# SNS COLLEGE OF TECHNOLOGY


**Coimbatore-35.**

**An Autonomous Institution**

**Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade**
**Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai**


## COURSE NAME : 19CSB201 – OPERATING SYSTEMS

## II YEAR/ IV SEMESTER

## UNIT – II Process Scheduling And Synchronization

## Topic: Deadlock: System Model & Characterization


Mr.N.Selvakumar

Assistant Professor

Department of Computer Science and Engineering

# Deadlocks

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

# Deadlock

A process requests resources; if the resources are not available at that time, the process enters a waiting state.

Sometimes, a waiting process is never again able to change state, because the **resources it has requested are held by other waiting processes**. This situation is called a deadlock.

# 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**

- A process must **request** a resource before using it and must **release** the resource after using it.
- A process **may request as many resources** as it requires to carry out its designated task.
- Obviously, the number of resources requested may **not exceed the total number of resources available** in the system. In other words, a process cannot request three printers if the system has only two.

# Deadlock Characterization

## Necessary Conditions

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 $P_0$.

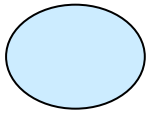
# Resource-Allocation Graph

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

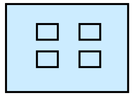
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the **active processes** in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all **resource** types in the system
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment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

# Resource-Allocation Graph (Cont.)

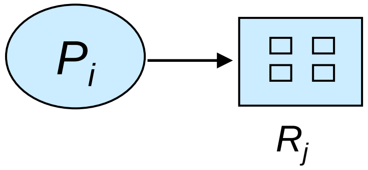
- Process



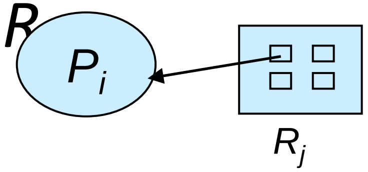
- Resource Type with 4 instances



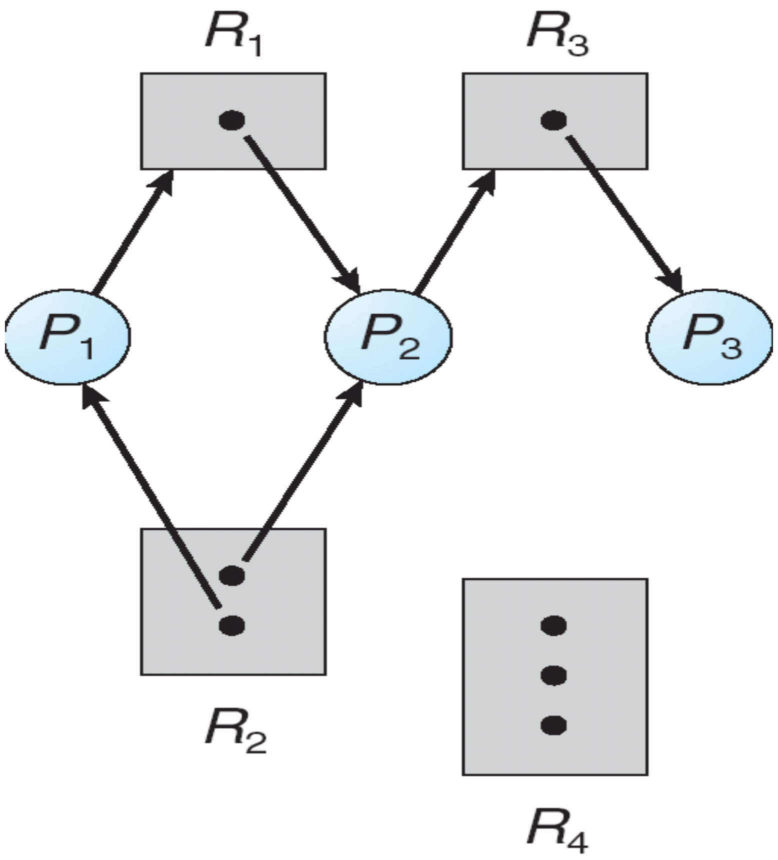
- $P_i$ requests instance of $R_j$




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

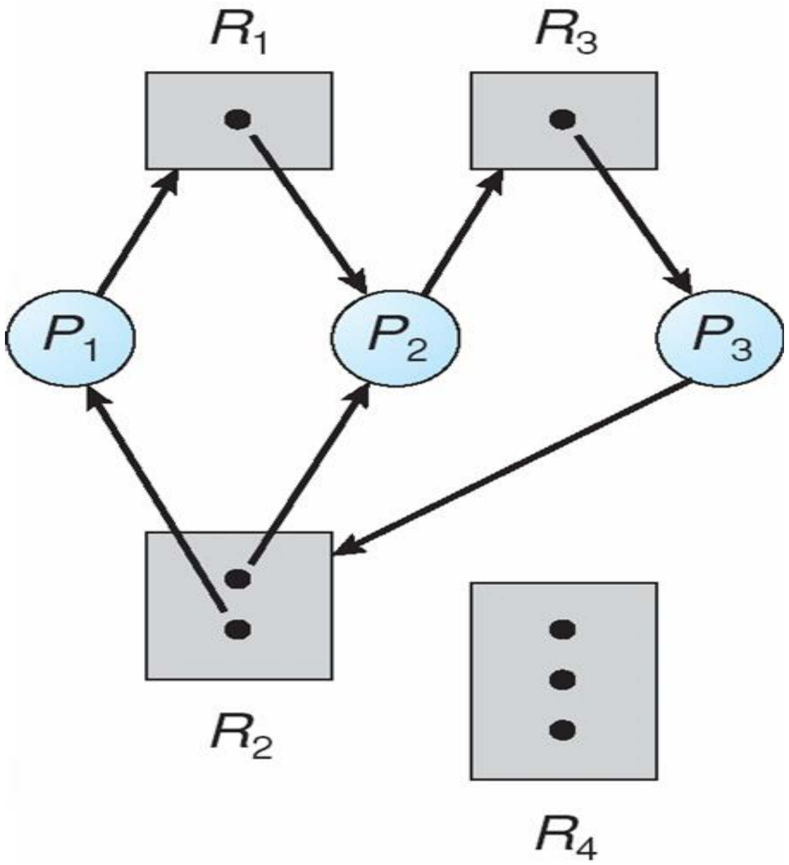


# Example of a Resource Allocation Graph



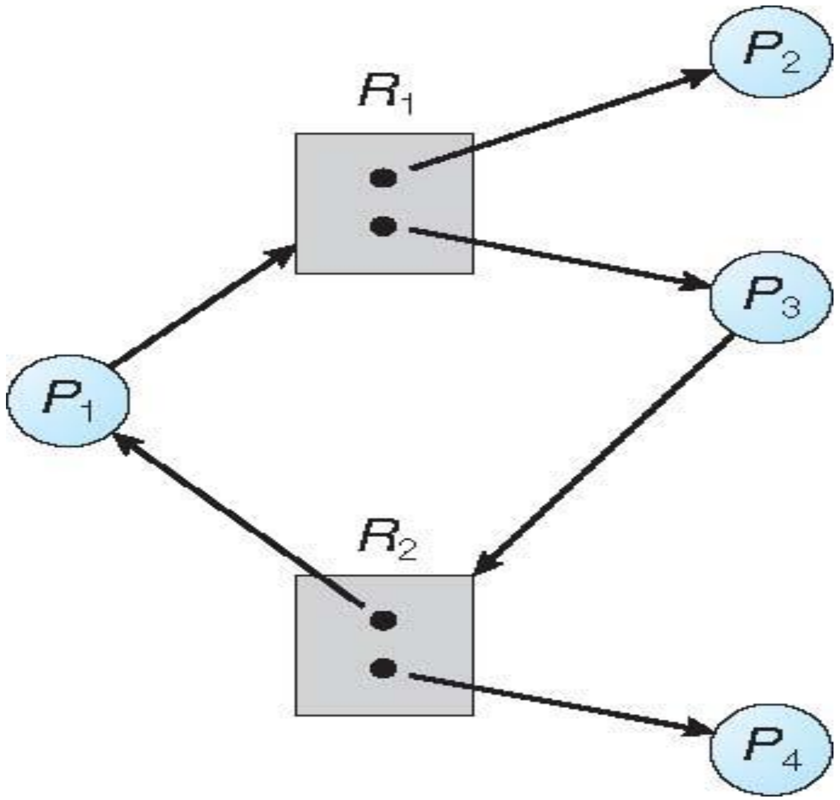

- The sets $P$, $R$, and $E$:
  - $P = \{P_1, P_2, P_3\}$
  - $R = \{R_1, R_2, R_3, R_4\}$
  - $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$
- Resource instances:
  - One instance of resource type $R_1$
  - Two instances of resource type $R_2$
  - One instance of resource type $R_3$
  - Three instances of resource type $R_4$
- Process states:
  - Process $P_1$ is holding an instance of resource type $R_2$ and is waiting for an instance of resource type $R_1$.
  - Process $P_2$ is holding an instance of $R_1$ and an instance of $R_2$ and is waiting for an instance of $R_3$.
  - Process $P_3$ is holding an instance of $R_3$.

# Resource Allocation Graph **With A Deadlock**




$$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$$
$$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$$

# 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**

# Methods for Handling Deadlocks

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

- we can deal with the deadlock problem in **one of three ways**:
- We can use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlocked state.
- We can allow the system to enter a deadlocked state, detect it, and recover.
- We can ignore the problem altogether and pretend that deadlocks never occur in the system.

# REFERENCES

**TEXT BOOKS:**

T1 Silberschatz, Galvin, and Gagne, “Operating System Concepts”, Ninth Edition, Wiley India Pvt Ltd, 2009.)

T2. Andrew S. Tanenbaum, “Modern Operating Systems”, Fourth Edition, Pearson Education, 2010

**REFERENCES:**

R1 Gary Nutt, “Operating Systems”, Third Edition, Pearson Education, 2004.

R2 Harvey M. Deitel, “Operating Systems”, Third Edition, Pearson Education, 2004.

R3 Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, “Operating System Concepts”, 9th Edition, John Wiley and Sons Inc., 2012.

R4. William Stallings, “Operating Systems – Internals and Design Principles”, 7th Edition, Prentice Hall, 2011


THANK
YOU!