

- Operating System (OS)



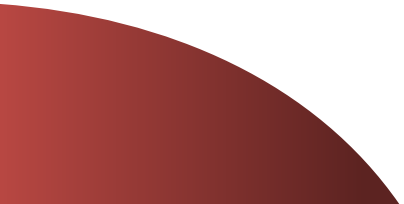
## Unit-2

# Process and Process Scheduling





# Process Concept



# What is Process?

- ▶ Process is a **program under execution**.
- ▶ Process is an **abstraction of a running program**.
- ▶ When program is ready for execution it becomes active and is known as process.
- ▶ **Program v/s Process**
  - A program is a set of instructions that are to perform a designated task, where as the process is an operation which takes the given instructions and perform the manipulations as per the code, called 'execution of instructions'.
  - Program does not compete for resources therefore is a passive entity.
  - Process compete for resources therefore is an active entity.

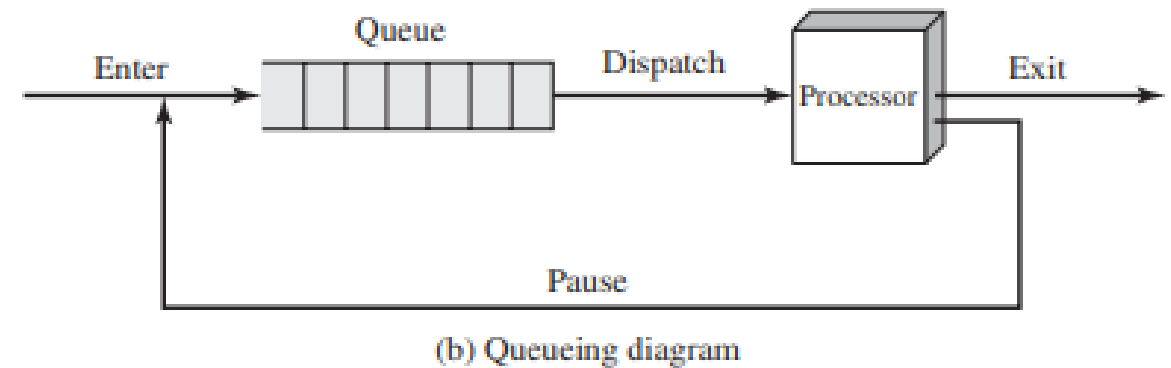
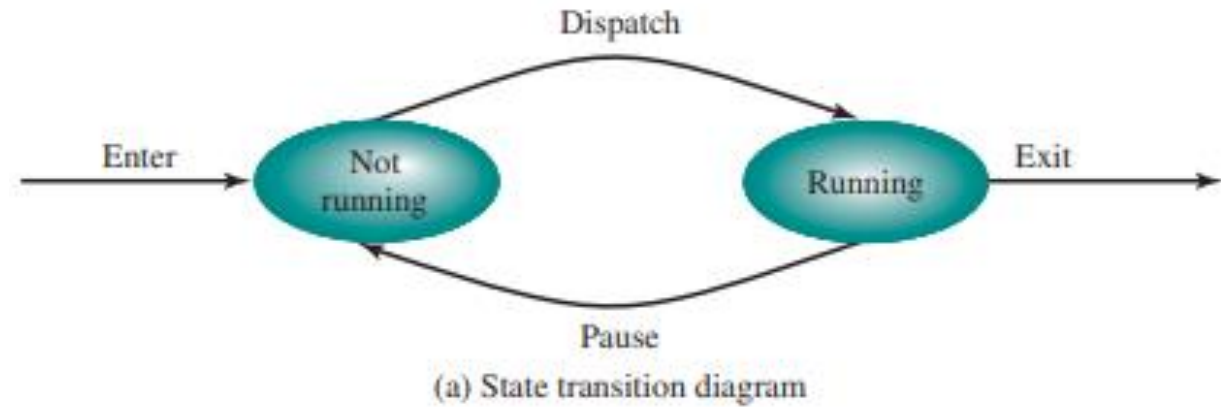


# Process States and State Transitions



# A Two-State Process Model

- In this model, a process may be in one of two states: Running or Not Running.
- When the OS creates a new process, it creates a process control block for the process and enters that process into the system in the Not Running state.
- The process exists, is known to the OS, and is waiting for an opportunity to execute.
- From time to time, the currently running process will be interrupted and the dispatcher portion of the OS will select some other process to run.
- The former process moves from the Running state to the Not Running state, and one of the other processes moves to Running State



# Process Creation and Termination

## Process Creation

- The OS builds a data structure to manage the process
- Traditionally, the OS created all processes
  - But it can be useful to let a running process create another.
- This action is called ***process spawning***
  - ***Parent Process*** is the original, creating process
  - ***Child Process*** is the new process

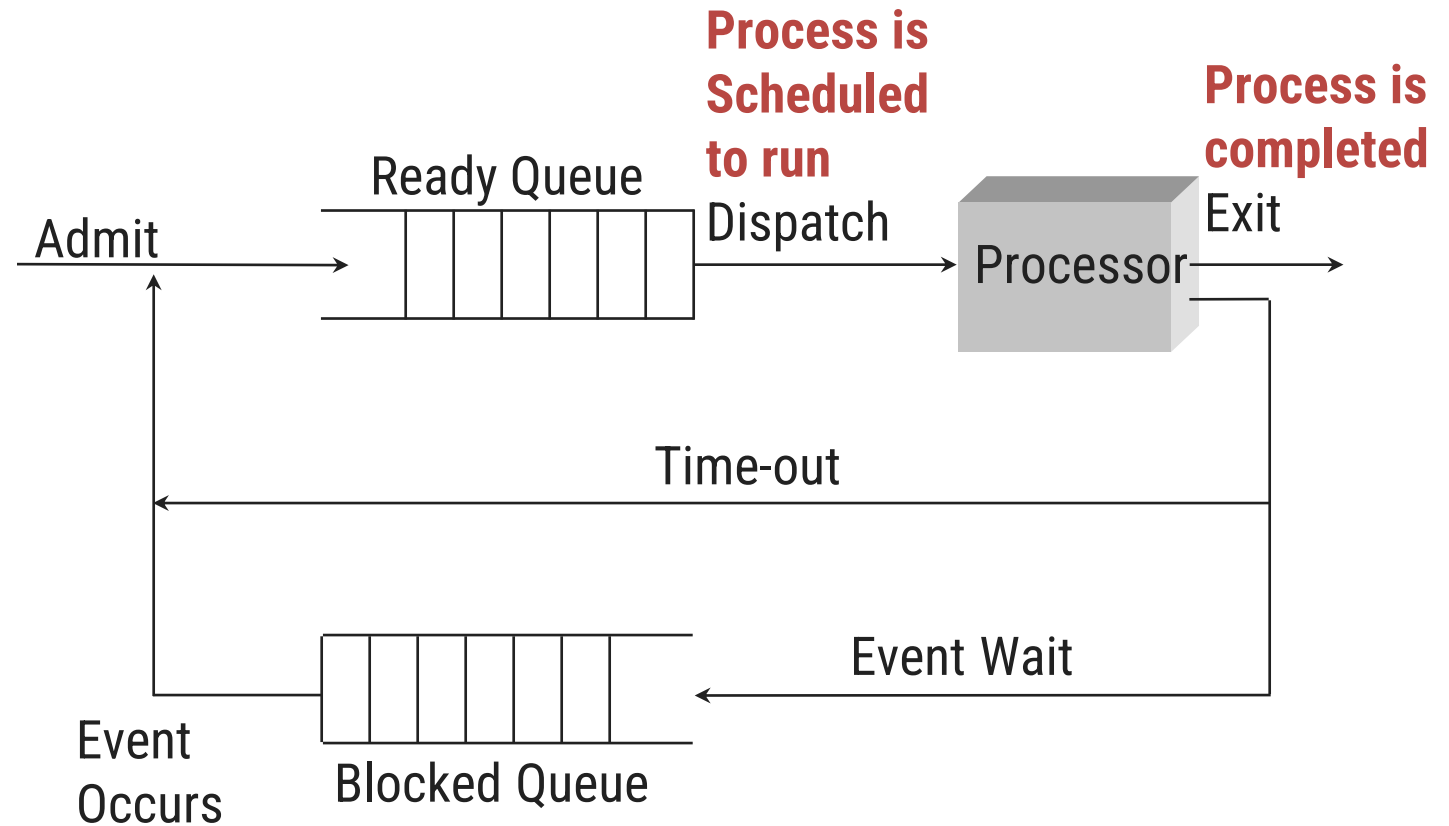
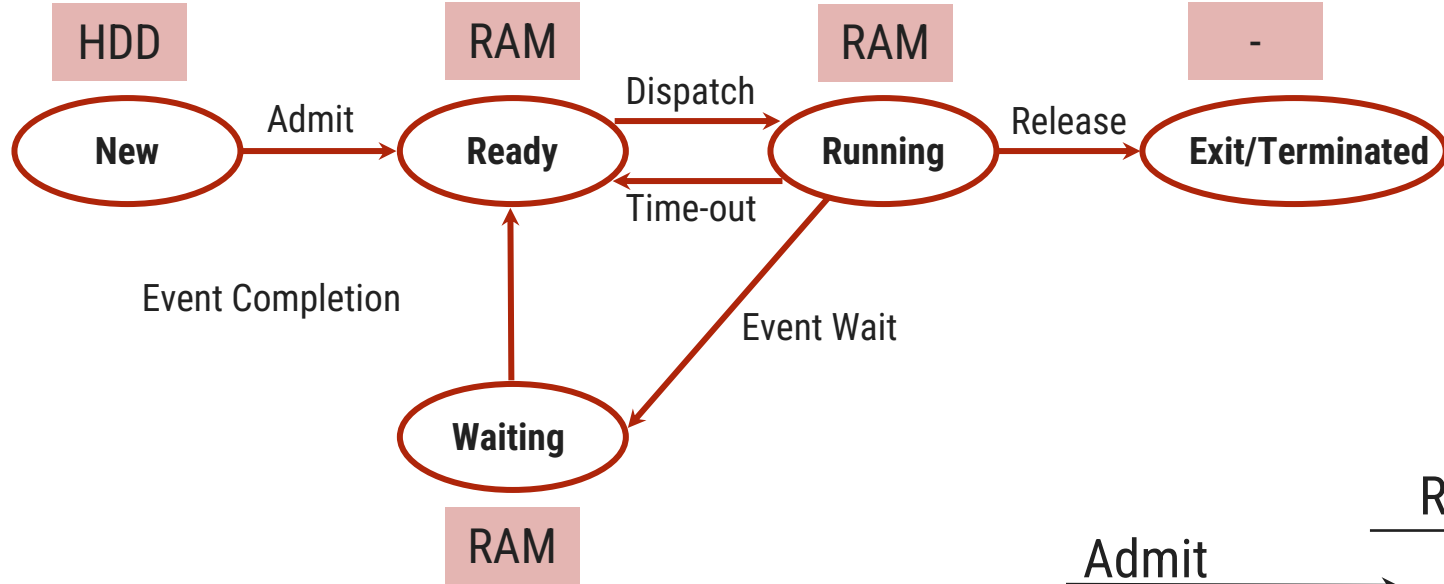
## Process Termination

- There must be some way that a process can indicate completion.
- This indication may be:
  - A HALT instruction generating an interrupt alert to the OS.
  - A user action (e.g. log off, quitting an application)
  - A fault or error
  - Parent process terminating

# Five Process State Model

- ▶ **New** – process is being **created**
  - ▶ **Ready** – process is **waiting to run (runnable)**, temporarily stopped to let another process run
  - ▶ **Running** – process is actually **using the CPU**
  - ▶ **Waiting** – **unable to run** until some external event happens
  - ▶ **Exit (Terminated)** – process has **finished the execution**
-

# Process State Transitions

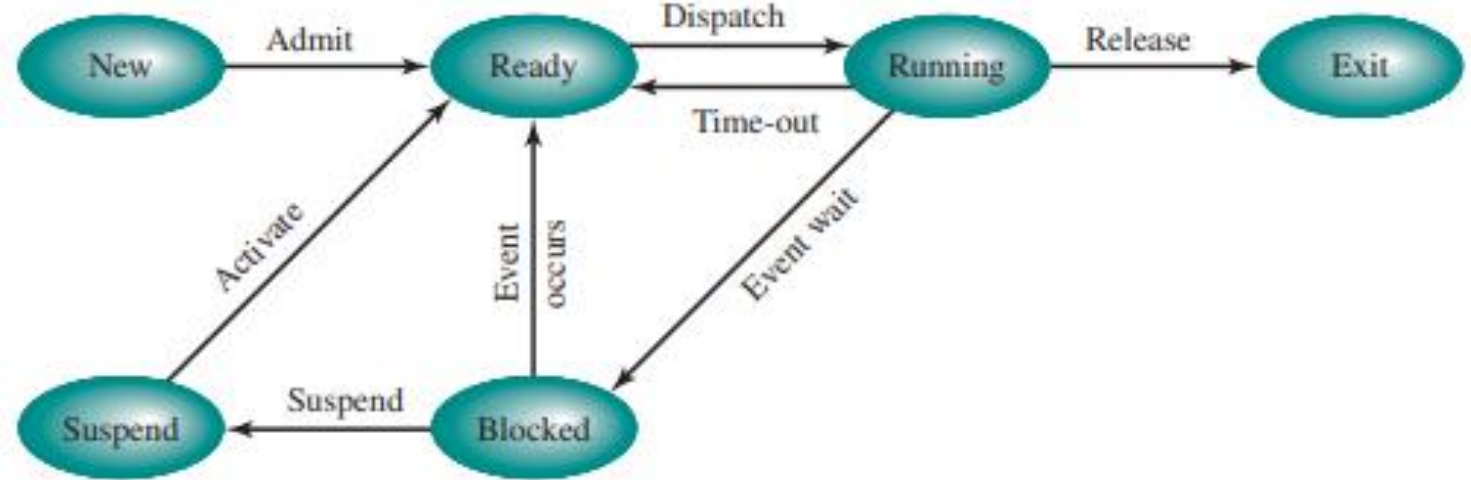




# Suspended Processes

## With one Suspend State

- Processor is faster than I/O so all processes could be waiting for I/O
  - Swap these processes to disk to free up more memory and use processor on more processes
- Blocked state becomes **suspend** state when swapped to disk.

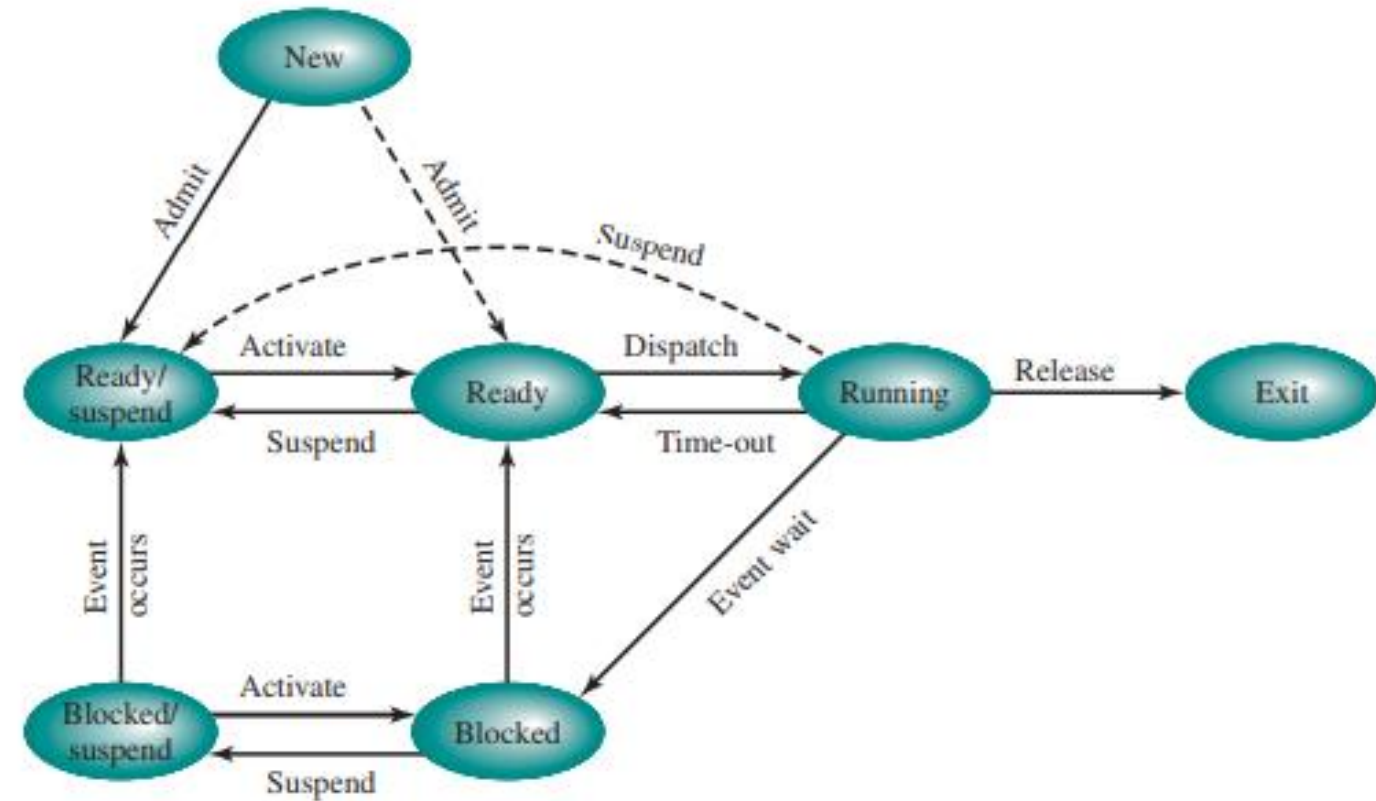


(a) With one suspend state

# Suspended Processes

## With two Suspend States

- **Ready:** The process is in main memory and available for execution
- **Blocked:** The process is in main memory and awaiting an event.
- **Blocked/Suspend:** The process is in secondary memory and awaiting an event.
- **Ready/Suspend:** The process is in secondary memory but is available for execution as soon as it is loaded into main memory.



(b) With two suspend states



# Process Control Block (PCB)

Section - 4



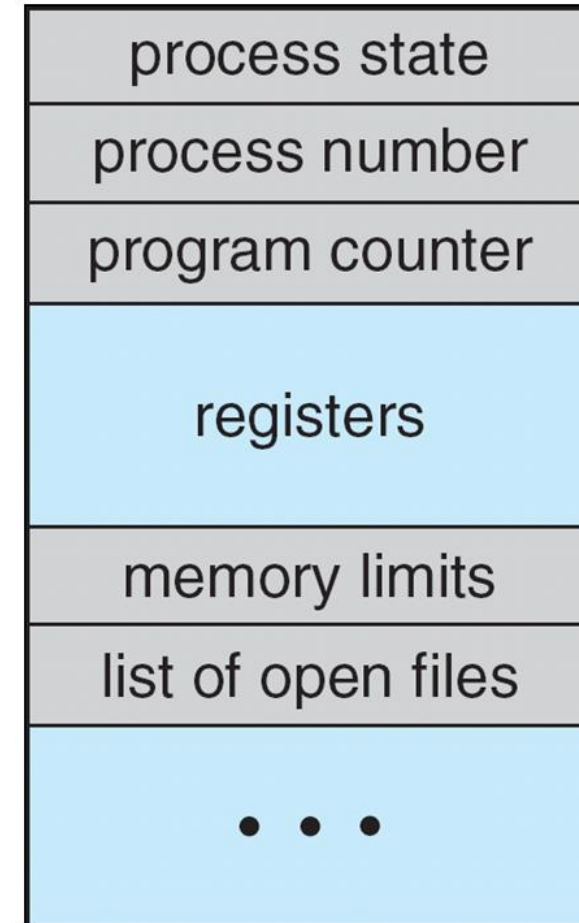
# What is Process Control Block (PCB)?

- ▶ A Process Control Block (PCB) is a **data structure maintained by the operating system for every process.**
- ▶ PCB is used for **storing the collection of information about the processes.**
- ▶ The PCB is **identified by an integer process ID (PID).**
- ▶ A PCB **keeps all the information needed to keep track of a process.**
- ▶ The PCB is maintained for a process **throughout its lifetime** and is **deleted once the process terminates.**
- ▶ The **architecture** of a PCB is completely **dependent on operating system** and may contain different information in different operating systems.
- ▶ PCB **lies in kernel memory space.**



# Fields of Process Control Block (PCB)

- ▶ **Process state.** The state may be new, ready, running, waiting, halted, and so on.
- ▶ **Program counter.** The counter indicates the address of the next instruction to be executed for this process.
- ▶ **CPU registers.** They include accumulators, index registers, stack pointers, and general-purpose registers. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward
- ▶ **CPU-scheduling information.** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- ▶ **Memory-management information.** This information include items such as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system.
- ▶ **Accounting information.** This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- ▶ **I/O status information.** This information includes the list of I/O devices allocated to the process, a list of open files, and so on.



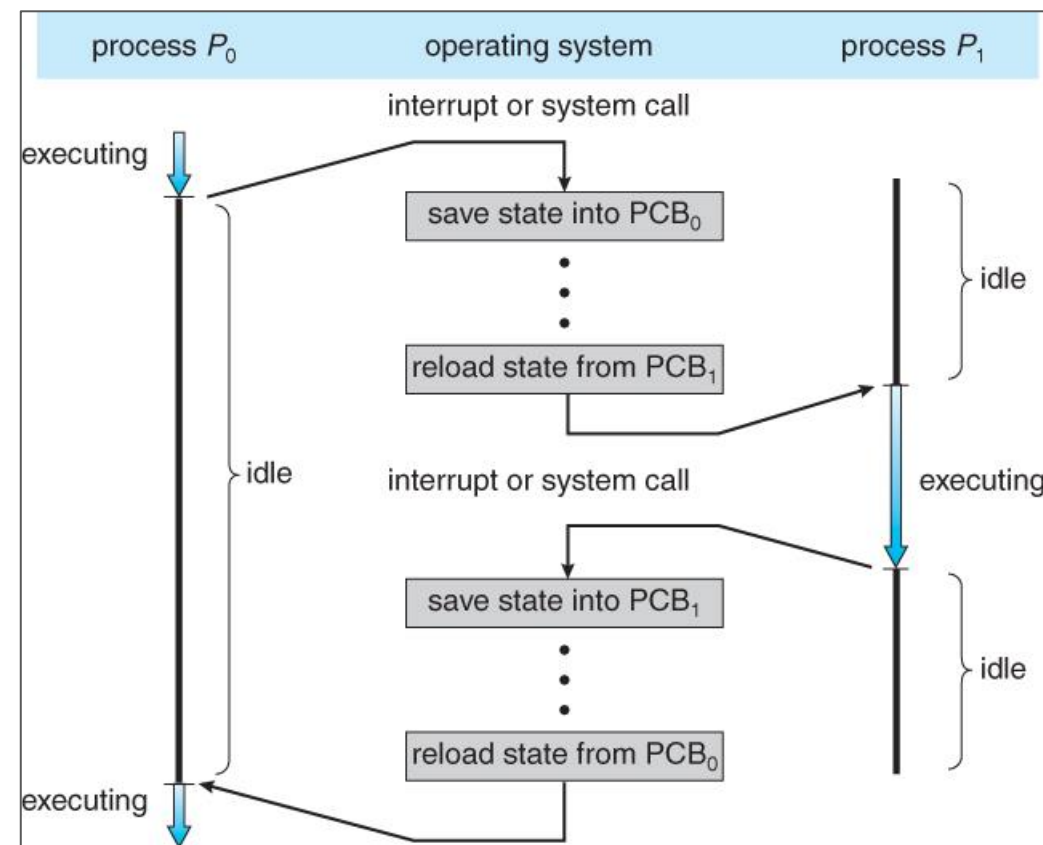


# Context switching



# Context switching

- ▶ Context switch means **stopping one process** and **restarting another process**.
- ▶ When an event occur, the **OS saves the state of an active process (into its PCB)** and **restore the state of new process (from its PCB)**.
- ▶ Context switching is **purely overhead** because system does not perform any useful work while context switch.
- ▶ Sequence of action:
  1. **OS takes control** (through interrupt)
  2. **Saves context of running process** in the process PCB
  3. **Reload context of new process** from the new process PCB
  4. **Return control** to new process





# **What is Process Scheduling?**





# What is Process Scheduling?

- ▶ Process scheduling is the **activity of the process manager** that handles **suspension of running process from CPU and selection of another process** on the basis of a particular strategy.
- ▶ The **part of operating system** that **makes the choice** is called **scheduler**.
- ▶ The **algorithm used by this scheduler** is called scheduling **algorithm**.
- ▶ Process scheduling is an essential part of a multiprogramming operating systems.





# Objectives (goals) of Scheduling



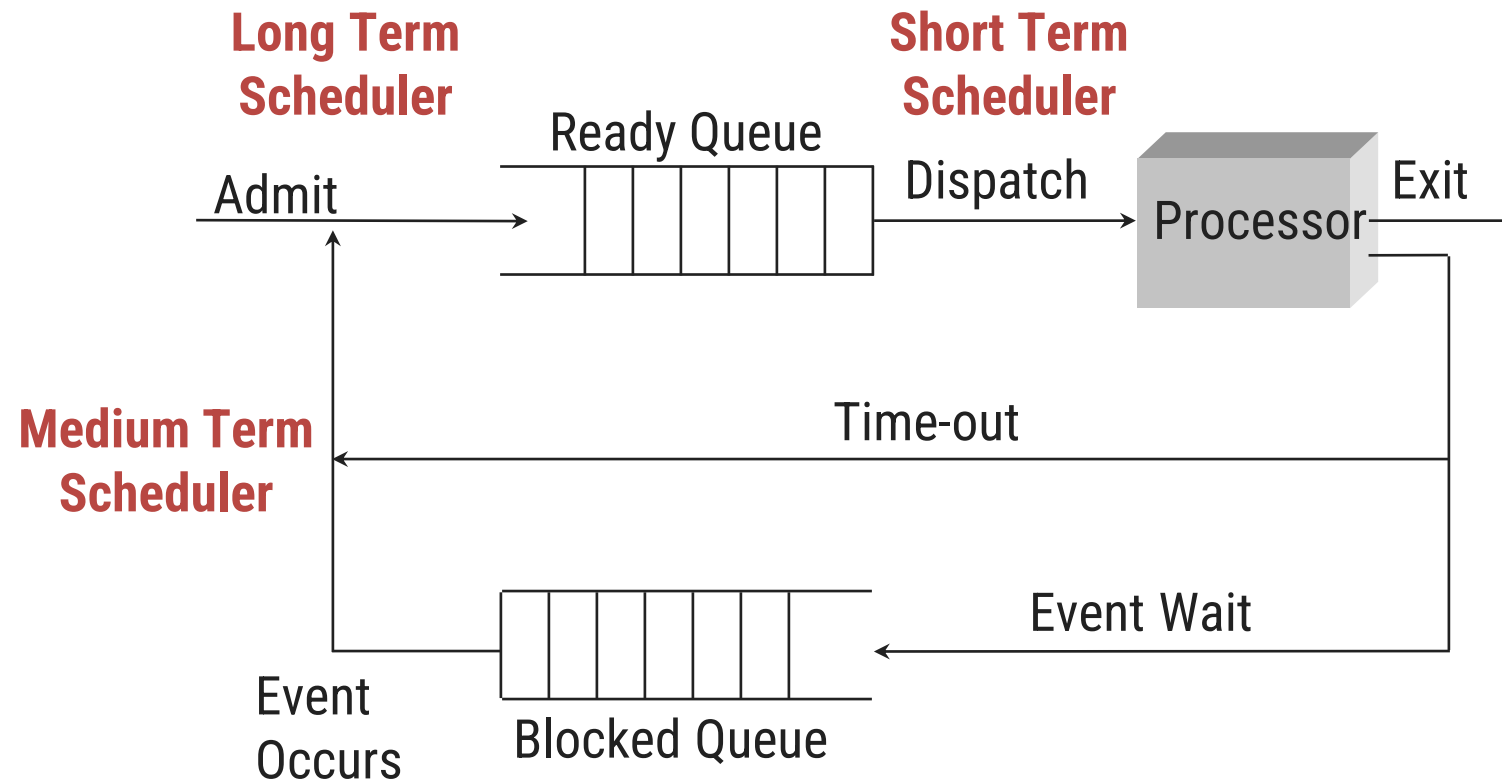
# Objectives (goals) of scheduling

- ▶ **Fairness**: giving each process a fair share of the CPU.
- ▶ **Balance**: keeping all the parts of the system busy (**Maximize**).
- ▶ **Throughput**: no of processes that are completed per time unit (**Maximize**).
- ▶ **Turnaround time**: time to execute a process from submission to completion (**Minimize**).
  - ↳ **Turnaround time** = Process finish time – Process arrival time
- ▶ **CPU utilization**: percent of time that the CPU is busy in executing a process.
  - ↳ keep CPU as busy as possible (**Maximized**).
- ▶ **Response time**: time between issuing a command and getting the result (**Minimized**).
- ▶ **Waiting time**: amount of time a process has been waiting in the ready queue (**Minimize**).
  - ↳ **Waiting time** = Turnaround time – Actual execution time



# Types of Schedulers

# Types of schedulers



# Types of schedulers

Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
It is a <b>job scheduler</b> .	It is a <b>CPU scheduler</b> .	It is a <b>process swapping scheduler</b> .
It <b>selects processes from pool</b> and loads them into memory for execution.	It <b>selects those processes which are ready</b> to execute.	It can <b>re-introduce the process</b> into memory and execution can be continued.
<b>Speed is lesser</b> than short term scheduler.	<b>Speed is fastest</b> among other two schedulers.	<b>Speed is in between both</b> short and long term scheduler.
It is almost <b>absent or minimal</b> in time sharing system.	It is also <b>minimal in time sharing system</b> .	It is <b>a part of time sharing systems</b> .



# **Types of Scheduling Algorithms**



# Types of Scheduling Algorithms

1. **Preemptive Scheduling Algorithms** - Preemptive scheduling is a method that may be used when a process switches from a running state to a ready state or from a waiting state to a ready state. The resources are assigned to the process for a particular time and then removed. If the resources still have the remaining CPU burst time, the process is placed back in the ready queue.

**Ex-** Shortest Remaining Time Next (SRTN), Round Robin (RR), Priority

2. **Non preemptive Scheduling Algorithm**- Non-preemptive scheduling is a method that may be used when a process terminates or switches from a running to a waiting state. When processors are assigned to a process, they keep the process until it is eliminated or reaches a waiting state. When the processor starts the process execution, it must complete it before executing the other process, and it may not be interrupted in the middle.

**Ex-** First Come First Served (FCFS), Shortest Job First (SJF)





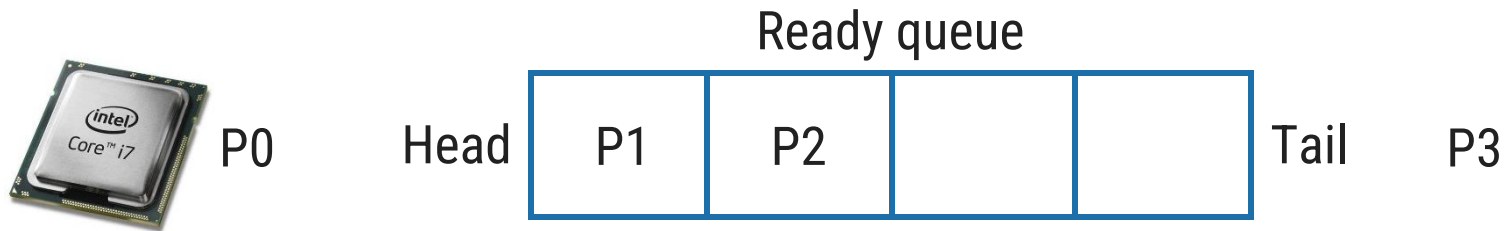
# First Come First Served (FCFS)



# First Come First Served (FCFS)

## ► Selection criteria:

- The **process that request first is served first**.
- It means that **processes are served in the exact order of their arrival**.



## ► Decision Mode:

- **Non preemptive**: Once a process is selected, it runs until it is blocked for an I/O or some other event or it is terminated.

## ► Implementation:

- This strategy can be easily **implemented by using FIFO** (First In First Out) queue.
- When CPU becomes free, a process from the first position in a queue is selected to run.

# First Come First Served (FCFS)

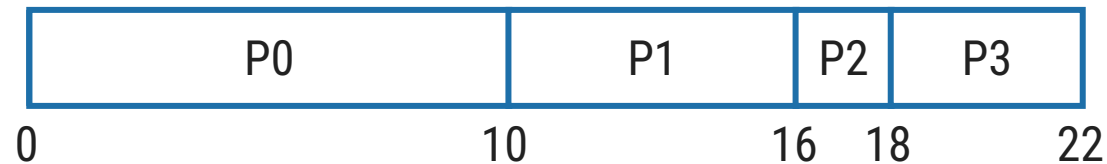
Process	Arrival Time (T <sub>0</sub> )	Burst/Execution Time (ΔT)	Finish Time (T <sub>1</sub> )	Turnaround Time (TAT = T <sub>1</sub> - T <sub>0</sub> )	Waiting Time (WT = TAT - ΔT)
P0	0	10			
P1	1	6			
P2	3	2			
P3	5	4			

- **Turnaround Time-** Finish Time – Arrival Time
- **Waiting Time-** Turnaround Time - Execution/Burst Time

# First Come First Served (FCFS)

Process	Arrival Time (T <sub>0</sub> )	Burst Time (ΔT)	Finish Time (T <sub>1</sub> )	Turnaround Time (TAT = T <sub>1</sub> - T <sub>0</sub> )	Waiting Time (WT = TAT - ΔT)
P0	0	10	10	10	0
P1	1	6	16	15	9
P2	3	2	18	15	13
P3	5	4	22	17	13

## ► Gantt Chart



► Average Turnaround Time:  $(10+15+15+17)/4 = 14.25$  ms.

► Average Waiting Time:  $(0+9+13+13)/4 = 8.75$  ms.

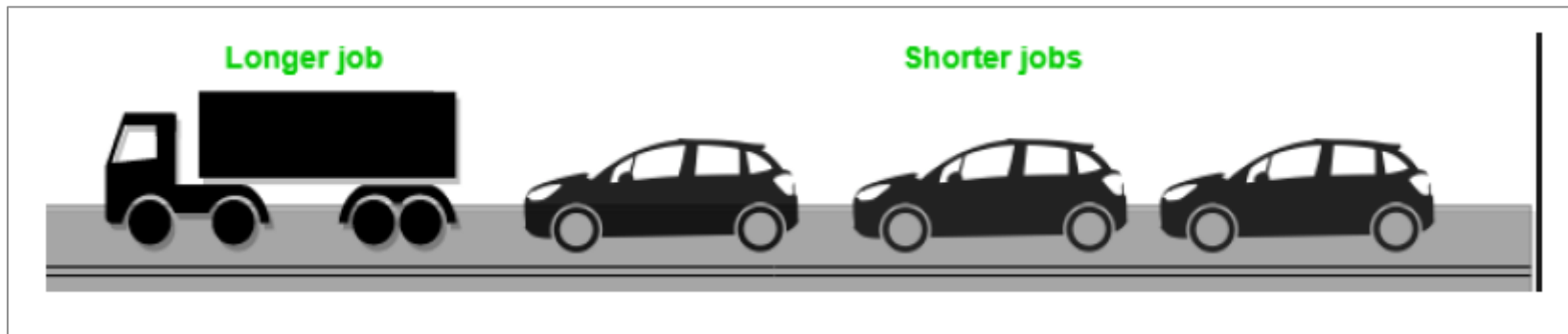
# First Come First Served (FCFS)

## ► Advantages

- **Simple** and **fair**.
- **Easy to understand** and **implement**.
- Every process will get a chance to run, so **starvation doesn't occur**.
  - Starvation is the **problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time**.

## ► Disadvantages

- **Not efficient** because average waiting time is too high.
- **Convoy effect is possible**. All small I/O bound processes wait for one big CPU bound process to acquire CPU.



- **CPU utilization may be less efficient** especially when a CPU bound process is running with many I/O bound processes.



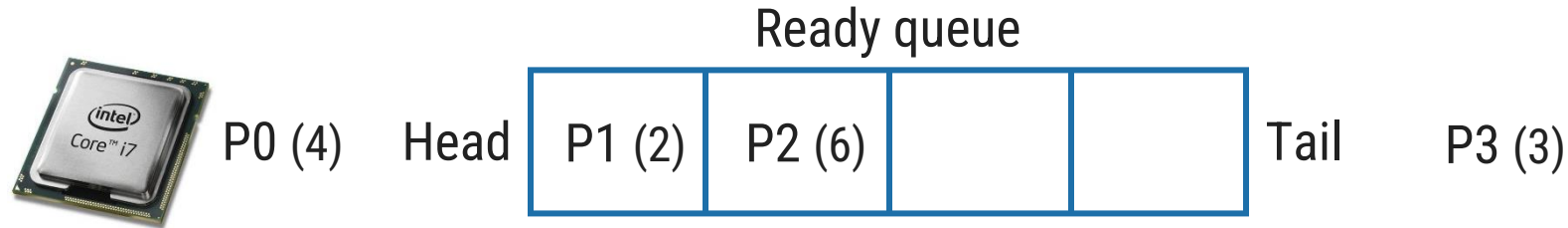
# Shortest Job First (SJF)



# Shortest Job First (SJF)

## ► Selection criteria:

- The process, that **requires shortest time to complete execution, is served first.**



## ► Decision Mode:

- **Non preemptive:** Once a process is selected, it runs until it is blocked for an I/O or some other event or it is terminated.

## ► Implementation:

- This strategy can be easily **implemented by using sorted FIFO** (First In First Out) queue.
- All processes in a queue are **sorted in ascending order based on their required CPU bursts.**
- When CPU becomes free, a process from the first position in a queue is selected to run.

# Shortest Job First (SJF)

Process	Arrival Time (T <sub>0</sub> )	Burst/Execution Time (ΔT)	Finish Time (T <sub>1</sub> )	Turnaround Time (TAT = T <sub>1</sub> - T <sub>0</sub> )	Waiting Time (WT = TAT - ΔT)
P0	0	10			
P1	1	6			
P2	3	2			
P3	5	4			

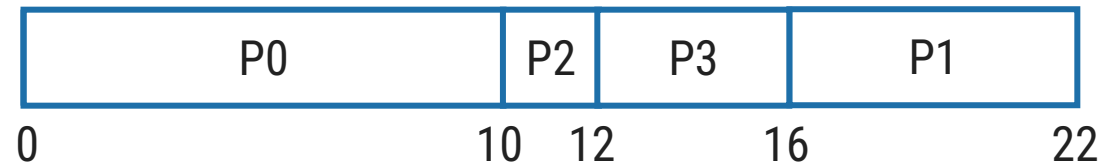
- **Turnaround Time-** Finish Time – Arrival Time
- **Waiting Time-** Turnaround Time - Execution/Burst Time



# Shortest Job First (SJF)

Process	Arrival Time (T <sub>0</sub> )	Burst Time ( $\Delta T$ )	Finish Time (T <sub>1</sub> )	Turnaround Time (TAT = T <sub>1</sub> - T <sub>0</sub> )	Waiting Time (WT = TAT - $\Delta T$ )
P0	0	10	10	10	0
P1	1	6	22	21	15
P2	3	2	12	9	7
P3	5	4	16	11	7

## ► Gantt Chart



► Average Turnaround Time:  $(10+21+9+11)/4 = 12.75$  ms.

► Average Waiting Time:  $(0+15+7+7)/4 = 7.25$  ms.

# Shortest Job First (SJF)

## ► Advantages

- **Less waiting time.**
- **Good response for short processes.**

## ► Disadvantages

- It is **difficult to estimate time required** to complete execution.
- **Starvation is possible for long process.** Long process may wait forever.
  - Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time.



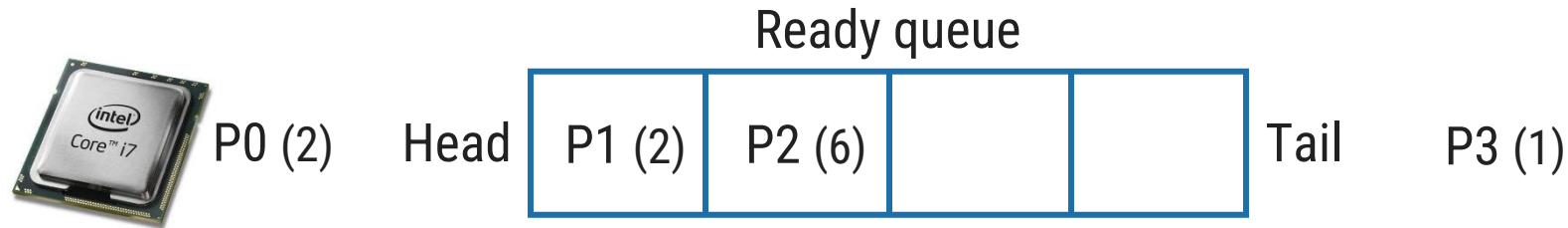
# Shortest Remaining Time Next (SRTN)



# Shortest Remaining Time Next (SRTN)

## ► Selection criteria:

- The process, **whose remaining run time is shortest, is served first**. This is a **preemptive version of SJF** scheduling.



## ► Decision Mode:

- **Preemptive:** When a new process arrives, its total time is compared to the current process remaining run time.
- If the **new process needs less time to finish than the current process**, the current process is suspended and the new job is started.

## ► Implementation:

- This strategy can also be implemented by using **sorted FIFO queue**.
- All processes in a queue are **sorted in ascending order on their remaining run time**.
- When CPU becomes free, a process from the first position in a queue is selected to run.

# Shortest Remaining Time Next (SRTN)

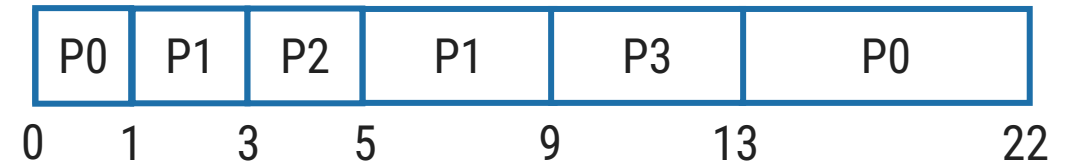
Process	Arrival Time (T <sub>0</sub> )	Burst/Execution Time (ΔT)	Finish Time (T <sub>1</sub> )	Turnaround Time (TAT = T <sub>1</sub> - T <sub>0</sub> )	Waiting Time (WT = TAT - ΔT)
P0	0	10			
P1	1	6			
P2	3	2			
P3	5	4			

- **Turnaround Time-** Finish Time – Arrival Time
- **Waiting Time-** Turnaround Time - Execution/Burst Time

# Shortest Remaining Time Next (SRTN)

Process	Arrival Time (T <sub>0</sub> )	Burst Time (ΔT)	Finish Time (T <sub>1</sub> )	Turnaround Time (TAT = T <sub>1</sub> - T <sub>0</sub> )	Waiting Time (WT = TAT - ΔT)
P0	0	10	22	22	12
P1	1	6	9	8	2
P2	3	2	5	2	0
P3	5	4	13	8	4

## ► Gantt Chart



► Average Turnaround Time: 10 ms

► Average Waiting Time: 4.5 ms

# Shortest Remaining Time Next (SRTN)

## ► Advantages

- ↳ **Less waiting time.**
- ↳ **Quite good response for short processes.**

## ► Disadvantages

- ↳ It is **difficult to estimate time required** to complete execution.
- ↳ **Starvation is possible for long process.** Long process may wait forever.
  - Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time.
- ↳ **Context switch overhead is there.**



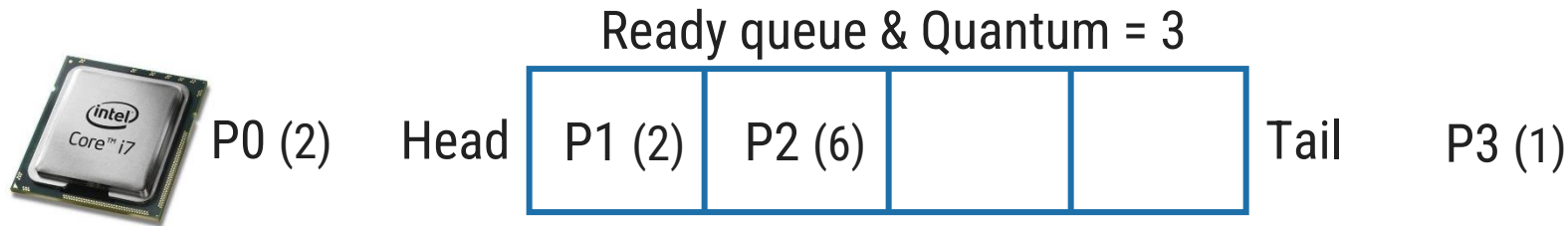
# Round Robin (RR)



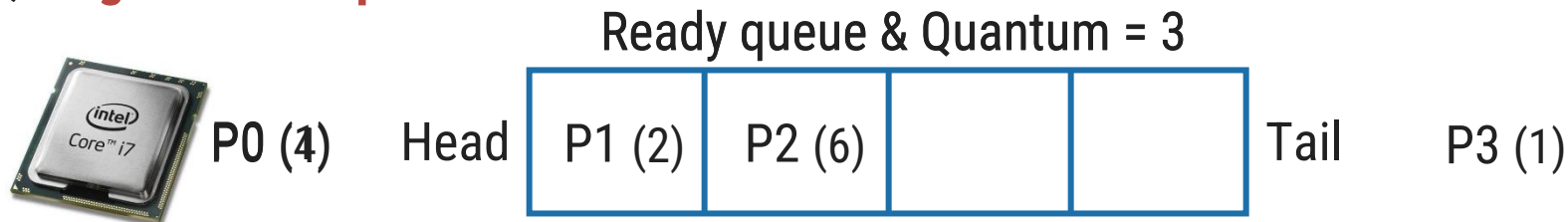
# Round Robin (RR)

## ► Selection criteria:

- ➔ Each selected process is **assigned a time interval, called time quantum or time slice**.
- ➔ Process is **allowed to run only for this time interval**.
- ➔ Here, two things are possible:
- ➔ First, **process is either blocked or terminated before the quantum has elapsed**. In this case the **CPU switching is done** and **another process is scheduled** to run.



- ➔ Second, **process needs CPU burst longer than time quantum**. In this case, process is **running at the end of the time quantum**.
- ➔ Now, it will be **preempted** and **moved to the end of the queue**.
- ➔ CPU will be **allocated to another process**.
- ➔ Here, **length of time quantum is critical to determine**.



# Round Robin (RR)

## ► Decision Mode:

- **Preemptive:** When a new process arrives, its total time is compared to the current process remaining run time.
- **Selection of new job is as per FCFS** scheduling algorithm.

## ► Implementation:

- This strategy can be implemented by using **circular FIFO queue**.
- If any process comes, or process releases CPU, or process is preempted. It is moved to the end of the queue.
- When CPU becomes free, a process from the first position in a queue is selected to run.

# Round Robin

Process	Arrival Time (T <sub>0</sub> )	Burst/Execution Time (ΔT)	Finish Time (T <sub>1</sub> )	Turnaround Time (TAT = T <sub>1</sub> - T <sub>0</sub> )	Waiting Time (WT = TAT - ΔT)
P0	0	10			
P1	1	6			
P2	3	2			
P3	5	4			

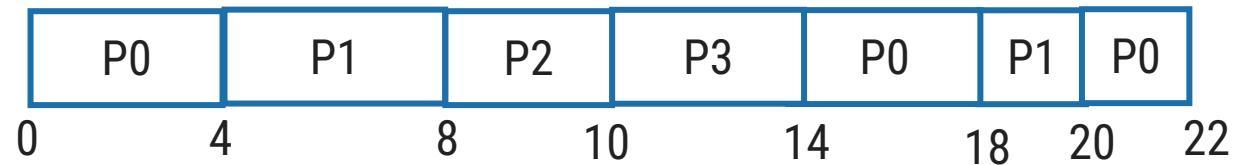
- **Turnaround Time-** Finish Time – Arrival Time
- **Waiting Time-** Turnaround Time - Execution/Burst Time

# Round Robin (RR)

Process	Arrival Time (T <sub>0</sub> )	Burst Time (ΔT)	Finish Time (T <sub>1</sub> )	Turnaround Time (TAT = T <sub>1</sub> - T <sub>0</sub> )	Waiting Time (WT = TAT - ΔT)
P0	0	10	22	22	12
P1	1	6	20	19	13
P2	3	2	10	7	5
P3	5	4	14	9	5

## ► Gantt Chart

■ Quantum time is 4 ms



► Avg. Turnaround Time: 14.25 ms

► Avg. Waiting Time: 8.75 ms

# Round Robin (RR)

## ► Advantages

- ↳ **Simplest, fairest and most widely used algorithms.**

## ► Disadvantages

- ↳ **Context switch overhead is there.**
- ↳ **Determination of time quantum is too critical.**
  - If it is **too short**, it causes frequent context switches and lowers CPU efficiency.
  - If it is **too long**, it causes poor response for short interactive process.



# **Priority** **(Non-Preemptive Priority)**

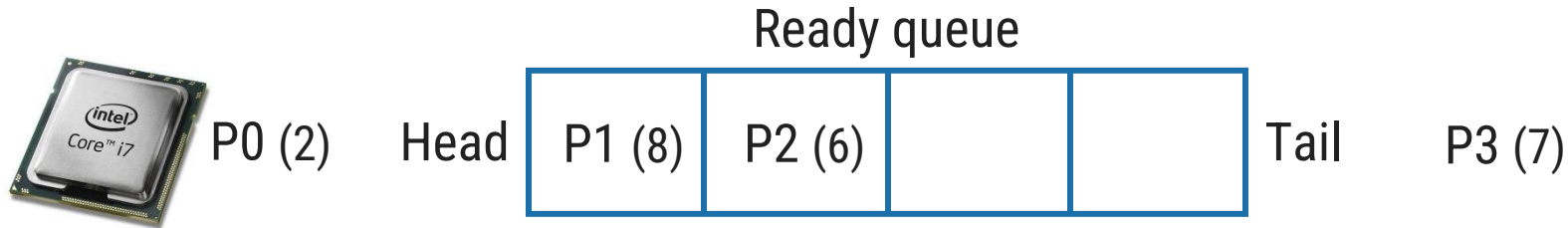
Section – 4.5.1



# Non-Preemptive Priority

## ▶ Selection criteria:

- The process, that **has highest priority, is served first.**



## ▶ Decision Mode:

- **Non preemptive: Once a process is selected, it runs until it is blocked** for an I/O or some other event or it is terminated.

## ▶ Implementation:

- This strategy can also be implemented by using **sorted FIFO queue.**
- All processes in a queue are **sorted based on their priority with highest priority process at front end.**
- When CPU becomes free, a process from the first position in a queue is selected to run.

# Priority (Non-Preemptive )

Process	Arrival Time (T <sub>0</sub> )	Burst Time (ΔT)	Priority	Finish Time (T <sub>1</sub> )	Turnaround Time (TAT = T <sub>1</sub> - T <sub>0</sub> )	Waiting Time (WT = TAT - ΔT)
P0	0	10	5			
P1	1	6	4			
P2	3	2	2			
P3	5	4	0			

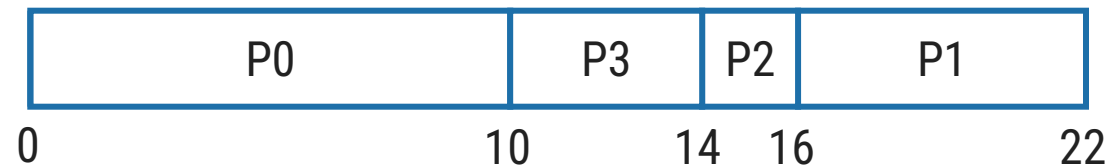
- **Turnaround Time-** Finish Time – Arrival Time
- **Waiting Time-** Turnaround Time - Execution/Burst Time



# Non-Preemptive Priority

Process	Arrival Time (T0)	Burst Time ( $\Delta T$ )	Priority	Finish Time (T1)	Turnaround Time (TAT = T1 - T0)	Waiting Time (WT = TAT - $\Delta T$ )
P0	0	10	5	10	10	0
P1	1	6	4	22	21	15
P2	3	2	2	16	13	11
P3	5	4	0	14	9	5

► Gantt Chart (small values for priority means higher priority of a process)



► Avg. Turnaround Time: 13.25 ms

► Avg. Waiting Time: 7.75 ms

# Non-Preemptive Priority

## ► Advantages

- **Priority is considered so critical processes can get even better response time.**

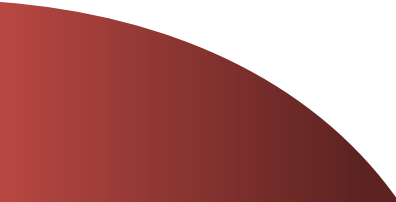
## ► Disadvantages

- **Starvation is possible for low priority processes.** It can be overcome by using technique called '**Aging**'.
- **Aging: gradually increases the priority of processes that wait in the system for a long time.**



# Priority (Preemptive Priority)

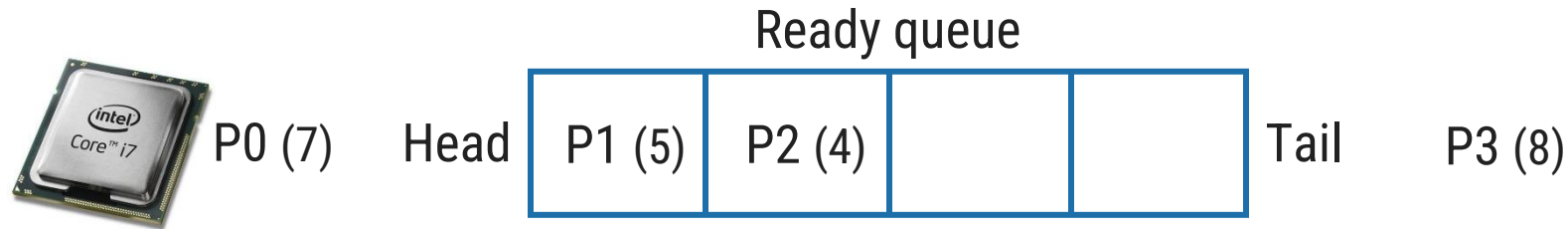
Section – 4.5.2



# Preemptive Priority

## ▶ Selection criteria:

- The process, that **has highest priority, is served first.**



## ▶ Decision Mode:

- **Preemptive:** When a new process arrives, its priority is compared with current process priority.
- If the **new process has higher priority than the current**, the current process is suspended and new job is **started.**

## ▶ Implementation:

- This strategy can also be implemented by using **sorted FIFO queue.**
- All processes in a queue are **sorted based on their priority with highest priority process at front end.**
- When CPU becomes free, a process from the first position in a queue is selected to run.

# Priority (Preemptive )

Process	Arrival Time (T0)	Burst Time (ΔT)	Priority	Finish Time (T1)	Turnaround Time (TAT = T1 - T0)	Waiting Time (WT = TAT - ΔT)
P0	0	10	5			
P1	1	6	4			
P2	3	2	2			
P3	5	4	0			

- **Turnaround Time-** Finish Time – Arrival Time
- **Waiting Time-** Turnaround Time - Execution/Burst Time

# Preemptive Priority

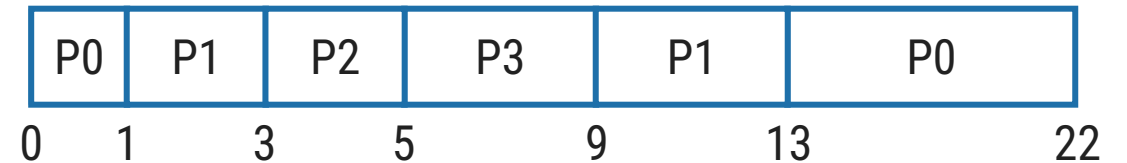
Process	Arrival Time (T0)	Burst Time ( $\Delta T$ )	Priority	Finish Time (T1)	Turnaround Time (TAT = T1 - T0)	Waiting Time (WT = TAT - $\Delta T$ )
P0	0	10	5	22	22	12
P1	1	6	4	13	12	6
P2	3	2	2	5	2	0
P3	5	4	0	9	4	0

## ► Gantt Chart

→ small values means higher priority

► Avg. Turnaround Time: 10 ms

► Avg. Waiting Time: 4.5 ms



# Preemptive Priority

## ► Advantages

- **Priority is considered so critical processes can get even better response time.**

## ► Disadvantages

- **Starvation is possible for low priority processes. It can be overcome by using technique called 'Aging'.**
- **Aging: gradually increases the priority of processes that wait in the system for a long time.**
- **Context switch overhead is there.**

# Exercise

1. Five batch jobs A to E arrive at same time. They have estimated running times 11,6,2,4 and 8 minutes. Their priorities are 3,5,2,1 and 4 respectively with 5 being highest priority. For each of the following algorithm determine mean process turnaround time. Quantum time is 1 minute.

- Round Robin, Priority Scheduling, FCFS, SJF.

2. Suppose that the following processes arrive for the execution at the times indicated. Each process will run the listed amount of time. Assume preemptive scheduling.

Process	Arrival Time (ms)	Burst Time (ms)
P1	0	8
P2	4	4
P3	1	1

- What is the turnaround time for these processes with Shortest Job First scheduling algorithm?



# Exercise

3. Consider the following set of processes with length of CPU burst time given in milliseconds.

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

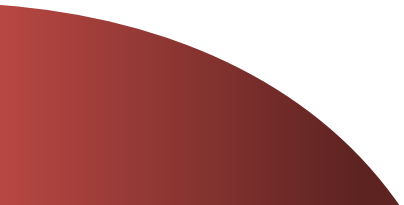
- Assume arrival order is: P1, P2, P3, P4, P5 all at time 0 and a smaller priority number implies a higher priority. Draw the Gantt charts illustrating the execution of these processes using preemptive priority scheduling.

# Questions

1. Consider Five Processes P1 to P5 arrived at same time. They have estimated running time 10, 2, 6, 8 and 4 seconds, respectively. Their Priorities are 3, 2, 5, 4 and 1, respectively with 5 being highest Priority. Find the average turnaround time and average waiting time for Round Robin (quantum time=3) and Priority Scheduling algorithm.
2. Consider the processes P1, P2, P3, P4 with burst time is 21, 3, 6 and 2 respectively, arrives for execution in the same order, with arrival time 0, draw GANTT chart and find the average waiting time using the FCFS and SJF scheduling algorithm.



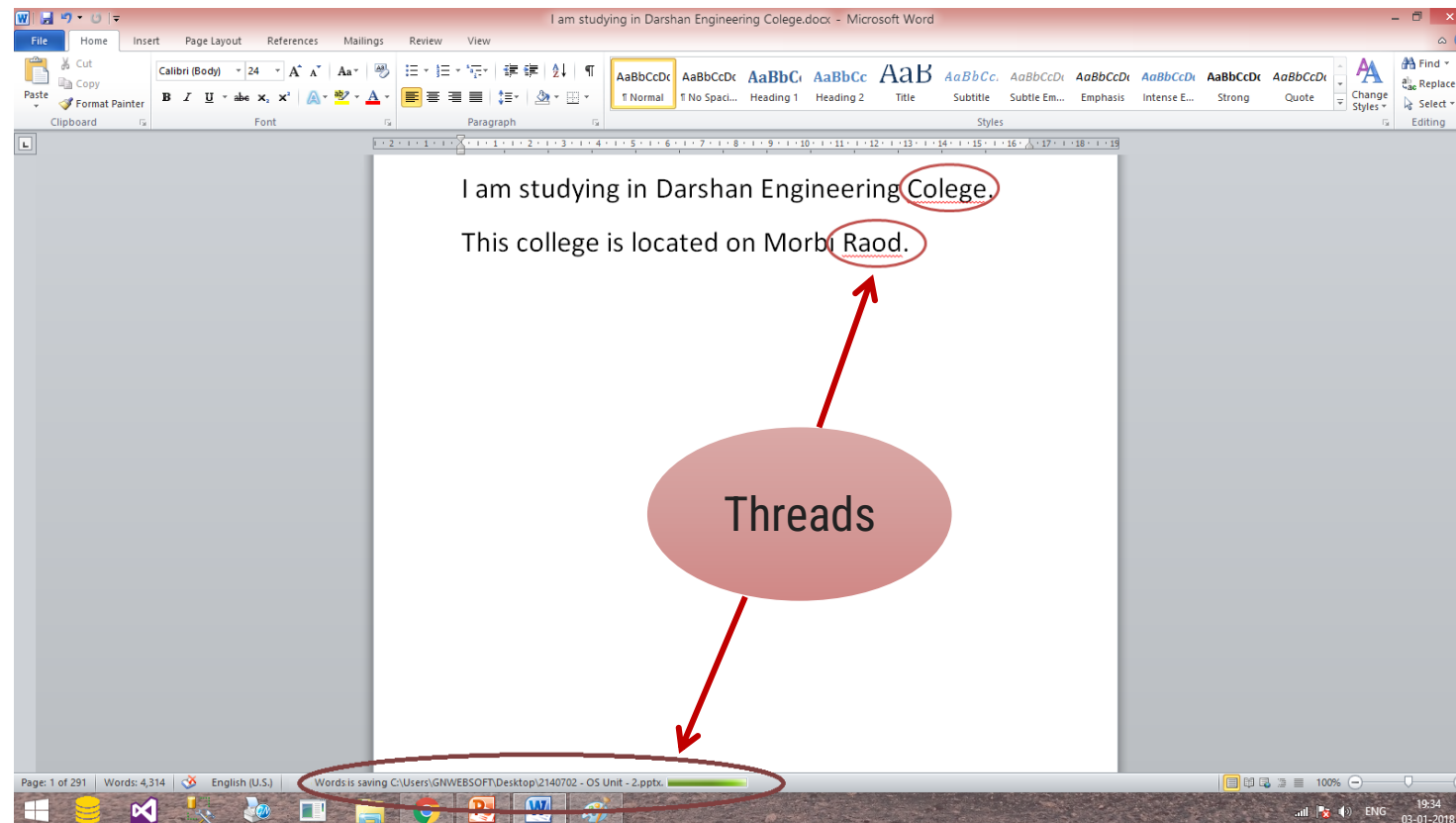
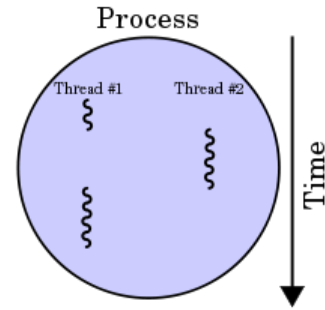
# Threads



# What is Threads?

- ▶ Thread is **light weight process** created by a process.

**Processes** are used to **execute large**, 'heavyweight' jobs such as working in word, while **threads** are used to carry out **smaller** or 'lightweight' jobs such as auto saving a word document.



# Thread Control Block

1. **Thread ID (TID)**- It is a unique identification number of the thread.
2. **PC**- Indicates the address value at which the next instruction of the thread will be executed by the processor.
3. **Registers**- CPU registers are used for the execution of a thread. While the thread is in execution, data registers, address registers, control registers, and status registers are used for executing and controlling the process.
4. **State**- A thread also has a number of states just like a process.
5. **Priority**- A priority number may be assigned to a thread as provided to a process.
6. **Event information**- This is the event for which a blocked process is waiting.
7. **Information related to scheduling**- The information related to scheduling of a thread, such as the waiting time and the execution span of the thread the last time it was running, is also stored.
8. **Pointer to owner process**- The thread will be created within a process and it needs to access the execution environment of its owner process.
9. **TCB pointer**- This is a pointer to another TCB that is used to maintain the scheduling list



# Comparison between Process and Thread



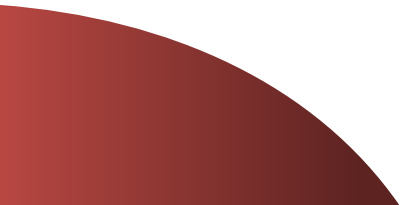
# Difference between Process & Thread

	Process	Thread
1.	Process means any program is in execution.	Thread means a segment of a process.
2.	The process takes more time to terminate.	The thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	The process is less efficient in terms of communication.	Thread is more efficient in terms of communication.
5.	If one process is blocked then it will not affect the execution of other processes	If a user-level thread is blocked, then all other user-level threads are blocked.
6.	The process has its own Process Control Block, Stack, and Address Space.	Thread has Parents' PCB, its own Thread Control Block, and Stack and common Address space.
7.	Changes to the parent process do not affect child processes.	Since all threads of the same process share address space and other resources, so any changes to the main thread may affect the behavior of the other threads of the process.
8.	The process does not share data with each other.	Threads share data with each other.



# Types of Threads

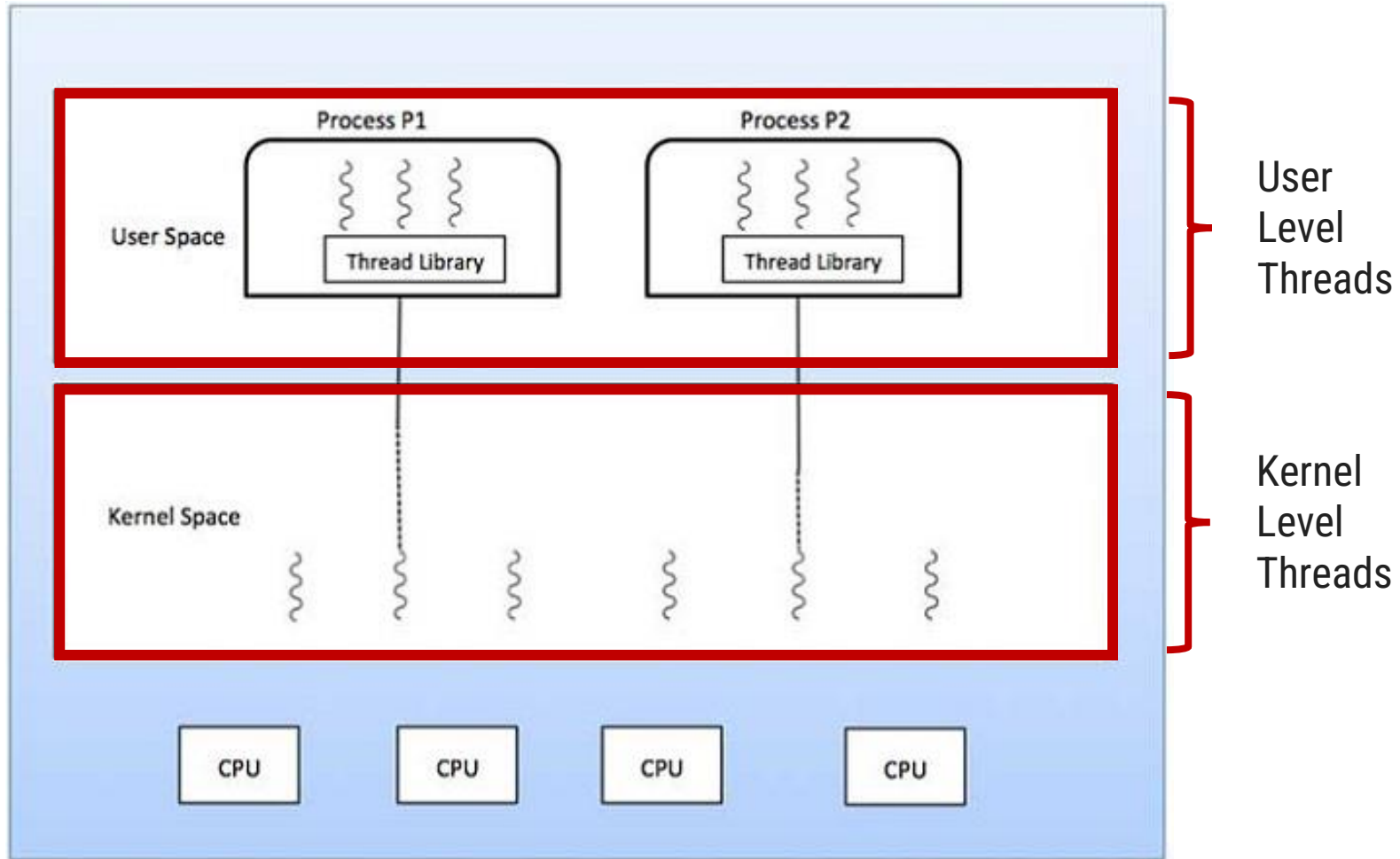
Section - 9





# Types of Threads

1. Kernel Level Thread
2. User Level Thread

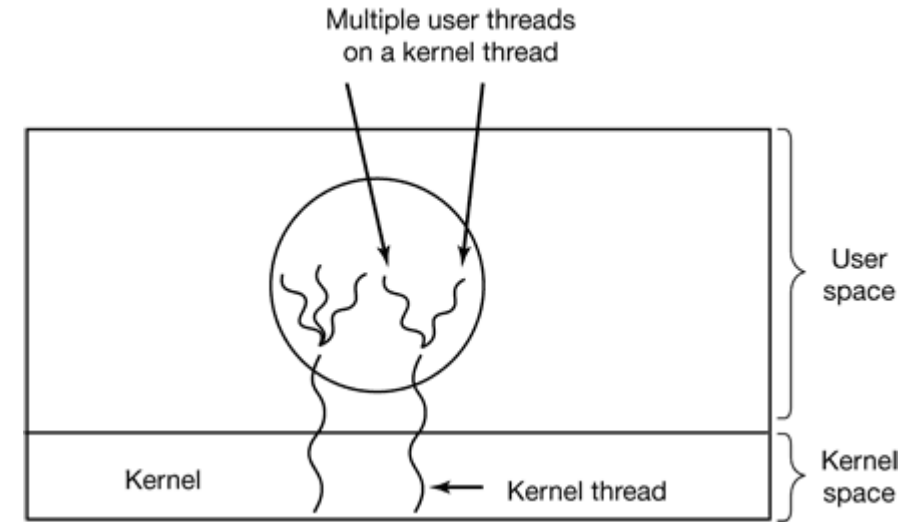


# Types of Threads

User Level Thread	Kernel Level Thread
User thread are implemented by users.	Kernel threads are implemented by OS.
OS doesn't recognize user level threads.	Kernel threads are recognized by OS.
Implementation of user threads is easy.	Implementation of kernel thread is complex.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Context switch requires hardware support.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread with in same process can continue execution.
Example : Java thread, POSIX threads.	Example : Window Solaris

# Hybrid Threads

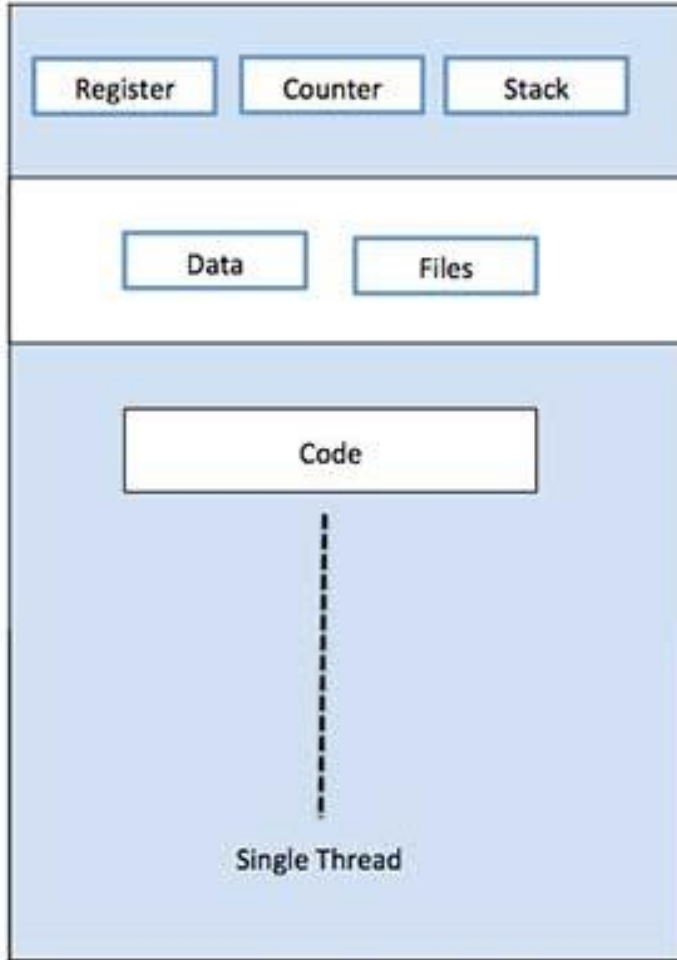
- ▶ Combines the advantages of user level and kernel level thread.
- ▶ It **uses kernel level thread** and then **multiplex user level thread on to some or all of kernel threads**.
- ▶ **Gives flexibility** to programmer that how many kernel level threads to use and how many user level thread to multiplex on each one.
- ▶ **Kernel is aware of only kernel level threads** and schedule it.



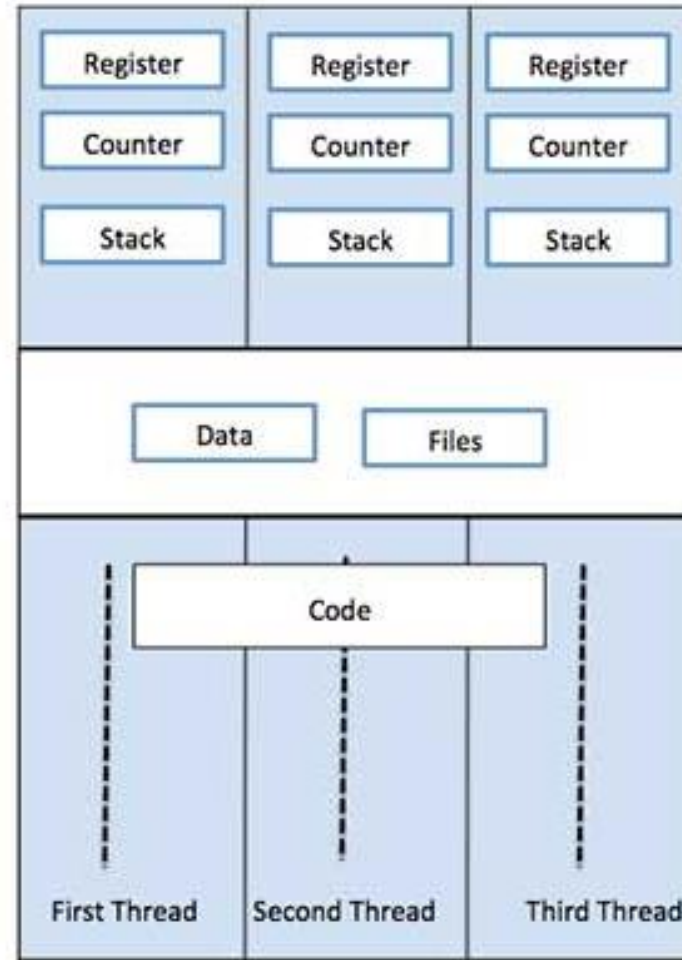


# Multithreading Concept

# Single Threaded Process VS Multiple Threaded Process



Single Process P with single thread



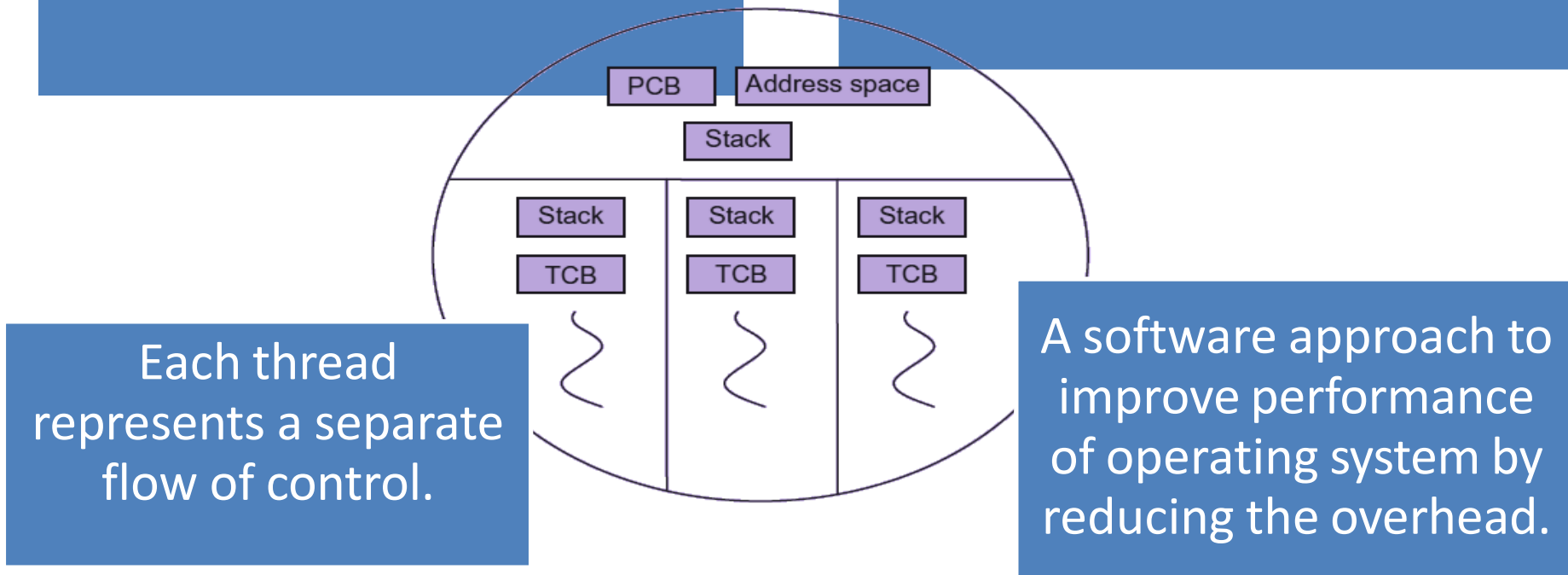
Single Process P with three threads

- A single-threaded process is a process with a single thread.
- A multi-threaded process is a process with multiple threads.
- The multiple threads have its own registers, stack and counter but they share the code and data segment.

# Multithreading Concept

Threads provide a way to improve application performance through parallelism.

Each thread belongs to exactly one process and no thread can exist outside a process.

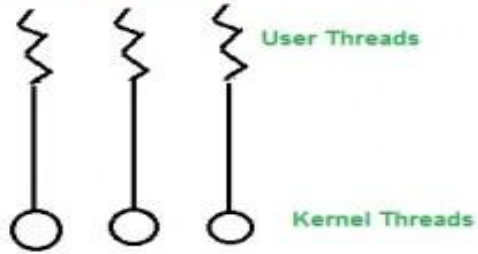




# Multithreading Models

# Multi Threading Models

One to One Model

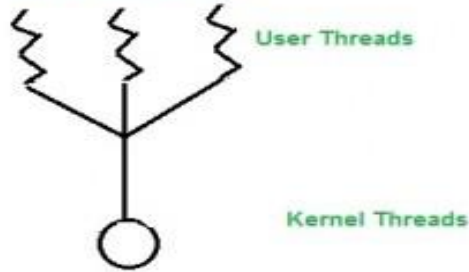


## One to One Model

Each user threads mapped to one kernel thread.

Problem with this model is that creating a user thread requires the corresponding kernel thread.

Many to One Model

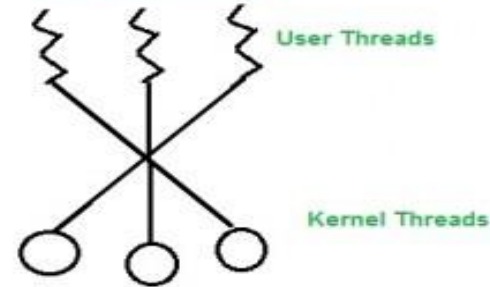


## Many to One Model

Multiple user threads mapped to one kernel thread.

Problem with this model is that a user thread can block entire process because we have only one kernel thread.

Many to Many Model



## Many to Many Model

Multiple user threads multiplex to more than one kernel threads.

Advantage with this model is that a user thread can not block entire process because we have multiple kernel thread.





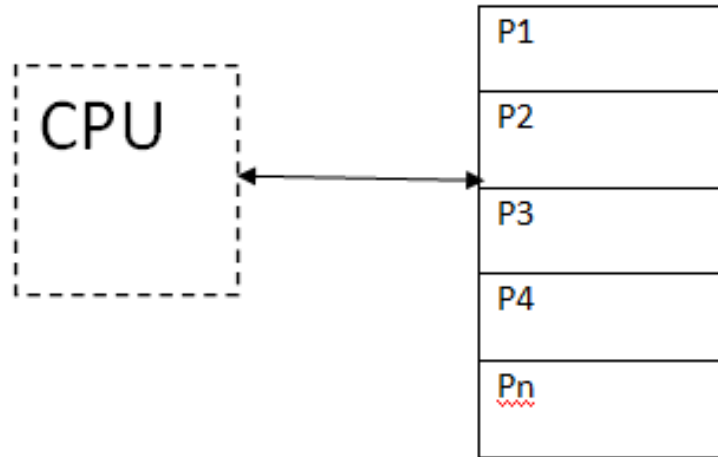
# **Difference between Multitasking & Multithreading**



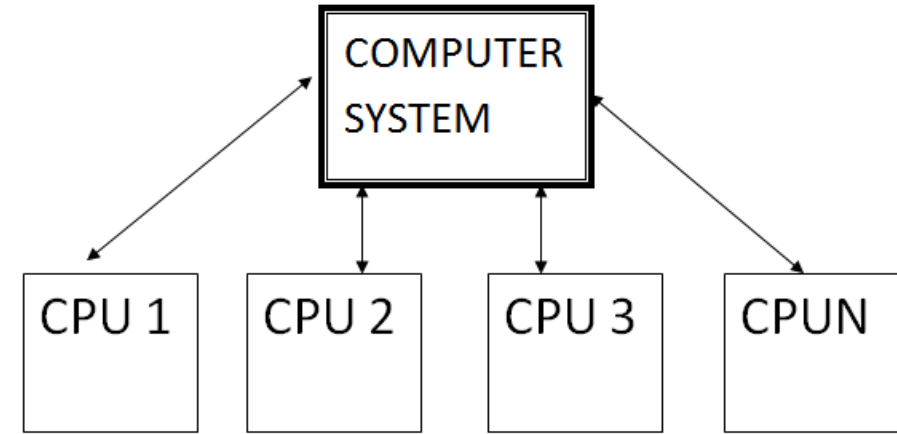
# Difference between Multitasking and Multithreading

	Multitasking	Multithreading
1	Multitasking involves often CPU switching between the tasks.	While in multithreading also, CPU switching is often involved between the threads.
2	In multitasking, the processes share separate memory.	While in multithreading, processes are allocated the same memory.
3	In multitasking, processes don't share the same resources, each process is allocated separate resources.	While in multithreading, each process shares the same resources.
4	Multitasking is slow compared to multithreading.	While multithreading is faster.
5	In multitasking, termination of a process takes more time.	While in multithreading, termination of thread takes less time.
6.	Involves running multiple independent processes or tasks	Involves dividing a single process into multiple threads that can execute concurrently
7	Each process or task has its own memory space and resources	Threads share the same memory space and resources of the parent process

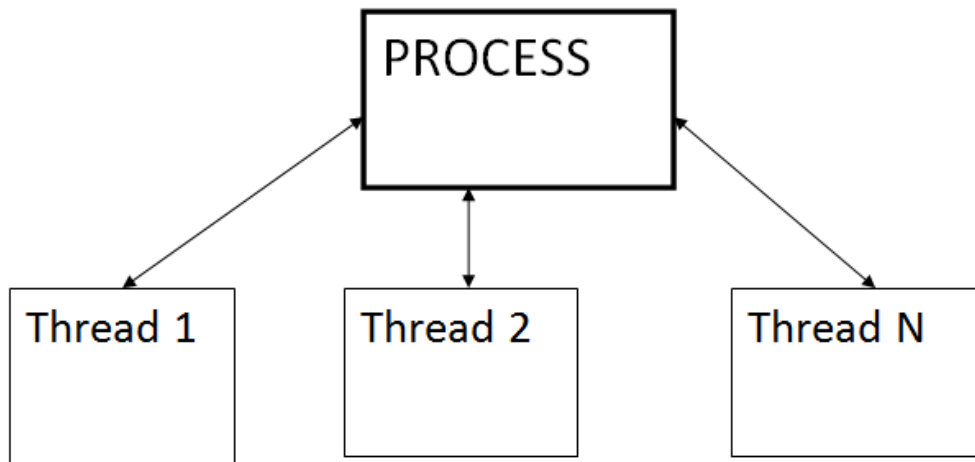
# Multiprogramming, Multiprocessing, Multithreading, Multitasking



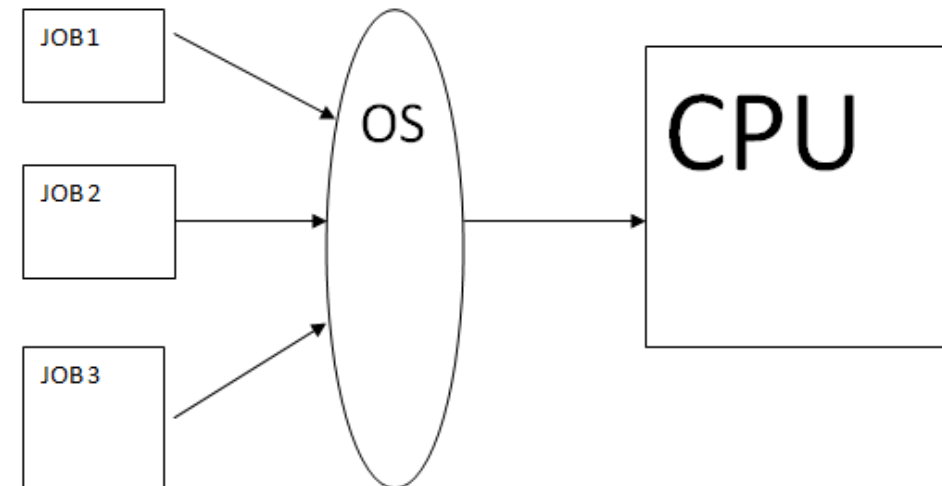
**Multiprogramming**



**Multiprocessing**



**Multithreading**



**Multitasking**

# Multiprogramming, Multiprocessing, Multithreading, Multitasking

Multiprogramming	Multitasking	Multithreading	Multiprocessing
Multiple programs run at the same time on the same single device.	Multiple tasks process a single resource simultaneously. It is defined as an extension of multiprogramming.	It is defined as an extension of multitasking.	Multiple processors are used for executing a single operation or job.
The process stays in the main memory.	The process stays in the CPU.	Multiple threads are processed on a single CPU.	The process switches between multiple processors to complete the execution.
Uses the batch operating system.	It uses a time-sharing operating system.	Here tasks keep on divided into subtasks.	Here multiple processors carry out the task of executing a job.
It is a slow process.	It follows the concept of context switching.	It allows a single process to get multiple statements for execution.	It can perform tasks in a short time as multiple processors are actively executing a task.



***Thank  
You***