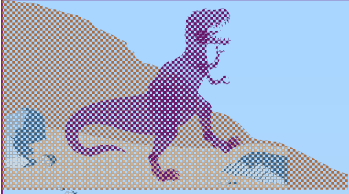


Chapter 6: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Algorithm Evaluation



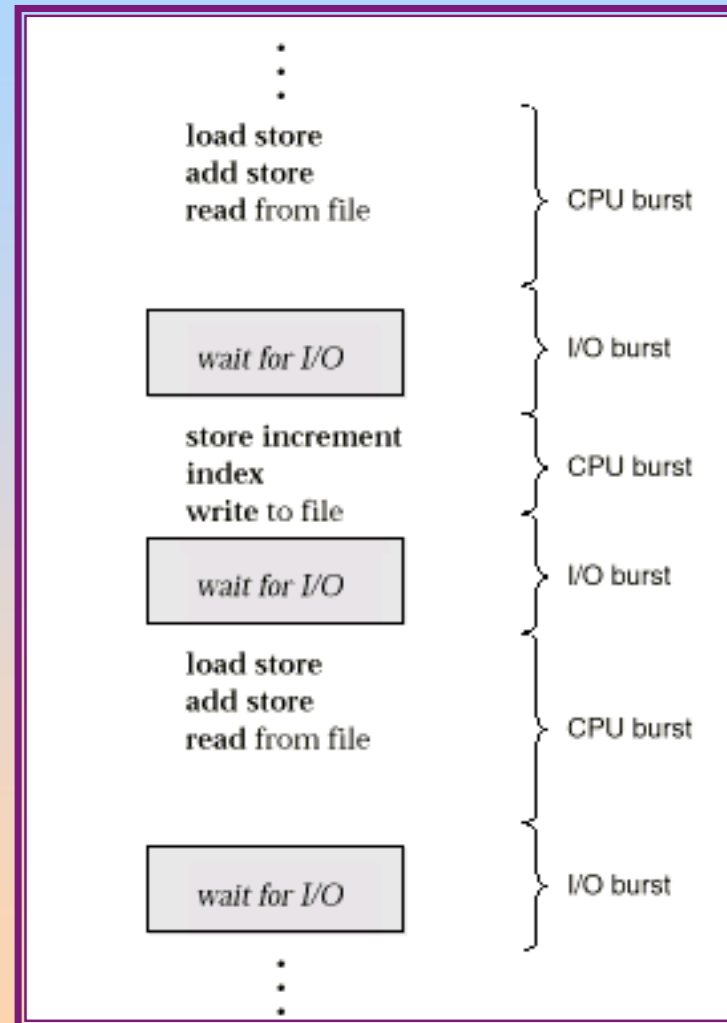


Basic Concepts

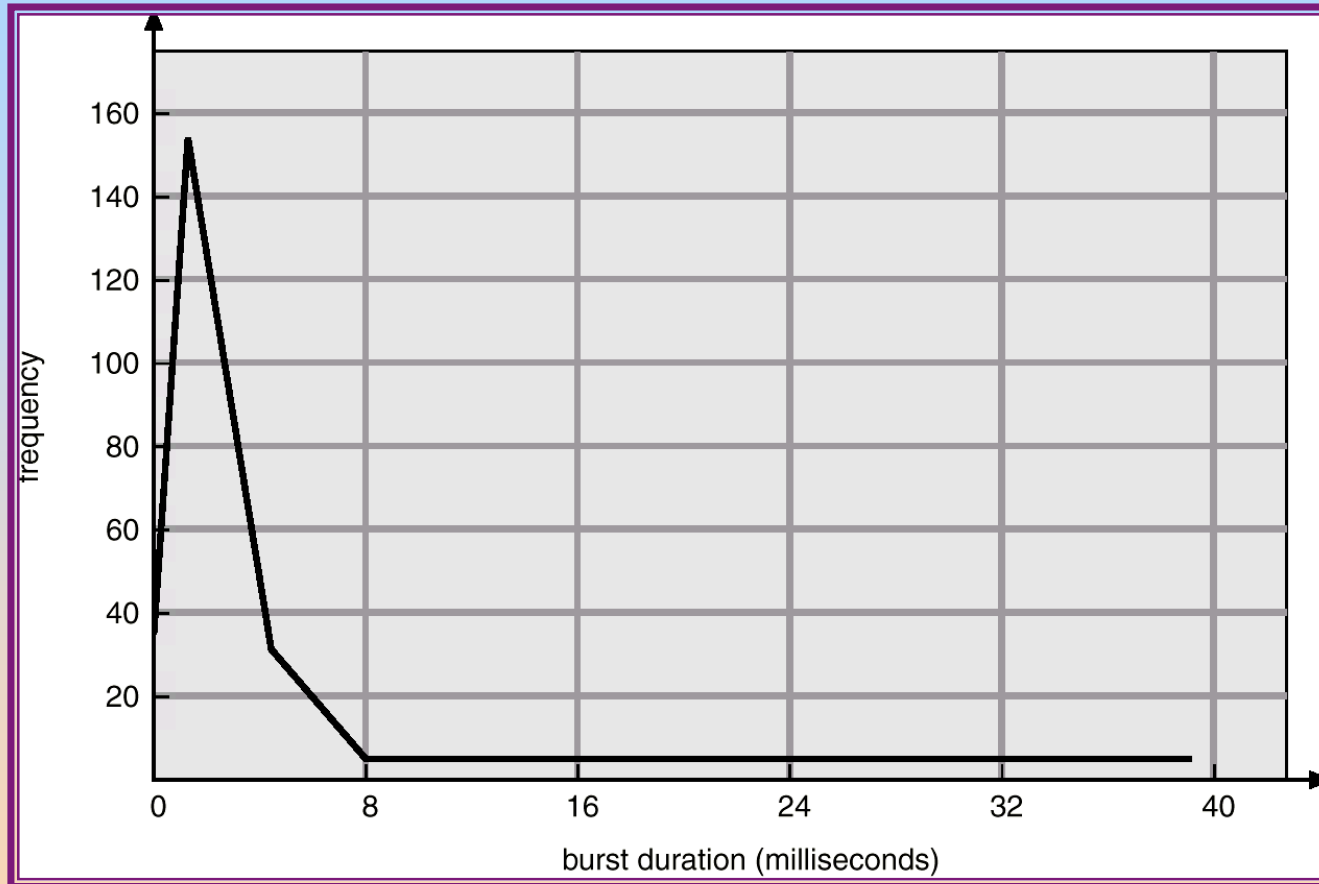
- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait.
- CPU burst distribution

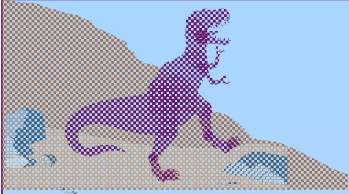


Alternating Sequence of CPU And I/O Bursts



Histogram of CPU-burst Times





CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
 1. Switches from running to waiting state.
 2. Switches from running to ready state.
 3. Switches from waiting to ready.
 4. Terminates.
- Scheduling under 1 and 4 is *nonpreemptive*.
- All other scheduling is *preemptive*.

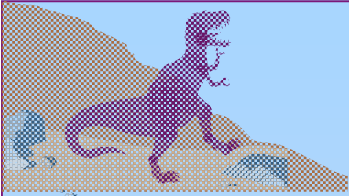




Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
 - ◆ switching context
 - ◆ switching to user mode
 - ◆ jumping to the proper location in the user program to restart that program
- *Dispatch latency* – time it takes for the dispatcher to stop one process and start another running.

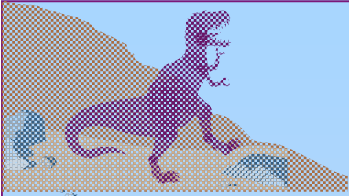




Scheduling Criteria

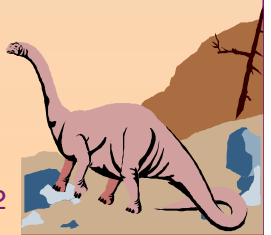
- CPU utilization – keep the CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process
- Waiting time – amount of time a process has been waiting in the ready queue
- Response time – amount of time it takes from when a request was submitted until the first response is produced, **not** output (for time-sharing environment)

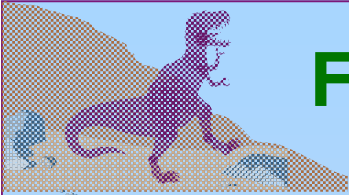




Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

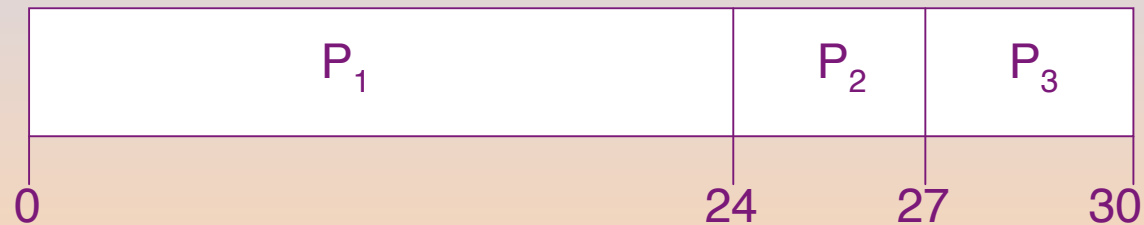




First-Come, First-Served (FCFS) Scheduling

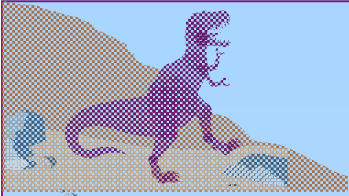
| <u>Process</u> | <u>Burst Time</u> |
|----------------|-------------------|
| P_1 | 24 |
| P_2 | 3 |
| P_3 | 3 |

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
The Gantt Chart for the schedule is:



- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$



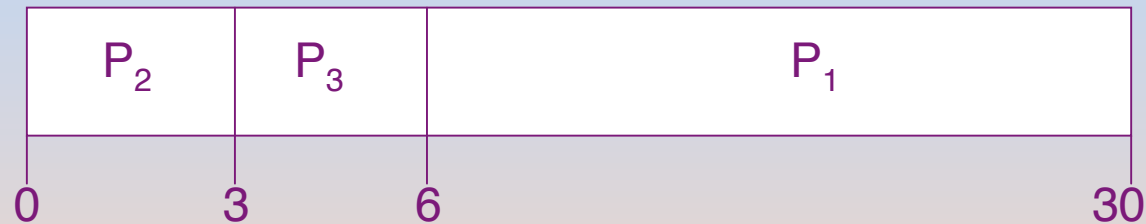


FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

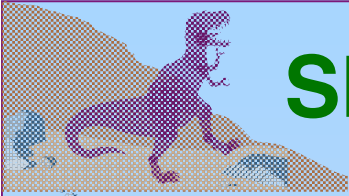
$$P_2, P_3, P_1.$$

■ The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case.
- *Convoy effect* short process behind long process

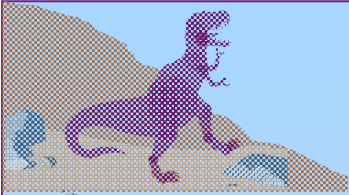




Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
 - ✦ nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
 - ✦ preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes.

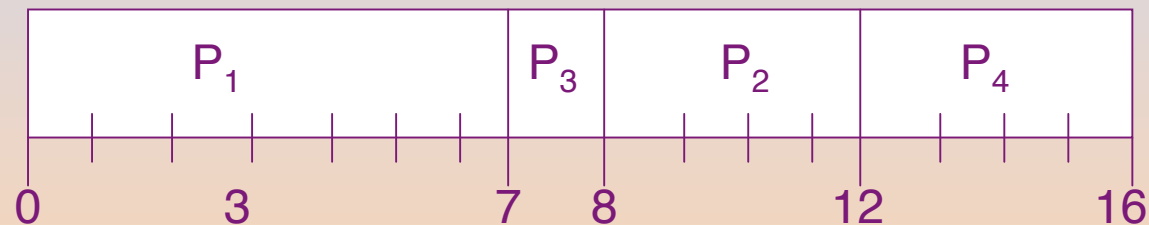




Example of Non-Preemptive SJF

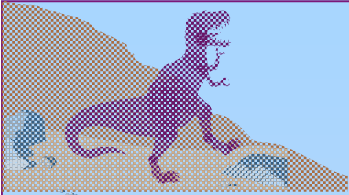
| <u>Process</u> | <u>Arrival Time</u> | <u>Burst Time</u> |
|----------------|---------------------|-------------------|
| P_1 | 0.0 | 7 |
| P_2 | 2.0 | 4 |
| P_3 | 4.0 | 1 |
| P_4 | 5.0 | 4 |

■ SJF (non-preemptive)



■ Average waiting time = $(0 + 6 + 3 + 7)/4 - 4$

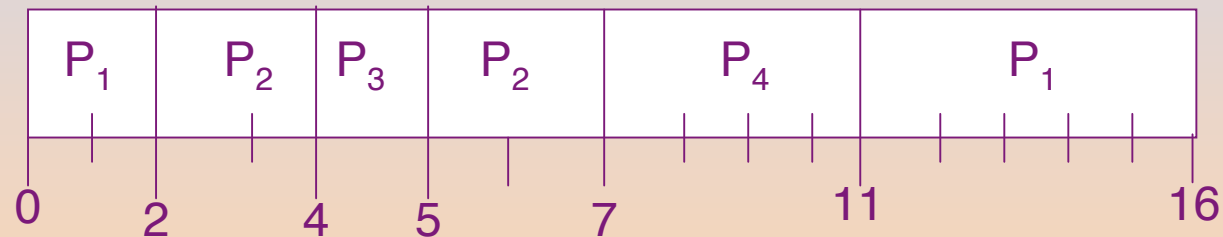




Example of Preemptive SJF

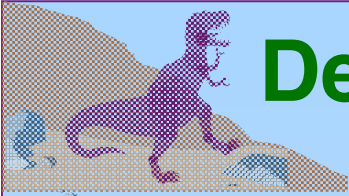
| <u>Process</u> | <u>Arrival Time</u> | <u>Burst Time</u> |
|----------------|---------------------|-------------------|
| P_1 | 0.0 | 7 |
| P_2 | 2.0 | 4 |
| P_3 | 4.0 | 1 |
| P_4 | 5.0 | 4 |

■ SJF (preemptive)



■ Average waiting time = $(9 + 1 + 0 + 2)/4 - 3$





Determining Length of Next CPU Burst

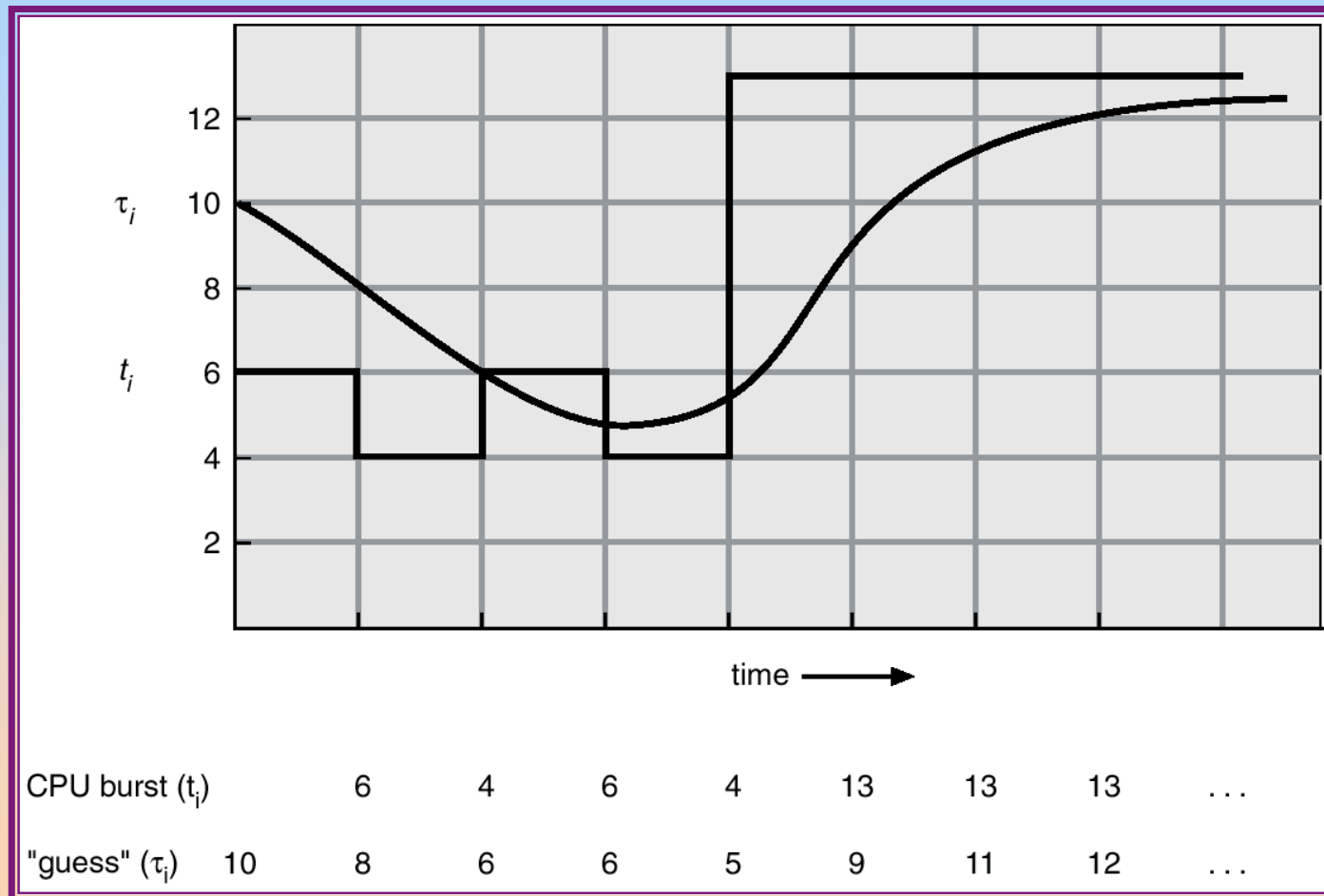
- Can only estimate the length.
- Can be done by using the length of previous CPU bursts, using exponential averaging.

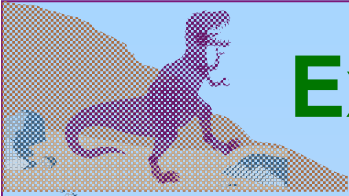
1. t_n = actual length of n^{th} CPU burst
2. \bar{t}_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define :

$$\bar{t}_{n+1} = \alpha t_n + (1 - \alpha) \bar{t}_n.$$



Prediction of the Length of the Next CPU Burst





Examples of Exponential Averaging

□ $\alpha = 0$

□ $\tau_{h+1} = \tau_h$

□ Recent history does not count.

□ $\alpha = 1$

□ $\tau_{h+1} = t_n$

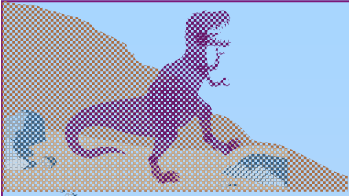
□ Only the actual last CPU burst counts.

□ If we expand the formula, we get:

$$\begin{aligned}\tau_{h+1} = & \alpha t_n + (1 - \alpha) \tau_{n-1} + \dots \\ & + (1 - \alpha)^j \tau_{n-1} + \dots \\ & + (1 - \alpha)^{n-1} \tau_0\end{aligned}$$

□ Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.

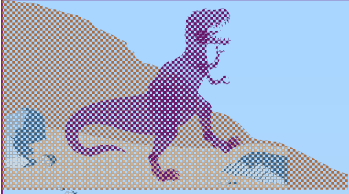




Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer \equiv highest priority).
 - ✦ Preemptive
 - ✦ nonpreemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time.
- Problem \equiv Starvation – low priority processes may never execute.
- Solution \equiv Aging – as time progresses increase the priority of the process.





Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q , then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Performance
 - ◆ q large □ FIFO
 - q small □ q must be large with respect to context switch, otherwise overhead is too high.

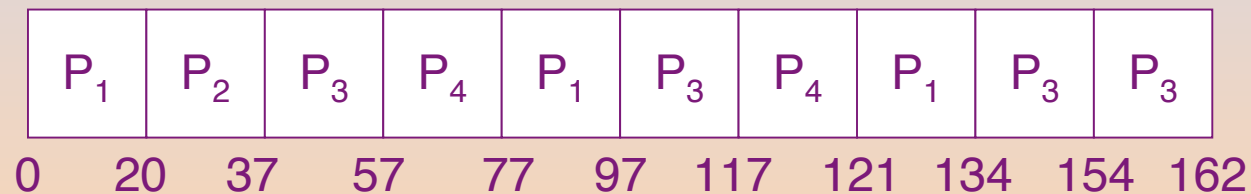




Example of RR with Time Quantum = 20

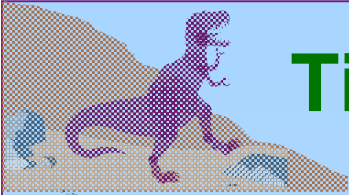
| <u>Process</u> | <u>Burst Time</u> |
|----------------|-------------------|
| P_1 | 53 |
| P_2 | 17 |
| P_3 | 68 |
| P_4 | 24 |

■ The Gantt chart is:

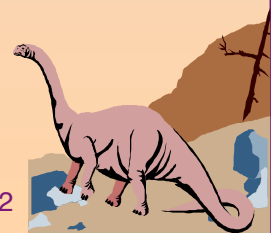
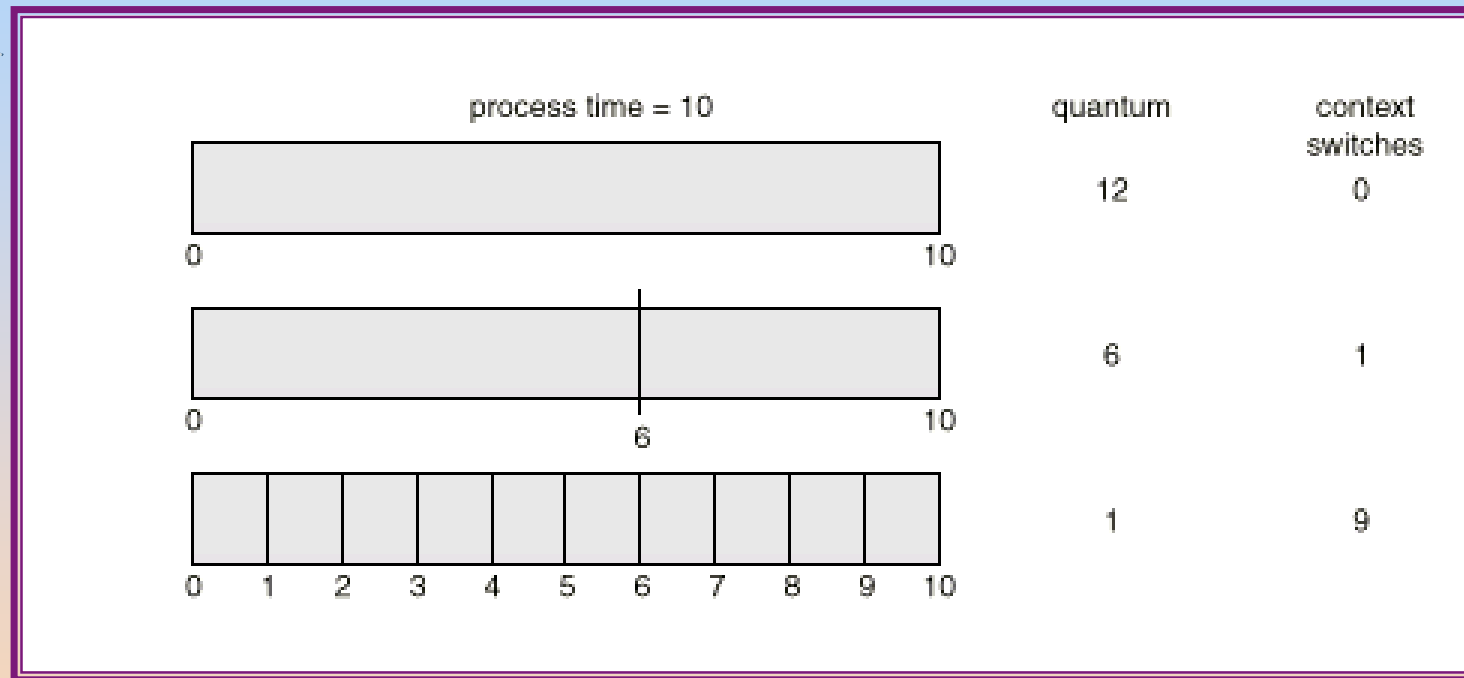


■ Typically, higher average turnaround than SJF, but better *response*.



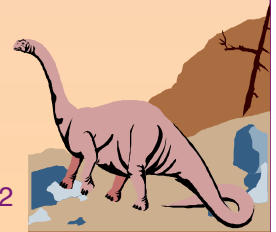
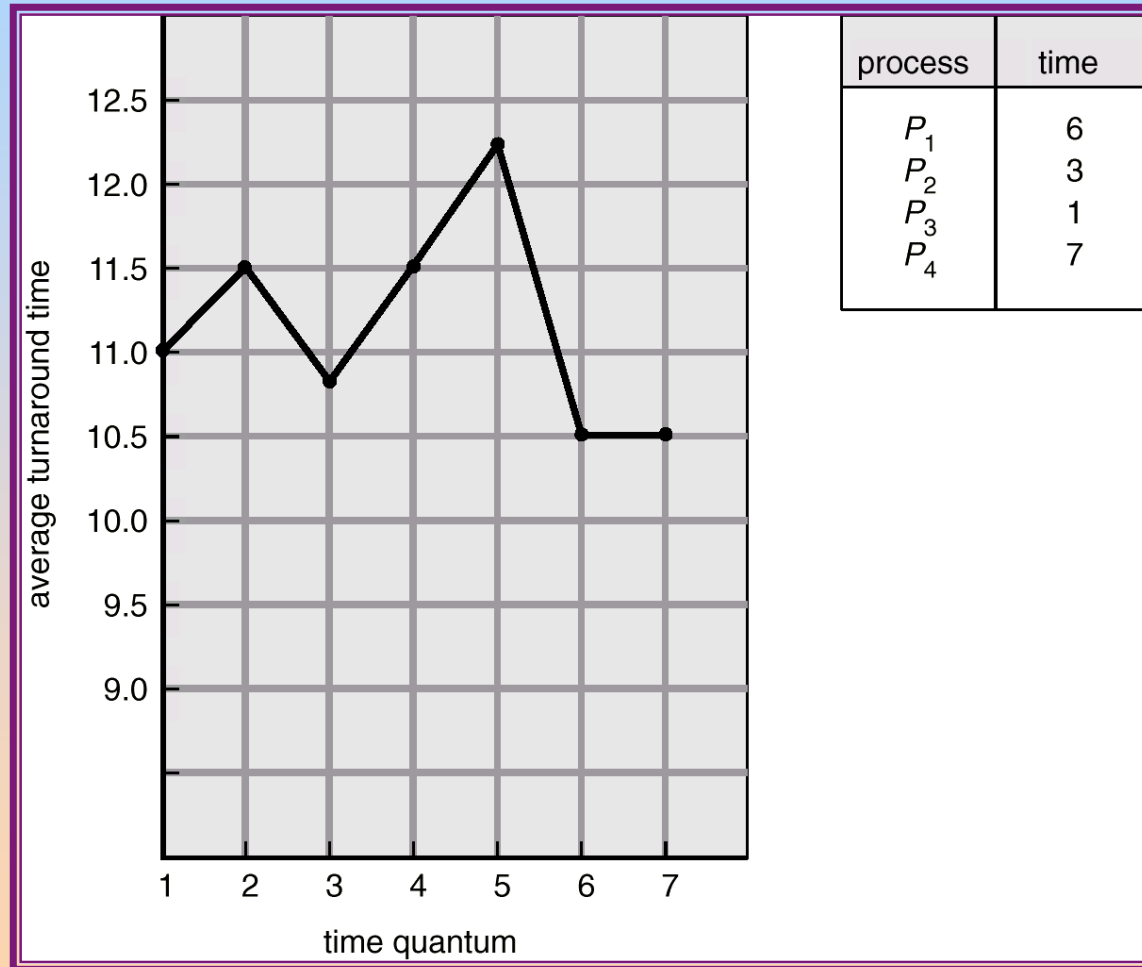


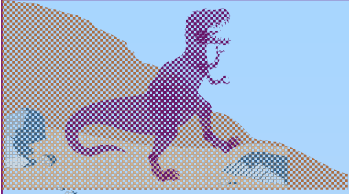
Time Quantum and Context Switch Time





Turnaround Time Varies With The Time Quantum





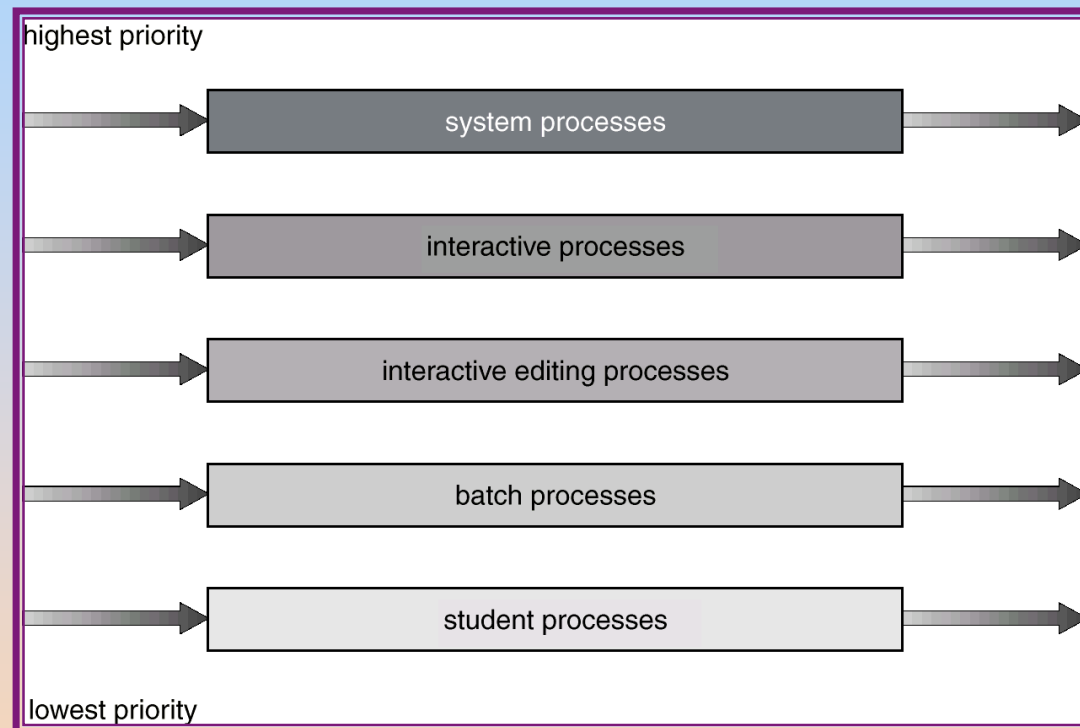
Multilevel Queue

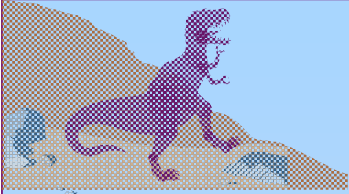
- Ready queue is partitioned into separate queues:
foreground (interactive)
background (batch)
- Each queue has its own scheduling algorithm,
foreground – RR
background – FCFS
- Scheduling must be done between the queues.
 - ✦ Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
 - ✦ Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
 - ✦ 20% to background in FCFS





Multilevel Queue Scheduling





Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way.
- Multilevel-feedback-queue scheduler defined by the following parameters:
 - ◆ number of queues
 - ◆ scheduling algorithms for each queue
 - ◆ method used to determine when to upgrade a process
 - ◆ method used to determine when to demote a process
 - ◆ method used to determine which queue a process will enter when that process needs service





Example of Multilevel Feedback Queue

■ Three queues:

- ◆ Q_0 – time quantum 8 milliseconds
- ◆ Q_1 – time quantum 16 milliseconds
- ◆ Q_2 – FCFS

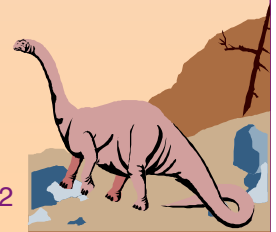
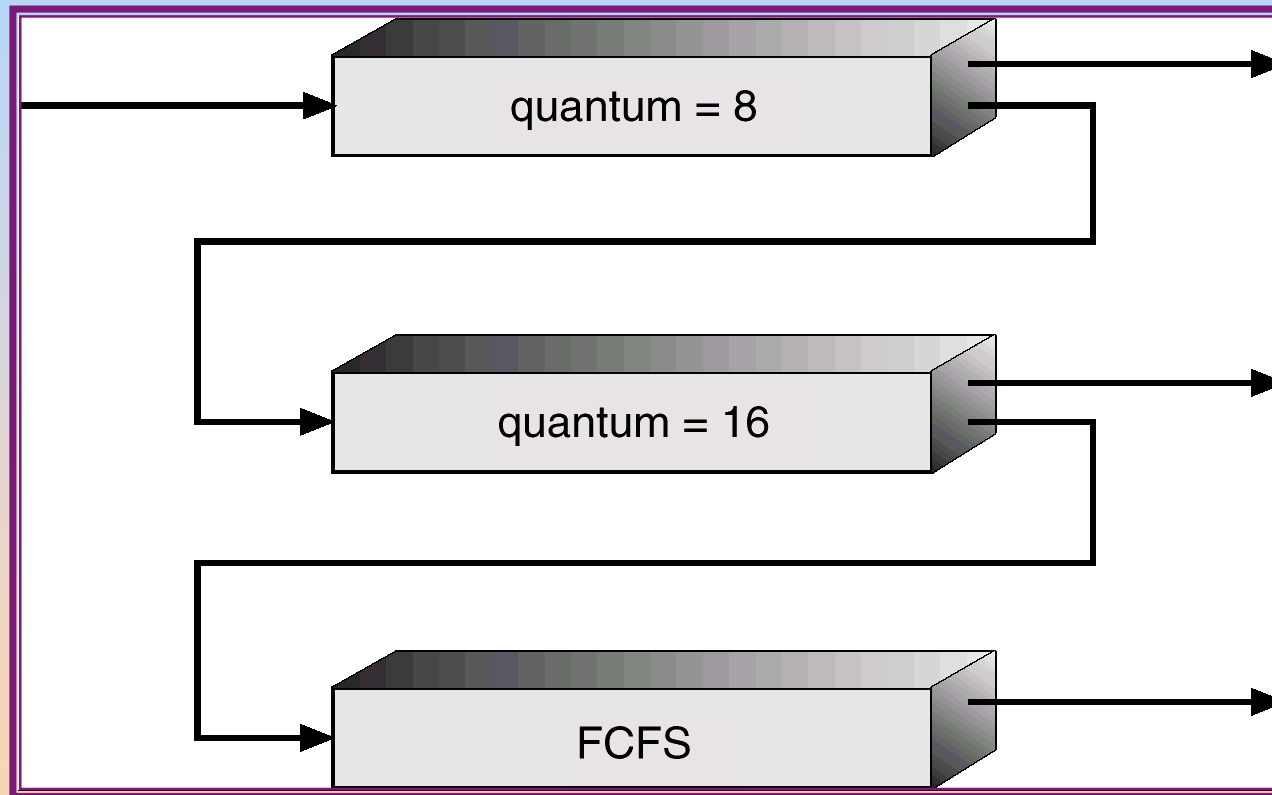
■ Scheduling

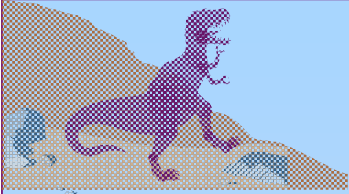
- ◆ A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1 .
- ◆ At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2 .





Multilevel Feedback Queues





Multiple-Processor Scheduling

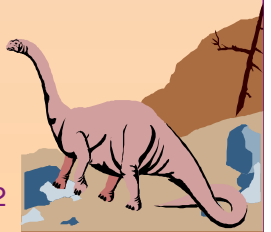
- CPU scheduling more complex when multiple CPUs are available.
- *Homogeneous processors* within a multiprocessor.
- *Load sharing*
- *Asymmetric multiprocessing* – only one processor accesses the system data structures, alleviating the need for data sharing.



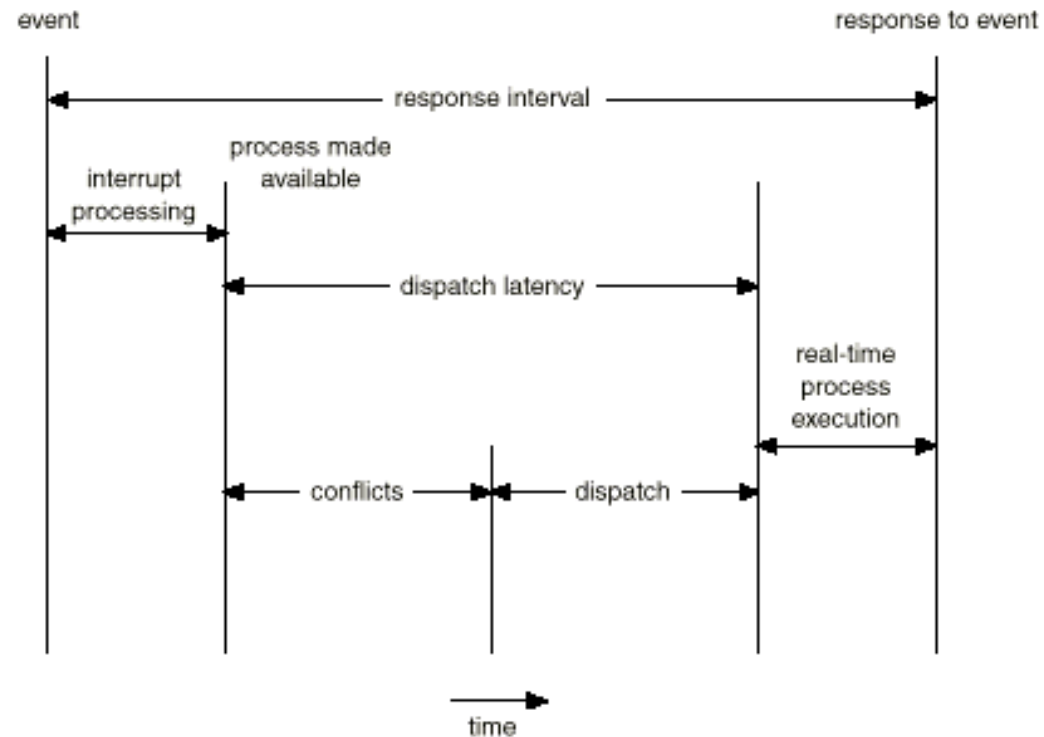


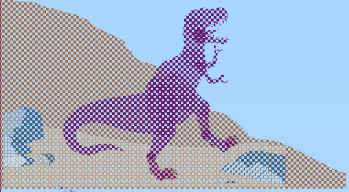
Real-Time Scheduling

- *Hard real-time* systems – required to complete a critical task within a guaranteed amount of time.
- *Soft real-time* computing – requires that critical processes receive priority over less fortunate ones.



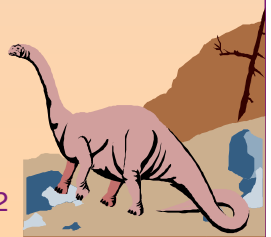
Dispatch Latency



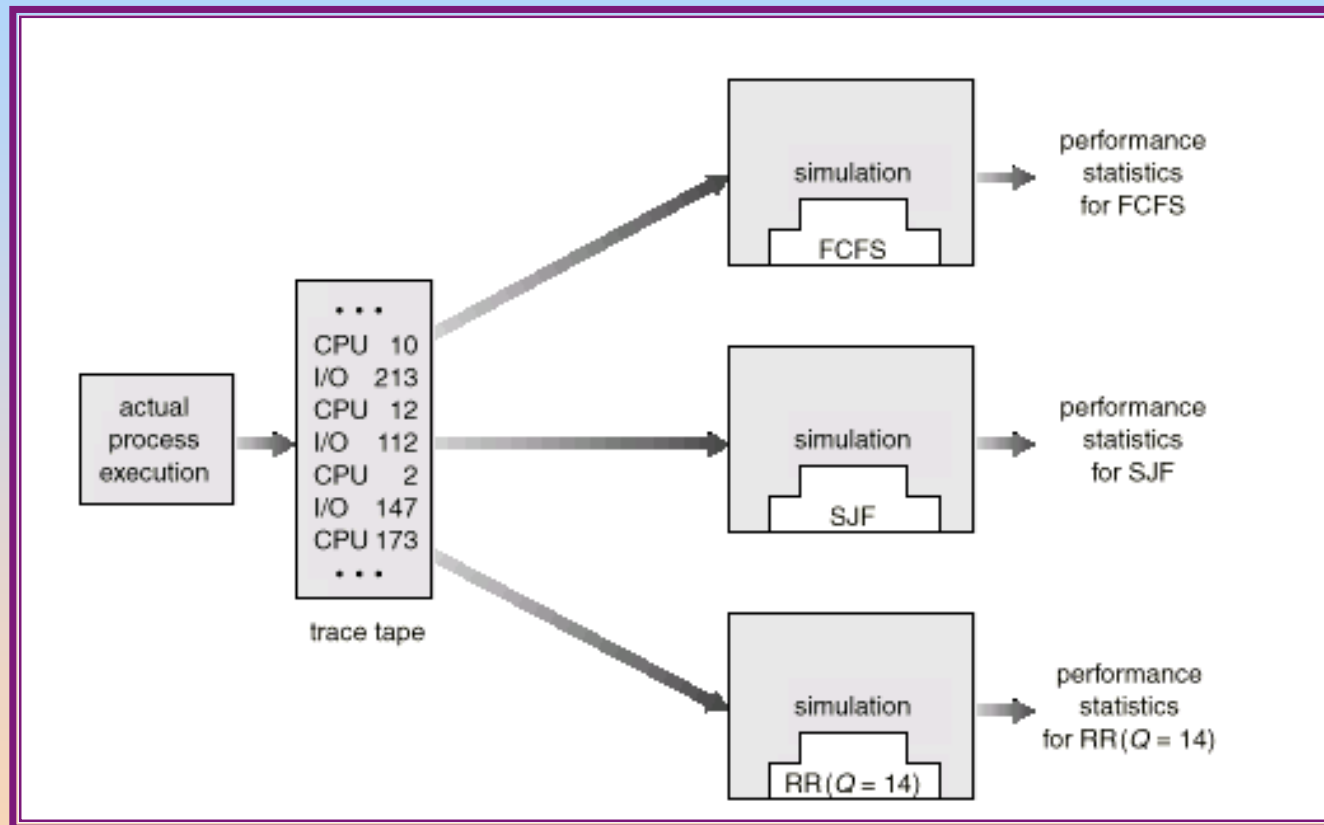


Algorithm Evaluation

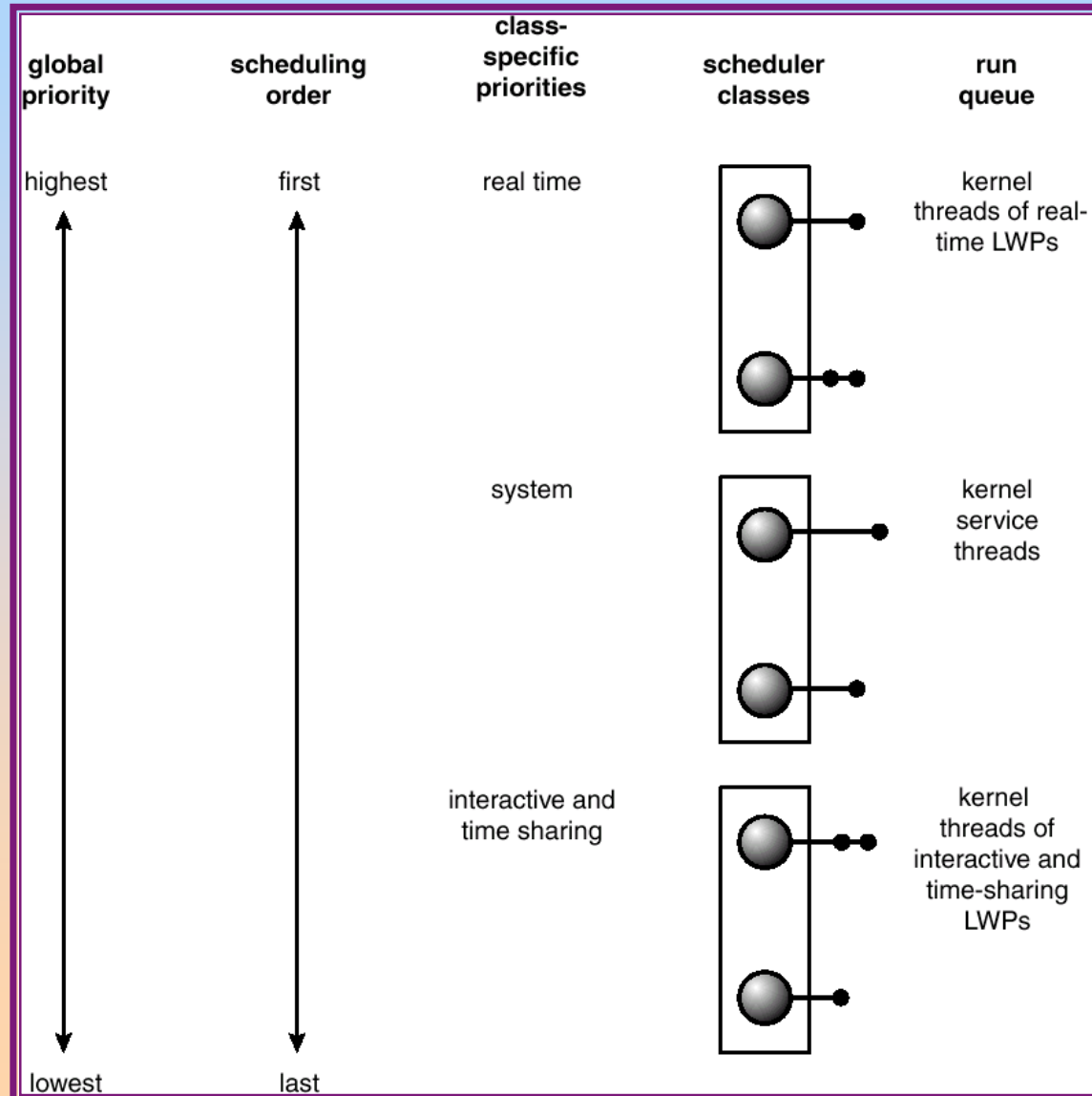
- Deterministic modeling – takes a particular predetermined workload and defines the performance of each algorithm for that workload.
- Queueing models
- Implementation

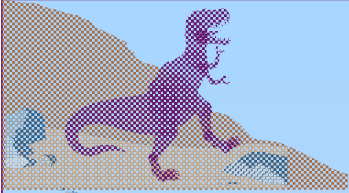


Evaluation of CPU Schedulers by Simulation



Solaris 2 Scheduling





Windows 2000 Priorities

| | real-time | high | above normal | normal | below normal | idle priority |
|---------------|-----------|------|--------------|--------|--------------|---------------|
| time-critical | 31 | 15 | 15 | 15 | 15 | 15 |
| highest | 26 | 15 | 12 | 10 | 8 | 6 |
| above normal | 25 | 14 | 11 | 9 | 7 | 5 |
| normal | 24 | 13 | 10 | 8 | 6 | 4 |
| below normal | 23 | 12 | 9 | 7 | 5 | 3 |
| lowest | 22 | 11 | 8 | 6 | 4 | 2 |
| idle | 16 | 1 | 1 | 1 | 1 | 1 |

