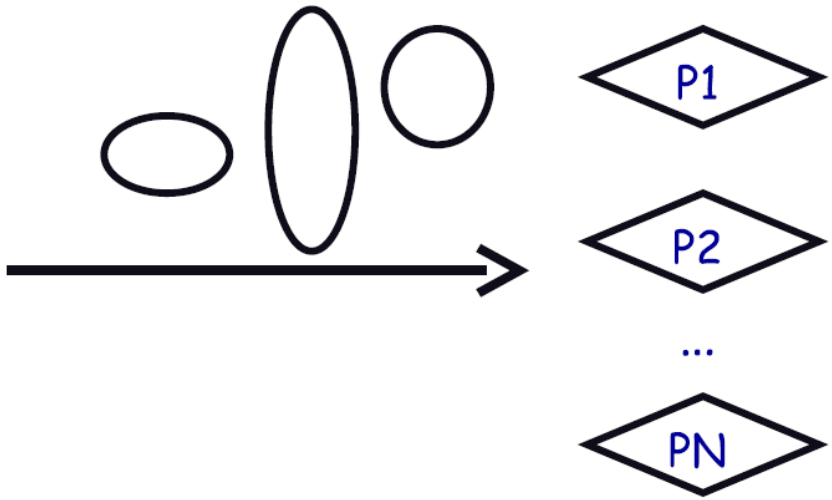
Operating Systems

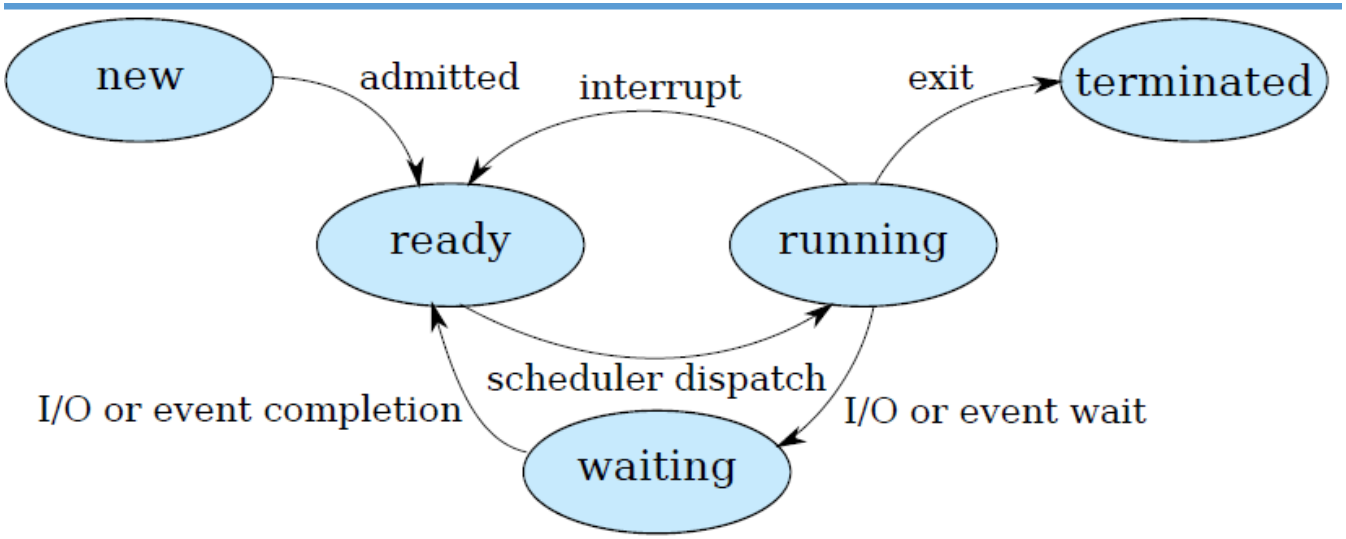
Scheduling: Introduction

CPU Scheduling

- The scheduling problem:
  - Have K jobs ready to run
  - Have N ≥ 1 CPUs
  - Which jobs to assign to which CPU(s)
- When do we make decision?



CPU Scheduling



- 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 new/waiting to ready
  4. Switches from ready to running
  5. Exits
- Non-preemptive schedules use 1 & 4 only
- Preemptive schedulers run at all five points

Scheduling Metrics

- Performance metric: Turnaround time
  - The time at which **the job completes** minus the time at which **the job arrived** in the system.

$$T_{turnaround} = T_{completion} - T_{arrival}$$

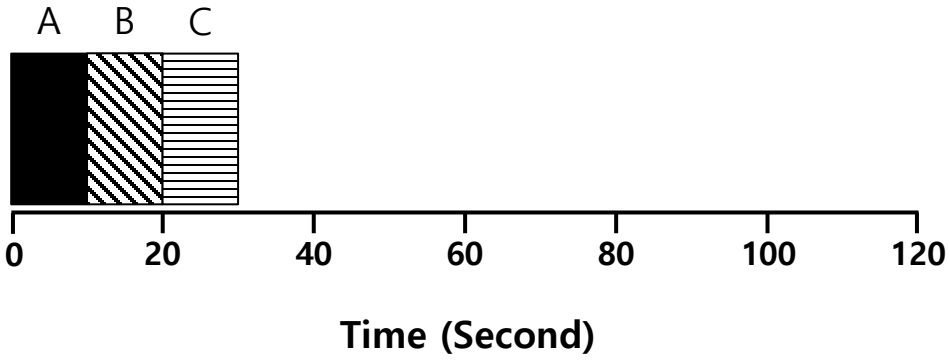
- Throughput – # of process that complete per unit time
- Response time – time from request to first response
  - (e.g., key press to character echo, not launch to exit)
- Above criteria are affected by secondary criteria
  - CPU utilization – fraction of time CPU doing productive work
  - Waiting time – time each proc waits in ready queue

Scheduling: Workload assumptions:

- Workload assumptions:
  1. Each job runs for the **same amount of time.**
  2. All jobs **arrive** at the same time.
  3. All jobs only use the **CPU** (i.e., they perform no I/O).
  4. The **run-time** of each job is known.

First In, First Out (FIFO)

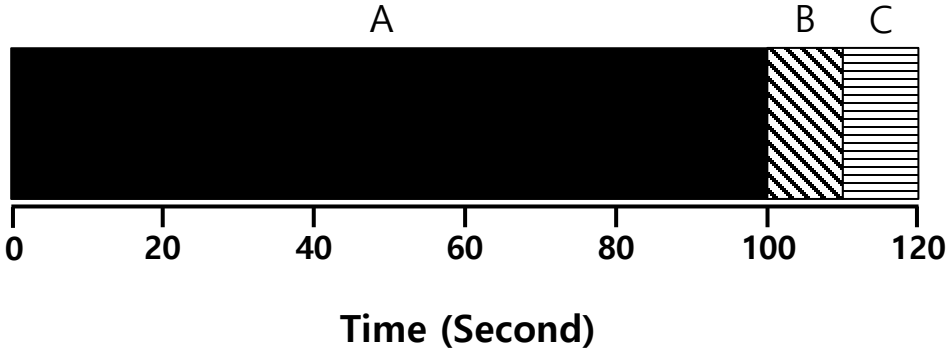
- First Come, First Served (FCFS)
  - Very simple and easy to implement
- Example:
  - A arrived just before B which arrived just before C (roughly at the same time).
  - Each job runs for 10 seconds.



$$Average\ turnaround\ time = \frac{10 + 20 + 30}{3} = 20\ sec$$

Why FIFO is not that great? – Convoy effect

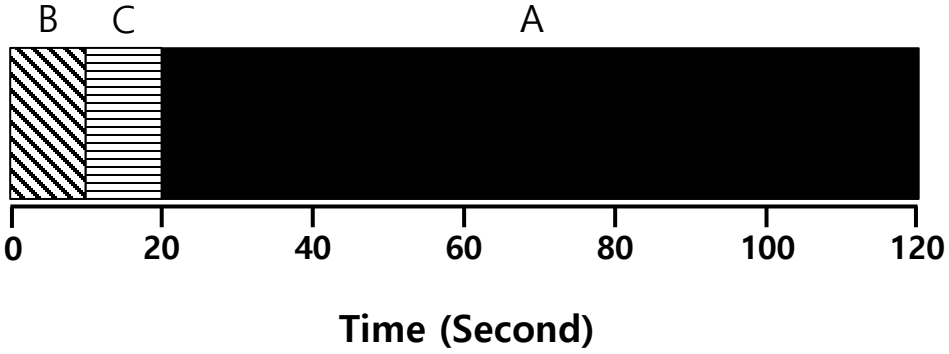
- Let's relax assumption 1: Each job **no longer** runs for the same amount of time.
- Example:
  - A arrived just before B which arrived just before C (roughly at the same time).
  - A runs for 100 seconds, B and C run for 10 each.



$$Average\ turnaround\ time = \frac{100 + 110 + 120}{3} = 110\ sec$$

Shortest Job First (SJF)

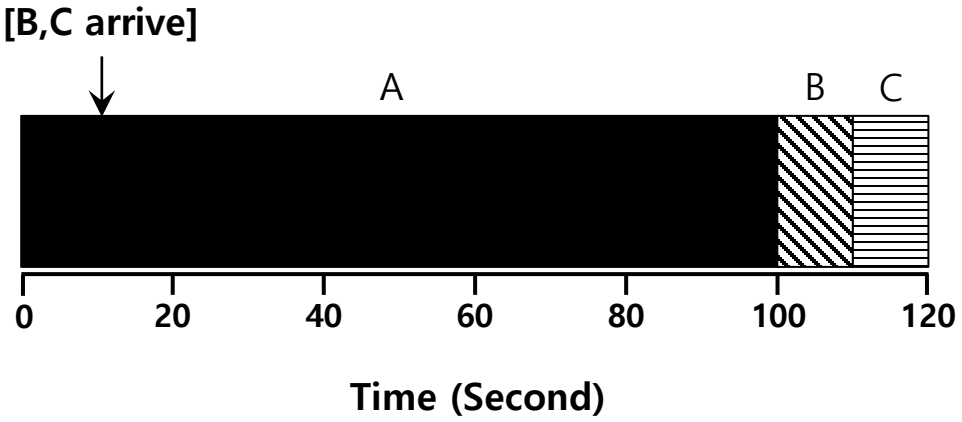
- Run the shortest job first, then the next shortest, and so on
  - Non-preemptive scheduler
- Example:
  - A arrived just before B which arrived just before C (roughly at the same time).
  - A runs for 100 seconds, B and C run for 10 each.



$$Average\ turnaround\ time = \frac{10 + 20 + 120}{3} = 50\ sec$$

SJF with Late Arrivals from B and C

- Let's relax assumption 2: Jobs can arrive at any time.
- Example:
  - A arrives at t=0 and needs to run for 100 seconds.
  - B and C arrive at t=10 and each need to run for 10 seconds



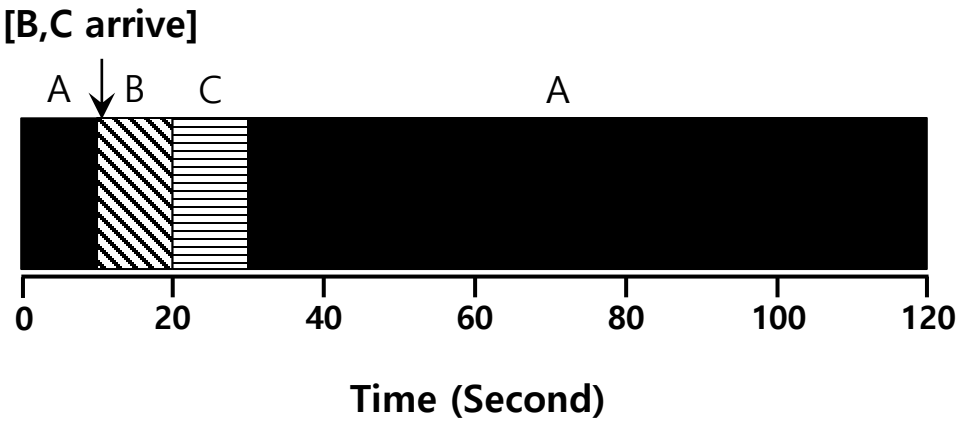
$$Average\ turnaround\ time = \frac{100 + (110 - 10) + (120 - 10)}{3} = 103.33\ sec$$

Shortest Time-to-Completion First (STCF)

- Add preemption to SJF
  - Also knows as Preemptive Shortest Job First (PSJF)
- A new job enters the system:
  - Determine of the remaining jobs and new job
  - Schedule the job which has the least time left

Shortest Time-to-Completion First (STCF)

- Example:
  - A arrives at t=0 and needs to run for 100 seconds.
  - B and C arrive at t=10 and each need to run for 10 seconds

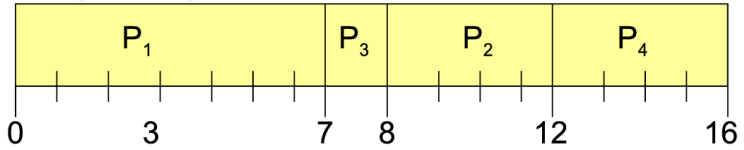


$$\textbf{Average turnaround time} = \frac{(120-0)+(20-10)+(30-10)}{3} = 50\ sec$$

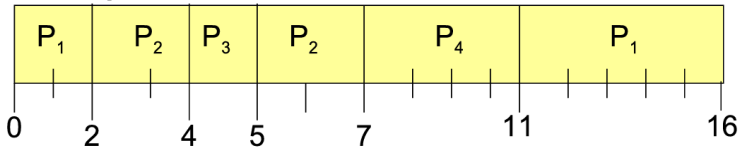
Example

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

- Non-preemptive



- Preemptive



SJF limitations

- Doesn't always minimize average turnaround time
  - Only minimizes waiting time, which minimizes response time
  - Example where turnaround time might be suboptimal?
- Can lead to unfairness or starvation
- In practice, can't actually predict the future
- But can estimate CPU burst length based on past

Response time

- The time from **when the job arrives** to the **first time it is scheduled**.

$$T_{response} = T_{firstrun} - T_{arrival}$$

  - STCF and related disciplines are not particularly good for response time.

How can we build a scheduler that is sensitive to response time?

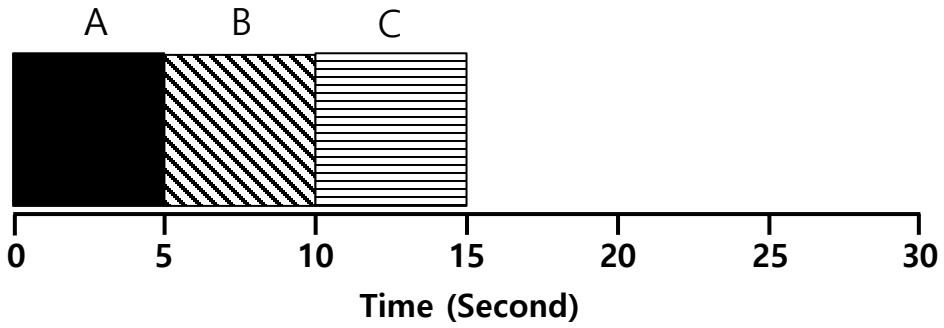
Round Robin (RR) Scheduling

- Time slicing Scheduling
  - Run a job for a time slice and then switch to the next job in the **run queue** until the jobs are finished.
    - Time slice is sometimes called a scheduling quantum.
  - It repeatedly does so until the jobs are finished.
  - The length of a time slice must be *a multiple of* the timer-interrupt period.

RR is fair, but performs poorly on metrics such as turnaround time

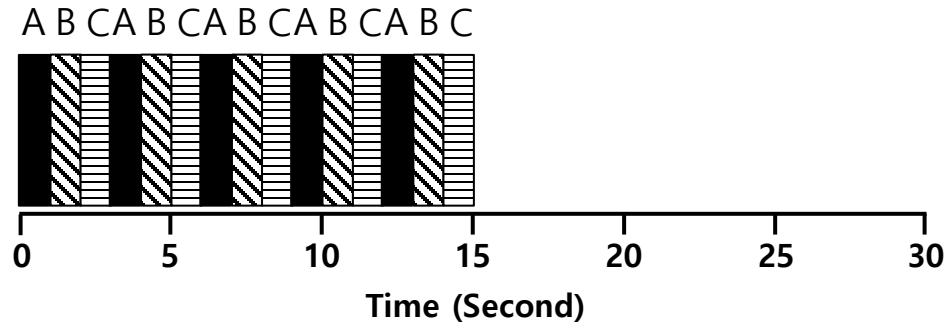
RR Scheduling Example

- A, B and C arrive at the same time.
- They each wish to run for 5 seconds.



SJF (Bad for Response Time)

$$T_{average\ response} = \frac{0 + 5 + 10}{3} = 5sec$$



RR with a time-slice of 1sec (Good for Response Time)

$$T_{average\ response} = \frac{0 + 1 + 2}{3} = 1sec$$

RR disadvantages

- Varying sized jobs are good …what about same-sized jobs?
- Assume 2 jobs of time=100 each:

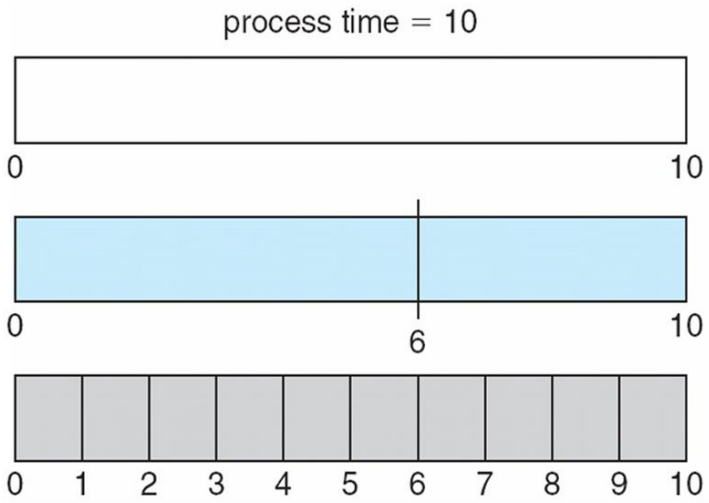
- Even if context switches were free…
  - What would average completion time be with RR? ***199.5***
  - How does that compare to FCFS? ***150***

The length of the time slice is critical.

- The shorter time slice
  - Better response time
  - The cost of context switching will dominate overall performance.
- The longer time slice
  - Amortize the cost of switching
  - Worse response time

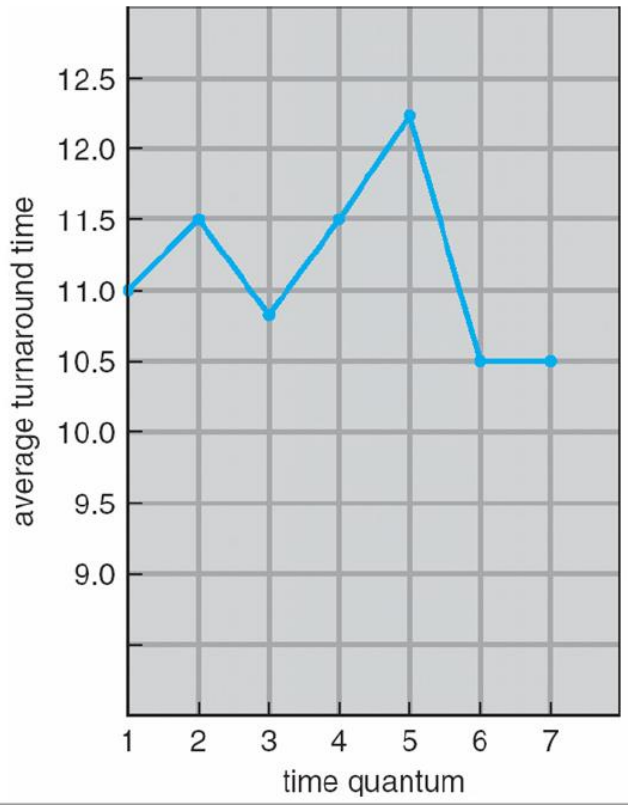
Deciding on the length of the time slice presents a trade-off to a system designer

The length of the time slice is critical.



- How to pick quantum?
  - Want much larger than context switch cost
  - Majority of bursts should be less than quantum
  - But not so large system reverts to FCFS
- Typical values: 10–100 msec

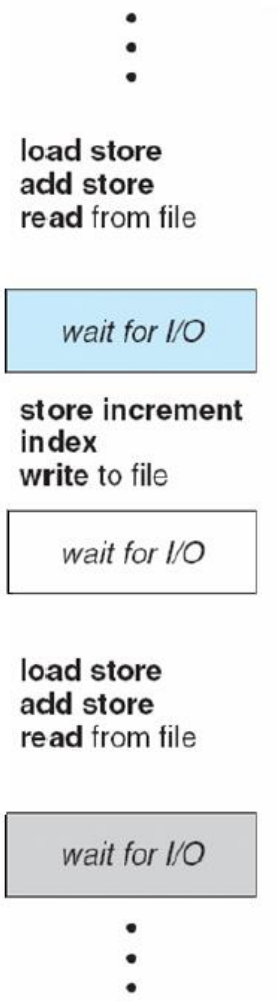
Turnaround time vs. quantum



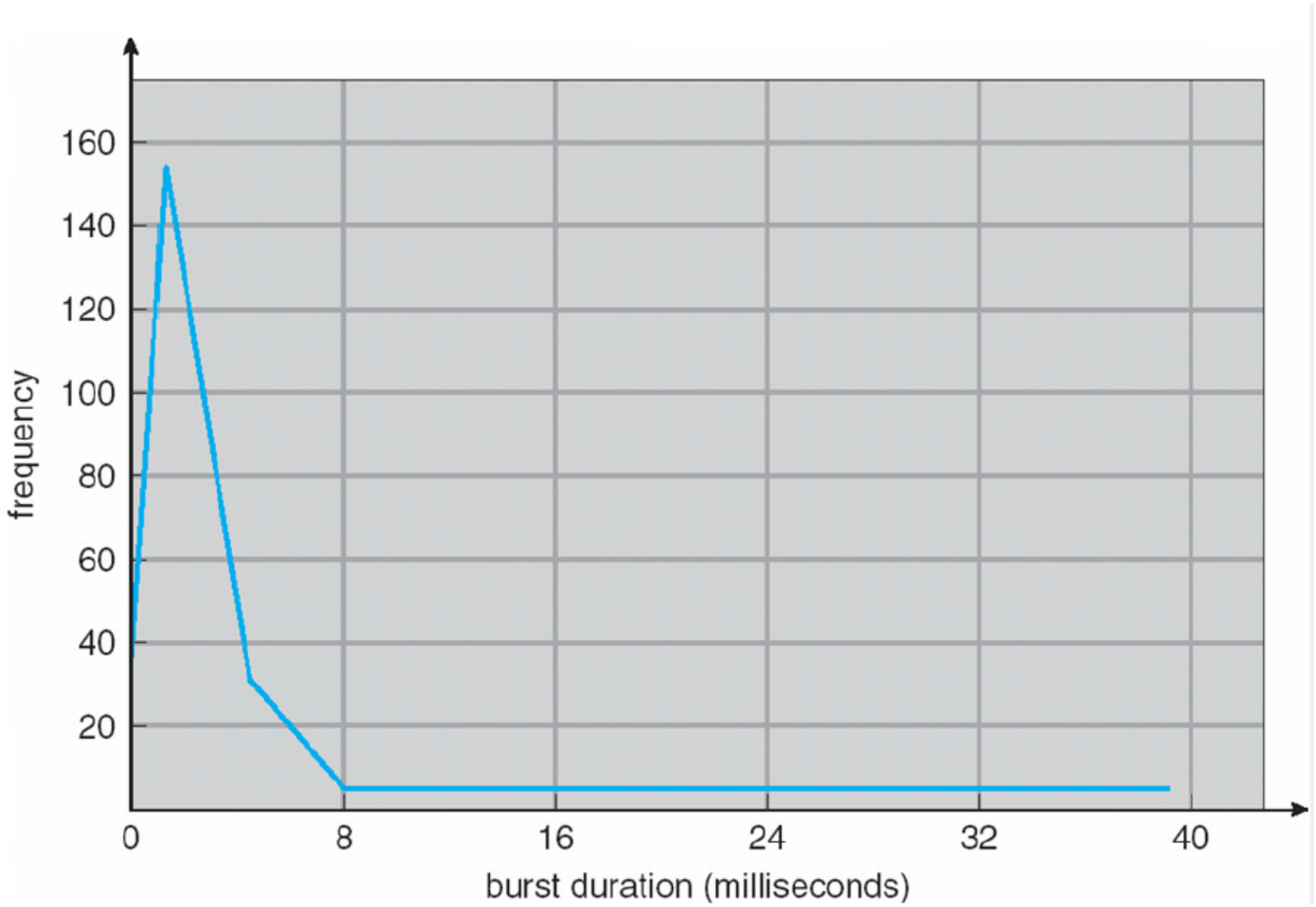
process	time
P_1	6
P_2	3
P_3	1
P_4	7

Bursts of computation & I/O

- Jobs contain I/O and computation
  - Bursts of computation
  - Then must wait for I/O
- To Maximize throughput
  - Must maximize CPU utilization
  - Also maximize I/O device utilization
- How to do?
  - Overlap I/O & computation from multiple jobs
  - Means response time very important for I/O-intensive jobs: I/O device will be idle until job gets small amount of CPU to issue next I/O request



Histogram of CPU-burst times

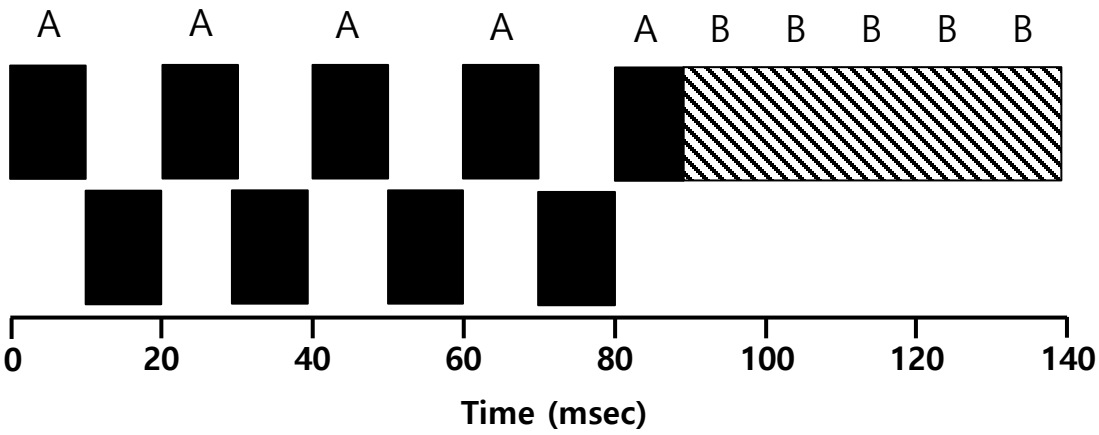


- **What does this mean for FCFS?**

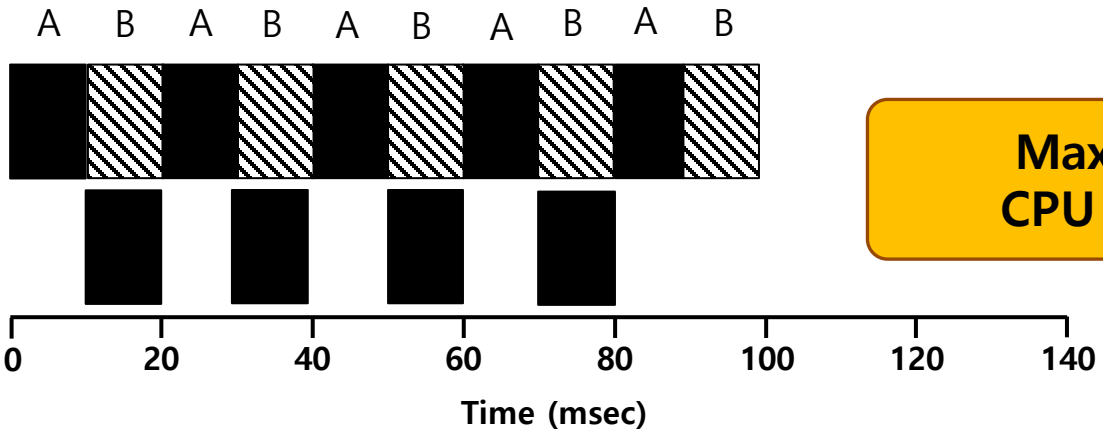
Incorporating I/O

- Let's relax assumption 3: The programs perform I/O
- Example:
  - A and B need 50ms of CPU time each.
  - A runs for 10ms and then issues an I/O request
    - I/Os each take 10ms
  - B simply uses the CPU for 50ms and performs no I/O
  - The scheduler runs A first, then B after

Incorporating I/O (Cont.)



Poor Use of Resources



Overlap Allows Better Use of Resources

Maximize the CPU utilization

Incorporating I/O (Cont.)

- When a job initiates an I/O request.
  - The job is blocked waiting for I/O completion.
  - The scheduler should schedule another job on the CPU.

- When the I/O completes
  - An interrupt is raised.
  - The OS moves the process from blocked back to the ready state.