

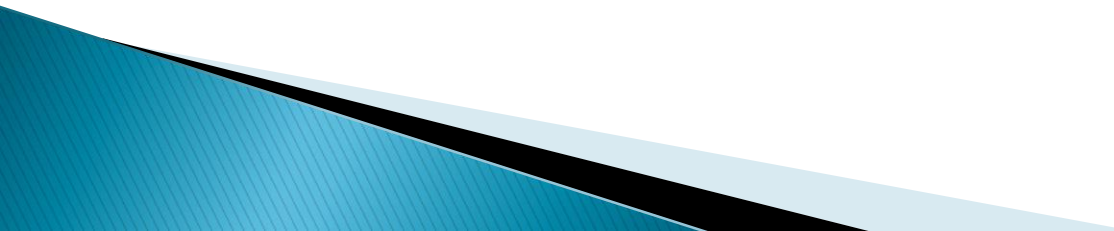
Deadlocks

In Picture Form:



Neither truck can proceed.

Deadlock Problem

- ▶ A system consists of a finite number of resources to be distributed among a number of competing processes.
 - ▶ The resources are partitioned into several types each of which consists of some number of identical instances.
 - ▶ Eg of resources : CPU, files, I/O devices(printers, disk, tape drive)
 - ▶ Eg: If a system has 2 CPUs, then the resource type CPU has 2 instances.
- 

Deadlock Problem

- ▶ A process may utilize a resource in only the following sequence:
 1. **Request:** If a request cannot be granted immediately(because the resource is being used by another process), then the requesting process must wait, until it can acquire the resource.
 2. **Use :** The process can operate on the resource.
 3. **Release :** The process releases the resource.
- ▶ The request ,release and use of resources are system calls.
- ▶ When a program to a hardware resource, it must ask the OS to provide access to that resource. This is done via something called a **system call**.
- ▶ Eg: to access a file(resource) open , close, read, write are system calls.

Deadlock Problem

- ▶ The operating system maintains a table which records whether each resource is free or allocated, and if a resource is allocated, it is allocated to which process.
- ▶ If a process requests a resource a resource that is currently allocated to another process, it is added to a queue of processes waiting for this resource.

What is a deadlock?

Eg :1 Consider a system with 3 tape drives.

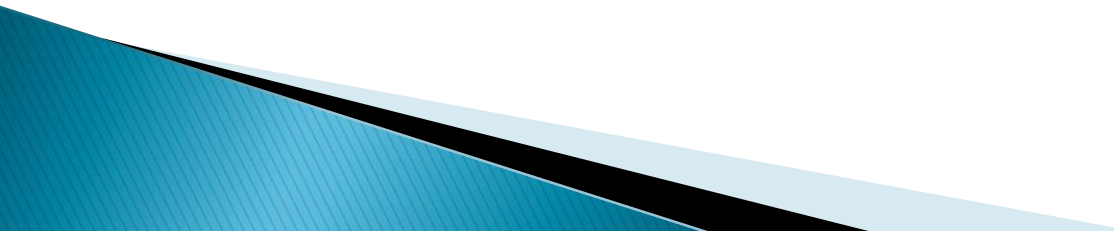
Suppose each of the 3 processes holds one of the tape drives. If each process now requests another tape drive, the 3 processes will be in a deadlock state.

Deadlock Problem

Deadlocks may also involve different resource types.

Eg :Consider a system with 1 printer and 1 tape drive.

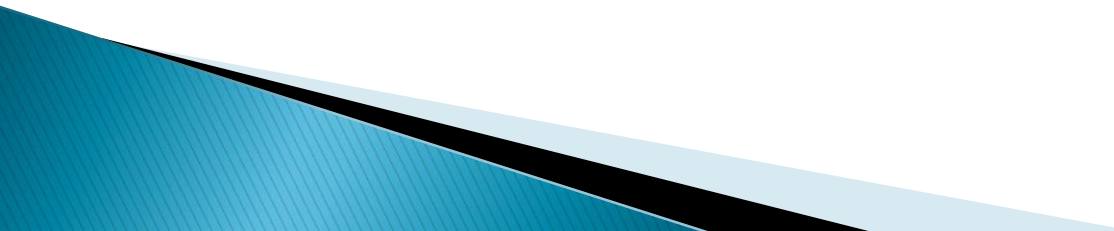
Suppose that Process P_i is holding the tape drive and the process P_j is holding the printer. If P_i requests the printer and P_j requests the tape drive, a deadlock occurs.



Deadlock Characterization

Necessary conditions for a deadlock:

Deadlock can arise if four conditions hold simultaneously.

- ▶ **Mutual exclusion:** only one process at a time can use a resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
 - ▶ **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
 - ▶ **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- 

Deadlock Characterization

Circular wait: there exists a set $\{P_0, P_1, \dots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , ..., P_{n-1} is waiting for a resource that is held by P_n , and P_n is waiting for a resource that is held by P_0 .

ALL THE FOUR CONDITIONS MUST HOLD SIMULTANEOUSLY FOR A DEADLOCK TO OCCUR



Resource Allocation graph

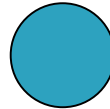
- ▶ Deadlocks can be described with the help of a directed graph called a Resource Allocation Graph(RAG)

A set of vertices V and a set of edges E .

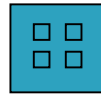
- ▶ V is partitioned into two types:
 - $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the processes in the system.
 - $R = \{R_1, R_2, \dots, R_m\}$, the set consisting of all resource types in the system.
- ▶ Request edge – directed edge $P_i \rightarrow R_j$
- ▶ Assignment edge – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph

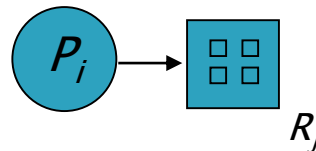
- ▶ Process



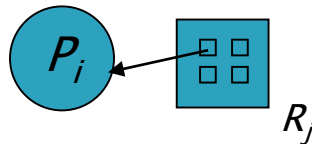
- ▶ Resource Type with 4 instances



- ▶ P_i requests instance of R_j – *request edge*



- ▶ P_i is holding an instance of R_j – *assignment edge*

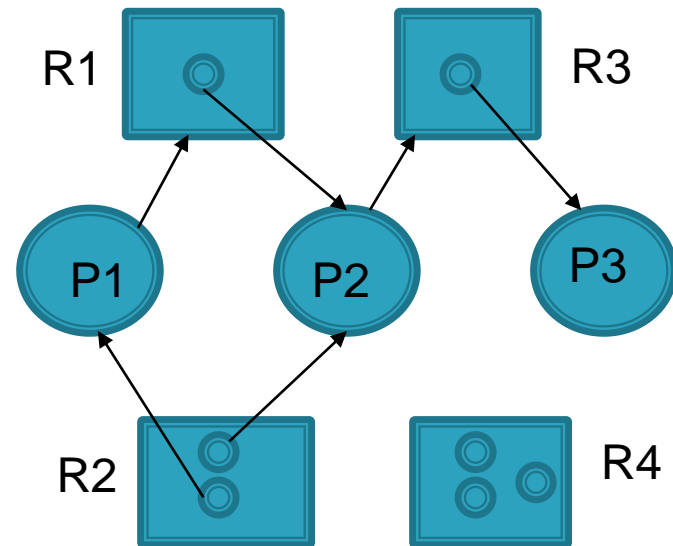


Eg of RAG

- ▶ $P = \{P1, P2, P3\}$
- ▶ $R = \{R1, R2, R3, R4\}$
- ▶ $E = \{P1 \rightarrow R1, P2 \rightarrow R3, R1 \rightarrow P2, R2 \rightarrow P2, R2 \rightarrow P1, R3 \rightarrow P3\}$

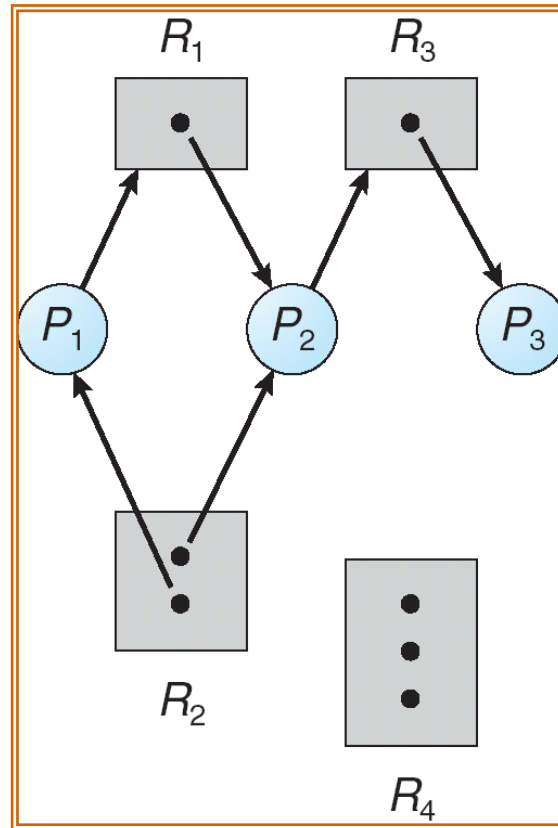
Resource instances

- ▶ 1 instance of R1
- ▶ 2 instances of R2
- ▶ 1 instance of R3
- ▶ 3 instances of R4



Example of a Resource Allocation Graph

► $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

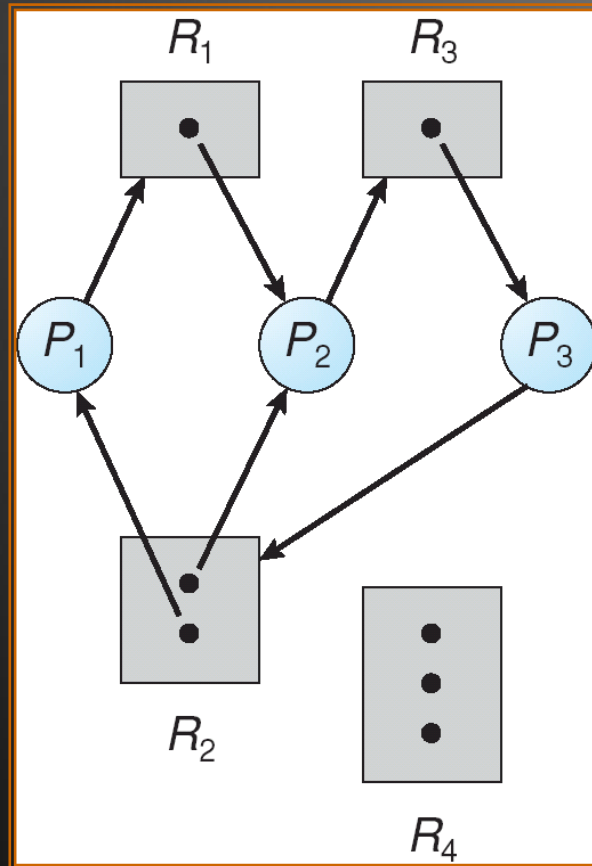


No cycle hence no deadlock

The Deadlock Condition

- ▶ It can be shown that if the graph contains **no cycles**, then **no** process in the system is **deadlocked**.
- ▶ If the graph does contain a **cycle**, then a **deadlock may** exist.

Resource Allocation Graph With A Deadlock



Cycles:

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

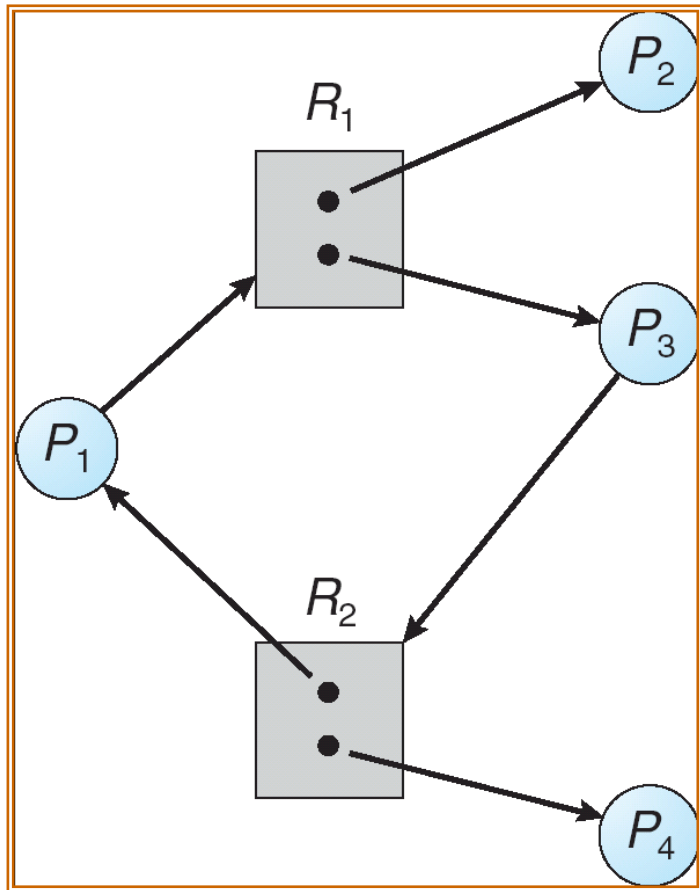
Processes P_1 , P_2 and P_3 are **deadlocked**:

P_2 is waiting for R_3 , which is held by P_3

P_3 is waiting for either P_1 or P_2 to release R_2

P_1 is waiting for P_2 to release R_1

Graph With A Cycle But No Deadlock



Cycle:

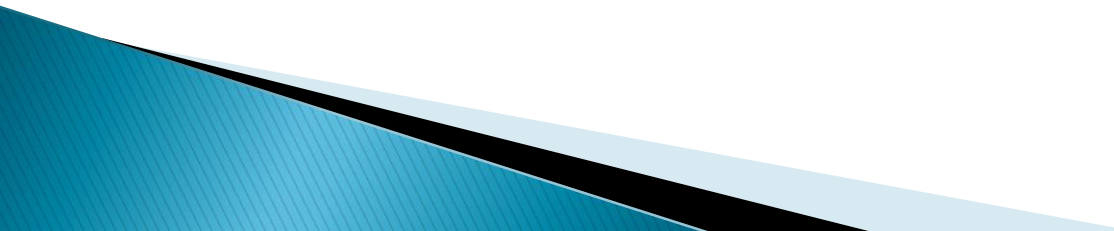
$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

There is **no deadlock**:

P_4 may release its instance on resource type R_2 . That resource can then be allocated to P_3 , breaking the cycle.

Necessary and sufficient conditions for deadlock

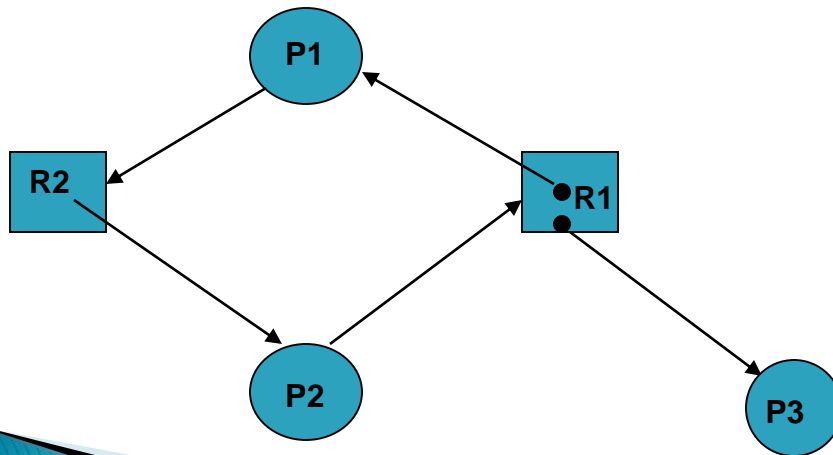
Necessary and sufficient conditions for deadlock

- A cycle is a necessary condition for a deadlock to occur.
 - But if a graph contains a cycle then a deadlock may or may not exist.
 - Hence presence of a cycle is a necessary but not a sufficient condition for the existence of a deadlock
- 

Necessary and sufficient conditions for deadlock

What is a knot?

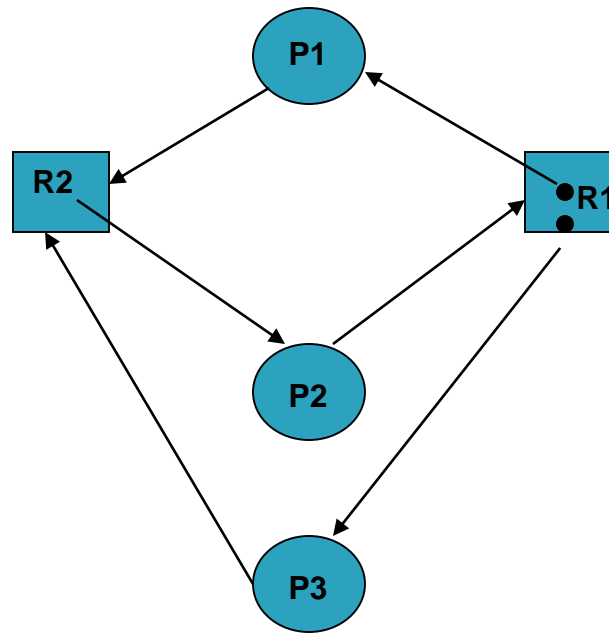
A knot K in a graph is a nonempty set of nodes such that for every node x in K , all nodes in K and only the nodes in K are reachable from x .



Cycle but no deadlock since resource R1 has 2 units and there are no knots

Necessary and sufficient conditions for deadlock

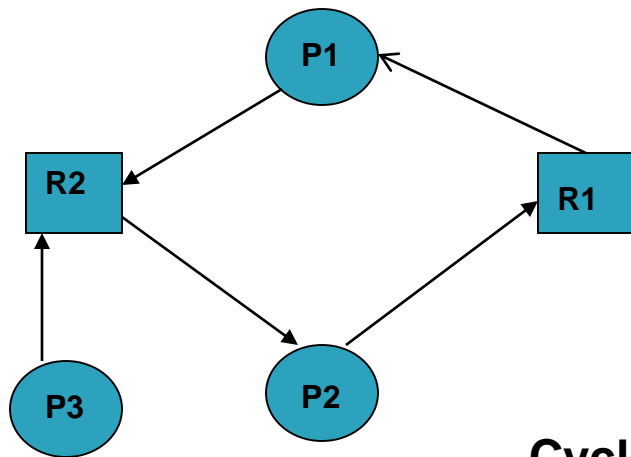
Suppose P3 requests for R2 and a request edge (R3,R2) is added to the graph then the modified graph----→



This graph has 2 cycles (P1,R2,P2,R1,P1) and (P3,R2,P2,R1,P3) and a knot (P1,P2,P3,R1,R2). **Since the graph contains a knot, it represents a deadlock**

Necessary and sufficient conditions for deadlock

Single instance of each resource



Cycle present hence P1,P2 and P3 are deadlocked

Necessary and sufficient conditions for deadlock

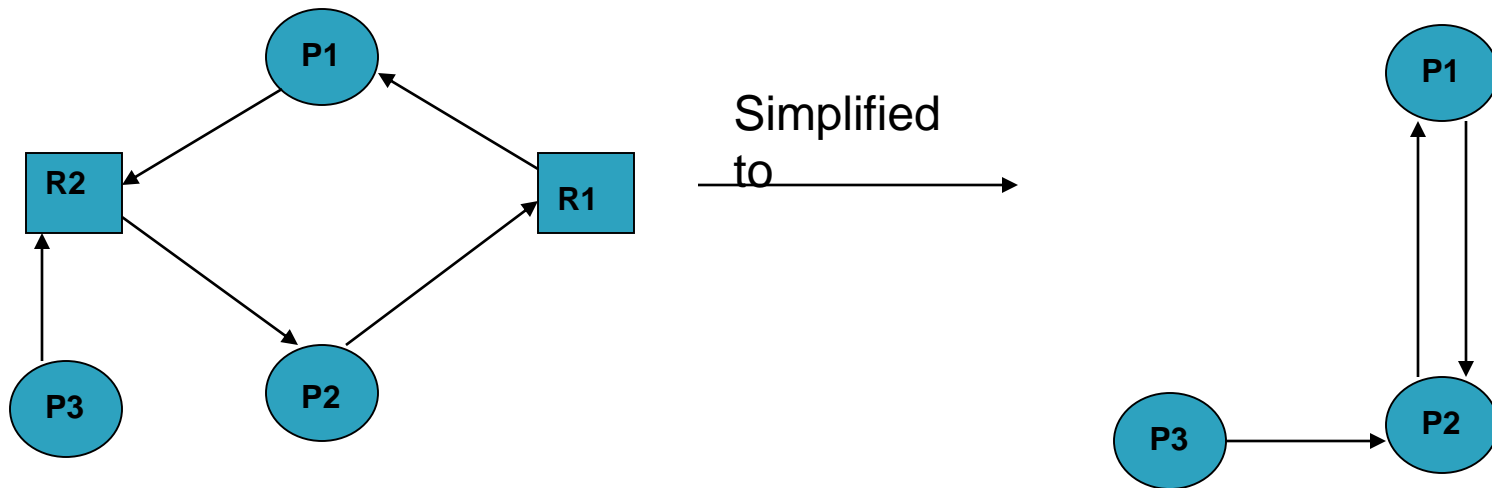
The sufficient conditions for deadlock is different for the following different cases :

1. A cycle in the graph is both a necessary and a sufficient condition for deadlock if all of the resource types requested by the processes forming the cycle have only a single unit each.
2. A cycle in the graph is a necessary but not a sufficient condition for a deadlock if one or more of the resource types requested by the processes forming the cycle have more than one unit. In this case a **knot is a sufficient condition for a deadlock**

Wait for graph

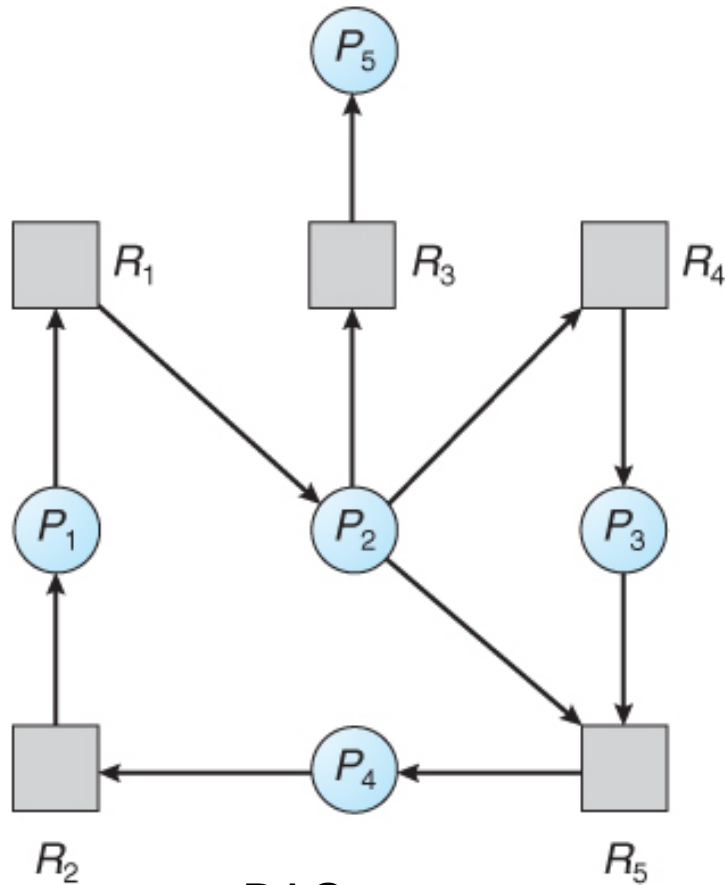
2. Wait for graph

When all resource types have a single unit each, simplified form of resource allocation graph called **wait for graph** could be used.

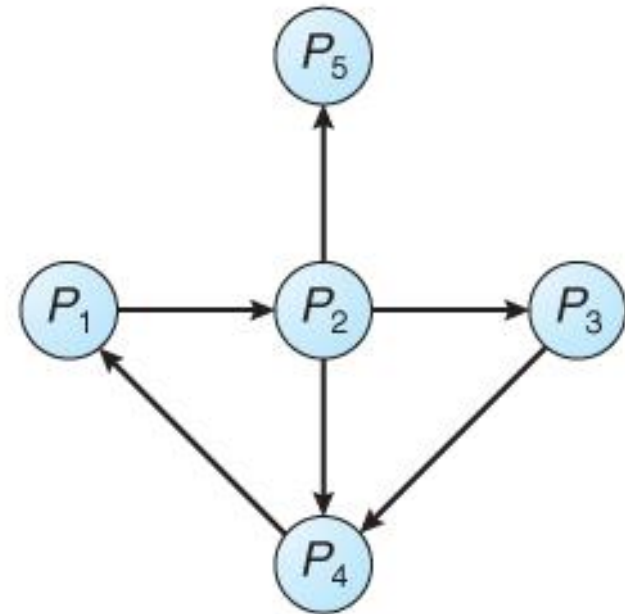


Wait for graph

Convert the following RAG to WFG



RAG



WFG

Methods for handling deadlocks

There are 3 different methods for dealing with the deadlock problem :

- ▶ Use a protocol to ensure that the system will never enter a deadlock state.(Deadlock avoidance and deadlock prevention)
- ▶ Allow the system to enter a deadlock state and then recover.(Deadlock Detection)

Methods for handling deadlocks

1. **Deadlock Prevention :**

A set of methods for ensuring that at least one of the necessary conditions cannot hold.

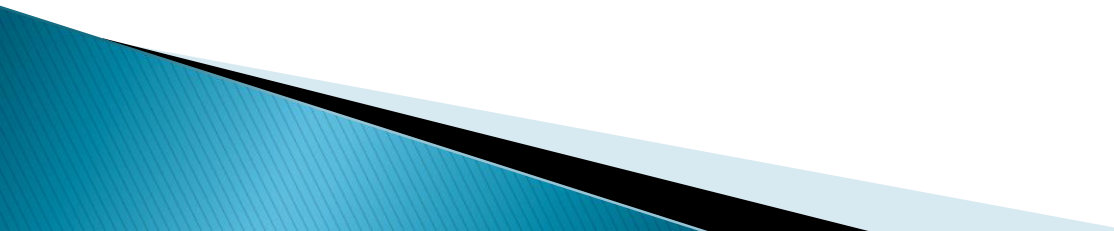
2. **Deadlock avoidance :**

Predicts the future state of the system for avoiding allocations that can eventually lead to a deadlock.

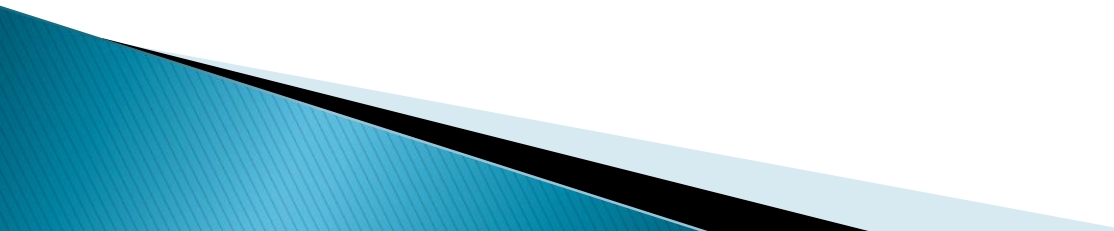
3. **Deadlock detection :**

If a system does not employ either a deadlock prevention or deadlock avoidance algorithm, then a deadlock situation may occur.

Hence an algorithm can be used to determine whether a deadlock has occurred, and an algorithm can be used to recover from deadlock.




Deadlock Prevention

- ▶ For a deadlock to occur, each of the 4 necessary conditions must hold.
 - ▶ By ensuring that at least one of the 4 conditions cannot hold, occurrence of deadlock can be prevented.
 - ▶ Examining each of the 4 necessary conditions separately.....
- 

Deadlock Prevention

1. Mutual exclusion

- ▶ For mutual exclusion to not happen, resources should be sharable.
 - ▶ But printer cannot be simultaneously shared by several processes.
 - ▶ Read only files are an example of sharable resource.
 - ▶ If several processes attempt to open a read only file at the same time, they can be granted simultaneous access to the file.
 - ▶ A process never waits for a sharable resource.
 - ▶ In general , however, we deadlocks cannot be prevented by denying mutual exclusion condition.
 - ▶ Some resources are intrinsically non sharable.
- 

Deadlock Prevention

2. Hold and wait

- ▶ This method denies the hold and wait condition by ensuring that whenever a process requests a resource, it does not hold any other resources.
- ▶ One of the following policies are used:
First...
 - A process must request all of the resources before it begins execution.
 - If all the needed resources are available, they are allocated to the process
 - If one of the requested resources are not available, none will be allocated and the process would just wait.

Deadlock Prevention

Second...

- Instead of requesting all its resources before its execution starts, a process may request a resource during its execution if it obeys the rule that it requests resources only when it holds no other resources.


Advantages of the second policy over the first :

1. Processes generally do not know how many resources they will need until they have started running.
2. A long process may require some resources only towards the end of execution. Hence using the first policy a process will unnecessarily hold these resources for the entire duration of its execution.

Deadlock Prevention

3. Circular wait

Denies circular wait

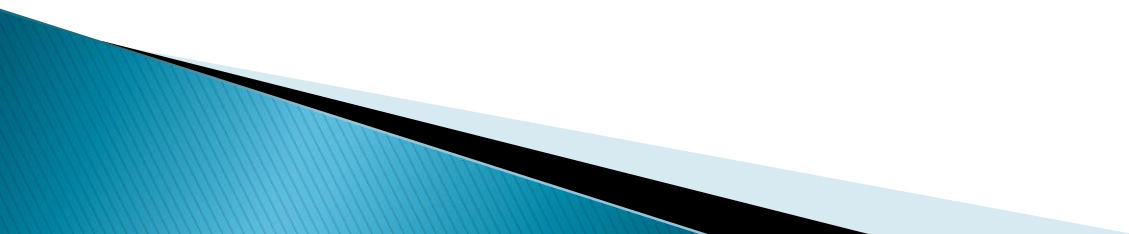
- Each resource type is assigned a unique global number to impose a total ordering of all resource types.
 - **A process should not request a resource with a number lower than the number of any resources that it is holding.**
 - That is if a process holds a resource type whose number is i , it may request a resource type having the number j only if $j > i$
 - Hence with this rule there can never be cycles in the resource allocation graph (denying circular wait condition) and hence making deadlock impossible to occur.
 - **Note : Does not require that a process must acquire all its resources in strictly increasing sequence**
- 

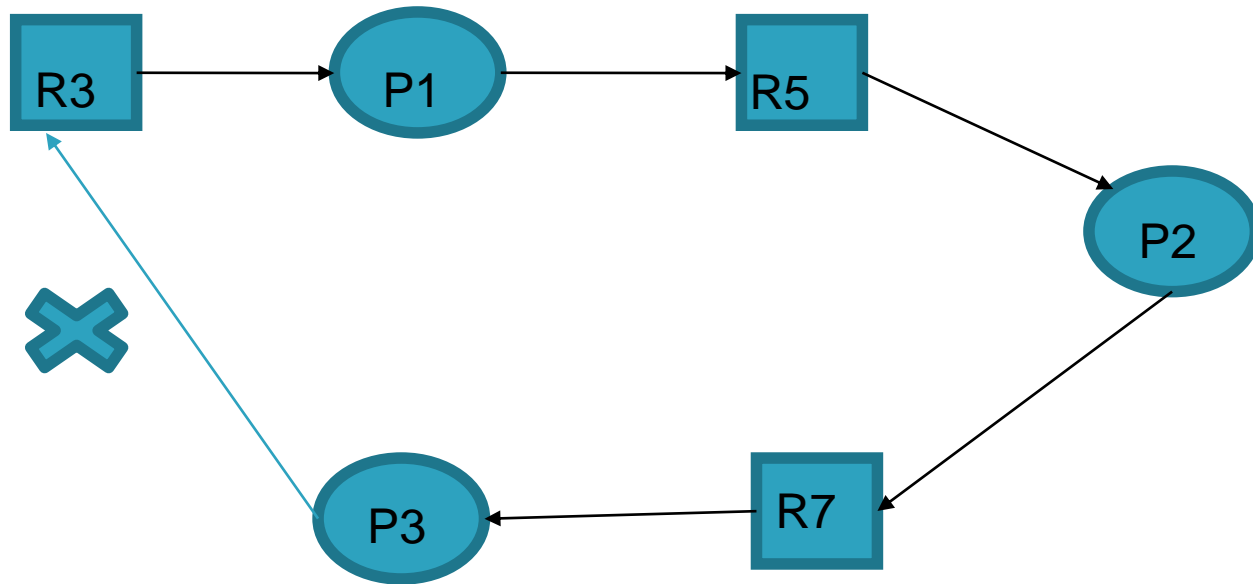
Deadlock Prevention

- For instance a process holding 2 resources numbered 3 and 7, may release the resource having number 7 before requesting a resource having number 5.
- The ordering of resources is decided according to the natural usage pattern of the resources.
- For eg : Since a tape drive is usually needed before the printer, it would be reasonable to assign a lower number to the tape drive than to the printer.

Drawbacks :

This natural ordering is not the same for all jobs.





Cycle is not completed

Deadlock Prevention

4. No Preemption

- ▶ No preemption is a necessary condition for a deadlock.
- ▶ To ensure that this condition does not hold we can use the following protocol.

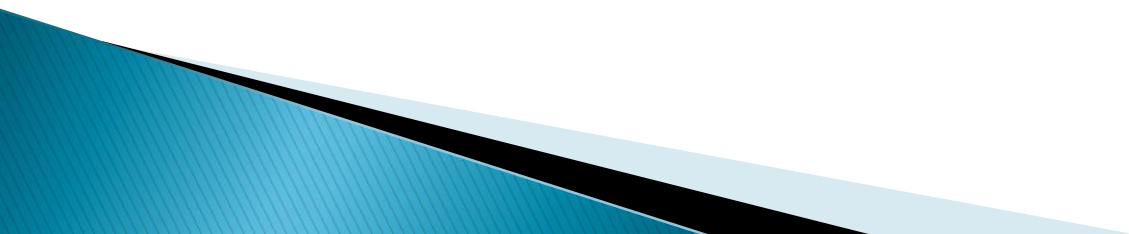
1. When a process requests for a resource that is not currently available, all the resources held by the process are taken away(preempted) from it and the process is blocked. The process is unblocked when the resource requested by it and the resources preempted from it becomes available and can be allocated it.

Deadlock Prevention

2. When a process requests a resource that is not currently available, the system checks if the requested resource is currently held by a process that is blocked, waiting for some other resource. If so the requested resource is taken away from the waiting process and given to the requesting process. Otherwise , the requesting process is blocked and waits for the requested resource to become available.

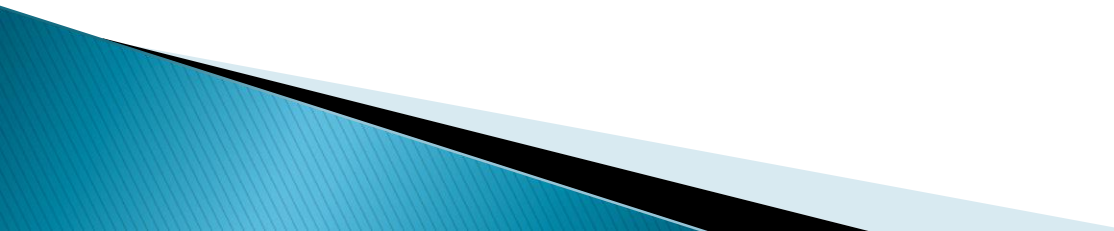
Some of the resources that this process is already holding may be taken away (preempted) from it while it is blocked, waiting for the allocation of the requested resource.

The process is unblocked when the resource requested by it and the resources preempted from it become available and can be allocated it.



Deadlock Avoidance

Dijkstra's Banker algorithm for deadlock

- ▶ Dijkstra designed an algorithm known as bankers algorithm.
 - ▶ This algorithm takes analogy of a bank, where customers can take loans from the bank.
 - No customer is granted loan exceeding bank's total capital.
 - All customers are given a maximum credit limit
 - No customer is allowed to borrow over the limit.
 - The sum of all loans won't exceed bank's total capital.
- 

Deadlock Avoidance

- ▶ Predicts the future state of the system for avoiding allocations that can eventually lead to a deadlock.

Steps for deadlock avoidance :

1. When a process requests for a resource, even if the resource is available for allocation, it is not immediately allocated to the process. Rather **the system simply assumes that the request is granted.**
2. With this assumption made and the advance knowledge of the resource usage of processes, the system performs some analysis to decide whether granting the process's request is **safe** or **unsafe**.
3. If **safe** , the **resource is allocated** to the process, **otherwise the request is deferred.**

Deadlock Avoidance

- ▶ Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need.

The banker's algorithm has 2 parts.

- ▶ The first part is a Safety Test algorithm that checks the current state of the system for its safe state.
- ▶ The second part is resource request handling algorithm that verifies whether the requested resources, when allocated to the process, affects the safe state.

If it does, the request is denied.

IN THIS WAY BANKERS'S ALGORITHM AVOIDS DEADLOCK.

- ▶ When a process requests an available resource, system must decide if immediate allocation leaves the system in a **safe state**
- ▶ System is in a safe state if there exists a **safe sequence** of all processes
- ▶ Sequence $\langle P_0, P_1, \dots, P_n \rangle$ is **safe** if for each P_i , the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j with $j < i$
 - If P_i 's resource needs are not immediately available, then P_i can wait until all P_j have finished
 - When P_j finishes, P_i can obtain needed resources, execute, return allocated resources, and terminate
 - When P_i terminates, P_{i+1} can obtain its needed resources, and so on

Deadlock Avoidance

- ▶ If system is in **safe state** \Rightarrow **no deadlock**
- ▶ If system is in **unsafe state** \Rightarrow **possibility of deadlock**
- ▶ **Avoidance** \Rightarrow ensure that system will never enter an unsafe state

Ex. There are 8 units of a particular resource type for which 3 processes P1, P2 and P3 competing

	P1	P2	p3
Max	4	5	6
Holds	2	2	2

Free =2

	P1	P2	p3
Max	4	5	6
Holds	4	2	2

Free =0

	P1	P2	p3
Max	-	5	6
Holds	0	2	2

Free =4

	P1	P2	p3
Max	-	5	6
Holds	0	5	2

Free =1

	P1	P2	p3
Max	-	5	6
Holds	0	2	6

Free =0

	P1	P2	p3
Max	-	-	6
Holds	0	0	2

Free =6

	P1	P2	p3
Max	-	5	-
Holds	0	2	0

Free =6

Deadlock Avoidance

Hence the analysis performed shows that this state is safe because there exists a sequence of allocations that allows all processes to complete.

Hence the two safe sequences are (P1,P2,P3) and (P1, P3, P2).

Hence the initial state is a safe state because it avoids deadlocks

Example of a safe state

- Suppose system has 12 instances of a particular resource

	<u>Maximum Needs</u>	<u>Currently Allocated</u>	<u>Need</u>
P0	10	5	5
P1	4	2	2
P2	9	2	7

(i.e. currently 3 free)

- At time t_0 , system is in safe state because there exists a process sequence that would allow each process to immediately be allocated its maximum needed resources. .. For example $\langle P1, P0, P2 \rangle$
- P1 could need 2 more, which results in 1 free, then P1 releases resulting in 5 free, then P0 could get all its maximum, then P1 releases, for a total of 10 free, then P2 could get his maximum.
- No suppose that at time t_1 , P2 requests 1 more. The question is, if we grant P1 1 more, will the resulting state be safe ? (i.e. will there be a safe sequence ?)

Example of an unsafe state

- Suppose system has 12 instances of a particular resource

	<u>Maximum Needs</u>	<u>Currently Allocated</u>	<u>Need</u>
P0	10	5	5
P1	4	2	2
P2	9	3	6

(i.e. currently 2 free)

- What are some potential process sequences now ?
- <P0, P1, P2> ... P0 can't get next 5 b/c only 2 free
- <P1, P2, P0> ... P1 can get next 2, run, (4 free), but then P2 can't
- <P2, P0, P1> ... P2 can't get next 6, b/c only 2 free
- <P0, P2, P1> ... P0 can't get next 5 b/c only 2 free
- <P1, P0, P2> ... P1 can get next 2, run, (4 free), but then P0 can't
- <P2, P1, P0> ... P2 can't get next 6 b/c only 2 free
- Therefore, granting P2 the extra resource, leads to unsafe state, which *COULD* lead to deadlock state.

Deadlock Avoidance

- ▶ **Dijkstra's Banker's algorithm** for deadlock avoidance in multiple instances of resources.

1. DATA STRUCTURES :

Total resources in a system

Stores the total number of resources in a system.

$\text{Total_Res}[i]=j$

It means there are j instances of resource type R_i in the system.

2. **Maximum demand of a process**

Whenever a process enters the system, it declares its maximum demand of resources.

$\text{Max}[i,j]=k$

It means, a process P_i has a total demand of k instances of resource type R_j .

Deadlock Avoidance

3. Current Allocation of instances of each type of resource

It indicates the current allocation status of all resource types to various processes in the system.

$\text{Alloc}[i,j]=k$

It means that process P_i is allocated k instances of resource type R_j .

4. Number of available resources.

Stores the current available instances of each resource type.

$\text{Av}[i]=j$

It means that j instances of resource type R_i are available.

It is the difference between the total resources and the allocated resources.

$\text{Av}[i] = \text{Total_Res}[i] - \sum \text{all processes } \text{Alloc}[i]$ where i is the resource type R_i .

Deadlock Avoidance

5. Current need of a process

Indicates the current remaining resource need of each process.

$$\text{Need}[i,j]=k$$

It means process P_i may require k more instances of resource type R_j , so that it can complete execution.

$$\text{Need}[i,j]=\text{Max}[i,j]-\text{Alloc}[i,j]$$

6. Request for a process

Stores the resource request for process P_i .

$$\text{Req}_i[j]=k$$

It means that process P_i has requested k instances of resource type R_j .

Deadlock Avoidance

Safety Test algorithm

Let `Current_Avail` and `Marked` be two vectors of length n and p respectively.

Safe string is an array to store process IDs.

1. `Current_Avail = Av`
2. Initialize `Marked` as :
for($i=1$; $i \leq p$; $i++$)
 `Marked[i] = false`;
3. Find a process P_i such that
 $Need_i \leq Current_Avail$ and `Marked[i] = false`
 If no such i exists then goto step 5

Deadlock Avoidance

4. if(found)

{

Current_Avail= Current_Avail+ Alloci

Marked[i]=true

Save the process number in SafeString[]

got to step 3

}

5. if (Marked[i]=true) for all processes, the system is in the safe state.

Print SafeString.

Otherwise the system is unsafe state and may be deadlocked.

Deadlock Avoidance

Resource Request handling algorithm

Let Req_i be the vector to store the resource request for process P_i .

1. If $Req_i \leq Need_i$, goto step2.
else raise an error condition, since the process has exceeded its maximum claim.
2. If $Req_i \leq Av$, goto step3.
else P_i must wait , since the resources are not available.

Deadlock Avoidance

3. Have the system pretend to have allocated the requested resources to process P_i , by modifying the state as follows :

$$Av = Av - Req_i$$

$$Alloc_i = Alloc_i + Req_i$$

$$Need_i = Need_i - Req_i$$

Execute the safety test , assuming the state has been changed

If(state is safe)

Change the state in actual and the resource will be allocated to the process

otherwise

keep the state unchanged and do not allocate the resource to the process.

Deadlock Avoidance

- ▶ Consider a system with the following information. Determine whether the system is in safe state.
- ▶ Total Res

R1	R2	R3
15	8	8

Process	Max			Alloc		
	R1	R2	R3	R1	R2	R3
P1	5	6	3	2	1	0
P2	8	5	6	3	2	3
P3	4	8	2	3	0	2
P4	7	4	3	3	2	0
P5	4	3	3	1	0	1

Av

R1	R2	R3
3	3	2

Find Need matrix

Process	Max			Alloc			Need		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	5	6	3	2	1	0	3	5	3
P2	8	5	6	3	2	3	5	3	3
P3	4	8	2	3	0	2	1	8	0
P4	7	4	3	3	2	0	4	2	3
P5	4	3	3	1	0	1	3	3	2

Safety Test Algorithm to find whether the new state is safe :

Current_Avail = Av = [3 3 2]

Marked	
P1	false
P2	false
P3	false
P4	false
P5	false

Find a sequence of processes such that each process satisfies the criteria $Need \leq Current_Avail$.

A process after finishing its execution, must release all the resources it is holding, so that the next process can avail them as per its need.

Process found	Current_Avail	SafeString []	Marked []
P5	$[3 \ 3 \ 2] + [1 \ 0 \ 1] = [4 \ 3 \ 3]$	P5	Marked[P5]=true
P4	$[4 \ 3 \ 3] + [3 \ 2 \ 0] = [7 \ 5 \ 3]$	P5,P4	Marked[P4]=true
P1	$[7 \ 5 \ 3] + [2 \ 1 \ 0] = [9 \ 6 \ 3]$	P5,P4,P1	Marked[P1]=true
P2	$[9 \ 6 \ 3] + [3 \ 2 \ 3] = [12 \ 8 \ 6]$	P5,P4,P1,P2	Marked[P2]=true
P3	$[12 \ 8 \ 6] + [3 \ 0 \ 2] = [15 \ 8 \ 8]$	P5,P4,P1,P2,P3	Marked[P3]=true

Since Marked value for all processes is true. Therefore, the system is in safe state.

The Safe String of processes is {P5,P4,P1,P2,P3}

i.e. The processes can be scheduled in this order, to have a safe state in the system.

Deadlock Avoidance

At time t_1 , if P4 requests 2 more instances of R1 and 2 instances of R3.

- ▶ Request vector $[2 \ 0 \ 2]$
- ▶ Executing the request handling algorithm, to find out whether the request can be granted.
- ▶ First check if $\text{Req}_4 \leq \text{Need}_4$. $[2 \ 0 \ 2] \leq [4 \ 2 \ 3]$
- ▶ Check if $\text{Req}_4 \leq \text{Av}$. $[2 \ 0 \ 2] \leq [3 \ 3 \ 2]$
- ▶ Assume to allocate these resources and update the following
$$\begin{aligned}\text{Av} &= \text{Av} - \text{Req}_4 = [3 \ 3 \ 2] - [2 \ 0 \ 2] = [1 \ 3 \ 0] \\ \text{Alloc}_4 &= \text{Alloc}_4 + \text{Req}_4 = [3 \ 2 \ 0] + [2 \ 0 \ 2] = [5 \ 2 \ 2] \\ \text{Need}_4 &= \text{Need}_4 - \text{Req}_4 = [4 \ 2 \ 3] - [2 \ 0 \ 2] = [2 \ 2 \ 1]\end{aligned}$$



Av

R1	R2	R3
1	3	0

Find Need matrix

Process	Max			Alloc			Need		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	5	6	3	2	1	0	3	5	3
P2	8	5	6	3	2	3	5	3	3
P3	4	8	2	3	0	2	1	8	0
P4	7	4	3	5	2	2	2	2	1
P5	4	3	3	1	0	1	3	3	2

Check again whether the system is in a safe state

Current_Avail=Av=[1 3 0].

There is no process whose $\text{Need} \leq \text{Current_Avail}$.

Hence no process can be started and thus the state would be unsafe if the request P4 is granted

Ex: 2 Consider the following snapshot of a system:

	Allocation	Max	Available
	ABCD	ABCD	ABCD
P0	0012	0012	1520
P1	1000	1750	
P2	1354	2356	
P3	0632	0652	
P4	0014	0656	

Answer the following questions using the banker's algorithm:

- What is the content of the matrix Need?
- Is the system in a safe state?
- If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately?

Total resources 3 14 12 12

	Allocation	Max	Need
	ABCD	ABCD	ABCD
P0	0012	0012	0000
P1	1000	1750	0750
P2	1354	2356	1002
P3	0632	0652	0020
P4	0014	0656	0642

Available

ABCD

1520

$$P0 - 1 \ 5 \ 2 \ 0 + 0 \ 0 \ 1 \ 2 = 1 \ 5 \ 3 \ 2$$

$$P2 - 1 \ 5 \ 3 \ 2 + 1 \ 3 \ 5 \ 4 = 2 \ 8 \ 8 \ 6$$

$$P3 - 2 \ 8 \ 8 \ 6 + 0 \ 6 \ 3 \ 2 = 2 \ 14 \ 11 \ 8$$

$$P4 - 2 \ 14 \ 11 \ 8 + 0 \ 0 \ 1 \ 4 = 2 \ 14 \ 12 \ 12$$

$$P1 - 2 \ 14 \ 12 \ 12 + 1 \ 0 \ 0 \ 0 = 3 \ 14 \ 12 \ 12$$

Yes – System is in the safe state

Total resources 3 14 12 12

	Allocation	Max	Need
	ABCD	ABCD	ABCD
P0	0012	0012	0000
P1	1000	1750	0750
P2	1354	2356	1002
P3	0632	0652	0020
P4	0014	0656	0642

Available

ABCD

1520

Request from P1 [0 4 2 0]

Check if $Req \leq Need$ $[0 \ 4 \ 2 \ 0] \leq 0750$

Check if $Req \leq Av$ $[0 \ 4 \ 2 \ 0] \leq 1520$

$Av = Av - Req1 = [1 \ 5 \ 2 \ 0] - [0 \ 4 \ 2 \ 0] = [1 \ 1 \ 0 \ 0]$

$Alloc \ 1 = Alloc \ 1 + Req1 = [1 \ 0 \ 0 \ 0] + [0 \ 4 \ 2 \ 0] = [1 \ 4 \ 2 \ 0]$

$Need1 = Need1 - Req1 = [0 \ 7 \ 5 \ 0] - [0 \ 4 \ 2 \ 0] = [0 \ 3 \ 3 \ 0]$

Perform the safety test

	Allocation	Max	Need
	ABCD	ABCD	ABCD
P0	0012	0012	0000
P1	1420	1750	0330
P2	1354	2356	1002
P3	0632	0652	0020
P4	0014	0656	0642

Available

ABCD

1 1 0 0

$P0 - 1\ 1\ 0\ 0 + 0\ 0\ 1\ 2 = 1\ 1\ 1\ 2$

$P2 - 1\ 1\ 1\ 2 + 1\ 3\ 5\ 4 = 2\ 4\ 6\ 6$

$P3 - 2\ 4\ 6\ 6 + 0\ 6\ 3\ 2 = 2\ 10\ 9\ 8$

$P4 - 2\ 10\ 9\ 8 + 0\ 0\ 1\ 4 = 2\ 10\ 10\ 12$

$P1 - 2\ 10\ 10\ 12 + 1\ 4\ 2\ 0 = 3\ 14\ 12\ 12$

Safe sequence P0 P2 P3 P4 P1

Note : Solution is not unique

Request can be granted

Example of Banker's Algorithm

- ▶ Ex :3
 - ▶ 5 processes P_0 through P_4 ;
3 resource types:
A (10 instances), B (5 instances), and C (7 instances)
- Snapshot at time T_0 :

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>
	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	

Is the state of the system safe ?

Example of Banker's Algorithm

	<u>Need</u>		
	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

$$P1-3\ 3\ 2 + 2\ 0\ 0 = 5\ 3\ 2$$

$$P3-5\ 3\ 2 + 2\ 1\ 1 = 7\ 4\ 3$$

$$P4-7\ 4\ 3 + 0\ 0\ 2 = 7\ 4\ 5$$

$$P0-7\ 4\ 5 + 0\ 1\ 0 = 7\ 5\ 5$$

$$P2-7\ 5\ 5 + 3\ 0\ 2 = 10\ 5\ 7$$

The system state: Safe

<p1 P3 P4 P0 P2> is a possible safe sequence

Example of Banker's Algorithm

- ▶ P_1 now requests (1,0,2).
 - Request \leq Need
 - Request \leq Available
 - Execute safety algorithm:
 - 1 0 2 \leq 1 2 2
 - 1 0 2 \leq 3 3 2

We pretend that this request has been fulfilled...

	<u>Allocation</u>			<u>Need</u>			<u>Available</u>		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	4	3	2	3	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

$$P1 - 2\ 3\ 0 + 3\ 0\ 2 = 5\ 3\ 2$$

$$P3 - 5\ 3\ 2 + 2\ 1\ 1 = 7\ 4\ 3$$

$$P4 - 7\ 4\ 3 + 0\ 0\ 2 = 7\ 4\ 5$$

$$P0 - 7\ 4\ 5 + 0\ 1\ 0 = 7\ 5\ 5$$

$$P2 - 7\ 5\ 5 + 3\ 0\ 2 = 10\ 5\ 7$$

Executing safety algorithm shows that sequence $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfies safety requirement hence P_1 's request can be granted.

When the system is in this state-----→

- ▶ Can request for (3,3,0) by P_4 be granted?
- ▶ Can request for (0,2,0) by P_0 be granted?

Example of Banker's Algorithm

- ▶ Request for (3,3,0) by P4 cannot be granted, since the resources are not available.
- ▶ A request for (0 2 0) by P0 cannot be granted, even though the resources are available , since the resulting state is unsafe

Example of Banker's Algorithm

Example 4:

- ▶ 4 types resources R1 R2 R3 R4 with 6 8 10 and 12 instances.
- ▶ 5 processes(P1...P5)

Max

	R1	R2	R3	R4
P1	3	1	2	5
P2	3	2	5	7
P3	2	6	3	1
P4	5	4	9	2
P5	1	3	8	9

Alloc

	R1	R2	R3	R4
P1	1	0	2	1
P2	1	1	2	0
P3	1	2	3	1
P4	1	1	1	1
P5	1	0	2	2

How does the OS react if P5 requests 7 instances of R4.
Can this request be granted?

Example of Banker's Algorithm

Need

	R1	R2	R3	R4
P1	2	1	0	4
P2	2	1	3	7
P3	1	4	0	0
P4	4	3	8	1
P5	0	3	6	7

Example of Banker's Algorithm

▶ $Av < 1 \ 4 \ 0 \ 7 >$

If P5 requests 7 instances of R4

$Req < 0 \ 0 \ 0 \ 7 >$

$Req \leq Need_5$ ---- yes

$Req \leq Av$ ----- yes

Assume the request is granted

Av becomes ---- $< 1 \ 4 \ 0 \ 0 >$

Example of Banker's Algorithm

Alloc

	R1	R2	R3	R4
P1	1	0	2	1
P2	1	1	2	0
P3	1	2	3	1
P4	1	1	1	1
P5	1	0	2	9

Need

	R1	R2	R3	R4
P1	2	1	0	4
P2	2	1	3	7
P3	1	4	0	0
P4	4	3	8	1
P5	0	3	6	0

$Av < 1 \ 4 \ 0 \ 0 >$

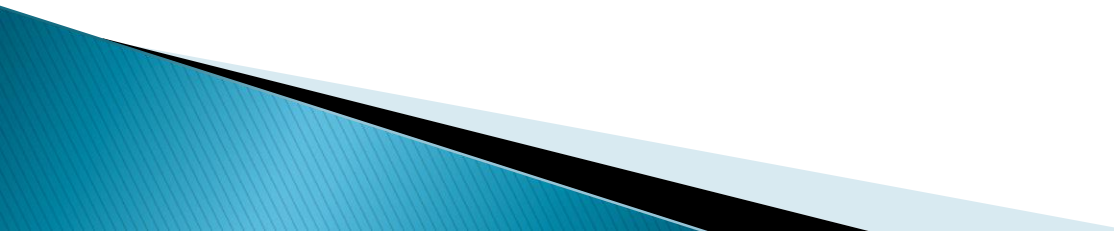
$P3 - 1 \ 4 \ 0 \ 0 + 1 \ 2 \ 3 \ 1 = 2 \ 6 \ 3 \ 1$

No process can proceed after this.

System would be in an unsafe state .

Hence request cannot be granted.

Deadlock Detection

- ▶ In deadlock detection we don't take the prior information from each process that is the Max.
 - ▶ But we allow processes to request and be allocated the resources and periodically invoke the deadlock detection algorithm
 - ▶ i.e if resources are available, they are allocated to a process.
- 

Deadlock Detection

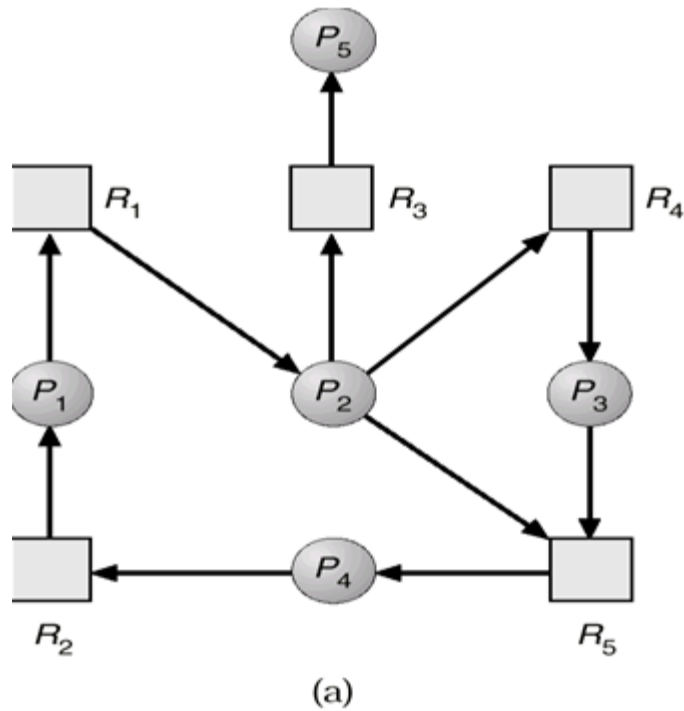
- ▶ If a system is not able to implement neither deadlock prevention nor a deadlock avoidance algorithm, it may lead to a deadlock situation.
- ▶ For deadlock detection, the system must provide
 - An algorithm that examines the state of the system to detect whether a deadlock has occurred
 - And an algorithm to recover from the deadlock
- ▶ Deadlock detection has 2 parts :
 1. Detection when there is single instance of each resource.
 2. Detection when there are multiple instance of each resource.

Deadlock Detection

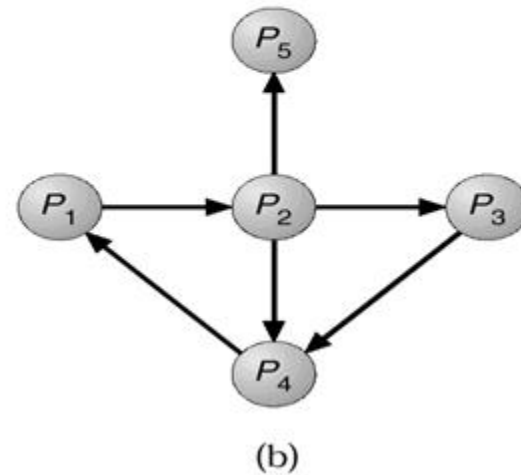
Single Instance of Each Resource Type

- ▶ Requires the creation and maintenance of a wait-for graph
 - Consists of a variant of the resource-allocation graph
 - The graph is obtained by **removing** the resource nodes from a resource-allocation graph and **collapsing** the appropriate edges
 - Consequently all nodes are processes
 - $P_i \rightarrow P_j$ if P_i is waiting for P_j .
- ▶ Periodically invoke an algorithm that searches for a cycle in the graph
 - If there is a cycle, there exists a deadlock

Deadlock Detection



Resource-Allocation Graph



Corresponding wait-for graph

Deadlock Detection

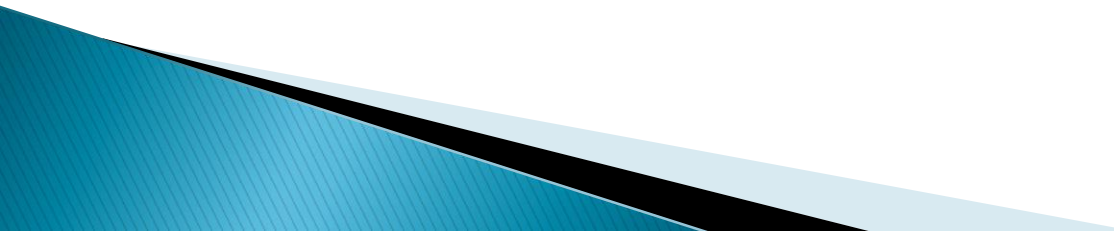
Several instances of resource type

- ▶ The WFG scheme is not applicable to a resource allocation system with multiple instances of each resource type.

Algorithm for deadlock detection

- ▶ Data structures similar to those in banker's algorithm
 - Available(Av)** A vector of length m indicates the number of available resources of each type.
 - Allocation(Alloc)**: An $p \times r$ matrix defines the number of resources of each type currently allocated to each process.
 - Request(Req)**: An $p \times r$ matrix stores the current request of each process.
 - $Req[i,j]=k$ means process P_i is requesting k instances of resource type R_j .

Deadlock Detection

- ▶ Goal of the detection algorithm is to check if there is any sequence in which all current requests can be met.
 - ▶ Note: If a process P_i 's request can be met, then P_i can potentially run to completion, and release all the resources it currently holds.
 - ▶ So for detection purpose, P_i 's current allocation can be added to current available.
- 

Deadlock Detection

- ▶ Detection algorithm

Let Current_Avail and Marked be two vectors of length n and p respectively.

Safe string is an array to store process IDs.

1. Current_Avail = Av
2. Initialize Marked as :
for($i=1$; $i \leq p$; $i++$)
If $Alloc_i \neq 0$ then Marked $[i] = \text{false}$; else
Marked $[i] = \text{true}$.
3. Find a process P_i such that
 $Req_i \leq \text{Current_Avail}$ and Marked $[i] = \text{false}$
If no such i exists then goto step 5

Deadlock Detection

4. if(found)

{

Current_Avail= Current_Avail+ Alloci

Marked[i]=true

Save the process number in SafeString[]

got to step 3

}

5. if (Marked[i]=false) for some i, then the system is in the deadlock state.

Moreover if (Marked[i]=false) then process P_i is deadlocked.

Note : Since $Req_i \leq Current_Avail$ for P_i , P_i is not involved in deadlock.

Example of Detection Algorithm

- Five processes P_0 through P_4 ; three resource types A (7 instances), B (2 instances), and C (6 instances)
- Snapshot at time T_0 :

	<u>Allocation</u>	<u>Request</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

$P_0 - 0\ 0\ 0 + 0\ 1\ 0 = 0\ 1\ 0$

$P_2 - 0\ 1\ 0 + 3\ 0\ 3 = 3\ 1\ 3$

$P_3 - 3\ 1\ 3 + 2\ 1\ 1 = 5\ 2\ 4$

$P_4 - 5\ 2\ 4 + 0\ 0\ 2 = 5\ 2\ 6$

$P_1 - 5\ 2\ 6 + 2\ 0\ 0 = 7\ 2\ 6$

Sequence $\langle P_0, P_2, P_3, P_4, P_1 \rangle$ will result in $Finish[i] = \text{true}$ for all i

No deadlock

Deadlock Detection

Suppose that process P2 makes an additional request for an instance of type C. The request matrix ----→

	Request		
	A	B	C
P0	0	0	0
P1	2	0	2
P2	0	0	1
P3	1	0	0
P4	0	0	2

System is deadlocked.

Although resources held by P0 are reclaimed , the number of available resources is not sufficient to fulfill the requests of the other processes.

▶ Ex: 2

▶ 4 resources total res=(4 2 3 1)

A= 0 0 1 0

2 0 0 1

0 1 2 0

R= 2 0 0 1

1 0 1 0

2 1 0 0

Av=(2 1 0 0)

- ▶ $P3 - 2\ 1\ 0\ 0 + 0\ 1\ 2\ 0 = 2\ 2\ 2\ 0$
- ▶ $P2 - 2\ 2\ 2\ 0 + 2\ 0\ 0\ 1 = 4\ 2\ 2\ 1$
- ▶ $P1 - 4\ 2\ 2\ 1 + 0\ 0\ 1\ 0 = 4\ 2\ 3\ 1$

SAFE state

Current state is not deadlocked

Recovery from deadlock

1. Process Termination

- ▶ Two methods used.
- ▶ Both the methods reclaims all resources allocated to the terminated processes.

a) Abort all deadlocked processes :

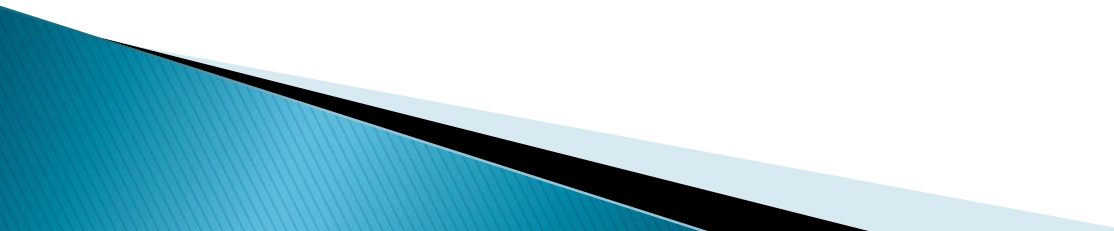
- ▶ This method clearly will break the deadlock cycle, but at great expense; the deadlocked processes may have computed for along time, and the results of these partial computations must be discarded and probably will have to be recomputed later

Recovery from deadlock

b) Abort one process at a time until the deadlock cycle is eliminated.

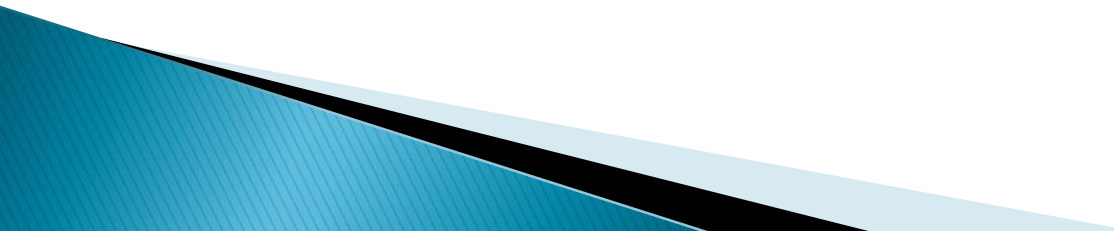
This method incurs considerable overhead, since, after each process is aborted, a deadlock detection algorithm must be invoked to determine whether any processes are still deadlocked.

If the partial termination method is used, then we must determine which deadlocked process (or processes) should be terminated.



Recovery from deadlock

Many factors may affect which process is chosen, including:

- ▶ What the priority of the process is
 - ▶ How long the process has computed and how much longer the process will compute before completing its designated task.
 - ▶ How many and what type of resources the process has used (for example, whether the resources are simple to preempt)
 - ▶ How many more resources the process needs in order to complete
 - ▶ Whether the process is interactive or batch
- 

Recovery from deadlock

2. Resource Preemption

- ▶ To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.
- ▶ If preemption is required to deal with deadlocks, then three issues need to be addressed:

1. Selection a victim.

- ▶ Which resources and which processes are to be preempted? As in process termination, we must determine the order of preemption to minimize cost. Cost factors may include such parameters as the number of resources a deadlocked process is holding and the amount of time the process has thus far consumed during its execution.

Recovery from deadlock

2. Rollback

- ▶ If we preempt a resource from a process, what should be done with that process? Clearly, it cannot continue with its normal execution; it is missing some needed resource. We must roll back the process to some safe state and restart it from that state.

3. Starvation

- ▶ How do we ensure that starvation will not occur? That is, how can we guarantee that resources will not always be preempted from the same process?
- ▶ It may happen that the same process is always picked as a victim. As a result, this process never completes its designated task. We must ensure that a process can be picked as a victim only a (small) finite number of times. The most common solution is to include the number of rollbacks in the cost factor.