Deadlock

Definition

- Deadlock is a situation which occurs when a process or thread enters a waiting state because a resource requested is being held by another waiting process, which in turn is waiting for another resource held by another waiting process. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock[1]

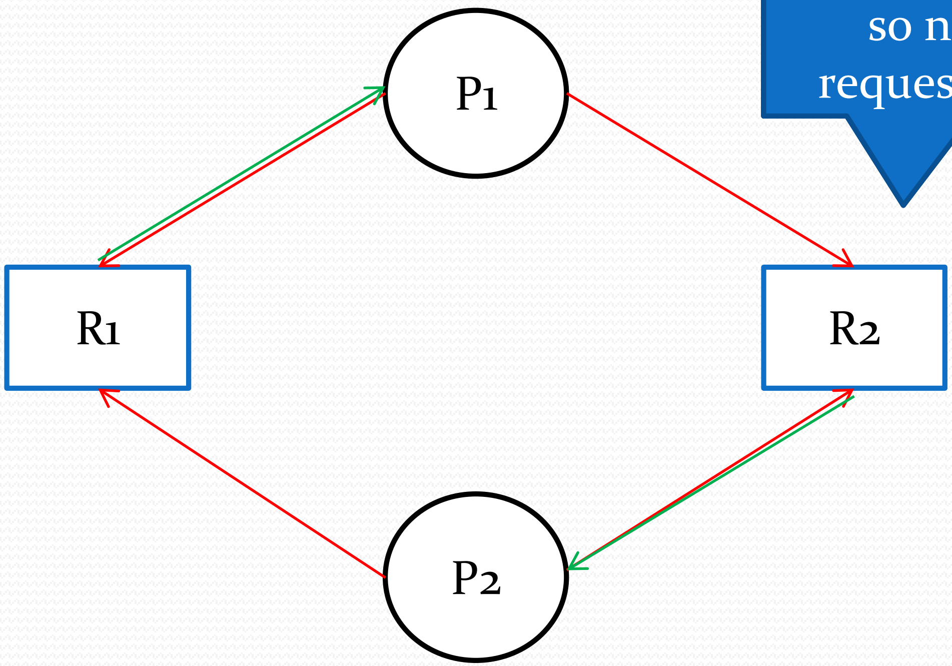
Silberschatz, Abraham (2006). Operating System Principles (7 ed.). Wiley-India. p. 237. ISBN 9788126509621

Example

- Two Process: P1 and P2
- Two Resource: R1 and R2

Both P1 and P2 will wait forever and the system is said to be in "Deadlock" state

Since R1 is held by P1 and R2 is held by P2, so none of the requests is fulfilled



Necessary conditions

- Mutual Exclusion
  - Only one process can use a resource at any given time
- Hold and wait
  - A process is holding a resource and waiting for atleast one another resource which is held by some other process.
- No preemption
  - Resource can not be taken back from a process until it has finished its task.
- Circular wait
  - A set {P_0, P_1,..P_n} of waiting processes must exist such that P_0 is waiting for a resource held by P_1, P_1 waiting for resource held by P_2, ...P_{n-1} is waiting for resource held by P_n and P_n waiting for a resource held by P_0

Resource Allocation Graph

- Directed graph
- Vertices – V
- Edges – E
- V is divided into

  processes P ={P_1, P_2, …, P_n}

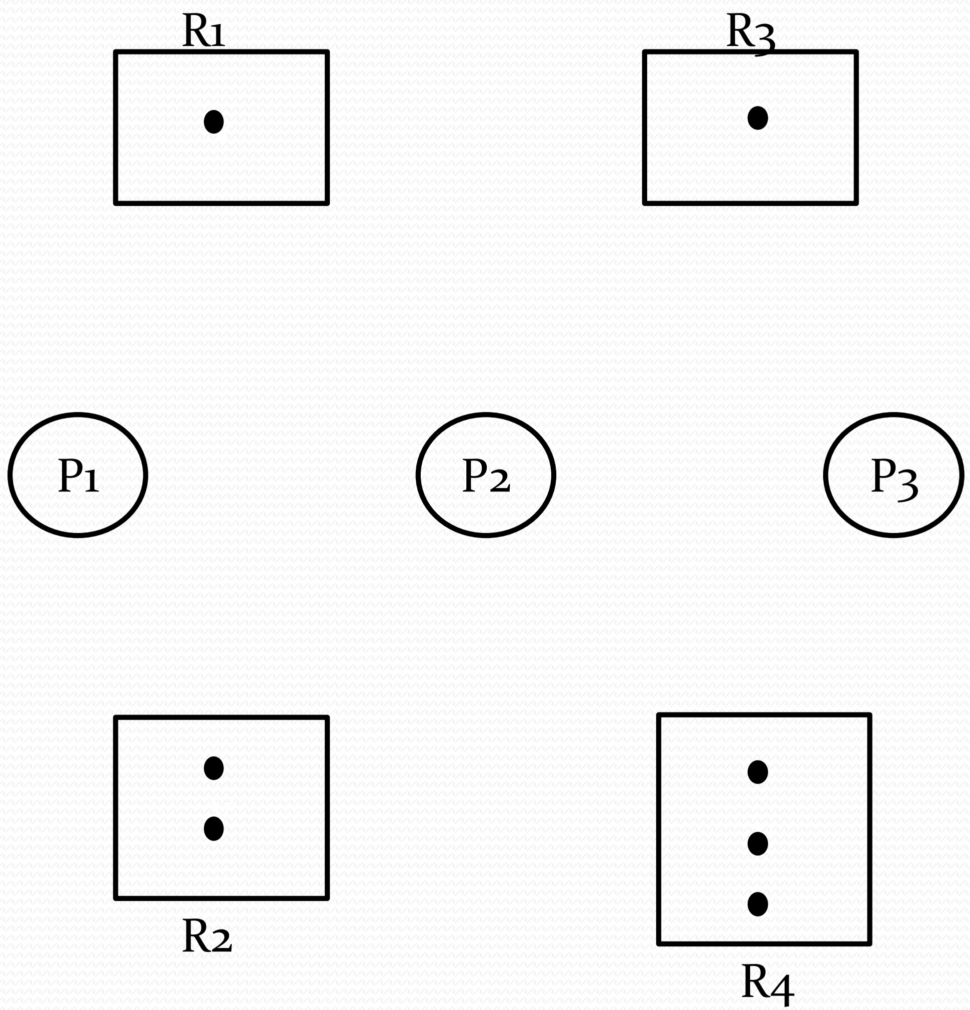
  resources R = {R_1, R_2, …, R_n}

- E

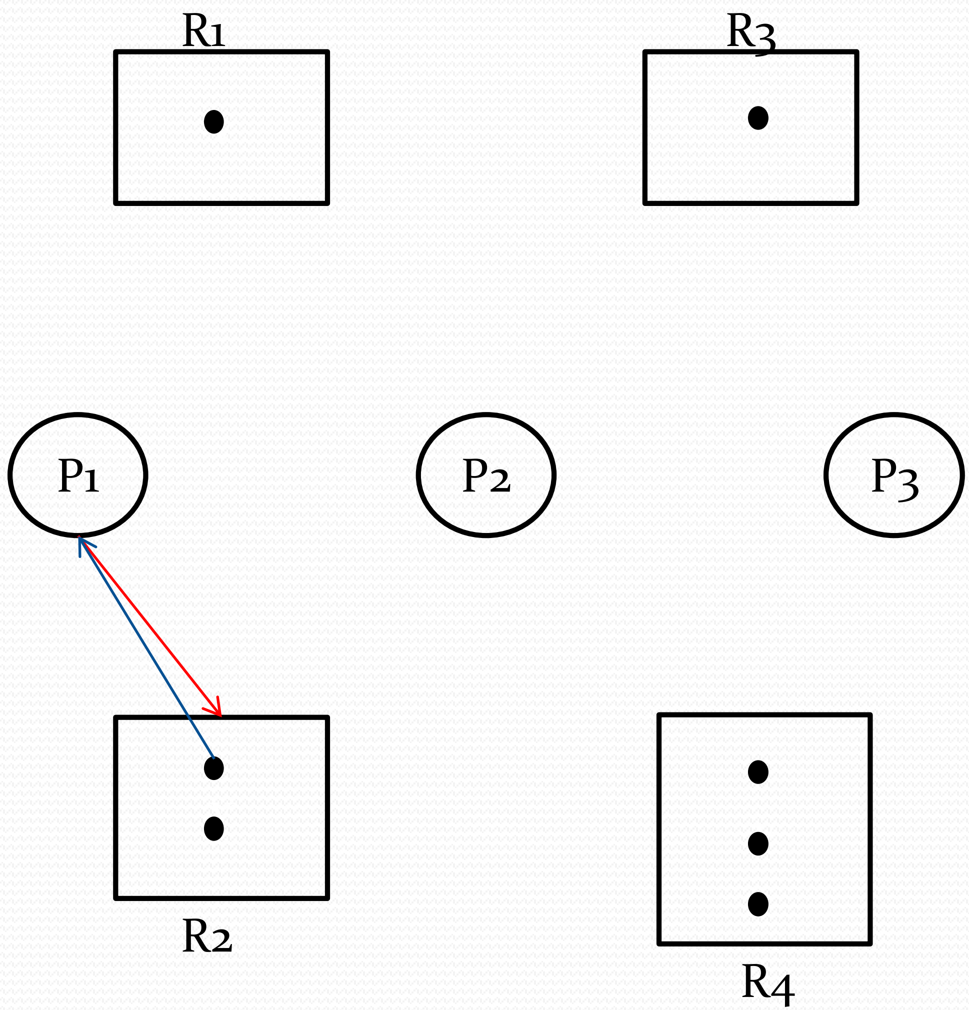
  Request Edge – from process to resource

  Assignment edge – from resource to process

- Process is represented by
- Resource is represented by
- Instance of a resource is represented by dot(•)

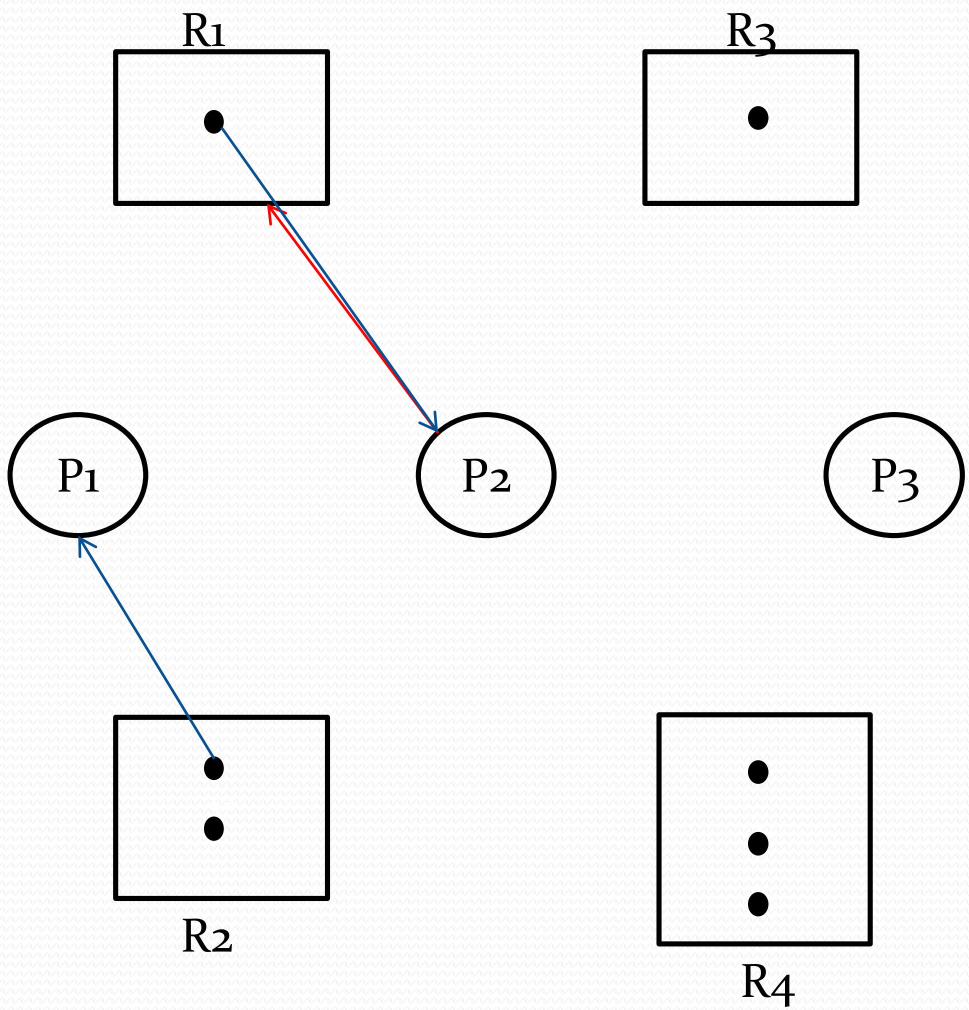


Initially No resource is allocated to any process

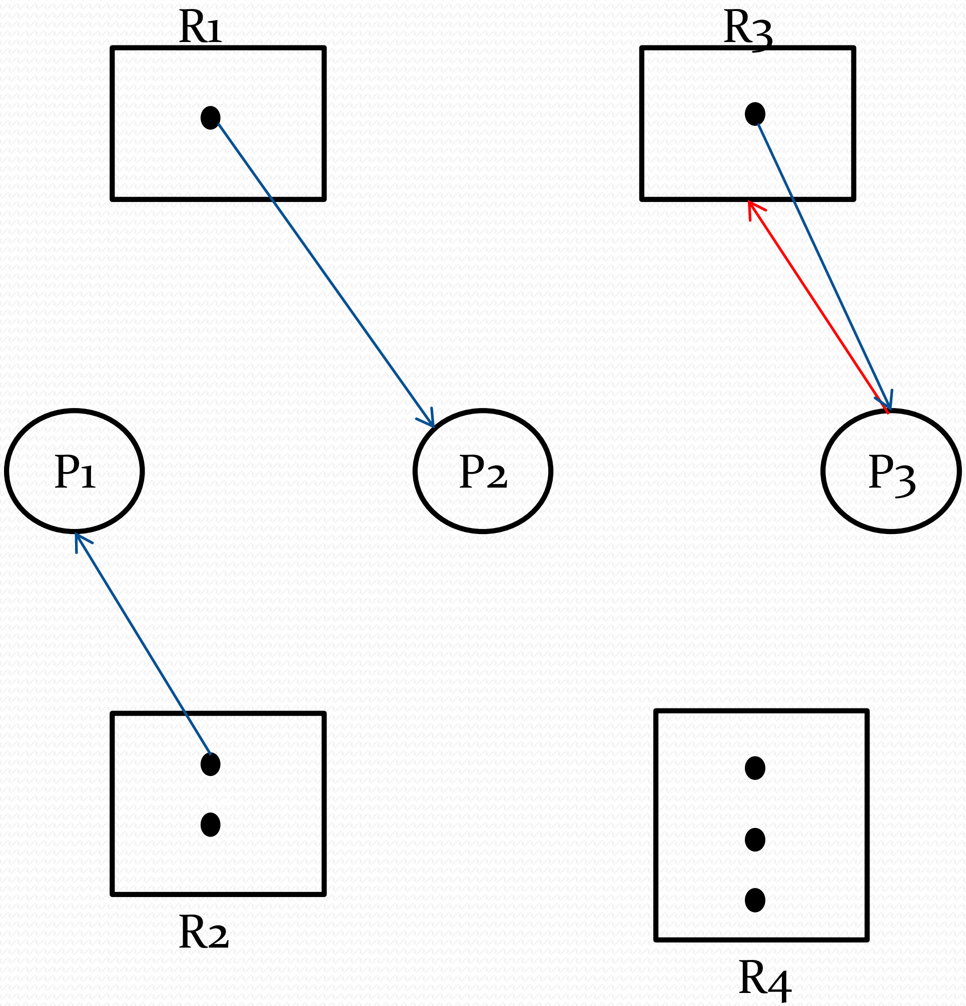


P1 requests R2

One instance of R2 is allocated to P1

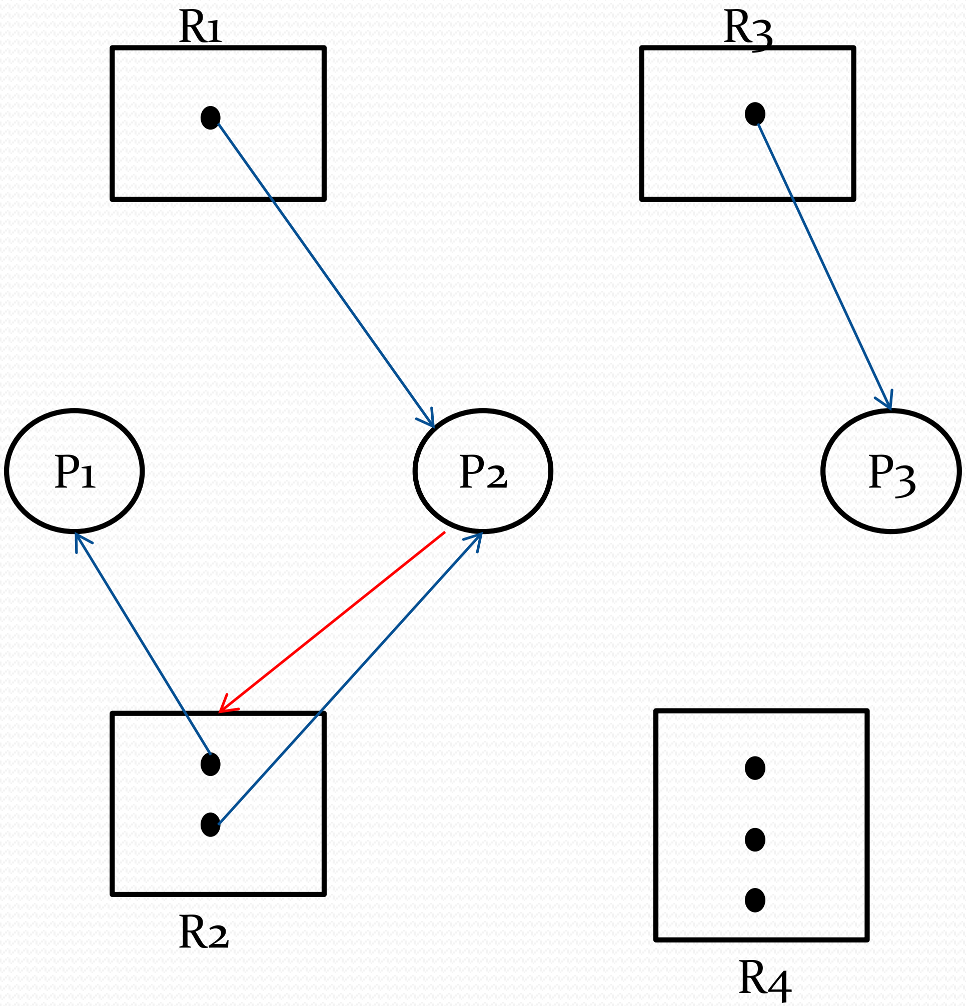


P2 requests R1
R2 is allocated to P2



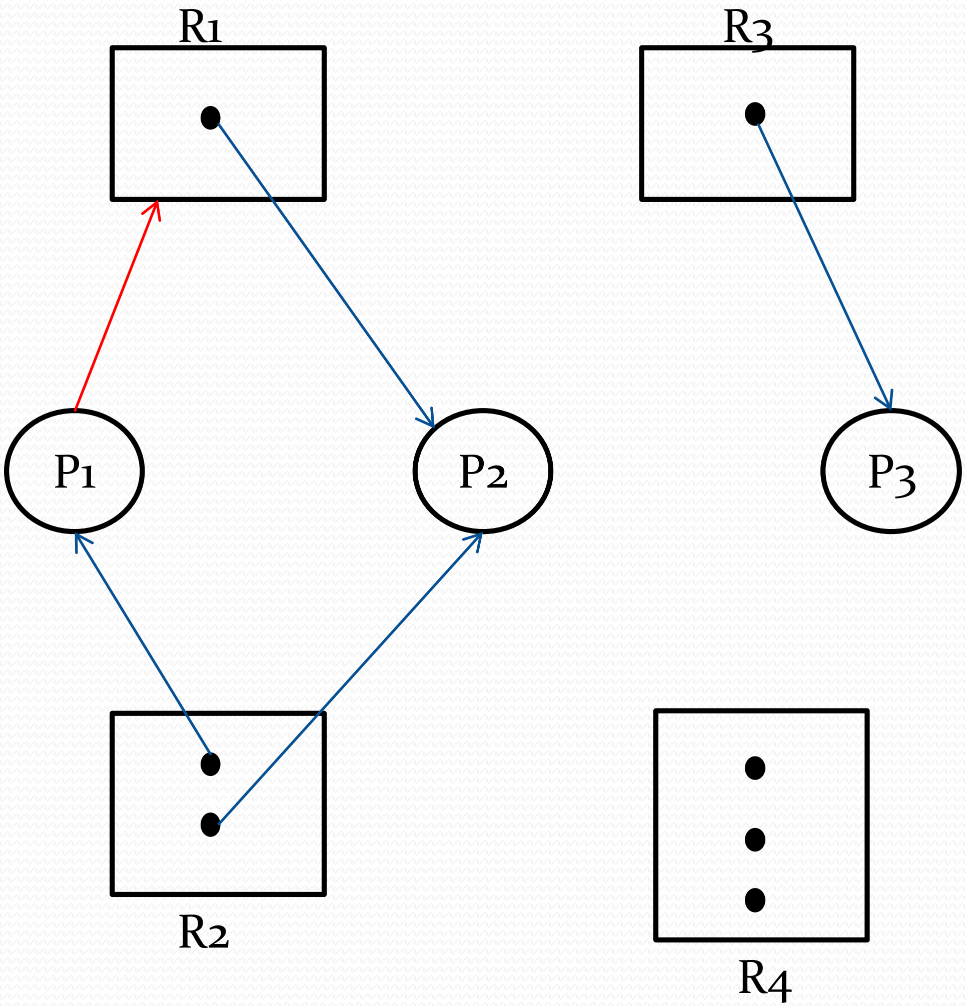
P3 requests R3

R3 is allocated to P3



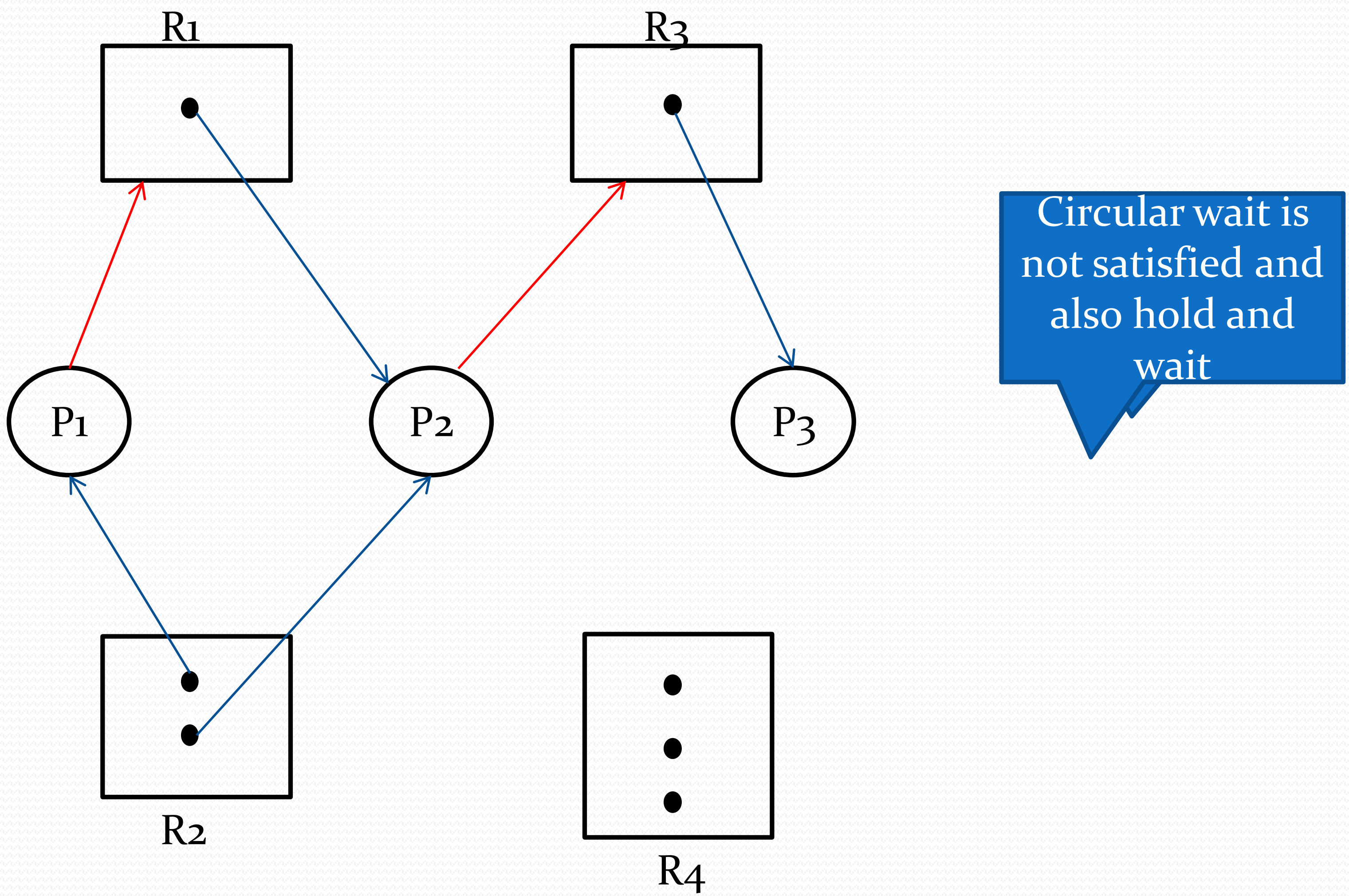
P2 requests R2

One instance of R2 is allocated to P2



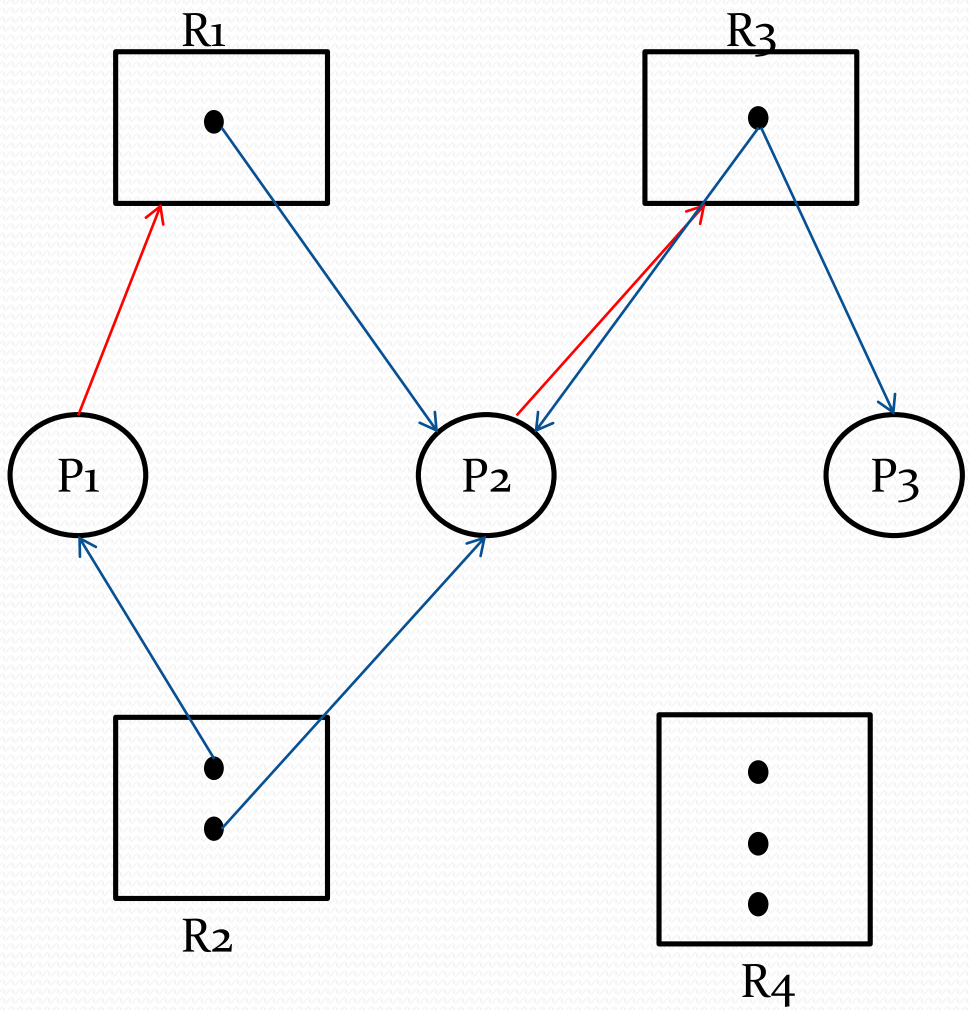
No. Circular wait is not satisfied

P1 requests R1
Request Can not be fulfilled as R1 is being used by P2



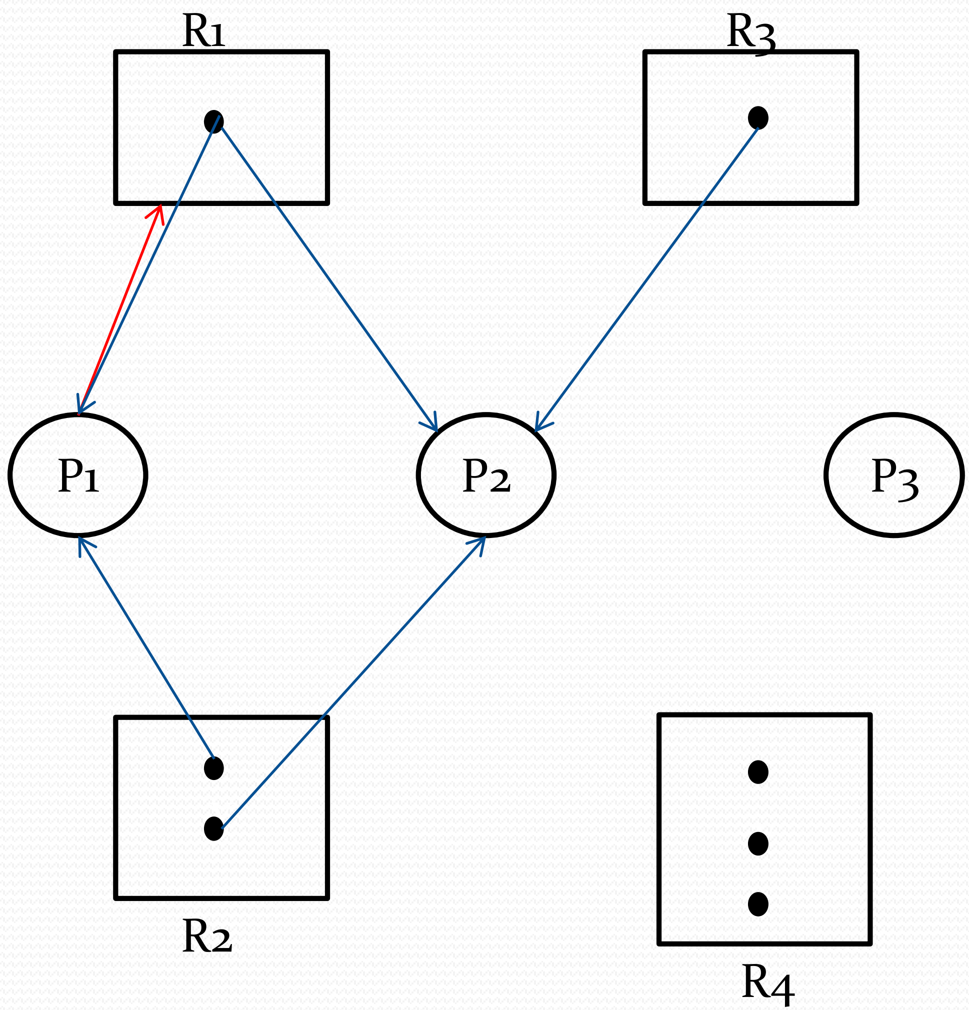
P2 requests R3

Request Can not be fulfilled as R3 is being used by P3



Still no deadlock

Since P3 after finishing can release R3, which can be then allocated to P2.

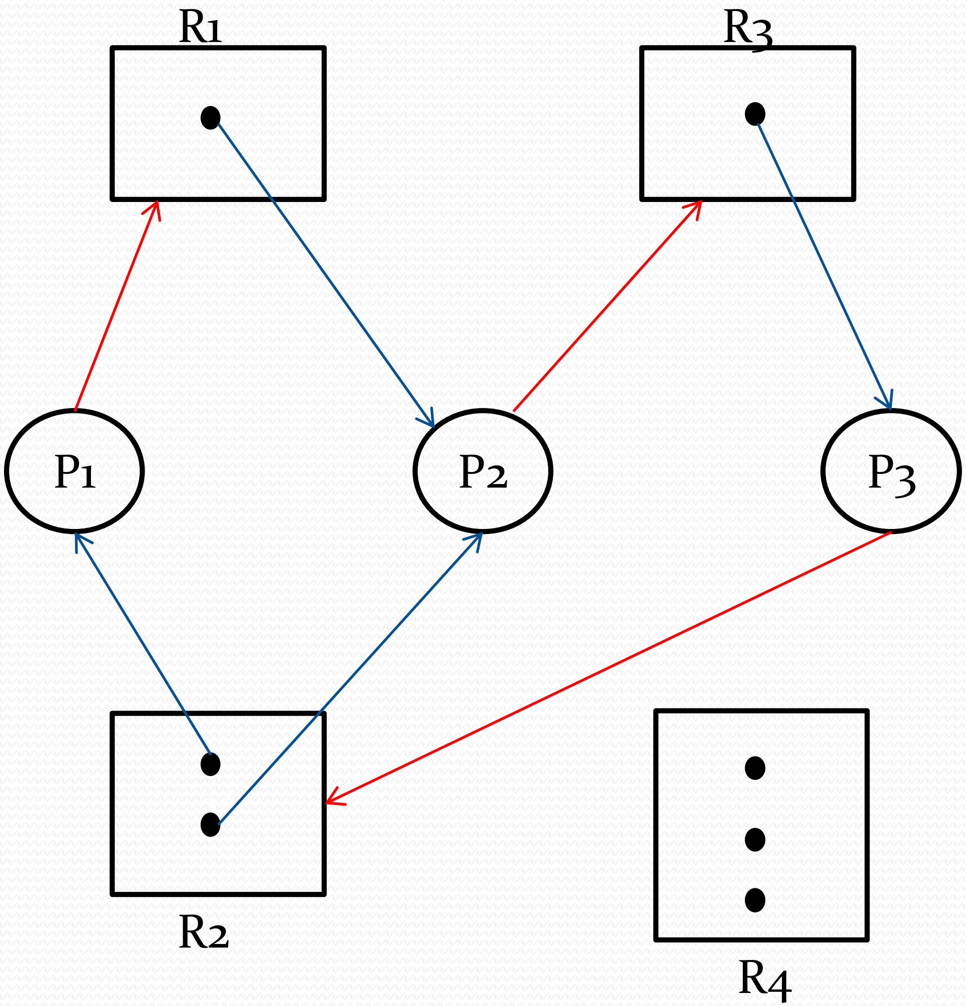


Still no deadlock

Since P3 after finishing can release R3, which can be then allocated to P2.

P2 will have all the required resources. It can finish and then release R1, which can be allocated to P1

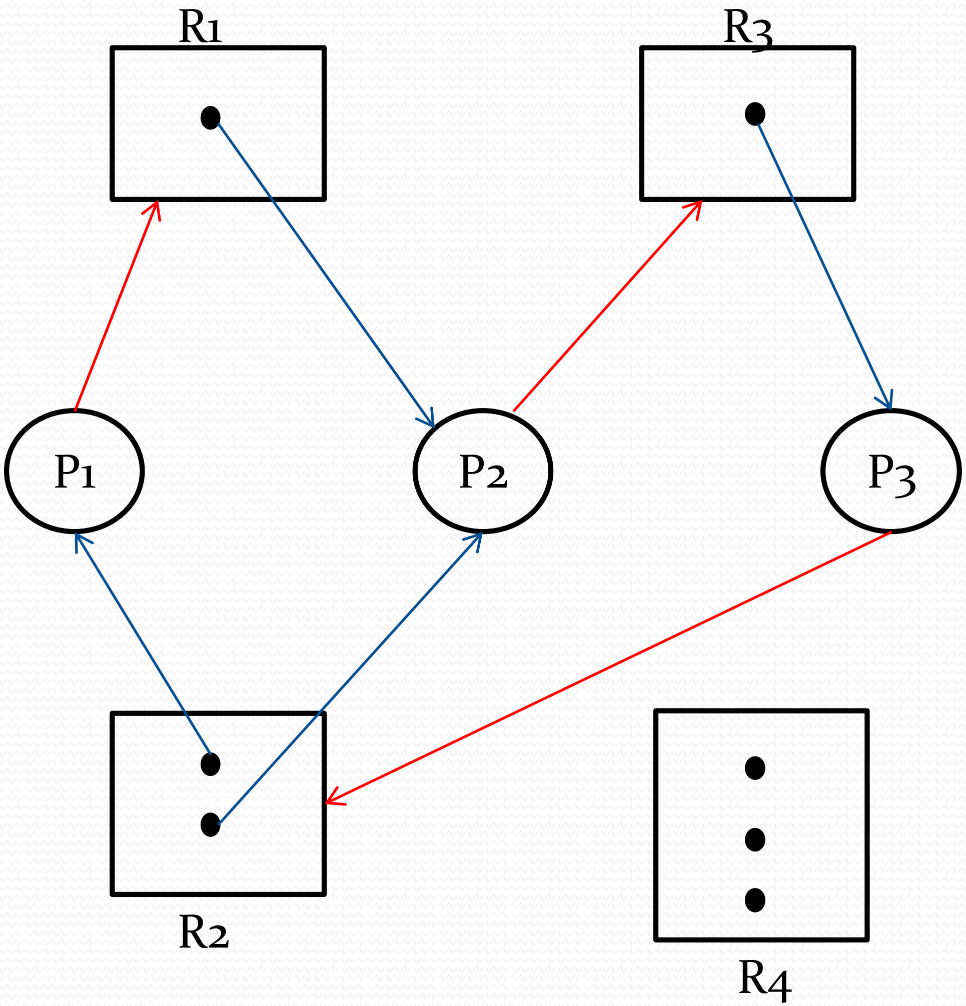
Finally P1 will have all the required resources.



Will there be a deadlock situation?

YES

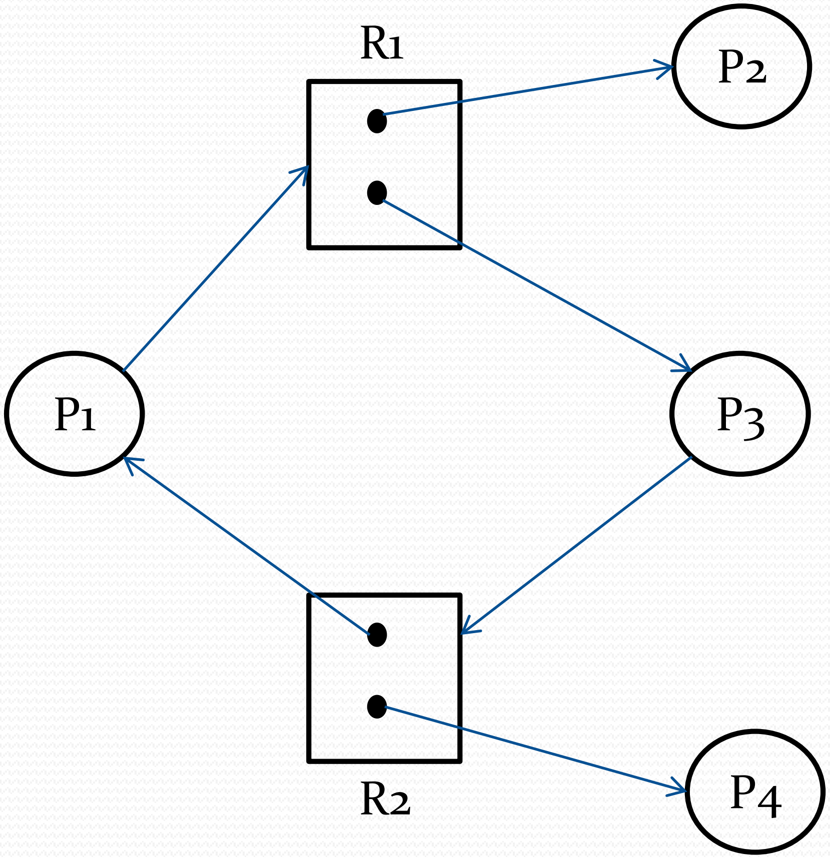
Now suppose P3 requests R2
Request Can not be fulfilled as no instance of R2 is free



Cycle 1 : P1 -> R1 -> P2 - > R3 -> P3 -> R2 -> P1

Cycle 2 : P2 - > R3 -> P3 -> R2 -> P2

Is there a deadlock?



No, becau e P2

Wait.

NO

MCQs

- What is the minimum number of process required for a deadlock to occur?

a) 1
b) 2
c) 3
d) 4

- What is the number of necessary conditions for a deadlock to occur?

a) 1
b) 2
c) 3
d) 4

MCQs

- Which of the following is not a necessary condition for deadlock to occur?
  1. Mutual exclusion
  2. Hold and wait
  3. Preemption
  4. Circular wait
- In a resource allocation graph the Processes are represented by
  1. Circle
  2. Rectangle
  3. Dot
  4. Square

Deadlock Prevention and Avoidance

Methods for handling deadlocks

- Prevent or avoid deadlock from occurring by using some protocol.
- Allow the system to enter deadlock, detect it and recover from it.
- Ignore the problem and pretend it never occurred.

Deadlock Prevention

- Prevent at least one of the necessary conditions from occurring.
- Mutual exclusion
  - Can not be denied since its an intrinsic property of the resource. E.g. printer is nonsharable
- Hold and wait
  - Whenever a process request a resource, it must not be holding any resource.
    1. Every process must be allocated all the resources before it begins execution.
    2. Whenever a process request for resources, it must release all the currently held resources (if any).
  - Disadvantage
    1. Low resource utilization
    2. Starvation is possible

Deadlock Preemption

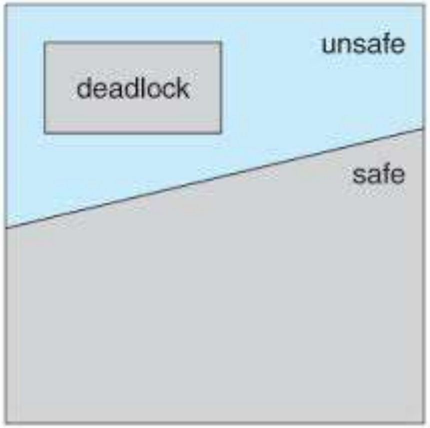
- No preemption
  - If the requested resources can not be allocated then the resources that are currently held by the process are preempted.
  - Can be applied to resources like CPU registers and memory space but not to printers and tape drives.
- Circular wait
  - Process can request for resources in some order only.
  - E.g. Suppose each process is numbered like Po, P1, p2,. .., Pn. Then they can request for resources held by higher numbered process only i.e. Po can request from P1, P2 and so on, p1 from P2, P3 and so on but not Po, P2 from P3, P4 and so on but not Po and P1.

Deadlock Avoidance

- Provide information to OS that which all resources any process will require during its lifetime.
- Then at every request the system can decide whether to grant that request or not.
- This decision depends upon
  - resources currently available
  - resources currently allocated to each process
  - future request and release of resources by each process

Deadlock Avoidance

- Safe State – state is safe if the system can allocate resources to every process in some order and still avoid deadlock.
- Safe sequence – is an order {P1, P2, ... Pn} in which the processes can be allocated resources safely.
- Safe state – No deadlock
- Unsafe state – Deadlock **may** occur



Example

Q. System has total 12 magnetic tapes and 3 processes. Individual process requirement is as below

Process	Max. need	Current Need
P0	10	5
P1	4	2
P2	9	2

Is the system in safe state at time T0 ?
Yes. Safe sequence is <P1,P0,P2>
Magnetic tapes remaining with system = 12-(5+2+2) = 3
P1 can be allocated its remaining two magnetic tapes.
After P1 finishes Magnetic tapes remaining with system = 5
P0 needs 5 more magnetic tapes which can be allocated. After P0 finishes P2's need of remaining 7 tapes can be fulfilled.

- Suppose P2 requests for 1 more magnetic tape at time T1. Should this request be granted?
- No
- Reason: Let us suppose the request is granted. The modified system state will be

Process	Max. need	Current Need
P0	10	5
P1	4	2
P2	9	3

- No safe sequence exists.
  - Remaining tapes with the system = 2
  - Request of P1 can be fulfilled. After P1 finishes tapes available = 4
  - P0 needs 5 and P2 needs 6 tapes. Both will keep on waiting, resulting in a deadlock

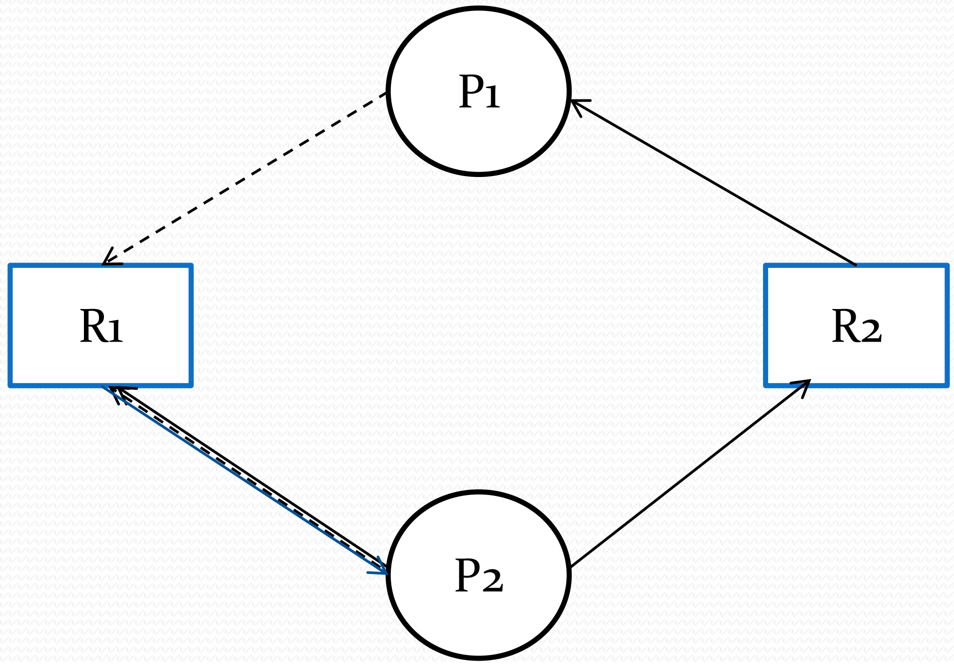
Deadlock Avoidance

- Conclusion
  - Do not grant the request immediately
  - Check whether granting the request will leave the system in safe state or not?
  - If yes, grant the request
  - If no, do not grant the request
- Disadvantage
  - Low resource utilization
- Algorithms for deadlock avoidance
  - Resource allocation graph algorithm
  - Banker's algorithm

Resource allocation graph algo.

- Applicable if each resource has only one instance.
- Claim edge : -------> from process to resource
  - Process may request resource in future
- When process requests resource change claim edge to request edge.
- Request edge is converted into assignment edge only if the conversion does not lead to the formation of a cycle in the graph.

Example



- Suppose P2 requests for R1. Can this request be granted?
- No, because granting the request will lead to the formation of cycle in the graph

Banker's Algorithm

Banker's algorithm

- Can be used when multiple instances exists.
- Every process
  - Tells the max. no. of instances of any resource it requires
  - The maximum need can not exceed the total number of resources in the system.
  - On every request the system determines whether granting that request will leave the system in safe state or not.
- Data structures required
  - Available – no. of available resources of each type
  - Max – maximum need of any process for any resource
  - Allocation – number of resources allocated to each process
  - Need – (Max – Allocation)

- Let **n** be the number of processes in the system and **m** be the number of resource types.
- Data structures required are:
- **Available**: A vector of length m indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type R_j available.
- **Max**: An *n×m* matrix defines the maximum demand of each process. If Max[i,j] = k, then P_i may request at most k instances of resource type R_j.
- **Allocation**: An *n×m* matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j] = k, then process P_i is currently allocated k instances of resource type R_j.
- **Need**: An *n×m* matrix indicates the remaining resource need of each process. If Need[i,j] = k, then P_i may need k more instances of resource type R_j to complete the task.
- Note: Need[i,j] = Max[i,j] - Allocation[i,j].

Safety algorithm

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively.
   Initialize:
   Work = *Available*
   Finish [*i*] = *false* for *i* =0,1,2,3, …, *n*.
2. Find an *i* such that both:
   (a) *Finish* [*i*] = *false*
   (b) $Need_i$ <= *Work*

If no such *i* exists, go to step 4.

3. *Work* = *Work* + $Allocation_i$
   Finish[*i*] = *true*
   go to step 2.
4. If *Finish* [*i*] == true for all *i*, then the system is in a safe state.

Banker's algorithm

Consider a system with five processes *Po* through *P4* and three resource types *A*, *B*, and C. Resource type *A* has 10 instances, resource type *B* has 5 instances, and resource type C has 7 instances. Suppose that, at time *To* , the following snapshot of the system has been taken:

Process	**Allocation**			**Max**			**Available**		
	A	B	C	A	B	C	A	B	C
Po	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
p4	0	0	2	4	3	3			

Step 1: Calculate Need : Max – Allocation

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
p4	0	0	2	4	3	3				4	3	1

Step 2: Find out safe sequence

<P1,P3,P4,P2,P0>

How to find out safe sequence?

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
Po	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
p4	0	0	2	4	3	3				4	3	1

- Can Need of Po be satisfied?
- No
- Can Need of P1 be satisfied?
- Yes
- After P1 finishes, updated Available

Available = Available + Allocation

Safe Sequence:
<P1

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	5	3	2	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
p4	0	0	2	4	3	3				4	3	1

Safe Sequence:
<P1,P3

- Can Need of P2 be satisfied?
- No
- Can Need of P3 be satisfied?
- Yes
- After P3 finishes, updated Available

  Available = Available + Allocation

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
Po	0	1	0	7	5	3	7	4	3	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
p4	0	0	2	4	3	3				4	3	1

Safe Sequence:
<P1,P3,P4

- Can Need of P4 be satisfied?
- Yes
- After P4 finishes, updated Available

  Available = Available + Allocation

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
Po	0	1	0	7	5	3	7	4	5	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
p4	0	0	2	4	3	3				4	3	1

Safe Sequence:
<P1,P3,P4 ,Po

- Can Need of Po be satisfied?
- Yes
- After Po finishes, updated Available

Available = Available + Allocation

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
Po	0	1	0	7	5	3	7	5	5	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
p4	0	0	2	4	3	3				4	3	1

Safe Sequence:
<P1,P3,P4 ,Po ,P2

- Can Need of P2 be satisfied?
- Yes
- After Po finishes, updated Available

Available = Available + Allocation

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	10	5	7	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
p4	0	0	2	4	3	3				4	3	1

Safe Sequence:
<P1,P3,P4 ,P0 ,P2>

Resource request algorithm

Request = request vector for process *Pi*. If $Request_i$ [*j*] = *k* then process *Pi* wants *k* instances of resource type R_j.

1. If $Request_i$ <= $Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.

2. If $Request_i$ <= *Available*, go to step 3. Otherwise *Pi* must wait, since resources are not available.

3. Pretend to allocate requested resources to *Pi* by modifying the state as follows:

$Available = Available = Request_i;$

$Allocation_i = Allocation_i + Request_i;$

$Need_i = Need_i - Request_i;;$

- *If safe => the resources are allocated to Pi.*
- *If unsafe =>* Pi *must wait, and the old resource-allocation state is restored*

Problem

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
p4	0	0	2	4	3	3			

- Suppose now that process P1 requests one additional instance of resource type *A* and two instances of resource type C,
- Can this request be granted?
- *Request1* = (1,0,2).
- *Is Request1 < Available*
- *i.e.*, (1,0,2) < (3,3,2), which is true.

- Pretend that request is granted

Process	**Allocation**			**Max**			**Need**			**Available**		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	7	4	3	**2**	**3**	**0**
P1	**3**	**0**	**2**	3	2	2	**0**	**2**	**0**			
P2	3	0	2	9	0	2	6	0	0			
P3	2	1	1	2	2	2	0	1	1			
p4	0	0	2	4	3	3	4	3	1			

- Find out safe sequence
- <P1, P3, P4, P0, P2>
- Since safe sequence exists, the request can be granted immediately.

Problem 2

Process	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
Po	0	1	0	7	5	3	7	4	3	**2**	**3**	**0**
P1	**3**	**0**	**2**	3	2	2	**0**	**2**	**0**			
P2	3	0	2	9	0	2	6	0	0			
P3	2	1	1	2	2	2	0	1	1			
p4	0	0	2	4	3	3	4	3	1			

- Can a request for (3,3,0) by P4 be granted?
- No, resources are not available
- Can a request for (0,2,0) by Po be granted?
- No, because granting the request will change the state to unsafe (no safe sequence exists)

MCQs

Q. Resource allocation graph can be used to avoid deadlock if there is/are ______ number of instances of any resource

a) 1
b) 2
c) 3
d) Any

Q. The data structure that is not required in Banker's algorithm is?

a) Need
b) Max
c) Resource
d) Allocation

- A state is safe, if :
  a) the system does not crash due to deadlock occurrence
  b) the system can allocate resources to each process in some order and still avoid a deadlock
  c) the state keeps the system protected and safe
  d) All of these
- A system is in a safe state only if there exists a :
  a) safe allocation
  b) safe resource
  c) safe sequence
  d) All of these

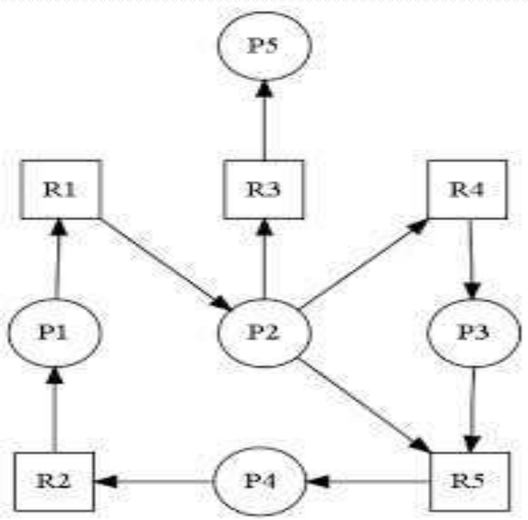
Deadlock detection and recovery

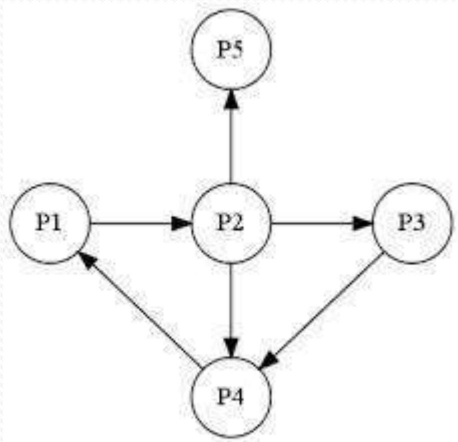
Deadlock detection

- Use an algorithm to detect deadlock (if any)
- Recover from the deadlock
- Single instance of each resource – *wait-for* graph
- Multiple instance of a resource

Single instance- Wait-for graph

- Obtained from resource allocation graph
- Remove the resource nodes
- Collapse the corresponding edges





Wait-for graph

- Edge from Pi to Pj
  - Pi is waiting for resource held by Pj
  - There are two edges Pi → Rq and Rq → Pj in resource allocation graph, for some resource Rq.
- Deadlock -> if there is a cycle in wait-for graph.
- Invoke an algorithm to detect for cycles periodically.
- Complexity of such an algorithm will be n^2, where n is the number of vertices.

Multiple instance

- *Available:* A vector of length *m* indicates the number of available resources of each type.
- *Allocation:* An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.
- *Request:* An *n* x *m* matrix indicates the current request of each process. If *Request* [*i*][*j*] = *k*, then process *Pi* is requesting *k* more instances of resource type. *Rj*.

Multiple instance

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively

Initialize:

(a) *Work* = *Available*

(b) For $i = 1, 2, \ldots, n$, if $Allocation_i \neq 0$, then

Finish[i] = false; otherwise, *Finish*[i] = *true*.

2. Find an index *i* such that both:

(a) *Finish*[*i*] == *false*

(b) $Request_i \le Work$

If no such *i* exists, go to step 4.

Multiple instance

3. $Work = Work + Allocation_i$

   $Finish[i] = true$

   go to step 2.

4. If $Finish[i]$ == false, for some i, 1 <= i <= n, then the system is in deadlock state.

Moreover, if $Finish[i]$ == *false*, then Pi is deadlocked.

Algorithm requires an order of $O(m \times n^2)$ operations to detect whether the system is in deadlocked state.

Example

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
p4	0	0	2	0	0	2			

- Is the system in deadlocked state?
- No. Safe sequence is <P0,P2,P3,P1,P4>
- If P2 request one additional instance of C, Is there a deadlock?
- Yes

Deadlock Detection

- When to invoke detection algorithm?
- Every time a process requests for a resource
  - Adv.
    - Processes involved in deadlock are identified
    - The process that caused the deadlock is identified
  - Disadv.
    - Overhead in computation time
- After fixed interval of time
  - Disadv.
    - Can not tell which process caused the deadlock

Deadlock Recovery

1. Process termination
2. Preempt resources from some of the deadlocked processes

Process termination

1. Terminate all the deadlocked processes.
2. Terminate processes one by one until deadlock is broken

- Which process to terminate?
  - What is the priority of the process?
  - How long the process has computed?
  - How much the process has finished its working?
  - How many resources are being used by the process?
  - How many total processes will be required to be terminated?

Resource preemption

Issues

1. Select a victim
2. Rollback
    - Rollback the process to some safe state.
    - Or abort the process and restart
3. Starvation

MCQs

1. Technique used for deadlock detection is

a) wait-for graph
b) Resource allocation graph
c) Tree
d) Stack

2. Which of the following is not a method from deadlock recovery?

a) Abort all processes
b) Resource preemption from processes
c) Circular wait
d) Abort processes one by one until deadlock is broken.

- If the wait for graph contains a cycle :
  a) then a deadlock does not exist
  b) then a deadlock exists
  c) then the system is in a safe state
  d) either b or c
- A deadlock avoidance algorithm dynamically examines the ___________, to ensure that a circular wait condition can never exist.
  a) resource allocation state
  b) system storage state
  c) operating system
  d) resources

References

- Silberschatz, Abraham, et al. *Operating system concepts*. Edition-8. Reading: Addison-Wesley.

File Management

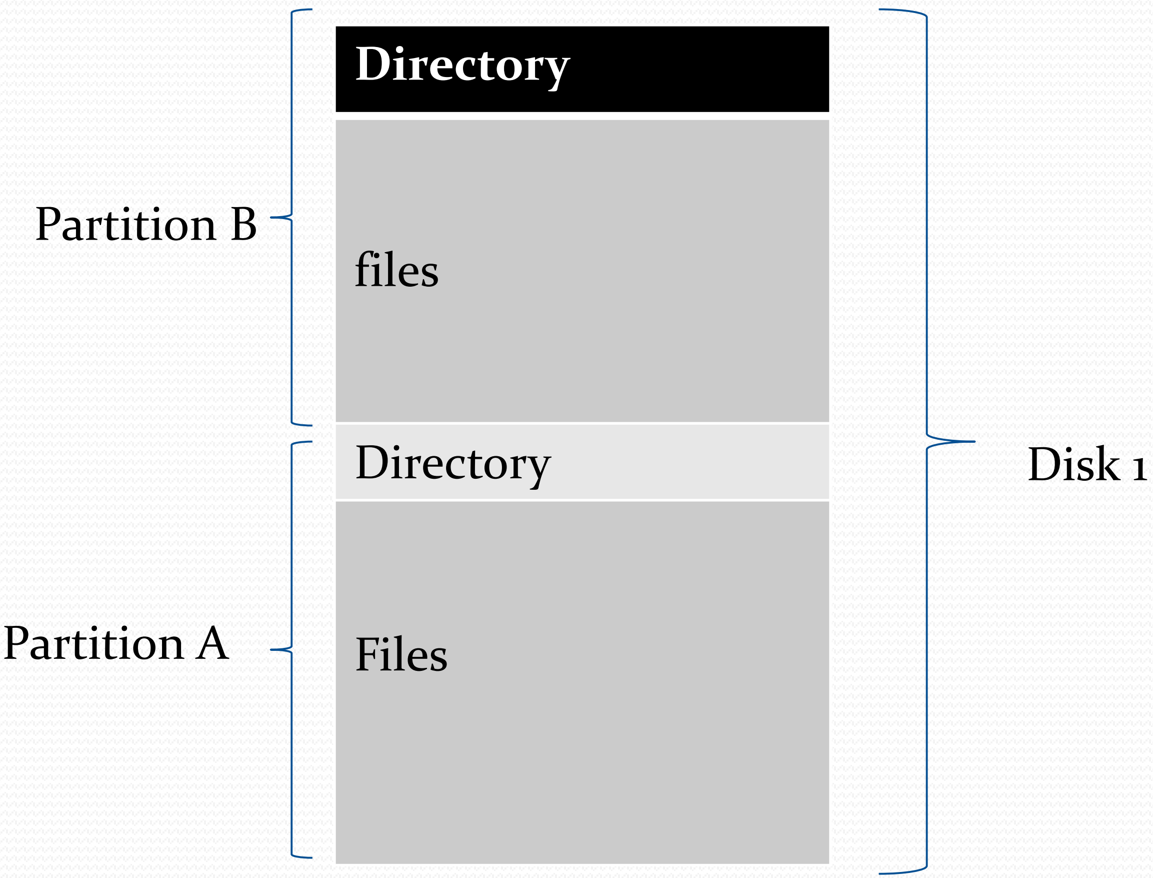
File

- File is a named collection of related information that is recorded on a secondary storage.
- Attributes:
  - Name
  - Identifier
  - Type
  - Location
  - Size
  - Protection – permissions
  - Time, date and user identification

- File Operations
  - Creation
  - Writing
  - Reading
  - Repositioning
  - Deleting
  - Truncating
- Access Methods
  - Sequential Access
  - Direct Access

Directory

- A directory contains information about files
  - Name
  - Location
  - Type
  - Size etc



Operations on a Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Directory Implementation

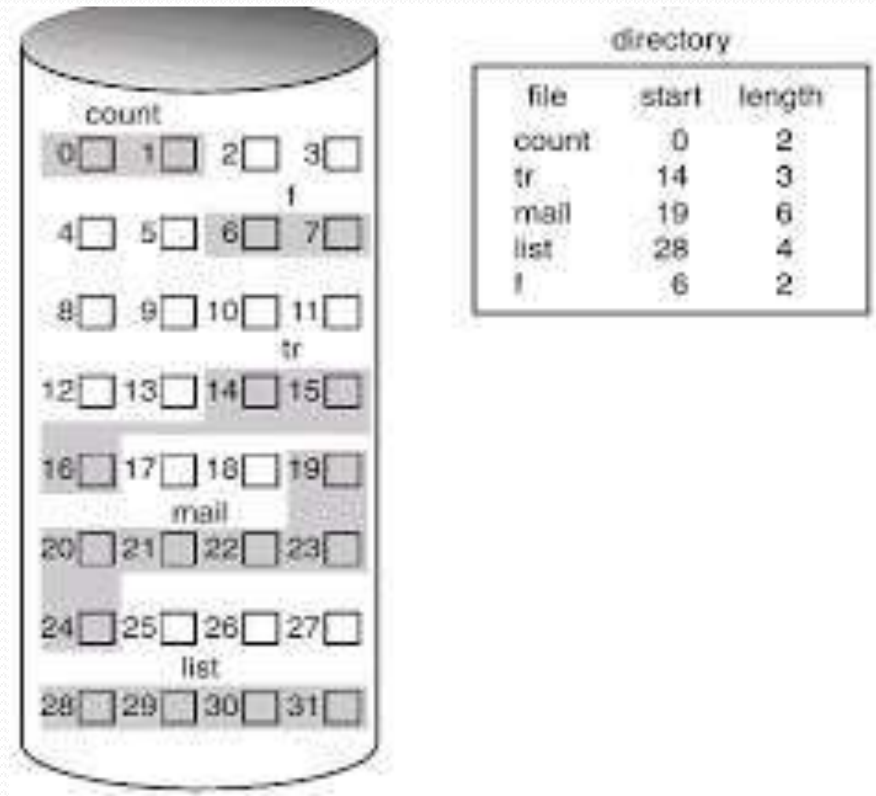
1. Linear List
   - Simple to program
   - Time consuming to execute
   - Create new file
     - Make sure it's a unique file name
     - Add entry at end of directory
   - Delete a file
     - Search for the file
     - Release the space allocated to it
     - Mark the space as unused or attach it to list of free directory entries or copy the last directory entry here.

2. Hash Table

- Linear list + hash structure
- Compute hash value of file names
- Decreases search time
- Collisions are to be avoided
- Disadvantage – hash functions are of fixed size (0-64 or 0-128 etc)

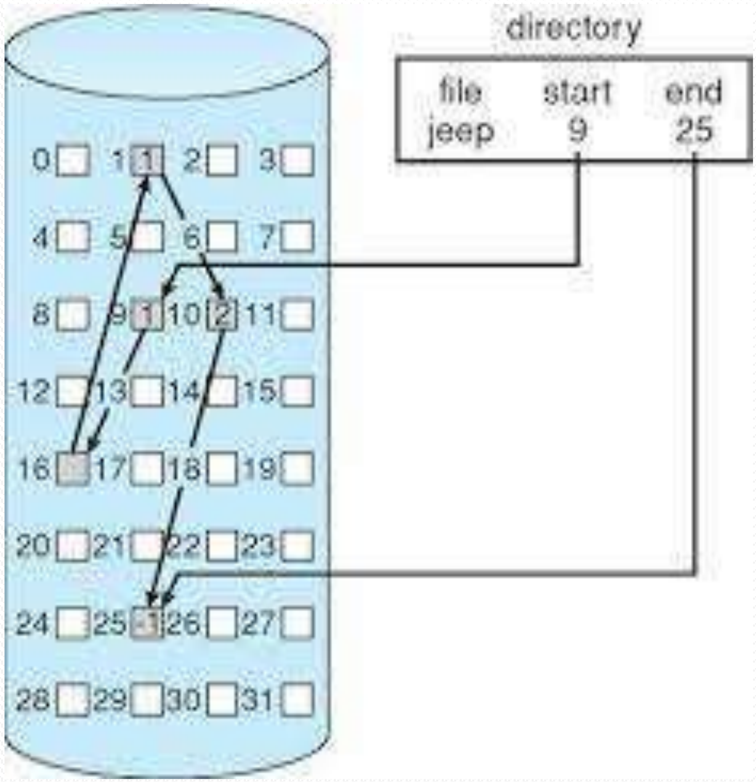
Allocation Methods

- Contiguous Allocation
  - File occupy contiguous blocks
  - Directory entry contains
    - File name
    - Starting block
    - Length (total blocks)
  - Access possible
    - Sequential
    - Direct
  - Problems
    - Finding space for new file
    - External fragmentation
    - Determining space requirement of a file

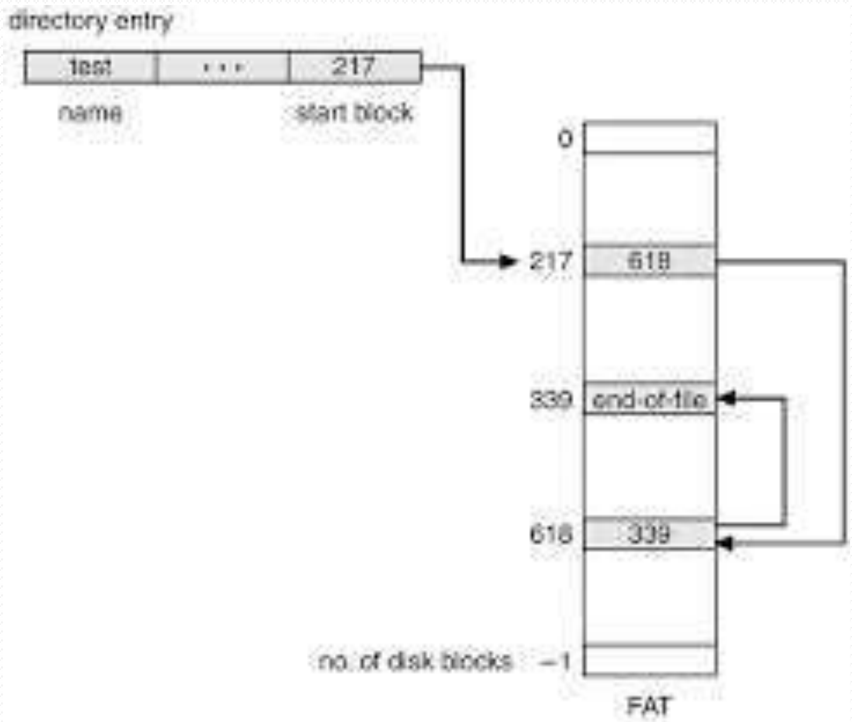


Linked Allocation

- Directory structure contains
  - Pointer to first and last block of file
- Advantages
  - No external fragmentation
  - No issue with increase in file size
- Disadvantages
  - Only sequential access
  - Reliability – loss of a pointer
  - Space required for pointers
  - Solution: make *cluster* of blocks
    - Problem: internal fragmentation



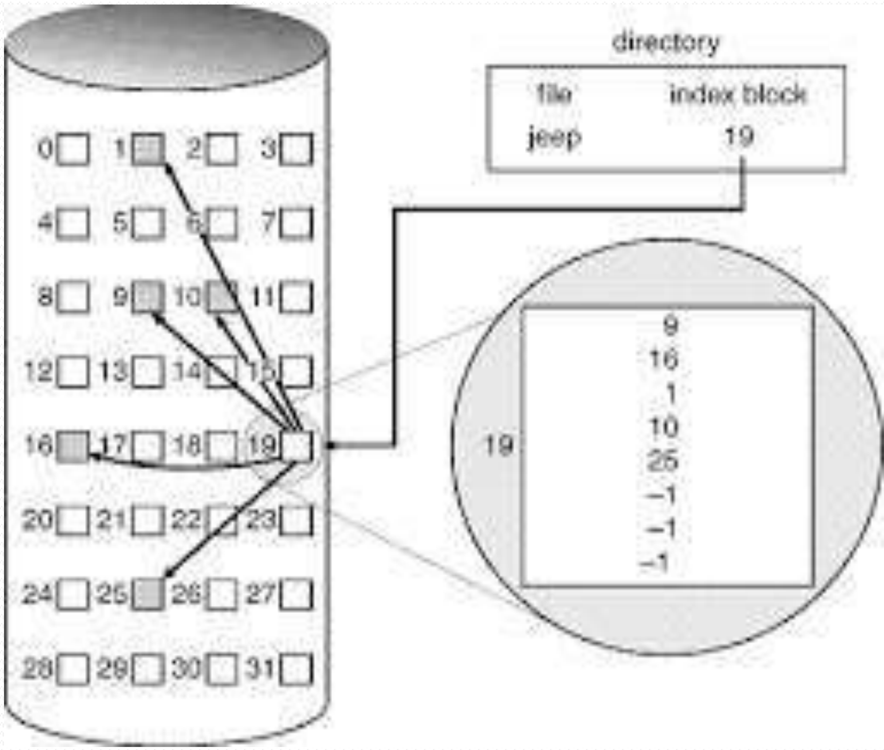
Example - FAT



directory entry
test
• • •
217
name
start block
0
217
618
339
end-of-file
618
339
no. of disk blocks
–1
FAT

Indexed Allocation

- Clubs all the pointers into one block – index block
- Directory entry contains
  - File name
  - Index block number
- Access
  - Direct
- Issue
  - Size of index block
  - Sol: multilevel indexing

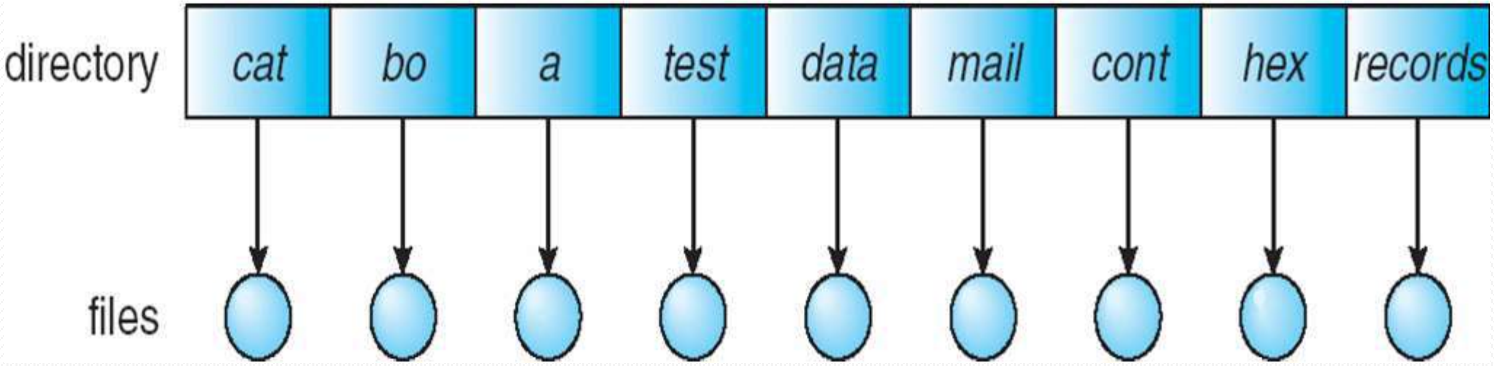


Performance

- Contiguous
  - Requires only 1 access to get a disk block
- Linked
  - Requires *i* disk reads to read i^{th} block
- Indexed
  - Depends on
    - level of indexing
    - Size of file
    - Position of desired block

Single-Level Directory

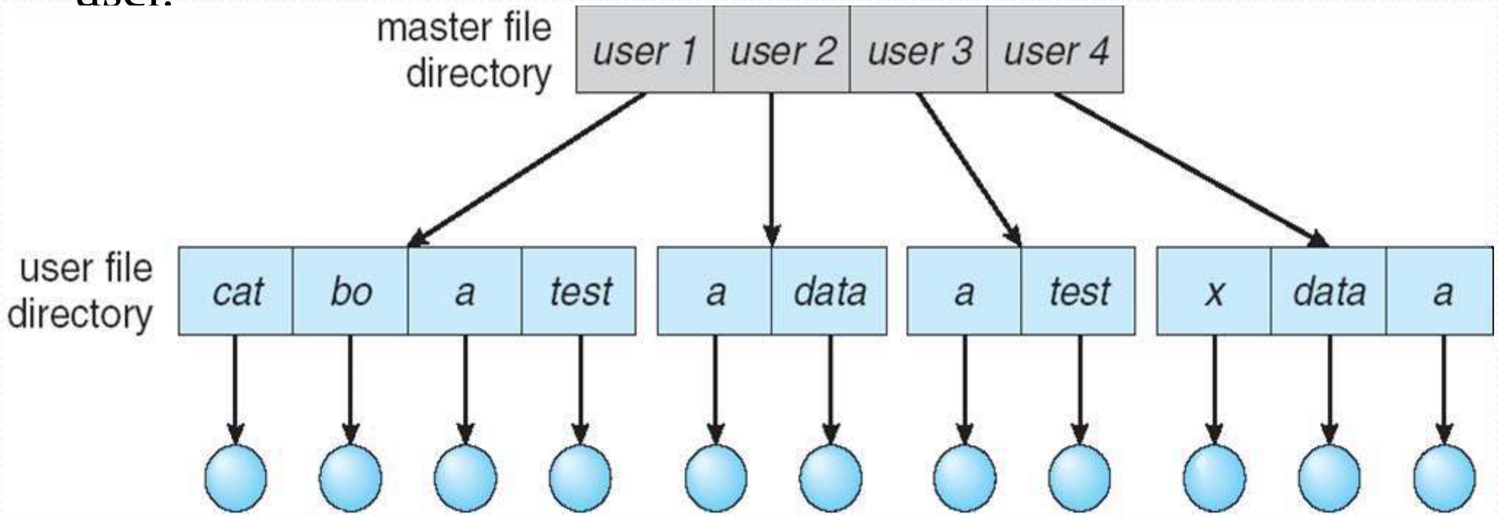
- A single directory for all users



- Easy to support and understand.
- Limitation:
- When number of files increases or when the system has When number of files increases or when the system has more than one user, then Naming problem occurs. All files should have unique names.

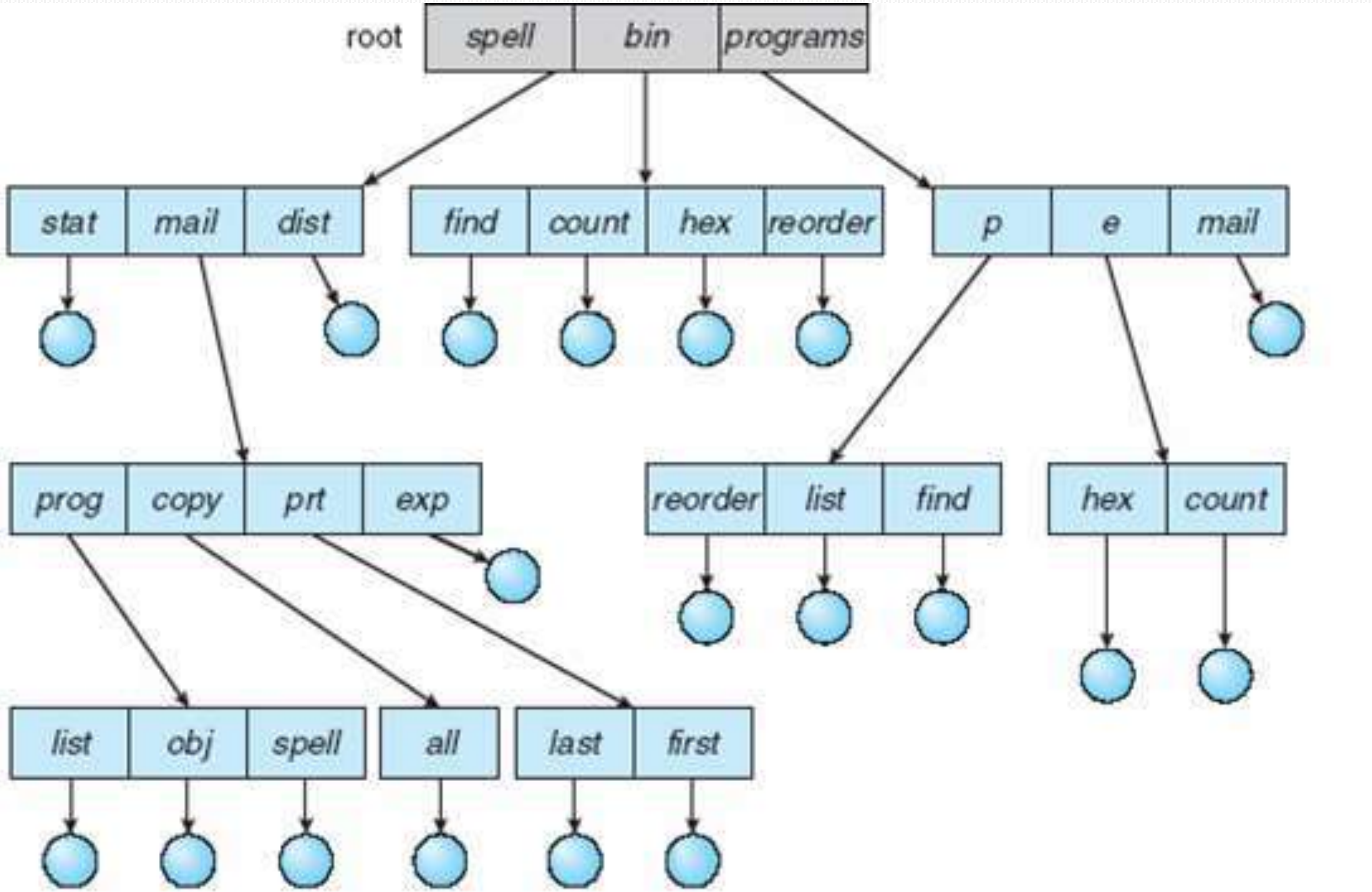
Two-Level Directory

- In two level directory, each user has his own user file directory(UFD). UFDs have the similar structure, but each lists only the files of a single user.



- Path name
- Can have the same file name for different user
- Efficient searching

Tree-Structured Directories



Tree-Structured Directories (Cont)

- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

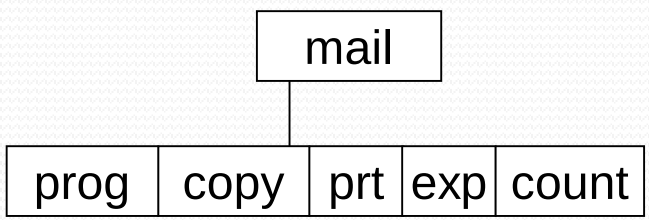
  rm <file-name>

- Creating a new subdirectory is done in current directory

  mkdir <dir-name>

  Example: if in current directory /mail

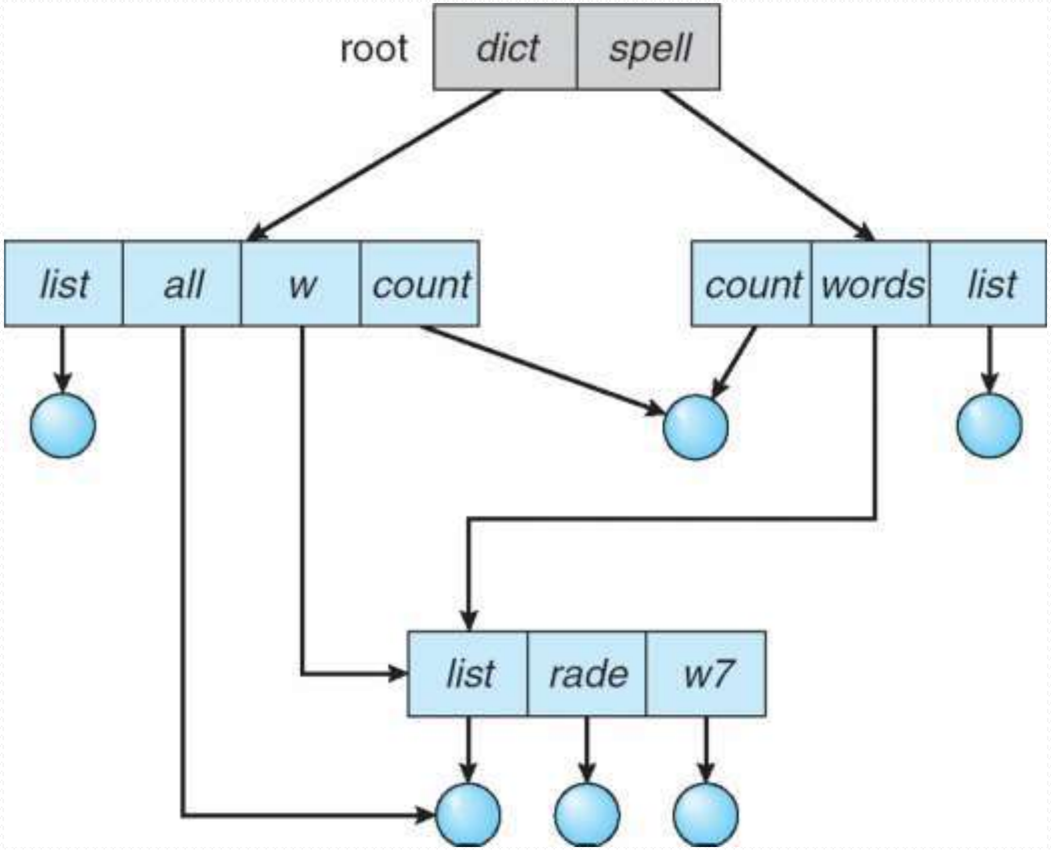
  mkdir count



Deleting “mail” $\Rightarrow$ deleting the entire subtree rooted by “mail”

Acyclic-Graph Directories

- Have shared subdirectories and files



Free Space management

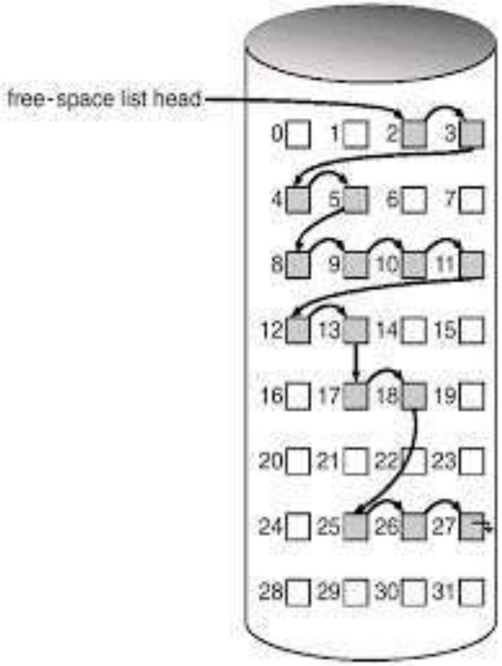
1. Bit vector
2. Linked list
3. Grouping
4. Counting

Bit vector

- Free space is implemented using bit vector
- Each block is represented by 1 bit
- 1 means free
- 0 means allocated
- E.g   001111001111110010110000 0............
- Advantage
  - Simple
  - Easy to find first free block
  - Easy to find n consecutive free blocks
- Disadvantage
  - Entire vector is to be kept in main memory – not possible for larger disks
  - 1-Tb disk with 4Kb block size requires 32 Mb to store bit map

Linked List

- Link all the free blocks
- Keep pointer to first free block in disk
- Disadvantage
  - Traversal is not easy



Grouping

- Store addresses of n free blocks in the first free block
- The first n-1 out of these are free while nth block contains addresses of another n free blocks.

Counting

- When several blocks are freed or allocated simultaneously
- Keep address of first free block and the number(n) of free contiguous free blocks that follow
- Adv.
  - Easy to locate group of free blocks