Operating System

Lecture #6

Process Management

Inter Process Communication

- Inter process communication is the mechanism provided by the operating system that allows processes to communicate with each other.

- This communication could involve a process letting another process know that some event has occurred or the transferring of data from one process to another.

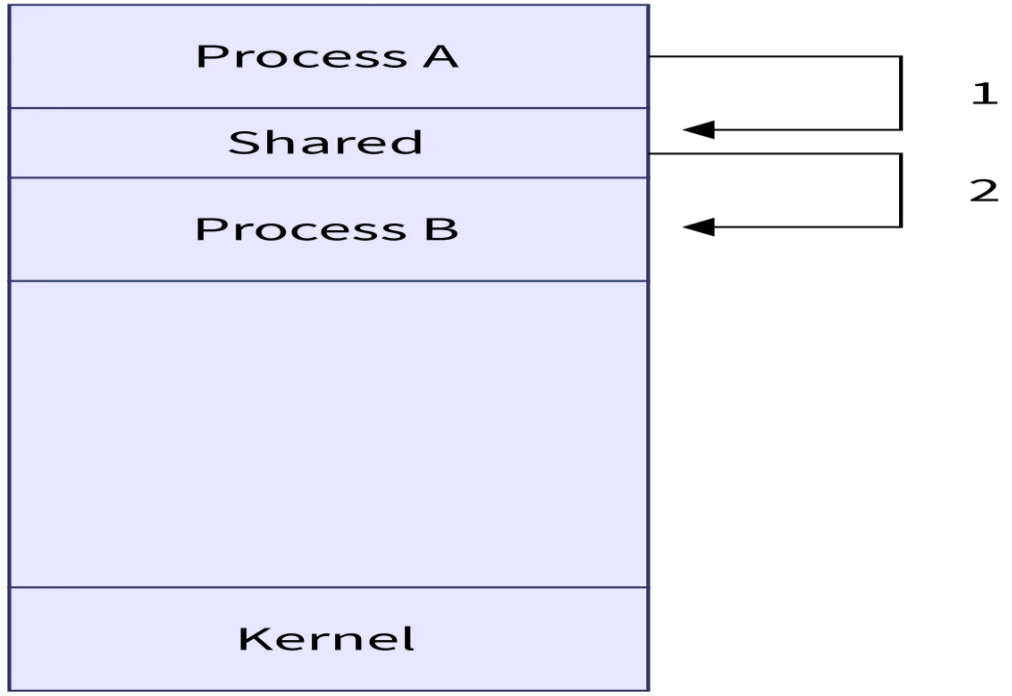
Inter Process Communication

- Processes executing concurrently in the operating system might be either **independent processes** or **cooperating processes.**
- **Cooperating Process** in the operating system is a process that gets affected by other processes under execution or can affect any other process under execution.
- An **independent process** in an operating system is one that does not affect or impact any other process of the system.

Cooperating processes

- It shares data with other processes in the system by directly sharing a memory or by sharing data through **files** or **messages**.
- Cooperating processes in OS requires a communication method that will allow the processes to exchange data and information.
- There are two methods by which cooperating process in OS can communicate:
- Cooperation by Sharing
- Cooperation by Message Passing

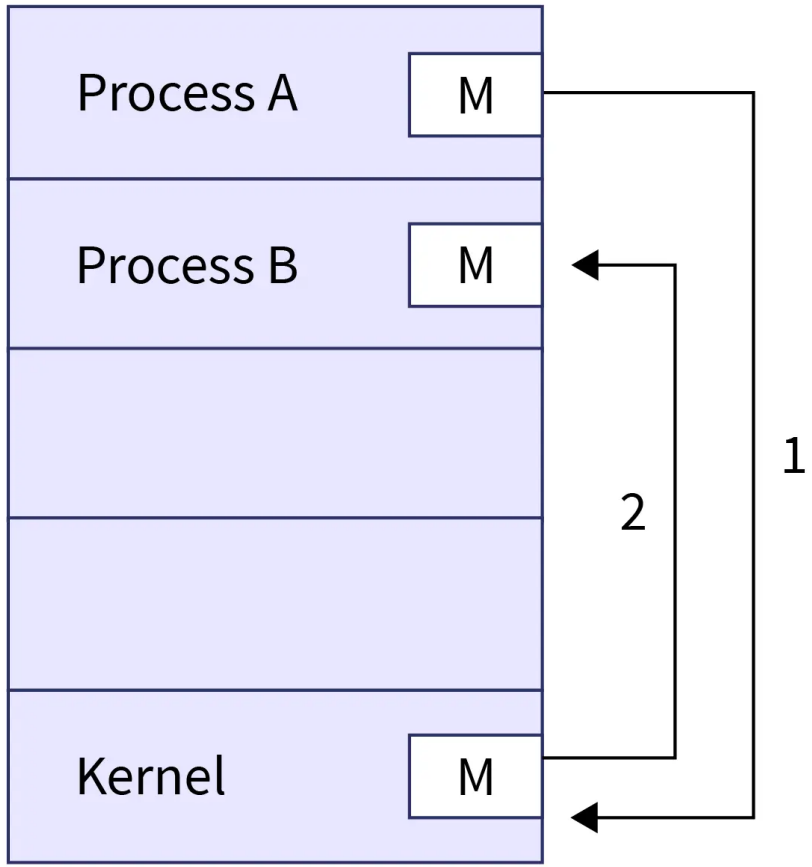
Cooperation by Sharing



The **cooperation processes** in OS can communicate with each other using the shared resource which includes **data**, **memory**, **variables**, **files** , etc.

Processes can then exchange the information by reading or writing data to the shared region. We can use a critical section that provides **data integrity** and avoids **data inconsistency**.

Cooperation by Memory



- The cooperating processes in OS can communicate with each other with the help of message passing. The production process will send the message and the consumer process will receive the same message.
- There is no concept of shared memory instead the producer process will first send the message to the **kernel** and then the **kernel** sends that message to the consumer process.

Task

- Explore the difference of Message passing and shared memory.

Inter Process Communication

- To send and receive messages in message passing, a communication link is required between two processes. There are various ways to implement a communication link.
- Direct and indirect communication

Direct communication

- In direct communication, each process must explicitly name the recipient or sender of the communication.
- `send(P, message) – send a message to process P`
- `receive(Q, message) – receive a message from process Q`
- Link is always created between two processes.
- One direct link can be used only between one pair of communicating processes.
- Two processes can use only a single direct link for communication.

Indirect communication

- In indirect communication, messages are sent and received through mailboxes or ports.
- `send(A, message) – send a message to mailbox A`
  `receive (A, message) – receive a message from mailbox A`
- Different pair of processes can use the same indirect link for communication.
- Also, two processes can two different indirect links for communication.

Threads

- A Thread some time called lightweight process(LMP).
- Thread is a sequential flow of tasks within a process.
- Threads are used to increase the performance of the applications.
- The idea is to achieve parallelism by dividing a process into multiple threads.

Threads

- Each thread has its own program counter, stack, and set of registers.
- But the threads of a single process might share the same code and data/file.
- **Threads are also termed as lightweight processes as they share common resources**

Threads

- **Let us take an example of a human body.**

- A human body has different parts having different functionalities which are working parallelly (Eg: *Eyes, ears, hands*, etc).
- Similarly in computers, a single process might have multiple functionalities running parallelly where each functionality can be considered as a thread

Threads

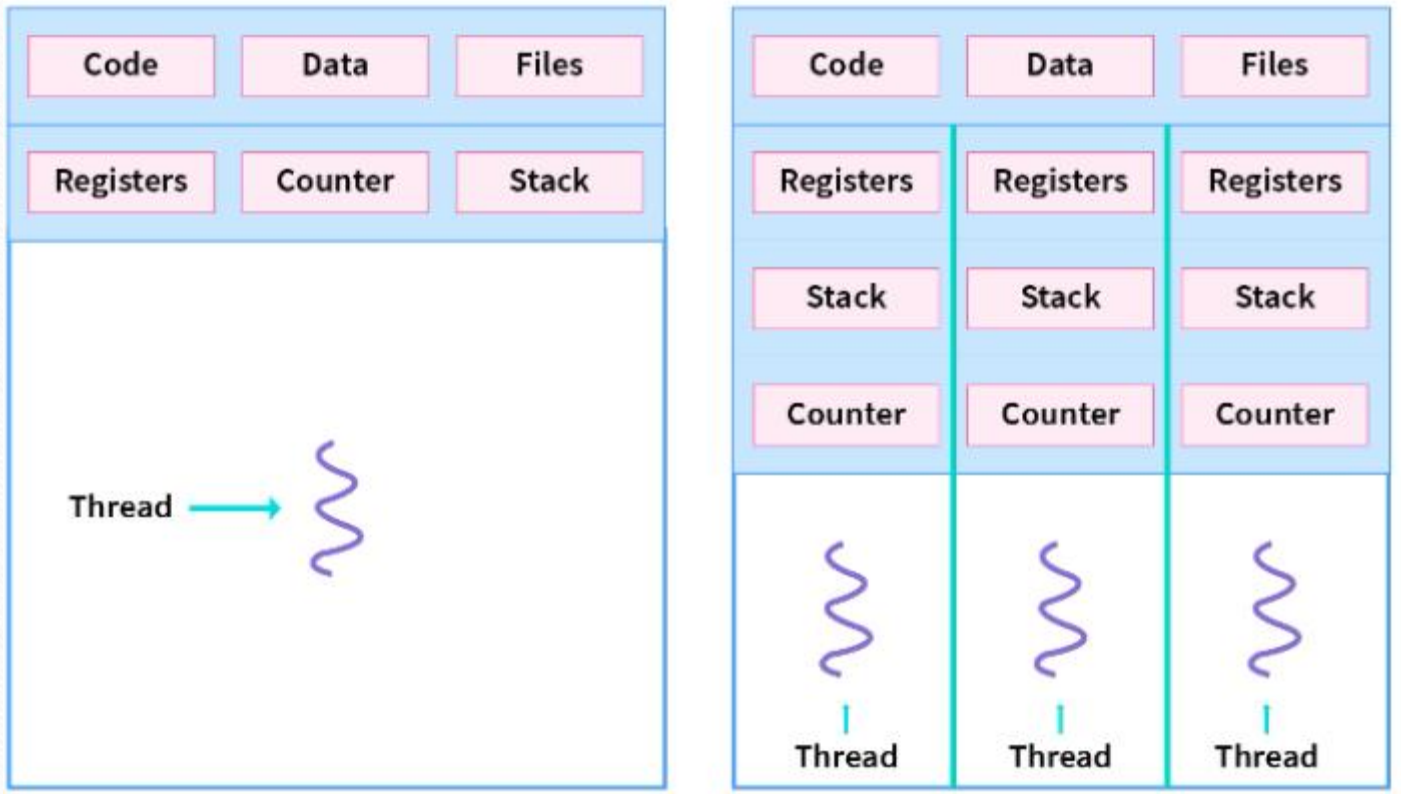
- Each thread has its own set of registers and stack space.
- There can be multiple threads in a single process having the same or different functionality.
- Threads are also termed lightweight processes.
- **Eg:** While playing a movie on a device the audio and video are controlled by different threads in the background.

Threads

Components of Thread

- A thread has the following three components:

1. Program Counter
2. Register Set
3. Stack space



Single-threaded process

Multithreaded process

Why do we need Threads

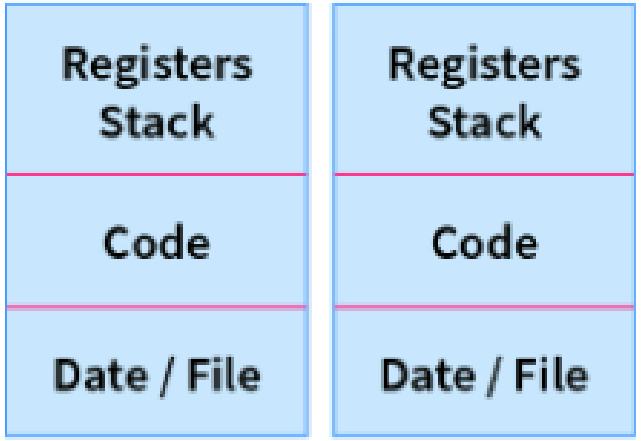
- Threads in the operating system provide multiple benefits and improve the overall performance of the system. Some of the reasons threads are needed in the operating system are:
- Since threads use the same data and code, the operational cost between threads is low.
- Creating and terminating a thread is faster compared to creating or terminating a process.
- Context switching is faster in threads compared to processes.

Process vs Thread

Process simply means any program in execution while the thread is a segment of a process. The main differences between process and thread are mentioned below:

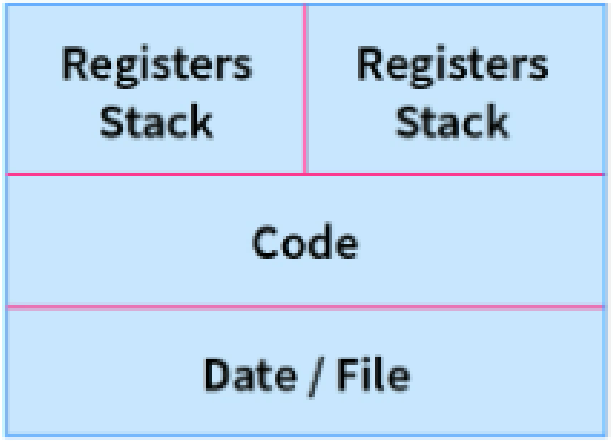
Process	Thread
Processes use more resources and hence they are termed as heavyweight processes.	Threads share resources and hence they are termed as lightweight processes.
Creation and termination times of processes are slower.	Creation and termination times of threads are faster compared to processes.
Processes have their own code and data/file.	Threads share code and data/file within a process.
Communication between processes is slower.	Communication between threads is faster.
Context Switching in processes is slower.	Context switching in threads is faster.
Processes are independent of each other.	Threads, on the other hand, are interdependent. (i.e they can read, write or change another thread's data)
Eg: Opening two different browsers.	Eg: Opening two tabs in the same browser.

PROCESS



Two Different Process

THREAD



Two Threads of a single process

Multithreading

- In Multithreading, the idea is to divide a single process into multiple threads instead of creating a whole new process.
- Multithreading is done to achieve parallelism and to improve the performance of the applications as it is faster in many ways

Advantages

- **Resource Sharing:** Threads of a single process share the same resources such as code, data/file.
- **Responsiveness:** Program responsiveness enables a program to run even if part of the program is blocked or executing a lengthy operation. Thus, increasing the responsiveness to the user.
- **Economy:** It is more economical to use threads as they share the resources of a single process. On the other hand, creating processes is expensive.

Types of Threads

- There are two type of Thread

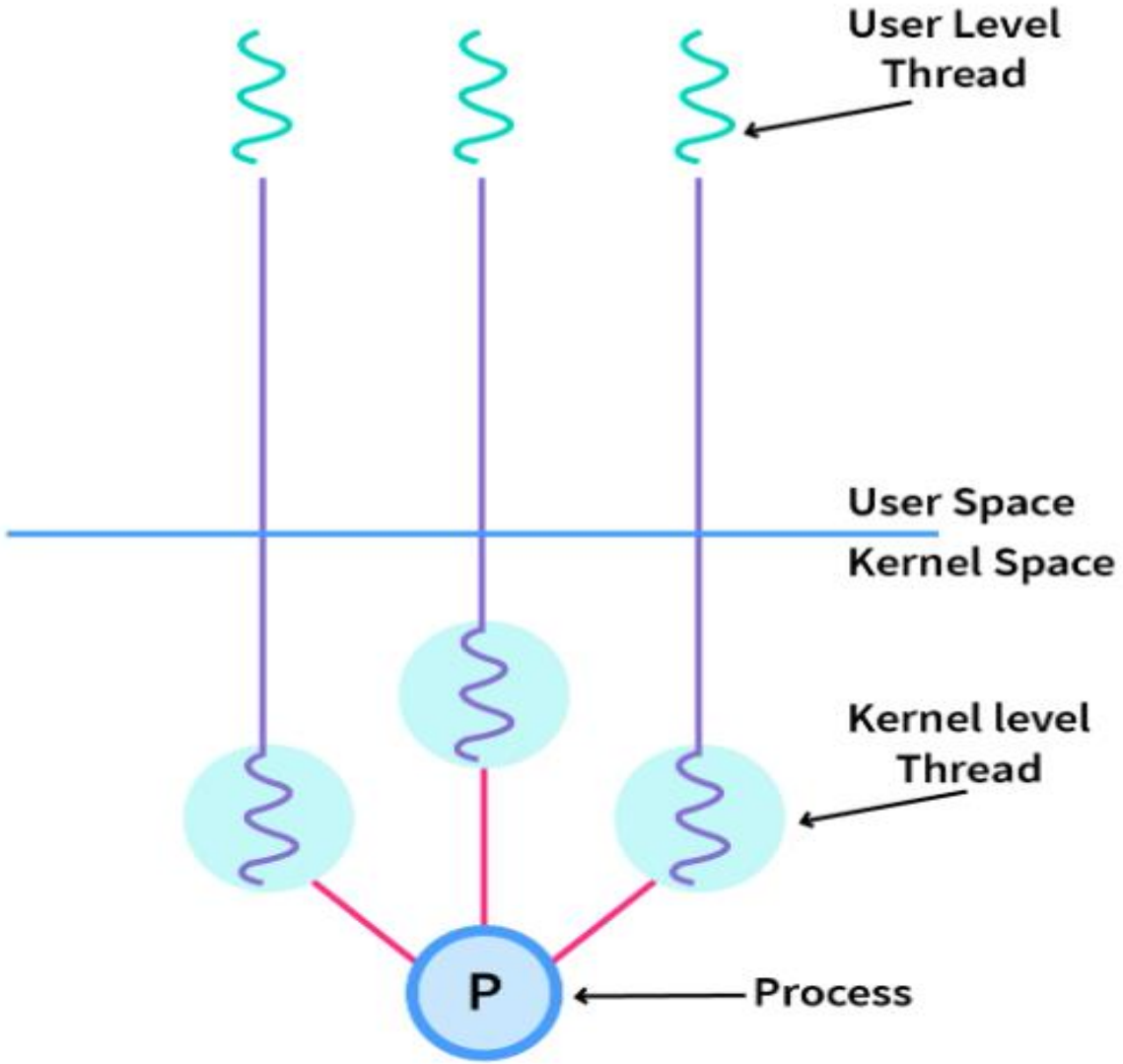
1. User level Thread
2. Kernel level Thread

1. User Level Thread:

- User-level threads are implemented and managed by the user and the kernel is not aware of it.
- User-level threads are **implemented using user-level libraries and the OS does not recognize these threads**.
- User-level thread is **faster to create and manage compared to kernel-level thread**.
- **Context switching in user-level threads is faster**.
- If one user-level thread performs a blocking operation then the entire process gets blocked.

2. Kernel level Thread

- **Kernel level threads are implemented and managed by the OS**.
- Kernel level threads are **implemented using system calls and Kernel level threads are recognized by the OS**.
- Kernel-level threads are **slower to create and manage compared to user-level threads**.
- **Context switching in a kernel-level thread is slower**.
- Even if one kernel-level thread performs a blocking operation, it does not affect other threads


User Level
Thread
User Space
Kernel Space
Kernel level
Thread
P
Process

Advantages of Threading

- Threads improve the overall performance of a program.
- Threads increases the responsiveness of the program
- Context Switching time in threads is faster.
- Threads share the same memory and resources within a process.
- Communication is faster in threads.
- Threads provide concurrency within a process.
- Enhanced throughput of the system.
- Since different threads can run parallelly, threading enables the utilization of the multiprocessor architecture to a greater extent and increases efficiency.

Chapter: CPU Scheduling

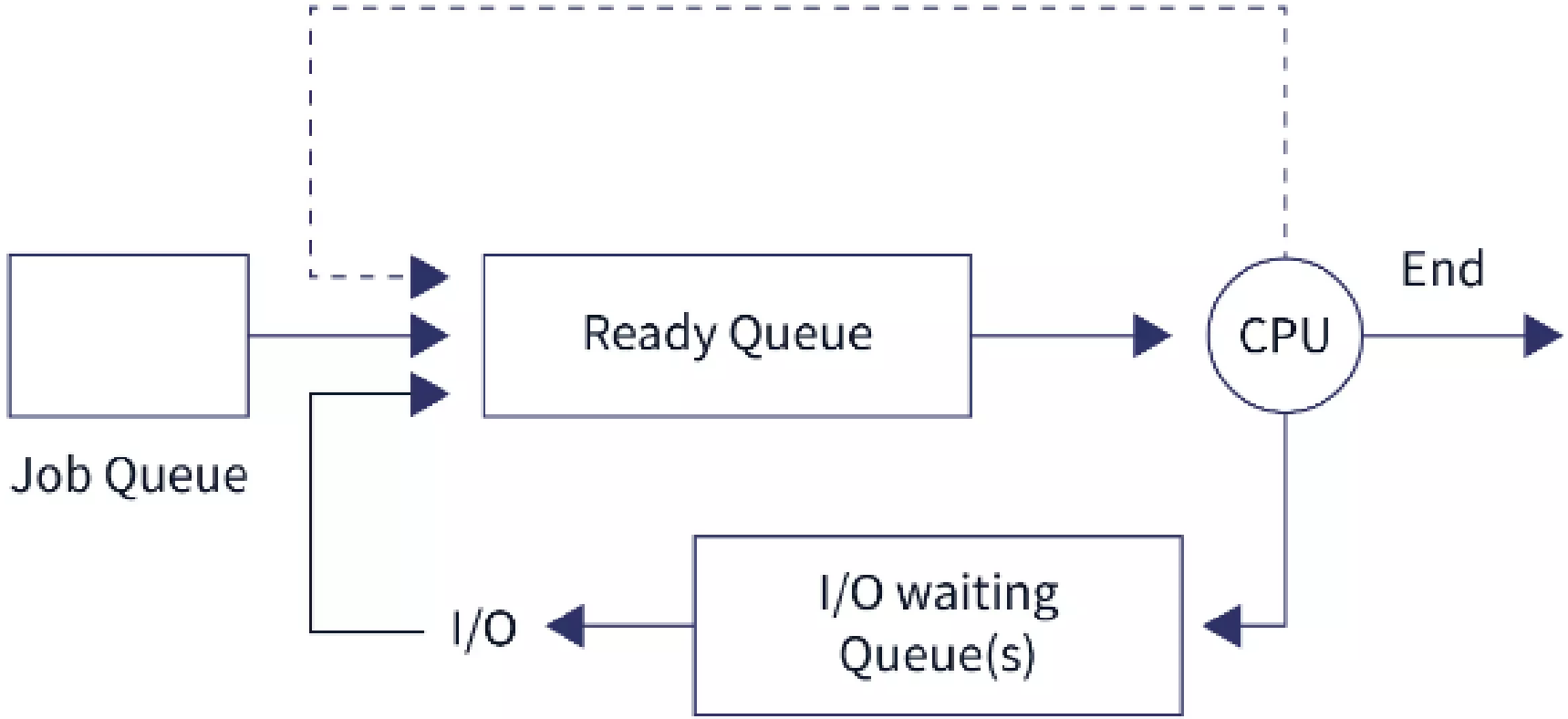
What is Scheduling

- CPU scheduling is the task performed by the CPU that decides the way and order in which processes should be executed.

- The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Scheduling

- Say you have a uniprogramming system (like MS DOS) and a process that requires I/O is being exectued.
- While it waits for the I/O operations to complete, the CPU remains idle.
- This is wrong, because we're wasting the resources of the CPU, as well as time and might result in some processes waiting too long for their execution.


Ready Queue
CPU
End
Job Queue
I/O
I/O waiting
Queue(s)

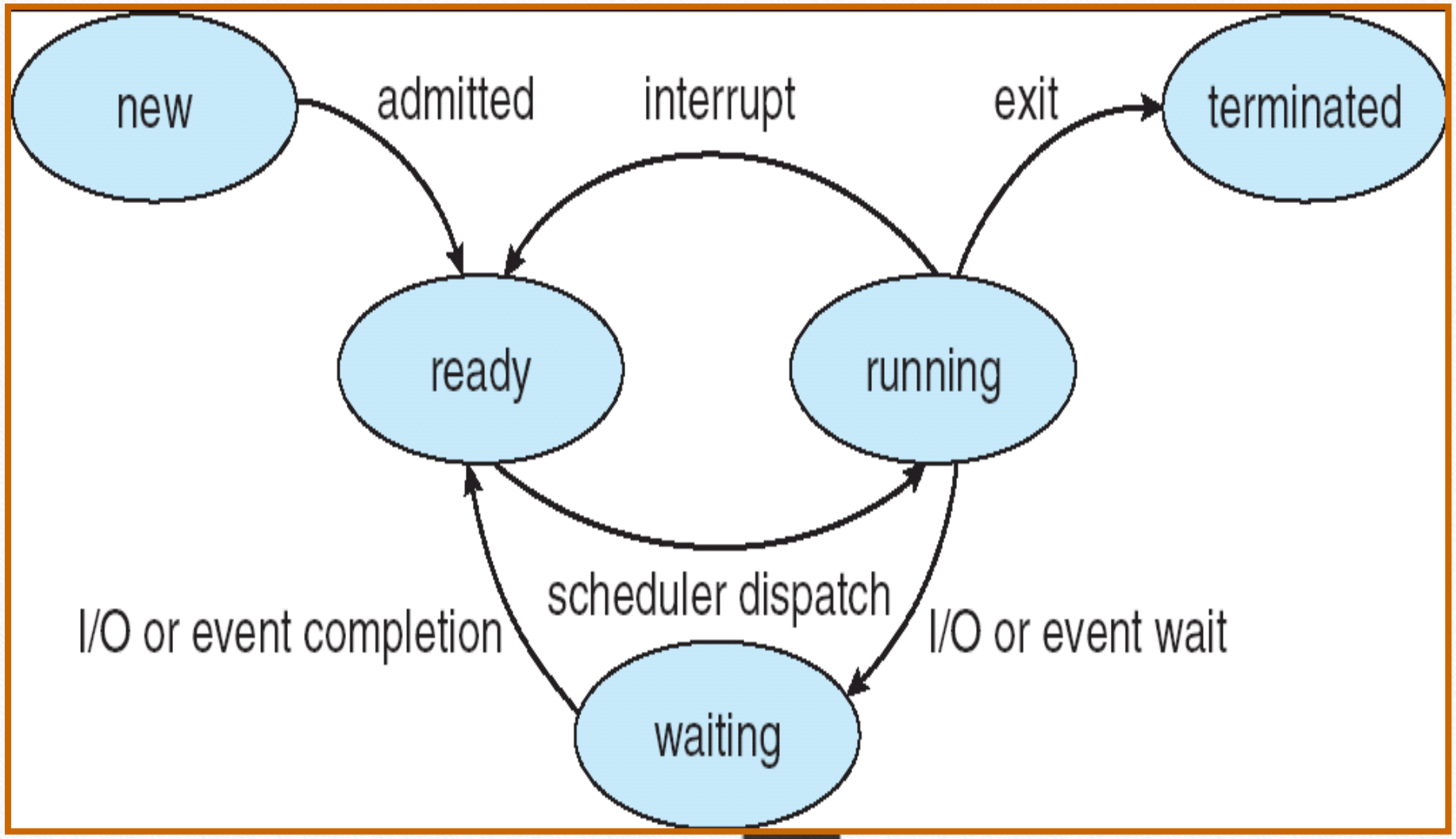
Scheduling

- In multiprogramming systems however, the CPU does not remain idle whenever a process currently executing waits for I/O.
- It starts the execution of other processes, making an attempt to maximize CPU utilization
- **So the main aim is better CPU utilization**

Scheduling

- How does the CPU decide which process should be executed next from the ready queue for maximum utilization of the CPU? This procedure of "scheduling" the processes, is called **CPU scheduling**.
- CPU scheduling is the task performed by CPU that decides the way processes would be executed.

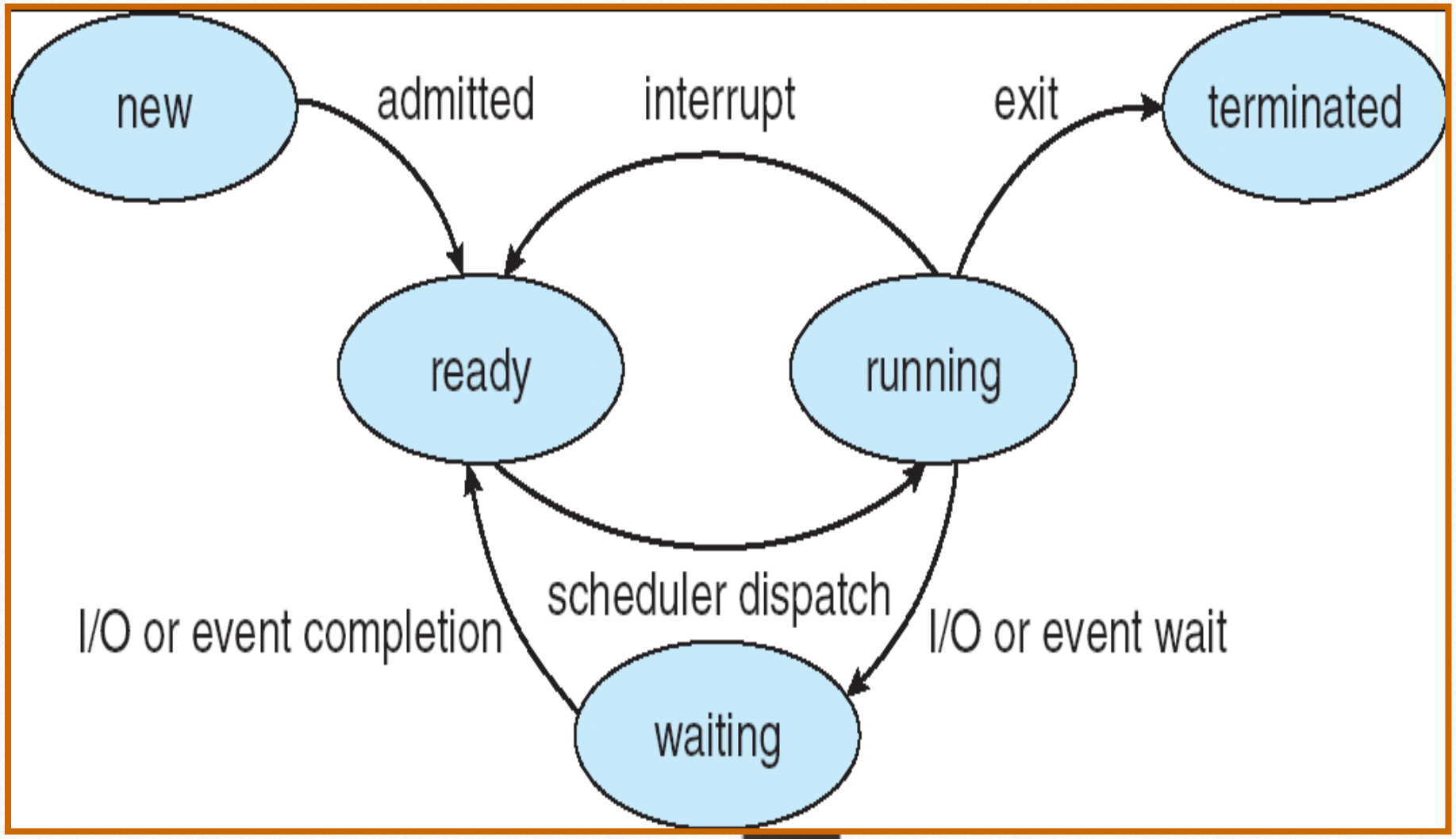
Process States



Scheduling

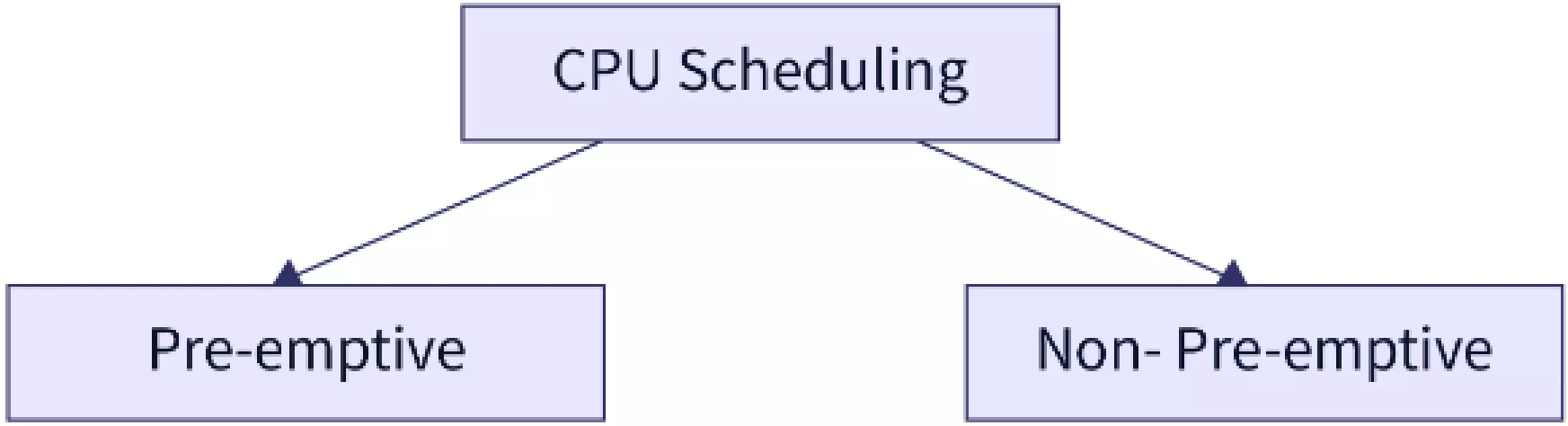
- Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types –
- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Process States



Comparison among Scheduler

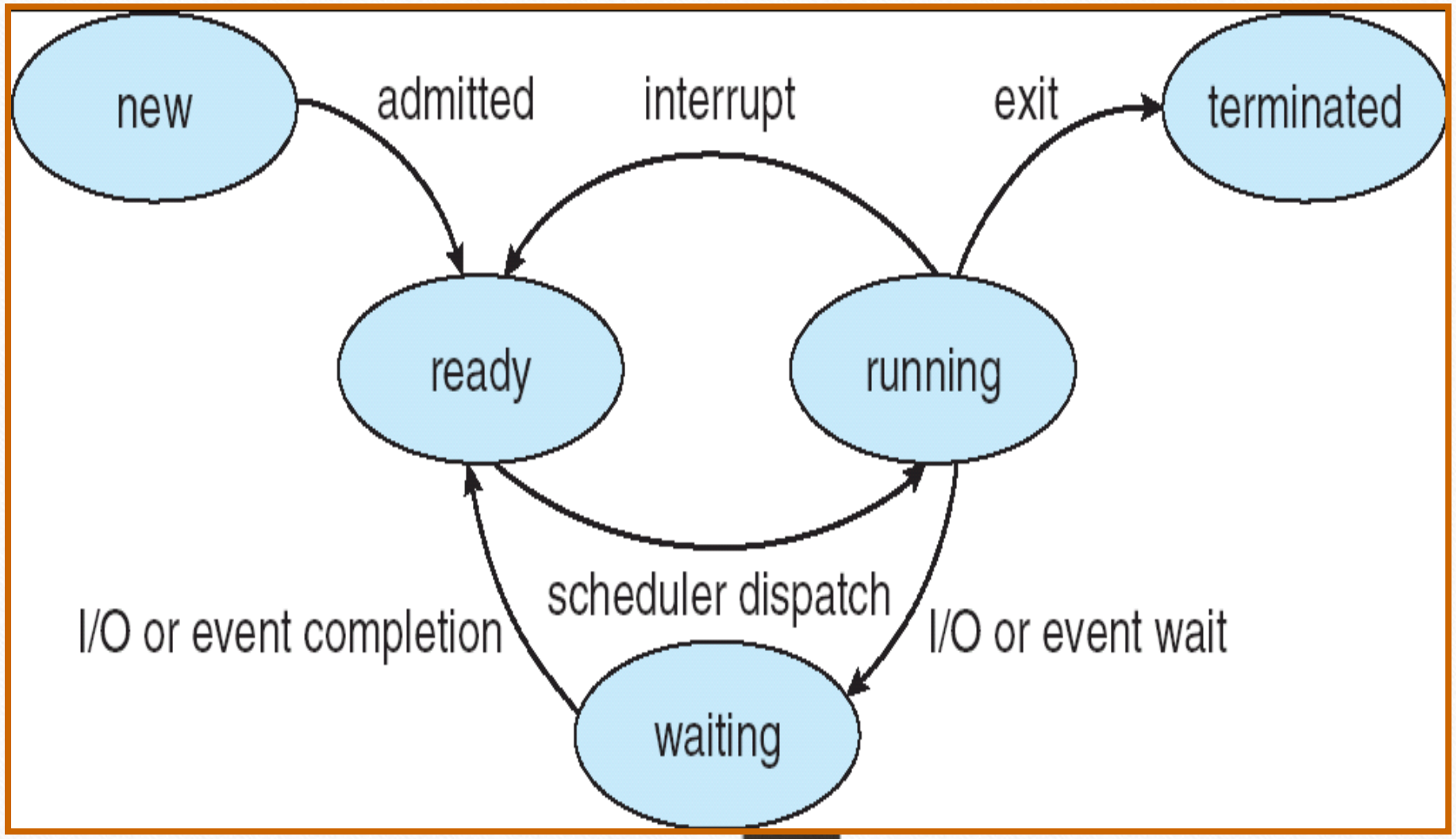
S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.


CPU Scheduling
Pre-emptive
Non- Pre-emptive

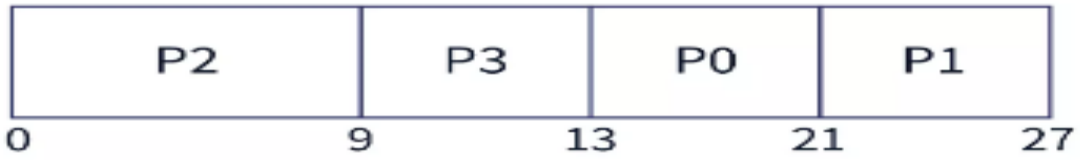
Types of scheduling

- There are two types of CPU scheduling:
  - Preemptive
    - CPU resources are allocated to a process for only a limited period of time and then those resources are taken back. A running process could be interrupted to execute a higher priority process.
  - Non-preemptive
    - New processes are executed only after the current executing process has completed its execution.

Process States



Process	Arrival Time	CPU Burst Time (in millisecond)
P0	2	8
P1	3	6
P2	0	9
P3	1	4



Scheduling algorithms

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

Scheduling Terminologies

1. **Arrival time:** Arrival time (AT) is the time at which a process arrives at the ready queue.
2. **Burst Time:** It is the time required by the CPU to complete the execution of a process, or the amount of **time required** for the execution of a process. It is also sometimes called the execution time or running time.
3. **Completion Time:** As the name suggests, completion time is the time at which a process completes its execution. It is not to be confused with burst time.

Scheduling Terminologies

1. **Turn-Around Time:** TAT, turn around time is simply the difference between completion time and arrival time **(Completion time - arrival time).**
2. **Waiting Time:** Waiting time (WT) of a process is the difference between turn around time and burst time (**TAT - BT),** i.e. the amount of time a process waits for getting CPU resources in the ready queue.
3. **Response Time:** Response time (RT) of a process is the time after which any process gets CPU resources allocated after entering the ready queue.

CPU scheduling has criteria that it uses to schedule processes in the most efficient manner:

- CPU utilization (maximize)
- Throughput (maximize)
- Turn Around Time (minimize)
- Waiting Time (minimize)
- Response Time (minimize)