Operating System

UNIT-02
Process and CPU Scheduling

By
Balaram Das
School of Engineering & Technology
GIET University, Gunupur

Process Concept

- When a program is loaded into the memory and execute the same, it becomes a process which performs all the tasks mentioned in the program.
- A process is basically a program in execution.
- The execution of a process must progress in a sequential fashion.

- **A process is an ‘active’ entity, whereas a program is a ‘passive’ entity.**
- **A single program can create many processes when run multiple times;**
- **A process is defined as an entity which represents the basic unit of work to be implemented in the system.**

- **A process will need certain resources-such as CPU time, memory, files, and I/O devices to accomplish its task.**
- **These resources are allocated to the process either when it is created or while it is executing.**

The Process

- When a program is loaded into memory, it may be divided into the four components to form a process. These components are:
- stack,
- heap,
- text and
- data.

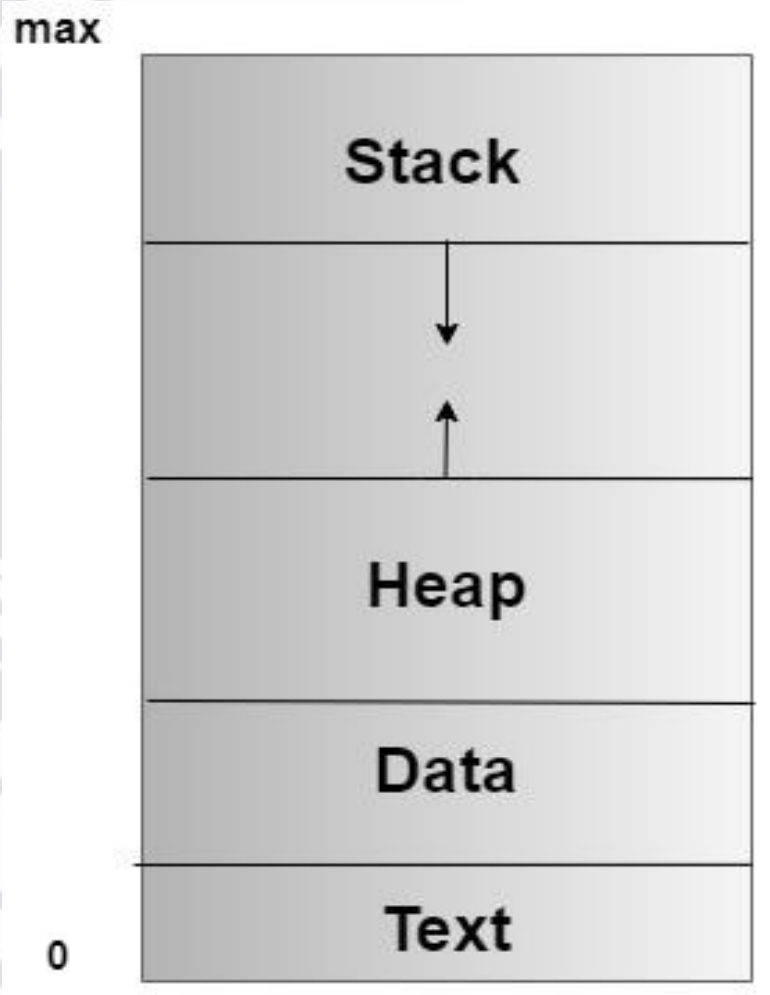


Figure: Process in the Memory

- The **Text section** is made up of the **compiled program code**, read in from non-volatile storage when the program is launched.
- The **Data section** is made up of the **global and static variables**, allocated and initialized prior to executing the main.
- The Heap is used for the **dynamic memory allocation** to a **process during its run time** and is managed via calls to new, delete, malloc, free, etc.
- The **Stack** is used for **local variables**. Space on the stack is reserved for local variables when they are declared.

Process State

- **As a process executes, it changes state.**
- **The state of a process is defined in part by the current activity of that process.**
- **A process may be in one of the following states:**
- **New: The process is being created.**
- **Ready: The process is waiting to be assigned to a processor.**
- **Running: Instructions are being executed.**
- **Waiting: The process is waiting for some event to occur (such as an I/O completion or reception of a signal).**
- **Terminated. The process has finished execution.**

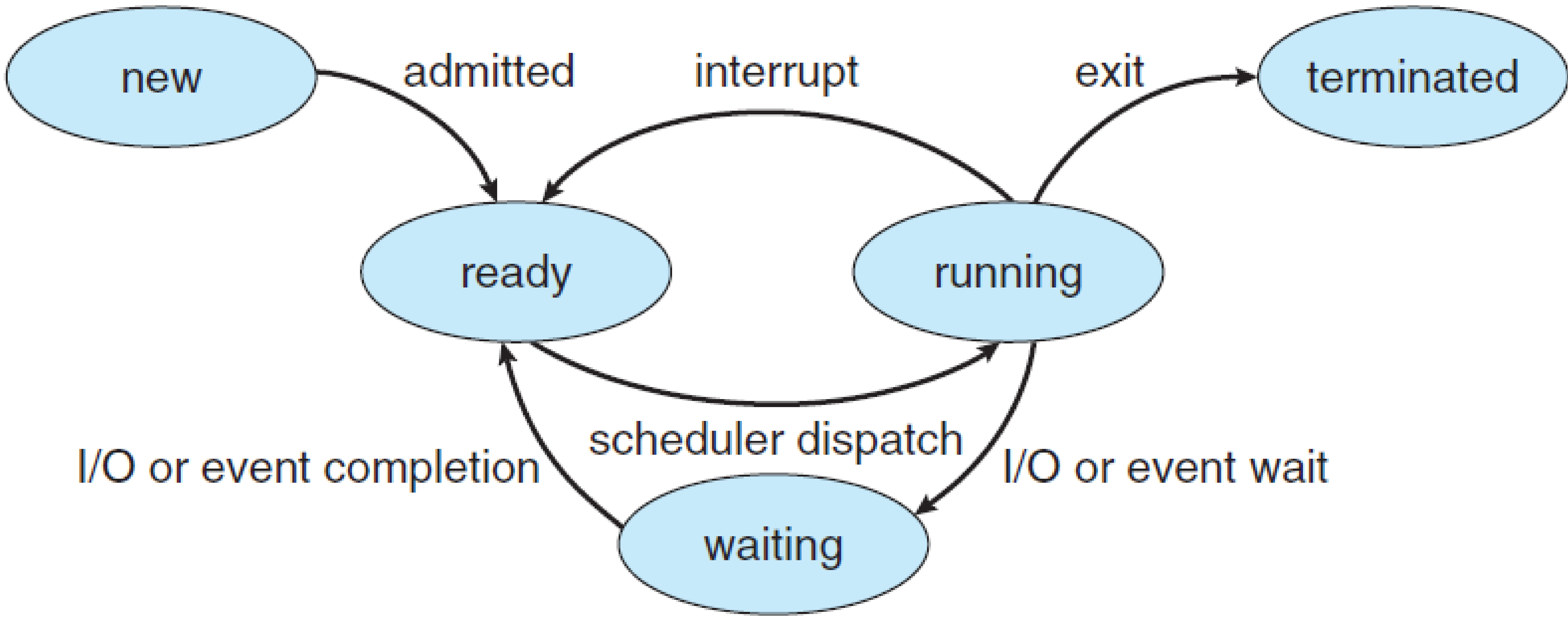


Figure 3.2 Diagram of process state

Process Control Block

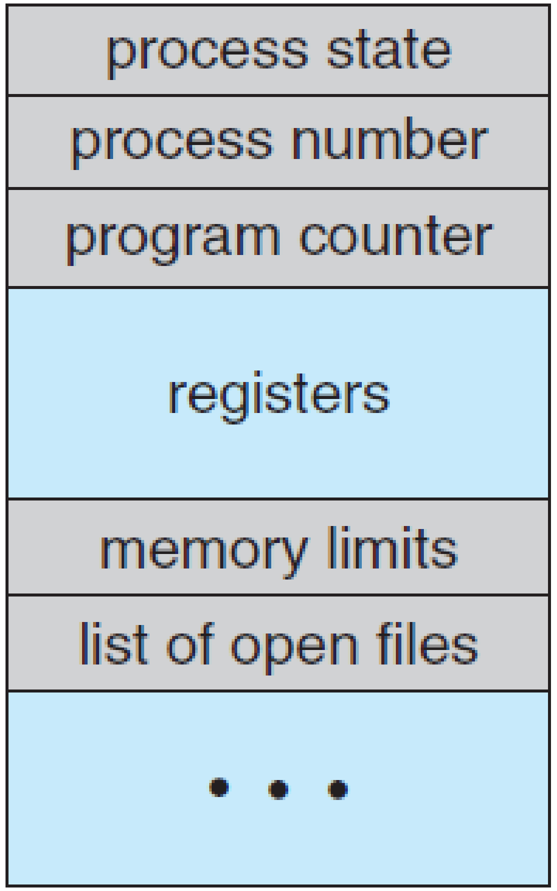
- A process control block (PCB) is a data structure used by computer operating systems to store all the information about a process.
- When a process is created, the operating system creates a corresponding process control block.
- It also defines the current state of the operating system.
- It is also called a task control block.

Attributes of a Process Control Block

- **Process Id:** A process id is a unique identity of a process. Each process is identified with the help of the process id.
- **Process state:** This specifies current state of a process i.e. new, ready, running, waiting or terminated.
- **Program counter:** The counter indicates the address of the next instruction to be executed for this process.

- **CPU registers.** Whenever an interrupt occurs and there is a context switch between the processes, the temporary information is stored in the CPU registers. So, that when the process resumes the execution it correctly gains from where it leaves.
- **CPU-scheduling information.** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- **Memory-management information:** It includes the page tables or the segment tables depending on the memory system used. It also contains the value of the base registers, limit registers etc.

- **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.



Threads

Introduction

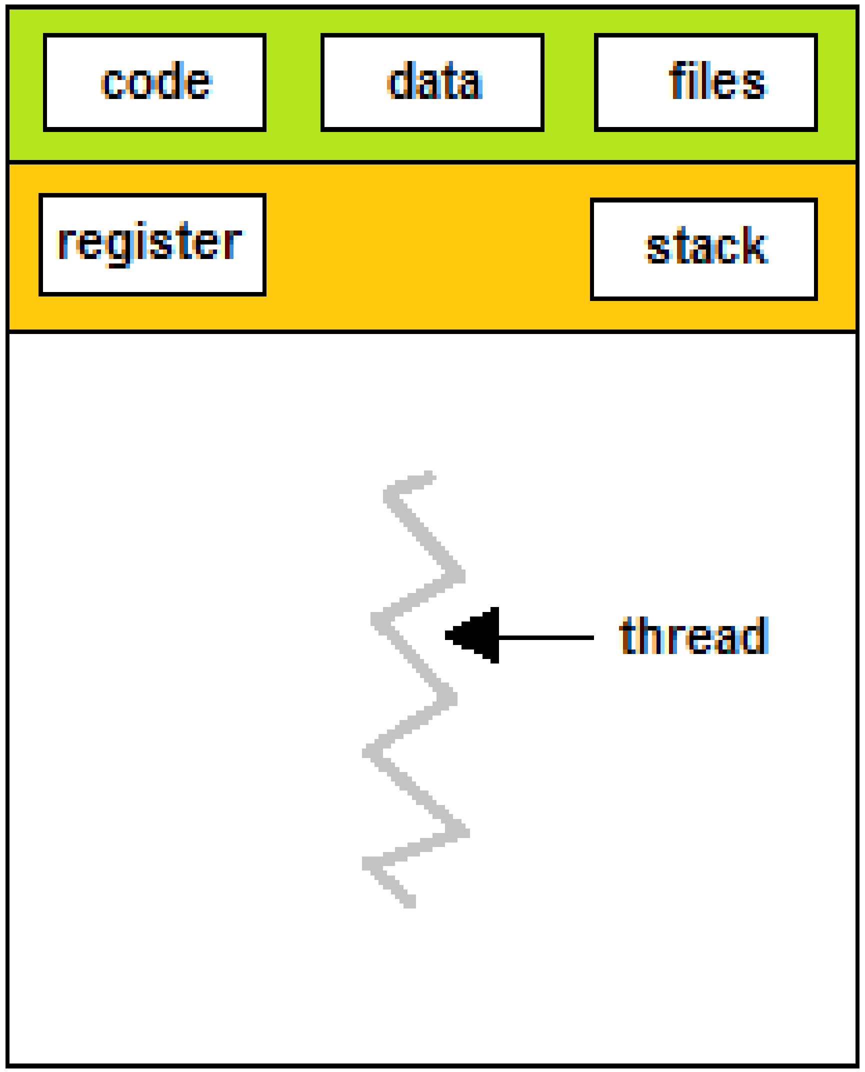
- **An application will have multiple processes and a process will have multiple threads.**
- **A process starts with a single thread(Primary thread) and later can create more thread from any of its threads.**
- **A process has at least one thread.**

- **All threads of a process have access to its memory and system resources.**
- **A thread is the smallest unit of execution to which processor allocates time.**
- **It consists of**
- **A program counter**
- **A set of registers**
- **A unique id**
- **A stack.**

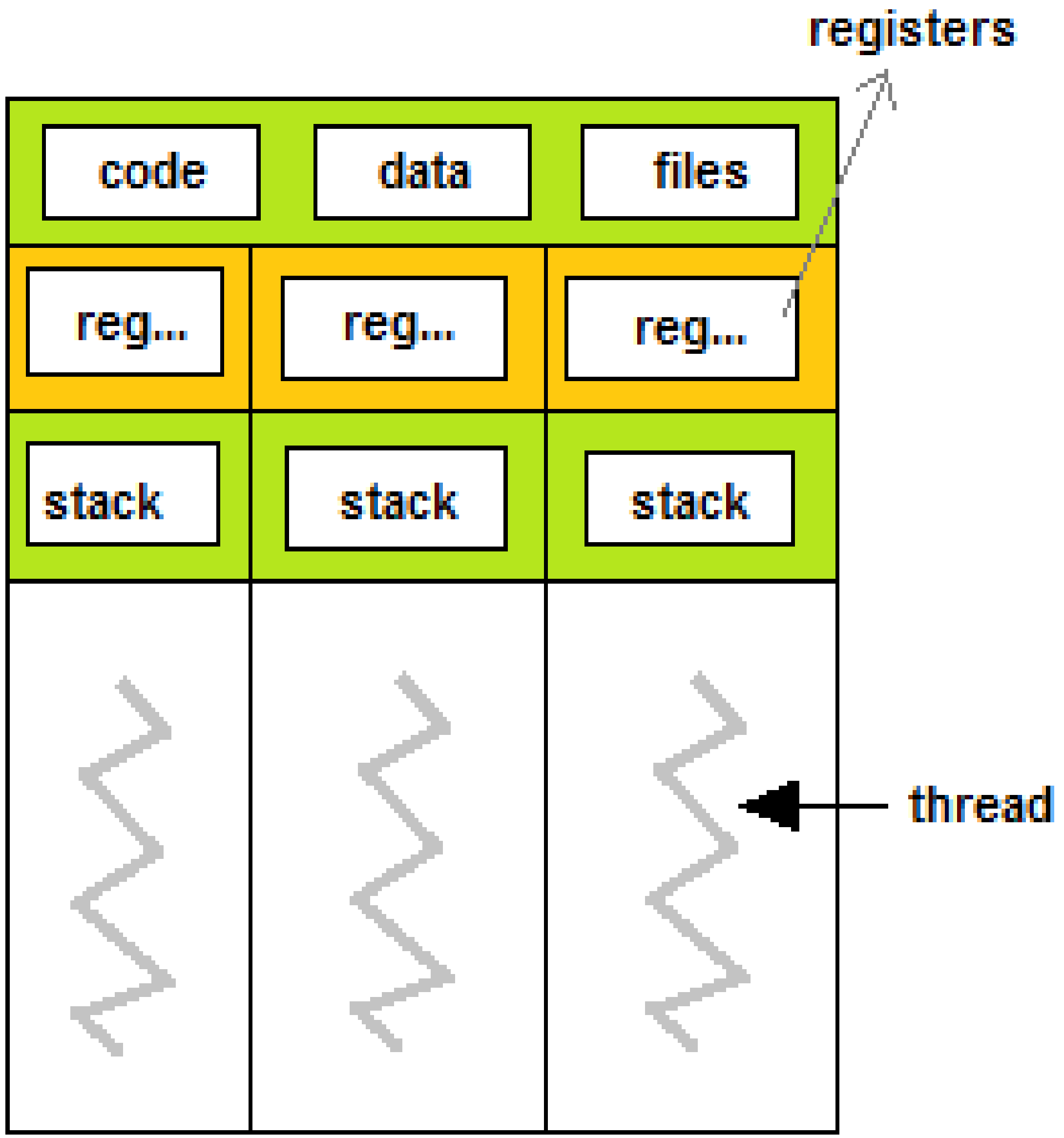
- **However, a thread itself is not a program. It cannot run on its own but runs within a program.**
- **On a multiprocessor computer, the system can simultaneously execute as many threads as there are processors on the computer.**
- **A *thread* is a basic unit of CPU utilization and is a single sequential flow of execution of tasks of a process.**

- **The program counter mainly keeps track of which instruction to execute next, a set of registers mainly hold its current working variables, and a stack mainly contains**
- **Threads are also known as Lightweight processes.**
- **Threads are mainly used to improve the performance of an operating system just by reducing the overhead thread.**
- **There can be more than one thread inside a process.**
- **Each thread of the same process makes use of a separate program counter and a stack of activation records and control blocks.**

- **Threads provide a suitable foundation for the parallel execution of applications on shared-memory multiprocessors.**
- **For example, in a browser, many tabs can be viewed as threads.**
- **MS Word uses many threads - formatting text from one thread, processing input from another thread, etc.**



single-threaded process



multithreaded process

Need of Thread:

- It takes far less time to create a new thread in an existing process than to create a new process.
- Threads can share the common data; they do not need to use Inter- Process communication.
- Context switching is faster when working with threads.
- It takes less time to terminate a thread than a process.

Types of Threads

- **In the operating system, there are two types of threads.**
- **User-level thread**
- **Kernel level thread**

User-level thread

- The operating system does not recognize the user-level thread.
- User threads can be easily implemented by the user.
- If a user performs a user-level thread blocking operation, the whole process is blocked.
- examples: Java thread, POSIX threads, etc.

Advantages of User-level threads

- **The user threads can be easily implemented than the kernel thread.**
- **User-level threads can be applied to such types of operating systems that do not support threads at the kernel-level.**
- **It is faster and efficient.**
- **Context switch time is shorter than the kernel-level threads.**
- **It does not require modifications of the operating system.**
- **It is simple to create, switch, and synchronize threads without the intervention of the process.**

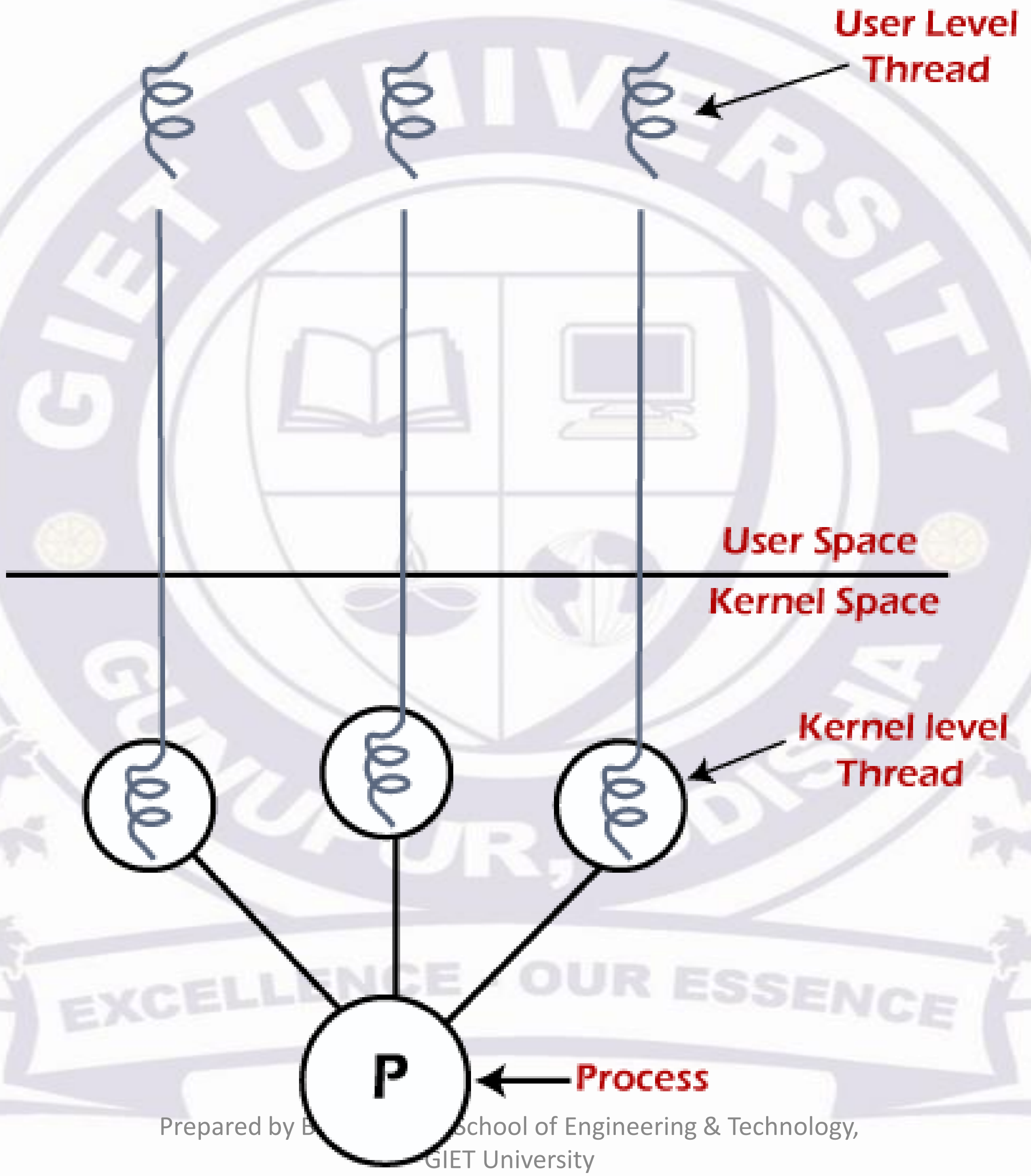
Disadvantages of User-level threads

- **User-level threads lack coordination between the thread and the kernel.**
- **If a thread causes a page fault, the entire process is blocked.**

Kernel level thread

- The kernel-level thread is implemented by the OS.
- There is a thread control block and process control block in the system for each thread and process in the kernel-level thread.
- The kernel knows about all the threads and manages them.
- The kernel-level thread offers a system call to create and manage the threads from user-space.

- **The implementation of kernel threads is more difficult than the user thread.**
- **Context switch time is longer in the kernel thread.**
- **If a kernel thread performs a blocking operation, the Banky thread execution can continue. Example: Window Solaris.**


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

Advantages of Kernel-level threads

- **The kernel-level thread is fully aware of all threads.**
- **The scheduler may decide to spend more CPU time in the process of threads being large numerical.**
- **The kernel-level thread is good for those applications that block the frequency.**

Disadvantages of Kernel-level threads

- **The kernel thread manages and schedules all threads.**
- **The implementation of kernel threads is difficult than the user thread.**
- **The kernel-level thread is slower than user-level threads.**

User Level threads	Kernel Level Threads
These threads are implemented by users.	These threads are implemented by Operating systems
These threads are not recognized by operating systems,	These threads are recognized by operating systems,
In User Level threads, the Context switch requires no hardware support.	In Kernel Level threads, hardware support is needed.
These threads are mainly designed as dependent threads.	These threads are mainly designed as independent threads.
In User Level threads, if one user-level thread performs a blocking operation then the entire process will be blocked.	On the other hand, if one kernel thread performs a blocking operation then another thread can continue the execution.
Example of User Level threads: Java thread, POSIX threads. **POSIX** stands for Portable Operating System Interface, and is an IEEE standard designed to facilitate application portability.	Example of Kernel level threads: Window Solaris. (**operating system** for Oracle Database and Java applications
Implementation of User Level thread is done by a thread library and is easy.	While the Implementation of the kernel-level thread is done by the operating system and is complex.
This thread is generic in nature and can run on any operating system.	This is specific to the operating system.

Benefits of Threads

- **Enhanced throughput of the system:** When the process is split into many threads, and each thread is treated as a job, the number of jobs done in the unit time increases. That is why the throughput of the system also increases.
- **Effective Utilization of Multiprocessor system:** When you have more than one thread in one process, you can schedule more than one thread in more than one processor.

- **Faster context switch:** The context switching period between threads is less than the process context switching. The process context switch means more overhead for the CPU.
- **Responsiveness:** When the process is split into several threads, and when a thread completes its execution, that process can be responded to as soon as possible.

- **Communication:** Multiple-thread communication is simple because the threads share the same address space, while in process, we adopt just a few exclusive communication strategies for communication between two processes.
- **Resource sharing:** Resources can be shared between all threads within a process, such as code, data, and files. Note: The stack and register cannot be shared between threads. There is a stack and register for each thread.

Process	Thread
A Process simply means any program in execution.	Thread means a segment of a process.
The process consumes more resources	Thread consumes fewer resources.
The process requires more time for creation.	Thread requires comparatively less time for creation than process.
The process is a heavyweight process	Thread is known as a lightweight process
The process takes more time to terminate	The thread takes less time to terminate.
Processes have independent data and code segments	A thread mainly shares the data segment, code segment, files, etc. with its peer threads
The process takes more time for context switching	The thread takes less time for context switching
Communication between processes needs more time as compared to thread	Communication between threads needs less time as compared to processes
For some reason, if a process gets blocked then the remaining processes can continue their execution	In case if a user-level thread gets blocked, all of its peer threads also get blocked.

Multithreading Models

- There must exist a relationship between User threads & Kernel threads.
- Some operating system provide a combined user level thread and Kernel level thread facility.
- Solaris is a good example of this combined approach.
- There are three common ways of establishing this relationship.

1. Many to one relationship
2. One to one relationship
3. Many to many relationship

Many to one relationship

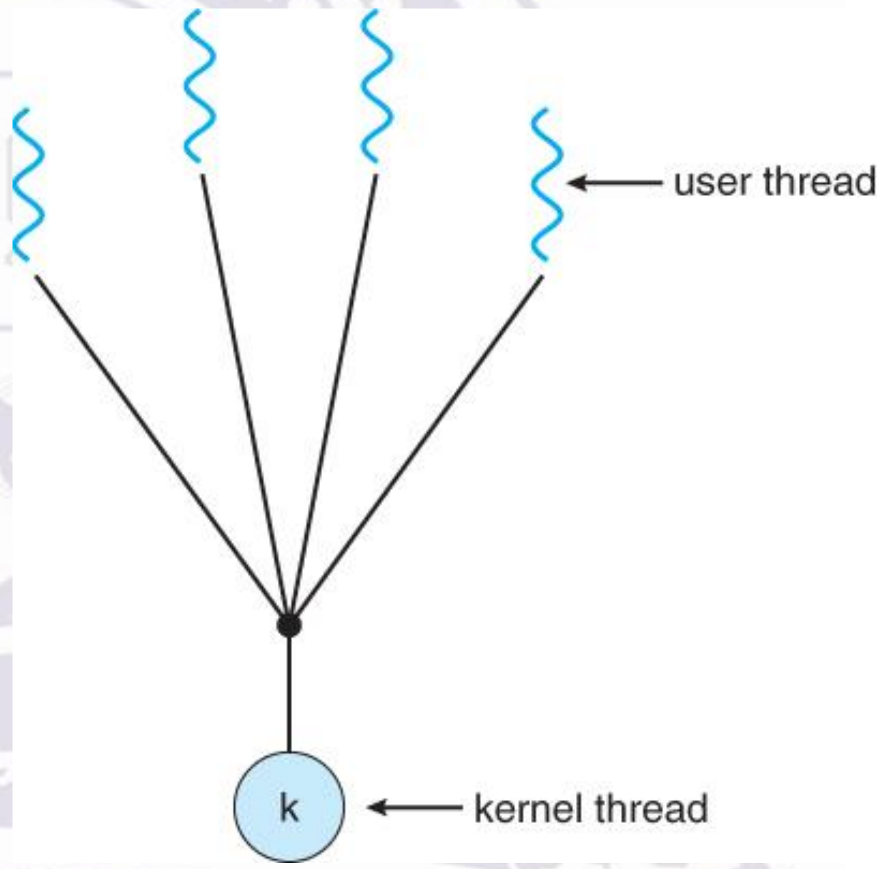
- **Many-to-one model maps many user level threads to one Kernel-level thread.**
- **Thread management is done by the thread library in user space, so it is efficient.**
- **Only one thread can access the Kernel at a time.**

Disadvantages

- **The entire process will block if thread makes a blocking system call.**

  (Blocking means waiting for some event, such as completion of an I/O operation)

- **Because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.**

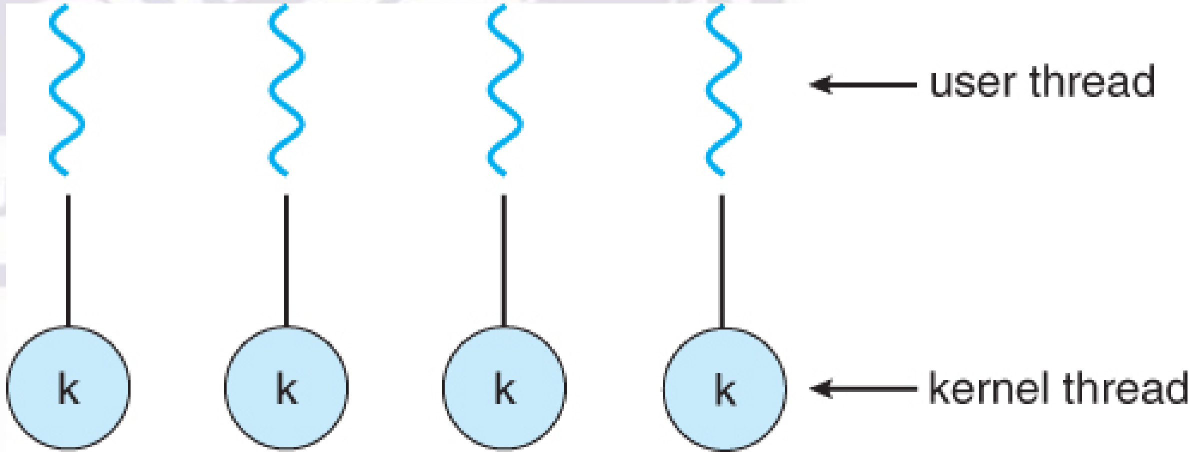


One to One Model

- There is one-to-one relationship , one user thread exactly mapped with one kernel thread.
- In this case if one user thread blocks then entire process will not blocked.
- Also allows multiple threads to run in parallel on multiprocessors.

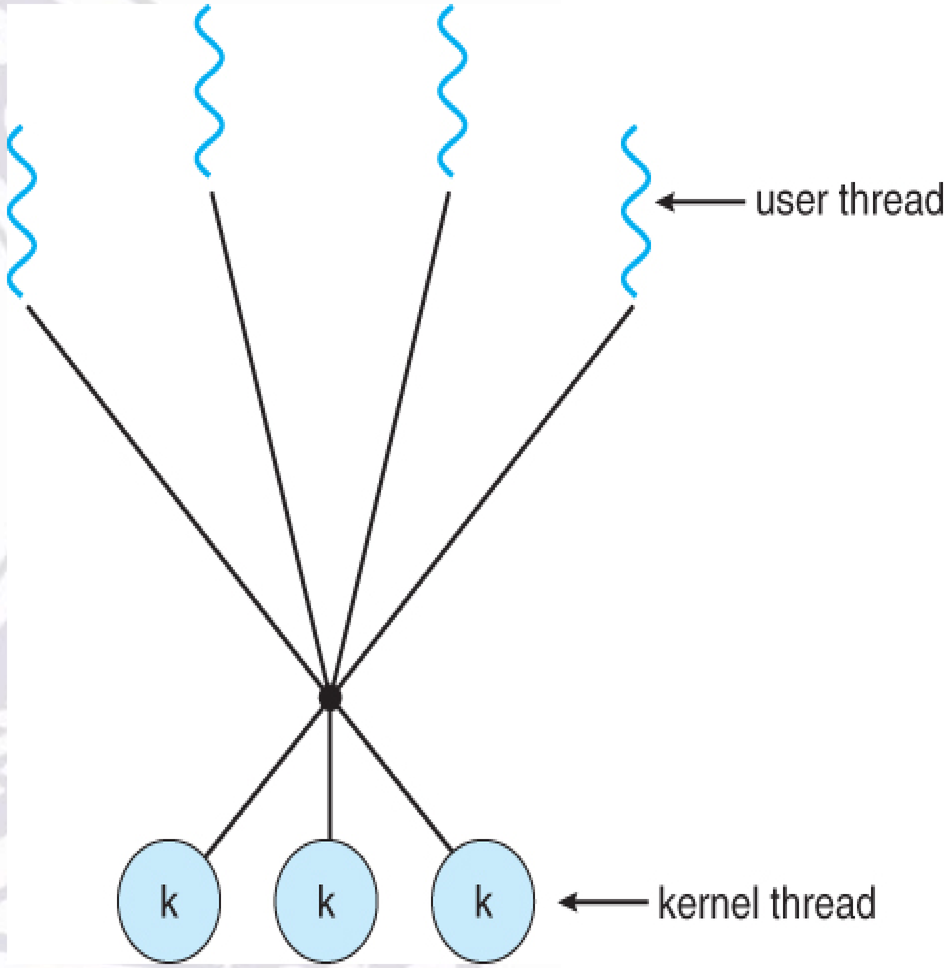
Disadvantages:

- Creating a user thread requires creating the corresponding kernel thread.
- Windows NT and windows 2000 use one to one relationship model.



Many to Many Model

- **The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.**
- **The no. of kernel threads may be specific to either a particular application or particular machine.**
- **developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine.**
- **Also when a thread performs a blocking system call, the kernel can schedule another thread for execution.**



Process Scheduling

Introduction

- **The objective of multiprogramming is to have some process running at all time , to maximize CPU utilization.**
- **The objective of time sharing is to switch the CPU among processes frequently that users can interact with each program while it is running.**
- **To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU.**

- **For a single processor system, there will never be more than one running processor.**
- **If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.**

Objective of Process Scheduling

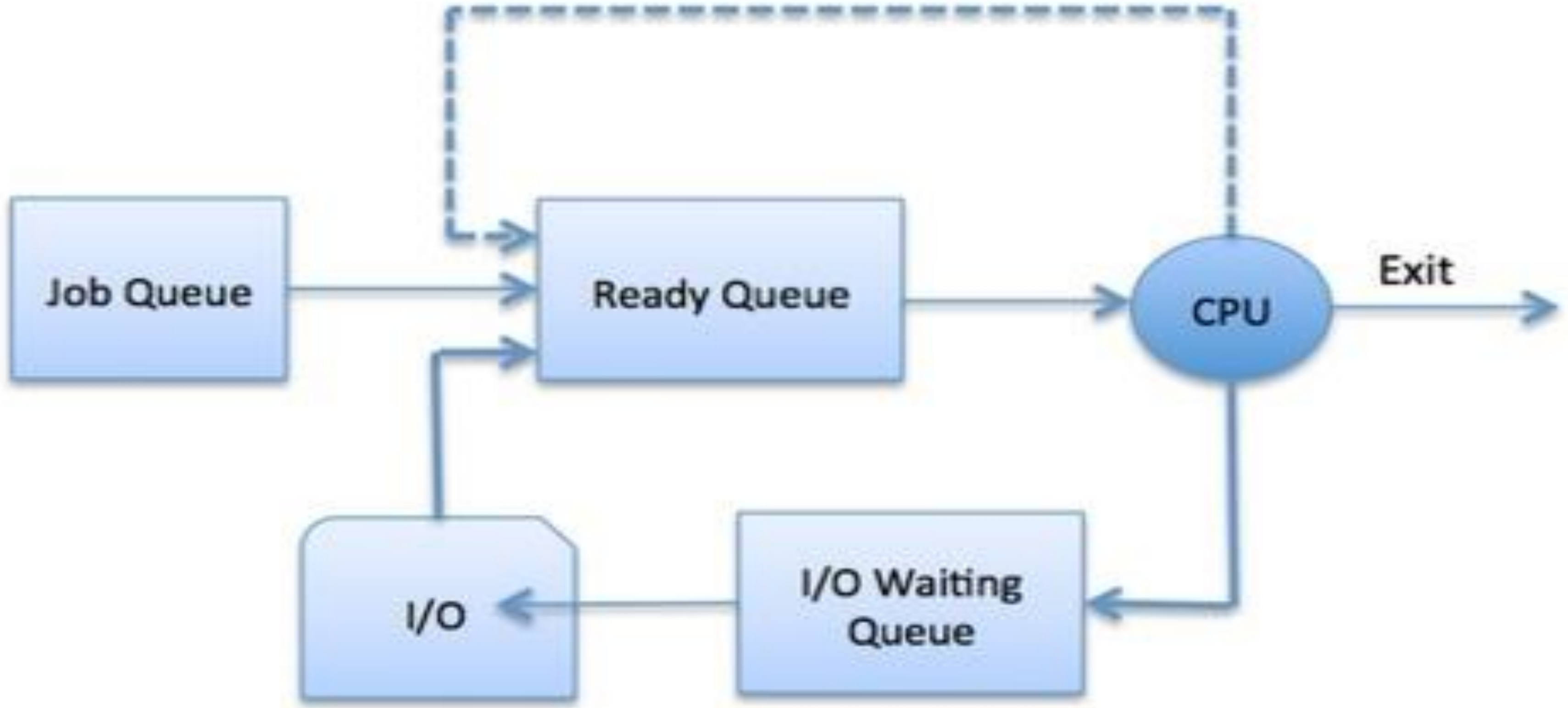
- **Max CPU utilization** *[Keep CPU as busy as possible]*
  Max throughput *[No. of processes that complete their execution per time unit]*
- **Min turnaround time** *[Time taken by a process to finish execution]*
- **Min waiting time** *[Time a process waits in ready queue]*
- **Min response time** *[Time when a process produces first response]*

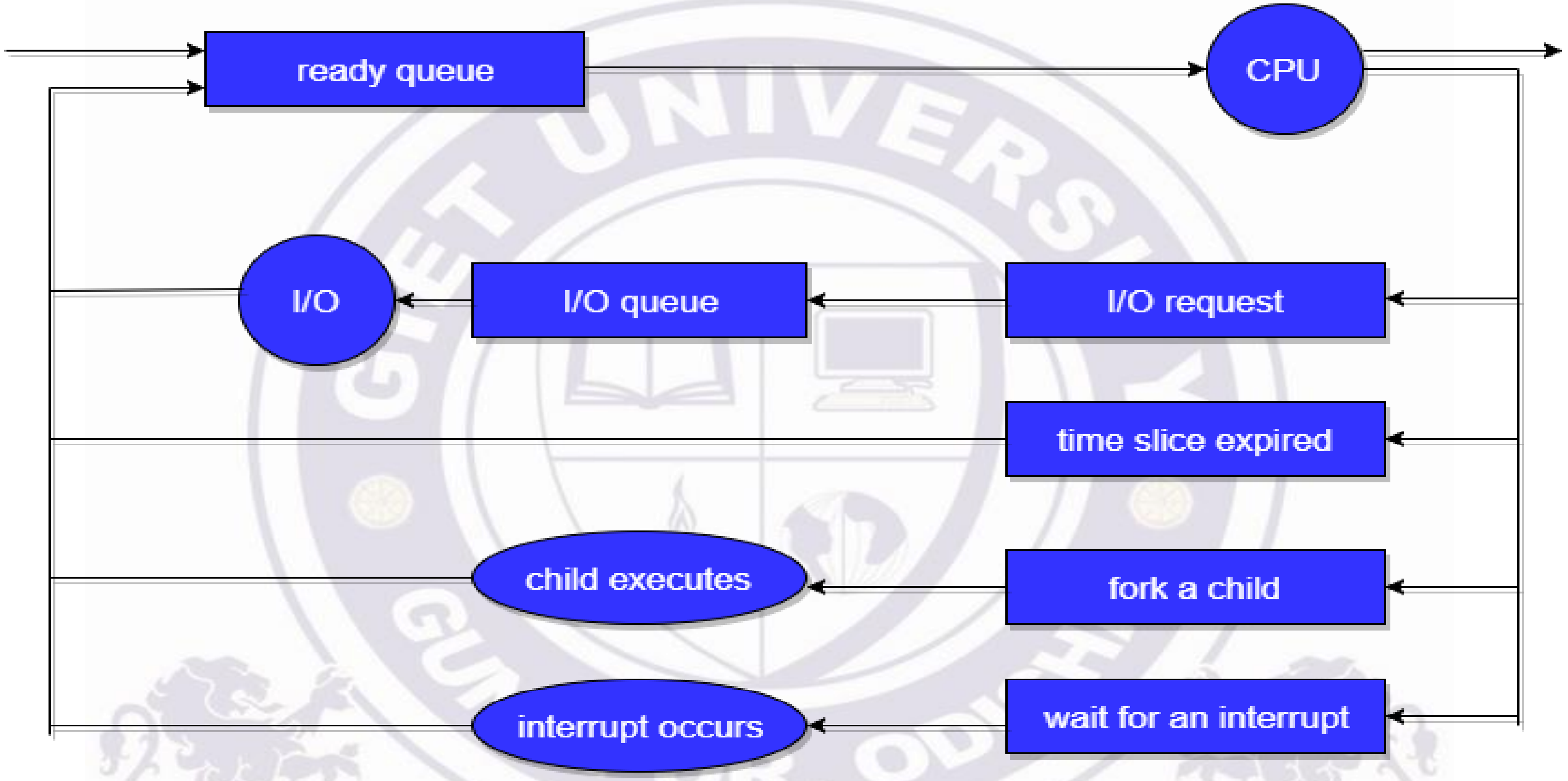
- ***"The act of determining which process is in the ready state, and should be moved to the running state is known as Process Scheduling".***
- **The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs.**

Scheduling Queues

- **In order to help the process scheduling, we have Scheduling queues.**
- **The OS maintains a separate queue for each of the process states.**
- **There are two queues: Job queue and Ready queue.**
- **As processes enter the system, they are put into a *job queue,* which consists of all processes in the system.**
- **It is maintained in the secondary memory.**

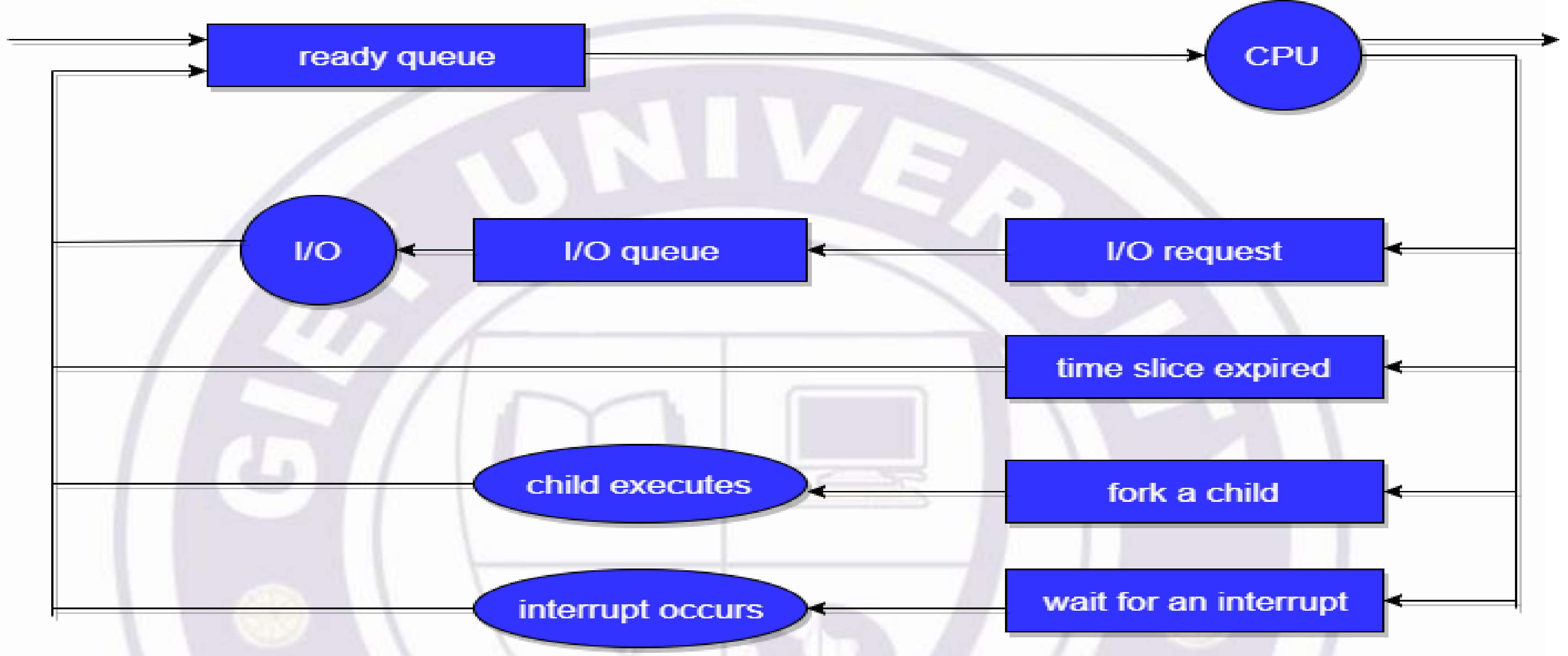
- ***Ready queue*** **– This queue keeps a set of all processes residing in main memory, ready and waiting to execute.**
- **Ready queue is maintained in primary memory.**
- **The list of processes waiting for a particular I/O device constitute *device queue*.**



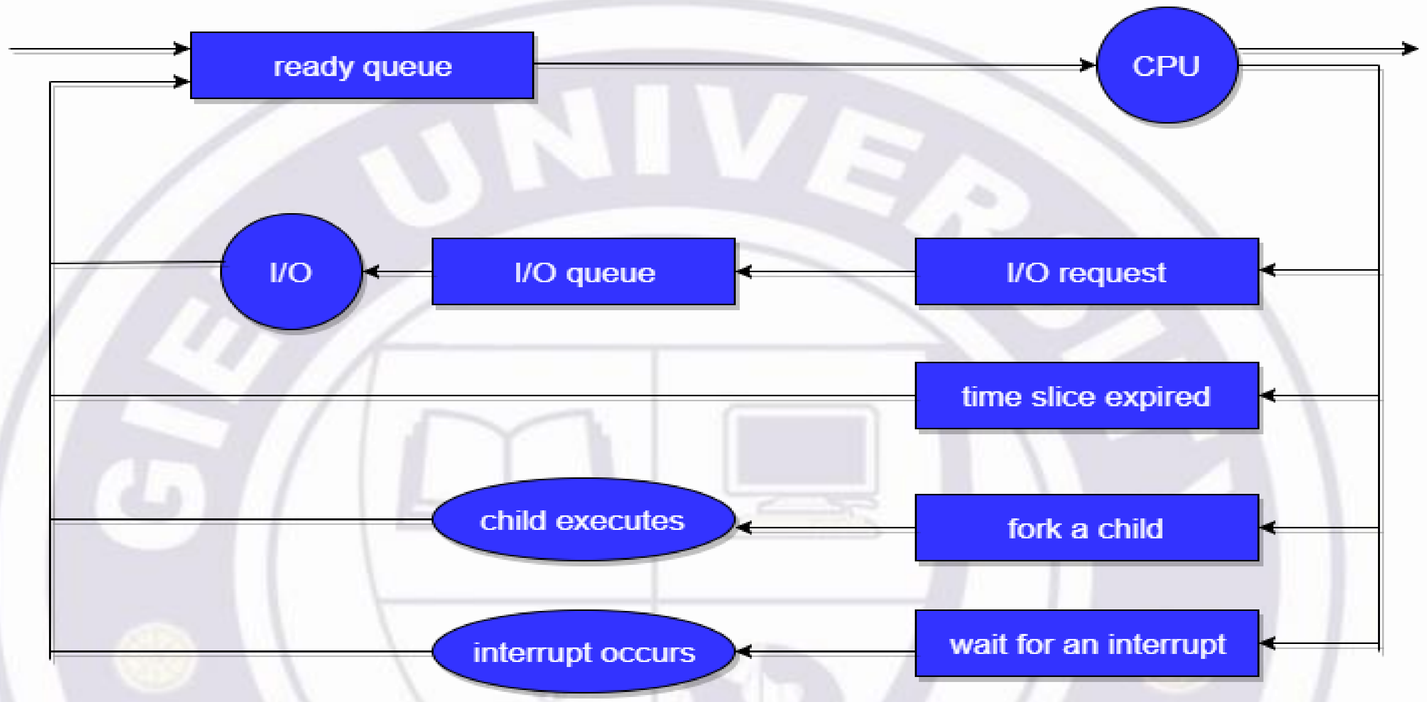


A common representation of process scheduling is a queueing diagram shown in fig.

Figure 3.6 Queueing-diagram representation of process scheduling



- **Each rectangular box represents a queue.**
- **The circles represent the resources that serve the queues.**
- **The arrows indicate the flow of processes in the system.**
- **A new process is initially put in the ready queue. It waits there until it is selected for execution, or dispatched.**



- **Once the process is allocated the CPU and is executing, one of several events could occur:**
  - ***The process could issue an I/O request and then be placed in an I/O queue.***
  - ***The process could create a new child process and wait for the child's termination.***
  - ***The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.***

- **In the first two cases, the process eventually switches from the waiting state to the ready state and is then put back in the ready queue.**
- **A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.**

Schedulers

- **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***

Long Term Scheduler

- **It is also called a job scheduler.**
- **A long-term scheduler determines which programs are admitted to the system for processing.**
- **It selects processes from the queue and loads them into memory for execution.**
- **Process loads into the memory for CPU scheduling.**
- **The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound.**

- **It also controls the degree of multiprogramming(*the number of processes in memory*).**
- **When a process changes the state from new to ready, then there is use of long-term scheduler.**
- **It is important that the long-term scheduler make a careful selection.**
- **In general, most processes can be described as either I/O bound or CPU bound.**

- An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations.
- A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations

Short Term Scheduler

- It is also called as CPU scheduler.
- Its main objective is to increase system performance in accordance with the chosen set of criteria.
- It is the change of ready state to running state of the process.

- It selects a process among the processes that are ready to execute and allocates CPU to one of them.
- It, also known as dispatchers, make the decision of which process to execute next.
- These schedulers are faster than long-term schedulers.

Medium Term Scheduler

- Medium-term scheduling is a part of swapping.
- It removes the processes from the memory.
- It reduces the degree of multiprogramming.
- The medium-term scheduler is in-charge of handling the swapped out-processes.

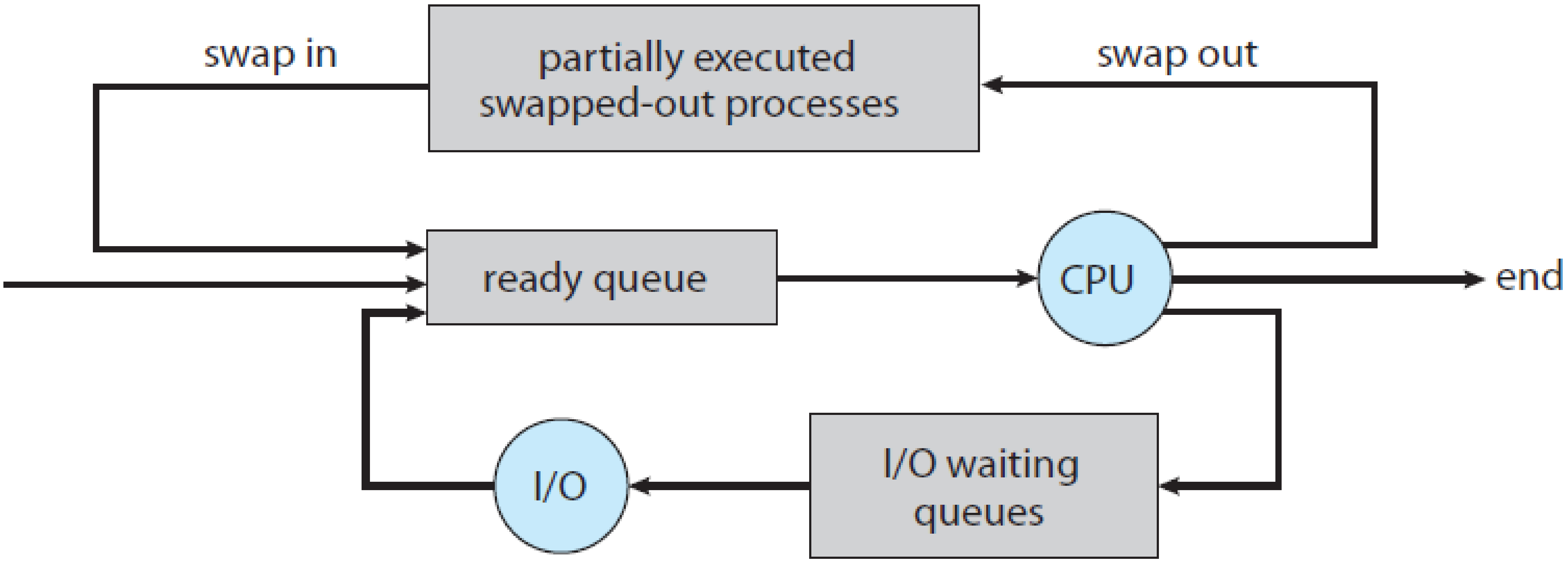
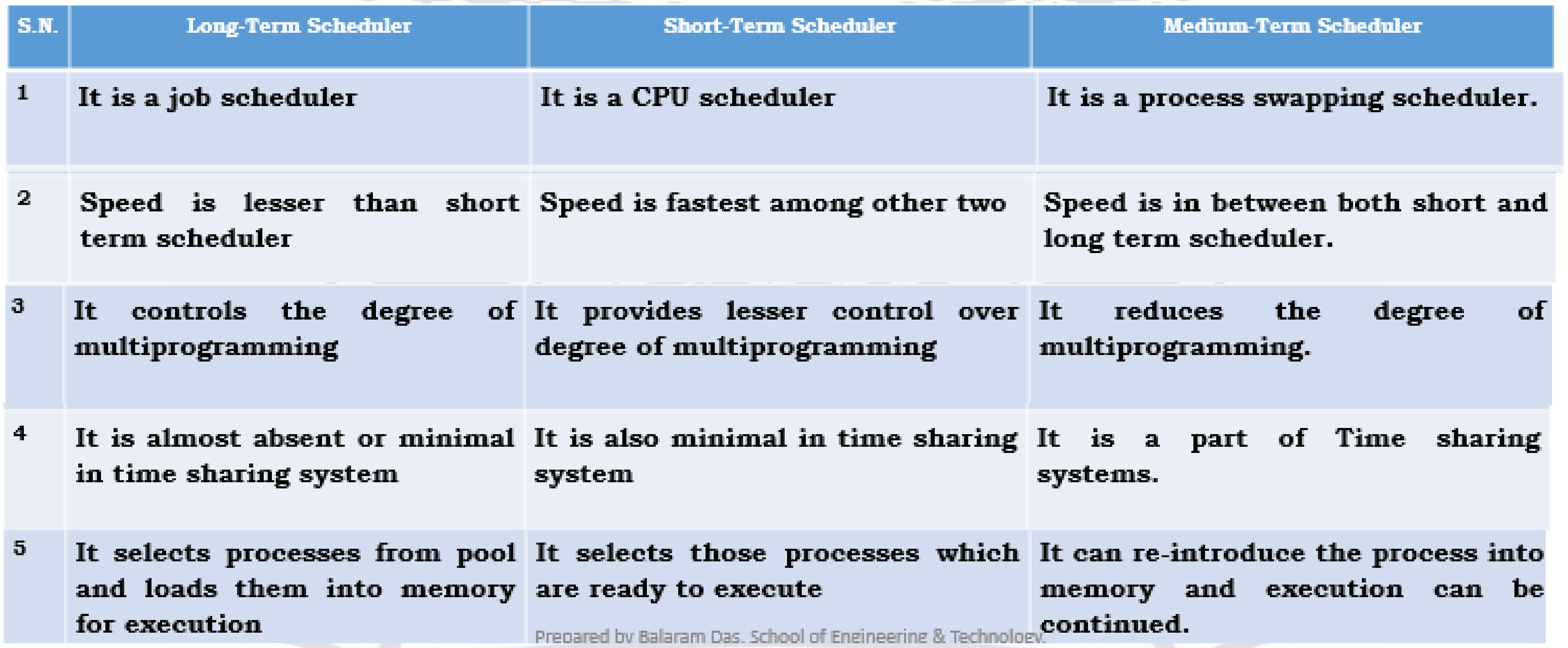


Figure: Addition of medium-term scheduling to the queueing diagram.

- **A running process may become suspended if it makes an I/O request.**
- **A suspended processes cannot make any progress towards completion.**
- **In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage.**
- **This process is called swapping, and the process is said to be swapped out or rolled out.**
- **Swapping may be necessary to improve the process mix.**

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.

Prepared by Balaram Das, School of Engineering & Technology,

Context Switch

- A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time.
- Using this technique, a context switcher enables multiple processes to share a single CPU.
- Context switching is an essential part of a multitasking operating system features.

- **When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block.**
- **After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc.**
- **At that point, the second process can start executing.**

- **Context switches are computationally intensive since register and memory state must be saved and restored.**
- **To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers.**

- When the process is switched, the following information is stored for later use.
- Program Counter
- Scheduling information
- Base and limit register value
- Currently used register
- Changed State
- I/O State information
- Accounting information

Context switching	Swapping
It is a procedure for storing the state of an old process and loading it into a new process.	Essentially, it is a method of replicating the entire process.
A context switch occurs when the kernel switches contexts when it transfers control of the CPU from one process to another already ready to run state.	Swapping happens when the entire process is moved to the disk.
A context switch determines whether a process is in the pause mode.	When it comes to Swapping, It deals with memory, how much memory is being swapped.
The context switch toggles the process from running to ready states, while the dispatcher is responsible for allocating CPU resources to processes present in the ready queue.	It is an OS term that we use for referring to the exchange of data between the disk and the main memory.
Active processes do context switching.	Inactive processes do swapping.
It offers a higher degree of multi-tasking.	It provides a more significant degree of multiprogramming.
It helps to get better utilization of the operating system.	It helps to get better utilization of memory.

Pre-emptive Scheduling

Introduction

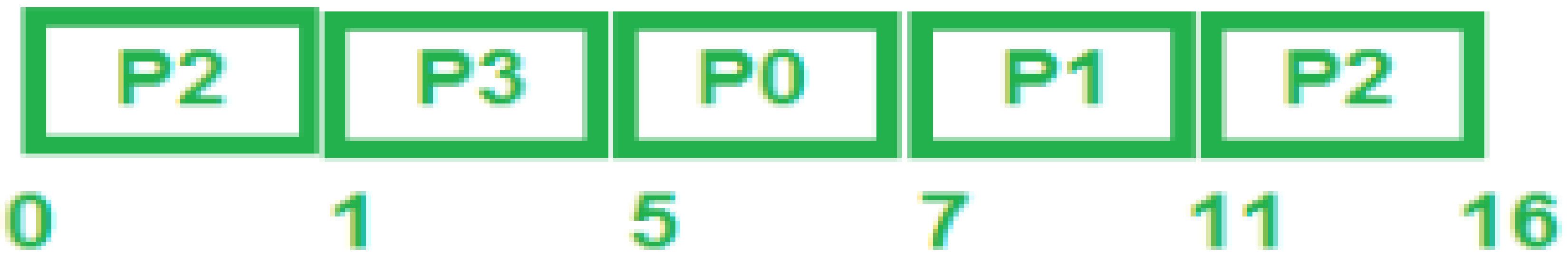
- **It is the responsibility of CPU scheduler to allot a process to CPU whenever the CPU is in the idle state.**
- **The CPU scheduler selects a process from ready queue and allocates the process to CPU.**
- ***"The scheduling which takes place when a process switches from running state to ready state or from waiting state to ready state is called Pre-emptive Scheduling".***

- The resources (mainly CPU cycles) are allocated to the process for a limited amount of time and then taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining.
- That process stays in the ready queue till it gets its next chance to execute.
- Algorithms based on pre-emptive scheduling are: Round Robin (RR), Shortest Remaining Time First (SRTF), Priority (pre-emptive version), etc.

- **Burst time:** It is the amount of time required by a process for executing on CPU. It is also called as execution time or running time.
- **Arrival Time:** Time at which the process arrives in the ready queue.
- **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

Process	Arrival Time	CPU Burst Time (in millisec.)
P0	3	2
P1	2	4
P2	0	6
P3	1	4



Preemptive Scheduling

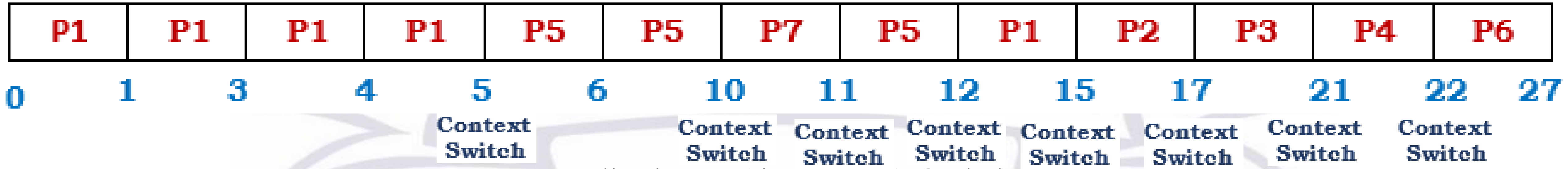
Explanation

- Firstly, the process **P2** comes at time **0**. So, the CPU is assigned to process **P2**.
- When process **P2** was running, process P3 arrived at time 1, and the remaining time for process **P2 (5 ms)** is greater than the time needed by process **P3 (4 ms)**. So, the processor is assigned to P3.
- When process **P3** was running, process **P1** came at time **2**, and the remaining time for process **P3 (3 ms)** is less than the time needed by processes **P1 (4 ms)** and **P2 (5 ms)**. As a result, **P3** continues the execution.
- When process **P3** continues the process, process **P0** arrives at time **3**. P3's remaining time **(2 ms)** is equal to **P0's** necessary time **(2 ms)**. So, process **P3** continues the execution.
- When process **P3** finishes, the CPU is assigned to **P0**, which has a shorter burst time than the other processes.
- After process **P0** completes, the CPU is assigned to process **P1** and then to process **P2**.

Lesser the number

Higher the priority

Processor	Arrival Time	Priority	Burst Time	Completion Time	Turnaround Time	Waiting Time	Response Time
P1	0	3	8	15	15	7	0
P2	1	4	2	17	16	14	14
P3	3	4	4	21	18	14	14
P4	4	5	1	22	18	17	17
P5	5	2	6	12	7	1	0
P6	6	6 (Lowest Priority)	5	27	21	16	16
P7	10	1 (Highest Priority)	1	11	1	0	0



Advantages of Pre-emptive Scheduling

- It is a more robust method because a process may not monopolize the processor.
- Each event causes an interruption in the execution of ongoing tasks.
- It improves the average response time.
- It is more beneficial when you use this method in a multi-programming environment.
- The operating system ensures that all running processes use the same amount of CPU.

Disadvantages of Pre-emptive Scheduling

- **It requires the use of limited computational resources.**
- **It takes more time suspending the executing process, switching the context, and dispatching the new incoming process.**
- **If several high-priority processes arrive at the same time, the low-priority process would have to wait longer.**

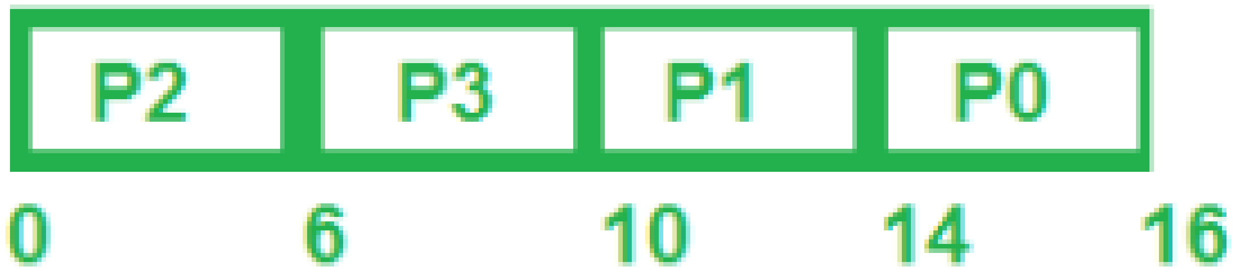
Non-Pre-emptive Scheduling

Introduction

- Non-pre-emptive Scheduling is used when a process terminates, or a process switches from running to the waiting state.
- In this scheduling, once the resources (CPU cycles) are allocated to a process, the process holds the CPU till it gets terminated or reaches a waiting state.

- **In this case, non-pre-emptive scheduling does not interrupt a process running CPU in the middle of the execution.**
- **Instead, it waits till the process completes its CPU burst time, and then it can allocate the CPU to another process.**
- **Algorithms based on non-pre-emptive scheduling are:_**
- **Shortest Job First (SJF basically non pre-emptive) and Priority (non pre-emptive version), etc.**

Process	Arrival Time	CPU Burst Time (in millisec.)
P0	3	2
P1	2	4
P2	0	6
P3	1	4



Non-Preemtive Scheduling

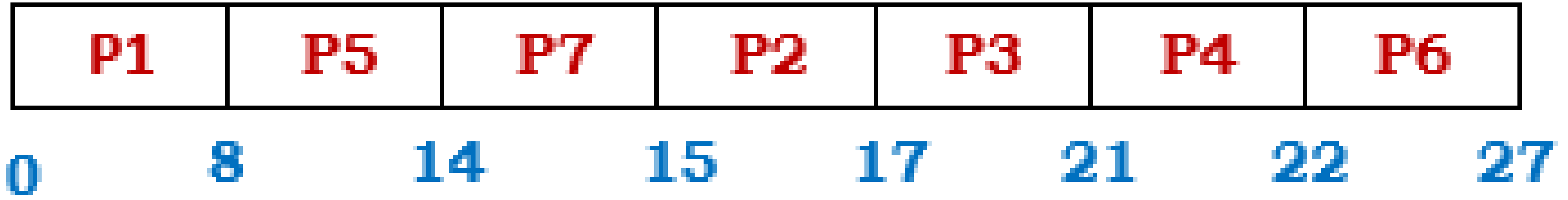
Explanation

- The process **P2** comes at **0**, so the processor is assigned to process **P2** and takes **(6 ms)** to execute.
- All of the processes, **P0, P1**, and **P3**, arrive in the ready queue in between. But all processes wait till process **P2** finishes its CPU burst time.
- After that, the process that comes after process **P2**, i.e., **P3**, is assigned to the CPU until it finishes its burst time.
- When process **P1** completes its execution, the CPU is given to process **P0**.

Lesser the number

Higher the priority

Processor	Arrival Time	Priority	Burst Time	Completion Time	Turnaround Time	Waiting Time	Response Time
P1	0	3	8	8	8	0	0
P2	1	4	2	17	16	14	14
P3	3	4	4	21	18	14	14
P4	4	5	1	22	18	17	17
P5	5	2	6	14	9	3	3
P6	6	6 (Lowest Priority)	5	27	21	16	16
P7	10	1 (Highest Priority)	1	15	5	4	4



Advantages of Non-Pre-emptive Scheduling

- It provides a low scheduling overhead.
- It is a very simple method.
- It uses less computational resources.
- It offers high throughput.

Disadvantages of Non-Pre-emptive Scheduling

- It has a poor response time for the process.
- A machine can freeze up due to bugs.
- It can lead to starvation especially for those real-time tasks.
- It can make real-time and priority Scheduling difficult.

Parameter	PRE-EMPTIVE SCHEDULING	NON-PRE-EMPTIVE SCHEDULING
Basic	In this resources(CPU Cycle) are allocated to a process for a limited time.	Once resources(CPU Cycle) are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
Interrupt	Process can be interrupted in between.	Process can not be interrupted until it terminates itself or its time is up.
Starvation	If a process having high priority frequently arrives in the ready queue, a low priority process may starve.	If a process with a long burst time is running CPU, then later coming process with less CPU burst time may starve.
Overhead	It has overheads of scheduling the processes.	It does not have overheads.
Flexibility	flexible	rigid
Cost	cost associated	no cost associated
CPU Utilization	CPU utilization is high.	CPU utilization is low
Examples	Examples: Round Robin and Shortest Remaining Time First.	Examples: First Come First Serve and Shortest Job First.

Dispatcher

Introduction

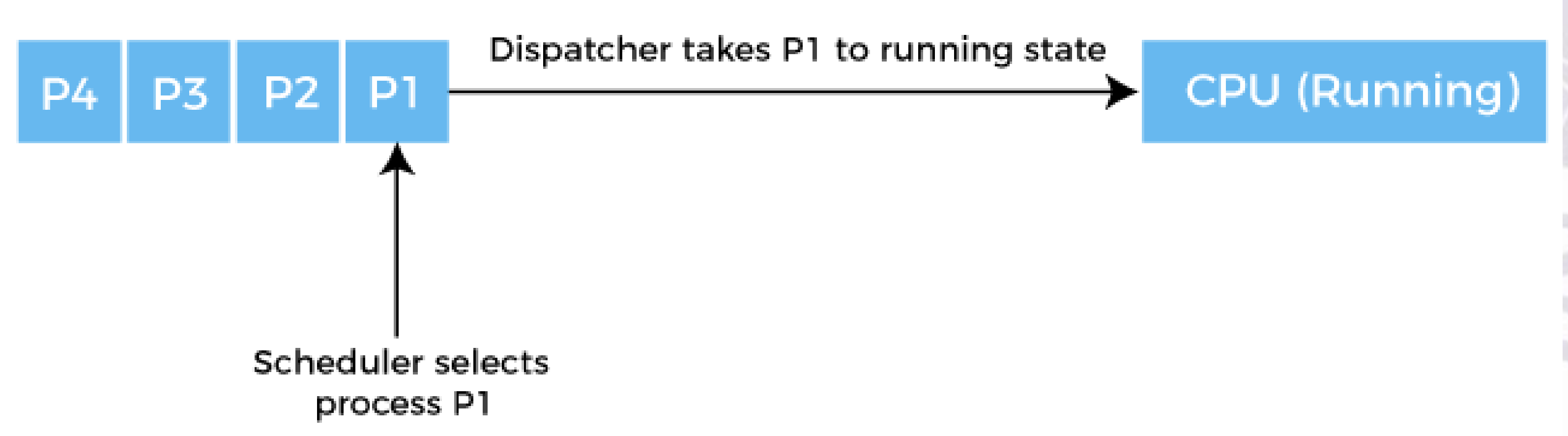
- A dispatcher is a special program which comes into play after the scheduler.
- When the scheduler completes its job of selecting a process, it is the dispatcher which performs the task of allocating the selected process to the CPU.

- ***“The dispatcher is the module that gives a process control over the CPU after it has been selected by the short-term scheduler.”***
- **This function involves the following:**
- **Switching context**
- **Switching to user mode**
- **Jumping to the proper location in the user program to restart that program**

- **When a running process goes to the waiting state for I/O operation etc., then the CPU is allocated to some other process.**
- **This switching of CPU from one process to the other is called context switching.**

- **When dispatching, the process changes from the ready state to the running state.**
- **The Dispatcher needs to be as fast as possible, as it is run on every context switch.**
- **The time consumed by the Dispatcher is known as *dispatch latency*.**

- **Example: There are 4 processes in the ready queue, P1, P2, P3, P4; Their arrival times are t0, t1, t2, t3 respectively.**
- **A First in First out (FIFO) scheduling algorithm is used.**
- **Because P1 arrived first, the scheduler will decide it is the first process that should be executed, and the dispatcher will remove P1 from the ready queue and give it to the CPU.**
- **The scheduler will then determine P2 to be the next process that should be executed, so when the dispatcher returns to the queue for a new process, it will take P2 and give it to the CPU.**
- **This continues in the same way for P3, and then P4.**


P4
P3
P2
P1
Dispatcher takes P1 to running state
CPU (Running)
Scheduler selects process P1

Scheduling Criteria

Introduction

- Different CPU scheduling algorithms have different properties, and the choice of a particular algorithm depends on the various factors.
- Many criteria have been suggested for comparing CPU scheduling algorithms.
- The criteria include the following:

- **CPU utilisation –**
- **The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible.**
- **Theoretically, CPU utilisation can range from 0 to 100% but in a real-time system, it varies from 40 to 90% depending on the load upon the system.**

- **Throughput –**
- **A measure of the work done by CPU is, the number of processes being executed and completed per unit time. This is called throughput.**
- **The throughput may vary depending upon the length or duration of processes.**

- **Turnaround time –**
- **For a particular process, an important criteria is how long it takes to execute that process.**
- **The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time.**

- **Waiting time –**
- **A scheduling algorithm does not affect the time required to complete the process once it starts execution.**
- **It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue.**

- **Response time –**
- **Response time is the amount of time it takes for the CPU to respond to a request made by a process.**
- **It is the duration between the arrival of a process and the first time it runs.**

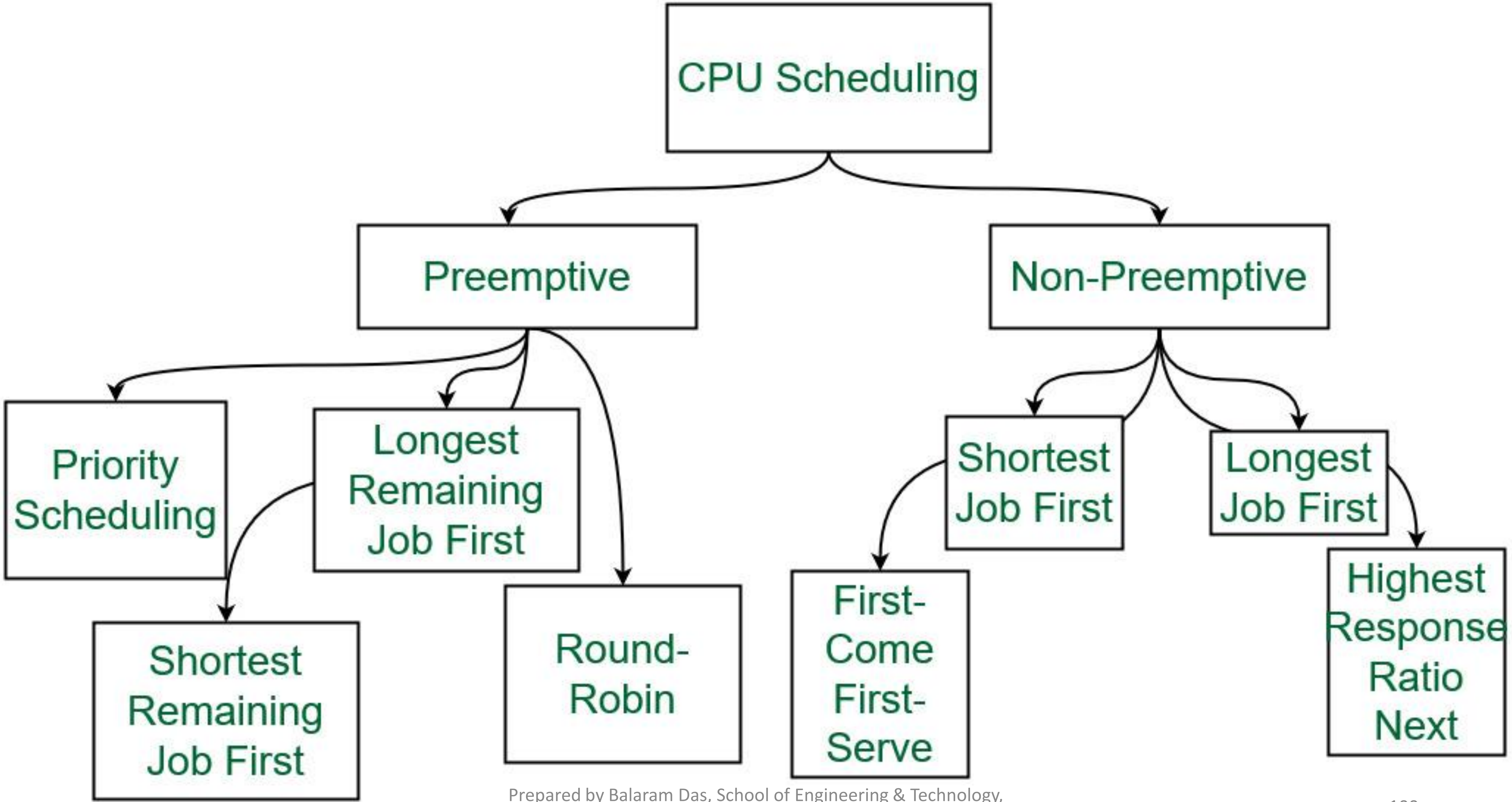
Scheduling algorithms

Why do we need scheduling?

- **A typical process involves both I/O time and CPU time.**
- **In a uni-programming system like MS-DOS, time spent waiting for I/O is wasted and CPU is free during this time.**
- **In multi programming systems, one process can use CPU while another is waiting for I/O.**
- **This is possible only with process scheduling.**

Scheduling Algorithm

Preemptive Scheduling	Non Preemptive scheduling
Shortest Remaining Time First (SRTF)	First Come First Serve (FCFS)
Longest Remaining Time First (LRTF)	Shortest Job First (SJF)
Round Robin	Longest Job First (LJF)
Priority based	Highest response Ratio Next (HRRN)
	Multilevel Queue


CPU Scheduling
Preemptive
Non-Preemptive
Priority Scheduling
Longest Remaining Job First
Shortest Remaining Job First
Round-Robin
Shortest Job First
Longest Job First
First-Come First-Serve
Highest Response Ratio Next

- **A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms.**
- **There are six popular process scheduling algorithms which we are going to discuss in this chapter –**
- **First-Come, First-Served (FCFS) Scheduling**
- **Shortest-Job-First (SJF) Scheduling**
- **Priority Scheduling**
- **Shortest Remaining Time**
- **Round Robin(RR) Scheduling**
- **Multiple-Level Queues Scheduling**

- **These algorithms are either non-pre-emptive or pre-emptive.**
- **Non-pre-emptive algorithms are designed so that once a process enters the running state, it cannot be pre-empted until it completes its allotted time,**
- **whereas the pre-emptive scheduling is based on priority where a scheduler may pre-empt a low priority running process anytime when a high priority process enters into a ready state.**

- **What are the different terminologies to take care of in any CPU Scheduling algorithm?**
- **Arrival Time:** Time at which the process arrives in the ready queue.
- **Completion Time:** Time at which process completes its execution.
- **Burst Time:** Time required by a process for CPU execution.
- **Turn Around Time:** Time Difference between completion time and arrival time.
- *Turn Around Time = Completion Time – Arrival Time*
- *Waiting Time = Turn Around Time – Burst Time*

- **The Purpose of a Scheduling algorithm**

1.Maximum CPU utilization

2.Fare allocation of CPU

3.Maximum throughput

4.Minimum turnaround time

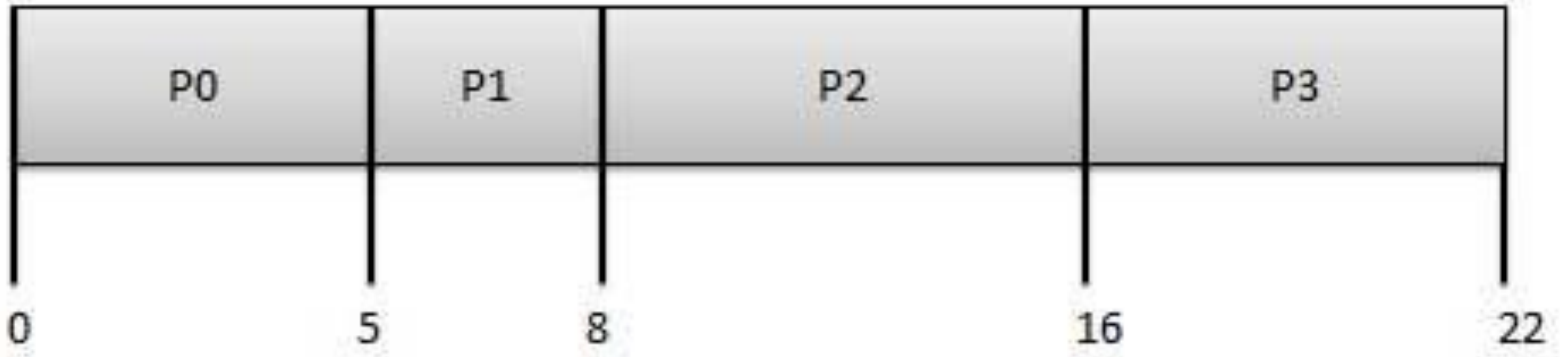
5.Minimum waiting time

6.Minimum response time

1. First Come First Serve (FCFS)

- It is a simple scheduling algorithm.
- It states that the process that requests the CPU first is allocated the CPU first and is implemented by using FIFO queue.
- It is a non-pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



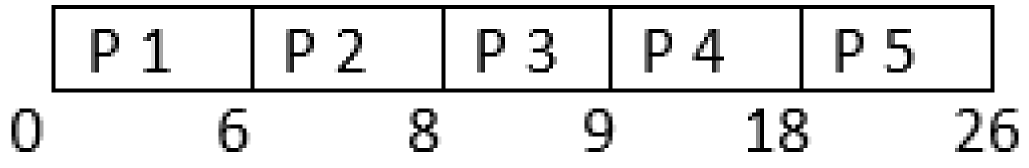
- **Wait time of each process is as follows −**

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

- **Average Wait Time: (0+4+6+13) / 4 = 5.75**

Process ID	Arrival Time	Burst Time	Completion Time	Turn around Time	Waiting Time
P1	0	6	6	6	0
P2	2	2	8	6	4
P3	3	1	9	6	5
P4	4	9	18	14	5
P5	5	8	26	21	13

Gantt Chart



- **Average Completion Time** = The Total Sum of Completion Times which is divided by the total number of processes is known as Average Completion Time.
- Average Completion Time =(CT1 + CT2 + CT3 + ... + CTn) / n
- Average CT = (6 + 8 +9 + 18 + 26) / 5 = 13.4

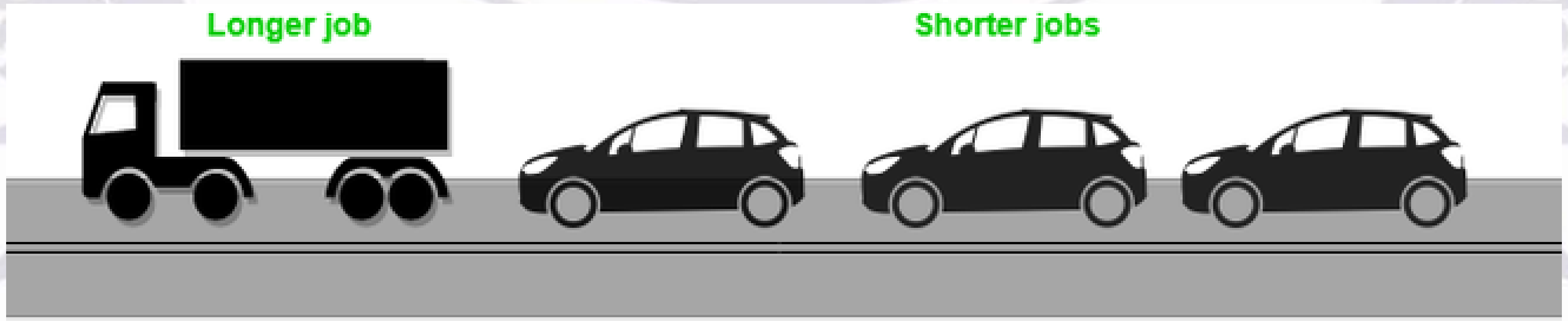
- **Average Turn Around Time** = The Total Sum of Turn Around Times which is divided by the total number of processes is known as Average Turn Around Time.
- Average Turn around time, TAT = (6 + 6 + 6 + 14 + 21)/5 = 10.6

- **Average Waiting Time** = The Total Sum of Waiting Times which is divided by the total number of processes is known as Average Waiting Time.
- Average Waiting Time = (WT1 + WT2 + WT3 + ... + WTn) / n
- Average WT = (0 + 4 + 5 + 5 + 13) / 5 = 5.4

- **Advantages of FCFS :**
- Simple
- Easy, useful and understandable
- First come, first served.

- **Disadvantages of FCFS :**
- Because of non-pre-emptive scheduling, the process will continue to run until it is finished.
- Short processes which are at the end of ready queue have to wait for a larger time.
- Throughput is not efficient.

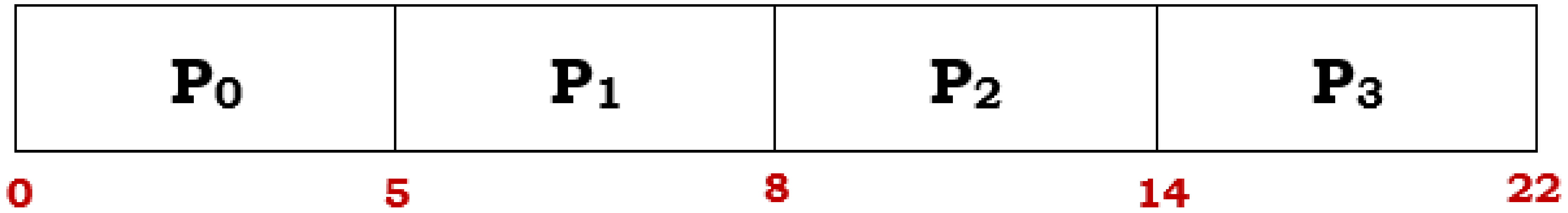
- FCFS suffers from **Convoy effect.**
- (*once CPU time has been allocated to a process, other processes can get CPU time only after the current process has finished. This property of FCFS scheduling leads to the situation called Convoy Effect.)*
- The average waiting time is much higher than the other algorithms.
- FCFS is very simple and easy to implement and hence not much efficient.



2. Shortest Job First (SJF)

- This is also known as shortest job next, or SJN.
- Process with the shortest burst time is scheduled first.
- If two processes have same burst time, then FCFS is used to break the tie.
- This is a non-pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Process	Arrival Time	Execution Time	Service Time	Waiting Time
P0	0	5	0	0 – 0 = 0
P1	1	3	5	5 - 1 = 4
P2	2	6	8	8 – 2 = 6
P3	3	8	14	14 - 3 = 11

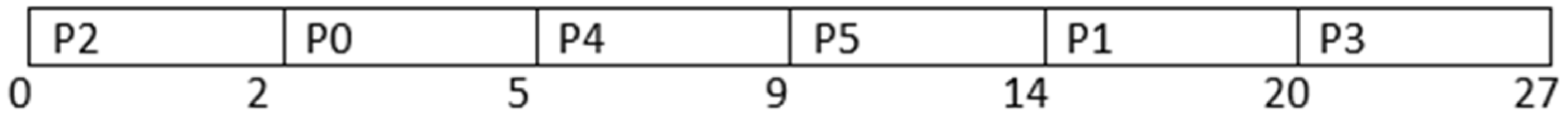


Average Wait Time: (0 + 4 + 6 + 11)/4 = 21 / 4 = 5.25

Non-Pre-Emptive Shortest Job First CPU Scheduling

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time TAT = CT - AT	Waiting Time WT = CT - BT
P0	1	3	5	4	1
P1	2	6	20	18	12
P2	0	2	2	2	0
P3	3	7	27	24	17
P4	2	4	9	7	4
P5	5	5	14	10	5

Gantt Chart:



- **Average Completion Time**

= (5 + 20 + 2 + 27 + 9 + 14) / 6 = 12.833

- **Average Waiting Time**

= (1 + 12 + 17 + 0 + 5 + 4) / 6 = 6.666

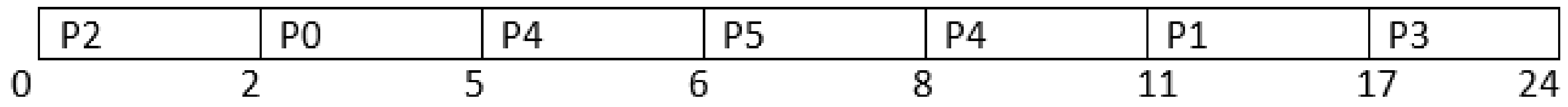
- **Average Turn Around Time**

= (4 +18 + 2 +24 + 7 + 10) / 6 = 6.166

Pre-Emptive Shortest Job First CPU Scheduling

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time TAT = CT - AT	Waiting Time WT = CT - BT
P0	1	3	5	4	1
P1	2	6	17	15	9
P2	0	2	2	2	0
P3	3	7	24	21	14
P4	2	4	11	9	5
P5	6	2	8	2	0

Gantt chart:



- **Average Completion Time**

= (5 + 17 + 2 + 24 + 11 +8) / 6 = 11.166

- **Average Turn Around Time**

= (4 +15 + 2 + 21 + 9 + 2) / 6 = 53 / 6 = 8.833

- **Average Waiting Time**

= (1 + 9 + 0 + 14 + 5 + 0) /6 = 29 / 6 = 4.833

- **Advantages**
- SJF is used because it has the least average waiting time than the other CPU Scheduling Algorithms
- SJF can be termed or can be called as long term CPU scheduling algorithm.
- **Disadvantages**
- Starvation is one of the negative traits Shortest Job First CPU Scheduling Algorithm exhibits.
- Often, it becomes difficult to forecast how long the next CPU request will take
- *Note: Starvation in operating system occurs when a process waits for an indefinite time to get the resource it requires. Starvation is a scheduler issue.*

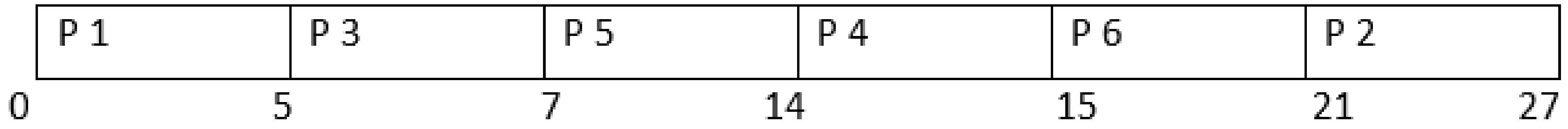
3. Priority Based Scheduling

- **Priority scheduling is a pre-emptive algorithm and one of the most common scheduling algorithms in batch systems.**
- **Each process is assigned a priority.**
- **Process with highest priority is to be executed first and so on.**
- **Processes with same priority are executed on first come first served basis.**
- **Priority can be decided based on memory requirements, time requirements or any other resource requirement.**

- **Advantages**
- The typical or average waiting time for Priority CPU Scheduling is shorter than First Come First Serve (FCFS).
- It is easier to handle Priority CPU scheduling
- It is less complex
- **Disadvantages**
- The Starvation Problem is one of the Pre emptive Priority CPU Scheduling Algorithm's most prevalent flaws. Because of this issue, a process must wait a longer period of time to be scheduled into the CPU. The hunger problem or the starvation problem is the name given to this issue.

Process	Arrival Time, AT	Execution Time or, Burst Time, BT	Prio rity	Completion Time, CT	TAT = CT-AT	Service Time	Waiting Time, TAT-BT or ST-AT
P1	0	5	5	5	5	0	0
P2	1	6	4	27	26	21	20
P3	2	2	0	7	5	5	3
P4	3	1	2	15	12	14	11
P5	4	7	1	14	10	7	3
P6	4	6	3	21	17	15	11

Gantt Chart:



- **Average Completion Time**

= (5 +27 +7 +15 +14 + 21) / 6 = 89 / 6 = 14.8333

- **Average Waiting Time**

= (0 + 20 + 3 + 11 + 3 + 11) / 6 = 48 / 6 = 7

- **Average Turn Around Time**

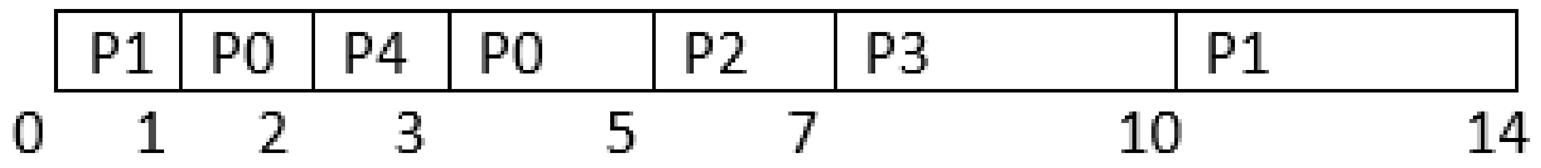
= (5 + 26 + 5 + 11 + 10 + 17) / 6 = 74 / 6 = 12.333

4. Round Robin Scheduling

- **Round Robin is the pre-emptive process scheduling algorithm.**
- **CPU is assigned to the process on the basis of FCFS for a fixed amount of time.**
- **This fixed amount of time is called as time quantum or time slice.**
- **After the time quantum expires, the running process is preempted and sent to the ready queue.**
- **Then, the processor is assigned to the next arrived process.**
- **Context switching is used to save states of pre-empted processes.**

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
P0	1	3	5	4	1
P1	0	5	14	14	9
P2	3	2	7	4	2
P3	4	3	10	6	3
P4	2	1	3	1	0

Gantt Chart:



- **Average Completion Time**

= (5+14+7+10+3)/5 = 39/5 = 7.8

- **Average Turn Around Time**

= (4+14+4+6+1)/5 = 29/5 = 4.8

- **Average Waiting Time** = (1+9+2+3+0)/5 = 15/5 = 3

- **Advantages-**
- It gives the best performance in terms of average response time.
- It is best suited for time sharing system, client server architecture and interactive system.
- **Disadvantages-**
- It leads to starvation for processes with larger burst time as they have to repeat the cycle many times.
- Its performance heavily depends on time quantum.
- Priorities can not be set for the processes.

- *With decreasing value of time quantum,*
- Number of context switch increases
- Response time decreases
- Chances of starvation decreases
- *Thus, smaller value of time quantum is better in terms of response time.*

- *With increasing value of time quantum,*
- Number of context switch decreases
- Response time increases
- Chances of starvation increases
- *Thus, higher value of time quantum is better in terms of number of context switch.*

- With increasing value of time quantum, Round Robin Scheduling tends to become FCFS Scheduling.
- When time quantum tends to infinity, Round Robin Scheduling becomes FCFS Scheduling.
- The performance of Round Robin scheduling heavily depends on the value of time quantum.
- The value of time quantum should be such that it is neither too big nor too small.

Example-01:

Consider the set of 5 processes whose arrival time and burst time are given below-

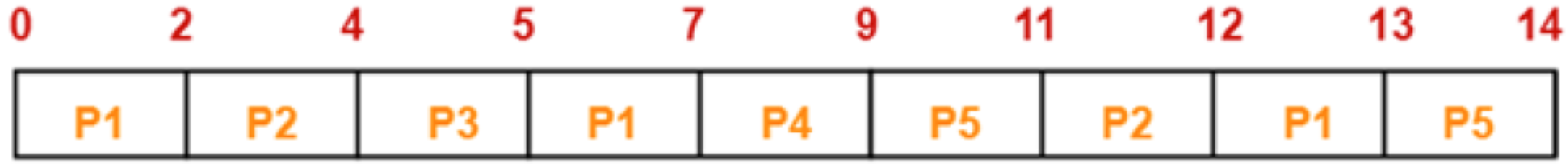
Process Id	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

If the CPU scheduling policy is Round Robin with time quantum = 2 unit, calculate the average waiting time and average turnaround time.

Solution-

Ready Queue- P5, P1, P2, P5, P4, P1, P3, P2, P1

Gantt Chart-



Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	13	13 – 0 = 13	13 – 5 = 8
P2	12	12 – 1 = 11	11 – 3 = 8
P3	5	5 – 2 = 3	3 – 1 = 2
P4	9	9 – 3 = 6	6 – 2 = 4
P5	14	14 – 4 = 10	10 – 3 = 7

Now,

- Average Turn Around time = (13 + 11 + 3 + 6 + 10) / 5 = 43 / 5 = 8.6 unit
- Average waiting time = (8 + 8 + 2 + 4 + 7) / 5 = 29 / 5 = 5.8 unit

Problem-02:

Consider the set of 6 processes whose arrival time and burst time are given below-

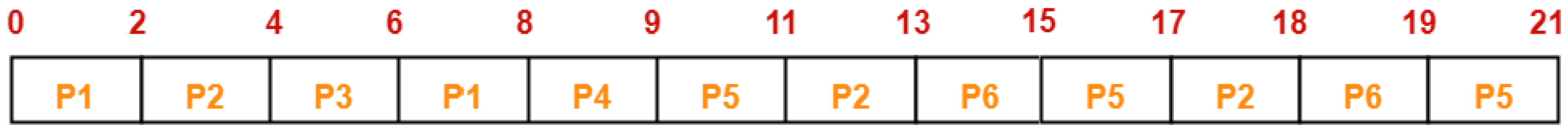
Process Id	**Arrival time**	**Burst time**
P1	0	4
P2	1	5
P3	2	2
P4	3	1
P5	4	6
P6	6	3

If the CPU scheduling policy is Round Robin with time quantum = 2, calculate the *average waiting time* and *average turnaround time.*

Solution-

Ready Queue- **P5, P6, P2, P5, P6, P2, P5, P4, P1, P3, P2, P1**

Gantt chart-



Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	**Exit time**	**Turn Around time**	**Waiting time**
P1	8	8 – 0 = 8	8 – 4 = 4
P2	18	18 – 1 = 17	17 – 5 = 12
P3	6	6 – 2 = 4	4 – 2 = 2
P4	9	9 – 3 = 6	6 – 1 = 5
P5	21	21 – 4 = 17	17 – 6 = 11
P6	19	19 – 6 = 13	13 – 3 = 10

Now,

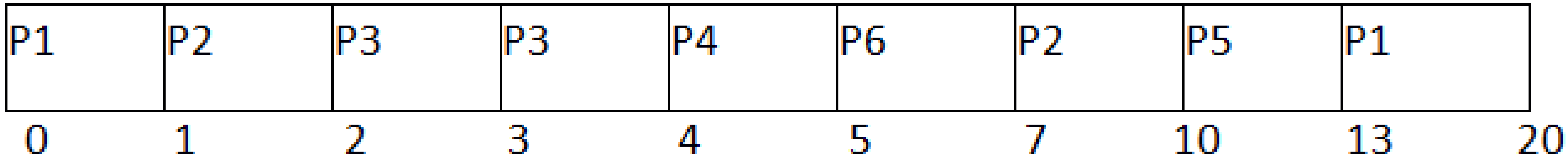
- Average Turn Around time = (8 + 17 + 4 + 6 + 17 + 13) / 6 = 65 / 6 = 10.84 unit
- Average waiting time = (4 + 12 + 2 + 5 + 11 + 10) / 6 = 44 / 6 = 7.33 unit

- **The benefits of round robin CPU Scheduling:**
- Every process receives an equal amount of CPU time, therefore round robin appears to be equitable.
- To the end of the ready queue is added the newly formed process.

4. Shortest Remaining Time

- **Shortest remaining time (SRT) is the pre-emptive version of the SJN algorithm.**
- **The processor is allocated to the job closest to completion, but it can be pre-empted by a newer ready job with shorter time to completion.**
- **Impossible to implement in interactive systems where required CPU time is not known.**
- **It is often used in batch environments where short jobs need to give preference.**

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time	Response Time
1	0	8	20	20	12	0
2	1	4	10	9	5	1
3	2	2	4	2	0	2
4	3	1	5	2	1	4
5	4	3	13	9	6	10
6	5	2	7	2	0	5



- **Advantages:**
- SRTF algorithm makes the processing of the jobs faster than SJN algorithm, given it's overhead charges are not counted.
- Allows for easier management of library updates or replacements without recompiling the program.
- Enables efficient memory usage, as libraries can be shared among multiple instances of the program.
- Provides better portability, as the program can be executed on different systems with compatible libraries available at runtime.

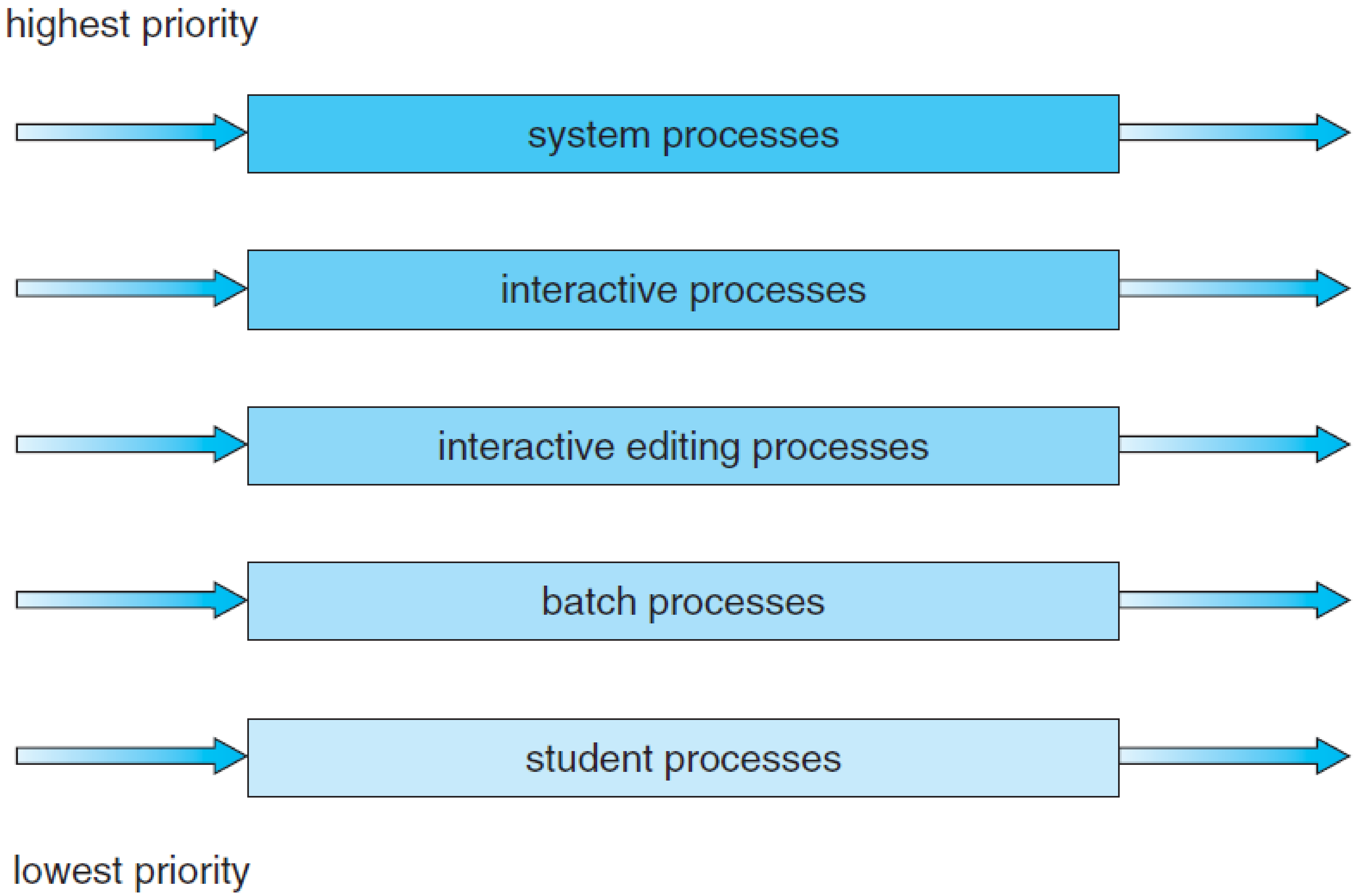
- **Disadvantages:**
- The context switch is done a lot more times in SRTF than in SJN, and consumes CPU’s valuable time for processing. This adds up to it’s processing time and diminishes it’s advantage of fast processing.
- Slightly slower program startup due to the additional linking process.
- Requires proper handling of library dependencies to ensure correct execution.
- Debugging can be slightly more complex, as libraries are separate entities loaded at runtime.

Multiple-Level Queues Scheduling

- Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.
- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.

- **Priorities are assigned to each queue.**
- **For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue.**
- **The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.**

- **Let's look at an example of a multilevel queue scheduling algorithm with**
- **five queues, listed below in order of priority:**
- **1. System processes**
- **2. Interactive processes**
- **3. Interactive editing processes**
- **4. Batch processes**
- **5. Student**


highest priority
system processes
interactive processes
interactive editing processes
batch processes
student processes
lowest priority

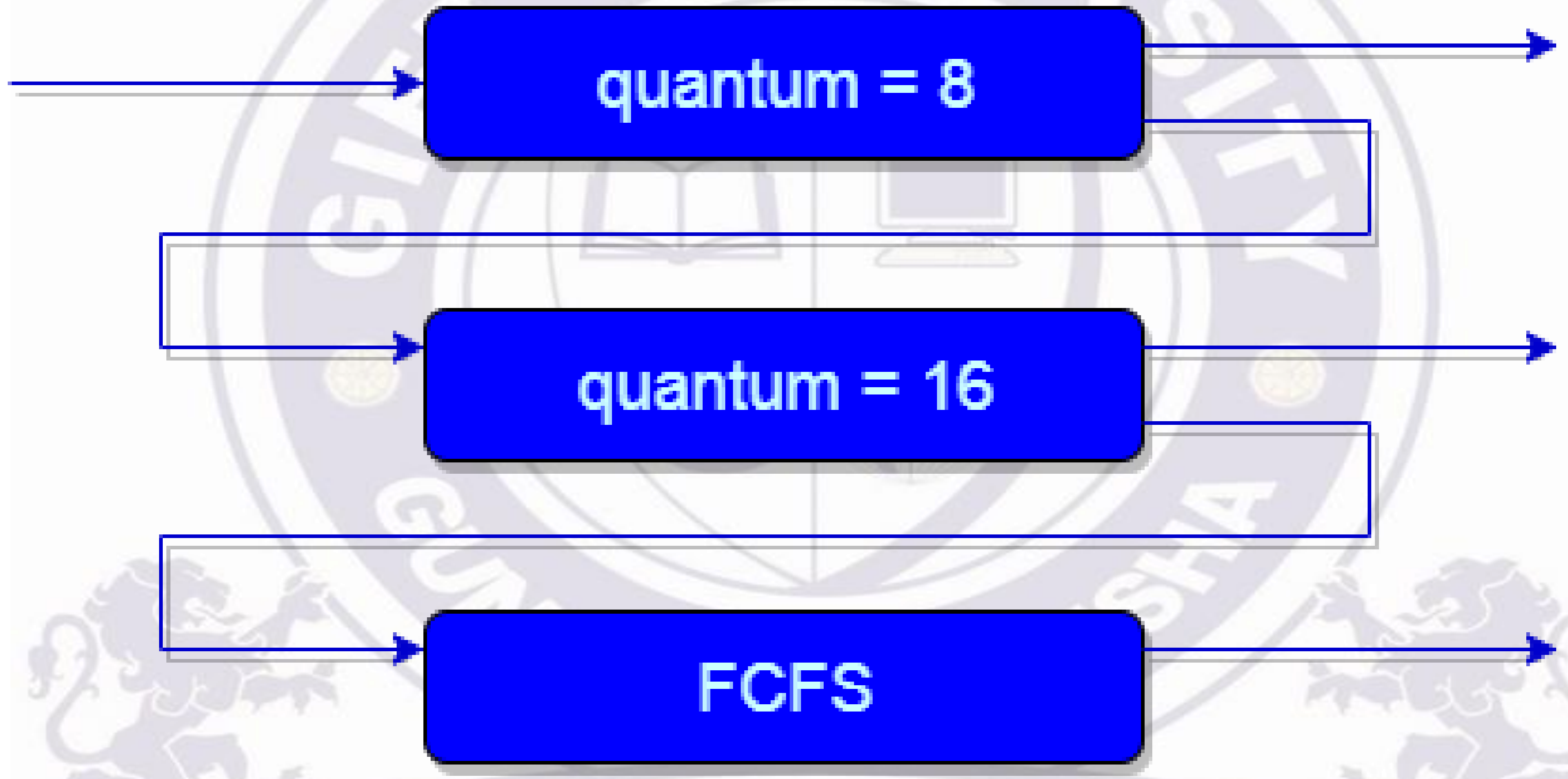
Multilevel Feedback Queue Scheduling

- **Multilevel feedback queue scheduling, however, allows a process to move between queues.**
- **The idea is to separate processes with different CPU-burst characteristics.**
- **If a process uses too much CPU time, it will be moved to a lower-priority queue.**
- **Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue.**
- **This form of aging prevents starvation.**

- **In general, a multilevel feedback queue scheduler is defined by the following parameters:**
- **The number of queues.**
- **The scheduling algorithm for each queue.**

- **The method used to determine when to upgrade a process to a higher-priority queue.**
- **The method used to determine when to demote a process to a lower-priority queue.**
- **The method used to determine which queue a process will enter when that process needs service.**

- **The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm.**
- **It can be configured to match a specific system under design.**
- **Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler.**
- **Although a multilevel feedback queue is the most general scheme, it is also the most complex.**


quantum = 8
quantum = 16
FCFS

Advantages of MFQS

- The advantages of MFQS are as follows:
- MFQS is a flexible scheduling algorithm.
- It allows the different processes to move between queues.
- Prevents CPU starvation.
- A technique called Aging helps promote a lower priority process to the next higher priority queue after a certain interval of time.

Disadvantages of MFQS

- **The disadvantages of MFQS are as follows:**
- **It is the most complex scheduling algorithm.**
- **It needs other means to be able to select the best scheduler.**
- **This process may have CPU overheads.**