Deadlock Handling

RESOURCE MANAGEMENT

Objectives:

- Define deadlocks which prevent sets of concurrent processes from completing their tasks;
- Present a number of different methods for preventing or avoiding deadlocks in a computer system.

Deadlock: definition

- Permanent blocking of a set of processes that compete for a system resource.

  (*by Eskicioglu & Marsland*)

- Situation wherein each of a collection of processes is waiting for something from other processes in the collection. Since all are waiting, none can provide any of the things being waited for.

  (by: *www.cs.wisc.edu)*

Deadlock : definition

- Situation wherein a process is requesting a resource which is withheld by another process and never get hold of the resource.

(*by Silverchatz & Galvin*)

Examples of Deadlock

- System has 2 tape drives
  - P1 and P2 each hold one tape drive and each needs another one.

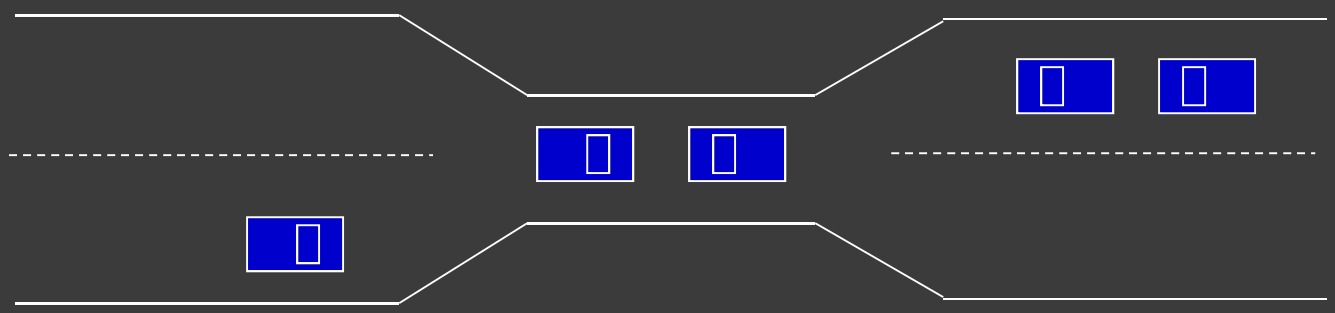
Semaphore: A=1, B = 1

P1:	P2:
Wait (A);	Wait (B);
Wait (B);	Wait (A);

Deadlock

- Deadlock occurs when a set of processes are in a wait state, because each process is waiting for a resource that is held by some other waiting process.
- Thus, all deadlocks involve conflicting resource that are needed by two or more processes.

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.

Additional Information

- Unlike some other problems in multiprogramming systems, there is no efficient solution to the deadlock problem in the general case. It is one area where there is a strong theory but it is completely ignored in practice because solutions to deadlock are expensive.

Resource

- A hardware device
  - E.g. tape drive, memory
- Or a piece of information
  - E.g. a locked record in a database

Categories of Resources

- Reusable
  - Something that can be safely used by one process at a time and is not depleted by that use. Processes obtain resources that they later release for reuse by others.
  - *Example:* CPU, Memory, specific I/O devices or files.

Categories of Resources

- Consumable
  - These can be created and destroyed. When a resource is acquired by a process, the resource ceases to exist.
  - *Examples:* Interrupts, signals, messages.

Types of Resources

- Preemptable resource
  - Resource that can be taken away from the process owning it with no ill effect (needs save/restore)
  - *Example:* Memory or CPU

Types of Resources

- Non-preemptable
  - Resource that cannot be taken away from its current owner without causing the computation to fail.
  - *Example:* Printer or floppy disk.

Note:

- Deadlocks occur only when sharing reusable and non-preemptable resources

Pop Exercise

- Classify the following as either sharable or non-sharable:
  - A. Hard disk
  - B. file
  - C. keyboard

System Model

- System consists of a finite number of resources and a finite number of processes competing for resources.

- Resources are partitioned into several types, each of which consists of some number of identical instances

System Model

- When a process requests an instance of a resource type, the allocation of any instance of the type will satisfy the request (otherwise, resource types have not been defined properly).

Resource Utilization: Protocol

- **Request** – if the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource.
- **Use** – process can utilize the resource
- **Release** – process relinquishes the resources.

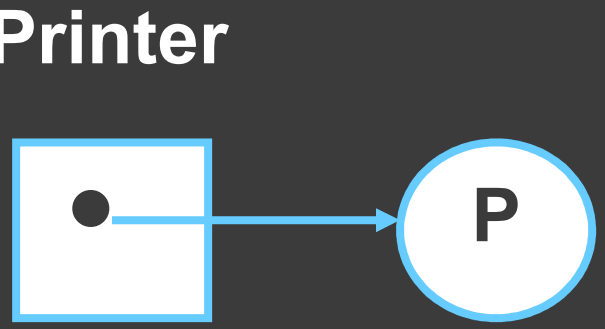
Note that…

- Set of processes is deadlocked when every process in the set is waiting for an event that can be caused only by another process in the set.

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously:

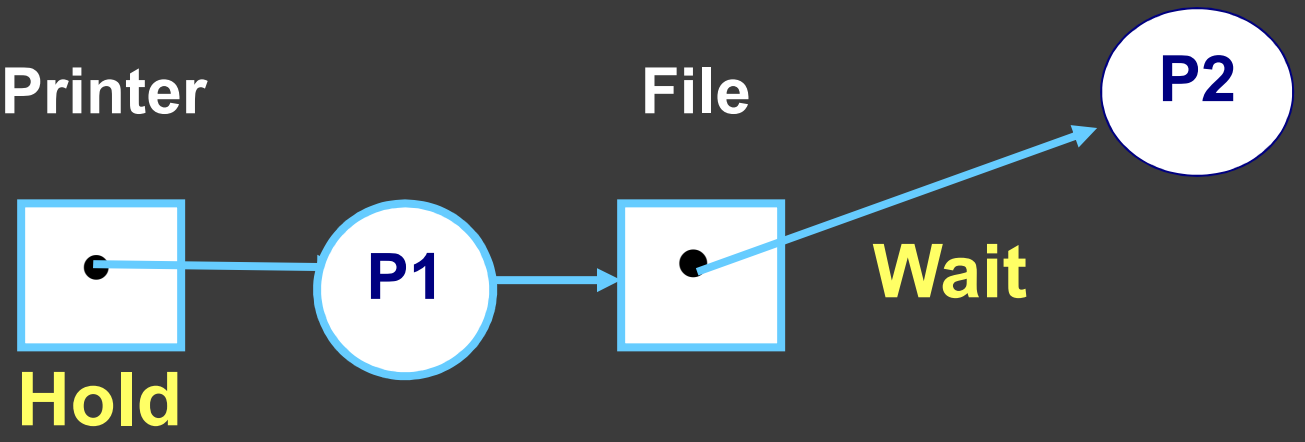
- **Mutual exclusion:**
  - Only one process at a time can use a resource.



Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously:

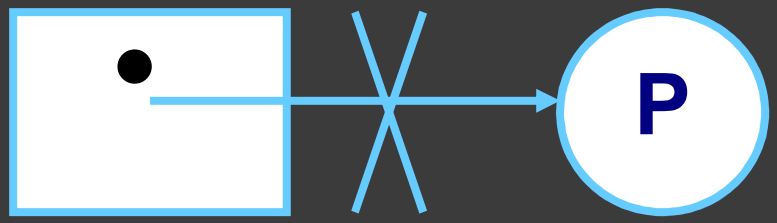
- **Hold and wait:**
  - Process holding at least one resource is waiting to acquire additional resources held by other processes.



Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously:

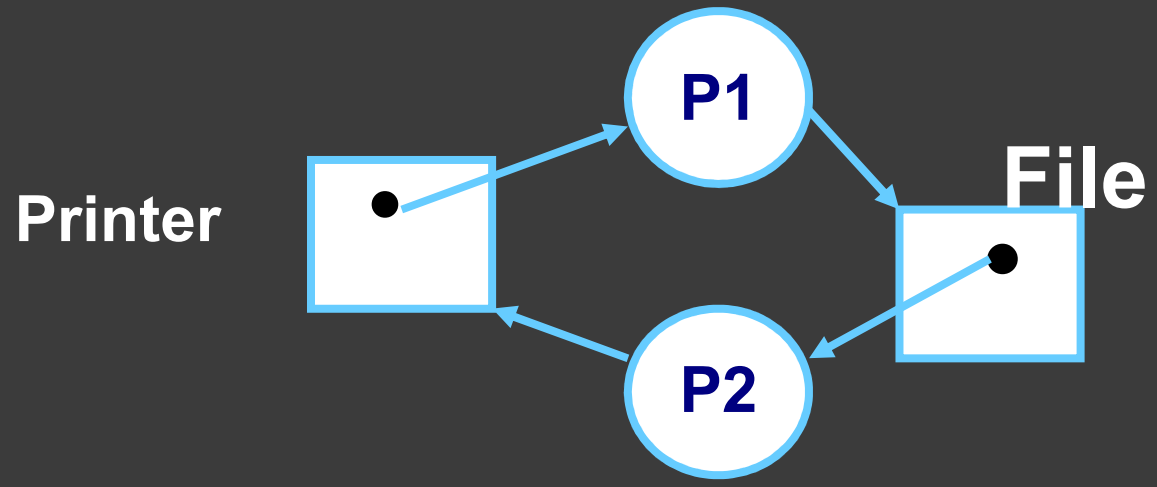
- **No preemption:**
  - Resource can only be released voluntarily by the process holding it, after that process has completed its task.



Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously:

- **Circular wait:**
  - There exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that *P0* → P1 → P2, …Pn →*P0.*



Additional characteristic

- **PROCESS IRREVERSIBLE**
  - Unable to reset to an earlier state where resources are not held.

Systems avoiding deadlocks

- Systems involving only shared resources cannot deadlock. – negates mutual exclusion
- System that aborts processes which is requesting a resource but currently being used. – negates hold and wait
- Preemption may be possible if a process does not use its resources until it has acquired all it needs – negates no preemption

Systems…

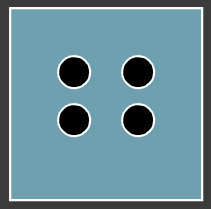
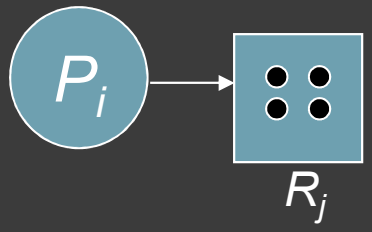
- Transaction processing systems provide checkpoints so that processes may back out of a transaction. – negates irreversible process
- System that detects or avoids deadlocks – prevent cycle.

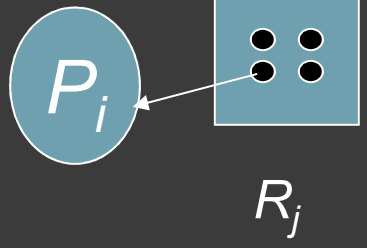
Resource-Allocation Graph (RAG)

- Consists of a set of vertices V and a set of edges E. Devised by Holt in 1972.
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the 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 $P_1 \rightarrow R_j$
- assignment edge – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (RAG)

- Process 

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

- P_i is holding an instance of R_j 

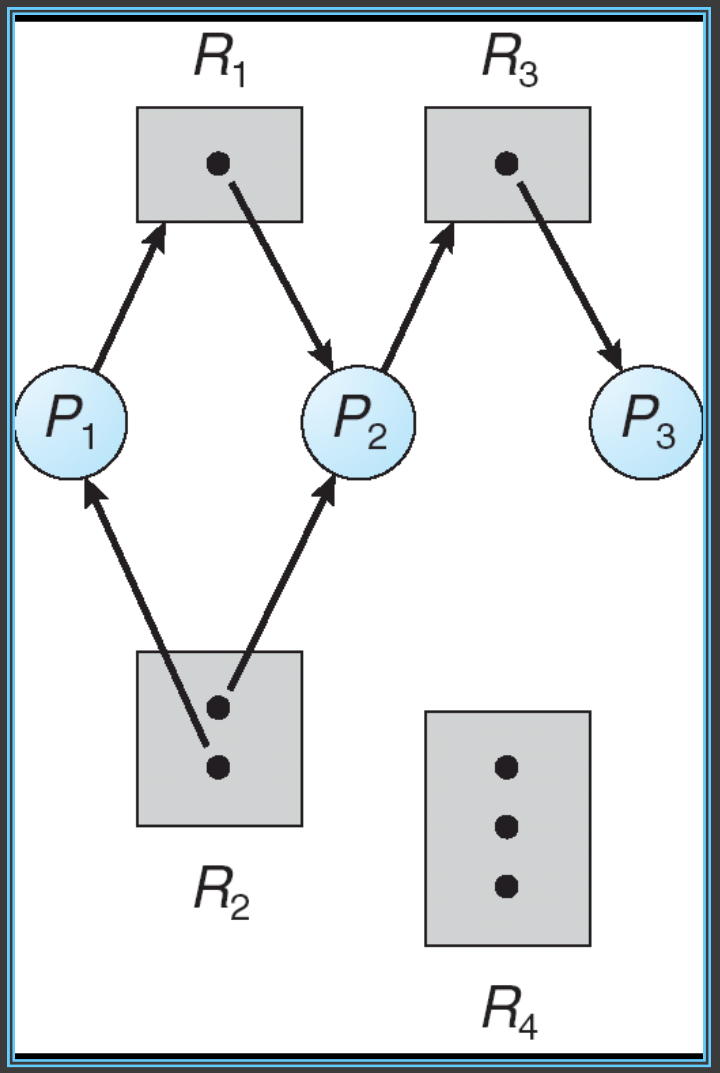
Sample Illustration:

- P = P1, P2, P3
- R = R1, R2, R3, R4
- Instances: R1=1, R2=2, R3=1, R4=3

- E={ P1 -> R1, P2 -> R3, R1 -> P2, R2 -> P1, R2 -> P2, R3 -> P3,

- Question: Draw the RAG

Solution:



Question: Is there a deadlock?

NONE.. Explanation later.

Basic Facts

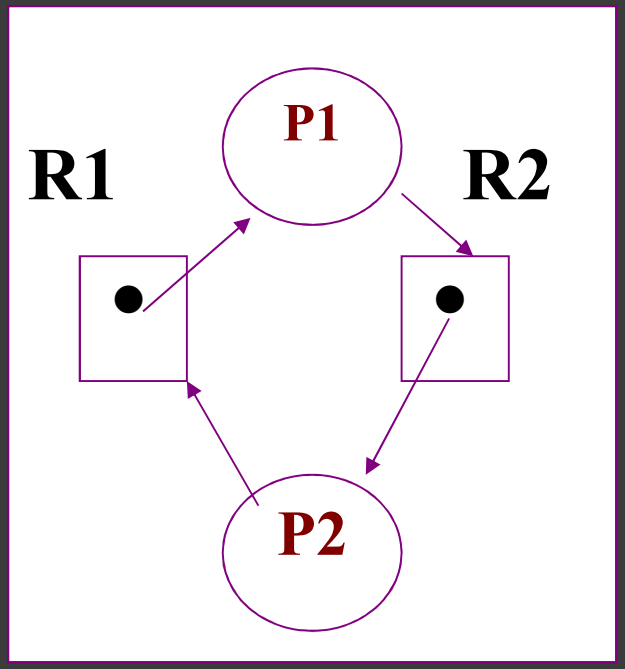
- If graph contains no cycle ⇒ then, no deadlock.
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, then possibility of deadlock only.

Exercises:

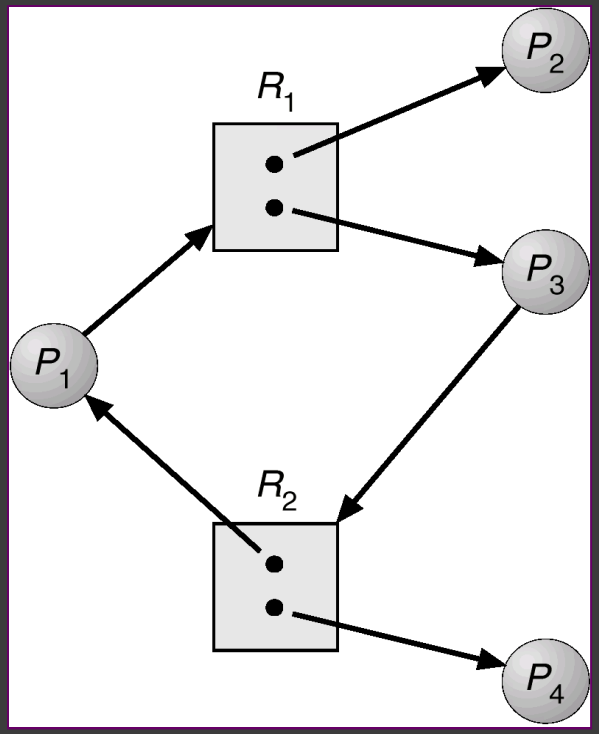
Determine:

a) The number of cycles

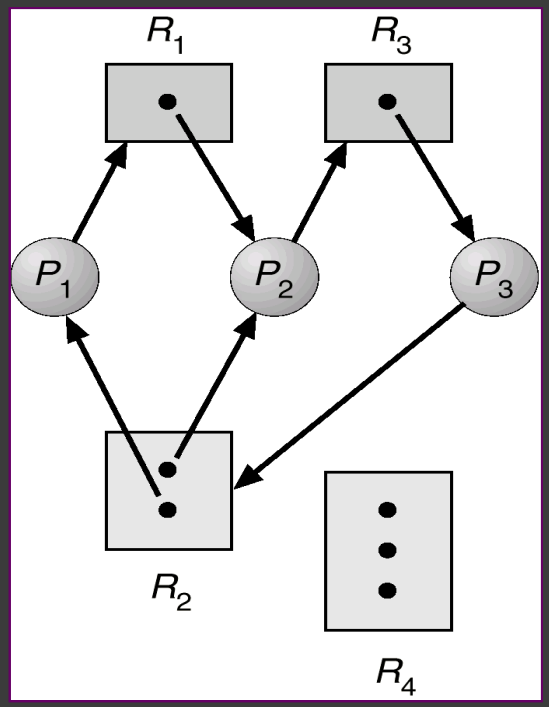
b) Is there a deadlock?



1 cycle with deadlock



1 cycle Possible deadlock

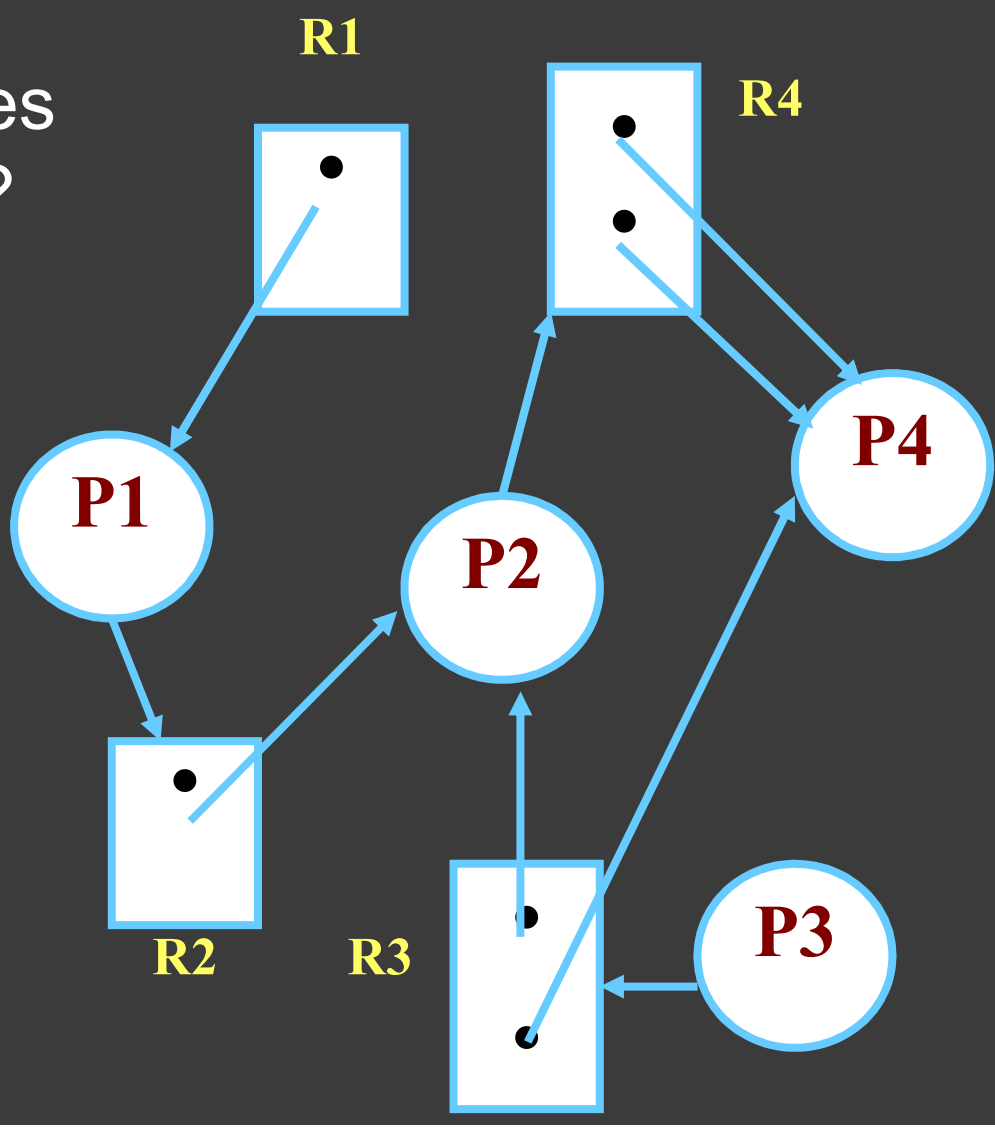


2 cycles with deadlock

Exercise#1:

Determine:
a) The number of cycles
b) Is there a deadlock?

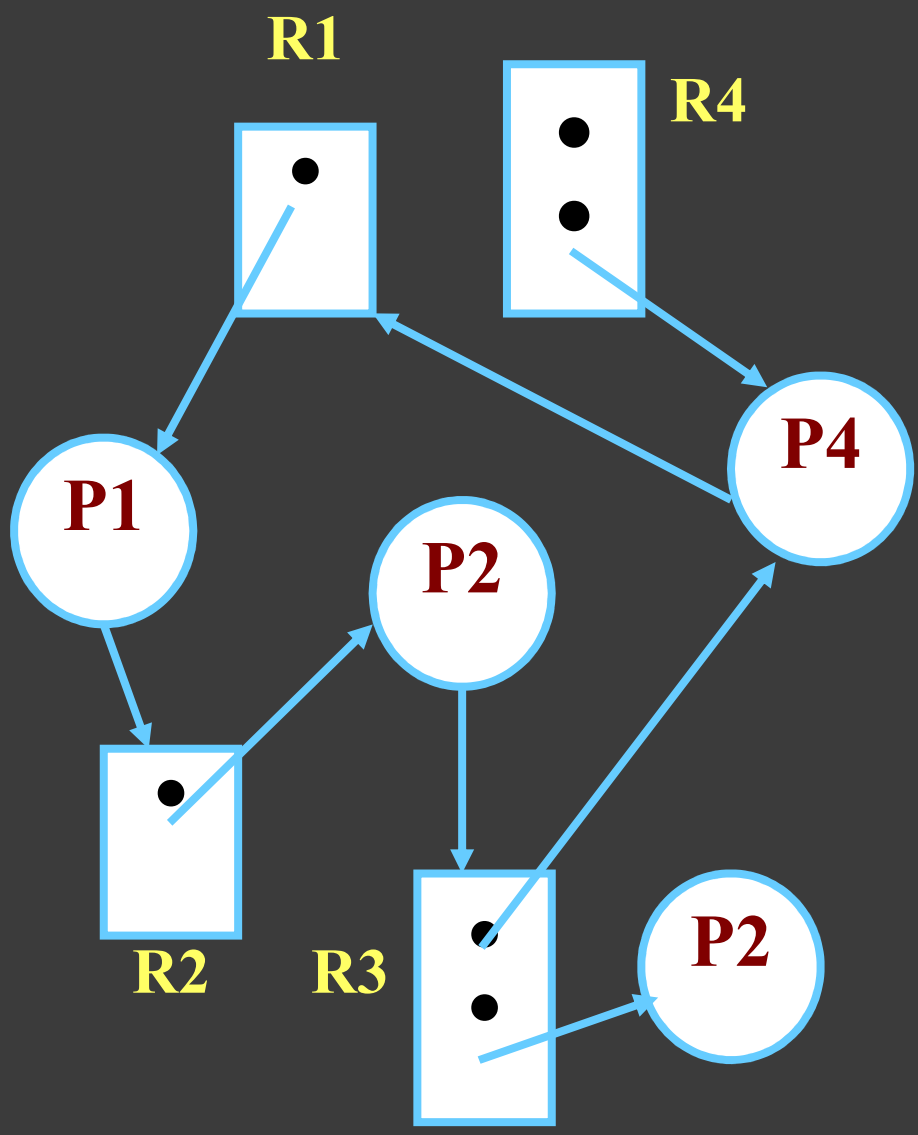
- No cycle
- Therefore, **no Deadlock**



Exercise#2:

Determine:
a) The number of cycles
b) Is there a deadlock?

- **1 cycle**
- **Possibility of Deadlock**



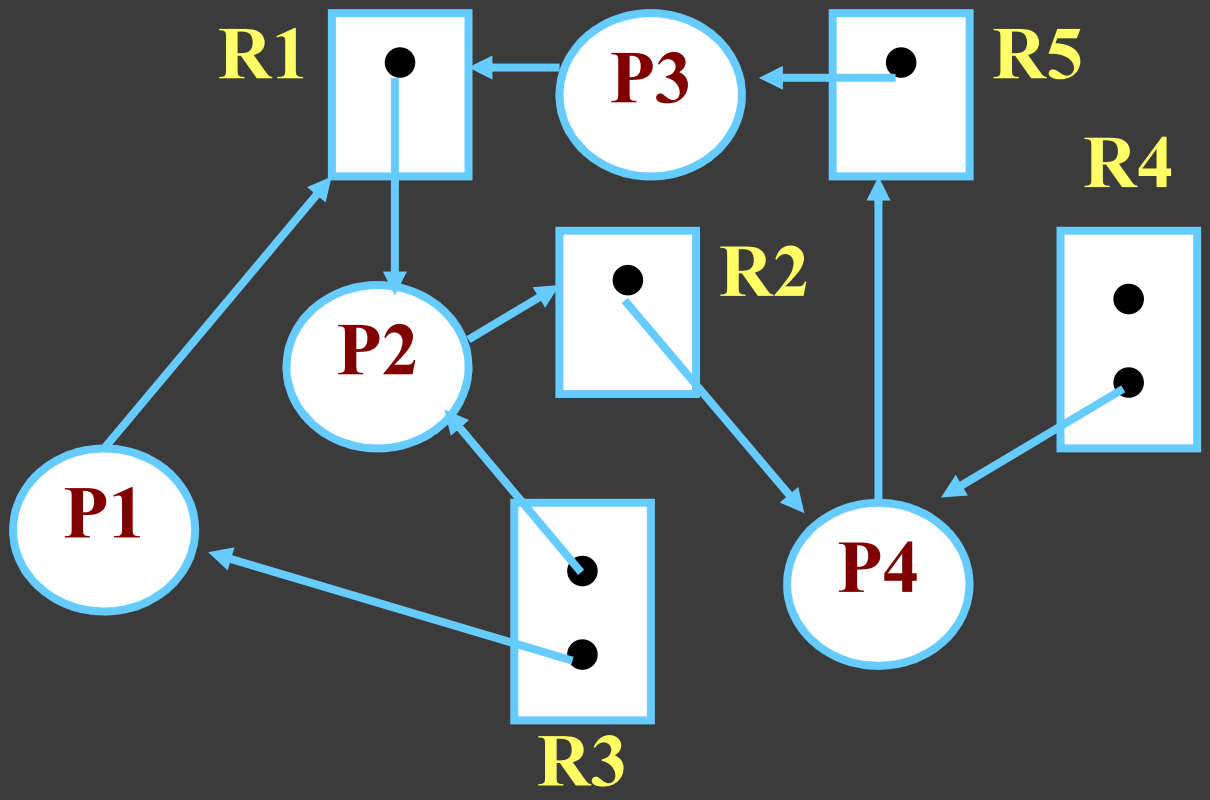
Exercise#3:

Determine:

a) The number of cycles

b) Is there a deadlock?

- **1 cycle**
- **With Deadlock**



Exercise#4:

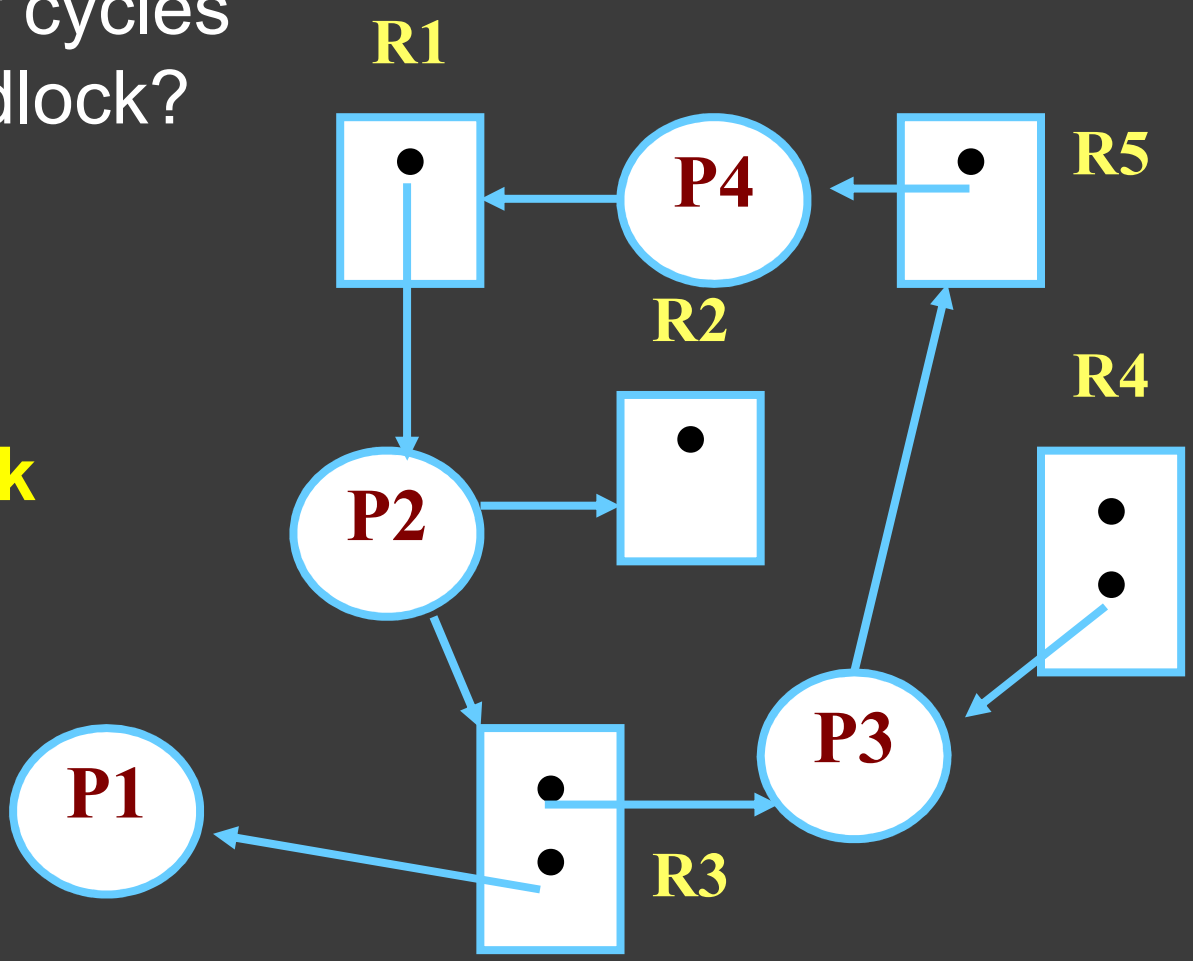
Determine:

a) The number of cycles

b) Is there a deadlock?

1 cycle

Possible deadlock



Methods for handling deadlocks

1. **Ostrich Solution:** Ignore the problem and pretend deadlocks never occur in the system. Used by most OS, including Unix.

Different people react to this strategy in different ways:

- *Mathematicians:* find deadlock totally unacceptable, and say that it MUST be prevented at all costs.
- *Engineers:* ask how serious it is, and do not want to pay a penalty in performance and convenience.

Methods for handling deadlocks

2. **Ensure that the system will never enter a deadlock state**.

- **Deadlock prevention** – Design a system in such a way that the 4 conditions leading to deadlock will never occur.

- **Deadlock avoidance** – make a decision dynamically checking whether the request will, if granted, potentially lead to a deadlock or not.

Methods for handling deadlocks

3. **Allow the system to enter a deadlock state and recover**

- **Deadlock detection and recovery**

1. Deadlock Prevention

- Preventing the occurrence of one of the necessary conditions leading to deadlock.
- Solves the problem of deadlock by limiting access to resources and by imposing restrictions on processes.

a. Mutual Exclusion

- This condition cannot be required for non-sharable resources but is a must for sharable resources.

b. Hold and Wait

- Must guarantee that whenever a process requests a resource, it does not hold any other resources. A process is blocked until all requests can be granted simultaneously.
- 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

c. No preemption

- If a process that is holding some resource 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 new ones that it is requesting.

d. Circular Wait

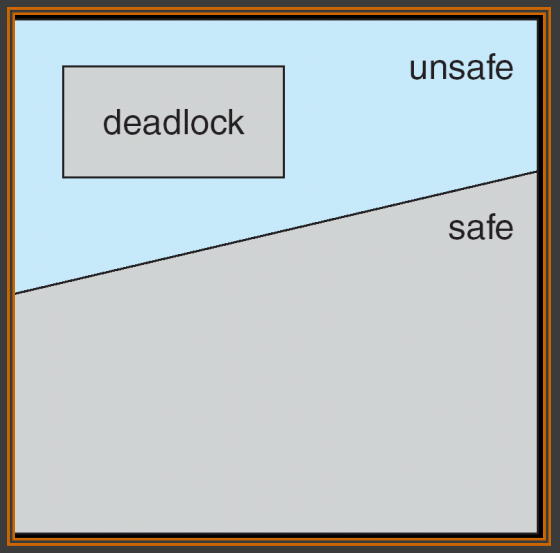
- Impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

2. Deadlock Avoidance

- Requires that the system has some additional *priori* (prior information) available.
- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need.
- 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.

Basic Facts

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



Ways to avoid deadlock by careful resource allocation

1. Resource trajectories
   – use of RAG representation

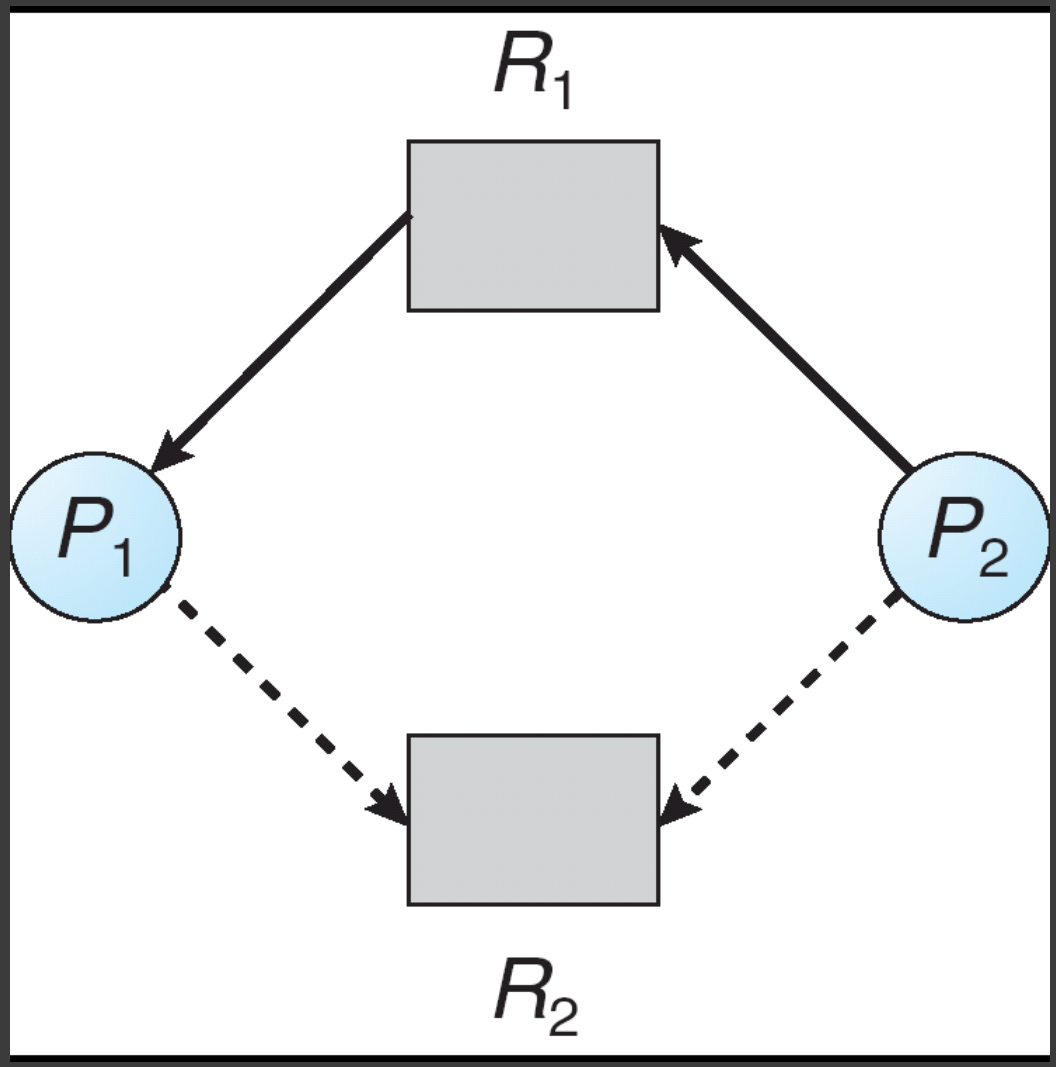
2. Safe/Unsafe states
   – for single instance of resource type. Use resource allocation

3. Dijkstra's Banker's Algorithm
   - for multiple instances of a resource type

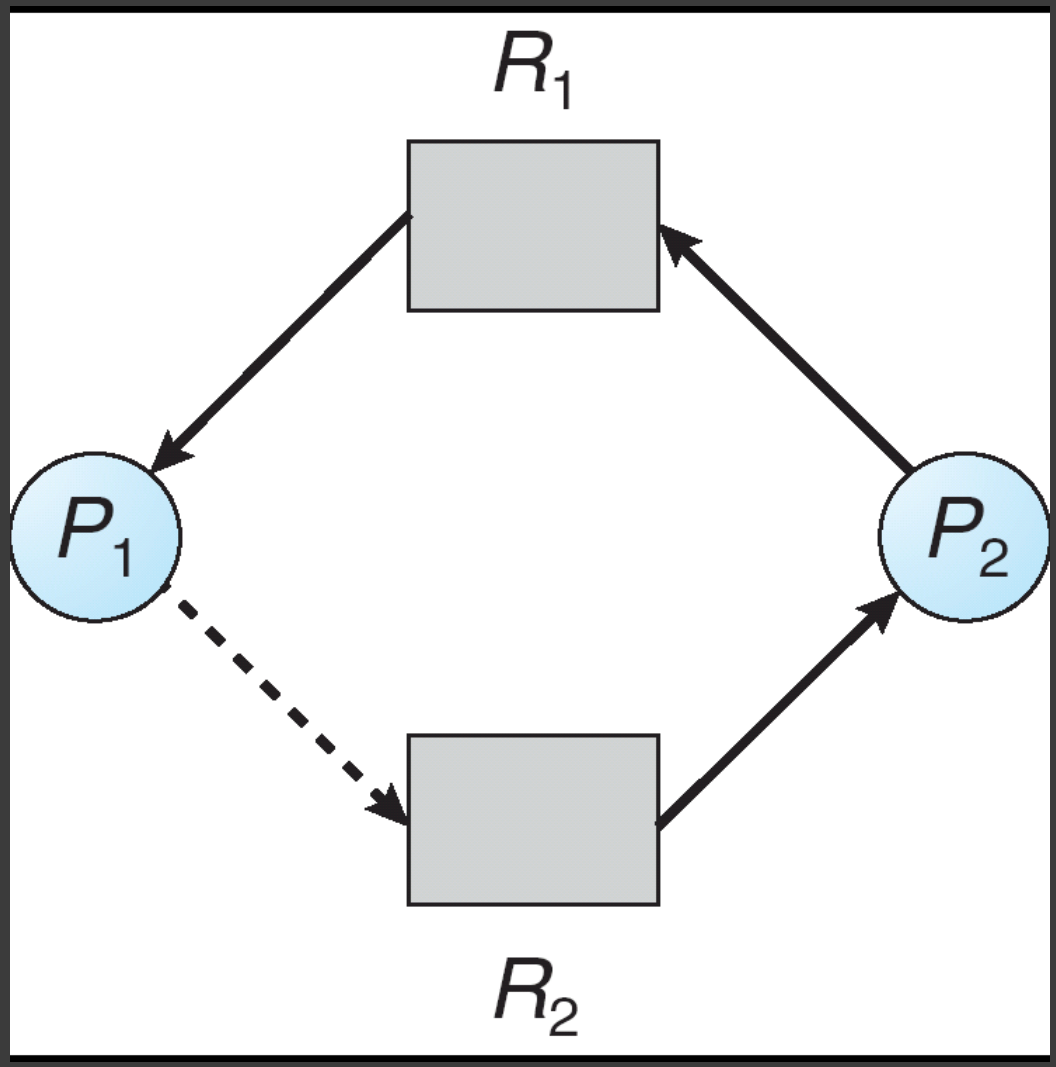
Resource-Allocation Graph Scheme

- *Claim edge:* $P_i \rightarrow R_j$ indicates process P_i requests for resource R_j; and represented by a dashed line.
- Claim edge converts to request edge when a process requests a resource.
- Request edge converts to an assignment edge when the resource is allocated to the process.
- When a resource is released by a process, the assignment edge is removed.

Resource Trajectory



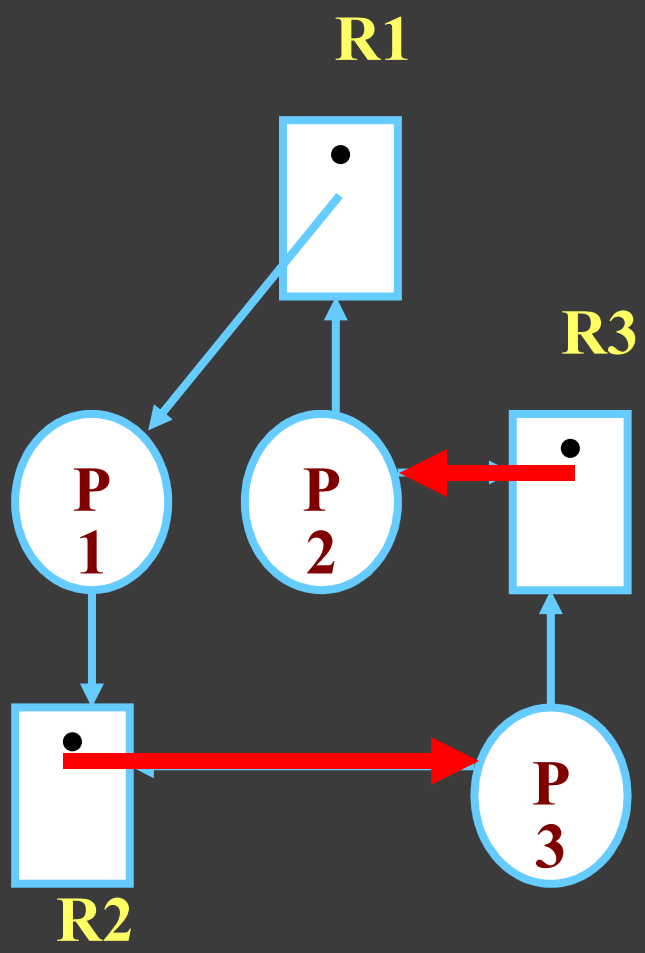
Resource Trajectory



Sample #1

- Given the RAG of the system (Initially on a safe state, grant a resource allocation request if and only if doing so will not introduce a cycle in the graph. (taking into consideration all assignment, request and claim edges). Otherwise, the requesting process has to wait.

Sample Algorithm



1. Operation : P2 requests R3
   Action : *Request Granted*
   System State : **SAFE**

2. Operation : P3 requests R2
   Action : *REQUEST DENIED!!!*
   System State : **Leads to deadlock**

Sample #2

- Consider a system with 15 identical instances of a resource

Process	Max Need	Curr Alloc	Remaining
P0	14	5	**9**
P1	10	5	**5**
P2	3	2	**1**

- Available resource: 15 – (5+5+2) = **3**
- Is it safe?
  - **Yes with a safe sequence of <P2, P1, P0>**
- What if P0 requests two more resources, will it be granted?
  - **No, OS cannot satisfy remaining needs of all processes. Unsafe**.

Sample #3

- Total instances: 8

Process	Max Need	Curr Allo	Remaining
P0	5	3	
P1	8	1	
P2	7	2	

- Is it safe or unsafe? What is the safe sequence?

Sample #4

- Total instances: 8

Process	Max Need	Curr Allo	Remaining
P0	5	3	
P1	8	1	
P2	7	2	

- Is it safe or unsafe? What is the safe sequence?

Banker's Algorithm

- 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* x *m* matrix. If *Max* [*i,j*] = *k*, then process P_i may request at most *k* instances of resource type R_j.
- ***Allocation:*** *n* x *m* matrix. If Allocation[*i,j*] = *k* then P_i is currently allocated *k* instances of R_j
- ***Need:*** *n* x *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*]**

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 and *i* such that both:

   (a) *Finish* [*i*] = *false*

   (b) $Need_i \leq Work$

   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*] = true** for all *i*, then the system is in a safe state.

Example of Banker's Algorithm

- There are 3 resource types:

  A (10 instances), *B* (5 instances), and *C* (7 instances).

- Snapshot at time T_0:

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

Need = Max – Allocation

- Is the system safe?
  - Safe: < P_1, P_3, P_4, P_2, P_0>

Example: P_1 Request (1,0,2)

- Check that Request ≤ Available (that is, (1,0,2) ≤ (3,3,2) ⇒ true.

	Allocation	*Need*	*Available*
	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 < P_1, P_3, P_4, P_0, P_2> 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 and Recovery

- Implies no conscious effort to prevent/avoid deadlocks

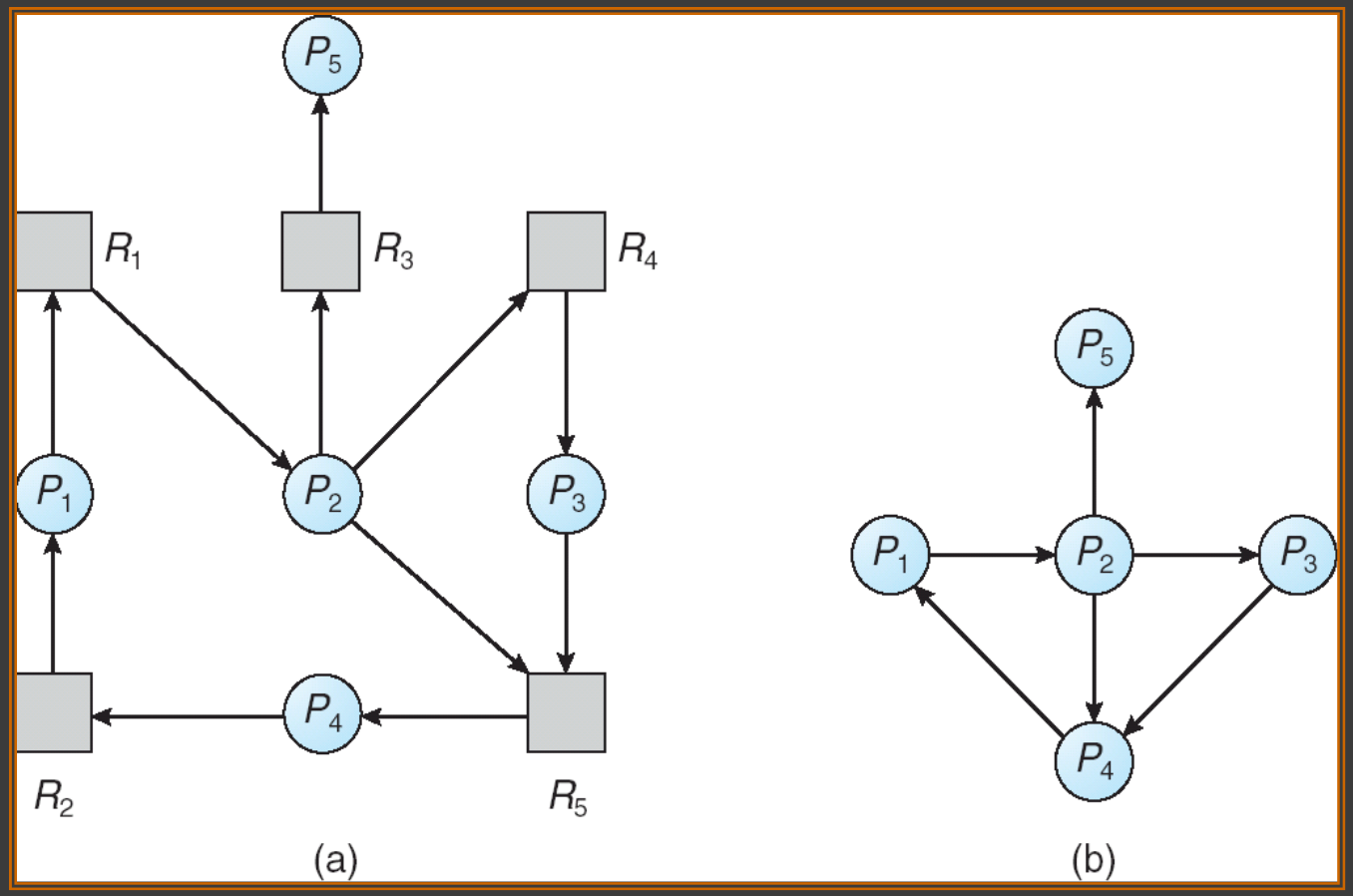
- Implementation:
  - At certain times during processing, the system executes an algorithm to determine whether or not a deadlock has occurred.
  - If so, an attempt to recover from the deadlock is made.

Detection Algorithm

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.

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* 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* $[i_j] = k$, then process P_i is requesting *k* more instances of resource type. R_j.

Detection Algorithm

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively Initialize:

   (a) *Work* = *Available*

   (b) For *i* = 1,2, …, *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.

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:

	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	

- Sequence <P_0, P_2, P_3, P_1, P_4> will result in *Finish*[*i*] = true for all *i*. Therefore, no deadlock

Example (cont.)

- *What if* P_2 requests an additional instance of type *C*?

	Request
	A B C
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

- 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 deadlock detection?
- Factors to consider:
  - How often a deadlock is likely to occur
  - How many process will need to be rolled back
- Approaches
  - Execute the algorithm at predetermined time interval
  - Execute the algorithm whenever a resource request cannot be granted immediately.

Recovery from deadlock:

- Who's responsibility?
  - Inform the operator and let him deal with the deadlock manually
  - Allow the system to recover from deadlock automatically.

Approach: I. 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 process
- How long process has computed and how much longer to completion
- Resources the process has used
- Resources process needs to complete
- How many process will need to be terminated?
- Is process interactive or batch?

Approach: II. 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

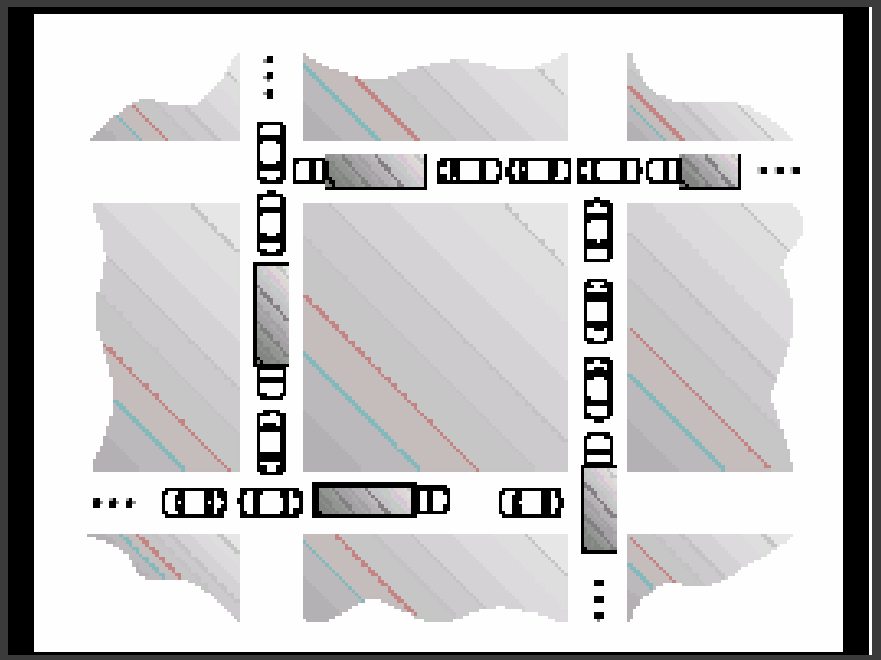
Combined Approach to Deadlock Handling

- 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.
- Use most appropriate technique for handling deadlocks within each class.

Exercises:

1. Consider the traffic deadlock depicted in the figure:
    - Show that the four necessary conditions for deadlock indeed hold in this example.
    - State a simple rule that will avoid deadlock in this system.



Exercises:

2. Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock-free?

Exercises:

3. Cinderella and the Prince are getting divorced. To divide their property, they have agreed on the following algorithm. Every morning, each one may send a letter to the other's lawyer requesting one item of property. Since it takes a day for letters to be delivered, they agreed that if both discover that they have requested the same item on the same day, the next day they will send a letter canceling the request. Among their property is their *dog (Woofer), doghouse, their canary (Tweety) and Tweety's cage*.

The animals love their houses, so it has been agreed that any division of property separating an animal from its house is invalid, requiring the whole division to start over from scratch. Both Cinderella and the prince desperately want Woofer. So they can go on (separate) vacations, each spouse has programmed a personal computer to handle the negotiation. When they came back from vacation, the computers were still negotiating. Why? Is deadlock possible? Is starvation possible? Discuss.

4. Consider the following and answer the questions that follow:

   P = P1, P2, P3, P4, P5

   R = R1, R2, R3, R4

   Instances: R1=2, R2=3, R3=1, R4=1

E={ R3 -> P1, P1 -> R2, R2 -> P2, R2 -> P4, R1 -> P2, R1 -> P3, P4 -> R4}

Answer the ff. questions:

   1: Is there a cycle? (Y/N)

   2: Is there a deadlock? (Y/N)

   3: Will P1's request for R2 be granted? (Y/N)

   4: Will P4's request for R4 be granted? (Y/N)

   5: How many available instances are there for R2?

5. Eight (8) processes P_0 through P_7 with 4 resource types: *A* (10 instances), *B* (5), *C* (11) and D (6).

	Allocation	*Max*
	A B C D	*A B C D*
P_0	0 1 2 1	7 4 9 2
P_1	2 1 0 0	3 2 2 0
P_2	0 0 2 0	9 0 7 2
P_3	2 1 0 2	2 2 4 3
P_4	0 0 2 0	4 3 6 3
P_5	2 0 2 0	2 1 3 2
P_6	2 1 1 1	5 2 2 1
P_7	1 0 2 0	7 4 8 5

Question: Is the system safe? Justify your answer.

End of presentation