

## CO - 2

**COURSE NAME : SYSTEM DESIGN AND INTRODUCTION TO CLOUD**

**COURSE CODE : 23AD2103A**

---

**TOPICS :** DEADLOCK: PREVENTION, DETECTION AND AVOIDANCE  
PERSISTENCE

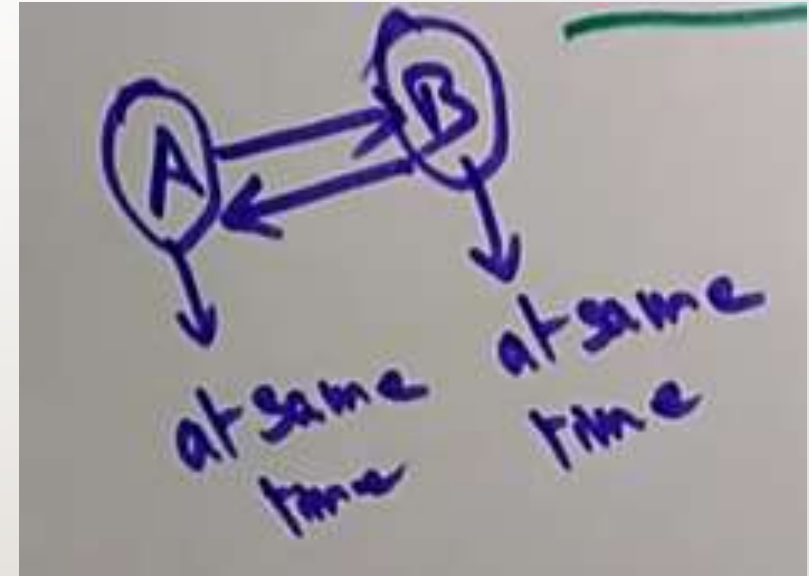
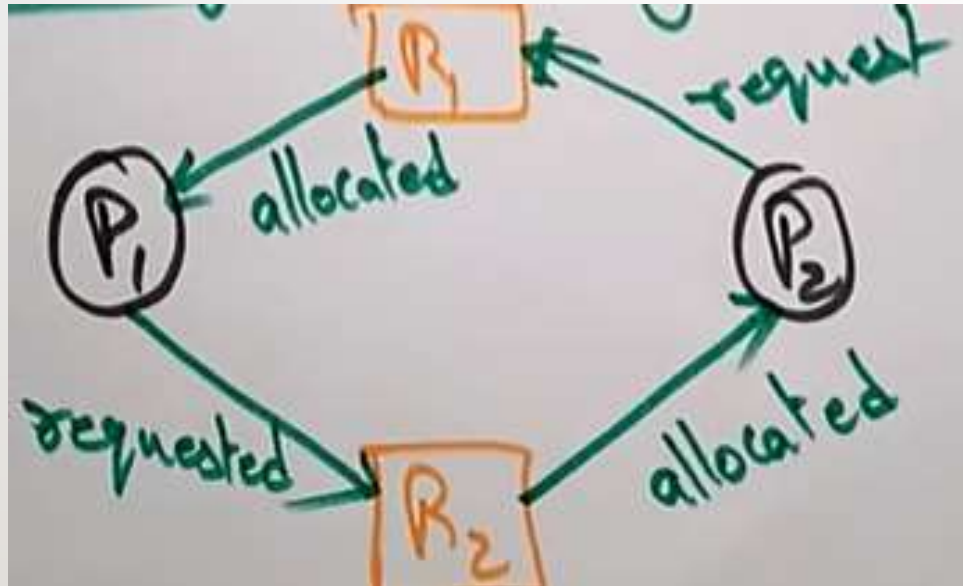
# SESSION DESCRIPTION

- Deadlock
  - Prevention
  - Detection
  - Avoidance
  - Persistence

# WHAT IS DEADLOCK?

- Deadlock is a situation in computing where two or more processes are unable to proceed because each is waiting for the other to release resources. Key concepts include mutual exclusion, resource holding, circular wait, and no preemption.
- OR
- A deadlock is a situation where a set of processes is blocked because each process is holding a resource and waiting for another resource acquired by some other process.

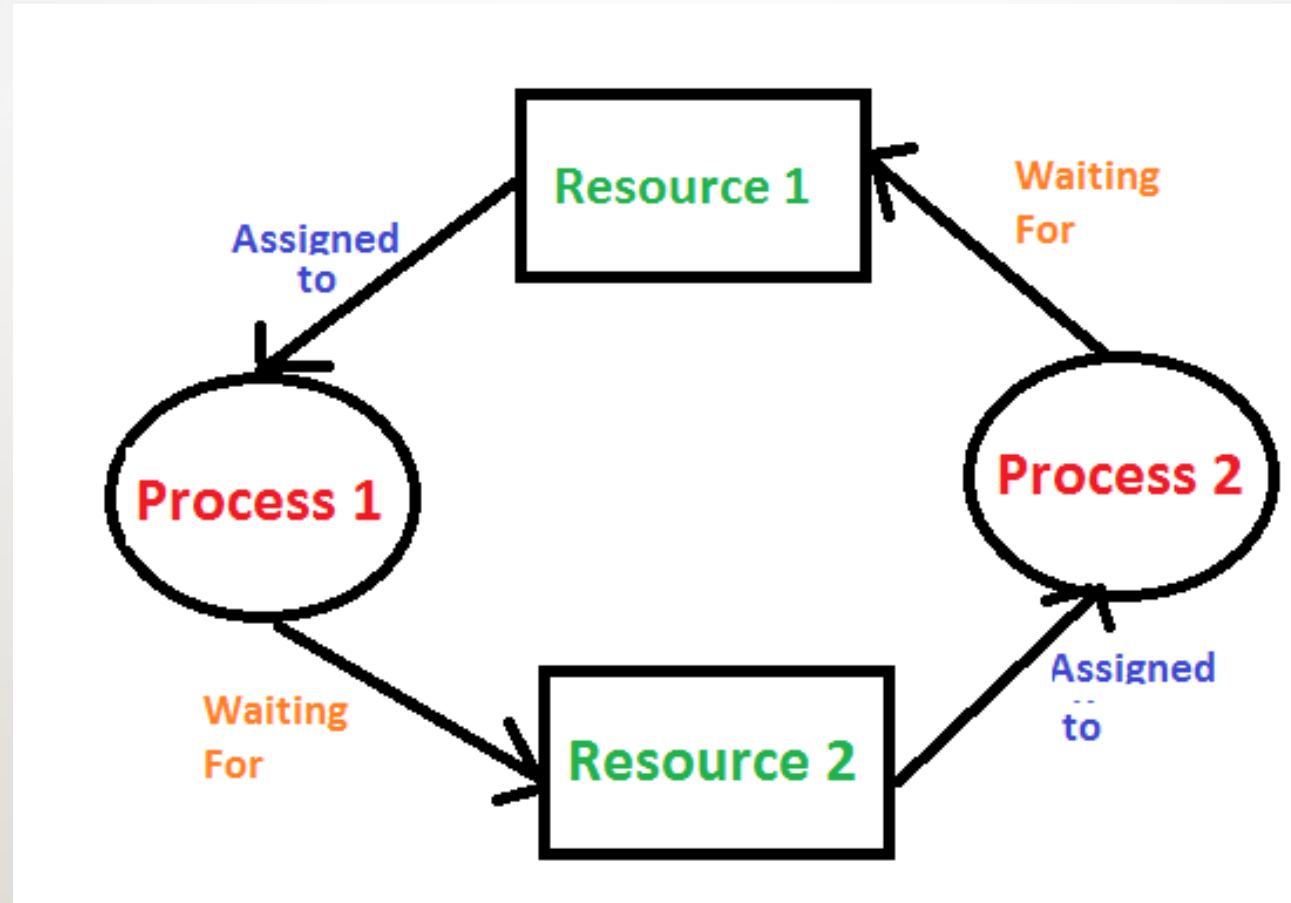
# DEADLOCK



# HOW DOES DEADLOCK OCCUR IN THE OPERATING SYSTEM?

- A process in an operating system uses resources in the following way.
- Requests a resource
- Use the resource
- Releases the resource
- A situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s).

# HOW DOES DEADLOCK OCCUR IN THE OPERATING SYSTEM?



# NECESSARY CONDITIONS FOR DEADLOCK IN OS

- Deadlock can arise if the following four conditions hold simultaneously (Necessary Conditions)
- **Mutual Exclusion:** Two or more resources are non-shareable (Only one process can use at a time).
- **Hold and Wait:** A process is holding at least one resource and waiting for resources.
- **No Preemption:** A resource cannot be taken from a process unless the process releases the resource.
- **Circular Wait:** A set of processes waiting for each other in circular form.

# DEADLOCK CONDITIONS

- **Mutual exclusion** – Not sharable – Resources cannot be sharable – Actually Only one process can use resource at a time.
- **Hold and Wait** – A process is holding one resources and waiting for another.
- **No preemption** – A resources can be released voluntarily by the process itself.
- **Circular Wait** –
- **Example** : 4 process A B C D
- A acquired some resources which is needed by B ; B acquired some resources which is needed by C ; C acquired some resources which is needed by D ; D acquired some resources which is needed by A.



# WHAT ARE THE METHODS FOR HANDLING DEADLOCK?

- There are three ways to handle deadlock
  - Deadlock Prevention or Avoidance
  - Deadlock Recovery
  - Deadlock Ignorance

# VARIOUS METHODS TO HANDLE DEADLOCK



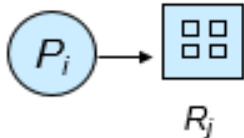
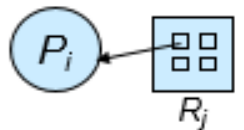
- **DEADLOCK IGNORANCE** : Better to ignore the deadlock, it will mostly done by the os. They will ignore the deadlock. Whenever deadlock occurs, system will hang. Solution is restarting the system. Suppose your os is having full of code, deadlock will occur. Decreasing system performance.
- **DEADLOCK PREVENTION** : Try to find solution before deadlock occurs.
- It says that try to remove or make false all the 4 conditions.
  - DEADLOCK AVOIDANCE
  - DEADLOCK DETECTION & RECOVERY

# WHAT IS DEADLOCK DETECTION?

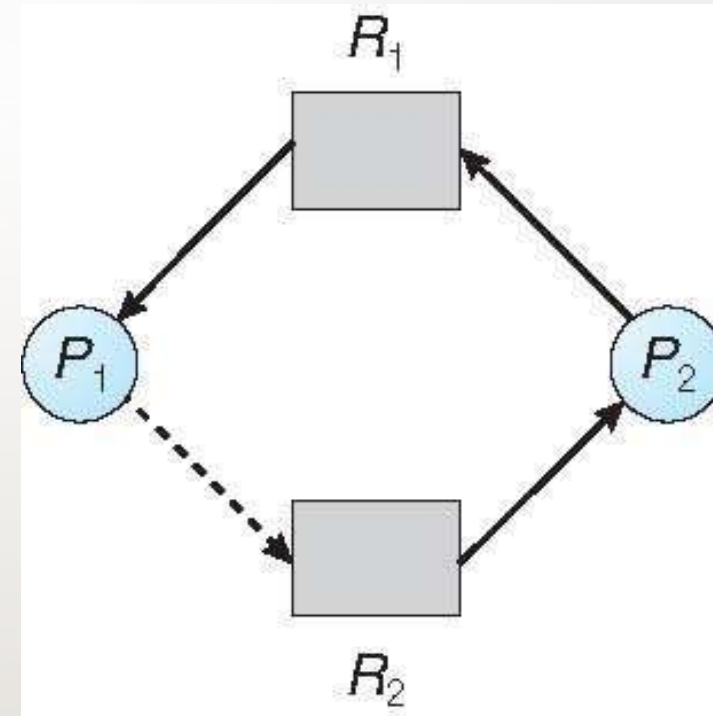
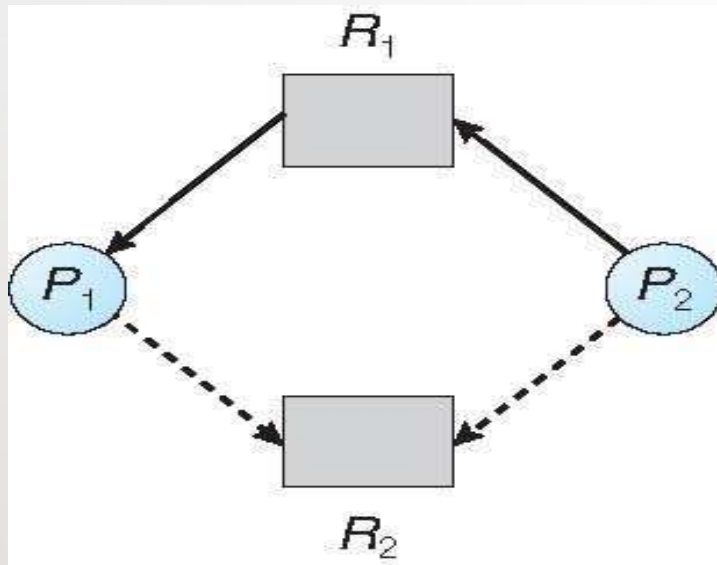
- Deadlock detection is a process in computing where the system checks if there are any sets of processes that are stuck waiting for each other indefinitely, preventing them from moving forward.
- Deadlock detection is the process of finding out whether any process are stuck in loop or not. There are several algorithms like
  - Resource Allocation Graph
  - Banker's Algorithm
  - These algorithms helps in detection of deadlock in Operating System.

# RESOURCE ALLOCATION GRAPH (RAG)

- A resource allocation graphs shows which resource is held by which process and which process is waiting for a resource of a specific kind.
- Resource allocation graph describe what the condition of the system as far as process and resources are concern like what number of resources are allocated and what is the request of each process. Everything can be represented in terms of graph.

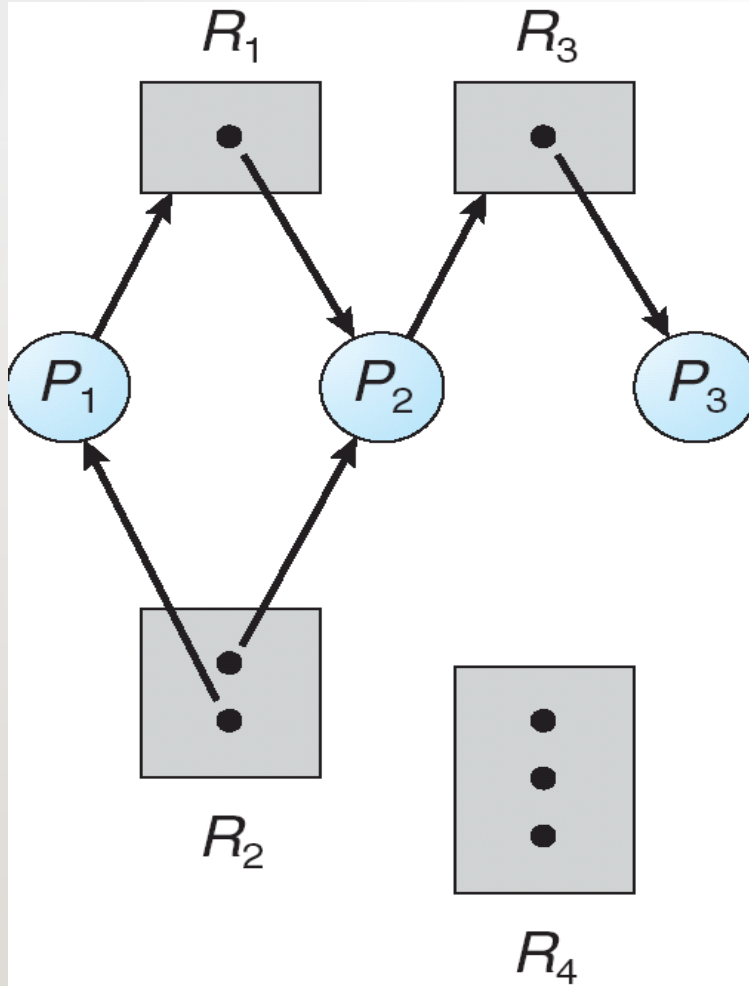
- Process 
- Resource Type with 4 instances 
- $P_i$  requests instance of  $R_j$   

- $P_i$  is holding an instance of  $R_j$   


# RESOURCE-ALLOCATION GRAPH

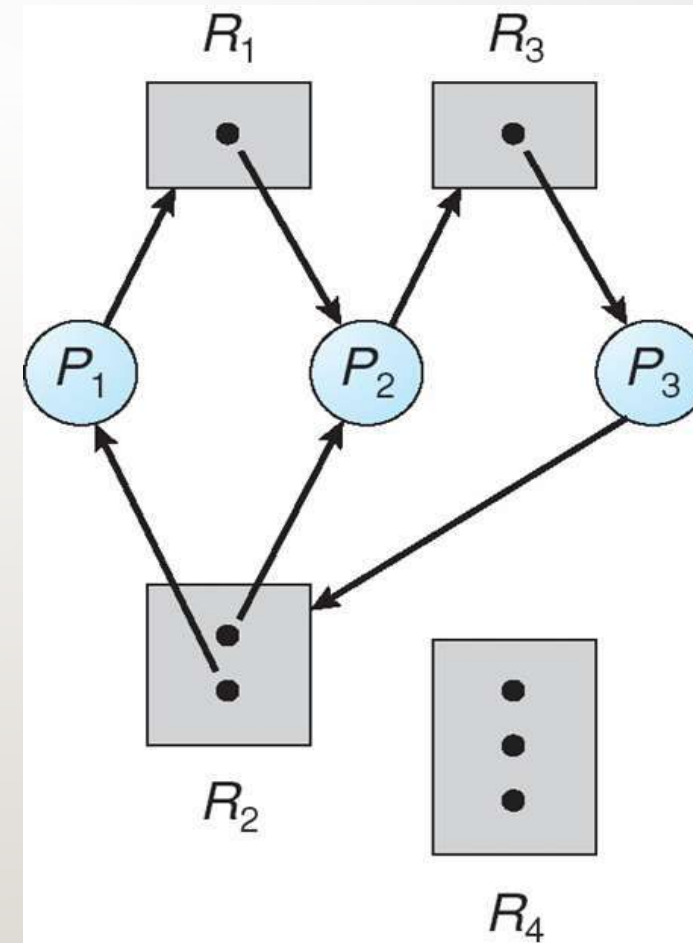


Unsafe State

# RESOURCE ALLOCATION GRAPH

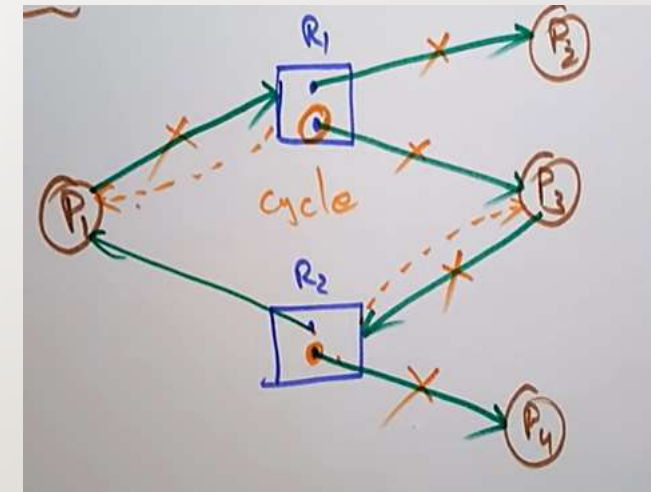
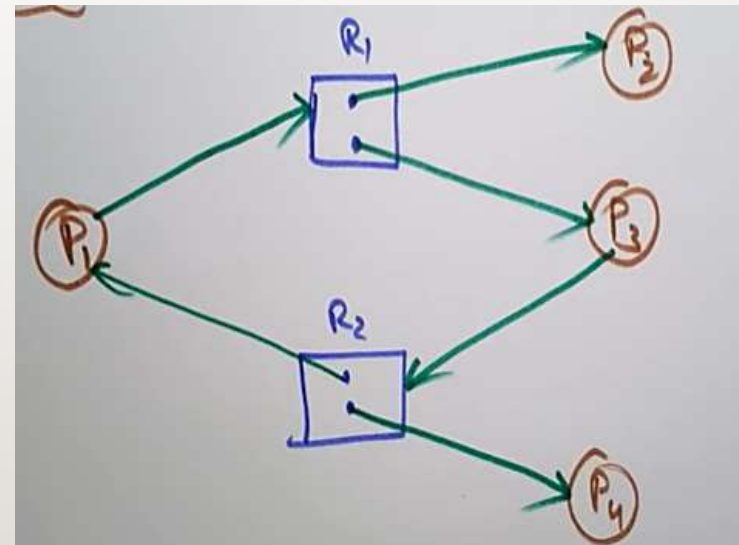
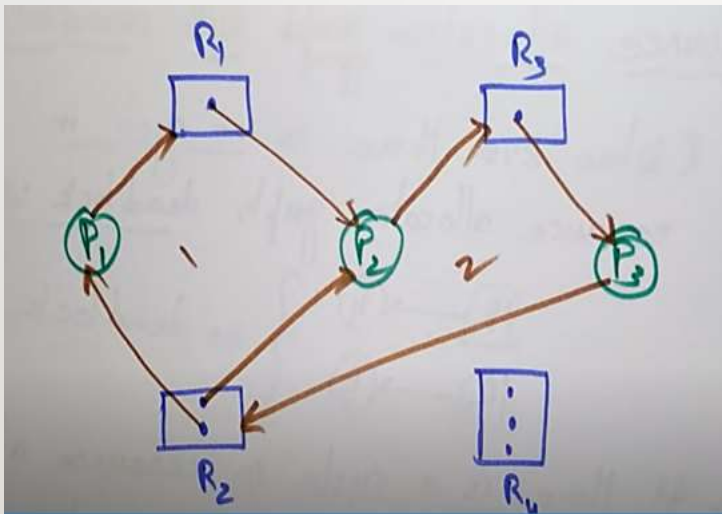


# RESOURCE ALLOCATION GRAPH WITH A DEADLOCK



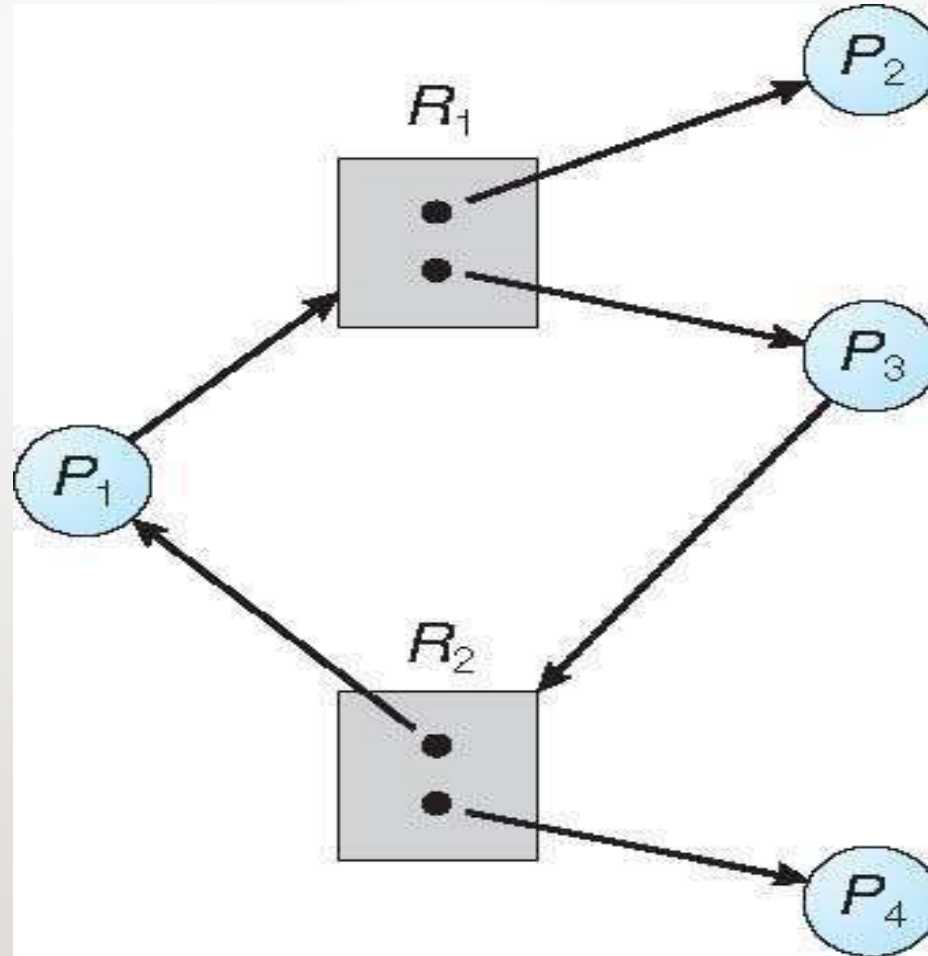
# WITH DEADLOCK ; WITHOUT DEADLOCK

Resources allocation graph with a cycle but no deadlock





# GRAPH WITH A CYCLE BUT NO DEADLOCK





- If graph contains no cycles  $\Rightarrow$  no deadlock
- If graph contains a cycle  $\Rightarrow$ 
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

# DEADLOCK PREVENTION

- Deadlock Prevention and Avoidance is the one of the methods for handling deadlock.
- In deadlock prevention the aim is to not let full-fill one of the required condition of the deadlock. This can be done by this method

# DEADLOCK PREVENTION

- **Mutual Exclusion** – not required for sharable resources (e.g., read-only files); must hold for non-sharable resources
- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources
  - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none allocated to it. Low resource utilization; starvation possible
- **No Preemption** –
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released

# DEADLOCK PREVENTION

- Preempted resources are added to the list of resources for which the process is waiting
- Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting
- **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

# DEADLOCK AVOIDANCE

- Requires that the system has some additional ***a priori*** information available
- Simplest and most useful model requires that each process declare the ***maximum number*** of resources of each type that it may need
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes

# WHAT IS DEADLOCK AVOIDANCE?

- Avoidance is kind of futuristic.
- By using the strategy of “Avoidance”, we have to make an assumption.
- We need to ensure that all information about resources that the process will need is known to us before the execution of the process.
- In prevention and avoidance, we get the correctness of data but performance decreases.

# WHAT IS DEADLOCK RECOVERY?

- If Deadlock prevention or avoidance is not applied to the software then we can handle this by deadlock detection and recovery. which consist of two phases:
- In the first phase, we examine the state of the process and check whether there is a deadlock or not in the system.
- If found deadlock in the first phase then we apply the algorithm for recovery of the deadlock.

# METHODS OF DEADLOCK RECOVERY

- There are several Deadlock Recovery Techniques:
- **Manual Intervention** : When a deadlock is detected, one option is to inform the operator and let them handle the situation manually.
- **Automatic Recovery** : An alternative approach is to enable the system to recover from deadlock automatically.
- **Process Termination** : Abort all Deadlocked Processes ; Abort one process at a time
- **Resource Preemption** : Resource preemption involves choosing which resources and processes should be preempted to break the deadlock.



# WHAT IS DEADLOCK IGNORANCE?

- If a deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take.
- In Deadlock, ignorance performance is better than the above two methods but the correctness of data is not there.

# WHAT IS DEADLOCK IGNORANCE?

- **Safe State**
- A safe state can be defined as a state in which there is no deadlock. It is achievable if:
- If a process needs an unavailable resource, it may wait until the same has been released by a process to which it has already been allocated. if such a sequence does not exist, it is an unsafe state.
- All the requested resources are allocated to the process.

# THANK YOU



## Team – System Design & Introduction to Cloud