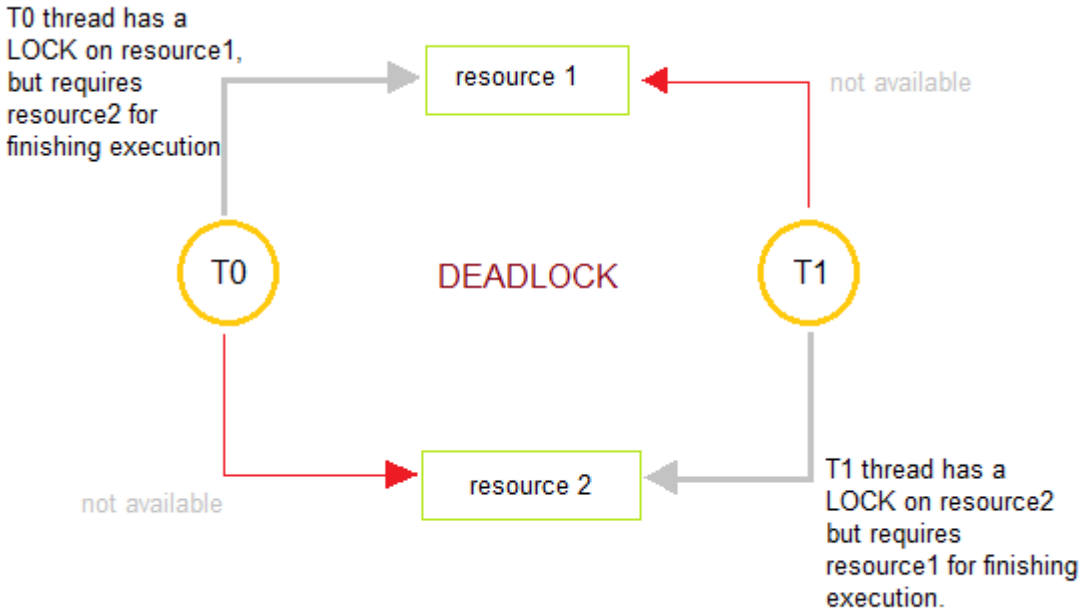
Deadlocks

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

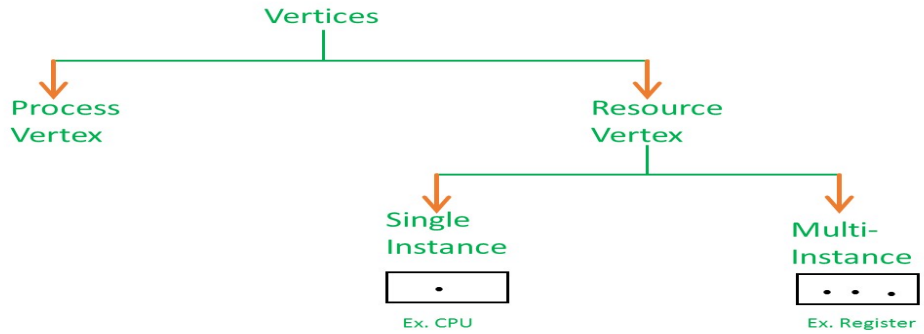


Resource Allocation Graph (RAG).

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.

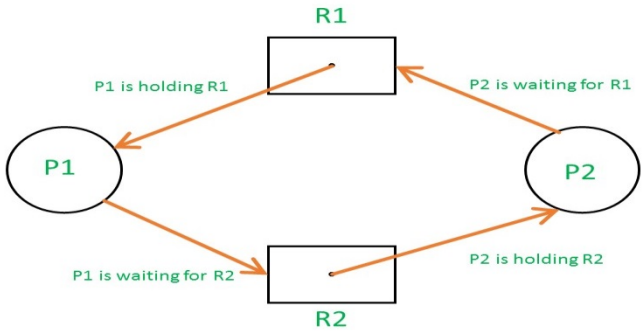


Now coming to the edges of RAG. There are two types of edges in RAG –

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.

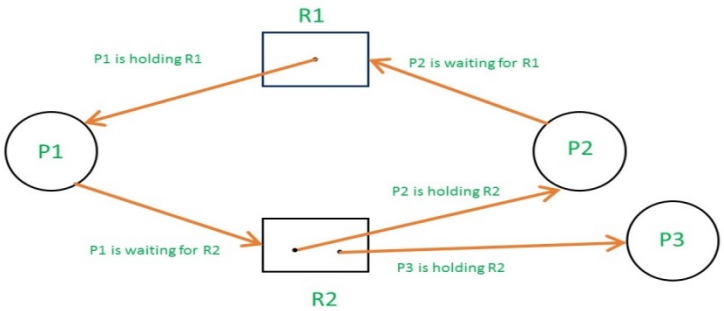
Example 1 (Single instances RAG) :



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 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 deadlock.

Example 2 (Multi-instances RAG) :



MULTI INSTANCES WITHOUT DEADLOCK

s P3 does not require any extra resource to complete the execution and after ompletion P3 release its own resource)

(As using new available resource we can satisfy the requirment of process P1 and P1 also release its prevous resource)

(Now easily we can satisfy the requirement of process P2)

Necessary conditions or characteristics for deadlock:

1. **Mutual Exclusion** : At least one unsharable resource - processes claim exclusive control of resources they need
2. **Hold and Wait** : Process *holds* one resource while waiting for another
3. **No Preemption** : Resources only released voluntarily - no interruption possible (i.e. cannot be forcefully withdrawn by another process)
4. **Circular Wait** : Circular chain of processes - each waiting for a resource held by another

Methods for Handling Deadlocks

Four strategies:

- ***Prevention***
- ***Avoidance***
- ***Detection***
- ***Recovery***

Deadlock Prevention

It's very simple trick as - *DENY* one of the four necessary conditions!

1. Make resources sharable (e.g spooling) $\equiv$ ***no mutual exclusion***
2. Processes MUST request ALL resources at the same time. Either all at start or release all before requesting more $\equiv$ ***no hold and wait*** *for* :
   Poor resource utilization and possible starvation
3. If process requests a resource which is unavailable - it must release all resources it currently holds - and try again later $\equiv$ ***allow preemption*** - loss of work
4. Impose an ordering on resource types. Process requests resources in a pre-defined order $\equiv$ ***no circular wait*** = this can be over restrictive

Avoidance

Bankers Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1) Let Work and Finish be vectors of length '**m**' and '**n**' respectively.

   Initialize: Work = Available

   Finish [i] = false; for i=1, 2, ..., n

2) Find an i such that both

   a) Finish [i] = false

   b) $Need_i \leq work$

      If no such i exists goto step (4)

3) Work = Work + $Allocation_i$

   Finish [i] = true

   goto step (2)

4) If Finish [i] = true for all i,

   then the system is in safe state.

- Processes request 1 resource at a time
- Request granted ONLY if it results in a safe state
- If request results in an unsafe state, it is denied and user holds resources and waits until request is eventually satisfied
- In finite time, all requests will be satisfied

Advantage: No deadlock and not as restrictive as deadlock prevention
Disadvantage: Fixed number of resources and users
Needs advanced knowledge of maximum needs
Unnecessary delays in avoiding unsafe states which may not lead to deadlock

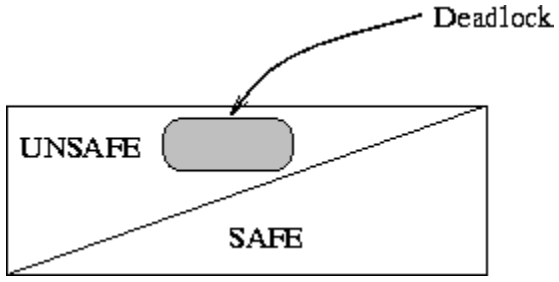


Figure 2: Bankers algorithm (safe/unsafe states)

Therefore, from Bankers algorithm :
Allow a thread (process) to proceed if : (Total available resources) - (number allocated) $\geq$ (maximum remaining that might be needed by the thread (process))

Example:

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:

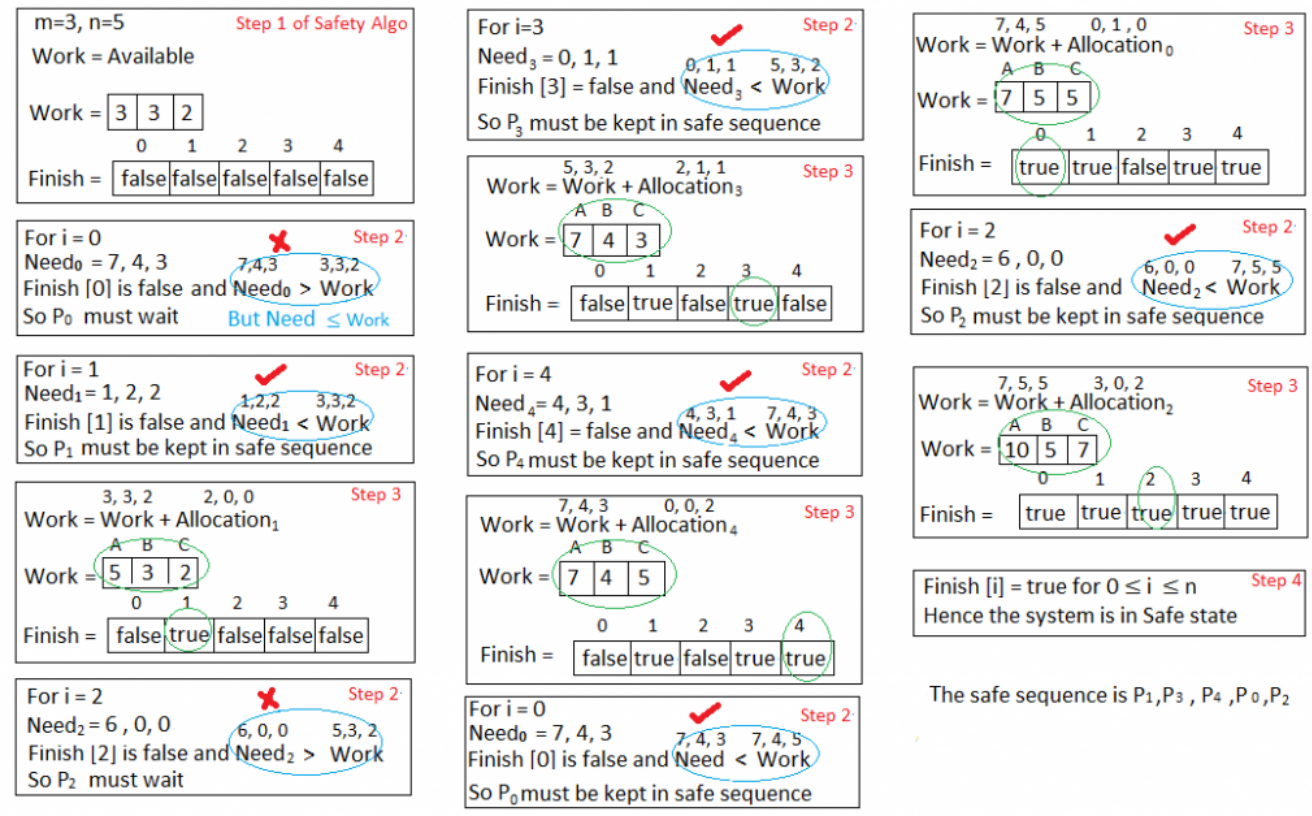
Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

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

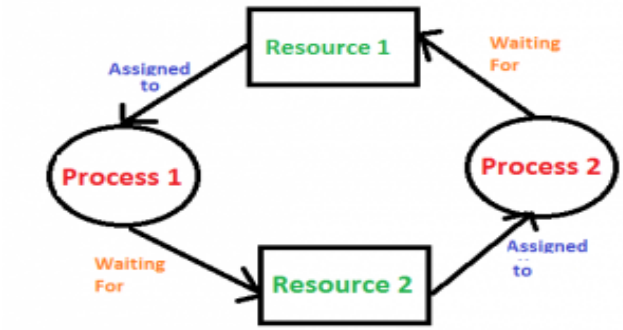
Question2. Is the system in a safe state? If Yes, then what is the safe sequence?
Applying the Safety algorithm on the given system,



Deadlock Detection

1. If resources have single instance:

In this case for Deadlock detection we can run an algorithm to check for cycle in the Resource Allocation Graph. Presence of cycle in the graph is the sufficient condition for deadlock.



In the above diagram, resource 1 and resource 2 have single instances. There is a cycle R1–>P1–>R2–>P2. So Deadlock is confirmed.

2. If there are multiple instances of resources:

Detection of cycle is necessary but not sufficient condition for deadlock detection, in this case system may or may not be in deadlock varies according to different situations. Again we need to find out whether closed path like above diagram exists, if yes then deadlocked state and if not, safe state.

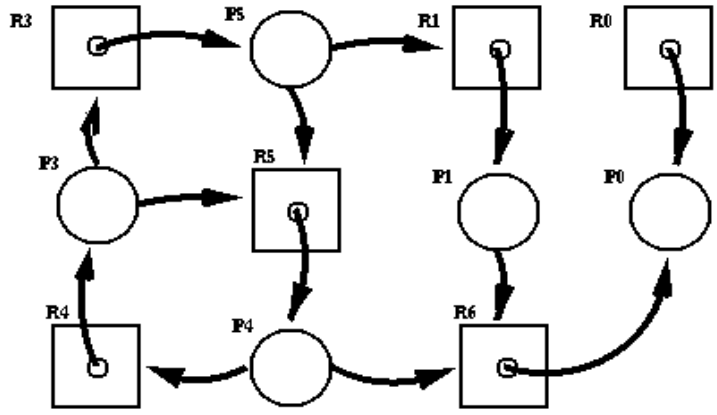
Deadlock Recovery

- **Abort all deadlock processes** and release resource - *too drastic* - will lead to loss of work
- **Abort one process at a time** - releasing resources until no deadlock
  - *priority ordering*, *process which has done least work*
- **Selectively restart** processes from a previous *checkpoint* i.e. before it claimed any resources
- **Successively withdraw resources** from a process and give to another process until deadlock is broken.

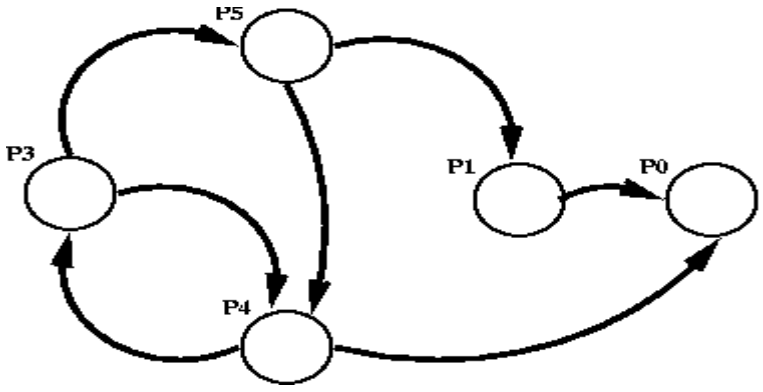
Wait-For-Graphs are also used to detect deadlocks

A Wait-For Graph (**WFG**) is a graph where Each node represents a process;
An edge, P_i --< P_j means that P_i is blocked waiting for P_j to release a resource.
Recall the single-instance RAG given earlier

Example1: A WFG can be derived from below graph.



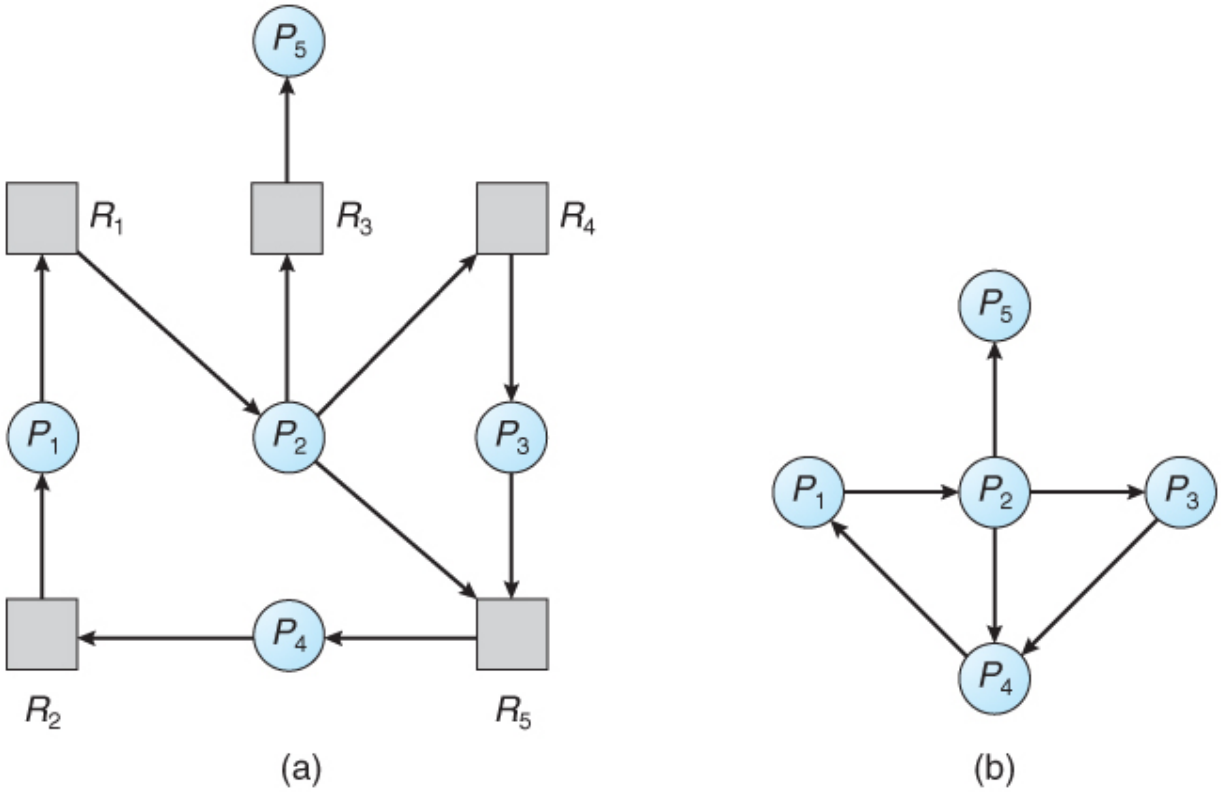
A Wait-For Graph (**WFG**) is the same as the SRAG with the resource elements stripped out.



The cycles can be seen more clearly in this form of the graph.

There is deadlock in the system if and only if there exists a cycle or a knot in the wait-for-graph.

Example2:



(a) Resource allocation graph. (b) Corresponding wait-for graph

Other example without banker's algorithm:

A system has 12 magnetic tape drives and 3 processes : P0, P1, and P2. Process P0 requires 10 tape drives, P1 requires 4 and P2 requires 9 tape drives. (Solve without using Banker's algorithm) Process : P0 , P1, P2 , Maximum Needs (Process-wise : P0 through P2 top to bottom) 10 ,4, 9 Currently allocated (process-wise) : 5 , 2 , 2

Process	Maximum Need	Allocated	Current Need
P0	10	5	5
P1	4	2	2
P2	9	2	7

- At time t0, the system is in a safe state. The sequence <P1, P0, P2> satisfies the safety condition.
- A system can go from a safe state to an unsafe state.
- Suppose that, at time t1, process P2 requests and is allocated one more tape drive. The system is no longer in a safe state.

Process	Maximum Need	Allocated	Current Need
P0	10	5	5
P1	4	2	2
P2	**9**	**3**	**6**

- Now only 2 tape drives are free.

- At this point, only process P1 can be allocated all its tape drives. When it returns them, the system will have only **four** available tape drives. Since process Po is allocated five tape drives but has a maximum of ten, it may request five more tape drives. If it does so, it will have to wait, because they are unavailable. Similarly, process P2 may request six additional tape drives and have to wait, resulting in a deadlock. Our mistake was in granting the request from process P2 for one more tape drive.