› FOR SOCIETY

Chapter 9 – Uniprocessor Scheduling
Lecture 8

> FOR SOCIETY

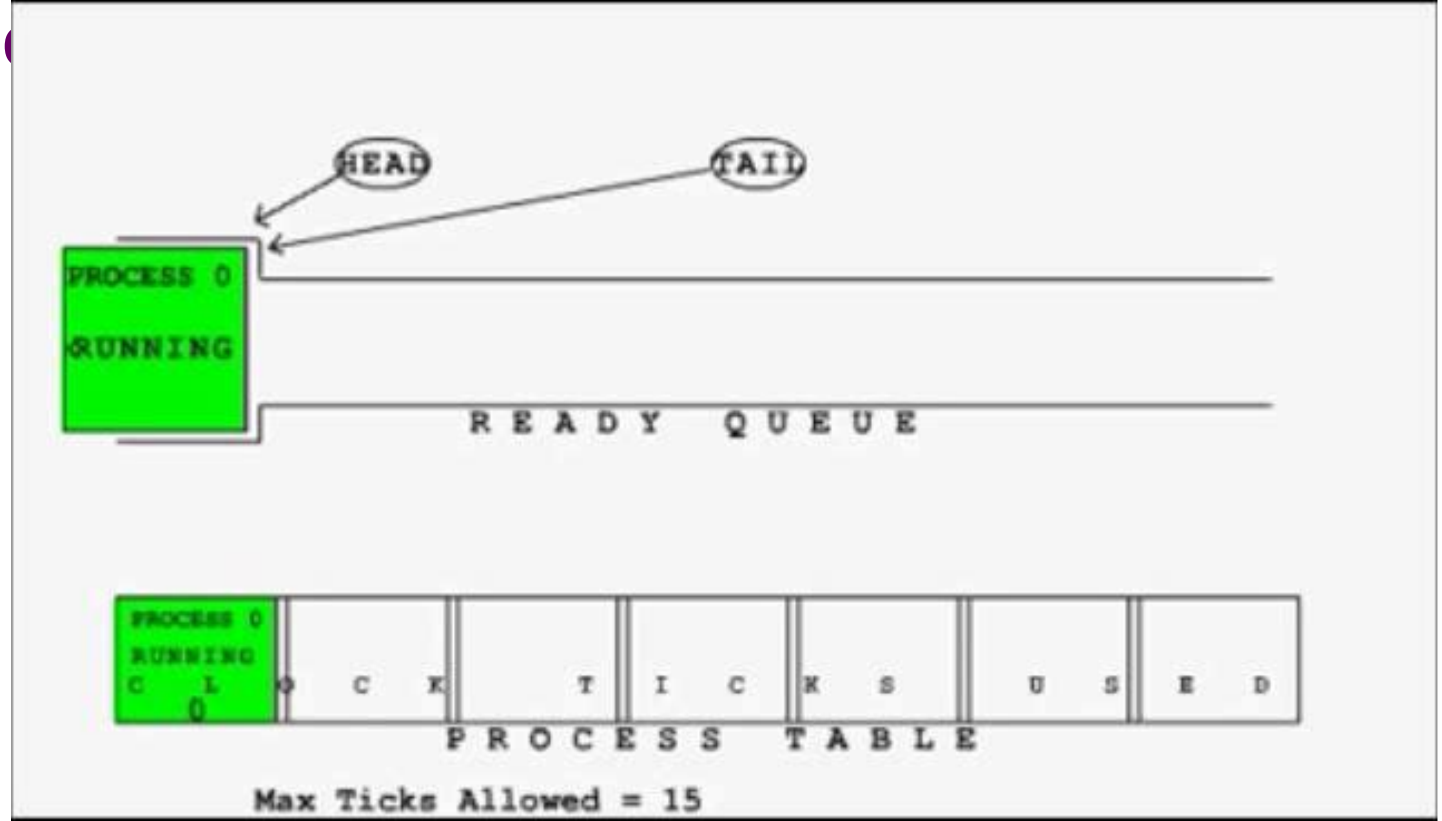
Roadmap

- **Types of Processor Scheduling**
- Scheduling Algorithms
- Traditional UNIX Scheduling

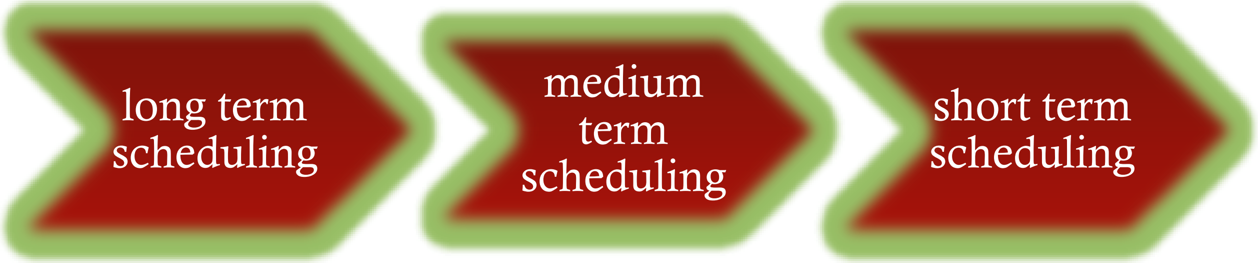
Scheduling

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

Sch


HEAD
TAIL
PROCESS 0
RUNNING
READY QUEUE
PROCESS 0
RUNNING
C L O C K T I C K S U S E D
0
PROCESS TABLE
Max Ticks Allowed = 15

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

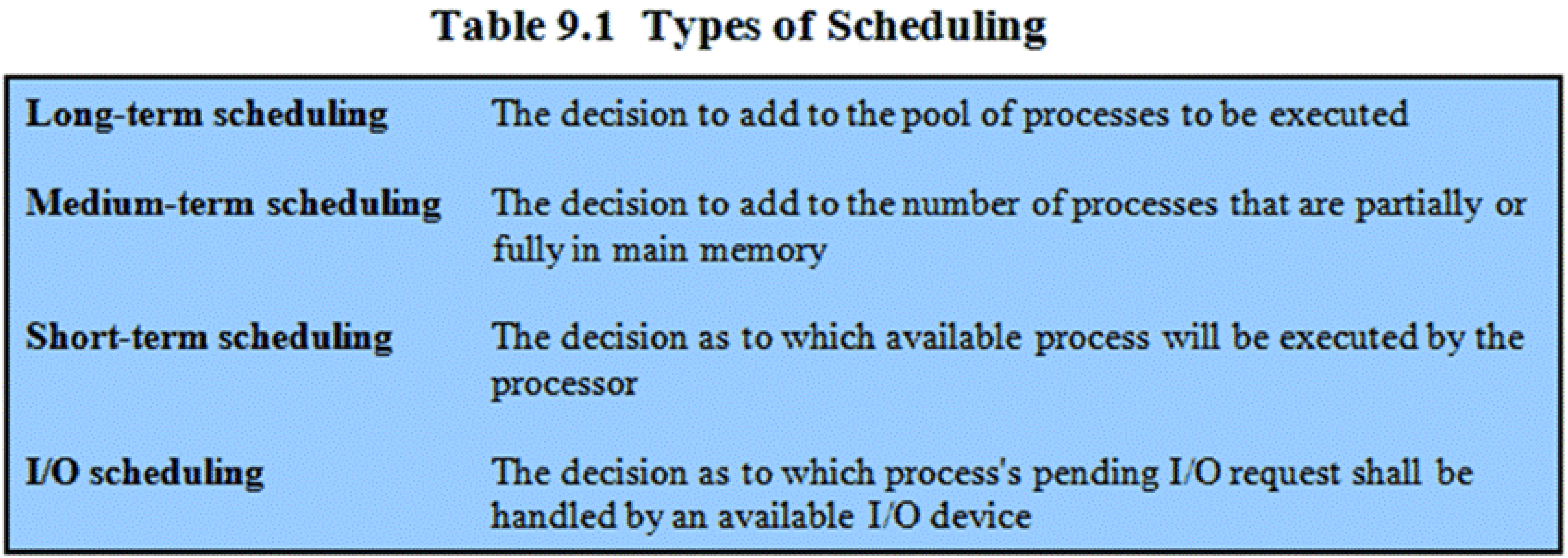
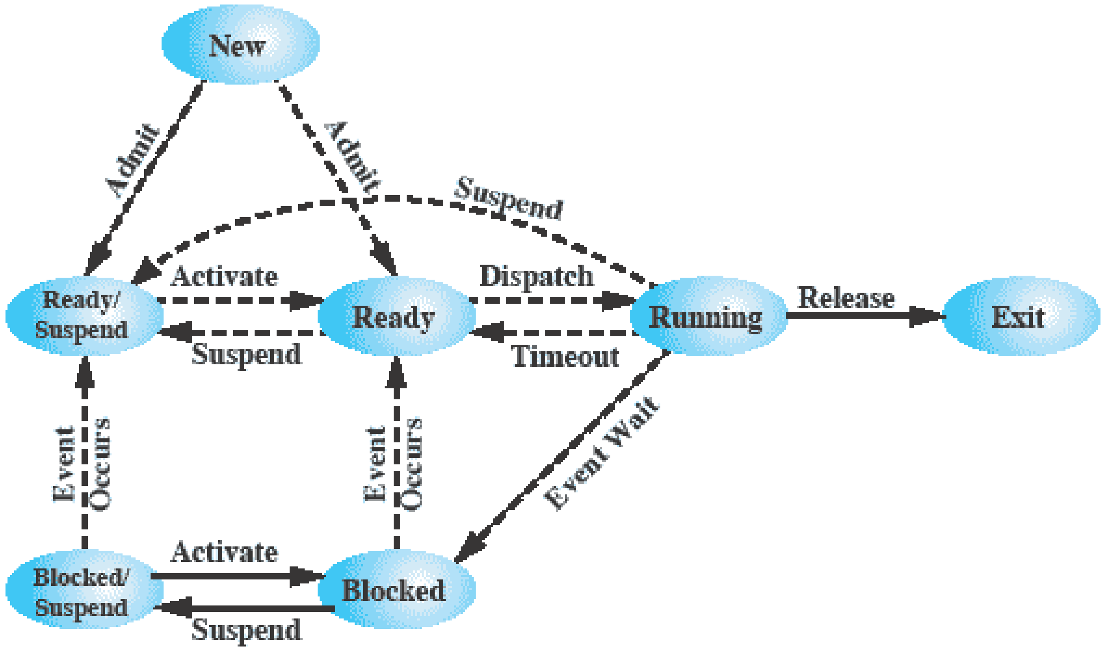


Table 9.1 Types of Scheduling

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

5 State model

Remember this diagram from Chapter 3



(b) With Two Suspend States

Scheduling and Process State Transitions

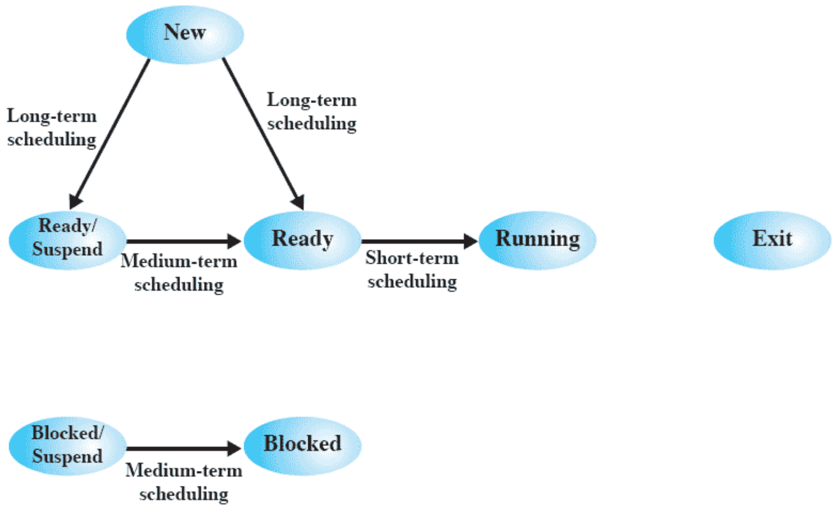
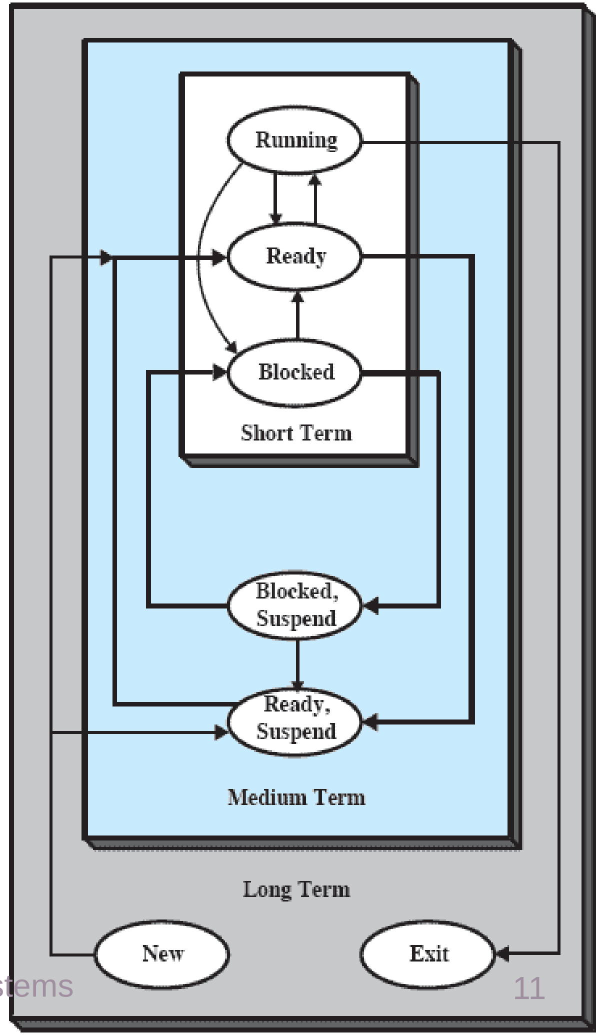


Figure 9.1 **Scheduling and Process State Transitions**

Nesting of Scheduling Functions



Queuing Diagram

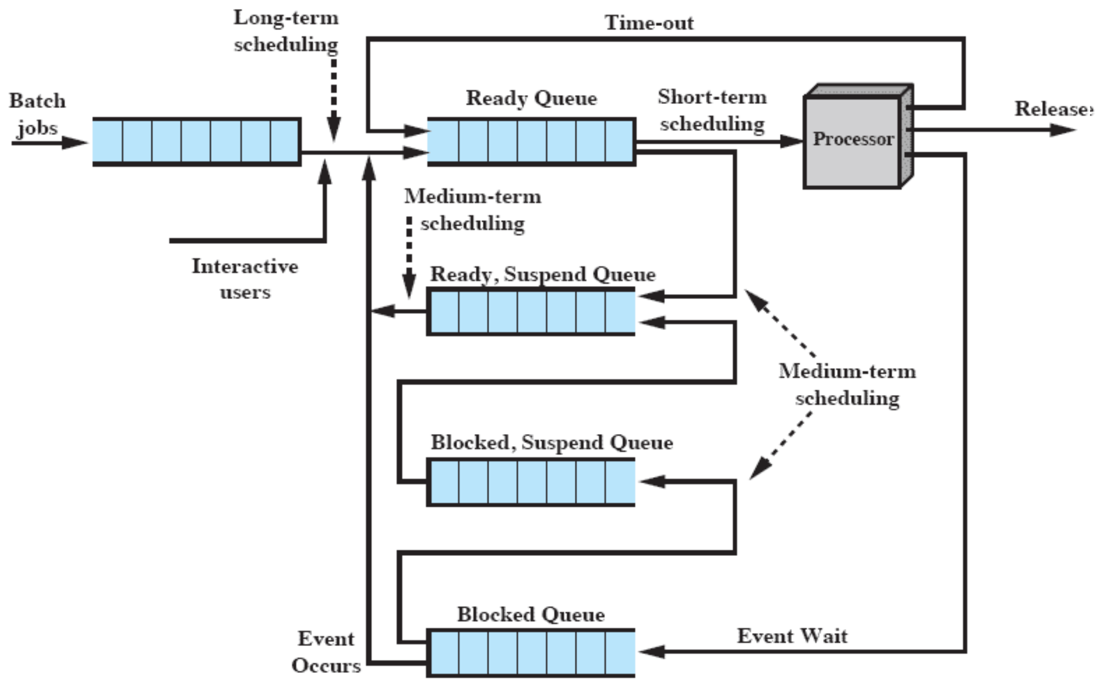
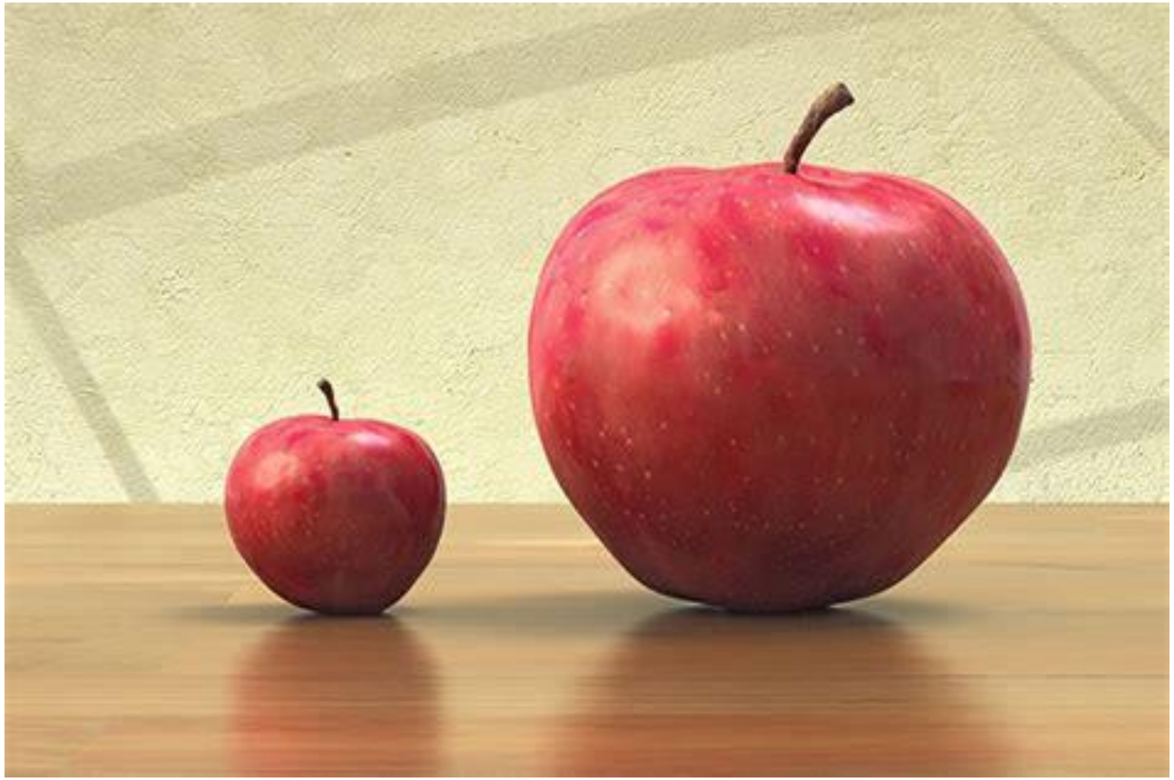


Figure 9.3 Queuing Diagram for Scheduling

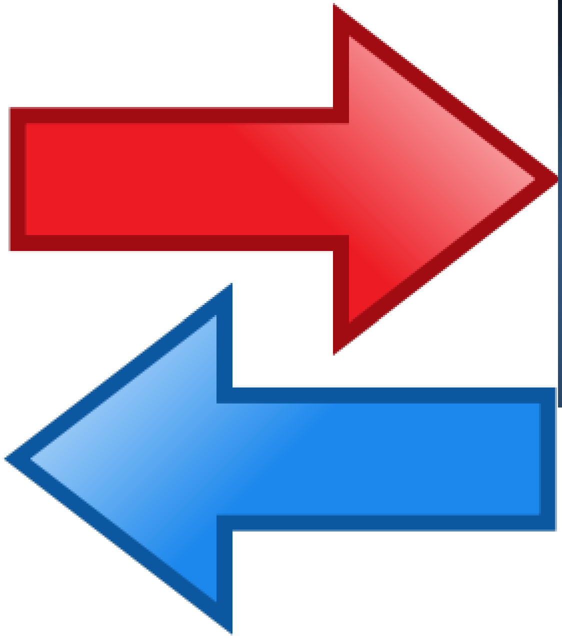
scheduling is a matter of managing queues

Long term scheduling

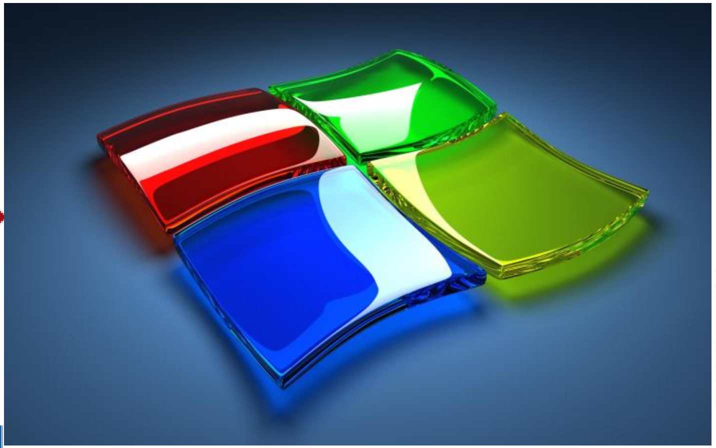
Medium Term Scheduling



Short Term Scheduling



I/O



System Calls (OS)

Signals

Timeout

› FOR SOCIETY

Roadmap

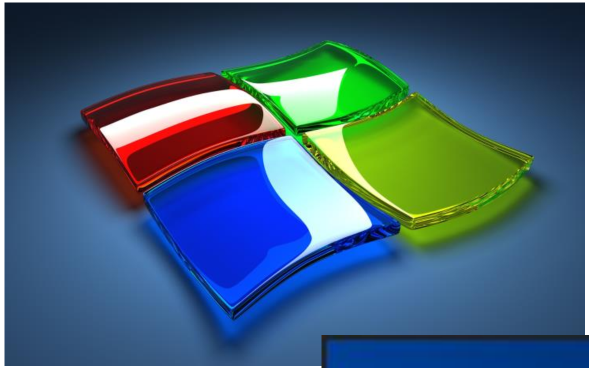
- Types of Processor Scheduling
- **Scheduling Algorithms**
- Traditional UNIX Scheduling

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

- User-oriented



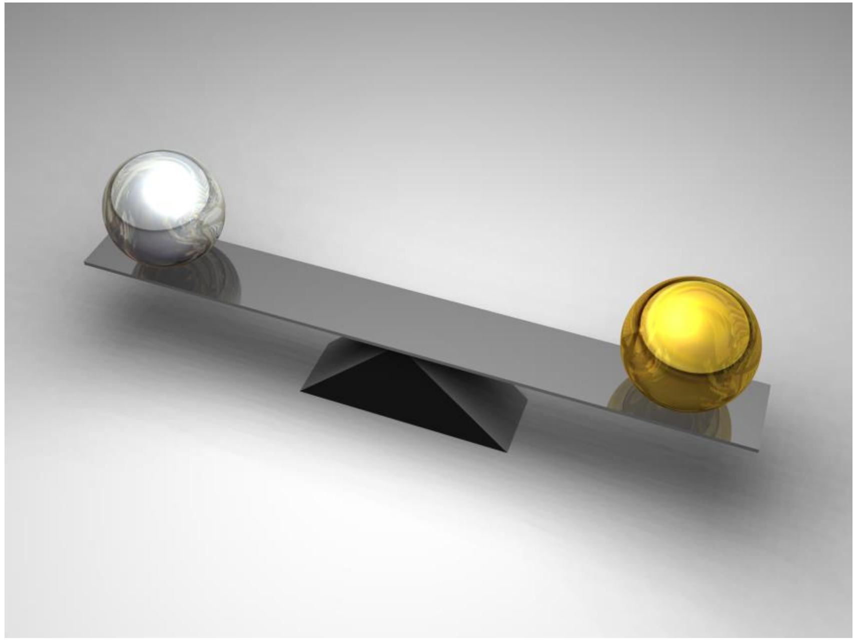
- System-oriented

Short-Term Scheduling Criteria: Performance

We could differentiate between performance related criteria, and those unrelated to performance

- Performance-related

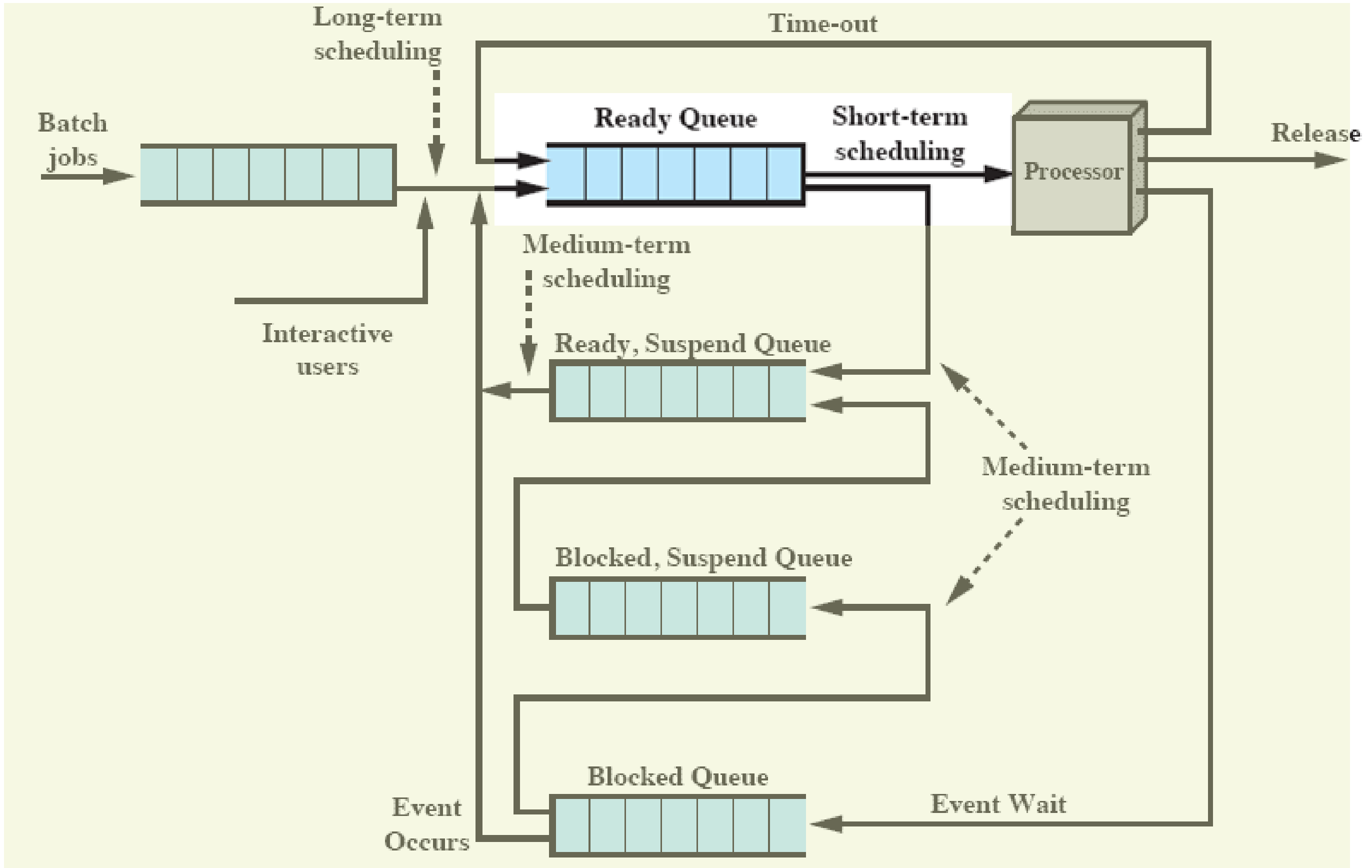
- Non-performance related



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

Queuing Diagram – One ready queue



Priority Queuing

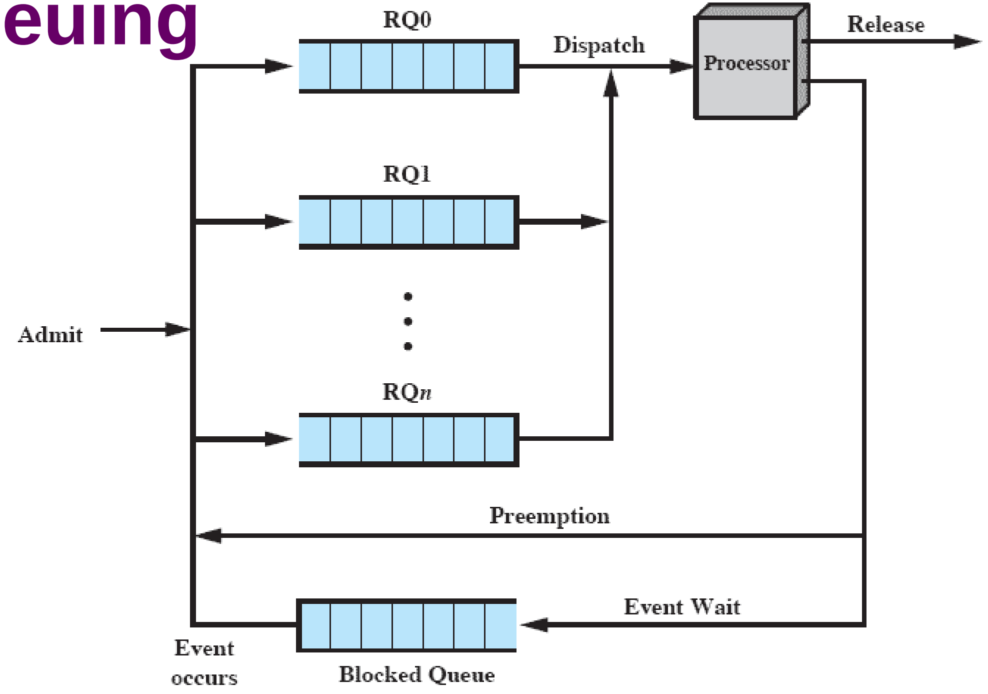


Figure 9.4 Priority Queuing

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

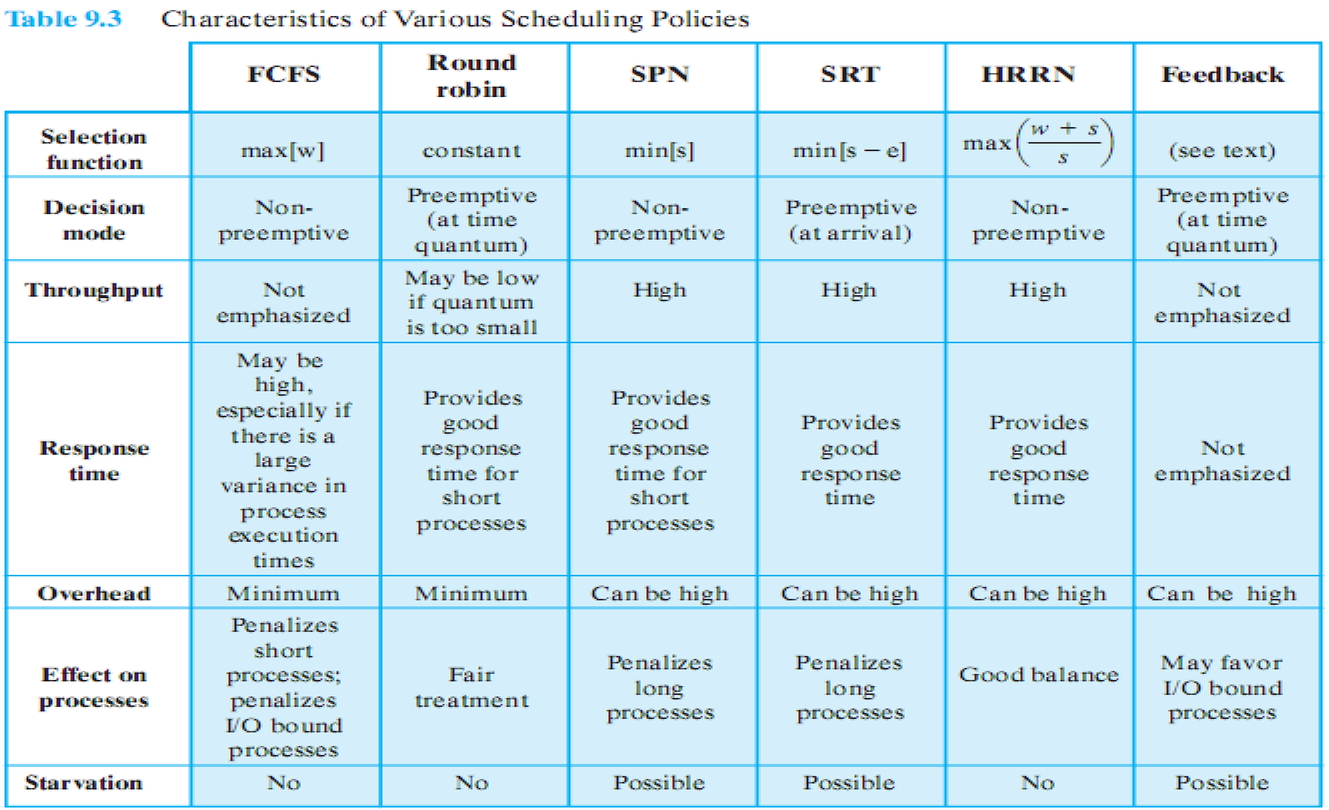


Table 9.3 Characteristics of Various Scheduling Policies

	FCFS	Round robin	SPN	SRT	HRRN	Feedback
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)
Throughput	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

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

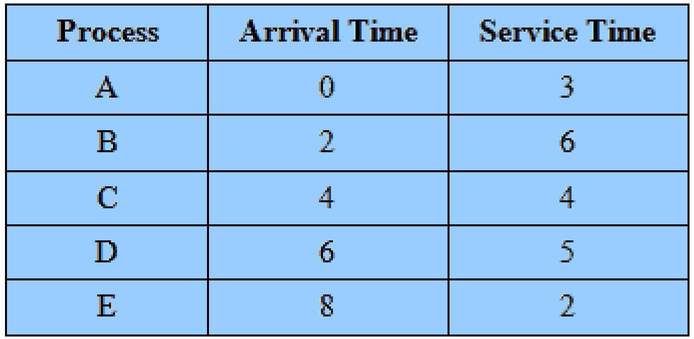
Non-preemptive vs Preemptive

- 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



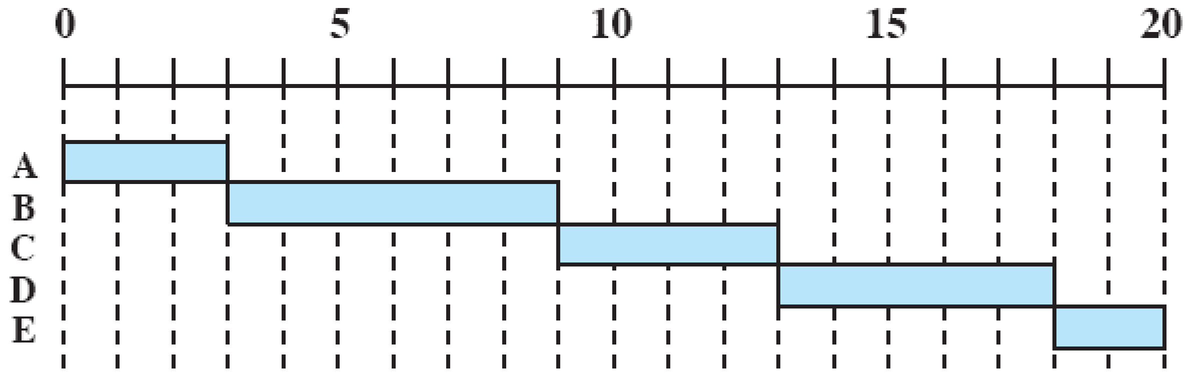
Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Service time represents total execution time

First-Come - First-Served

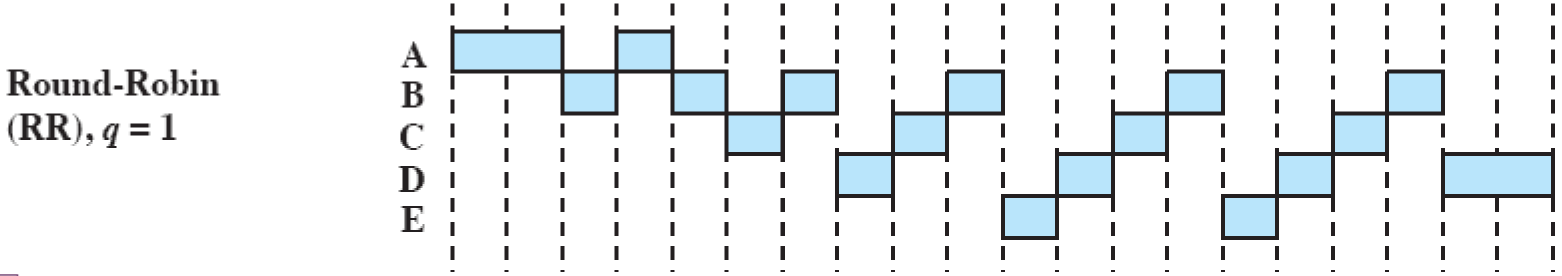
- Each process joins the Ready queue
- When the current process ceases to execute, the longest (oldest) process in the Ready queue is selected

First-Come-First Served (FCFS)

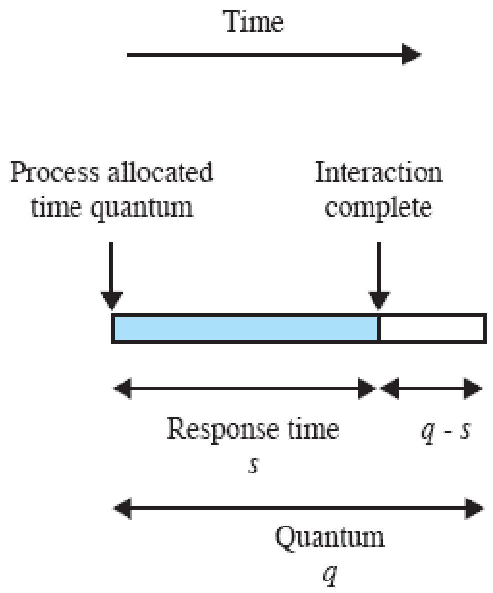


Round Robin

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

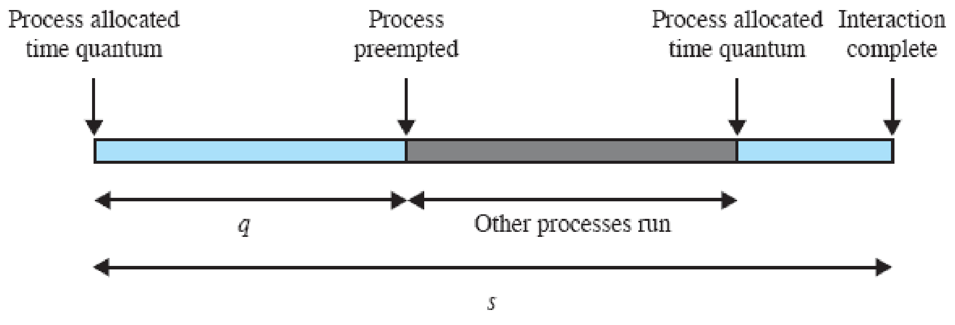


Effect of Size of Preemption Time Quantum



(a) Time quantum greater than typical interaction

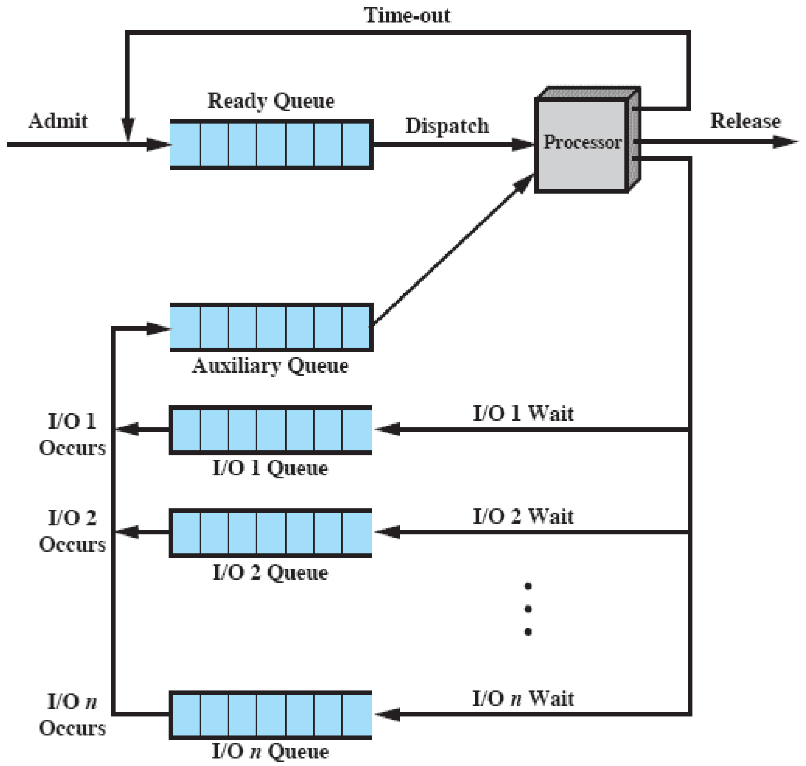
Effect of Size of Preemption Time Quantum


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 Round Robin’



Scheduling overview

- (Non) preemptive
- Priority

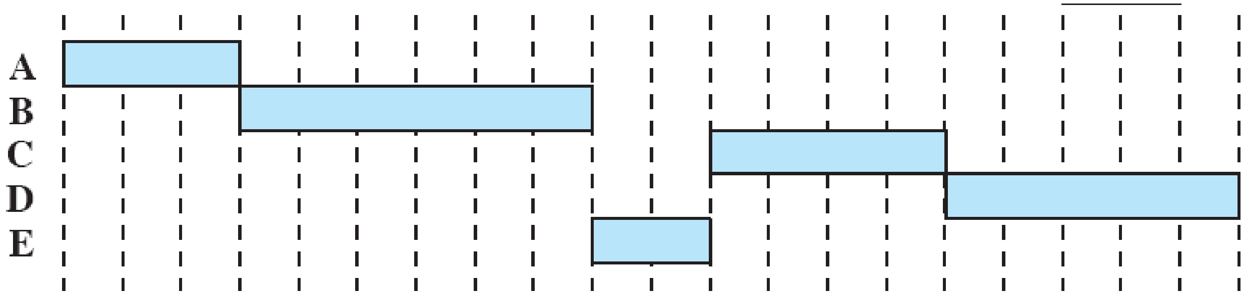
- FCFS
- Round Robin
- Shortest process Next
- Shortest remaining time
- Highest response ratio next
- Feedback Sheduling

Prediction needed

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

Overall performance is significantly improved *throughput* & *waiting time*,

but:

- 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

Calculating Program 'Burst' *based on observation of instances*

- Where:
  - T_i = processor execution time for the *i*th instance of this process
  - S_i = predicted value for the *i*th instance
  - S_1 = predicted value for first instance; not calculated

$$S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i$$

Exponential Averaging

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

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

Trust latest measures more

Exponential Smoothing Coefficients

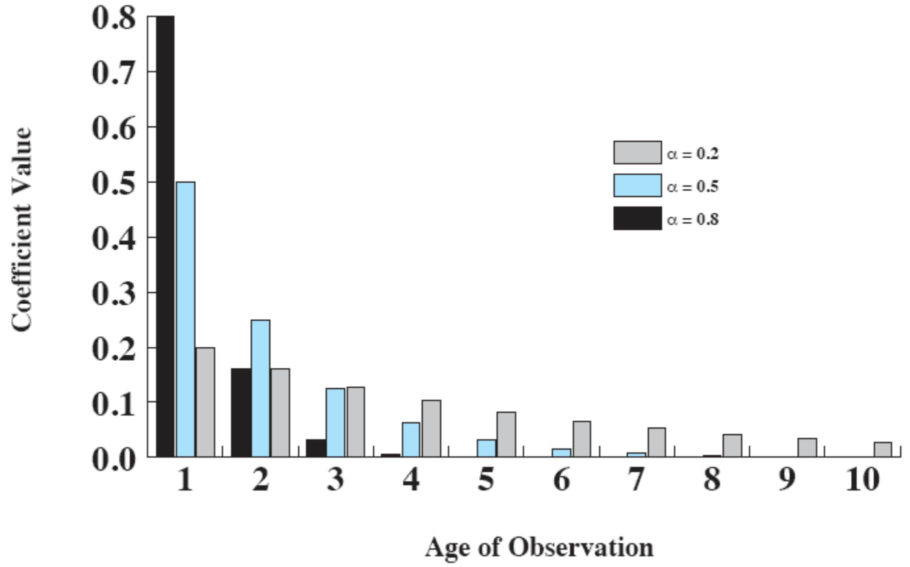
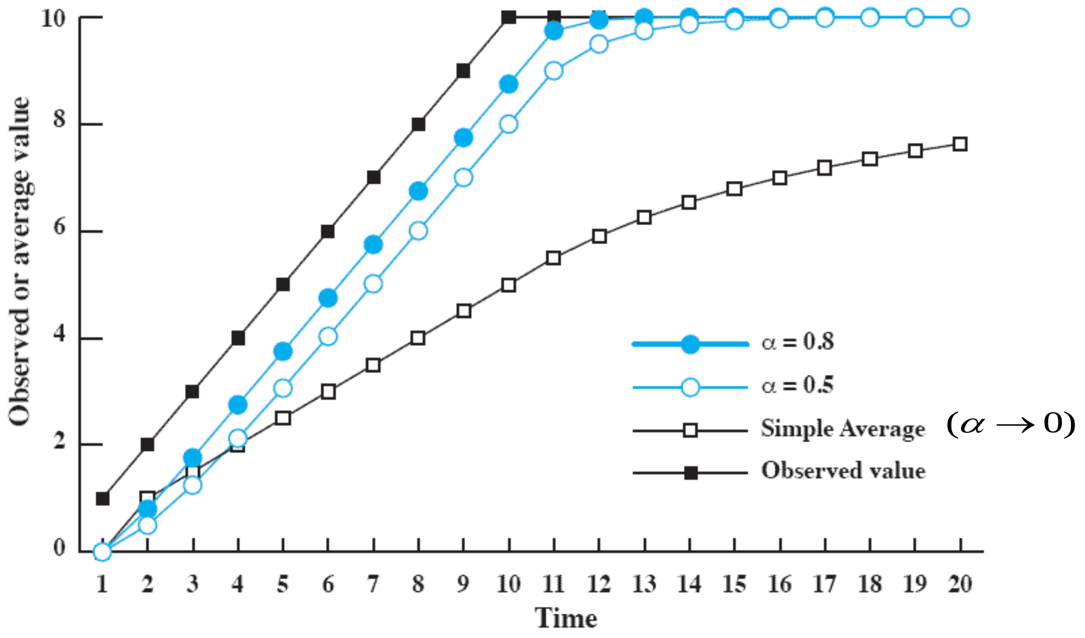


Figure 9.8 Exponential Smoothing Coefficients

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

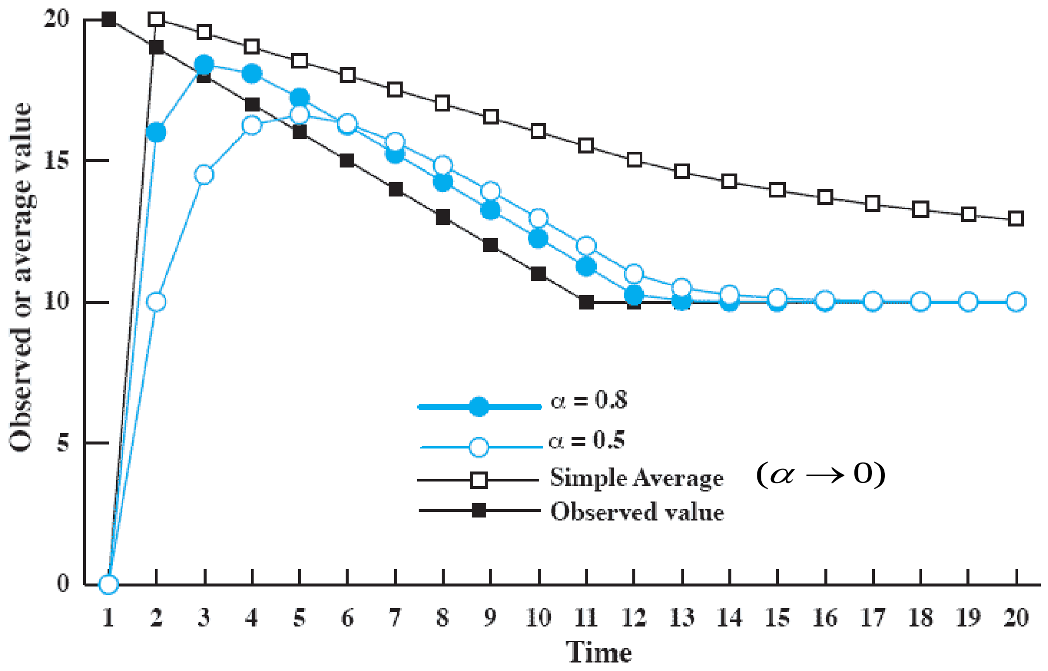
- Advantage low α ?
- Advantage high α ?
- What's $(\alpha \rightarrow 0)$?

Use Of Exponential Averaging



(a) Increasing function

Use Of Exponential Averaging



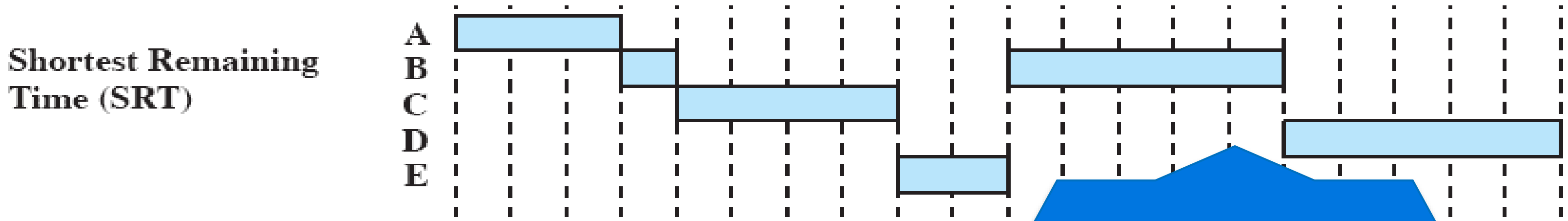
(b) Decreasing function

Exponential Averaging
Conclusions

- exponential averaging tracks changes in process behaviour faster than does simple averaging
- larger value of *α* results in a more rapid reaction to the change in the observed value
- Lower value of *α* better level out peak changes in the observed value

Shortest Remaining Time

- Preemptive version of Shortest Process Next policy
- Must estimate processing time and choose the shortest (+administer *remaining time*)



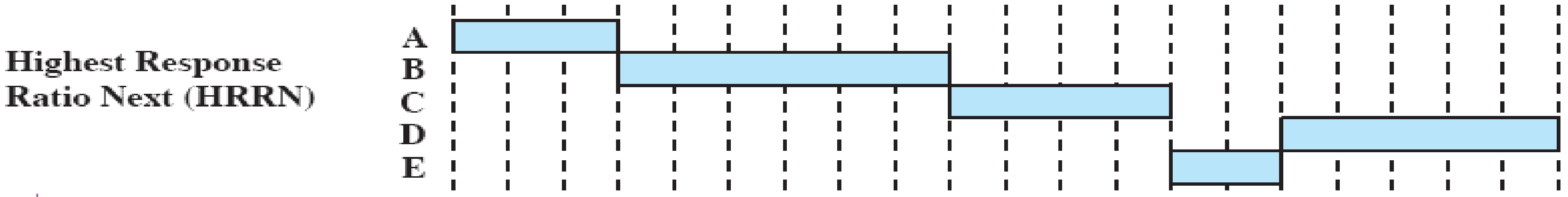
-C Preempts B
-Better than SPN
-Risk of starvation of longer processes

Highest Response Ratio Next

- Problem in SPN: can result in starvation
- Choose next process with the largest ratio

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

Ratio $$R = \frac{w+s}{s}$$



Scheduling overview

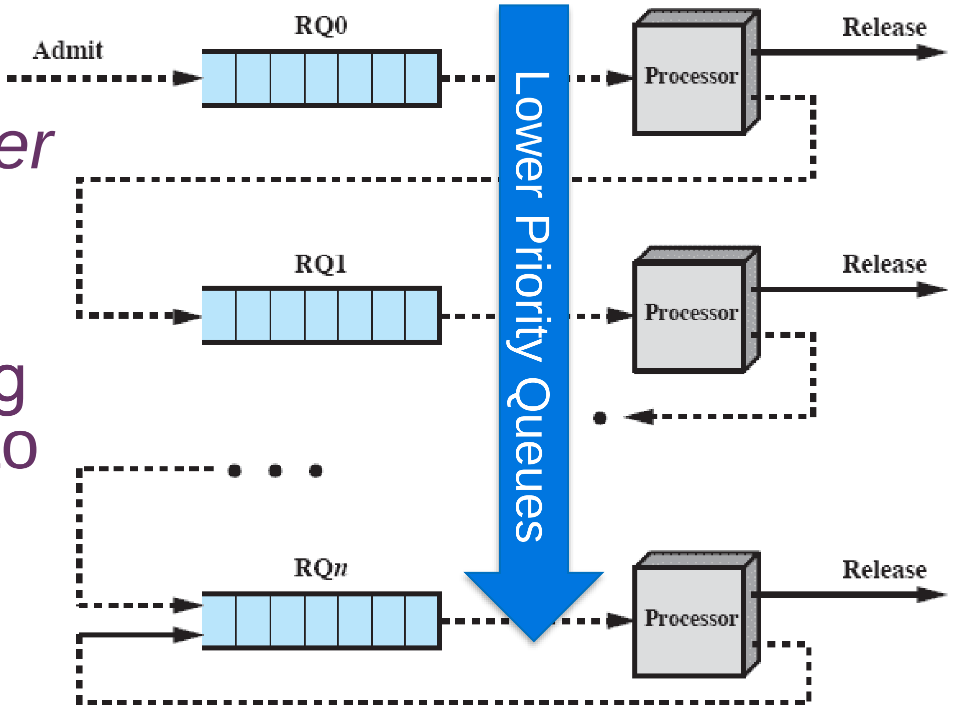
- (Non) preemptive
- Priority

- FCFS
- Round Robin
- Shortest process Next
- Shortest remaining time
- Highest response ratio next
- Feedback Sheduling

Prediction needed

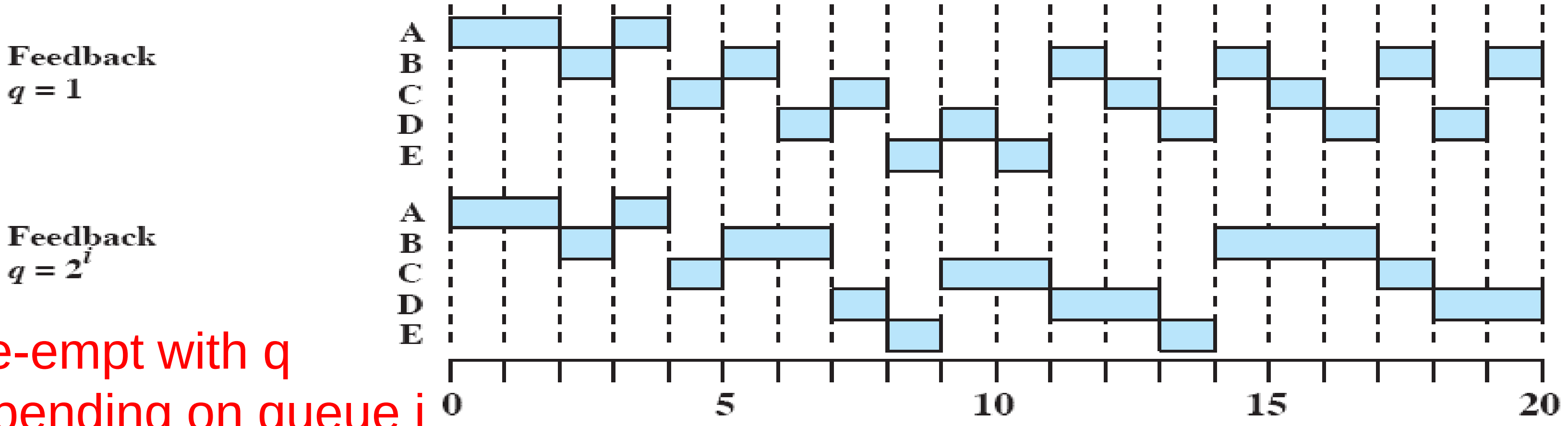
Feedback Scheduling
We do not know length process!

- Penalize jobs that have been running longer (~*prefer shorter jobs*)
- Preemption (time q)
- Don't know remaining time process needs to execute



Feedback Performance

- Variations exist, simple version pre-empts periodically, similar to round robin
  - But can lead to starvation (e.g. lots of small jobs)



Pre-empt with q depending on queue i

› FOR SOCIETY

Performance comparison

Normalized response time

- 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

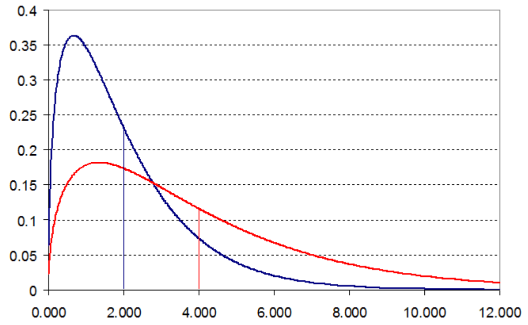
T_s = average service time; average time spent in Running state

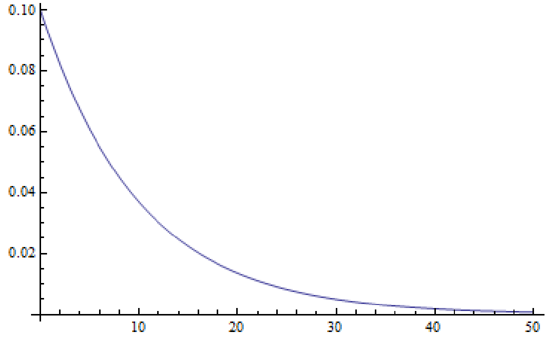
ρ = processor utilization

Experiment

In Practice:

- Arrival distribution:
  - Poisson
- Length per process
  - Exponential

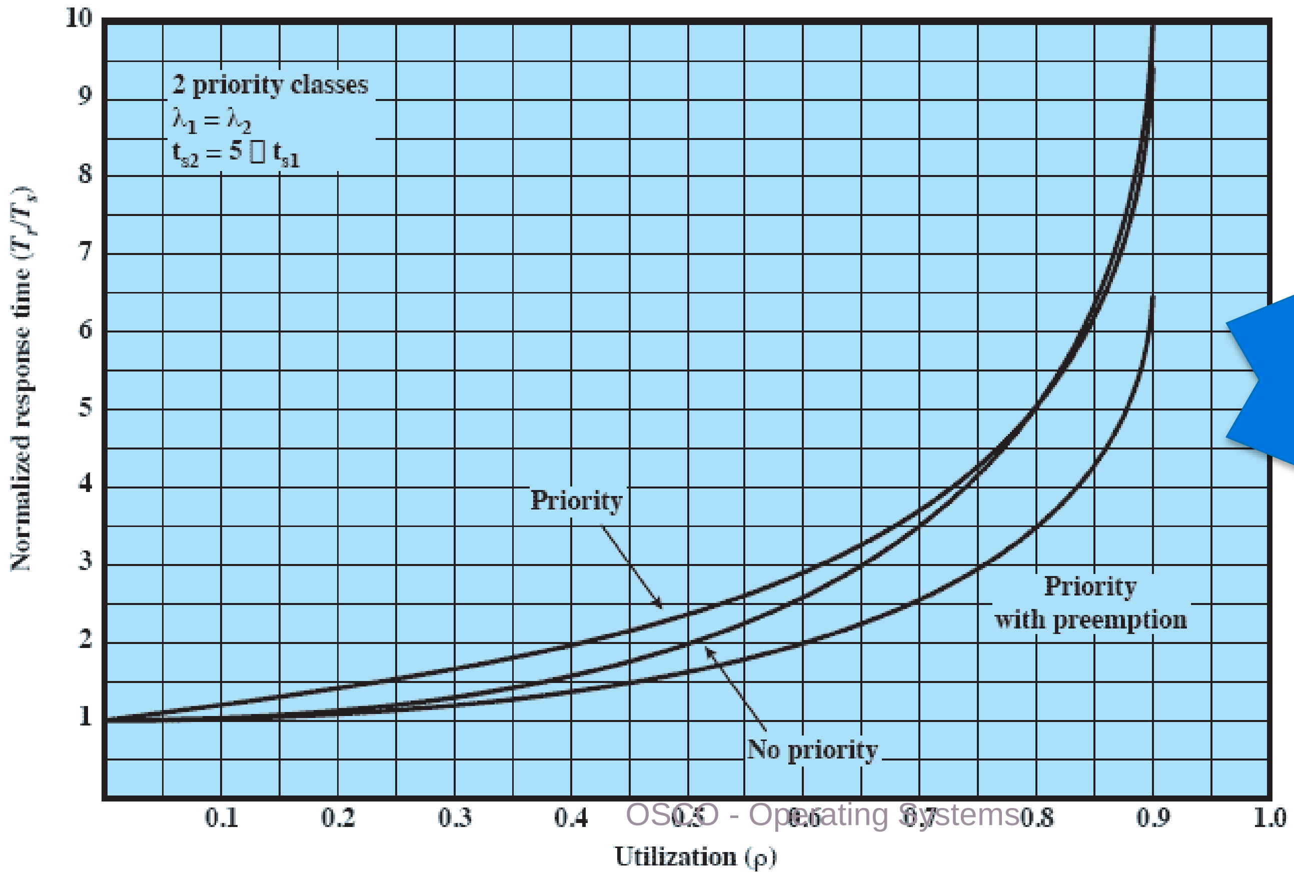




Experiment:

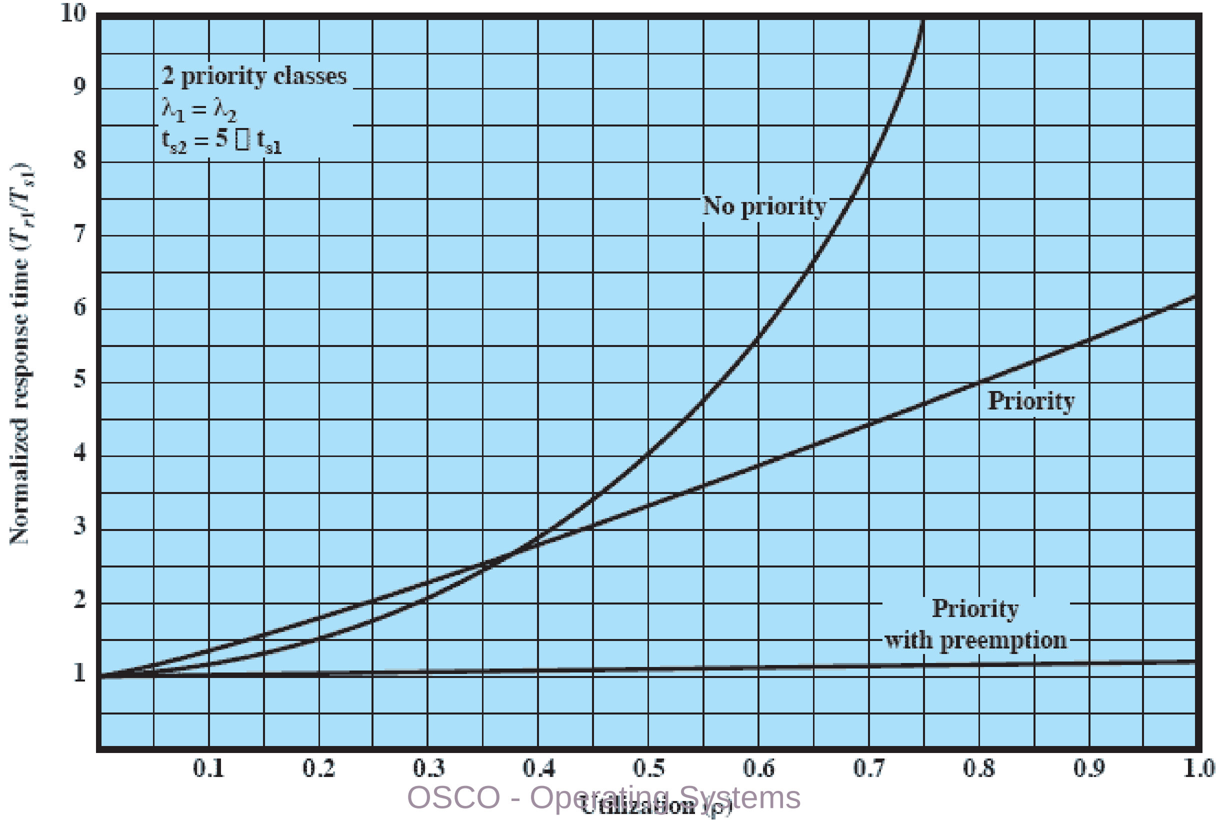
- Priority = Length (2 classes)
- Equal number long and short processes

Overall Normalized Response Time



- -Preference priority for large jobs r>0.8
- -Better with preemption
- -Allmost insignificant

Normalized Response Time for (HP) Shorter Processes



Normalized Response Time for (LP) Longer Processes

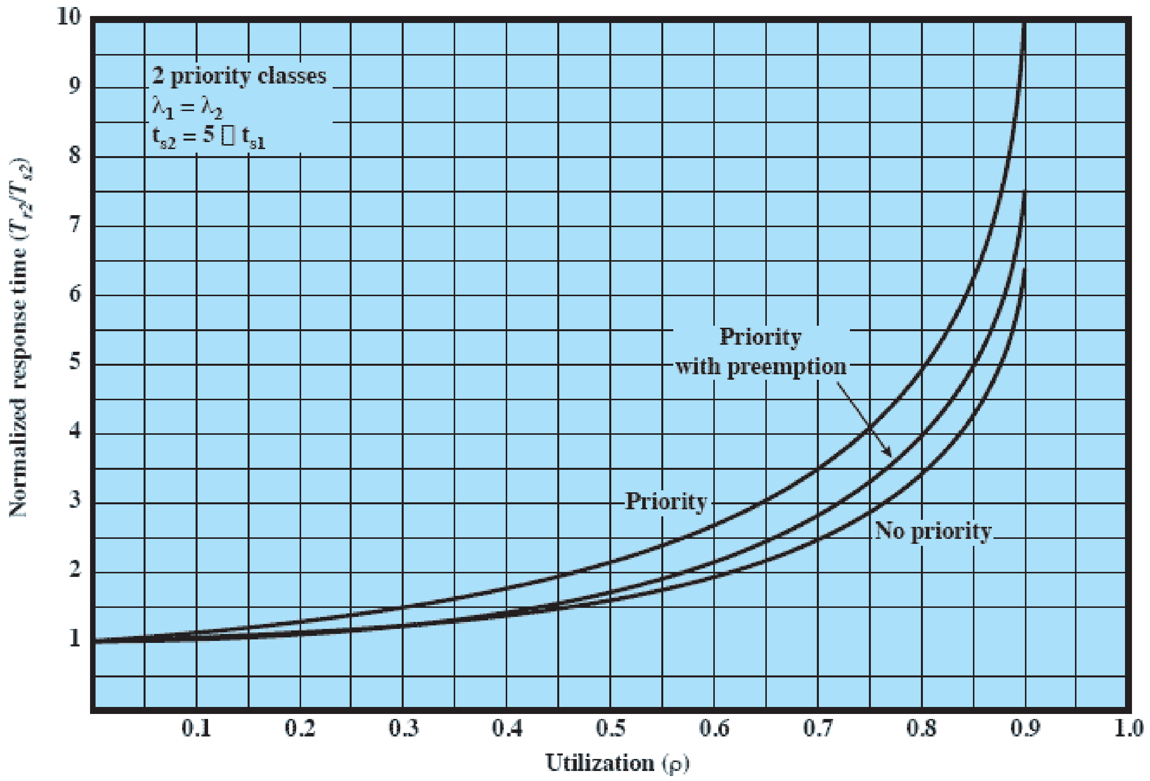


Figure 9.13 Normalized Response Time for Longer Processes

Normalized Turnaround Time and Waiting Time

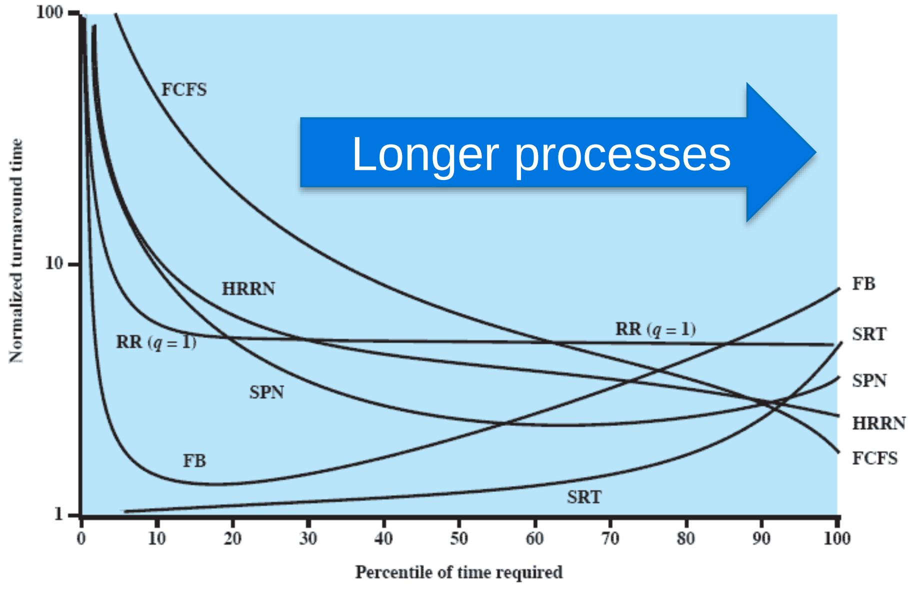


Figure 9.14 Simulation Results for Normalized Turnaround Time

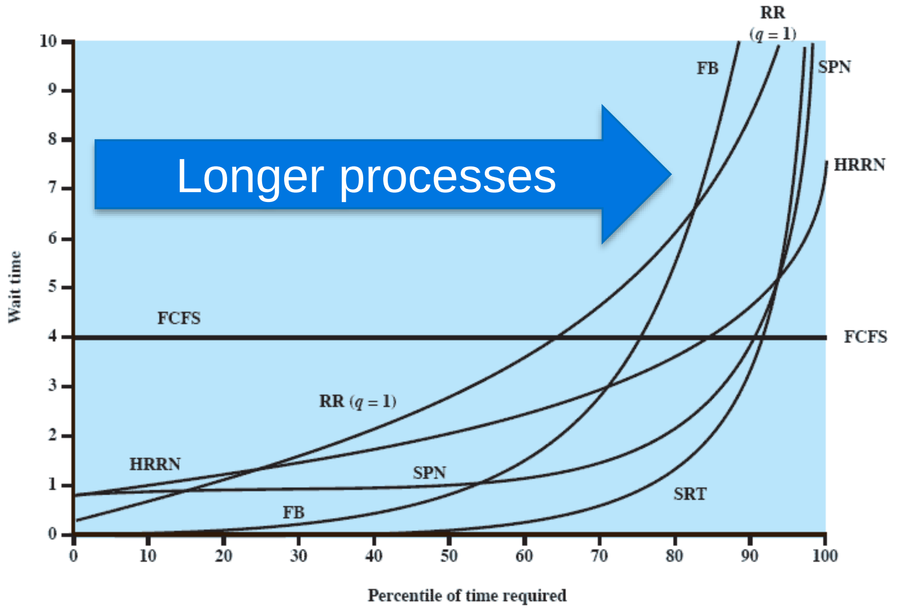


Figure 9.15 Simulation Results for Waiting Time

Mini Assessment

Assignment