# Operating Systems

***CSN-232***

**Dr. R. Balasubramanian**

**Associate Professor**

**Department of Computer Science and Engineering**

**Indian Institute of Technology Roorkee**

**Roorkee 247 667**

*balarfcs@iitr.ac.in*

*https://sites.google.com/site/balaiitr/*

# 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**

# 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
- **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 $P_0$.

# Deadlock with Mutex Locks

- Deadlocks can occur via system calls, locking, etc.
- mutex deadlock

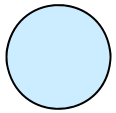
# Resource-Allocation Graph

A set of vertices $V$ and a set of edges $E$.

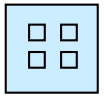
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** – directed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

# Resource-Allocation Graph (Cont.)

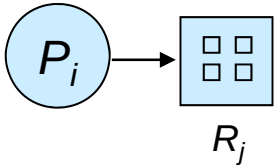
- Process



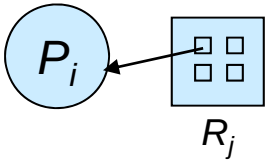
- Resource Type with 4 instances



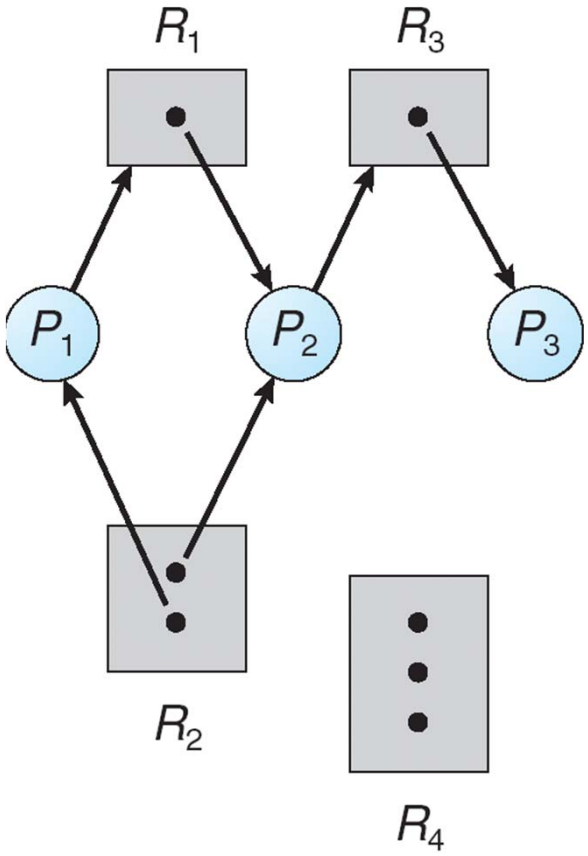
- $P_i$ requests instance of $R_j$



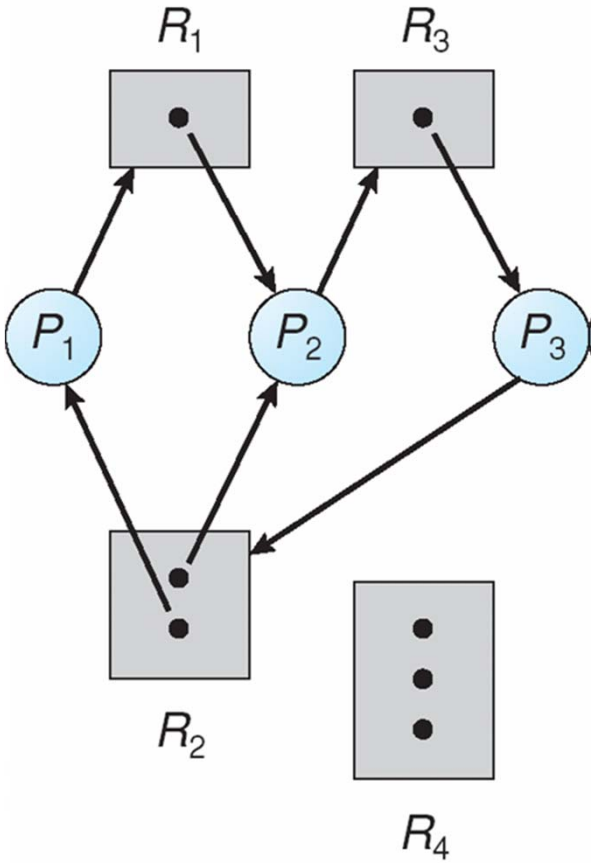

- $P_i$ is holding an instance of $R_j$



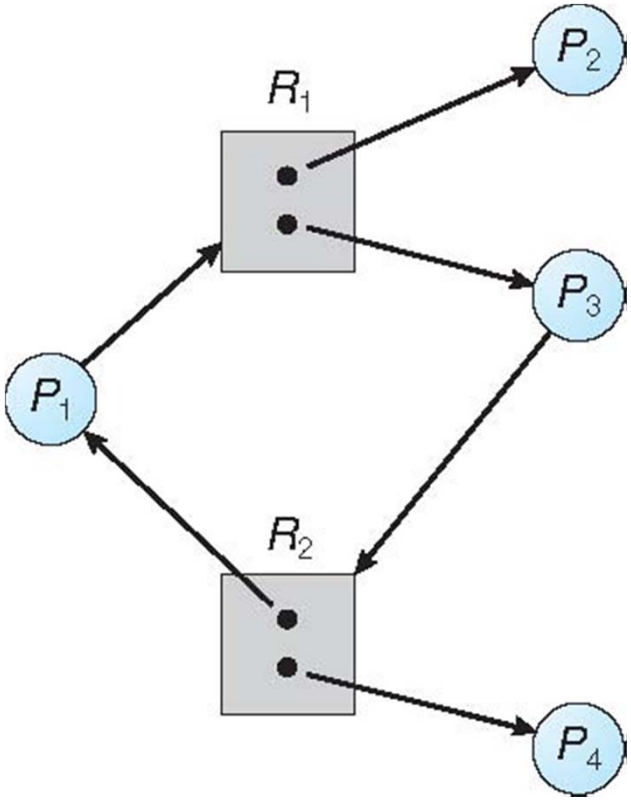
# Example of a Resource Allocation Graph



# Resource Allocation Graph With A Deadlock



# Graph With A Cycle But No Deadlock

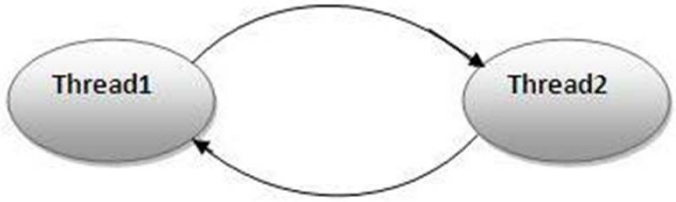


# Basic Facts

- 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

# Deadlock in JAVA

- Deadlock in JAVA is a part of multithreading.
- Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread.
- Since, both threads are waiting for each other to release the lock, the condition is called deadlock.



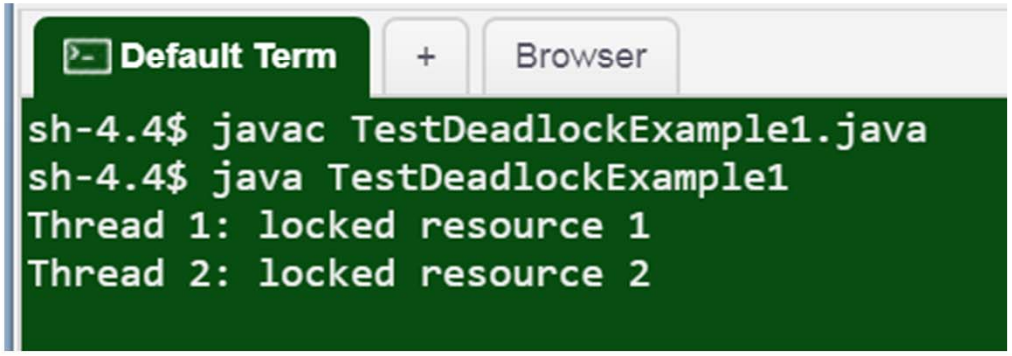
```java
public class TestDeadlockExample1 {
  public static void main(String[] args) {
    final String resource1 = "IITR";
    final String resource2 = "CSE";
    // t1 tries to lock resource1 then resource2
    Thread t1 = new Thread() {
      public void run() {
          synchronized (resource1) {
           System.out.println("Thread 1: locked resource 1");

           try { Thread.sleep(100);} catch (Exception e) {}

           synchronized (resource2) {
            System.out.println("Thread 1: locked resource 2");
           }
         }
      }
    };

```

```
19
20      // t2 tries to lock resource2 then resource1
21      Thread t2 = new Thread() {
22        public void run() {
23          synchronized (resource2) {
24            System.out.println("Thread 2: locked resource 2");
25
26            try { Thread.sleep(100);} catch (Exception e) {}
27
28            synchronized (resource1) {
29              System.out.println("Thread 2: locked resource 1");
30            }
31          }
32        }
33      };
34
35
36      t1.start();
37      t2.start();
38    }
39  }
```




**https://goo.gl/NX3SVF**