PROCESS MANAGEMENT (OS)

PROCESS MANAGEMENT

Process Definition:

A process is a program in execution. For example, when we write a program in C or C++ and compile it, the compiler creates binary code. The original code and binary code are both programs. When we actually run the binary code, it becomes a process.

PROCESS MANAGEMENT

Difference between Program and Process

- A process is an ‘active’ entity instead of a program, which is considered a ‘passive’ entity.
- A single program can create many processes when run multiple times; for example, when we open a .exe or binary file multiple times, multiple instances begin (multiple processes are created).
- Process is created then Memory is allocated and PCB is created to keep various data and pointers related to addresses.

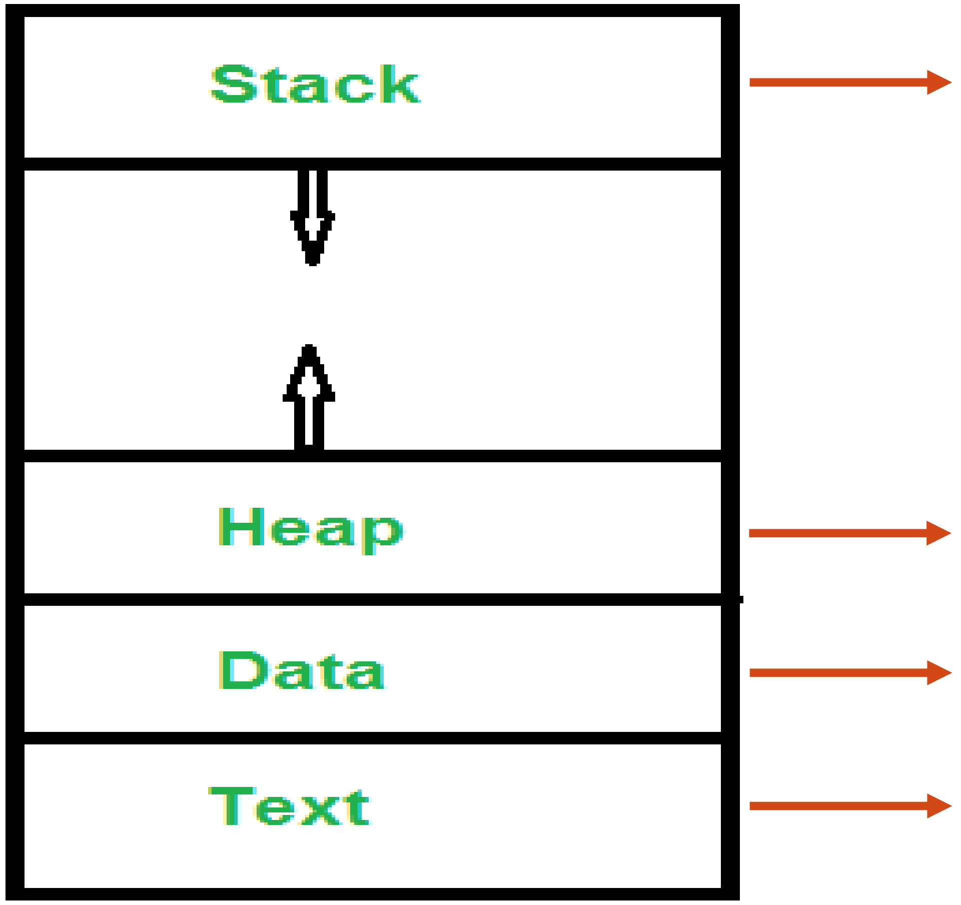
PROCESS MANAGEMENT

Process Management Tasks

- Create a process or child process
- Ready the process
- Block a process
- Suspend a process
- Resume a process
- Delay a process
- Terminate a process
- Wait for a child process to Terminate
- Change the priority of a process

PROCESS MANAGEMENT

Memory Layout of Process



The stack contains temporary data, such as function parameters, returns addresses, and local variables.

Dynamically allocated memory to process during its run time.

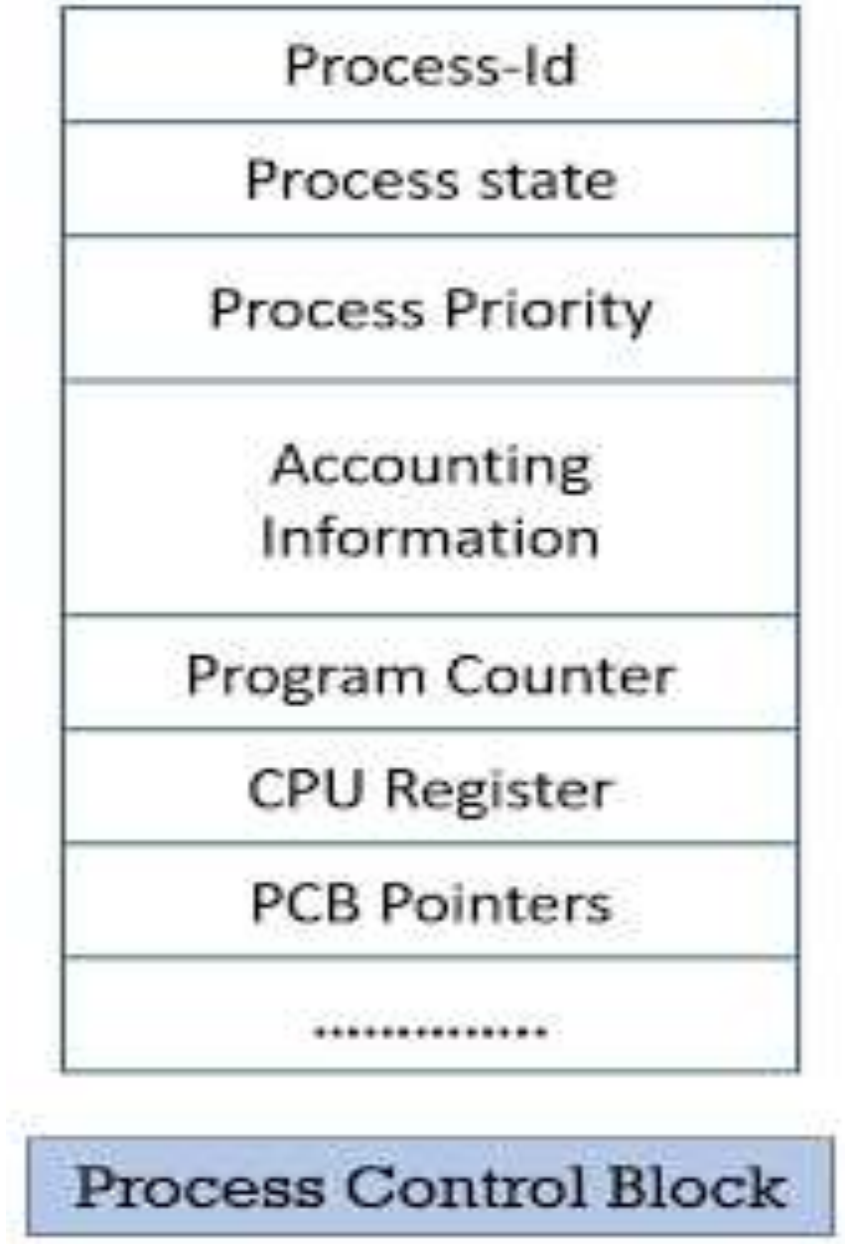
Global Data

Contains binary code

PROCESS MANAGEMENT

PCB (Process Control Block)

A process control block (PCB) contains information about the process, i.e. registers, quantum, priority, etc. The process table is an array of PCBs, that means logically contains a PCB for all of the current processes in the system.



PROCESS MANAGEMENT

PCB (Process Control Block)

1. **Pointer:** It is a stack pointer that is required to be saved when the process is switched from one state to another to retain the current position of the process.
2. **Process state:** It stores the respective state of the process.
3. **Process priority:** Every process is assigned a priority according to which it is scheduled for execution
4. **Accounting Info:** Tracks resource usage for monitoring and optimization, including CPU time, memory usage, and I/O activities.
5. **Program counter:** It stores the counter,**:** which contains the address of the next instruction that is to be executed for the process.

5. **Register:** These are the CPU registers which include the accumulator, base, registers, and general-purpose registers.
6. **Memory limits:** This field contains the information about memory management system used by the operating system. This may include page tables, segment tables, etc.
7. **Open files list :** This information includes the list of files opened for a process.

PROCESS MANAGEMENT

PCB (Process Control Block) Need ....

- **Process Management:** PCBs are used to represent and manage each process or thread in the system. They store essential information about the process, such as its program counter, stack pointer, registers, and state (e.g., running, ready, or blocked).
- **Context Switching:** The operating system relies on PCBs to perform context switching, which is the process of saving the current state of a running process and loading the state of another process that's ready to run. PCBs contain the information needed for these context switches, making multitasking possible.
- **Process Scheduling:** PCBs store information related to process priorities, resource requirements, and execution history. This information is crucial for the operating system's scheduler to determine the next process to run. It helps in achieving fair allocation of CPU time among processes and threads.
- **Resource Management:** PCBs maintain information about the resources a process holds or needs. This includes data on open files, allocated memory, CPU time used, and more. This data helps the operating system ensure that processes operate within their resource limits and avoid resource conflicts.

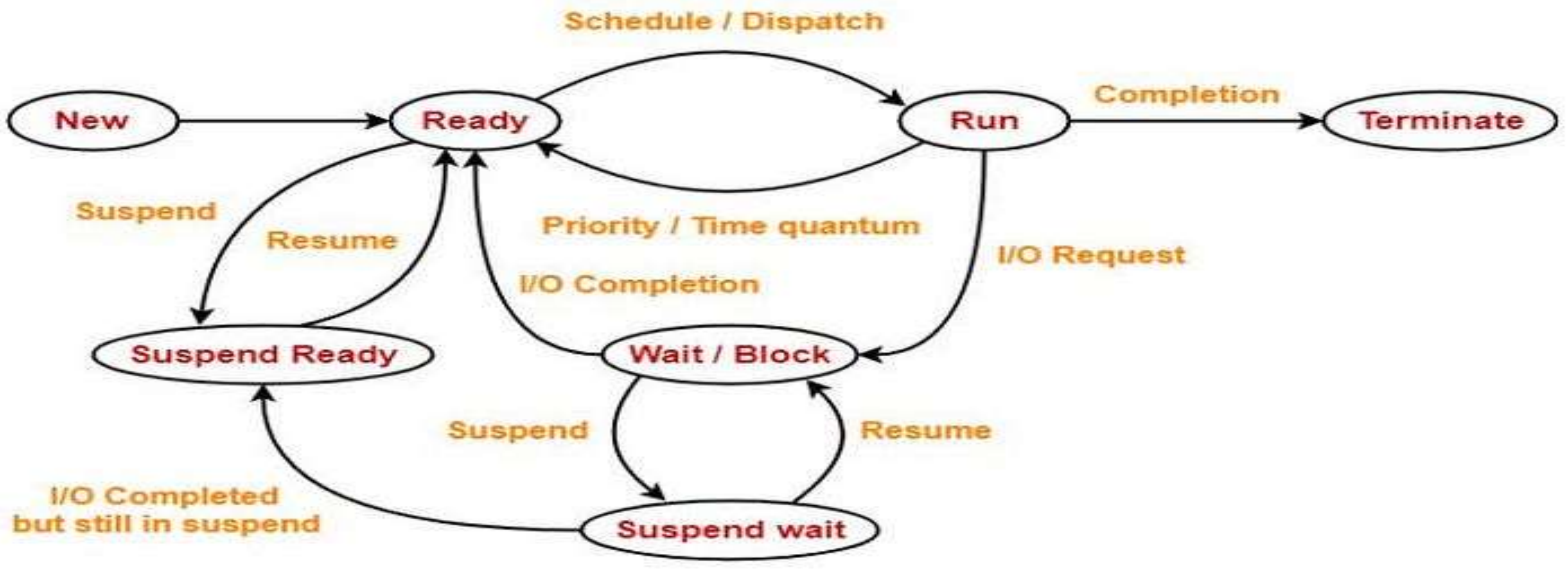
PROCESS MANAGEMENT

PCB (Process Control Block) Need ………

- **Inter-Process Communication:** PCBs can include fields for managing inter-process communication, like message queues, semaphores, and signals. This information is necessary for processes to communicate and synchronize with one another.
- **Fault Handling:** When a process encounters an error or exception (e.g., a divide-by-zero error), the PCB stores information about the error, which is vital for debugging and potentially recovering from the fault.
- **Process Termination:** When a process completes its execution or is terminated by the operating system or another process, the PCB is responsible for releasing resources associated with the process, such as memory, file handles, and any other allocated resources.
- **Accounting and Monitoring:** PCBs often contain fields for monitoring and accounting purposes, such as process creation and termination timestamps, the amount of CPU time consumed, and other statistics. These details are used for performance analysis and system monitoring.
- **Security and Permissions:** PCBs may contain information about the security attributes and permissions associated with a process, ensuring that processes adhere to access control

PROCESS MANAGEMENT

Process State diagram



Process State Diagram

PROCESS MANAGEMENT

Process States in Operating System

The states of a process are as follows:

- **New (Create):** In this step, the process is about to be created but not yet created. It is the program that is present in secondary memory that will be picked up by OS to create the process.
- **Ready:** New -> Ready to run. After the creation of a process, the process enters the ready state i.e. the process is loaded into the main memory. The process here is ready to run and is waiting to get the CPU time for its execution. Processes that are ready for execution by the CPU are maintained in a queue called ready queue for ready processes.
- **Run:** The process is chosen from the ready queue by the CPU for execution and the instructions within the process are executed by any one of the available CPU cores.
- **Blocked or Wait:** Whenever the process requests access to I/O or needs input from the user or needs access to a critical region(the lock for which is already acquired) it enters the blocked or waits for the state. The process continues to wait in the main memory and does not require CPU. Once the I/O operation is completed the process goes to the ready state.
- **Terminated or Completed:** Process is killed as well as PCB is deleted. The resources allocated to the process will be released or deallocated.
- **Suspend Ready:** Process that was initially in the ready state but was swapped out of main memory(refer to Virtual Memory topic) and placed onto external storage by the scheduler is said to be in suspend ready state. The process will transition back to a ready state whenever the process is again brought onto the main memory.
- **Suspend wait or suspend blocked:** Similar to suspend ready but uses the process which was performing I/O operation and lack of main memory caused them to move to secondary memory. When work is finished it may go to suspend ready

PROCESS MANAGEMENT

Types of Process:

CPU bound processes: If the process is intensive in terms of CPU operations, then it is called CPU bound process.

I/o bound processes : If the process is intensive in terms of I/O operations then it is called I/O bound process.

PROCESS MANAGEMENT

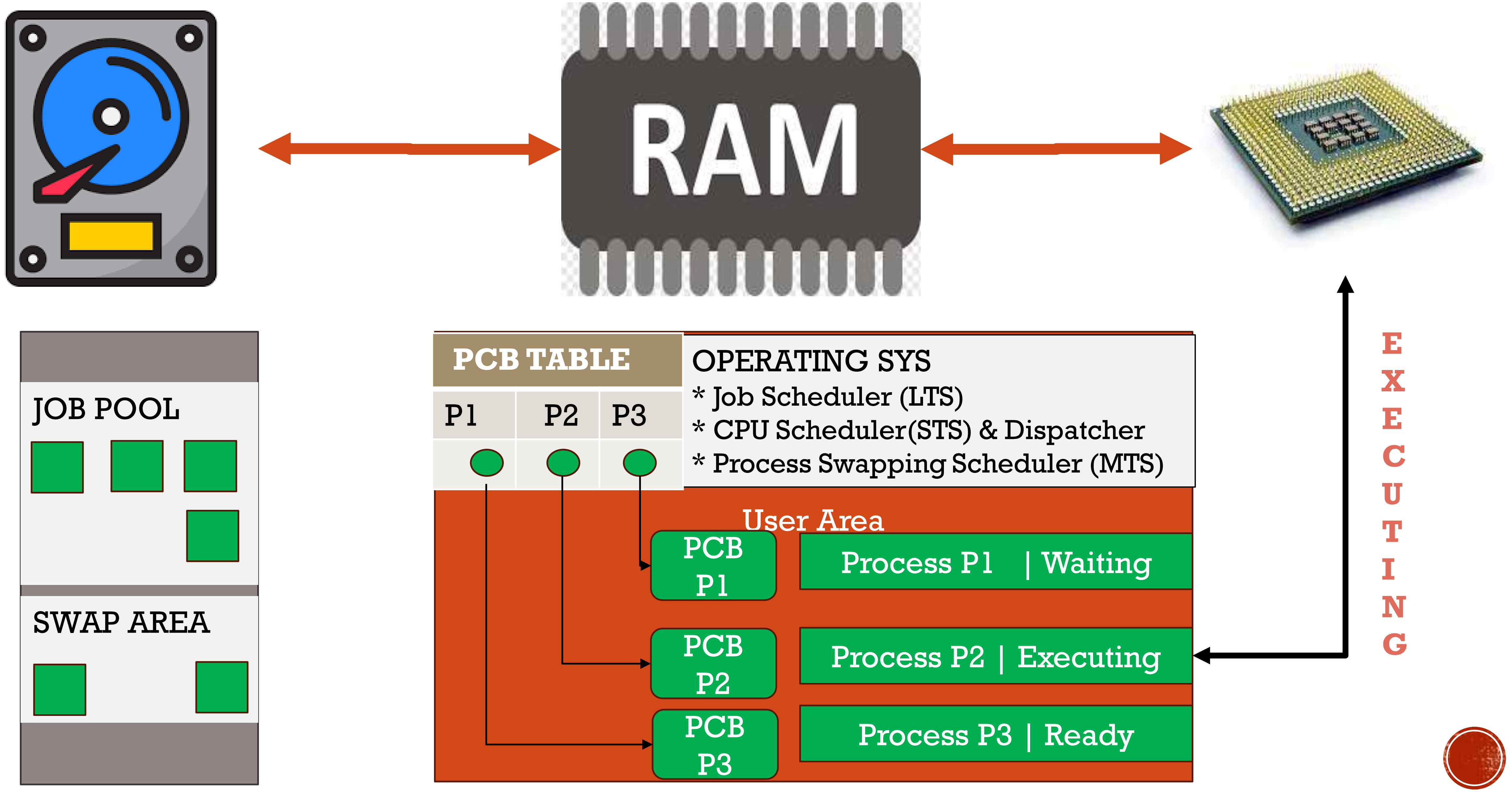
Types of Schedulers

- **Long-term – performance:** Decides how many processes should be made to stay in the ready state. This decides the degree of multiprogramming. Once a decision is taken it lasts for a long time which also indicates that it runs infrequently. Hence it is called a long-term scheduler.

- **Short-term – Context switching time:** Short-term scheduler will decide which process is to be executed next and then it will call the dispatcher. A dispatcher is a software that moves the process from ready to run and vice versa. In other words, it is context switching. It runs frequently. Short-term scheduler is also called CPU scheduler.

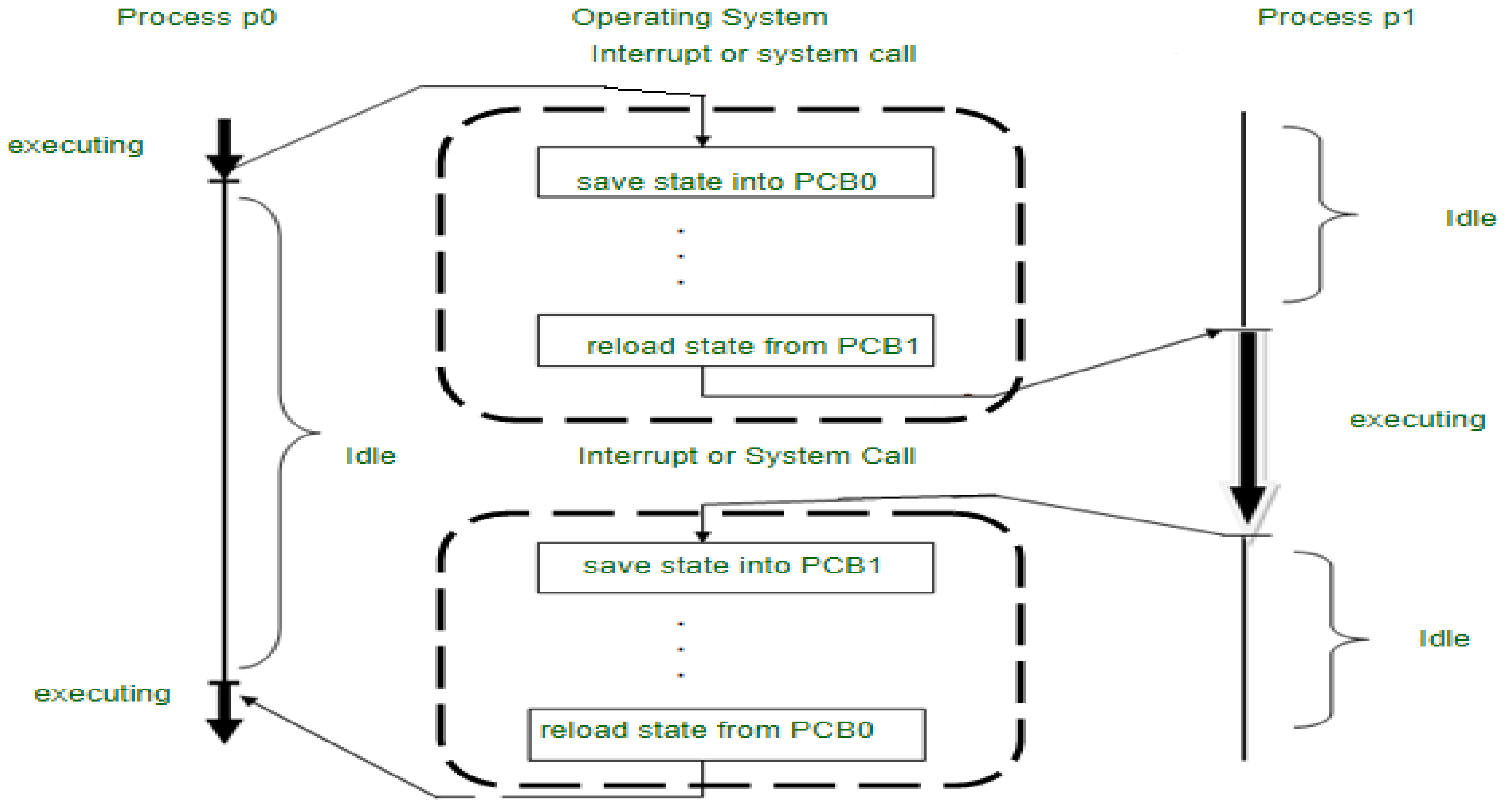
- **Medium-term – Swapping time:** Suspension decision is taken by the medium-term scheduler. The medium-term scheduler is used for swapping which is moving the process from main memory to secondary and vice versa. The swapping is done to reduce degree of multiprogramming.

PROCESS MANAGEMENT



PROCESS MANAGEMENT

CONTEXT SWITCHING



CPU SCHEDULING

- Definition

CPU Scheduling is a process that allows one process to use the CPU while another process is delayed (in standby) due to unavailability of any resources such as I/O etc., thus making full use of the CPU. The purpose of CPU Scheduling is to make the system more efficient, faster, and fairer.

CPU SCHEDULING

CPU Scheduling Algorithms:

- **First Come First Serve (FCFS)**
- **Shortest Job First (SJF) [Non Preemptive]**
- **Shortest Remaining Time First [Preemptive]**
- **Priority Scheduling [Preemptive & Non Preemptive]**
- **Round robin**

CPU SCHEDULING

Criteria for selecting CPU Scheduling Algorithms

- **CPU utilization : CPU utilization** can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the load upon the system.

- **Throughput :** number of processes being executed and completed per unit of time is called **throughput.**

- **Turnaround Time : Turn-around time** is the sum of times spent waiting to get into memory, waiting in the ready queue, executing in CPU, and waiting for I/O.

```
Turn Around Time = Completion Time - Arrival Time
```

- **Waiting Time :** Time spent by process in ready queue

```
Waiting Time = Turnaround Time - Burst Time.
```

- **Response Time:** time taken from submission of the process of the request until the first response is produced. This measure is called **response time.**

```
Response Time = CPU Allocation Time(when the CPU was allocated for the first) - Arrival Time
```

CPU SCHEDULING

First Come First Serve (FCFS)

First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first and is implemented by using FIFO queue.

- **Advantages of FCFS:**
- Easy to implement
- First come, first serve method

- **Disadvantages of FCFS:**
- FCFS suffers from **Convoy effect**.
- The average waiting time is much higher than the other algorithms.

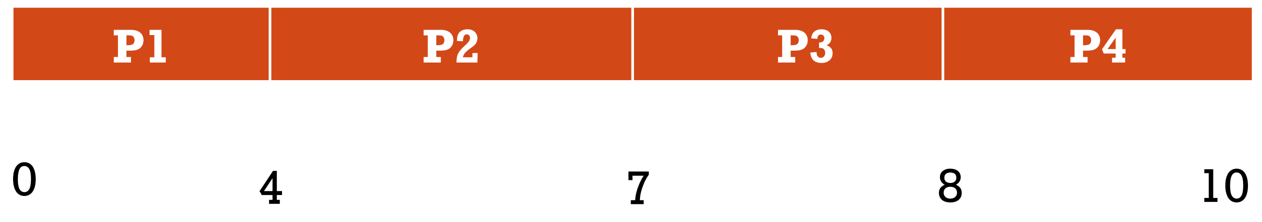
CPU SCHEDULING

Example: First Come First Serve (FCFS)

Consider the following table of arrival time and burst time for five processes P1, P2, P3, P4 and P5.

Processes	Arrival Time	Burst Time
P1	0	4
P2	1	3
P3	2	1
P4	3	2

Gantt Chart Algorithm



P1 = 0 – 0 = 0
P2 = 4 – 1 = 3
P3 = 7 – 2 = 5
P4 = 8 – 3 = 5

Average Waiting time= (0+3+5+5)/4= 3.25

CPU SCHEDULING

Shortest Job First (SJF) [Non Preemptive]

Shortest job first (SJF) is a scheduling process that selects the waiting process with the smallest execution time to execute next.

Advantages of Shortest Job first:

- As SJF reduces the average waiting time thus, it is better than the first come first serve scheduling algorithm.

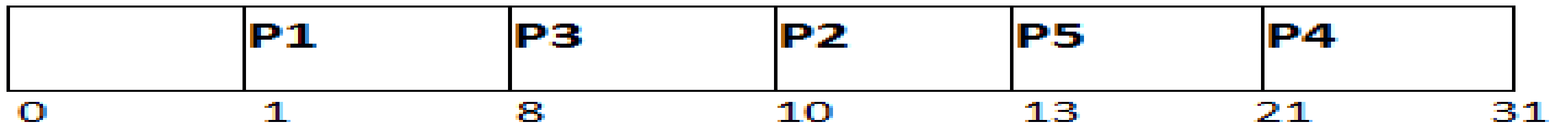
Disadvantages of SJF:

- One of the demerit SJF has is starvation.

CPU SCHEDULING

Example: Shortest Job First (SJF) [Non Preemptive]

PID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	1	7	8	7	0
2	3	3	13	10	7
3	6	2	10	4	2
4	7	10	31	24	14
5	9	8	21	12	4



CPU SCHEDULING

❖**Shortest Remaining Time First [Preemptive]**

Shortest remaining time first is the preemptive version of the Shortest job first. In SRTF the process with the smallest amount of time remaining until completion is selected to execute.

Advantages of SRTF:

- In SRTF the short processes are handled very fast.
- The system also requires very little overhead since it only makes a decision when a process completes or a new process is added.

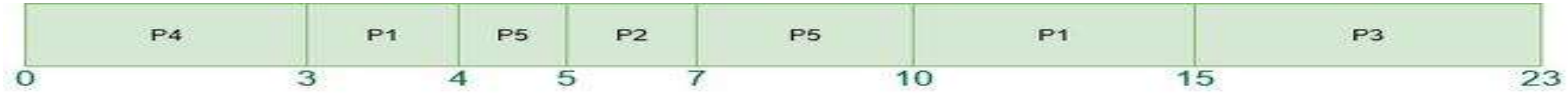
Disadvantages of SRTF:

- Like the shortest job first, it also has the potential for process starvation.

CPU SCHEDULING

- **Example: Shortest Remaining Time First [Preemptive]**

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



CPU SCHEDULING

❖ **Example: Shortest Remaining Time First [Preemptive]**

- **Turn Around time** = Completion time – arrival time
- **Waiting Time** = Turn around time – burst time

Process	Completion Time	Turn Around Time	Waiting Time
P1	15	15-2 = 13	13-6 = 7
P2	7	7-5 = 2	2-2 = 0
P3	23	23-1 = 22	22-8 = 14
P4	3	3-0 = 3	3-3 = 0
P5	10	10-4 = 6	6-4 = 2

•**Average Turn around time** = (13 + 2 + 22 + 3 + 6)/5 = 9.2
•**Average waiting time** = (7 + 0 + 14 + 0 + 2)/5 = 23/5 = 4.6

CPU SCHEDULING

❖Priority Scheduling [Preemptive and Non Preemptive]

Preemptive Priority CPU Scheduling Algorithm is a pre-emptive method of CPU scheduling algorithm that works based on the priority of a process. Each process is assigned a priority by OS. In the case of any conflict, that is, where there are more than one processor with equal value, then the most important CPU planning algorithm works on the basis of the FCFS (First Come First Serve) algorithm.

Advantages of Priority Scheduling:

- The average waiting time is less than FCFS
- Less complex

Disadvantages of Priority Scheduling:

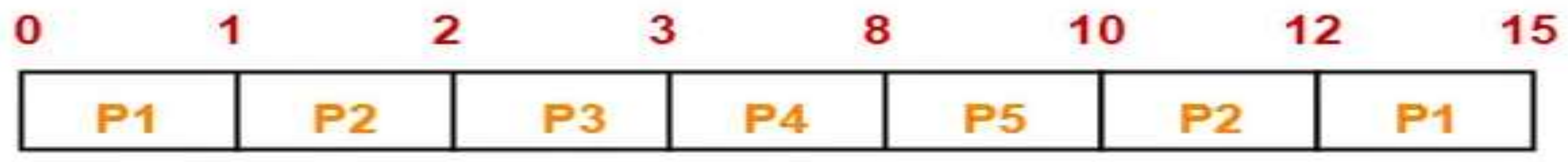
- One of the most common demerits of the Preemptive priority CPU scheduling algorithm is the Starvation Problem.

CPU SCHEDULING

Example: Priority Scheduling [Preemptive]

Consider the set of 5 processes whose arrival time and burst time are given below-
If the CPU scheduling policy is priority preemptive, calculate the average waiting time and average turn around time. (Higher number represents higher priority)

Process Id	Arrival time	Burst time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5



Gantt Chart

CPU SCHEDULING

❖**Round robin**

Round Robin is a CPU scheduling algorithm where each process is cyclically assigned a fixed time slot **(Time Slice/Time Quantum)**

Advantages of Round robin:

- Round robin seems to be fair as every process gets an equal share of CPU.
- The newly created process is added to the end of the ready queue.

Disadvantage of Round Robin:

- Spends more time on context switching.
- Performance depends on time quantum.
- Processes don't have priorities.

CPU SCHEDULING

Example: Round robin

Consider the following table of arrival time and burst time for four processes **P1, P2, P3, and P4** and given **Time Quantum = 2**

Process	**Burst Time**	**Arrival Time**
P1	5 ms	0 ms
P2	4 ms	1 ms
P3	2 ms	2 ms
P4	1 ms	4 ms

CPU SCHEDULING

Example: Round robin

- **Completion Time:** Time at which process completes its execution.
- **Turn Around Time:** Time Difference between completion time and arrival time.

Turn Around Time = Completion Time – Arrival Time

- **Waiting Time(W.T):** Time Difference between turn around time and burst time.

Waiting Time = Turn Around Time – Burst Time

Processes	AT	BT	CT	TAT	WT
P1	0	5	12	12-0 = 12	12-5 = 7
P2	1	4	11	11-1 = 10	10-4 = 6
P3	2	2	6	6-2 = 4	4-2 = 2
P4	4	1	9	9-4 = 5	5-1 = 4

- ***Average Turn around time*** *= (12 + 10 + 4 + 5)/4 = 31/4 = 7.7*
- ***Average waiting time*** *= (7 + 6 + 2 + 4)/4 = 19/4 = 4.7*

PROCESS MANAGEMENT
INTER PROCESS COMMUNICATION (IPC)

IPC

A system can have two types of processes i.e. independent or cooperating. Cooperating processes affect each other and may share data and information among themselves.

Inter-Process Communication or IPC provides a mechanism to exchange data and information across multiple processes, which might be on single or multiple computers connected by a network. This is essential for many tasks, such as:

- Sharing data
- Coordinating activities
- Managing resources
- Achieving modularity

APPROACHES FOR INTER PROCESS COMMUNICATION

- ❖ SHARED MEMORY
- ❖ MESSAGE QUEUE
- ❖ MESSAGE PASSING
- ❖ DIRECT COMMUNICATION
- ❖ INDIRECT COMMUNICATION
- ❖ PIPES
- ❖ FIFO

Examples of IPC systems

1. Posix: uses shared memory method.
2. Mach : uses message passing
3. Windows XP : uses message passing using local procedural calls

Communication in client/server Architecture:

There are various mechanism:

•Pipe
•Socket
•Remote Procedural calls (RPCs)

APPROACHES FOR INTER PROCESS COMMUNICATION

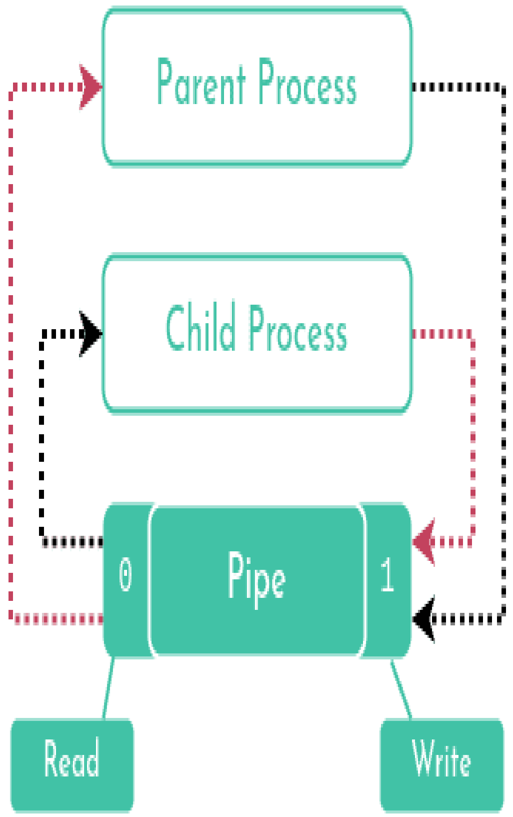
PIPE METHOD:

A **pipe** is a section of shared memory meant to facilitate the communication between processes. It is a unidirectional channel: a pipe has a read end and a write end. So a process can write to the write end of the pipe. The data is then stored in a memory buffer until it is read by another process from the pipe's read end.

A pipe is a sort of file, stored outside of the file system, that has no name or any other particular attribute. But we can handle it like a file thanks to its **two file descriptors**.

Let's keep in mind that there is a limit to a pipe's size, which varies depending on the operating system.

APPROACHES FOR INTER PROCESS COMMUNICATION



Creating a Pipe

We can create a pipe with the aptly-named `pipe` system call. Here is its prototype in the `<unistd.h>` library :

```
int pipe(int pipefd[2]);
```

As its only parameter, `pipe` takes an array of two integers where the two file descriptors should be stored. Of course, these file descriptors represent the pipe's two ends:

- `pipefd[0]`: the **read** end
- `pipefd[1]`: the **write** end

There are two points to keep in mind:

- If a process attempts to read from an empty pipe, `read` will remain blocked until data is written to it.
- Inversely, if a process tries to write to a full pipe (one that has reached its size limit), `write` will remain blocked until enough data has been read to allow the write operation to complete.

APPROACHES FOR INTER PROCESS COMMUNICATION

PROBLEMS DUE TO PROCESS COMMUNICATION

- ✓ **Race Conditions:** When multiple processes or threads access shared resources simultaneously, their execution order can lead to unpredictable outcomes and data corruption.

- ✓ **Data Inconsistency:** Inconsistent data can arise when one process modifies shared data while another is using it, leading to incorrect results.

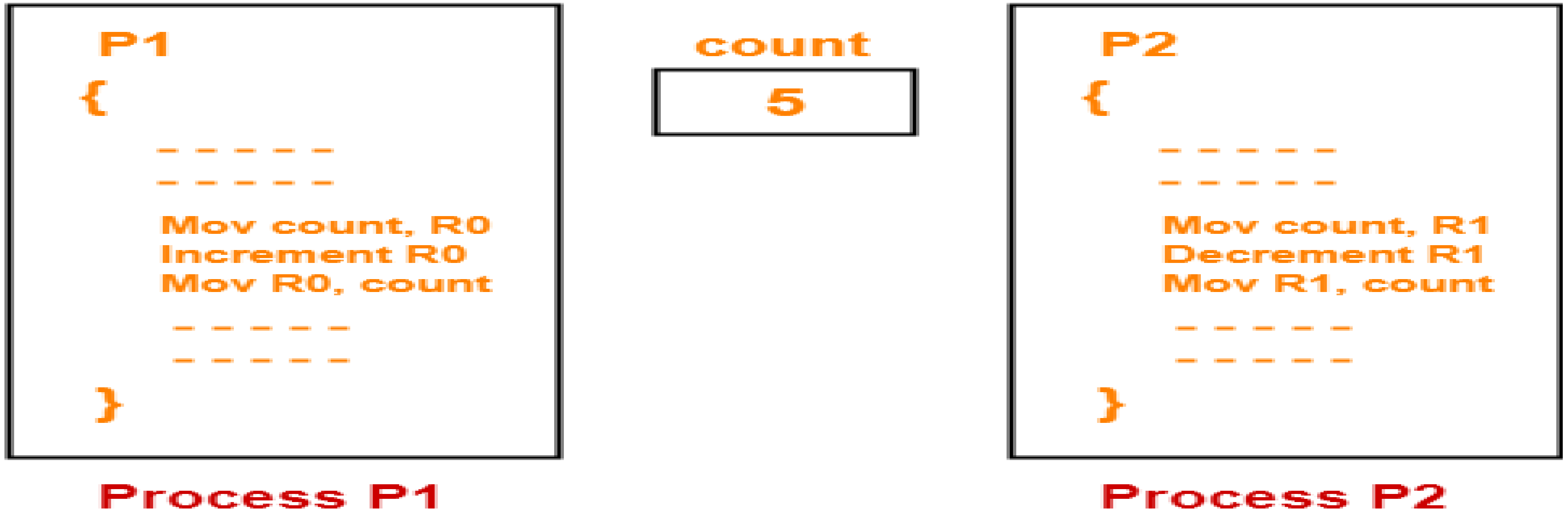
- ✓ **Deadlocks:** A deadlock occurs when multiple processes are waiting indefinitely for resources held by others, causing a standstill in execution.

- ✓ **Starvation:** If a process consistently gets preempted by other processes, it might never get a chance to access the critical section, leading to starvation.

PROCESS SYNCHRONIZATION

When more than one process is executing the same code or accessing the same memory or any shared variable in that condition there is a possibility that the output or the value of the shared variable may get wrong because outcome depends on the particular order in which the access takes place. This situation is called **Race Condition**

Example:



PROCESS SYNCHRONIZATION

The **critical section** refers to the segment of code where processes access shared resources, such as common variables and files, and perform write operations on them.

Solutions to the critical section problem:

Any solution to the critical section problem must satisfy the following requirements:

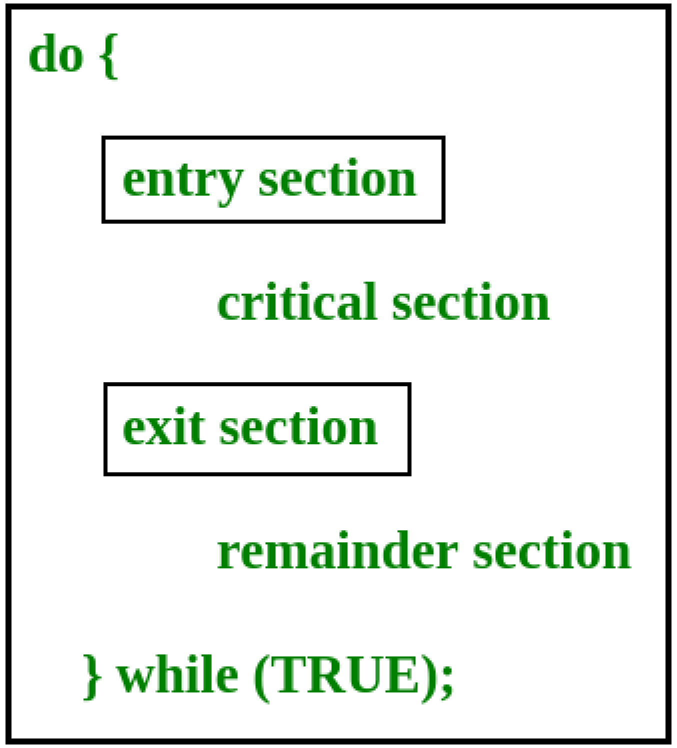
Mutual exclusion: When one process is executing in its critical section, no other process is allowed to execute in its critical section.

Progress: When no process is executing in its critical section, and there exists a process that wishes to enter its critical section, it should not have to wait indefinitely to enter it.

Bounded waiting: There must be a bound on the number of times a process is allowed to execute in its critical section, after another process has requested to enter its critical section and before that request is accepted.

PROCESS SYNCHRONIZATION

Generalized Structure of Process considered for Critical Section Problem Solution:



```
do {

    entry section

            critical section

    exit section

            remainder section

   } while (TRUE);
```

PROCESS SYNCHRONIZATION

SOLUTIONS OF CRITICAL SECTION PROBLEM:

- ✓ MUTEX
- ✓ LOCKS
- ✓ MONITORS
- ✓ SEMAPHORE

PROCESS SYNCHRONIZATION

SEMAPHORE

Semaphore in OS is a synchronization tool used to regulate access to shared resources such as files, memory, or network connections. It is essentially a variable that controls access to the shared resource.

Semaphores are integer variables, their value acts as a signal, which allows or does not allow a process to access the critical section of code or certain other

There are mainly two types of Semaphores, or two types of signaling integer variables:

- ✓ Binary Semaphores
- ✓ Counting Semaphores

PROCESS SYNCHRONIZATION

SEMAPHORE

Operations in Semaphores
Wait() and signal() are the two basic operations used to manipulate semaphores in an operating system.

- **Wait():**

When a process or thread performs a wait() operation on a semaphore, it checks the current value of the semaphore. If the value is positive, the process or thread acquires the semaphore and decrements its value. If the value is zero, the process or thread is blocked and added to a queue of waiting processes until the semaphore's value becomes positive.

Syntax of Wait():

```
    wait(S) {   while (S<=0);  S--;    }
```

- **Signal():**

When a process or thread performs a signal() operation on a semaphore, it increments the semaphore's value. If there are any processes or threads waiting on the semaphore, one of them is unblocked and allowed to acquire the semaphore.

Syntax for Wait():

```
signal(S) { S++; }
```

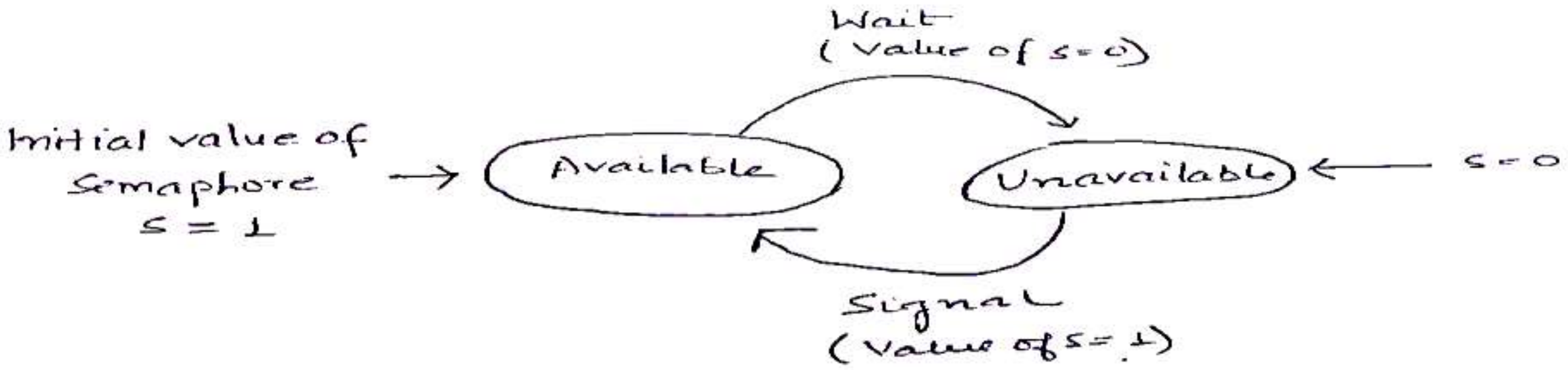
PROCESS SYNCHRONIZATION

BINARY SEMAPHORE

In these types of Semaphores the integer value of the semaphore can only be either 0 or 1. If the value of the Semaphore is 1, it means that the process can proceed to the critical section (the common section that the processes need to access). However, if the value of the binary semaphore is 0, then the process cannot continue to the critical section of the code.

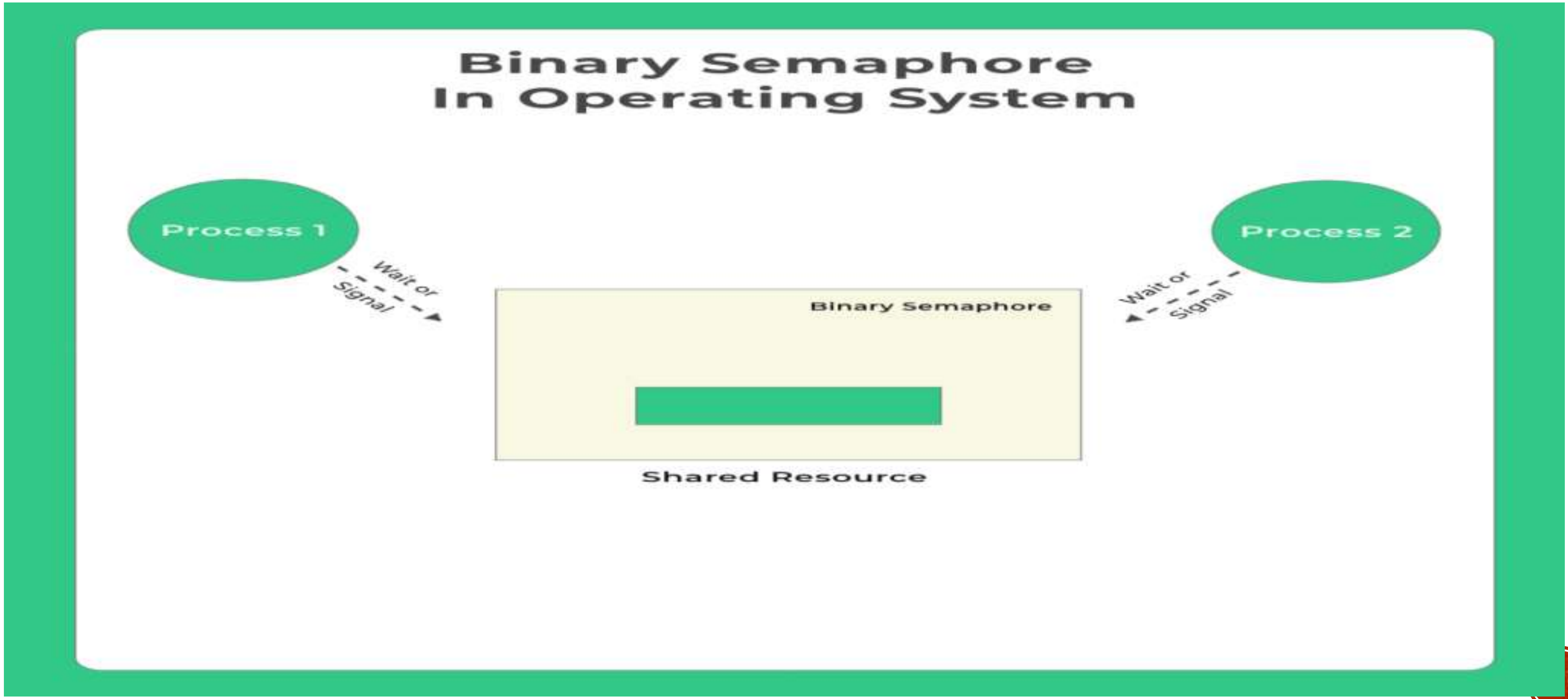
PROCESS SYNCHRONIZATION

BINARY SEMAPHORE



PROCESS SYNCHRONIZATION

BINARY SEMAPHORE



PROCESS SYNCHRONIZATION

COUNTING SEMAPHORE

Counting semaphores are signaling integers that can take on any integer value. Using these Semaphores we can coordinate access to resources and here the Semaphore count is the number of resources available. If the value of the Semaphore is anywhere above 0, processes can access the critical section or the shared resources. The number of processes that can access the resources/code is the value of the semaphore. However, if the value is 0, it means that there aren't any resources that are available or the critical section is already being accessed by a number of processes and cannot be accessed by more processes. Counting semaphores are generally used when the number of instances of a resource is more than 1, and multiple processes can access the resource.