

**Operating System Fundamentals**  
**Prof. Santanu Chattopadhyay**  
**Department of Electronics and Electrical Communication Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 37**  
**Deadlock**

So, next we will look into something called a resource allocation graph. So, this graph will be used very frequently while discussing about problems on deadlock.

(Refer Slide Time: 00:33)

**Resource-Allocation Graph**

A set of vertices  $V$  and a set of edges  $E$ .

$V = P \cup R$   
 $P \cap R = \phi$

- $V$  is partitioned into two types:
  - $P = \{P_1, P_2, \dots, P_n\}$ , the set consisting of all the processes in the system  $\circ$
  - $R = \{R_1, R_2, \dots, R_m\}$ , the set consisting of all resource types in the system  $\square$
- **request edge** – directed edge  $P_i \rightarrow R_j$   $\circ \dots \rightarrow \square$
- **assignment edge** – directed edge  $R_j \rightarrow P_i$   $\square \rightarrow \circ$

So, in this particular; in this particular graph what we are trying to represent is the situation of the system. So, in terms of the processes and resources that are there in the system. So, this any as we know that any graph is defined to be a set of vertices and a set of edges. So, we have got the set of vertices  $V$  and the set of edges  $E$  and  $V$  is partitioned into two types.

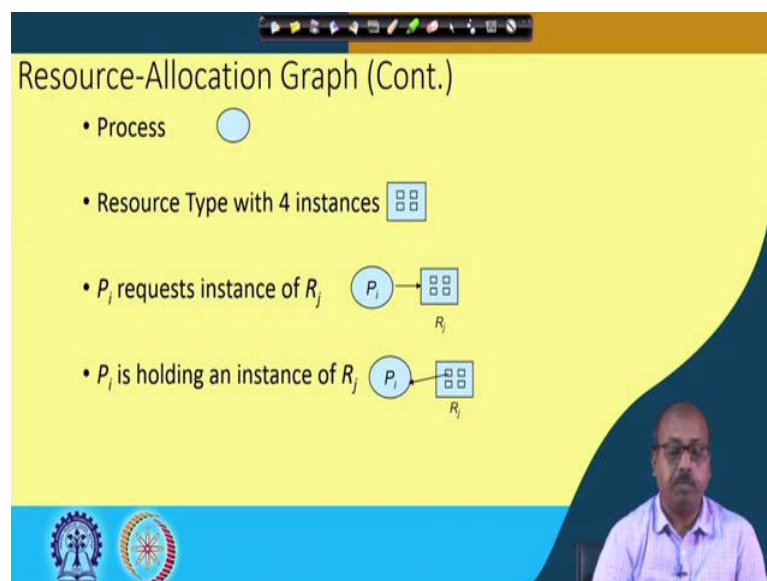
So, there are some vertices which are which represent processes and there are some vertices which represent the resources. So, this vertex set  $P$  is basically consisting of two sets  $V$  is equal to  $P$  union  $R$  such that this  $P$  intersection  $R$  is a null set because they are two distinct set from vertices and processes and resources. So, we have got  $n$  processes  $P_1, P_2$  up to  $P_n$  for making this process set and we have got a set of  $m$  resources  $R_1, R_2, R_m$ .

Now, the edges so, they are also there are two types of edges one is the called a request edge another is called an assignment edge. So, request edge when process  $P_i$  requests for resource  $R_j$ . So, we will have an edge coming directed edge from  $P_i$  to  $R_j$ . So, that is a request edge and we have got an assignment state which is a directed edge  $R_j$  to  $P_i$ .

So, if this resource  $R_j$  is allocated to process  $P_i$  then this edge will be appearing there in the graph, on the other hand if process  $P_i$  is currently requesting for the source  $R_j$  then we will have an edge from  $P_i$  to  $R_j$ . So, we have got two types of edges request edge and assignment edge and we have got two types of vertices which are processes and resources.

So, in some there are different notations like sometimes some notations you will find that this processes they are represented by circles and these resources they are represented by square boxes in the diagram. Similarly this edge is represented by a solid edge and sorry this edge is represented by a solid edge and this is represented by a dotted edge. Of course, this convention is there in some books and some in some places you will find that convention is not followed, but this is quite easy to understand. So, whenever I say that it is at request edge. So, this runs from process to resource and whenever it is an assignment edge it runs from resource to process.

(Refer Slide Time: 03:14)



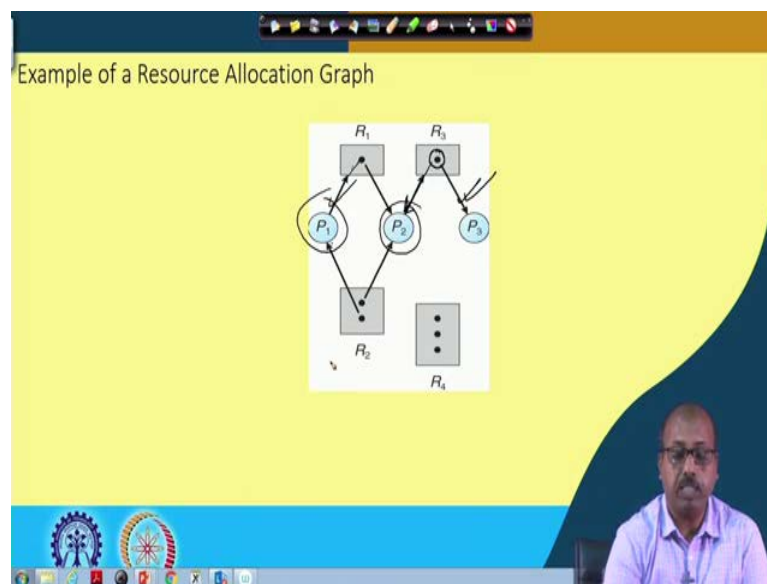
So, processes we will follow this convention that processes they will be represented by this ellipse or circle then resource type with 4 instances. So, it will be represented by a

box and within that we have got 4 rectangles small rectangles. So, they are actually instances of the resource. So, we have got 4 instances of the resource. So, there are 4 boxes.

Now, when  $P_i$  requests for an instance of  $R_j$ . So, all these instances they are equally capable. So, they are capability wise there is no difference. So, any of the instance given to a resource will satisfy its request its requirement. So, we have got an edge from  $P_i$  to  $R_j$  and that edge ends that arrow ends at the boundary of this outer box ok.

On the other hand when it is an allocation that is when  $R_j$ 's one instance of  $R_j$  is allocated to process  $P_i$ . So, from that instance we will have an arrow coming to the process  $P_i$ . So, that is that the allocation edge ok. So, this way we will be representing this resource allocation graph.

(Refer Slide Time: 04:19)



So, this is a typical situation like in this particular case we have got four types of resources  $R_1$ ,  $R_2$ ,  $R_3$  and  $R_4$ . Out of that  $R_1$  and  $R_2$  so they are single instance resources. So, there are; there is only one instance of  $R_1$  and  $R_3$  available.  $R_2$  is a 2 instance resource. So, there are 2 instances of  $R_2$  and  $R_4$ ;  $R_4$  is a 3 instance resource.

So, there are 3 instances of  $R_4$  available and we have got 3 processes. In this particular case  $P_1$ ,  $P_2$  and  $P_3$ ,  $P_1$  is requesting for  $R_1$  and  $P_1$  is currently holding one instance of  $R_2$ .  $P_2$  it is currently holding one instance of  $R_1$  and one instance of  $R_2$  and it is

requesting for one instance of R 3 and one instance of R 3 is currently held by P 3 and P 3 is not requesting anything more. Now is the situation a deadlock situation or not, you see since P 3 is not requesting for any more resource. So, we can assume that P 3 will be set P 3 does not require anything else, so, after sometime P 3 will be over.

So, once P 3 is over so, this resource will become free and at that time I can give this resource to P 2. So, I can give this resource to P 2 and in that case P 2 is already having these one instance of R 1, one instance of R 2 and now it gets one instance of R 3 which was requested for. So, P 2 will be over after some time and once this is over. So, this R 1 instance will become free and so P 1 it already has got one instance of R 2 and this request can also be granted and then P 1 will also be through.

So, this particular situation, so, this is not in a deadlock condition ok. So, this is the situation is quite good. So, system is in a good health we can say. So, this is an example of resource allocation graph.

(Refer Slide Time: 06:18)

The slide is titled "Basic Facts" and contains the following text:

- If graph contains no cycles  $\Rightarrow$  no deadlock
- If graph contains a cycle  $\Rightarrow$ 
  - If only one instance per resource type, then deadlock exist
  - If several instances per resource type, then possibility of deadlock

Below the text is a resource allocation graph (RAG) diagram. It shows three processes: P1, P2, and P3. P1 has one instance of R1 and one instance of R2. P2 has one instance of R1, one instance of R2, and one instance of R3. P3 has one instance of R3. There are arrows indicating resource allocation: R1 from P1 to P2, R2 from P1 to P2, and R3 from P3 to P2. A cycle is shown: P1 holds R1, which is held by P2, which holds R3, which is held by P3, which holds R3, which is held by P2, which holds R1, which is held by P1. A curved arrow indicates the cycle.

The fact is if the graph contains no cycles then there is no deadlock. Like if you look into this in this diagram. So, if you see that if you run a cycle detection algorithm and this edge from the instance of R 1. So, if you take it from this one at the boundary of R 1 then these edge detection cycle. So, it will find and cycle detecting; it is cycle detection procedure. So, it can find you will see that there is no cycle here.

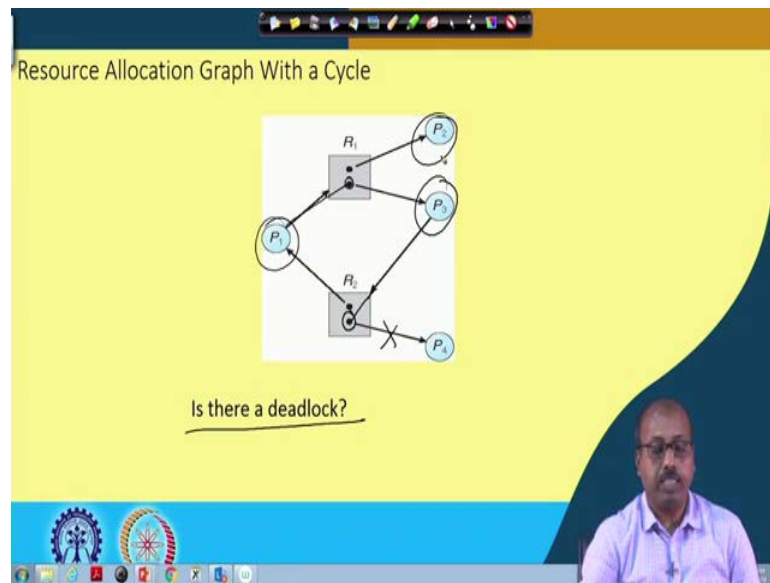
So, this there is no cycle in this graph. So, we can say that there is no deadlock in this system. So, if the graph does not contain any cycle then there is no deadlock. Now if the graph contains a cycle then there can be two situations; one situation is only one instance per resource type then deadlock exists, if there are several instances of resource type then possibility of deadlock. So, what we mean is that if there is no cycle then naturally there is no deadlock because that circular wait condition will be violated. So, there is no deadlock, but if there is a cycle.

So, if there is a cycle. So, then there can be a deadlock. So, it is like this that it is suppose this process P 1 it is asking for resource R 1 then the resource R 1 is currently held by process P 2 and it is asking for resource R 2. R 2 is currently held by process P 3 and P 3 is asking for another resource R 3 which is currently this R 3 is in turn held by P 1. So, if you; so, this is the resource allocation graph where P 1 is holding R 3 asking for R 1, P 2 is holding R 1 asking for R 2 and P 3 holding R 2 asking for R 3.

Now, this type of situation. So, if you run a cycle detection algorithm it will find that there is a cycle here and there is a deadlock because here I we assume that there are only single instances of R 1, R 2 and R 3 that are there in the system. So, that this is a deadlock, but if I have got multiple instances like suppose there are two instances of R 1.

So, even if one instance is given to R 2 the other instance is free. So, these other instance can be given to P 1. So, if there are multiple instances then it may or may not be a deadlock situation. So, that is why it is say there are possibility of deadlock. So, that may be there. So, if there is only one instance of resource then deadlock does exist if there is a cycle and if there are multiple instances then it is a possibility of deadlock. So, deadlock. So, we have to do more rigorous check to see whether there is any deadlock or not.

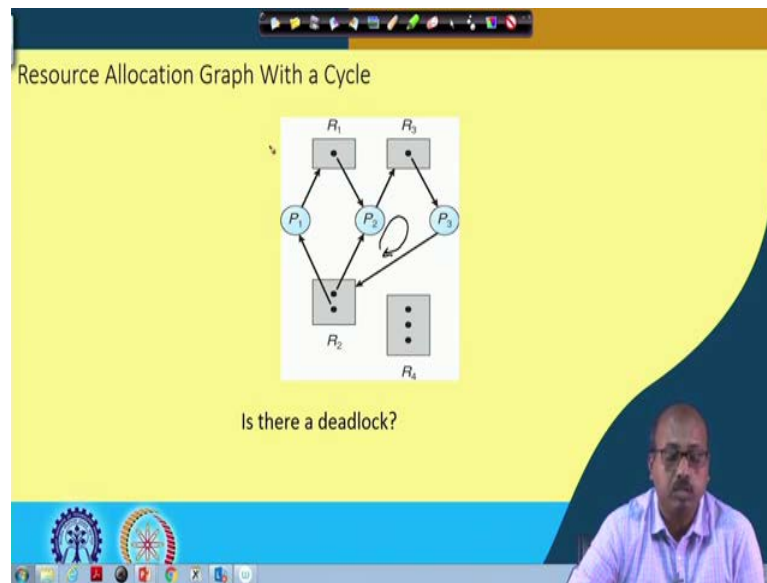
(Refer Slide Time: 09:08)



Now, in this particular case you see that we have got a cycle ok. So, this cycle is there. If you run a cycle detection algorithm in this graph so this is a cycle, but this is not a deadlock situation because I can always give this instance to P 3 and so sorry I can. So this is not there. Now to answer this question is there any deadlock now. So, we have to see like whether we can satisfy all the requests. So, P 4 is currently holding this particular instance of resource.

So, once P 4 can be is satisfied. In fact, so after some time this resource this edge will not be there. So, at that time this resource will be free and it can be given to P 3 ok. So, if it is given to P 3 then P 3 will be over so, this resource will become free; this resource will become free then P 1 can be given this resource and then P 1 will also be over and P 2 will also be over. So, like that you see this particular graph there is no deadlock. So, though there was a cycle; there was a cycle like this, but there is no deadlock here.

(Refer Slide Time: 10:30)



So, existence of cycle does not mean a deadlock always, but let us see what about this thing, what about this graph. Now you see can I do any of these processes can I satisfy any of these processes. So, you see this P 3; for P 3 so, P 3 is holding this R 3, but it requires one instance of R 2 also, but both of them are unavailable one of them held by P 1 another held by P 2.

Similarly P 1 it is holding one instance of R 2, but it is requiring one instance of R 1 which is not available. Similarly P 2 it is holding one instance of R 1 and it is asking for one which is holding one instance of R 1 one instance of R 2 also, but it is asking for one instance of R 3 for completion. So, P 2 cannot also proceed and there is a cycle; so, we have got a cycle here.

So we have got a cycle around the nodes R 2, P 2, R 3, P 3, R 2. So, there is a cycle. So there is a cycle and they here there is a deadlock also. So compared to the previous case so there was a cycle, but there was no deadlock, but in this case there is a cycle and there is a deadlock. So, that has to be judged more carefully. So judging this multi instance situation, so that is slightly more complex compared to single instance cases.

(Refer Slide Time: 11:55)

**Methods for Handling Deadlocks**

- 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
- Ignore the problem and pretend that deadlocks never occur in the system; used by many operating systems, including UNIX

State of a system  
Consider all processes for their requests and allocations

$P_1$	$R_1, R_4, R_5$	$R_2, R_3$
$P_2$	$R_2, R_5$	$R_1, R_4$
		}

Now, how can we handle this deadlock. So, one way for handling deadlock is that to ensure that the system will never enter into a deadlock state. So by this, so first of before doing that you need to define what is the state of a system. So state of a system; state of a system so this is a situation. So if you take this individual processes and look into their. So, consider all processes and you look into their resource requests and current allocation, consider all processes for their requests and allocations.

Now so at one point of time say maybe the process P 1 it is holding the resource R 1, R 4 and R 5 and it is asking for the resource R 2 and R 3 maybe process P 2 it is holding the resource R 2 and R 5 and it is asking for some resource R 1 and R 4. So this way if we just list down all the processes in terms their current holding of resources and current request for resources. So that will constitute one state of the system.

Now, this state may be a deadlock state as we have seen previously that if we cannot satisfy all the processes their requests then given time up to infinity. So that situation so, that is a deadlock situation. So, that type of those states are deadlock states. Now when a system initially starts, so, there are no processes in the system.

So, they are only the initial process created by the operating system is there. So, that is a deadlock free state. So, at that point there is no deadlock. So, starting from the initial state of the system so as the new processes are created they are requesting for the



resource and all. So, this from this initial state we come to a new state where some processes have been created and their requests have come.

So, this way it will as and when a new process requests for a resource or a process releases a resource this system state changes in terms of this process requests and allocations. So, that way it goes through a number of state transitions goes through a number of state transitions. Now if we know that out of the whole if this is the total state space of the process or of the system out of that may be a subset of this total state space. So, this is the deadlocks they are all deadlock states; they are all deadlock state.

So, we initially started at say this state at the initial state. Now if we find that we are at a state from where there is a possibility of a transition to a deadlock state then if we can stop the system at this point. So, I will not do anything, so that the system can come to a state from where it may go to a deadlock. So, we have to somehow ensure that this does not happen. So, you will be doing a deadlock prevention and deadlock avoidance. So, that I will not it will not go into deadlock.

So, this is one possible way of handling it. So, it we will ensure that we will never enter into a deadlock state or we can allow the system into a deadlock state and then recover. So, we find that the system has reached a deadlock state in one of the states here and then we do a recovery. So, we detect this situation and do a recover it is a recovery essentially means we want to we have to terminate some process or we have to; or we have to do something. So, that we can with some of the resources are taken back from the processes forcefully. So like that we have to do something. So that we come out of this deadlock state.

So allow the system to enter into a deadlock state and then recover from that. The third alternative is known as the Ostrich algorithm which says that ignore the problem and pretend that deadlocks never occur in the system. So this algorithm is known as Ostrich algorithm because this ostrich in the desert how do they survive when there is a desert storm.

So actually just lies down on the sand itself. So after some time that desert storm stops and then it again comes up and goes. So like that if there is a deadlock. So while depending on the frequency of occurrence of the deadlock and the cost of this deadlock

like in a telephone exchange if there is a deadlock then what can happen is that many calls may get stuck in between. So, if that can be tolerated then fine.

So, maybe the system, the telephone operator or the system administrator will reset the system. We said the telephone exchange and then it will be sorted out, but resetting the telephone exchange means all the calls that are going on and all other real time activities that are going on so they will get disrupted.

So, that may or may not be tolerable. On the other hand on a small computer system where some users some say students programs are running even if there is a deadlock maybe we just reboot the system and the students they resubmit their jobs. So, that can happen.

So, depending upon the situation so, we may not do anything for deadlock because we will see that this deadlock detection and recovery. So, they are they do not come free of cost. So, we have to spend CPU time and resources system resources for doing those checks and recovery mechanisms. So, one possibility is that ignore that there is a possibility of deadlock and if we find that the frequency at which this deadlock occurs. So, if deadlock is very infrequent then of course, resetting the system does not cause much harm.

So, this way we can have some ignore mechanism and we pretend that deadlocks never occur in the system and if the system the administrator finds that the system throughput is very very low even if there are a large number of processes in the system. So, that may be because of the deadlock condition. So, the system administrator may kill some of the processes or may reboot the system. So, that these deadlock gets eliminated from the system. So, that is fine.

(Refer Slide Time: 18:47)

**Deadlock Prevention**

- Ensure that at least one of the necessary conditions for deadlocks does not hold. Can be accomplished by restraining the ways request can be made
- **Mutual Exclusion** – Must hold for non-sharable resources that can be accessed simultaneously by various processes. Therefore cannot be used for prevention.
- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources
  - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none allocated to it.
  - Low resource utilization; starvation possible

Handwritten notes:  $P_1: R_1, R_2, R_3$ ,  $P_2: R_2, R_4$

Now, how do we do this prevention of deadlock. As we said that there are four necessary conditions for a deadlock to exist. So, if you can violate any of them, then this deadlock can be resolved. The first one is the mutual exclusion condition. So, it says that must hold for non shareable resources that can be accessed simultaneously by various processes therefore, cannot be used for prevention.

So, this mutual exclusion so, if you can violate the mutual exclusion. So, if you say that these processes they will be allowed to access the resource simultaneously. So, as I said that there are several resource type in the system  $R_1, R_2, R_3$  etcetera.

Now, this so, some of these resources are preventable resource. Sorry some of these resources are shareable resource, some of the some of their non shareable resource. So, if it is non shareable resource then only this question of mutual exclusion comes. So, if they are shareable resource then I can allow multiple processes to access this resource simultaneously.

So, if  $R_2$  is a shareable resource then multiple processes they can access this resource simultaneously as I was telling for example, the computer screen. So, their multiple processes can write onto the screen simultaneously maybe in different regions in different windows and all, but so, that is possible. So, for non share; for shareable resource this mutual exclusion can be violated, but of course, there are certain resources which are non shareable in nature and for them we cannot violate the mutual exclusion

problem; so, mutual exclusion issue and so, for them we cannot use this mutual exclusion for deadlock prevention.

Second one is hold and wait. So, hold and wait means the a process while requesting for some resource it already holds some resources, a process maybe it is P 1 is holding resources R 1 R 2 and R 3 now it is asking for R 4. So, how can I violate this type of requirement? So, we must guarantee that whenever a process requests a resource it does not hold any other resources. So, if this thing can be guaranteed.

So, how can we guarantee this? Maybe we know that in a process 1 it requires the resources R 1, R 2 and R 3 in it is entire lifetime it requires the resources R 1, R 2, R 3. P 2 on the other hand in it is entire lifetime requires the resources R 2 and R 4. So, if this is the complete code for P 1 so, at the beginning itself; at the beginning itself it grabs all these resources.

(Refer Slide Time: 21:31)

**Deadlock Prevention (Cont.)**

- **No Preemption** – not practical for most systems
  - If a process A that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held by A are released
  - Preempted resources are added to the list of resources for which the process is waiting
  - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting
- **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration. Can be used in practice.

So, in the initial part of the code so, it does that; in the initial part of the code. So, here actually this resource request allocation is done. So, only if the all the resources are available and it can be allocated to P 1 then only P 1 will come into the actual computational part and the usage of the resources. So, all the recover; all the processes they must put their request at the beginning of execution.

So, this is one possibility, but of course, the difficulty with this particular approach is that we are maybe we are requiring these resources much later in the system and may be that all these resources are not required simultaneously also. so, but even in that case so, these resources are held by this process for the entire lifetime of the process. So, that way there is a gross under utilization of the resources.

Other possibility is that allow processes to request resources only when the process has none allocated to it. So, if so they; so, it cannot be the case that at present process 1 has got R 1 with it and now it is asking for R 2 that cannot be there. If it wants to the request for R 2 first it has to release R 1 and then request for R 1 and R 2 together ok.

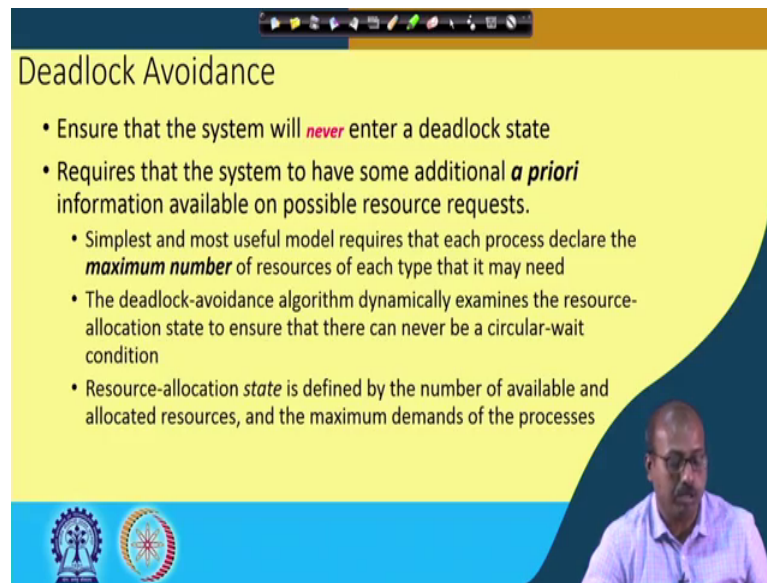
So, that is one possibility. So, it is like this that here it requests for R 1 and here it requests for R 2. So, what it has to do it has to release at R 1 at this point and then request for both R 2 and R 1 again. So, that if it gets both of them then it will proceed now the problem is that. So, releasing R 1 in between maybe other process. Some other process will grab this resource R 1 and it will not get it immediately. So, that way this process will have to suffer and whether it is possible to release R 1 in between, so, that is also an issue.

For example if R 1 is a printer then you cannot release it in between. So, this type of problems are there, but if it is depending on the type of resource so, if it is possible that we can violate this hold and wait condition then this deadlock will not occur, deadlock can be prevented.

So, problem is that low resource utilization and starvation is possible because I said that this all the resources required by the process they should be requested at the beginning itself. Now it may so happen that a process requires large number of resources and as a result maybe so all the resources are not available simultaneously.

So, there are many other processes in the system that actually holds one or other of these resources. As a result this process never gets a chance for execution because of this low resource utilization. So, this is the hold and wait condition so, if you can violate this hold and wait then it is fine.

(Refer Slide Time: 24:29)



The slide is titled "Deadlock Avoidance" and features a yellow background with a blue wave-like shape on the right side. At the top, there is a navigation bar with various icons. The main content consists of a bulleted list of points. In the bottom right corner, there is a small video inset showing a man with glasses and a mustache, wearing a light blue shirt, speaking. At the bottom left, there are two logos: one of a gear and a person, and another of a circular emblem with a star.

### Deadlock Avoidance

- Ensure that the system will *never* enter a deadlock state
- Requires that the system to have some additional *a priori* information available on possible resource requests.
  - Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need
  - The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition
  - Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes

So, another thing is that the system will never enter into a deadlock state. So, requires that a system to have some additional a priori knowledge about this system that is. So, another condition that is there for this deadlock avoidance is a deadlock prevention is the no preemption method. So, no preemption so, we cannot if it is possible that a resource is given to a process and that resource can be taken back in between.

So, as I was telling that if process 1 is holding some resource; if process 1 is holding the resource R 1 and it is if I find that process 2 also request for R 1 process 2 also request for R 1. Now if we think that I will give R 1 to P 2 then I have to take back R 1 from P 1 first. So, if this is a non preemptive resource then that is not possible.

For example, if it is a printer which is a non preemptable resource I cannot take it back from the process. On the other hand if it is say CPU so, CPU time (Refer time: 25:39) say that the process can be taken off from the CPU. So, that way if no preemption can be violated then of course, deadlock cannot occur because I can take the resource from one process and give it to others so, that those processes complete and then only the resources are given to the original process.

So, if your process A that is holding some resources requests another resource that cannot be immediately allocated to it, then all the resources currently held by A are released. So, this is the situation like process A has requested some resource and it is found that the resource is not available; since without getting the new resource process a

cannot proceed further. So, as I, so, we still that if since you cannot proceed any more you give me back all the resources that you are currently holding.

So, if these resources are released from process A, this preempted resources are added to the list of resources for which the process is waiting that is there and process will be restarted only when it can regain its old resources as well as the new resource it is requesting. So, the situation is like this. So, I have got this process A; process A is currently holding the resource say R 1, R 2 and R 3 and now it is requesting for a new resource R 4.

Now, R 4 is not available. So, as a result what is done this R 1, R 2, R 3. So, they are also taken back e from process A. So, this R 1 R 2 R 3 they are also taken back from the process A and for process A we note that it requires all these resources R 1, R 2, R 3 and R 4 for progressing further. Now these resources since they are free now they have been taken back from a and they are all preemptable resource that is why they could be taken back and this R 1 R 2 R 3 they are taken back from A.

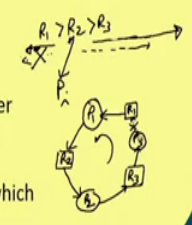
So, some other process will utilize which was maybe some other process B which was holding this R 4 and there that was also requiring this resources R 1, R 2, R 3. So, thus process B will get a chance for execution getting all these resources and it will eventually release R 4. So, later on this R 1, R 2, R 3, R 4 this entire bunch can be given to process a for completion.

So, if we have got this preempted preemptable resource then this I can then these resources can be preempted from the process and then they can be put into the free a pool of free resources for allocation to other processes and this particular process we put all these resources on the request list. So, process will be restarted only when it can regain all it is old resources as well as the new ones that it is requesting and the circular waiting. So, circular waiting, so, it will impose a total ordering of all resource types and require that each process requests resources in an increasing order of enumeration.

(Refer Slide Time: 28:42)

### Deadlock Prevention (Cont.)

- **No Preemption** – not practical for most systems
  - If a process A that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held by A are released
  - Preempted resources are added to the list of resources for which the process is waiting
  - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting
- **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration. Can be used in practice.



So, this particular strategy it what it tells is that we have got this resources, but this resources have got some order. So, if I have got say resources R 1, R 2 and R 3 we make the order like this R 1 is greater than R 2 greater than R 3. Now any process say P I, if it is currently holding the resource say R 2; if it is currently holding the resource R 2 then it can request for resource which are down the line. So, it can request for the resource a R 3, R 4 etcetera, but it cannot request for resource R 1 ok. So, it cannot request for the resource which are of say, greater or which are also lesser then it is type.

So, it can be either way so, it can be so, P i is restricted to access resources on the lower side or the resources on the higher side. So, either side is blocked. So, either it can go it can request for resources in this side of the chain or it can request resources in this side of the chain. So, if we do not allow this one then it can request only for this thing. Now if P 1 is holding say if we have got this type of ordering and P 1 is currently having R 1 with it. So, P 1 is currently having R 1 with it then it can request for R 2 and R 3. So, it can put a request for R 2.

Similarly P 2; so, P 2, so, this P 2 is currently holding R 2. So, P 2 as per this chain, so, it can request for R 3 it can request for R 3, but it cannot request for R 1. So, this P 3 is currently holding R 3 and as a result as per this chain, so, it cannot request for R 1 and R 2. So, we cannot say that we cannot it cannot happen that P 3 also request for this R 1. So, that cannot happen because P 3 is holding R 3. So, it cannot request for R 1 because



R 1 is of higher value than R 3. So, this way this change so, this cycle cannot be there. So, this circular weight cannot happen. So, in this edge cannot exists. So, the circular weight cannot happen. So, this deadlock can be prevented.

So, violating any of these conditions it is possible. So, we will see many say computer system say it is possible to order the resources and accordingly we can do this deadlock prevention. We will continue with this in the next class.