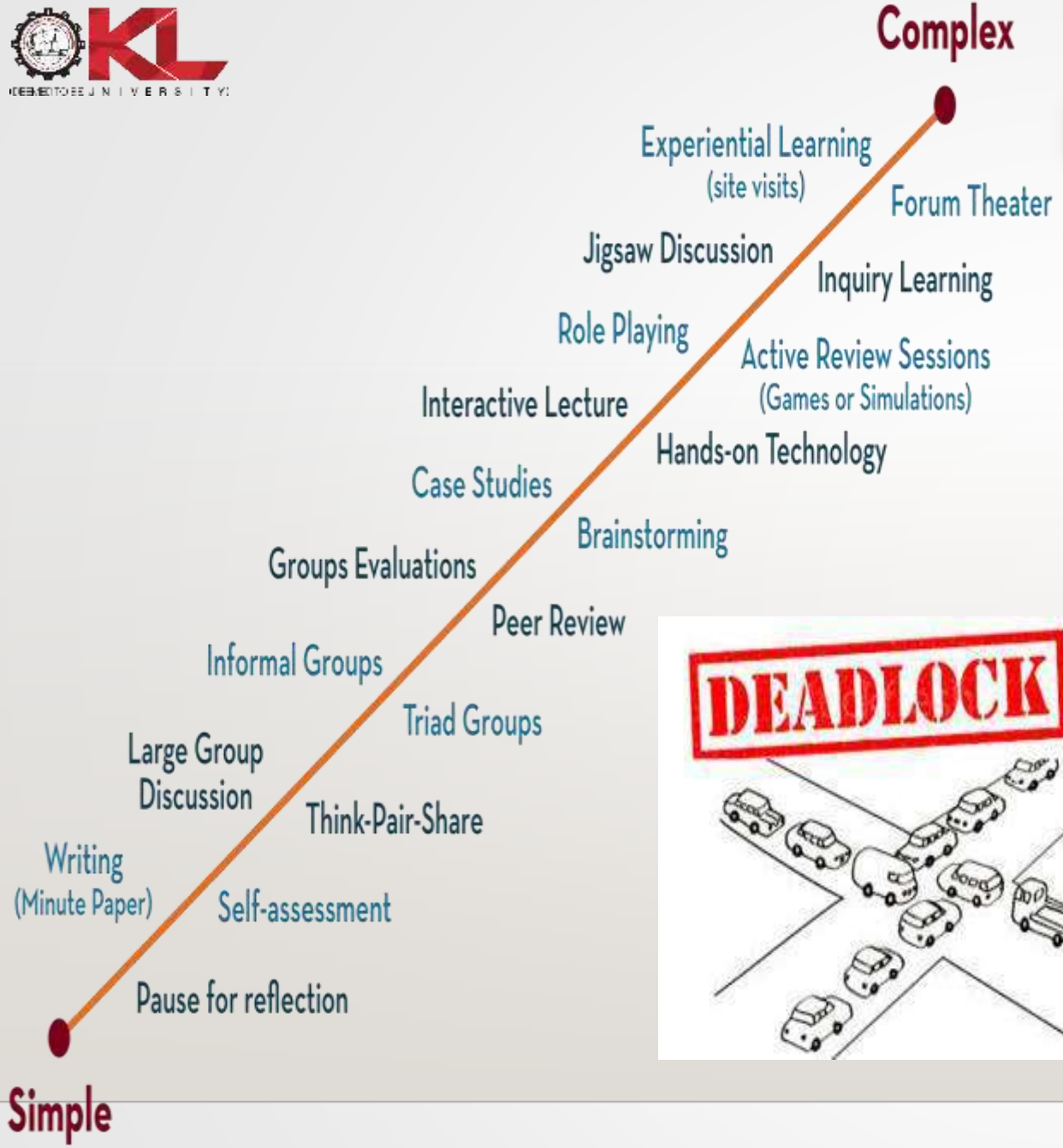


CO3 **Session - 5**

COURSE NAME: OPERATING SYSTEMS

COURSE CODE: 23CS2104R/A

Deadlock: Characterisation, Prevention, Avoidance.

AIM OF THE SESSION

To familiarize students with the basic concept of Deadlocks.

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1. Demonstrate what is meant by Deadlock.
2. Demonstrate what is meant by a Characterisation of A deadlock.
3. Describe the Methods of Handling Deadlocks .
4. Describe the banker's Algorithm.

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Defines what Deadlock is.
2. Defines Resource Allocation Graph.
3. Summarize the Concept of Deadlock.

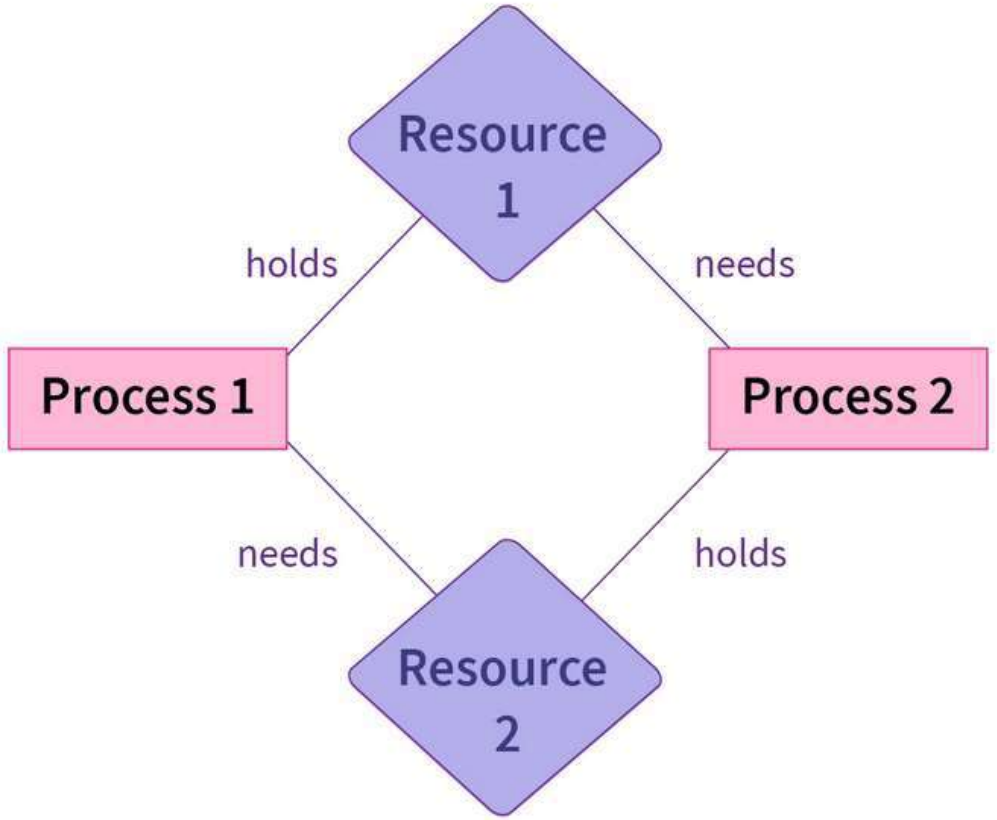
DEADLOCK

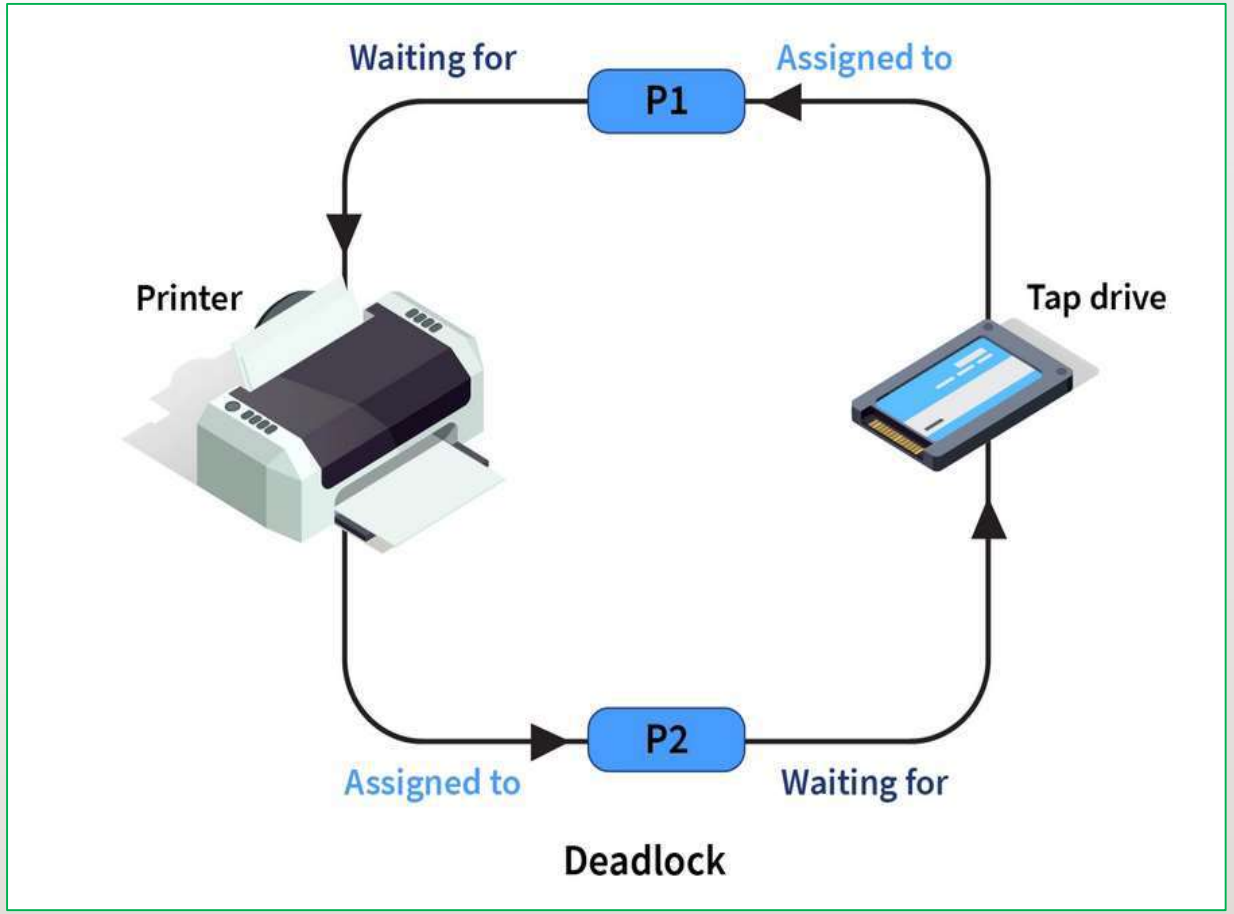
All the processes in a system require some resources such as central processing unit(CPU), file storage, input/output devices, etc to execute it.

Once the execution is finished, the process releases the resource it was holding. However, when many processes run on a system they also compete for these resources they require for execution. This may arise a deadlock situation.

Definition :A **deadlock** is a situation in which more than one process is blocked because it is holding a resource and also requires some resource that is acquired by some other waiting process . Therefore, none of the processes gets executed.

DEADLOCK EXAMPLE





SYSTEM MODEL

- System consists of resources
- Resource types $R_1, R_2, \ldots, R_m$

  CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:

A process in operating system uses resources in the following way.

- **Requests a resource**
- **Use the resource**
- **Releases the resource**

CHARACTERISATION OF A DEADLOCK

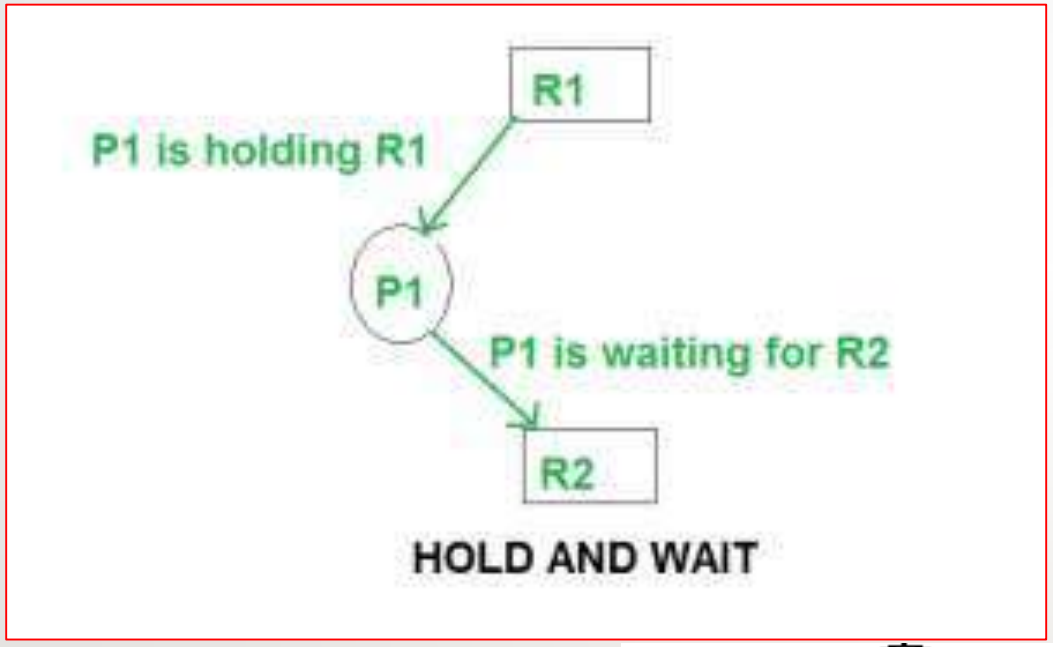
Necessary Conditions :

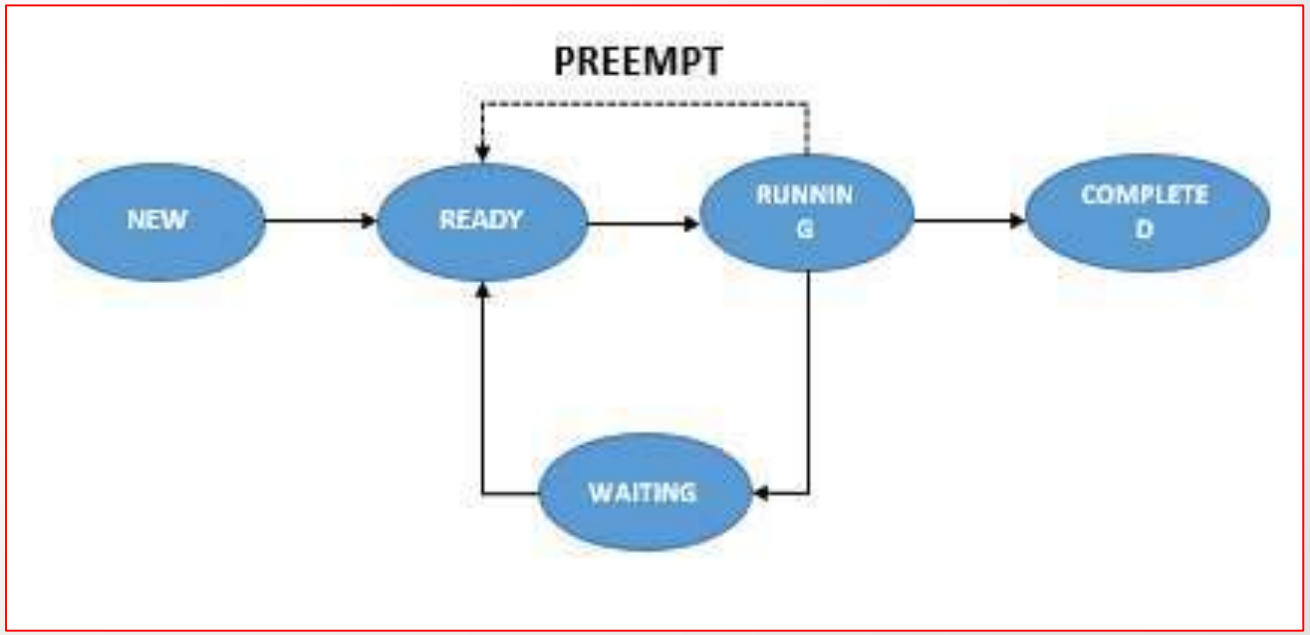
The **four** necessary conditions for a deadlock to arise are as follows.

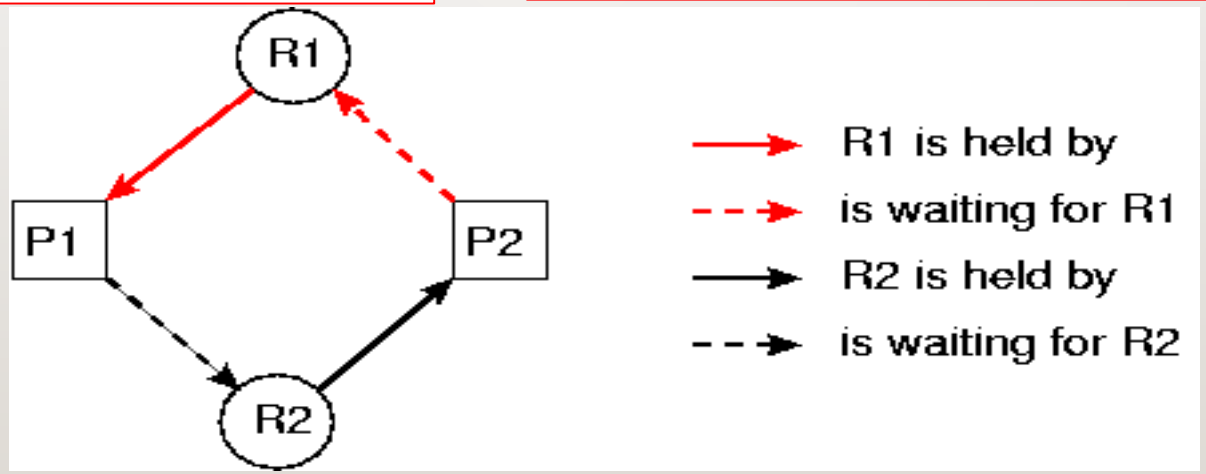
1. **Mutual Exclusion**: Only one process can use a resource at any given time i.e. the resources are non-sharable.
2. **Hold and wait**: A process is holding at least one resource at a time and is waiting to acquire other resources held by some other process.
3. **No preemption**: A resource cannot be taken from a process unless the process releases the resource.
4. **Circular Wait**: A set of processes are waiting for each other in a circular fashion.

For example, lets say there are a set of processes {P0,P1,P2,P3} such that P0 depends on P1, P1 depends on P2, P2 depends on P3 and P3 depends on P0. This creates a circular relation between all these processes and they have to wait forever to be executed.

CHARACTERISATION OF A DEADLOCK







DEADLOCK CHARACTERIZATION(CONTINUES)

Resource-Allocation Graph:

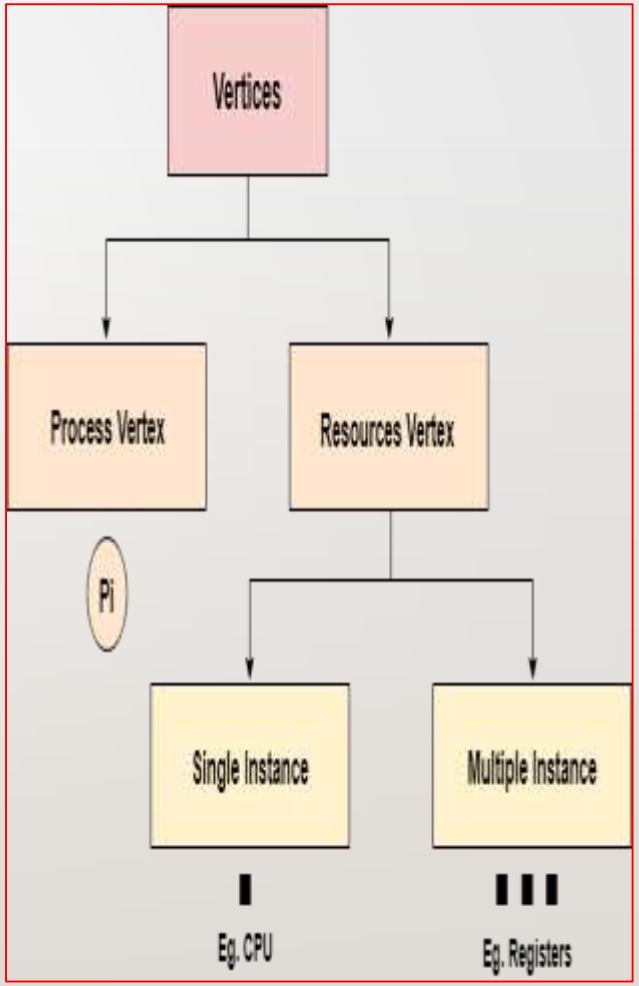
The resource allocation graph is the **pictorial representation of the state of a system**.

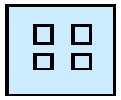
As its name suggests, the resource allocation graph is the complete information about all the processes which are **holding some resources or waiting for some resources**.

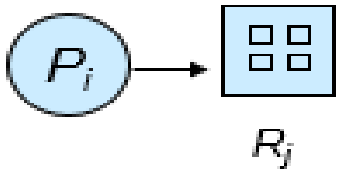
In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle. Let's see the types of vertices and edges in detail.

- A set of vertices *V* and a set of edges *E*.
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system

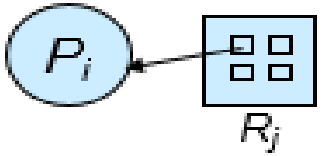
  $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system



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



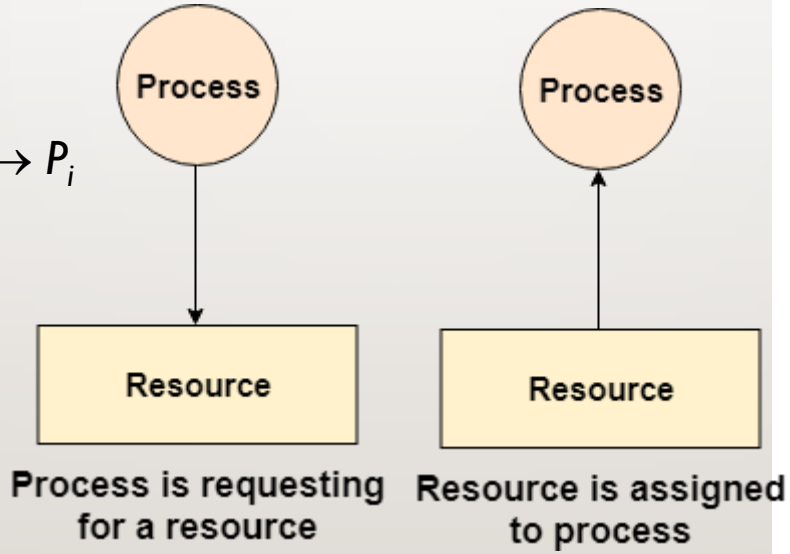
- P_i is holding an instance of R_j

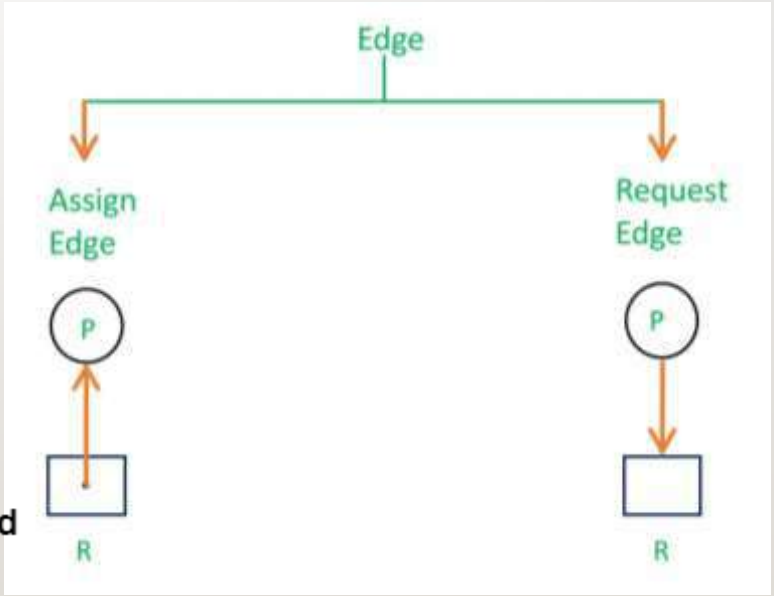


- Edges in RAG are also of two types, one represents assignment and other represents the wait of a process for a resource.
- A resource is shown as assigned to a process if the tail of the arrow is attached to an instance to the resource and the head is attached to a process.
- A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.

Request edge – directed edge $P_i \rightarrow R_j$

Assignment edge – directed edge $R_j \rightarrow P_i$



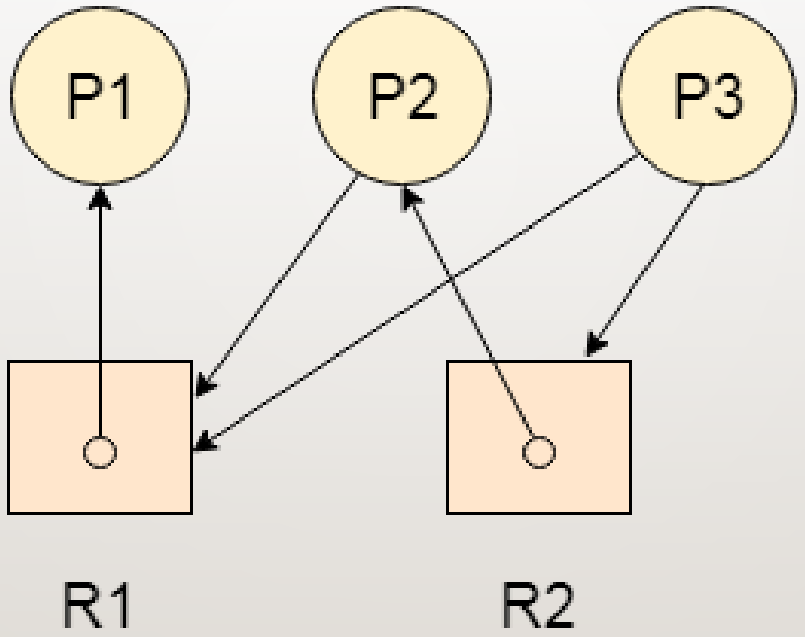


Example

Let's consider 3 processes P1, P2 and P3, and two types of resources R1 and R2.

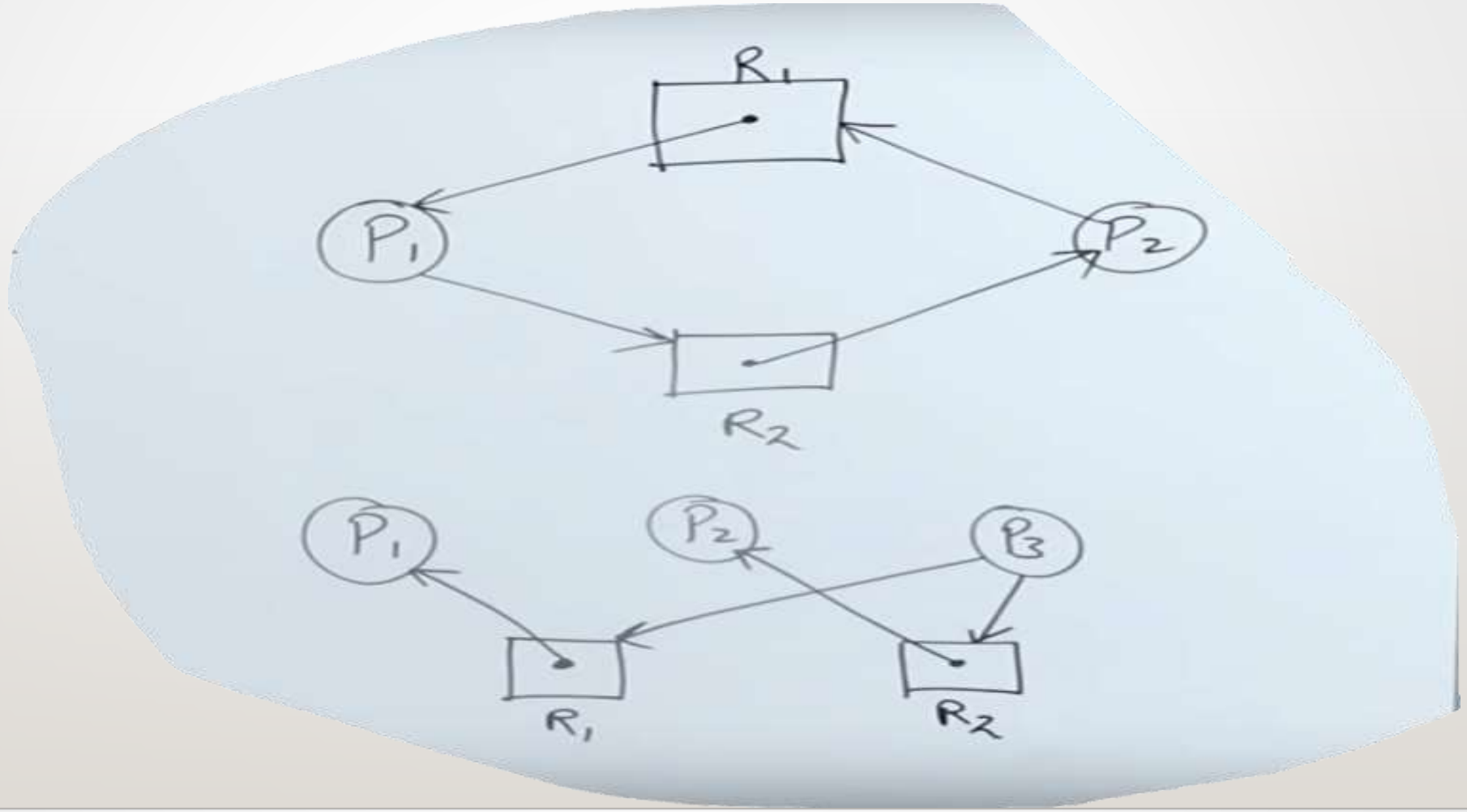
The resources are having 1 instance each.

According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2.

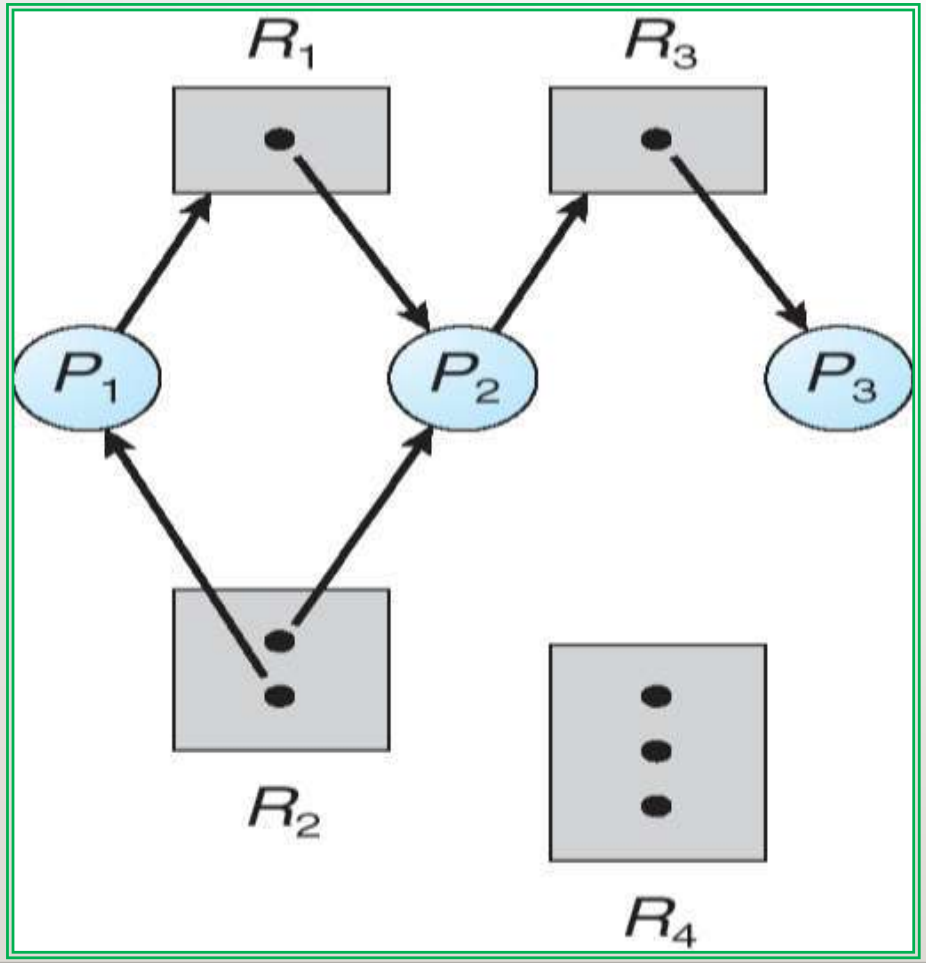


The graph is deadlock free since no cycle is being formed in the graph.

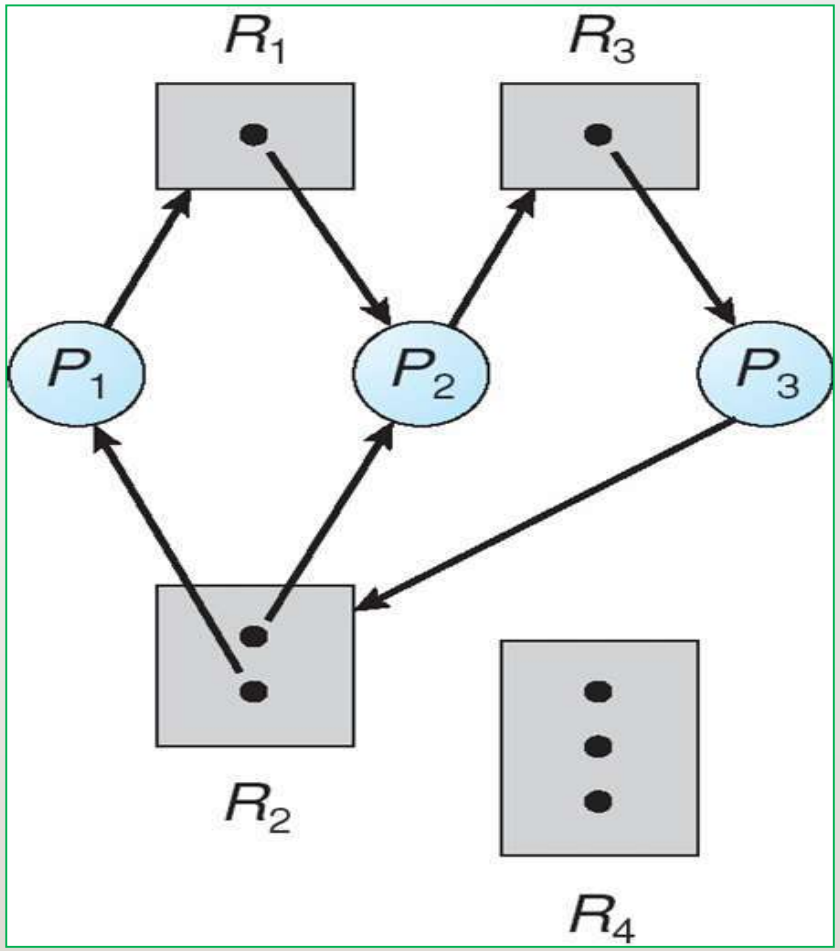
SINGLE INSTANCE RAG



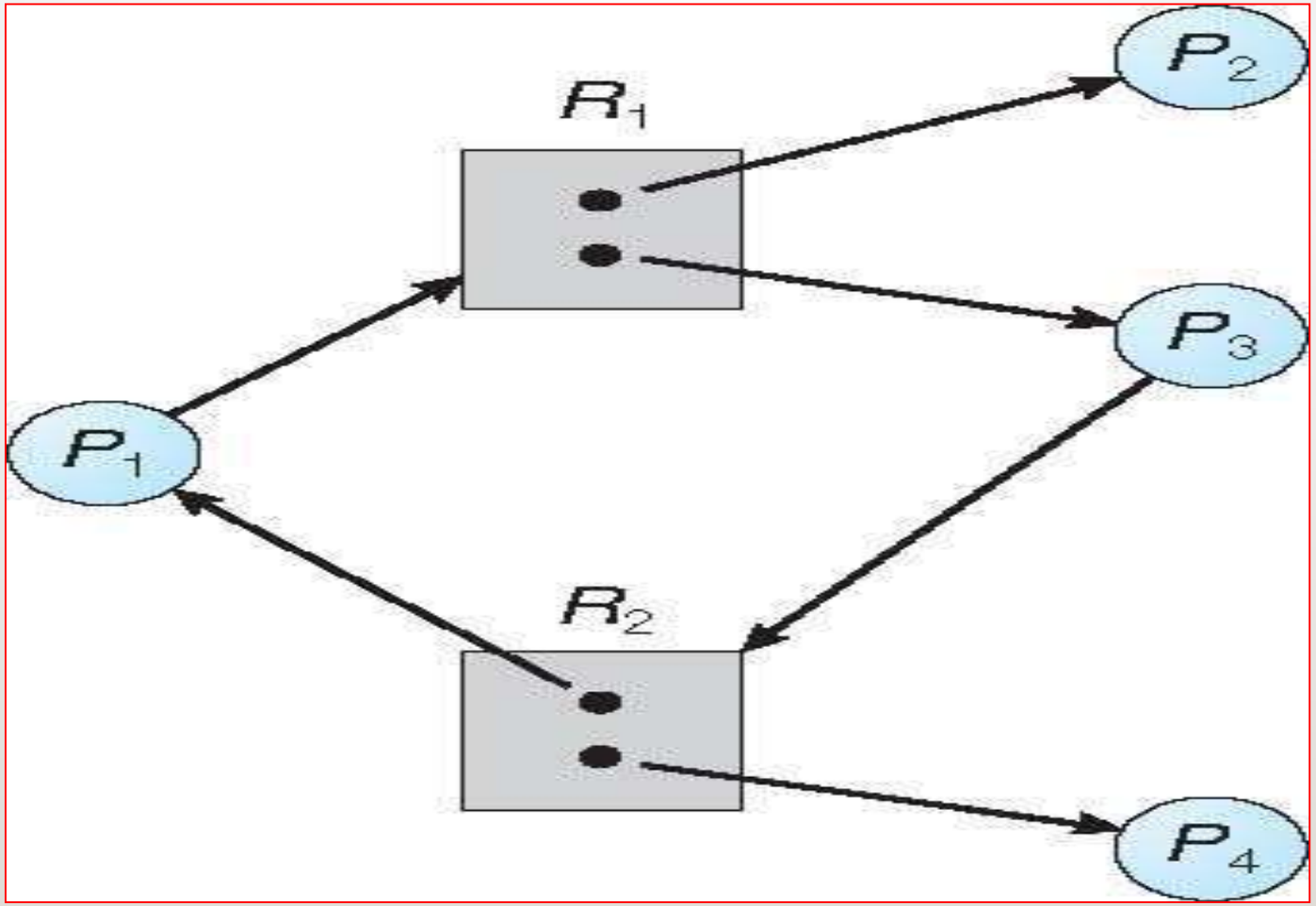
RESOURCE ALLOCATION GRAPH



RESOURCE ALLOCATION GRAPH WITH A DEADLOCK



GRAPH WITH A CYCLE BUT NO DEADLOCK



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

DIFFERENCE BETWEEN STARVATION AND DEADLOCK

Sr.	Deadlock	Starvation
1	Deadlock is a situation where no process got blocked and no process proceeds	Starvation is a situation where the low priority process got blocked and the high priority processes proceed.
2	Deadlock is an infinite waiting.	Starvation is a long waiting but not infinite.
3	Every Deadlock is always a starvation.	Every starvation need not be deadlock.
4	The requested resource is blocked by the other process.	The requested resource is continuously be used by the higher priority processes.
5	Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously.	It occurs due to the uncontrolled priority and resource management.

METHODS OF HANDLING DEADLOCKS

- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recover from Deadlock

DEADLOCK PREVENTION

The possibility of deadlock is excluded before making requests, by eliminating one of the necessary conditions for deadlock.

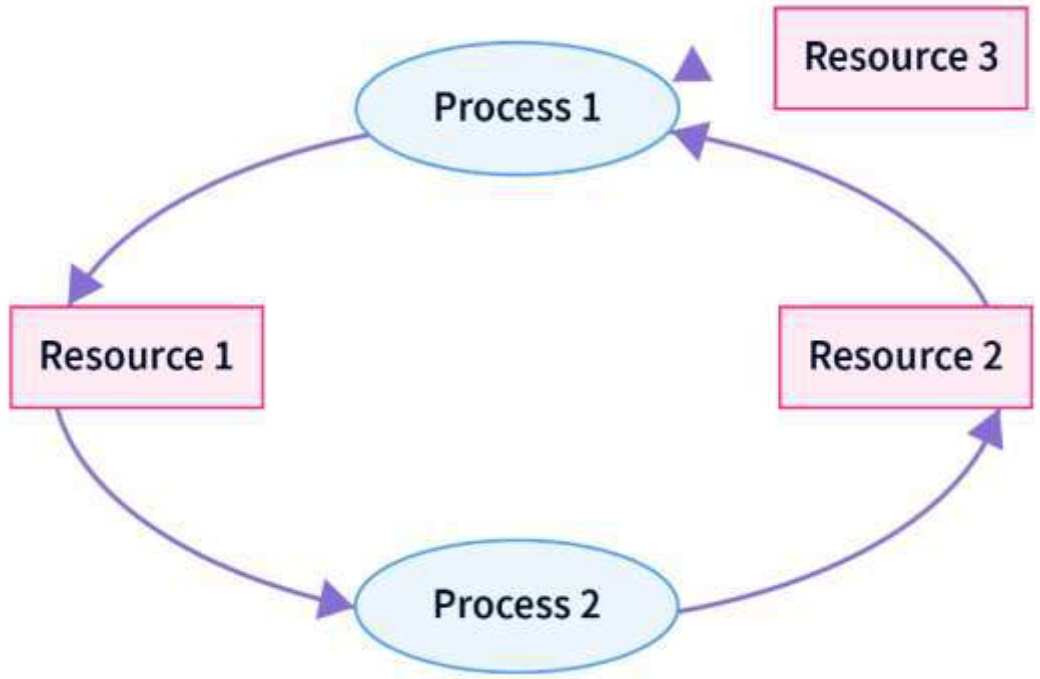
1.Mutual Exclusion

The mutual exclusion must be hold. That is , atleast one resource must be non sharable.

- Some resources are inherently non shareable, **for example**, Printers. For non shareable resources, processes require exclusive control of the resources.
- A **mutual exclusion** means that unshareable resources cannot be accessed simultaneously by processes.
- Sharable resources (**Read Only Files**) do not require mutual exclusion so don't cause deadlock but some resources can't be shared among processes, leading to a deadlock.

2. Hold and Wait

- Hold and wait is a condition in which a process is holding one resource while simultaneously waiting for another resource being held by another process. The process cannot continue till it gets all the required resources.



- Resource 1 is allocated to Process 2
- Resource 2 is allocated to Process 1
- Resource 3 is allocated to Process 1
- Process 1 is waiting for Resource 1 and holding **Resource 2** and **Resource 3**
- Process 2 is waiting for Resource 2 and holding Resource 1

There are two ways to eliminate hold and wait:-

1.By eliminating wait:

- The process specifies the resources it requires in advance so that it does not have to wait for allocation after execution starts.

For Example: Process1 declares in advance that it requires both Resource1 and Resource2

2. By eliminating hold:

- The process has to release all resources it is currently holding before making a new request.

For Example :Process1 has to release Resource2 and Resource3 before requesting Resource1.

3.No preemption

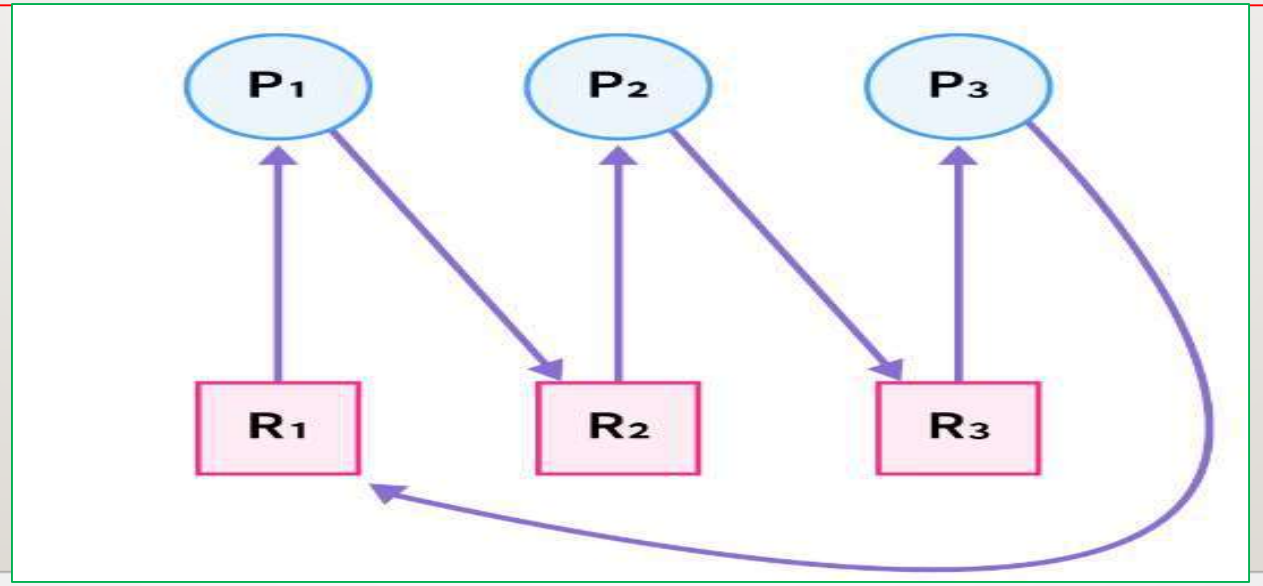
- Preemption is temporarily interrupting an executing task and later resuming it.

For example, if process P1 uses a resource and a high-priority process P2 requests for the resource, process P1 is stopped and the resources are allocated to P2.

4. Circular Wait: In circular wait, two or more processes wait for resources in a circular order. We can understand this better by the diagram given below:

- To eliminate circular wait, we assign a priority to each resource. A process can only request resources in increasing order of priority.

In the example process **P3** is requesting resource **R1**, which has a number lower than resource **R3** which is already allocated to process **P3**. So this request is invalid and cannot be made, as **R1** is already allocated to process **P1**.



DEADLOCK AVOIDANCE

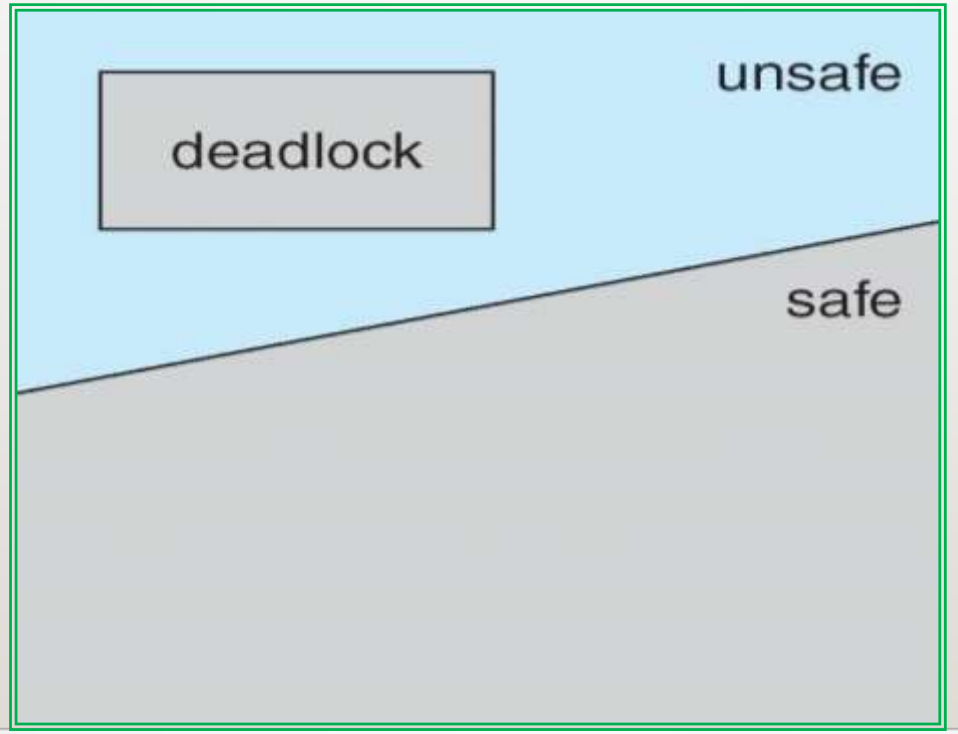
In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause a deadlock in the system.

- The state of the system will continuously be checked for safe and unsafe states.
- In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.

1. **Resource Allocation Graph**
2. **Banker's Algorithm(**Multiple instances of a resource type)

SAFE, UNSAFE, DEADLOCK STATE

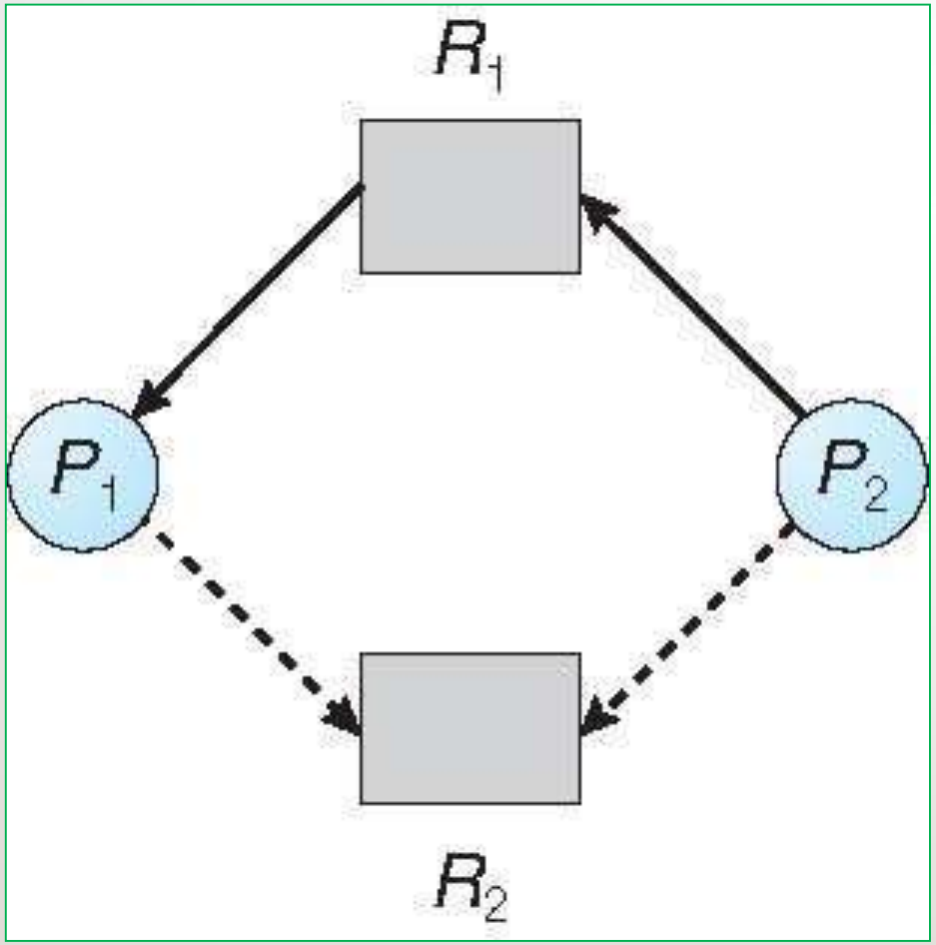
- If a system is in safe state ⇒ no deadlocks
- If a system is in unsafe state ⇒ possibility of deadlock
- Avoidance ⇒ ensure that a system will never enter an unsafe state.

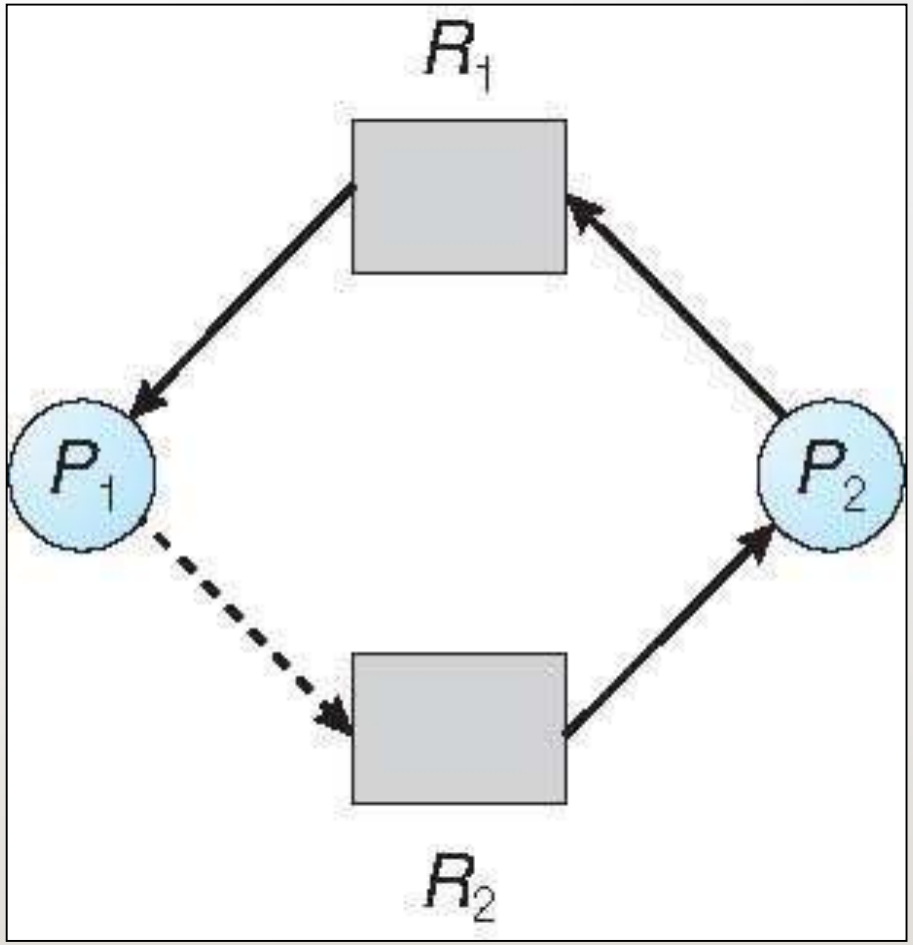


RESOURCE-ALLOCATION GRAPH SCHEME

- **Claim edge** P_i ---→ R_j indicated that process P_i may request resource R_j; represented by a **dashed line**.
- Claim edge converts to request edge when a process requests a resource.
- Request edge converted to an assignment edge when the resource is allocated to the process.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- Resources must be claimed *a priori* in the system.

RESOURCE-ALLOCATION GRAPH

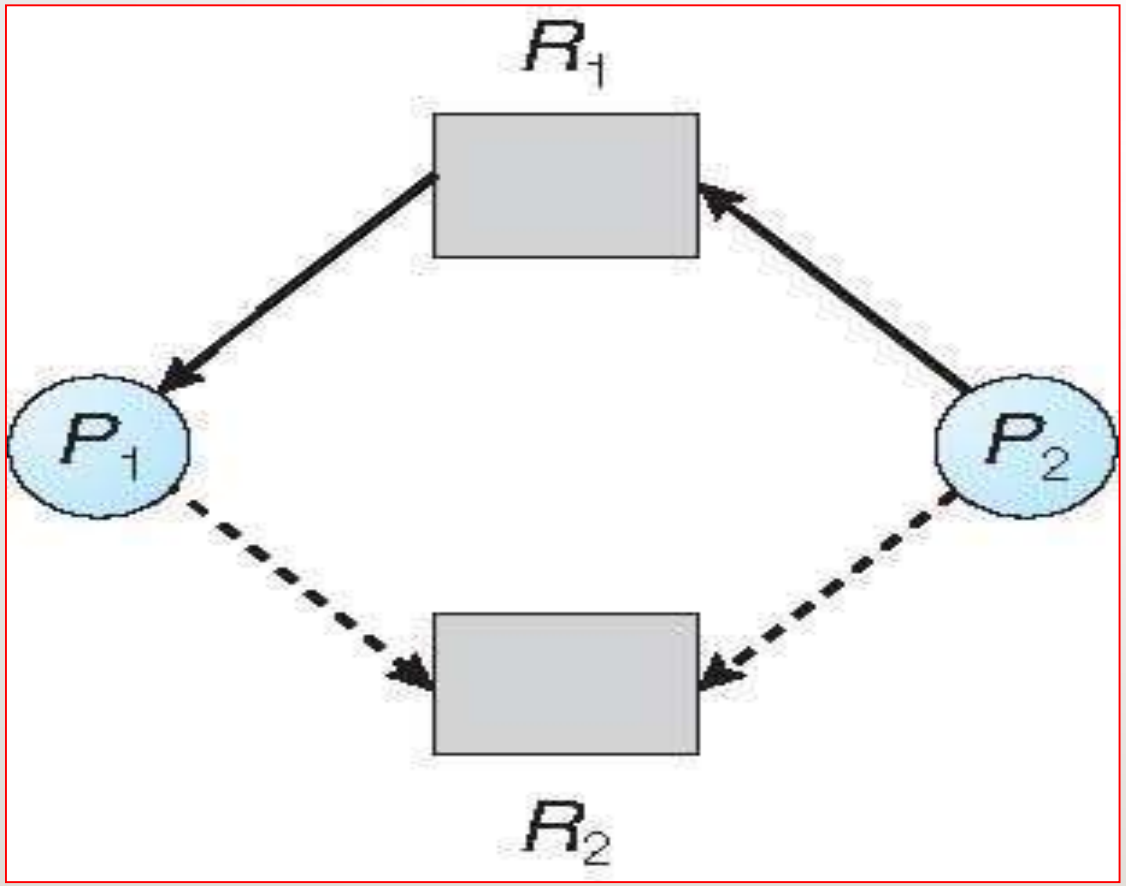


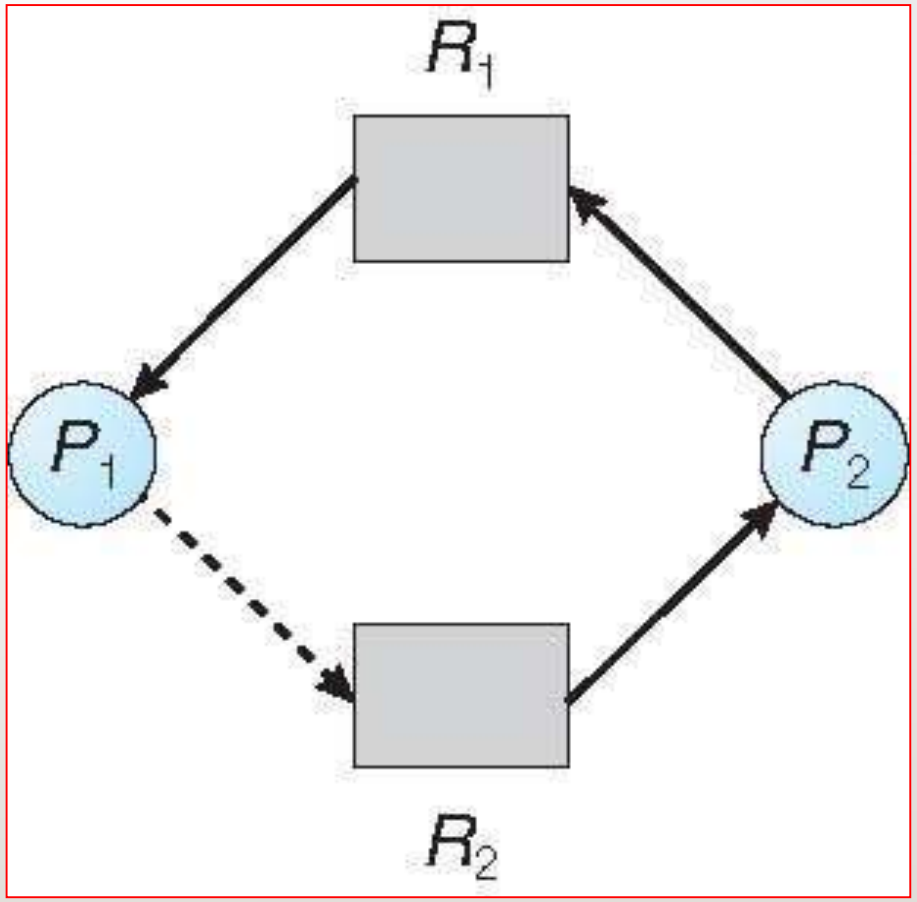


RESOURCE-ALLOCATION GRAPH ALGORITHM

- Suppose that process P_i requests a resource R_j
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph.

RESOURCE-ALLOCATION GRAPH





Unsafe State

BANKER'S ALGORITHM

- Multiple instances
- Each process must a priori claim maximum use
- When a process requests a resource it may have to wait
- When a process gets all its resources it must return them in a finite amount of time

DATA STRUCTURES FOR THE BANKER'S ALGORITHM

- Let n = number of processes, and m = number of resources types

- **Available**: Vector of length m. If available [j] = k, there are k instances of resource type R_j available
- **Max**: n x m matrix. If Max [i,j] = k, then process P_i may request at most k instances of resource type R_j
- **Allocation**: n x m matrix. If Allocation[i,j] = k then P_i is currently allocated k instances of R_j
- **Need**: n x m matrix. If $Need$[i,j] = k, then P_i may need k more instances of R_j to complete its task

$$Need\ [i,j] = Max[i,j] - Allocation\ [i,j]$$

SAFETY ALGORITHM

1. Let ***Work*** and ***Finish*** be vectors of length *m* and *n*, respectively. Initialize:

   Work = Available

   Finish [i] = false for i = 0, 1, …, n- 1

2. Find an ***i*** such that both:

   (a) ***Finish [i] = false***

   (b) $\boldsymbol{Need_i \leq Work}$

   If no such ***i*** exists, go to step 4

3. $\boldsymbol{Work = Work + Allocation_i}$

   Finish[i] = true

   go to step 2

4. If ***Finish [i] == true*** for all ***i***, then the system is in a safe state

RESOURCE-REQUEST ALGORITHM FOR PROCESS P_i

Request$_i$ = request vector for process P_i. If ***Request***$_i$ **[*j*] = *k*** then process P_i wants ***k*** instances of resource type R_j

1. If ***Request***$_i \leq$ ***Need***$_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If ***Request***$_i \leq$ ***Available***, go to step 3. Otherwise P_i must wait, since resources are not available
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

Available = Available – Request$_i$***;***

Allocation$_i$ **=** ***Allocation***$_i$ **+** ***Request***$_i$**;**

Need$_i$ **=** ***Need***$_i$ **–** ***Request***$_i$***;***

- If safe $\Rightarrow$ the resources are allocated to P_i
- If unsafe $\Rightarrow$ P_i must wait, and the old resource-allocation state is restored

THANK YOU

Team - Operating System