Operating System

BCSE303L

Module 3 - Scheduling

Dr. Naveenkumar Jayakumar
Associate Professor
Department of Computational Intelligence
PRP 217-4
naveenkumar.jk@vit.ac.in

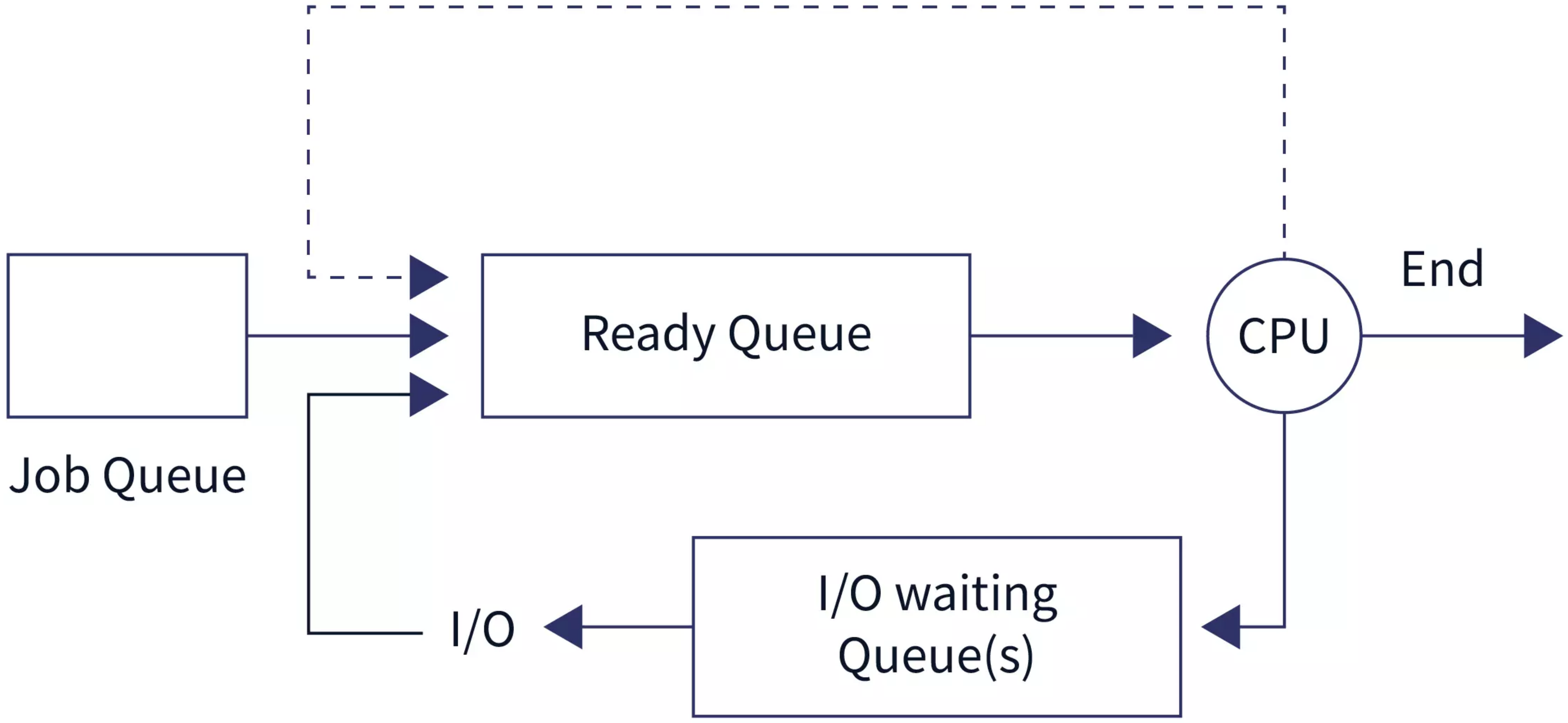
Scheduling - Definition

- The activity of **deciding which process gets** to use the **CPU** at any given time.
- It is the mechanism by which the process manager removes a currently running process from the CPU and selects another one to take its place, based on a specific strategy or algorithm.
- In multiprogramming operating systems, which allow multiple processes to be loaded into memory simultaneously and share the CPU over time.
- It is the mechanism of selecting a process from a ready queue and allotting CPU to this process for execution.

Scheduling - Objectives

- The operating system schedules the processes in such a way that the **CPU doesn’t sit idle** and keeps processing some or the other process.
- The **primary objective** of process scheduling is to optimize system performance according to several key metrics:
  - **Maximize CPU Utilization** Keep the CPU as busy as possible to prevent wasted cycles.
  - **Minimize Response Time** Reduce the time it takes from a user's request to the start of the response.
  - **Minimize Waiting Time** Decrease the amount of time a process spends in the ready queue waiting for the CPU.
  - **Fair Allocation** Ensure each process gets a fair share of the CPU's time.

Scheduling - Working



Scheduling - Working

- Scheduling is managed by system software called schedulers, which move processes between different states using scheduling queues.
- The operating system maintains several queues to manage processes
  - **Job Queue/Task Queue/Process Queue:** Contains all processes in the system.
  - **Ready Queue** Holds processes that are in main memory and are ready and waiting to execute.
  - **Device Queues/I/O Queue/Wait Queue:** Contain processes that are blocked because they are waiting for an I/O device to become available

Scheduling - Working

- ❑ There are generally three types of schedulers that operate at different frequencies and for different purposes
  - ❑ **Long-Term Scheduler (or Job Scheduler)** This scheduler selects processes from the job queue and loads them into memory for execution. It controls the degree of multiprogramming (the number of processes in memory) and aims to create a balanced mix of CPU-bound and I/O-bound jobs.
  - ❑ **Short-Term Scheduler (or CPU Scheduler)** This is the most frequently executed scheduler. It selects a process from the ready queue and allocates the CPU to it. Its main goal is to optimize CPU performance and utilization.
  - ❑ **Medium-Term Scheduler (Present in some systems)** This scheduler is involved in swapping processes out of memory to reduce the degree of multiprogramming and later swapping them back in to continue execution.

Scheduling - Categories

- ❑ **Non-Preemptive Scheduling** Once the CPU has been allocated to a process, that process keeps the CPU until it either terminates or switches to a waiting state (e.g., for an I/O operation). **The resource cannot be forcibly taken away**.

- ❑ **Preemptive Scheduling** The operating system can forcibly remove a running process from the CPU and reallocate it to another process. This often happens when a higher-priority process arrives or when a running process has exceeded its allocated time slice

Scheduling – Metrics Measured

- **CPU Burst Time**
  - CPU Burst Time is the amount of time a process needs to run on the CPU to complete its computations.
  - A process's life is made up of cycles of CPU bursts and I/O waiting periods
  - CPU bursts can vary in length depending on the nature of the process
  - CPU burst time is critical for scheduling algorithms to determine when to switch processes to optimize CPU utilization and responsiveness

Scheduling – Metrics Measured

- ❑ **I/O Burst Time**
  - ❑ I/O Burst Time is the period a process spends waiting for an Input/Output (I/O) operation to finish .
  - ❑ This happens when the process needs to read data from a disk, get input from a user, or send data to a printer .
  - ❑ During this time, the process is not using the CPU; it is "blocked" and waiting for an external device.
  - ❑ I/O bursts are typically followed by CPU bursts as processes alternate between computation and I/O activities.

Scheduling – Metrics Measured

- Arrival Time
  - Arrival Time is the exact moment a process enters the ready queue, meaning it is ready to be executed and is waiting for the CPU to become available .
  - It is the official entry time of a process into the system to compete for CPU time

Scheduling – Metrics Measured

- **Completion Time**
  - Completion Time is the time at which a process finishes its execution and exits the system
  - It marks the moment the process has completed all its CPU bursts and I/O operations and is officially done.

Scheduling – Metrics Measured

- **Turnaround Time**
  - Turnaround Time is the total time a process spends in the system, from its arrival to its completion .
  - It measures the entire lifecycle of a process and includes both the time spent executing on the CPU and the time spent waiting (in the ready queue and for I/O)

Turnaround Time = Completion Time - Arrival Time

Scheduling – Metrics Measured

- **Waiting Time**
  - Waiting Time is the total time a process spends in the ready queue, waiting for its turn to use the CPU .
  - It specifically measures the time a process is ready to run but is forced to wait because the CPU is busy with other processes.
  - It does not include the time spent executing or waiting for I/O.

Waiting Time = Turnaround Time - CPU Burst Time

Scheduling – Metrics Measured

- ❑ **Response Time**
  - ❑ Response Time is the time elapsed from when a process arrives until it gets the CPU for the first time .
  - ❑ This metric is crucial for interactive systems because it reflects how quickly the system acknowledges and begins to handle a user's request.
  - ❑ The time from when a process is submitted until the first response is produced

Response Time = Time of First Response - Arrival Time

Scheduling – Uniprocessor - FCFS

- **First Come First Serve**
  - It is a **non-pre-emptive, arrival-order scheduling policy** for the **ready queue** in a uniprocessor system
  - Every newly admitted process Pi is appended to the tail of the queue.
  - The dispatcher removes the head process, loads its context, and runs it until it voluntarily yields:
    - terminates, or
    - executes a blocking system call (e.g., I/O, semaphore P operation, page fault).

Scheduling – Uniprocessor - FCFS

❑ First Come First Serve

- ❑ When the running process blocks, the dispatcher selects the next ready process (again, queue head).
- ❑ When a blocked process becomes ready, it is always enqueued at the tail—never reinserted ahead of older entrants.
- ❑ Because no timer pre-emption occurs, the CPU busy interval is a sequence of maximal CPU bursts.

Scheduling – Uniprocessor - FCFS

- **First Come First Serve**
  - The ready queue is treated as a simple FIFO: the process that arrives first gets the CPU first, and once a process starts execution it runs to completion of its current CPU burst (i.e. non-preemptive).

Scheduling – Uniprocessor - FCFS

❑ First Come First Serve – Example

Consider the system image at the current moment and suggest how FCFS will work in this:

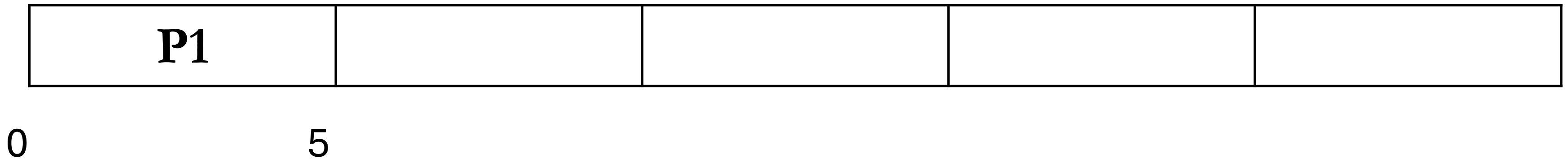
Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Scheduling – Uniprocessor - FCFS

❑ First Come First Serve – Example

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Constructing Gantt Chart

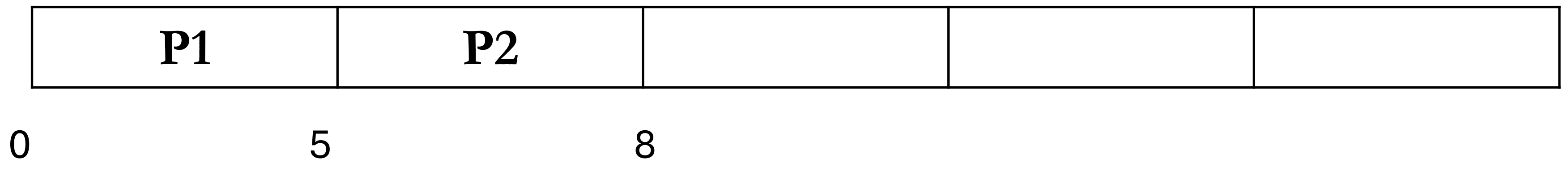


Scheduling – Uniprocessor - FCFS

❑ First Come First Serve – Example

Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

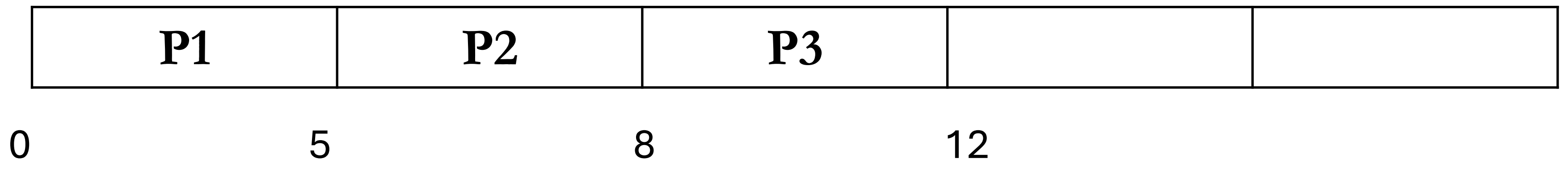


Scheduling – Uniprocessor - FCFS

❑ First Come First Serve – Example

Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

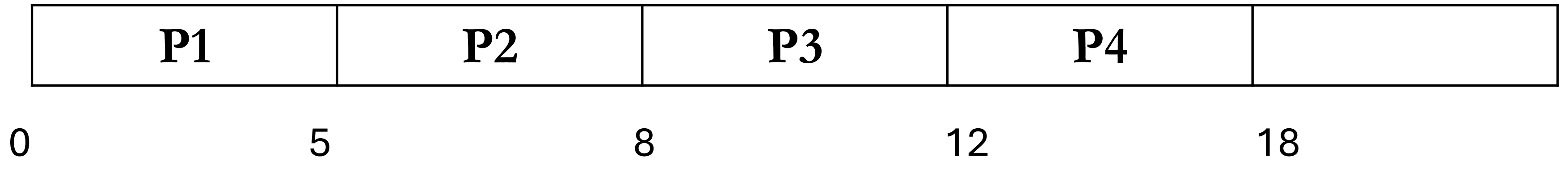


Scheduling – Uniprocessor - FCFS

- ❑ First Come First Serve – Example

Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

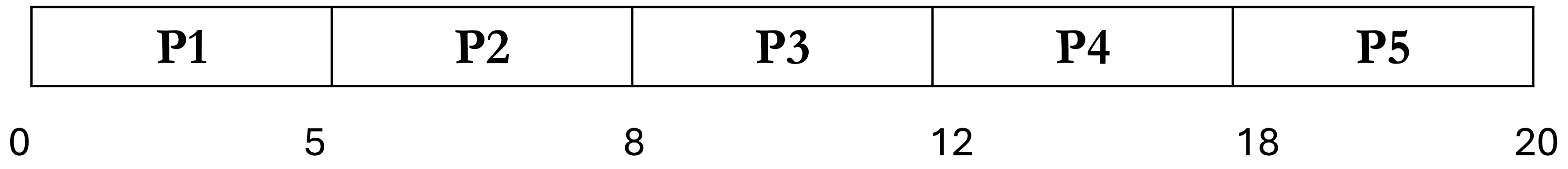


Scheduling – Uniprocessor - FCFS

❑ First Come First Serve – Example

Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2



Scheduling – Uniprocessor - FCFS

❑ First Come First Serve – Example

Calculating other Metrics

Process	Arrival Time	CPU Burst	Completion Time	Turnaround Time (Completion Time – Arrival Time)	Waiting Time (Turnaround Time – Burst Time)	Response Time time until first scheduled – Arrival (for non-preemptive, RT = WT).
P1	0	5	5	5 – 0 = 5	5 – 5 = 0	0 – 0 = 0
P2	2	3	8	8 – 2 = 6	6 – 3 = 3	5 – 2 = 3
P3	4	4	12	12 – 4 = 8	8 – 4 = 4	8 – 4 = 4
P4	6	6	18	18 – 6 = 12	12 – 6 = 6	12 – 6 = 6
P5	8	2	20	20 – 8 = 12	12 – 2 = 10	18 – 8 = 10
Average				8.6	4.6	

Scheduling – Uniprocessor - SJF

- **Shortest Job First**
  - Shortest Job First (SJF) is a CPU scheduling algorithm that always **selects the process with the smallest next CPU burst to execute next**.
  - By **prioritizing shorter jobs**, SJF minimizes the average waiting time in the ready queue.
    - **Non-Preemptive SJF**
      - Once the CPU is assigned to a process, it runs to completion of its current CPU burst.
      - New arrivals—even if shorter—must wait.

Scheduling – Uniprocessor - SJF

- ❑ Shortest Job First
  - ❑ Preemptive SJF (a.k.a. Shortest Remaining Time First, SRTF)
    - ❑ If a **new process arrives whose remaining CPU burst is shorter** than the **remaining time of the running process**, the CPU is preempted and given to the new process.

Scheduling – Uniprocessor - SJF

❑ Shortest Job First (Non – Preemptive) – Example

Consider the system image at the current moment and suggest how SJF will work in this:

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

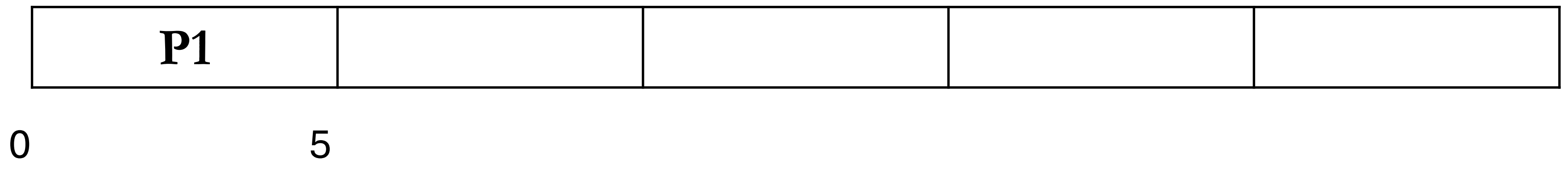
Scheduling – Uniprocessor - SJF

❑ SJF (Non-Preemptive)– Example

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Constructing Gantt Chart

At time 0 only P1 arrives → CPU allotted to P1



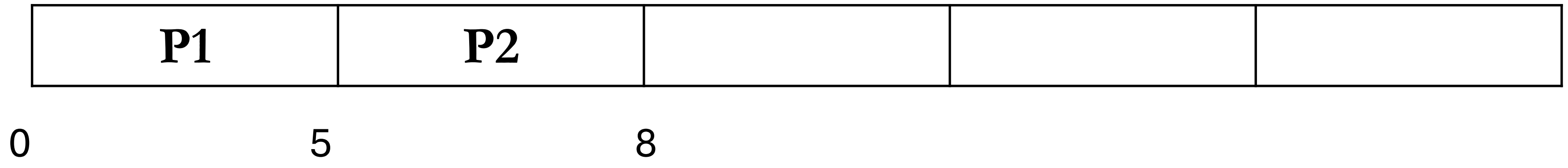
Scheduling – Uniprocessor - SJF

❑ SJF (Non-Preemptive)– Example

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Constructing Gantt Chart

P1 will run till time 5, In between P2 & P3 arrives. P2 has the shortest CPU burst time hence → CPU allotted to P2



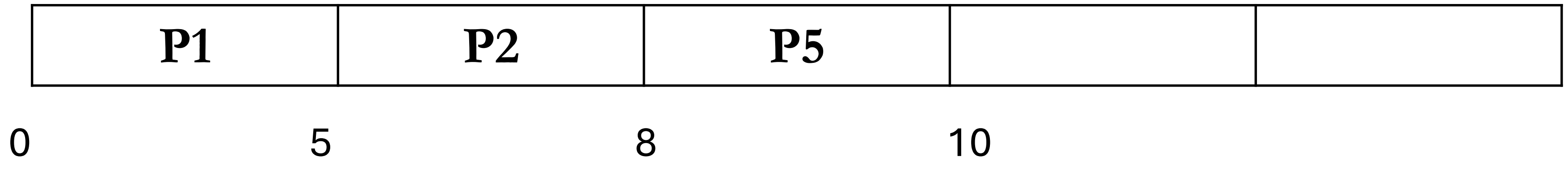
Scheduling – Uniprocessor - SJF

❑ SJF (Non-Preemptive)– Example

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Constructing Gantt Chart

P2 will run till time 8, In between P4 & P5 arrives and already P3 is waiting. Out of P3, P4 & P5, P5 has shortest CPU Burst time, Hence P5 gets CPU



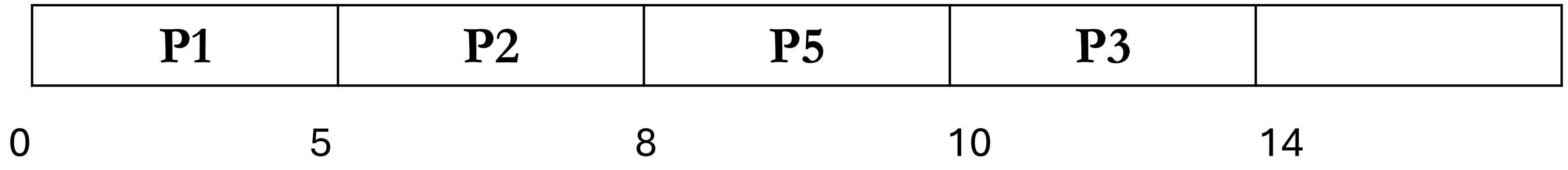
Scheduling – Uniprocessor - SJF

❑ **SJF (Non-Preemptive)– Example**

Constructing Gantt Chart

P5 will run till time 10, P3 & P4 are waiting. Out of both P3 has the shortest CPU Burst Time hence P3 → CPU allocated

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2



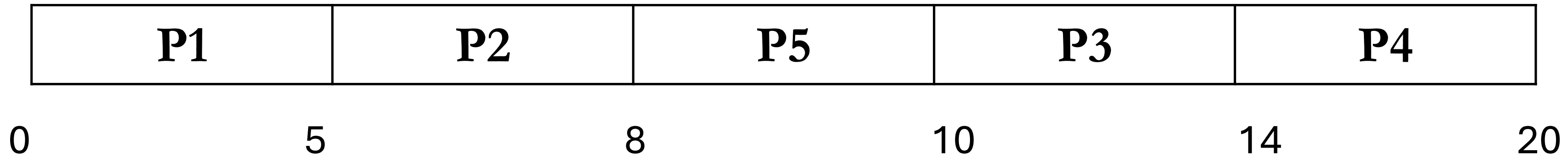
Scheduling – Uniprocessor - SJF

❑ SJF (Non-Preemptive)– Example

Constructing Gantt Chart

P3 will run till time 14, After P3 completes only P4 is left hence CPU allocated to it.

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2



Scheduling – Uniprocessor - SJF

❑ Shortest Job First (Non – Preemptive) – Example

Calculating other Metrics

Process	Arrival Time	CPU Burst	Completion Time	Turnaround Time (Completion Time – Arrival Time)	Waiting Time (Turnaround Time – Burst Time)	Response Time time until first scheduled – Arrival (for non-preemptive, RT = WT).
P1	0	5	5	5 – 0 = 5	5 – 5 = 0	0 – 0 = 0
P2	2	3	8	8 – 2 = 6	6 – 3 = 3	5 – 2 = 3
P5	8	2	10	10 – 8 = 2	2 – 2 = 0	8 – 8 = 0
P3	4	4	14	14 – 4 = 10	10 – 4 = 6	10 – 4 = 6
P4	6	6	20	20 – 6 = 14	14 – 6 = 8	14 – 6 = 8
Average				**7.4**	**3.4**	

Scheduling – Uniprocessor - SRTF

❑ Shortest Job First (Preemptive) – Example

Consider the system image at the current moment and suggest how SJF will work in this:

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

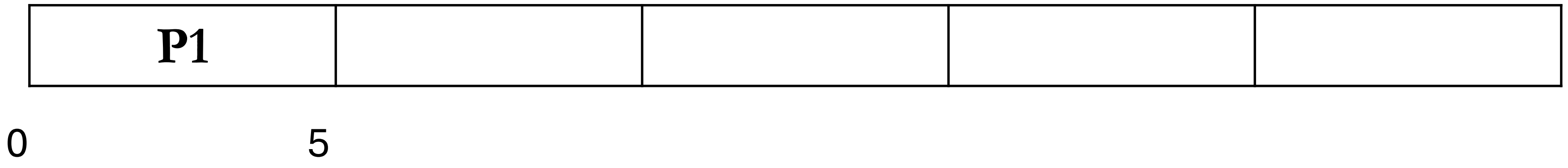
Scheduling – Uniprocessor - SRTF

❑ Shortest Job First (Preemptive)

Constructing Gantt Chart

At time 0 only P1 arrives → CPU allotted to P1
At time 2 P1 remains with 3 Bursts and P2 arrives with 3 BT so let P1 Continue and complete.

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2



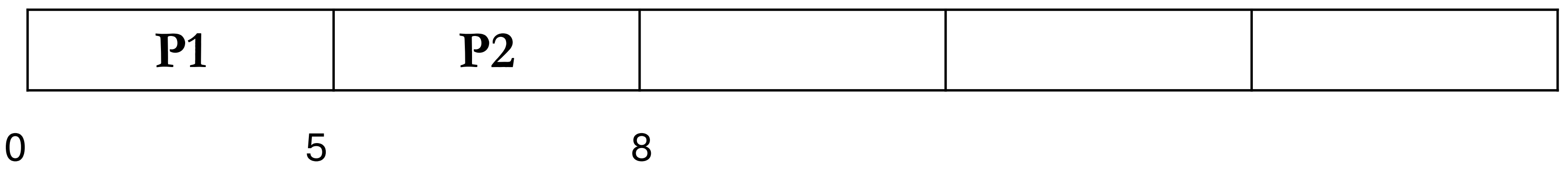
Scheduling – Uniprocessor - SRTF

❑ Shortest Job First (Preemptive)

Constructing Gantt Chart

At time 4 P3 arrives with 4 BT but P2 is waiting with 3 BT hence P2 is allocated to CPU

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2



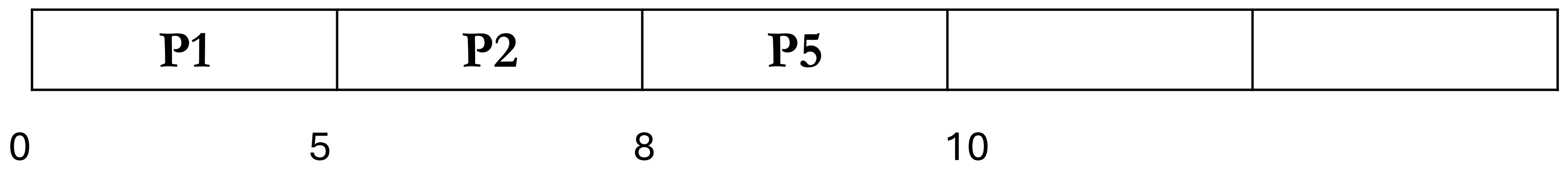
Scheduling – Uniprocessor - SRTF

❑ Shortest Job First (Preemptive)

Constructing Gantt Chart

At time 8 P4 & P5 arrives with 6 & 2 BT respectively, but P3 is also waiting with 4 BT, Shortest BT is P5 hence it is allocated.

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2



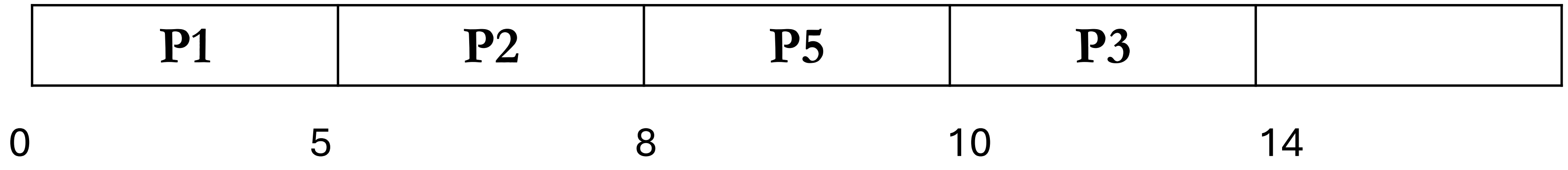
Scheduling – Uniprocessor - SRTF

❑ Shortest Job First (Preemptive)

Constructing Gantt Chart

Out of remaining process P3 has shortest BT hence allocated

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2



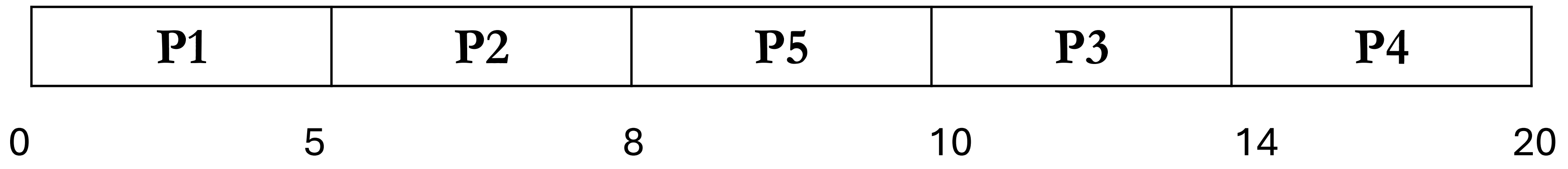
Scheduling – Uniprocessor - SRTF

❑ Shortest Job First (Preemptive)

Constructing Gantt Chart

P3 will run till time 14, After P3 completes only P4 is left hence CPU allocated to it.

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2



Scheduling – Uniprocessor - SRTF

❑ Shortest Job First (Preemptive) – Example

Calculating other Metrics

Process	Arrival Time	CPU Burst	Completion Time	Turnaround Time (Completion Time – Arrival Time)	Waiting Time (Turnaround Time – Burst Time)	Response Time time until first scheduled – Arrival (for non-preemptive, RT = WT).
P1	0	5	5	5 – 0 = 5	5 – 5 = 0	0 – 0 = 0
P2	2	3	8	8 – 2 = 6	6 – 3 = 3	5 – 2 = 3
P5	8	2	10	10 – 8 = 2	2 – 2 = 0	8 – 8 = 0
P3	4	4	14	14 – 4 = 10	10 – 4 = 6	10 – 4 = 6
P4	6	6	20	20 – 6 = 14	14 – 6 = 8	14 – 6 = 8
Average				**7.4**	**3.4**	

Scheduling – Uniprocessor – Priority

- Priority Scheduling
  - Each ready process is assigned an explicit priority value
  - The scheduler always dispatches the highest-priority ready process next
  - **Pre-emptive priority** – a newly-arriving job with higher priority immediately pre-empts the running job
  - **Non-pre-emptive priority** – the running job keeps the CPU until it blocks or finishes; new jobs wait even if they have higher priority
  - If several jobs share the same priority, the scheduler falls back to FCFS
  - Very low-priority jobs may wait forever while higher-priority jobs keep arriving; ageing (gradually boosting a job's priority the longer it waits) is a common fix.

Scheduling – Uniprocessor – Priority

❑ Priority Scheduling – (Preemptive) Example

Consider the system image at the current moment and suggest how Priority scheduling will work if the lower number are considered as high priority in this:

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

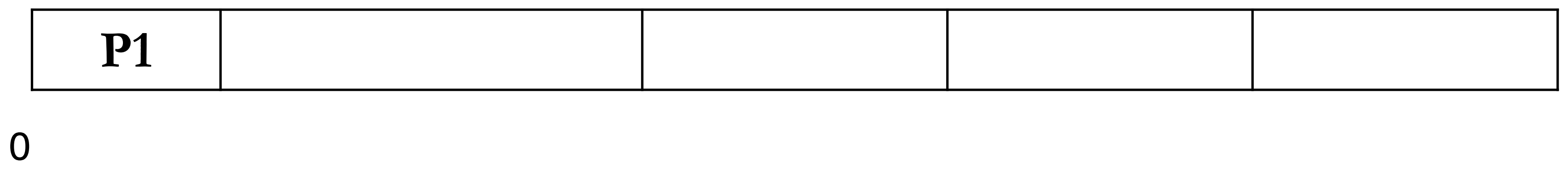
Scheduling – Uniprocessor – Priority

- Priority Scheduling – (Preemptive)

Constructing Gantt Chart

At time 0 only P1 arrives → CPU allotted to P1

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2



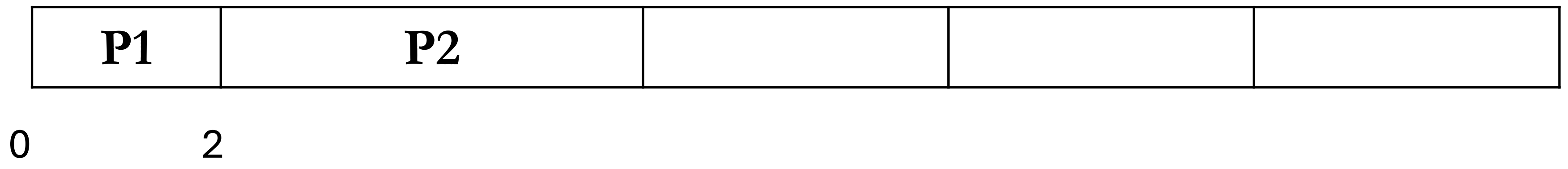
Scheduling – Uniprocessor – Priority

❑ **Priority Scheduling – (Preemptive)**

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

Constructing Gantt Chart

At time 2 → P1 Priority < P2 Priority, P1 preempted and 3 CPU BT pending for P1



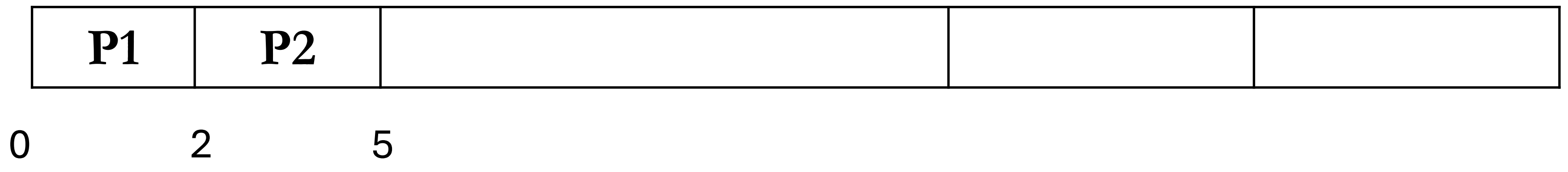
Scheduling – Uniprocessor – Priority

❑ Priority Scheduling – (Preemptive)

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

Constructing Gantt Chart

At time 4, P3 priority < P2 & P1 priority, hence allow P2 to complete the CPU BT till 5.



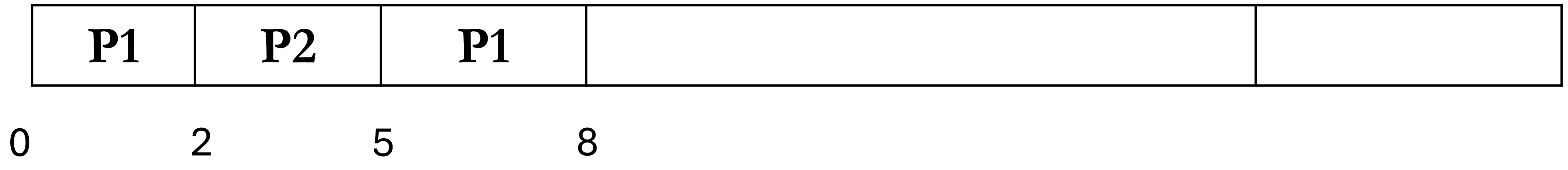
Scheduling – Uniprocessor – Priority

❑ Priority Scheduling – (Preemptive)

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

Constructing Gantt Chart

P2 completes. Since P1 has higher Priority, it takes CPU and completes its remaining BT.
At time 6, P4 Priority < P3 & P1 hence P1 continues till time 8.



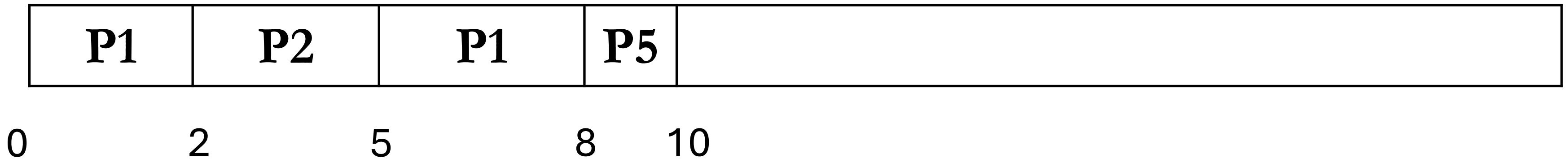
Scheduling – Uniprocessor – Priority

- **Priority Scheduling – (Preemptive)**

Constructing Gantt Chart

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

P1 completes. At time 8, P5 Priority > P3 & P4, Hence P5 will take CPU



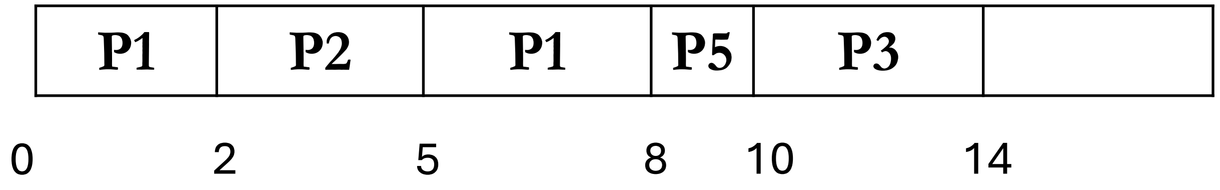
Scheduling – Uniprocessor - SRTF

❑ **Priority Scheduling – (Preemptive)**

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

Constructing Gantt Chart

P5 Completes. P3 Priority > P4 Priority. Hence P3 runs



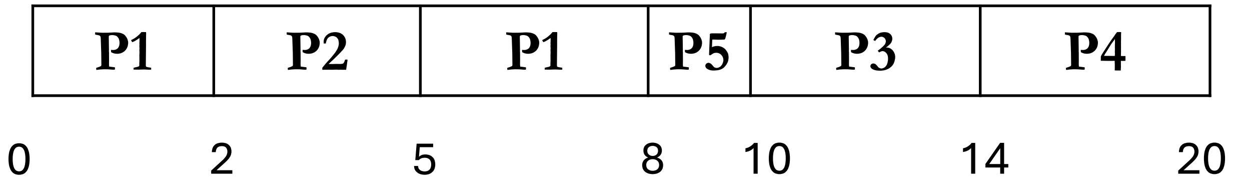
Scheduling – Uniprocessor – Priority

❑ Priority Scheduling – (Preemptive)

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

Constructing Gantt Chart

P3 Completes. P4 runs



Scheduling – Uniprocessor – Priority

❑ Priority Scheduling – (Preemptive) – Example

Calculating other Metrics

Process	Arrival Time	CPU Burst	Completion Time	Turnaround Time (Completion Time – Arrival Time)	Waiting Time (Turnaround Time – Burst Time)	Response Time time until first scheduled – Arrival (for non-preemptive, RT = WT).
P1	0	5	8	8 – 0 = 8	8 – 5 = 3	0 – 0 = 0
P2	2	3	5	5 – 2 = 3	3 – 3 = 0	2 – 2 = 0
P5	8	2	10	10 – 8 = 2	2 – 2 = 0	8 – 8 = 0
P3	4	4	14	14 – 4 = 10	10 – 4 = 6	10 – 4 = 6
P4	6	6	20	20 – 6 = 14	14 – 6 = 8	14 – 6 = 8
Average				**7.4**	**3.4**	

Scheduling – Uniprocessor – Priority

❑ **Priority Scheduling – (Non-Preemptive) Example**

Consider the system image at the current moment and suggest how Priority scheduling will work if the lower number are considered as high priority in this:

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

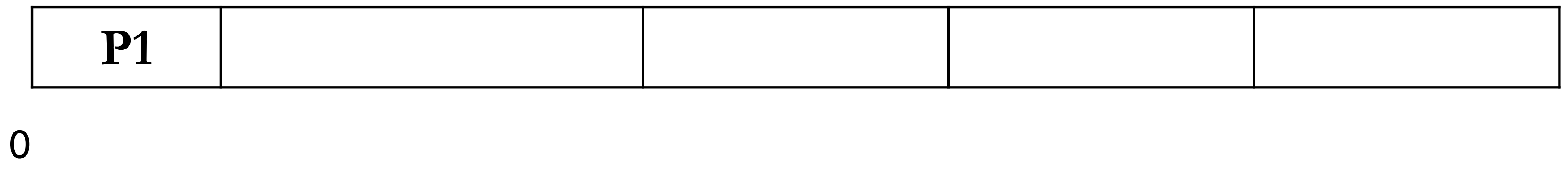
Scheduling – Uniprocessor – Priority

❑ Priority Scheduling – (Non-Preemp

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

Constructing Gantt Chart

At time 0 only P1 arrives → CPU allotted to P1



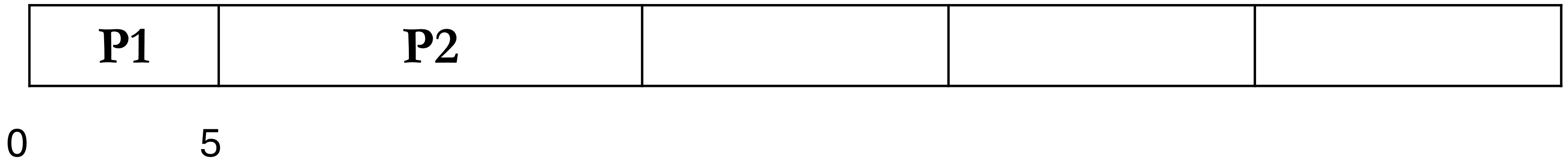
Scheduling – Uniprocessor – Priority

❑ Priority Scheduling – (Non-Preemp

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

Constructing Gantt Chart

At time 2 → P1 Priority < P2 Priority, P1 cannot be preempted and P1 continues



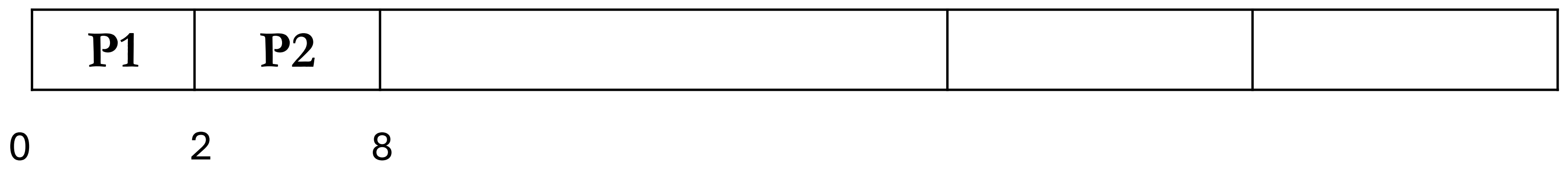
Scheduling – Uniprocessor – Priority

❑ Priority Scheduling – (Non-Preemp

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

Constructing Gantt Chart

At time 4, is P3 priority < P2 & P1 priority, hence allow P2 to complete the CPU BT till 8.

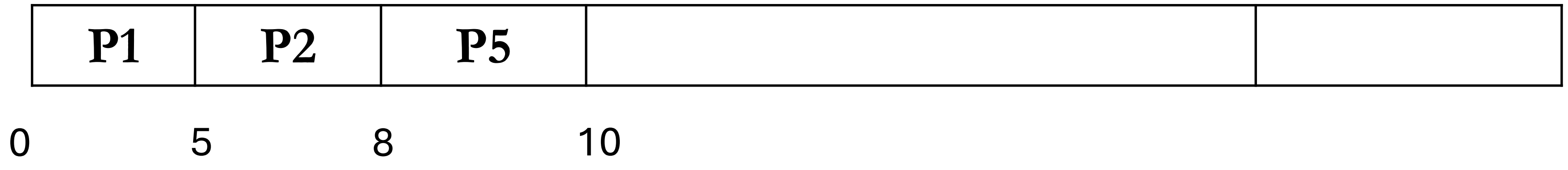


Scheduling – Uniprocessor – Priority

❑ Priority Scheduling – (Non-Preemp

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

Constructing Gantt Chart

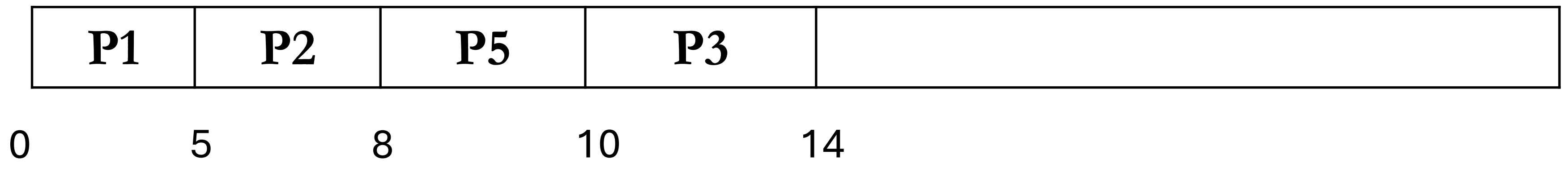


Scheduling – Uniprocessor – Priority

❑ Priority Scheduling – (Non-Preemp

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

Constructing Gantt Chart

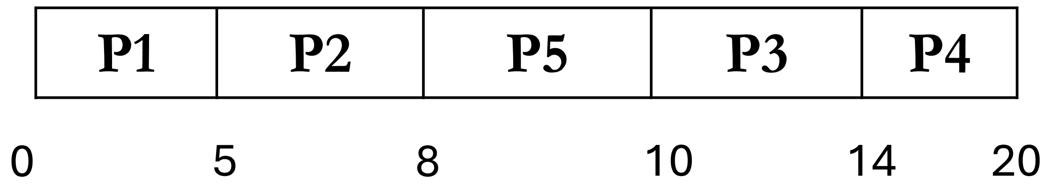


Scheduling – Uniprocessor - SRTF

❑ Priority Scheduling – (Non-Preemp

Constructing Gantt Chart

Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2



Scheduling – Uniprocessor – Priority

❑ Priority Scheduling – (Non-Preemptive) – Example

Calculating other Metrics

Process	Arrival Time	CPU Burst	Completion Time	Turnaround Time (Completion Time – Arrival Time)	Waiting Time (Turnaround Time – Burst Time)	Response Time time until first scheduled – Arrival (for non-preemptive, RT = WT).
P1	0	5	5	5 – 0 = 5	5 – 5 = 0	0 – 0 = 0
P2	2	3	8	8 – 2 = 6	6 – 3 = 3	5 – 2 = 3
P5	8	2	10	10 – 8 = 2	2 – 2 = 0	8 – 8 = 0
P3	4	4	14	14 – 4 = 10	10 – 4 = 6	10 – 4 = 6
P4	6	6	20	20 – 6 = 14	14 – 6 = 8	14 – 6 = 8
Average				**7.4**	**3.4**	

Scheduling – Uniprocessor – Round Robin

- ❑ It is a preemptive algorithm that gives each process a fixed amount of time, known as a **time quantum** or **time slice**, to **run on the CPU**.
- ❑ This method ensures that all processes get a fair share of the CPU's time.
- ❑ The Round Robin algorithm works by assigning each process a turn to execute in a cyclical fashion.
- ❑Note: The preempted process is added to the tail of the ready queue first, followed by the newly arrived process.

Scheduling – Uniprocessor – Round Robin

- ❑**Ready Queue:** All active processes are kept in a ready queue, which is typically managed in a First-In, First-Out (FIFO) order.
- ❑ **Time Quantum**: The system defines a fixed time quantum
- ❑ **Process Execution**: The scheduler selects the first process from the ready queue and allows it to run on the CPU for the duration of one-time quantum
- ❑ **Process Completes**: If the process finishes its execution before the time quantum expires, it voluntarily releases the CPU. The scheduler then immediately selects the next process in the queue.
- ❑ **Time Quantum Expires**: If the process is still running when the time quantum ends, the CPU is preempted (taken away) from the process. The process is then moved to the back of the ready queue to await its next turn
- ❑ **Cyclic Repetition**: The scheduler continues this cycle, picking the next process from the front of the queue, until all processes have completed their execution

Scheduling – Uniprocessor – Round Robin

- ❑ Consider the system image at the current moment and suggest how Round Robin scheduling will work if the given time quantum is 2 seconds.

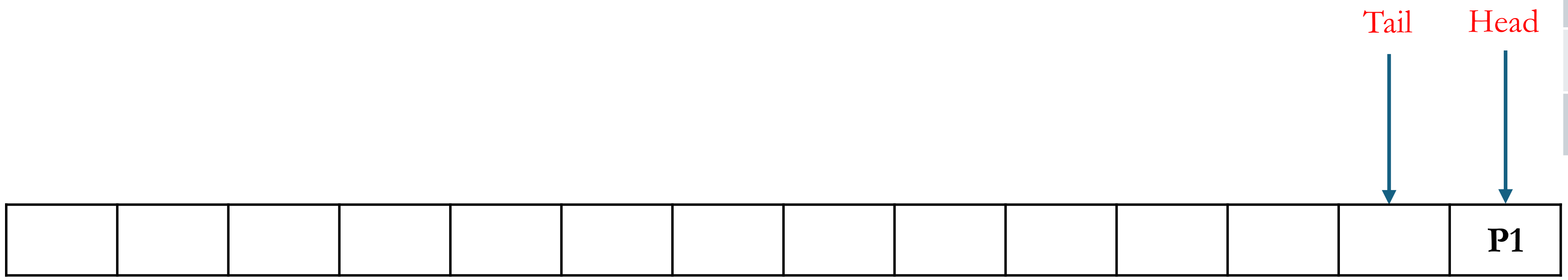
Process	Arrival Time	CPU Burst	Priority
P1	0	5	2
P2	2	3	1
P3	4	4	3
P4	6	6	4
P5	8	2	2

Scheduling – Uniprocessor – Round Robin

Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 0**



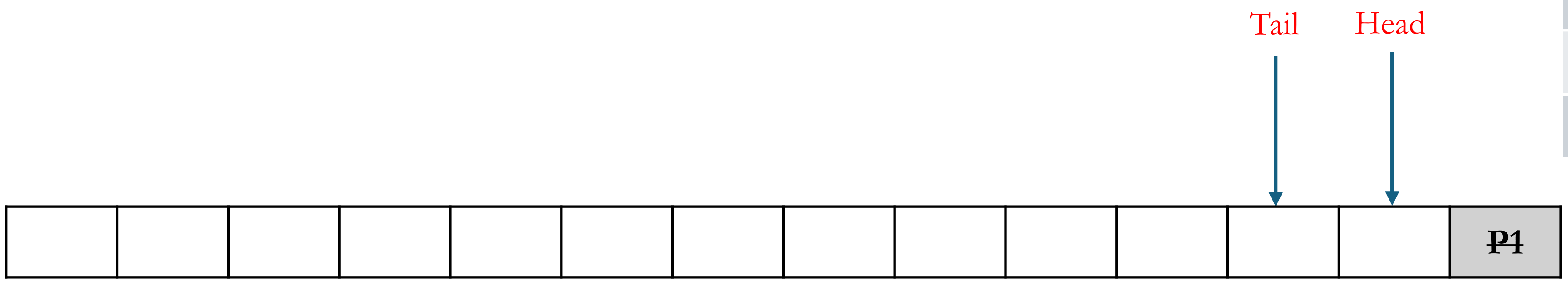
Gantt

Scheduling – Uniprocessor – Round Robin

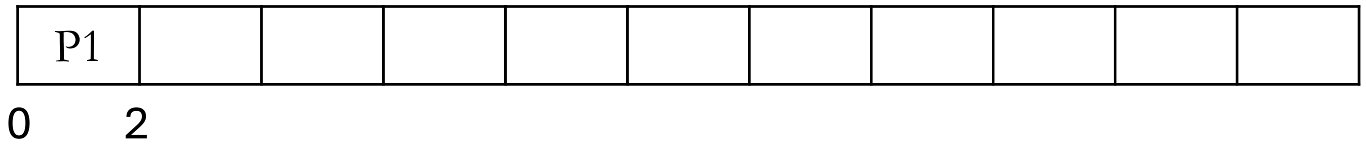
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 0**



Gantt

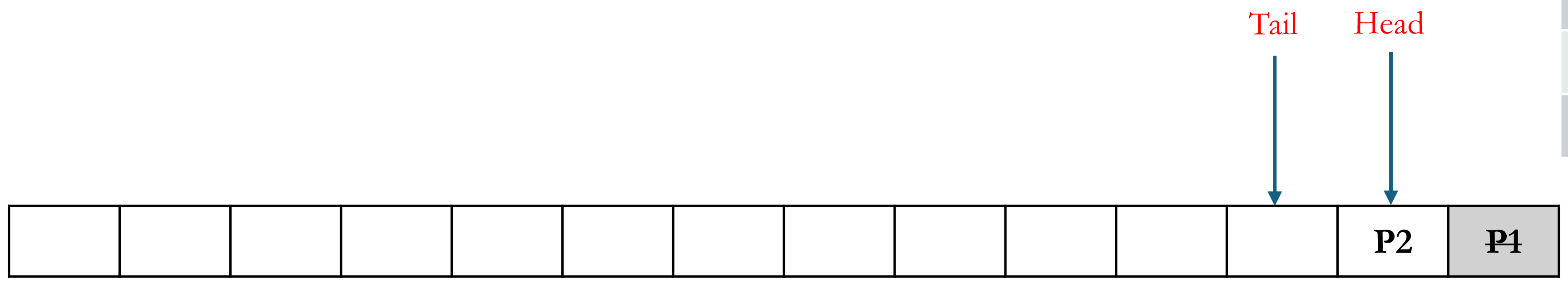


Scheduling – Uniprocessor – Round Robin

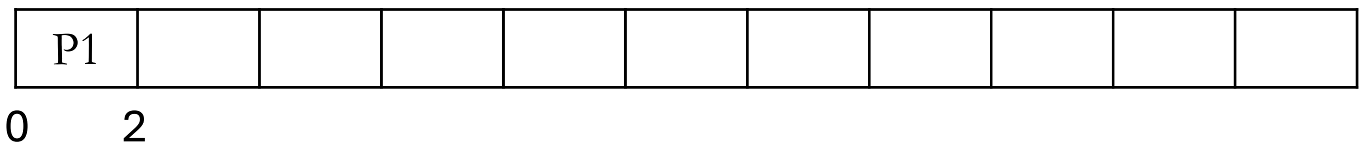
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 2**



Gantt

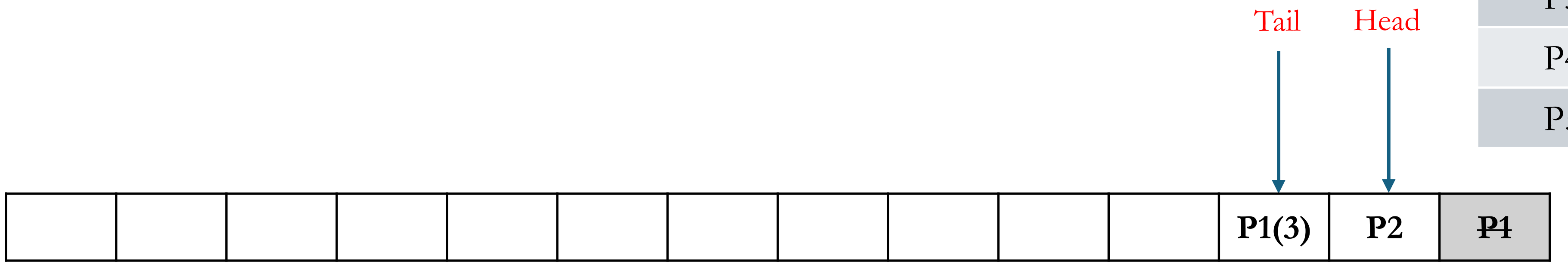


Scheduling – Uniprocessor – Round Robin

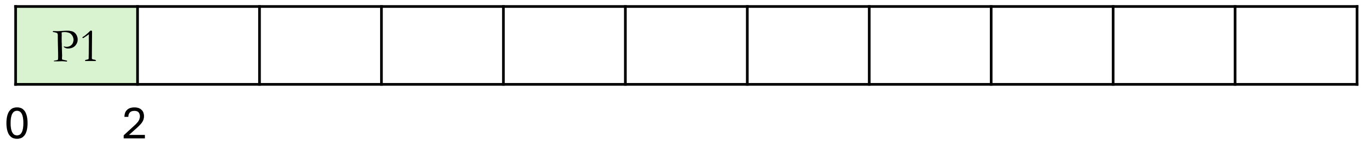
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 2**



Gantt

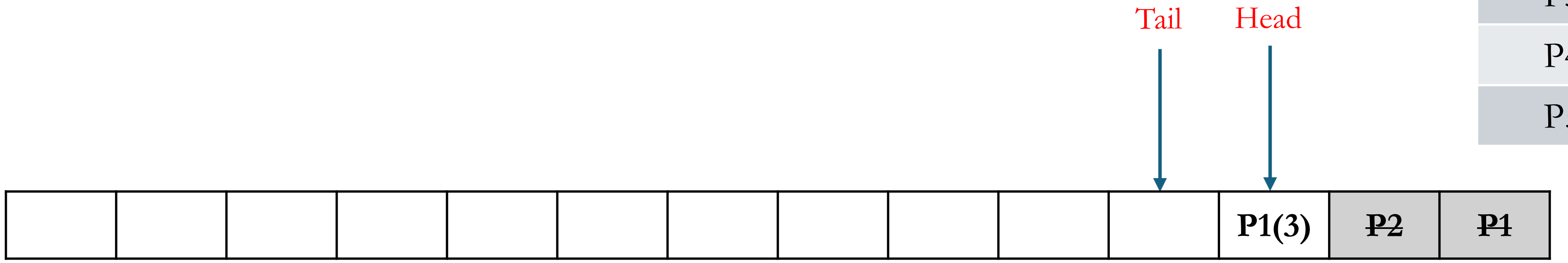


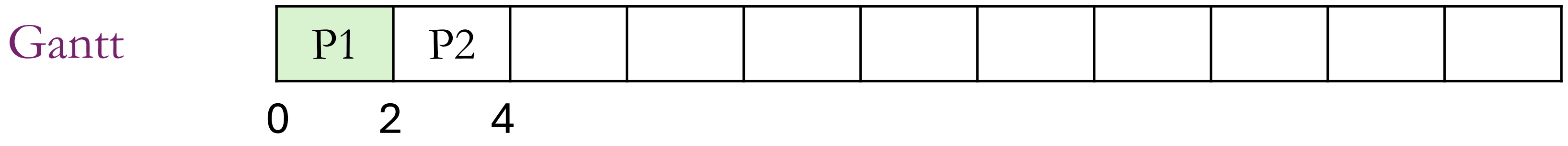
Scheduling – Uniprocessor – Round Robin

Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 2**



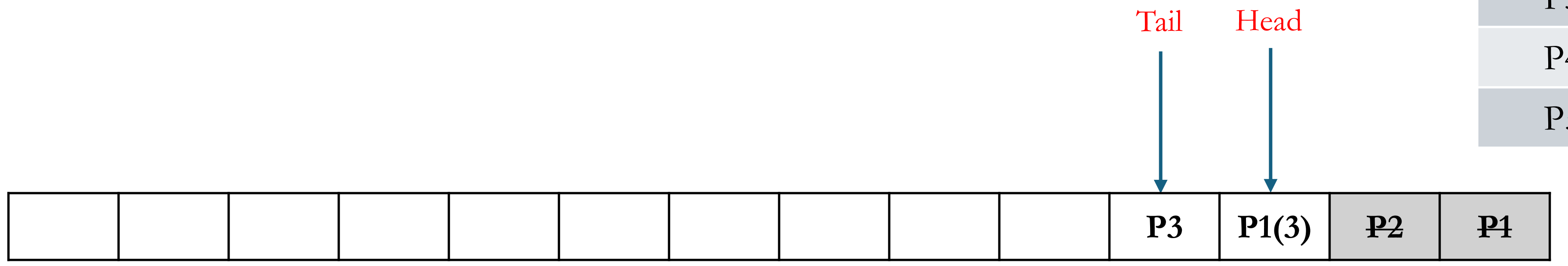


Scheduling – Uniprocessor – Round Robin

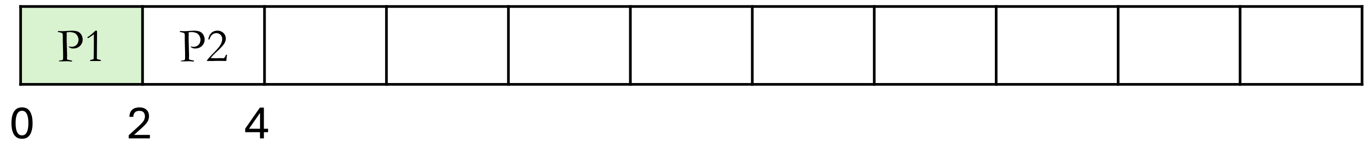
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 4**



Gantt

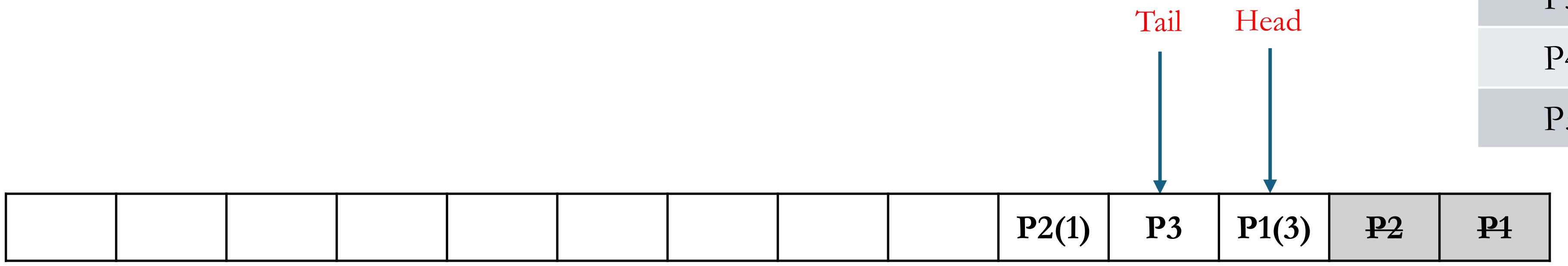


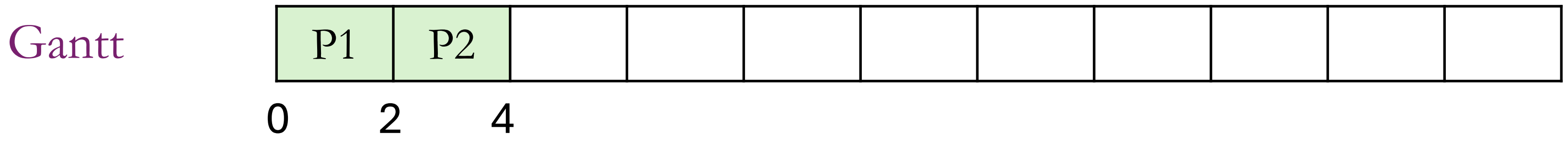
Scheduling – Uniprocessor – Round Robin

Constructing Gantt Chart

Ready Queue Image – **At Time 4**

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2



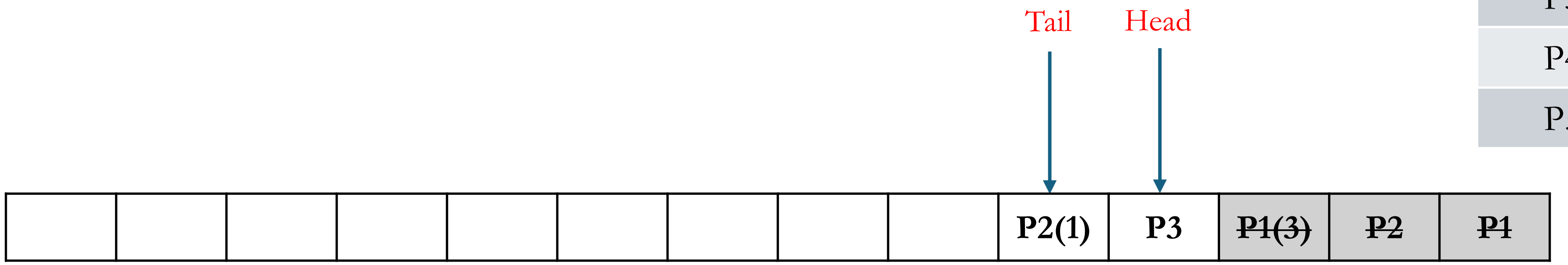


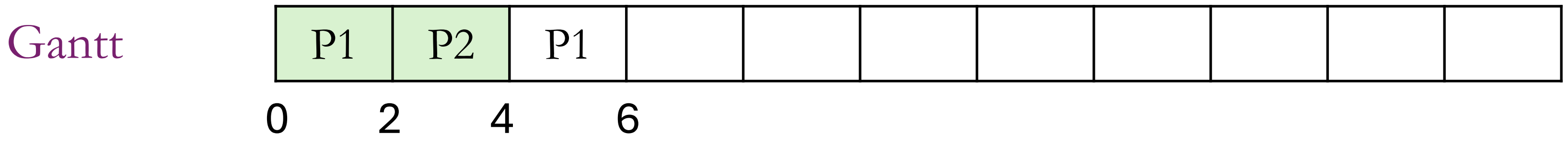
Scheduling – Uniprocessor – Round Robin

Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 4**



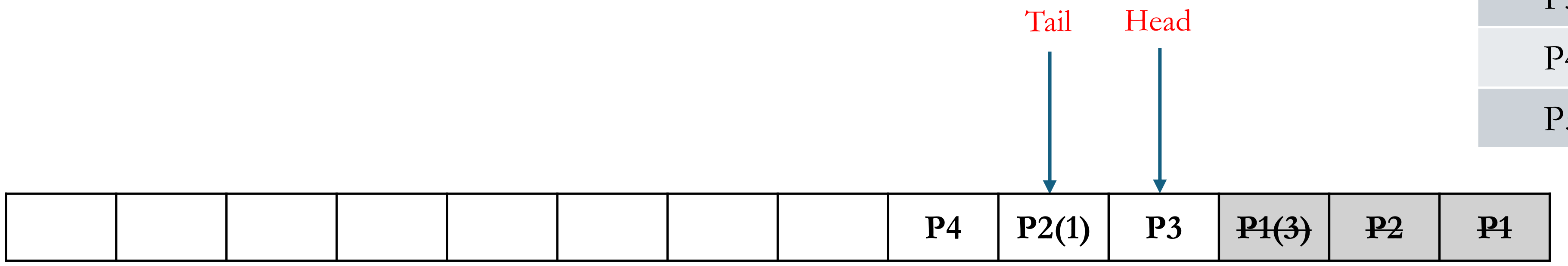


Scheduling – Uniprocessor – Round Robin

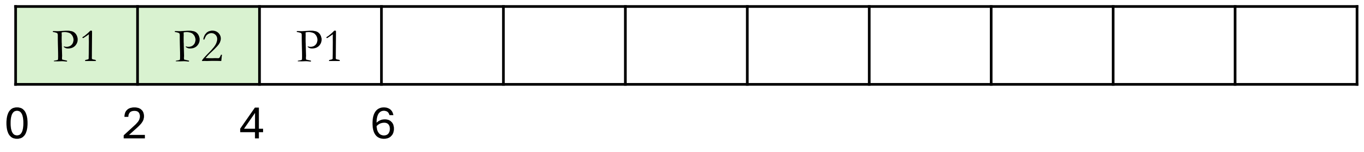
Constructing Gantt Chart

Ready Queue Image – **At Time 6**

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2



Gantt

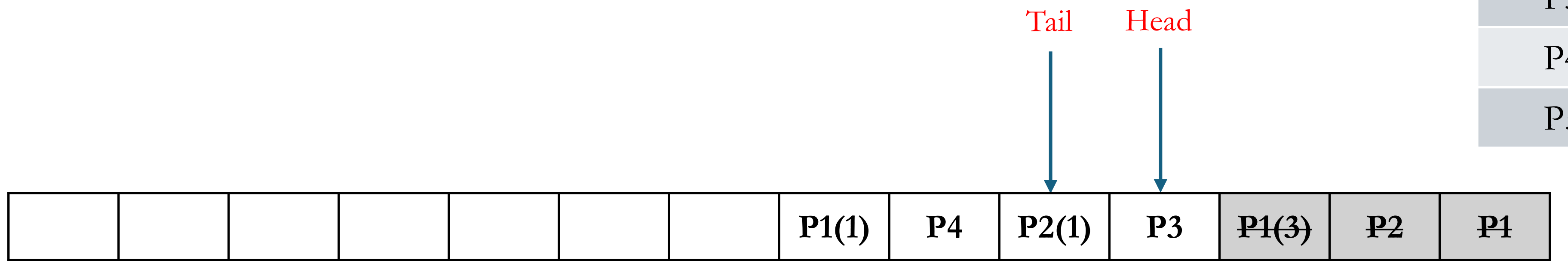


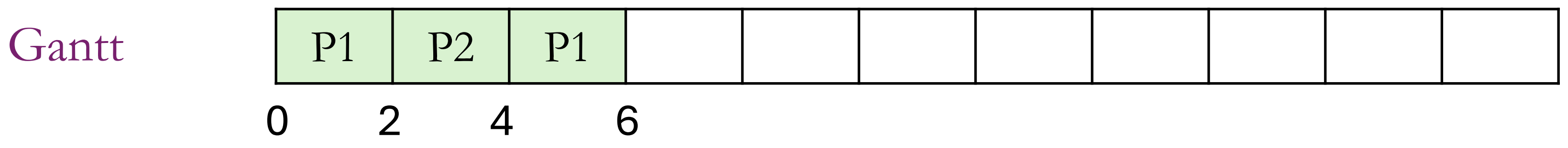
Scheduling – Uniprocessor – Round Robin

Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 6**



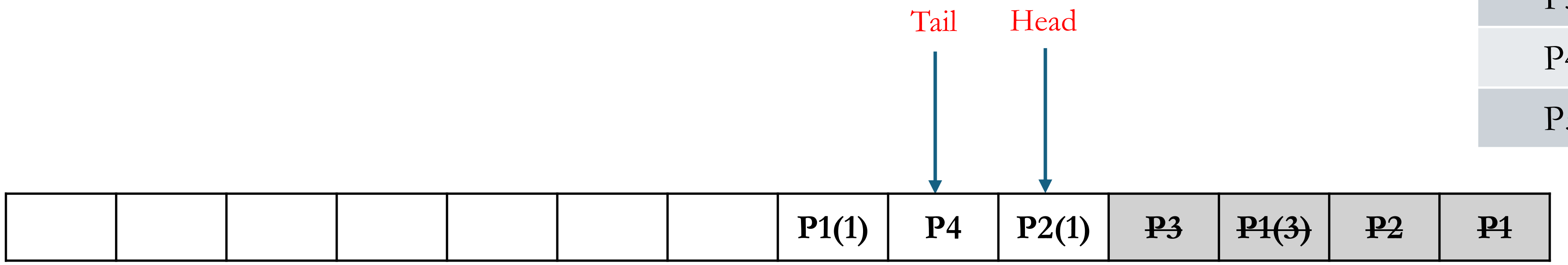


Scheduling – Uniprocessor – Round Robin

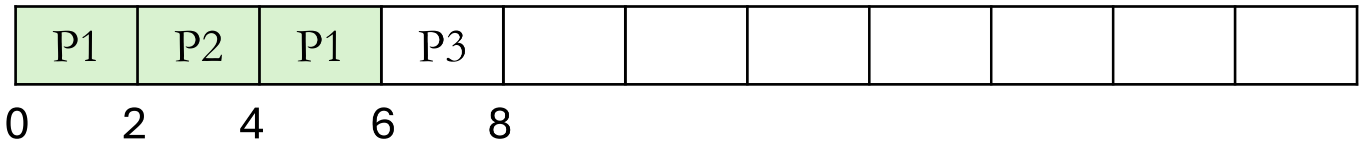
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 6**



Gantt

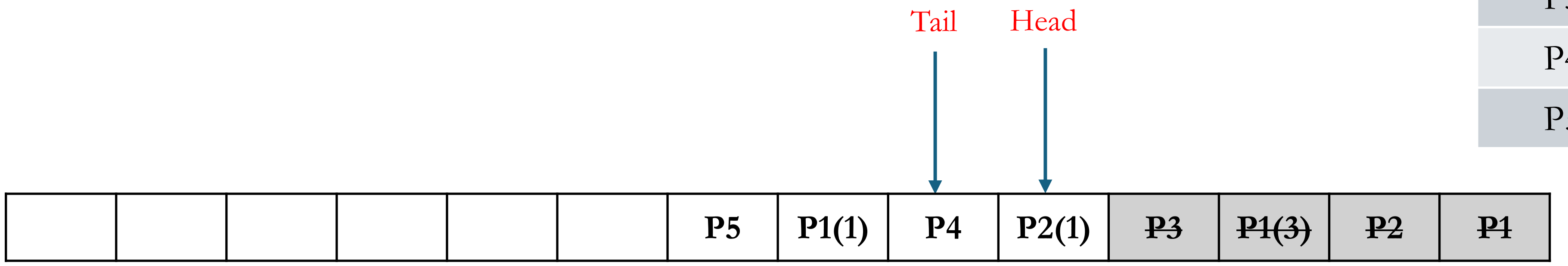


Scheduling – Uniprocessor – Round Robin

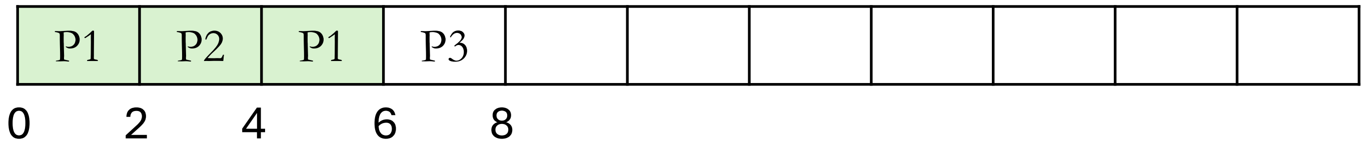
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 8**



Gantt

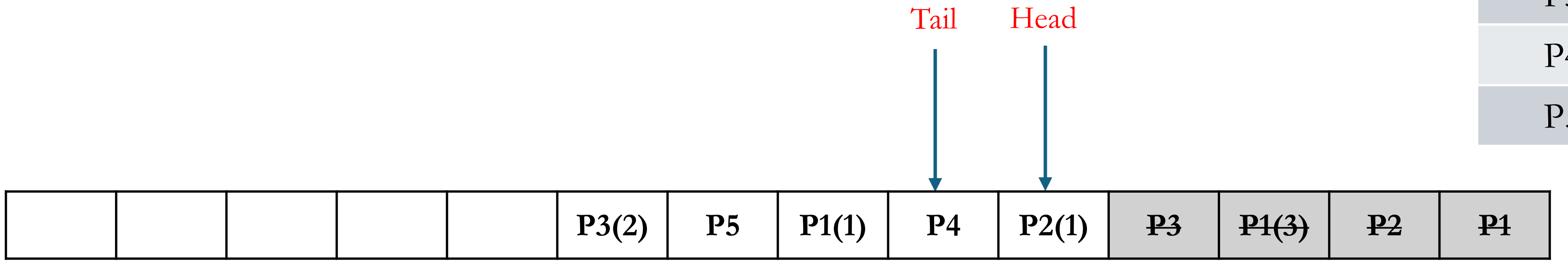


Scheduling – Uniprocessor – Round Robin

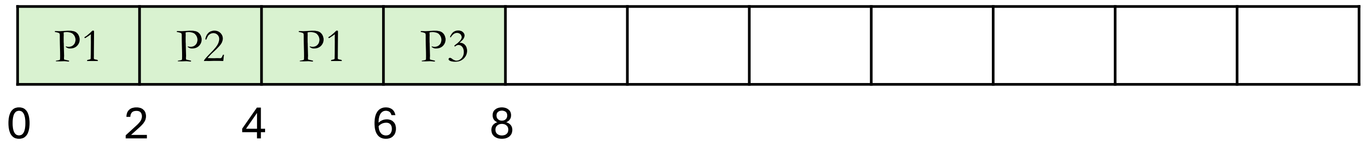
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 8**



Gantt

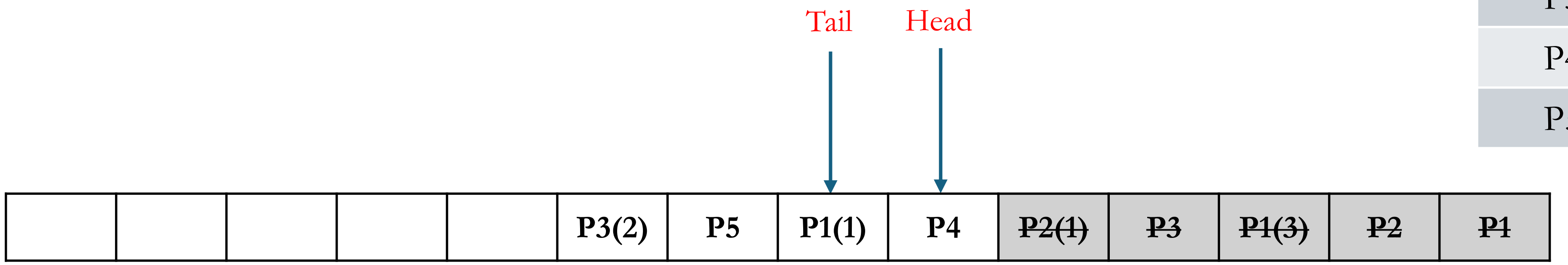


Scheduling – Uniprocessor – Round Robin

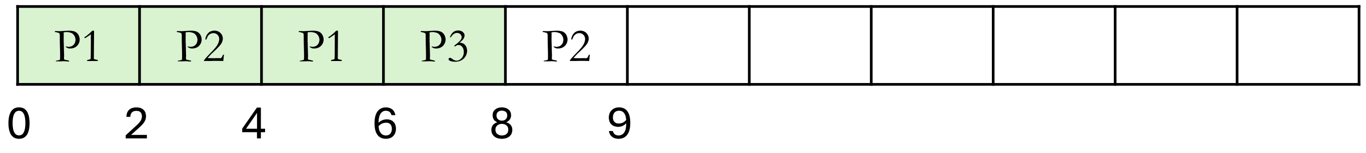
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 8**



Gantt

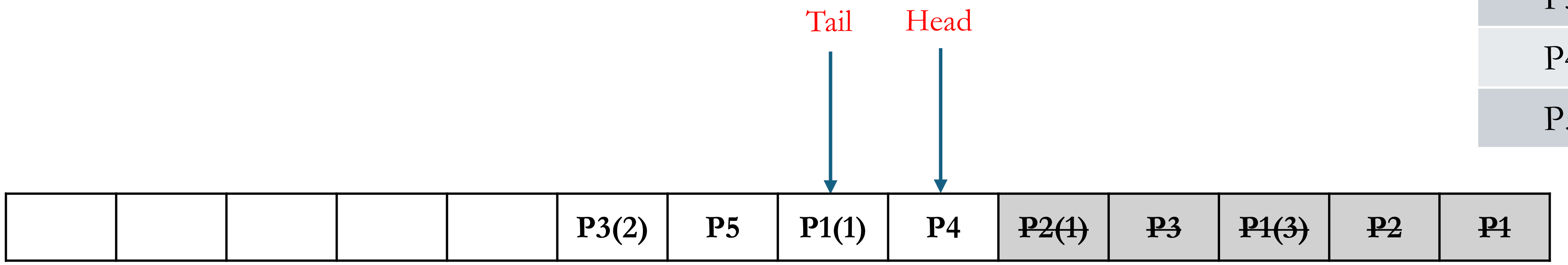


Scheduling – Uniprocessor – Round Robin

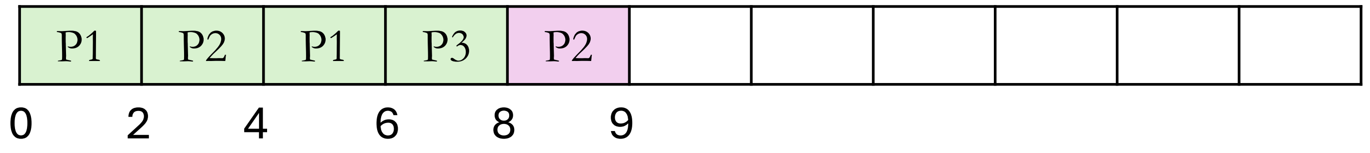
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 9**



Gantt

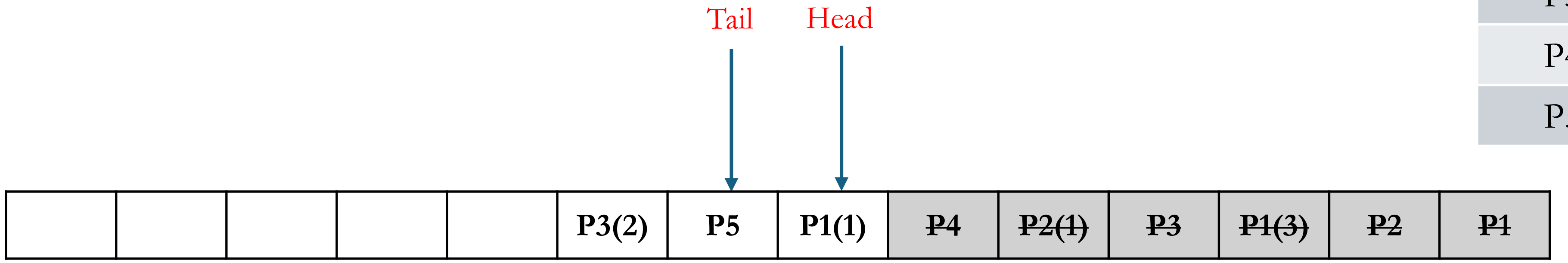


Scheduling – Uniprocessor – Round Robin

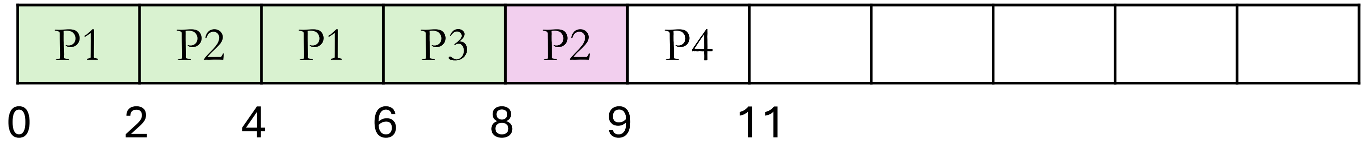
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 9**



Gantt

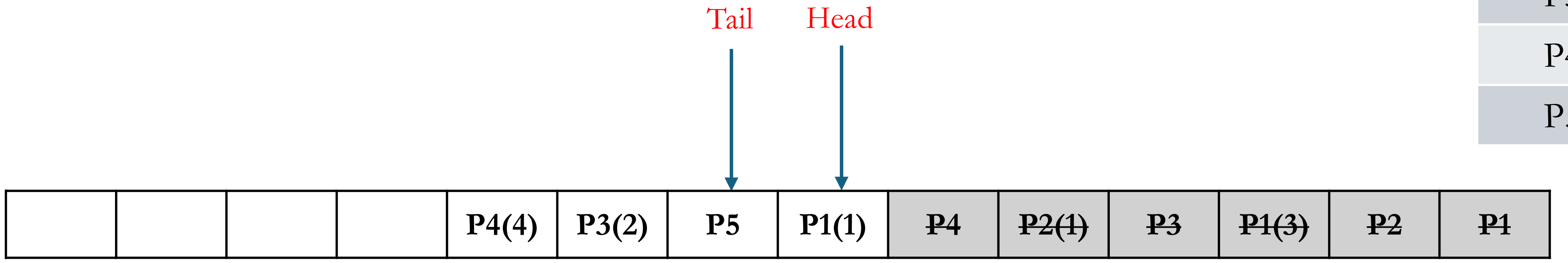


Scheduling – Uniprocessor – Round Robin

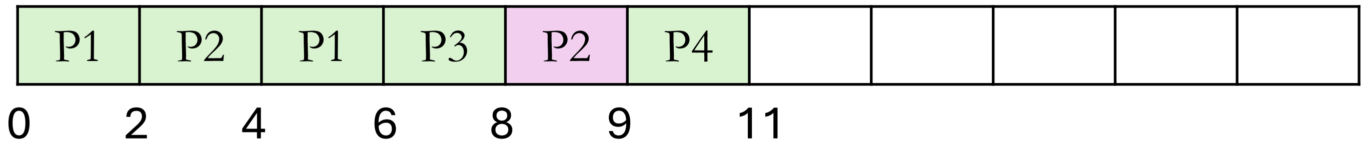
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 11**



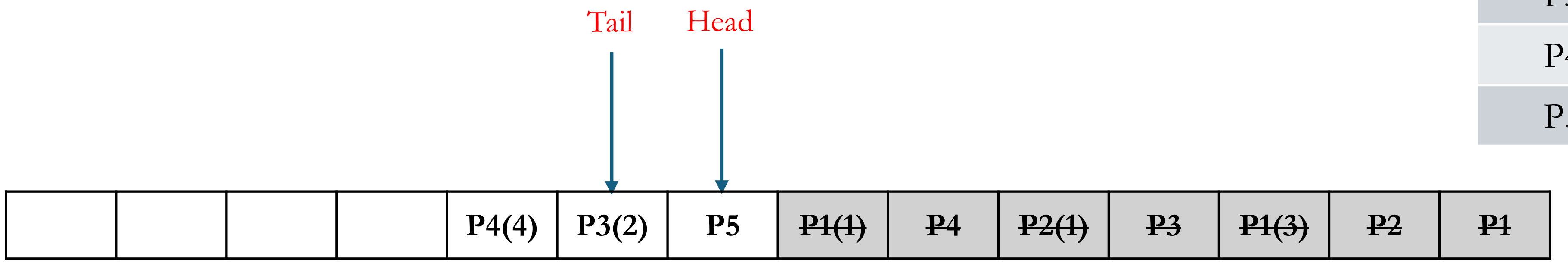
Gantt



Scheduling – Uniprocessor – Round Robin

Constructing Gantt Chart

Ready Queue Image – **At Time 11**



Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Gantt

P1	P2	P1	P3	P2	P4	P1				

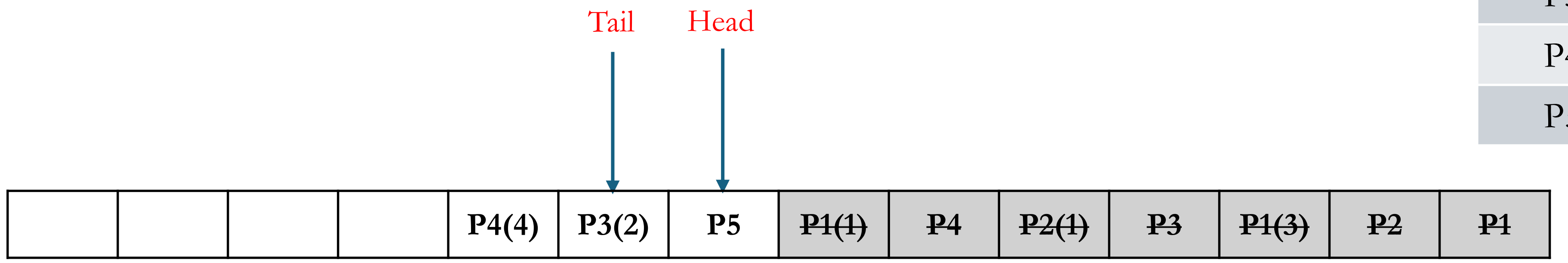
0 2 4 6 8 9 11 12

Scheduling – Uniprocessor – Round Robin

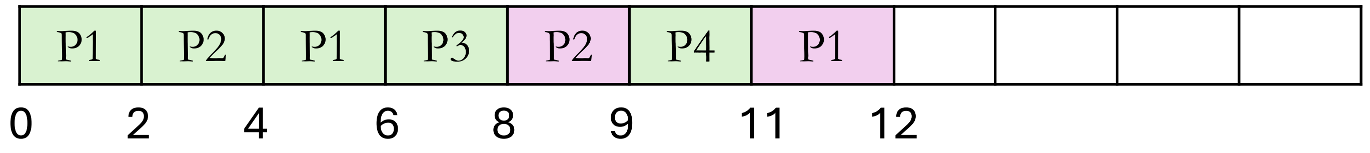
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 12**



Gantt

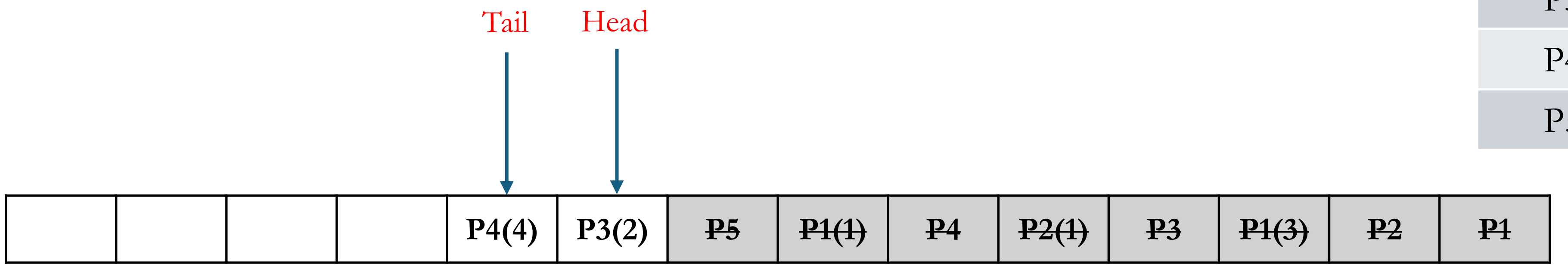


Scheduling – Uniprocessor – Round Robin

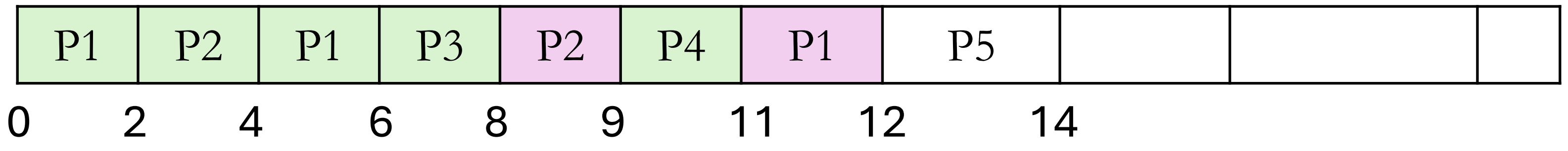
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 12**



Gantt

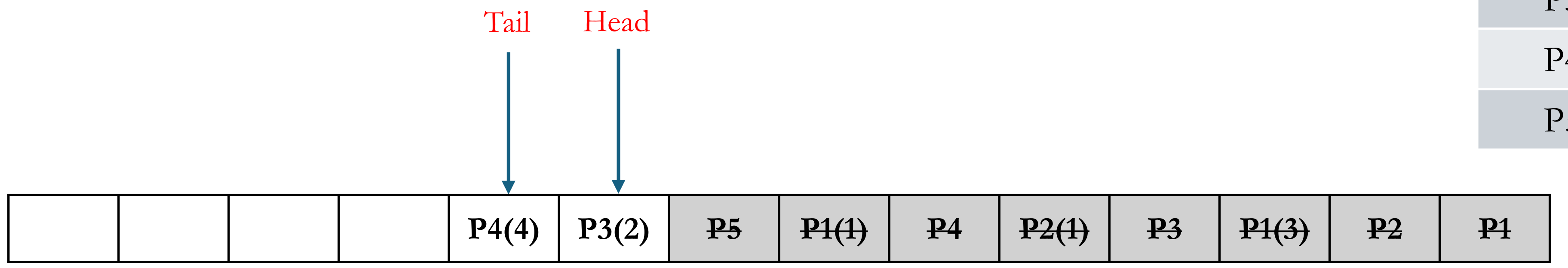


Scheduling – Uniprocessor – Round Robin

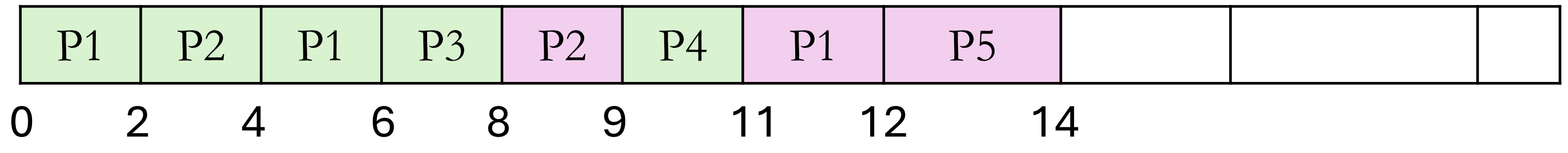
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 14**



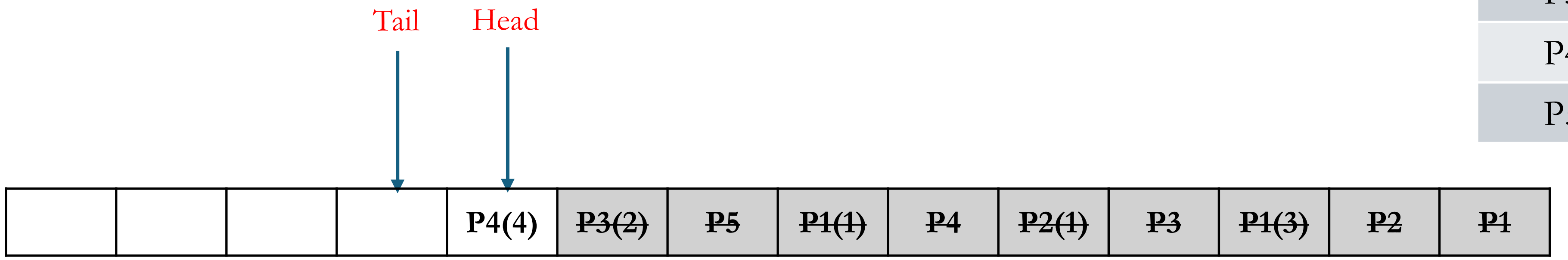
Gantt



Scheduling – Uniprocessor – Round Robin

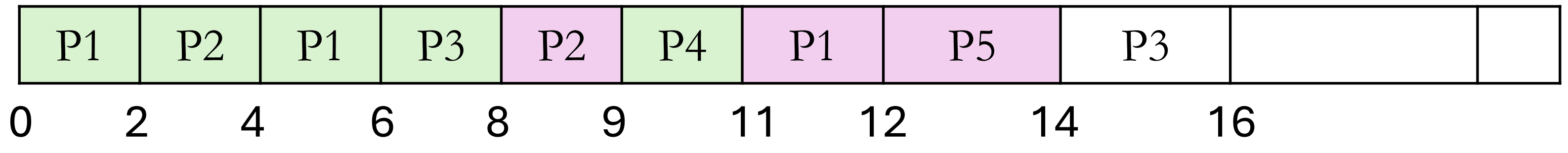
Constructing Gantt Chart

Ready Queue Image – **At Time 14**



Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Gantt

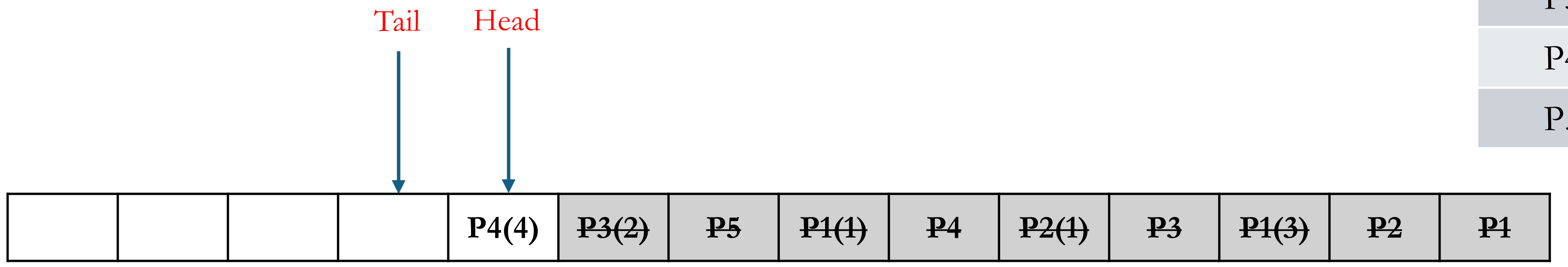


Scheduling – Uniprocessor – Round Robin

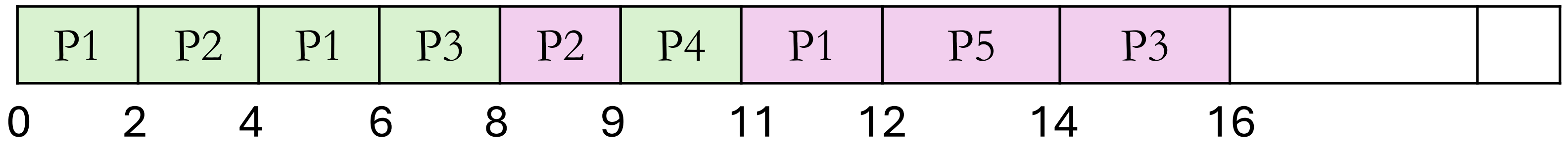
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 16**



Gantt

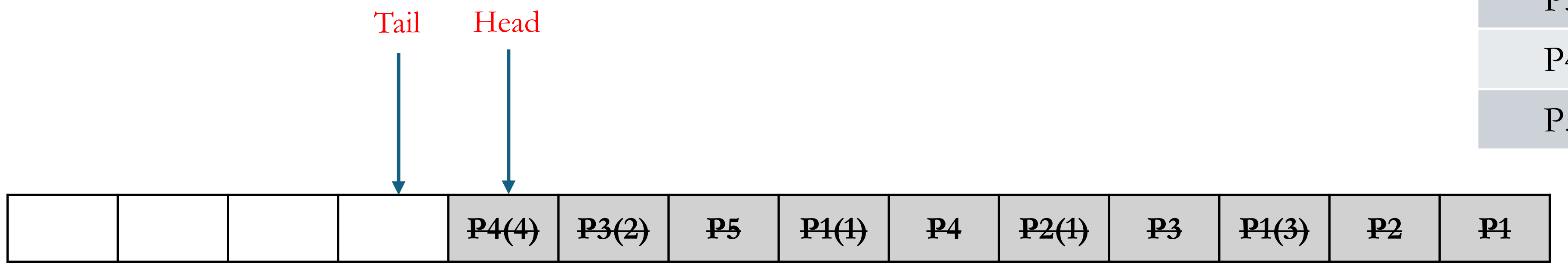


Scheduling – Uniprocessor – Round Robin

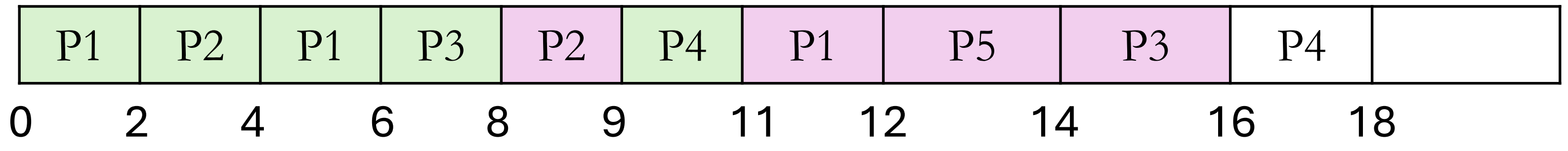
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 16**



Gantt

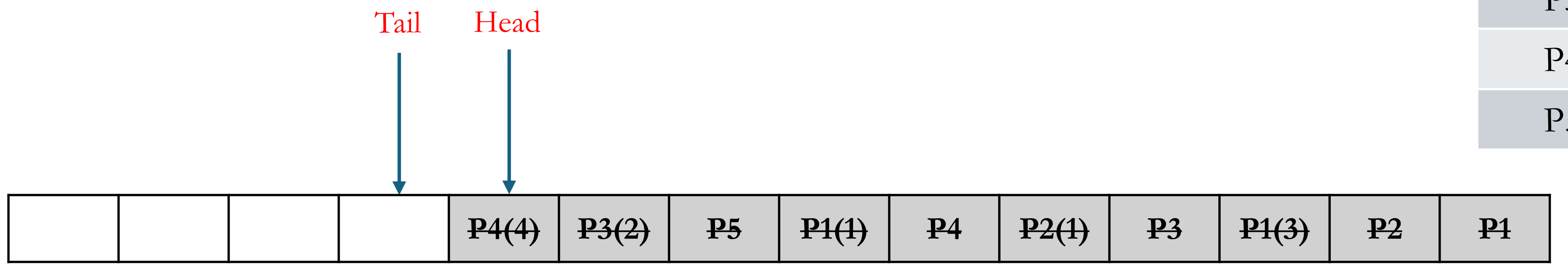


Scheduling – Uniprocessor – Round Robin

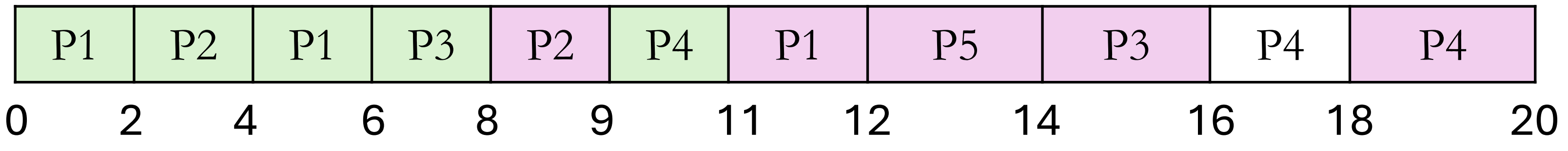
Constructing Gantt Chart

Process	Arrival Time	CPU Burst
P1	0	5
P2	2	3
P3	4	4
P4	6	6
P5	8	2

Ready Queue Image – **At Time 18**



Gantt



Scheduling – Uniprocessor – Round Robin

❑ Round Robin Scheduling – (Preemptive) – Example

Calculating other Metrics

Process	Arrival Time	CPU Burst	Completion Time	Turnaround Time (Completion Time – Arrival Time)	Waiting Time (Turnaround Time – Burst Time)	Response Time time until first scheduled – Arrival (for non-preemptive, RT = WT).
P1	0	5	12	12 – 0 = 12	12 – 5 = 7	0 – 0 = 0
P2	2	3	9	9 – 2 = 7	7 – 3 = 4	2 – 2 = 0
P3	4	4	16	16 – 4 = 12	12 – 4 = 8	6 – 4 = 2
P4	6	6	20	20 – 6 = 14	14 – 6 = 8	9 – 6 = 3
P5	8	2	14	14 – 8 = 6	6 – 2 = 4	12 – 8 = 4
Average				**10.2**	**6.2**	

Scheduling – Uniprocessor – Example

- ❑ Run the process with the highest priority. Processes with the same priority run round-robin with Time slice = 2.

Process	Arrival Time	Burst Time	Priority
p1	0	4	3
p2	0	5	2
p3	0	8	2
p4	0	7	1
p5	0	3	3

Scheduling – Uniprocessor – Example

- Run the process with the highest priority. Processes with the same priority run round-robin with Time slice = 2.

Process	Arrival Time	Burst Time	Priority
p1	0	4	3
p2	0	5	2
p3	0	8	2
p4	0	7	1
p5	0	3	3

p4	p2	p3	p2	p3	p2	p3	p1	p5	p1	p5

Scheduling – Uniprocessor – Example

- ❑ Run the process with the highest priority. Processes with the same priority run round-robin with Time slice = 2.

Process	Arrival Time	Burst Time	Priority	Completion Time	Turnaround Time (Completion Time – Arrival Time)	Waiting Time (Turnaround Time – Burst Time)
p1	0	4	3	26	26 – 0 = 26	26 – 4 = 22
p2	0	5	2	16	16 – 0 = 16	16 – 5 = 11
p3	0	8	2	20	20 – 0 = 20	20 – 8 = 12
p4	0	7	1	7	7 - 0 = 7	7 – 7 = 0
p5	0	3	3	27	27 - 0 = 27	27 – 3 = 24
Average Turnaround Time					19	
Average Waiting Time						13.8

Multi-Level Queue Scheduling

- ❑ Multilevel Queue Scheduling is like creating multiple, separate waiting lines for the CPU (the computer's processor) instead of just one.
- ❑ Processes are sorted into these different lines, or queues, based on their characteristics, such as priority, memory size, or process type (Interactive, Batch or User).
- ❑ Once a process is placed in a queue, it generally stays there

Multi-Level Queue Scheduling

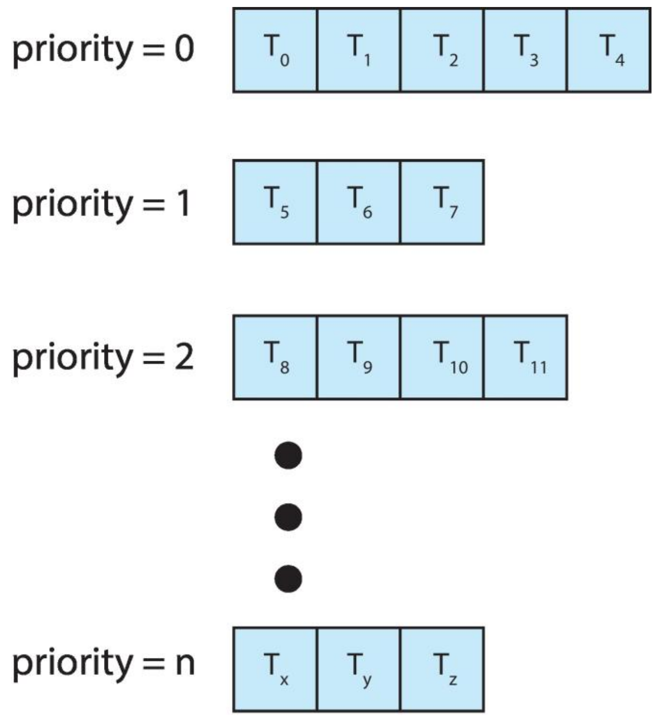
- ❑ **Multiple Queues**: The main ready queue is divided into several separate queues.
- ❑ **Process Sorting**: Processes are permanently assigned to a specific queue based on a property, like whether they are interactive (foreground) or batch (background) jobs.
- ❑ **Independent Scheduling**: Each queue can have its own unique scheduling algorithm. For example, a high-priority queue might use Round Robin (RR) scheduling to ensure responsiveness, while a low-priority queue might use First-Come, First-Served (FCFS).
- ❑ **Scheduling Between Queues**: **There is also a scheduling method for the queues themselves**. This is typically a fixed-priority preemptive scheduling system. **This means that no process in a lower-priority queue can run unless all the higher-priority queues are empty**. If a high-priority process arrives while a low-priority process is running, the **low-priority process is interrupted (preempted).**

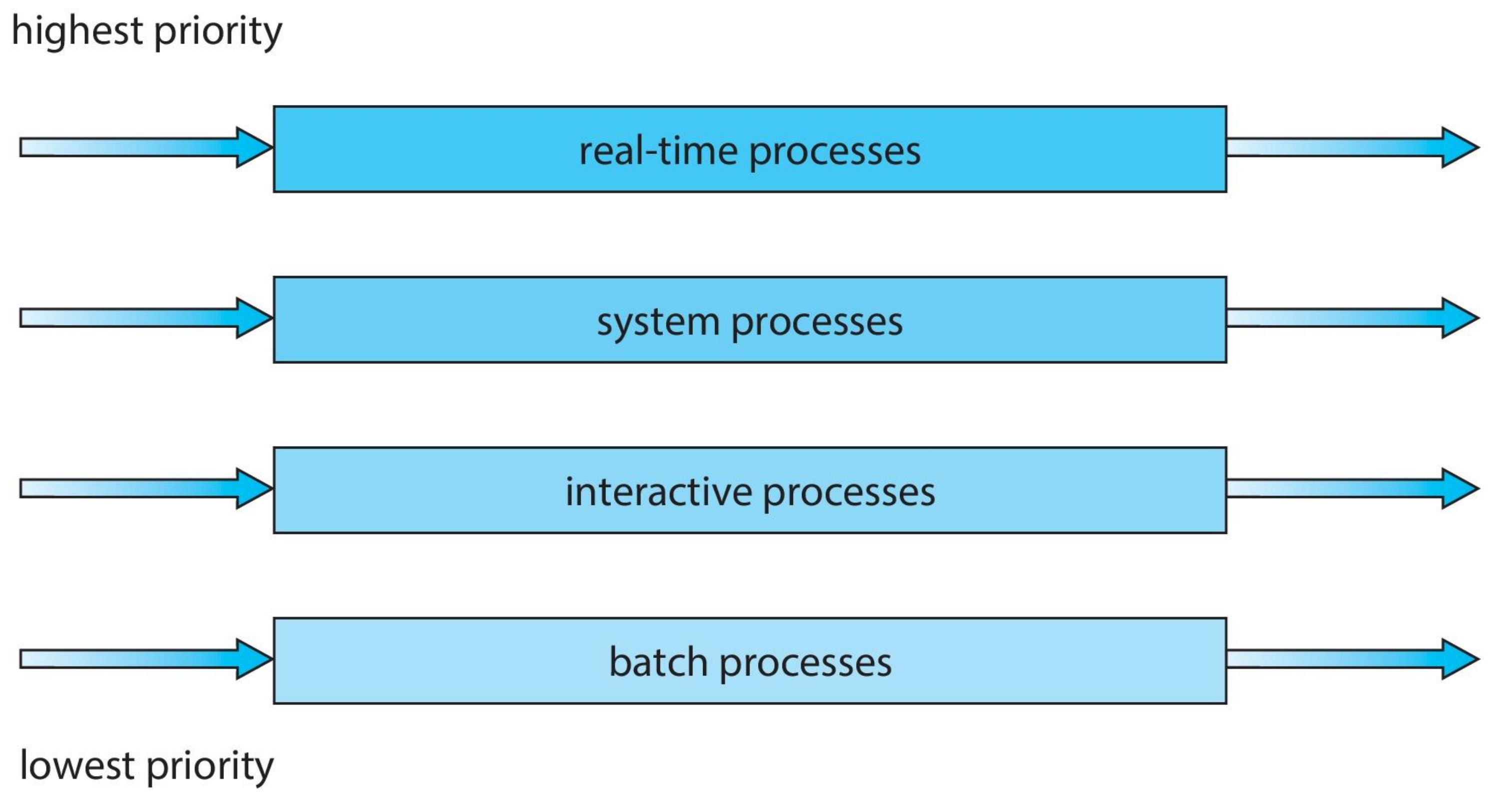
Multi-Level Queue Scheduling

- The ready queue consists of multiple queues
- **Multilevel queue scheduler defined by the following parameters:**
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine which queue a process will enter when that process needs service
  - Scheduling among the queues

Multi-Level Queue Scheduling

- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!





Multi-Level Queue Scheduling - Example

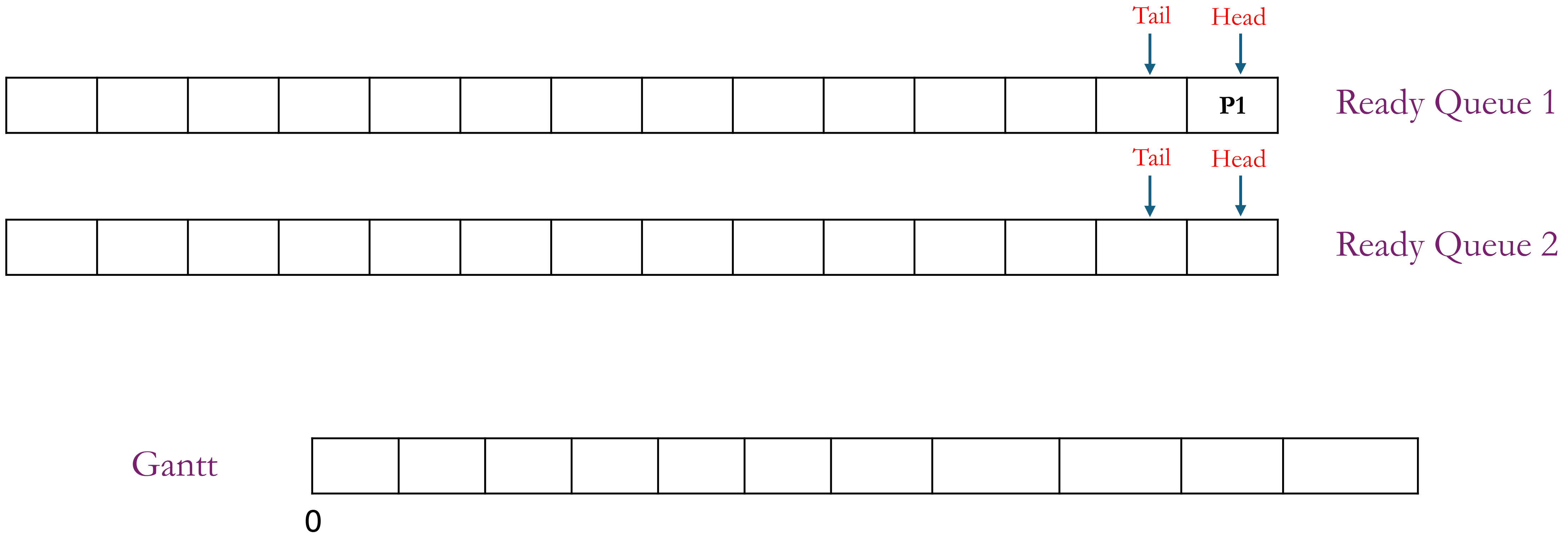
- Consider a system with four processes and two ready queues. The scheduling is governed by the following rules:
  - There are two queues: Q1 (High Priority) and Q2 (Low Priority).
  - Processes in Q1 have absolute priority over processes in Q2. If a Q1 process arrives while a Q2 process is running, the Q2 process will be preempted.
  - Q1 uses a Round Robin (RR) scheduling algorithm with a time quantum of 2ms.
  - Q2 uses a First-Come, First-Served (FCFS) scheduling algorithm.

Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)

Multi-Level Queue Scheduling - Example

- At time 0 → P1 arrives and enters Q1. As it's the only process, it starts executing

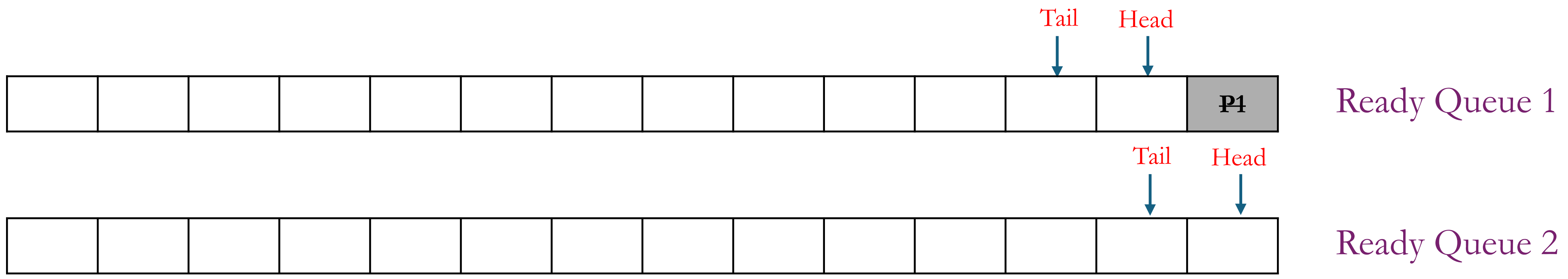
Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)

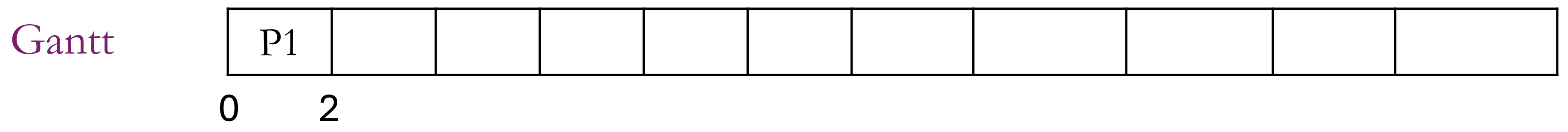


Multi-Level Queue Scheduling - Example

- At time 0 → P1 arrives and enters Q1. As it's the only process, it starts executing

Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)

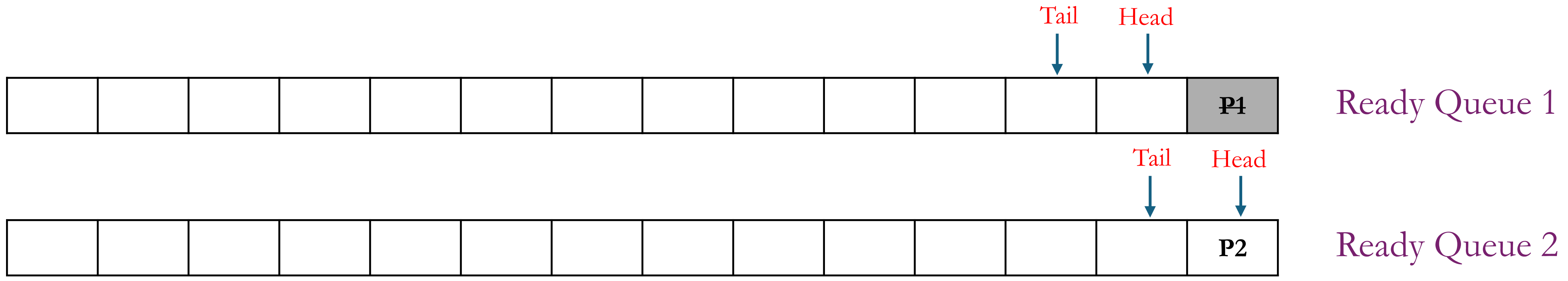


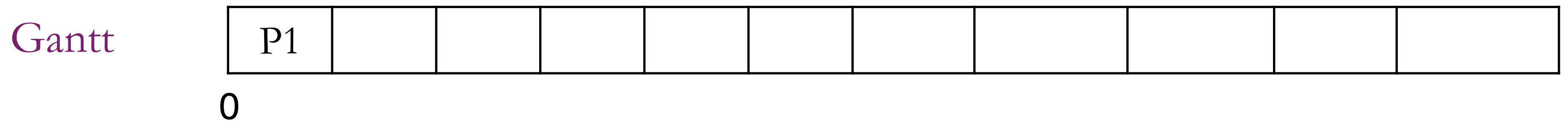


Multi-Level Queue Scheduling - Example

- At time 1 → P2 arrives and is placed in Q2. P1 continues to run because Q1 has higher priority

Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)

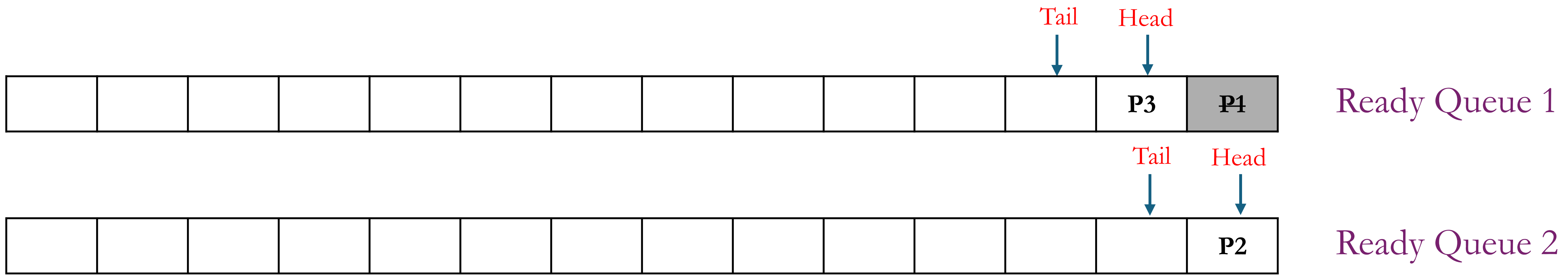


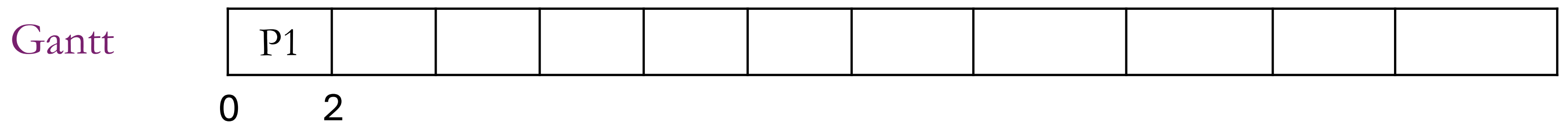


Multi-Level Queue Scheduling - Example

- At time 2 → P3 arrives and is placed in Q1. P1 is Preempted and added back to Ready Queue 1

Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)

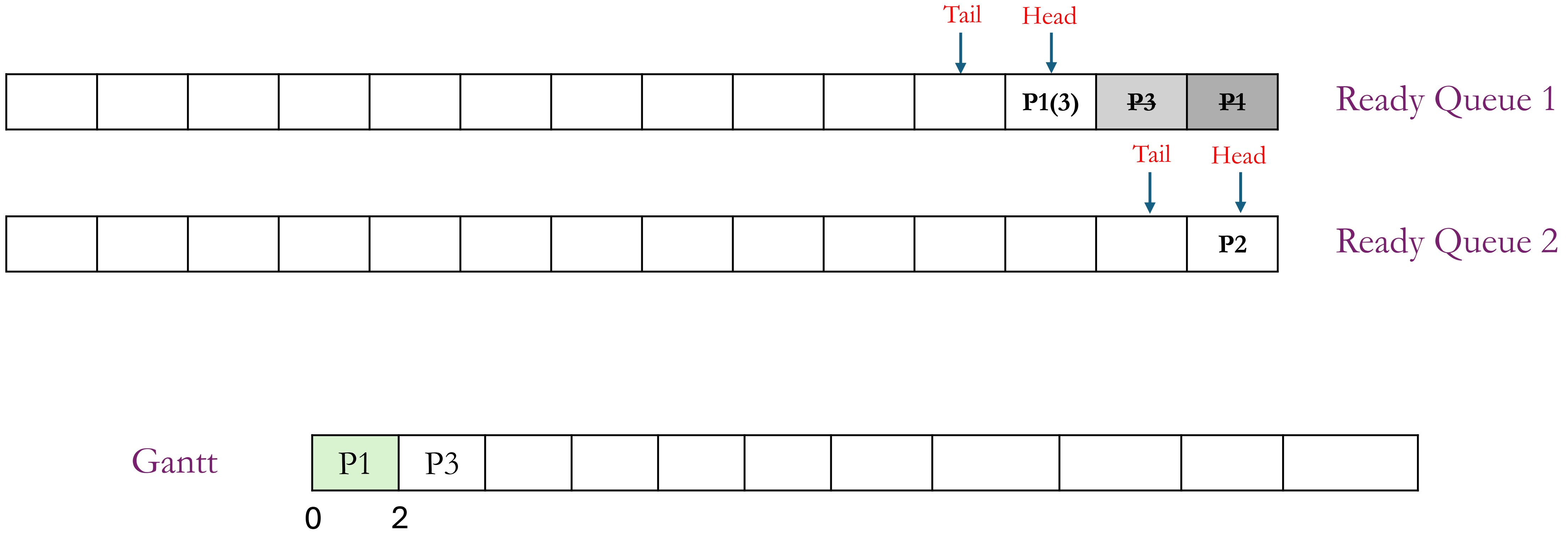




Multi-Level Queue Scheduling - Example

- At time 2 → P3 arrives and is placed in Q1. P1 is Preempted and added back to Ready Queue 1

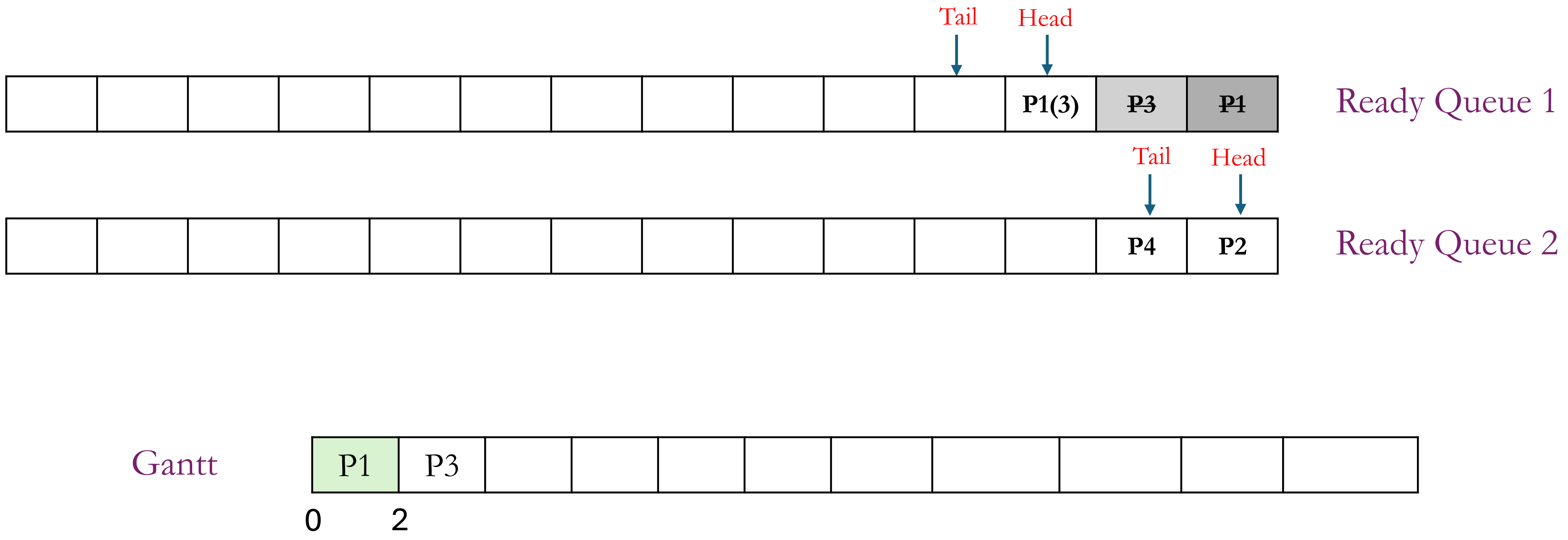
Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)



Multi-Level Queue Scheduling - Example

- At time 3 → P4 arrives and is placed in Q2. P3 is continuing its execution since it has high priority.

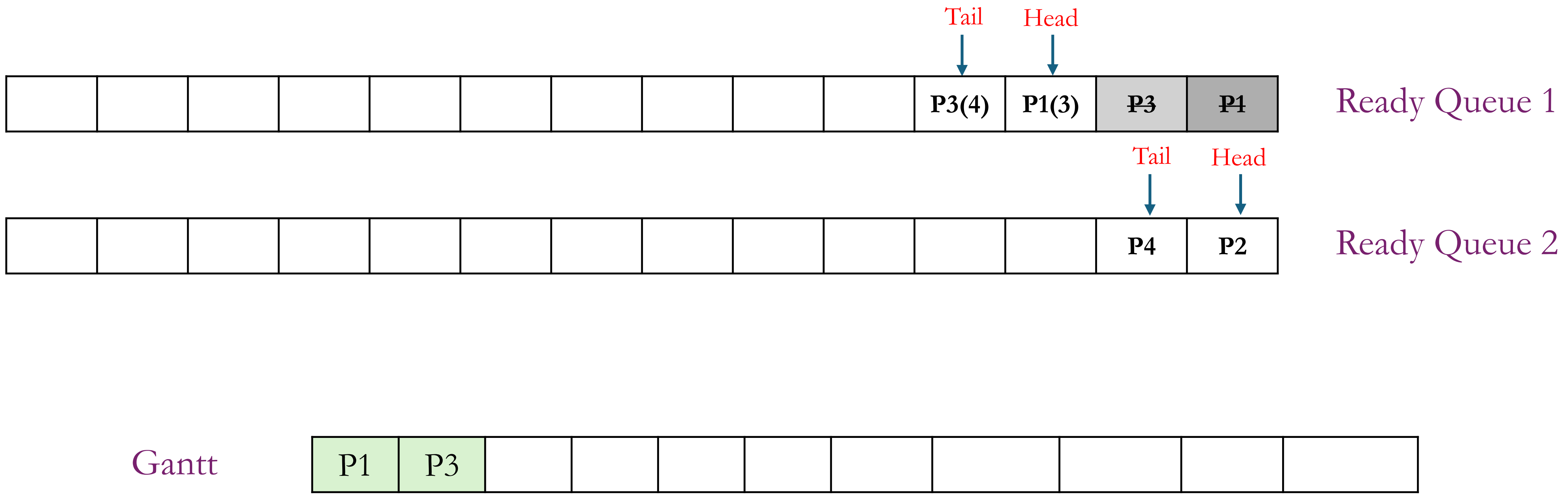
Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)



Multi-Level Queue Scheduling - Example

- At time 4 → P3 complete the quantum and is preempted. P1 is given CPU to execute.

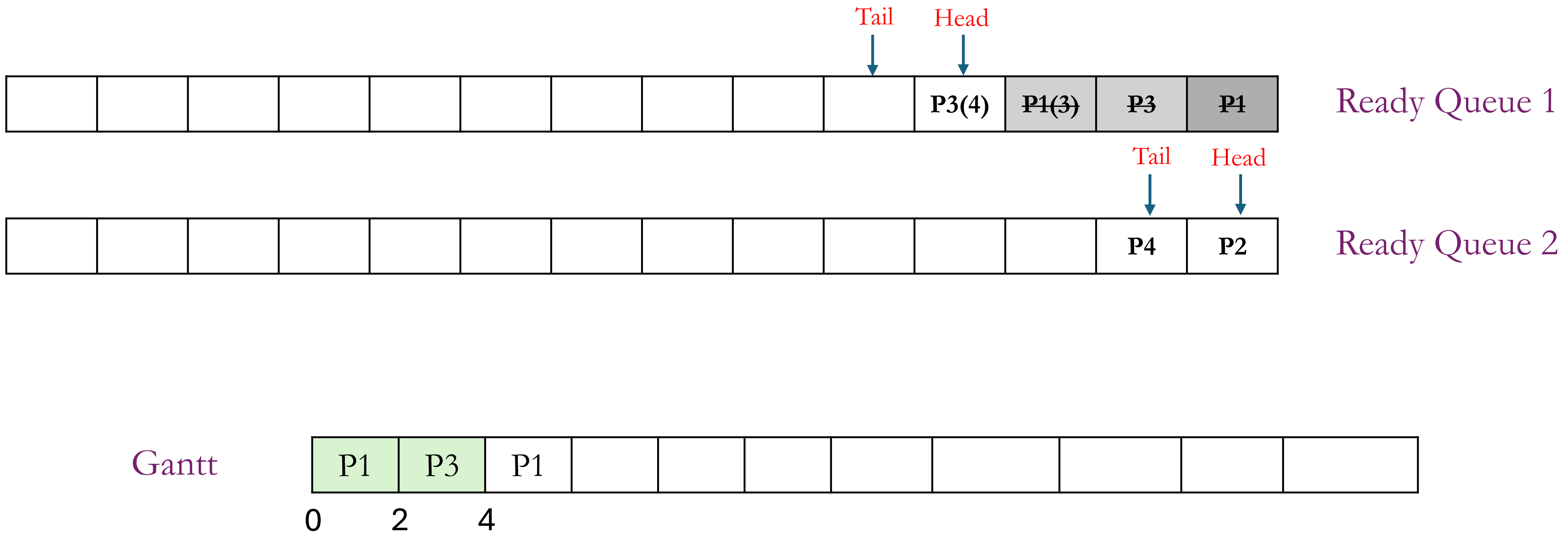
Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)



Multi-Level Queue Scheduling - Example

- At time 4 → P3 complete the quantum and is preempted. P1 is given CPU to execute.

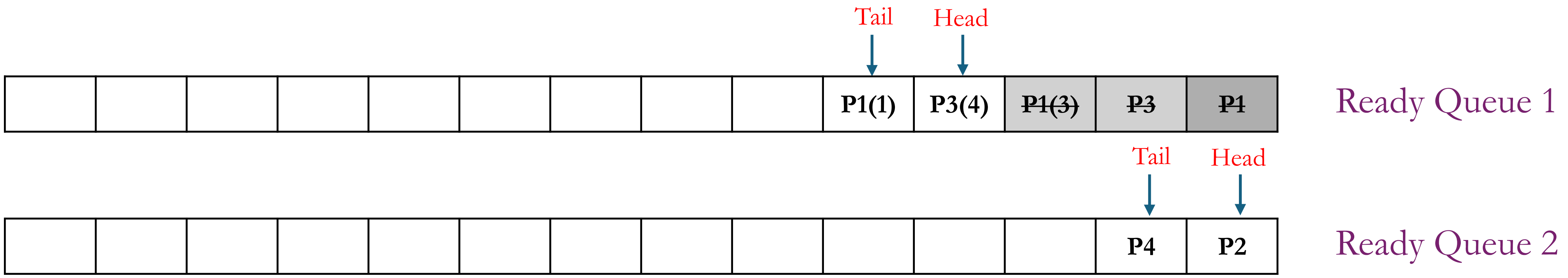
Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)

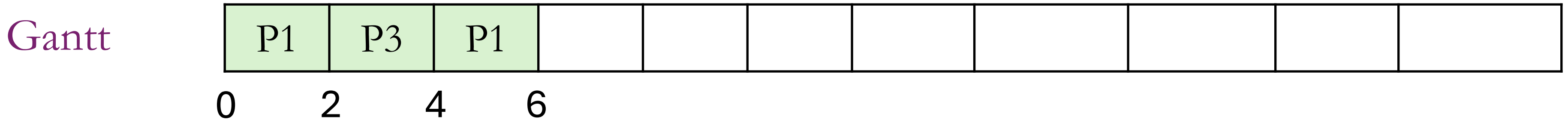


Multi-Level Queue Scheduling - Example

- At time 6 → P1 completes the quantum and is preempted. P3 is given CPU to execute.

Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)

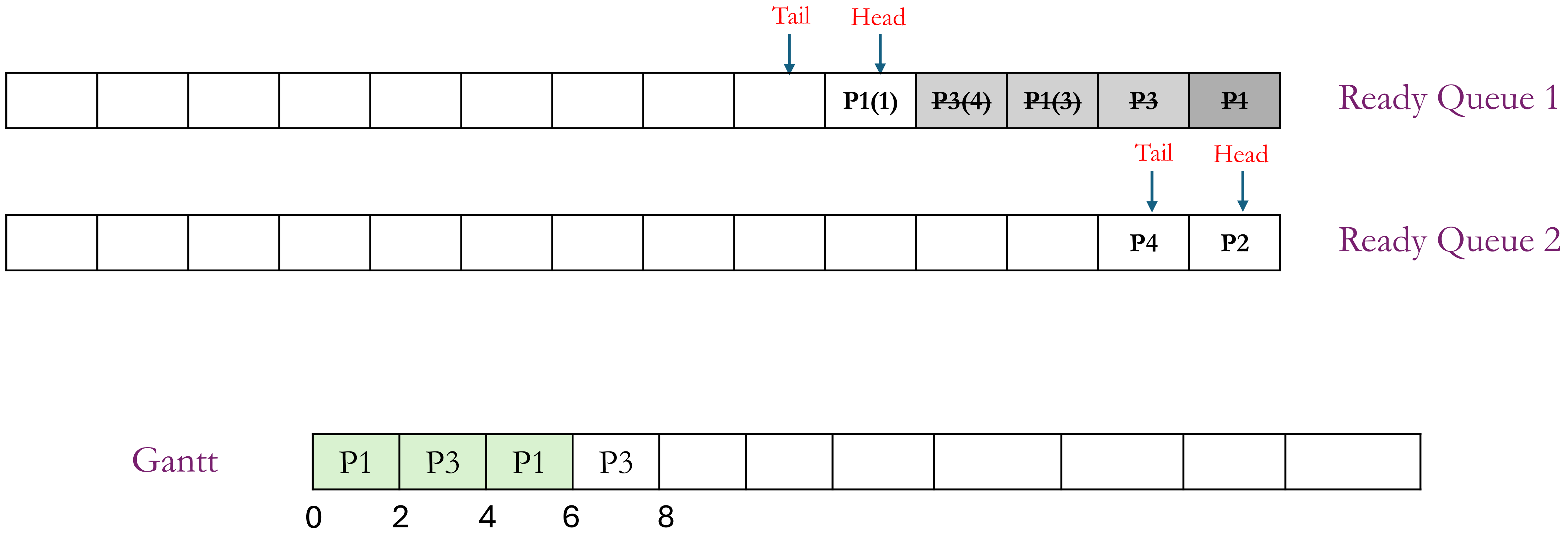




Multi-Level Queue Scheduling - Example

- At time 6 → P1 completes the quantum and is preempted. P3 is given CPU to execute.

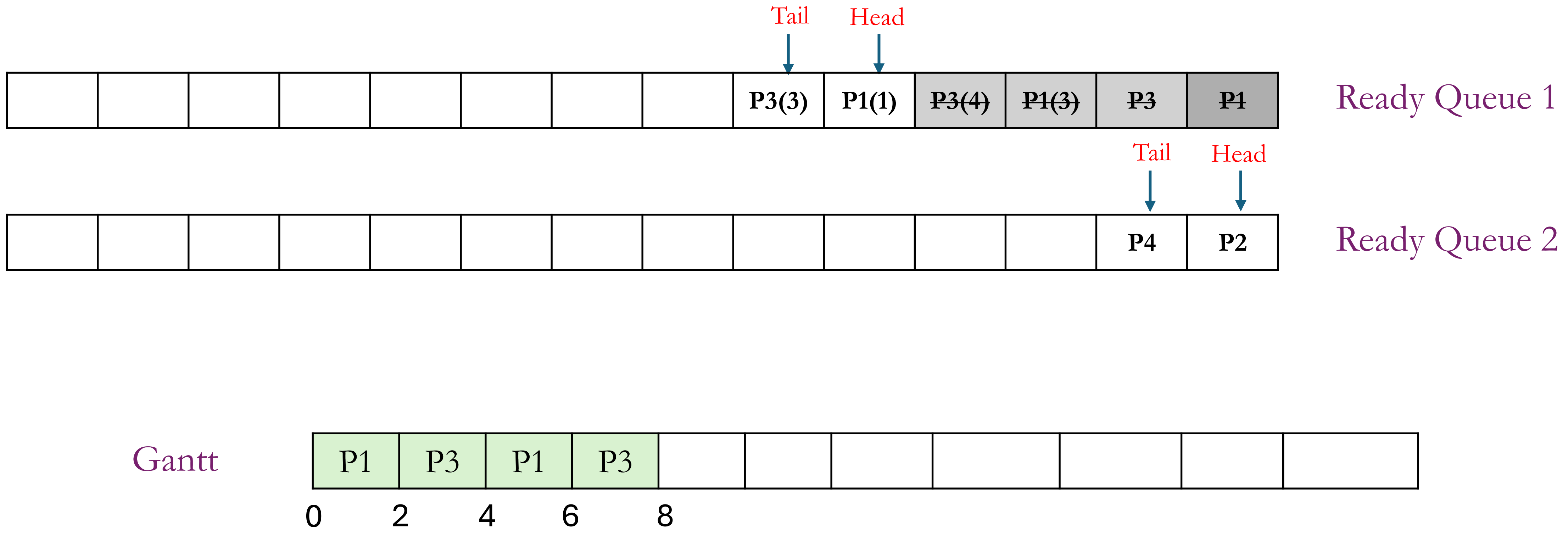
Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)



Multi-Level Queue Scheduling - Example

- At time 8 → P3 completes the quantum and is preempted. P1 is given CPU to execute.

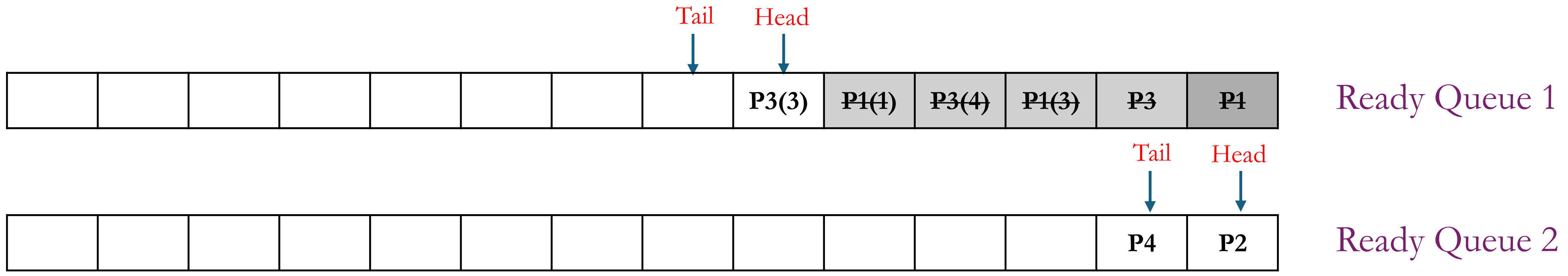
Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)

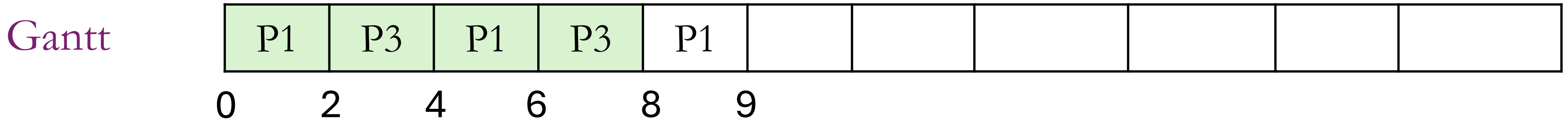


Multi-Level Queue Scheduling - Example

- At time 8 → P3 completes the quantum and is preempted. P1 is given CPU to execute.

Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)

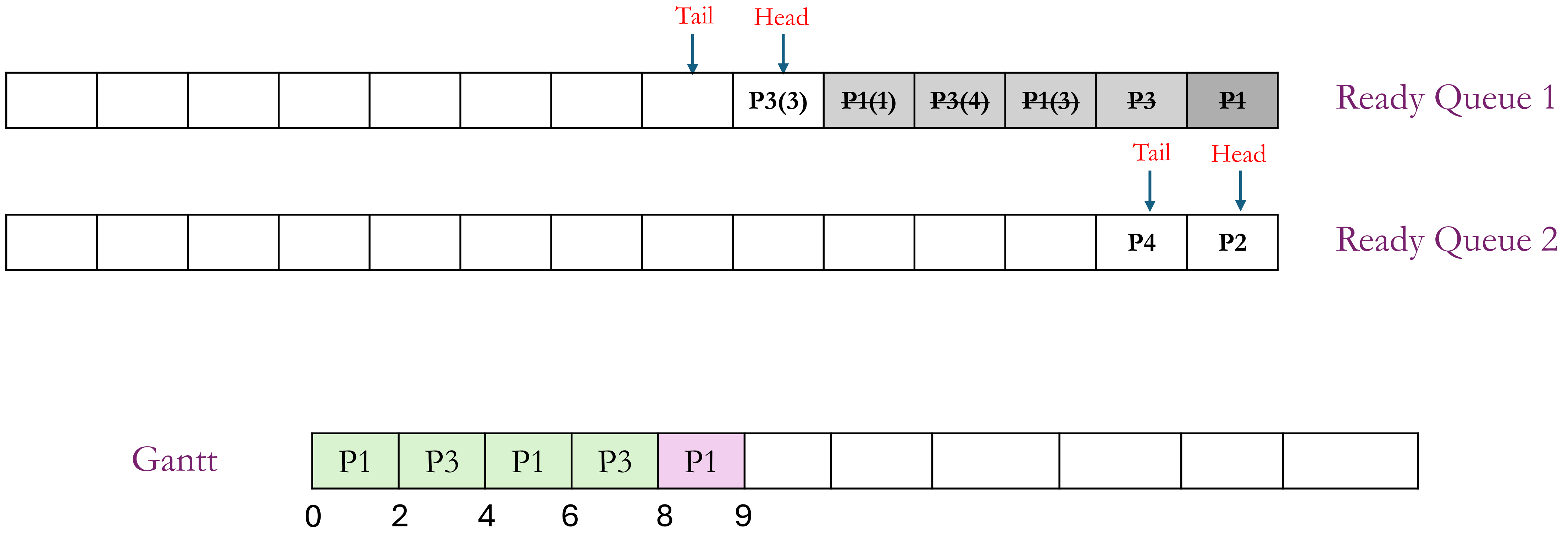




Multi-Level Queue Scheduling - Example

- At time 9 → P1 terminates and P3 gets the CPU.

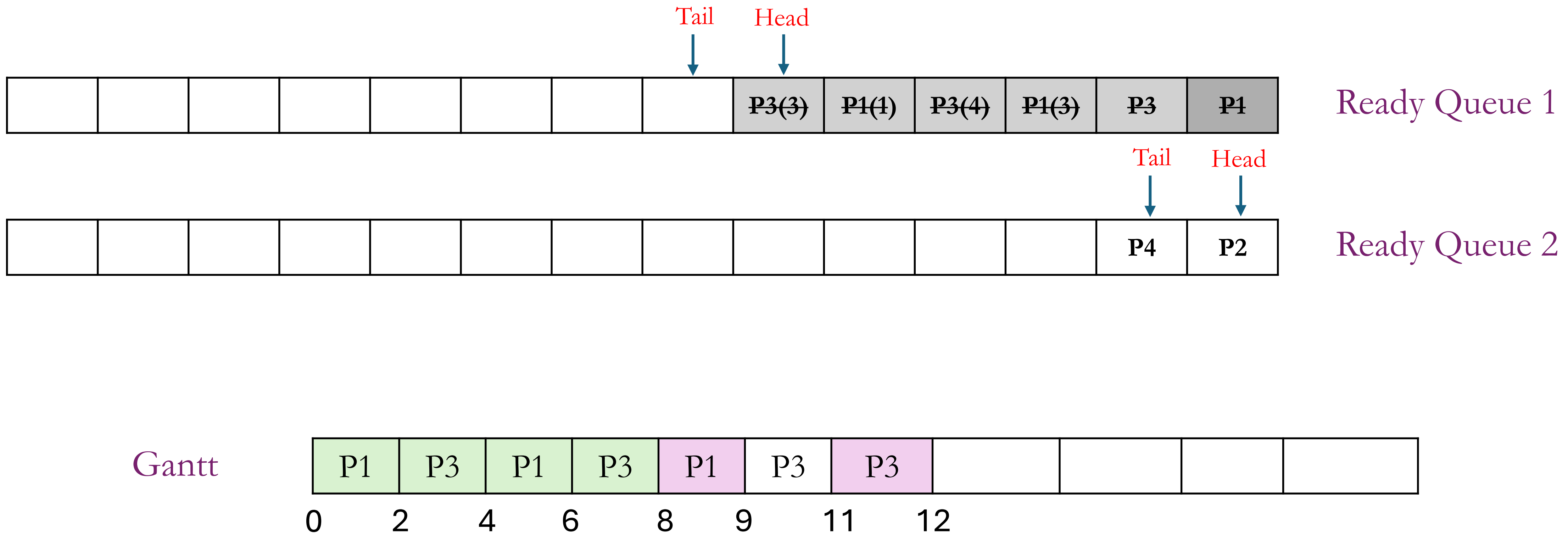
Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)



Multi-Level Queue Scheduling - Example

- At time 11 → P3 gets the CPU since it is the only high priority process.

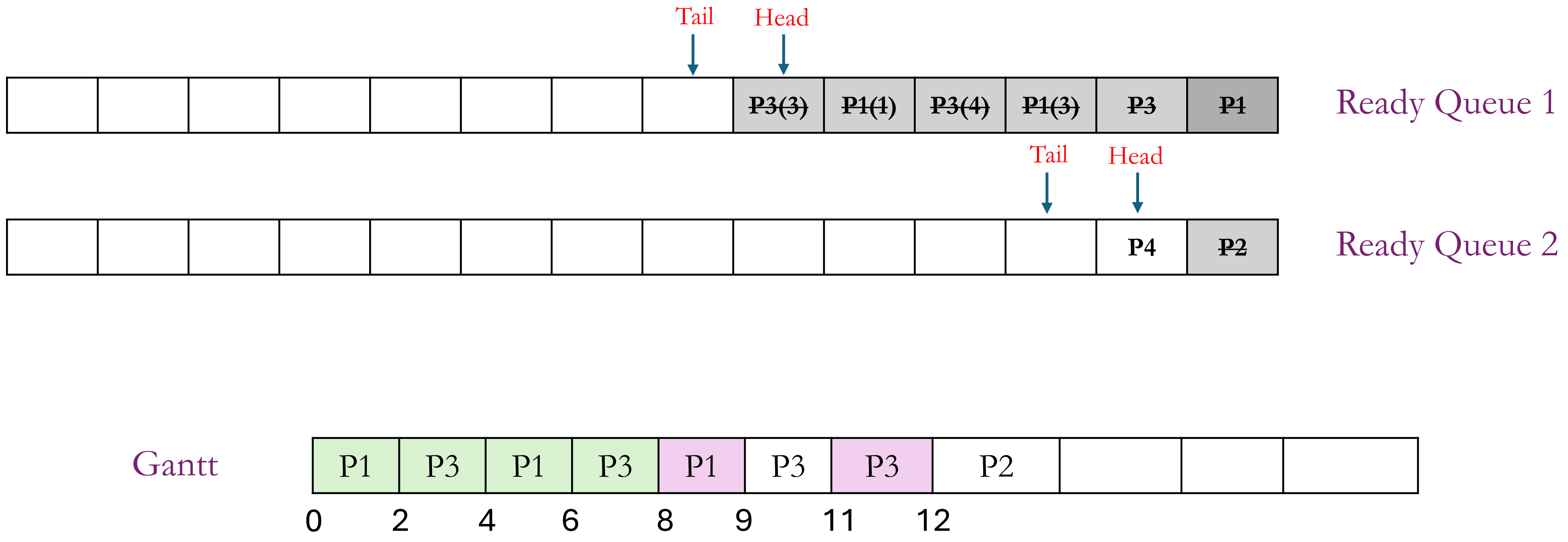
Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)



Multi-Level Queue Scheduling - Example

- At time 12 → All process in Queue 1 is terminated and P2 starts executing.

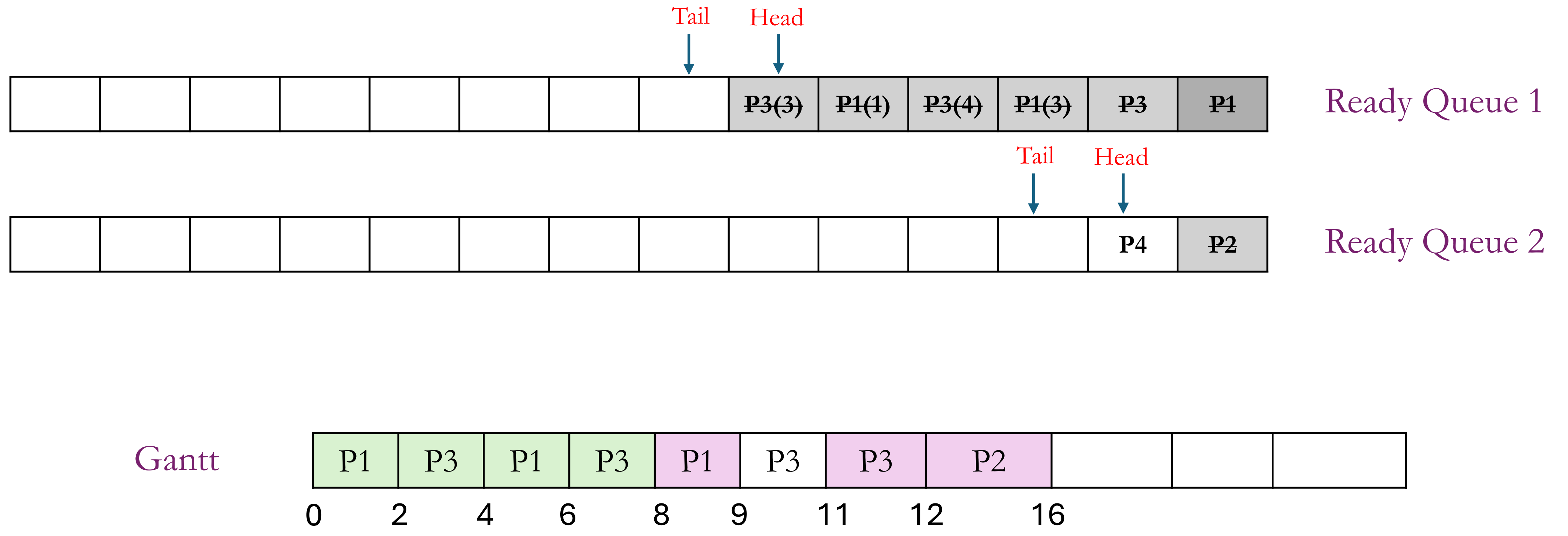
Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)



Multi-Level Queue Scheduling - Example

- At time 12 → All process in Queue 1 is terminated and P2 starts executing.

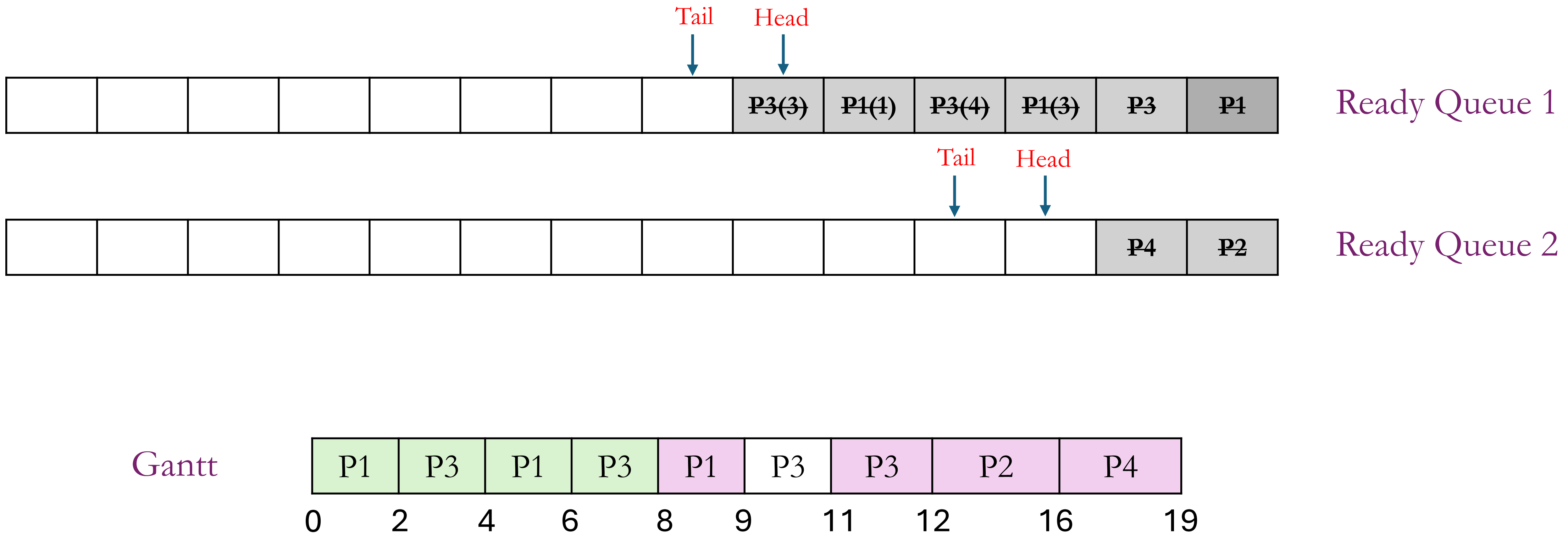
Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)



Multi-Level Queue Scheduling - Example

- At time 16 → All process in Queue 1 is terminated and P4 starts executing.

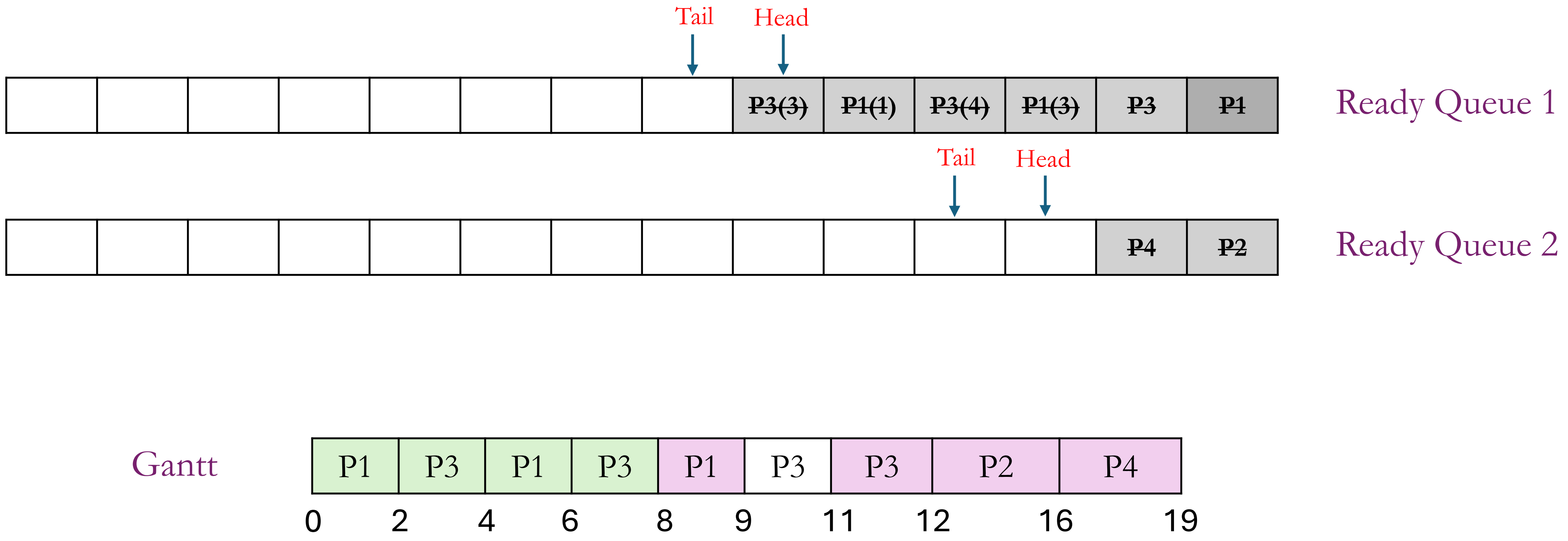
Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)



Multi-Level Queue Scheduling - Example

- At time 16 → All process in Queue 1 is terminated and P4 starts executing.

Process	Arrival Time	Burst Time	Priority
P1	0	5	1(Q1)
P2	1	4	2(Q2)
P3	2	7	1(Q1)
P4	3	3	2(Q2)



Multi-Level Queue Scheduling - Example

- Metrics Calculation

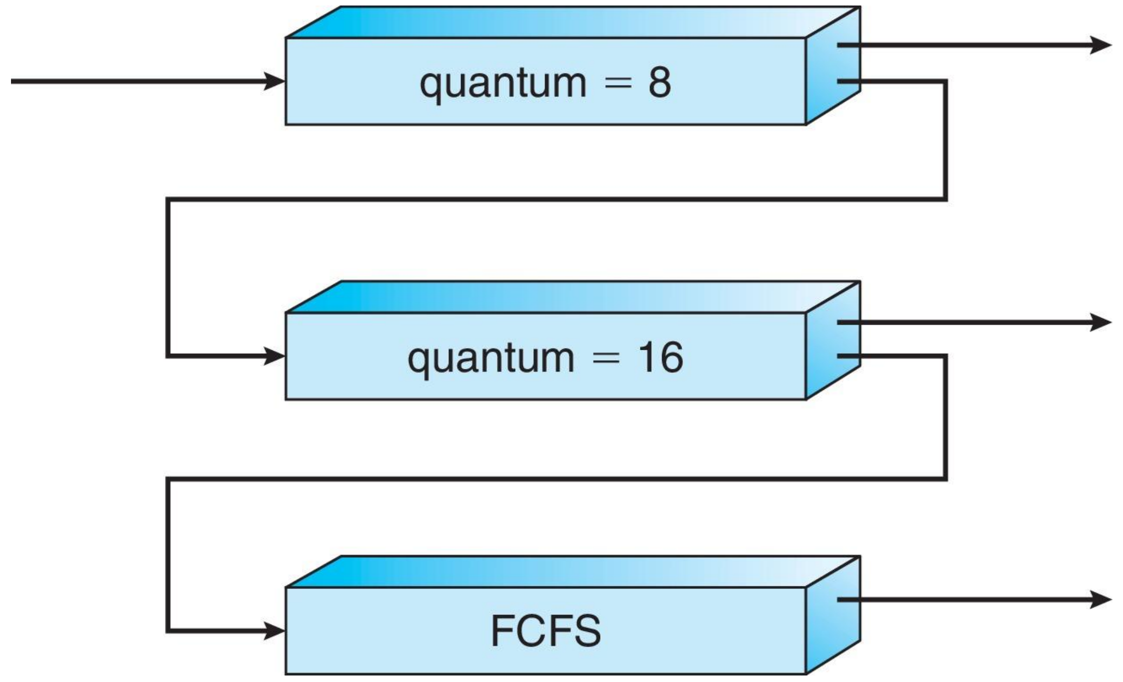
Process	Arrival Time	CPU Burst	Priority and Ready Queue	Finish Time/ Completion Time	Turnaround Time (Completion Time – Arrival Time)	Waiting Time (Turnaround Time – Burst Time)	Response Time time until first scheduled – Arrival (for non-preemptive, RT = WT).
P1	0	5	1(Q1)	9	9 – 0 = 9	9 – 5 = 4	0 – 0 = 0
P2	1	4	2(Q2)	16	16 – 1 = 15	15 – 4 = 11	12 – 1 = 11
P3	2	7	1(Q1)	12	12 – 2 = 10	10 – 7 = 3	2 – 2 = 0
P4	3	3	2(Q2)	19	19 – 3 = 16	16 – 3 = 13	16 – 3 = 13
Average					**12.5**	**7.75**	**6.0**

Multi-Level Feedback Queue Scheduling

- A process can move between the various queues. (It allows processes to move between different priority queues)
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine when to upgrade a process
  - Method used to determine when to demote a process
  - Method used to determine which queue a process will enter when that process needs service
  - Aging can be implemented using multilevel feedback queue

Multi-Level Feedback Queue Scheduling

- **Three queues:**
  - Q0 – RR with time quantum 8 milliseconds
  - Q1 – RR time quantum 16 milliseconds
  - Q2 – FCFS
- **Scheduling**
  - A new process enters queue Q0 which is served in RR
  - When it gains CPU, the process receives 8 milliseconds
  - If it does not finish in 8 milliseconds, the process is moved to queue Q1
  - At Q1 job is again served in RR and receives 16 additional milliseconds
  - If it still does not complete, it is preempted and moved to queue Q2

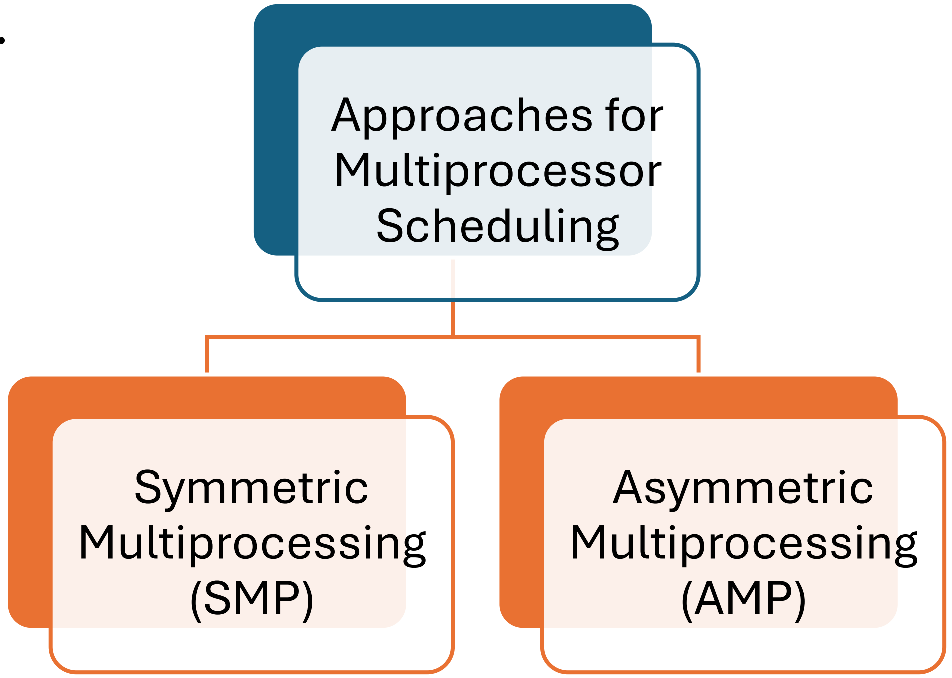


Multi-Processors Scheduling

- A Multi-processor is a system that has more than one processor but shares the same memory, bus, and input/output devices.
- Multiprocessor scheduling is the method used to decide which processes should run when there are multiple processors available.
- The main goal is to design a scheduling system that keeps all the processors as busy as possible, improving the overall performance and throughput of the system.

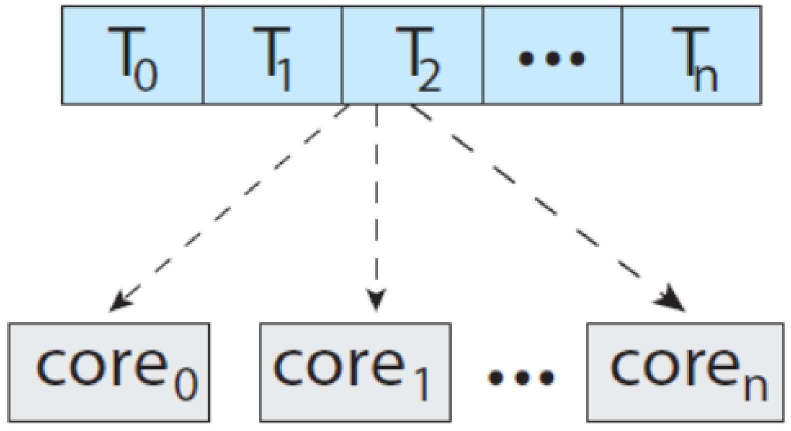
Multi-Processors Scheduling

- **Why is it this Complex?**
  - Load balancing is a problem since more than one processors are present.
  - Processes executing simultaneously may require access to shared data.
  - Cache affinity should be considered in scheduling.


Approaches for Multiprocessor Scheduling
Symmetric Multiprocessing (SMP)
Asymmetric Multiprocessing (AMP)

Multi-Processors Scheduling - AMP

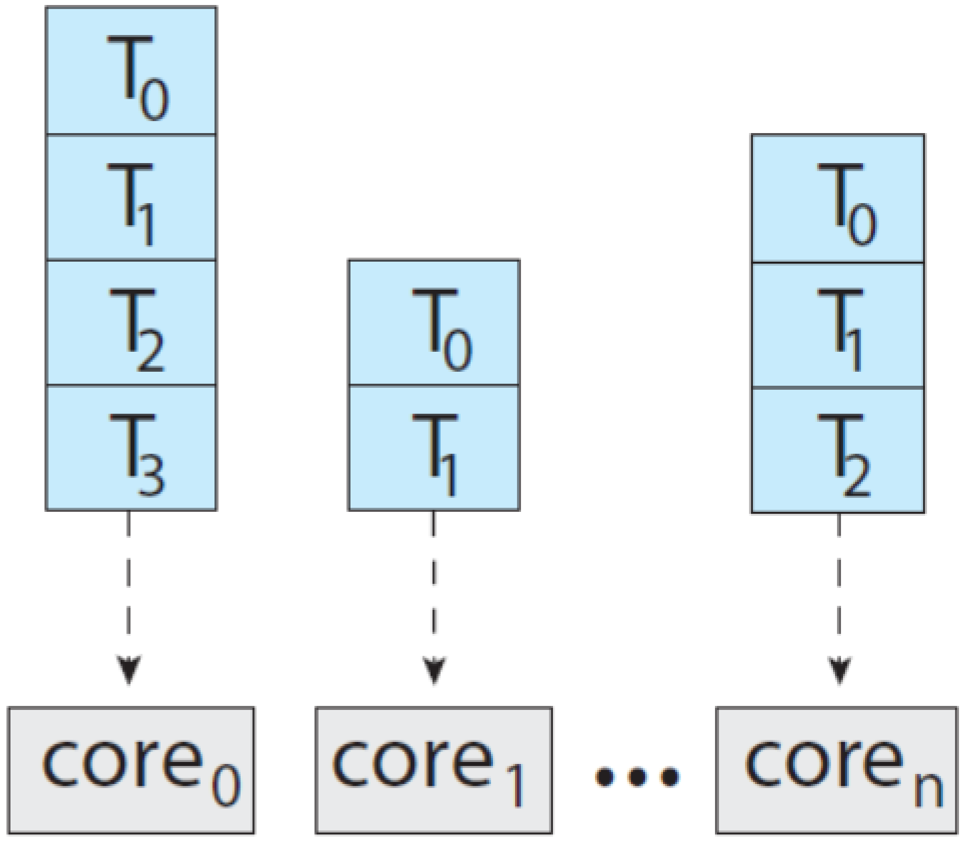
- In this approach, **one processor acts as the "master"** and the **others are "slaves."**
- The **master** processor is **responsible for all scheduling decisions and assigning processes** to the slave processors.
- This is a simple approach because all the complex scheduling logic is handled by one processor, but it can create a bottleneck if the master processor gets overwhelmed.
- If the master server goes down, the whole system comes to a halt.
- However, if one of the slave servers goes down, the rest of the system keeps working.



common ready queue
(a)

Multi-Processors Scheduling - SMP

- This is the more common approach.
- In SMP, **each processor is self-scheduling**.
- All processes are kept in a common ready queue, and each processor independently picks a process to run from that queue.
- This is more balanced because the scheduling work is distributed among all the processors.
- However, it's more complex to design because the processors need to be careful not to pick the same process at the same time.



per-core run queues
(b)

Multi-Processors Scheduling - SMP

- **Pattern of Deployment**
    - **Symmetrical Scheduling with global queues**: If the processes to be executed are in a common queue or a global queue, the scheduler for each processor checks this global-ready queue and selects a process to execute.
    - **Symmetrical Scheduling with per queues**: If the processors in the system have their own private ready queues, the scheduler for each processor checks their own private queue to select a process.

Multi-Processors Scheduling - SMP

- **Processor Affinity**
  - When a process runs on a processor, it builds up a cache of data in that processor's memory.
  - If the process is moved to a different processor, this cache becomes invalid, and the new processor has to rebuild it.
  - This can be inefficient.
  - To avoid this, schedulers often try to keep a process running on the same processor.
  - This is called processor affinity.

Multi-Processors Scheduling - SMP

Soft Affinity

- The operating system tries to keep a process on the same processor but doesn't guarantee it.
- If the system needs to move the process for load balancing, it will.

Hard Affinity

- The system allows a process to specify a subset of processors on which it can run, ensuring it never moves to a processor outside that set.

Multi-Processors Scheduling - SMP

- Load Balancing
- In systems with Symmetric Multiprocessing, it's possible for one processor to be very busy while another is idle.
- Load balancing is the process of distributing the workload evenly across all processors to keep them all busy. This can be done in two ways:
  - **Push Migration**: A specific task periodically checks the load on each processor and, if it finds an imbalance, moves processes from overloaded processors to idle or less-busy ones.
  - **Pull Migration**: An idle processor pulls a waiting task from a busy processor to execute.