Deadlocks
(Chapitre 6)

Francesco Bronzino
ArchiSys

ENS DE LYON

* Slides based on Tanenbaum, "Modern Operating Systems" 3e

Sujets

Deadlocks

Un **Deadlock** est une situation dans laquelle deux ou plusieurs threads (ou processus) sont bloqués

- Chacun attend une condition qui ne peut jamais se produire
- Le programme cesse de s'exécuter

Lorsque l'on utilise deux mutex ou plus, de telles situations peuvent se produire

- Il est nécessaire pour le programmeur de les prévoir et de les éviter

Exemple

Thread A:

```
pthread_mutex_lock(mutex1) ; // <--- LOCK 1
pthread_mutex_lock(mutex2) ; // <--- LOCK 2
... Section critique ...
pthread_mutex_unlock(mutex2) ;
pthread_mutex_unlock(mutex1) ;
```

Thread B:

```
pthread_mutex_lock(mutex2) ; // <--- LOCK 2
pthread_mutex_lock(mutex1) ; // <--- LOCK 1
... Section critique ...
pthread_mutex_unlock(mutex1) ;
pthread_mutex_unlock(mutex2) ;
```

Are deadlocks always a problem?

- Dans le partiel, j'ai vous ai demandé d'implémenter une solution basée sur une pipe pour le problème des writers/readers.
- Avez-vous dû envisager la possibilité d'un deadlock ?
- **Pourquoi ?**

Les deadlocks

Comment éviter les blocages :

- Utiliser d'autres types de synchronisation lorsque c'est possible :
    - Pipe, FIFO
- Utiliser un faible nombre de mutex
- Modélisation de l'utilisation de nombreux mutex

Preemptable and Nonpreemptable Resources

Sequence of events required to use a resource:

1. Request the resource.
2. Use the resource.
3. Release the resource.

A preemptable resource is one that can be taken away from the process owning it with no ill effects

Resource acquisition

One resource:

```
pthread_mutex_lock(mutex1) ; // <--- LOCK 1
... Section critique ...
pthread_mutex_unlock(mutex1) ;
```

Two resources:

```
pthread_mutex_lock(mutex1) ; // <--- LOCK 1
pthread_mutex_lock(mutex2) ; // <--- LOCK 2
... Section critique ...
pthread_mutex_unlock(mutex1) ;
pthread_mutex_unlock(mutex2) ;
```

Resource acquisition: Deadlock

Thread A:

```
pthread_mutex_lock(mutex1) ; // <--- LOCK 1
pthread_mutex_lock(mutex2) ; // <--- LOCK 2
... Section critique ...
pthread_mutex_unlock(mutex2) ;
pthread_mutex_unlock(mutex1) ;
```

Thread B:

```
pthread_mutex_lock(mutex2) ; // <--- LOCK 2
pthread_mutex_lock(mutex1) ; // <--- LOCK 1
... Section critique ...
pthread_mutex_unlock(mutex1) ;
pthread_mutex_unlock(mutex2) ;
```

Resource acquisition: Deadlock free

Thread A:

```
pthread_mutex_lock(mutex1) ; // <--- LOCK 1
pthread_mutex_lock(mutex2) ; // <--- LOCK 2
... Section critique ...
pthread_mutex_unlock(mutex1) ;
pthread_mutex_unlock(mutex2) ;
```

Thread B:

```
pthread_mutex_lock(mutex1) ; // <--- LOCK 1
pthread_mutex_lock(mutex2) ; // <--- LOCK 2
... Section critique ...
pthread_mutex_unlock(mutex1) ;
pthread_mutex_unlock(mutex2) ;
```

Introduction to deadlocks

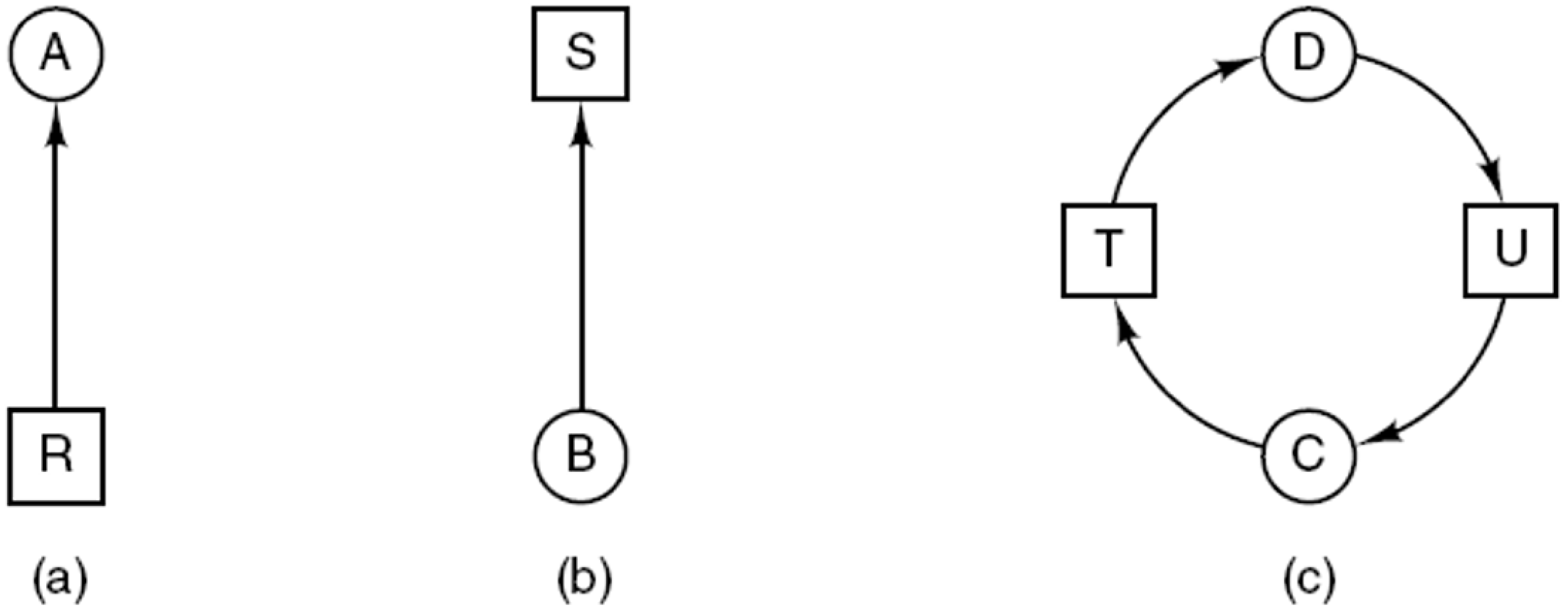
Formal definition:

```
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
```

Conditions for resource deadlocks

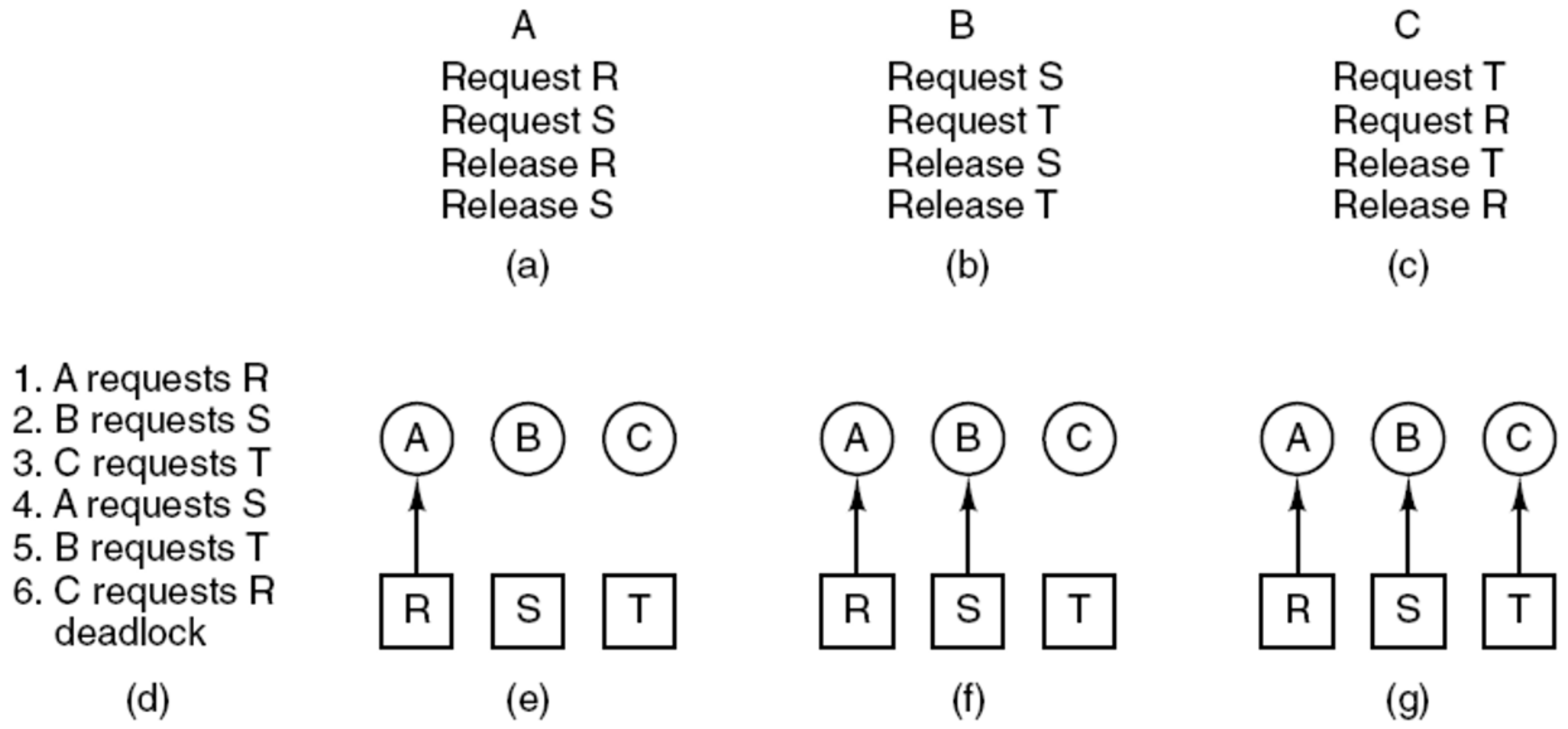
1. Mutual exclusion condition
2. Hold and wait condition
3. No preemption condition
4. Circular wait condition

Deadlock modeling (1)



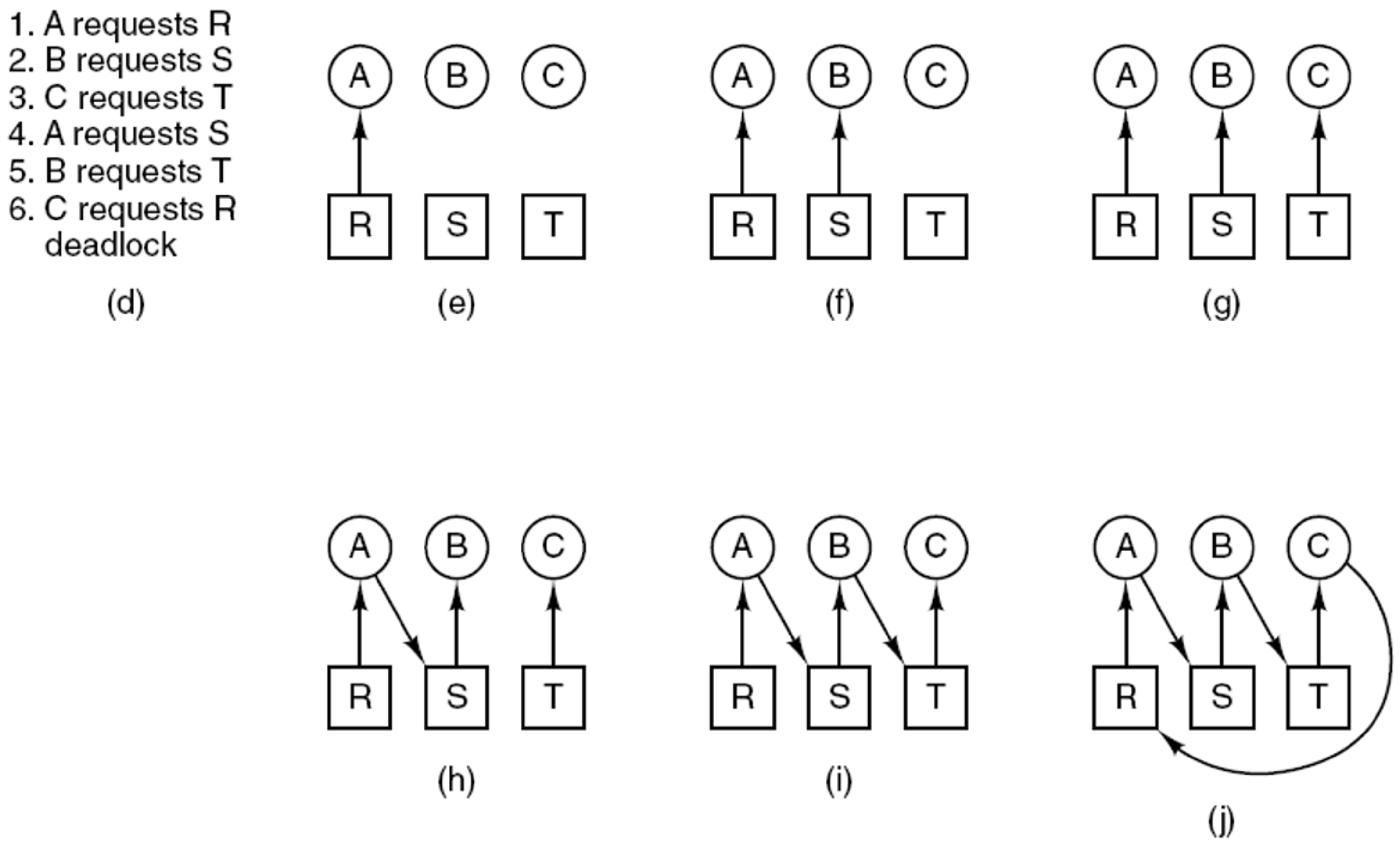
Resource allocation graphs: (a) Holding a resource. (b) Requesting a resource. (c) Deadlock.

Deadlock modeling (2)



An example of how deadlock occurs and how it can be avoided

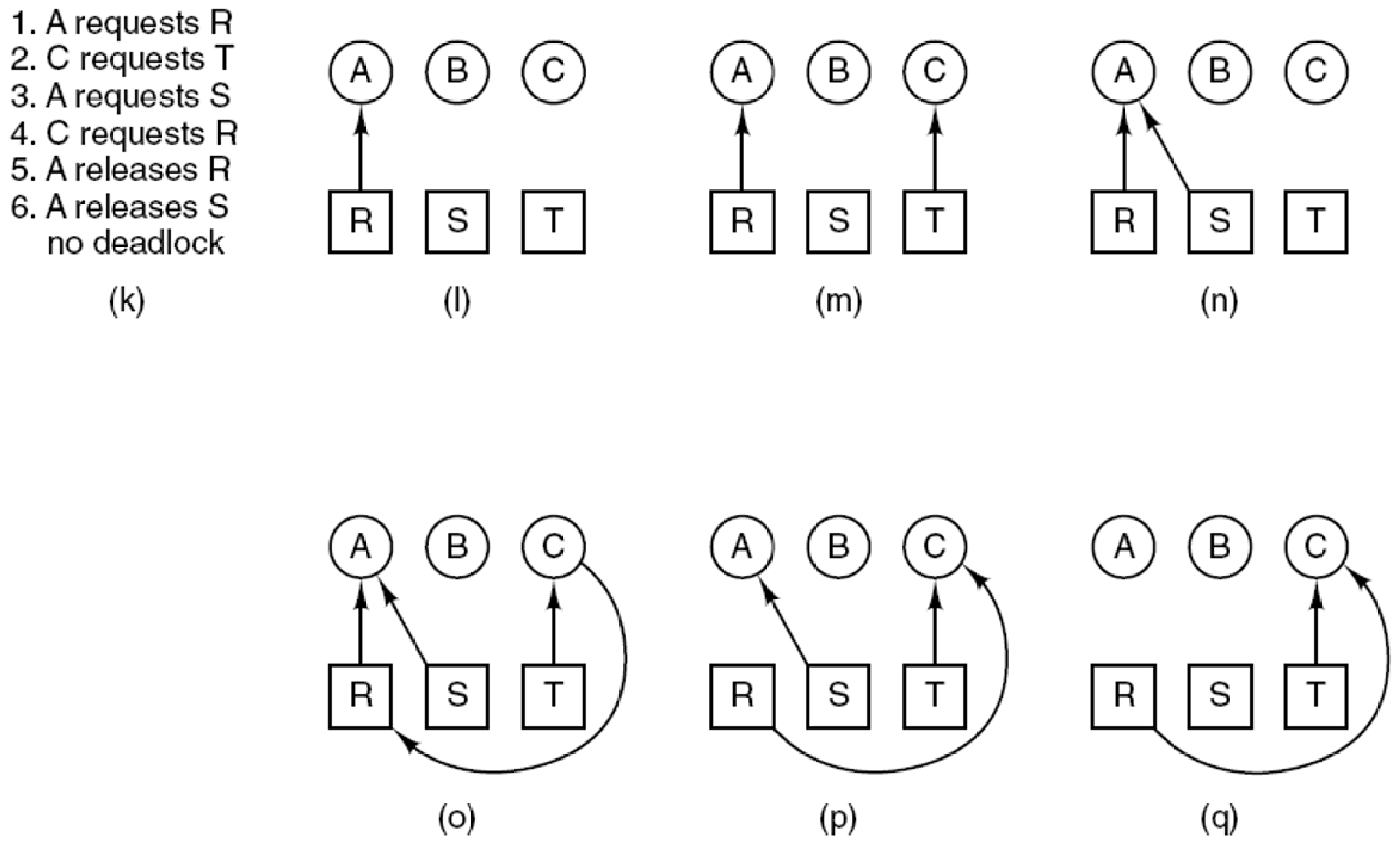
Deadlock modeling (3)



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

An example of how deadlock occurs and how it can be avoided

Deadlock modeling (4)


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

An example of how deadlock occurs and how it can be avoided

Deadlock modeling (5)

Strategies for dealing with deadlocks:

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

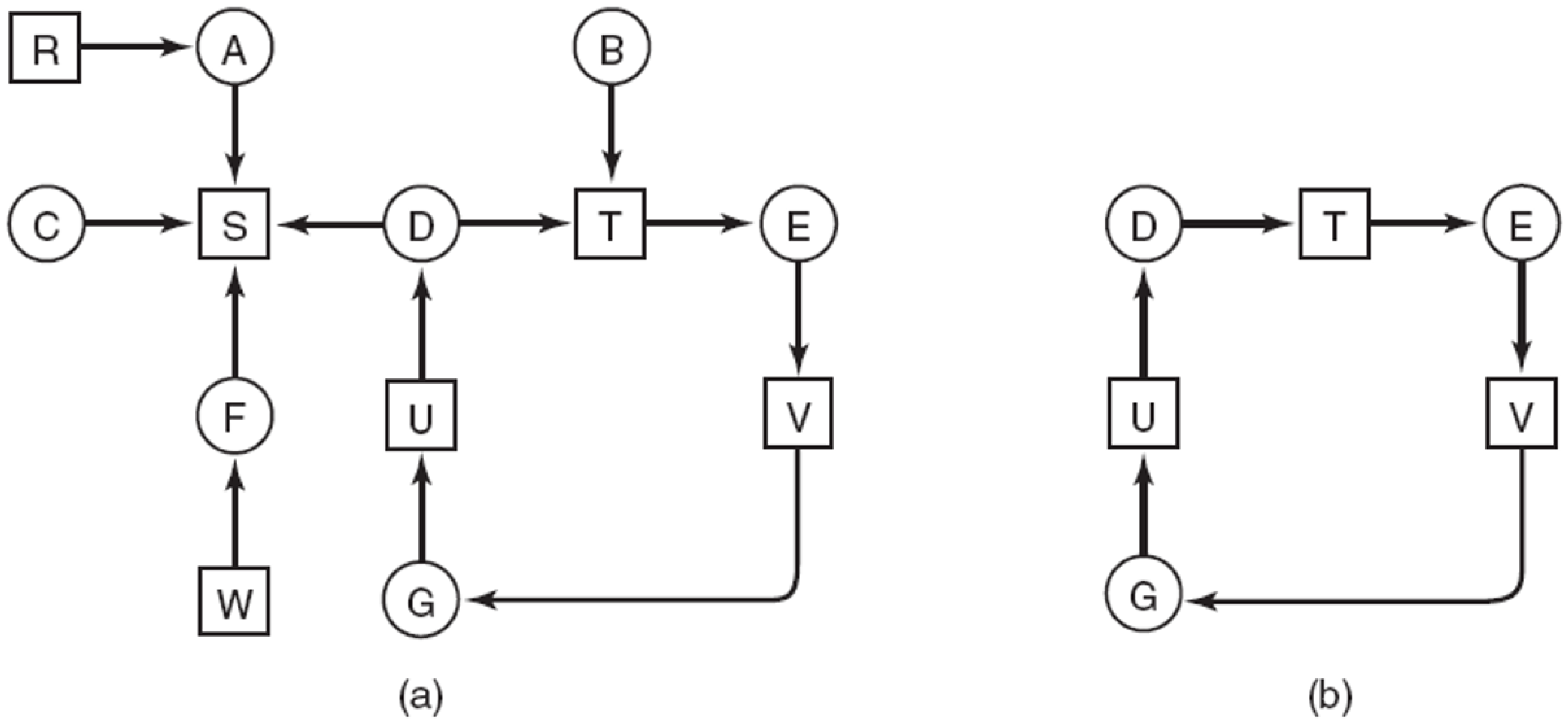
The ostrich algorithm

I.e., just ignore the problem.

Deadlock detection and recovery

- The system does not attempt to prevent deadlocks from occurring.
- It lets them occur
- It tries to detect when this happens
- Then takes some action to recover after the fact

Deadlock detection with one resource of each type (1)



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

Deadlock detection with one resource of each type (2)

Algorithm for detecting deadlock:

```
1. For each node, N in the graph, perform the following five
steps with N as the starting node.

2. Initialize L to the empty list, 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 it does, graph contains a cycle (listed in L),
algorithm terminates.

4. From given node, see if 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. If this is initial node, graph does not contain any cycles,
algorithm terminates. Otherwise, dead end. Remove it, go back
to previous node, make that one current node, go to step 3.
```

Deadlock detection with multiple resources of each type (1)

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

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

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

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

The four data structures needed by the deadlock detection algorithm

Deadlock detection with multiple resources of each type (2)

Deadlock detection algorithm:

1. Look for an unmarked process, Pi , 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.
3. If no such process exists, the algorithm terminates.

Deadlock detection with multiple resources of each type (3)

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

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

Current allocation matrix

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

Request matrix

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

An example for the deadlock detection algorithm

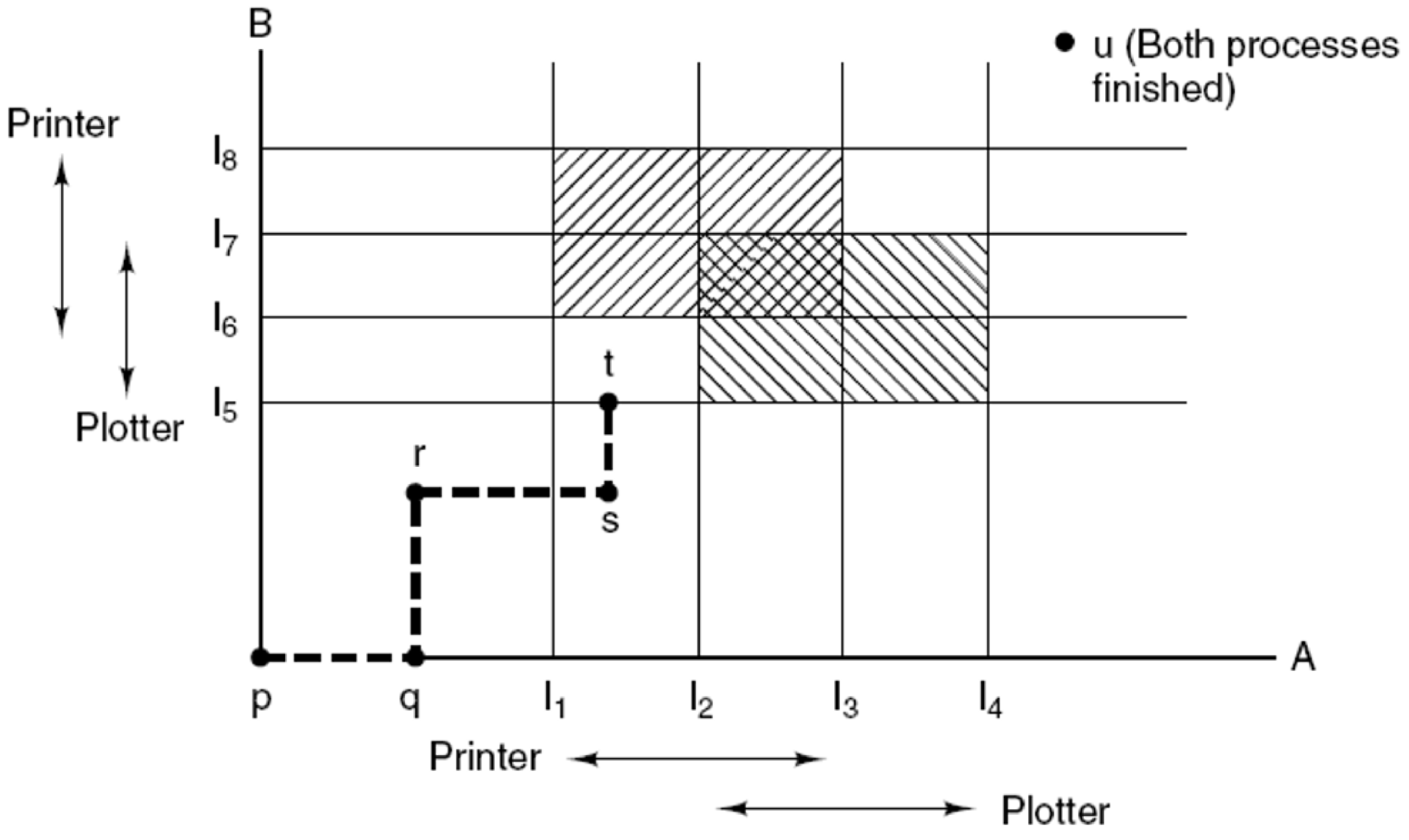
When to check for deadlocks?

Recovery from deadlock

- Recovery through preemption
- Recovery through rollback
- Recovery through killing processes

Which one is the better one?s

Deadlock avoidance



Two process resource trajectories

Safe and unsafe states (1)

	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)

Demonstration that the state in (a) is safe

Safe and unsafe states (2)

	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

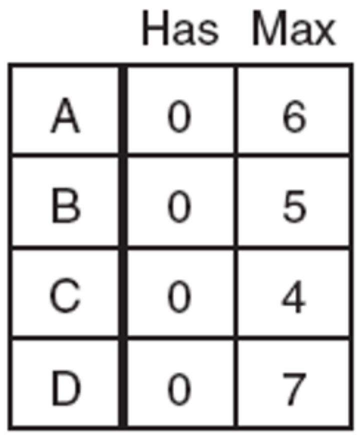
Free: 0
(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4
(d)

Demonstration that the state in (b) is not safe

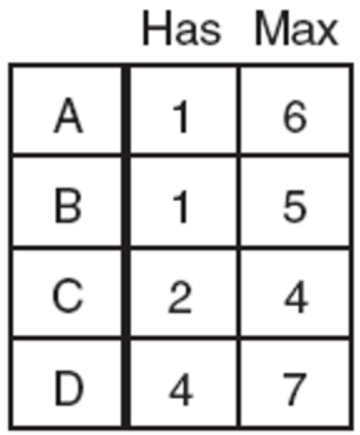
The banker's algorithm for a single resource



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

Free: 10

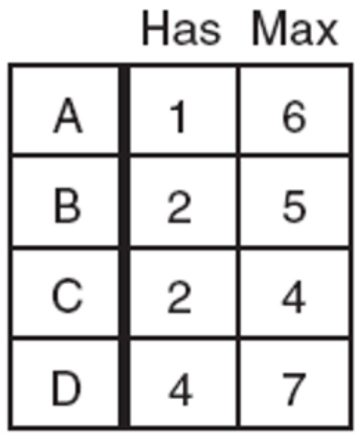
(a)



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

Free: 2

(b)



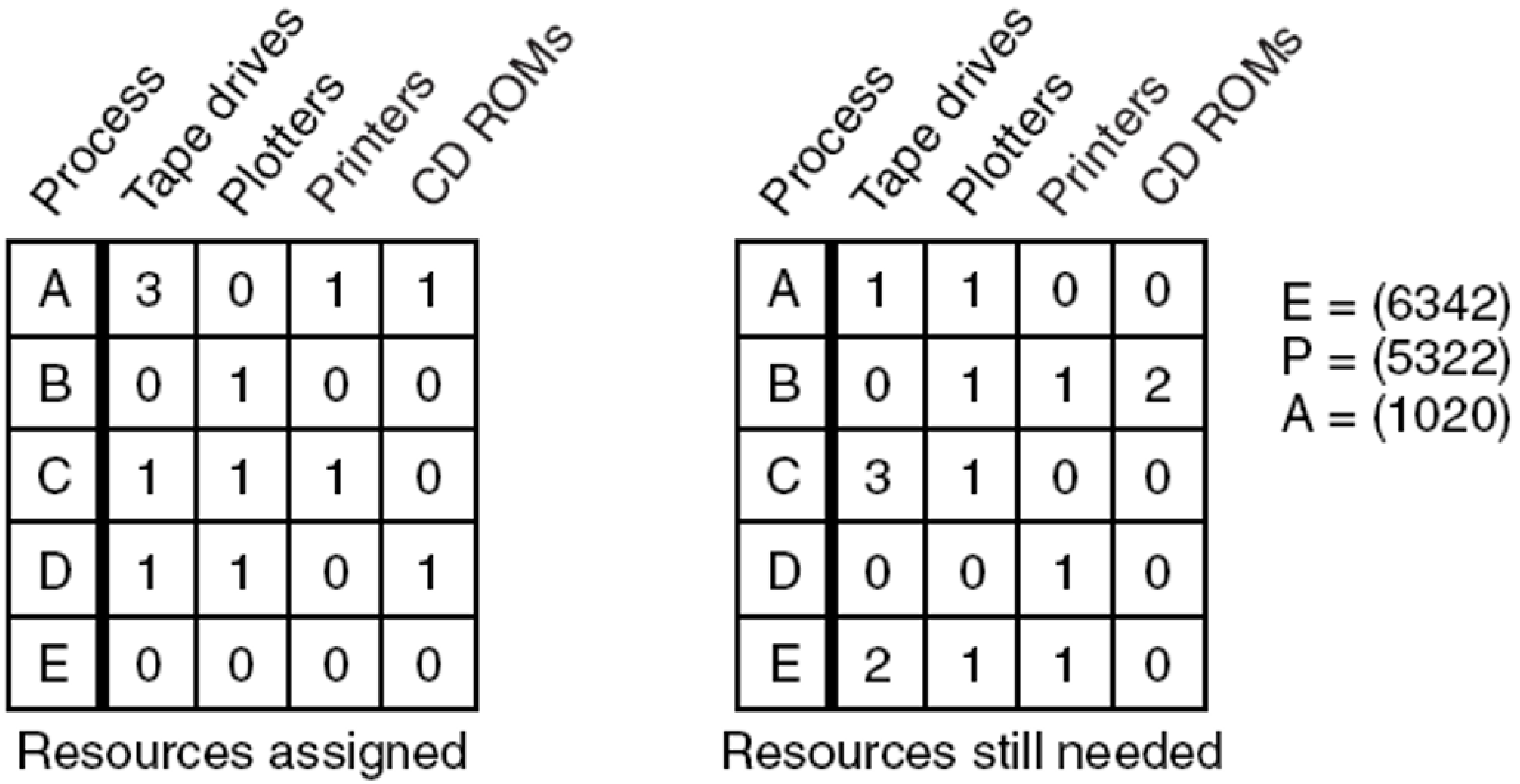
	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)

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

The banker's algorithm multiple resources (1)



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)

The banker’s algorithm with multiple resources

The banker's algorithm multiple resources (2)

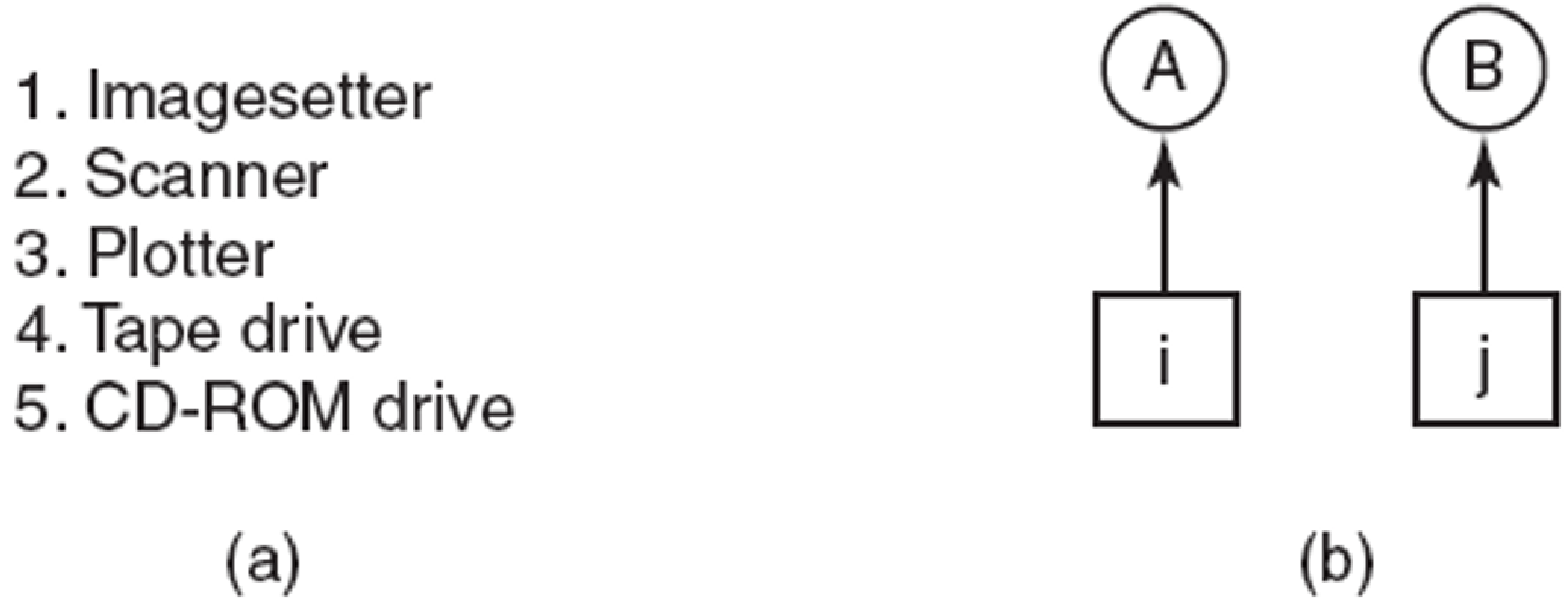
Algorithm for checking to see if a state is safe:

1. Look for row, R, whose unmet resource needs all ≤ A. If no such row exists, system will eventually deadlock since no process can run to completion
2. Assume process of row chosen requests all resources it needs and finishes. Mark process as terminated, add all its resources to the A vector.
3. Repeat steps 1 and 2 until either all processes marked terminated (initial state was safe) or no process left whose resource needs can be met (there is a deadlock).

Deadlock prevention

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

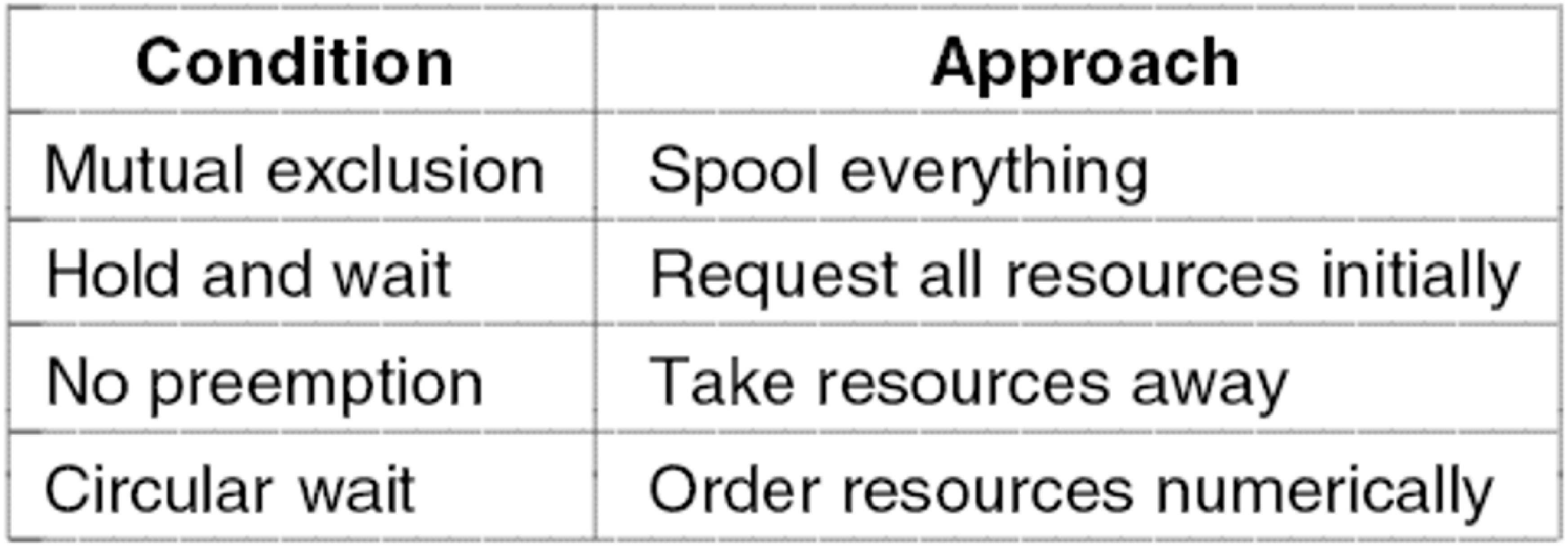
Attacking the circular wait condition



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

(a) Numerically ordered resources. (b) A resource graph.

Approaches to deadlock prevention



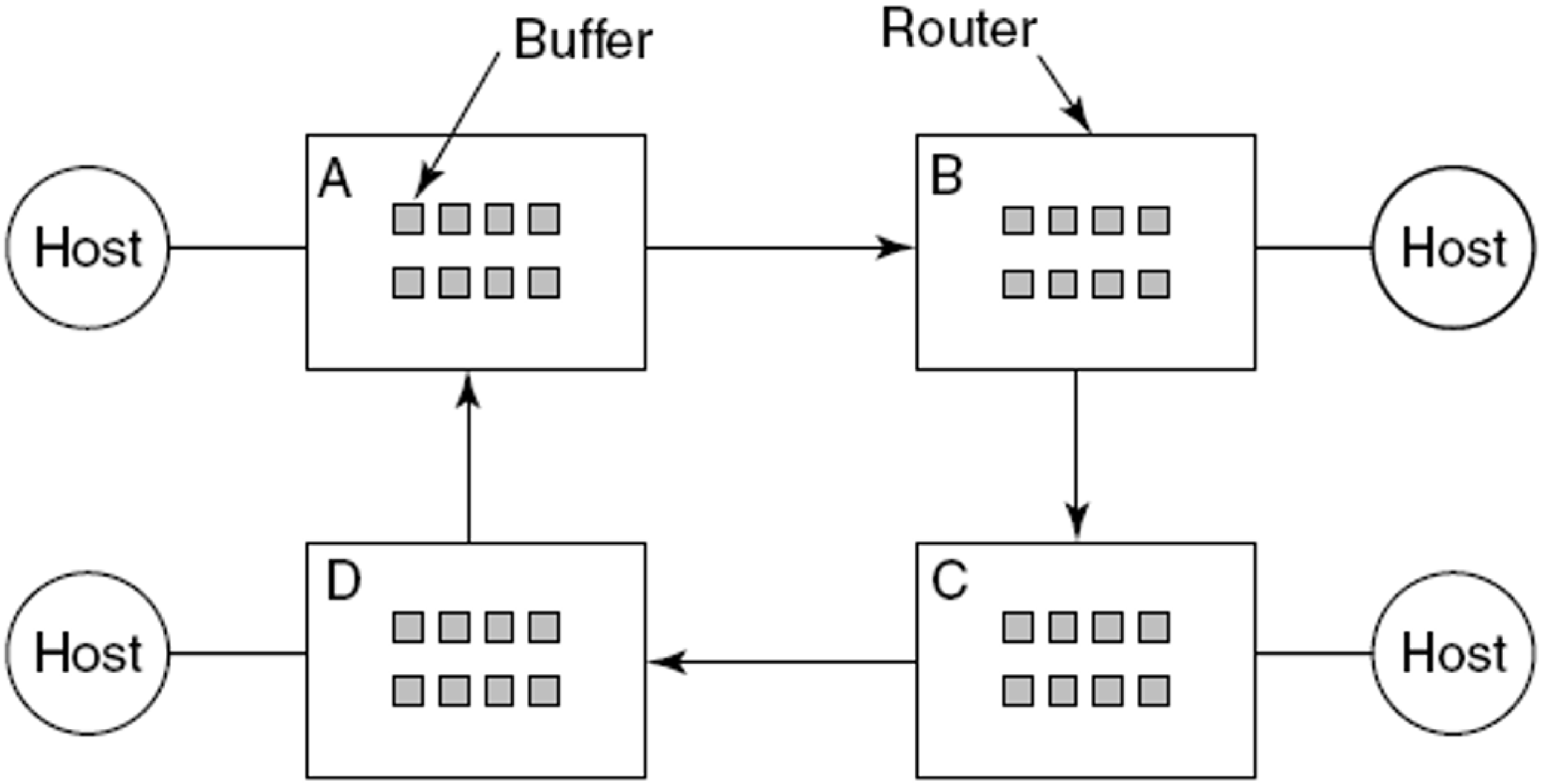
Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

Summary of approaches to deadlock prevention.

Other issues

- Two-phase locking
- Communication deadlocks
- Livelock
- Starvation

Communication deadlocks

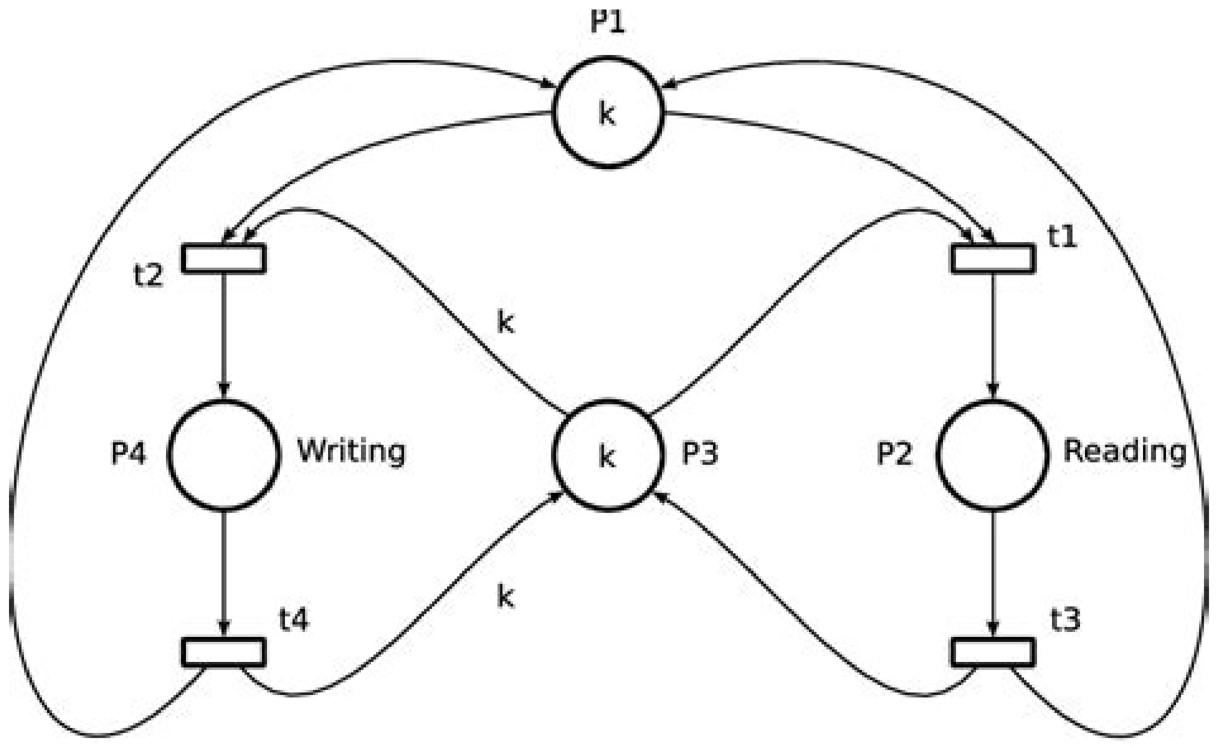


A resource deadlock in a network.

Verifiable concurrency

- Deductive verification
  - Using some logical formalism to prove using theorems that software satisfies its specifications.
- Model checking
  - Use software to automatically check that software satisfies its specifications
- Testing
  - Check executions of software according to some coverage scheme

Petri nets



The readers and writers problem as a petri net