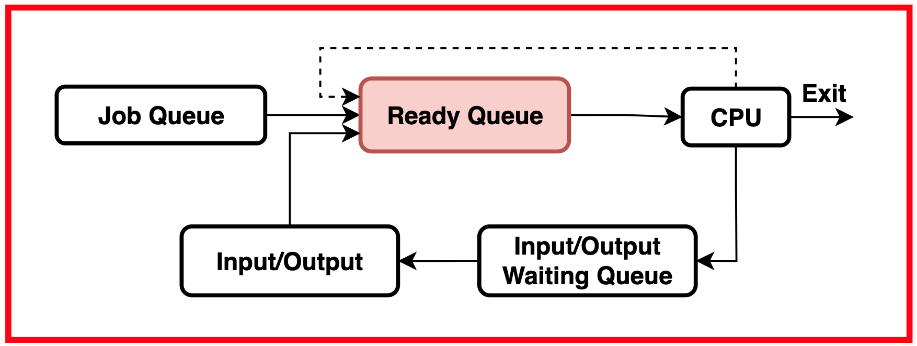
PROCESS MANAGEMENT

1. **Process Scheduling Algorithm (Pre-emptive and Non-Preemptive)**

Process Scheduling handles the **efficient allocation of system resources** like CPU time and memory to multiple processes or threads running at the same time. It ensures processes are executed in a way that **optimises system performance**, resource utilisation, and user responsiveness.



This mechanism allows the operating system to select a process from the **ready queue** and give it CPU time for execution. It decides which process should run next, how long it should run, and when to preempt or suspend it to let other processes execute.

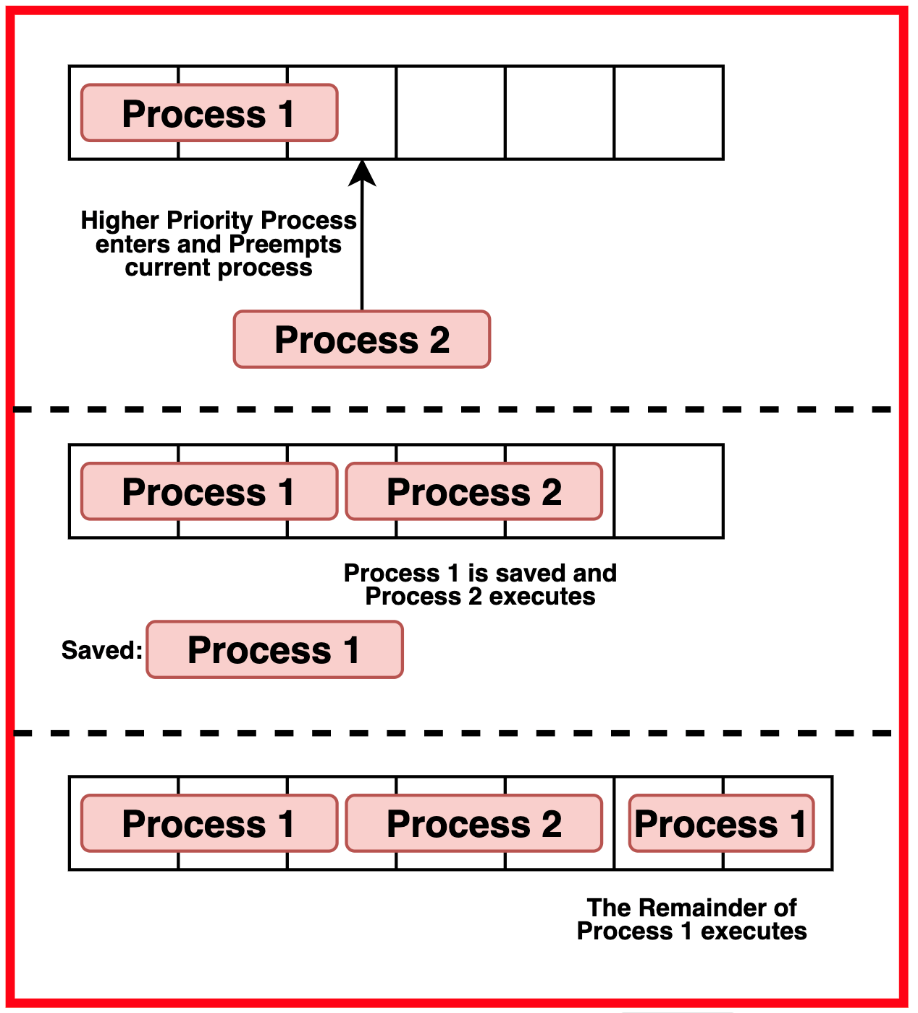
**Pre-emptive and Non-Preemptive Scheduling**

**Preemptive Scheduling**

In **Preemptive Scheduling**, the operating system can **interrupt a currently running process** and give the CPU to another process. This can happen if a higher-priority process arrives, a timer indicates the current process's time slice is over, or an event needing immediate attention occurs.

In preemptive scheduling, processes may be **suspended or preempted** even if they haven't voluntarily given up the CPU. The operating system has more control over process execution, which allows it to allocate CPU resources dynamically.

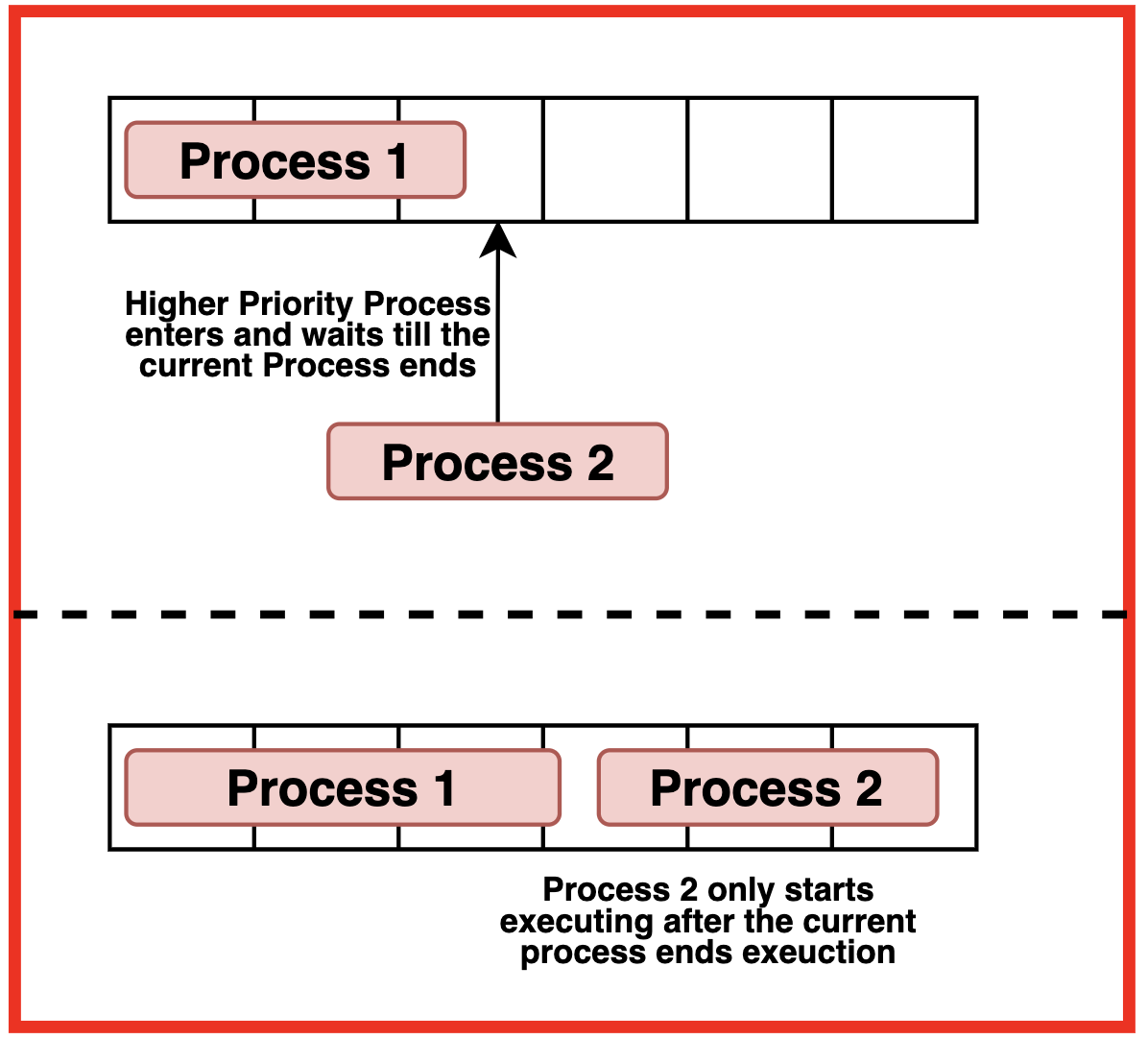
This type of scheduling is ideal for situations where **responsiveness and fairness** are crucial, such as in real-time systems and multitasking environments.



**Non-Preemptive Scheduling**

In **Non-Preemptive Scheduling**, a running process keeps the CPU until it voluntarily releases it by completing its task, waiting for an I/O operation, or entering a waiting state.

Here, processes are **not interrupted** while executing. They keep control of the CPU until they finish or willingly yield it. This type of scheduling is easier to implement and usually has **lower overhead** compared to preemptive scheduling, but it might lead to **slower responsiveness** if long-running processes dominate CPU time.



**Differences between Pre-emptive and Non-Preemptive Scheduling**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Preemptive** | **Non-Preemptive** |
| **Interruption** | Processes can be interrupted while executing | Processes cannot be interrupted while executing |
| **Control of CPU** | Operating system controls CPU allocation | Processes retain control of the CPU until completion |
| **Response Time** | Typically shorter response times | Response times may be longer |
| **Fairness** | Allows for fairness by using priority-based scheduling | May face fairness issues if long-running processes dominate the CPU |
| **Complexity** | More complex due to dynamic process switching | Simpler to implement and manage |
| **Resource Utilisation** | Generally more efficient CPU utilisation | May be less efficient if processes monopolise CPU |
| **Suitable Environments** | Ideal for multitasking and real-time systems | Suitable for simpler systems or those needing predictable behavior |

**Real-world Examples**

**Preemptive Scheduling**

* Preemptive scheduling is crucial for modern multitasking operating systems like Windows, macOS, and Linux. These systems run multiple processes at the same time and ensure each process gets fair CPU time, allowing high-priority tasks to be executed promptly.
* In **real-time systems**, such as automotive systems, preemptive scheduling ensures timely responses to sensor inputs or critical actions like applying brakes, which requires quick reactions.
* For **network servers**, preemptive scheduling is beneficial for handling multiple client requests simultaneously. It ensures that no single client request dominates CPU resources, keeping the system responsive for all clients.

**Non-Preemptive Scheduling**

* Non-preemptive scheduling is used in **resource-constrained embedded systems** where timing requirements are strict. It simplifies timing analysis and ensures predictable execution, which is ideal for systems like industrial automation or IoT devices.
* It is also suitable for **batch processing systems** where tasks run sequentially without interruption. For example, in payroll systems, each calculation task can complete uninterrupted, ensuring data consistency.
* Non-preemptive scheduling can be used in **single-user systems** where users expect consistent behavior rather than responsiveness, such as in personal computers running desktop applications.