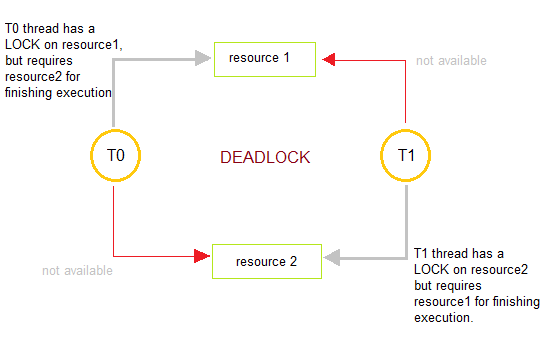
What is a Deadlock?

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.



How to avoid Deadlocks

Deadlocks can be avoided by avoiding at least one of the four conditions, because all this four conditions are required simultaneously to cause deadlock.

1. **Mutual Exclusion**

Resources shared such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process.

1. **Hold and Wait**

In this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.

1. **No Preemption**

Preemption of process resource allocations can avoid the condition of deadlocks, where ever possible.

1. **Circular Wait**

Circular wait can be avoided if we number all resources, and require that processes request resources only in strictly increasing(or decreasing) order.

Handling Deadlock

The above points focus on preventing deadlocks. But what to do once a deadlock has occured. Following three strategies can be used to remove deadlock after its occurrence.

1. **Preemption**

We can take a resource from one process and give it to other. This will resolve the deadlock situation, but sometimes it does causes problems.

1. **Rollback**

In situations where deadlock is a real possibility, the system can periodically make a record of the state of each process and when deadlock occurs, roll everything back to the last checkpoint, and restart, but allocating resources differently so that deadlock does not occur.

1. **Kill one or more processes**

This is the simplest way, but it works.

## What is a Livelock?

There is a variant of deadlock called livelock. This is a situation in which two or more processes continuously change their state in response to changes in the other process(es) without doing any useful work. This is similar to deadlock in that no progress is made but differs in that neither process is blocked or waiting for anything.

A human example of livelock would be two people who meet face-to-face in a corridor and each moves aside to let the other pass, but they end up swaying from side to side without making any progress because they always move the same way at the same time.

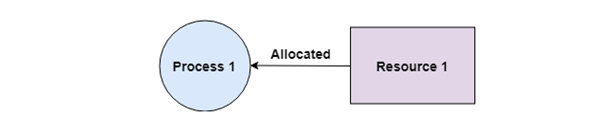
**Deadlock Characterization/DEADLOCK PREVENTION**

A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.

A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive. They are given as follows:

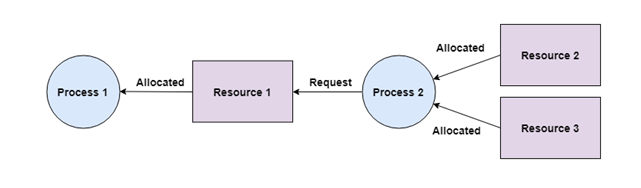
**Mutual Exclusion**

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.



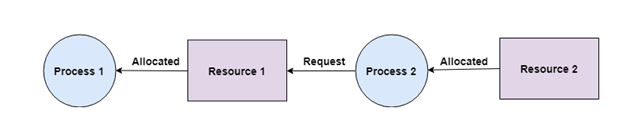
**Hold and Wait**

A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



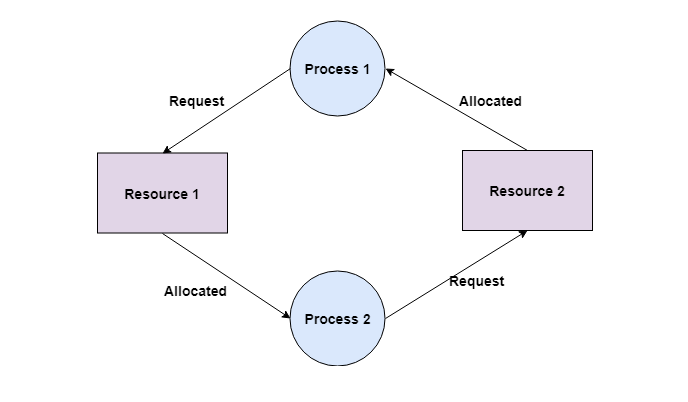
**No Pre-emption**

A resource cannot be pre-empted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot pre-empt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



**Circular Wait**

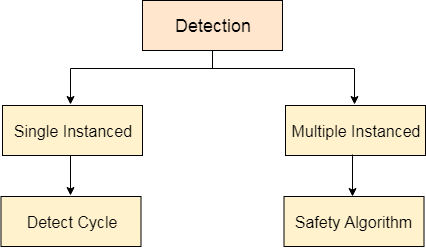
A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



Deadlock Detection and Recovery

In this approach, The OS doesn't apply any mechanism to avoid or prevent the deadlocks. Therefore the system considers that the deadlock will definitely occur. In order to get rid of deadlocks, The OS periodically checks the system for any deadlock. In case, it finds any of the deadlock then the OS will recover the system using some recovery techniques.

The main task of the OS is detecting the deadlocks. The OS can detect the deadlocks with the help of Resource allocation graph.



In single instanced resource types, if a cycle is being formed in the system then there will definitely be a deadlock. On the other hand, in multiple instanced resource type graph, detecting a cycle is not just enough. We have to apply the safety algorithm on the system by converting the resource allocation graph into the allocation matrix and request matrix.

Recovery from Deadlock in Operating System

Prerequisite – [Deadlock Detection And Recovery](https://www.geeksforgeeks.org/deadlock-detection-recovery/)  
When a [Deadlock Detection Algorithm](https://www.geeksforgeeks.org/operating-system-deadlock-detection-algorithm/) determines that a deadlock has occurred in the system, the system must recover from that deadlock. There are two approaches of breaking a [Deadlock](https://www.geeksforgeeks.org/operating-system-process-management-deadlock-introduction/):

**1. Process Termination:**  
To eliminate the deadlock, we can simply kill one or more processes. For this, we use two methods:

* **(a). Abort all the Deadlocked Processes:**  
  Aborting all the processes will certainly break the deadlock, but with a great expenses. The deadlocked processes may have computed for a long time and the result of those partial computations must be discarded and there is a probability to recalculate them later.
* **(b). Abort one process at a time until deadlock is eliminated:**  
  Abort one deadlocked process at a time, until deadlock cycle is eliminated from the system. Due to this method, there may be considerable overhead, because after aborting each process, we have to run deadlock detection algorithm to check whether any processes are still deadlocked.

**2. Resource Pre-emption:**  
To eliminate deadlocks using resource pre-emption, we pre-empt some resources from processes and give those resources to other processes. This method will raise three issues –

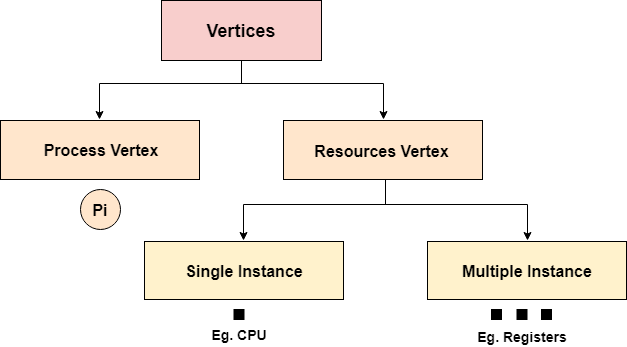
* **(a). Selecting a victim:**  
  We must determine which resources and which processes are to be pre-empted and also the order to minimize the cost.
* **(b). Rollback:**  
  We must determine what should be done with the process from which resources are pre-empted. One simple idea is total rollback. That means abort the process and restart it.
* **(c). Starvation:**  
  In a system, it may happen that same process is always picked as a victim. As a result, that process will never complete its designated task. This situation is called **Starvation** and must be avoided. One solution is that a process must be picked as a victim only a finite number of times.

# 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.

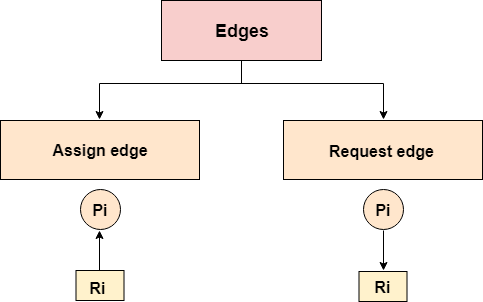
It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

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.



Vertices are mainly of two types, Resource and process. Each of them will be represented by a different shape. Circle represents process while rectangle represents resource.

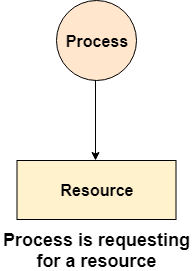
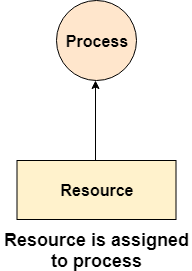
A resource can have more than one instance. Each instance will be represented by a dot inside the rectangle.



Edges in RAG are also of two types, one represents assignment and other represents the wait of a process for a resource. The above image shows each of them.

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.

### 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.

