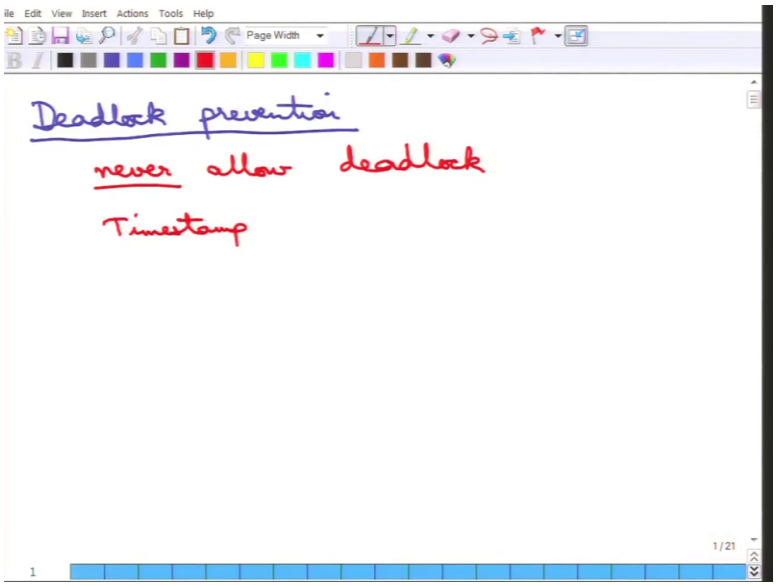
Fundamentals of Database Systems
Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 43
Concurrency Control: Deadlock Prevention and Deadlock Detection

We will continue with the Concurrency Control protocols, as we have already seen many protocols, locks and timestamps based ordering protocol and so on, also the multiple granularity locking. But, if you have noticed in all of them, there is one recurring problem which is the deadlock. So, most protocols can fall into this trap of deadlock, so the basic time, the basic to pale protocol can fall in the trap of deadlock.

But, the timestamp ordering protocols cannot of course, but the locking protocols can always go into this deadlocks stay. So, next what we will be studying is to see how to handle this deadlock.

(Refer Slide Time: 00:51)

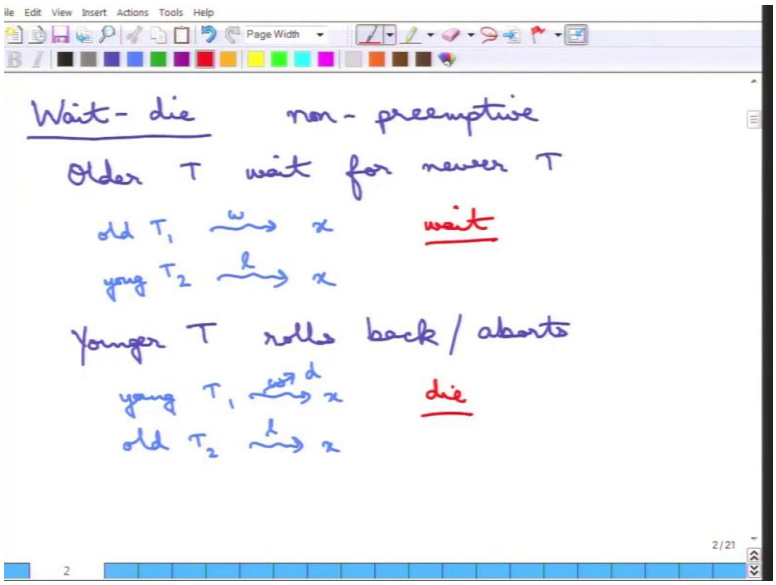


So, the first thing is the deadlock prevention that we will study. So, what I mean by deadlock prevention is that, this type of scheme, the deadlock prevention schemes never allow a system to enter into a deadlock. So, they never allow deadlock, so if these schemes are run, the system will never deadlock. So, we have already seen an example of this, the timestamp based ordering protocols are of course, deadlock prevention schemes.

So, the timestamp based, the protocols that use the timestamp, the basic timestamp ordering protocol and the validation based protocol, they are we have seen that these are, this deadlock prevention scheme, because they never allow deadlock, then there are two as... So, what happens in that is that, so in this deadlock prevention schemes is essentially… In the timestamps based ordering protocol what happens is that, when a transaction or when an operation of a transaction appears out of order, then one of them must roll back.

If you remember in that read timestamp while we are checking the read timestamp and the write timestamp, there are two ways. We said either the transaction T that is trying to write it should roll back or the transaction T 1 or T 2 for which it is not correct should roll back. So, now,... So, which one should roll back that is a question. So, whether it is T or which is it is T 1 or T 2 that is the thing, so for that there are these two schemes.

(Refer Slide Time: 02:21)



So, the first one is called a wait die scheme. In a wait die scheme, this is also, this is a non pre emptive scheme and we will see what does a non pre emptive mean, but just remember from the time being that this is a non pre emptive scheme. What happens is that, here the older transactions wait for newer transactions. So, suppose there is an old transaction, the T 1 that wants a particular lock on an item x and T 2 which is a younger transaction. So, T 1 is old and T 2 is young, T 1 is currently holding that lock.

So, then the older transaction simply waits for the young one to finish, so the old waits that is fine. So, if the older comes after the younger and the T 2 is already holding the lock and the older one wants to it, it just waits. The younger ones; however, do not wait, if a younger transaction wants this thing wants a lock, they simply roll back. So; that means, the younger ones are do not wait for the older ones.

So, if the younger one comes and sees that an older one is already holding the lock, it simply dies, so that is why it is the thing. So, this is what is happening. So, the younger one, young one is holding, is waiting for a lock it does not wait for a lock, it essentially just dies. Because, and older transaction is currently holding the lock, so this is why it is dies. It just rolls back or aborts which is the same as abort so; that means, it is dies and this is why this is called a wait die scheme, so this is the thing.

And the younger ones as you can see can die many, many times, because what happens is that when a transaction rolls back or dies or aborts, whatever is the terminology, it restarts with a new timestamp. So, when a young one restarts, it restarts with of course, a younger timestamps, so the chance that if will die again is more. So, it can just keep on dying again and again and this is called non pre emptive, because nobody removes any other transaction.

So, the old transaction simply waits for the young one to finish and the young ones simply dies. So, nobody pre empts another transaction, so nobody hurriedly removes another transaction, so that is why this is called a wait die transaction.

(Refer Slide Time: 04:59)

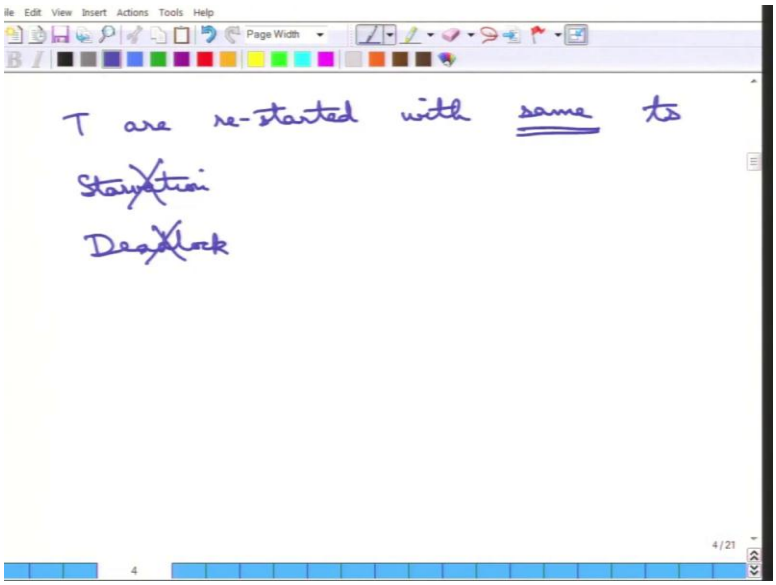
So, the other thing in this is the wound wait scheme, this is pre emptive and again we will see what does pre emptive mean later. But, let us see what happens is that the older transactions kill newer ones, so this is interesting, so kill young transactions. Now, what does this mean? So; that means, that suppose there is a young transaction T 1 which is holding the data item x and an old transaction T 2 wants the data item x. So, at that point since the old transaction has come, the young is holding to it, the young is made to release the lock and the young is essentially it is, it dies.

So, the young one dies and the old one essentially now gets the lock and continues with this. So, the young one is made to die, so that is why it is the older transactions kill young ones that is the thing. And what happens to the younger ones? The younger ones simply waits, so an old transaction is holding the lock and the young transaction comes, it simply waits. So, the younger one is simply waiting, so in the terminology the old one is set to wound the young one. So, it stabs the young one, it kills the young one whatever is the thing that is why it calls and the younger one dies. So, this is it wounds it, so that is why it is called a wound wait protocol.

Now, this is pre emptive. Why is this pre emptive? Because, when an older one comes and it sees, it wants the lock which is held by young one, it pre empts the young one, it forcibly aborts the young one, it forcibly removes the lock from the, it forcibly makes the younger one, unlock it. So, it removes the named data item whatever the data item from

the young one, it makes it, release it and it preemptively grabs the data item, even though the old ones holding it, so that is why this is a pre emptive data item. Now, transactions in this case what may happen is that, transactions may be started with the same timestamp.

(Refer Slide Time: 07:22)



So, transactions are restarted, it may be restarted with a younger timestamp, with a new timestamp or restarted with the same timestamp, so this is the other thing. Now, one can show that there is no starvation in this schemes, there is no starvation. Why is that? If the transactions are started with the same timestamp, why is that, is that at some pointing time, if the starvation… When the starvation happens is that, remember that when a transaction wants a data item and it cannot gets it, the next time it wants it is, by the time some other transactions have got it and so on and so forth.

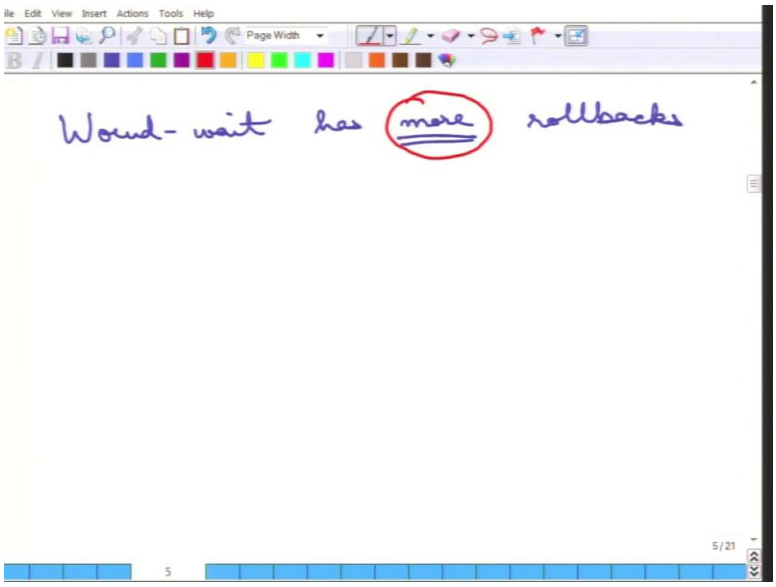
Now, if a transaction is restarted and it is restarted with the same data item, at some pointing time this transaction must be the oldest one that is around. Because, every new one, every other one has got it and so on and so forth, every other older transaction has finished and this is what is being starved. So, then it will be getting it, it will wound all the young ones and it will getting it, so this there will be no starvation.

On the other hand, the same thing what the happens for the wait die is that, at some pointing time this transaction must be the oldest one. So, it must simply get it, because it will wait for the young ones to finish and another transaction which is also wants it will

not be getting it, because it is younger than this, so this is the older one. So, and the end it will get it, so there is no starvation and as we have already the understood that there will be no deadlocks. So, this is free from starvation and this is free from deadlocks.

Now, among these two things which is the better one as it happens that in the wait die transaction the young ones keeps on dying. Because, every time it is restarted there is some old one that is being done. So, young one reaches to this place, but the old one is already holding the lock and it dies it restarts and it reaches the same place and it dies. So, it keeps on happening, so it dies many, many times.

(Refer Slide Time: 09:29)



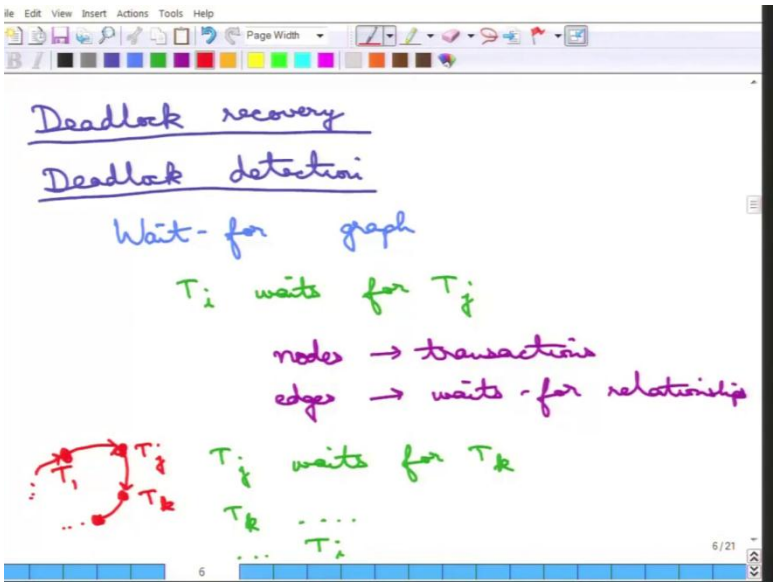
So, in that case essentially the wound wait has more roll backs. So, wound wait protocol, the wound wait scheme rather has more roll backs than the wait die scheme. So, among the two things the wait die is preferred. So, between the two schemes the wait die and wound wait the question is which one is more preferable. Now, what happens in the wait die is that young transactions may die many times, now what does it happen is that when a young transaction reaches to a point where it wants a lock it looks for an old it sees that there is an old transaction that is holding to it, so it dies.

And it is restarted by the time it restarts reaches the same point it may be happening that the old transaction is again holding it, so it again dies. But, the chances of that the old transaction finishing is larger that is in the wait die in the wound wait what happens is that the wound wait, the older transactions kill the younger ones. So, if a old transaction

some very, very old transaction that was supposed to happen much earlier comes it will kill all the young ones.

So, even though the younger transactions die many more times in an wait die and older transaction may kill more transactions in a wound wait. So, in general wound wait has more roll backs. So, what is preferred is the wait die scheme? So, this is about the deadlock prevention we can also talk something about the deadlock recovery.

(Refer Slide Time: 11:00)



So, deadlock prevention schemes will never go into deadlock, so there is no question of recovery. But, this locking schemes they do not they are not deadlock prevention schemes, so they may go into deadlock. Now, the question is once a system goes into deadlock it must be detected and it must be recovered. So, the system must not keep on waiting of course, it must be the deadlock must be broken. So, how is the way to be done?

So, the first thing to recover from a deadlock is to identify the deadlock. So, that is called the deadlock detection. So, how is a deadlock detected? This is very similar to the to some concept that we read earlier anyway. So, the concept that it uses is something called a wait for graph, so this is a wait for graph and we in the conflict serializability we saw graph, this is kind of the same thing and what happens is that in this wait for graph suppose T i waits for T j that what does this mean, the T i waits for T j is that T j is holding a lock on an item that T i wants.

So, then in the graph... So, the graph the nodes are transactions and the ages are according to this wait for things, so the ages are the waits for relationships. So, in the above case when T i waits for T j there will be a node T i then it will have a directed edge to node T j. Now, what happens in a deadlock is that T i waits for T j then T j waits for T k, then T k waits for something else and finally, that something else waits for T i that is why it is a deadlock.

So, now, you can clearly see what is going to happen in this graph is that now T j waits for T k then T k waits for something else, then something else, something else that waits for T i. So, that is the… What does that mean is that, for this graph to say whether there is a deadlock or not, there must be a cycle in the graph. So, only when there is a cycle in the graph it can be detected that there is a deadlock. So, the how does one detect if there is a cycle in this directed graph that is the same as we did in earlier in the conflict serializability graph, etcetera.

So, one can run a depth first algorithm to find out if there is a graph, if there is a cycle then there is a deadlock. Because, in that cycle all the transactions are waiting in a circular manner, if there is no cycle on the other hand then there is no deadlock. So, it is a, this thing.