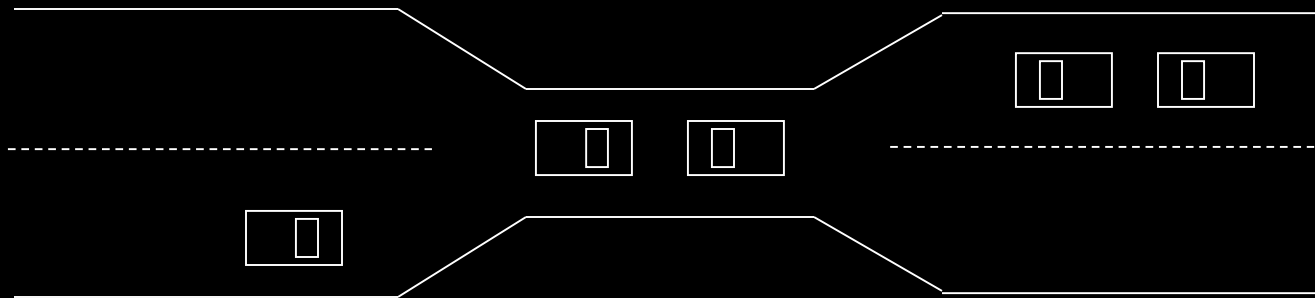


Chapter 6: Deadlocks

- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock
- Combined Approach to Deadlock Handling

Bridge Crossing Example



- ▶ Traffic only in one direction.
- ▶ Each section of a bridge can be viewed as a resource.
- ▶ If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- ▶ Several cars may have to be backed up if a deadlock occurs.
- ▶ Starvation is possible.

The Deadlock Problem

- ▶ A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- ▶ Example
 - ▶ System has 2 tape drives.
 - ▶ P_1 and P_2 each hold one tape drive and each needs another one.
- ▶ Example
 - ▶ semaphores A and B , initialized to 1

P_0	P_1
<i>wait (A);</i>	<i>wait(B)</i>
<i>wait (B);</i>	<i>wait(A)</i>

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- ▶ **Mutual exclusion:** only one process at a time can use a resource.
- ▶ **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
- ▶ **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- ▶ **Circular wait:** there exists a set $\{P_0, P_1, \dots, P_{n-1}\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , ..., P_{n-1} is waiting for a resource that is held by P_0 .

System Model

- Resource types R_1, R_2, \dots, R_m
CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
 - request
 - use
 - release

Resource-Allocation Graph

A set of vertices V and a set of edges E .

- V is partitioned into two types:
 - $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the processes in the system.
 - $R = \{R_1, R_2, \dots, R_m\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P_1 \rightarrow R_j$
- assignment edge – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (Cont.)

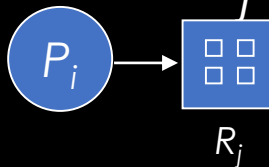
- ▶ Process



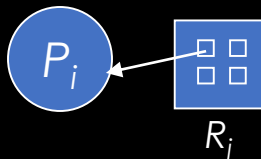
- ▶ Resource Type with 4 instances



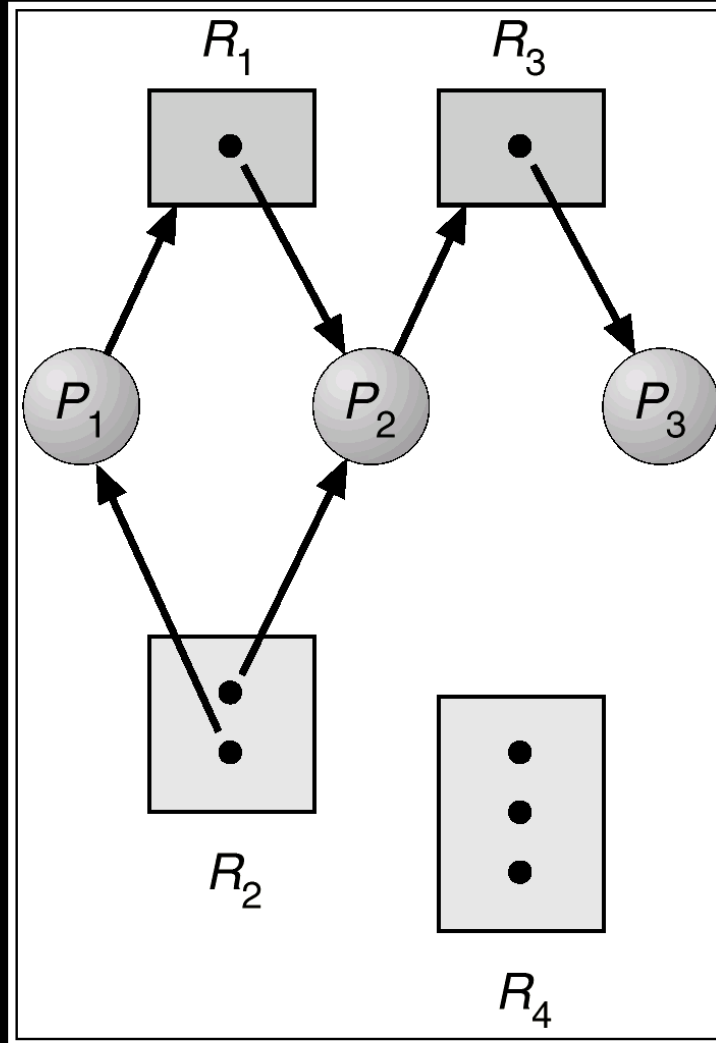
- ▶ P_i requests instance of R_j



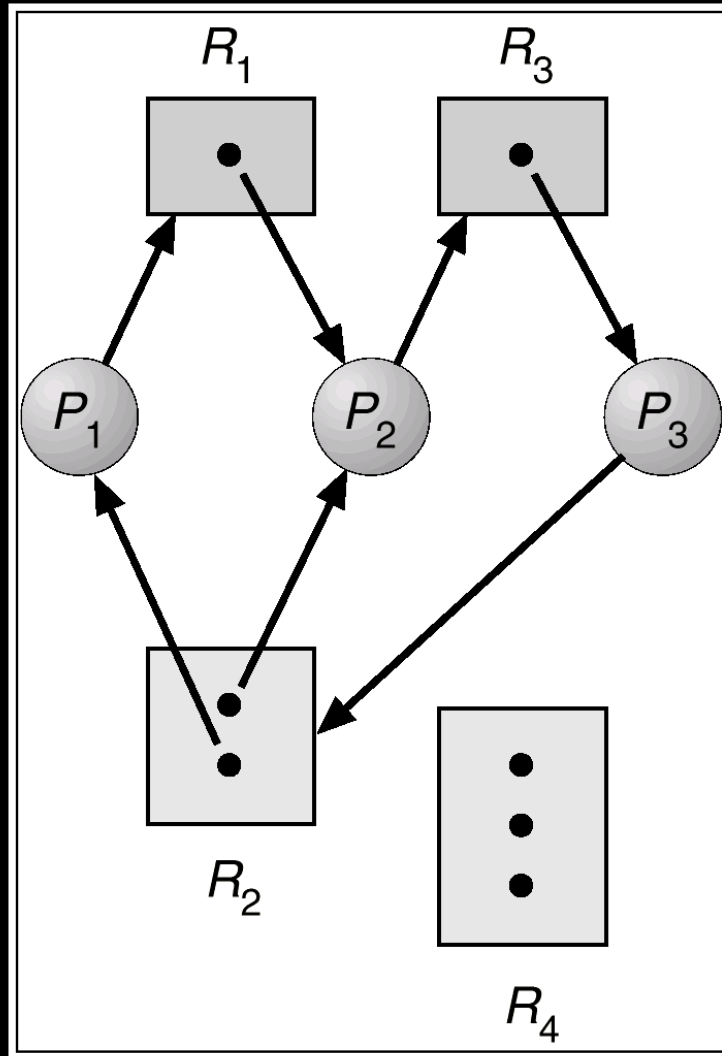
- ▶ P_i is holding an instance of R_j



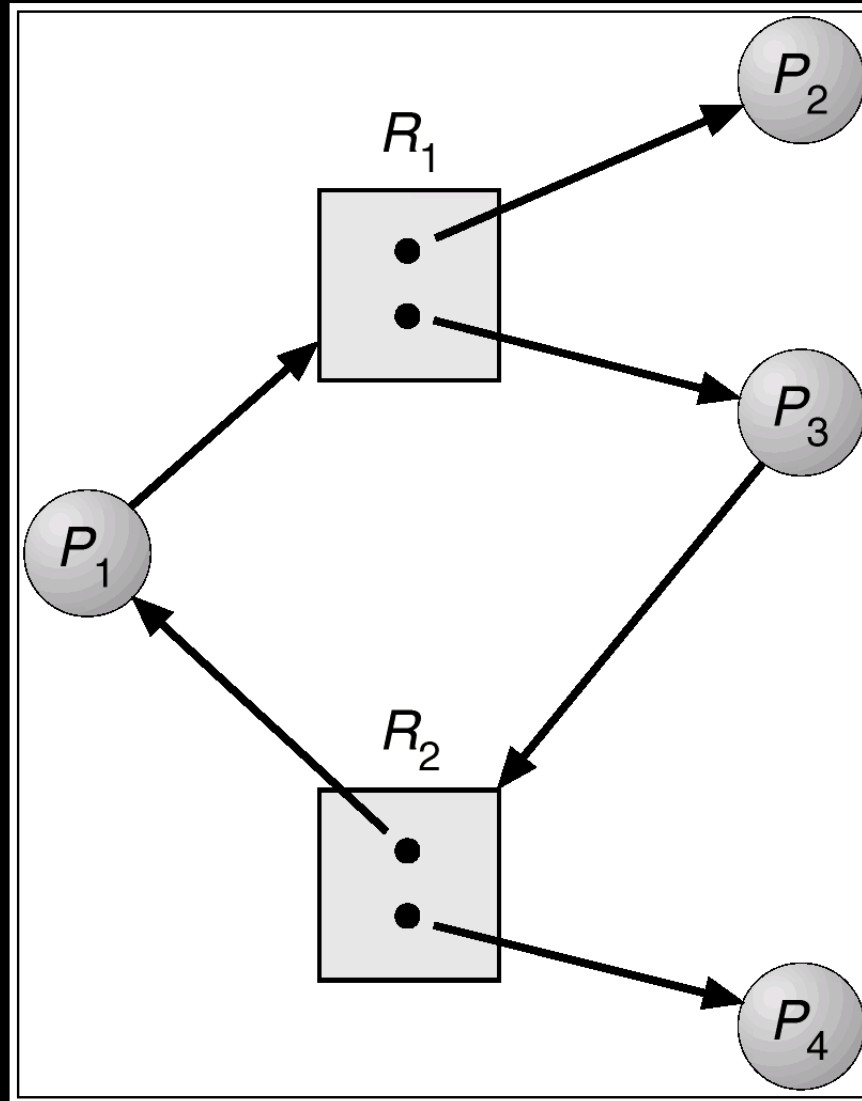
Example of a Resource Allocation Graph



Resource Allocation Graph With A Deadlock



Resource Allocation Graph With A Cycle But No Deadlock



Basic Facts

- If graph contains no cycles \Rightarrow no deadlock.
- If graph contains a cycle \Rightarrow
 - if only one instance per resource type, then deadlock.
 - if several instances per resource type, possibility of deadlock.

Methods for Handling Deadlocks

- Ensure that the system will *never* enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

Deadlock Prevention

Restrain the ways request can be made.

- **Mutual Exclusion** – not required for sharable resources; must hold for nonsharable resources.
- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources.
 - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.
 - Low resource utilization; starvation possible.

Deadlock Prevention (Cont.)

▶ **No Preemption** –

- ▶ If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
- ▶ Preempted resources are added to the list of resources for which the process is waiting.
- ▶ Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

▶ **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

Deadlock Avoidance

Requires that the system has some additional *a priori* information available.

- ▶ Simplest and most useful (??) model requires that each process declare the *maximum number* of resources of each type that it may need.
- ▶ Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes.
- ▶ The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.

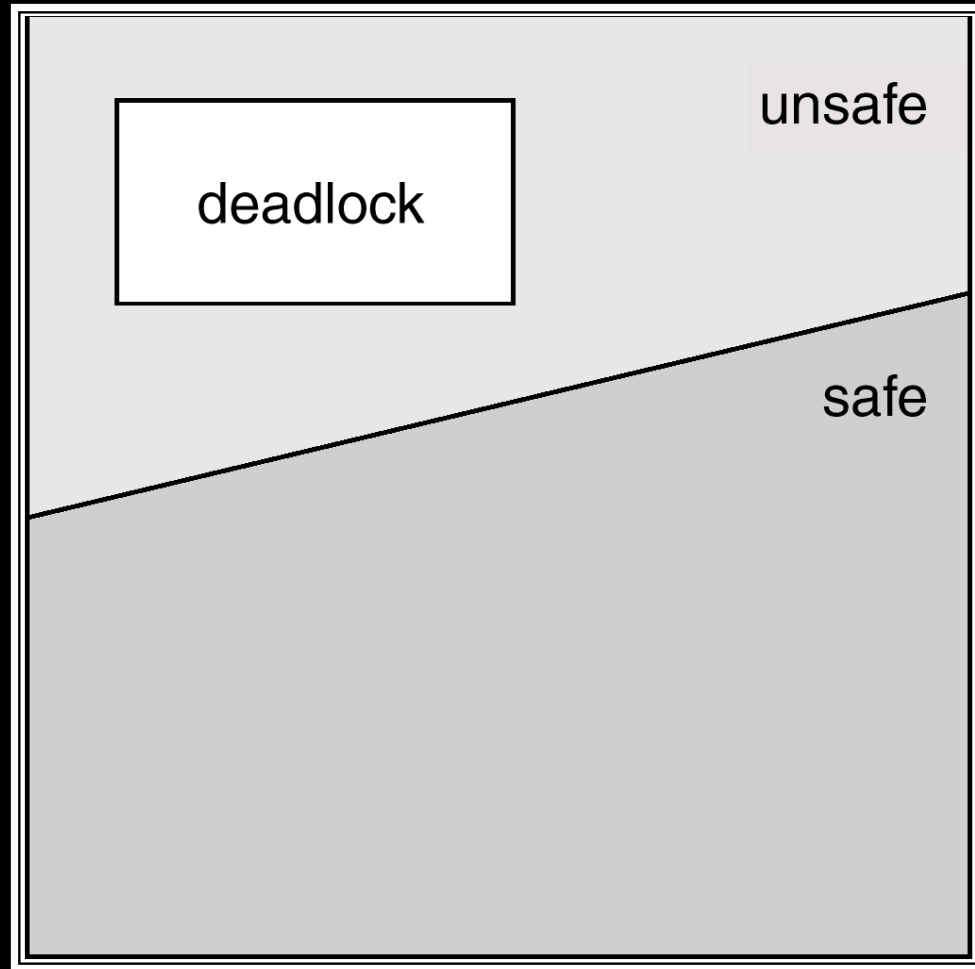
Safe State

- ▶ When a process requests an available resource, system must decide if immediate allocation leaves the system in a *safe state*.
- ▶ System is in safe state if there exists a safe sequence of all processes.
- ▶ Sequence $\langle P_1, P_2, \dots, P_n \rangle$ is safe if for each P_i , the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j , with $j < i$.
 - ▶ If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
 - ▶ When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
 - ▶ When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

Basic Facts

- If a system is in safe state \Rightarrow no deadlocks.
- If a system is in unsafe state \Rightarrow possibility of deadlock.
- Avoidance \Rightarrow ensure that a system will never enter an unsafe state.

Safe, Unsafe , Deadlock State



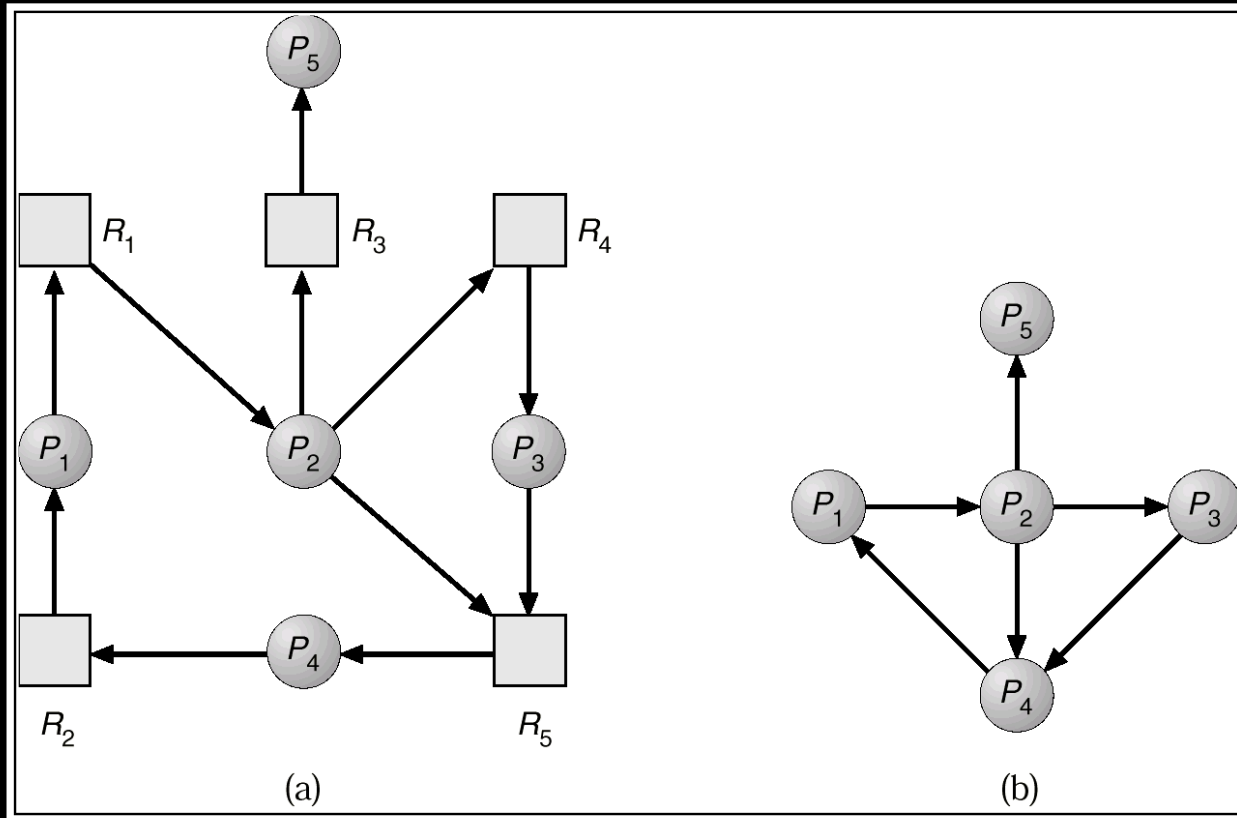
Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

Single Instance of Each Resource Type

- Maintain *wait-for* graph
 - Nodes are processes.
 - $P_i \rightarrow P_j$ if P_i is waiting for P_j .
- Periodically invoke an algorithm that searches for a cycle in the graph.
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph

Corresponding wait-for graph

Several Instances of a Resource Type

- ▶ *Available*: A vector of length m indicates the number of available resources of each type.
- ▶ *Allocation*: An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.
- ▶ *Request*: An $n \times m$ matrix indicates the current request of each process. If $Request [i,j] = k$, then process P_i is requesting k more instances of resource type R_j .
- ▶ Complexity $O(m * n^2)$

Detection Algorithm

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively
Initialize:
 - (a) *Work* = *Available*
 - (b) For $i = 1, 2, \dots, n$, if $Allocation_i \neq 0$, then $Finish[i] = false$; otherwise, $Finish[i] = true$.
2. Find an index *i* such that both:
 - (a) $Finish[i] == false$
 - (b) $Request_i \leq Work$If no such *i* exists, go to step 4.
3. $Work = Work + Allocation_i$
 $Finish[i] = true$
go to step 2.
4. If $Finish[i] == false$, for some $i, 1 \leq i \leq n$, then the system is in deadlock state. Moreover, if $Finish[i] == false$, then P_i is deadlocked.

Detection Algorithm (Cont.)

Algorithm requires an order of $O(m \times n^2)$ operations to detect whether the system is in deadlocked state.

Example of Detection Algorithm

- ▶ Five processes P_0 through P_4 ; three resource types A (7 instances), B (2 instances), and C (6 instances).
- ▶ Snapshot at time T_0 :

	<u>Allocation</u>			<u>Request</u>			<u>Available</u>		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

- ▶ Sequence $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ will result in $Finish[i] = \text{true}$ for all i .

Example (Cont.)

- ▶ P_2 requests an additional instance of type C.

	<u>Request</u>		
	A	B	C
P_0	0	0	0
P_1	2	0	1
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

- ▶ State of system?
 - ▶ Can reclaim resources held by process P_0 , but insufficient resources to fulfill other processes; requests.
 - ▶ Deadlock exists, consisting of processes P_1 , P_2 , P_3 , and P_4 .

Detection-Algorithm Usage

- When, and how often, to invoke depends on:
 - How often a deadlock is likely to occur?
 - How many processes will need to be rolled back?
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock.

Recovery from Deadlock: Process Termination

- ▶ Abort all deadlocked processes.
- ▶ Abort one process at a time until the deadlock cycle is eliminated.
- ▶ In which order should we choose to abort?
 - ▶ Priority of the process.
 - ▶ How long process has computed, and how much longer to completion.
 - ▶ Resources the process has used.
 - ▶ Resources process needs to complete.
 - ▶ How many processes will need to be terminated.
 - ▶ Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

- Selecting a victim – minimize cost.
- Rollback – return to some safe state, restart process for that state.
- Starvation – same process may always be picked as victim, include number of rollback in cost factor.

Combined Approach to Deadlock Handling

- Combine the three basic approaches
 - prevention
 - avoidance
 - detection

allowing the use of the optimal approach for each of resources in the system.

- Partition resources into hierarchically ordered classes.

Traffic Deadlock

