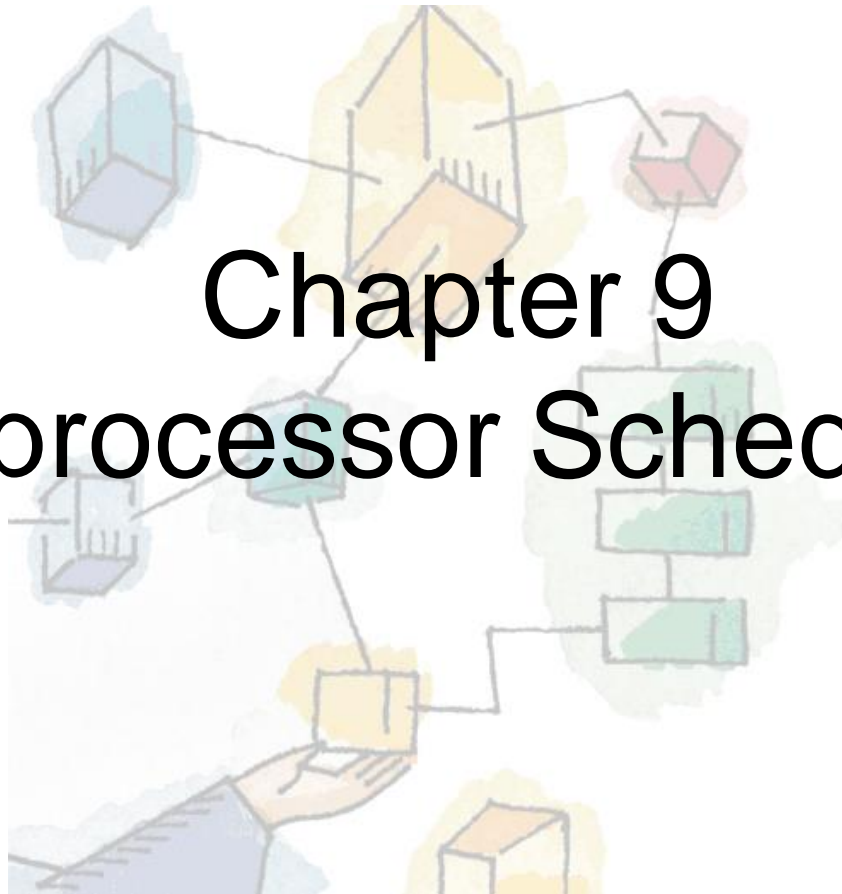


*Operating Systems:  
Internals and Design Principles, 6/E*  
William Stallings

# Chapter 9

## Uniprocessor Scheduling





# Roadmap

## → Types of Processor Scheduling

- Scheduling Algorithms
- Traditional UNIX Scheduling

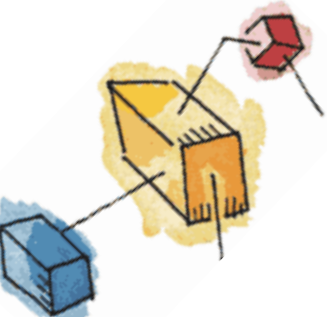




# Scheduling

- An OS must allocate resources amongst competing processes.
- The resource provided by a processor is execution time
  - The resource is allocated by means of a schedule





# Overall Aim of Scheduling

- The aim of processor scheduling is to assign processes to be executed by the processor over time,
  - in a way that meets system objectives, such as response time, throughput, and processor efficiency.

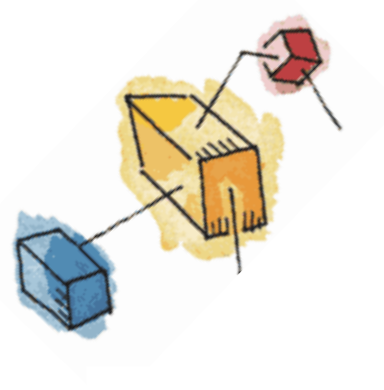




# Scheduling Objectives

- The scheduling function should
  - Share time ***fairly*** among processes
  - Prevent starvation of a process
  - Use the processor efficiently
  - Have low overhead
  - Prioritise processes when necessary (e.g. real time deadlines)





# Types of Scheduling

**Table 9.1 Types of Scheduling**

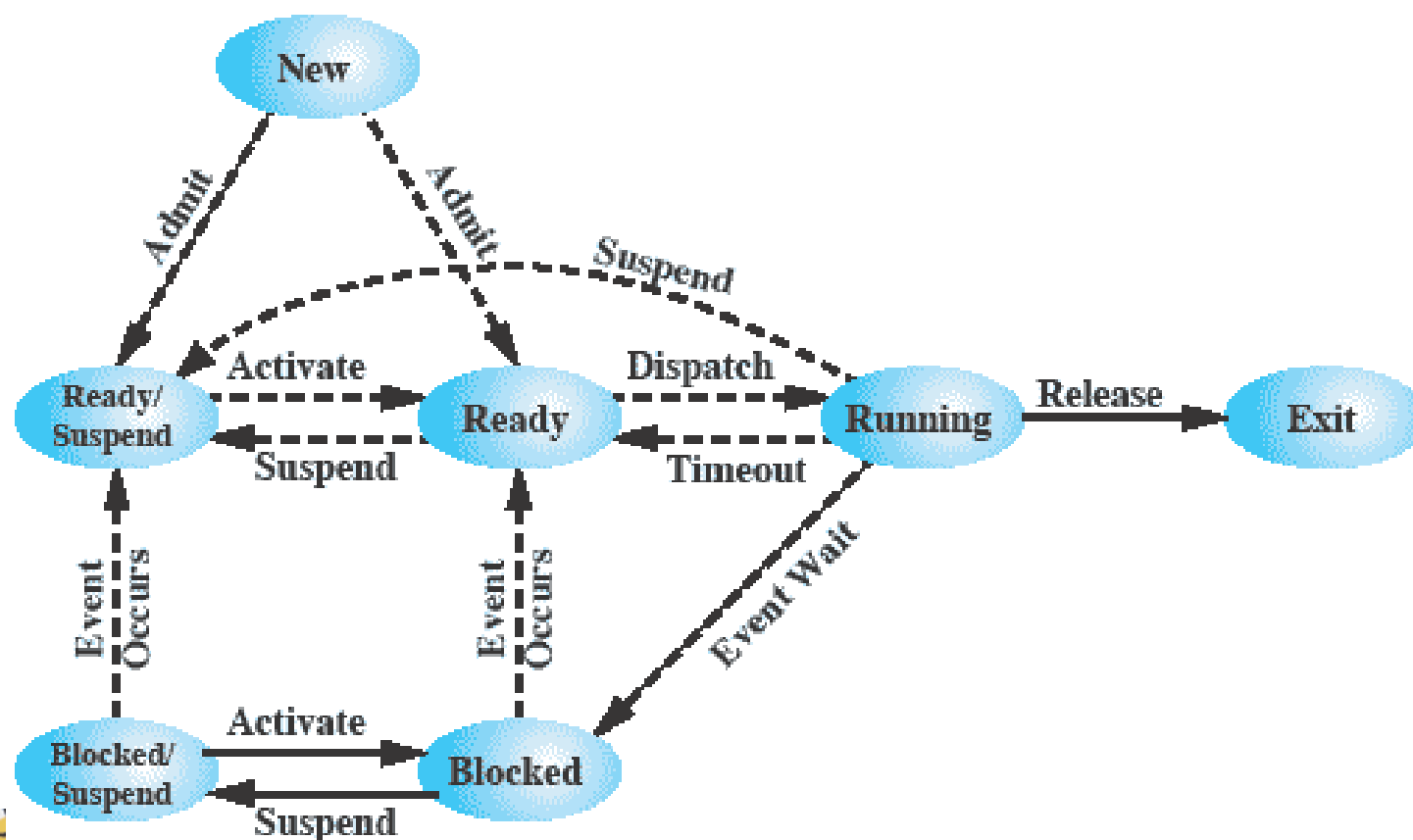
<b>Long-term scheduling</b>	The decision to add to the pool of processes to be executed
<b>Medium-term scheduling</b>	The decision to add to the number of processes that are partially or fully in main memory
<b>Short-term scheduling</b>	The decision as to which available process will be executed by the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





# Two Suspend States

- Remember this diagram from Chapter 3



(b) With Two Suspend States



# Scheduling and Process State Transitions

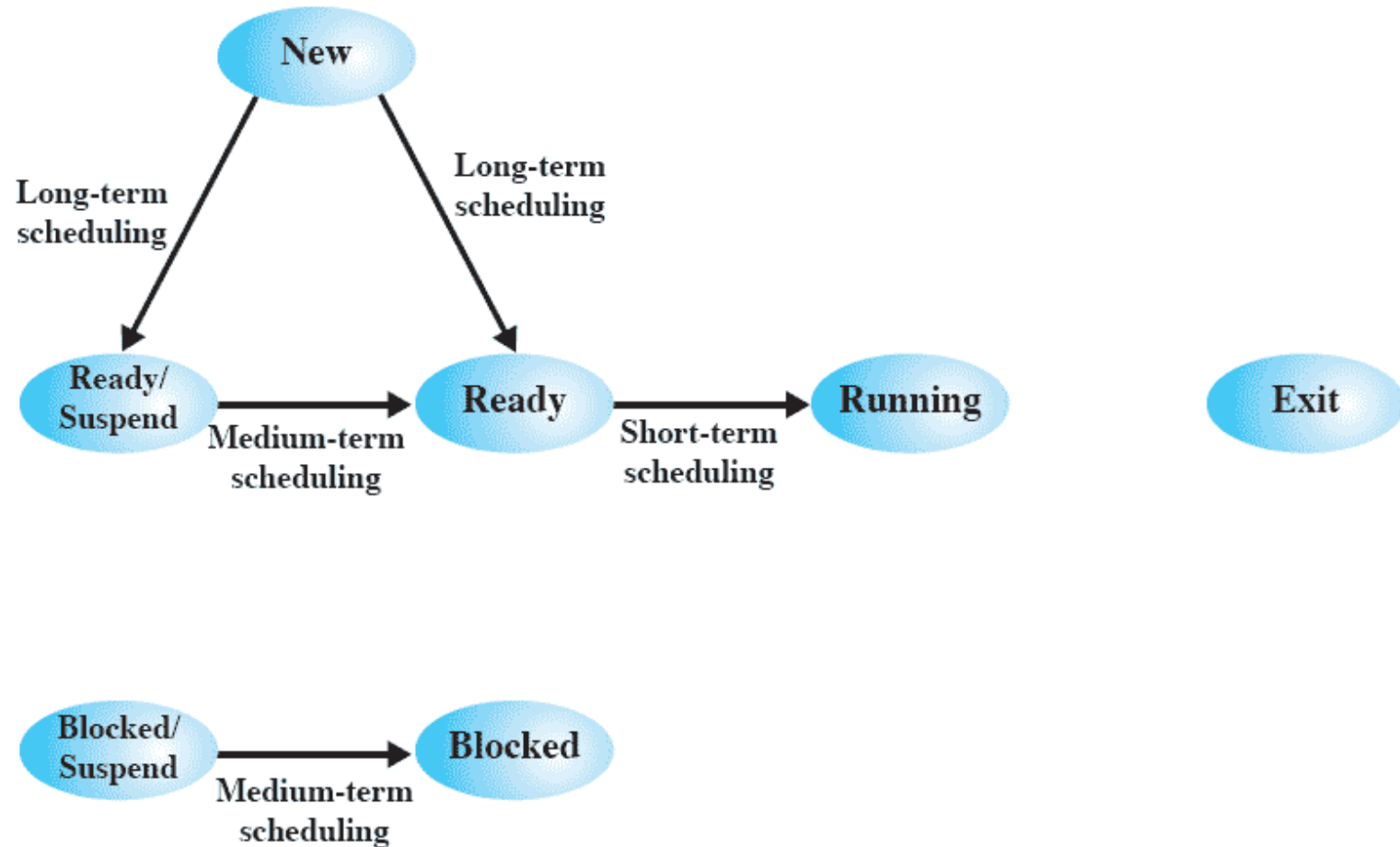
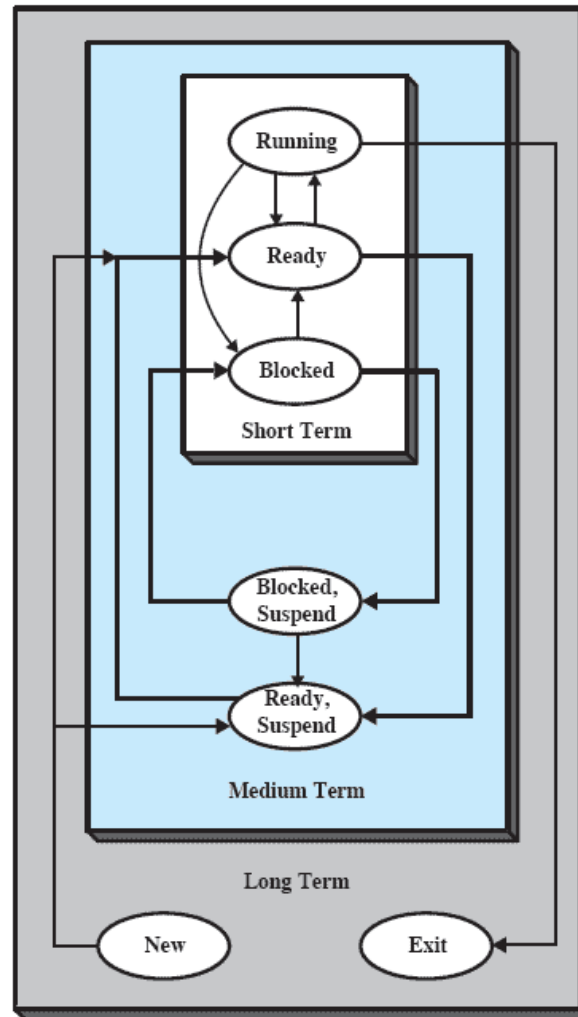


Figure 9.1 Scheduling and Process State Transitions



# Nesting of Scheduling Functions



# Queuing Diagram

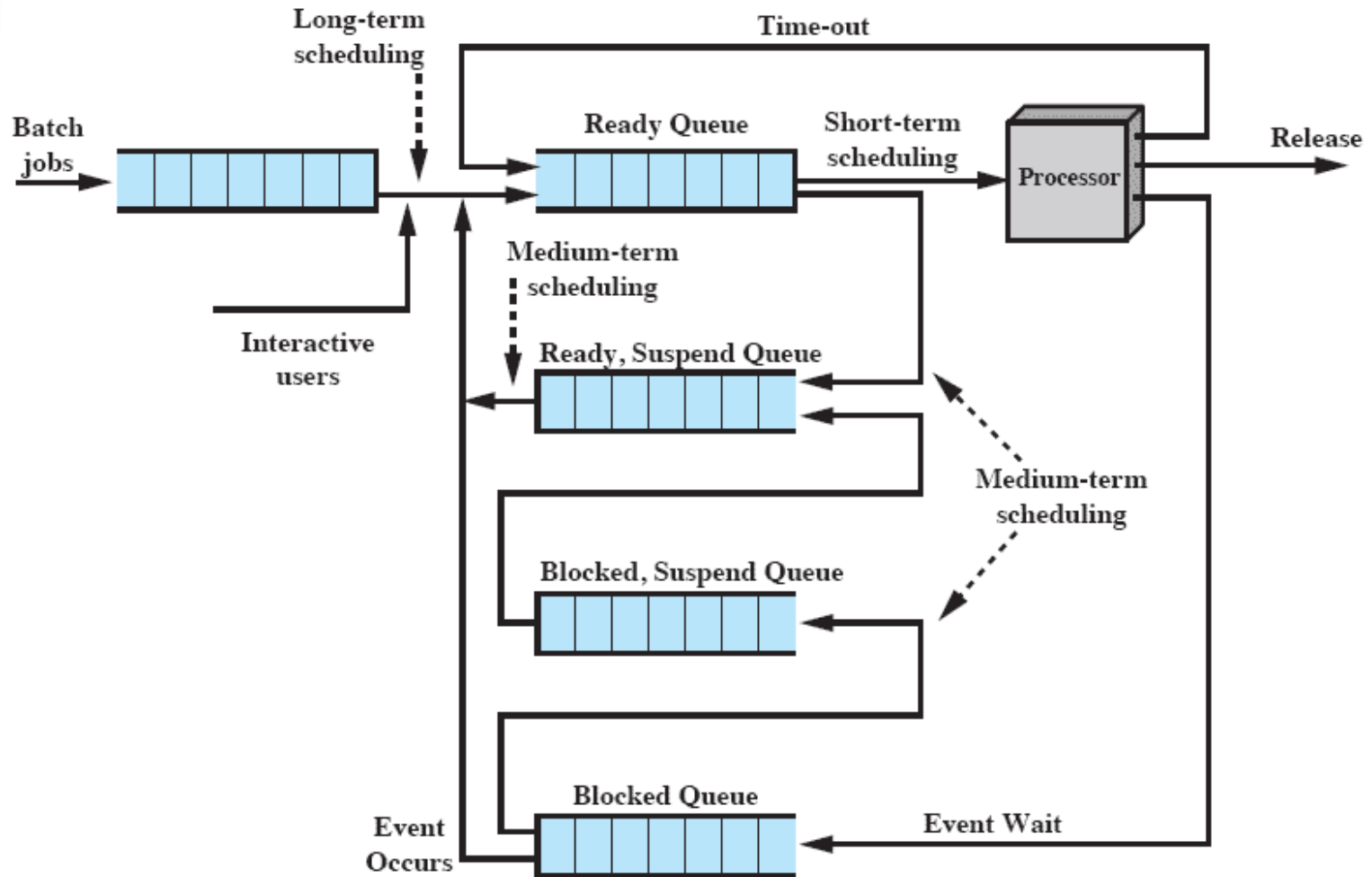


Figure 9.3 Queuing Diagram for Scheduling



# Long-Term Scheduling

- Determines which programs are admitted to the system for processing
  - May be first-come-first-served
  - Or according to criteria such as priority, I/O requirements or expected execution time
- Controls the degree of multiprogramming
- More processes, smaller percentage of time each process is executed



# Medium-Term Scheduling

- Part of the swapping function
- Swapping-in decisions are 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





# Roadmap

- Types of Processor Scheduling

→ Scheduling Algorithms

- Traditional UNIX Scheduling

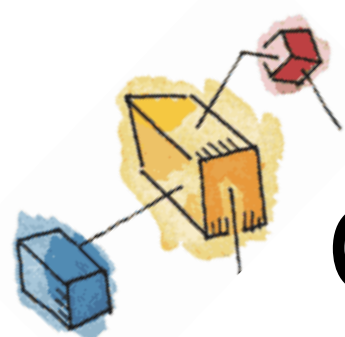




# Aim of Short Term Scheduling

- Main objective is to allocate processor time to optimize certain aspects of system behaviour.
- A set of criteria is needed to evaluate the scheduling policy.





# Short-Term Scheduling

## Criteria: User vs System

- We can differentiate between user and system 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

- We could differentiate between performance related criteria, and those unrelated to performance
- Performance-related
  - Quantitative, easily measured
  - E.g. response time and throughput
- Non-performance related
  - Qualitative
  - Hard to measure





# Interdependent Scheduling Criteria

## User Oriented, Performance Related

**Turnaround time** This is the interval of time between the submission of a process and its completion. Includes actual execution time plus time spent waiting for resources, including the processor. This is an appropriate measure for a batch job.

**Response time** For an interactive process, this is the time from the submission of a request until the response begins to be received. Often a process can begin producing some output to the user while continuing to process the request. Thus, this is a better measure than turnaround time from the user's point of view. The scheduling discipline should attempt to achieve low response time and to maximize the number of interactive users receiving acceptable response time.

**Deadlines** When process completion deadlines can be specified, the scheduling discipline should subordinate other goals to that of maximizing the percentage of deadlines met.

## User Oriented, Other

**Predictability** A given job should run in about the same amount of time and at about the same cost regardless of the load on the system. A wide variation in response time or turnaround time is distracting to users. It may signal a wide swing in system workloads or the need for system tuning to cure instabilities.





# Interdependent Scheduling Criteria cont.

## System Oriented, Performance Related

**Throughput** The scheduling policy should attempt to maximize the number of processes completed per unit of time. This is a measure of how much work is being performed. This clearly depends on the average length of a process but is also influenced by the scheduling policy, which may affect utilization.

**Processor utilization** This is the percentage of time that the processor is busy. For an expensive shared system, this is a significant criterion. In single-user systems and in some other systems, such as real-time systems, this criterion is less important than some of the others.

## System Oriented, Other

**Fairness** In the absence of guidance from the user or other system-supplied guidance, processes should be treated the same, and no process should suffer starvation.

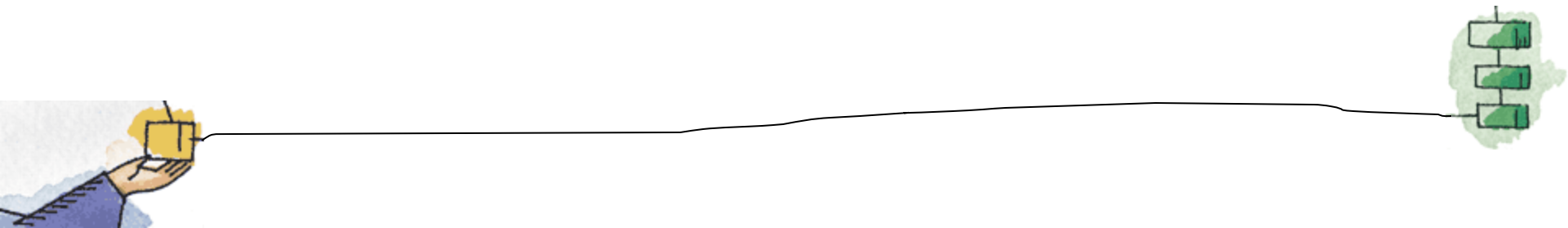
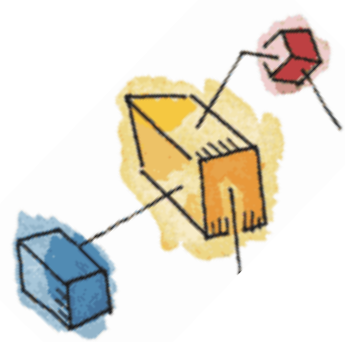
**Enforcing priorities** When processes are assigned priorities, the scheduling policy should favor higher-priority processes.

**Balancing resources** The scheduling policy should keep the resources of the system busy. Processes that will underutilize stressed resources should be favored. This criterion also involves medium-term and long-term scheduling.



# 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



# Priority Queuing

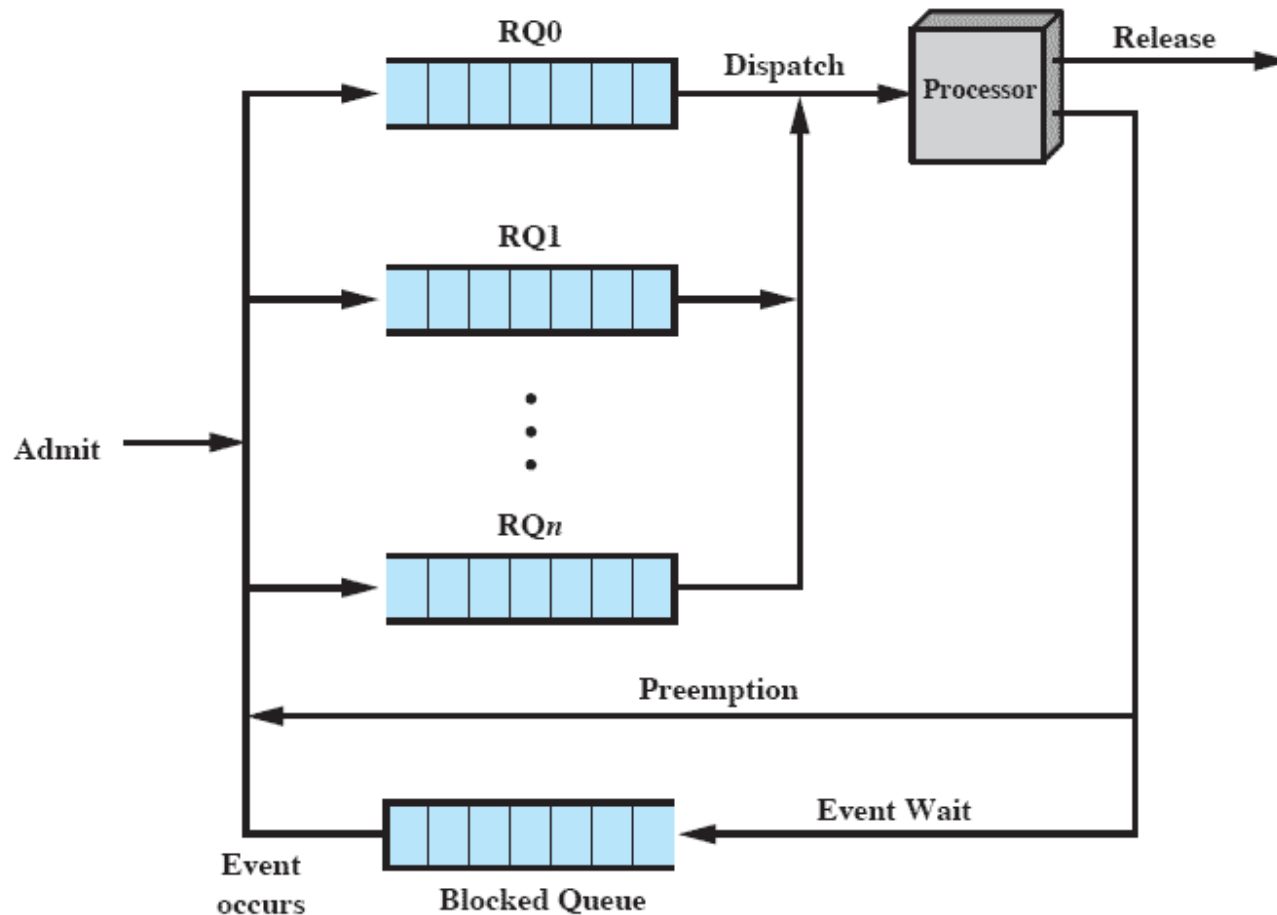


Figure 9.4 Priority Queuing



# Starvation

- Problem:
  - Lower-priority may suffer starvation if there is a steady supply of high priority processes.
- Solution
  - Allow a process to change its priority based on its age or execution history





# Alternative Scheduling Policies

**Table 9.3** Characteristics of Various Scheduling Policies

	FCFS	Round robin	SPN	SRT	HRRN	Feedback
<b>Selection function</b>	$\max[w]$	constant	$\min[s]$	$\min[s - e]$	$\max\left(\frac{w + s}{s}\right)$	(see text)
<b>Decision mode</b>	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
<b>Throughput</b>	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
<b>Response time</b>	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
<b>Overhead</b>	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
<b>Effect on processes</b>	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
<b>Starvation</b>	No	No	Possible	Possible	No	Possible





# Selection Function

- Determines which process is selected for execution
- If based on execution characteristics then important quantities are:
  - $w$  = time spent in system so far, waiting
  - $e$  = time spent in execution so far
  - $s$  = total service time required by the process, including  $e$ ;







# Decision Mode

- Specifies the instants in time at which the selection function is exercised.
- Two categories:
  - Nonpreemptive
  - Preemptive





# Nonpreemptive vs Preemptive

- Non-preemptive
  - 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 ready state by the OS
  - Preemption may occur when new process arrives, on an interrupt, or periodically.



# Process Scheduling Example

- Example set of processes, consider each a batch job

**Table 9.4 Process Scheduling Example**

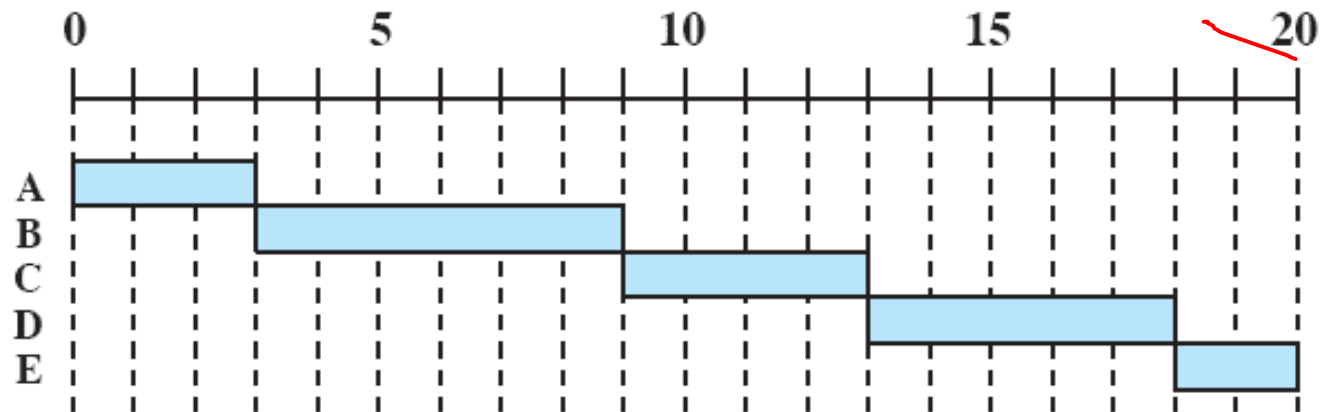
Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

– Service time represents total execution time

# First-Come-First-Served

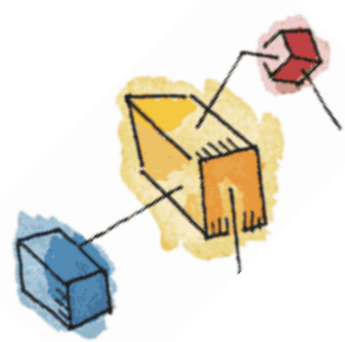
- Each process joins the Ready queue
- When the current process ceases to execute, the longest process in the Ready queue is selected

First-Come-First  
Served (FCFS)



# First-Come-First-Served

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

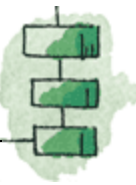
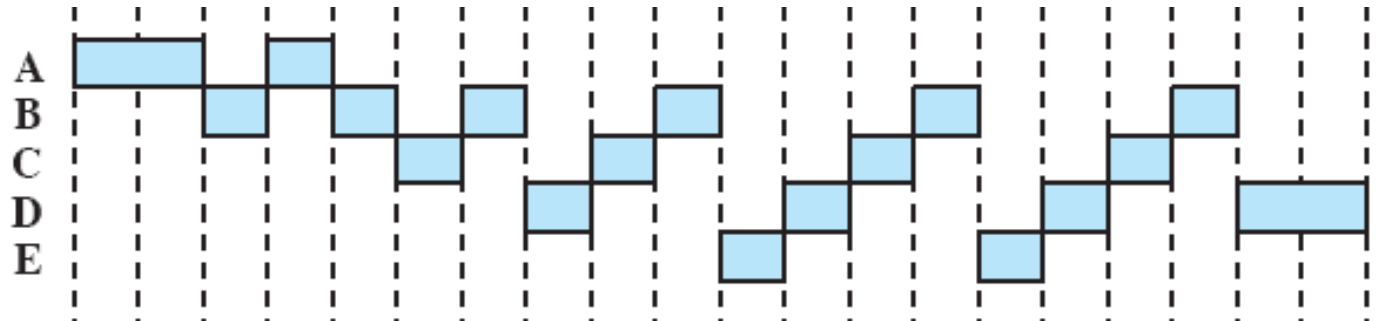




# Round Robin

- Uses preemption based on a clock
  - also known as time slicing, because each process is given a slice of time before being preempted.

Round-Robin  
(RR),  $q = 1$



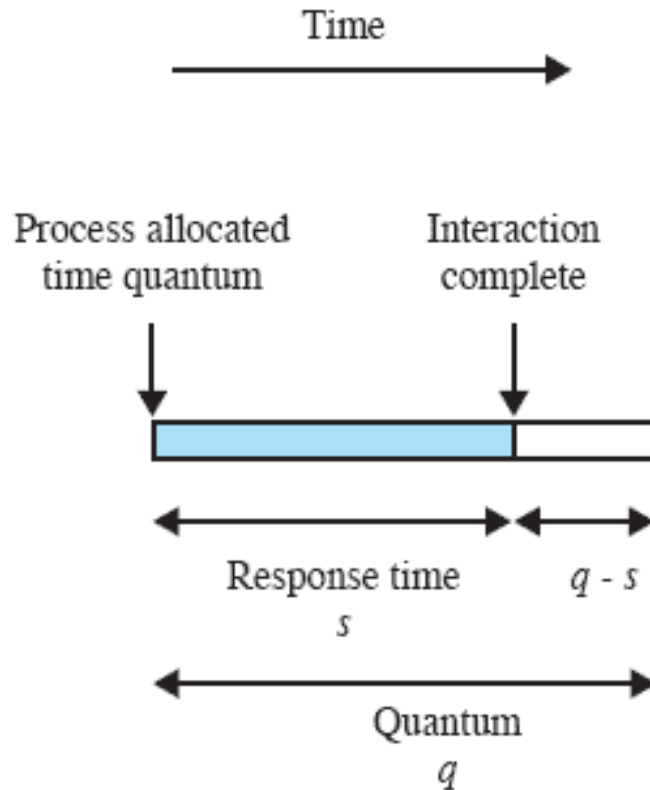


# Round Robin

- Clock interrupt is generated at periodic intervals
- When an interrupt occurs, the currently running process is placed in the ready queue
  - Next ready job is selected



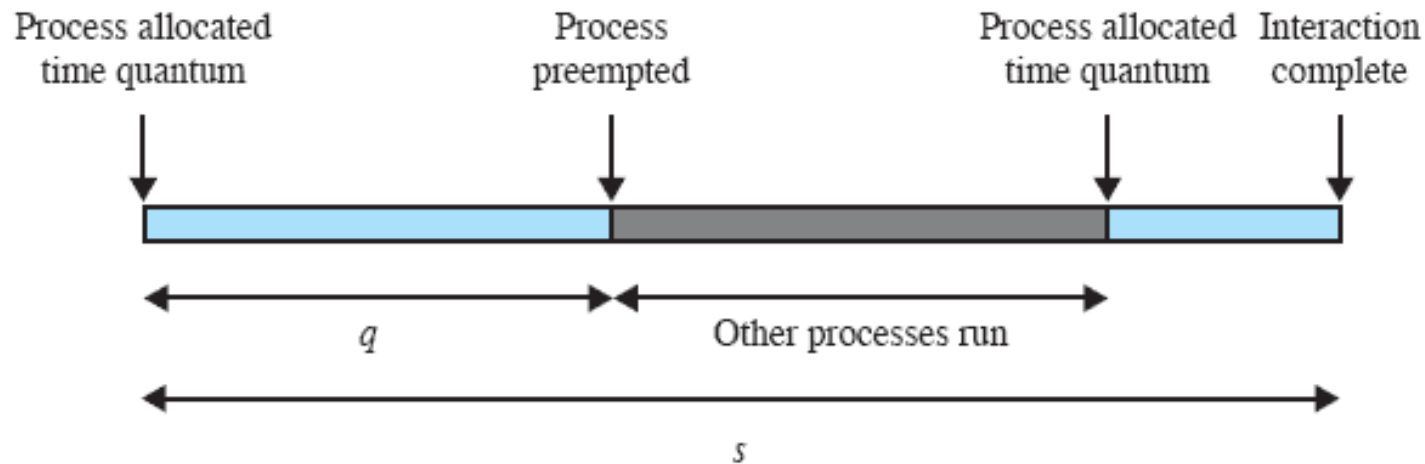
# Effect of Size of Preemption Time Quantum



(a) Time quantum greater than typical interaction



# Effect of Size of Preemption Time Quantum



(b) Time quantum less than typical interaction

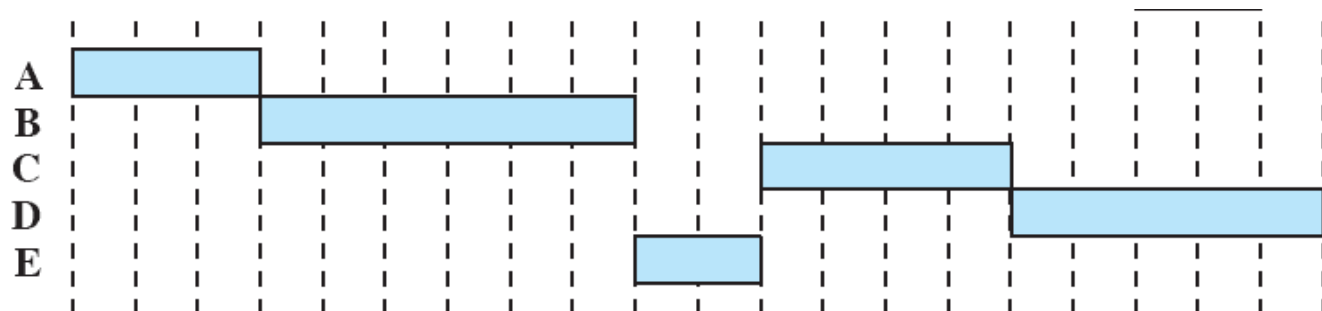
**Figure 9.6 Effect of Size of Preemption Time Quantum**



# Shortest Process Next

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

Shortest Process  
Next (SPN)





# Shortest Process Next

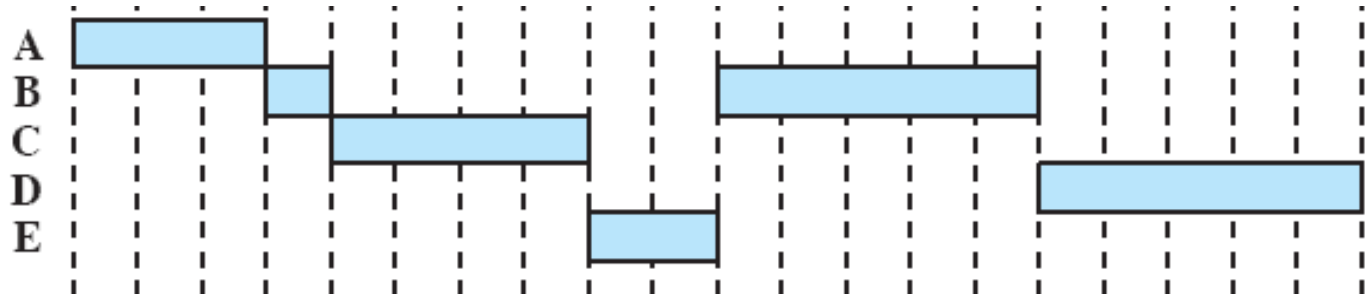
- Predictability of longer processes is reduced
- If estimated time for process not correct, the operating system may abort it
- Possibility of starvation for longer processes



# Shortest Remaining Time

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

Shortest Remaining Time (SRT)



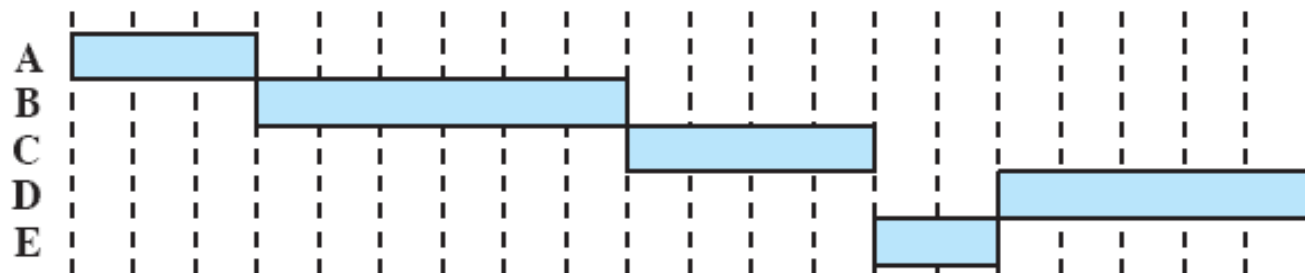


# Highest Response Ratio Next

- Choose next process with the greatest ratio

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

Highest Response Ratio Next (HRRN)





# Feedback Scheduling

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

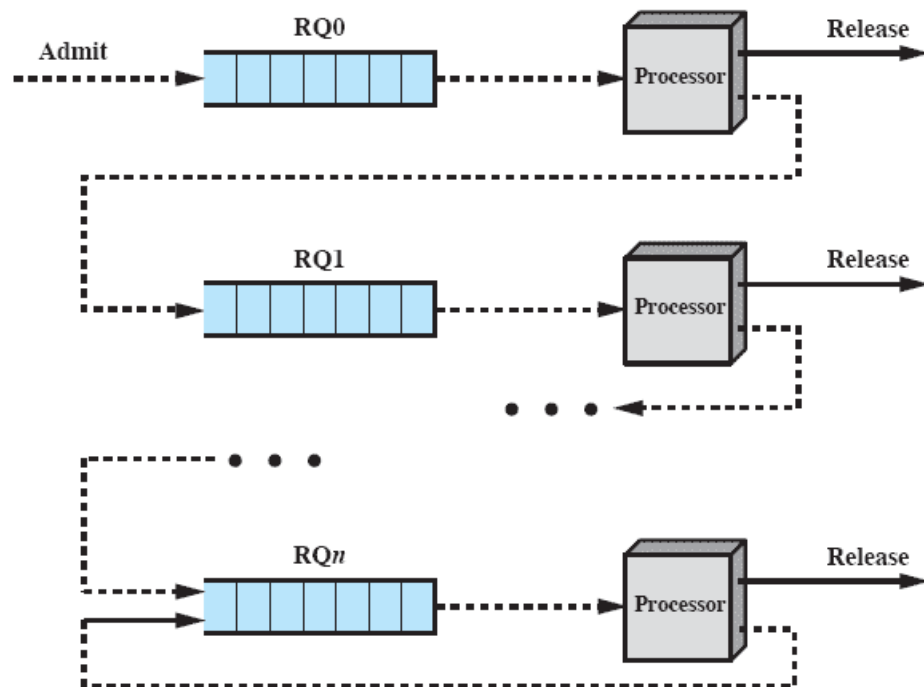


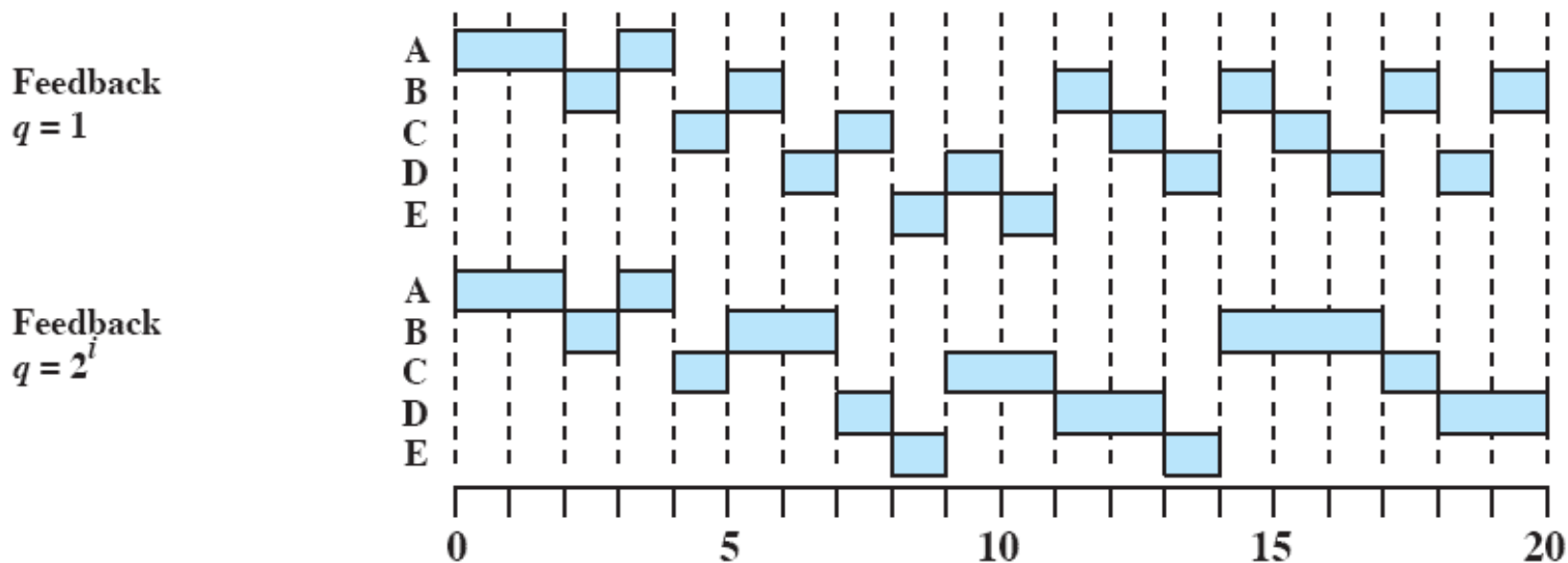
Figure 9.10 Feedback Scheduling





# Feedback Performance

- Variations exist, simple version pre-empts periodically, similar to round robin
  - But can lead to starvation



# Performance Comparison

- Any scheduling discipline that chooses the next item to be served independent of service time obeys the relationship:

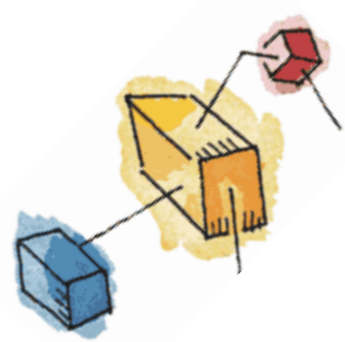
$$\frac{T_r}{T_s} = \frac{1}{1 - \rho}$$

where

$T_r$  = turnaround time or residence time; total time in system, waiting plus execution

$T_s$  = average service time; average time spent in Running state

$\rho$  = processor utilization





# Formulas

**Table 9.6 Formulas for Single-Server Queues with Two Priority Categories**

- Assumptions:
1. Poisson arrival rate.
  2. Priority 1 items are serviced before priority 2 items.
  3. First-come-first-served dispatching for items of equal priority.
  4. No item is interrupted while being served.
  5. No items leave the queue (lost calls delayed).

## (a) General formulas

$$\begin{aligned}\lambda &= \lambda_1 + \lambda_2 \\ \rho_1 &= \lambda_1 T_{s1}; \quad \rho_2 = \lambda_2 T_{s2} \\ \rho &= \rho_1 + \rho_2 \\ T_s &= \frac{\lambda_1}{\lambda} T_{s1} + \frac{\lambda_2}{\lambda} T_{s2} \\ T_r &= \frac{\lambda_1}{\lambda} T_{r1} + \frac{\lambda_2}{\lambda} T_{r2}\end{aligned}$$

## b) No interrupts; exponential service times

$$\begin{aligned}T_{r1} &= T_{s1} + \frac{\rho_1 T_{s1} + \rho_2 T_{s2}}{1 - \rho_1} \\ T_{r2} &= T_{s2} + \frac{T_{r1} - T_{s1}}{1 - \rho}\end{aligned}$$

## (c) Preemptive-resume queuing discipline; exponential service times

$$\begin{aligned}T_{r1} &= T_{s1} + \frac{\rho_1 T_{s1}}{1 - \rho_1} \\ T_{r2} &= T_{s2} + \frac{1}{1 - \rho_1} \left( \rho_1 T_{s2} + \frac{\rho T_s}{1 - \rho} \right)\end{aligned}$$

# Overall Normalized Response Time

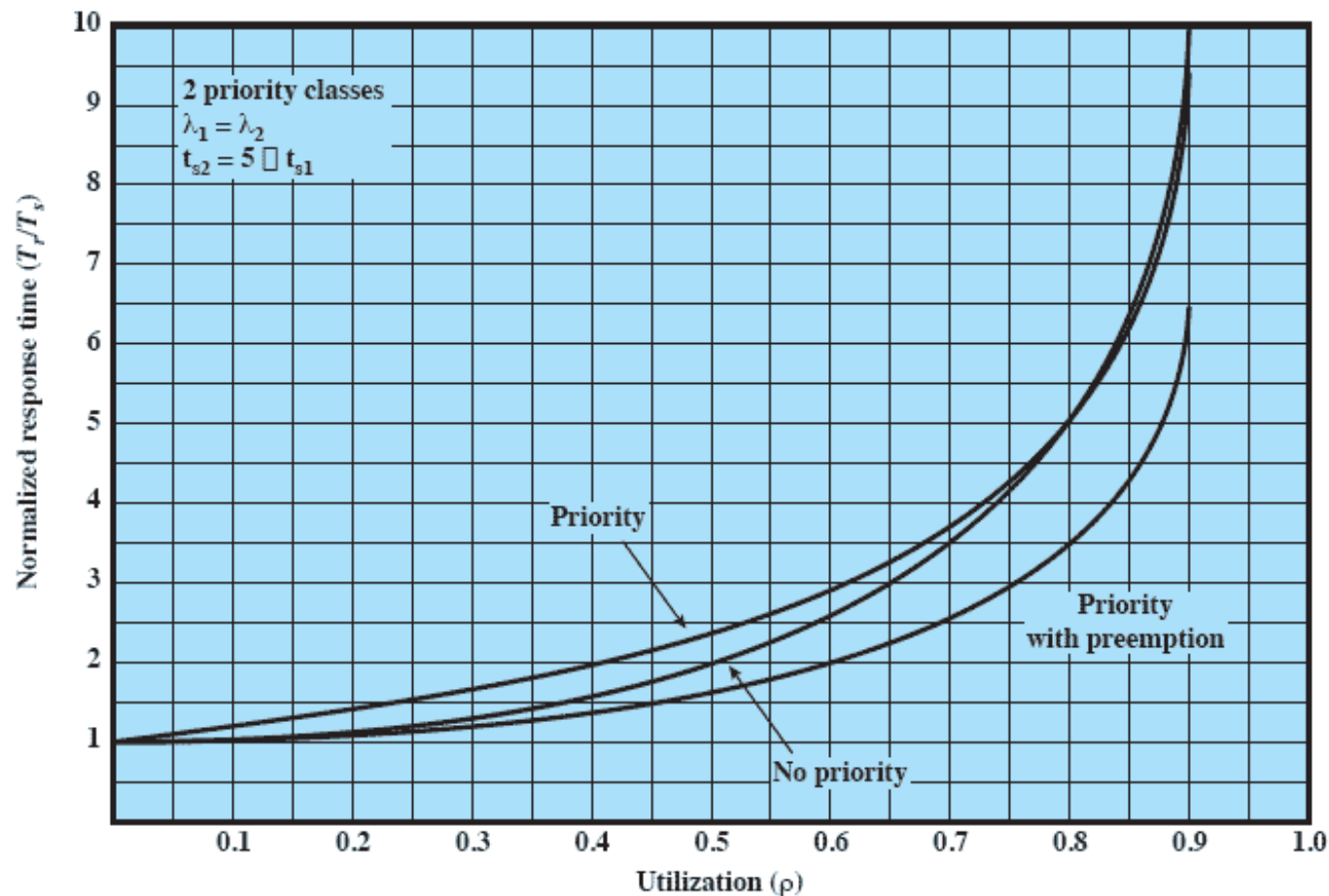


Figure 9.11 Overall Normalized Response Time

# Normalized Response Time for Shorter Process

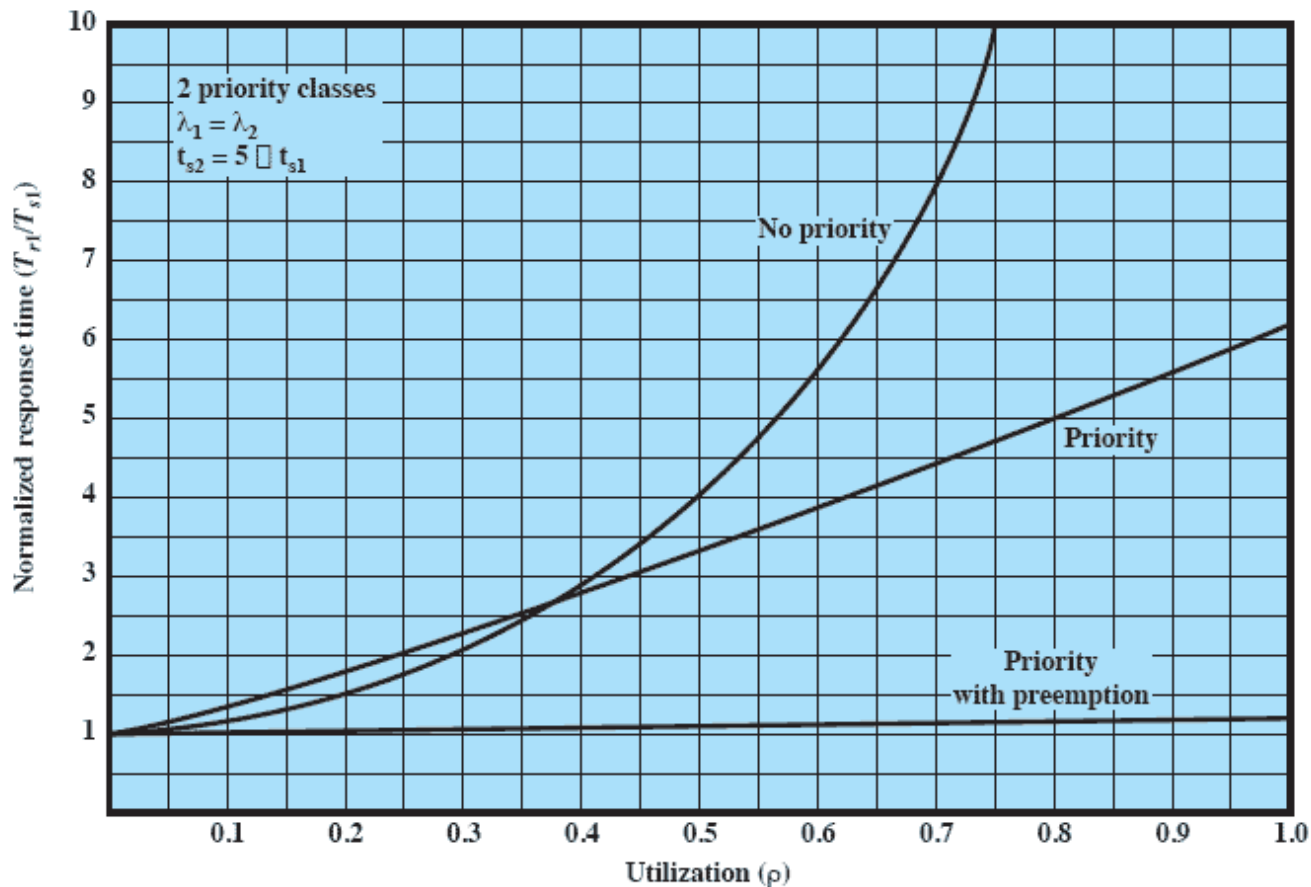


Figure 9.12 Normalized Response Time for Shorter Processes

# Normalized Response Time for Longer Processes

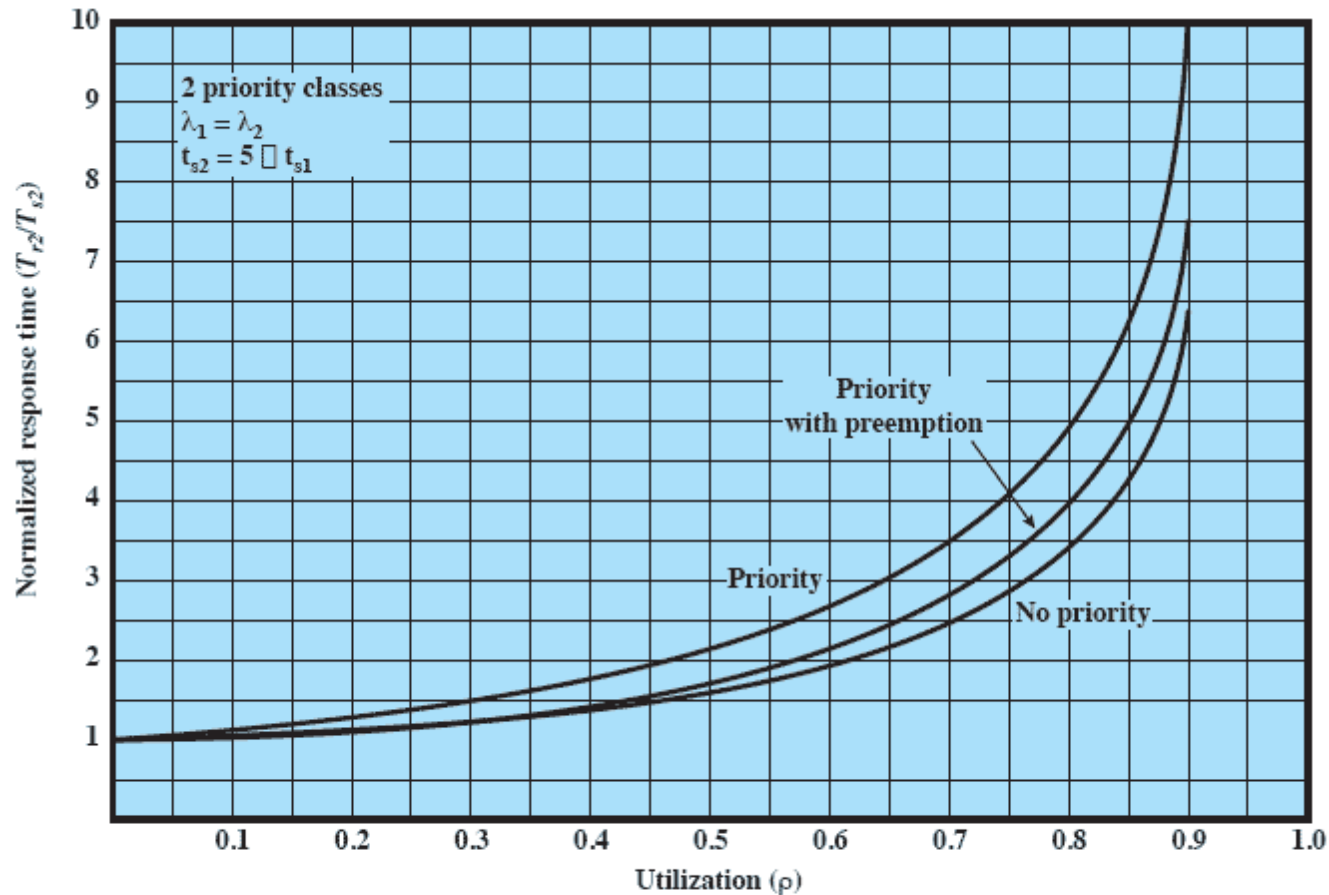


Figure 9.13 Normalized Response Time for Longer Processes

# Normalized Turnaround Time

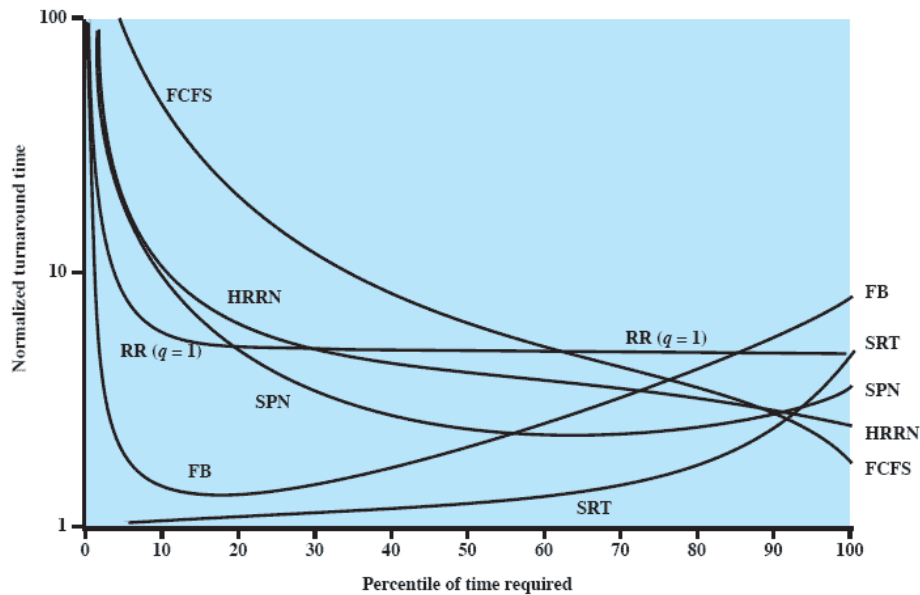


Figure 9.14 Simulation Results for Normalized Turnaround Time

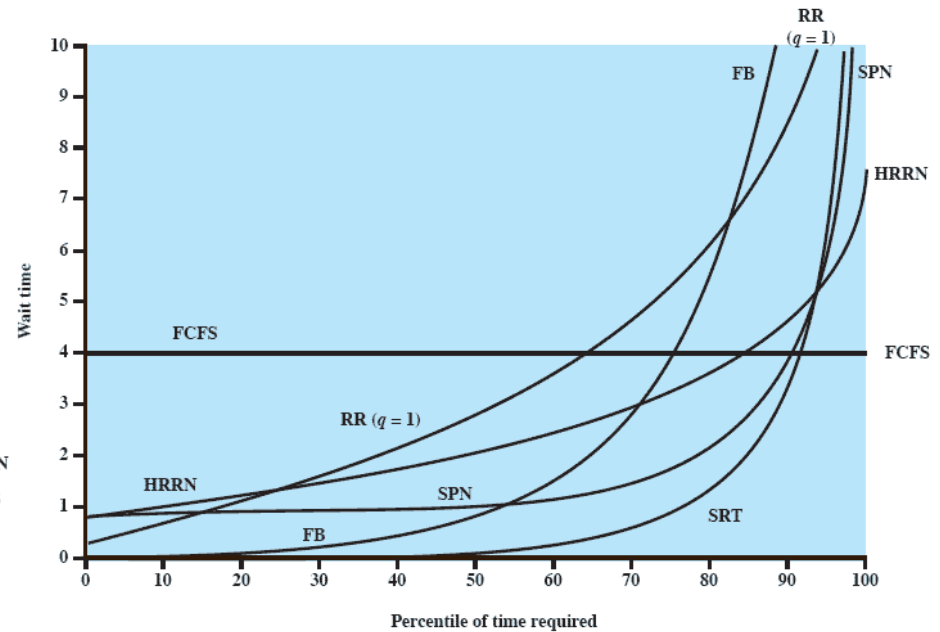


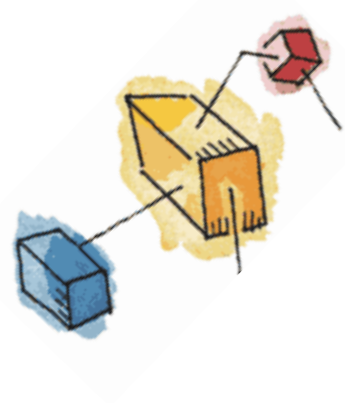
Figure 9.15 Simulation Results for Waiting Time



# 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





# Fair-Share Scheduler

Time	Process A			Process B			Process C		
	Priority	Process CPU count	Group CPU count	Priority	Process CPU count	Group CPU count	Priority	Process CPU count	Group CPU count
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

Colored rectangle represents executing process

Figure 9.16 Example of Fair Share Scheduler — Three Processes, Two Groups





# Roadmap

- Types of Processor Scheduling
- Scheduling Algorithms

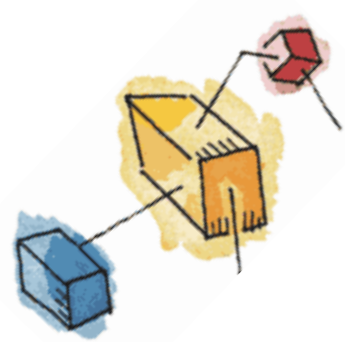
→ Traditional UNIX Scheduling

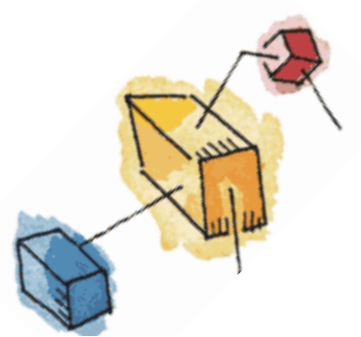




# Traditional UNIX Scheduling

- Multilevel feedback using round robin within each of the priority queues
- If a running process does not block or complete within 1 second, it is preempted
- Priority is based on process type and execution history.





# Scheduling Formula

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + nice_j$$

where

$CPU_j(i)$  = measure of processor utilization by process  $j$  through interval  $i$

$P_j(i)$  = priority of process  $j$  at beginning of interval  $i$ ; lower values equal higher priorities

$Base_j$  = base priority of process  $j$

$nice_j$  = user-controllable adjustment factor





# Bands

- Priorities are recomputed once per second
- Base priority divides all processes into fixed bands of priority levels
  - Swapper (highest)
  - Block I/O device control
  - File manipulation
  - Character I/O device control
  - User processes (lowest)



# Example of Traditional UNIX Process Scheduling

Time	Process A		Process B		Process C	
	Priority	CPU count	Priority	CPU count	Priority	CPU count
0	60	0 1 2 . . 60	60	0	60	0
1	75	30	60 1 2 . . 60	0	60	0
2	67	15	75	30	60 1 2 . . 60	0
3	63	7 8 9 . . 67	67	15	75	30
4	76	33	63 7 8 9 . . 67	7	67	15
5	68	16	76	33	63	7

Colored rectangle represents executing process

Figure 9.17 Example of Traditional UNIX Process Scheduling