MODERN OPERATING SYSTEMS

Third Edition
ANDREW S. TANENBAUM

Chapter 6
Deadlocks

Preemptable and Nonpreemptable Resources

Sequence of events required to use a resource:

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

Resource Acquisition (1)

```
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. Using a semaphore to protect resources. (a) One resource. (b) Two resources.

Resource Acquisition (2)

```
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)

Figure 6-2. (a) Deadlock-free code.

Resource Acquisition (3)

Figure 6-2. (b) Code with a potential deadlock.

```
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)

Introduction To Deadlocks

Deadlock can be defined formally as follows:

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)

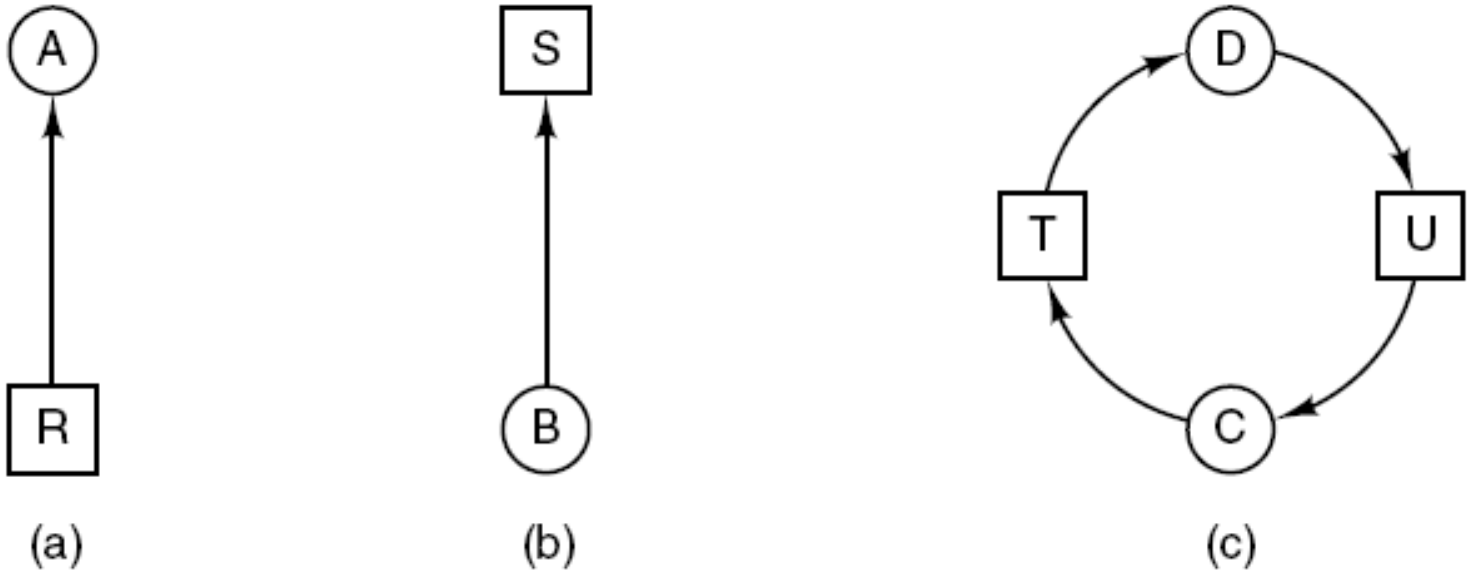


Figure 6-3. Resource allocation graphs. (a) Holding a resource. (b) Requesting a resource. (c) Deadlock.

Deadlock Modeling (2)

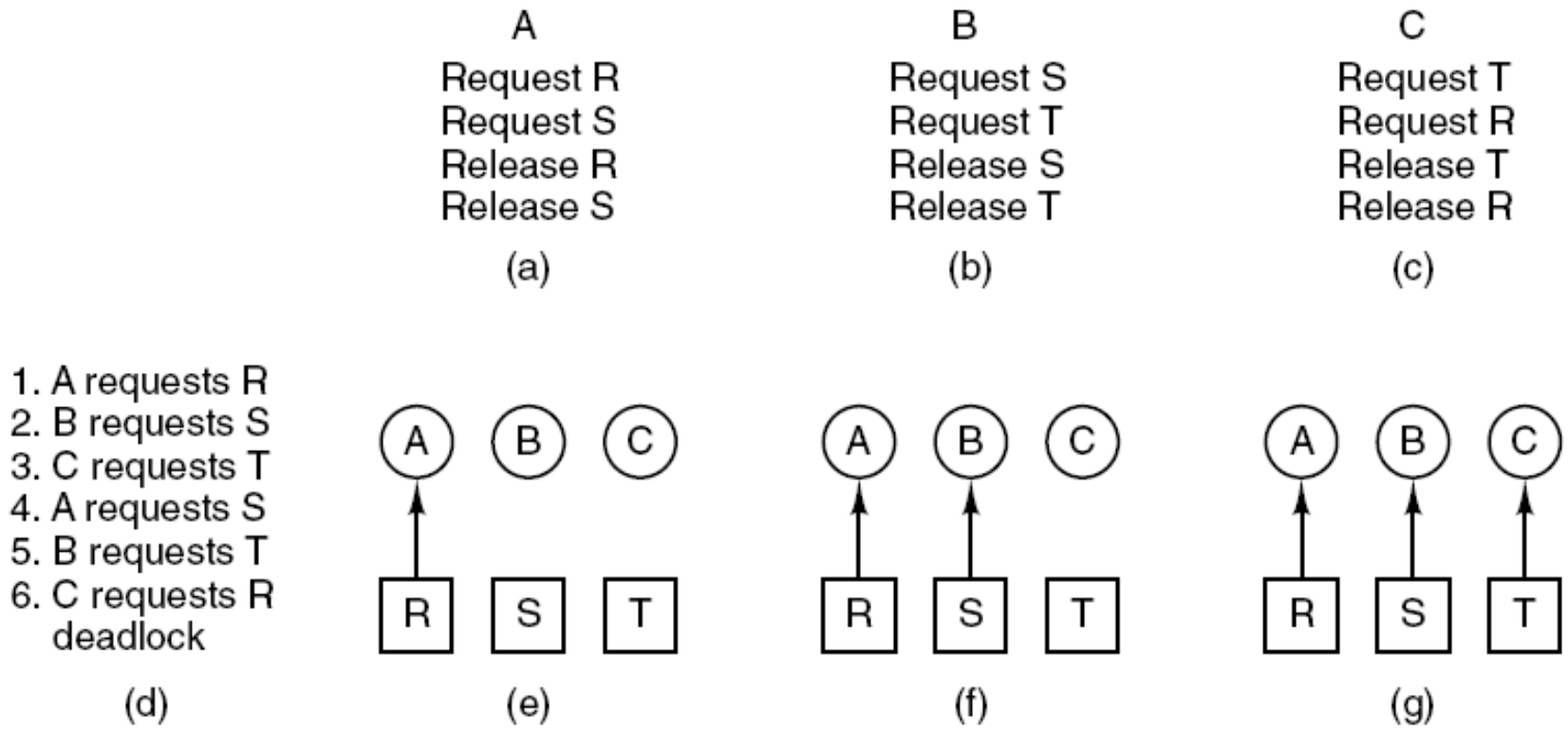


Figure 6-4. An example of how deadlock occurs and how it can be avoided.

Deadlock Modeling (3)

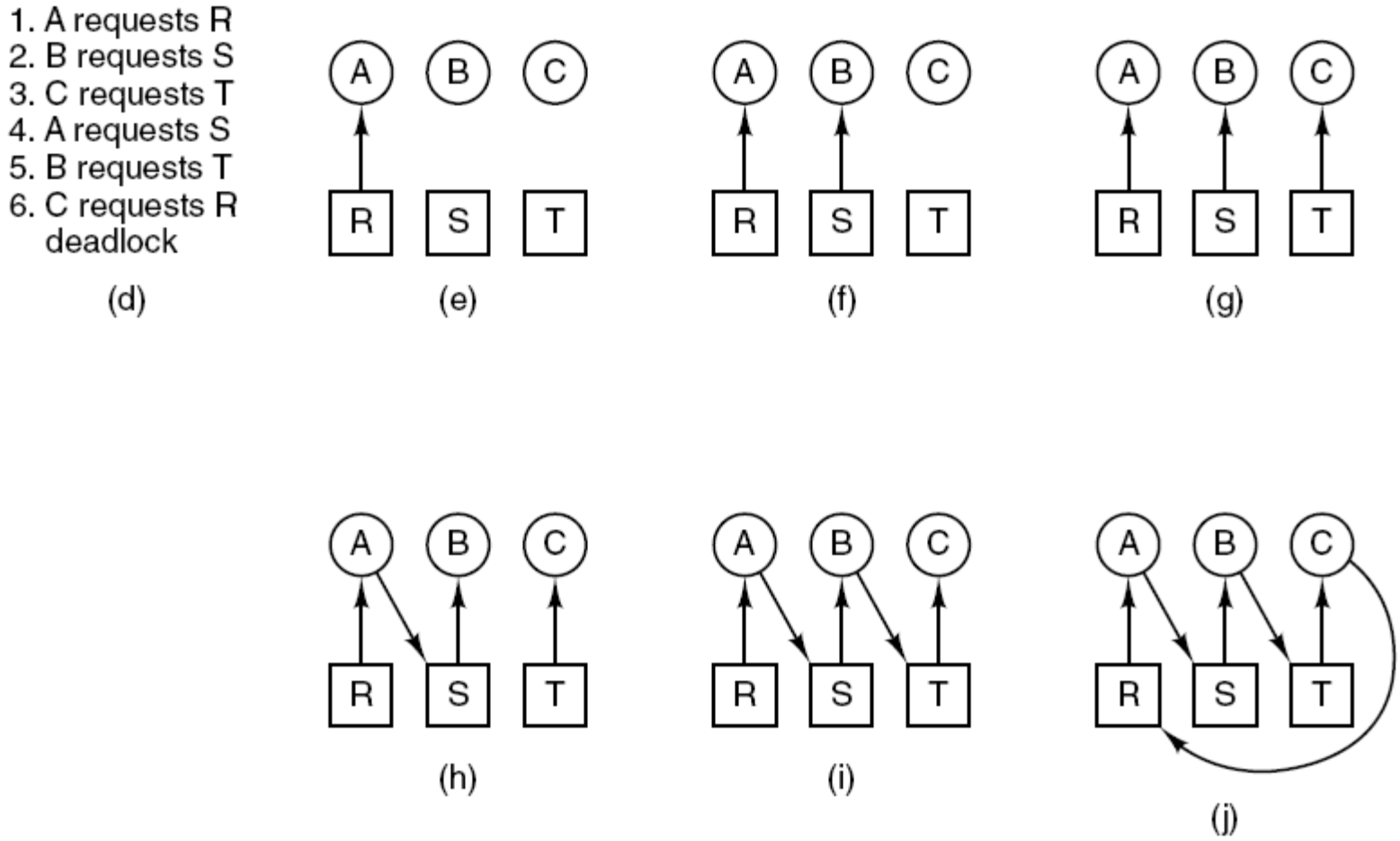


Figure 6-4. An example of how deadlock occurs and how it can be avoided.

Deadlock Modeling (4)

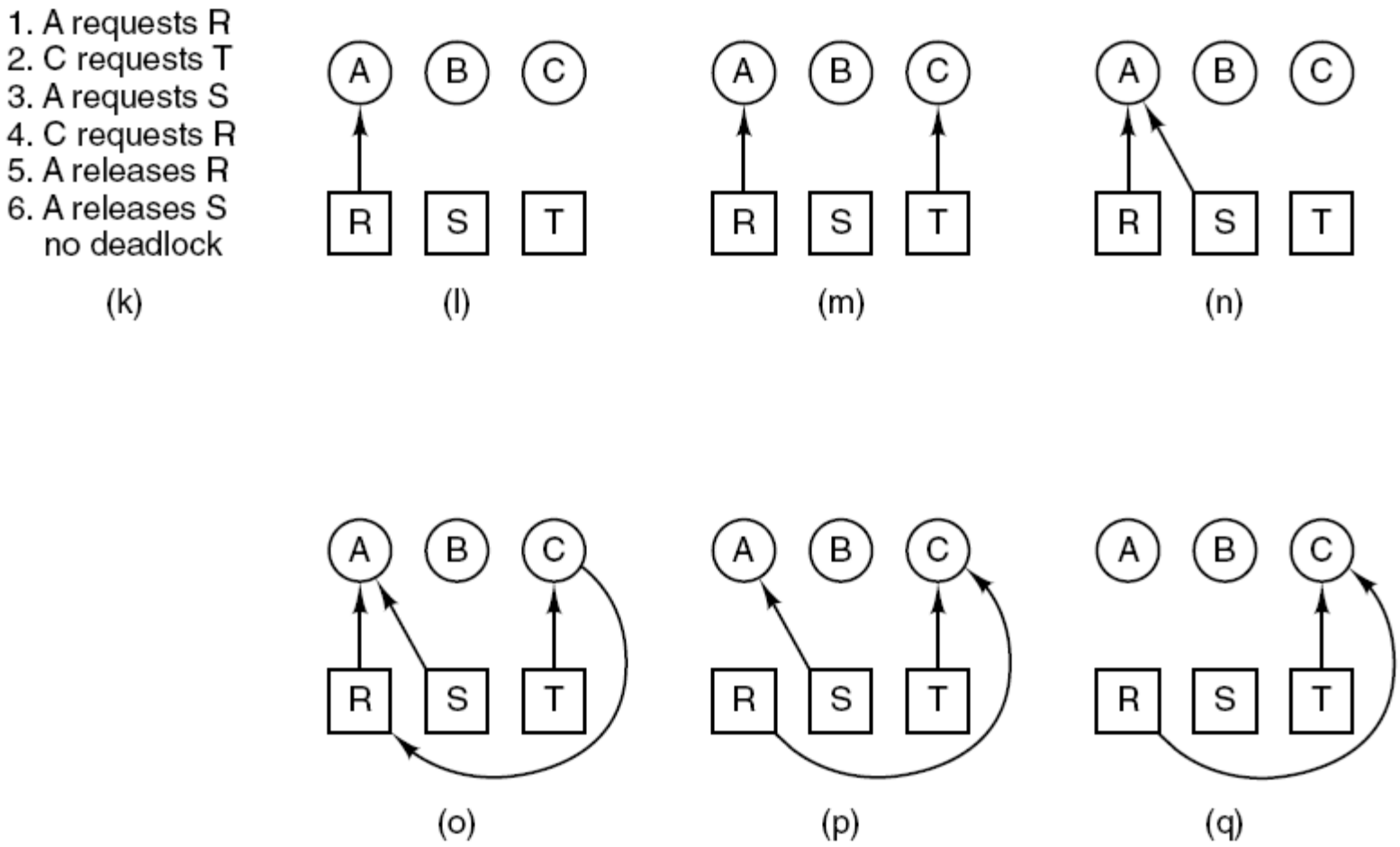


Figure 6-4. An example of how deadlock occurs and how it can be avoided.

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.

Deadlock Detection with One Resource of Each Type (1)

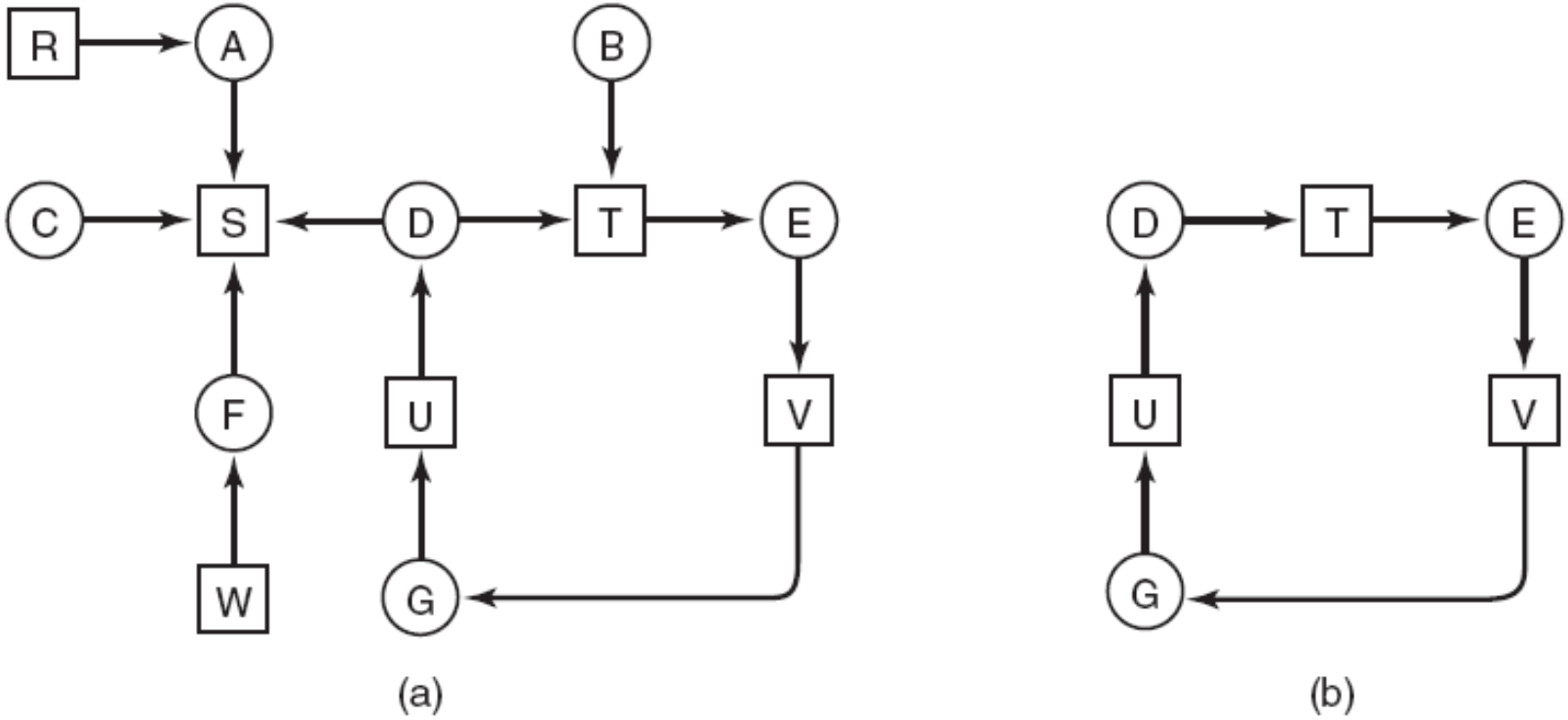


Figure 6-5. (a) A resource graph. (b) A cycle extracted from (a).

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.

…

Deadlock Detection with One Resource of Each Type (3)

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)

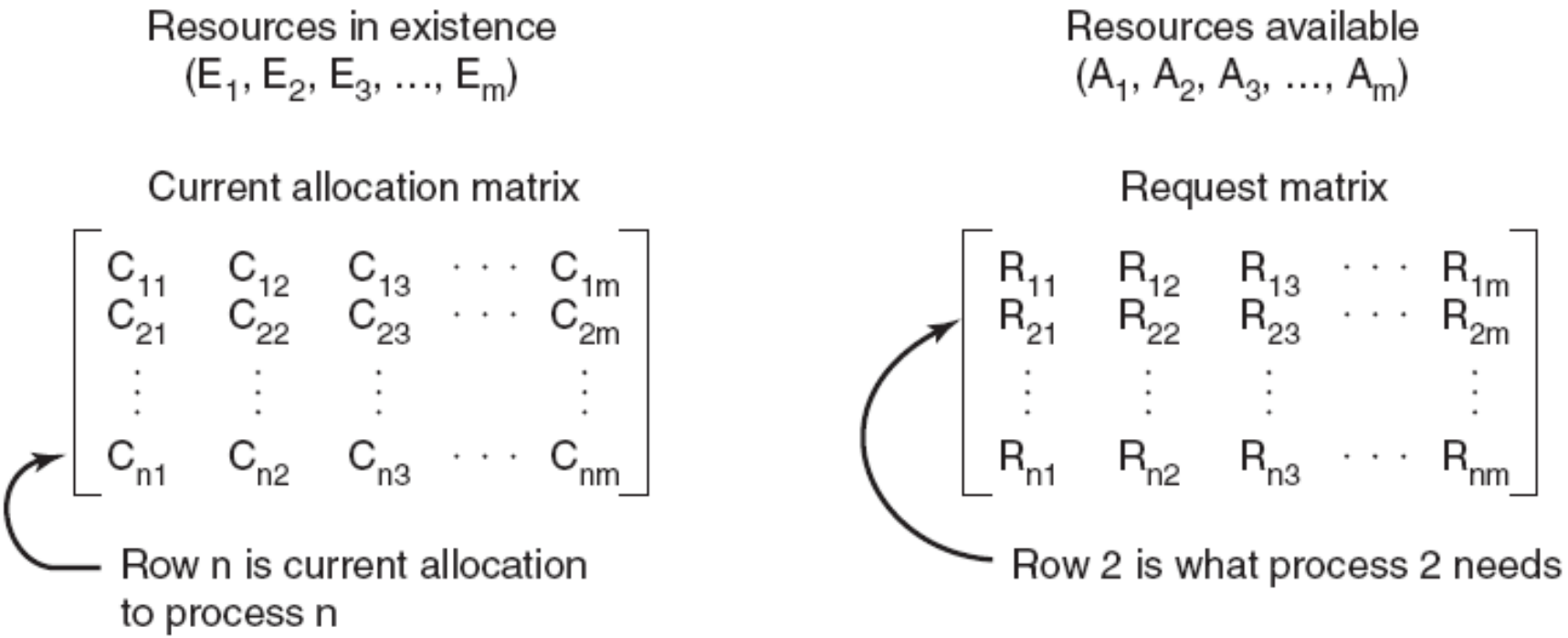


Figure 6-6. The four data structures needed by the deadlock detection algorithm.

Deadlock Detection with Multiple Resources of Each Type (2)

Deadlock detection algorithm:

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

Deadlock Detection with Multiple Resources of Each Type (3)

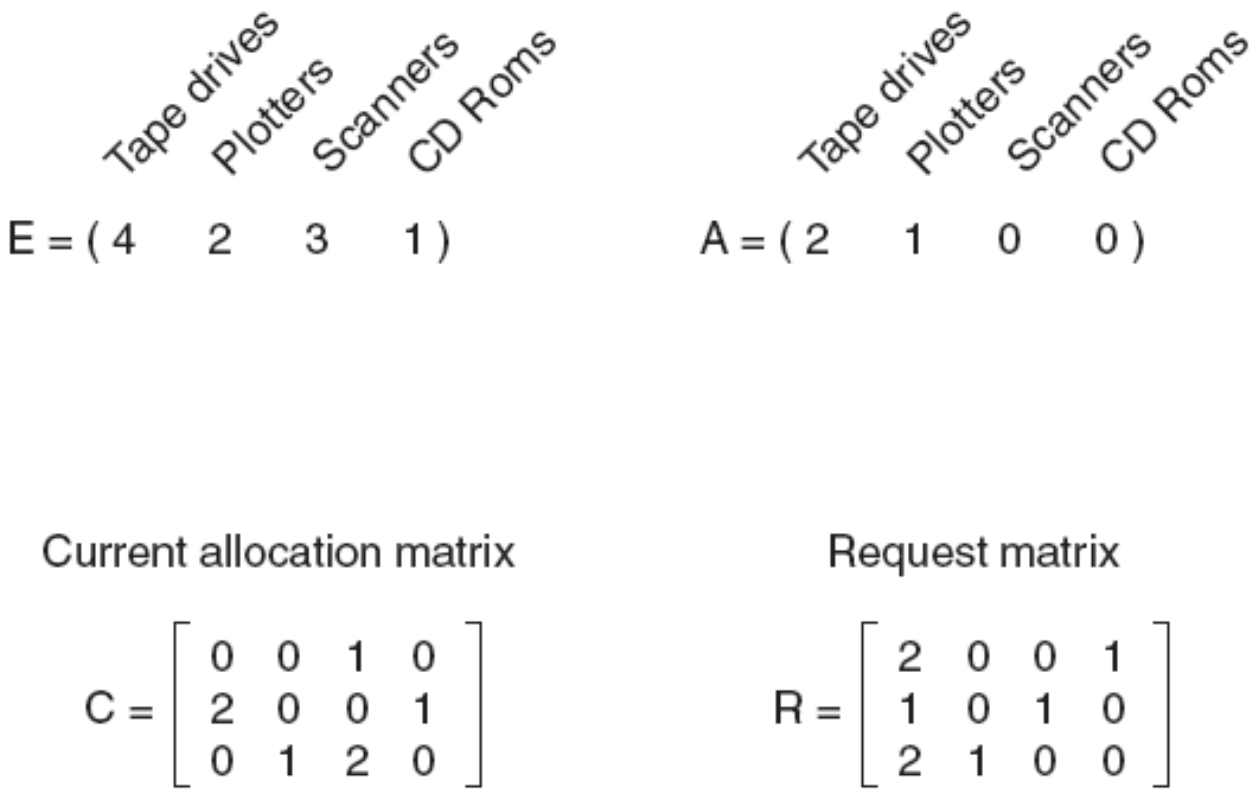


Figure 6-7. An example for the deadlock detection algorithm.

Recovery from Deadlock

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

Deadlock Avoidance

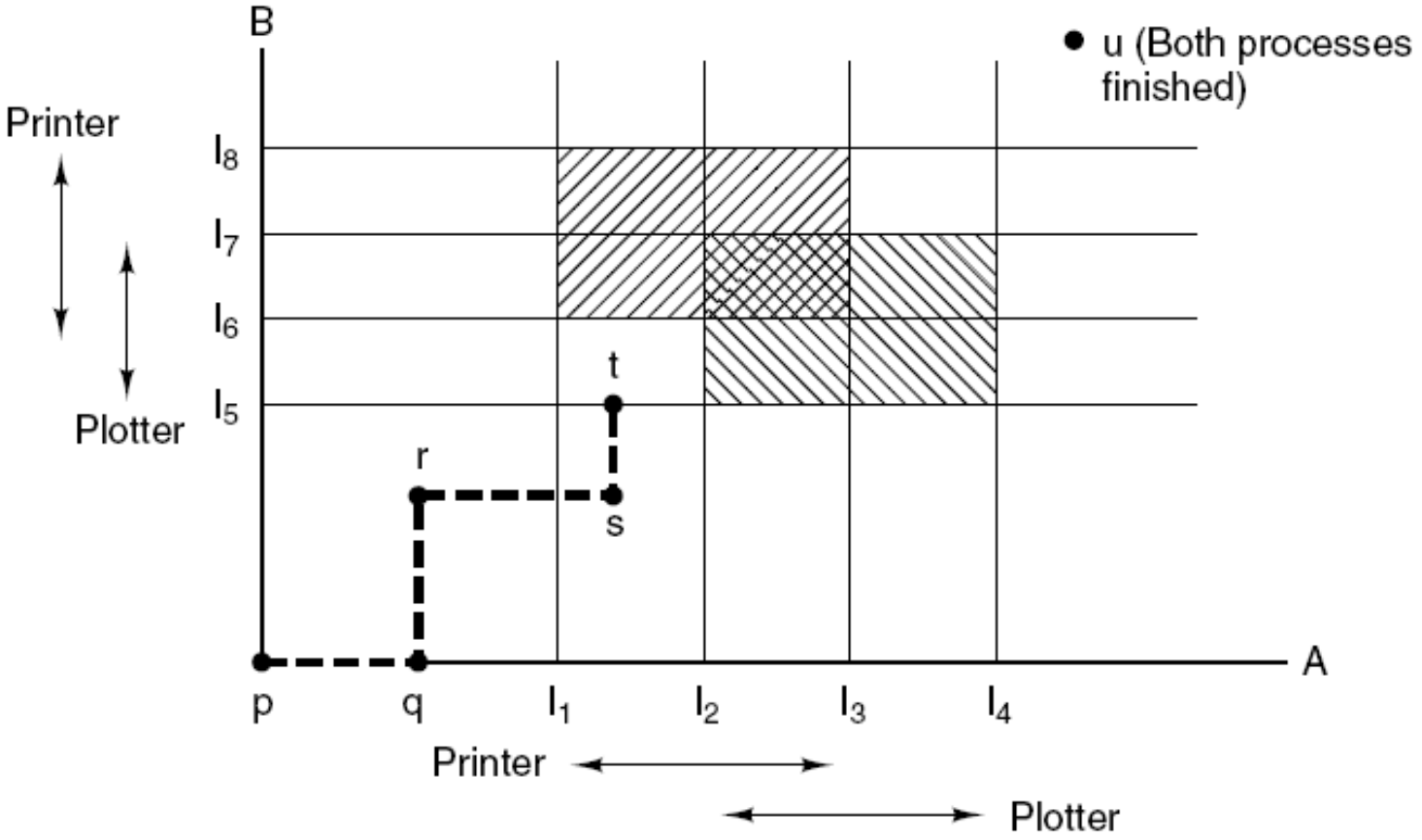


Figure 6-8. Two process resource trajectories.

Safe and Unsafe States (1)

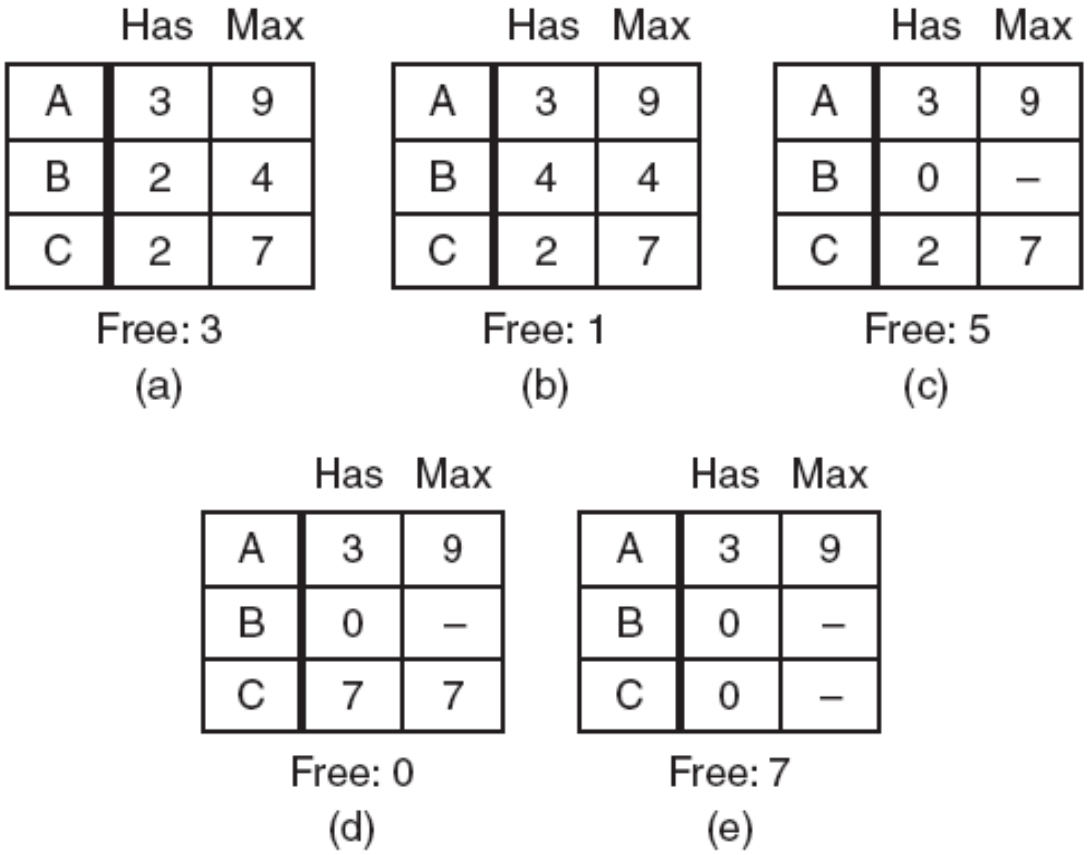


Figure 6-9. Demonstration that the state in (a) is safe.

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)

Figure 6-10. 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)

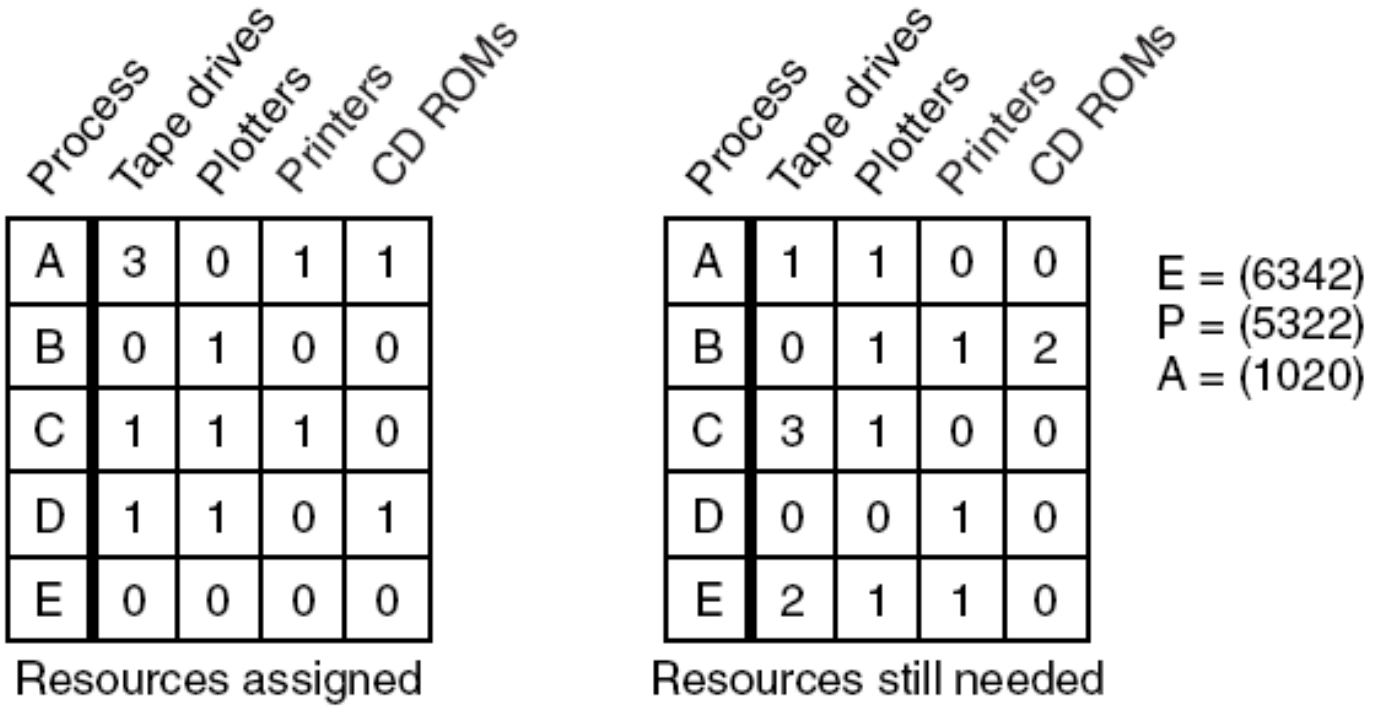
	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)

Figure 6-11. Three resource allocation states: (a) Safe. (b) Safe. (c) Unsafe.

The Banker's Algorithm for 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)

Figure 6-12. The banker's algorithm with multiple resources.

The Banker's Algorithm for 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).

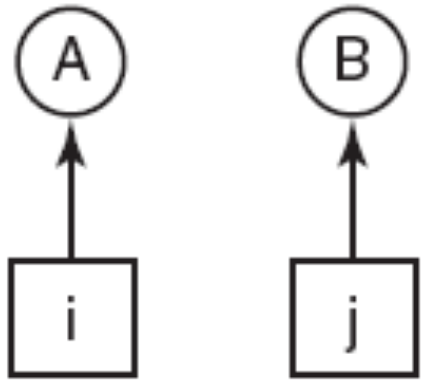
Deadlock Prevention

- 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)



(b)

Figure 6-13. (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

Figure 6-14. Summary of approaches to deadlock prevention.

Other Issues

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

Communication Deadlocks

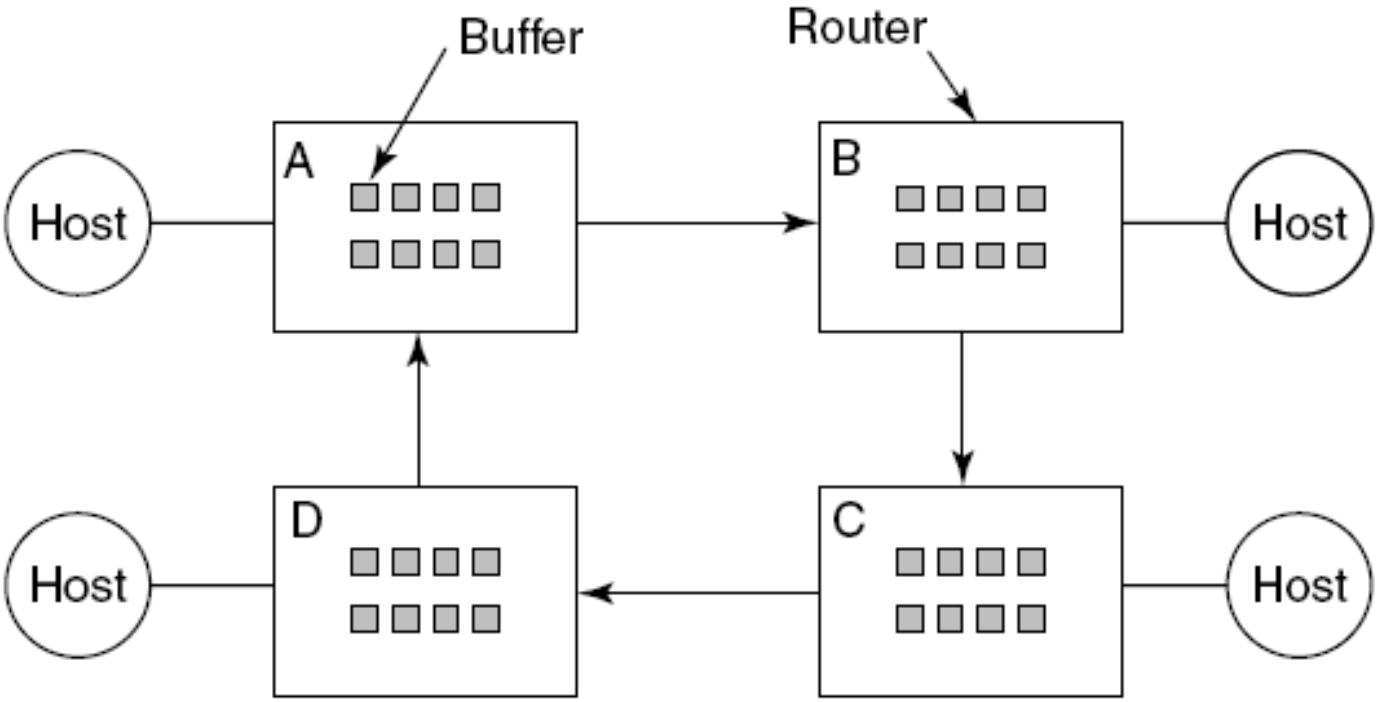


Figure 6-15. A resource deadlock in a network.

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);
}
```

Figure 6-16. Busy waiting that can lead to livelock.