

# Deadlocks

Fall 2016

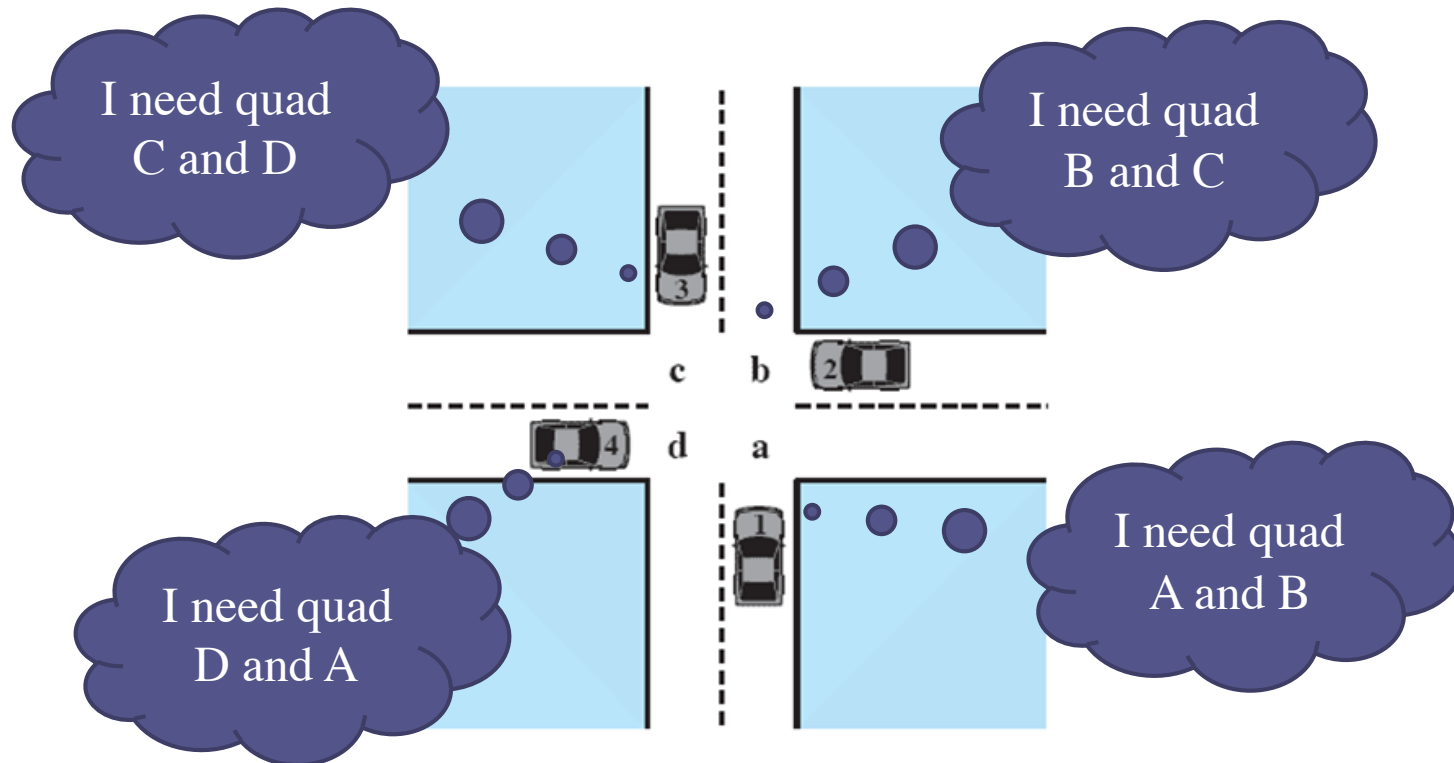
# Reading

- Chapter 6, Sec: 6.1,6.2, 6.4.1, 6.4.3, 6.5.2, 6.6, “Modern Operating Systems, Fourth Ed.”, Andrew S. Tanenbaum

# Outline

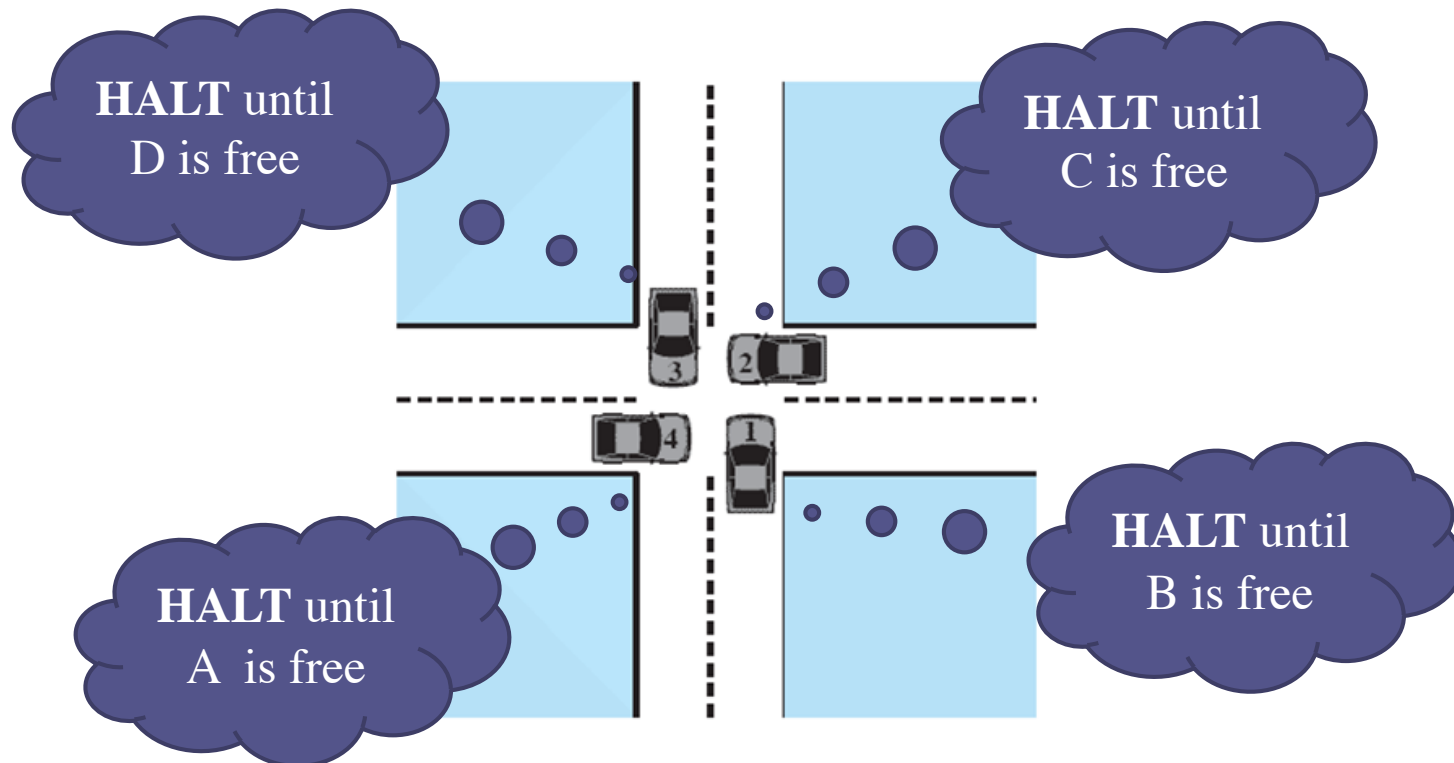
- **Introduction**
- Resources
- Conditions for Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and Recovery

# Potential Deadlock



William Stallings, "Operating Systems: Internals and Design Principles", 7th Edition

# Actual Deadlock



William Stallings, "Operating Systems: Internals and Design Principles", 7th Edition

# Outline

- Introduction
- **Resources**
- Conditions for Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and Recovery

# Preemptable and Nonpreemptable Resources

- Preemptable resource: one that can be taken away from the process owning it with no ill effects
  - Example: memory in PCs
- Nonpreemptable resource: one that cannot be taken away from its current owner without potentially causing failure
  - Example: blu-ray, printer
- Deadlocks involve nonpreemptable resources


# Using Resources

- Abstract sequence of events required to use a resource
  1. Request the resource
  2. Use the resource
  3. Release the resource
- If resource request fails, the process will block or keep trying to acquire the resource

# Deadlock Example

- Threads need exclusive access to multiple resources at the same time
- Example: scanner (RX) and DVD burner (RY)

Thread A	Thread B
Lock (RX)	Lock (RY)
Lock (RY)	Lock (RX)
.....	.....
UnLock(RY)	UnLock(RX)
UnLock(RX)	UnLock(RY)



# Revisiting the Example

- Example: scanner (RX) and DVD burner (RY)
- Note: order of acquiring locks is important

Thread A	Thread B
Lock (RX)	Lock (RX)
Lock (RY)	Lock (RY)
.....	.....
UnLock(RY)	UnLock(RY)
UnLock(RX)	UnLock(RX)

# Outline

- Introduction
- Resources
- **Conditions for Deadlocks**
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and Recovery

# Deadlocks

- 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
- Notes:
  - All the processes in the set are blocked, therefore any of them will not initiate the event that wakes other processes in the set
  - Blocking is permanent
  - None of the processes in the set can release the resources it already have

# Conditions for Resource Deadlocks

- Four conditions must be present for a resource deadlock to occur
  - Mutual exclusion condition. Each resource is either currently assigned to exactly one process or is available.
  - Hold-and-wait condition. Processes currently holding resources that were 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 list of two or more processes, each of which is waiting for a resource held by the next member of the chain

evidence necessary

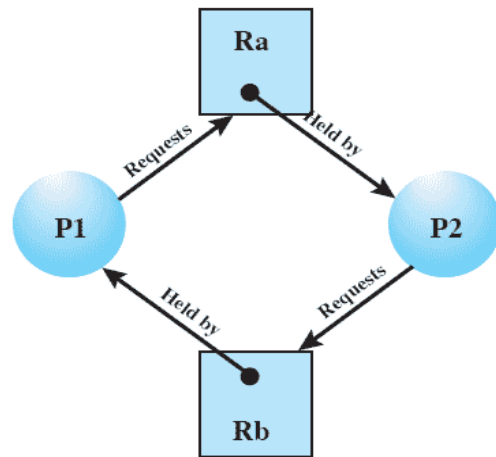
# Resource Allocation Graph



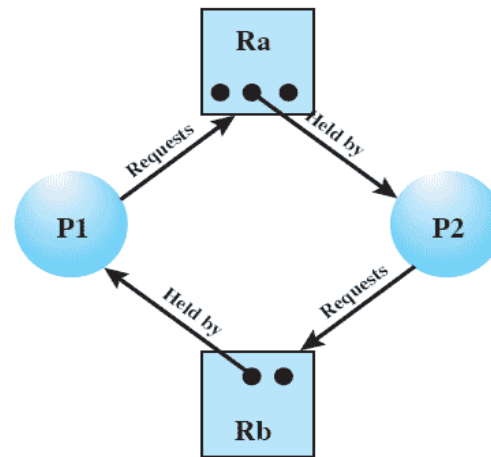
(a) Resource is requested



(b) Resource is held

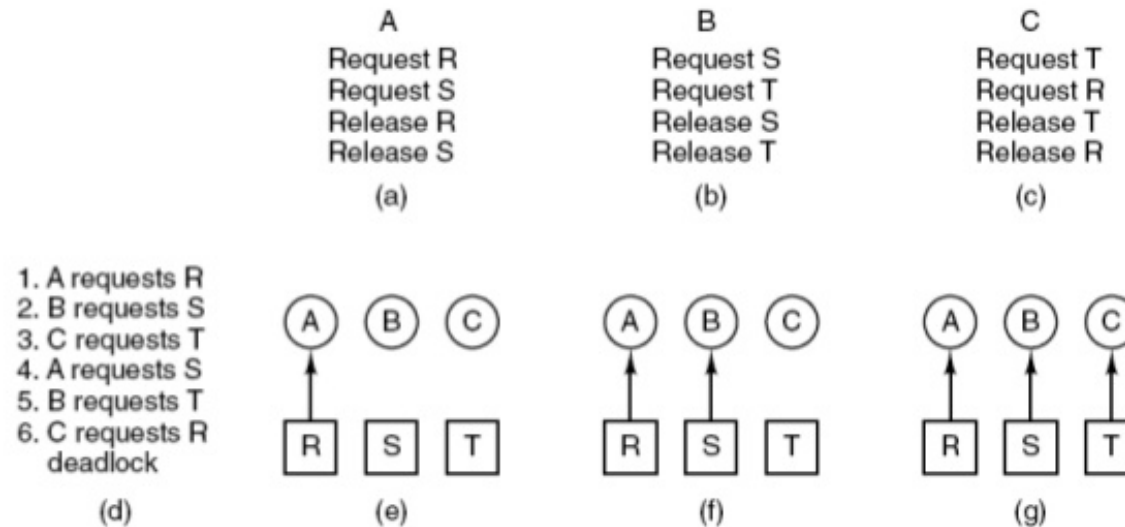


(c) Circular wait



(d) No deadlock

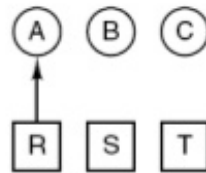
# Resource Allocation Graph - Example(1)



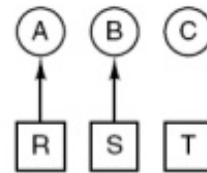
# Resource Allocation Graph - Example(2)

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

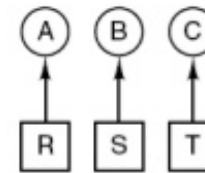
(d)



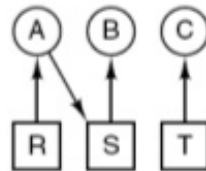
(e)



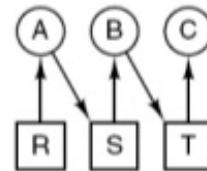
(f)



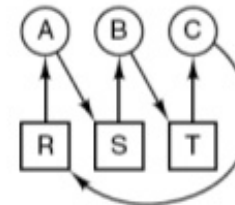
(g)



(h)



(i)

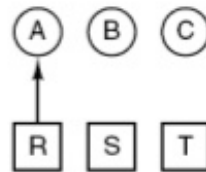


(ii)

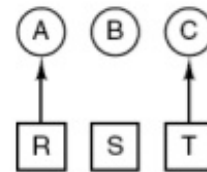
# Resource Allocation Graph - Example(3)

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

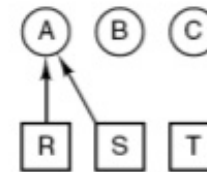
(k)



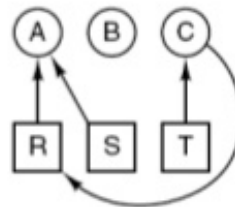
(l)



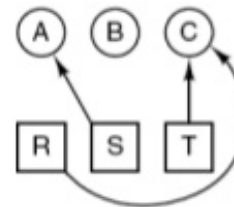
(m)



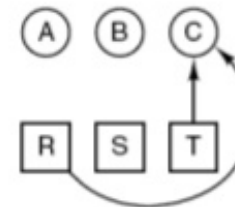
(n)



(o)



(p)



(q)

# Dealing with Deadlocks

- Three Strategies:
  - Detection and recovery. Let them occur, detect them, and take action.
  - Dynamic avoidance by careful resource allocation.
  - Prevention, by structurally negating one of the four conditions.

# Outline

- Introduction
- Resources
- Conditions for Deadlocks
- **Deadlock Prevention**
- Deadlock Avoidance
- Deadlock Detection and Recovery

# Prevention - Eliminating Mutual Exclusion

- No resources are assigned exclusively to a single process
  - If data can be read only, and therefore shared by many processes
  - Not realistic. Example: printer
- Applicable idea:
  - Avoid assigning a resource unless absolutely necessary
  - Try to make sure that as few processes as possible may actually claim the resource

# Prevention - Eliminating Hold-and-Wait

- This goal can be achieved by enforcing that a process needs to acquire all of its resources before starting execution
- Cons:
  - Not all the processes know all the resources they will need before starting
  - Not optimal usage of resources
- Another alternative: temporarily release all the resources a process currently holds to request a new resource

# Prevention - Eliminating No-Preemption

- A process can release a resource if needed by another process
- Cons:
  - Not possible in some cases. Example, a process is in the middle of printing a file
- Suitable for resources that can be virtualized. Example memory, printer spooler

# Prevention - Eliminating Circular Wait

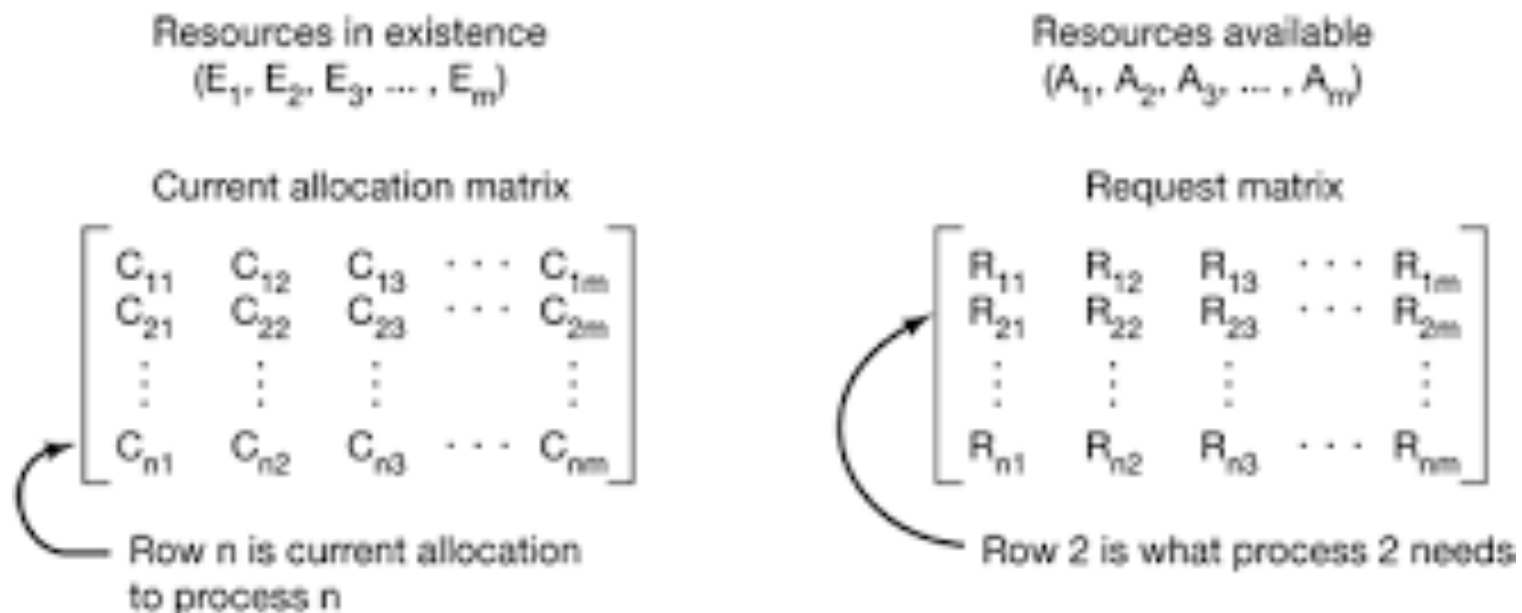
- Eliminating circular wait:
  - A process is entitled only to a single resource at any moment. If it needs a second one, it must release the first one
  - Provide a global numbering of all the resources. Resources can only be acquired in that order
- Cons:
  - What order is suitable for all applications?
  - There can be a very large number of resources, and numbering them might not be straightforward

# Outline

- Introduction
- Resources
- Conditions for Deadlocks
- Deadlock Prevention
- **Deadlock Avoidance**
- Deadlock Detection and Recovery

# Deadlock Avoidance

- The system dynamically decide whether granting a resource is **safe or not** and make the allocation only when it is safe



# Safe and Unsafe States

- 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 resources immediately
- An unsafe state is not a deadlock state
  - It means that there is a potential of deadlock

# Safe and Unsafe States - Example

- $E=[10]$ : there are 10 instances of the resource
- Safe state example, a is safe

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

- Unsafe state example, b is unsafe

Has Max			Has Max			Has Max			Has Max		
A	3	9	A	4	9	A	4	9	A	4	9
B	2	4	B	2	4	B	4	4	B	—	—
C	2	7	C	2	7	C	2	7	C	2	7
Free: 3			Free: 2			Free: 0			Free: 4		
(a)			(b)			(c)			(d)		

# Banker's Algorithm

- Proposed by Dijkstra in 1965
- Based on how banks worked at that time: do not grant money to a customer unless he/she can repay it
- Algorithm:
  - Find row,  $R$ , whose unmet resource needs are all smaller than or equal to  $A$ . If no such row is found, “unsafe”
  - Assume the process of the chosen row requests all the resources. Mark that process as terminated and add all of its resources to the  $A$  vector.
  - Repeat steps 1 and 2 until either all processes are marked terminated (“safe”), some processes are cannot terminate (“unsafe”)

# Banker's Algorithm - Example 1 - Step 1

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

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 needed

6	3	4	2
---	---	---	---

Max resources

5	3	2	2
---	---	---	---

Total assigned

1	0	2	0
---	---	---	---

Available

- **D can get its resources and terminate**

# Banker's Algorithm - Example 1 - Step 2

A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	0	0	0	1
E	0	0	0	0

Resources assigned

A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	<hr/>			
E	2	1	1	0

Resources needed

6	3	4	2
---	---	---	---

Max resources

4	2	2	1
---	---	---	---

Total assigned

2	1	2	1
---	---	---	---

Available

- A or E can get all its resources and terminate

# Banker's Algorithm - Example 1 - Step 3

A	0	0	0	0
B	0	1	0	0
C	1	1	1	0
D	0	0	0	1
E	0	0	0	0

Resources assigned

A	<hr/>			
B	0	1	1	2
C	3	1	0	0
D	<hr/>			
E	2	1	1	0

Resources needed

6	3	4	2
---	---	---	---

Max resources

1	2	1	0
---	---	---	---

Total assigned

5	1	3	2
---	---	---	---

Available

- All remaining processes can get their resources and terminate

**Safe**

# Banker's Algorithm - Example 2 - Step 1

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

A	1	1	0	0
B	0	1	2	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources needed

6	3	4	2
---	---	---	---

Max resources

5	3	3	2
---	---	---	---

Total assigned

1	0	1	0
---	---	---	---

Available

- D can complete
- A or E can complete
- Other processes can complete
- Now, E requests one item from third resource (is it safe?)

Therefore, this is a **safe state**

# Banker's Algorithm - Example 2 - Step 2

A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	1	0

Resources assigned

A	1	1	0	0
B	0	1	2	2
C	3	1	0	0
D	0	0	1	0
E	2	1	0	0

Resources needed

6	3	4	2
---	---	---	---

Max resources

5	3	4	2
---	---	---	---

Total assigned

1	0	0	0
---	---	---	---

Available

- No process can complete
- Therefore, this is a **unsafe state**

# Evaluating Banker's Algorithm

- In theory the algorithm is wonderful, in practice it is essentially useless
  - Processes usually do not know in advance the maximum resources they will need
  - The number of processes is not fixed
  - The number of available resources is not fixed (a DVD can break or we plug a new resource)
- In reality, use heuristics based on Banker's algorithm
  - Example: throttle network traffic when buffer utilization reaches higher than, say, 70%

# Outline

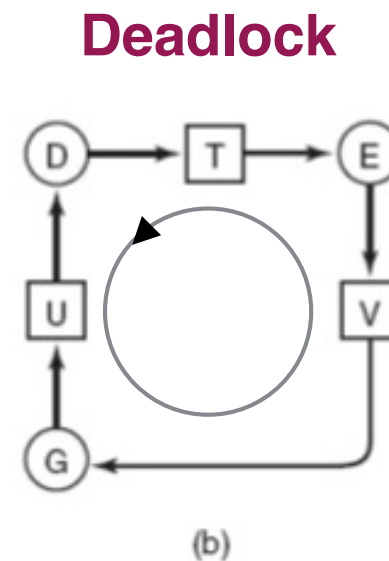
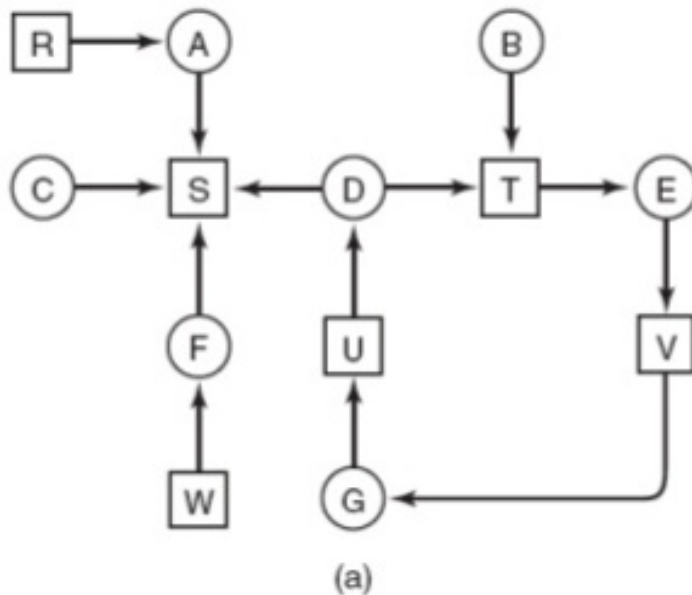
- Introduction
- Resources
- Conditions for Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- **Deadlock Detection and Recovery**

# Deadlock Detection and Recovery

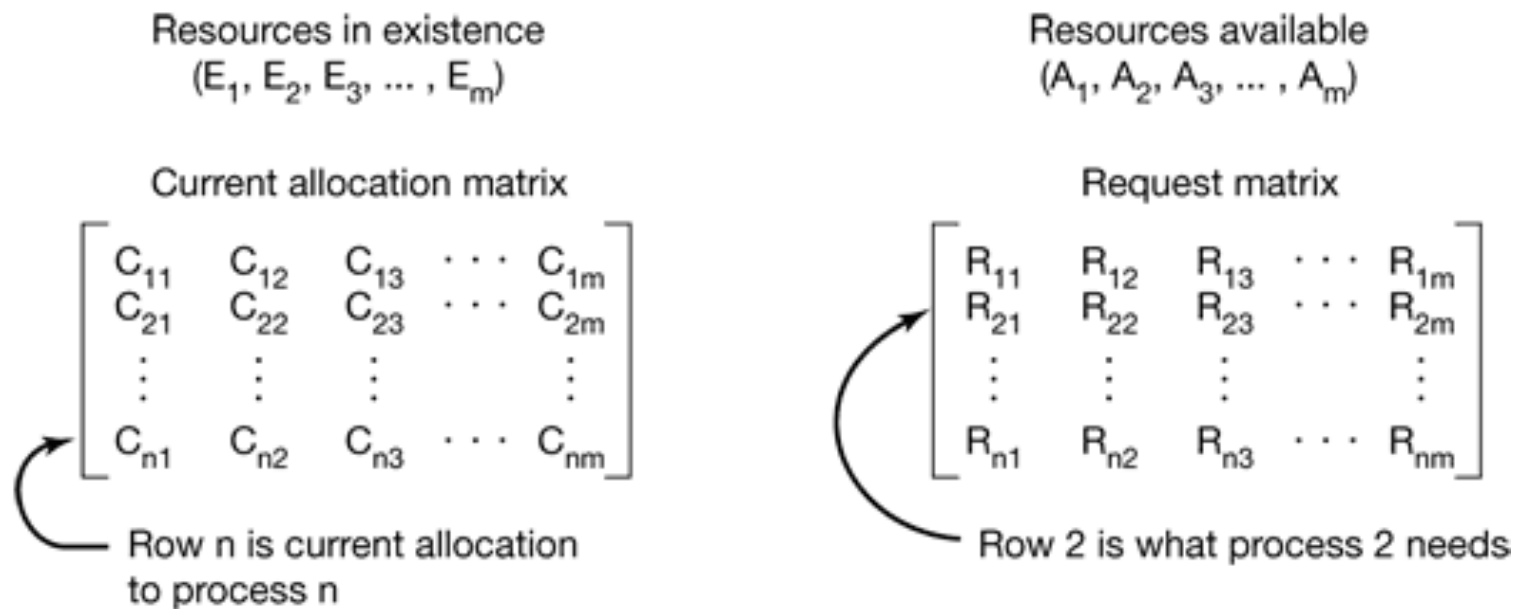
- Steps:
  - No ahead of time steps, deadlocks can occur
  - Detect when a deadlock occurs
  - Take an action to recover from this deadlock

# Deadlock Detection Using Resource Allocation Graphs

1. Process A holds R and wants S.
2. Process B holds nothing but wants T.
3. Process C holds nothing but wants S.
4. Process D holds U and wants S and T.
5. Process E holds T and wants V.
6. Process F holds W and wants S.
7. Process G holds V and wants U.



# Deadlock Detection with Multiple Resources of Each Type - Data Structures



$$\sum_{i=1}^n C_{ij} + A_j = E_j$$

# Deadlock Detection with Multiple Resources of Each Type

- Overview:
  - Each process is initially unmarked
  - As the algorithm progresses, processes will be marked, indicating that they can complete and are not deadlocked
- Deadlock detection algorithm
  - Look for an unmarked process,  $P_i$ , for which the  $i^{\text{th}}$  row of  $R$  is less than or equal to  $A$
  - If such a process is found, add the  $i^{\text{th}}$  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

# Deadlock Detection - Example 1

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

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

Current allocation matrix

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

Request matrix

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

- P3 can complete,  $A = (2 \ 2 \ 2 \ 0)$
- P2 can complete,  $A = (2 \ 2 \ 2 \ 0)$
- P1 can complete, therefore, no deadlock

# Deadlock Detection - Example 1

0	0	1	0
2	0	0	1
0	1	2	0

C, Resources assigned

2	0	0	1
1	0	1	0
2	1	0	0

R, Resources needed

4	2	3	1
---	---	---	---

E, Max resources

2	1	0	0
---	---	---	---

A, Available

- P3 can complete, A = (2 2 2 0)
- P2 can complete, A = (2 2 2 0)
- P1 can complete, therefore, no deadlock

# Deadlock Detection - Example 2

0	0	1	0
2	0	0	1
0	1	2	0

C, Resources assigned

2	0	0	1
1	0	1	0
2	1	1	0

R, Resources needed

4	2	3	1
---	---	---	---

E, Max resources

2	1	0	0
---	---	---	---

A, Available

- None of the processes can complete
- Deadlock

# Recovery from Deadlock

- Preemption: reallocate resources
  - Difficult or impossible
- Rollback: process is reset to an earlier moment when it did not have the resource undoing any completed operations
  - Checkpointing might be used: a process state is written to a file so that it can be restarted later
- Killing processes: kill one or more processes involved in the circular wait
  - Choose process with fewer resources, can easily be restarted, fewer computations

# Discussion

# Communication Deadlocks

- Processes communication by sending/receiving messages can deadlock
- Example,
  - A sends a request to B, and blocks waiting for B's reply
  - The message got lost
  - A is waiting for B's reply and B is waiting for A's request
- Timeouts are used

# Starvation

- Some processes never get service
- There is no forward progress in both deadlock and starvation
  - Progress can eventually be made in case of starvation
  - In the deadlock case, all processes involved are blocked

Thank You!