Deadlock

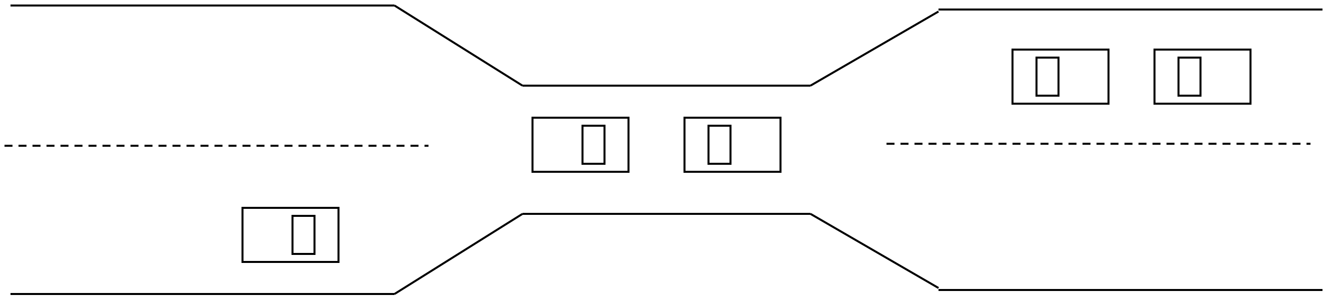
The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set

- Example
  - semaphores *A* and *B*, initialized to 1

P_0	P_1
wait (A);	*wait(B)*
wait (B);	*wait(A)*

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.

System Model

- Resource types $R_1, R_2, \ldots, R_m$

  CPU cycles, memory, I/O devices. …

- Each resource type R_i has W_i instances
- Each process utilizes a resource as follows:
  - request
  - use
  - release

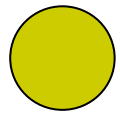
Necessary Conditions for Deadlock

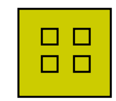
- Mutual exclusion**:** only one process at a time can use a resource
- Hold and wait**:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- No preemption**:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
- Circular wait**:** there exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0

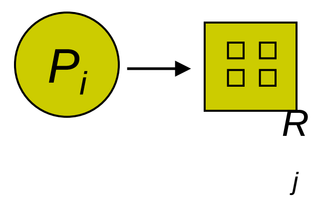
Resource-Allocation Graph

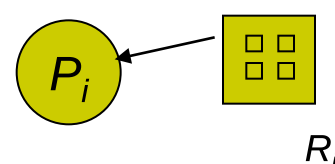
- A set of vertices V and a set of edges E
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system.
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P_1 \rightarrow R_j$
- assignment edge – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (Cont.)

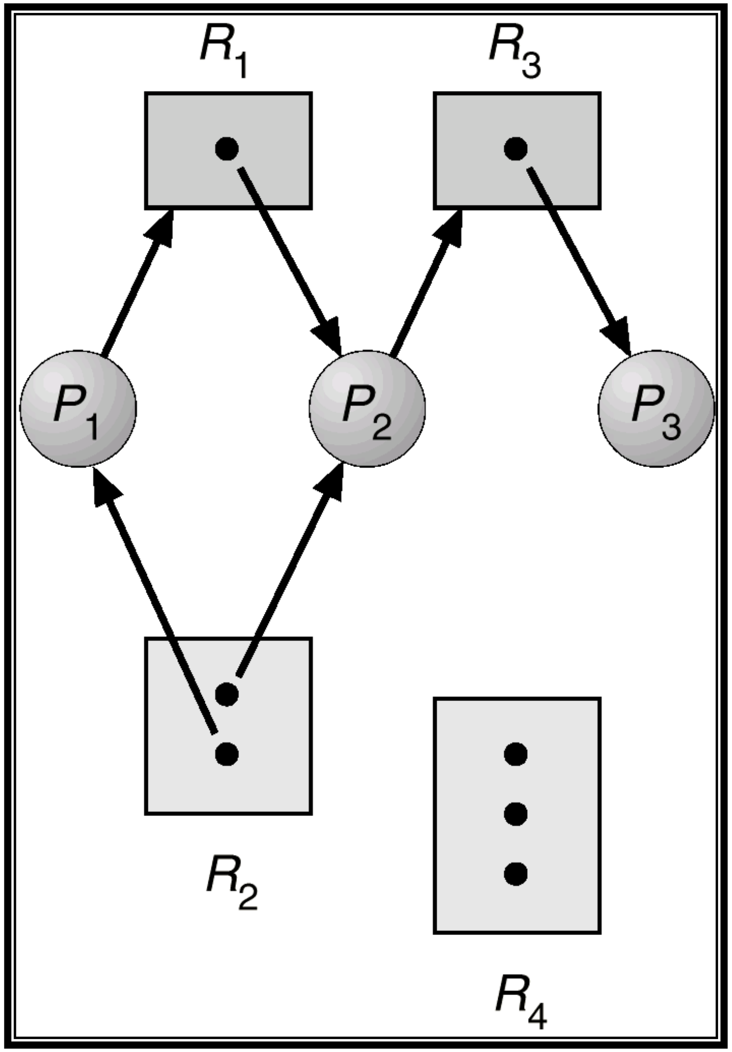
- Process 

- Resource Type with 4 instances 

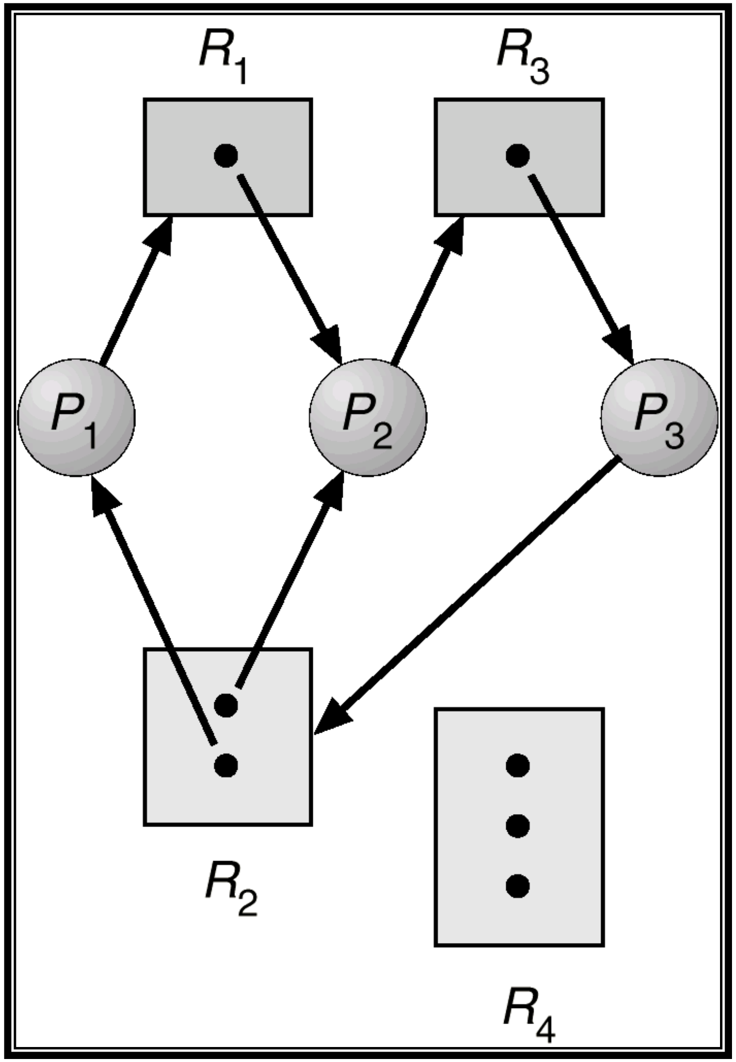
- P_i requests instance of R_j 

- P_i is holding an instance of R_j 

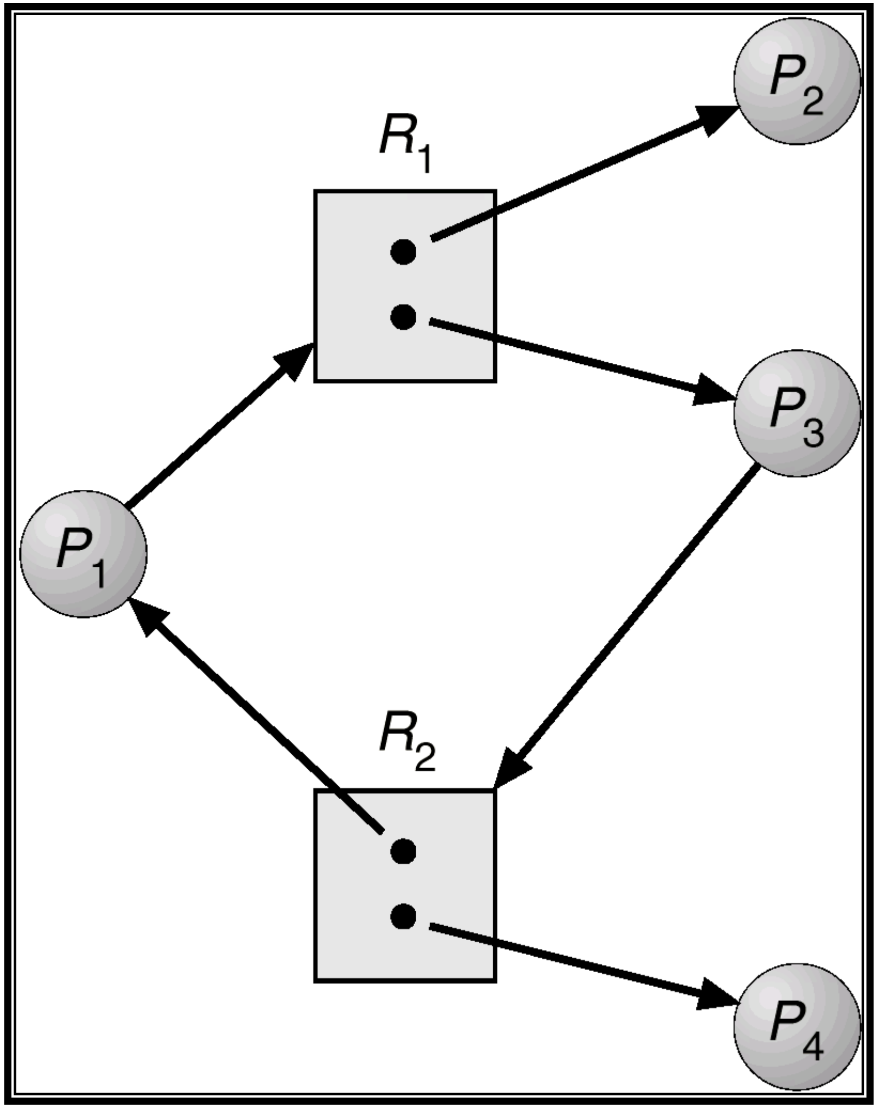
Example of a Resource Allocation Graph



Resource Allocation Graph With A Deadlock



Resource Allocation Graph With A Cycle But 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

Methods for Handling Deadlocks

- Ensure that the system will *never* enter a deadlock state
  - Avoidance, prevention
- Allow the system to enter a deadlock state and then recover
  - Detection, recovery
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

Deadlock Prevention

- Restrain the ways request can be made (Break one of the necessary conditions)
  - Mutual Exclusion – not required for sharable resources; must hold for nonsharable 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

Deadlock Prevention (Cont.)

- 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

Deadlock Avoidance

- Requires that the system has some additional *a priori* information available
  - Example: 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 safe sequence of all processes
- Sequence $<P_1, P_2, \ldots, P_n>$ is safe if for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j<i$.
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on

Avoidance Algorithm Basic

- If a system is in safe state $\Rightarrow$ no deadlocks
- If a system is in unsafe state $\Rightarrow$ possibility of deadlock
- Avoidance algorithms ensure that a system will never enter an unsafe state while allocating a resource
  - Banker's Algorithm
- Too costly for any practical operating system

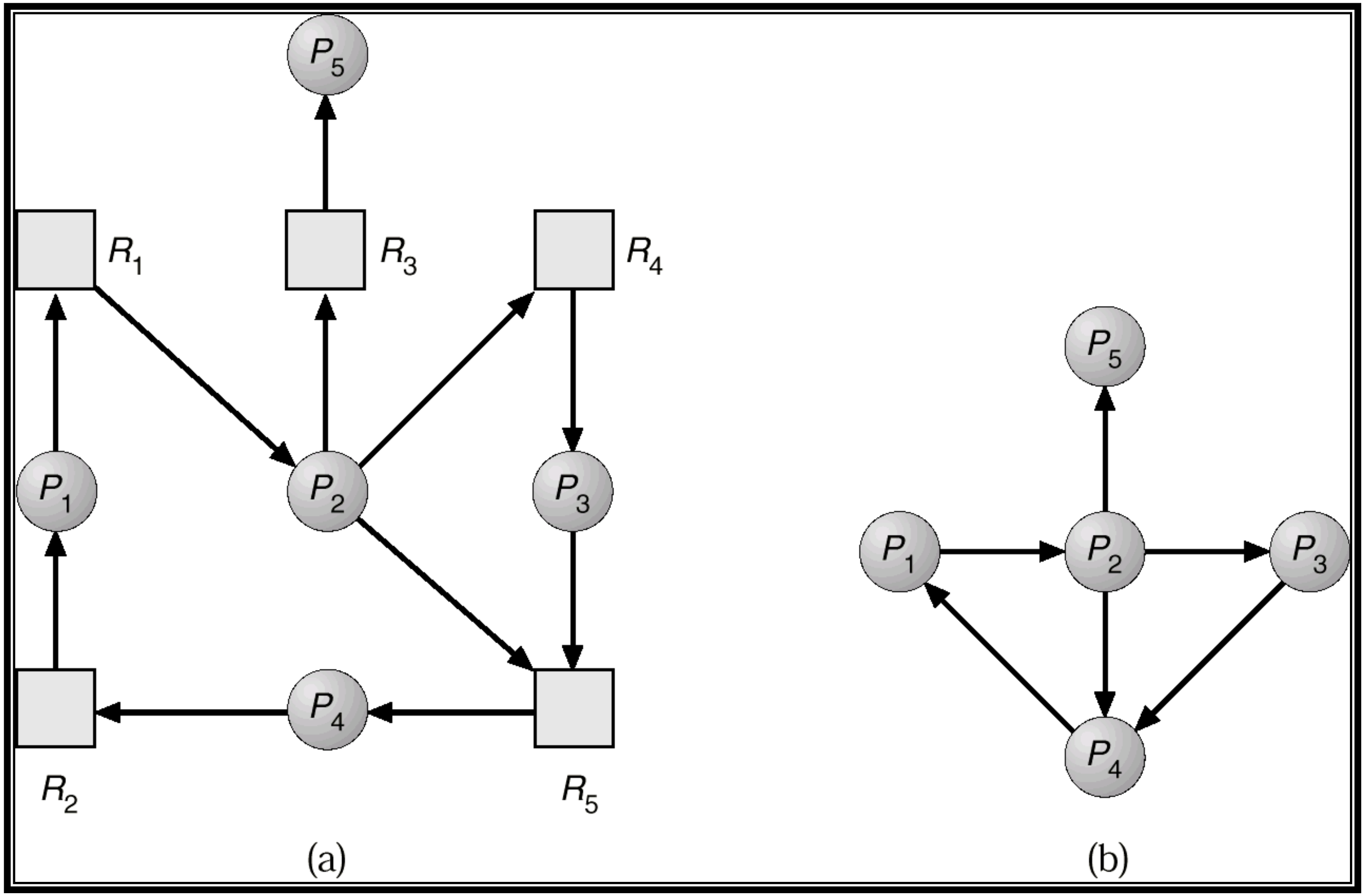
Deadlock Detection

- Allow system to enter deadlock state sometimes
- Detection algorithm to find if deadlock
  - When should it be invoked?
- Recovery scheme to break deadlock

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

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph

Corresponding wait-for graph

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?