K22 - Operating Systems: Design Principles and Internals

Fall 2025 @dit

Vaggelis Atlidakis

Lecture 13

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

```c
/*
 * test_k22tree.c
 *
 * Υλοποίηση του userspace προγράμματος που:
 *  1. Κάνει iterative doubling μέχρι να χωρέσει όλο το δέντρο διεργασιών
 *  2. Καλεί το k22tree syscall
 *  3. Τυπώνει το αποτέλεσμα σε δενδρική μορφή (pre-order DFS)
 *  4. Χρησιμοποιεί stack για να υπολογίσει depth
 *
 * Τα σχόλια είναι αναλυτικά και γραμμένα ώστε να φαίνεται ξεκάθαρα
 * ότι η εργασία έγινε από εμένα.
 */

#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>

#include <linux/k22info.h>

#ifndef __NR_k22tree
#define __NR_k22tree 467
#endif
```

Programming assignment #1

> Reasons for teams that got zero?

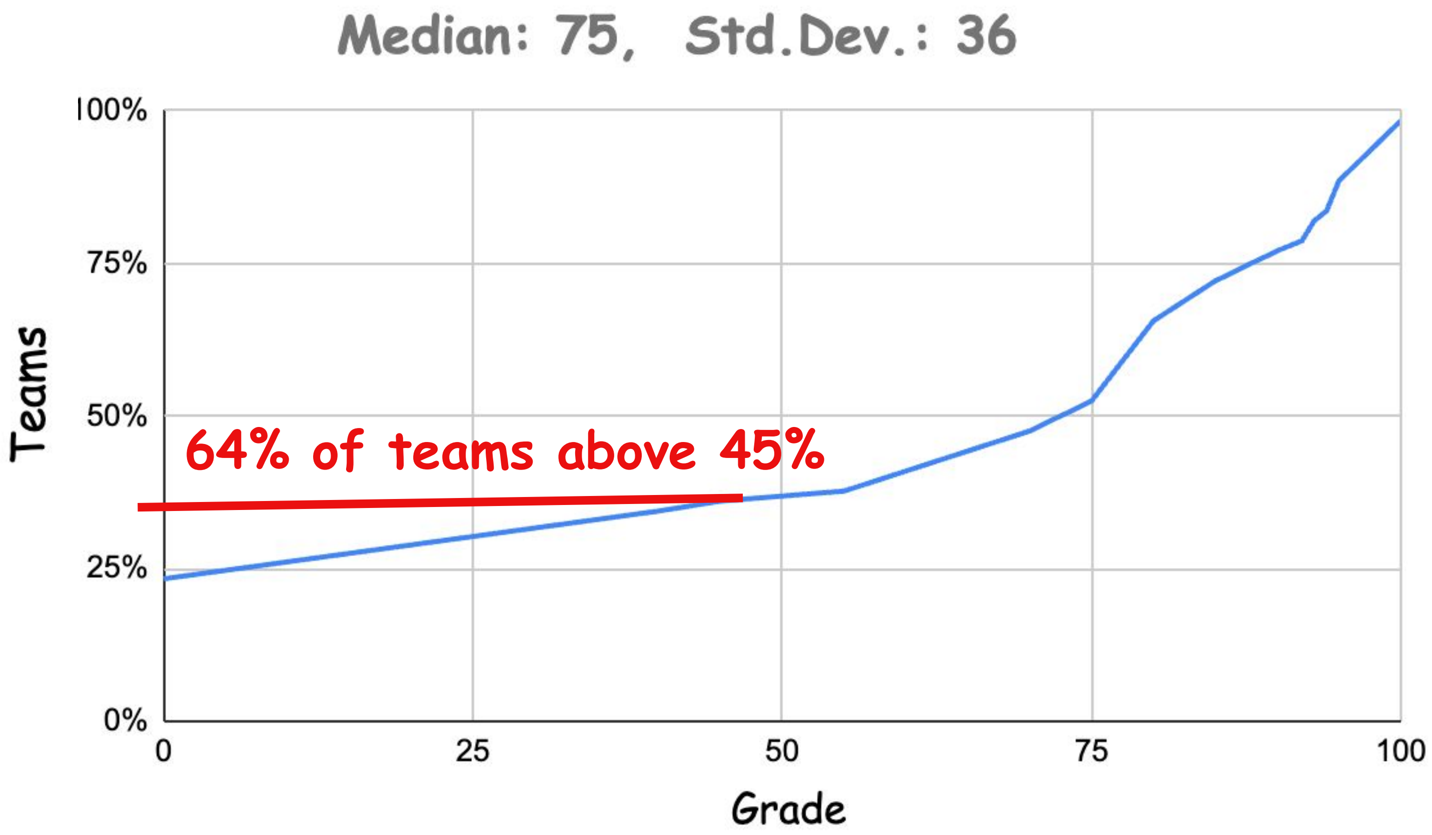
- **Correctness is not incremental**
  > The mentality "*my code is correct to an extend*" stops in K22.

- **If you cannot use a linter or git to produce/apply a patch**
  > You are not ready for this course

- **If you superficially ctrl-c/ctrl-v whatever LLMs produce**
  > You are not ready for this school

- **Finally:** Recursion is not for the kernel <- Let LLMs know about it

Programming assignment #1



Programming assignment #1

- Honest assessment of what's coming next

- **Programming assignment #2 is announced**
  - This is no toy system call
  - It will be painful; so, start working on it now!
  - Teams w/ < 55-60% in #1 => Reconsider taking K22
  - You are welcome to stay with us

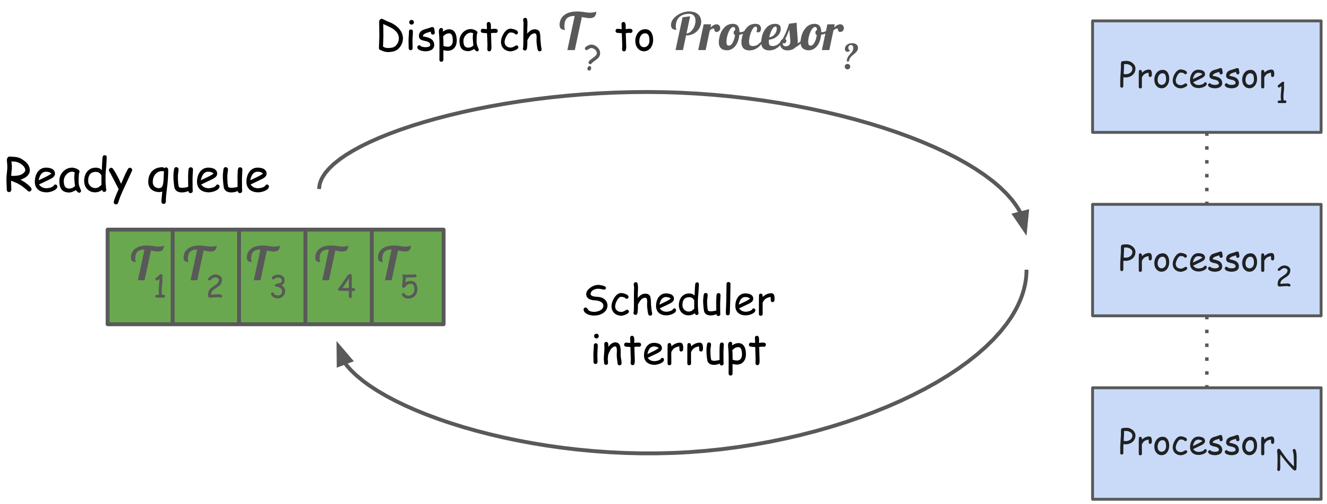
Overview

- 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**

The scheduling problem

> 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?



The scheduling problem

> 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?

> **Quantitative goals**

- Minimize avg. completion time of all jobs
- Minimize the avg. response time of all jobs (latency)
- Maximize #jobs completed per unit of time (throughput)

The scheduling problem

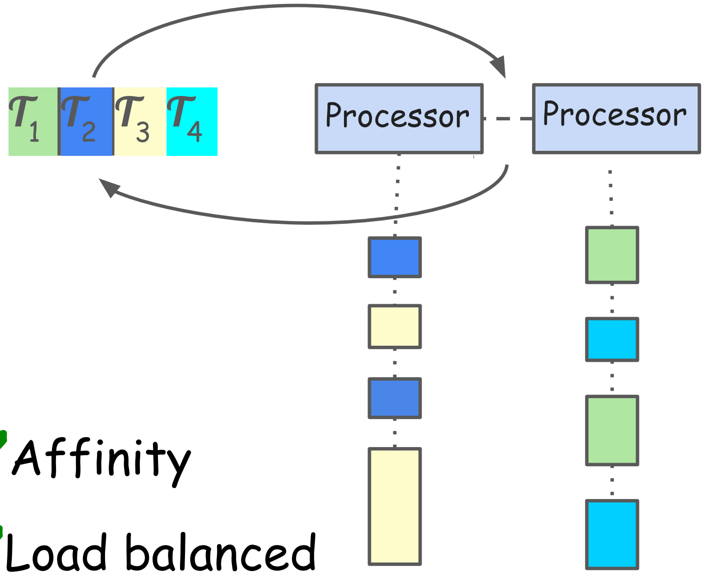
> Given k tasks ready to run in a system with N available processors, which task should be dispatched to which processor at any given point in time?

> **Qualitative goals**

- Jobs receive a similar share of available processors' time
- Upper bound on the maximum latency of jobs
- Uniform load across all available processors

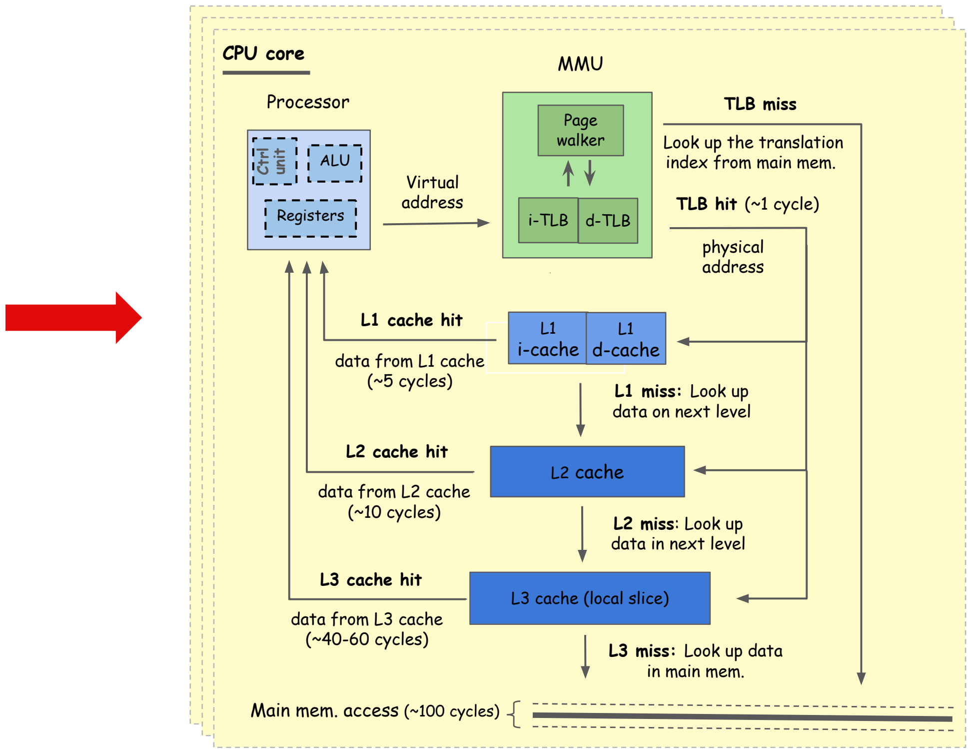
SMP load balancing and processor affinity

Processor affinity?



✔Affinity

✔Load balanced



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

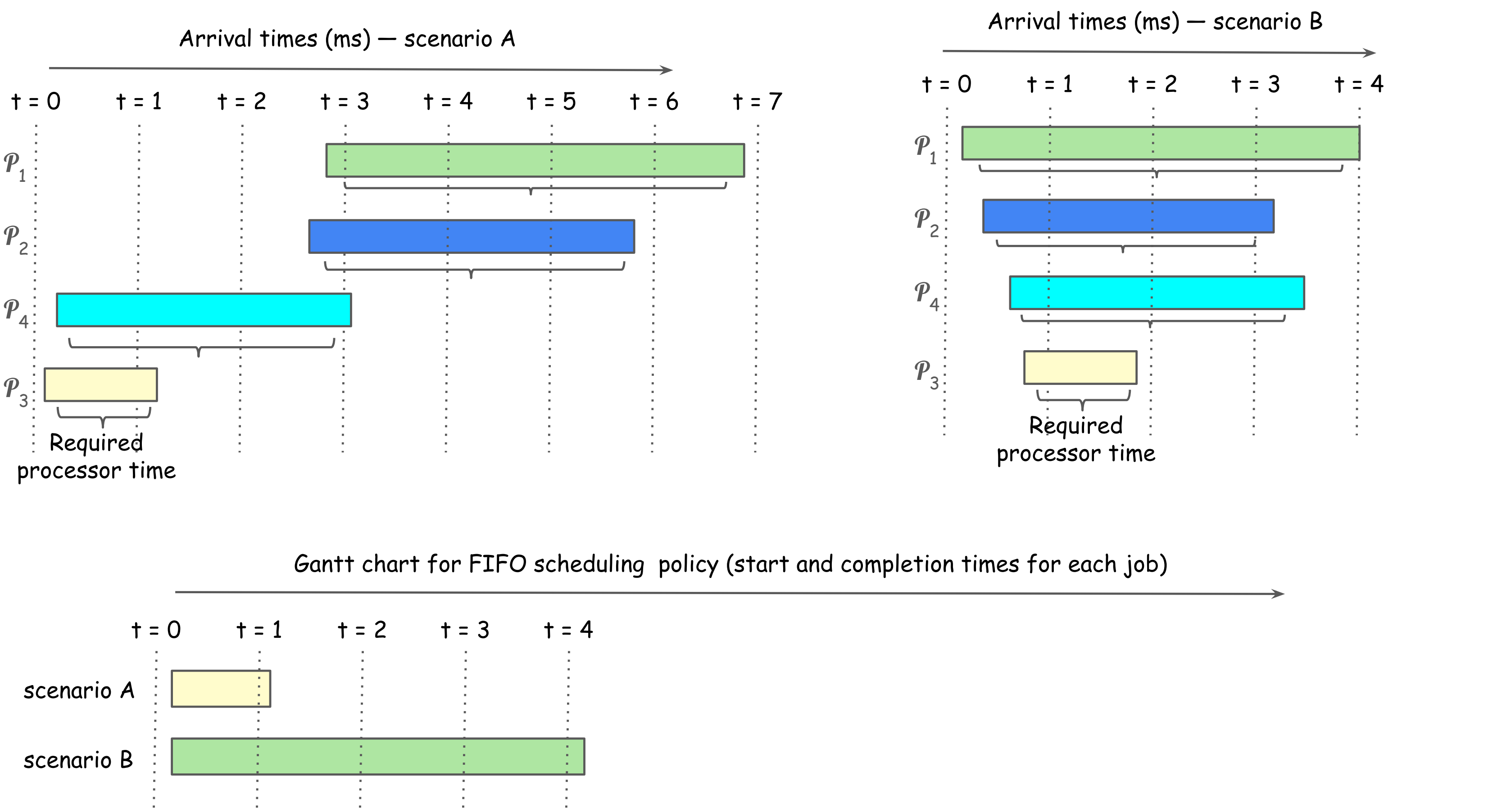
- CPU-bound
  - Their tasks spend most time doing intensive computation
  - Rarely yield voluntarily and rarely need to perform I/O
  - Example: Scientific simulations / computations
  - Sched. goals: **Balanced processor time / avoid starvation**
  - Sched. algorithms: RR with large quanta

Workloads and scheduling requirements

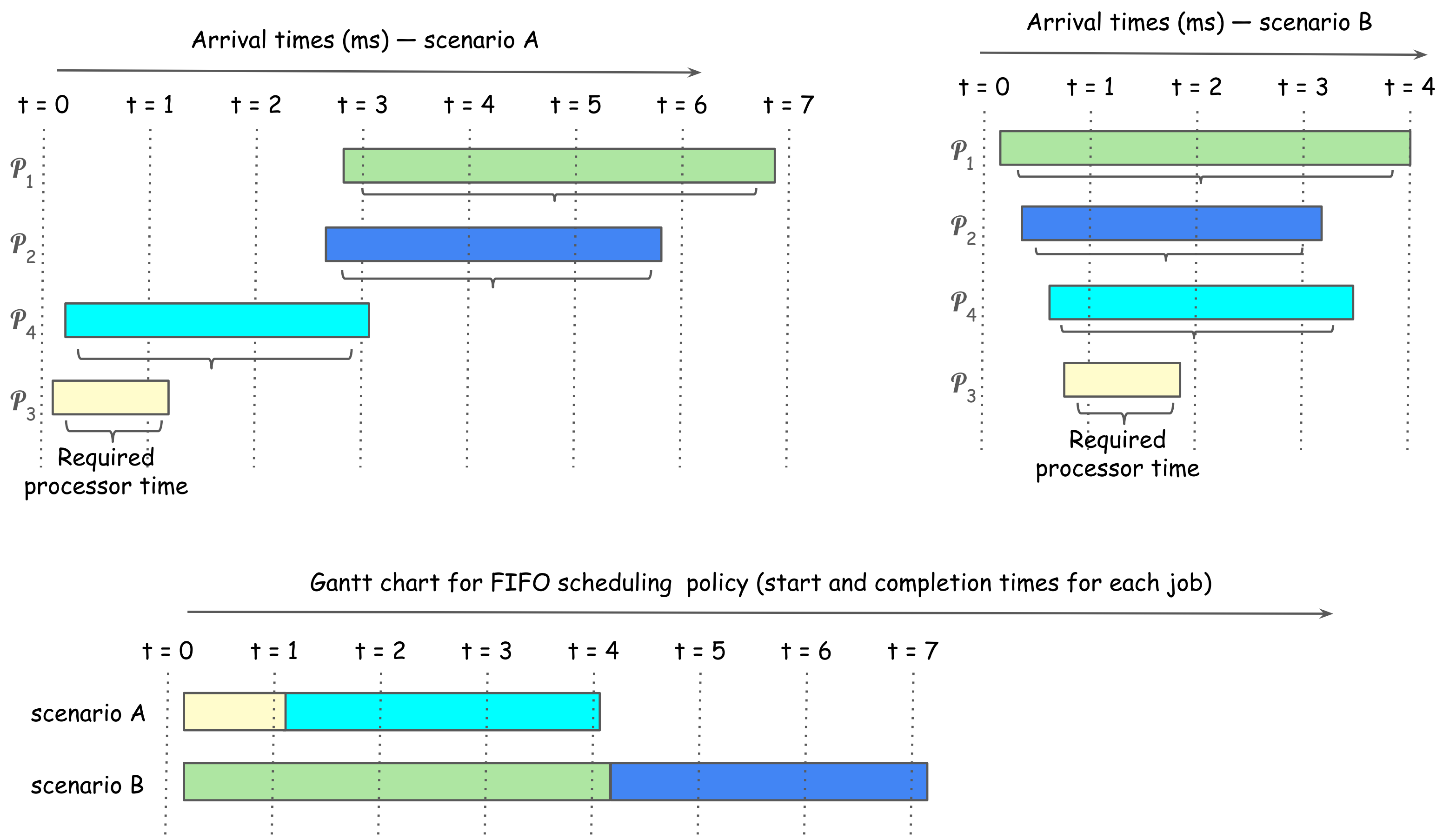
CPU- vs I/O-bound workloads

- 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

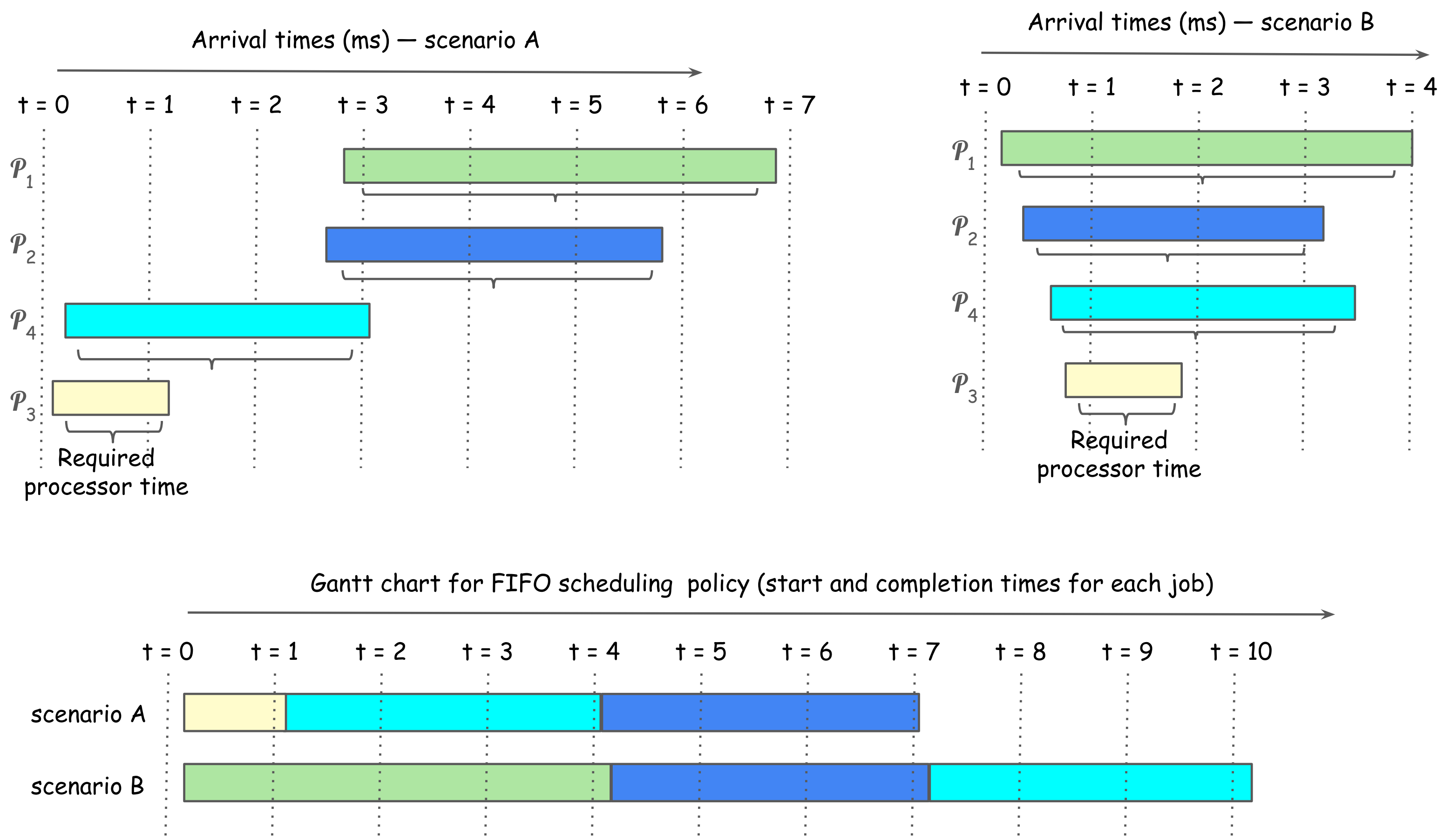
First in First Out scheduling policy (SCHED_FIFO)



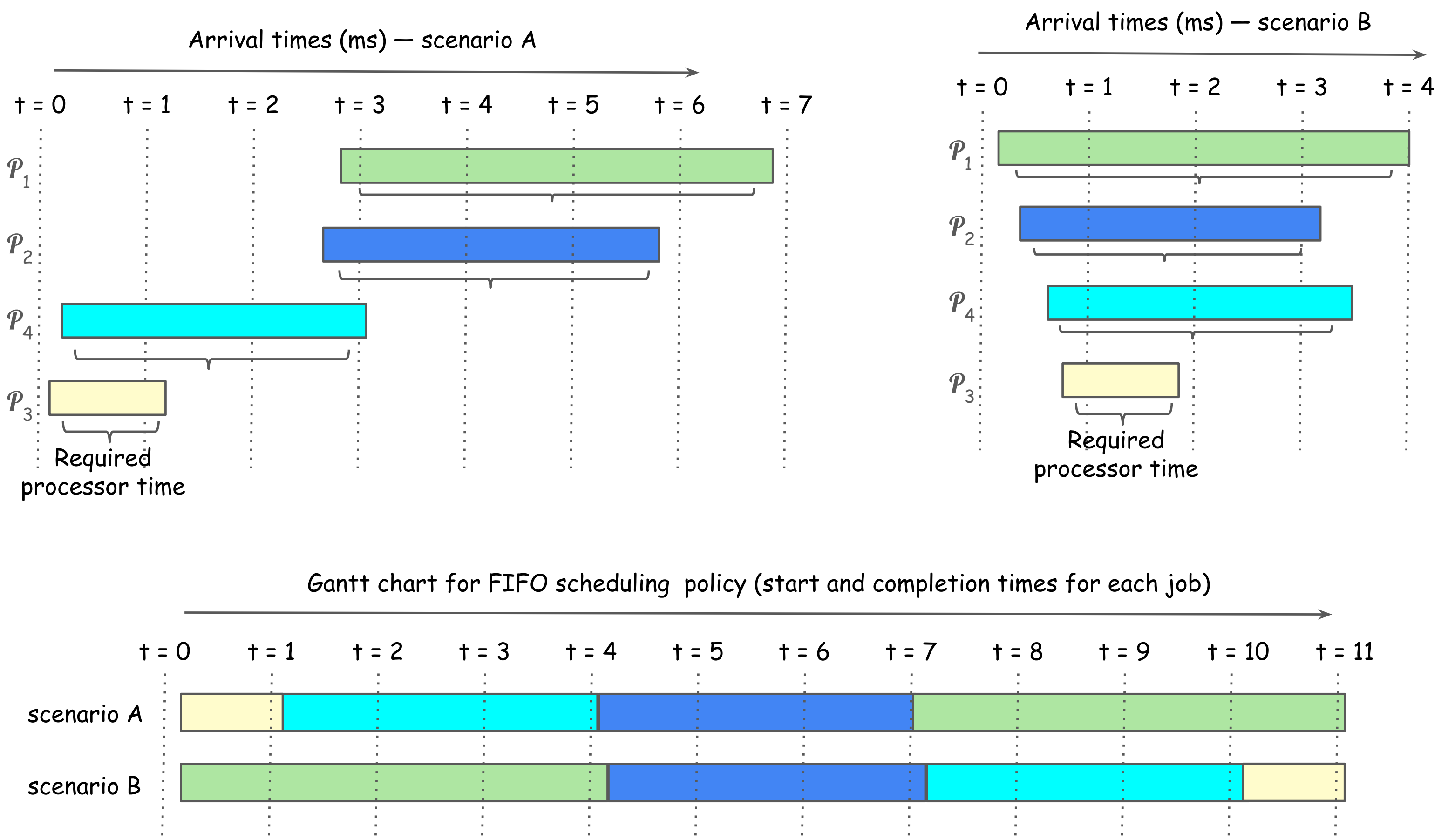
First in First Out scheduling policy (SCHED_FIFO)



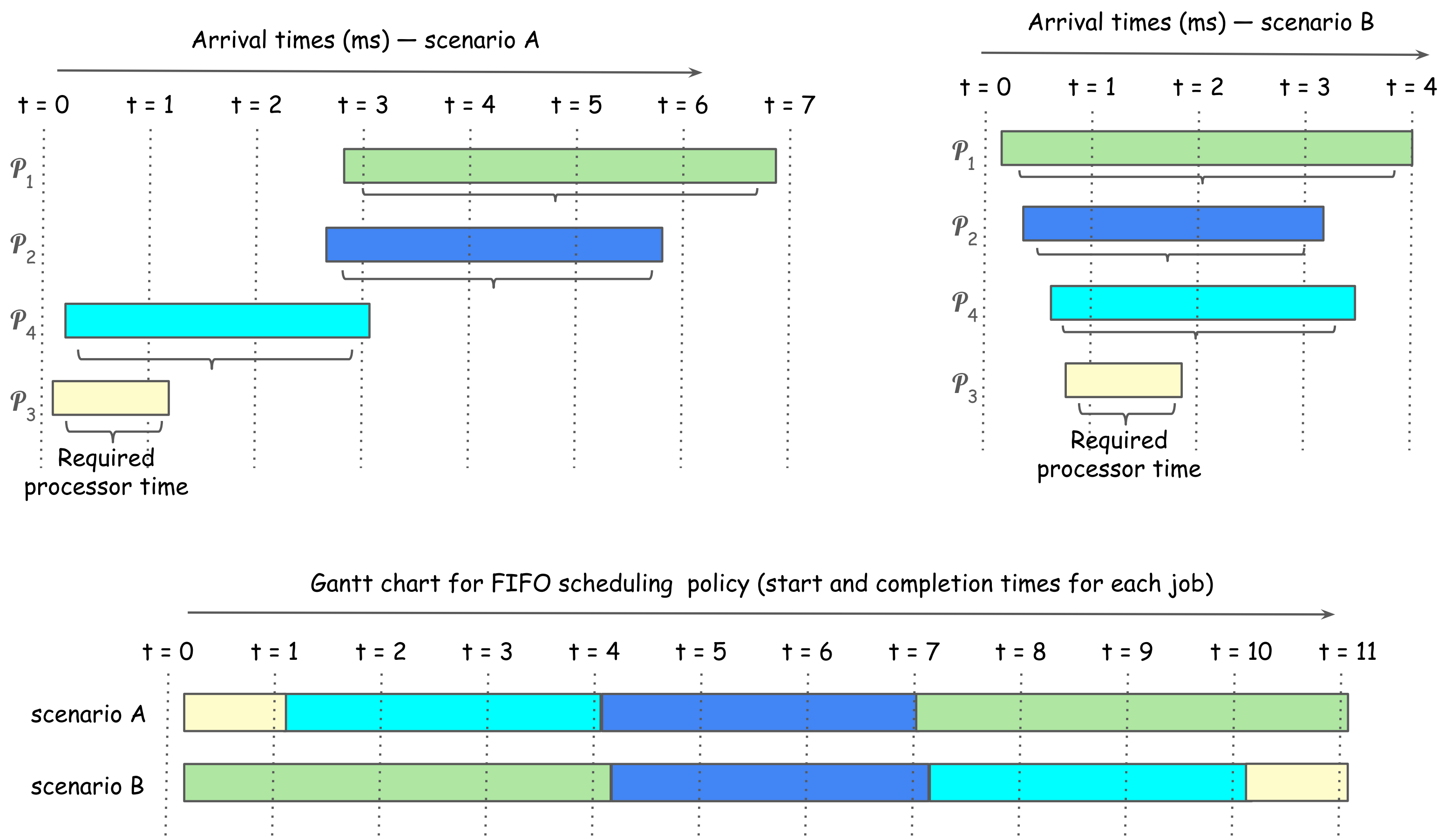
First in First Out scheduling policy (SCHED_FIFO)



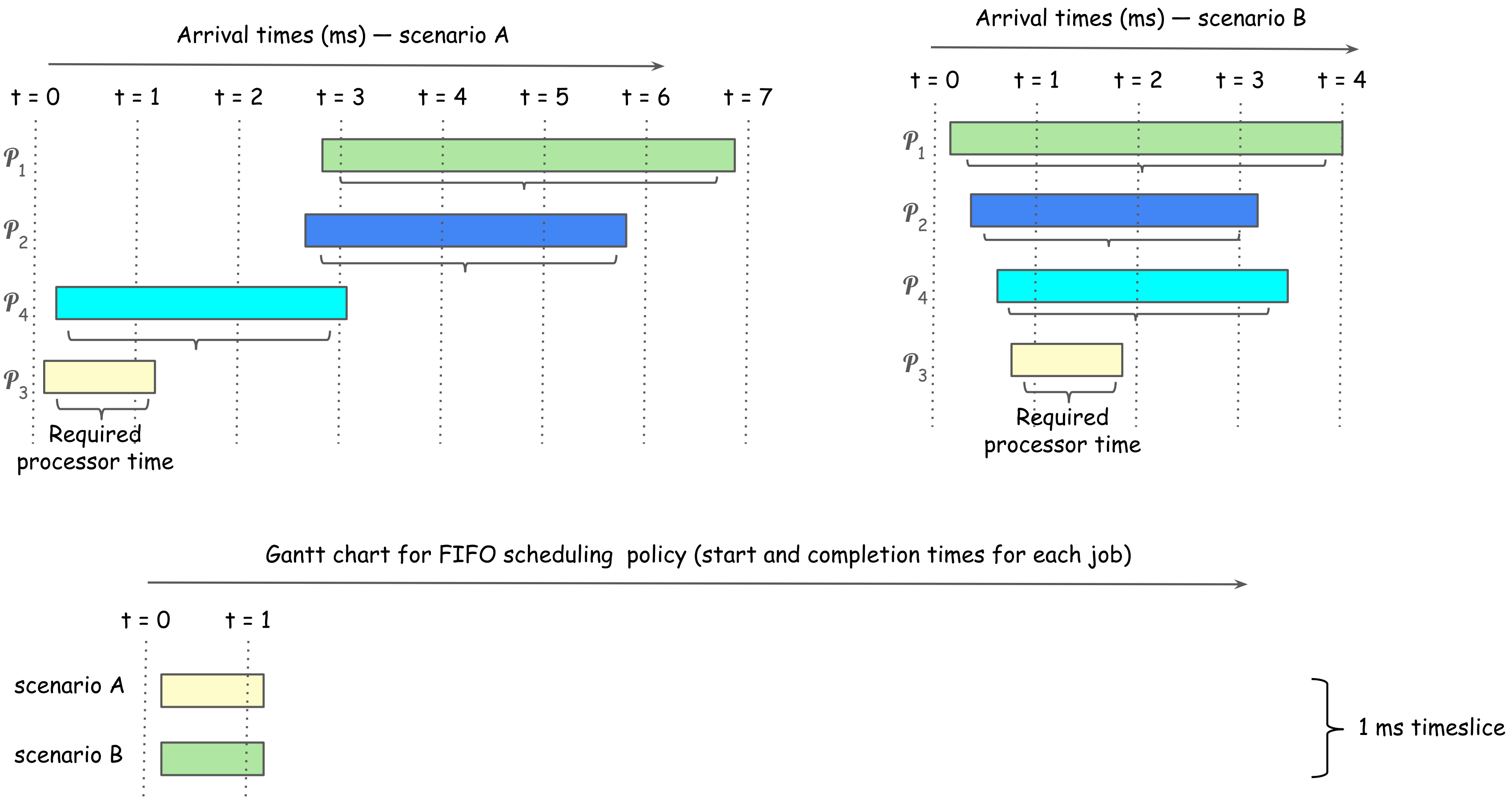
First in First Out scheduling policy (SCHED_FIFO)



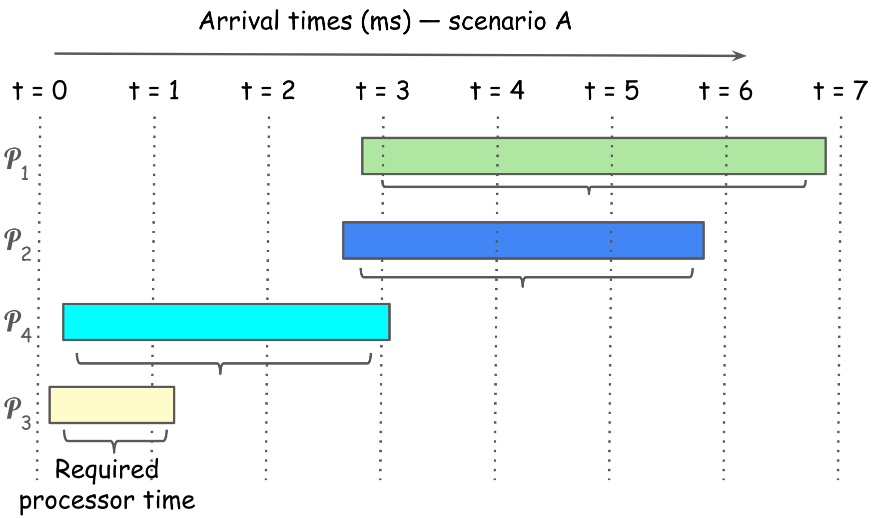
SCHED_FIFO: Avg.completion and response time?

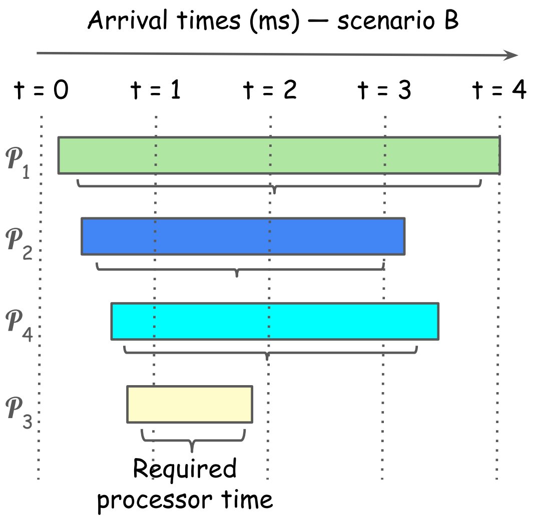


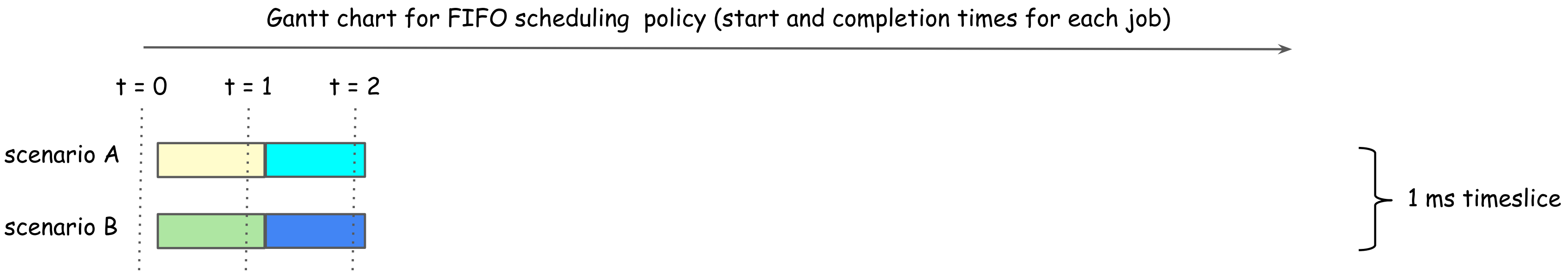
Round Robin scheduling policy (SCHED_RR)



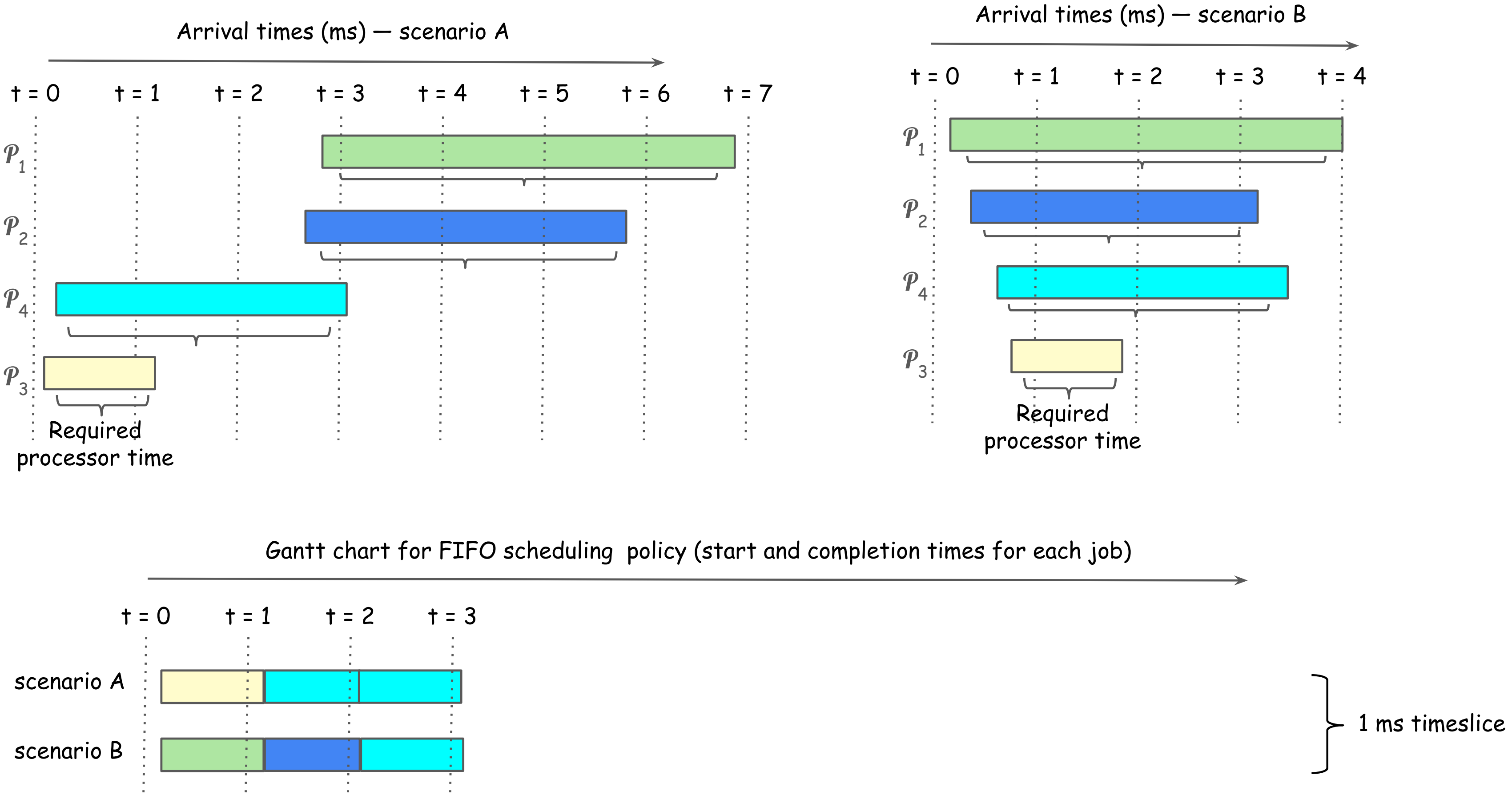
Round Robin scheduling policy (SCHED_RR)



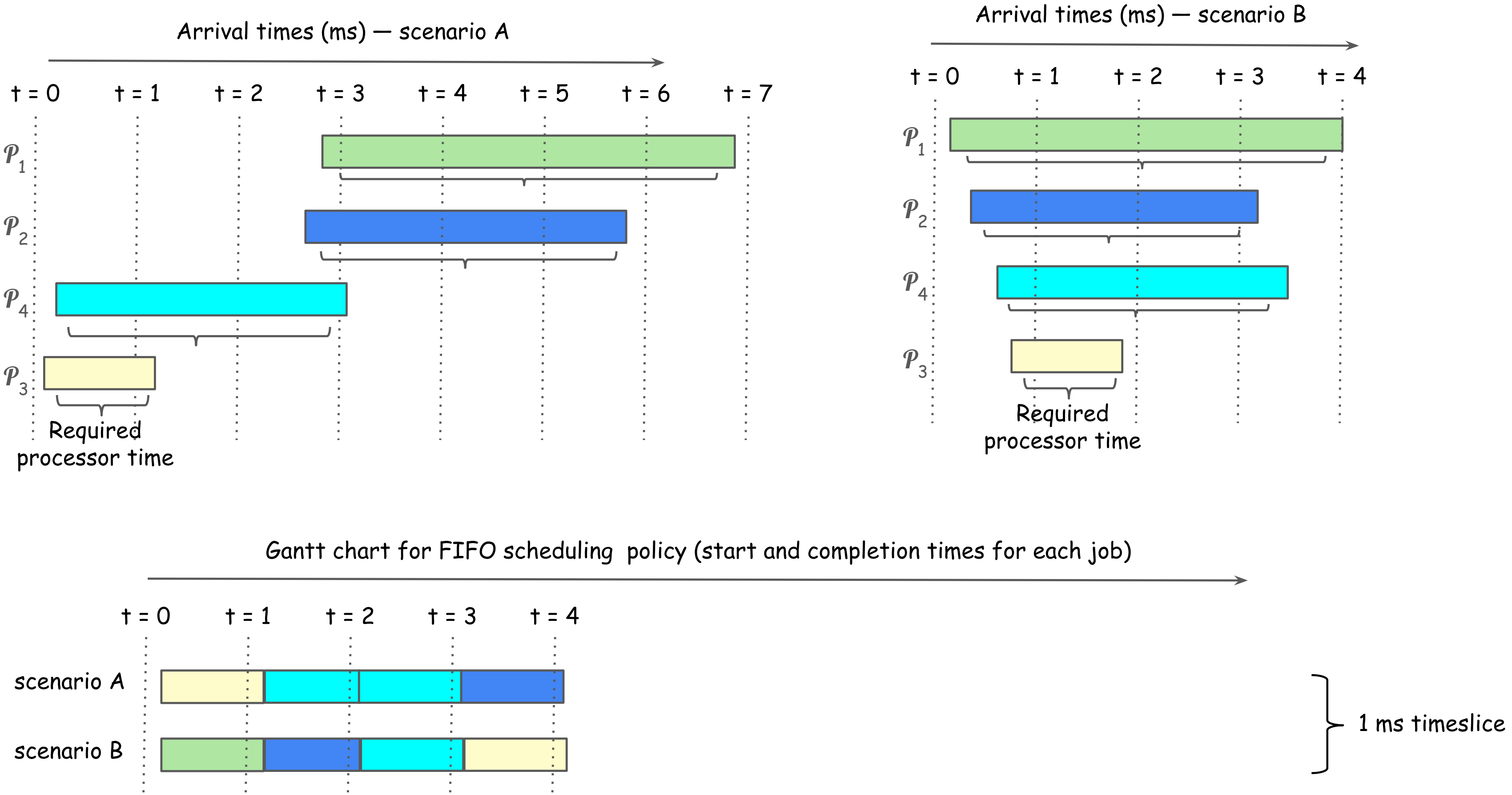




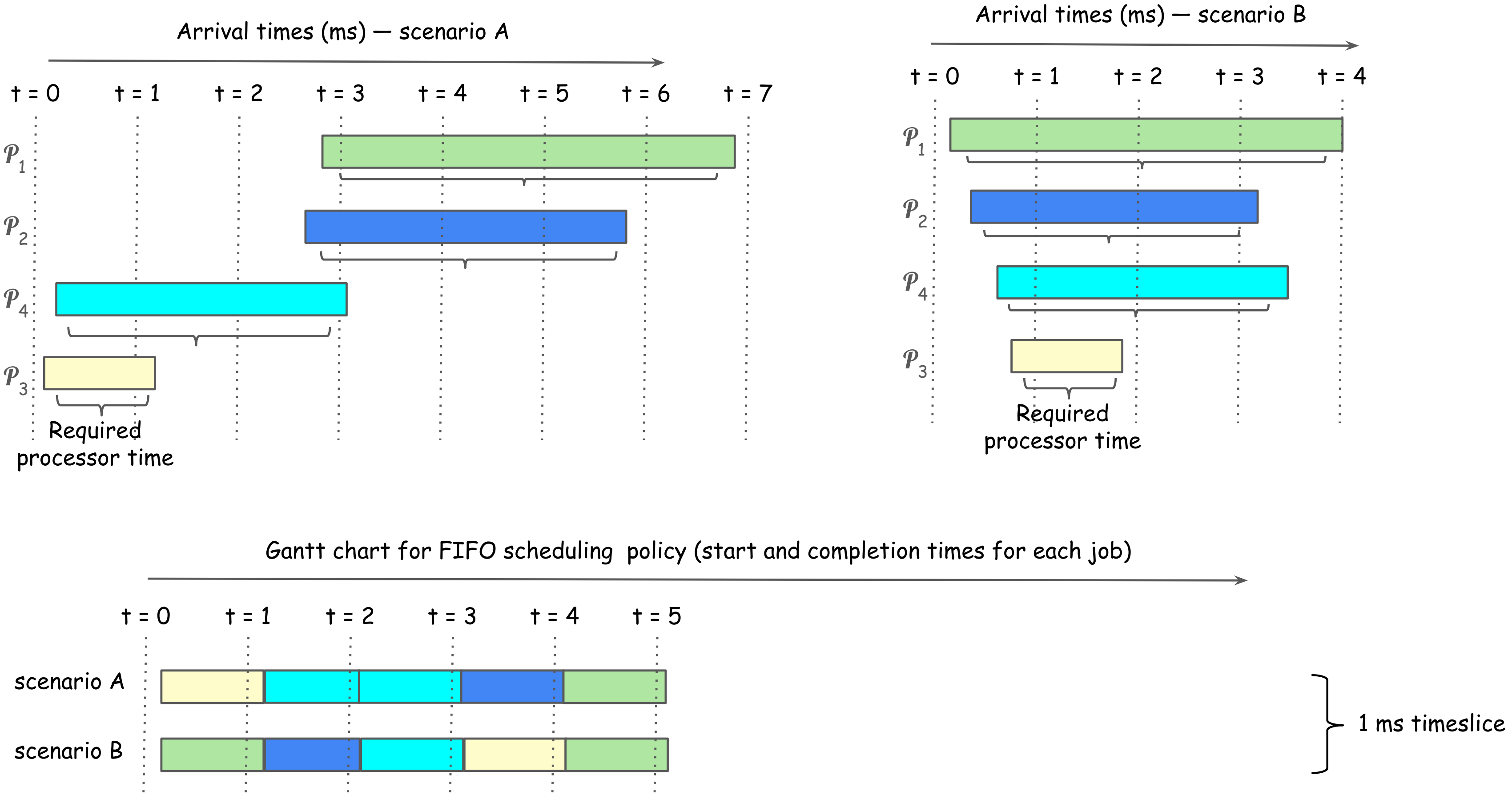
Round Robin scheduling policy (SCHED_RR)



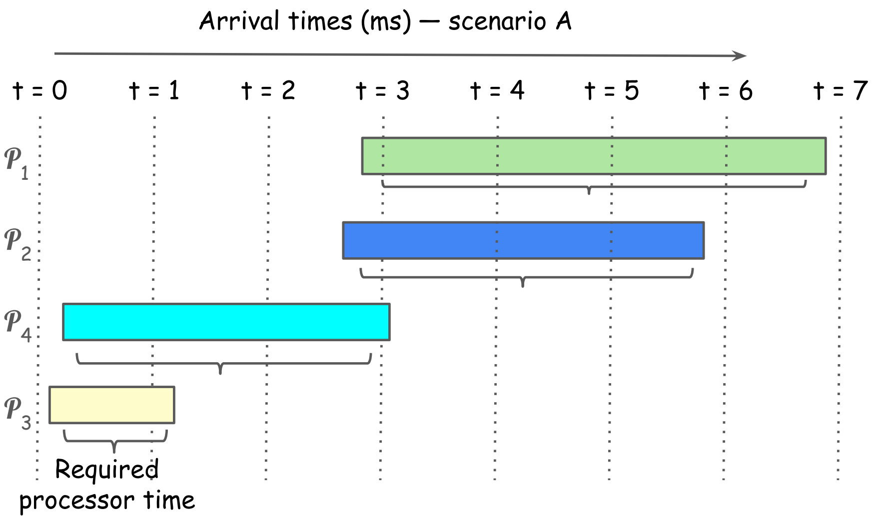
Round Robin scheduling policy (SCHED_RR)

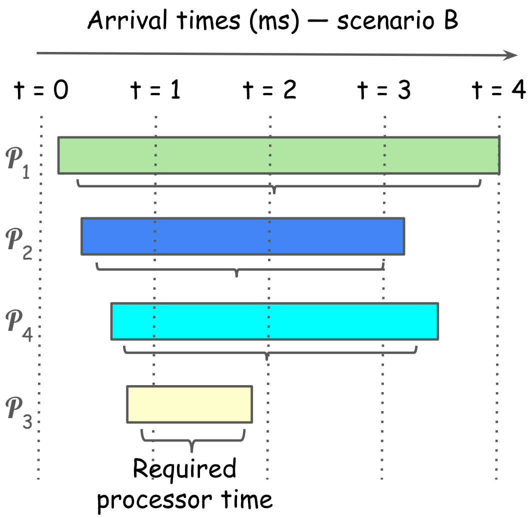


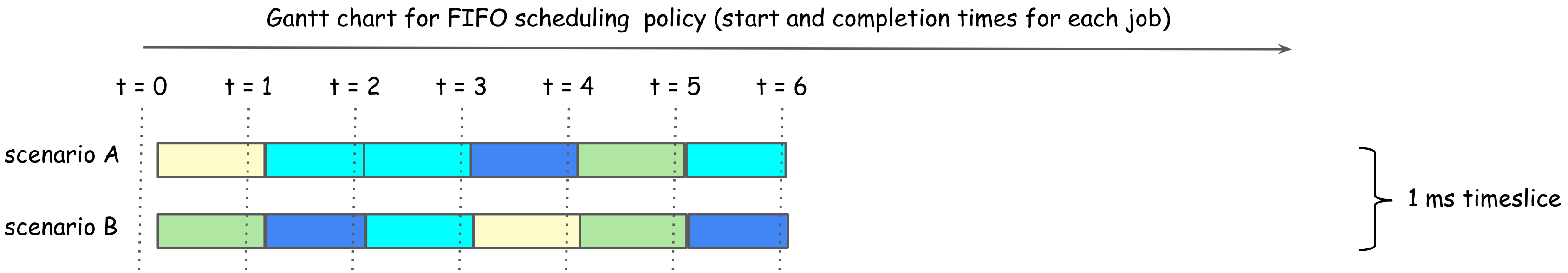
Round Robin scheduling policy (SCHED_RR)



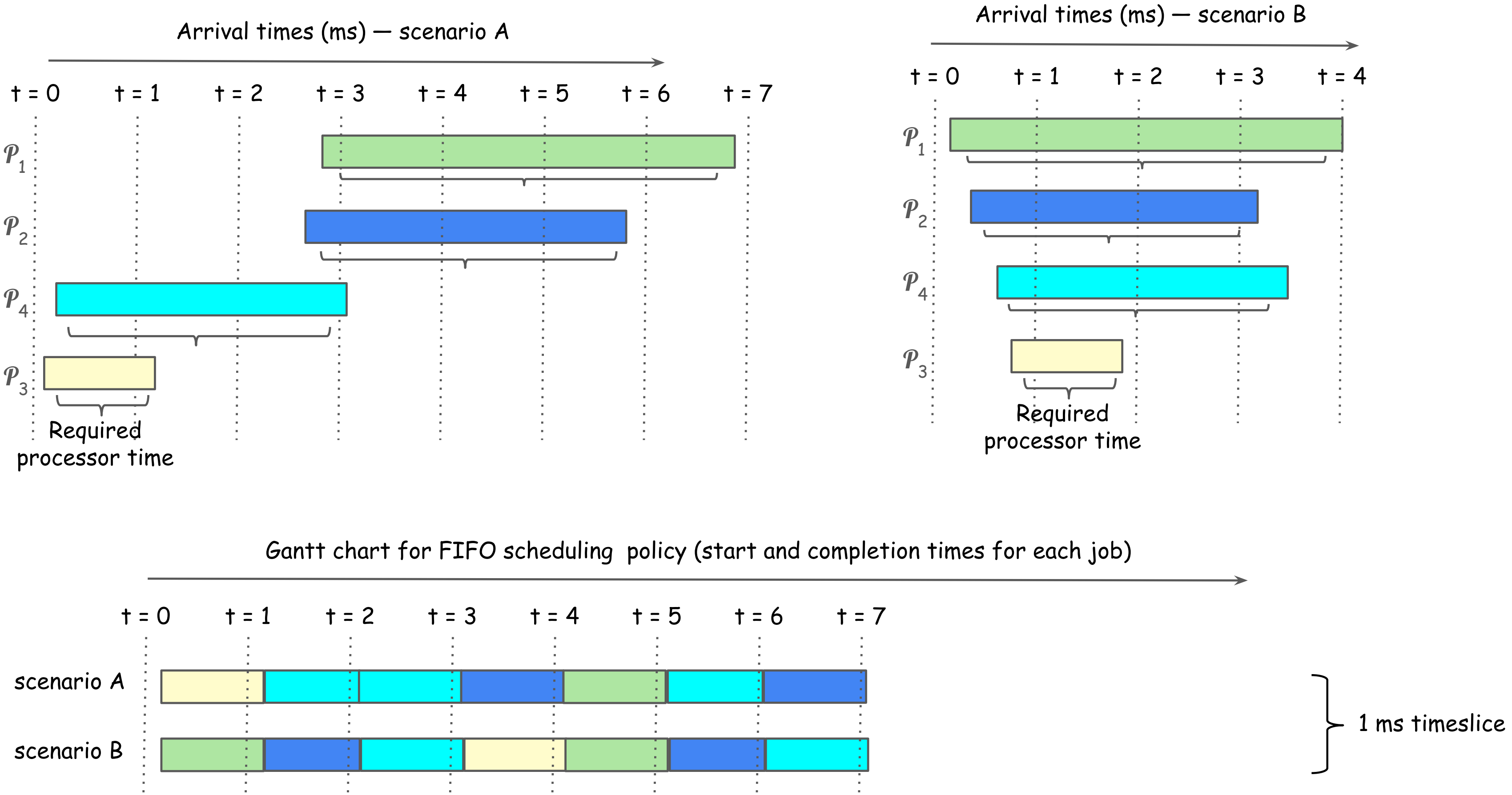
Round Robin scheduling policy (SCHED_RR)



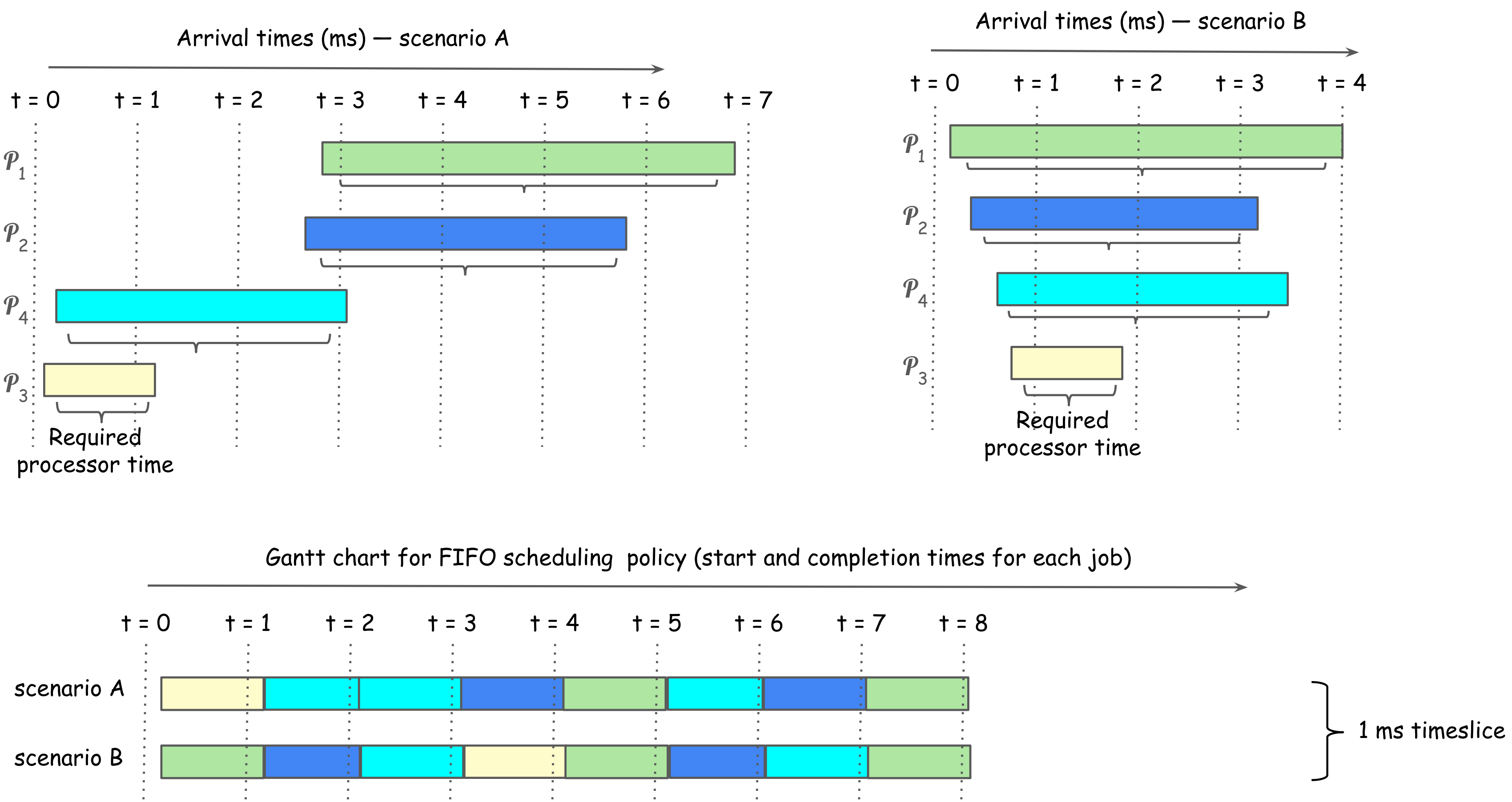




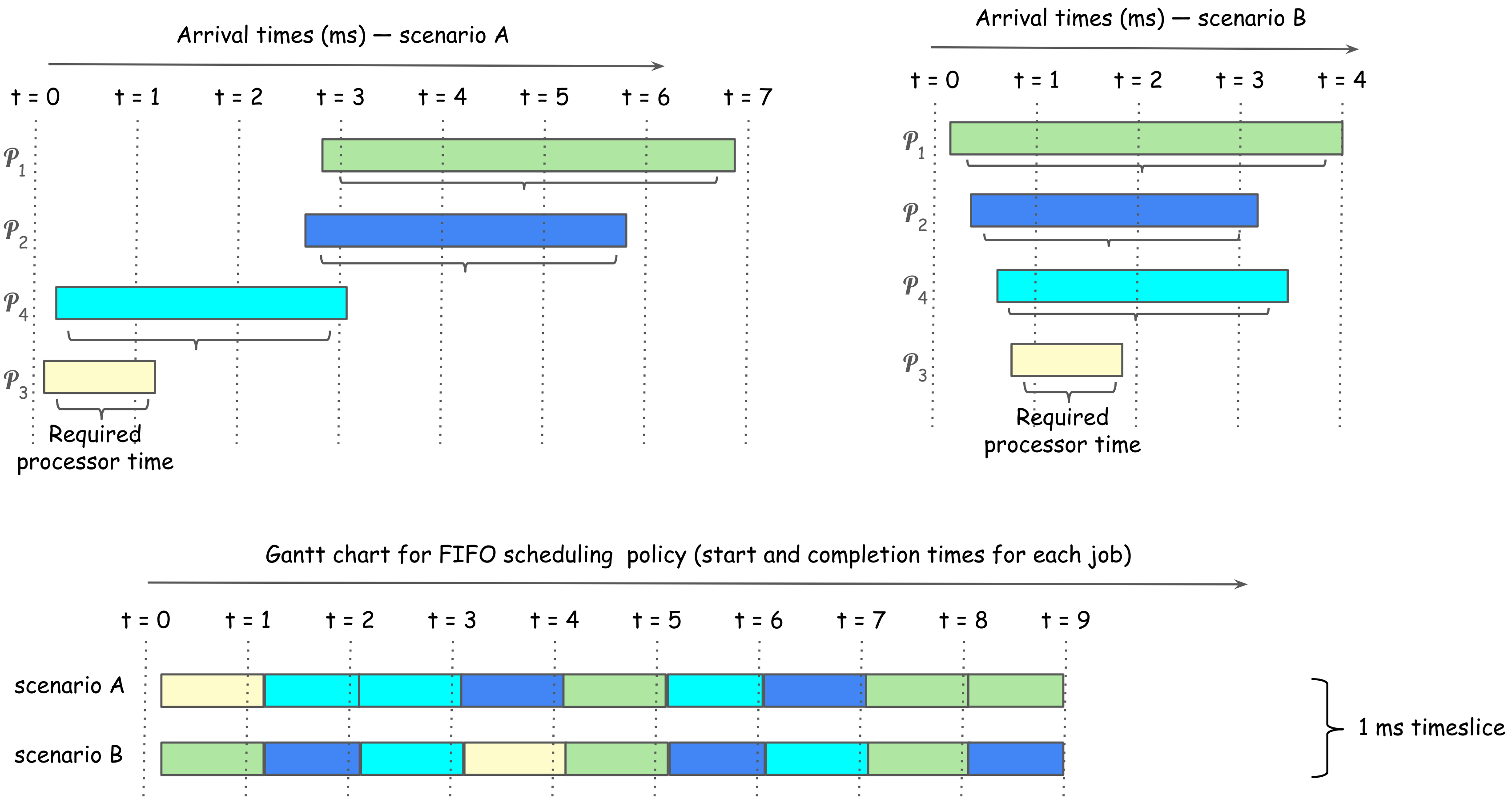
Round Robin scheduling policy (SCHED_RR)



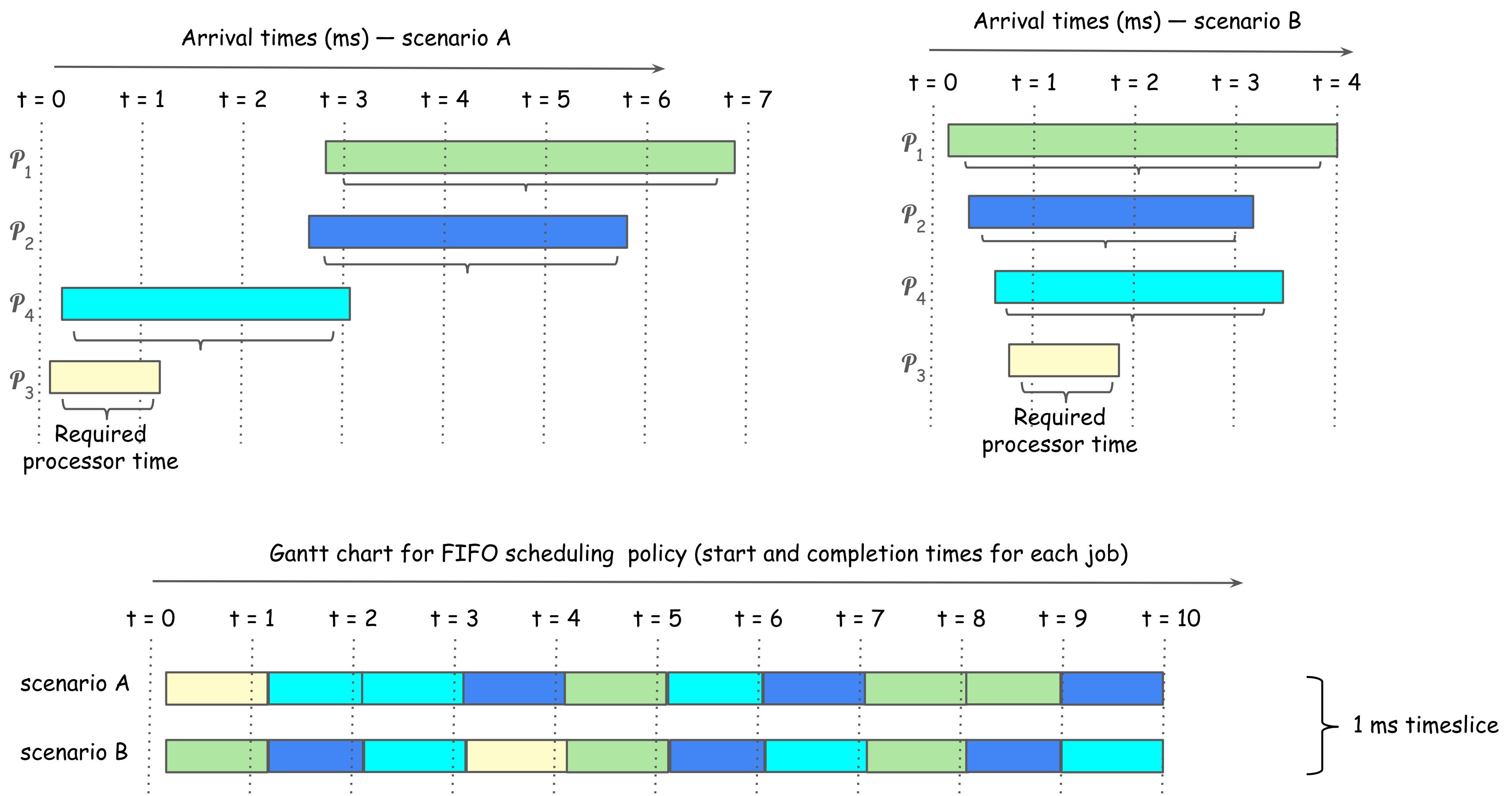
Round Robin scheduling policy (SCHED_RR)



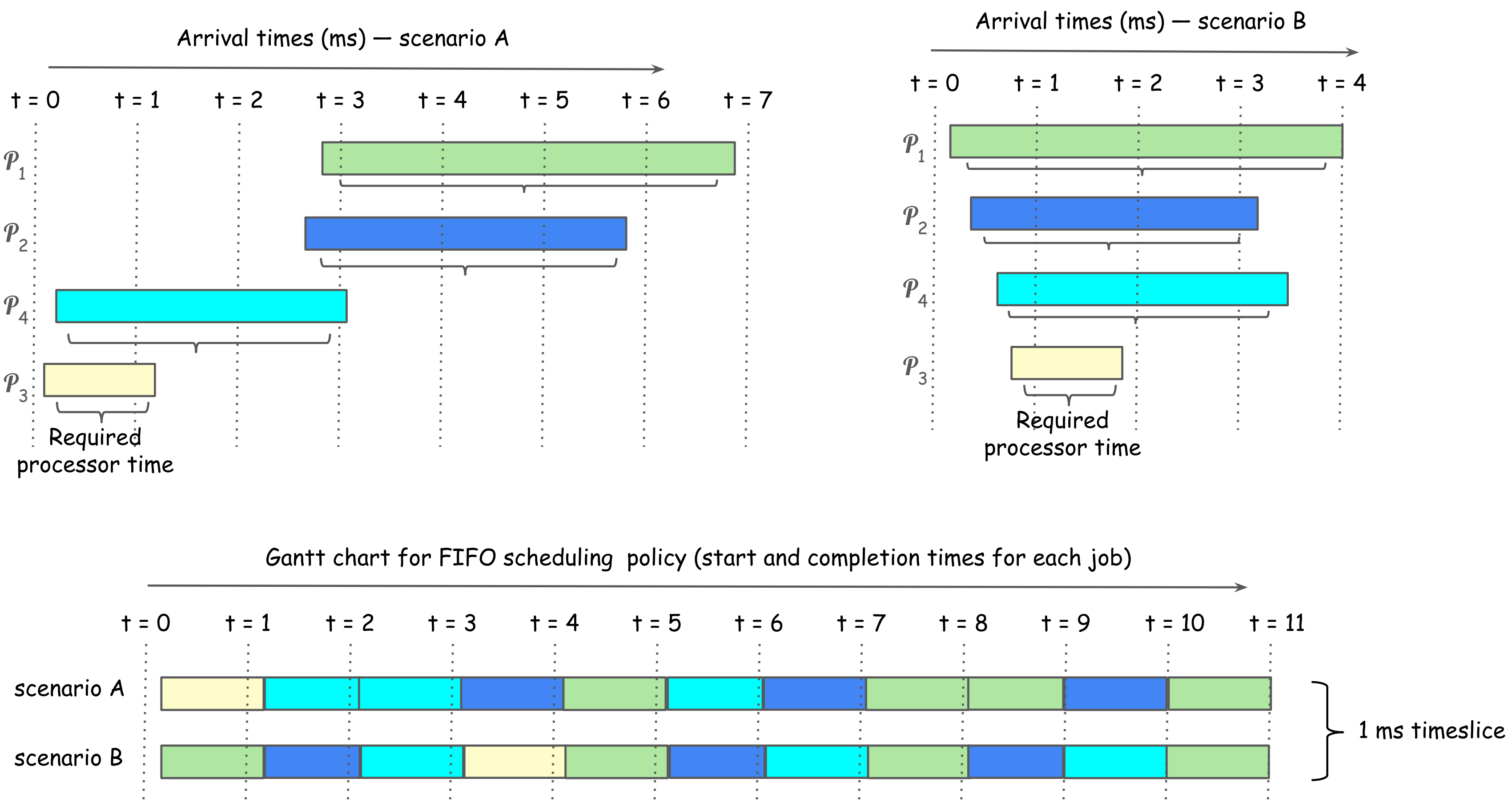
Round Robin scheduling policy (SCHED_RR)



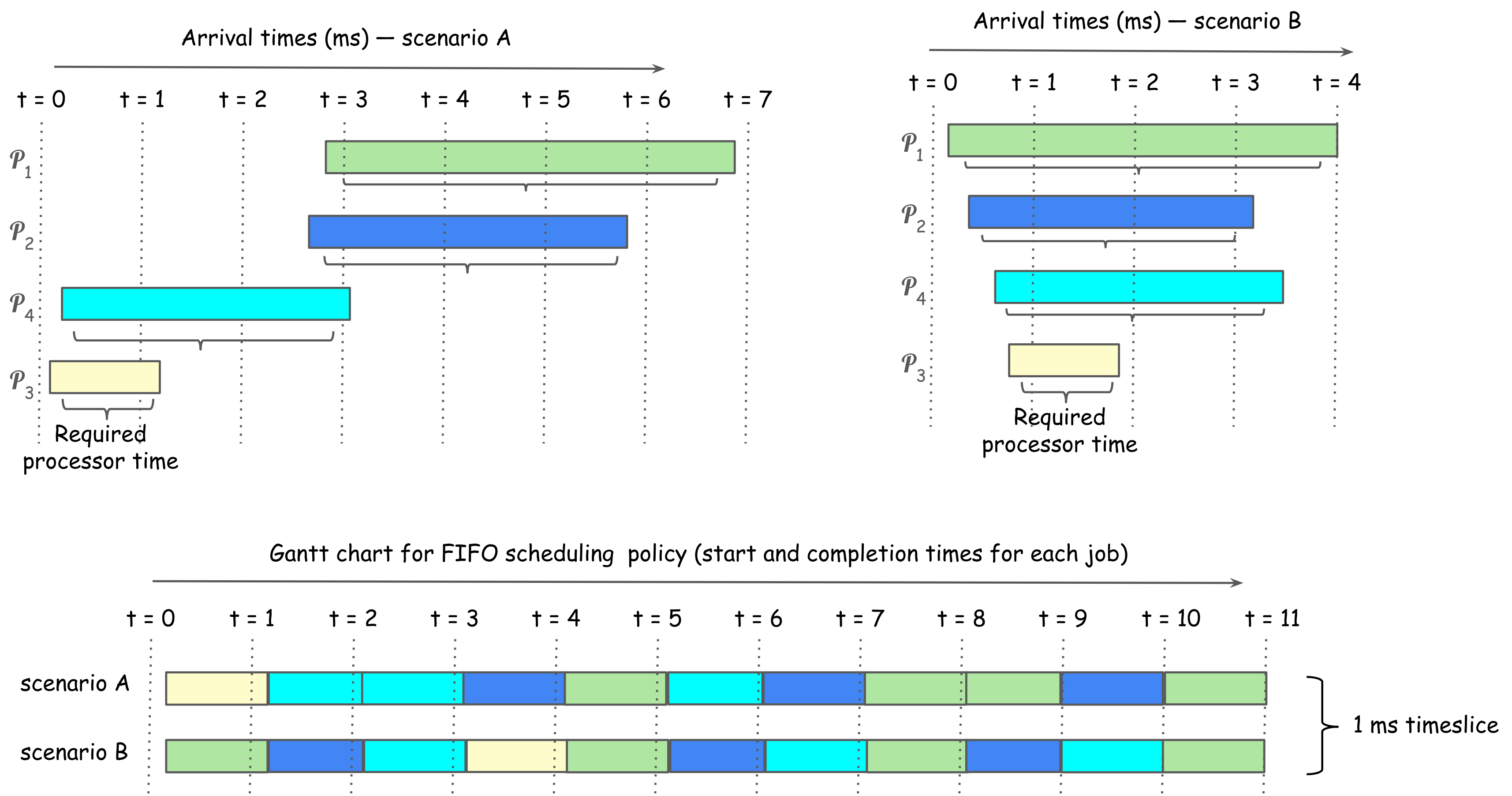
Round Robin scheduling policy (SCHED_RR)



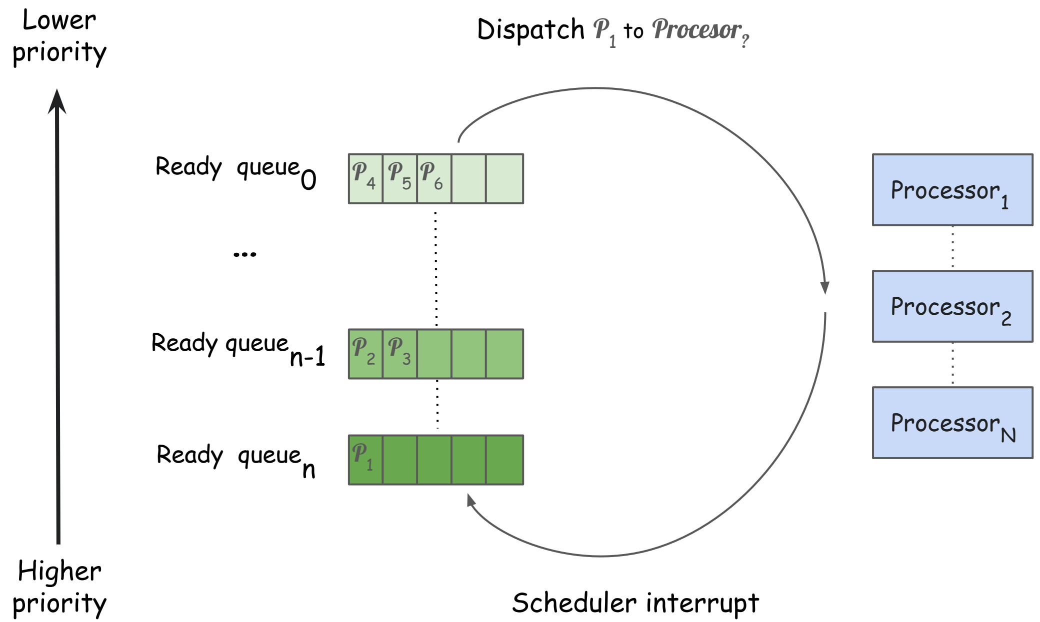
Round Robin scheduling policy (SCHED_RR)



SCHED_RR: Avg.completion and response time?



Hierarchical priority-based scheduling



Ready for your first bug in the outer universe

Ready for your first bug in the outer universe

Mars Rover: software and hardware

- Works Real-Time Operating System (RTOS)

Mars Rover: software and hardware

> Works Real-Time Operating System (RTOS)

- Tasks must meet strict timing constraints

Mars Rover: software and hardware

> Works Real-Time Operating System (RTOS)

- Tasks must meet strict timing constraints
- Preemptive with priority-based scheduling

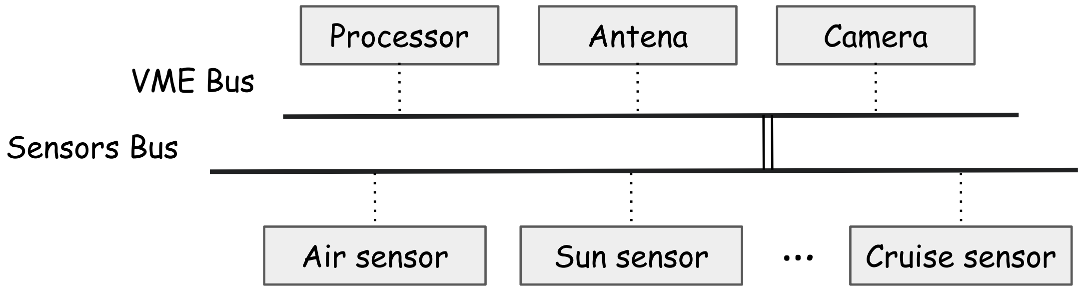
Mars Rover: software and hardware

> 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)

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)

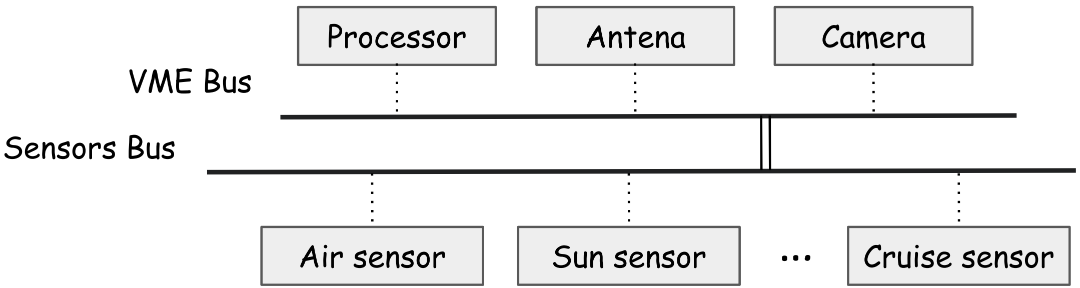
- Tasks must meet strict timing constraints
- Preemptive with priority-based scheduling
- Scheduler ticks at 8 Hz (i.e., every 125ms)

> Hardware overview

- Data from sensor bus to the VMA bus (to antenna)
- Processor signal from VMA bus to sensor bus (cruise)

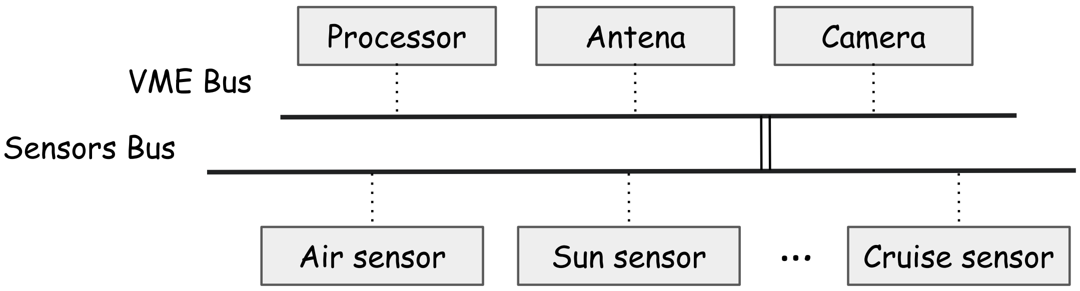
Mars Rover: software and hardware

- Synchronization



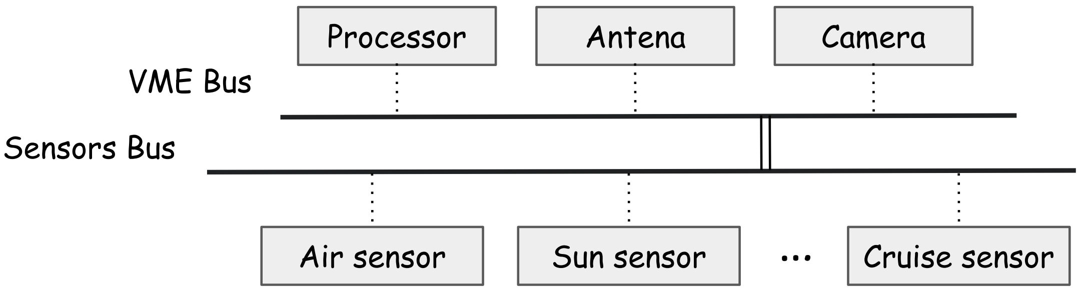
Mars Rover: software and hardware

- **Synchronization**
  - sched_tsk: Decides who transmits data next



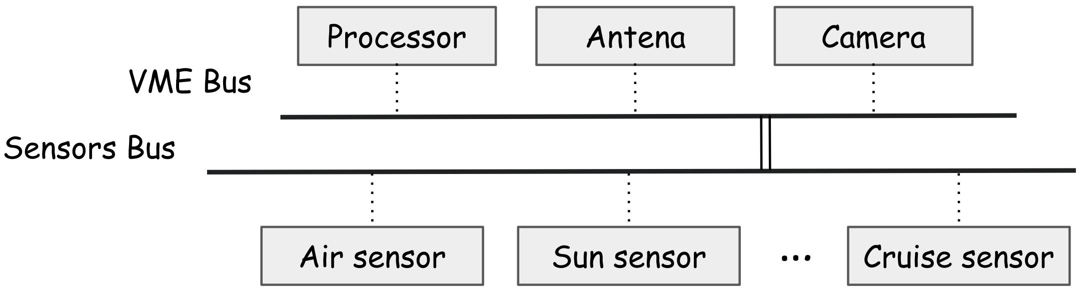
Mars Rover: software and hardware

- **Synchronization**
  - sched_tsk: Decides who transmits data next
  - dist_tsk: Decides who receives data next



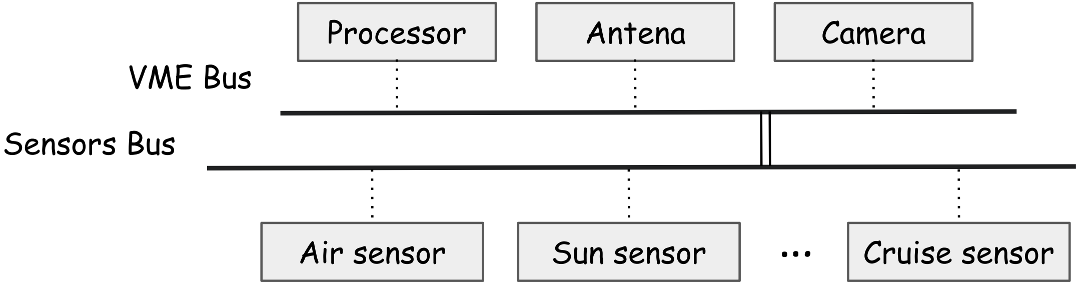
Mars Rover: software and hardware

- **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



Mars Rover: software and hardware

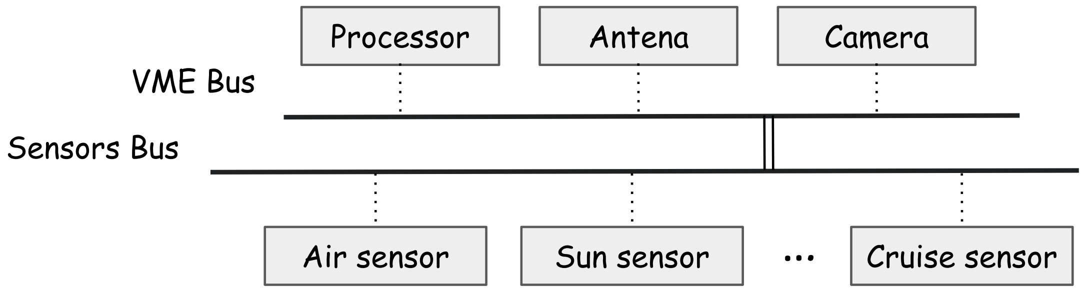
- **Synchronization**
  - sched_tsk: Decides who transmits data next
  - dist_tsk: Decides who receives data next
  - comm_tsk: Uses the antenna to transmit data to Earth
  - asi_tsk: Uses the air sensor for scientific computations



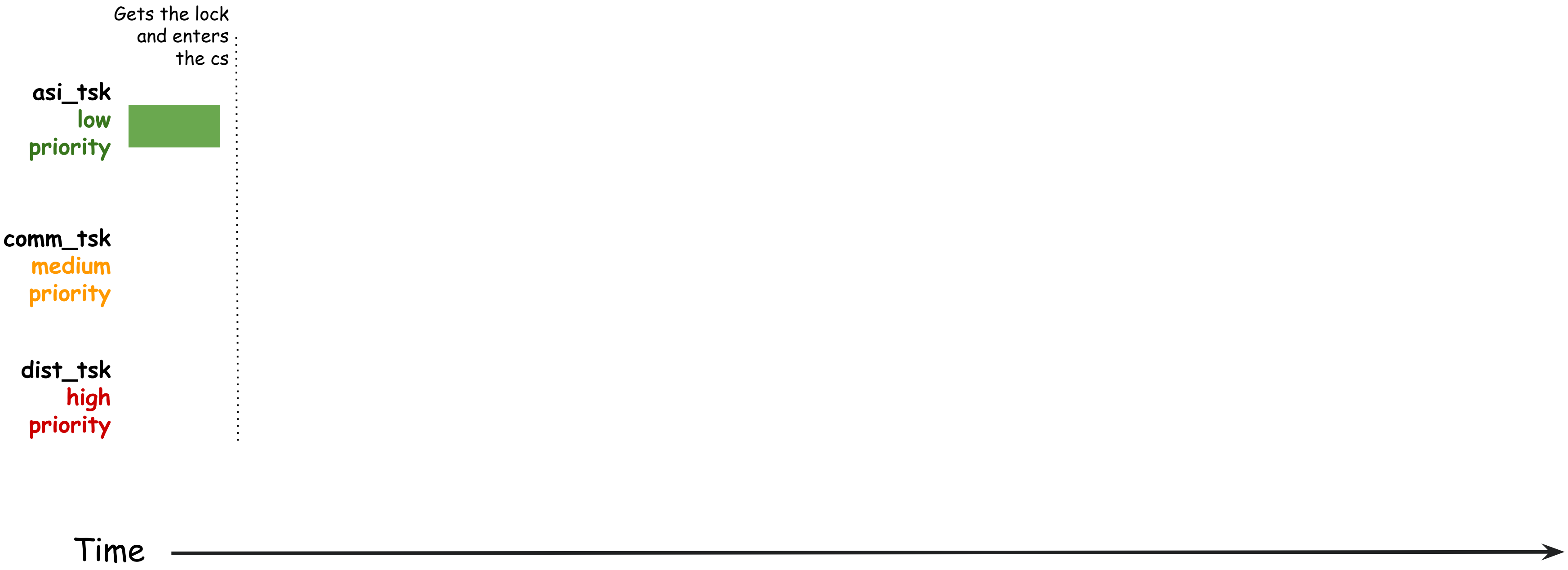
Mars Rover: software and hardware

> **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

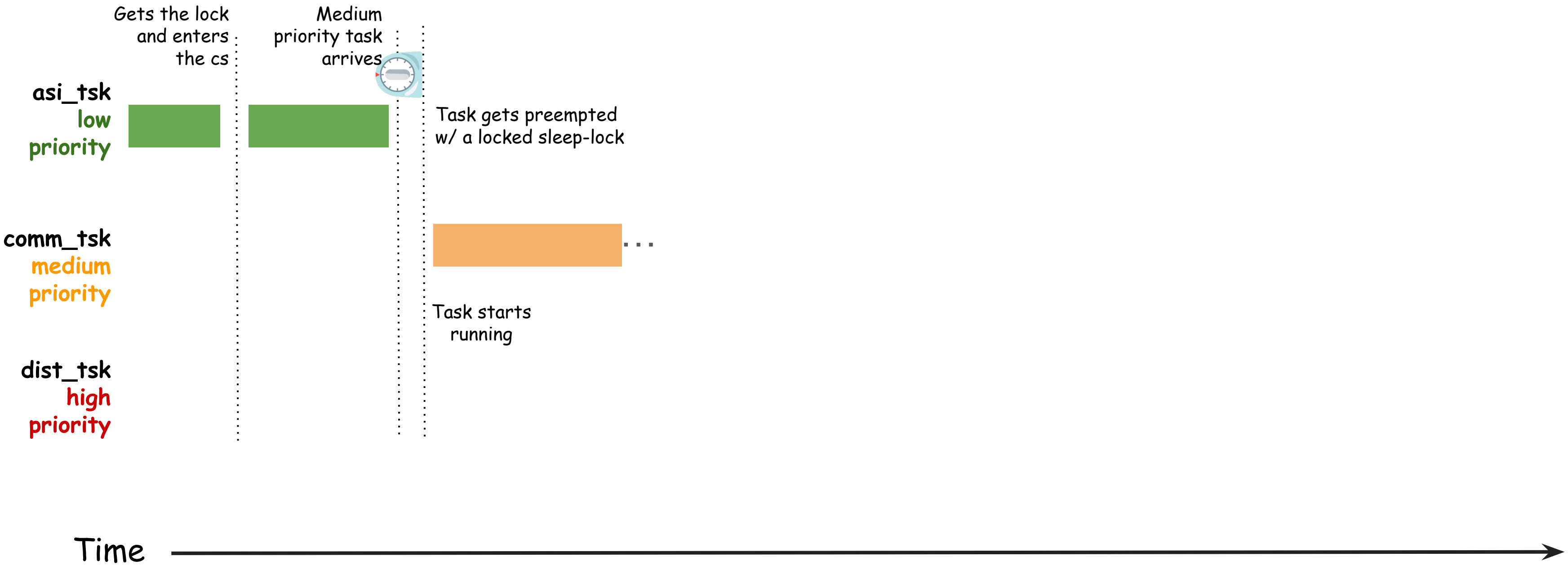
Priorities: sched_tsk > dist_tsk (high) > comm_tsk (medium) > asi_tsk (low)



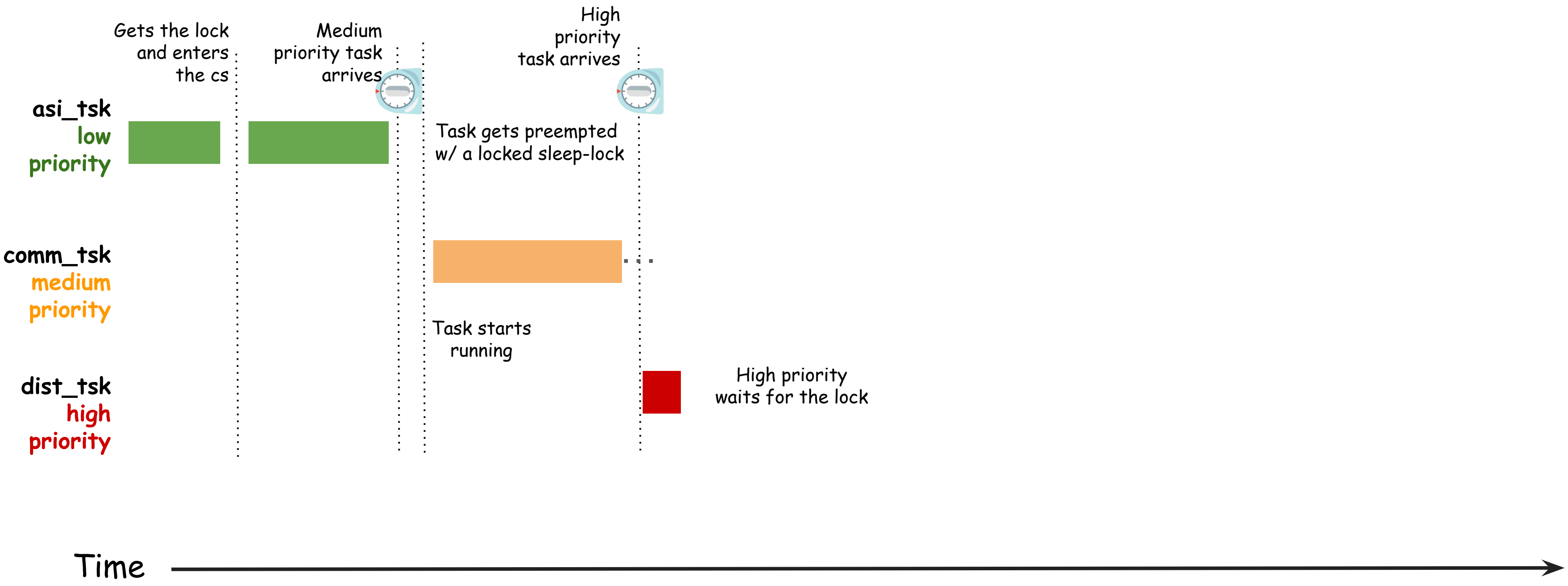
Ready for your first bug in the outer universe?



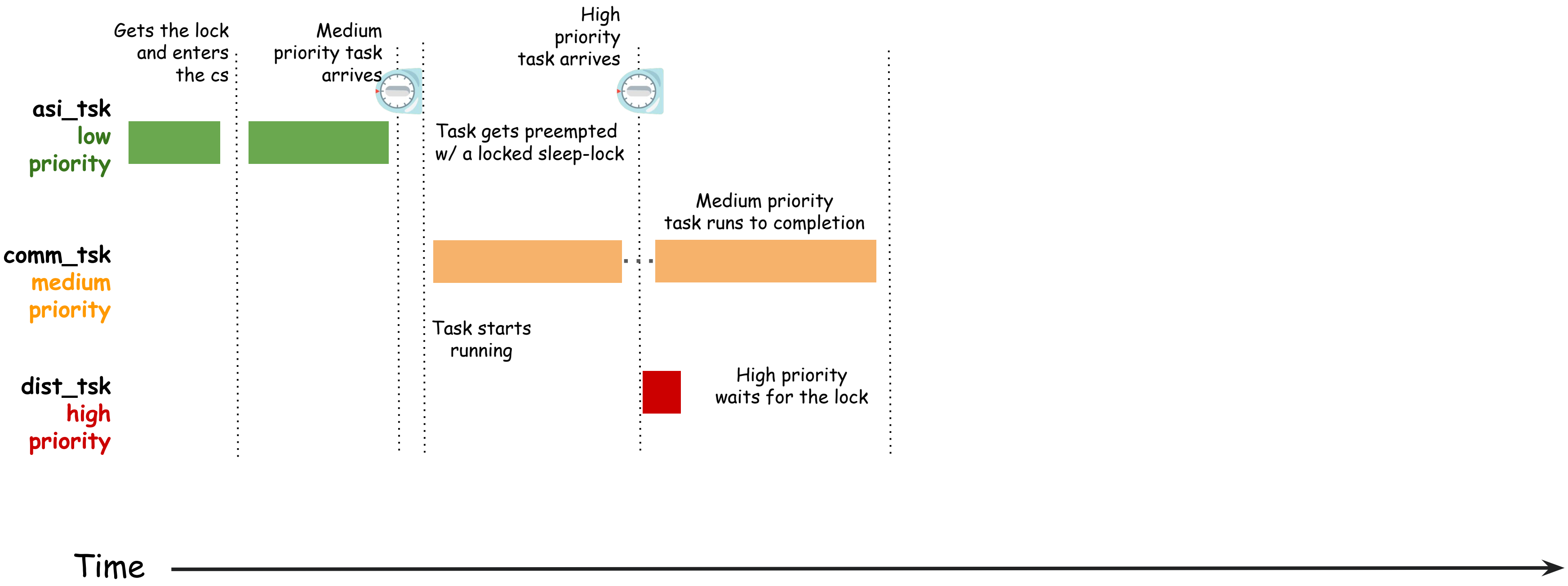
Ready for your first bug in the outer universe?



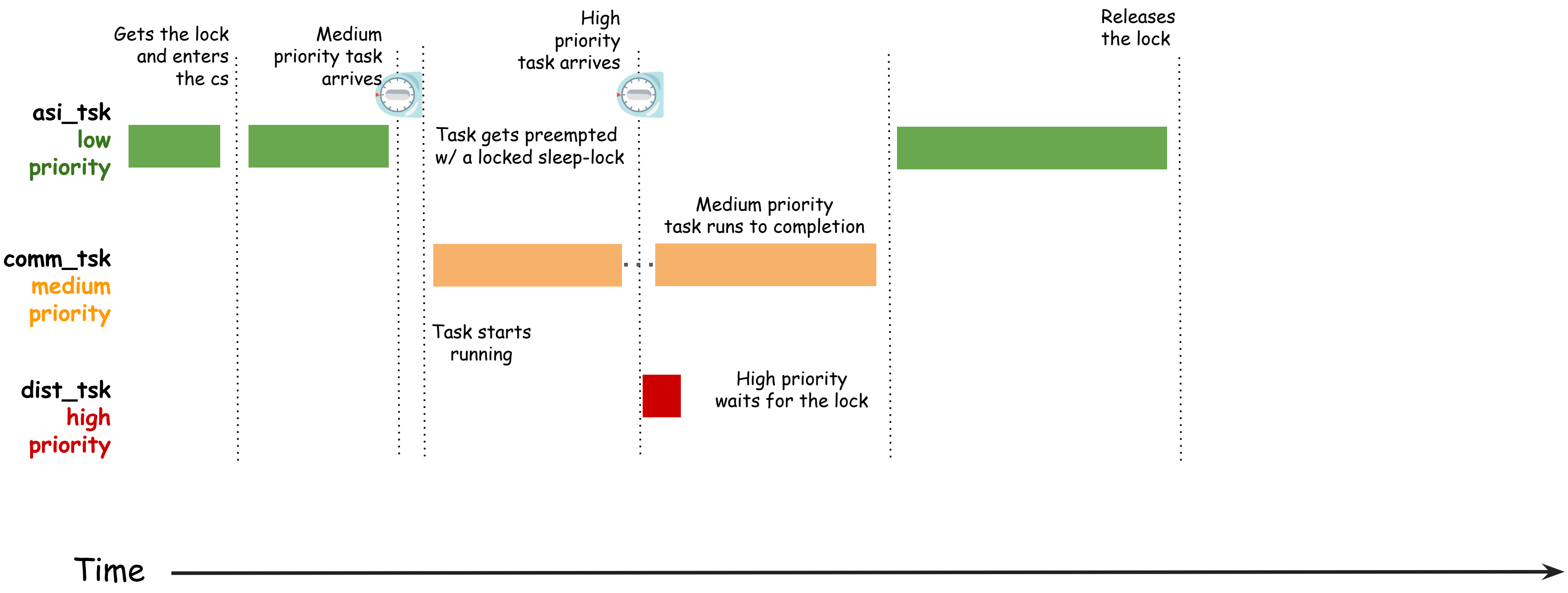
Ready for your first bug in the outer universe?



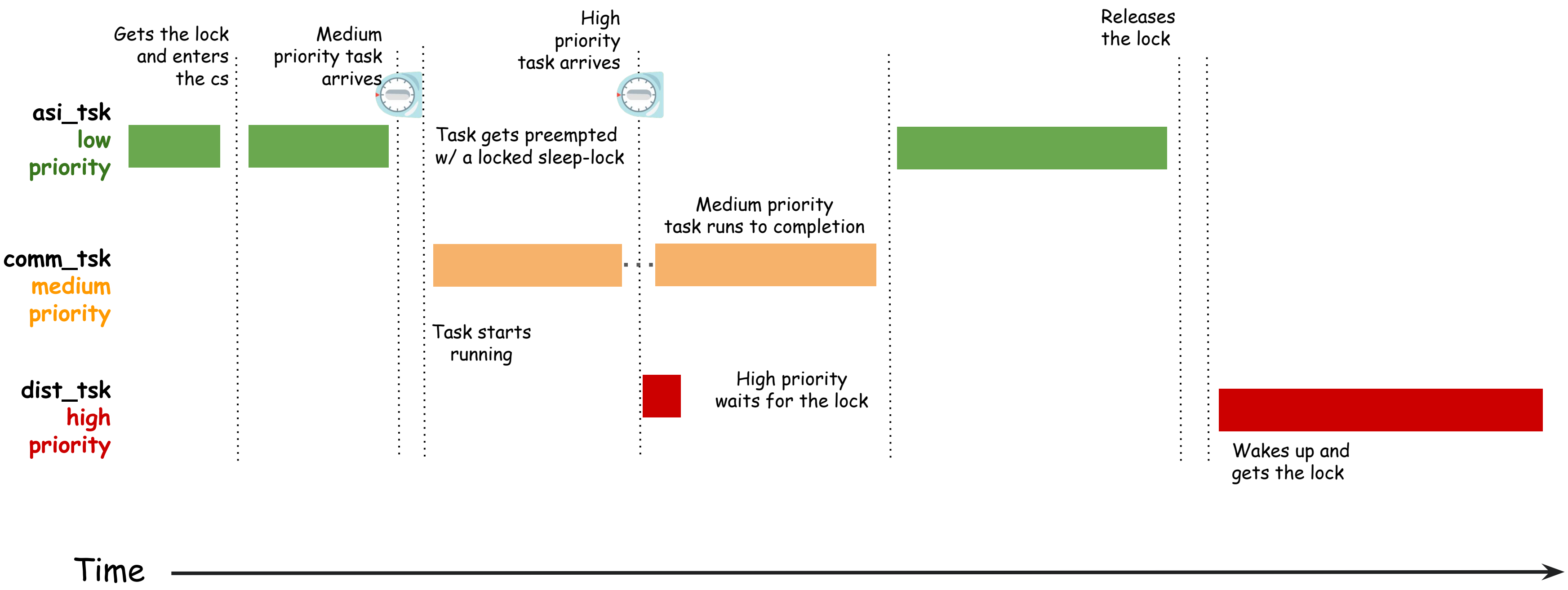
Ready for your first bug in the outer universe?



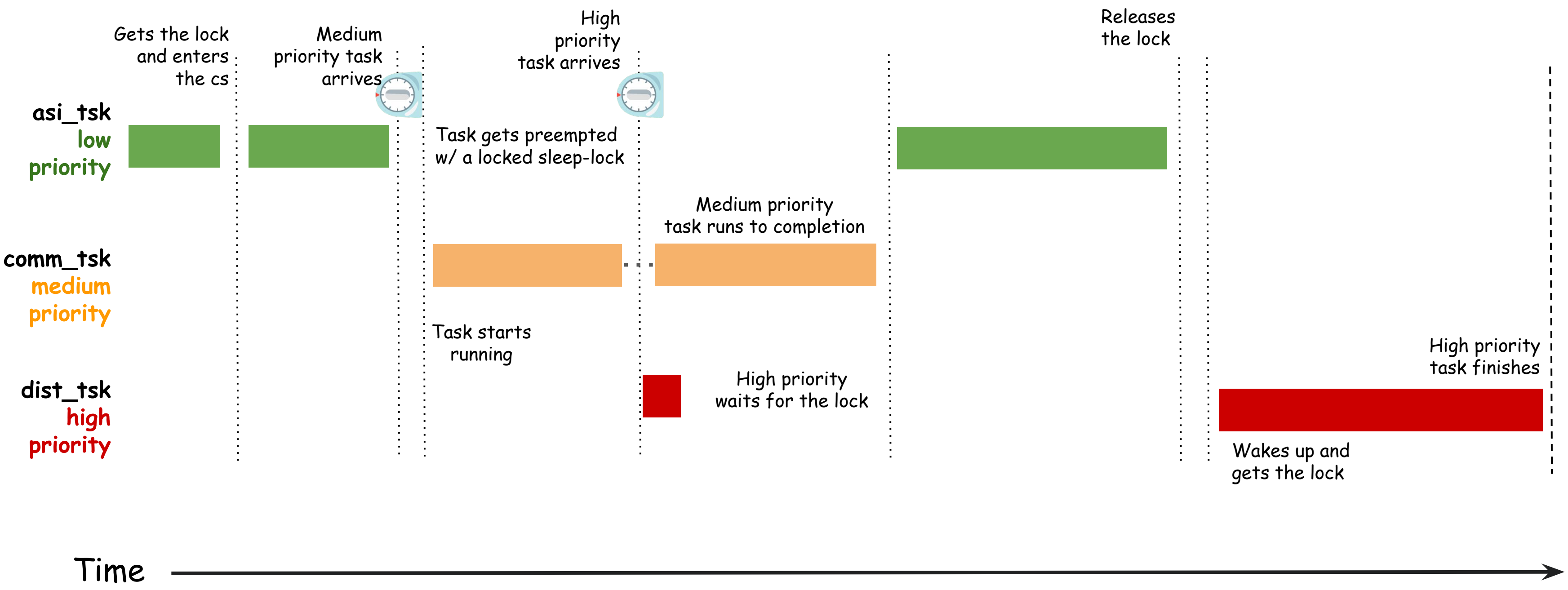
Ready for your first bug in the outer universe?



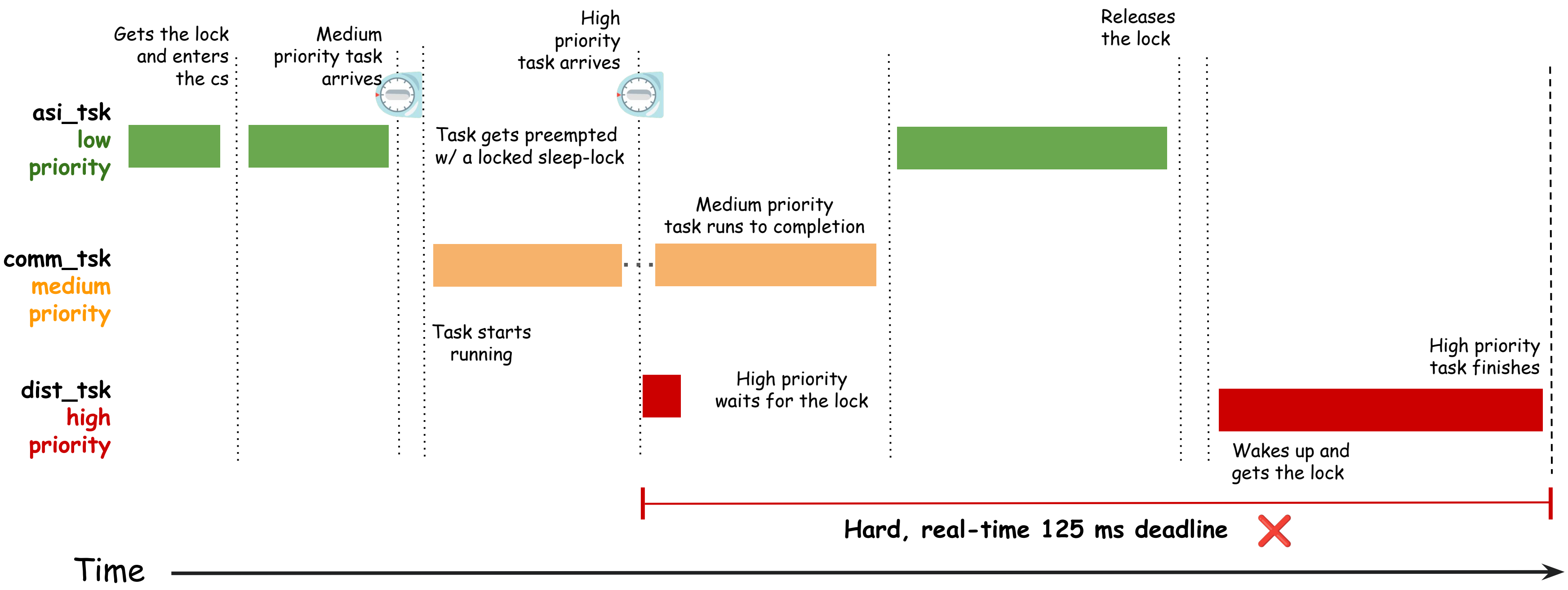
Ready for your first bug in the outer universe?



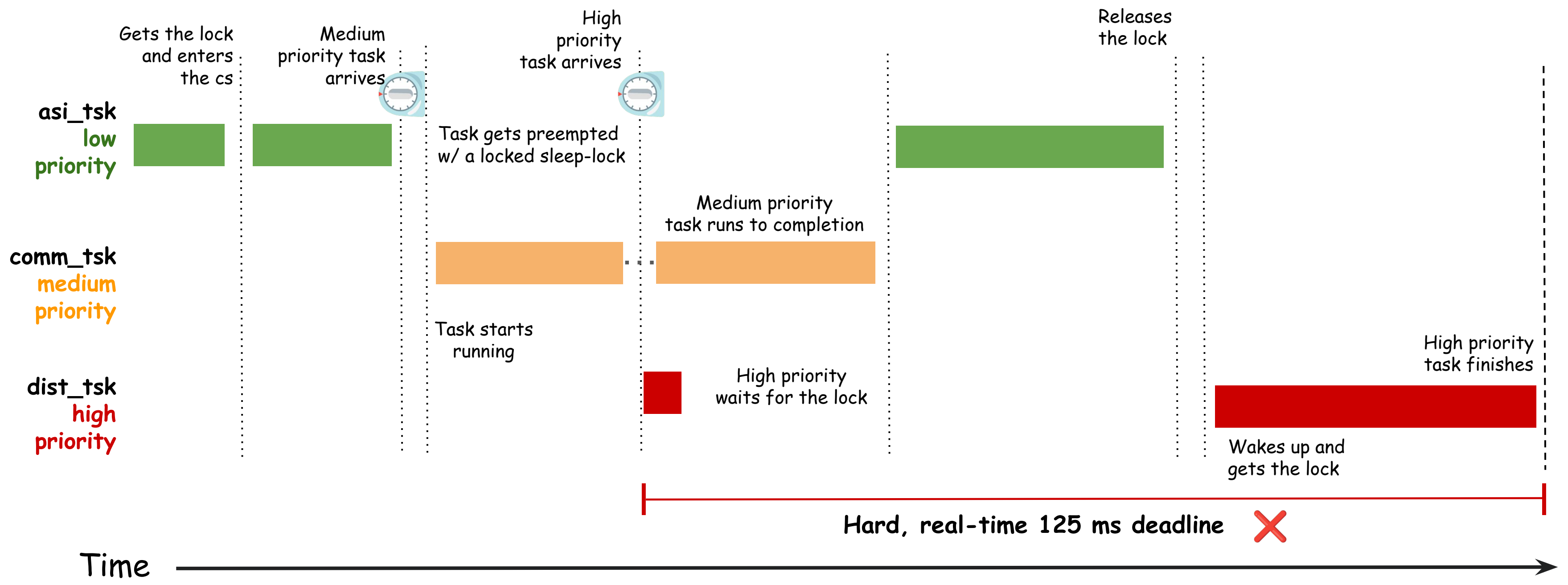
Ready for your first bug in the outer universe?



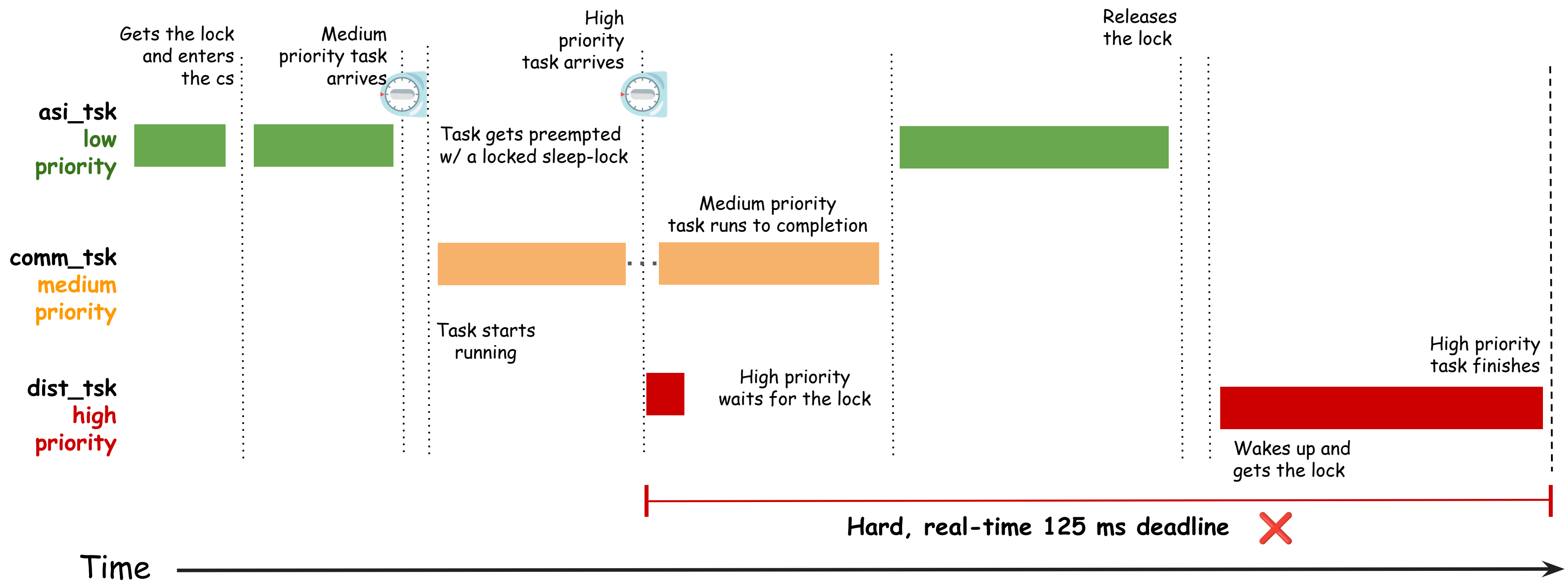
Ready for your first bug in the outer universe?



Classic example of priority inversion bug



Solution?



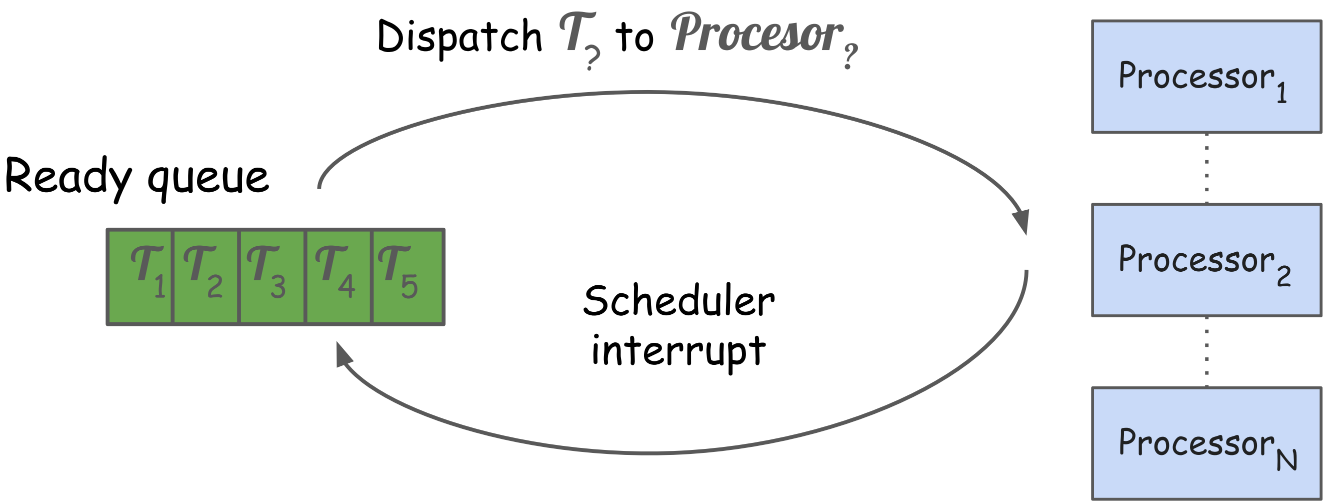
Gets the lock and enters the cs
Medium priority task arrives
High priority task arrives
Releases the lock
asi_tsk
low priority
Task gets preempted w/ a locked sleep-lock
Medium priority task runs to completion
comm_tsk
medium priority
Task starts running
High priority task finishes
dist_tsk
high priority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

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