Unit 4 Deadlocks

Marwadi University

Computer Engineering Diploma

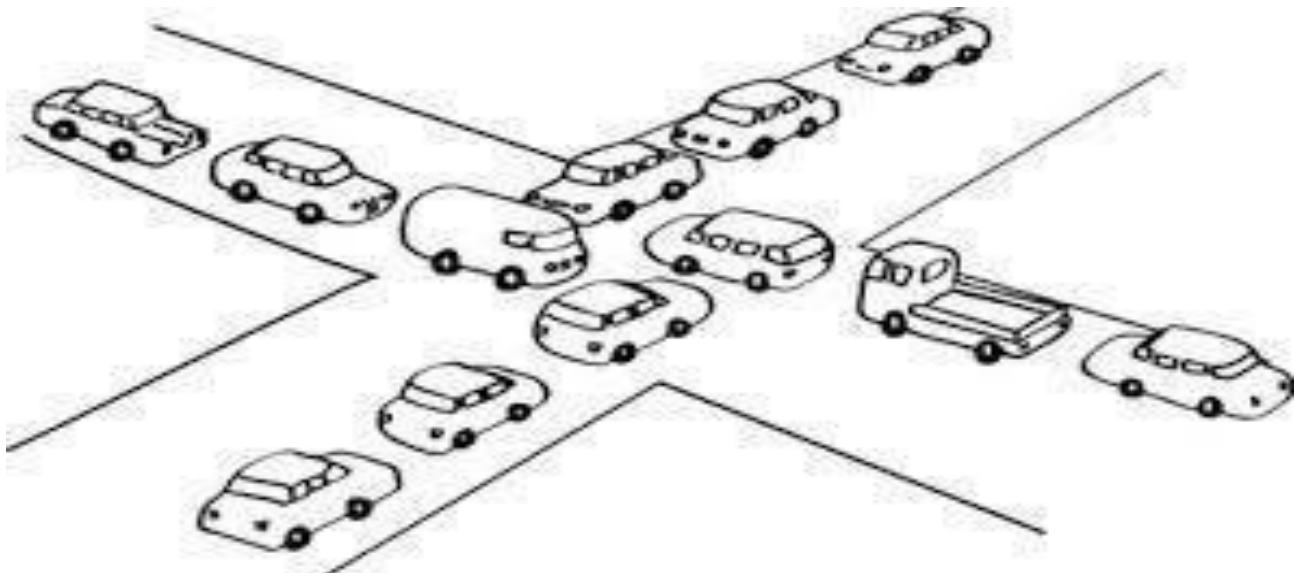
Unit no 4
Deadlocks
Operating System
09ce2405

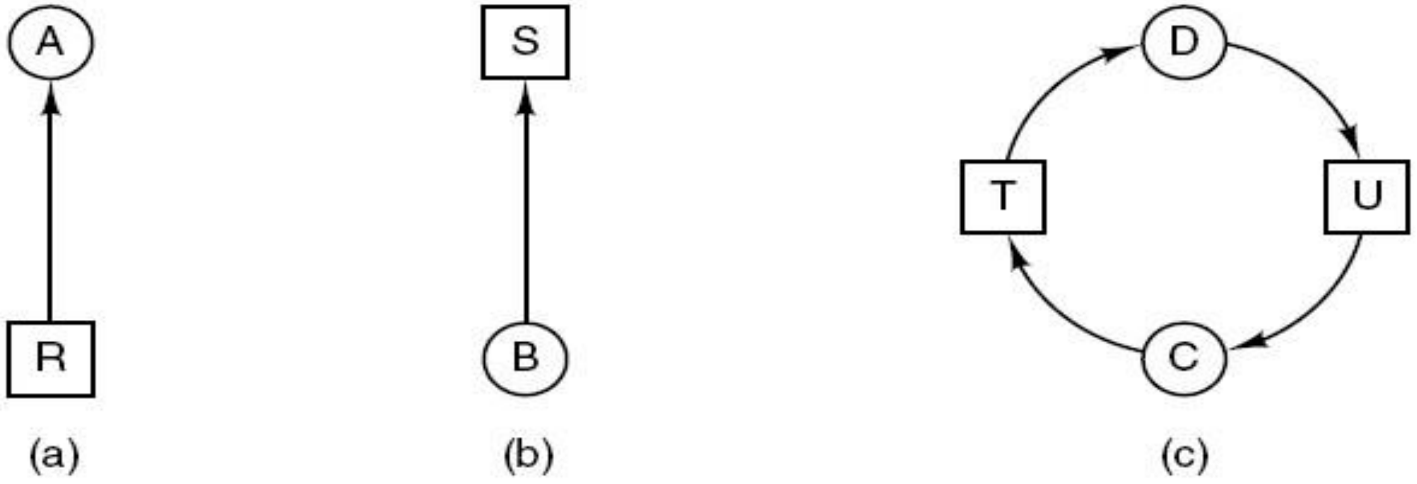
By Prof. Foram Chovatiya

What is Deadlock?

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- “A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause”.

Deadlock





Deadlock Modeling

- Fig. (a) represents that Resource R is given to Process A.
- Fig. (b) represents that Process B requests for Resource S.
- Fig (c) Suppose process D holds resource T and process C holds resource U.
- Now process D requests resource U and process C requests resource T but none of process will get this resource because these resources are already hold by other process so both can be blocked, with neither one be able to proceed, this situation is called deadlock.

Deadlock Characteristics or 4 necessary conditions for deadlock.

There are four conditions that must hold for deadlock:

1. Mutual exclusion condition:-

- One or more than one resource are non-shareable (Only one process can use at a time)

2. Hold and wait condition:-

- A process is holding at least one resource and waiting for resources.

3. No preemption condition:-

- A resource cannot be taken from a process unless the process releases the resource.

Deadlock Characteristics or 4 necessary conditions for deadlock.

4. **Circular wait condition:-**
   - There must be a circular chain of 2 or more processes.
   - Each process is waiting for resource that is held by next member of the chain.
   - There exists a set{P0,P1,…,Pn}of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2,…,Pn-1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0.

NOTE:

All four of these conditions must be present for a deadlock to occur.

System Model

Resource types R1,R2,...,Rn

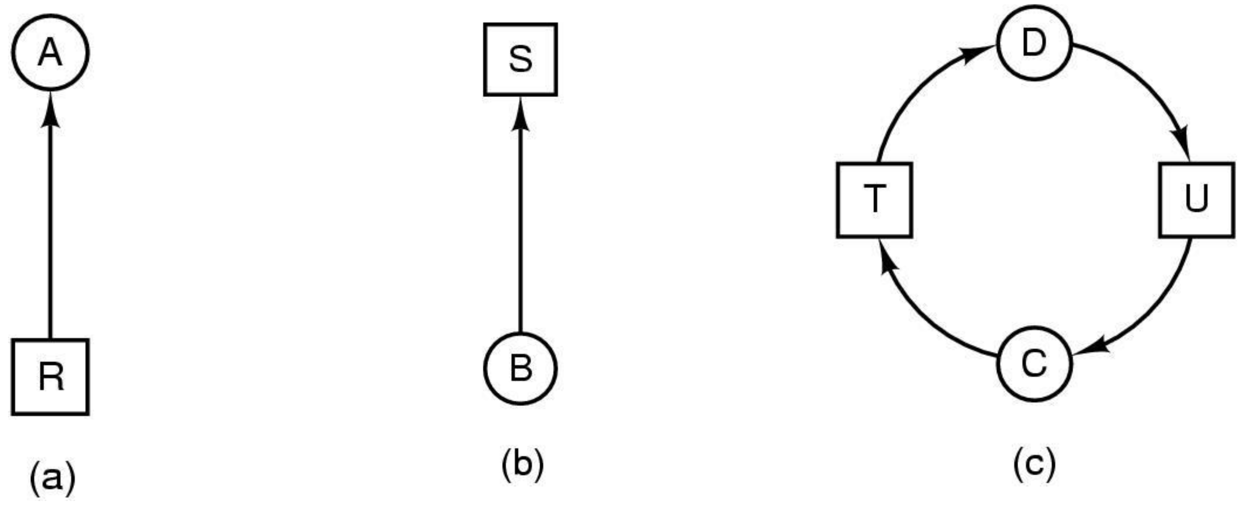
Example - CPU cycles, Memory Space, I/O devices

Each process utilizes a resource as follows:

1. Request
2. Use
3. Release

Deadlock Modeling

Modeled with directed graphs:-

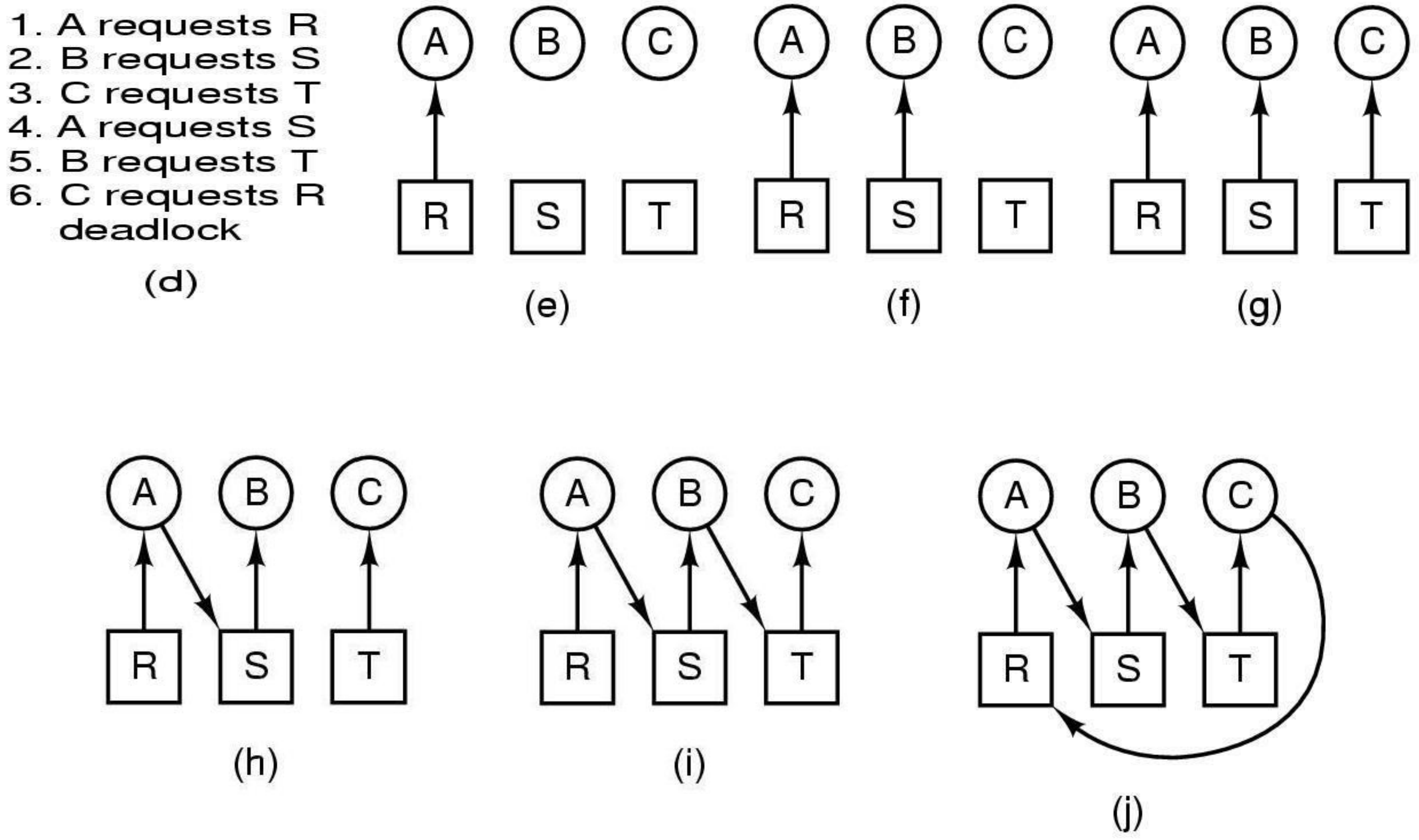


(a) Resource R assigned to process A

(b) Process B is requesting/waiting for resource S

(c) Process C and D are in deadlock over resources T and U

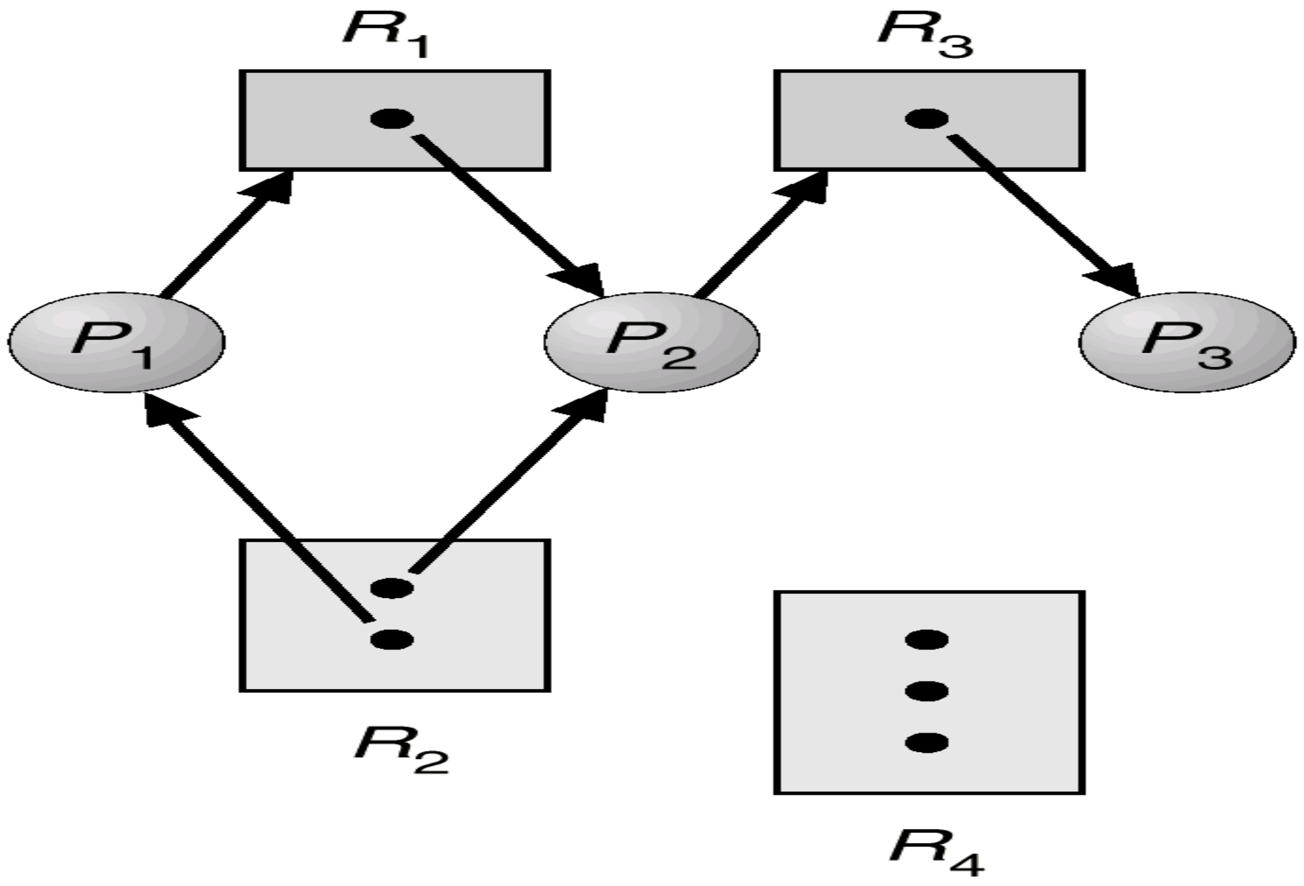
How it Occur?



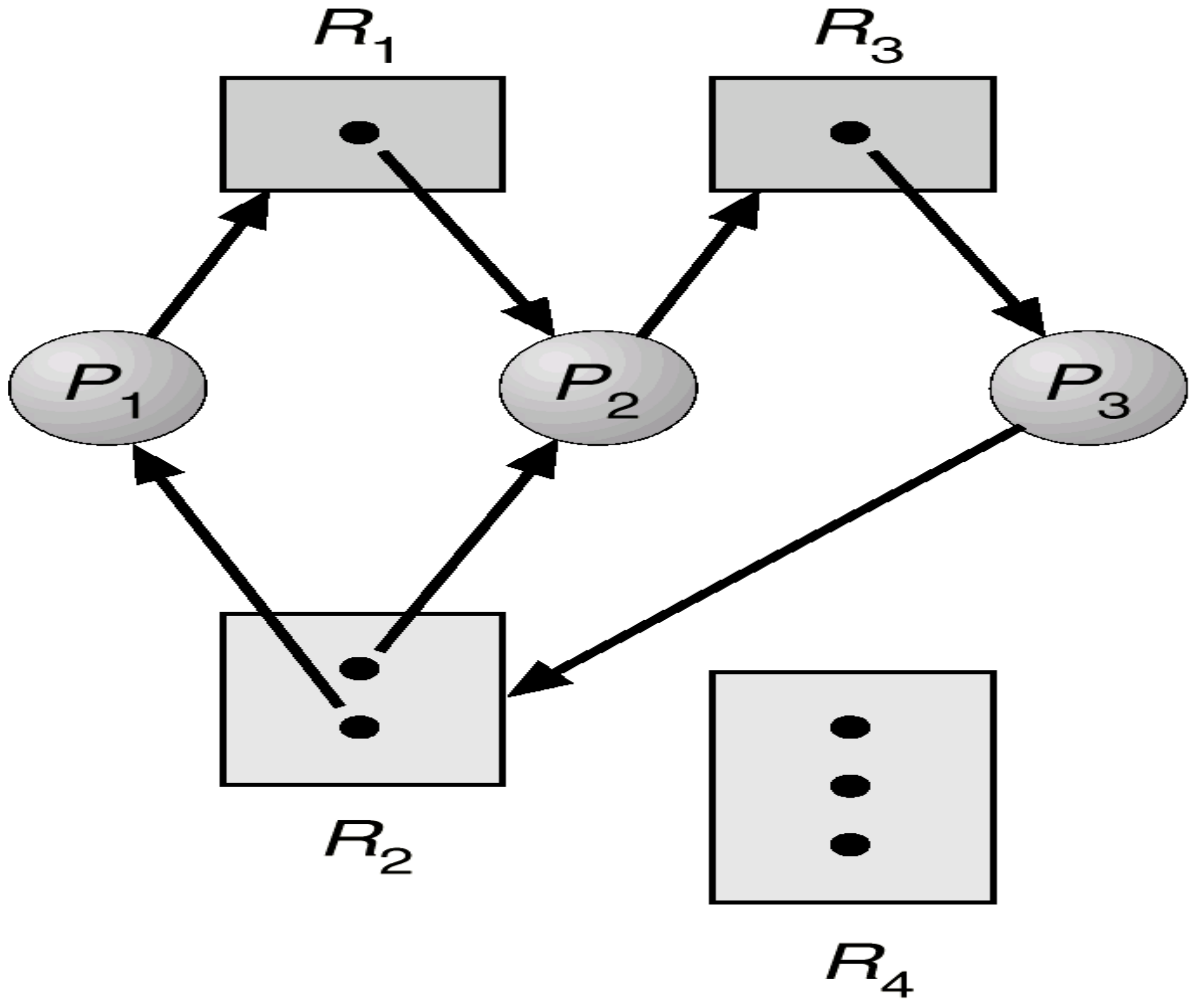
1. A requests R
2. B requests S
3. C requests T
4. A requests S
5. B requests T
6. C requests R
deadlock

(d)

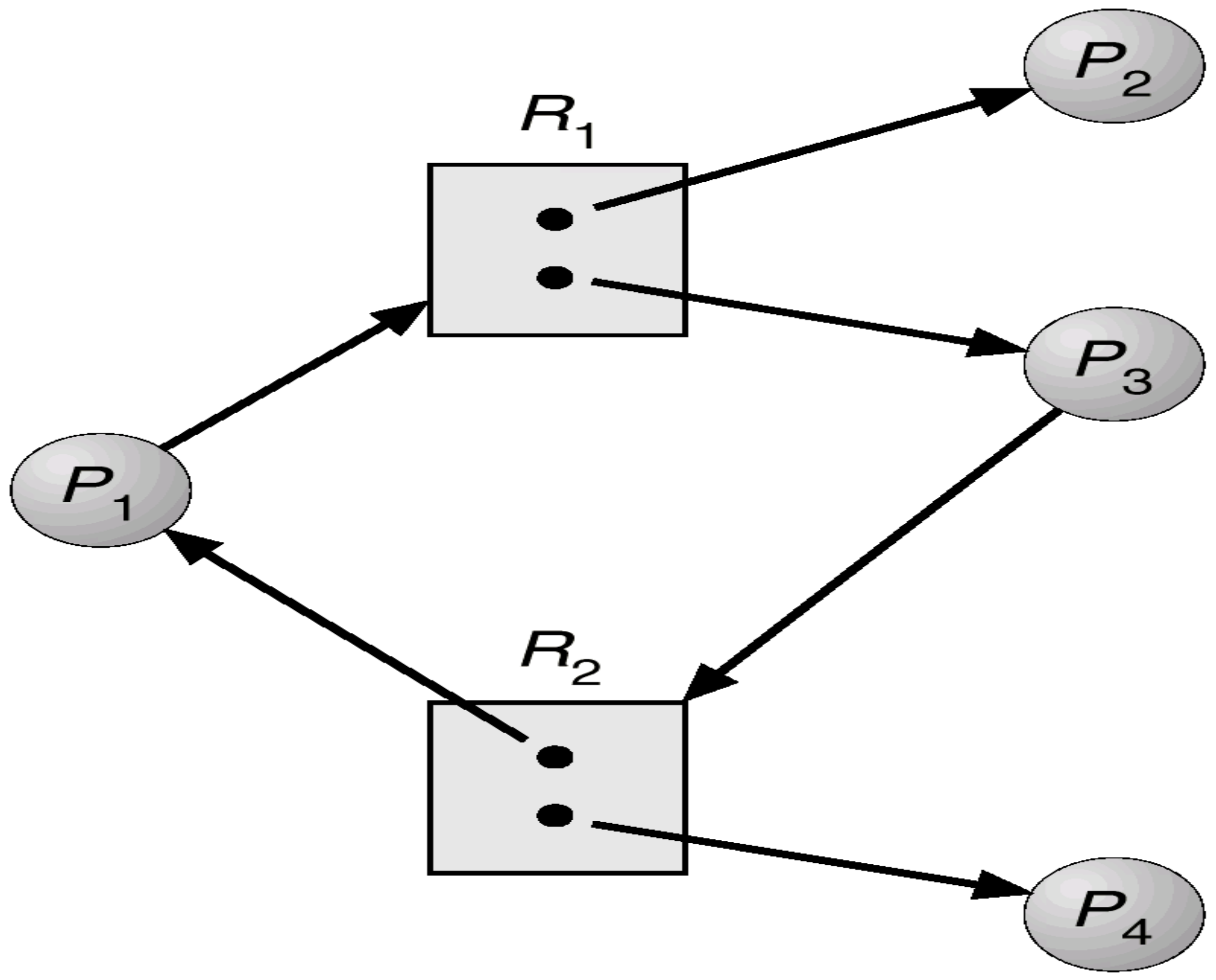
Example of Resource Allocation Graph



Resource Allocation Graph with a Deadlock



Resource Allocation Graph With A Cycle



Moral of the Story

- ❑ If graph contains no cycles -> no deadlock.
- ❑ If graph contains a cycle ->

- If only one instance per resource type, then deadlock.
- if different instances per resource type, possibility of deadlock

Deadlocks

Strategies to deal with the deadlocks

1. Just ignore the problem. Maybe if you ignore it, it will ignore you.
2. Detection and recovery. Let them occur, detect them, and take action.
3. Deadlock avoidance by careful resource allocation.
4. Prevention, by structurally negating one of the four conditions.

Methods to handle Deadlock

1. **Just ignore the problem**
2. **Prevention**
3. **Detection and recovery**
4. **Avoidance**

Deadlocks

1st Solution

Just Ignore the Problem

Just Ignore the Problem

- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.
- Ensure that the system will *never* enter in deadlock state.

Deadlock ignorance. OR Ostrich Algorithm.

- Pretend (imagine) that there’s no problem.
- This is the easiest way to deal with problem.
- This algorithm says that stick your head in the sand and pretend (imagine) that there is no problem at all.
- When storm approaches an ostrich puts his head in sand and pretend that there is no problem at all.
- This strategy suggests to ignore the deadlock because deadlocks occur rarely, but system crashes due to hardware failures, compiler errors, and operating system bugs frequently, then not to pay a large penalty in performance or convenience to eliminate deadlocks.

Deadlock ignorance.
OR
Ostrich Algorithm.

Deadlock ignorance. OR Ostrich Algorithm.

- **This method is reasonable if**
  - Deadlocks occur very rarely
  - Cost of prevention is high
- **UNIX and Windows take this approach**
  - Resources (memory, CPU, disk space) are plentiful
  - Deadlocks over such resources rarely occur
  - Deadlocks typically handled by rebooting

Methods to handle Deadlock

1. **Just ignore the problem**
2. **Prevention**
3. **Detection and recovery**
4. **Avoidance**

Deadlocks

2nd Solution

Deadlock Prevention

Deadlock Prevention

- **Deadlock Prevention can be achieved by dissatisfying any one/more of following condition.**
- Mutual Exclusion
- Hold and Wait
- No Preemption
- Circular Wait

Deadlock Prevention

- ❑ Mutual Exclusion – Make Some Resources Sharable
- Example:- Allow more than one reader to read File.
- Problem : Not possible for All Type of Resources.

Deadlock Prevention

- Hold and Wait

Method 1: Each process will make request and be allocated all its resources before it begins execution.

Method 2 : allows a process to request resources only when it does not hold any other resources.

Deadlock Prevention

- Method1-Example:

- We consider a process that copies data from a DVD drive to a file on disk, sorts the file, and then prints the results to a printer.
- If all resources must be requested at the beginning of the process, then the process must initially request the DVD drive, disk file, and printer.
- It will hold the printer for its entire execution, even though it needs the printer only at the end.

- Problem : Low resource utilization

Deadlock Prevention

- Method 2-Example:

- A process to request initially only the DVD drive and disk file.
- It copies from the DVD drive to the disk and then releases both the DVD drive and the disk file.
- The process must then again request the disk file and the printer.
- After copying the disk file to the printer, it releases these two resources and terminates.
- Problem: Starvation Possible
- Starvation occurs if a process is indefinitely postponed. This may happen if the process requires a resource for execution that it is never allotted or if the process is never provided the processor for some reason.

Deadlock Prevention

- **No Preemption–**

- If a process that is holding some resources and requests another resource that can not be immediately allocated to it, then all resources currently being held are released.

- Preempt any process from deadlock cycle.

Problem: Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

Deadlock Prevention

- **Circular Wait–**

- Each resource will be assigned with a numerical number.
- A process can request for the resources only in increasing order of numbering.

- **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.

Methods to handle Deadlock

1. **Just ignore the problem**

2. **Prevention**

3. **Detection and recovery**

4. **Avoidance**

Deadlock Detection and Recovery

3rd Solution- Deadlock Detection & Recovery

Allow system to enter deadlock state

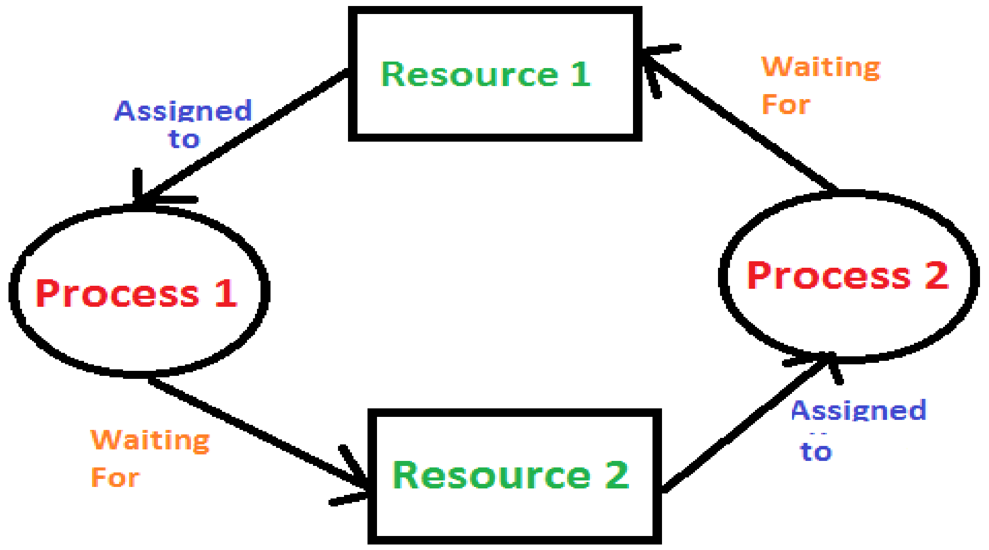
- Detection algorithm
- Recovery scheme

3rd Solution- Deadlock Detection & Recovery

- If deadlocks do occur, the operating system must detect and resolve them.
- **Deadlock detection algorithms**, such as the Wait-For Graph, are used to identify deadlocks, and recovery algorithms, such as the Rollback and Abort algorithm, are used to resolve them.
- **The recovery algorithm** releases the resources held by one or more processes, allowing the system to continue to make progress.

3rd Solution- Deadlock Detection & Recovery

- **If resources have a single instance –**
  In this case for Deadlock detection, we can run an algorithm to check for the cycle in the Resource Allocation Graph. The presence of a cycle in the graph is a 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.



3rd Solution- Deadlock Detection & Recovery

- **Deadlock Recovery :**
  A traditional operating system such as Windows doesn’t deal with deadlock recovery as it is a time and space-consuming process. Real-time operating systems use Deadlock recovery.
- **1. Killing the process –**
  Killing all the processes involved in the deadlock. Killing process one by one. After killing each process check for deadlock again keep repeating the process till the system recovers from deadlock. Killing all the processes one by one helps a system to break circular wait condition.
- 2. **Resource Preemption –**
  Resources are preempted from the processes involved in the deadlock, preempted resources are allocated to other processes so that there is a possibility of recovering the system from deadlock. In this case, the system goes into starvation.

Methods to handle Deadlock

1. Just ignore the problem

2. Prevention

3. Detection and recovery

4. Avoidance

Deadlock Avoidance

4th Solution- Deadlock Avoidance

- If we have future information

Max resource requirement of each process before they execute

- Can we guarantee that deadlocks will never occur?

Avoidance Approach:

- Before granting resource, check if state is safe
- If the state is safe? No deadlock!

Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
- System is in safe state if there exists a safe sequence of all processes.

❑ Sequence <P1,P2,…,Pn> is safe if

- for each Pi, the resources that Pi can still request can be satisfied by
- currently available resources + resources held by all the Pj.

Safe State

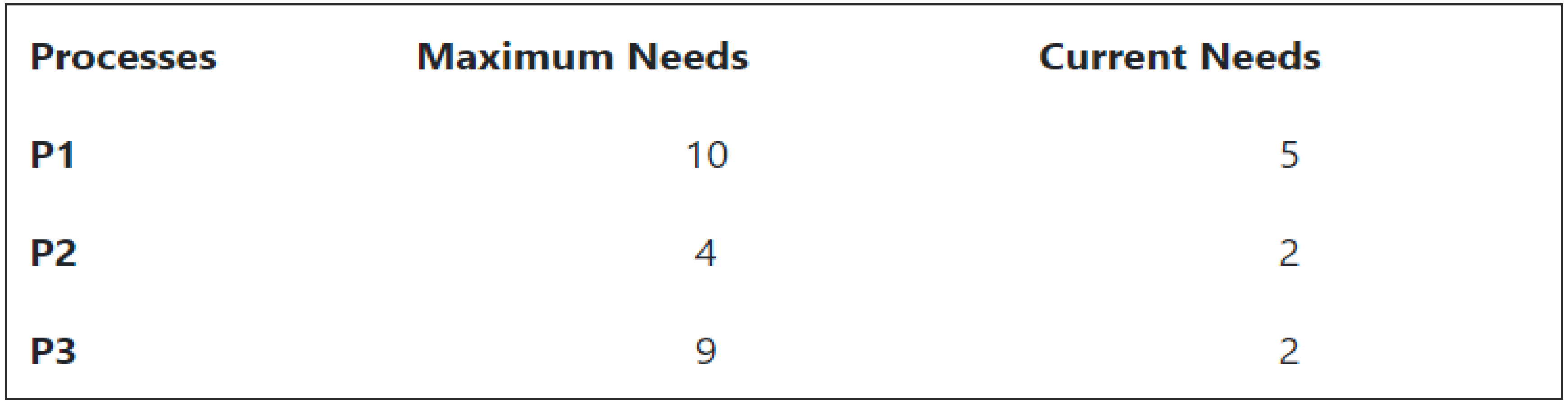
- If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished.
- When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate.

Deadlock Avoidance

Safe State and Unsafe State

Deadlock Avoidance Example:-

- Let us consider a system having 12 magnetic tapes and three processes P1, P2, P3.
- Process P1 requires 10 magnetic tapes,
- Process P2 may need as many as 4 tapes,
- Process P3 may need up to 9 tapes.
- Suppose at a time T0, process P1 is holding 5 tapes, process P2 is holding 2 tapes and process P3 is holding 2 tapes. (There are 3 free magnetic tapes)

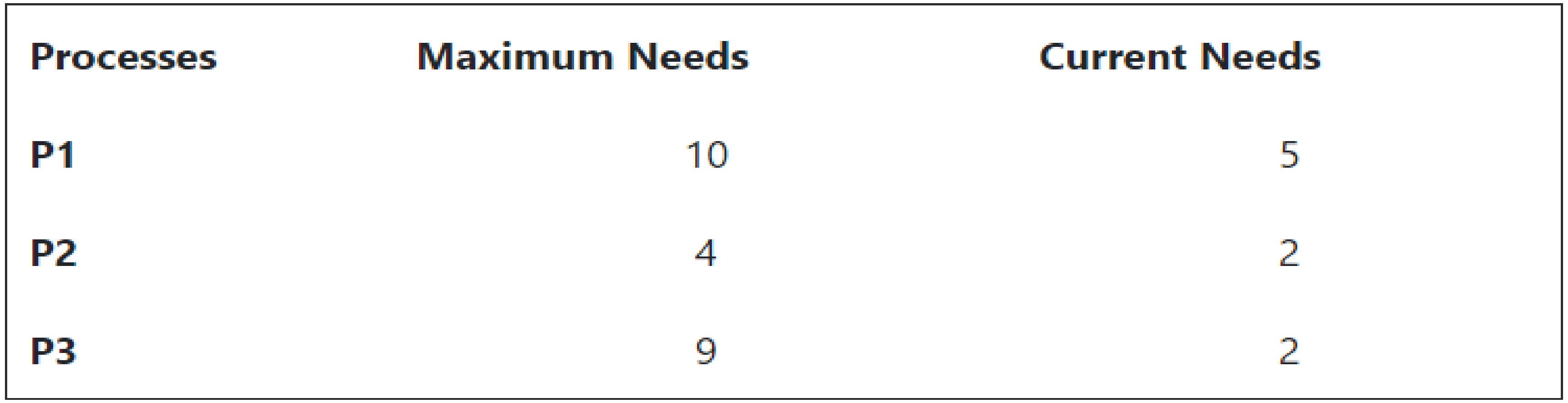


Processes	Maximum Needs	Current Needs
P1	10	5
P2	4	2
P3	9	2

Deadlock Avoidance

Safe State and Unsafe State

Deadlock Avoidance Example



Processes	Maximum Needs	Current Needs
P1	10	5
P2	4	2
P3	9	2

- So at time T0, the system is in a safe state.
- The sequence is <P2,P1,P3> satisfies the safety condition.

Deadlock Avoidance

Safe State and Unsafe State

Deadlock Avoidance Example

- Process P2 can immediately be allocated all its tape drives and then return them.
- After the return the system will have 5 available tapes, then process P1 can get all its tapes and return them (the system will then have 10 tapes).
- finally, process P3 can get all its tapes and return them (The system will then have 12 available tapes).

Deadlock Avoidance

Safe State and Unsafe State

Deadlock Avoidance Example

- A system can go from a safe state to an unsafe state.
- Suppose at time T1, process P3 requests and is allocated one more tape.
- The system is no longer in a safe state. At this point, only process P2 can be allocated all its tapes.
- When it returns them the system will then have only 4 available tapes.
- Since P1 is allocated five tapes but has a maximum of ten so it may request 5 more tapes.
- If it does so, it will have to wait because they are unavailable.
- Similarly, process P3 may request its additional 6 tapes and have to wait which then results in a deadlock

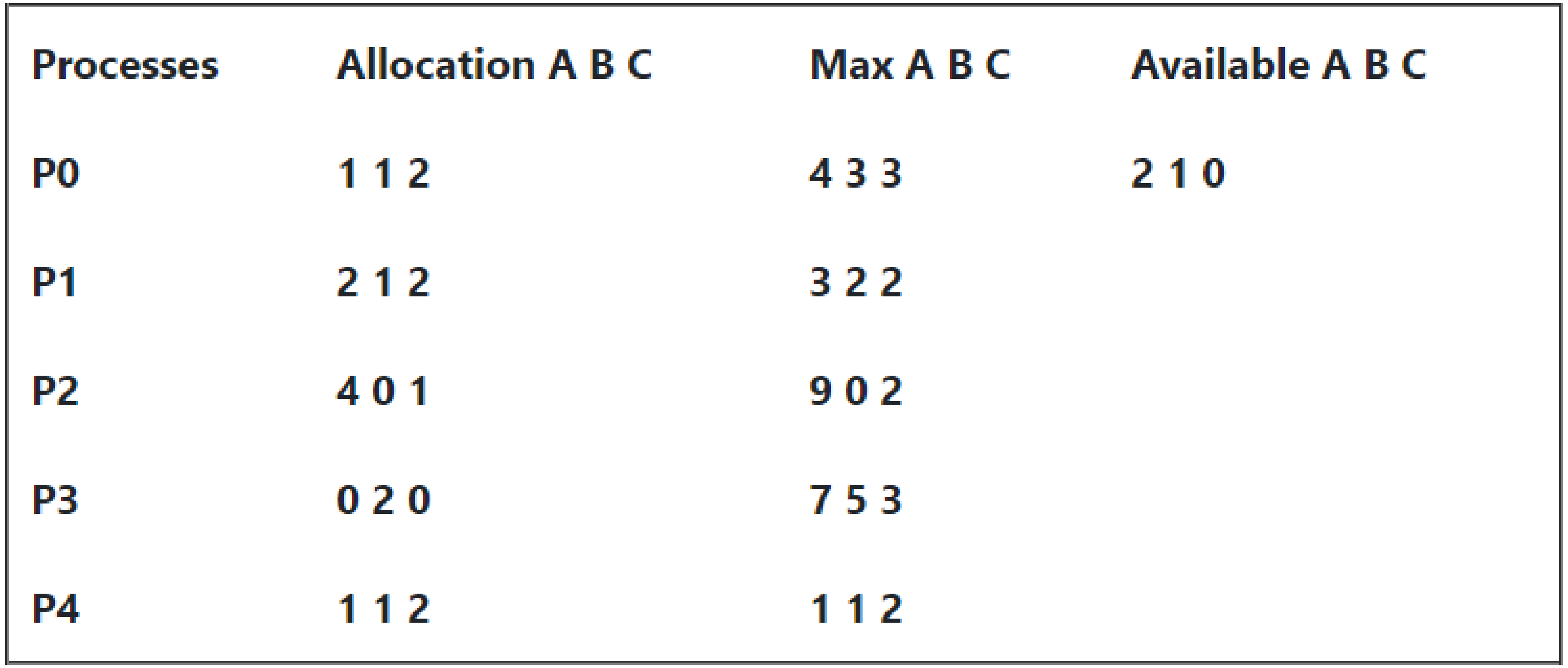
Deadlock Avoidance

Bankers Algorithm:-

- Banker's algorithm is a **deadlock avoidance algorithm**.
- It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.
- Consider there are n account holders in a bank and the sum of the money in all of their accounts is S.
- Every time a loan has to be granted by the bank, it subtracts the **loan amount** from the **total money** the bank has.
- Then it checks if that difference is greater than S.
- It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.
- Banker's algorithm works in a similar way in computers.

Deadlock Avoidance

Bankers Algorithm.



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

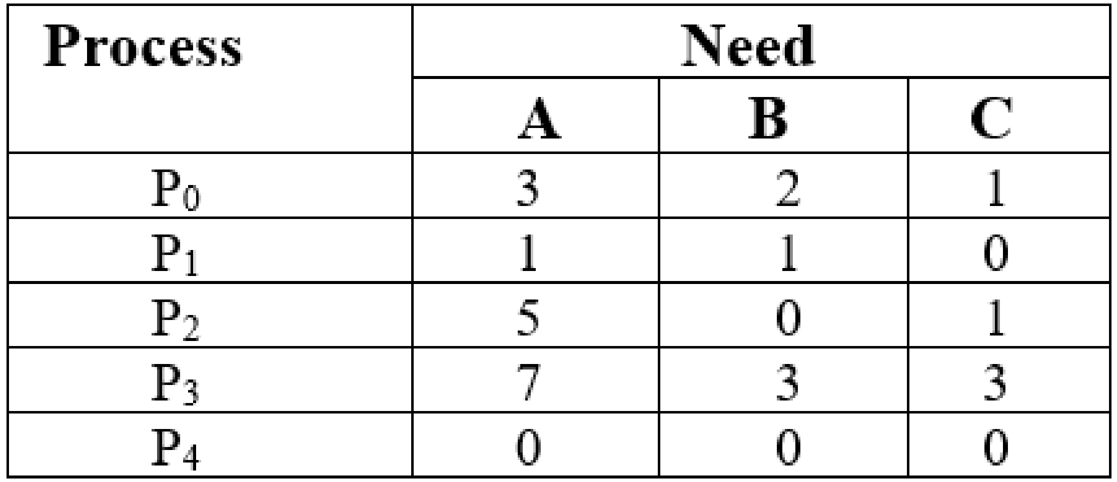
1. Calculate the content of the need matrix?
2. Check if the system is in a safe state?
3. Determine the total sum of each type of resource?

Deadlock Avoidance

Bankers Algorithm.

The Content of the need matrix can be calculated by using the formula given below:

Need = Max – Allocation



Process	Need		
	A	B	C
P_0	3	2	1
P_1	1	1	0
P_2	5	0	1
P_3	7	3	3
P_4	0	0	0

Deadlock Avoidance

Bankers Algorithm.

Let us now check for the safe state.

Safe sequence:

1. For process P0, Need = (3, 2, 1) and

   Available = (2, 1, 0)

   Need <=Available = False

   So, the system will move to the next process.

2. For Process P1, Need = (1, 1, 0)

   Available = (2, 1, 0)

   Need <= Available = True

   Request of P1 is granted.

   Available = Available + Allocation

   = (2, 1, 0) + (2, 1, 2)

   = **(4, 2, 2) (New Available)**

Deadlock Avoidance

Bankers Algorithm.

3. For Process P2, Need = (5, 0, 1)

   Available = (4, 2, 2)

   Need <= Available = False

   So, the system will move to the next process.

4. For Process P3, Need = (7, 3, 3)

   Available = (4, 2, 2)

   Need <= Available = False

   So, the system will move to the next process.

Deadlock Avoidance

Bankers Algorithm.

5. For Process P4, Need = (0, 0, 0)
   Available = (4, 2, 2)
   Need <= Available = True
   Request of P4 is granted.
   Available = Available + Allocation
   = (4, 2, 2) + (1, 1, 2)
   = (5, 3, 4) now, (New Available)
6. Now again check for Process P2, Need = (5, 0, 1)
   Available = (5, 3, 4)
   Need <= Available = True
   Request of P2 is granted.
   Available = Available + Allocation
   = (5, 3, 4) + (4, 0, 1)
   = (9, 3, 5) now, (New Available)

Deadlock Avoidance

Bankers Algorithm.

7. Now again check for Process P3, Need = (7, 3, 3)
   Available = (9, 3, 5)
   Need <=Available = True
   The request for P3 is granted.
   Available = Available +Allocation
   = (9, 3, 5) + (0, 2, 0) = (9, 5, 5)
8. Now again check for Process P0, = Need (3, 2, 1)
   = Available (9, 5, 5)
   Need <= Available = True
   So, the request will be granted to P0.
   Safe sequence: < P1, P4, P2, P3, P0>

Deadlock Avoidance

Bankers Algorithm.

The system allocates all the needed resources to each process.

So, we can say that the system is in a safe state.

The total amount of resources will be calculated by the following formula:

The total amount of resources = sum of columns of allocation + Available

= [8 5 7] + [2 1 0]

= [10 6 7]

Deadlock Avoidance

- **Bankers algorithm Example.**

- 5 processes P_0 through P_4;
- 3 resource types A (10 instances), B (7 instances), and C (7instances).

	ALLOCATED			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	2	2	4	3	3			

Deadlock Avoidance

- Bankers algorithm Example.

	ALLOCATED			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	2	2	4	3	3			

Need is defined to be Max – Allocation. Find safe sequence?

	A	B	C
Po	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
p4	4	1	1