University of Lucknow | Centenary Year
लखनऊ विश्वविद्यालय | शताब्दी वर्ष

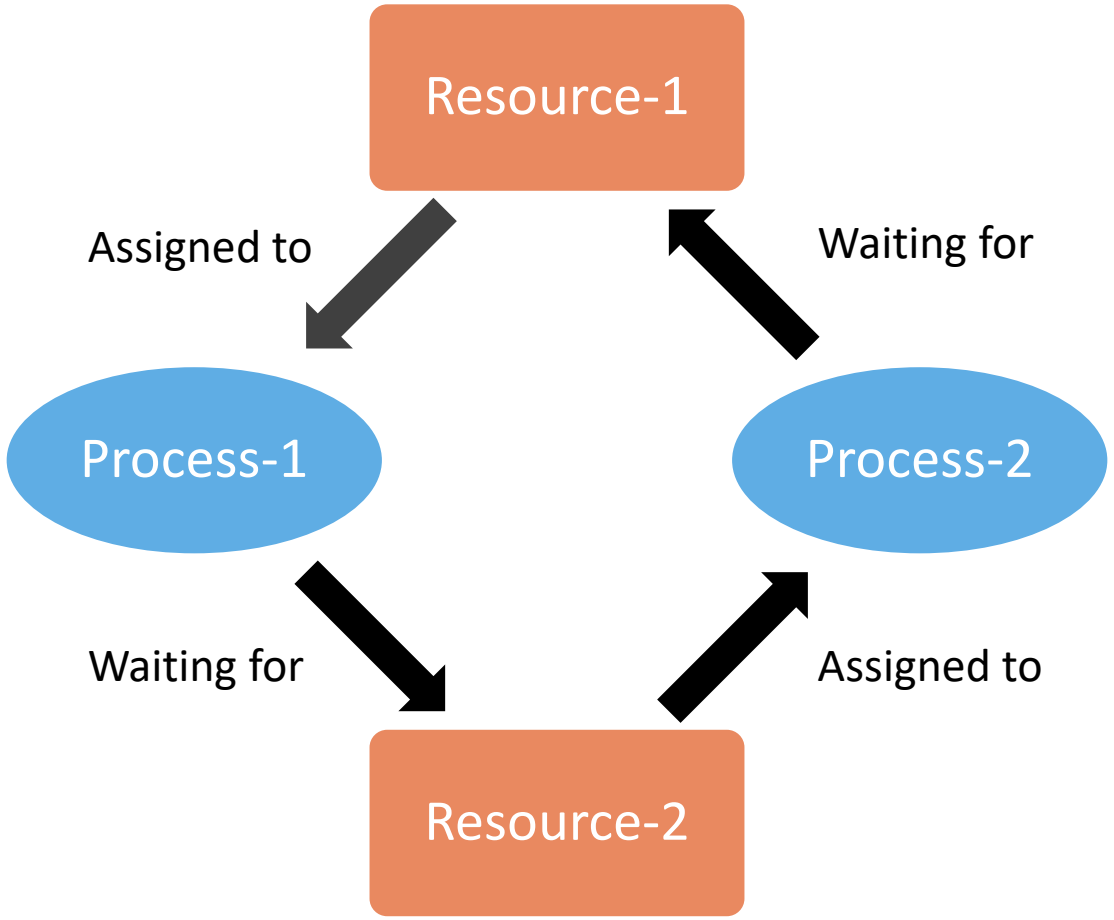
Operating System: Deadlock Handling

Dr. Puneet Misra
Department of Computer Science,
University of Lucknow,
Lucknow PIN-226007
<puneetmisra@gmail.com>

Agenda

- Deadlock
- Deadlock Problem & Example- Bridge Crossing Problem
- The cause, System Model & Normal Resource Allocation Operation
- Necessary conditions for deadlock
- Methods for handling Deadlock
- Resource Allocation Graph (RAG)
- Deadlock Prevention
- Deadlock Avoidance
- Recovery from Deadlock

Deadlock:



- A deadlock is a situation wherein two or more competing actions are each waiting for the other to finish, and thus neither ever does. It is often seen in a paradox like the "chicken or the egg".
- It is a condition in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function.

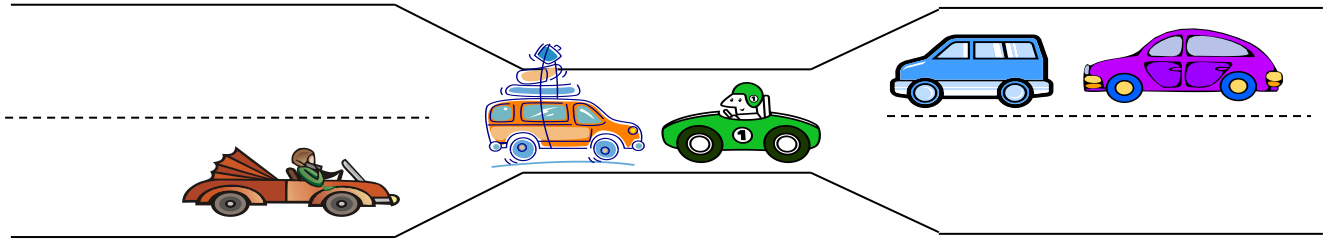
Deadlock Problem:

Deadlock refers to a specific condition when two or more processes are each waiting for each other to release a resource, or more than two processes are waiting for resources in a circular chain.

Example

- Client applications using the database may require exclusive access to a table.
- Two client applications C1 and C2.
- If C1 holds lock on table, attempts to obtain on the other i.e. already held by C2.
- This may lead to deadlock if the C2 application then attempts to obtain the lock that is held by the C1 application.

Example- Bridge Crossing



- Each segment of road can be viewed as a resource
  - Car must own the segment under them
  - Must acquire segment that they are moving into
- For bridge: must acquire both halves
  - Traffic only in one direction at a time
  - Problem occurs when two cars in opposite directions on bridge: each acquires one segment and needs next
- 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

More examples:

- “It takes money to make money.”
- “You can't get a job without experience; you can't get experience without a job.”

The cause of deadlocks:

Each process needing what another process has. This results is blocking from sharing resources such as memory, devices, links.

System Model:

A system consists of a finite no. of resources for no. of competing processes:

✓Resource types R1, R2, . . ., Rm

✓CPU cycles, memory space, files and I/O devices

Normal Resource Allocation Operation

Each resource type Ri has Wi instances. Each process utilizes a resource as follows:

1. Request a resource: if request not granted, then process should wait to acquire.
2. Use the resource: process can operate on the resources.
3. Release the resource: the process releases the resources.

Deadlock Characterization:

Necessary Conditions for Deadlock to occur: Deadlock can arise if following four conditions hold simultaneously:

- ✓**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 {P0, P1, …, P0} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, …, Pn–1 is waiting for a resource that is held by Pn, and P0 is waiting for a resource that is held by P0.

Resource Allocation Graph (RAG)

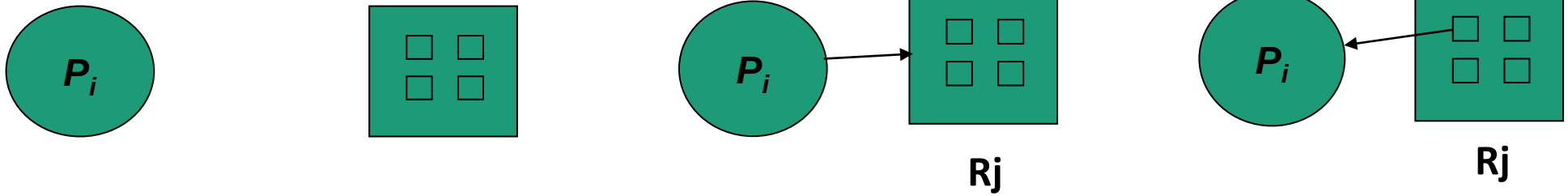
A visual (mathematical) way to determine if a deadlock has, or may occur.

- G = (V, E) The graph contains nodes and edges.

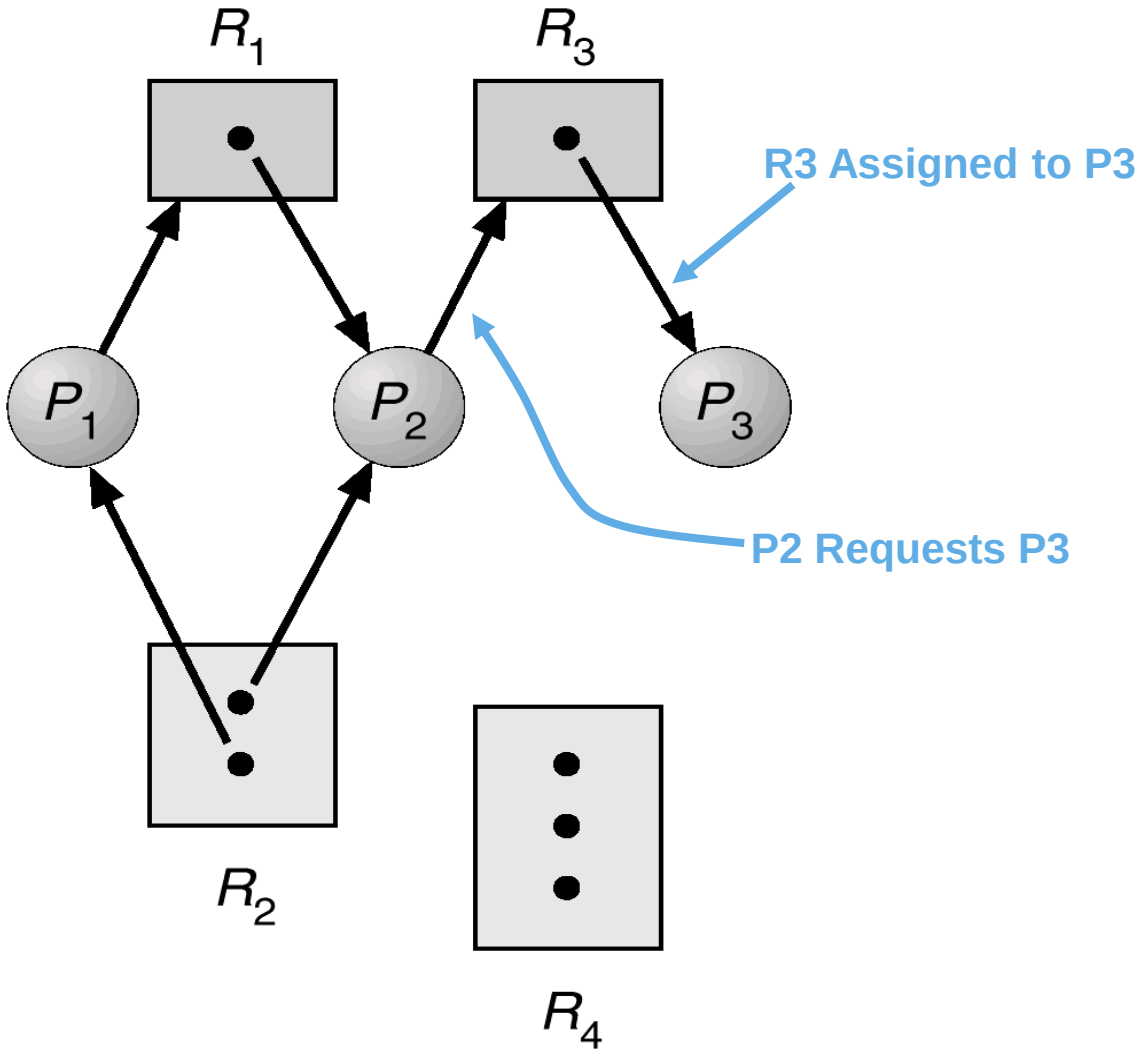
✓V Nodes consist of processes = { P1, P2, P3, ...} and resource types { R1, R2, ...}

✓E Edges are (Pi, Rj) or (Ri, Pj)

- An arrow from the process to resource indicates the process is requesting the resource. An arrow from resource to process shows an instance of the resource has been allocated to the process.
- Process is a circle, resource type is square; dots represent number of instances of resource in type. Request points to square, assignment comes from dot.

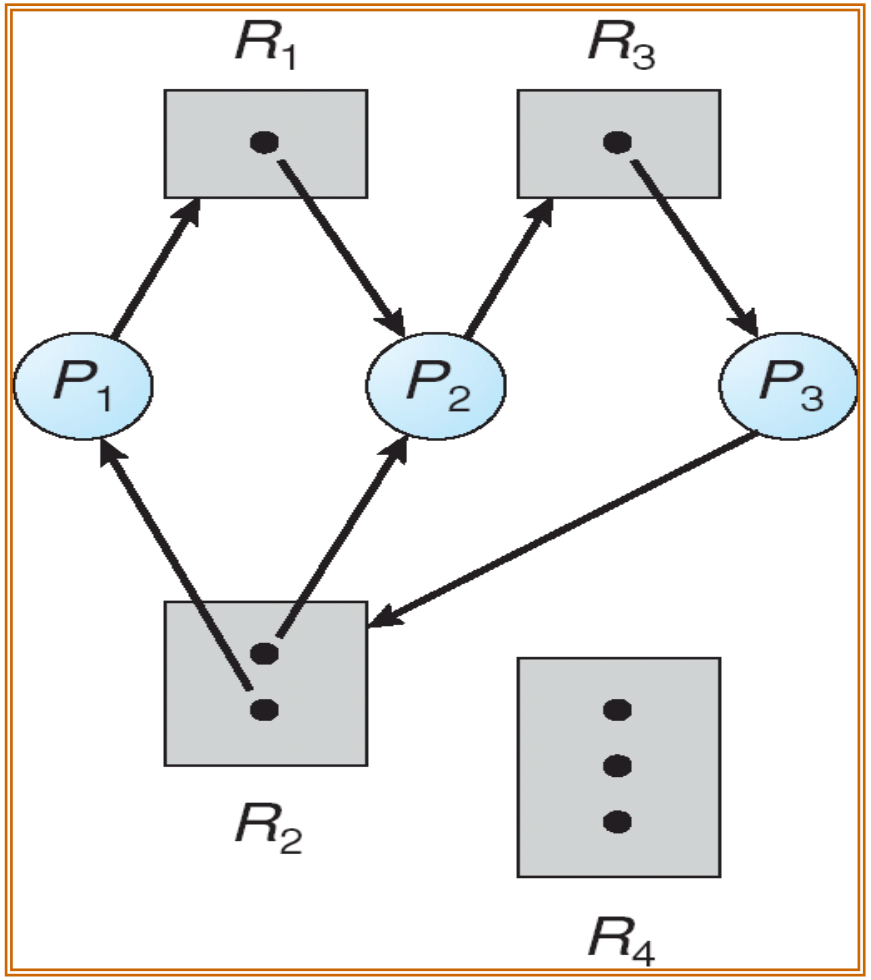


- If the graph contains no cycles, then no process is deadlocked.
- If there is a cycle, then:
  - If resource types have multiple instances per resource type, then possibility of deadlock MAY exist.
  - If each resource type has one instance per resource type, then deadlock has occurred.

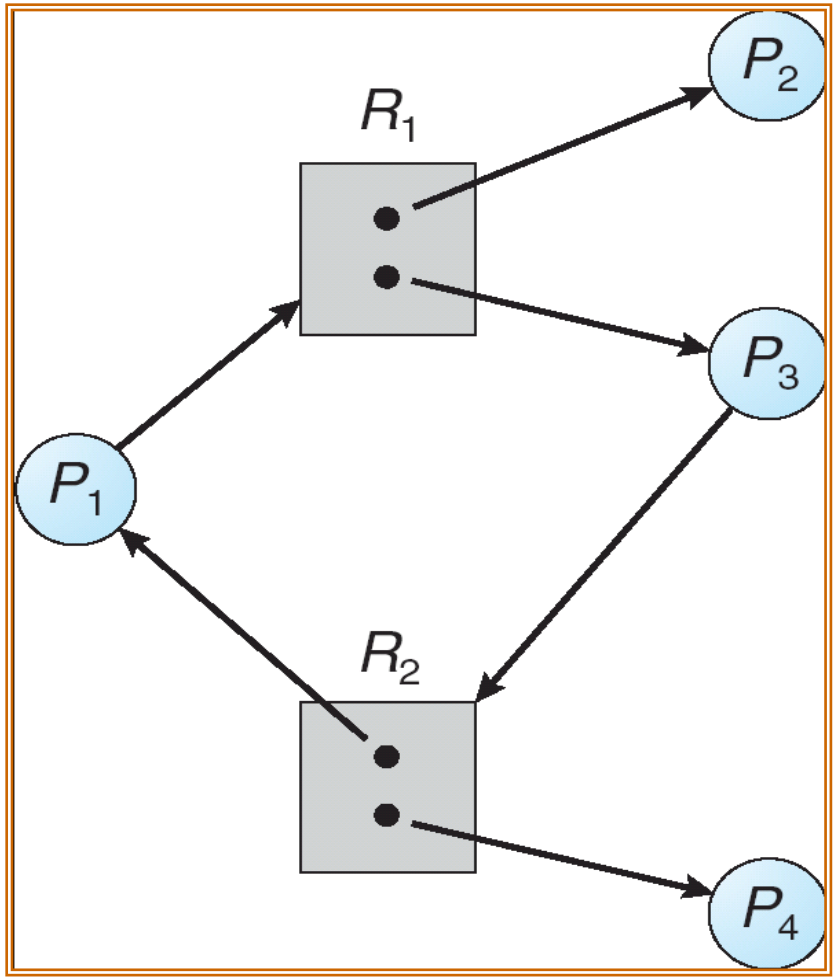


Resource allocation graph

Resource Allocation Graph with a Deadlock



Resource Allocation Graph with a Cycle But No Deadlock



Methods for handling deadlock

- **Prevention/Avoidance**
  - ✓ **Prevention** **Prevent any one of the 4 conditions from happening.**
  - ✓ **Avoidance** **Allow all deadlock conditions but calculate cycles about to happen and stop dangerous operations.**
    - ❑ Ensure that system will never enter a deadlock
    - ❑ Need to monitor all lock acquisitions
    - ❑ Selectively deny those that might lead to deadlock
- **Detection/Correction**
  - ✓ **Detection** **Know a deadlock has occurred.**
  - ✓ **Recovery** **Regain the resources.**
    - ❑ Allow system to enter deadlock and then recover
    - ❑ Requires deadlock detection algorithm
    - ❑ Some technique for selectively pre-empting resources and/or terminating tasks
- **Ignore** ← **Most Operating systems do this!!**
  - Ignore the problem and pretend that deadlocks never occur in the system
  - used by most operating systems, including UNIX

Deadlock Prevention

Do not allow one of the four conditions to occur.

1. Mutual exclusion:
   - Automatically holds for printers and other non-sharables.
   - Shared entities (read only files) don't need mutual exclusion (and aren’t susceptible to deadlock.)
   - Prevention not possible, since some devices are intrinsically non-sharable
2. Hold and wait:
   - Collect all resources before execution.
   - A particular resource can only be requested when no others are being held. A sequence of resources is always collected beginning with the same one.
   - Utilization is low, starvation possible.

Deadlock Prevention (contd..)

Do not allow one of the four conditions to occur.

3. No pre-emption:
   - Release any resource already being held if the process can't get an additional resource.
   - Allow pre-emption - if a needed resource is held by another process, which is also waiting on some resource, steal it. Otherwise wait.
4. Circular wait:
   - Number resources and only request in ascending order.

EACH of these prevention techniques may cause a decrease in utilization and/or resources. For this reason, prevention isn't necessarily the best technique.

Prevention is generally the easiest to implement.

Deadlock Avoidance

If there is prior knowledge of how resources will be requested, it's possible to determine if we are entering an "unsafe" state.

- Possible states are:
  - Deadlock No forward progress can be made.
  - Unsafe state A state that may allow deadlock.
- Safe state A state is safe if a sequence of processes exist such that there are enough resources for the first to finish, and as each finishes and releases its resources there are enough for the next to finish.
- The rule is simple that If a request allocation would cause an unsafe state, do not honor that request.
- NOTE: All deadlocks are unsafe, but all unsafes are NOT deadlocks.

Deadlock Avoidance (contd…)

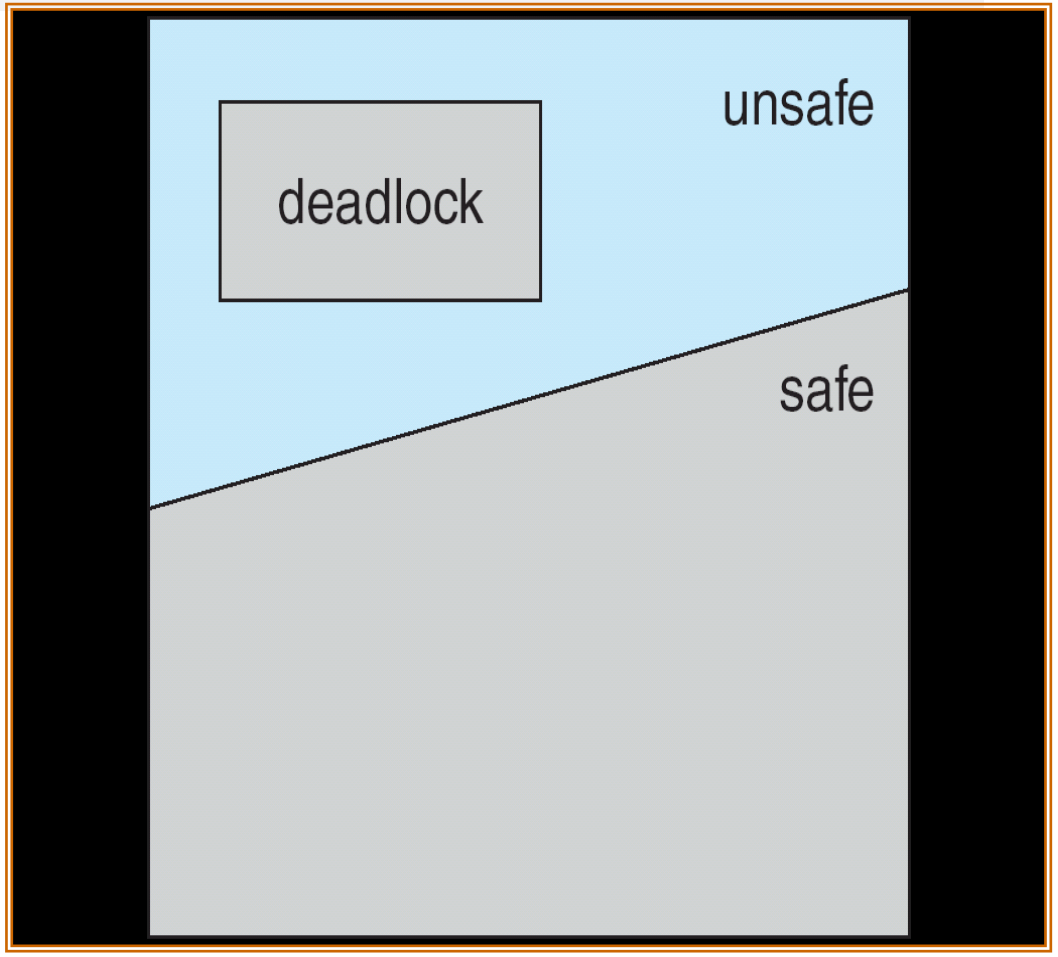
- Requires that the system has some additional a priori information available.
- Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure 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.

Safe State:

- 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 <P1, P2, …, Pn> of ALL the processes is the system such that for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj, with j < i.
- That is:
  - If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished.
  - When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate.
  - When Pi terminates, Pi +1 can obtain its needed resources, and so on.

Safe, Unsafe & Deadlock States:

- If System is in a safe state ⇒ no deadlocks
- If system is deadlocked ⇒ state is unsafe
- System is in unsafe state ⇒ possibility of deadlock.
  - OS cannot prevent processes from requesting resources in a sequence that leads to deadlock
- Deadlock Avoidance
  - Ensures system will never enter an unsafe state
  - Thereby avoids the possibility of deadlock



Deadlock Avoidance Concepts:

- Key Ideas:
  - Initially the system is in a safe state
  - Whenever a process requests an available resource, system will allocate resource immediately only if the resulting state is still safe!
  - Otherwise, requesting process must wait.
- Why does this work?
  - By induction, all reachable states are safe states
  - By definition, all safe states are not deadlocked

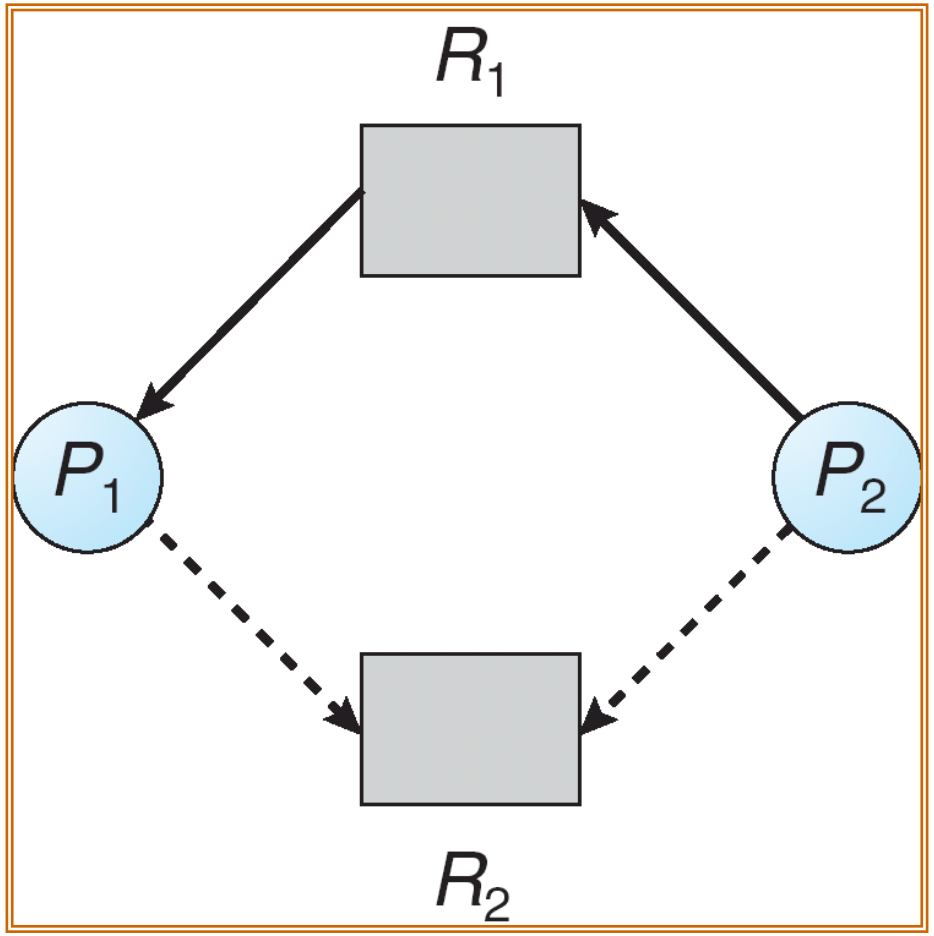
Avoidance Algorithms

- Single instance of a resource type.
  - Use a resource-allocation graph
  - Cycles are necessary and sufficient for deadlock

- Multiple instances of a resource type.
  - Use the banker's algorithm
  - Cycles are necessary, but not sufficient for deadlock

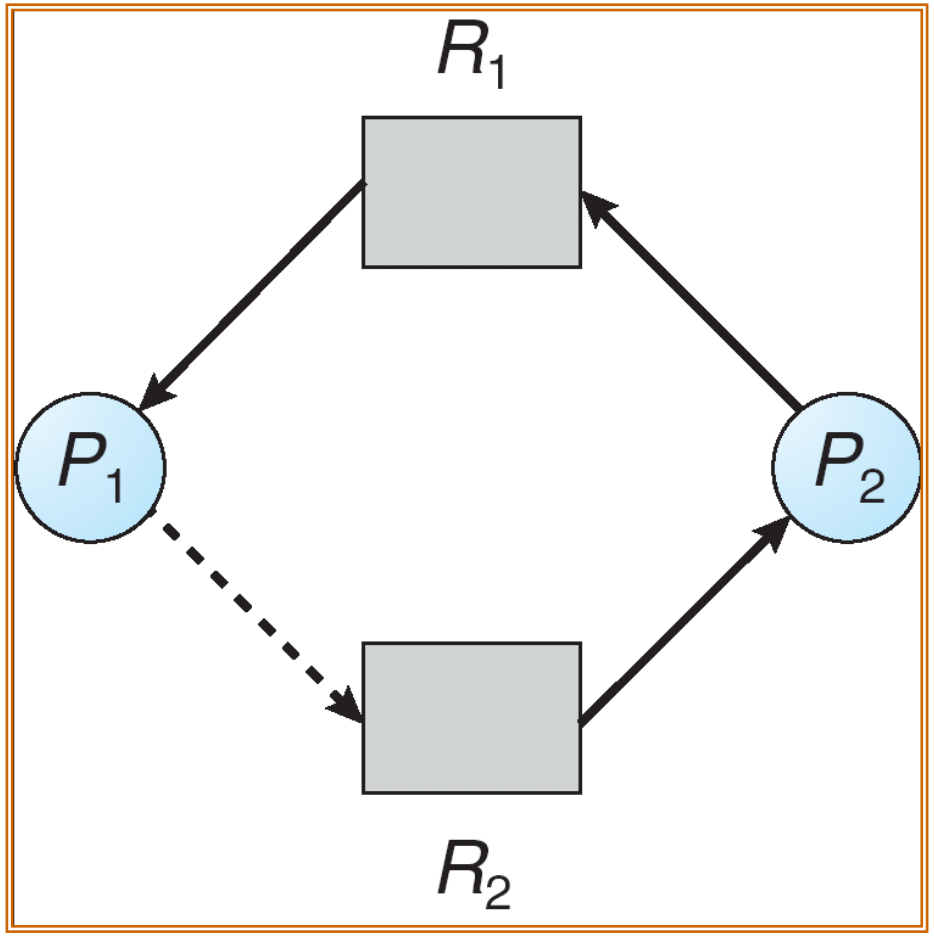
Resource Allocation Graph

- P2 requesting R1, but R1 is already allocated to P1.
- Both processes have a claim on resource R2
- What happens if P2 now requests resource R2?



Unsafe State in Resource Allocation Graph

- Consider the given RAG which cannot allocate resource R2 to process P2
- Why? Because resulting state is unsafe
  - P1 could request R2, thereby creating deadlock!



Resource Allocation Graph Algorithm

- Used only when there is a single instance of each resource type
- Suppose that process Pi requests a resource Rj
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph

Banker’s Algorithm

- Multiple instances.
- Each process claims maximum resource needs a priori.
- When a process requests a resource it may have to wait.
- When a process gets all of its resources it must return them in a finite amount of time.

Data Structures for 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 Rj available.
- Max: n x m matrix. If Max [i,j] = k, then process Pi may request at most k instances of resource type Rj.
- Allocation: n x m matrix. If Allocation[i,j] = k then Pi is currently allocated k instances of Rj.
- Need: n x m matrix. If Need[i,j] = k, then Pi may need k more instances of Rj to complete its task.
  - Need [i,j] = Max[i,j] – Allocation [i,j].

Safety Algorithm:

- Let *Work* and *Finish* be vectors of length *m* and *n*, respectively. Initialize:
  - *Work* = *Available*
  - *Finish* [*i*] = *false* for *i* = 0, 1, …, *n*- 1.
- Find and *i* such that both:
  - (a) *Finish* [*i*] = *false*
  - (b) $Need_i \leq Work$
  - If no such *i* exists, go to step 4.
- $Work = Work + Allocation_i$
  Finish[*i*] = *true*
  go to step 2.
- If *Finish* [*i*] == true for all *i*, then the system is in a safe state.

Resource Request Algorithm

Process P_i

Request = request vector for process P_i. If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j.

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

  $Available = Available - Request;$

  $Allocation_i = Allocation_i + Request_i;$

  $Need_i = Need_i - Request_i;$
- If safe $\leq$ the resources are allocated to P_i.
- If unsafe $\leq P_i$ must wait, and the old resource-allocation state is restored

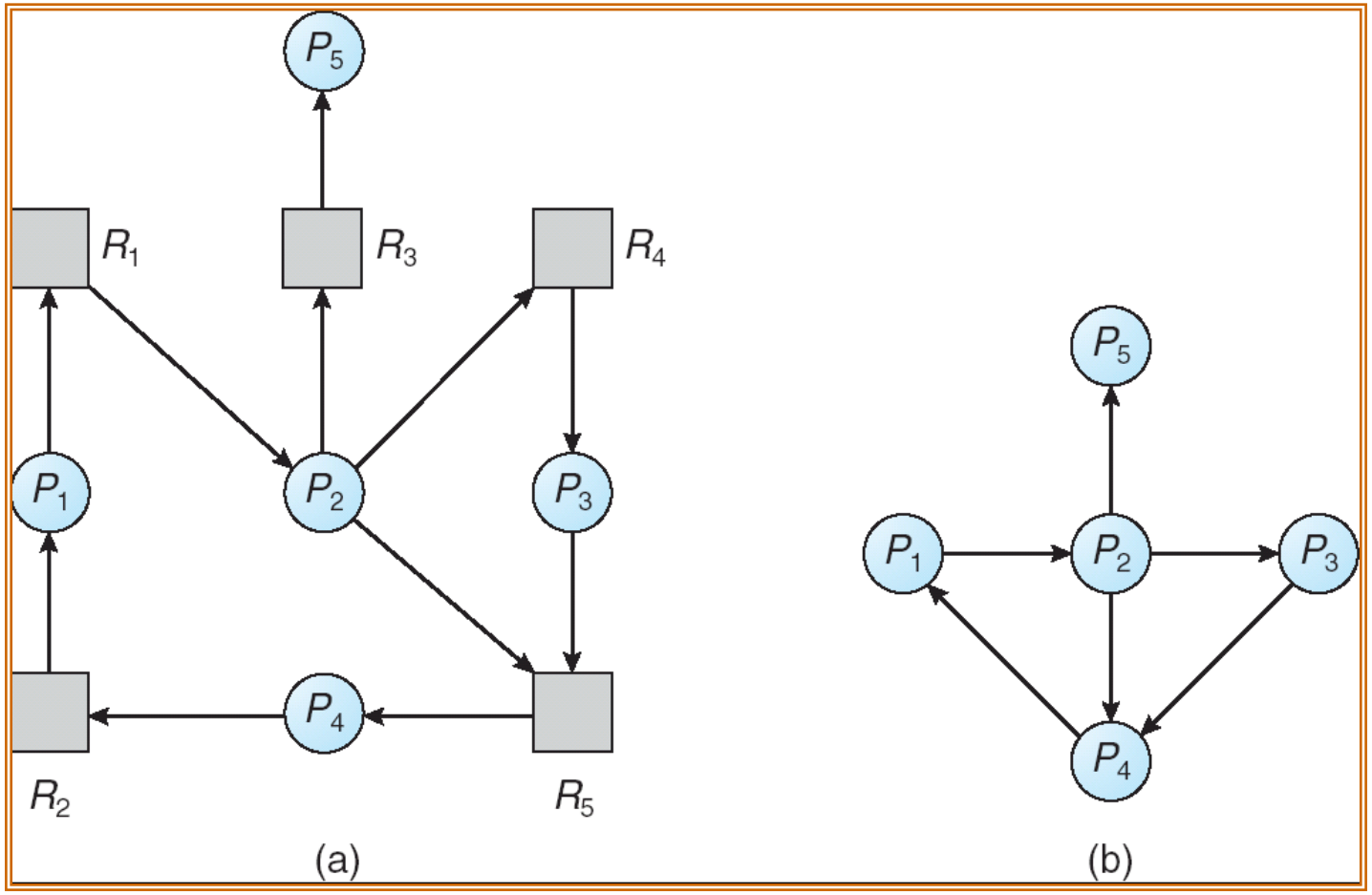
Deadlock Detection:

- 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. 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 & 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 [ij] = 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

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[1] == 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.

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

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: 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.
  - ✓Is process interactive or batch?

Recovery From Deadlock: Resource Pre-emption

- 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.