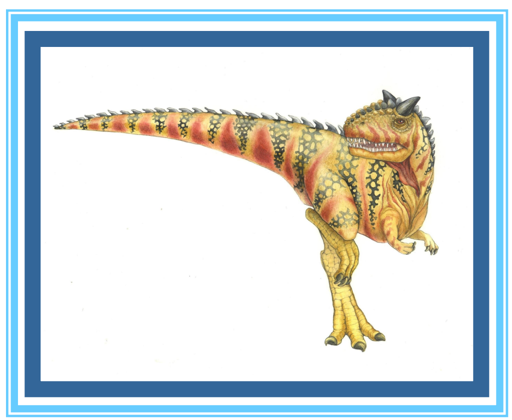
Chapter 7: Deadlocks
Lecture #09



Chapter 7: Deadlocks

- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

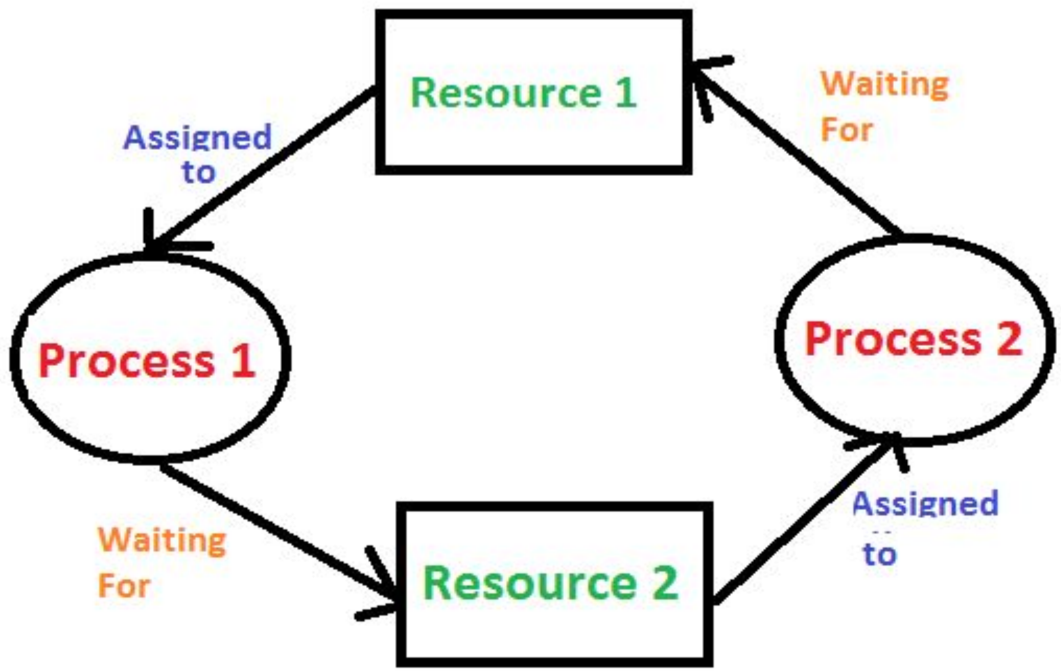
Chapter Objectives

- To develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks
- To present a number of different methods for preventing or avoiding deadlocks in a computer system

System Model

- System consists of resources
- Resource types $R_1, R_2, \ldots, R_m$

  CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - **request**
  - **use**
  - **release**



Real World Deadlock Example

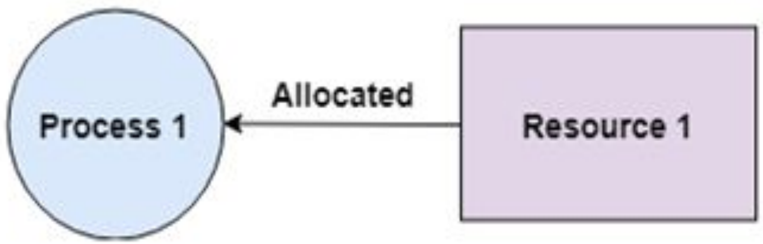
Example of Deadlock in OS

- A deadlock involving same resource type:
  - To illustrate a deadlocked state, consider a system with three CD RW drives.
  - Suppose each of three processes holds one of these CD RW drives. If each process now requests another drive, the three processes will be in a deadlocked state. Each is waiting for the event “CD RW is released,” which can be caused only by one of the other waiting processes. This example illustrates a deadlock involving the same resource type.
- A deadlock involving different resource types:
  - Deadlocks may also involve different resource types. For example, consider a system with one printer and one DVD drive. Suppose that process Pi is holding the DVD and process Pj is holding the printer. If Pi requests the printer and Pj requests the DVD drive, a deadlock occurs.

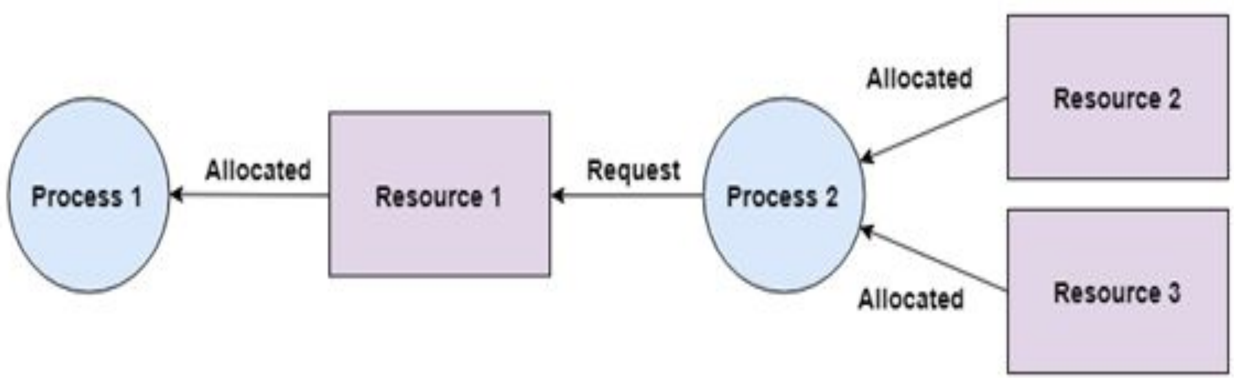
Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

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

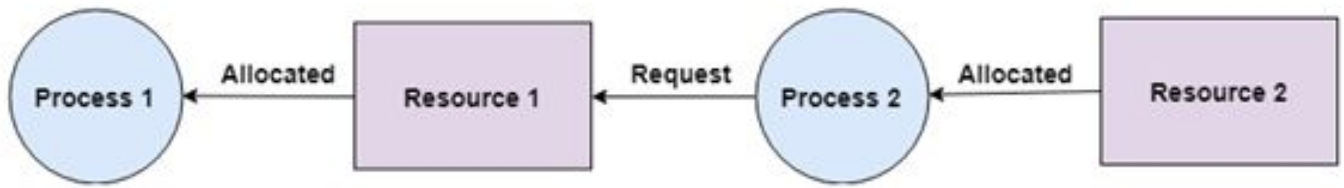


- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes

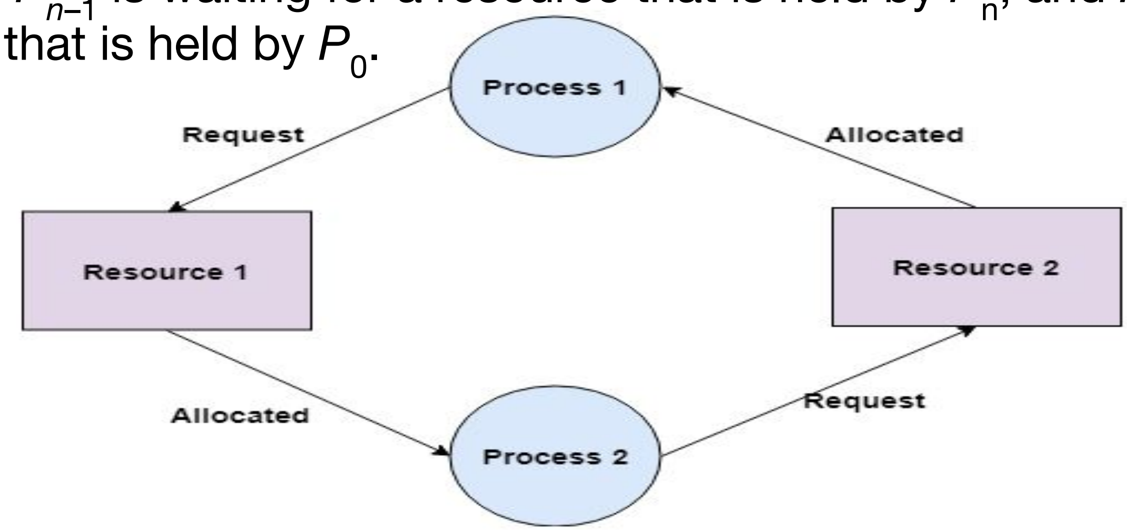


Deadlock Characterization

- **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 $\{P_0, P_1, \ldots, P_n\}$ 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 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)

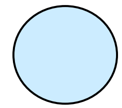
- An arrow from the **process** to **resource** indicates the process is **requesting** the resource. An arrow from **resource** to **process** shows an instance of the resource has been **allocated** to the process.

- **request edge** – directed edge $P_i \rightarrow R_j$

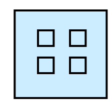
- **assignment edge** – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (Cont.)

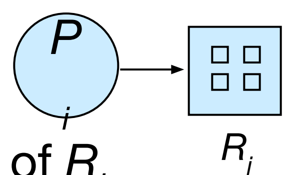
- Process



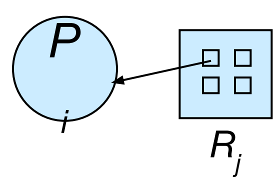
- Resource Type with 4 instances



- P_i requests instance of R_j

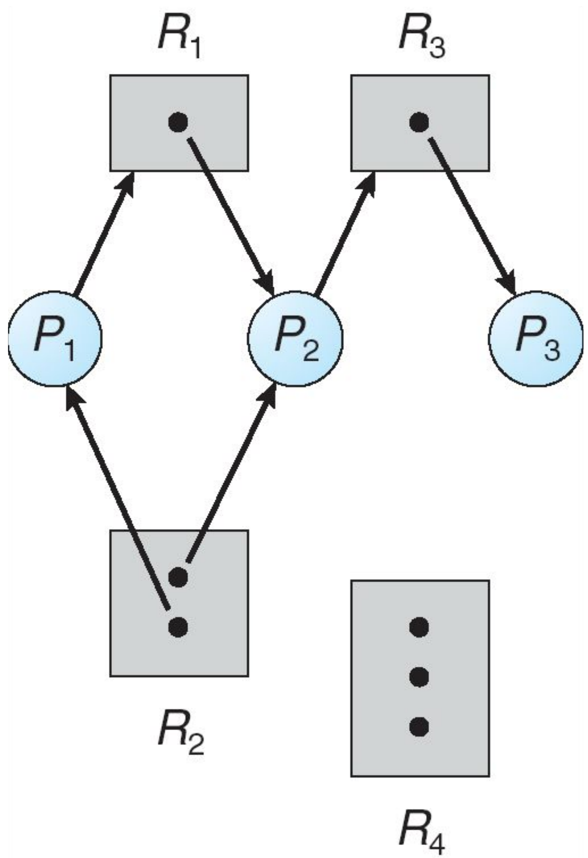


- P_i is holding an instance of R_j



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.

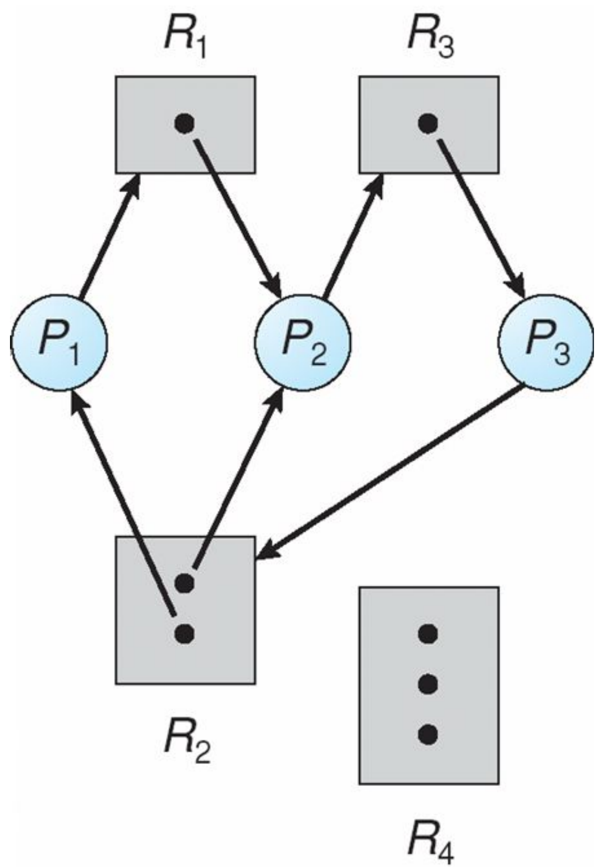
Example of a Resource Allocation Graph



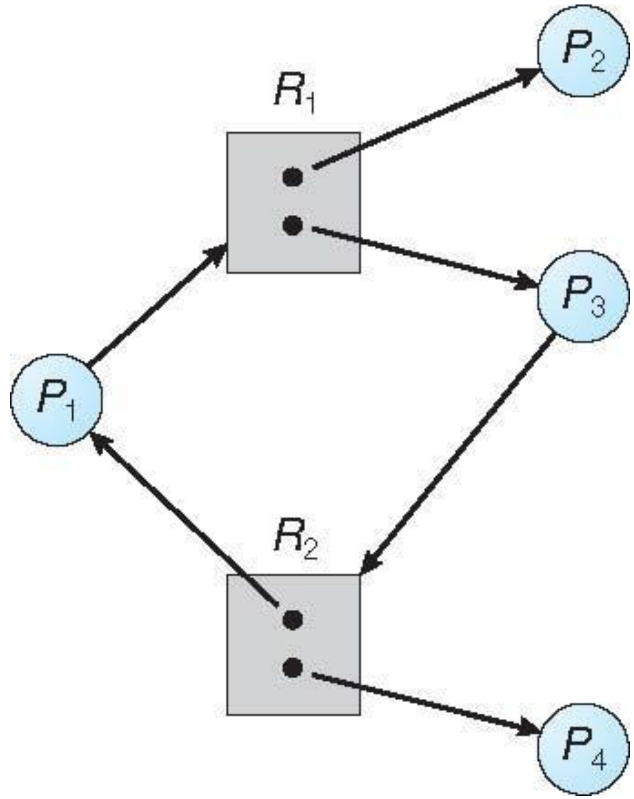
The resource-allocation graph shown in depicts the following situation. • The sets P, R, and E: ◦

- P = {P1, P2, P3}
- R = {R1, R2, R3, R4}
- E = {P1 → R1, P2 → R3, R1 → P2, R2 → P2, R2 → P1, R3 → P3}

Resource Allocation Graph With A Deadlock



Graph With A Cycle But No Deadlock



Basic Facts

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

HOW TO HANDLE DEADLOCKS – GENERAL STRATEGIES

There are three methods:

Strategy

- Ignore Deadlocks:

Most Operating systems do this!!

- Ensure deadlock **never** occurs using either
    - **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..
- **Allow** deadlock to happen. This requires using both:
    - **Detection:** Know a deadlock has occurred.
    - **Recovery** : Regain the resources.