K22 – Operating Systems: Design Principles and Internals

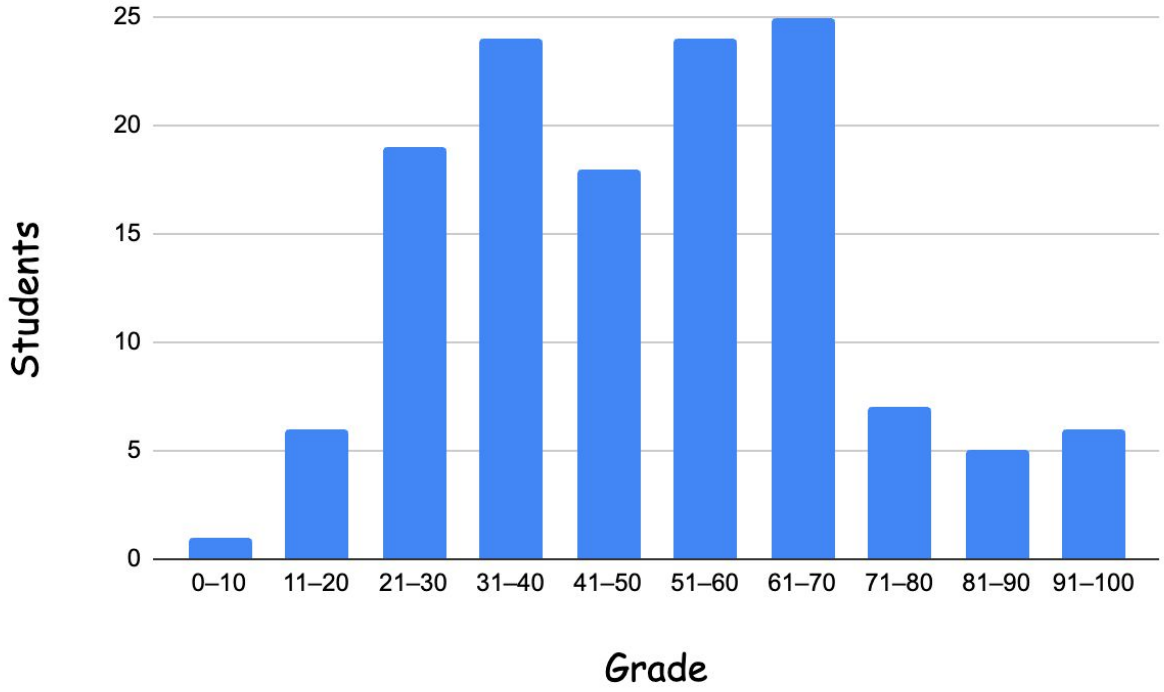
Fall 2025 @dit

Vaggelis Atlidakis

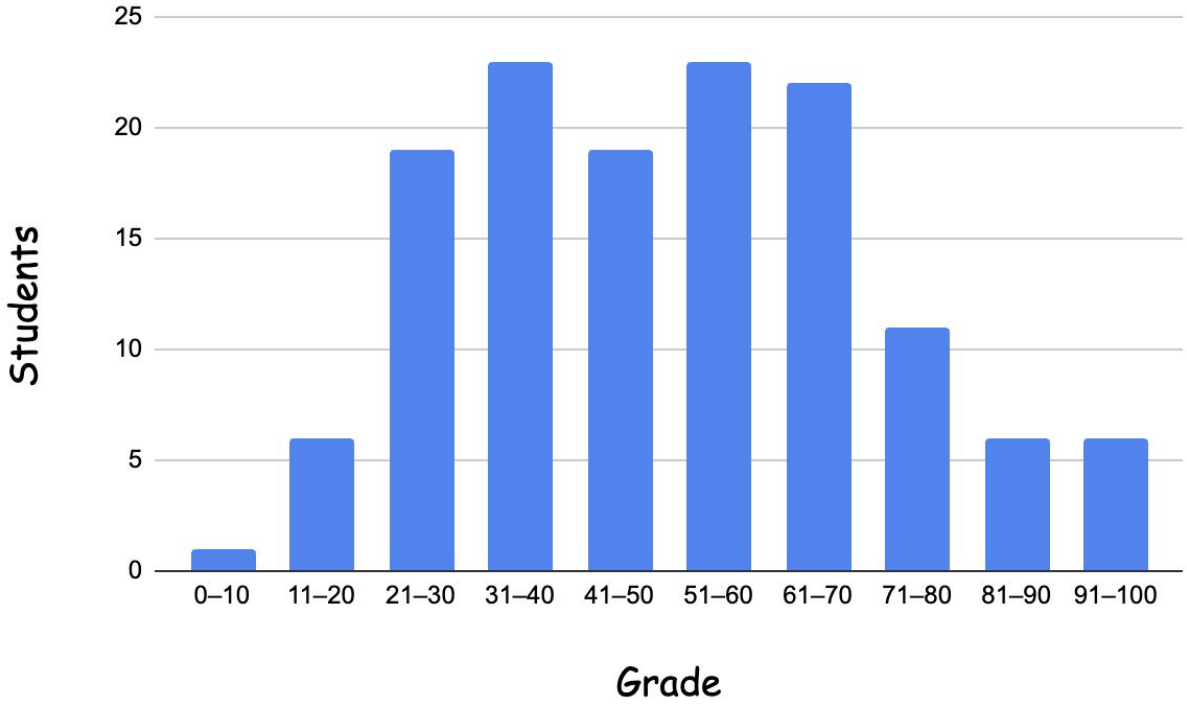
Lecture 16

References: Similar OS courses @Columbia, @Stanford, @UC San Diego, @Brown, @di (previous years); and textbooks: Operating Systems: Three Easy Pieces, Operating Systems: Principles and Practice, Operating System Concepts, Linux Kernel Development, Understanding the Linux Kernel

Midterm (median 51 / std.dev.: 21)



Revised midterm (median 51 / std.dev.: 21)



Overview

- We'll start from hardware and follow a question-oriented approach
  - ~~Intro [Q: What is an OS?]~~
  - ~~Events [Q: When does the OS run?]~~
  - ~~Runtime [Q: How does a program look like in memory?]~~
  - ~~Processes [Q: What is a process?]~~
  - ~~IPC [Q: How do processes communicate?]~~
  - ~~Threads [Q: What is a thread?]~~
  - ~~Synchronization [Q: What goes wrong w/o synchronization?]~~
  - **Time Management [Q: What is scheduling?]**
  - Memory Management [Q: What is virtual memory?]
  - Files [Q: What is a file descriptor?]
  - Storage Management [Q: How do we allocate disk space to files?]

* Basic (H/W & S/W)
* Abstractions
* Primitives
* **Mechanisms**

Workloads and scheduling requirements

Real time workloads: Hard real time and soft real time

- Hard real time
  - Their tasks must finish within specific deadlines
  - Example: Pacemakers, Airbag deployment systems, Autopilots
  - Sched. goals: **Zero miss rate; Guarantees every time**
  - Sched. algorithms: Earliest Deadline First (EDF)
- Soft real time
  - Their tasks must receive priority over lower-priority tasks
  - Example: Video Streaming / Multimedia applications
  - Sched. goals: **Bounded latency**
  - Sched. algorithms: Priority-based scheduling

Workloads and scheduling requirements

CPU- vs I/O-bound workloads

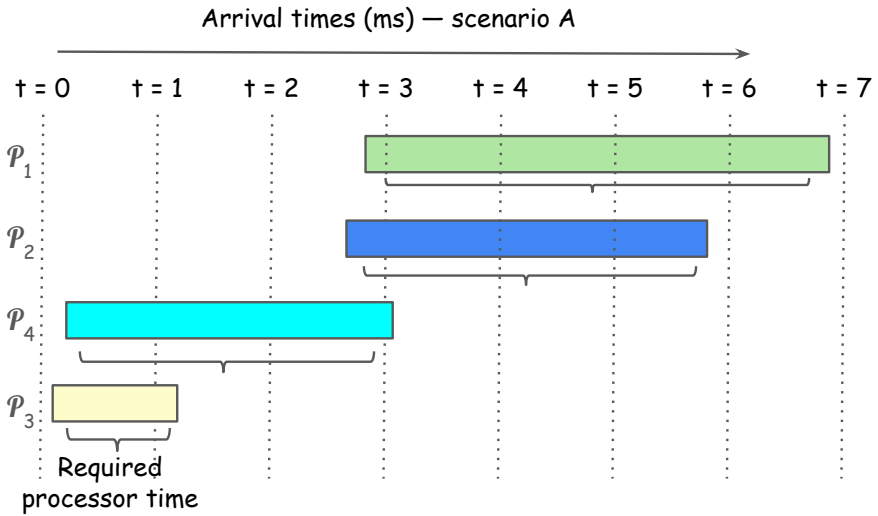
- CPU-bound
  - Their tasks spend most time doing intensive computation
  - Rarely yield voluntarily and rarely need to perform I/O
  - Example: Scientific simulations / computations
  - Sched. goals: **Balanced processor time / avoid starvation**
  - Sched. algorithms: RR with large quanta

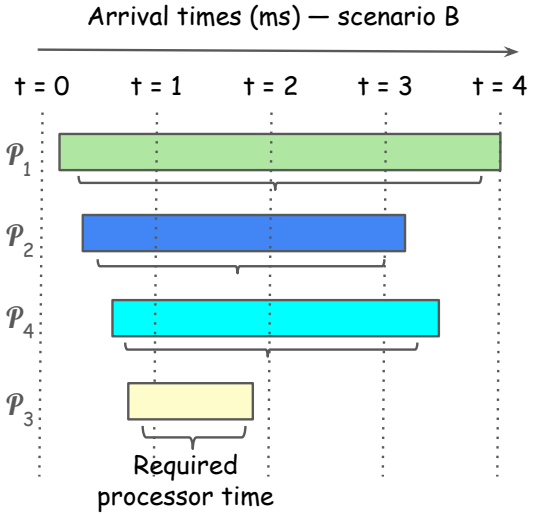
Workloads and scheduling requirements

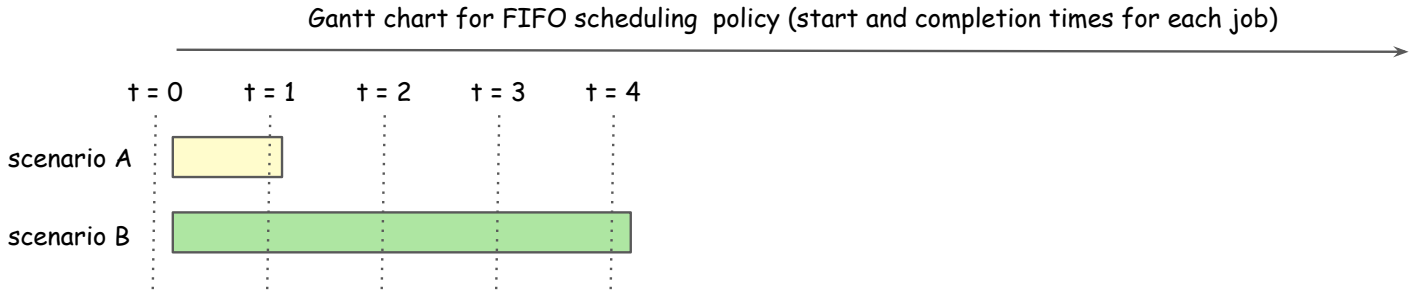
CPU- vs I/O-bound workloads

- I/O-bound workloads
  - Their tasks spend most of their time waiting for I/O
  - Short processor bursts and then block again
  - Example: Downloading a file / fetching data from disk
  - Sched. goals: Minimize I/O device idle periods by promptly allocating the processor for the brief time needed to initiate I/O requests (usually via DMA)
  - Sched. algorithms: Priority-based favoring I/O-bound tasks

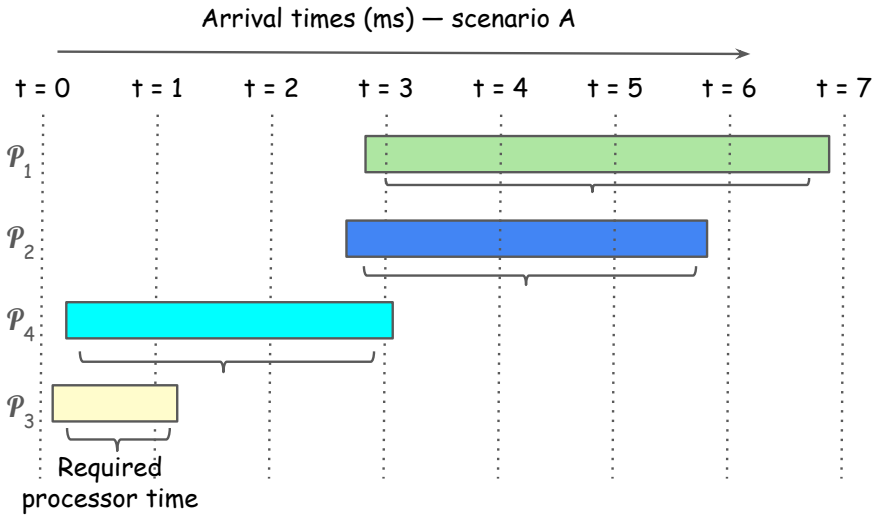
First in First Out scheduling policy (SCHED_FIFO)

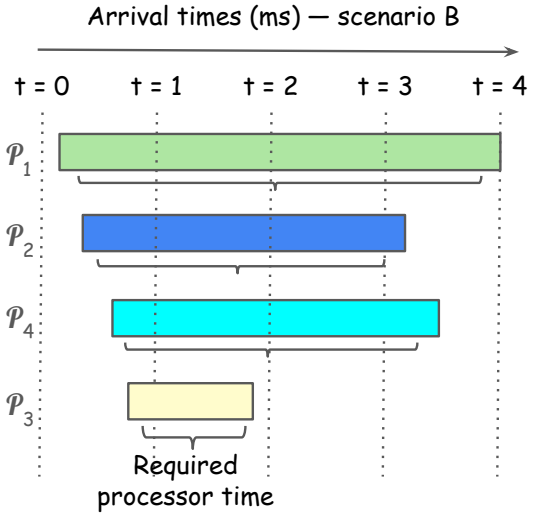


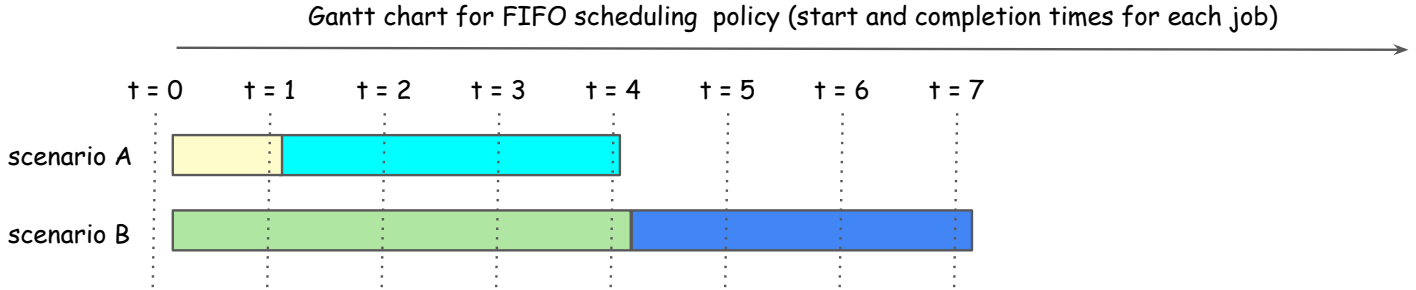




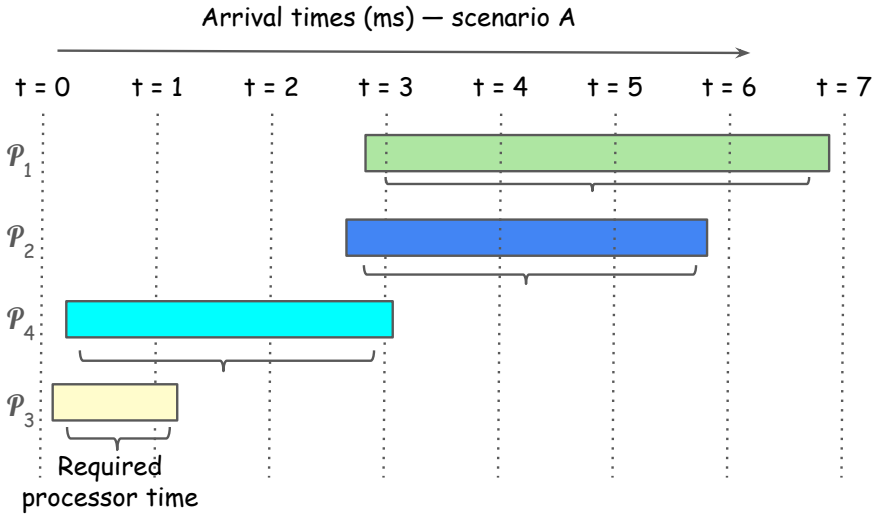
First in First Out scheduling policy (SCHED_FIFO)

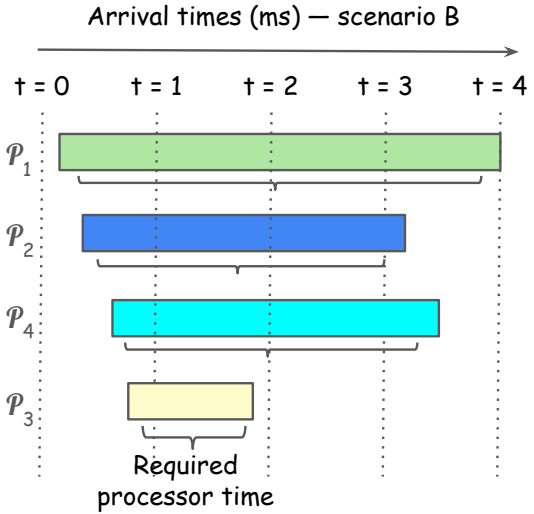


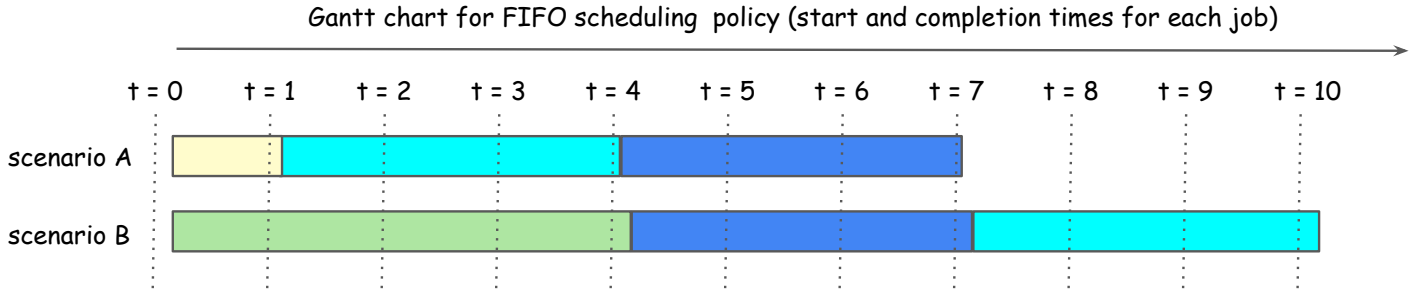




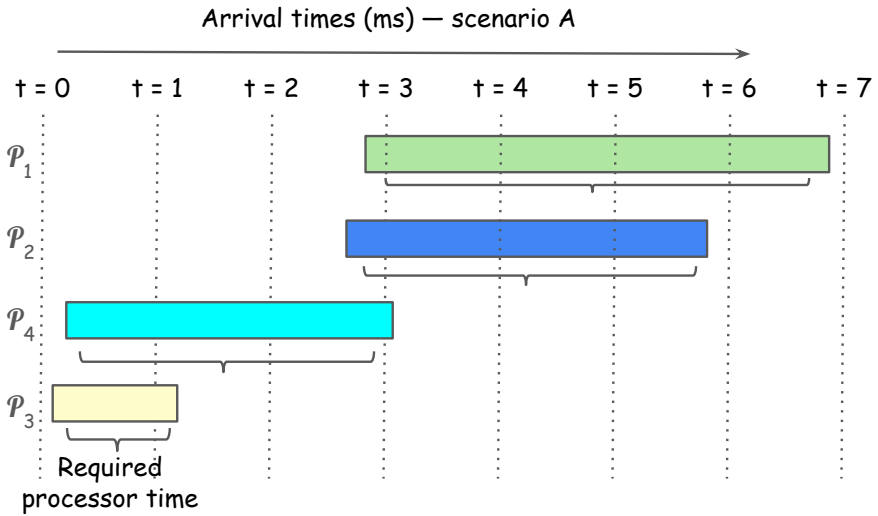
First in First Out scheduling policy (SCHED_FIFO)

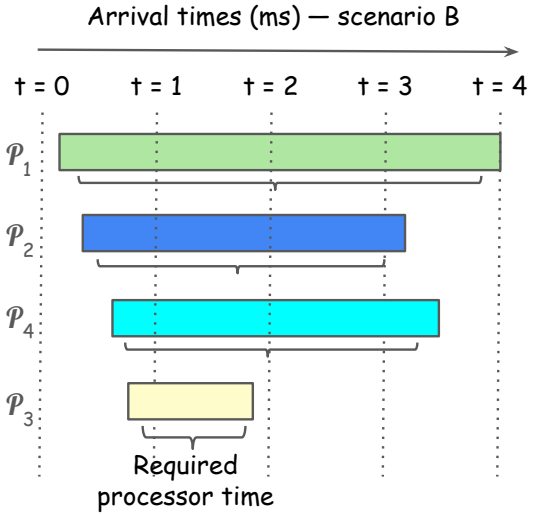


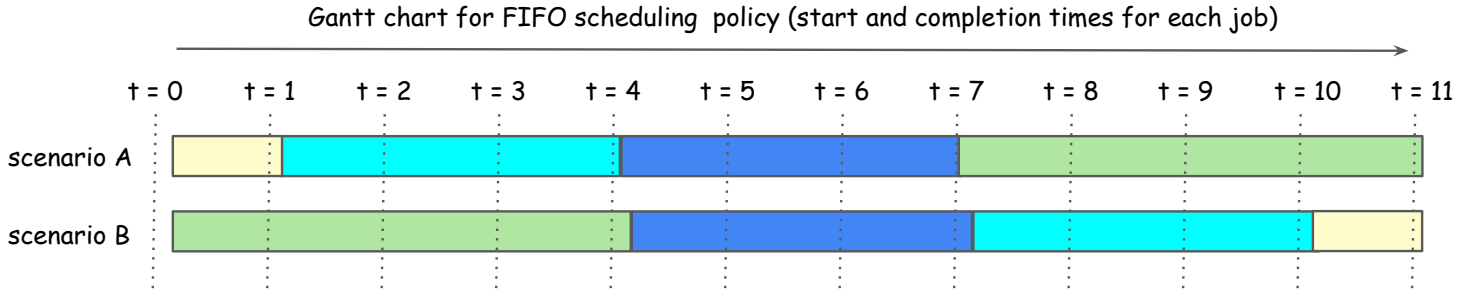




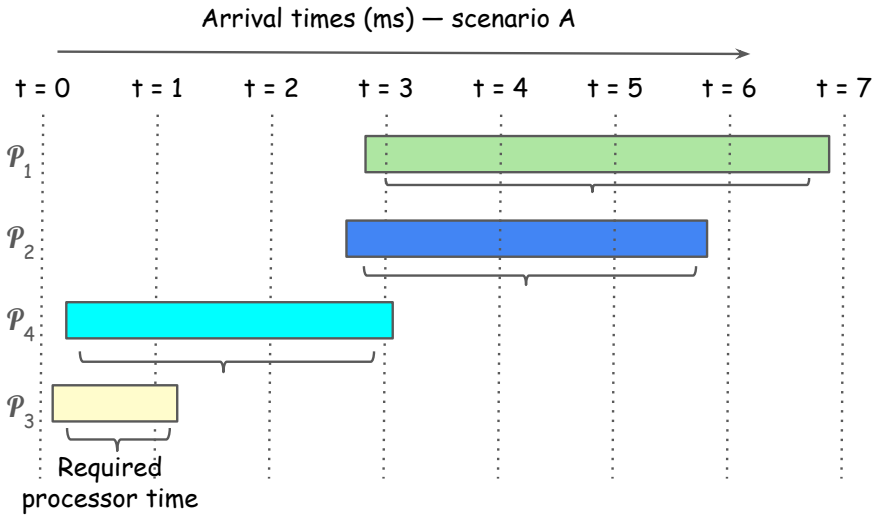
First in First Out scheduling policy (SCHED_FIFO)

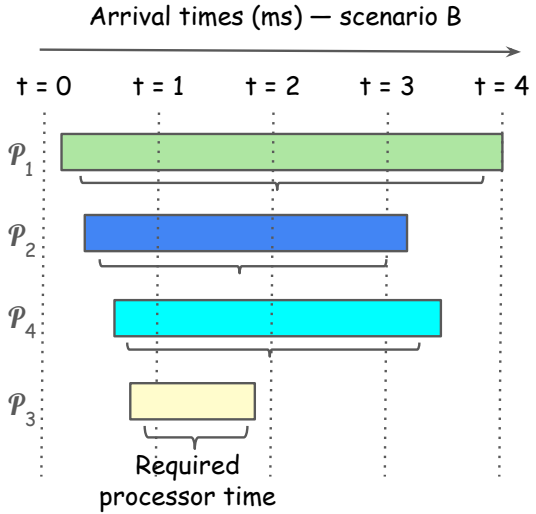


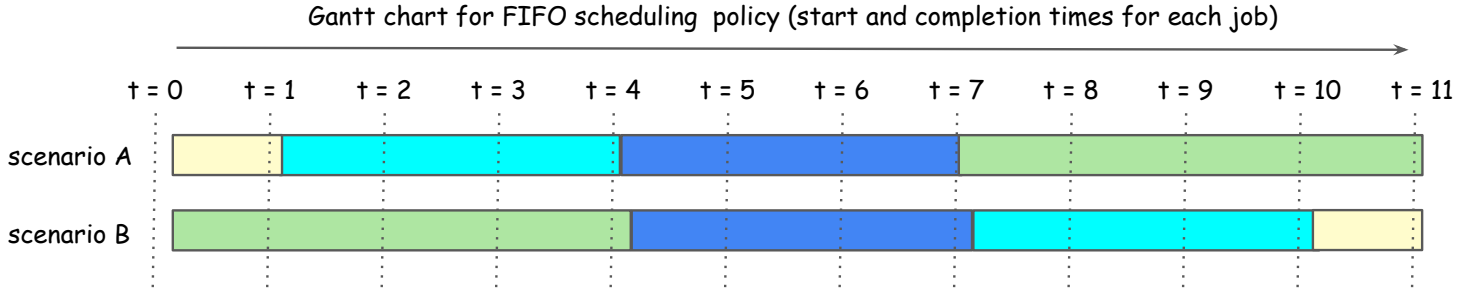




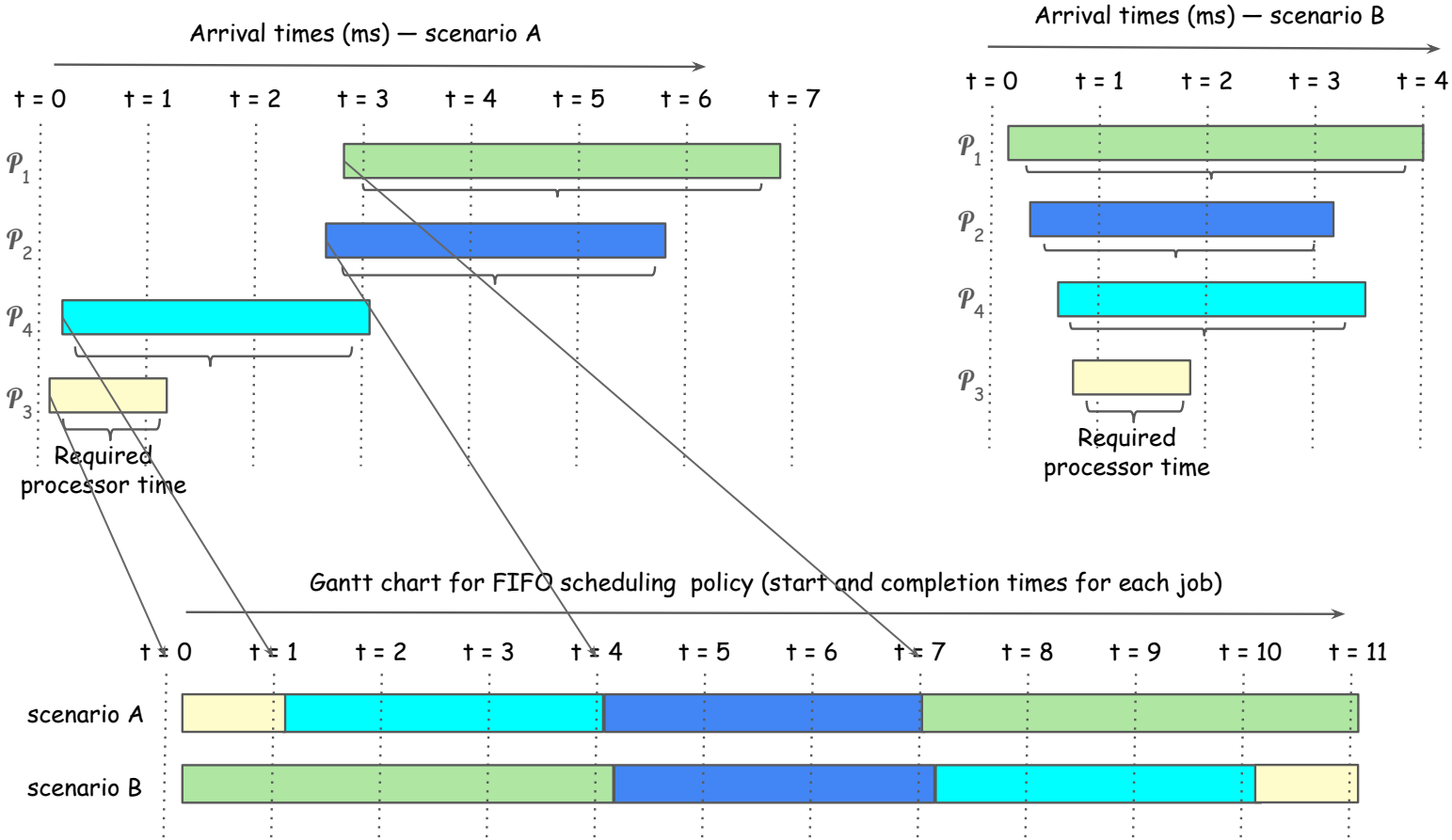
Calculating response time (latency)





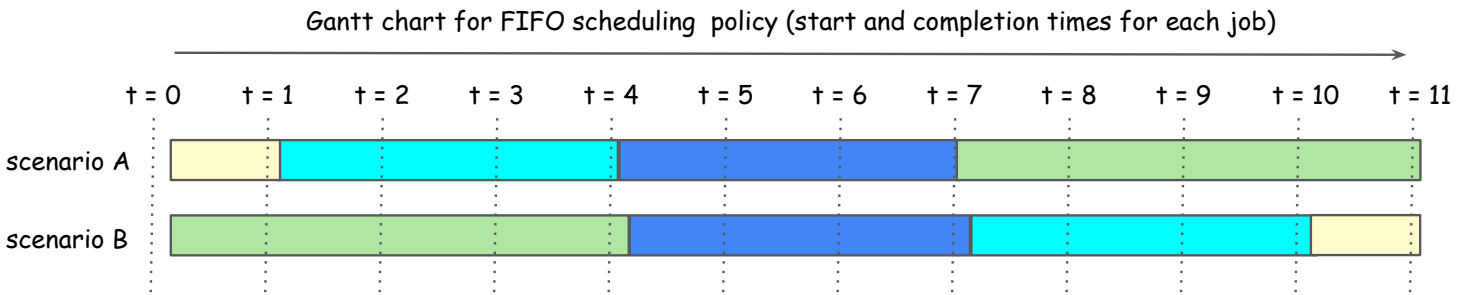


Calculating response time (latency)



Calculating response time (latency)

- **Average**
  - **Scenario A**: (0ms + 1ms + 1ms + 4ms) / 4 = **1.5ms**
  - **Scenario B**: (0ms + 4ms + 7ms + 10ms) / 4 = **5.25ms**



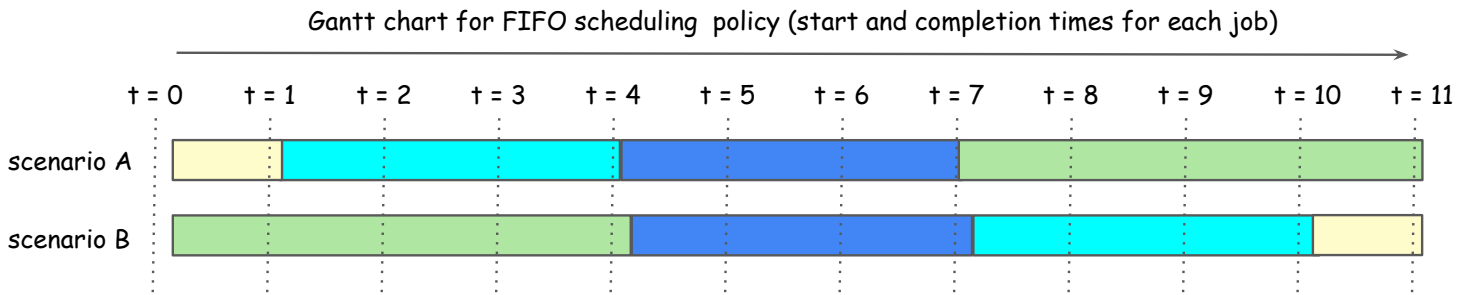
Calculating response time (latency)

> Average

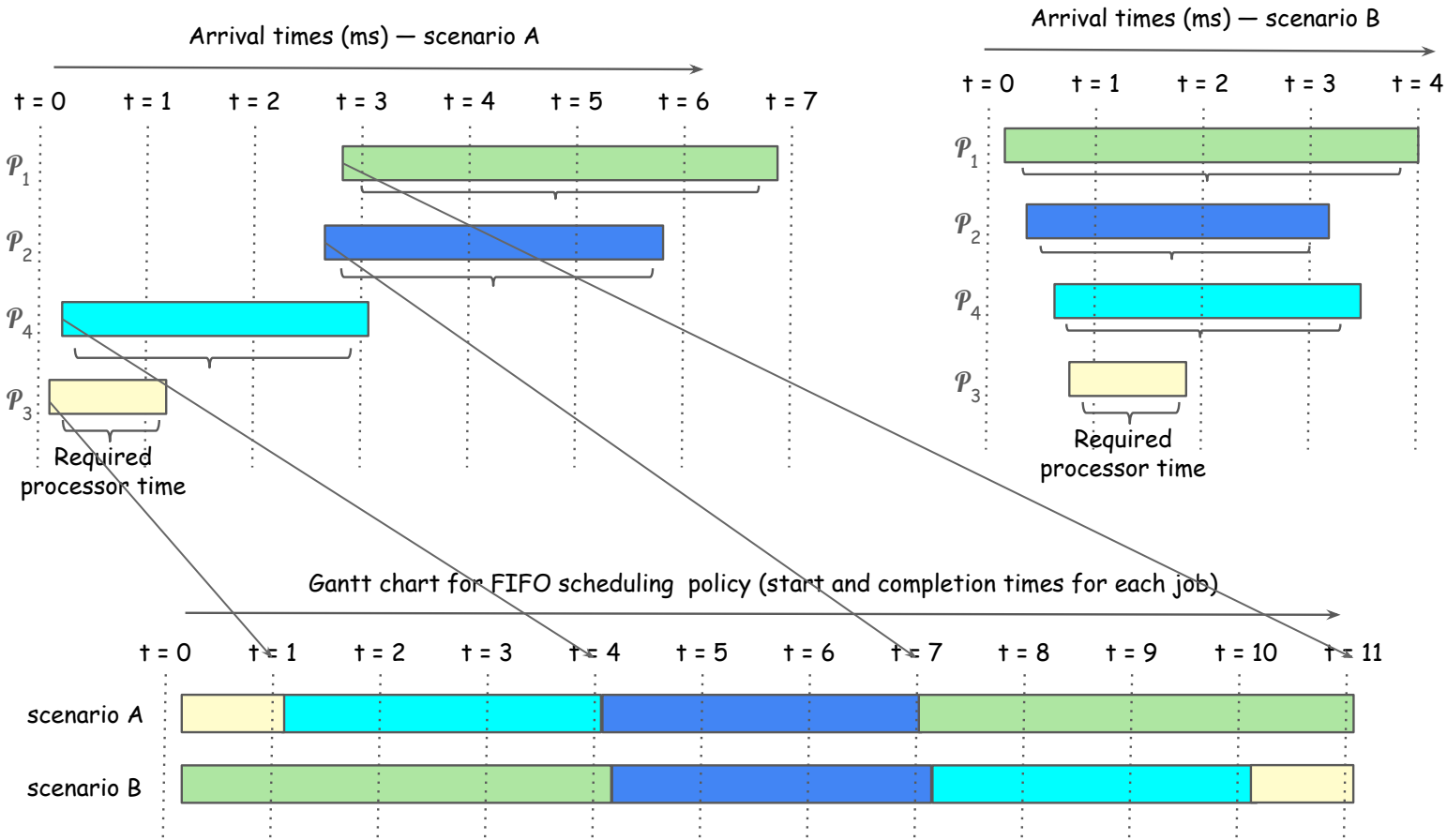
- **Scenario A**: (0ms + 1ms + 1ms + 4ms) / 4 = **1.5ms**
- **Scenario B**: (0ms + 4ms + 7ms + 10ms) / 4 = **5.25ms**

> Worst case

- **Scenario A**: (0ms + 1ms + 1ms + **4ms**) => **4ms**
- **Scenario B**: (0ms + 4ms + 7ms + **10ms**) => **10ms**

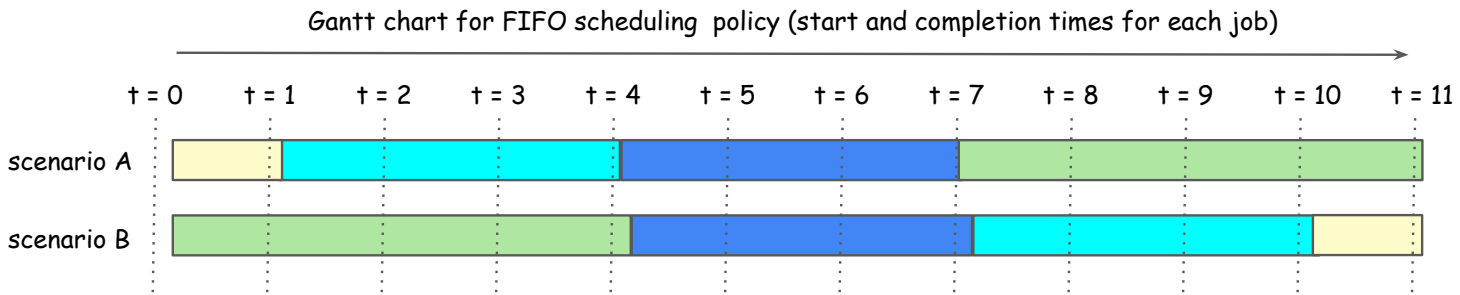


Calculating completion time



Calculating completion time

- **Average**
  - **Scenario A**: (1ms + 4ms + 4ms + 8ms) / 4 = **4.25ms**
  - **Scenario B**: (4ms + 7ms + 10ms + 11ms) / 4 = **8ms**



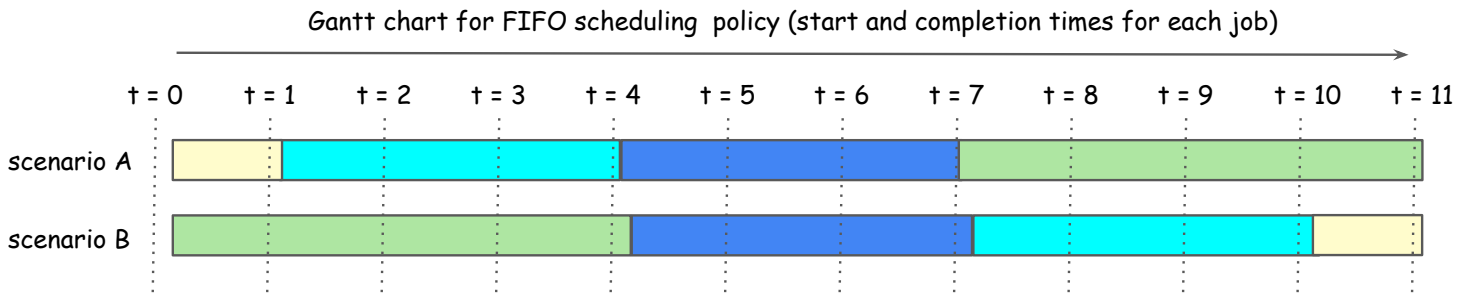
Calculating completion time

> Average

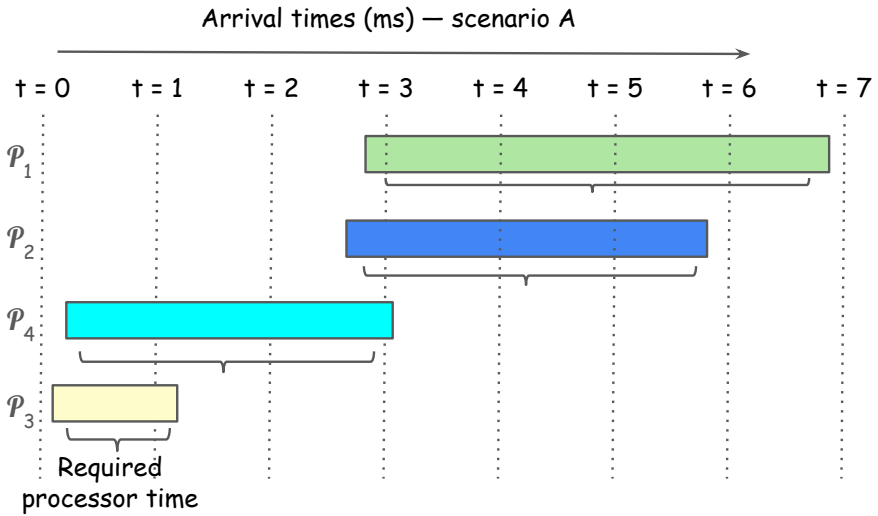
- **Scenario A**: (1ms + 4ms + 4ms + 8ms) / 4 = **4.25ms**
- **Scenario B**: (4ms + 7ms + 10ms + 11ms) / 4 = **8ms**

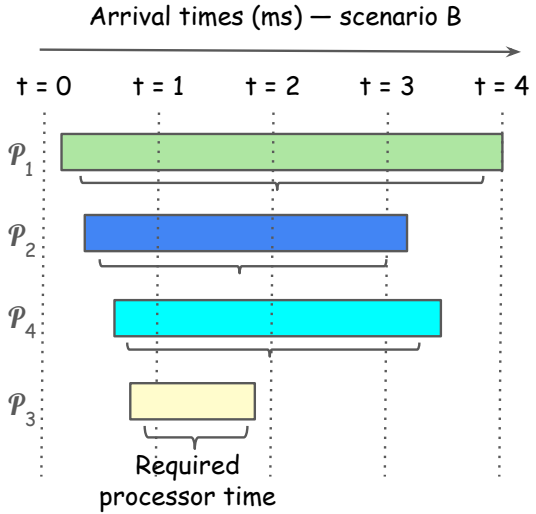
> Completion time (worst)

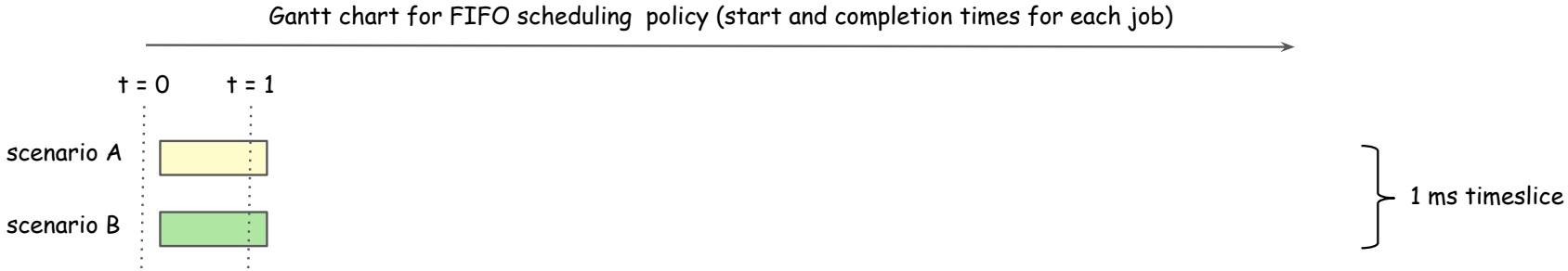
- **Scenario A**: (1ms + 4ms + 4ms + **8ms**) => **8ms**
- **Scenario B**: (4ms + 7ms + 10ms + **11ms**) => **11ms**



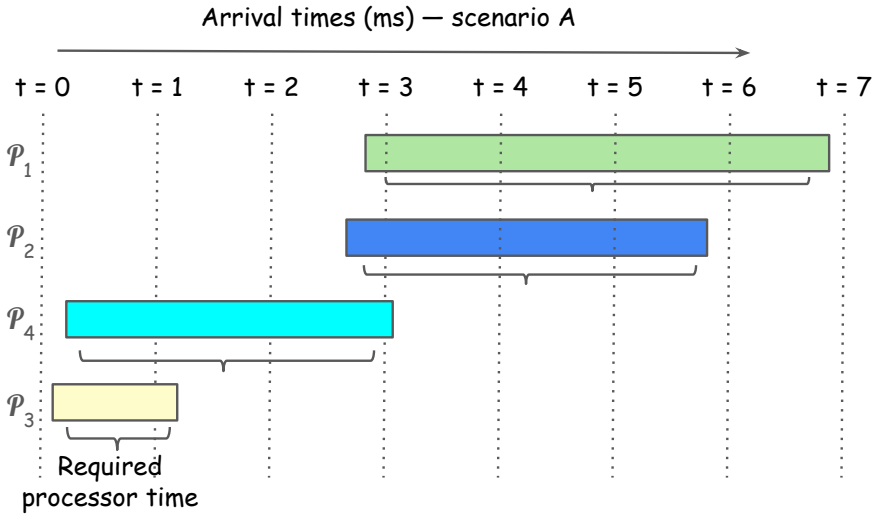
Round Robin scheduling policy (SCHED_RR)

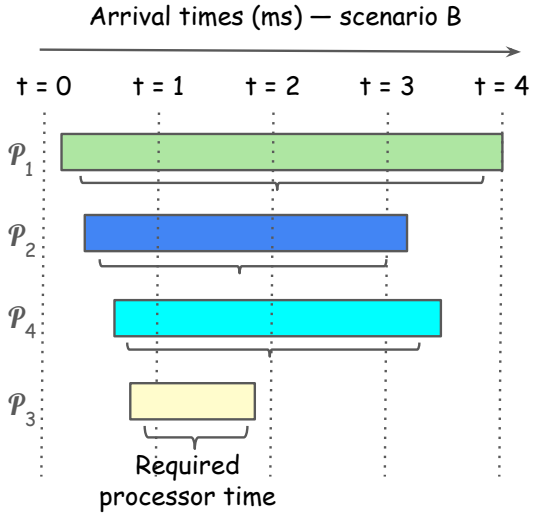


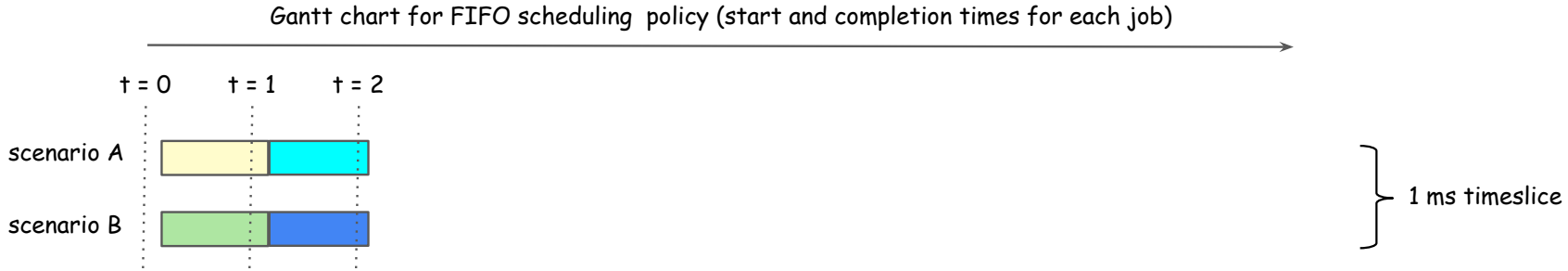




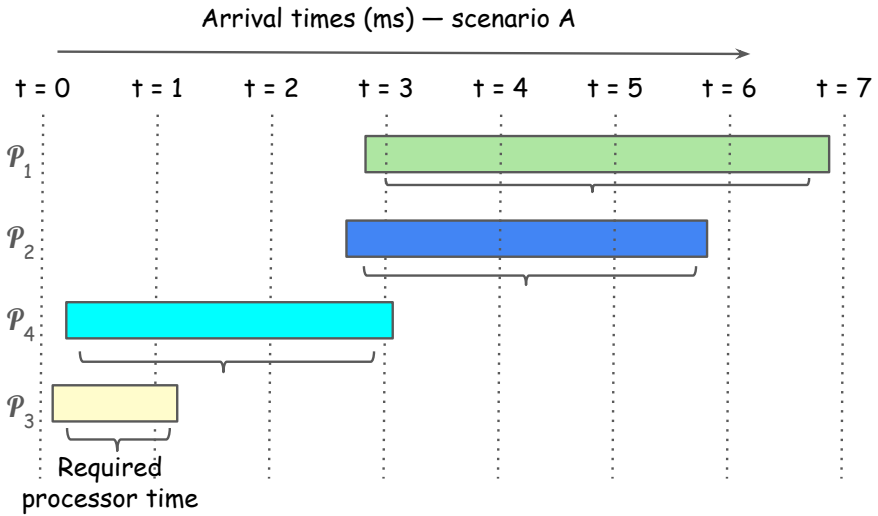
Round Robin scheduling policy (SCHED_RR)

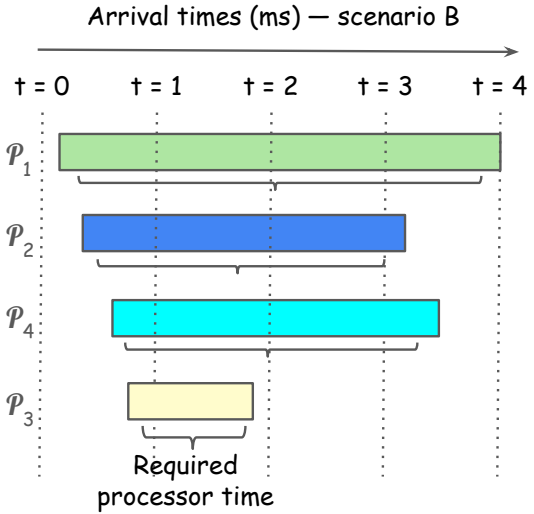


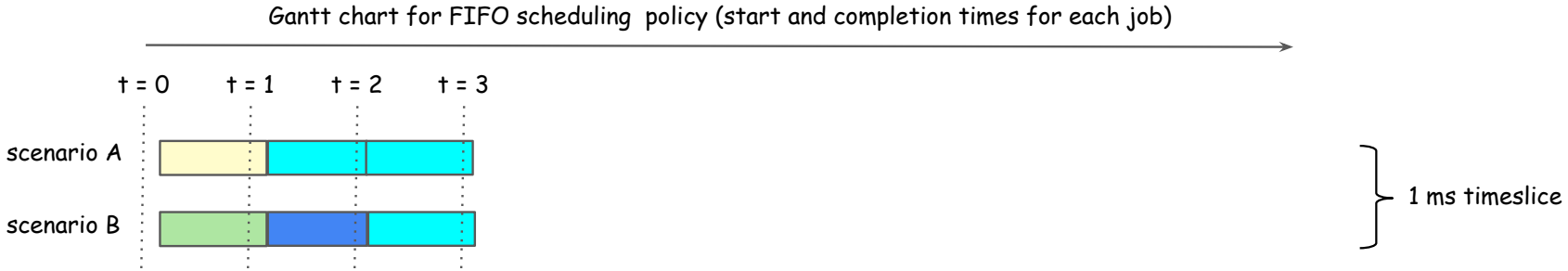




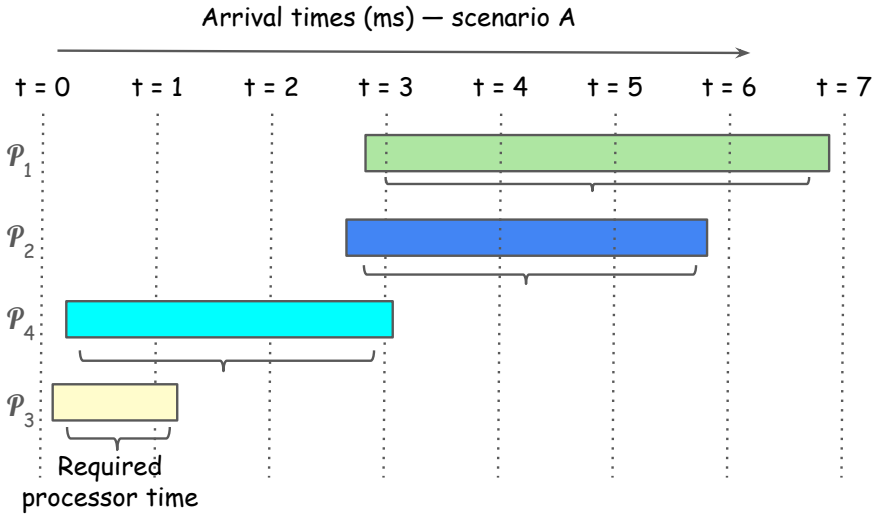
Round Robin scheduling policy (SCHED_RR)

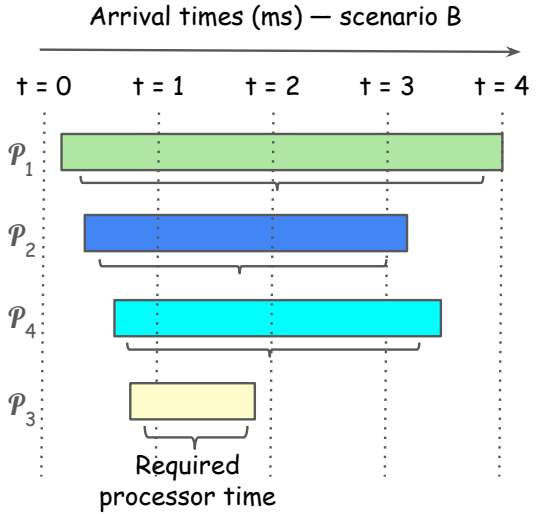


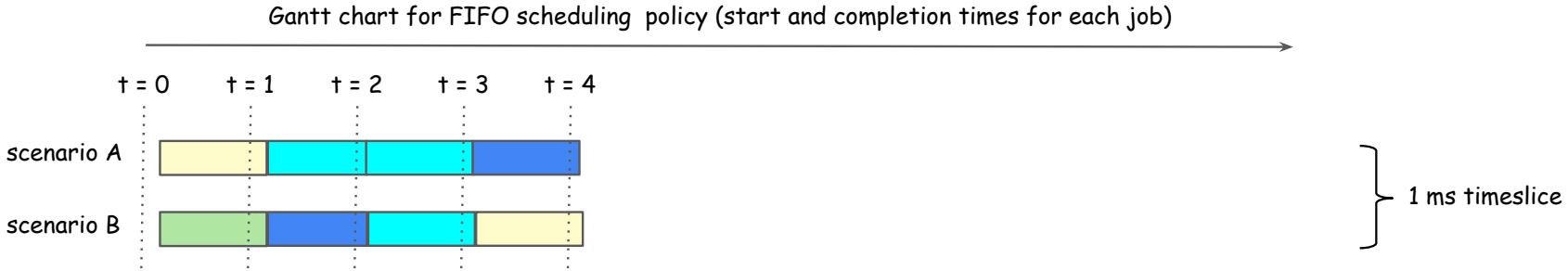




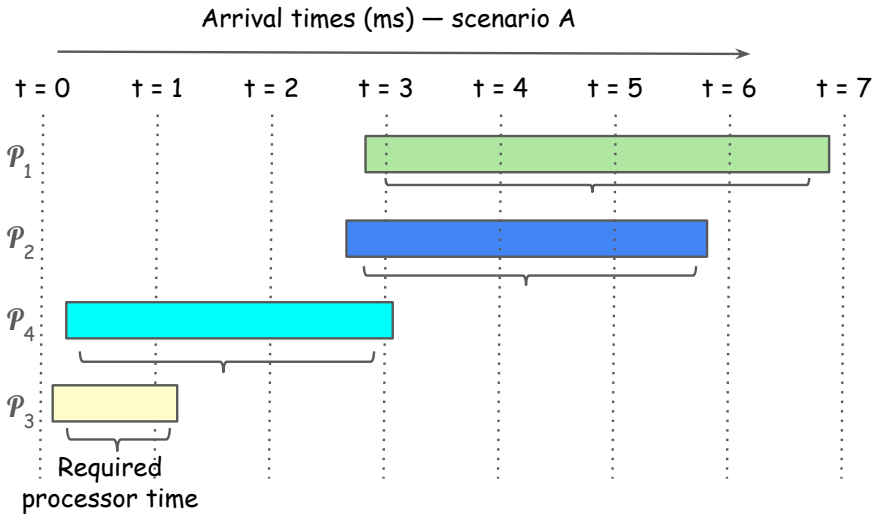
Round Robin scheduling policy (SCHED_RR)

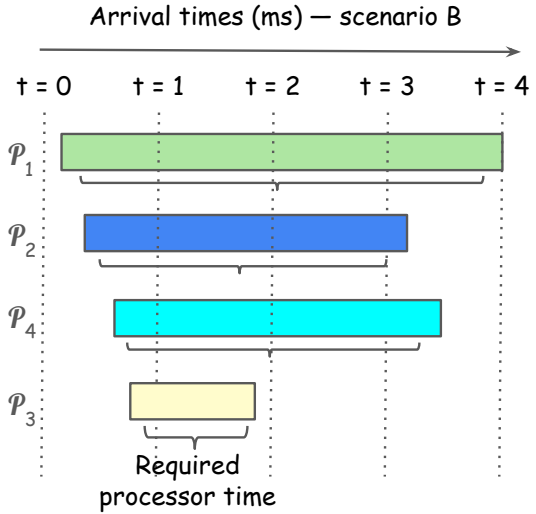


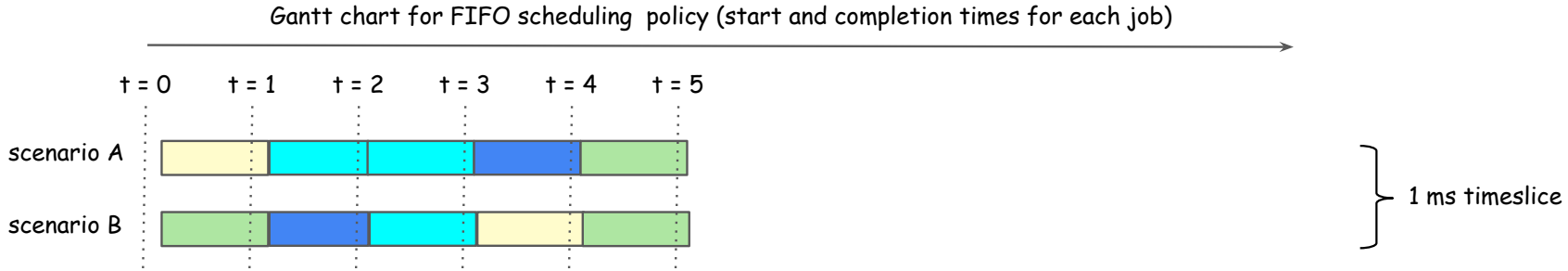




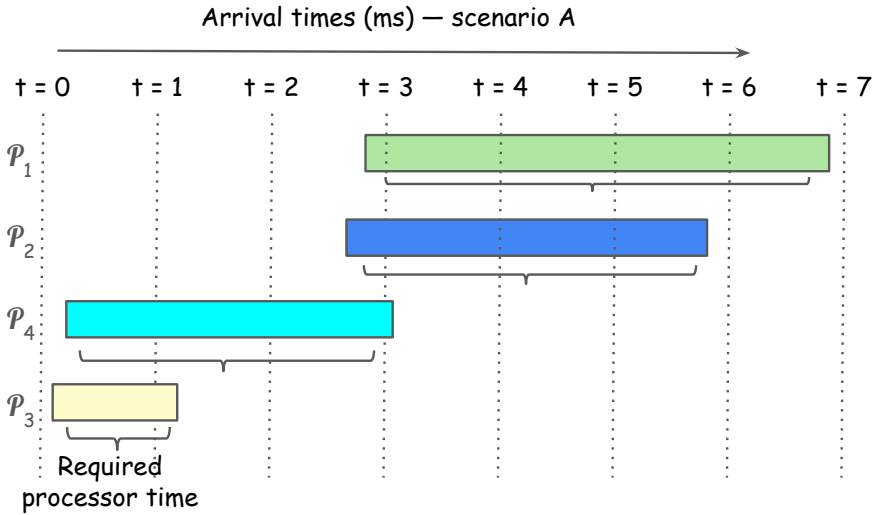
Round Robin scheduling policy (SCHED_RR)

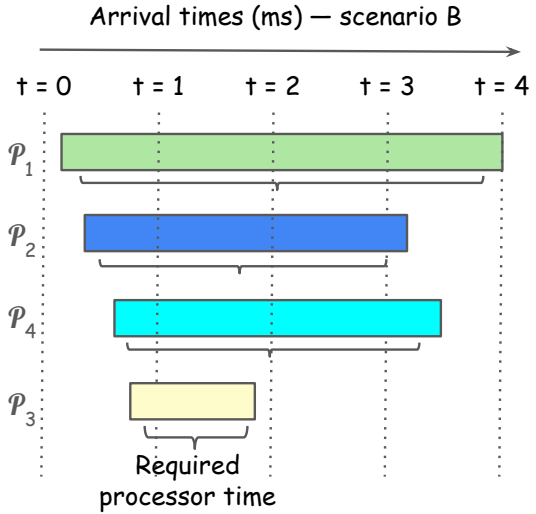


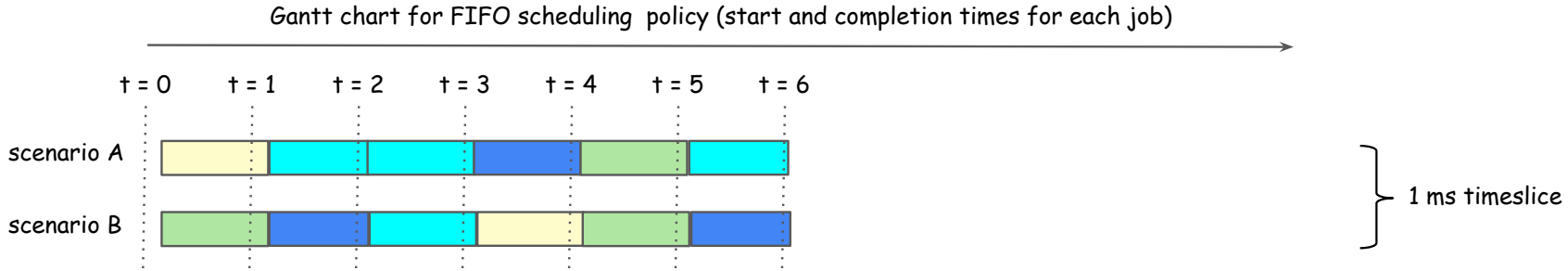




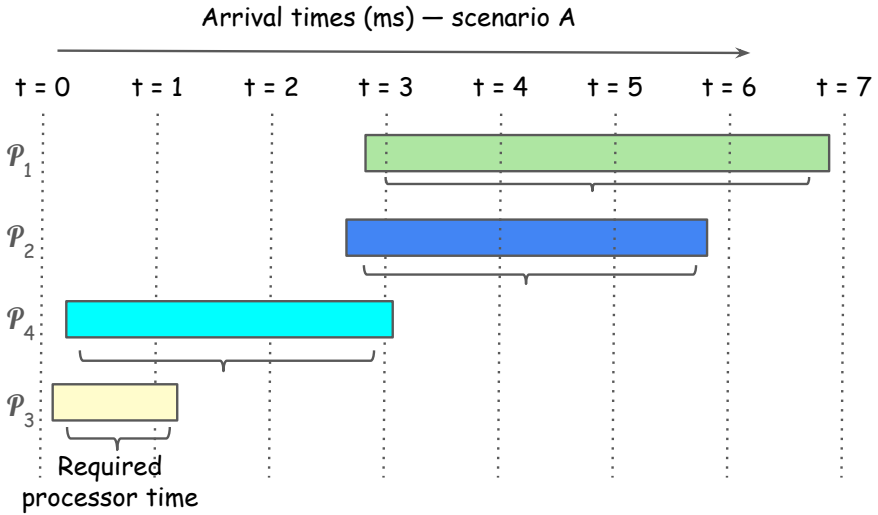
Round Robin scheduling policy (SCHED_RR)

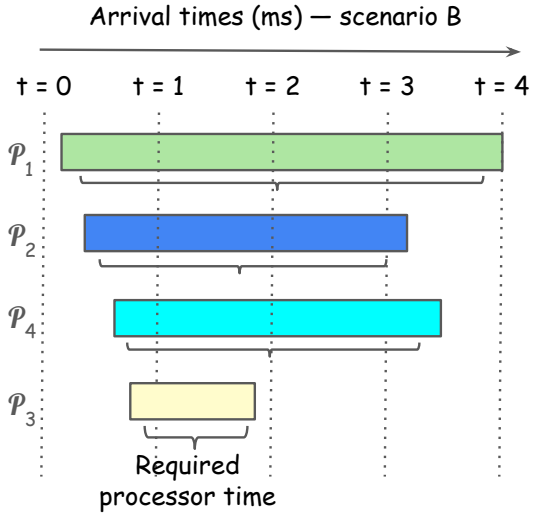


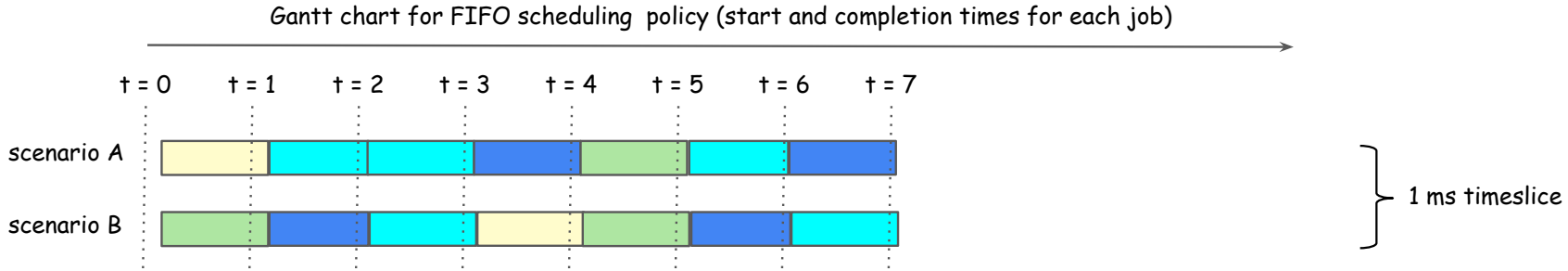




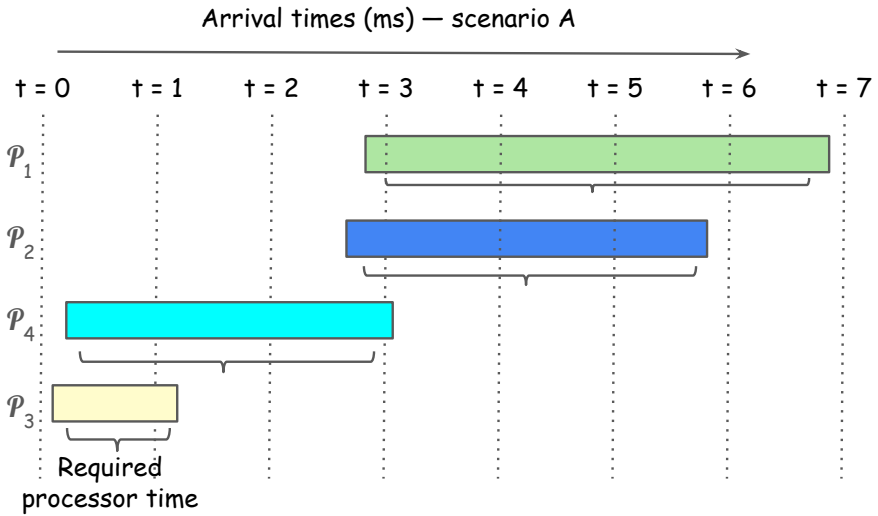
Round Robin scheduling policy (SCHED_RR)

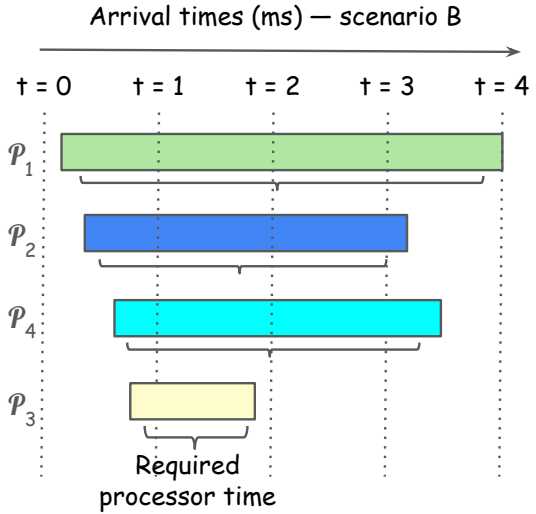


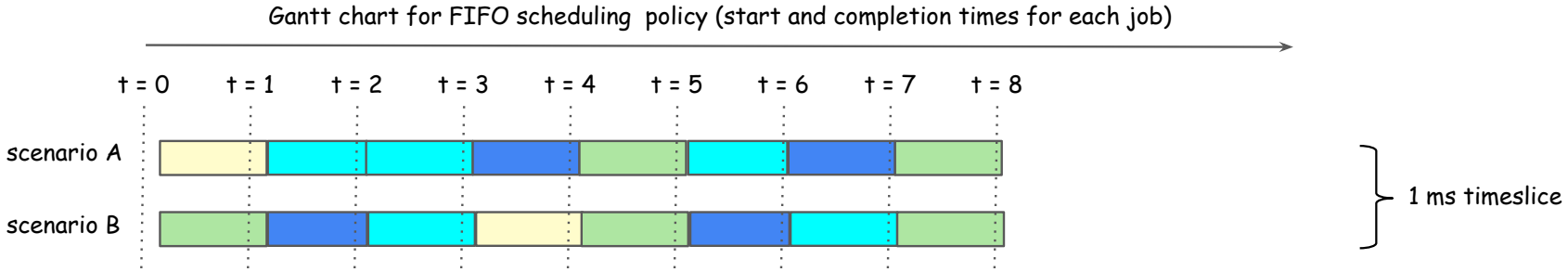




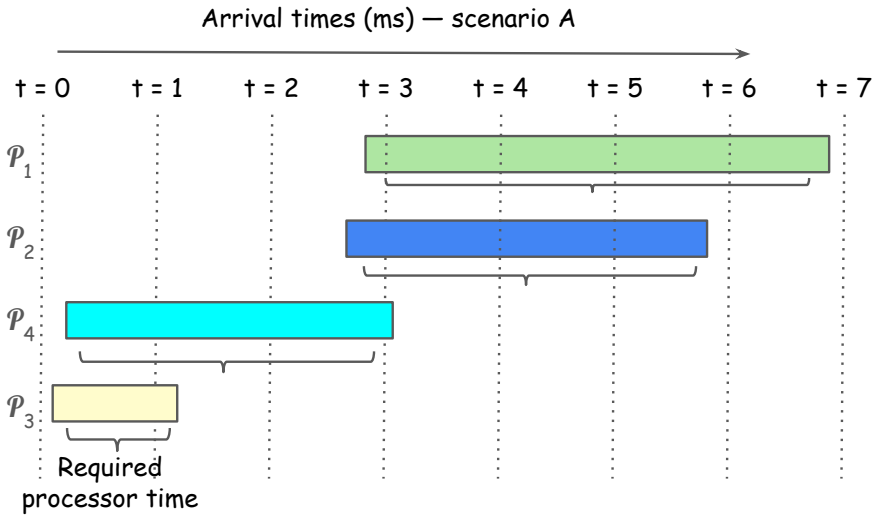
Round Robin scheduling policy (SCHED_RR)

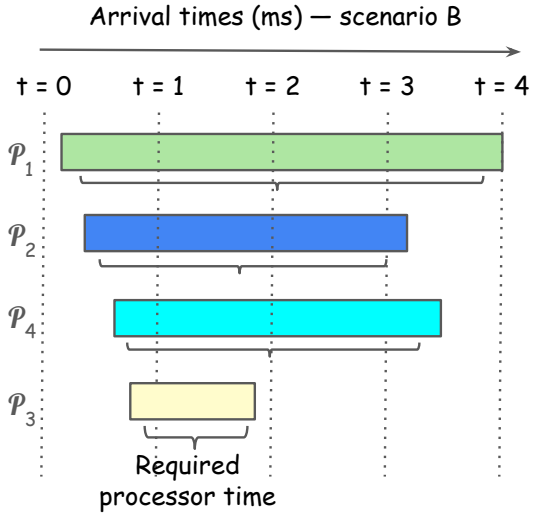


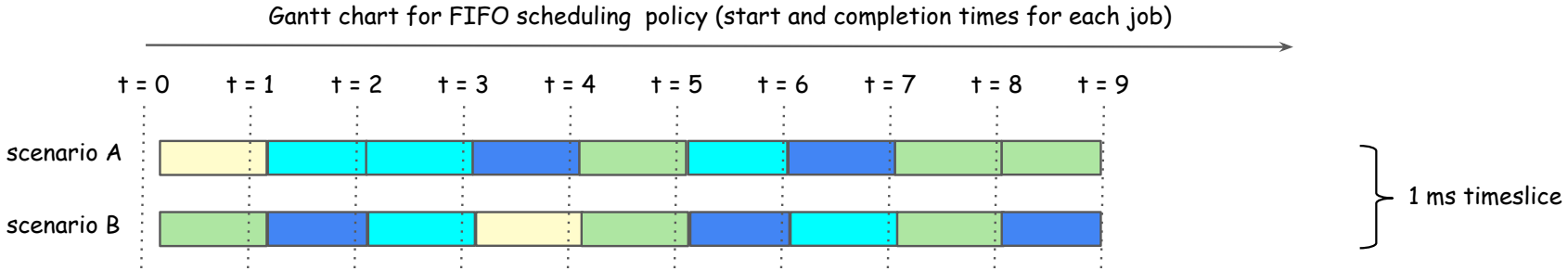




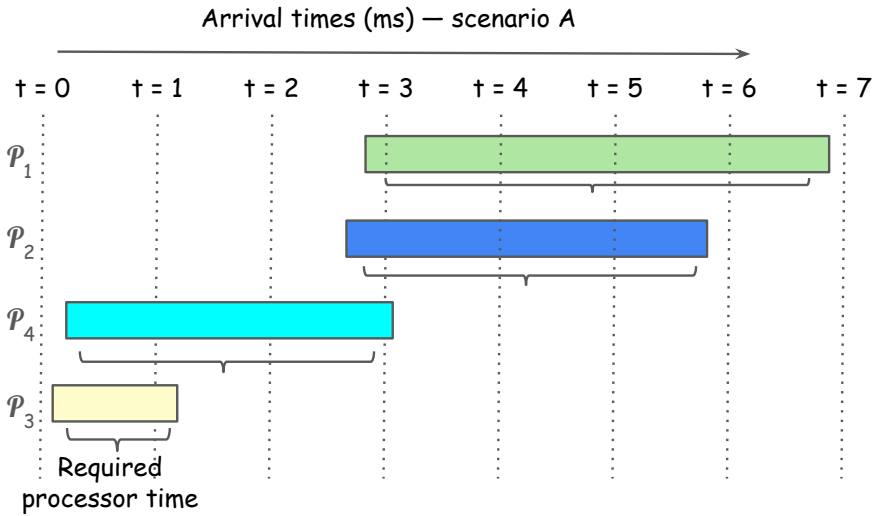
Round Robin scheduling policy (SCHED_RR)

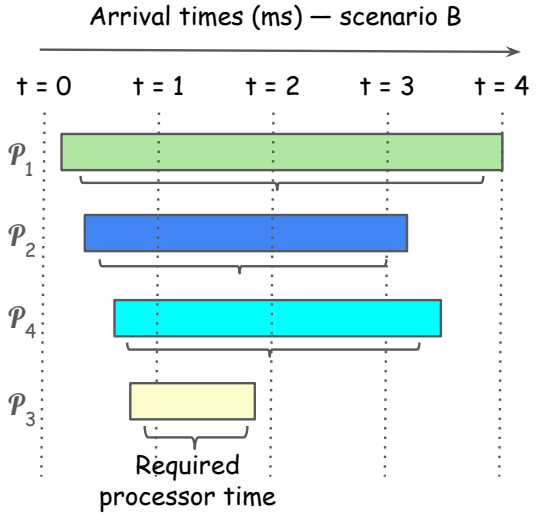


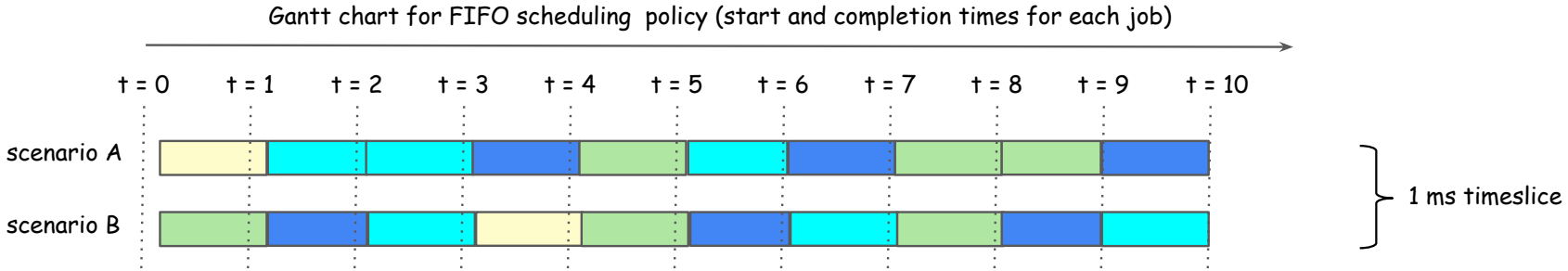




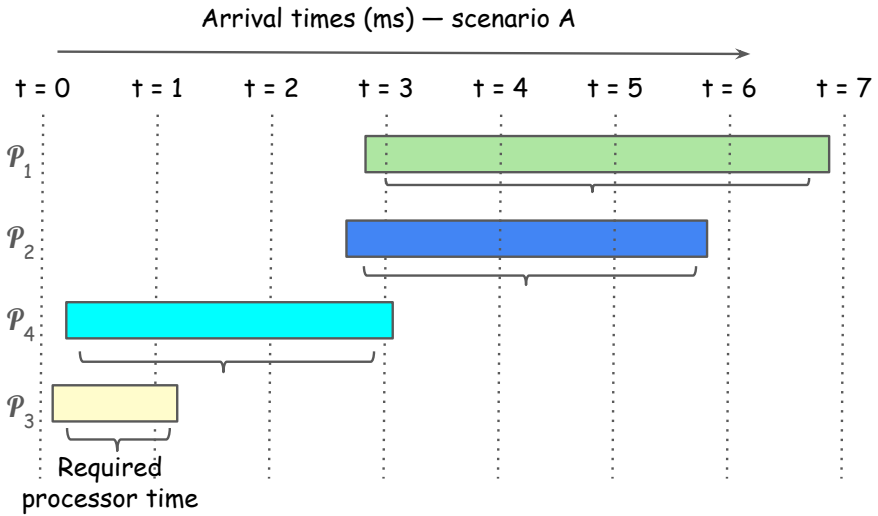
Round Robin scheduling policy (SCHED_RR)

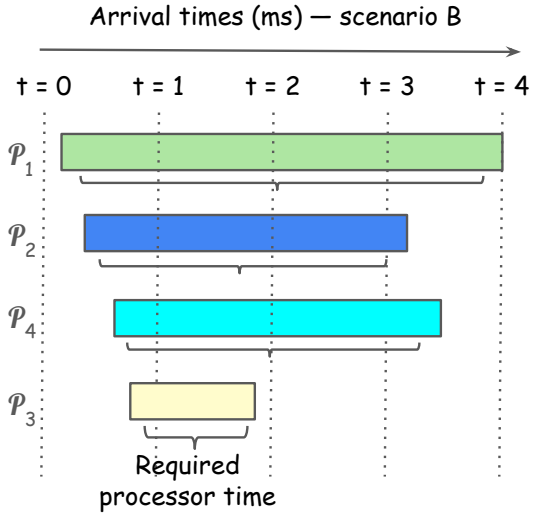


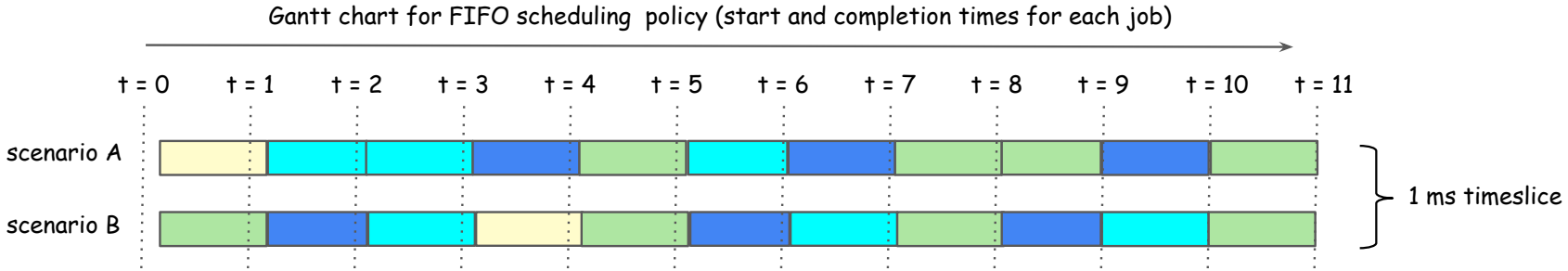




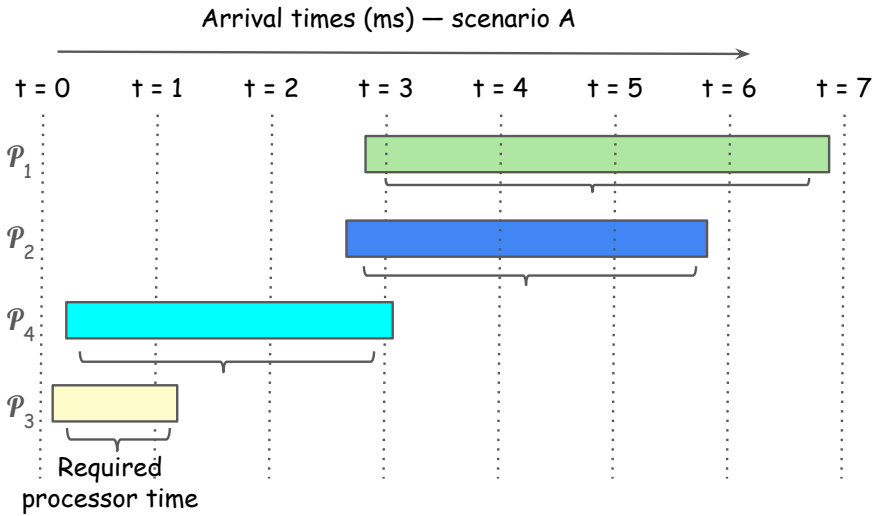
Round Robin scheduling policy (SCHED_RR)

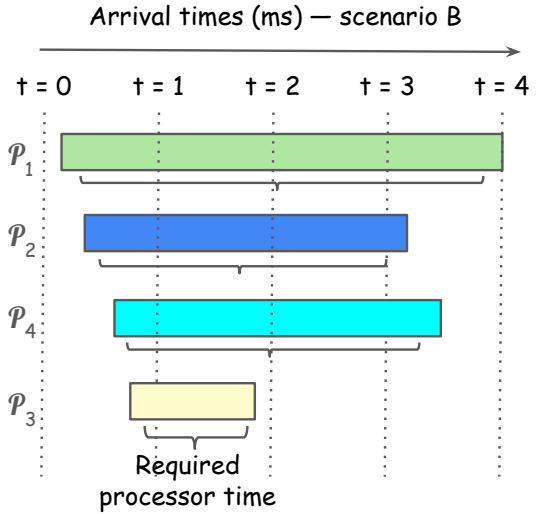


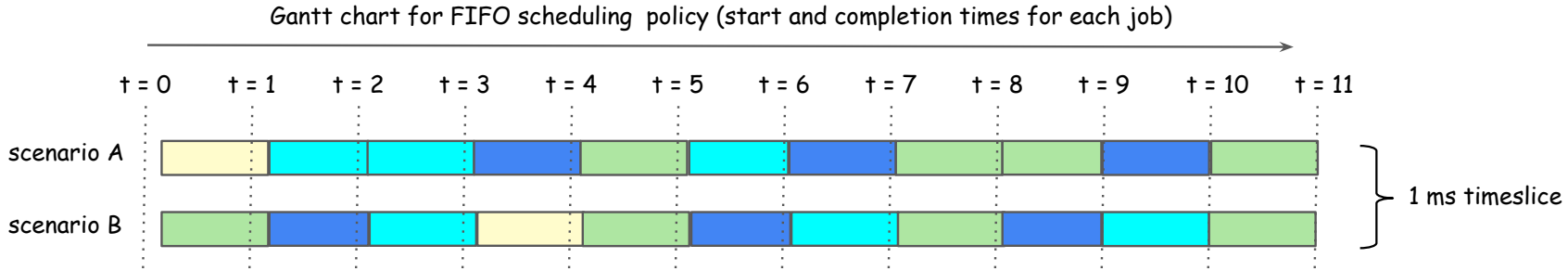




Calculating response time (latency)







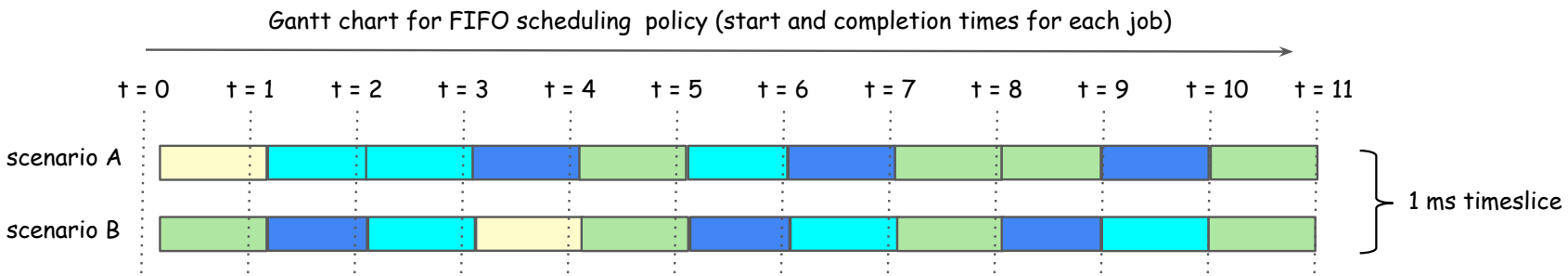
Calculating response time (latency)

› Average

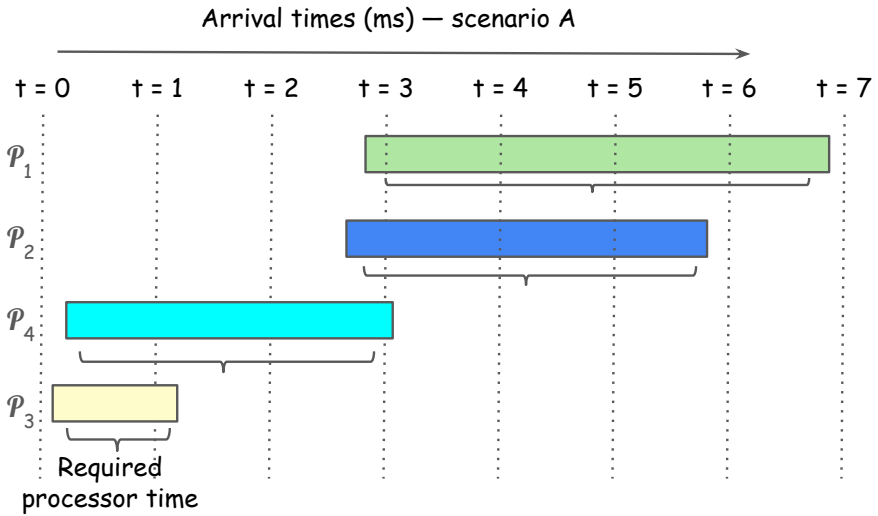
- **Scenario A**: (0ms + 1ms + 0ms + 1ms) / 4 = **0.5ms**
- **Scenario B**: (0ms + 1ms + 2ms + 3ms) / 4 = **1.5ms**

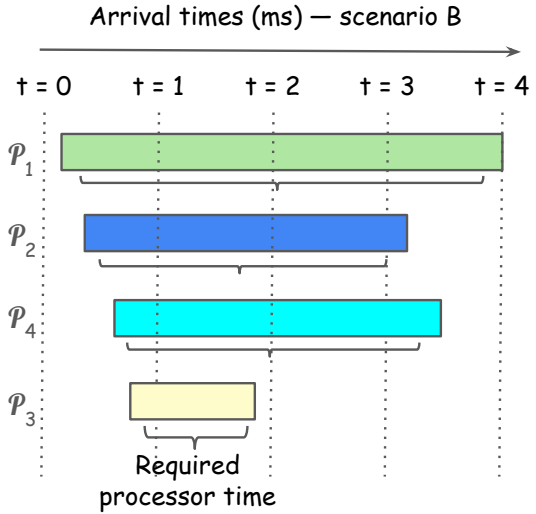
› Worst case

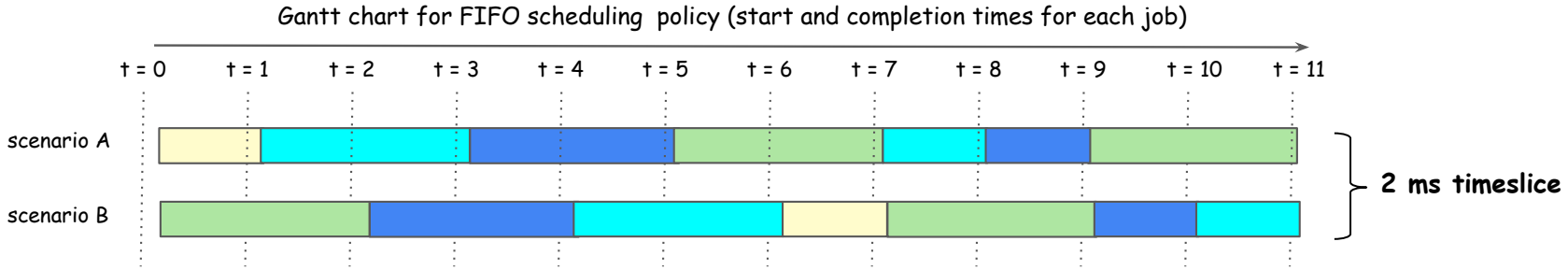
- **Scenario A**: (0ms + 1ms + 0ms + **1ms**) => **1ms**
- **Scenario B**: (0ms + 1ms + 2ms + **3ms**) => **3ms**



Calculating response time (latency)







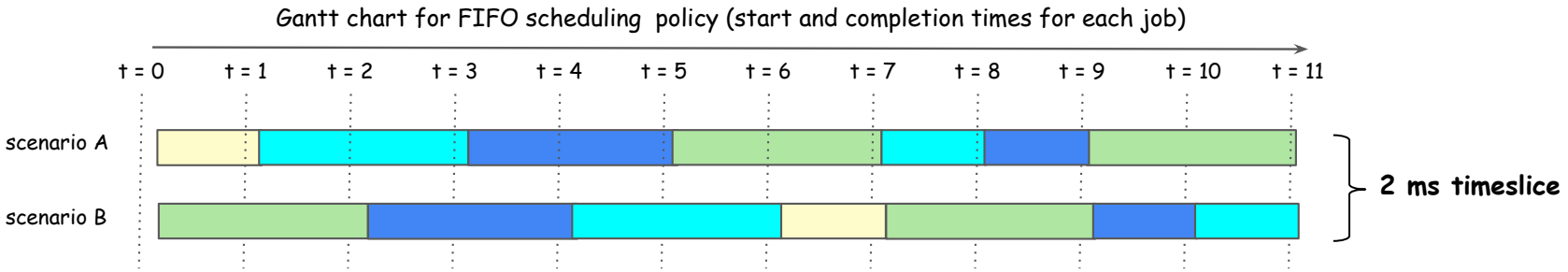
Calculating response time (latency)

› Average

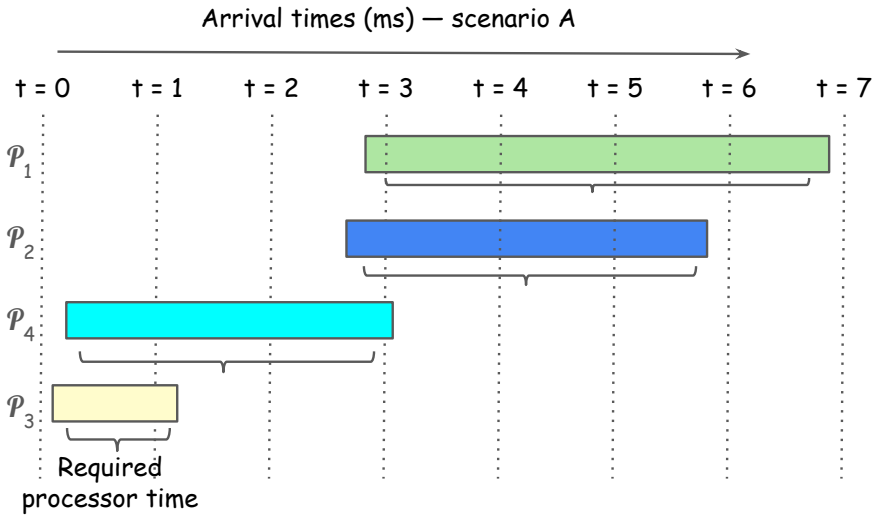
- **Scenario A**: (0ms + 1ms + 0ms + 3ms) / 4 = **1ms**
- **Scenario B**: (0ms + 2ms + 4ms + 6ms) / 4 = **3ms**

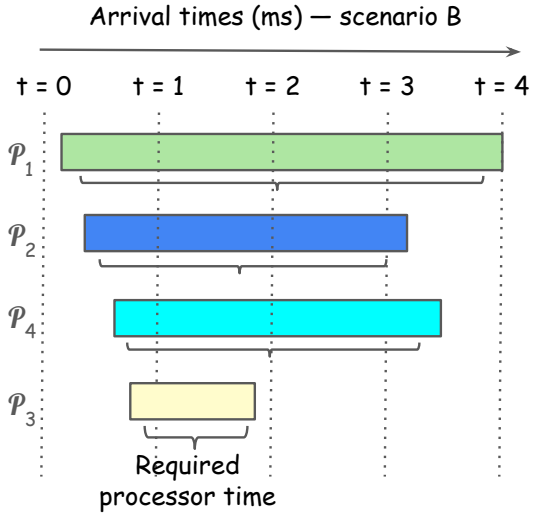
› Worst case

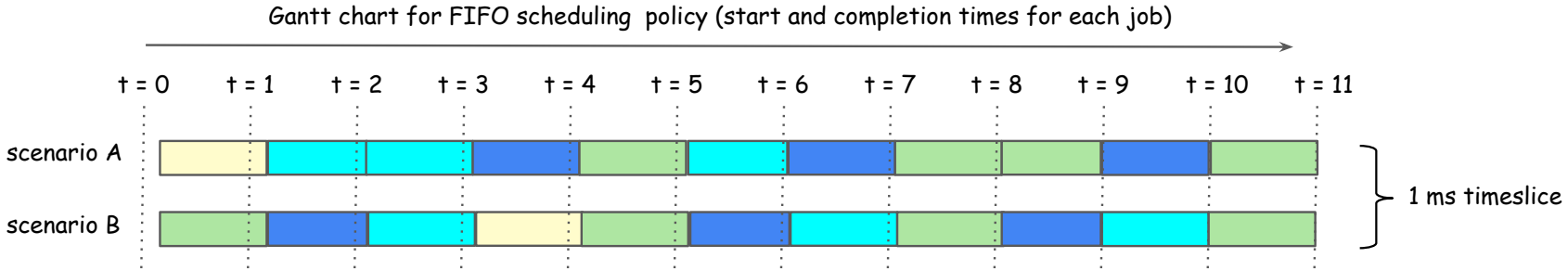
- **Scenario A**: (0ms + 1ms + 0ms + **3ms**) => **3ms**
- **Scenario B**: (0ms + 2ms + 4ms + **6ms**) => **6ms**



Calculating completion time







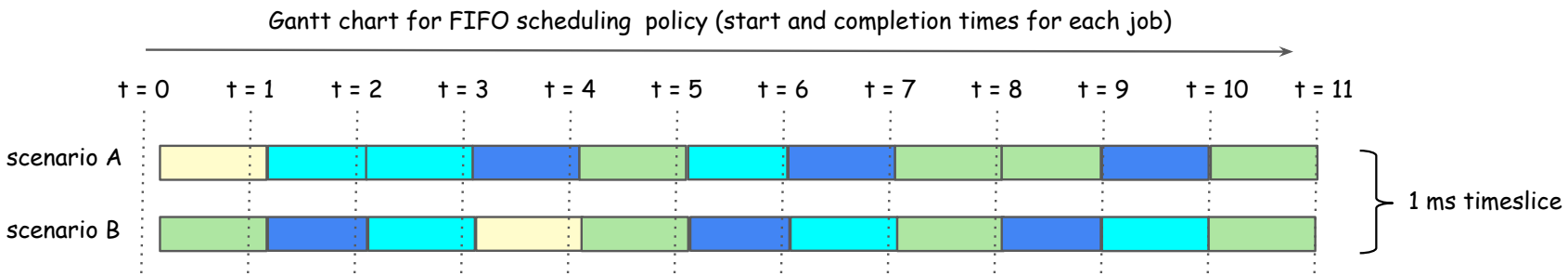
Calculating completion time

› Average

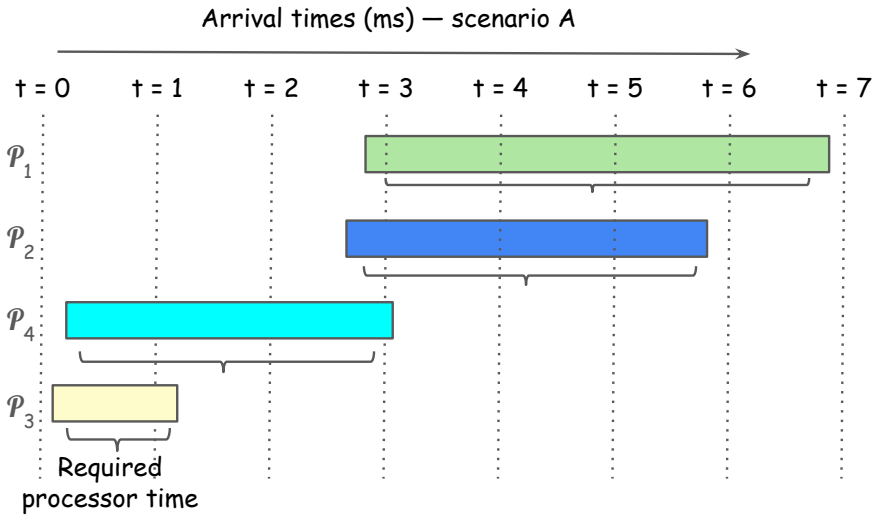
- **Scenario A**: (1ms + 6ms + 7ms + 8ms) / 4 = **5.5ms**
- **Scenario B**: (11ms + 10ms + 9ms + 4ms) / 4 = **8.5ms**

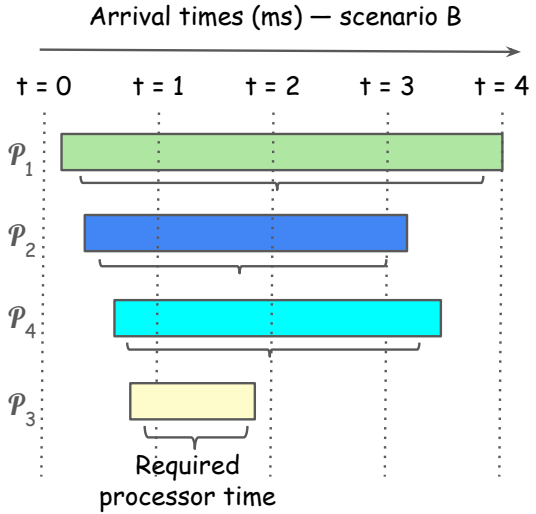
› Worst case

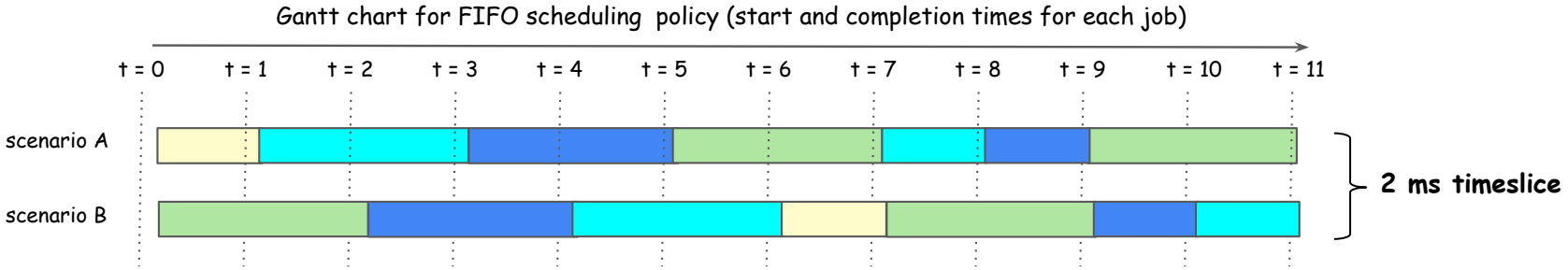
- **Scenario A**: (1ms + 6ms + 7ms + **8ms**) => **8ms**
- **Scenario B**: (**11ms** + 10ms + 9ms + 4ms) => **11ms**



Calculating completion time







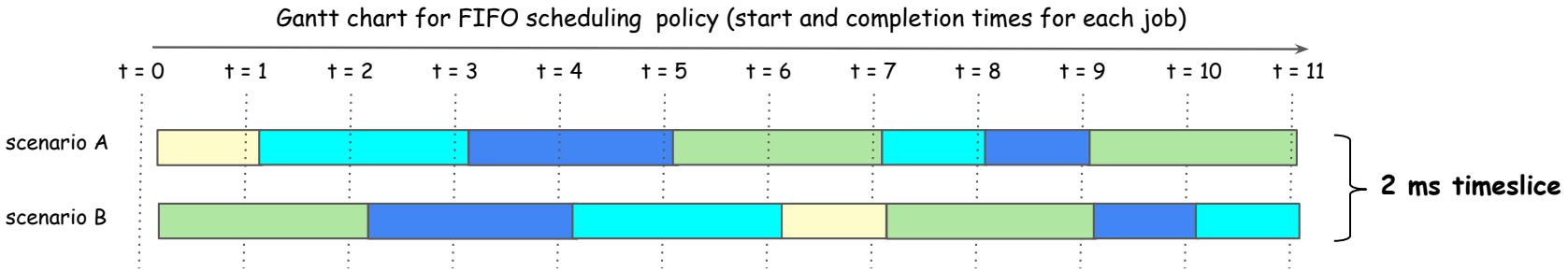
Calculating completion time

› Average

- **Scenario A**: (1ms + 3ms + 6ms + 8ms) / 4 = **4.5ms**
- **Scenario B**: (11ms + 10ms + 9ms + 7ms) / 4 = **9.25ms**

› Worst case

- **Scenario A**: (1ms + 3ms + 6ms + **8ms**) => **8ms**
- **Scenario B**: (**11ms** + 10ms + 9ms + 7ms) => **11ms**



Comparative view

- Response time
  - FIFO
    - Average: 1.5ms (sc.-A) vs. 5.25ms (sc.-B)
    - Worst: 4ms (sc.-A) vs. 10ms (sc.-B)
  - RR
    - Average: 0.5ms (sc.-A) vs. 1.5ms (sc.-B) / 1ms (sc.-A) vs. 3ms (sc.-B)
    - Worst: 1ms (sc.-A) vs. 3ms (sc.-B) / 3ms (sc.-A) vs. 6ms (sc.-B)

Comparative view

> Response time

>> FIFO

- Too much dependency on arrival times
- One long task may delay everyone (convoy effect)

>> RR

- Good pick when response time matters
- Dependency on the size of time slice => Need to balance number of ctx switches

Comparative view

> Response time

>> FIFO

- Too much dependency on arrival times
- One long task may delay everyone (convoy effect)

>> RR

- Good pick when response time matters => Need small timeslice
- Small time slice => Too many ctx switches / Large timeslice => Becomes FIFO

> Completion time

>> FIFO

- Average: 4.25ms (sc.-A) vs. 8ms (sc.-B)
- Worst: 8ms (sc.-A) vs. 11ms (sc.-B)

>> RR

- Average: 5.5ms (sc.-A) vs. 8.5ms (sc.-B) / 4.5ms (sc.-A) vs. 9.25ms (sc.-B)
- Worst: 8ms (sc.-A) vs. 11ms (sc.-B) / 8ms (sc.-A) vs. 11ms (sc.-B)

Comparative view

> Response time

>> FIFO

- Too much dependency on arrival times
- One long task may delay everyone (convoy effect)

>> RR

- Good pick when response time matters => Need small timeslice
- Small time slice => Too many ctx switches / Large timeslice => Becomes FIFO

> Completion time

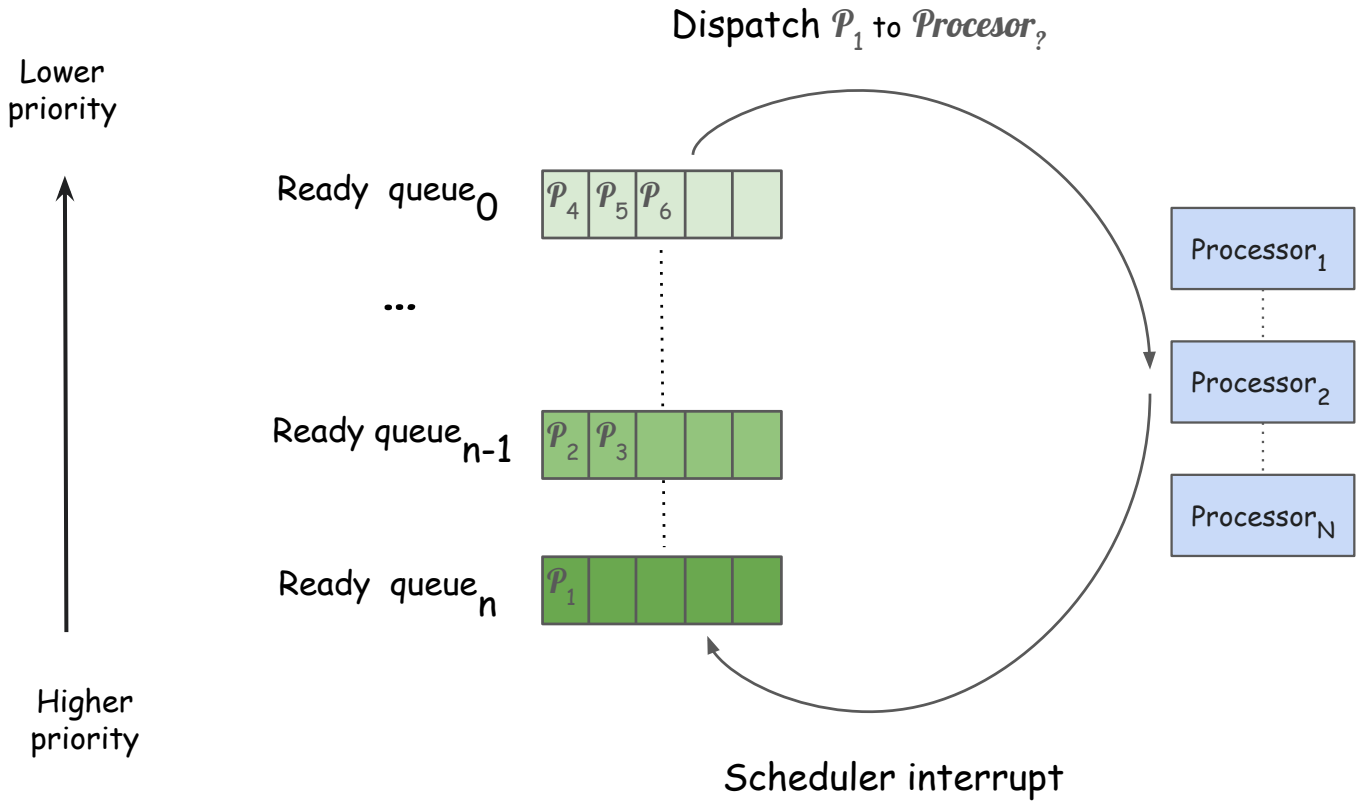
>> FIFO

- Good pick when completion time matters

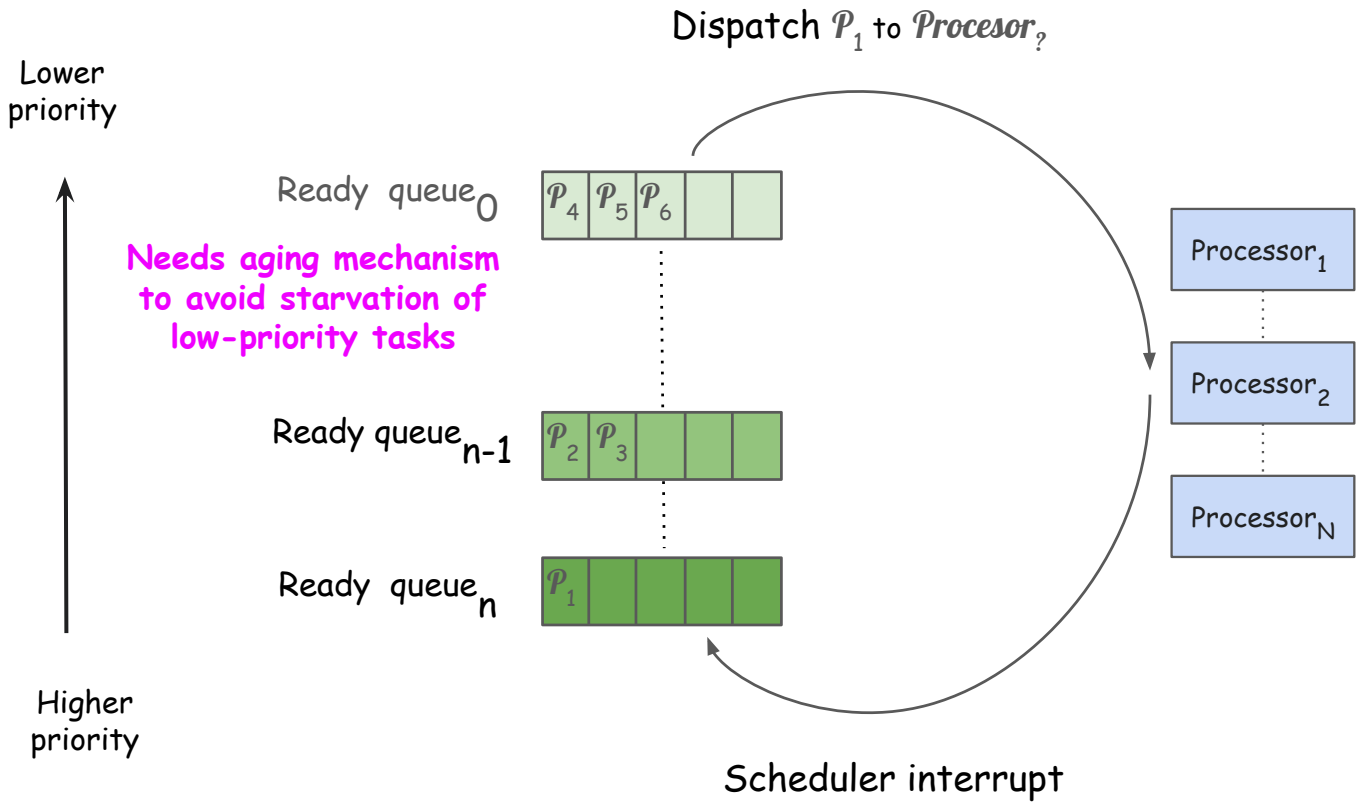
>> RR

- Small timeslice => Bad pick for longer tasks
- Need to balance number of ctx switches

Hierarchical priority-based scheduling



Hierarchical priority-based scheduling

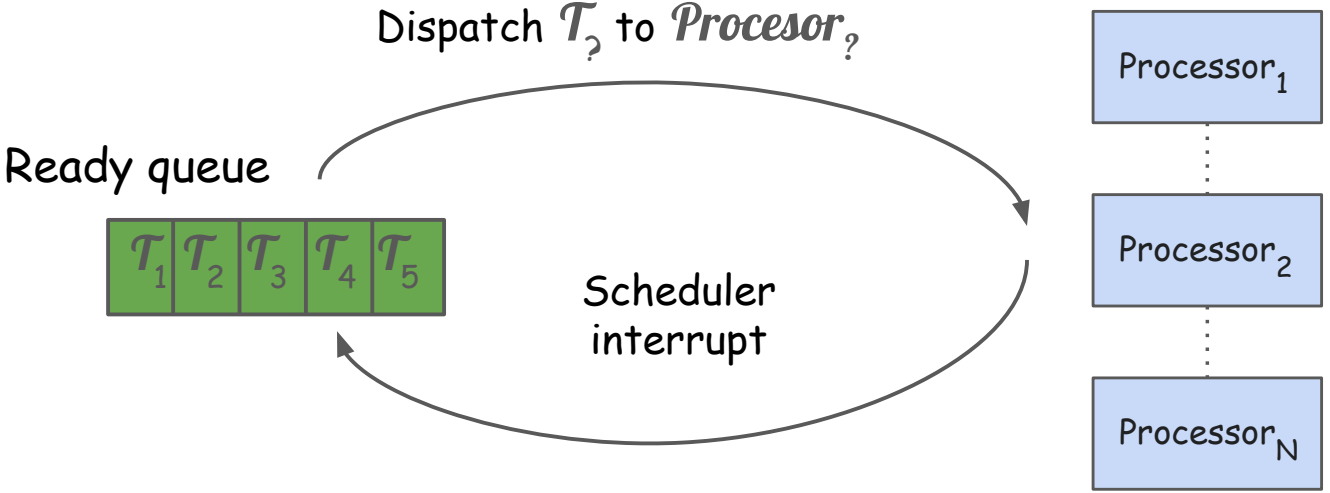


The Linux scheduler

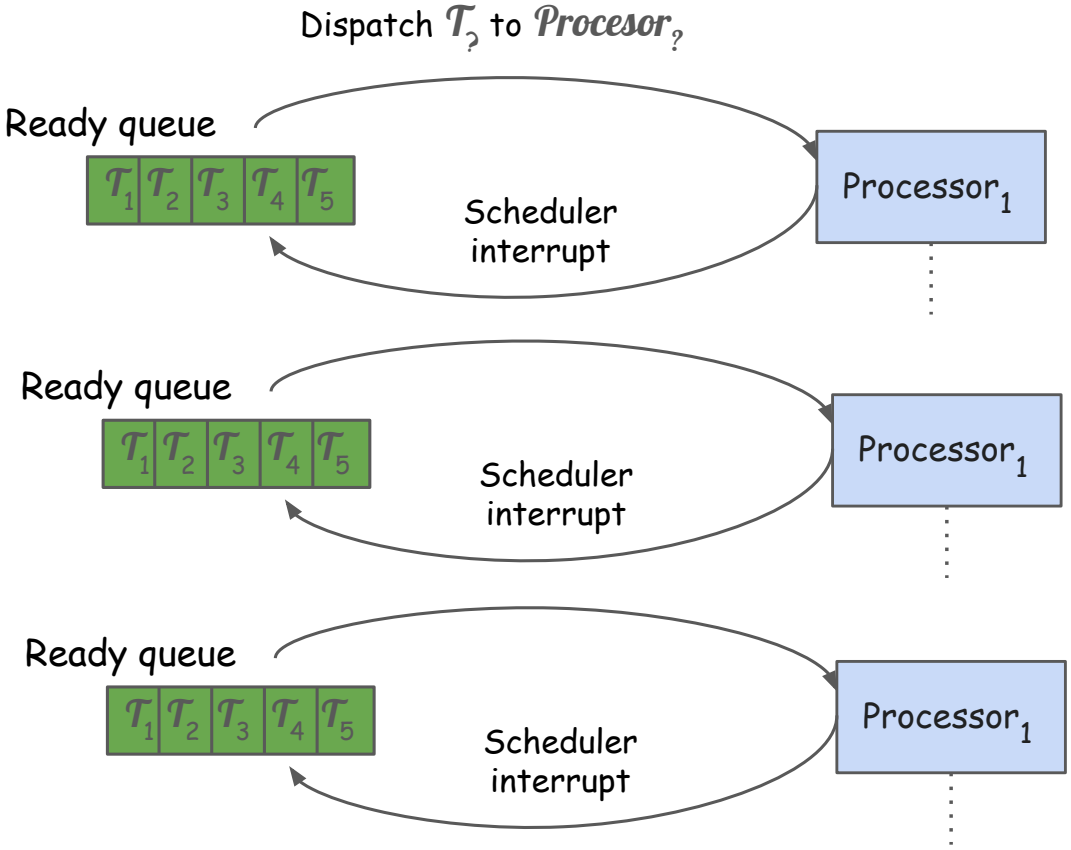
Precedence Order	Scheduler class	Implemented policies	Usecase	POSIX compliance
1	stop_sched_class	Run Linux kernel-internal tasks	Only used internally by the kernel; preempts anything running in the local processor	No
2	dl_sched_class	SCHED_DEADLINE	Hard real-time tasks whose execution deadlines must be met	No
3	rt_sched_class	SCHED_FIFO, SCHED_RR	Soft real-time tasks (e.g., audio daemon) with priorities [1–99]	Yes
4	cfs_sched_class, eevdf_sched_class	SCHED_NORMAL, SCHED_BATCH, SCHED_IDLE	User tasks with "nice" values in the range [-20–19]	Partially Yes
5	idle_sched_class	Run the Linux kernel "idle" task	Runs when the local processor is idle, and has no other task to run	No

Unicore scheduling

> Given k tasks ready to run in a system with N available processors, which task should be dispatched to which processor at any given point in time?



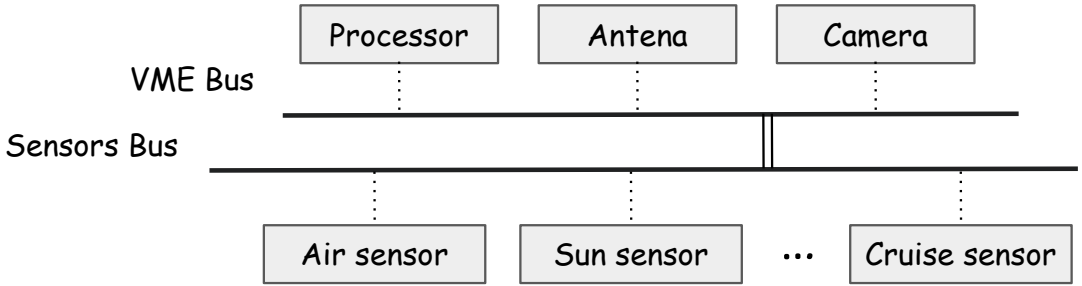
Multicore scheduling



Ready for your first bug in the outer universe

Mars Rover: software and hardware

- Works Real-Time Operating System (RTOS)
  - Tasks must meet strict timing constraints
  - Preemptive with priority-based scheduling
  - Scheduler ticks at 8 Hz (i.e., every 125ms)
- Hardware overview



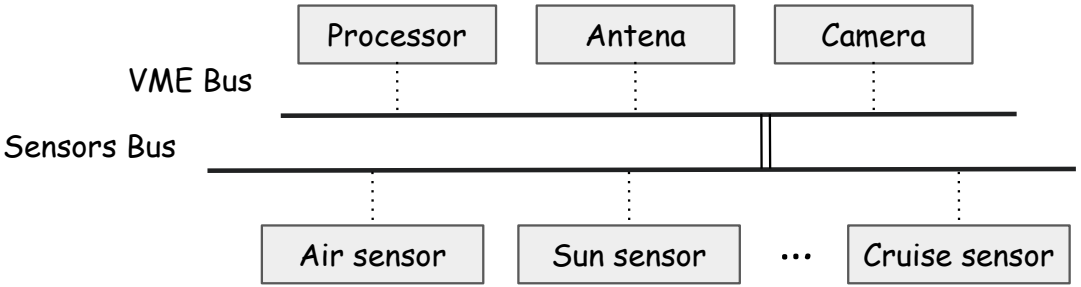
Mars Rover: software and hardware

- **Works Real-Time Operating System (RTOS)**
  - Tasks must meet strict timing constraints
  - Preemptive with priority-based scheduling
  - Scheduler ticks at 8 Hz (i.e., every 125ms)
- **Hardware overview**
  - Data from sensor bus to the VMA bus (to antenna)
  - Processor signal from VMA bus to sensor bus (cruise)

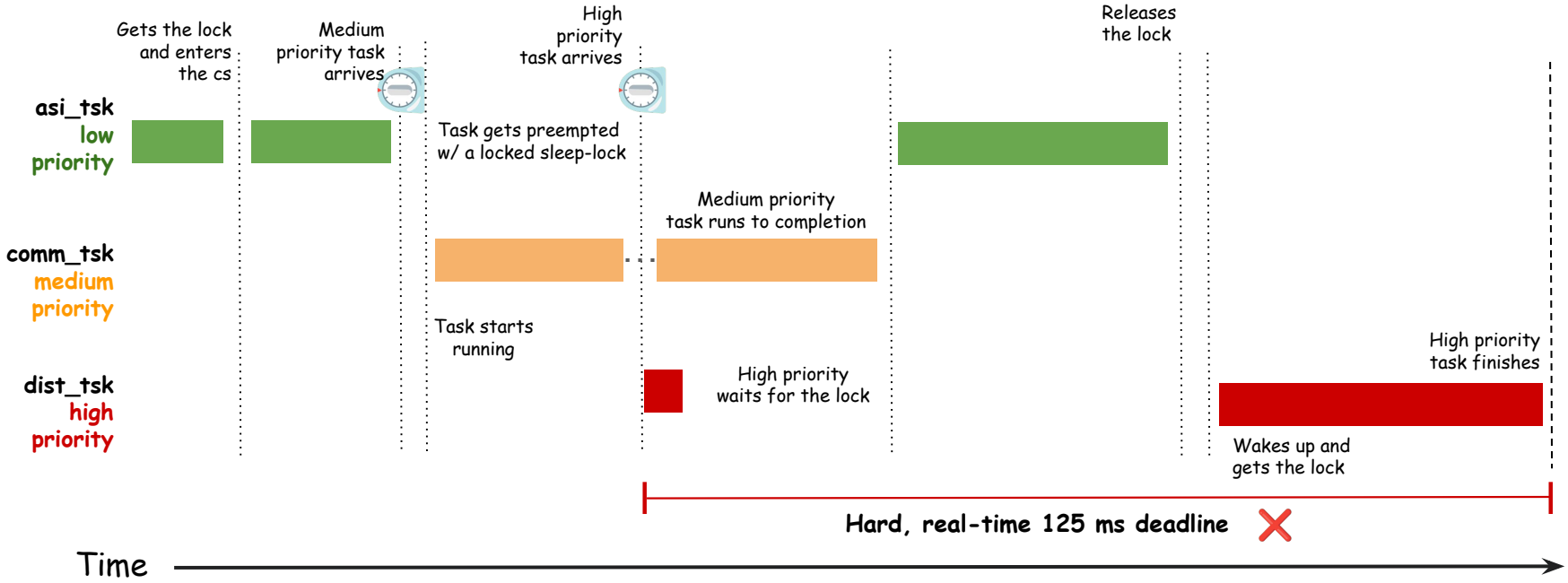
Mars Rover: software and hardware

- **Synchronization**
  - sched_tsk: Decides who transmits data next
  - dist_tsk: Decides who receives data next
  - comm_tsk: Uses the antenna to transmit data to Earth
  - asi_tsk: Uses the air sensor for scientific computations

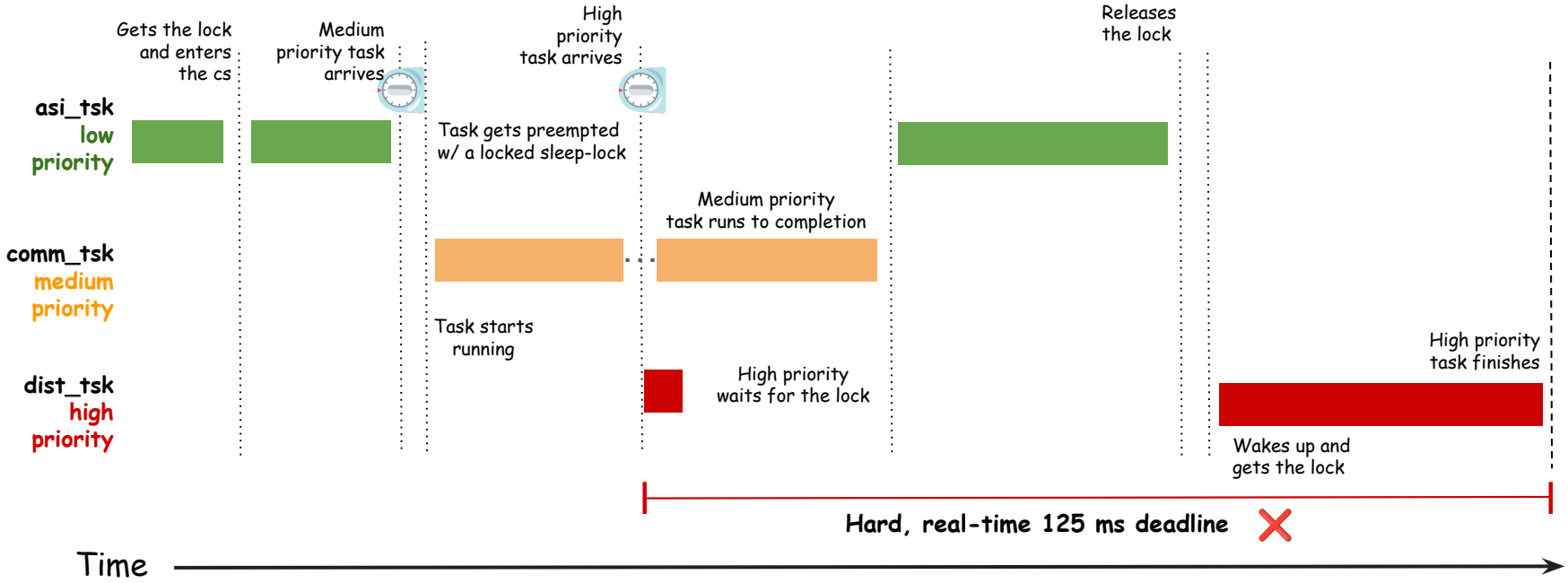
Priorities: sched_tsk > dist_tsk (high) > comm_tsk (medium) > asi_tsk (low)



Classic example of priority inversion bug



Solution?


Gets the lock and enters the cs
Medium priority task arrives
High priority task arrives
Releases the lock
asi_tsk
low priority
Task gets preempted w/ a locked sleep-lock
Medium priority task runs to completion
comm_tsk
medium priority
Task starts running
High priority task finishes
dist_tsk
high priority
High priority waits for the lock
Wakes up and gets the lock
Hard, real-time 125 ms deadline
Time