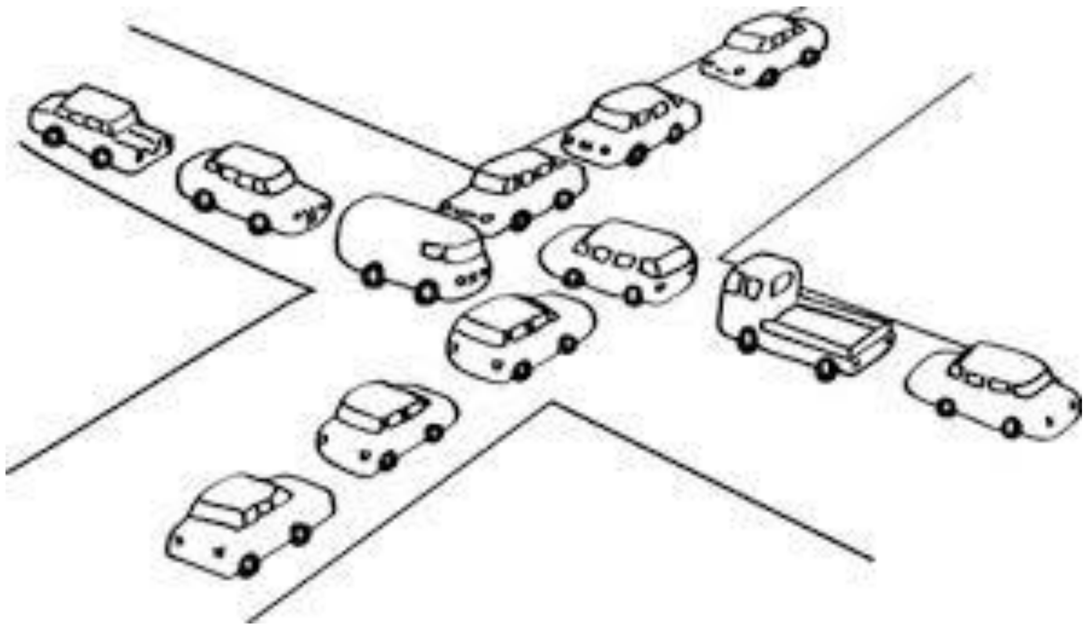
COS3023 Operating Systems and Concurrency

Topic 4- Input/Output : Deadlocks

Lecturer : Ms Sha

Learning Objective

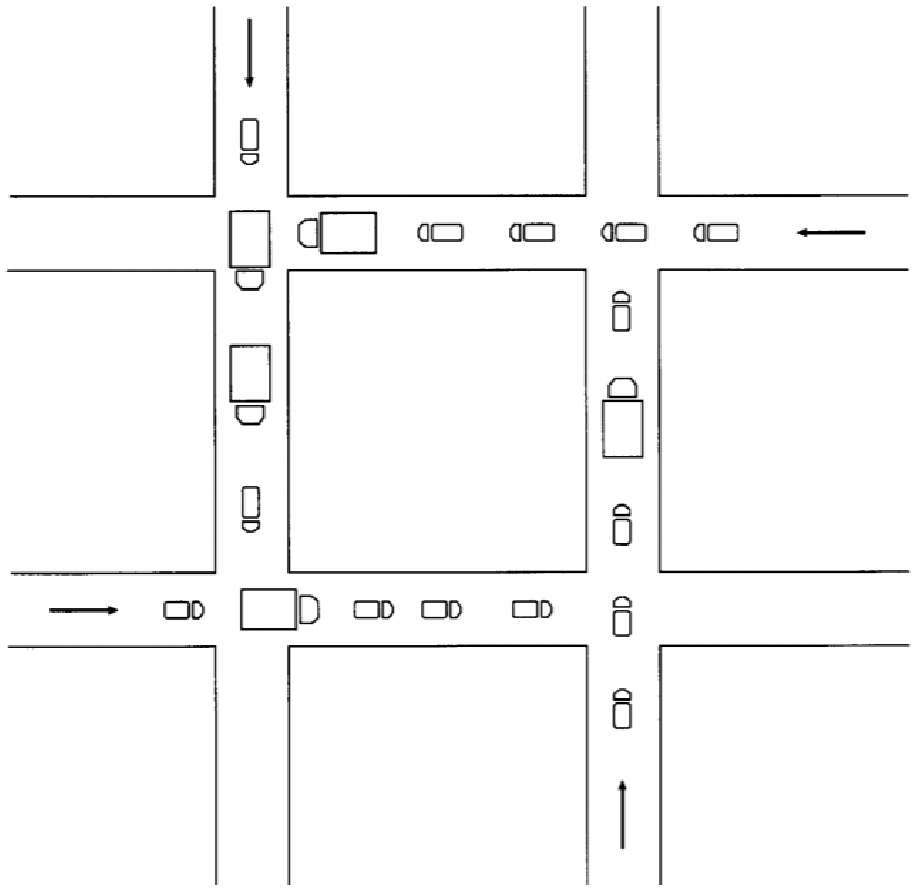
- To understand about the condition of deadlocks and methods to handle deadlocks.



Deadlocks

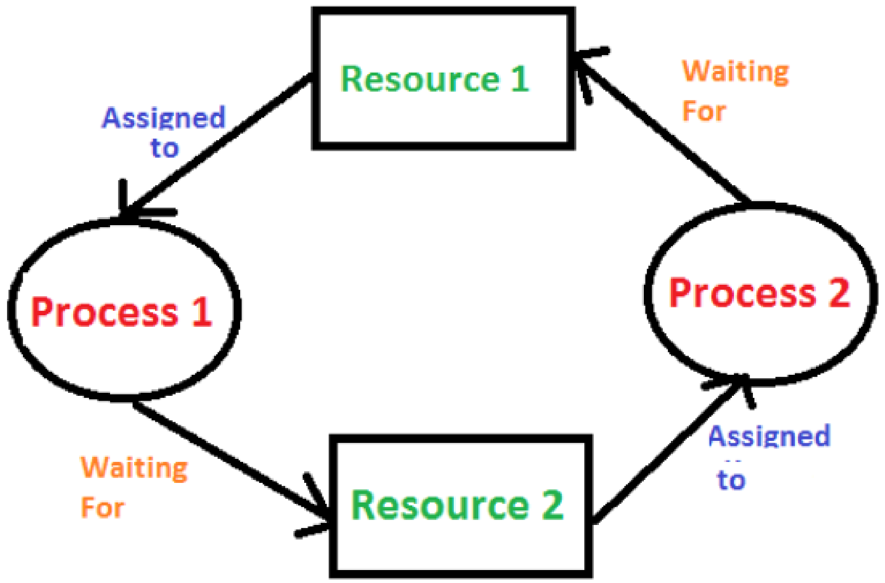
Deadlocks

- What are they?
- When do they occur?
- How to deal with them?

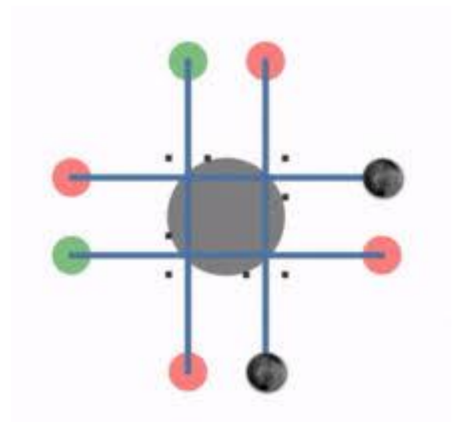


Deadlock

- Deadlock ➔ deadlock problem
- It is a situation when 2 or more processes sharing the same resource are effectively preventing each other from accessing the resources.



Conditions for Deadlock

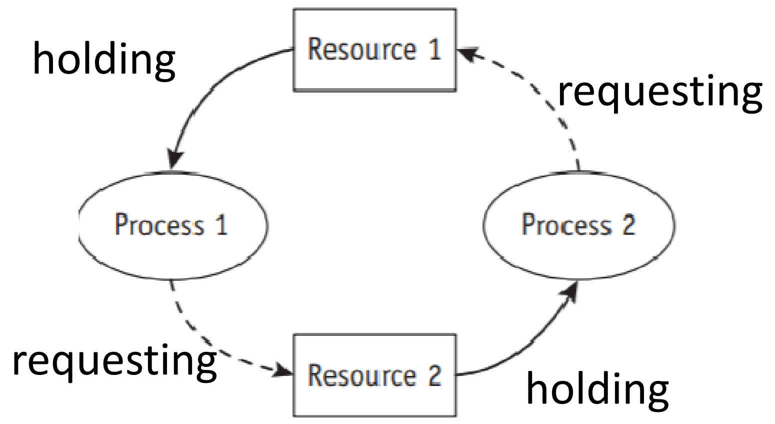


- Deadlock may occur when:
  1. Processes in Mutual exclusive
  2. Processes Hold & Wait
  3. No preemption
  4. Processes Circular Wait

Conditions for Deadlock

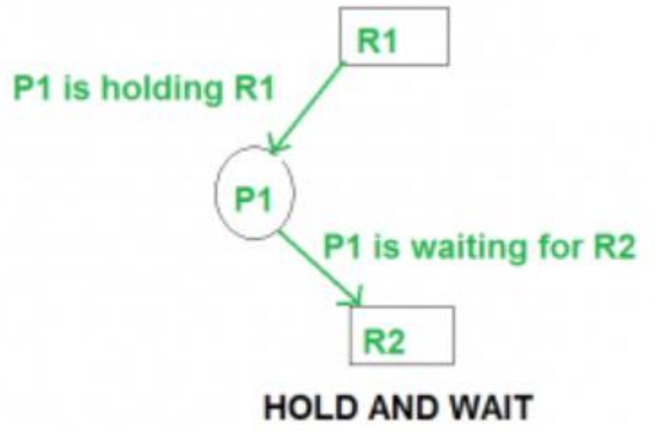
1.Processes in Mutual exclusive

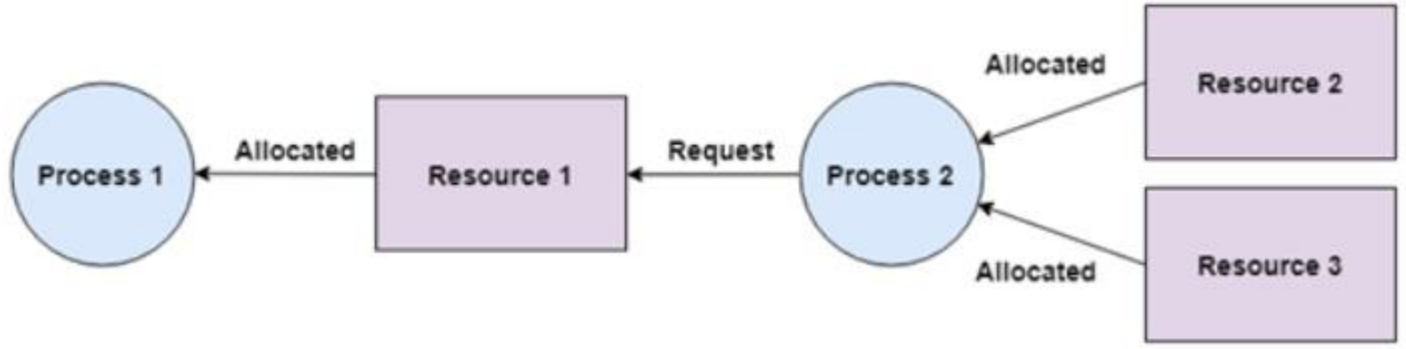
- No sharing/resources are not sharable
- Only one process can use resource at a time.



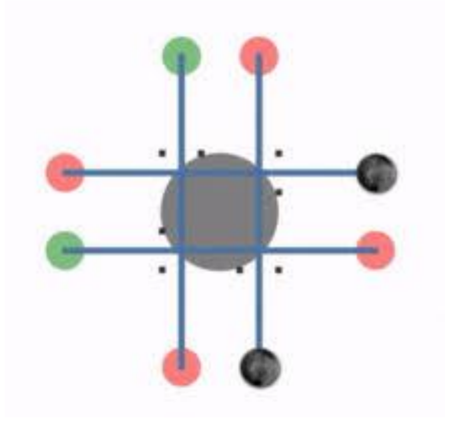
2. Processes Hold & Wait

- Holding one resource and waiting for others



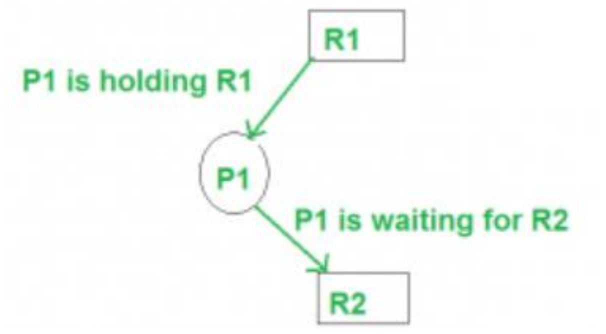


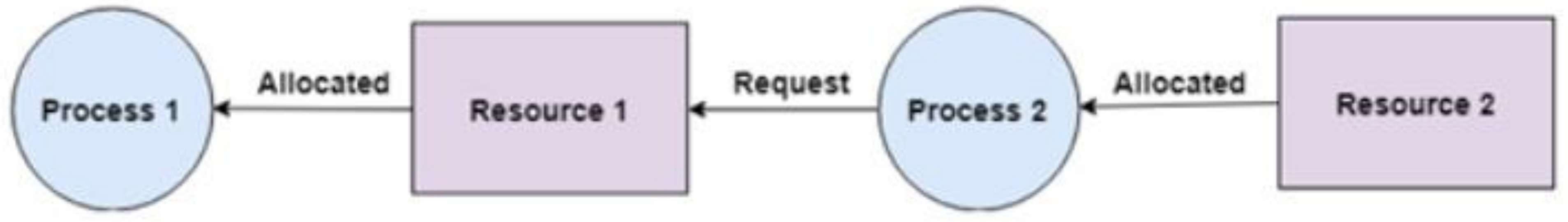
Conditions for Deadlock



3. No preemption

- A resource can be released voluntarily by process itself.
- P1 is holding R1, and at the same time requesting for R2. But, it does not releasing R1.



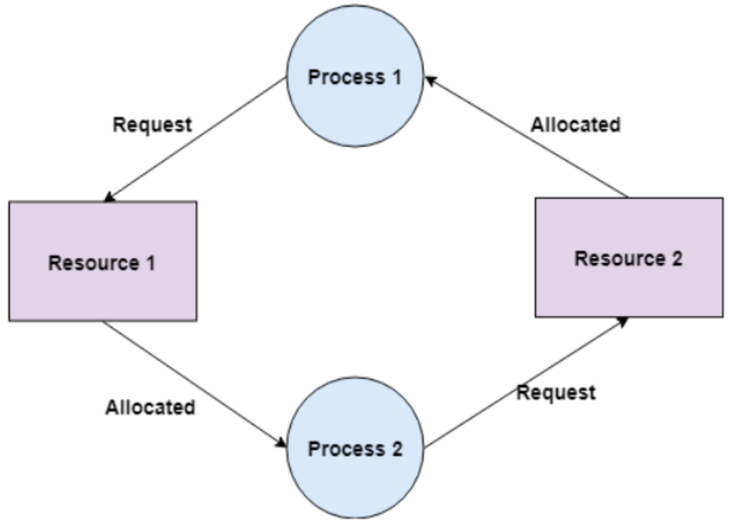


Conditions for Deadlock

4. Processes Circular Wait

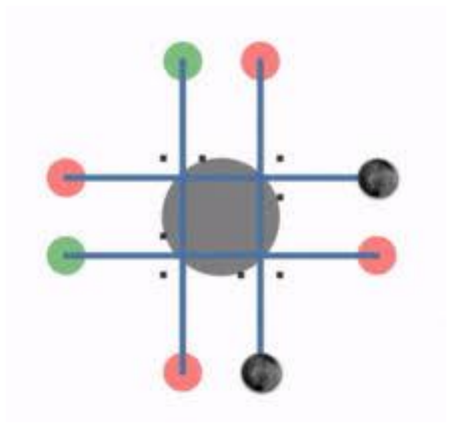
4 processes (A B C D)

each process involved in the impasse(no progress/deadlock) is waiting for another to voluntarily release the resource, so that at least one will be able to continue on and eventually arrive at the destination.



Conditions for Deadlock

All four conditions are required for the deadlock to occur, and as long as all four conditions are present the deadlock will continue; but if one condition can be removed, the deadlock will be resolved



Methods to handle deadlock

1) Deadlock ignorance

Let it happens and reboot the system

2) Deadlock prevention .

The idea is to not let the system into deadlock state.

3) Deadlock avoidance.

ensure that a system will never enter an unsafe state

4) Deadlock detection and recovery: Let deadlock occur, then do preemption to handle it once occurred.

1.Deadlock Ignorance

If deadlock is very rare, then let it happen and reboot the system, and the problem is rectify.

2. Deadlock Prevention

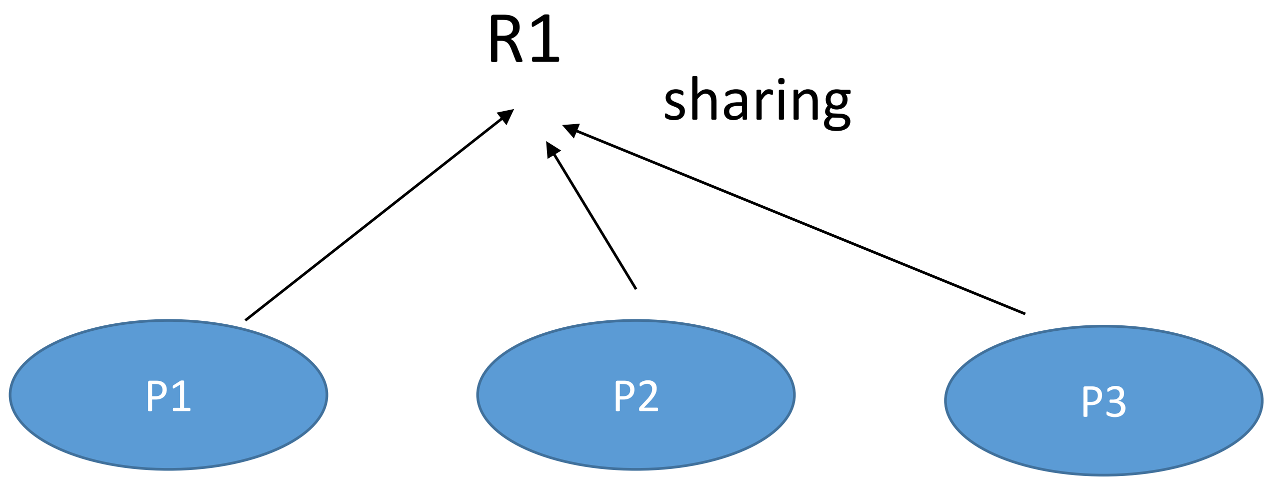
- Try to finds solution before deadlock occurs.
- Deadlock prevention says;
  - try to remove or make false all conditions
  - At least try to remove or make false any one of the condition.

How to make the conditions false??

2. Deadlock Prevention

- To prevent deadlock:

1. Make mutual exclusion false, ➔ just share the resources.



But, no possible for some cases. Eg: printer is not shareable.

So this technique may not suitable for some cases.

2. Deadlock Prevention

- To prevent deadlock:

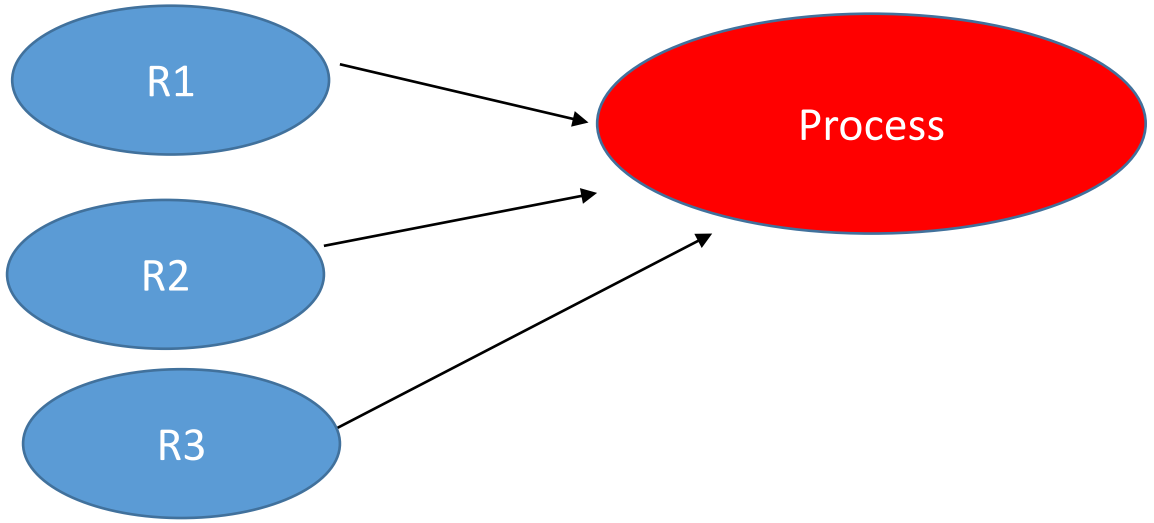
2. No preemption ➔ make the no preemption false

2. Deadlock Prevention

- To prevent deadlock:

3. Hold & Wait ➔ try to do no hold & wait, then deadlock can prevented.

How? Try to give all resources to a process before it starts.



2. Deadlock Prevention

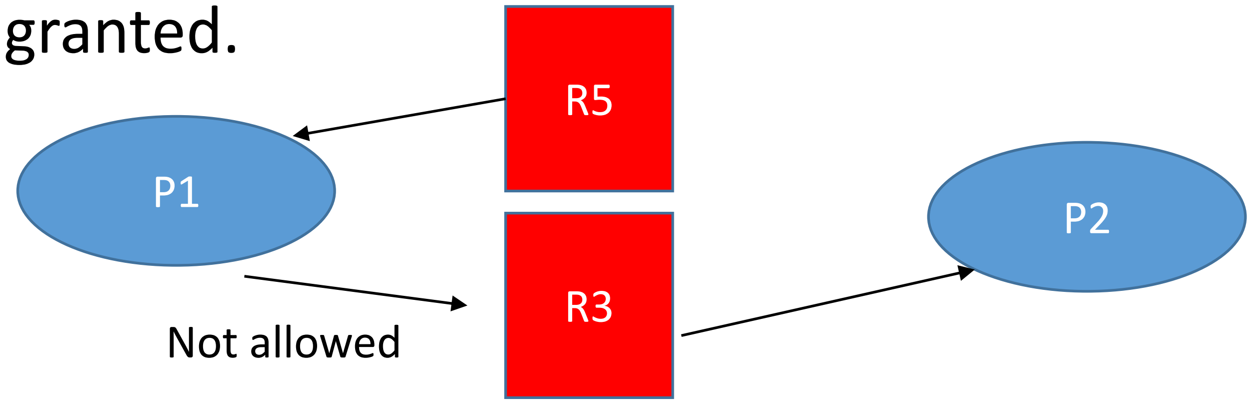
- To prevent deadlock:

4. Circular wait. ➔ try to make circular wait false.

How?

- To remove circular wait, just give the numbering to all resources.
- A process can request for the resources only in increasing order of numbering.

Eg: 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 Prevention

3. Deadlock avoidance

- For avoiding deadlocks is to require additional information about how resources are to be requested.
- Each request requires the following:

1) The resource currently available.
2) The resource currently allocated to each process.
3) The future requests and releases of each process.

- The above information is used to decide whether the current request can be satisfied or must wait to avoid a possible future deadlock.
- A deadlock-avoidance algorithm dynamically examines the **resource-allocation state** to ensure that a circular wait condition can never exist. The resource-allocation state is defined by the number of available and allocated resources and the maximum demands of the processes.

3. Deadlock avoidance

Basic facts is :

1) If a system is in safe state , then there is no deadlock

A **state** is safe if the system can allocate **all resources** requested by all processes.

1) If a system is in unsafe state, possibly deadlock may occur.

3. Deadlock avoidance

Avoidance : ensure that a system will never enter an unsafe state.

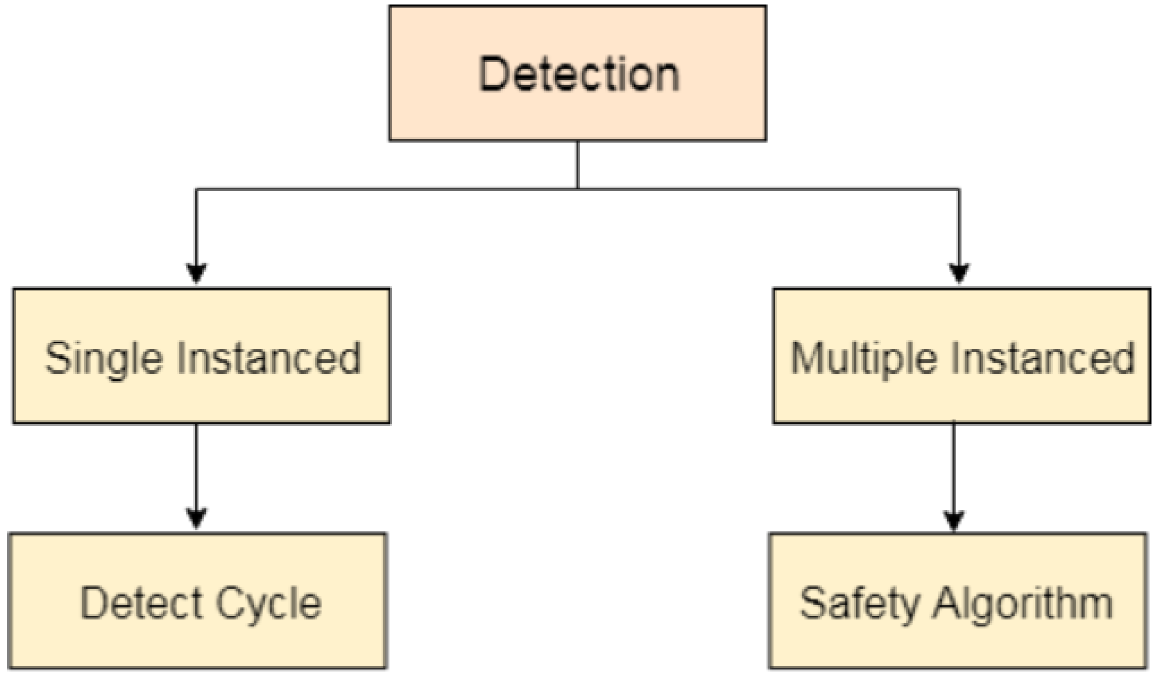
4. Deadlock detection and recovery

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.

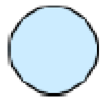
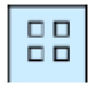
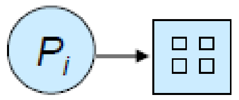
Deadlock detection and recovery

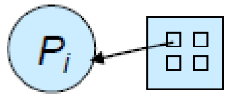
- The OS can detect the deadlocks with the help of **Resource allocation graph.**



Deadlock detection and recovery

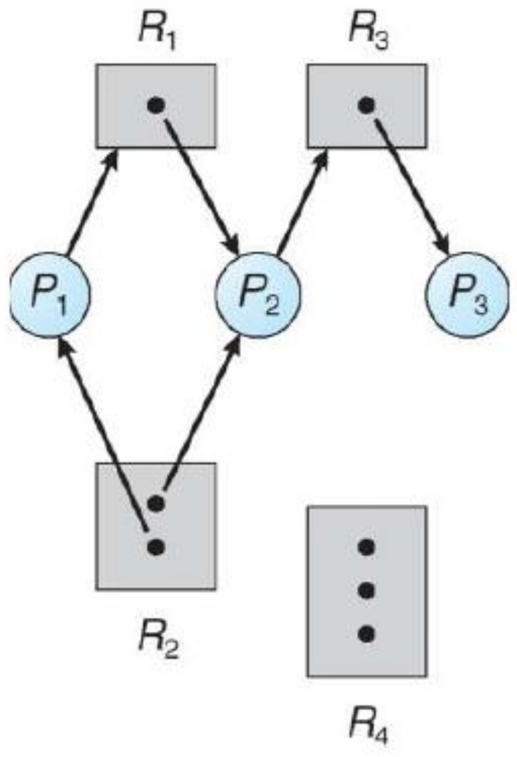
Resource allocation graph

- Process 
- Resource Type with 4 instances 
- P_i requests instance of R_j 

- P_i is holding an instance of R_j 

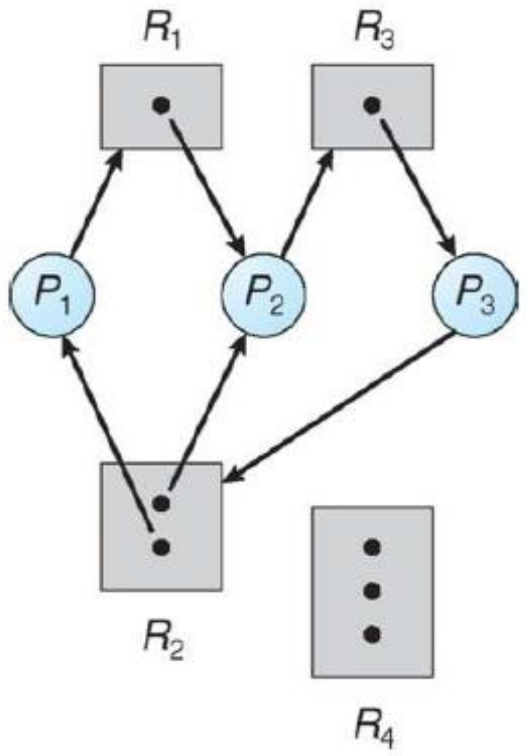
Deadlock detection and recovery

Example for Resource allocation graph



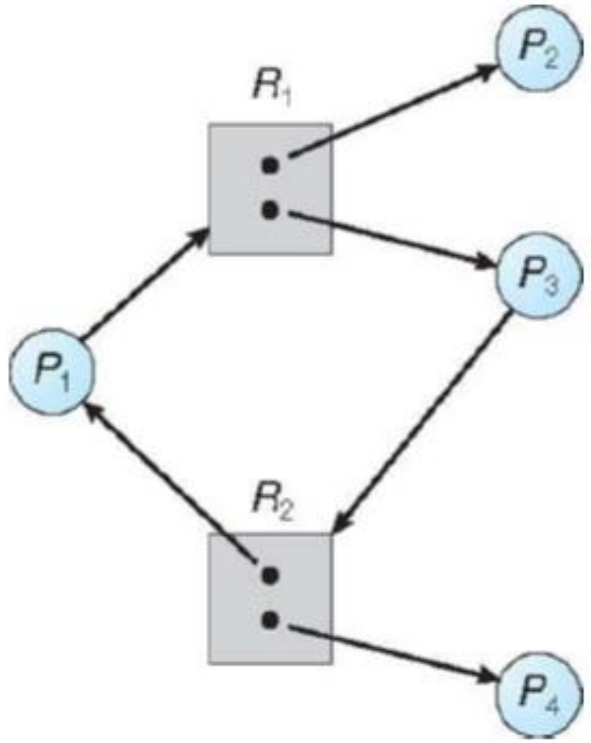
Deadlock detection and recovery

Resource allocation graph with deadlock



Deadlock detection and recovery

Graph With A Cycle



Deadlock detection and recovery

Basic Facts

- If graph contains no cycles $\Rightarrow$ no deadlock
- If graph contains a cycle $\Rightarrow$
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

4. Deadlock detection and recovery

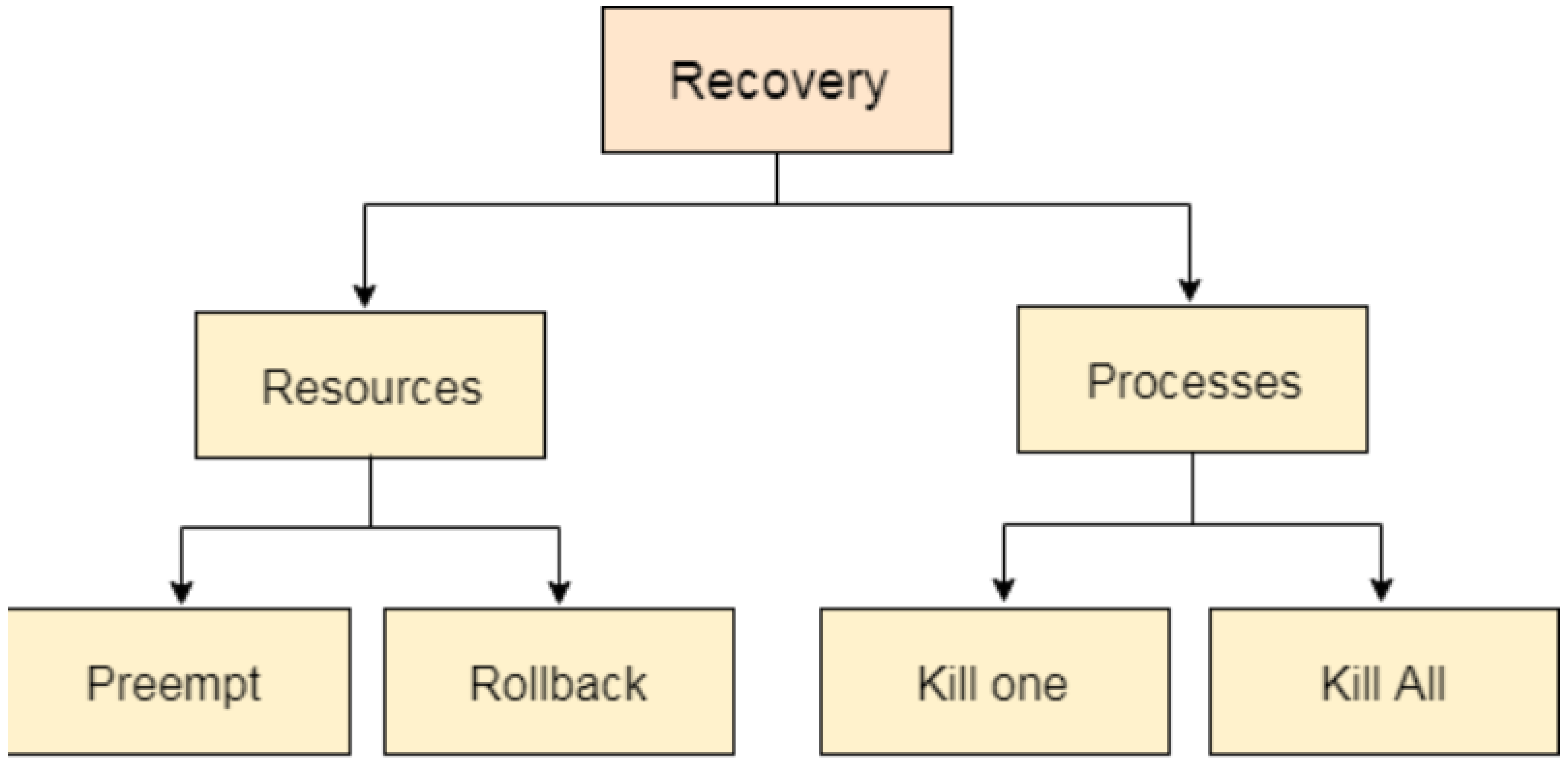
In single instanced resource types, if a cycle is being formed in the system then there will definitely be a deadlock.

in multiple instanced resource type graph, detecting a cycle is not just enough.

4. Deadlock detection and recovery

In order to recover the system from deadlocks, either OS considers resources or processes.

4. Deadlock detection and recovery



4. Deadlock detection and recovery

- **For Resource**

Preempt the resource

We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.

Rollback to a safe state

System passes through various states to get into the deadlock state. The operating system can rollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

- The moment, we get into deadlock, we will rollback all the allocations to get into the previous safe state.

4. Deadlock detection and recovery

- **For Process**

Kill a process

Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

Kill all process

This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.

The Banker’s Algorithm

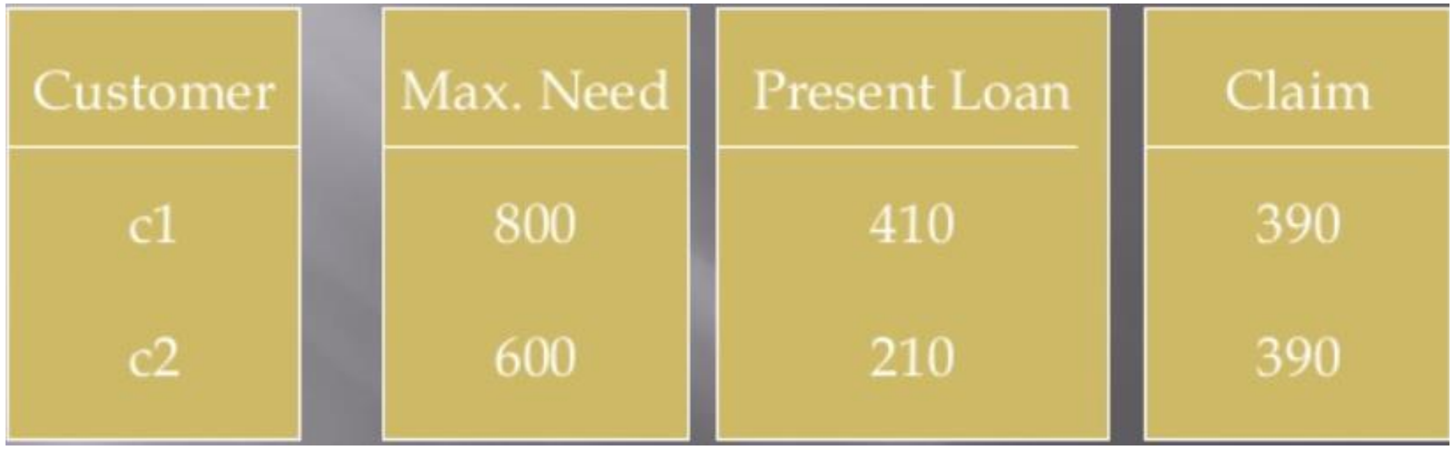
- Banker’s algorithm is a deadlock avoidance algorithm.

- The Banker's Algorithm gets its name because it is a method that bankers could use to assure that when they lend out resources they will still be able to satisfy all their clients. (A banker won't loan out a little money to start building a house unless they are assured that they will later be able to loan out the rest of the money to finish the house.)

- When a process starts up, it must state in advance the maximum allocation of resources it may request, up to the amount available on the system.

- When a request is made, the scheduler determines whether granting the request would leave the system in a safe state. If not, then the process must wait until the request can be granted safely.

The Banker's problem



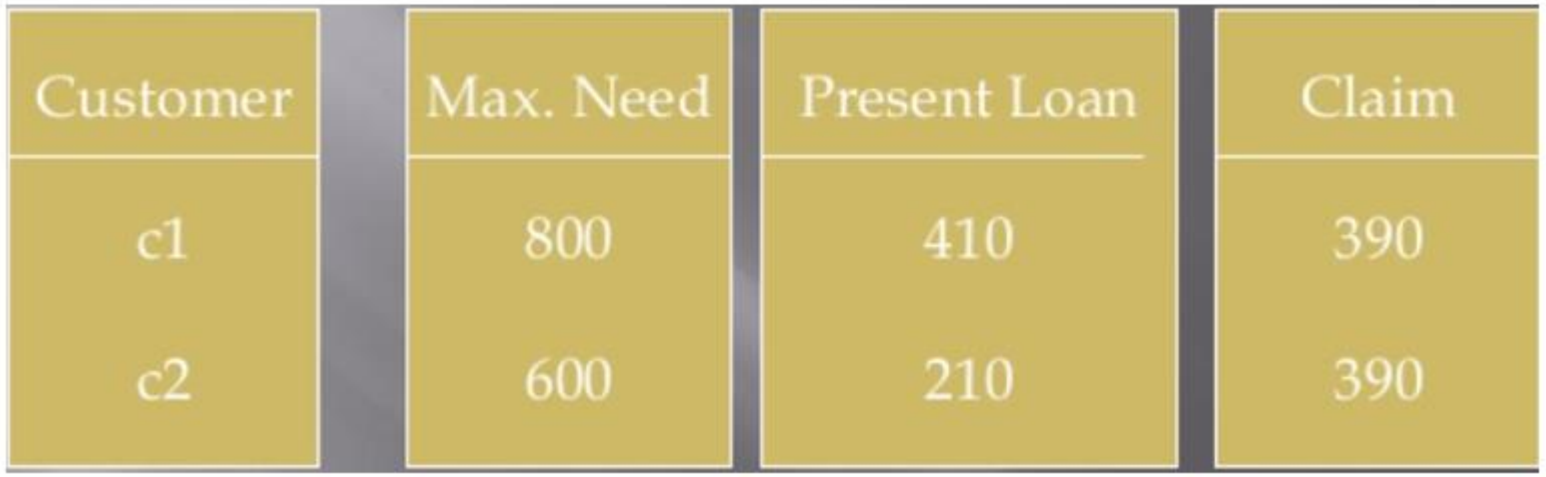
Customer	Max. Need	Present Loan	Claim
c1	800	410	390
c2	600	210	390

- Suppose total bank capital is RM1000
- Current cash: RM1000 – (RM410 + RM210) = RM380

The Banker’s algorithm

Definitions:

- ❑ Each process has a LOAN,CLAIM, MAXIMUM NEED
  - o LOAN: current number of resources held
  - o MAXIMUM NEED: total number resources needed to complete
  - o CLAIM: = (MAXIMUM – LOAN)



Customer	Max. Need	Present Loan	Claim
c1	800	410	390
c2	600	210	390

The Banker’s algorithm

Assumptions:

- ❑ Establish a LOAN ceiling (MAXIMUM NEED) for each process

  MAXIMUM NEED < total number of resources available

- ❑ Total loans for a process must be less than or equal to MAXIMUM NEED

- ❑ Loaned resources must be returned back in finite time

The Banker's algorithm

Example:

Considering a system with five processes P_0 through P_4 and three resources of type A, B, C (A-Printer, B-CD ROM, C- Hard drive). 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:

Current available(work) = Total – total Recource allocated
= 10 - 7
= 3

Process	Allocation			Max			Available **(Work)**		
	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			

Total Resource allocation: 7 2 5

Q1: What will be the content of the Need Matrix?

Need [i, j] = Max [i, j] – Allocation [i, j]

Example:

Need[P_o, A] = 7 - 0 = 7

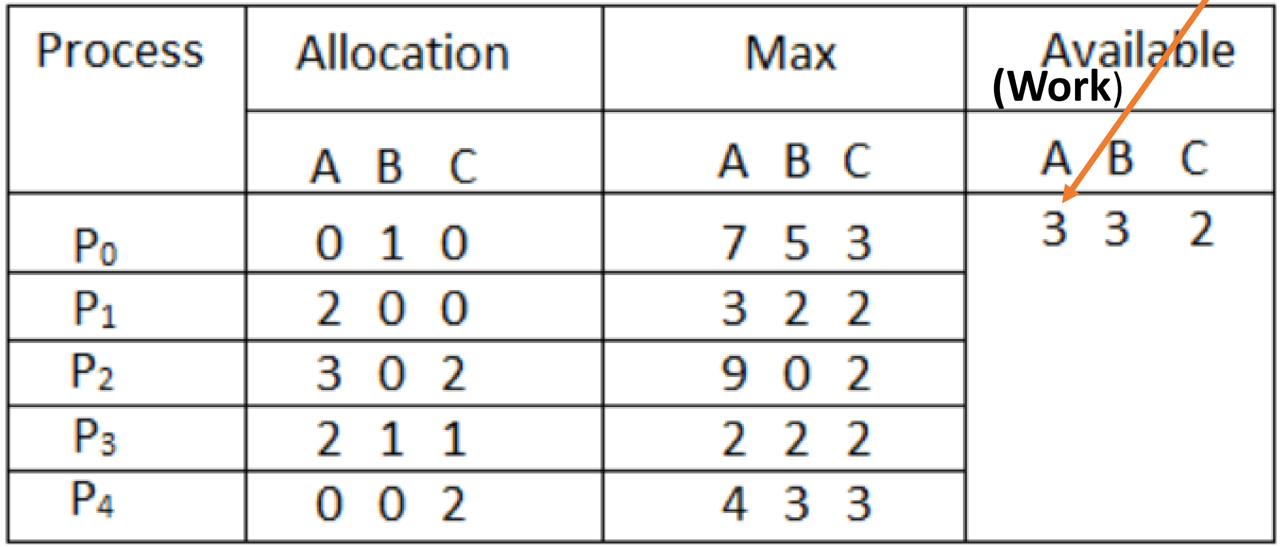
Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

The Banker's algorithm

Example:

Considering a system with five processes P_0 through P_4 and three resources of type A, B, C (A-Printer, B-CD ROM, C- Hard drive). 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:

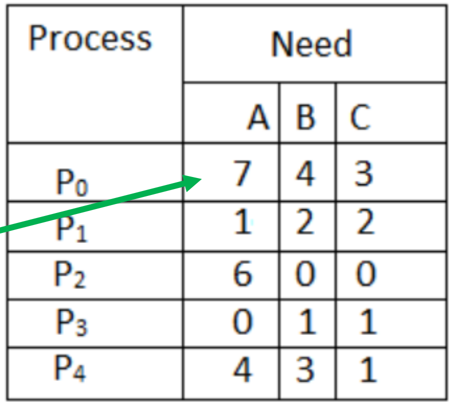
Current available(work) = Total – total Recource allocated
= 10 - 7
= 3



Process	Allocation			Max			Available **(Work)**		
	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			

Total Resource allocation: 7 2 5

Q1: What will be the content of the Need Matrix?



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

Need [i, j] = Max [i, j] – Allocation [i, j]

Example:

Need[P_o, A] = 7 - 0 = 7

Question2. Is the system in a safe state? If Yes, then what is the safe sequence?

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

Banker's formula:

Need <= Available(Work)

So,

P_o: 7 4 3 <= 3 3 2 X (not in a safe state), P_0 must wait

P_1: 1 2 2 <= 3 3 2 √ (must be kept in safe sequence)

then, work = work + allocation

= 3 3 2 + 2 0 0

= 5 3 2

P_2: 6 0 0 <= 5 3 2 X (not in a safe state), P_2 must wait

P_3: 2 1 1 <= 5 3 2 √ (must be kept in safe sequence)

then, work = work + allocation

= 5 3 2 + 2 1 1

= 7 4 3

P_4: 4 3 1 <= 7 4 3 √ then, work = work + allocation

(must be kept in safe sequence)

= 7 4 3 + 0 0 2

= 7 4 5

P_0 and P_2 are in the waiting queue:

Banker's formula:
Need <= current available(Work)

So,

$$\begin{tikzcd}
P_o: & 7 \arrow[rrrr, bend left] & 4 \arrow[rrrr, bend left] & 3 \arrow[rrrr, bend left] & <= & 7 & 4 & 5
\end{tikzcd}$$

√ (must be kept in safe sequence)
then, work = work + allocation
= 7 4 5 + 0 1 0
= 7 5 5

$$\begin{tikzcd}
P_2: & 6 \arrow[rrrr, bend left] & 0 \arrow[rrrr, bend left] & 0 \arrow[rrrr, bend left] & <= & 7 & 5 & 5
\end{tikzcd}$$

√ (must be kept in safe sequence)
then, work = work + allocation
= 7 5 5 + 3 0 2
= 10 5 7

All processes can be kept in safe sequence, hence the system is in Safe State
The safe sequence is P_1, P_3, P_4, P_0, P_2

Banker’ s Algorithm

Process	Allocation			Maximum			Available (current work)		
	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	5	3	2
P2	3	0	2	9	0	2	7	4	3
P3	2	1	1	4	2	2	7	4	5
P4	0	0	2	5	3	3	7	5	5

Banker's Algorithm

Example:

Process	Allocation			Maximum			Available (current work)		
	A	B	C	A	B	C	A	B	C
P0	1	0	1	2	1	1	2	1	1
P1	2	1	2	5	4	4	3	1	2
P2	3	0	0	3	1	1	6	1	2
P3	1	0	1	1	1	1	7	1	3
Total	**7**	**1**	**4**						

Need (Maximum-Allocation)		
A	B	C
1	1	0
3	3	2
0	1	1
0	0	0

Banker's formula:

Need <= Available(Work), then current work =current work + allocation

P0: 1 1 0 <= 2 1 1 √ current work = 2 1 1 + 1 0 1
= 3 1 2

P1: 3 2 2 <= 3 1 2 X

P2: 0 1 1 <= 3 1 2 √ current work = 3 1 2 + 3 0 0
= 6 1 2

P3: 0 0 0 <= 6 1 2 √ current work = 6 1 2 + 1 0 1
= 7 1 3

Banker's Algorithm

Example:

Process	Allocation			Maximum			Available (current work)		
	A	B	C	A	B	C	A	B	C
P0	1	0	1	2	1	1	2	1	1
P1	2	1	2	5	4	4	3	1	2
P2	3	0	0	3	1	1	6	1	2
P3	1	0	1	1	1	1	7	1	3
Total	**7**	**1**	**4**						

Need (Maximum-Allocation)		
A	B	C
1	1	0
3	3	2
0	1	1
0	0	0

Banker's formula:

Need <= Available(Work), then current work =current work + allocation

P1: 3 2 2 <= 7 1 3 X

P1 cannot be kept in safe sequence, hence the system is in Unsafe State

Summary

1. What is a deadlock?
2. List and explain the conditions necessary and sufficient to produce a deadlock.
3. Identify three strategies which can be used to deal with deadlocks.
4. How can we prevent the occurrence of a deadlock occurs?
5. What is Banker's algorithm?
6. How Banker's algorithm works?