# CPU Scheduling

Heechul Yun

# Agenda

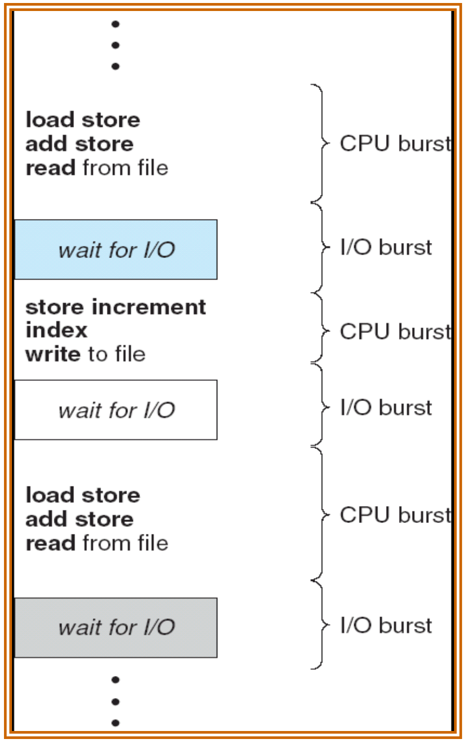
- Introduction to CPU scheduling
- Classical CPU scheduling algorithms

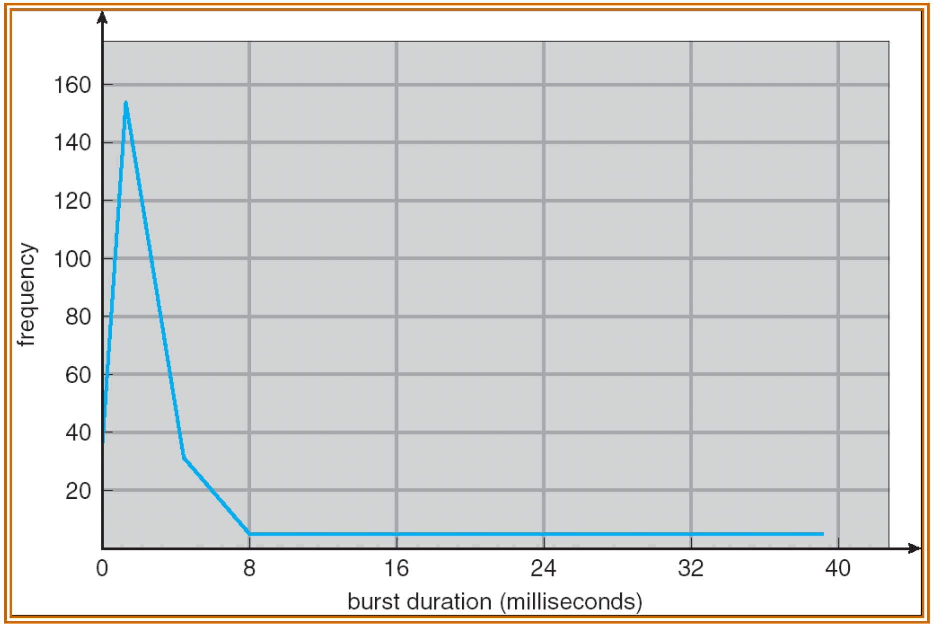
# CPU Scheduling

- CPU scheduling is a **policy** to decide
  - **Which** thread to run next?
  - **When** to schedule the next thread?
  - **How long**?

- Context switching is a **mechanism**
  - To change the running thread

# Assumption: CPU Bursts

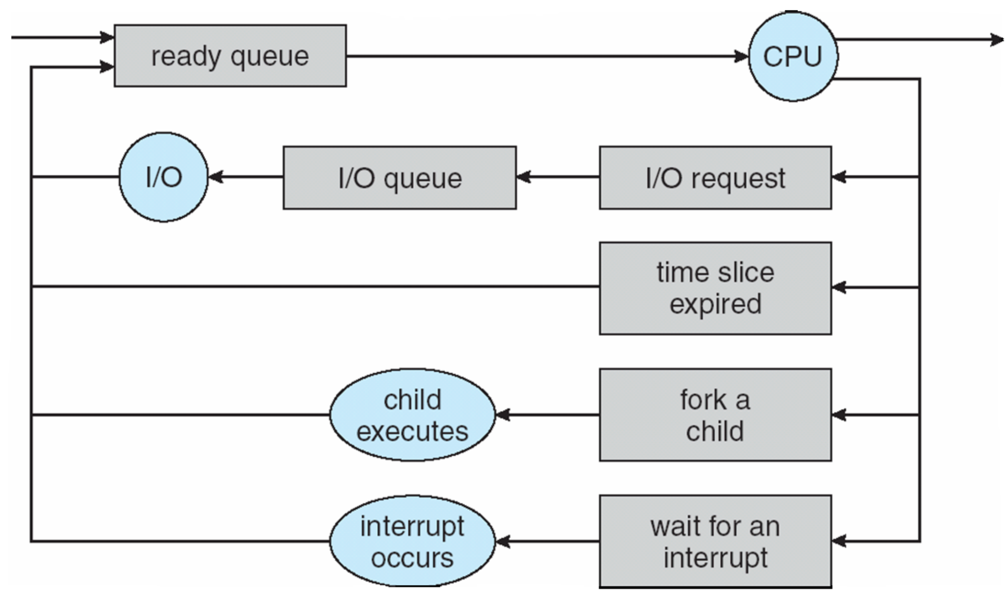



- Execution model
  - Program uses the CPU for a while and the does some I/O, back to use CPU, …, keep alternating

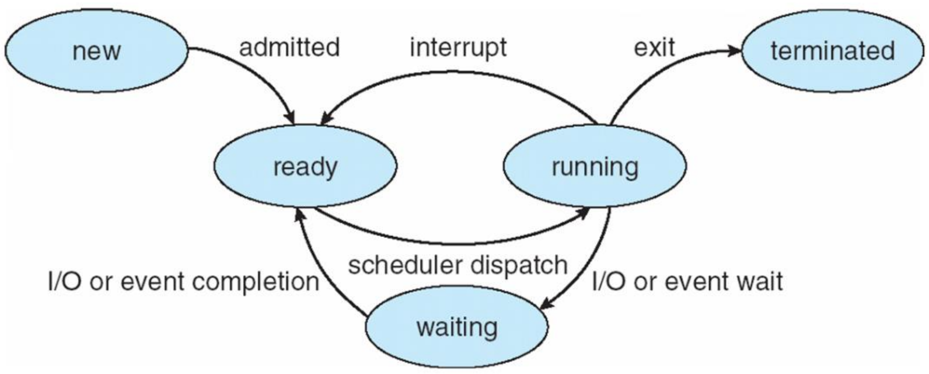
# CPU Scheduler

- An OS component that determines which thread to run, at what time, and how long
  - Among threads in the **ready queue**



# CPU Scheduler

- When the scheduler runs?



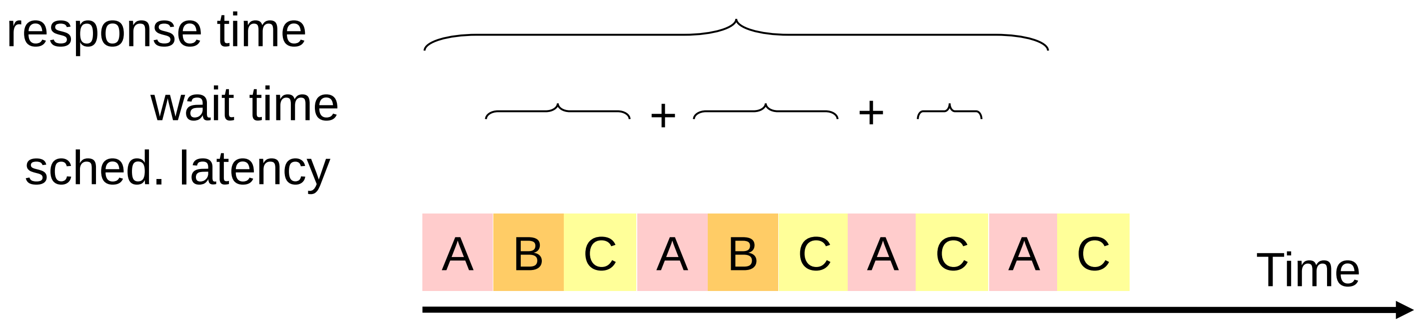

  – The running thread finishes
  – The running thread voluntarily gives up the CPU
    - yield, block on I/O, …
  – The OS **preempts** the current running thread
    - quantum expire (timer interrupt)

# Performance Metrics for CPU Scheduling

- CPU utilization
  - % of time the CPU is busy doing something
- Throughput
  - #of jobs done / unit time
- **Response time (Turn-around time)**
  - Time to complete a task (ready -> complete)
- **Waiting time**
  - Time spent on waiting in the ready queue
- **Scheduling latency**
  - Time to schedule a task (ready -> first scheduled)

# Example

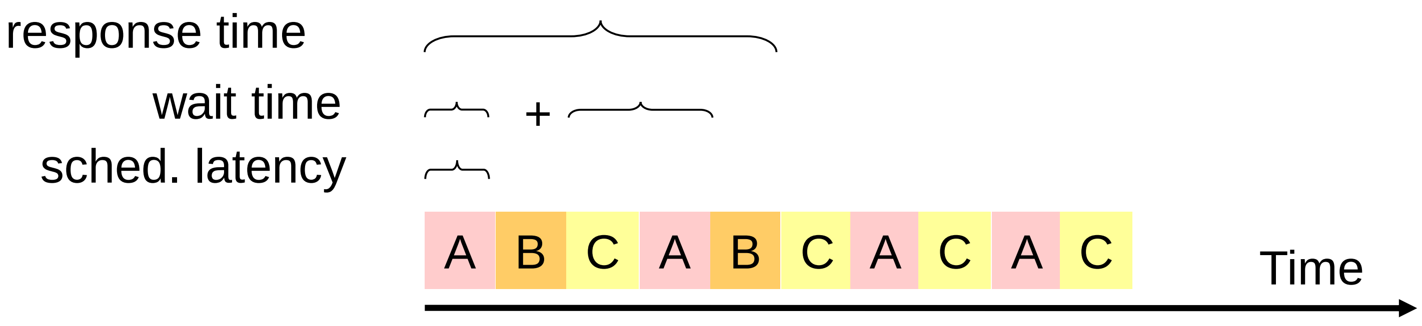
- Assumption: A, B, C are released at time 0




- The times of Process **A**
  - Response time: 9
  - Wait time: 5
  - Sched. latency: 0

# Example

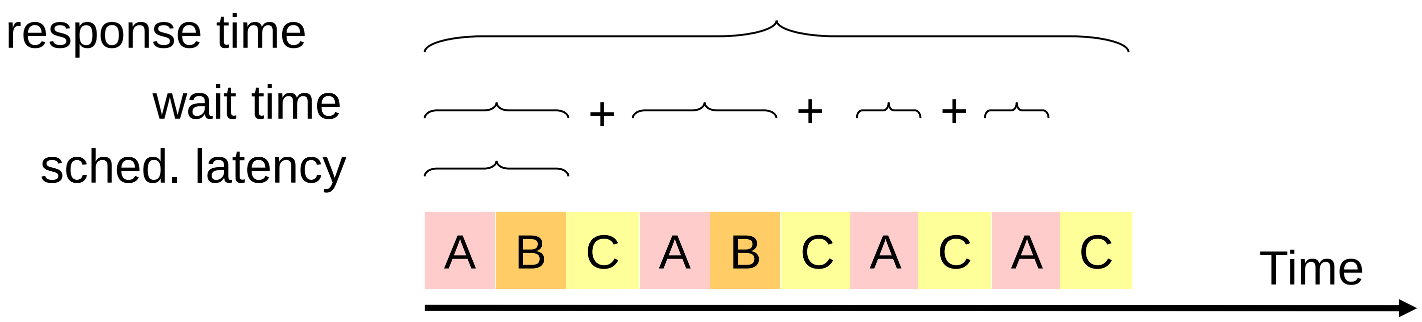
- Assumption: A, B, C are released at time 0




- The times of Process **B**
  - Response time: 5
  - Wait time: 3
  - Latency: 1

# Example

- Assumption: A, B, C are released at time 0




- The times of Process **C**
  - Response time: 10
  - Wait time: 6
  - Latency: 2

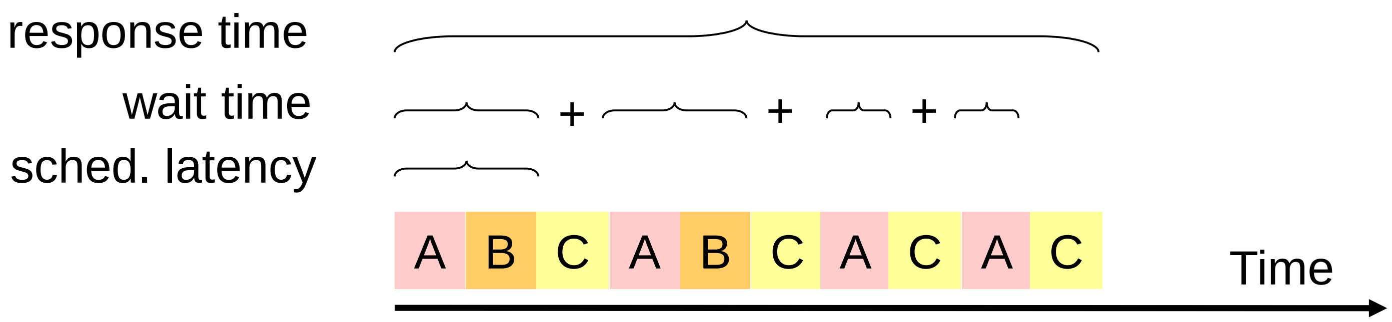
# Recap: CPU Scheduling

- CPU scheduling is a **policy** to decide
  - **Which** thread to run next?
  - **When** to schedule the next thread?
  - **How long**?

- Context switching is a **mechanism**
  - To change the running thread

# Recap: Example Metrics

- Assumption: A, B, C are released at time 0



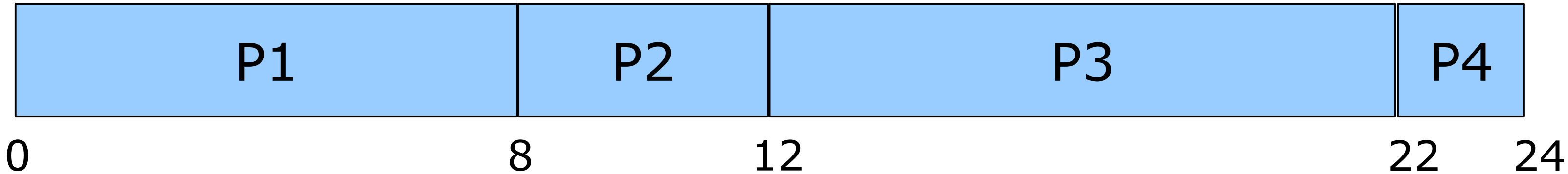

- The times of Process **C**
  - Response time: 10
  - Wait time: 6
  - Latency: 2

# Workload Model and Gantt Chart

- Workload model

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 8 |
| P2 | 1 | 4 |
| P3 | 1 | 10 |
| P4 | 6 | 2 |

- Gantt chart
  - bar chart to illustrate a particular schedule



# Agenda

- Basic scheduling policies
  - First-come-first-serve (FCFS)
  - Shortest-job-first (SJF)
  - Shortest-remaining-time-first (SRTF)
  - Round-robin (RR)

# Scheduling Policy Goals

- Maximize throughput (minimize avg. waiting time)
  - High throughput (#of jobs done / time) is good
- Minimize scheduling latency
  - Important to interactive applications (games, editor, …)
- Fairness
  - Make all threads progress equally

- Goals often conflicts
  - Frequent context switching may be good for reducing response time, but not so much for maximizing throughput

# First-Come, First-Served (FCFS)

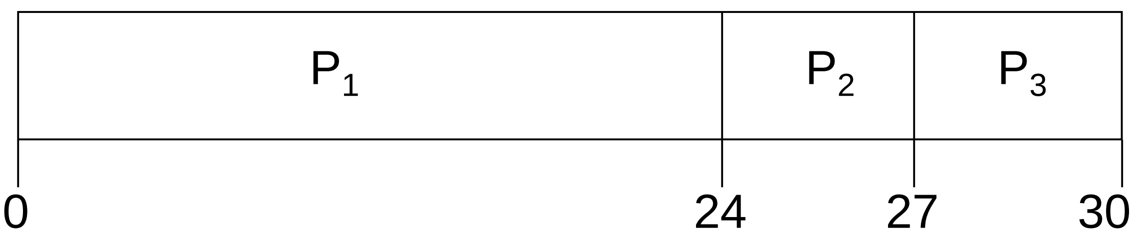
- FCFS
  - Assigns the CPU based on the order of the requests.
  - Implemented using a FIFO queue.

# FCFS

- Example

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 24 |
| P2 | 0 | 3 |
| P3 | 0 | 3 |

  – Suppose that the processes arrive in the order: $P_1$ , $P_2$ , $P_3$



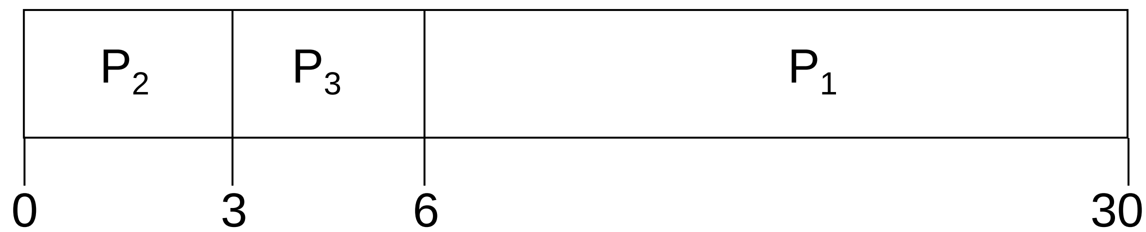

  – Waiting time?
    - P1 = 0; P2 = 24; P3 = 27
  – Average waiting time
    - (0 + 24 + 27)/3 = 17

# FCFS

- Example 2

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 24 |
| P2 | 0 | 3 |
| P3 | 0 | 3 |

  – Suppose that the processes arrive in the order: $P_2$ , $P_3,$ $P_1$




  – Waiting time?
    - P1 = 6; P2 = 0; P3 = 3
  – Average waiting time
    - (6 + 0 + 3)/3 = 3
  – Much better than previous case → performance varies greatly depending on the scheduling order

# Shortest Job First (SJF)

- Can we always do better than FIFO?
  - Yes: **if** you know the tasks' CPU burst times

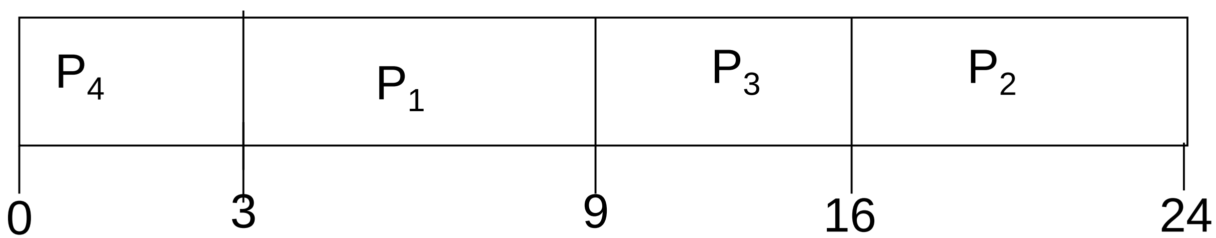
- Shortest Job First (SJF)
  - Order jobs based on their burst lengths
  - Executes the job with the shortest CPU burst first
  - SJF is optimal
    - Achieves minimum average waiting time

# Shortest Job First (SJF)

- Example

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 6 |
| P2 | 0 | 8 |
| P3 | 0 | 7 |
| P4 | 0 | 3 |

  - Gantt chart



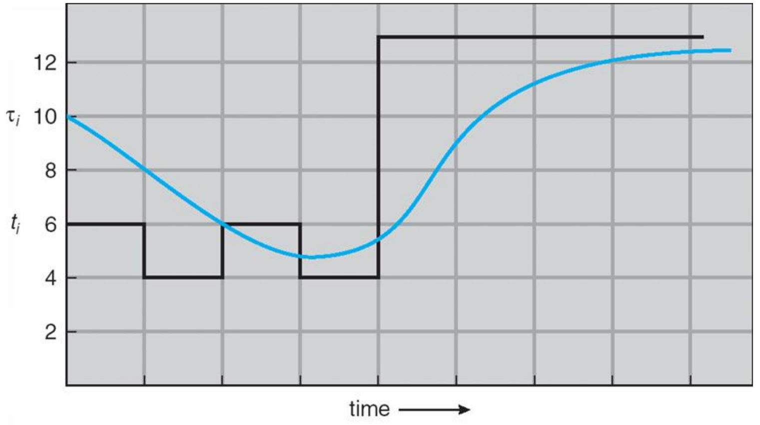

  - Average waiting time?
    - (3 + 16 + 9 + 0) / 4 = 7

- How to know the CPU burst time **in advance**?

# Determining CPU Burst Length

- Can only estimate the length
  - Next CPU burst similar to previous CPU bursts ?
  - Predict based on the past history
- Exponential weighted moving average (EWMA)
  - of past CPU bursts

1. $t_n$ = actual length of $n^{th}$ CPU burst
2. $\tau_{n+1}$ = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define: $\tau_{n=1} = \alpha\, t_n + (1-\alpha)\tau_n$.

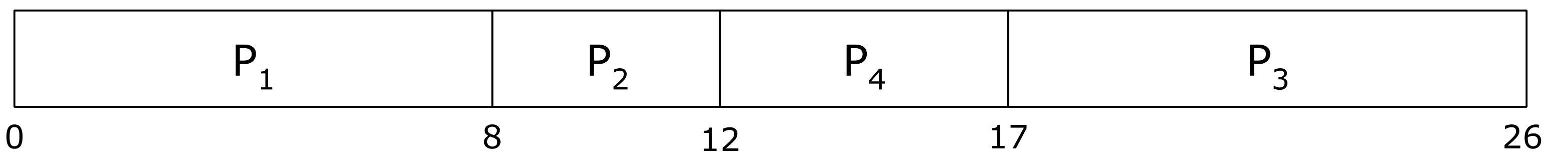



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| CPU burst ($t_i$) | | 6 | 4 | 6 | 4 | 13 | 13 | 13 | … |
| "guess" ($\tau_i$) | 10 | 8 | 6 | 6 | 5 | 9 | 11 | 12 | … |

# Shortest Job First (SJF)

- What if jobs don't arrive at the same time?

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 8 |
| P2 | 1 | 4 |
| P3 | 2 | 9 |
| P4 | 3 | 5 |




– Average waiting time
  - (0+7+15+9)/4 = 7.5

# Shortest Remaining Time First (SRTF)

- Preemptive version of SJF
- New shorter job preempt longer running job

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 8 |
| P2 | 1 | 4 |
| P3 | 2 | 9 |
| P4 | 3 | 5 |

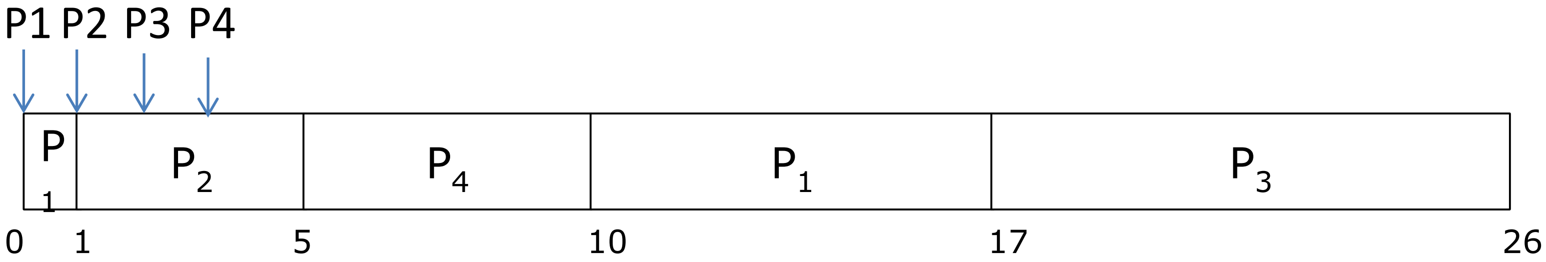
| $P_1$ | $P_2$ | $P_4$ | $P_1$ | $P_3$ | |
|---|---|---|---|---|---|
| 0 | 1 | 5 | 10 | 17 | 26 |

- Average waiting time
  - (9 + 0 + 15 + 2 ) / 4 = 6.5

# Quiz: SRTF

- Average waiting time?

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 8 |
| P2 | 1 | 4 |
| P3 | 2 | 9 |
| P4 | 3 | 5 |




- (9 + 0 + 15 + 2 ) / 4 = 6.5

# So Far…

- FIFO
  - In the order of arrival
  - Non-preemptive
- SJF
  - Shortest job first.
  - Non preemptive
- SRTF
  - Preemptive version of SJF

# Issues

- FIFO
  - Bad average response time
- SJF/SRTF
  - Good average waiting time
  - IF you know or can predict the future
- Time-sharing systems
  - Multiple users share a machine
  - Need high interactivity → low **scheduling latency**

# Round-Robin (RR)

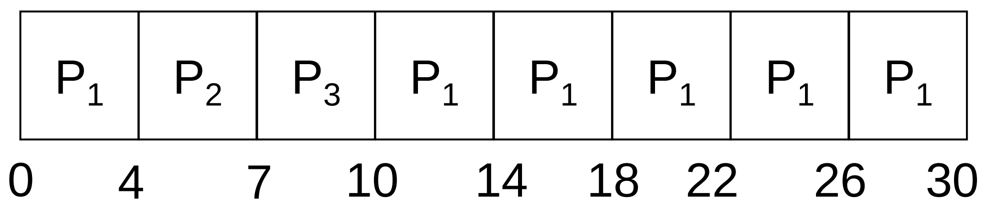
- FIFO with preemption
- Simple, fair, and easy to implement
- Algorithm
  - Each job executes for a fixed time slice: **quantum**
  - When quantum expires, the scheduler preempts the task
  - Schedule the next job and continue...

# Round-Robin (RR)

- Example
  - Quantum size = 4

| Process | Burst Times |
|---|---|
| P1 | 24 |
| P2 | 3 |
| P3 | 3 |

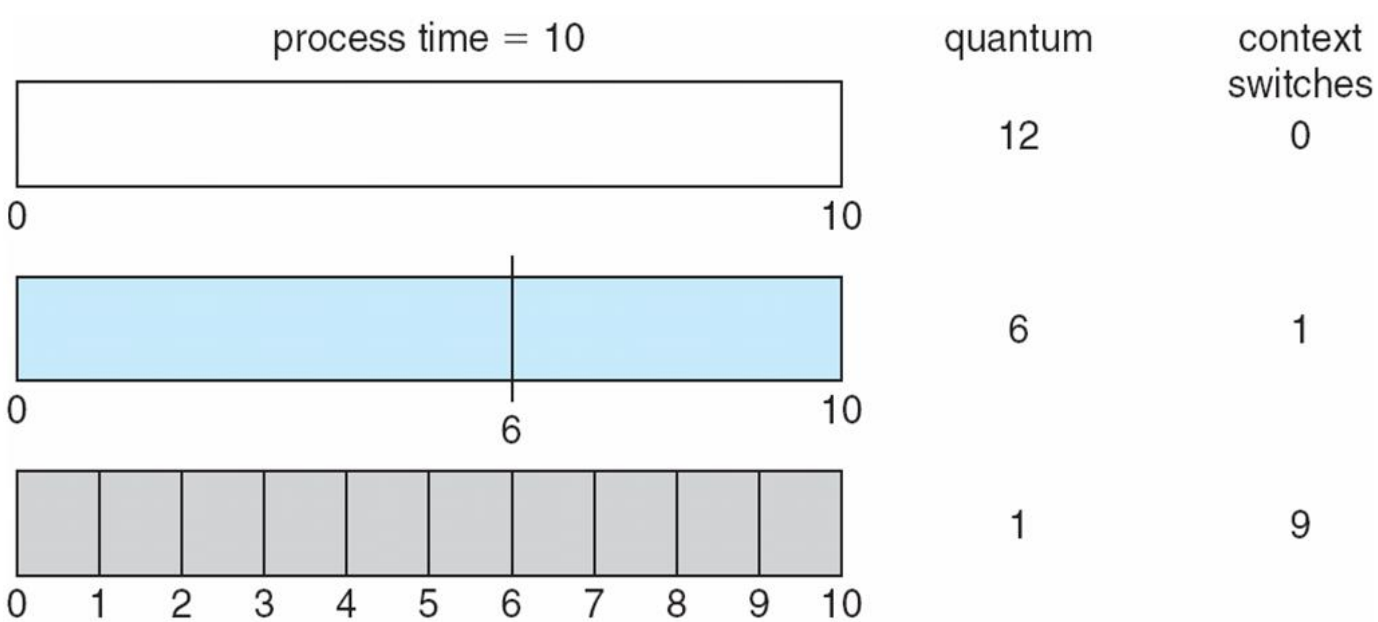
  - Gantt chart




  - Sched. Latency (between ready to first schedule)
    - P1: 0, P2: 4, P3: 7. average response time = (0+4+7)/3 = 3.67
  - Waiting time
    - P1: 6, P2: 4, P3: 7. average waiting time = (6+4+7)/3 = 5.67

# How To Choose Quantum Size?

- Quantum length
  - Too short → high overhead (why?)
  - Too long → bad scheduling latency
    - Very long quantum → FIFO

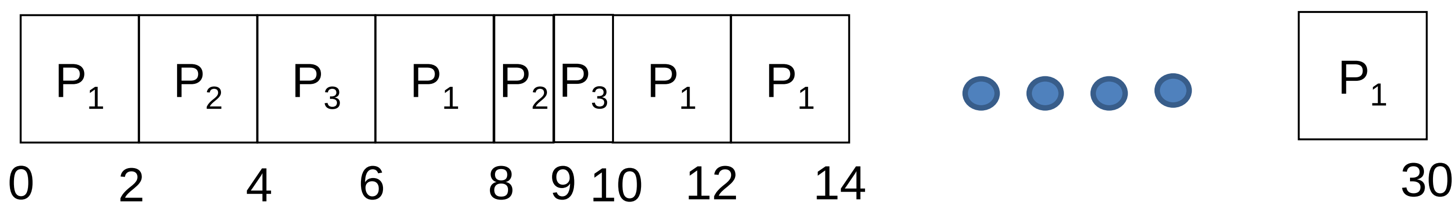


# Round-Robin (RR)

- Example
  - Quantum size = 2

| Process | Burst Times |
|---|---|
| P1 | 24 |
| P2 | 3 |
| P3 | 3 |

  - Gantt chart




  - Scheduling latency
    - P1: 0, P2: 2, P3: 4. average response time = (0+2+4)/3 = 2
  - Waiting time
    - P1: 6, P2: 6, P3: 7. average waiting time = (6+6+7)/3 = 6.33

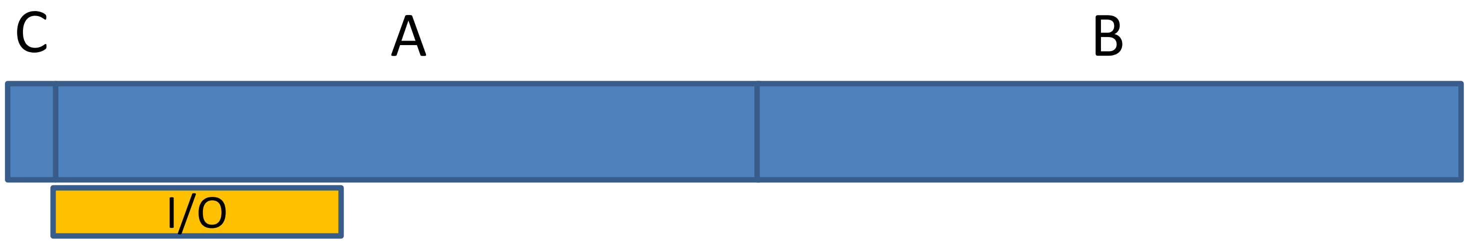
# Discussion

- Comparison between FCFS, SRTF(SJF), and RR
  - What to choose for smallest average waiting time?
    - SRTF (SFJ) is the optimal
  - What to choose for better interactivity?
    - RR with small time quantum (or SRTF)
  - What to choose to minimize scheduling overhead?
    - FCFS

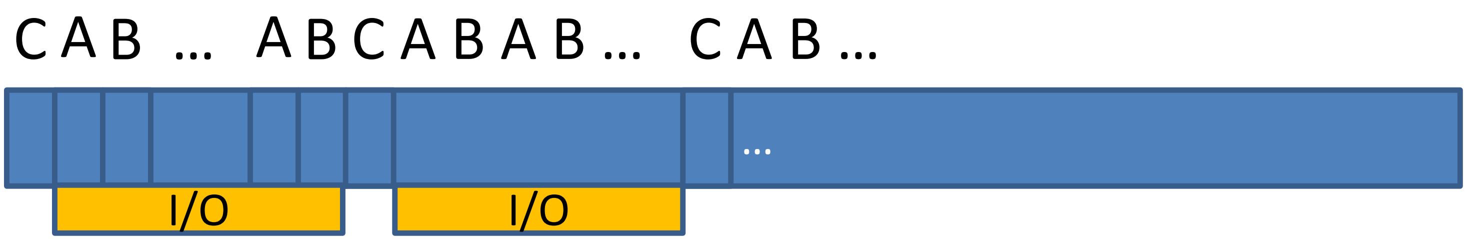
# Example


- Task A and B
  - CPU bound, run an hour
- Task C
  - I/O bound, repeat(1ms CPU, 9ms disk I/O)
- FCFS?
  - If A or B is scheduled first, C can begins an hour later
- RR and SRTF?

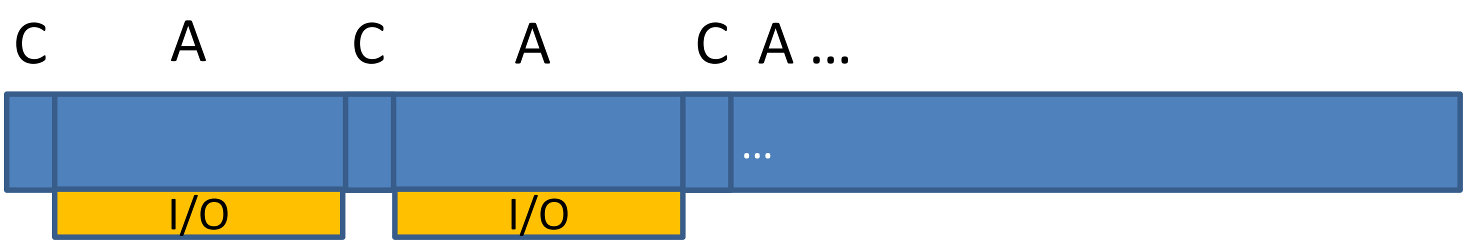
# Example Timeline




RR with 100ms time quantum




RR with 1ms time quantum




SRTF

# Summary

- First-Come, First-Served (FCFS)
  - Run to  completion in order of arrival
  - Pros: simple, low overhead, good for batch jobs
  - Cons: short jobs can stuck behind the long ones
- Round-Robin (RR)
  - FCFS with preemption. Cycle after a fixed time quantum
  - Pros: better interactivity (low average scheduling latency)
  - Cons: performance is dependent on the quantum size
- Shortest Job First (SJF)/ Shorted Remaining Time First (SRTF)
  - Shorted job (or shortest remaining job) first
  - Pros: optimal average waiting time
  - Cons: you need to know the future, long jobs can be starved by short jobs

KU THE UNIVERSITY OF KANSAS