Operating System
Session 5

Hakim Sabzevari University
Dr.Malekzadeh

Deadlocks

Types of resources

- Generally resources can be:
  - Sharable resources: can be used by more than one process at a time. Examples are Program code and Data areas if read only.
  - Non-shareable/Exclusive resources: can only be used by one process at a time. Examples are CPU, RAM, Data areas that need to be written, and most external devices like scanners and printers. The non-sharable resources can be:
    - Preemptable resources
    - Non-Preemptable resources

Types of resources cont…

- Preemptable Resource: these resources **can be** taken away from its owner process with no ill-effect. Examples are Memory and CPU.

- Non-Preemptable: these resources **can not be** taken away from its owner process without causing the process to fail. If a process has begun to burn a CD-ROM, suddenly taking the CD recorder away from it and giving it to another process will result in a garbled CD. CD recorders are not preemptable at an arbitrary moment.

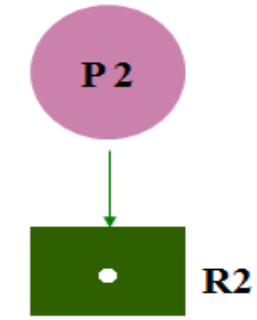
Resource utilization

- A process may request as many resources as it requires to carry out its tasks. In a normal operation, a process may utilize a resource in the following sequence:
  - Request the resource: A process requests a resource. If the resource is available at that time, the OS allocates the resource to the process. If the resource is not available at that time, the process enters a waiting state until it can acquire the resource.
  - Use the resource: The process can operate on the resource.
  - Release the resource: The process releases the resource after using it.

- The request and release of resources are system calls.

Resource Allocation Graph

- Resource Allocation Graphs (RAG)/System Resource Allocation Graphs (SRAG) are drawn in order to see the allocation relations of processes and resources easily.
  - The processes are circles.
  - The resources are squares.
  - An arc (directed line) from a process P to a resource R signifies that process P has requested the resource R but it not yet been allocated.
  - An arc from a resource R to a process P indicates that the resource R has been allocated to process P.
  - For each instances of a resource type, there is a dot in the resource square. A computer may have multiple instances of a resource type (multiple printers, pen drives, ...).



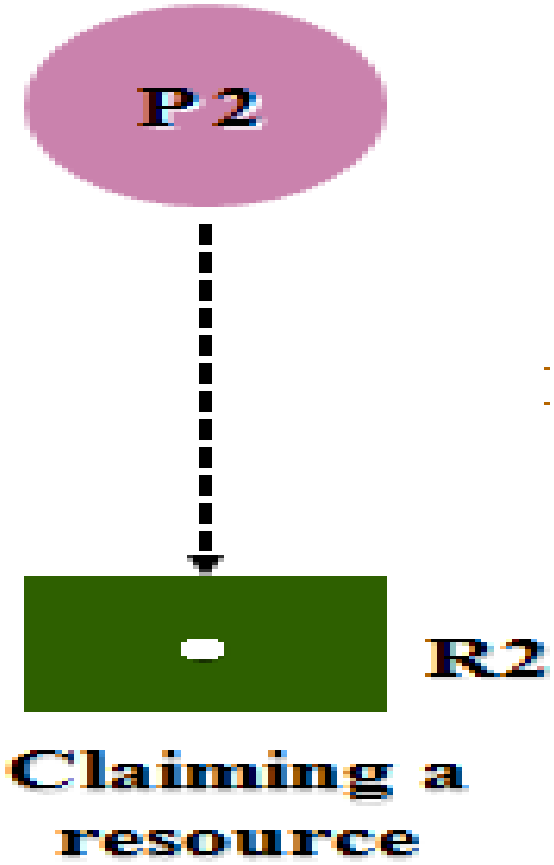
Requesting a resource

Resource Allocation Graph cont…

- Resource-Allocation Graph is a set of node vertices V and a set of edges E; **G=(V,E)**:
  - V can be 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
  - E is edge which can be (Pi, Rj) or (Ri, Pj). The E can be three types:
    - Claim edges
    - Request edges
    - Assignment edges

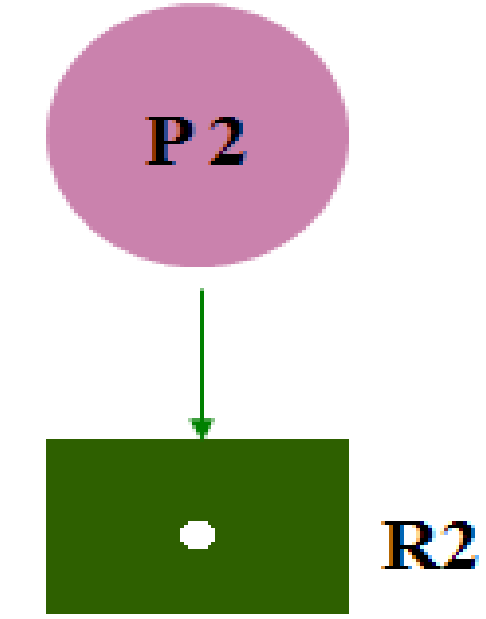
Claim edge

- Claim edge is a directed dashed line Pi → Rj indicated that process Pj may request resource Rj in the future. Claim edge converts to request edge when a process requests a resource.



Request edge

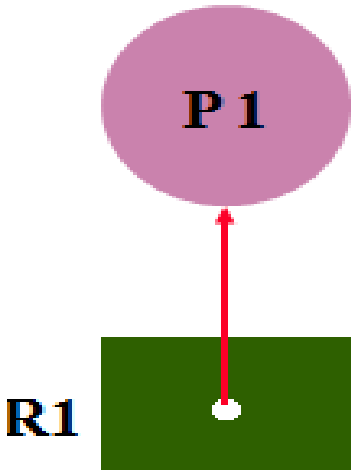
- Request edge is a directed edge $P_i \rightarrow R_j$ at which Process P_i is waiting for resource R_j. Request edge converted to an assignment edge when the resource is allocated to the process.



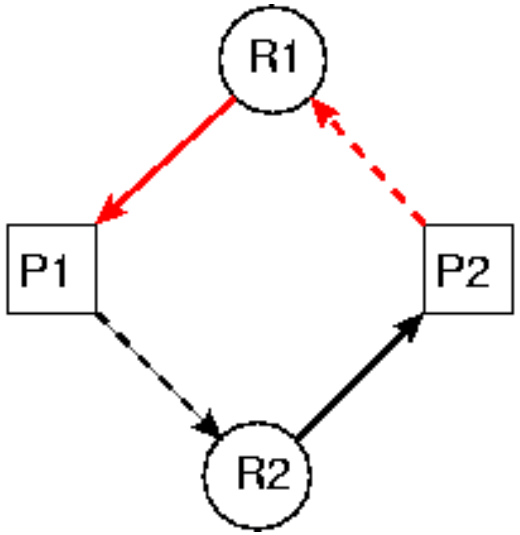
Requesting a resource

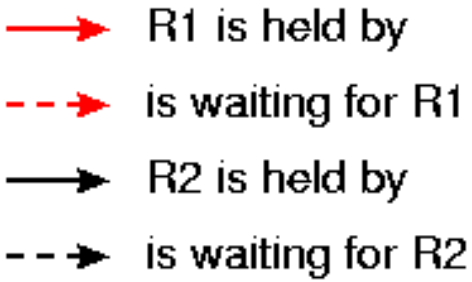
Assignment edge

- Assignment edge is a directed edge $R_j \rightarrow P_i$ at which resource R_j has been allocated to the Process P_i. When a resource is released by a process, assignment edge reconverts to a claim edge.



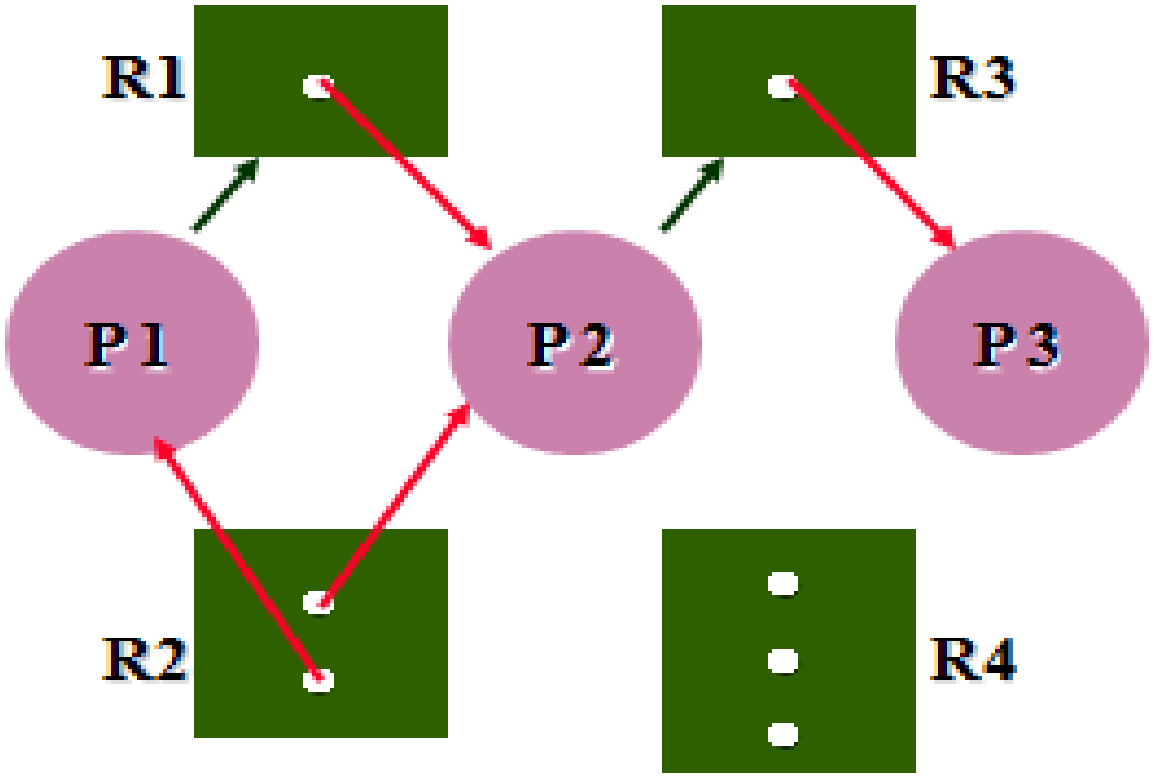
Holding a resource



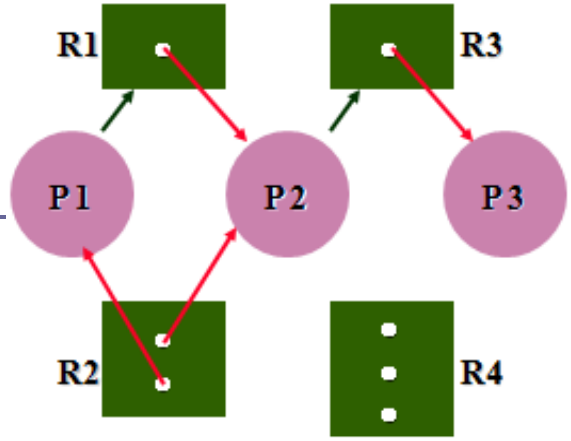


Resource Allocation Graph cont…

- A request edge points to only the square, while an assignment edge points to one of the dots in the square.



Example:



- The sets: *P*, *R*, and *E:*
  - $P = \{P_1, P_2, P_3\}$
  - $R = \{R_1, R_2, R_3, R_4\}$
  - $E = \{P_1 \rightarrow R_1, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, P_2 \rightarrow R_3, R_3 \rightarrow P_3\}$
- Process states:
  - Process P1 is holding an instance of resource type R2, and is waiting for an instance of resource type R1.
  - Process P2 is holding an instance of resource types R1 and R2, and is waiting for an instance of resource type R3.
  - Process P3 is holding an instance of resource type R3.
- Resource instances:
  - One instance of resource type R1
  - Two instances of resource type R2
  - One instance of resource type R3
  - Three instances of resource type R4

Deadlock

- Consider the two following sentence:
  - It takes money to make money.
  - You can't get a job without experience; you can't get experience without a job.

No vehicles can move forward to clear traffic jam

Example: Traffic Jam

Deadlock cont…

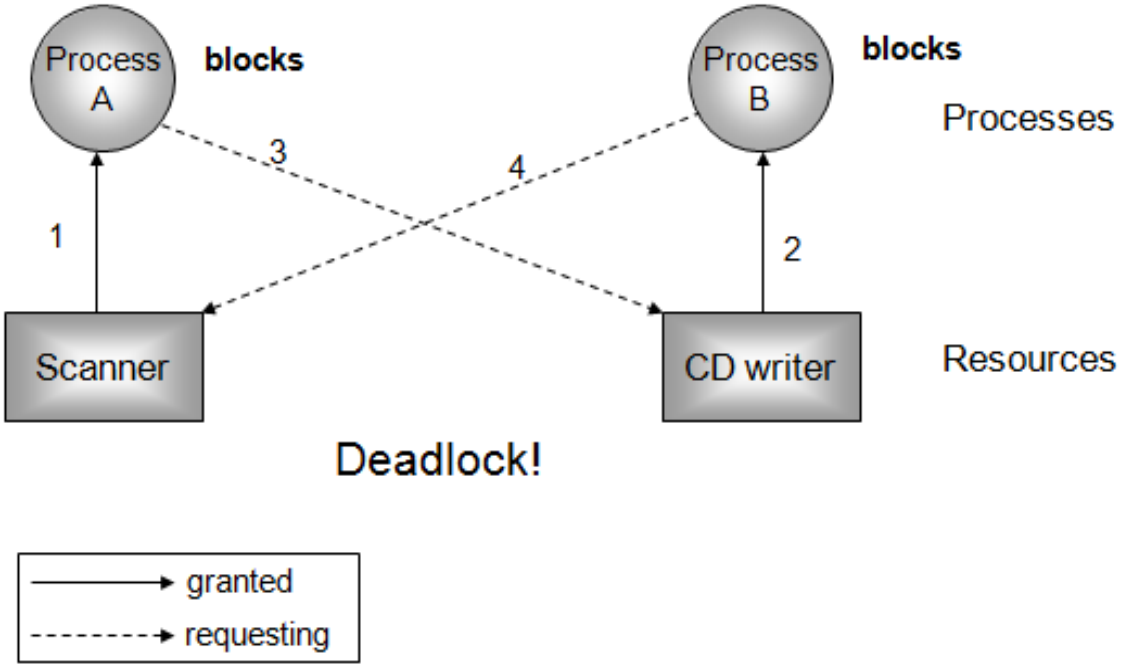
- Sometimes, a waiting process is never able to change state, because the resources it has requested are held by other waiting processes. This situation is called deadlock. Deadlock is the blocking of a set of processes that compete for system resources.

- A set of processes is in a deadlock state when every process in the set is waiting for an event (release) that can be done only by another process in that set.

Occurs when a thread waits for a condition that never occurs.

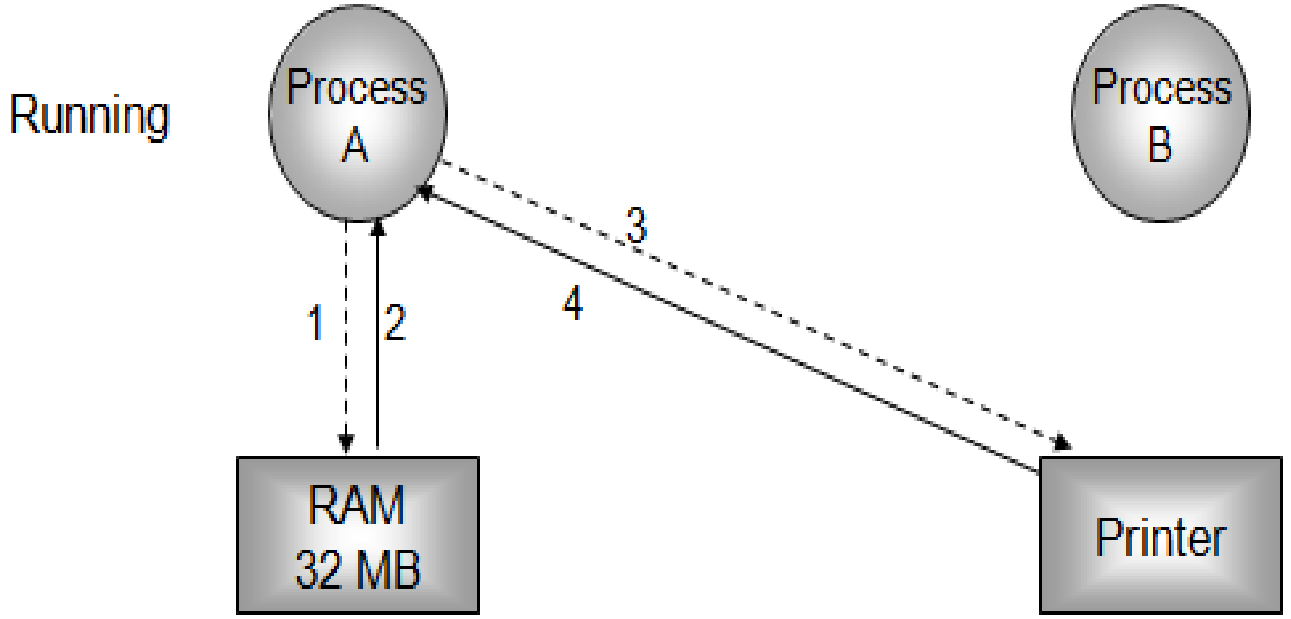
Example1:

- Consider a system with one CD Writer and one scanner. Process A requests the scanner and process B requests the CD Writer. Both requests are granted. Now A requests the CD Writer (without giving up the scanner) and B requests the scanner (without giving up the CD Writer). Neither request can be granted so both processes enter a deadlock situation.



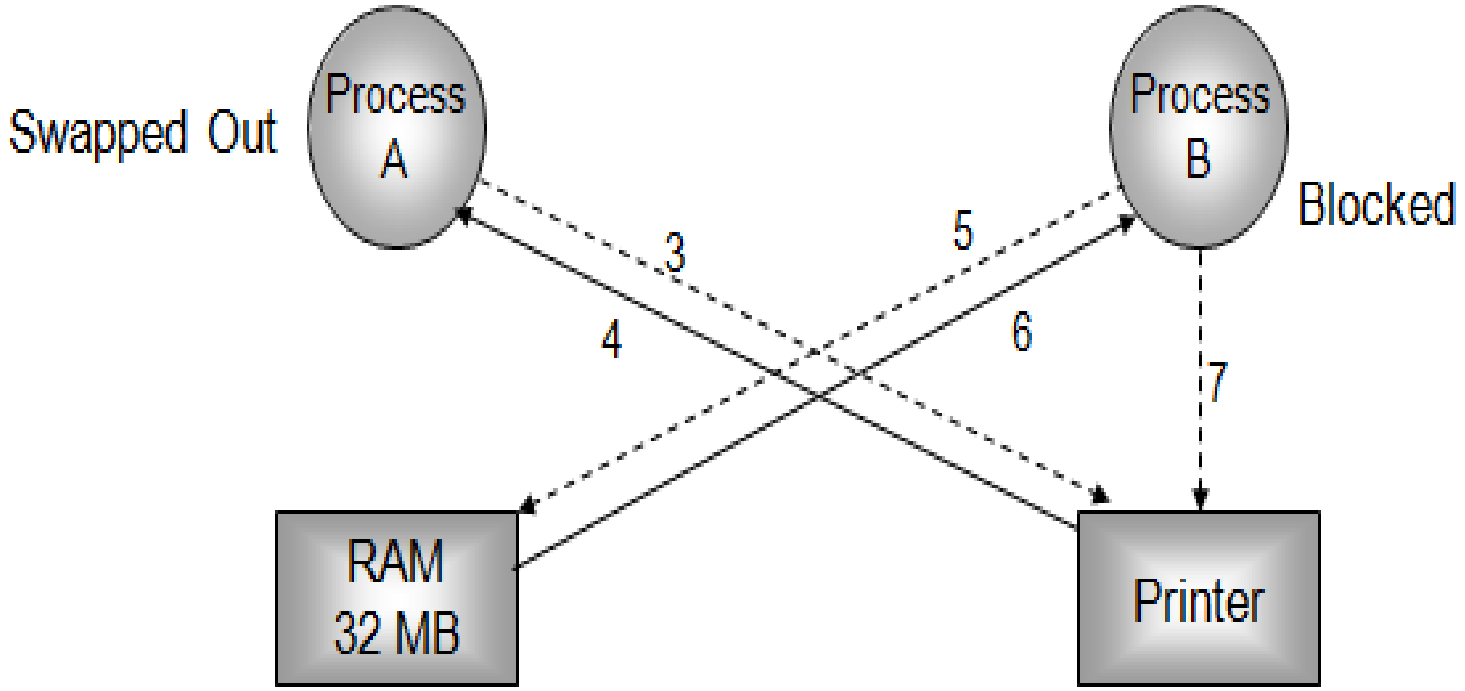
Example2:

- In the following diagram there is no deadlock since there is only one process asking for resources which are allocated to it.



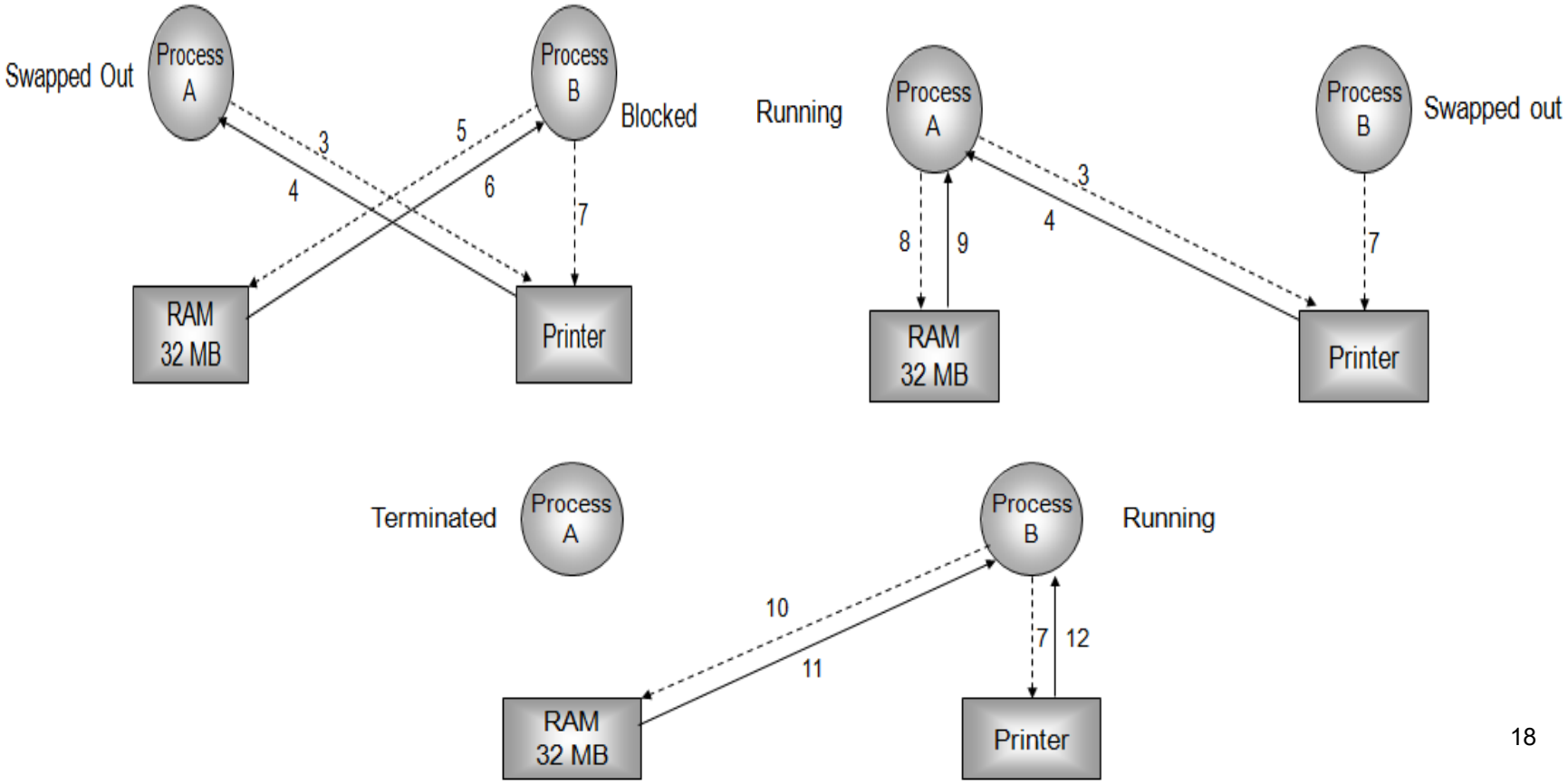
Example3:

- Now consider that A gets swapped out and B is run now. A releases Memory and B uses it. Since A is swapped out without completion of its task, it cannot run. Hence if B requests printer, it is blocked, cannot run and deadlock occurs.



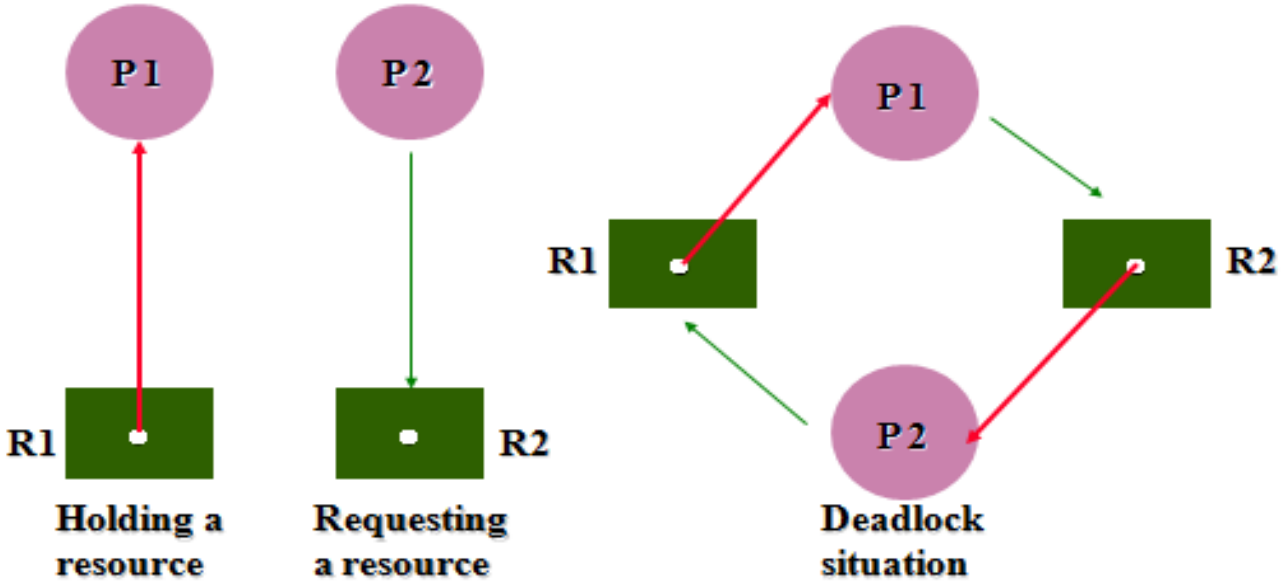
Example3 solution:

- Since the resource is Preemptable, so the solution is to take memory from B which will swap B out, then run A until completion so that it can release printer.



Deadlock conditions

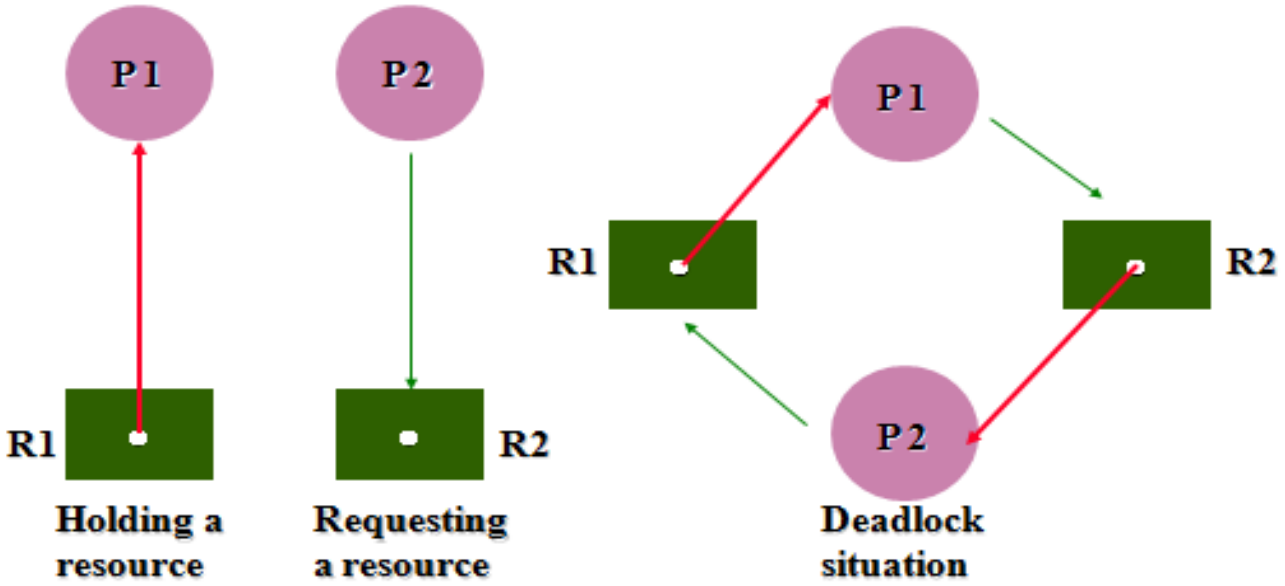
- Four conditions must be simultaneously present for a deadlock to occur:
  - Mutual Exclusion (ممانعت متقابل) (if I have it, you can't have it): if one process holds a resource, other processes requesting that resource must wait until the process releases it (only one process at a time can use the resource).
  - No Preemption (I only give it up when I'm done with it): The resources are released voluntarily; neither another process nor the OS can force a process to release a resource. To do this non-sharable resources are used.



Deadlock conditions cont…

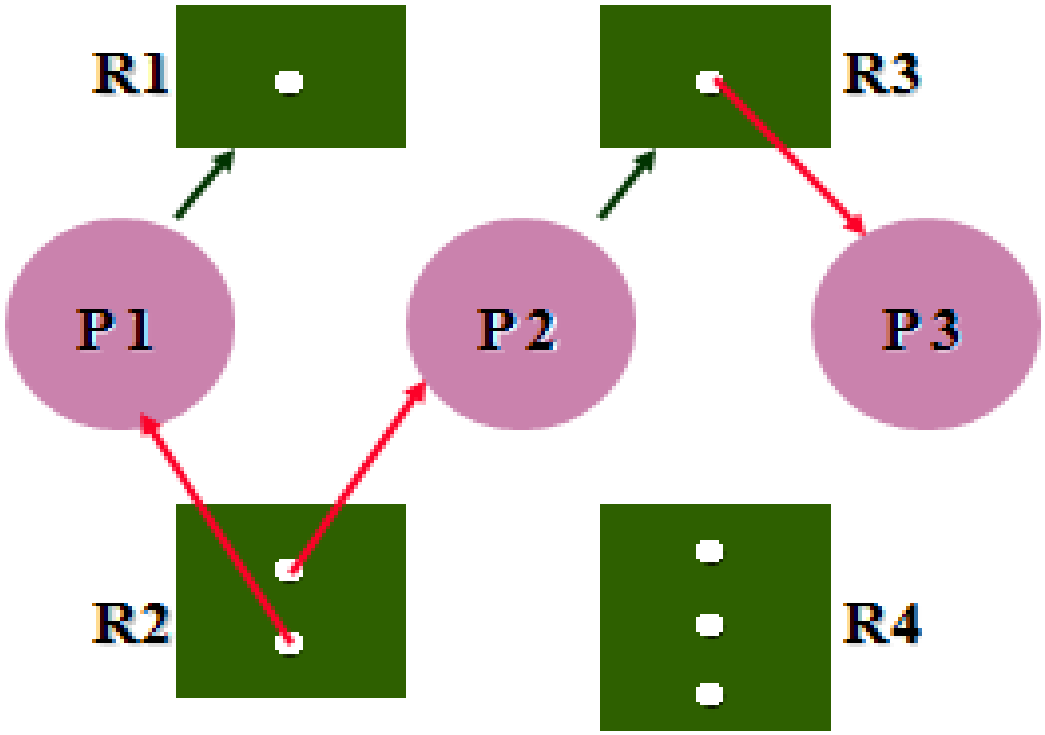
- Hold and Wait (once I get it, I can hold it even if I want something else): A process holds at least one resource and requests another resource **that is held by another process**.

- Circular Wait: A circular chain of processes exist, in which each process holds a resource requested by the next process in the chain.



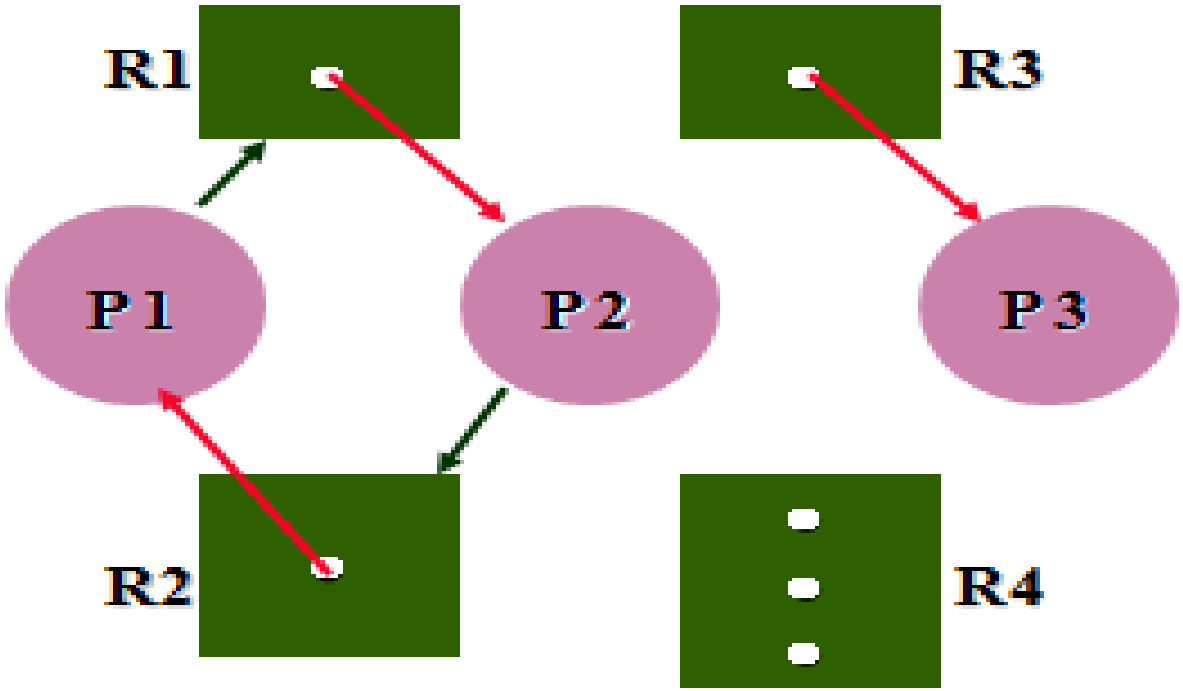
Tips to check deadlock

- If no cycle exists in the resource allocation graph, there is no deadlock.
- If there is a cycle, there **may be** a deadlocked state.

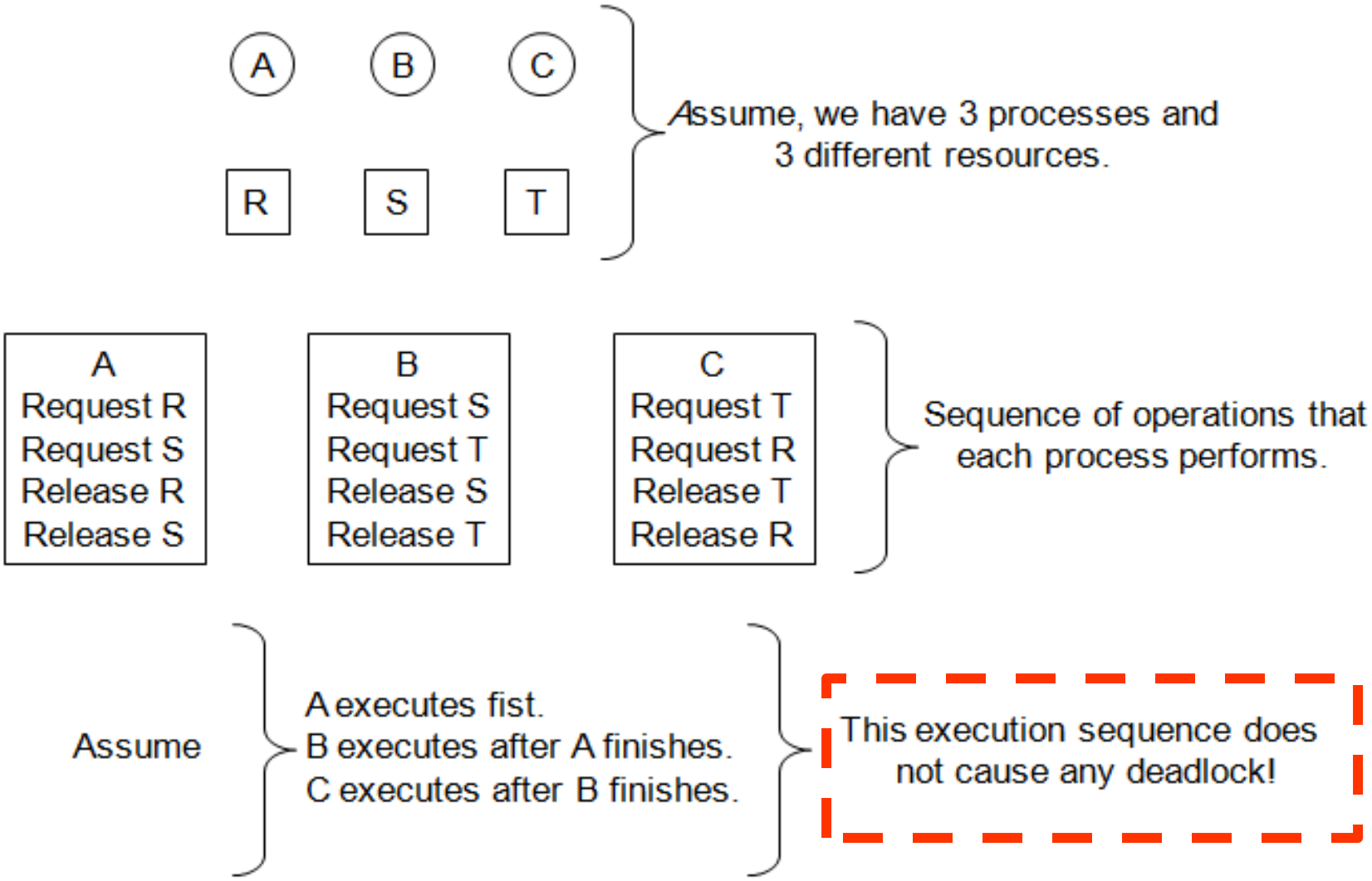


Tips to check deadlock cont…

- If there is a cycle in the graph and each resource has only one instance, then there is a deadlock.
- If there is a cycle in the graph and each resource has more than one instance, then deadlock **MAY** exist.



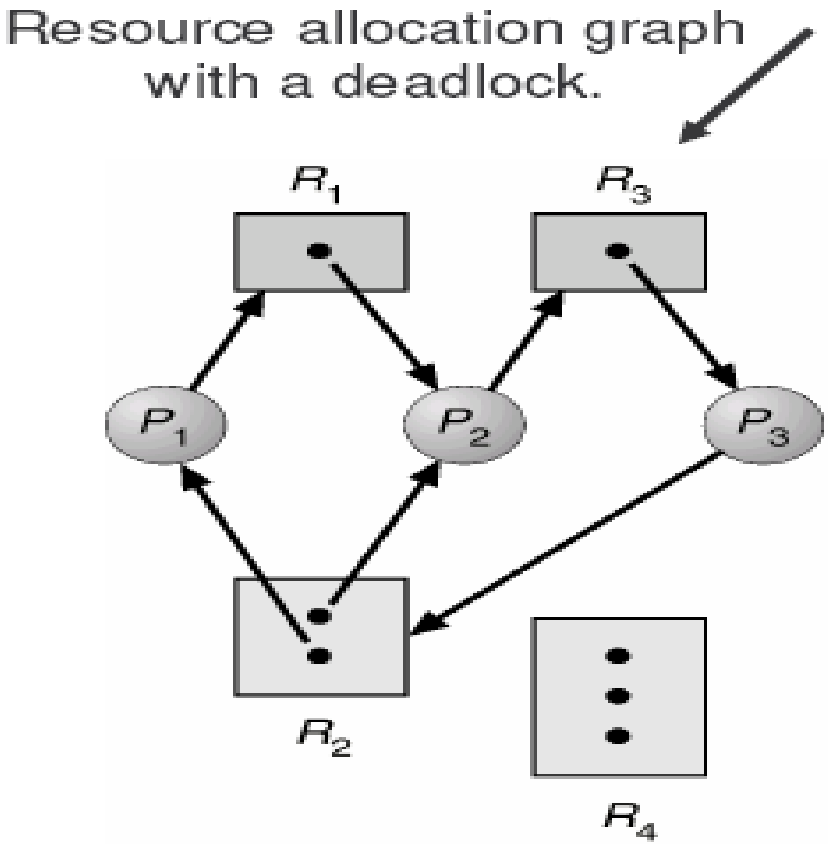
Example1:


A
B
C
R
S
T
Assume, we have 3 processes and 3 different resources.
A
Request R
Request S
Release R
Release S
B
Request S
Request T
Release S
Release T
C
Request T
Request R
Release T
Release R
Sequence of operations that each process performs.
Assume
A executes fist.
B executes after A finishes.
C executes after B finishes.
This execution sequence does not cause any deadlock!

Example2:

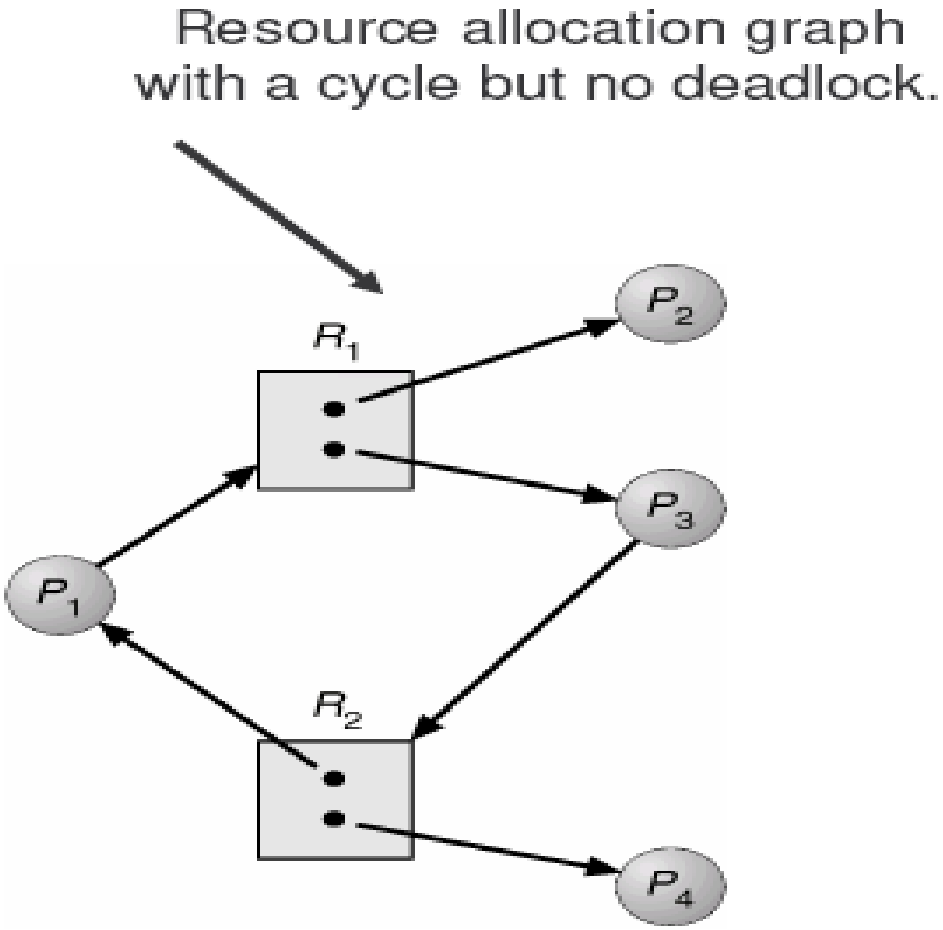
- Process P2 is waiting for the resource R3, which is held by process P3. Process P3, on the other hand, is waiting for either process P1 or process P2 to release resource R2. In addition, process P1 is waiting for process P2 to release resource R1.

- Thus, P1, P2, and P3 are deadlocked.



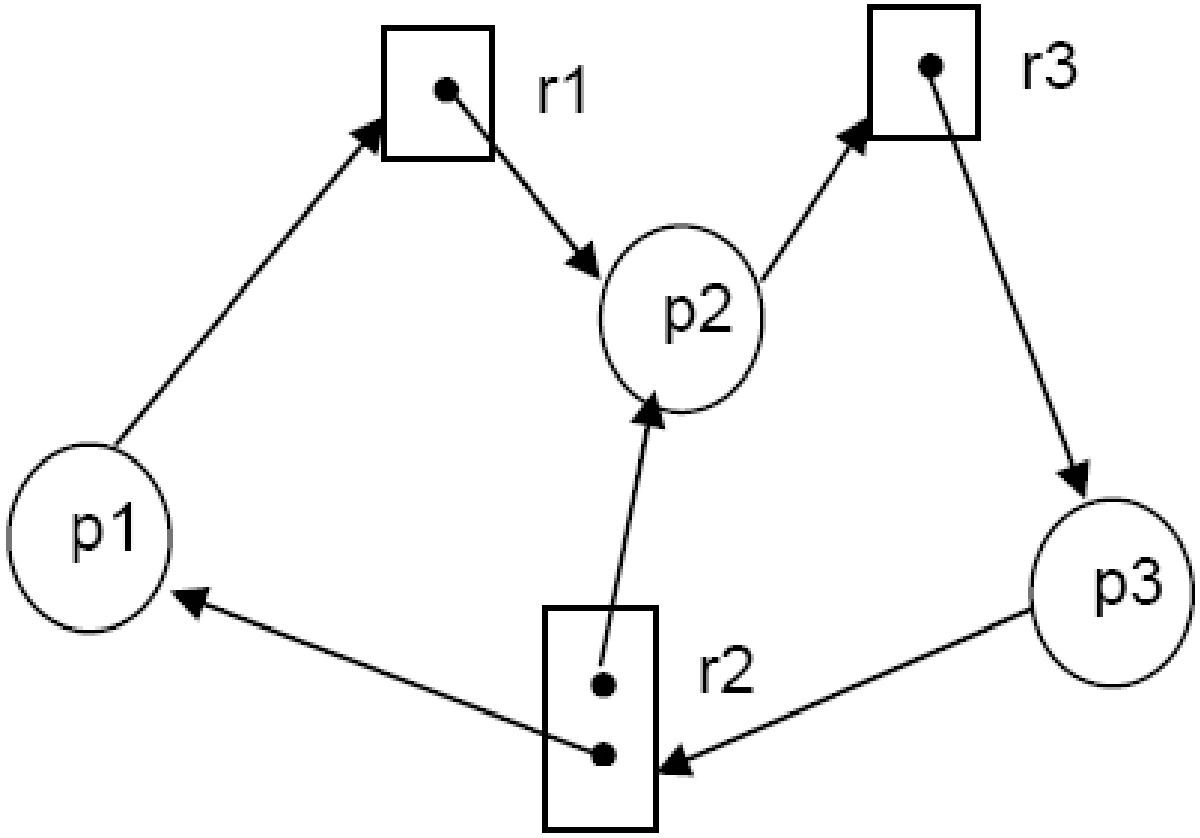
Example3:

- Cycle without deadlock:



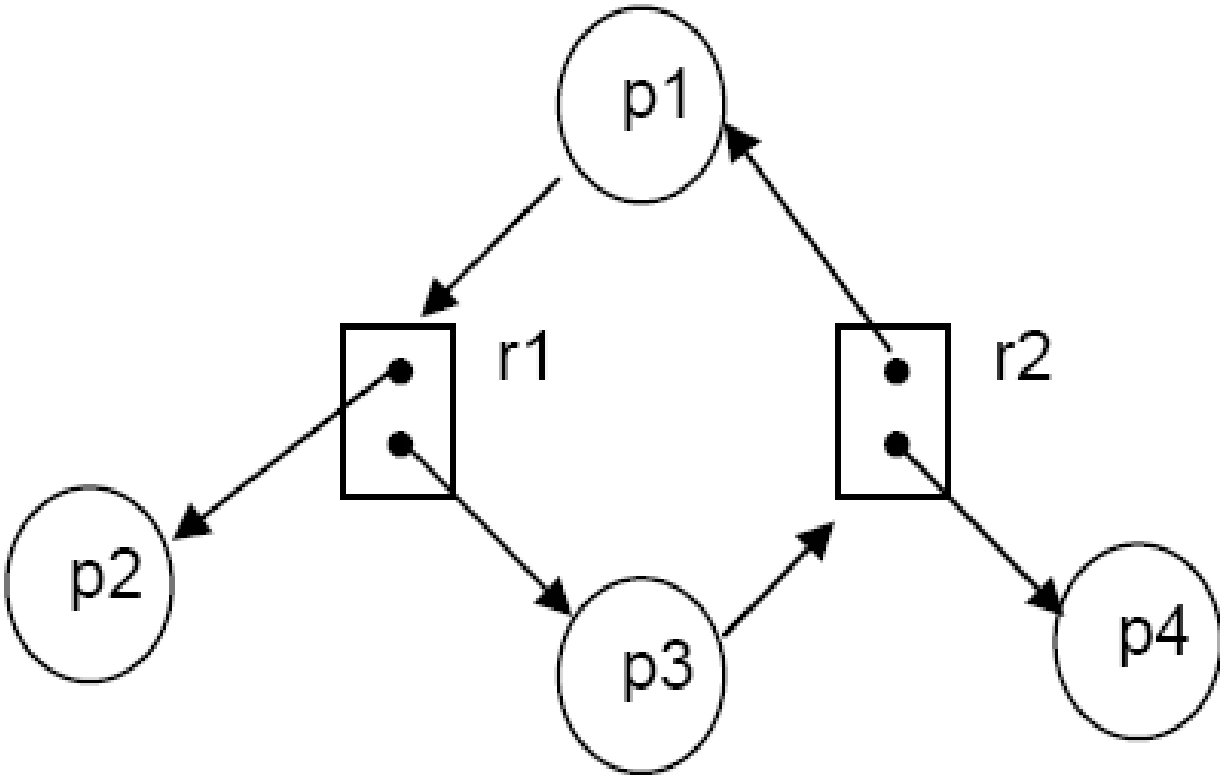
Example4:

- cycle with deadlock: There are cycles, so a deadlock may exist. The p1, p2 and p3 are deadlocked.



Example5:

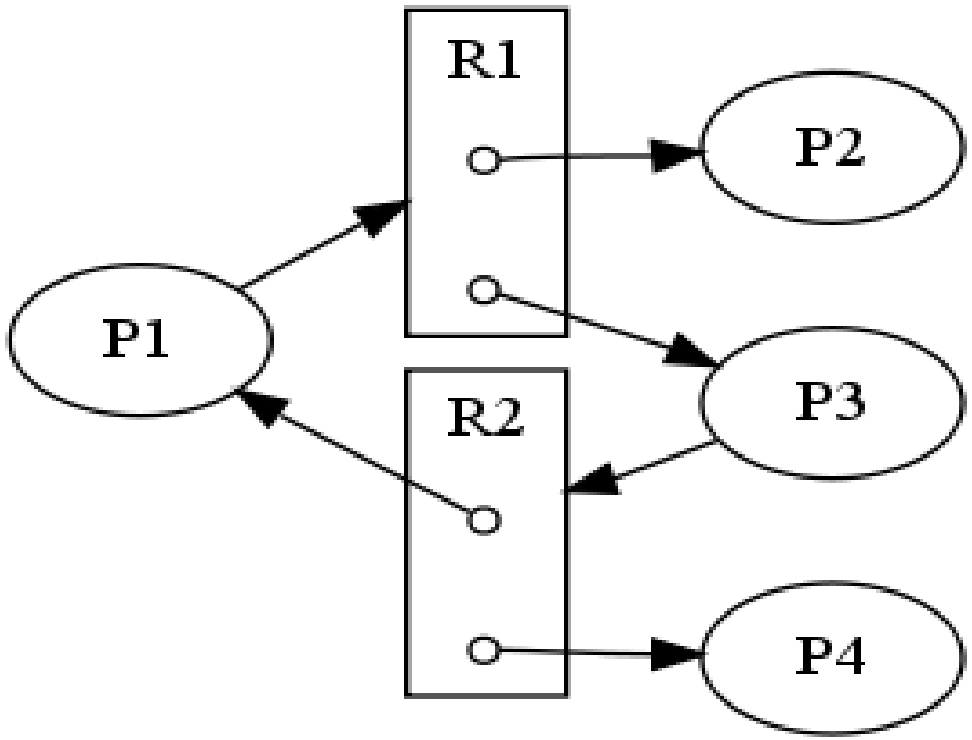
- cycle without deadlock: There is a cycle, however there is no deadlock. If p4 releases r2, r2 may be allocated to p3, which breaks the cycle.



Example6:

- with a cycle but no deadlock:

resource allocation graph with a cycle but no deadlock:



Handling Deadlocks

- There are strategies to deal with deadlocks:
  - Ignore deadlocks: Ostrich Approach
  - Ensure deadlock never occurs using either:
    - Prevention: Prevent any one of the 4 conditions from happening.
    - Avoidance: calculate cycles about to happen and stop dangerous operations.
  - Allow deadlock to happen. This requires using both:
    - Detection: Know a deadlock has occurred.
    - Recovery: Regain the resources.

"Prevent" means to keep something from happening.
"Avoid" means to keep away from something.

Ignore deadlocks

- Sticks your head in the sand and ignores the problem. Good solution for rare and not frequent deadlock cases. We can ignore the problem altogether and pretend that deadlocks never occur in the system. If the likelihood of a deadlock is small and the cost of avoiding a deadlock is high it might be better to ignore the problem. This approach is used by many operating systems including UNIX and Windows, and the Java VM.

Deadlock Prevention

- Deadlock prevention algorithms ensure that **at least one** of the necessary conditions for deadlock cannot occur and hence that deadlocks cannot hold:
  - Prevent mutual exclusion
  - Prevent hold and wait
  - Prevent No preemption
  - Prevent Circular wait

Prevent mutual exclusion

(if I have it, you can't have it)

- Deadlocks due to mutual exclusion occur when processes are granted exclusive access to resources. To avoid this, we can make all resources sharable. Sharable recourses do not require mutual exclusion and do not create deadlocks. However, not all resources can be shared.

Prevent hold and wait

(once I get it, I can hold it even if I want something else)

- Processes are allowed to hold at least one resource while waiting for other resources that are being held by other processes. To avoid this, we need to ensure that whenever a process requests a resource, it doesn't hold any other resources.

- To do this, there are several protocols:
  - Request all resources (at once) at beginning of process execution.
  - Request all resources (at once) at any point in the program
  - To get a new resource, the process has to release all its current resources, and then tries to acquire new one plus old ones all at once.
  - A particular resource can only be requested when no other process is holding it.

Prevent No preemption

(I only give it up when I'm done with it)

- Resources are released voluntarily. To avoid this and allow preemption (take resources away):
  1. Allow preemption: if a needed resource is held by another process, which is also waiting on some resource, **steal it**. **Otherwise wait**. It means, if a process requests some resources, we first check whether they are available. If they are, we allocate them. If they are not available, we check whether they are allocated to some other process that is waiting for additional resources. If so, we preempt the desired resources from the waiting process and allocate them to the requesting process.

Prevent No preemption cont…

2. If a process is holding some resources and requests another resource that cannot be immediately allocated to it, then it must release all its resources that are currently allocated to it (it means all resources currently being held are preempted). The process will be restarted only when it can regain its old resources, as well as the new ones that is requesting.

Prevent Circular wait

each process waiting for a resource held by the next process of the chain

- To avoid this, there are two protocols:
  1. The resources are numbered in the same order that they are normally needed. A process can initially request any number of instances of a resource Ri. After that, the process can request instances of resource type Rj if and only if F(Rj) > F(Ri). It is easy to see that a cycle is no longer possible. For example if a process holds resources #34 and #54, it can request only resources #55 and higher. This is the most common approach used in operating systems.

     Example: Order: disk drive, printer, CDROM

     Process A requests disk drive, then printer

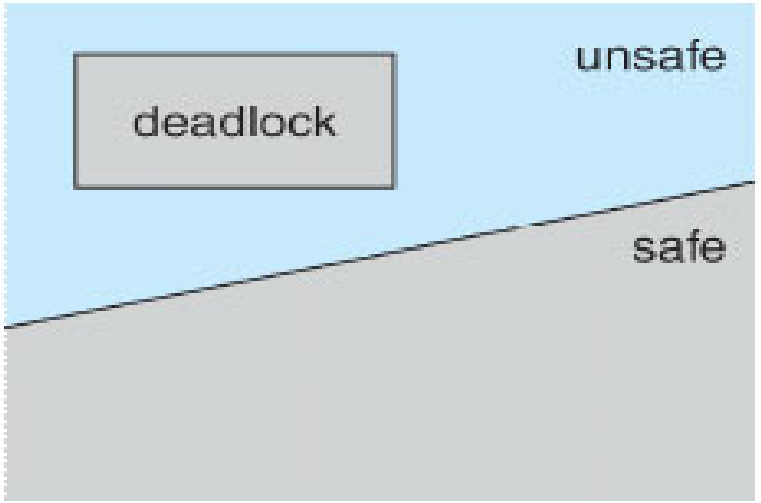
     Process B requests disk drive, then printer. Process B does not request printer, then disk drive, which could lead to deadlock

Prevent Circular wait cont…

2. Alternatively, whenever a process requests an instance of resource type Rj, it has released any resources Ri such that F(Ri) >= F(Rj).

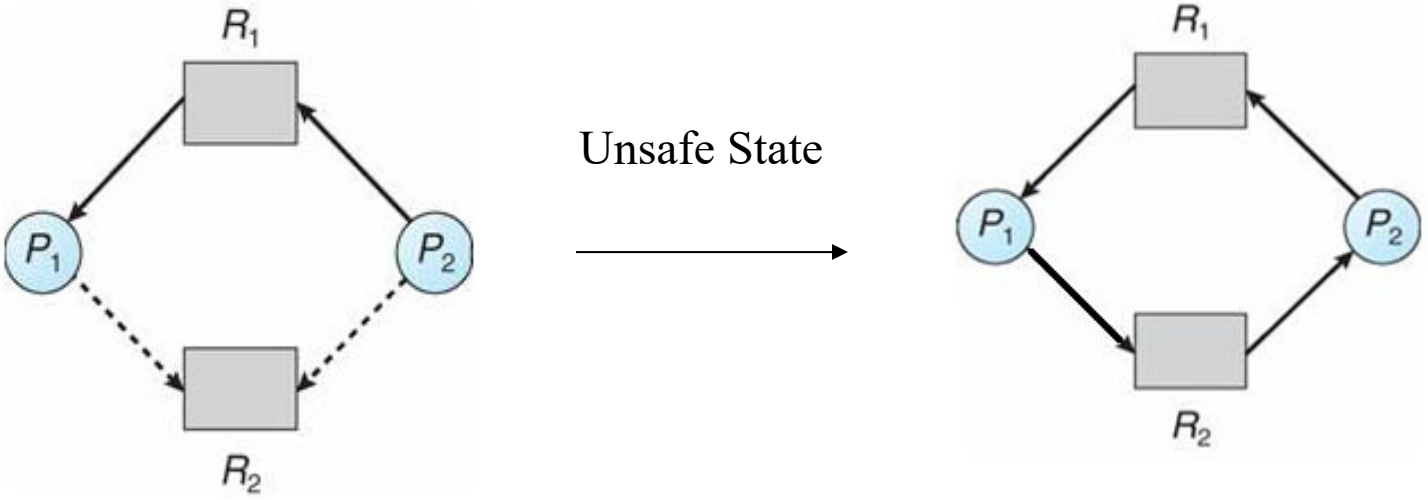
Deadlock Avoidance; Single/Multiple Instance Resources

- Avoidance algorithms ensure that the system will always remain in a safe state (a state that could not result in deadlock). Initially, the system is in a safe state. Whenever a process requests a resource that is available, the system must decide whether the resource can be allocated immediately **or** the process must wait. The request is granted only if the allocation keeps the system in safe state. If converting a Request edge to a Assignment edge creates a deadlock, then don't grant request.



Example:

- Granting P2's request for R2 creates a deadlock, so don't grant it.



Deadlock Avoidance cont…

- Banker's Algorithm is a deadlock avoidance algorithm when there is multiple instance of a resource type.

- To implement Banker's algorithm, we need the following data structures:
  - Available: number of available resources of each type
  - Maximum/Claim: maximum demand of resources by each process
  - Allocation/Holding/initial claim: number of resources of each type currently allocated to each process
  - Need: remaining resource need of each process; Need=Max-Allocation.

Example1; Banker's algorithm for single instance resources

- Consider a system with 12 hard disks and 3 processes (P1, P2, P3) each requests its maximum demand. The snapshot at time T_{now} (initial state) is as follow. Is the system in a safe state? If so, which sequence satisfies the safety criteria?

Process	Maximum	Holding	Needs
P_1	10	5	5
P_2	4	2	2
P_3	9	2	7

Example1 cont…

- At time T_{now}, we have 3 hard disks left free (12-5-2-2):
  - Available =12-5-2-2=3 ➔
  - 2 for P2 → remain=1, make free=4 → total free=5
  - 5 for P1→ remain=0, make free=10→total free=10
  - 7 for P3→ remain=3, make free=9 → total free=12
  - All processes can terminate, thus < P2, P1, P3> was safe.

Process	Maximum	Holding	Needs
P_1	10	5	5
P_2	4	2	2
P_3	9	2	7

Example2; Banker's algorithm for single instance resources

Process	Maximum	Holding	Needs
P1	10	5	5
P2	4	2	2
P3	9	3	6

- Available=12-5-2-3=2 ➔
- 2 for P2 → remain=0, make free=4 → total free=4
- P1 allocated 5, may request 5 more → has to wait
- P3 allocated 3, may request 6 more → has to wait

Example3; Banker's algorithm for single instance resources

- Consider a system with 10 Blue-ray drives and 3 processes (A, B, and C):

Process	Maximum	Holding	Needs
A	9	3	6
B	4	2	2
C	7	2	5

- Available =10-3-2-2=3 ➔
- 2 for B → remain=1, make free by B=4 → total free=5
- 5 for C→ remain=0, make free by C=7→total free=7
- 6 for A→ remain=1, make free by A=9 → total free=10

Example4; Banker's algorithm for multiple instance resources

- Consider five processes (p0, p1, p2, p3, p4) with three resources (A is scanner, B is printer, C is DVD). The available resources at the moment are: A=3, B=3, C=2.

Process	Max			Allocation			Need		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

Example4 cont…

Process	Max			Allocation			Need		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

Available: A=3, B=3, C=2

- 1A for P1→ remain A=2, make free A=3→total free A=5
- 2B for P1→ remain B=1, make free B=2→total free B=3
- 2C for P1→ remain C=0, make free C=2→total free C=2
- 0A for P3→ remain A=5, make free A=2→total free A=7
- 1B for P3→ remain B=2, make free B=2→total free B=4
- 1C for P3→ remain C=1, make free C=2→total free C=3
- 4A for P4→ remain A=3, make free A=4→total free A=7
- 3B for P4→ remain B=1, make free B=3→total free B=4
- 1C for P4→ remain C=2, make free C=3→total free C=5
- 6A for P2→ remain A=1, make free A=9→total free A=10
- 0B for P2→ remain B=4, make free B=0→total free B=4
- 0C for P2→ remain C=5, make free C=2→total free C=7
- 7A for P0→ remain A=3, make free A=7→total free A=10
- 4B for P0→ remain B=0, make free B=5→total free B=5
- 3C for P0→ remain C=4, make free C=3→total free C=7
- The system is in a safe state since the sequence < P1, P3, P4, P2, P0> satisfies safety criteria.

Deadlock Detection; Single Instance Resources

- If all resources have only a single instance, then we can define a deadlock detection algorithm that uses a variant of the resource-allocation graph, called wait-for graph (WFG). We obtain WFG graph from the resource-allocation graph by removing the resources.

- An algorithm is periodically invoked that searches for a cycle in the graph. If there exists a cycle in wait-for graph, there is a deadlock in the system, and the processes forming the part of cycle are blocked in the deadlock.

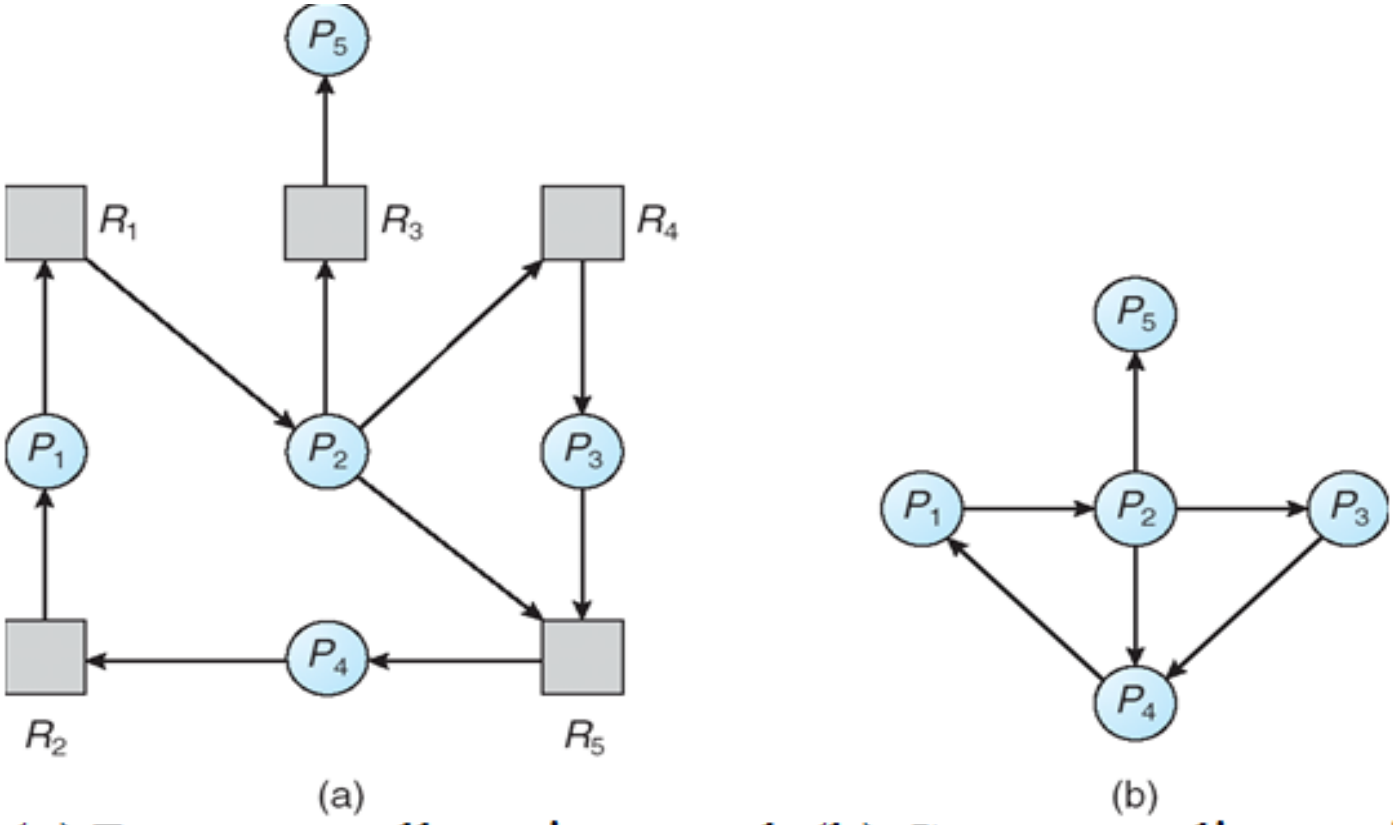
Wait-for graph

- An edge from P_i to P_j in a wait-for graph implies that process P_i is waiting for process P_j to release a resource that P_i needs. Thus, an edge $P_i \rightarrow P_j$ contains two edges $P_i \rightarrow R_q$ and $R_q \rightarrow P_j$ for some resource R_q.

- Pros of WFG:
  - With about half the number of nodes and edges, the WFG requires only about half the storage of a RAG.

- Cons of WFG:
  - The WFGs can only be used for RAGs with single-instance resources. In contrast, resources for RAGs can have any number of instances.

Example:



(a) Resource-allocation graph (b) Corresponding wait-for graph

Deadlock Detection; Multiple Instance Resources

- For deadlock detection Modification of Banker's algorithm is used:
  - We need a request matrix instead of the claim matrix
  - Disregard processes without any allocation (not holding resources)

Example1:

- There are five processes each demands resources as shown in the Request matrix with the allocated resources as shown in Allocate matrix and the available resources shown in the Avail matrix. The snapshot at time T_o is as follow. Is the system deadlocked? If not which sequence will results in Finish=true?

Allocate[i,j]

	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2

Request[i,j]

	A	B	C
P0	0	0	0
P1	2	0	2
P2	0	0	0
P3	1	0	0
P4	0	0	2

Avail[j]

A	B	C
0	0	0

Process	Request			Allocation		
	A	B	C	A	B	C
P0	0	0	0	0	1	0
P1	2	0	2	2	0	0
P2	0	0	0	3	0	3
P3	1	0	0	2	1	1
P4	0	0	2	0	0	2

Example1 cont…

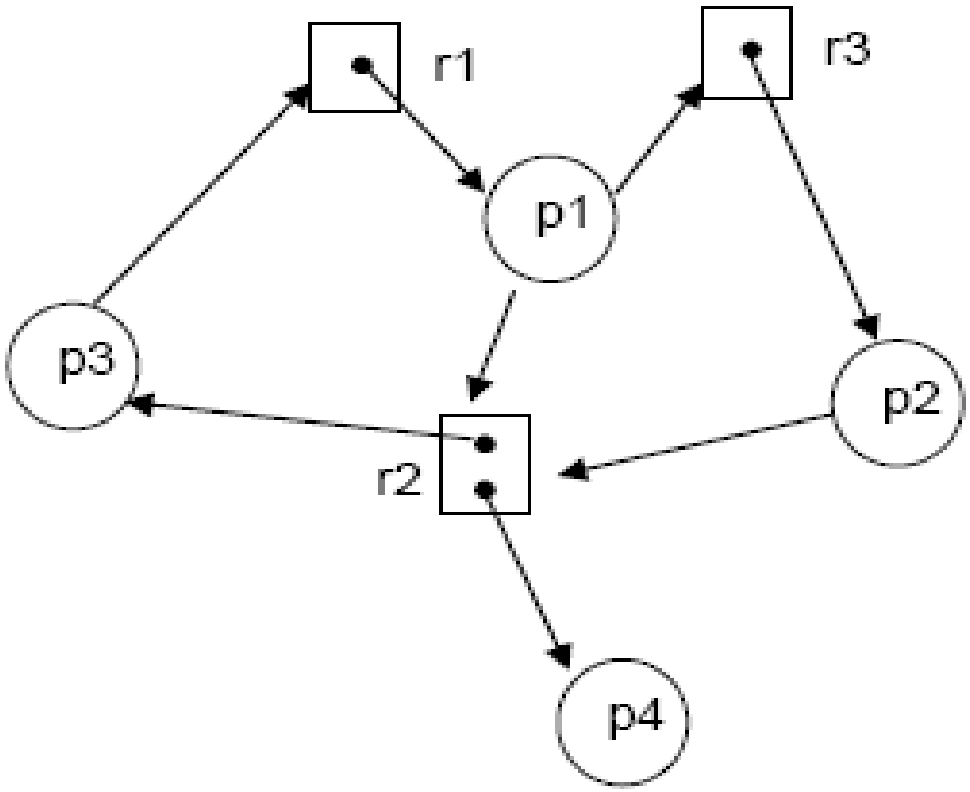
Process	Request			Allocation		
	A	B	C	A	B	C
P0	0	0	0	0	1	0
P1	2	0	2	2	0	0
P2	0	0	0	3	0	3
P3	1	0	0	2	1	1
P4	0	0	2	0	0	2

- 0A for P2→ remain A=0, make free A=3→total free A=3
- 0B for P2→ remain B=0, make free B=0→total free B=0
- 0C for P2→ remain C=0, make free C=3→total free C=3
- 0A for P0→ remain A=3, make free A=0→total free A=3
- 0B for P0→ remain B=0, make free B=1→total free B=1
- 0C for P0→ remain C=3, make free C=0→total free C=3
- 1A for P3→ remain A=2, make free A=3→total free A=5
- 0B for P3→ remain B=1, make free B=1→total free B=2
- 0C for P3→ remain C=3, make free C=1→total free C=4
- 2A for P1→ remain A=3, make free A=4→total free A=7
- 0B for P1→ remain B=2, make free B=0→total free B=2
- 2C for P1→ remain C=2, make free C=2→total free C=4
- 0A for P4→ remain A=7, make free A=0→total free A=7
- 0B for P4→ remain B=2, make free B=0→total free B=2
- 2C for P4→ remain C=2, make free C=4→total free C=6
- System is not deadlocked, sequence <P2, P0, P3, P1, P4> will result in Finish[i] = true for all i processes.

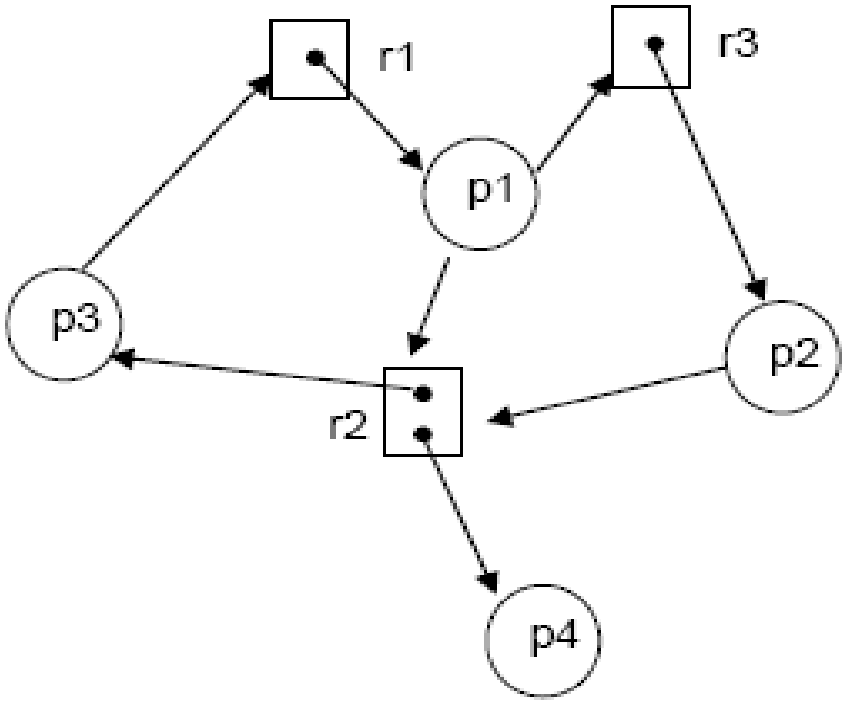
Note that both P0 and P2 have requests that can be satisfied by what is available. Select whichever you like, so we select P2.

Example2:

- Examine whether the system whose resource allocation graph is given below is deadlocked or not.



Example2 cont…



$$\text{Allocation} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{Request} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{Available} = [0 \;\; 0 \;\; 0]$$

Deadlock Recovery

- When a detection algorithm determines that a deadlock exists, there are two options for breaking the deadlock get the resources back:
  - Process Termination: to abort one or more processes to break the circular wait.
  - Resource Preemption: to preempt some resources from one or more of the deadlocked processes.

Process Termination

- There are two methods that can be used for terminating the processes to recover from the deadlock:
  - Terminating one process at a time until the circular-wait condition is eliminated. It involves an overhead of invoking a deadlock detection algorithm after terminating each process to detect whether circular-wait condition is eliminated or not which is time consuming.
  - Terminating all processes involved in the deadlock. This method will definitely ensure the recovery of a system from the deadlock. The disadvantage of this method is that it is expensive because many processes may have executed for a long time; close to their completion. As a result, all the computations performed till the time of termination are discarded.

Resource Preemption

- In general, it is easier to preempt the resource, than to terminate the process. We can preempt the resources from the processes one by one and allocate them to other processes until the circular-wait condition is eliminated.