CPU Scheduling

Course Code: CSC 2209 Course Title: Operating Systems

Dept. of Computer Science
Faculty of Science and Technology

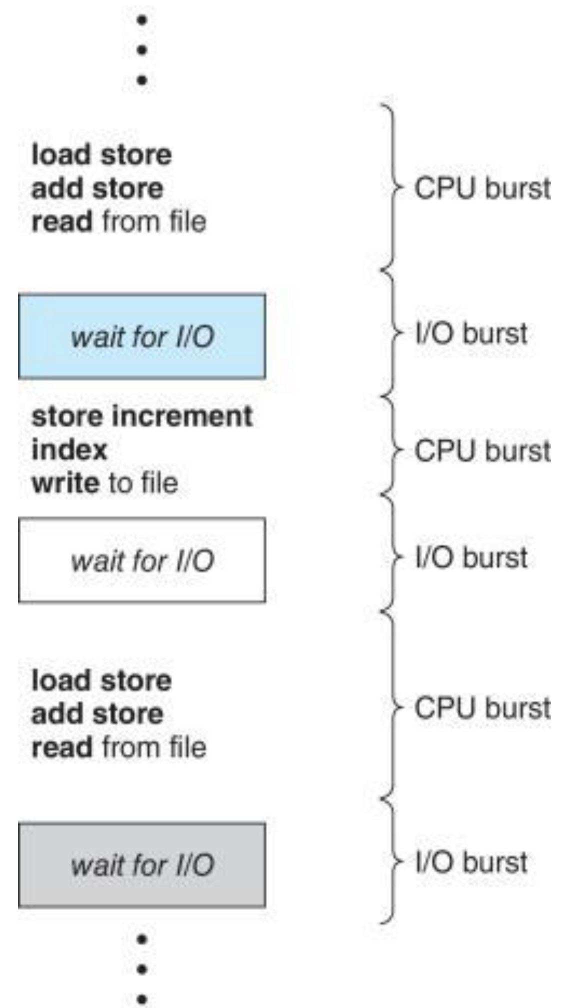
Lecturer No:	**06**	**Week No:**	**06**	**Semester:**	
Lecturer:	*Name & email*				

Lecture Outline

1. Basic Concepts
2. Scheduling Criteria
3. Scheduling Algorithms

Basic Concepts

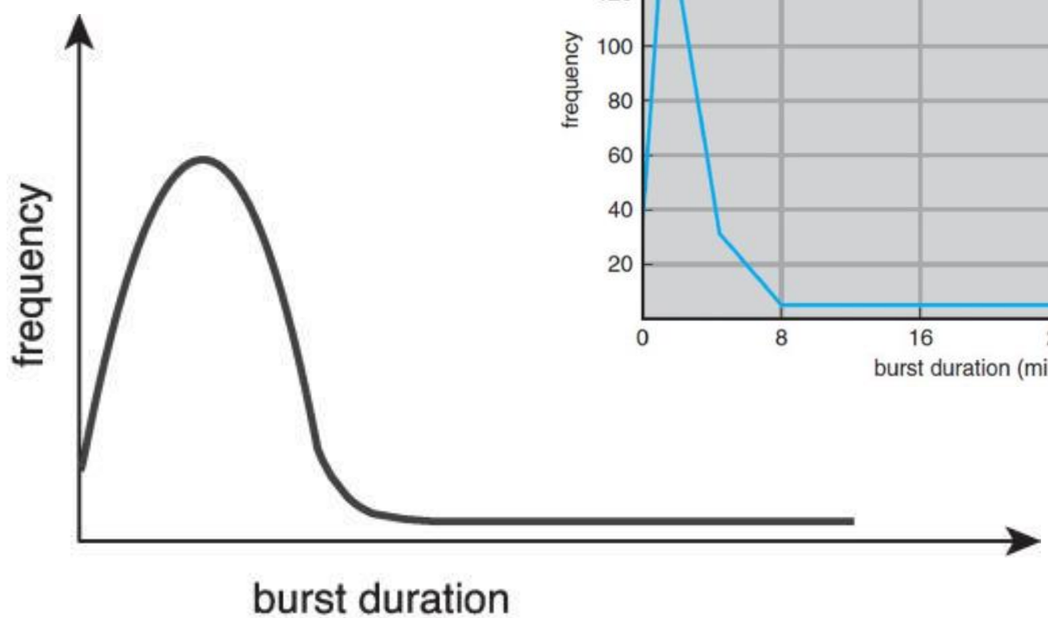
- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern

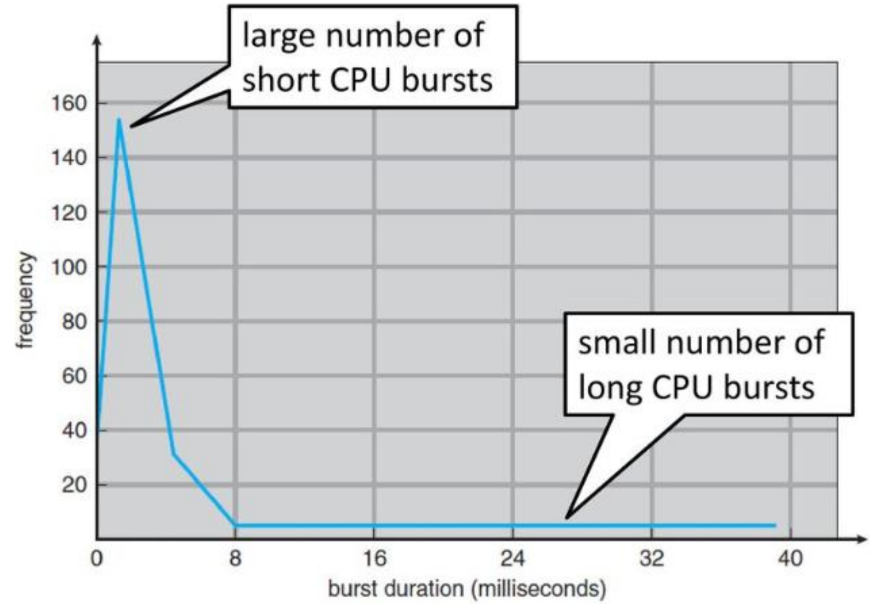


Histogram of CPU-burst Times

Large number of short bursts

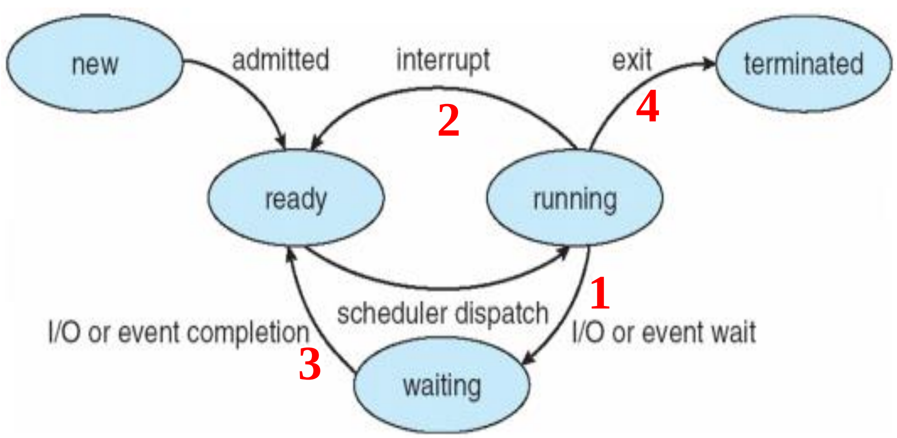
Small number of longer bursts





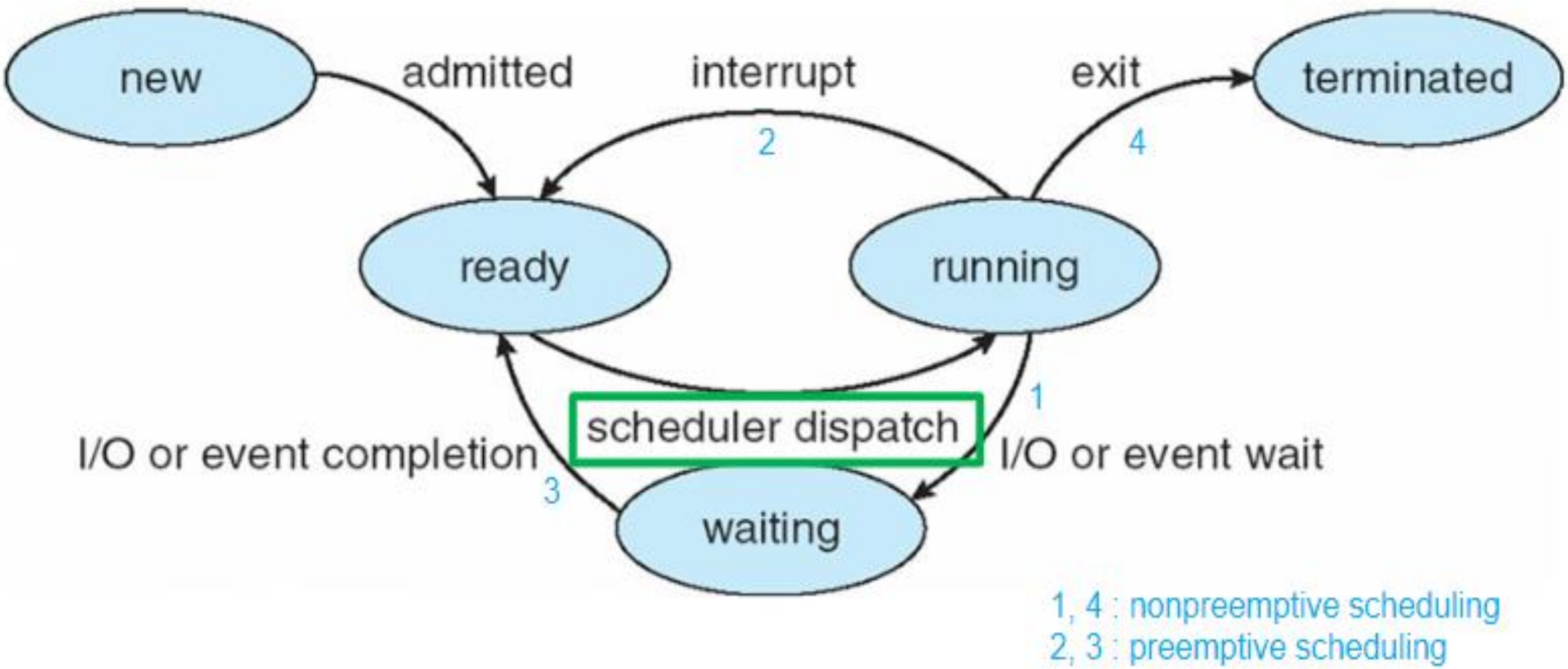
CPU Scheduler

- The **CPU scheduler** selects from among the processes in <u>ready queue</u>, 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 **nonpreemptive**
- All other scheduling is **preemptive**
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

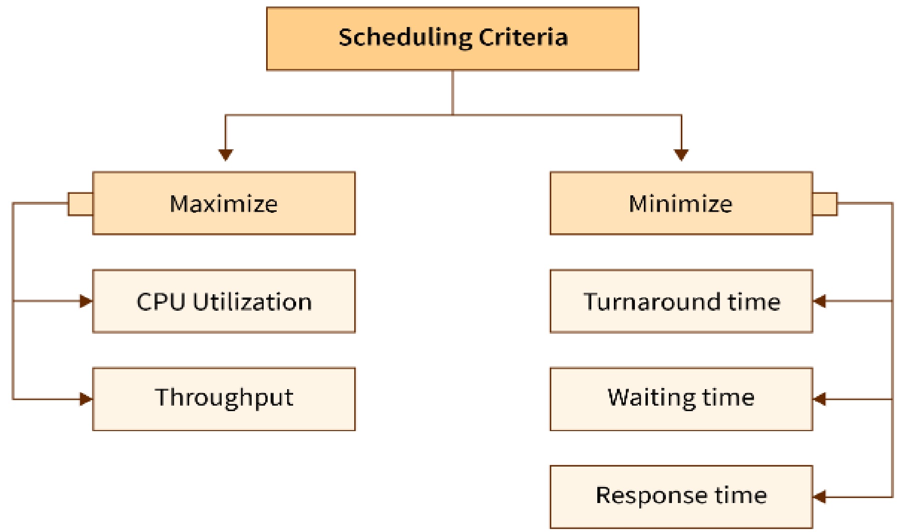
CPU Scheduler

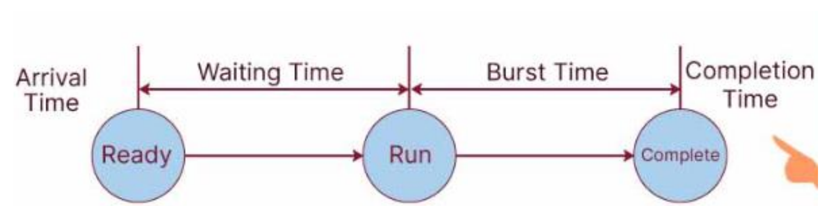


- Many processes are in ready or waiting states in multiprogrammed operating systems.

Scheduling Criteria

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



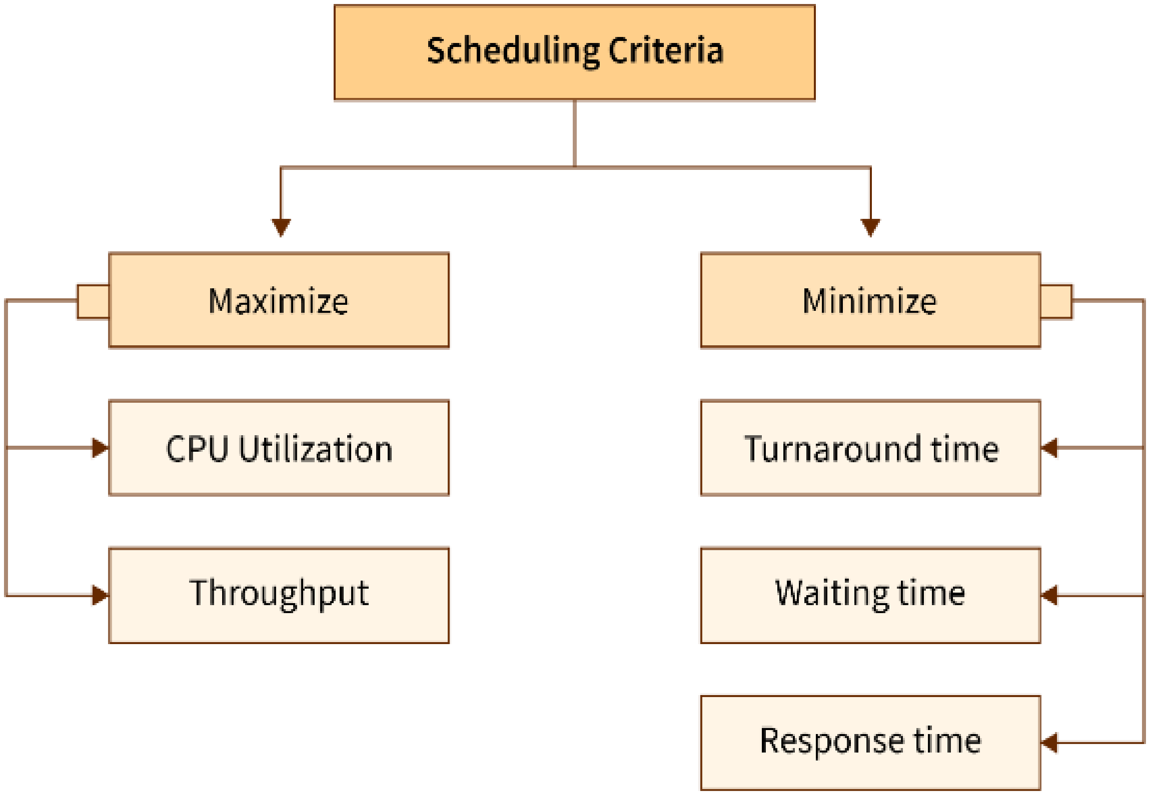


Turnaround time (TAT) = Completion time - Arrival time

Waiting time (WT) = Turnaround time - CPU burst time

Scheduling Algorithm Optimization Criteria

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



First- Come, First-Served (FCFS) Scheduling

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: $(0 + 24 + 27)/3 = 17$

FCFS Scheduling (cont'd)

Process	Brust Time
P_1	24
P_2	3
P_3	3

- 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
  - Consider one CPU-bound and many I/O-bound processes

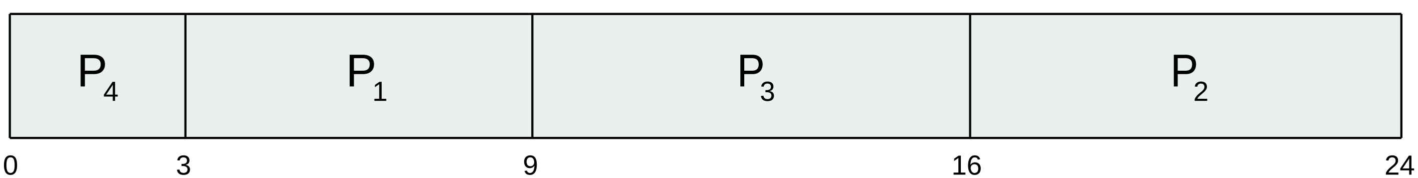
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

Example of SJF (Non preemptive)

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
- Waiting time for $P_1 = 3$; $P_2 = 16$; $P_3 = 9$; $P_4 = 0$

Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  -> Then pick process with shortest predicted next CPU burst

- 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. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$ = weight of recent and past history
4. Define : $\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n$

t_n : contains the most recent information

τ_n : stores the past history

- Commonly, α set to ½

- Preemptive version called **shortest-remaining-time-first**

Determining Length of Next CPU Burst

Example:

Suppose we have the following sequence of recent CPU burst times for a process:

$t_1 = 10\,\text{ms},\ t_2 = 15\,\text{ms},\ t_3 = 8\,\text{ms},\ t_4 = 12\,\text{ms}$

Assume $\alpha = 0.5$ (smoothing factor) and $\tau_0 = 10$ ms (initial estimate).

$$\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n$$

Calculation:

$\tau_1 = 0.5 \cdot 10 + 0.5 \cdot 10 = 10\,\text{ms}$

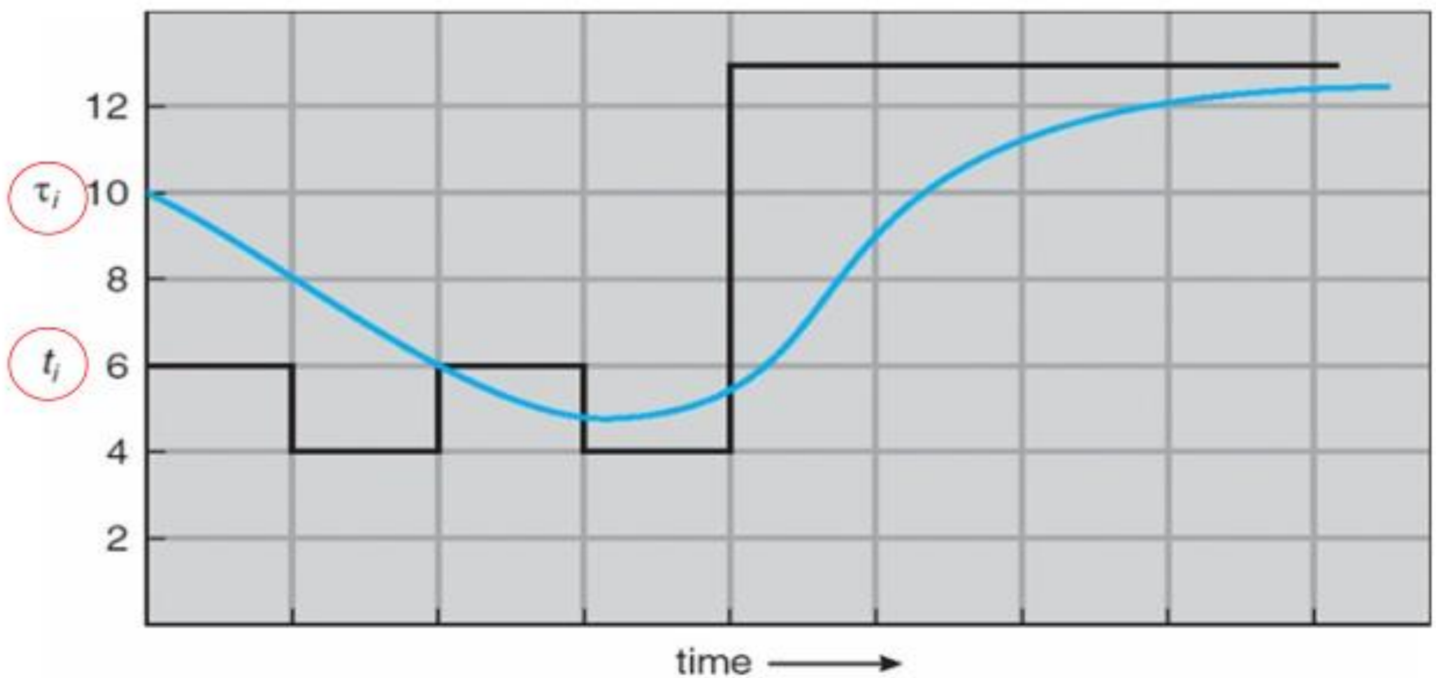
$\tau_2 = 0.5 \cdot 15 + 0.5 \cdot 10 = 12.5\,\text{ms}$

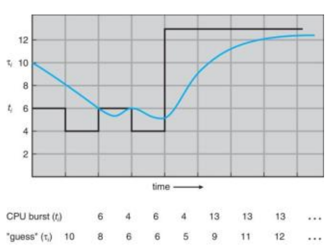
$\tau_3 = 0.5 \cdot 8 + 0.5 \cdot 12.5 = 10.25\,\text{ms}$

$\tau_4 = 0.5 \cdot 12 + 0.5 \cdot 10.25 = 11.125\,\text{ms}$

Therefore, the prediction for the next CPU burst length τ_5 would be approximately 11.125 ms using this method.

Prediction of the Length of the Next CPU Burst





CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

α set to ½

8=½*6 + ½*10

6=½*4 + ½*8

Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

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

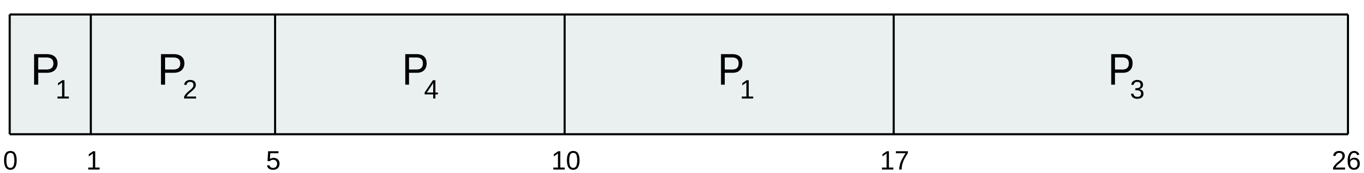
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



- Average waiting time = [(10-1-0)+(1-0-1)+(17-0-2)+(5-0-3)]/4 = 26/4 = 6.5 ms
- Waiting time = Total/Actual WT – previous total exec time – AT (Formula)

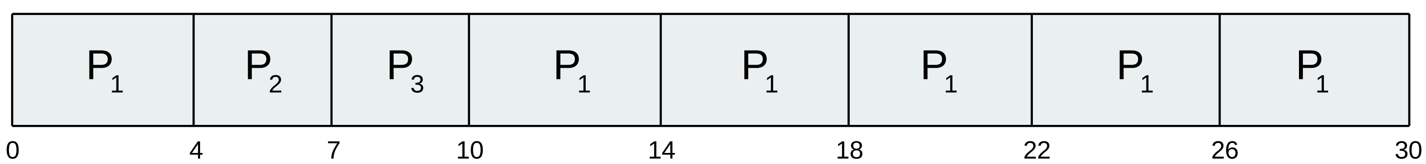
Round Robin (RR)

- 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

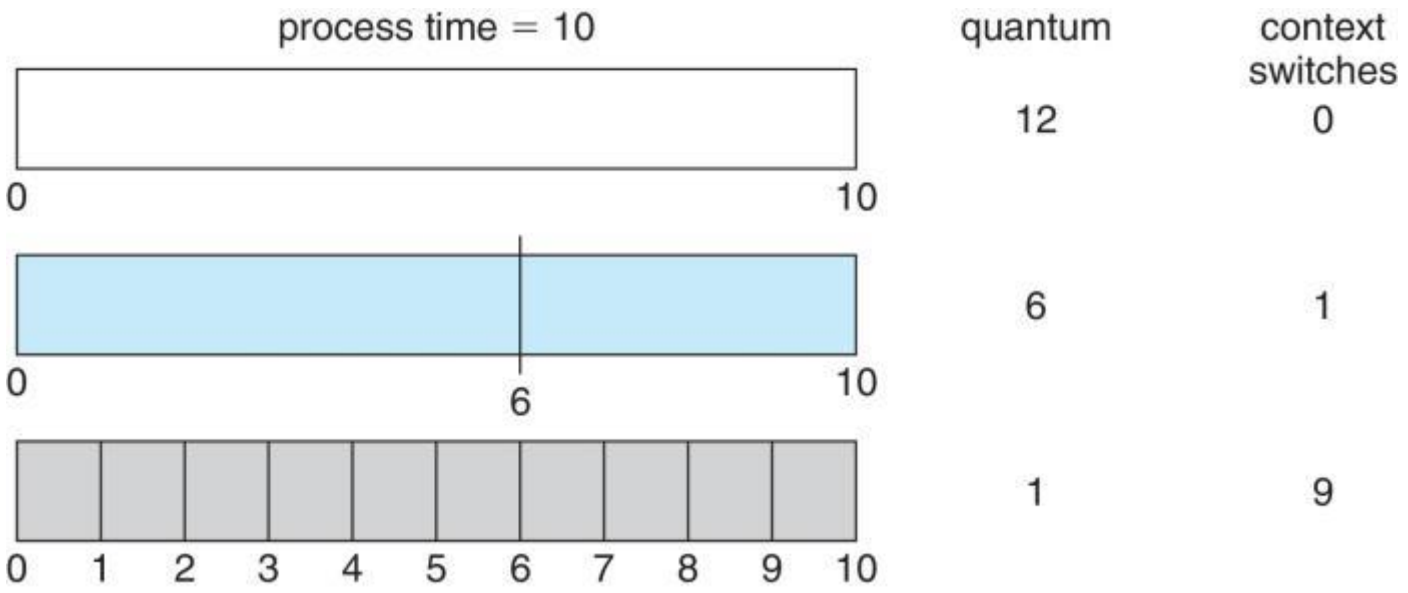
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

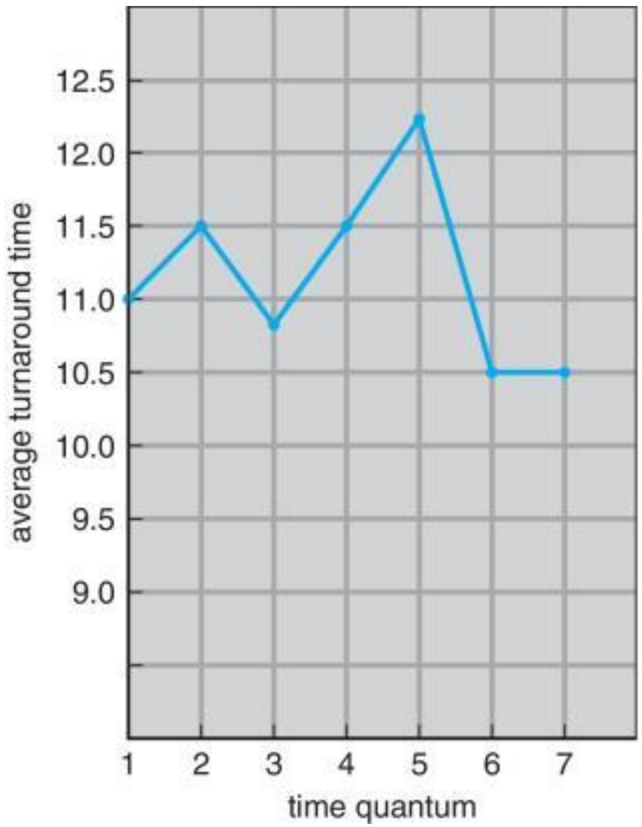


- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

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

80% of CPU bursts
should be shorter than q

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Non-preemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **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

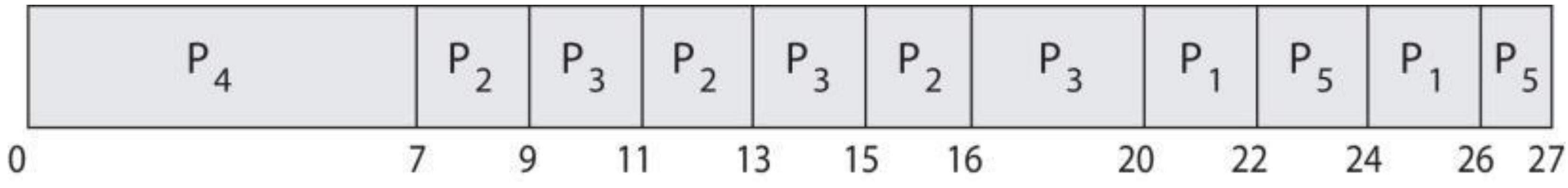
- Priority scheduling Gantt Chart

- Average waiting time = 8.2 msec

Priority Scheduling w/ Round-Robin

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

- Run the process with the highest priority. Processes with the same priority run round-robin
- Gantt Chart with 2 ms time quantum



Books

- Operating Systems Concept
  - Written by Galvin and Silberschatz
  - Edition: 9^{th}

References

- Operating Systems Concept
  - Written by Galvin and Silberschatz
  - Edition: 9th