

# Module 6: Deadlocks

■ CIS 21012 – Platform Technologies

*Shafana M. Shareef*

*South Eastern University of Sri Lanka*

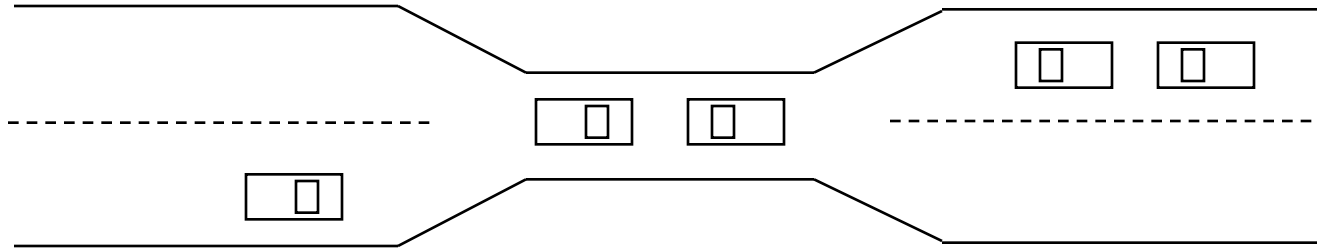
# Chapter 5: Deadlocks

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

# Chapter Objectives

- To develop a description of **deadlocks**, which prevent sets of concurrent processes from completing their tasks
- To present a number of different methods for preventing or avoiding deadlocks in a computer system

# The Deadlock: 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.

# Deadlock

- A system is said to be in Deadlock when every processes in the set waits for an event that can be caused by another process in the set(system).

# 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  $\{P_0, P_1, \dots, P_n\}$  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  $P_n$ , and  $P_n$  is waiting for a resource that is held by  $P_0$ .

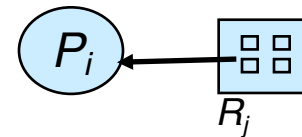
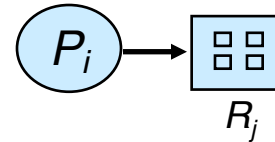
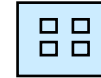
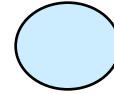
# Resource-Allocation Graph

**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  $P_i \rightarrow R_j$
- **assignment edge** – directed edge  $R_j \rightarrow P_i$

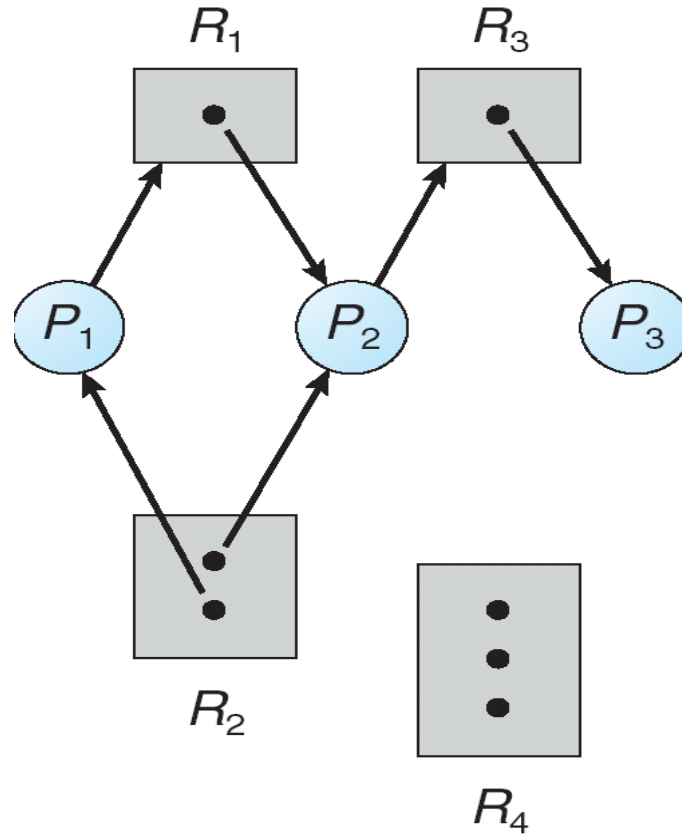
# Resource-Allocation Graph (Cont.)

- Process
- Resource Type with 4 instances
- $P_i$  requests instance of  $R_j$
- $P_i$  is holding an instance of  $R_j$

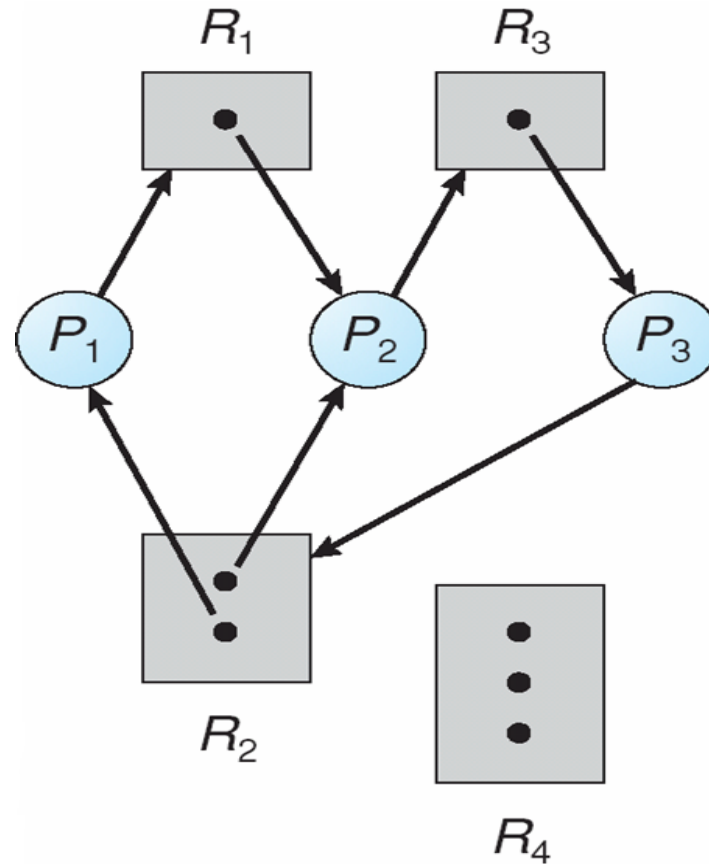




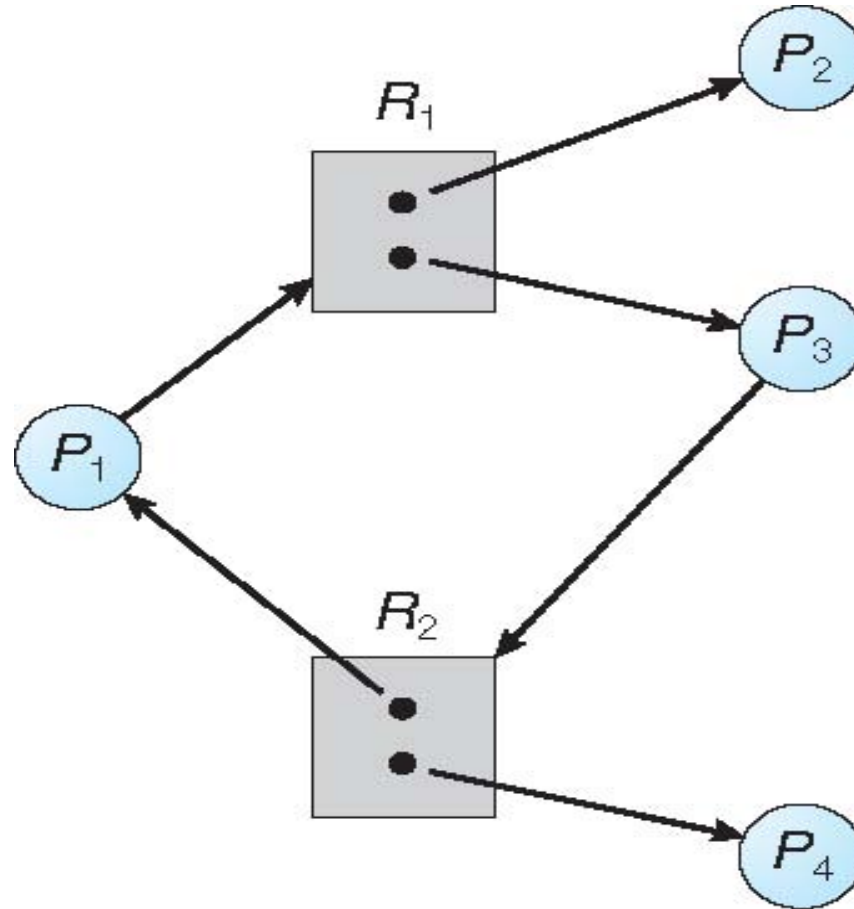
# Example of a Resource Allocation Graph



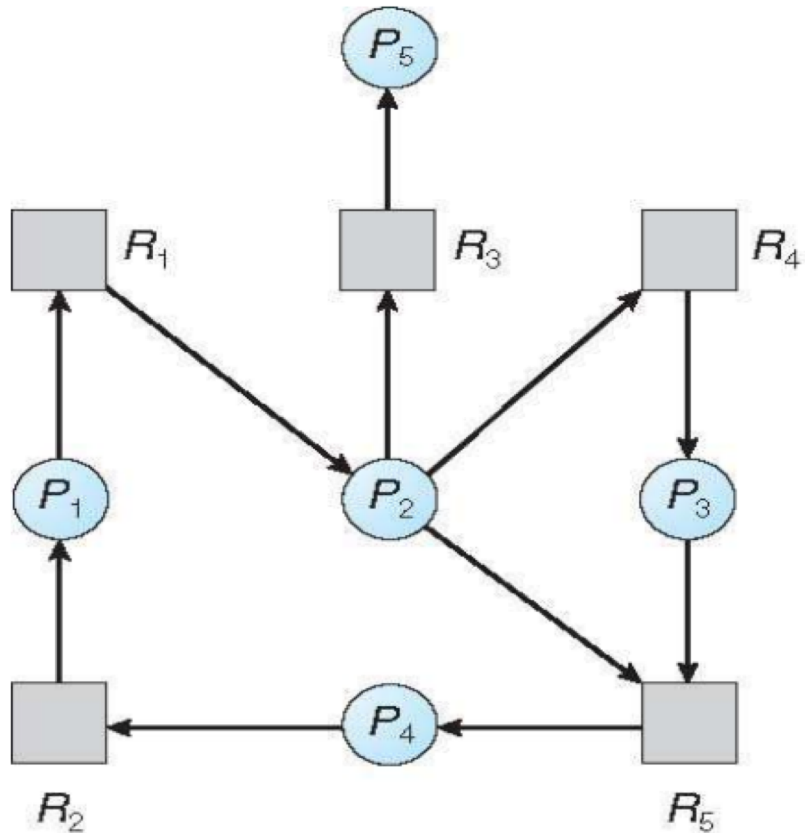
# Resource Allocation Graph With A Deadlock



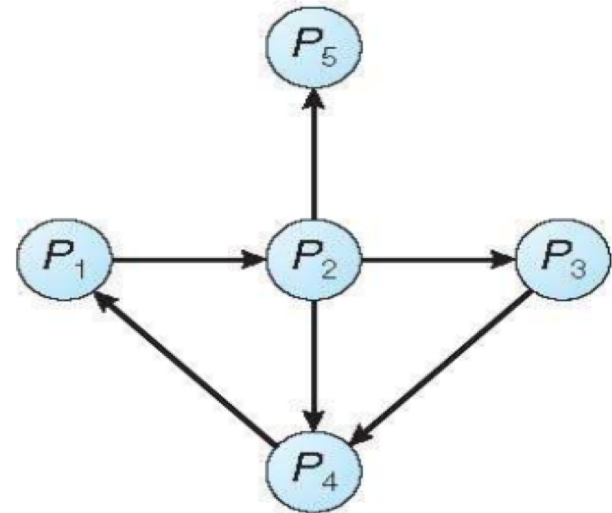
# Graph With A Cycle But No Deadlock



# Resource-Allocation Graph and Wait-for Graph



(a)



(b)

# 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:
  - Deadlock avoidance
- Allow the system to enter a deadlock state and then recover
  - Deadlock prevention

# Deadlock Prevention

## Restrain the ways request can be made

- **Mutual Exclusion** – not required for sharable resources (e.g., read-only files); must hold for non-sharable 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 allocated to it.
  - 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
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes

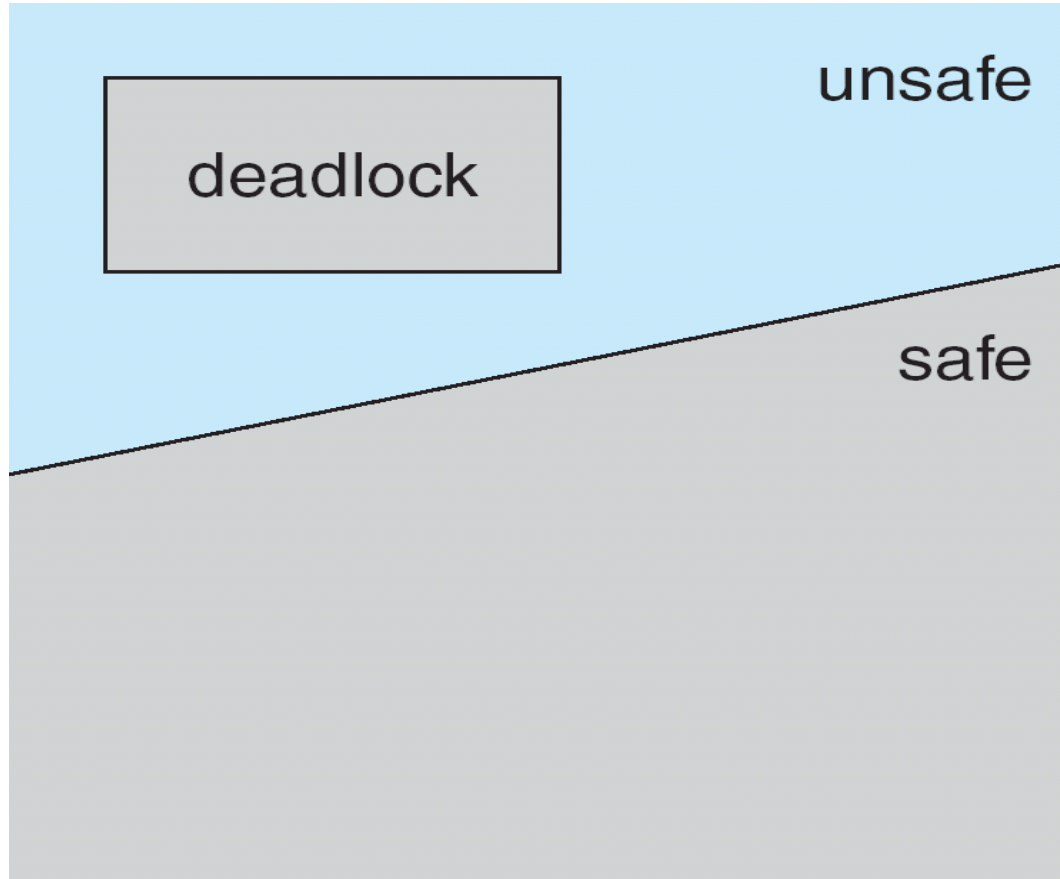
# Safe State

- 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 sequence  $\langle P_1, P_2, \dots, P_n \rangle$  of ALL the processes in the systems such that 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  $j < i$
- That is:
  - 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



# Banker's Algorithm

- Multiple instances
- Each process must a priori claim maximum use
- When a process requests a resource it may have to wait
- When a process gets all its resources it must return them in a finite amount of time

# Example of Banker's Algorithm

- 5 processes  $P_0$  through  $P_4$ ;  
3 resource types:  
A (10 instances), B (5 instances), and C (7 instances)
- Snapshot at time  $T_0$ :

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>
	A B C	A B C	A B C
$P_0$	0 1 0	7 5 3	3 3 2
$P_1$	2 0 0	3 2 2	
$P_2$	3 0 2	9 0 2	
$P_3$	2 1 1	2 2 2	
$P_4$	0 0 2	4 3 3	

# Example (Cont.)

- The content of the matrix ***Need*** is defined to be ***Max – Allocation***

	<u><i>Need</i></u>
	<i>A B C</i>
$P_0$	7 4 3
$P_1$	1 2 2
$P_2$	6 0 0
$P_3$	0 1 1
$P_4$	4 3 1

- The system is in a safe state since the sequence  $\langle P_1, P_3, P_4, P_2, P_0 \rangle$  satisfies safety criteria

# Example: $P_1$ Request (1,0,2)

- Check that Request  $\leq$  Available (that is,  $(1,0,2) \leq (3,3,2) \Rightarrow$  true

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
$P_0$	0 1 0	7 4 3	2 3 0
$P_1$	3 0 2	0 2 0	
$P_2$	3 0 2	6 0 0	
$P_3$	2 1 1	0 1 1	
$P_4$	0 0 2	4 3 1	

- Executing safety algorithm shows that sequence  $\langle P_1, P_3, P_4, P_0, P_2 \rangle$  satisfies safety requirement
- Can request for (3,3,0) by  $P_4$  be granted?
- Can request for (0,2,0) by  $P_0$  be granted?



# Deadlock Detection

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

End of Chapter 7