Chapter 3

Deadlocks

sun@hit.edu.cn

Resources

- Examples of computer resources
  - printers
  - tape drives
  - tables
- Processes need access to resources in reasonable order
- Suppose a process holds resource A and requests resource B
  - at same time another process holds B and requests A
  - both are blocked and remain so forever

Resources (1)

- Deadlocks occur when …
  - processes are granted exclusive access to devices, files, and so forth
  - we refer to these objects generally as <u>resources</u>
- Preemptable resources
  - can be taken away from a process with no ill effects
- Nonpreemptable resources
  - will cause the process to fail if taken away

Resources (2)

- Sequence of events required to use a resource
  1. request the resource
  2. use the resource
  3. release the resource

- Must wait if request is denied
  - requesting process may be blocked
  - may fail with error code

Resource Acquisition

```
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)

- **Using a semaphore to protect resources.**

Resource Acquisition

```
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);
}
```

```
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);
}
```

- **Which has a potential deadlock?**

Introduction to Deadlocks

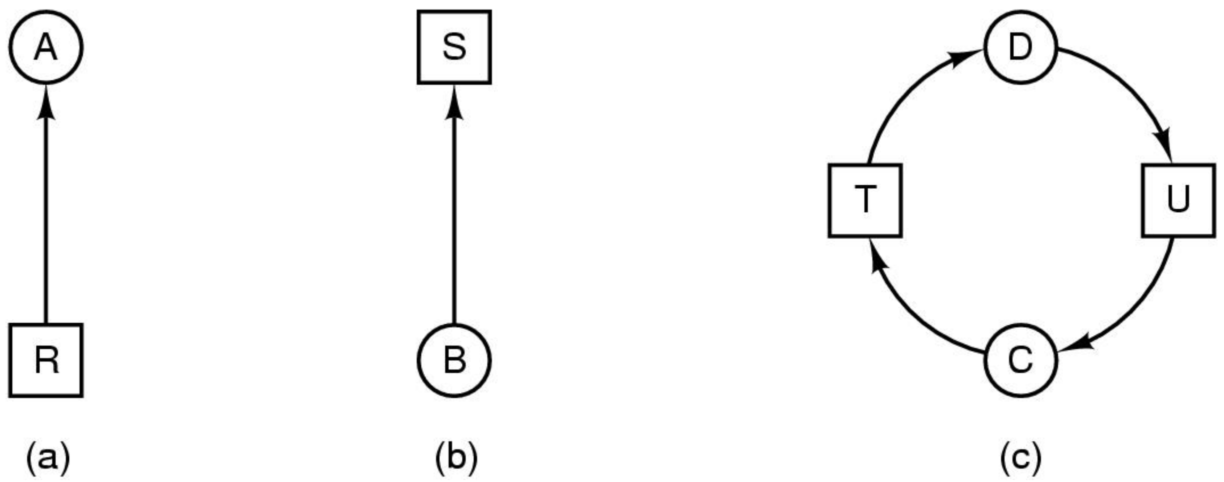
- **Formal definition :**
  A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause
- **Usually the event is release of a currently held resource**
- **None of the processes can …**
  - **run**
  - **release resources**
  - **be awakened**

Four Conditions for Deadlock

1. Mutual exclusion condition
   - each resource assigned to 1 process or is available
2. Hold and wait condition
   - process holding resources can request additional
3. No preemption condition
   - previously granted resources cannot forcibly taken away
4. Circular wait condition
   - must be a circular chain of 2 or more processes
   - each is waiting for resource held by next member of the chain

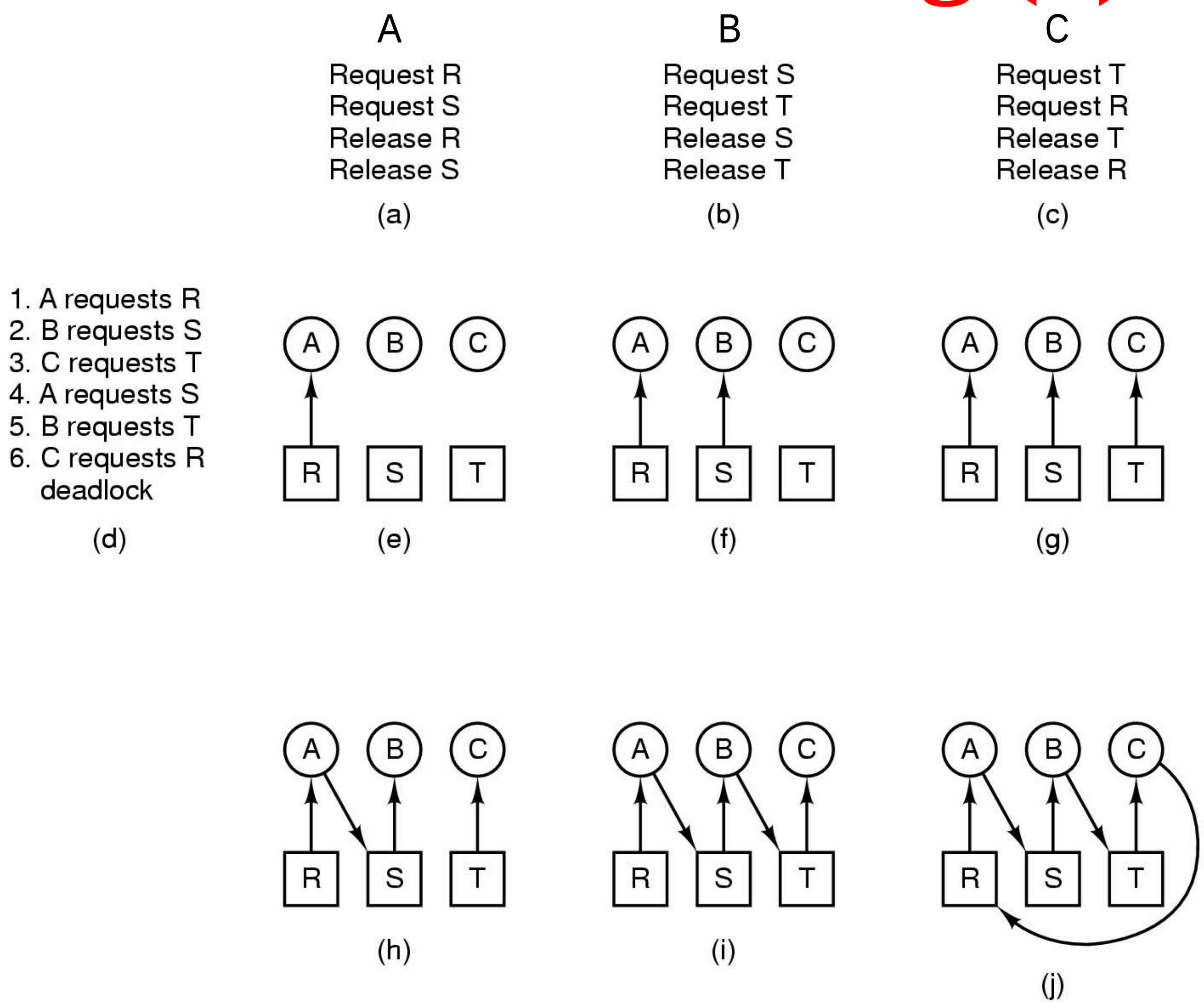
Deadlock Modeling (2)

- Modeled with directed graphs



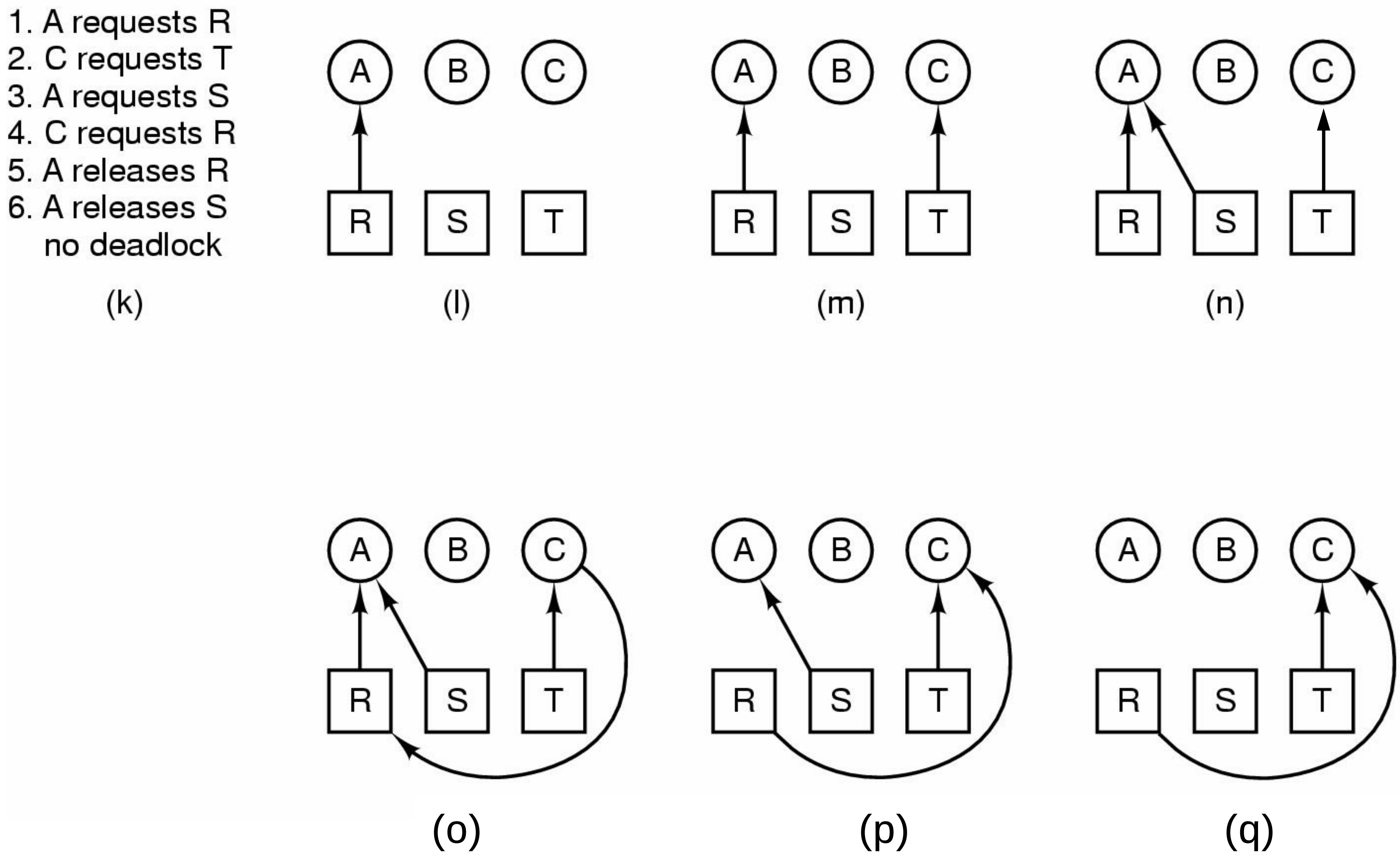
  - resource R assigned to process A
  - process B is requesting/waiting for resource S
  - process C and D are in deadlock over resources T and U

Deadlock Modeling (3)



How deadlock occurs

Deadlock Modeling (4)



How deadlock can be avoided

Deadlock Modeling (5)

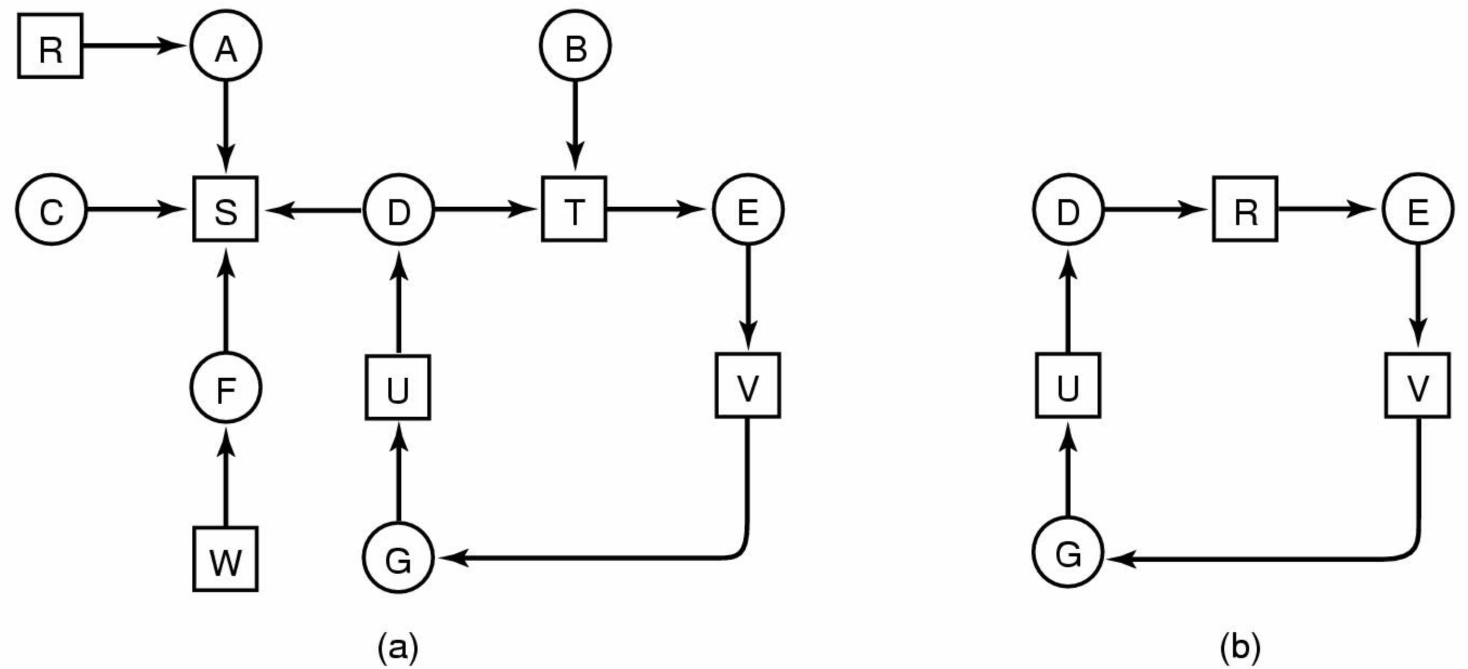
Strategies for dealing with Deadlocks

1. just ignore the problem altogether
2. detection and recovery
3. dynamic avoidance
   - careful resource allocation
4. prevention
   - negating one of the four necessary conditions

The Ostrich Algorithm

- Pretend there is no problem
- Reasonable if
  - deadlocks occur very rarely
  - cost of prevention is high
- UNIX and Windows takes this approach
- It is a trade off between
  - convenience
  - correctness

Detection with One Resource of Each Type (1)



- Note the resource ownership and requests
- A cycle can be found within the graph, denoting deadlock

Detection with One Resource of Each Type (2)

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

Data structures needed by deadlock detection algorithm

sun@hit.edu.cn

Detection with One Resource of Each Type (3)

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

Current allocation matrix

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

Request matrix

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

An example for the deadlock detection algorithm

Detection with One Resource of Each Type (3)

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

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

Current allocation matrix

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

Request matrix

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

An example for the deadlock detection algorithm

Recovery from Deadlock (1)

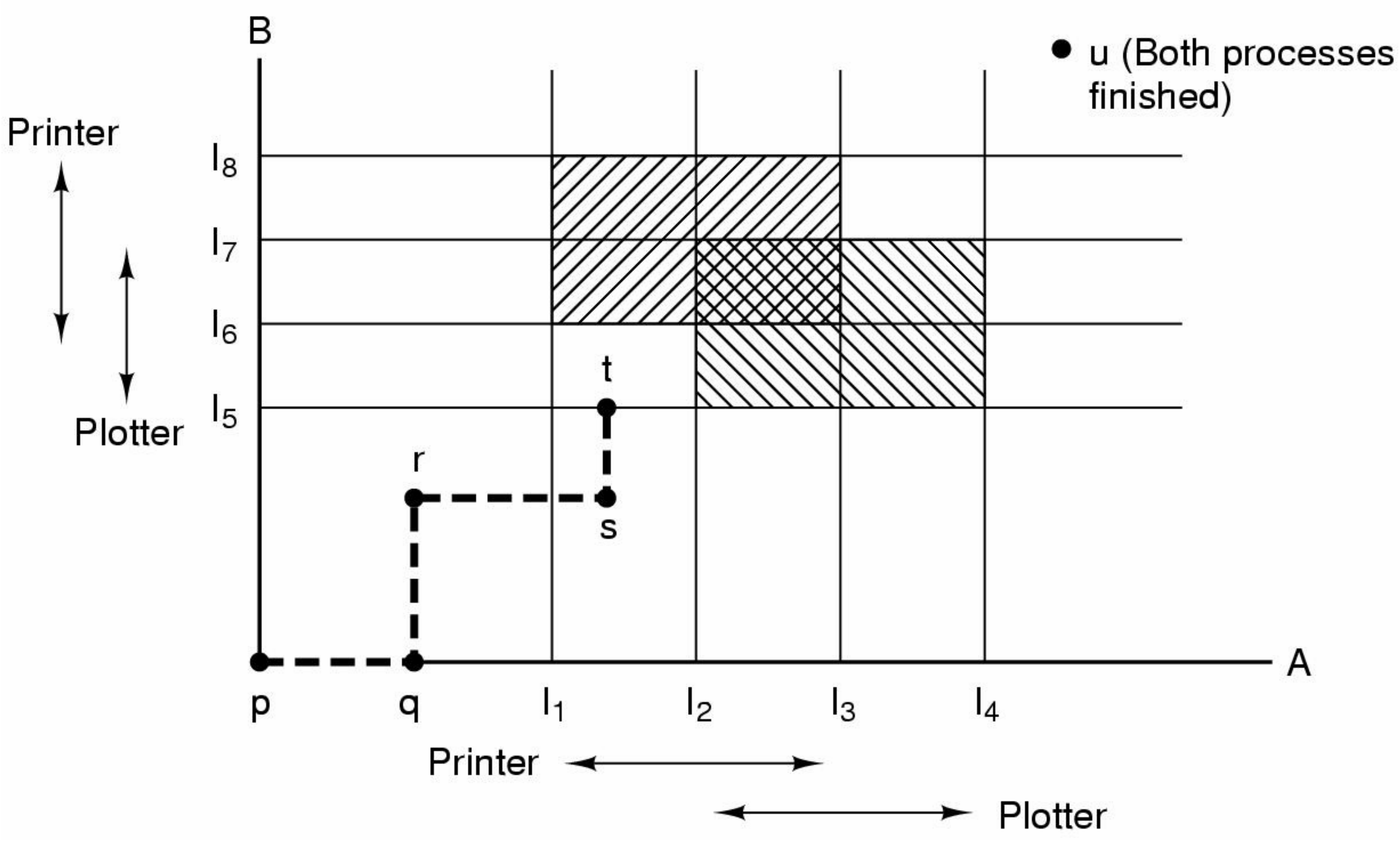
- Recovery through preemption
  - take a resource from some other process
  - depends on nature of the resource
- Recovery through rollback
  - checkpoint a process periodically
  - use this saved state
  - restart the process if it is found deadlocked

Recovery from Deadlock (2)

- Recovery through killing processes
  - crudest but simplest way to break a deadlock
  - kill one of the processes in the deadlock cycle
  - the other processes get its resources
  - choose process that can be rerun from the beginning

Deadlock Avoidance

Resource Trajectories



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 sate 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**
  - **safe**
  - **safe**
  - **unsafe**

Banker's Algorithm for Multiple Resources

Process	Tape drives	Plotters	Scanners	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	Scanners	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)

Example of banker's algorithm with multiple resources

Deadlock Prevention

Attacking the Mutual Exclusion Condition

- Some devices (such as printer) can be spooled
  - only the printer daemon uses printer resource
  - thus deadlock for printer eliminated
- Not all devices can be spooled
- Principle:
  - avoid assigning resource when not absolutely necessary
  - as few processes as possible actually claim the resource

Attacking the Hold and Wait Condition

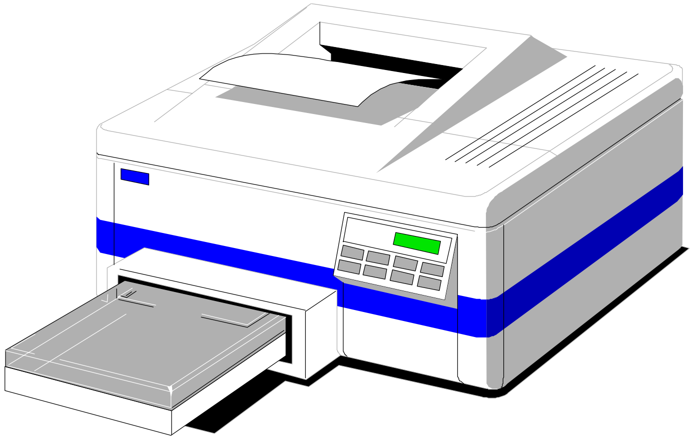
- Require processes to request resources before starting
  - a process never has to wait for what it needs

- Problems
  - may not know required resources at start of run
  - also ties up resources other processes could be using

- Variation:
  - process must give up all resources
  - then request all immediately needed

Attacking the No Preemption Condition

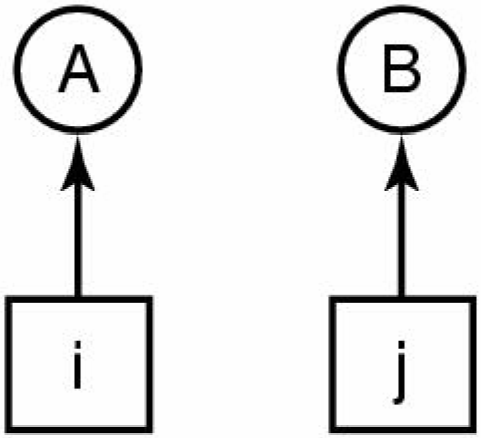
- This is not a viable option
- Consider a process given the printer
  - halfway through its job
  - now forcibly take away printer
  - !!??



Attacking the Circular Wait Condition (1)

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

(a)



(b)

- **Numerically ordered resources**
- **A resource graph**

Summary of 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