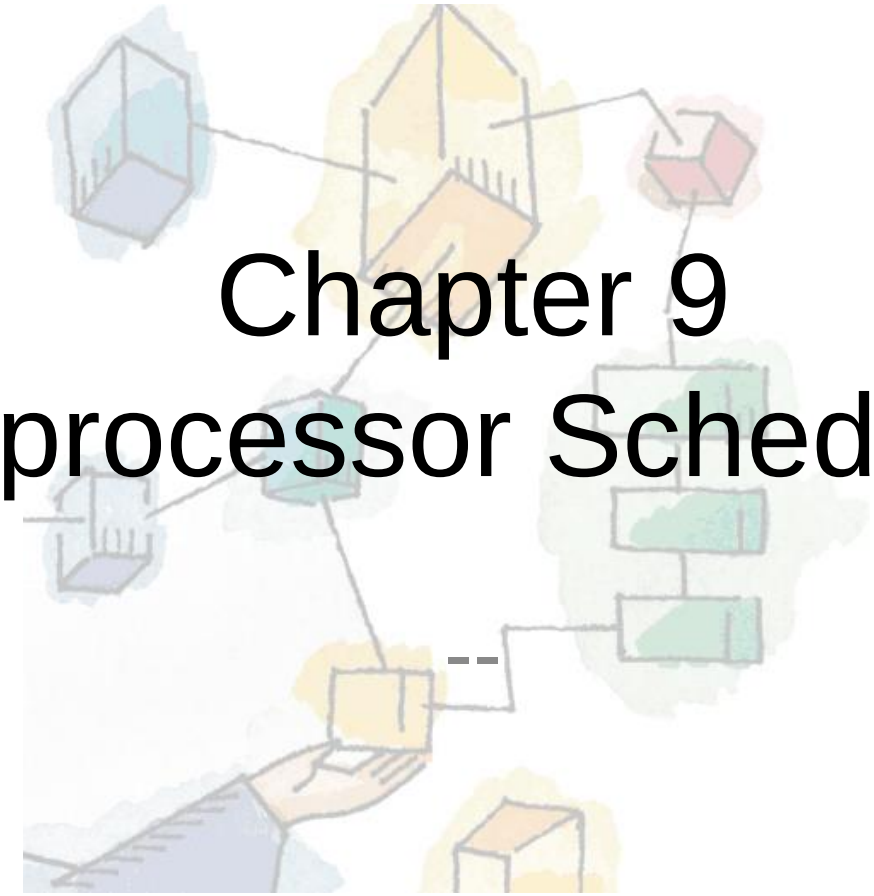
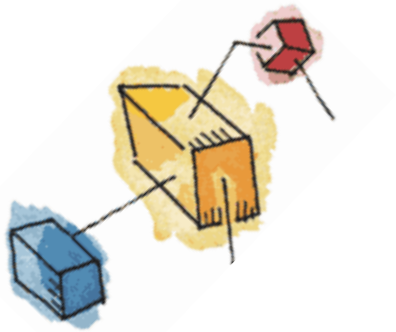


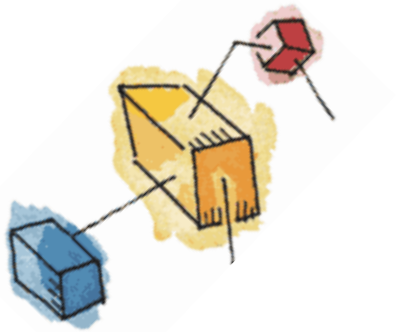
Chapter 9
Uniprocessor Scheduling

--



Roadmap

- Types of Processor Scheduling
- Scheduling Algorithms



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

- 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*
  - *Prioritise* processes when necessary

Types of Scheduling

- **Long-term scheduling** is performed when a *new process* is created.
  - This is a decision whether to add a new process to the set of processes that are currently active.

- **Medium-term scheduling** is a part of the *swapping* function.
  - This is a decision whether to add a process to those that are at least partially in main memory and therefore available for execution.

- **Short-term scheduling** is the actual decision of which ready process to *execute next*.

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

Process State Transition Diagram Concepts

- **Ready:** The process is in *main memory* and *available* for execution.
- **Blocked:** The process is in *main memory* and *awaiting* an event.
- **Blocked/Suspend:** The process is in *secondary memory* and *awaiting* an event.
- **Ready/Suspend:** The process is in *secondary memory* but is *available* for execution as soon as it is loaded into main memory.

Scheduling and Process State Transitions

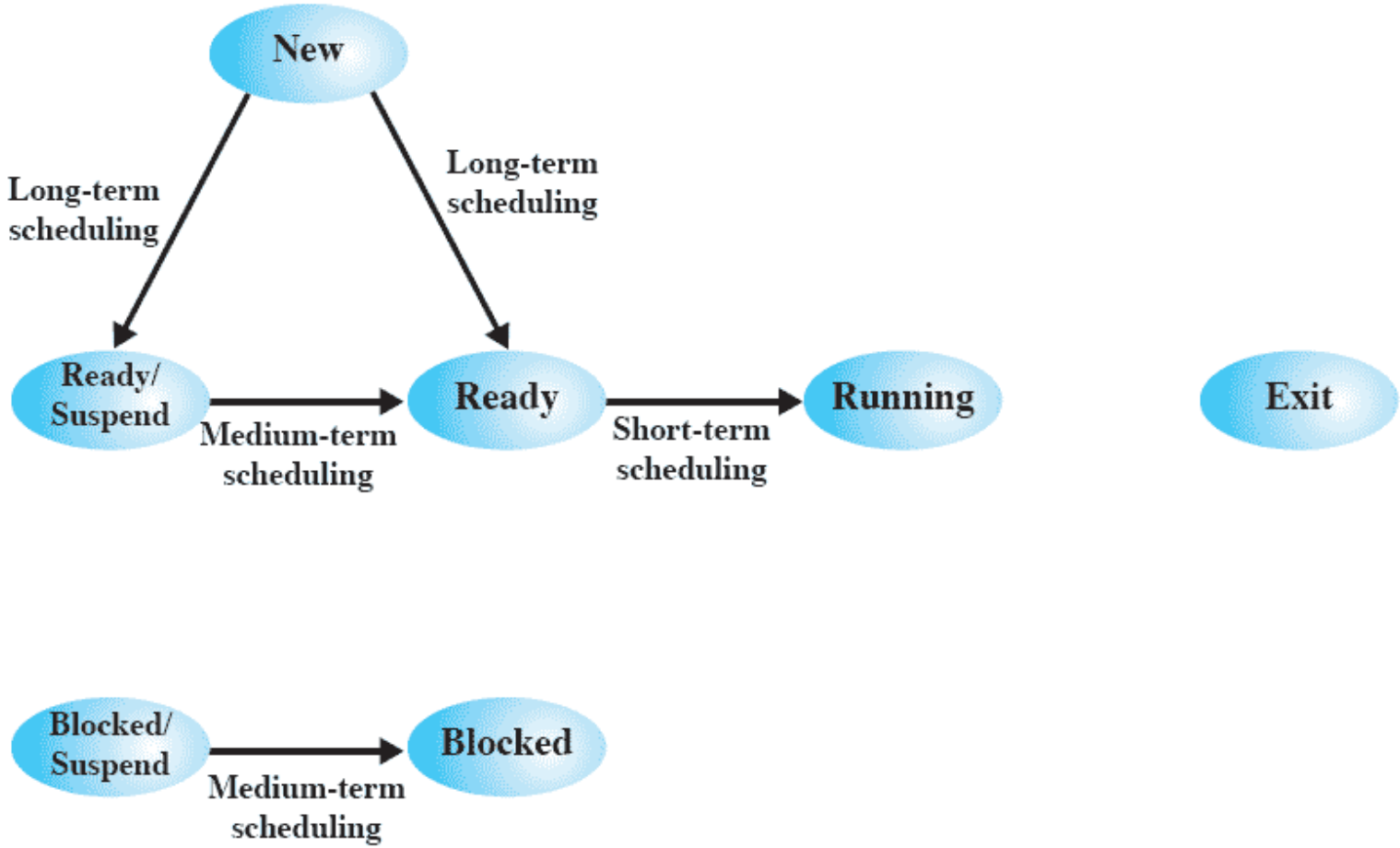


Figure 9.1 Scheduling and Process State Transitions

Queuing Diagram

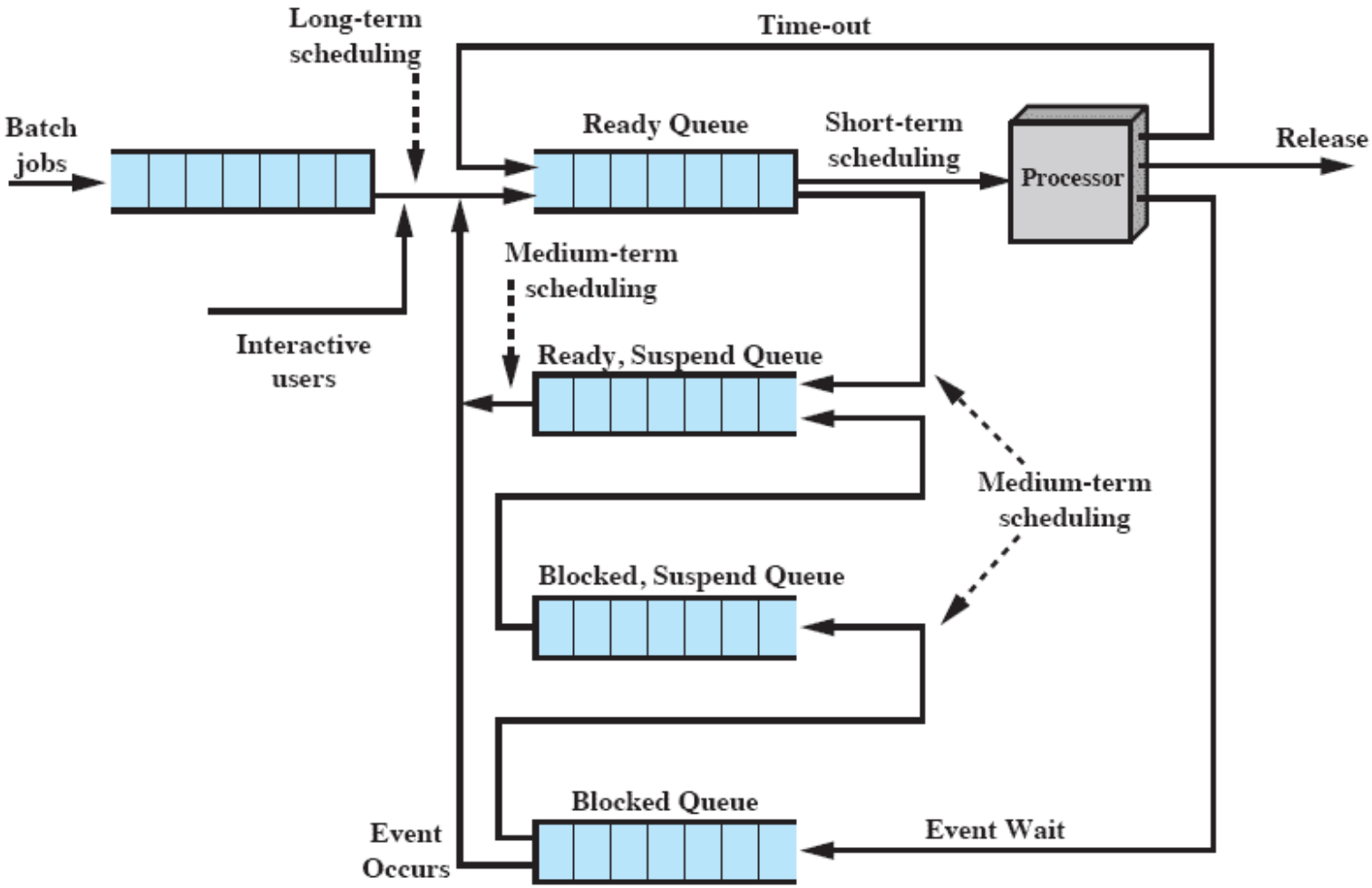
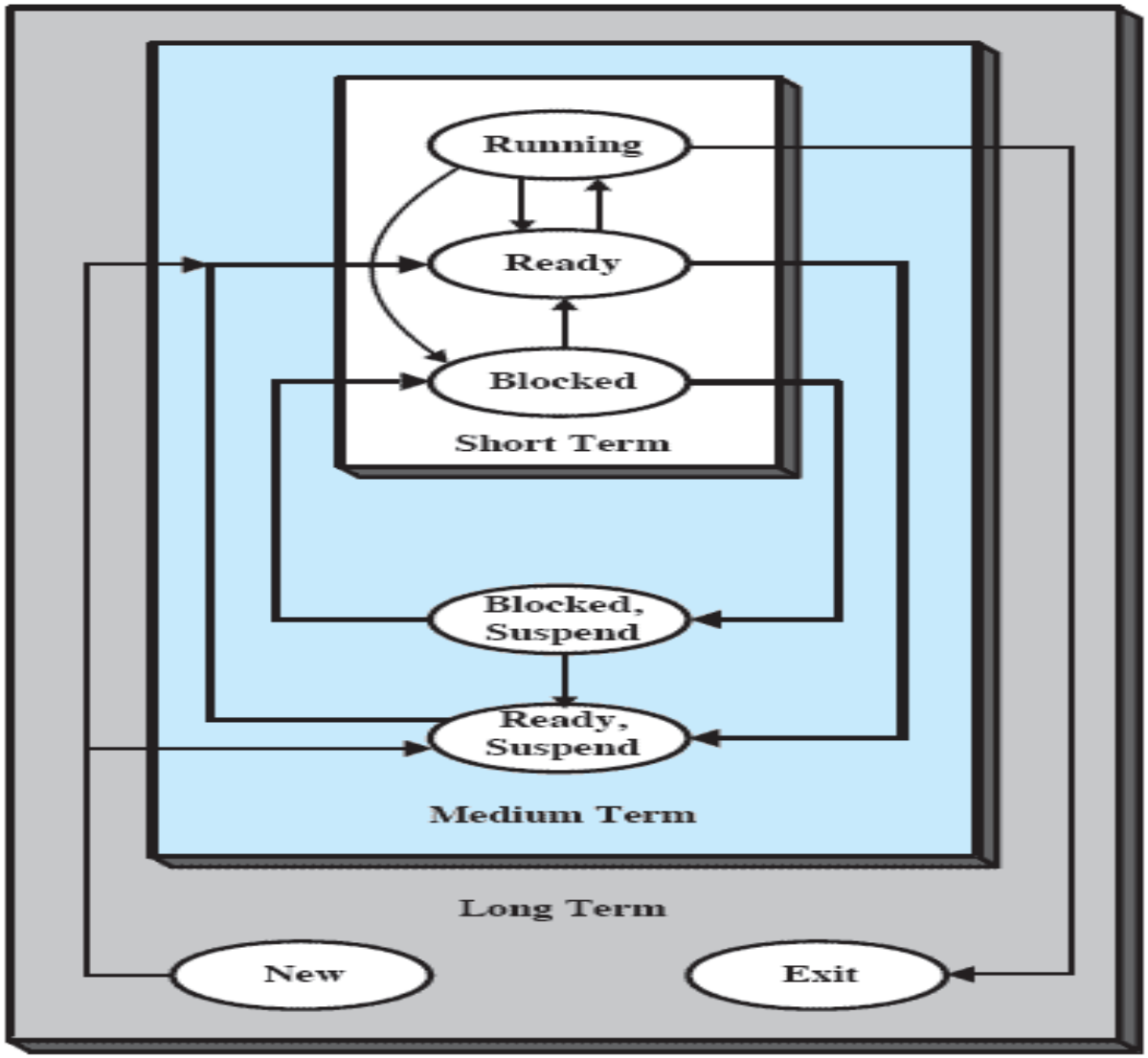


Figure 9.3 Queuing Diagram for Scheduling

Nesting of Scheduling Functions



Long-Term Scheduling

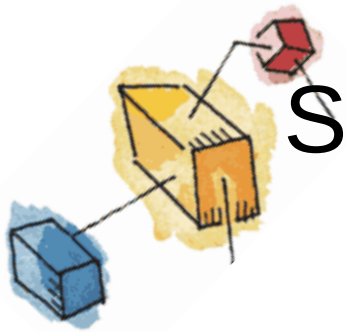
- Determines which programs are *admitted to the system* for processing
  - May be *first-come-first-served*
  - Or according to criteria such as *priority*, *I/O requirements* or *expected execution time*
- Controls the *degree of multiprogramming*
  - More processes, _____ (less/more) percentage of 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



Short-Term Scheduling Criteria: User vs System

- User-oriented Criteria
  - Response Time
- System-oriented Criteria
  - Effective and efficient utilization of the processor

Short-Term Scheduling Criteria: Performance vs Non-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.

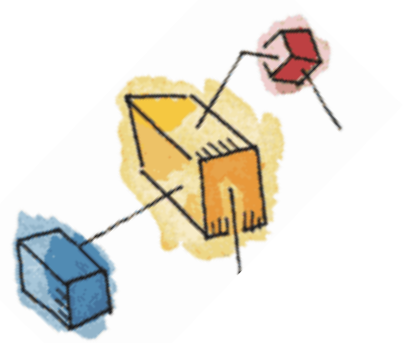
Processor utilization This is the percentage of time that the processor is busy. For an expensive shared system, this is a significant criterion. In single-user systems and in some other systems, such as real-time systems, this criterion is less important than some of the others.

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

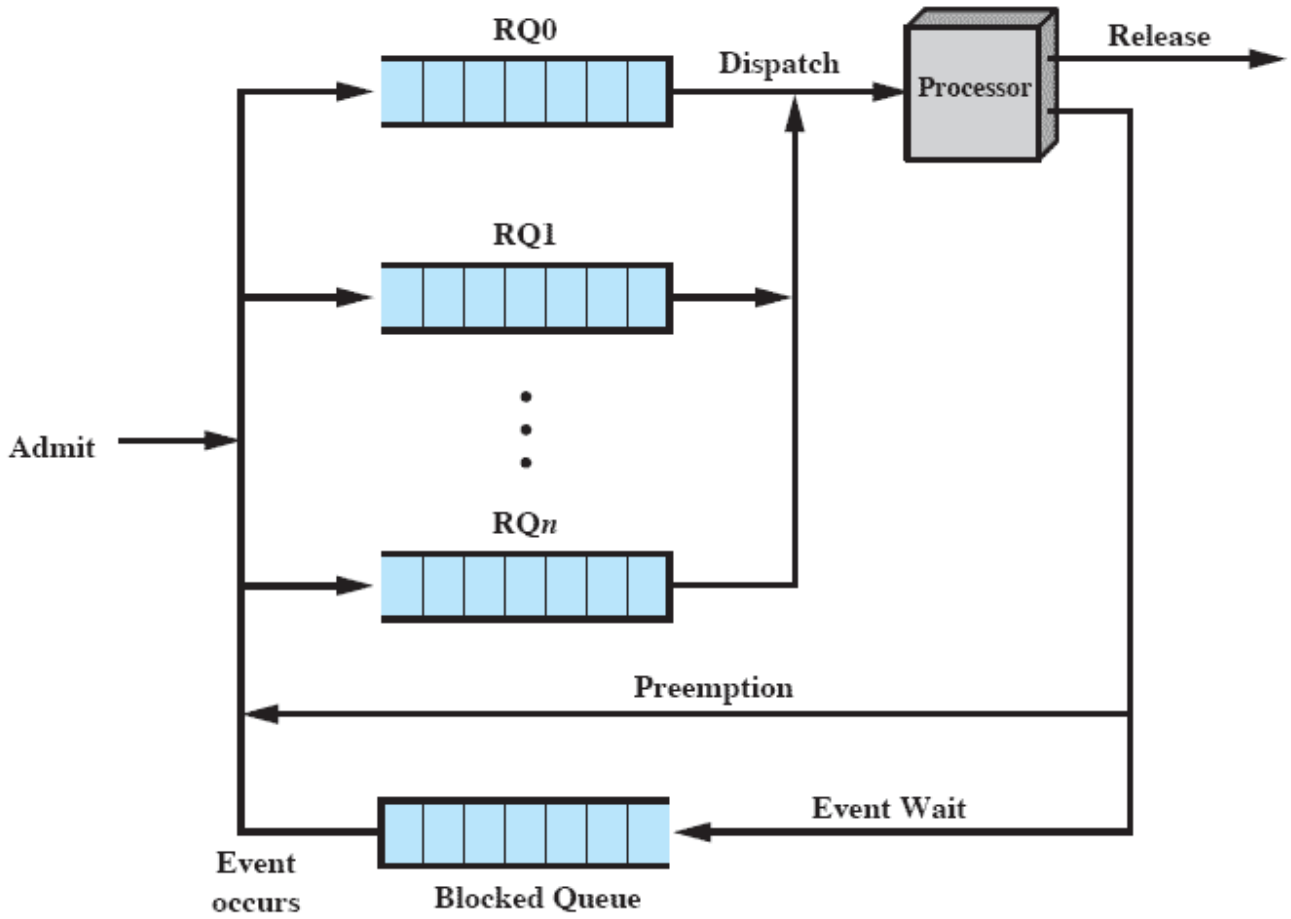


Figure 9.4 Priority Queuing

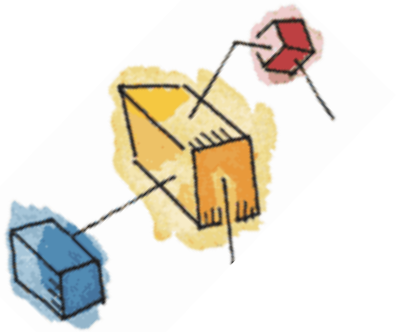
Starvation

- 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

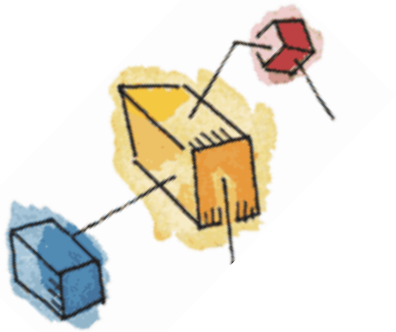
	FCFS	Round robin	SPN	SRT	HRRN	Feedback
Selection function	max[w]	constant	min[s]	min[s − e]	$\max\left(\frac{w+s}{s}\right)$	(see text)
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	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

- Major terms are:
  - ***w*** = time spent in system so far, *waiting*
  - ***e*** = time spent in *execution* so far
  - ***s*** = total service time required by the process
  - **Turn Around Time (TAT)** = Service Time + Waiting Time (Finish Time – Arrival Time)
  - **Normalized TAT =** Tr / Ts (Relative delay experienced by a process)

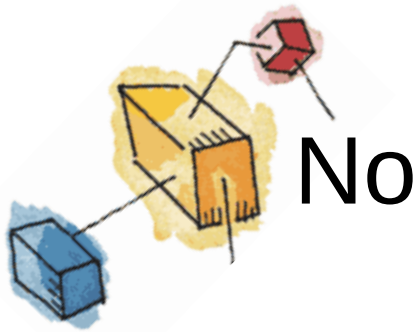


Decision Mode

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

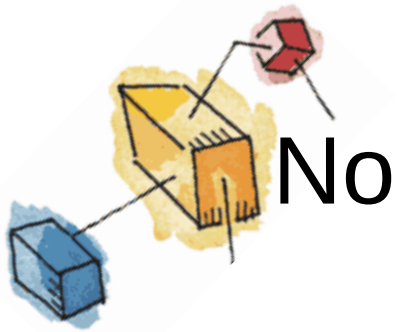
- Two categories:

  – Nonpreemptive
  – Preemptive



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.



Non-preemptive or Preemptive?

- Which decision mode should be preferred?
  - Preemptive
    - Preemptive policy may *increase overhead*
    - But prevents one process from *monopolizing* the processor