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., **user-time > sys-time**	Spends most of its running time on I/O, i.e., **sys-time > user-time**
Examples - AI course assignments.	**Examples** - **/bin/ls**, networking programs.

CPU执行阶段（CPU burst）

程序在CPU上执行计算指令，如加法、赋值、循环等。

• 特点：CPU繁忙、计算密集。

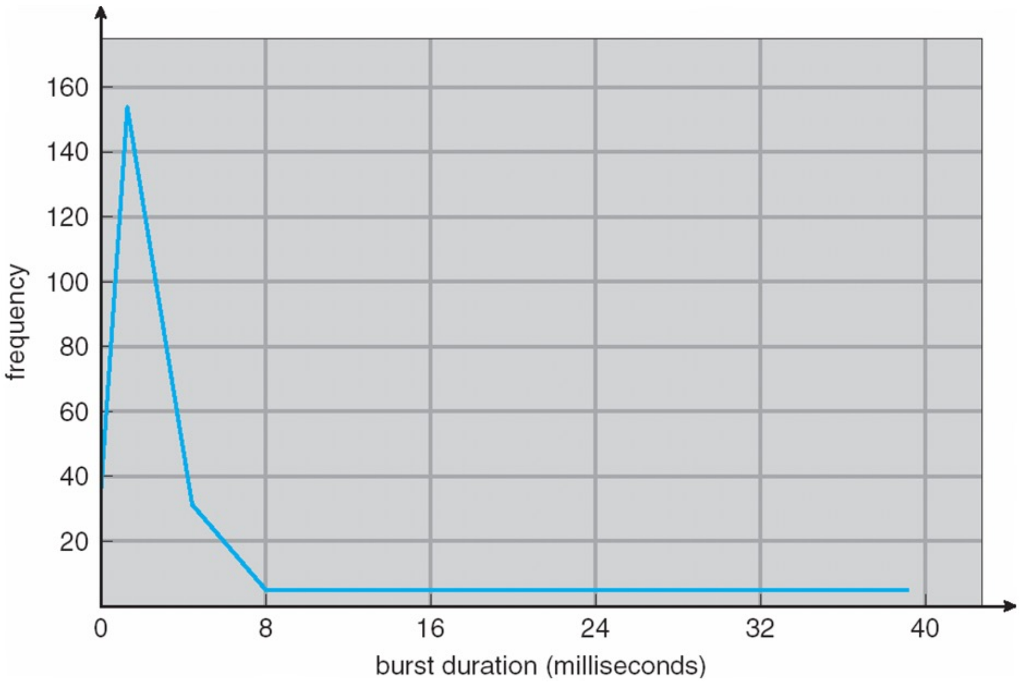
• 示例：执行 x = x + y、数组排序、矩阵运算。

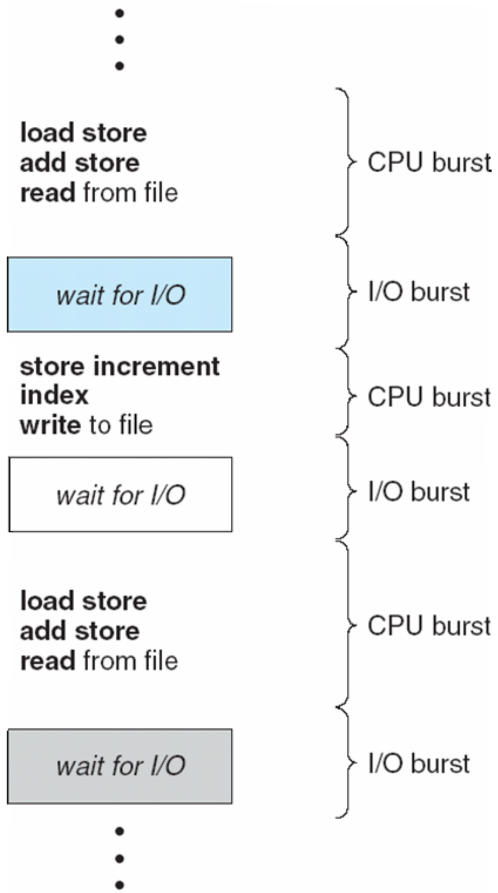
I/O等待阶段（I/O burst）

程序等待外部设备完成输入或输出操作，如读写文件、网络请求等。

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**

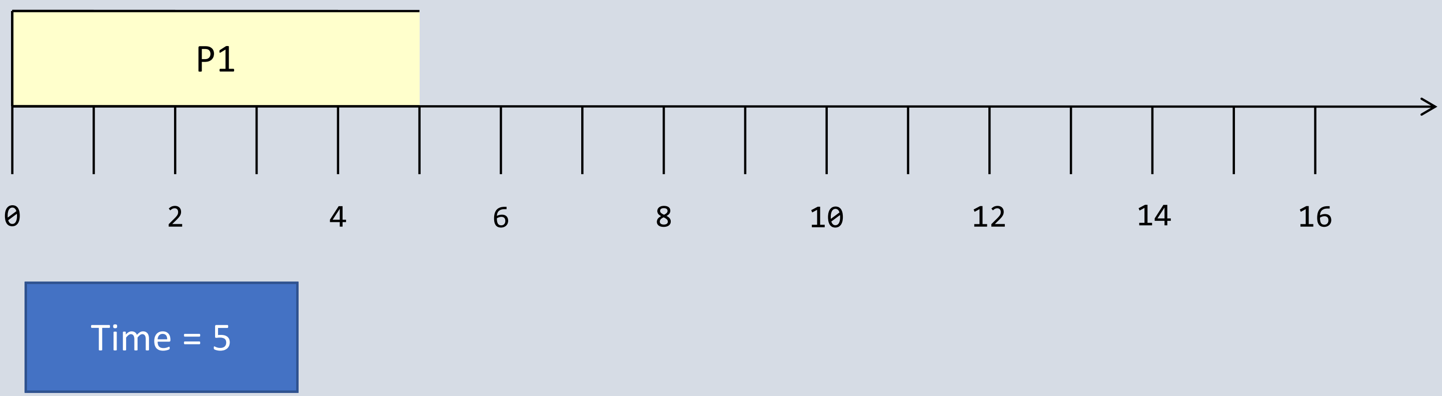
Scheduling Algorithm Optimization Criteria

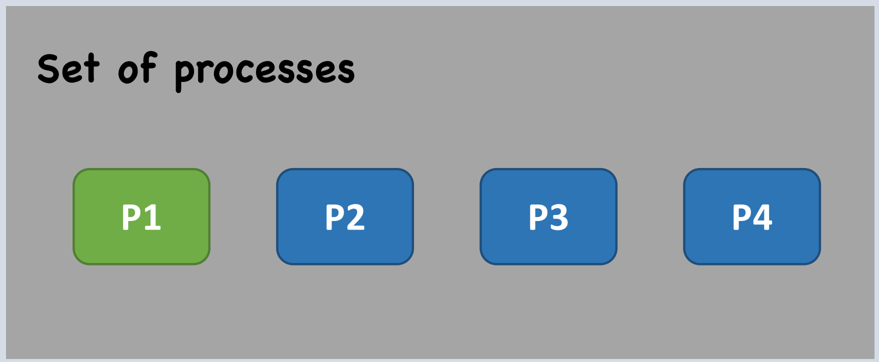
- Given a set of processes, with
  - **Arrival time**: the time they arrive in the CPU ready queue (from waiting state or from new state)
  - **CPU requirement**: their expected CPU burst time
- Minimize average turnaround time
  - **Turnaround time**: The time between the arrival of the task and the 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

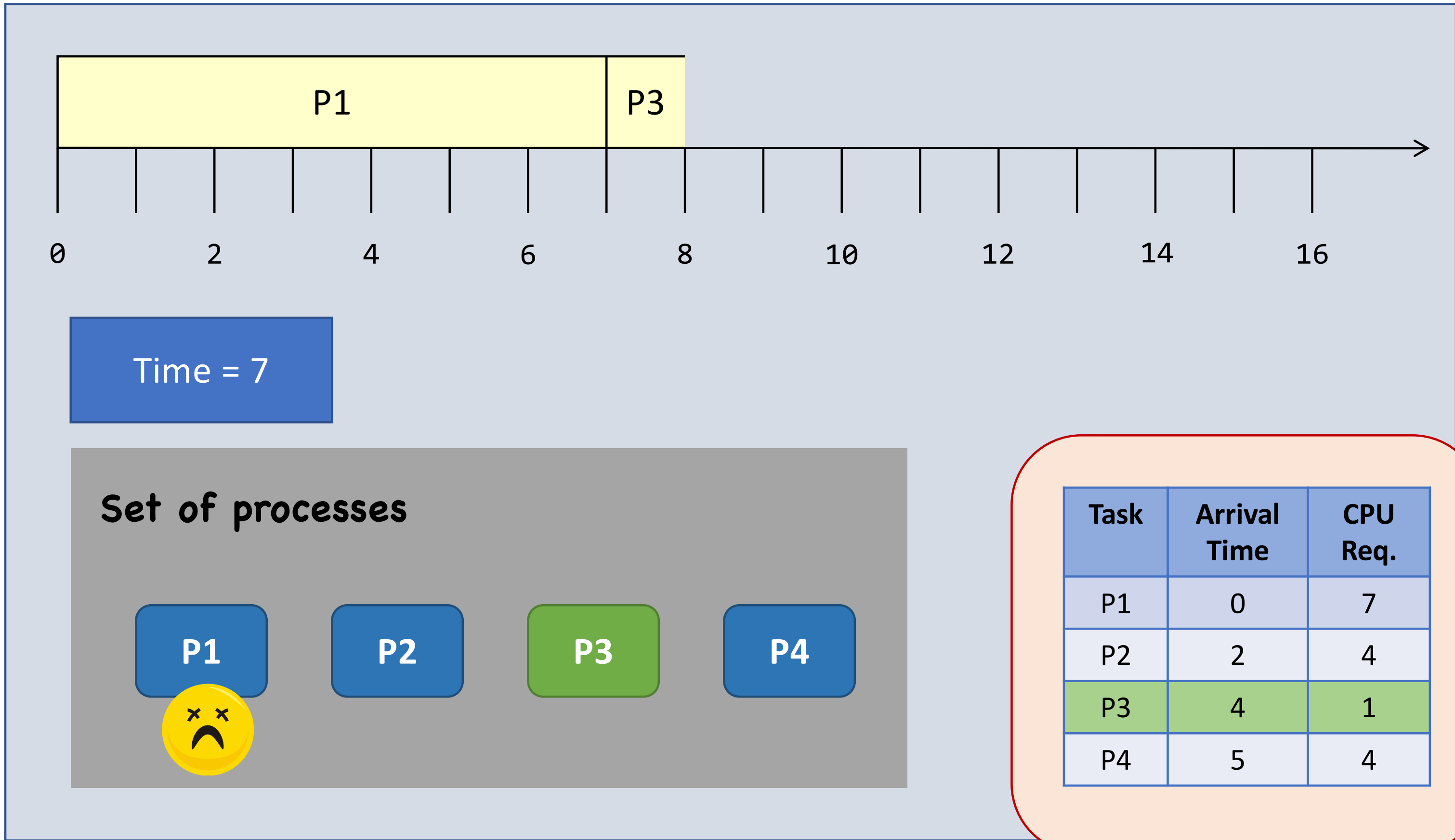
Non-preemptive SJF





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

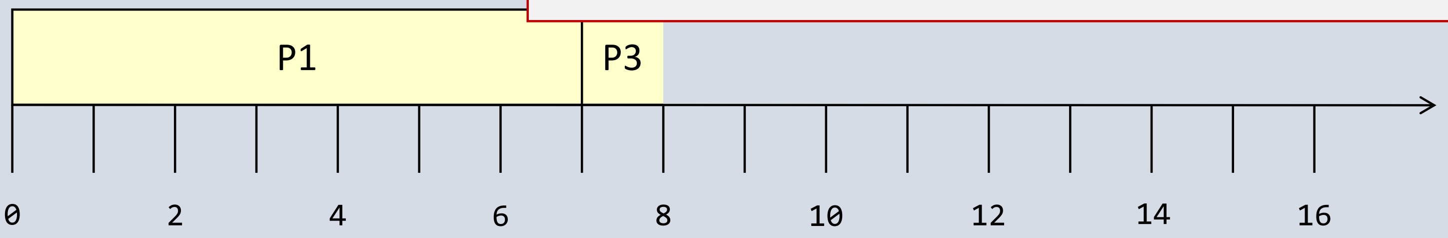
Non-preemptive SJF



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

Non-preemptive SJF

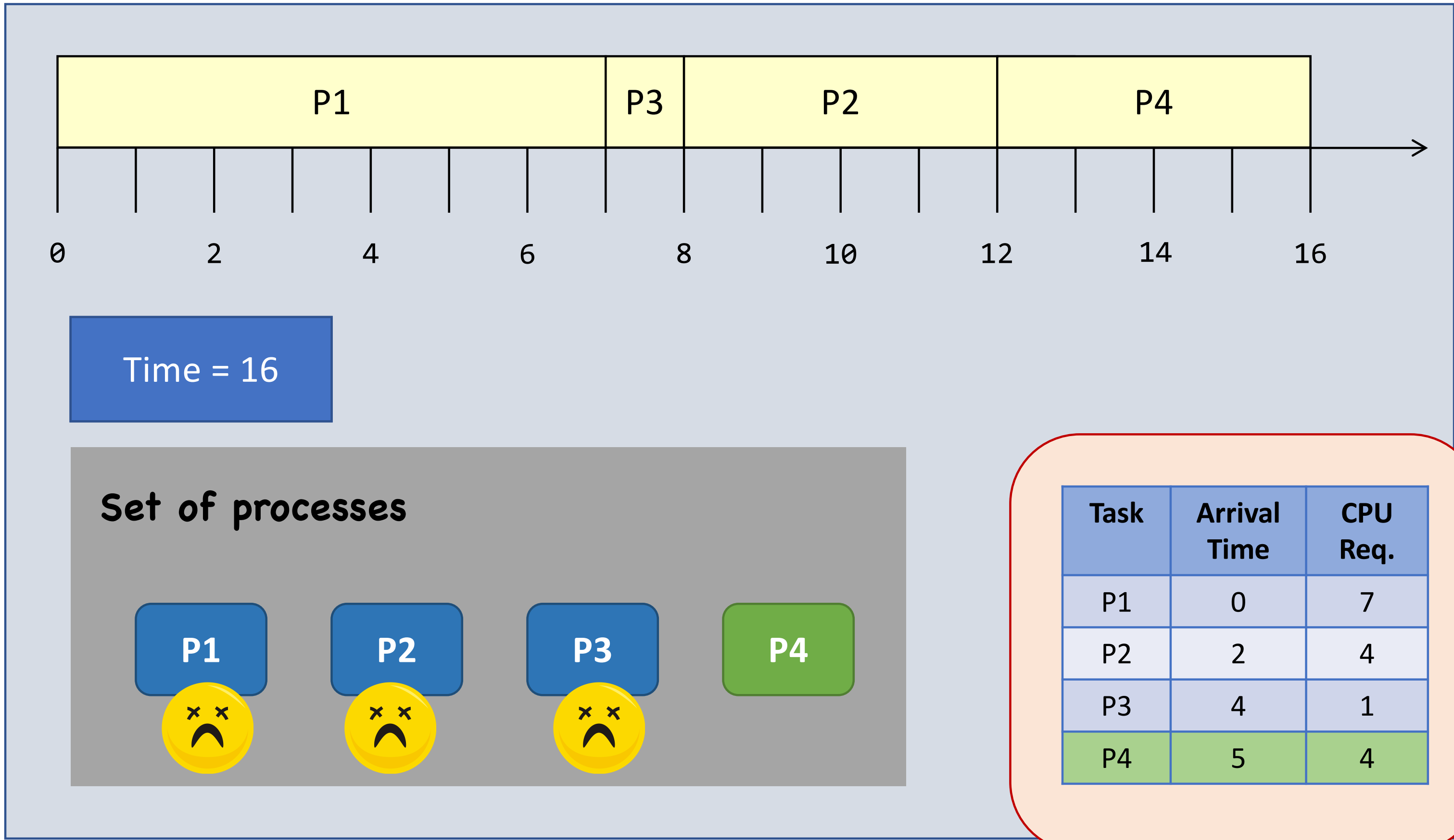
In this example, we use **FIFO** to break the tie.



Time = 7

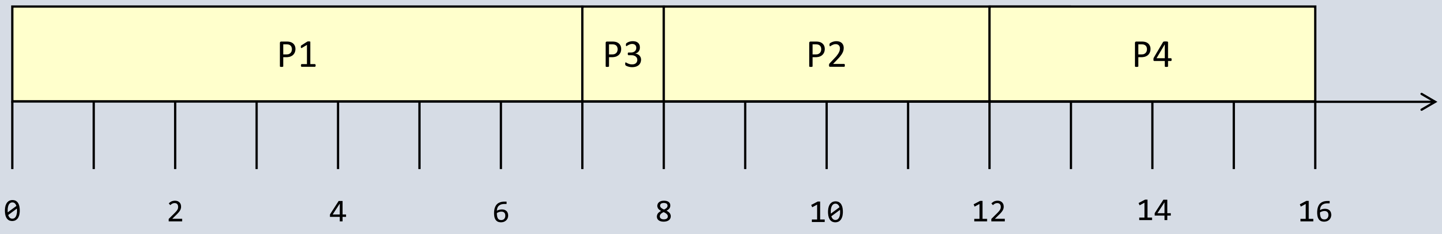
Task	Arrival Time	CPU Req.
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Non-preemptive SJF



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

Non-preemptive SJF



Waiting time: 开始减到达

P1 = 0; P2 = 6; P3 = 3; P4 = 7;

Average = (0 + 6 + 3 + 7) / 4 = 4.

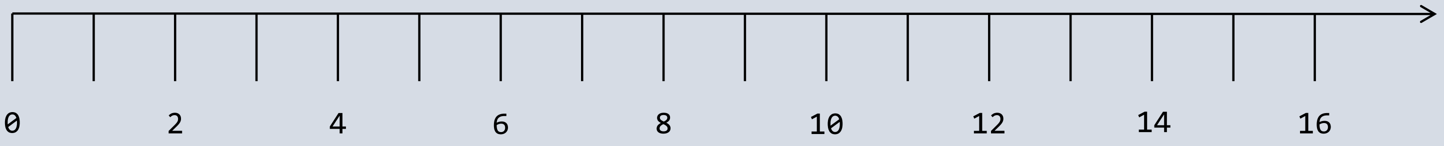
Turnaround time: 到达减完成

P1 = 7; P2 = 10; P3 = 4; P4 = 11;

Average = (7 + 10 + 4 + 11) / 4 = 8.

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

Preemptive SJF

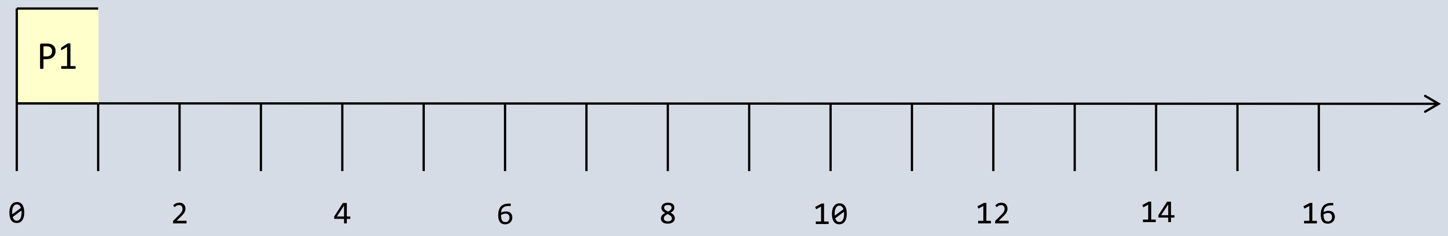


Whenever a new process arrives in the ready queue (either from waiting or from new state), the scheduler steps in and selects the next task based on **their remaining CPU requirements**.

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



Time = 0

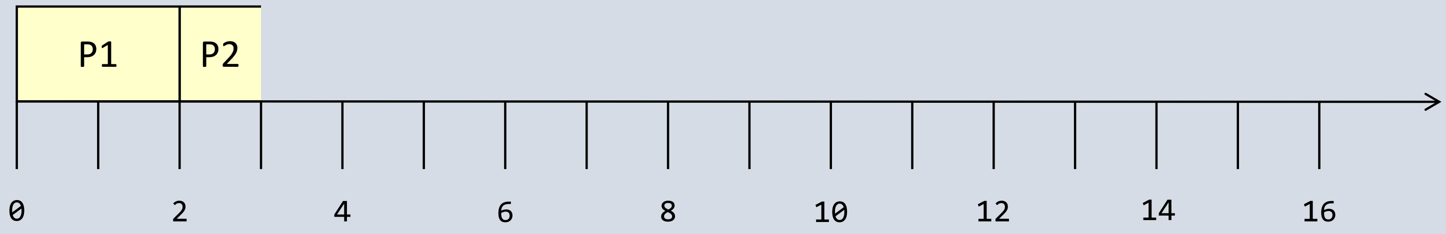
Set of processes

P1

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



Time = 2

P2 is selected!

Set of processes

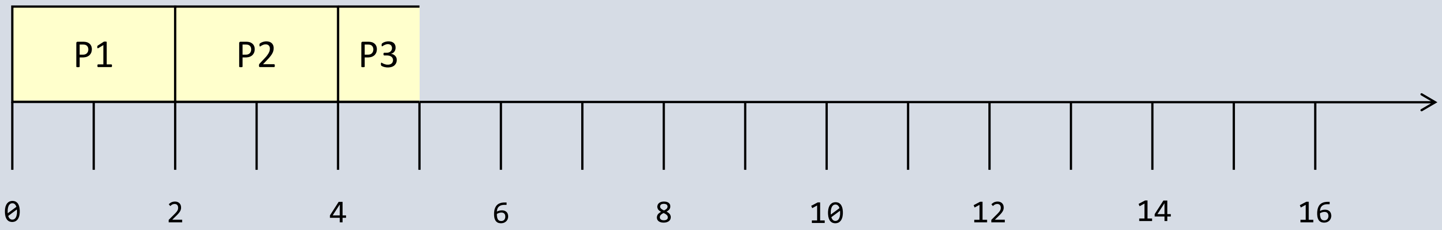
P1

P2

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

Animation; don't print

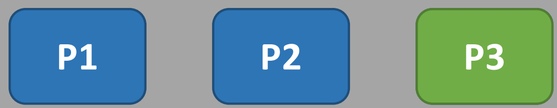
Preemptive SJF



Time = 4

P3 is selected!

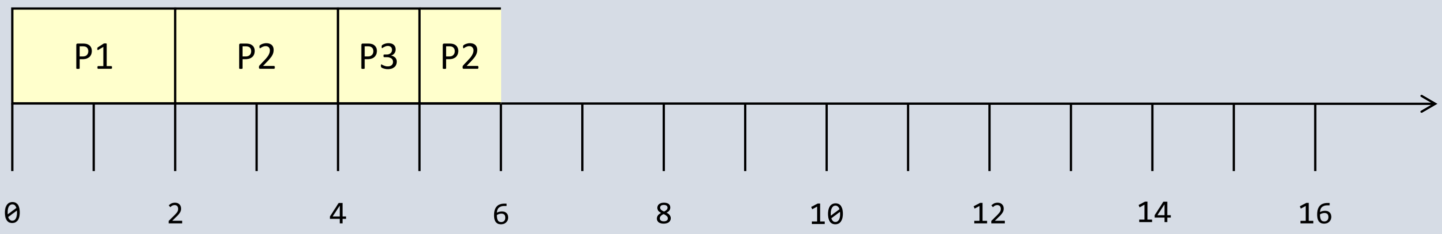
Set of processes



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

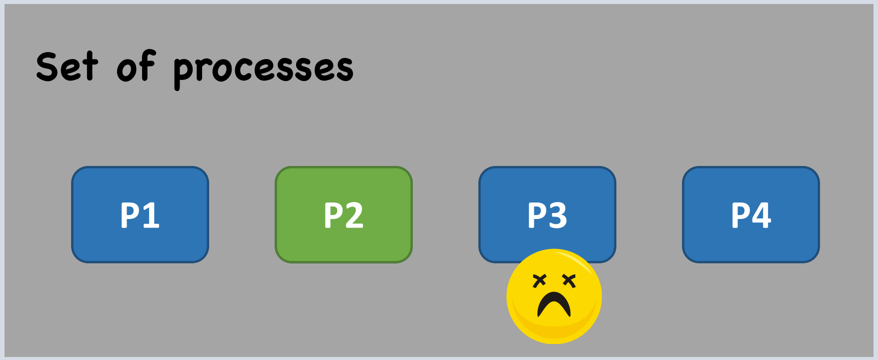
Animation; don't print

Preemptive SJF



Time = 5

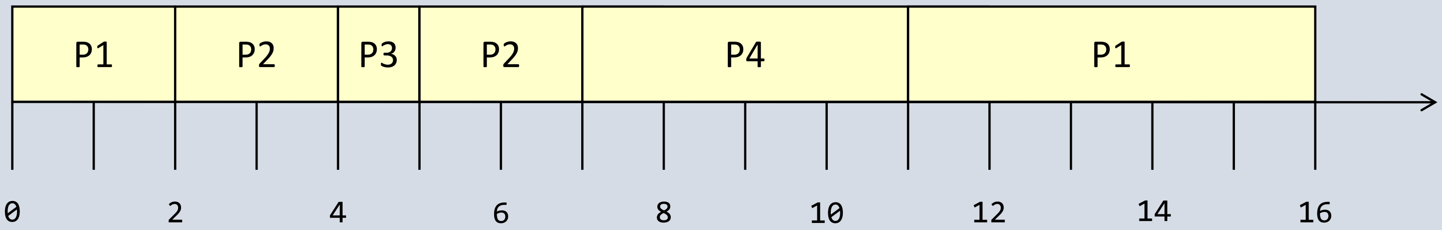
P2 is selected!



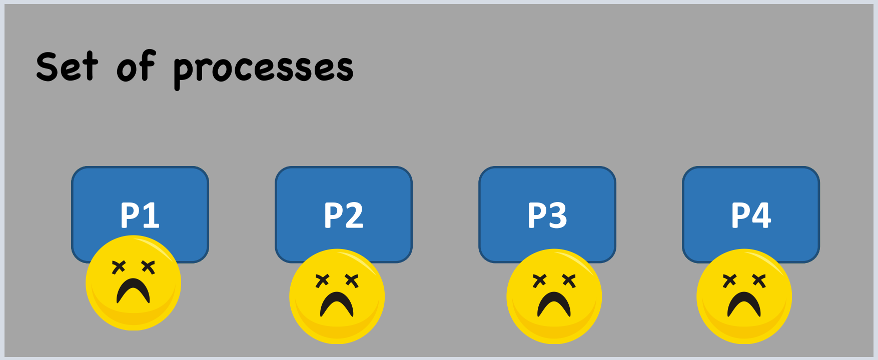
Task	Arrival Time	CPU Req. Initial & Remain	
P1	0	7	5
P2	2	4	2
P3	4	1	0
P4	5	4	4

Animation; don't print

Preemptive SJF

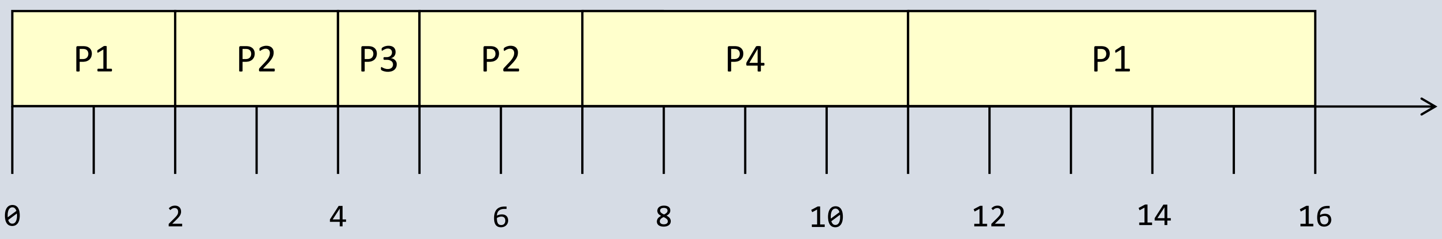


Time = 16



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

Preemptive SJF



Waiting time:

对应的turnaround time–waiting time=req 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) 时间片轮转调度

每个进程轮流获得一小段固定的 CPU 时间（称为 time quantum / 时间片），时间片用完就切换到下一个进程。

- 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

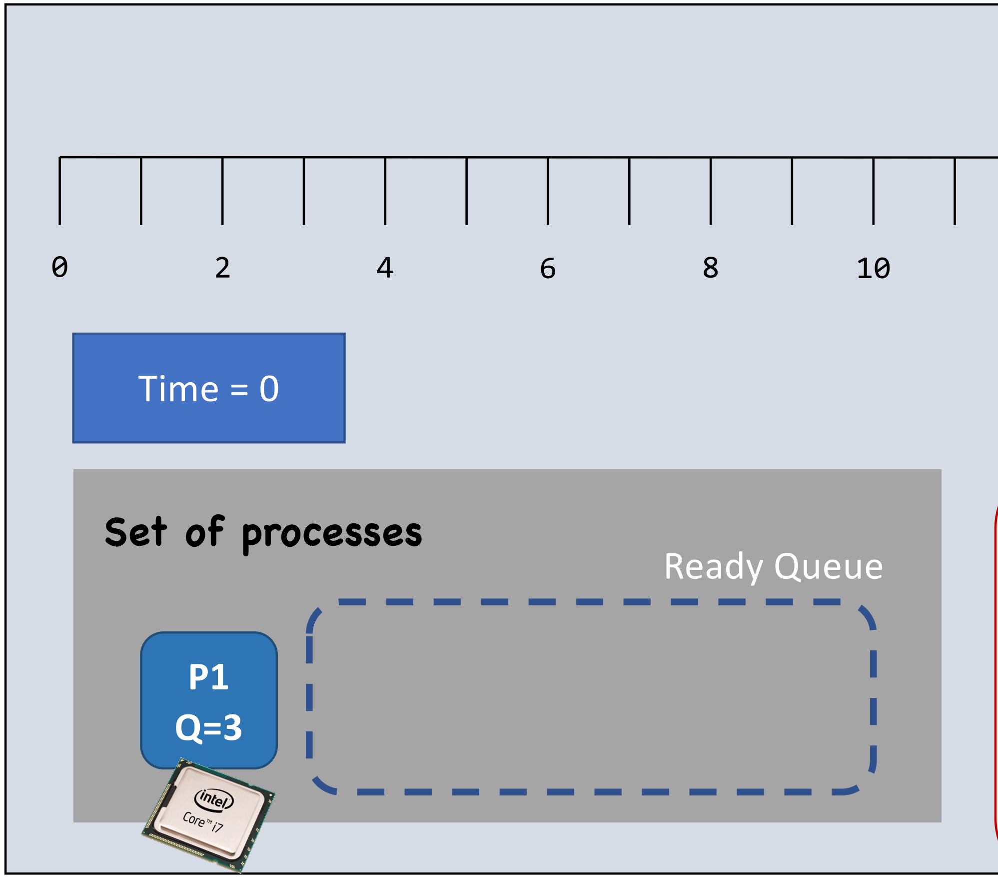
新进程不会立刻打断当前正在运行的进程；

它会等前面的进程轮完一圈后再执行；

因此系统的公平性得以保持

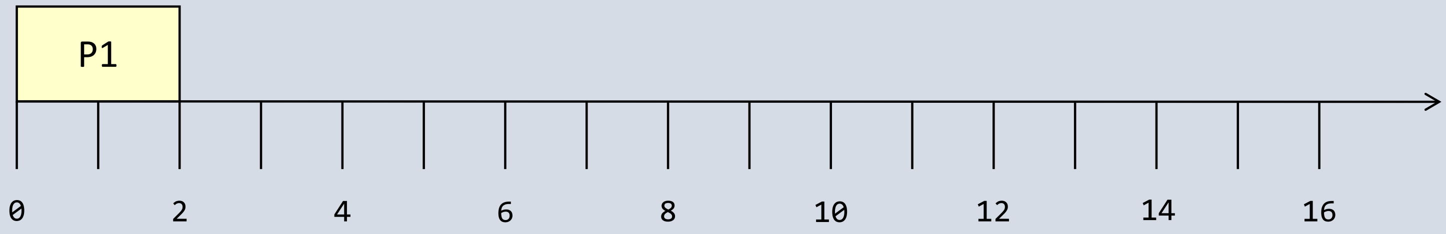
Animation; don't print

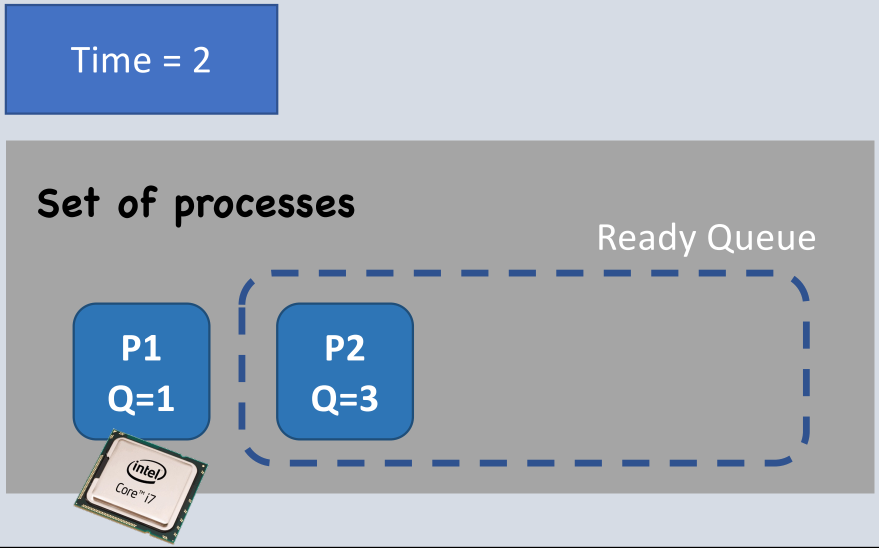
Round Robin (Quantum = 3)



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

Round Robin (Quantum = 3)

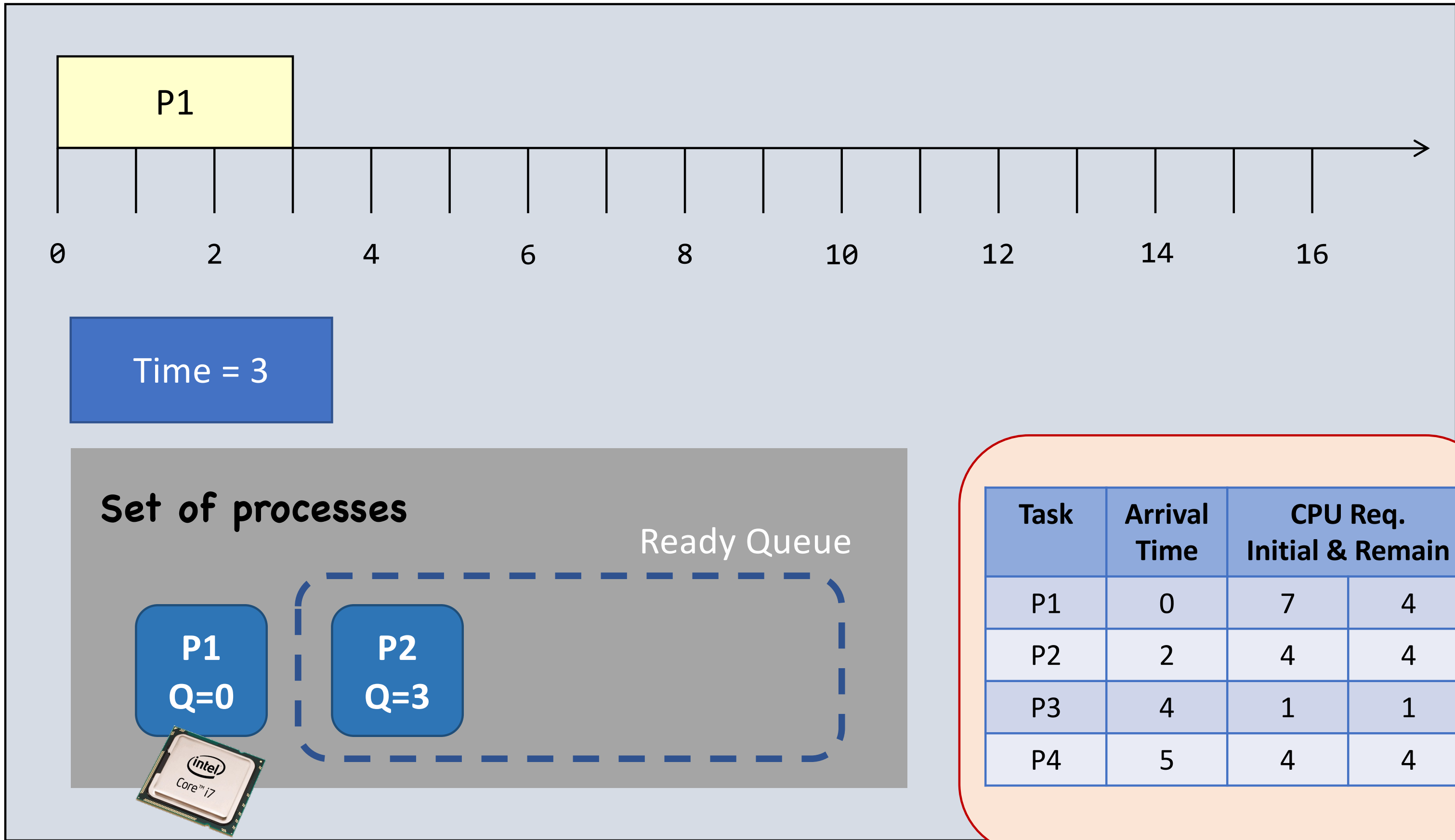




Task	Arrival Time	CPU Req. Initial & Remain	
P1	0	7	5
P2	2	4	4
P3	4	1	1
P4	5	4	4

Animation; don't print

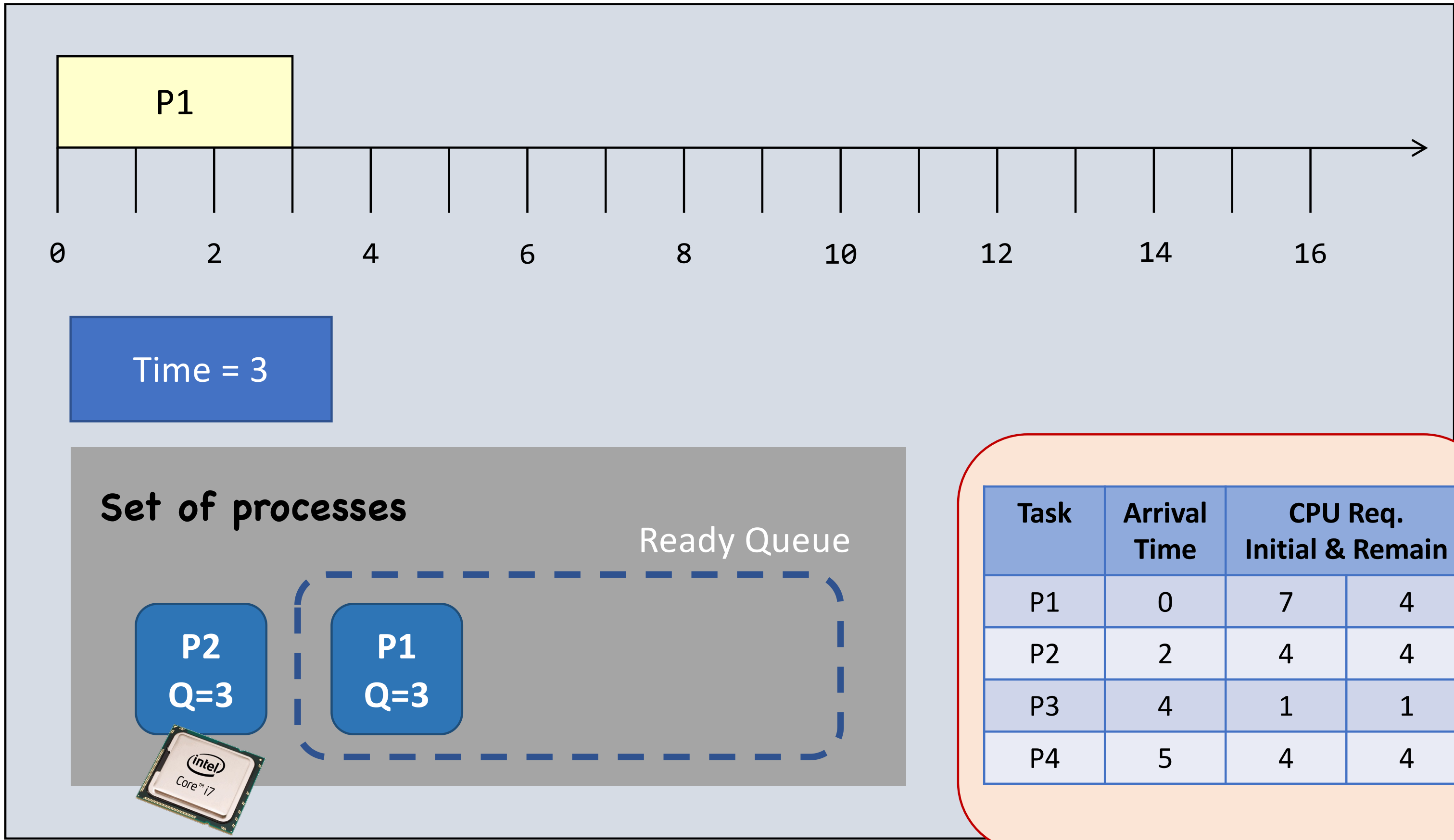
Round Robin (Quantum = 3)



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

Animation; don't print

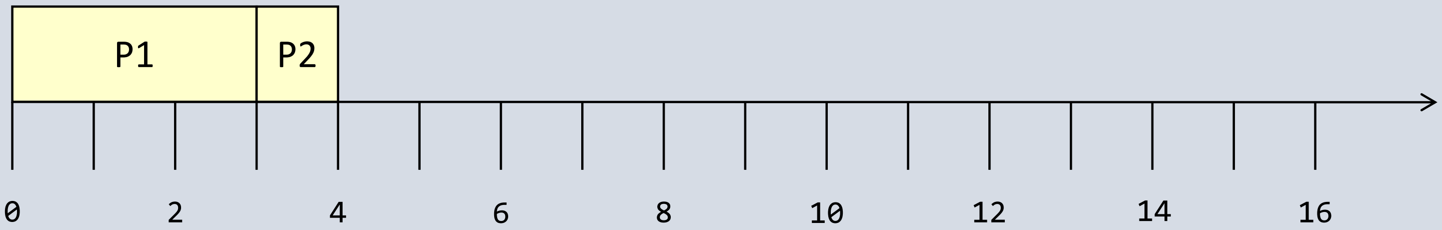
Round Robin (Quantum = 3)



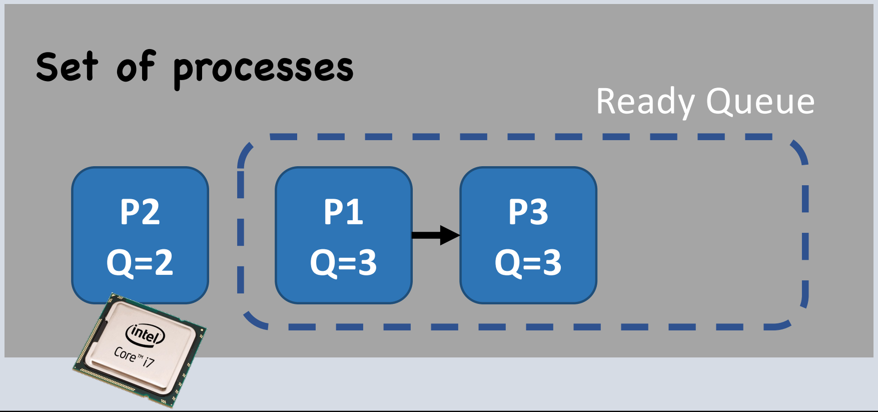
Task	Arrival Time	CPU Req. Initial & Remain	
P1	0	7	4
P2	2	4	4
P3	4	1	1
P4	5	4	4

Animation; don't print

Round Robin (Quantum = 3)



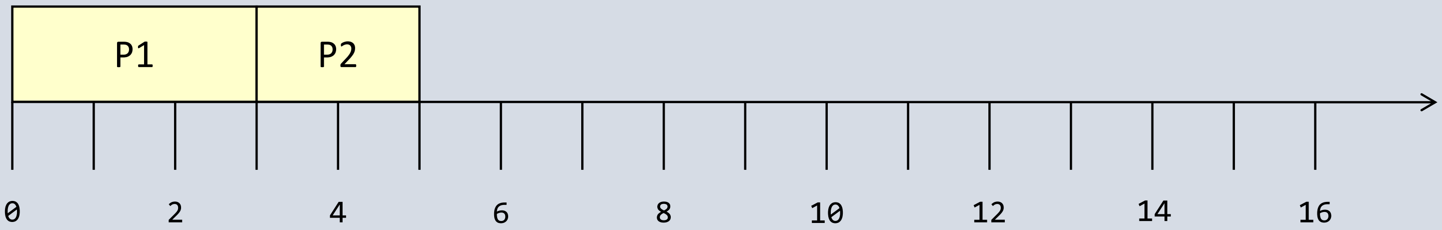
Time = 4



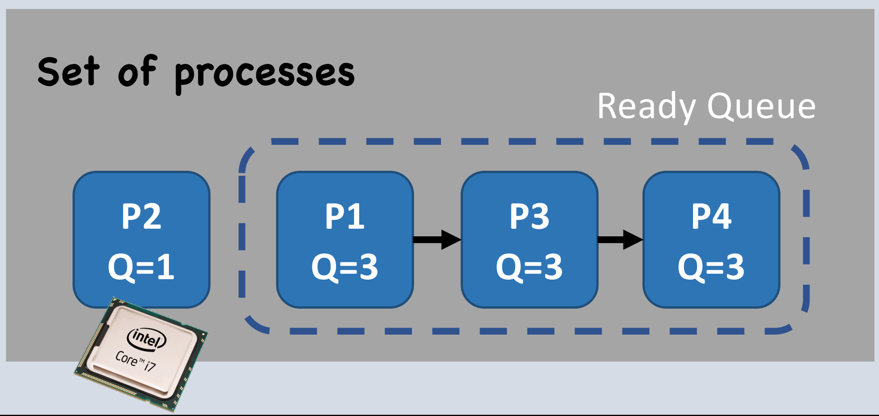
Task	Arrival Time	CPU Req. Initial & Remain	
P1	0	7	4
P2	2	4	3
P3	4	1	1
P4	5	4	4

Animation; don’t print

Round Robin (Quantum = 3)



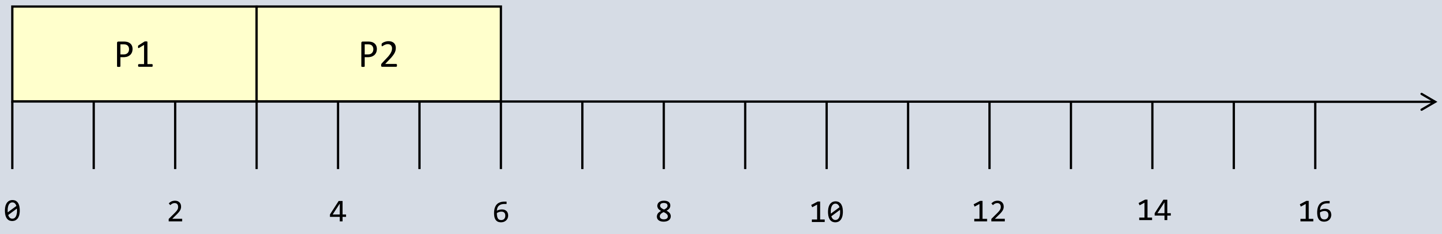
Time = 5



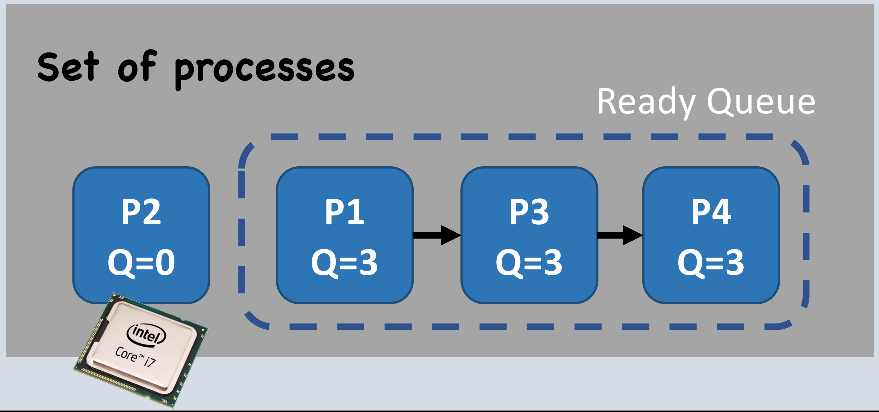
Task	Arrival Time	CPU Req. Initial & Remain	
P1	0	7	4
P2	2	4	2
P3	4	1	1
P4	5	4	4

Animation; don't print

Round Robin (Quantum = 3)



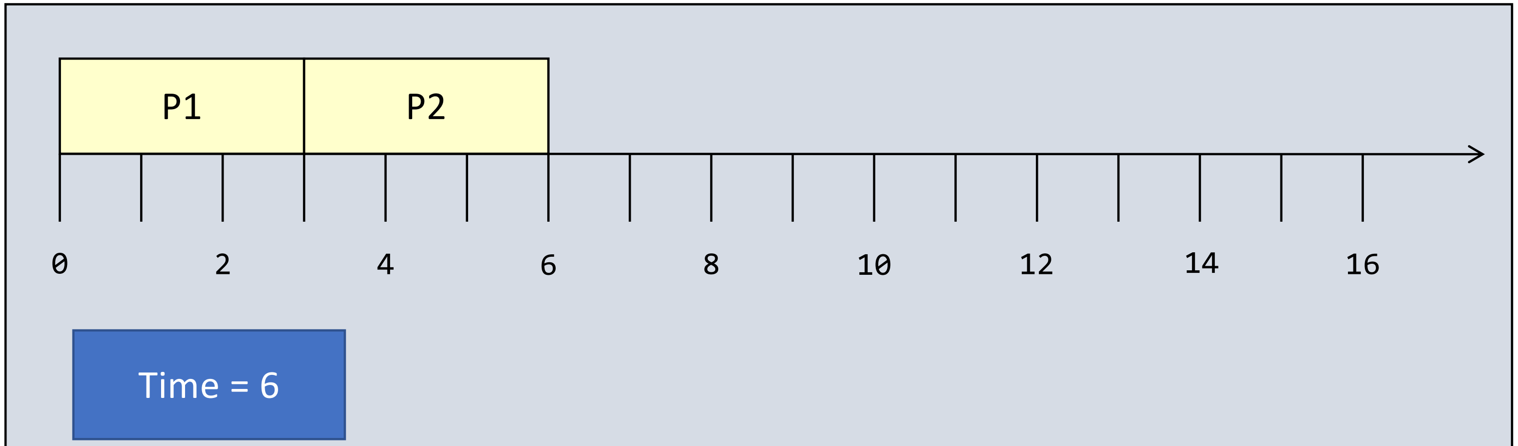
Time = 6

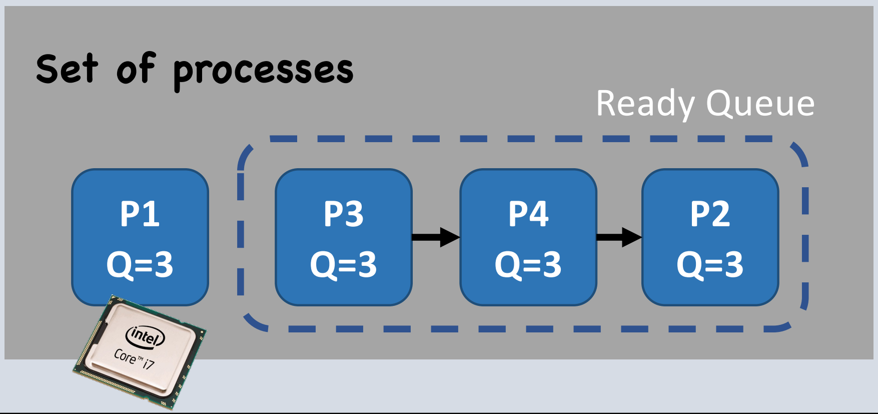


Task	Arrival Time	CPU Req. Initial & Remain	
P1	0	7	4
P2	2	4	1
P3	4	1	1
P4	5	4	4

Animation; don't print

Round Robin (Quantum = 3)

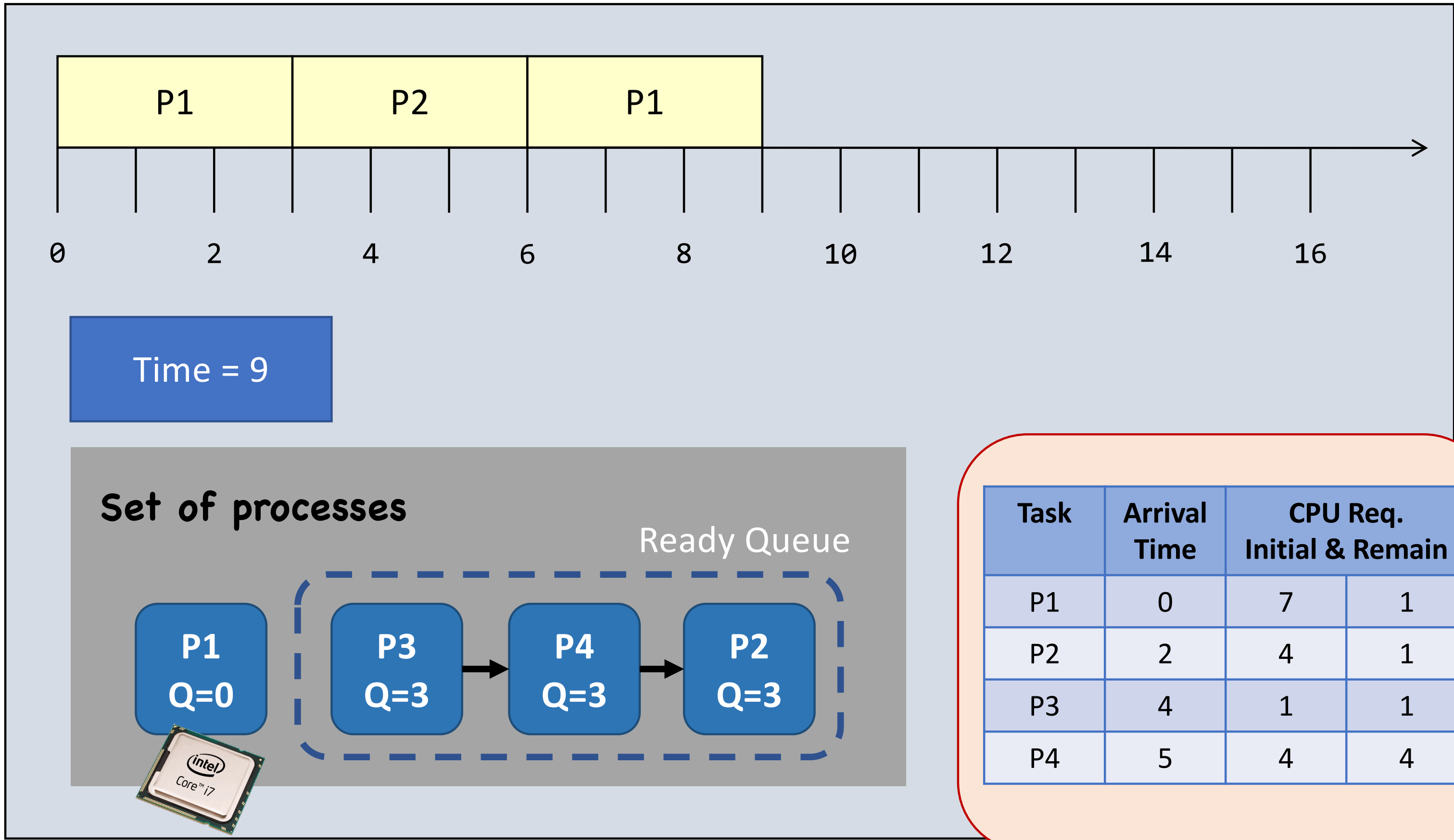




Task	Arrival Time	CPU Req. Initial & Remain	
P1	0	7	4
P2	2	4	1
P3	4	1	1
P4	5	4	4

Animation; don't print

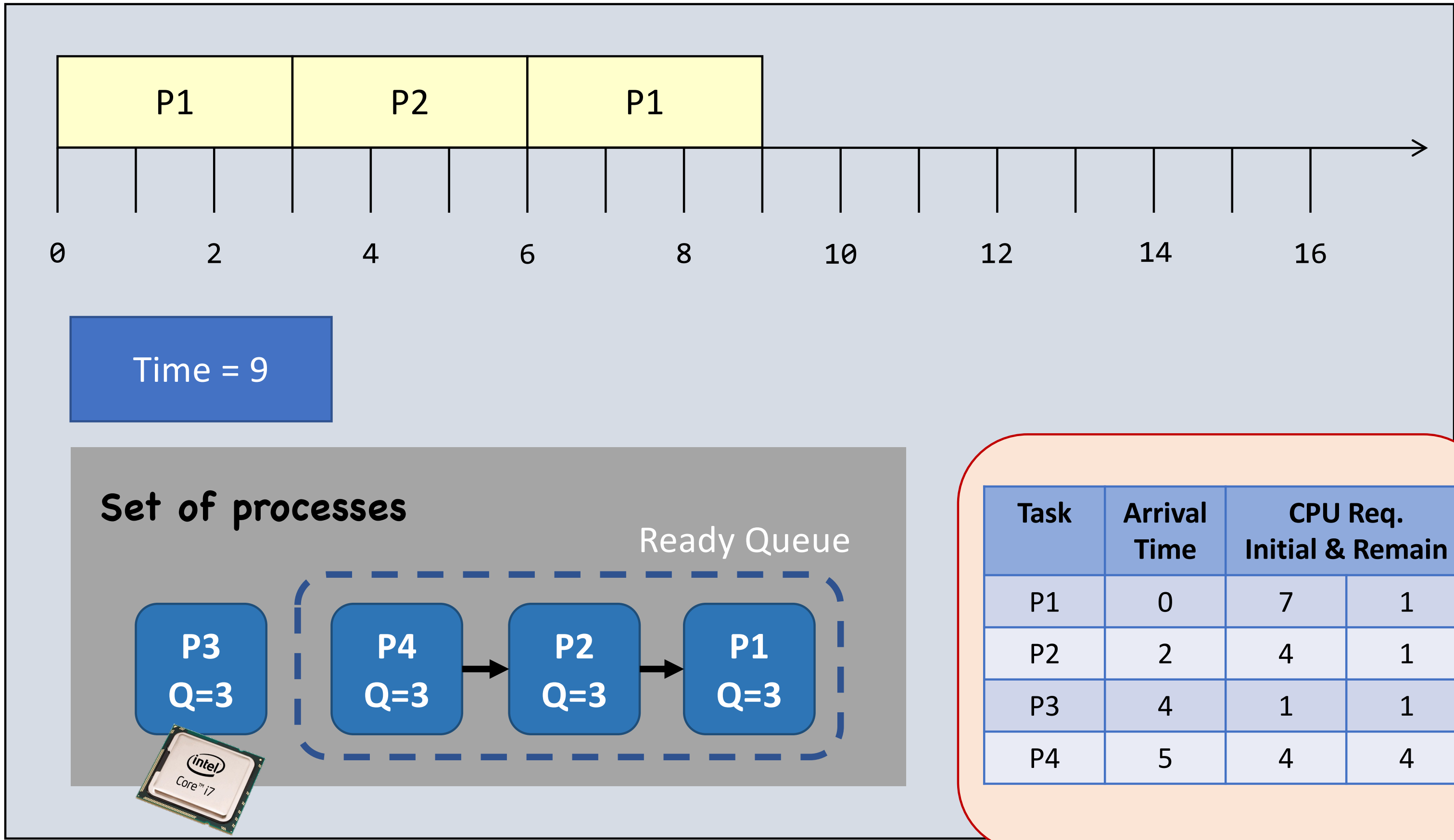
Round Robin (Quantum = 3)



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

Animation; don't print

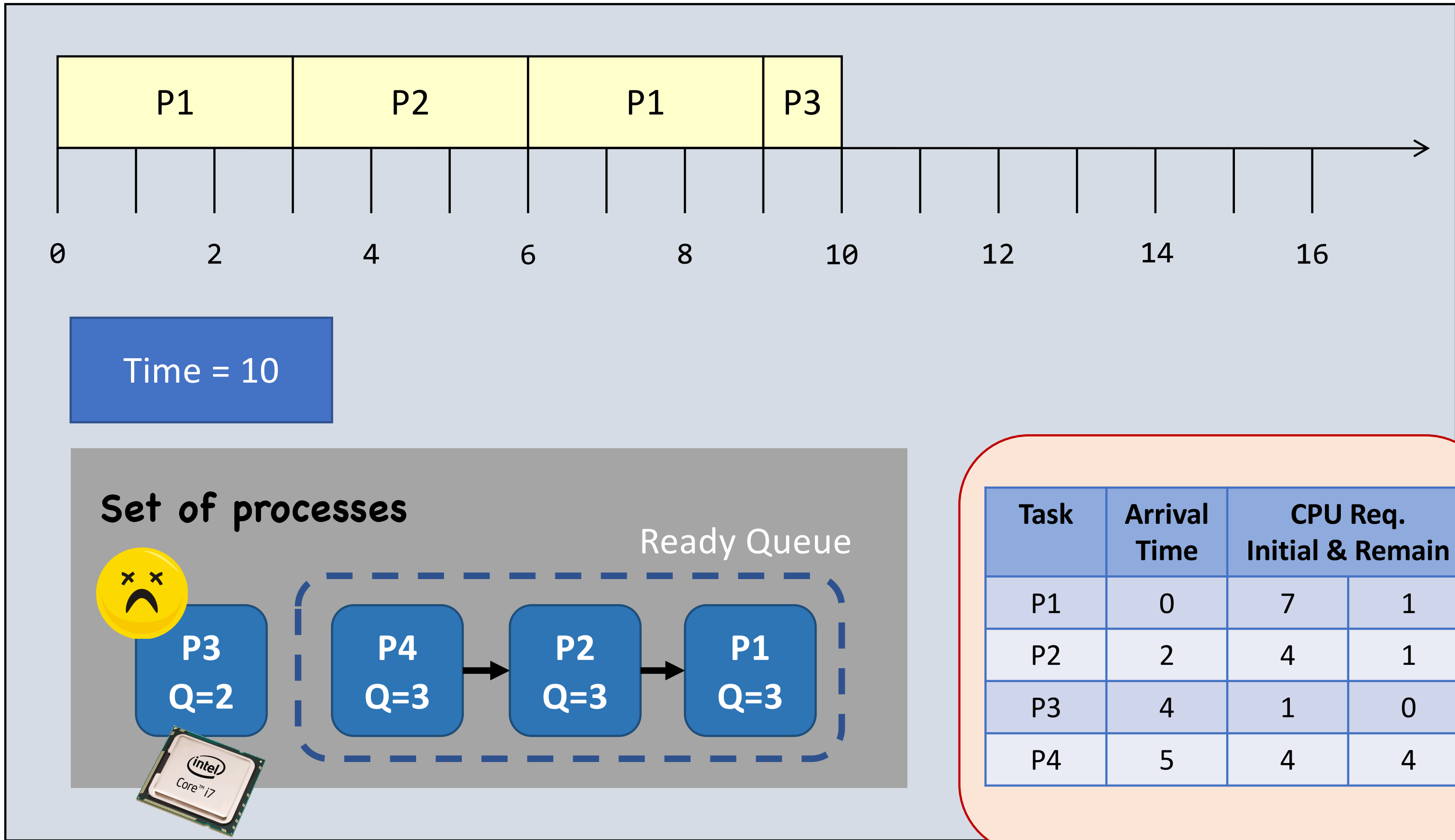
Round Robin (Quantum = 3)



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

Animation; don’t print

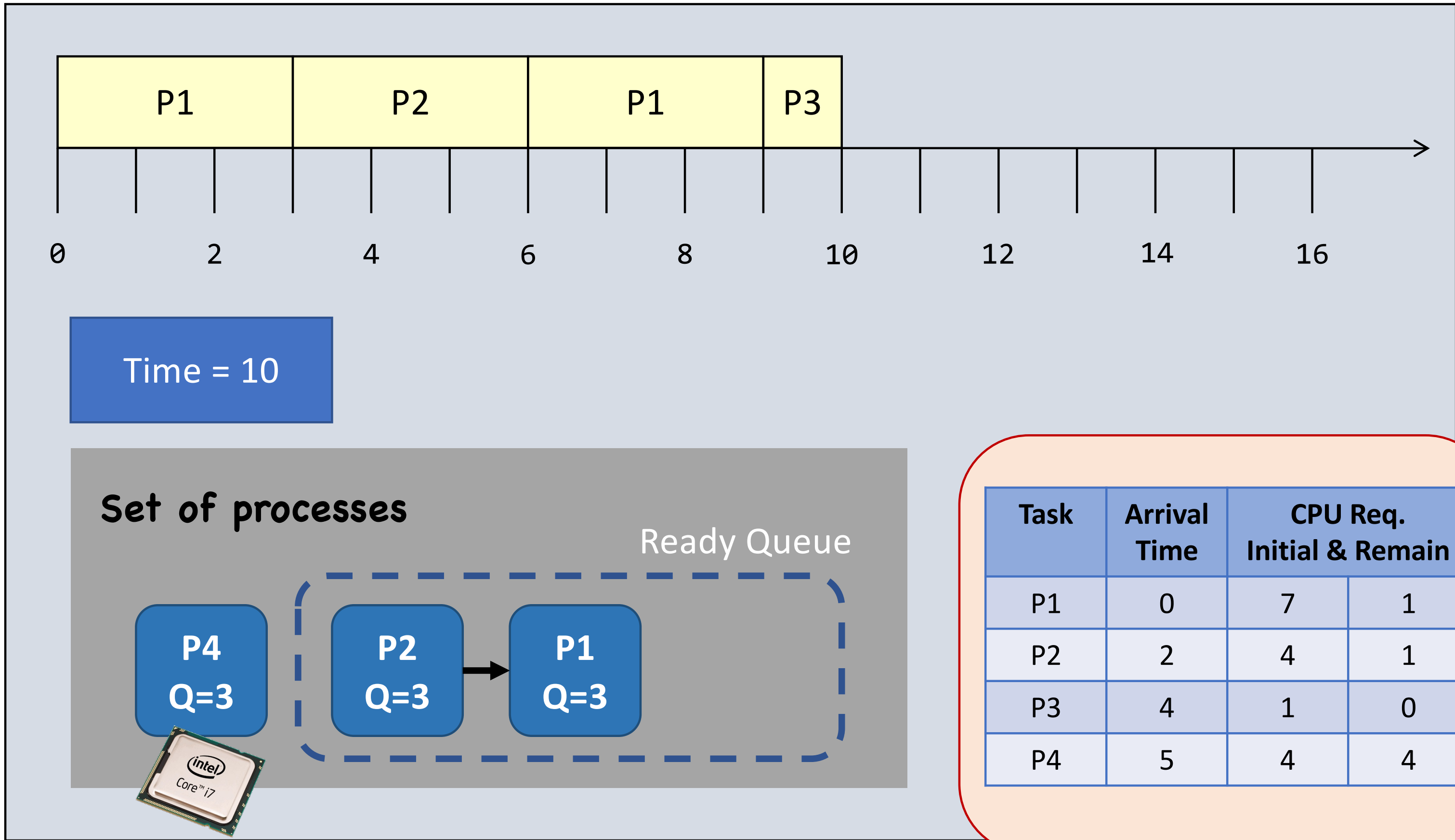
Round Robin (Quantum = 3)



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

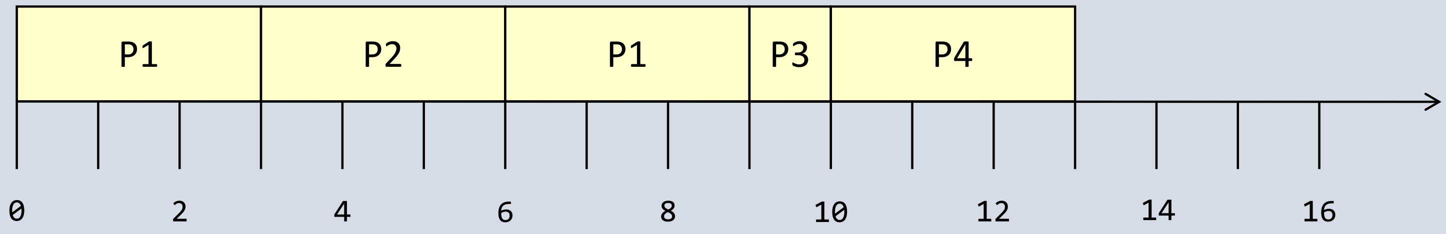
Animation; don't print

Round Robin (Quantum = 3)

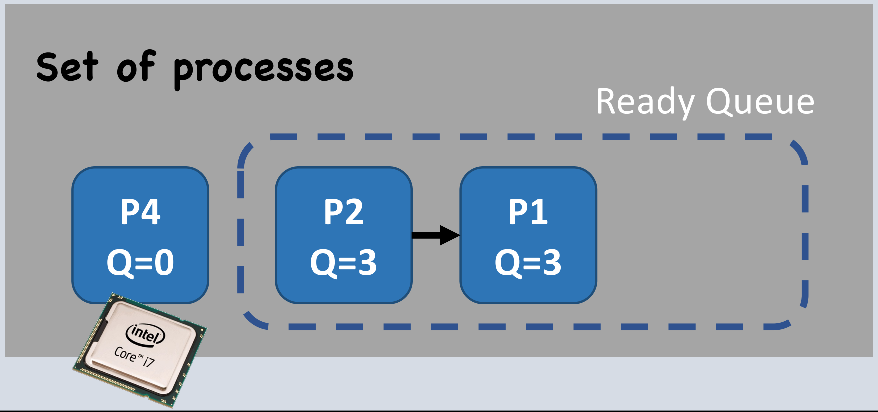


Task	Arrival Time	CPU Req. Initial & Remain	
P1	0	7	1
P2	2	4	1
P3	4	1	0
P4	5	4	4

Round Robin (Quantum = 3)



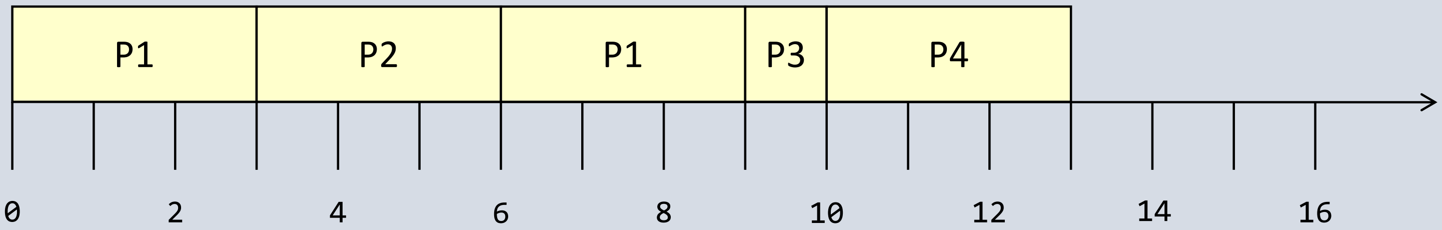
Time = 13



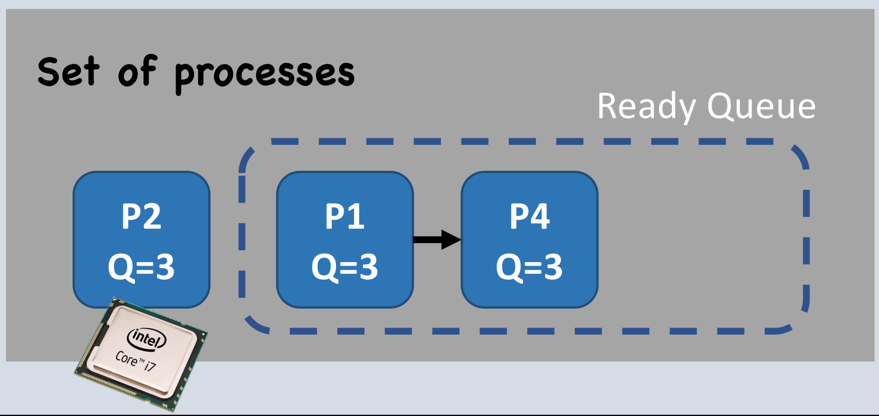
Task	Arrival Time	CPU Req. Initial & Remain	
P1	0	7	1
P2	2	4	1
P3	4	1	0
P4	5	4	1

Animation; don't print

Round Robin (Quantum = 3)



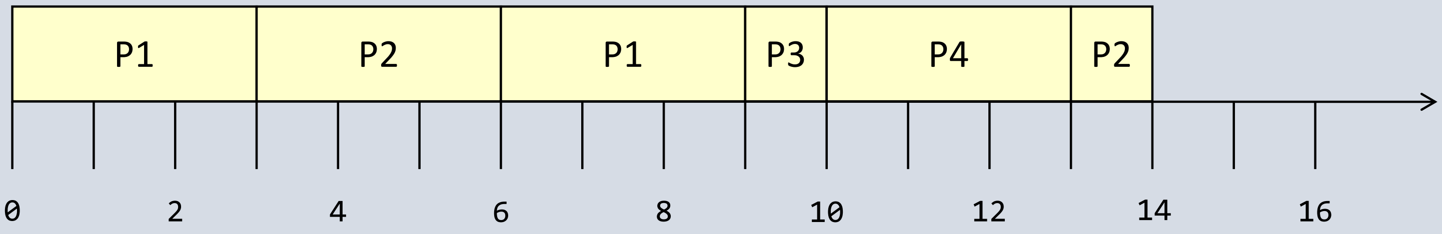
Time = 13



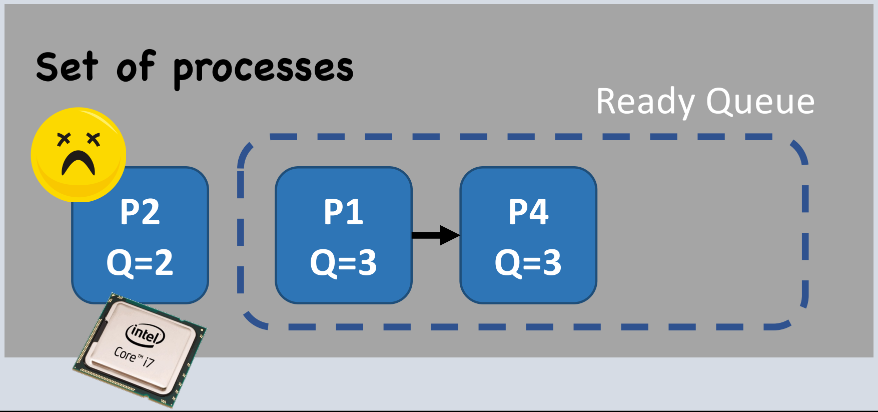
Task	Arrival Time	CPU Req. Initial & Remain	
P1	0	7	1
P2	2	4	1
P3	4	1	0
P4	5	4	1

Animation; don't print

Round Robin (Quantum = 3)



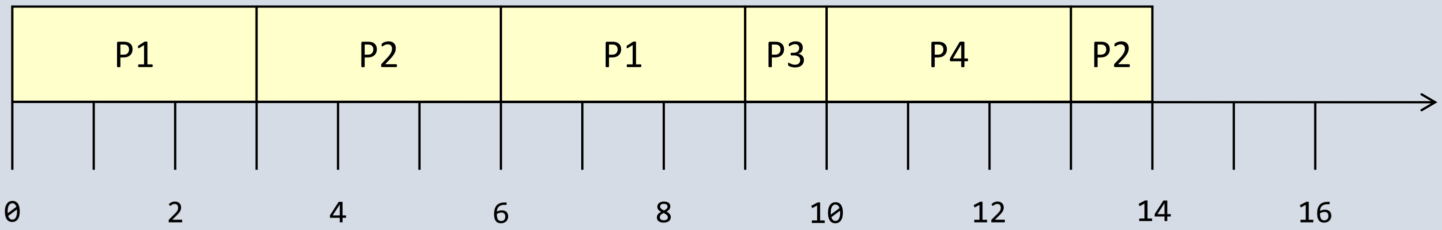
Time = 14



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

Animation; don't print

Round Robin (Quantum = 3)



Time = 14

Set of processes

Ready Queue

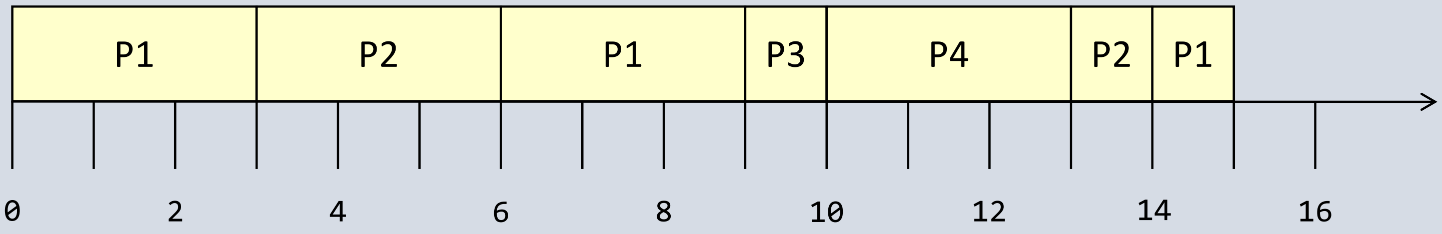
P1
Q=3

P4
Q=3

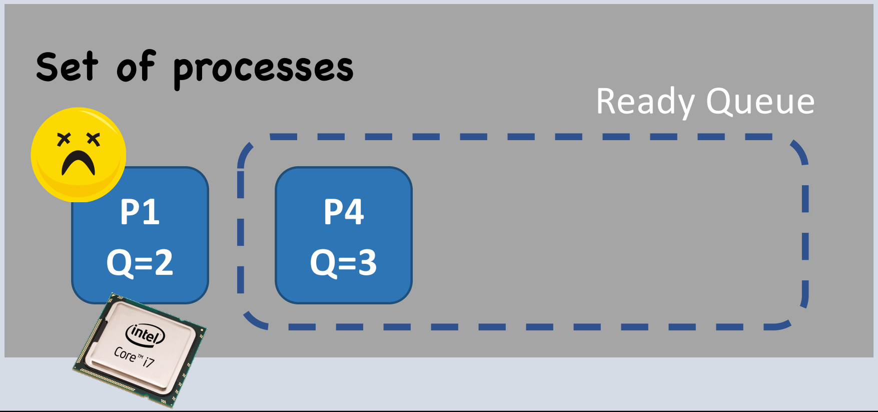
Task	Arrival Time	CPU Req. Initial & Remain	
P1	0	7	1
P2	2	4	0
P3	4	1	0
P4	5	4	1

Animation; don't print

Round Robin (Quantum = 3)



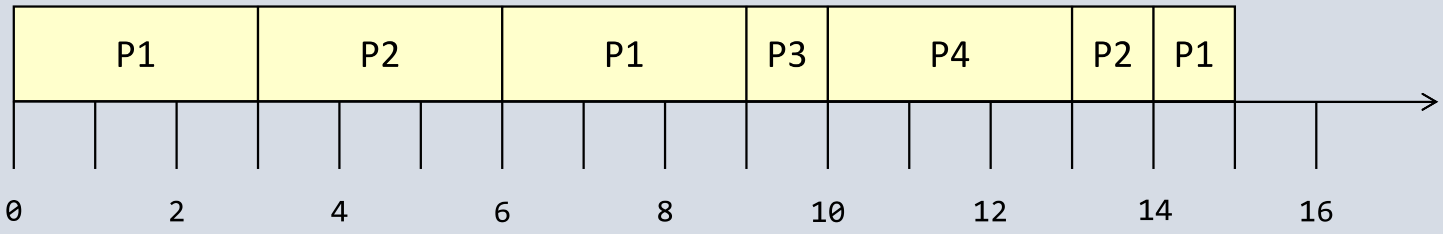
Time = 15



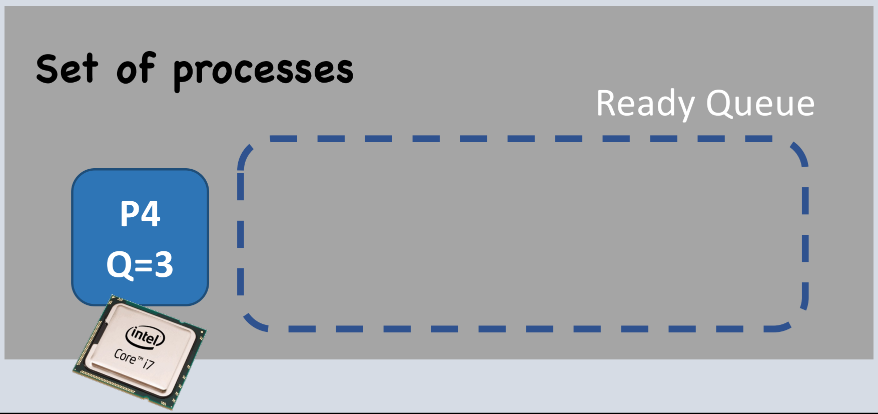
Task	Arrival Time	CPU Req. Initial & Remain	
P1	0	7	0
P2	2	4	0
P3	4	1	0
P4	5	4	1

Animation; don't print

Round Robin (Quantum = 3)



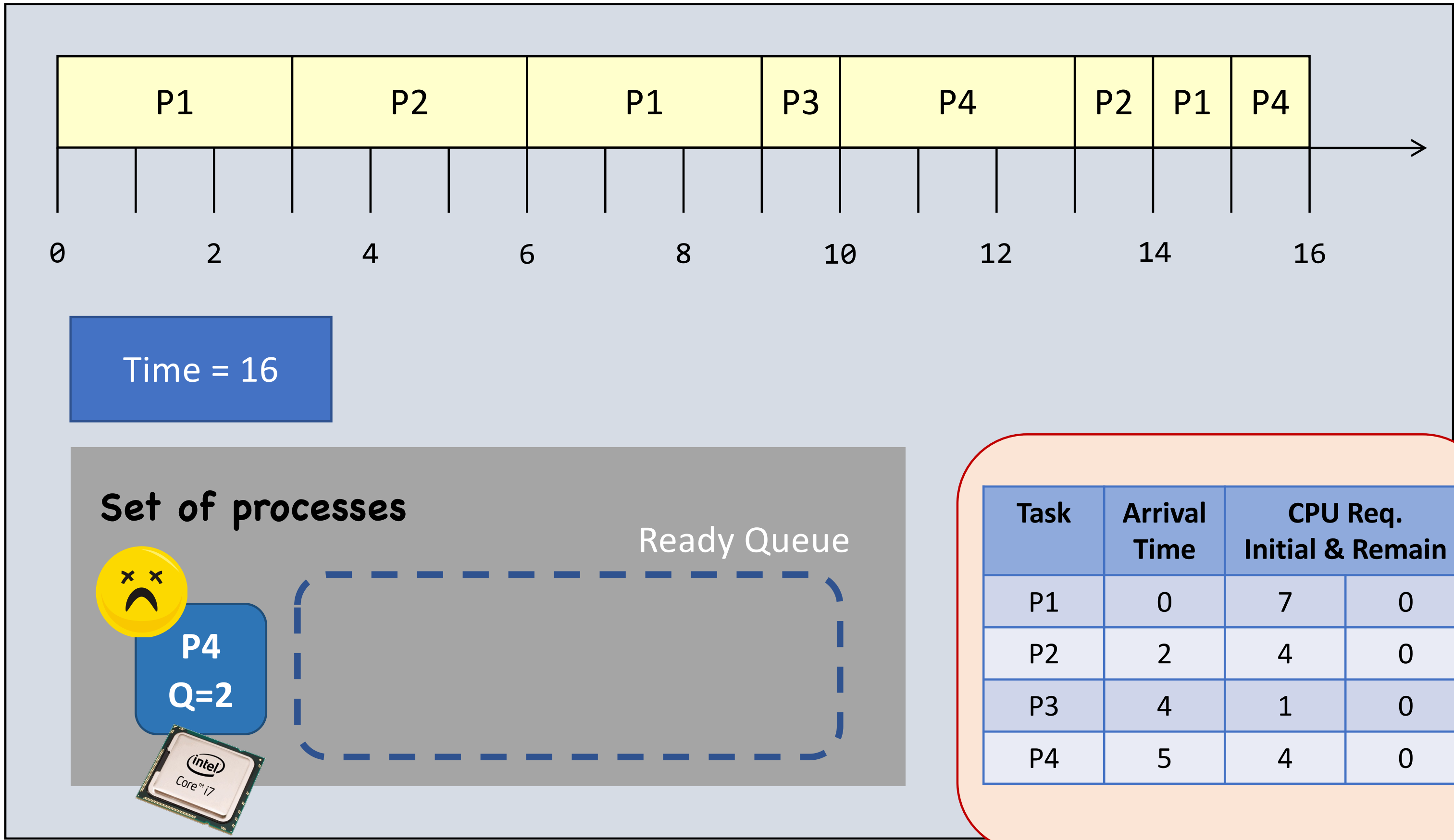
Time = 15



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

Animation; don't print

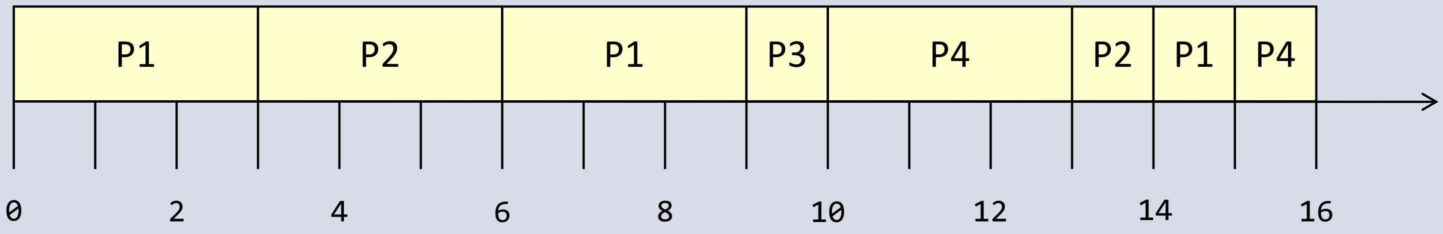
Round Robin (Quantum = 3)



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

Animation; don't print

Round Robin (Quantum = 3)



Waiting time:

P1 = 8; P2 = 8; P3 = 5; P4 = 7;

Average = (8 + 8 + 5 + 7) / 4 = 7

Turnaround time:

P1 = 15; P2 = 12; P3 = 6; P4 = 11;

Average = (15 + 12 + 6 + 11) / 4 = 11

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

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!

在 RR 调度下，每个进程都会定期获得 CPU 执行机会；

不会出现长任务一直霸占 CPU、短任务"卡住"的情况；

所以从用户视角来看，程序都"活着"，不会"假死（frozen）"。

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ 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

谁的 下一次 CPU burst time 最短，谁就优先执行；

Priority Scheduling (Cont'd)

- Problem ≡ **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 ≡ **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!