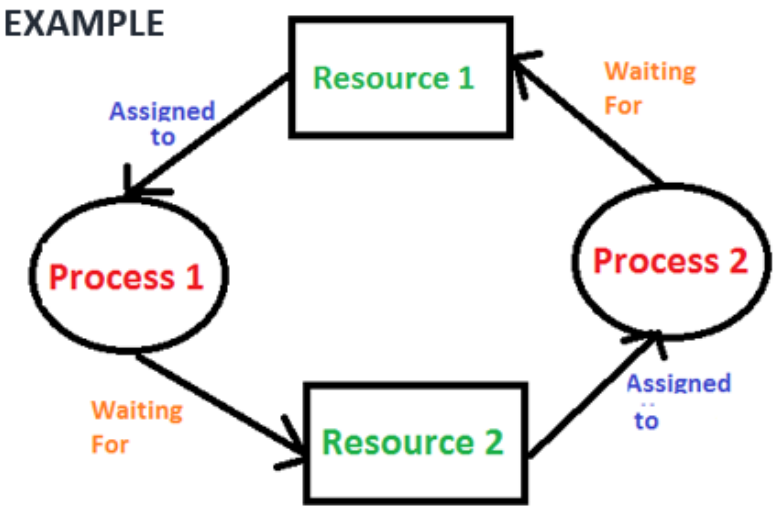
1. Introduction of Deadlock in Operating System

A 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.

System Model

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

1. *Requests a resource*
2. *Use the resource*
3. *Releases the resource*



2. Conditions OF deadlock

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

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

Conditions OF deadlock

1. Mutual Exclusion

By this condition,

- There must exist at least one resource in the system which can be used by only one process at a time.
- If there exists no such resource, then deadlock will never occur.
- Printer is an example of a resource that can be used by only one process at a time.

2. Hold and Wait-

By this condition,

- There must exist a process which holds some resource and waits for another resource held by some other process.

3. No Preemption-

By this condition,

- Once the resource has been allocated to the process, it can not be preempted.
- It means resource can not be snatched forcefully from one process and given to the other process.
- The process must release the resource voluntarily by itself.

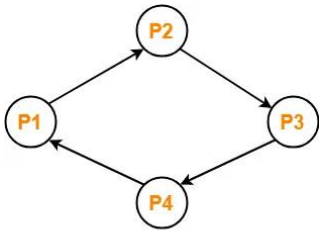
4. Circular Wait-

By this condition,

- All the processes must wait for the resource in a cyclic manner where the last process waits for the resource held by the first process.

Here,

- Process P1 waits for a resource held by process P2.
- Process P2 waits for a resource held by process P3.
- Process P3 waits for a resource held by process P4.
- Process P4 waits for a resource held by process P1.



Circular Wait

I. Important Note-

- ***All these 4 conditions must hold simultaneously for the occurrence of deadlock.***
- ***If any of these conditions fail, then the system can be ensured deadlock free.***

3. The various strategies for handling deadlock are-

1. ***Deadlock Prevention***
2. ***Deadlock Avoidance***
3. ***Deadlock Detection and Recovery***
4. ***Deadlock Ignorance***

1. Deadlock Prevention-

- This strategy involves designing a system that
- violates one of the four necessary conditions required for the occurrence of deadlock.
- This ensures that the system remains free from the deadlock.

The various conditions of deadlock occurrence may be violated as-

II. Mutual Exclusion-

- To violate this condition, all the system resources must be such that they can be used in a shareable mode.
- In a system, there are always some resources which are mutually exclusive by nature.
- So, this condition can not be violated.

III. Hold and Wait-

This condition can be violated in the following ways-

I. *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 a printer at a later time and we have allocated a printer before the start of its execution printer will remain blocked till it has completed its execution.*

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

IV. No Preemption-

- *This condition can by violated by forceful preemption.*
- *Consider a process is holding some resources and request other resources that can not be immediately allocated to it.*
- *Then, by forcefully preempting the currently held resources, the condition can be violated.*

A process is allowed to forcefully preempt the resources possessed by some other process only if-

1. *It is a high priority process or a system process.*
2. *The victim process is in the waiting state.*

V. Circular Wait-

- *This condition can be violated by not allowing the processes to wait for resources in a cyclic manner.*

To violate this condition, the following approach is followed-

I. *Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing. order of numbering.*

II. *For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.*

2. Deadlock Avoidance-

- This strategy involves maintaining a set of data using which a decision is made whether to entertain the new request or not.
- If entertaining the new request causes the system to move in an unsafe state, then it is discarded.
- This strategy requires that every process declares its maximum requirement of each resource type in the beginning.
- The main challenge with this approach is predicting the requirement of the processes before execution.
- **Banker's Algorithm** is an example of a deadlock avoidance strategy.

3. Deadlock Detection and Recovery-

- This strategy involves waiting until a deadlock occurs.
- After deadlock occurs, the system state is recovered.
- The main challenge with this approach is detecting the deadlock.

4. Deadlock Ignorance-

- This strategy involves ignoring the concept of deadlock and assuming as if it does not exist.
- This strategy helps to avoid the extra overhead of handling deadlock.
- Windows and Linux use this strategy and it is the most widely used method.
- It is also called as **Ostrich approach**.

2. Deadlock Avoidance-

- Resource-Allocation Graph Scheme
- Banker's Algorithm
  - intro
  - Data Structures
- Request Algorithm
- Safety Algorithm
- Examples

3. Deadlock Detection

I. Single Instance of Each Resource Type
   a. Resource-Allocation Graph and Wait-for Graph
II. Several Instances of a Resource Type
   a. Detection Algorithm
III. Recovery from Deadlock:
   a. Process Termination
   b. Resource Preemption

I. Single Instance of Each Resource Type

- If each resource category has a single instance, then we can use a variation of the resource-allocation graph known as a ***wait-for graph***.
- A wait-for graph can be constructed from a resource-allocation graph by eliminating the resources and collapsing the associated edges, as shown in the figure below.
- An arc from Pi to Pj in a wait-for graph indicates that process Pi is waiting for a resource that process Pj is currently holding.
- As before, cycles in the wait-for graph indicate deadlocks.
- This algorithm must maintain the wait-for graph, and periodically search it for cycles.

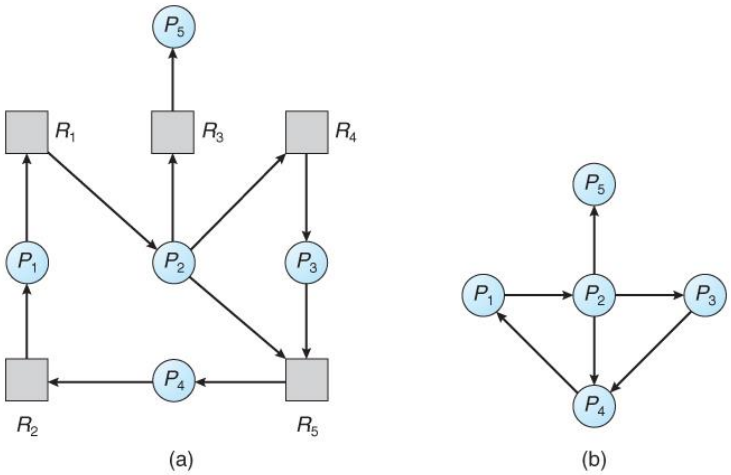


Figure 7.9 - (a) Resource allocation graph. (b) Corresponding wait-for graph

II. Several Instances of a Resource Type

- The detection algorithm outlined here is essentially the same as the Banker's algorithm, with two subtle differences:
- In step 1, the Banker's Algorithm sets Finish[i] to false for all i. The algorithm presented here sets Finish[i] to false only if Allocation[i] is not zero. If the currently allocated resources for this process are zero, the algorithm sets Finish[i] to true. This is essentially assuming that IF all of the other processes can finish, then this process can finish also. Furthermore, this algorithm is specifically looking for which processes are involved in a deadlock situation, and a process that does not have any resources allocated cannot be involved in a deadlock, and so can be removed from any further consideration.

III. Recovery From Deadlock

- There are three basic approaches to recovery from deadlock:
    1. Inform the system operator, and allow him/her to take manual intervention.
    2. Terminate one or more processes involved in the deadlock
    3. Preempt resources.

7.7.1 Process Termination

- Two basic approaches, both of which recover resources allocated to terminated processes:
    - Terminate all processes involved in the deadlock. This definitely solves the deadlock, but at the expense of terminating more processes than would be absolutely necessary.
    - Terminate processes one by one until the deadlock is broken. This is more conservative, but requires doing deadlock detection after each step.
- In the latter case there are many factors that can go into deciding which processes to terminate next:
    - Process priorities.
    - How long the process has been running, and how close it is to finishing.
    - How many and what type of resources is the process holding. (Are they easy to preempt and restore?)
    - How many more resources does the process need to complete.
    - How many processes will need to be terminated
    - Whether the process is interactive or batch.
    - (Whether or not the process has made non-restorable changes to any resource.)

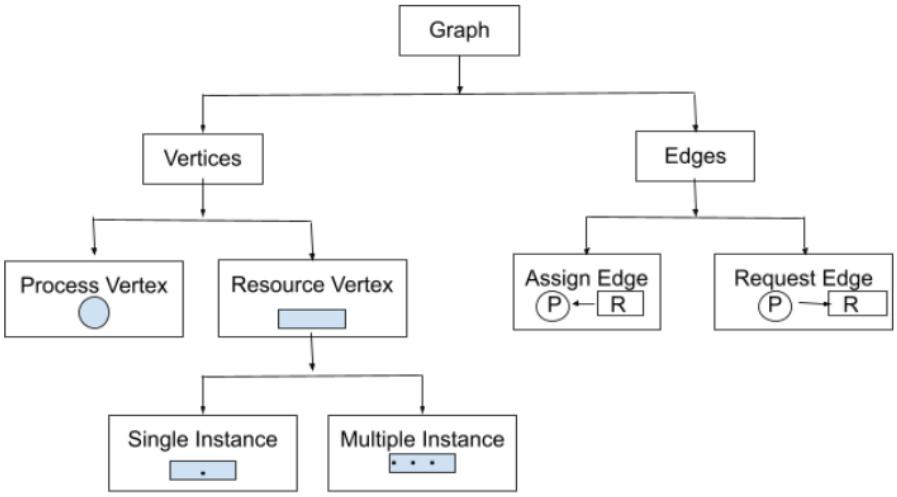
7.7.2 Resource Preemption

- When preempting resources to relieve deadlock, there are three important issues to be addressed:
    1. **Selecting a victim** - Deciding which resources to preempt from which processes involves many of the same decision criteria outlined above.
    2. **Rollback** - Ideally one would like to roll back a preempted process to a safe state prior to the point at which that resource was originally allocated to the process. Unfortunately it can be difficult or impossible to determine what such a safe state is, and so the only safe rollback is to roll back all the way back to the beginning. (I.e. abort the process and make it start over.)
    3. **Starvation** - How do you guarantee that a process won't starve because its resources are constantly being preempted? One option would be to use a priority system, and increase the priority of a process every time its resources get preempted. Eventually it should get a high enough priority that it won't get preempted any more.

4. Resource Allocation Graph in Operating System

1. *We use the resource allocation graph for the pictographic representation of the state of a system.*
2. ***The resource allocation graph contains all the information related to the processes that are** holding some resources and also waiting for some more resources.*
3. ***The resource allocation graph contains all the information related to all the instances of the resources** means the information about available resources and the resources which the process is being using.*

4. *In the Resource Allocation Graph, we use a circle to represent the process and rectangle to represent the resource.*



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.

5. Bankers algorithm

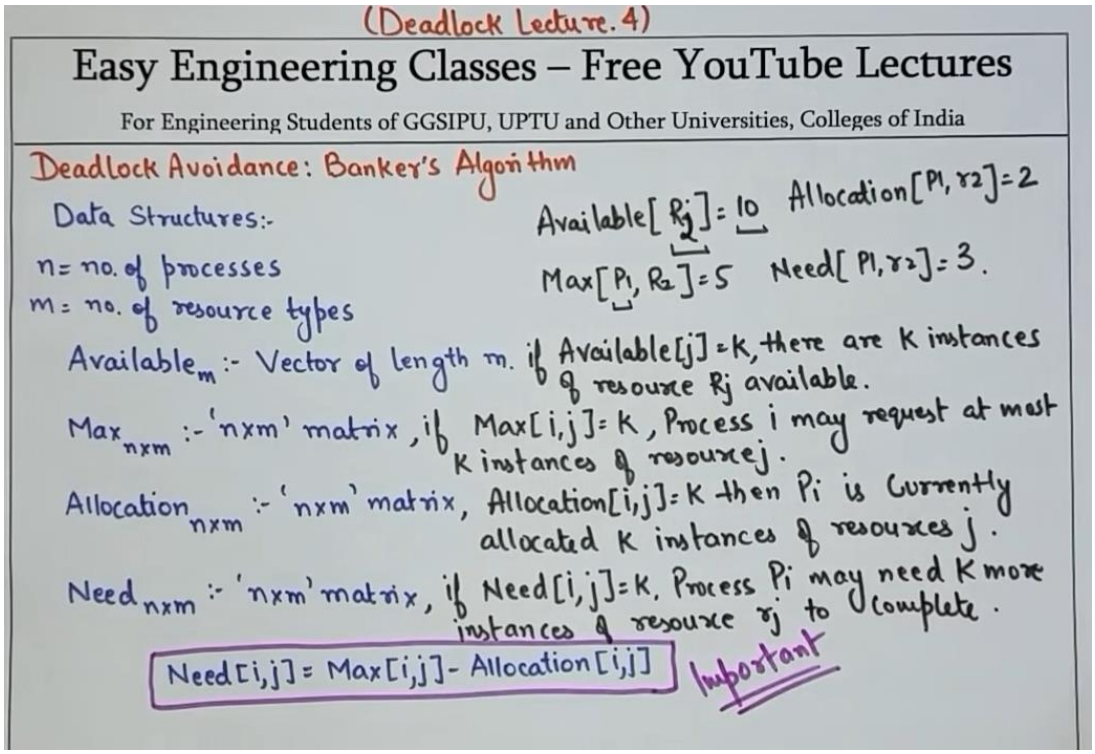
Bankers algorithm in OS is a deadlock avoidance algorithm and it is also used to safely allocate the resources to each process within the system.

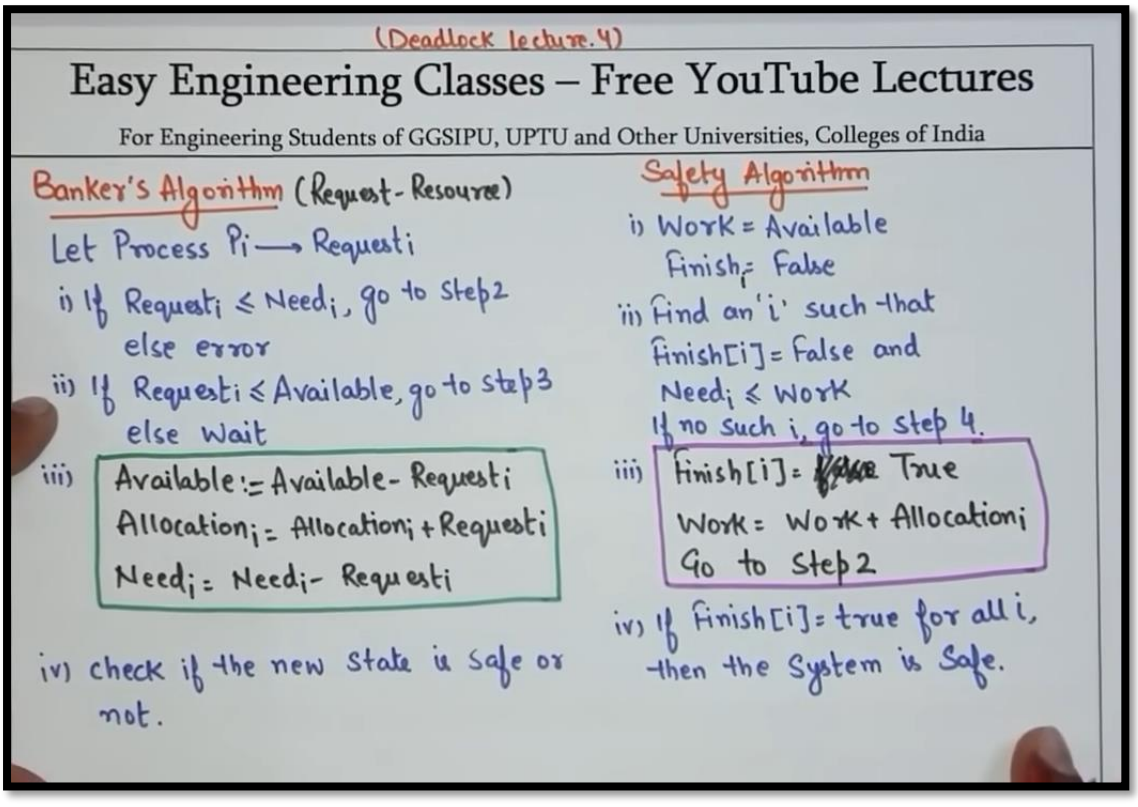
As the name suggests, Bankers algorithm in OS is mostly used in the banking systems to identify whether a loan should be given to a person or not

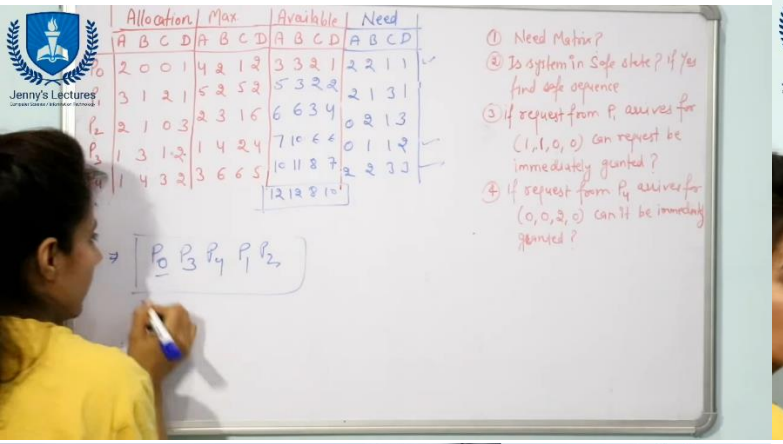
Similarly, it works in an operating system:

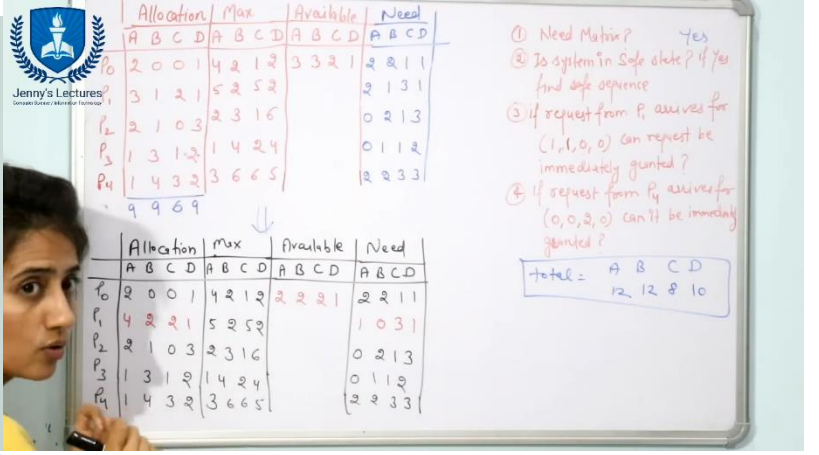
1. When a new process is created in a computer system,
2. the process must provide all types of information to the operating system
3. like upcoming processes, requests for their resources, counting them, and delays.
4. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system.
5. Therefore, it is also known as **deadlock avoidance algorithm** or **deadlock detection** in the operating system.

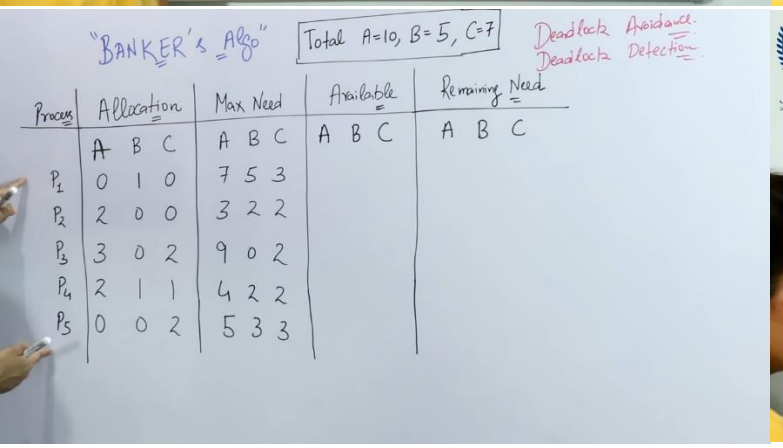
3. Data structures Required in the Bankers Algorithm Implementation

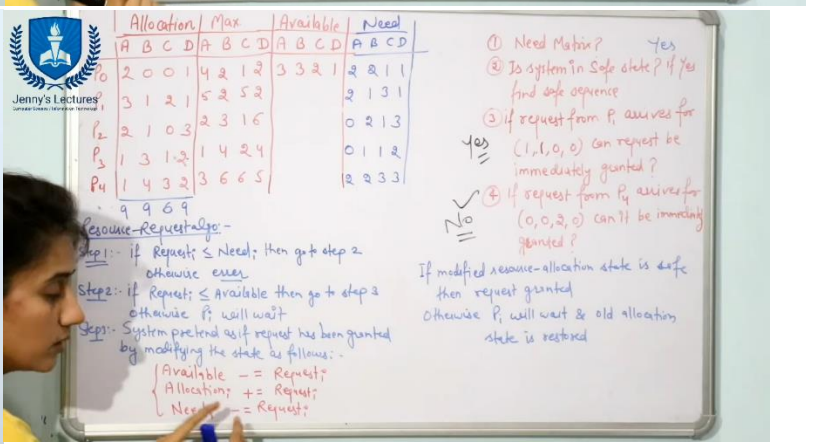





Jenny's Lectures
Allocation | Max | Available | Need
A B C D | A B C D | A B C D | A B C D
P0 2 0 0 1 | 4 2 1 2 | 3 3 2 1 | 2 2 1 1
P1 3 1 2 1 | 5 2 5 2 | 5 3 2 2 | 2 1 3 1
P2 2 1 0 3 | 2 3 1 6 | 6 6 3 4 | 0 2 1 3
P3 1 3 1 2 | 1 4 2 4 | 7 10 6 6 | 0 1 1 2
P4 1 4 3 2 | 3 6 6 5 | 10 11 8 7 | 2 2 3 3
12 12 8 10
⇒ P0 P3 P4 P1 P2
① Need Matrix?
② Is system in Safe state? If Yes find safe sequence
③ If request from P1 arrives for (1,1,0,0) can request be immediately granted?
④ If request from P4 arrives for (0,0,2,0) can't be immediately granted?


Jenny's Lectures
Allocation | Max | Available | Need
A B C D | A B C D | A B C D | A B C D
P0 2 0 0 1 | 4 2 1 2 | 3 3 2 1 | 2 2 1 1
P1 3 1 2 1 | 5 2 5 2 | 2 1 3 1
P2 2 1 0 3 | 2 3 1 6 | 0 2 1 3
P3 1 3 1 2 | 1 4 2 4 | 0 1 1 2
P4 1 4 3 2 | 3 6 6 5 | 2 2 3 3
9 9 6 9
Allocation | Max | Available | Need
A B C D | A B C D | A B C D | A B C D
P0 2 0 0 1 | 4 2 1 2 | 2 2 2 1 | 2 2 1 1
P1 4 2 2 1 | 5 2 5 2 | 1 0 3 1
P2 2 1 0 3 | 2 3 1 6 | 0 2 1 3
P3 1 3 1 2 | 1 4 2 4 | 0 1 1 2
P4 1 4 3 2 | 3 6 6 5 | 2 2 3 3
① Need Matrix? Yes
② Is system in Safe state? If Yes find safe sequence
③ If request from P1 arrives for (1,1,0,0) can request be immediately granted?
④ If request from P4 arrives for (0,0,2,0) can't be immediately granted?
Total = A B C D
12 12 8 10


"BANKER's Algo"
Total A=10, B=5, C=7
Deadlock Avoidance
Deadlock Detection
Process | Allocation | Max Need | Available | Remaining Need
A B C | A B C | A B C | A B C
P1 0 1 0 | 7 5 3
P2 2 0 0 | 3 2 2
P3 3 0 2 | 9 0 2
P4 2 1 1 | 4 2 2
P5 0 0 2 | 5 3 3


Jenny's Lectures
Allocation | Max | Available | Need
A B C D | A B C D | A B C D | A B C D
P0 2 0 0 1 | 4 2 1 2 | 3 3 2 1 | 2 2 1 1
P1 3 1 2 1 | 5 2 5 2 | 2 1 3 1
P2 2 1 0 3 | 2 3 1 6 | 0 2 1 3
P3 1 3 1 2 | 1 4 2 4 | 0 1 1 2
P4 1 4 3 2 | 3 6 6 5 | 2 2 3 3
9 9 6 9
Resource-Request algo:-
Step 1:- if Request_i ≤ Need_i then go to step 2 otherwise error
Step 2:- if Request_i ≤ Available then go to step 3 otherwise P_i will wait
Step 3:- System pretend as if request has been granted by modifying the state as follows:-
Available -= Request_i;
Allocation_i += Request_i;
Need_i -= Request_i;
① Need Matrix? Yes
② Is system in Safe state? If Yes find safe sequence
③ If request from P1 arrives for (1,1,0,0) can request be immediately granted? Yes
④ If request from P4 arrives for (0,0,2,0) can't be immediately granted? No
If modified resource-allocation state is safe then request granted
Otherwise P_i will wait & old allocation state is restored