Deadlock

Organized By: Vinay Arora
Assistant Professor
CSED, TU

Disclaimer

This is NOT A COPYRIGHT MATERIAL

<u>Content has been taken mainly from the following books</u>:

Operating Systems Concepts By Silberschatz & Galvin,
Operating Systems: Internals and Design Principles By William Stallings
www.os-book.com
www.cs.jhu.edu/~yairamir/cs418/os2/sld001.htm
www.personal.kent.edu/~rmuhamma/OpSystems/os.html
http://msdn.microsoft.com/en-us/library/ms685096(VS.85).aspx
http://www.computer.howsttuffworks.com/operating-system6.htm
http://williamstallings.com/OS/Animations.html
Etc…

Deadlock – Real Life Scenario

System Model

- 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.
- Each process utilizes a resource as follows:
  - request
  - use
  - release

Deadlock Characterization

- *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 $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that P0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by

  $P_2, \ldots, P_{n-1}$ is waiting for a resource that is held by

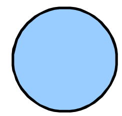
  P_n, and P_0 is waiting for a resource that is held by P_0.

RAG

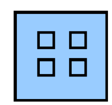
- A set of vertices V and a set of edges E.
- V is partitioned into two types:
  - P = {P1, P2, …, Pn}, the set consisting of all the processes in the system.
  - R = {R1, R2, …, Rm}, the set consisting of all resource types in the system.
- request edge – directed edge P1 → Rj
- assignment edge – directed edge Rj → Pi

Representation

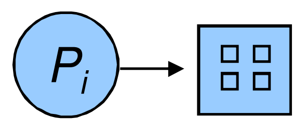
- Process



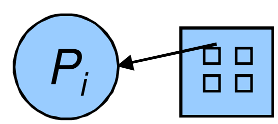
- Resource Type with 4 instances



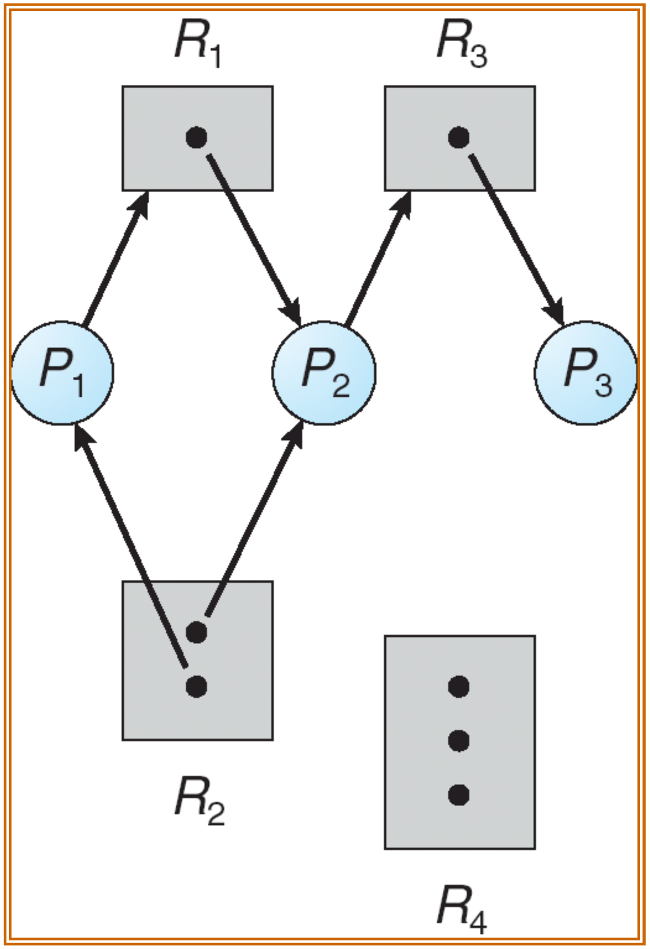
- Pi requests instance of Rj



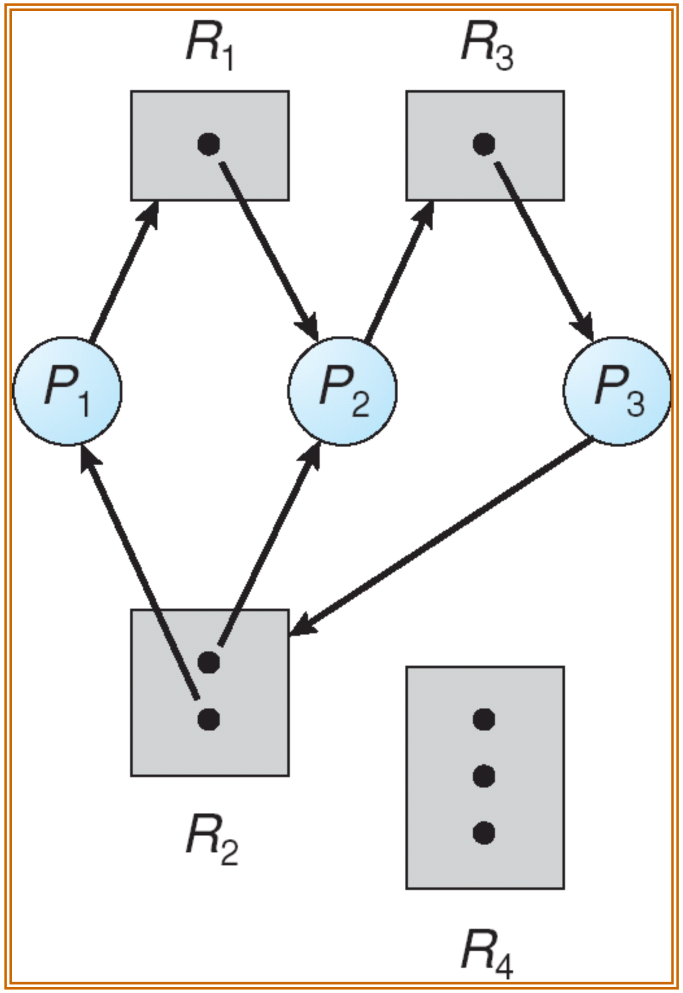
- Pi is holding an instance of Rj



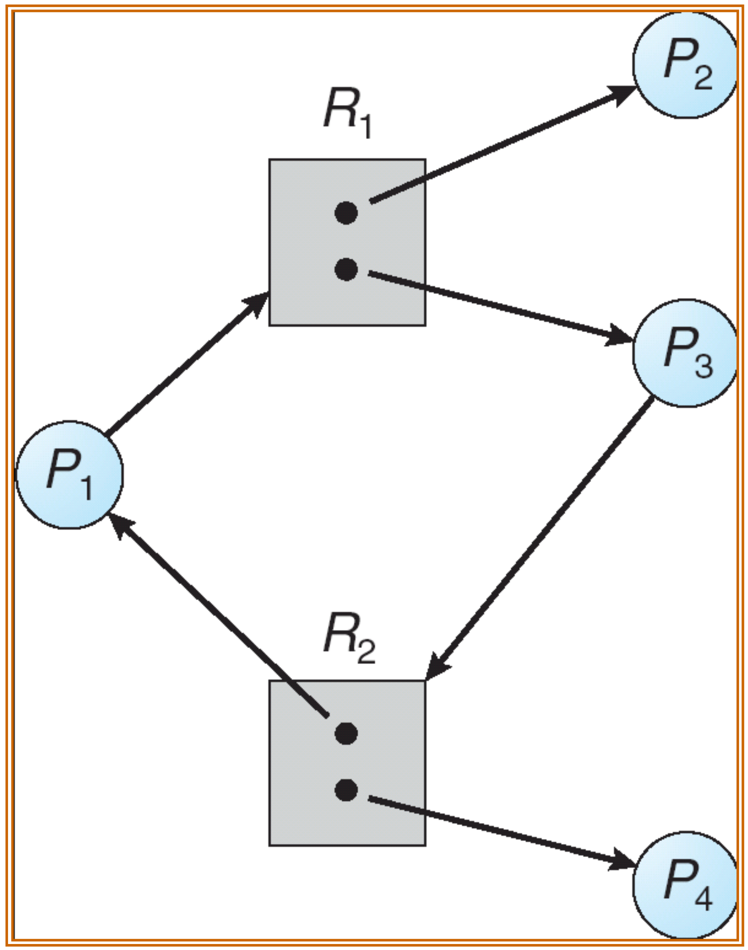
Example - RAG



RAG with Deadlock



RAG with 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.

Deadlock Prevention

- Mutual Exclusion – not required for sharable resources; 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.

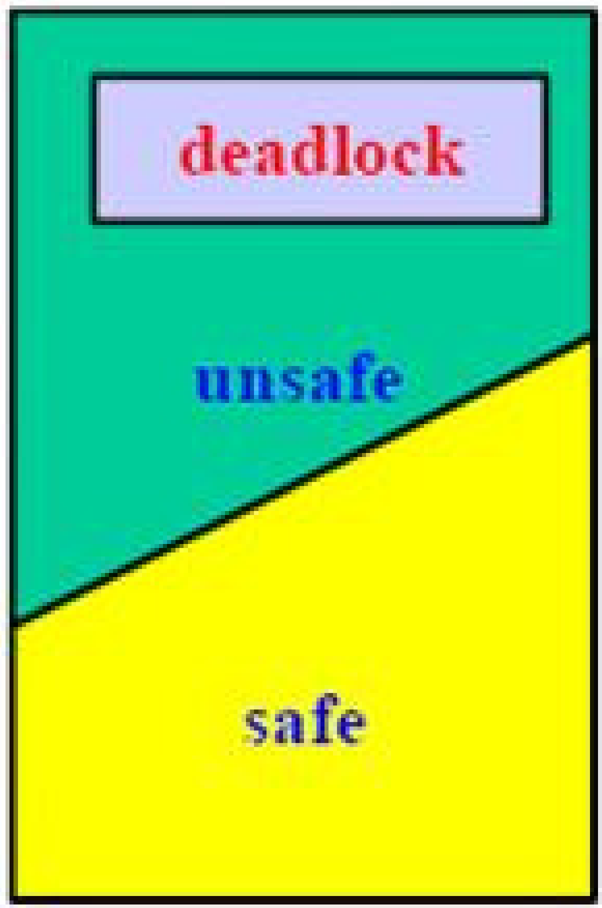
  - Low resource utilization; starvation possible.

- 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.

Safe, Unsafe & Deadlock State



Banker's Algorithm – Safety Algo.

1. Let Work and Finish be vectors of length m and n, respectively.
   Initialize Work = Available and Finish[i]=False for i=0,1,2……,n-1

2. Find an index i such that both

   Finish[i] = = False

   $Need_i$ < 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.

Banker's Algo. – Resource Request Algo.

1. If $Request_i \leq Need_i$, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.

2. If $Request_i \leq Available$, go to step 3. Otherwise, P_i must wait, since the resources are not available.

3. Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows:

$$Available = Available – Request_i$$

$$Allocation_i = Allocation_i + Request_i$$

$$Need_i = Need_i - Request_i$$

Problem Statement

	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	

$Request_1 = (1,0,2)$

After Calculating the Matrix Execute for below mentioned Requests

$Request_0 = (0,2,0)$

$Request_4 = (3,3,0)$

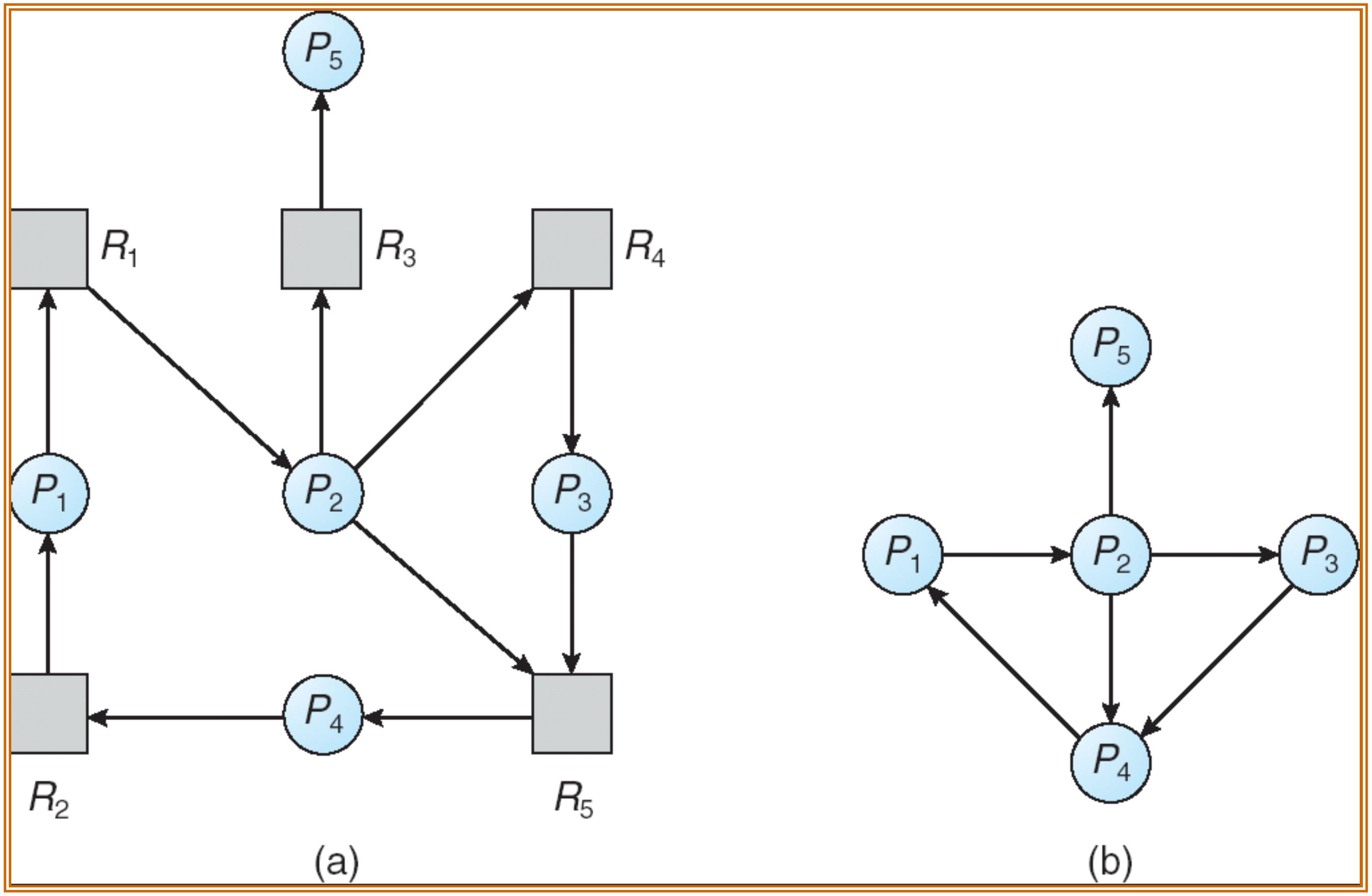
Deadlock Detection

- 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.

RAG & Wait For Graph



Several Instances of resource Type

- Available: A vector of length m indicates the number of available resources of each type.

- Allocation: An n x m matrix defines the number of resources of each type currently allocated to each process.

- Request: An n x m matrix indicates the current request of each process. If Request [ij] = k, then process Pi is requesting k more instances of resource type. Rj.

Deadlock Detection

1. Let Work and Finish be vectors of length m and n, respectively.
   Initialize Work = Available. For i=0,1,2……,n-1, if $Allocation_i$ Not equal to ZERO, then Finish[i] = False; Otherwise,Finish[i] = True

2. Find an index i such that both
   Finish[i] = = False
   $Request_i \leq Work$
   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, then the system is in Deadlock State.

Problem Statement

	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	

Request
A B C
0 0 0
2 0 2
0 0 1
1 0 0
0 0 2

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.

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.

Thnx…