OPERATING SYSTEMS

Deadlocks - 1

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Course Syllabus - Unit 2

12 Hours

Unit 2: Threads & Concurrency

Introduction to Threads, types of threads, Multicore Programming, Multithreading Models, Thread creation, Thread Scheduling, PThreads and Windows Threads, Mutual Exclusion and Synchronization: software approaches, principles of concurrency, hardware support, Mutex Locks, Semaphores. Classic problems of Synchronization: Bounded-Buffer Problem, Readers -Writers problem, Dining Philosophers Problem concepts. Synchronization Examples - Synchronisation mechanisms provided by Linux/Windows/Pthreads. Deadlocks: principles of deadlock, tools for detection and Prevention.

OPERATING SYSTEMS

Course Outline

13	Introduction to Threads, types of threads, Multicore Programming, Multithreading Models	4.1 - 4.3	42.8
14	Thread creation, Thread Scheduling	5.4	
15	Pthreads and Windows Threads	4.4	
16	Mutual Exclusion and Synchronization: software approaches,	6.1-6.2	
17	principles of concurrency, hardware support	6.3-6.4	
18	Mutex Locks, Semaphores	6.5, 6.6	
19	Classic problems of Synchronization: Bounded-Buffer Problem, Readers-Writers problem	6.7-6.8	
20	Dining-Philosophers Problem	6.8	
21	Synchronization Examples: Synchronisation mechanisms provided by Linux/Windows/Pthreads.	6.9	
22	Deadlocks: principles of deadlock, Deadlock Characterization	7.1-7.3	
23	Deadlock Prevention, Deadlock example	7.4-7.5	
24	Deadlock Detection, Algorithm	7.6	

Topic Outline

- Background

5

- System Model
- Deadlock Characterization
- Methods of Handling Deadlocks
  - Resource Allocation Graph
  - Deadlock Prevention

Deadlocks: Background

- **Deadlock Definition:** A set of processes are deadlocked when every process in the set is waiting for an event that can only be generated by some process in the set.

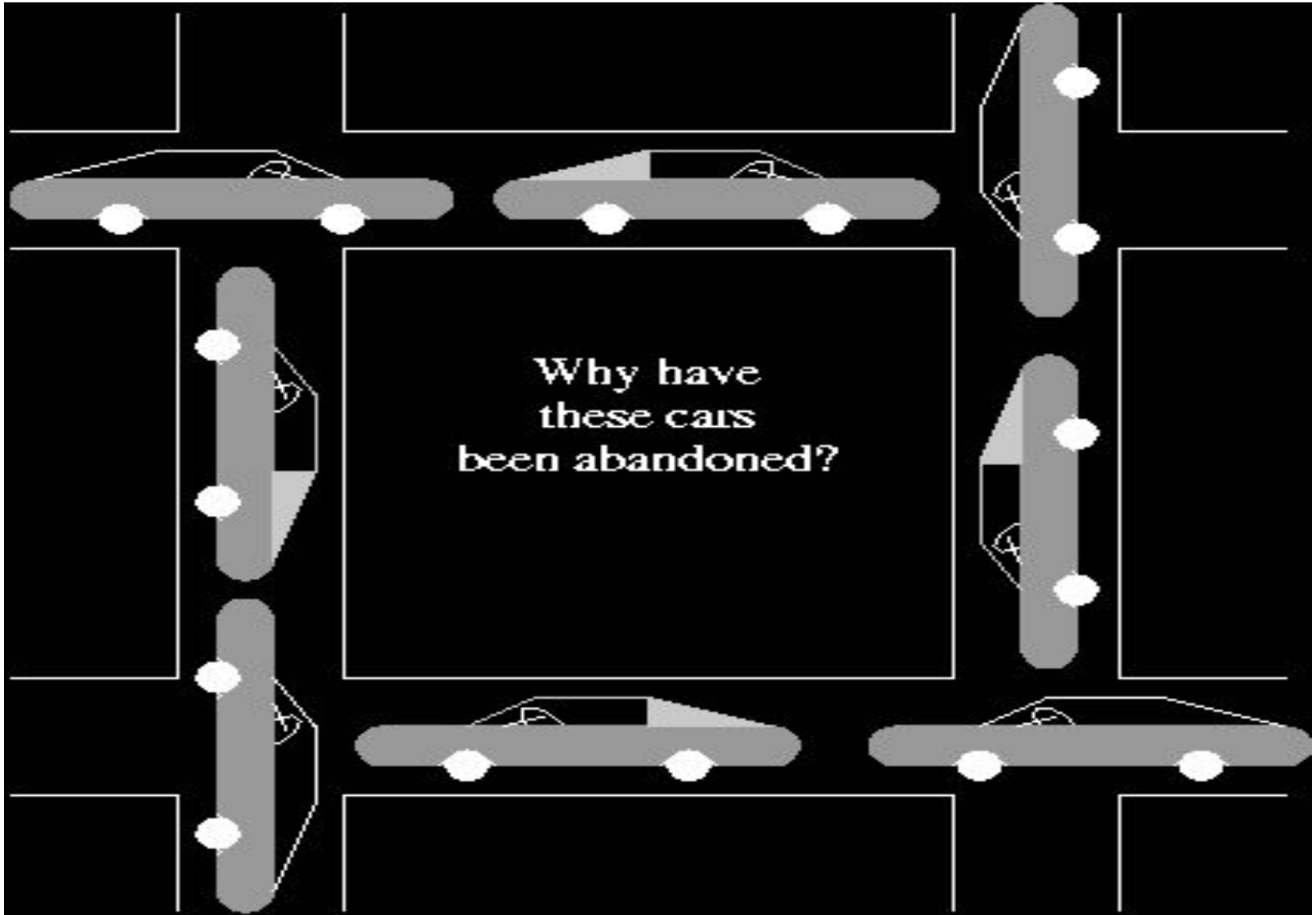
- **P={ P0, P1 }, R = { R0, R1 };**

- **R0=>Large Memory, R1=>Printer**

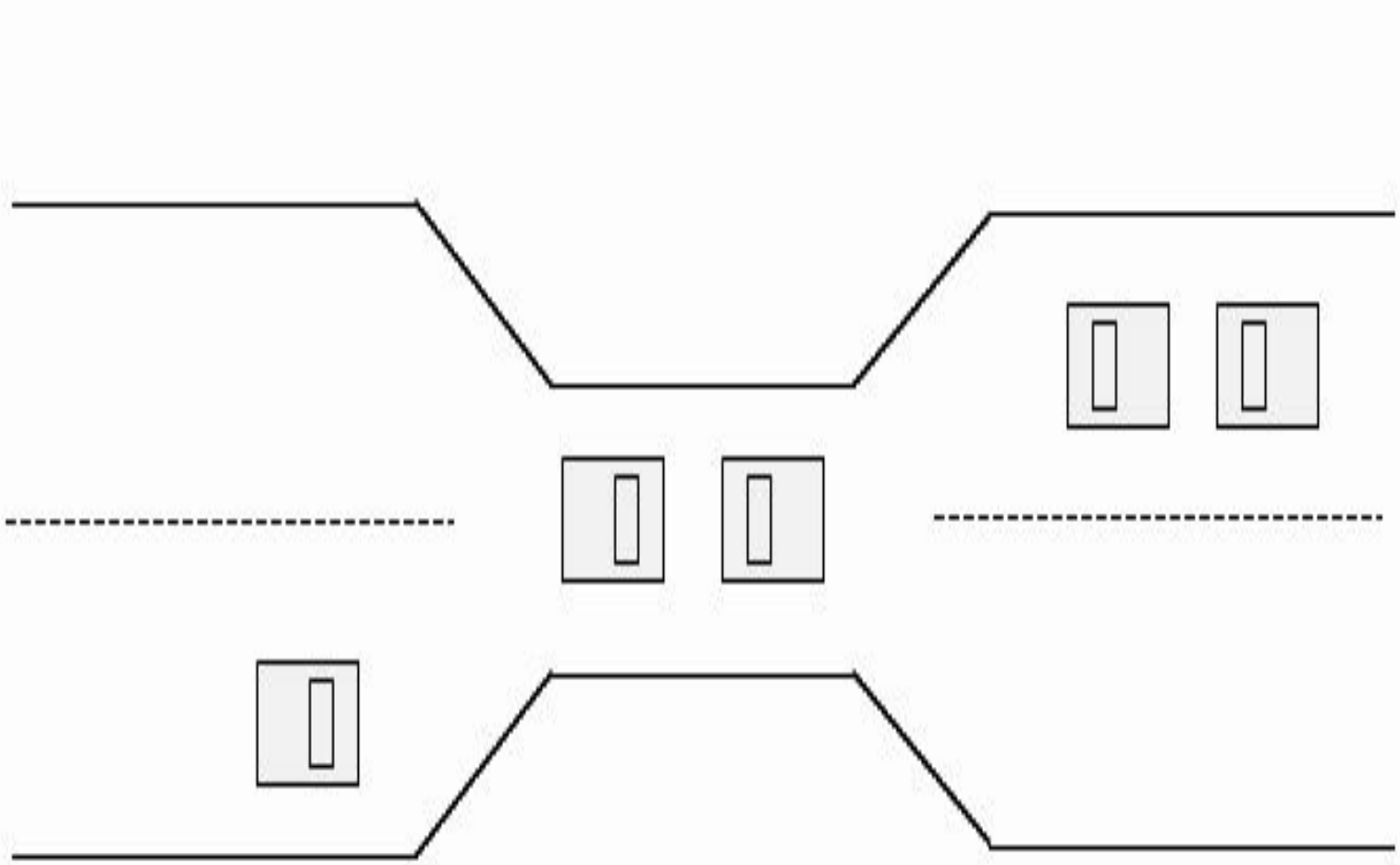
    - A process=>**P0** that is printing=>**R1** a large Image is waiting for more memory=>**R0**.
    - A visualization process=>**P1** with lots of memory=>**R0** is waiting to use the printer=>**R1**.

- **P0=>Process that is using printer R1 holding large memory R0; P1=>Visualisation Process holding large memory R0 waiting for the printer R1**

Deadlocks: Background



Deadlocks: Background

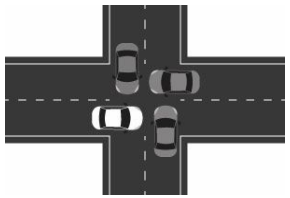


Deadlocks: Background

Deadlocks: System Model

- System consists of resources
  - Resource types=> **R = {R1 , R2 , . . ., Rm}**
    - CPU cycles, memory space, I/O devices
  - Each resource type Ri has Wi instances.
    - Example for w=5, R2w =>**{ R2.1, R2.2, R2.3, R2.4,R2.5}**

- There are n process in the system=> **P = {P0, P1 ...Pn-1}**



Deadlocks: System Model

A process may utilize a resource in only the following sequence:

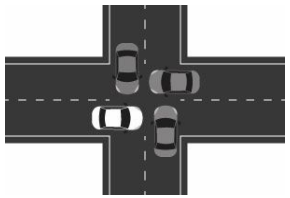
- **Request:** If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource.
- **Use:** The process can operate on the resource
- **Release:** The process releases the resource.
    - The request and release of resources are system calls
    - Request and release of other resources can be accomplished through the wait and signal operations on semaphores.

Deadlocks: System Model

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.

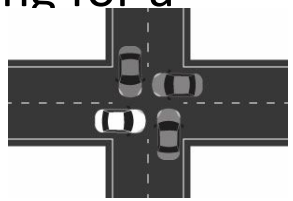
- A system consists of a finite number of resources to be distributed among a number of competing processes.

- A process must request a resource before using it, and must release the resource after using it.



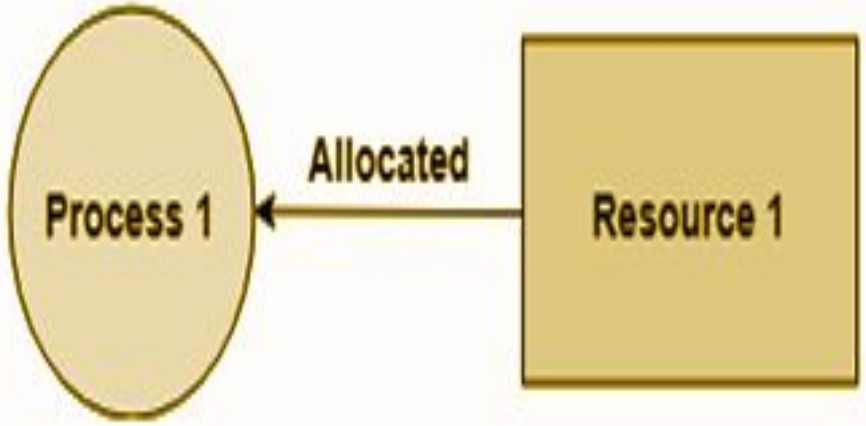
Deadlocks: Deadlock Characterization

- 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 {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 , ..., P n–1 is waiting for a resource that is held by Pn , and Pn is waiting for a resource that is held by P0



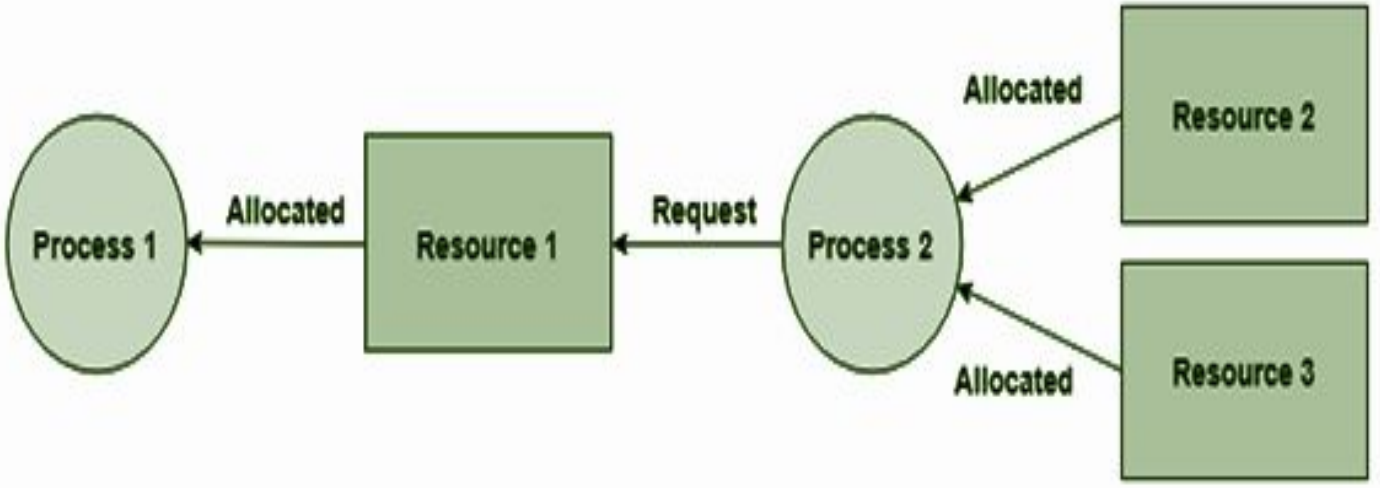
Deadlocks: Deadlock Characterization

- Deadlock can arise if four conditions hold simultaneously
  - **Mutual exclusion:** only one process at a time can use a resource



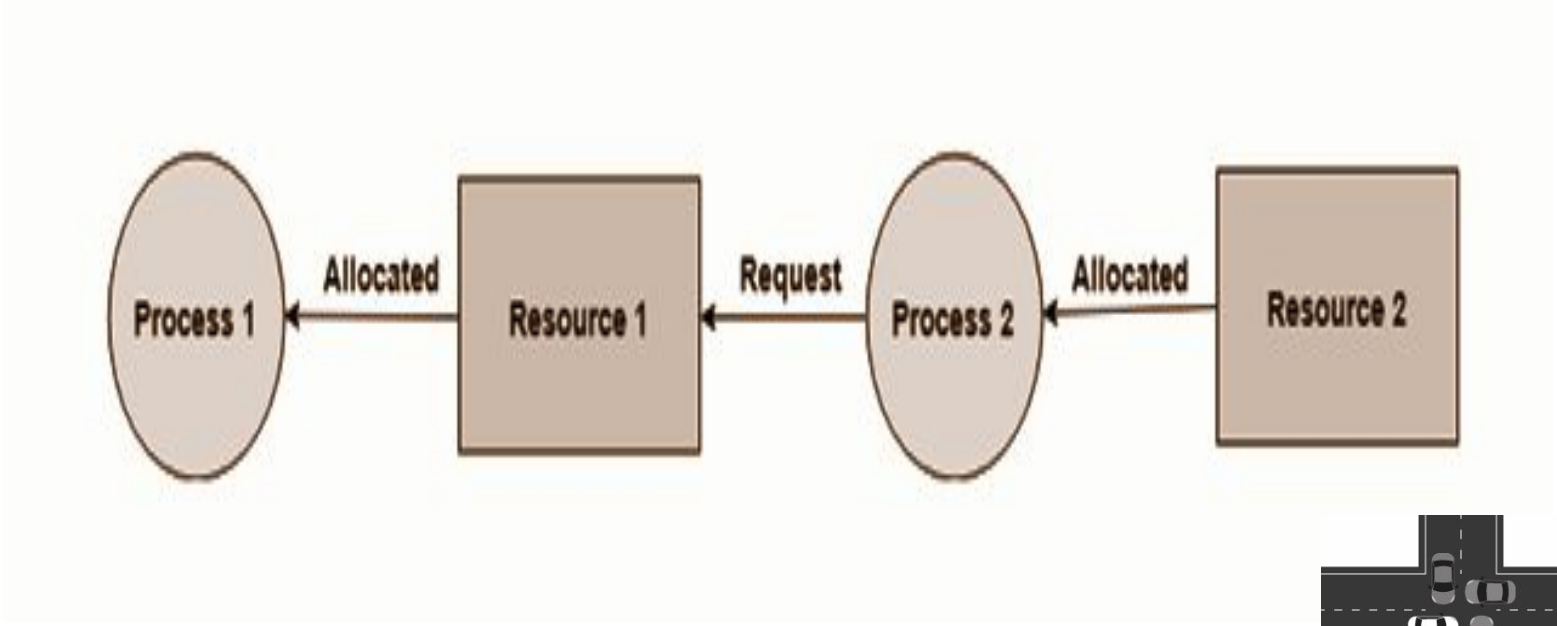
Deadlocks: Deadlock Characterization

- Deadlock can arise if four conditions hold simultaneously
  - **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes



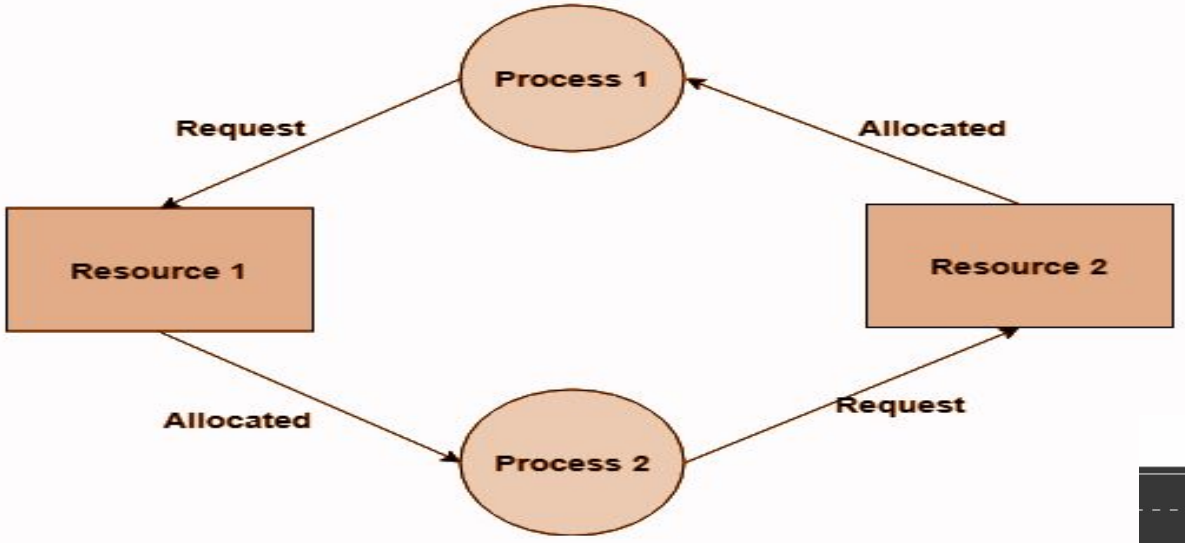
Deadlocks: Deadlock Characterization

- Deadlock can arise if four conditions hold simultaneously
  - **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task



Deadlocks: Deadlock Characterization

- Deadlock can arise if four conditions hold simultaneously
  - **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



Deadlocks: Deadlock Handling Methods

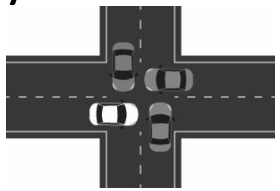
- Ensure that the system will never enter a deadlock state
    - Deadlock prevention
    - Deadlock avoidance
- Allow the system to enter a deadlock state and then recover - Deadlock Detection and Deadlock Recovery
- Ignore the problem and pretend that deadlocks never occur in the system; used by most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

Deadlocks: Resource Allocation Graph - RAG

- The resource allocation graph is the pictorial representation of the state of a system. As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.

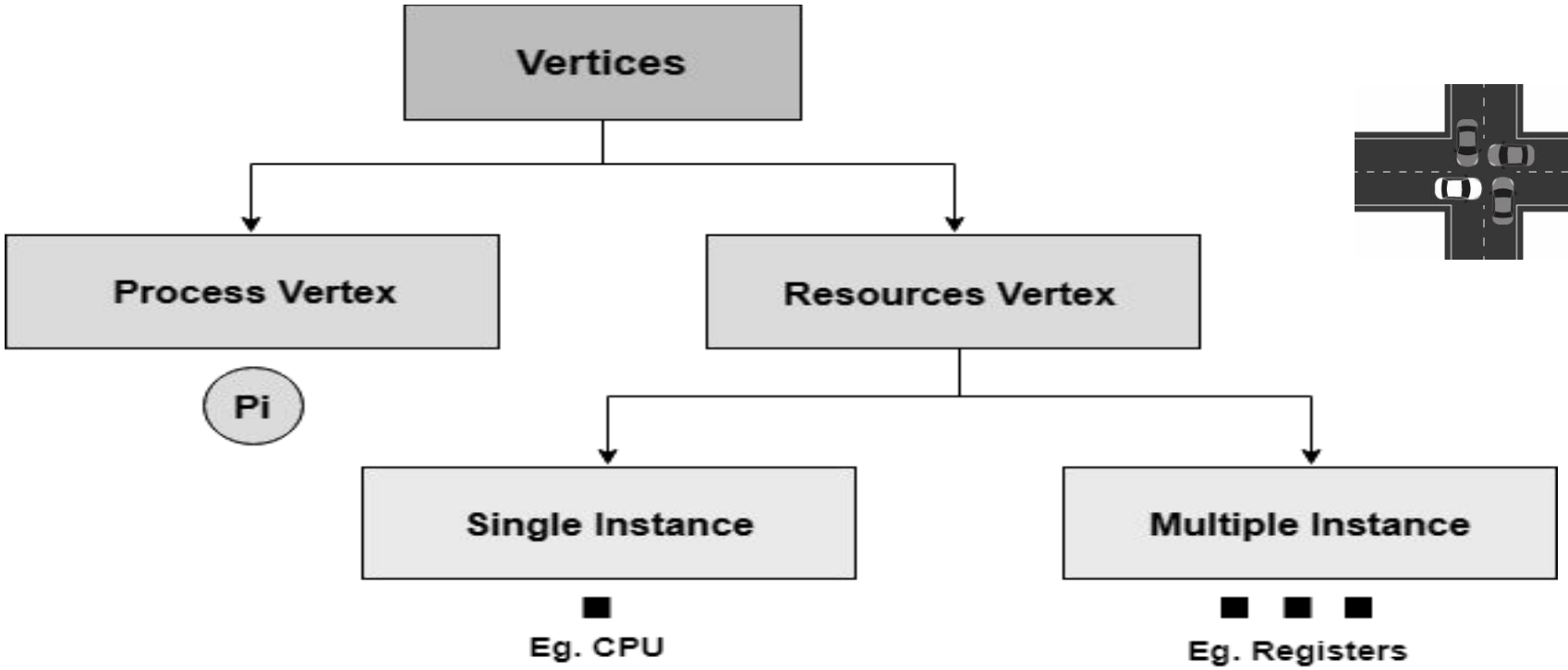
- It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

- In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle.



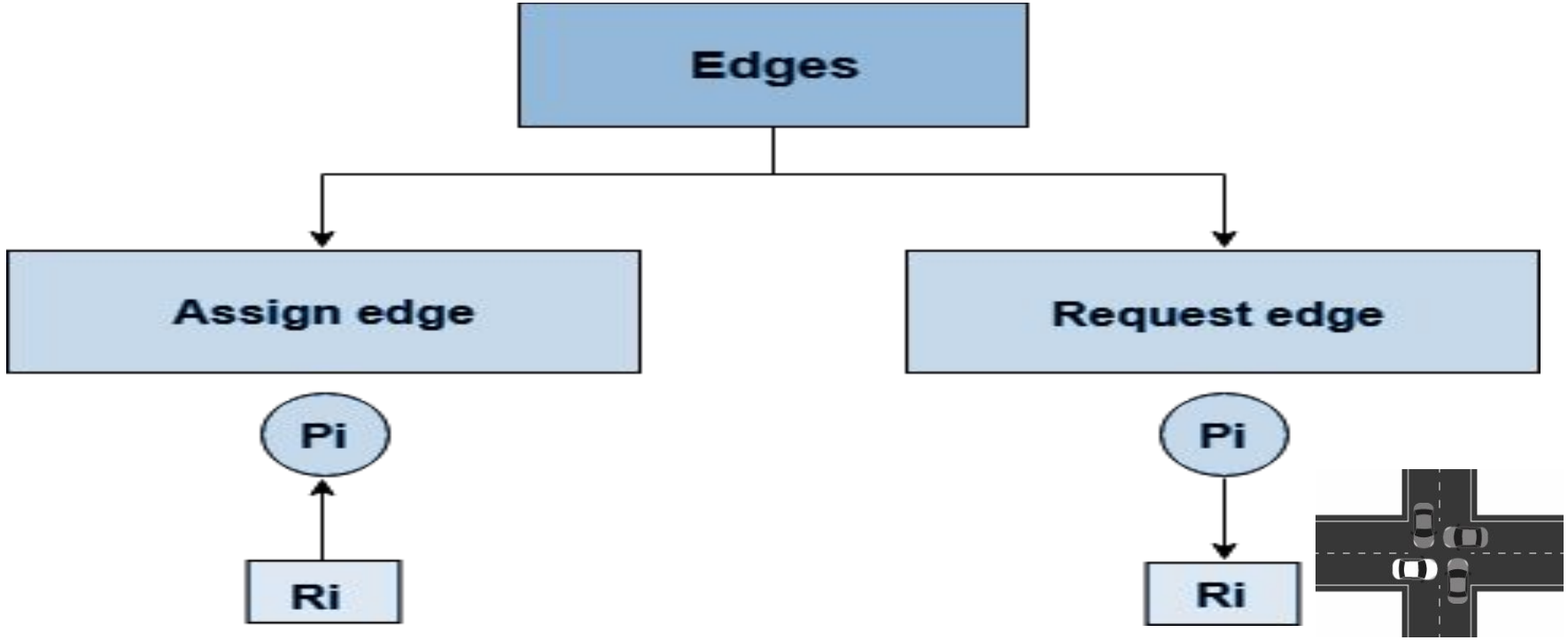
Deadlocks: Resource Allocation Graph - RAG

- Vertices are mainly of two types, Resource and process. Each of them will be represented by a different shape. Circle represents process while rectangle represents resource.
- A resource can have more than one instance. Each instance will be represented by a dot inside the rectangle.



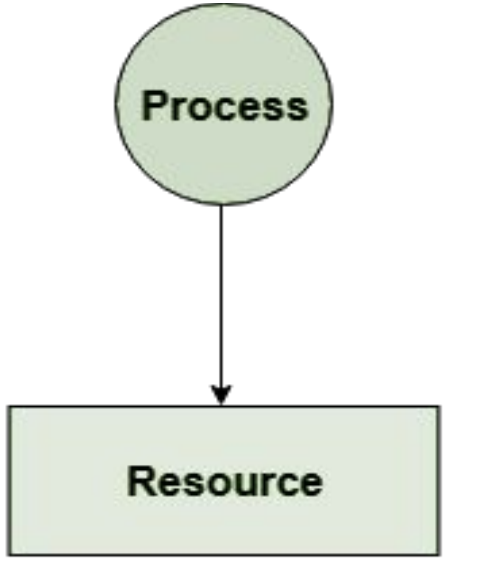
Deadlocks: Resource Allocation Graph - RAG

- Edges in RAG are also of two types, one represents assignment and other represents the wait of a process for a resource. The above image shows each of them.
- A resource is shown as assigned to a process if the tail of the arrow is attached to an instance to the resource and the head is attached to a process.
- A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.

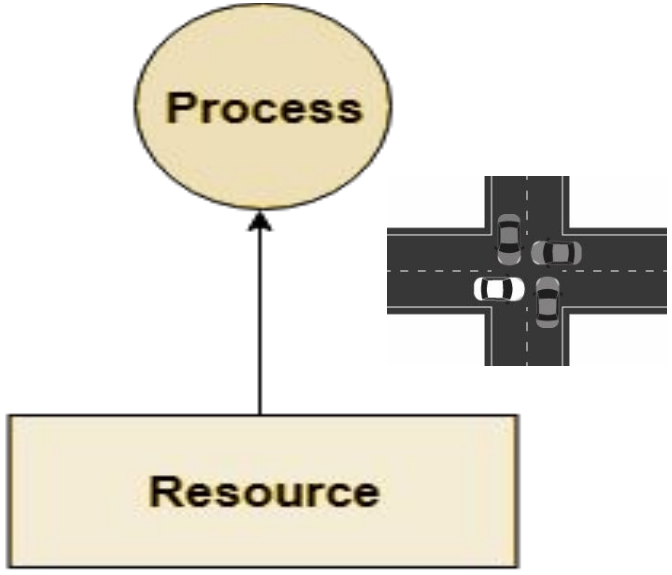


Deadlocks: Resource Allocation Graph - RAG

- Edges in RAG are also of two types, one represents assignment and other represents the wait of a process for a resource. The above image shows each of them.
- A resource is shown as assigned to a process if the tail of the arrow is attached to an instance to the resource and the head is attached to a process.
- A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.

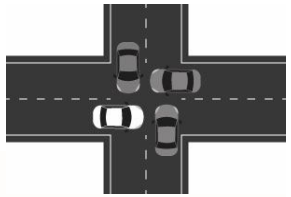


Process is requesting for a resource

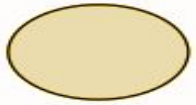


Resource is assigned to process

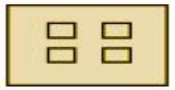
Deadlocks: Resource Allocation Graph - RAG



- RAG - Components
  - Process



  - Resource Type with 4 instances



  - P_i requests instance of R_j

  - P_i is holding an instance of R

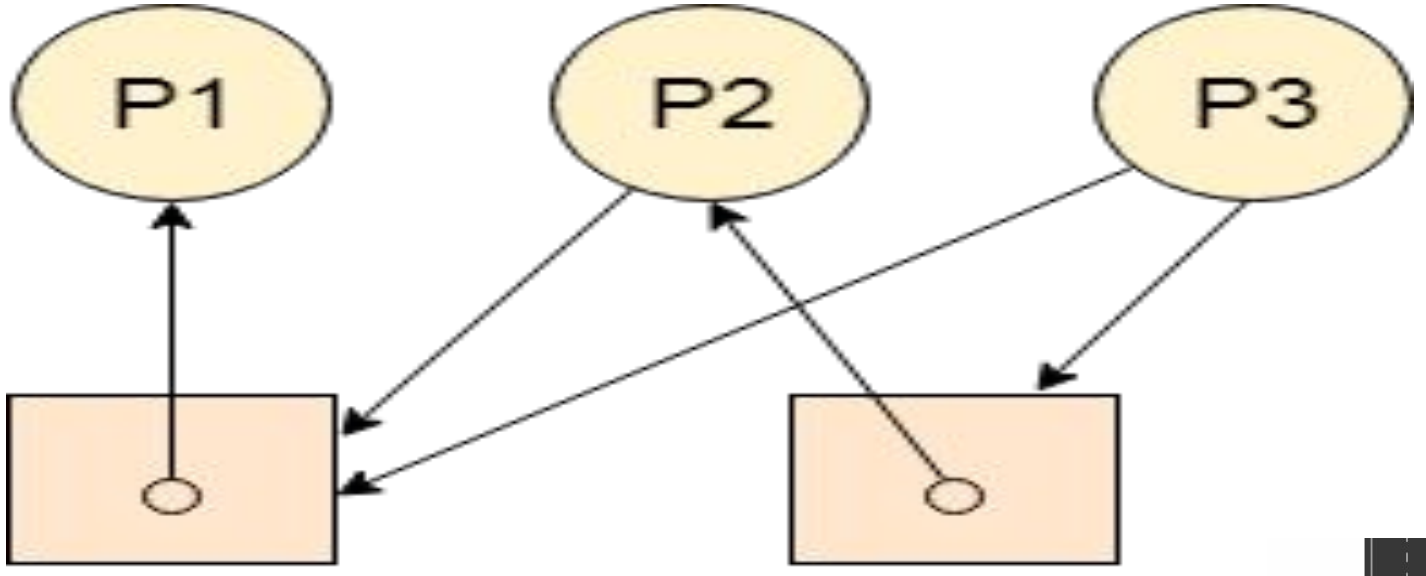
Deadlocks: Resource Allocation Graph - RAG

- RAG - 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

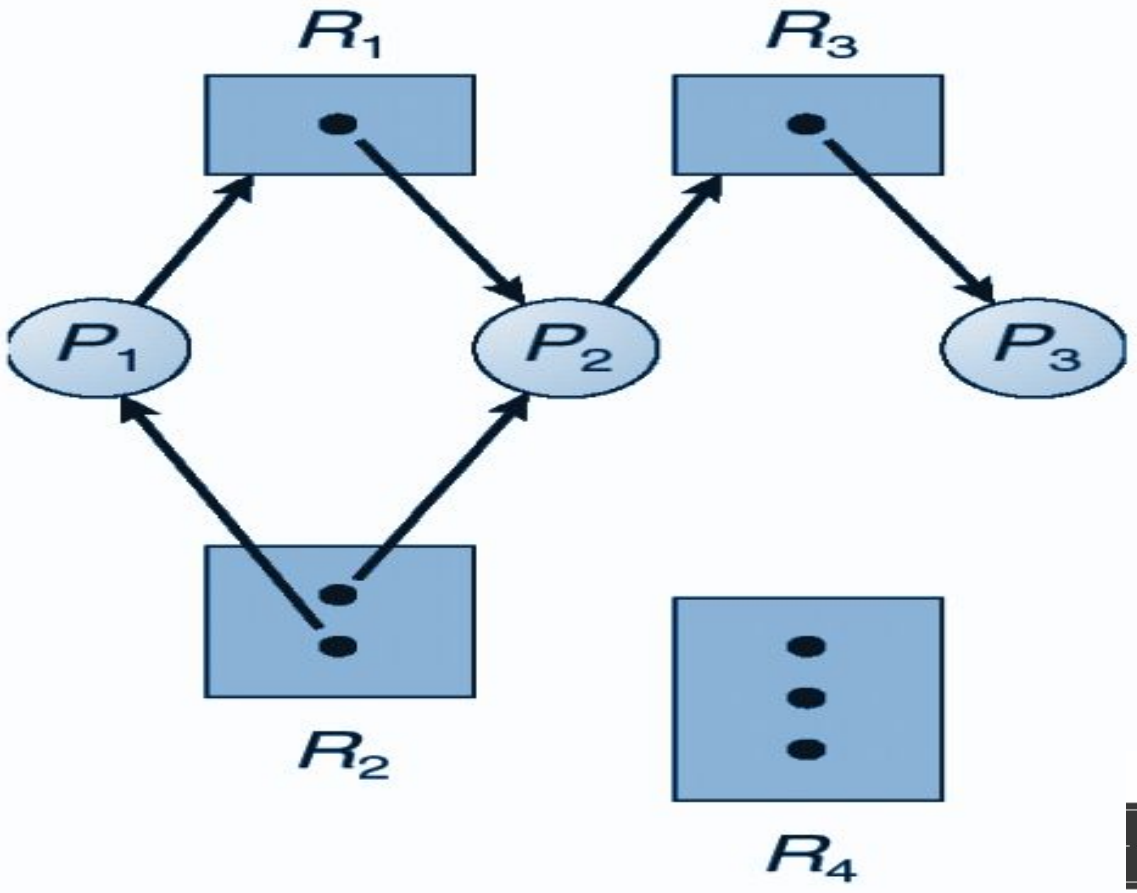
Deadlocks: Resource Allocation Graph - RAG

- Narrate and write the RAG story covering, the number of processes,resources and its instances, # of edges and types, #of cycles, Deadlock State =>{No, Yes, Possible}



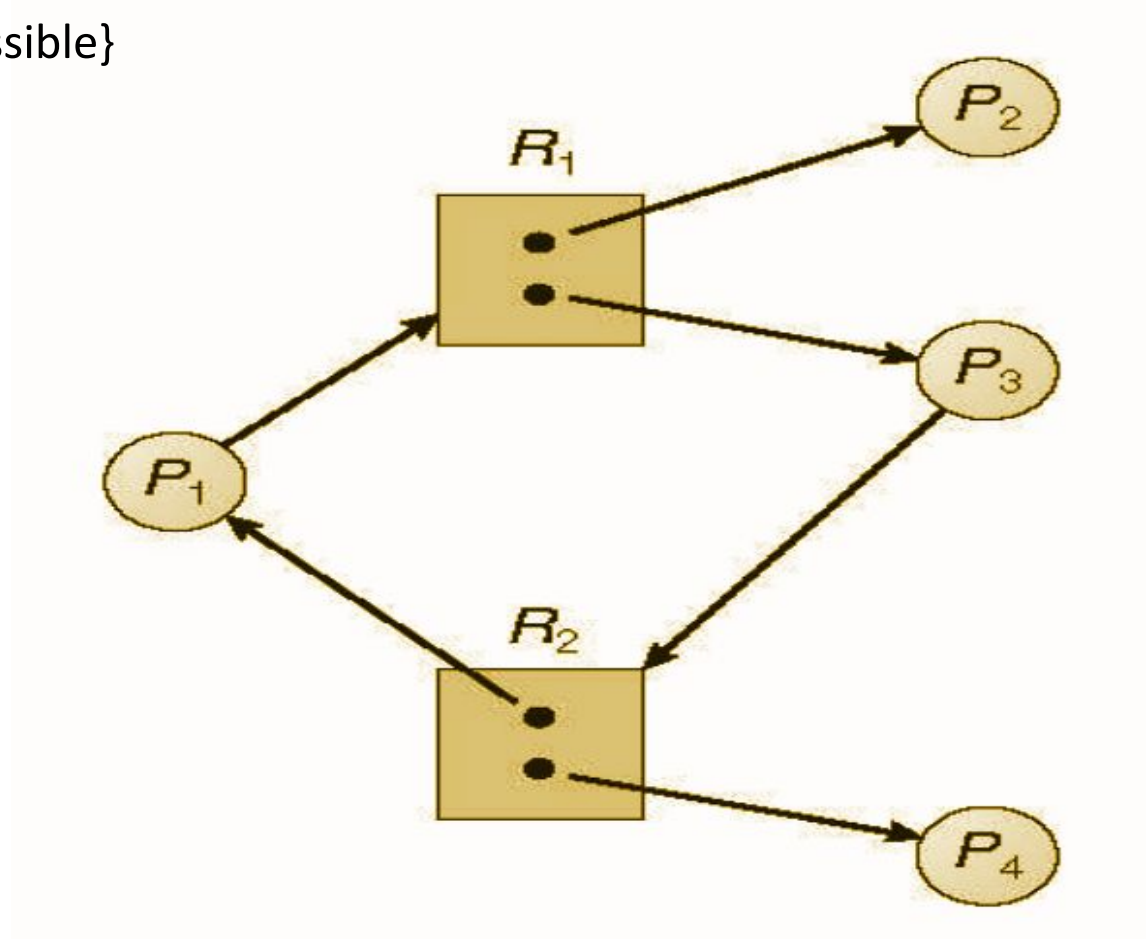
Deadlocks: Resource Allocation Graph - RAG

- Narrate and write the RAG story covering, the number of processes,resources and its instances, # of edges and types, #of cycles, Deadlock State =>{No, Yes, Possible}



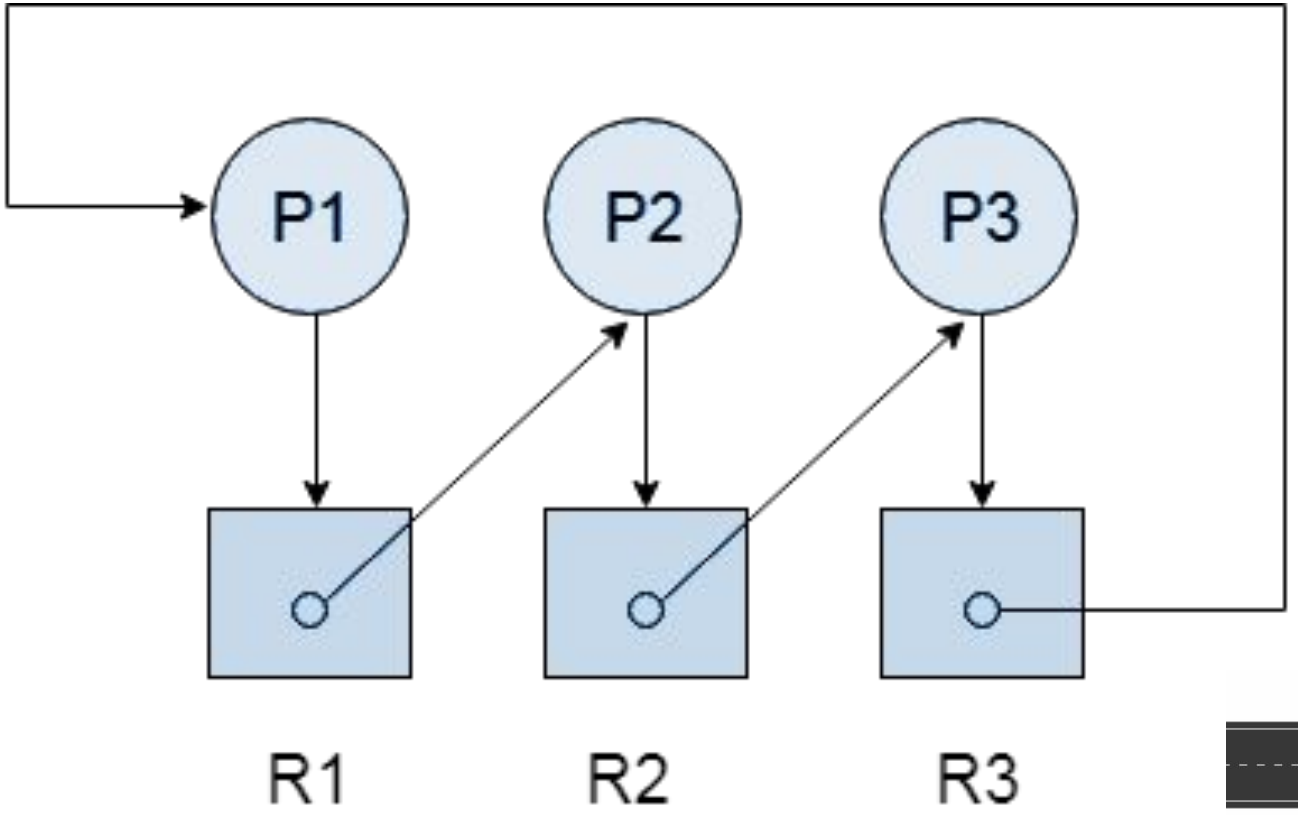
Deadlocks: Resource Allocation Graph - RAG

- Narrate and write the RAG story covering, the number of processes,resources and its instances, # of edges and types, #of cycles, Deadlock State =>{No, Yes, Possible}



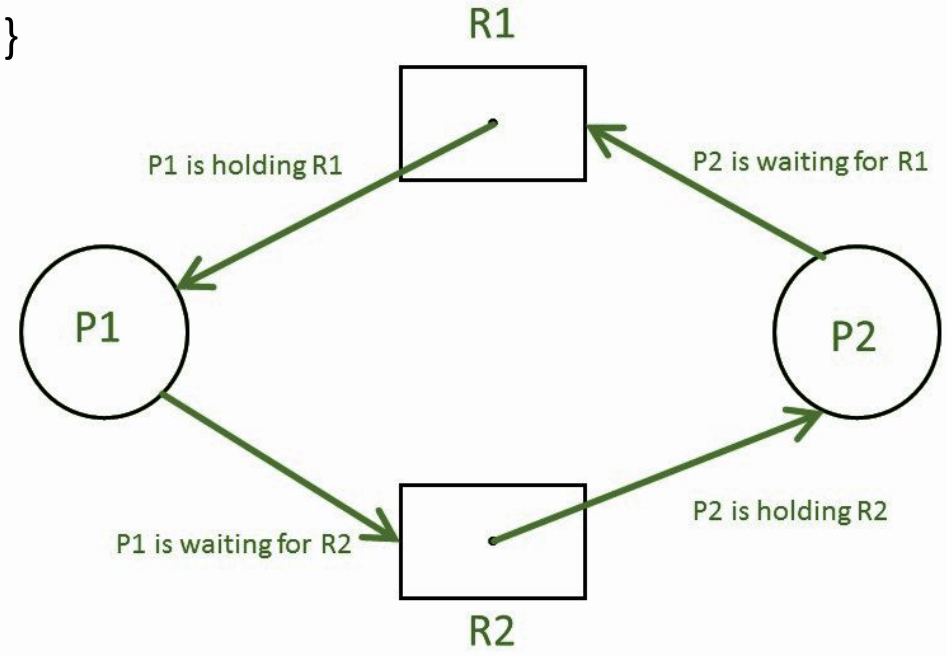
Deadlocks: Resource Allocation Graph - RAG

- Narrate and write the RAG story covering, the number of processes,resources and its instances, # of edges and types, #of cycles, Deadlock State =>{No, Yes, Possible}



Deadlocks: Resource Allocation Graph - RAG

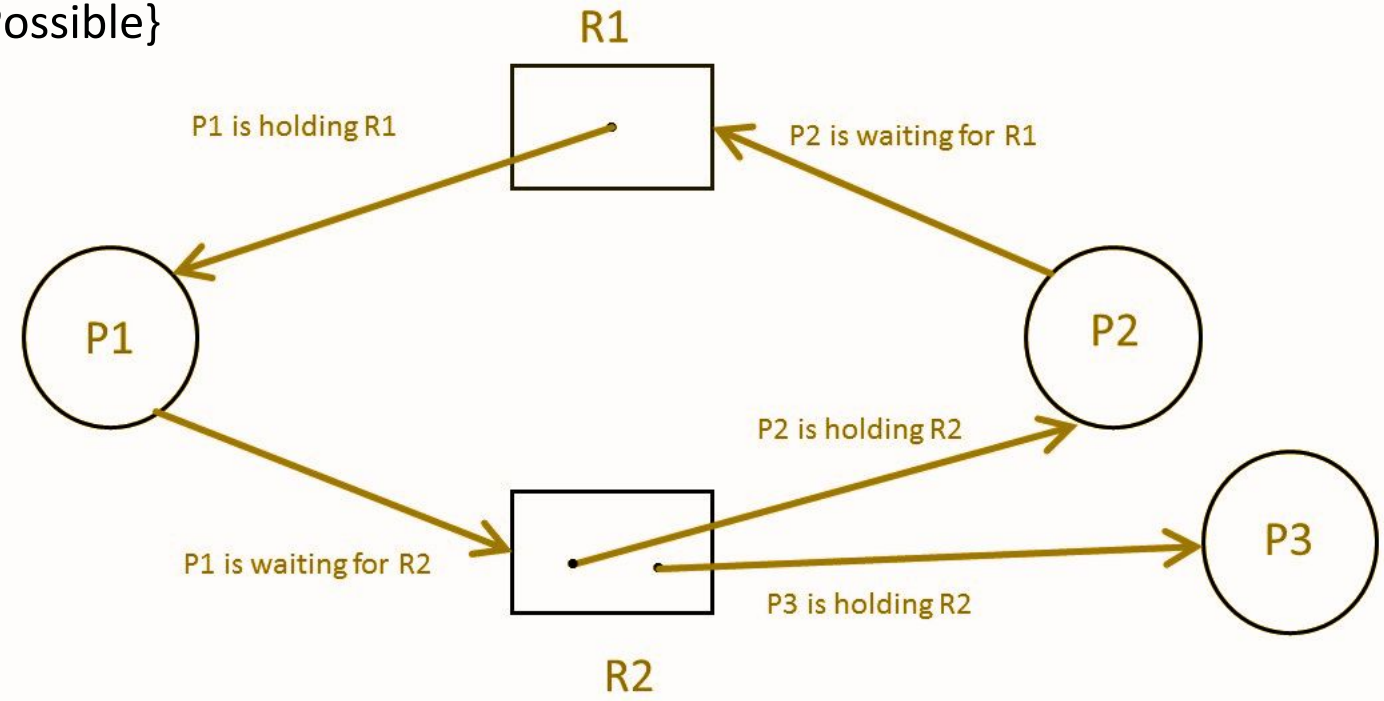
- Narrate and write the RAG story covering, the number of processes,resources and its instances, # of edges and types, #of cycles, Deadlock State =>{No, Yes, Possible}



SINGLE INSTANCE RESOURCE TYPE WITH DEADLOCK

Deadlocks: Resource Allocation Graph - RAG

- Narrate and write the RAG story covering, the number of processes,resources and its instances, # of edges and types, #of cycles, Deadlock State =>{No, Yes, Possible}



MULTI INSTANCES WITHOUT DEADLOCK

Deadlocks: Deadlock Prevention

- Havender in his pioneering work showed that since all four of the conditions are necessary for deadlock to occur, it follows that deadlock might be prevented by denying any one of the conditions.

Elimination of "Mutual Exclusion" Condition

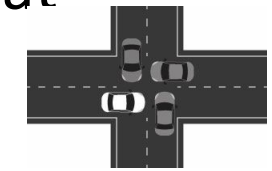
- The mutual exclusion condition must hold for non-sharable resources.
- That is, several processes cannot simultaneously share a single resource.
- This condition is difficult to eliminate because some resources, such as the tap drive and printer, are inherently non-shareable.
- Note that shareable resources like read-only-file do not require mutually exclusive access and thus cannot be involved in deadlock.

Deadlocks: Deadlock Prevention

Elimination of "Hold and Wait" Condition

- There are two possibilities for elimination of the second condition. The first alternative is that a process request be granted all of the resources it needs at once, prior to execution.

- The second alternative is to disallow a process from requesting resources whenever it has previously allocated resources. This strategy requires that all of the resources a process will need must be requested at once.



Deadlocks: Deadlock Prevention

Elimination of "Hold and Wait" Condition

- The system must grant resources on "all or none" basis. If the complete set of resources needed by a process is not currently available, then the process must wait until the complete set is available.

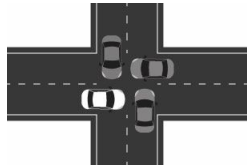
- While the process waits, however, it may not hold any resources. Thus the "wait for" condition is denied and deadlocks simply cannot occur. This strategy can lead to serious waste of resources

Deadlocks: Deadlock Prevention

Elimination of "Hold and Wait" Condition

- For example, a program requiring ten HDDs must request and receive all ten drives before it begins executing. If the program needs only one HDD to begin execution and then does not need the remaining HDD for several hours. Then substantial computer resources (9 HDDs) will sit idle for several hours.

- This strategy can cause indefinite postponement (starvation). Since not all the required resources may become available at once.



Deadlocks: Deadlock Prevention

Elimination of “No-preemption” Condition

- The non-preemption condition can be alleviated by forcing a process waiting for a resource that cannot immediately be allocated to relinquish all of its currently held resources, so that other processes may use them to finish. Suppose a system does allow processes to hold resources while requesting additional resources.

- Consider what happens when a request cannot be satisfied. A process holds resources a second process may need in order to proceed while second process may hold the resources needed by the first process. This is a deadlock. This strategy require that when a process that is holding some resources is denied a request for additional resources.

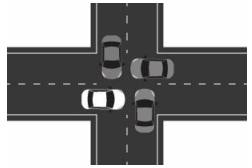
- The process must release its held resources and, if necessary, request them again together with additional resources. Implementation of this strategy denies the “no-preemptive” condition effectively.

Deadlocks: Deadlock Prevention

Elimination of "No-preemption" Condition: High Cost

- When a process release resources the process may lose all its work to that point. One serious consequence of this strategy is the possibility of indefinite postponement (starvation).

- A process might be held off indefinitely as it repeatedly requests and releases the same resources.



Deadlocks: Deadlock Prevention

Elimination of "Circular Wait" Condition

- The last condition, the circular wait, can be denied by imposing a total ordering on all of the resource types and than forcing, all processes to request the resources in order (increasing or decreasing).

- This strategy impose a total ordering of all resources types, and to require that each process requests resources in a numerical order (increasing or decreasing) of enumeration. With this rule, the resource allocation graph can never have a cycle.

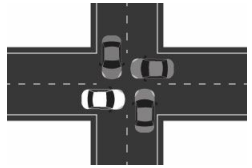
  For example, provide a global numbering of all the resources, as shown

  1 = Card reader

  2 = Printer

  3 = Plotter

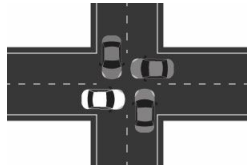
  4 = Tape drive



Deadlocks: Deadlock Prevention

Elimination of "Circular Wait" Condition

- Now the rule is this: Processes can request resources whenever they want to, but all requests must be made in numerical order. A process may request first printer and then a tape drive (order: 2, 4), but it may not request first a plotter and then a printer (order: 3, 2). The problem with this strategy is that it may be impossible to find an ordering that satisfies everyone.



THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com