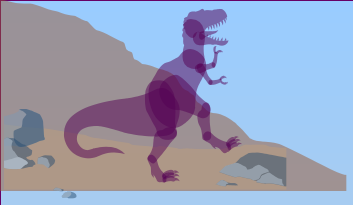
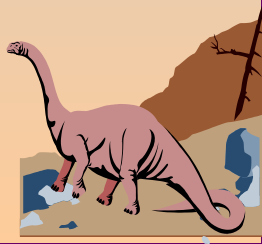


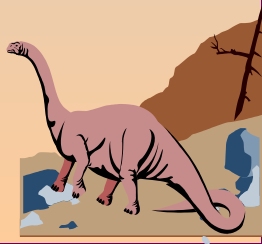
Chapter 7: Deadlocks

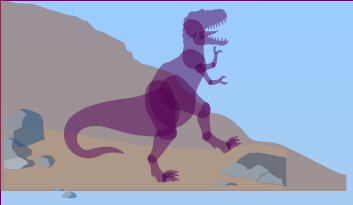
- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks



The Deadlock Problem

- A set of processes is in **deadlock** state when every process in the set is waiting for an event that can be caused only by another process in the set.
- Under normal mode of operation, a process utilize a resource in the following sequence:
  - **Request** –If the request can not granted immediately, then the process has to wait until it can acquire the resource.Also the process is added to the waiting queue of the device.
  - **Use** – The process can operate on the resource.
  - **Release** – The process releases the resource.
- Resources may be physical resources(tape drives, memory space and CPU cycles) or logical resources(semaphores, files)

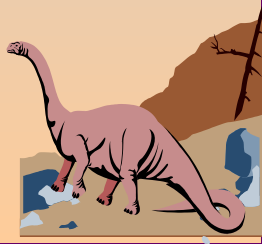


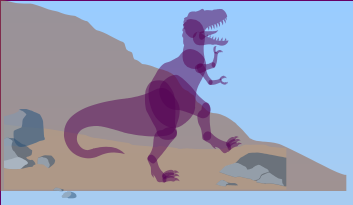


Example of deadlock

- Example- deadlock with same kind of device
  - System has 2 tape drives.
  - P_1 and P_2 each hold one tape drive and each needs another one.
- Example- deadlock with different device
  - System has one tape drive and one printer.
  - P_1 is holding printer and P_2 is holding tape drive.
  - Now if P1 requests tape drive and P2 requests printer,system is in deadlock.
- Example of logical resource (semaphore)
  - semaphores A and B, initialized to 1

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



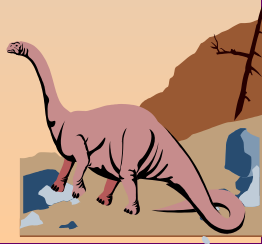


Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** Only one process at a time can use a resource, so other requesting processes must wait.
- **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, …, Pn} 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,……

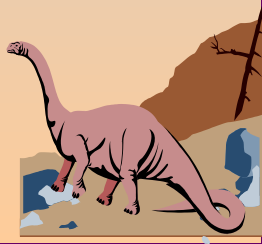
  Pn-1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0.



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 Pi → Rj
- **assignment edge** – directed edge Rj → Pi .
- When a resource is allocated to the process, the request edge is removed and assignment edge is added to the graph.

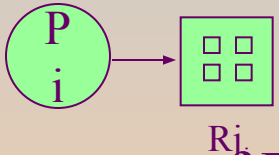


Resource-Allocation Graph (Cont.)

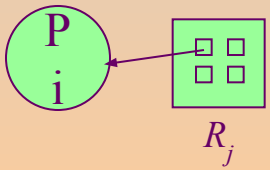
- Process

- Resource Type with 4 instances

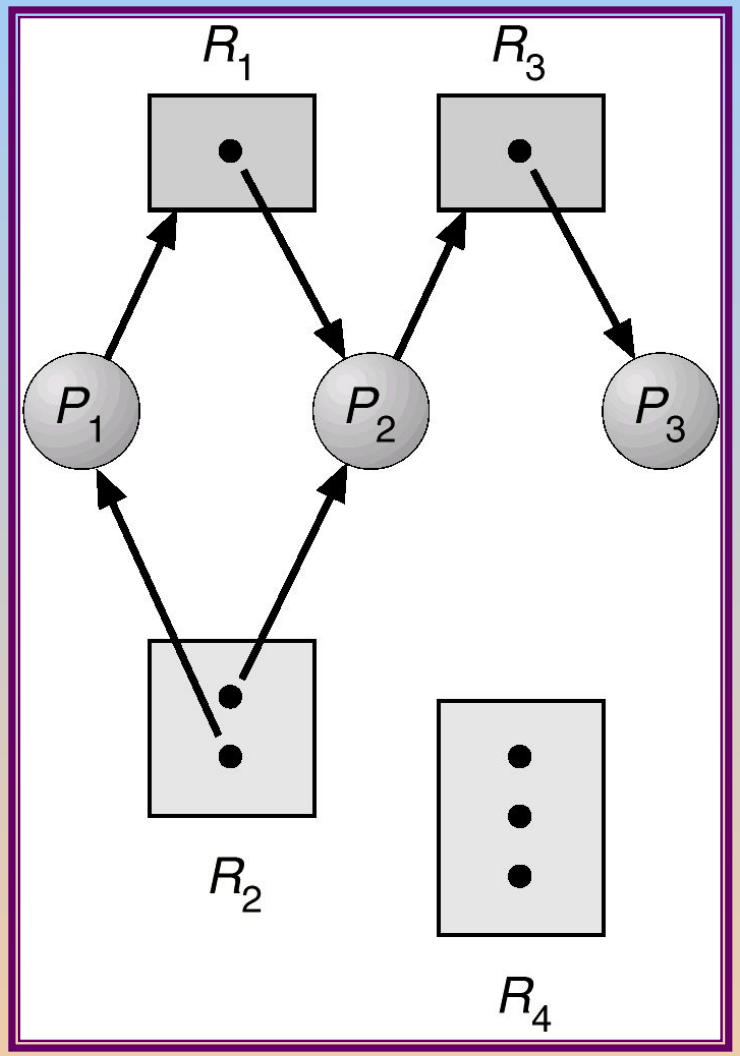
- P_i requests instance of R_j

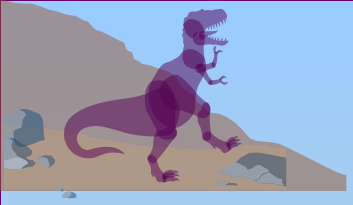


- P_i is holding an instance of R_j

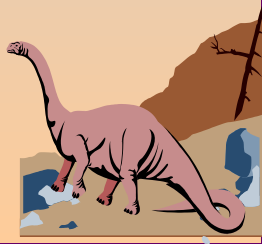


Example of a Resource Allocation Graph

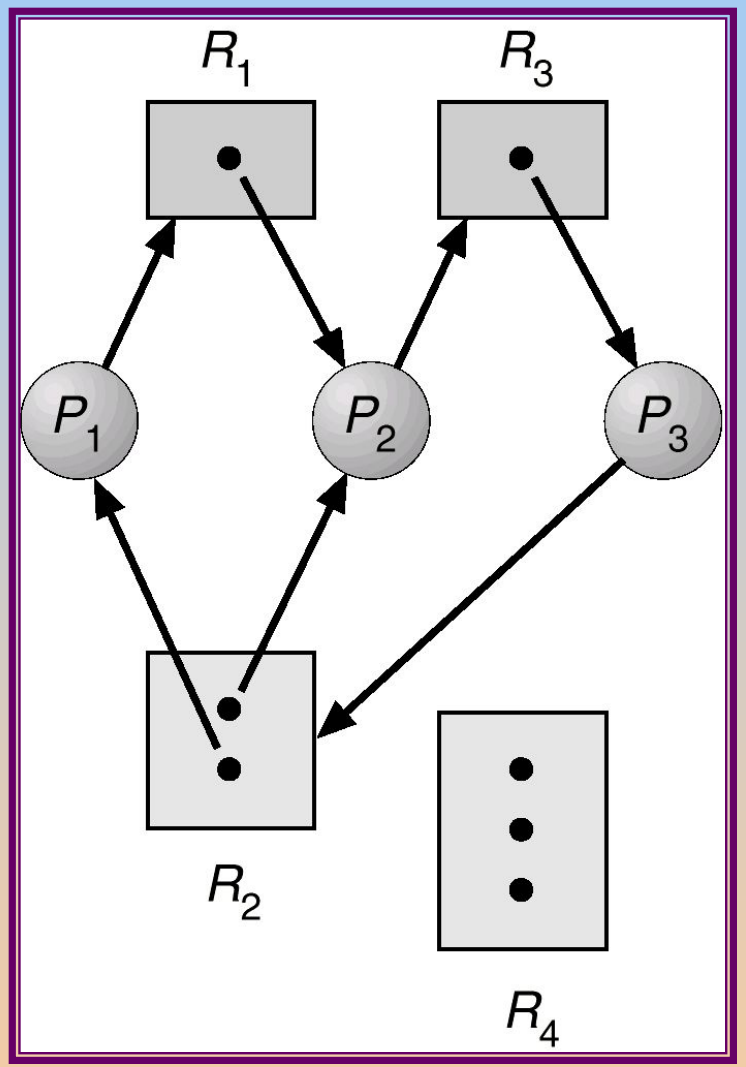




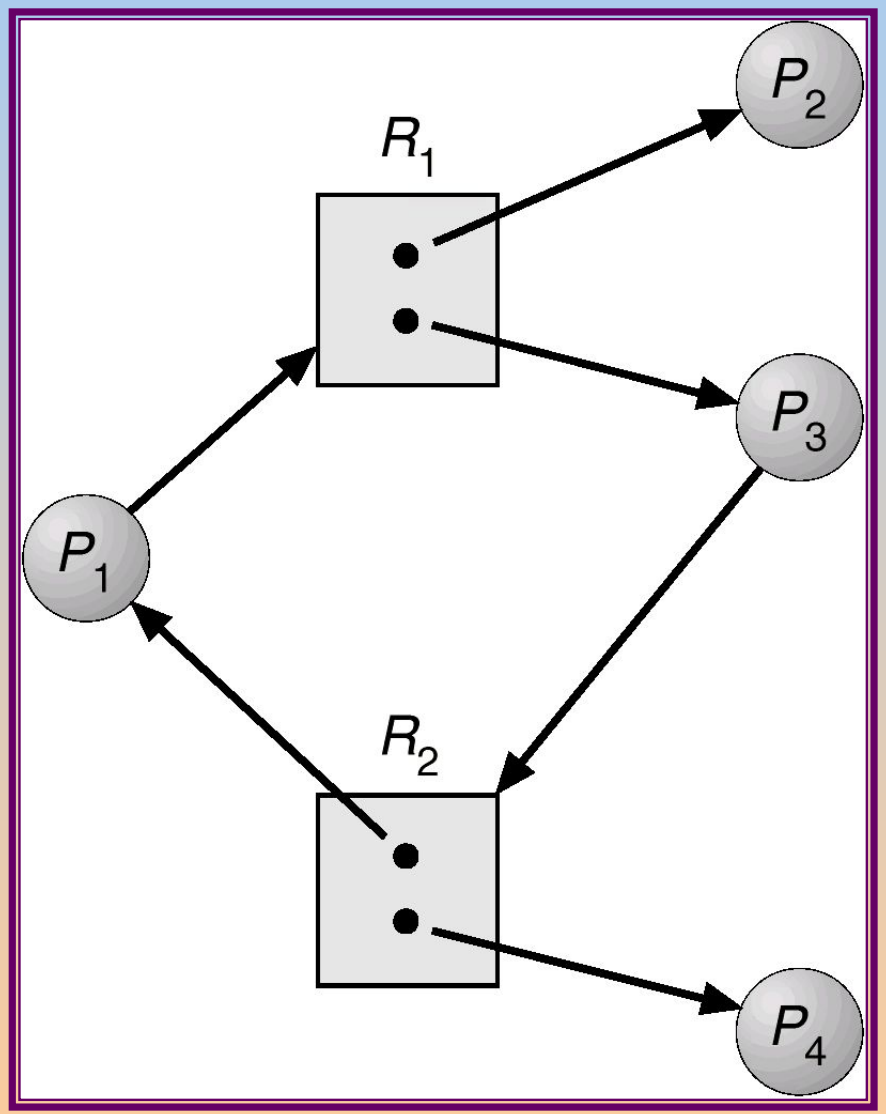
- If each resource in the graph has exactly one instance of a resource then existence of cycle in the graph implies deadlock.
- But if multiple instances of a resource exist then occurrence of cycle does not necessarily imply that deadlock has occurred.
- So for single instance resource allocation, existence of cycle is necessary and sufficient condition for deadlock, whereas for multiple instance resources, occurrence of cycle is necessary but not sufficient condition for deadlock.

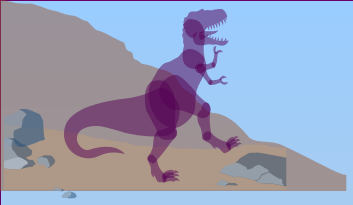


Resource Allocation Graph With A Deadlock



Resource Allocation Graph With A Cycle But No Deadlock

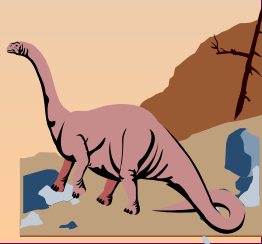


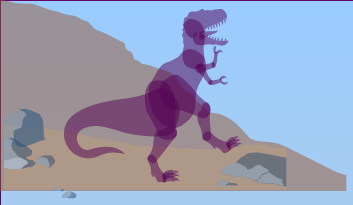


Methods for Handling Deadlocks

We can deal with deadlocks in one of the three ways:

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





- **Deadlock Prevention** – ensures that deadlock will never occur by making sure that one of the 4 conditions required for deadlock is not met.
- **Deadlock Avoidance** – requires that OS should know beforehand about the resource requirements of the processes.

  OS then makes sure before allocating resources to the processes that it does not allocate such that cycle exist in the resource allocation graph.