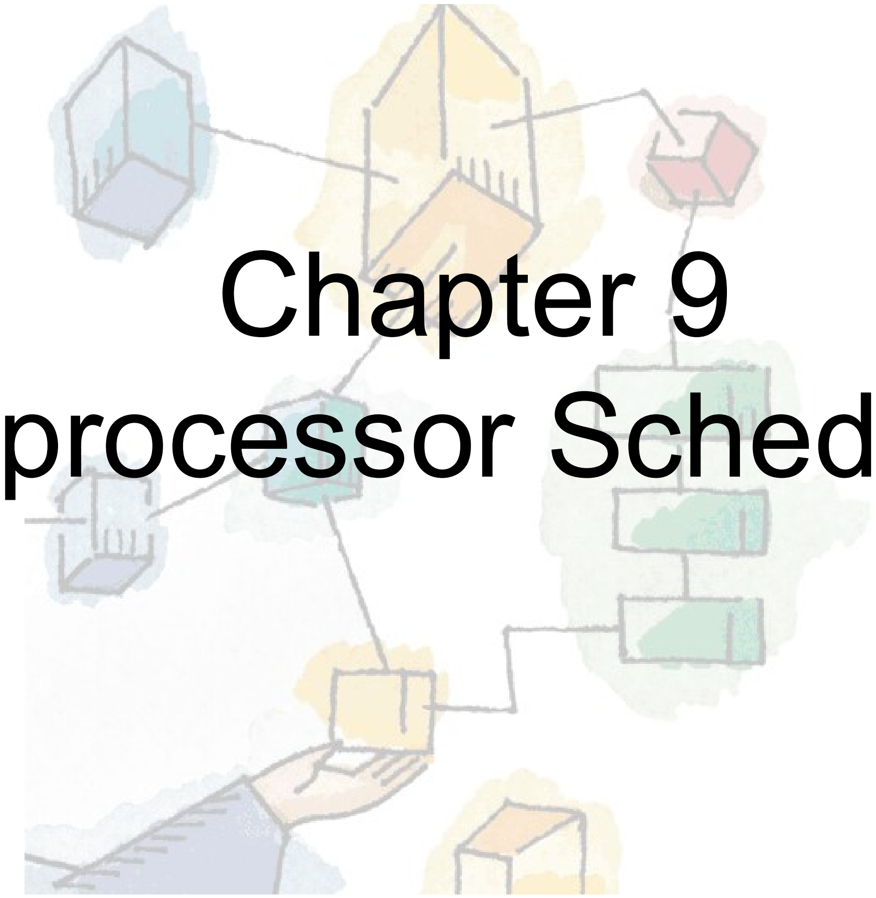
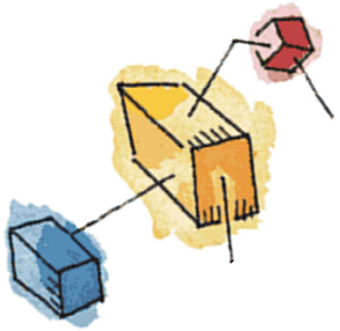
Operating Systems:
Internals and Design Principles
William Stallings



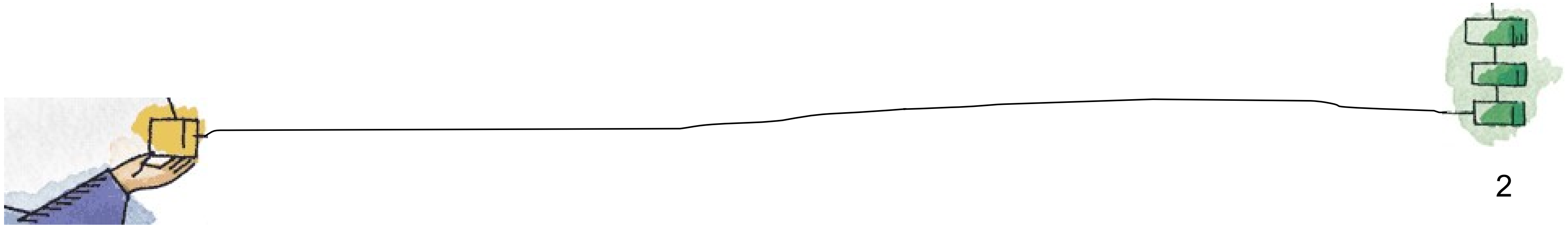
Chapter 9 Uniprocessor Scheduling

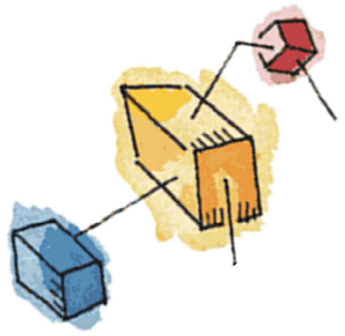


Roadmap

→ Types of Processor Scheduling

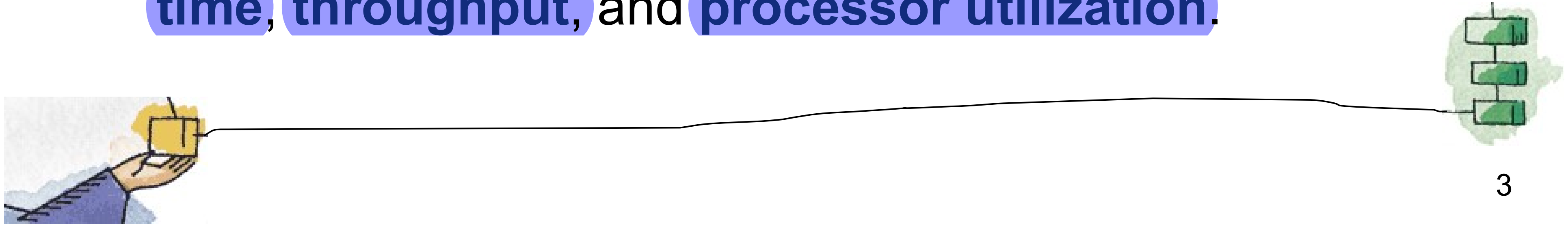
- Scheduling Algorithms
  - FCFS (First-Come-First-Serve)
  - RR (Round-Robin)
  - SPN (Shortest-Process-Next)
  - SRT (Shortest-Remaining-Time)
  - HRRN (Highest-Response-Ratio-Next)
  - FB (Feedback)
- Fair-Share Scheduling

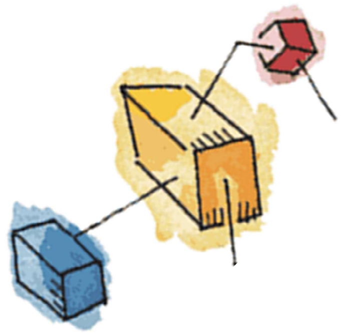




Scheduling

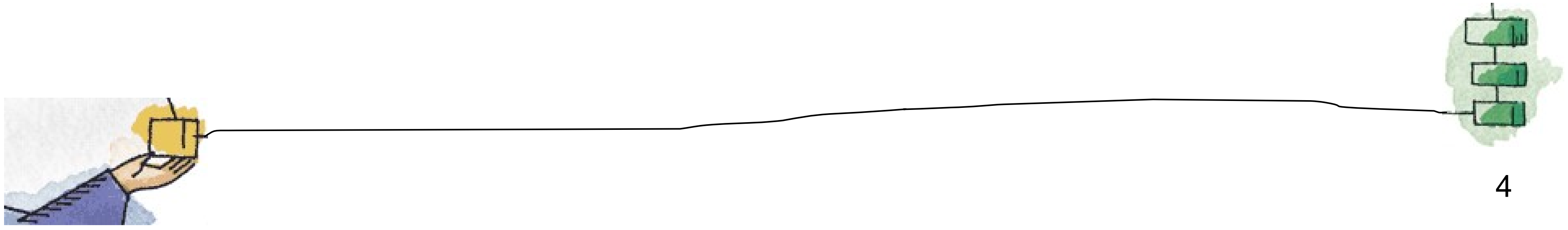
- An OS must allocate resources amongst competing processes.
- The resource provided by a processor is execution time.
- Processor resource is allocated by means of **scheduling** - determines which processes will wait and which will progress.
- The aim of processor scheduling is to assign processes to be executed by the processor in a way that meets system objectives, such as **response time**, **turnaround time**, **throughput**, and **processor utilization**.





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

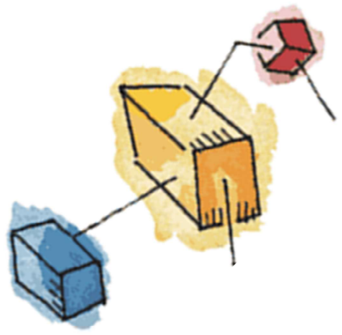
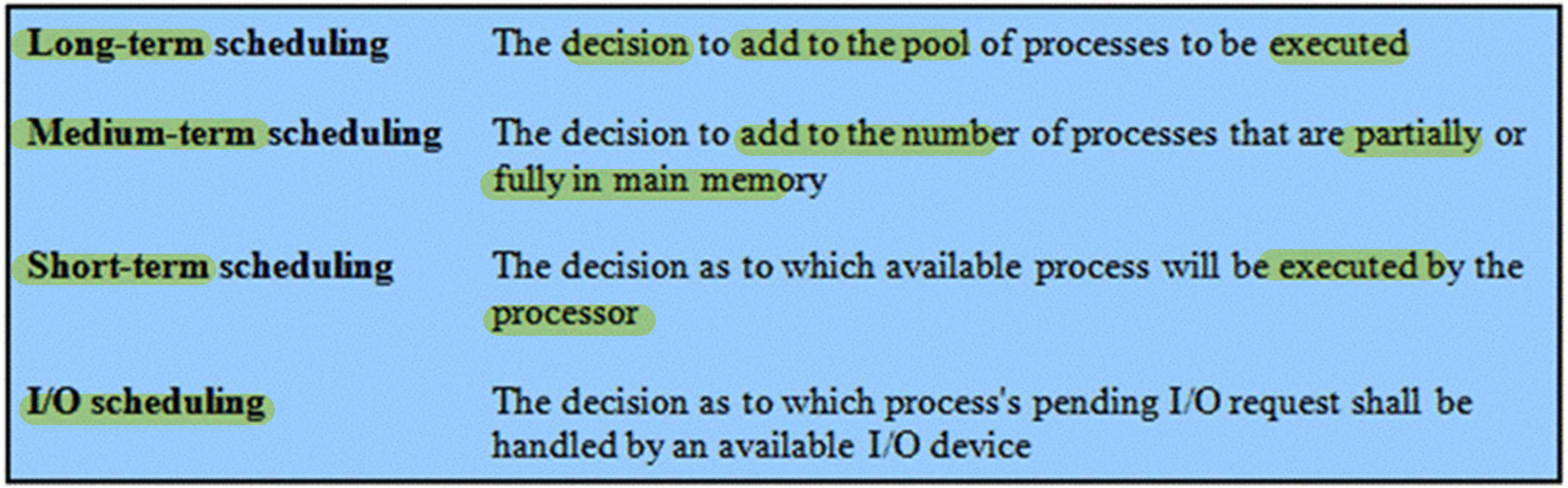
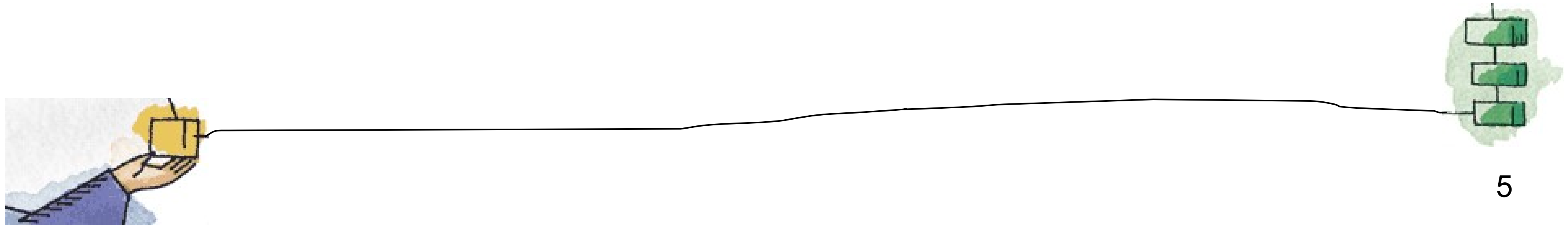


Table 9.1 Types of Scheduling

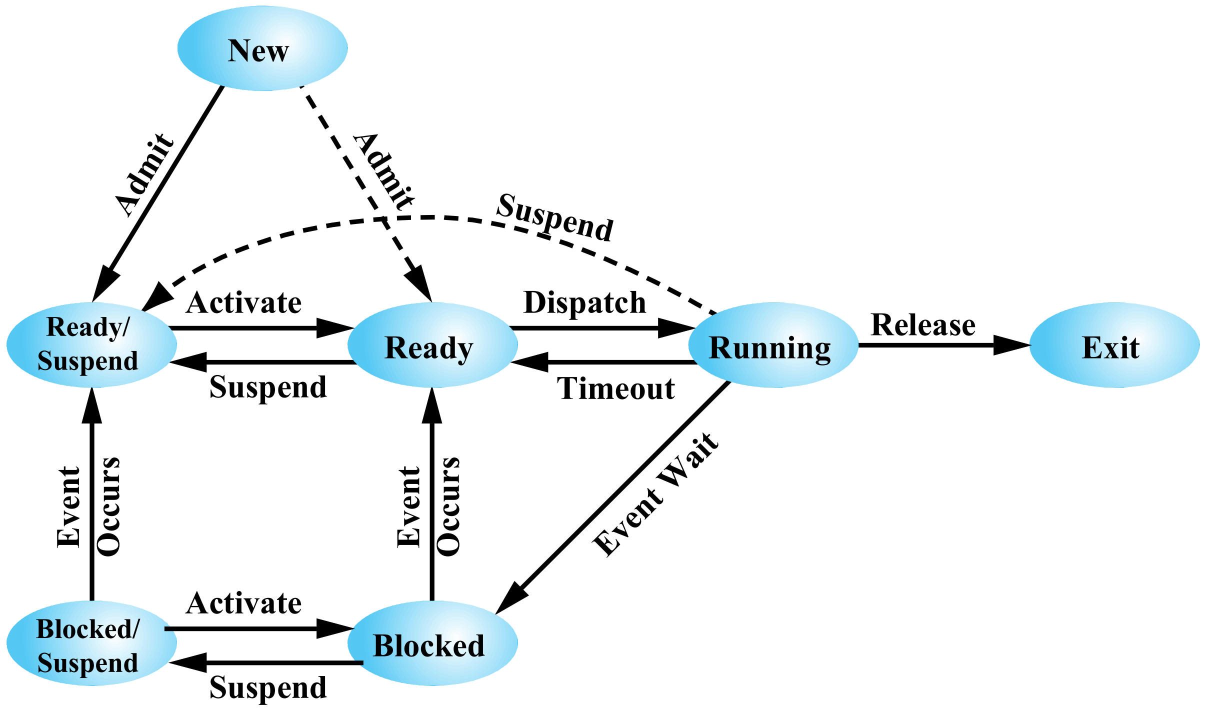


Long-term scheduling	The decision to add to the pool of processes to be executed
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory
Short-term scheduling	The decision as to which available process will be executed by the processor
I/O scheduling	The decision as to which process's pending I/O request shall be handled by an available I/O device



Two Suspend States

- Remember this diagram from Chapter 3?



(b) With Two Suspend States

Figure 3.9 Process State Transition Diagram with Suspend States

Scheduling and Process State Transitions

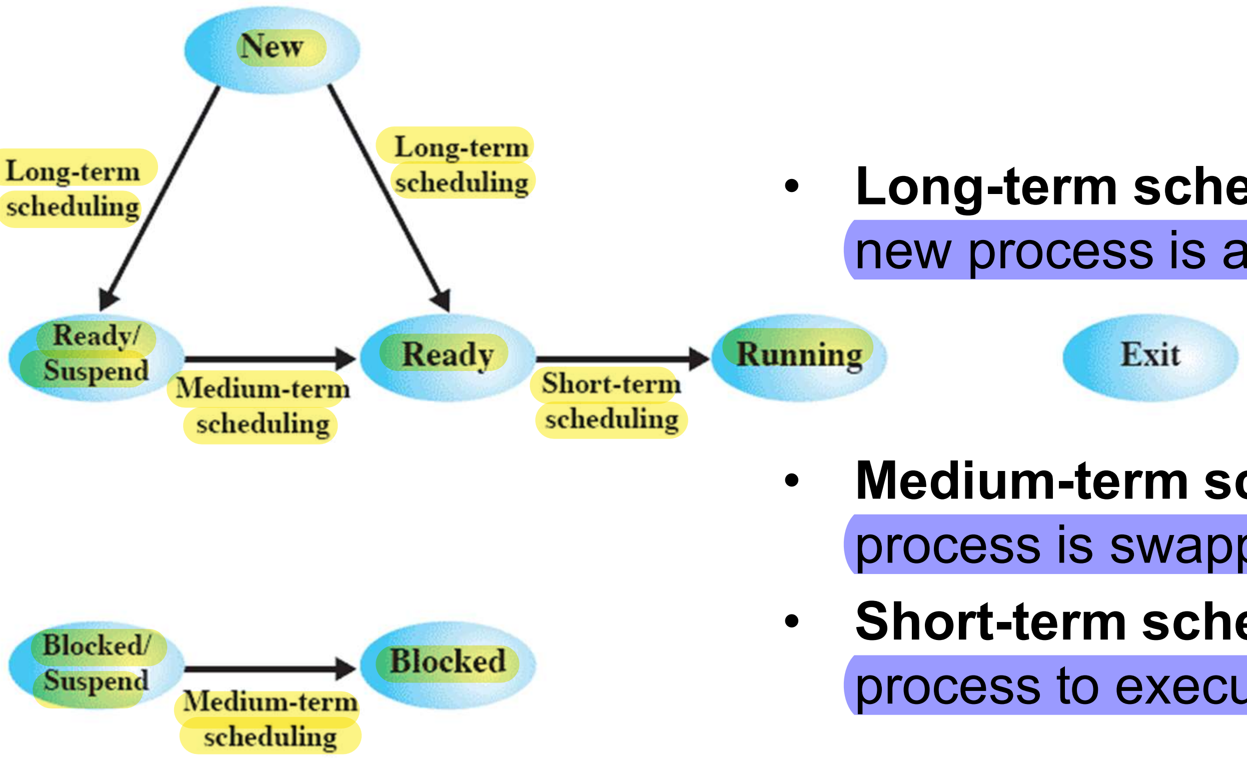


Figure 9.1 Scheduling and Process State Transitions

- **Long-term scheduling:** whether a new process is admitted.
- **Medium-term scheduling:** whether a process is swapped in.
- **Short-term scheduling:** which ready process to execute next.

Focus of this chapter

Queuing Diagram

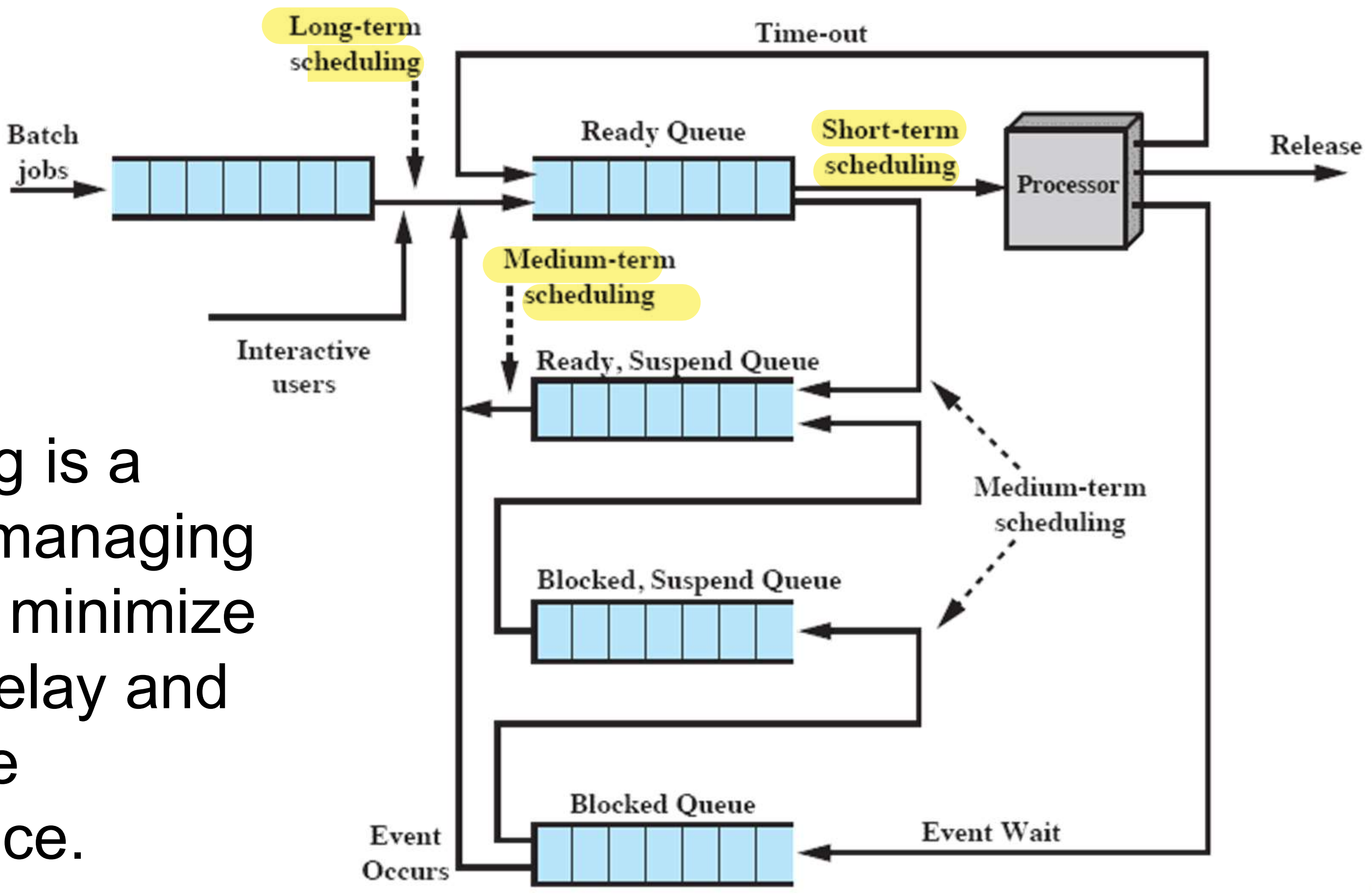
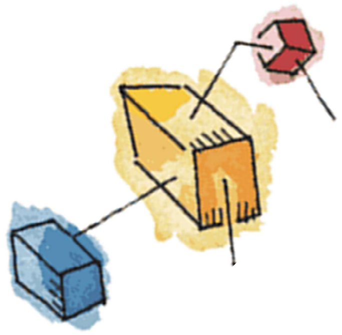


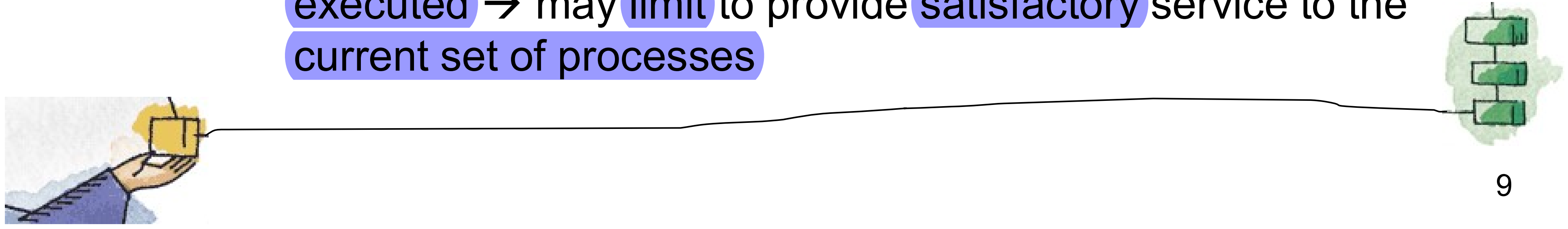
Figure 9.3 Queuing Diagram for Scheduling

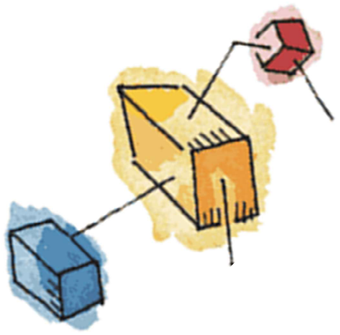
Scheduling is a matter of managing queues to minimize queuing delay and to optimize performance.



Long-Term Scheduling

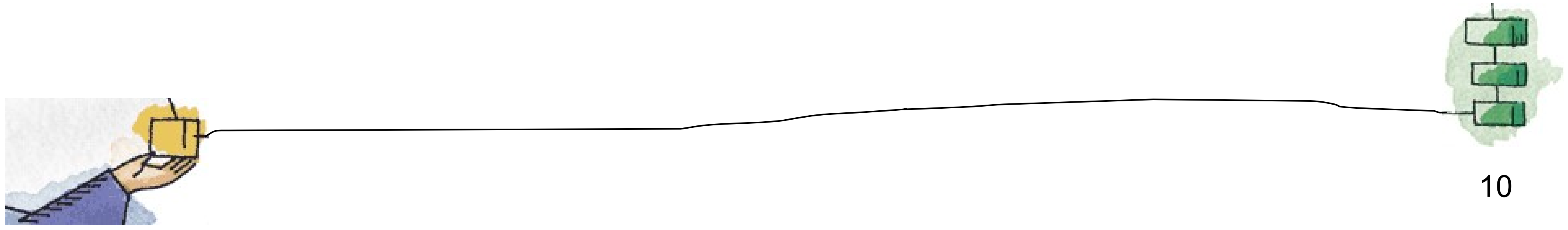
- Determines which programs are admitted to the system for processing
  - May be first-come-first-served
  - Or, according to criteria such as priority, I/O requirements or expected execution time
- Controls the ***degree of multiprogramming***
  - If more processes are admitted,
    - 👍 less likely that all processes will be blocked → better processor utilization
    - 👎 smaller percentage of time that each process can be executed → may limit to provide satisfactory service to the current set of processes

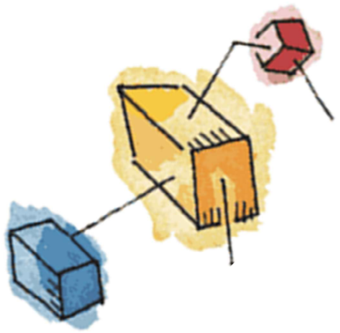




Medium-Term Scheduling

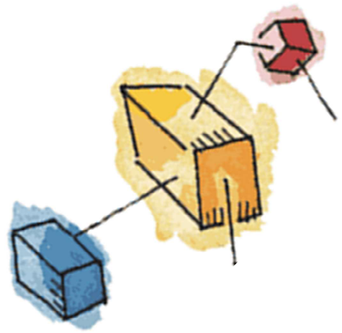
- Part of the swapping function
- Swapping-in decisions are based on
  - the need to manage the degree of multiprogramming
  - the memory requirements of the swapped-out processes





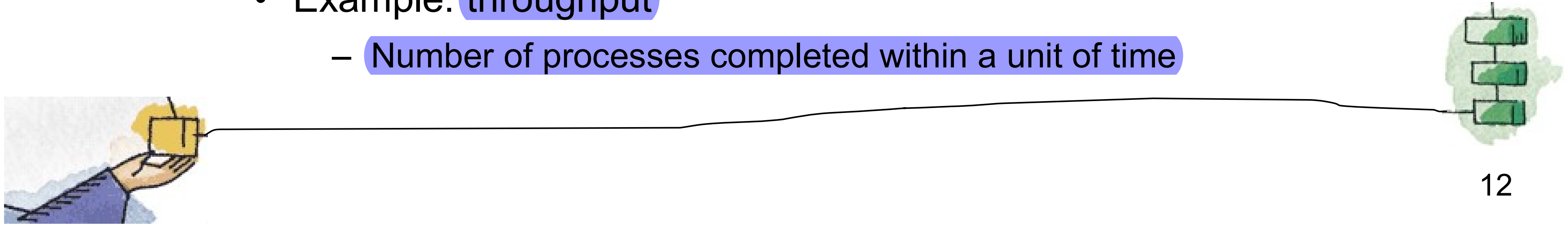
Short-Term Scheduling

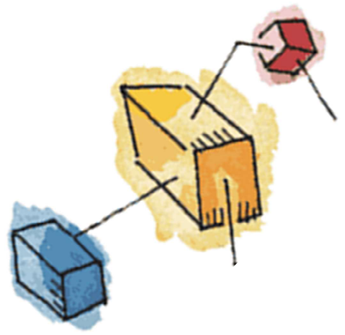
- Short-term scheduler is also known as the **dispatcher**
- Executes most frequently to decide which process to execute next
- Invoked when an event occurs that may lead to the **blocking of the current process** or that may provide an opportunity to **preempt a currently running process** in favor of another
  - Clock interrupts
  - I/O interrupts
  - OS calls
  - Signals (e.g., semaphores)



Short-Term Scheduling Criteria: User / System

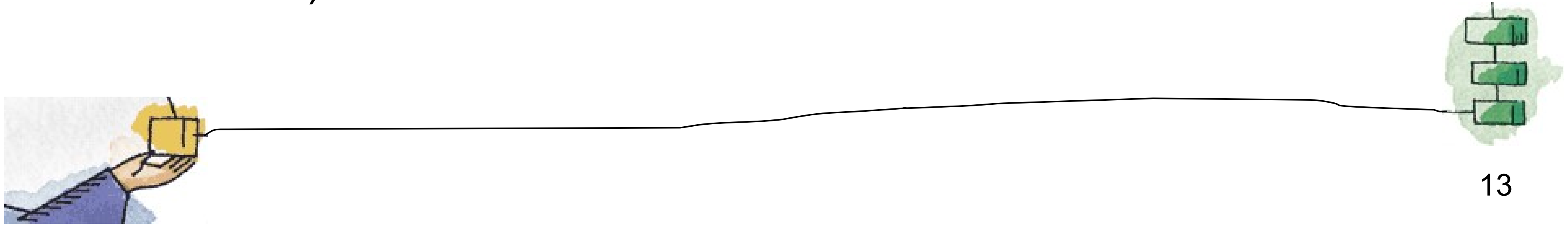
- Main objective is to allocate processor time to optimize certain aspects of system behaviour.
- A set of criteria is needed to evaluate the scheduling policy
  - User-oriented criteria
    - Behavior of the system as perceived by individual user or process
    - Example: response time in an interactive system
      - Elapsed time between the submission of a request until there is output.
  - System-oriented criteria
    - Effective and efficient utilization of the processor
    - Example: throughput
      - Number of processes completed within a unit of time

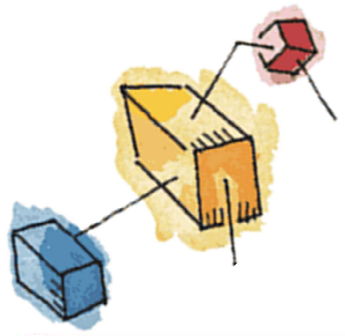




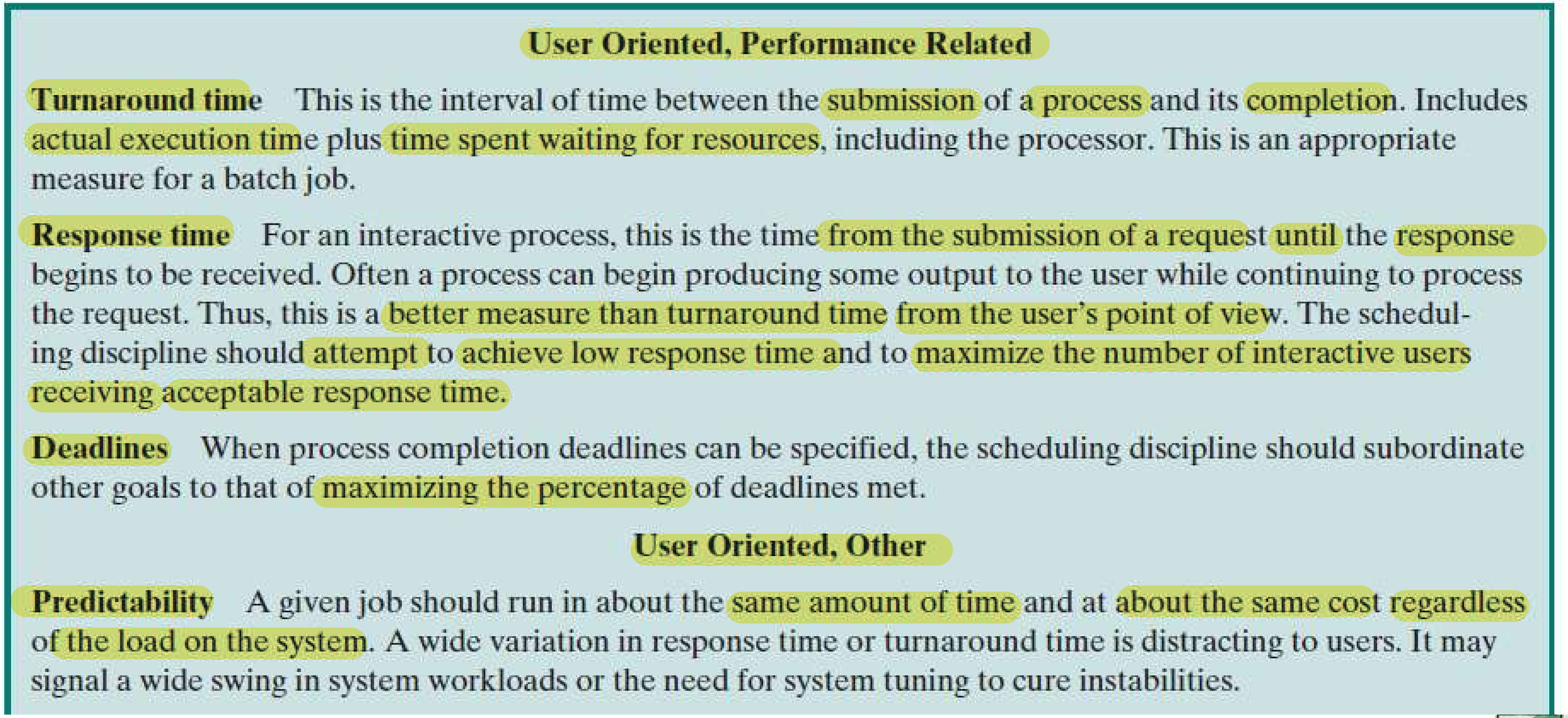
Short-Term Scheduling Criteria: Performance

- Performance-related
  - Quantitative
  - Easily measured
  - Example: response time and throughput
- Non-performance related
  - Qualitative
  - Hard to measure
  - Example: predictability (stable service provided to users over time)





Scheduling Criteria



User Oriented, Performance Related

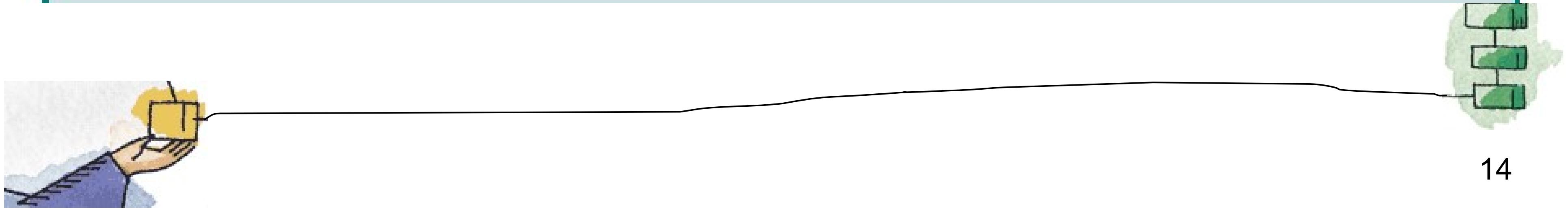
Turnaround time This is the interval of time between the submission of a process and its completion. Includes actual execution time plus time spent waiting for resources, including the processor. This is an appropriate measure for a batch job.

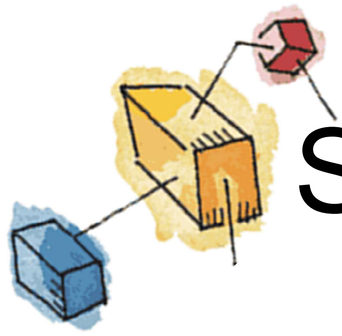
Response time For an interactive process, this is the time from the submission of a request until the response begins to be received. Often a process can begin producing some output to the user while continuing to process the request. Thus, this is a better measure than turnaround time from the user's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

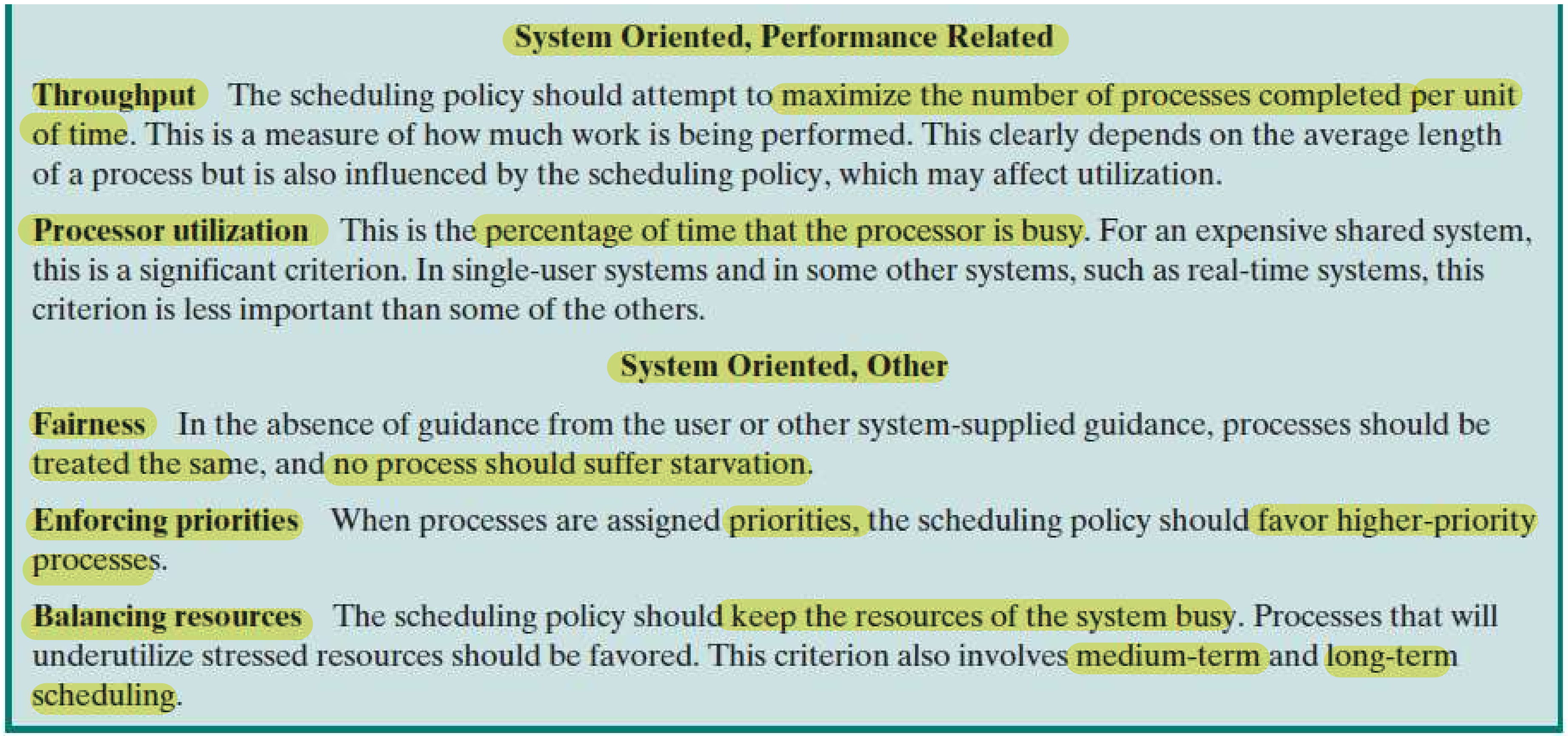
User Oriented, Other

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 (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.

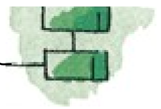
Processor utilization This is the percentage of time that the processor is busy. For an expensive shared system, this is a significant criterion. In single-user systems and in some other systems, such as real-time systems, this criterion is less important than some of the others.

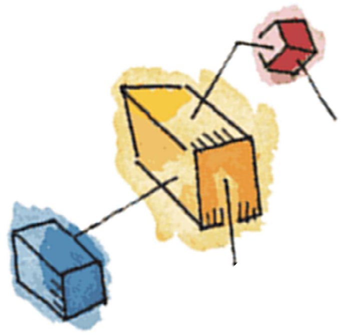
System Oriented, Other

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.

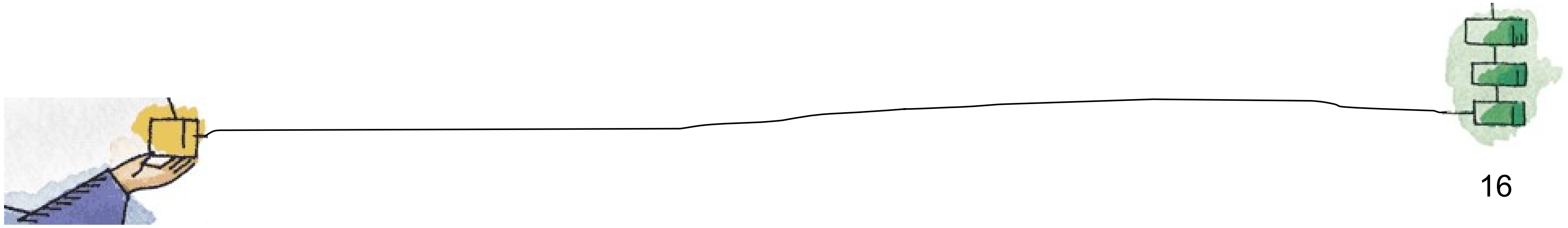
Balancing resources The scheduling policy should keep the resources of the system busy. Processes that will underutilize stressed resources should be favored. This criterion also involves medium-term and long-term scheduling.

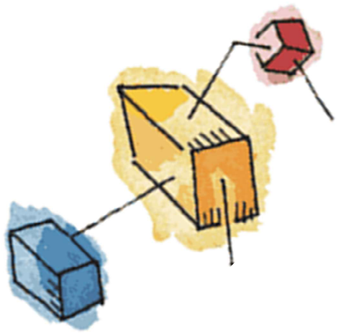




Interdependent Scheduling Criteria

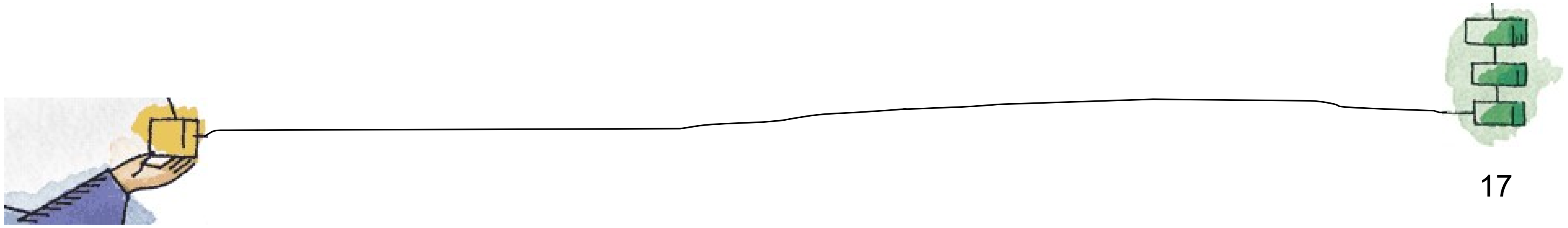
- Impossible to optimize all criteria simultaneously.
  - Example: response time vs. throughput
    - good response time requires frequent process switching which increases the system overhead, reducing throughput
- Design of a scheduling policy involves compromising among competing requirements.





Priorities

- In many systems, each process is assigned a priority.
- Scheduler will always choose a process of higher priority over one of lower priority
  - 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



Priority Queuing

- Have multiple ready queues, in descending order of priority: RQ0, RQ1, …, RQ*n*.
- The scheduler will start at the highest-priority ready queue (RQ0).
- 👎 Lower priority processes may suffer starvation.

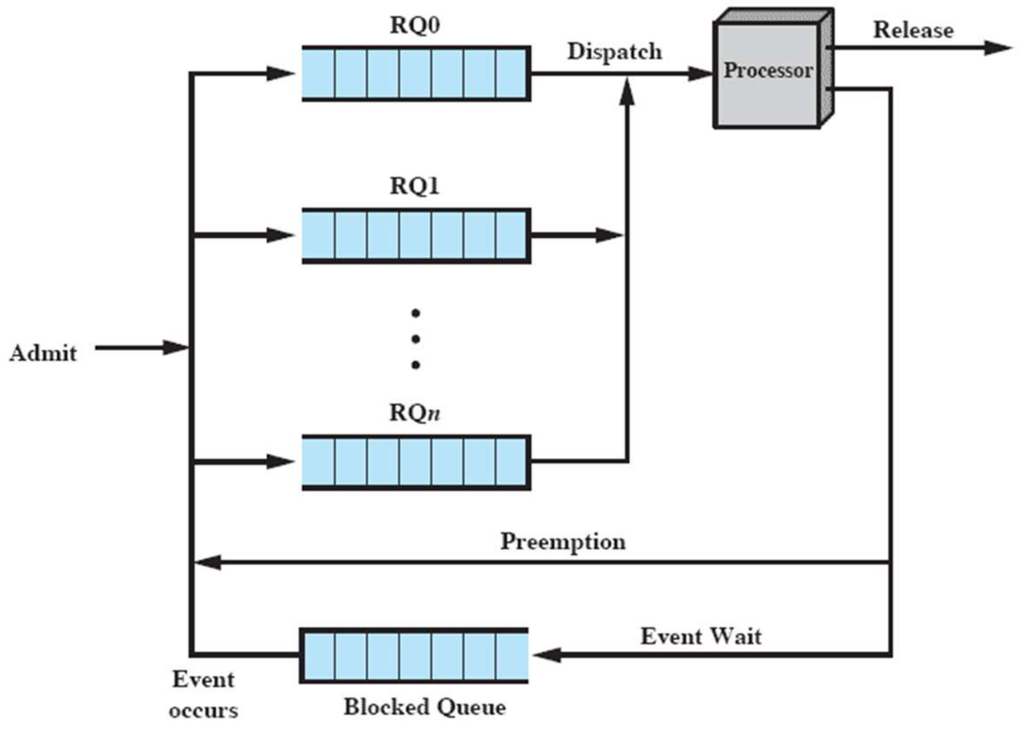


Figure 9.4 Priority Queuing

Alternative Scheduling Policies

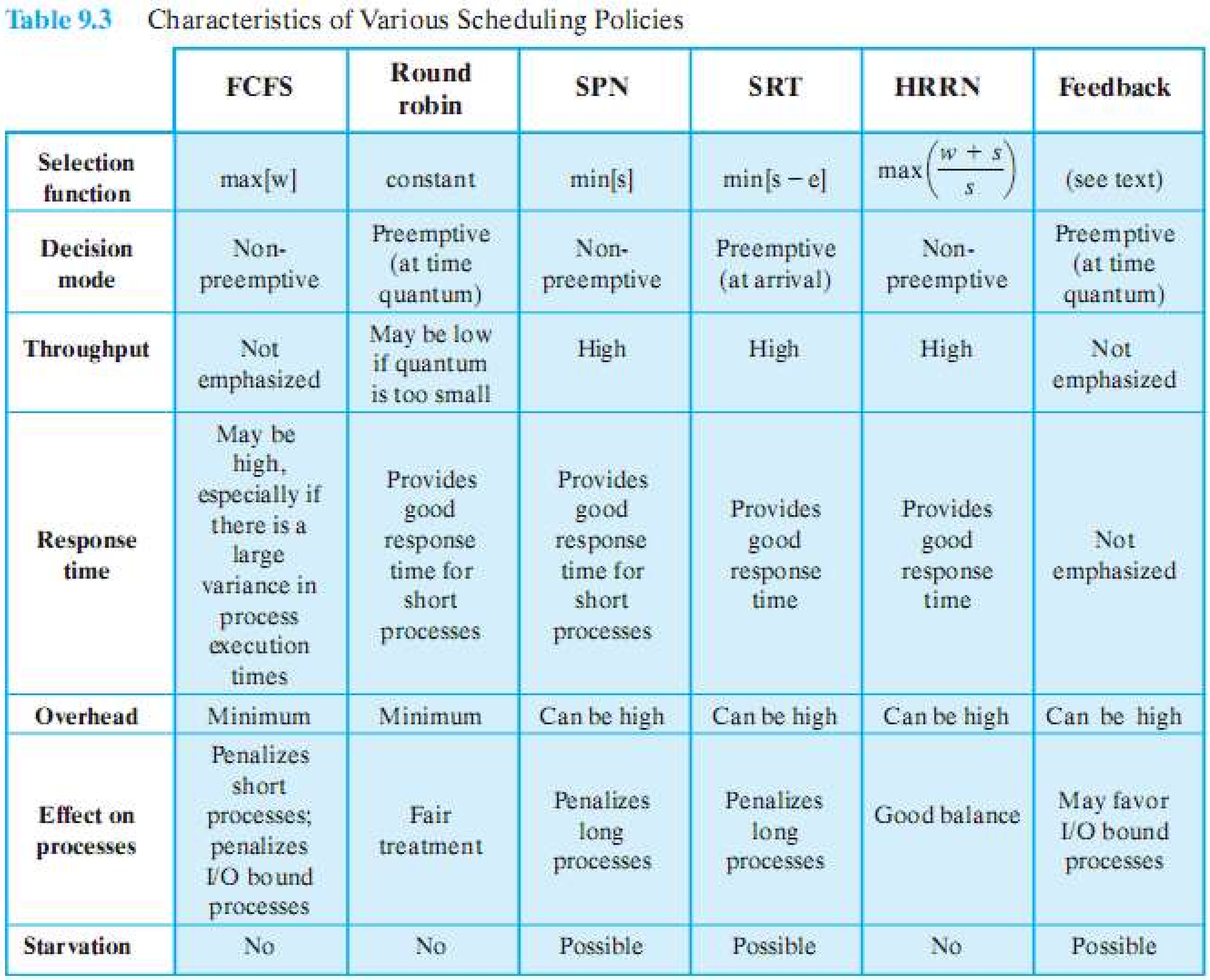
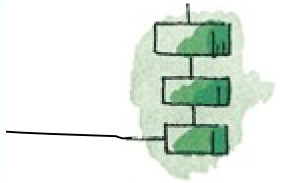


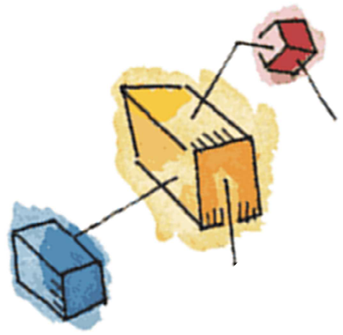
Table 9.3 Characteristics of Various Scheduling Policies

	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



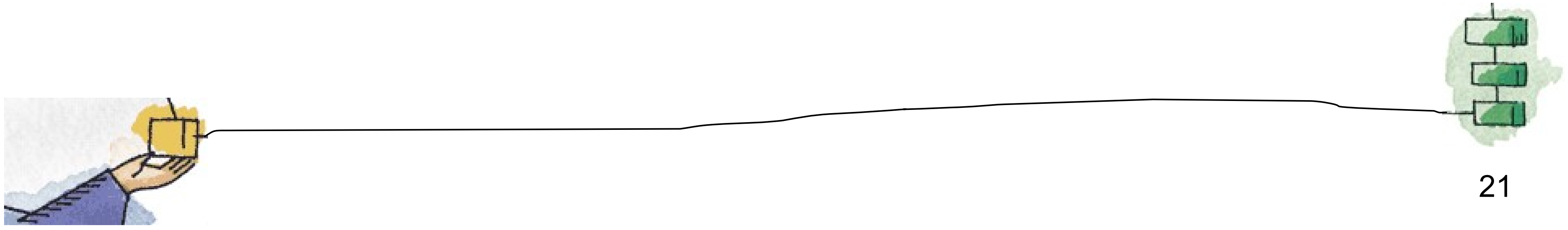
Roadmap

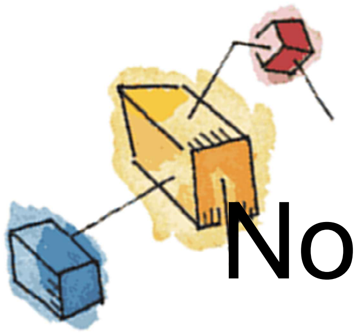
- Types of Processor Scheduling
- → Scheduling Algorithms
  - FCFS (First-Come-First-Serve)
  - RR (Round-Robin)
  - SPN (Shortest-Process-Next)
  - SRT (Shortest-Remaining-Time)
  - HRRN (Highest-Response-Ratio-Next)
  - FB (Feedback)
- Fair-Share Scheduling



Selection Function

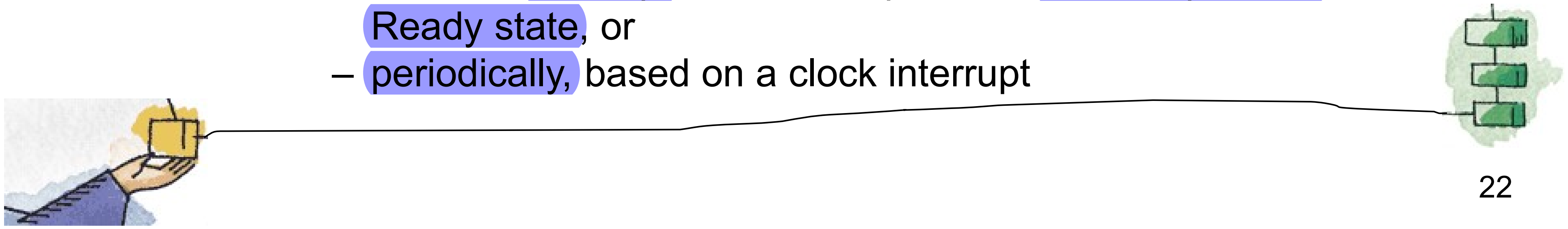
- Determines which process is selected next for execution
- Important quantities based on execution characteristics:
  - ***w*** = time spent waiting in system so far
  - ***e*** = time spent in execution so far
  - ***s*** = total service time required by the process, including *e;* generally, this quantity must be estimated or supplied by the user

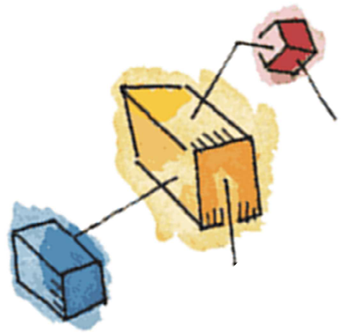




Decision Mode: Non-preemptive vs Preemptive

- Specifies the instants in time at which the selection function is exercised.
- Two categories of decision mode:
  - **Non-preemptive**
    - Once a process is in the running state, it will continue until it **terminates or blocks itself** for I/O or OS service
  - **Preemptive**
    - Currently running process may be **interrupted** and moved to ready state by the OS
    - Preemption may occur
      - 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



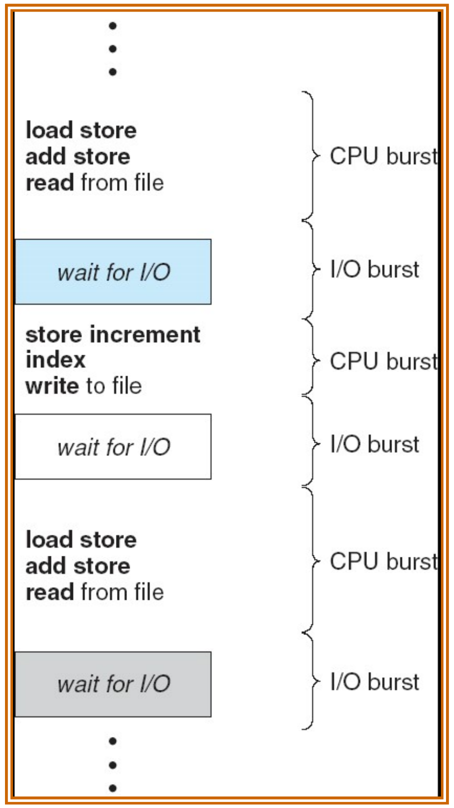


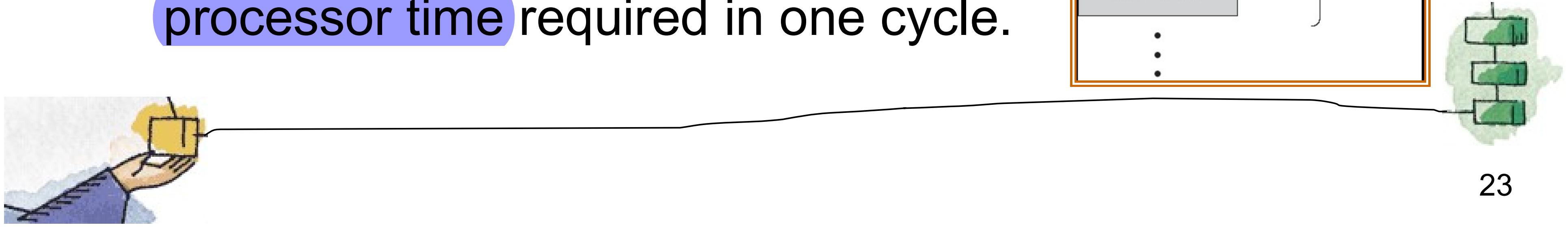
Process Scheduling Example

- Example set of processes

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

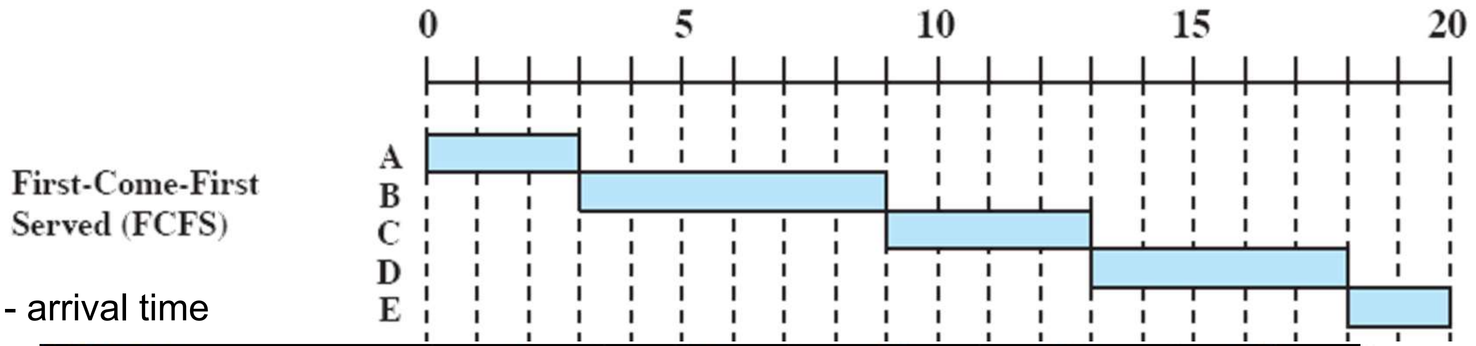
- Process requires alternate use of the processor and I/O in a repetitive fashion.
- The service times represent the processor time required in one cycle.





First-Come-First-Served (FCFS)

- Also known as first-in-first-out (FIFO) or strict queuing
- When the current process ceases to execute, select the process that has been in the ready queue the longest

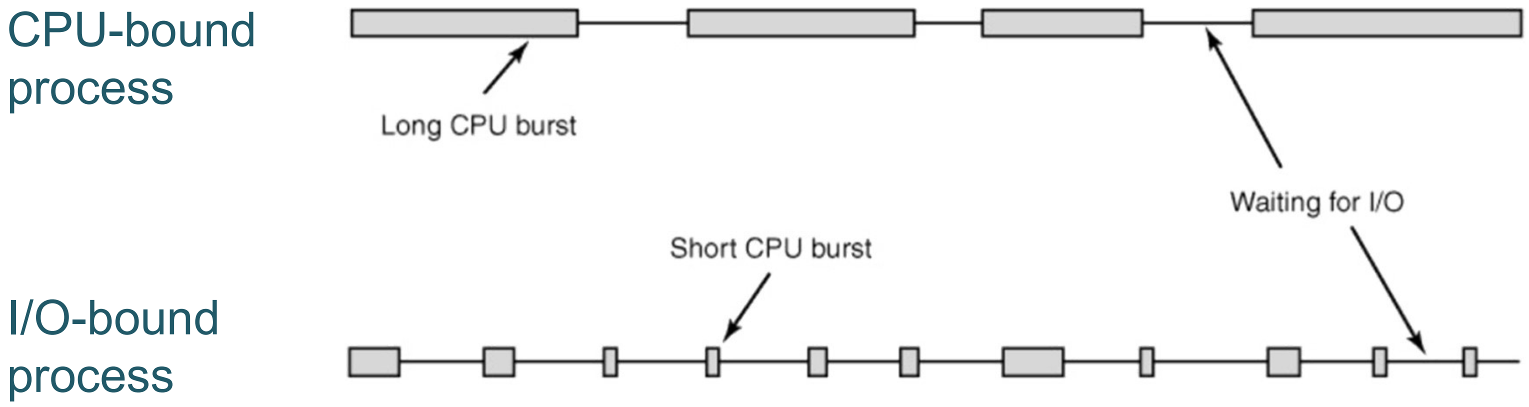


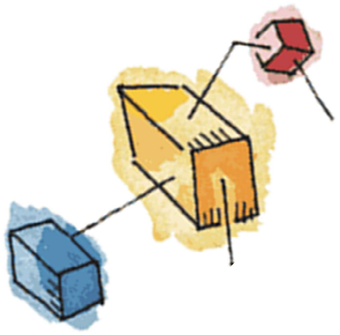
Tr = finish time - arrival time

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

FCFS Performance

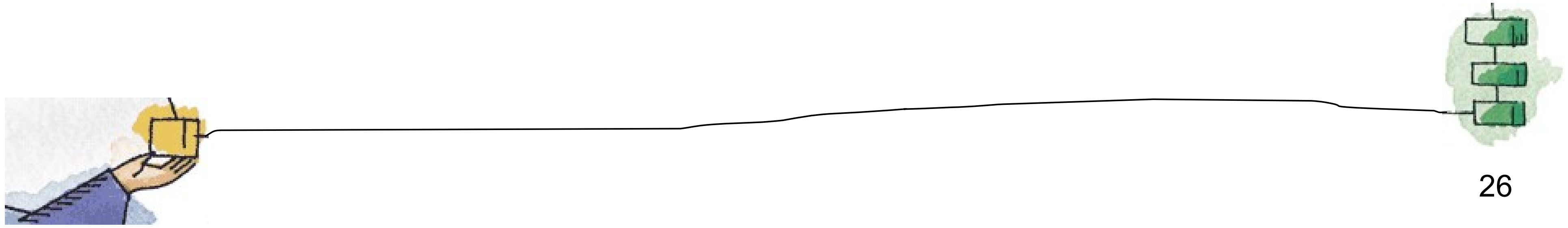
- Favors long processes over short ones
  - A short process has to wait a long time when it arrives just after a long process
- Favors CPU-bound processes over I/O-bound ones
  - May result in inefficient use of both the processor and the I/O devices



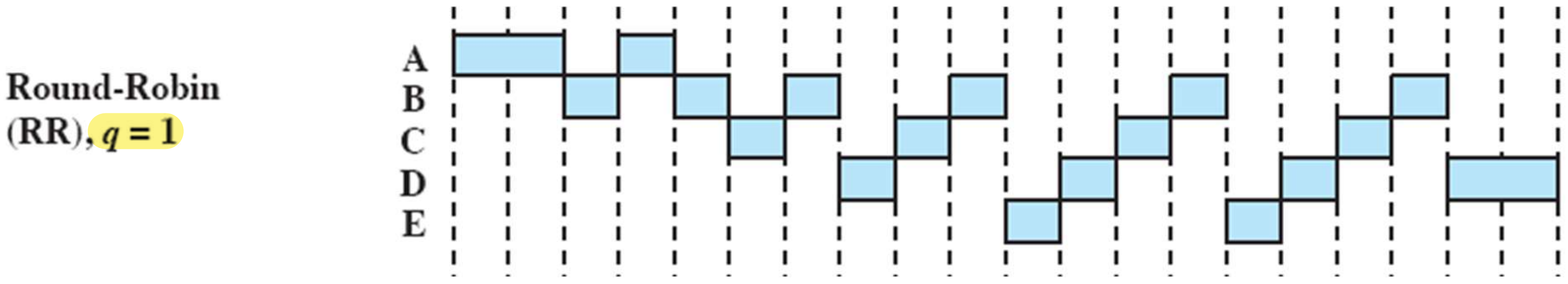


Round Robin (RR)

- RR uses **preemption** based on a clock
  - Clock interrupts are generated at periodic intervals
  - When an interrupt occurs, the currently running process is placed in the ready queue, select next ready job on a FCFS basis
  - also known as ***time slicing***, because each process is given a slice of time (***time quantum***) before being preempted.
  - 👍 Reduce the penalty that short jobs suffer with FCFS
  - 👍 Particularly effective in a general-purpose time-sharing system



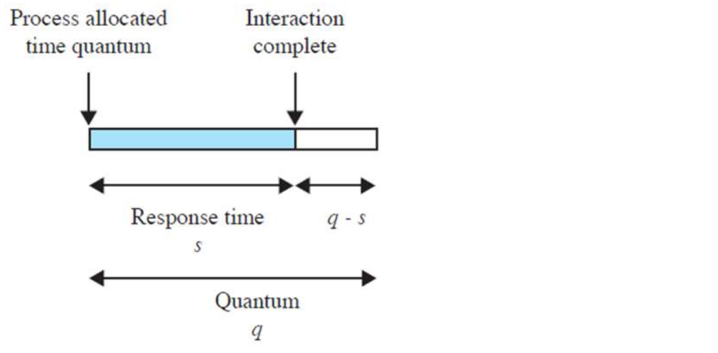
Round Robin



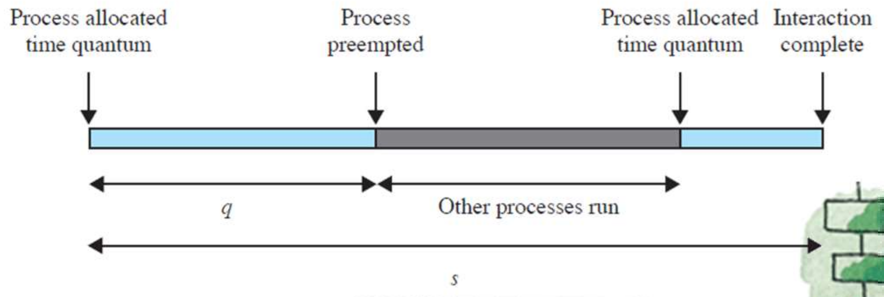
Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
RR $q = 1$						
Finish Time	4	18	17	20	15	
Turnaround Time (T_r)	4	16	13	14	7	10.80
T_r/T_s	1.33	2.67	3.25	2.80	3.50	2.71

Effect of Size of Preemption Time Quantum

- Principal design issue: length of the time quantum
  - Very short time quanta should be avoided because there is processing over-head in handling the clock interrupt and performing the scheduling and dispatching function.
  - The time quantum should be slightly greater than the time required for a typical interaction.

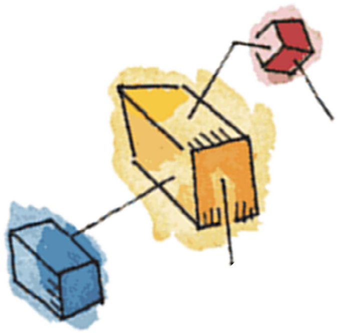


(a) Time quantum greater than typical interaction



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

(b) Time quantum less than typical interaction

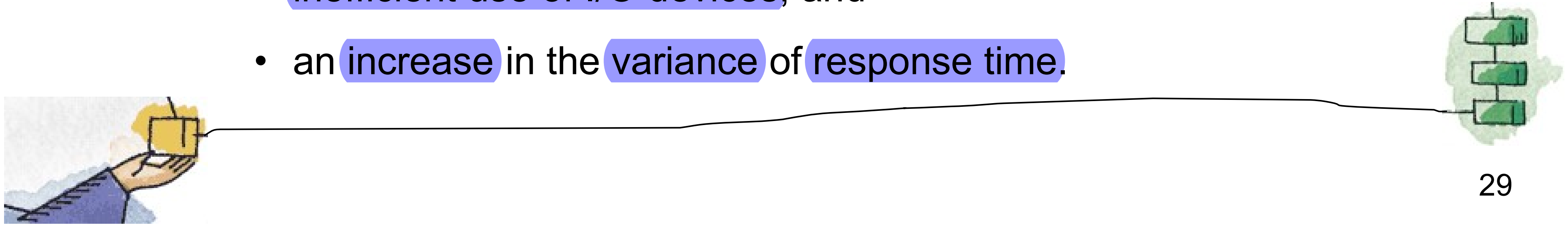


RR Performance

- 👎 Favors CPU-bound processes over I/O-bound ones
  - An I/O-bound process uses a processor for a ***short*** period and then is ***blocked*** before joining the ready queue again

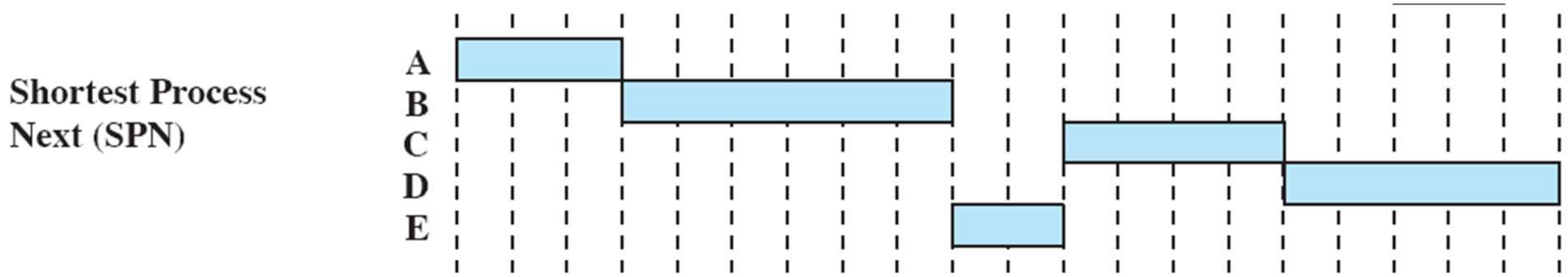
    vs.

    A CPU-bound process uses a ***complete*** time quantum while executing and ***immediately*** returns to the ready queue.
  - CPU-bound processes receive a greater portion of processor time →
    - poor performance for I/O-bound processes
    - inefficient use of I/O devices, and
    - an increase in the variance of response time.

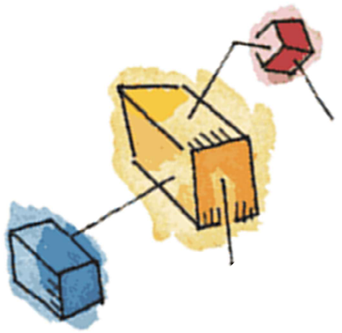


Shortest Process Next (SPN)

- **Nonpreemptive** policy
- Select process with ***shortest expected*** processing time
- 👍 Reduce the bias in favor of long processes in FCFS by allowing short processes jump ahead of longer processes.

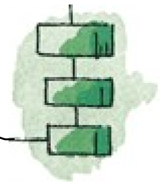


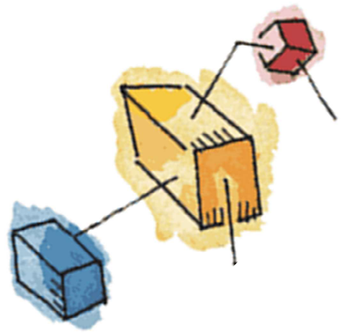
Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
Finish Time	3	9	15	20	11	
Turnaround Time (T_r)	3	7	11	14	3	7.60
T_r/T_s	1.00	1.17	2.75	2.80	1.50	1.84



SPN Performance

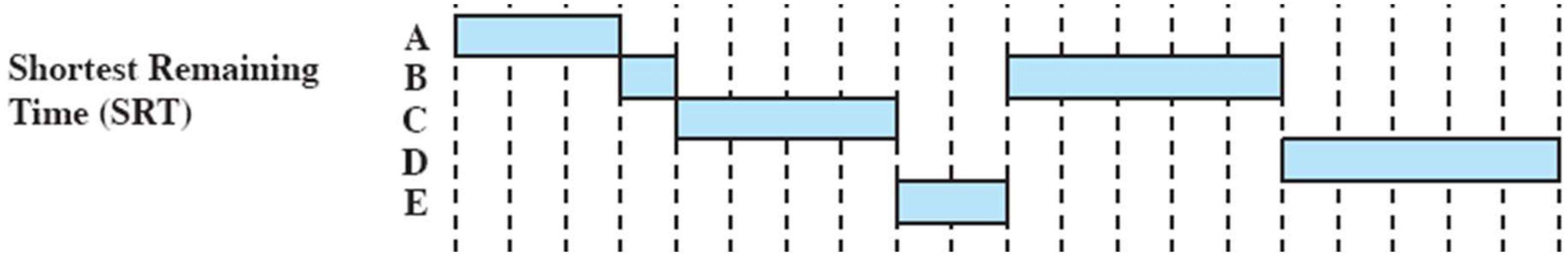
- 👍 Overall performance is significantly improved in terms of response time
- 👎 Need to know or estimate the required processing time of each process
  - For batch jobs, programmers may be required to supply an estimate
  - For repeating jobs, statistics may be gathered
- 👎 Variability of response times is increased, especially for longer processes
- 👎 Possibility of starvation for longer processes
- 👎 Because of the lack of preemption, late short processes may still be heavily penalized.



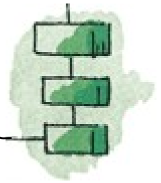


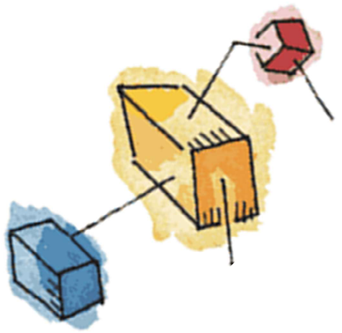
Shortest Remaining Time (SRT)

- **Preemptive** version of SPN
- Select the process with the ***shortest expected remaining*** processing time



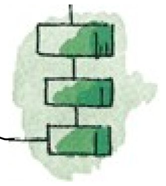
Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
SRT						
Finish Time	3	15	8	20	10	
Turnaround Time (T_r)	3	13	4	14	2	7.20
T_r/T_s	1.00	2.17	1.00	2.80	1.00	1.59

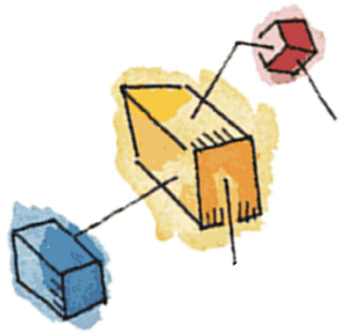




SRT Performance

- 👍 Does not have the bias in favor of long processes found in FCFS.
- 👍 Unlike RR, no additional interrupts are generated, reducing overhead.
  - Preemption may occur when a new process becomes ready.
- 👍 Give superior turnaround time performance to SPN, because a short job is given immediate preference to a running longer job.
- 👎 Must estimate processing time and record elapsed service time.
- 👎 A risk of starvation of longer processes.



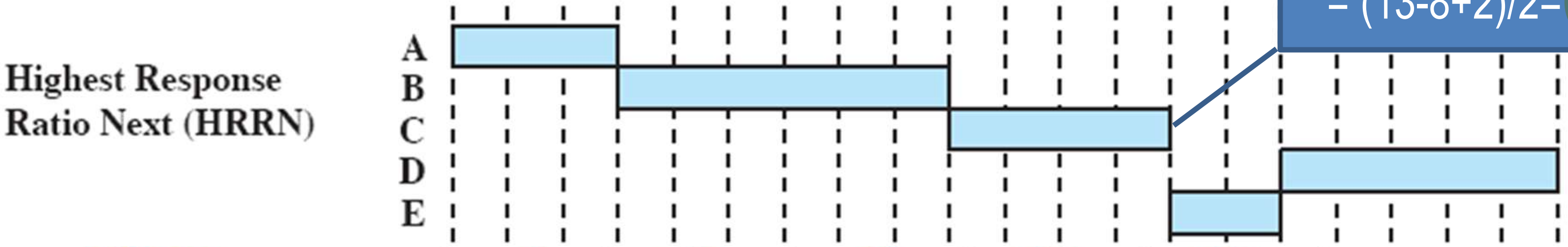


Highest Response Ratio Next

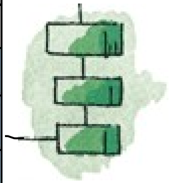
- **Non-preemptive**
- Select next process with the ***greatest ratio***, trying to minimize the ***normalized turnaround time***

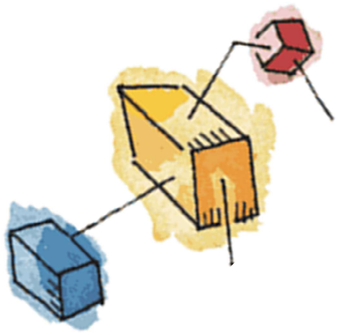
$$Ratio = \frac{time\ spent\ waiting + expected\ service\ time}{expected\ service\ time}$$

Response ratio (D) = (13-6+5)/5 = 2.4
Response ratio (E) = (13-8+2)/2= 3.5



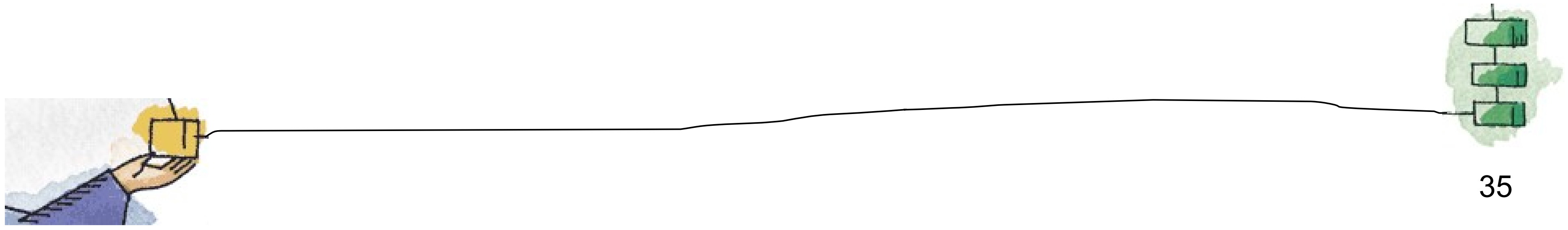
Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
HRRN						
Finish Time	3	9	13	20	15	
Turnaround Time (T_r)	3	7	9	14	7	8.00
T_r/T_s	1.00	1.17	2.25	2.80	3.5	2.14





HRRN Performance

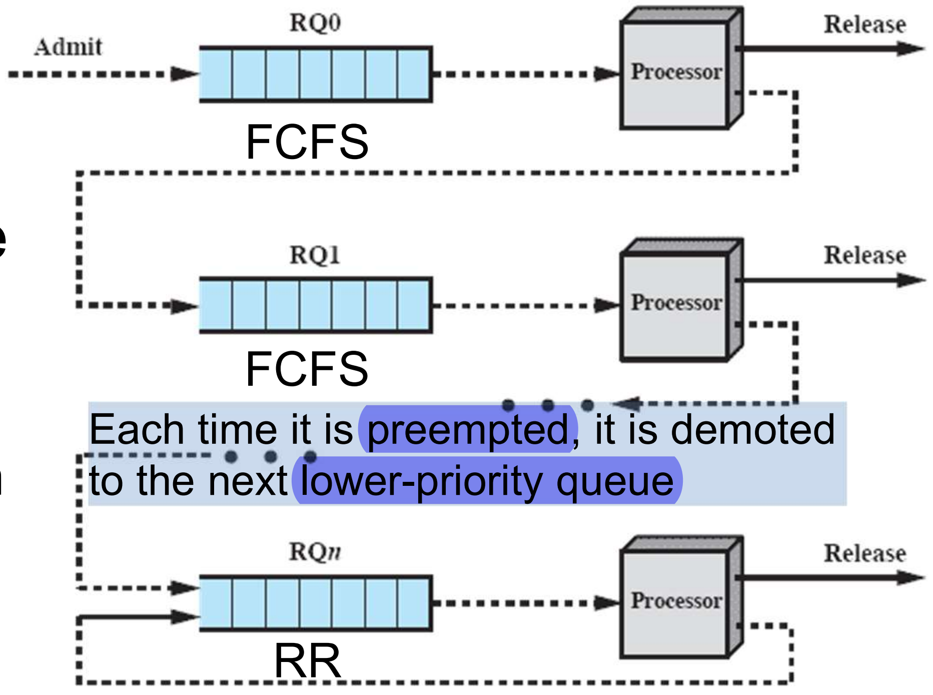
- 👍 Shorter jobs are favored (a smaller denominator yields a larger ratio).
- 👍 Longer processes will eventually get past competing shorter jobs (aging without service also increases the ratio).
- 👎 As with SRT and SPN, the expected service time must be estimated.



Feedback Scheduling (FB)

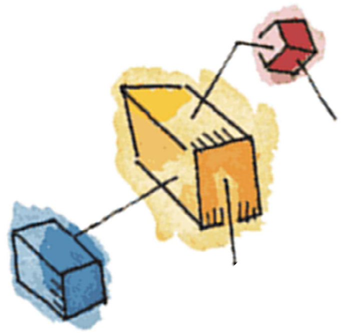
- Service time may not be available or cannot be estimated.
- Alternative solution: scheduling with preemptive and dynamic priority mechanism
  - Penalize jobs that have been running longer by demoting to the next lower-priority queue
  - Favors newer, shorter processes over older, longer ones.

When a process first enters the system, it is placed in the highest-priority queue



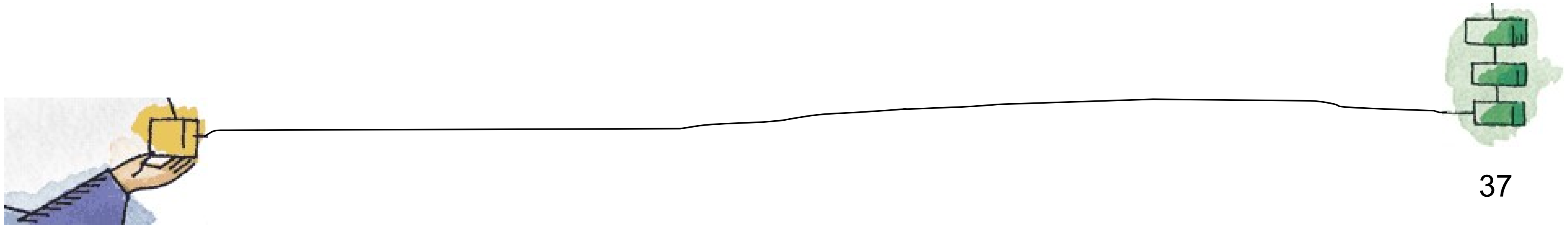
Each time it is preempted, it is demoted to the next lower-priority queue

Figure 9.10 Feedback Scheduling



Feedback Performance

- Variations:
  - Pre-empts periodically, similar to round robin
    - Longer processes may suffer starvation if new jobs are entering the system frequently
  - A process scheduled from RQ*i* is allowed to execute 2^i time units before preemption
    - allow a greater time allocation at lower priority
  - Promote a process to a higher-priority queue after it spends a certain amount of time waiting for service in its current queue



Performance Comparison

- It is impossible to make definitive comparisons because relative performance will depend on a variety of factors.

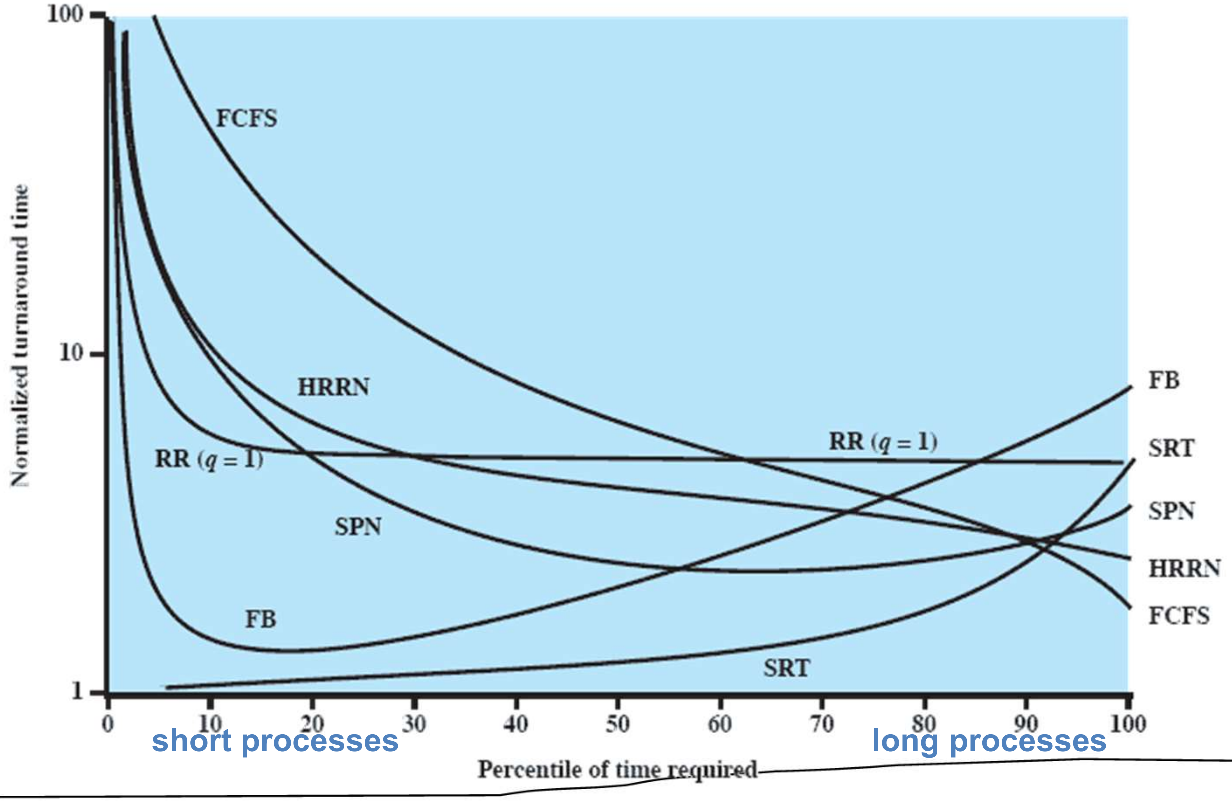


Figure 9.14 Simulation Results for Normalized Turnaround Time

Roadmap

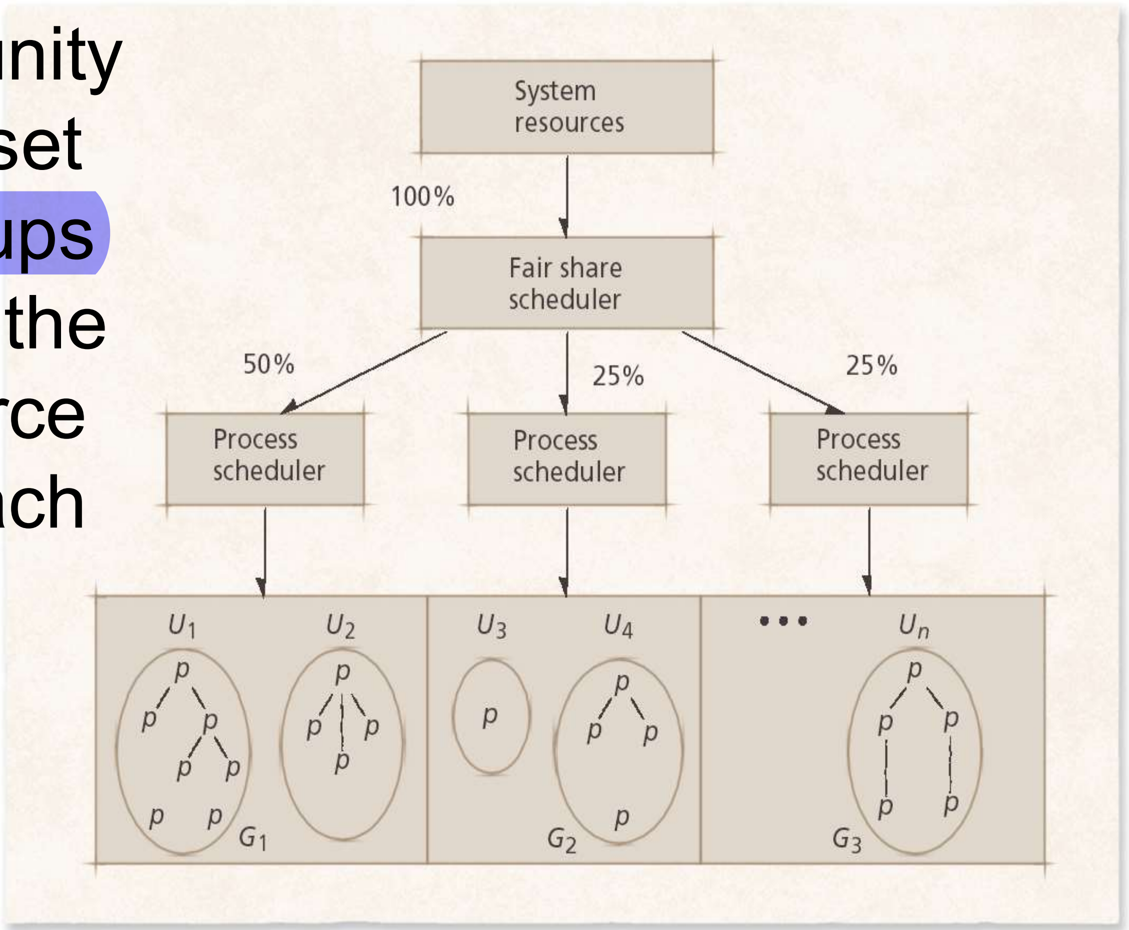
- Types of Processor Scheduling
- Scheduling Algorithms
  - FCFS (First-Come-First-Serve)
  - RR (Round-Robin)
  - SPN (Shortest-Process-Next)
  - SRT (Shortest-Remaining-Time)
  - HRRN (Highest-Response-Ratio-Next)
  - FB (Feedback)
- → Fair-Share Scheduling

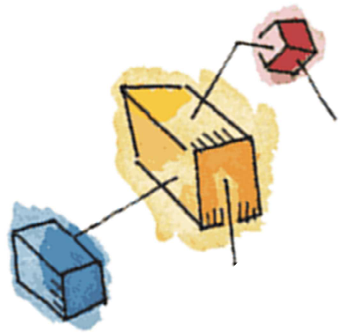
Fair-Share Scheduling

- User's application runs as a collection of processes (threads).
- User is more concerned about the performance of the application, not a particular process.
- ***Fair-share scheduling***: make scheduling decisions based on process sets.
- This concept can be extended to groups of users.

Fair-Share Scheduling

- The user community is divided into a set of fair-share groups and a fraction of the processor resource is allocated to each group.





Fair-Share Scheduling

- Scheduling is done on the basis of priority.
- The fair-share groups are prioritized by how close they are to achieving their fair share.
  - Groups doing poorly receive higher priority
  - Groups doing well receive lower priority
- Objectives:
  - To give ***fewer*** resources to users who have had ***more than their fair share*** and
  - To give ***more*** to those who have had ***less than their fair share***