Operating Systems: Internals and Design Principles

Chapter 9 Uniprocessor Scheduling

Ninth Edition
By William Stallings

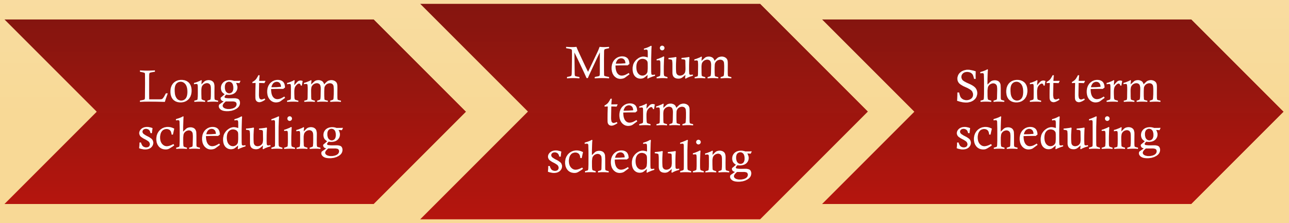
Table 9.1

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

Processor Scheduling

- Aim is to assign processes to be executed by the processor in a way that meets system objectives, such as response time, throughput, and processor efficiency
- Broken down into three separate functions:



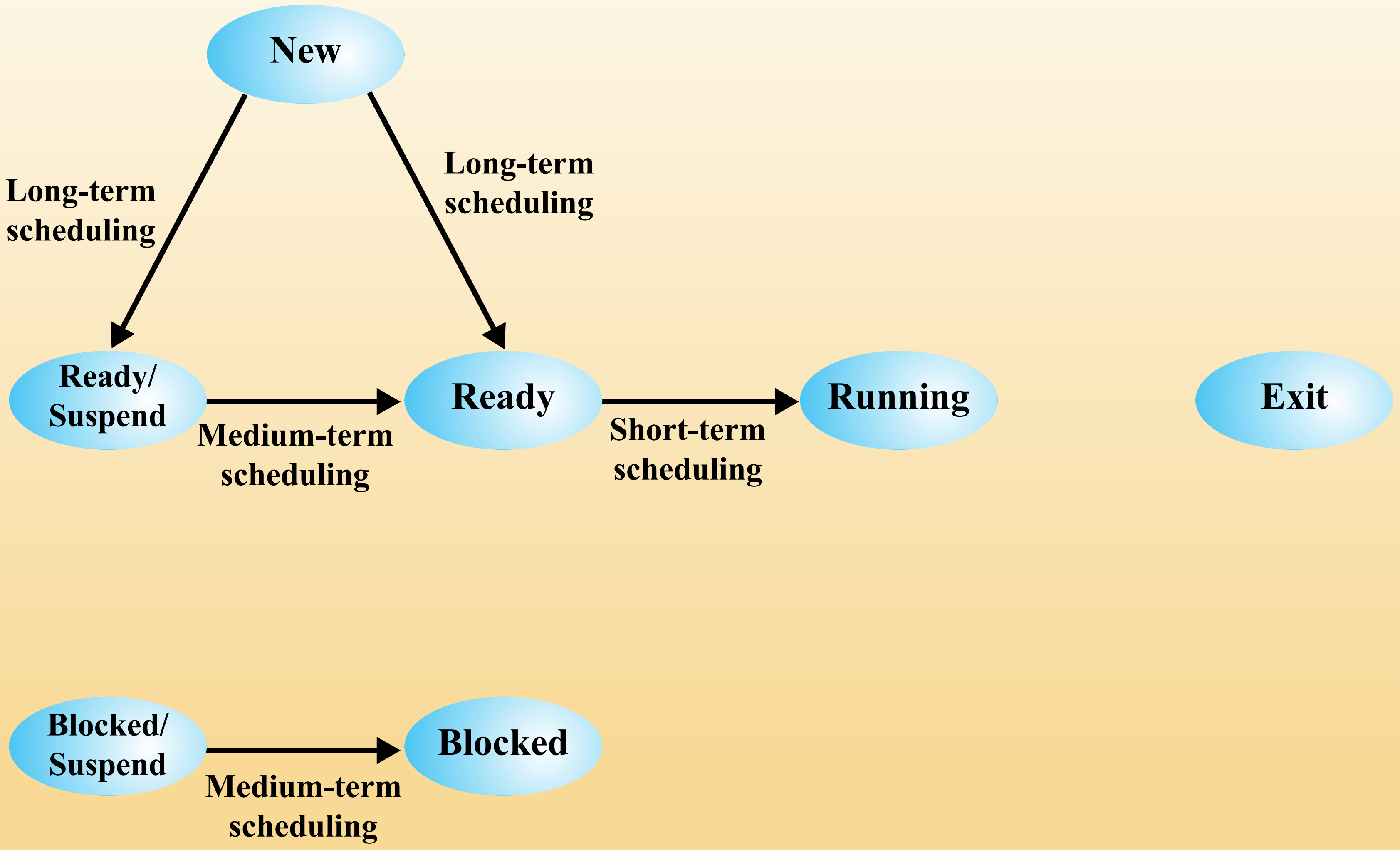


Figure 9.1 Scheduling and Process State Transitions

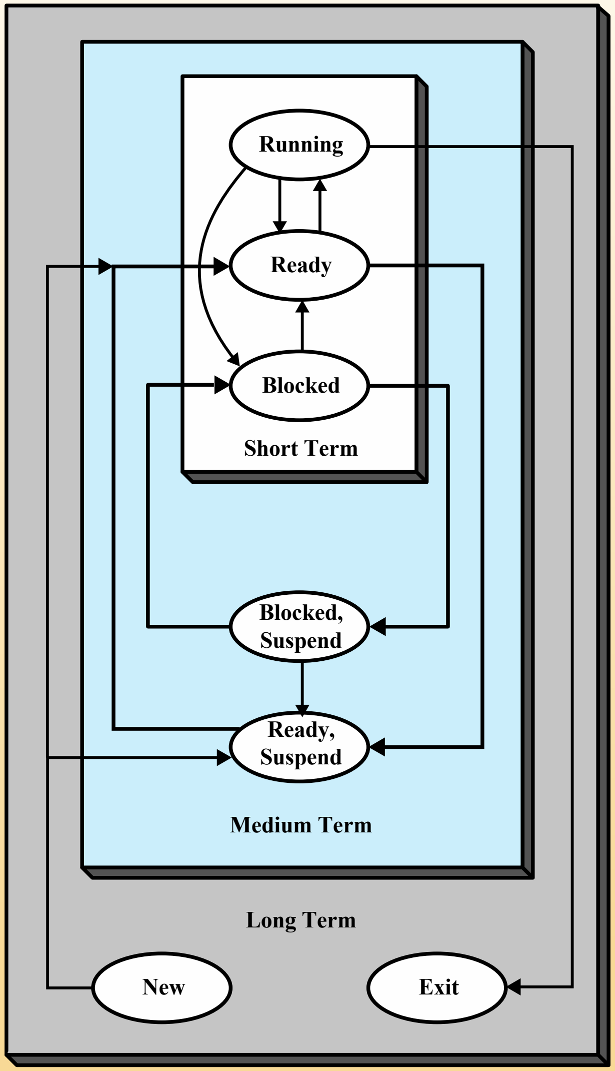


Figure 9.2 Levels of Scheduling

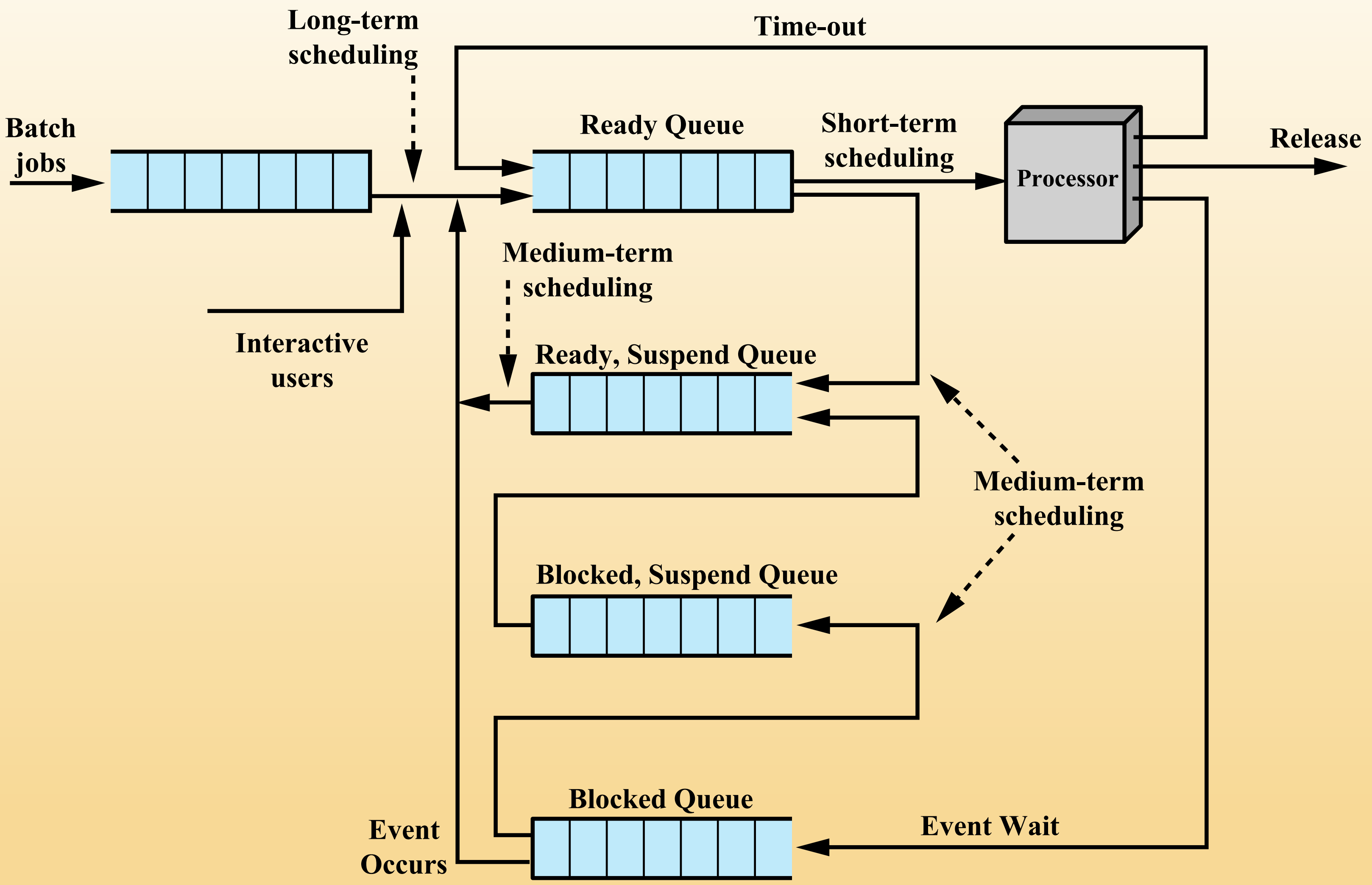
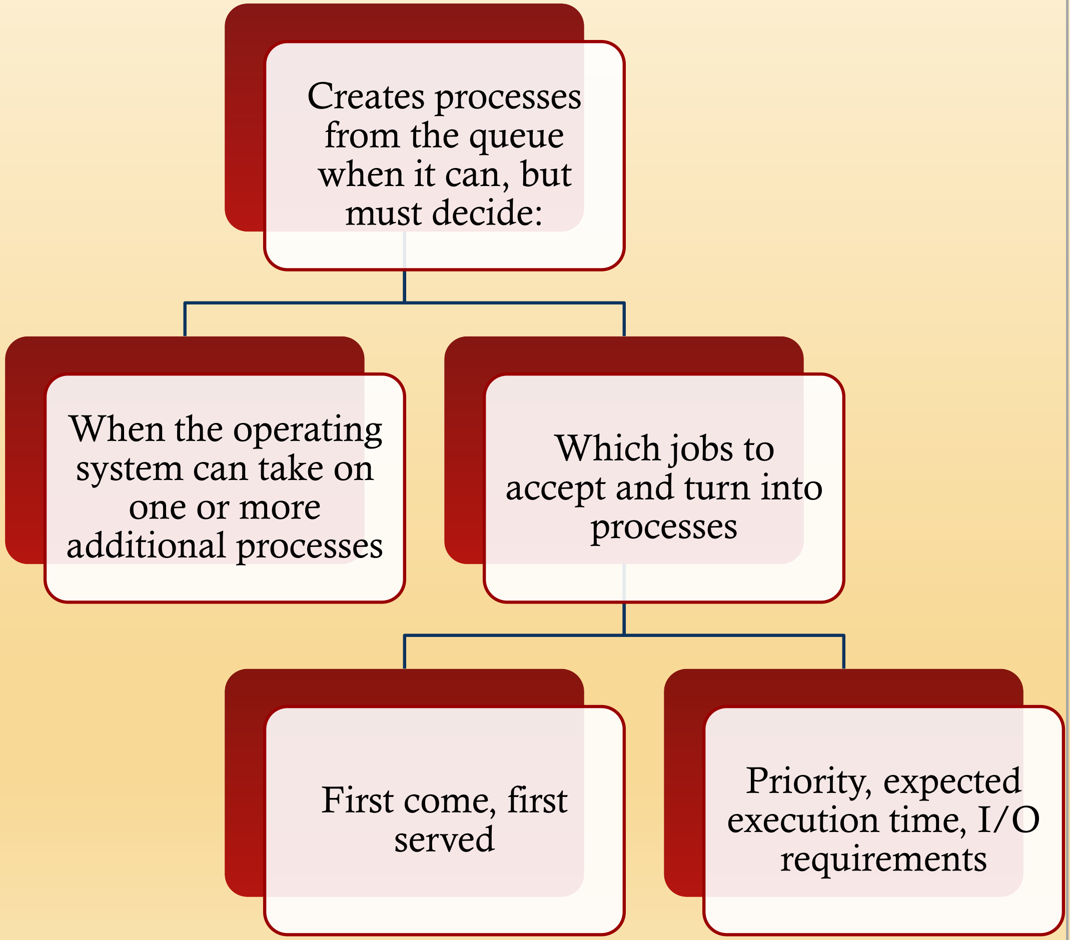


Figure 9.3 Queuing Diagram for Scheduling

Long-Term Scheduler

- Determines which programs are admitted to the system for processing
- Controls the degree of multiprogramming
  - The more processes that are created, the smaller the percentage of time that each process can be executed
  - May limit to provide satisfactory service to the current set of processes



Creates processes from the queue when it can, but must decide:
When the operating system can take on one or more additional processes
Which jobs to accept and turn into processes
First come, first served
Priority, expected execution time, I/O requirements

Medium-Term Scheduling

- Part of the swapping function
- Swapping-in decisions are based on the need to manage the degree of multiprogramming
    - Considers the memory requirements of the swapped-out processes

Short-Term Scheduling

- Known as the dispatcher
- Executes most frequently
- Makes the fine-grained decision of which process to execute next
- Invoked when an event occurs that may lead to the blocking of the current process or that may provide an opportunity to preempt a currently running process in favor of another

Examples:

- Clock interrupts
- I/O interrupts
- Operating system calls
- Signals (e.g., semaphores)

Short Term Scheduling Criteria

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

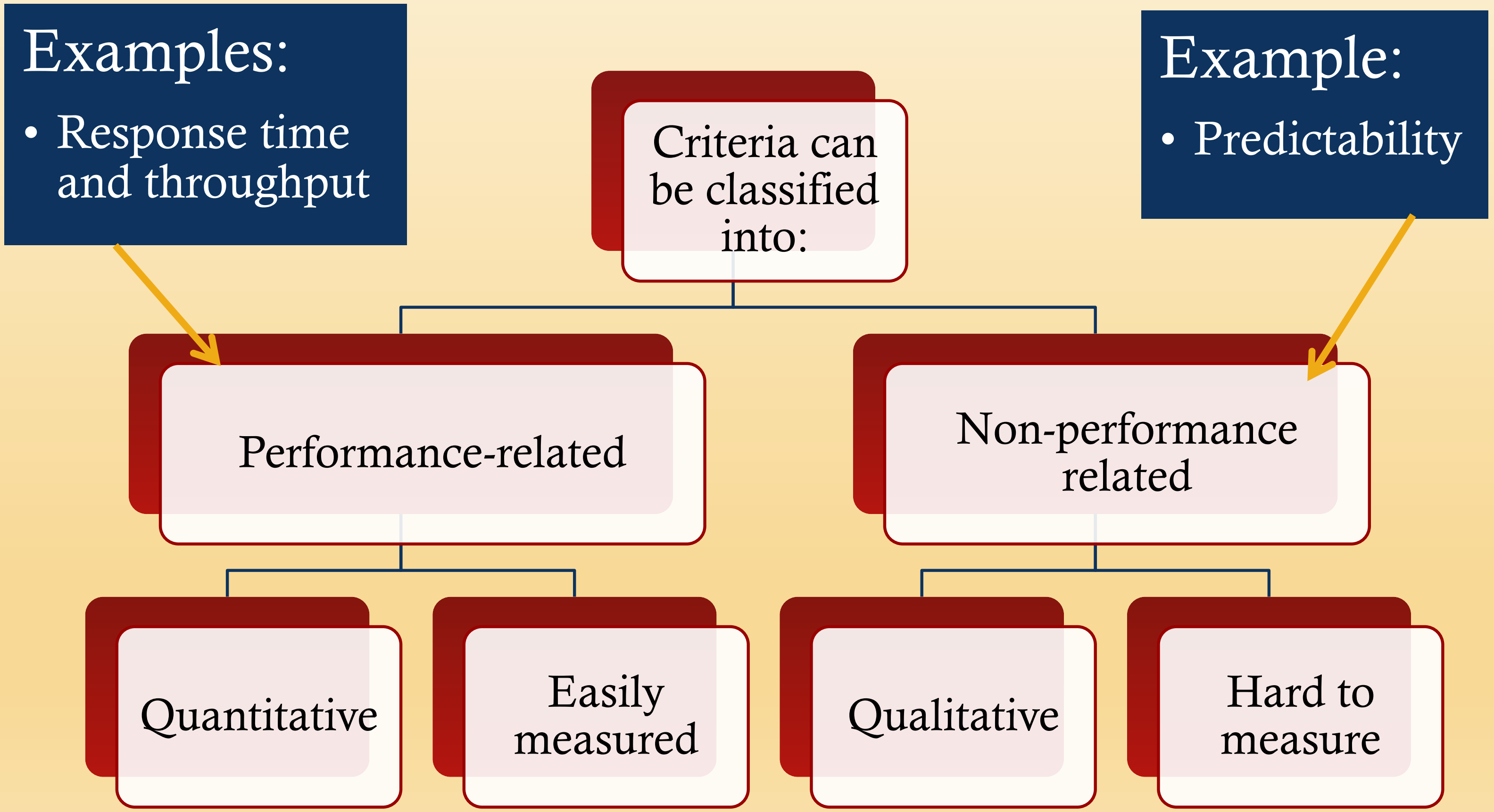
User-oriented criteria

- Relate to the behavior of the system as perceived by the individual user or process (such as response time in an interactive system)
- Important on virtually all systems

System-oriented criteria

- Focus is on effective and efficient utilization of the processor (rate at which processes are completed)
- Generally of minor importance on single-user systems

Short-Term Scheduling Criteria: Performance



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.

Table 9.2 Scheduling Criteria

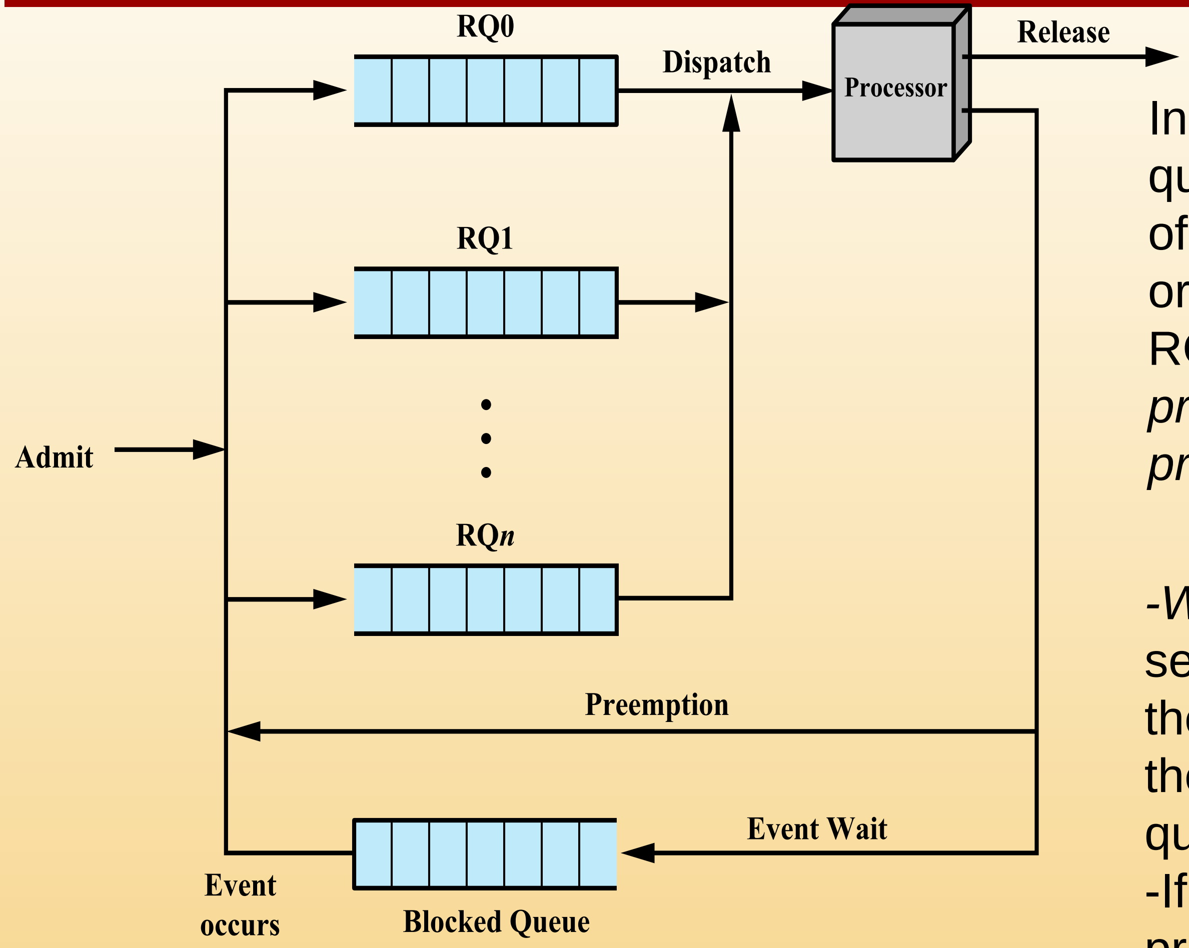


Figure 9.4 Priority Queuing

Instead of a single ready queue, we provide a set of queues, in descending order of priority: RQ0, RQ1, . . . , RQ *n , with priority[RQ i] > priority[RQ j] for i > j .*

-When a scheduling selection is to be made, the scheduler will start at the highest-priority ready queue (RQ0).
-If there are one or more processes in the queue, a process is selected using some scheduling policy. If RQ0 is empty, then RQ1 is examined, and so on.

	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

Table 9.3 Characteristics of Various Scheduling Policies

Selection Function

- Determines which process, among ready processes, is selected next for execution
- May be based on priority, resource requirements, or the execution characteristics of the process
- 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*; generally, this quantity must be estimated or supplied by the user

Decision Mode

- Specifies the instants in time at which the selection function is exercised

- Two categories:
  - Nonpreemptive : Once the resources (CPU cycles) is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state.
  - Preemptive : Interrrupt the running process and switch the CPU into another process.

Nonpreemptive vs Preemptive

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 ready state by the OS
- Decision to preempt may be performed when a new process arrives, when an interrupt occurs that places a blocked process in the Ready state, or periodically, based on a clock interrupt

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

Table 9.4
Process Scheduling Example

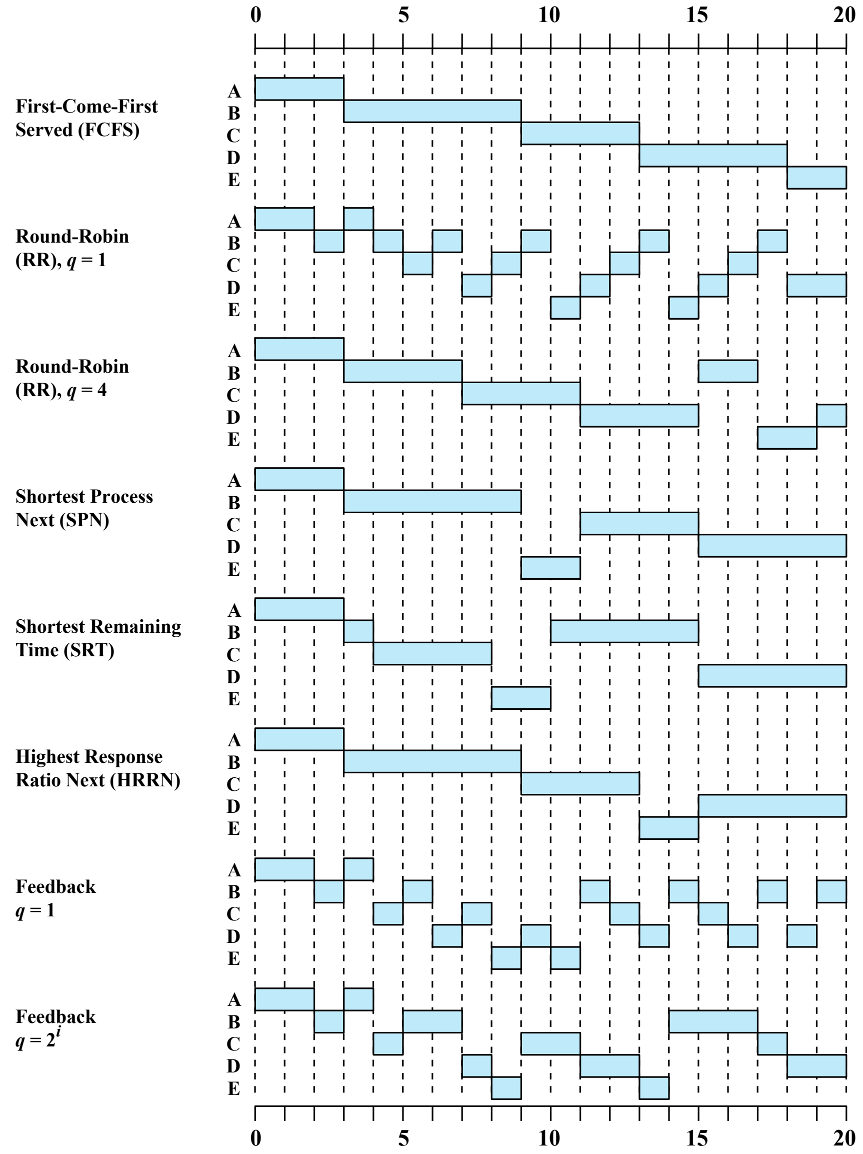


Figure 9.5 A Comparison of Scheduling Policies

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 9.5

A Comparison of Scheduling Policies

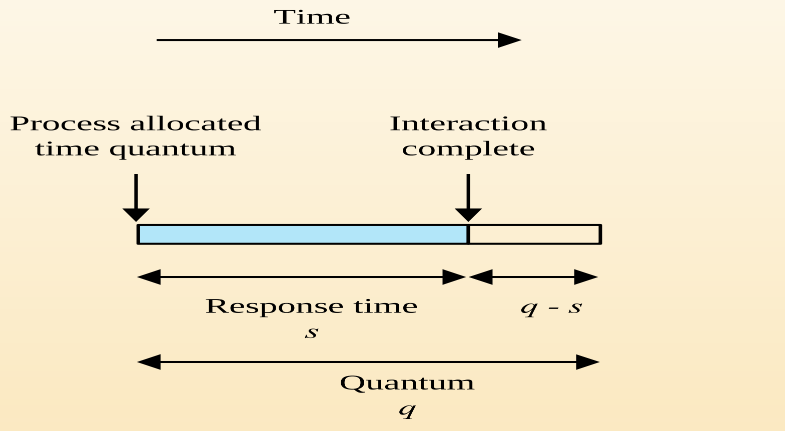
(Table is on page 408 in textbook)

First-Come-First-Served (FCFS)

- Simplest scheduling policy
- Also known as first-in-first-out (FIFO) or a strict queuing scheme
- As each process becomes ready, it joins the ready queue
- When the currently running process ceases to execute, the process that has been in the ready queue the longest is selected for running
- Performs much better for long processes than short ones
- Tends to favor processor-bound processes over I/O-bound processes

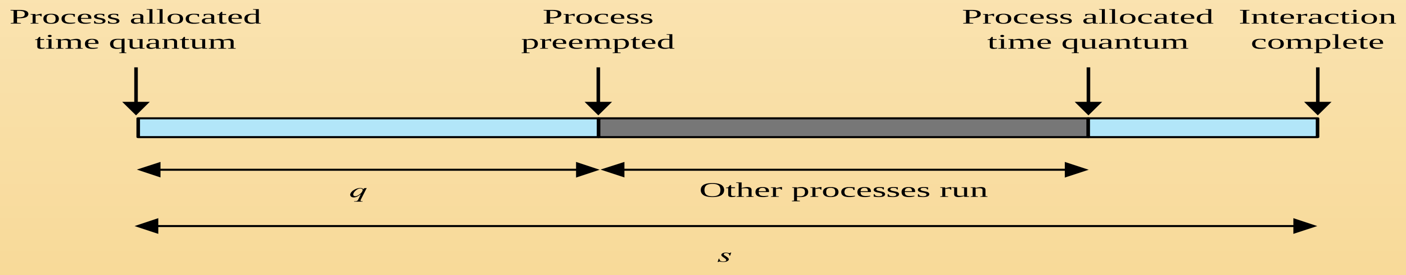
Round Robin

- Uses preemption based on a clock
- A clock interrupt is generated at periodic intervals. When the interrupt occurs, the currently running process is placed in the ready queue, and the next ready job is selected on a FCFS basis.
- Also known as **time slicing** because each process is given a slice of time before being preempted
- Principal design issue is the length of the time quantum, or slice, to be used
- Particularly effective in a general-purpose time-sharing system or transaction processing system
- One drawback is its relative treatment of processor-bound and I/O-bound processes



(a) Time quantum greater than typical interaction

-One useful guide is that the time quantum should be slightly greater than the time required for a typical interaction or process function.
-If it is less, then most processes will require at least two-time quanta.


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

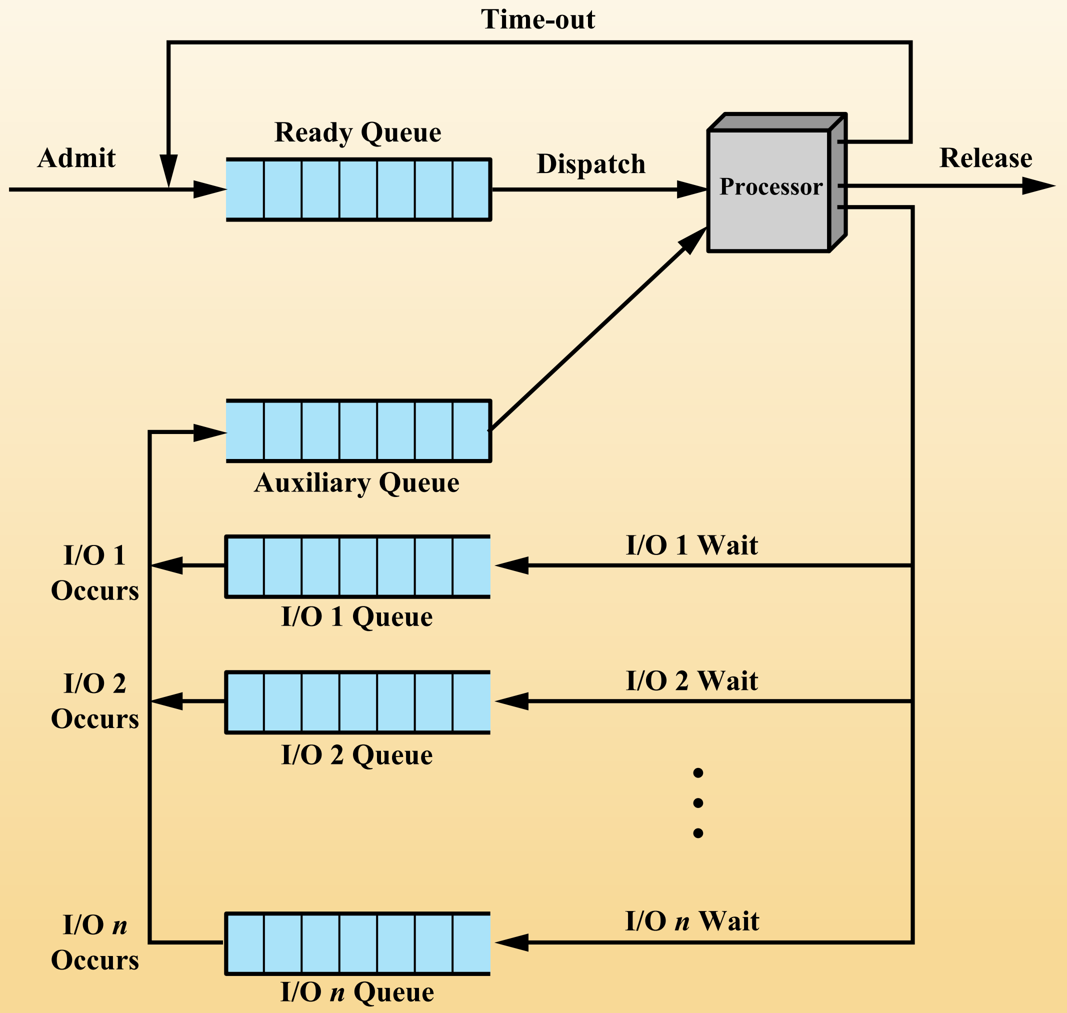


Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler

- [HALD91] suggests a refinement to round robin that he refers to as a virtual round robin (VRR) and that avoids this unfairness.
- The new feature is an FCFS auxiliary queue to which processes are moved after being released from an I/O block.
- When a dispatching decision is to be made, processes in the auxiliary queue get preference over those in the main ready queue.

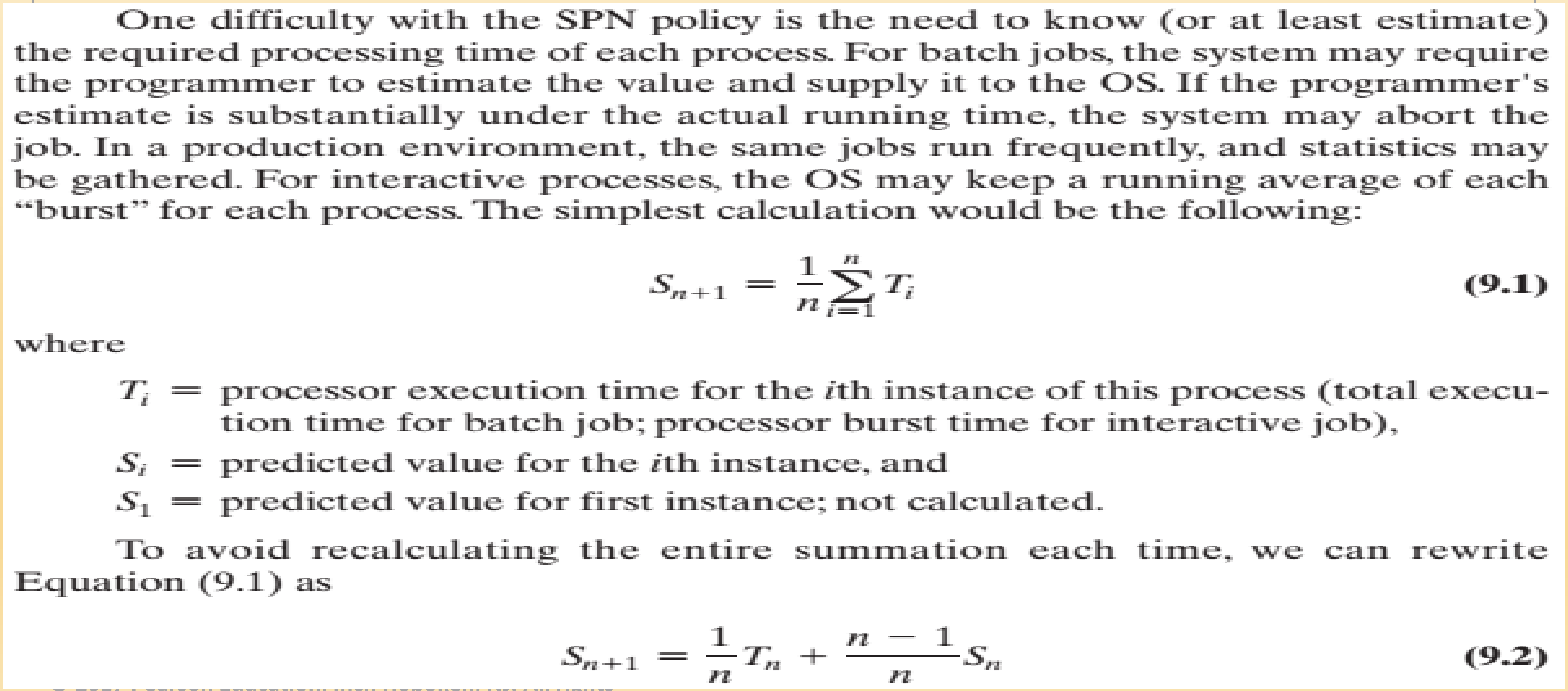
Shortest Process Next (SPN)

- Nonpreemptive policy in which the process with the shortest expected processing time is selected next
- A short process will jump to the head of the queue
- Possibility of starvation for longer processes

- One difficulty is the need to know, or at least estimate, the required processing time of each process
- If the programmer's estimate is substantially under the actual running time, the system may abort the job

SPN

One difficulty with the SPN policy is the need to know (or at least estimate) the required processing time of each process.



One difficulty with the SPN policy is the need to know (or at least estimate) the required processing time of each process. For batch jobs, the system may require the programmer to estimate the value and supply it to the OS. If the programmer's estimate is substantially under the actual running time, the system may abort the job. In a production environment, the same jobs run frequently, and statistics may be gathered. For interactive processes, the OS may keep a running average of each "burst" for each process. The simplest calculation would be the following:

$$S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i \qquad (9.1)$$

where

T_i	=	processor execution time for the *i*th instance of this process (total execution time for batch job; processor burst time for interactive job),
S_i	=	predicted value for the *i*th instance, and
S_1	=	predicted value for first instance; not calculated.

To avoid recalculating the entire summation each time, we can rewrite Equation (9.1) as

$$S_{n+1} = \frac{1}{n}T_n + \frac{n-1}{n}S_n \qquad (9.2)$$

SPN

A common technique for predicting a future value on the basis of a time series of past values is exponential averaging.

$$S_{n+1} = \alpha T_n + (1 - \alpha)S_n$$

where α is a constant weighting factor ($0 < \alpha < 1$) that determines the relative weight given to more recent observations relative to older observations. Compare with Equation (9.2). By using a constant value of α, independent of the number of past observations, Equation (9.3) considers all past values, but the less recent ones have less weight. To see this more clearly, consider the following expansion of Equation (9.3):

$$S_{n+1} = \alpha T_n + (1 - \alpha)\alpha T_{n-1} + \ \dots \ + (1 - \alpha)^i \alpha T_{n-i} + \ \dots \ + (1 - \alpha)^n S_1 \qquad \textbf{(9.4)}$$

Because both α and $(1 - \alpha)$ are less than 1, each successive term in the preceding equation is smaller. For example, for $\alpha = 0.8$, Equation (9.4) becomes

$$S_{n+1} = 0.8T_n + 0.16T_{n-1} + 0.032T_{n-2} + 0.0064T_{n-3} + \ \dots \ + (0.2)^n S_1$$

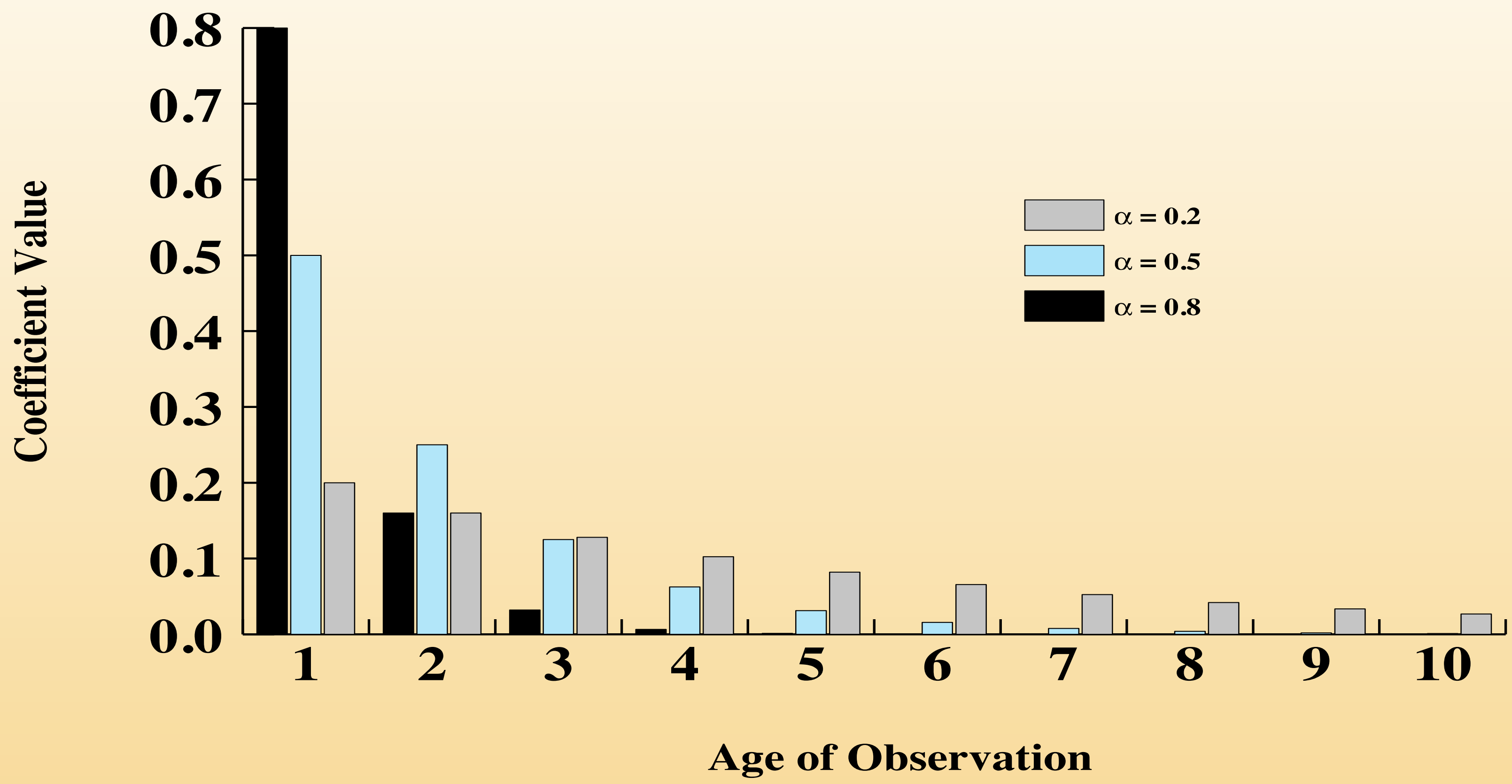
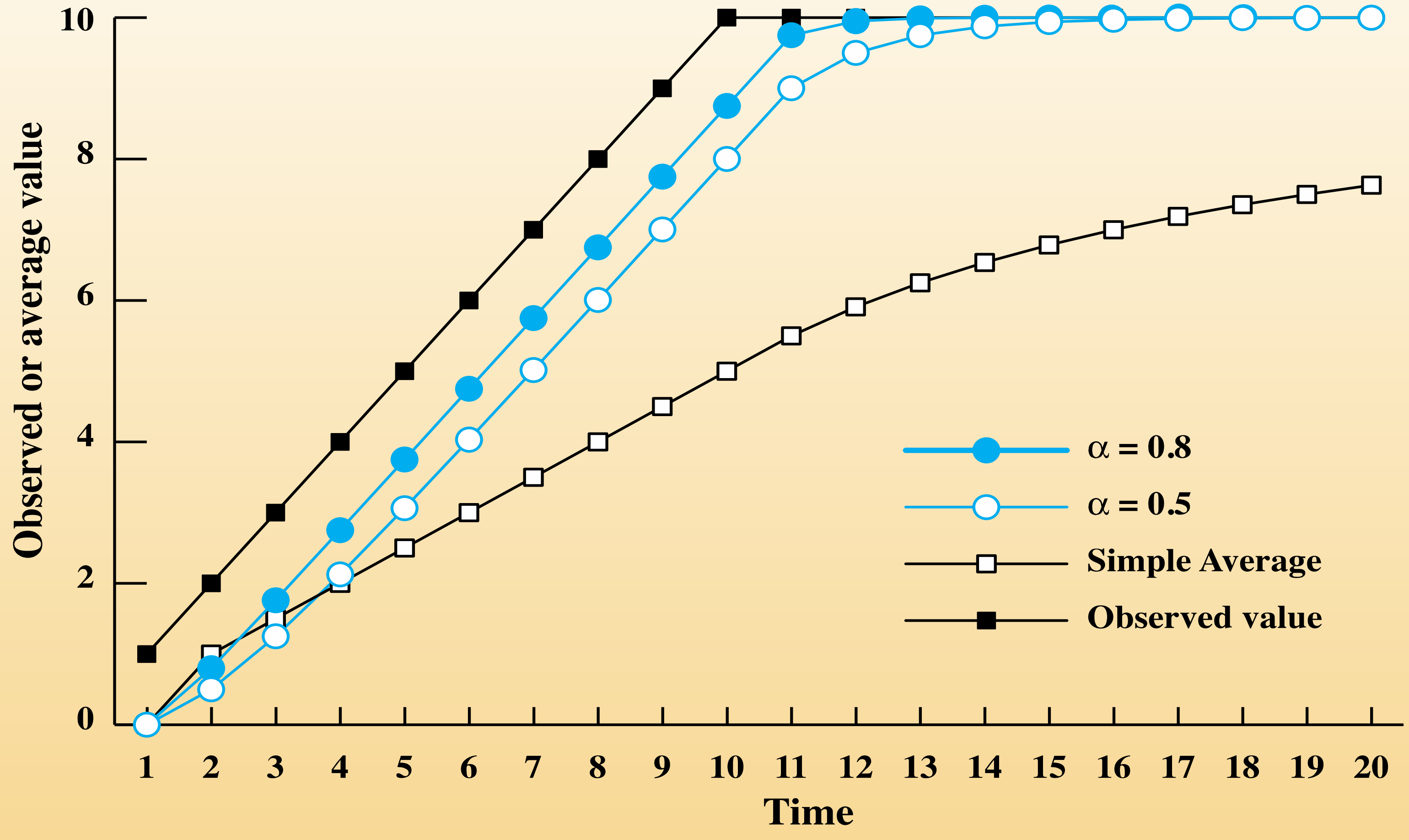


Figure 9.8 Exponential Smoothing Coefficients

The larger the value of α , the greater is the weight given to the more recent observations.



(a) Increasing function

Figure 9.9 compares simple averaging with exponential averaging (for two different values of α). Here (Figure 9.9a), the observed value begins at 1, grows gradually to a value of 10, and then stays there.

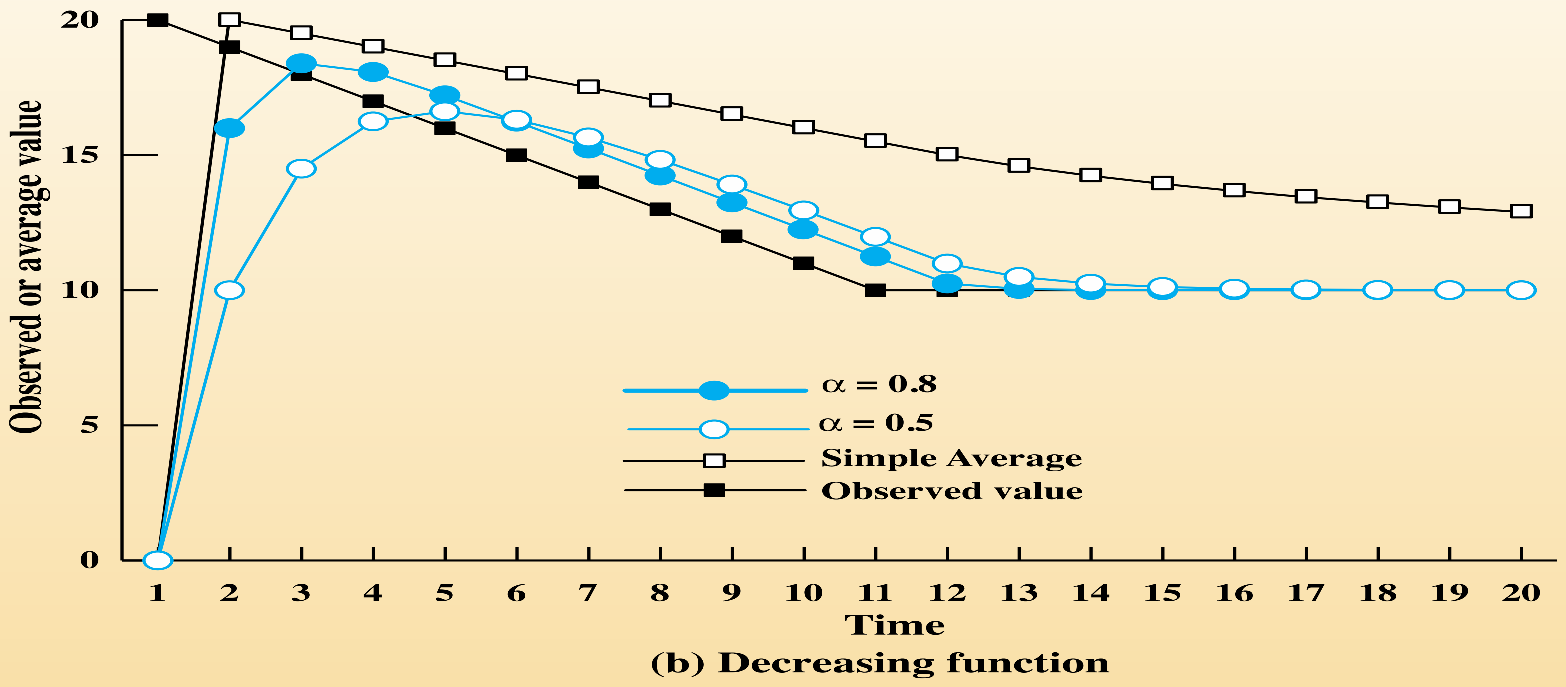


Figure 9.9 Use of Exponential Averaging

While here (Figure 9.9b), the observed value begins at 20, declines gradually to 10, and then stays there.

-Note that exponential averaging tracks changes in process behavior faster than does simple averaging and that the larger value of *α* results in a more rapid reaction to the change in the observed value.

Shortest Remaining Time (SRT)

- Preemptive version of SPN
- Scheduler always chooses the process that has the shortest expected remaining processing time
- Risk of starvation of longer processes
- Should give superior turnaround time performance to SPN because a short job is given immediate preference to a running longer job

Highest Response Ratio Next (HRRN)

- Chooses next process with the greatest ratio
- Attractive because it accounts for the age of the process
- While shorter jobs are favored, aging without service increases the ratio so that a longer process will eventually get past competing shorter jobs

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

Feedback scheduling

If we have no indication of the relative length of various processes, then none of SPN, SRT, and HRRN can be used. Another way of establishing a preference for shorter jobs is to penalize jobs that have been running longer. In other words, if we cannot focus on the time remaining to execute, let us **focus on the time spent in execution so far**.

The way to do this is as follows.

- Scheduling is done on a preemptive (at time quantum) basis, and a dynamic priority mechanism is used.
- When a process first enters the system, it is placed in RQ0 (see Figure 9.4).
- After its first preemption, when it returns to the Ready state, it is placed in RQ1. Each subsequent time that it is preempted, it is demoted to the next lower-priority queue.

A short process will complete quickly, without migrating very far down the hierarchy of ready queues. A longer process will gradually drift downward. Thus, newer, shorter processes are favored over older, longer processes.

Feedback scheduling

Figure 9.10 illustrates the feedback scheduling mechanism by showing the path that a process will follow through the various queues. This approach is known as multilevel feedback , meaning that the operating system allocates the processor to a process and, when the process blocks or is preempted, feeds it back into one of several priority queues.

There are a number of variations on this scheme.

- A simple version is to perform preemption in the same fashion as for round robin: at periodic intervals. Our example shows this (see Figure 9.5 and Table 9.5) for a quantum of one time unit. Note that in this case, the behavior is similar to round robin with a time quantum of q = 1. Even with the allowance for greater time allocation at lower priority, a longer process may still suffer starvation.
- A possible remedy is to promote a process to a higher-priority queue after it spends a certain amount of time waiting for service in its current queue.

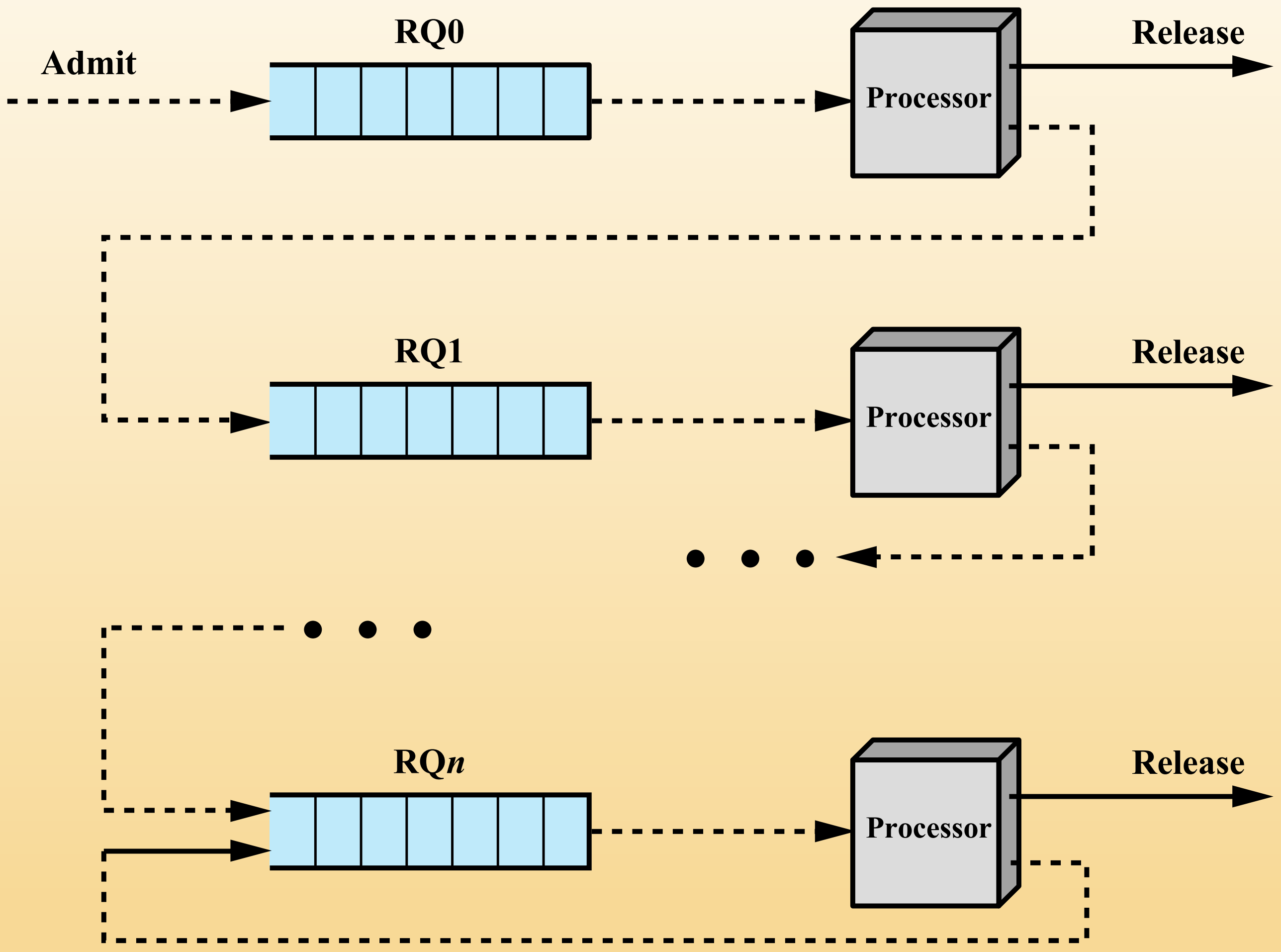
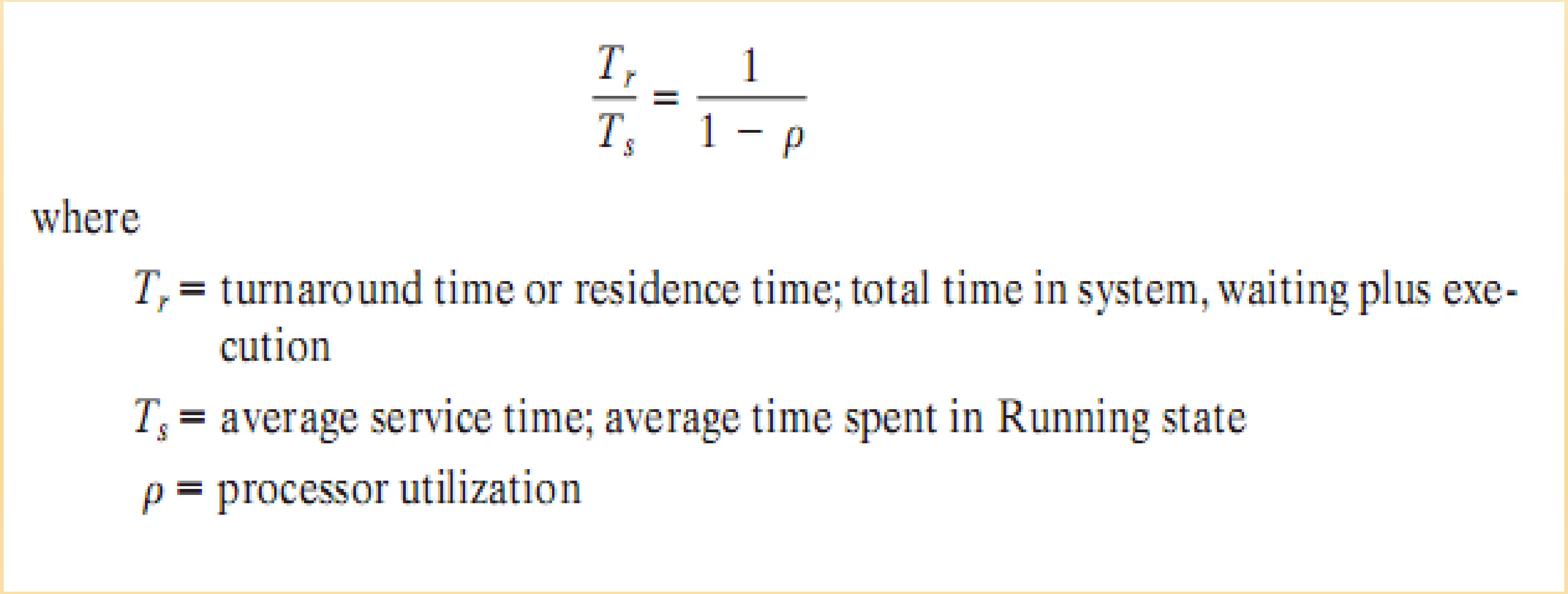


Figure 9.10 Feedback Scheduling

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

ρ = processor utilization

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

$$\lambda = \lambda_1 + \lambda_2$$

$$\rho_1 = \lambda_1 T_{s1}; \quad \rho_2 = \lambda_2 T_{s2}$$

$$\rho = \rho_1 + \rho_2$$

$$T_s = \frac{\lambda_1}{\lambda} T_{s1} + \frac{\lambda_2}{\lambda} T_{s2}$$

$$T_r = \frac{\lambda_1}{\lambda} T_{r1} + \frac{\lambda_2}{\lambda} T_{r2}$$

(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

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

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

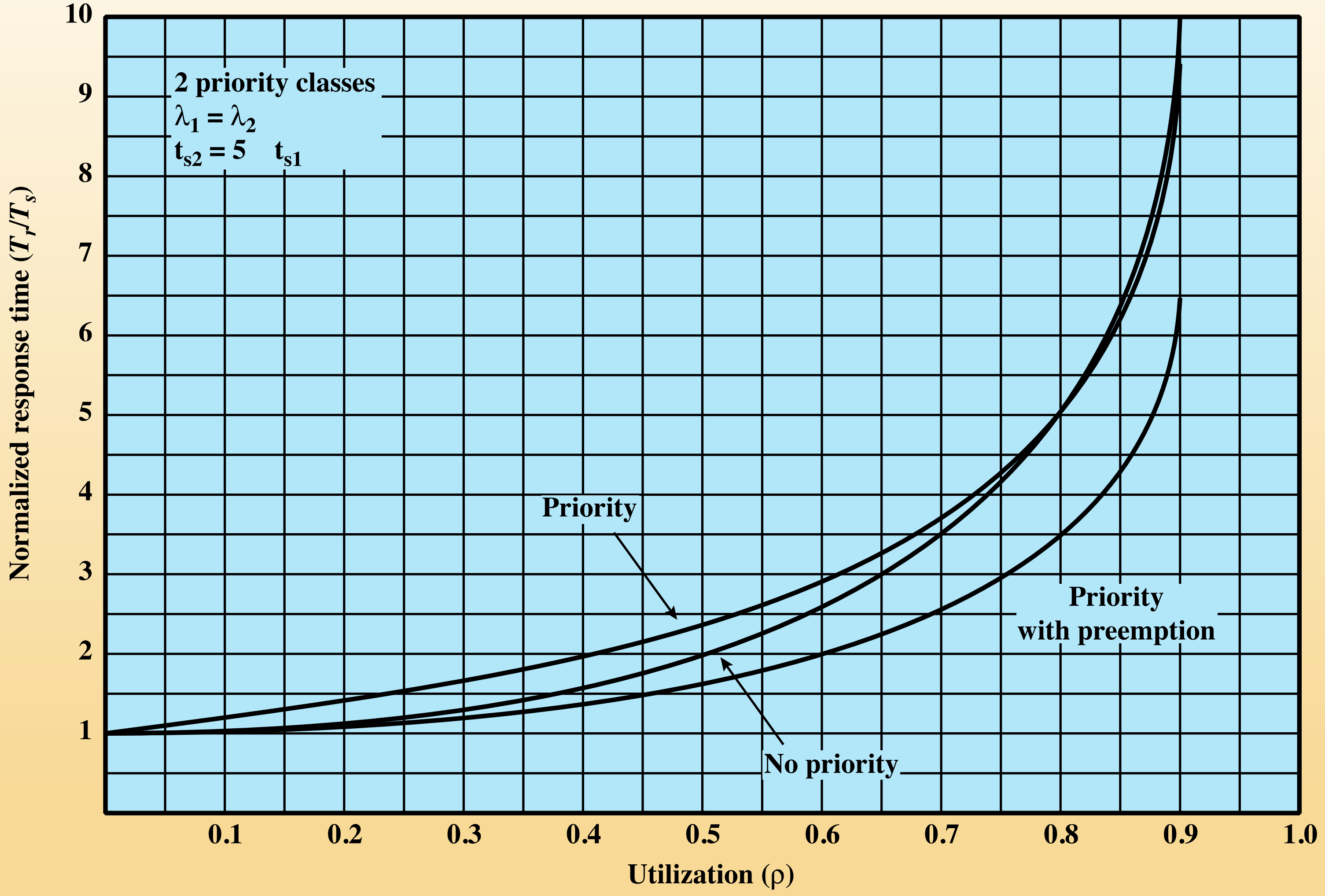


Figure 9.11 Overall Normalized Response Time

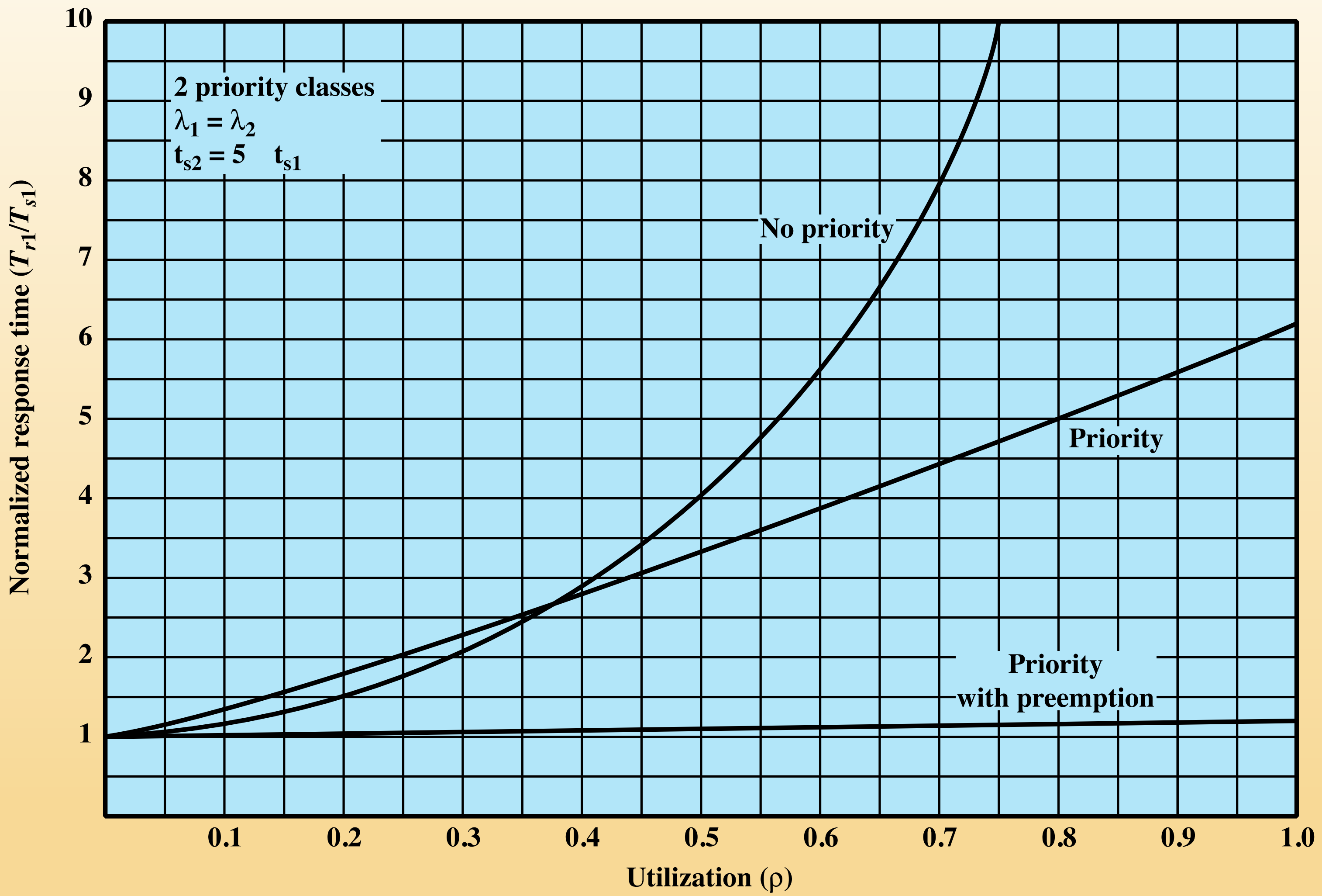


Figure 9.12 Normalized Response Time for Shorter Processes

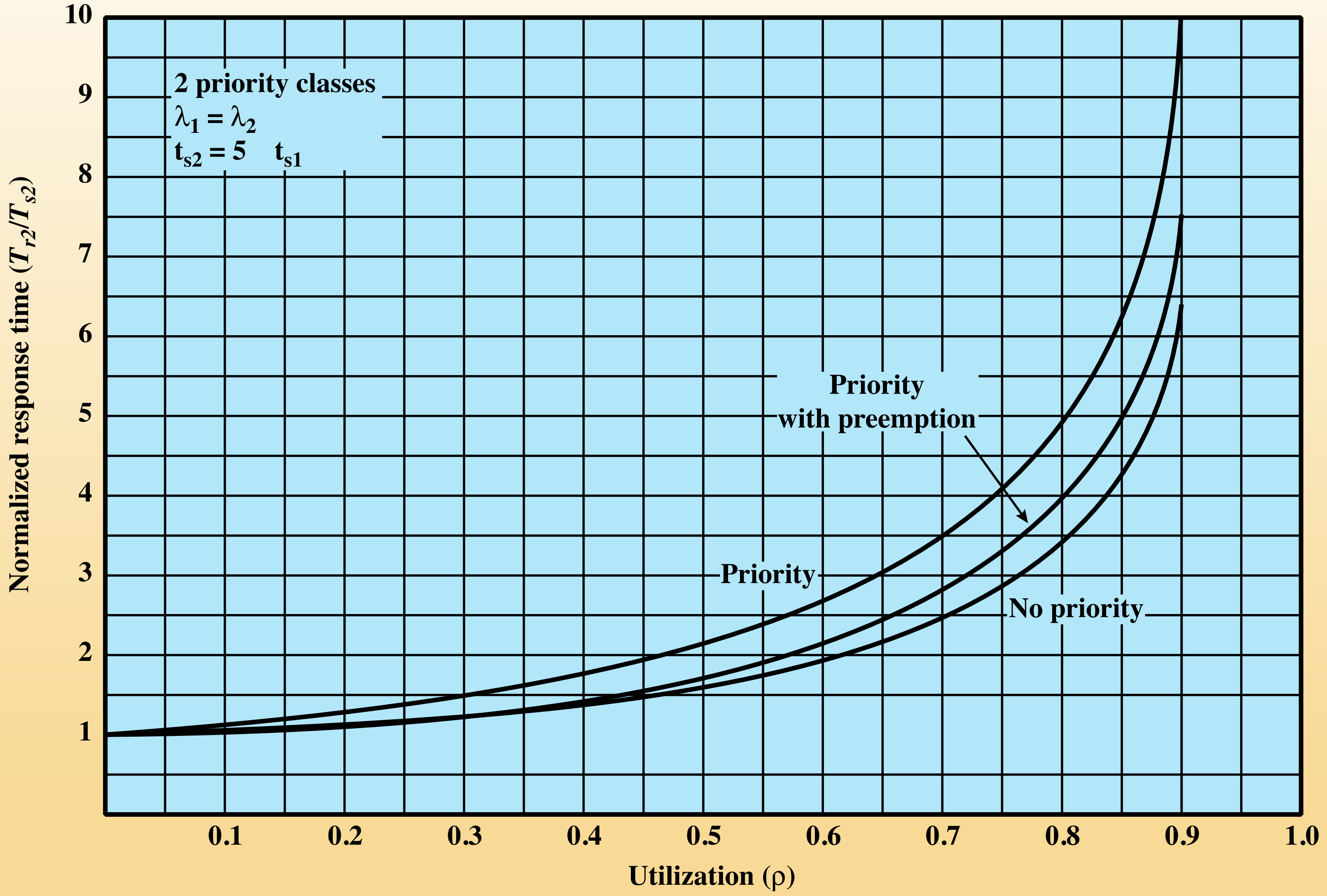


Figure 9.13 Normalized Response Time for Longer Processes

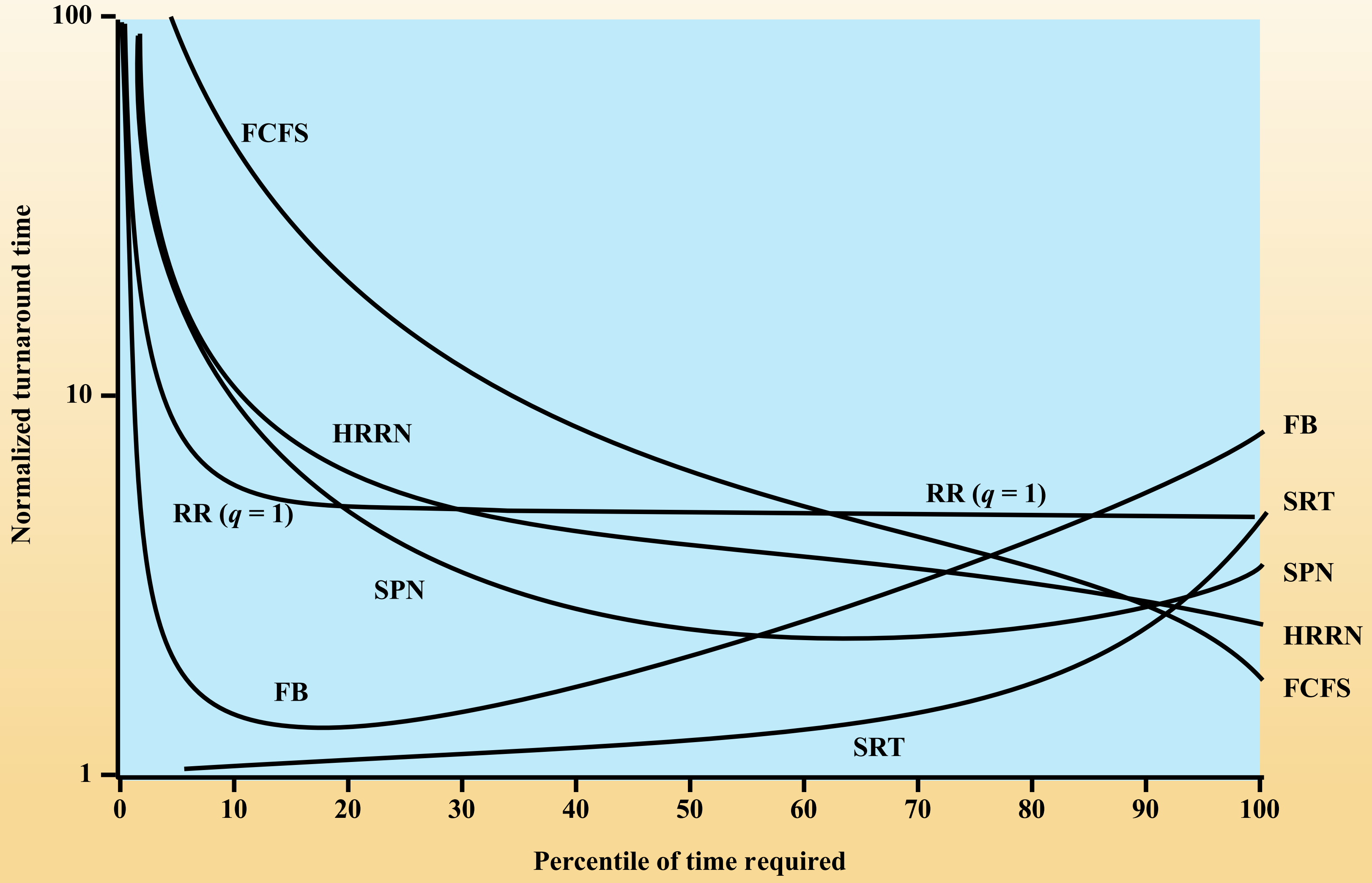


Figure 9.14 Simulation Results for Normalized Turnaround Time

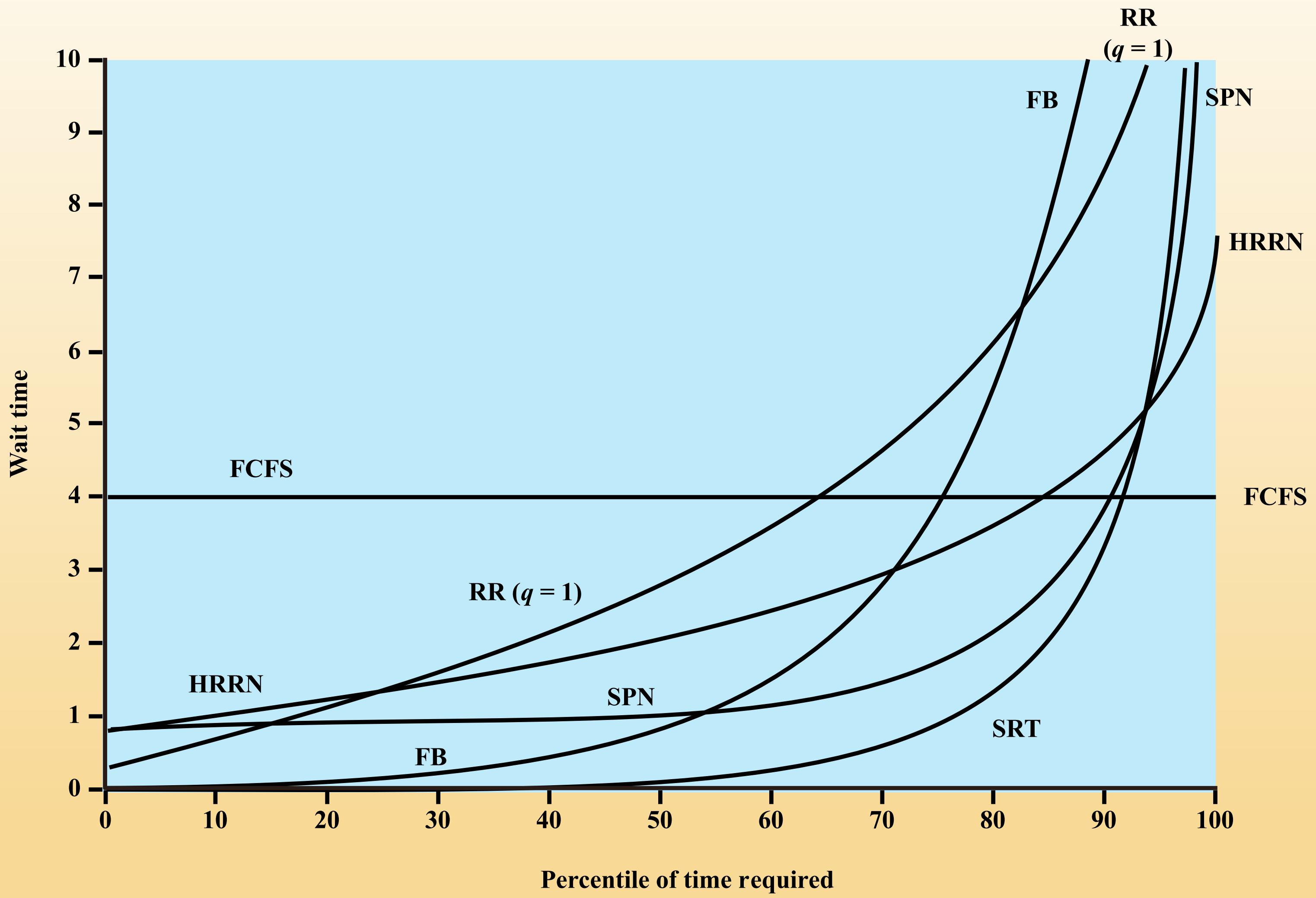


Figure 9.15 Simulation Results for Waiting Time

Fair-Share Scheduling

- Scheduling decisions based on the process sets
- Each user is assigned a share of the processor
- Objective is to monitor usage to give fewer resources to users who have had more than their fair share and more to those who have had less than their fair share

Fair Share Scheduler (FFS)

- The following formulas apply for process *j* in group *k* :

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

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

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + \frac{GCPU_k(i)}{4 \times W_k}$$

where

$CPU_j(i)$	=	measure of processor utilization by process *j* through interval *i*,
$GCPU_k(i)$	=	measure of processor utilization of group *k* 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*, and
W_k	=	weighting assigned to group *k*, with the constraint that and $0 < W_k \leq 1$ and $\sum W_k = 1$.

	Group 1			Group 2					
	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

Colored rectangle represents executing process

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

Traditional UNIX Scheduling

- Used in both SVR3 and 4.3 BSD UNIX
  - These systems are primarily targeted at the time-sharing interactive environment
- Designed to provide good response time for interactive users while ensuring that low-priority background jobs do not starve
- Employs multilevel feedback using round robin within each of the priority queues
- Makes use of one-second preemption
- 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

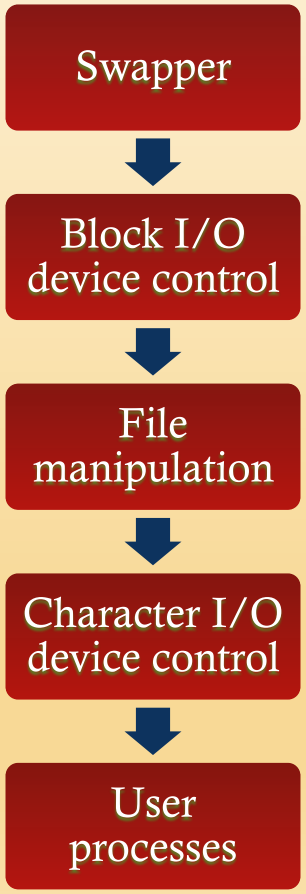
$P_j(i)$ = priority of process j at beginning of interval i; lower values equal higher priorities

$Base_j$ = base priority of process j

$nice_j$ = user-controllable adjustment factor

Bands

- Used to optimize access to block devices and to allow the operating system to respond quickly to system calls
- In decreasing order of priority, the bands are:



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

Summary

- Types of processor scheduling
  - Long-term scheduling
  - Medium-term scheduling
  - Short-term scheduling
- Traditional UNIX scheduling

- Scheduling algorithms
  - Short-term scheduling criteria
  - The use of priorities
  - Alternative scheduling policies
  - Performance comparison
  - Fair-share scheduling