

# Deadlocks

## Chapter 6

# Deadlock

- » Computers are full of resources that can only be used by one process at a time.
  - Printers
  - Open-files table
- » OS has the ability to temporarily grant a process exclusive access to certain resources.
- » Some processes needs exclusive access to not one resource, but several.

# Deadlocks

- » Suppose, for example, two processes each want to record a scanned document on a Blu-ray disc.
  - Process A requests permission to use the scanner and is granted it.
  - Process B is programmed differently and requests the Blu-ray re-corder first and is also granted it.

# Deadlocks

- Now A asks for the Blu-ray recorder, but the request is suspended until B releases it.
- Unfortunately, instead of releasing the Blu-ray recorder, B asks for the scanner.
- At this point both processes are blocked and will remain so forever. This situation is called a deadlock.

# Deadlocks

- » Deadlocks can also occur across machines.
  - Network resources
- » 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.
- » Takeaway: Deadlocks can occur with hardware and software resources.

# Deadlocks

- » A major class of deadlocks involves resources to which some process has been granted exclusive access.
  - **Resource**: object granted, or anything that must be acquired, used, and released over the course of time
  - May be several identical instances may be available, such as three Blu-ray drives.
    - When several copies of a resource are available, any one of them can be used to satisfy any request for the resource.

# Resources

» Two types:

- Preemptable: one that can be taken away from the process owning it with no ill effects.
  - Memory
- Nonpreemptable: one that cannot be taken away from its current owner without potentially causing failure.
  - If a process has begun to burn a Blu-ray, suddenly taking the Blu-ray recorder away from it and giving it to another process will result in a garbled Blu-ray.

# Resources

- » Whether a resource is preemptible depends on the context.
  - PC memory is preemptible because pages can always be swapped out to disk to recover it.
  - A smartphone that does not support swapping or paging, cannot avoid deadlocks by just swapping out a memory hog.



# Deadlocks

- » In general, deadlocks involve nonpreemptable resources.
  - Potential deadlocks that involve preemptable resources can usually be resolved by reallocating resources from one process to another.
  - Sequence of events required to use a resource
    1. Request the resource.
    2. Use the resource.
    3. Release the resource.

# Resources

- » In some operating systems, the process is automatically blocked when a resource request fails, and awakened when it becomes available.
- » In other systems, the request fails with an error code, and it is up to the calling process to wait a little while and try again.
  - Busy wait

# Resource Acquisition

- » For some kinds of resources, such as records in a database system, it is up to the user processes rather than the system to manage resource usage themselves.
  - Semaphore, or mutex

```
typedef int semaphore;  
semaphore resource_1;
```

```
void process_A(void) {  
    down(&resource_1);  
    use_resource_1( );  
    up(&resource_1);  
}
```

(a)

```
typedef int semaphore;  
semaphore resource_1;  
semaphore resource_2;
```

```
void process_A(void) {  
    down(&resource_1);  
    down(&resource_2);  
    use_both_resources( );  
    up(&resource_2);  
    up(&resource_1);  
}
```

(b)

Single process is easy

# Resource Acquisition

```
typedef int semaphore;  
semaphore resource_1;  
semaphore resource_2;  
  
void process_A(void) {  
    down(&resource_1);  
    down(&resource_2);  
    use_both_resources( );  
    up(&resource_2);  
    up(&resource_1);  
}  
  
void process_B(void) {  
    down(&resource_1);  
    down(&resource_2);  
    use_both_resources( );  
    up(&resource_2);  
    up(&resource_1);  
}
```

(a)

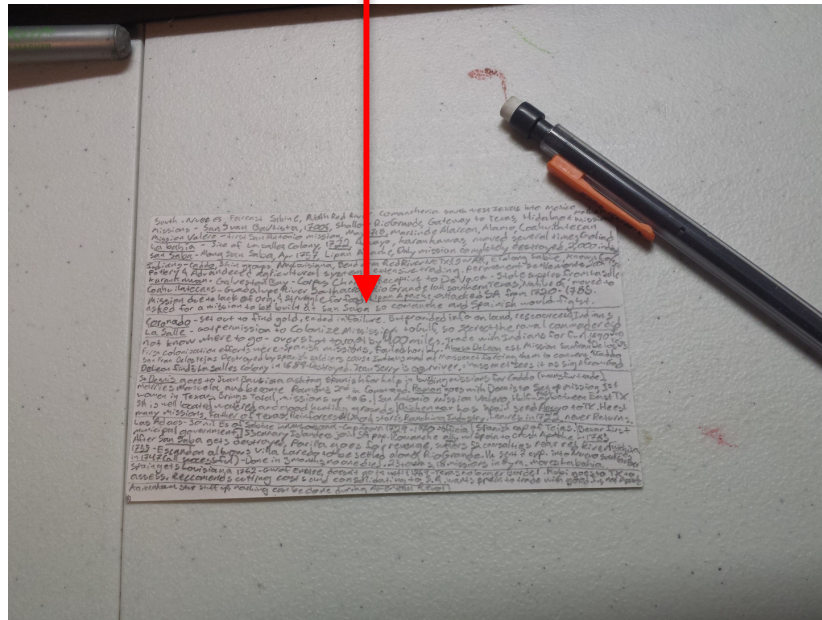
```
semaphore resource_1;  
semaphore resource_2;  
  
void process_A(void) {  
    down(&resource_1);  
    down(&resource_2);  
    use_both_resources( );  
    up(&resource_2);  
    up(&resource_1);  
}  
  
void process_B(void) {  
    down(&resource_2);  
    down(&resource_1);  
    use_both_resources( );  
    up(&resource_1);  
    up(&resource_2);  
}
```

(b)

- (a) Deadlock-free code.  
(b) Code with a potential deadlock.

# Deadlocks

» Deadlock: 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.



# Deadlocks

- » For this model, we assume that processes are single threaded and that no interrupts are possible to wake up a blocked process.
- » The no-interrupts condition is needed to prevent an otherwise deadlocked process from being awakened by an alarm, and then causing events that release other processes in the set.

# Deadlocks

- » The number of processes and the number and kind of resources possessed and requested are unimportant.
  - Resource deadlock

# Conditions for deadlock

1. Mutual exclusion condition. Each resource is either currently assigned to exactly one process or is available.
2. Hold-and-wait condition. Processes currently holding resources that were granted earlier can request new resources.
3. No-preemption condition. Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.
4. Circular wait condition. There must be a circular list of two or more processes, each of which is waiting for a resource held by the next member of the chain.





# Conditions for deadlock

- » All four of these conditions must be present for a resource deadlock to occur. If one of them is absent, no resource deadlock is possible.
  - Gotta collect ‘em all.

# Deadlock Modeling

## » Directed graph

- Two kinds of nodes: processes, shown as circles, and resources, shown as squares.
- A directed 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.

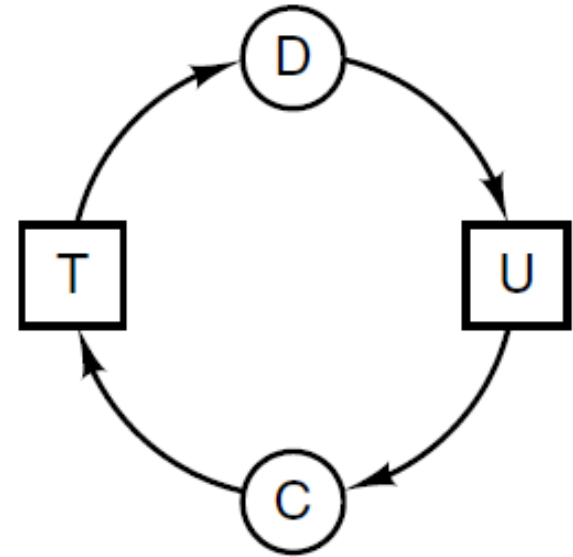
# Deadlock Modeling



(a)



(b)



(c)

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

# Deadlock Modeling

» Strategies are used for dealing with deadlocks:

1. Ignore the problem, maybe it will go away.
2. Detection and recovery. Let deadlocks occur, detect them, and take action.
3. Dynamic avoidance by careful resource allocation.
4. Prevention, by structurally negating one of the four required conditions.



# Ostrich Algorithm

- » Simplest approach
- » Stick your head in the sand and pretend there is no problem.
  - UNIX approach
- » Different reactions based on field
  - Mathematicians find it unacceptable and say that deadlocks must be prevented



# Ostrich Algorithm

- » Engineers ask how often the problem is expected, how often the system crashes for other reasons, and how serious a deadlock is.
  - If deadlocks occur on the average once every five years, but system crashes due to hardware failures and operating system bugs occur once a week, who cares. Let it deadlock.
    - Closest alligator to the boat.

# Deadlock Detection and Recovery

» Don't attempt to prevent deadlocks.

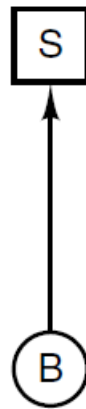
1. Let them occur
2. Detect them
3. Take action to recover

# Deadlock Detection and Recovery

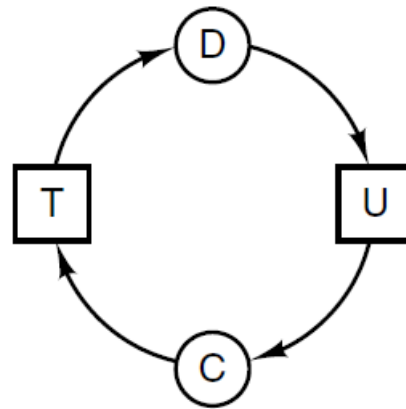
- » Simplest case: there is only one resource of each type.
- » Construct a resource graph. If this graph contains one or more cycles, a deadlock exists.



(a)



(b)



(c)

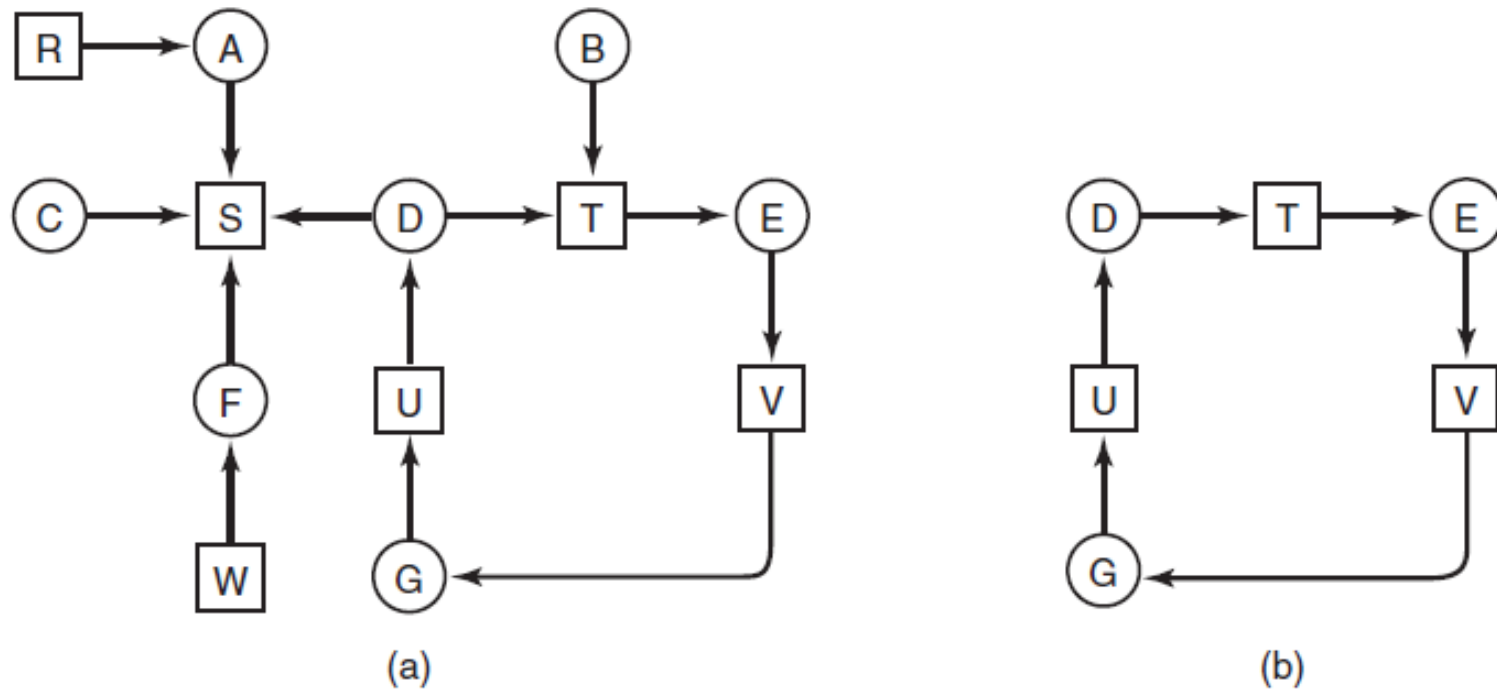


# Deadlock Detection and Recovery

- » Consider a system with seven processes, A through G, and six resources, R through W. The state of which resources are currently owned and which ones are currently being requested is as follows:
- » 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.

Is our system deadlocked?

# Deadlock Detection and Recovery



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

# Deadlock Detection and Recovery

» Visually it's easy to see. Algorithm is tougher.

1. For each node,  $N$  in the graph, perform following five steps with  $N$  as starting node.
2. Initialize  $L$  to empty list, and designate all arcs as unmarked.
3. Add current node to end of  $L$ , check to see if node now appears in  $L$  two times. If so, graph contains a cycle (listed in  $L$ ) and algorithm terminates

# Deadlock Detection and Recovery

4. From given node, see if there are any unmarked outgoing arcs. If so, go to step 5; if not, go to step 6.
5. Pick unmarked outgoing arc at random, mark it. Then follow to new current node and go to step 3.
6. If this is initial node, graph does not contain cycles, algorithm terminates. Otherwise, dead end. Remove it and go back to the previous node.

# Deadlock Detection and Recovery

What this algorithm does is take each node, in turn, as the root of what it hopes will be a tree, and do a depth-first search on it.

If it ever comes back to a node it has already encountered, then it has found a cycle.

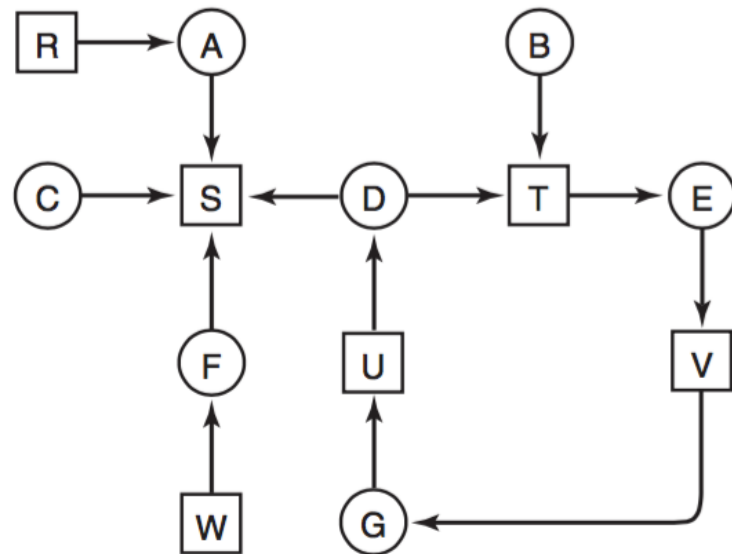
If it exhausts all the arcs from any given node, it backtracks to the previous node.

# Deadlock Detection and Recovery

If it backtracks to the root and cannot go further, the subgraph reachable from the current node does not contain any cycles.

If this property holds for all nodes, the entire graph is cycle free, so the system is not deadlocked.

# Deadlock Detection and Recovery

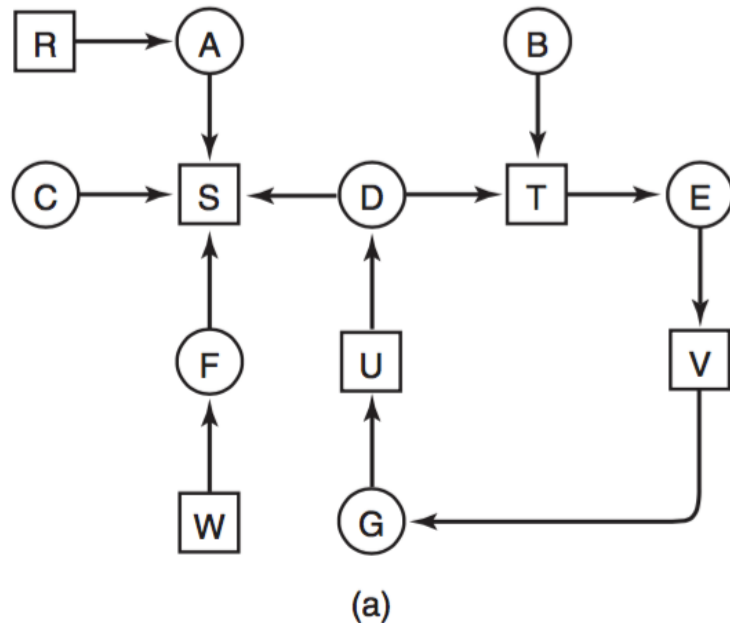


(a)

- » Starting at R, initialize L to an empty list.
- » Add R
- » Move to A and add it to L
- » Move to S and add it to L
- » Backtrack to A
- » Backtrack to R

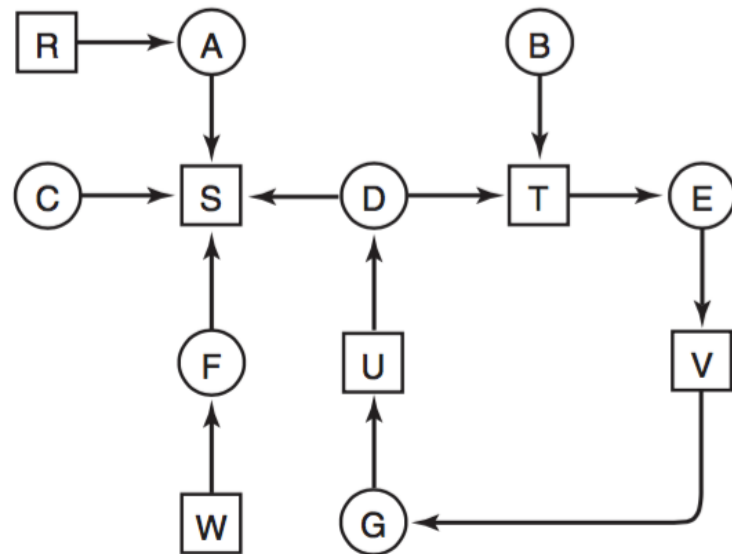
# Deadlock Detection and Recovery

- » Restart the algorithm at A, repeat
- » Restart at B
- » From B we continue to follow outgoing arcs until we get to D, at which time  
 $L = [B, T, E, V, G, U, D]$





# Deadlock Detection and Recovery



(a)

- » 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]
- » Cycle discovered.
- » Suboptimal algorithm but effective

# Deadlock Detection: Multiple Resources

- » When multiple copies of some of the resources exist, a different approach is needed to detect deadlocks
  - Use a matrix-based algorithm



# Deadlock Detection: Multiple Resources

- »  $n$  processes,  $P_1$  through  $P_n$ .
- » Let the number of resource classes be  $m$ , with  $E_1$  resources of class 1,  $E_2$  resources of class 2, and generally,  $E_i$  resources of class  $i$  ( $1 \leq i \leq m$ ).
- »  $E$  is the **existing resource vector**. It gives the total number of instances of each resource in existence.
- »  $A$  is the available resource vector, with  $A_i$  giving the number of instances of resource  $i$  that are currently available

# Deadlock Detection: Multiple Resources

» Two arrays

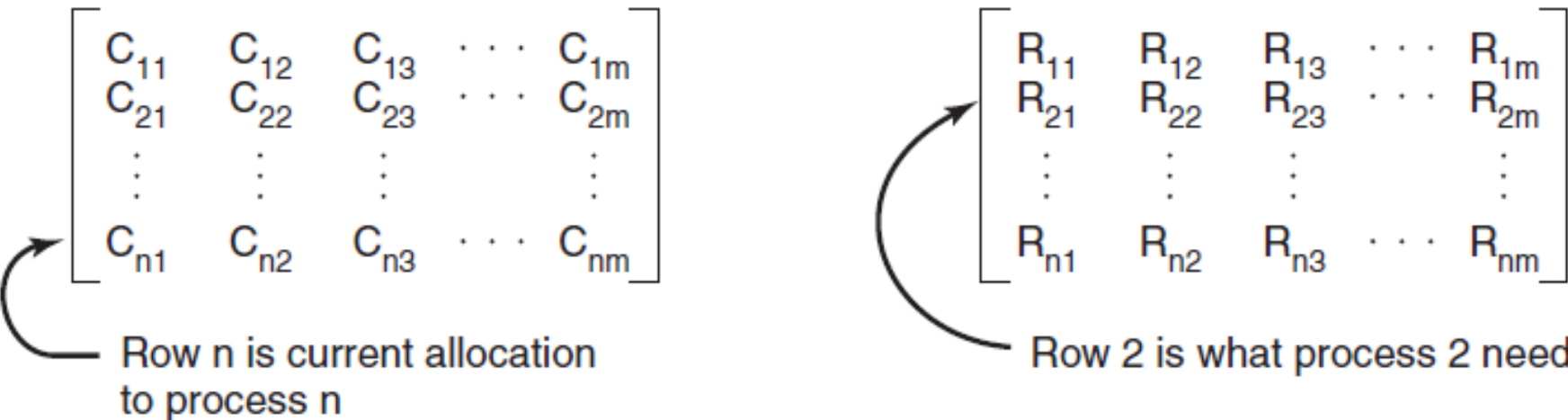
- C: the current allocation matrix
  - The  $i$ th row of C tells how many instances of each resource class  $P_i$  currently holds.
  - $C_{ij}$  is the number of instances of resource  $j$  that are held by process  $i$
- R: the request matrix.
  - $R_{ij}$  is the number of instances of resource  $j$  that  $P_i$  wants.

# Deadlock Detection: Multiple Resources

Resources in existence  
( $E_1, E_2, E_3, \dots, E_m$ )

Resources available  
( $A_1, A_2, A_3, \dots, A_m$ )

Current allocation matrix



$C_{11}$	$C_{12}$	$C_{13}$	$\dots$	$C_{1m}$
$C_{21}$	$C_{22}$	$C_{23}$	$\dots$	$C_{2m}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
$C_{n1}$	$C_{n2}$	$C_{n3}$	$\dots$	$C_{nm}$

Request matrix

$R_{11}$	$R_{12}$	$R_{13}$	$\dots$	$R_{1m}$
$R_{21}$	$R_{22}$	$R_{23}$	$\dots$	$R_{2m}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
$R_{n1}$	$R_{n2}$	$R_{n3}$	$\dots$	$R_{nm}$

Row n is current allocation  
to process n

Row 2 is what process 2 needs

The four data structures needed  
by the deadlock detection algorithm.

# Deadlock Detection: Multiple Resources

- » Let us define the relation  $A \leq B$  on two vectors  $A$  and  $B$  to mean that each element of  $A$  is less than or equal to the corresponding element of  $B$ .
  - Mathematically,  $A \leq B$  holds if and only if  $A_i \leq B_i$  for  $1 \leq i \leq m$ .

# Deadlock Detection: Multiple Resources

- » Each process is initially said to be unmarked. As the algorithm progresses, processes will be marked, indicating that they are able to complete and are thus not deadlocked.
- » When the algorithm terminates, any unmarked processes are known to be deadlocked.
  - Assumes a worst-case scenario: all processes keep all acquired resources until they exit.

# Deadlock Detection: Multiple Resources

» The deadlock detection algorithm can now be given as follows.

1. Look for an unmarked process,  $P_i$ , for which the  $i$ th row of  $R$  is less than or equal to  $A$ .
2. If such a process is found, add the  $i$ th row of  $C$  to  $A$ , mark the process, and go back to step 1.



# Deadlock Detection: Multiple Resources

	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

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

» Process 1 has one scanner. Process 2 has two tape drives and a CD drive. Process 3 has a plotter and two scanners.

# Deadlock Detection: Multiple Resources

	Tape drives	Plotters	Scanners	CD Roms
$E =$	4	2	3	1
$A =$	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

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

- » Look for a process whose resource request can be satisfied.
  - Process 1 can't be satisfied due to no CD drives available

# Deadlock Detection: Multiple Resources

$$E = \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix}$$

Tape drives  
Plotters  
Scanners  
CD Roms

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}$$

Tape drives  
Plotters  
Scanners  
CD Roms

Current allocation matrix

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

Request matrix

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

» The second cannot be satisfied either, because there is no scanner free.

# Deadlock Detection: Multiple Resources

$$E = \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix}$$

Tape drives  
Plotters  
Scanners  
CD Roms

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}$$

Tape drives  
Plotters  
Scanners  
CD Roms

Current allocation matrix

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

Request matrix

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

» The third one can be satisfied, so process 3 runs and eventually returns all its resources, giving

$$A = (2 \ 2 \ 2 \ 0)$$

# Deadlock Detection: Multiple Resources

$$E = \begin{matrix} & \begin{matrix} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{matrix} \\ \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix} \end{matrix}$$

$$A = \begin{matrix} & \begin{matrix} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{matrix} \\ \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix} \end{matrix}$$

Current allocation matrix

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

Request matrix

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

» At this point process 2 can run and return its resources, giving

$$A = (4 \ 2 \ 2 \ 1)$$

The remaining process can run. There is no deadlock in the system.

# Deadlock Detection: Multiple Resources

- » Suppose that process 3 needs a Blu-ray drive as well as the two tape drives and the plotter.
- » So when do we do the check?
  - Check every time a resource request is made
    - Detect them as early as possible,
    - CPU intensive
  - Check every n minutes
  - Check when CPU utilization has dropped below some threshold.

# Recovery from Deadlock

## » Recovery through Preemption

- 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.
- Difficult. Sometimes impossible.

# Recovery from Deadlock

## » Recovery through Rollback

- Have processes checkpointed periodically.
  - Process state is written to a file so that it can be restarted later.
- When a deadlock is detected. The process that owns a needed resource is rolled back to a point in time before it acquired that resource by starting at one of its earlier checkpoints.



# Recovery from Deadlock

- » All the work done since the checkpoint is lost
- » If the restarted process tries to acquire the resource again, it will have to wait until it becomes available.

# Recovery from Deadlock

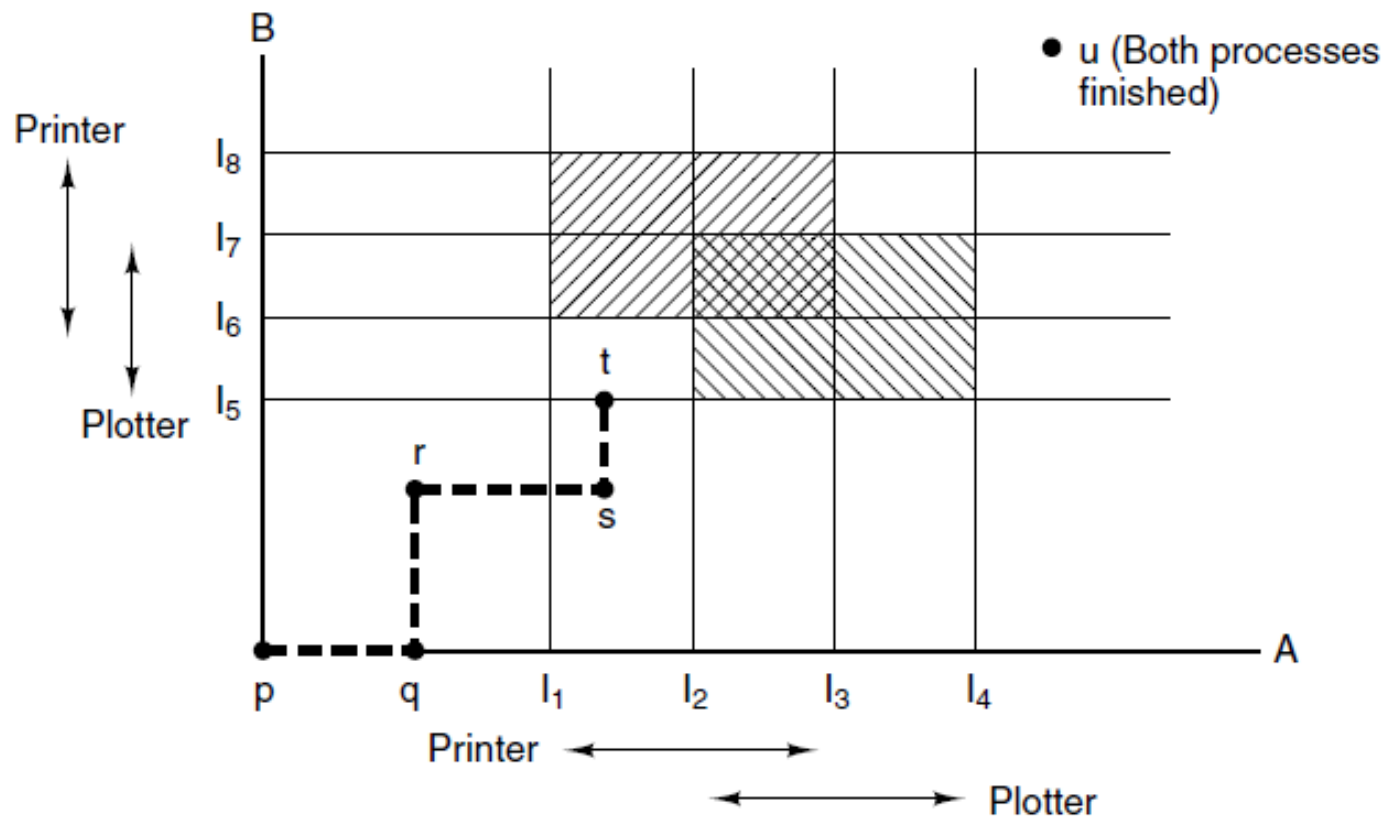
## » Recovery through Killing Processes

- Kill a process holding a resource
  - Repeat until you clear the deadlock
- Can also kill a process not in the deadlocked cycle if it has needed resources
- Best to kill a process that can be easily restarted and re-run.
  - Don't kill the process writing info to your database leaving the DB in an indeterminate state.

# Deadlock Avoidance

## » Resource Trajectories

- Main algorithms for deadlock avoidance are based on the concept of safe states.



# Deadlock Avoidance

## » Safe and Unsafe States

- At any instant of time, there is a current state consisting of E, A, C, and R.
- A state is said to be safe if there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of r immediately.



# Deadlock Avoidance

Has Max			Has Max			Has Max		
A	3	9	A	3	9	A	3	9
B	2	4	B	4	4	B	0	–
C	2	7	C	2	7	C	2	7
Free: 3			Free: 1			Free: 5		
(a)			(b)			(c)		

Has Max			Has Max		
A	3	9	A	3	9
B	0	–	B	0	–
C	7	7	C	0	–
Free: 0			Free: 7		
(d)			(e)		

- » A total of 10 instances of the resource exist,
- » A has three instances of the resource but may need as many as nine eventually.
- » B currently has two and may need four altogether, later.
- » C also has two but may need an additional five.

# Deadlock Avoidance

Has Max			Has Max			Has Max		
A	3	9	A	3	9	A	3	9
B	2	4	B	4	4	B	0	–
C	2	7	C	2	7	C	2	7
Free: 3			Free: 1			Free: 5		
(a)			(b)			(c)		

Has Max			Has Max		
A	3	9	A	3	9
B	0	–	B	0	–
C	7	7	C	0	–
Free: 0			Free: 7		
(d)			(e)		

- » The state is safe because there exists a sequence of allocations that allows all processes to complete.
  - The scheduler can simply run B exclusively, until it asks for and gets two more instances of the resource.

# Deadlock Avoidance

	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)

- » This time A requests and gets another resource
- » The scheduler runs B until it asked for all its resources
- » We only have four instances of the resource free, and each of the active processes need 5

# Banker's Algorithm for a Single Resource

- » Dijkstra (1965)
- » It is modeled on the way a small-town banker might deal with a group of customers to whom he has granted lines of credit.
  - Checks to see if granting the request leads to an unsafe state. If so, the request is denied.
    - » Not the way the 2008 housing market worked



# Banker's Algorithm for a Single Resource

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)

- » Customers A, B, C, and D, each granted a certain number of credit units
- » The banker knows that not all customers will need their maximum credit immediately, so he has reserved only 10 units rather than 22 to service them

# Banker's Algorithm for a Single Resource

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)

Unsafe



- » The customers go about their respective businesses, making loan requests from time to time (i.e., asking for resources) leading to (b)
- » This state is safe because with two units left, the banker can delay any requests except C's, thus letting C finish and release all four of his resources.

# Banker's Algorithm for Multiple Resource

	Process	Tape drives	Plotters	Printers	CD ROMs
A	3	0	1	1	
B	0	1	0	0	
C	1	1	1	0	
D	1	1	0	1	
E	0	0	0	0	

Resources assigned

	Process	Tape drives	Plotters	Printers	CD ROMs
A	1	1	0	0	
B	0	1	1	2	
C	3	1	0	0	
D	0	0	1	0	
E	2	1	1	0	

Resources still needed

E = (6342)  
P = (5322)  
A = (1020)

- » Can be generalized to handle multiple resources.
- » The matrix on the left shows how many of each resource are currently assigned to each of the five processes.
- » The matrix on the right shows how many resources each process still needs in order to complete

# Banker's Algorithm for Multiple Resource

	Process	Tape drives	Plotters	Printers	CD ROMs
A	3	0	1	1	
B	0	1	0	0	
C	1	1	1	0	
D	1	1	0	1	
E	0	0	0	0	

Resources assigned

	Process	Tape drives	Plotters	Printers	CD ROMs
A	1	1	0	0	
B	0	1	1	2	
C	3	1	0	0	
D	0	0	1	0	
E	2	1	1	0	

Resources still needed

E = (6342)  
P = (5322)  
A = (1020)

- » These matrices are just C and R earlier.
- » Processes must state their total resource needs before executing, so that the system can compute the right-hand matrix at each instant.

# Banker's Algorithm for Multiple Resource

- » The algorithm for checking to see if a state is safe can now be stated.
  1. Look for a row,  $R$ , whose unmet resource needs are all smaller than or equal to  $A$ . If no such row exists, the system will eventually deadlock since no process can run to completion
  2. Assume the process of the chosen row requests all the resources it needs and finishes. Mark that process as terminated and add all of its resources to the  $A$  vector.
  3. Repeat steps 1 and 2 until either all processes are marked terminated (in which case the initial state was safe) or no process is left whose resource needs can be met (in which case the system was not safe).

# Banker's Algorithm for Multiple Resource

- » Every OS book since 1965 covers the Banker's Algorithm.
- » In theory the algorithm is wonderful, in practice it is essentially useless because processes rarely know in advance what their maximum resource needs will be.
  - Process count also varies over the life of a system.

# Deadlock Prevention

- » Deadlock avoidance is essentially impossible
  - Can't predict the future
  - Instead, prevent one of the four conditions
- » Attacking the Mutual-Exclusion Condition
  - If no resource were ever assigned exclusively to a single process, we would never have deadlocks.
    - Make data read only
    - Spool print jobs to disk
  - Avoid assigning a resource unless absolutely necessary, and try to make sure that as few processes as possible may actually claim the resource.

# Deadlock Prevention

- » Attacking the Hold-and-Wait Condition
  - If we can prevent processes that hold resources from waiting for more resources, we can eliminate deadlocks.
  - Require all processes to request all their resources before starting execution.
    - How do they know?
  - Not optimal utilization
  - Variation: require a process requesting a resource to first temporarily release all the resources it currently holds.



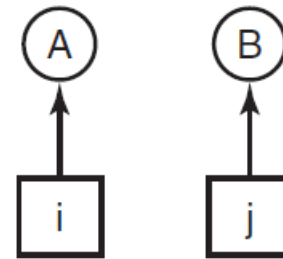
# Deadlock Prevention

- » Attacking the No-Preemption Condition
  - Can't steal real printer. Virtualize.
  - Can't virtualize your database

# Deadlock Prevention

1. Imagesetter
2. Printer
3. Plotter
4. Tape drive
5. CD-ROM drive

(a)



(b)

## » Attacking the Circular Wait Condition

- Process is entitled only to a single resource at any moment.
- Number the resources globally.
  - Requests must be made in numeric order

# Two-Phase Locking

- » Two step process
  - Lock all resources needed
  - Do the work
- » If a process can't lock all the resources it releases all currently held and starts over
  - Not acceptable to just terminate a process partway through because a resource is not available
  - Requires careful arrangement by the programmer

# Communication Deadlocks

- » Resource deadlock is a problem of competition synchronization.
- » Suppose process A sends a request message to process B, and then blocks until B sends back a reply message.
  - Suppose that the request message gets lost.
  - A is blocked waiting for the reply. B is blocked waiting for a request asking it to do something.  
We have a communication deadlock.
  - Use timeouts