

# Operating System

Unit #5



**Marwadi**  
University

Department of  
Computer Engineering

**Operating System**

Sem 4

01CE1401

4 Credits

Prof. Urvi Bhatt

# Principles of Deadlock

- Deadlock - system model
- Deadlock and its characterization with example
- Deadlock prevention techniques with example
- Detection and avoidance of a deadlock
- Methods to get recovery form deadlock

# Outline

- Three problems
- Requirement of File System
- File concept
  - File Naming
  - File Attributes
  - File operation
  - File Structure
  - File Types
  - File Access Methods
  - File Allocation Methods
- Directory
  - Operations on Directory
  - Directory structure
  - Directory Implementation
- Free Space Management

# Introduction To Deadlocks

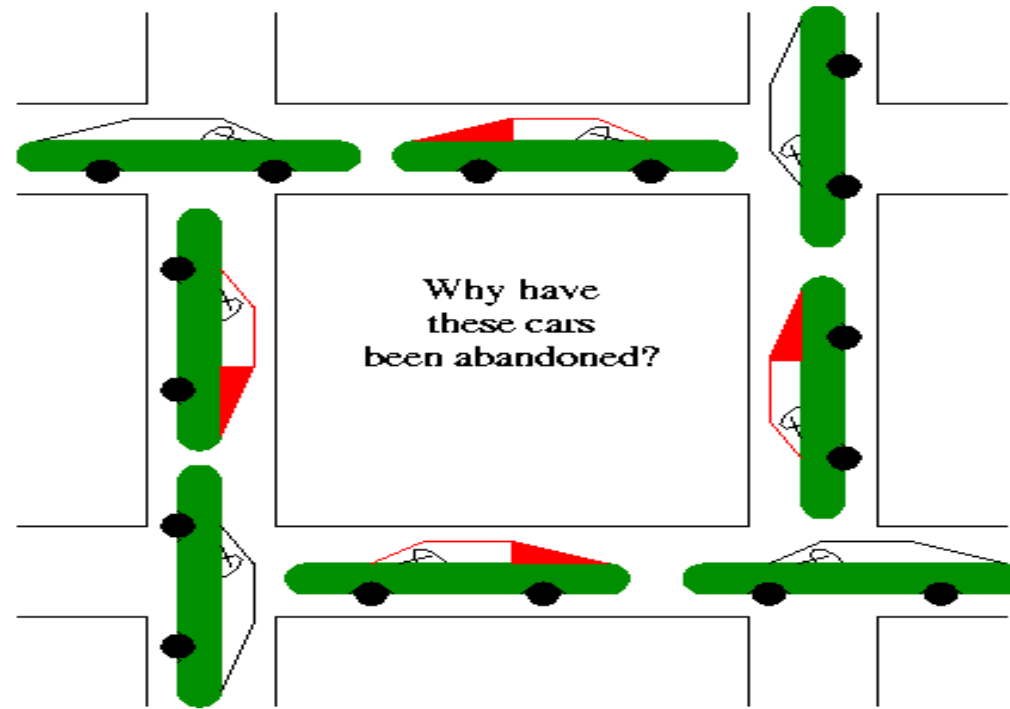
## Definition:

- A deadlock consists of a set of blocked processes, each holding a resource and waiting to acquire a resource held by another process in the set.

## Example:

- two processes each want to record a scanned document on a CD.
- process 1 requests for scanner & gets it
- process 2 requests for CD writer & gets it
- process 1 requests CD writer but is blocked
- process 2 requests scanner but is blocked.
- At this point both processes are blocked and will remain so forever,
- This situation is called a **deadlock**

# Introduction To Deadlocks



We can summarize deadlock as,  
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.

# Resources

## Definition

- Deadlock can occur when processes have been granted exclusive access to devices, files and so forth. These objects are referred to as resources.

## Resource can be

- Hardware or Piece of information

## Resource can be categorized as

- Preemptable Resources or Non-preemptable Resources

# Resources

## Preemptable Resources

- A preemptable resource is one that can be taken away from the process with no side effect.
- Memory is an example of a preemptable resource.
- Eg: a system with 32 MB of user memory, one printer, and two 32-MB processes that each want to print something.
- Process *A* requests and gets the printer, then starts to compute the values to print. Before it has finished with the computation, it exceeds its time quantum and is swapped out.
- Process *B* now runs and tries to print, unsuccessfully, to acquire the printer.
- Potentially, we now have a deadlock situation, because *A* has the printer and *B* has the memory, and neither can proceed without the resource held by the other.
- Fortunately, it is possible to preempt (take away) the memory from *B* by swapping it out and swapping *A* in. Now *A* can run, do its printing, and then release the printer. No deadlock occurs.

# Resources

## Non-preemptable Resources

- A **nonpreemptable resource**, in contrast, is one that cannot be taken away from its current owner without causing the computation to fail.
- If a process has begun to burn a CD-ROM, suddenly taking the CD recorder away from it and giving it to another process will result in a garbled CD, CD recorders are not preemptable at an arbitrary moment
- In general, deadlocks involve nonpreemptable resources.
- Potential deadlocks that involve preemptable resources can usually be resolved by reallocating resources from one process to another.
- Thus our treatment will focus on non preemptable resources.



**Resources:**  
Sequence of  
events to use

Request the resource

A green downward-pointing arrow connecting the first step to the second.

Use the resource

A yellow-green downward-pointing arrow connecting the second step to the third.

Release the resource

# Deadlock characteristics

Deadlock can arise if four conditions hold simultaneously. All of four these conditions must be present for a deadlock to occur. If one of them is absent, no deadlock is possible.

## Mutual exclusion

- only one process at a time can use a resource

## Hold and wait

- a process holding at least one resource is waiting to acquire additional resources held by other processes

# Deadlock characteristics

## No pre-emption

- a resource can be released only voluntarily by the process holding it after that process has completed its task

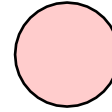
## Circular wait

- 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  $P_0$ .

# Deadlock Modeling :Resource- Allocation Graph



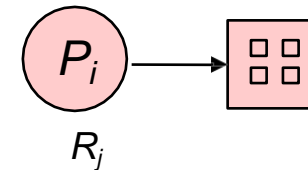
Process



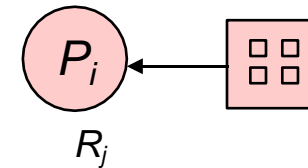
Resource Type with 4 instances



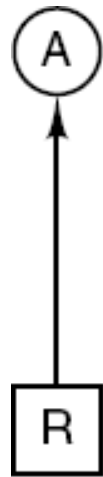
$P_i$  requests instance of  $R_j$



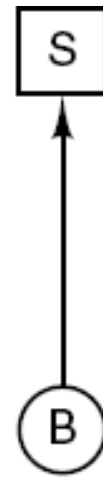
$P_i$  is holding an instance of  $R_j$



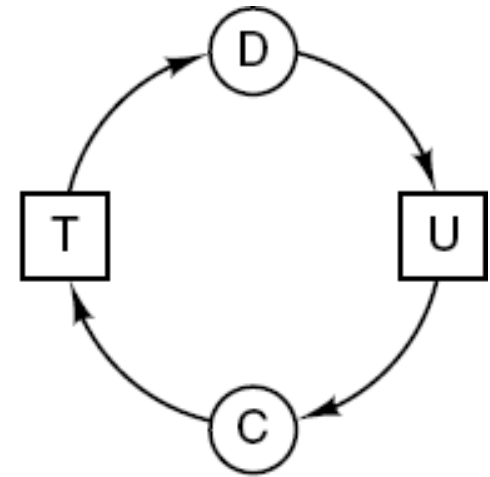
# Deadlock Modeling (1)



(a)



(b)



(c)

Figure : Resource allocation graphs.

(a) Holding a resource. (b) Requesting a resource. (c) Deadlock.

A hold R  
B hold S  
C hold T  
A request S  
B request T  
C request R  
So, Deadlock

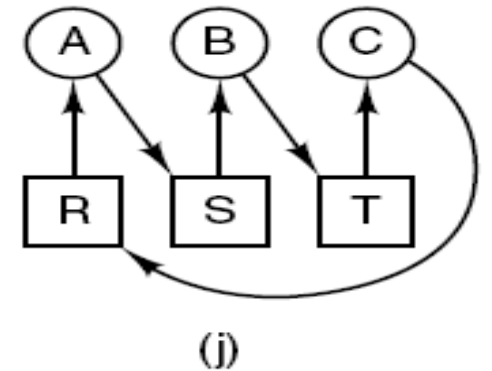
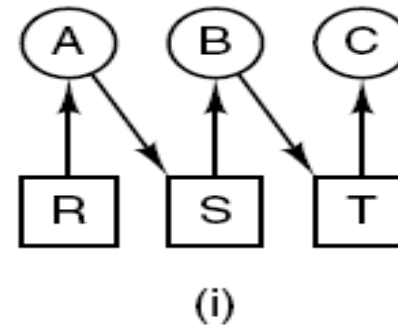
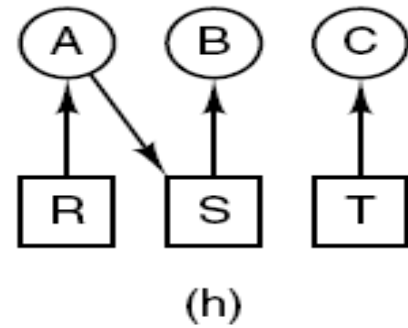
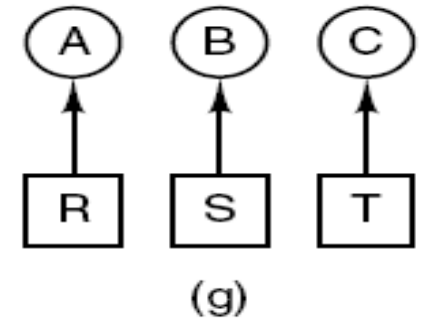
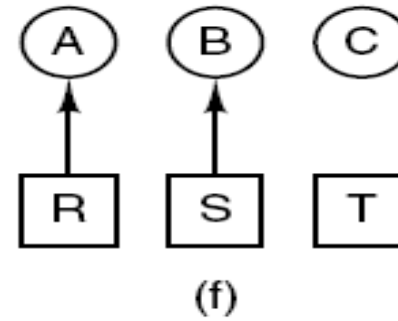
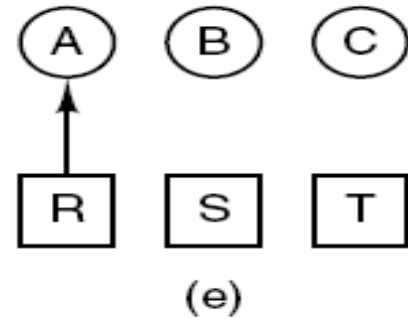
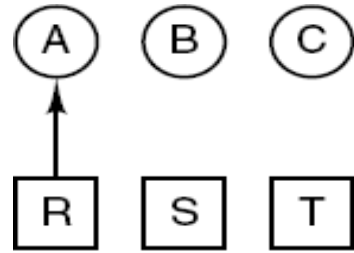
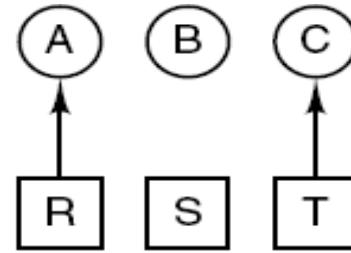


Figure : An example of how deadlock occurs and how it can be avoided.

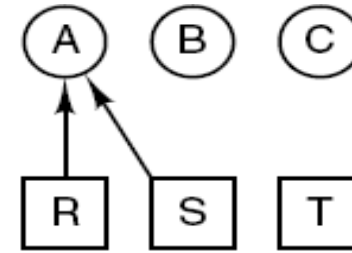
# Deadlock Modeling (3)



(l)



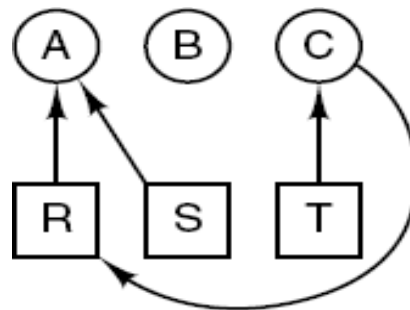
(m)



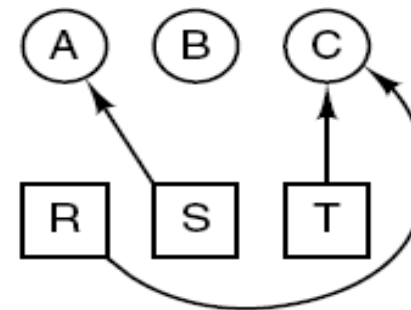
(n)

Figure : if OS knew that process 3 leads to deadlock then it could suspend process 3

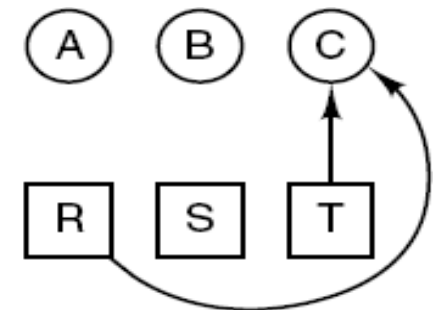
No deadlock:  
By releasing  
R from A,  
S from A



(o)



(p)



(q)

# Deadlock Modeling: Strategies for dealing with deadlocks:

## Ignorance

- Let deadlocks occur, just ignore it.

## Prevention

- Ensure that the system will *never* enter a deadlock state, by structurally negating one of the four required conditions.

## Avoidance

- Ensure that the system will *never* enter an unsafe state.

## Dynamic avoidance:

- by careful resource allocation.



# Deadlock Ignorance

- Deadlock Ignorance is the most widely used approach among all the mechanism.
- This is being used by many operating systems mainly for end user uses.
- In this approach, the Operating system assumes that deadlock never occurs.
- It simply ignores deadlock.
- This approach is best suitable for a single end user system where User uses the system only for browsing and all other normal stuff.
- In these types of systems, the user has to simply restart the computer in the case of deadlock.
- Windows and Linux are mainly using this approach.

# Deadlock Ignorance (Cont.)

## Ostrich Algorithm

- The ostrich algorithm means that the deadlock is simply ignored and it is assumed that it will never occur.
- This is done because in some systems the cost of handling the deadlock is much higher than simply ignoring it as it occurs very rarely.
- So, it is simply assumed that the deadlock will never occur and the system is rebooted if it occurs by any chance.

## *Deadlock Prevention*

- Attacking the mutual exclusion condition
- Attacking the hold and wait condition
- Attacking the no preemption condition
- Attacking the circular wait condition

# *Approaches to Deadlock Prevention*

Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

□Figure : Summary of approaches to deadlock prevention.

# Deadlock Prevention (Cont.)

## Mutual Exclusion

- Mutual section from the resource point of view is the fact that a resource can never be used by more than one process simultaneously which is fair enough but that is the main reason behind the deadlock.
- If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource.
- However, if we can be able to violate resources behaving in the mutually exclusive manner then the deadlock can be prevented.

# Deadlock Prevention (Cont.)

## Hold and Wait

- Hold and wait condition lies when a process holds a resource and waiting for some other resource to complete its task.
- Deadlock occurs because there can be more than one process which are holding one resource and waiting for other in the cyclic order.
- This can be implemented practically if a process declares all the resources initially.
- However, this sounds very practical but can't be done in the computer system because a process can't determine necessary resources initially.

## Deadlock Prevention (Cont.)

### No Preemption

- Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.
- This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has done till now can become inconsistent.
- Consider a printer is being used by any process. If we take the printer away from that process and assign it to some other process then all the data which has been printed can become inconsistent and ineffective and also the fact that the process can't start printing again from where it has left which causes performance inefficiency.

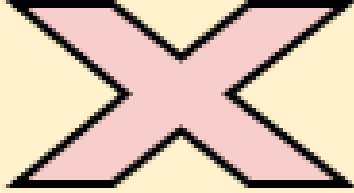
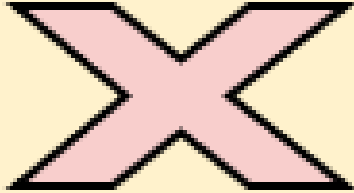
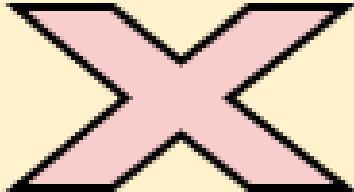
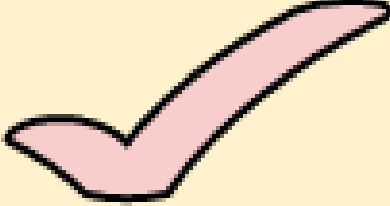
# Deadlock Prevention (Cont.)

## Circular Wait

- To violate circular wait, we can assign a priority number to each of the resource.
- A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.



## Deadlock Prevention (Cont.)

Condition	Approach	Is Practically Possible?
Mutual Exclusion	Spooling	
Hold and Wait	Request for all the resources initially	
No Preemption	Snatch all the resources	
Circular Wait	Assign priority to each resources and order resources numerically	

# Deadlock Avoidance

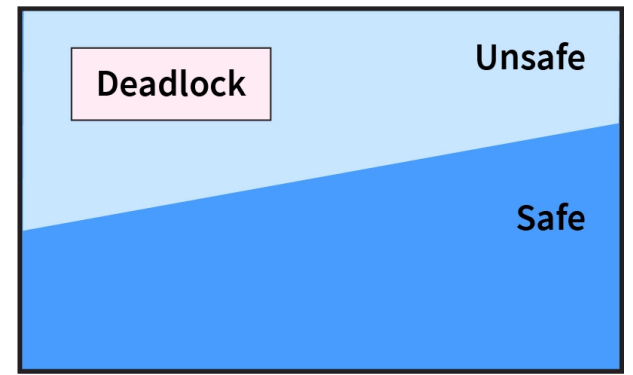
In most systems, however, resources are requested one at a time.

The system must be able to decide whether granting a resource is safe or not and only make the allocation when it is safe.

## Deadlock Avoidance (Cont.)

- In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system.
- The state of the system will continuously be checked for safe and unsafe states.
- In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.
- The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need.
- The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

# Deadlock Avoidance (Cont.)



- **Safe State**
- If Operating System is able to allocate or satisfy the maximum resource requirements of all the processes in any order then the system is said to be in Safe State.
- So safe state does not lead to Deadlock.

## Unsafe State -

- If Operating System is not able to prevent Processes from requesting resources which can also lead to Deadlock, then the System is said to be in an Unsafe State.
- Unsafe State does not necessarily cause deadlock it may or maynot causes deadlock.

# Banker's Algorithm

- Deadlock Avoidance is used by Operating System to avoid Deadlock by using Banker's algorithm.
- Banker's Algorithm checks for the s-state(safe state or unsafe state) before actually allocating the resources to the Processes.

## Banker's Algorithm (Cont.)

- Banker's Algorithm
  - Single resource instance of Each type
  - Multiple Resource instance of Each type

## Banker's Algorithm (Cont.)

When working with a banker's algorithm, it requests to know about three things:

- How much each process can request for each resource in the system. It is denoted by the [**MAX**] request.
- How much each process is currently holding each resource in a system. It is denoted by the [**ALLOCATED**] resource.
- It represents the number of each resource currently available in the system. It is denoted by the [**AVAILABLE/FREE**] resource.

# Banker's Algorithm (Cont.)

## Advantages

- It contains various resources that meet the requirements of each process.
- Each process should provide information to the operating system for upcoming resource requests, the number of resources, and how long the resources will be held.
- It helps the operating system manage and control process requests for each type of resource in the computer system.
- The algorithm has a Max resource attribute that represents indicates each process can hold the maximum number of resources in a system.



## Banker's Algorithm (Cont.)

### Disadvantages

- It requires a fixed number of processes, and no additional processes can be started in the system while executing the process.
- The algorithm does no longer allows the processes to exchange its maximum needs while processing its tasks.
- Each process has to know and state their maximum resource requirement in advance for the system.
- The number of resource requests can be granted in a finite time, but the time limit for allocating the resources is one year.

## Banker's Algorithm for Single resource instance of Each type

- Example: 3 process A, B and C are available with single resource R<sub>1</sub>. Free resource are 3. Find Safe Sequence and Total Instance for R<sub>1</sub>.

Process	Allocated	Max
A	3	9
B	2	4
C	2	7

Solution:

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

So, the content of Need Matrix is:

Process	Allocated	Max	Need
A	3	9	6
B	2	4	2
C	2	7	1

Safe Sequence is: B, C, A

## Banker's Algorithm for Multiple Resource instance of Each type

**Example:** Consider a system that contains five processes P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub>, P<sub>5</sub> and the three resource types A, B and C. Following are the resources types: A has 10, B has 5 and the resource type C has 7 instances.

**Answer the following questions using the banker's algorithm:**

What is the reference of the need matrix?

Determine if the system is safe or not.

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P <sub>1</sub>	0	1	0	7	5	3	3	3	2
P <sub>2</sub>	2	0	0	3	2	2			
P <sub>3</sub>	3	0	2	9	0	2			
P <sub>4</sub>	2	1	1	2	2	2			
P <sub>5</sub>	0	0	2	4	3	3			

$$\text{Need } [i] = \text{Max } [i] - \text{Allocation } [i]$$

Solution:

Process	Need		
	A	B	C
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	1

## Solution (Cont.)

**Ans. 2: Apply the Banker's Algorithm:**

Available Resources of A, B and C are 3, 3, and 2.

Now we check if each type of resource request is available for each process.

**Step 1:** For Process P<sub>1</sub>:

Need  $\leq$  Available

7, 4, 3  $\leq$  3, 3, 2 condition is **false**.

**So, we examine another process, P<sub>2</sub>.**

**Step 2:** For Process P<sub>2</sub>:

Need  $\leq$  Available

1, 2, 2  $\leq$  3, 3, 2 condition **true**

New available = available + Allocation

(3, 3, 2) + (2, 0, 0)  $\Rightarrow$  5, 3, 2

**Similarly, we examine another process P<sub>3</sub>.**

**Step 3:** For Process P<sub>3</sub>:

P<sub>3</sub> Need  $\leq$  Available

6, 0, 0  $\leq$  5, 3, 2 condition is **false**.

**Similarly, we examine another process, P<sub>4</sub>.**

**Step 4:** For Process P<sub>4</sub>:

P<sub>4</sub> Need  $\leq$  Available

0, 1, 1  $\leq$  5, 3, 2 condition is **true**

New Available resource = Available + Allocation

5, 3, 2 + 2, 1, 1  $\Rightarrow$  7, 4, 3

**Similarly, we examine another process P<sub>5</sub>.**

## Solution (Cont.)

**Step 5:** For Process P<sub>5</sub>:

$P_5 \text{ Need} \leq \text{Available}$

$4, 3, 1 \leq 7, 4, 3$  condition is **true**

New available resource = Available + Allocation

$7, 4, 3 + 0, 0, 2 \Rightarrow 7, 4, 5$

Now, we again examine each type of resource request for processes P<sub>1</sub> and P<sub>3</sub>.

**Step 6:** For Process P<sub>1</sub>:

$P_1 \text{ Need} \leq \text{Available}$

$7, 4, 3 \leq 7, 4, 5$  condition is **true**

New Available Resource = Available + Allocation

$7, 4, 5 + 0, 1, 0 \Rightarrow 7, 5, 5$

**So, we examine another process P<sub>2</sub>.**

**Step 7:** For Process P<sub>3</sub>:

$P_3 \text{ Need} \leq \text{Available}$

$6, 0, 0 \leq 7, 5, 5$  condition is true

New Available Resource = Available + Allocation

$7, 5, 5 + 3, 0, 2 \Rightarrow 10, 5, 7$

Hence, we execute the banker's algorithm to find the safe state and the safe sequence like P<sub>2</sub>, P<sub>4</sub>, P<sub>5</sub>, P<sub>1</sub> and P<sub>3</sub>.

# Banker's Algorithm for Multiple Resource instance of Each type

**Example:**

Considering a system with five processes  $P_0$  through  $P_4$  and three resources of type A, B, C. 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:

Find Safe Sequence.

Process	Allocation	Max	Available
	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	



**What will be the content of the Need matrix?**

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

So, the content of Need Matrix is:

Process	Need		
	A	B	C
P <sub>0</sub>	7	4	3
P <sub>1</sub>	1	2	2
P <sub>2</sub>	6	0	0
P <sub>3</sub>	0	1	1
P <sub>4</sub>	4	3	1

Solution:

Safe sequence is: P<sub>1</sub>, P<sub>3</sub>, P<sub>4</sub>, P<sub>0</sub>, P<sub>2</sub>

## Drawbacks

- In practice this algorithm is essentially useless because processes rarely know in *advance what their maximum resource needs*.
- In addition, the *number of processes is not fixed*, but dynamically varying as new users log in and out.
- Furthermore, resources that were thought to be available can *suddenly vanish* (tape drives can break).
- Thus in practice, few, if any, existing systems use the banker's algorithm for avoiding deadlocks.

# Deadlock Detection & Recovery

When this technique is used, the system does not attempt to prevent deadlocks from occurring.

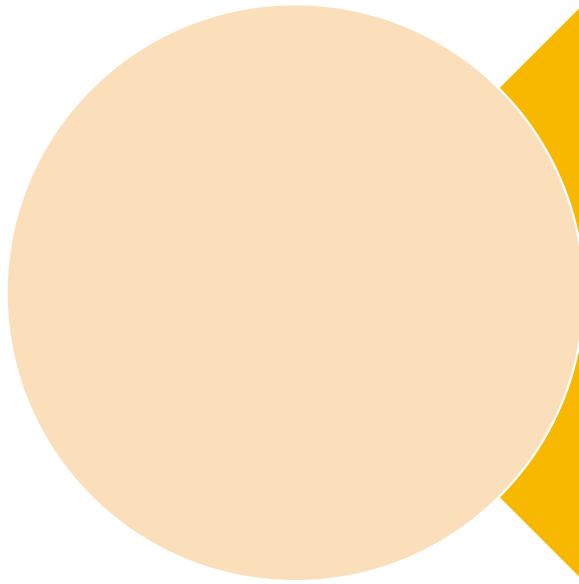
Instead, it lets them occur, tries to detect when this happens, and then takes some action to recover after the fact.

# Deadlock Detection & Recovery

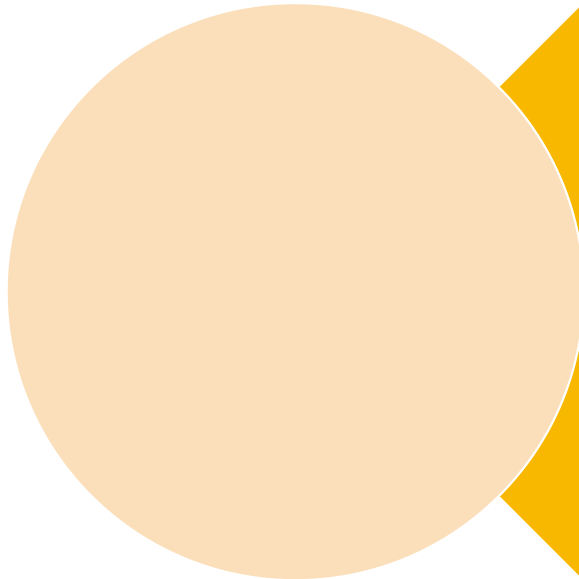
For deadlock detection, the system must provide

- 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: one Resources of each type



system with only one  
resources of each type  
exists. i.e. one scanner,  
one printer, one CD  
recorder, etc.



For this system we can  
draw wait for graph or  
resource allocation graph.

- If there is cycle in that graph  
then deadlock exist.
- If there is no cycle in that  
graph then no deadlock exist.

# Deadlock Detection: one Resources of each type

- There are seven process, A to G, and six resources R through W .
- Process A hold R and wants S.
- Process B holds nothing but wants T
- Process C holds nothing but wants S.
- Process D holds U and wants S and T
- Process E holds T and wants V
- Process F holds W and wants S.
- Process G holds V and wants U.
- Is this system deadlock ? And if so which processes are involved ?

## Answer of previous slide :

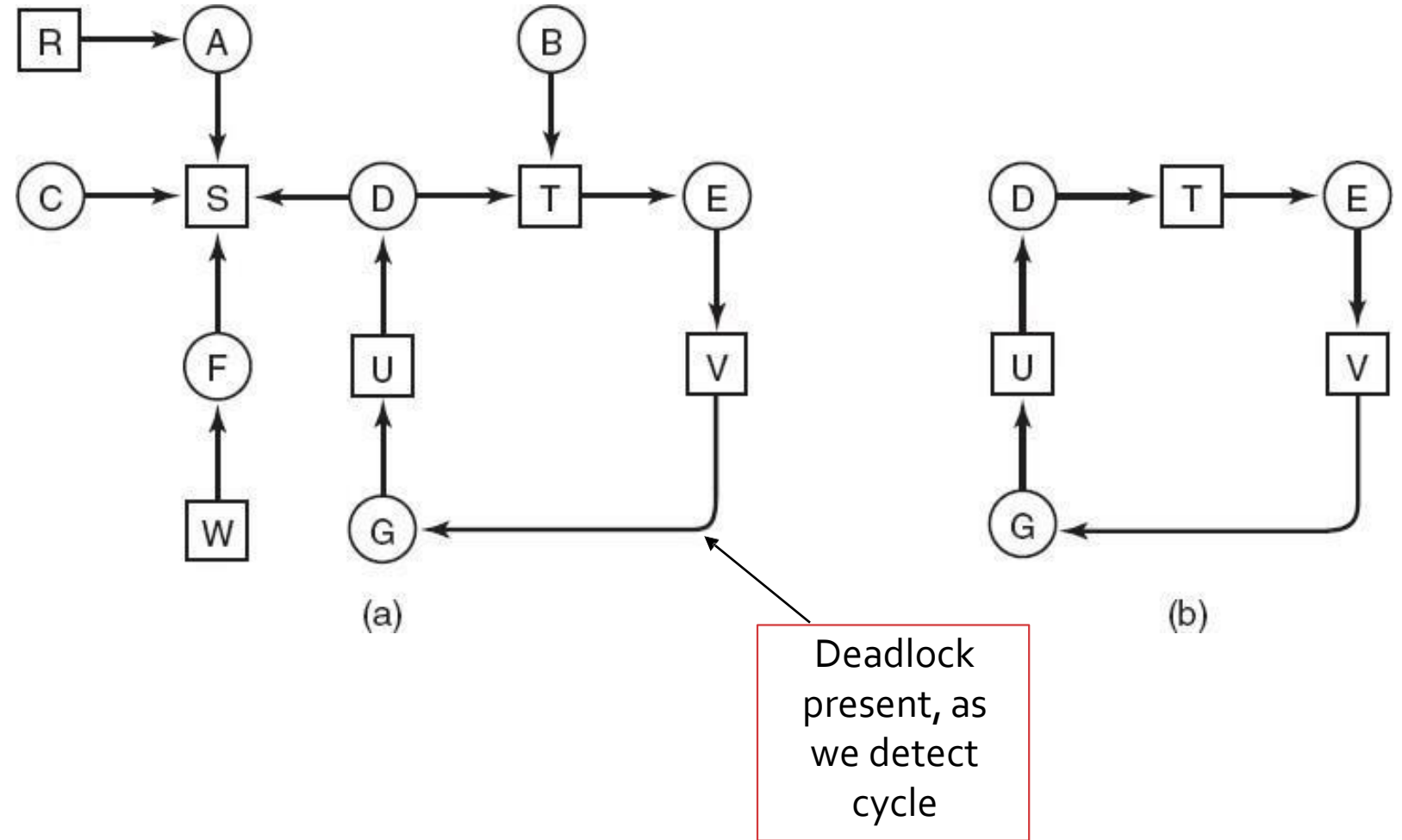


Figure : (a) A resource graph. (b) A cycle extracted from (a).

# Deadlock Detection: Multiple Resources of Each Type

For multiple resources we will use *Matrix algorithm*.

Existing resource vector

- It gives the total number of instances of each resource in existence. For example, if class 1 is tape drives, then  $E1 = 2$  means the system has two tape drives.

Available resource vector

- with  $A_i$  giving the number of instances of resource  $i$  that are currently available (i.e., unassigned).
- If both of our two tape drives are assigned,  $A1$  will be 0.



# Deadlock Detection: Multiple Resources of Each Type

E =

Tape Drive	Plotters	Scanners	CD Roms
4	2	3	1

total no of each resource

C =

Process	Tape Drive	Plotters	Scanners	CD Roms
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0

no of resources held by each process

A =

Tape Drive	Plotters	Scanners	CD Roms
2	1	0	0

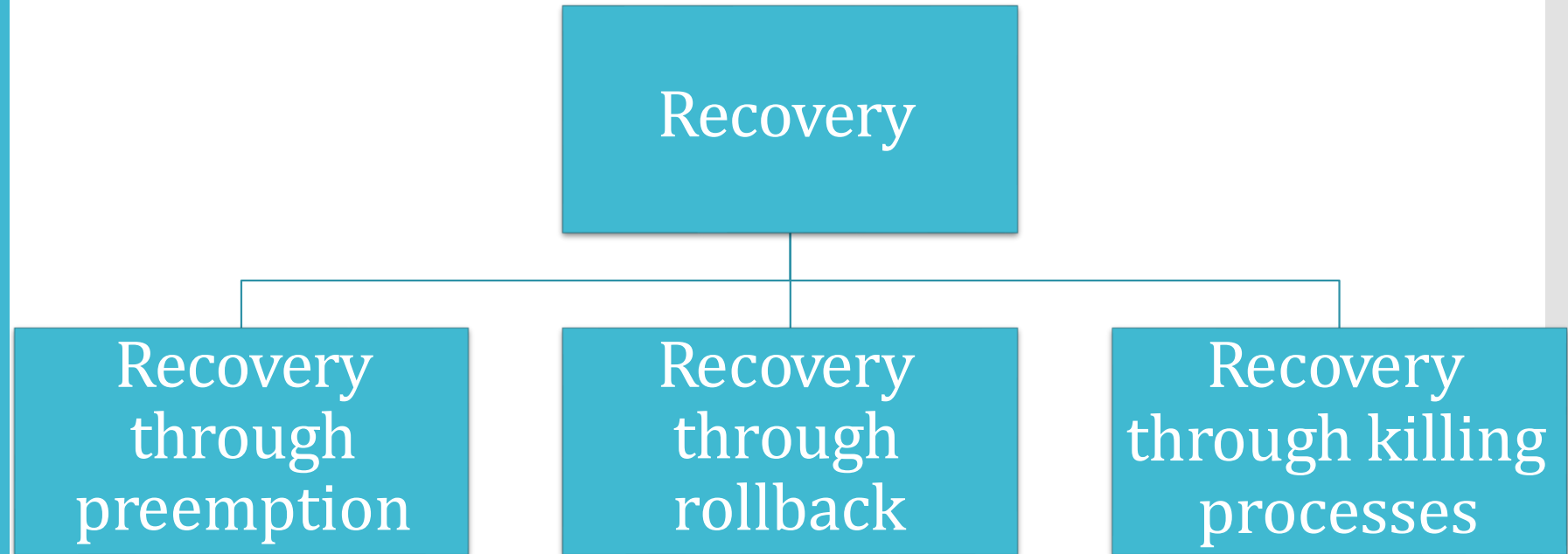
no of resources that are available (free)

R =

Process	Tape Drive	Plotters	Scanners	CD Roms
P1	2	0	0	1
P2	1	0	1	1
P3	2	1	0	0

no of resources still needed by each process to proceed

# Recovery from Deadlock



# Recovery from Deadlock

## Recovery through Preemption

- In some cases it may be possible to temporarily take a resource away from its current owner and give it to another process.
- The ability to take a resource away from a process, have another process use it, and then give it back without the process noticing it is highly dependent on the nature of the resource.

# Recovery from Deadlock

## Recovery through Rollback

- If the system designers and machine operators know that deadlocks are likely, they can arrange to have processes **check pointed** periodically.
- Check pointing a process means that its state is written to a file so that it can be restarted later.
- The checkpoint contains not only the *memory image*, *but also the resource state*, that is, which resources are currently assigned to the process.
- When a deadlock is detected, it is easy to see which resources are needed. To do the recovery, a process that owns a needed resource is rolled back to a point in time before it acquired some other resource by starting one of its earlier checkpoints.

# Recovery from Deadlock

## Recovery through Killing Processes

- simplest way to break a deadlock is to kill one or more processes. One possibility is to kill a process in the cycle.
- With a little luck, the other processes will be able to continue.
- If this does not help, it can be repeated until the cycle is broken.
- Alternatively, a process not in the cycle can be chosen as the victim in order to release its resources.
- For example, one process might hold a printer and want a plotter, with another process holding a plotter and wanting a printer. These two are deadlocked. A third process may hold another identical printer and another identical plotter and be happily running. Killing the third process will release these resources and break the deadlock involving the first two.