Operating Systems

Chapter 7: Deadlocks

Dr. Ahmed Hagag

Scientific Computing Department,
Faculty of Computers and Artificial Intelligence
Benha University

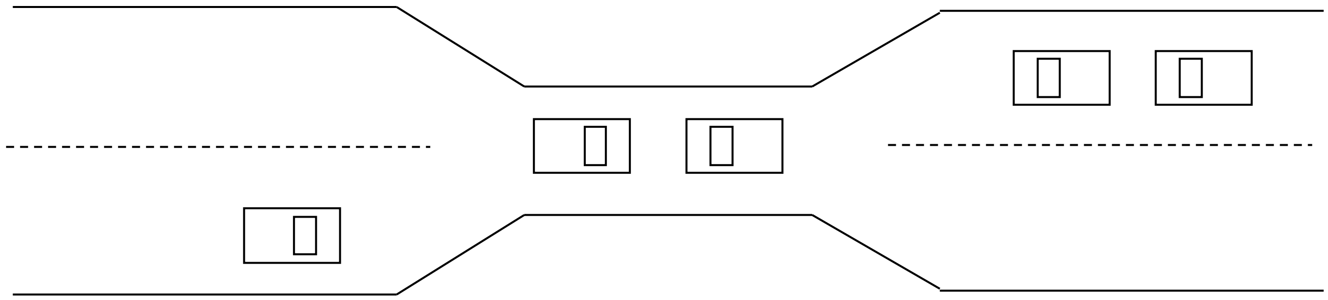
2019

Ch. 7: Deadlocks

- Introduction to Deadlock.
- Deadlock Characterization.
- Methods for Handling Deadlocks.
- Deadlocks Avoidance.
- Deadlocks Detection and Recovery.

Introduction (1/3)

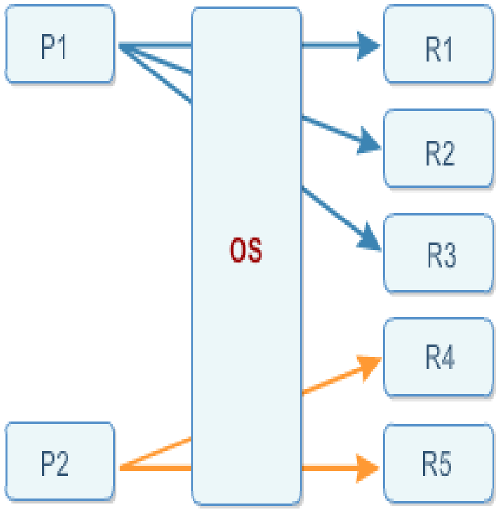
- Generally speaking, deadlock, involves conflicting needs for resources by two or more request orders. A common example is a traffic deadlock.
  - If deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
  - Several cars may have to back up if deadlock occurs.
  - Starvation is possible.



Introduction (2/3)

Deadlock in computer system (1/2)

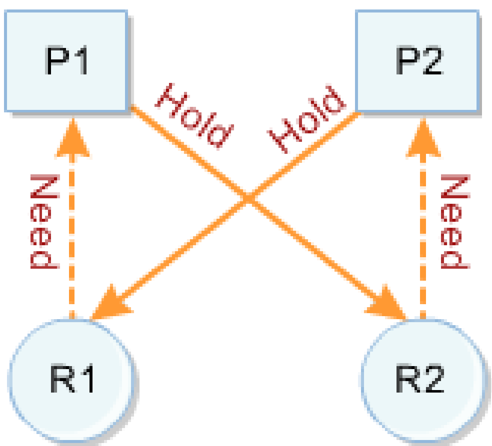
- A computer system consists of a **finite number of resources** to be distributed among a number of competing processes.
- An operating system is a **resource allocator** i.e., there are many resources that can be allocated to only one process at a time.
- Each process utilizes a resource as follows
  - **request**
  - **use**
  - **release**

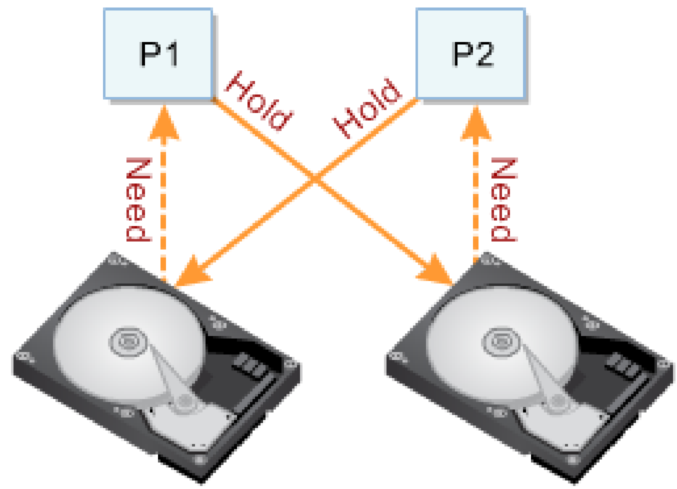


Introduction (2/3)

Deadlock in computer system (2/2)

- General example of deadlock in a computer system:





Introduction (3/3)

Types of resources in computer (1/5)

- Resource types $R_1, R_2, \ldots, R_m$

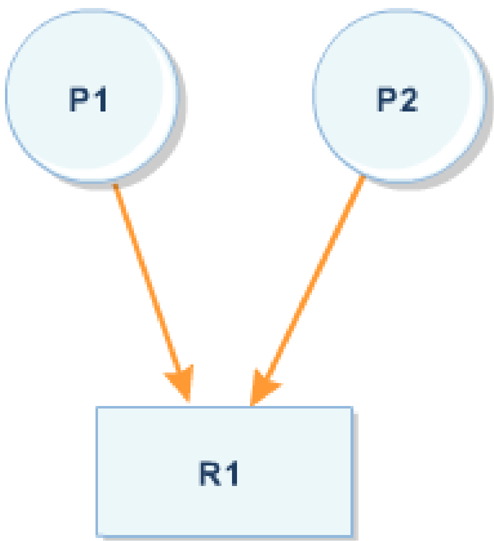
  CPU cycles, memory space, I/O devices

- Each resource type R_i has W_i instances.

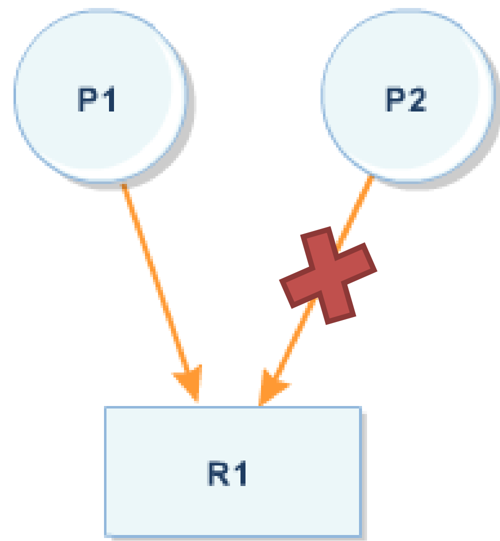
Introduction (3/3)

Types of resources in computer (2/5)

- **Sharable resources** can be used by more than one process at a time. A **consumable resource** can only be used by one process.



Sharable resources



Consumable resources

Introduction (3/3)

Types of resources in computer (3/5)

- Resources can be pre-emptable or non pre-emptable.
  - Memory is an example of a **pre-emptable** resource, but

  - A printer is a **non-preemptable** one.

Introduction (3/3)

Types of resources in computer (4/5)

- **Reusable**: used by one process at a time and then returned.
  - ➢ Processors, I/O channels, main and secondary memory, files, databases, and semaphores.
  - ➢ Deadlock may occur if each process holds one resource and requests another.

Introduction (3/3)

Types of resources in computer (5/5)

- **Consumable**: created (produced) and destroyed (consumed) by a process.
  - Interrupts, signals, messages, data in I/O buffers.
  - Deadlock may occur if receive_message() is blocking.

Deadlock Characterization (1/11)

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion.**
- **Hold and wait.**
- **No preemption.**
- **Circular wait.**

Deadlock Characterization (2/11)

Deadlock can arise if four conditions hold simultaneously.

1. **Mutual exclusion**: only one process at a time can use a resource.
2. **Hold and wait**: a process holding at least one resource is waiting to acquire additional resources held by other processes.

Deadlock Characterization (3/11)

Deadlock can arise if four conditions hold simultaneously.

3. **No preemption**: a resource can be released only willingly by the process holding it, after that process has completed its task.

4. **Circular wait**: there exists a set $\{P_0, P_1, \ldots, P_n\}$ 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.

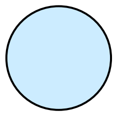
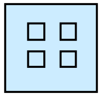
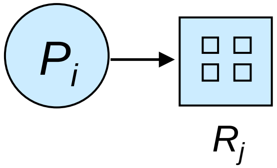
Deadlock Characterization (4/11)

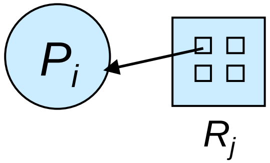
Resource-Allocation Graph (1/2)

- A set of vertices $\boldsymbol{V}$ and a set of edges $\boldsymbol{E}$.
- $\boldsymbol{V}$ is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all processes in the system.
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.
- **Request edge** – directed edge $P_i \rightarrow R_j$
- **Assignment edge** – directed edge $R_j \rightarrow P_i$

Deadlock Characterization (4/11)

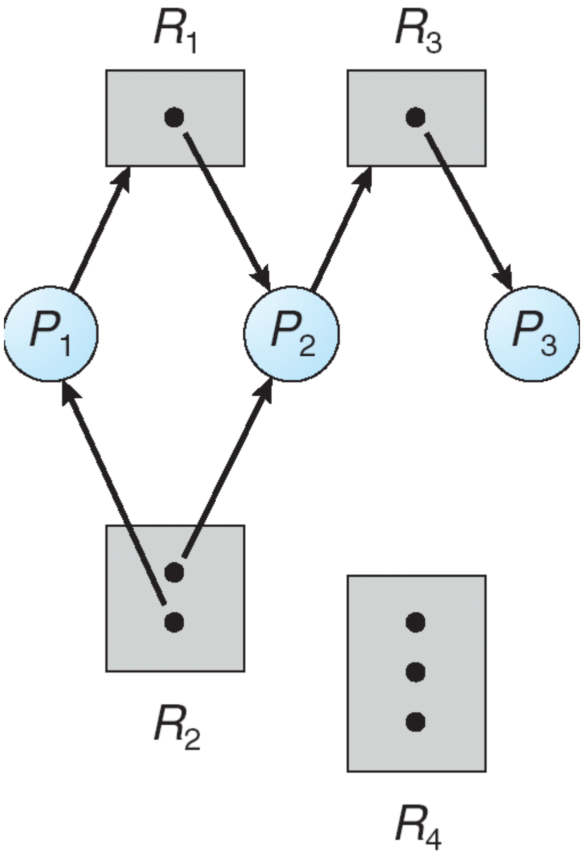
Resource-Allocation Graph (2/2)

- Process 
- Resource Type with 4 instances 
- P_i requests instance of R_j 

- P_i is holding an instance of R_j 

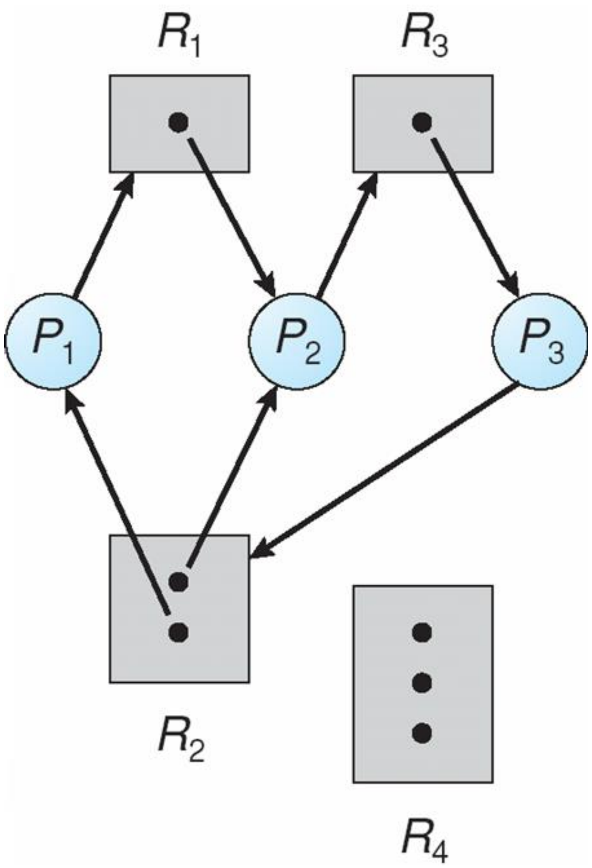
Deadlock Characterization (5/11)

Example of a Resource Allocation Graph



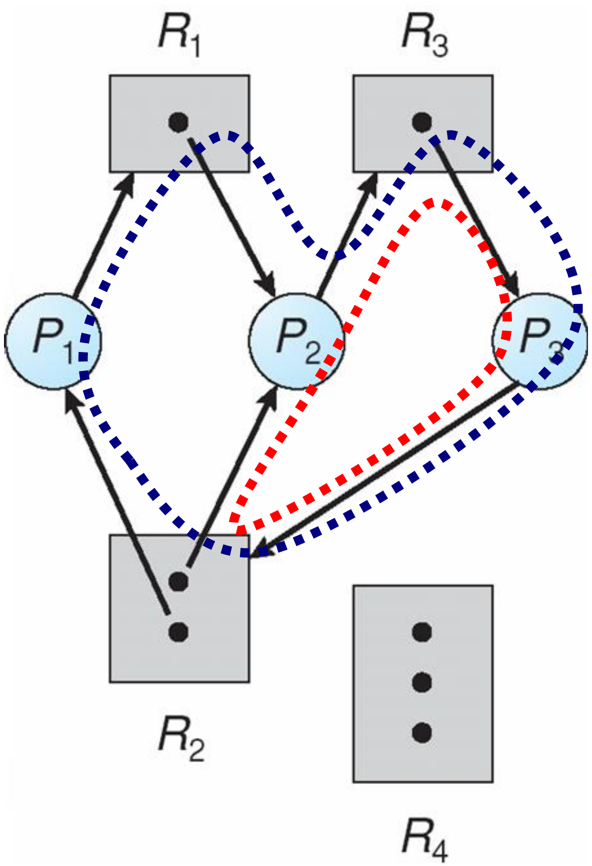
Deadlock Characterization (6/11)

Resource Allocation Graph With A Deadlock



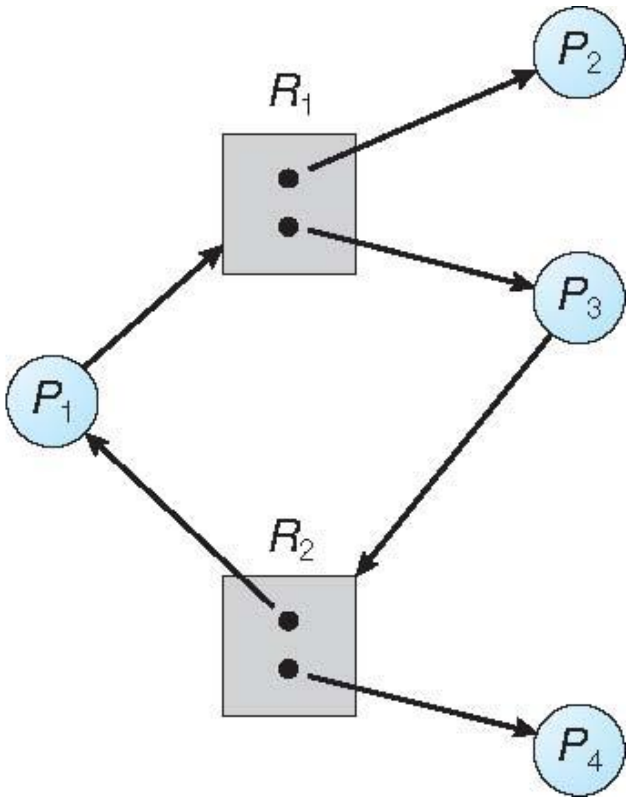
Deadlock Characterization (6/11)

Resource Allocation Graph With A Deadlock



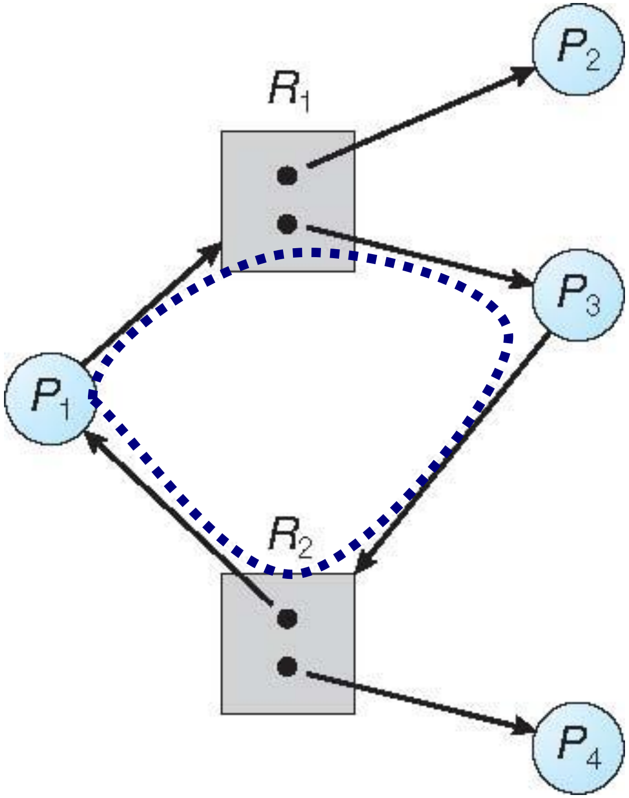
Deadlock Characterization (7/11)

Graph With A Cycle But No Deadlock



Deadlock Characterization (7/11)

Graph With A Cycle But No Deadlock



Deadlock Characterization (8/11)

Basic Facts

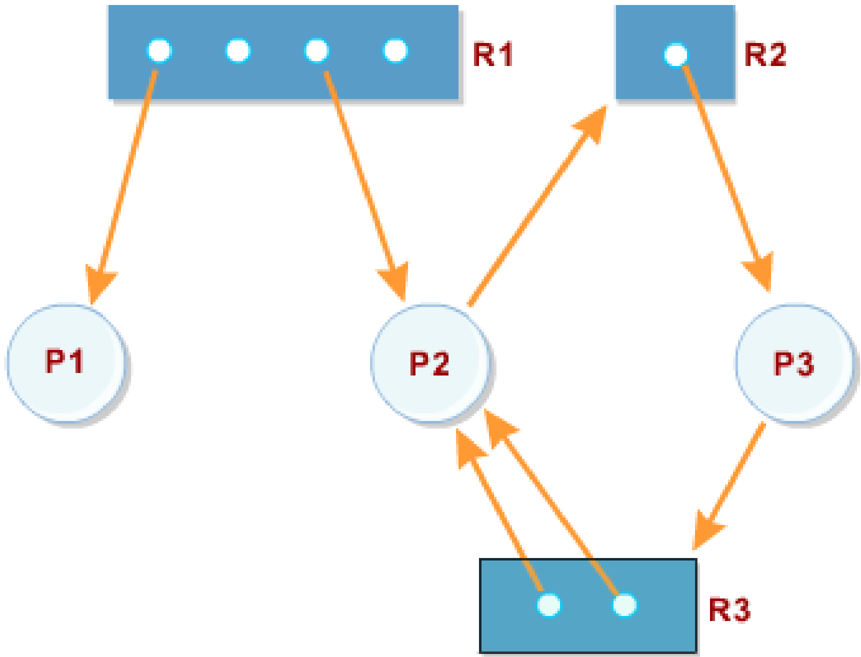
If graph contains **no cycles** ⇒ **no deadlock.**

If graph contains a **cycle** ⇒

- if only one instance per resource type, then deadlock.
- if several instances per resource type, possibility of deadlock.

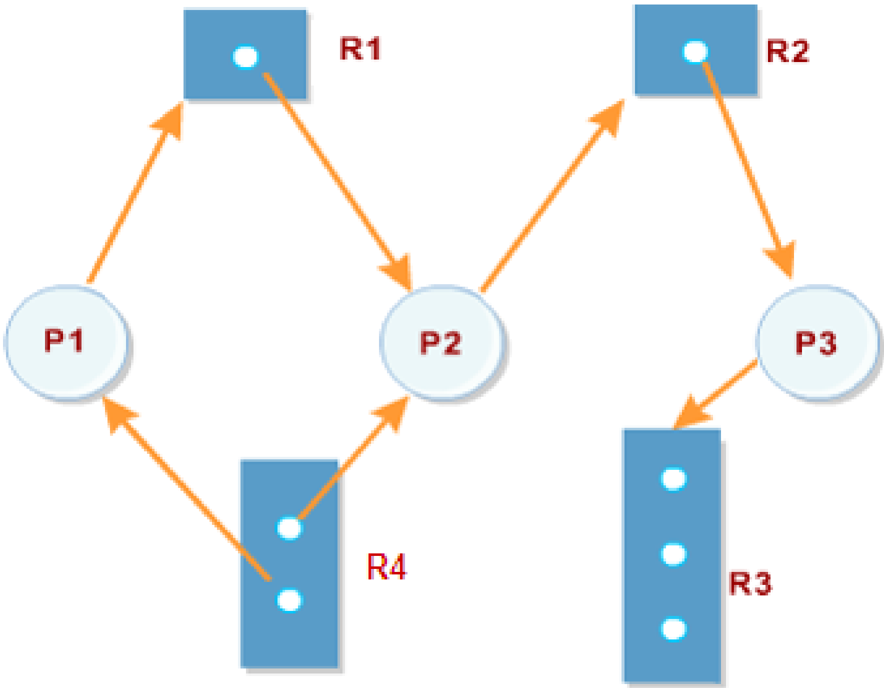
Deadlock Characterization (9/11)

With a Deadlock or Without ??



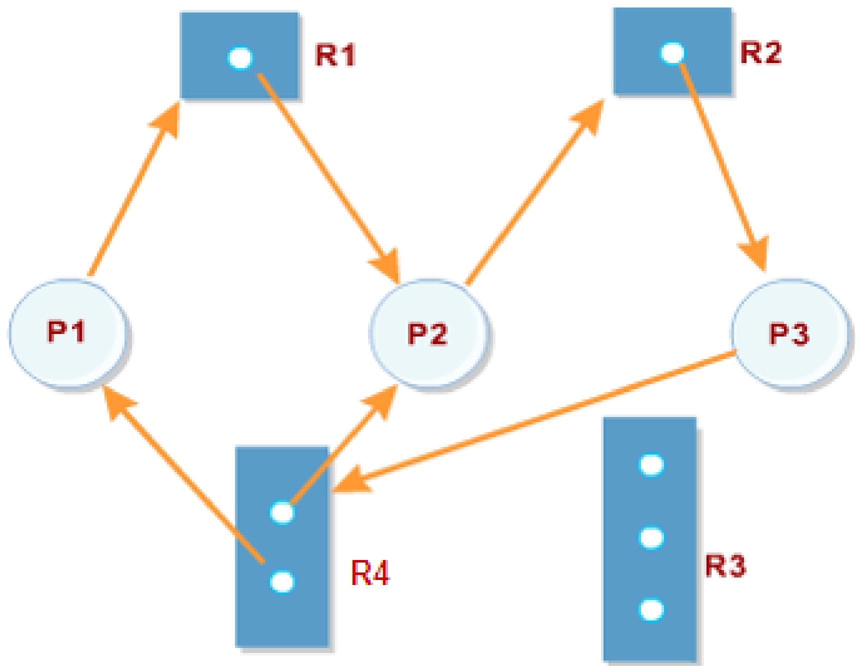
Deadlock Characterization (10/11)

With a Deadlock or Without ??



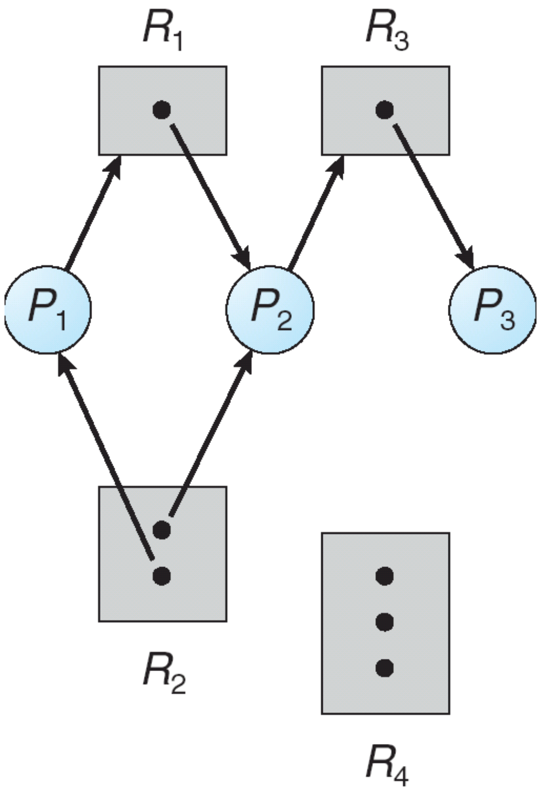
Deadlock Characterization (11/11)

With a Deadlock or Without ??



Example Deadlocks (1/10)

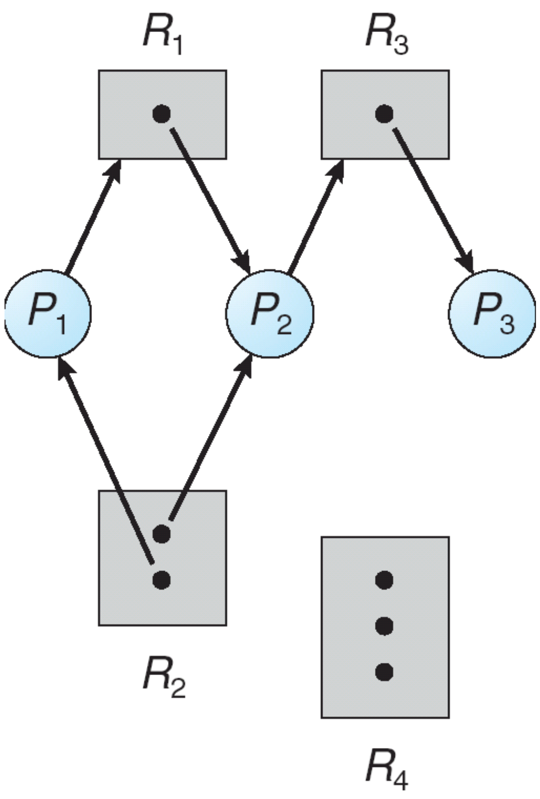
1. Describe the following Resource-Allocation Graph.
2. Is this graph contain a cycle ?
3. Is there a deadlock? Why?



Example Deadlocks (2/10)

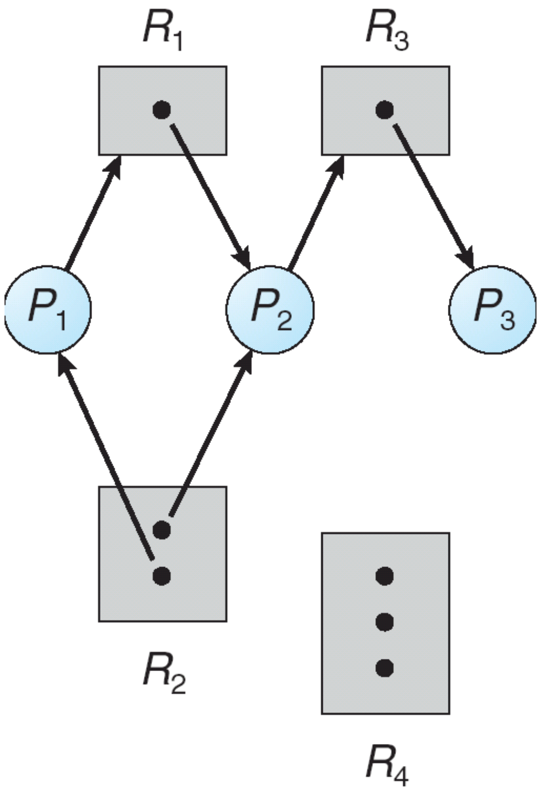
1. Describe the following Resource-Allocation Graph.

- There are 3 processes P_1, P_2, P_3
- There are 4 resources:
  - R_1 (1 instance),
  - R_2 (2 instances),
  - R_3 (1 instance),
  - R_4 (3 instances).
- P_1 holding 1 instance from R_2
- P_1 request 1 instance from R_1
- P_2 holding 1 instance from R_2
- P_2 holding 1 instance from R_1
- P_2 request 1 instance from R_3
- P_3 holding 1 instance from R_3



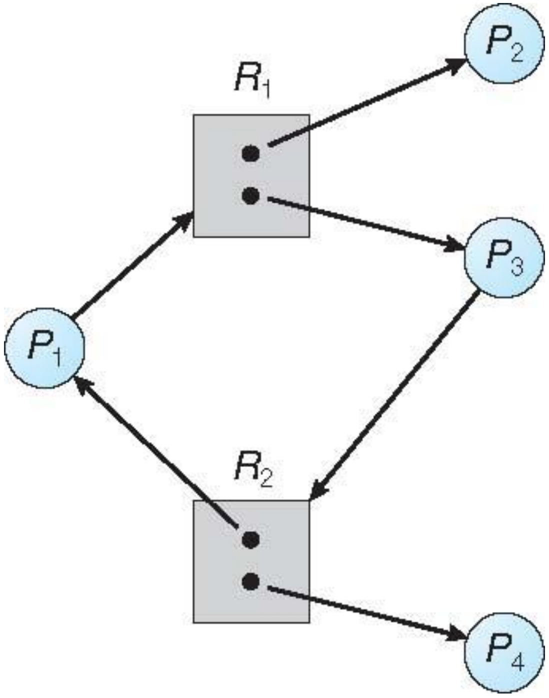
Example Deadlocks (3/10)

2. Is this graph contain a cycle ?
   - No cycle.
3. Is there a deadlock? Why?
   - No deadlock. Because there is no cycle.



Example Deadlocks (4/10)

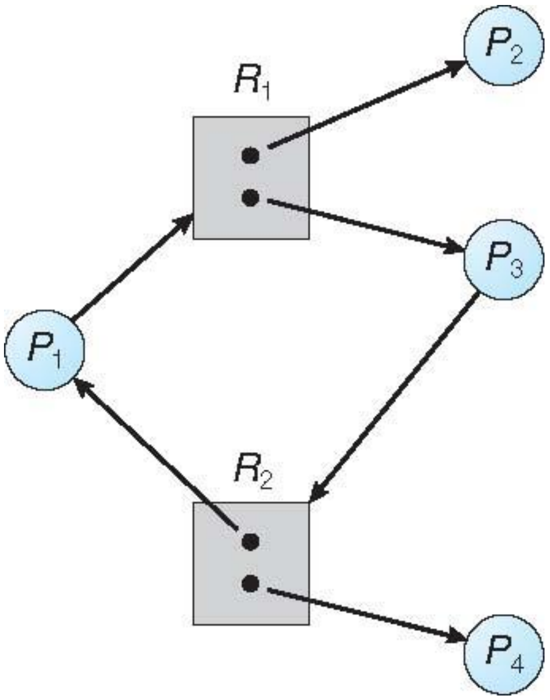
1. Describe the following Resource-Allocation Graph.
2. Is this graph contain a cycle ?
3. Is there a deadlock? Why?



Example Deadlocks (5/10)

1. Describe the following Resource-Allocation Graph.

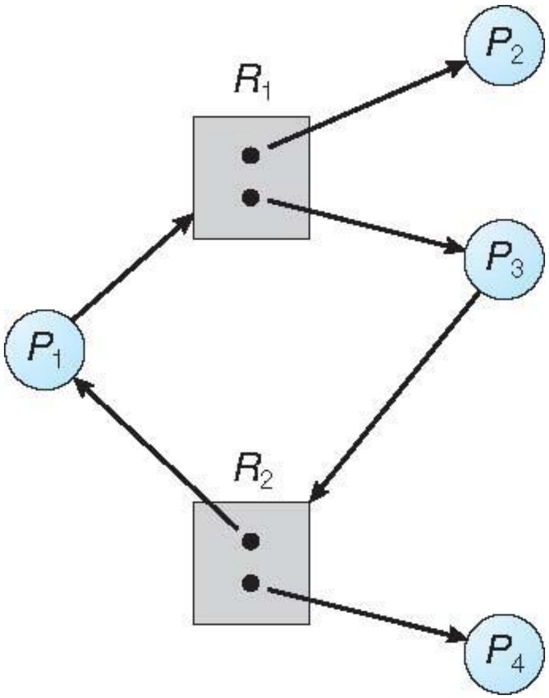
- There are 4 processes P_1, P_2, P_3, P_4
- There are 2 resources:
  - ➢ R_1(2 instances),
  - ➢ R_2(2 instances).
- P_1 holding 1 instance from R_2
- P_1 request 1 instance from R_1
- P_2 holding 1 instance from R_1
- P_3 holding 1 instance from R_1
- P_3 request 1 instance from R_2
- P_4 holding 1 instance from R_2



Example Deadlocks (6/10)

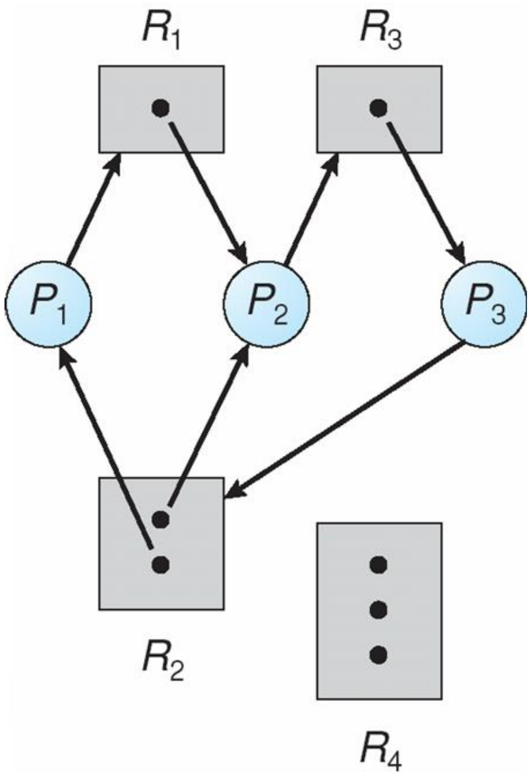
2. Is this graph contain a cycle ?
   - There is one cycle $< R_2, P_1, R_1, P_3, R_2 >$.

3. Is there a deadlock? Why?
   - No deadlock.
   - Because P_2 and P_4 will finish their jobs over time.
   - Then, 1 instance from R_1 and 1 instance from R_2 will be free for P_1 and P_3.



Example Deadlocks (7/10)

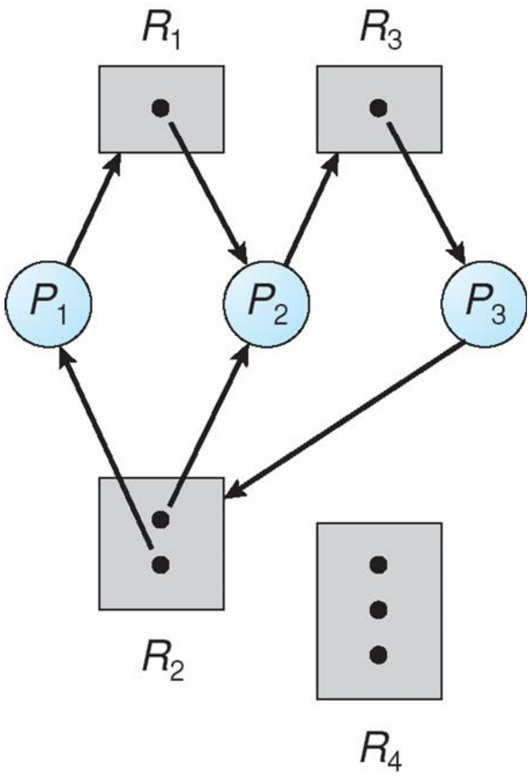
1. Describe the following Resource-Allocation Graph.
2. Is this graph contain a cycle ?
3. Is there a deadlock? Why?



Example Deadlocks (8/10)

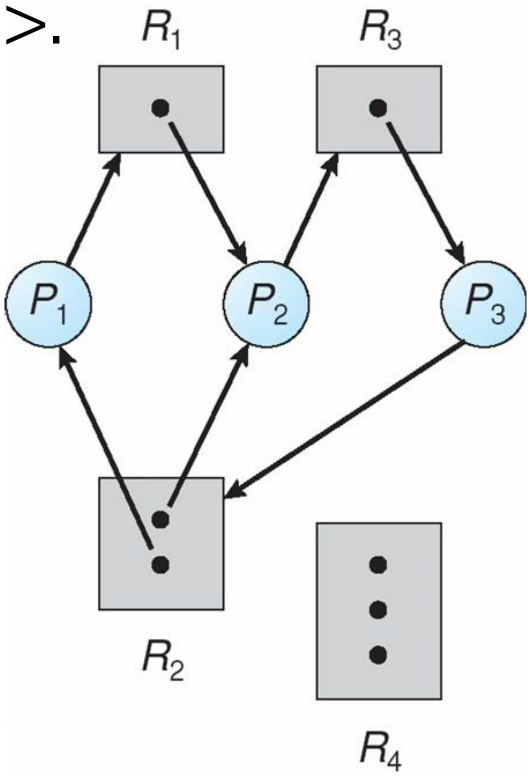
1. Describe the following Resource-Allocation Graph.

- There are 3 processes P_1, P_2, P_3
- There are 4 resources:
  - ➢ R_1(1 instance),
  - ➢ R_2(2 instances),
  - ➢ R_3(1 instance),
  - ➢ R_4(3 instances).
- P_1 holding 1 instance from R_2
- P_1 request 1 instance from R_1
- P_2 holding 1 instance from R_2
- P_2 holding 1 instance from R_1
- P_2 request 1 instance from R_3
- P_3 holding 1 instance from R_3
- P_3 request 1 instance from R_2



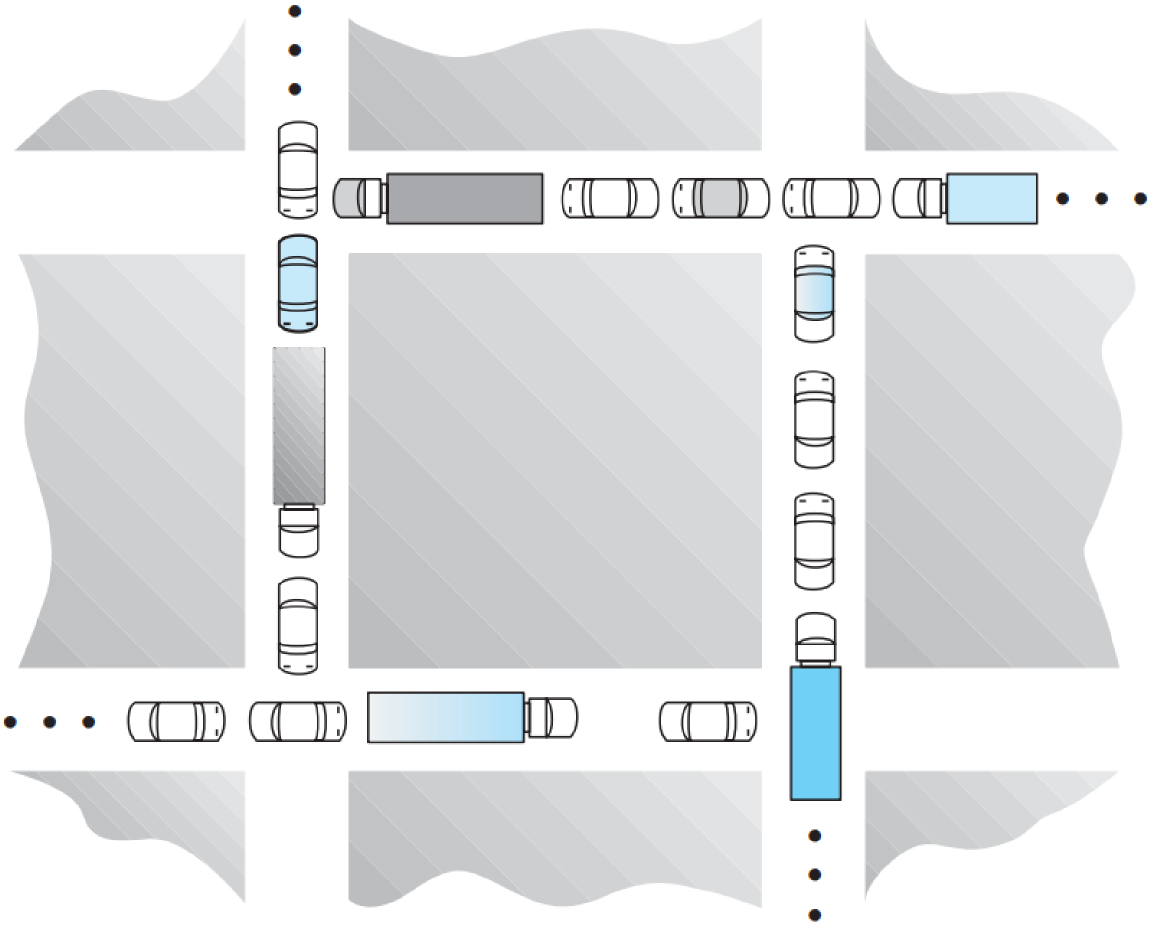
Example Deadlocks (9/10)

2. Is this graph contain a cycle ?
   - There is one cycle $< R_2, P_1, R_1, P_2, R_3, P_3, R_2 >$.



3. Is there a deadlock? Why?
   - Yes there is a deadlock.
   - Because P_1 is waiting for P_2 and,
   - P_2 is waiting for P_3 and,
   - P_3 is waiting for P_1 and P_2. Over time.

Example Deadlocks (10/10)



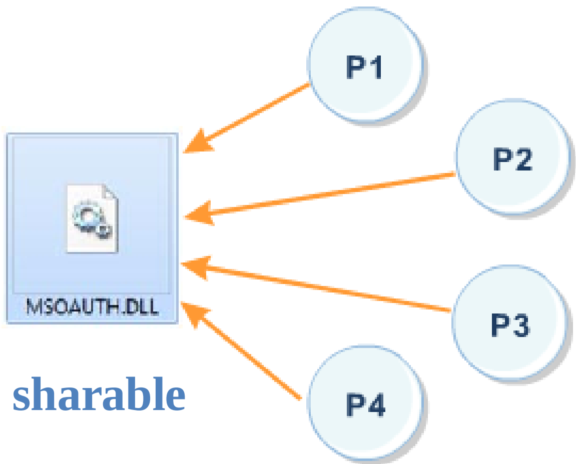
Methods for Handling Deadlocks

- Ensure that the system will ***never*** enter a deadlock state:
  - Deadlock **prevention.**
  - Deadlock **avoidance.**
- 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.

Deadlock Prevention (1/8)

Mutual Exclusion – cannot be broken

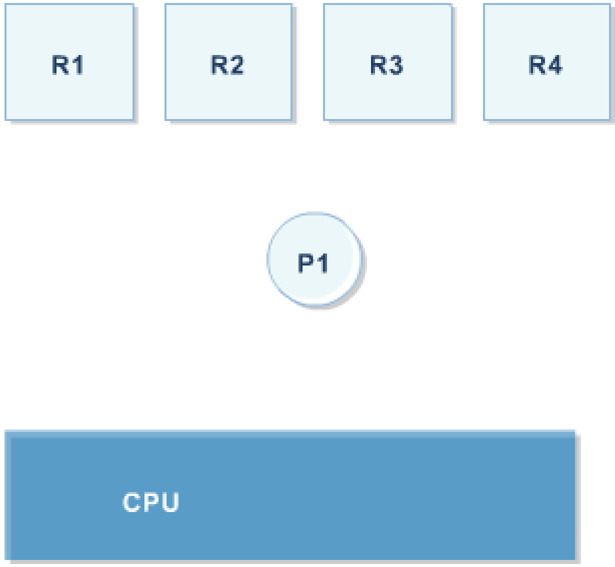
- We **cannot** prevent deadlocks by denying mutual exclusion because some resources are non-sharable.
- Sharable resources (e.g., read-only files) can be accessed concurrently.



Deadlock Prevention (2/8)

Hold and Wait – can be broken if

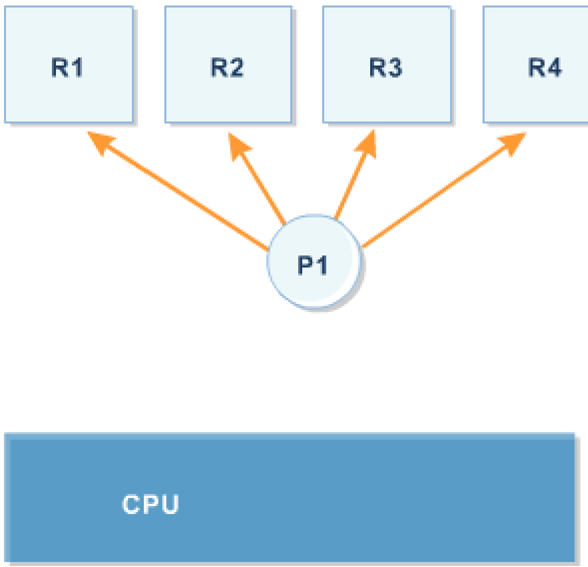
- A process requests a resource only if it does not hold any other resources.
- A process requests and is allocated all its resources before it begins execution.



Deadlock Prevention (2/8)

Hold and Wait – can be broken if

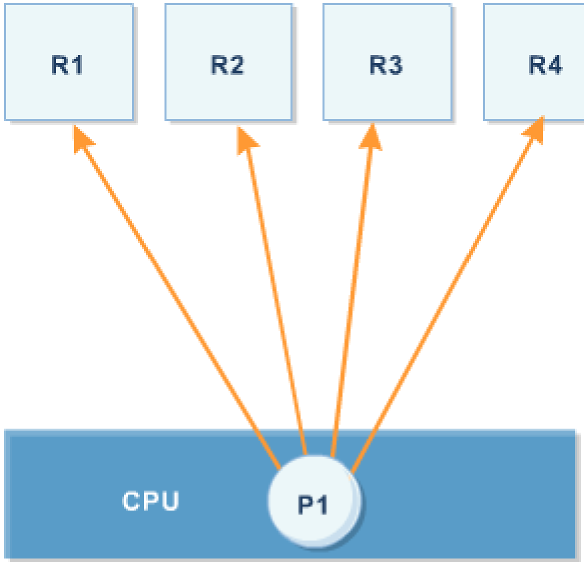
- A process requests a resource only if it does not hold any other resources.
- A process requests and is allocated all its resources before it begins execution.



Deadlock Prevention (2/8)

Hold and Wait – can be broken if

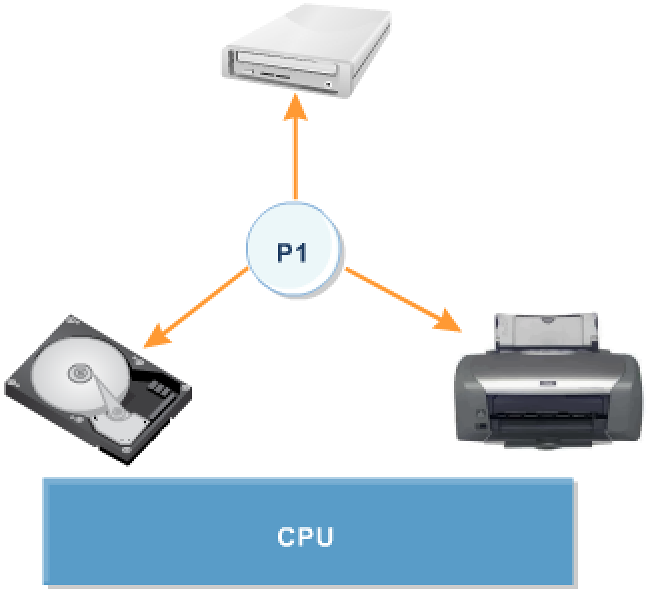
- A process requests a resource only if it does not hold any other resources.
- A process requests and is allocated all its resources before it begins execution.



Deadlock Prevention (3/8)

Hold and Wait – can be broken if

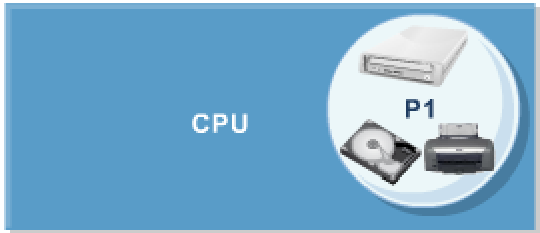
- A process requests a resource only if it does not hold any other resources.
- A process requests and is allocated all its resources before it begins execution.



Deadlock Prevention (3/8)

Hold and Wait – can be broken if

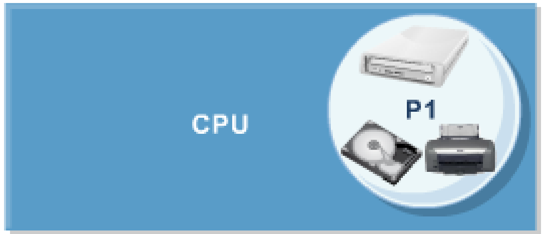
- A process requests a resource only if it does not hold any other resources.
- A process requests and is allocated all its resources before it begins execution.



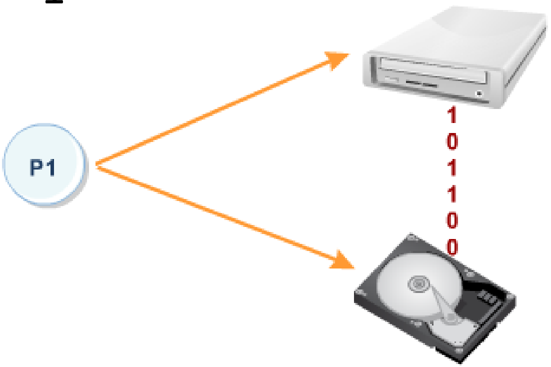
Deadlock Prevention (3/8)

Hold and Wait – can be broken if

- A process requests a resource only if it does not hold any other resources.
- A process requests and is allocated all its resources before it begins execution.



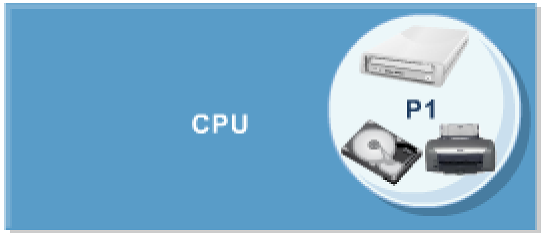
1



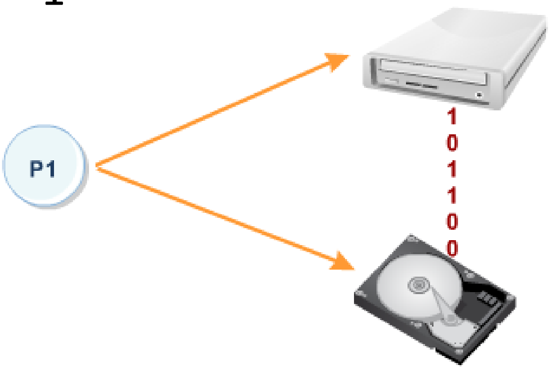
Deadlock Prevention (3/8)

Hold and Wait – can be broken if

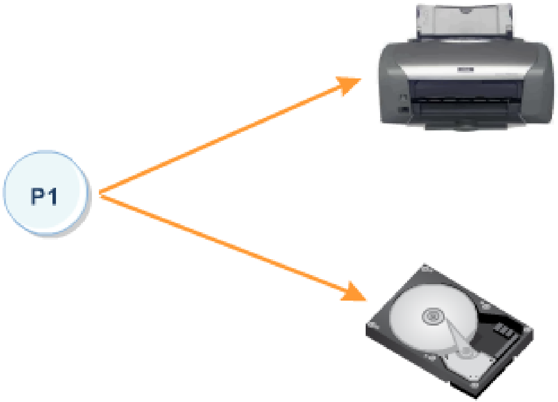
- A process requests a resource only if it does not hold any other resources.
- A process requests and is allocated all its resources before it begins execution.



1



2



Deadlock Prevention (4/8)

Hold and Wait – can be broken if

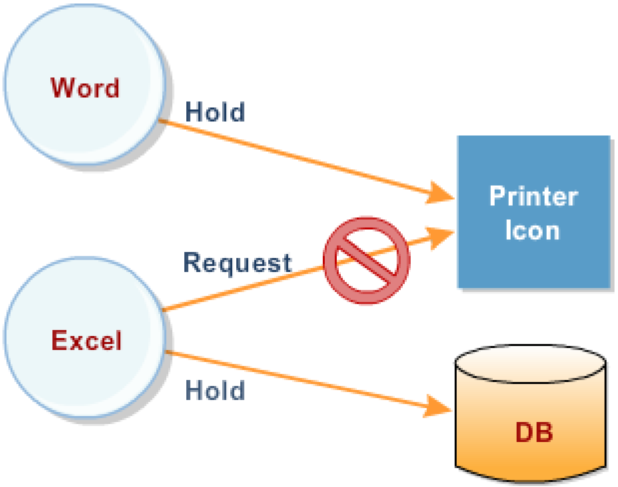
- A process requests a resource only if it does not hold any other resources.
- A process requests and is allocated all its resources before it begins execution.

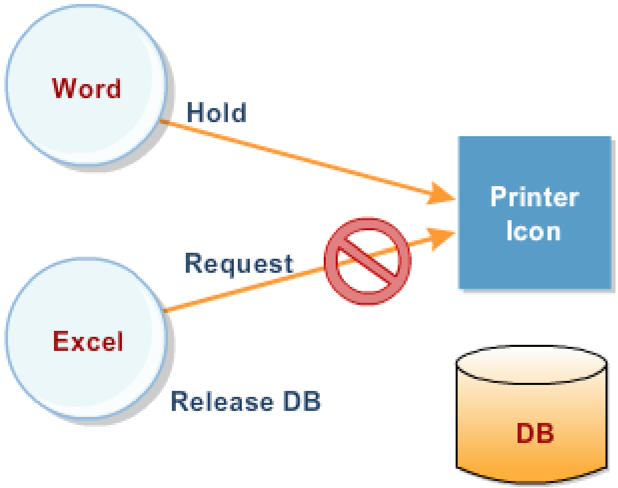
- **Disadvantages:**
  - Low resource utilization; starvation possible.

Deadlock Prevention (5/8)

No Preemption – can be broken if

- If a process holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.

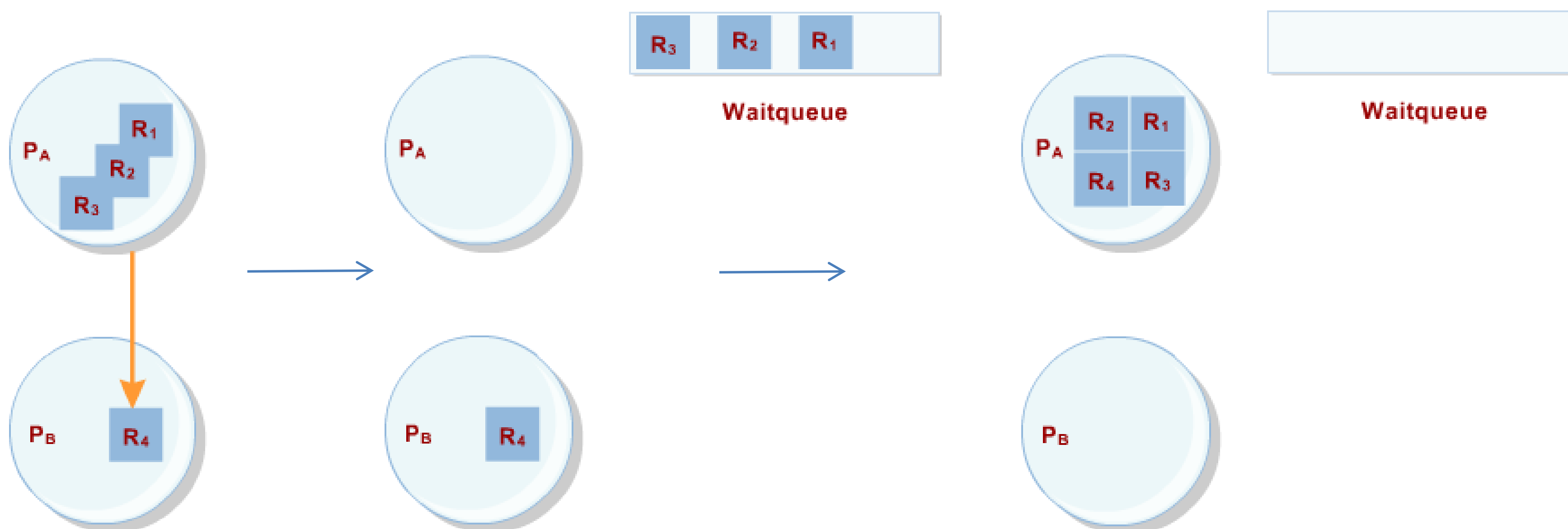




Deadlock Prevention (6/8)

No Preemption – can be broken if

- 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.



Deadlock Prevention (7/8)

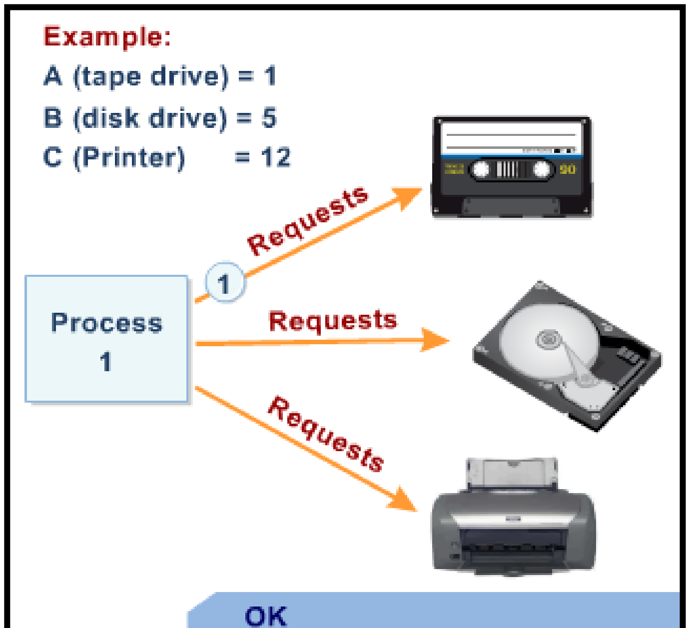
No Preemption – can be broken if

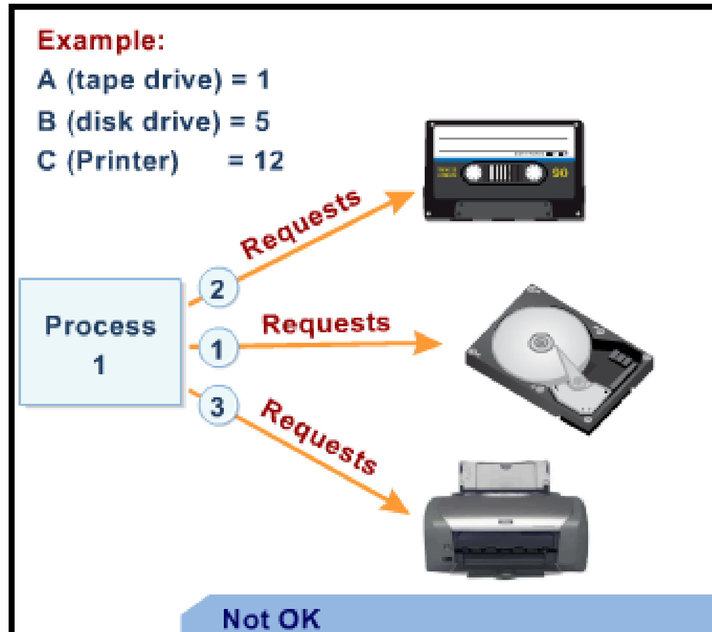
- **Problems?**
- Difficult to use with resources whose state are not easily saved, e.g., printers and tape drives. (In contrast to CPU registers and memory space).

Deadlock Prevention (8/8)

Circular Wait – can be broken if

- Impose a total ordering on all resource types, and
- Require that each process requests resources in an increasing order of enumeration.





Deadlocks Avoidance (1/29)

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.

Deadlocks Avoidance (2/29)

- 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 <P_1, P_2, …, P_n> 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 $j < i$.

Deadlocks Avoidance (3/29)

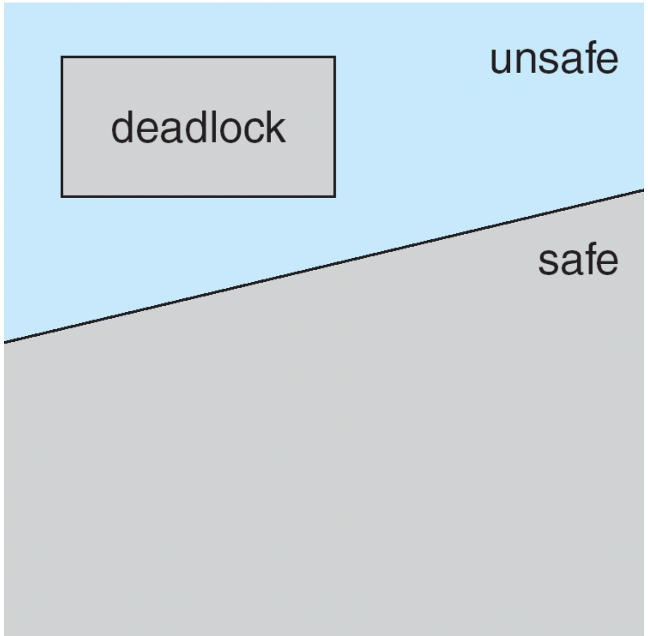
<u>That is:</u>

- 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.

Deadlocks Avoidance (4/29)

Basic Facts

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



Deadlocks Avoidance (5/29)

Avoidance Algorithms:

- Single instance of a resource type $\Rightarrow$
  - ➢ Use a resource-allocation graph.
- Multiple instances of a resource type $\Rightarrow$
  - ➢ Use the banker’s algorithm.

Deadlocks Avoidance (6/29)

Resource-Allocation Graph:

- **Claim edge** $P_i \rightarrow R_j$ indicated that process P_i may request resource R_j; represented by a dashed line.

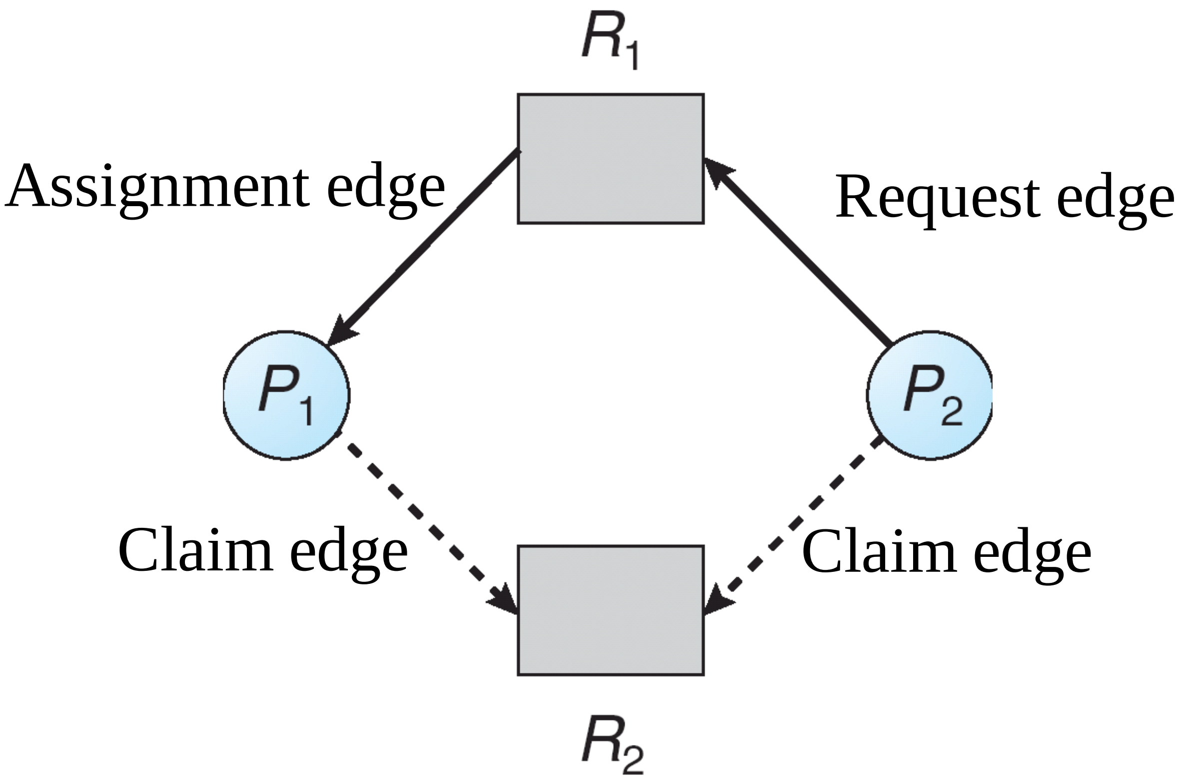
- Claim edge converts to request edge when a process requests a resource.

Deadlocks Avoidance (7/29)

- Request edge converted to an assignment edge when the resource is allocated to the process.
- When a resource is released by a process, assignment edge reconverts to a claim edge.

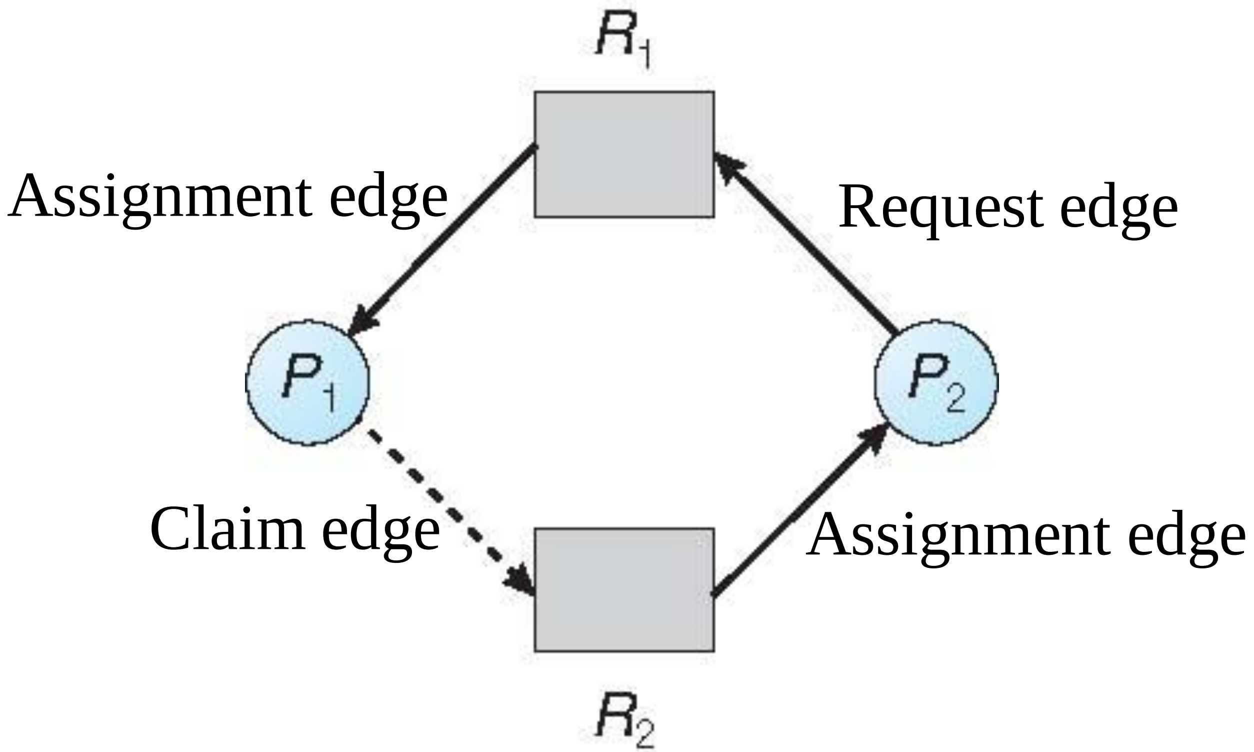
Deadlocks Avoidance (8/29)

Resource-Allocation Graph:



Deadlocks Avoidance (9/29)

Unsafe State In Resource-Allocation Graph:



A cycle, as mentioned, indicates that the system is in an unsafe state. If P_1 requests R_2, and P_2 holding R_2, then a deadlock will occur.

Deadlocks Avoidance (10/29)

<u>Resource-Allocation Graph Algorithm:</u>

- Suppose that process P_i requests resource R_j
- The request can be granted only if converting the request edge to assignment edge does **not create a cycle** in the resource allocation graph.

Deadlocks Avoidance (11/29)

Banker's Algorithm:

- Multiple instances of resources.
- 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.

Deadlocks Avoidance (12/29)

<u>Data Structures for the Banker's Algorithm:</u>

Let n = number of processes, and m = number of resources types.

- **Available**: Vector of length m. If available $[j] = k$, there are k instances of resource type R_j available
- **Max**: $n \times m$ matrix. If $Max\,[i,j] = k$, then process P_i may request at most k instances of resource type R_j
- **Allocation**: $n \times m$ matrix. If Allocation$[i,j] = k$ then P_i is currently allocated k instances of R_j
- **Need**: $n \times m$ matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task

$$Need\,[i,j] = Max[i,j] - Allocation\,[i,j]$$

Deadlocks Avoidance (13/29)

Banker's Algorithm (Safety Algorithm):

1. Let ***Work*** and ***Finish*** be vectors of length *m* and *n*, respectively.
   Initialize:
   Work = Available
   ***Finish [i] = false* for *i* = 0, 1, …, *n*- 1**

2. Find an ***i*** such that both:
   (a) ***Finish [i] = false***
   (b) ***Need****$_i$* **≤ *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] == true*** for all ***i***, then the system is in a safe state.

Deadlocks Avoidance (14/29)

Banker's Algorithm (Resource-Request Algorithm):

$\boldsymbol{Request_i}$ = request vector for process $\boldsymbol{P_i}$. If $\boldsymbol{Request_i}$ **[*j*] = *k*** then process $\boldsymbol{P_i}$ wants ***k*** instances of resource type $\boldsymbol{R_j}$

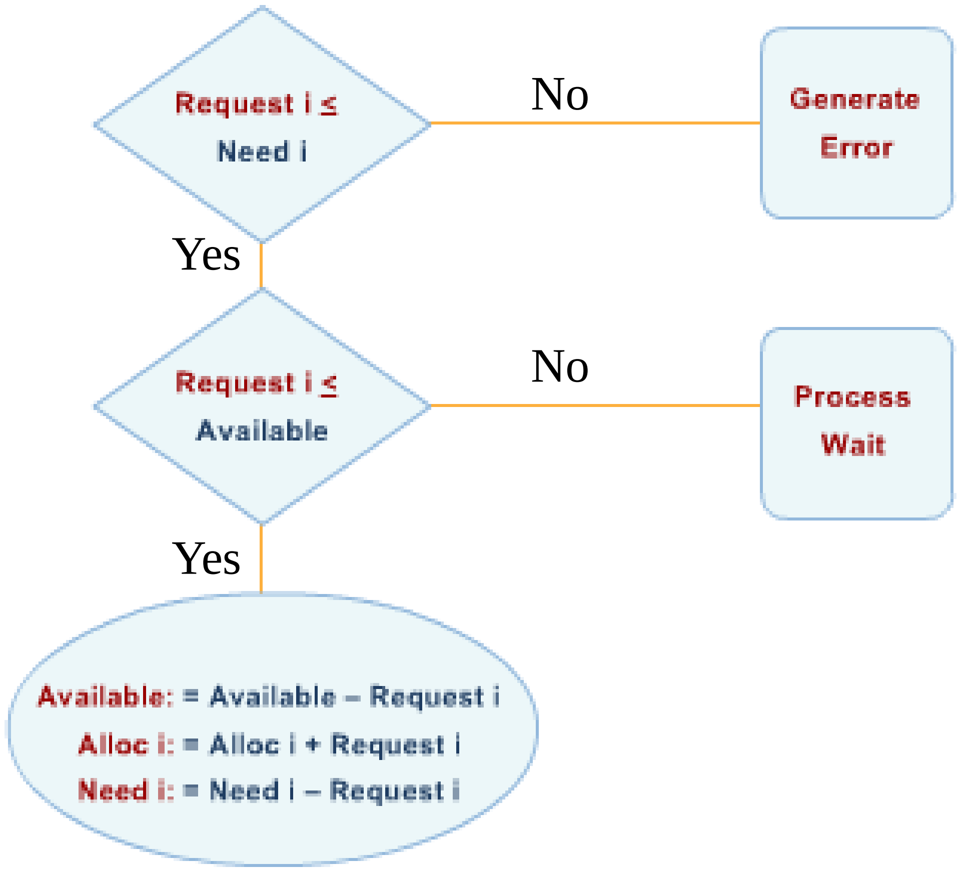
1. If $\boldsymbol{Request_i \le Need_i}$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $\boldsymbol{Request_i \le Available}$, go to step 3. Otherwise $\boldsymbol{P_i}$ must wait, since resources are not available.
3. Pretend to allocate requested resources to $\boldsymbol{P_i}$ by modifying the state as follows:

$$\boldsymbol{Available = Available - Request_i;}$$
$$\boldsymbol{Allocation_i = Allocation_i + Request_i;}$$
$$\boldsymbol{Need_i = Need_i - Request_i;}$$

- If safe $\Rightarrow$ the resources are allocated to $\boldsymbol{P_i}$.
- If unsafe $\Rightarrow$ $\boldsymbol{P_i}$ must wait, and the old resource-allocation state is restored.

Deadlocks Avoidance (15/29)

Banker's Algorithm (Resource-Request Algorithm):



Deadlocks Avoidance (16/29)

Banker's Algorithm (Example):

Consider the following snapshot of a system:

	Allocation	*Max*	*Available*
	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	

a. What is the content of the matrix ***Need*** ?

b. Is the system in a safe state? Why?

Deadlocks Avoidance (17/29)

Banker's Algorithm (Example):

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

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
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		

Deadlocks Avoidance (17/29)

Banker's Algorithm (Example):

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

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 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		

Deadlocks Avoidance (17/29)

Banker's Algorithm (Example):

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

	Allocation	Max	Available	Need
	$A\ B\ C$	$A\ B\ C$	$A\ B\ C$	$A\ B\ C$
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (18/29)

Banker's Algorithm (Example):

b. Is the system in a safe state? Why?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (18/29)

Banker's Algorithm (Example):

b. Is the system in a safe state? Why?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (18/29)

Banker's Algorithm (Example):

b. Is the system in a safe state? Why?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	$\boxed{3\ 3\ 2}$	7 4 3
P_1	2 0 0	3 2 2		$\boxed{1\ 2\ 2}$
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (18/29)

Banker's Algorithm (Example):

b. Is the system in a safe state? Why?

		Allocation	*Max*	*Available*	*Need*
		A B C	*A B C*	*A B C*	*A B C*
	P_0	0 1 0	7 5 3	$\boxed{5\ 3\ 2}$	$\boxed{7\ 4\ 3}$
1	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
	P_2	3 0 2	9 0 2		6 0 0
	P_3	2 1 1	2 2 2		0 1 1
	P_4	0 0 2	4 3 3		4 3 1

Banker's Algorithm (Example):

b. Is the system in a safe state? Why?

		Allocation	Max	Available	Need
		A B C	A B C	A B C	A B C
	P_0	0 1 0	7 5 3	$\boxed{5\ 3\ 2}$	7 4 3
1	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
	P_2	3 0 2	9 0 2		$\boxed{6\ 0\ 0}$
	P_3	2 1 1	2 2 2		0 1 1
	P_4	0 0 2	4 3 3		4 3 1

Banker's Algorithm (Example):

b. Is the system in a safe state? Why?

		Allocation	*Max*	*Available*	*Need*
		A B C	*A B C*	*A B C*	*A B C*
	P_0	0 1 0	7 5 3	5 3 2	7 4 3
1	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
	P_2	3 0 2	9 0 2		6 0 0
	P_3	2 1 1	2 2 2		0 1 1
	P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (18/29)

Banker's Algorithm (Example):

b. Is the system in a safe state? Why?

		Allocation	*Max*	*Available*	*Need*
		A B C	*A B C*	*A B C*	*A B C*
	P_0	0 1 0	7 5 3	$\boxed{7\ 4\ 3}$	$\boxed{7\ 4\ 3}$
1	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
	P_2	3 0 2	9 0 2		6 0 0
2	~~P_3~~	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
	P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (18/29)

Banker's Algorithm (Example):

b. Is the system in a safe state? Why?

		Allocation	*Max*	*Available*	*Need*
		A B C	*A B C*	*A B C*	*A B C*
3	~~P_0~~	~~0 1 0~~	~~7 5 3~~	7 5 3	~~7 4 3~~
1	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
	P_2	3 0 2	9 0 2		6 0 0
2	~~P_3~~	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
	P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (18/29)

Banker's Algorithm (Example):

b. Is the system in a safe state? Why?

		Allocation	*Max*	*Available*	*Need*
		A B C	*A B C*	*A B C*	*A B C*
3	P_0	~~0 1 0~~	~~7 5 3~~	$\boxed{10\ 5\ 5}$	~~7 4 3~~
1	P_1	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
4	P_2	~~3 0 2~~	~~9 0 2~~		~~6 0 0~~
2	P_3	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
	P_4	0 0 2	4 3 3		$\boxed{4\ 3\ 1}$

Banker's Algorithm (Example):

b. Is the system in a safe state? Why?

Order	Process	*Allocation* A B C	*Max* A B C	*Available* A B C	*Need* A B C
3	~~P_0~~	~~0 1 0~~	~~7 5 3~~	$\boxed{10\ 5\ 7}$	~~7 4 3~~
1	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
4	~~P_2~~	~~3 0 2~~	~~9 0 2~~		~~6 0 0~~
2	~~P_3~~	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
5	~~P_4~~	~~0 0 2~~	~~4 3 3~~		~~4 3 1~~

Banker's Algorithm (Example):

b. Is the system in a safe state? Why?

		Allocation	*Max*	*Available*	*Need*
		A B C	*A B C*	*A B C*	*A B C*
3	~~P_0~~	~~0 1 0~~	~~7 5 3~~	$\boxed{10\ 5\ 7}$	~~7 4 3~~
1	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
4	~~P_2~~	~~3 0 2~~	~~9 0 2~~		~~6 0 0~~
2	~~P_3~~	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
5	~~P_4~~	~~0 0 2~~	~~4 3 3~~		~~4 3 1~~

The system is in a **safe state** since the sequence < P_1, P_3, P_0, P_2, P_4> **satisfies safety criteria**.

Deadlocks Avoidance (19/29)

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for (3,0,0), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (19/29)

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for (3,0,0), can the request be granted immediately?

	Allocation	Max	Available	Need
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (20/29)

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for (3,0,0), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (21/29)

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for (3,0,0), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	0 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	6 0 2	9 0 2		3 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (22/29)

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for $\boxed{(3,0,0)}$, can the request be granted immediately?

	Allocation A B C	*Max* A B C	*Available* A B C	*Need* A B C
P_0	0 1 0	7 5 3	$\boxed{0\ 3\ 2}$	$\boxed{7\ 4\ 3}$
P_1	2 0 0	3 2 2		1 2 2
P_2	6 0 2	9 0 2		3 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (22/29)

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for (3,0,0), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	0 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	6 0 2	9 0 2		3 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (22/29)

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for (3,0,0), can the request be granted immediately?

	Allocation	Max	Available	Need
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	0 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	6 0 2	9 0 2		3 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for $\boxed{(3,0,0),}$ can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	$\boxed{0\ 3\ 2}$	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	6 0 2	9 0 2		3 0 0
P_3	2 1 1	2 2 2		$\boxed{0\ 1\ 1}$
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (22/29)

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for $\boxed{(3,0,0)}$, can the request be granted immediately?

		Allocation	*Max*	*Available*	*Need*
		A B C	*A B C*	*A B C*	*A B C*
	P_0	0 1 0	7 5 3	$\boxed{2\ 4\ 3}$	$\boxed{7\ 4\ 3}$
	P_1	2 0 0	3 2 2		1 2 2
	P_2	6 0 2	9 0 2		3 0 0
1	~~P_3~~	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
	P_4	0 0 2	4 3 3		4 3 1

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for (3,0,0), can the request be granted immediately?

	Allocation	Max	Available	Need
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	2 4 3	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	6 0 2	9 0 2		3 0 0
1 ~~P_3~~	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
P_4	0 0 2	4 3 3		4 3 1

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for $\boxed{(3,0,0)}$, can the request be granted immediately?

		Allocation	Max	Available	Need
		A B C	A B C	A B C	A B C
	P_0	0 1 0	7 5 3	$\boxed{4\ 4\ 3}$	$\boxed{7\ 4\ 3}$
$\boxed{2}$	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
	P_2	6 0 2	9 0 2		3 0 0
$\boxed{1}$	~~P_3~~	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
	P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (22/29)

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for $\boxed{(3,0,0)}$, can the request be granted immediately?

		Allocation	*Max*	*Available*	*Need*
		A B C	*A B C*	*A B C*	*A B C*
	P_0	0 1 0	7 5 3	$\boxed{4\ 4\ 3}$	7 4 3
2	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
	P_2	6 0 2	9 0 2		$\boxed{3\ 0\ 0}$
1	~~P_3~~	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
	P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (22/29)

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for $\boxed{(3,0,0),}$ can the request be granted immediately?

		Allocation	*Max*	*Available*	*Need*
		A B C	*A B C*	*A B C*	*A B C*
	P_0	0 1 0	7 5 3	$\boxed{10\ 4\ 5}$	$\boxed{7\ 4\ 3}$
2	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
3	~~P_2~~	~~6 0 2~~	~~9 0 2~~		~~3 0 0~~
1	~~P_3~~	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
	P_4	0 0 2	4 3 3		4 3 1

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for $\boxed{(3,0,0)}$, can the request be granted immediately?

		Allocation	Max	Available	Need
		A B C	A B C	A B C	A B C
4	P_0	~~0 1 0~~	~~7 5 3~~	$\boxed{10\ 5\ 5}$	~~7 4 3~~
2	P_1	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
3	P_2	~~6 0 2~~	~~9 0 2~~		~~3 0 0~~
1	P_3	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
	P_4	0 0 2	4 3 3		$\boxed{4\ 3\ 1}$

Deadlocks Avoidance (22/29)

Banker's Algorithm (Example):

c. If a request from process P_2 arrives for $\boxed{(3,0,0)}$, can the request be granted immediately?

		Allocation	*Max*	*Available*	*Need*
		A B C	*A B C*	*A B C*	*A B C*
4	~~P_0~~	~~0 1 0~~	~~7 5 3~~	$\boxed{10\ 5\ 7}$	~~7 4 3~~
2	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
3	~~P_2~~	~~6 0 2~~	~~9 0 2~~		~~3 0 0~~
1	~~P_3~~	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
5	~~P_4~~	~~0 0 2~~	~~4 3 3~~		~~4 3 1~~

Yes you can granted this request immediately, because the system will be in a **safe state** since the sequence $< P_3, P_1, P_2, P_0, P_4 >$ **satisfies safety criteria**.

Deadlocks Avoidance (23/29)

Banker's Algorithm (Example):

d. If a request from process P_0 arrives for (3,2,2), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (23/29)

Banker's Algorithm (Example):

d. If a request from process P_0 arrives for $\boxed{(3,2,2),}$ can the request be granted immediately?

	Allocation A B C	Max A B C	Available A B C	Need A B C
P_0	0 1 0	7 5 3	3 3 2	$\boxed{7\ 4\ 3}$
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (24/29)

Banker's Algorithm (Example):

d. If a request from process P_0 arrives for $\boxed{(3,2,2)}$, can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (24/29)

Banker's Algorithm (Example):

d. If a request from process P_0 arrives for (3,2,2), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (25/29)

Banker's Algorithm (Example):

d. If a request from process P_0 arrives for (3,2,2), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	3 3 2	7 5 3	0 1 0	4 2 1
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (25/29)

Banker's Algorithm (Example):

d. If a request from process P_0 arrives for (3,2,2), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	3 3 2	7 5 3	0 1 0	4 2 1
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (25/29)

Banker's Algorithm (Example):

d. If a request from process P_0 arrives for (3,2,2), can the request be granted immediately?

	Allocation A B C	Max A B C	Available A B C	Need A B C
P_0	3 3 2	7 5 3	0 1 0	4 2 1
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Banker's Algorithm (Example):

d. If a request from process P_0 arrives for (3,2,2), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	3 3 2	7 5 3	0 1 0	4 2 1
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (25/29)

Banker's Algorithm (Example):

d. If a request from process P_0 arrives for (3,2,2), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	3 3 2	7 5 3	0 1 0	4 2 1
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (25/29)

Banker's Algorithm (Example):

d. If a request from process P_0 arrives for (3,2,2), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	3 3 2	7 5 3	0 1 0	4 2 1
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Deadlocks Avoidance (25/29)

Banker's Algorithm (Example):

d. If a request from process P_0 arrives for $\boxed{(3,2,2)}$, can the request be granted immediately?

	Allocation	Max	Available	Need
	A B C	A B C	A B C	A B C
P_0	3 3 2	7 5 3	0 1 0	4 2 1
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

NO you can **not** granted this request immediately, because the system will be in a **unsafe state** since the available resources after this request are not enough for any process.

Deadlocks Avoidance (26/29)

Banker's Algorithm (Example2):

Consider the following snapshot of a system:

	Allocation	*Max*	*Available*
	A B C D	*A B C D*	*A B C D*
P_0	3 0 1 4	5 1 1 6	0 3 0 1
P_1	2 2 1 0	3 2 1 1	
P_2	3 1 2 1	3 3 2 1	
P_3	0 5 1 0	4 6 1 2	
P_4	4 2 1 2	6 3 2 5	

a. What is the content of the matrix ***Need*** ?

b. Is the system in a safe state? Why?

Deadlocks Avoidance (27/29)

Banker's Algorithm (Example2):

a. What is the content of the matrix ***Need*** ?

	Allocation	*Max*	*Available*	*Need*
	A B C D	*A B C D*	*A B C D*	*A B C D*
P_0	3 0 1 4	5 1 1 6	0 3 0 1	
P_1	2 2 1 0	3 2 1 1		
P_2	3 1 2 1	3 3 2 1		
P_3	0 5 1 0	4 6 1 2		
P_4	4 2 1 2	6 3 2 5		

Banker's Algorithm (Example2):

a. What is the content of the matrix ***Need*** ?

	Allocation	*Max*	*Available*	*Need*
	A B C D	*A B C D*	*A B C D*	*A B C D*
P_0	3 0 1 4	5 1 1 6	0 3 0 1	2 1 0 2
P_1	2 2 1 0	3 2 1 1		1 0 0 1
P_2	3 1 2 1	3 3 2 1		0 2 0 0
P_3	0 5 1 0	4 6 1 2		4 1 0 2
P_4	4 2 1 2	6 3 2 5		2 1 1 3

Deadlocks Avoidance (27/29)

Banker's Algorithm (Example2):

b. Is the system in a safe state? Why?

	Allocation	*Max*	*Available*	*Need*
	A B C D	*A B C D*	*A B C D*	*A B C D*
P_0	3 0 1 4	5 1 1 6	$\boxed{0\ 3\ 0\ 1}$	$\boxed{2\ 1\ 0\ 2}$
P_1	2 2 1 0	3 2 1 1		1 0 0 1
P_2	3 1 2 1	3 3 2 1		0 2 0 0
P_3	0 5 1 0	4 6 1 2		4 1 0 2
P_4	4 2 1 2	6 3 2 5		2 1 1 3

Deadlocks Avoidance (27/29)

Banker's Algorithm (Example2):

b. Is the system in a safe state? Why?

	Allocation	Max	Available	Need
	A B C D	A B C D	A B C D	A B C D
P_0	3 0 1 4	5 1 1 6	0 3 0 1	2 1 0 2
P_1	2 2 1 0	3 2 1 1		1 0 0 1
P_2	3 1 2 1	3 3 2 1		0 2 0 0
P_3	0 5 1 0	4 6 1 2		4 1 0 2
P_4	4 2 1 2	6 3 2 5		2 1 1 3

Deadlocks Avoidance (27/29)

Banker's Algorithm (Example2):

b. Is the system in a safe state? Why?

	Allocation A B C D	Max A B C D	Available A B C D	Need A B C D
P_0	3 0 1 4	5 1 1 6	0 3 0 1	2 1 0 2
P_1	2 2 1 0	3 2 1 1		1 0 0 1
P_2	3 1 2 1	3 3 2 1		0 2 0 0
P_3	0 5 1 0	4 6 1 2		4 1 0 2
P_4	4 2 1 2	6 3 2 5		2 1 1 3

Deadlocks Avoidance (27/29)

Banker's Algorithm (Example2):

b. Is the system in a safe state? Why?

	Allocation A B C D	*Max* A B C D	*Available* A B C D	*Need* A B C D
	$\overline{\text{ }}$			
P_0	3 0 1 4	5 1 1 6	$\boxed{3\ 4\ 2\ 2}$	$\boxed{2\ 1\ 0\ 2}$
P_1	2 2 1 0	3 2 1 1		1 0 0 1
1 ~~P_2~~	~~3 1 2 1~~	~~3 3 2 1~~		~~0 2 0 0~~
P_3	0 5 1 0	4 6 1 2		4 1 0 2
P_4	4 2 1 2	6 3 2 5		2 1 1 3

Banker's Algorithm (Example2):

b. Is the system in a safe state? Why?

		Allocation	*Max*	*Available*	*Need*
		A B C D	*A B C D*	*A B C D*	*A B C D*
2	~~P_0~~	~~3 0 1 4~~	~~5 1 1 6~~	$\boxed{6\ 4\ 3\ 6}$	~~2 1 0 2~~
	P_1	2 2 1 0	3 2 1 1		$\boxed{1\ 0\ 0\ 1}$
1	~~P_2~~	~~3 1 2 1~~	~~3 3 2 1~~		~~0 2 0 0~~
	P_3	0 5 1 0	4 6 1 2		4 1 0 2
	P_4	4 2 1 2	6 3 2 5		2 1 1 3

Deadlocks Avoidance (27/29)

Banker's Algorithm (Example2):

b. Is the system in a safe state? Why?

		Allocation	Max	Available	Need
		A B C D	A B C D	A B C D	A B C D
2	~~P_0~~	~~3 0 1 4~~	~~5 1 1 6~~	$\boxed{8\ 6\ 4\ 6}$	~~2 1 0 2~~
3	~~P_1~~	~~2 2 1 0~~	~~3 2 1 1~~		~~1 0 0 1~~
1	~~P_2~~	~~3 1 2 1~~	~~3 3 2 1~~		~~0 2 0 0~~
	P_3	0 5 1 0	4 6 1 2		$\boxed{4\ 1\ 0\ 2}$
	P_4	4 2 1 2	6 3 2 5		2 1 1 3

Banker's Algorithm (Example2):

b. Is the system in a safe state? Why?

Order	Process	Allocation A B C D	Max A B C D	Available A B C D	Need A B C D
2	~~P_0~~	~~3 0 1 4~~	~~5 1 1 6~~	$\boxed{8\ 11\ 5\ 6}$	~~2 1 0 2~~
3	~~P_1~~	~~2 2 1 0~~	~~3 2 1 1~~		~~1 0 0 1~~
1	~~P_2~~	~~3 1 2 1~~	~~3 3 2 1~~		~~0 2 0 0~~
4	~~P_3~~	~~0 5 1 0~~	~~4 6 1 2~~		~~4 1 0 2~~
	P_4	4 2 1 2	6 3 2 5		$\boxed{2\ 1\ 1\ 3}$

Banker's Algorithm (Example2):

b. Is the system in a safe state? Why?

		Allocation	*Max*	*Available*	*Need*
		A B C D	*A B C D*	*A B C D*	*A B C D*
2	P_0	~~3 0 1 4~~	~~5 1 1 6~~	$\boxed{12\ 13\ 6\ 8}$	~~2 1 0 2~~
3	P_1	~~2 2 1 0~~	~~3 2 1 1~~		~~1 0 0 1~~
1	P_2	~~3 1 2 1~~	~~3 3 2 1~~		~~0 2 0 0~~
4	P_3	~~0 5 1 0~~	~~4 6 1 2~~		~~4 1 0 2~~
5	P_4	~~4 2 1 2~~	~~6 3 2 5~~		~~2 1 1 3~~

Deadlocks Avoidance (27/29)

Banker's Algorithm (Example2):

b. Is the system in a safe state? Why?

Order		*Allocation* A B C D	*Max* A B C D	*Available* A B C D	*Need* A B C D
2	~~P_0~~	~~3 0 1 4~~	~~5 1 1 6~~	$\boxed{12\ 13\ 6\ 8}$	~~2 1 0 2~~
3	~~P_1~~	~~2 2 1 0~~	~~3 2 1 1~~		~~1 0 0 1~~
1	~~P_2~~	~~3 1 2 1~~	~~3 3 2 1~~		~~0 2 0 0~~
4	~~P_3~~	~~0 5 1 0~~	~~4 6 1 2~~		~~4 1 0 2~~
5	~~P_4~~	~~4 2 1 2~~	~~6 3 2 5~~		~~2 1 1 3~~

The system is in a **safe state** since the sequence $< P_2, P_0, P_1, P_3, P_4 >$ **satisfies safety criteria**.

Deadlocks Avoidance (28/29)

Banker's Algorithm (Example2):

c. If a request from process P_2 arrives for (0,3,0,0), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C D	*A B C D*	*A B C D*	*A B C D*
P_0	3 0 1 4	5 1 1 6	0 3 0 1	2 1 0 2
P_1	2 2 1 0	3 2 1 1		1 0 0 1
P_2	3 1 2 1	3 3 2 1		0 2 0 0
P_3	0 5 1 0	4 6 1 2		4 1 0 2
P_4	4 2 1 2	6 3 2 5		2 1 1 3

Banker's Algorithm (Example2):

c. If a request from process P_2 arrives for $\boxed{(0,3,0,0),}$ can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C D	*A B C D*	*A B C D*	*A B C D*
P_0	3 0 1 4	5 1 1 6	0 3 0 1	2 1 0 2
P_1	2 2 1 0	3 2 1 1		1 0 0 1
P_2	3 1 2 1	3 3 2 1		$\boxed{0\ 2\ 0\ 0}$
P_3	0 5 1 0	4 6 1 2		4 1 0 2
P_4	4 2 1 2	6 3 2 5		2 1 1 3

NO you can **not** granted this request, it raise error condition, since process has exceeded its maximum claim.

Deadlocks Avoidance (29/29)

Banker's Algorithm (Example2):

d. If a request from process P_3 arrives for (1,1,0,0), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C D	*A B C D*	*A B C D*	*A B C D*
P_0	3 0 1 4	5 1 1 6	0 3 0 1	2 1 0 2
P_1	2 2 1 0	3 2 1 1		1 0 0 1
P_2	3 1 2 1	3 3 2 1		0 2 0 0
P_3	0 5 1 0	4 6 1 2		4 1 0 2
P_4	4 2 1 2	6 3 2 5		2 1 1 3

Deadlocks Avoidance (29/29)

Banker's Algorithm (Example2):

d. If a request from process P_3 arrives for (1,1,0,0), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C D	*A B C D*	*A B C D*	*A B C D*
P_0	3 0 1 4	5 1 1 6	0 3 0 1	2 1 0 2
P_1	2 2 1 0	3 2 1 1		1 0 0 1
P_2	3 1 2 1	3 3 2 1		0 2 0 0
P_3	0 5 1 0	4 6 1 2		4 1 0 2
P_4	4 2 1 2	6 3 2 5		2 1 1 3

Banker's Algorithm (Example2):

d. If a request from process P_3 arrives for $\boxed{(1,1,0,0),}$ can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C D	*A B C D*	*A B C D*	*A B C D*
P_0	3 0 1 4	5 1 1 6	$\boxed{0\ 3\ 0\ 1}$	2 1 0 2
P_1	2 2 1 0	3 2 1 1		1 0 0 1
P_2	3 1 2 1	3 3 2 1		0 2 0 0
P_3	0 5 1 0	4 6 1 2		$\boxed{4\ 1\ 0\ 2}$
P_4	4 2 1 2	6 3 2 5		2 1 1 3

Deadlocks Avoidance (29/29)

Banker's Algorithm (Example2):

d. If a request from process P_3 arrives for (1,1,0,0), can the request be granted immediately?

	Allocation	*Max*	*Available*	*Need*
	A B C D	*A B C D*	*A B C D*	*A B C D*
P_0	3 0 1 4	5 1 1 6	0 3 0 1	2 1 0 2
P_1	2 2 1 0	3 2 1 1		1 0 0 1
P_2	3 1 2 1	3 3 2 1		0 2 0 0
P_3	0 5 1 0	4 6 1 2		4 1 0 2
P_4	4 2 1 2	6 3 2 5		2 1 1 3

NO you can **not** granted this request immediately, P_3 must **wait**, since **resources are not available**.

Deadlocks Detection (1/16)

Deadlocks Detection:

If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. In this environment, the system may provide:

- An algorithm that examines the state of the system to determine whether a deadlock has occurred.
- An algorithm to recover from the deadlock.

Deadlocks Detection (2/16)

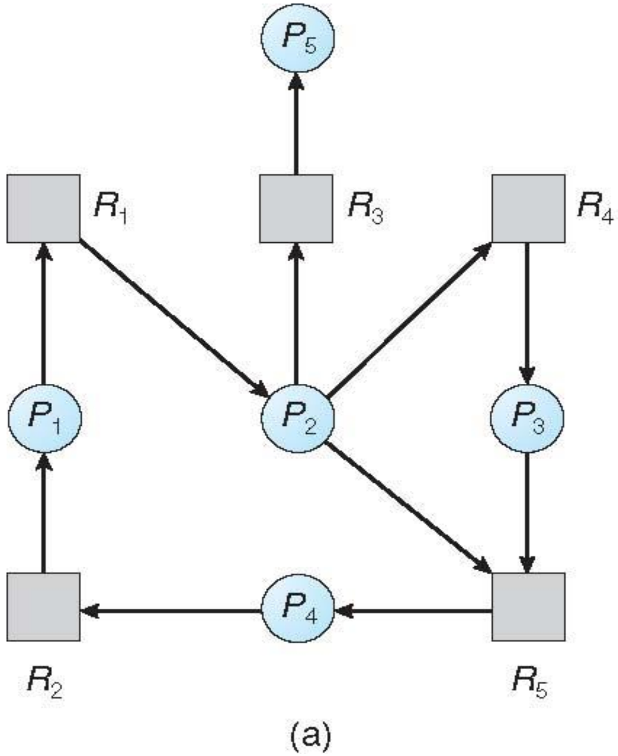
Single Instance of Each Resource Type:

- 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. If there is a cycle, there exists a deadlock.
- 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.

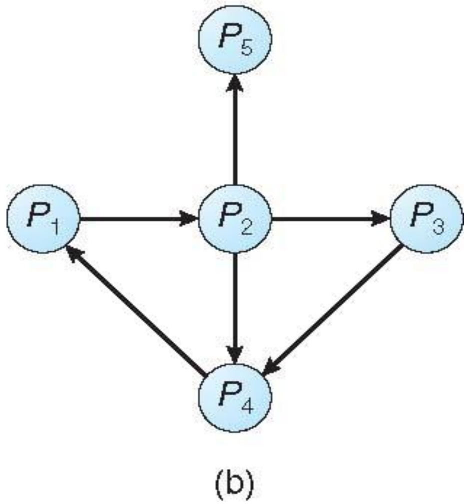
Deadlocks Detection (3/16)

Single Instance of Each Resource Type:

- Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph



Corresponding wait-for graph

Deadlocks Detection (4/16)

Several Instance 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.

Deadlocks Detection (5/16)

Detection Algorithm: (1/2)

1. Let ***Work*** and ***Finish*** be vectors of length m and n, respectively
   Initialize:
   (a) $Work = Available$
   (b) For $i = 1,2, \ldots, n$, 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

Deadlocks Detection (5/16)

Detection Algorithm: (2/2)

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.

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

Deadlocks Detection (6/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*
	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	

Deadlocks Detection (7/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	0 0 0	$\boxed{0\ 0\ 0}$	$\boxed{0\ \ 0\ \ 0}$
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		

Deadlocks Detection (8/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*
	A B C	*A B C*	*A B C*	*A B C*	
P_0	0 1 0	0 0 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			

Deadlocks Detection (8/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	Request	Available	Work	Finish
	$A\ B\ C$	$A\ B\ C$	$A\ B\ C$	$A\ B\ C$	
P_0	0 1 0	0 0 0	0 0 0	0 0 0	false
P_1	2 0 0	2 0 2			false
P_2	3 0 3	0 0 0			false
P_3	2 1 1	1 0 0			false
P_4	0 0 2	0 0 2			false

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*
	A B C	*A B C*	*A B C*	*A B C*	
P_0	0 1 0	0 0 0	0 0 0	0 0 0	false
P_1	2 0 0	2 0 2			false
P_2	3 0 3	0 0 0			false
P_3	2 1 1	1 0 0			false
P_4	0 0 2	0 0 2			false

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*
	A B C	*A B C*	*A B C*	*A B C*	
P_0	0 1 0	$\boxed{0\ 0\ 0}$	0 0 0	$\boxed{0\ 0\ 0}$	$\boxed{\text{false}}$
P_1	2 0 0	2 0 2			false
P_2	3 0 3	0 0 0			false
P_3	2 1 1	1 0 0			false
P_4	0 0 2	0 0 2			false

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*
	A B C	*A B C*	*A B C*	*A B C*	
P_0	0 1 0	0 0 0	0 0 0	0 0 0	false
P_1	2 0 0	2 0 2			false
P_2	3 0 3	0 0 0			false
P_3	2 1 1	1 0 0			false
P_4	0 0 2	0 0 2			false

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*	
	A B C	*A B C*	*A B C*	*A B C*		
P_0	0 1 0	0 0 0	0 0 0	0 1 0	true	1
P_1	2 0 0	2 0 2			false	
P_2	3 0 3	0 0 0			false	
P_3	2 1 1	1 0 0			false	
P_4	0 0 2	0 0 2			false	

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	Request	Available	Work	Finish	
	A B C	A B C	A B C	A B C		
P_0	0 0 0	0 0 0	0 0 0	$\boxed{0\ 1\ 0}$	true	$\boxed{1}$
P_1	2 0 0	2 0 2			false	
P_2	3 0 3	0 0 0			false	
P_3	2 1 1	1 0 0			false	
P_4	0 0 2	0 0 2			false	

Deadlocks Detection (9/12)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*	
	A B C	*A B C*	*A B C*	*A B C*		
P_0	0 0 0	0 0 0	0 0 0	$\boxed{0\ 1\ 0}$	true	$\boxed{1}$
P_1	2 0 0	2 0 2			false	
P_2	3 0 3	$\boxed{0\ 0\ 0}$			$\boxed{\text{false}}$	
P_3	2 1 1	1 0 0			false	
P_4	0 0 2	0 0 2			false	

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*	
	A B C	*A B C*	*A B C*	*A B C*		
P_0	0 0 0	0 0 0	0 0 0	$\boxed{0\ 1\ 0}$	true	1
P_1	2 0 0	2 0 2			false	
P_2	$\boxed{3\ 0\ 3}$	0 0 0			false	
P_3	2 1 1	1 0 0			false	
P_4	0 0 2	0 0 2			false	

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*	
	A B C	*A B C*	*A B C*	*A B C*		
P_0	0 0 0	0 0 0	0 0 0	$\boxed{3\ 1\ 3}$	true	$\boxed{1}$
P_1	2 0 0	2 0 2			false	
P_2	$\boxed{3\ 0\ 3}$	0 0 0			$\boxed{\text{true}}$	$\boxed{2}$
P_3	2 1 1	1 0 0			false	
P_4	0 0 2	0 0 2			false	

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	Request	Available	Work	Finish	
	A B C	A B C	A B C	A B C		
P_0	0 0 0	0 0 0	0 0 0	3 1 3	true	1
P_1	2 0 0	2 0 2			false	
P_2	0 0 0	0 0 0			true	2
P_3	2 1 1	1 0 0			false	
P_4	0 0 2	0 0 2			false	

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*	
	A B C	*A B C*	*A B C*	*A B C*		
P_0	0 0 0	0 0 0	0 0 0	3 1 3	true	1
P_1	2 0 0	2 0 2			false	
P_2	0 0 0	0 0 0			true	2
P_3	2 1 1	1 0 0			false	
P_4	0 0 2	0 0 2			false	

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	Request	Available	Work	Finish	
	$A\ B\ C$	$A\ B\ C$	$A\ B\ C$	$A\ B\ C$		
P_0	0 0 0	0 0 0	0 0 0	$\boxed{3\ 1\ 3}$	true	$\boxed{1}$
P_1	2 0 0	2 0 2			false	
P_2	0 0 0	0 0 0			true	$\boxed{2}$
P_3	$\boxed{2\ 1\ 1}$	1 0 0			$\boxed{\text{false}}$	
P_4	0 0 2	0 0 2			false	

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*	
	A B C	*A B C*	*A B C*	*A B C*		
P_0	0 0 0	0 0 0	0 0 0	5 2 4	true	1
P_1	2 0 0	2 0 2			false	
P_2	0 0 0	0 0 0			true	2
P_3	2 1 1	1 0 0			true	3
P_4	0 0 2	0 0 2			false	

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	Request	Available	Work	Finish	
	A B C	A B C	A B C	A B C		
P_0	0 0 0	0 0 0	0 0 0	$\boxed{5\ 2\ 4}$	true	1
P_1	2 0 0	2 0 2			false	
P_2	0 0 0	0 0 0			true	2
P_3	0 0 0	0 0 0			true	3
P_4	0 0 2	$\boxed{0\ 0\ 2}$			$\boxed{\text{false}}$	

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*	
	A B C	*A B C*	*A B C*	*A B C*		
P_0	0 0 0	0 0 0	0 0 0	5 2 4	true	1
P_1	2 0 0	2 0 2			false	
P_2	0 0 0	0 0 0			true	2
P_3	0 0 0	0 0 0			true	3
P_4	0 0 2	0 0 2			false	

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*	
	A B C	*A B C*	*A B C*	*A B C*		
P_0	0 0 0	0 0 0	0 0 0	5 2 6	true	1
P_1	2 0 0	2 0 2			false	
P_2	0 0 0	0 0 0			true	2
P_3	0 0 0	0 0 0			true	3
P_4	0 0 2	0 0 2			true	4

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*	
	A B C	*A B C*	*A B C*	*A B C*		
P_0	0 0 0	0 0 0	0 0 0	5 2 6	true	1
P_1	2 0 0	2 0 2			false	
P_2	0 0 0	0 0 0			true	2
P_3	0 0 0	0 0 0			true	3
P_4	0 0 0	0 0 0			true	4

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*	
	A B C	*A B C*	*A B C*	*A B C*		
P_0	0 0 0	0 0 0	0 0 0	$\boxed{5\ 2\ 6}$	true	1
P_1	$\boxed{2\ 0\ 0}$	2 0 2			$\boxed{\text{false}}$	
P_2	0 0 0	0 0 0			true	2
P_3	0 0 0	0 0 0			true	3
P_4	0 0 0	0 0 0			true	4

Deadlocks Detection (9/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	Request	Available	Work	Finish	
	A B C	*A B C*	*A B C*	*A B C*		
P_0	0 0 0	0 0 0	0 0 0	7 2 6	true	1
P_1	2 0 0	2 0 2			true	5
P_2	0 0 0	0 0 0			true	2
P_3	0 0 0	0 0 0			true	3
P_4	0 0 0	0 0 0			true	4

Deadlocks Detection (10/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation	Request	Available	Work	Finish	
	A B C	A B C	A B C	A B C		
P_0	0 0 0	0 0 0	0 0 0	7 2 6	true	1
P_1	0 0 0	0 0 0			true	5
P_2	0 0 0	0 0 0			true	2
P_3	0 0 0	0 0 0			true	3
P_4	0 0 0	0 0 0			true	4

Deadlocks Detection (10/16)

Detection Algorithm (Example1):

a. Is the system is in deadlock state? Why?

	Allocation A B C	*Request* A B C	*Available* A B C	*Work* A B C	*Finish*	
P_0	0 0 0	0 0 0	0 0 0	7 2 6	true	1
P_1	0 0 0	0 0 0			true	5
P_2	0 0 0	0 0 0			true	2
P_3	0 0 0	0 0 0			true	3
P_4	0 0 0	0 0 0			true	4

We claim that the system is **not** in a **deadlocked** state. Indeed, if we execute our algorithm, we will find that the sequence $< P_0, P_2, P_3, P_4, P_1>$ results in *Finish*[*i*] == *true* for all *i*.

Deadlocks Detection (11/16)

Detection Algorithm (Example 2):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*
	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 1**	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Deadlocks Detection (12/16)

Detection Algorithm (Example 2):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	0 0 0	$\boxed{0\ 0\ 0}$	$\boxed{0\ 0\ 0}$
P_1	2 0 0	2 0 2		
P_2	3 0 3	0 0 1		
P_3	2 1 1	1 0 0		
P_4	0 0 2	0 0 2		

Deadlocks Detection (13/16)

Detection Algorithm (Example 2):

a. Is the system is in deadlock state? Why?

	Allocation	Request	Available	Work	Finish
	A B C	A B C	A B C	A B C	
P_0	0 1 0	0 0 0	0 0 0	0 0 0	false
P_1	2 0 0	2 0 2			false
P_2	3 0 3	0 0 1			false
P_3	2 1 1	1 0 0			false
P_4	0 0 2	0 0 2			false

Deadlocks Detection (14/16)

Detection Algorithm (Example 2):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*
	A B C	*A B C*	*A B C*	*A B C*	
P_0	0 1 0	0 0 0	0 0 0	0 0 0	false
P_1	2 0 0	2 0 2			false
P_2	3 0 3	0 0 1			false
P_3	2 1 1	1 0 0			false
P_4	0 0 2	0 0 2			false

Deadlocks Detection (14/16)

Detection Algorithm (Example 2):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*
	A B C	*A B C*	*A B C*	*A B C*	
P_0	0 1 0	$\boxed{0\ 0\ 0}$	0 0 0	$\boxed{0\ 0\ 0}$	$\boxed{\text{false}}$
P_1	2 0 0	2 0 2			false
P_2	3 0 3	0 0 1			false
P_3	2 1 1	1 0 0			false
P_4	0 0 2	0 0 2			false

Deadlocks Detection (14/16)

Detection Algorithm (Example 2):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*
	A B C	*A B C*	*A B C*	*A B C*	
P_0	$\boxed{0\ 1\ 0}$	0 0 0	0 0 0	$\boxed{0\ 0\ 0}$	$\boxed{\text{false}}$
P_1	2 0 0	2 0 2			false
P_2	3 0 3	0 0 1			false
P_3	2 1 1	1 0 0			false
P_4	0 0 2	0 0 2			false

Deadlocks Detection (14/16)

Detection Algorithm (Example 2):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*	
	A B C	*A B C*	*A B C*	*A B C*		
P_0	$\boxed{0\ 1\ 0}$	0 0 0	0 0 0	$\boxed{0\ 1\ 0}$	$\boxed{\text{true}}$	$\boxed{1}$
P_1	2 0 0	2 0 2			false	
P_2	3 0 3	0 0 1			false	
P_3	2 1 1	1 0 0			false	
P_4	0 0 2	0 0 2			false	

Deadlocks Detection (14/16)

Detection Algorithm (Example 2):

a. Is the system is in deadlock state? Why?

	Allocation	Request	Available	Work	Finish	
	A B C	A B C	A B C	A B C		
P_0	0 0 0	0 0 0	0 0 0	0 1 0	true	1
P_1	2 0 0	2 0 2			false	
P_2	3 0 3	0 0 1			false	
P_3	2 1 1	1 0 0			false	
P_4	0 0 2	0 0 2			false	

Deadlocks Detection (15/16)

Detection Algorithm (Example 2):

a. Is the system is in deadlock state? Why?

	Allocation	*Request*	*Available*	*Work*	*Finish*	
	A B C	*A B C*	*A B C*	*A B C*		
P_0	0 0 0	0 0 0	0 0 0	0 1 0	true	1
P_1	2 0 0	2 0 2			false	
P_2	3 0 3	0 0 1			false	
P_3	2 1 1	1 0 0			false	
P_4	0 0 2	0 0 2			false	

We claim that the system is now **deadlocked**. Although we can reclaim the resources held by process P_0, the number of available resources is not sufficient to fulfill the requests of the other processes. Thus, a deadlock exists, consisting of processes P_1, P_2, P_3, and P_4.

Deadlocks Detection (16/16)

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?
    - One for each disjoint cycle.
- 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 (1/3)

<u>Recovery from Deadlock:</u>

- Process Termination.
- Resource Preemption.

Recovery from Deadlock (2/3)

Process Termination: (1/2)

1. Abort all deadlocked processes.
2. Abort one process at a time until the deadlock cycle is eliminated.

Recovery from Deadlock (2/3)

Process Termination: (2/2)

3. 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 (3/3)

Resource Preemption:

- If preemption is required to deal with deadlocks, then three issues need to be addressed:

1. **Selecting a victim** – Which resources and which processes are to be preempted? (minimize cost).
2. **Rollback** – return to some safe state, restart process for that state.
3. **Starvation** – same process may always be picked as victim, include number of rollback in cost factor.

Thank You

Dr. Ahmed Hagag

ahagag@fci.bu.edu.eg

https://www.youtube.com/channel/UCzYgAyyZTLfnLFjQexOKxbQ

https://www.facebook.com/ahmed.hagag.71/