

# Lecture 4

# CPU Scheduling

Prof. Yinqian Zhang

Fall 2025

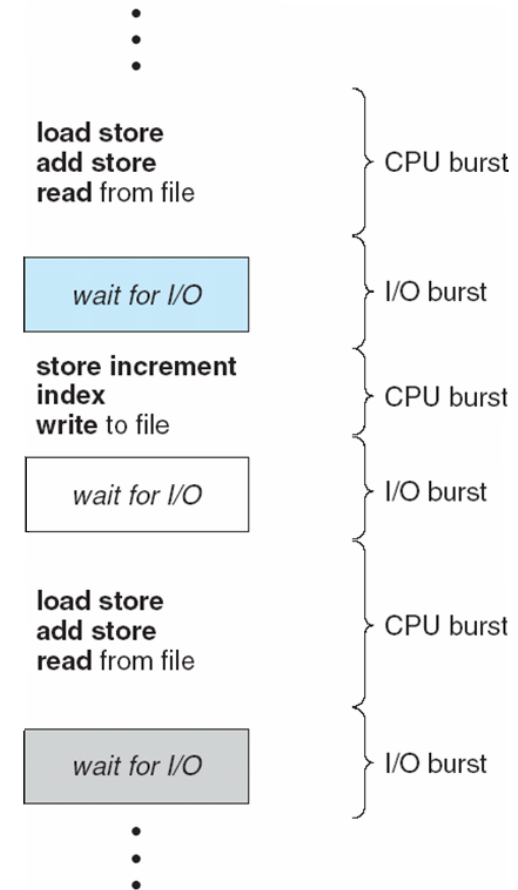
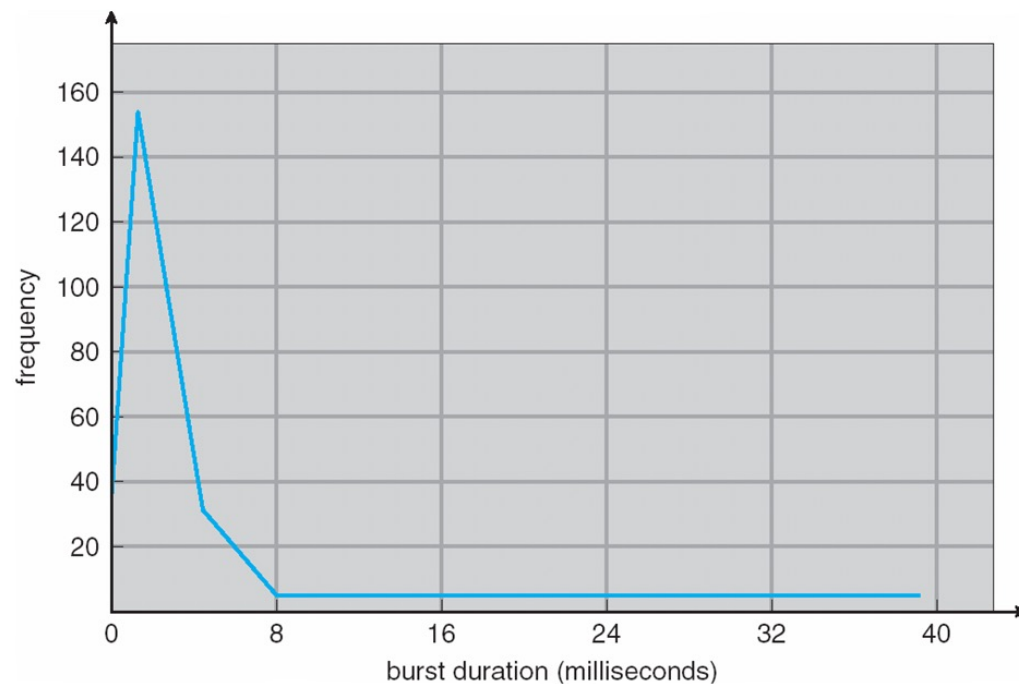
# CPU Scheduling

- Scheduling is important when multiple processes wish to run on a single CPU
  - CPU scheduler decides which process to run next
- Two types of processes
  - CPU bound and I/O bound

CPU-bound Process	I/O-bound process
Spends most of its running time on the CPU, i.e., <b>user-time</b> > <b>sys-time</b>	Spends most of its running time on I/O, i.e., <b>sys-time</b> > <b>user-time</b>
<u>Examples</u> - AI course assignments.	<u>Examples</u> - <b>/bin/l</b> s, networking programs.

# CPU Burst

- Process execution consists of a *cycle* of CPU execution and I/O wait
- **CPU burst** distribution



# CPU Scheduler

- CPU scheduler selects one of the processes that are ready to execute and allocates the CPU to it
- 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
- A scheduling algorithm takes place **only** under circumstances 1 and 4 is **non-preemptive**
- All other scheduling algorithms are **preemptive**

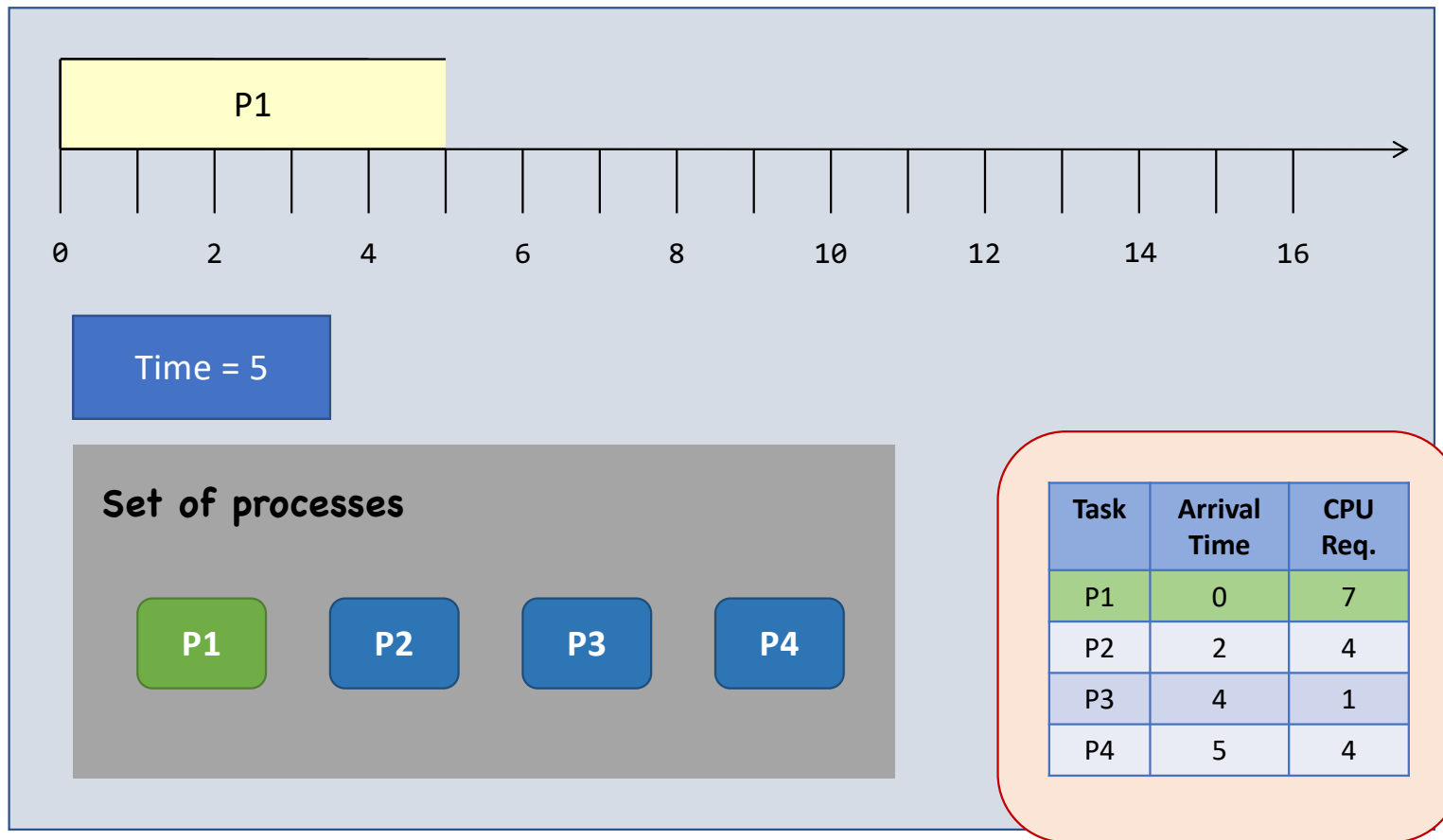
# Scheduling Algorithm Optimization Criteria

- Given a set of processes, with
  - **Arrival time:** the time they arrive in the CPU ready queue (from waiting state or from new state)
  - **CPU requirement:** their expected CPU burst time
- Minimize average turnaround time
  - **Turnaround time:** The time between the arrival of the task and the time it is blocked or terminated.
- Minimize average waiting time
  - **Waiting time:** The accumulated time that a task has waited in the ready queue.
- Reduce the number of context switches

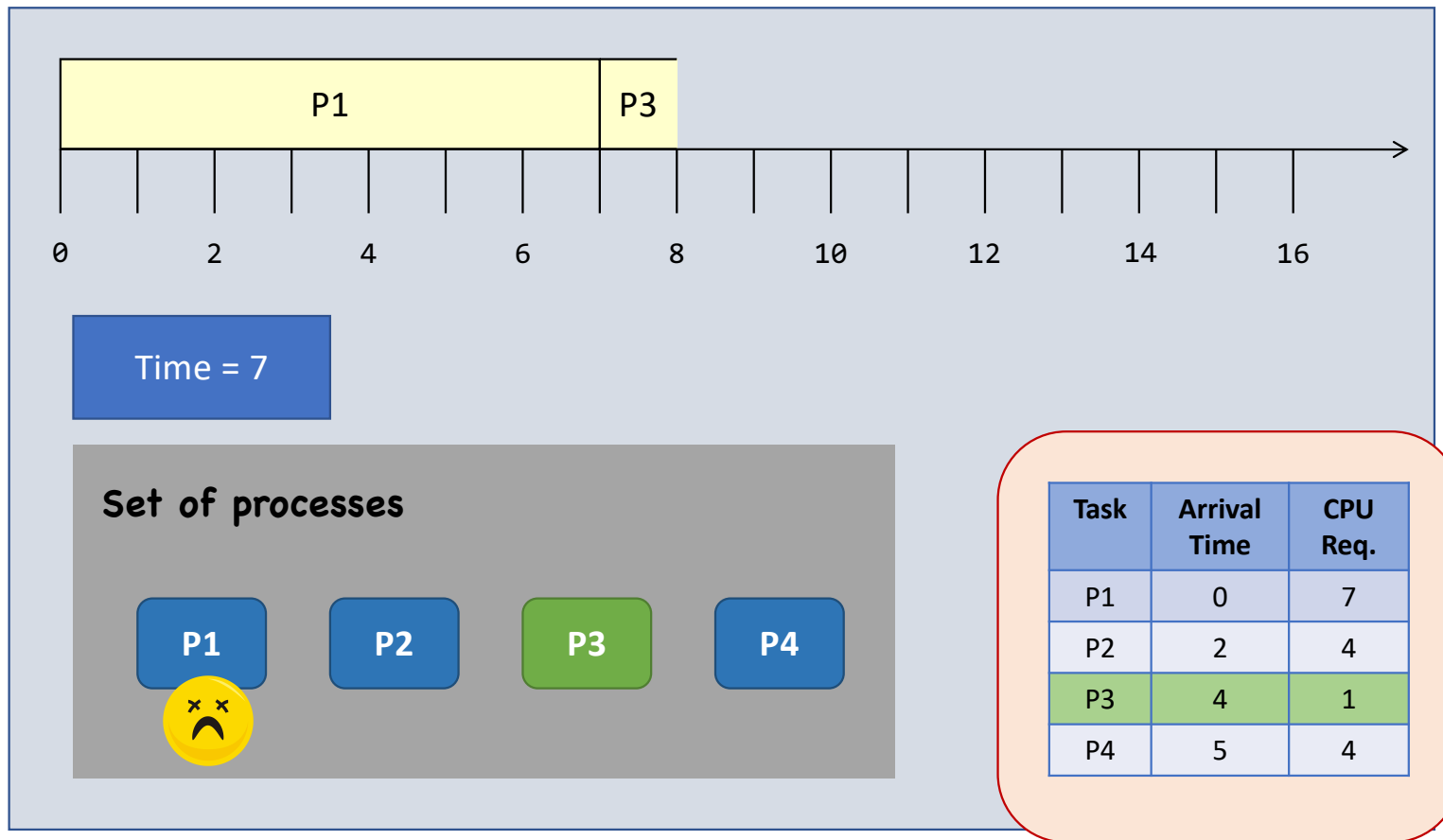
# Different Algorithms

- Shortest-job-first (SJF)
- Round-robin (RR)
- Priority scheduling

# Non-preemptive SJF



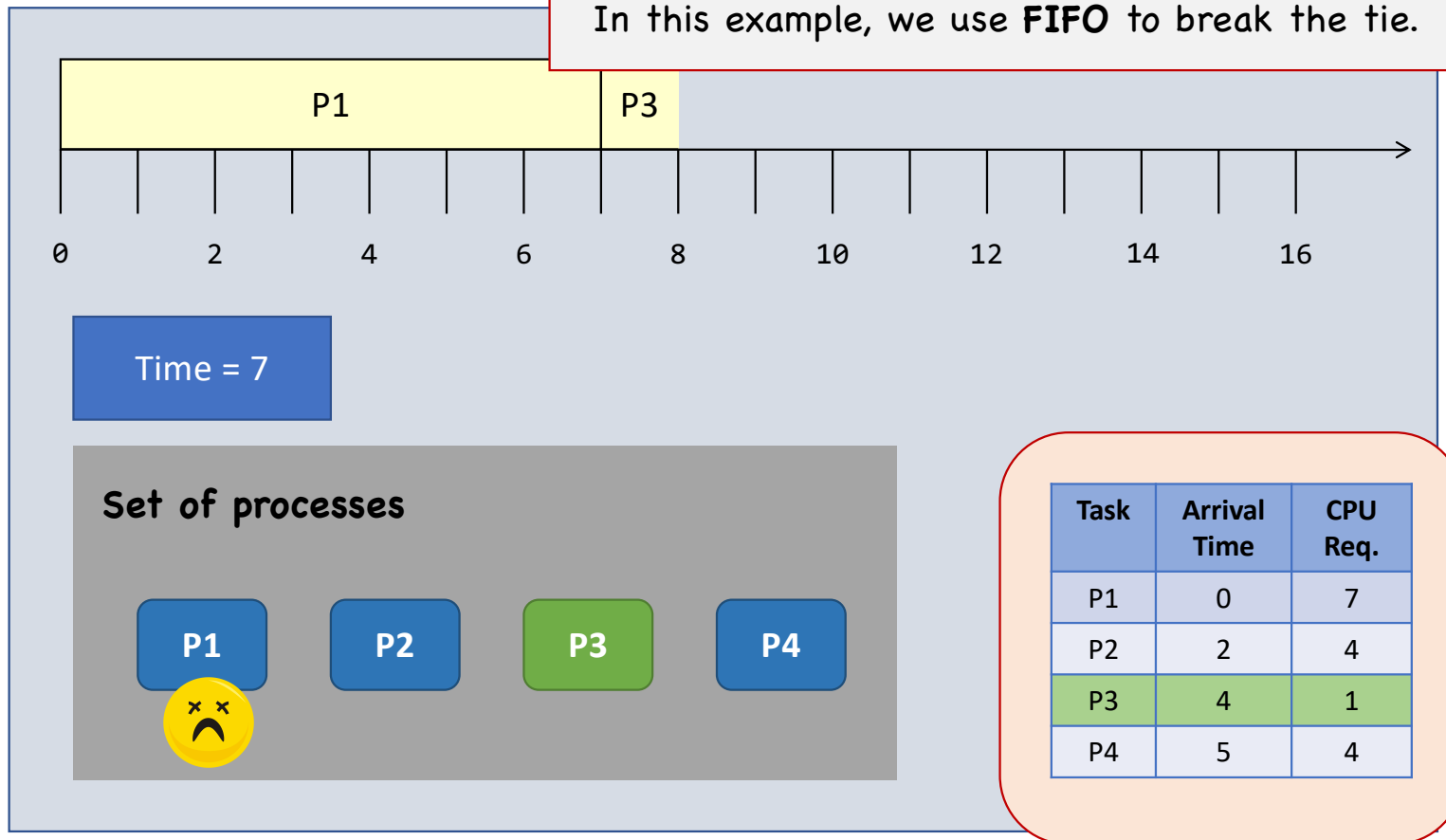
# Non-preemptive SJF



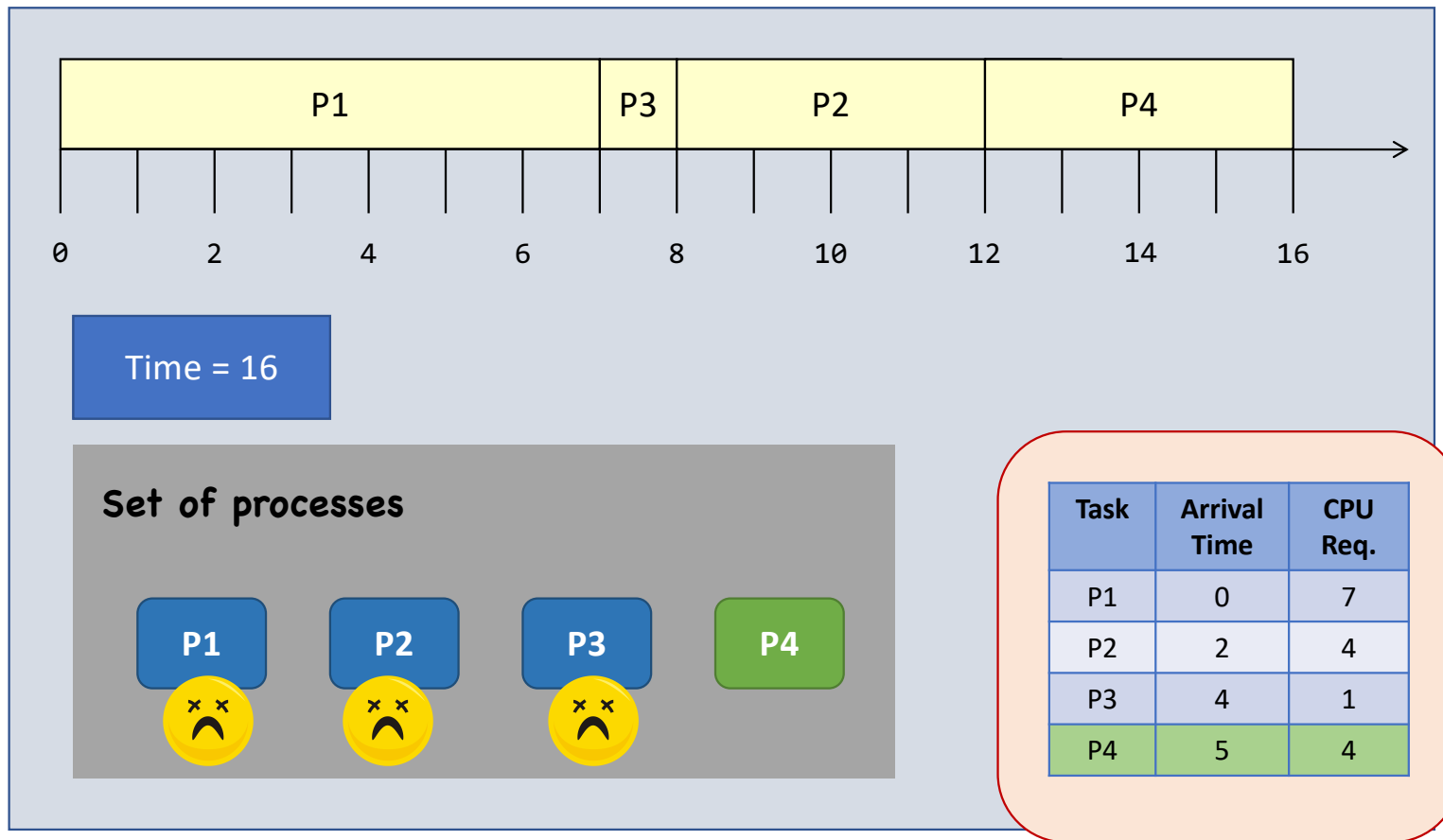


# Non-preemptive SJF

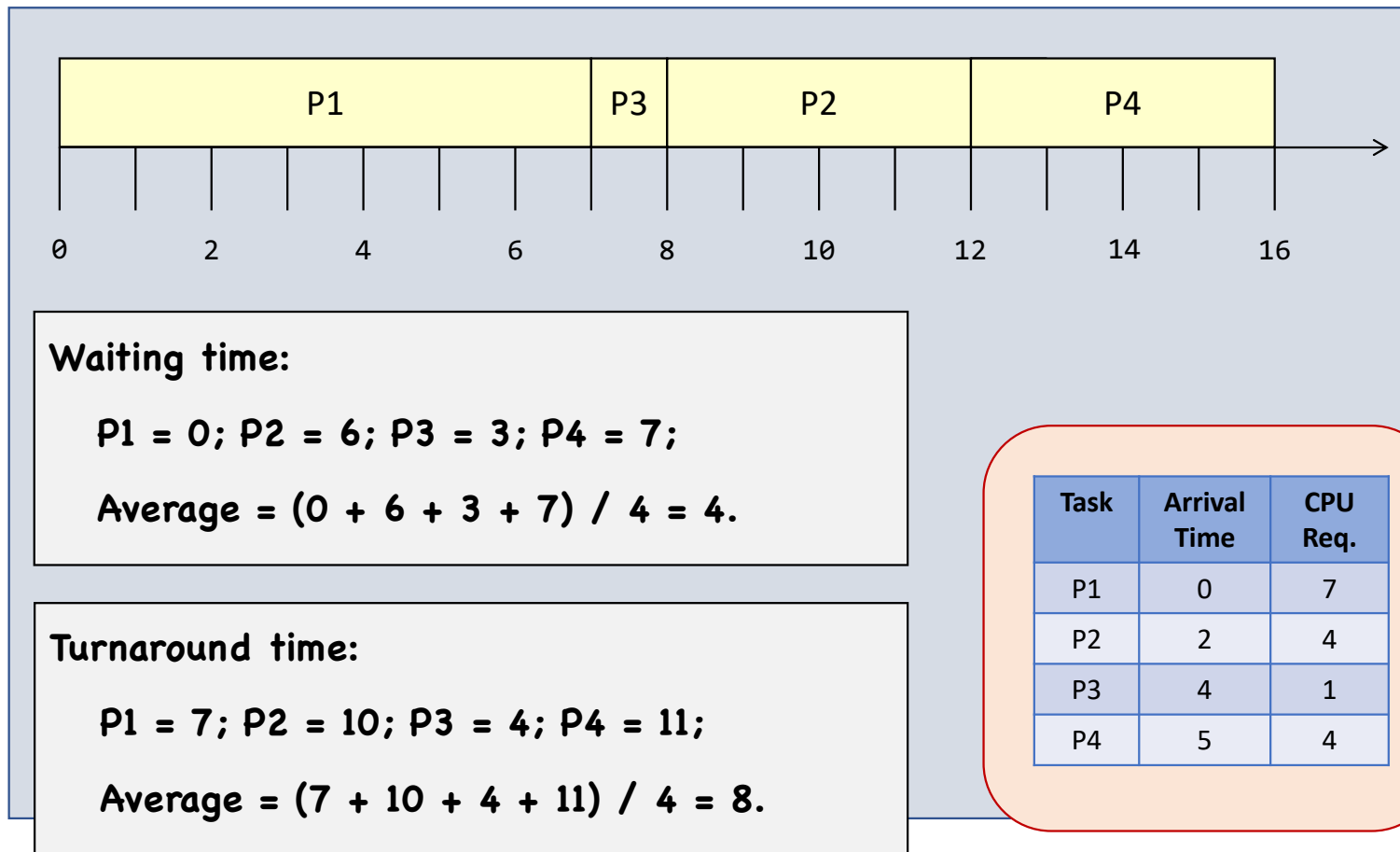
In this example, we use **FIFO** to break the tie.



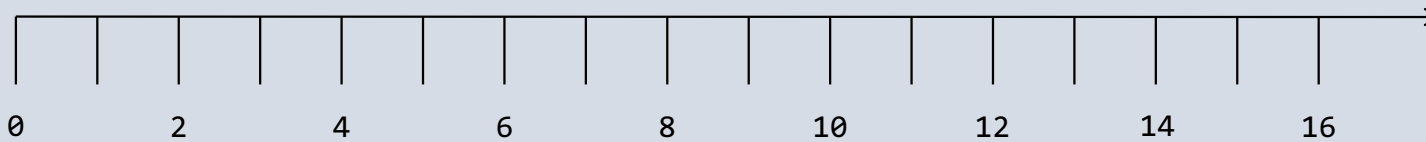
# Non-preemptive SJF



# Non-preemptive SJF



# Preemptive SJF

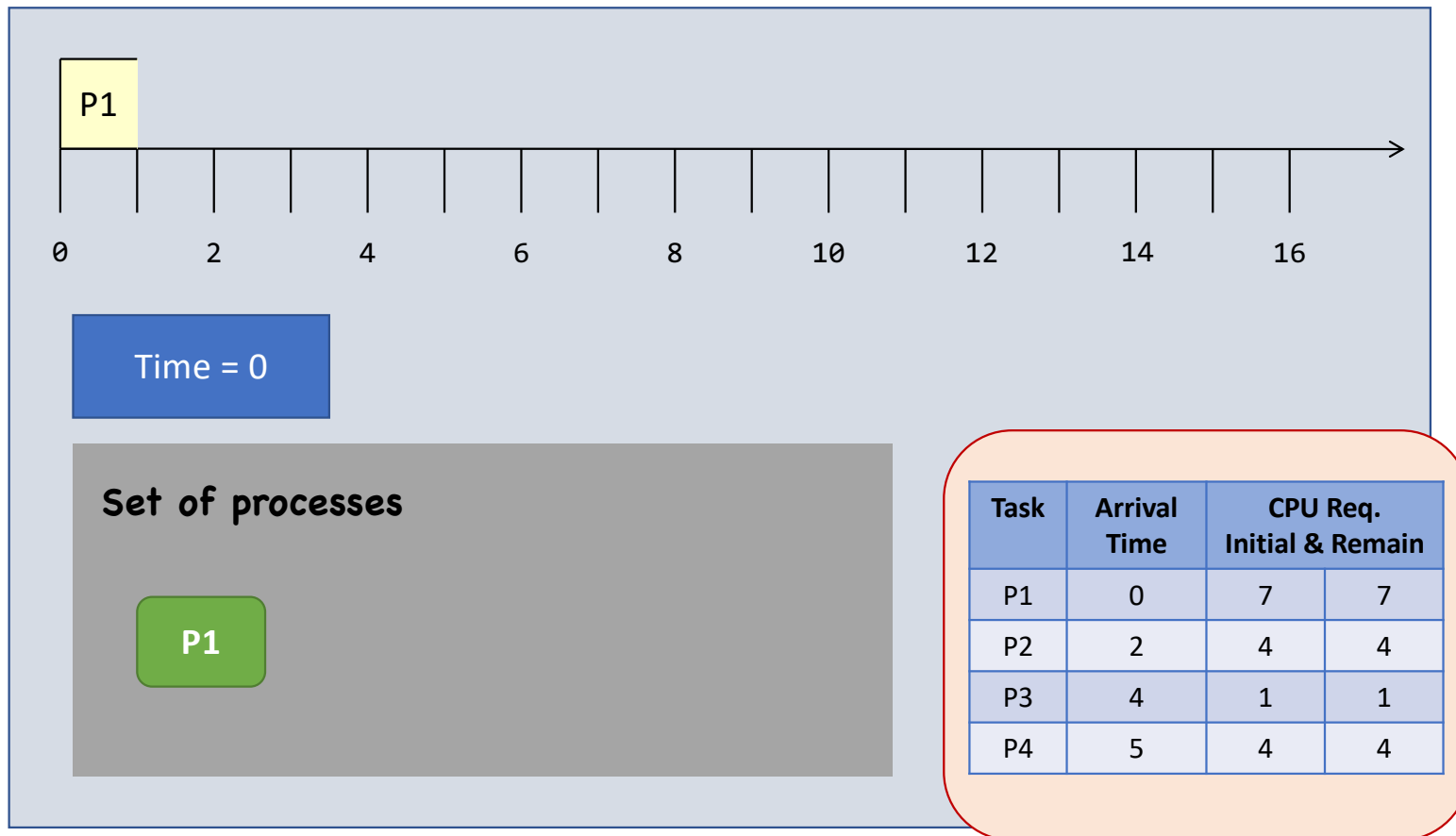


Whenever a new process arrives in the ready queue (either from waiting or from new state), the scheduler steps in and selects the next task based on **their remaining CPU requirements**.

Task	Arrival Time	CPU Req.	
		Initial	Remain
P1	0	7	7
P2	2	4	4
P3	4	1	1
P4	5	4	4

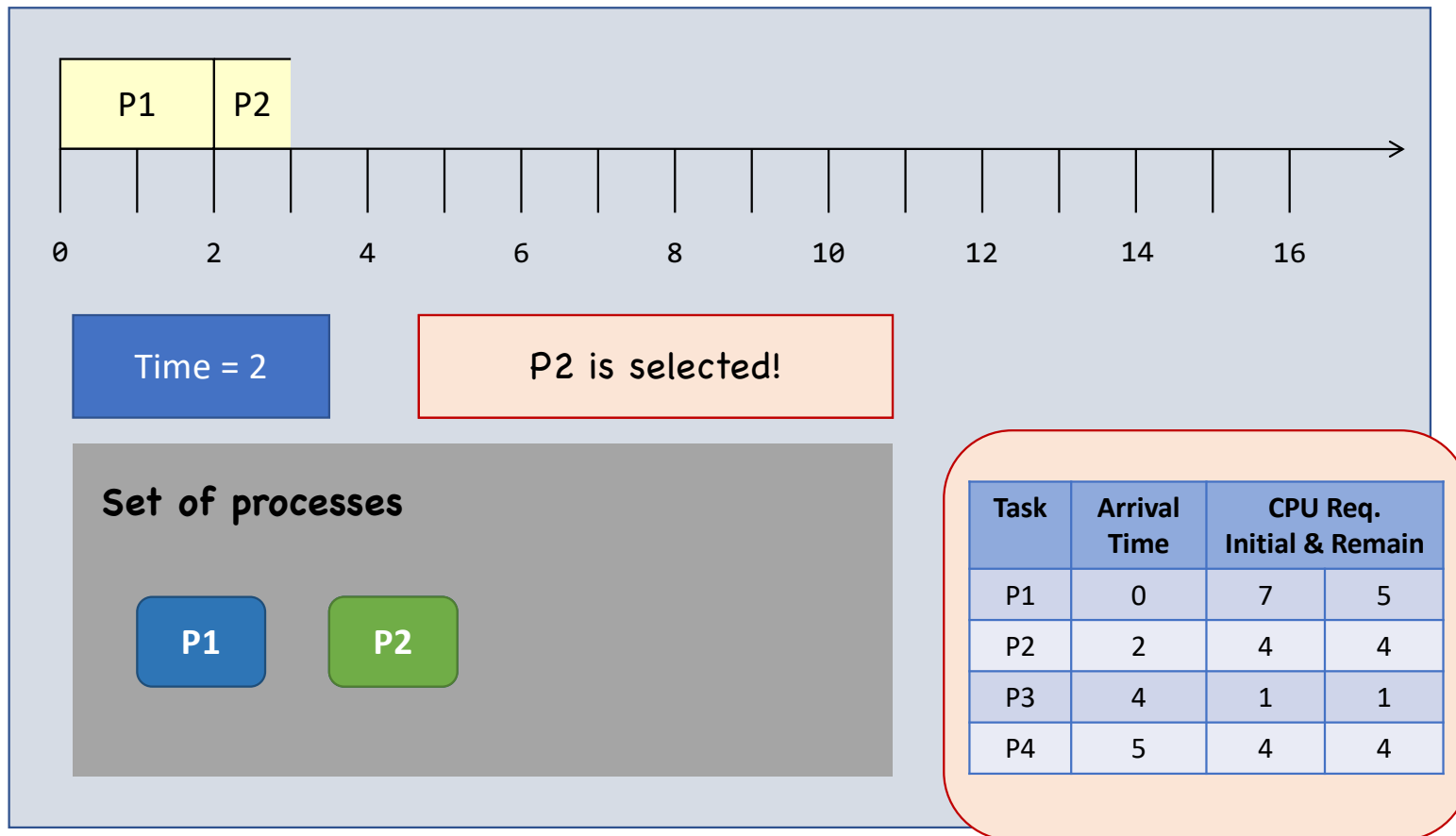
Animation; don't print

# Preemptive SJF



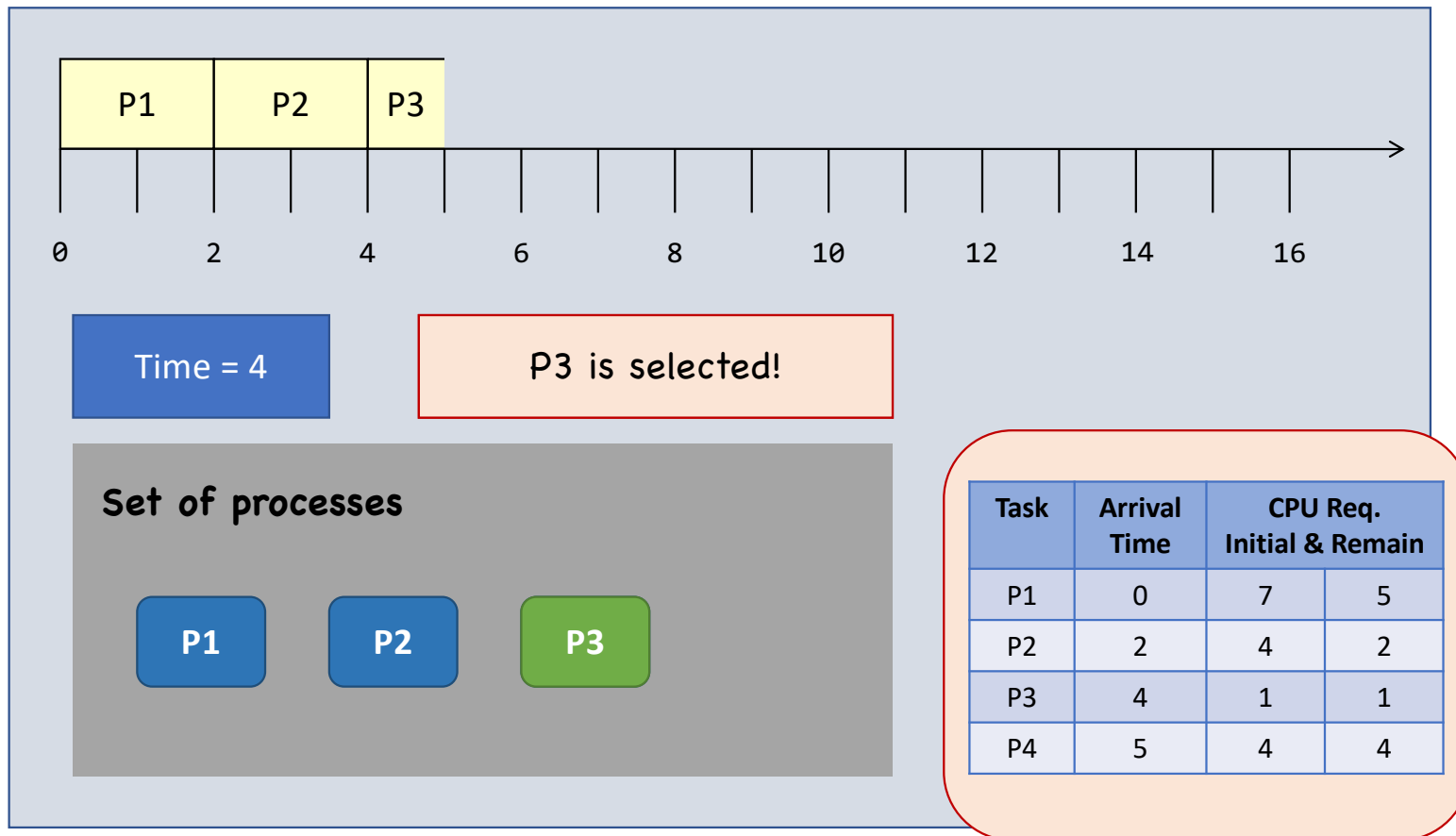
Animation; don't print

# Preemptive SJF



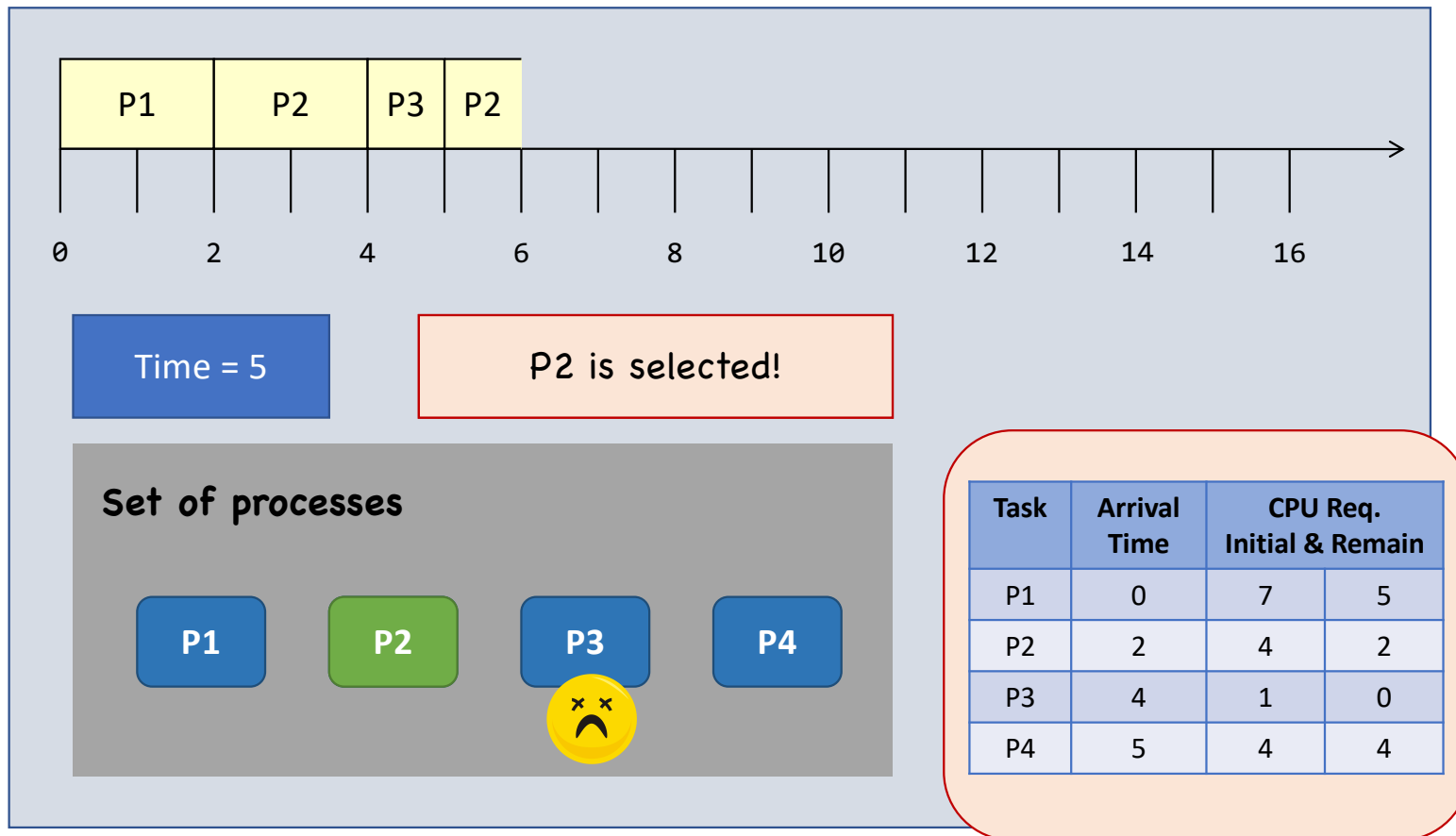
Animation; don't print

# Preemptive SJF



Animation; don't print

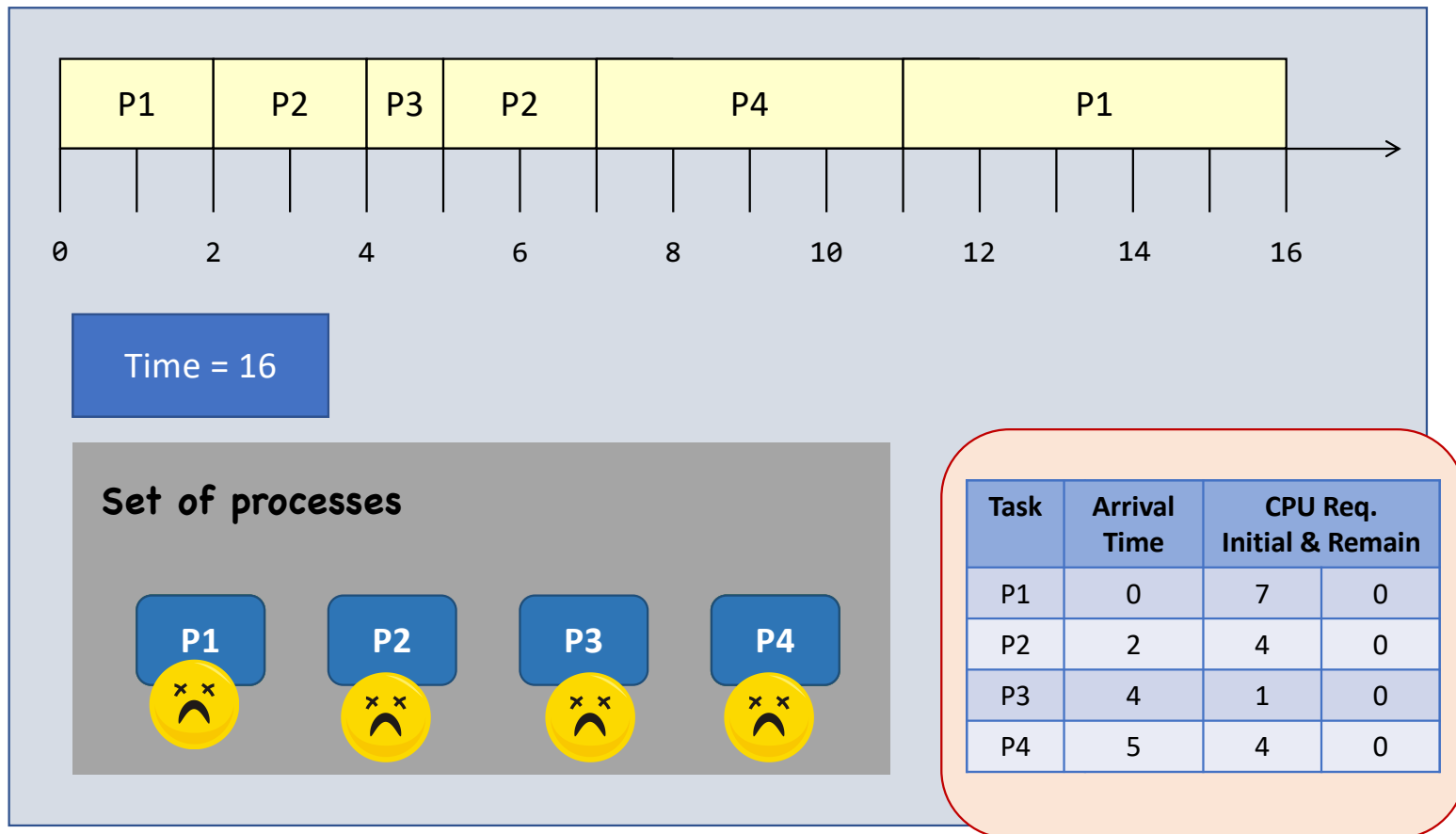
# Preemptive SJF



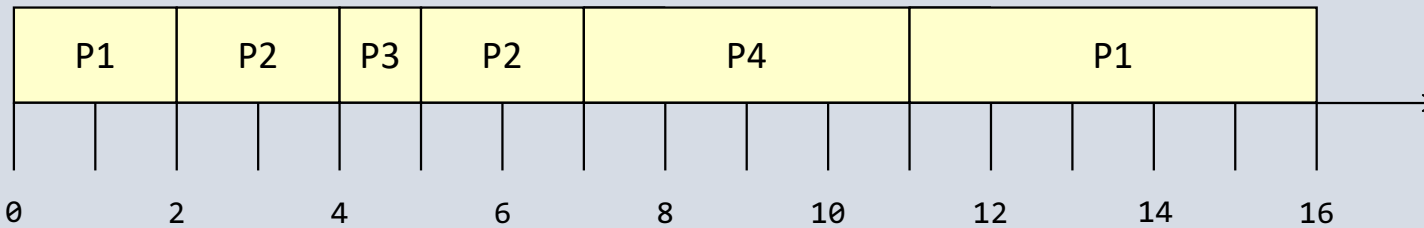


Animation; don't print

# Preemptive SJF



# Preemptive SJF



**Waiting time:**

$P1 = 9; P2 = 1; P3 = 0; P4 = 2;$

$Average = (9 + 1 + 0 + 2) / 4 = 3.$

**Turnaround time:**

$P1 = 16; P2 = 5; P3 = 1; P4 = 6;$

$Average = (16 + 5 + 1 + 6) / 4 = 7.$

Task	Arrival Time	CPU Req. Initial & Remain	
P1	0	7	0
P2	2	4	0
P3	4	1	0
P4	5	4	0

# SJF: Preemptive or Not?

	Non-preemptive SJF	Preemptive SJF
Average waiting time	4	3 (smallest)
Average turnaround time	8	7 (smallest)
# of context switching	3	5 (largest)

The waiting time and the turnaround time decrease at the expense of the increased number of context switches.

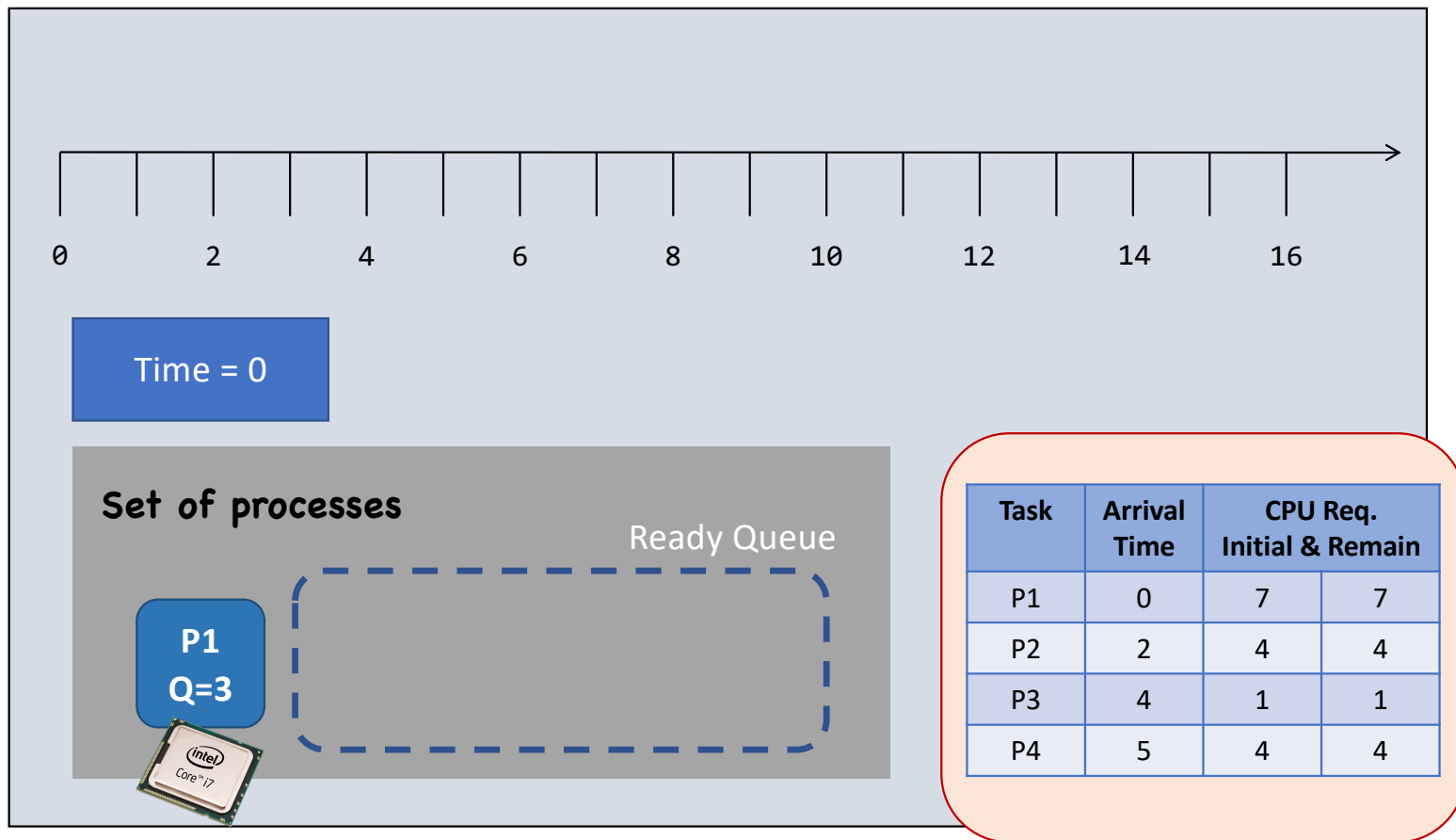
Task	Arrival Time	CPU Req.
P1	0	7
P2	2	4
P3	4	1
P4	5	4

# Round Robin (RR)

- Round-Robin (RR) scheduling is preemptive.
  - Every process is given a **quantum** (the amount of time allowed to execute).
  - Whenever the quantum of a process is used up (i.e., 0), the process is preempted, placed at the end of the queue, with its quantum re-charged
  - Then, the scheduler steps in and it chooses the next process which has a non-zero quantum to run.
  - Processes are therefore running one-by-one as a circular queue
- New processes are added to the tail of the ready queue
  - New process's arrival won't trigger a new selection decision

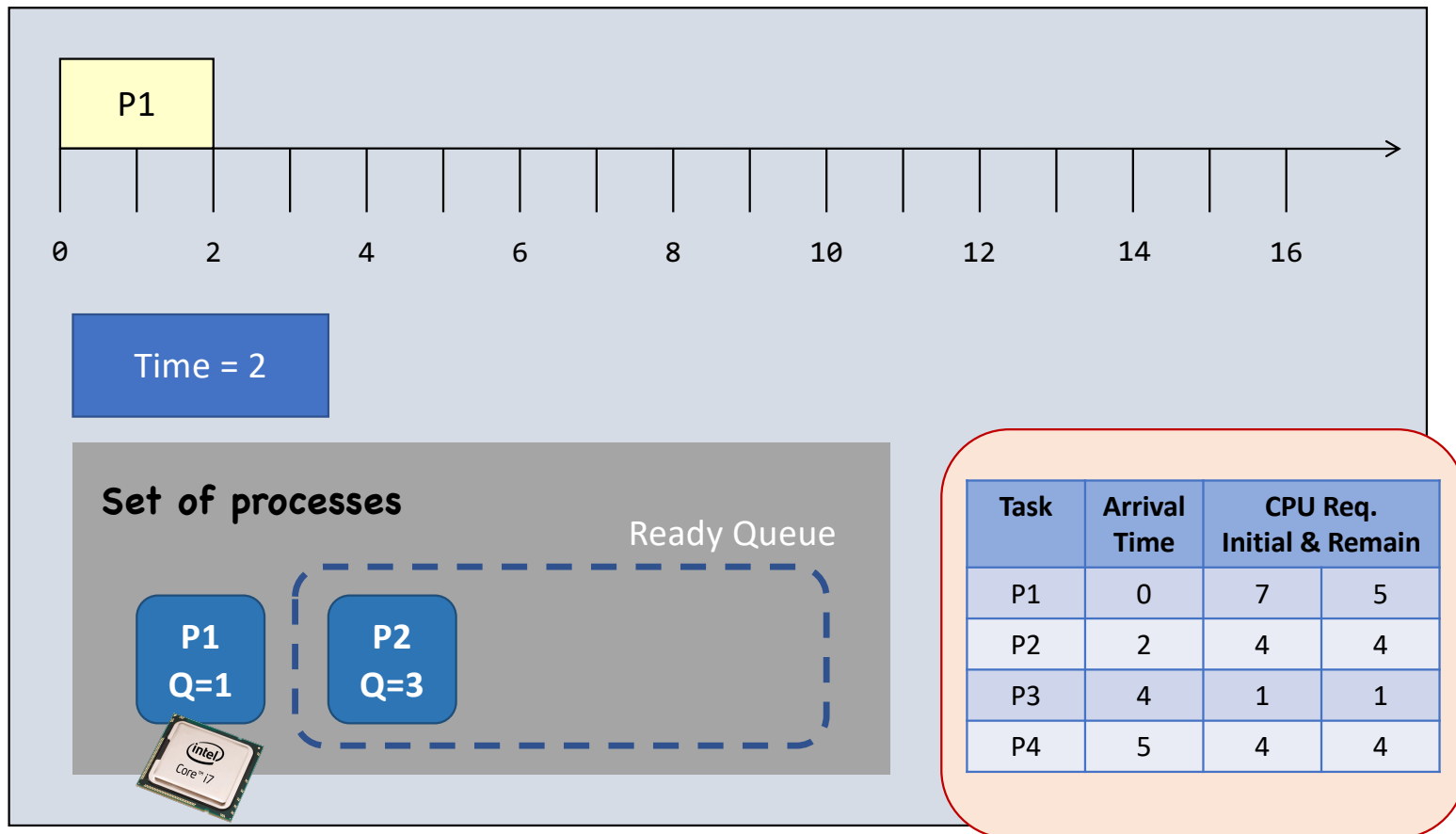
Animation; don't print

# Round Robin (Quantum = 3)



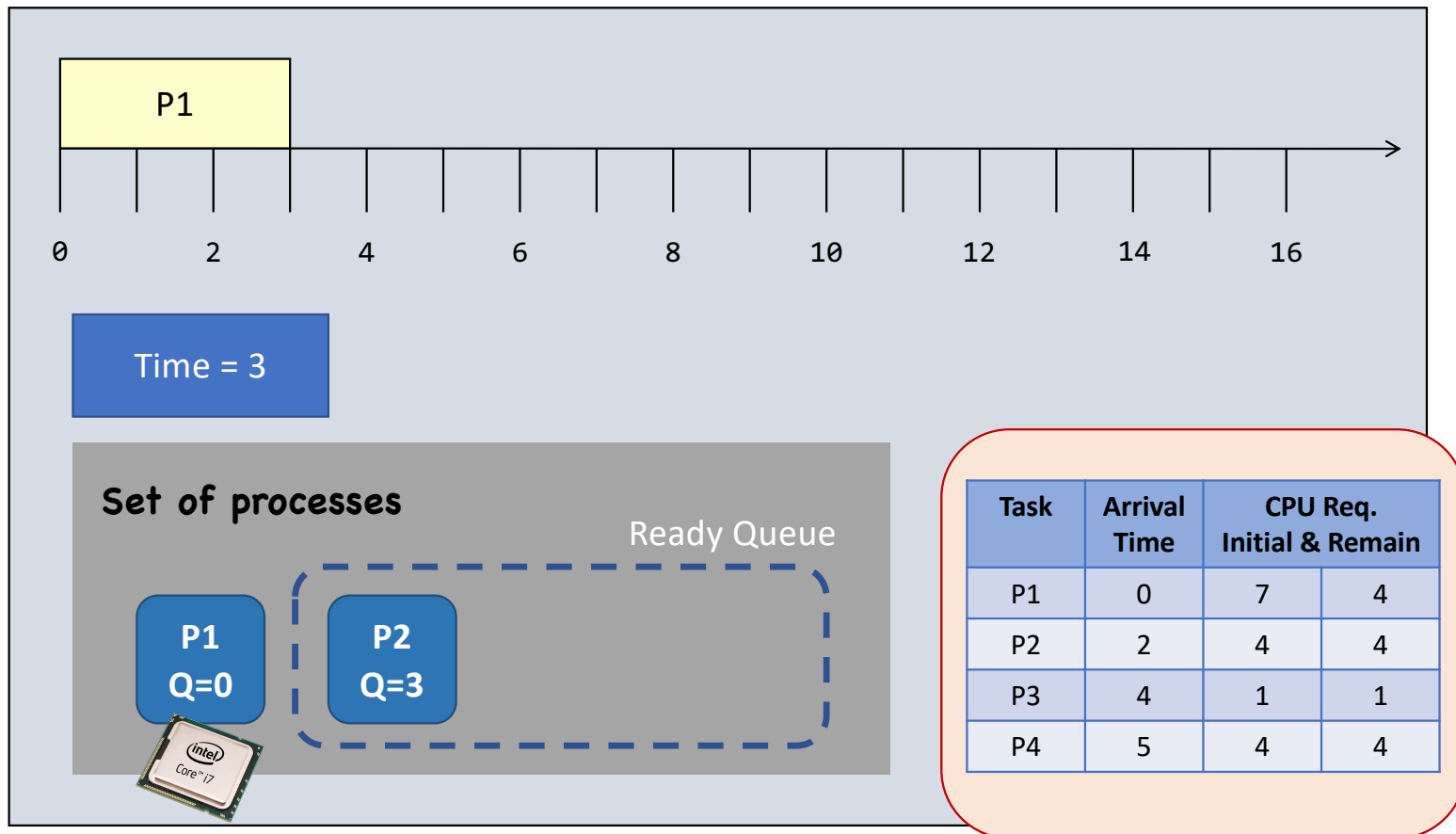
Animation; don't print

# Round Robin (Quantum = 3)



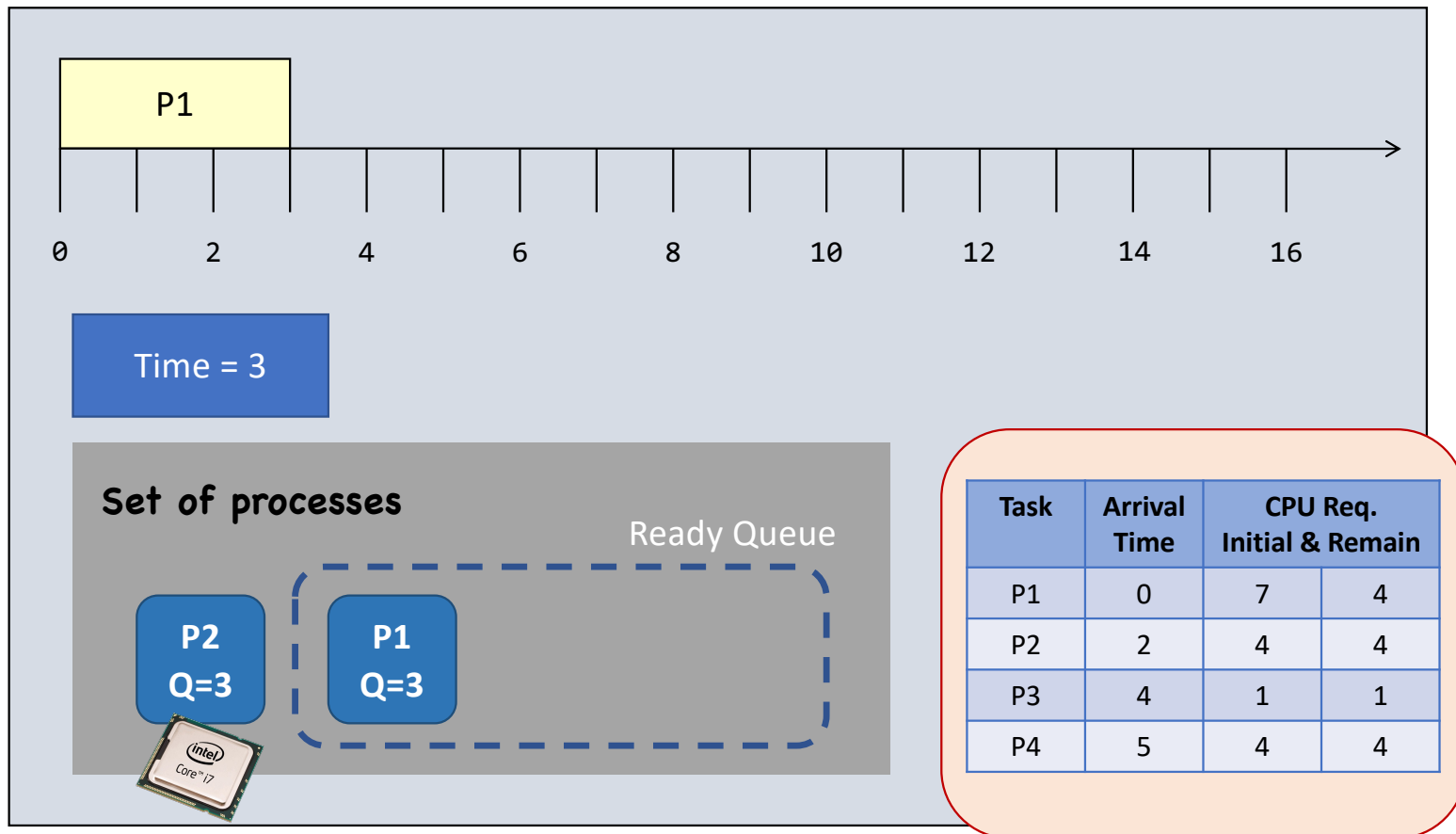
Animation; don't print

# Round Robin (Quantum = 3)



Animation; don't print

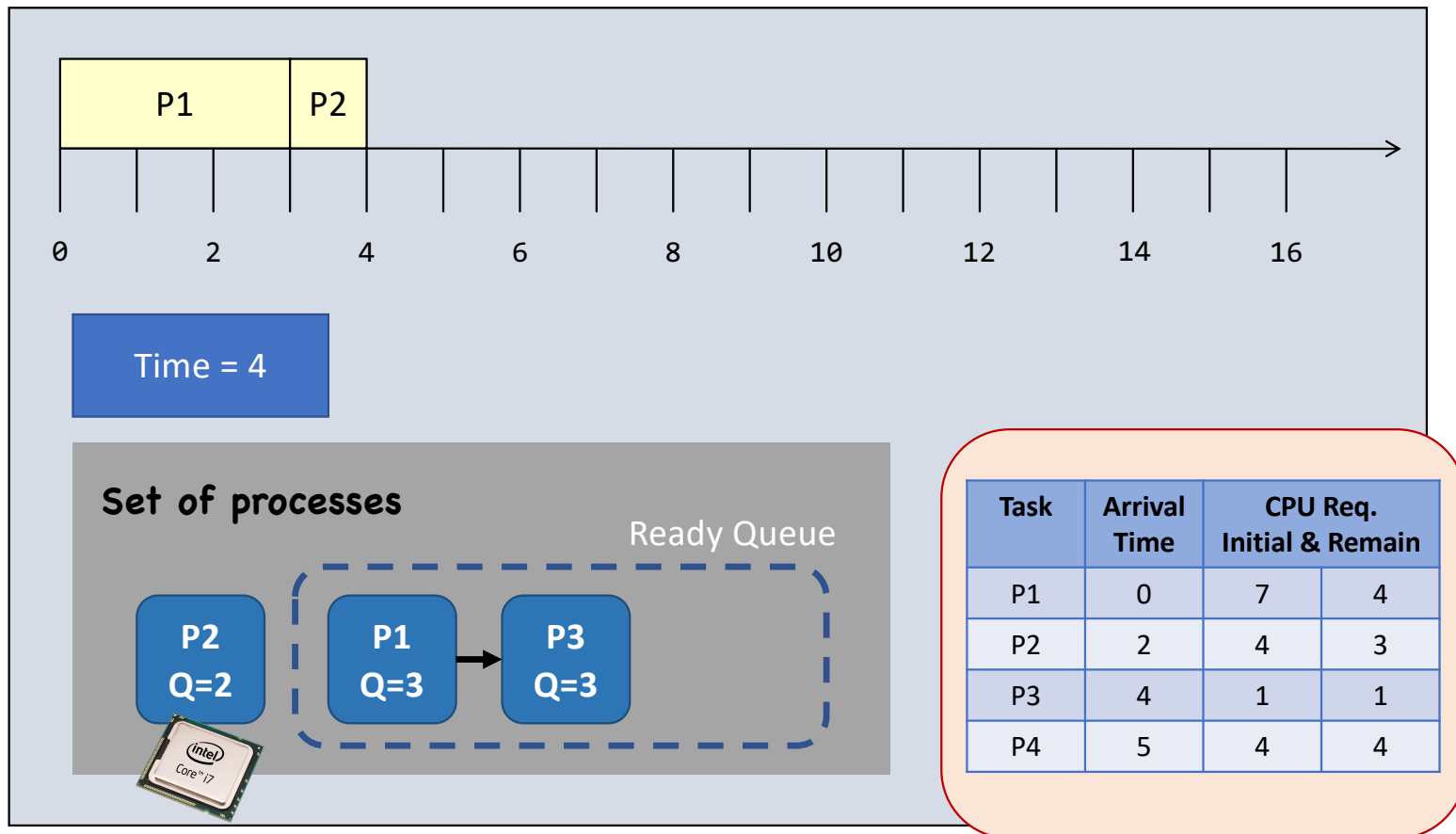
# Round Robin (Quantum = 3)





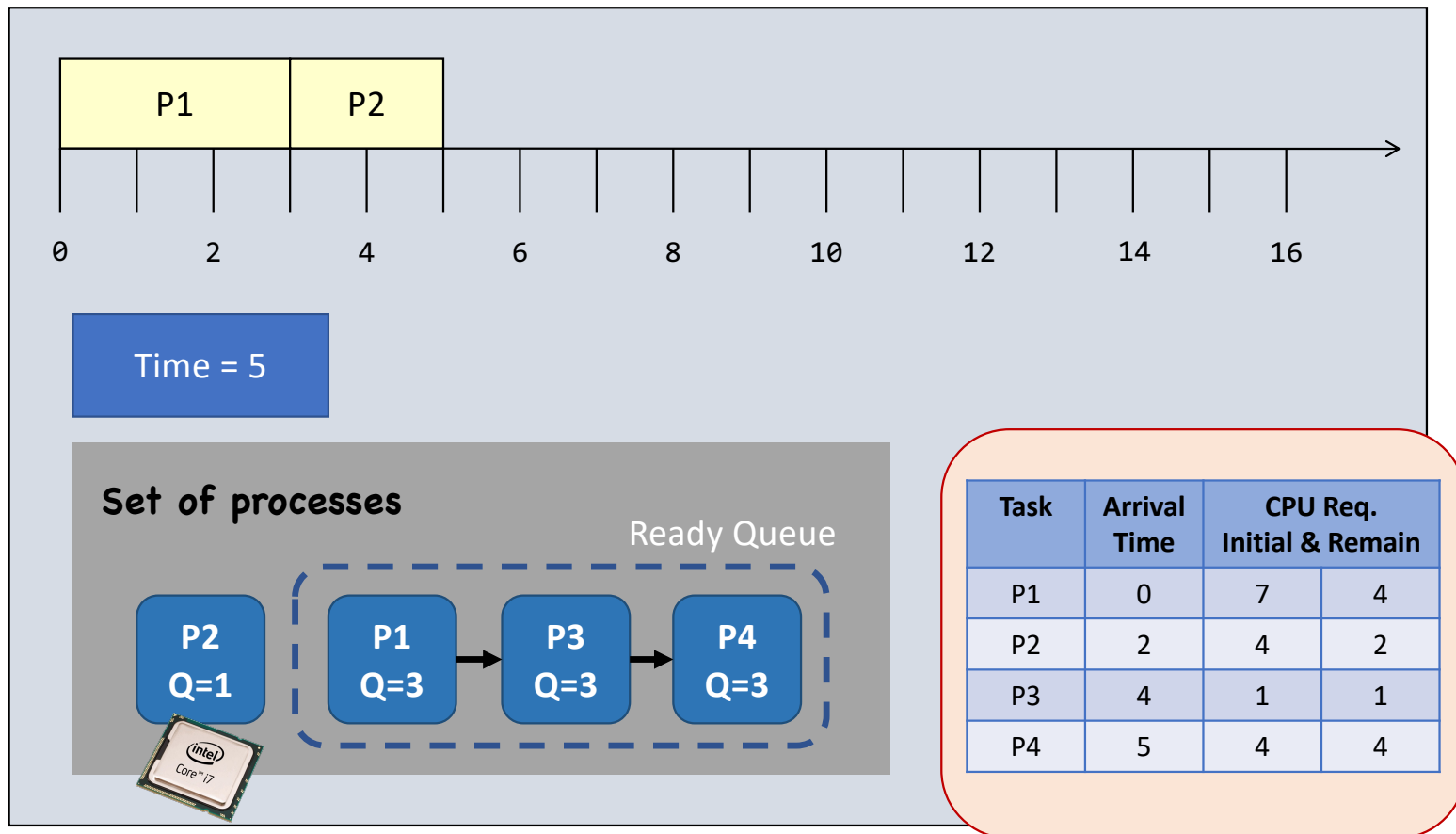
Animation; don't print

# Round Robin (Quantum = 3)



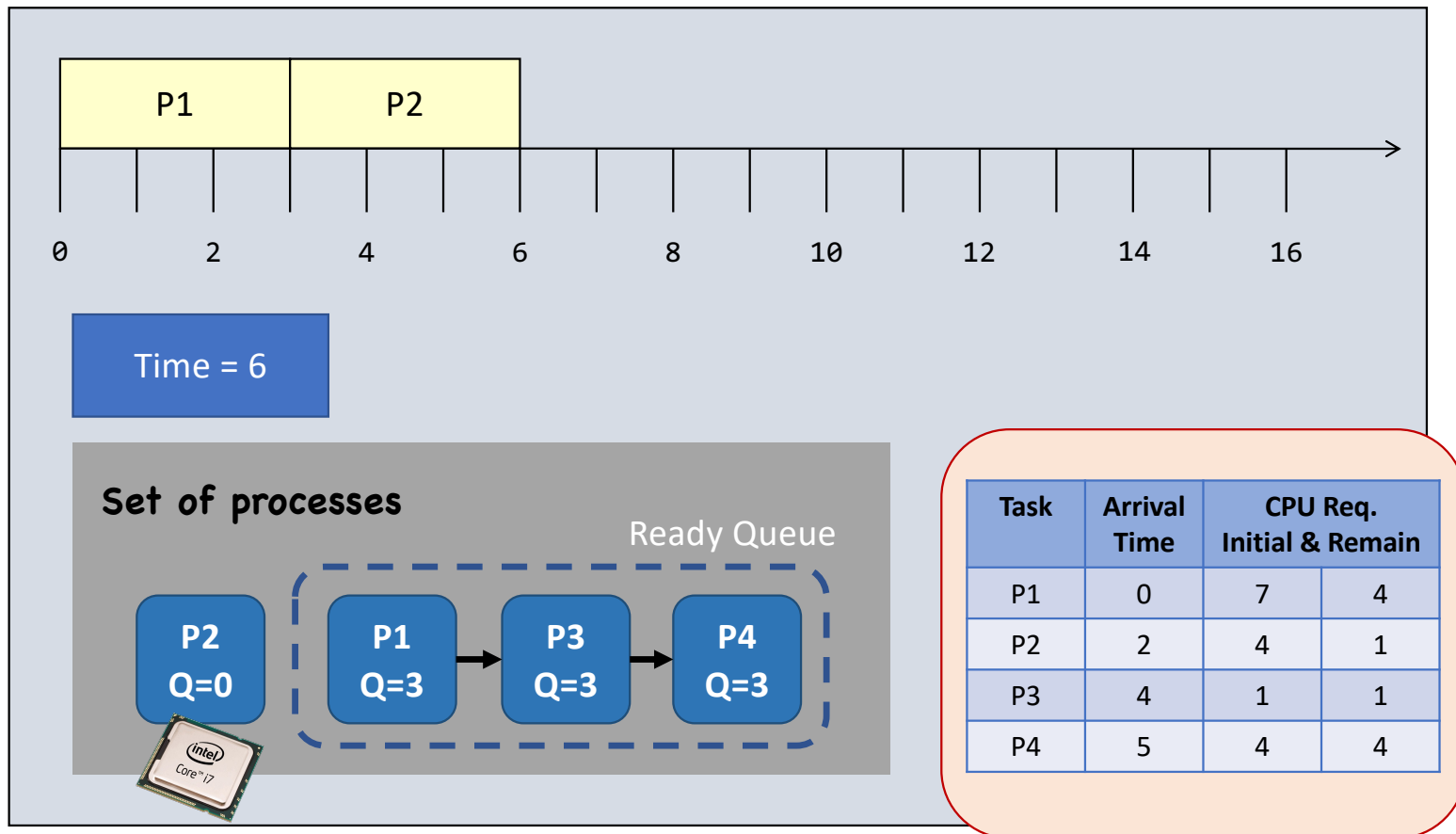
Animation; don't print

# Round Robin (Quantum = 3)



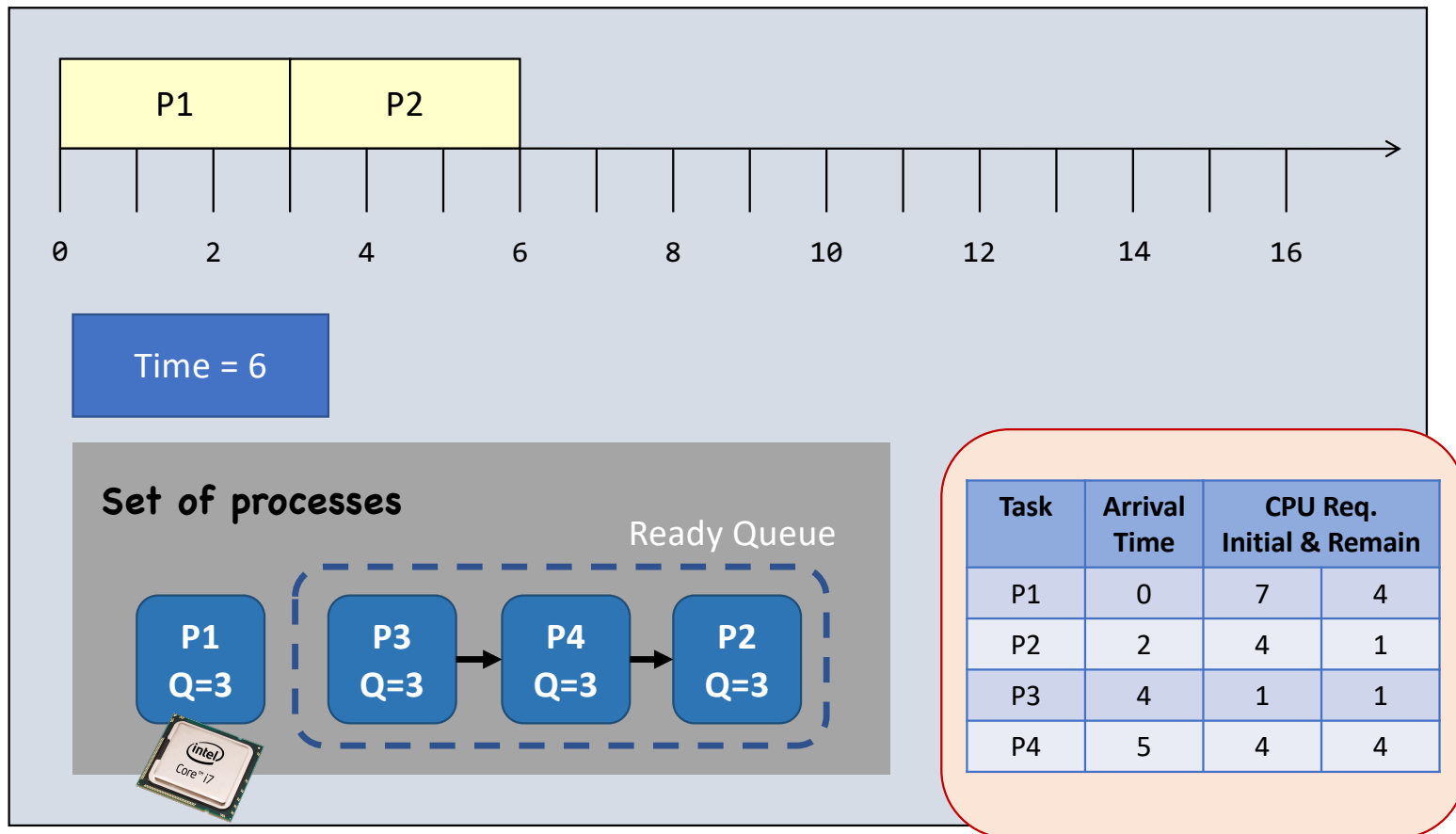
Animation; don't print

# Round Robin (Quantum = 3)



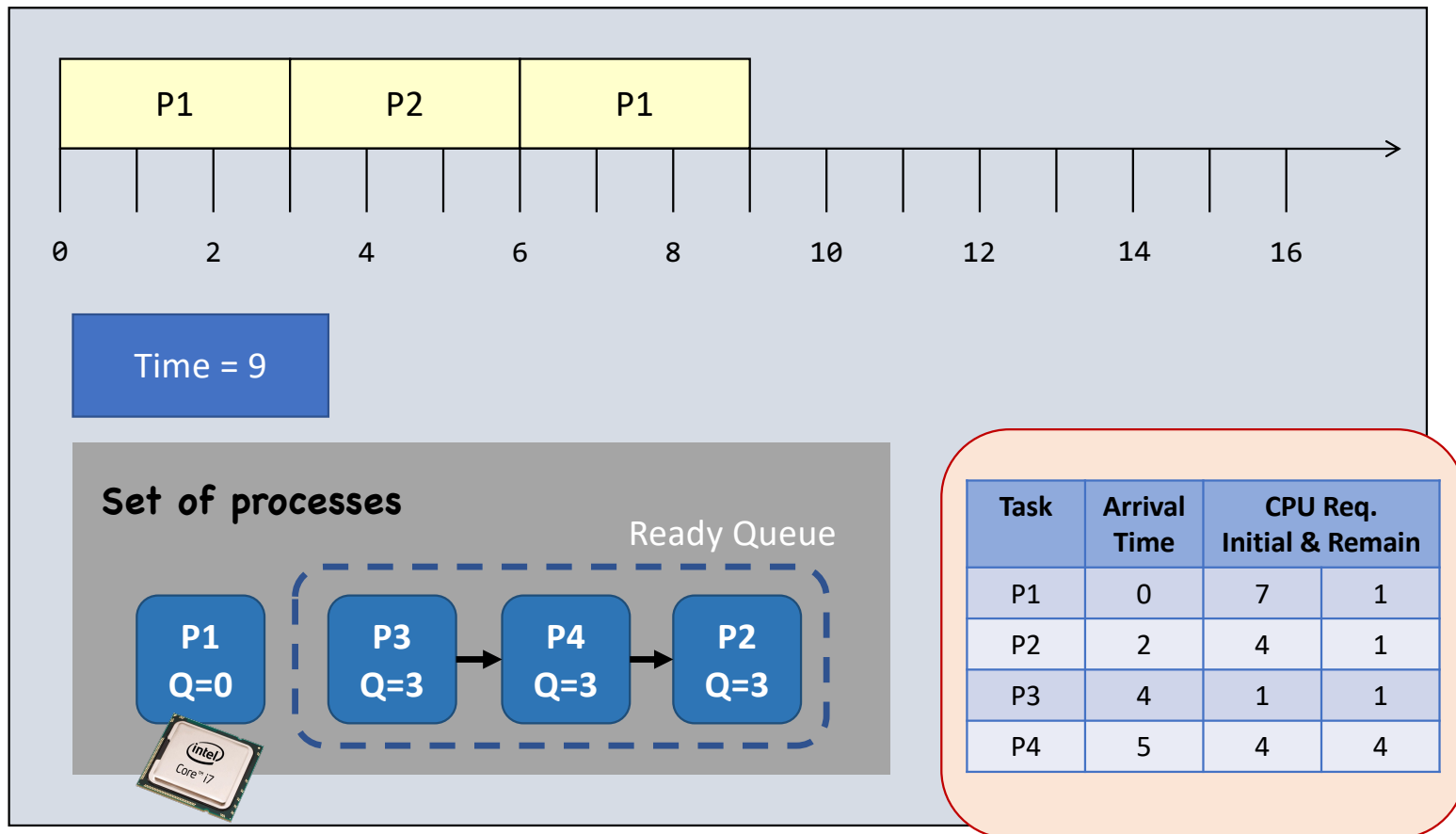
Animation; don't print

# Round Robin (Quantum = 3)



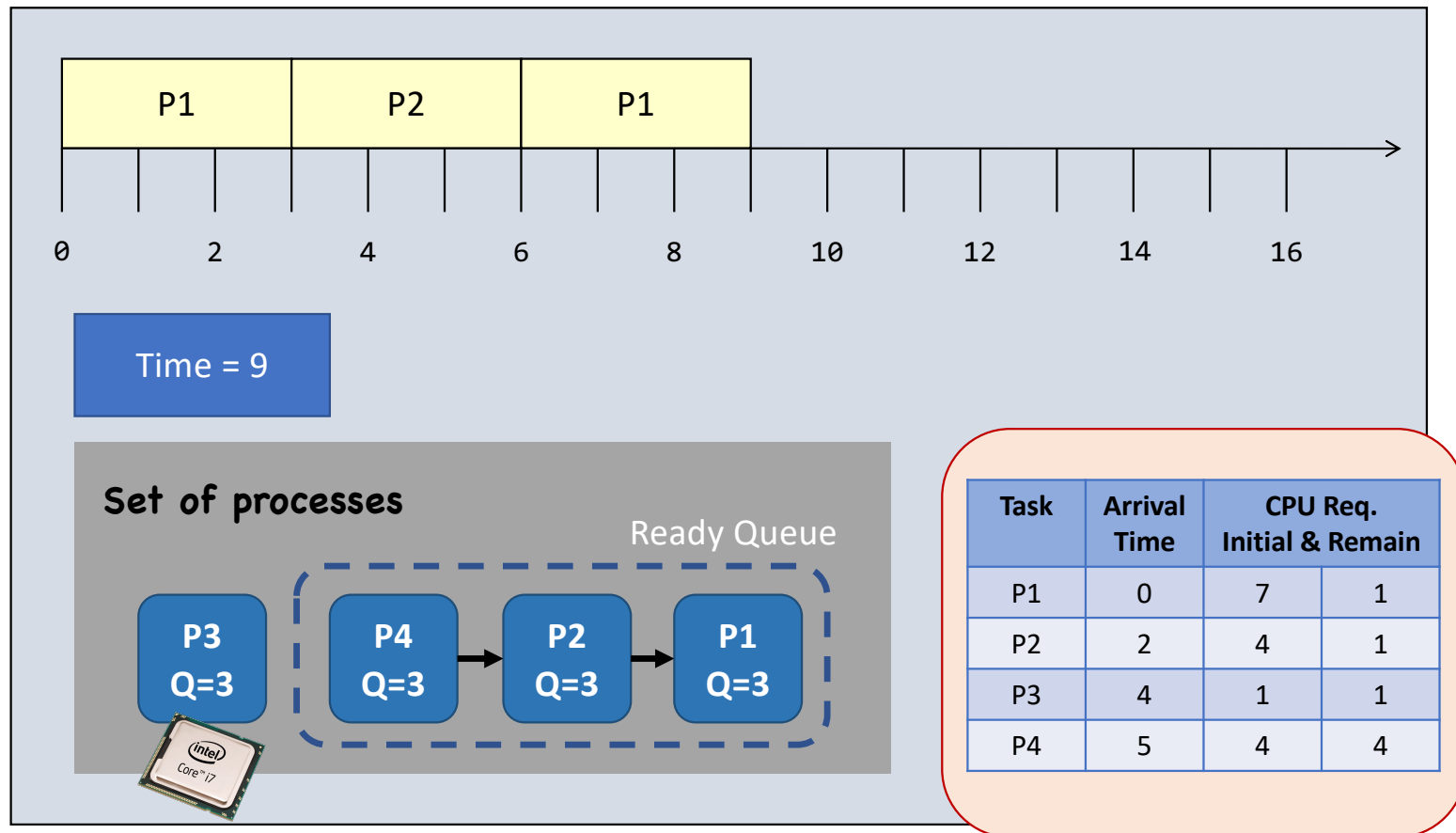
Animation; don't print

# Round Robin (Quantum = 3)



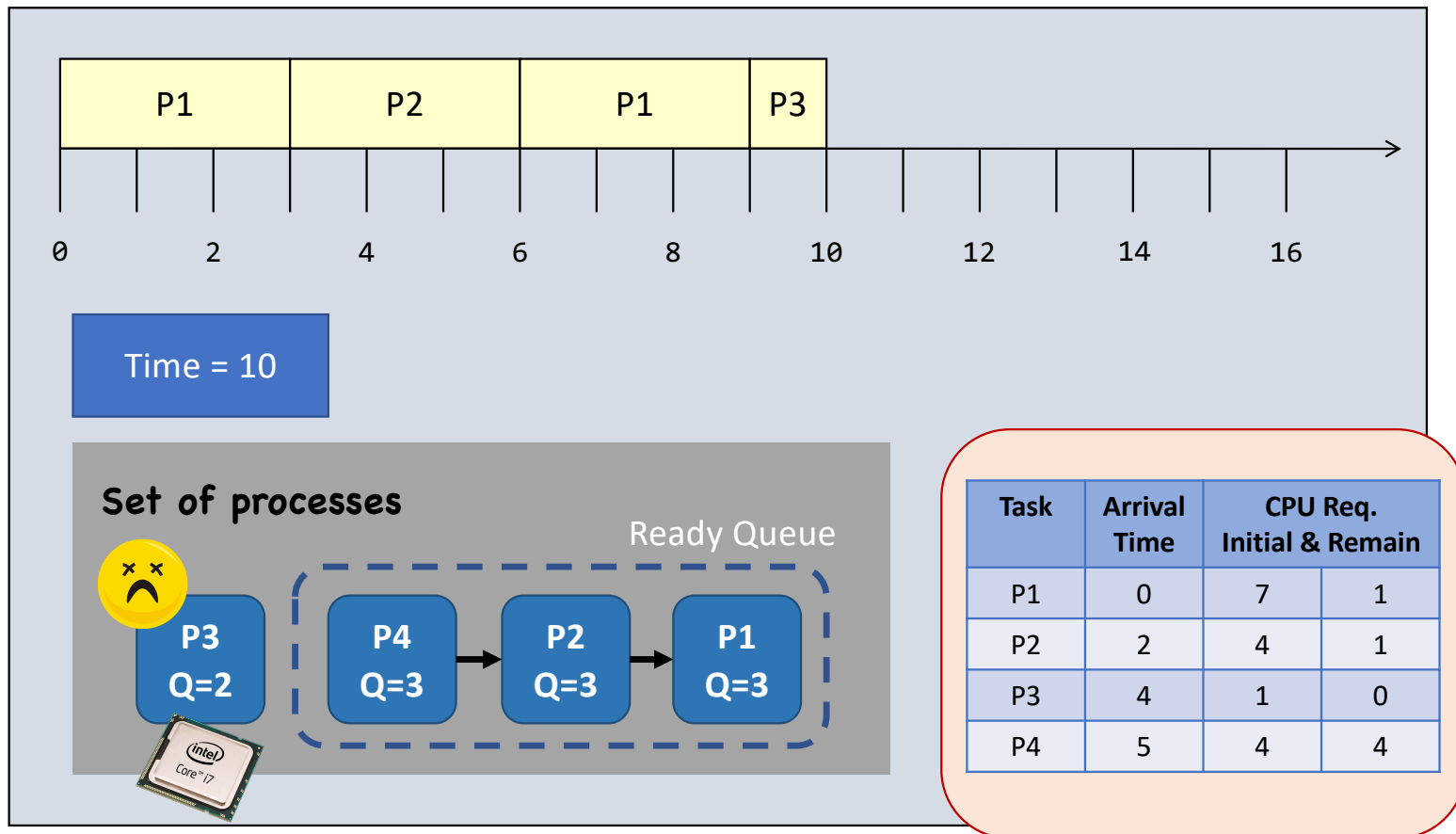
Animation; don't print

# Round Robin (Quantum = 3)



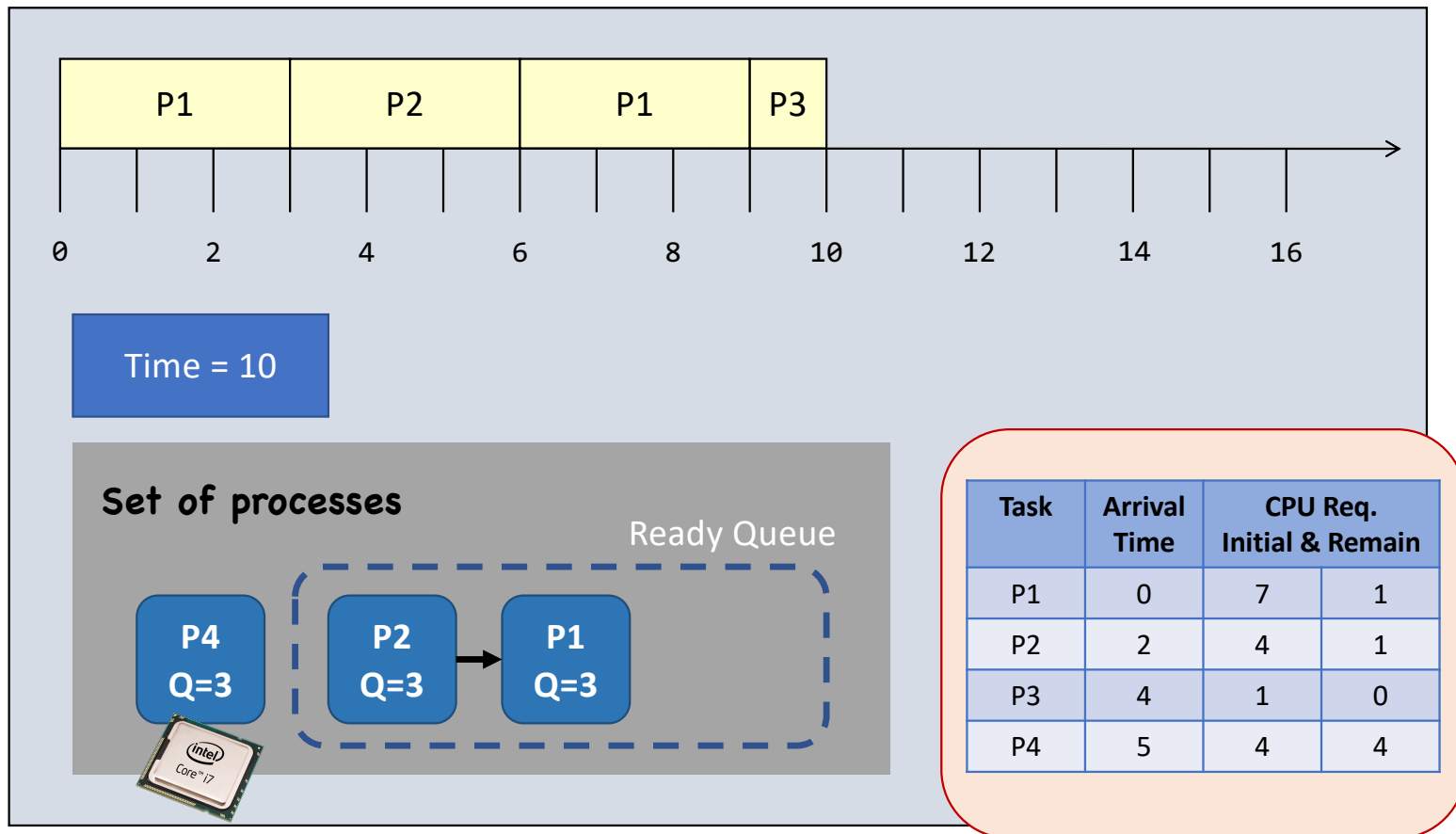
Animation; don't print

# Round Robin (Quantum = 3)



Animation; don't print

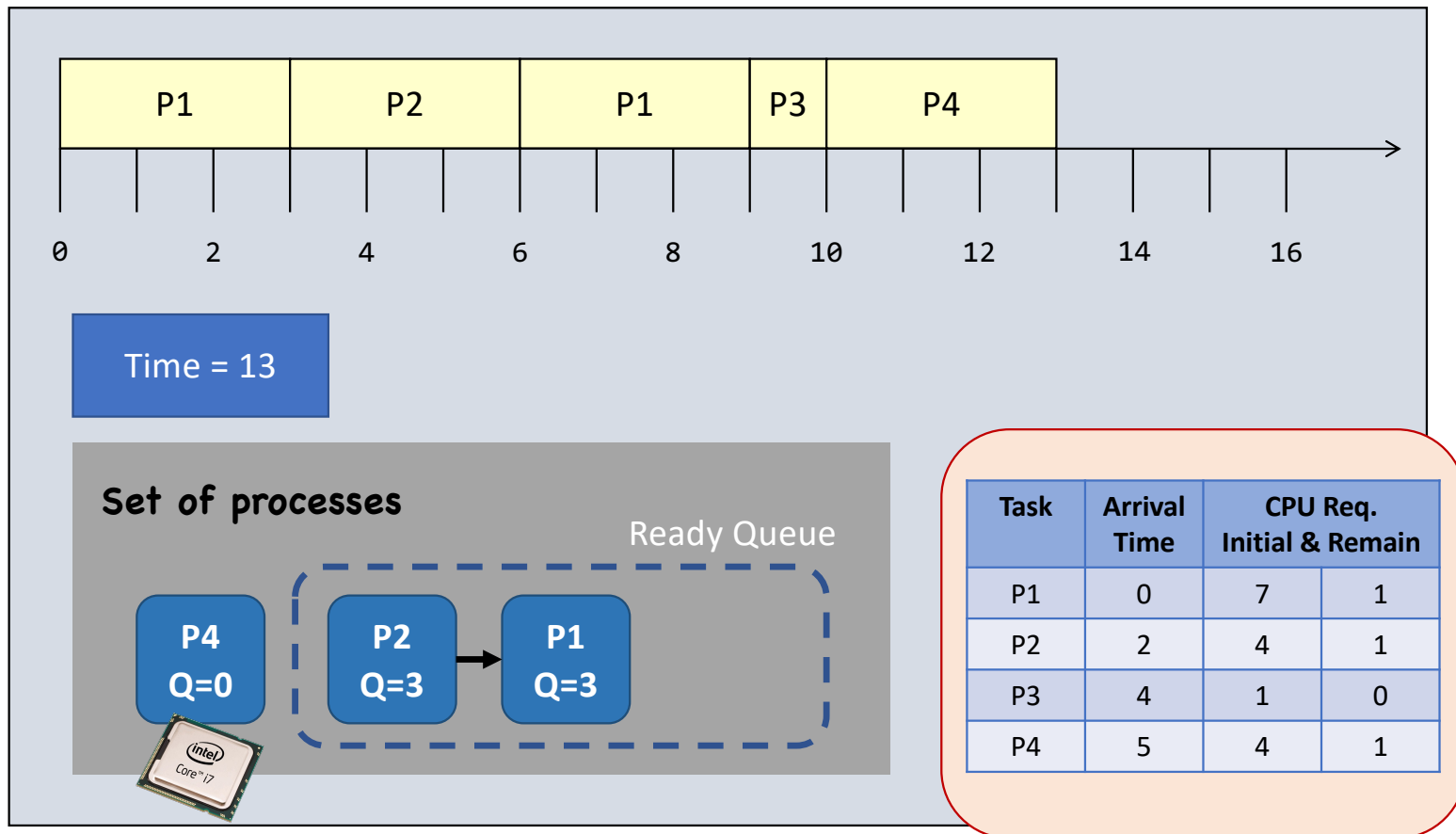
# Round Robin (Quantum = 3)





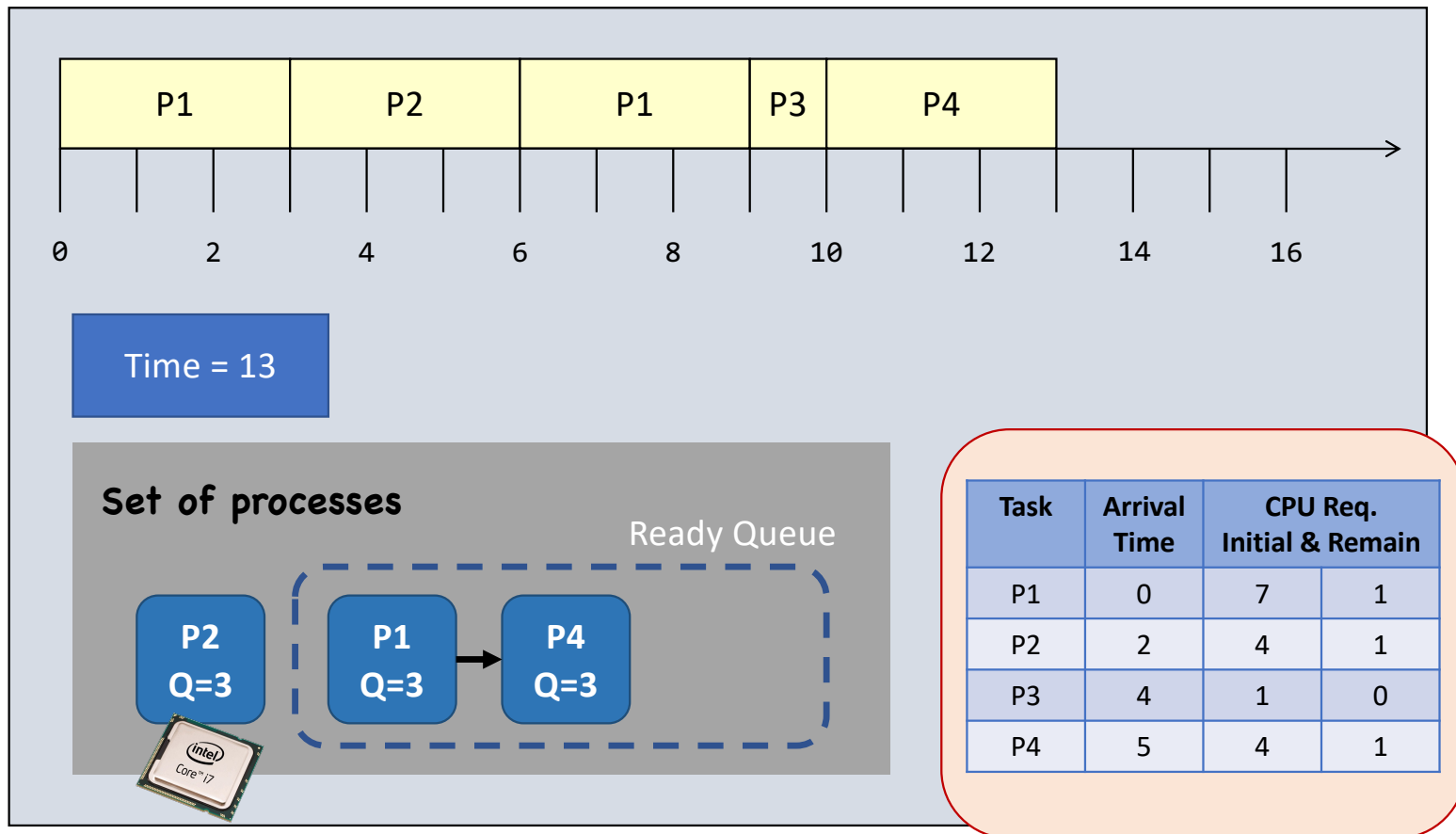
Animation; don't print

# Round Robin (Quantum = 3)



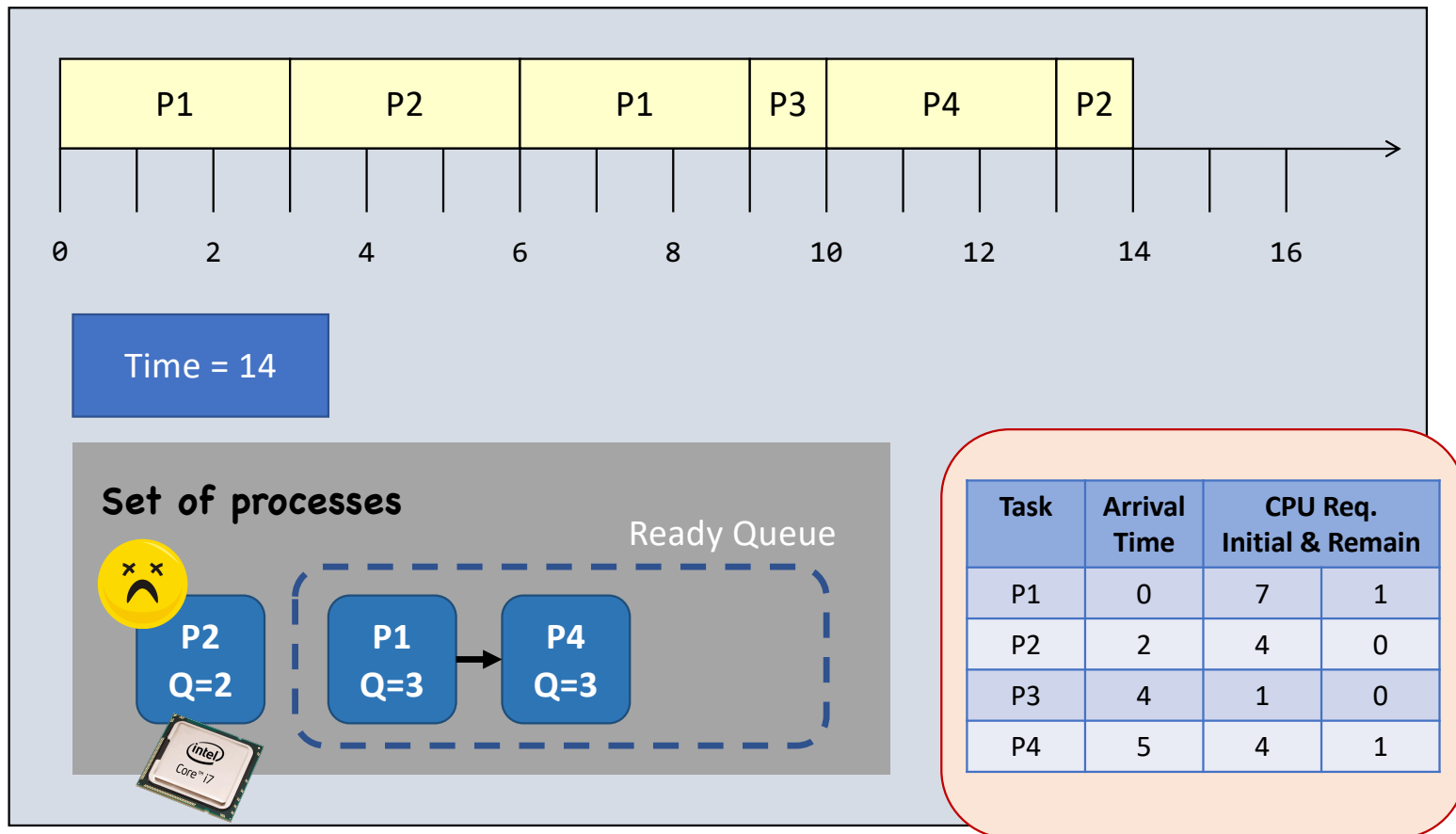
Animation; don't print

# Round Robin (Quantum = 3)



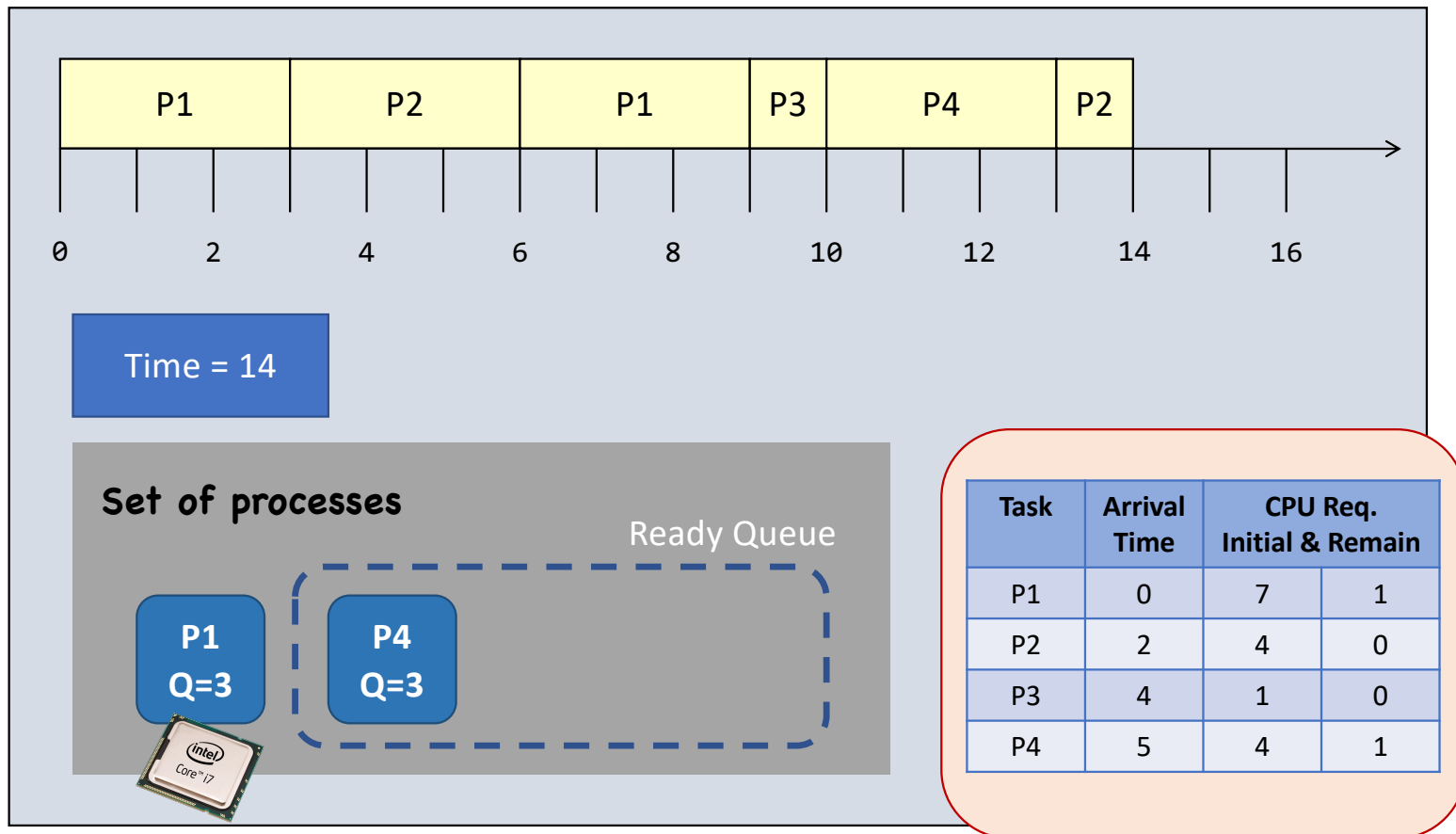
Animation; don't print

# Round Robin (Quantum = 3)



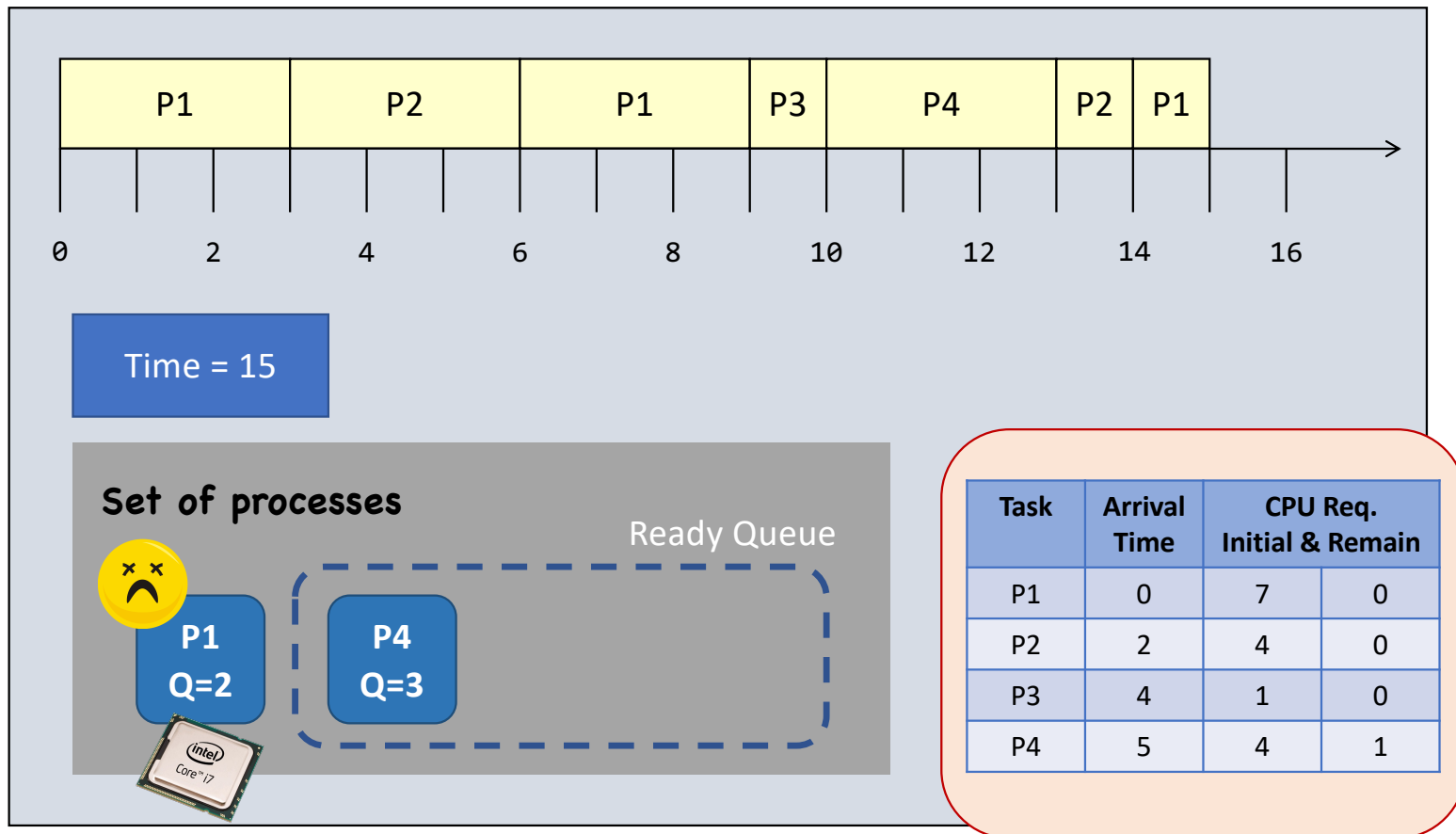
Animation; don't print

# Round Robin (Quantum = 3)



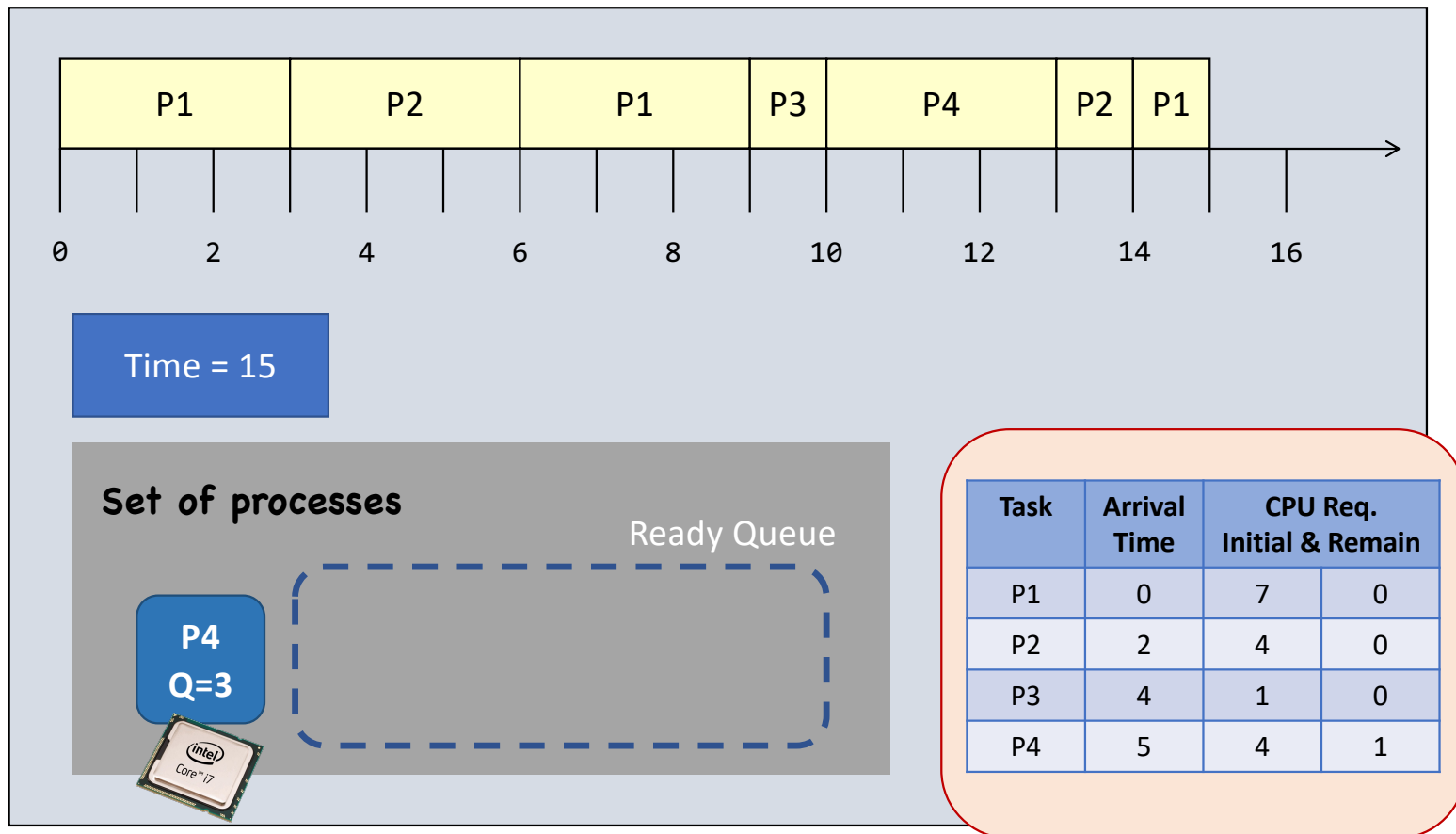
Animation; don't print

# Round Robin (Quantum = 3)



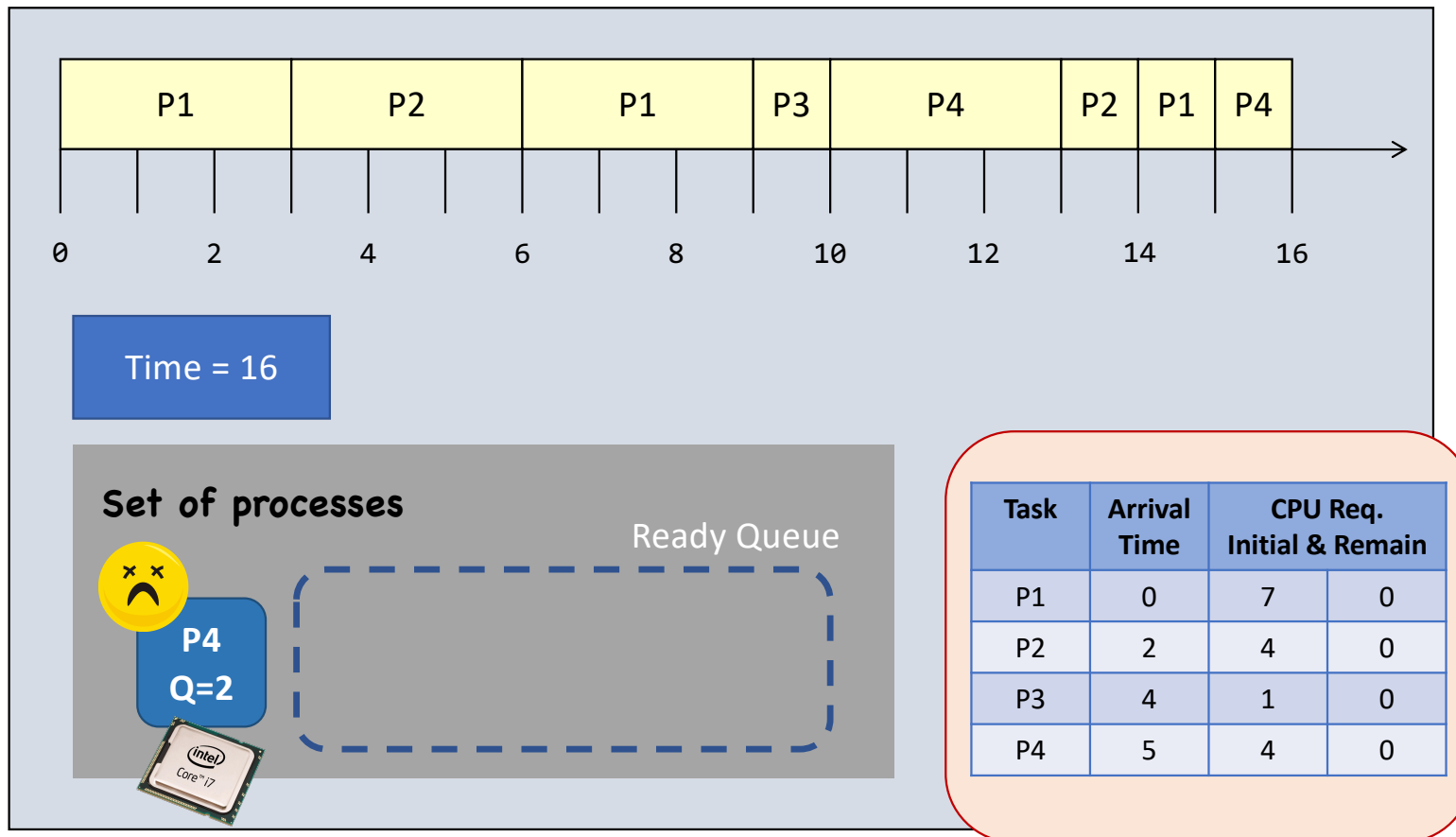
Animation; don't print

# Round Robin (Quantum = 3)

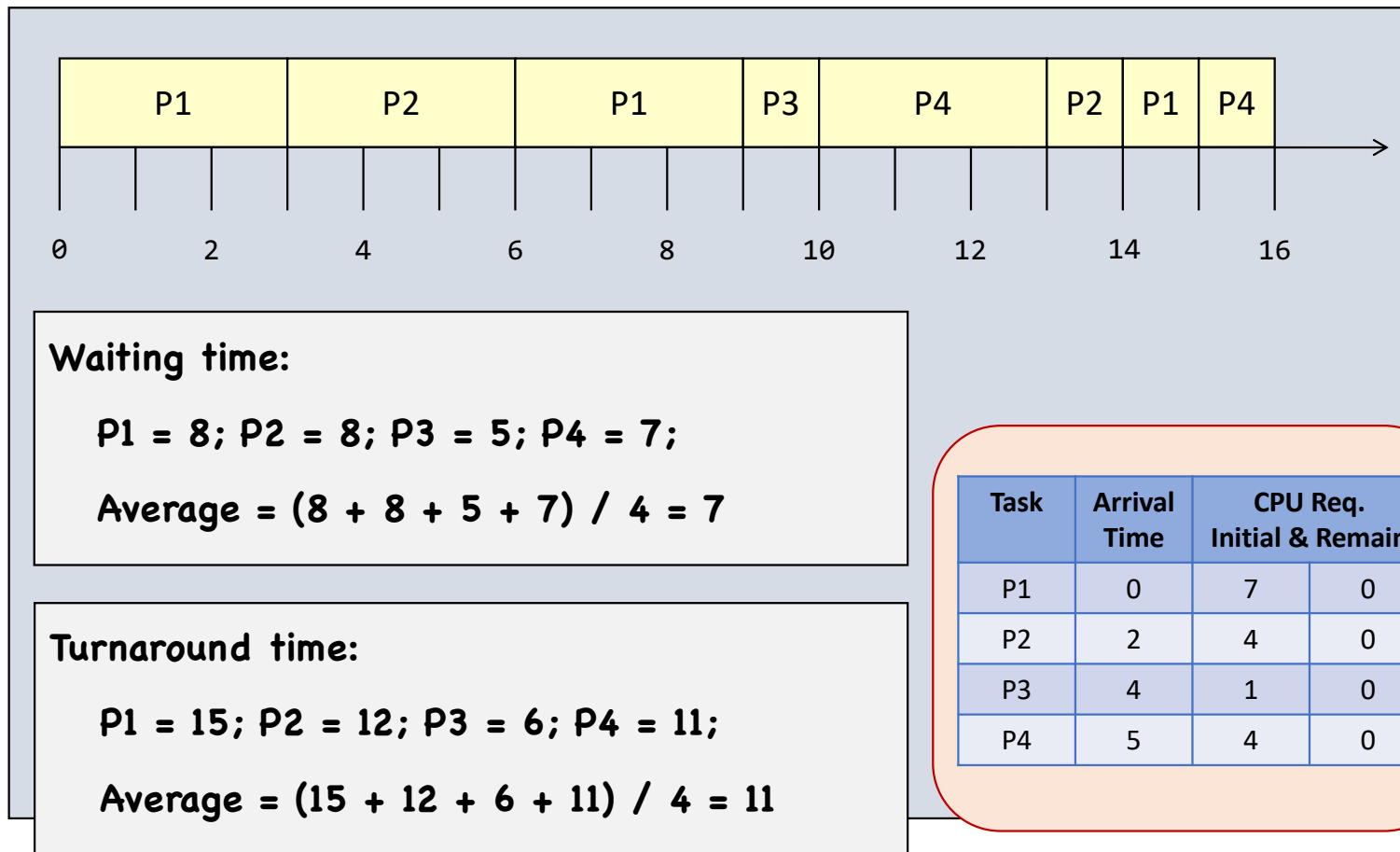


Animation; don't print

# Round Robin (Quantum = 3)



# Round Robin (Quantum = 3)





# RR v.s. SJF

	Non-preemptive SJF	Preemptive SJF	RR
Average waiting time	4	3	7 (largest)
Average turnaround time	8	7	11 (largest)
# of context switching	3	5	7 (largest)



So, the RR algorithm gets all the bad! Why do we still need it?

**The responsiveness of the processes** is great under the RR algorithm. E.g., you won't feel a job is "frozen" because every job gets the CPU from time to time!

# Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer  $\equiv$  highest priority)
  - Nonpreemptive: newly arrived process simply put into the queue
  - Preemptive: if the priority of the newly arrived process is higher than priority of the currently running process---preempt the CPU
- Static priority and dynamic priority
  - static priority: fixed priority throughout its lifetime
  - dynamic priority: priority changes over time
- SJF is a priority scheduling where priority is the next CPU burst time

# Priority Scheduling (Cont'd)

- Problem  $\equiv$  **Starvation** – low priority processes may never execute
  - Rumors has it that when they shut down the IBM 7094 at MIT in 1973, they found a low priority process that had been submitted in 1967 and had not yet been run.
- Solution  $\equiv$  **Aging** – as time progresses increase the priority of the process
  - Example: priority range from 127 (low) to 0 (high)
  - Increase priority of a waiting process by 1 every 15 minutes
  - 32 hours to reach priority 0 from 127

# Thank you!

