Chapter 9 - Lecture

Stallings - 9e

Aim of Scheduling

- Assign processes to be executed by the processor(s)
  - Response time
  - Throughput
  - Processor utilization
  - Tardiness etc.

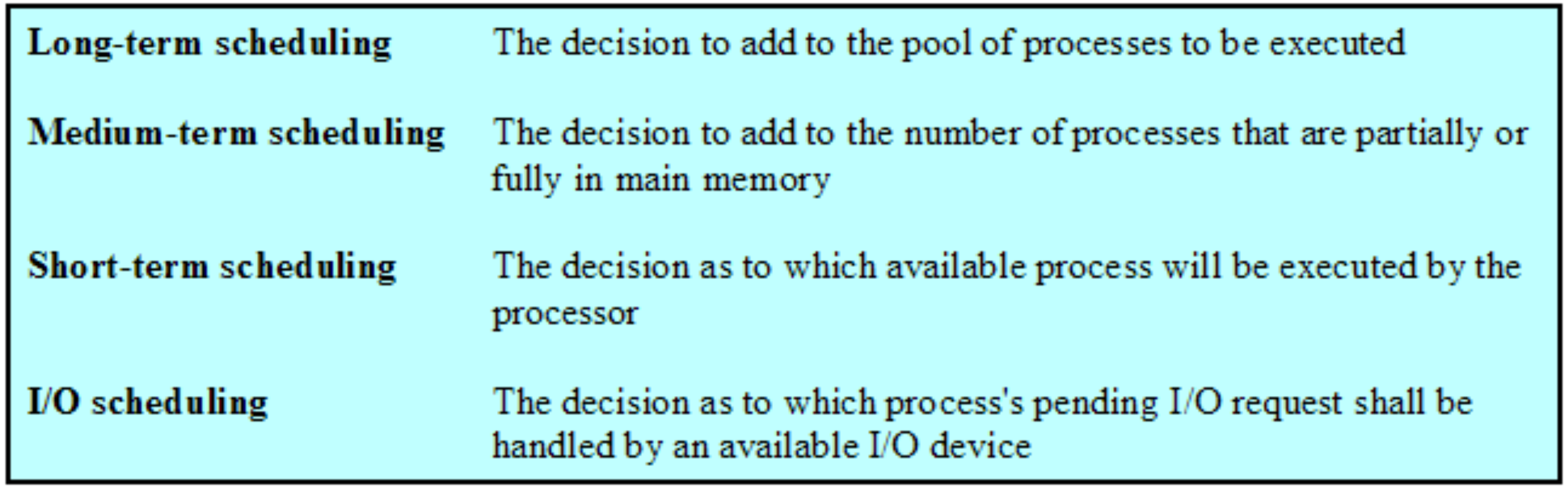
Aim of Scheduling

- Assign processes to be executed by the processor(s)
  - Response time
  - Throughput
  - Processor utilization
  - Tardiness etc.

Scheduling Environments

- Single vs. multiple processors
- Static vs. dynamic process arrival
- Preemptive vs. nonpreemptive
- Independent vs. dependent tasks
- etc.

Table 9.1 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

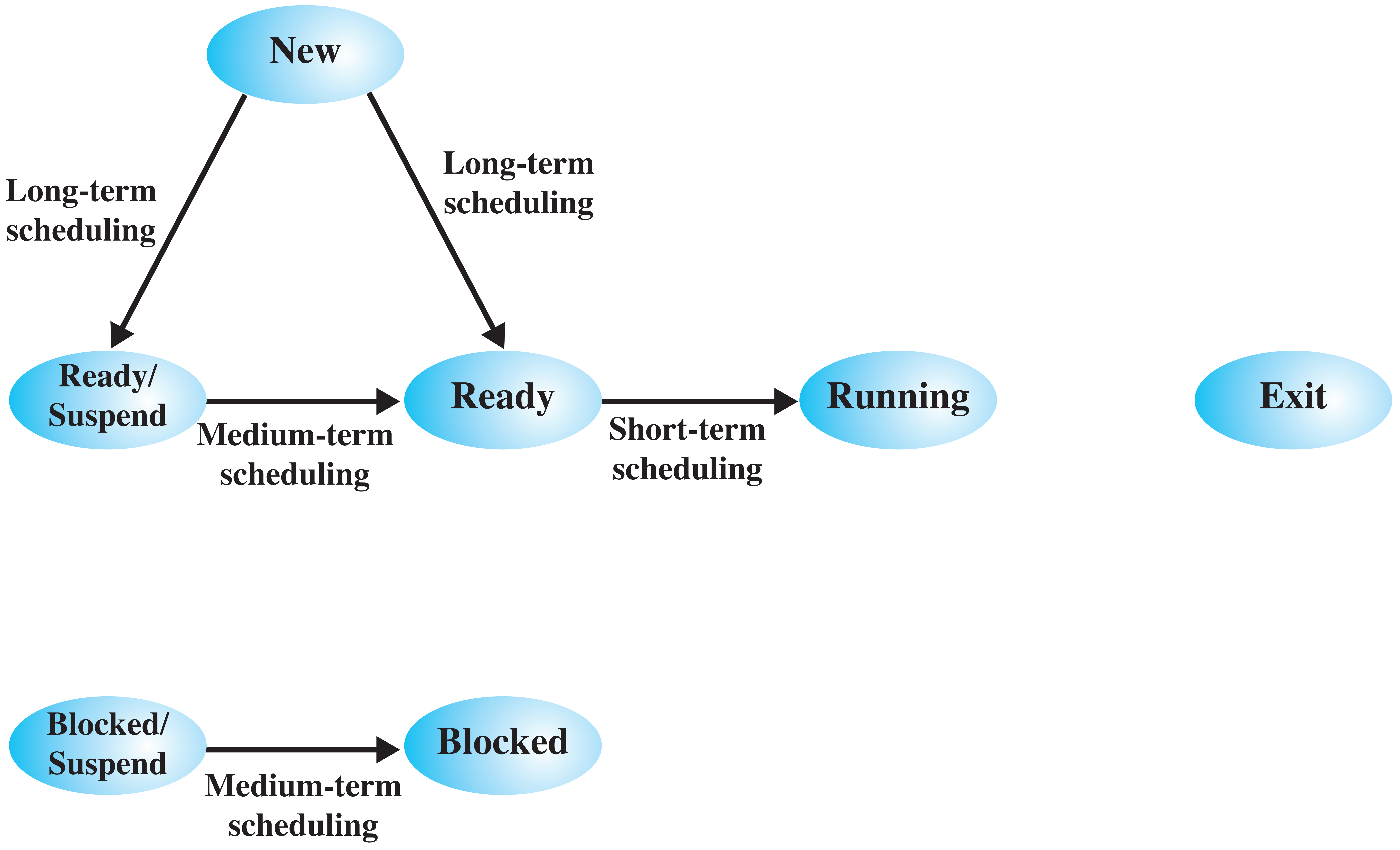


Figure 9.1 Scheduling and Process State Transitions

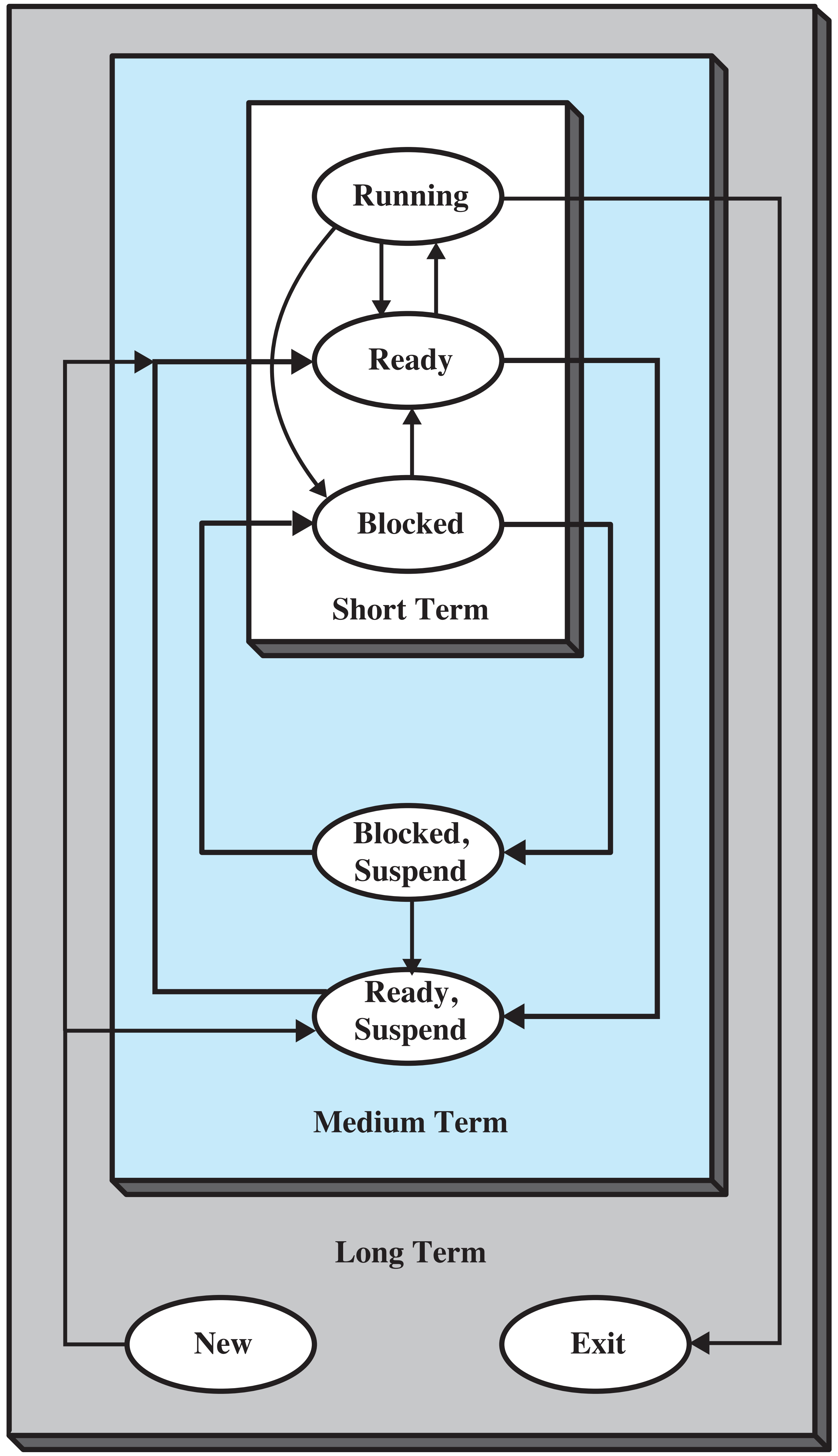


Figure 9.2 Levels of Scheduling

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

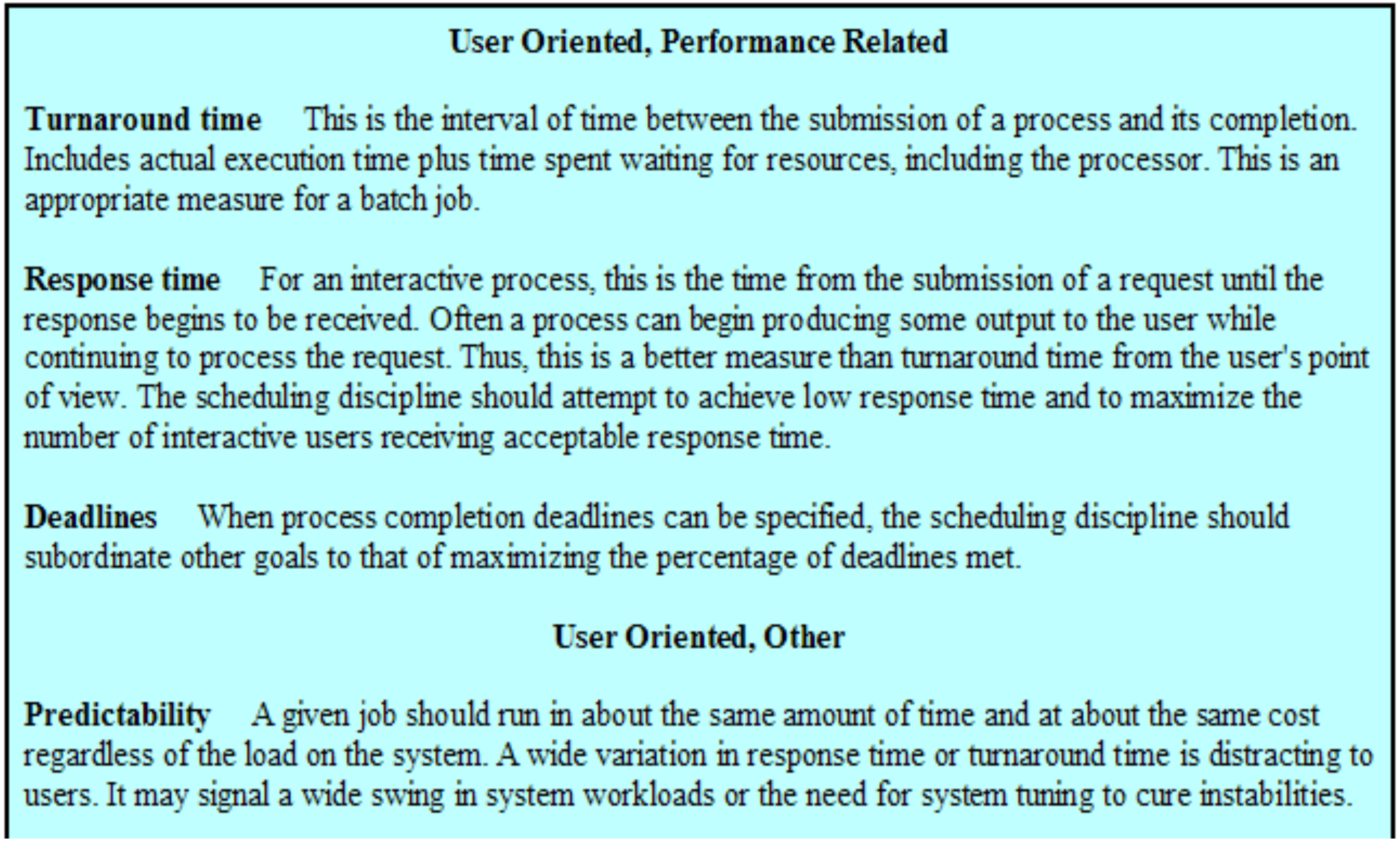
Short-Term Scheduling Criteria

- 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

Table 9.2 Scheduling Criteria



User Oriented, Performance Related

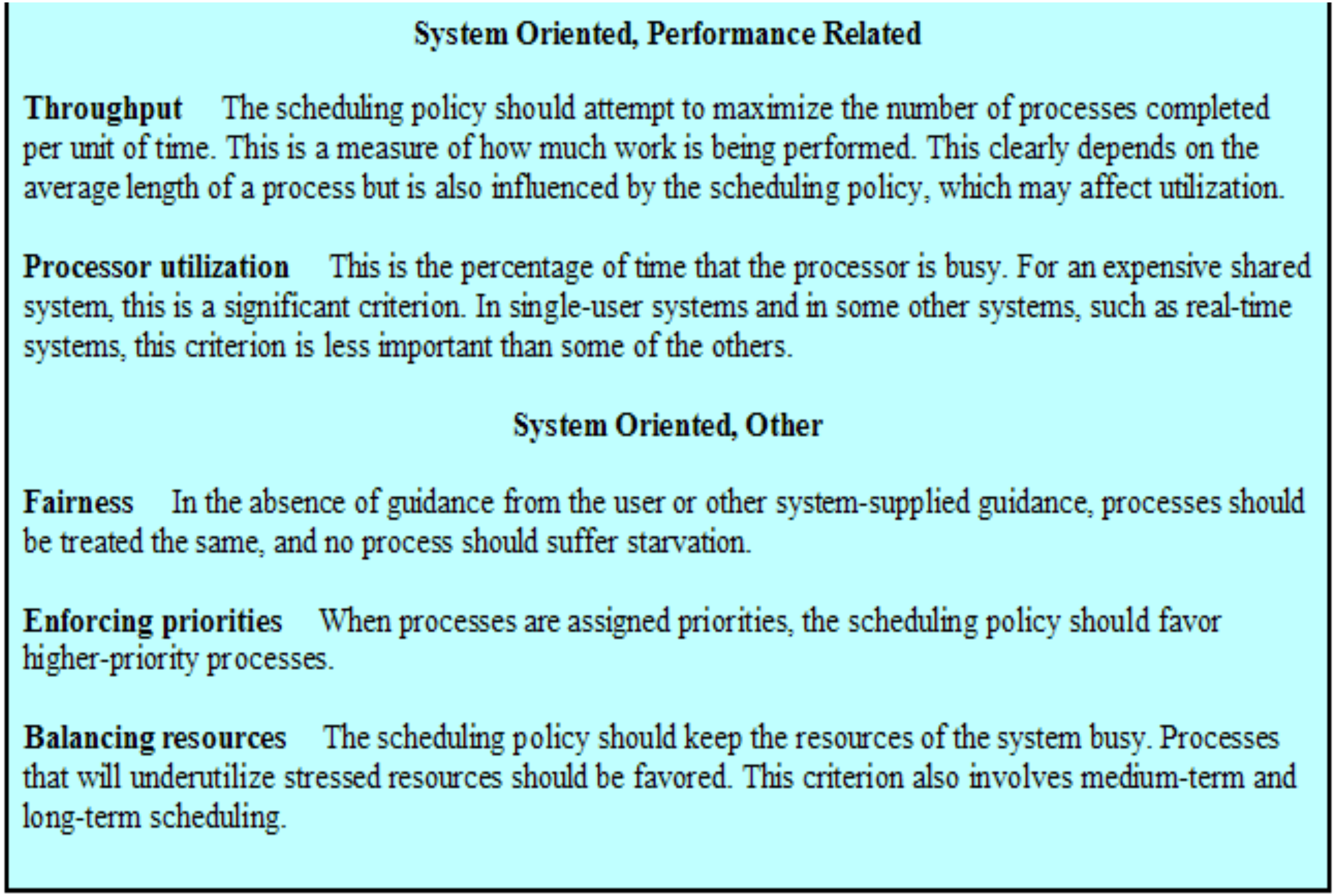
Turnaround time This is the interval of time between the submission of a process and its completion. Includes actual execution time plus time spent waiting for resources, including the processor. This is an appropriate measure for a batch job.

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.



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.

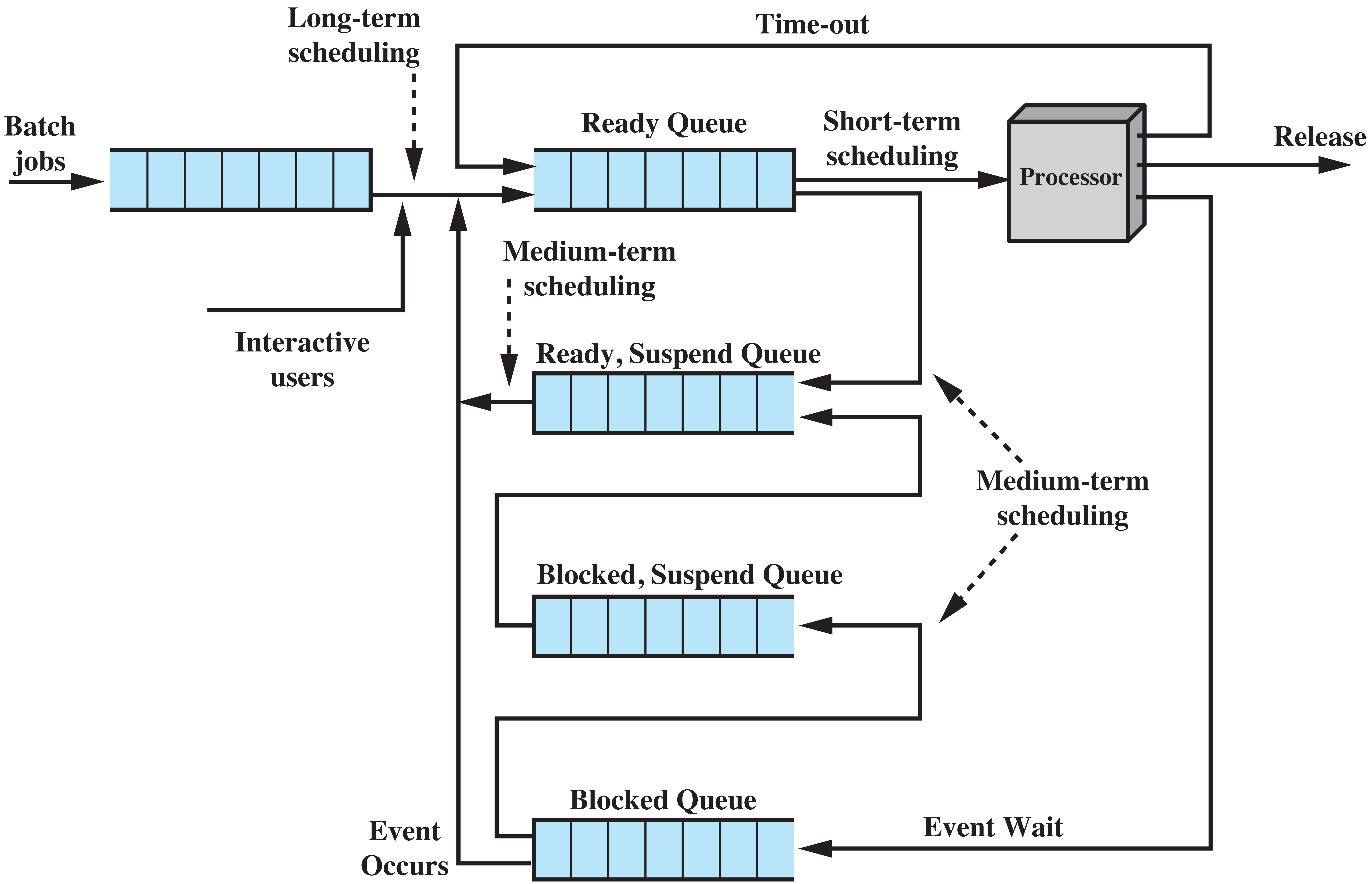


Figure 9.3 Queuing Diagram for Scheduling

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
- Lower-priority may suffer starvation
  - Allow a process to change its priority based on its age or execution history

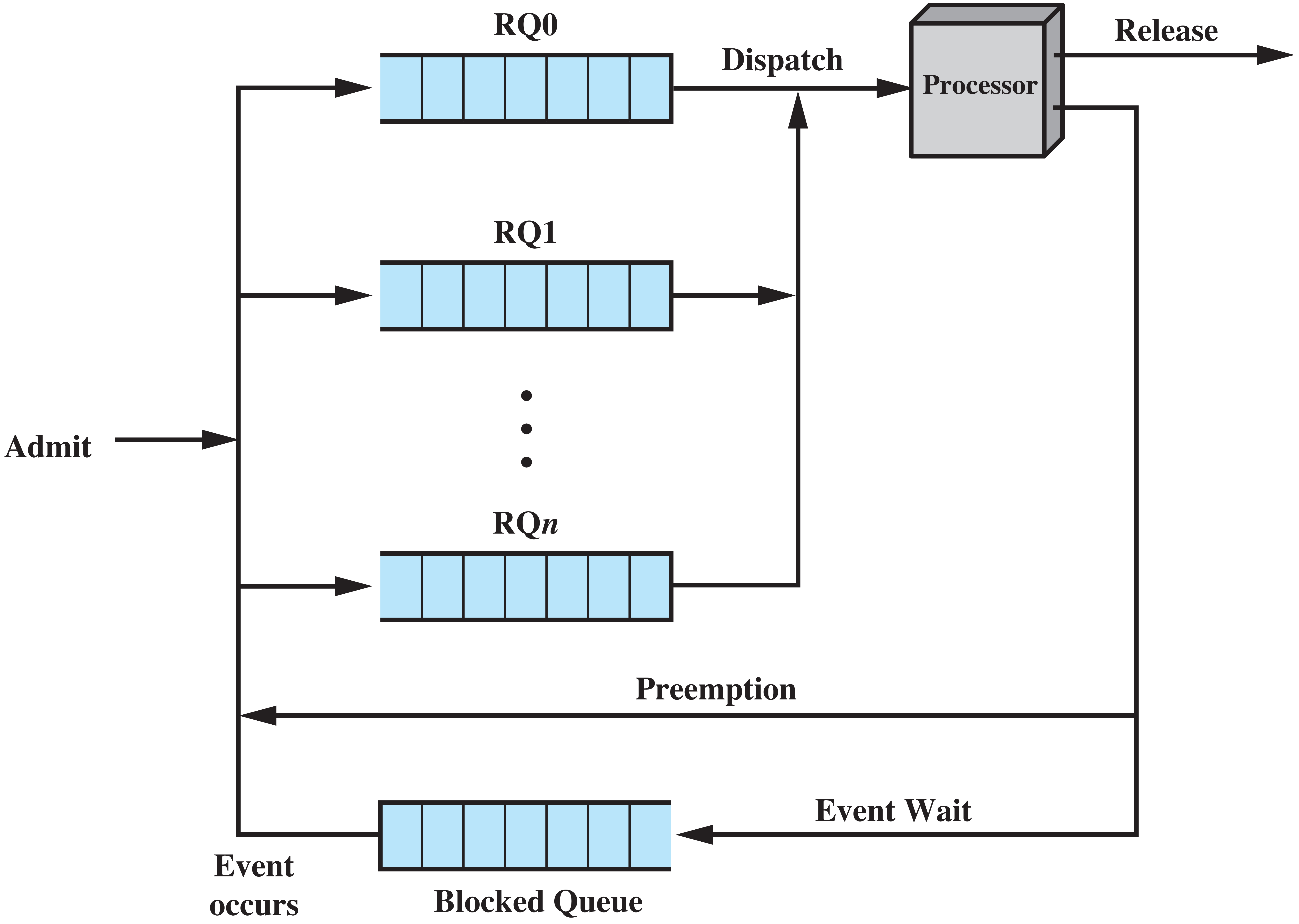


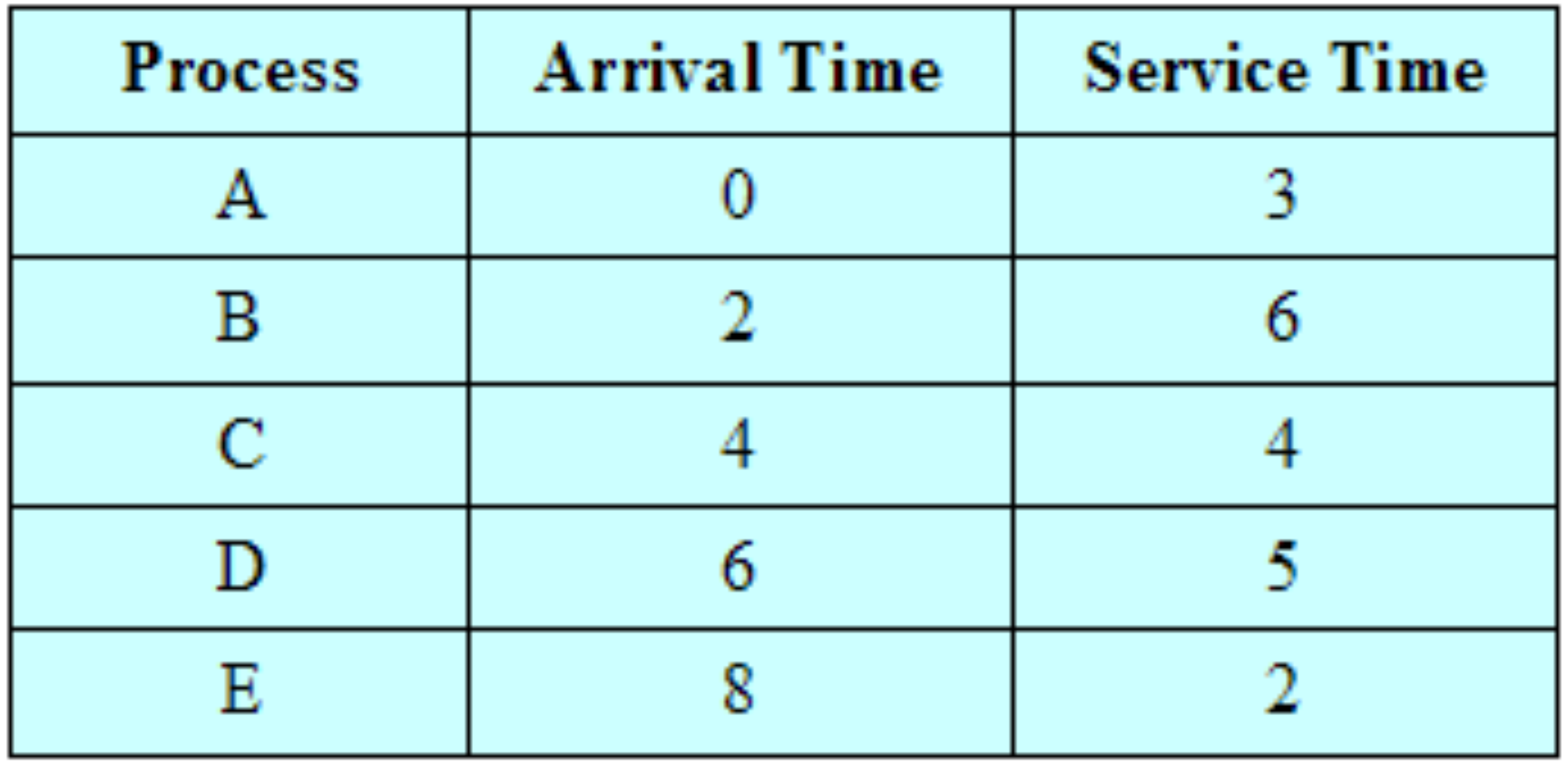
Figure 9.4 Priority Queuing

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

Table 9.4 Process Scheduling Example

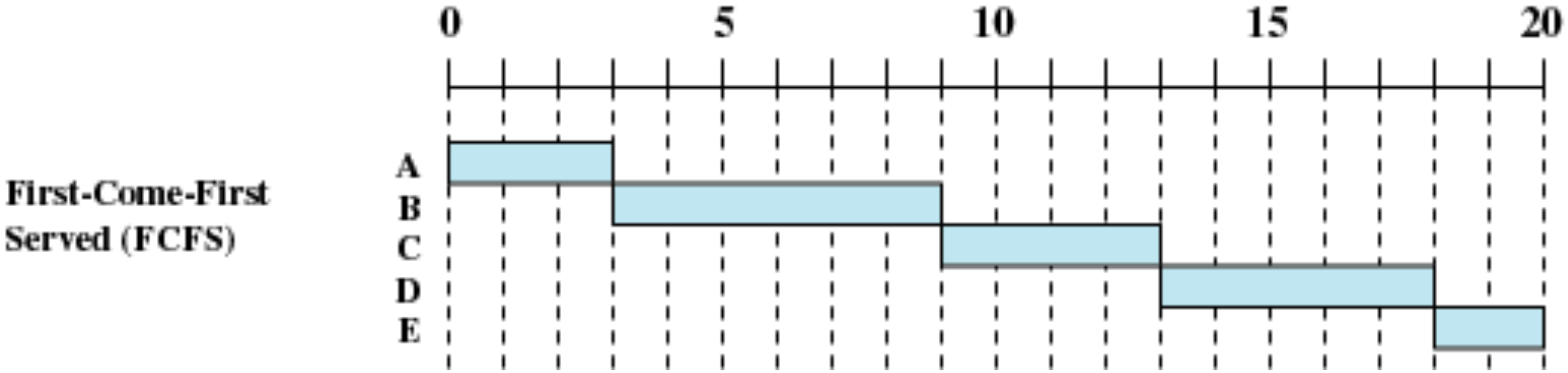


Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

First-Come-First-Served (FCFS)

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



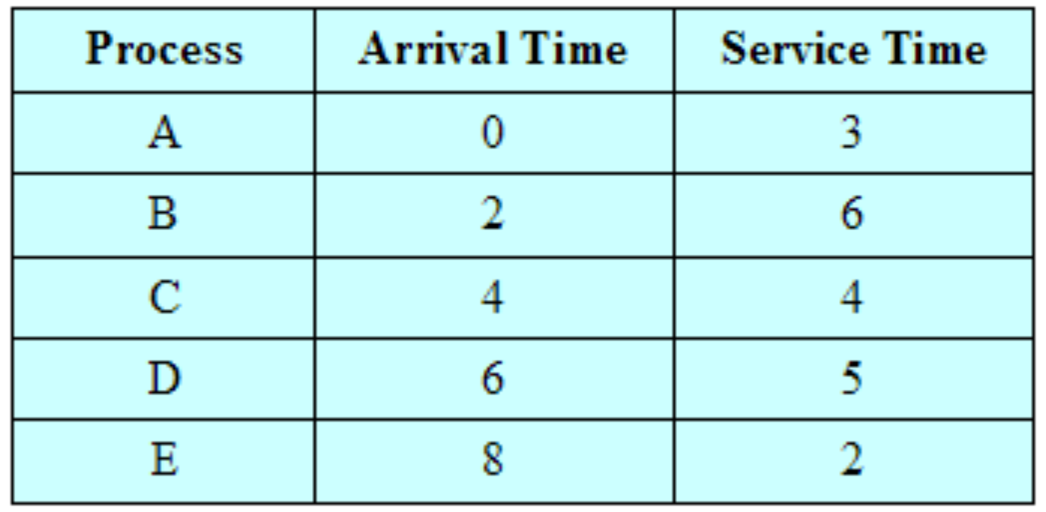
- Each process joins the Ready queue
- When the current process ceases to execute, the oldest process in the Ready queue is selected

First-Come-First-Served (FCFS)

- Also called FIFO
- Performs much better for long processes
  - 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

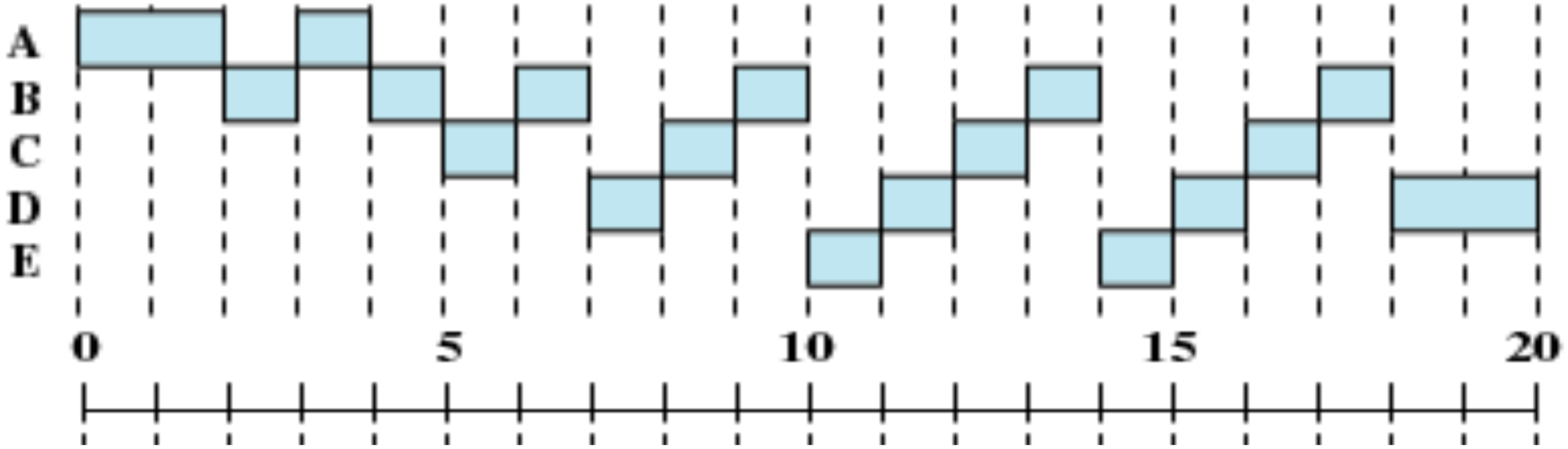
Round-Robin

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

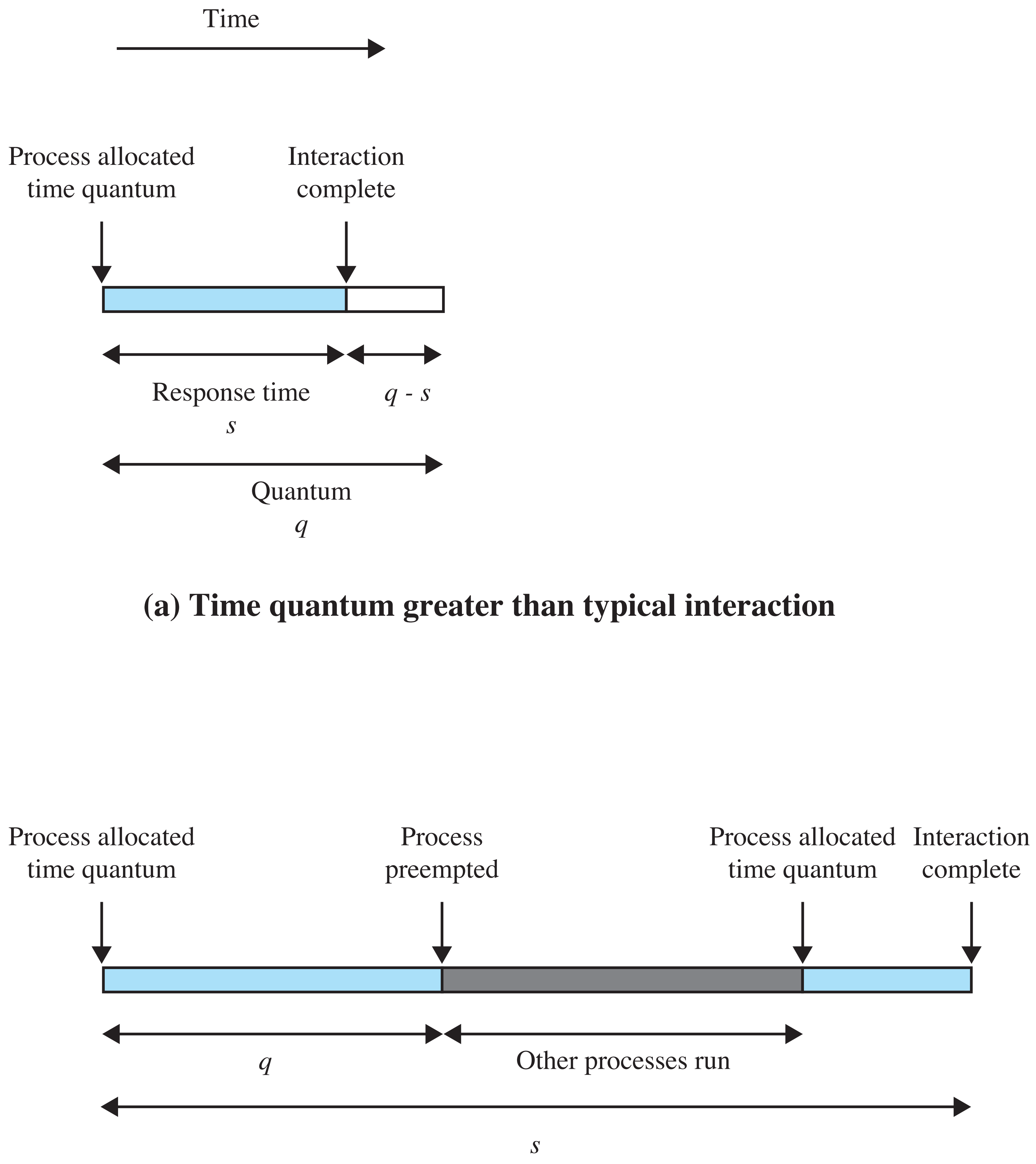
Round-Robin (RR), $q = 1$



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

Round-Robin

- Clock interrupt is generated at periodic intervals
- When an interrupt occurs, the currently running process is placed in the read queue
  - Next ready job is selected
- Known as *time slicing*



(a) Time quantum greater than typical interaction

(b) Time quantum less than typical interaction

Figure 9.6 Effect of Size of Preemption Time Quantum

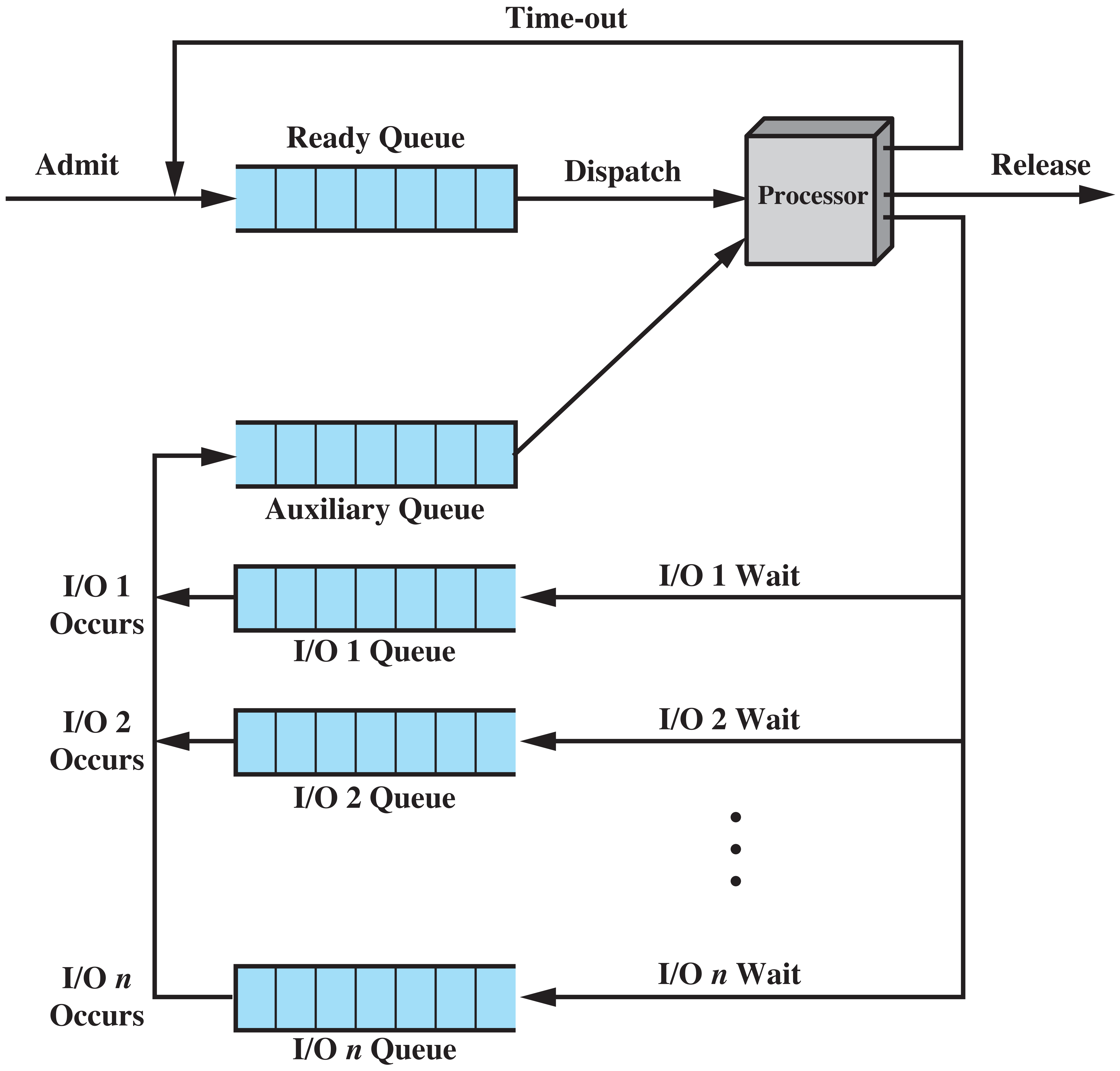


Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler

Shortest Process Next

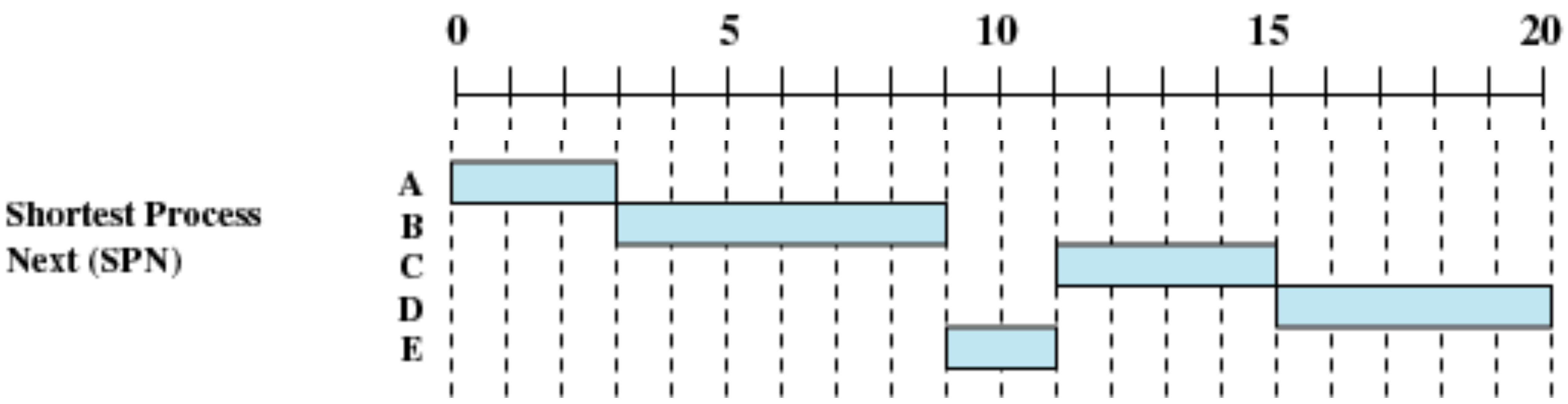
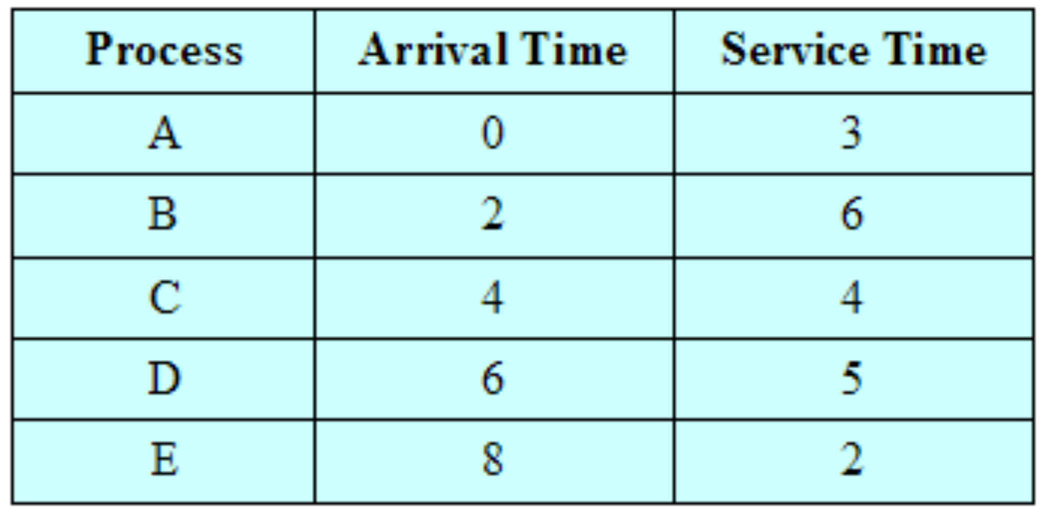


Table 9.4 Process Scheduling Example



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

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

Shortest Process Next

- Need to predict (or estimate) run time
- If estimated time for process not correct, the operating system may abort it
- Possibility of starvation for longer processes

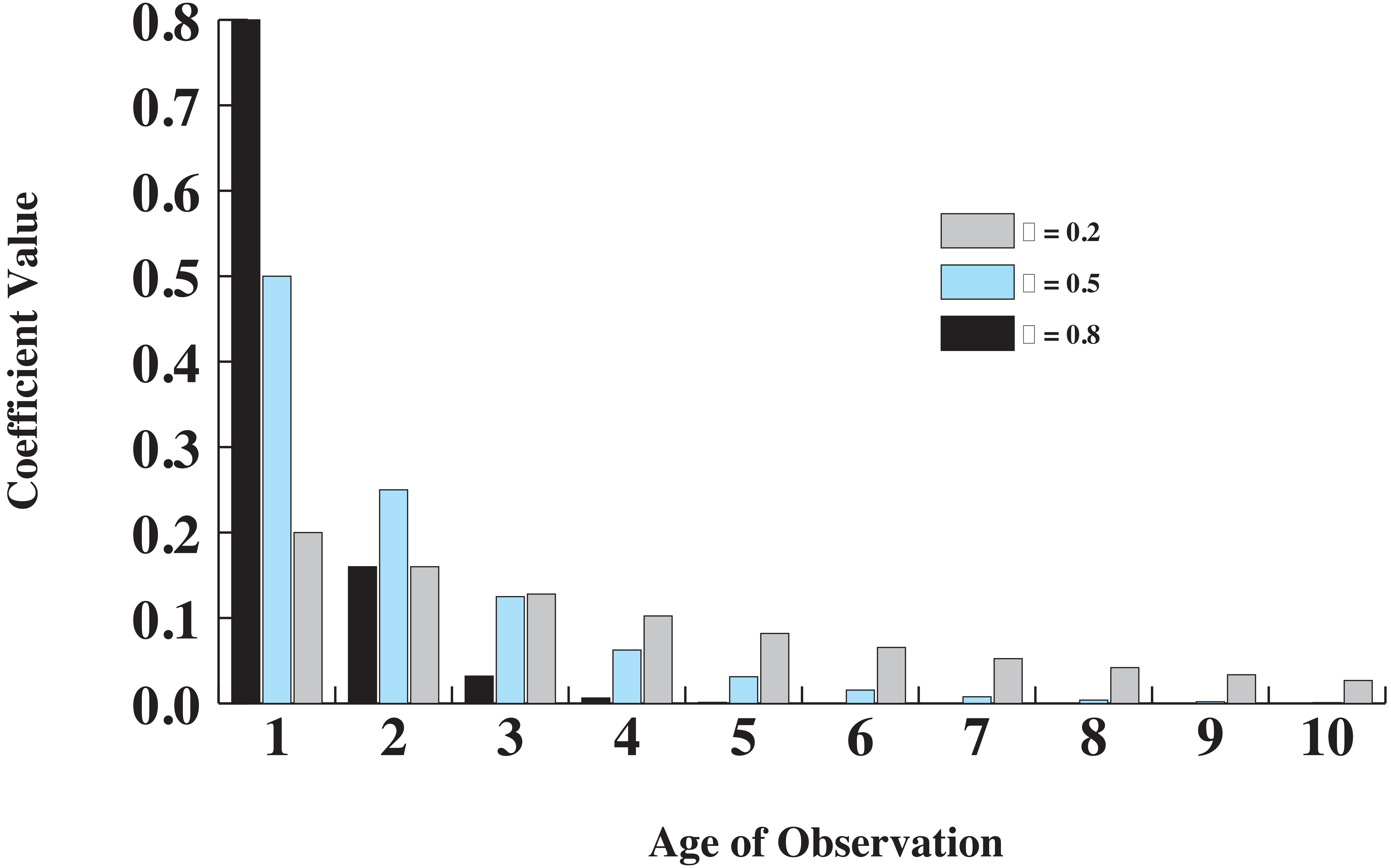
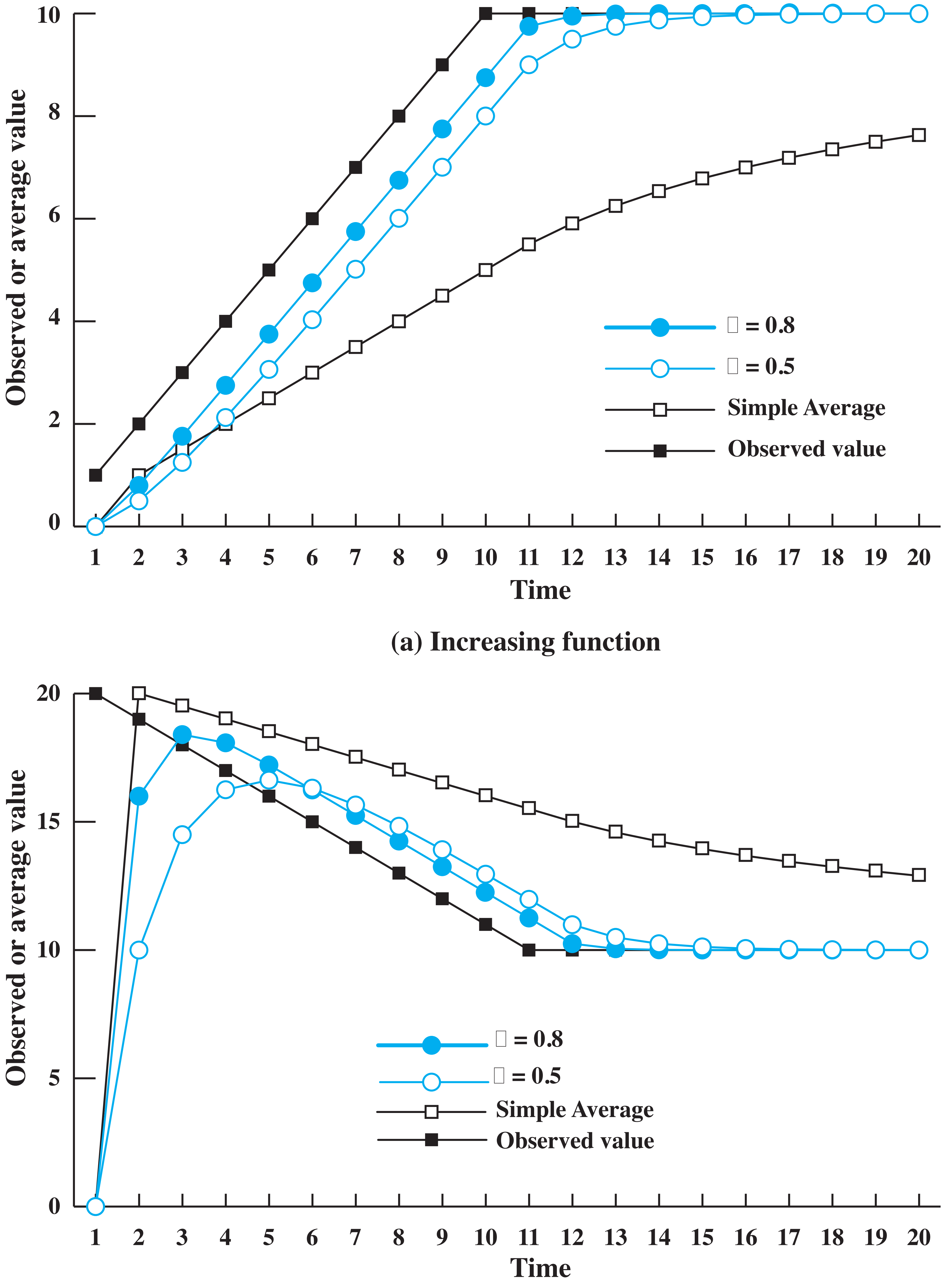


Figure 9.8 Exponential Smoothing Coefficients



(a) Increasing function

(b) Decreasing function

Figure 9.9 Use of Exponential Averaging

Shortest Remaining Time (SRT)

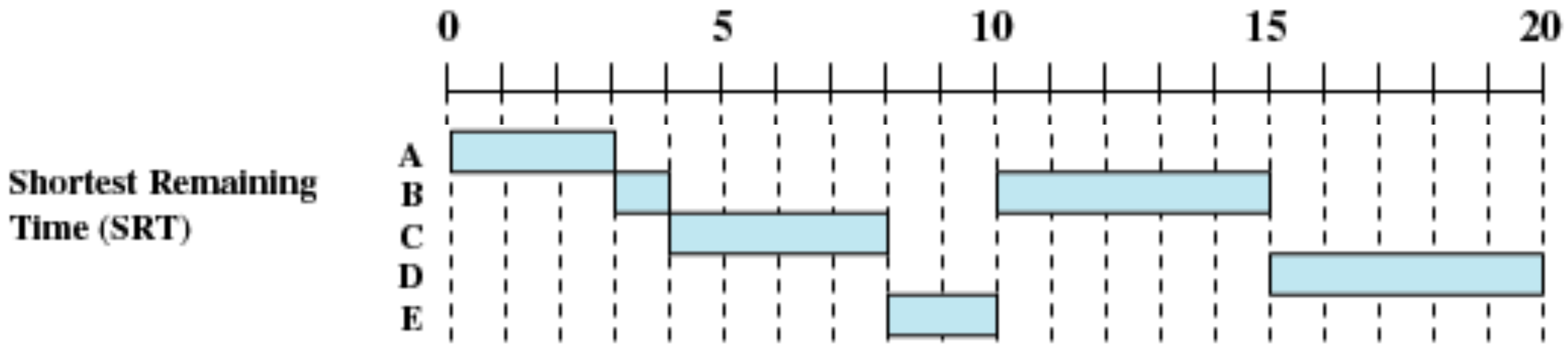
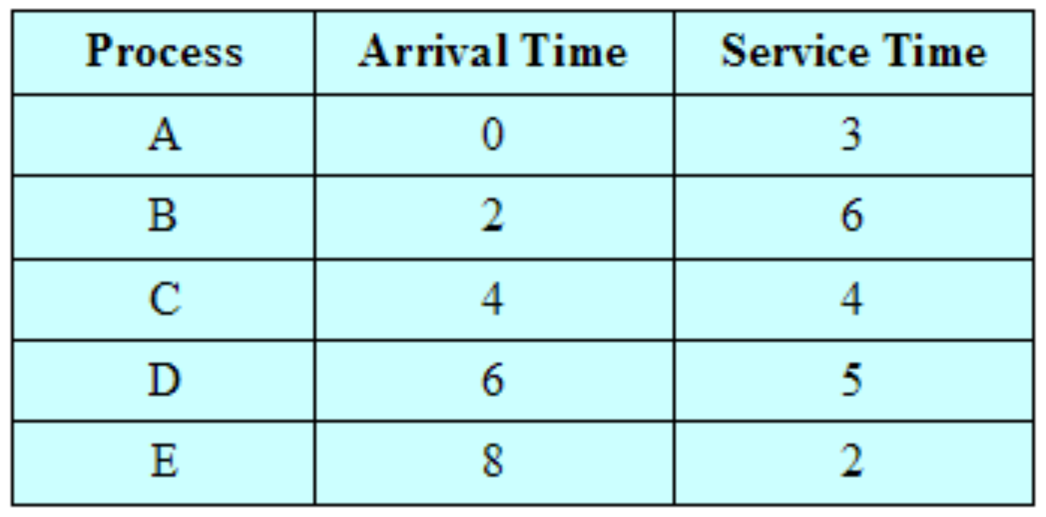


Table 9.4 Process Scheduling Example



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

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

Response Time and Ratio

- Response Ratio *R* is
  - total time spent waiting and executing normalized to the execution time
  - *w*: waiting time (waiting for a processor)
  - *s*: expected service (execution) time
  - Note: In scheduling theory response time is called **flow time** $F_i = C_i - r_i$
    - i.e., completion time minus ready time
    - this is the sum of waiting and processing times

Highest Response Ratio Next (HRRN)

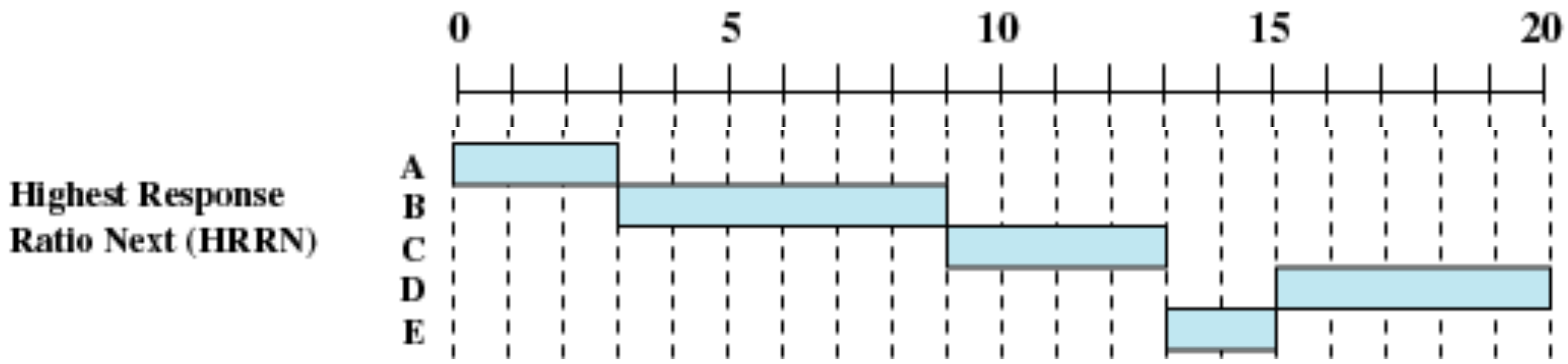
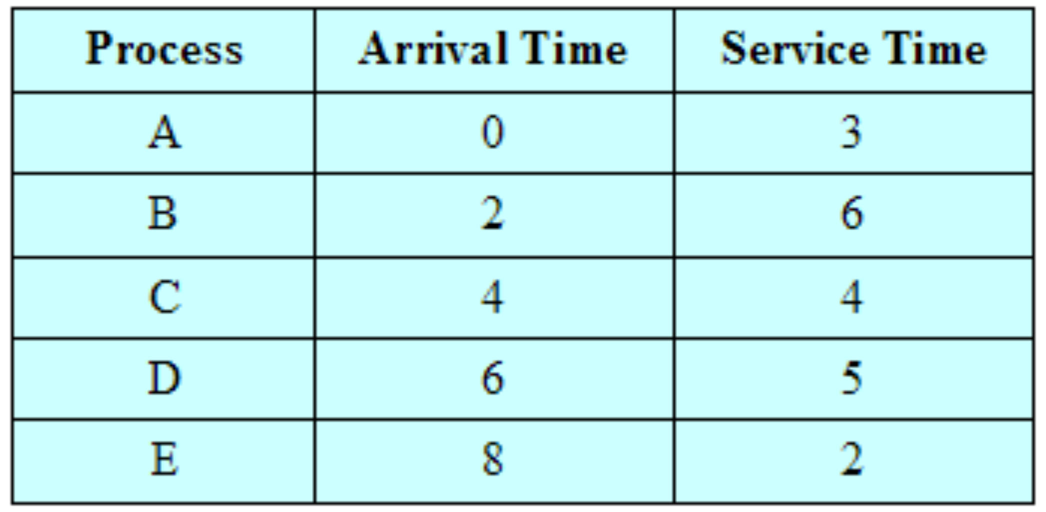


Table 9.4 Process Scheduling Example



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

- Choose next process with the greatest response ratio

Feedback

- SPN, SRT and HRRN require that something is known about the execution times
  - e.g., expected execution time
- Alternative policies
  - give preference to shorter tasks by penalizing tasks that have been running longer

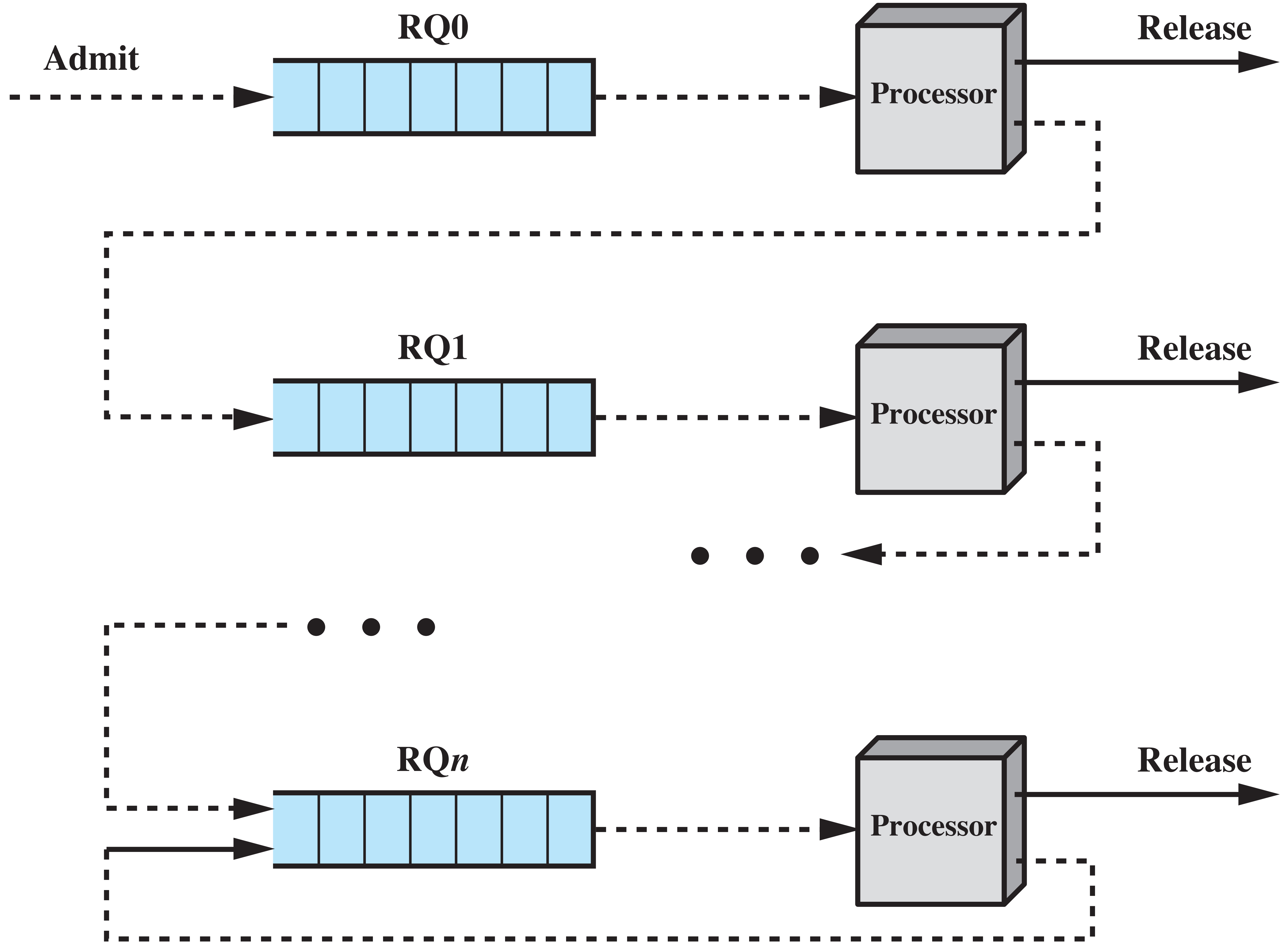
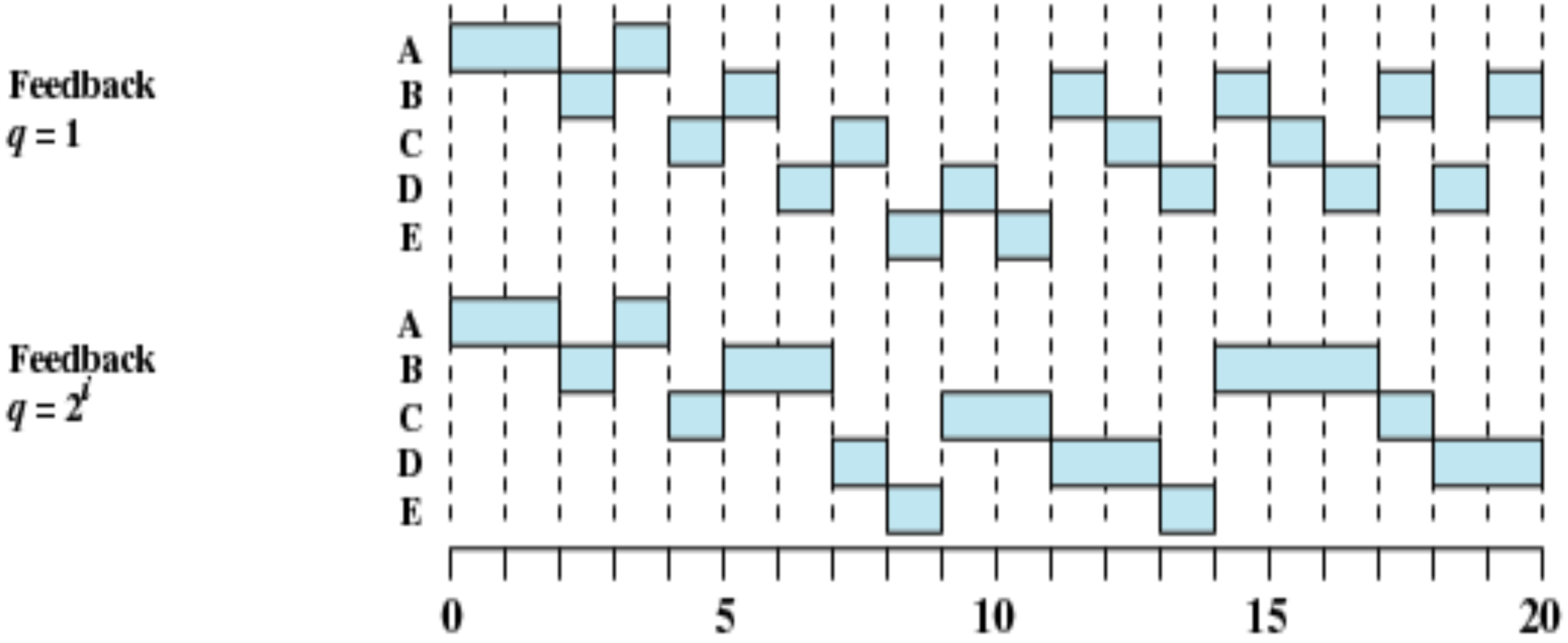


Figure 9.10 Feedback Scheduling

Feedback

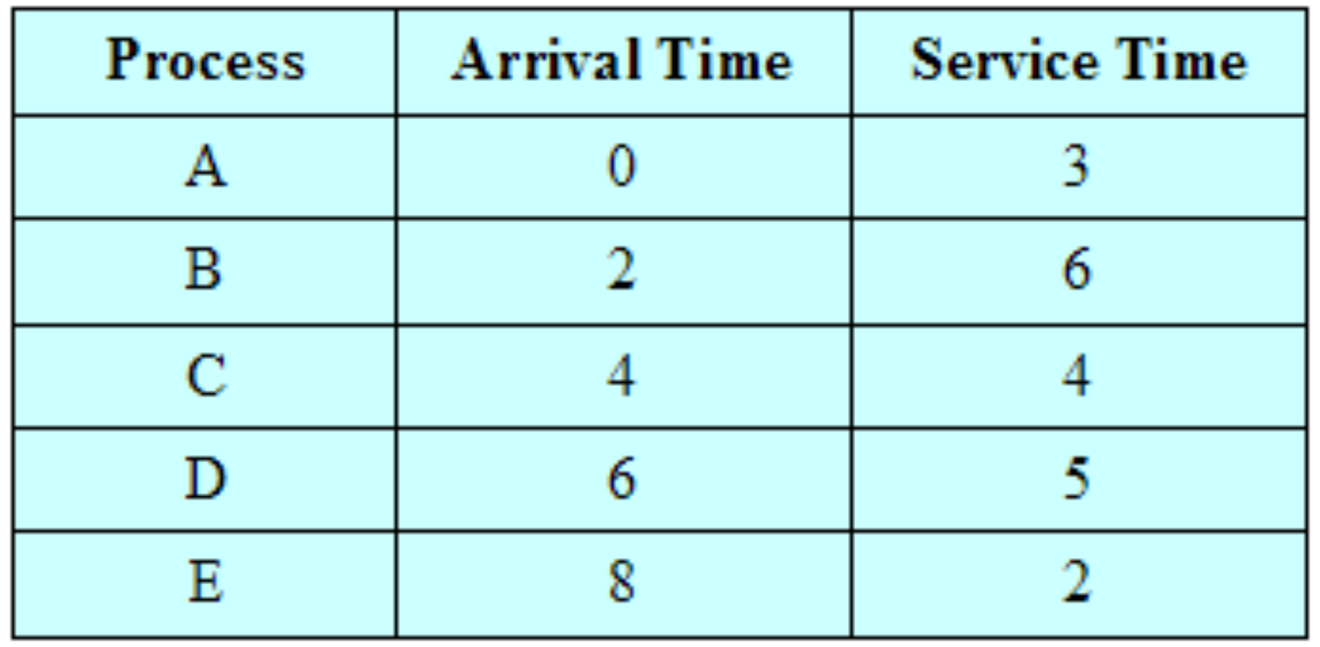
- Potential problems
  - starvation
  - low response times for longer tasks
  - many solutions exists, e.g.,
    - use fixed quantum
      - $q = 1$
    - use different quantum in consequent queues
      - $q = 2^i$ for queue i
      - starvation still possible though
        - » solution: “promote” jobs to higher queue after some time

Feedback



- Don’t know remaining time process needs to execute

Table 9.4 Process Scheduling Example



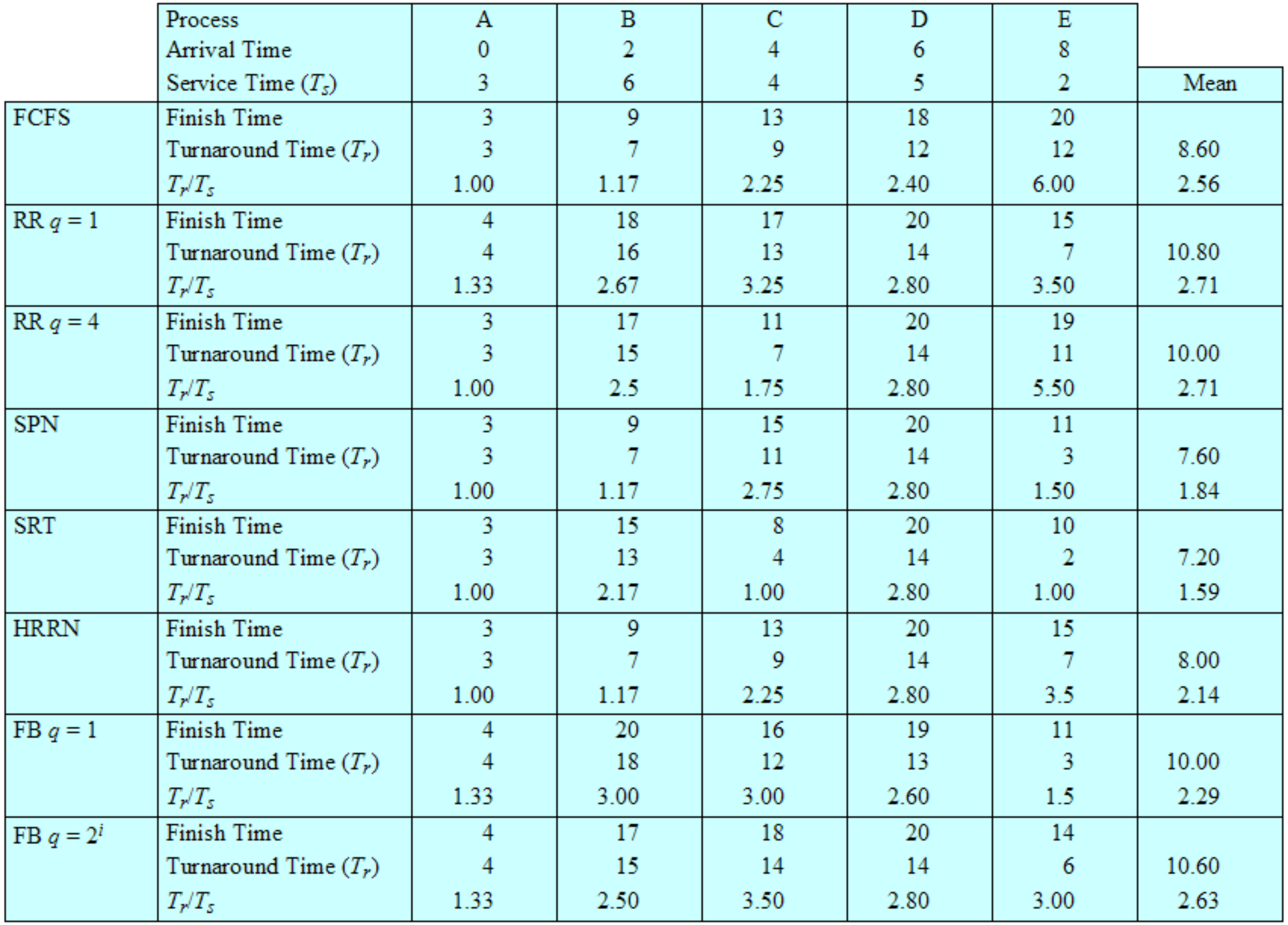
Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Table 9.3 Characteristics of Various Scheduling Policies

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

w = time spent waiting
e = time spent in execution so far
s = total service time required by the process, including e

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
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

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-in-first-out 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

$\lambda = \lambda_1 + \lambda_2$	arrival rate
$\rho_1 = \lambda_1 T_{s1};\ \rho_2 = \lambda_2 T_{s2}$ $\rho = \rho_1 + \rho_2$	utilization
$T_s = \frac{\lambda_1}{\lambda} T_{s1} + \frac{\lambda_2}{\lambda} T_{s2}$	average service time
$T_r = \frac{\lambda_1}{\lambda} T_{r1} + \frac{\lambda_2}{\lambda} T_{r2}$	turnaround time

b) No interrupts; exponential service times

$$T_{r1} = T_{s1} + \frac{\rho_1 T_{s1} + \rho_2 T_{s2}}{1 - \rho_1}$$

$$T_{r2} = T_{s2} + \frac{T_{r1} - T_{s1}}{1 - \rho}$$

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

$$T_{r1} = T_{s1} + \frac{\rho_1 T_{s1}}{1 - \rho_1}$$

$$T_{r2} = T_{s2} + \frac{1}{1 - \rho_1}\left(\rho_1 T_{s2} + \frac{\rho T_s}{1 - \rho}\right)$$

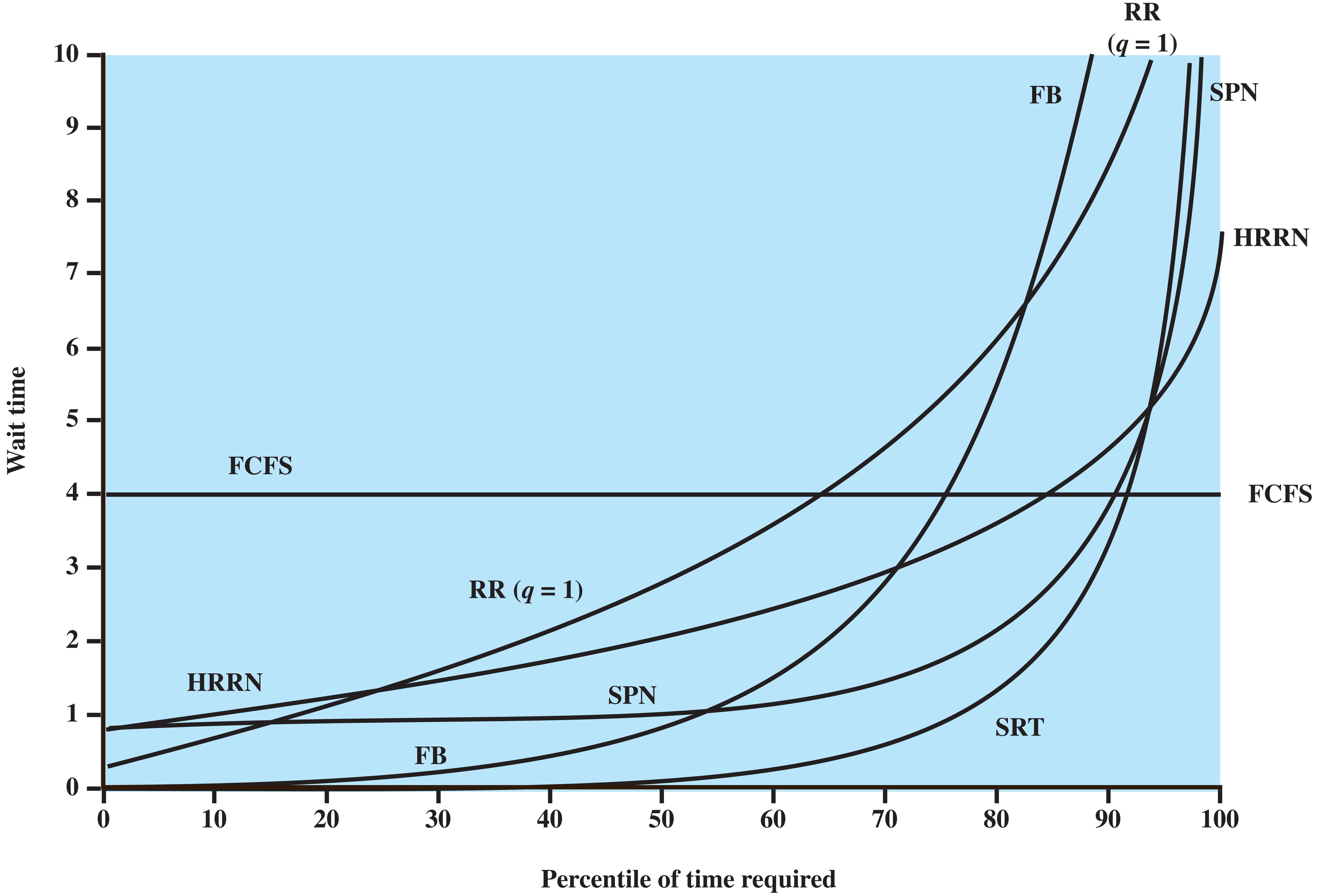


Figure 9.15 Simulation Results for Waiting Time

Fair-Share Scheduling

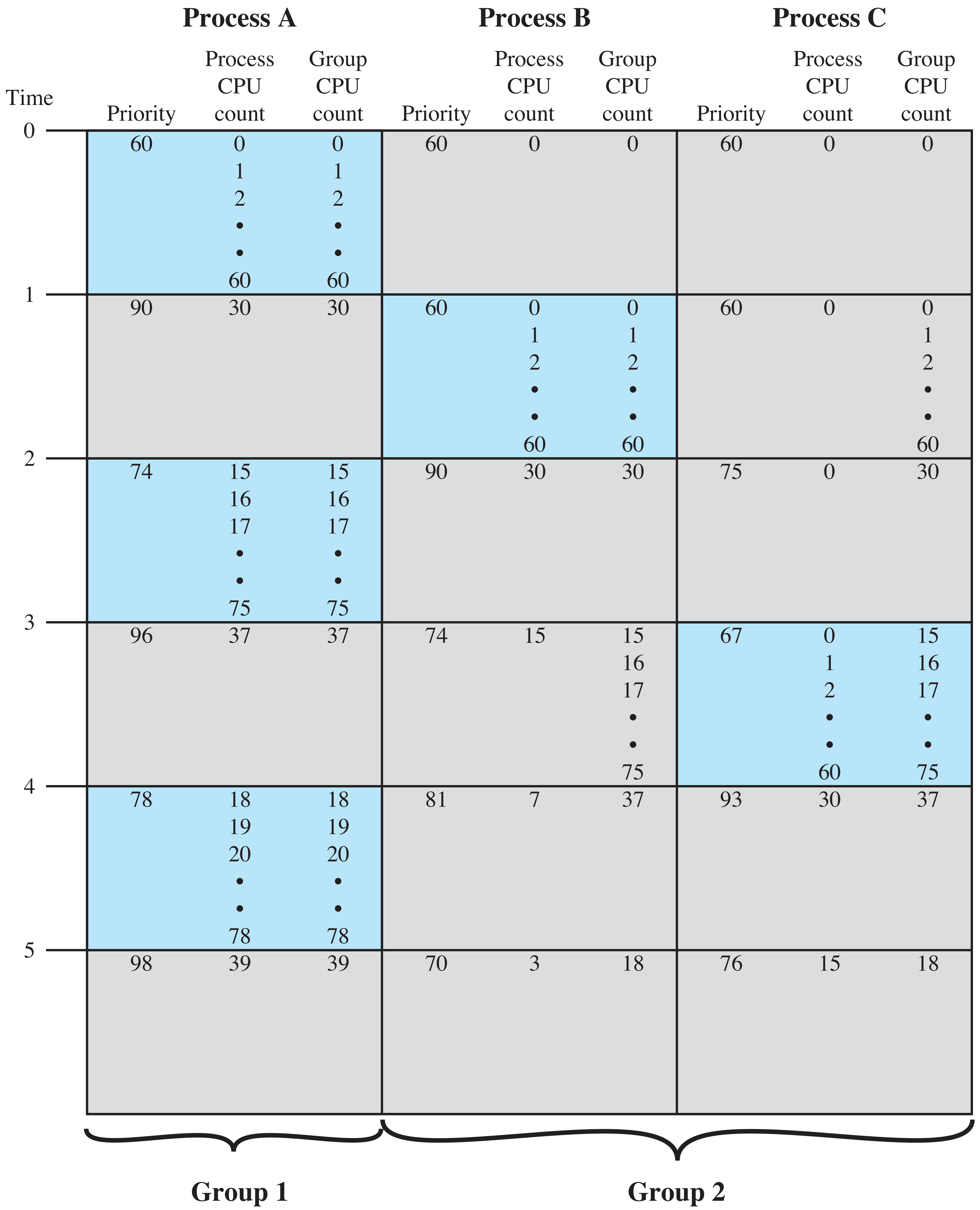
- All previous approaches treat collection of ready processes as single pool
- User's application runs as a collection of processes (threads)
  - concern about the performance of the application, not single process; (this changes the game)
  - need to make scheduling decisions based on process sets

Fair-Share Scheduling

- Philosophy can be extended to groups
  - e.g. time-sharing system,
    - all users from one department treated as group
    - the performance of that group should not affect other groups significantly
      - e.g. as many people from the group log in performance degradation should be primarily felt in that group

Fair-Share Scheduling

- Fair share
  - each user is assigned a weight that corresponds to the fraction of total use of the resources
  - scheme should operate approximately linear
    - e.g. if user A has twice the weight of user B, then (in the long run), user A should do twice the work than B.



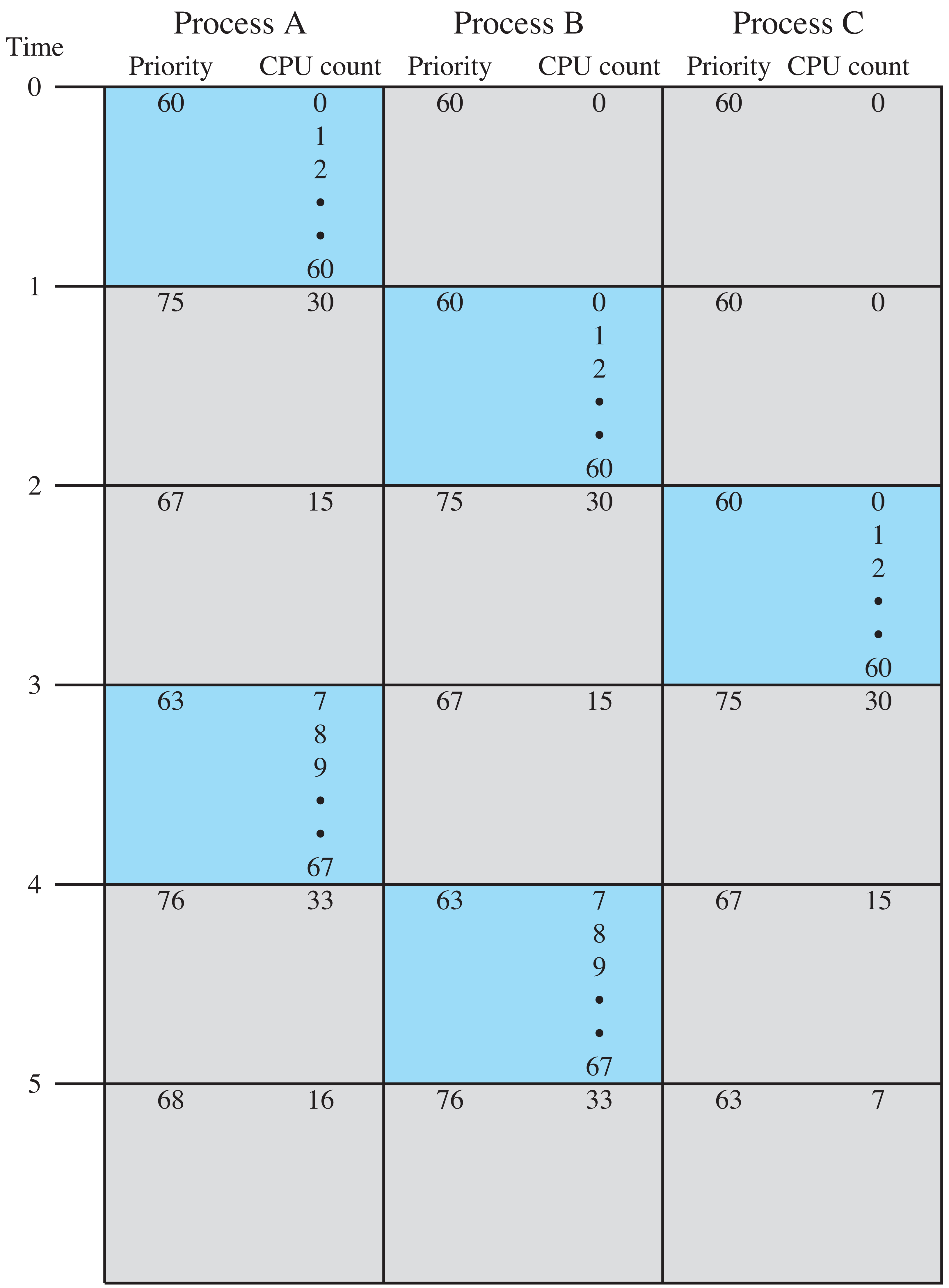
	Process A			Process B			Process C		
Time	Priority	Process CPU count	Group CPU count	Priority	Process CPU count	Group CPU count	Priority	Process CPU count	Group CPU count
0	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0	60	0	0
1	90	30	30	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0 1 2 • • 60
2	74	15 16 17 • • 75	15 16 17 • • 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			Group 2					

Colored rectangle represents executing process

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

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



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