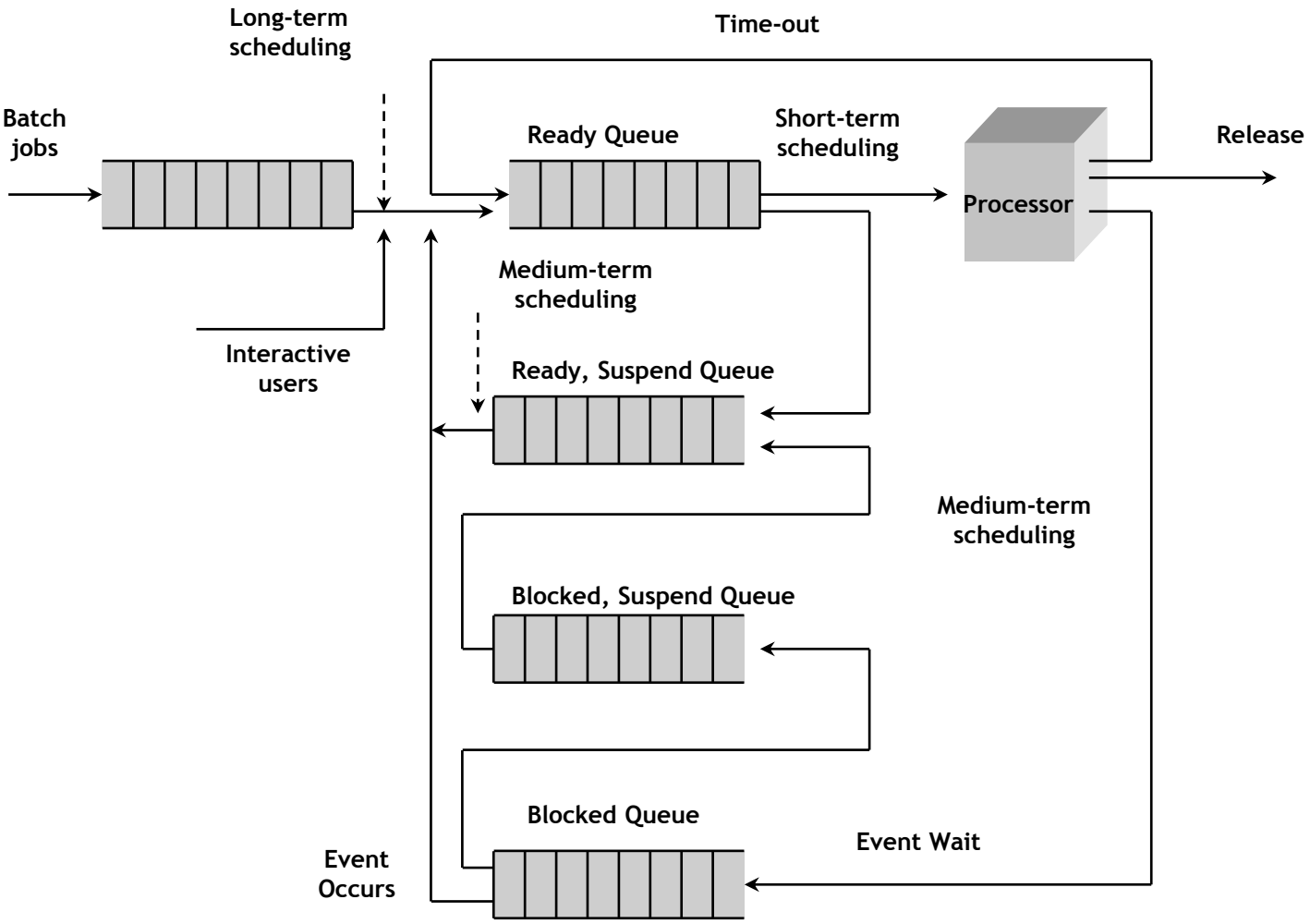
Short-Term Scheduling Criteria

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

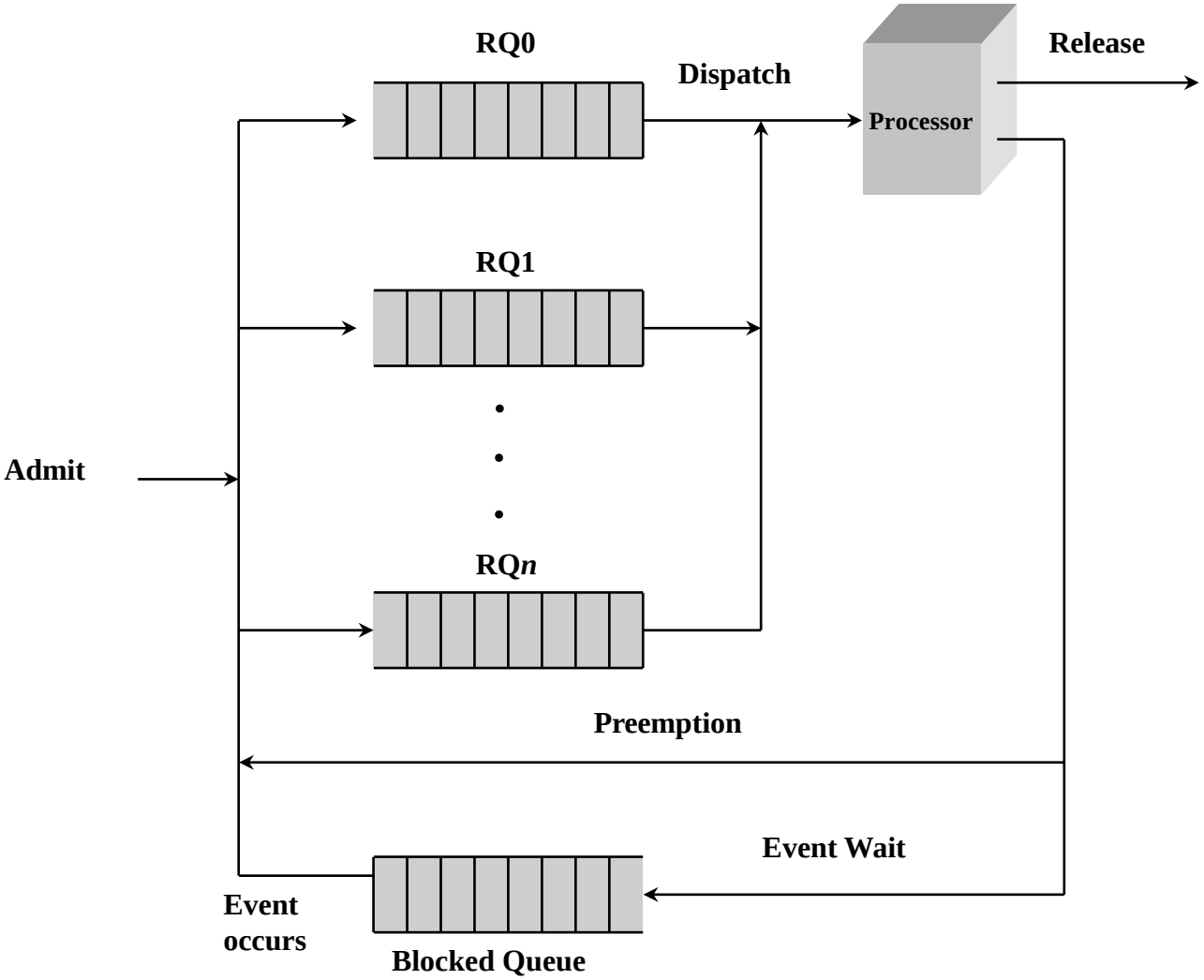
Priorities

- 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



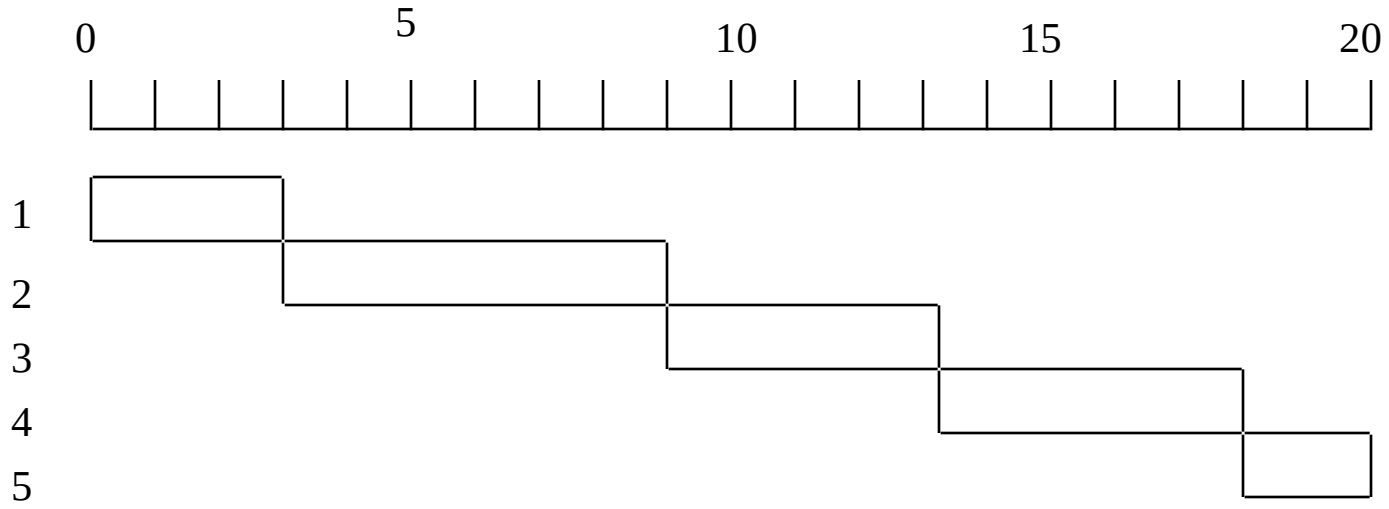
Decision Mode

- 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

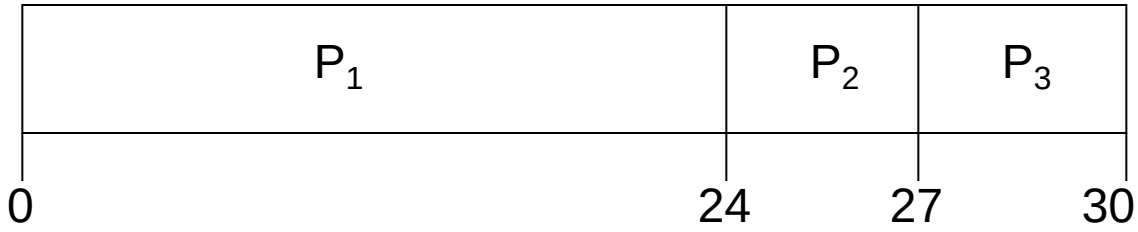
- A short process may have to wait a very long time before it can execute
- Favours CPU-bound processes
  - I/O processes have to wait until CPU-bound process completes

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:



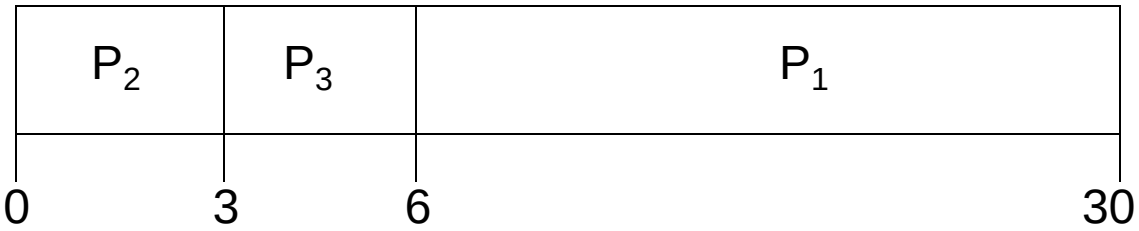
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

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. (non –preemptive)
- *Convoy effect* short process behind long process

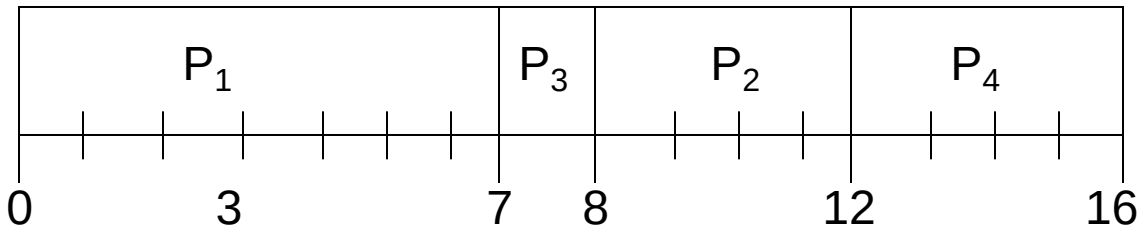
Shortest-Job-First (SJF) Scheduling

- 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)

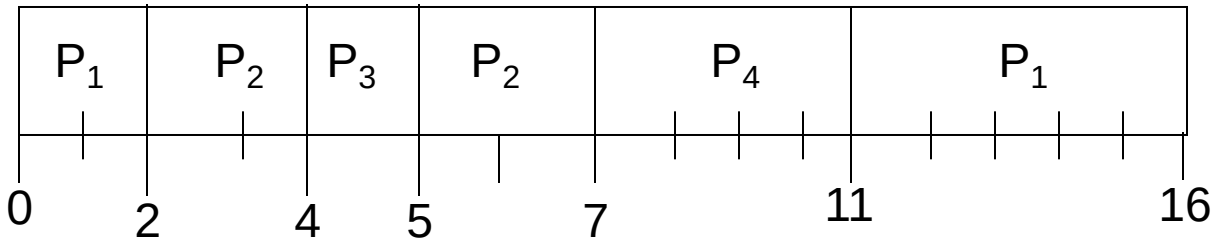


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

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

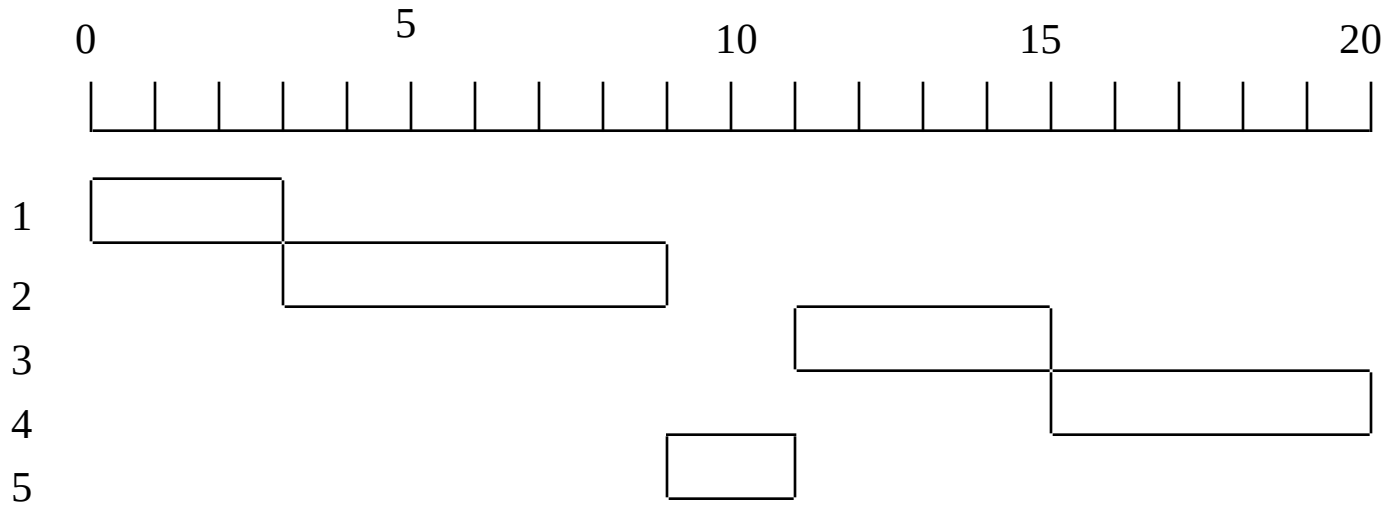
- SJF (preemptive)



- 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

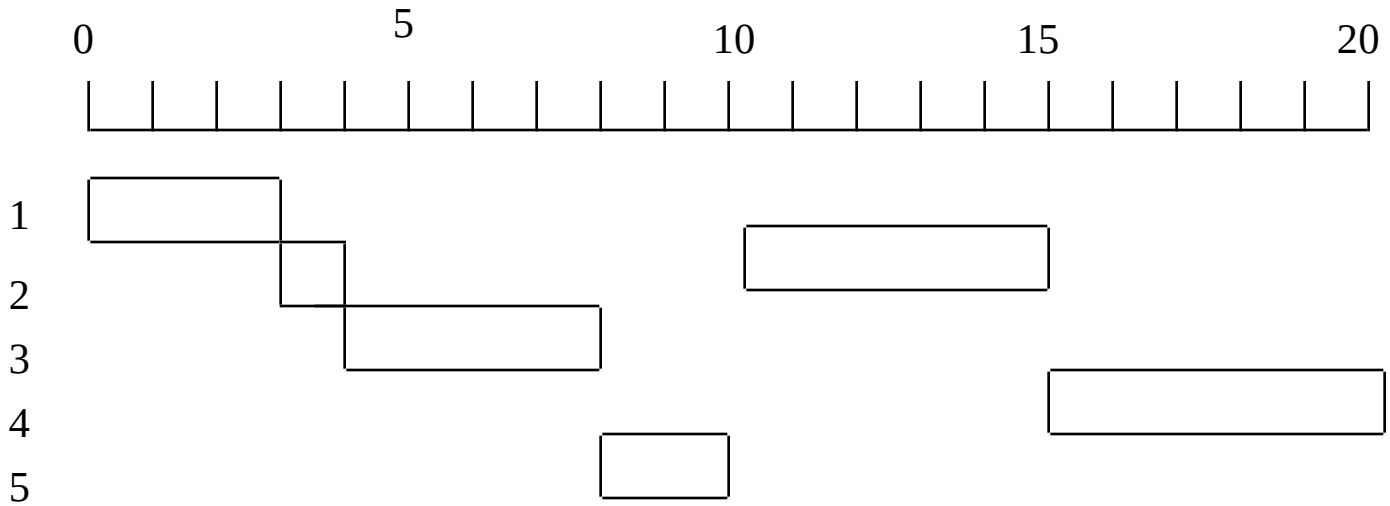


Shortest Process Next

- 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

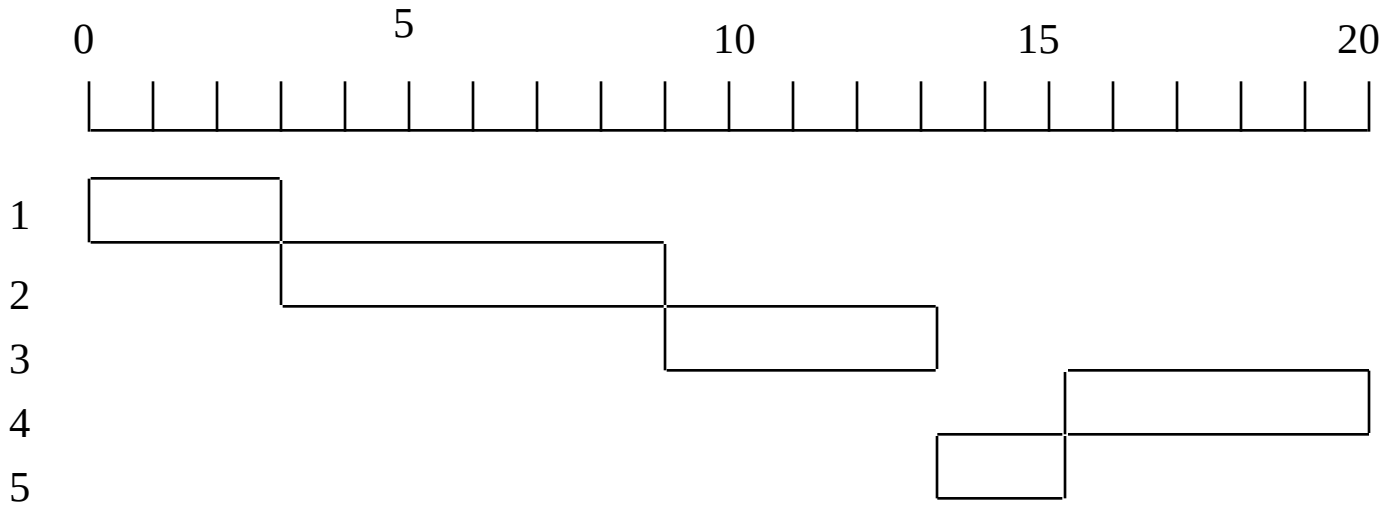
- Preemptive version of shortest process next policy
- Must estimate processing time



Highest Response Ratio Next (HRRN)

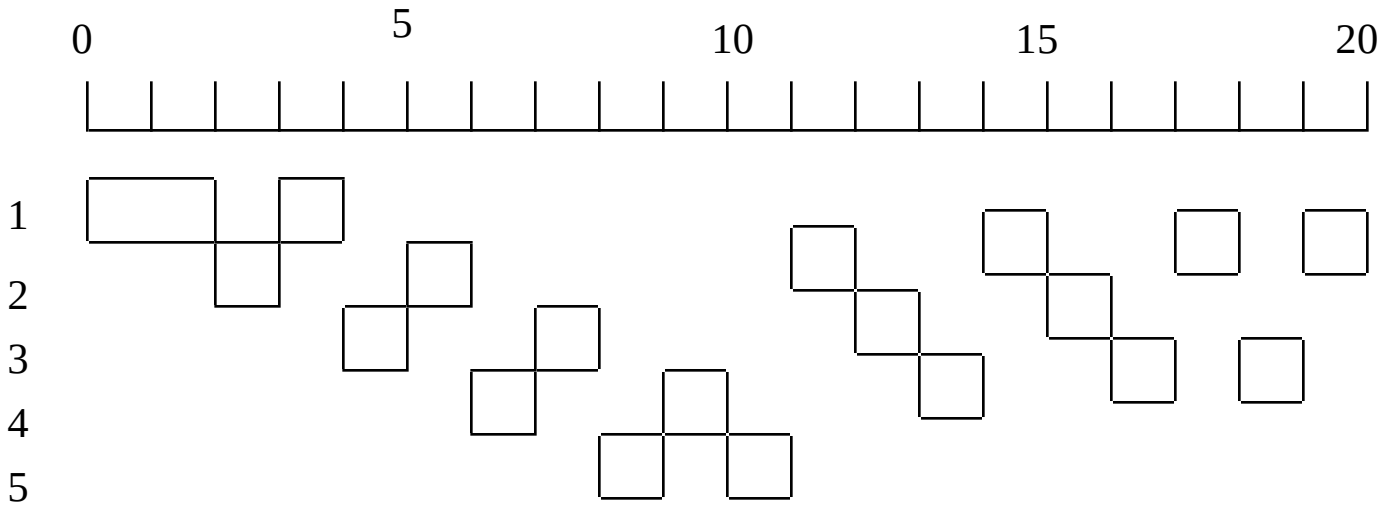
- Choose next process with the highest ratio

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

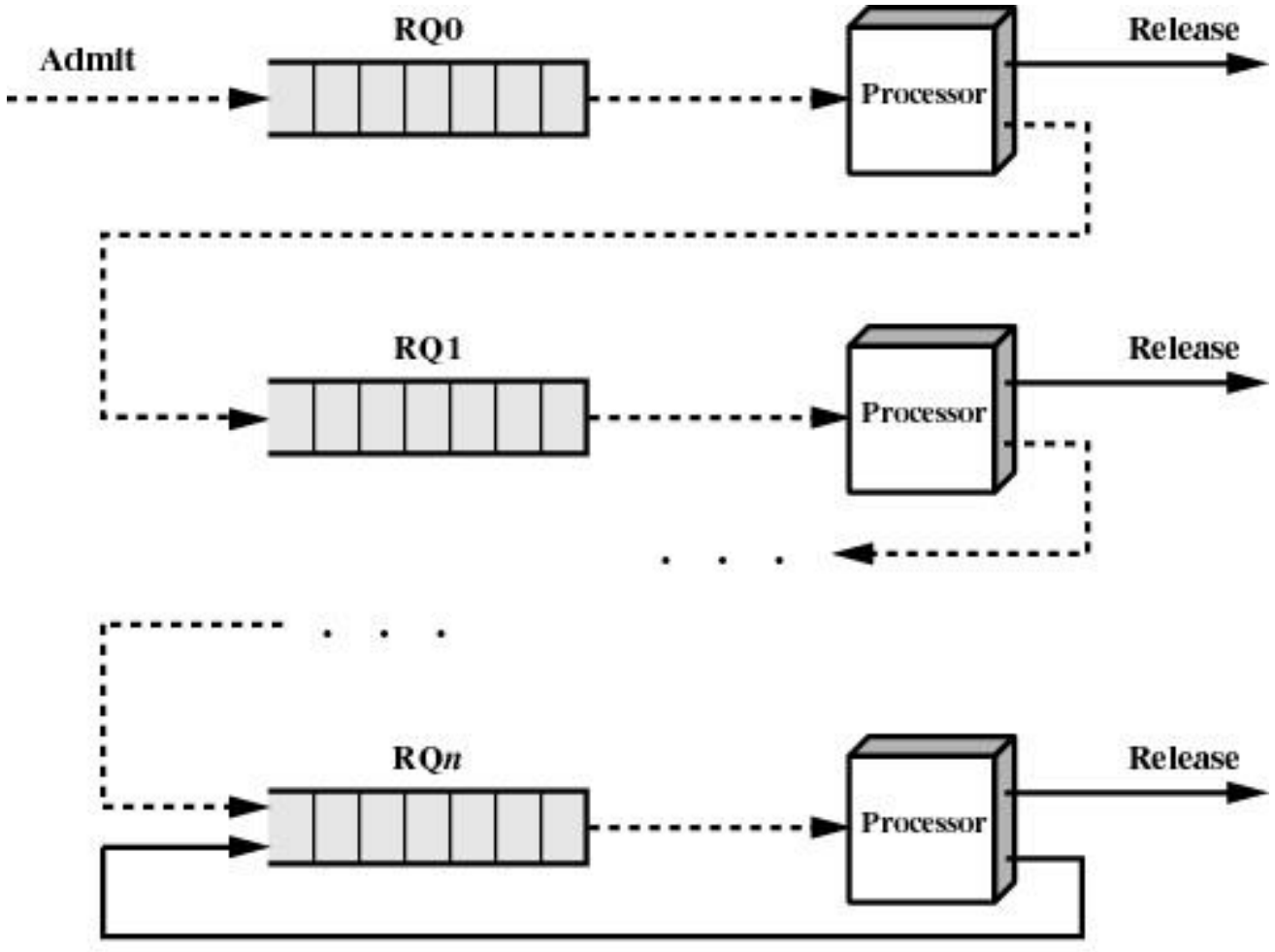


Feedback

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



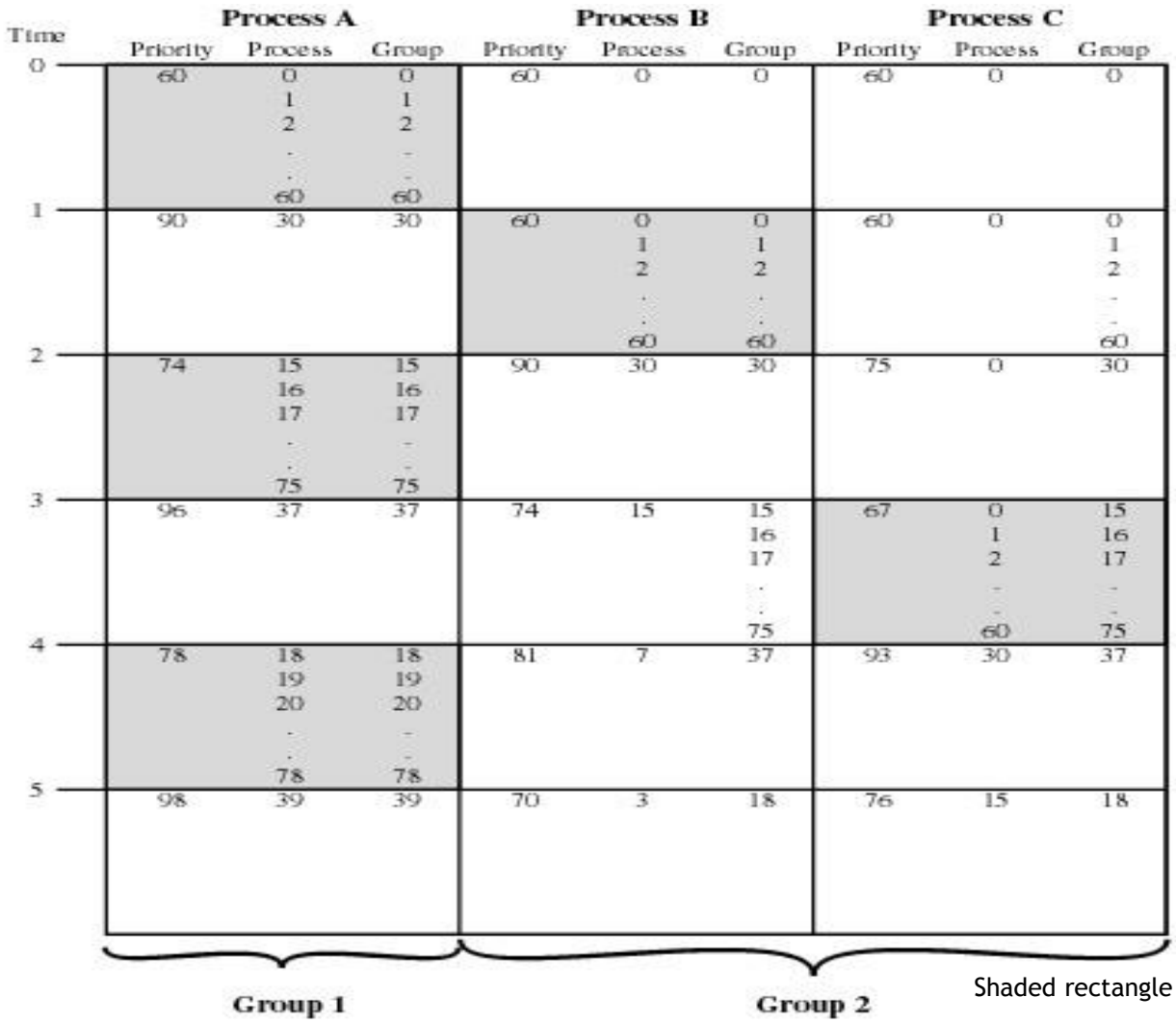
Feedback Scheduling



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

Group 1 Group 2

Shaded rectangle rapresent executing process

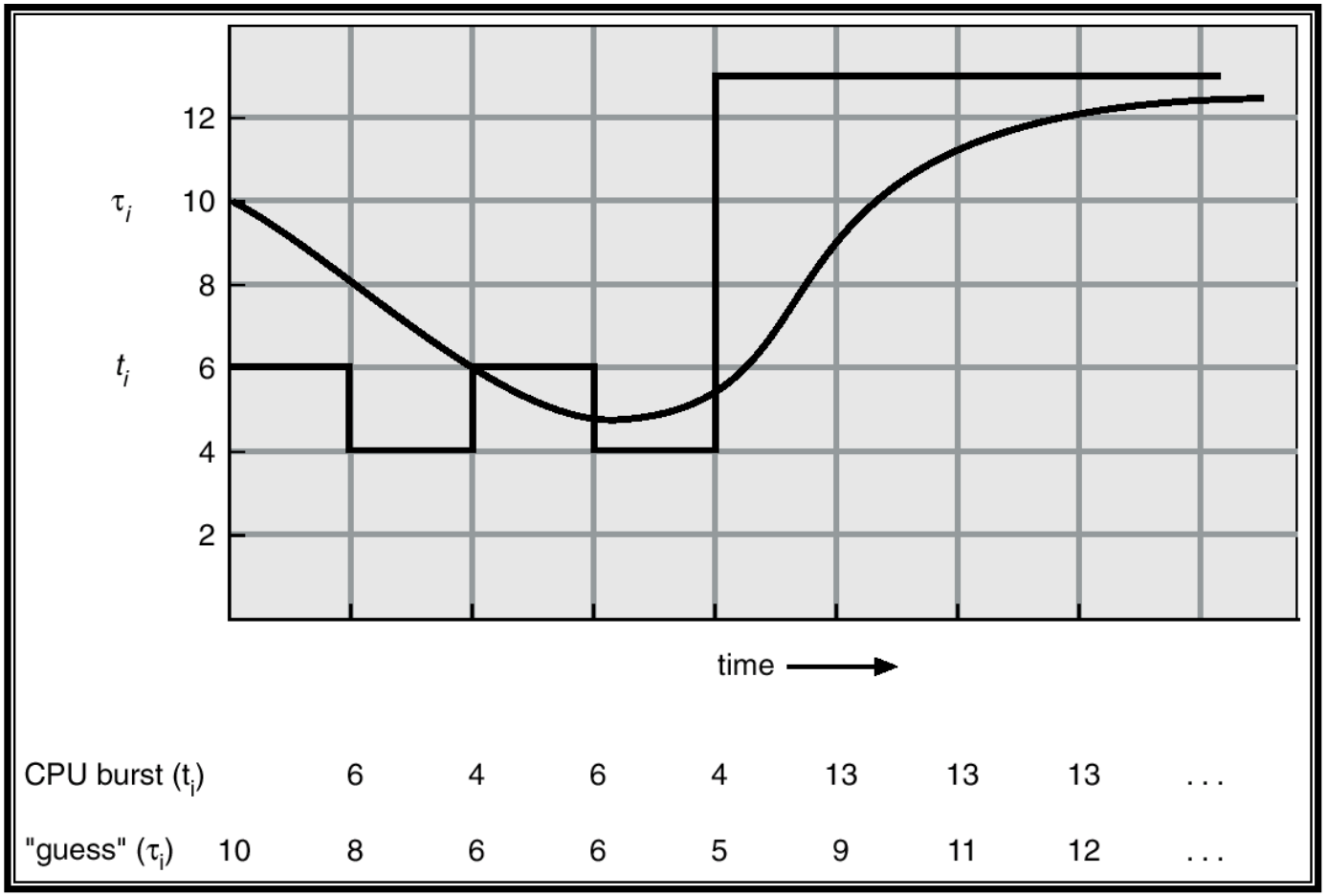
Determining Length of Next CPU Burst

- 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. τ_{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" (τ_i)	10	8	6	6	5	9	11	12	. . .

Examples of Exponential Averaging

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

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

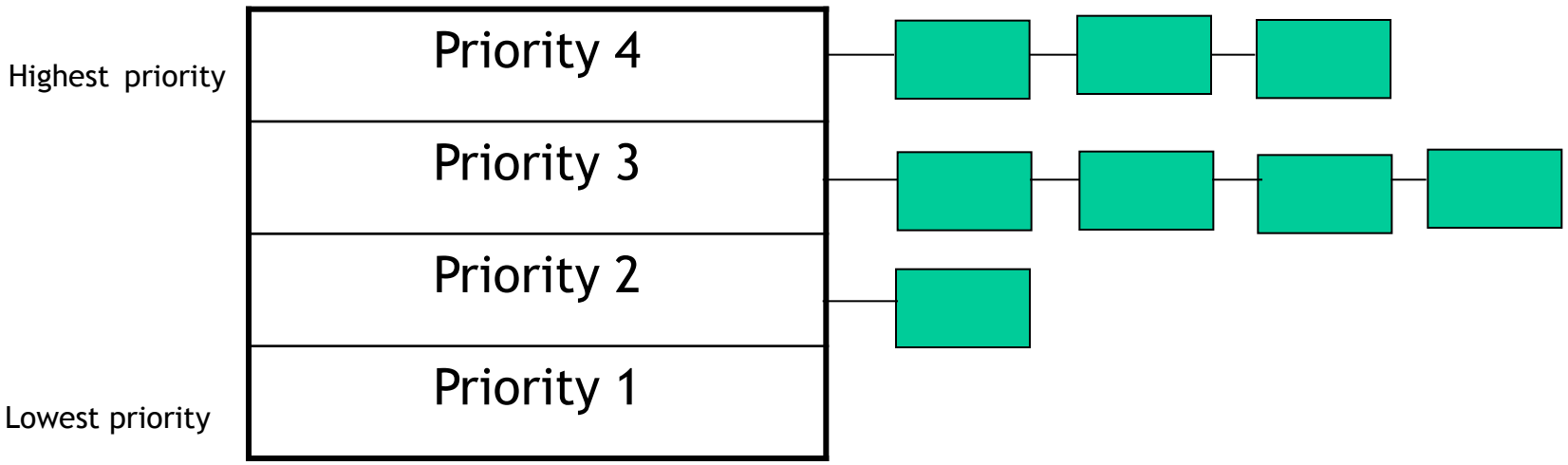
- Since both α and (1 - α) are less than or equal to 1, each successive term has less weight than its predecessor

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
- Define priority: internal or external

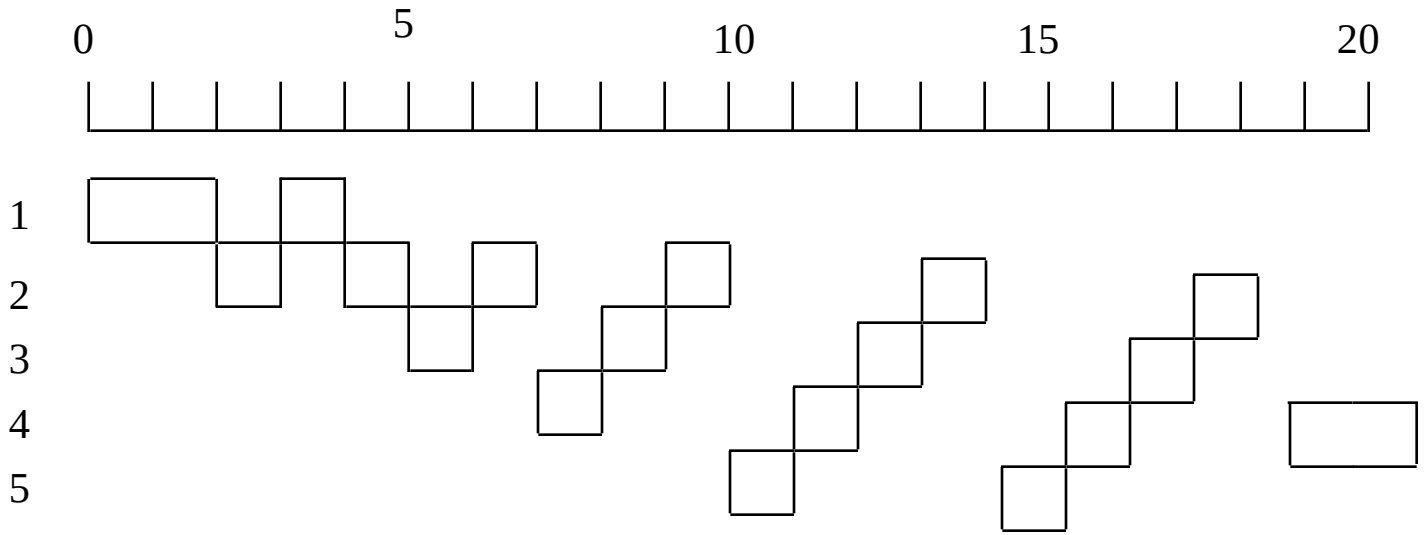
Priority Scheduling

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



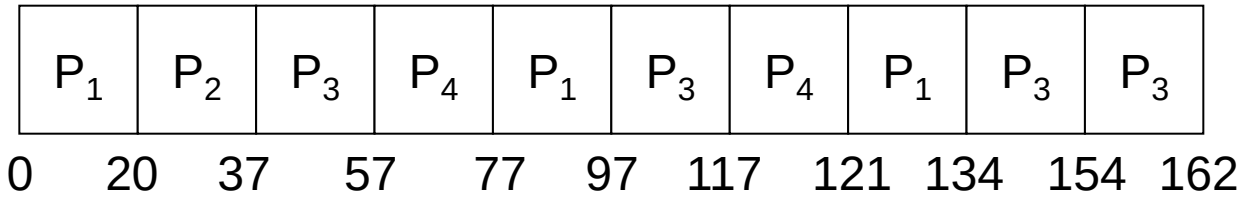
Round Robin (RR)

- 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

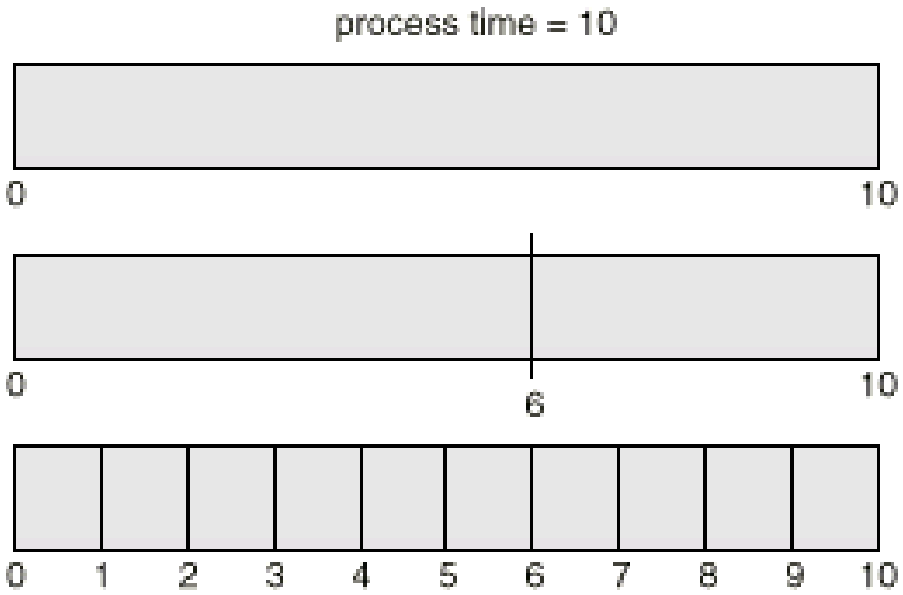
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



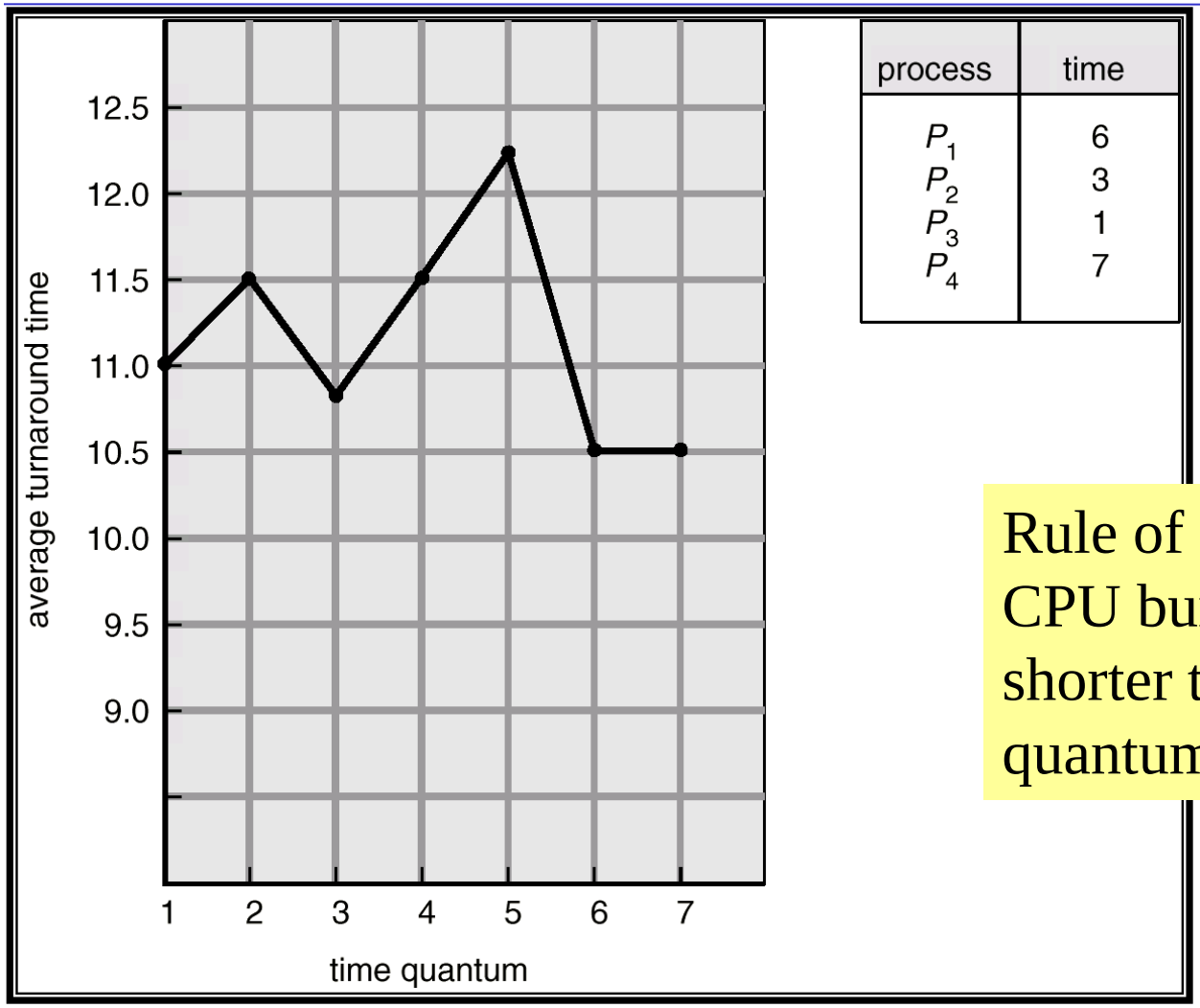
- Typically, higher average turnaround than SJF, but better *response*

How a Smaller Time Quantum Increases Context Switches



quantum	context switches
12	0
6	1
1	9

Turnaround time varies with the time quantum



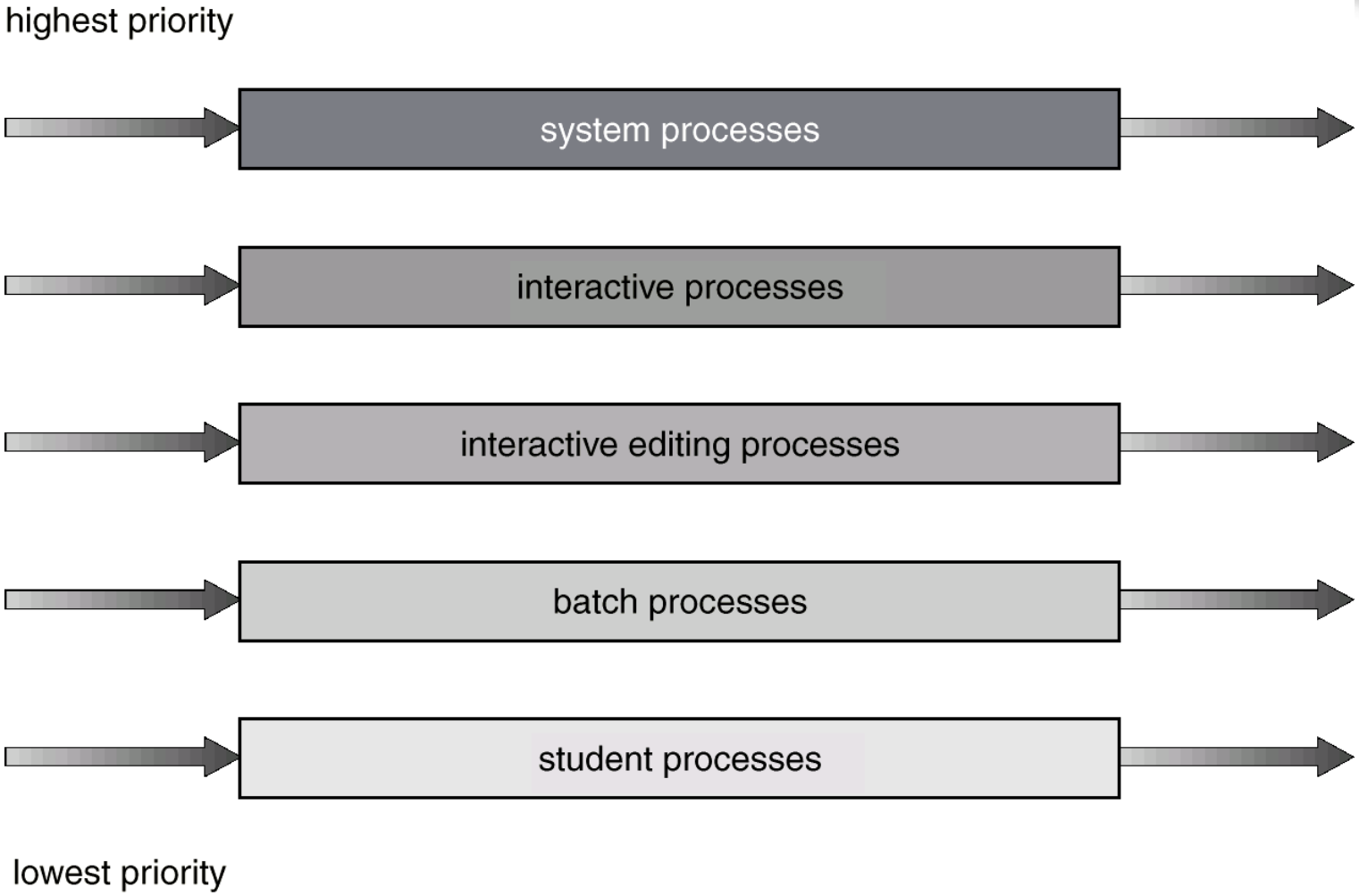
process	time
P_1	6
P_2	3
P_3	1
P_4	7

Rule of thumb: 80% of CPU bursts should be shorter than the time quantum

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

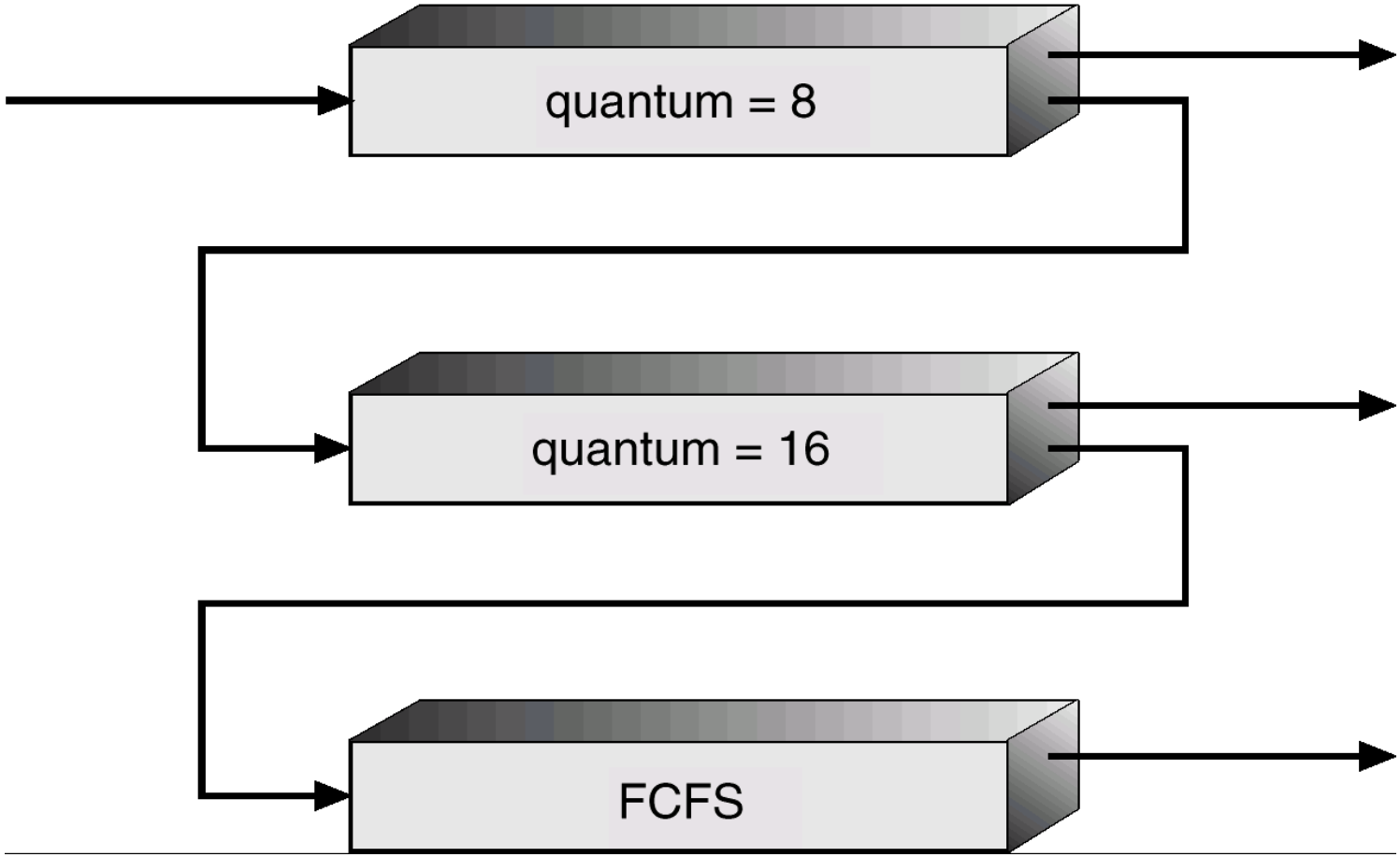
Multilevel Queue Scheduling



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

Multiple-Processor Scheduling

- CPU scheduling more complex
- Homogeneous processors within a multiprocessor
  - Separate vs. common ready queue
  - Load sharing
- Symmetric Multiprocessing (SMP) – each processor makes its own scheduling decisions
  - Access and update a common data structure
  - Must ensure two processors do not choose the same process; none lost
- Asymmetric multiprocessing – only the master process accesses the system data structures

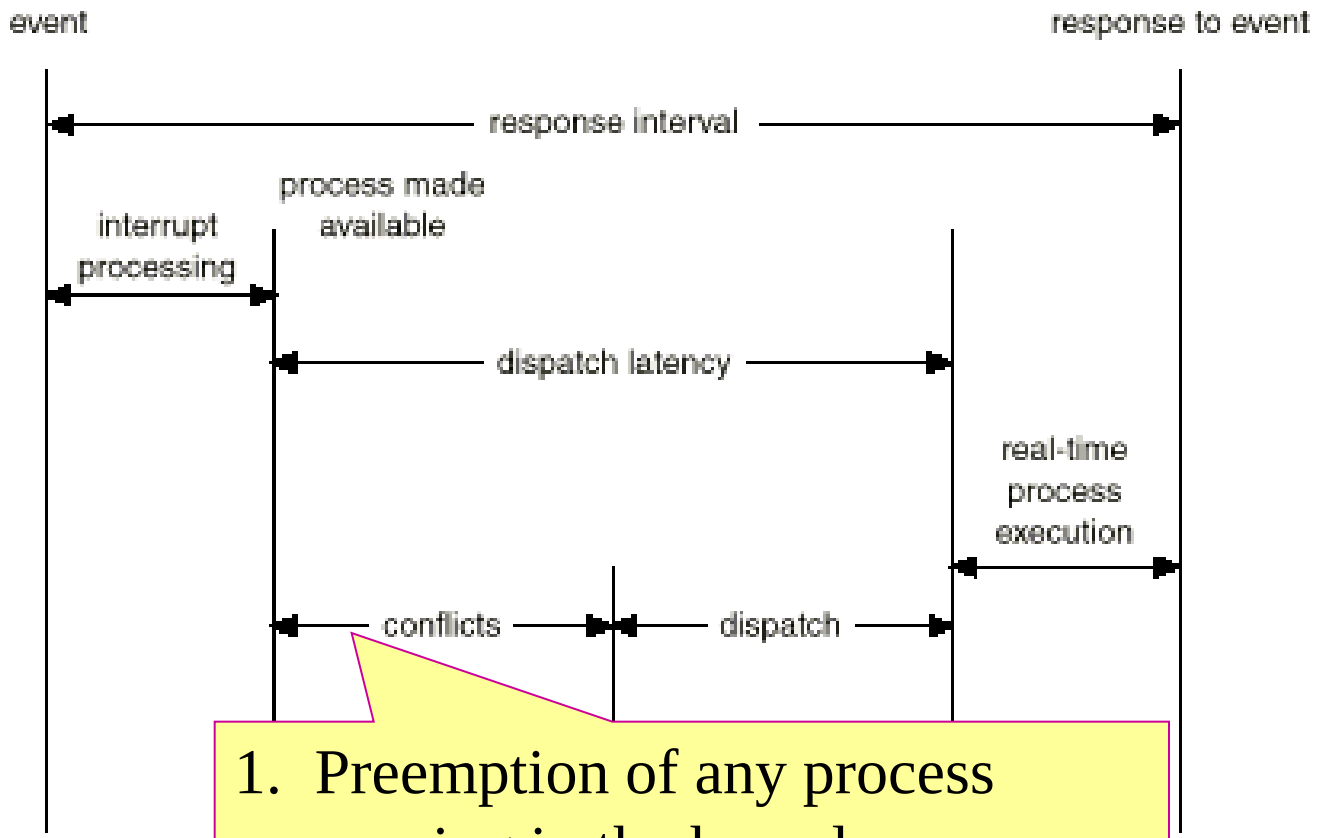
Real-Time Scheduling

- *Hard real-time systems* – required to complete a critical task within a guaranteed amount of time
  - A process is submitted with a statement of the amount of time in which it needs to complete or perform I/O. The scheduler uses the statement to admit (and guarantee) or reject the process
  - Resource reservation – requires the scheduler knows exactly how long each type of OS functions takes to perform
  - Hard real-time systems are composed of special-purpose SW running on HW dedicated to their critical process, and lack the full functionality of modern computers and OS

Real-Time Scheduling (cont.)

- *Soft real-time computing* – requires that critical processes receive priority over less fortunate ones.
  - Ex. Multimedia, high-speed interactive graphics…
- Related scheduling issues for soft real-time computing
  - Priority scheduling – real-time processes have the highest priority
    - The priority of real-time process must not degrade over time
  - Small dispatch latency
    - May be long because many OS wait either for a system call to complete or for an I/O block to take place before a context switch
      - Preemption point
      - Make the entire kernel preemptible (ex. Solaris 2)

Dispatch Latency



1. Preemption of any process running in the kernel
2. Release by low-priority processes resources needed by the high-priority process

Algorithm Evaluation

- Define the criteria for evaluation and comparison
  - Ex. Maximize CPU utilization under the constraint that the maximum response time is 1 second
- Evaluation methods
  - Deterministic modeling
  - Queuing models
  - Simulations
  - Implementation
- Environment in which the scheduling algorithm is used will change
  - Your algorithm is good today, but still good tomorrow?

Deterministic Modeling

- Analytic evaluation – use the given algorithm and the system workload to produce a formula or number that evaluate the performance of the algorithm for that workload
  - Deterministic modeling is one analytic evaluation
- Deterministic modeling – takes a particular predetermined workload and defines the performance of each algorithm for that workload
  - Simple, fast, and give exact numbers
  - Require exact numbers for input, and answers apply only to the input
  - Too specific, and require too much exact knowledge to be useful

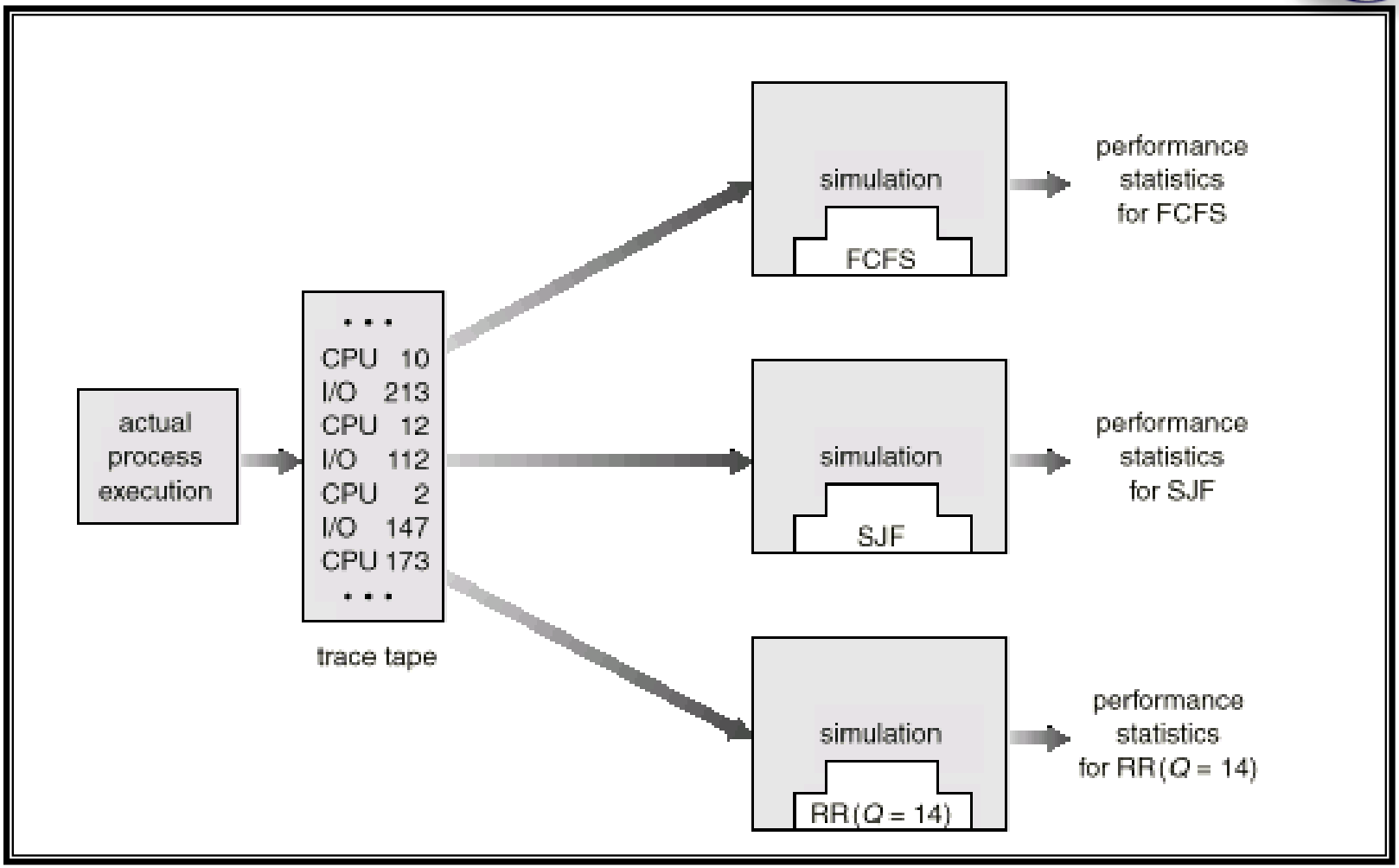
Queueing Models

- Use the distribution of CPU and I/O bursts and distribution of process arrival time… to compute the average throughput, utilization, waiting time…
  - Mathematical and statistical analysis
- Little's formula for a system in a steady sate
  - $n = \lambda * W$ (n: average no. of processes in the system)
    - λ: average arrival rate for new processes in the queue
    - W: average waiting time for a process in the queue
- Can only handle simple algorithms and distributions
- Approximation of a real system – accuracy?

Simulations

- Programming a model of the computer system
  - Use software data structure to model queues, CPU, devices, timers…
  - Simulator modifies the system state to reflect the activities of the devices, CPU, the scheduler, etc.
- Data to drive the simulation
  - Random-number generator according to probability distributions
    - Processes, CPU- and I/O-burst times, arrivals/departures
  - Trace tape – the usage logs of a real system
- Disadvantage – expensive

Evaluation of CPU Schedulers by Simulation



Implementation

- Code a scheduling algorithm, put it in OS, and see…
- Put the actual algorithm in the real system for evaluation under real operating conditions
- Difficulty – cost
  - Reaction of the users to a constantly changing OS

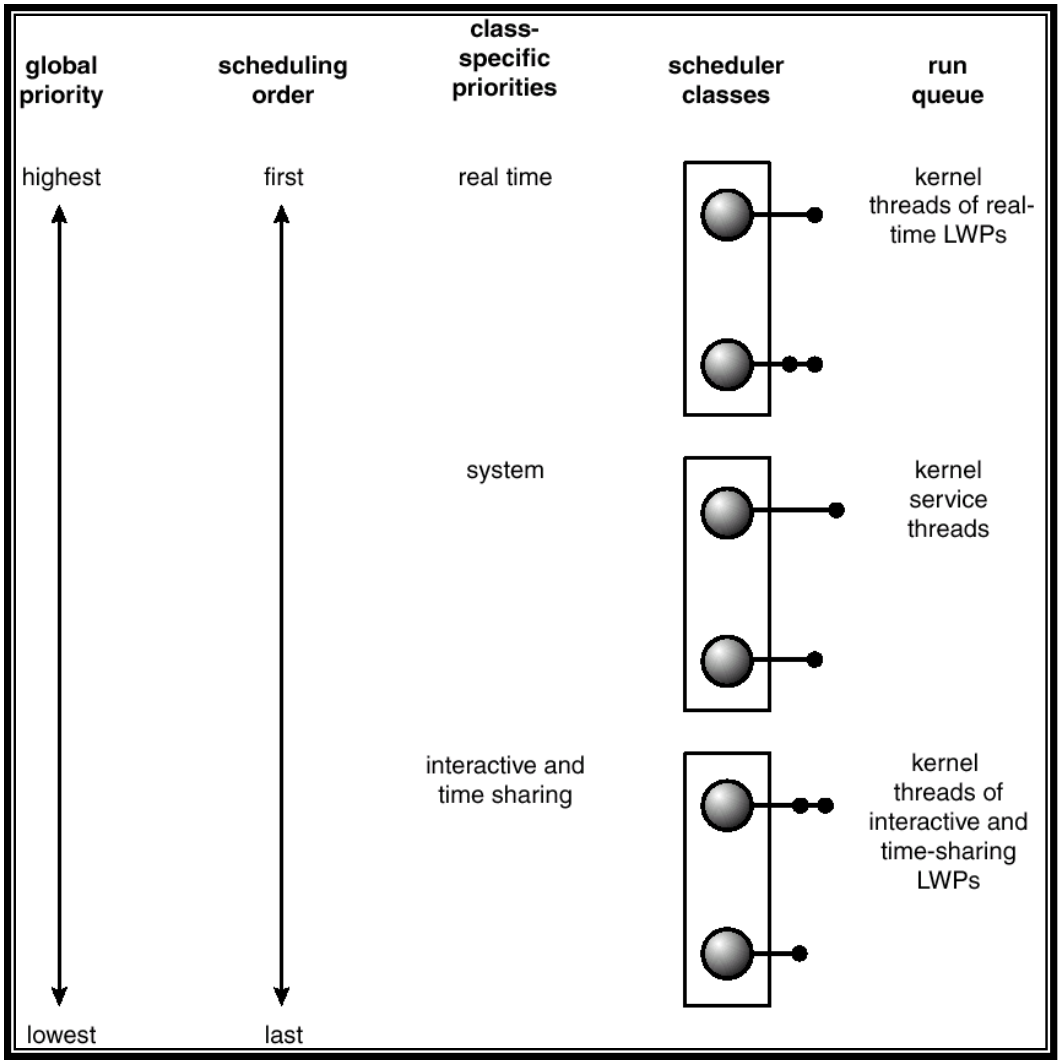
Thread Scheduling

- Process Local Scheduling – How the threads library decides which thread to put onto an available LWP
  - Software-library concern
- System Global Scheduling – How the kernel decides which kernel thread to run next
  - OS concern

Solaris 2 Scheduling

- Priority-based process scheduling
  - Classes: real time, system, time sharing, interactive
    - Each class has different priority and scheduling algorithm
- Each LWP assigns a scheduling class and priority
- Time-sharing/interactive: multilevel feedback queue
- Real-time processes run before a process in any other class
- System class is reserved for kernel use (paging, scheduler)
  - The scheduling policy for the system class does not time-slice
- The selected thread runs on the CPU until it blocks, uses its time slices, or is preempted by a higher-priority thread
  - Multiple threads have the same priority → RR

Solaris 2 Scheduling



Windows 2000 Scheduling

- Priority-based preemptive scheduling
- A running thread will run until it is preempted by a higher-priority one, terminates, time quantum ends, calls a blocking system call
- 32-level priority scheme
  - Variable (1-15) and real-time (16-31) classes, 0 (memory manage)
  - A queue for each priority. Traverses the set of queues from highest to lowest until it finds a thread that is ready to run
  - Run the *idle thread* when no ready thread
  - *Base priority* of each priority class
    - Initial priority for a thread belonging to that class

Windows 2000 Priorities

	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

Windows 2000 Scheduling (Cont.)

- The priority of variable-priority processes will adjust
  - Lower (not below base priority) when its time quantum runs out
  - Priority boosts when it is released from a wait operation
    - The boost level depends on the reason for wait
    - Waiting for keyboard I/O gets a large priority increase
    - Waiting for disk I/O gets a moderate priority increase
  - Process in the foreground window get a higher priority

Linux Scheduling

- Separate Time-sharing and real-time scheduling algorithms
- Allow only processes in user mode to be preempted
  - A process may not be preempted while it is running in kernel mode, even if a real-time process with a higher priority is available to run
  - Soft real-time system
- Time-sharing: Prioritized, credit-based scheduling
  - The process with the most credits is selected
  - A timer interrupt occurs → the running process loses one credit
    - Zero credit ➔ select another process
  - No runnable processes have credits ➔ re-credit ALL processes
    - CREDITS = CREDITS * 0.5 + PRIORITY
  - Priority: real-time > interactive > background

Linux Scheduling (Cont.)

- Real-time scheduling
  - Two real-time scheduling classes: FCFS (non-preemptive) and RR (preemptive)
    - PLUS a priority for each process
  - Always runs the process with the highest priority
    - Equal priority➔runs the process that has been waiting longest