# Lecture4 CPU Scheduling

## 1. CPU Scheduling

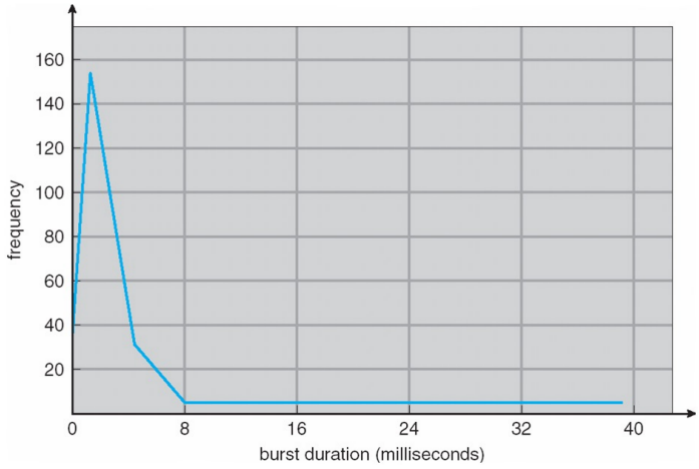
- Scheduling is important when multiple processes wish to run  on a single CPU
    - CPU scheduler decides which process to run next

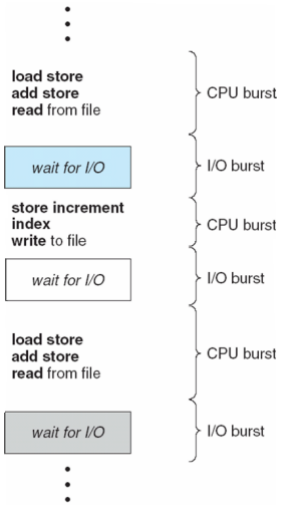
### Two types of processes

| | CPU-bound Process | I/O-bound process |
|---|---|---|
| **Characteristic** | Spends most of its running time on the CPU<br>user-time > sys-time | Spends most of its running time on I/O<br>sys-time > user-time |
| **Example** | AI course assignments | /bin/ls, 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**

# 2. Scheduling Algorithm Optimization Criteria

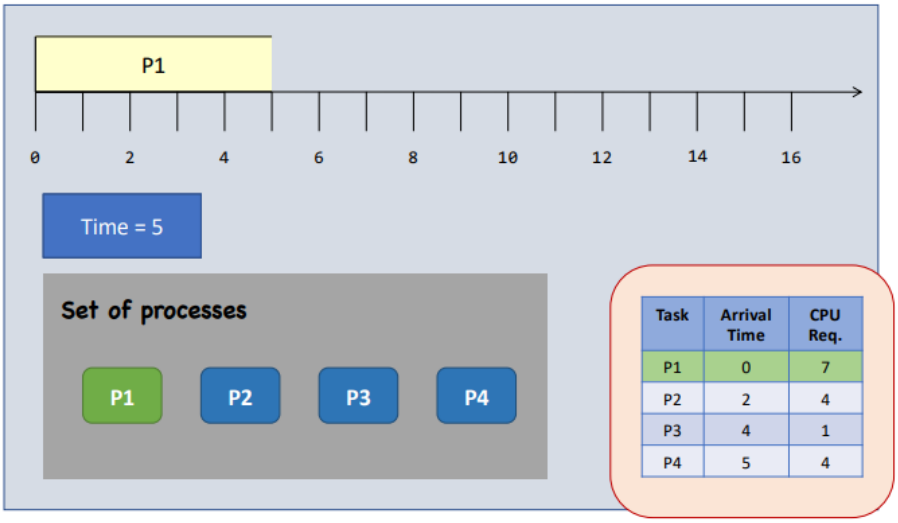
- Given a set of processes, with
    - **Arrival time**: the time they arrive in the CPU ready queue (from waiting state or from new state)
    - **CPU requirement**: their expected CPU burst time
- Minimize average **turnaround** time: The time between the arrival of the task and the time it is blocked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

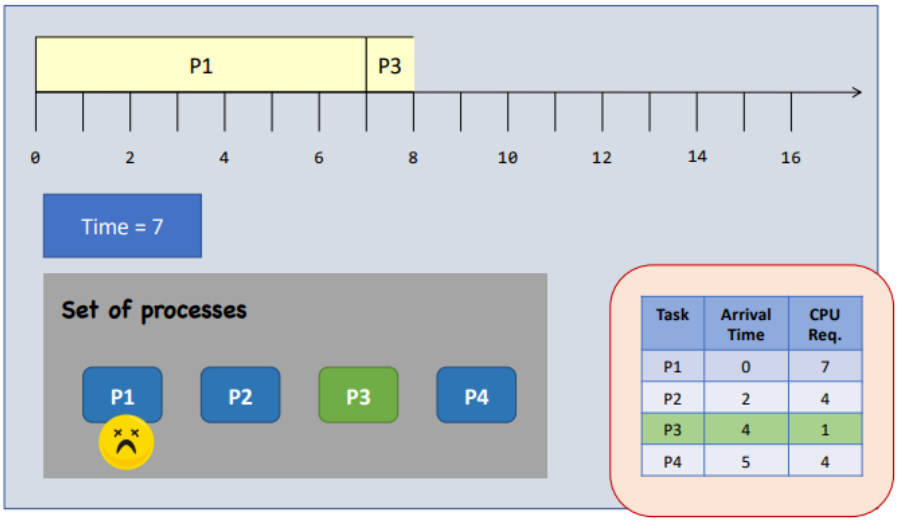
- Minimize average **waiting** time: The accumulated time that a task has waited in the ready queue
- Reduce the number of **context switches**

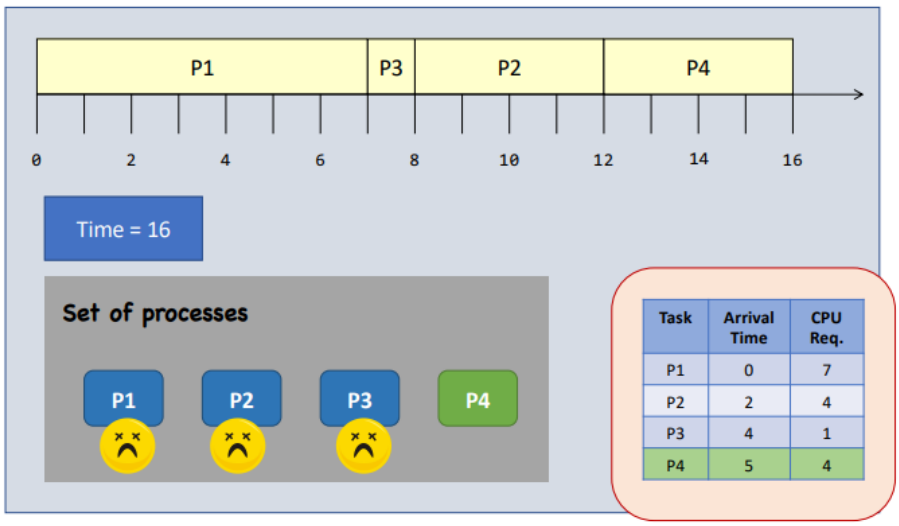
# Shortest-job-first (SJF)

## Non-preemptive SJF

Non-preemptive SJF will execute one process until it **finishes its CPU request**



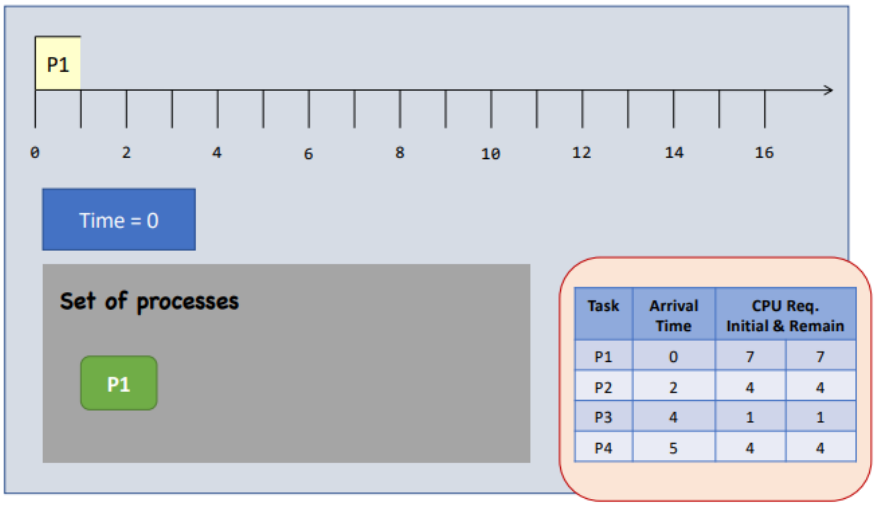



- waiting time:
  - P1 = 0, P2 = 6, P3 = 3, P4 = 7
  - Average = (0 + 6 + 3 + 7) / 4 = 4
- turnaround time:
  - P1 = 7, P2 = 10, P3 = 4, P4 = 11
  - Average = (7 + 10 + 4 + 11) / 4 = 8

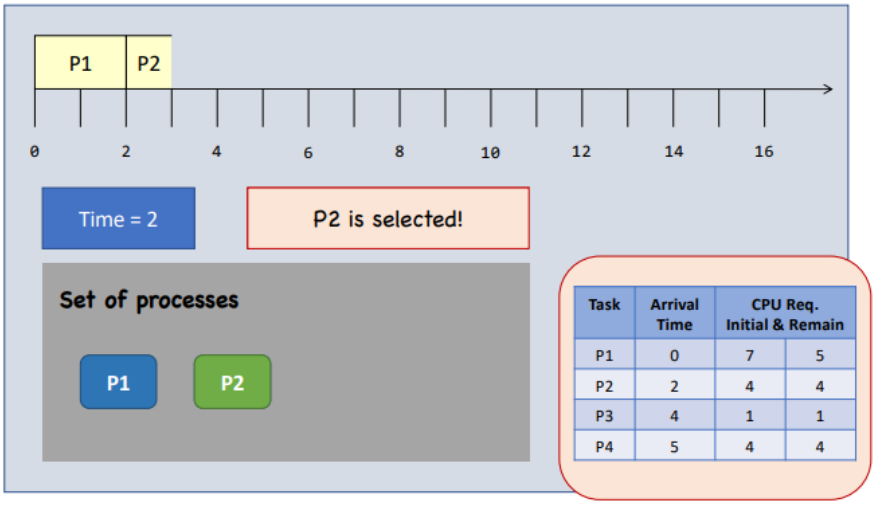
## Preemptive SJF

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**

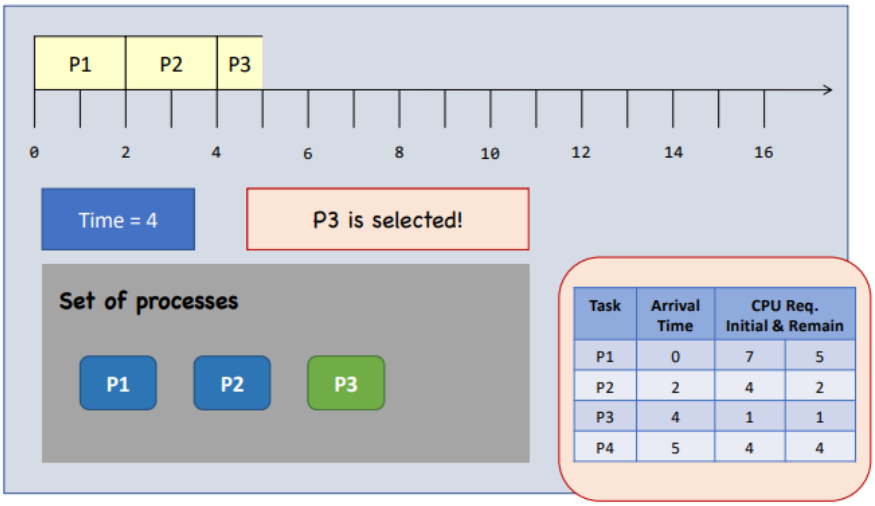
Suppose the time interrupt = 2

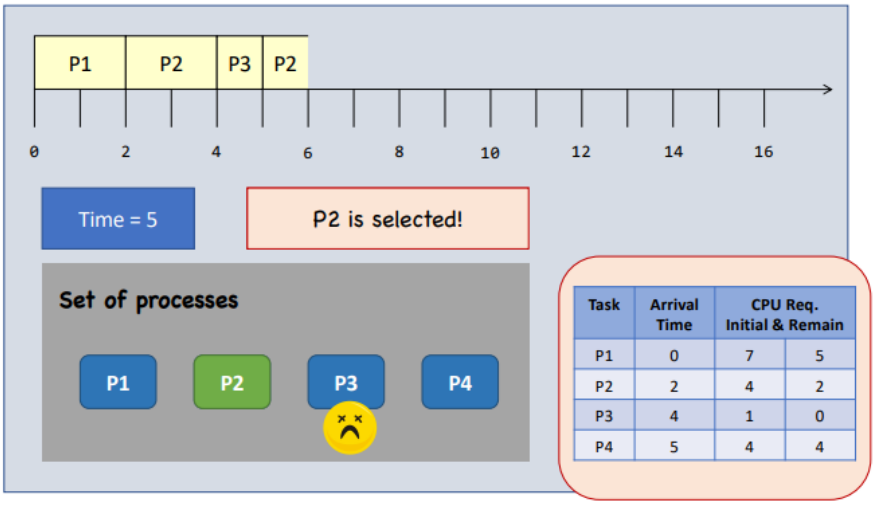


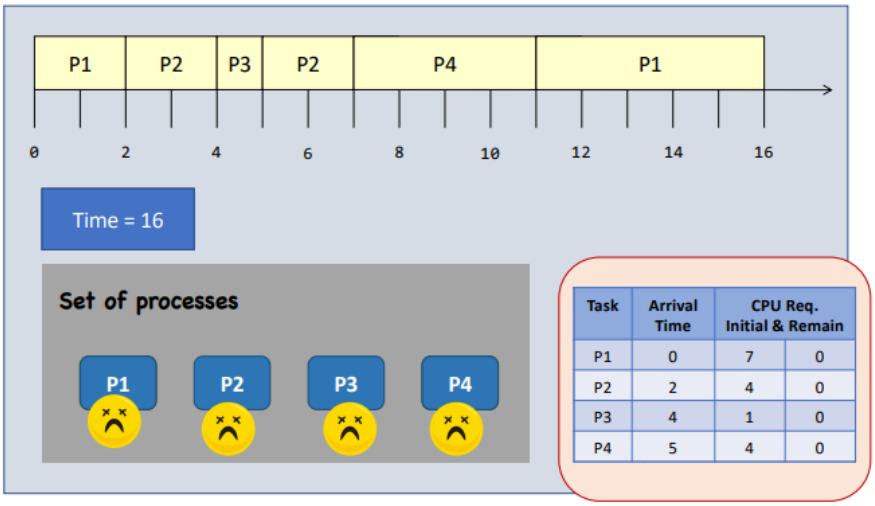

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 7 |
| P2 | 2 | 4 | 4 |
| P3 | 4 | 1 | 1 |
| P4 | 5 | 4 | 4 |




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




- 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

## 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**.

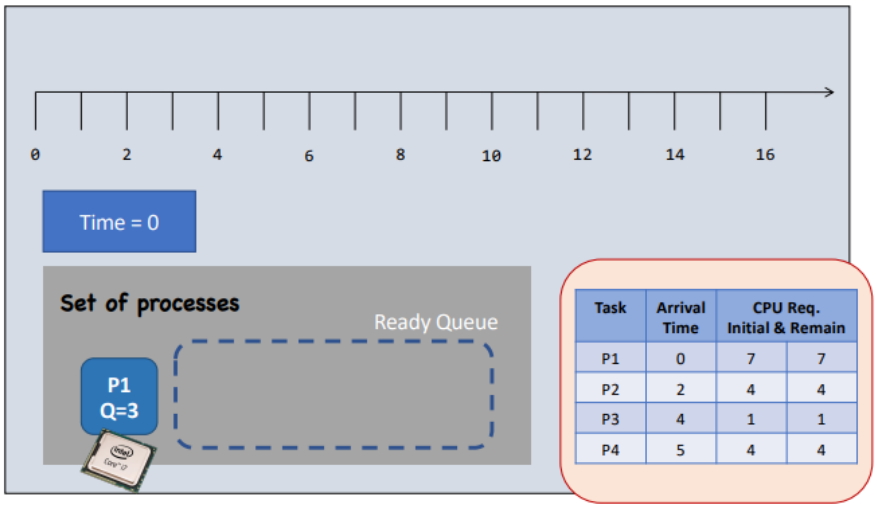
| Task | Arrival Time | CPU Req. |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |

# Round-robin (RR)

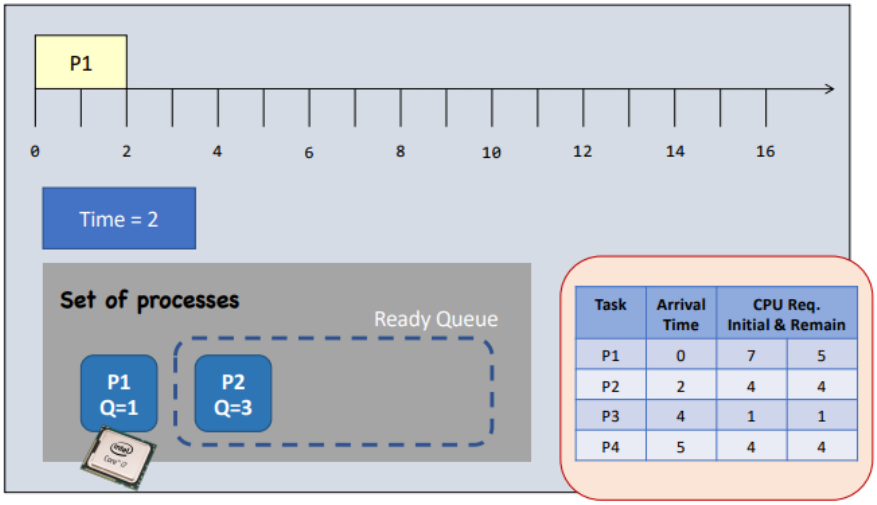
- Round-Robin (RR) scheduling is preemptive
  - Every process is given a **quantum** (the amount of time allowed to execute)
  - Whenever the quantum of a process is used up (i.e., 0), the process is preempted, placed at the end of the queue, with its quantum recharged
  - Then, the scheduler steps in 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

## RR with Quantum

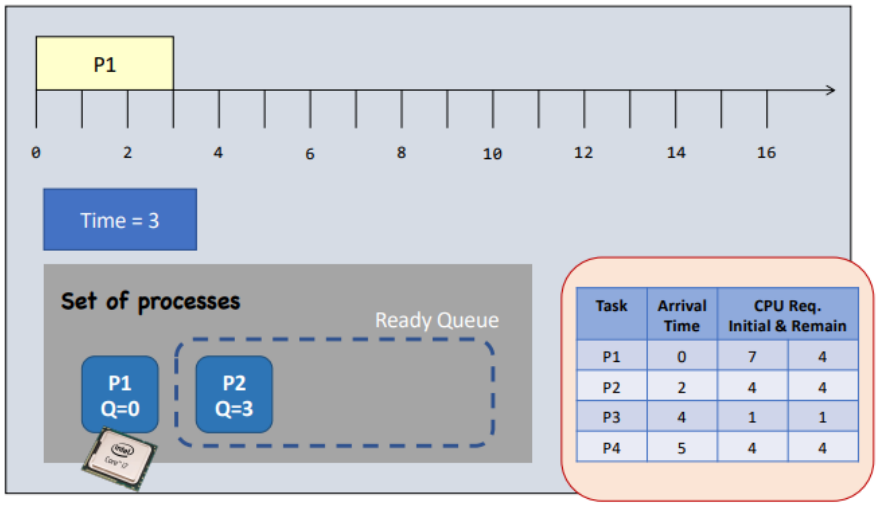
Suppose a RR schedule with Quantum = 3



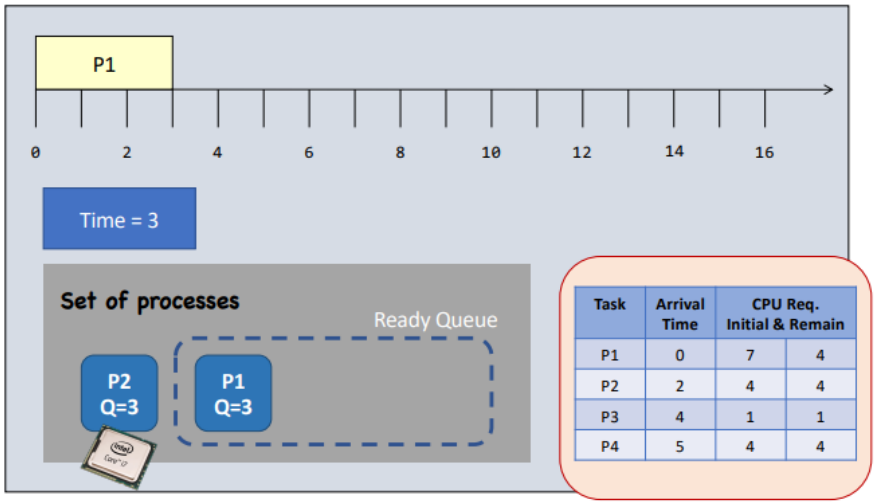

| Task | Arrival Time | CPU Req. Initial | CPU Req. Remain |
|---|---|---|---|
| P1 | 0 | 7 | 7 |
| P2 | 2 | 4 | 4 |
| P3 | 4 | 1 | 1 |
| P4 | 5 | 4 | 4 |



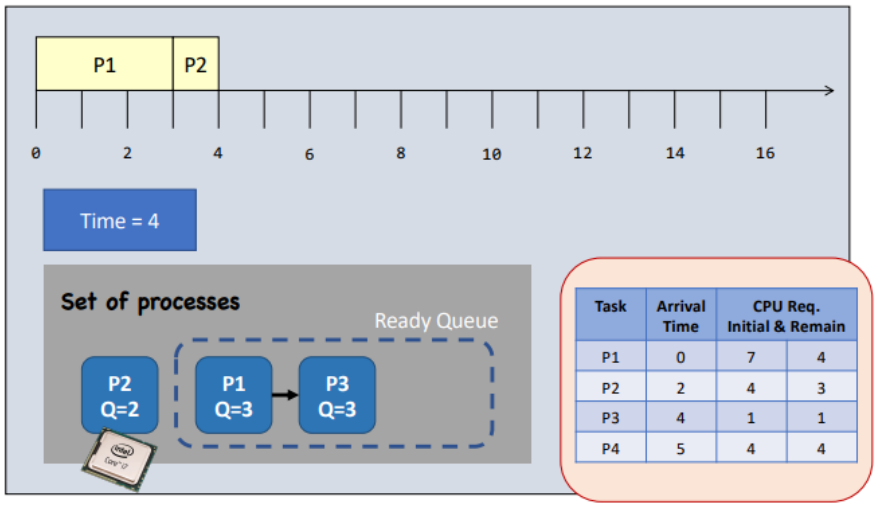

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 5 |
| P2 | 2 | 4 | 4 |
| P3 | 4 | 1 | 1 |
| P4 | 5 | 4 | 4 |



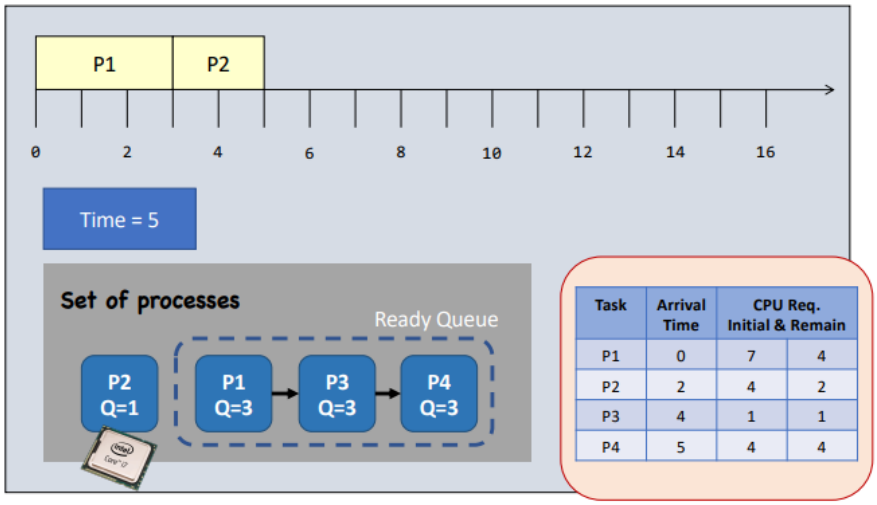

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 4 |
| P2 | 2 | 4 | 4 |
| P3 | 4 | 1 | 1 |
| P4 | 5 | 4 | 4 |



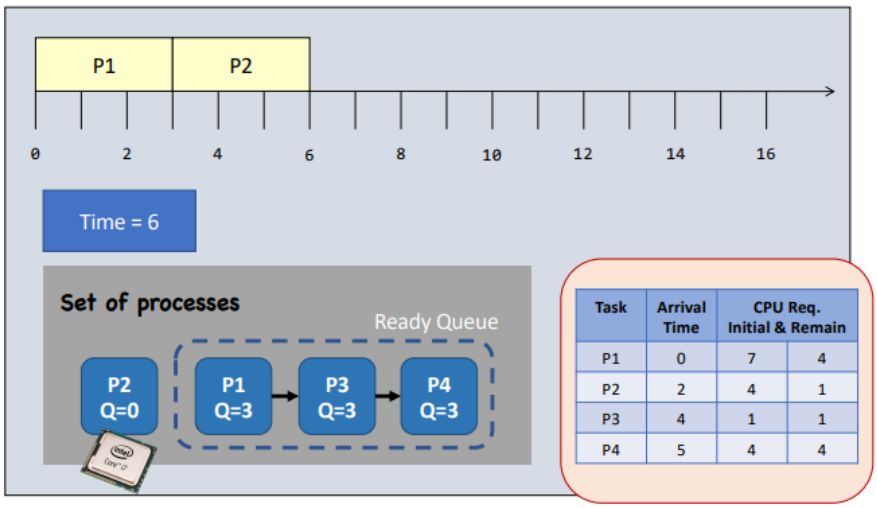

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 4 |
| P2 | 2 | 4 | 4 |
| P3 | 4 | 1 | 1 |
| P4 | 5 | 4 | 4 |



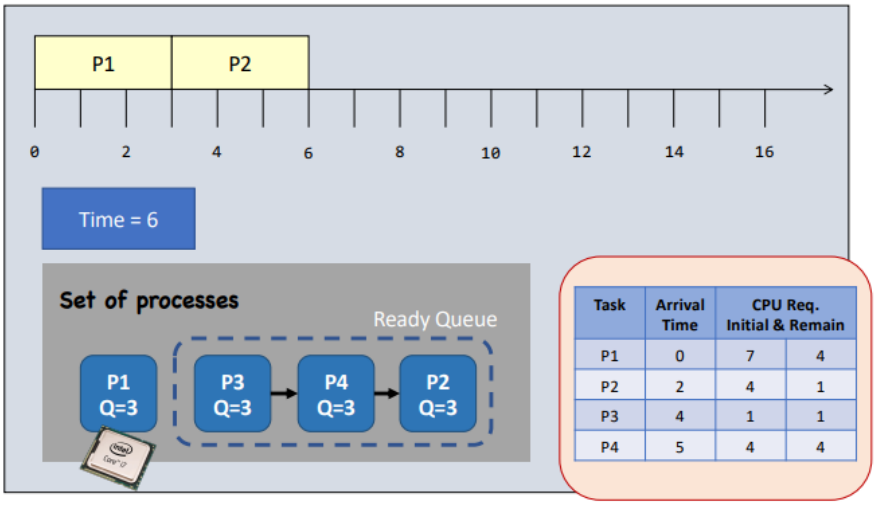

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 4 |
| P2 | 2 | 4 | 3 |
| P3 | 4 | 1 | 1 |
| P4 | 5 | 4 | 4 |



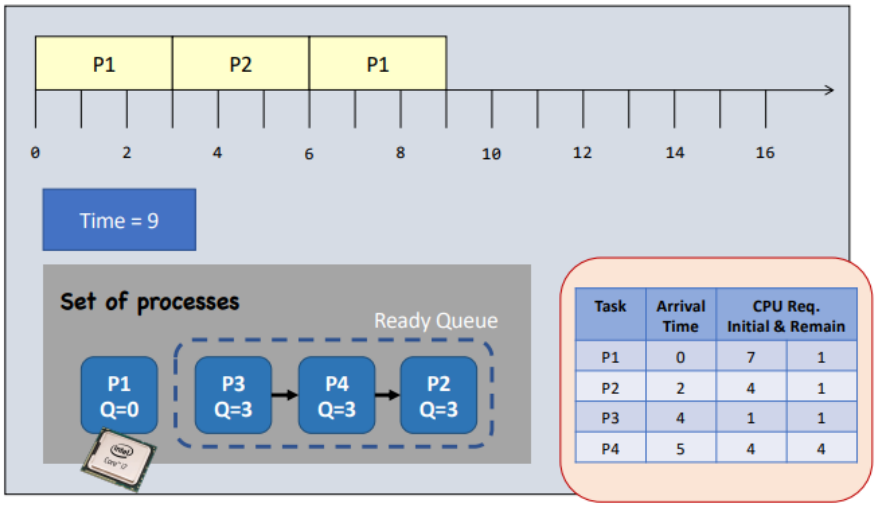

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 4 |
| P2 | 2 | 4 | 2 |
| P3 | 4 | 1 | 1 |
| P4 | 5 | 4 | 4 |



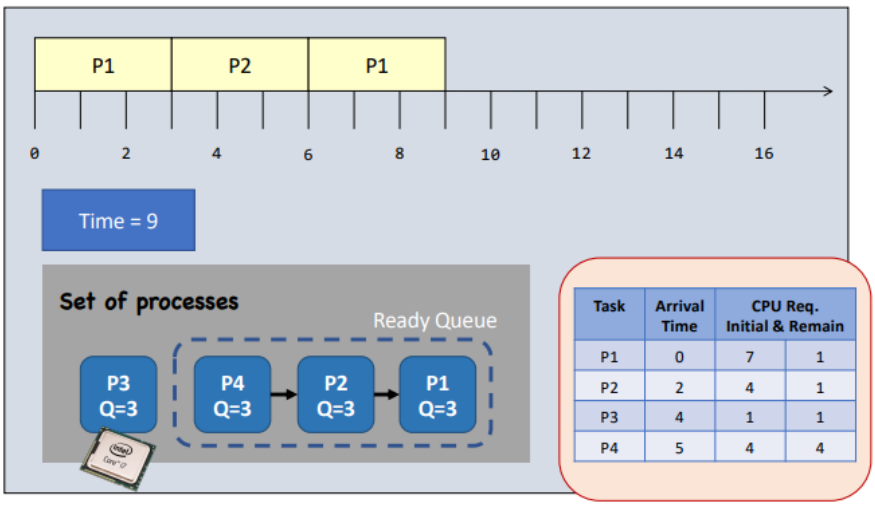

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 4 |
| P2 | 2 | 4 | 1 |
| P3 | 4 | 1 | 1 |
| P4 | 5 | 4 | 4 |



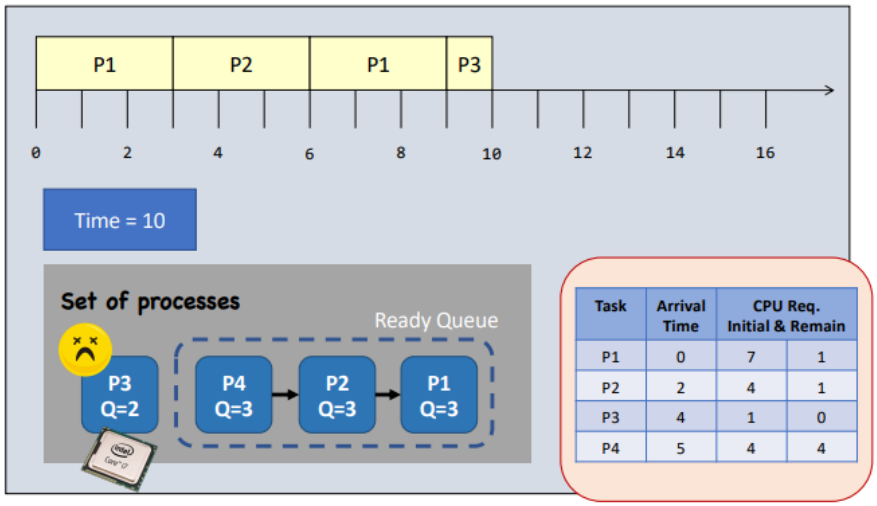

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 4 |
| P2 | 2 | 4 | 1 |
| P3 | 4 | 1 | 1 |
| P4 | 5 | 4 | 4 |



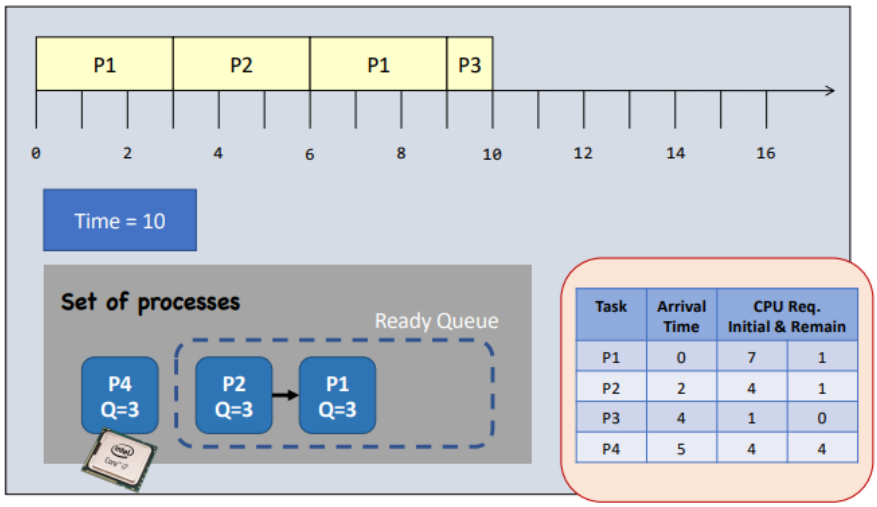

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 1 |
| P2 | 2 | 4 | 1 |
| P3 | 4 | 1 | 1 |
| P4 | 5 | 4 | 4 |



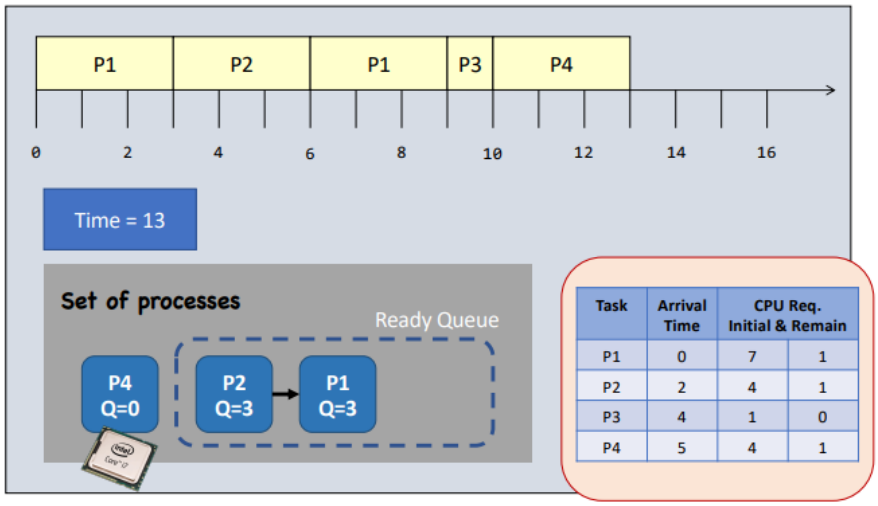

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 1 |
| P2 | 2 | 4 | 1 |
| P3 | 4 | 1 | 1 |
| P4 | 5 | 4 | 4 |



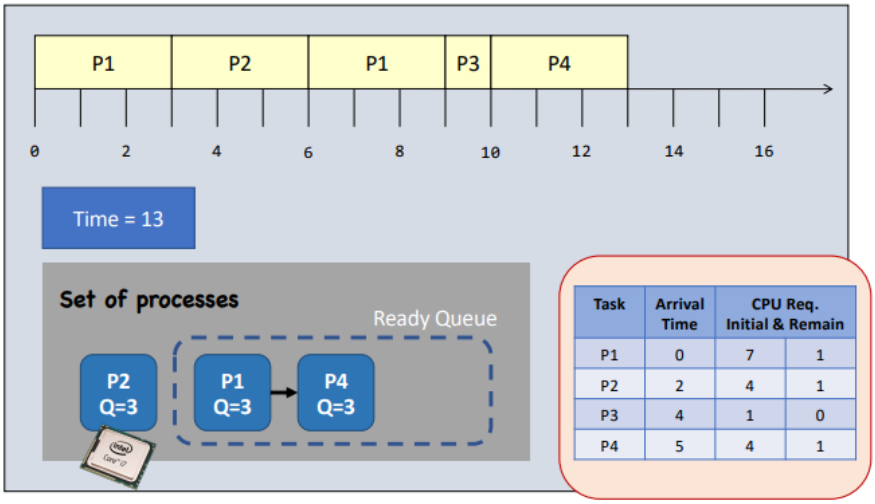

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 1 |
| P2 | 2 | 4 | 1 |
| P3 | 4 | 1 | 0 |
| P4 | 5 | 4 | 4 |



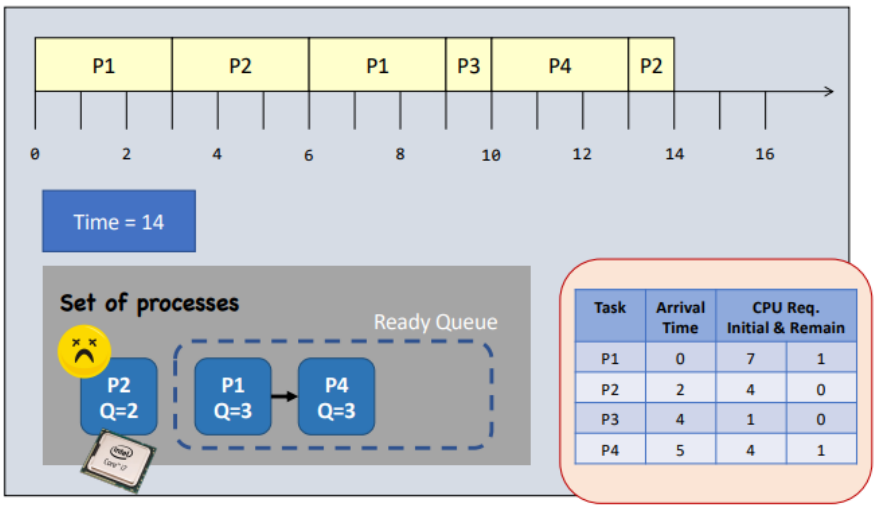

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 1 |
| P2 | 2 | 4 | 1 |
| P3 | 4 | 1 | 0 |
| P4 | 5 | 4 | 4 |



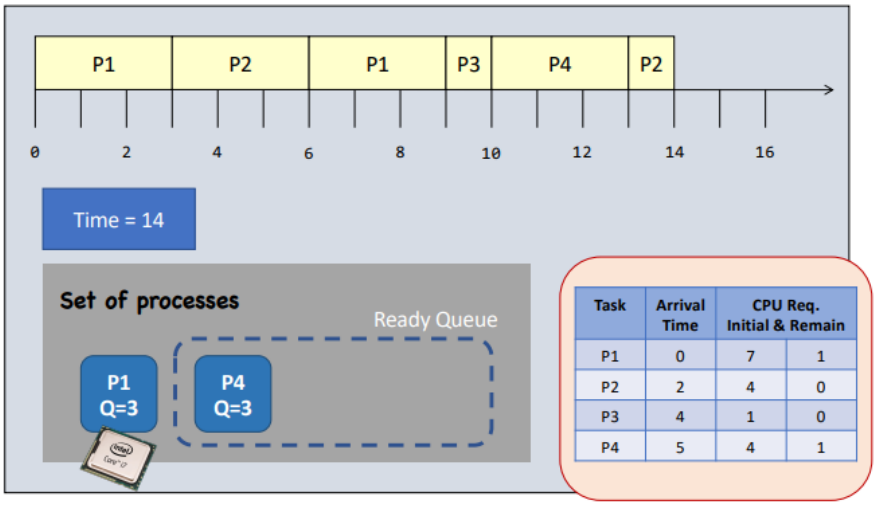

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 1 |
| P2 | 2 | 4 | 1 |
| P3 | 4 | 1 | 0 |
| P4 | 5 | 4 | 1 |



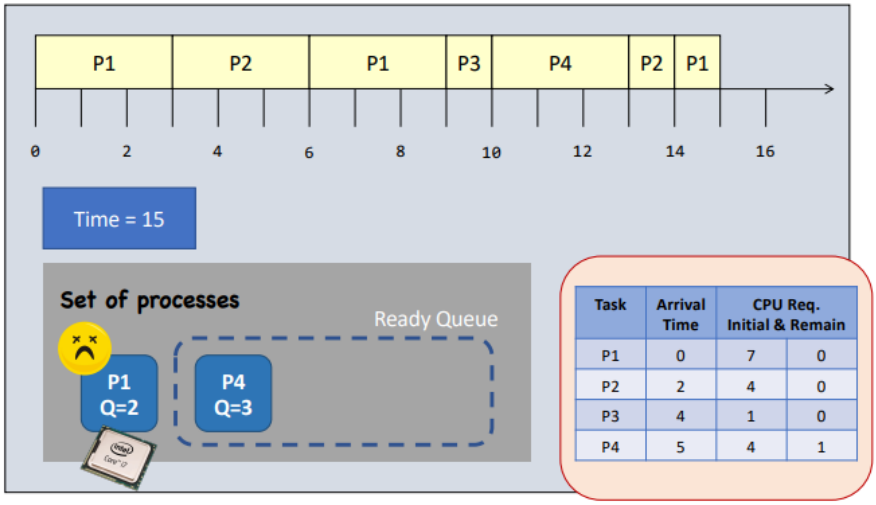

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 1 |
| P2 | 2 | 4 | 1 |
| P3 | 4 | 1 | 0 |
| P4 | 5 | 4 | 1 |

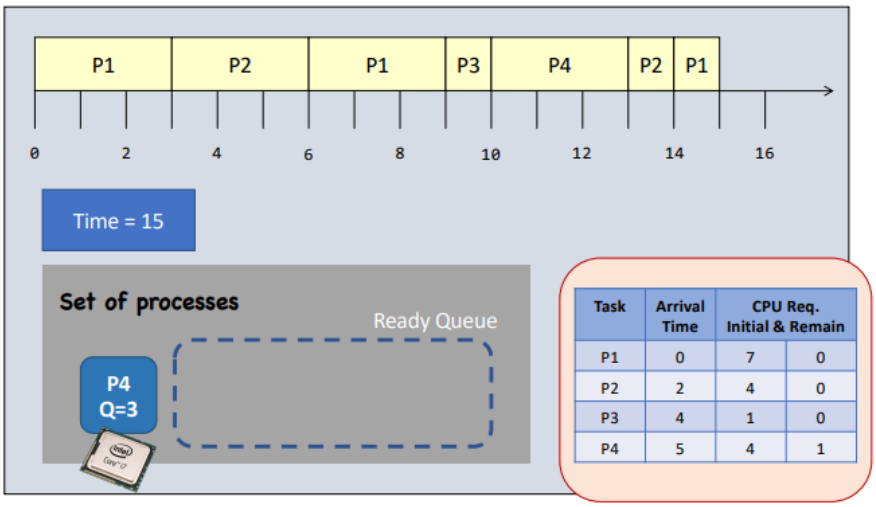


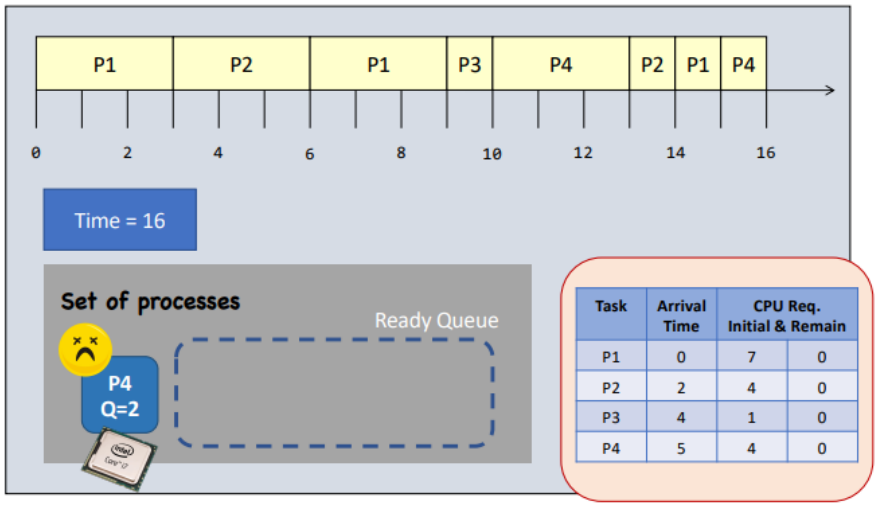

| Task | Arrival Time | CPU Req. Initial & Remain | |
|---|---|---|---|
| P1 | 0 | 7 | 1 |
| P2 | 2 | 4 | 0 |
| P3 | 4 | 1 | 0 |
| P4 | 5 | 4 | 1 |




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



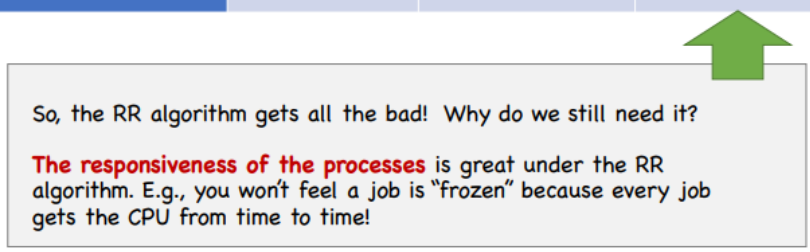




- waiting time
  - P1 = 8, P2 = 8, P3 = 5, P4 = 7
  - Average = (8 + 8 + 5 + 7) / 4 = 7
- turnaround time
  - P1 = 15, P2 = 12, P3 = 6, P4 = 11
  - Average = (15 + 12 + 6 + 11) / 4 = 11

### 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) |




## 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)
  - Non-preemptive: 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

### Problem

- Problem **Starvation**, low priority processes may never execute
- Solution: **Aging**, as time progresses increase the priority of the process
  - 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

## Completely Fair Scheduler

- Scheduling class
  - Default scheduling class: CFS
  - Real-time scheduling class
- Varying length scheduling quantum
  - Traditional UNIX scheduling uses 90ms fixed scheduling quantum
  - CFS assigns a proposition of CPU processing time to each task
- **Nice value**

    - -20 to +19, default nice is 0
    - **Lower nice value** indicates a **higher relative priority**
    - Higher value is “being nice”
    - Task with lower nice value receives higher proportion of CPU time
- **Virtual run time**
    - Each task has a per-task variable `vruntime`
    - Decay factor
        - Lower priority has higher rate of decay
        - nice = 0 virtual run time is identical to actual physical run time
        - A task with nice > 0 runs for 200 milliseconds, its `vruntime` will be **higher than** 200 milliseconds
        - A task with nice < 0 runs for 200 milliseconds, its `vruntime` will be **lower than** 200 milliseconds
- Lower virtual run time, higher priority
    - To decide which task to run next, scheduler chooses the task that has the **smallest** `vruntime` value
    - Higher priority can preempt lower priority

## Example

- Two tasks have the same nice value
    - One task is I/O bound and the other is CPU bound
    - `vruntime` of I/O bound will be shorter than `vruntime` of CPU bound
    - I/O bound task will eventually have higher priority and preempt CPU-bound tasks whenever it is ready to run