CISC 7310X
C10: Deadlocks

Hui Chen

Department of Computer & Information Science

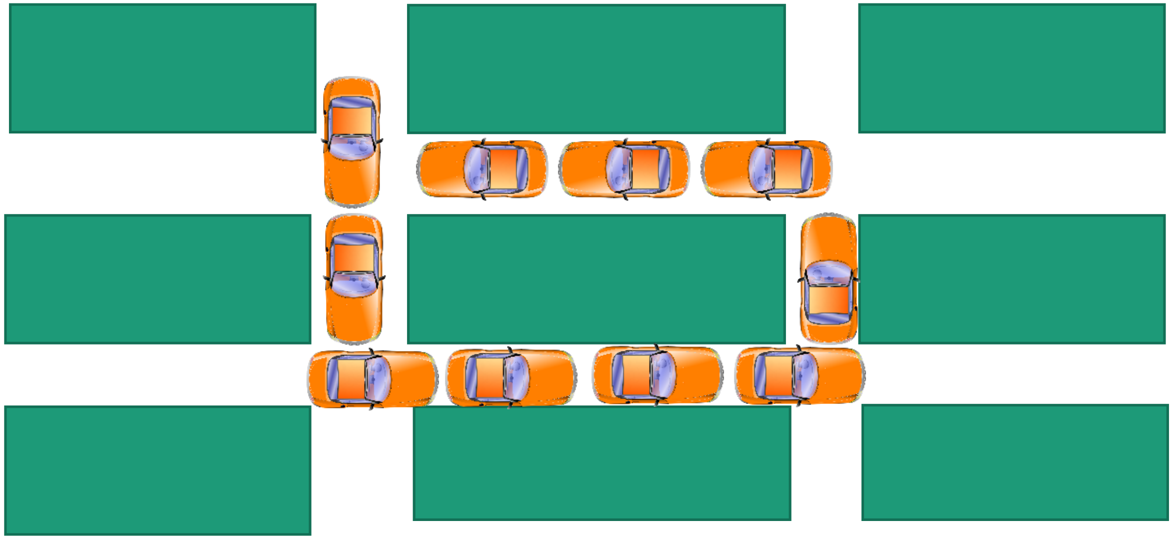
CUNY Brooklyn College

Outline

- Concept of deadlock
- Necessary conditions
- Models of deadlocks
  - Resource allocation graph
  - Matrix-based model
- Deadlock detection and recovery
- Deadlock avoidance
- Deadlock prevention
- Resource deadlocks and communication deadlocks
- Livelock and starvation

Problem when Sharing Resources

- A proposed legislature in the history
  - "When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone."



Deadlock

- Multiple processes share resources
- Two or more waiting processes request the same resources, and can not change state

System Model

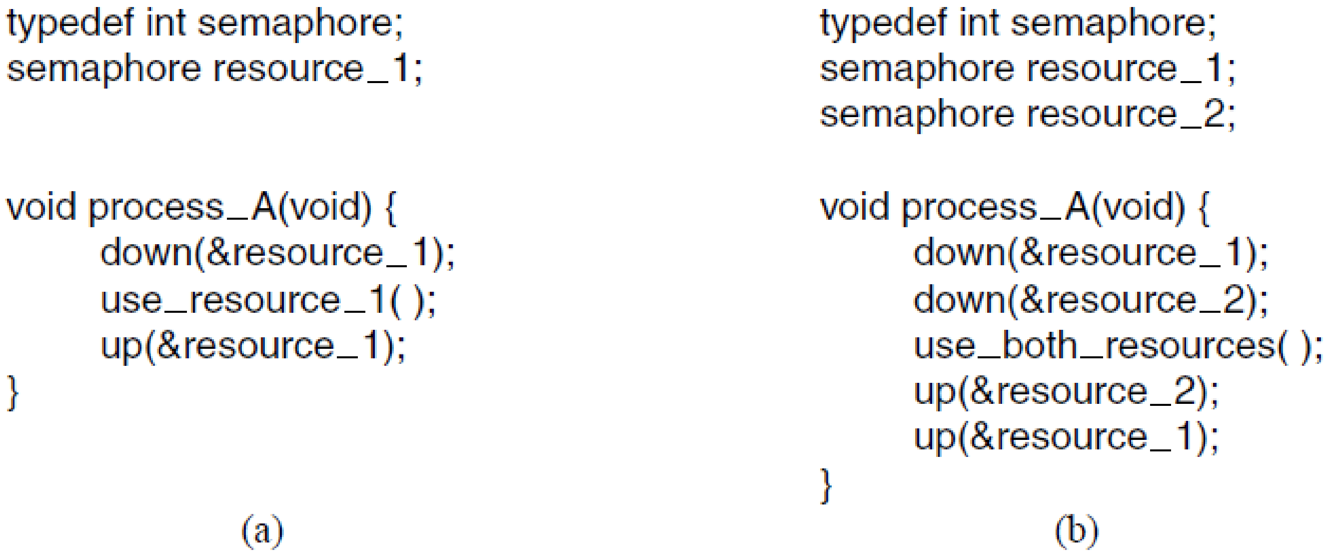
- N resources distributed among M Processes
- Non-preemptable resources
  - Request the resource
  - Use the resource
  - Release the resource
- Deadlock
  - Every process in the set is waiting for an event to be triggered by another in the set (request or release resource)

Deadlock: Example

- Example
  - 2 processes, P1 and P2 share two 2 CD-RW drives (D1, D2)
  - P1 is using D1, P2 is using D2
  - P1 requests D2 before releasing D1; P2 requests D1 before releasing D2
- We must be careful when designing multi-threaded/multi-processed applications

Resource with Semaphore or Mutex

- *Access non-preemptive resource with* semaphore (request, use, release)



```
typedef int semaphore;
semaphore resource_1;


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

(a)

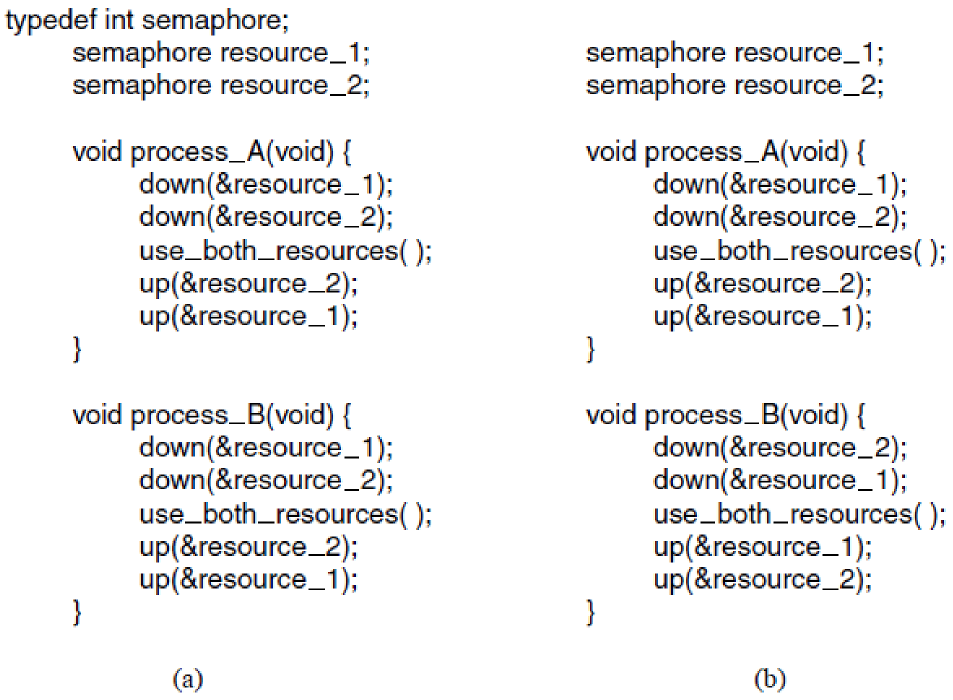
```
typedef int semaphore;
semaphore resource_1;
semaphore resource_2;

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

(b)

- [Figure 6-1 in Tanenbaum & Bos, 2014]

Sharing Resources

- Deadlock free and deadlock



```
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)

- [Figure 6-2 in Tanenbaum & Bos, 2014]

Remarks

- In previous example (Figure 6-2 (b)), whether the deadlock happens or not depends on the result of a race
  - Difficult to debug because it only happens sporadically
  - Difference between deadlock free and deadlocked code is subtle in coding style

Deadlock: Formal Definition

- A set of processes are deadlocked if
    - each process in the set waiting for an event
    - and that event can be caused only by another process

Necessary Conditions

- *A* deadlock can arise if the following 4 conditions hold simultaneously in a system (Coffman et al., 1971)

1. Mutual exclusion
2. Hold-and-wait
3. No preemption
4. Circular wait

Mutual Exclusion

- *A* shared resource must be shared in a mutual exclusive fashion, i.e.,
- *A* resource is either currently assigned to exactly one process or is available

Hold-and-Wait

- Process holding a resource is allowed to wait for another, i.e.,
- Process currently holding resources that were granted earlier can request new resources

No Preemption

- One process cannot preemptively take a resource from another process, i.e.,
- Resources previously granted cannot be forcibly taken away from a process, and they must be explicitly released by the process holding them.

Circular Wait

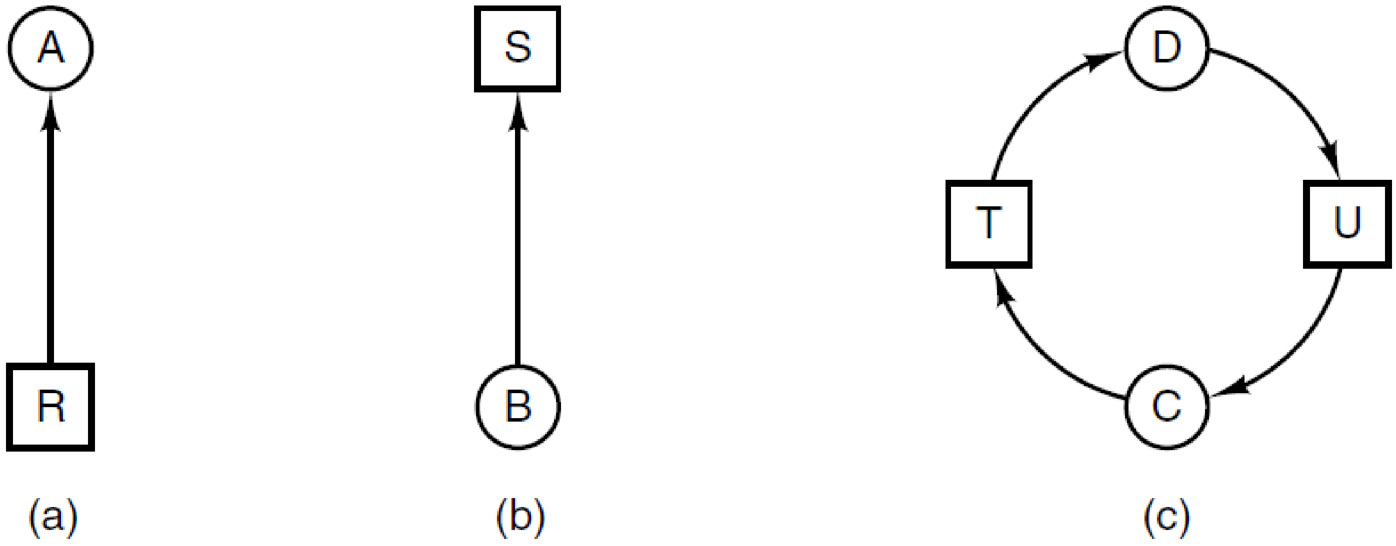
- There must be a circular list of two or more processes, each of which is wait for a resource held by another process in the list, e.g.,
- {P1, P2, P3}: P1 is waiting for P2 (to release a resource), P2 is waiting for P3, and P3 is waiting for P1.

Modeling Deadlocks

- Resource allocation graph (Holt, 1972)

Resource-Allocation Graph

- Circle: process; Square: resource; arrow: (Resource → Process, Process → Resource, i.e., is being held/assigned to or requests)



- Resource allocation graphs. (a) Holding a resource. (b) Requesting a resource. (c) Deadlock. [Figure 6-3 in Tanenbaum & Bos, 2014]

Resource-Allocation Graph Modeling: Example

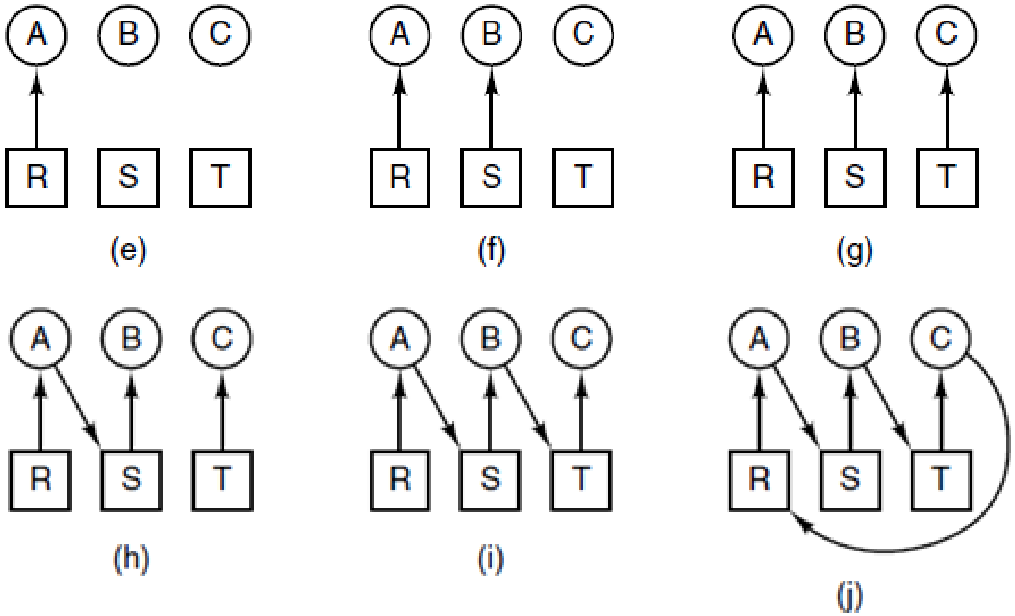
- Three processes: A, B, C
- Three resources: R, S, T

Schedule with Deadlock

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)

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) (j)

Schedule without Deadlock

A
Request R
Request S
Release R
Release S

(a)

B
Request S
Request T
Release S
Release T

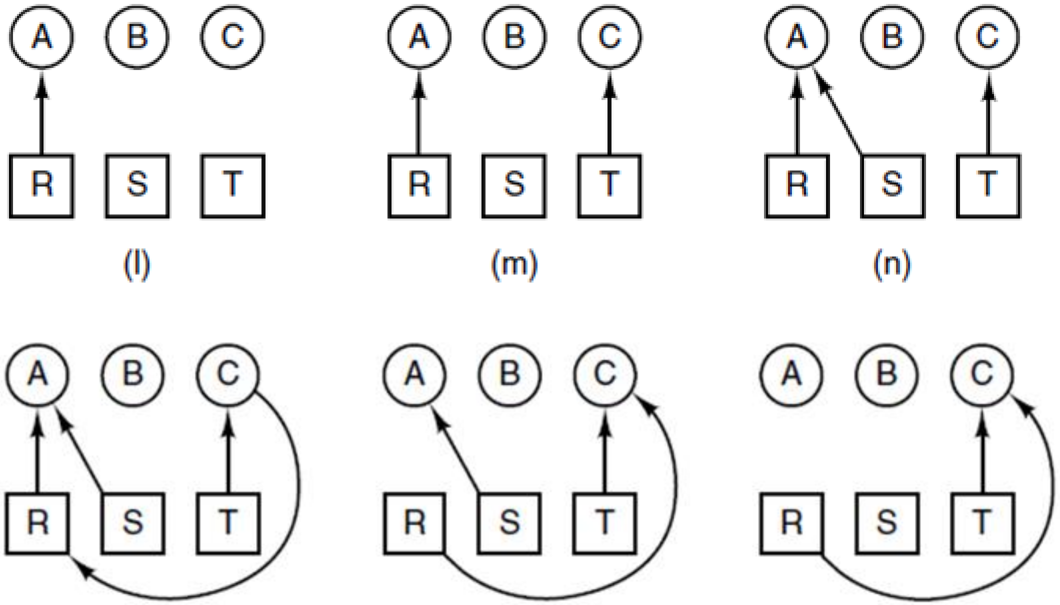
(b)

C
Request T
Request R
Release T
Release R

(c)

1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S
   no deadlock

(k)



(l) (m) (n)

(o) (p) (q)

Deadlock Handling Strategies

1. The Ostrich Algorithm. Ignore the problem, maybe it will go away.
2. Detection and recovery. Let deadlocks occur, detect them, and take action.
3. Dynamic avoidance. Carefully allocate resources.
4. Prevention. By structurally negate one of the four required conditions.

The Ostrich Algorithm

The deadlock regarding the resources in my system happens once in a blue moon …

Detection and Recovery

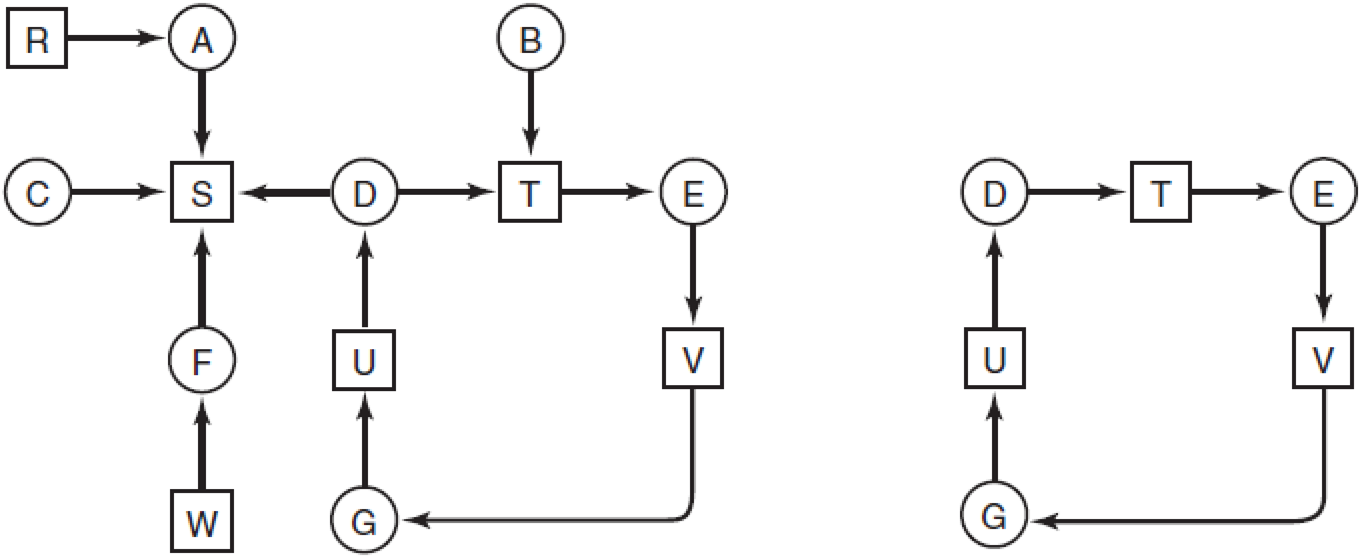
- One resource of each type
- Several instances of a resource type

One Resource of Each type: Example

Example of a system: is it deadlocked?

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

Resource Allocation Graph



(a) (b)

- [Figure 6-5 in Tanenbaum & Bos, 2014]

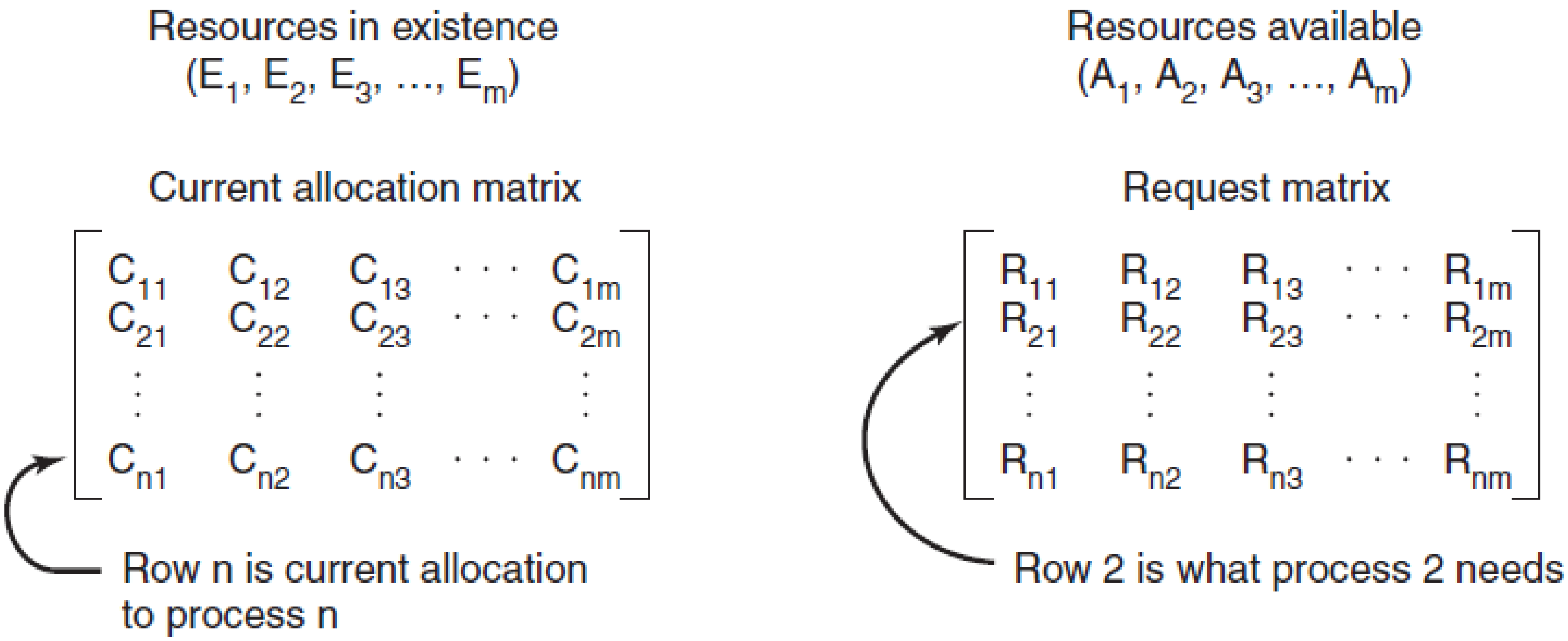
Detecting Cycle in Resource Allocation Graph

- For each node in the graph
    - Do a depth first search, check if cycle exists
- Complexity of the algorithm: $O(N^2)$

Several Instances of a Resource

- Detection algorithm: n processes, m types of resources. Data structures:
  - E: Existing Resources. A vector length of m indicates total number of resources of each type of resources in existence
  - A: Available. A vector length of m indicates the number of available resources of each type
  - C: Current Allocation. An n x m matrix defines the number of resources each type currently allocated to each process
  - R: Request. An n x m matrix indicates the current request of each process.
  - Invariance: $\sum_{i=1}^{n} C_{ij} + A_j = E_j$

Data Structures



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

Current allocation matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

Row n is current allocation to process n

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

Request matrix

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Row 2 is what process 2 needs

- [Figure 6-6 in Tanenbaum & Bos, 2014]

Detection Algorithm

- Basic operation: comparing two vectors X and Y
  - $X \leq Y$ holds if and only if $X_i \leq Y_i$ for $1 \leq i \leq m$.
- Each process is initially unmarked

1. Look for unmarked process, P_i , for which the i-th row of R, $R_i \leq A$.
   - With available resources, can P_i run to completion?
2. If such a process is found, add the i-th row of C to A, mark the process, go back to step 1.
   - Its resources become available to others when P_i run to completion
3. If no such process exists, algorithm terminates

- Deadlock if there still exists an unmarked process

Detection Algorithm: Example

- Is there a deadlock?

$$E = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (\,4 & 2 & 3 & 1\,) \end{array} \qquad A = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (\,2 & 1 & 0 & 0\,) \end{array}$$

Current allocation matrix

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

Request matrix

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

- [Figure 6-7 in Tanenbaum & Bos, 2014]

Recovery from Deadlock

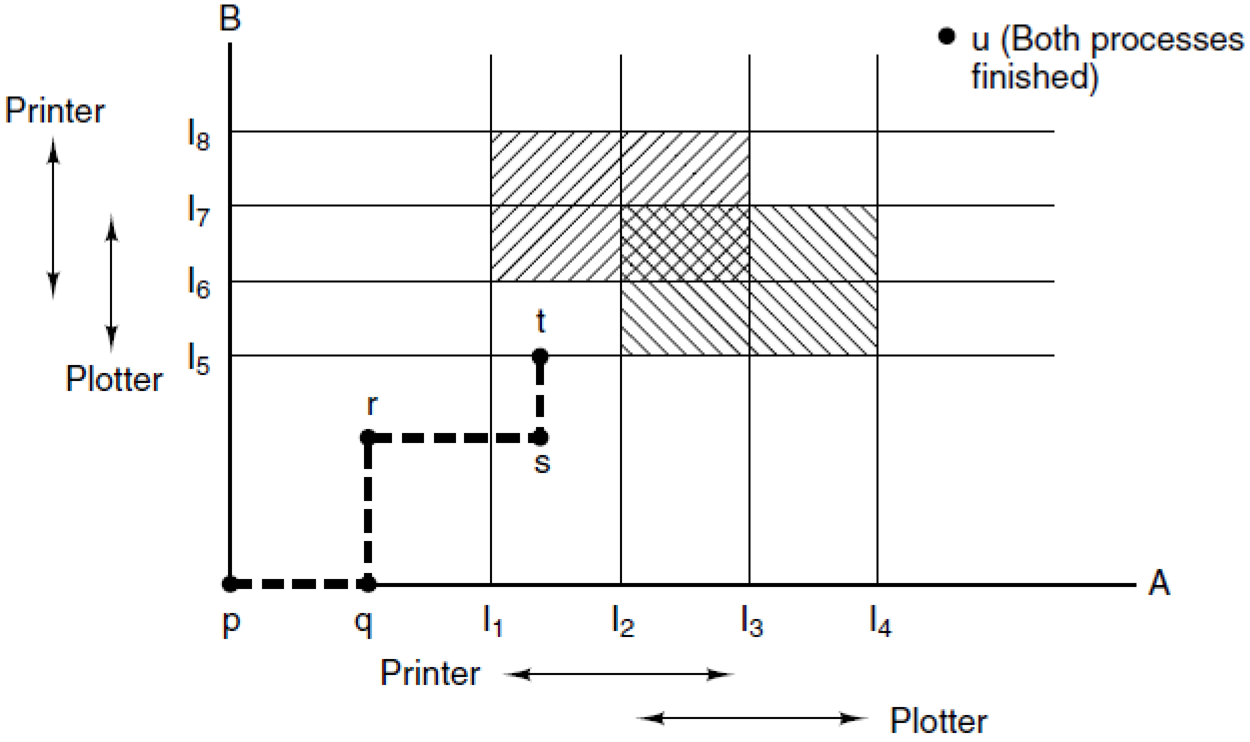
- Possible Methods of recovery (though none are “attractive”):

1. Preemption
2. Rollback
3. Killing processes

Deadlock Avoidance

- Now assume a process requests a resource at a time.
- Question: can the system decide whether it is safe (without causing a deadlock) to granting the resource to the resource upon the request?

Deadlock Avoidance: Resource Trajectory



- [Figure 6-8 in Tanenbaum & Bos, 2014]

Safe and Unsafe State

- Safe state
  - The system can allocate resources to each process in some order and still avoid a deadlock
  - A safe state is not a deadlocked state
- Unsafe state
  - A deadlocked state is an unsafe state
  - An unsafe state may not be a deadlock state
  - An unsafe state is a state that may lead to a deadlock

Safe State: Example

- State consists of vectors & matrices, E, A, C, R
- A resources has 10 instances
- Does exist a scheduling order of processes A, B, C, and allow all of them to complete?
  - The following sequence shows that (a) is safe

	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

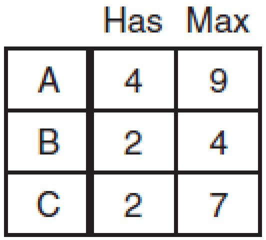
Free: 0
(d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7
(e)

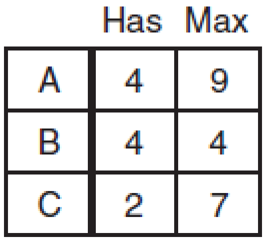
Unsafe State: Example

- A resources has 10 instances
- Does exist a scheduling order of processes A, B, C, and allow all of them to complete?
  - (b) is unsafe: you can run B to completion, but no sufficient resources for A or C to complete



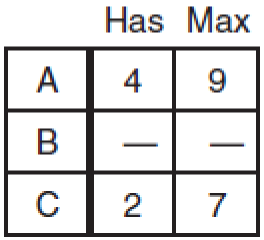
	Has	Max
A	4	9
B	2	4
C	2	7

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)

Banker's Algorithm

- Due to Dijkstra (1965)
- Banker's algorithm for a single resource
- Banker's algorithm for multiple resources

Single Resource: Example

- Three resource allocation states:
  (a) Safe. (b) Safe. (c) Unsafe.

	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)

Multiple Resources: Example

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)

Banker's Algorithm for Multiple Resources

1. Look for a row, R, whose unmet resource needs are all smaller than or equal to *A*. If no such row exists, system will eventually deadlock.
2. Assume the process of row chosen requests all resources needed and finishes. Mark that process as terminated, add its resources to the *A* vector.
3. Repeat steps 1 and 2 until either all processes are marked terminated (safe state) or no process is left whose resource needs can be met (deadlock)

Deadlock Prevention

- Recall 4 necessary deadlock conditions
  - Break one, free of deadlocks
- Mutual exclusion
- Hold and wait
- No Preemption
- Circular wait

Attacking Mutual Exclusion

- Example
  - Make data read-only
  - Avoid assigning a resource unless absolutely necessary
  - Try to make sure as few processes possible may actually claim the resource

Breaking Hold-and-Wait

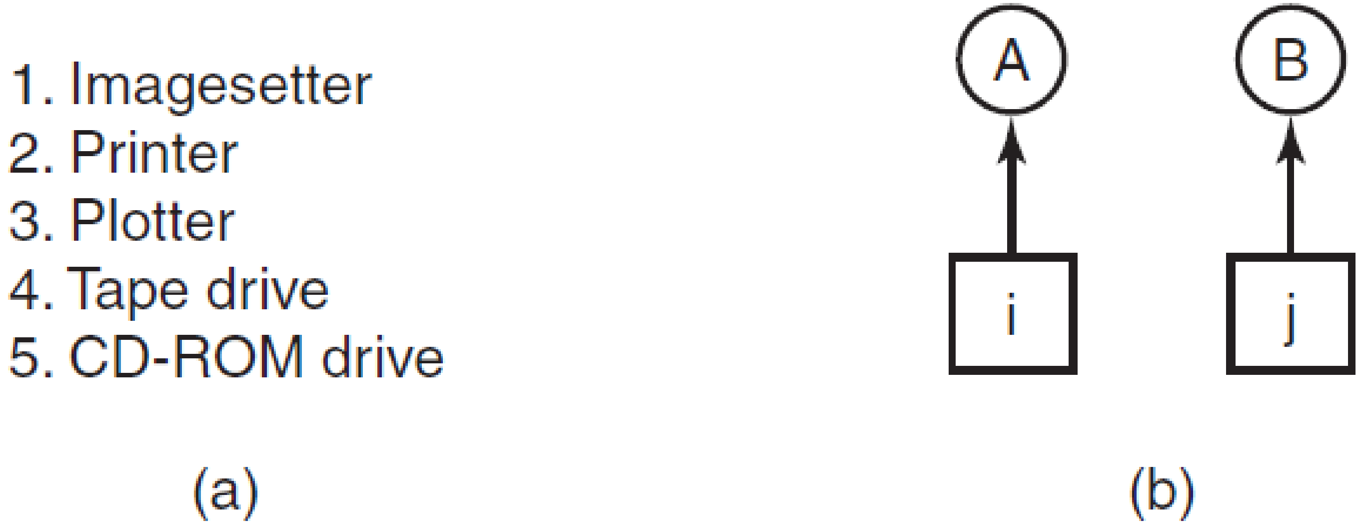
- Example
  - Require all processes to request all their resources before starting execution
    - Nothing or all, with all then run to completion
  - Require a process that is requesting resource to temporarily release all the resources it currently holds.

Attacking Non-Preemption

- Example
    - Use disk (assuming the space virtually infinite)

Breaking Circular Wait

- Order resources numerically, requests must be made in numerical order



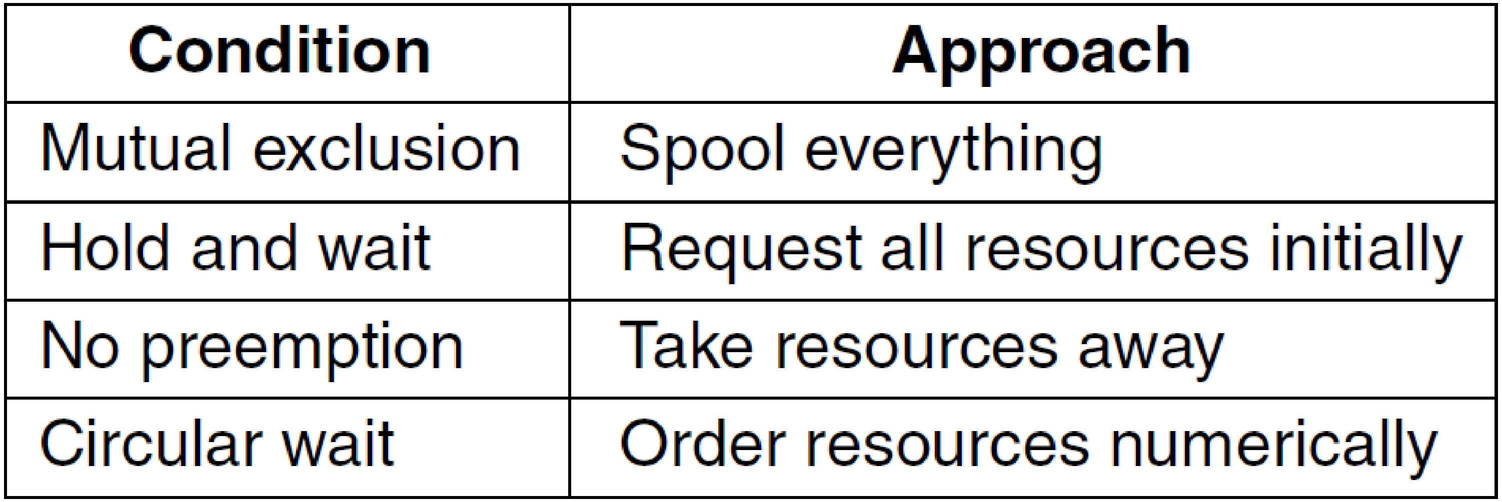
1. Imagesetter
2. Printer
3. Plotter
4. Tape drive
5. CD-ROM drive

(a) (b)

- [Figure 6-13 in Tanenbaum & Bos, 2014]

Deadlock Prevention: Summary

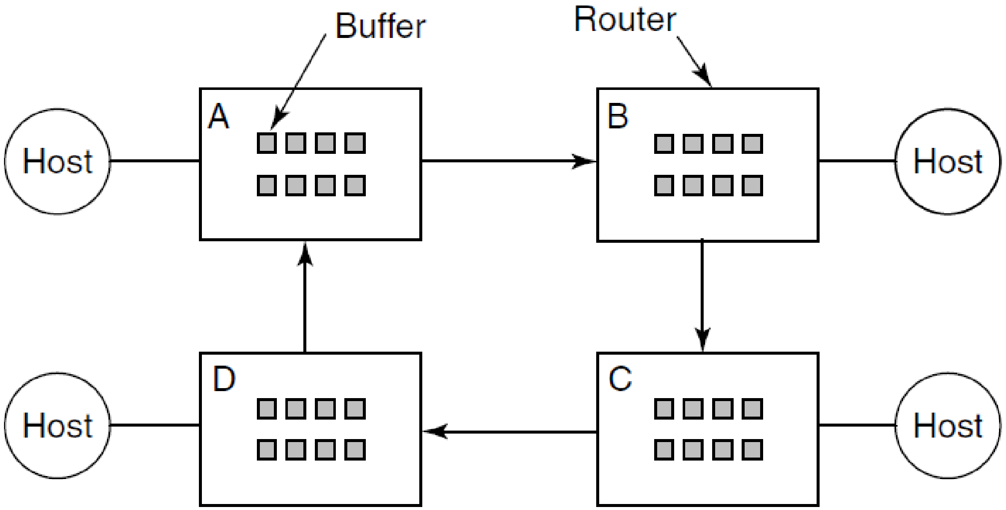
- Attacking the 4 necessary conditions



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

Communication Deadlocks

- Resource sharing is only one source of deadlocks



- [Figure 6-15 in Tanenbaum & Bos, 2014]

Livelock

- A process may choose to give up the lock (resource) it already acquired whenever it notices it cannot obtain the next lock (resource) it needs
- The process tries it again at a short delay
- Livelock
    - Processes may change states, but no progress is being made

Livelock: Example

- Busy-waiting can leads to livelock

```
void process_A(void) {
        enter_region(&resource_1);
        enter_region(&resource_2);
        use_both_resources( );
        leave_region(&resource_2);
        leave_region(&resource_1);
}
```

```
void process_B(void) {
        enter_region(&resource_2);
        enter_region(&resource_1);
        use_both_resources( );
        leave_region(&resource_1);
        leave_region(&resource_2);
}
```

Starvation

- A problem closely related to deadlock and livelock
- Example
  - N processes want to access a shared printer, which one should get it?
  - Policy
    - Choose a smallest file to print from the list of requests
    - Consider there is a constant stream of processes with short files, the process with a large file will have to wait indefinitely (to be starved off the resource)

Questions

- Concept of deadlock
- 4 necessary deadlock conditions
- How to deal with deadlocks?
    - Model: resource allocation graph
    - The Ostrich algorithm
        - Used most often by most operating systems (e.g., Unix and Windows)
        - Discussion thus very important for multithread/multiprocessed application developers
    - Deadlock detection and recovery
    - Deadlock (dynamic) avoidance
    - Deadlock prevention: attacking 4 necessary conditions
- Resources deadlocks and communication deadlocks
- Concepts of livelocks and starvation

Assignments

- Team
- Individual