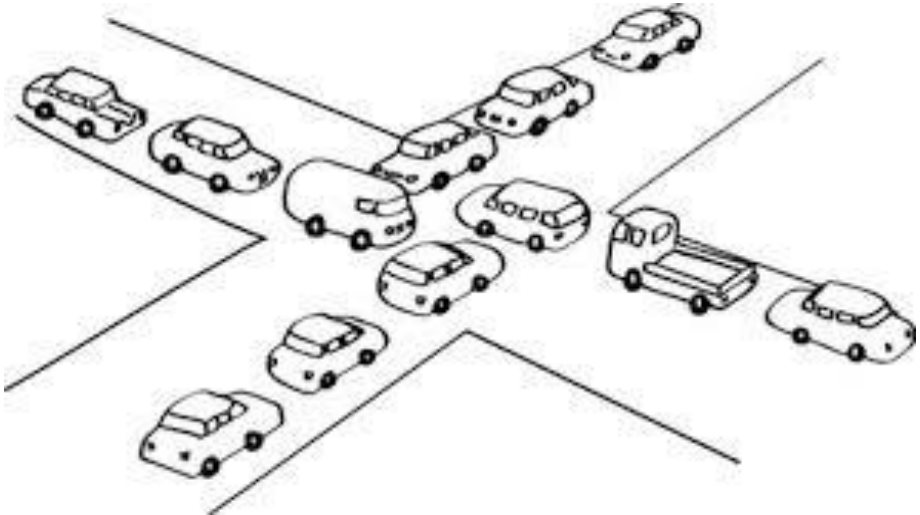
DEADLOCKS

Chapter 3

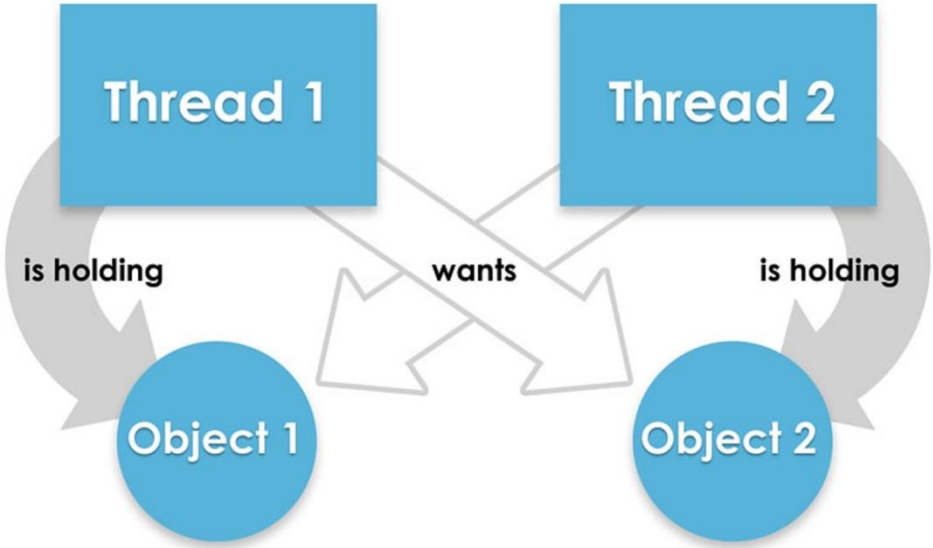


TOPICS

- 6.1 Deadlocks: Introduction
  - Condition for Deadlock
  - Deadlock Modeling
- 6.2 Deadlocks: Detection Recovery and Prevention
  - Deadlock detection with one Resource of each type
  - Deadlock detection with Multiple Resource of Each type
  - Deadlock Prevention

DEADLOCKS

- Hardware Resources: printers, tape drives, slots etc.
- a process needs exclusive access to resources.
- A situation where Process A holds scanner and request for cd recorder and Process B holds cd recorder and request for scanner is called **Deadlock.**
- **Deadlock** can also occur in LAN where resources are shared to multiple machines.



DEADLOCKS IN SW RESOURCE

- In a database system.
- a program may have to lock several records it is using, to avoid race conditions. If process *A* locks record *R1* and process *B* locks record *R2*, and then each process tries to lock the other one's record, we also have a deadlock.
- Thus deadlocks can occur on hardware resources or on software resources.

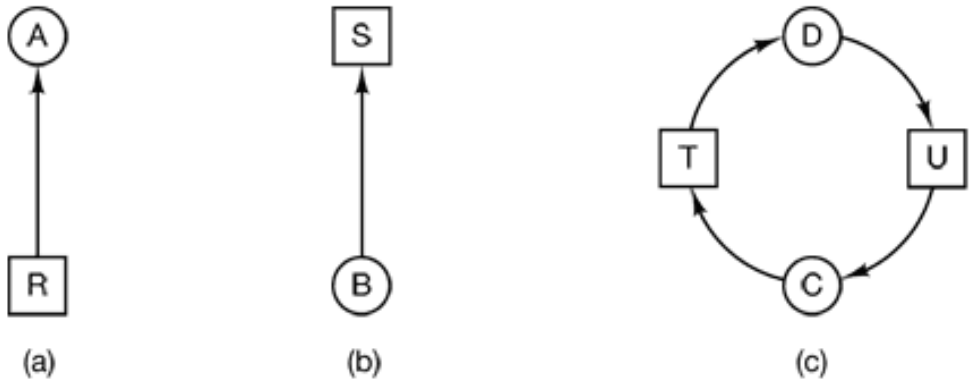
- **Formal definition:** *A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause*.

CHARACTERSTICS FOR DEADLOCK

- Coffman et al. (1971) showed that four conditions must hold for there to be a deadlock:
- **Mutual exclusion condition**. Each resource is either currently assigned to exactly one process or is available.
- **Hold and wait condition**. Processes currently holding resources granted earlier can request new resources.
- **No preemption condition**. Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.
- **Circular wait condition**. There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain.
- All four of these conditions must be present for a deadlock to occur. If one of them is absent, no deadlock is possible. By negating any one of the above conditions we can prevent deadlock.

DEADLOCK MODELING

- four conditions can be modeled using directed graphs
- An arc from a resource node (square) to a process node (circle) means that the resource has previously been requested by, granted to, and is currently held by that process.
- An arc from a process to a resource means that the process is currently blocked waiting for that resource



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

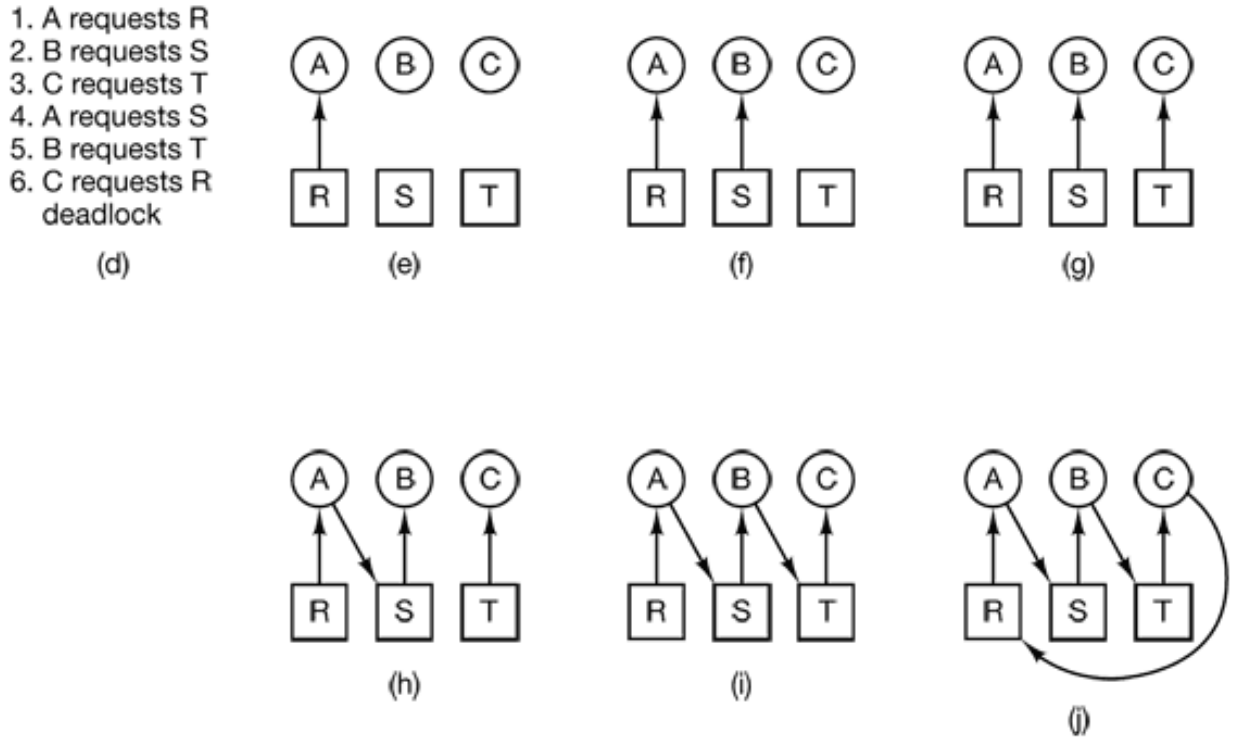
DEADLOCK MODELING

- Sequential execution of process shows no deadlocks.
- If no I/O in processes then SJF is better than Round Robin.
- But then there is no Parallelism.

A	B	C
Request R	Request S	Request T
Request S	Request T	Request R
Release R	Release S	Release T
Release S	Release T	Release R
(a)	(b)	(c)

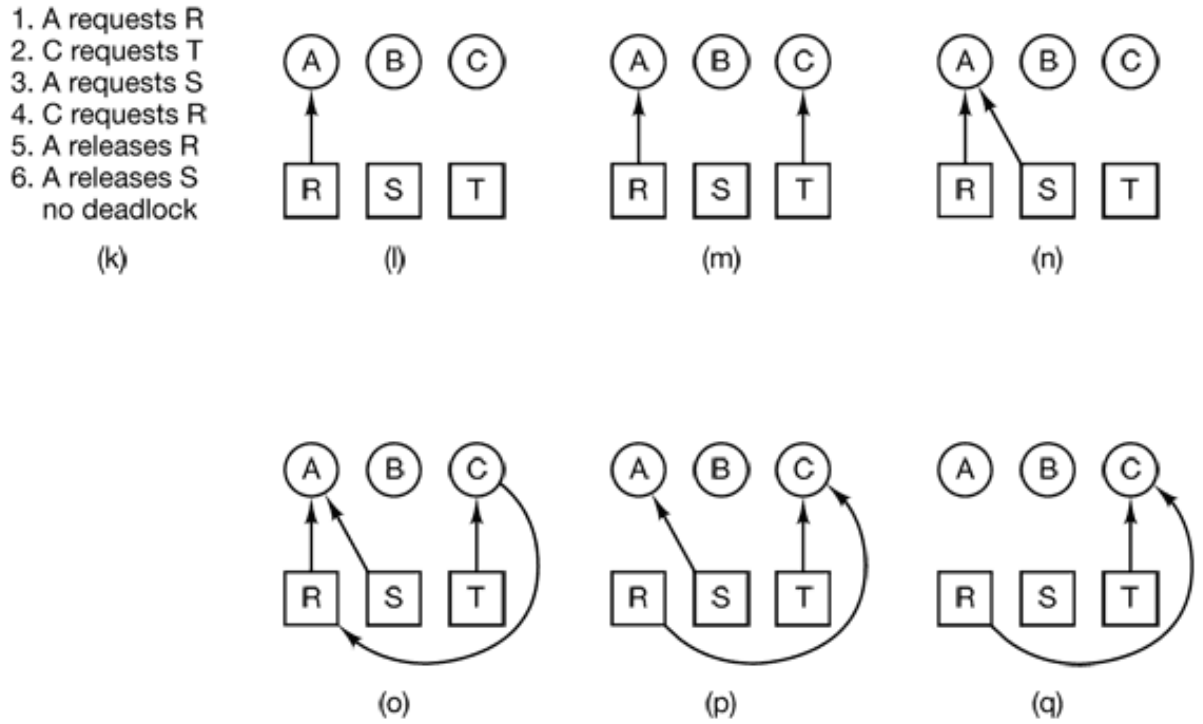
DEADLOCK MODELING

- Using Round Robin for parallelism may lead to deadlock



DEADLOCK MODELING

- if the operating system knew about the impending deadlock, it could suspend *B* instead of granting it *S*



DEADLOCK HANDLING

- In general, **four strategies** are used for dealing with deadlocks.
  - **Strategy 1.** Just ignore the problem altogether. Maybe if you ignore it, it will ignore you.
  - **Strategy 2.** Detection and recovery. Let deadlocks occur, detect them, and take action.
  - **Strategy 3.** Deadlock avoidance by careful resource allocation.
  - **Strategy 4.** Prevention, by structurally negating one of the four conditions necessary to cause a deadlock.

STRATEGY 1. THE OSTRICH ALGORITHM

- stick your head in the sand and pretend there is no problem at all.
- Mathematicians find it totally unacceptable and say that deadlocks must be prevented at all costs.
- Engineers ask how often the problem is expected, how often the system crashes for other reasons, and how serious a deadlock is.
- Most operating systems potentially suffer from deadlocks that are not even detected.

STRATEGY 2 DEADLOCK DETECTION AND RECOVERY

- 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.

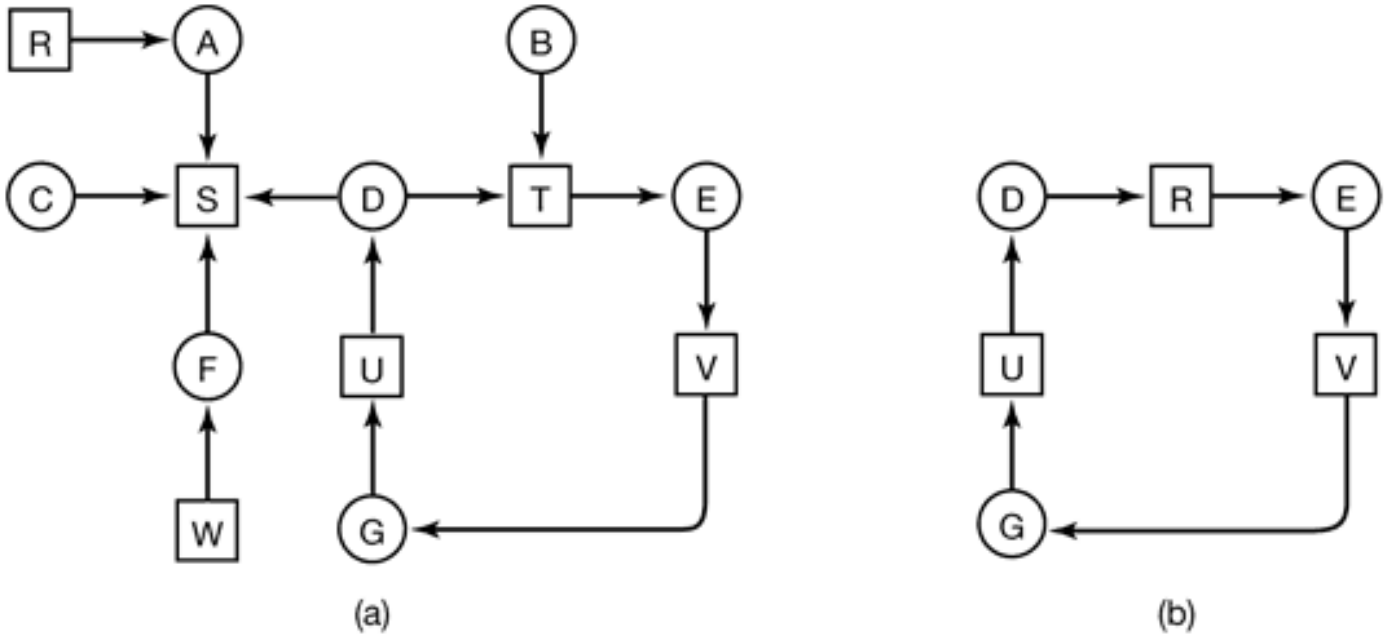
- Different approaches
  - **Deadlock Detection with One Resource of Each Type**
  - **Deadlock Detection with Multiple Resource of Each Type**

DEADLOCK DETECTION WITH ONE RESOURCE OF EACH TYPE

- only one resource of each class exists.
  - Such a system might have one scanner, one CD recorder, one plotter, and one tape drive.
- Is this System Deadlocked? If so which process are involved?
  - Process *A* holds *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*.

DEADLOCK DETECTION WITH ONE RESOURCE OF EACH TYPE

- We can find out easily where is deadlock cycle through visual inspection



DEADLOCK DETECTION WITH ONE RESOURCE OF EACH TYPE

- for use in actual systems we need a formal algorithm for detecting deadlocks
- 1. For each node, *N* in the graph, perform the following 5 steps with *N* as the starting node.
- 2. Initialize *L* to the empty list, and designate all the arcs as unmarked.
- 3. Add the current node to the end of *L* and check to see if the node now appears in *L* two times. If it does, the graph contains a cycle (listed in *L*) and the algorithm terminates.
- 4. From the given node, see if there are any unmarked outgoing arcs. If so, go to step 5; if not, go to step 6.
- 5. Pick an unmarked outgoing arc at random and mark it. Then follow it to the new current node and go to step 3.
- 6. We have now reached a dead end. Remove it and go back to the previous node, that is, the one that was current just before this one, make that one the current node, and go to step 3. If this node is the initial node, the graph does not contain any cycles and the algorithm terminates.

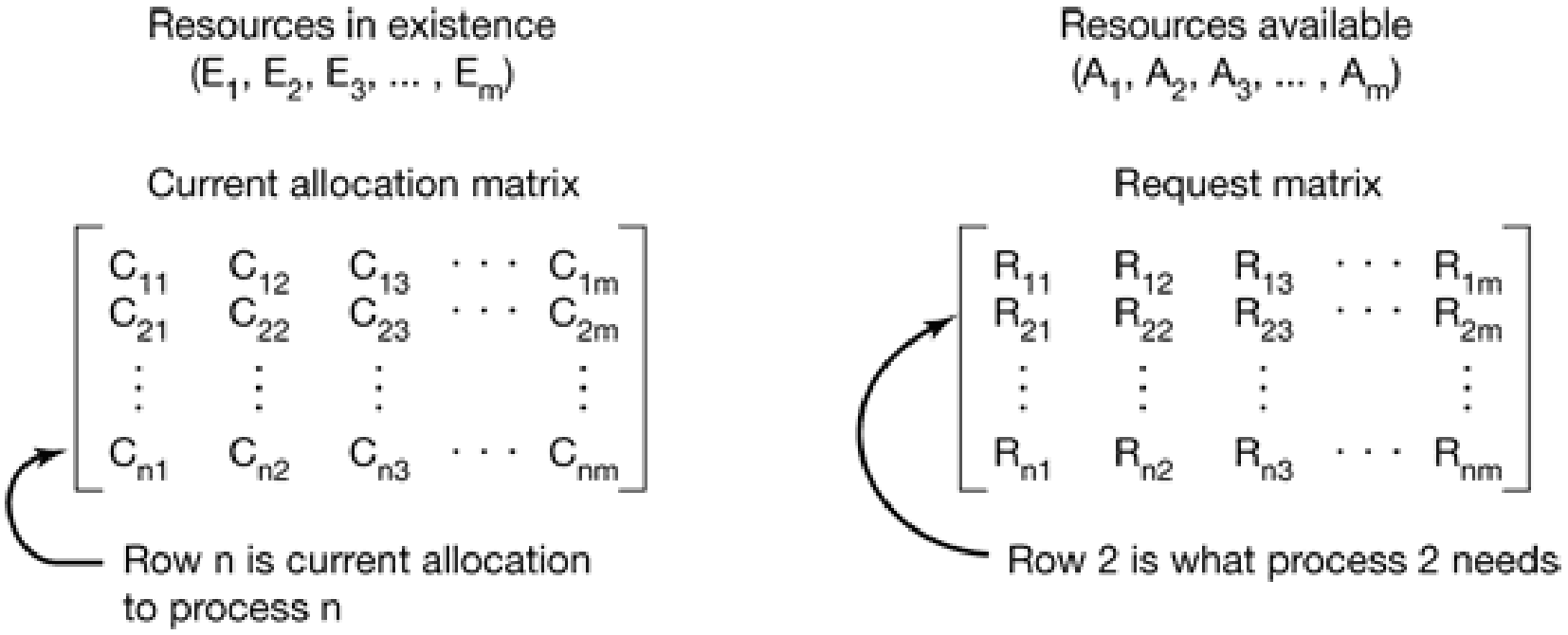
EG. : LET US USE IT ON THE GRAPH OF ABOVE FIG

- The order of processing the nodes is arbitrary, so let us just inspect them from left to right, top to bottom, first running the algorithm starting at R then successively, A, B, C, S, D, T, E, F, and so forth. If we hit a cycle, the algorithm stops.
- We start at R and initialize L to the empty list. Then we add R to the list and move to the only possibility, A, and add it to L, giving $L = [R, A]$. From A we go to S, giving $L = [R, A, S]$. S has no outgoing arcs, so it is a dead end, forcing us to backtrack to A. Since A has no unmarked outgoing arcs, we backtrack to R, completing our inspection of R.
- Now we restart the algorithm starting at A, resetting L to the empty list. This search, too, quickly stops, so we start again at B. From B we continue to follow outgoing arcs until we get to D, at which time $L = [B, T, E, V, G, U, D]$. Now we must make a (random) choice. If we pick S we come to a dead end and backtrack to D. The second time we pick T and update L to be $[B, T, E, V, G, U, D, T]$, at which point we discover the cycle and stop the algorithm.

DEADLOCK DETECTION WITH MULTIPLE RESOURCE OF EACH TYPE

- n processes. P_1 through P_n
- E is the **existing resource vector.**
- A be the **available resource vector.**
- two arrays, C, the **current allocation matrix**, and R, the **request matrix**.
- C_{ij} is the number of instances of resource j that are held by process i.
- R_{ij} is the number of instances of resource j that P_i wants.

THE FOUR DATA STRUCTURES NEEDED BY THE DEADLOCK DETECTION ALGORITHM.



ALGORITHM

- The deadlock detection algorithm can now be given, as follows.
  - Look for an unmarked process, P_i, for which the i-th row of R is less than or equal to A.
  - If such a process is found, add the i-th row of C to A, mark the process, and go back to step 1.
  - If no such process exists, the algorithm terminates.
- When the algorithm finishes, all the unmarked processes, if any, are deadlocked.

Tape drives Plotters Scanners CD Roms

E = (4 2 3 1)

Tape drives Plotters Scanners CD Roms

A = (2 1 0 0)

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

- third one can be satisfied, so process 3 runs and eventually returns all its resources, giving
- A = (2 2 2 0)
- At this point process 2 can run and return its resources, giving
- A = (4 2 2 1)
- Now the remaining process can run. There is no deadlock in the system.
- What if 3rd process needs a CD-ROM drive as well as the two tape drives and the plotter?

RECOVERY FROM DEADLOCK

- **Recovery through Preemption**
  - temporarily take a resource away from its current owner and give it to another process
- **Recovery through Rollback**
  - 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 through Killing Processes**
  - kill one or more processes.

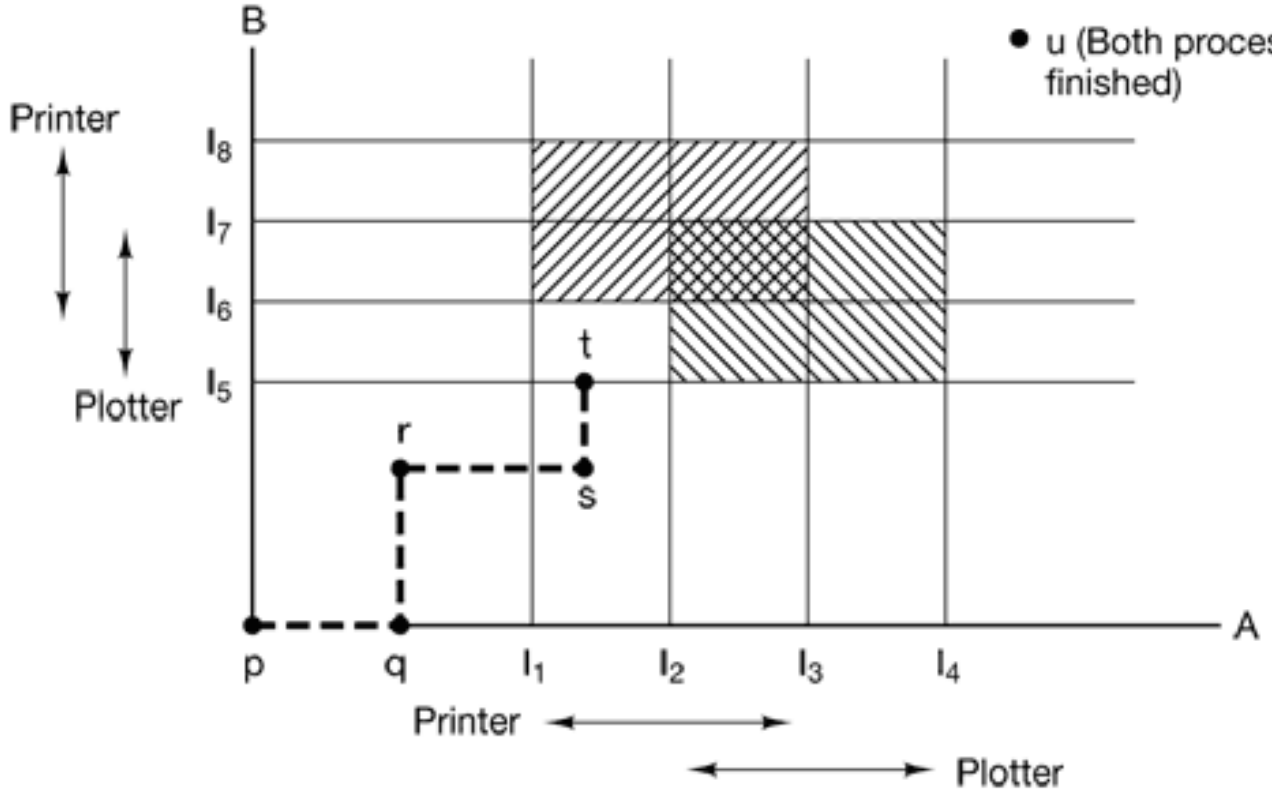
DEADLOCK AVOIDANCE

deadlock avoidance by careful resource allocation

- Decide whether granting a resource is safe or not, and only make the allocation when it is safe.
- Need extra information in advanced, maximum number of resources of each type that a process may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.

- **1 Resource Trajectories**
- **2 Safe and Unsafe States**
- **3 The Banker's Algorithm for a Single Resource**
- **4 The Banker's Algorithm for Multiple Resources**

1 RESOURCE TRAJECTORIES



- To avoid the deadlock, *B* should be suspended until *A* has requested and released the plotter.

SAFE AND UNSAFE STATES

- Demonstration that the state in (a) is safe.
- A state is said to be safe if it is not deadlocked and there is a some scheduling order in which every process can run to completion.

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

(b)

	Has	Max
A	3	9
B	0	—
C	2	7

Free: 5

(c)

	Has	Max
A	3	9
B	0	—
C	7	7

Free: 0

(d)

	Has	Max
A	3	9
B	0	—
C	0	—

Free: 7

(e)

SAFE AND UNSAFE STATES

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3
(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2
(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0
(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4
(d)

- Demonstration that the state in (b) is not safe.
- the difference between a safe state and an unsafe state is that
  - from a safe state the system can *guarantee* that all processes will finish;
  - from an unsafe state, no such guarantee can be given.

THE BANKER'S ALGORITHM

- Models on the way of banking system to ensure that the bank never allocates its available cash such that it can no longer satisfy the needs of all its customers.
- When a process request the set of resources, the system determine whether the allocation of these resources will left the system in safe state, if it will the resources are allocated, otherwise process must wait until some other process release enough resources.

THE BANKER'S ALGORITHM

- Which one is safe state?

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

(a)

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

(b)

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)

PROBLEM WITH BANKER'S ALGORITHM

- Algorithms requires fixed number of resources, some processes dynamically changes the number of resources.
- Algorithms requires the number of resources in advanced, it is very difficult to predict the resources in advanced.
- Algorithms predict all process returns within finite time, but the system does not guarantee it.