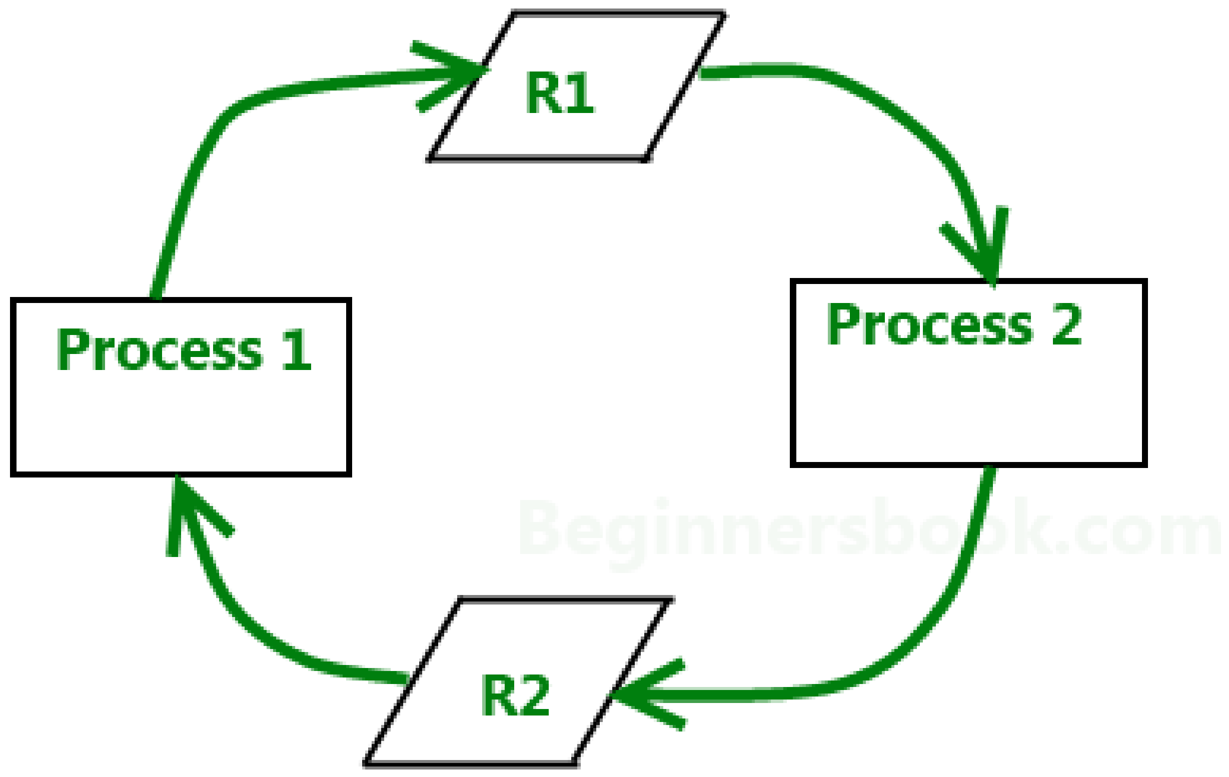
Basics of Dead-Lock

- In a multiprogramming environment, several processes may compete for a finite number of resources. 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.
- A set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set. E.g. nawab, nuclear bomb
- Starvation is long waiting but deadlock is infinite waiting



Process P1 holds resource R2 and requires R1 while Process P2 holds resource R1 and requires R2.

System model

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

- **Request**. The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.
- **Use**. The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
- **Release**. The process releases the resource.

Necessary conditions for deadlock

A deadlock can occur if all these 4 conditions occur in the system simultaneously.

- **Mutual exclusion**: - At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released. And the resource Must be desired by more than one process. i.e. one by one e.g. Printer. E.g. sects and money, board
- **Hold and wait**: - A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes. E.g. Plate and spoon
- **No pre-emption**: - Resources cannot be pre-empted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed 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 is waiting for a resource held by P_2, ..., P_{n-1} is waiting for a resource held by P_n, and P_n is waiting for a resource held by P_0.

We emphasize that all four conditions must hold for a deadlock to occur.

Deadlock Handling methods

- **Prevention**: - Design such protocols that there is no possibility of deadlock.
- **Avoidance**: - Try to avoid deadlock in run time so ensuring that the system will never enter a deadlocked state.
- **Detection**: - We can allow the system to enter a deadlocked state, then detect it, and recover.
- **Ignorance**: - We can ignore the problem altogether and pretend that deadlocks never occur in the system.

Last solution is the one used by most operating systems, including Linux and Windows. It is then up to the application developer to write programs that handle deadlocks.

Some researchers have argued that none of the basic approaches alone is appropriate for the entire spectrum of resource-allocation problems in operating systems. The basic approaches can be combined, however, allowing us to select an optimal approach for each class of resources in a system.

Polio, health, coaching / hospitals, frequency + severity

Q A Dead-lock in an Operating System is **(NET-DEC-2010)**

(A) Desirable process	**(B)** Undesirable process
(C) Definite waiting process	**(D)** All of the above

Answer: C

Prevention

It means designing such systems where there is no possibility of existence of deadlock. For that we have to remove one of the four necessary condition of deadlock.

Mutual exclusion: -In prevention approach, there is no solution for mutual exclusion as resource can't be made sharable as it is a hardware property and process also can't be convinced to do some other task. In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are intrinsically non-sharable.

Hold & wait: -

- In conservative approach, process is allowed to run if & only if it has acquired all the resources.
- An alternative protocol allows a process to request resources only when it has none. A process may request some resources and use them. Before it can request any additional resources, it must release all the resources that it is currently allocated.
- Wait time outs we place a max time outs up to which a process can wait. After which process must release all the holding resources & exit.

No pre-emption: -

- If a process is holding some resources and requests another resource that cannot be immediately allocated to it (that is, the process must wait), then all resources the process is currently holding are pre-empted. The process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
- Alternatively, if a process requests some resources, we first check whether they are available. If they are, we allocate them. If they are not, we check whether they are allocated to some other process that is waiting for additional resources. If so, we pre-empt the desired resources from the waiting process and allocate them to the requesting process. If the resources are neither available nor held by a waiting process, the requesting process must wait. While it is waiting, some of its resources may be pre-empted, but only if another process requests them. A process can be restarted only when it is allocated the new resources it is requesting and recovers any resources that were pre-empted while it was waiting.

Circular wait: -

- We can eliminate circular wait problem by giving a natural number mapping to every resource and then any process can request only in the increasing order and if a process wants a lower number, then process must first release all the resource larger than that number and then give a fresh request.

Q Consider the following policies for preventing deadlock in a system with mutually exclusive resources.
I) Process should acquire all their resources at the beginning of execution. If any resource is not available, all resources acquired so far are released.
II) The resources are numbered uniquely, and processes are allowed to request for resources only in increasing resource numbers
III) The resources are numbered uniquely, and processes are allowed to request for resources only in decreasing resource numbers
IV) The resources are numbered uniquely. A process is allowed to request for resources only for a resource with resource number larger than its currently held resources
Which of the above policies can be used for preventing deadlock? **(GATE-2015) (2 Marks)**
(A) Any one of I and III but not II or IV
(B) Any one of I, III and IV but not II
(C) Any one of II and III but not I or IV
(D) Any one of I, II, III and IV
Answer: (D)

Q Resources are allocated to the process on non-sharable basis is **(NET-JUNE-2012)**

(A) mutual exclusion	**(B)** hold and wait
(C) no pre-emption	**(D)** circular wait

Ans: a

Q Which of the following is scheme to deal with deadlock? (NET-JUNE-2012)

(A) Time out	**(B)** Time in
(C) Both (A) & (B)	**(D)** None of the above

Ans: a

Q An operating system implements a policy that requires a process to release all resources before making a request for another resource. Select the TRUE statement from the following: **(GATE-2008) (2 Marks)**
(A) Both starvation and deadlock can occur
(B) Starvation can occur but deadlock cannot occur
(C) Starvation cannot occur but deadlock can occur
(D) Neither starvation nor deadlock can occur
Answer: (B)

Q Which of the following is NOT a valid deadlock prevention scheme? **(GATE-2000) (2 Marks)**

(A) Release all resources before requesting a new resource

(B) Number the resources uniquely and never request a lower numbered resource than the last one requested.

(C) Never request a resource after releasing any resource

(D) Request and all required resources be allocated before execution.

Answer: (C)

Q Consider a system with 3 processes that share 4 instances of the same resource type. Each process can request a maximum of K instances. Resource instances can be requested and released only one at a time. The larges value of K that will always avoid deadlock is ________. **(GATE-2018) (1 Marks)**

Answer: 2

Q Consider a system having 'm' resources of the same type. These resources are shared by three processes P1, P2and P3 which have peak demands of 2, 5 and 7 resources respectively. For what value of 'm' deadlock will not occur? **(NET-AUG-2016)**

a) 70 **b)** 14 **c)** 13 **d)** 7

Answer: C

Q A system has 6 identical resources and N processes competing for them. Each process can request at most 2 resources. Which one of the following values of N could lead to a deadlock? **(GATE-2015) (1 Marks)**

a) 1 **b)** 2 **c)** 3 **d)** 6

Answer: D

Q A system contains three programs and each requires three tape units for its operation. The minimum number of tape units which the system must have such that deadlocks never arise is **(GATE-2014) (2 Marks)**

(A) 6 **(B)** 7 **(C)** 8 **(D)** 9

Answer: (B)

Q Consider a system having m resources of the same type. These resources are shared by 3 processes A, B and C which have peak demands of 3, 4 and 6 respectively. For what value of m deadlock will not occur? **(NET-DEC-2012)**

(A) 7 **(B)** 9 **(C)** 10 **(D)** 13

Answer: d

Q A computer has 6 tape drives with 'n' processes competing for them. Each process may need two drives. For which values of 'n' is the system deadlock free? **(NET-JUNE-2008)**

(A) 1 **(B)** 2 **(C)** 3 **(D)** 6

Answer: C

Q A computer has six tape drives, with n processes competing for them. Each process may need two drives. What is the maximum value of n for the system to be deadlock free? **(GATE-1998) (2 Marks)**

(A) 6 **(B)** 5 **(C)** 4 **(D)** 3

Answer: (B)

Q An operating system contains 3 user processes each requiring 2 units of resource *R*. The minimum number of units of *R* such that no deadlocks will ever arise is **(GATE-1997) (2 Marks)**

(A) 3 **(B)** 5 **(C)** 4 **(D)** 6

Answer: (C)

Q Consider a system having m resources of the same type. These resources are shared by 3 processes A, B and C, which have peak demands of 3, 4 and 6 respectively. For what value of m deadlock will not occur? **(GATE-1993) (2 Marks)**

(a) 7 **(b) 9** **(c) 10** **(d) 13** **(e) 15**

Answer: (D) & (E)

Q A computer system has 6 tape drives, with n process completing for them. Each process may need 3 tape drives. The maximum value of n for which the system is guaranteed to be deadlock free is **(GATE-1992) (2 Marks)**

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

Answer: (A)

Q Suppose n processes, P1, …. Pn share m identical resource units, which can be reserved and released one at a time. The maximum resource requirement of process Pi is Si, where Si>0. Which one of the following is a sufficient condition for ensuring that deadlock does not occur? **(GATE-2005) (2 Marks)**

a) $\forall_i$, $S_i < m$ **b)** $\forall_i$, $S_i < n$

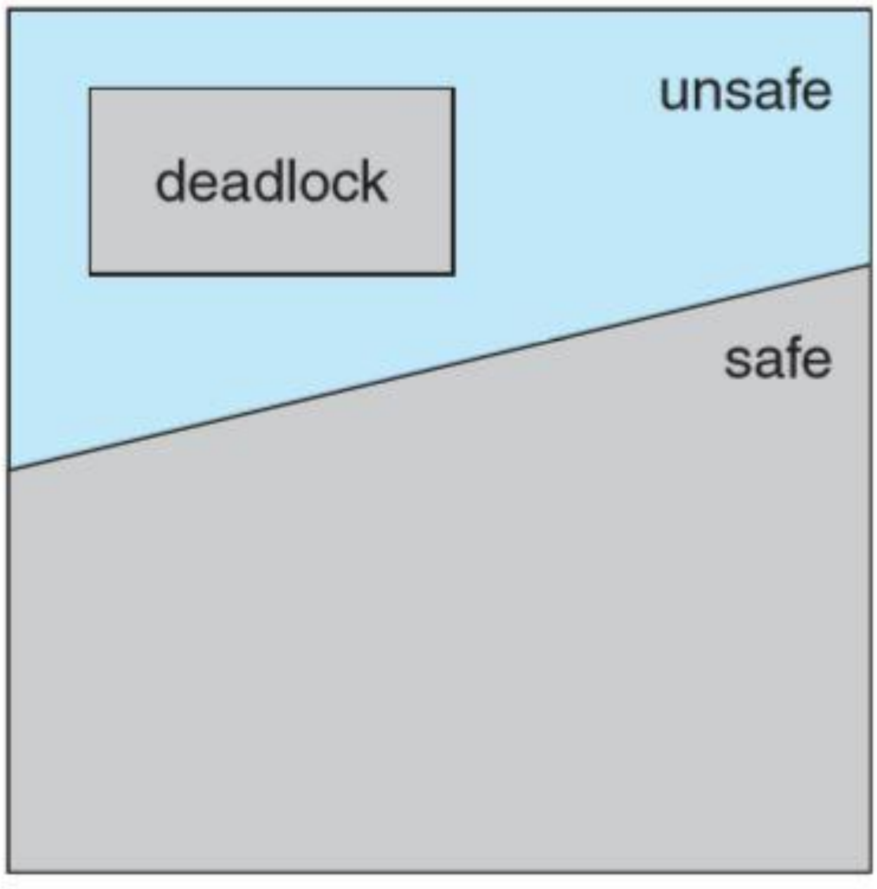
c) $\sum_{i=1}^{n} si < (m + n)$ **d)** $\sum_{i=1}^{n} si < (m * n)$

Answer: (C)

Avoidance

- **Problem with Prevention: -** Different deadlock Prevention approach put different type of restrictions or conditions on the processes and resources Because of which system becomes slow and resource utilization and reduced system throughput.
- An alternative method for avoiding deadlocks is to require additional information about how resources are to be requested. which resources a process will request and use during its lifetime i.e. maximum number of resources of each type that it may need.
- With this additional knowledge, the operating system can decide for each request whether process should wait or not.
- So, in order to avoid deadlock in run time, System try to maintain some books for recorder and like a banker, whenever someone ask for a loan(resource), it is granted only when the books allow. Let's understand by an example
- Here we use the idea of how a bank perform its operation, bank don't have its own money for business. But some people put their money in account while other borrow money from the bank, and with the difference in the interest bank operate. Whenever someone ask for a loan first bank will check whether it has sufficient funds to allow the operation.

- A deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that a circular-wait condition can never exist. The resource- allocation state is defined by the number of available and allocated resources and the maximum demands of the processes before allowing that request first, we check, if there exist "some sequence in which we can satisfies demand of every process without going into deadlock, if yes, this sequence is called safe sequence" and request can be allowed. Otherwise there is a possibility of going into deadlock.
- More formally, a system is in a safe state only if there exists a safe sequence. A sequence of processes < P_1, P_2, ..., P_n> is a safe sequence for the current allocation state if, for each P_i, the resource requests that P_i can still make can be satisfied by the currently available resources plus the resources held by all P_j, with $j < i$. In this situation, if the resources that P_i needs are not immediately available, then P_i can wait until all P_j have finished. When they have finished, P_i can obtain all of its needed resources, complete its designated task, return its allocated resources, and terminate. When P_i terminates, P_{i+1} can obtain its needed resources, and so on. If no such sequence exists, then the system state is said to be unsafe.
- Given the concept of a safe state, we can define avoidance algorithms that ensure that the system will never deadlock. The idea is simply to ensure that the system will always remain in a safe state. Initially, the system is in a safe state. Whenever a process requests a resource that is currently available, the system must decide whether the resource can be allocated immediately or whether the process must wait. The request is granted only if the allocation leaves the system in a safe state.



Banker's Algorithm

Several data structures must be maintained to implement the banker's algorithm. These data structures encode the state of the resource-allocation system. We need the following data structures, where n is the number of processes in the system and m is the number of resource types:

- **Available:** A vector of length m indicates the number of available resources of each type. If Available[j] equals k, then k instances of resource type R_j are available.
- **Max:** An n*m matrix defines the maximum demand of each process. If Max[i][j] equals k, then process 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] equals k, then process P_i is currently allocated k instances of resource type R_j.
- **Need/Demand/Requirement:** An n*m matrix indicates the remaining resource need of each process. If Need[i][j] equals k, then process P_i may need k more instances of resource type R_j to complete its task. Note that Need[i][j] = Max[i][j] − Allocation[i][j].
- These data structures vary over time in both size and value.
- We can treat each row in the matrices Allocation and Need as vectors and refer to them as $Allocation_i$ and $Need_i$. The vector $Allocation_i$ specifies the resources currently allocated to process P_i; the vector $Need_i$ specifies the additional resources that process P_i may still request to complete its task.

Safety Algorithm

- We can now present the algorithm for finding out whether or not a system is in a safe state. This algorithm can be described as follows:

```
1- Let Work and Finish be vectors of length m and n, respectively. Initialize Work = Available
and Finish[i] = false for i = 0, 1, ..., n − 1.
2- Find an index i such that both
     Finish[i] == false
     Needi ≤ Work
   If no such i exists, go to step 4.
3- Work = Work + Allocationi
     Finish[i] = true
     Go to step 2.
4- If Finish[i] == true for all i, then the system is in a safe state.
```

- This algorithm may require an order of $m*n^2$ operations to determine whether a state is safe.

Q A single processor system has three resource types X, Y and Z, which are shared by three processes. There are 5 units of each resource type. Consider the following scenario, where the column alloc denotes the number of units of each resource type allocated to each process, and the column request denotes the number of units of each resource type requested by a process in order to complete execution. Which of these processes will finish LAST? **(GATE-2007) (2 Marks)**

	Allocation			request		
	X	Y	Z	X	Y	Z
P_0	1	2	1	1	0	3
P_1	2	0	1	0	1	2
P_2	2	2	1	1	2	0

(A) P_0 **(B)** P_1
(C) P_2 **(D)** None of the above, since the system is in a deadlock
Answer: (C)

Resource-Request Algorithm

- Next, we describe the algorithm for determining whether requests can be safely granted. Let $Request_i$ be the request vector for process P_i. If $Request_i$ [j] k, then process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

1- If Requesti <= Needi, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2- If Requesti <= Available, go to step 3. Otherwise, Pi must wait, since the resources are not available.
3- Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

    Available = Available – Requesti;
    Allocationi = Allocationi + Requesti;
    Needi = Needi – Requesti;

- If the resulting resource-allocation state is safe, the transaction is completed, and process Pi is allocated its resources. However, if the new state is unsafe, then Pi must wait for Requesti, and the old resource-allocation state is restored.

Q Consider the following snapshot of a system running n concurrent processes. Process ii is holding Xi instances of a resource RR, 1≤i≤n. Assume that all instances of R are currently in use. Further, for all i, process i can place a request for at most Yi additional instances of R while holding the Xi instances it already has. Of the n processes, there are exactly two processes p and q such that Yp=Yq=0. Which one of the following conditions guarantees that no other process apart from p and q can complete execution? **(GATE-2019) (2 Marks)**

a) $X_p + X_q < \text{Min}\{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$

b) $X_p + X_q < \text{Max}\{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$

c) $\text{Min}(X_p, X_q) \ge \text{Min}\{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$

d) $\text{Min}(X_p, X_q) \le \text{Max}\{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$

Q In a system, there are three types of resources: E, F and G. Four processes P_0, P_1, P_2 and P_3 execute concurrently. At the outset, the processes have declared their maximum resource requirements using a matrix named Max as given below. For example, Max [P_2, F] is the maximum number of instances of F that P_2 would require. The number of instances of the resources allocated to the various processes at any given state is given by a matrix named Allocation.

Consider a state of the system with the Allocation matrix as shown below, and in which 3 instances of E and 3 instances of F are the only resources available.

	Allocation			**Max**		
	E	F	G	E	F	G
P_0	1	0	1	4	3	1
P_1	1	1	2	2	1	4
P_2	1	0	3	1	3	3
P_3	2	0	0	5	4	1

From the perspective of deadlock avoidance, which one of the following is true? (GATE-2018) (2 Marks)

(A) The system is in *safe* state

(B) The system is not in *safe* state, but would be *safe* if one more instance of E were available

(C) The system is not in *safe* state, but would be *safe* if one more instance of F were available

(D) The system is not in *safe* state, but would be *safe* if one more instance of G were available

Answer: (A)

Q Consider a system with five processes P0 through P4 and three resource types A, B and C. Resource type A has seven instances, resource type B has two instances and resource type C has six instances suppose at time T0we have the following allocation.

	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	2	2	0	0	2			

If we implement Deadlock detection algorithm, we claim that system is ___________. **(NET-NOV-2017)**

(1) Semaphore **(2)** Deadlock state **(3)** Circular wait **(4)** Not in deadlock state

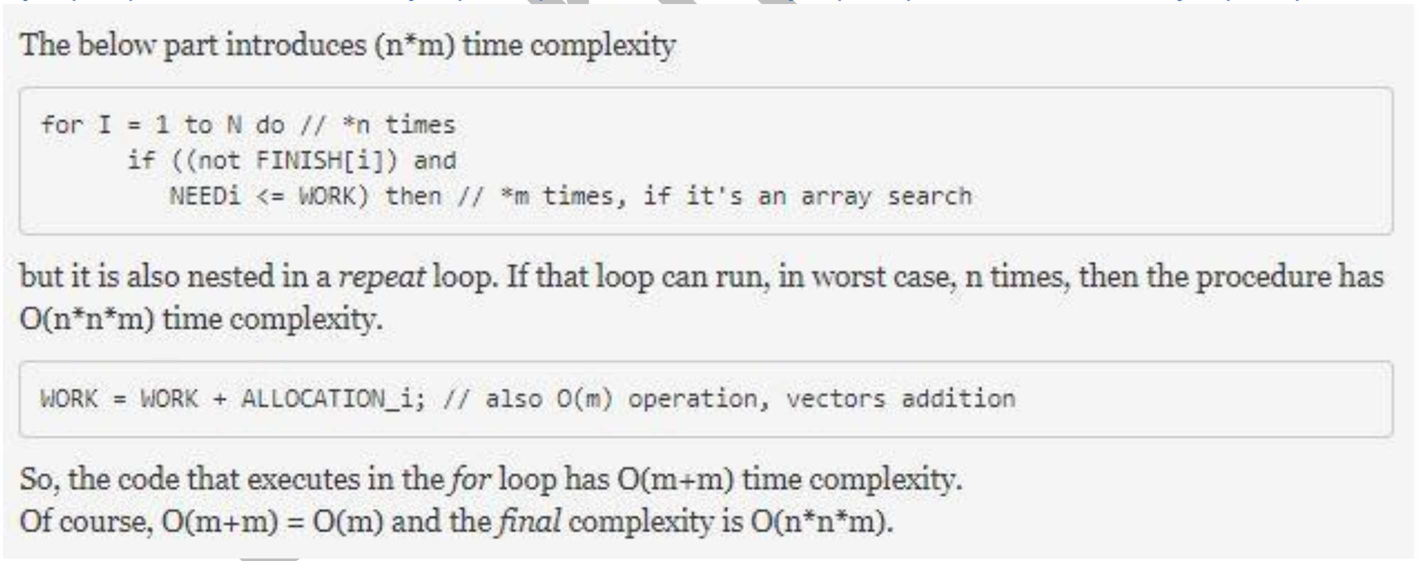
a) (1), (2) and (3) **b)** (1) and (3)

c) (3) and (4) **d)** All are correct

Answer: D

Q Consider a system which have 'n' number of processes and 'm' number of resource types. The time complexity of the safety algorithm, which checks whether a system is in safe state or not, is of the order of: **(NET-AUG-2016)**

a) O(mn) **b)** O(m^2n^2) **c)** O(m^2n) **d)** O(mn^2)



The below part introduces (n*m) time complexity

```
for I = 1 to N do // *n times
      if ((not FINISH[i]) and
         NEEDi <= WORK) then // *m times, if it's an array search
```

but it is also nested in a *repeat* loop. If that loop can run, in worst case, n times, then the procedure has O(n*n*m) time complexity.

```
WORK = WORK + ALLOCATION_i; // also O(m) operation, vectors addition
```

So, the code that executes in the *for* loop has O(m+m) time complexity.
Of course, O(m+m) = O(m) and the *final* complexity is O(n*n*m).

Q Suppose there are four processes in execution with 12 instances of a Resource R in a system. The maximum need of each process and current allocation are given below:

Process	Max Need	Allocation
P_1	8	3
P_2	9	4
P_3	5	2
P_4	3	1

With reference to current allocation, is system safe? If so, what is the safe sequence? **(NET-JULY-2016)**

a) No **b)** Yes, P1P2P3P4 **c)** Yes, P4P3P1P2 **d)** Yes, P2P1P3P4

Answer: (c)

Q Consider a system with twelve magnetic tape drives and three processes P_1, P_2 and P_3. Process P_1 requires maximum ten tape drives, process P_2 may need as many as four tape drives and P_3 may need up to nine tape drives. Suppose that at time t_1, process P_1 is holding five tape drives, process P_2 is holding two tape drives and process P_3 is holding three tape drives. At time t_1, system is in: **(NET-DEC-2015)**

Process	**Max Need**	**Allocation**	**Current Need**
P_1	10	5	
P_2	4	2	
P_3	9	3	

A) safe state **b)** unsafe state

c) deadlocked state **d)** starvation state

Ans b

Q A system has four processes and five resources. The current allocation and maximum needs are as follows: **(NET-DEC-2015)**

	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
	1	0	2	1	1	1	1	2	1	3	0	0	X	1	1
	2	0	1	1	0	2	2	2	1	0					
	1	1	0	1	0	2	1	3	1	0					
	1	1	1	1	0	1	1	2	2	1					

The smallest value of x for which the above system in safe state is ___________.

a) 1 **b)** 3 **c)** 2 **d)** Not safe for any value of x

Q A system shares 9 tape drives. The current allocation and maximum requirement of tape drives for 3 processes are shown below:

Process	Current Allocation	Maximum Requirement
P_1	3	7
P_2	1	6
P_3	3	5

Which of the following best describes the current state of the system? **(GATE-2014) (2 Marks)**

(A) Safe, Deadlocked

(B) Safe, Not Deadlocked

(C) Not Safe, Deadlocked

(D) Not Safe, Not Deadlocked

Answer: (B)

Q An operating system has 13 tape drives. There are three processes P_1, P_2 & P_3. Maximum requirement of P_1 is 11 tape drives, P_2 is 5 tape drives and P_3 is 8 tape drives. Currently, P_1 is allocated 6 tape drives, P_2 is allocated 3 tape drives and P_3 is allocated 2 tape drives. Which of the following sequences represent a safe state? **(NET-DEC-2014)**

Process	Max Need	Allocation	Current Need
P_1	11	6	
P_2	5	3	
P_3	8	2	

(A) P_2 P_1 P_3

(B) P_2 P_3 P_1

(C) P_1 P_2 P_3

(D) P_1 P_3 P_2

ans a

Q An operating system uses the Banker's algorithm for deadlock avoidance when managing the allocation of three resource types X, Y, and Z to three processes P_0, P_1, and P_2. The table given below presents the current system state. Here, the Allocation matrix shows the current number of resources of each type allocated to each process and the Max matrix shows the maximum number of resources of each type required by each process during its execution.

	Allocation			Max		
	X	Y	Z	X	Y	Z
P_0	0	0	1	8	4	3
P_1	3	2	0	6	2	0
P_2	2	1	1	3	3	3

There are 3 units of type X, 2 units of type Y and 2 units of type Z still available. The system is currently in a safe state. Consider the following independent requests for additional resources in the
current state:
REQ1: P0 requests 0 units of X, 0 units of Y and 2 units of Z
REQ2: P1 requests 2 units of X, 0 units of Y and 0 units of Z
Which one of the following is TRUE? **(GATE-2014) (2 Marks)**
(A) Only REQ1 can be permitted.
(B) Only REQ2 can be permitted.
(C) Both REQ1 and REQ2 can be permitted.
(D) Neither REQ1 nor REQ2 can be permitted
Answer: (B)

Q Consider a system with five processes P0 through P4 and three resource types R1 R and R3. Resource type R1 has 10 instances, R2 has 5 instances and R3 has 7 instances. Suppose that at time T0, the following snapshot of the system has been taken: (NET-JUNE-2014)
Assume that now the process P1 requests one additional instance of type R1 and two instances of resource type R3. The state resulting after this allocation will be

	Allocation			**Request**			**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	3	9	0	2			
P3	2	1	1	2	2	2			
P4	0	2	2	4	3	3			

(A) Ready state **(B)** Safe state **(C)** Blocked state **(D)** Unsafe state

Q An operating system using banker's algorithm for deadlock avoidance has ten dedicated devices (of same type) and has three processes P_1, P_2 and P_3 with maximum resource requirements of 4, 5 and 8 respectively. There are two states of allocation of devices as follows: **(NET-JUNE-2013)**

State 1	Processes	P_1	P_2	P_3
	Device allocated	2	3	4
State 2	Processes	P_1	P_2	P_3
	Device allocated	0	2	4

Which of the following is correct ?
(A) State 1 is unsafe and state 2 is safe.
(B) State 1 is safe and state 2 is unsafe.

(C) Both, state 1 and state 2 are safe. **(D)** Both, state 1 and state 2 are unsafe.
Ans: a

Q Which of the following is NOT true of deadlock prevention and deadlock avoidance schemes?**(GATE-2008) (2 Marks)**
(A) In deadlock prevention, the request for resources is always granted if the resulting state is safe
(B) In deadlock avoidance, the request for resources is always granted if the result state is safe
(C) Deadlock avoidance is less restrictive than deadlock prevention
(D) Deadlock avoidance requires knowledge of resource requirements a priori
Answer: (A)

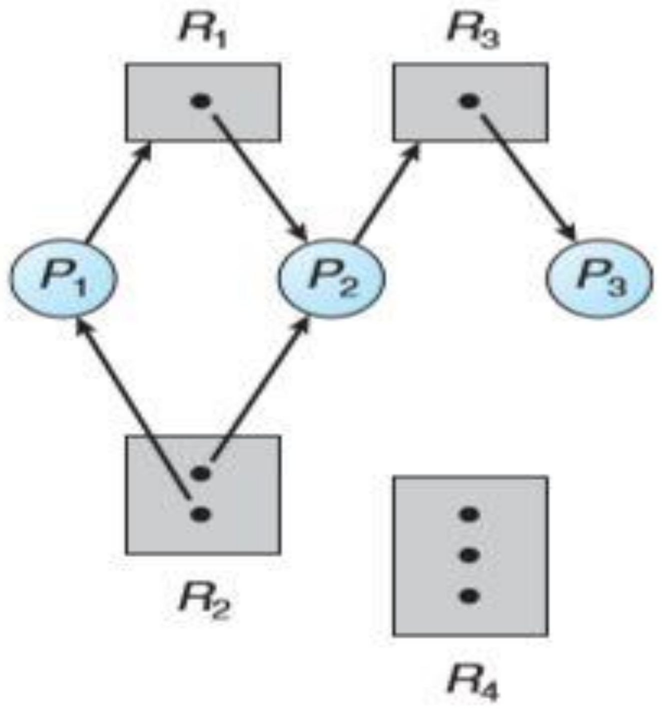
Q Consider the following snapshot of a system running n processes. Process P_i is holding X_i instances of a resource R, $1 <= i <= n$. currently, all instances of R are occupied. Further, for all i, process i has placed a request for an additional Y_i instances while holding the X_i instances it already has. There are exactly two processes p and q such that $Y_p = Y_q = 0$. Which one of the following can serve as a necessary condition to guarantee that the system is not approaching a deadlock? **(GATE-2006) (2 Mark)**
(A) $\min (X_p, X_q) < \max (Y_k)$ where $k != p$ and $k != q$
(B) $X_p + X_q >= \min (Y_k)$ where $k != p$ and $k != q$
(C) $\max (X_p, X_q) > 1$
(D) $\min (X_p, X_q) > 1$
Answer: (B)

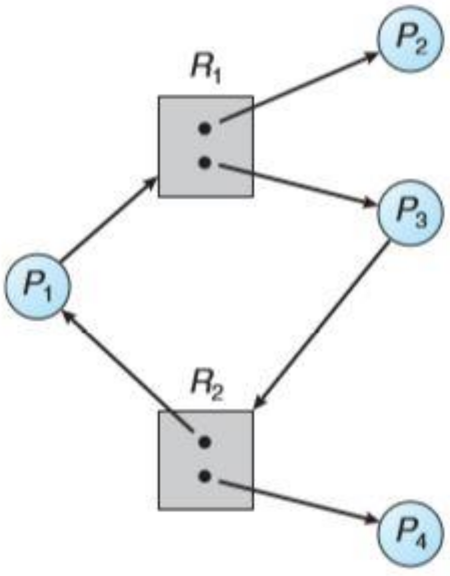
Q Banker's algorithm is used for .... purpose. **(NET-DEC-2005)**
(A) Deadlock avoidance **(B)** Deadlock removal
(C) Deadlock prevention **(D)** Deadlock continuations
Answer: A

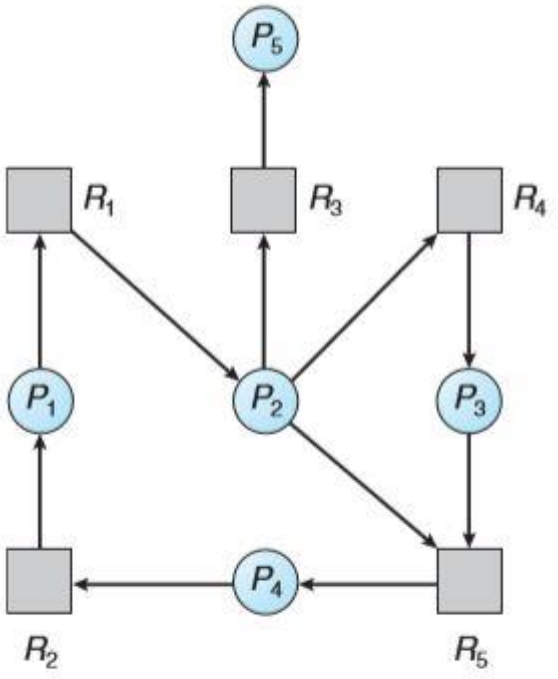
Q Bankers algorithm is for. **(NET-JUNE-2005)**
(A) Dead lock Prevention **(B)** Dead lock Avoidance
(C) Dead lock Detection **(D)** Dead lock creation
Answer: B

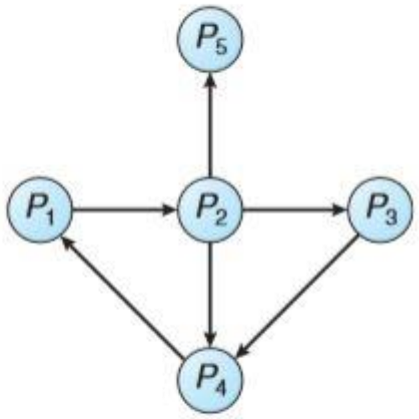
Resource Allocation Graph

- Deadlock can be described more precisely in terms of a directed graph called a system resource-allocation graph. This graph consists of a set of vertices V and a set of edges E.
- The set of vertices V is partitioned into two different types of nodes: P = {P_1, P_2, ..., P_n}, the set consisting of all the active processes in the system, and R = {R_1, R_2, ..., R_m}, the set consisting of all resource types in the system.
- A directed edge from process P_i to resource type R_j is denoted by $P_i \rightarrow R_j$; it signifies that process P_i has requested an instance of resource type R_j and is currently waiting for that resource. A directed edge from resource type R_j to process P_i is denoted by $R_j \rightarrow P_i$; it signifies that an instance of resource type R_j has been allocated to process P_i. A directed edge $P_i \rightarrow R_j$ is called a request edge; a directed edge $R_j \rightarrow P_i$ is called an assignment edge.
- Pictorially, we represent each process P_i as a circle and each resource type R_j as a rectangle. Since resource type R_j may have more than one instance, we represent each such instance as a dot within the rectangle. Note that a request edge points to only the rectangle R_j, whereas an assignment edge must also designate one of the dots in the rectangle.
- When process P_i requests an instance of resource type R_j, a request edge is inserted in the resource-allocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to the resource, it releases the resource. As a result, the assignment edge is deleted.









Cycle is resource allocation graph

Deadlock detection and recovery

- When should we invoke the detection algorithm? It depends on two factors:
    - How often is a deadlock likely to occur?
    - How many processes will be affected by deadlock when it happens?
- **Active approach:** If deadlocks occur frequently, then the detection algorithm should be invoked frequently. Of course, invoking the deadlock-detection algorithm for every resource request will incur considerable overhead in computation time.
- **Lazy approach:** A less expensive alternative is simply to invoke the algorithm at defined intervals—for example, once per hour or whenever CPU utilization drops below 40 percent.

Recovery from Deadlock

- When a detection algorithm determines that a deadlock exists, several alternatives are available. One possibility is to inform the operator that a deadlock has occurred and to let the operator deal with the deadlock manually. Another possibility is to let the system recover from the deadlock automatically. There are two options for breaking a deadlock.
- **Process Termination:** One is simply to abort one or more processes to break the circular wait.
- **Pre-empt Resource:** The other is to pre-empt some resources from one or more of the deadlocked processes.

Process Termination

- **Abort all deadlocked processes**: This method clearly will break the deadlock cycle, but at great expense. The deadlocked processes may have computed for a long time, and the results of these partial computations must be discarded and probably will have to be recomputed later.
- **Abort one process at a time**: until the deadlock cycle is eliminated. This method incurs considerable overhead, since after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.
- Aborting a process may not be easy. If the process was in the midst of updating a file, terminating it will leave that file in an incorrect state. Similarly, if the process was in the midst of printing data on a printer, the system must reset the printer to a correct state before printing the next job.

How to choose a victim

If the partial termination method is used, then we must determine which deadlocked process (or processes) should be terminated. This determination is a policy decision, similar to CPU-scheduling decisions. The question is basically an economic one; we should abort those processes whose termination will incur the minimum cost. Unfortunately, the term minimum cost is not a precise one. Many factors may affect which process is chosen, including:

- What the priority of the process is
- How long the process has computed and how much longer the process will compute before completing its designated task
- How many and what types of resources the process has used (for example, whether the resources are simple to pre-empt)
- How many more resources the process needs in order to complete
- How many processes will need to be terminated?
- Whether the process is interactive or batch

Resource Pre-emption

- To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken. If preemption is required to deal with deadlocks, then three issues need to be addressed:
- **Selecting a victim** Which resources and which processes are to be preempted? As in process termination, we must determine the order of preemption to minimize cost. Cost factors may include such parameters as the number of resources a deadlocked process is holding and the amount of time the process has thus far consumed.
- **Rollback.** If we preempt a resource from a process, what should be done with that process? Clearly, it cannot continue with its normal execution; it is missing some needed resource. We must roll back the process to some safe state and restart it from that state. Since, in general, it is difficult to determine what a safe state is, the simplest solution is a total rollback: abort the process and then restart it. Although it is more effective to roll back the process only as far as necessary to break the deadlock, this method requires the system to keep more information about the state of all running processes.
- **Starvation.** How do we ensure that starvation will not occur? That is, how can we guarantee that resources will not always be preempted from the same process? In a system where victim selection is based primarily on cost factors, it may happen that the same process is always picked as a victim. As a result, this process never completes its designated task, a starvation situation any practical system must address. Clearly, we must ensure that a process can be picked as a victim only a (small) finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

Q Consider a system with 4 types of resources R1 (3 units), R2 (2 units), R3 (3 units), R4 (2 units). A non-pre-emptive resource allocation policy is used. At any given instance, a request is not entertained if it cannot be completely satisfied. Three processes P1, P2, P3 request the sources as follows if executed independently. **(GATE-2009) (2 Marks)**

Process P1:	Process P2:	Process P3:
t=0: requests 2 units of R2	t=0: requests 2 units of R3	t=0: requests 1 unit of R4
t=1: requests 1 unit of R3	t=2: requests 1 unit of R4	t=2: requests 2 units of R1
t=3: requests 2 units of R1	t=4: requests 1 unit of R1	t=5: releases 2 units of R1
t=5: releases 1 unit of R2 and 1 unit of R1.	t=6: releases 1 unit of R3	t=7: requests 1 unit of R2
t=7: releases 1 unit of R3	t=8: Finishes	t=8: requests 1 unit of R3
t=8: requests 2 units of R4		t=9: Finishes
t=10: Finishes		

Which one of the following statements is TRUE if all three processes run concurrently starting at time t=0?

(A) All processes will finish without any deadlock
(B) Only P1 and P2 will be in deadlock.
(C) Only P1 and P3 will be in a deadlock.
(D) All three processes will be in deadlock
Answer: (A)

Q Simplest way of deadlock recovery is **(NET-DEC-2013)**

a) Roll back	**b)** Pre-empt resource
c) Lock one of the processes	**d)** Kill one of the processes

Ans: d

Ignorance

- In the absence of algorithms to detect and recover from deadlocks, we may arrive at a situation in which the system is in a deadlocked state yet has no way of recognizing what has happened. In this case, the undetected deadlock will cause the system's performance to deteriorate, because resources are being held by processes that cannot run and because more and more processes, as they make requests for resources, will enter a deadlocked state. Eventually, the system will stop functioning and will need to be restarted manually.
- Although this method may not seem to be a viable approach to the deadlock problem, it is nevertheless used in most operating systems. Expense is one important consideration. Ignoring the possibility of deadlocks is cheaper than the other approaches. Since in many systems, deadlocks occur infrequently (say, once per year), the extra expense of the other methods may not seem worthwhile. In addition, methods used to recover from other conditions may be put to use to recover from deadlock. In some circumstances, a system is in a frozen state but not in a deadlocked state. We see this situation, for example, with a real-time process running at the highest priority (or any process running on a nonredemptive scheduler) and never returning control to the operating system. The system must have manual recovery methods for such conditions and may simply use those techniques for deadlock recovery.

Q Consider the solution to the bounded buffer producer/consumer problem by using general semaphores S,F and E. The semaphore S is the mutual exclusion semaphore initialized to 1. The semaphore F corresponds to the number of free slots in the buffer and is initialized to N. The semaphore E corresponds to the number of elements in the buffer and is initialized to 0. **(GATE-2006) (2 Mark)**

Producer()	Consumer()
{	{
while(T)	while(T)
{	{
Produce()	wait(E)//UnderFlow
wait(F)//OverFlow	wait(S)
Wait(S)	pick()
append()	signal(S)
signal(S)	signal(F)
signal(E)	consume()
}	}
}	}

Which of the following interchange operations may result in a deadlock?

I) Interchanging Wait (F)(F) and Wait (S)(S) in the Producer process

II) Interchanging Signal (S)(S) and Signal (F)(F) in the Consumer process

a) (I) only **b)** (II) only

c) Neither (I) nor (II) **d)** Both (I) and (II)

Ans: a

Q In a certain operating system, deadlock prevention is attempted using the following scheme. Each process is assigned a unique timestamp, and is restarted with the same timestamp if killed. Let Ph be the process holding a resource R, Pr be a process requesting for the same resource R, and T(Ph) and T(Pr) be their timestamps respectively. The decision to wait or preempt one of the processes is based on the following algorithm.

```
 if T(Pr) < T(Ph)
     then kill Pr
else wait
```

Which one of the following is TRUE? **(GATE-2004) (2 Marks)**

(A) The scheme is deadlock-free, but not starvation-free

(B) The scheme is not deadlock-free, but starvation-free

(C) The scheme is neither deadlock-free nor starvation-free

(D) The scheme is both deadlock-free and starvation-free

Answer: (A)

Q The following is a code with two threads, producer and consumer, that can run in parallel. Further, S and Q are binary semaphores equipped with the standard P and V operations.

```
semaphore S = 1, Q = 0;
integer x;
producer:                  consumer:
while (true) do             while (true) do
P(S);                       P(Q);
x = produce ();             consume (x);
V(Q);                       V(S);
done                          done
```

Which of the following is TRUE about the program above? **(GATE-2008) (2 Marks)**

(A) The process can deadlock

(B) One of the threads can starve

(C) Some of the items produced by the producer may be lost

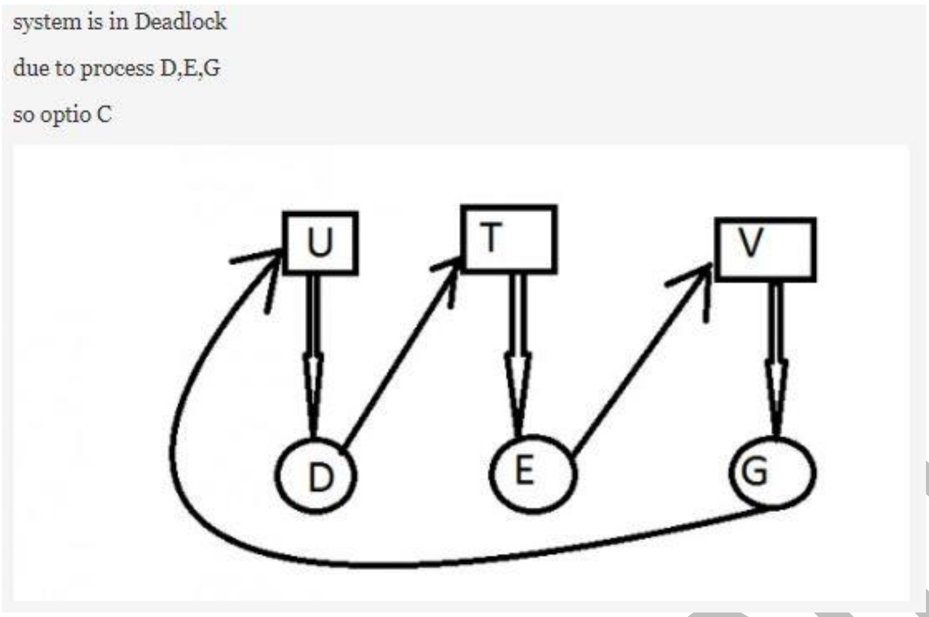
(D) Values generated and stored in ‘x’ by the producer will always be consumed before the producer can generate a new value

Answer: (D)

Q Consider a system with seven processes A through G and six resources R through W. Resource ownership is as follows : process A holds R and wants T process B holds nothing but wants T process C holds nothing but wants S process D holds U and wants S & T process E holds T and wants V process F holds W and wants S process G holds V and wants U Is the system deadlocked ? If yes, _______ processes are deadlocked. **(NET-AUG-2016)**

a) No **b)** Yes, A, B, C **c)** Yes, D, E, G **d)** Yes, A, B, F

Ans: c


system is in Deadlock

due to process D,E,G

so optio C

Q Consider a system with five processes P0 through P4 and three resource types R1 R and R3. Resource type R1 has 10 instances, R2 has 5 instances and R3 has 7 instances. Suppose that at time T0, the following snapshot of the system has been taken: **(NET-JUNE-2014)**

	Allocation			**Request**			**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	3	9	0	2			
P3	2	1	1	2	2	2			
P4	0	2	2	4	3	3			

Assume that now the process P1 requests one additional instance of type R1 and two instances of resource type R3. The state resulting after this allocation will be

(A) Ready state **(B)** Safe state **(C)** Blocked state **(D)** Unsafe state

Ans: b