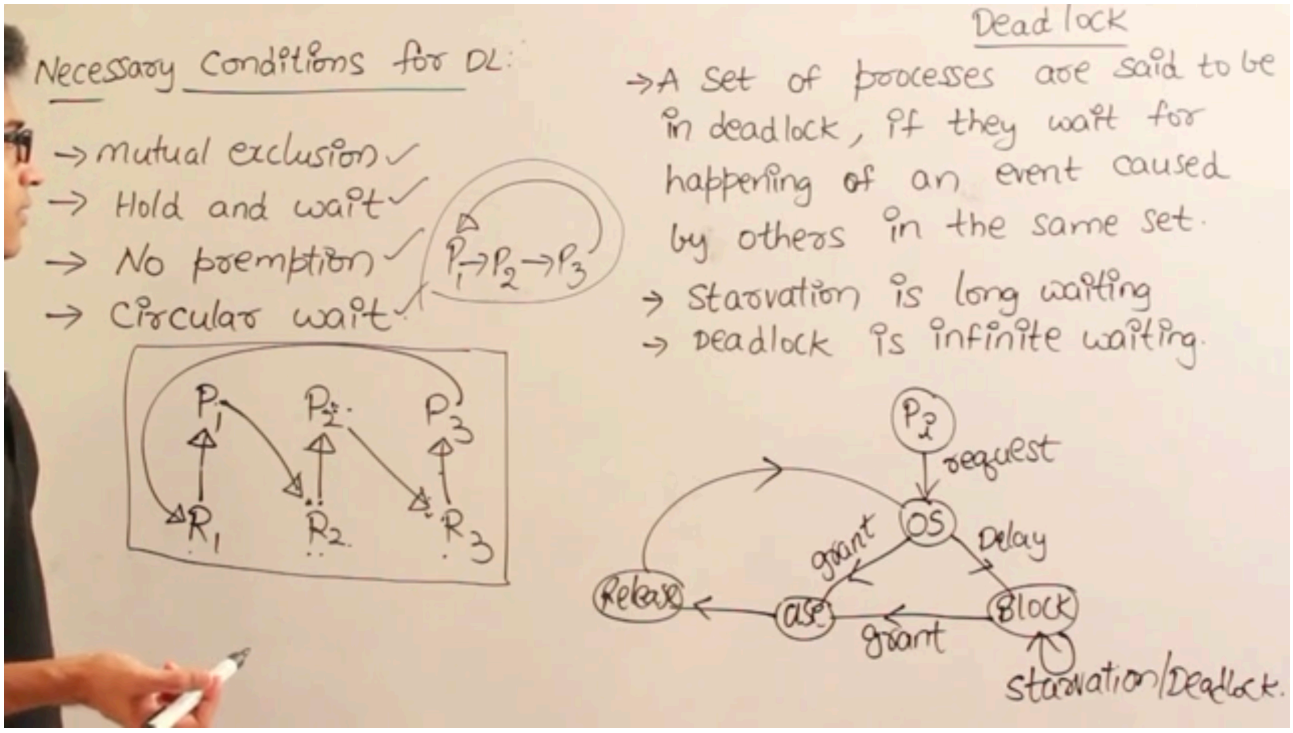
Lecture #1: Introduction to Deadlocks

A classic example of deadlock: A son asking his father to buy him a bike, and the father asking his son to learn the bike before buying him the bike.

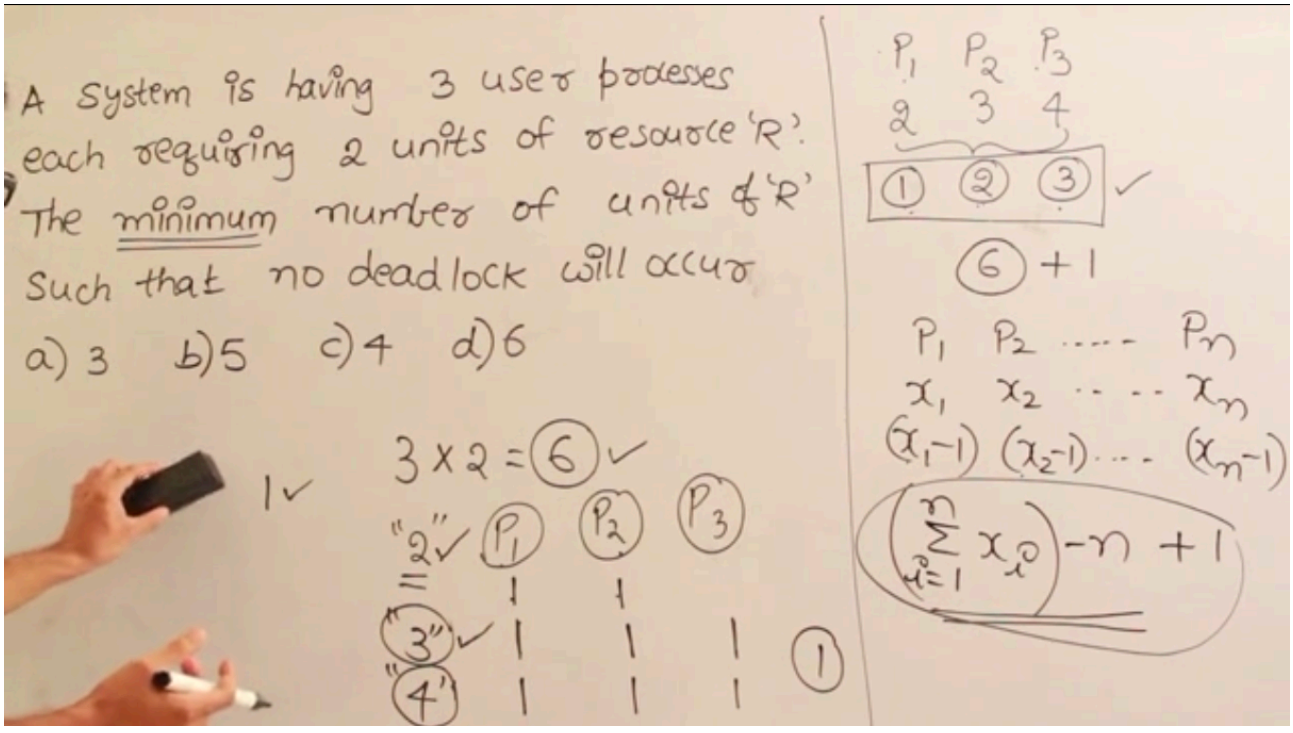
This is a deadlock as the son cannot learn the bike before the father has bought him the bike, but before learning the bike, father will not buy him the bike.

A resource in a computer can be anything, ranging from a file, printer, resistor to a mutex.



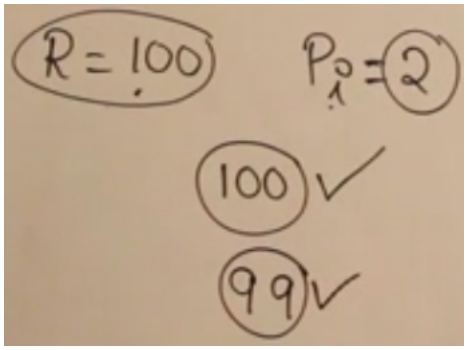
Lecture #2: Gate 1997 Question and More Examples

To find the minimum number of resources required so that no deadlock is possible, find the max number of resources from which deadlock is still possible, and then add one to it.

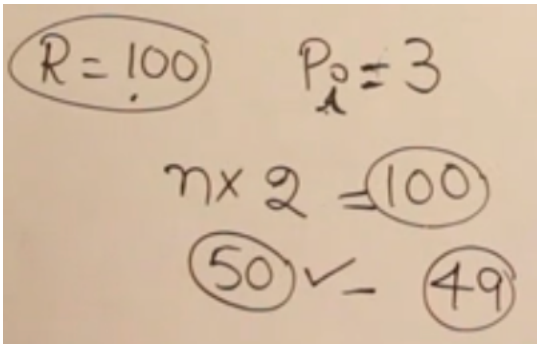


Q. Max number of processes which are required so that there is no deadlock when there are 100 resources and each process takes 2 resources?

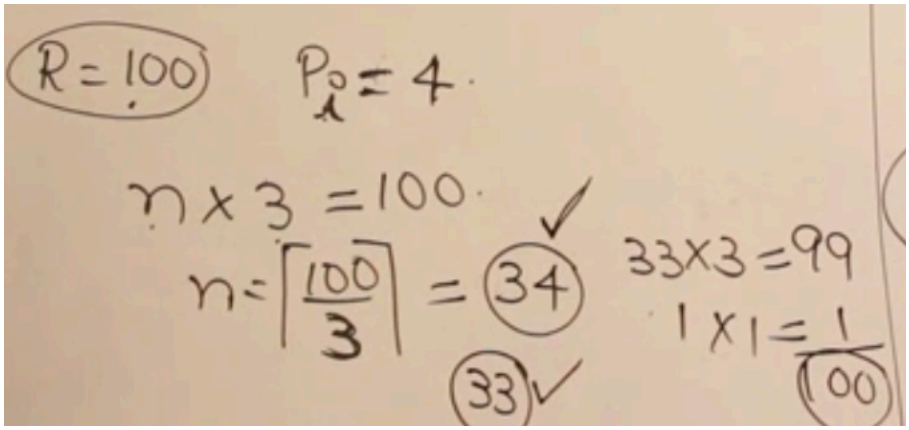
We should ask the minimum number of processes required to cause a deadlock. It is 100 in our case if each takes 1 resource. ∴ Max no of processes = 100 - 1 = 99



In the above question, if each process requires 3 resources instead of 2, then the minimum number of processes to cause a deadlock will be 50. ∴ Max no of processes = 50 -1 = 49

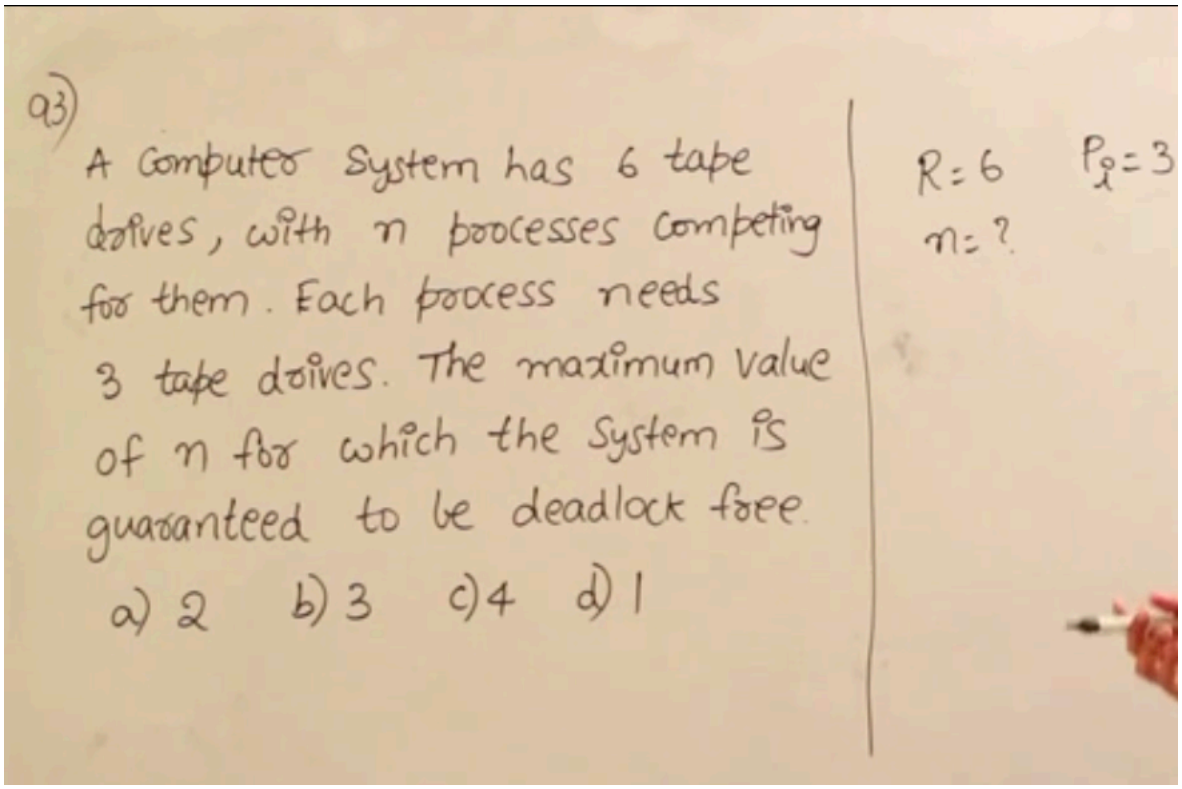


If we have 4 instead of 3, then again the minimum number of processes to cause a deadlock will be 34 (as 3*34 = 102 > 100) ∴ Max no of processes = 34 - 1 = 33



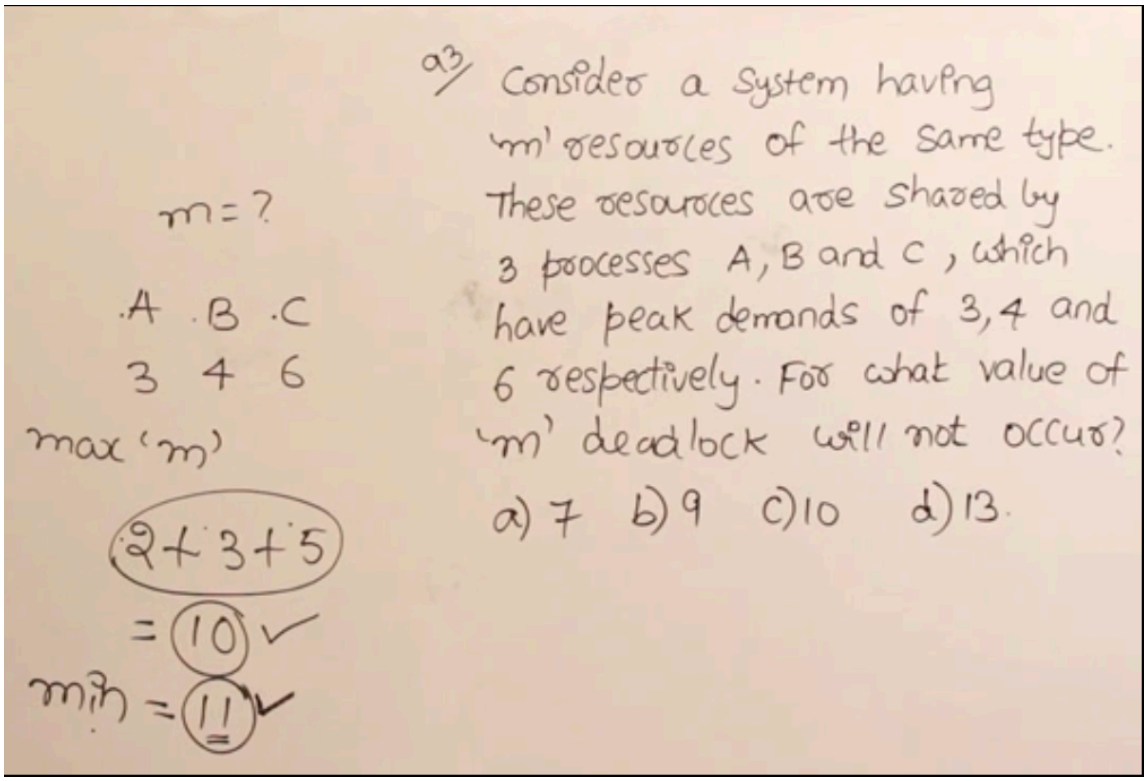
∴ the general formula becomes $n = \left\lceil \frac{R}{P_i - 1} \right\rceil - 1$ where R = number of resources present, P_i is the number of resources each process takes, and n is our answer.

Lecture #3: Gate 1992 Question



Using our previous formula, we get answer as 6/2 - 1 = 2.

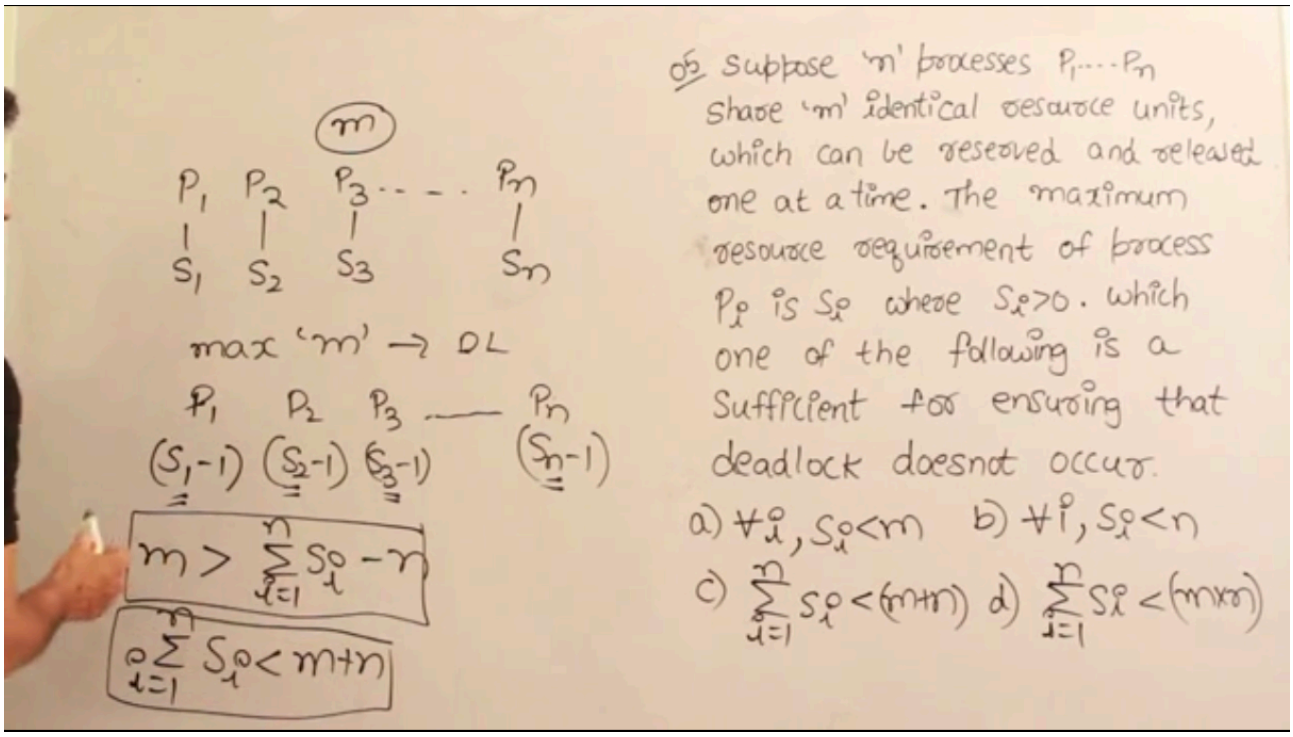
Lecture #4: Gate 1993 Question on Minimum Resources required



From our previous discussion, we found the general formula to be as $\sum_{\forall i} R_i - n + 1$

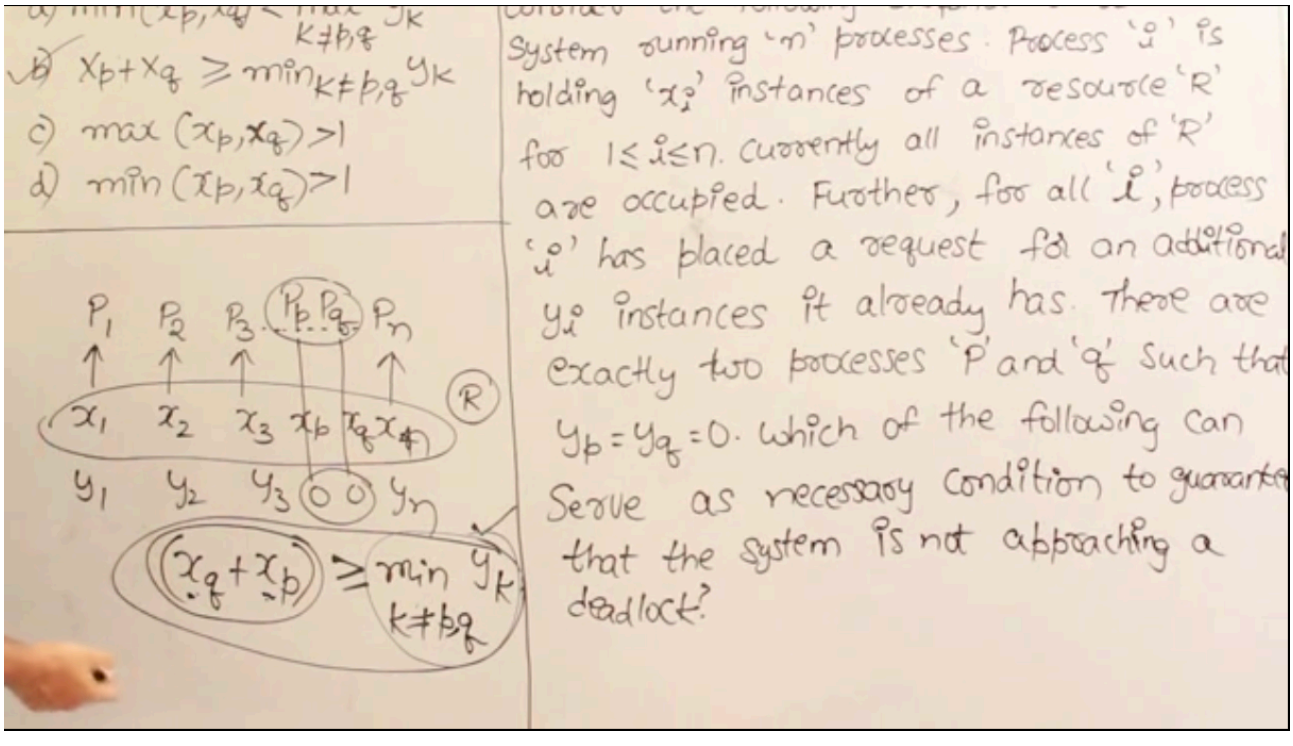
∴ Minimum resources = 13 - 3 +1 = 11.
∴ for any number of resources above 11, deadlock will not. ∴ d) 13

Lecture #5: Gate 2005 Question



Using our derived formula also, we find the answer to be the same.

Lecture #6: Gate 06 Question about necessary condition for Deadlock

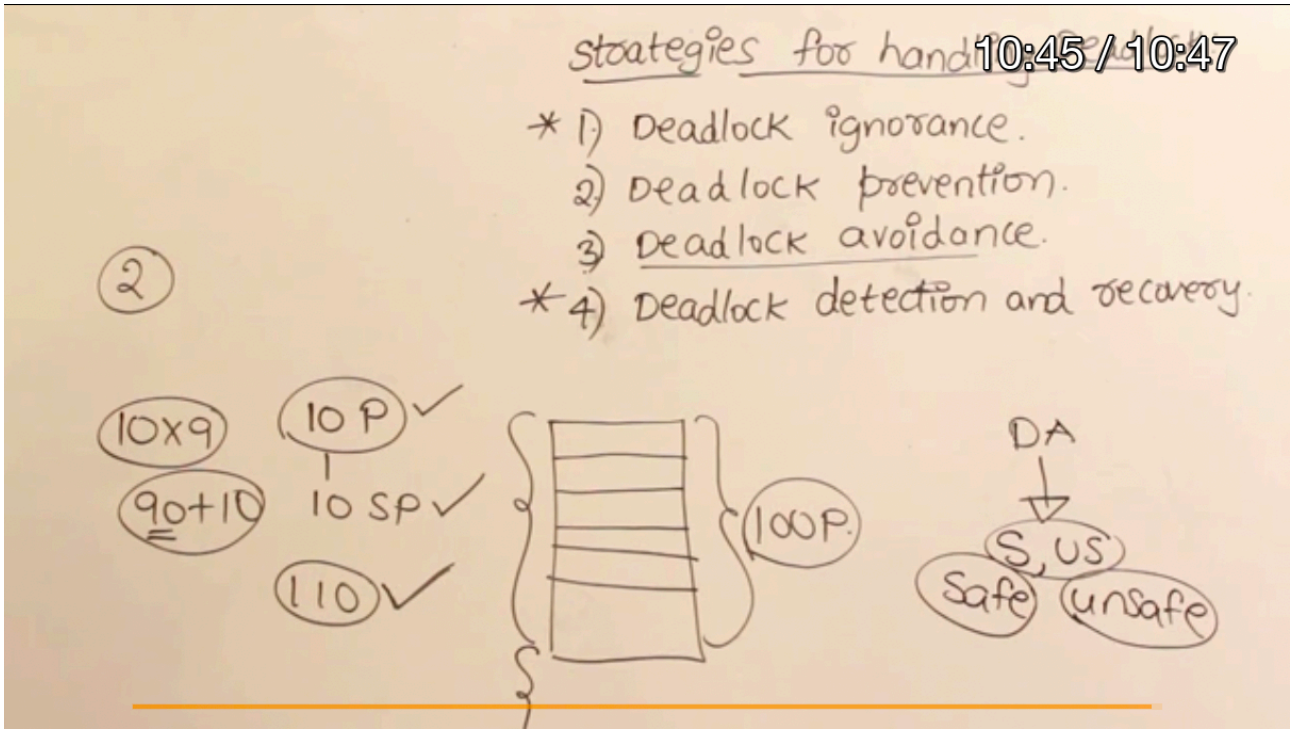


Necessary condition —> Both x_p and x_q added should be able to satisfy at least one requirement of any process

Sufficient condition —> Both x_p and x_q added should be greater than all of the other requirements combined

∴ answer = b)

Lecture #7: Deadlock Handling Mechanisms



Deadlock ignorance —> Ostrich Algorithm

It is a result of trade-off between correctness (mathematics approach) and convenience (engineering approach). Since deadlock is so rare, engineers decided to ignore it as a trade-off for performance. Similar to an Ostrich when a storm arrives, it buries its head in the ground and ignores there's a storm at all.
∴
• System pretends there is no problem
• It is reasonable if deadlocks occur very rarely, as cost of prevention is high
• Example of "cost" is: only one process runs at a time

Deadlock prevention —> Disabling at least one of the four necessary conditions for deadlock

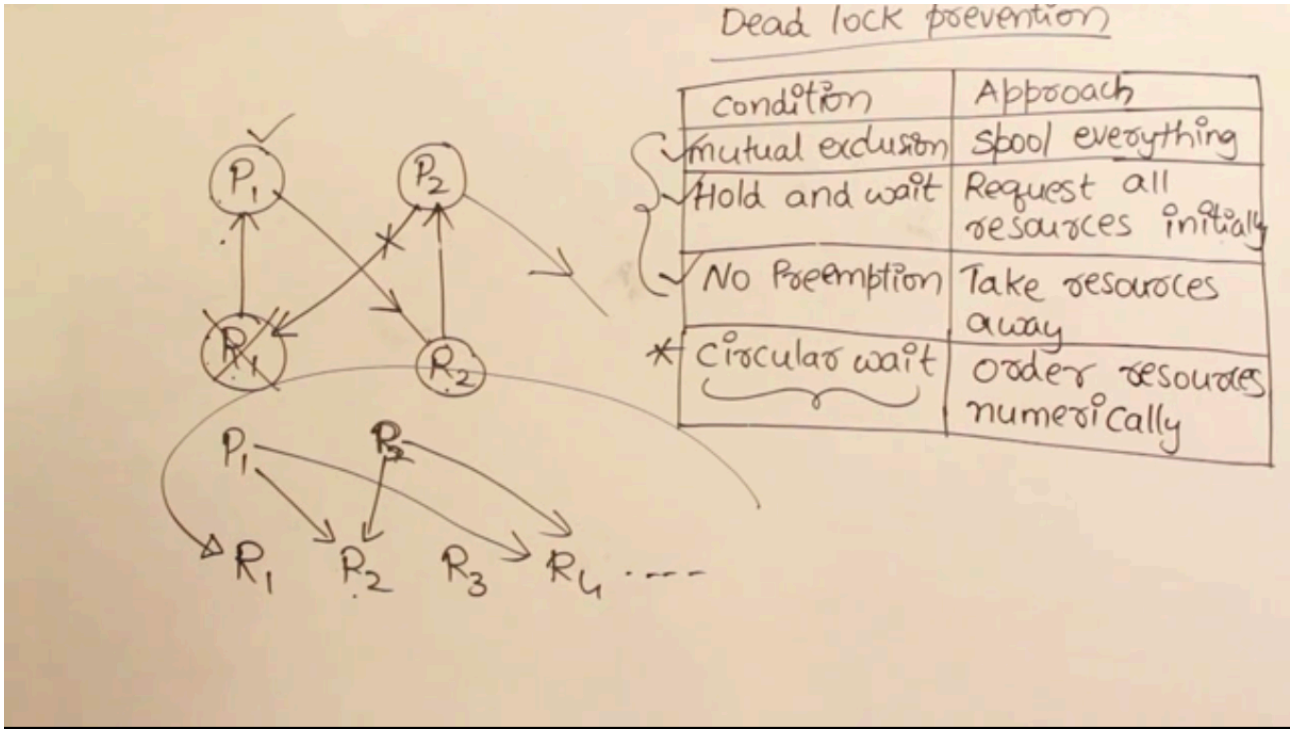
Deadlock avoidance —> Avoiding it at every state (Safe state or unsafe state) (Bankers' algorithm)

Deadlock detection and recovery —> Cycle detection in Resource Allocation Graphs (RAGs)

Out of these, Deadlock ignorance is the most popular deadlock handling mechanism, with deadlock detection and recovery being the second most popular mechanism and also the most practical method.

Lecture #8: Deadlock Prevention

Deadlock is like a chair standing on four legs (the four necessary conditions).
So if we need to break the deadlock, we have to break one of its legs (or conditions) so that it does not stand.



Out of the four possible approaches, only the 4th approach is suggested as it is practical (Requirement of resource need not declared before its execution like other approaches require to be implemented) and does not end up producing inconsistent results.

Lecture #9: Safe, Unsafe, Deadlock avoidance and Bankers Algorithm

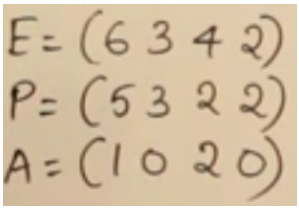
From a state, if we can find out some order in which all the process needs can be satisfied, then such a state is called as safe state.

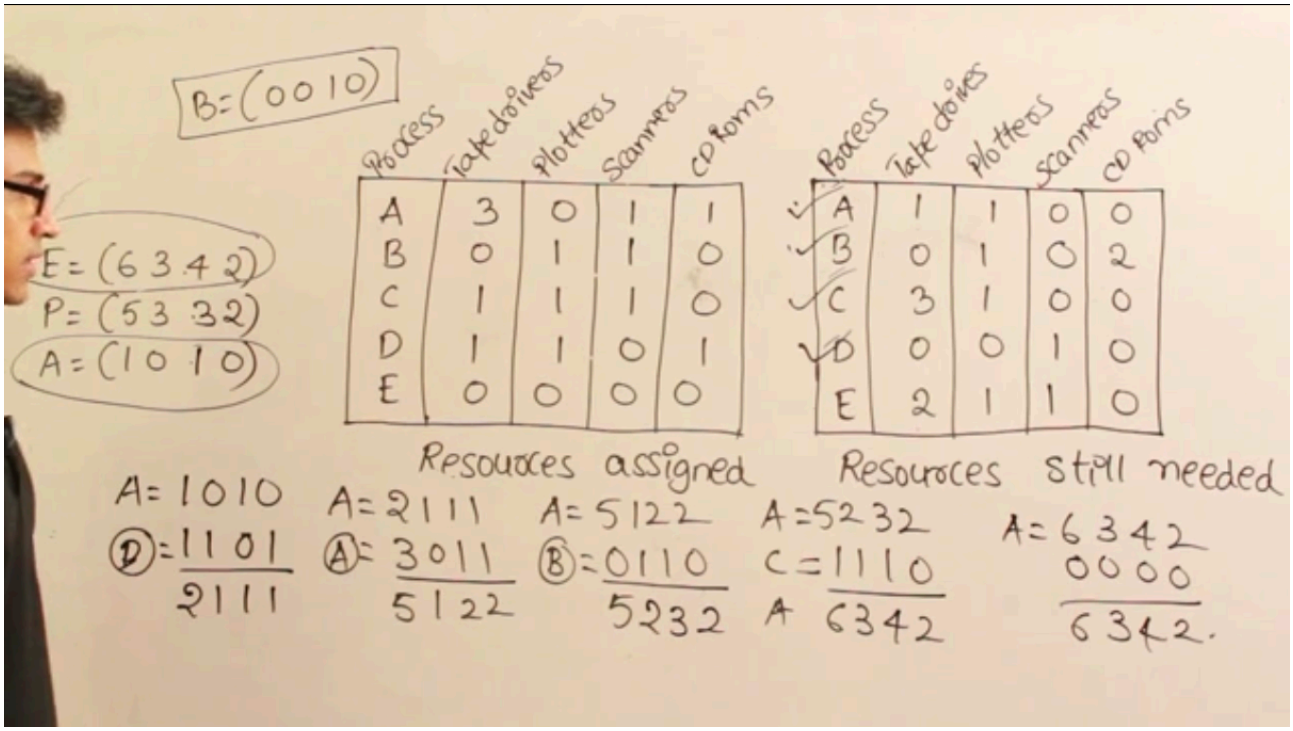
Do not memorise the terminology used as it may change depending on context.

In this terminology, vector E means the total number of resources available in system in total.

P is how many resources are assigned currently to all the processes. It can be calculated by summing the columns of the resource assigned matrix.

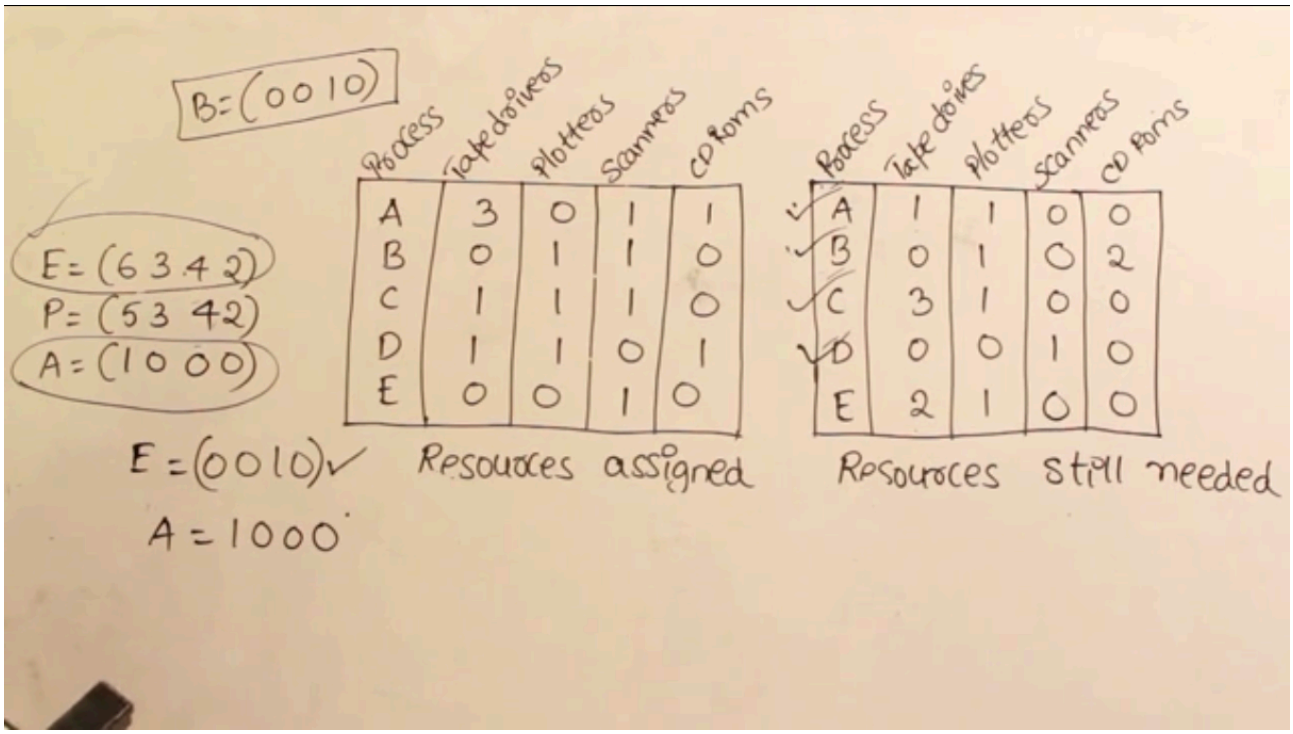
A is how many resources are available. This equal to $E - P$.





A process's resource requirements are dynamic.
Example: -
Even though some processes might require 3 scanners and 2 printers in total, at any given instance, it might not ask for all them and only some of them.

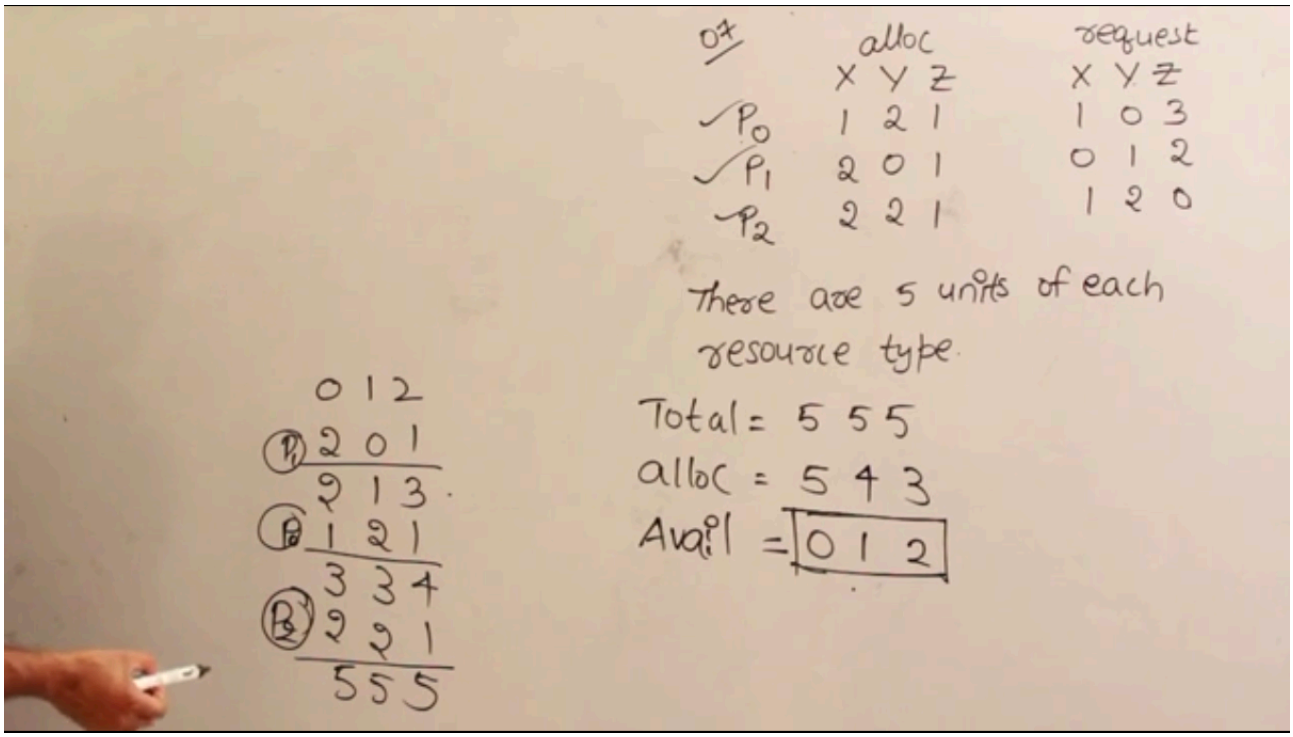
∴ Whenever a process asks from the operating system for a resource, it theoretically calculates whether giving it at the resource its asking for would result in a safe state or not. If it does end up being safe, OS will allocate the resource to the process, otherwise it will delay the request. This is popularly known as deadlock avoidance algorithm or **Bankers algorithm.** It was proposed by Dijkstra.



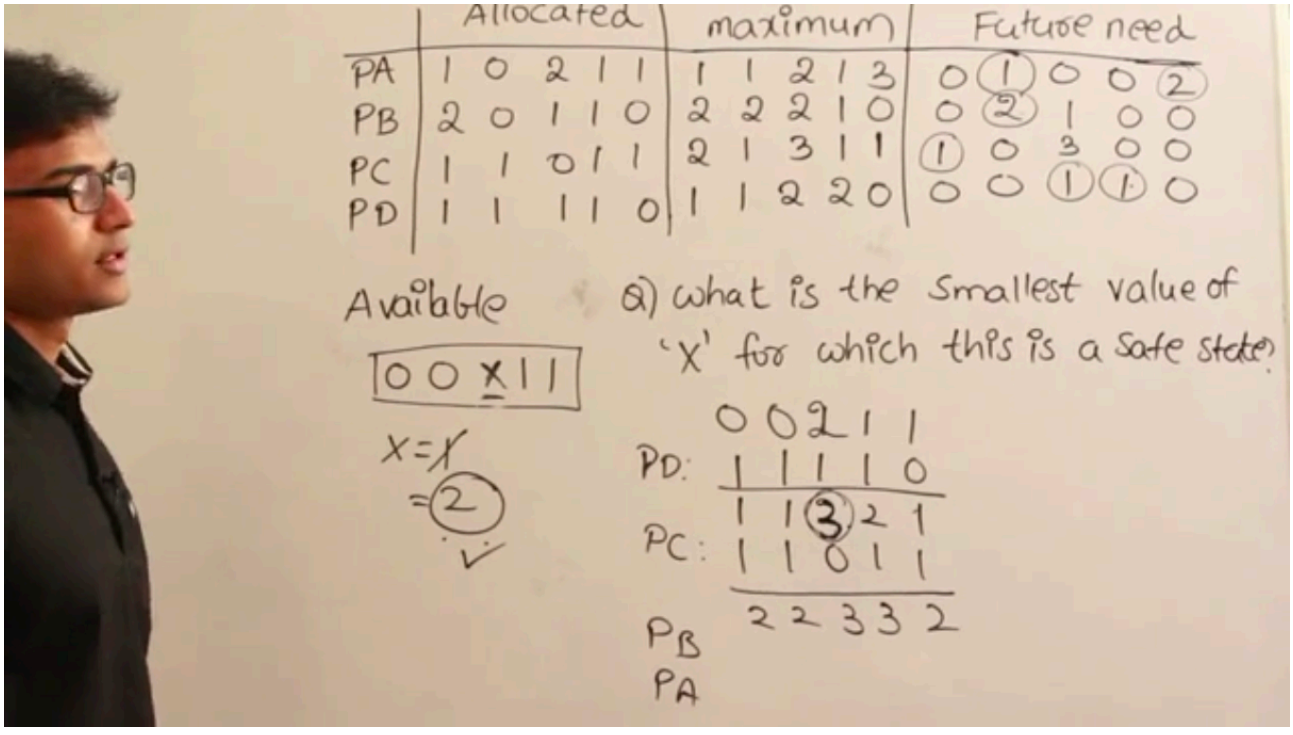
If the available vector cannot satisfy the need of even that process which has the minimum requirement, then there is guaranteed to be a deadlock.
In the above example, When calculating whether giving 1 scanner to E is safe or not, we see that it is not able to satisfy the need of D, the process with least requirement.

∴ The necessary condition for there to be no deadlock is for the available vector to satisfy the need of the least resource requiring process. (Lecture #6)

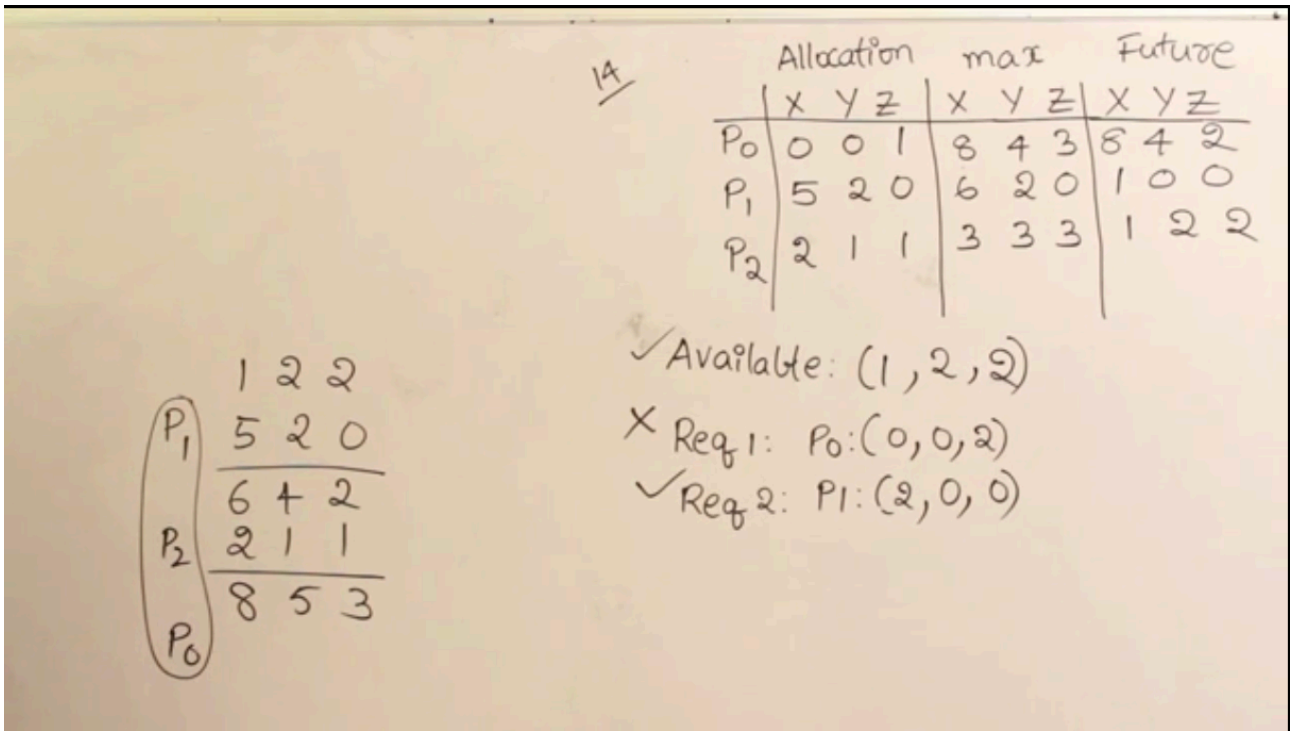
Lecture #10: Gate 2007 Question on Safe State



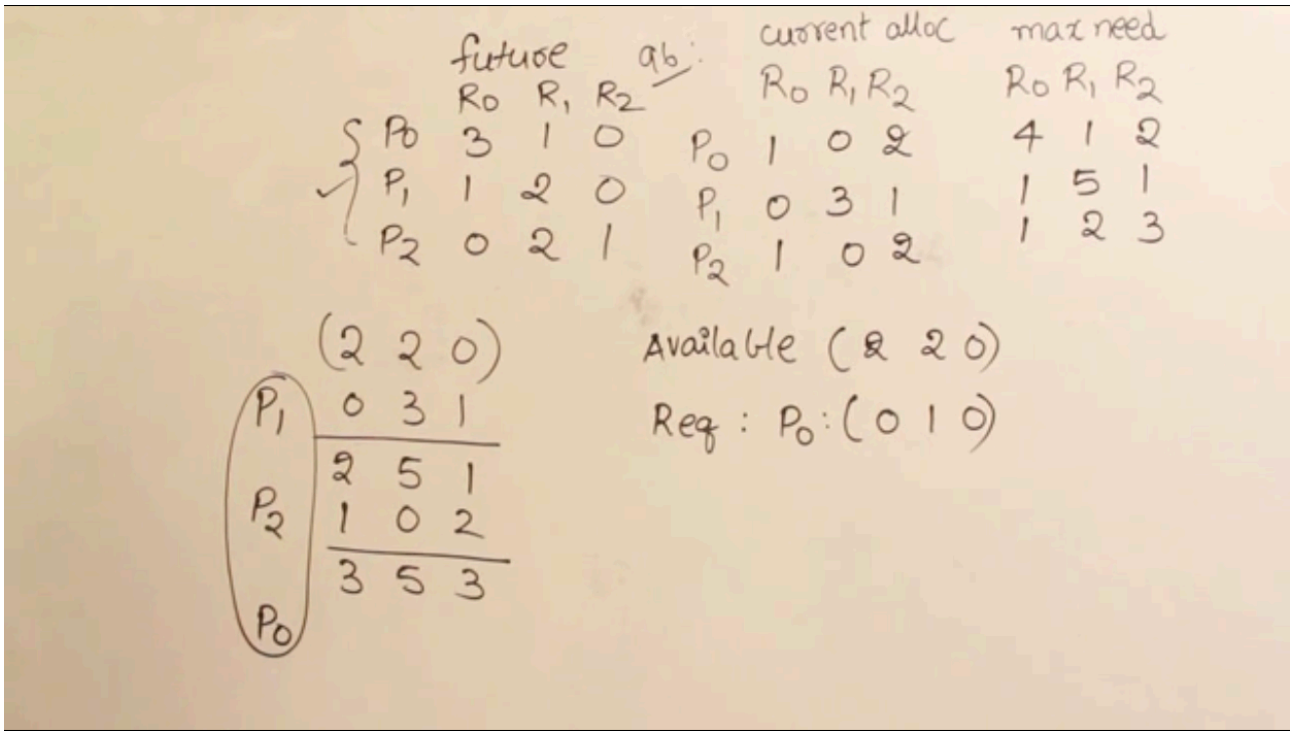
Lecture #11: Question on Safe State



Lecture #12: Gate 2014 Question on Bankers Algorithm

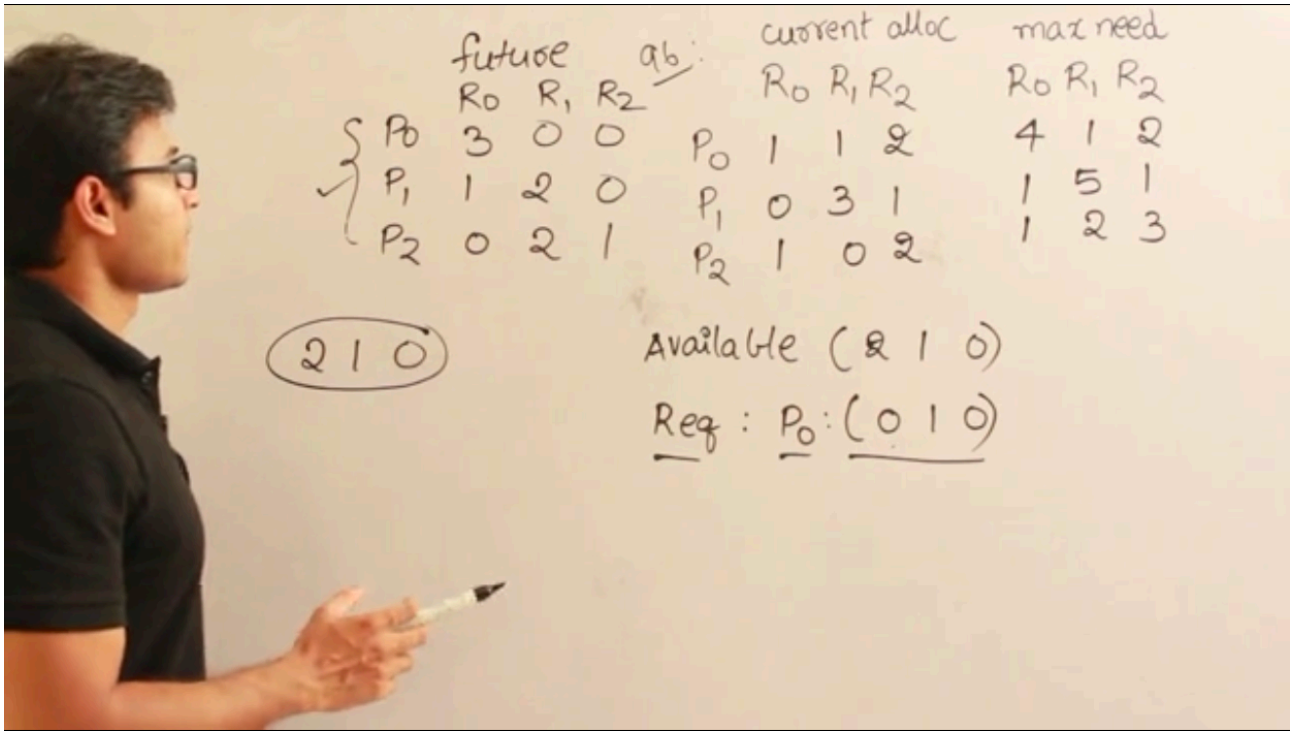


Lecture #13: Gate 2014 Question on Bankers Algorithm

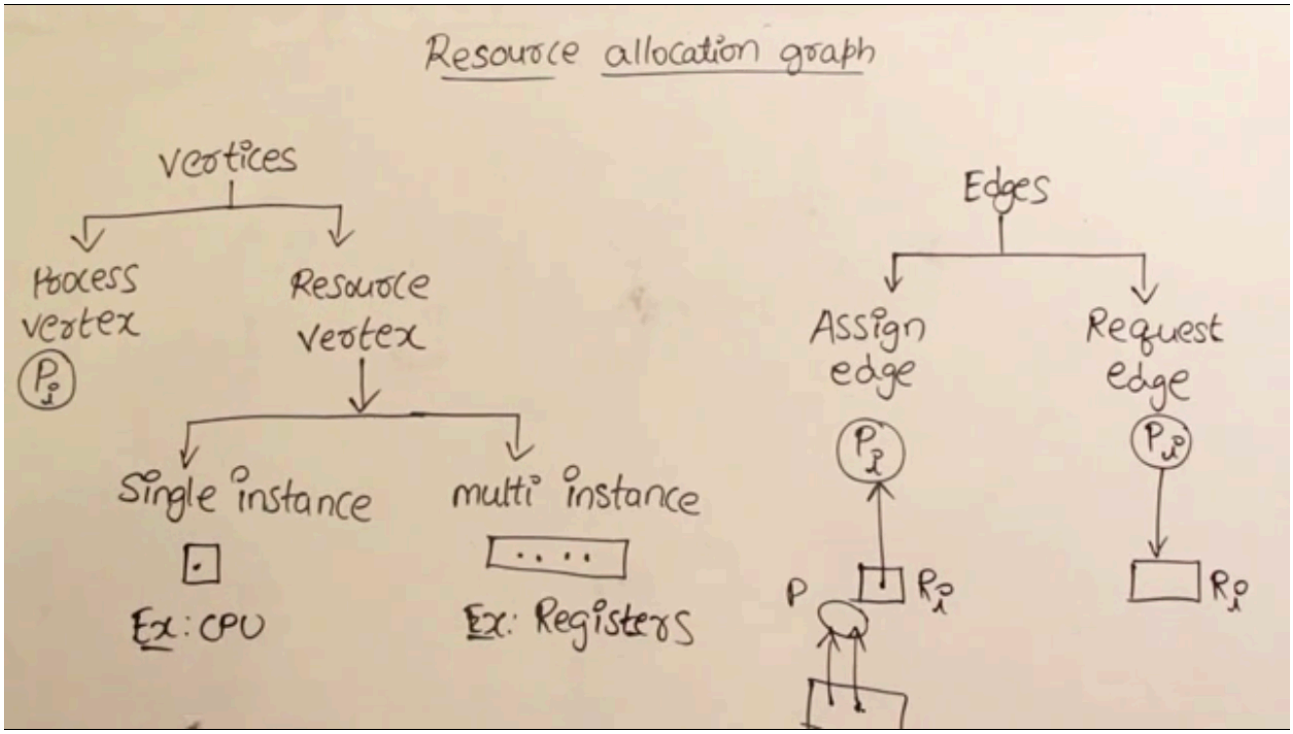


If a state is safe, then there still can be a combination allocation which may lead to deadlock.

∴ Safe state —> A state in which there is at least one sequence of resource allocation which satisfies all the processes' needs.



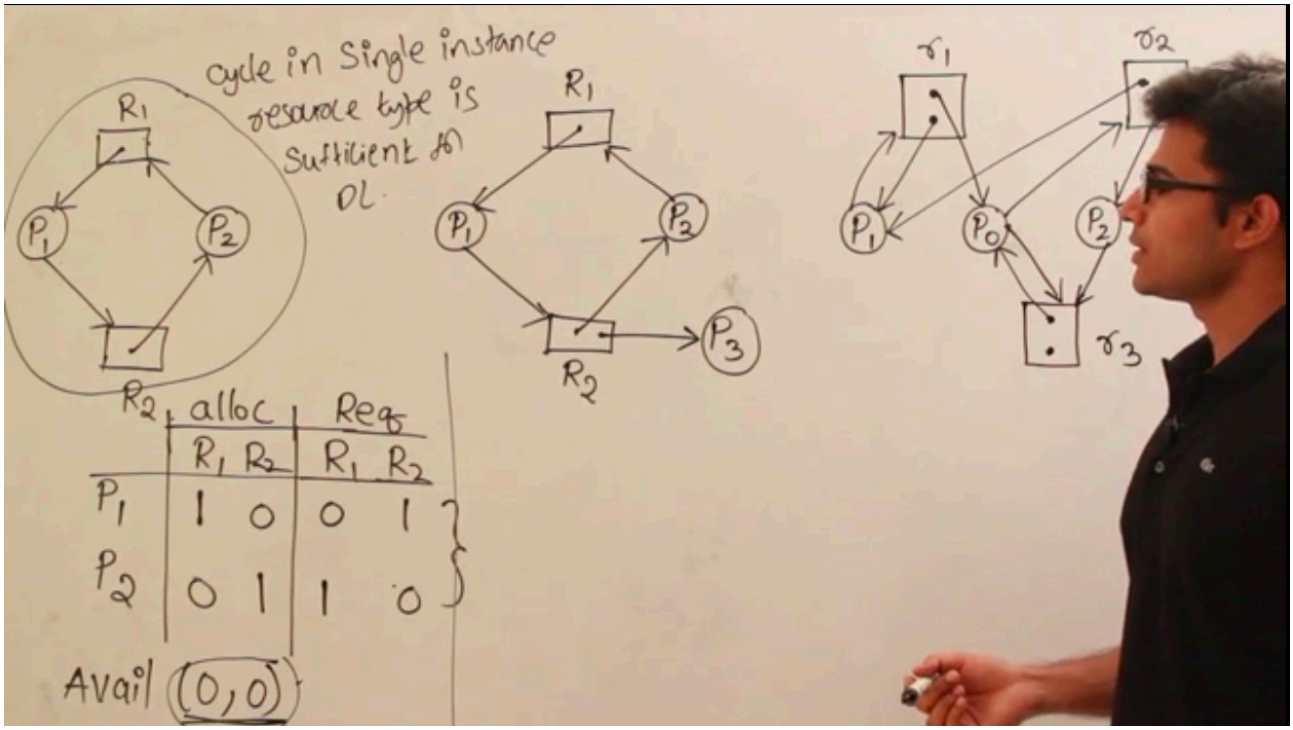
Lecture #14: Resource Allocation Graph



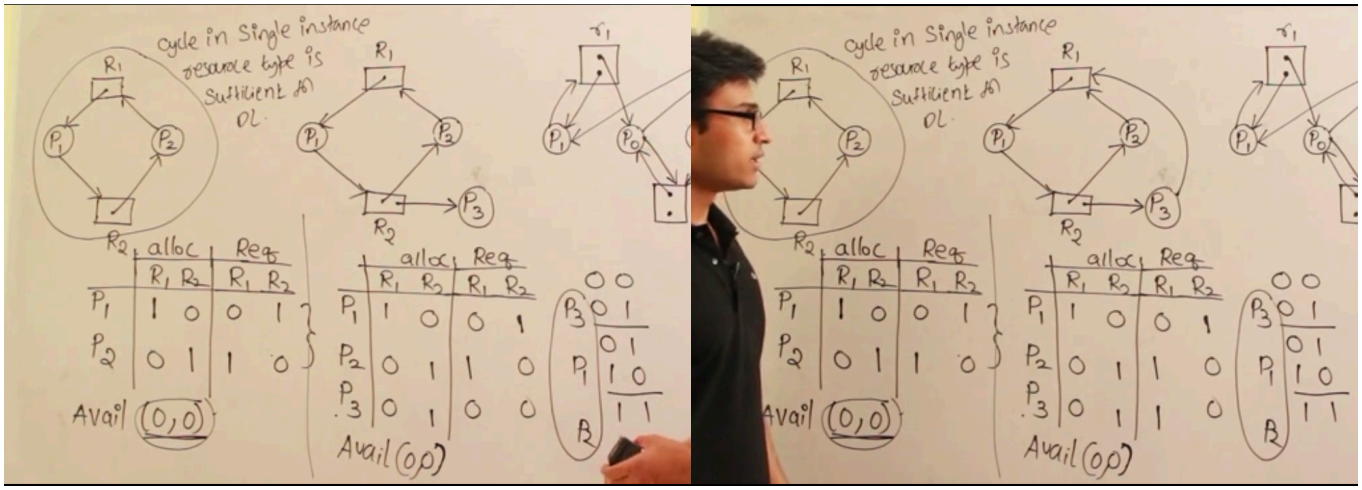
The information which we have been storing in a resource matrix and vectors can also be represented as graphs which is knowns as Resource Allocation Graph (RAG).

The only advantage of representing it as graphs is because diagrams are easier to understand and more intuitive, and *sometimes* when representing it as graph, we may be able to tell if there deadlock or not, which is very useful.

Lecture #15: Resource Allocation Graph Examples

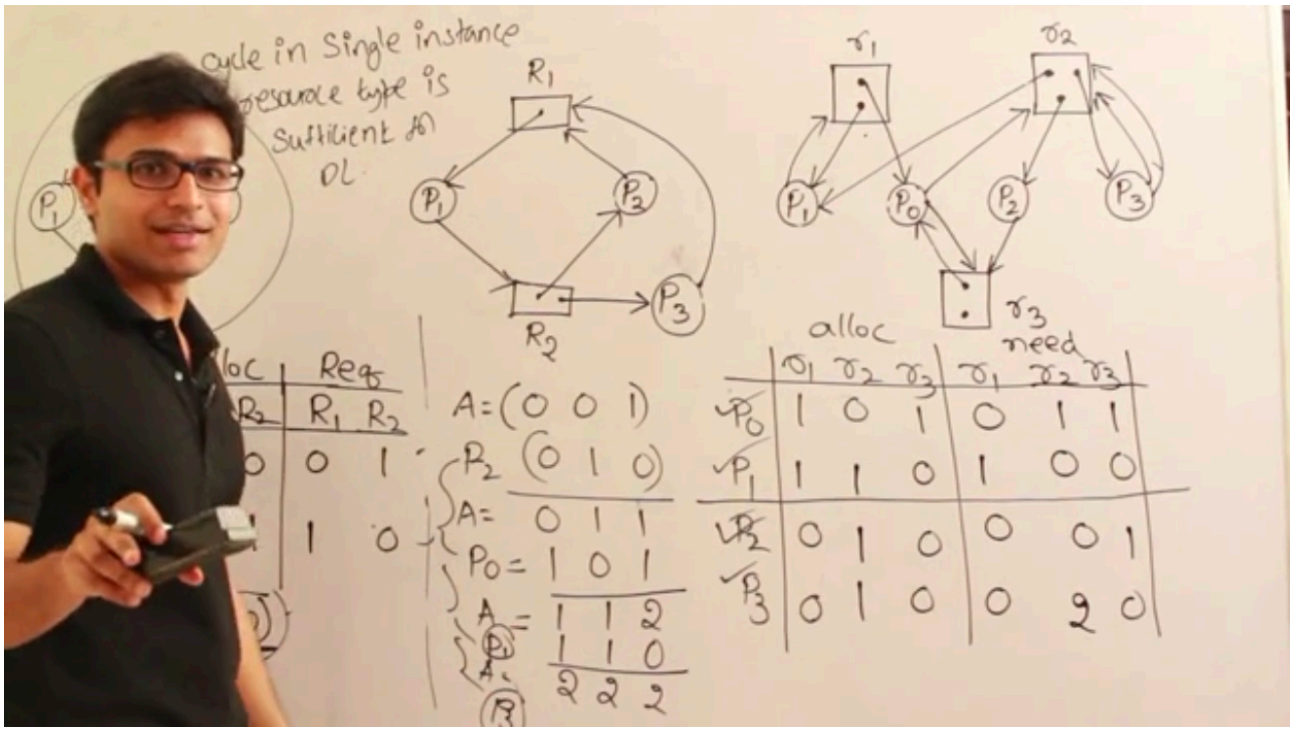


In case of single instance graphs or resource type, if there is a cycle, then there definitely a deadlock. ∴ Cycle is a sufficient condition for there to be a deadlock.



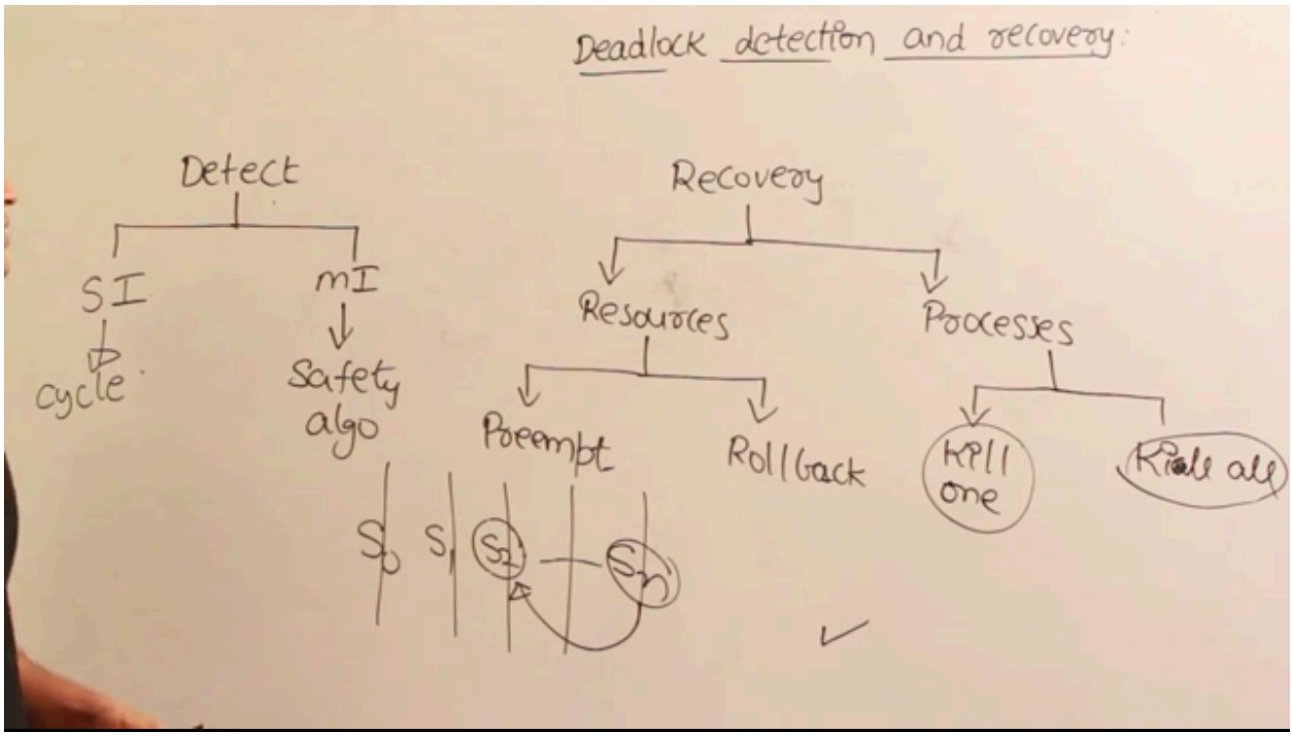
In a multi-instance graph, a cycle may not necessarily mean a deadlock. But for there to be a deadlock, there must be a cycle! ∴ Cycle here (in multi-instance graphs) is not a sufficient condition, but a necessary condition!

—> Gate 1994 Question, asking whether it is safe or not, whether there deadlock or not.



If graphs are confusing, then convert them into table and then answer the question. lol 😂

Lecture #16: Deadlock Detection and Recovery



In single instance graphs, a deadlock can be detected by detecting a cycle in it.
And a cycle can be detected using traversal algorithms like Depth First Search or DFS.

In multi-instance graphs, since cycle isn't the sufficient condition for a deadlock, we use Bankers algorithm or Safety algorithm to detect if there is a deadlock or not.

For recovery, we can either i) Preempt ii) Rollback iii) Kill one process iv) Kill all processes

Preempting process requires manual intervention, and therefore it is not recommended.

Rolling back means changing the state of the system to one of the previous states when it was safe. For this we need checkpointing.

The first two methods just deal with resources. The last two methods which deal with processes is -
Kill one process - meaning killing one process which has been holding up a resource due to which deadlock is being caused. We need to find the process which when killed/removed from the graph will end up breaking the loop causing deadlock. Also, killing the process which has done the least amount of work should be preferable.

Kill all processes - This method will definitely work as killing everything will naturally reset the system to its natural state with no deadlocks, but there is obvious problem with this approach, and that is all the work that has been done will be lost by the processes. So instead of killing all, we would prefer killing just one process