SAVEETHA ENGINEERING COLLEGE

AUTONOMOUS

NAAC A GRADE

Accredited by NBA

Affiliated to Anna University | Approved by AICTE

19CS405 OPERATING SYSTEMS

Topic – Deadlock

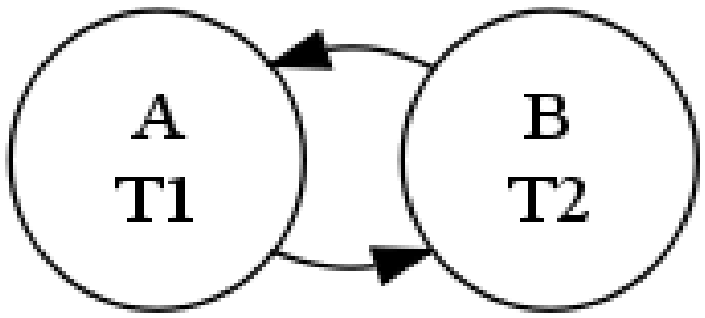
Dr. K. Suresh Kumar,
Associate Professor,
Department of Information Technology.

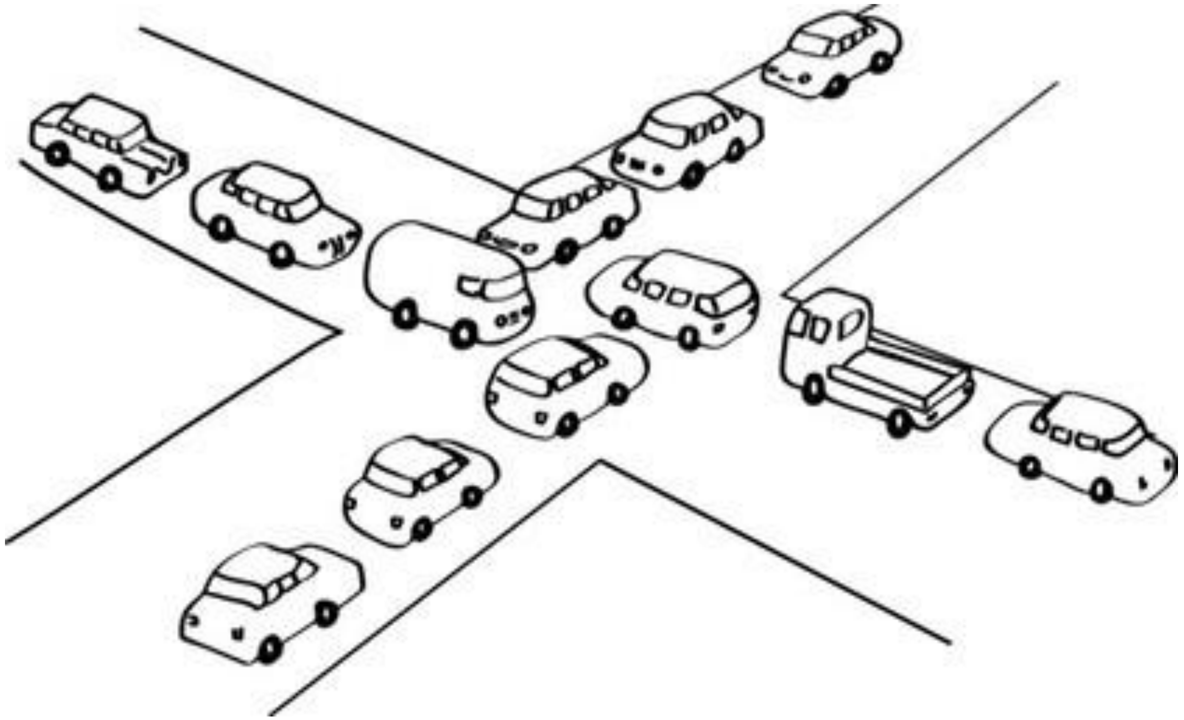
UNIT II - PROCESS MANAGEMNET

Processes - Process Concept, Process Scheduling, Operations on Processes, Inter-process Communication; CPU Scheduling - Scheduling criteria, Scheduling algorithms, Multiple processor scheduling, Real time scheduling; Threads- Overview, Multithreading models, Threading issues; Process Synchronization - The critical-section problem, Synchronization hardware, Mutex locks, Semaphores, Classic problems of synchronization, Critical regions, Monitors**; Deadlock – System model, Deadlock characterization, Methods for handling deadlocks,** Deadlock prevention, Deadlock avoidance, Deadlock detection, Recovery from deadlock.

What is Dead Lock?

- A set of ***two or more processes are deadlocked*** if they are ***blocked*** (i.e., in the waiting state) each ***holding a resource and waiting to acquire a resource held by another process*** in the set.
- A process is deadlocked if it is waiting for an event which is never going to happen.





Example:

- a system has two tape drives (T1,T2)
- two processes are deadlocked if each holds one tape drive and has requested the other

Example

Example: semaphores A and B, each initialized to 1:

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

Deadlock depends on the dynamics of the execution.

Illustrates that it is difficult to identify and test for deadlocks which may occur only under certain circumstances.

System Model

System model:

resource types: $R_1, R_2, ..., R_n$

each resource R has W_i instances

each process utilizes a resource as follows:

// ***request*** (e.g., open() system call)

// ***use***

// ***release*** (e.g., close() system call)

Any instance of a resource type can be used to satisfy a request of that resource.

Conditions Necessary for Deadlock

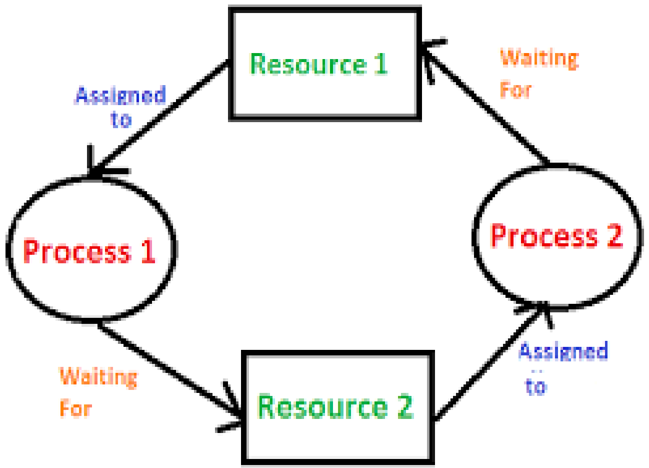
All of the following ***four necessary conditions*** must hold simultaneously for deadlock to occur:

mutual exclusion: only one process can use a resource at a time.

hold and wait: a process holding at least one resource is waiting to acquire additional resources which are currently held by other processes.

no pre-emption: a resource can only be released voluntarily by the process holding it.

circular wait: a cycle of process requests exists



Circular wait implies the hold and wait condition. Therefore, these conditions are not completely independent.

Resource Allocation Graph

A resource allocation graph contains a set of ***vertices V*** and a set of ***edges E***.

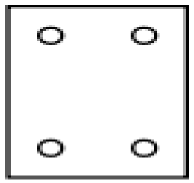
V is partitioned into two types:

- ✓ **P** = $\{P_1, P_2, \ldots, P_n\}$ is the set of all processes.
- ✓ **R** = $\{R_1, R_2, \ldots, R_m\}$ is the set of all resources.

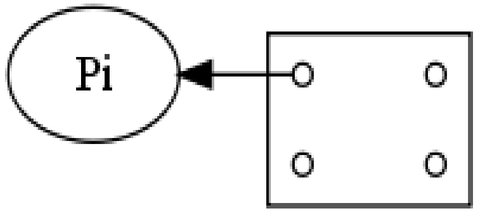
A ***request*** is represented by a directed edge from P_i to R_j.

An ***assignment*** is represented by a directed edge from R_j to P_i.

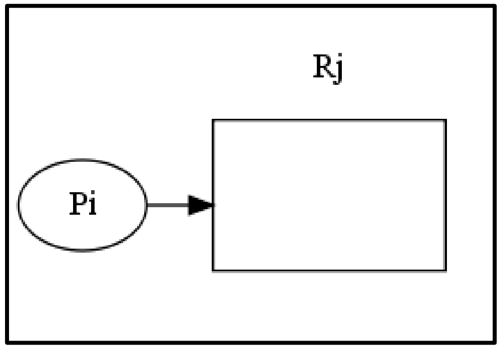
resource type with four instances:



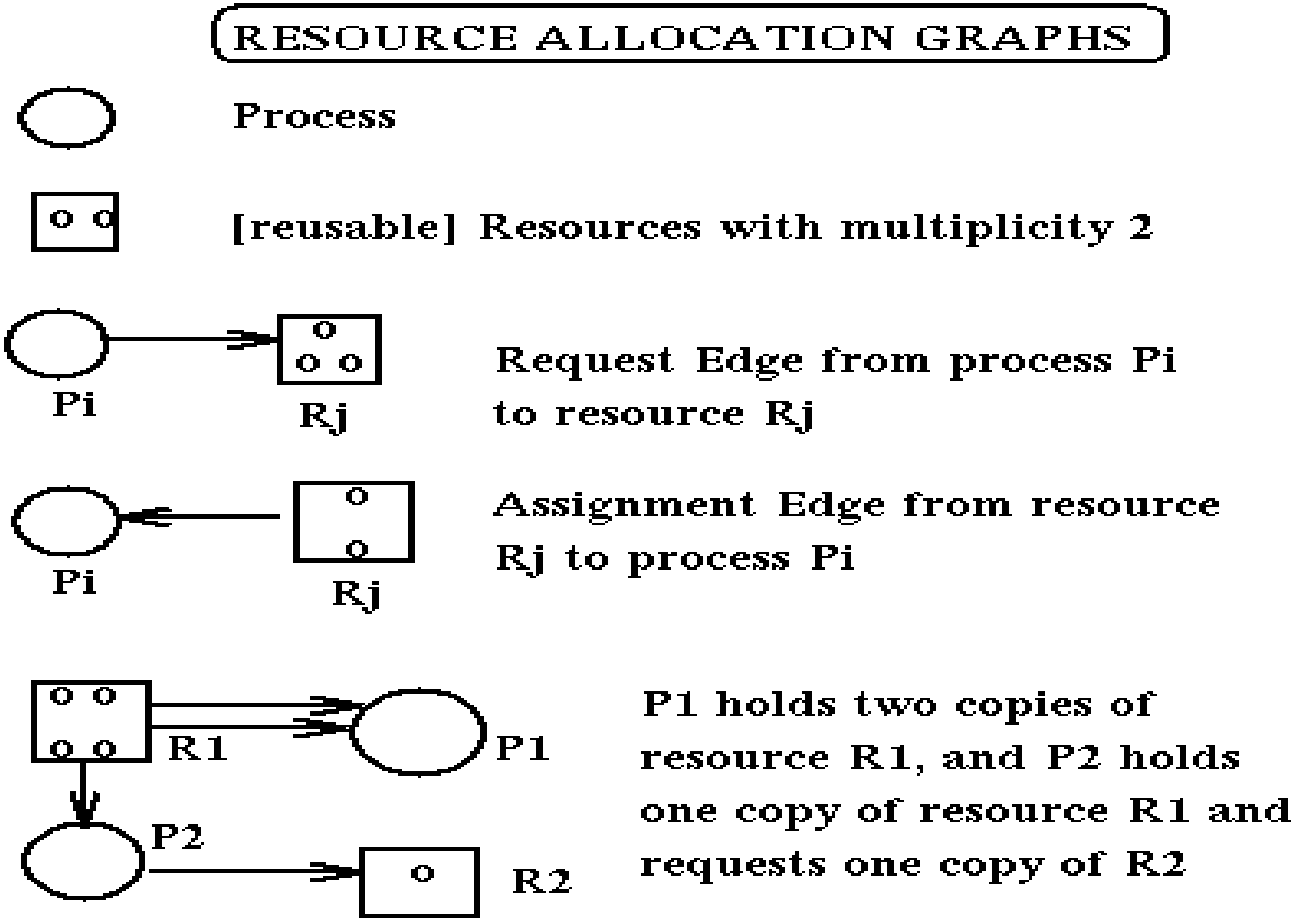
P_i is holding an instance of R_j



P_i requests an instance of R



Sample Resource Allocation Graphs

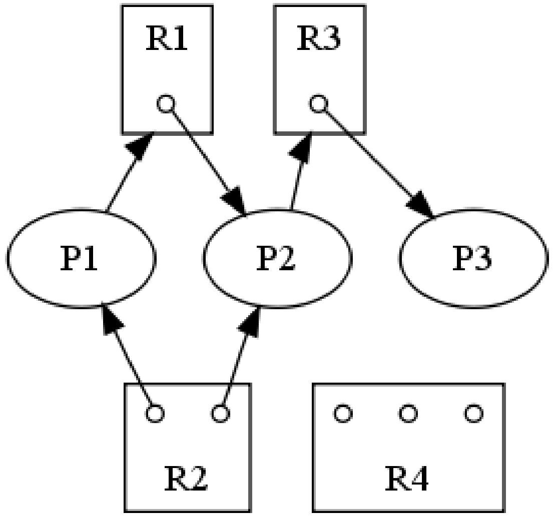


Sample Resource Allocation Graphs

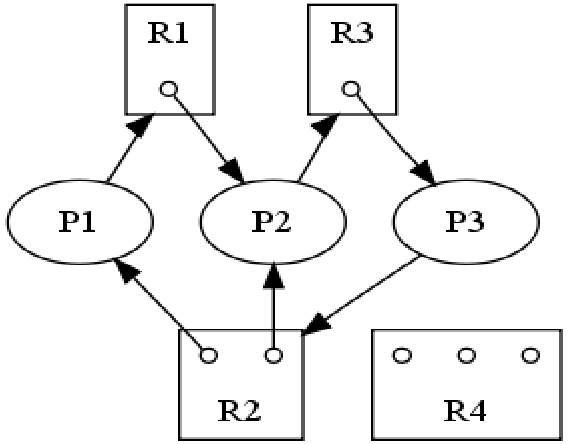
resource allocation graph without deadlock

P_1 wants a resource held by P_2

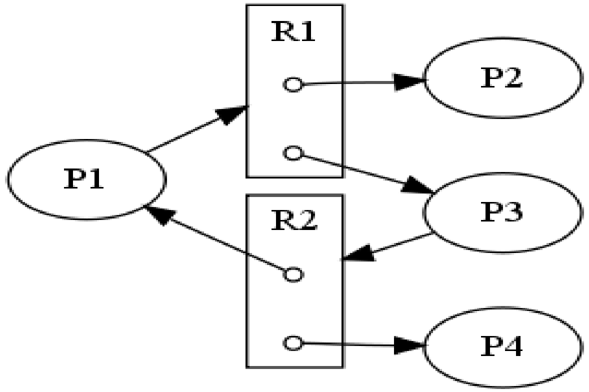
no process is requesting an instance of R_4



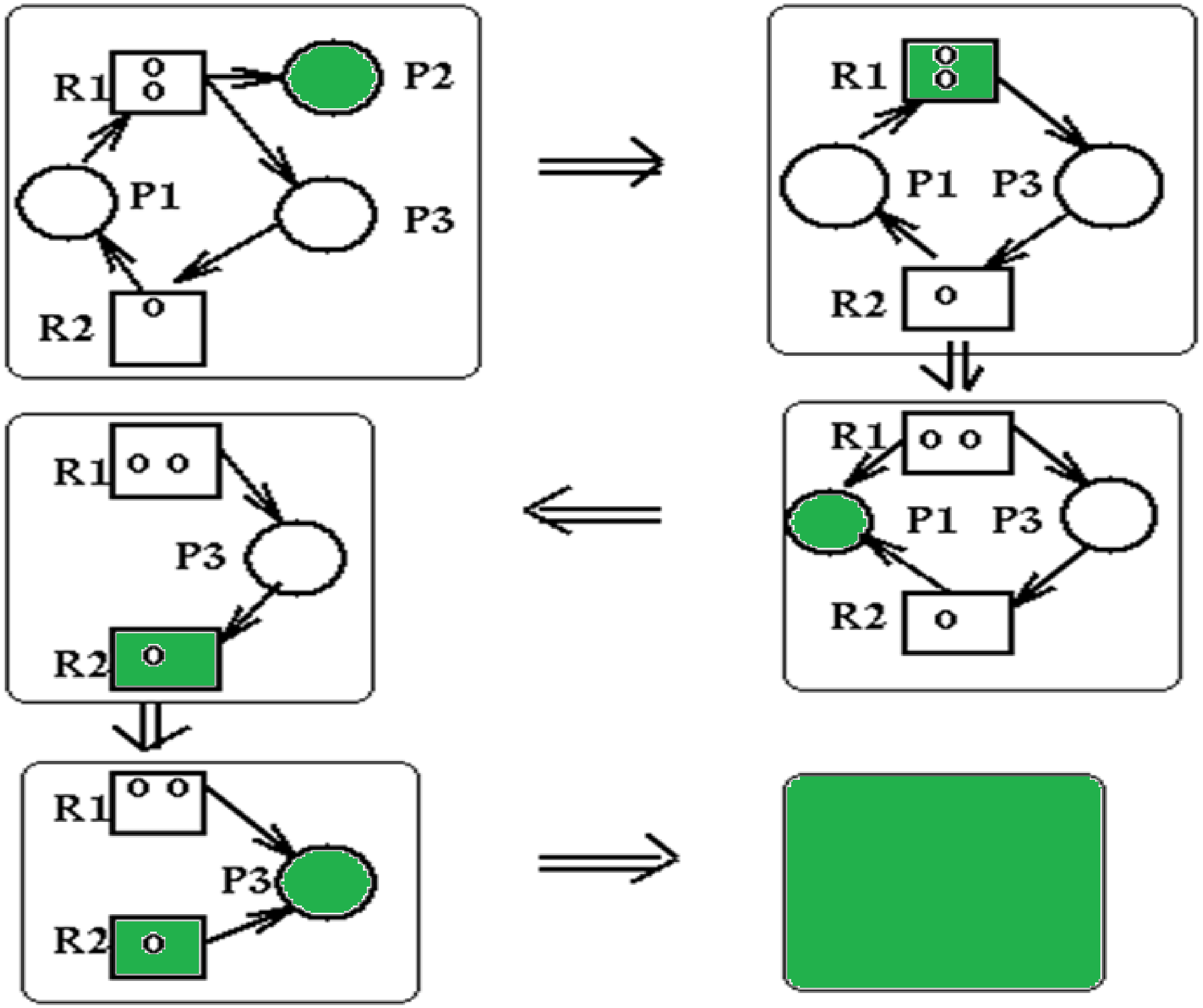
resource allocation graph with a cycle and deadlock



resource allocation graph with a cycle but no deadlock

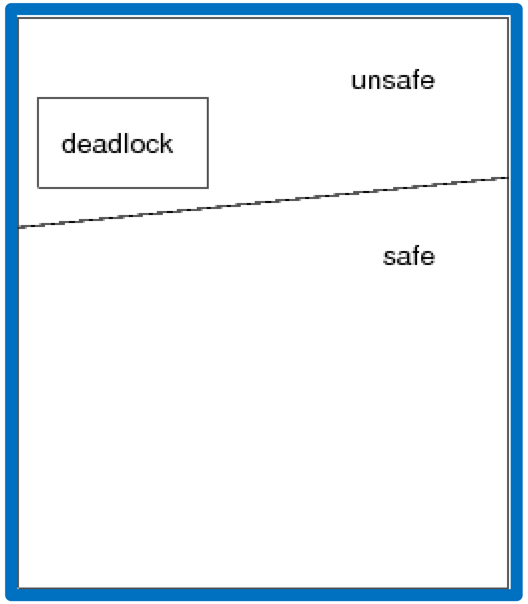


Sample Resource Allocation Graphs



Possibility of Deadlock

- ✓ If a resource allocation graph contains ***no cycles***, then **no process is deadlocked**.
- ✓ If a resource allocation graph ***contains a cycle***, then a **deadlock <u>may</u> exist**.
- ✓ Therefore, ***a cycle*** means deadlock is ***possible***, but ***not necessarily present***.
- ✓ A cycle is not sufficient proof of the presence of deadlock. A cycle is a *necessary* condition for deadlock, but not a *sufficient* condition for deadlock.
- ✓ difference between *necessary* and *sufficient*



<u>Example</u>

- ✓ getting a 6.0 GPA is *sufficient* to graduate, but it is not *necessary*
- ✓ passing OS subject is *necessary*, but not *sufficient*

Resource Allocation Graph Summary

- ✓ if a resource allocation graph ***does not contain a cycle***, then there is ***absolutely no possibility of deadlock.***
- ✓ if a resource allocation graph contains ***a cycle***, then there is the ***possibility of deadlock.***
- ✓ if each resource type has exactly ***one instance***, then a ***cycle implies*** that ***deadlock has occurred.***
- ✓ if the ***cycle involves*** only ***a set of resource types***, each of which has ***only a single instance***, then a ***deadlock has occurred.***
- ✓ if ***all instances*** of a resource are ***allocated to a process*** in a cycle, then ***there is deadlock.***

Methods for Handling Deadlock

- The following are methods for addressing the possibility of deadlock: ensure that the system never enters a deadlocked state:

✓ **Deadlock prevention** - *Deadlock prevention* means to block at least one of the four conditions required for **deadlock** to occur. If we are able to block any one of them then ***deadlock can be prevented***.

✓ **Deadlock avoidance** - the request for any resource will be granted if the ***resulting state of the system doesn't cause deadlock*** in the system. The state of the system will continuously be checked for safe and unsafe states,

✓ **Deadlock detection and recovery**: allow the system to enter a deadlocked state, then deal with and eliminate the problem