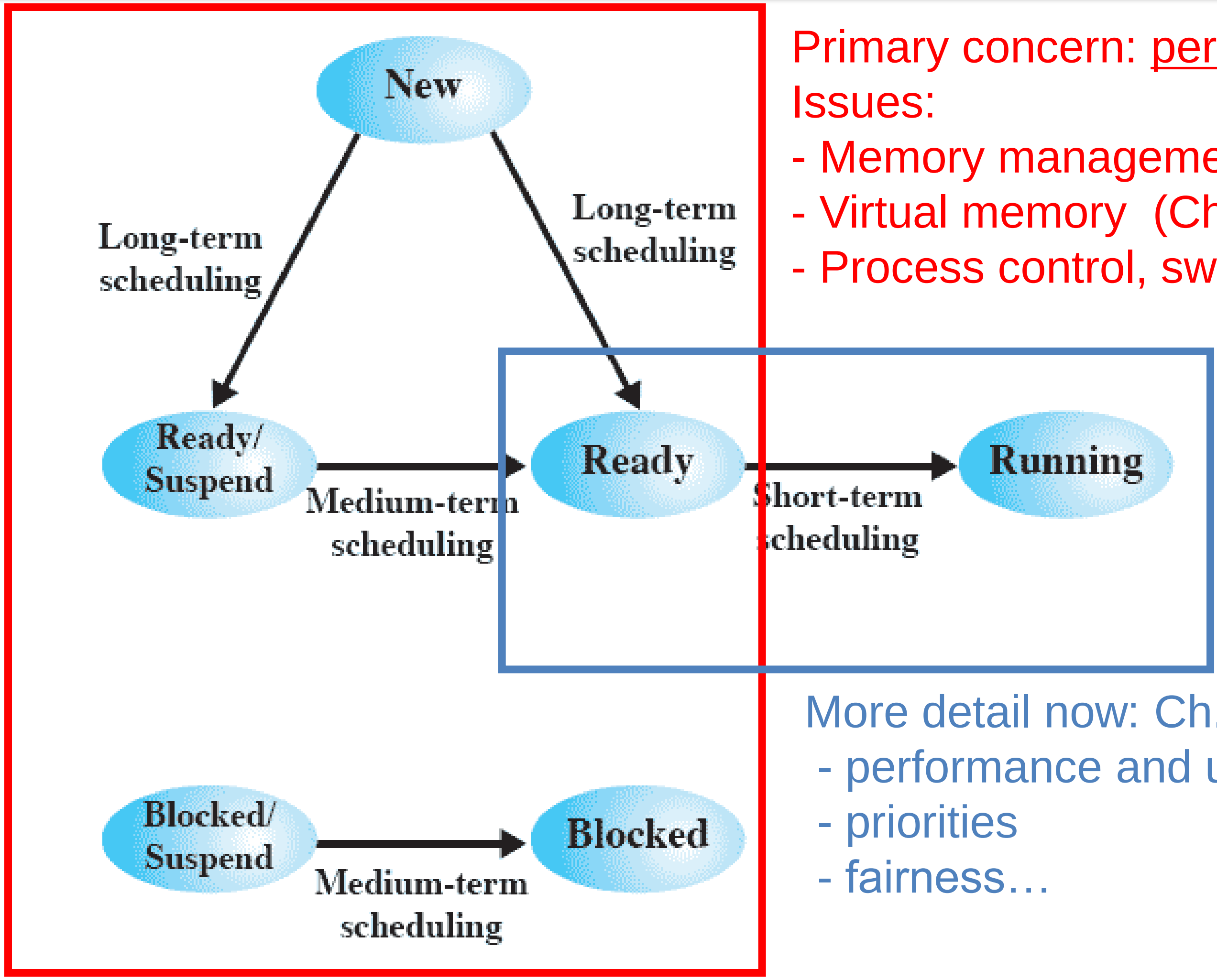
Chapter 9
Uniprocessor Scheduling

(based on original slides by Pearson)

Aim of Processor Scheduling

- Assign processes to be executed by the processor(s)
- Affect the performance, by determining which process will wait and which will progress
- Scheduler must meet objectives
  - Response time
  - Throughput
  - Processor efficiency
  - Temperature
  - Power

Scheduling and Process State Transitions



Primary concern: performance & I/O

Issues:

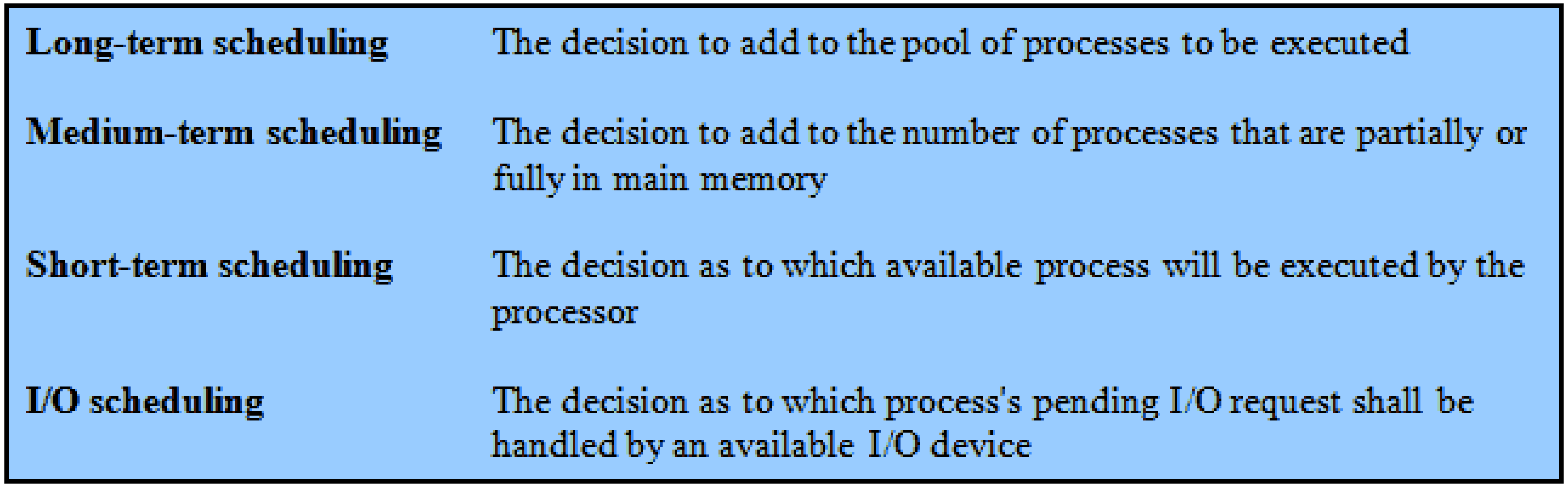
- Memory management (Ch. 7)
- Virtual memory (Ch. 8)
- Process control, swapping (Ch. 3)

More detail now: Ch. 3 was quite simplistic

- performance and user-orientation
- priorities
- fairness…

Figure 9.1 Scheduling and Process State Transitions

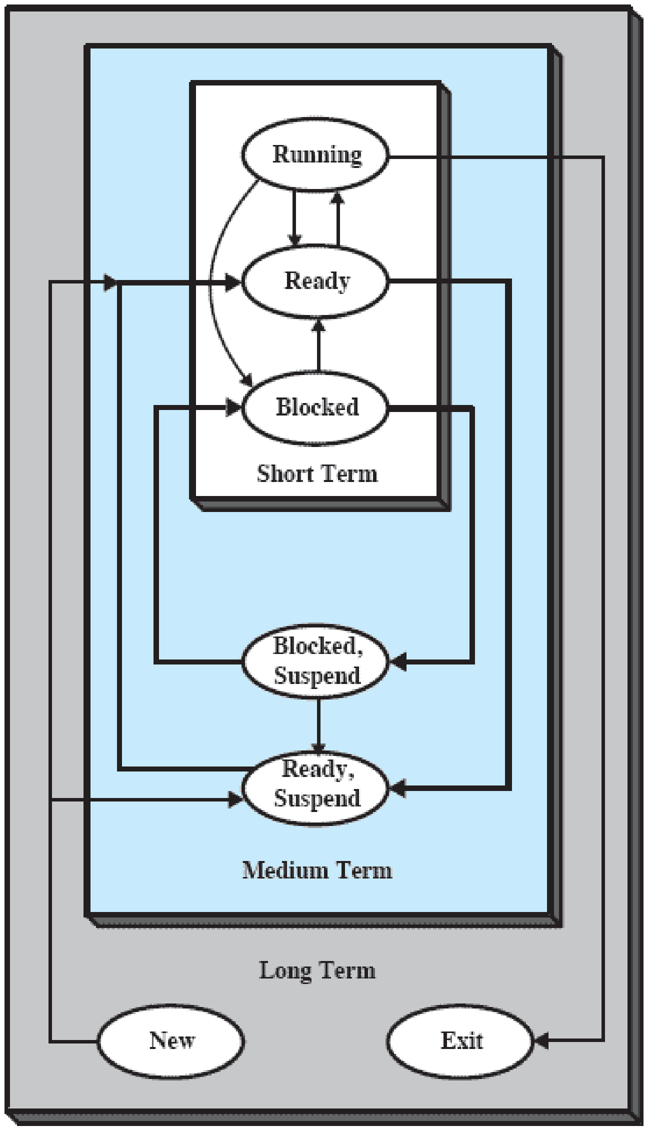
Types of Scheduling



Long-term scheduling	The decision to add to the pool of processes to be executed
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory
Short-term scheduling	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

Proceeds from coarse-grained to fine grained from top to bottom.

Levels of Scheduling



Queuing Diagram

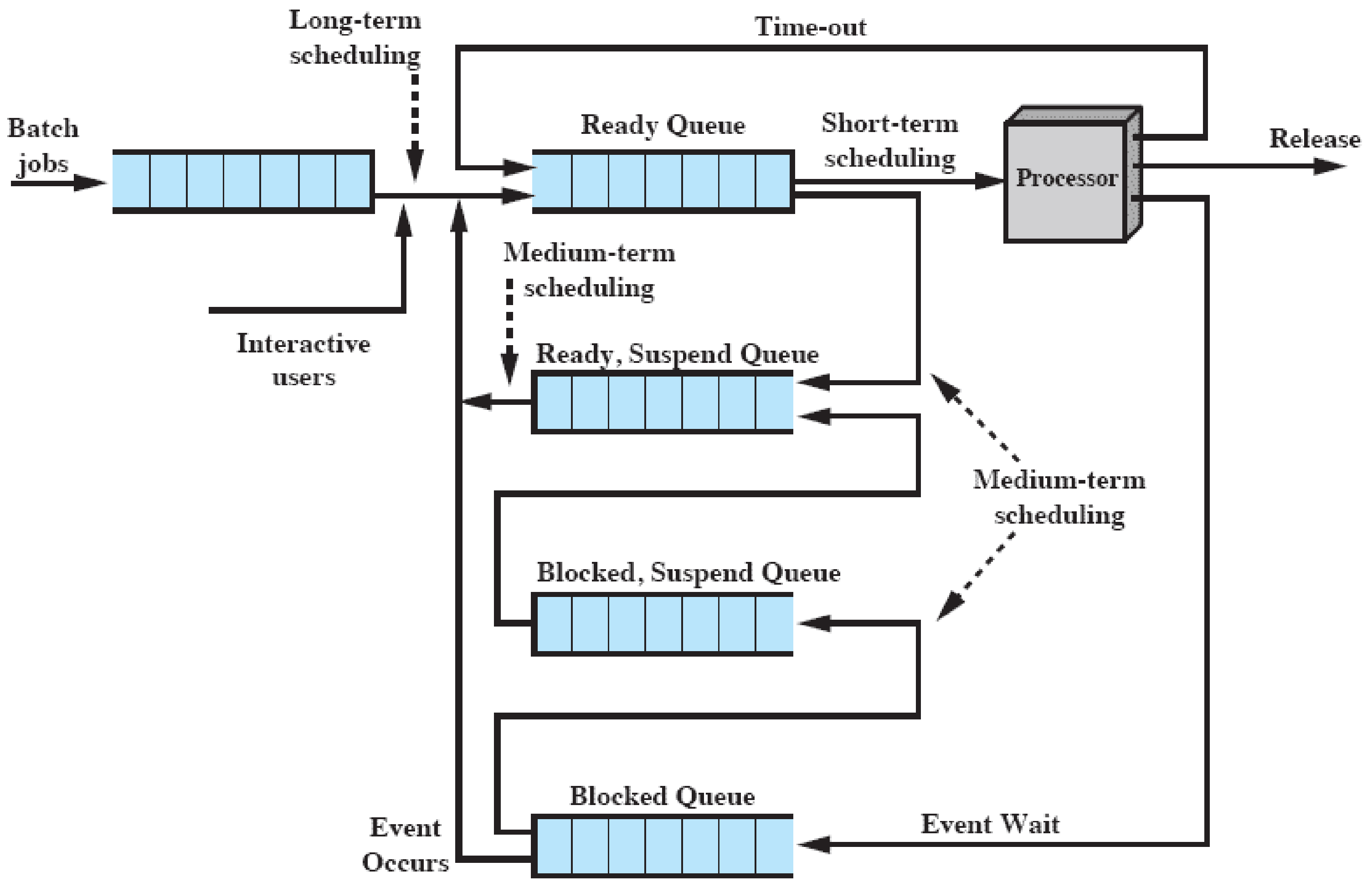


Figure 9.3 Queuing Diagram for Scheduling

Long-Term Scheduling

- Control program admission, thereby controls the level of multiprogramming
- Two questions for the scheduler:
  - Can it take on another process? (metrics, requirements)
  - Which process should it take on? (FIFO, priority, execution time, I/O requirements, …)
- More processes, smaller percentage of time each process is executed

Medium-Term Scheduling

- Part of the swapping function
- Depends on the availability of virtual memory
- 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, for example
  - Clock interrupts
  - I/O interrupts
  - Operating system calls
  - Signals

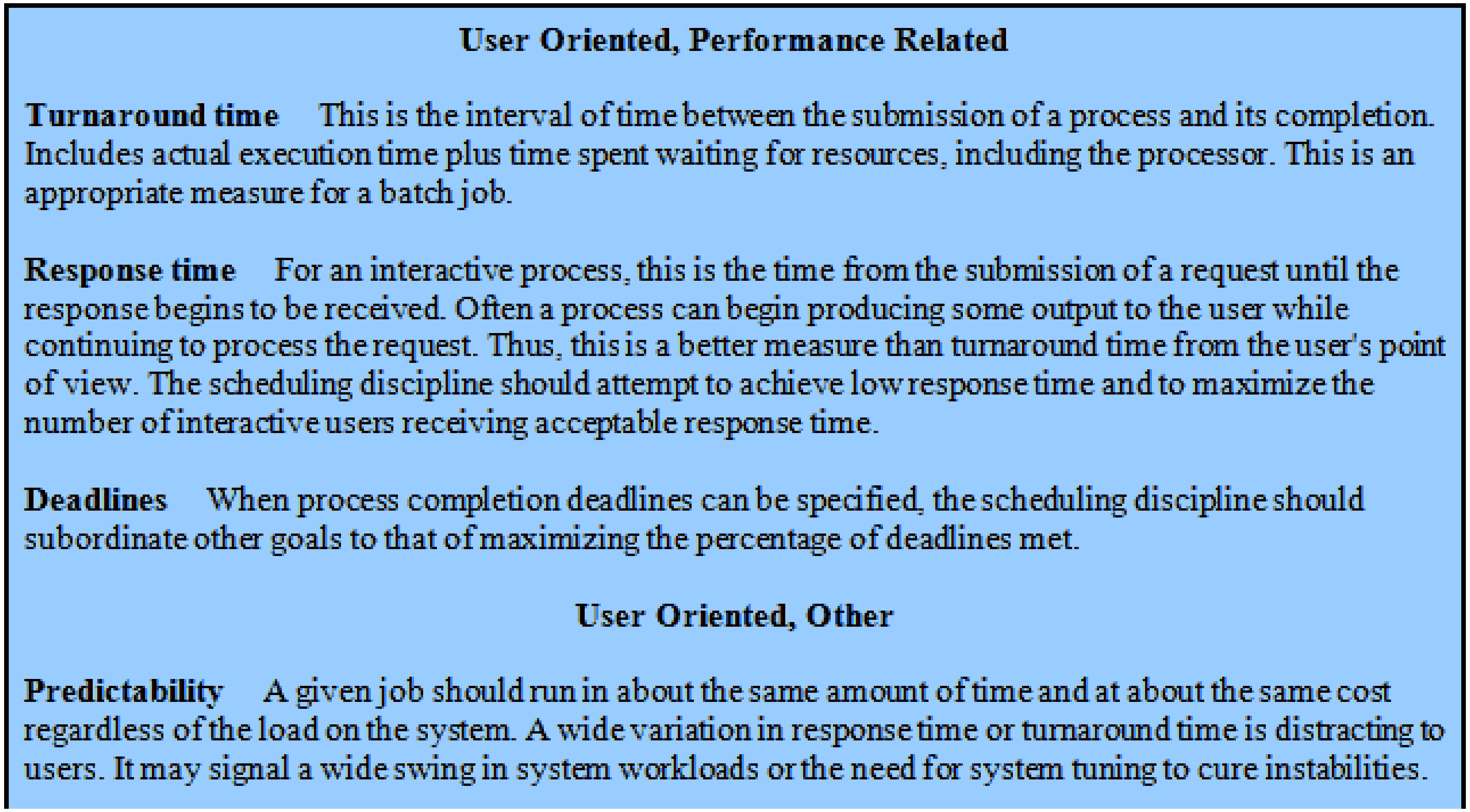
Short-Term Scheduling Criteria

- User-oriented vs system-oriented criteria

- UO: What is good for the user?
- SO: What is good for the system?

- Requires quantitative, measurable metrics

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.

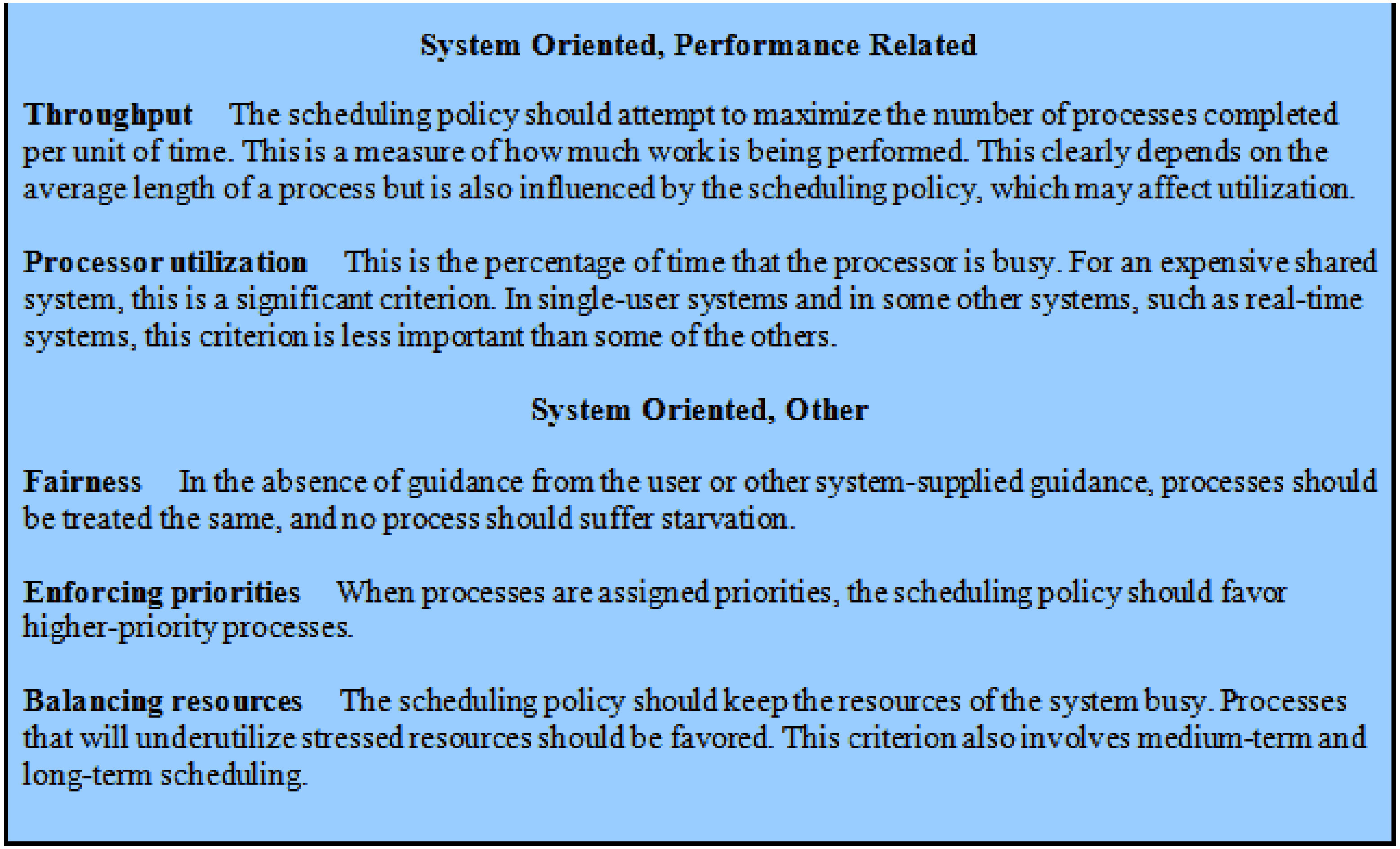
Response time For an interactive process, this is the time from the submission of a request until the response begins to be received. Often a process can begin producing some output to the user while continuing to process the request. Thus, this is a better measure than turnaround time from the user's 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.

Scheduling Criteria



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
- Use multiple ready queues to represent multiple levels of priority
- Lower-priority may suffer starvation
  - Allow a process to change its priority based on its age or execution history

Priority Queuing

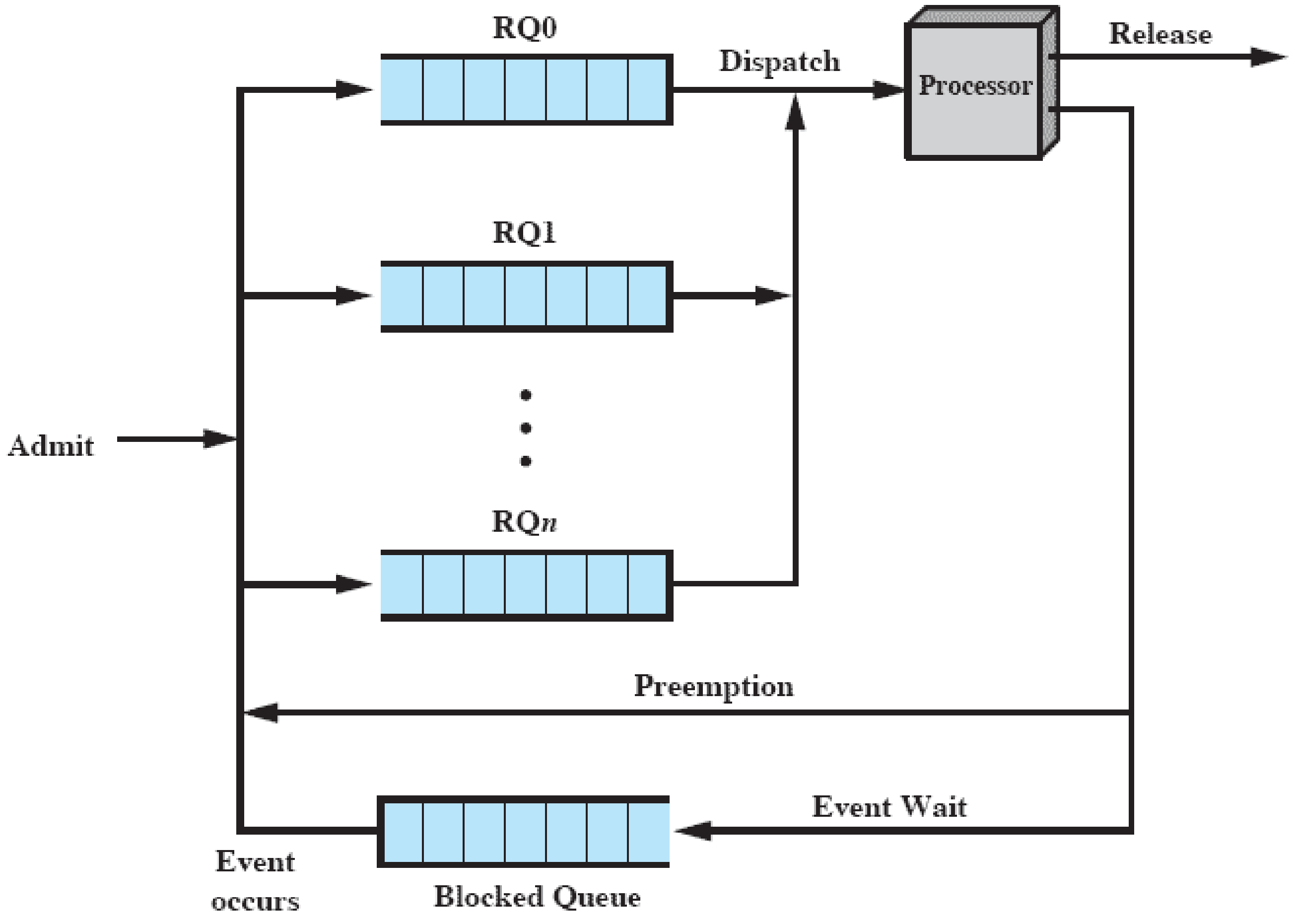


Figure 9.4 Priority Queuing

Decision Mode For Multiprogramming

- 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
- Cooperative
  - Developer programs the context switch

Process Scheduling Example

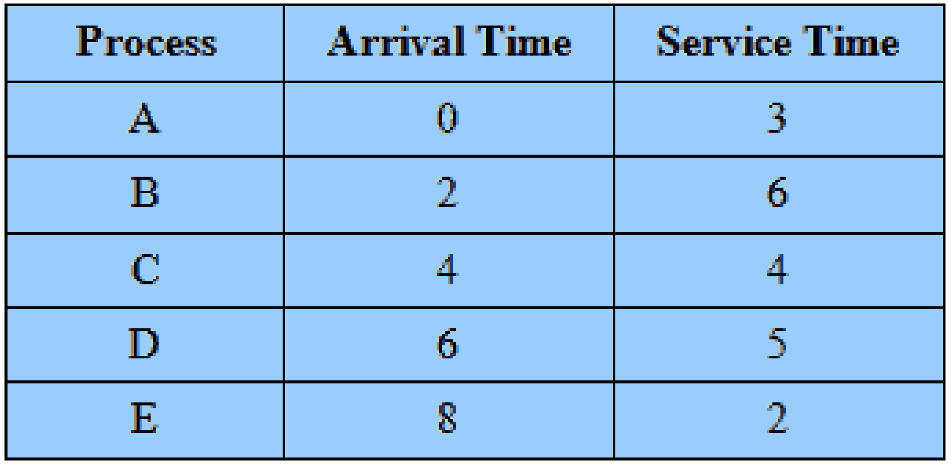
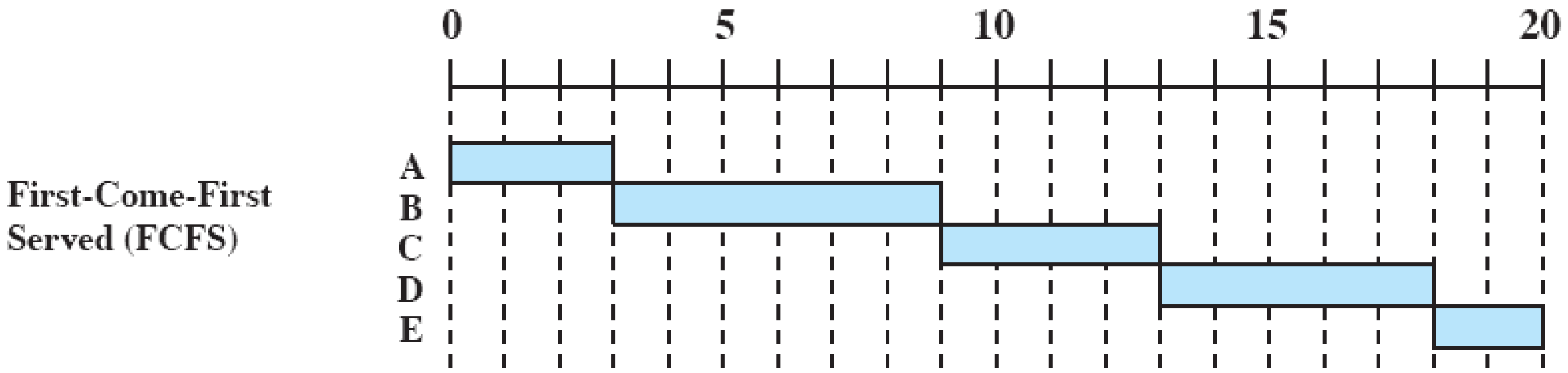


Table 9.4 Process Scheduling Example

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

First-Come-First-Served

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



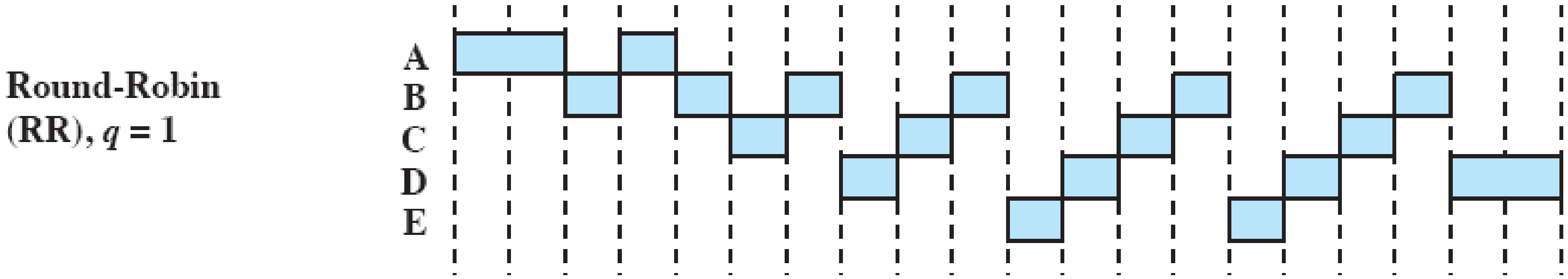
First-Come-First-Served

- A short process may have to wait a very long time before it can execute
- Good for computation-intensive processes
- Favors CPU-bound processes
  - I/O processes will block soon and then have to wait until CPU-bound process completes

Round Robin

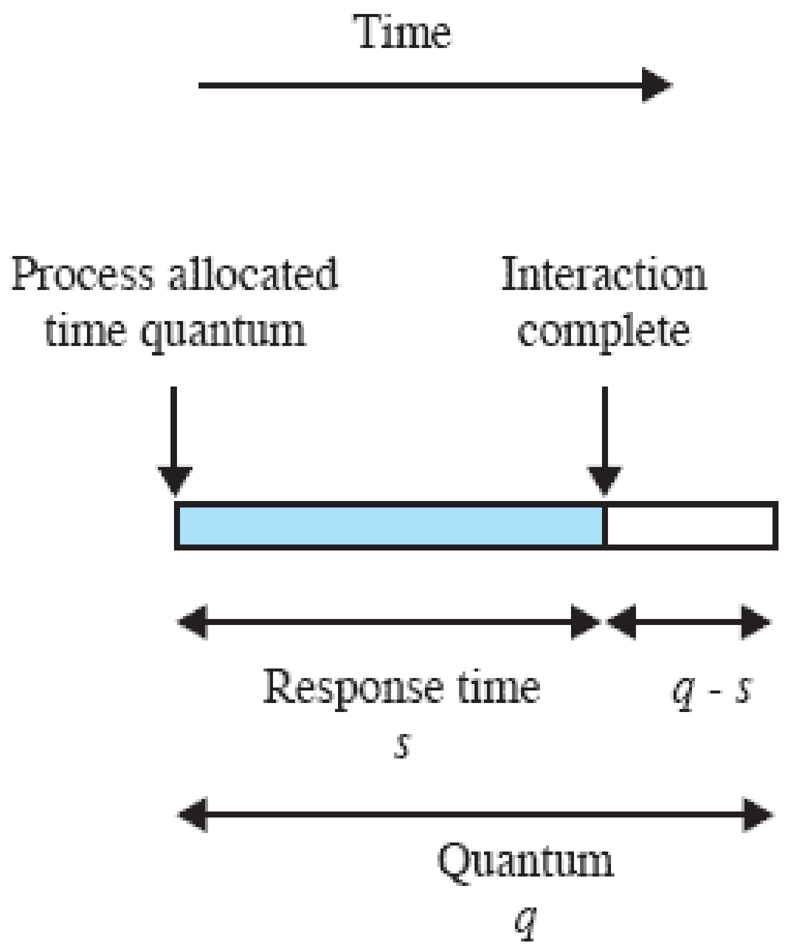
- Uses preemption based on a clock
- Clock interrupt is generated at periodic intervals
- When an interrupt occurs, the currently running process is placed in the ready queue
  - Next ready job is selected
- Known as time slicing

Round Robin



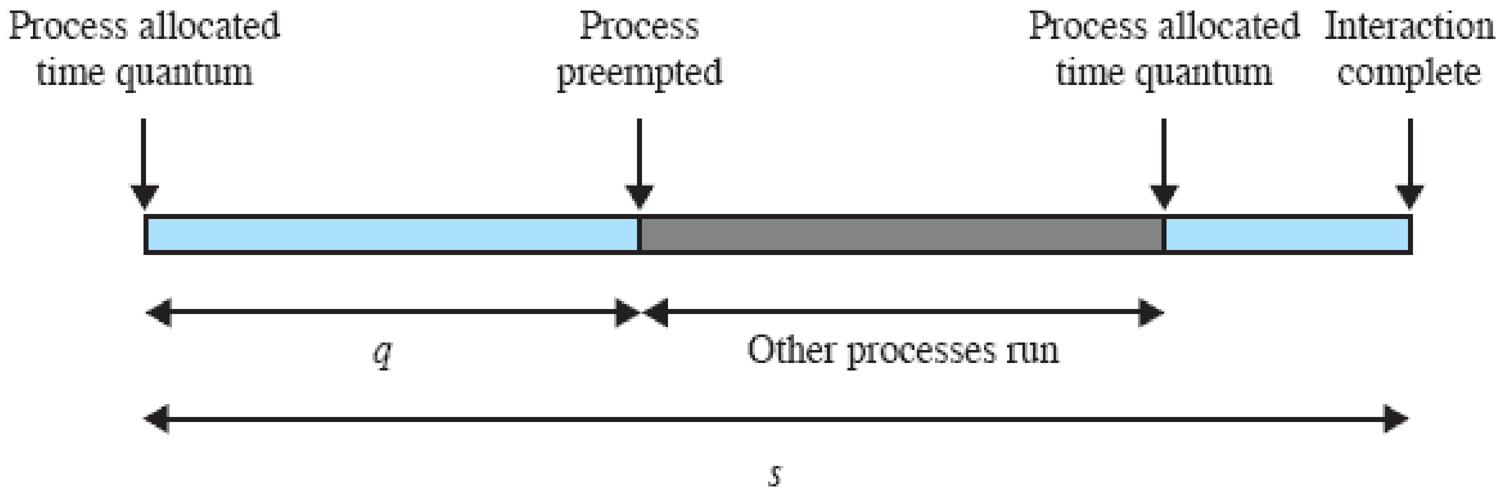
- Good overall strategy
- Processor-bound processes receive an unfair share

Effect of Size of Preemption Time Quantum



(a) Time quantum greater than typical interaction

Effect of Size of Preemption Time Quantum


Process allocated time quantum
Process preempted
Process allocated time quantum
Interaction complete
q
Other processes run
s

(b) Time quantum less than typical interaction

Figure 9.6 Effect of Size of Preemption Time Quantum

Virtual RR

Aux. queue raises priority of I/O heavy processes.

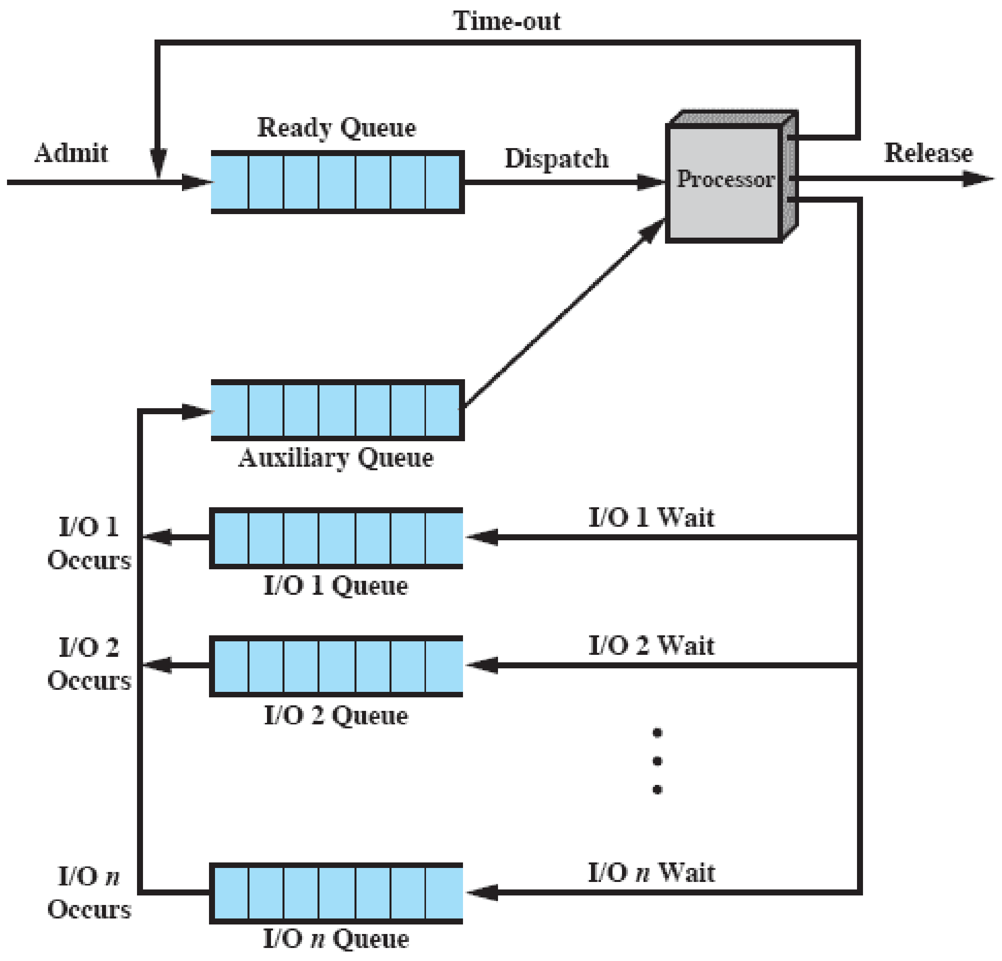
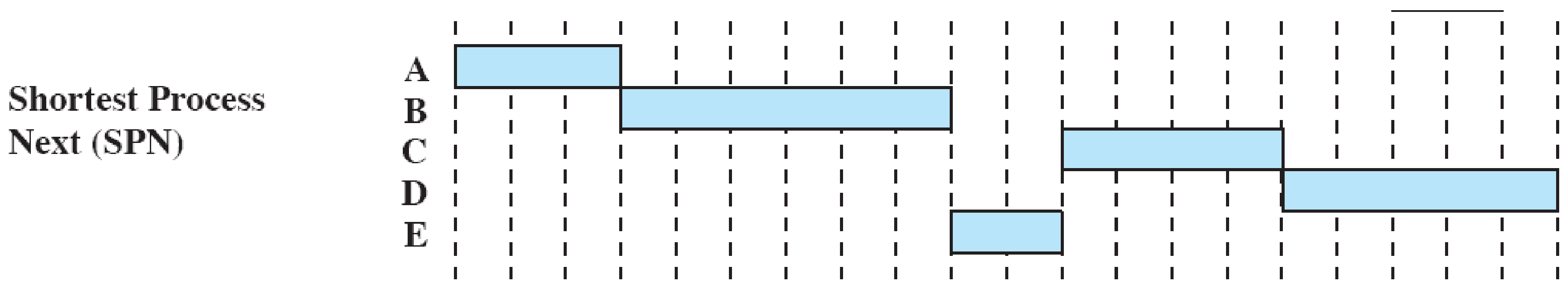


Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler

Shortest Process Next

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

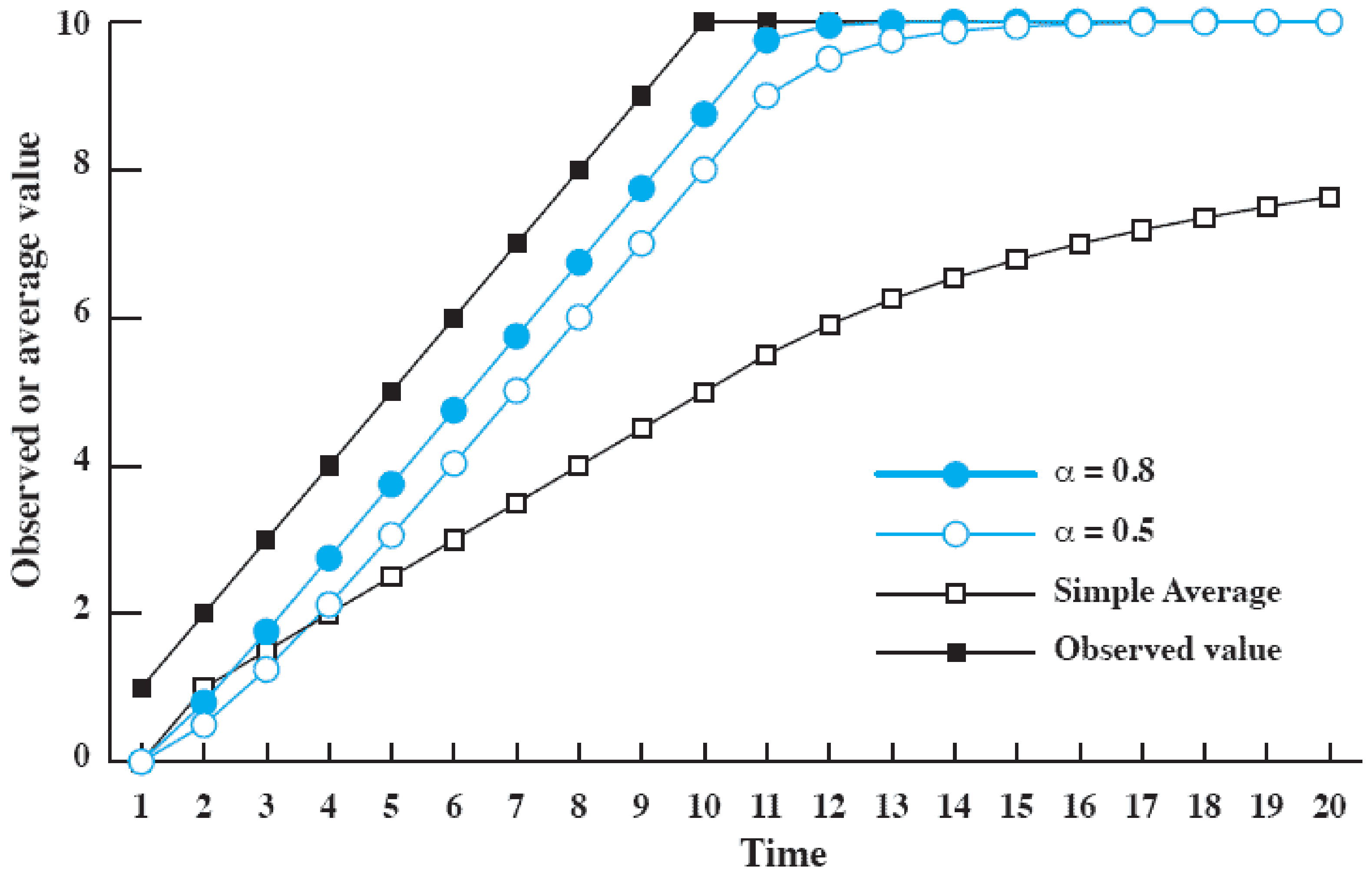


Shortest Process Next

- 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.

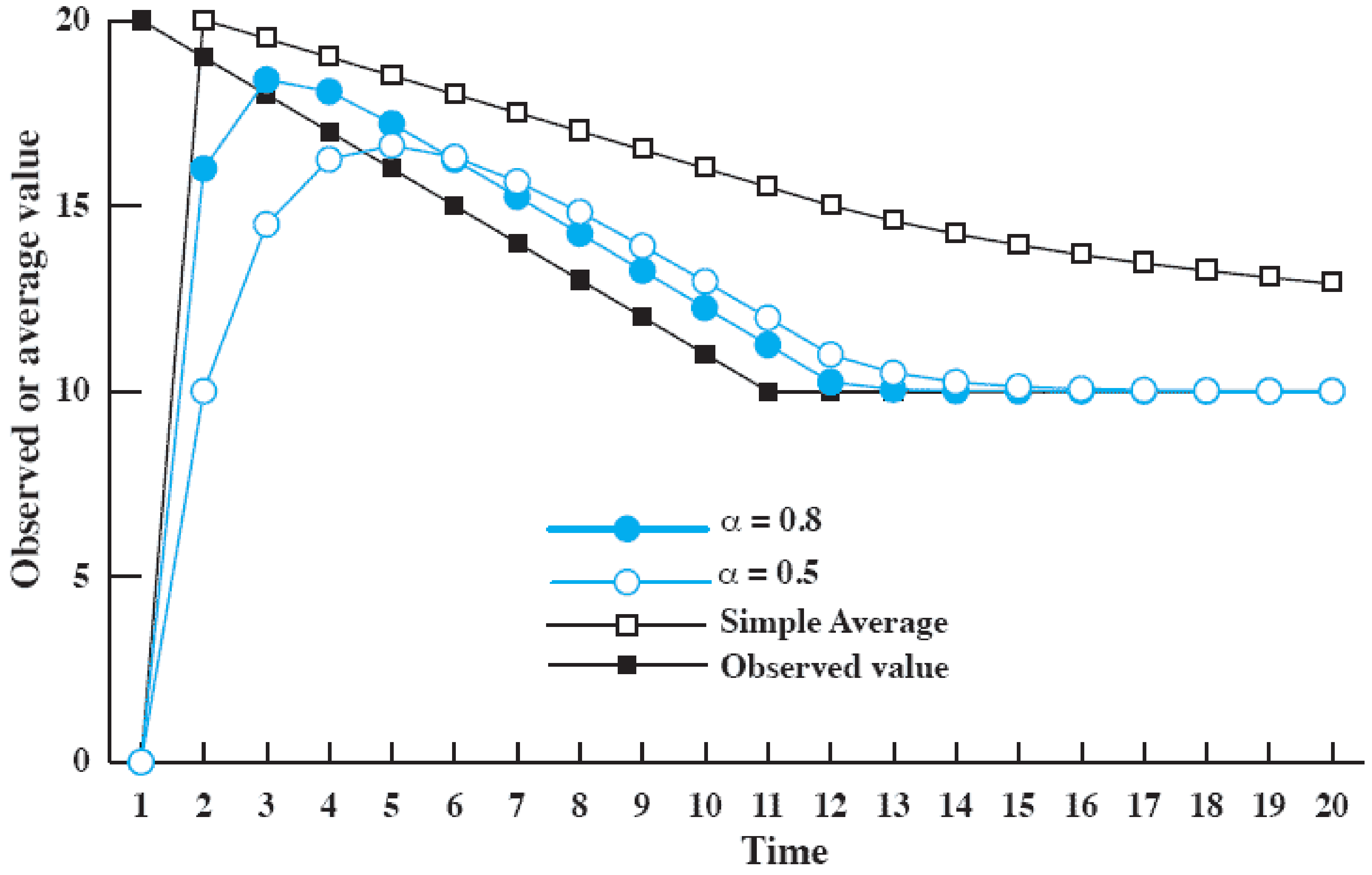
- Big limitation: Know the service time!

Use Of Exponential Averaging



(a) Increasing function

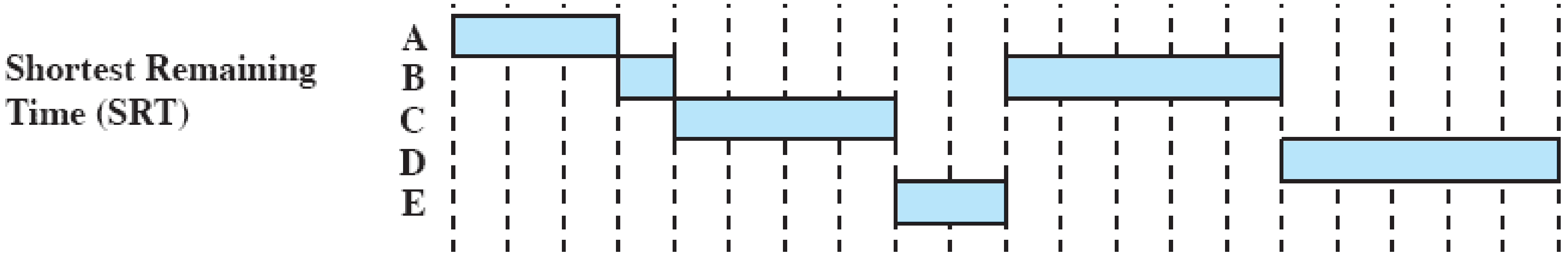
Use Of Exponential Averaging



(b) Decreasing function

Shortest Remaining Time

- Preemptive version of SPN policy
- Must estimate processing time
- Starvation of long processes



Shortest Remaining Time

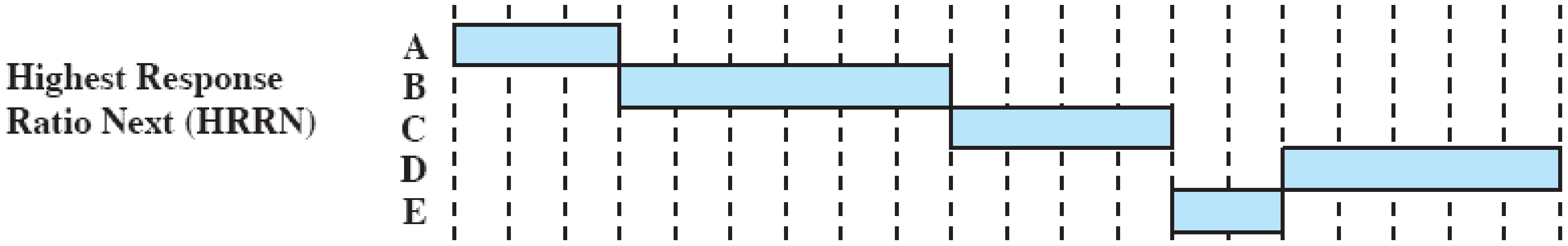
- Advantage:
  - No additional interrupt like in RR
  - Better general turn-around time than SPN
  - No bias for long processes like in FCFS

- Disadvantage:
  - Elapsed service time must be recorded

Highest Response Ratio Next

- Choose next process with the greatest ratio

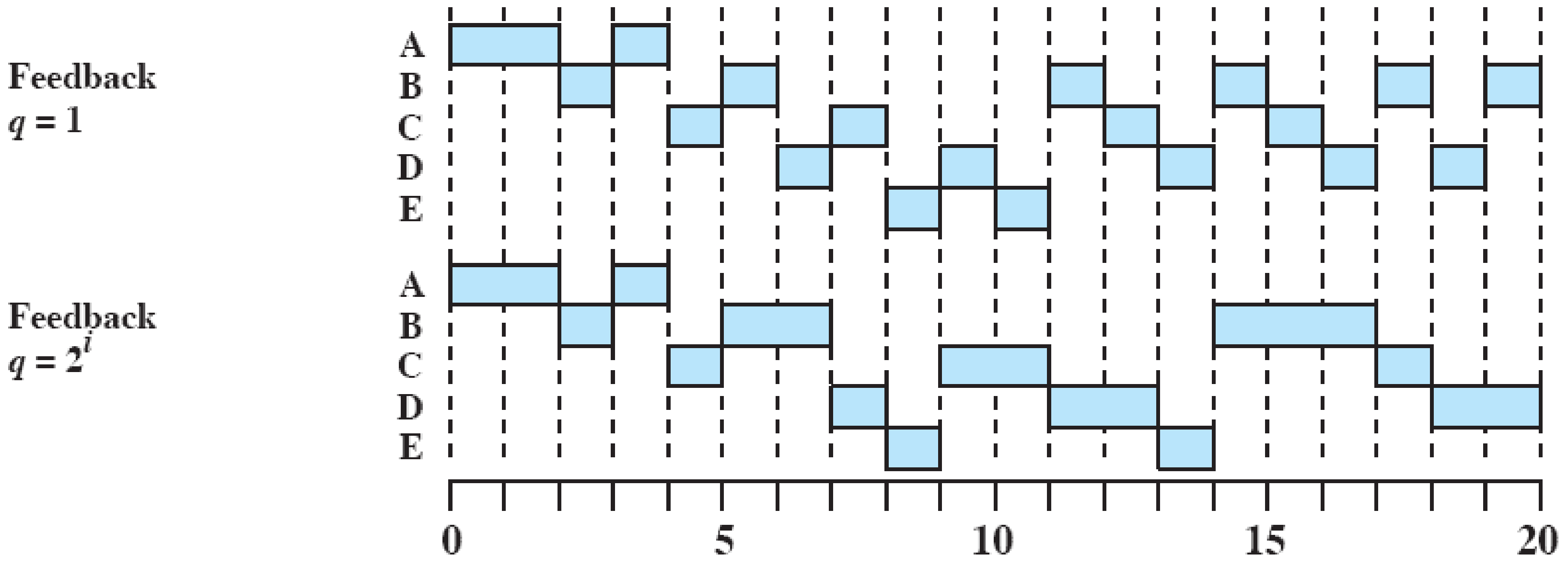
$$Ratio = \frac{time\ spent\ waiting + expected\ service\ time}{expected\ service\ time}$$



- Accounts for age and short processes (better ratio)
- Still requires knowing the service time

Feedback-based Scheduling

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



Feedback Scheduling

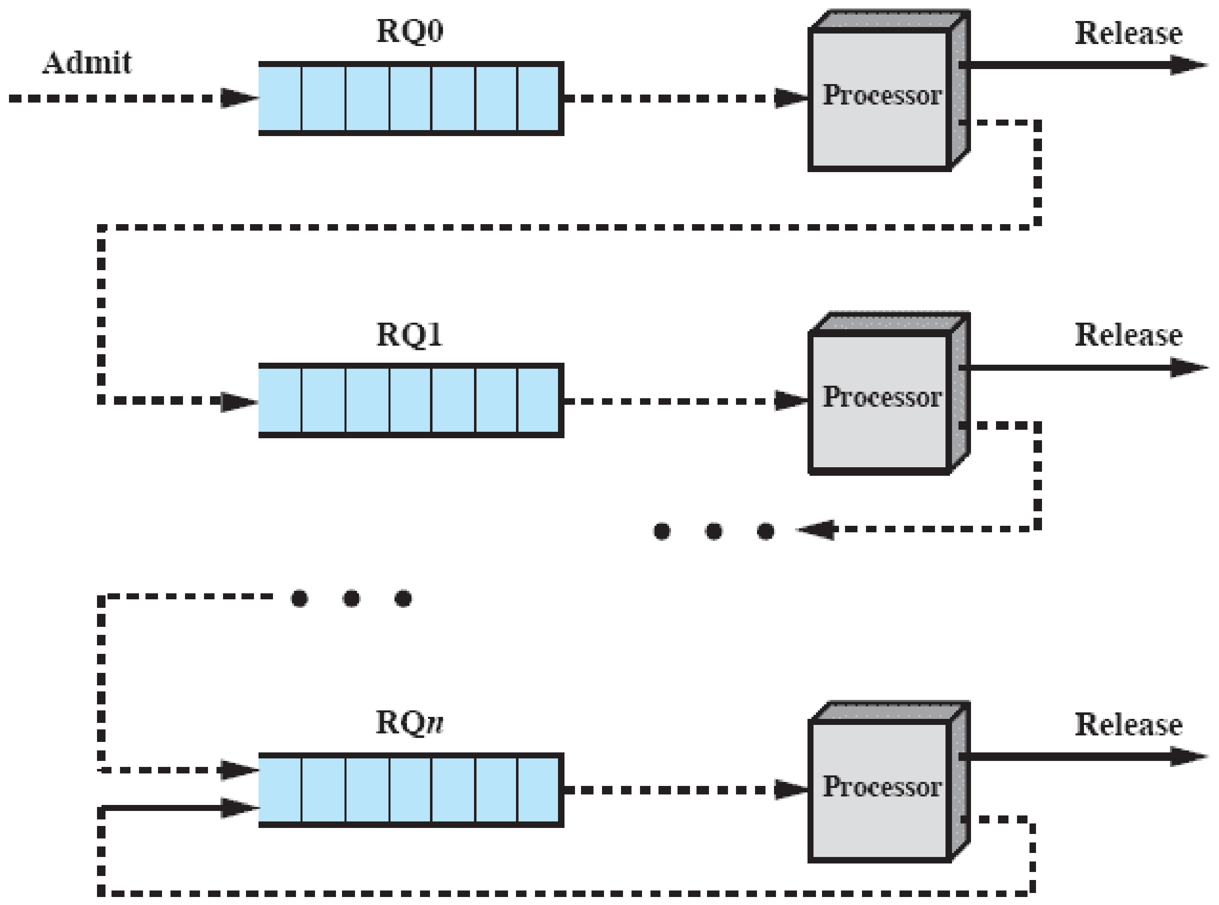
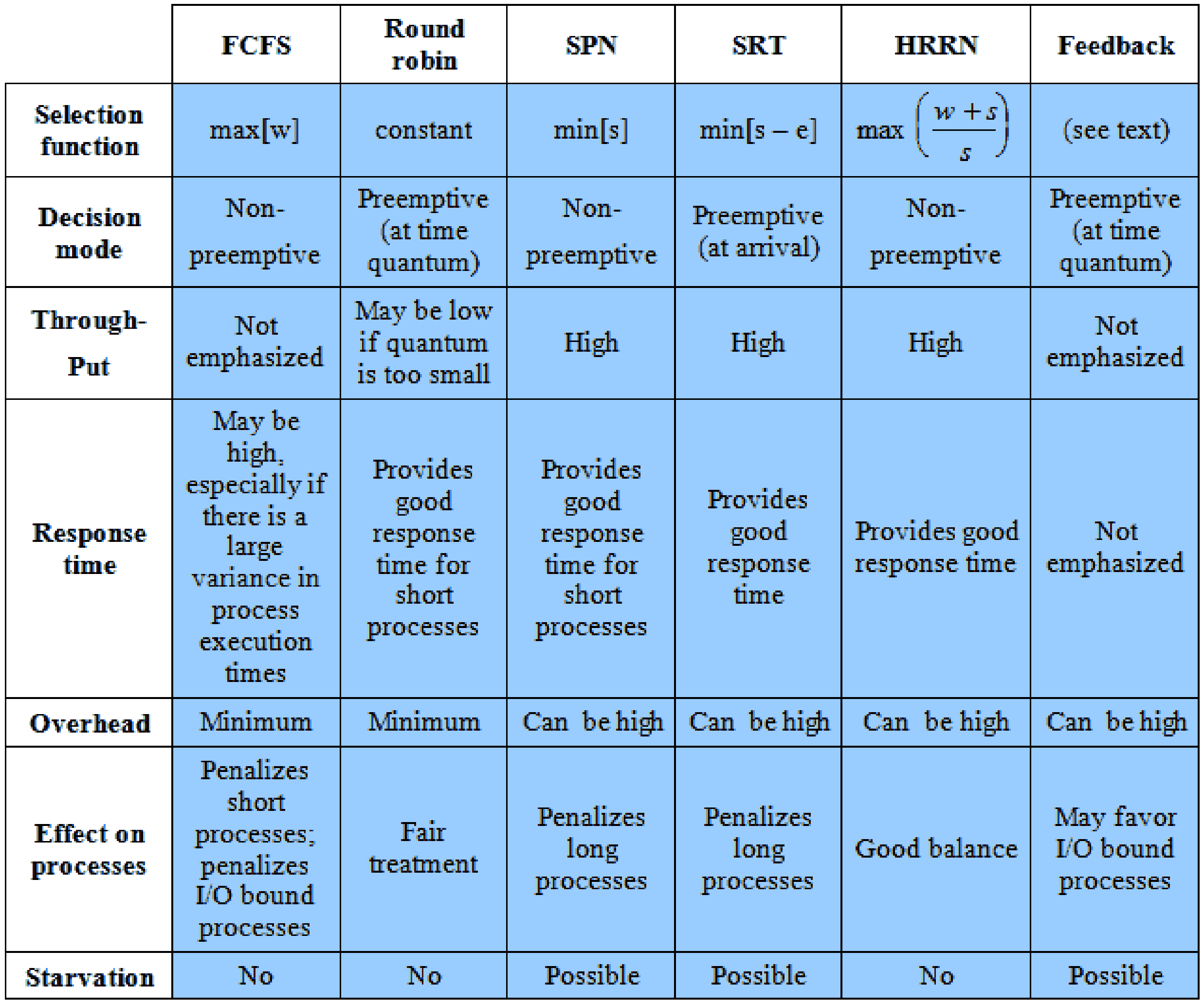


Figure 9.10 Feedback Scheduling

Scheduling Policies

Table 9.3 Characteristics of Various Scheduling Policies



	FCFS	Round robin	SPN	SRT	HRRN	Feedback
Selection function	max[w]	constant	min[s]	min[s – e]	$\max\left(\frac{w+s}{s}\right)$	(see text)
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Through-Put	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response time	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
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

Scheduling Policies

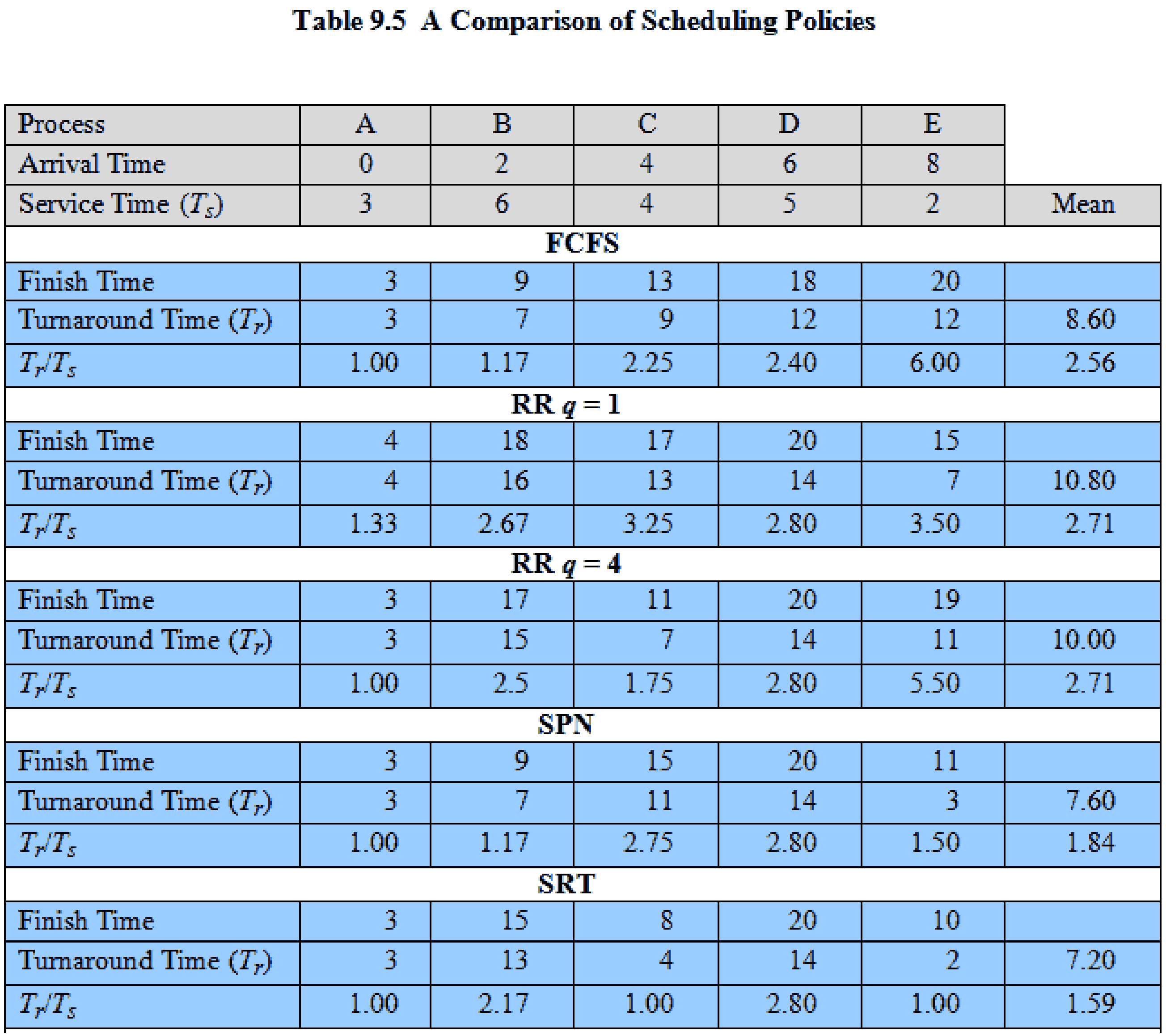
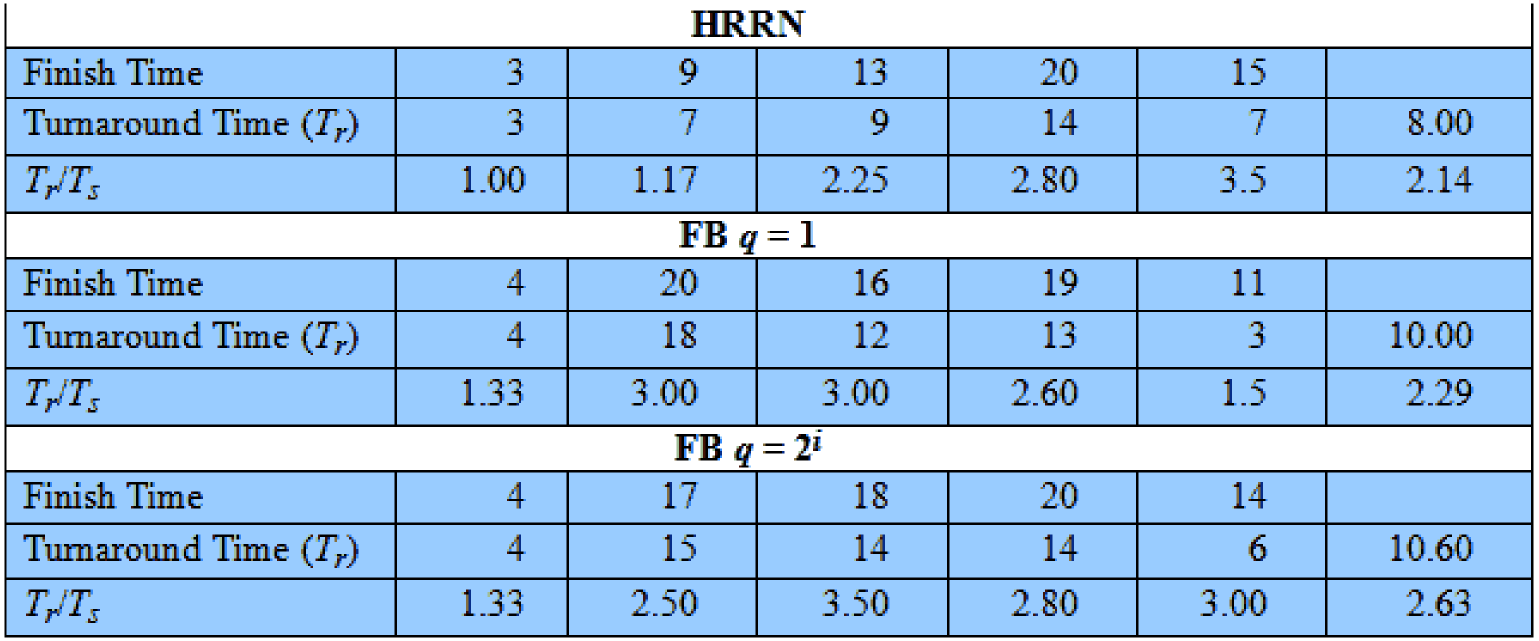


Table 9.5 A Comparison of Scheduling Policies

Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
FCFS						
Finish Time	3	9	13	18	20	
Turnaround Time (T_r)	3	7	9	12	12	8.60
T_r/T_s	1.00	1.17	2.25	2.40	6.00	2.56
RR $q = 1$						
Finish Time	4	18	17	20	15	
Turnaround Time (T_r)	4	16	13	14	7	10.80
T_r/T_s	1.33	2.67	3.25	2.80	3.50	2.71
RR $q = 4$						
Finish Time	3	17	11	20	19	
Turnaround Time (T_r)	3	15	7	14	11	10.00
T_r/T_s	1.00	2.5	1.75	2.80	5.50	2.71
SPN						
Finish Time	3	9	15	20	11	
Turnaround Time (T_r)	3	7	11	14	3	7.60
T_r/T_s	1.00	1.17	2.75	2.80	1.50	1.84
SRT						
Finish Time	3	15	8	20	10	
Turnaround Time (T_r)	3	13	4	14	2	7.20
T_r/T_s	1.00	2.17	1.00	2.80	1.00	1.59

Comparison of Scheduling Policies

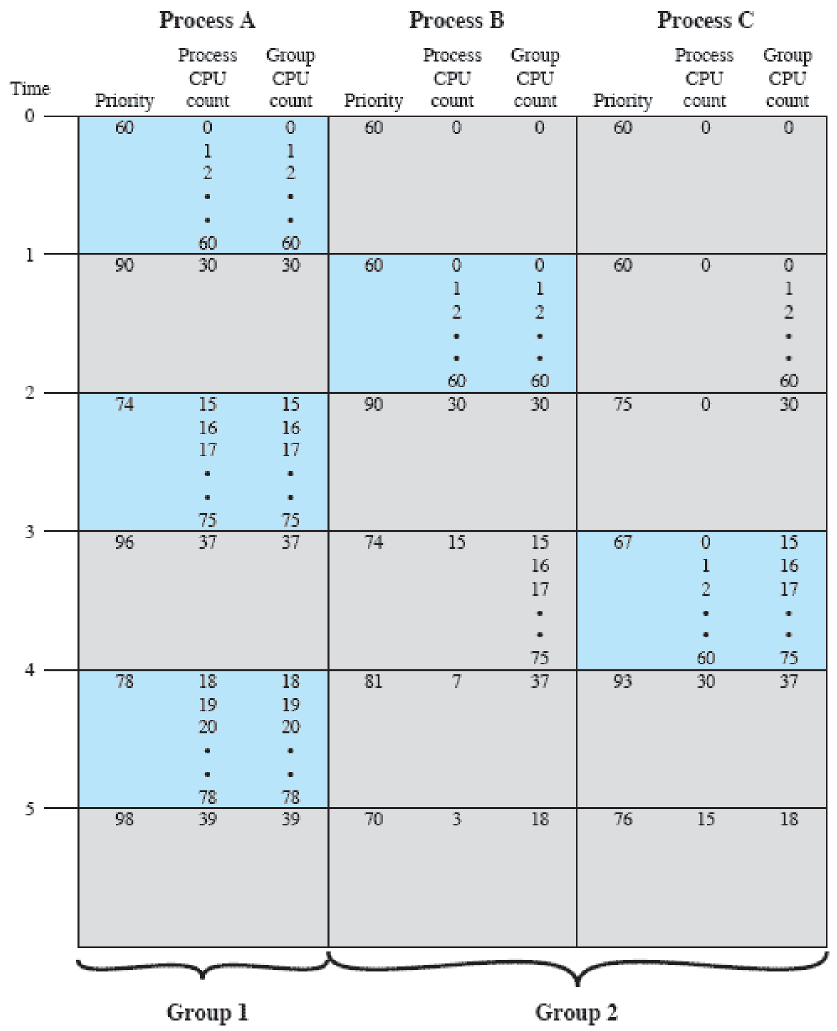


HRRN						
Finish Time	3	9	13	20	15	
Turnaround Time (T_r)	3	7	9	14	7	8.00
T_r/T_s	1.00	1.17	2.25	2.80	3.5	2.14
FB $q = 1$						
Finish Time	4	20	16	19	11	
Turnaround Time (T_r)	4	18	12	13	3	10.00
T_r/T_s	1.33	3.00	3.00	2.60	1.5	2.29
FB $q = 2^i$						
Finish Time	4	17	18	20	14	
Turnaround Time (T_r)	4	15	14	14	6	10.60
T_r/T_s	1.33	2.50	3.50	2.80	3.00	2.63

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



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

Colored rectangle represents executing process

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

Weight: 0.5
B&C are in the same group

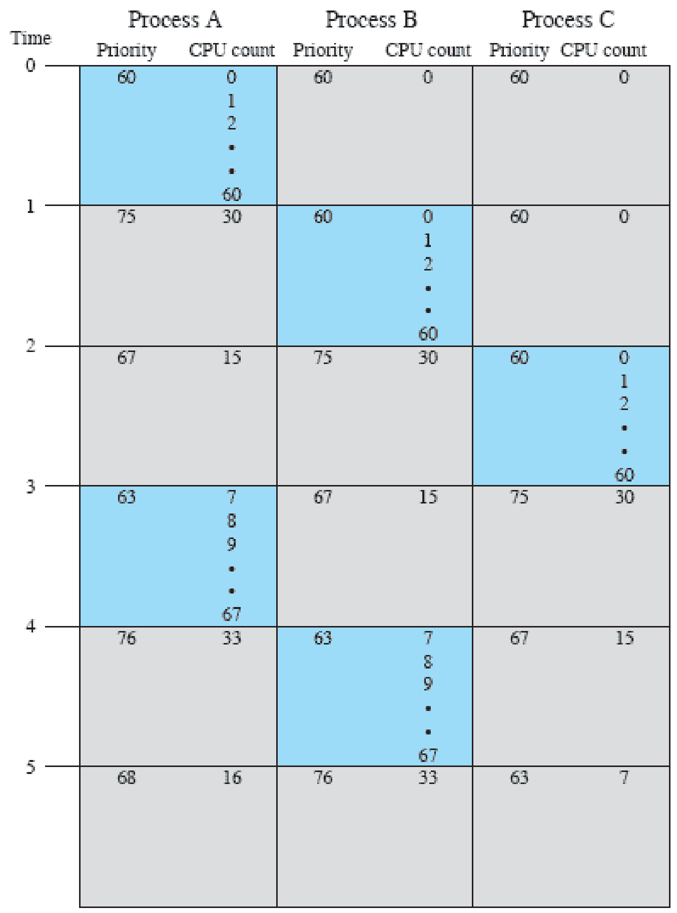
Traditional UNIX Scheduling

- 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
- Priorities are recomputed once per second
- Base priority divides all processes into fixed bands of priority levels

Bands

- Decreasing order of priority
  - Swapper
  - Block I/O device control
  - File manipulation
  - Character I/O device control
  - User processes

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	0 1 2 • • 60	60	0
2	67	15	75	30	60	0 1 2 • • 60
3	63	7 8 9 • • 67	67	15	75	30
4	76	33	63	7 8 9 • • 67	67	15
5	68	16	76	33	63	7

Colored rectangle represents executing process

Figure 9.17 Example of Traditional UNIX Process Scheduling