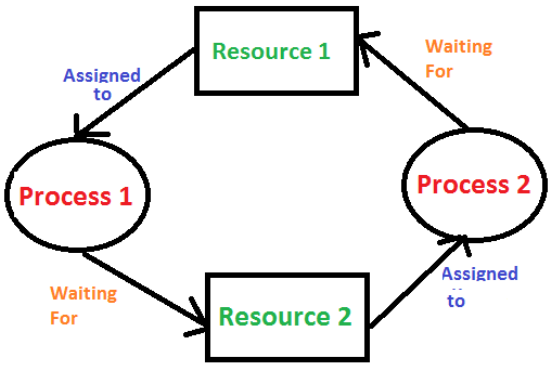
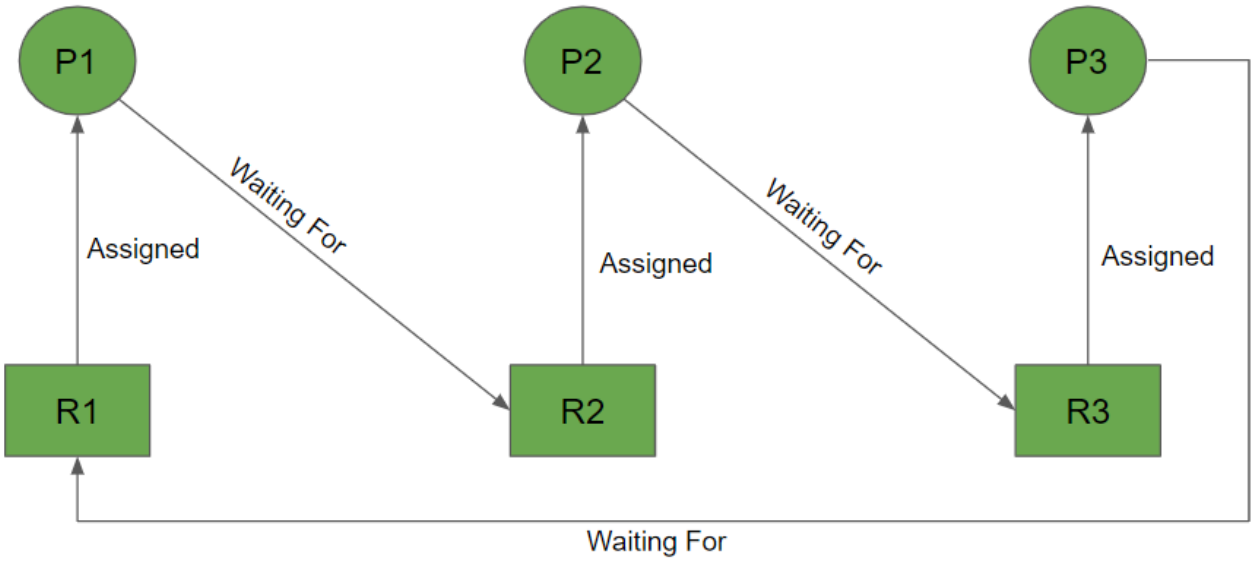
Deadlock - Characteristics

A process in operating systems uses different resources and uses resources in the following way.
1) Requests a resource
2) Use the resource
2) Releases the resource

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other. A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s). For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for the resource



This can also occur for more than two processes as shown in the below diagram:



Deadlock can arise if the following four conditions hold simultaneously (Necessary Conditions)

1. **Mutual Exclusion:** One or more than one resource are non-sharable (Only one process can use at a time)

2. **Hold and Wait:**A process is holding at least one resource and waiting for resources.

3. **No Preemption:**A resource cannot be taken from a process unless the process releases the resource.

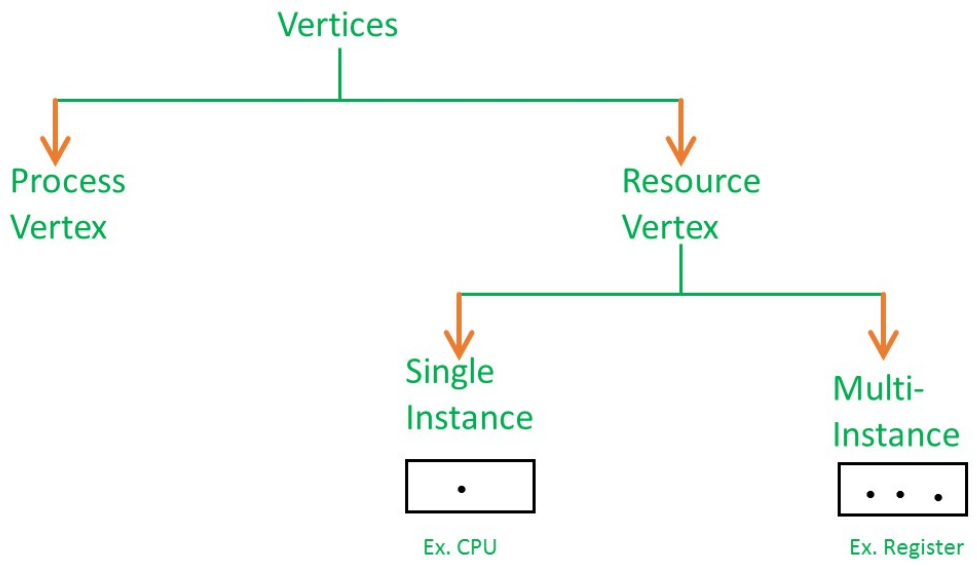
4. **Circular Wait:** A set of processes are waiting for each other in circular form.

Resource allocation graph: As Banker's algorithm using some kind of table like allocation, request, available all that thing to understand what is the state of the system. Similarly, if you want to understand the state of the system instead of using those tables, actually tables are very easy to represent and understand it, but then still you could even represent the same information in the graph. That graph is called **Resource Allocation Graph (RAG)**.

So, the resource allocation graph is explained to us what is the state of the system in terms of **processes and resources**. Like how many resources are available, how many are allocated and what is the request of each process. Everything can be represented in terms of the diagram. One of the advantages of having a diagram is, sometimes it is possible to see a deadlock directly by using RAG, but then you might not be able to know that by looking at the table. But the tables are better if the system contains lots of process and resource and Graph is better if the system contains less number of process and resource.
We know that any graph contains vertices and edges. So RAG also contains vertices and edges. In RAG vertices are two type -

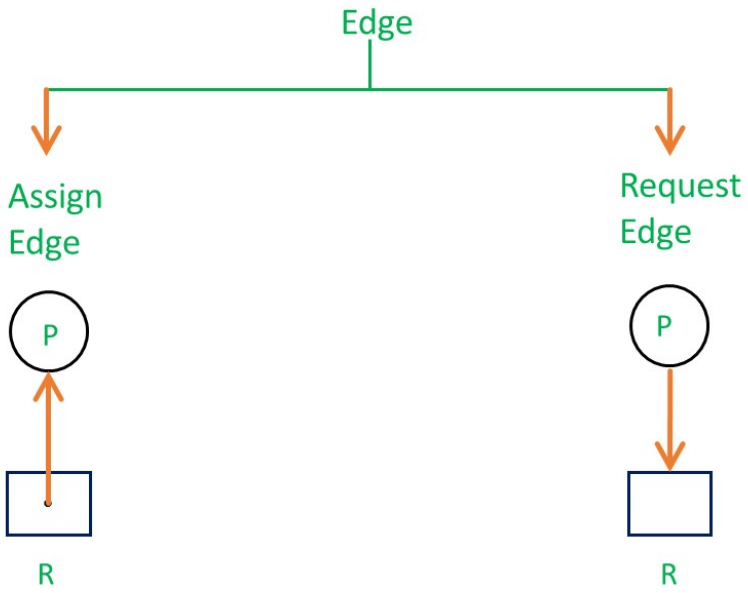
1. **Process vertex -** Every process will be represented as a process vertex.Generally, the process will be represented with a circle.

2. **Resource vertex -** Every resource will be represented as a resource vertex. It is also two type -
    - **Single instance type resource -** It represents as a box, inside the box, there will be one dot.So the number of dots indicate how many instances are present of each resource type.
    - **Multi-resource instance type resource -** It also represents as a box, inside the box, there will be many dots present.



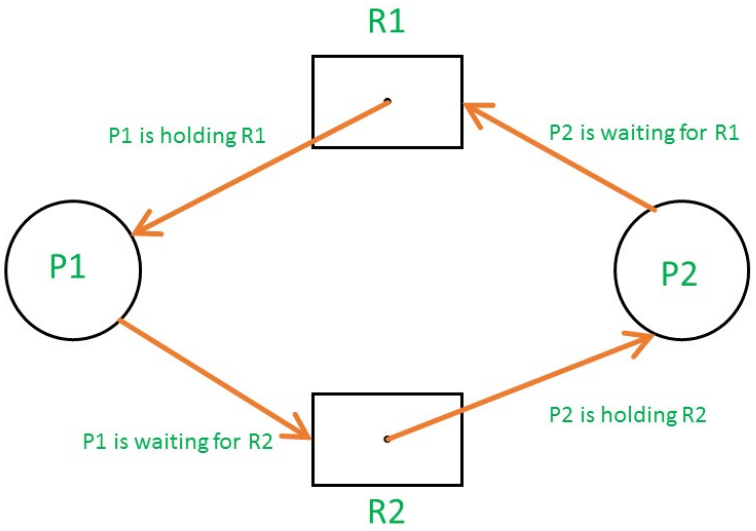
Now coming to the edges of RAG. There are two types of edges in RAG -

1. Assign Edge - If you already assign a resource to a process then it is called Assign edge.
2. Request Edge - It means in future the process might want some resource to complete the execution, that is called request edge.



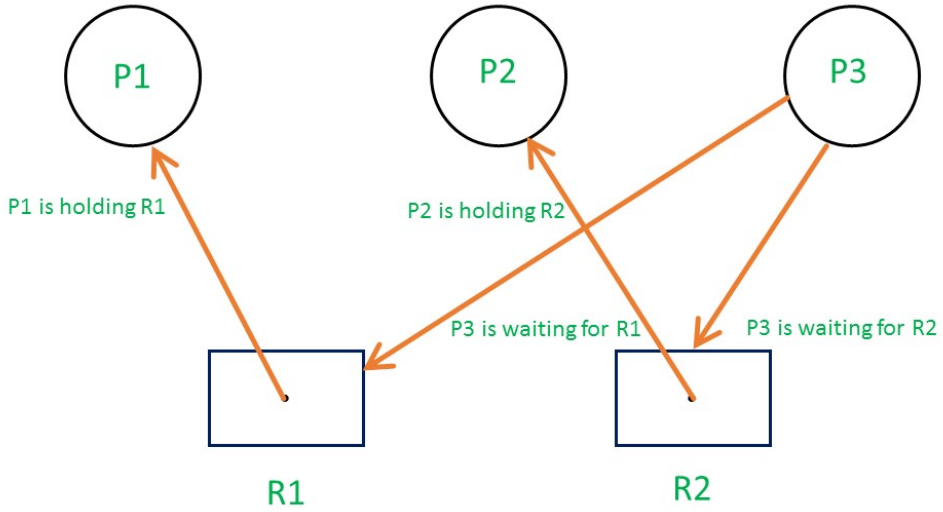
So, if a process is using a resource, an arrow is drawn from the resource node to the process node. If a process is requesting a resource, an arrow is drawn from the process node to the resource node.

Example 1 (Single instances RAG) -



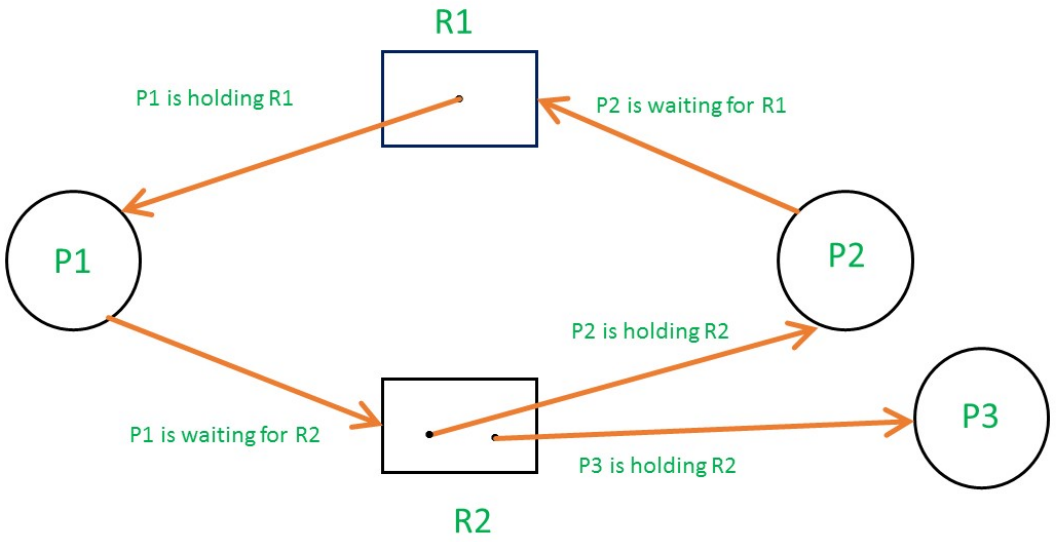
SINGLE INSTANCE RESOURCE TYPE WITH DEADLOCK

If there is a cycle in the Resource Allocation Graph and each resource in the cycle provides only one instance, then the processes will be in a deadlock. For example, if process P1 holds resource R1, process P2 holds resource R2 and process P1 is waiting for R2 and process P2 is waiting for R1, then process P1 and process P2 will be in a deadlock.



SINGLE INSTANCE RESOURCE TYPE WITHOUT DEADLOCK

Here's another example, that shows Processes P1 and P2 acquiring resources R1 and R2 while process P3 is waiting to acquire both resources. In this example, there is no deadlock because there is no circular dependency.
So cycle in single-instance resource type is the sufficient condition for deadlock.
Example 2 (Multi-instances RAG) -



MULTI INSTANCES WITHOUT DEADLOCK

From the above example, it is not possible to say the RAG is in a safe state or in an unsafe state. So to see the state of this RAG, let's construct the allocation matrix and request matrix.

Process	Allocation		Request	
	Resource		Resource	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	0	0

- The total number of processes are three; P1, P2 & P3 and the total number of resources are two; R1 & R2.

Allocation matrix -

- For constructing the allocation matrix, just go to the resources and see to which process it is allocated.
- R1 is allocated to P1, therefore write 1 in allocation matrix and similarly, R2 is allocated to P2 as well as P3 and for the remaining element just write 0.

Request matrix -

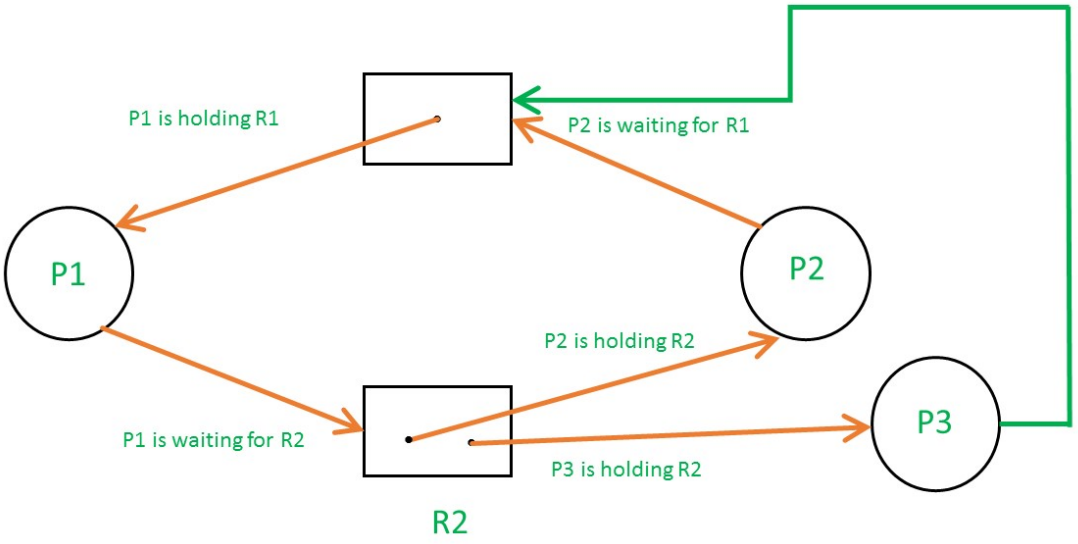
- In order to find out the request matrix, you have to go to the process and see the outgoing edges.
- P1 is requesting resource R2, so write 1 in the matrix and similarly, P2 requesting R1 and for the remaining element write 0.

So now available resource is = (0, 0).

Checking deadlock (safe or not) -

```
Available = 0  0   (As P3 does not require any extra resource to complete the execution and after
   P3       0  1    completion P3 release its own resource)
_____________________
New Available = 0  1  (As using new available resource we can satisfy the requirment of process P1
   P1           1  0   and P1 also release its prevous resource)
_____________________
New Available = 1  1 (Now easily we can satisfy the requirement of process P2)
   P2           0  1
_____________________
New Available =  1  2
```

So, there is no deadlock in this RAG. Even though there is a cycle, still there is no deadlock. Therefore in multi-instance resource cycle is not sufficient condition for deadlock.



MULTI INSTANCES WITH DEADLOCK

The above example is the same as the previous example except that, the process P3 requesting for resource R1. So the table becomes as shown in below.

Process	Allocation		Request	
	Resource		Resource	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	1	0

So,the Available resource is = (0, 0), but requirement are (0, 1), (1, 0) and (1, 0).So you can't fulfill any one requirement.Therefore, it is in deadlock.

Therefore, every cycle in a multi-instance resource type graph is not a deadlock, if there has to be a deadlock, there has to be a cycle. So, in case of RAG with a multi-instance resource type, the cycle is a necessary condition for deadlock, but not sufficient.

Deadlock - Handling Deadlock

Methods for handling deadlock There are three ways to handle deadlock

1. **Deadlock Prevention:** The OS accepts all the sent requests. The idea is to not to send a request that might lead to a deadlock condition.

2. **Deadlock Avoidance:** The OS very carefully accepts requests and checks whether if any request can cause deadlock and if the process leads to deadlock, the process is avoided.

3. **Deadlock Detection and Recovery:** Let deadlock occur, then do preemption to handle it once occurred.

4. **Ignore the problem altogether:** If the deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take.

Deadlock Characteristics As discussed in the previous post, deadlock has following characteristics.

1. Mutual Exclusion
2. Hold and Wait
3. No preemption
4. Circular wait

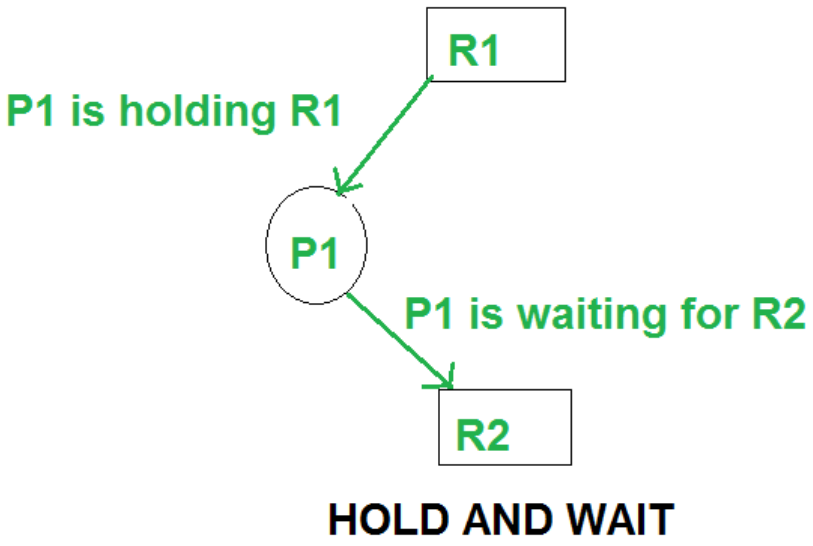
Deadlock Prevention

We can prevent Deadlock by eliminating any of the above four conditions.

- **Eliminate Mutual Exclusion:** It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tap drive and printer, are inherently non-shareable. It can be avoided to some extent using Spooling.

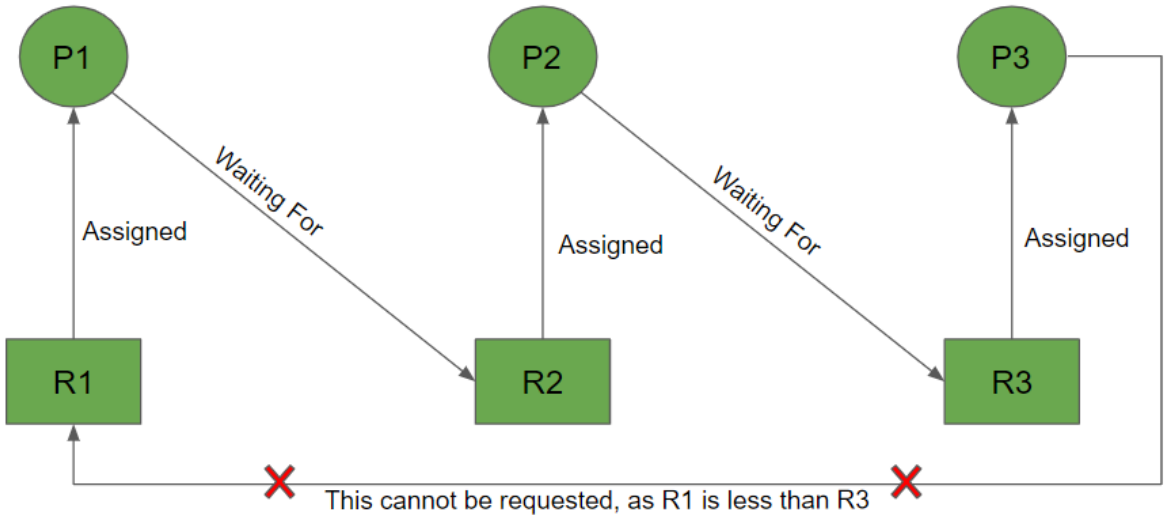
- **Eliminate Hold and wait:**
    1. Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. for example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remain blocked till it has completed its execution.

    2. The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.



- **Eliminate No Preemption:** Preempt resources from the process when resources required by other high priority processes.

- **Eliminate Circular Wait:** Each resource will be assigned with a numerical number. A process can request the resources only in increasing order of numbering.
  For Example, if the P1 process is allocated R1, P2 has R2 and P3 has R3, then P3 cannot request R1 which is less than R3.



Deadlock Avoidance
Deadlock avoidance can be done with Banker's Algorithm.

If a system does not employ either a deadlock prevention or deadlock avoidance algorithm then a deadlock situation may occur. In this case-

- Apply an algorithm to examine state of system to determine whether deadlock has has occurred or not.
- Apply an algorithm to recover from the deadlock. For more refer- Deadlock Recovery

Deadlock Detection Algorithm/Bankers Algorithm: It is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.
The algorithm employs several time varying data structures:

- **Available-** A vector of length m indicates the number of available resources of each type.
- **Allocation-** An n*m matrix defines the number of resources of each type currently allocated to a process. Column represents resource and resource represent process.
- **Request-** An n*m matrix indicates the current request of each process. If request[i][j] equals k then process P_i is requesting k more instances of resource type R_j.

We treat rows in the matrices Allocation and Request as vectors, we refer them as $Allocation_i$ and $Request_i$.

Steps of Algorithm:

1. Let *Work* and *Finish* be vectors of length m and n respectively. Initialize *Work= Available*. For *i=0, 1, ..., n-1*, if $Allocation_i$ = 0, then *Finish[i]* = true; otherwise, *Finish[i]*= false.
2. Find an index i such that both
   a) *Finish[i] == false* b) $Request_i$ <= *Work* If no such *i* exists go to step 4.
3. *Work= Work+* $Allocation_i$ *Finish[i]= true* Go to Step 2.
4. If *Finish[i]== false* for some i, 0<=i<n, then the system is in a deadlocked state. Moreover, if *Finish[i]==false* the process P_i is deadlocked.

Example 1:

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

1. In this, Work = [0, 0, 0] &

   Finish = [false, false, false, false, false]

2. *i=0* is selected as both Finish[0] = false and [0, 0, 0]<=[0, 0, 0].

3. Work =[0, 0, 0]+[0, 1, 0] =>[0, 1, 0] &

   Finish = [true, false, false, false, false].

4. *i=2* is selected as both Finish[2] = false and [0, 0, 0]<=[0, 1, 0].

5. Work =[0, 1, 0]+[3, 0, 3] =>[3, 1, 3] &

   Finish = [true, false, true, false, false].

6. *i=1* is selected as both Finish[1] = false and [2, 0, 2]<=[3, 1, 3].

7. Work =[3, 1, 3]+[2, 0, 0] =>[5, 1, 3] &

   Finish = [true, true, true, false, false].

8. *i=3* is selected as both Finish[3] = false and [1, 0, 0]<=[5, 1, 3].

9. Work =[5, 1, 3]+[2, 1, 1] =>[7, 2, 4] &

   Finish = [true, true, true, true, false].

10. *i=4* is selected as both Finish[4] = false and [0, 0, 2]<=[7, 2, 4].

11. Work =[7, 2, 4]+[0, 0, 2] =>[7, 2, 6] &

Finish = [true, true, true, true, true].

12. Since Finish is a vector of all true it means **there is no deadlock** in this example.

Example 2:
Considering a system with five processes P_0 through P_4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t_0 following snapshot of the system has been taken:

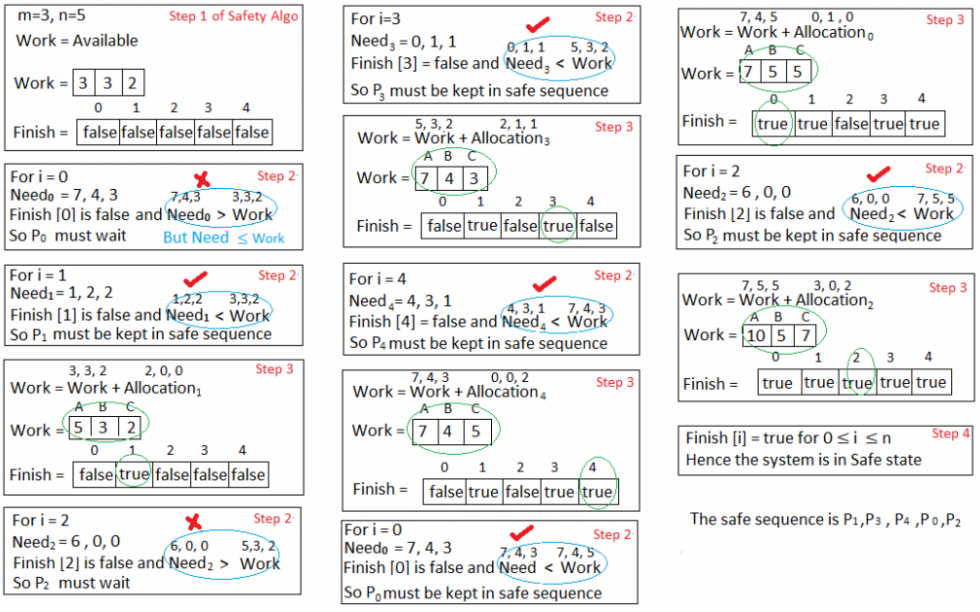
Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Question1. What will be the content of the Need matrix?
Need [i, j] = Max [i, j] – Allocation [i, j]

So, the content of Need Matrix is:

Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Question2. Is the system in a safe state? If Yes, then what is the safe sequence?
Applying the Safety algorithm on the given system,



Question3. What will happen if process P_1 requests one additional instance of resource type A and two instances of resource type C?

A B C
$Request_1$ = 1, 0, 2

To decide whether the request is granted we use Resource Request algorithm

Step 1

1, 0, 2 ≤ 1, 2, 2
$Request_1 \leq Need_1$

Step 2

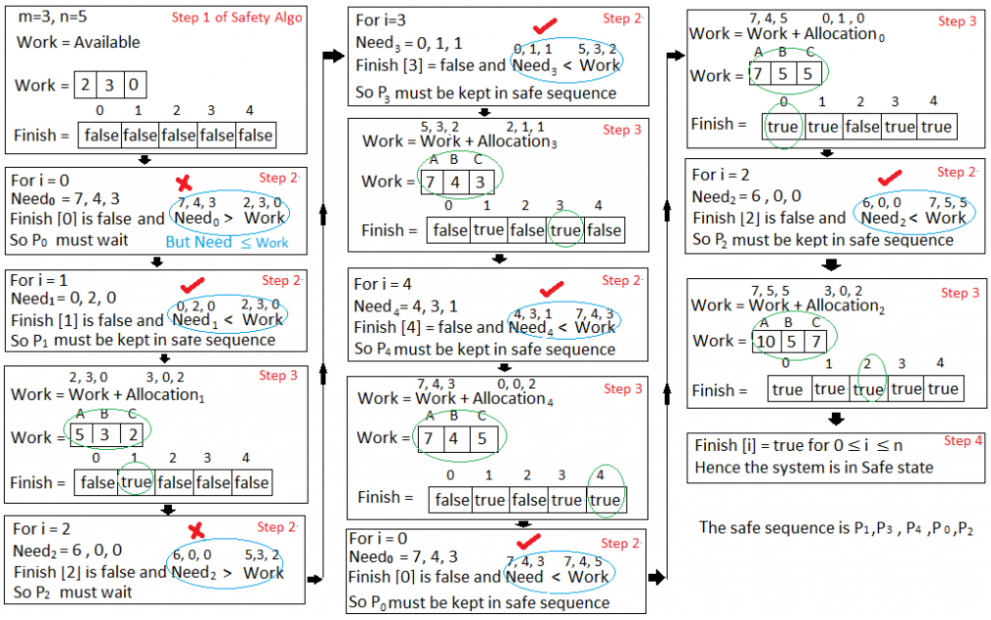
1, 0, 2 ≤ 3, 3, 2
$Request_1 \leq Available$

Step 3

$Available = Available - Request_1$
$Allocation_1 = Allocation_1 + Request_1$
$Need_1 = Need_1 - Request_1$

Process	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

We must determine whether this new system state is safe. To do so, we again execute Safety algorithm on the above data structures.



Hence the new system state is safe, so we can immediately grant the request for process P_1.

Recovery From Deadlock

There are three basic approaches to recover from deadlock:

1. Allow users to take manual intervention and inform the system operator.
2. Terminate one or more than one process which is involved in the deadlock.
3. Preempt resources.

How to terminate the process:

- There are two basic approaches to terminate processes and recover the resources allocated to those processes.
  - Terminate all processes involved in deadlock. Deadlock will be recovered but at the expense of more processes termination.
  - Terminate processes one by one until the deadlock is broken, this is a conservative approach and requires deadlock detection after each step.

- In another approach there are many factors that will decide which processes to terminate next:
  - Process priorities.
  - Running time of processes and remaining time to finish.
  - What type of resources are held by process and how many processes are held by the process.
  - How many requests for resources are made by the process.
  - How many processes need to be terminated.
  - Is the process interactive or batch?
  - Is there any non-restorable changes has been made by any process?

Resources preemption: There are three main issues to be addressed while relieving deadlock

- **Selecting a victim** It is a decision making the call to decide which resource must be released by which process. Decision criteria are discussed in the above section.
- **Rollback** Once the resources are taken back from a process, it is hard to decide the safe state in rollback or what is safe rollback. So, the safest rollback is the starting or beginning of the process.
- **Statrvation:** Once the preemption of resources started there is higher chance of starvation of process. Starvation can be avoided or can be decreased by the priority system.