# Operating system

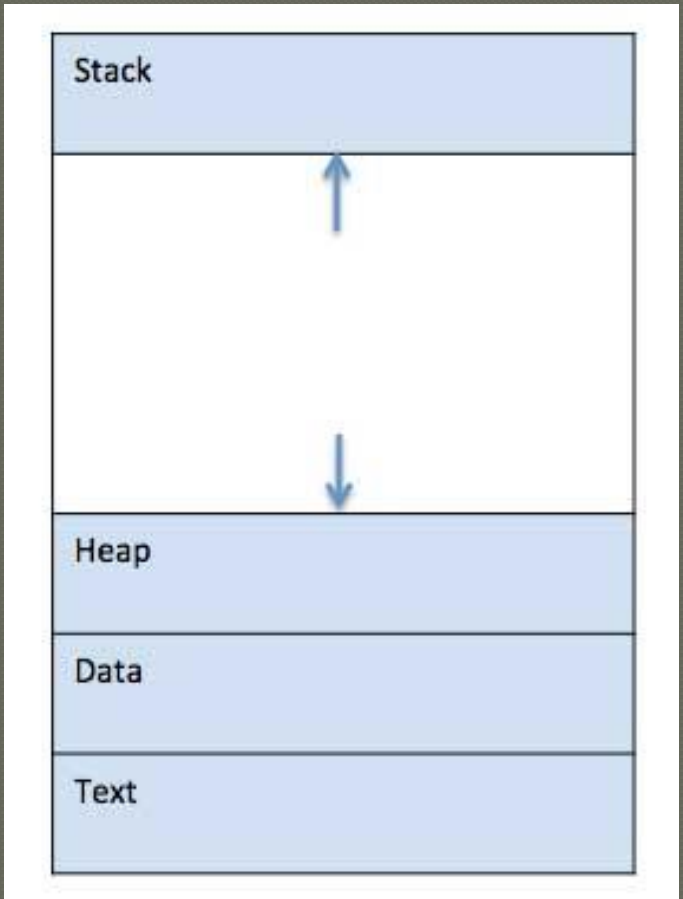
Unit-2

Process Management

# Process concept

- A process is basically a program in execution. The execution of a process must progress in a sequential fashion.
- A process is defined as an entity which represents the basic unit of work to be implemented in the system.
- To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

- When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory ─



1. **Stack:** The process Stack contains the temporary data such as method/function parameters, return address and local variables.

2. **Heap:** This is dynamically allocated memory to a process during its run time.

3. **Text:** This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.

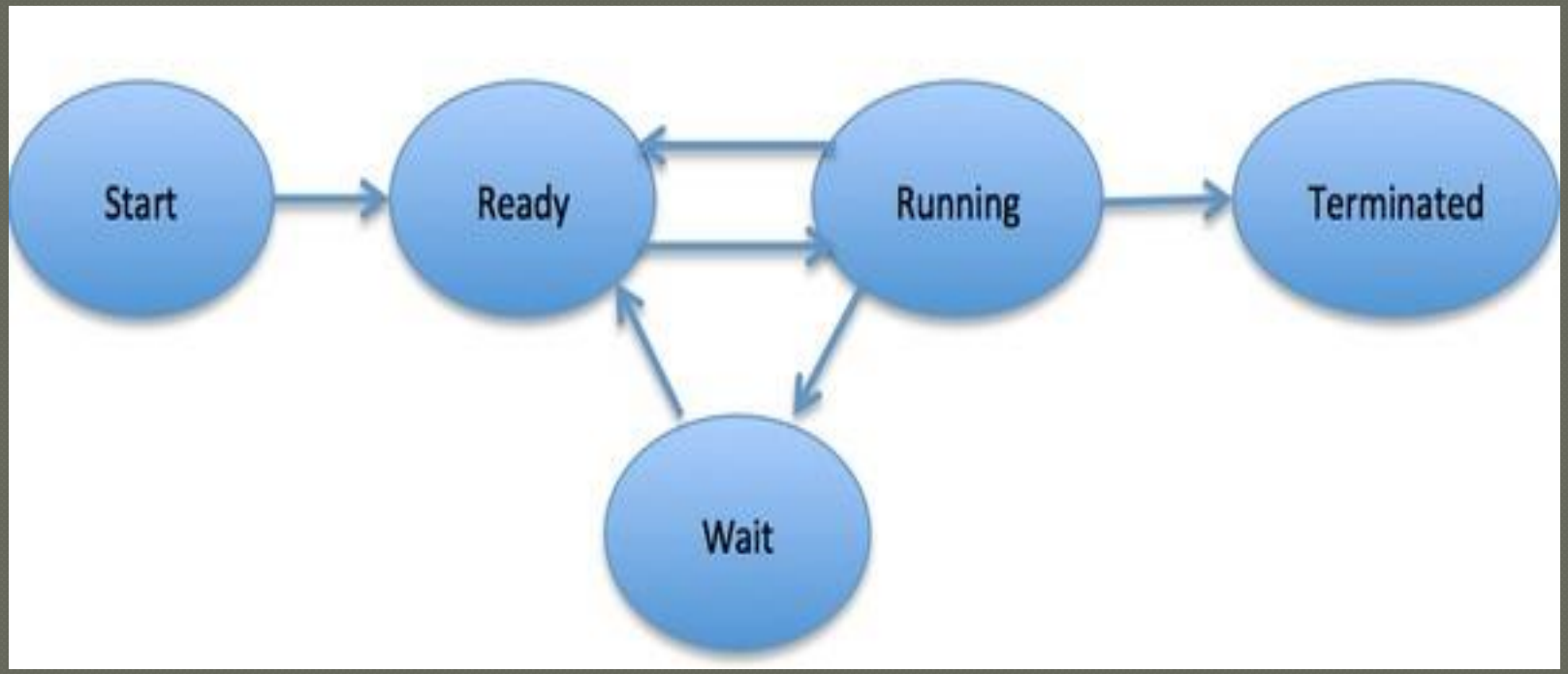
4. **Data:** This section contains the global and static variables.

# Process State

- When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

- In general, a process can have one of the following five states at a time.

# State Transition Diagram



1. **Start:** This is the initial state when a process is first started/created.

---

2. **Ready:** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.

3. **Running:** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

4. **Waiting:** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.

5. **Terminated or Exit:** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

# Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process.

- **Process State:** The current state of the process i.e., whether it is ready, running, waiting, or whatever.
- **Process privileges:** This is required to allow/disallow access to system resources.
- **Process ID:** Unique identification for each of the process in the operating system.

- **Pointer:** A pointer to parent process.

- **Program Counter:** Program Counter is a pointer to the address of the next instruction to be executed for this process.

- **CPU registers:** Various CPU registers where process need to be stored for execution for running state.

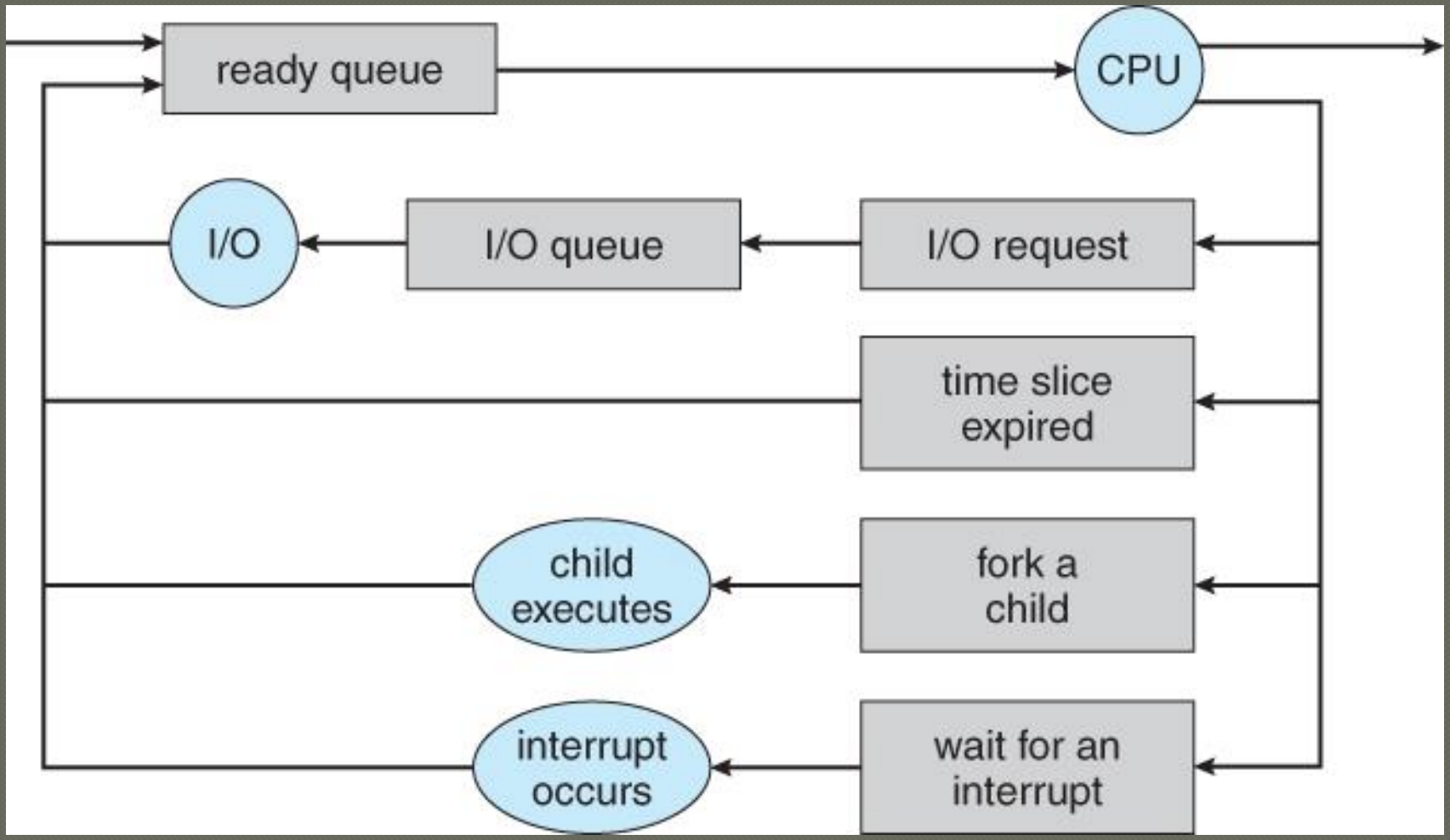
- **CPU Scheduling Information:** Process priority and other scheduling information which is required to schedule the process.

- **Memory management information:** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
- **Accounting information:** This includes the amount of CPU used for process execution, time limits, execution ID etc.
- **IO status information:** This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB –


The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

# Queuing Diagram



# Types of schedulers

- Process Scheduling handles the selection of a process for the processor on the basis of a scheduling algorithm and also the removal of a process from the processor. It is an important part of multiprogramming operating system.

- There are many scheduling queues that are used in process scheduling. When the processes enter the system, they are put into the job queue. The processes that are ready to execute in the main memory are kept in the ready queue. The processes that are waiting for the I/O device are kept in the I/O device queue.

The different schedulers that are used for process scheduling are −

1. **Long Term Scheduler:** The job scheduler or long-term scheduler selects processes from the storage pool in the secondary memory and loads them into the ready queue in the main memory for execution.

   The long-term scheduler controls the degree of multiprogramming. It must select a careful mixture of I/O bound and CPU bound processes to yield optimum system throughput. If it selects too many CPU bound processes then the I/O devices are idle and if it selects too many I/O bound processes then the processor has nothing to do.

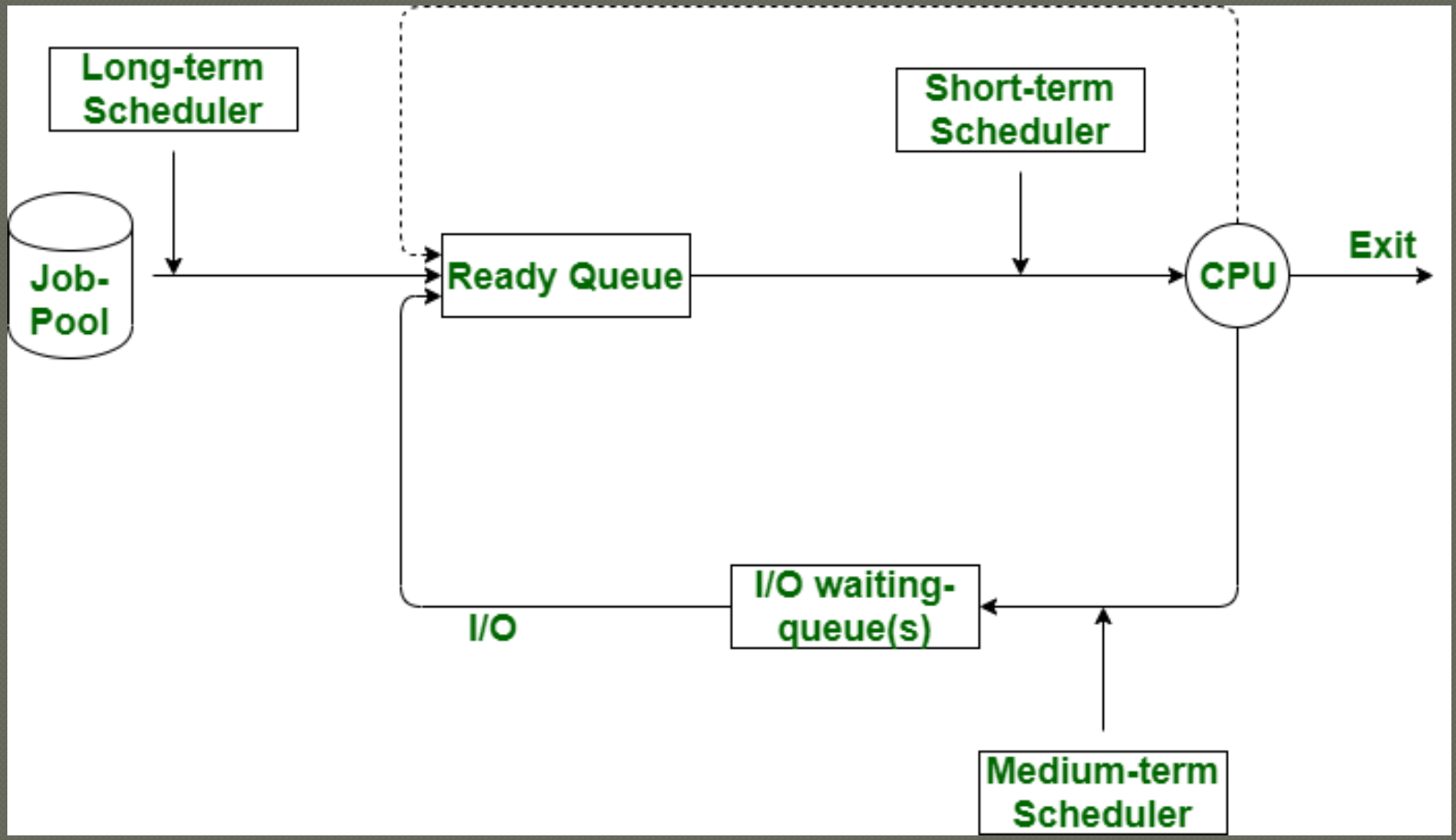
   The job of the long-term scheduler is very important and directly affects the system for a long time.

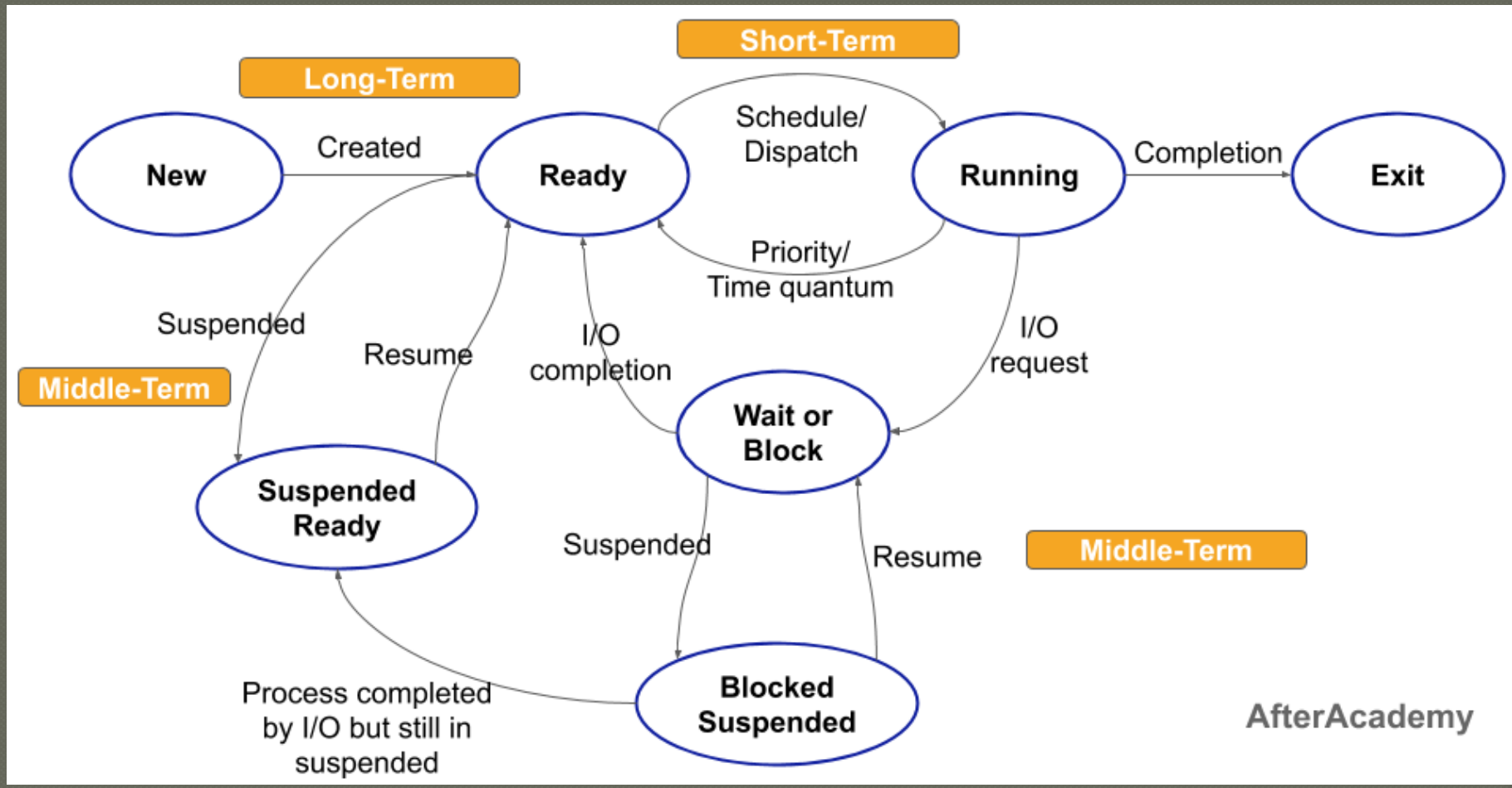
2. Short Term Scheduler: The short-term scheduler selects one of the processes from the ready queue and schedules them for execution .A scheduling algorithm is used to decide which process will be scheduled for execution next.

   The short-term scheduler executes much more frequently than the long-term scheduler as a process may execute only for a few milliseconds.

   The choices of the short term scheduler are very important. If it selects a process with a long burst time, then all the processes after that will have to wait for a long time in the ready queue. This is known as starvation and it may happen if a wrong decision is made by the short-term scheduler.

3.Medium Term Scheduler : The medium-term scheduler swaps out a process from main memory. It can again swap in the process later from the point it stopped executing. This can also be called as suspending and resuming the process. This is helpful in reducing the degree of multiprogramming. Swapping is also useful to improve the mix of I/O bound and CPU bound processes in the memory.


Long-term Scheduler
Short-term Scheduler
Job-Pool
Ready Queue
CPU
Exit
I/O
I/O waiting-queue(s)
Medium-term Scheduler


Short-Term
Long-Term
Schedule/
Dispatch
New
Created
Ready
Running
Completion
Exit
Priority/
Time quantum
Suspended
Resume
I/O
completion
I/O
request
Middle-Term
Wait or
Block
Suspended
Ready
Suspended
Resume
Middle-Term
Process completed
by I/O but still in
suspended
Blocked
Suspended
AfterAcademy

# Context Switching

- Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier.

# Context Switching Steps

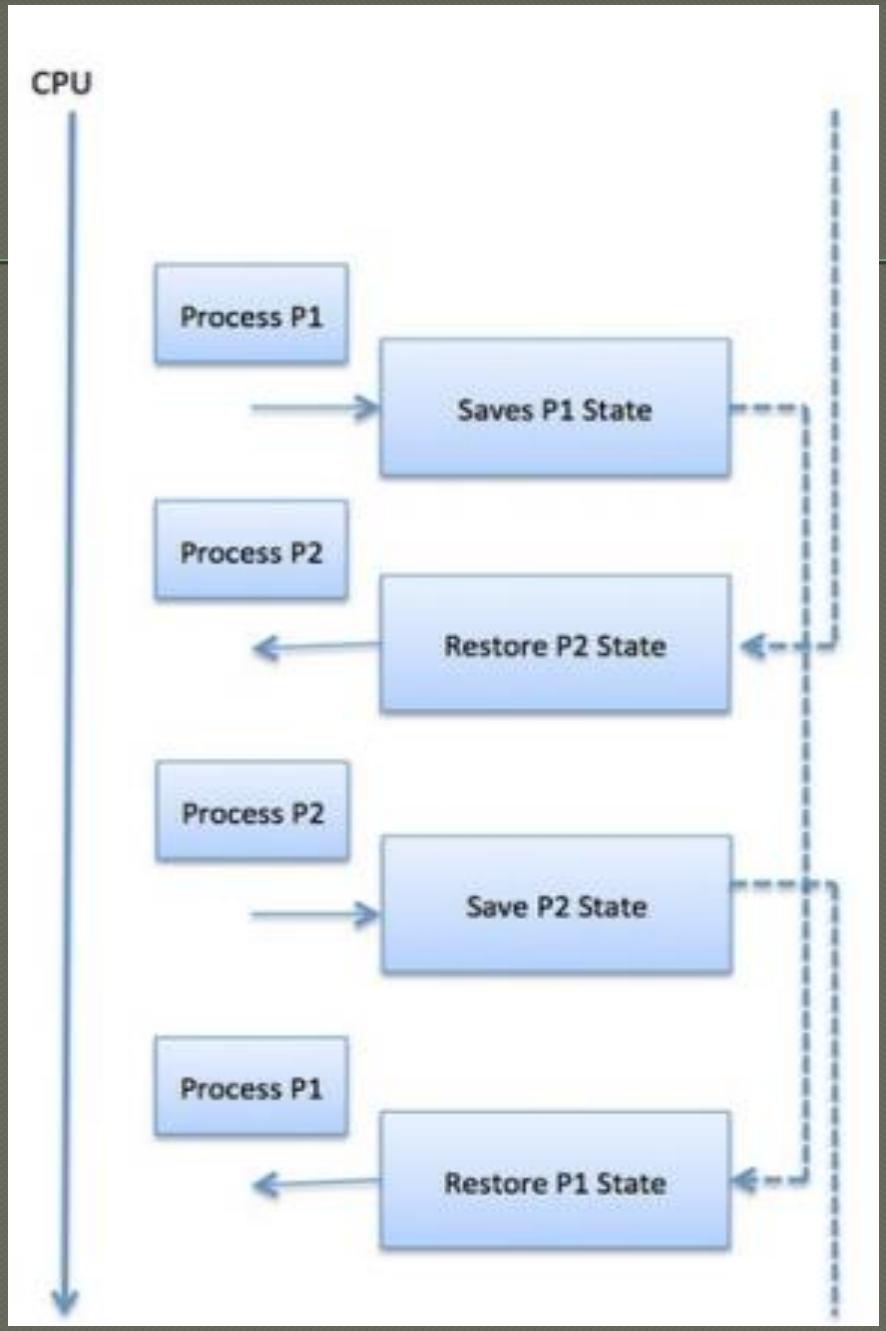
- Save the context of the process that is currently running on the CPU. Update the process control block and other important fields.
- Move the process control block of the above process into the relevant queue such as the ready queue, I/O queue etc.
- Select a new process for execution.
- Update the process control block of the selected process. This includes updating the process state to running.
- Update the memory management data structures as required.
- Restore the context of the process that was previously running when it is loaded again on the processor. This is done by loading the previous values of the process control block and registers.

There are three major triggers for context switching. These are given as follows −

- **Multitasking:** In a multitasking environment, a process is switched out of the CPU so another process can be run. The state of the old process is saved and the state of the new process is loaded. On a pre-emptive system, processes may be switched out by the scheduler.

- **Interrupt Handling:** The hardware switches a part of the context when an interrupt occurs. This happens automatically. Only some of the context is changed to minimize the time required to handle the interrupt.

- **User and Kernel Mode Switching:** A context switch may take place when a transition between the user mode and kernel mode is required in the operating system.

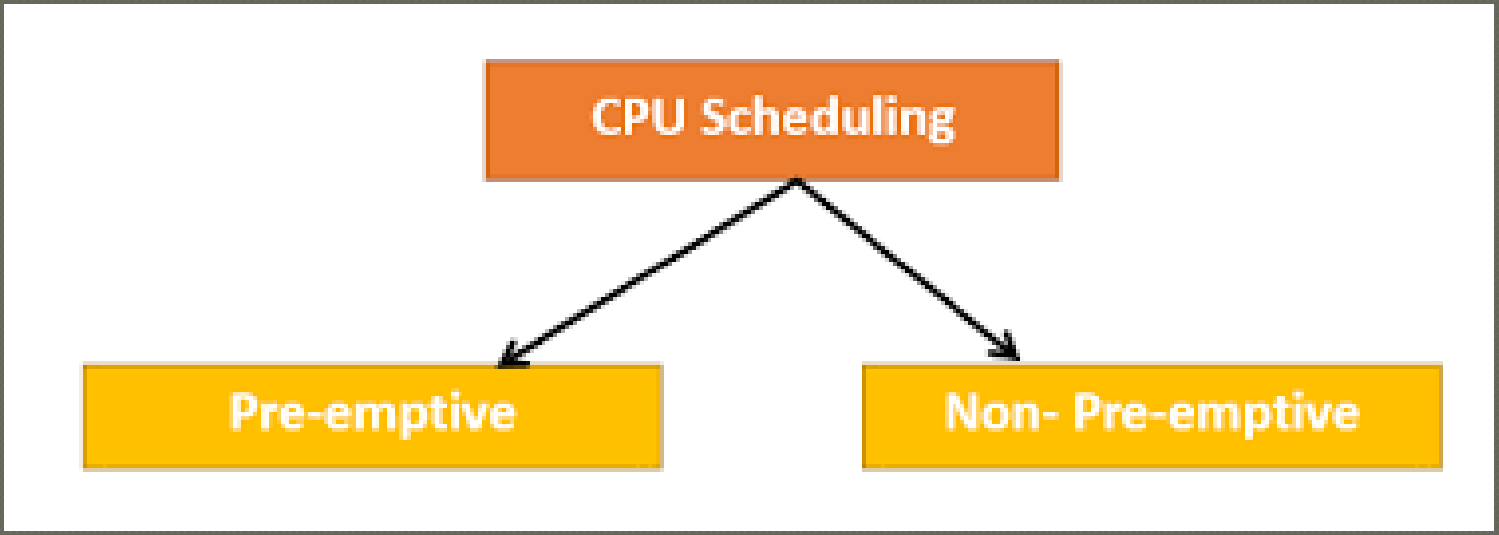

CPU
Process P1
Saves P1 State
Process P2
Restore P2 State
Process P2
Save P2 State
Process P1
Restore P1 State

# Dispatcher

- 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 takes that process to the desired state/queue. 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

# CPU Scheduling Algorithms

**CPU Scheduling** is a process of determining which process will own **CPU** for execution while another process is on hold. The main task of **CPU scheduling** is to make sure that whenever the **CPU** remains idle, the OS at least select one of the processes available in the ready queue for execution.



# Pre-emptive cpu scheduling

- **Preemptive Scheduling** is a CPU scheduling technique that works by dividing time slots of CPU to a given process. The time slot given might be able to complete the whole process or might not be able to it. When the burst time of the process is greater than CPU cycle, it is placed back into the ready queue and will execute in the next chance. This scheduling is used when the process switch to ready state.
- Algorithms that are backed by preemptive Scheduling are **round-robin (RR), priority, SRTF (shortest remaining time first).**

# Non Pre-emptive cpu scheduling

- **Non-preemptive Scheduling** is a CPU scheduling technique the process takes the resource (CPU time) and holds it till the process gets terminated or is pushed to the waiting state. No process is interrupted until it is completed, and after that processor switches to another process.
- Algorithms that are based on non-preemptive Scheduling are FCFS, non-preemptive priority, and shortest Job first.

# The criteria for judging the performance of scheduling algorithms

1. Processor utilization: processor utilization is the average fraction of time during which the processor is busy. Being busy refers to the processor not being idle and includes both time spent executing user programs and executing the operating system . scheduling algorithms that utilizes CPU very much is considered better. CPU can be utilized from 0-100%.

2. Throughput:- Throughput refers to the amount of work completed in a unit of time . One way to express throughput is by means of number of user jobs executed in a unit of time.

3.Turnaround time:-turn around time is defined as time that elapses from the moment a program or a job is submitted until it is completed by a system. It is the time spent in a system , and it may be expressed as a sum of the job service time(execution time) and waiting time.

4. Waiting time:-time that a process spends waiting for resource allocation due to contentions with others in multiprogramming system. It is the sum of the periods spent waiting in the ready queue. waiting time may be expressed as turnaround time less the actual execution time.

waiting time= turnaround time-execution time(burst time)

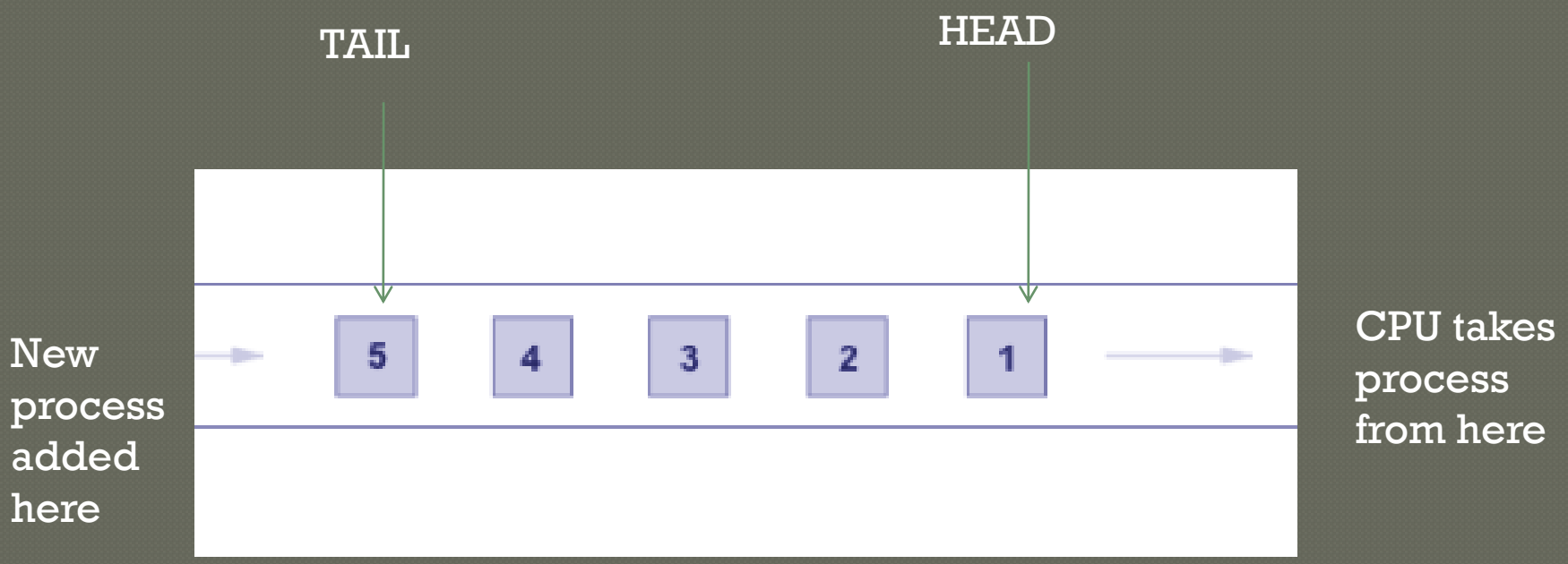
5.Response time:-A process can produce some output fairly early and can continue computing new results while results are being output to the user . Thus another measure is the time from the submission of a request until the first response is produced. This measure is called response time ,the amount of time it takes to start responding.

It is desirable to maximize CPU Utilization ,throughput and minimize turnaround time , waiting time , response time.

# FIRST COME FIRST SERVE(FCFS) SCHEDULING ALGORITHM

- It is non-preemptive type of scheduling algorithm.
- It is the simplest CPU scheduling algorithm.
- Job which comes first in ready queue is executed first by the CPU.
- In this policy ready queue is considered as FIFO queue , ie , the job comes in ready queue is added at the tail of the queue and the CPU takes the process from the head of the queue for execution.

# Ready queue in FCFS



# Advantages

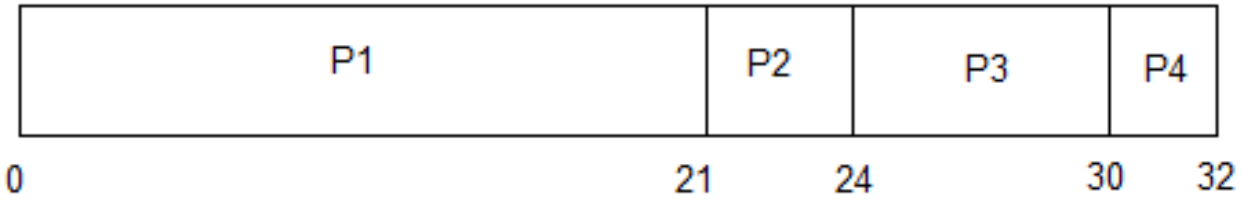
- It is simple scheduling algorithm.

## Disadvantages

- The average waiting time is often long.
- Not suitable for time sharing system.
- It may cause of **convoy effect**, where all other processes wait for a long process to get off the CPU.

# Example of FCFS

| PROCESS | BURST TIME |
|---|---|
| P1 | 21 |
| P2 | 3 |
| P3 | 6 |
| P4 | 2 |




This is the GANTT chart for the above processes

- Let all the process arrive at the time 0.

- Turn around time for process P1= 21-0
- Turn around time for process P2= 24-0
- Turn around time for process P3= 30-0
- Turn around time for process P4= 32-0

- average turn around time=(21+24+30+32)/4=26.75 ms

- Waiting time for process P1=0
- Waiting time for process P2=21
- Waiting time for process P3=24
- Waiting time for process P4=30

- Average waiting time=(0+21+24+30)/4=18.75 ms

# Shortest Job First(SJF) Algorithm

- This is very good CPU Scheduling algorithm.
- The idea is that the job that has the smallest next cpu burst is allocated to the CPU for execution first.
- Where more than one jobs have same length of next CPU burst the FCFS scheduling is used.
- This scheduling should be known as shortest next CPU burst rather than shortest job first.

# Advantage

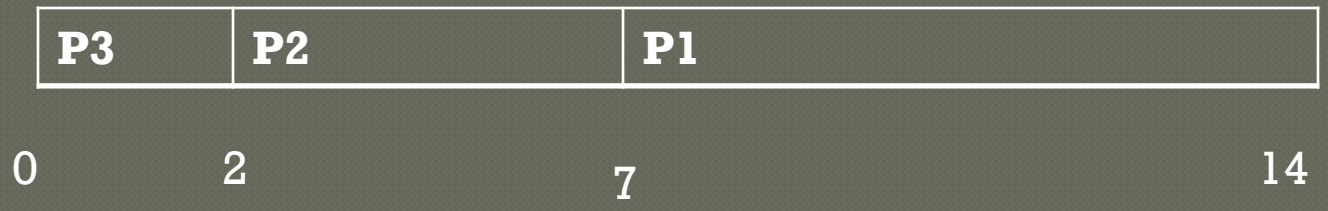
- It is optimal cpu scheduling algorithm in connection with waiting time.

## Disadvantage

It can not be implemented for the short term scheduler since there is no way to find the length of the next CPU burst.

# Example

| Process | CPU Burst Time |
|---|---|
| P1 | 7 |
| P2 | 5 |
| P3 | 2 |



- Turn around time for process P1=14-0
- Turn around time for process P2=7-0
- Turn around time for process P3=2-0

- Average turn around time =(14+7+2)/3=7.6 ms

- Waiting time for process P1=14-7=7
- Waiting time for process P2=7-5=2
- Waiting time for process P3=2-2=0

- Average waiting time=(7+2+0)/3=3 ms

# Non Preemptive Priority Algorithm

- In the Non Preemptive Priority scheduling, The Processes are scheduled according to the priority number assigned to them. Once the process gets scheduled, it will run till the completion. Generally, the lower the priority number, the higher is the priority of the process. Priority can be decided based on memory requirements, time requirements or any other resource requirement.
- If processes have equal priority then FCFS is used.

# Advantage

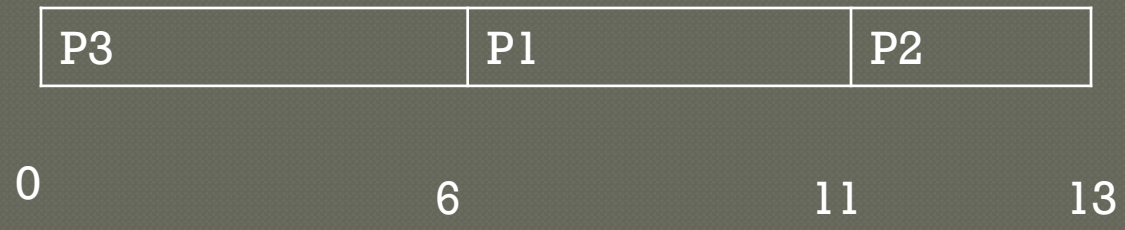
- There may be a number of criteria for the priority scheduling . It is very useful scheduling.

## Disadvantage

- Starvation is the major problem with priority scheduling ie some low priority processes are waiting indefinitely for the cpu.

# Example

| Process | Arrival time | Burst time | Priority |
|---|---|---|---|
| P1 | 0 | 5 | 2 |
| P2 | 0 | 2 | 3 |
| P3 | 0 | 6 | 1 |



- Turn Around time for P1=11-0
- Turn Around time for P2=13-0
- Turn Around time for P3=6-0

- Average Turn Around Time=(11+13+6)/3
  ie 10 ms

- Waiting time for P1= 11-5 => 6
- Waiting time for P2= 13-2 => 11
- Waiting time for P3= 6-6 => 0

- Average waiting time= (6+11+0)/3=>5.6 ms

# Round Robin (RR) Algorithm

- The idea of Round Robin scheduling is just like FCFS but it is preemptive.
- Each process is provided a fix time to execute, it is called a **time quantum or time slice.**
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Here ready queue is treated like a circular queue.
- New process is added to the tail of the queue .The process is picked for the execution from the head of the queue.
- The Timer is set to interrupt after one time slice.

- There may be cases as follows:

1. In first case, process CPU burst time is less than one time slice, then it will be completely executed and will release CPU and next process is executed.
2. In second case, an interrupt occurs at one time slice. A context switch will be made and process will be put at the tail of ready queue . The next process is executed.

- Performance of RR is totally depends on size of time slice.
- If time slice is too large then RR is just like FCFS.
- If time slice is too small then RR is just like process sharing, ie process will take much time to wait.

# Example

TQ=3MS

| Process | Arrival time | Burst Time |
|---|---|---|
| P1 | 0 | 5 |
| P2 | 1 | 6 |
| P3 | 2 | 3 |
| P4 | 3 | 1 |
| P5 | 4 | 5 |
| P6 | 6 | 4 |

READY QUEUE ~~P2~~,~~P3~~,P4, P1,P5,P6,P2,P5,P6


0 3 6 9 10 12 15 18 21 23 24

- Turn Around time for P1=12-0 ie 12
- Turn Around time for P2=21-1 ie 20
- Turn Around time for P3=9-2 ie 7
- Turn Around time for P4=10-3 ie 7
- Turn Around time for P5=23-4 ie 19
- Turn Around time for P6=24-6 ie 18
- Average turn around time= (12+20+7+7+19+18)/6=>13.8 ms

- Waiting time for P1=12-5 => 7
- Waiting time for P2= 20-6 => 14
- Waiting time for P3= 7-3 =>4
- Waiting time for P4= 7-1 => 6
- Waiting time for P5= 19-5 => 14
- Waiting time for P6= 18-4 => 14

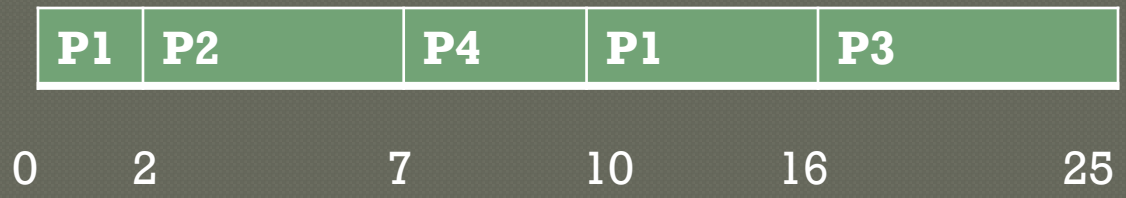
Average waiting time=
(7+14+4+6+14+14)/6=>9.8 ms

# Shortest Remaining Time First(SRTF)

- This Algorithm is the preemptive version of SJF scheduling. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.
- Once all the processes are available in the ready queue, No preemption will be done and the algorithm will work as SJF scheduling. The context of the process is saved in the Process Control Block when the process is removed from the execution and the next process is scheduled. This PCB is accessed on the next execution of this process.

# Example

| Process | Arrival time | Burst time |
|---|---|---|
| P1 | 0 | 8 |
| P2 | 2 | 5 |
| P3 | 3 | 9 |
| P4 | 4 | 3 |



- Turn Around time for P1=16-0 ie 16
- Turn Around time for P2=7-2 ie 5
- Turn Around time for P3=25-3 ie 22
- Turn Around time for P4=10-4 ie 6

- Average Turn Around time= (16+5+22+6)/4=>12.25 ms

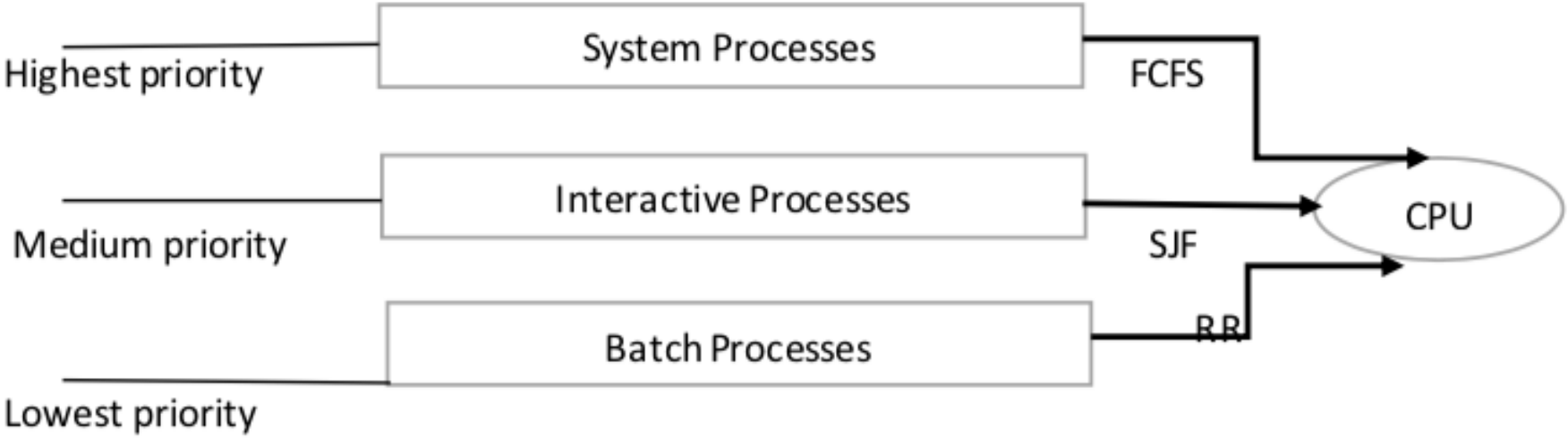
- Waiting time for P1=16-8 => 8
- Waiting time for P2= 5-5=> 0
- Waiting time for P3= 22-9 =>13
- Waiting time for P4= 6-3 => 3

Average waiting time= (8+0+13+3)/4=>6 ms

# Multi-Level Queue Scheduling

- Multi-level queue 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.


Highest priority
System Processes
FCFS
Medium priority
Interactive Processes
SJF
CPU
Batch Processes
R R
Lowest priority

# Advantages

- More efficient scheduling algorithm since each process is handled differently according to its property.

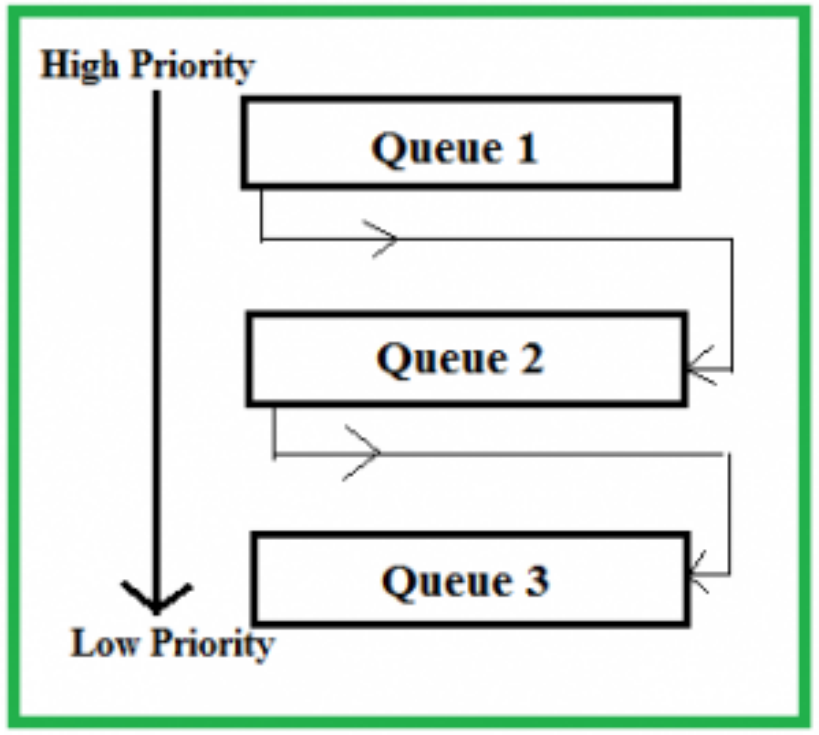
## Disadvantages

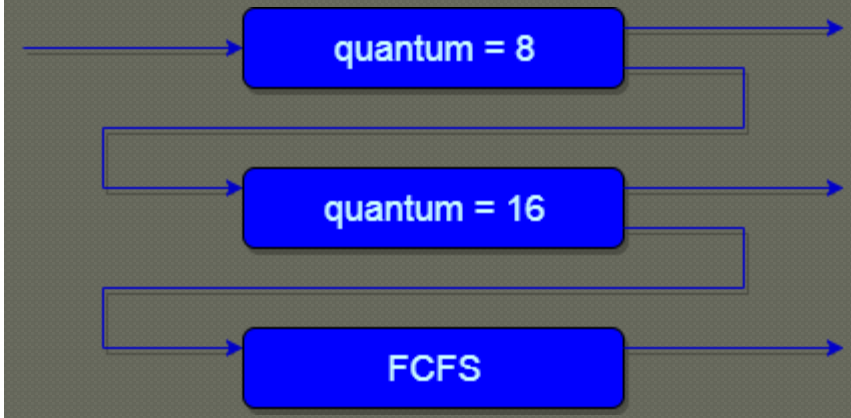
- The process can not move between queues that provide inflexibility.
- If a process that waits too long time in a lower priority queue may cause starvation.

# Multi-Level Feedback Queue Scheduling

- One of the major disadvantage of multilevel queue scheduling is indefinite postponement of lower priority processes. To eliminate this problem the multilevel feedback queue allows to move processes between queues.
- Here idea is that processes are prioritized according to the CPU time, so that I/O bound processes found higher priority than CPU bound processes. So I/O bound processes are put in upper queues and CPU bound processes are put in lower queues.

- Processes that wait to much time in lower queue may be moved to higher priority queue . This is called aging.
- Aging is solution of starvation problem


High Priority
Queue 1
Queue 2
Queue 3
Low Priority


quantum = 8
quantum = 16
FCFS

| Process | Burst time | Arrival time |
|---|---|---|
| P1 | 0 | 0 |
| P2 | ~~29 19~~ 9 0 | 0 |
| P3 | 0 | 0 |
| P4 | 0 | 0 |
| P5 | ~~12 2 0~~ | 0 |

TQ=10

READY QUEUE

~~P1 P2 P3 P4 P5 P2 P5 P2~~

| P1 | P2 | P3 | P4 | P5 | P2 | P5 | P2 |
|---|---|---|---|---|---|---|---|

0 10 20 23 30 40 50 52 61