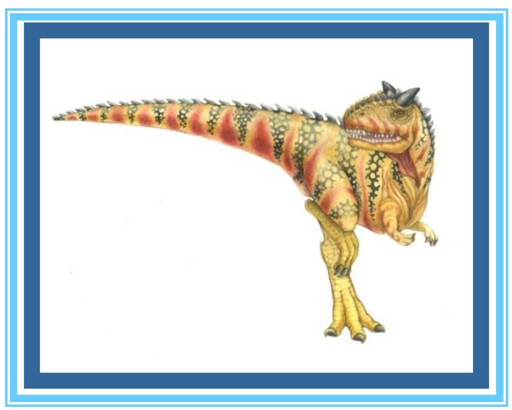
Processes Scheduling



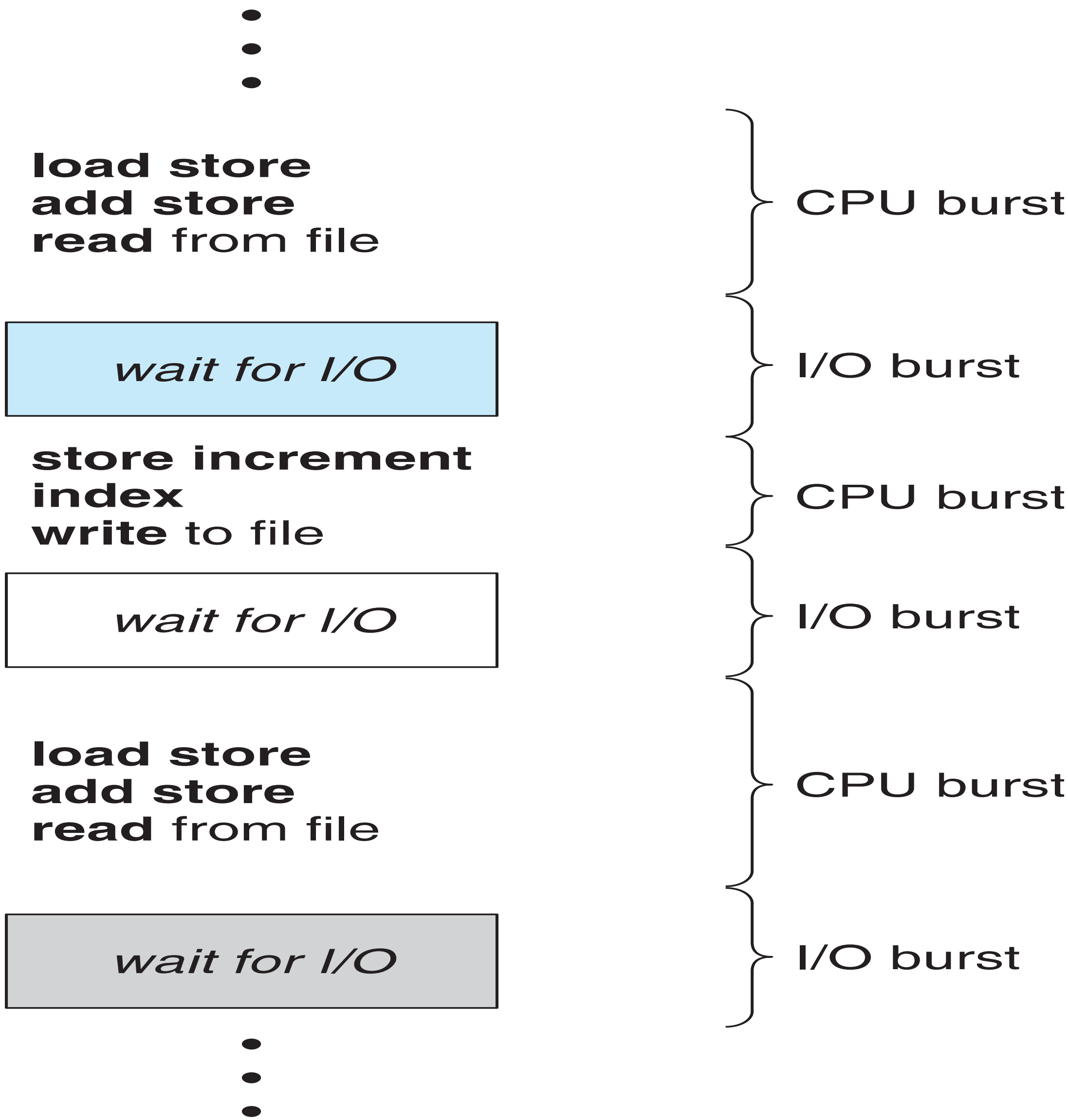
Process Scheduling

- Process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
- Process scheduling is an essential part of a Multiprogramming operating systems.
- Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

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.
- Process execution begins with CPU burst
- That is followed by I/O burst, which is followed by another CPU burst then another I/O burst and so on.
- Eventually, the final CPU burst ends with a system request to terminate execution.

Basic Concepts…



Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – Number 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)

Scheduling Algorithm Optimization Criteria

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

CPU Scheduler

- **Short-term scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- 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 **non-preemptive**
- All other scheduling is **preemptive**
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

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

Preemptive Vs Non-preemptive Scheduling

- **Non-preemptive Scheduling: A** scheduling discipline is non-preemptive if, once a process has been given the CPU, the CPU cannot be taken away from that process.
- Following are some characteristics of non-preemptive scheduling:

1. In non-preemptive system, short jobs are made to wait by longer jobs but the overall treatment of all processes is fair.
2. In non-preemptive system, response times are more predictable because incoming high priority jobs can not displace waiting jobs.
3. In non-preemptive scheduling, a scheduler executes jobs in the following two situations.:
   - When a process switches from running state to the waiting state.
   - When a process terminates.

Preemptive Vs Non-preemptive Scheduling

- **Preemptive Scheduling:** A scheduling discipline is preemptive if, once a process has been given the CPU can taken away.
- The strategy of allowing processes that are logically runnable to be temporarily suspended is called Preemptive Scheduling and it is contrast to the "run to completion" method.

Some Scheduling Algorithms

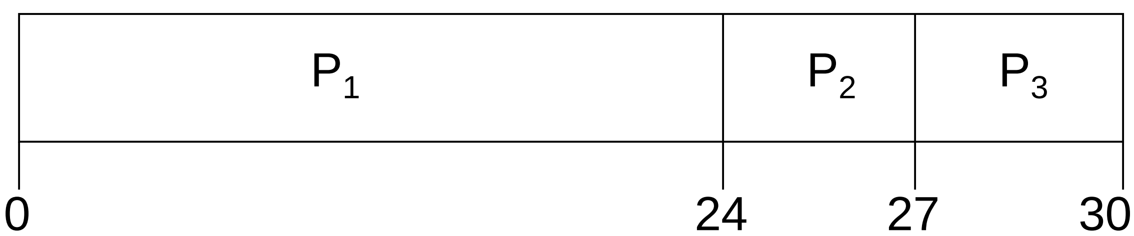
- First-Come, First-Served (FCFS)
- Shortest-Job-First (SJF) Scheduling
- Priority Scheduling
- Round Robin (RR)
- Multiple-Level Queues Scheduling

FCFS

- Example:

Process	Burst Time
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

Continue…

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.

Performance of FCFS Scheduling

- FCFS scheduling is non-preemptive.
- Convoy effect: all the processes waits for the one big process to get off the CPU.
- The FCFS algorithm is particularly troublesome for time-sharing system, where it is important that each user get a share of CPU at regular interval

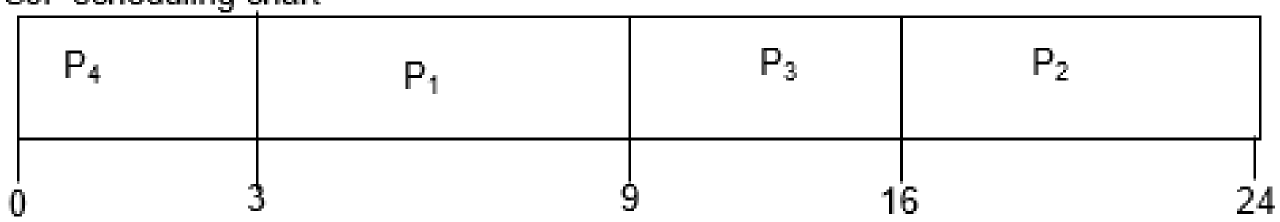
SJF Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - Non-preemptive – 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 know as the Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes.

SJF without arrival Time

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart

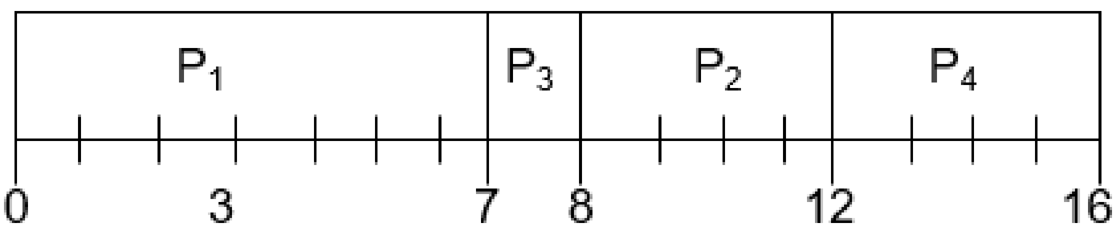


- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

SJF with arrival Time (non-preemptive)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

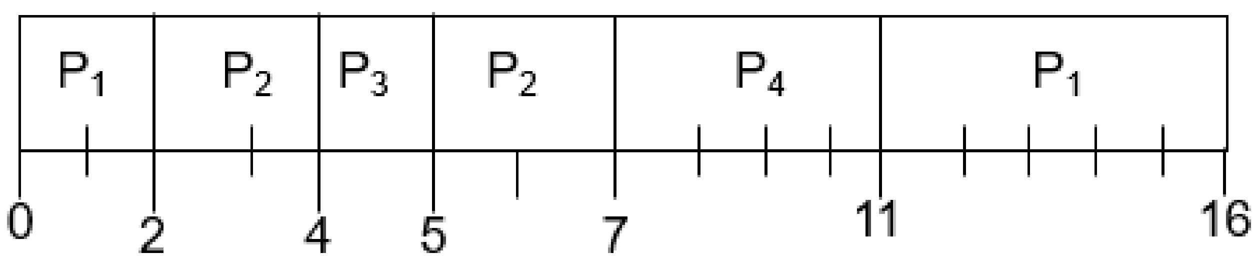


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

SJF with arrival Time (preemptive)

Process	Arrival Time	Burst Time
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

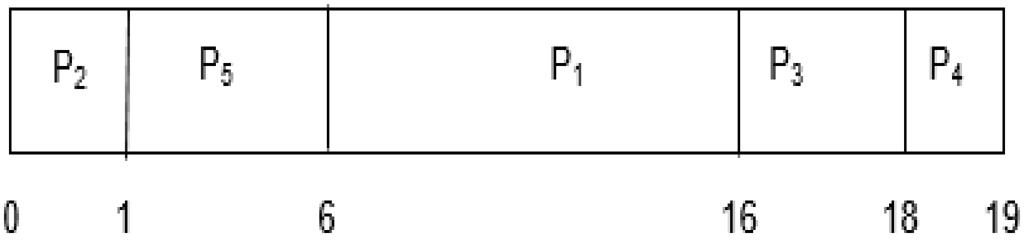
Priority Scheduling

- 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
  - Non-preemptive
- SJF is priority scheduling where priority is the inverse of 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.

Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

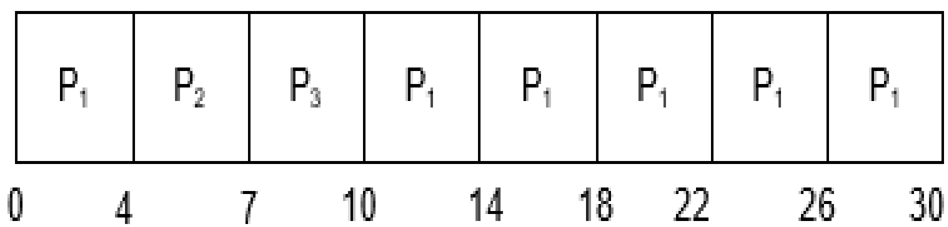
Round Robin (RR)

- Each process gets a small unit of CPU time (**time quantum** *q*), 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.
- Timer interrupts every quantum to schedule next process
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

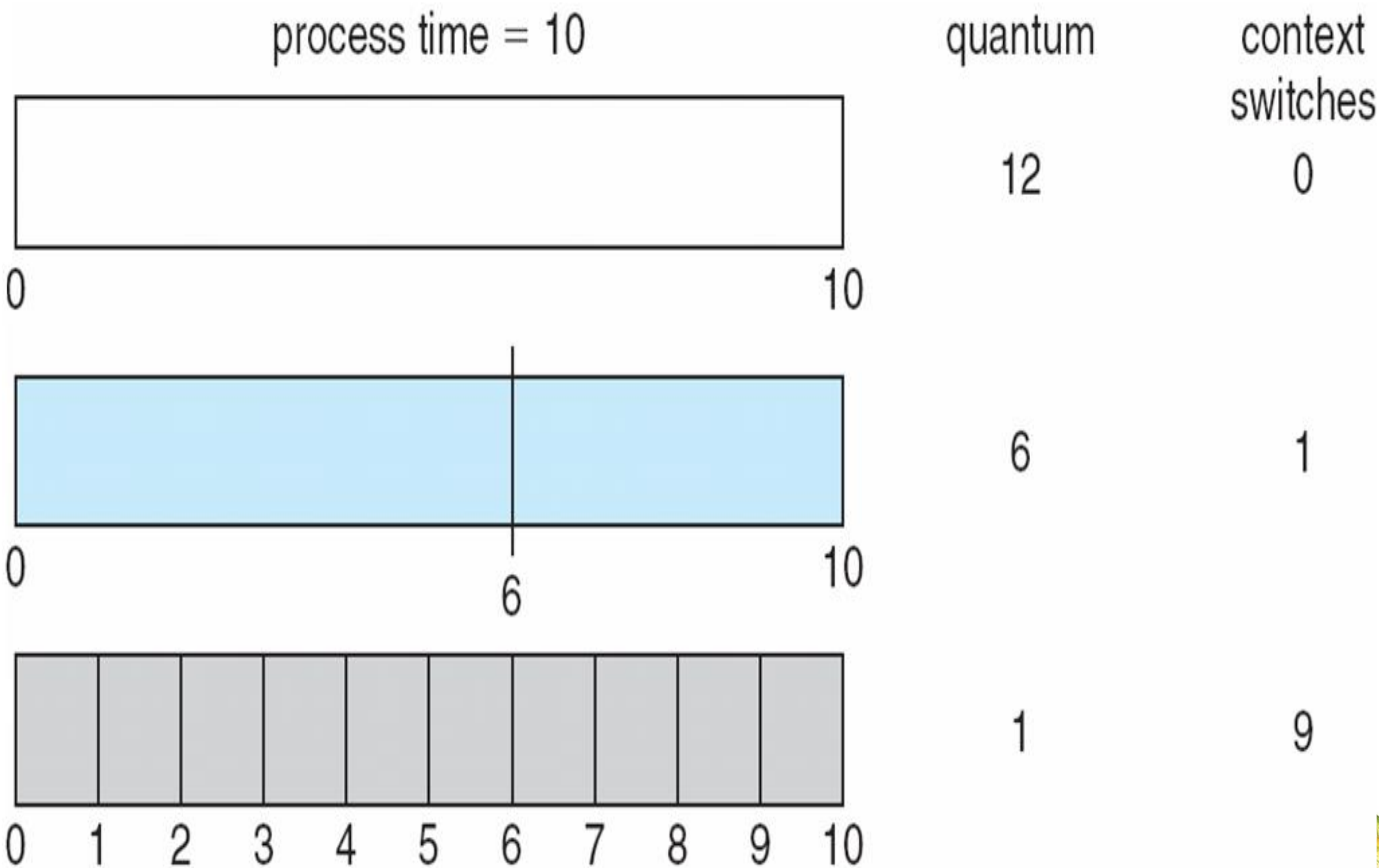
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

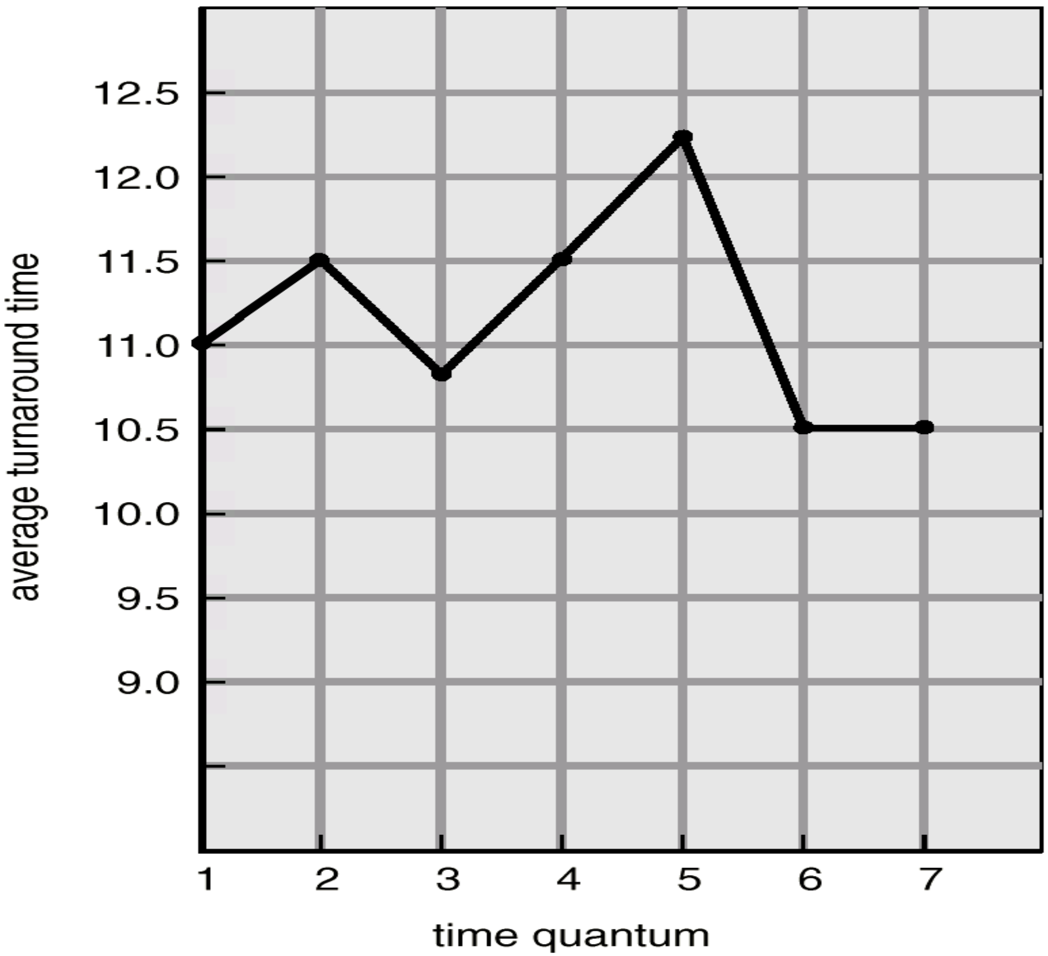


- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7