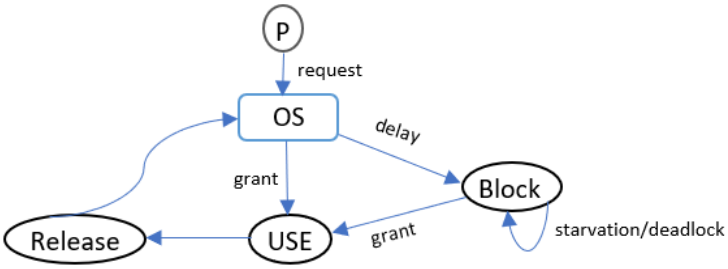
Deadlock:

Deadlock is a situation in which multiple process in multiprogramming environment may compete for a finite number of resources. If a process requests for resources and the resources is not available at that time, then process enters a waiting state. Sometimes, this waiting process is never again able to change state as the requested resources are held by some other waiting processes.



In normal mode of operation, the process may utilize the resource in the given manner as:

1. **Request (system call)**: If the process request a resource, if the request cannot grant immediately, then the process must wait until it can acquire the resource.
2. Use: the process can use the resource.
3. **Release (system call)**: the process releases the resource.

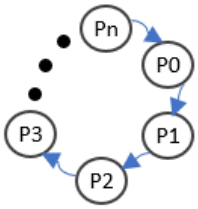
Necessary conditions for deadlock:

1. **Mutual Exclusion:** Only one process can use the resource at a time i.e. at least one resource must be in non-sharable mode. If any other process requests the resource and the resource is not available at that time, then the process must wait until the resource has been released.

2. **Hold and Wait:** Process must be holding at least one resource and requests for additional resources that are currently being held by other processes.

3. **No preemption:** Process cannot release its resources until it gets finished.

4. **Circular wait:** A set of processes waiting for each other in circular manner i.e. a set of waiting processes {p0, p1, p2 … pn} is exists in such a way that p0 is waiting for a resource held by p1, p1 is waiting for a resource held by p2, pn is waiting for a resource help by p0.



Strategies for deadlock handling:

We can handle deadlock in one of three ways:

1. **Deadlock ignorance**: we can ignore the problem and pretend like deadlock never occur in the system. This situation is used by most operating system like Windows.

2. **Deadlock prevention**: we can prevent the deadlock by ensuring that system will never enter in deadlock state. It provides various methods for ensuring that at least one of the necessary conditions for deadlock cannot hold.

3. **Deadlock avoidance**: we can avoid the deadlock by ensuring that system will never enter in deadlock state. It requires that operating system will be provided with some additional information like which resource a process will request or how resources are to be requested.

4. **Deadlock detection and recovery**: we can allow the system to enter in deadlock state, detect it and recover it.

Learn more about deadlock: https://www.geeksforgeeks.org/operating-system-process-management-deadlock-introduction/
Starvation and Deadlock: https://www.geeksforgeeks.org/deadlock-starvation-and-livelock/

1. **Deadlock prevention**:

   We can prevent the occurrence of a deadlock by ensuring that at least one of the necessary conditions for deadlock cannot hold.

a. **Mutual exclusion**: If a resource could have been used by more than one process at the same time (resource sharing between multiple process at the same time) then process never enter in waiting state.

b. **Hold and wait**: to ensure that the hold and wait condition never occur in the system then we must ensure that whenever a process requests a resource, it doesn't hold any other resources.
   Two protocols:
   Protocol 1: allocate resources to the process before it begins execution.
   Protocol 2: allow process to request to resources only when it is not holding any resource. Whenever a process holding some resources and requesting for additional resource so before requesting it must release all its resources.

   Both protocols have some disadvantages like –
   *Resource utilization may be low as resources may be allocated but unused for a long time.
   *Starvation may be possible.
   *Reduced throughput.

c. **No preemption**: if a process requests for some resources, we first check whether they are available or not. If they are, then allocate them to requesting process otherwise we have to check whether they are held by some other process that is waiting for other resource then we preempt the desired resources from this waiting process and allocate them to requesting process.

d. **Circular wait**: in this, we impose a total ordering of all resource types and a process can request the resources in increasing manner.

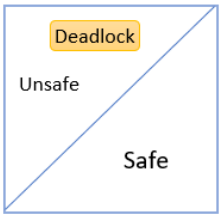
Condition	Approach
Mutual Exclusion	Spool everything
Hold and Wait	Request all resources initially
No Preemption	Take resources away
Circular Wait	Order resources numerically.

2. **Deadlock Avoidance:**
   In order to avoid deadlock, an operating system will be provided with some additional information like which resource a process will request or how resources are to be requested.

The basic idea to avoid deadlock is to allocate resources only if the resulting state is a safe state. If unsafe states are avoided, then deadlock is avoided as well.

Safe state: is that state in which processes are run in sequence or we can say that the operating system can allocate resources to each requesting process in some order and still avoid deadlock. Whenever the system have safe sequence then system always in a safe state.

Unsafe state: is that state in which multiple resources are running and requesting resources that may leads to deadlock.



Note: A safe state is not a deadlock state. A deadlocked state always an unsafe state but not all unsafe states are deadlocks. An unsafe state may lead to deadlock.

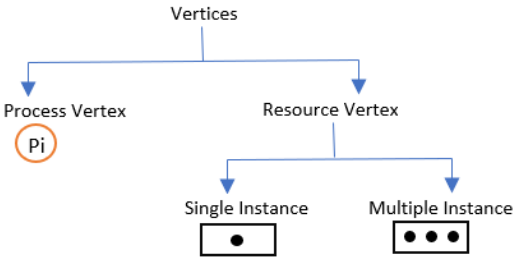
In order to avoid deadlock we can use banker's algorithm:

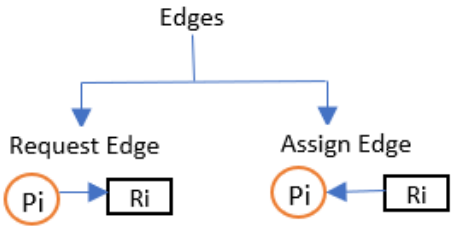
Banker's algorithm: it is the resource allocation and avoidance algorithm. Whenever process requests for resources, it always checks for the safe state. If after granting all requests for resources, it ensures the system remain in safe state.

Learn more about deadlock prevention and avoidance: https://www.geeksforgeeks.org/deadlock-prevention/

Resource Allocation Graph Algorithm (RAG):

It is the graphical representation of the system's state. This shows complete information about all the instances of resources whether they are available or being used by which process.





This graph contains set of vertices V and set of edges E.

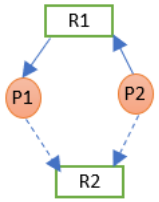
Set of vertices can be divided into two different types –
P = {p0, p1, …., pn}, set of all active vertices.
R = {r0, r1, …., rn}, set of all resource types.

Set of edges can be divided into various types-

Request Edge: A directed edge from process to resource which is denoted as Pi → Rj that shows process Pi requested for an instance of resource type Rj.

Assign Edge: A directed edge from resource to process which is denoted as Rj → Pi that shows an instance of resource type Rj has been allocated to the process Pi.

Claim Edge: A directed edge from process to resource which is denoted as Pi --> Rj that indicates that the process Pi may request resource Rj at some time in the future. This edge resembles a request edge in direction but is represented by dashed line.



Learn more about RAG with the help of example: https://www.geeksforgeeks.org/operating-system-resource-allocation-graph-rag/

Banker's Algorithm:

Resource allocation graph is not applicable to the system with multiple instances of each resource type. So, to overcome from this limitation we will use banker's algorithm. It is the resource allocation and avoidance algorithm. Whenever process requests for resources, it always checks for the safe state. If after granting all requests for resources, it ensures the system remain in safe state.
Whenever the new process enters the system, it must declare the maximum number (not greater than the total number of resources in the system) of instances of resources that it may need.

In order to implement banker's algorithm, we need to maintain several data structures. These data structure encode the state of state of resource allocation system.
Let n = number of processes in the system.
m = number of resource types.

Following data structures need to be maintain:

1. Available:
   Available[j] = k (k instances of resource type Rj are available).
2. Max:
   Max[i][j] = k (which means that the process Pi may request at most k instances of resource type Rj).
3. Allocation:
   Allocation[i][j] = k (which means that the process Pi is allocated k instances of resource type Rj).
4. Need:
   Need[i][j] = k (which means process Pi may need k more instances of resource type Rj to complete its task).
   Need[i][j] = Max[i][j] – Allocation[i][j].

Safety Algorithm:

This algorithm is used to find out whether the system is in safe state or not. This can be described as:

1. Let Work and Finish be vectors of length m and n
   Initialize, Work = Available and Finish[i]= false.
2. Find an i such that both
   a. Finish[i] == false
   b. Need <= Work
   If no such exists then go to step 4.

3. Work += Allocation.
4. Finish[i] = true.
   Go to step 2.
5. If Finish[i] == true for all i, then the system is in safe state.

Please refer these links in the given sequence:

1. Introduction and example: https://www.geeksforgeeks.org/operating-system-bankers-algorithm/
2. Methods of resource allocation: https://www.geeksforgeeks.org/methods-of-resource-allocation-to-processes-by-operating-system/
3. Program: https://www.geeksforgeeks.org/program-bankers-algorithm-set-1-safety-algorithm/
4. Numerical: Print all safe state: https://www.geeksforgeeks.org/operating-system-bankers-algorithm-print-safe-state-safe-sequences/

Example:
Consider a system with 5 processes <P0, P1, P2, P3, P4> and 3 resource types A B C (10, 5, 7) .
The following snapshot of the system has been given:

	Allocation			**Max**			**Available**		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

a. Find Need Matrix
b. Is the system is in safe state. If yes, then find the safe sequence.

i. Need Matrix = Max – Allocation

	A	**B**	**C**
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

ii. For safe sequence we use safety algorithm –
If Need <= Available, then execute process and New Available = Available + Allocation
Else do not execute go forward.

P0 = Need <= Available
7 4 3 <= 3 3 2 ---- false then this process is not executed and go forward

P1 = Need <= Available

1 2 2 <= 3 3 2 ---- true then execute the process
New Available = Available + Allocation
New Available = 3 3 2 + 2 0 0
New Available = 5 3 2

P2 = Need <= Available
6 0 0 <= 5 3 2 ---- false then process is not executed and go forward

P3 = Need <= Available
0 1 1 <= 5 3 2 ---- true then execute the process
New Available = Available + Allocation
New Available = 5 3 2 + 2 1 1
New Available = 7 4 3

P4 = Need <= Available
4 3 1 <= 7 4 3 ---- true then execute the process
New Available = Available + Allocation
New Available = 7 4 3 + 0 0 2
New Available = 7 4 5

Till now, not executed process are – P0 and P2

P0 = Need <= Available
7 4 3 <= 7 4 5 ---- true then execute the process
New Available = Available + Allocation
New Available = 7 4 5 + 0 1 0
New Available = 7 5 5

P2 = Need <= Available
6 0 0 <= 7 5 5 ---- true then execute the process
New Available = Available + Allocation
New Available = 7 5 5 + 3 0 2
New Available = 10 5 7 (this is same as available in system as given in question)

So, safe state sequence = <P1, P3, P4, P0, P2>

Deadlock Detection:

Before starting this, first read these links:

1. Deadlock detection algorithm: https://www.geeksforgeeks.org/operating-system-deadlock-detection-algorithm/
2. Deadlock detection and recover: https://www.geeksforgeeks.org/deadlock-detection-recovery/

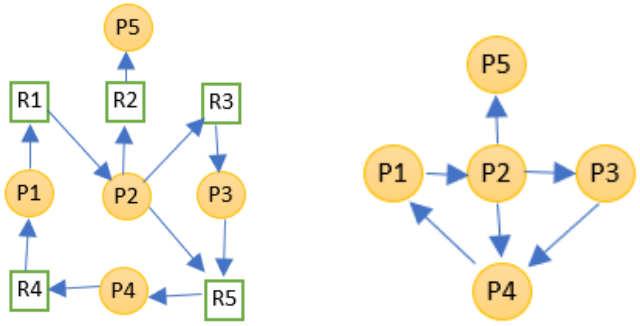
If the system doesn't employ either the deadlock prevention or deadlock avoidance, then the system may enter in the deadlock. In this scenario we must detect the deadlock and try to recover from that.

Deadlock detection:

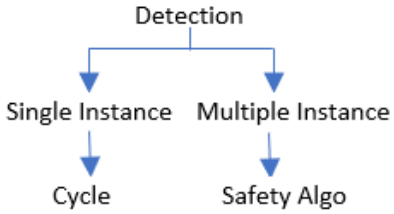
Deadlock detection algorithm is used to determine that deadlock exists.

1. **Single instance of each resource type**:

If each resource type has only single instance, then we can use Wait-For Graph which is a variant of resource-allocation graph.
We can get wait-for graph from resource-allocation graph by removing the resource nodes and collapsing the appropriate edge.
Deadlock may exist if wait for graph contains cycle.



2. **Multiple Instances of a resource type:**
The wait for graph is not applicable to the system having multiple instances of a resource type. so for this , it uses safety algorithm.



Recovery from Deadlock:

There are two methods to recover from deadlock are:

1. **Process termination:**
   a. Kill all deadlocked processes: this is more expensive as in this we break all the deadlocked cycles.
   b. Kill one process at a time until the deadlocked cycle is removed: there is an overhead as whenever the process is killed then deadlock detection algorithm must be invoked to determine the whether any process is still deadlocked.
2. **Resource preemption**:
   a. Preemption: in this we determine the which process and which processes are to be preempted. In this we preempt the resource to recover from deadlock.
   b. Rollback: in this we rollback the process to some safe state and restart it from that state.