Operating System

GATEWAY CLASSES

ONE SHOT

Unit-3

CPU Scheduling & Deadlock

CS / IT / CS Allied/ MCA

By Dr. Kapil Kumar Sir

Operating System
Unit-3
Syllabus

CPU Scheduling: Scheduling Concepts, Performance Criteria, Process States, Process Transition Diagram, Schedulers, Process Control Block (PCB), Process address space, Process identification information, Threads and their management, Scheduling Algorithms, Multiprocessor Scheduling.

Deadlock: System model, Deadlock characterization, Prevention, Avoidance and detection, Recovery from deadlock.

Process Concept

- Early computer system allowed only one program to be executed at a time. This program had complete control of the system and had access to all the system's resources.
- Now a days computer system allows multiple programs to be loaded into memory and executed concurrently. (multiprogramming, multitasking)
- This evolution required strong control and more categorization of the various programs; and these needs resulted in the notion of a process, which is a program in execution.
- A process is the unit of work in a modern time sharing system.
- A system consists of a collection of process, by switching the CPU between processes the operating system can make the computer more productive.

Difference Between Program and Process

S.No.	Program	Process
1.	Set of instructions designed to complete a specific task.	Process is an instance of an executing program.
2.	Program is a passive entity as it resides in the secondary memory.	Process is a active entity as it is created during execution and loaded into the main memory.
3.	Program exists at a single place and continues to exist until it is deleted.	Process exists for a limited span of time as it gets terminated after the completion of task.
4.	Program is a static entity.	Process is a dynamic entity.
5.	Program does not have any resource requirement, it only requires memory space for storing the instructions.	Process has a high resource requirement, it needs resources like CPU, memory address, I/O during its lifetime.

Imp Process States or Process Life Cycle

- As a process executes it changes state.
- Each process may be in one of the following primary states:

1 ✓ **New:** The process is being created. For example written a C program and stored in secondary memory i.e. program is stored in secondary memory in stable state.

LTS

2 ✓ **Ready:**

- The process is waiting to be assigned to a processor (CPU)
- Processes are in ready queue,
- Active state,
- A process is in primary memory (RAM), allocation of memory
- Now process is in a faster memory and ready for execution.
- There will be n number of processes in secondary memory. The long term scheduler will select some processes from secondary memory and will send to the primary memory in ready queue.
- Long term scheduler controls the degree of multiprogramming.

3 ✓ **Running:** A process will be dispatched or scheduled to processor (CPU) for its execution.

4 ✓ **Waiting:** The process in waiting for sample event to occur (such as an I/o completion or reception of a signal)

5 ✓ **Terminated:** The process has finished execution. Deallocation of resources.

✓ **Minimum number of states for a process : 4,** i.e. new , ready , running, terminated.

✓ Now suppose CPU is running a process but some higher priority process comes in ready queue then the running process will be sent to the ready queue and higher priority process will start its execution in CPU.

- This is concept of multitasking i.e. multiple processes are being executed at a time,
- Another reason of shifting a process from CPU to ready queue may be expiration of time quantum in round robin scheduling algorithm.
- This is done by short term scheduler (or CPU scheduler or dispatcher) that selects a process from the ready queue and allocate CPU to that process for its execution.

- ✓ If CPU is executing a process completely at once, this is non preemptive scheduling.
- ✓ If a process stopped in between due to certain reasons (higher priority process comes, time quantum expires), this is preemptive scheduling.
- ✓ Suppose CPU is executing a process,
- During execution suppose process ask for I/O request,
- Say reading a file which is available with secondary memory,
- In this situation, CPU can not help in reading and writing of data,
- Its primary responsibility is to execute the instruction,
- In this situation the process will go in waiting state or block state
- After I/O completion, that process will go to the ready state (ready queue).
- Suppose, this blocked process completed its I/O request but do not get chance for its execution then that process may go to the ready state.


Schedulers 3 – a – LTS
b – MTS
c – STS
(maintain Degree of Multiprogramming)
Job Queue
1
eg 1000 PROGRAMS
n no. of processes
LTS select some processes from n no. of processes for admitted multiprogramming
new
SM
Stable State
Process is being created
Prog 2 o/p Prog
Ready Queue
2
R/W
RAM
PM
ready
1 Process
STS
Schedular | dispatcher
CPU Schedular
running
3 CPU
RR
interrupt | multitasking
Time Quantum
exit
4
Terminated
deallocation of resources
RAM
PM
I/O or event wait
I/O or event Completion
5
waiting
OS block
RAM

4 Basic state (minimum)

1. New ⟶ Sm / stable
2. Ready → Pm
3. Running → CPU
   Execution
4. Terminated ⇒ Resource deallocation
   CPU + memory
   + I/O
5. waiting

6. suspended wait
7. suspended Ready

5 Basic states

1 New ✓ *
2 Ready ✓ *
3 waiting
4. Running ✓ *
5 Termination ✓ *

min = 4

7 states

NON-PREEMPTION

Time Quantum Expires | Higher Priority process

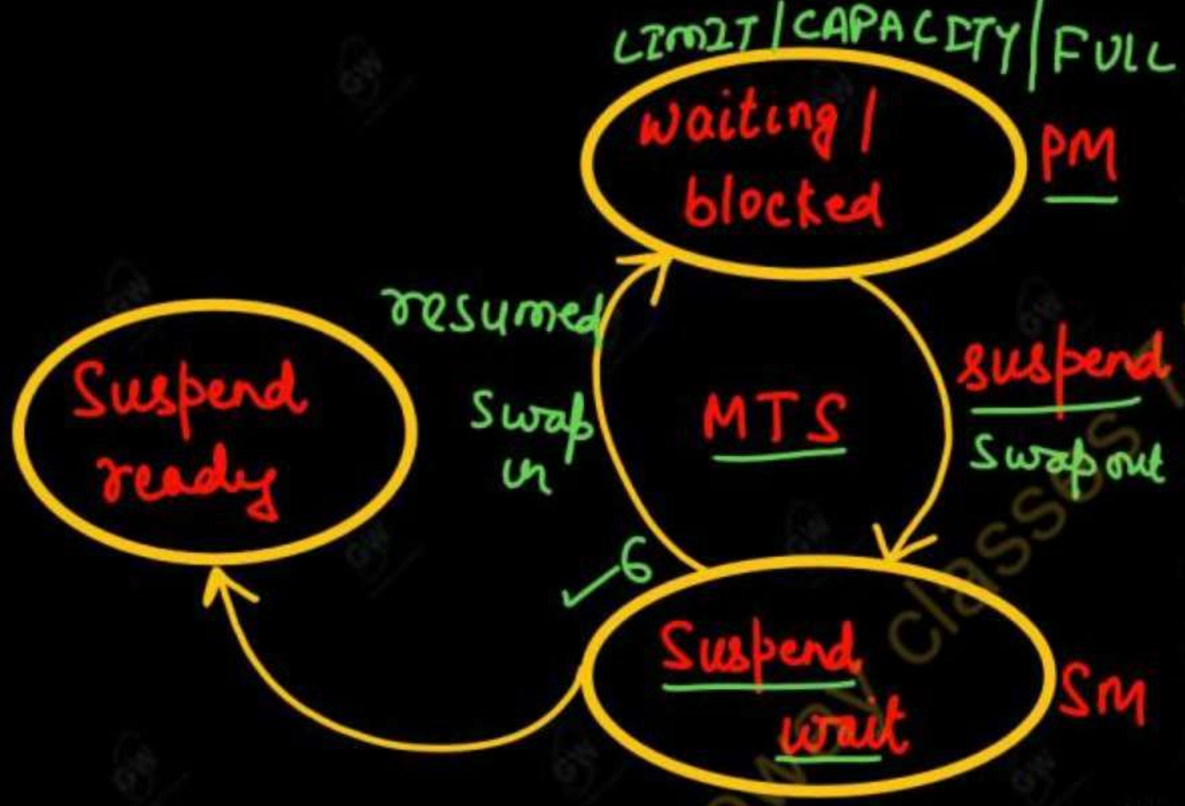
PREEMPTIVE

Two More States

1. Suspend Ready State

- While the ready queue can accommodate numerous processes, its capacity within main memory remains finite.
- In main memory, the ready queue's capacity is limited, preventing the accommodation of numerous or all processes. In instances where priority processes enter the queue but space is lacking, the scheduler must select a lower-priority process, suspend its execution, and subsequently move it out of main memory—a state termed "suspend ready."
- Once space becomes available in the ready queue, the process can be resumed or swapped back into memory.
- This pivotal task is managed by the Mid-term scheduler.

2. Suspend-wait State



- If waiting/blocked processes waiting for I/O request and if more and more processes will go for I/O request i.e. more processes are coming in waiting state then at one time memory will be full and then Operating System have to pick some processes and swapped out these processes in secondary memory, that state is known as "suspend wait" and then due to availability of space in "waiting/blocked" state in primary memory, the process will be resumed.
- In suspend –wait state the process is also busy with I/O operations. This task is done by the medium – term scheduler or mid-term scheduler.

CPU Bound and I/O Bound Processes

LTS (new ⇒ Ready)

- In operating systems, processes are often classified based on their behavior and resource usage.
- Two common classifications are CPU-bound and I/O-bound processes:
- **CPU-Bound Processes:**
  - A CPU-bound process primarily requires the CPU for execution.
  - These processes typically perform a lot of computations, such as mathematical calculations, sorting algorithms, or intensive data processing tasks.
  - CPU-bound processes spend most of their time executing instructions and utilize minimal I/O operations.
  - Examples include scientific simulations, rendering tasks in graphics applications, or cryptographic computations.

- **I/O-Bound Processes:**
  - An I/O-bound process relies heavily on input/output (I/O) operations for its execution.
  - These processes frequently interact with external devices or resources such as disk drives, network interfaces, or user input/output devices.
  - I/O-bound processes often spend a significant amount of time waiting for I/O operations to complete, rather than actively using the CPU.
  - Examples include file I/O operations, network communication, database queries, or user interactions in interactive applications.
- CPU-bound processes are characterized by high CPU utilization and minimal I/O activity, while I/O-bound processes heavily rely on I/O operations and may spend considerable time waiting for these operations to finish.
- Operating systems need to manage both types of processes efficiently to ensure optimal system performance and resource utilization.

✓ Process Control Block (PCB) | TASK CONTROL BLOCK

- ❑ Each process is represented in the operating system by PCB.
- ❑ It contains many pieces of information associated with a specific process such as:
- ✓ **Process State:-** The state may be new, ready, running, waiting, halted etc.

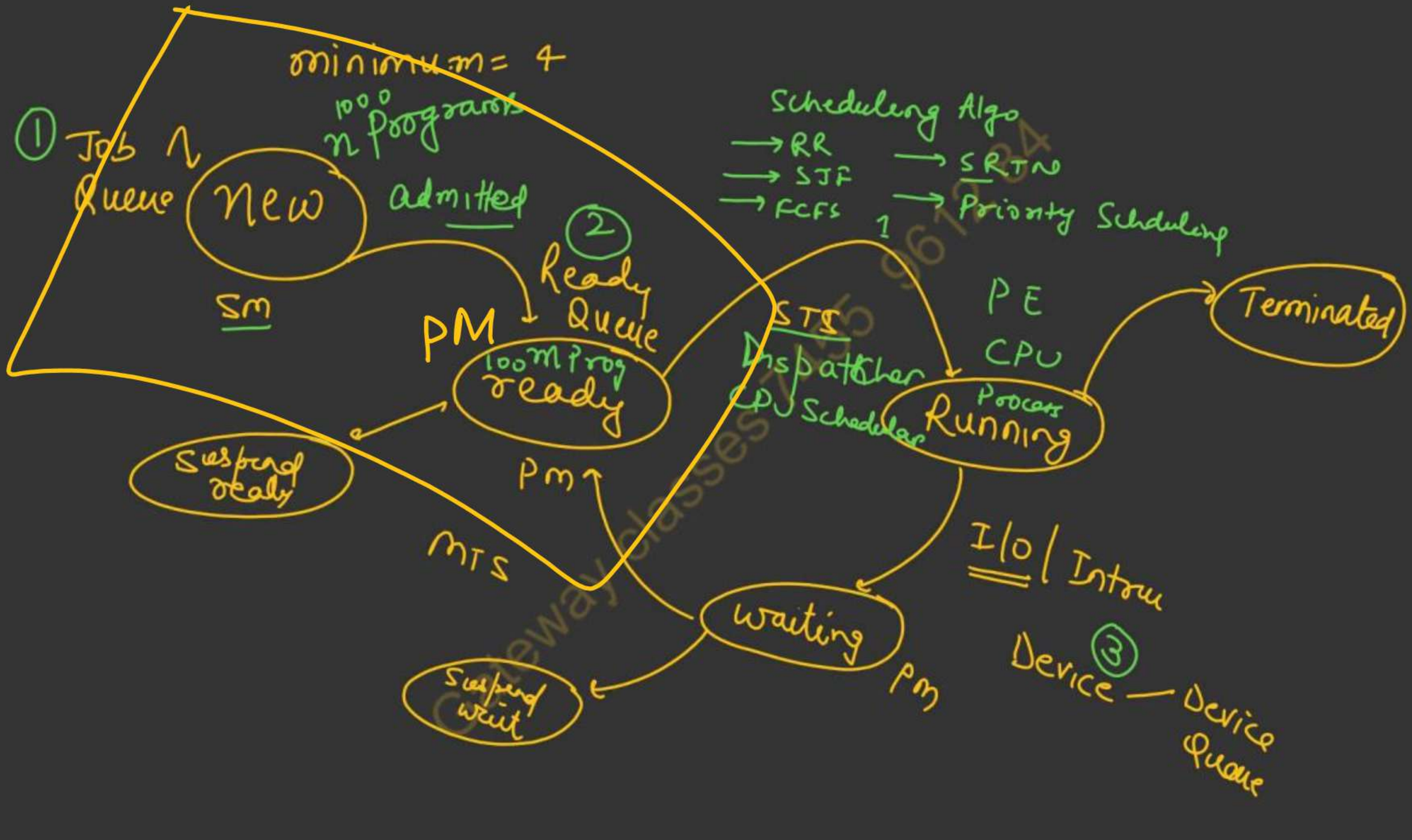
✓ PCB

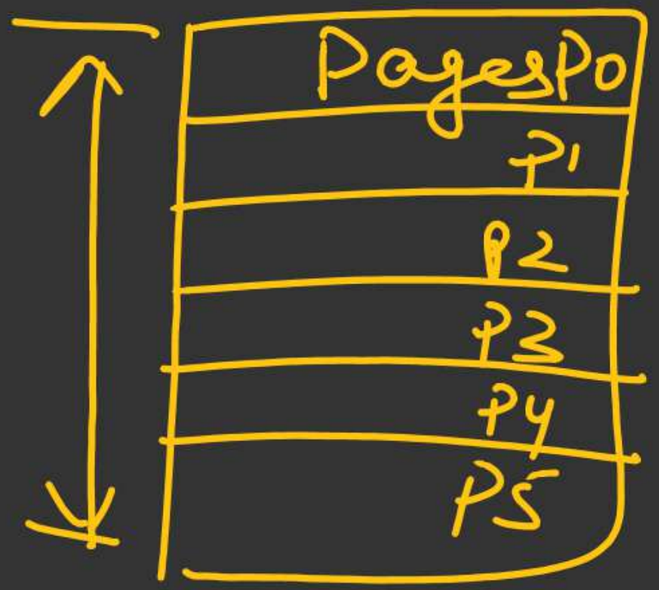
1	PROCESS ID / NUMBER
2	PROCESS STATE
3	PROGRAM COUNTER Pc
4	CPU REGISTERS
5	CPU SCHEDULING INFO
6	MEMORY MGMT INFO / MEM. LIMITS
7	ACCOUNTING INFO
8.	I/O STATUS INFO. (I/O)

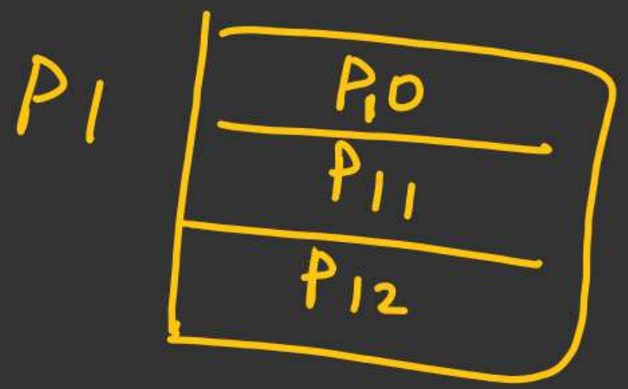
- ✓ **Program Counter:-** The program counter indicates the address of the instruction to be executed next.
- ✓ **CPU Registers:-** They include AC, index registers, stack pointers and general purpose registers. Along with the PC, this state information must be saved when an interrupt occurs, to allow 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 may include such information as the value of the base and limit registers, the page table or the segment table 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 etc.
- ✓ **I/O State Information:-** List of I/O devices allocated to the process, a list of open files etc.

Scheduling Queues

- As processes enter the system they are put into a "job queue" which consist of all processes in the system.
- The job queue contains all the processes residing on the disk waiting to be brought into memory for execution.
- Processes in the job queue are typically in the "new" state, waiting to be admitted into the system.
- When a process is admitted into the system, it is moved from the job queue to the ready queue.
- The processes that are residing in the main memory and are ready and waiting to executing are kept on a list called the "ready queue".
- Ready queue is a queue of processes that are ready to be executed by the CPU.
- Processes in the ready queue are waiting for their turn to be allocated CPU time for execution.


minimum = 4
1000 n Programs
① Job Queue
New
admitted
SM
② Ready Queue
PM
loomiprog ready
Suspend ready
PM
MTS
STS
Dispatcher
CPU Scheduler
Scheduling Algo
RR
SJF
FCFS
SRTN
Priority Scheduling
1
PE
CPU
Process Running
Terminated
I/O / Intrw
③ Device
Device Queue
waiting
PM
Suspend wait


Pages P0
P1
P2
P3
P4
P5


P1
P10
P11
P12

Ready Queue

- The operating system's scheduler selects processes from the ready queue based on scheduling algorithms like First Come First Serve (FCFS), Shortest Job First (SJN), Round Robin, etc. Prio. — .
- When a process is allocated the CPU it execute for a while and eventually quits, is interrupted or waits for the occurrence of a particular event such as the completion of an I/O request.

Device Queue

- **Device queues** are used to manage I/O devices such as disk drives, printers, and network interfaces.
- Each I/O device typically has its own queue containing requests from processes waiting for I/O operations to be completed.
- Device queues ensure that I/O operations are handled in an orderly manner and prevent conflicts or resource contention among processes.

- Suppose the process makes an I/O request to a shared device, such as a disk.
- Since there are many processes in the system the disk may be busy with the I/O request of some other process.
- The process therefore may have to wait for the disk.
- The list of processes waiting for a particular I/O device is called a "device queue".
- Each device has its own device queue.
- A common representation of process scheduling is a "Queuing Diagram".

Schedulers

- Schedulers in operating systems are software components responsible for managing the allocation of system resources to processes.
- They play a crucial role in determining which processes get to use the CPU, for how long, and in what order.
- There are several types of schedulers, each serving a specific purpose within the operating system:
- Long-Term Scheduler (or Job Scheduler)
  - The long-term scheduler selects processes from the job queue (or creates new processes) and loads them into memory for execution.
  - The long-term scheduler controls the "degree of multiprogramming" i.e. the number of processes in memory.
  - Long-term scheduling decisions are made infrequently, as they involve moving processes between secondary storage and main memory.

- 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.
- It is important that the long-term scheduler select a good process mix of I/O bound and CPU bound processes.
  - If all processes are I/O bound the ready queue will almost always be empty and the short term scheduler will have little to do.
  - It all processes are CPU bound the I/O waiting queue will almost always be empty, devices will go unused and again the system will be unbalanced.
  - The system with the best performance will thus have a combination of CPU bound and I/O bound processes.

❑ Short-Term Scheduler (or CPU Scheduler)

- The short-term scheduler selects which ready process in the ready queue will be executed next and allocates CPU time to it.
- It makes frequent decisions, often on the order of milliseconds or microseconds.
- Short-term scheduling is concerned with optimizing CPU utilization, throughput, response time, and fairness among processes.
- Common scheduling algorithms used by the short-term scheduler include First Come First Serve (FCFS), Round Robin (RR), Shortest Job Next (SJN), and Priority Scheduling etc.

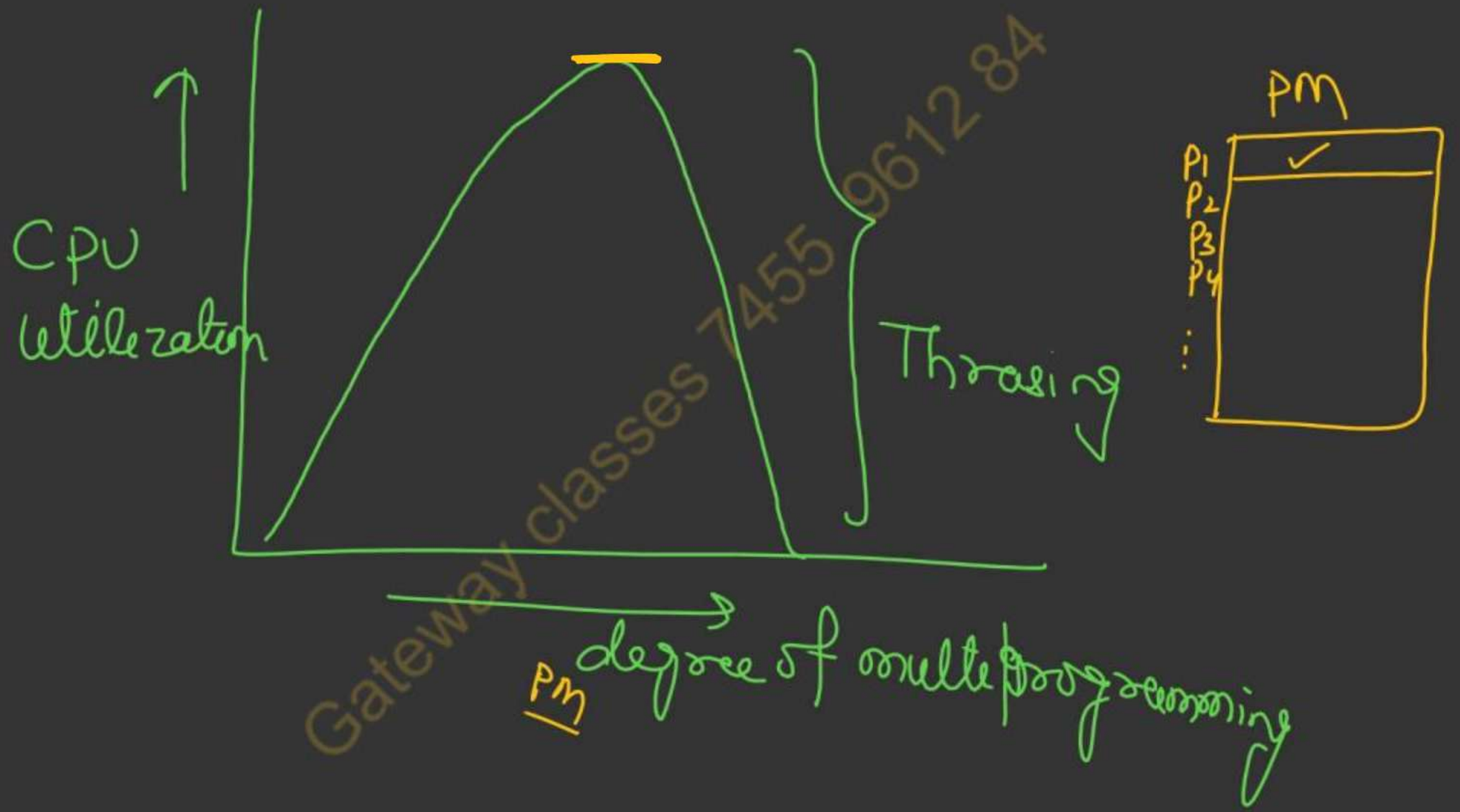
❑ Medium-Term Scheduler (or Mid Term Scheduler)

- The medium-term scheduler may be present in some operating systems, especially those with memory management techniques like swapping or paging.
- It performs process swapping, moving processes between main memory and secondary storage to manage memory usage efficiently.

- The medium-term scheduler helps prevent memory thrashing and ensures that processes have adequate memory resources for execution.
- Thrashing in operating systems occurs when the system spends a significant amount of time swapping data between main memory (RAM) and disk, rather than executing useful tasks.
- This happens when the system is overloaded with too many processes competing for limited resources, leading to excessive paging or swapping activity.
- Thrashing severely degrades system performance, causing a slowdown in overall operations.

❑ Schedulers work together to ensure efficient utilization of system resources, including CPU time, memory, and I/O devices, while meeting performance objectives such as throughput, response time, and fairness.

❑ Each scheduler operates at a different level of granularity and makes decisions based on specific criteria to achieve its objectives within the operating system.



- Some operating system such as time sharing systems may introduce an additional intermediate level of scheduling i.e. "medium-term scheduling".
- The key idea behind a medium-term scheduler is that sometimes it may be advantageous to remove processes from memory and thus reduce the degree of multiprogramming.
- Later, the process can be reintroduced into memory and its execution can be continued where it left off. This scheme is called swapping.
- The process is swapped out and is later swapped in by the medium term scheduler.

Context Switch

- Interrupts cause the Operating System to change a CPU from its current task and to run a kernel routine.
- When an interrupt occurs the system needs to save the current context of the process running on the CPU so that it can restore that context when its processing is done, essentially suspending the process and then resuming it.
- The context is represented in the Process Control Block (PCB) or Task Control Block of the process; it includes the value of the CPU registers, the process state and memory management information etc.
- Generally, we perform a state save of the current state of the CPU be it in kernel or user mode and then a state-restore to resume operations.
- Switching the CPU to another process requires performing a state-save of the current process and a state-restore of a different process. This task is known as context switch.

- When a context switch occurs the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
- Context-switch time is overhead, because the system does no useful work while switching.

Operations on Processes

- **Process Creation:-**

- ✓ A process may create several new processes via a create-process system call, during the course of execution.
- ✓ The creating process is called a parent process, and the new processes are called the children of that process.
- ✓ Each of these new processes may in turn create other processes, forming a tree of processes.
- ✓ Most operating system including the Unix and the Windows family of operating systems identify processes according to a new process identifier (pid), which is typically an integer number.

- When a process creates a new process, two possibilities exist in terms of execution:

1. **Forking:**

- Forking is a method where the parent process creates a copy of itself to form the child process.
- After forking, both the parent and child processes continue execution from the same point in the code, but they have different process IDs (PIDs).
- The child process usually inherits certain characteristics from the parent process, such as memory space and open file descriptors.
- Forking is a common method used in Unix-based operating systems.

2. Spawning or Creating a new process:

- In this method, the parent process explicitly requests the operating system to create a new process.
- Unlike forking, the new process is not a copy of the parent process, but rather a new process entirely.
- The new process typically begins execution at a specified entry point in the code.
- This method is common in modern operating systems and is used in platforms like Windows.

❑ Process Termination:

- A process terminates when it finishes executing its final statement and ask the operating system to delete it by using exit system call.
- All the resources of the process including physical and virtual memory, files, I/O buffers are de-allocated by the operating system.

Process Scheduling

- CPU scheduling is the basis of multi programmed OSs by switching the CPU among processes the operating system can make the computer more productive.
- On a single processor system only one process can run at a time; any others must wait until the CPU is true and can be rescheduled.
- The objective of multi programming is to have some process running at all times to maximize CPU utilization.
- The idea is as follows:-

- A process is executed until it must wait, typically for the completion of some I/O request.
- In a simple computer system, the CPU then sits idle.
- All this waiting time is wasted; no useful work is accomplished.
- With multi programming we try to use this time productively.
- Several processes are kept in memory at one time.

- When one process has to wait the operating system takes the CPU away from that process and gives the CPU to another process.
- This pattern continues every time one process has to wait another process can take over use of the CPU.
- Scheduling of this kind is a fundamental operating system function.
- Almost all computer resources are scheduled before use.
- The CPU is one of the primary computer resources; thus its scheduling is central to operating system design.

CPU-I/O Burst Cycle

- The success of CPU scheduling depends on an observation property of processes: Process execution consists of a "Cycle" of CPU execution and I/O wait.
- Processes execution begins with a CPU burst. That is followed by an I/O burst which is followed by another CPU burst, then another I/O burst and so on.

V. Imp **CPU Scheduler** (Short term Scheduler)

- Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed.

Ready $\xrightarrow{P}$ CPU running (dispatcher)

- The selection process is carried out by the short term scheduler or CPU scheduler.
- The CPU scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.

Preemptive & Non-preemptive Scheduling

- CPU – Scheduling decisions may take under the following four circumstances:

1. When a process switches from the running state to the waiting state. (I/O request).
2. When a process switches from the running state to the ready state (for example, when an interrupt occurs).
3. When a process switches from the waiting state to the ready state (for example at completion of I/O).
4. When a process terminates.

- For situation 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution.
- There is a choice, for situations 2 & 3.
- When scheduling takes place under circumstances 1 & 4, we say that the scheduling scheme is non - preemptive. Otherwise, it is preemptive.

Difference between Preemptive & Non-preemptive Scheduling

	PREEMPTIVE SCHEDULING	NON-PREEMPTIVE SCHEDULING
1.	In preemptive scheduling, resources (CPU) are allocated to a process for a limited time.	Once resources (CPU) are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
2.	Process can be interrupted in between.	Process can not be interrupted until it terminates itself or its time is up.
3.	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.

Difference between Preemptive & Non-preemptive Scheduling

	PREEMPTIVE SCHEDULING	NON-PREEMPTIVE SCHEDULING
4.	In preemptive scheduling, CPU utilization is high.	It is low in non preemptive scheduling.
5.	Preemptive scheduling waiting time is less.	Non-preemptive scheduling waiting time is high.
6.	Preemptive scheduling response time is less.	Non-preemptive scheduling response time is high.
7.	The OS has greater control over the scheduling of processes.	The OS has less control over the scheduling of processes.
8.	Examples of preemptive scheduling are Round Robin and Shortest Remaining Time First.	Examples of non-preemptive scheduling are First Come First Serve and Shortest Job First.

Scheduling Criteria

- Before start discussing about the scheduling algorithms, we need to discuss about the scheduling criteria on the basis of which scheduling algorithms can be compared and analyzed.
- In operating systems, scheduling criteria are used to determine the order in which processes are executed by the CPU.
- Different scheduling algorithms use various criteria to make decisions about process scheduling.
- Some common scheduling criteria include:
  1. CPU Utilization
  2. Throughput
  3. Turnaround Time
  4. Waiting Time
  5. Response Time

CPU Utilization:

- We want to keep the CPU as busy as possible. We do not want that the CPU remains idle.
- Maximizing CPU utilization is one of the primary goals of a scheduling algorithm. More the CPU is busy that means more the work is being done.
- A good scheduling algorithm tries to keep the CPU as busy as possible to ensure efficient resource utilization.

Throughput:

- If the CPU is busy executing processes, then the work is being done.
- Throughput refers to the number of processes that are completed per unit of time.
- Scheduling algorithms aim to maximize throughput by executing processes in a timely manner.

Turnaround Time:

- Turnaround time is the total time taken to execute a process from the time of submission to the time of completion.
- From the point of view of a particular process, the important criteria is how long it takes to execute that process.

- The interval from the time of submission of a process to the time of completion is the turnaround time.
- Turnaround time is the sum of the periods spent waiting to get into the memory, waiting in the ready queue, executing on the CPU and doing I/O.
- Scheduling algorithms strive to minimize turnaround time to improve system responsiveness.

❑ **Waiting Time:**

P0 — 12 ms
4 4 4

- Waiting time is the total time a process spends waiting in the ready queue before it gets executed.
- Minimizing waiting time helps in improving overall system performance.

❑ **Response Time:**

- Response time is the time taken from the submission of a request until the first response is produced.
- Scheduling algorithms aim to minimize response time to ensure that interactive processes receive prompt attention.

Fairness:

- Fairness ensures that each process gets a fair share of CPU time over the long run.
- Scheduling algorithms should prevent starvation and ensure that all processes receive adequate CPU time.

- Different scheduling algorithms prioritize these criteria differently based on the requirements of the system and the nature of the workload.

Scheduling Algorithms

- Scheduling algorithms are methods used in computer science and operating systems to manage the allocation of resources and execution of tasks or processes in an efficient manner.
- These algorithms determine the order in which tasks are executed on a system's central processing unit (CPU) or other resources, aiming to optimize factors such as throughput, response time, turnaround time, and fairness.
- There are several types of scheduling algorithms, each designed to address different system requirements and priorities.

Ready → Running

Some common scheduling algorithms include:

1. First-Come, First-Served (FCFS)
2. Shortest Job Next (SJN) / Shortest Job First (SJF) and Preemptive SJF or SRTN
3. Priority Scheduling (Preemptive and Non Preemptive)
4. Round Robin (RR)
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling

Scheduling Algorithms

- CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated with CPU.

Short term scheduler

Ready queue ⟹ CPU running state

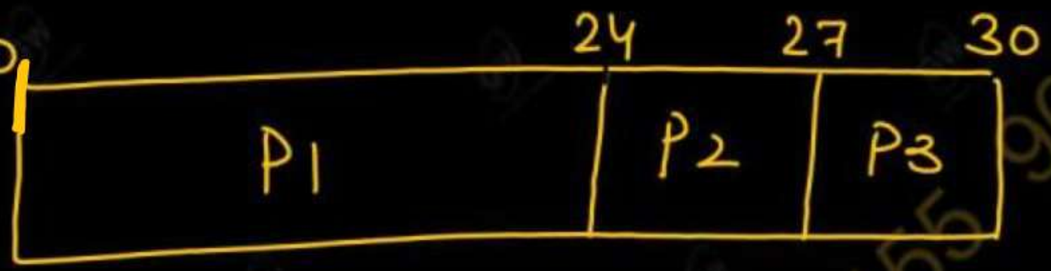
1. FCFS (First Come First Serve) Scheduling:

- FCFS (First-Come, First-Serve) is one of the simplest scheduling algorithms used in operating system.
- In FCFS scheduling, the process that arrives first is executed first, and so on.
- It operates on the principle of a queue (DS) where the processes are arranged in the order they arrive. (FIFO)
- To understand that how the CPU will be allocated to which process, consider the following set of processes that arrive at time 0:
- **Example 1:-** Consider the following three processes with CPU burst:-

Ready Processes ID	CPU Burst Time (in ms) (B.T.)
P1	24
P2	3
P3	3

- If the processes arrive in the order P1, P2 and P3 and are served in the FCFS order, we get the result shown in the following "Gantt Chart:" (WT, Avg. W.T.)

Soln:



- The Gantt chart is showing the waiting time of each process, now calculate the Average Waiting Time (ans=17)-

Processes ID	CPU Burst Time (ms)	Waiting Time	Average Waiting Time
P1	24	0 ✓	$\frac{0+24+27}{3} = \frac{51}{3}$ ✓ $= 17$ ms.
P2	3	24 ✓	
P3	3	27 ✓	
Total	30	51	

- If the processes arrive in the order P2, P3 and P1, we get the result shown in the following Gantt Chart:

- The Gantt chart is showing the waiting time of each process, now calculate the Average Waiting Time in this scenario (ans=3)-

Processes	CPU Burst Time (ms)	Waiting Time	Average Waiting Time
P1	24	6	$\frac{6+0+3}{3} = \frac{9}{3} = 3$ ms.
P2	3	0	
P3	3	3	
Total		9	

- In this scenario the average waiting time has reduced substantially from 17 ms to 3 ms.
- Thus the average waiting time under FCFS policy generally not minimal and may vary substantially if the process's CPU burst time vary greatly.

- The FCFS scheduling algorithm is non preemptive i.e. once the CPU has been allocated to process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O.
- The FCFS algorithm is thus particularly troublesome for time sharing systems, where it is important that each user get a share of the CPU at regular intervals (fare chance).
- It would be disastrous to allow one process to keep the CPU for an extended period.
- There is a "convoy effect" as all the other wait for the one big process to get off the CPU.
- This effect result in lower CPU and device utilization than might be possible if the shorten processes were allowed to go first.
- In other words the convoy effect may be defined as, " if the processes with higher CPU burst time arrived before the processes with smaller burst time, then, smaller processes have to wait for a long time for longer processes to release the CPU". This is a disadvantage of FCFS scheduling algorithm.

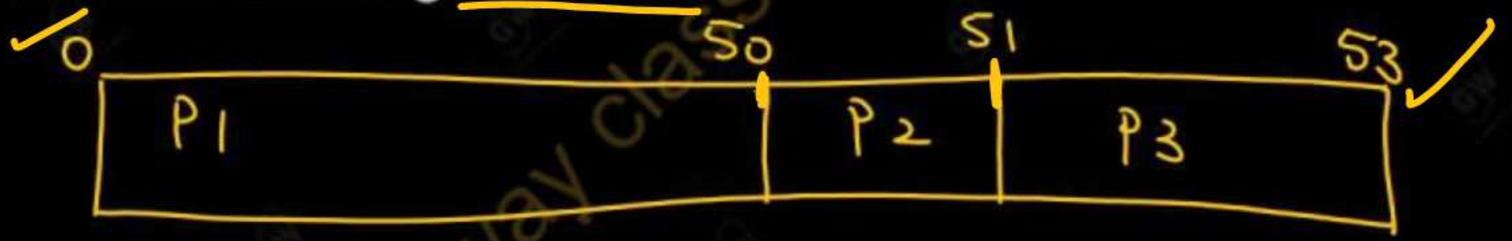
Example 2: Consider the following 3 processes with their Arrival Time and CPU Burst Time:

Processes ID	Arrival Time	CPU burst Time
P1	0	50
P2	1	1
P3	1	2

Calculate the (i) Average Turn Around Time and (ii) Average Waiting Time, if FCFS scheduling is followed.

TAT, ATAT, WT, AW.T.

Solution: Consider the following Gantt Chat:



We Know that:-

Turnaround Time = Completion Time – Arrival Time

CT-AT

Waiting Time = Turnaround Time – CPU Burst Time

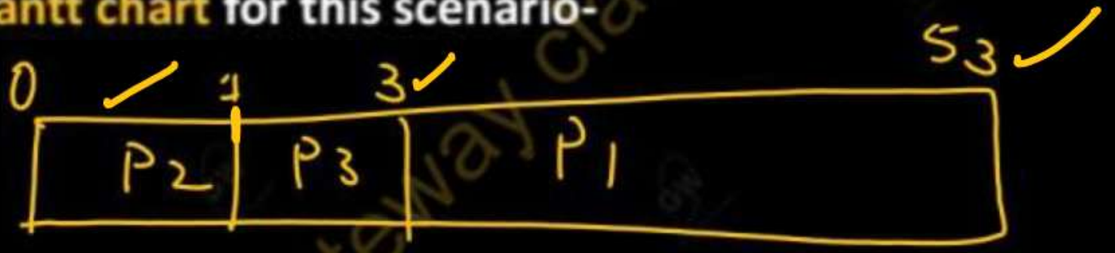
TT-CPUBT

Process I.D / Process / PID	Arrival Time A.T.	CPU Burst Time / Burst Time / B.T	Completion Time C.T.	Turnaround Time (T.A.T.) = C.T. – A.T.	Average T.A.T. (ans=50.6)	Waiting Time (W.T.) = T.A.T. – CPU B.T.(ans=33)	Average W.T.
P1	0	50	50	50 – 0 = 50	152 / 3 = 50.6 ms	50 – 50 = 0	99 / 3 = 33
P2	1	1	51	51 – 1 = 50		50 – 1 = 49	
P3	1	2	53	53 – 1 = 52		52 – 2 = 50	
Total				152		99	

- Now observe the different scenario and change the Arrival Time of the processes and schedule the processes with less CPU burst time before the processes has higher CPU burst time and observe the impact on Average Turnaround Time and Average Waiting Time:

Processes	Arrival Time	CPU burst Time
P1	1	50
P2	0	1
P3	0	2

- Arrival time is changed now.
- Draw the Gantt chart for this scenario-



- Now calculate the Average Turnaround Time and Average Waiting Time-

Process I.D.	A.T.	CPU B.T.	Completion Time (C.T.)	T.A.T. = C.T. –A.T.	Average T.A.T.	W.T.= T.A.T. – CPU B.T.	Average W.T.
P1	1	50	53	53-1 = 52	$\frac{56}{3} = 18.6$	52-50 = 2	$\frac{3}{3} = 1$
P2	0	1	1	1-0 = 1		1-1 = 0	
P3	0	2	3	3-0 = 3		3-2 = 1	
Total				56		3	

- It is observed that-

In the First Case	Average Waiting time =33
In the Second Case	Average Waiting time =1

- Because in first case, the process having a very long CPU burst time has arrived before the processes having less CPU burst time.
- In this case, processes with less CPU burst time or small processes has to wait for the processes having long CPU burst time because of this the average waiting time is being increased.

- In the second case, processes with less CPU burst time or small processes are coming first and do not wait for the execution of processes with long CPU burst time because of this the average waiting time is being decreased. This feature where average waiting time is being increased known as **Convoy effect**.

Example 5: Consider the following set of six processes with their arrival time and CPU burst time:

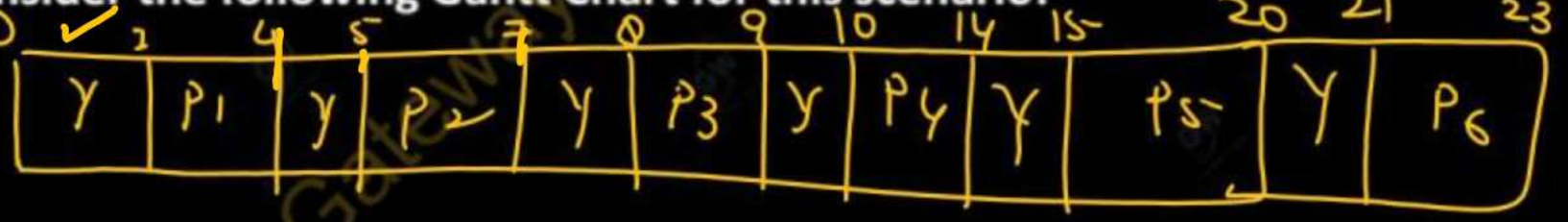
Process ID	Arrival Time	CPU Burst Time
P1	0	3
P2	1	2
P3	2	1
P4	3	4
P5	4	5
P6	5	2

FCFS

Joop

If FCFS scheduling algorithm is followed and there is 1 unit of overhead in scheduling the processes, find the efficiency of the algorithm.

Solution:- Consider the following Gantt Chart for this scenario:



Here Y denoted the unit overhead in scheduling the processes.

Now Calculate:

Useless or wasted time = 6 * = 6 * 1 = 6 units

Total Time = 23 units

Useful time = 23 – 6 = 17 units

Efficiency (η) = useful time / total time

= 17 / 23

= .7391

= 73.91 %

Ans

Shortest Job First (SJF) Scheduling Algorithm

n STS 1
Ready → Running

- The SJF algorithm selects the process with the smallest execution time next to execute.
- Shortest Job First (SJF) is a non-preemptive or preemptive scheduling algorithm that selects the waiting process with the smallest execution time to execute next.
- The idea behind SJF scheduling is to minimize the average waiting time of processes by executing shorter jobs first.
- If the CPU burst of two processes are the same, FCFS scheduling is used to break the tie.
- SJF can be implemented in two ways:

1. Non - Preemptive Shortest Job First (SJF) or Shortest Job Next (SJN)
2. Preemptive SJF or Shortest Remaining Time Next (SRTN) or Shortest Remaining time First (SRTF)

Non preemptive SJF or SJF or SJN

Preemptive SJF or SRTN or SRTF

Non-Preemptive SJF (Shortest Job First)

- In this version, once a process starts executing, it continues until it terminates or voluntarily relinquishes the CPU.
- The scheduler selects the process with the smallest burst time from the ready queue to execute next.
- This algorithm can lead to starvation for longer processes if shorter ones constantly arrive.
- Starvation refers to a situation where a process is unable to proceed because it's constantly being bypassed by other processes, even though it's eligible and waiting for execution.
- The SJF algorithm is optimal in that it give the minimum average waiting time for a given set of processes.
- Moving a short process before a long one decreases the waiting time of the short process more that it increase the waiting time of the long process. Consequently the average waiting time decreases.
- The real difficulty with SJF is knowing the length of the next CPU request.

Example 1: Consider the following four processes and CPU burst time:

Process ID	CPU Burst Time (ms)
P1	6
P2	8
P3	7
P4	3

SJF ⇒ non preemptive

Calculate the Average Waiting Time using SJF scheduling and compare the same with FCFS scheduling algorithm.

WT, AWT

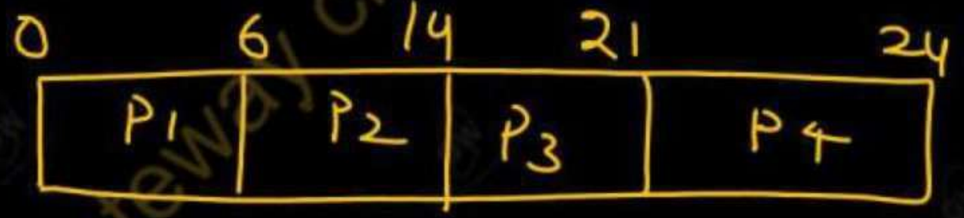
Solution:- Consider the following Gantt Chart following SJF:

Now Calculate the Waiting Time and Average Waiting Time:-

Process ID	CPU Burst Time	Waiting Time	Average Waiting Time (7)
P1	6	3	$\frac{28}{4}$
P2	8	16	$= 7$ ms
P3	7	9	
P4	3	0	
Total		28	

Now compare it with the FCFS scheduling:-

Consider the following Gantt Chart following FCFS scheduling:



Now Calculate the Waiting and Average Waiting Time considering the FCFS scheduling:-

Process ID	CPU Burst Time	Waiting Time	Average Waiting Time (10.25)
P1	6	0	4) 41 (10.25 ms
P2	8	6	4
P3	7	14	10
P4	3	21	8
Total		41	20

It is observed that -

Average Waiting Time in SJF	7 ms
Average Waiting Time in FCFS	10.25 ms

Preemptive SJF or Shortest Remaining Time Next (SRTN) SRTF

- The preemptive version of SJF is known as Shortest Remaining Time First or Shortest Remaining Time Next or SRTN where the currently executing process can be interrupted if a new process with a shorter CPU burst time arrives.
- In SRTN, a running process can be interrupted (preempted) by a new arriving process with a shorter burst time.
- When a new process arrives or the currently running process completes its CPU burst, the scheduler selects the process with the smallest remaining burst time.
- It requires the ability to estimate the remaining time for each process, which is usually not known in advance.
- Preemptive SJF involves more overhead due to context switching.

Example 2: Following preemptive SJF or SRTN or SRTF scheduling, consider the following four processes with arrival time and CPU burst time:

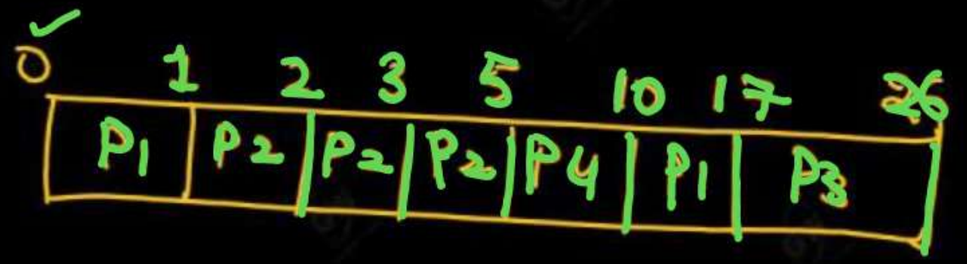
Process ID	Arrival Time	CPU Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

8-1=7 ✓
4-1=3-1=2-2=0 ✓
9-9=0 ✓
5-5=0 ✓
26

Calculate the Average Waiting Time using preemptive SJF scheduling and compare it with non preemptive SJF.

Solution:-

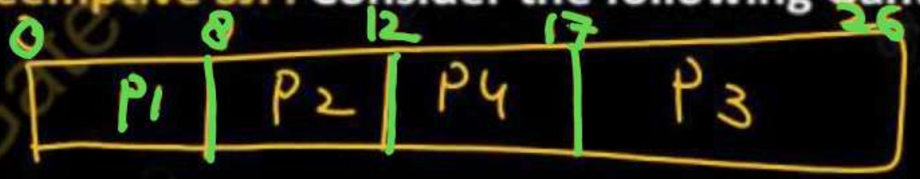
Preemptive SJF:- Consider the following Gantt Chart:-



Now calculate the Average Waiting Time:

Process ID	Arrival Time (A.T.)	CPU Burst Time	Completion Time (C.T.)	Turn Around Time TAT = CT - AT	Average Turn around Time (ATAT)	Waiting Time = TAT- CPU B.T.	Average Waiting Time (6.5)
P1	0	8	17	17-0=17	4)52(13	17-8=09	4)26(6.5
P2	1	4	5	05-1=04	4	04-4=0	24
P3	2	9	26	26-2=24	12	24-9=15	20
P4	3	5	10	10-3=07	12	07-5=02	20
TOTAL				52	X	26	X

Now compare it with non preemptive SJF: Consider the following Gantt Chart for non preemptive SJF:



❑ Now Calculate the Average Waiting Time:

Process ID	Arrival Time (A.T.)	CPU Burst Time	Completion Time (C.T.)	Turn Around Time TAT = CT - AT	Average Turn around Time (ATAT)	Waiting Time =TAT-CPU B.T.	Average Waiting Time(7.75)
					14.25		
P1	0	8	8 ✓	8 - 0 = 8 ✓		8-8=0 ✓	
P2	1	4	12 ✓	12-1=11 ✓		11-4=7 ✓	
P3	2	9	26 ✓	26-2=24 ✓		24-9=15 ✓	
P4	3	5	17 ✓	17-3=14 ✓		14-5=9 ✓	
TOTAL				57		✓ 31	

4) 57 (14.25
4
17
16
10
8
29

4) 31 (7.75
28
30
28
20
20

❑ It is observed that:-

Average Waiting Time in Non Pre-emptive SJF	7.75
Average Waiting Time in Pre-emptive SJF or SRTN	6.5

Problems With SJF

- While Shortest Job First (SJF) scheduling algorithm offers advantages like minimizing average waiting time, it also has several disadvantages:

- ***Difficulty in Predicting CPU Burst Times:***

✓ SJF requires accurate predictions or estimations of CPU burst times for each process.

✓ However, in real-world scenarios, it's often challenging to accurately predict the exact duration of CPU bursts.

✓ This limitation can lead to poor scheduling decisions, resulting in increased waiting times and decreased system performance.

✓ The real difficulty with SJF algorithm is knowing the length of the next CPU burst and it is a difficult process to know the next CPU burst.

✓ Although the SJF algorithm is optimal, it cannot be implemented at the level of short term CPU scheduling.

✓ Practically there is no way to know the length of next CPU burst.

- ✓ One approach is to try to approximate SJF scheduling.
- ✓ We may not know the length of the next CPU burst, but we may be able to predict its value.
- ✓ We expect that the next CPU burst will be similar in the length to the previous one.
- ✓ Thus by computing an approximation of the length of the next CPU burst, we can pick the process with the shortest predicted CPU burst.

- *Possibility of Starvation:*

- ✓ Long processes, also known as CPU-bound processes, may suffer from starvation in SJF scheduling.
- ✓ If short processes continually arrive, long processes might never get a chance to execute because they are constantly preempted by shorter ones.
- ✓ This can lead to unfairness and decreased overall system performance.

- ***Increased Complexity for Process Management:***

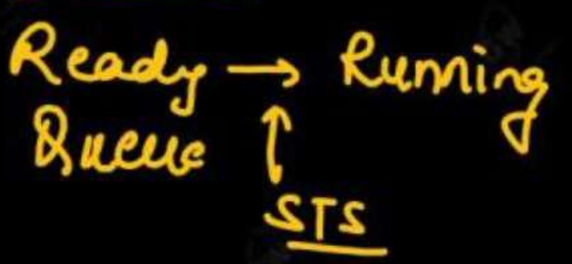
✓ Implementing SJF scheduling requires maintaining a ready queue of processes sorted based on their expected CPU burst times.

✓ This introduces additional complexity in process management, including the need for efficient sorting algorithms and data structures.

❑ While SJF scheduling can be beneficial in certain scenarios, its disadvantages make it less suitable for general-purpose operating systems, especially in environments with unpredictable workload characteristics or when precise CPU burst time estimations are challenging to obtain.

Priority Scheduling

- Priority scheduling is a scheduling algorithm where each process is assigned a priority.
- The scheduler selects the process with the highest priority for execution.
- Priority scheduling can be either non-preemptive or preemptive.
- Non-preemptive Priority Scheduling:



- Here, a running process is allowed to continue until it completes its CPU burst or voluntarily relinquishes the CPU.
- In this case, a higher-priority process can only be executed after the currently running process finishes.

Preemptive Priority Scheduling:

- In preemptive priority scheduling, a running process can be preempted by a higher-priority process arriving in the system.
- The CPU is then allocated to the higher-priority process, even if the currently executing process has not completed its CPU burst.

- However, it can suffer from starvation, where lower-priority processes may never get a chance to execute if higher-priority processes continually arrive.
- To mitigate starvation, some systems implement aging mechanisms to gradually increase the priority of waiting processes over time.
- Aging in OS is a scheduling technique used to prevent starvation in operating systems.
- It involves gradually increasing the priority of processes that have been waiting for a long time. It increases the chance of them getting the necessary resources to execute. Thus it reduces the risk of starvation.

Non–preemptive Priority Scheduling

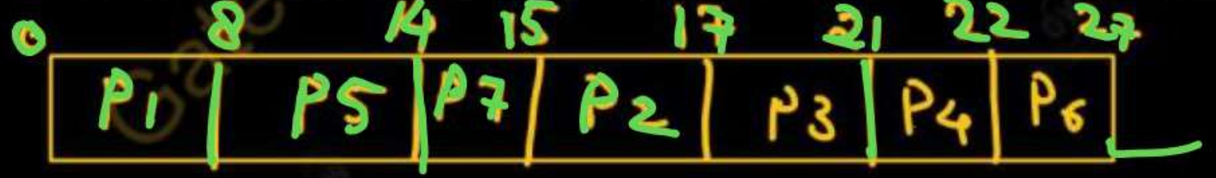
Example 1: Consider the following seven processes with their associated priority number, arrival time and CPU burst time:

Process ID	Priority Number	Arrival Time	CPU Burst Time
P1	3	0	8
P2	4	1	2
P3	4	3	4
P4	5	4	1
P5	2	5	6
P6	6	6	5
P7	1	10	1

27

Following the non-preemptive priority scheduling, calculate average turnaround time, average waiting time and average response time. Assume lesser the number higher the priority.

Solution:- Consider the following Gantt Chart for non preemptive priority scheduling:-



Response time

Non preemptive ⇒ waiting time

preemptive ⇒ Response time will be different

Now calculate the Average Turnaround Time, Average Waiting Time and Average Response Time:

Process ID	Priority No.	Arrival Time (A.T.)	CPU Burst Time	Completion Time (C.T.)	Turn Around Time TAT = CT - AT	Average Turn around Time (ATAT)(13.57)	Waiting Time =TAT-CPU B.T.	Average Waiting Time (9.71)	Response Time =First time CPU Allocation - AT	Average Response Time 9.71
P1	3	0	8	8	8 - 0 = 8		8 - 8 = 0		0 - 0 = 0	
P2	4	1	2	17	17 - 1 = 16	7)95(13.57	16 - 2 = 14	7)68(9.71	15 - 1 = 14	7)68(
P3	4	3	4	21	21 - 3 = 18	7 / 25 / 21 / 40 / 35 / 50 / 49 / 1	18 - 4 = 14	63 / 50 / 49 / 10 / 7 / 1	17 - 3 = 14	
P4	5	4	1	22	22 - 4 = 18		18 - 1 = 17		21 - 4 = 17	
P5	2	5	6	14	14 - 5 = 9		9 - 6 = 3		8 - 5 = 3	
P6	6	6	5	27	27 - 6 = 21		21 - 5 = 16		22 - 6 = 16	
P7	1	10	1	15	15 - 10 = 5		5 - 1 = 4		14 - 10 = 4	
TOTAL					95		68		68	

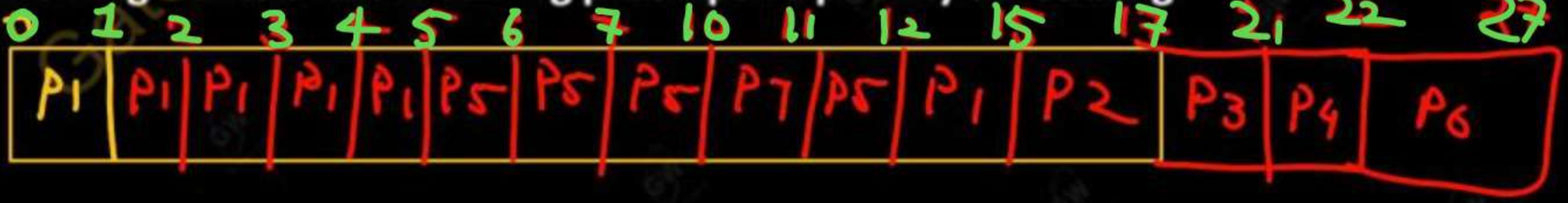
Preemptive Priority Scheduling

Example 2: Consider the following seven processes with the associated priority number, arrival time and CPU burst time:

Process ID	Priority No.	Arrival Time	CPU Burst Time
P1	3	0	8-1=7-1=6-1=5-1=4-1=3-3=0
P2	4	1	2-2=0
P3	4	3	4-4=0
P4	5	4	1-1=0
P5	2	5	6-1=5-1=4-3=1-1=0
P6	6	6	5-5=0
P7	1	10	1-1=0

Following the preemptive priority scheduling, calculate average turnaround time, average waiting time and response time. Assume lesser the number higher the priority.

Solution:- Consider the following Gantt Chart following preemptive priority scheduling:-



Now calculate the Average Turnaround Time, Average Waiting Time and Average Response Time:

Process ID	Priority No.	Arrival Time (A.T.)	CPU Burst Time	Completion Time (C.T.)	Turn Around Time TAT = CT - AT	Average Turn around Time (ATAT) (13.7)	Waiting Time =TAT-CPU B.T.	Average Waiting Time (9.85)	Response Time = First Time CPU Allocation-AT	Average Response Time (8.7)
P1	3	0	8	15	15-0=15		15-8=7		0-0=0	
P2	4	1	2	17	17-1=16		16-2=14		15-1=14	
P3	4	3	4	21	21-3=18		18-4=14		17-3=14	
P4	5	4	1	22	22-4=18		18-1=17		21-4=17	
P5	2	5	6	12	12-5=7		7-6=1		5-5=0	
P6	6	6	5	27	27-6=21		21-5=16		22-6=16	
P7	1	10	1	11	11-10=1		1-1=0		10-10=0	
TOTAL					96		69		61	

Avg

Avg

Avg

7)96(13.71
7
26
21
50
49
1

7)69(9.85
63
60
56
40
35
5

7)61(8.71
56
50
49
1

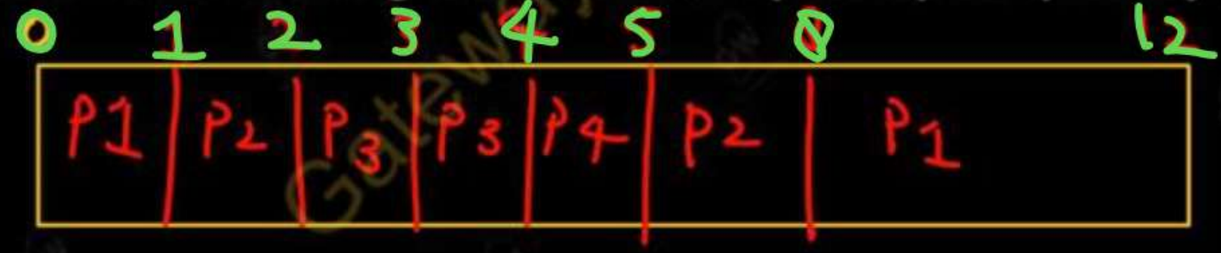
Example 4: Consider the following seven processes with the associated priority number, arrival time and CPU burst time:

Process ID	Priority No.	Arrival Time	CPU Burst Time
P1	10	0	5-1=4-4=0
P2	20	1	4-1=3-3=0
P3	30	2	2-1=1-1=0
P4	40	4	1-1=0

12

Following the preemptive priority scheduling, calculate average turnaround time, average waiting time and response time. Assume higher the priority number higher the priority.

Solution:- Consider the following Gantt Chart for preemptive priority scheduling:-



Now calculate the Average Turnaround Time, Average Waiting Time and Response Time:

Process ID	Priority No.	Arrival Time (A.T.)	CPU Burst Time	Completion Time (C.T.)	Turn Around Time TAT = CT - AT	Average Turn around Time (ATAT) (5.5)	Waiting Time =TAT-CPU B.T.	Average Waiting Time (2.5)	Response Time = First Time CPU Allocation- A.T. Ans 0
P1	10	0	5	12	12-0=12	4)22(5.5 20 20 20 (x)	12-5=7	4)10(2.5 8 20 20 y	0-0=0
P2	20	1	4	8	8-1=7		7-4=3		1-1=0
P3	30	2	2	4	4-2=2		2-2=0		2-2=0
P4	40	4	1	5	5-4=1		1-1=0		4-4=0
TOTAL					22		10		

Limitations with Priority Scheduling

- A major drawback with priority scheduling algorithm is "indefinite blocking or starvation".
- A process that is ready to run but waiting for the CPU can be considered blocked.
- A priority scheduling algorithm can leave some low priority processes waiting indefinitely.
- In a heavily loaded computer system, a steady stream of higher priority processes can prevent a low –priority process form ever getting the CPU.
- A solution to the problem of indefinite blockage of low–priority processes is aging.
- Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.
- For example increase the priority of a waiting process by 1 every 15 seconds etc..

Round Robin Scheduling / fairshare scheduling

- Round Robin (RR) scheduling is a preemptive scheduling algorithm.
- It is designed to handle Time-sharing systems, where multiple processes share the CPU.
- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- In Round Robin scheduling, each process is assigned a fixed time slice or time quantum.
- The CPU switches between processes in a circular order, executing each for a time period not exceeding the quantum i.e. the ready queue is treated as a circular queue.

Key features of Round Robin scheduling:

- Time Quantum -
  - Each process is allocated a small unit of time called a time quantum or time slice.
  - This time quantum defines the maximum amount of time a process can execute before it is preempted by the scheduler and moved to the end of the ready queue.

Circular Queue -

- Processes waiting in the ready queue are organized in a circular queue data structure.
- The scheduler selects the process at the front of the queue for execution.
- After its time quantum expires, the process is moved to the end of the queue, and the next process in line is selected.

Preemption -

- Round Robin scheduling is preemptive, meaning that the scheduler can interrupt the execution of a process when its time quantum expires, allowing other processes to execute.

Fairness -

- Since each process is given equal priority and execution time, Round Robin scheduling ensures fairness among processes.

- **Response Time -** Quick RR
  - Round Robin scheduling offers better response time compared to FCFS scheduling, as it allows the CPU to switch between processes at regular intervals.
- Round Robin scheduling is widely used in time-sharing systems, interactive systems, and in scenarios where fairness and preemptive scheduling are important.
- However, it may suffer from high context-switching overhead if the time quantum is too small or if there are too many processes in the system.
- Adjusting the time quantum is crucial to balancing fairness and minimizing overhead.

Example 1: Consider the following three processes with their CPU burst time:

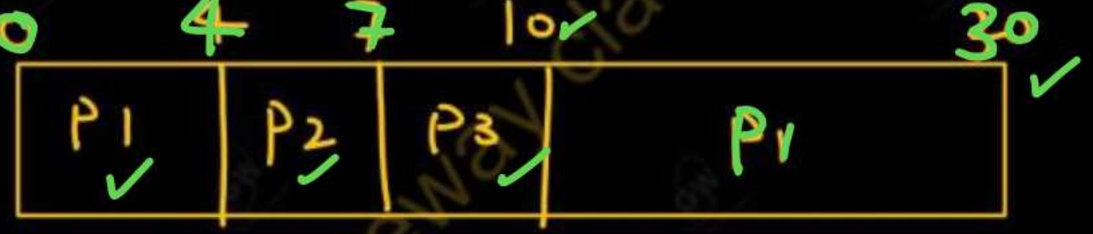
Process ID	AT	CPU Burst Time
P1	0	24 - 4 = 20
P2	0	3 - 3 = 0
P3	0	3 - 3 = 0
		30

RR

Calculate the average turnaround time and average waiting time using Round Robin scheduling algorithm. Assume the time quantum (time slice) is 4 ms and arrival time is 0.

Solution: Consider the following Gantt chart for Round Robin Scheduling:-

GANTT CHART



Ready Queue

RQ: P1 | P2 | P3 | P1

Now calculate the Average Turnaround Time and Average Waiting Time:

Process ID	CPU Burst Time	Completion Time (C.T.)	Turn Around Time (TAT) = CT – AT	Average Turn around Time (ATAT)(15.66)	Waiting Time =TAT-CPU B.T.	Average Waiting Time (5.66)
P1	24	30	30		30-24 = 6	
P2	3	7	7		7-3 = 4	
P3	3	10	10		10-3 = 7	
TOTAL			47		17	

Ans

3)47(15.66
3
17
15
20
18
20

Ans

3)17(5.66
15
20
18
2

Example 2: Consider the following five processes with their CPU burst time and arrival time:

Process ID	Arrival Time	CPU Burst Time
P1	0	8-3=5-3=2-2=0
P2	5	2-2=0
P3	1	7-3=4-3=1-1=0
P4	6	3-3=0
P5	8	5-3=2-2=0

25

Calculate the average turnaround time, average waiting time and average response time using Round Robin scheduling algorithm. Assuming the time quantum is 3 ms.

Solution:- Consider the following Gantt chart following Round Robin scheduling:-

GANTT CHART

Ready Queue

RQ: P1 P3 P1 P2 P4 P3 P5 P1 P3 P5

Now calculate the Average Turnaround Time, Average Waiting Time and average Response Time:

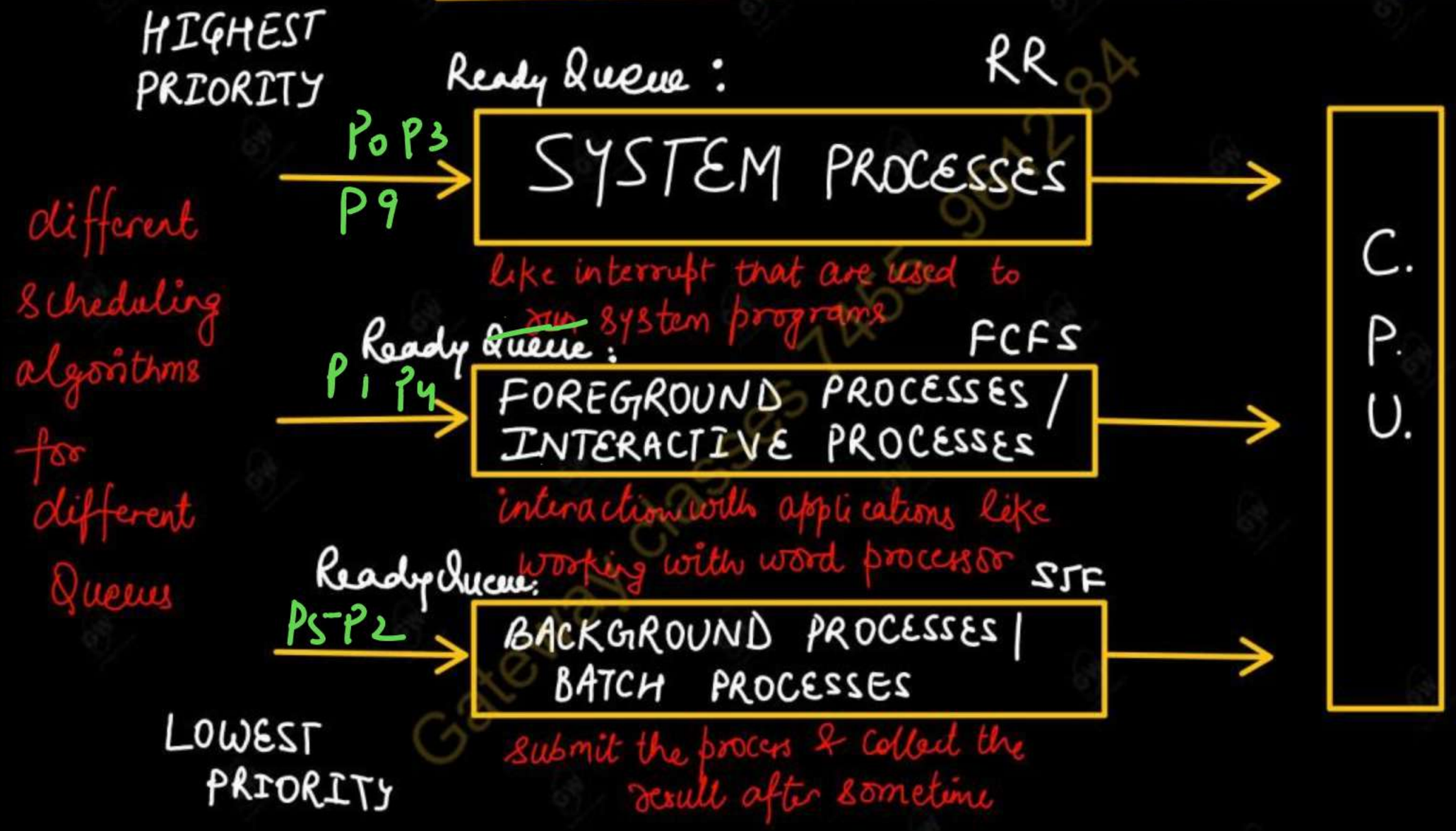
Process ID	Arrival Time	CPU Burst Time	Completion Time (C.T.)	Turn Around Time TAT = CT - AT	Average Turn around Time (ATAT)(15)	Waiting Time =TAT-CPU B.T.	Average Waiting Time (10)	Response Time= First Time CPU All. – A.T.	Average Response Time (4)
P1	0	8	22	22-0 =22	5)75(15	22-8=14	10	0-0=0	4
P2	5	2	11	11-5=6		6-2=4	5)50(	9-5=4	5)20(
P3	1	7	23	23-1=22		22-7=15		3-1=2	
P4	6	3	14	14-6=8		8-3=5		11-6=5	
P5	8	5	25	25-8=17		17-5=12		17-0=9	
TOTAL				75		50		20	

Ready

Multilevel Queue Scheduling

- In this scheduling processes are classified into groups. A common division is made between foreground (interactive) processes, background (batch) processes or some time real-time processes.
- These types of processes have different response-time requirements and so may have different scheduling needs.
- In addition, foreground processes may have priority (externally defined) over background processes.
- A multilevel queue scheduling algorithm partitions the ready queue into several separate queues. In other scheduling, there was only one ready queue for all processes.

MULTILEVEL QUEUE SCHEDULING



Some Key Characteristics of Multilevel Queue (MLQ) Scheduling

- **Multiple Queues:**
  - In MLQ scheduling, processes are divided into multiple queues based on their priority, with each queue having a different priority level.
- **Priority Assignment:**
  - Priorities are assigned to processes based on their type, characteristics, and importance.
  - For example, interactive processes like user input/output may have a higher priority than batch processes like file backups.
- **Preemption:**
  - Preemption is allowed in MLQ scheduling, which means a higher priority process can preempt a lower priority process, and the CPU is allocated to the higher priority process.

Scheduling Algorithm:

- Different scheduling algorithms can be used for each queue, depending on the requirements of the processes in that queue.
- For example, Round Robin (RR) scheduling may be used for interactive processes, while First Come First Serve (FCFS) scheduling may be used for batch processes.

Multilevel Feedback Queue Scheduling

- In multilevel queue scheduling algorithm, processes are permanently assigned to queue when they enter the system. Processes do not move form one queue to the other.
- Multilevel Feedback Queue Scheduling (MLFQ) CPU Scheduling is like Multilevel Queue (MLQ) Scheduling but in this process can move between the queues.
- The idea is to separate processes according to the characteristics of their CPU busts.
- If a process uses too much CPU time, it will be moved to a lower priority queue.
- In addition, 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.

Features of Multilevel Feedback Queue (MLFQ) Scheduling

- **Multiple Queues:**
  - Similar to MLQ scheduling, MLFQ scheduling divides processes into multiple queues based on their priority levels.
  - However, unlike MLQ scheduling, processes can move between queues based on their behavior and needs.
- **Priorities Adjusted Dynamically:**
  - The priority of a process can be adjusted dynamically based on its behavior.
  - Higher-priority processes are given more CPU time and lower-priority processes are given less.

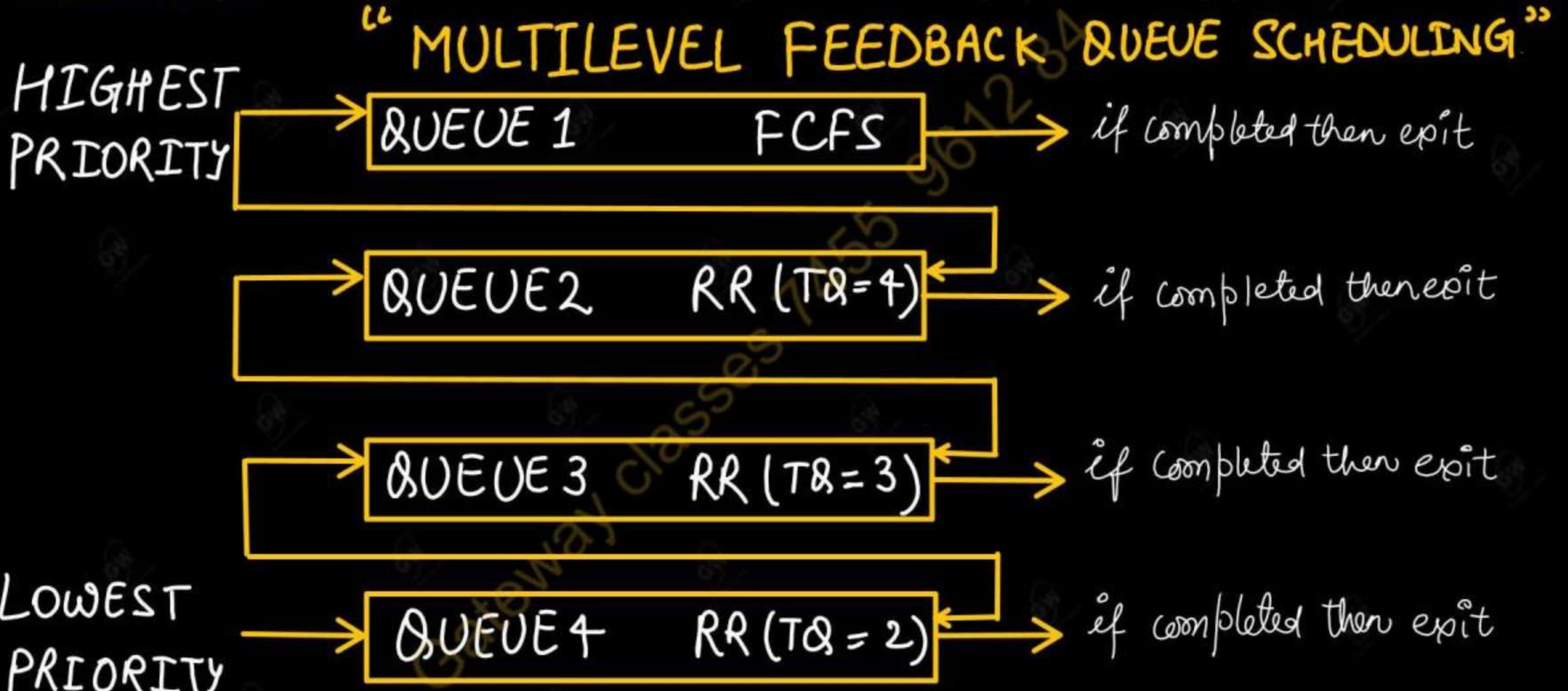
Feedback Mechanism:

- MLFQ scheduling uses a feedback mechanism to adjust the priority of a process based on its behavior over time.
- For example, if a process in a lower-priority queue uses up its time slice, it may be moved to a higher-priority queue to ensure it gets more CPU time.

Preemption:

- Preemption is allowed in MLFQ scheduling, meaning that a higher-priority process can preempt a lower-priority process to ensure it gets the CPU time it needs.

- For example, consider a multilevel feedback queue scheduling with following queues-



- In general, a multilevel feedback queue scheduler is defined by the following parameters:

(i) The number of queues

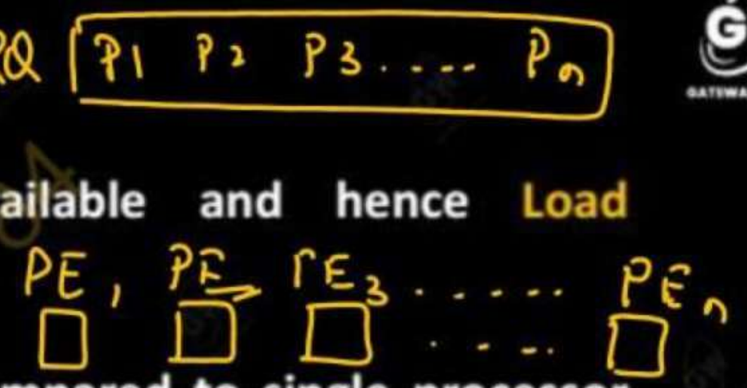
(ii) The scheduling algorithm for each queue.

(iii) The method used to determine when to upgrade a process to the higher priority queue.

(iv) The method used to determine when to demote a process to a lower priority queue.

(v) The method used to determine which queue a process will enter when that process needs service.

Multiprocessor Scheduling



- In multiple-processor scheduling, multiple CPU's are available and hence Load Sharing becomes possible.
- However, multiple processor scheduling is more complex as compared to single processor scheduling.
- Multiprocessor scheduling refers to the process of efficiently assigning tasks or processes to multiple processors in a multiprocessor system.
- Unlike single-processor systems where tasks are scheduled to run on a single CPU, multiprocessor systems have the advantage of parallel processing, allowing multiple tasks to execute simultaneously.

Why is multiple-processor scheduling important?

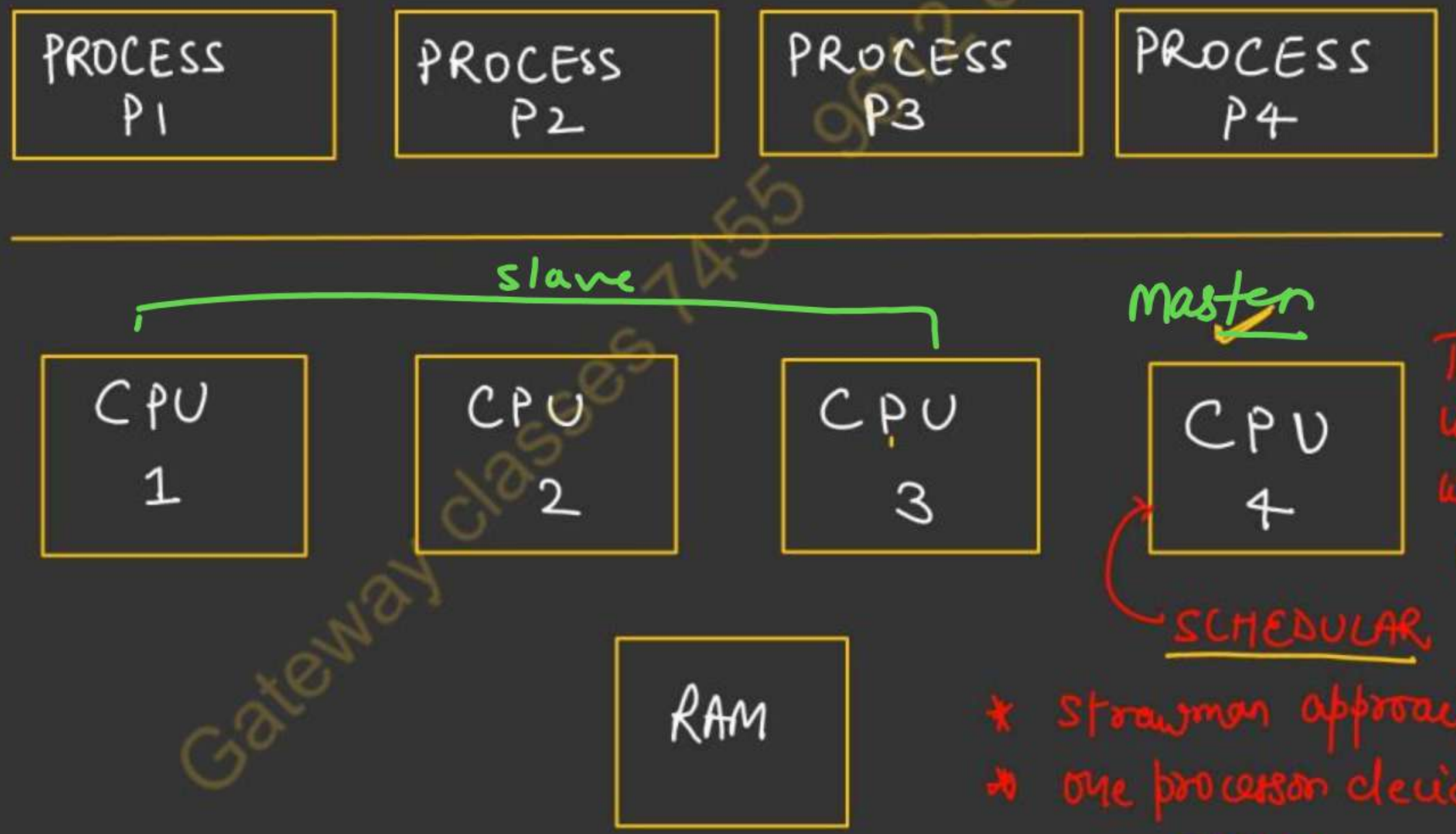
- Multiple-processor scheduling is important because it enables a computer system to perform multiple tasks simultaneously, which can greatly improve overall system performance and efficiency.

Approaches to Multiple-Processor Scheduling

- One approach is when all the scheduling decisions and I/O processing are handled by a single processor which is called the Master Server and the other processors executes only the user code. This is simple and reduces the need of data sharing. This entire scenario is called Asymmetric Multiprocessing.
- A second approach uses Symmetric Multiprocessing where each processor is self scheduling.
- All processes may be in a common ready queue or each processor may have its own private queue for ready processes.
- The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.

* Simple to implement

ASYMMETRIC MULTIPROCESSOR SCHEDULING



* Strawman approach
* One processor decides for everyone

Advantages -

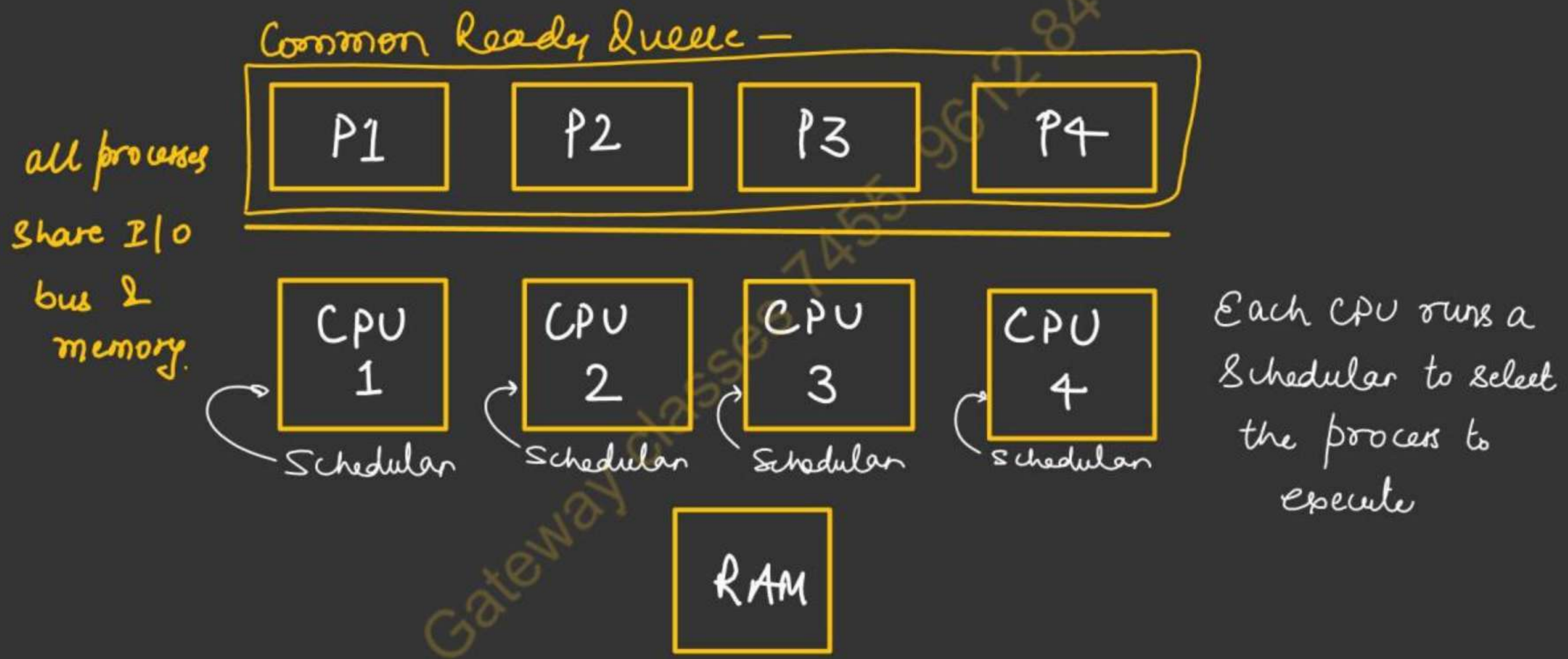
① Easy to implement

② It will have one local queue wherein all ready processes will be available, based on some mechanism the process will be scheduled from the local queue to run on any one of the CPU

disadvantages -

① Performance degradation – All CPUs are dependent on one master CPU Scheduler to provide some process for execution.

SYMMETRIC MULTIPROCESSOR SCHEDULING

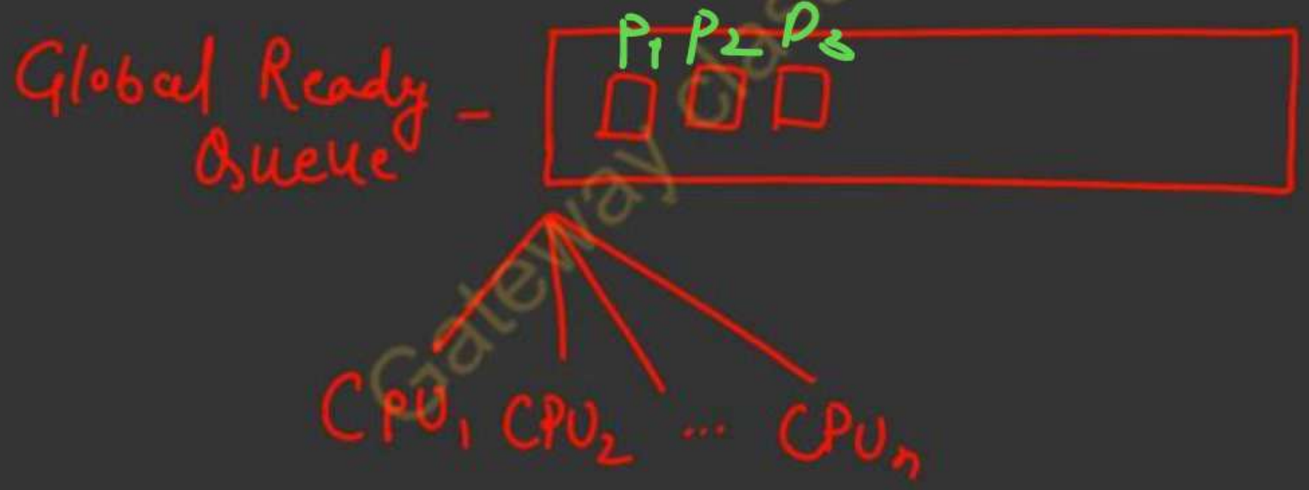


Two approaches -

(i) global queue

(ii) Each CPU has its own local queue

(i) Symmetrical scheduling using global queue :-



* Same CPU processor may select same process from global ready queue

Solution - At a time, only one processor will access the RQ.

Advantages - Good CPU utilization, fair to all processes

Disadvantages - ① Not scalable - Contention for the global queue

② Locking mechanism is needed in scheduling

↳ not a good idea

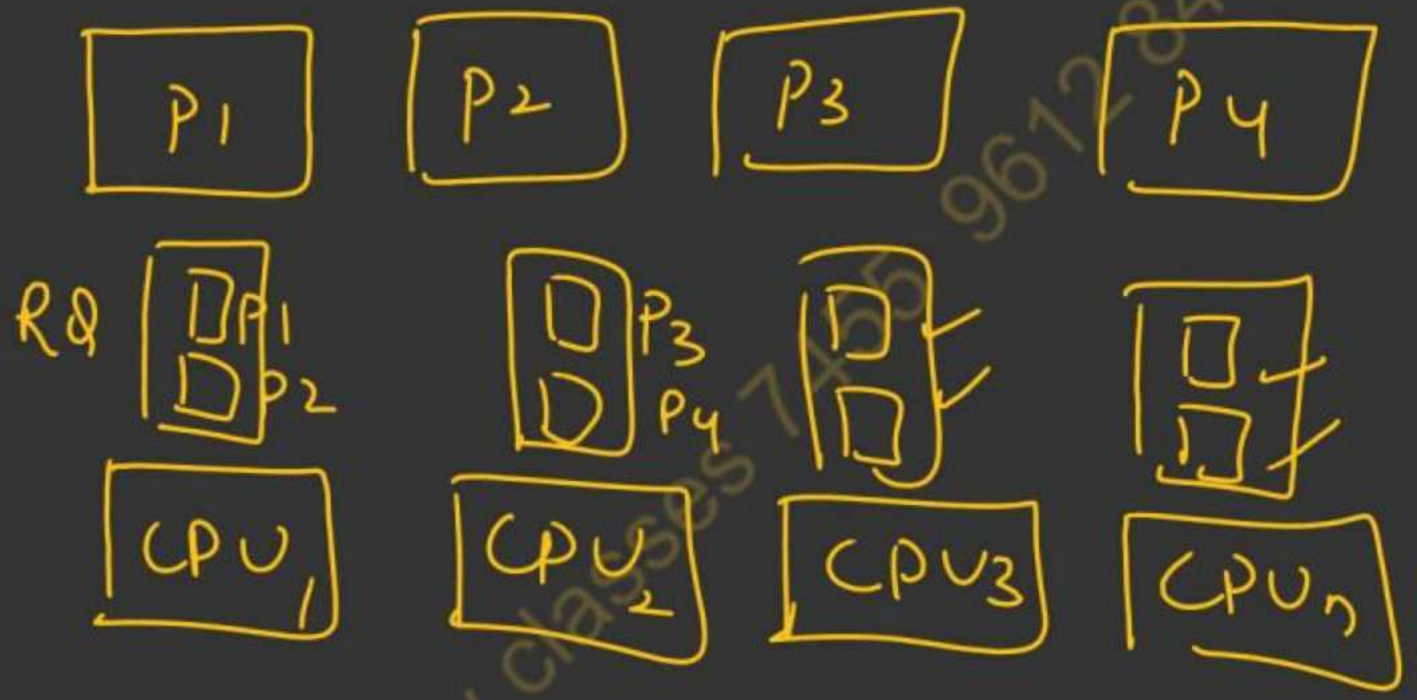
② Symmetrical Scheduling with separate queue for each CPU -

* It uses static partitioning of processes

Advantage - ① Easy to implement

② No locking mechanism required

③ Scalable



Disadvantage - ① load imbalance - some CPUs ready queue may have many processes, some may have less

How does multiple-processor scheduling work?

- **Load Balancing:**
  - Multiple-processor scheduling works by dividing tasks among multiple processors in a computer system, which allows tasks to be processed simultaneously and reduces the overall time needed to complete them.
  - The operating system tries to evenly distribute the workload among the available processors to maximize overall system throughput.
- Load balancing algorithms monitor the utilization of each processor and may migrate tasks between processors to achieve balance.
- There are two general approaches to load balancing:
  - **Push Migration** – In push migration a task routinely checks the load on each processor and if it finds an imbalance then it evenly distributes load on each processors by moving the processes from overloaded to idle or less busy processors.
  - **Pull Migration** – Pull Migration occurs when an idle processor pulls a waiting task from a busy processor for its execution.

❑ Processor Affinity:

- Processor Affinity means a processes has an affinity for the processor on which it is currently running.
- When a process runs on a specific processor there are certain effects on the cache memory.
- The data most recently accessed by the process populate the cache for the processor and as a result successive memory access by the process are often satisfied in the cache memory.
- Now if the process migrates to another processor, the contents of the cache memory must be invalidated for the first processor and the cache for the second processor must be repopulated.
- Because of the high cost of invalidating and repopulating caches, most of the SMP(symmetric multiprocessing) systems try to avoid migration of processes from one processor to another and try to keep a process running on the same processor.
- This is known as **Processor Affinity**.

Synchronization:

- In multiprocessor systems, tasks running on different processors may need to synchronize their execution, for example, to coordinate access to shared resources or to ensure consistency.
- Synchronization mechanisms like locks, semaphores, and barriers are used to manage concurrency and maintain system integrity.

Priority-Based Scheduling:

- Multiprocessor scheduling may involve assigning priorities to tasks based on factors such as their importance, deadlines, or resource requirements.
- Priority-based scheduling algorithms ensure that critical tasks are executed in a timely manner and that system resources are allocated efficiently.

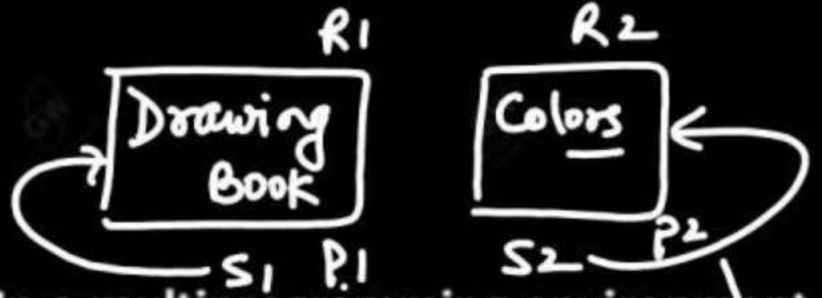
❑ Parallelism and Concurrency:

- Multiprocessor scheduling exploits parallelism and concurrency to improve system performance.
- Parallelism involves executing multiple tasks simultaneously on different processors, while concurrency involves interleaving the execution of multiple tasks to make efficient use of processor time.

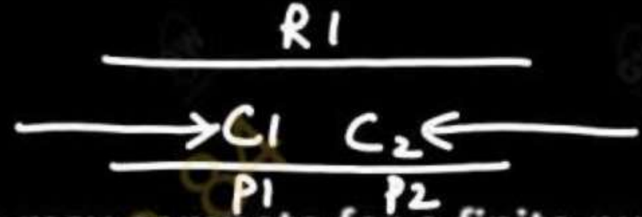
❑ Dynamic Scheduling:

- In dynamic scheduling, tasks are scheduled at runtime based on system conditions and workload characteristics.
- Dynamic scheduling algorithms adapt to changes in workload, resource availability, and system load to optimize performance.

❑ Multiprocessor scheduling plays a crucial role in maximizing the performance and efficiency of multiprocessor systems by effectively managing system resources and workload distribution.



Deadlock

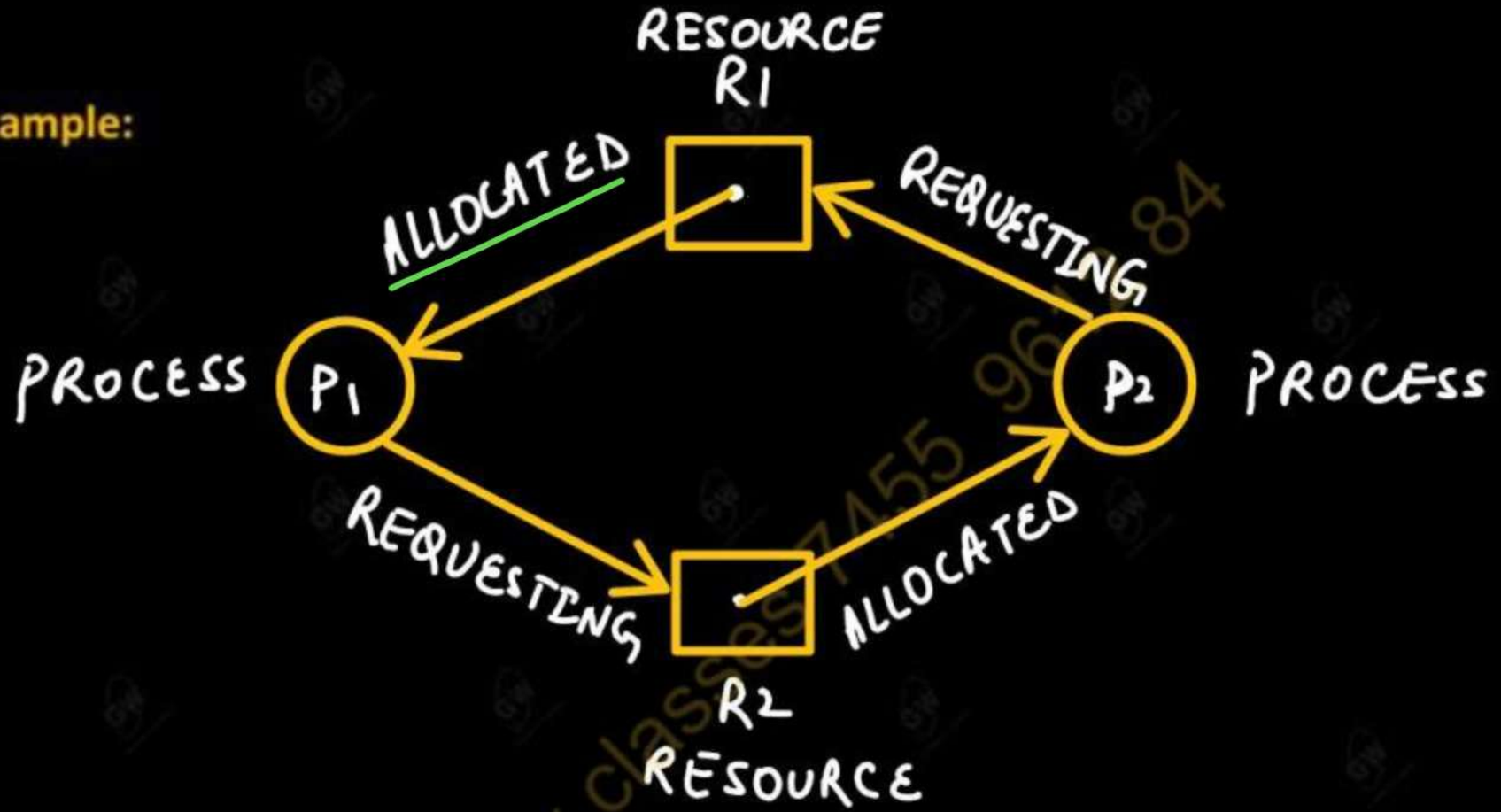


- In a multiprogramming environment, several processes may compete for a finite number of resources.

waiting state waiting state

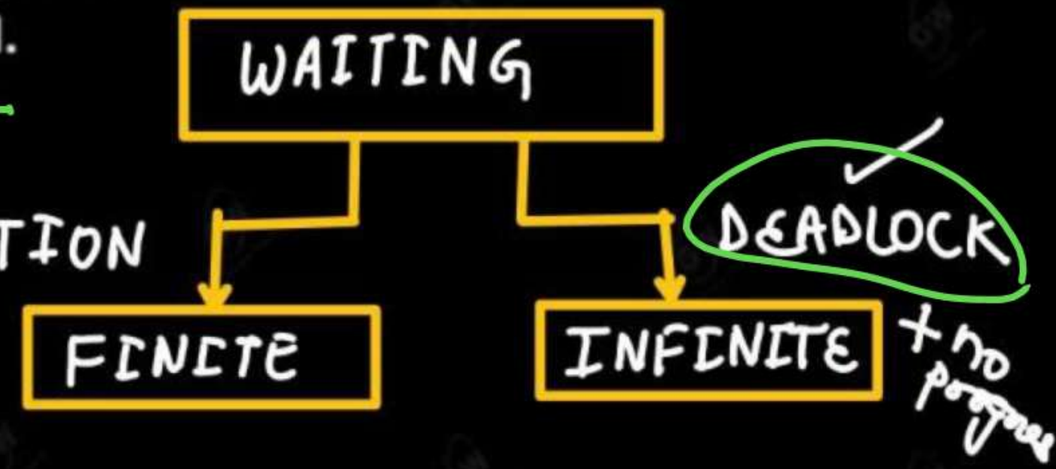
- A process requests resources; if the resources are not available at the time, the process enters a waiting state.
- Sometimes, a waiting process is never again able to change state because the resources it has requested are hold by other waiting processes. This situation is called a deadlock.
- A process is waiting for a resource which is held by another waiting process. (deadlock)
- If two or more processes are waiting on happening of some event but that event does not happen that is called the deadlock and those processes are in deadlock state.

- For example:



- Resources R1 is held by process P1 and process P1 is requesting for recourse R2 and
- Recourse R2 is held by process P2 and process P2 is requesting for recourse R1
- This situation is a kind of deadlock.

- Waiting for something for infinite time in which there is no progress for waiting processes.
- Processes waits for one another's action indefinitely.
- Deadlock is different from starvation.
- In starvation we can not say that there is no progress but in deadlock there is no progress.
- In starvation because of higher priority processes, low priority processes do not get CPU but here progress is there i.e. CPU is running some other processes i.e. progress is there, CPU is not blocked.
- But in deadlock no progress is there and no process is getting CPU.
- In starvation, after some time process may be executed.



- A process must request for a resource before using it and must release the resource after using it.
- A process may request as many resources as it requires to carry out its designated task.
- Under the normal mode of operation a process may utilize a resource in only the following sequence:

a) **Request:-** The process requests the resource. If the request can not be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.

b) **Use:-** The process can operate on the resource for example, if the resource is a printer the process can print on the printer.

c) **Release:-** The process releases the resource.

Deadlock Characterization

❑ Four Necessary Conditions:

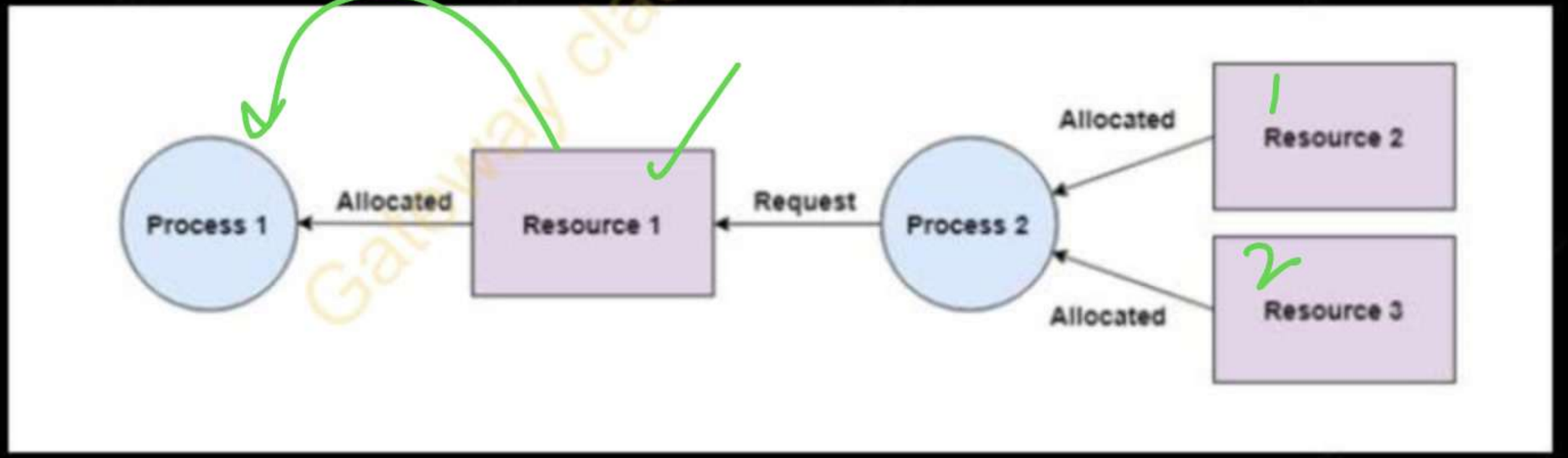
A deadlock situation can arise if the following four conditions (Coffman conditions) hold simultaneously in a system.

1. Mutual Exclusion:

- This condition requires that at least one resource be held in a non-shareable mode, which means that only one process can use the resource at any given time.
- If another process requests that resource, the requesting process must be delayed until the resource has been released.
- In other words, there should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.

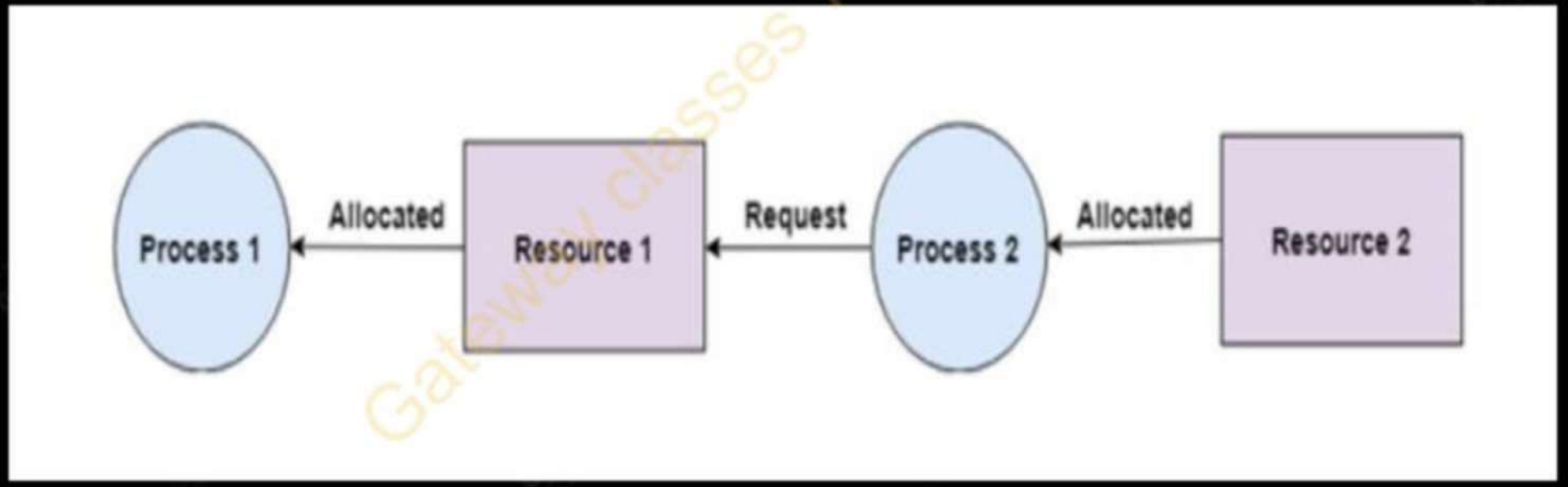
2. Hold & Wait:

- The hold and wait condition specifies that a process must be holding at least one resource while waiting for other processes to release resources that are currently held by other processes.
- In other words, a process can hold multiple resources and still request more resources from other processes which are holding them.
- In the diagram, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



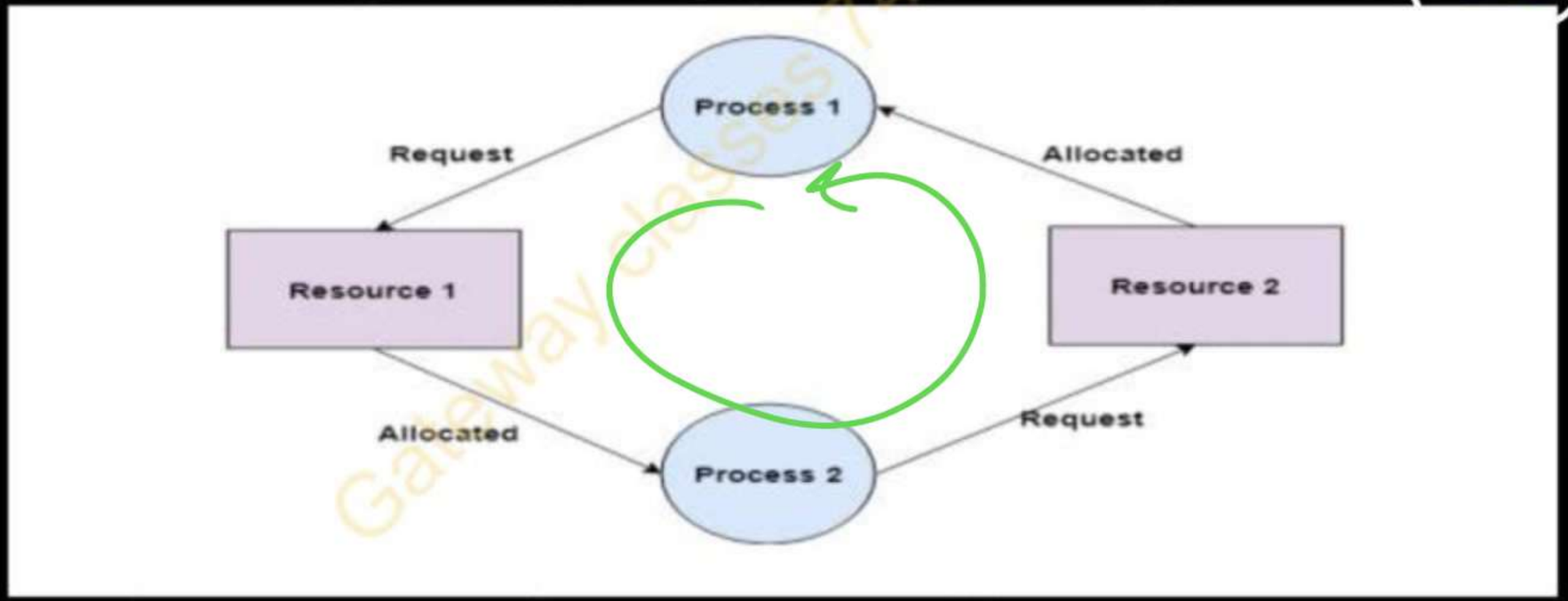
3. No Preemption:

- Resources cannot be preempted; that is a resource can be released only voluntarily by the process holding it, after that process has completed the task.
- In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



4. Circular Wait:

- A set $\{P_0, P_1, P_2, \ldots\ldots P_n\}$ of waiting processes must exist such that P_0 is waiting for a resource held by process P_1, process P_1 is waiting for a resource held by process P_2, ... ,process Pn-1 and waiting for a resource held by process P_n and process P_n is waiting for a resource held by P_0.
- **For example:** Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



- ❑ All four above mentioned conditions must hold for a deadlock to occur.

Difference Between Deadlock and Starvation

S. No.	Deadlock	Starvation
1.	All processes keep waiting for each other to complete and none get executed i.e. No Progress.	High priority processes keep executing and low priority processes are blocked.
2.	Resources are blocked by the processes.	Resources are continuously utilized by high priority processes.
3.	Necessary conditions Mutual Exclusion, Hold and Wait, No preemption, and Circular Wait.	Priorities are assigned to the processes.
4.	Also known as Circular wait.	Also known as lived lock.
5.	It can be prevented by avoiding the necessary conditions for deadlock.	It can be prevented by Aging.

Resource – Allocation Graph (RAG)

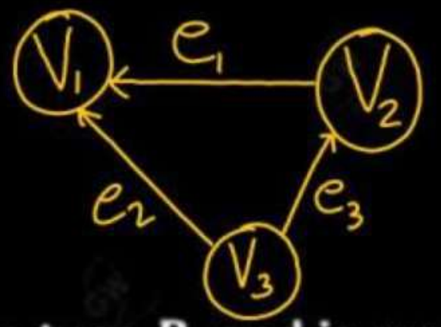
- Deadlock can be described more precisely in terms of a directed graph called a resource allocation graph or system resource allocation graph (RAG). SRAG
- A resource allocation graphs shows which resource is held by which process and which process is waiting for a resource of a specific kind.
- This graph consists of a set of vertices V (Process and Resource Vertex) and a set of edges E.
- The set of vertices V is partitioned into two different types of nodes:

→ 1. Set of Processes –

P = { P_1, P_2, P_3 ..... P_n}, the set consisting of all the active processes in the system.

→ 2. Set of Resources –

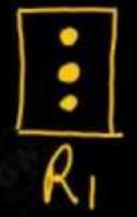
R = { R_1, R_2, R_3 ..... R_n}, the set consisting of all resource types in the system.



- A directed edge from process P_i to resource type R_j is denoted by

$$P_i \rightarrow R_j;$$

It signifies that process P_i has requested an instance of resource type R_j and is currently waiting for that resource.



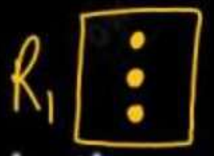
- A directed edge from resource type R_j to process P_i is denoted by

$$R_j \rightarrow P_i;$$

It signifies that an instance of resource type R_j has been allocated to process P_i.

- A directed edge $P_i \rightarrow R_j$ is called a "request edge";
- A directed edge $R_j \rightarrow P_i$ is called an "assignment edge".

- Pictorially, we represent each process P_i as a circle and each resource type R_j as a rectangle.
- Since resource type R_j may have more than one instances, we represent each such instance as a dot within the rectangle.



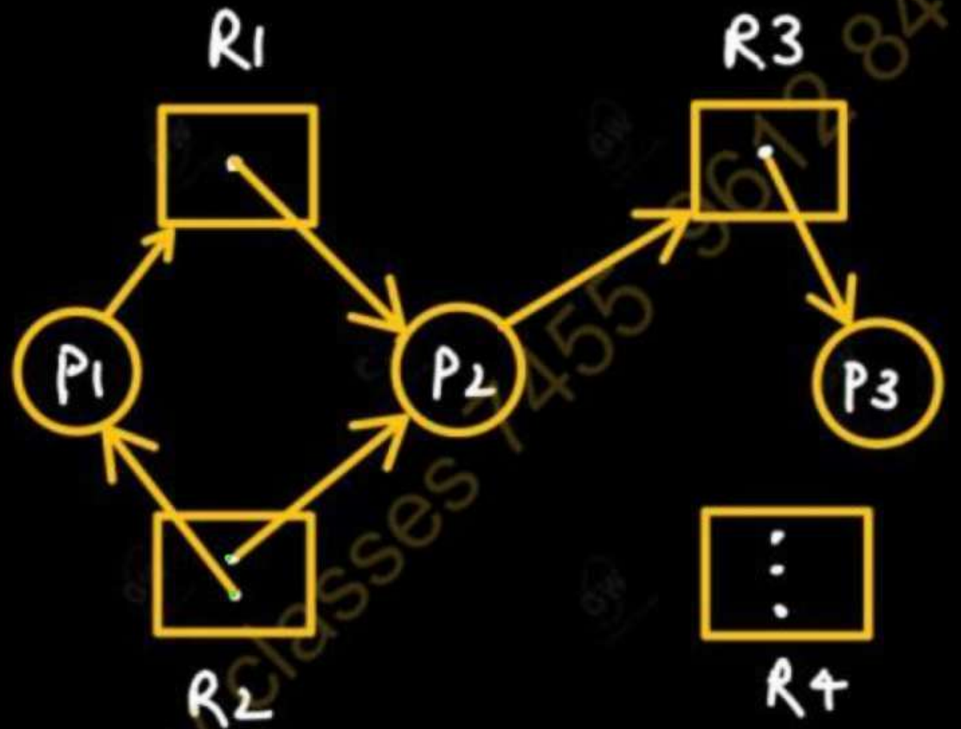
- A request edge points to only the rectangle R_j where as an assignment edge must also designate one of the dots in the rectangle.

- When process P_i requests an instance of resource type R_j a request edge is inserted in the resource allocation graph.

- When this request can be fulfilled the request edge is "instantaneously" transformed to an assignment edge.
- When the process no longer needs a access to the resource; as a result the assignment is deleted.

Example 1 :- Consider the following resource – allocation graph:

RAG



$R_1 - 1$
$R_2 - 2$
$R_3 - 1$
$R_4 - 3$

P_1 P_2 P_3

The set P, R and E-

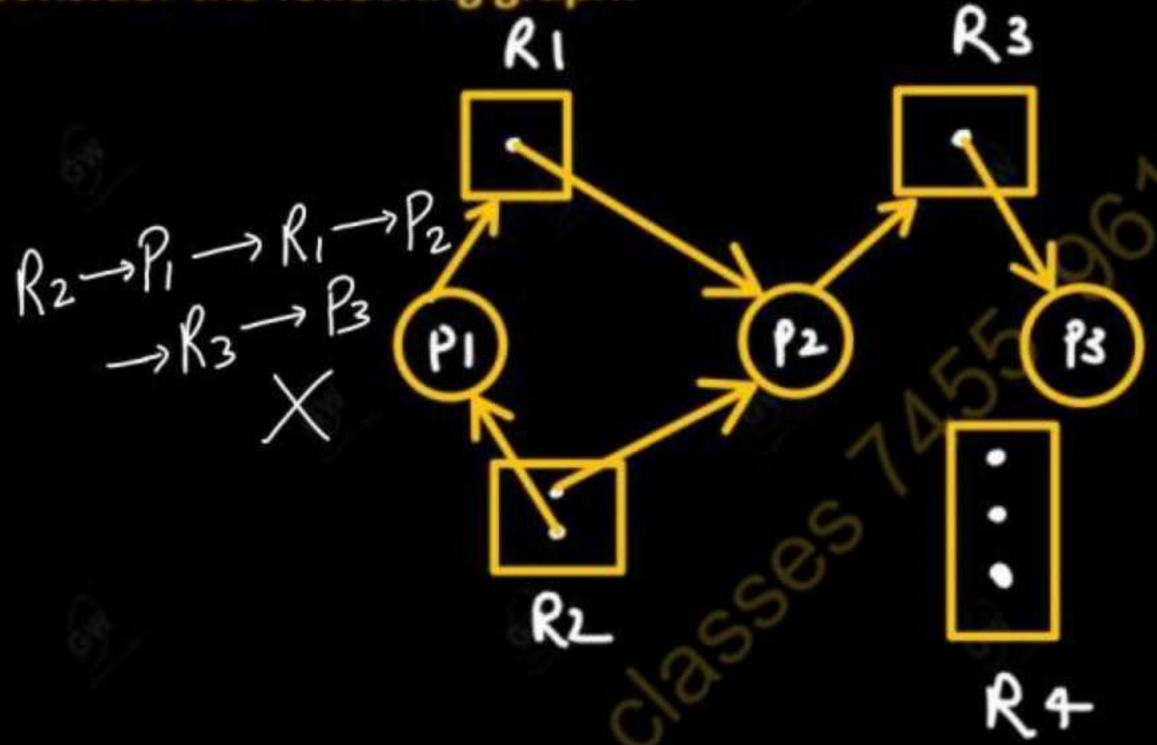
Set of P = { P1, P2, P3 }

Set of R = { R1, R2, R3, R4 }

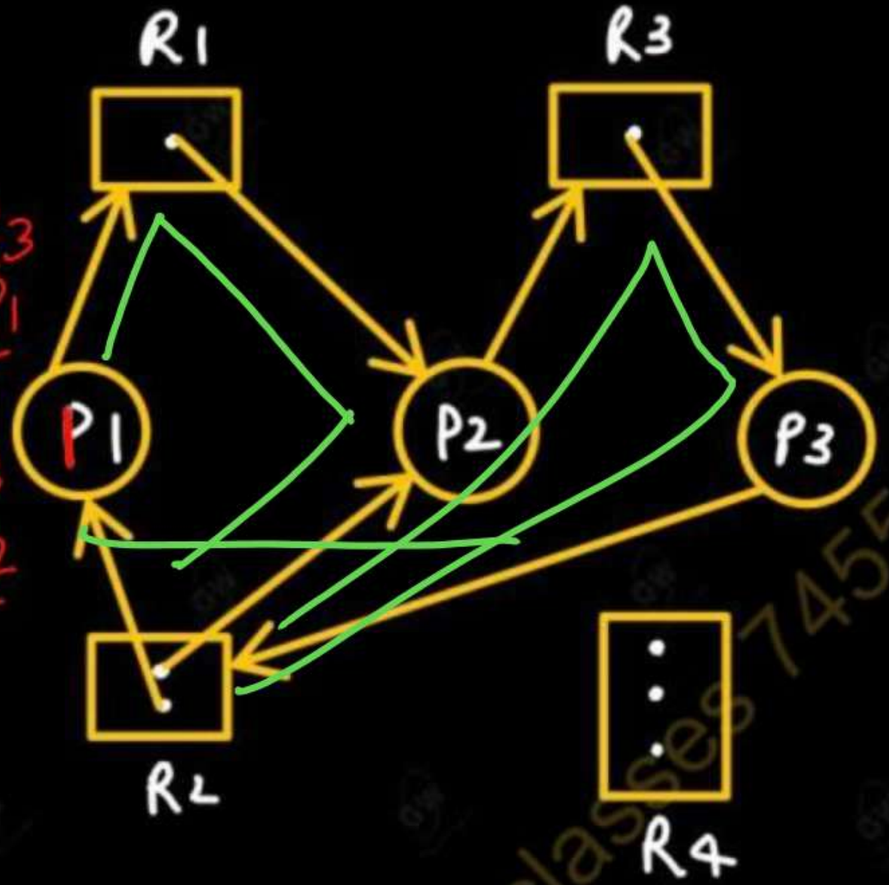
Set of E = { P1 → R1 , P2 → R3, R1 → P2, R2 → P2, R2 → P1, R3 → P3 }

- **Resource Instances:**
  - One instance of Resource type R_1
  - Two instances of Resource type R_2
  - One instance of Resource type R_3
  - Three instances of Resource type R_4
- **Process States:**
  - Process P_1 is holding an instance of Resource type R_2 and is waiting for one instance of Resource type R_1.
  - Process P_2 is holding an instance of Resource type R_1 and an instance of Resource type R_2 and requesting for an instance of Resource type R_3.
  - Process P_3 is holding an instance of resource type R_3.
- *Given the definition of a resource – allocation graph it can be shown that if the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle then a deadlock may exist.*

Example 2: Consider the following graph:



- Suppose that P_3 requests an instance of Resource type R_2.
- Since no Resource instance is currently available, a request edge is added to the graph as shown in the next graph –



$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

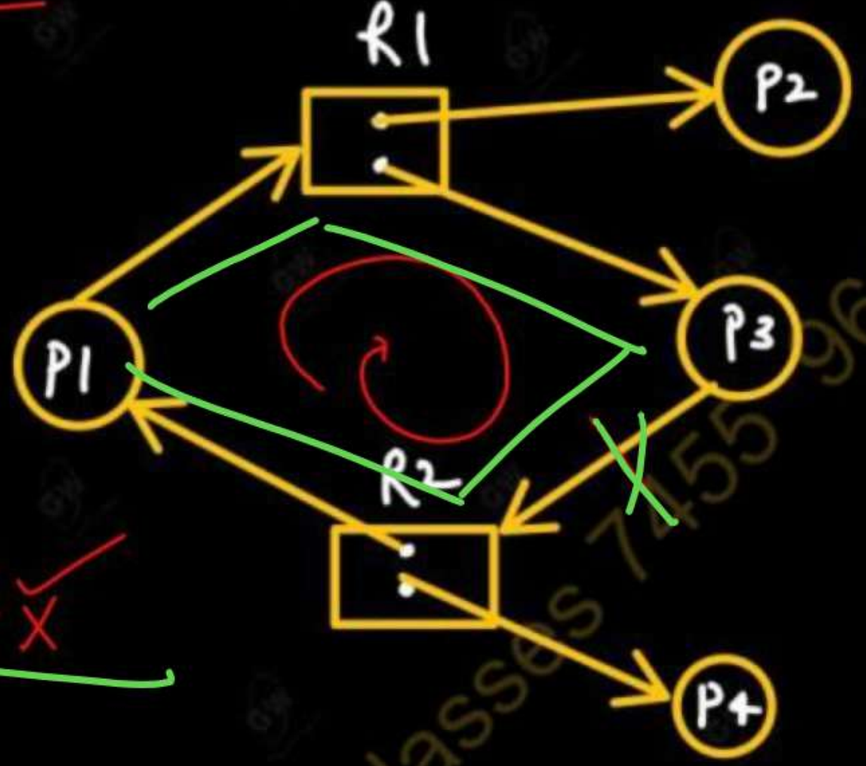
At this time 2 cycles exists in the graph.-

① $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

② $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

- Processes P_1, P_2 and P_3 are deadlocked.
- Process P_2 is waiting for the resource R_3, which is hold by process P_3. Process P_3 is waiting for either process P_1 or process P_2 to release resource R_2. In addition, Process P_1 is waiting for process P_2 to release resource R_1.

❑ **Example 3:- Consider the following RAG –**



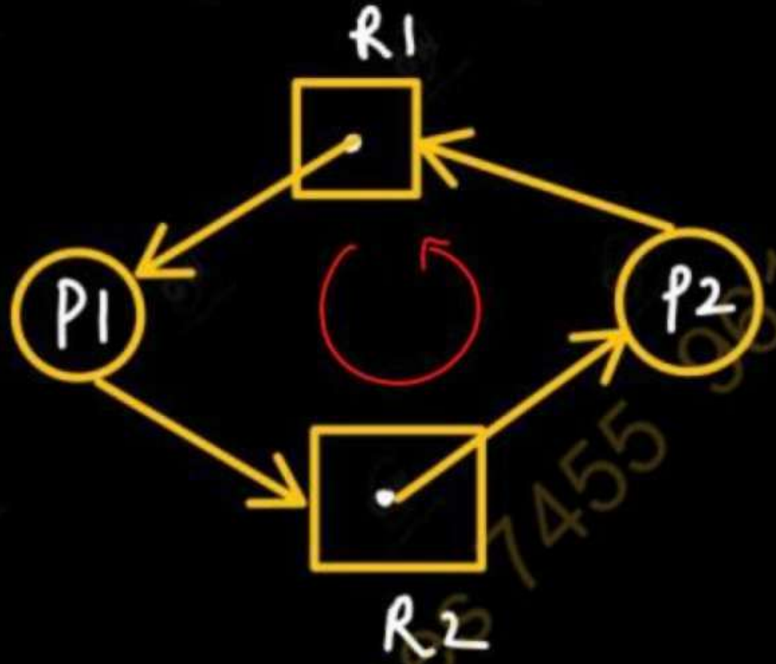
(i) SIR + Cycle = Deadlock ✓

(ii) MIR + Cycle = Deadlock ✗

RAG with a cycle but "no deadlock"

- ❑ In this RAG, there is also a cycle:- P1 → R1 → P3 → R2 → P1
- ❑ However there is no deadlock observe that process P_4 may release its instance of resource type R_2 which can be further allocated to P_3, breaking the cycle.
- ❑ In summary, if a resource – allocation graph does not have a cycle, then the system is not in a deadlocked state. If there is a cycle then the system may or may not be in a deadlocked state.

❑ **Example 4: - Consider the following RAG with single instance of a resource:**



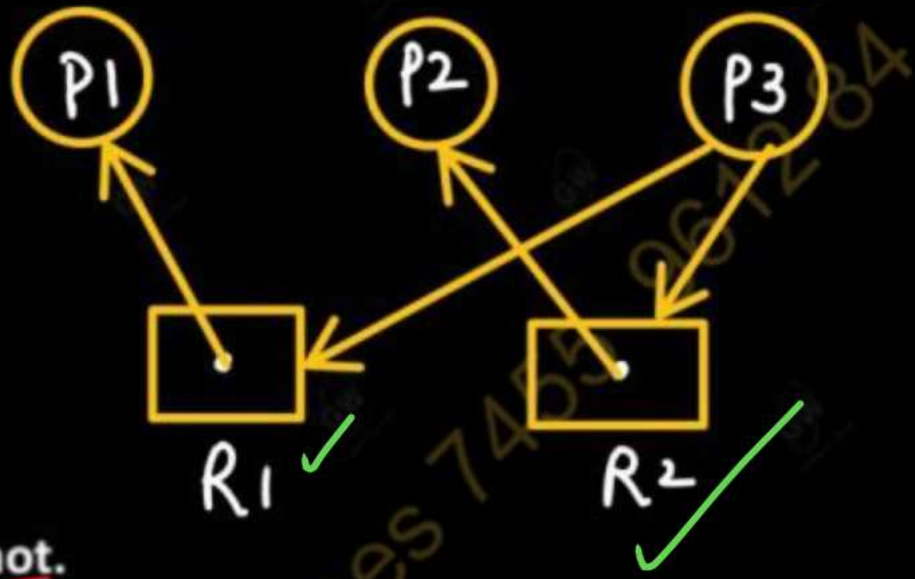
RAG has deadlock or not.

Solution:- Consider the following table:-

Processes	Allocation Matrix		Request Matrix		Availability Matrix	
	R1	R2	R1	R2	R1	R2
P1	1	0	0	1	0	0
P2	0	1	1	0		

- Through the availability (R1,R2) (0,0), no process can be executed i.e. the request of no any process can be fulfilled.
- In this case, there is a deadlocked situation.

❑ Example 5: - Consider the following RAG with single instance of a resource type:



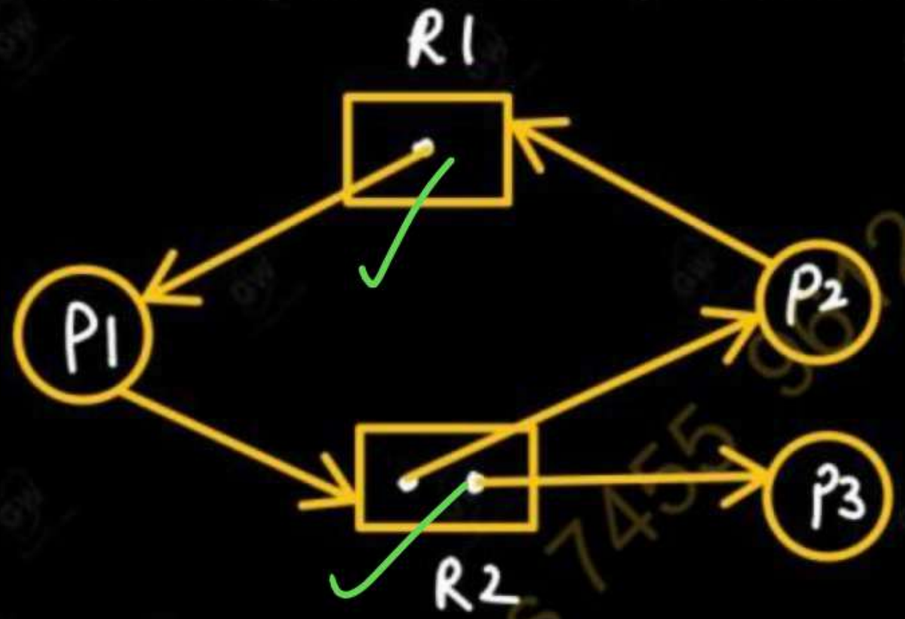
Identify, RAG has deadlock or not.

Solution:- Consider the following table:-

Processes	Allocation Matrix		Request Matrix		Availability Matrix	
	R1	R2	R1	R2	R1	R2
P1	1	0	0	0	0	0
P2	0	1	0	0		
P3	0	0	1	1		

- Through the availability (R1,R2) (0,0), the request of process P1 can be fulfilled. After execution P1 will release (1,0) resources.
- Through the availability (R1,R2) (1,0), the request of process P2 can be fulfilled. After execution P2 will release (0,1) resources.
- Through the current availability (R1,R2) (1,1), the request of process P3 can be fulfilled. After execution P3 will release (0,0) resources.
- In this case, all processes have been executed i.e. no deadlock.

❑ **Example 6: - Consider the following RAG with multiple instances of a resource type:**



Identify, RAG has deadlock or not.

Solution:- Consider the following table:-

Processes	Allocation Matrix		Request Matrix		Availability Matrix	
	R1	R2	R1	R2	R1	R2
P1	1	0	0	1	0	0
P2	0	1	1	0		
P3	0	1	0	0		

- Through the current availability (R1,R2) (0,0), the request of process P3 can be fulfilled. After execution P3 will release (0,1) resource type.
- Through the availability (R1,R2) (0,1), the request of process P1 can be fulfilled. After execution, P1 will release (1,0) resource type.
- Through the current availability (R1,R2) (1,1), the request of process P2 can be fulfilled.
- That is all processes have been executed successfully i.e. no deadlock.
- It is observed that if there is a cycle in the graph with single instances there will always be deadlock.
- But if the RAG with multiple instances of a resource type, deadlock may on may not be observed.

Methods for Handling Deadlocks

- There are various methods of handling the deadlock -
  1. Deadlock Ignorance (ostrich method)
  2. Deadlock Prevention
  3. Deadlock Avoidance
  4. Deadlock Detection & Recovery

1 Deadlock Ignorance

- This method is also known as ostrich method.
- This method says just ignore the deadlock.
- This approach may simplify the design of the operating system and reduce overhead, but it comes with significant risks.
- Deadlocks can occur in real-world systems, and when they do, they can lead to system crashes, hangs, or other undesirable outcomes.
- Ignoring deadlocks altogether can result in unreliable system behavior.
- The ostrich method is generally considered a poor approach to managing deadlocks, as it leaves the system vulnerable to potentially serious problems.
- Modern operating systems typically employ more sophisticated techniques to handle deadlocks effectively and ensure the stability and reliability of the system.

Advantages:

- **Simplicity:** Ignoring the possibility of deadlock can make the design and implementation of the operating system simpler and less complex.

Disadvantages:

- **Unpredictability:** Ignoring deadlock can lead to unpredictable behavior, making the system less reliable and unstable.
- **System Crashes:** If a deadlock does occur and the system is not prepared to handle it, it can cause the entire system to crash, resulting in data loss and other problems.

2

Deadlock Prevention

- Deadlock prevention provides a set of methods for ensuring that at least one of the necessary condition must not hold.
- For a deadlock to occur each of the 4 necessary conditions must hold.
- By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of deadlock.

(i) Mutual Exclusion (i.e. no mutual exclusion) or Eliminate Mutual Exclusion:-

- The mutual exclusion condition must hold for non-sharable resources.
- For example, a printer can not be simultaneously shared by various processes.
- Sharable resources, do no require mutually exclusive access and thus can not be involved in a deadlock.
- Read – only files are a good example of a sharable resource.

- If several processes attempt to open a read only file at the same time, they can be granted simultaneous access to the file.
- A process never needs to wait for a sharable resource.
- It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

(ii) Hold & Wait: - (i.e. no hold and wait)

- To ensure that hold-and-wait condition never occurs in the system we must guarantee that, whenever a process requests a resource, it does not hold any other resources.
- Therefore, one protocol that can be used requires each process to request and be allocated all its resources before it begins execution.
- Another way to allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization.
- For example, if a process requires a printer at a later time and we have allocated a printer before the start of its execution printer will remain blocked till it has completed its execution.
- The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.
- An alternative protocol allows a process to request resources only when it has none.
- A process may request some resources and use them. Before if can request any additional resources, however, it must release all the resources that it is currently allocated.

(iii) No Preemption: - (i.e. preemption is allowed)

no no-preemption

- The third necessary condition for deadlock is that there be no preemption of resources that have already been allocated.
- Eliminate No Preemption i.e. preempt resources from the process when resources are required by other high-priority processes.
- To ensure that No-Preemption condition does not hold, we can use the following protocol.
- If a process is holding some resources and requests another resource that can not be immediately allocated to it i.e. the process must wait, then all resources the process is currently holding are preempted.
- In other words, these resources are implicitly released.
- The preempted resources are added to the list of resources for which the process is waiting.
- The process will be restarted only when it can regain its old resources as well as the new ones that it is requesting.

- Alternatively, if a process requests some resources, we first check whether they are available.
- If they are, we allocate them. It they are not we check whether they are allocated to some other process that is waiting for additional resources.
- If so, we preempt the desired resources form the waiting process and allocate them to the requesting process.
- If the resources are neither available nor hold by a waiting process, the requesting process must wait.
- While it is waiting, some of its resources may be preempted, but only if another process requests them.
- A process can be restarted only when it is allocated the new resources it is requesting and recovers any resources that were preempted while it was waiting.

(iv) Circular Wait: - (i.e. no circular wait)

- The fourth & final condition for deadlocks is the circular wait condition.
- One way to ensure that this condition never holds is to impose a total ordering of all resource types and to require that each process requests resources in an increasing order of enumeration.
- The logic is just assign an integer number to each resources and after assigning a number, make sure that the process can request the resources either in increasing order only or in decreasing order only i.e. strictly increasing or strictly decreasing order.
- For example, suppose we have the following four resources with some integer value:-

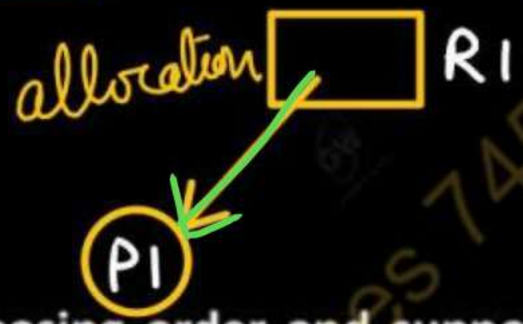
  1 – Printer – R1

  2 – Scanner – R2

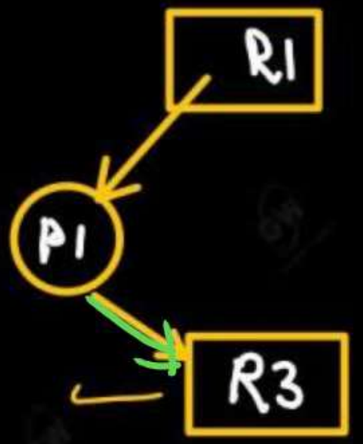
  3 – CPU – R3

  4 – Register – R4

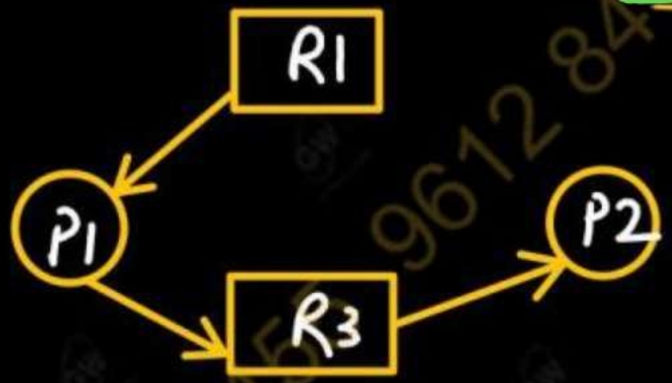
- The condition is whenever a process is requesting for a resource, it will request strictly in the increasing order.
- For example, suppose Process P1 comes and request for Resource 1 i.e. Printer -



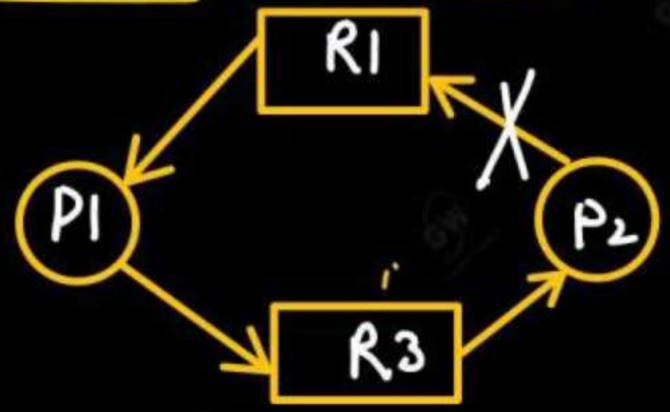
- Now it can request in increasing order and suppose Process P1 request for Resource 3 i.e. CPU. Then there is no problem, it can request for R3 -



- Now suppose other Process P2 come and P2 request for CPU i.e. Resource R3 then there is no problem, it can request for R3 -



- Now suppose P2 request for the resource number lesser than 3 that is not allowed i.e. a process can not request for the resource number less than the previous request, now suppose P2 request for Resource 1 i.e. Printer, it can not request, it is not allowed.



- If this request can not be made we can say that no circular wait.
- Say process P2 can request for resource R4, yes it is possible.
- P2 has requested for resource number 3, then it can request for any other resource which is having the number greater then 3.

Deadlock Avoidance

- During allocation of resources to processes, it is to be checked that whether it is safe state or not i.e. every at every stage of resource allocation check for safe and unsafe state.
- This is done through the Banker's algorithm. It was developed by Edsger Dijkstra.
- Banker's Algorithm is also known as Safety Algorithm.
- Banker's Algorithm is a resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, and after granting a request system remains in the safe state it allows the request, and if there is no safe state it doesn't allow the request made by the process.
- When resources having multiple instances than it is better to use Banker's algorithm.

- Why this algorithm is known as Banker's algorithm? Because this algorithm could be used in Baking system so that banks never run out of resource (money) and always be in safe state.
- The Banker's Algorithm ensures that resources are allocated in such a way that the system will always remain in a safe state, meaning that deadlock will not occur.
- This is achieved by only granting resource requests that do not lead to unsafe situations, where a process can finish and release its resources, allowing other processes to proceed without deadlock.
- By using the Banker's Algorithm, the operating system ensures efficient and deadlock-free resource allocation, contributing to system stability and reliability.

- For avoiding deadlock through /Banker's algorithm some additional information is required:

1. How many instances of each resource are available, each process can request maximum, request [MAX] i.e. maximum request of a process (2-D array)
2. How many instances of each resource each process currently holds? [Allocation]. (2-D array) i.e. allocation of resources
3. How many instances of each resource is available in the system? [Available]

Example 1 : Consider the following five processes P0, P1, P2 ,P3 and P4 with ALLOCATION, MAX and AVAILABLE matrices in the following table:

Process ID	ALLOCATION				MAX				AVAILABLE			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

Need

5x4 5x4 5x4

(i) Identify the NEED matrix.

(ii) Is system in Safe state?

(iii) If system is in safe state, identify the safe sequence.

Solution:-

Process ID	ALLOCATION (RESOURCES 1 2 3 4)				MAX				AVAILABLE				NEED (MAX - ALLOCATION)				flag
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	
P0	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0	1
P1	1	0	0	0	1	7	5	0	+0	0	1	2	1	7	5	0	1
P2	1	3	5	4	2	3	5	6	1	5	3	2	1	0	0	2	1
P3	0	6	3	2	0	6	5	2	+1	3	5	4	0	0	2	0	1
P4	0	0	1	4	0	6	5	6	2	8	8	6	0	6	4	2	1
									+0	6	3	2					
									2	14	11	8					

Total Resources available

+0 0 1 4

2 14 12 12

+1 0 0 0

3 14 12 12

Step 1: Need of process P0 and P3 can be fulfilled.

Step 2: Need of P0 will be fulfilled and after execution it will release (0, 0, 1, 2) resources that will be added to AVAILABLE.

Step 3: Now need of process P2 and P3 can be fulfilled. Fulfill the need of P2 and after execution it will release (1, 3, 5, 4) that will be added to AVAILABLE.

Step 4: Now need of process P3 and P4 can be fulfilled. Fulfill the need of P3 and after execution it will release (0, 6, 3, 2) that will be added to AVAILABLE.

Step 5: Now the remaining processes are P1 and P4. The need of both can be fulfilled. Fulfill the need of P4 and after execution it will release (0, 1, 1, 4) that will be added to AVAILABLE.

Step 6: Now the remaining process is P1. The need of P1 can be fulfilled. Fulfill the need of P1 and after execution it will release (1, 0, 0, 0) that will be added to AVAILABLE.

The safe sequence is - P0, P2, P3, P4, P1.

The system is in safe state.

Banker's Algorithm

Input : Processes

n : Number of Processes

m : Number of Resources

Step 1 : flag [i] = 0 for i = 0 to (n – 1) &

Find NEED [n] [m] = MAX [n] [m] – ALLOCATION [n] [m]

Step 2: Find a process P_i such that –

flag [i] = 0 & $NEED_i$ < = AVAILABLE

Step 3: If such i (process) exists then

flag [i] = 1,

AVAILABLE = AVAILABLE + $ALLOCATION_i$

go to step 2

Otherwise go to step 4

Step 4: If flag[i] = 1 for all i (processes) then

system is in safe state

otherwise

system is unsafe state.

Step 5: Stop

Example 2 : Consider the following five processes P1, P2 ,P3, P4 and P5 with ALLOCATION and MAX matrices in the following table:

Process ID	ALLOCATION			MAX		
	A	B	C	A	B	C
P1	0	1	0	7	5	3
P2	2	0	0	3	2	2
P3	3	0	2	9	0	2
P4	2	1	1	4	2	2
P5	0	0	2	5	3	3

Total Instances, A = 10, B= 5 and C = 7.

(i) Identify the NEED matrix.

(ii) Is system in Safe state?

(iii) If system is in safe state, identify the safe sequence.

Solution:- Total Instances, A = 10, B= 5 and C = 7.

Process ID	ALLOCATION			MAX			AVAILABLE			NEED = MAX - ALLOCATION			FLAG
	A	B	C	A	B	C	A	B	C	A	B	C	
P1	0	1	0	7	5	3	3	3	2	7	4	3	0
P2	2	0	0	3	2	2	5	3	2	1	2	2	0
P3	3	0	2	9	0	2	7	4	3	6	0	2	0
P4	2	1	1	4	2	2	7	4	5	2	1	1	0
P5	0	0	2	5	3	3	7	5	5	5	3	1	0
Total	7	2	5				10	5	7				

Step 1: Through the current availability of resources, the Need of processes P2 and P4 can be fulfilled. Need of P2 will be fulfilled and after execution it will release (2, 0, 0) resources that will be added to the AVAILABLE.

Step 2: Through the current availability of resources, the Need of process P4 can be fulfilled. Fulfill the need of P4 and after execution it will release (2, 1, 1) that will be added to AVAILABLE.

Step 3: Through the current availability of resources, the Need of all remaining processes P5, P1 and P3 can be fulfilled. Fulfill the need of P5 and after execution it will release (0, 0, 2) that will be added to AVAILABLE.

Step 4: Fulfill the need of P1 and after execution it will release (0, 1, 0) that will be added to AVAILABLE.

Step 5: Now the remaining process is P3. The need of P3 can be fulfilled. Fulfill the need of P3 and after execution it will release (3, 0, 2) that will be added to AVAILABLE.

The safe sequence is - **P2, P4, P5, P1, P3**

The system is in safe state i.e. no deadlock is observed in the system.

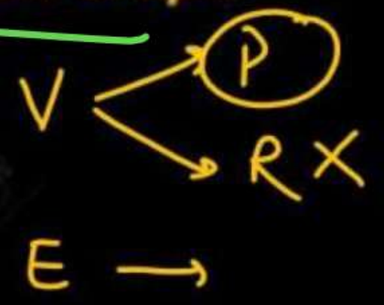
Deadlock Detection & Recovery

- Deadlock detection and recovery are techniques used in operating systems to handle situations where multiple processes are unable to proceed because each is waiting for a resource held by another process.
- These are the algorithms which basically examines the state of the system at the particular point of time and then determines whether deadlock has arrived or not.
- If deadlock situation is there, finally some recovery techniques would be applied to recover from that situation.
- There are two types of deadlock detection algorithm:

1. If the system is having single instance of resources - In this environment the algorithm used is wait for graph (enhanced version of RAG)
2. System is having multiple ~~cretonnes~~ instances of the resources - In this environment Banker's algorithm is used to detect deadlock.

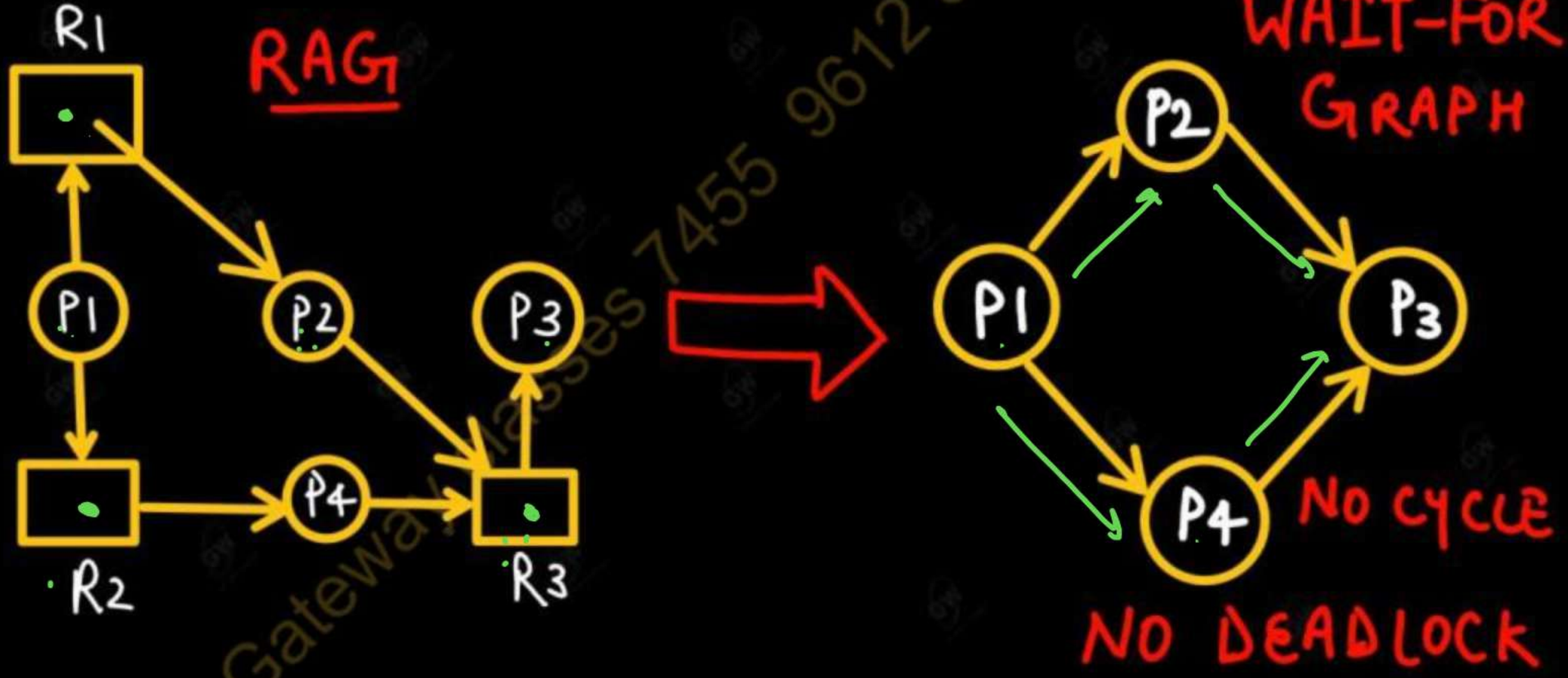
Difference between Wait-For Graph & Resource Allocation Graph

- Wait-for graph is an enhanced version of RAG.
- Remove resources from RAG then finally it becomes wait-for graph.
- In wait-for graph we have only processes (as vertices of graph).
- In wait-for graph (single instance) just detect cycle, if cycle is there in wait-for graph then we can say the system is in deadlocked state.
- If multiple instances of resources are there and cycle in RAG, then deadlock may or may not exist.

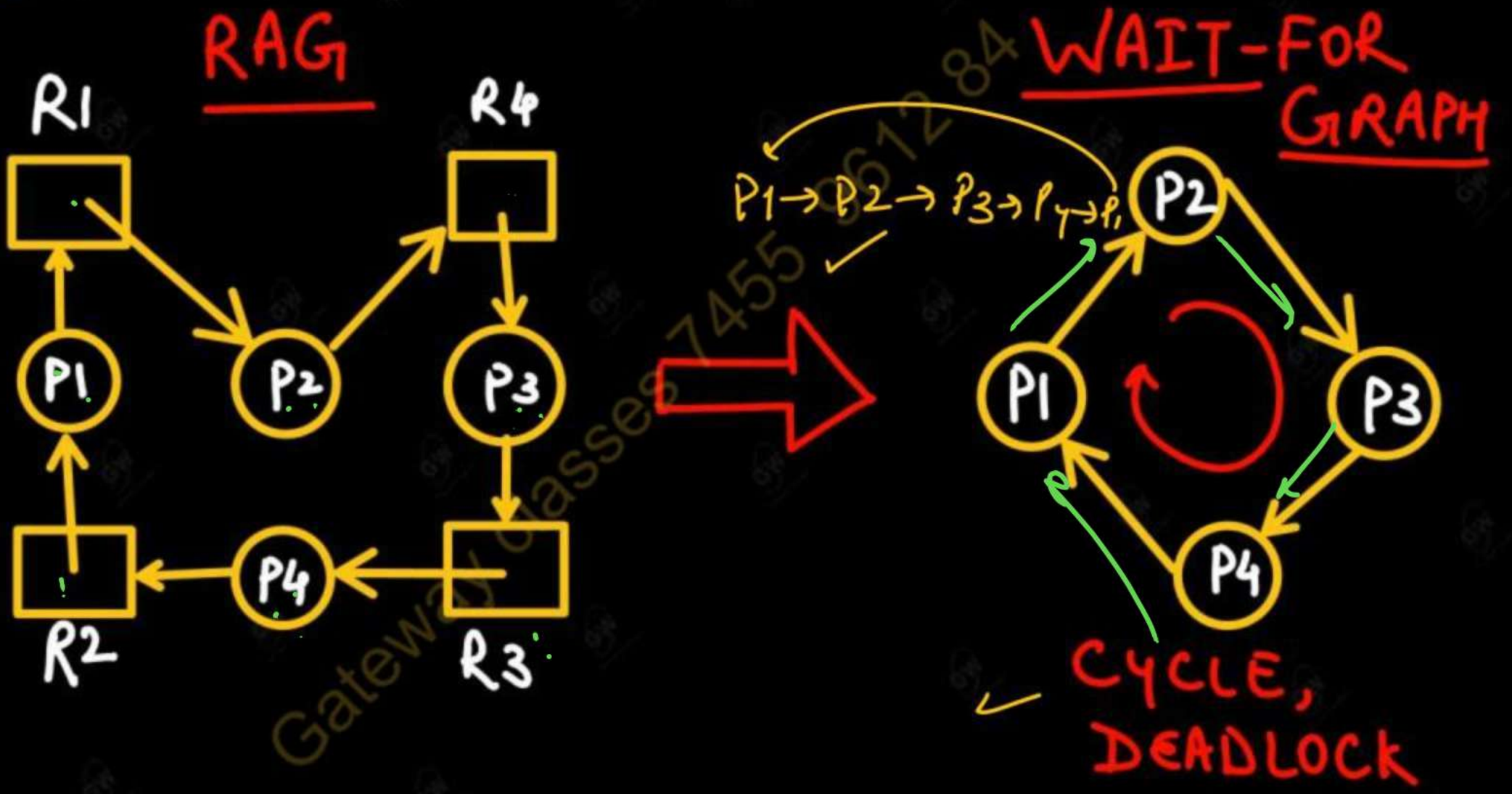


Conversion of Resource Allocation Graph to Wait-for Graph

Example 1:- Convert the following RAG to Wait for Graph:



Example 2:- Convert the following RAG to Wait-for Graph:



Deadlock Recovery

- There are two approaches of deadlock recovery.

  (1) Pessimistic Approach (Process Termination)

  (2) Optimistic Approach (Preemption of resources & processes)

(1) Pessimistic Approach:-

- Process termination or killing of processes takes place.

Two methods:-

(a) Abort all Deadlocked Processes : -

- It may be costly. Suppose we have 5 processes and executed in the manner:

  P1 - 90%

  P2 - 90%

  P3 - 10%

  P4 - 10%

  P5 - 20%

- If all the processes from P1 to P5 will be aborted, P1 & P2 has to start its execution again even when they have been executed 90%.
- Therefore, it is not a good idea to abort all the deadlocked processes.

(b) Abort one process at a time and decide next to abort after deadlock detection:

- The idea is, we should terminate that process whose termination will cause minimum overhead and minimum cost to the system.
- **Selection of Process:**

- There are some factors on the basis of which a process can be selected –
  1. Priority of process (low priority process may be selected).
  2. For how long the process has completed? If any process has completed 90% of its execution then it is not better to abort that process.

3. How many and what type of resources, process has used?

4. Whether a process is interactive or batch process.

5. How many resources the process needs to complete its execution?

Drawback:

- ❑ Detecting and invoking deadlock detection algorithm again and again, it is total overhead.

(2) Optimistic Approach of Deadlock:-

- Preemption of resources and processes i.e. preempt some resources from processes and give these resources to another processes until the deadlock is broken.
- It needs to decide that from which process, the resource will be taken off.
- When system is implementing this kind of approach of deadlock recovery "preemption of resources & processes" then basically it needs to handle 3 problems: -
  - (1) Selecting a victim
  - (2) Rollback
  - (3) Starvation

(1) Selecting a victim

- Means which resource or process is to be preempted. (on the basis of some specific criteria & minimum cost).

(2) Rollback

- When we select a process and preempt resources from that process, then obviously that process cannot continue its normal execution.
- Then that process has to be roll backed to previous safe state and that process will start its execution from that safe state only.
- In this case the system has to maintain the information of state of all the running processes.
- Total Roll Back:- Abort the process and restart. (the above way is better than total rollback).

(3) Starvation

- Selecting a process and preempting resources form that process.
- But if again the same process is selected for preemption then it may cause starvation problem i.e. that process will never be able to complete its execution.
- It must be ensured that same process must not be selected again and again.
- A process must be selected as a victim for some finite number of times.
- Number of rollbacks can be computed in the system to ensure the same.

AKTU PYQs

Q.1. What is the need for Process Control Block (PCB)? 2015-16, 2 marks

Q.2. Draw process state transition diagram. 2015-16, 2 marks

Q.3. Difference between Process and Program. 2016-17, 2 marks

Q.4. What do you understand by Process? Explain various states of process with suitable diagram. Explain process control block. 2016-17, 10 marks

Q.5. Describe the typical elements of the process control block. 2018-19, 2 marks

Q.6. Define Process. Explain various steps involved in change of a process state with neat transition diagram. 2018-19, 7 marks

Q.7. What is process control block? 2018-19, 2 marks

Q.8. Explain in detail about the Process control Block (PCB) in CPU scheduling. 2021-22, 10 marks

Q.9. Define process and process control block. Also, describe process state transition diagram in detail. 2022-23, 10 marks

Q.10. Explain threads. 2016-17, 2 marks

AKTU PYQs

Q. 1. Explain the long term, mid term and short term schedulers. Describe the differences among them.

2014-15, 2017-18, 10 marks

AKTU PYQs

Q.1. Discuss the performance criteria for CPU Scheduling.

2015-16, 10 Marks, 2017-18, 5 Marks, 2021-22, 2 Marks

Q.2. What are the various scheduling criteria for CPU scheduling? 2017-18, 2018-19, 2 Marks

AKTU PYQs

Q.1. Consider the following processes:

Process	Arrival time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Draw Gantt chart and find the average waiting time and average turnaround time:

(i) FCFS Scheduling (ii) SRTF Scheduling 2017-18, 7 Marks

Q.2. Consider the following process:

Process	Arrival time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

What is the average waiting and turn around time for these process with

(i) FCFS Scheduling (ii) Preemptive SJF Scheduling 2018-19, 7 Marks

Q.3. Explain the various CPU scheduling techniques with Gantt charts clearly as indicated by (process name, arrival time, process time) for the following (A,0,4), (B,2,7), (C,3,2) and (D,2,2) for FCFS and SRTF.

2018-19, 10 marks

Process Name	AT	CPU Burst time
A	0	4
B	2	7
C	3	2
D	2	2

TAT
ATAT
WT
AWT

AKTU PYQs

Q.1. Consider the processes, CPU burst time and Arrival time given below:

Processes	CPU burst time	Arrival time
P1	8	0
P2	4	1
P3	9	2
P4	5	3

Draw the Gantt chart and calculate the following by using SRTF CPU Scheduling Algorithm,

(i) Average Waiting Time,

(ii) Average Turn Around time.

2015-16, 10 Marks

Q.2. Consider the following process:

Process	Arrival time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Draw Gantt chart and find the average waiting time and average turnaround time:

(i) FCFS Scheduling (ii) SRTF Scheduling 2017-18, 7 Marks

Q.3. Consider the following process:

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

Draw Gantt chart and find the average waiting time and average turnaround time:

(i) SRTF Scheduling (ii) Round robin (time quantum:3) 2017-18, 7 Marks

Q.4. For the following data find the average Turnaround time and average waiting time for:

(i) Preemptive SJF (ii) Round-Robin (Time quantum:3) 2017-18, 10 marks

SRTN

Job	Arrival time	Burst Time
1	0	10
2	1	3
3	3	2
4	4	4

Q.5. Consider the following process:

Process	Arrival time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

What is the average waiting and turn around time for these process with -

(i) FCFS Scheduling (ii) Preemptive SJF Scheduling 2018-19, 7 marks

Q.6. Consider the set of processes given in the table and the following scheduling algorithms:

(i) Round Robin (Quantum 2)

(ii) SJF

Non Preemptive

Process ID	Arrival Time	Execution Time
A	0	4
B	2	7
C	3	3
D	3.5	3
E	4	5

Draw the Gantt chart and find the average waiting time and turnaround time for the algorithms.

2018-19, 10 Marks

Q.7. Explain the various CPU scheduling techniques with Gantt charts clearly as indicated by (process name, arrival time, process time) for the following (A,0,4), (B,2,7), (C,3,2) and (D.2.2) for FCFS and SRTF.

2021-22, 10 Marks

AKTU PYQs

Q.1. For the following processes, draw Gantt chart to illustrate the execution using:

(i) Preemptive priority scheduling (ii) Non-Preemptive priority scheduling.

Also, Calculate average waiting time and average turnaround time.

(Assumption: A larger priority number has higher priority.)

Process	Arrival Time	Burst Time	Priority
A	0	5	4
B	2	4	2
C	2	2	6
D	4	4	3

2022-23, 10 Marks

AKTU PYQs

Q.1. Consider the following process:

Process Id	Arrival Time	CPU Burst Time	Priority
P1	0	6	3
P2	1	4	1
P3	2	5	2
P4	3	8	4

Draw Gantt chart and find the average waiting time and average turnaround time:

(i) SRTF Scheduling (SRTN (Preemptive SJF)) (ii) Round Robin (time quantum:3) 2017-18, 7 Marks

Q.2. For the following data find the average Turnaround time and average waiting time for:

(i) Preemptive SJF (SRTN / SRTF) (ii) Round-Robin (time quantum : 3)

Job	Arrival time	Burst Time
1	0	10
2	1	3
3	3	2
4	4	4

2018-19, 10 Marks

AKTU PYQs

Q.1. What is the difference between preemptive and non preemptive? State, why strict non preemptive scheduling is unlikely to be used in computer center? Explain the operation of multilevel scheduling. 2014-15, 10 Marks

Q.2. Explain the following scheduling algorithms:

(i) Multilevel feedback queue scheduling

(ii) Multiprocessor Scheduling 2014-15, 10 Marks

Q.3. Define the multilevel feedback queues scheduling. 2015-16, 2 Marks

Q.4. What is the advantage of having different time quantum size on different levels of a multilevel queuing system? 2015-16, 2 Marks

Q.5. Explain the following terms:

(i) Multilevel feedback Queue Scheduling

(ii) Fixed portioning vs. variable partitioning 2022-23, 10 Marks

AKTU PYQs

Q.1. What is a deadlock? Discuss the necessary conditions for deadlock with examples.

2016-17, 7.5 Marks, 2018-19, 10 Marks

Q.2. Write the condition for deadlock. Explain the protocol to be used to break the circular wait condition.

2017-18, 10 Marks

Q.3. Differentiate between Deadlock and Starvation in detail. 2018-19, 2 Marks

Q.4. Explain in detail about the Deadlock System model and Deadlock characterization.

2021-22, 10 Marks

Q.5. Explain starvation problem and its solution. 2022-23, 2 Marks

Q.6. Explain Deadlock and necessary conditions for deadlock. Also, Explain resource allocation graph with suitable diagram. 2022-23, 10 Marks

AKTU PYQs

Q.1. Describe Resource Request Algorithm. 2017-18, 2 marks

Q.2. Explain Deadlock and necessary conditions for deadlock. Also, Explain resource allocation graph with suitable diagram. 2022-23, 10 marks

AKTU PYQs

Q.1. Write the condition for deadlock. Explain the protocol to be used to break the circular wait condition.

2017-18, 10 Marks

AKTU PYQs

Q.1. Describe Banker's algorithm for deadlock avoidance. Consider a system with three process and three resources. The snapshot of a system at time t_0 is given below:

PROCESSES	ALLOCATION			MAX			AVAILABLE		
	A	B	C	A	B	C	A	B	C
P_0	2	2	3	3	6	8	7	7	10
P_1	2	0	3	4	3	3			
P_2	1	2	4	3	4	4			

(i) Is the current allocation in safe state?

(ii) Would the following requests be granted in the current state?

(a) Process P_2 requests (1,0,0)

(b) Process P_1 requests (1,0,0)

2014-15, 10 Marks

Q.2. Consider the following snapshot of the system:-

	Allocation	Max	Available
	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	1 5 2 0
P1	1 0 0 0	1 7 5 0	
P2	1 3 5 4	2 3 5 6	
P3	0 6 3 2	0 6 5 2	
P4	0 0 1 4	0 6 5 6	

Answer the following questions using the banker's algorithm:-

(i) What is the content of the matrix Need?

(ii) Is the system in a safe state? If yes then find the Safe sequence.

(iii) If a request from process PI arrives for (0, 4, 2, 0) can the request be granted immediately?

2015-16, 10 Marks

Q.3. Describe Banker's algorithm for safe allocation. 2016-17, 7.5 Marks

Q.4. Consider the following snapshot of a system:

	ALLOCATION			MAX			AVAILABLE		
PROCESSES	R1	R2	R3	R1	R2	R3	R1	R2	R3
$P1$	2	2	3	3	6	8	7	7	10
$P2$	2	0	3	4	3	3			
$P3$	1	2	4	3	4	4			

Answer the following questions using the banker's algorithm:

1. What is the content of matrix need?
2. Is the system in a safe state? 2017-18, 7 Marks

Q.5. Consider the following snapshot of a system.

PROCESSES	ALLOCATION				MAX				AVAILABLE			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Answer the following using banker's algorithm.

(i) Find the Need Matrix

(ii) Is the system in safe state

(iii) (iii) If the request from P1 for <0,4,2,0>, can the request be granted immediately.

2017-18, 10 Marks

Q.6. What is a safe state and an unsafe state? 2018-19, 2 Marks

Q.7. Consider the following snapshot of a system:

	ALLOCATION			MAX			AVAILABLE		
PROCESSES	R1	R2	R3	R1	R2	R3	R1	R2	R3
$P1$	2	2	3	3	6	8	7	7	10
$P2$	2	0	3	4	3	3			
$P3$	1	2	4	3	4	4			

Answer the following questions using the banker's algorithm:

(i) What is the content of the matrix need? (ii) Is the system in a safe state?

2018-19, 7 Marks

Q.8. Write and explain Banker's algorithm for avoidance of deadlock. 2018-19, 10 Marks

Q.9. What do you mean by the safe state and an unsafe state? 2021-22, 2 marks

Q.10. Consider the given snapshot of a system with five processes (PO,P1, P2, P3,P4) and three resources (A,B,C).

	Allocation	Max	Available
Process/Resource	A B C	A B C	A B C
P0	1 1 2	4 3 3	2 1 0
P1	2 1 2	3 2 2	
P2	4 0 1	9 0 2	
P3	0 2 0	7 5 3	
P4	1 1 2	11 2 3	

(i) Calculate the content of Need Matrix.

(ii) Apply safety algorithm and check the current system is in safe state or not.

(iii) If the request from process P1 arrives for (1, 1, 0), can the request be granted immediately?

2022-23, 10 Marks

AKTU PYQs

Q.1. Explain Deadlock Recovery.

2017-18, 2 Marks