Chapter 5

Process Management

Understanding Operating Systems, Fourth Edition

Objectives

You will be able to describe:

- Several causes of system deadlock
- The difference between preventing and avoiding deadlocks
- How to detect and recover from deadlocks
- The concept of process starvation and how to detect and recover from it

Objectives (continued)

You will be able to describe:

- The concept of a race, and how to prevent it
- The difference between deadlock, starvation, and race

Overview

- A lack of process synchronization results into deadlock or starvation
  - **Deadlock:**
    - A system-wide tangle of resource requests that begins when two or more jobs are put on hold
    - Each job waiting for a vital resource to become available
    - The jobs come to a standstill
    - Resolved via external intervention
  - **Starvation:** Infinite postponement of a job

Deadlock

- Affects more than one job, hence more serious than starvation
- System (not just a few programs) is affected as resources are being tied up
  - e.g., Traffic jam
- More prevalent in interactive systems
- Deadlocks quickly become critical situations in real-time systems
- No simple and immediate solution to a deadlock

Deadlock (continued)

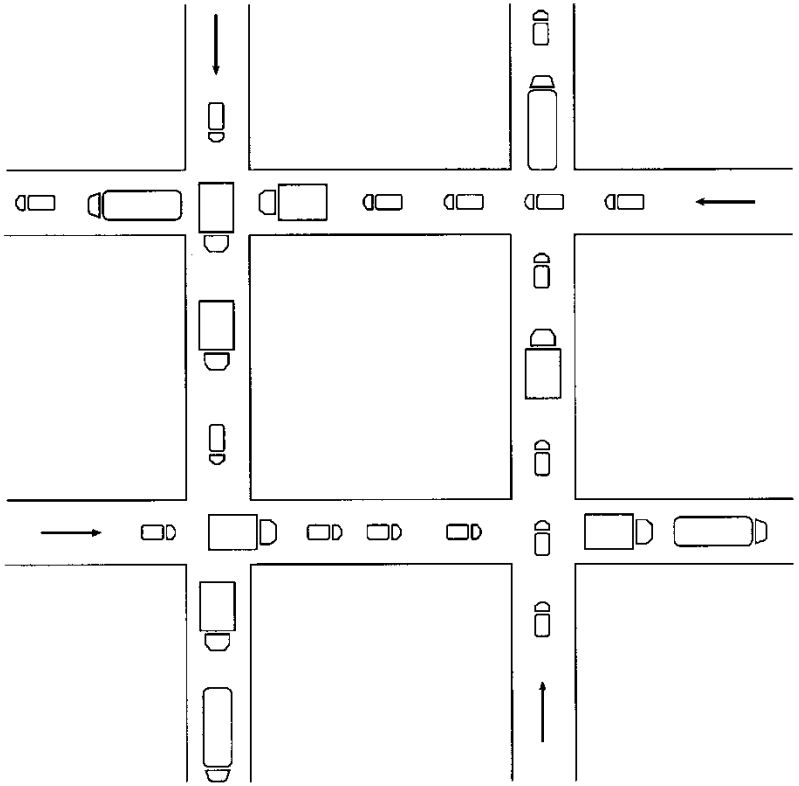


Figure 5.1: A classic case of traffic deadlock

Seven Cases of Deadlock

- **Different cases of Deadlock:**
  - Deadlocks on file requests
  - Deadlocks in databases
  - Deadlocks in dedicated device
  - Deadlocks in multiple device
  - Deadlocks in spooling
  - Deadlocks in disk sharing
  - Deadlocks in a network

Case 1: Deadlocks on File Requests

- Occurs if jobs are allowed to request and hold files for the duration of their execution
- **Example:** Refer to Figure 5.2 (next slide)
  - P1 has access to F1 but requires F2 also
  - P2 has access to F2 but requires F1 also
  - Deadlock remains until a program is withdrawn or forcibly removed and its file is released
  - Any other programs that require F1 or F2 are put on hold as long as this situation continues

Case 1: Deadlocks on File Requests (continued)

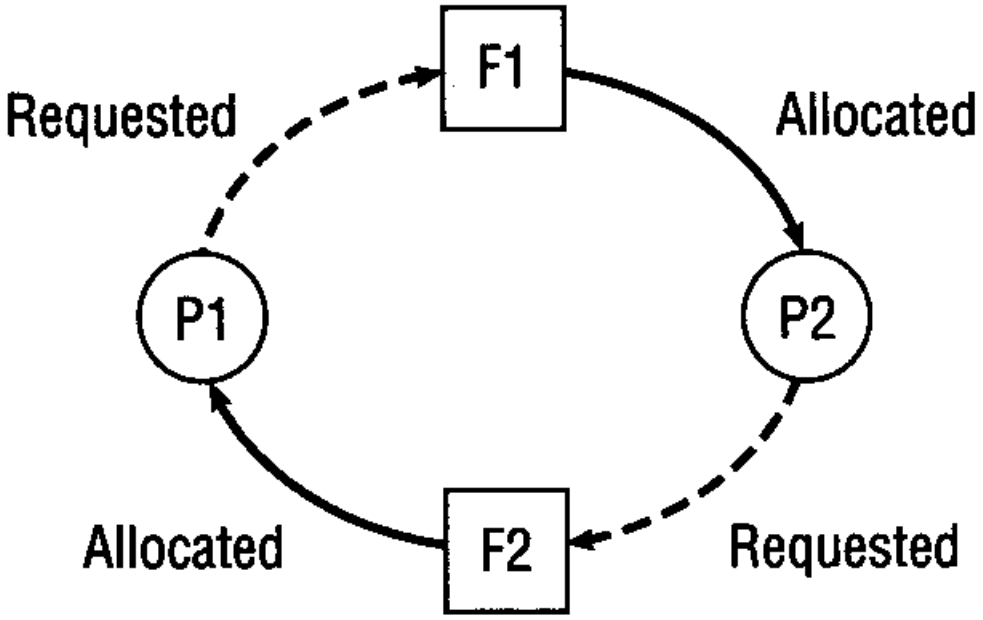


Figure 5.2: A case of deadlock on file requests

Case 2: Deadlocks in Databases

- Occurs if two processes access and lock records in a database
- **Example:** Two processes (P1 and P2), each of which needs to update two records (R1 and R2)
  - Following sequence leads to a deadlock:
    - P1 accesses R1 and locks it
    - P2 accesses R2 and locks it
    - P1 requests R2, which is locked by P2
    - P2 requests R1, which is locked by P1

Case 2: Deadlocks in Databases (continued)

- **Locking:** A technique through which the user locks out all other users while working with the database. Can be done at three different levels:
  - Entire database for the duration of the request
  - A subsection of the database
  - Individual record until the request is completed
- **Race between processes:** A condition resulting when locking is not used
  - Causes incorrect final version of data
  - Depends on order in which each process executes it

Case 2: Deadlocks in Databases (continued)

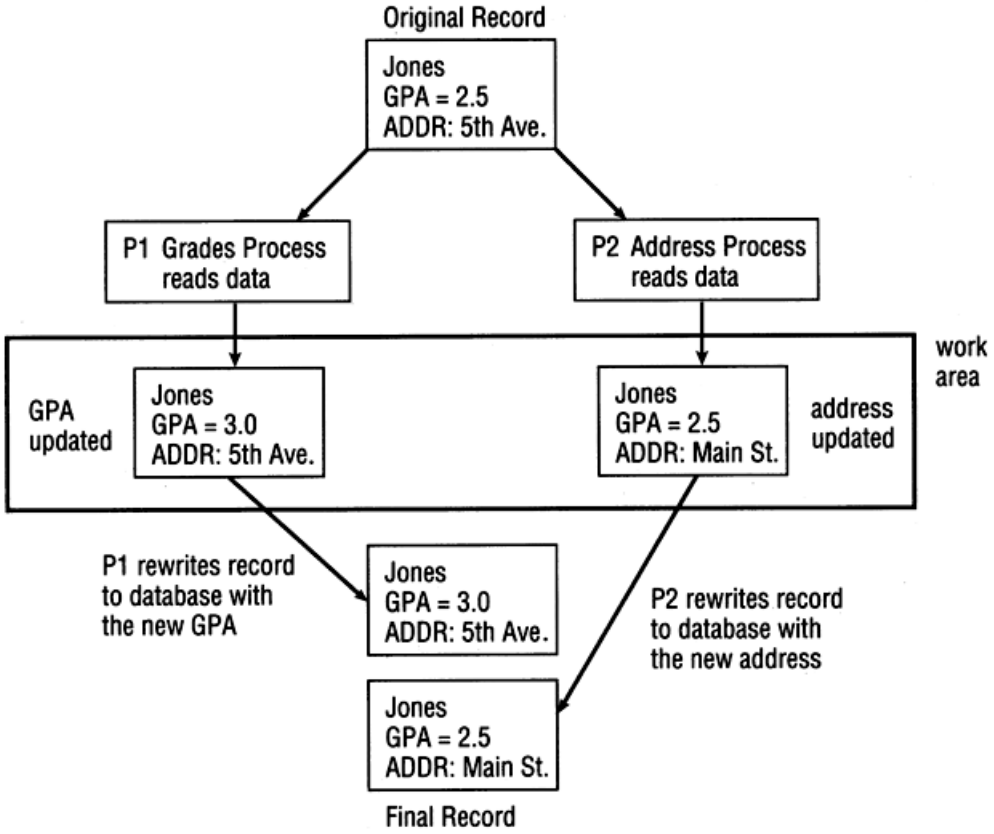


Figure 5.3: A case of “race between processes”

Case 3: Deadlocks in Dedicated Device Allocation

- Occurs when there is a limited number of dedicated devices
- **Example:** Each program (P1 and P2) needs two tape drives (only available). Soon the following sequence transpires:
  - P1 requests tape drive 1 and gets it
  - P2 requests tape drive 2 and gets it
  - P1 requests tape drive 2 but is blocked
  - P2 requests tape drive 1 but is blocked

Case 4: Deadlocks in Multiple Device Allocation

- Occurs when several processes request, and hold on to, dedicated devices while other processes act in a similar manner
- **Example:** Three programs and three dedicated devices: tape drive, printer, and plotter
    - P1 requests and gets the tape drive
    - P2 requests and gets the printer
    - P3 requests and gets the plotter
    - P1 requests the printer but is blocked
    - P2 requests the plotter but is blocked
    - P3 requests the tape drive but is blocked

Case 4: Deadlocks in Multiple Device Allocation (continued)

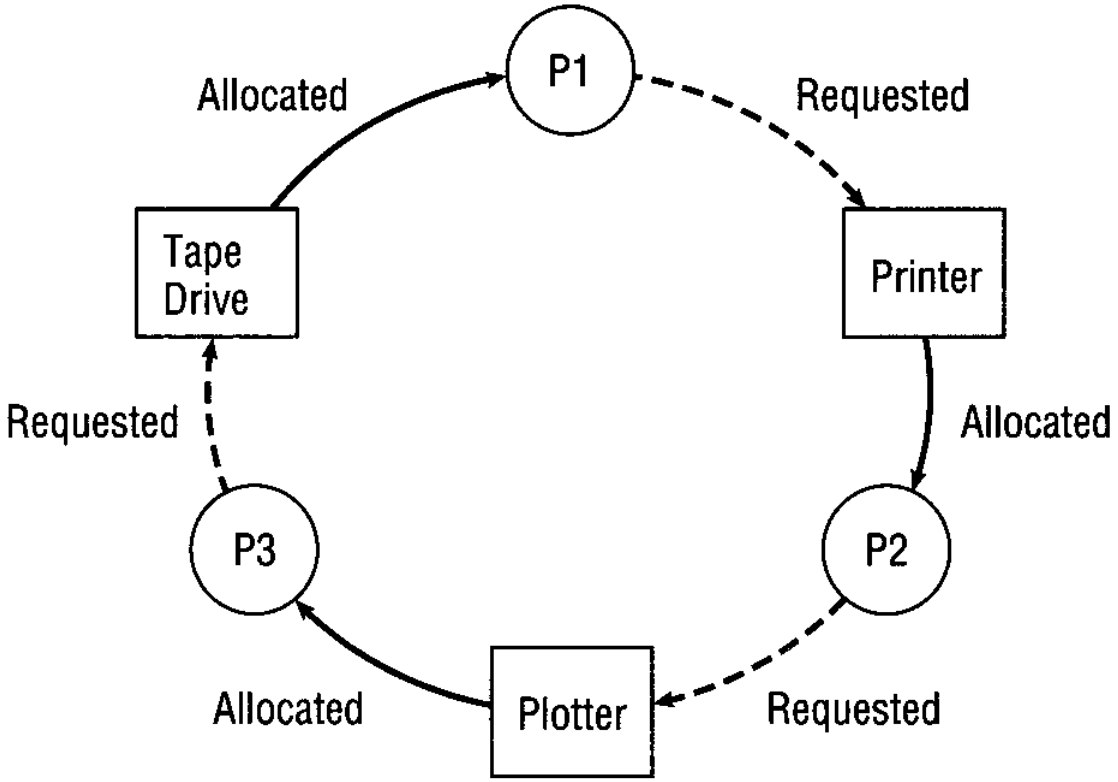


Figure 5.4: Deadlock in multiple device allocation

Case 5: Deadlocks in Spooling

- **Virtual device:** Sharable device—e.g., a printer transformed by installing a high-speed device, a disk, between it and the CPU
- **Spooling:** Disk accepts output from several users and acts as a temporary storage area for all output until printer is ready to accept it
- **Deadlock in spooling:** If printer needs all of a job's output before it will begin printing, but spooling system fills available disk space with only partially completed output

Case 6: Deadlocks in a Network

- Occurs when the network doesn't have protocols to control the flow of messages through the network
- **Example:** Refer to Figure 5.5 (next slide)
  - Consider seven computers on a network, each on different nodes
  - Direction of the arrows indicates the flow of messages
  - Deadlock occurs if all the available buffer space fills

Case 6: Deadlocks in a Network (continued)

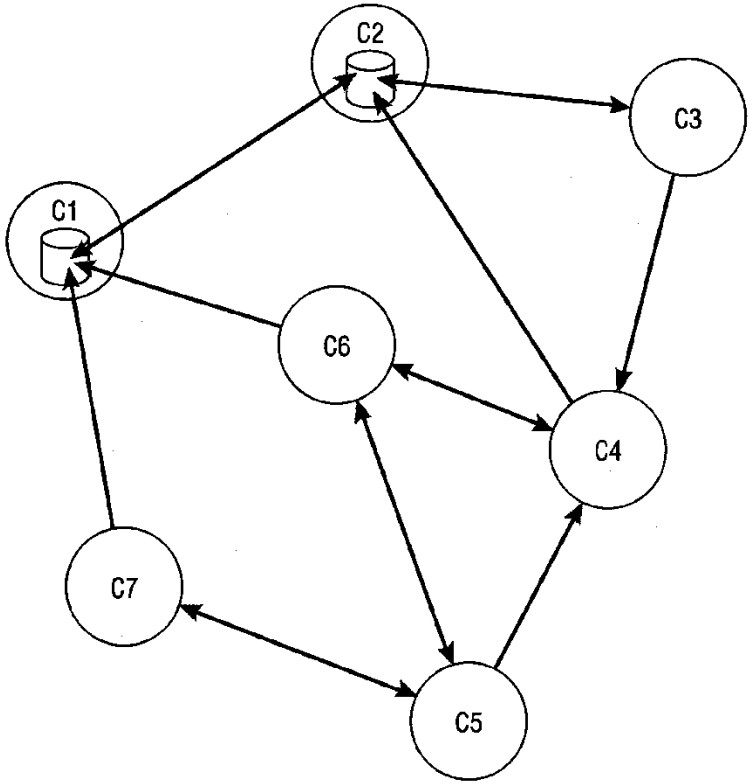


Figure 5.5: Deadlock in a network

Case 7: Deadlocks in Disk Sharing

- Occurs when competing processes send conflicting commands to access a disk
- **Example:** Refer to Figure 5.6 (next slide)
  - Two processes are each waiting for an I/O request to be filled:
    - One at cylinder 20 and one at cylinder 310
  - Neither can be satisfied because the device puts each request on hold when it tries to fulfill the other

Case 7: Deadlocks in Disk Sharing (continued)

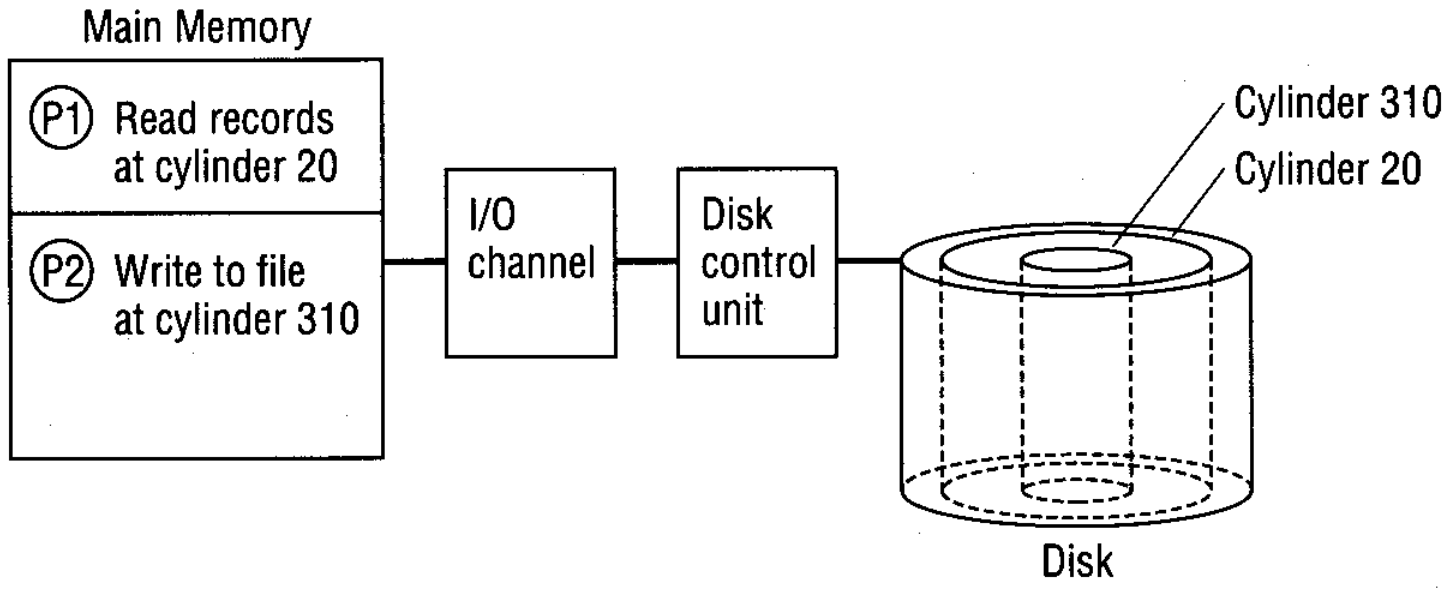


Figure 5.6: Deadlock in disk sharing

Conditions for Deadlock

- Deadlock preceded by simultaneous occurrence of the following **four conditions**:
  - Mutual exclusion
  - Resource holding
  - No preemption
  - Circular wait
- Removal of only one condition can resolve the deadlock

Conditions for Deadlock (continued)

- **Mutual exclusion:** Act of allowing only one process to have access to a dedicated resource
- **Resource holding:** Act of holding a resource and not releasing it; waiting for the other job to retreat
- **No preemption:** Lack of temporary reallocation of resources
- **Circular wait:** Each process involved in impasse is waiting for another to voluntarily release the resource so that at least one will be able to continue

Modeling Deadlocks

- **Directed graphs:**
  - Processes represented by circles
  - Resources represented by squares
  - Solid arrow from a resource to a process means that process is holding that resource
  - Solid arrow from a process to a resource means that process is waiting for that resource
  - Direction of arrow indicates flow
  - If there's a cycle in the graph then there's a deadlock involving the processes and the resources in the cycle

Modeling Deadlocks (continued)

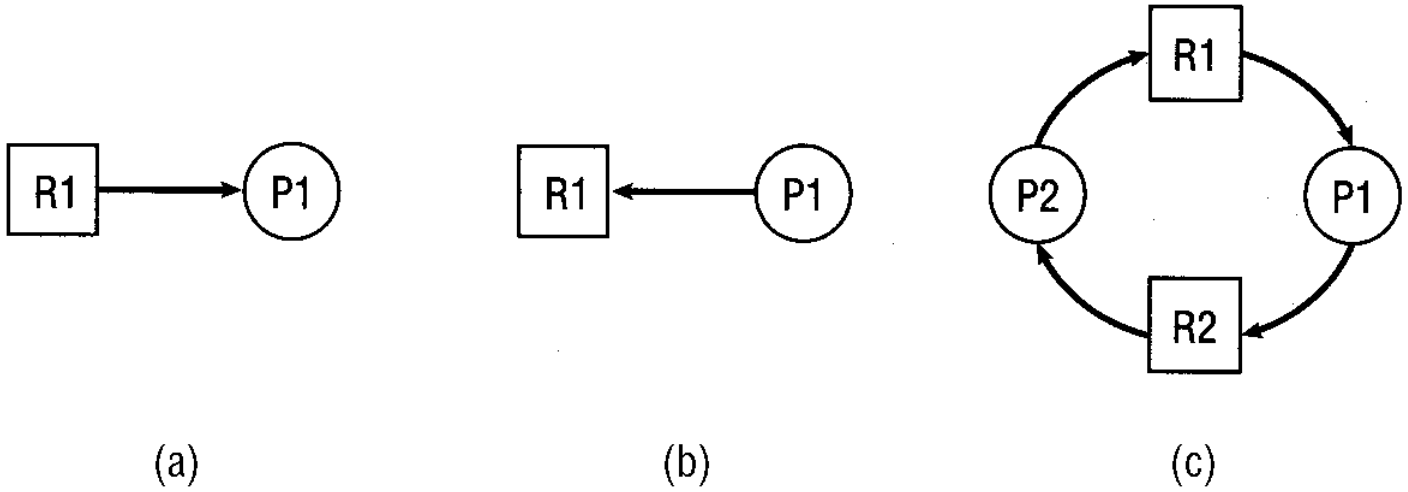


Figure 5.7: Examples of directed graphs

Scenario 1

Event	Action
1	P1 requests and is allocated the printer R1.
2	P1 releases the printer R1.
3	P2 requests and is allocated the disk drive R2.
4	P2 releases the disk R2.
5	P3 requests and is allocated the plotter R3.
6	P3 releases the plotter R3.

Table 5.1: First scenario's sequence of events is shown in the directed graph in Figure 5.8.

Scenario 1 (continued)

The system will stay free of deadlocks if all resources are released before they're requested by the next process

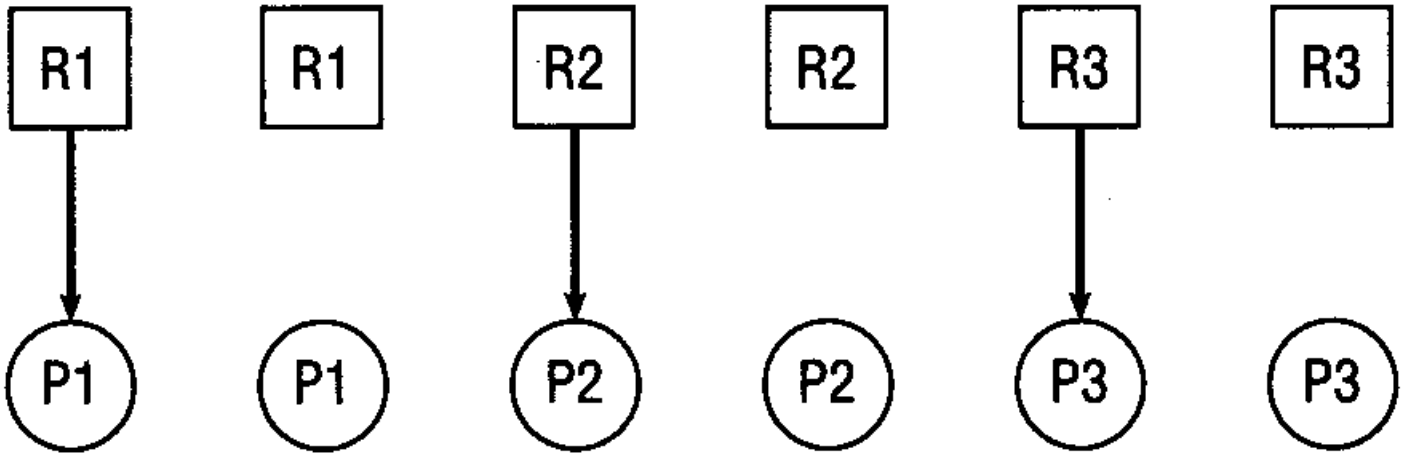


Figure 5.8: First scenario

Scenario 2

Event	Action
1	P1 requests and is allocated R1.
2	P2 requests and is allocated R2.
3	P3 requests and is allocated R3.
4	P1 requests R2.
5	P2 requests R3.
6	P3 requests R1.

Table 5.2: Second scenario's sequence of events is shown in the directed graph in Figure 5.9.

Scenario 2 (continued)

Deadlock occurs because every process is waiting for a resource being held by another process, but none will be released without operator intervention

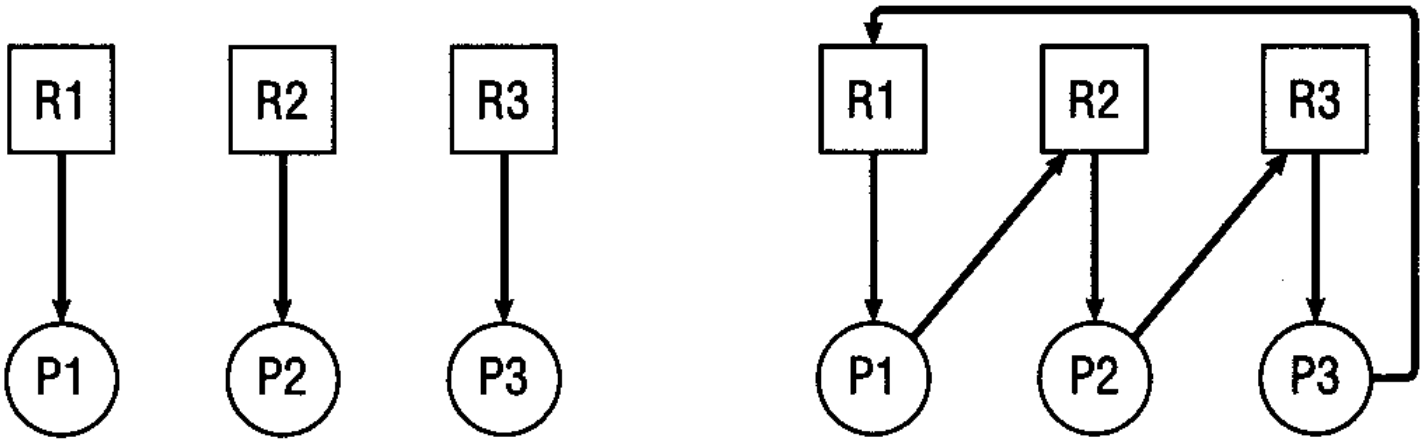


Figure 5.9: Second scenario

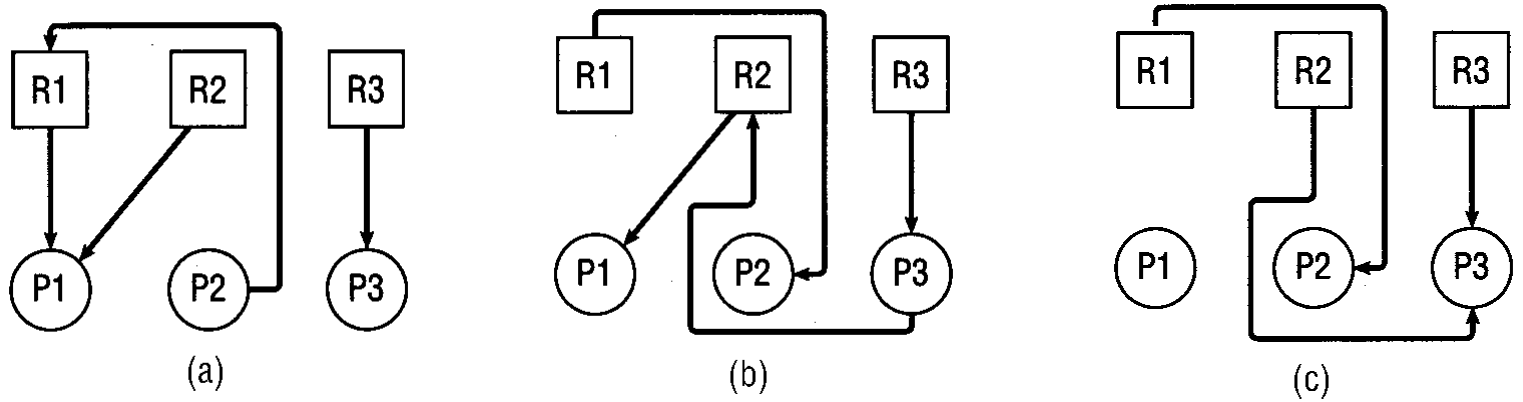
Scenario 3

Event	Action
1	P1 requests and is allocated R1.
2	P1 requests and is allocated R2.
3	P2 requests R1.
4	P3 requests and is allocated R3.
5	P1 releases R1, which is allocated to P2.
6	P3 requests R2.
7	P1 releases R2, which is allocated to P3.

Table 5.3: Third scenario's sequence of events is shown in the directed graph in Figure 5.10

Scenario 3 (continued)

- Resources are released before deadlock can occur



After event 4 the directed graph looks like (a); event 5 breaks the deadlock and the graph soon looks like (b); event 7 breaks the deadlock and the graph soon looks like (c)

Figure 5.10: Third scenario

Another Example

These graphs can cluster the devices of the same type into one entity, shown by a rectangle

The arrows show the links between the single resource and the processes using it

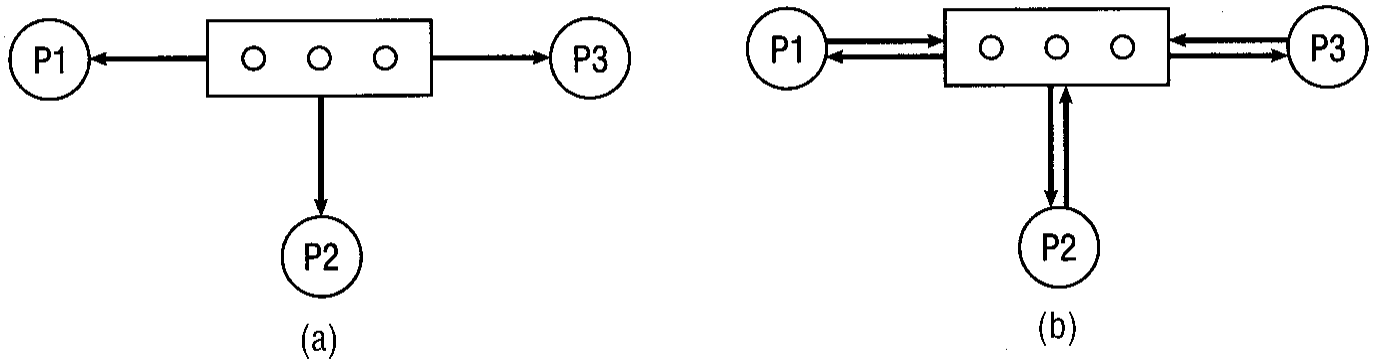


Figure 5.11: (a) A fully allocated cluster of resources; (b) When all three request another unit, deadlock occurs.

Strategies for Handling Deadlocks

- **Strategies to deal with deadlocks** include
  - **Prevention:** Prevent one of the four conditions from occurring
  - **Avoidance:** Avoid the deadlock if it becomes probable
  - **Detection:** Detect the deadlock when it occurs and recover from it gracefully

Strategies for Handling Deadlocks (continued)

- **Prevention:** Necessary to eliminate one of the four conditions, while same condition can't be eliminated from every resource
  - **Mutual exclusion** is necessary
    - Exceptions include devices where mutual exclusion can be bypassed by spooling
  - **Resource holding** can be avoided by forcing each job to request, at creation time, every resource it will need
    - Significantly decreases degree of multiprogramming
    - Peripheral devices would be idle

Strategies for Handling Deadlocks (continued)

- **No preemption** could be bypassed by allowing OS to deallocate resources from jobs
  - Okay if state of job can be easily saved and restored
  - Not accepted to preempt dedicated I/O device or files during modification
- **Circular wait** can be bypassed if the operating system prevents the formation of a circle
  - Use **scheme of hierarchical ordering**
    - Requires that jobs anticipate order in which they will request resources
    - Difficult to satisfy all users

Avoidance

- Deadlock can be avoided if system knows ahead of time sequence of requests associated with each of the active processes
- **Dijkstra's Bankers Algorithm** (1965) regulates resources allocation to avoid deadlock
  - No customer granted loan exceeding bank's total capital
  - All customers given a maximum credit limit
  - No customer allowed to borrow over the limit
  - The sum of all loans won't exceed bank's total capital

Avoidance (continued)

Safe state: Bank still has enough money left after loans to satisfy the maximum requests of C1, C2, or C3

Customer	Loan amount	Maximum credit	Remaining credit
C1	0	4,000	4,000
C2	2,000	5,000	3,000
C3	4,000	8,000	4,000
Total loaned: $6,000			
Total capital fund: $10,000			

Table 5.4: The bank started with $10,000 and has remaining capital of $4,000 after these loans

Avoidance (continued)

Unsafe state: Bank has not enough money left after loans to satisfy the maximum requests of C1, C2, or C3

Customer	Loan amount	Maximum credit	Remaining credit
C1	2,000	4,000	2,000
C2	3,000	5,000	2,000
C3	4,000	8,000	4,000
Total loaned: $9,000			
Total capital fund: $10,000			

Table 5.5: The bank has remaining capital of only $1,000 after these loans and therefore is in an “unsafe state”

Avoidance (continued)

Same banking principles can be applied to an operating system

Job No.	Devices Allocated	Maximum Required	Remaining Needs
1	0	4	4
2	2	5	3
3	4	8	4
Total number of devices allocated: 6			
Total number of devices in system: 10			

Table 5.6: A safe state: six devices are allocated and four units are still available

Avoidance (continued)

Job No.	Devices Allocated	Maximum Required	Remaining Needs
1	2	4	2
2	3	5	2
3	4	8	4
Total number of devices allocated: 9			
Total number of devices in system: 10			

Table 5.7: An unsafe state: only one unit is available but every job requires at least two to complete its execution

Avoidance (continued)

- To avoid deadlock, OS must make sure:
  - Never satisfy a request that moves it from a safe state to an unsafe one
  - Must identify the job with the smallest number of remaining resources
  - Number of available resources is always equal to, or greater than, the number needed for the selected job to run to completion

Avoidance (continued)

- **Problems with Banker's Algorithm:**
  - Jobs must state the maximum number of resources needed
  - Number of total resources for each class must remain constant
  - Number of jobs must remain fixed
  - Overhead cost incurred can be quite high
  - Resources aren't well utilized because the algorithm assumes the worst case
  - Scheduling suffers as a result of poor utilization and jobs are kept waiting for resource allocation

Detection

- Deadlocks can be detected by building directed resource graphs and looking for cycles
- Algorithm used to detect circularity can be executed whenever it is appropriate
- **The detection algorithm:**
  1. Remove a process that is currently using a resource and not waiting for one
  2. Remove a process that's waiting only for resource classes that aren't fully allocated
  3. Go back to Step 1 and continue with Steps 1 and 2 until all connecting lines have been removed

Detection (continued)

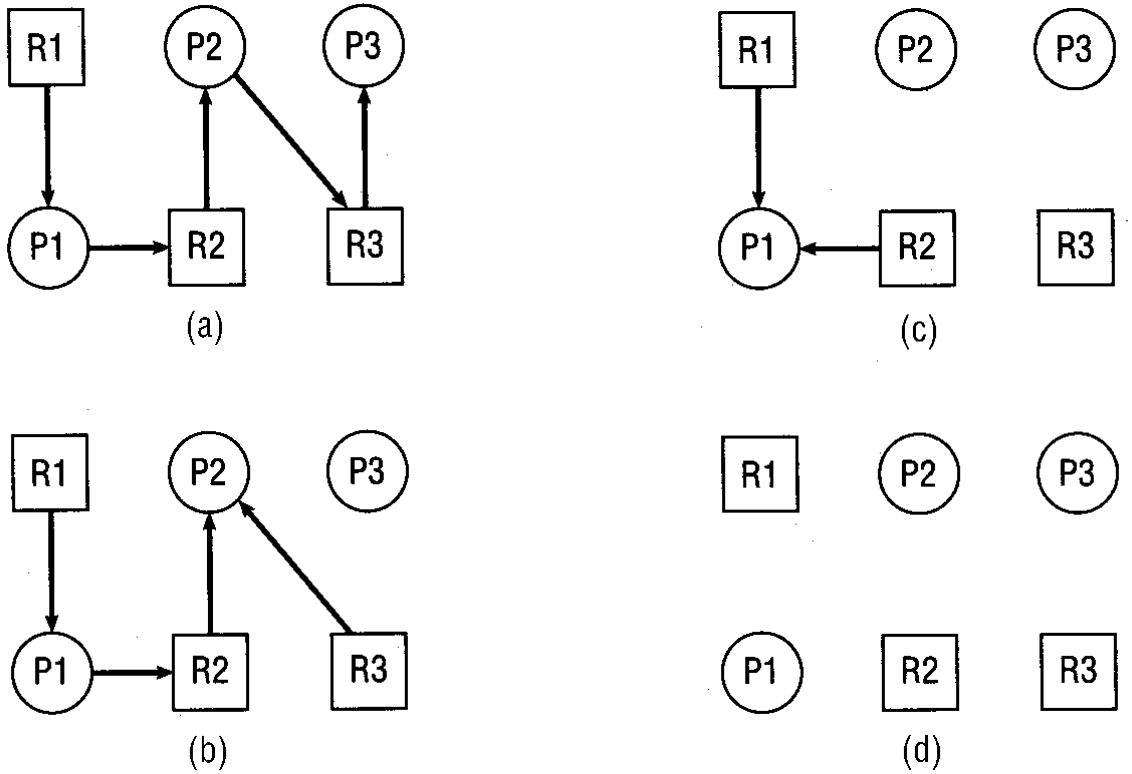
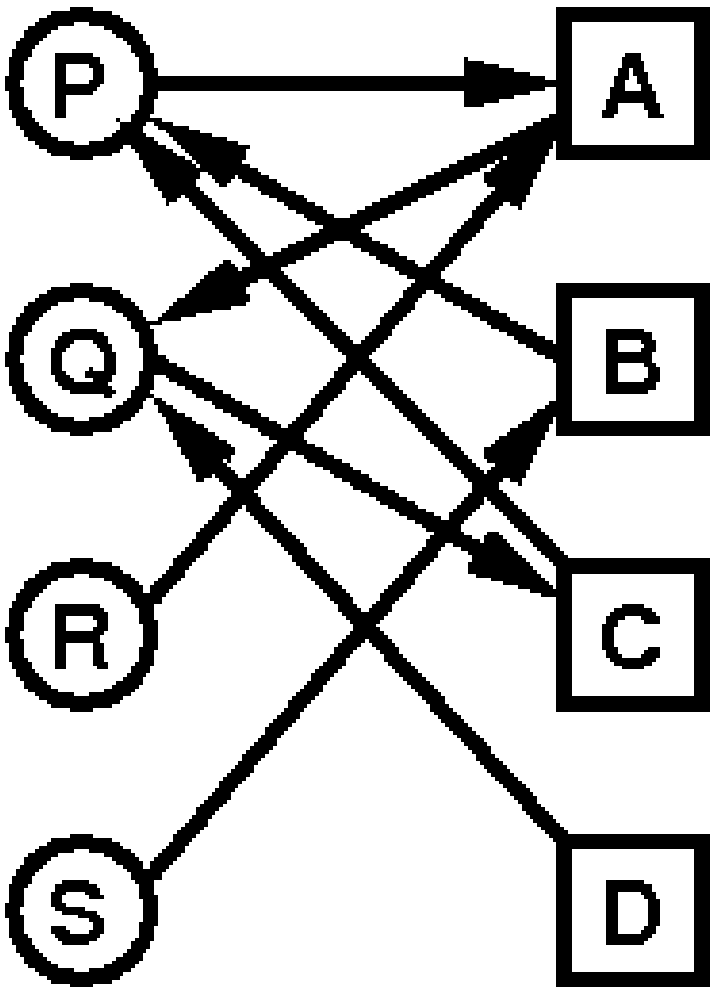
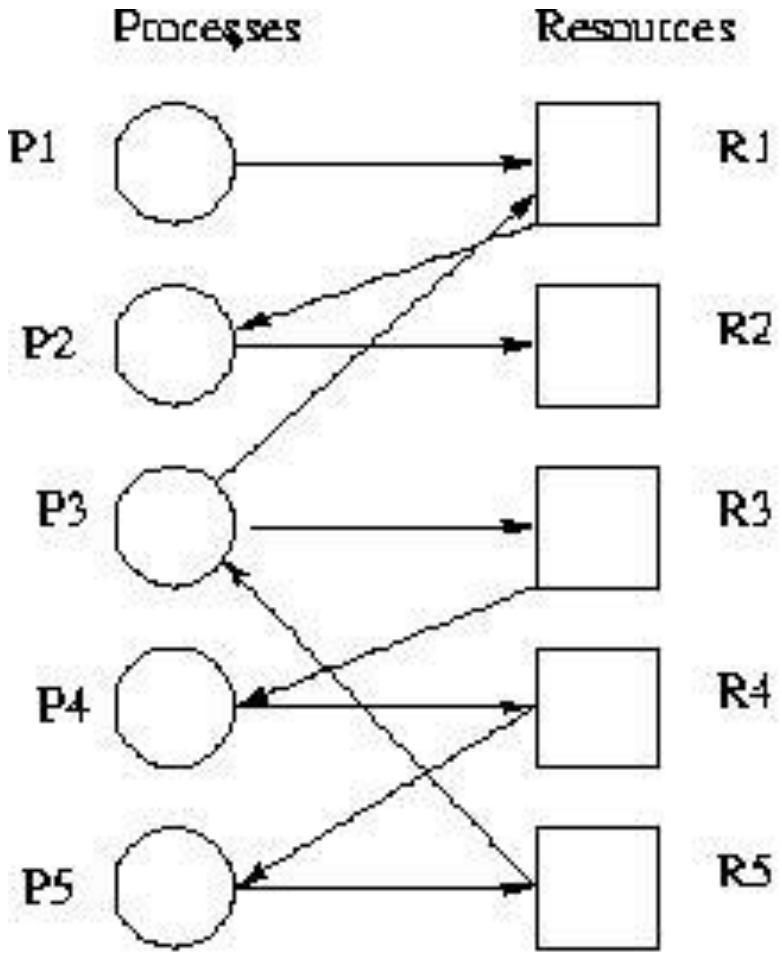


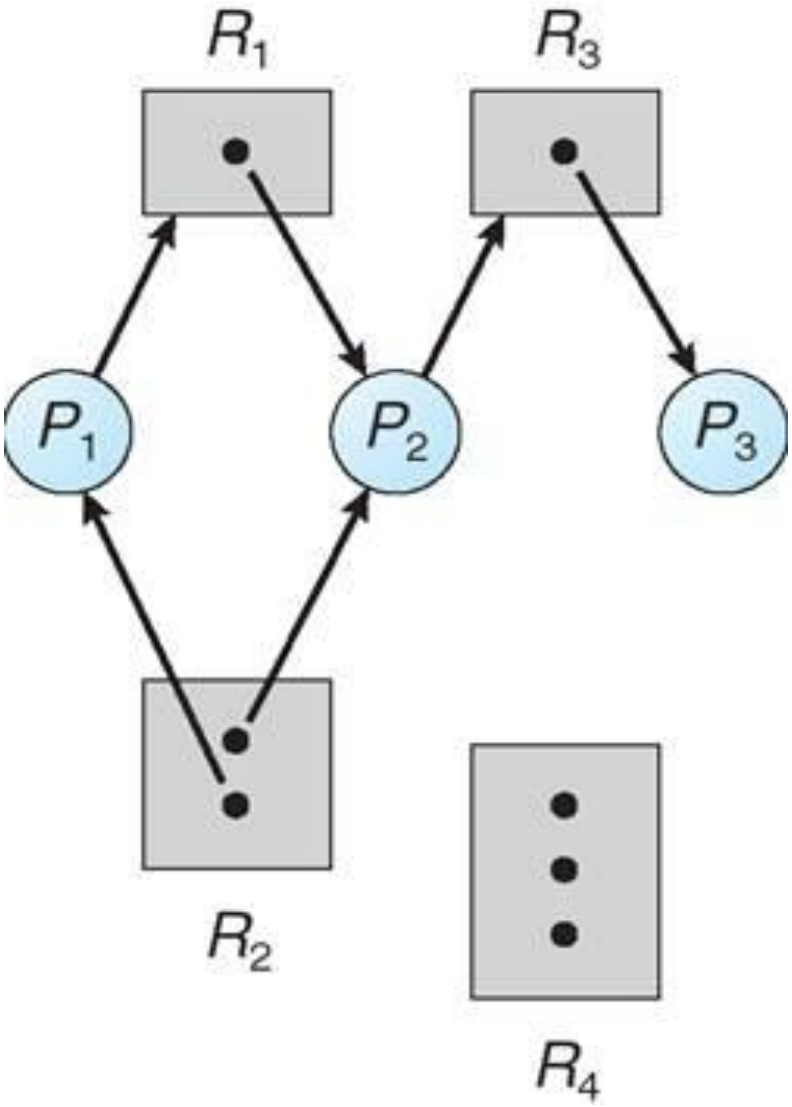
Figure 5.12: Deadlock-free system because the graph can be completely reduced, as shown in (d)



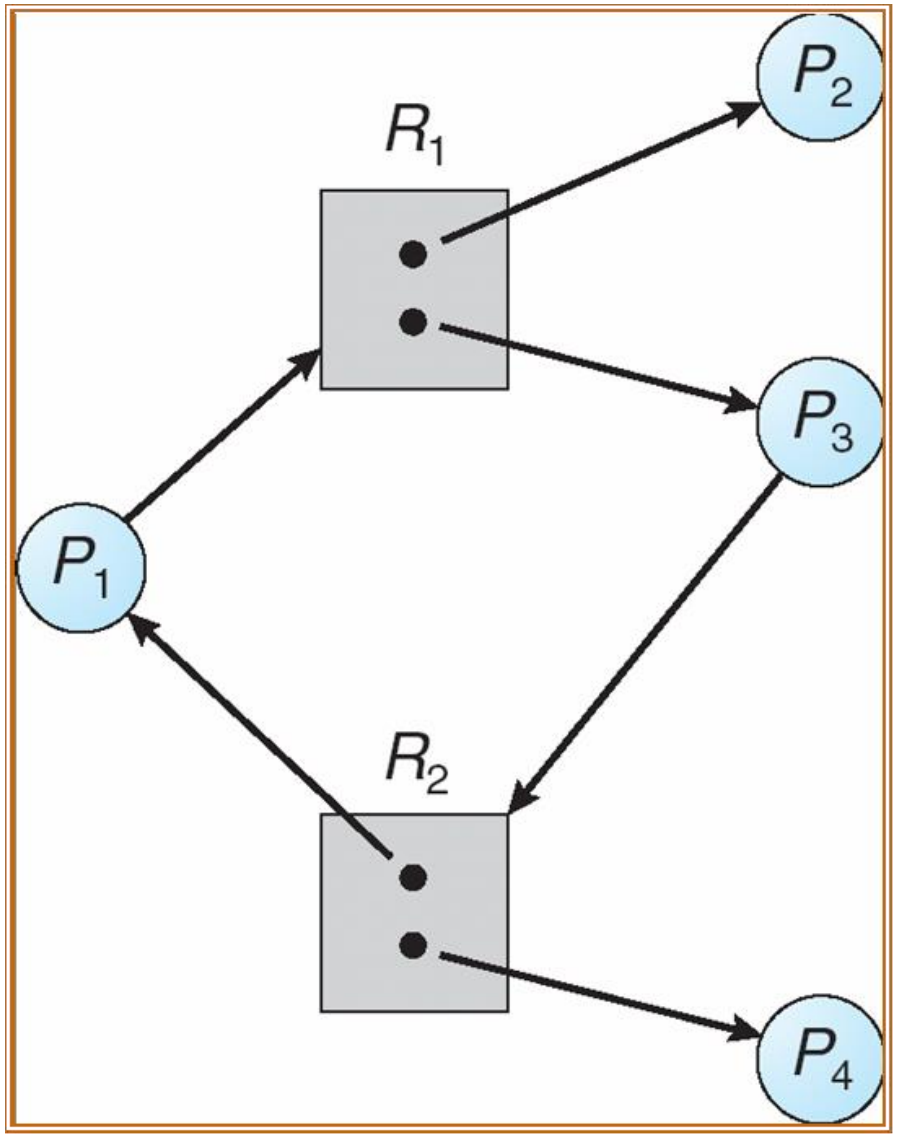
Is there a deadlock ?



Is there a deadlock ?



Is there a deadlock ?



Is there a deadlock ?

Recovery

- **Deadlock must be untangled once detected**, so the system returns to normal quickly
- All recovery methods have at least one victim
- **Recovery Methods:**
  - Terminate every job that's active in the system and restart them from the beginning
  - Terminate only the jobs involved in the deadlock and ask their users to resubmit them
  - Identify jobs involved in deadlock and terminate them one at a time

Recovery (continued)

- (continued):
  - Jobs that keep a record (snapshot) of their progress can be interrupted
  - Select a nondeadlocked job, preempt its resources, allocate them to a deadlocked process
  - Stop new jobs from entering system, which allows nondeadlocked jobs to run to completion so they'll release their resources (no victim)

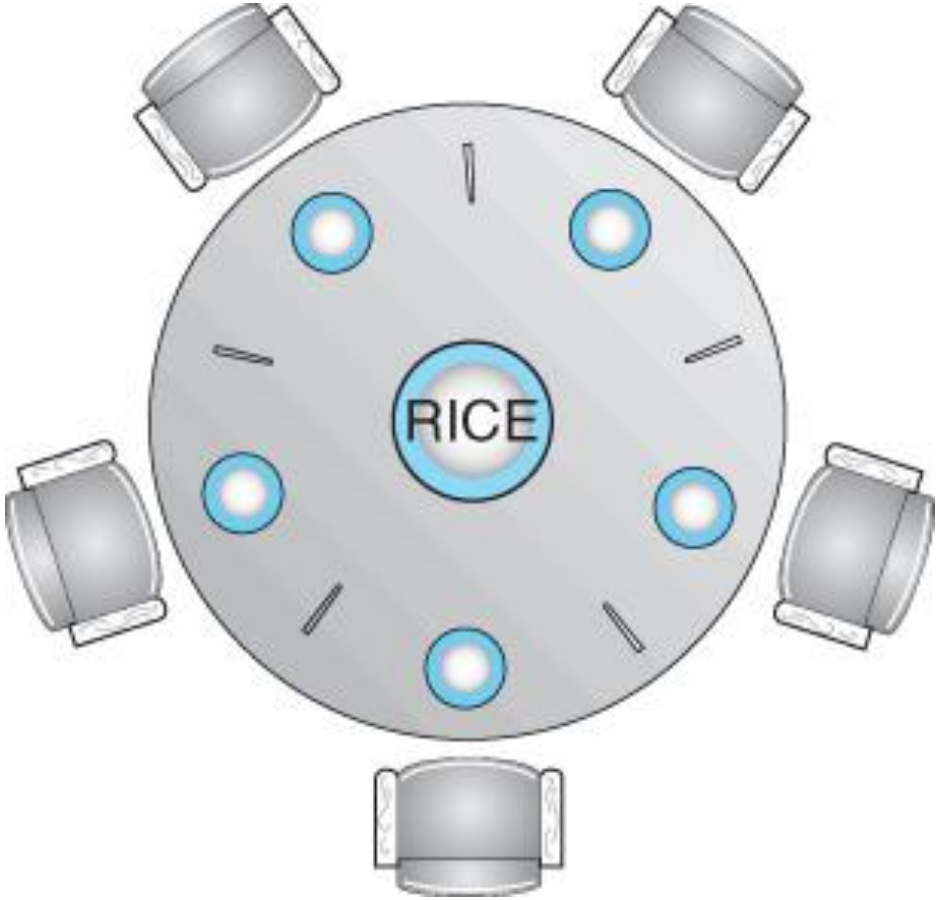
Recovery (continued)

- Select the victim that will have the **least-negative effect** on the system
- **Factors** to be considered to select a victim:
  - Priority of job under consideration: high-priority jobs are usually untouched
  - CPU time used by job: jobs close to completion are usually left alone
  - Number of other jobs that would be affected if this job were selected as the victim
  - Jobs that are modifying data shouldn't be selected for termination

Starvation

- Job is prevented from execution as it's kept waiting for resources that never become available
  - Result of conservative allocation of resources
- **Example:**
  - "The dining philosophers table" Dijkstra (1968)
- **To avoid starvation:**
  - Implement algorithm to track how long each job has been waiting for resources (aging)
  - Block new jobs until the starving jobs have been satisfied

Starvation (continued)



Each philosopher must have two chop sticks to begin eating, one on the right and one on left.

Unless chopsticks (resources) are allocated fairly, some philosophers may starve

Starvation — The dining philosophers' table

Summary

- Every operating system must dynamically allocate a limited number of resources while avoiding the two extremes of deadlock and starvation
- Several methods of dealing with deadlocks include prevention, avoidance, and detection and recovery
- Deadlocks can be prevented by not allowing at least one of the four conditions of a deadlock to occur in the system at the same time as the others
- Prevention algorithms are complex and involve high overhead to routinely execute them

Summary (continued)

- Deadlocks can be avoided by clearly identifying safe states and unsafe states
- System should keep enough resources in reserve to guarantee that all jobs can run to completion
- In an avoidance policy, system's resources aren't allocated to their fullest potential
- System must be prepared to detect and recover from the deadlocks
- Detection option usually relies on the selection of at least one "victim"