Operating Systems

Lecture 6b

Dr Ronald Grau

School of Engineering and Informatics

Spring term 2018

Previously

Deadlocks

- Deadlock conditions
- Methods of handling deadlocks

Today

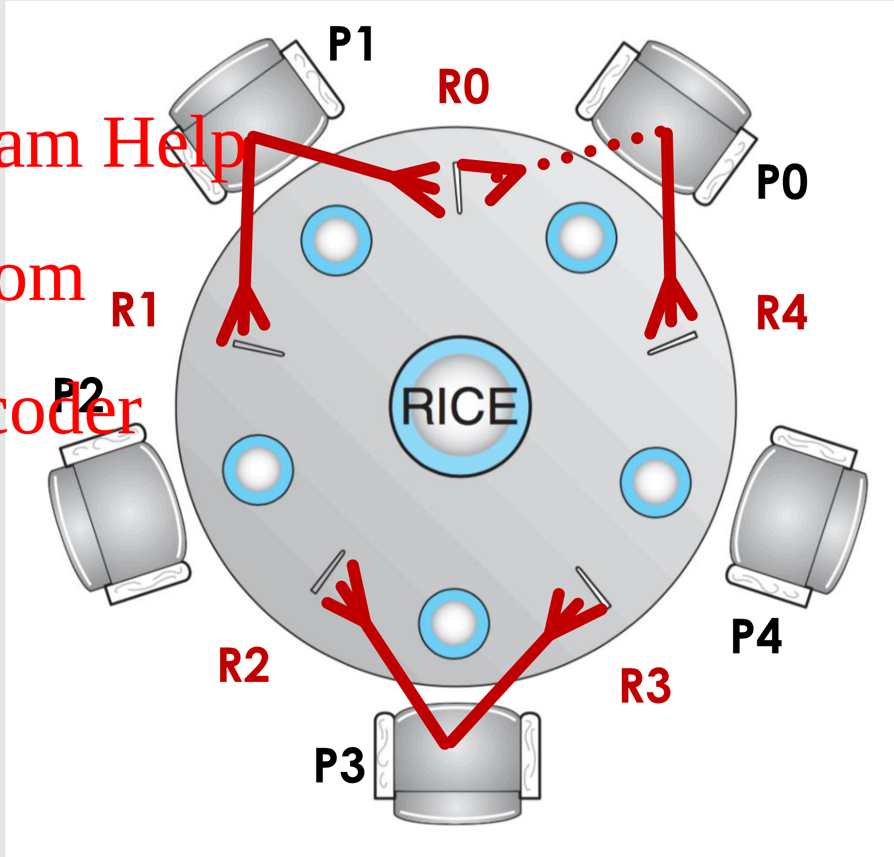
More about deadlocks

- More deadlock avoidance
- Resource Allocation Graphs
- Deadlock detection and recovery

Recap: Prevention of deadlocks

Prevention by design

- Make sure that at least one of the conditions for a deadlock does not occur:
  - Mutual exclusion
  - Hold-and-wait
  - No-preemption
  - Circular wait



Recap: Avoidance of deadlocks

Resource *Allocation* Denial

`Request[i]` for process P_i

1. If `Request[i] <= Need[i]`, go to step 2.
   Otherwise, error (maximum exceeded).

2. If `Request[i] < Available`, go to step 3.
   Otherwise, P_i must wait (resources not available).

3. Pretend to fulfill request:

```
Available := Available - Request[i]
Allocation[i] := Allocation[i] + Request[i]
Need[i] := Need[i] - Request[i]
```

   Approve request if resulting state is safe.
   Otherwise, restore state and set P_i to wait.

Recap: Avoidance of deadlocks

Allocation			
	R_0	R_1	R_2
P_0	0	1	0
P_1	2	0	0
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

Need			
	R_0	R_1	R_2
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

Available		
R_0	R_1	R_2
3	3	2

- Assume request `(1 0 2)` by P_1

Recap: Avoidance of deadlocks

Allocation			
	R_0	R_1	R_2
P_0	0	1	0
P_1	2	0	0
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

Need			
	R_0	R_1	R_2
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

Available		
R_0	R_1	R_2
3	3	2

- Assume request `(1 0 2)` by P_1

Request can be served

Allocation			
	R_0	R_1	R_2
P_0	0	1	0
P_1	3	0	2
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

Need			
	R_0	R_1	R_2
P_0	7	4	3
P_1	0	2	0
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Available		
R_0	R_1	R_2
2	3	0

Sufficient resources to continue with P_1

Recap: Avoidance of deadlocks

Allocation			
	R_0	R_1	R_2
P_0	0	1	0
P_1	3	0	2
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

Need			
	R_0	R_1	R_2
P_0	7	4	3
P_1	0	2	0
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Available		
R_0	R_1	R_2
2	3	0

- Assume request `(3 3 0)` by P_4
  Request cannot be granted (insufficient resources)
- Assume request `(0 2 0)` by P_0
  Request cannot be granted (would result in unsafe state with only {2,1,0} available resources)

Avoidance of deadlocks

Resource *Initiation* Denial

- Admit process P_n only when its resource requests cannot cause a deadlock

$$\sum_{i=0}^{n-1} \texttt{Need[i][j]} + \texttt{Need[n][j]} \texttt{ <= } \texttt{Available[j]}$$

- Assumes that processes know in advance the amount of resources they will need
- Assumes worst-case resource allocation
  (all processes request their maximum resources at the same time)

Resource allocation graphs

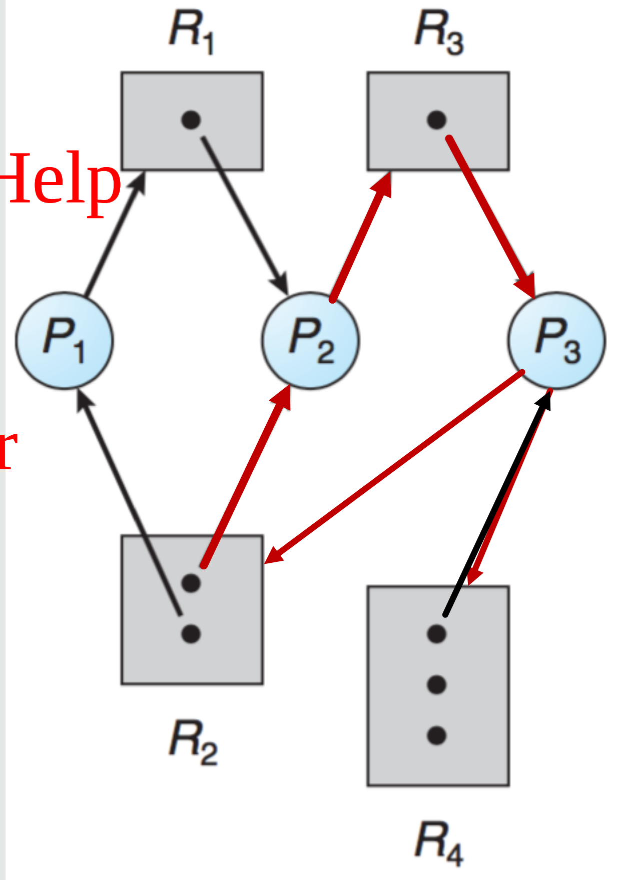
Graphical models of resource allocation

- Process nodes P_i
- Resource nodes R_i
- Number of instances of each resource type

- Request edge $P_i \rightarrow R_j$
- Assignment edge $R_j \rightarrow P_i$

What happens if $P_3 \rightarrow R_2$?

Cycles in the graph indicate a deadlock.



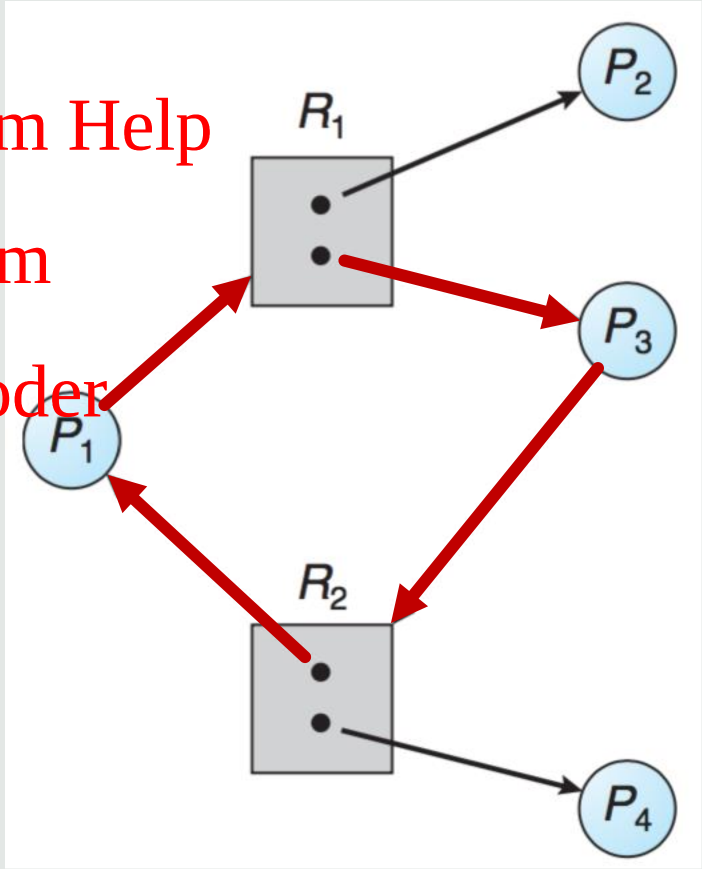
Resource allocation graphs

Graphical models of resource allocation

- Process nodes P_i
- Resource nodes R_i
- Number of instances of each resource type

- Request edge $P_i \rightarrow R_j$
- Assignment edge $R_j \rightarrow P_i$

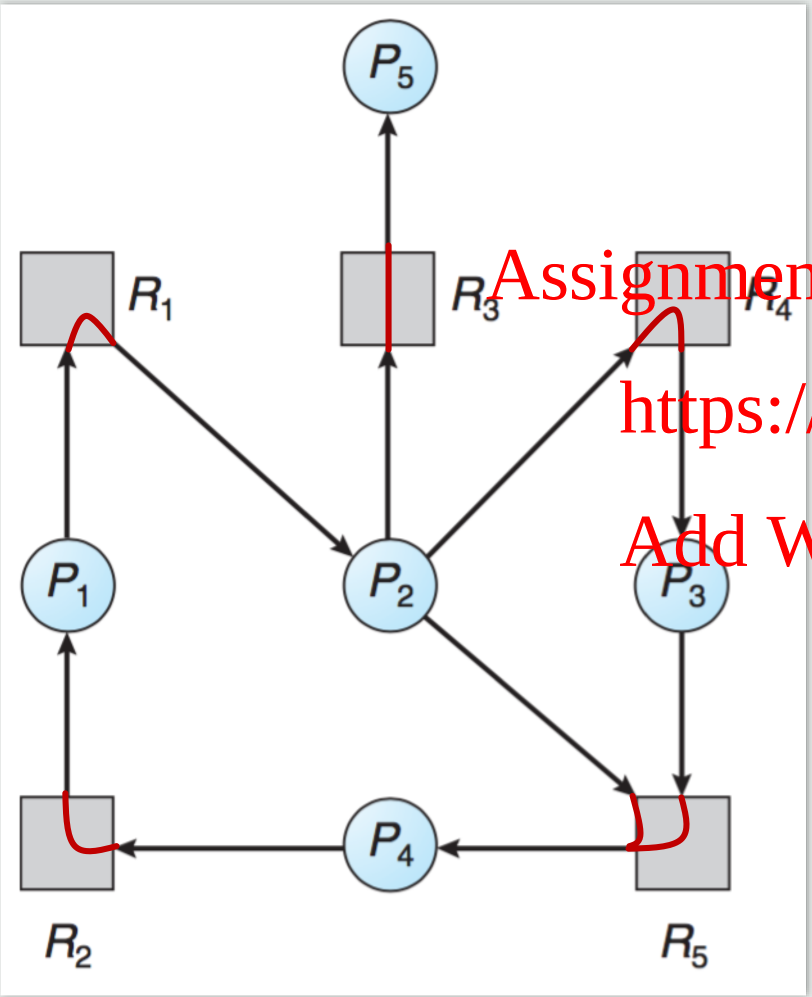
Is there a deadlock in this system? **Yes.**



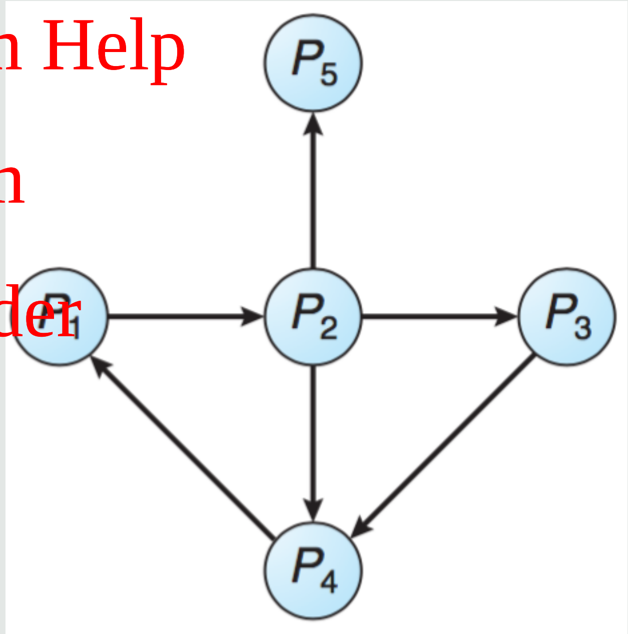
Deadlock Detection

- Always grant resource requests when resources are available
- Examine the state of the system to determine whether a deadlock has occurred
- If so, recover from the deadlock
- Drawbacks:
  - Runtime overhead for deadlock detection
  - Potential losses due to recovery

Deadlock Detection – Single resources



Resource allocation graph



Wait graph

Deadlock Detection – Multiple resources

- There are `Available[j]` instances of each resource type R_j
- `Allocation[i][j]` is the number of instances of resource type R_j held by process P_i
- `Request[i][j]` is the number of additional instances of resource type R_j requested by process P_i

1. `Work := Available`
   `Finish[i] := false` if `Allocation[i] != 0`; otherwise `Finish[i] := true`
2. Find an index **i** such that
   `Finish[i] = false ^ Request[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 ] == false` for some **i** , then the system is in a deadlocked state.
   P_i is deadlocked if `Finish[i] == false`.

Deadlock Detection – Multiple resources

Example 1

Allocation			
	R_0	R_1	R_2
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

Request			
	R_0	R_1	R_2
P_0	0	0	0
P_1	2	0	2
P_2	0	0	0
P_3	1	0	0
P_4	0	0	2

Finish	
P_0	false
P_1	false
P_2	false
P_3	false
P_4	false

Work		
R_0	R_1	R_2
0	0	0

Deadlock Detection – Multiple resources

Example 1

Allocation			
	R_0	R_1	R_2
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

Request			
	R_0	R_1	R_2
P_0	0	0	0
P_1	2	0	2
P_2	0	0	0
P_3	1	0	0
P_4	0	0	2

Finish	
P_0	false
P_1	false
P_2	false
P_3	false
P_4	false

Work		
R_0	R_1	R_2
0	0	0

Deadlock Detection – Multiple resources

Example 1

Allocation			
	R_0	R_1	R_2
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

Request			
	R_0	R_1	R_2
P_0	0	0	0
P_1	2	0	2
P_2	0	0	0
P_3	1	0	0
P_4	0	0	2

Finish	
P_0	true
P_1	false
P_2	false
P_3	false
P_4	false

Work		
R_0	R_1	R_2
0	1	0

Deadlock Detection – Multiple resources

Example 1

Allocation			
	R_0	R_1	R_2
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

Request			
	R_0	R_1	R_2
P_0	0	0	0
P_1	2	0	2
P_2	0	0	0
P_3	1	0	0
P_4	0	0	2

Finish	
P_0	true
P_1	false
P_2	false
P_3	false
P_4	false

Work		
R_0	R_1	R_2
0	1	0

Deadlock Detection – Multiple resources

Example 1

Allocation			
	R_0	R_1	R_2
P_0	0	1	0
P_1	2	0	0
P_2	0	0	0
P_3	2	1	1
P_4	0	0	2

Request			
	R_0	R_1	R_2
P_0	0	0	0
P_1	2	0	2
P_2	0	0	0
P_3	1	0	0
P_4	0	0	2

Finish	
P_0	true
P_1	false
P_2	true
P_3	false
P_4	false

Work		
R_0	R_1	R_2
3	1	3

Deadlock Detection – Multiple resources

Example 1

Allocation			
	R_0	R_1	R_2
P_0	0	0	0
P_1	0	0	0
P_2	0	0	0
P_3	0	0	0
P_4	0	0	0

Request			
	R_0	R_1	R_2
P_0	0	0	0
P_1	0	0	0
P_2	0	0	0
P_3	0	0	0
P_4	0	0	0

Finish	
P_0	true
P_1	true
P_2	true
P_3	true
P_4	true

Work		
R_0	R_1	R_2
7	2	6

All requests served – no deadlock

Deadlock Detection – Multiple resources

Example 2

Allocation			
	R_0	R_1	R_2
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

Request			
	R_0	R_1	R_2
P_0	0	0	0
P_1	2	0	2
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

Finish	
P_0	false
P_1	false
P_2	false
P_3	false
P_4	false

Work		
R_0	R_1	R_2
0	0	0

Deadlock Detection – Multiple resources

Example 2

Allocation			
	R_0	R_1	R_2
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

Request			
	R_0	R_1	R_2
P_0	0	0	0
P_1	2	0	2
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

Finish	
P_0	false
P_1	false
P_2	false
P_3	false
P_4	false

Work		
R_0	R_1	R_2
0	0	0

Deadlock Detection – Multiple resources

Example 2

Allocation			
	R_0	R_1	R_2
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

Request			
	R_0	R_1	R_2
P_0	0	0	0
P_1	2	0	2
P_2	0	0	0
P_3	1	0	0
P_4	0	0	2

Finish	
P_0	true
P_1	false
P_2	false
P_3	false
P_4	false

Work		
R_0	R_1	R_2
0	1	0

Deadlock Detection – Multiple resources

Example 2

Allocation	R_0	R_1	R_2
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

Request	R_0	R_1	R_2
P_0	0	0	0
P_1	2	0	2
P_2	0	0	0
P_3	1	0	0
P_4	0	0	2

Finish	
P_0	true
P_1	false
P_2	false
P_3	false
P_4	false

Work		
R_0	R_1	R_2
0	1	0

No further requests can be served - Processes P_1, … , P_4 deadlocked

Deadlock Detection

When to do deadlock detection?

- How often is a deadlock likely to occur?
- How many processes will be affected by deadlock when it happens?

- Run it
  - on every resource request?
  - every hour?
  - based on metrics?

Deadlock Recovery

- Abort
  - all processes involved in deadlock
  - individual processes until there is no deadlock anymore
- Rollback
  - all processes involved in deadlock
  - individual processes until there is no deadlock anymore

- How to choose which processes to kill or preempt?
  - Minimise “loss"
  - Least CPU time?
  - Fewest allocated resources?
- Deadlock could happen again

Combined Deadlock Handling Strategy

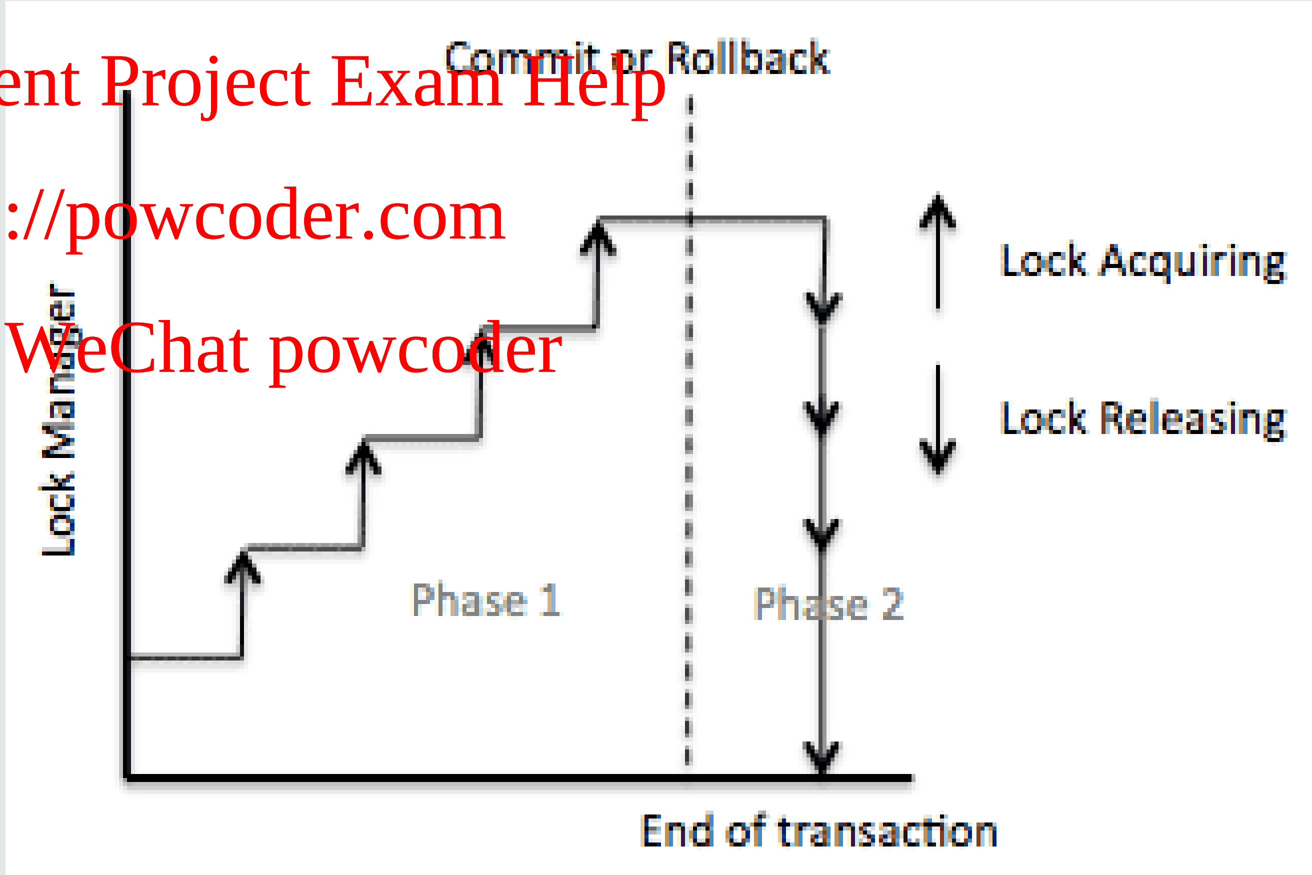
For example:

- Group resources into resource types:
  - Disks
  - I/O devices
  - Files
  - Main memory
- Prevent circular wait between these classes
- Use appropriate methods within these classes, e.g.
  - Prevent no-preemption on main memory
  - Avoidance on I/O devices
  - . . .

Two-Phase Locking

As used in databases

- Similar to preventing no-preemption



Summary

Deadlocks

- Methods for handling deadlocks
  - Deadlock prevention
  - Deadlock avoidance
  - Deadlock detection and recovery

Read

- Tanenbaum & Bos., Modern Operating Systems
  - Chapter 6
- Silberschatz et al., Operating System Concepts
  - Chapter 7

Next Lecture

- Introduction
- Operating System Architectures
- Processes
- Threads - Programming
- Process Scheduling - Evaluation
- Process Synchronisation
- Deadlocks
- **Memory Management**
- File Systems
- Input / Output
- Security and Virtualisation