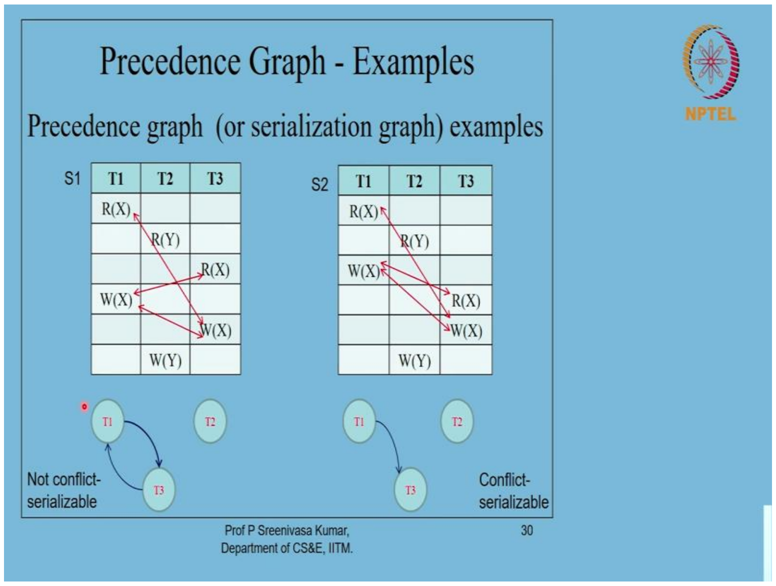
Database Systems
Prof. Dr. Sreenivasa Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Concurrency Control using Locks
Lecture – 38

So, we had last lecture seeing how to represent operations of a bunch of transactions that are going together and we call that as a schedule and then looked at what is a notion.

(Refer Slide Time: 00:43)



Precedence Graph - Examples

Precedence graph (or serialization graph) examples

S1

T1	T2	T3
R(X)		
	R(Y)	
		R(X)
W(X)		
		W(X)
	W(Y)	

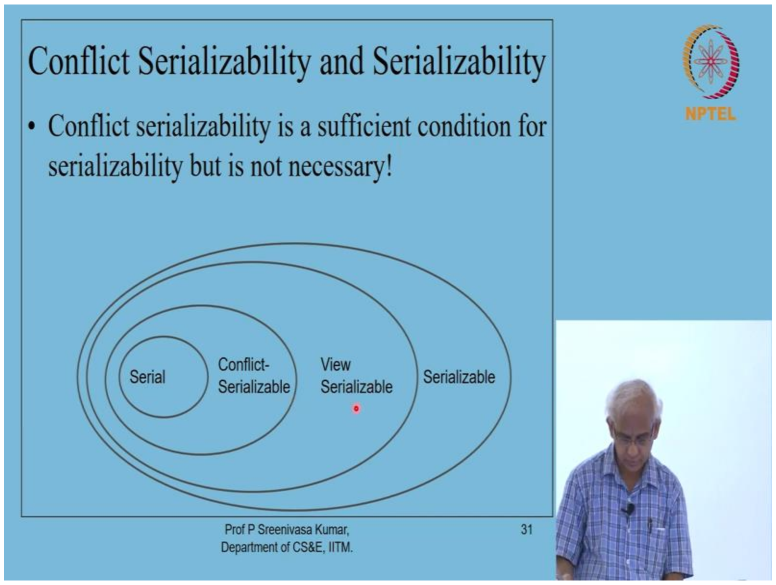
S2

T1	T2	T3
R(X)		
	R(Y)	
W(X)		
		R(X)
		W(X)
	W(Y)	

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 30

What is this notion of serializable schedules? So serializable schedules will actually give us a good way of running the transactions because the serializable schedules if one can show that they can actually avoid all this problems of problems that arise due to transactions trying to access the same database item and then trying to update. Now, we also seen how to detect whether a particular schedule is a serializable schedule or not, complex serializable schedule or not. So, to actually complete that discussion, I need to also introduce one more notion of serializability.

(Refer Slide Time: 01:44)



Conflict Serializability and Serializability

- Conflict serializability is a sufficient condition for serializability but is not necessary!

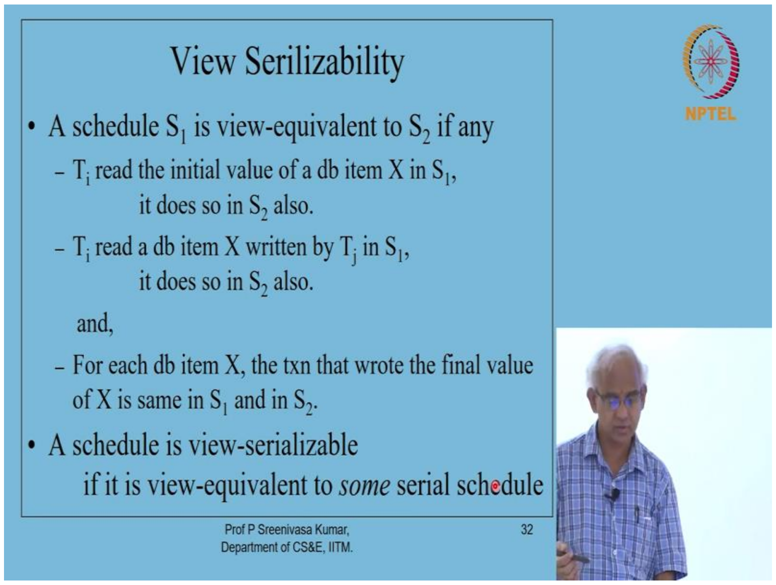
Serial
Conflict-Serializable
View Serializable
Serializable
NPTEL

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 31

Called view serializable. So, there is a so we will actually not go into much more detail about this, but this is the you know, general notion of serializable schedules that means these are somehow equivalent serializable. Serial schedule says this class of schedules that are there inside this, conflict serializable schedules are larger bunch of schedules, larger set of schedules.

You can see why this is the largest schedule because serial schedules do not allow you to do any interleaving of operations at all, whereas here in conflict serializable schedules, we do allow interleaving of operations of schedules. And so you get a larger bunch of no bigger set of permutations of operations. Now, there is also one more notion called view serializability and that is actually offers you a larger chunk of schedules compared to conflict serializable schedules. So, let me define that in the next slide.

(Refer Slide Time: 03:05)


View Serilizability

- A schedule S_1 is view-equivalent to S_2 if any
  - T_i read the initial value of a db item X in S_1, it does so in S_2 also.
  - T_i read a db item X written by T_j in S_1, it does so in S_2 also.

  and,
  - For each db item X, the txn that wrote the final value of X is same in S_1 and in S_2.
- A schedule is view-serializable if it is view-equivalent to *some* serial schedule

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 32

So, schedule S 1 is in a similar way to conflict equivalence we have defined view equals. So, schedule S 1 is view equivalent to S 2 if these conditions satisfied. So, if any transaction T i read the initial value of the db item X in S 1 then it also does so, in this other schedules S 2. So, in both S 1 and S 2 if T i is the one that read the initial value of some database item if some transaction has read, it will be the same in both S 1 S 2.

So, for every db item what the transaction the reads it is first in same in both and if your transaction T i reads a db item X written by some other transaction T j then the same thing happens in S 2 also in the same relative order, then for each of these database items with the transaction that wrote the final value for the X is again going to be the same in both S 1 S 2. So, if this is the case we call a schedule S 1 as view equivalent to schedule S 2.

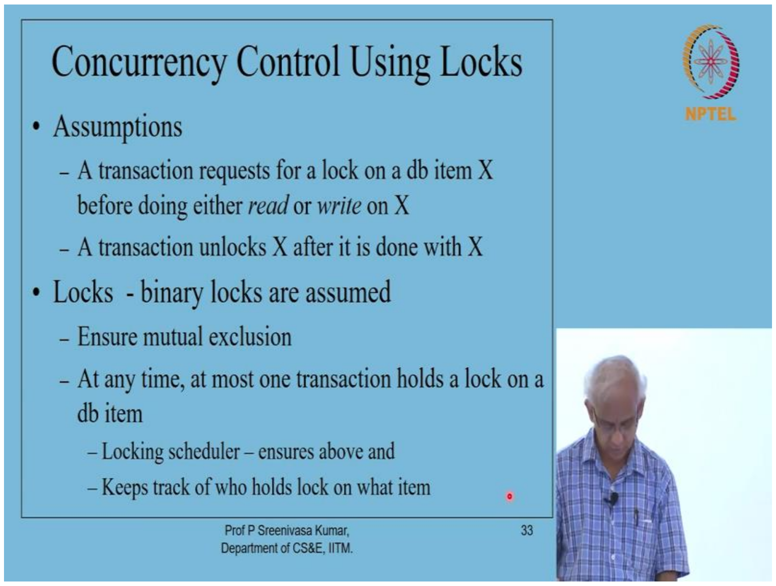
And so, using this notion of view equivalence, we will define view serializable schedule. A schedule is said to view serializable if it is view equivalent to some serial schedule, very important, we are not bothered about any particular if it is view equivalent to some serial schedule. So, we will not go more deep into view serializability. Let us see, we have understood complex serializability.

Let us see how we can actually realize conflict serializability in practice notice one thing here that we have we are taking a peek at the sequence in which the operations of multiple transactions are actually occurring. And then said that some of these histories are permissible some of these histories are not permissible. Especially we are looking for those histories that constitute, conflict serializable schedules.

And we have a way of checking whether there is conflict serializable are not given this history we will be able to check all this is fine. But then you know, transactions are actually running simultaneously and they have been submitted and then the system is actually running them. So is it practical to kind of stop this is system take a history or dependency that all whatever you have done in the last one minute in but what do you have done?

How do we actually realize it what do you have done you are taking a history look at the history and then say this is not good then what do you do practically? So there comes a very beautiful area. So, we will talk about this in this lecture.

(Refer Slide Time: 06:34)



Concurrency Control Using Locks

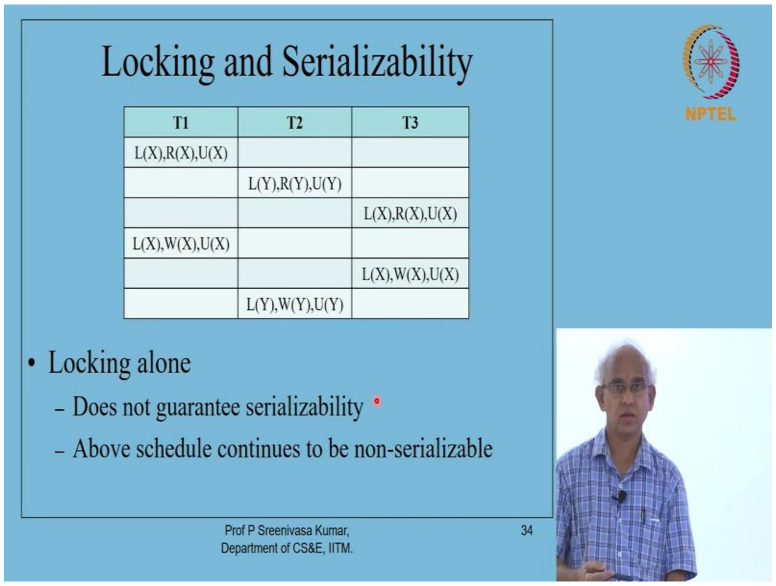
- Assumptions
  - A transaction requests for a lock on a db item X before doing either *read* or *write* on X
  - A transaction unlocks X after it is done with X
- Locks - binary locks are assumed
  - Ensure mutual exclusion
  - At any time, at most one transaction holds a lock on a db item
    - Locking scheduler – ensures above and
    - Keeps track of who holds lock on what item

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 33

So, before we go into that let us introduce locks you are actually already exposed to locks in operating systems context because this is locks is the established mechanism through which you will be able to get mutual exclusive. So we will make use of locks so we will see how we can achieve concurrency control using locks. So, the assumptions that we make here is a transaction request for a lock on any db item X before it is either doing reading or writing of it.

So, we will go for a simpler model where we use binary locks. So it is asking if it has to ask for a lock obtain that and then you will read or write and they get this item and transaction unlocks it after it is done with it. And so we will also assume that there is a locking scheduler or locking manager which is kind of keeping track of the situation. So, the binary locks we will assume the as you have studied in operation systems they will ensure mutual exclusion. So at any point of time at most one transaction holds a lock on some db item and who is holding the lock is going to be kept track of by this locking schedule, so we will assume that there is a locking schedule.

(Refer Slide Time: 08:30)



Locking and Serializability

T1	T2	T3
L(X),R(X),U(X)		
	L(Y),R(Y),U(Y)	
		L(X),R(X),U(X)
L(X),W(X),U(X)		
		L(X),W(X),U(X)
	L(Y),W(Y),U(Y)	

- Locking alone
  - Does not guarantee serializability
  - Above schedule continues to be non-serializable

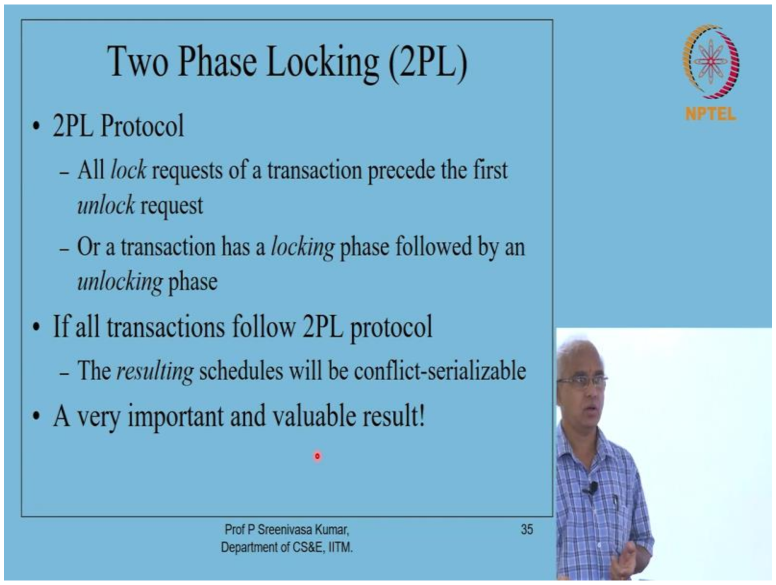
Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

34

Now just having locks by itself actually does not ensure serialized. So I have taken that same example which we have been looking at in the last lecture which is actually not serializable and introduces locks lock X, read item unlock it. That is what I will replace this RX by lock X RX and lock X. So it is actually to be expanded as it is sequence and like that. So before every operation, I have simply introduced locks but the fundamental nature of the schedule actually has not changed.

So, it will continue to give problems for us the conflict it does not guarantee complex serialized. So, here comes a very important idea as to how to make use of locks and actually be able to realize complex serializable schedules.

(Refer Slide Time: 09:36)


Two Phase Locking (2PL)

- 2PL Protocol
  - All *lock* requests of a transaction precede the first *unlock* request
  - Or a transaction has a *locking* phase followed by an *unlocking* phase
- If all transactions follow 2PL protocol
  - The *resulting* schedules will be conflict-serializable
- A very important and valuable result!

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 35

We introduce what is called two phase lock, have you heard about two phase locking in any other context. So two phase locking is a locking protocol that has to be followed by all the parties, all the transactions up to now simply follow this particular rule what it says is that all lock requests of a transaction have to proceed the first unlock request. So let me state again, all the lock requests.

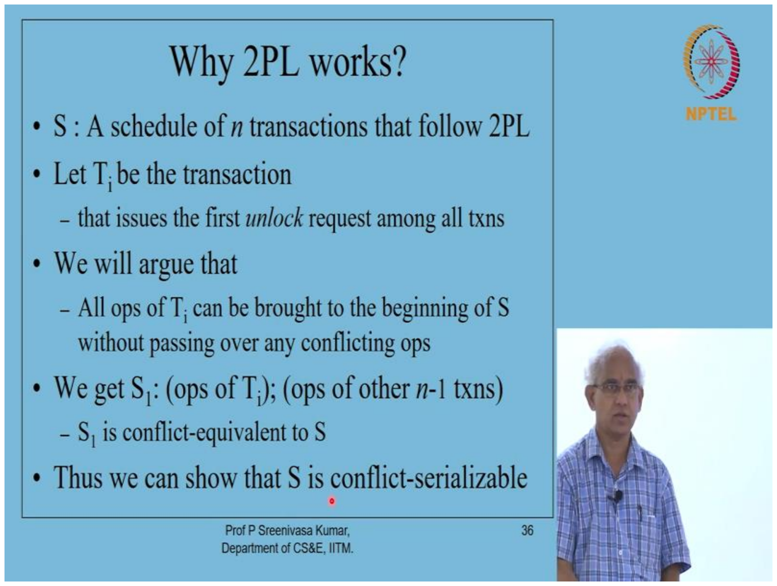
So you keep on making, you know, for every accessing any db access item, you have to lock it first and then unlock it. Now what this is saying this 2PL protocol is saying is that, so keep on asking for whatever locks that you want. And after you issue the first unlock operation, do not ask for any more locks. So it is going to have 2 phases. In the 1st phase you keep on asking for whatever locks that you want.

Because you want to access database items, you need locks, so you keep asking for locks and then operate on the items also simultaneously operate on those items. Because you have acquired a lock, you can operate on items. So keep on doing that. But do not load release locks. If you still have an intention of asking for one more item it have an intention of asking for one more item do not release the lock, the first time you release a lock unlock it. Then after that, you cannot again ask for locks keep on unlocking that.

So that is what this protocol says a transaction has a locking phase followed by an unlocking phase. So this is a real simple beautiful rule or protocol. And what we are going to show is if all transactions follow this simple 2PL protocol, nice a beautiful protocol then one can I actually argue that the resulting schedules will be conflict serialized resulting schedules whatever be the so this is a beautiful way of realizing conflict serialized otherwise, what options you have?.

You have to stop and then look for the history and then check whether you know the history that has happened is conflict serializable or they are not practical but what is practical is that you say, here is a rule all of you follow this rule and if you all if all transaction follow this 2PL protocol, then the resulting schedule will always be conflict serialization. So, a major problem of how do we ensure conflict serializability is solved by this simple protocol call this two phase locking protocol. So very, very important and valuable result for us in practice. Now, in the remaining part of the lecture let me argue has to why exactly how exactly this 2PL protocol allows conflict?

(Refer Slide Time: 13:23)



Why 2PL works?

- S : A schedule of *n* transactions that follow 2PL
- Let T_i be the transaction
  - that issues the first *unlock* request among all txns
- We will argue that
  - All ops of T_i can be brought to the beginning of S without passing over any conflicting ops
- We get S_1: (ops of T_i); (ops of other *n*-1 txns)
  - S_1 is conflict-equivalent to S
- Thus we can show that S is conflict-serializable

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 36

Only conflict serializable, students why 2PL actually want? So, this is kind of inverse trying to make it prove we make it small proof here. Not very difficult actually is not compared to the other proof that we have. So let us look at it. So S is a schedule of some n transactions, all of which are following 2PL all of us we assume that are following two phase locking protocol now, let T i some one particular transaction T i be the transaction that issues the first unblock request among all the transactions there are a bunch of transactions, so some n number of transactions.

So, if you now, look at the schedule. So in the schedule now, in addition to the read write operations, we have also the lock and unlock operations. So, let us denote them by L subscript I and U subscript I and they are locking specific items. So L i of X, U i of X will come into picture in the operations. So if you look at the schedule, it is easy to find out as to what are we what is this particular transaction that is issuing the first unlock among all transactions very easy.

Let that be T i, the T i is the one that issues the first unlock request among all transactions. Now, what we will argue is that this all the operations of this particular transaction T i can be brought to the beginning of schedule without passing over any conflicting operations. So, if you imagine this transaction and the history or the schedule or the long chain of these operations, so, the T i is the transaction that is issued is first unlock operation compared to all the other transactions.

Now the T i is operations are spread somewhere here, somewhere in the schedule. So my argument here is a will good all you that I can take each of these T i operations and then actively swap them all the way we will beginning of the schedule. How when can you swap as long as it does not conflict with the preceding operation I can swap it. So my argument here is that the operations of T i can be brought to the beginning of S without passing over any conflicting operations.

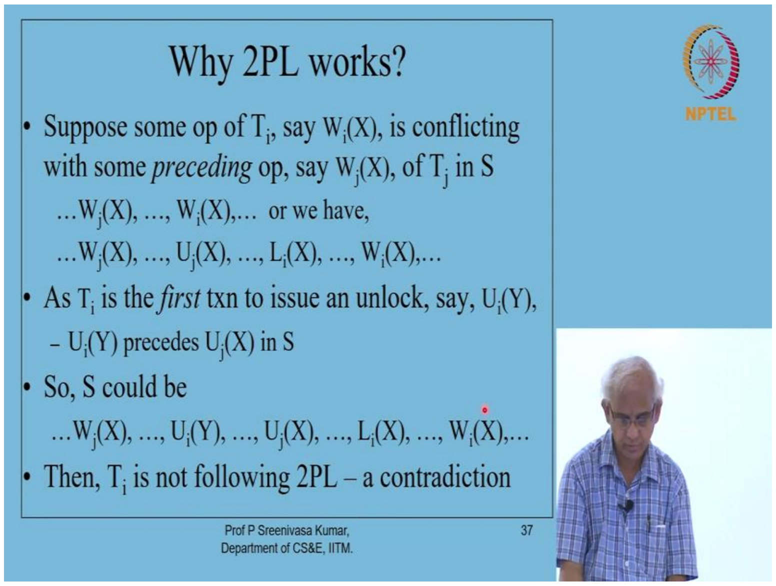
Suppose we are able to show this, then we from S will get a new schedule called S 1. Where all the operations of T i are all together followed by operations of all this n-1 number of other transaction. And S 1 is actually conflict equivalent to S because as I told you that we are able to bring the operations of T i to the beginning of the schedule without passing over conflicting operations. So, because of that, this S 1 will be actually conflict equivalent to S.

So if we succeed doing this, now, we have reduced S 2 S 1 where operations of T i come first and then operation of n-1 will come. Now, we can actually do an inductive argument and then say that now, let me focus on these n-1 transactions among them, let me find out what is the one that has the first unlock operation, I can repeat the same argument and then bring the operations of that particular transaction all the way to the beginning of that S 1 when this part.

And then continue doing that, then I will realize, I will be able to show that this S the initial S that I have been given is actually conflict equivalent to some serial schedule in which you know, you can also see what is serial schedule, in that the operations of the transaction that we should be first unlock operation will come first. Then the operations of the one that gave next unlock operation will come next like that, in the order of the unlocking of the items.

So this is your, so let us see how actually we can do that. So, all that remains to be seen is it how can you actually do this, that the operations of T i can be brought to the beginning of S without passing over any conflicting operation, we are able to show that then we can repeat that on the remaining part of the schedule and then the able to convert this given schedule to a serial schedule. And so, the given schedule is actually is a comprehensive.

(Refer Slide Time: 19:37)



Why 2PL works?

- Suppose some op of T_i, say $W_i(X)$, is conflicting with some *preceding* op, say $W_j(X)$, of T_j in S
  $\ldots W_j(X), \ldots, W_i(X), \ldots$ or we have,
  $\ldots W_j(X), \ldots, U_j(X), \ldots, L_i(X), \ldots, W_i(X), \ldots$
- As T_i is the *first* txn to issue an unlock, say, $U_i(Y)$,
  – $U_i(Y)$ precedes $U_j(X)$ in S
- So, S could be
  $\ldots W_j(X), \ldots, U_i(Y), \ldots, U_j(X), \ldots, L_i(X), \ldots, W_i(X), \ldots$
- Then, T_i is not following 2PL – a contradiction

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 37

So now let us focus on that part again. Let us focus a little bit on that part. So this argument will happen by prove by conflict. So let us see, suppose some operation of T i say W i X is conflicting with some preceding operation of preceding operation say W j of X in this schedule S let us assume that so if we assume S suppose we have let a confliction, then we can show that such kind of thing does not exist. Such kind of things does not arise.

So let us assume that let us suppose that operation T i, some operation T i, let us take it as W i of X is conflicting with some preceding operation of operation. And let us also choose W j of X. In this case, it is so that means it is happening with this W j X comes. So the schedule is having some preceding operations, W j X comes here and then after some time W i X comes, so it is kind of now, if such thing is what is actually happening in the schedule, then we can now insert the lock and unlock operations.

So, W j X must have done an, unlock on X only then I can this transaction T i can write do a write on that, it can lock it and then do write on that. So, U j of X is represents the unlock of the transaction j on the item X must have happened after that lock i of X must have happened only then W i X have happened. So, this is how the sequence can be. Now, let us bring in the assumption about T i what is the assumption about T i?

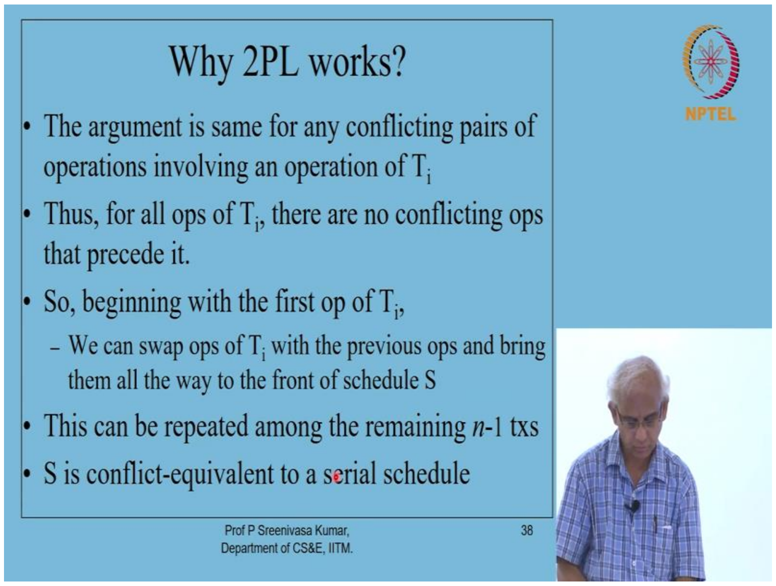
T i is the transaction that did the first unlock among all the other transactions. So, it did some unlock. So obviously that unlock must have come here because this unlock is happening here. So as a T i as T i is the first transaction to issue an unlock let us say U i of Y and some item it has released a lock. So this U i must precedes this because this is also an unlock operation. So, but we know that T i is the one that we give the first unlock operation. So T i is a unlock operation U i of Y should come somewhere here.

But that immediately you can see is a contradiction because the transaction T i is forced to put his unlock here and then after that lock this item X here then transaction T i is not actually following the 2PL protocol because after it has done an unlock it cannot ask for a lock. So, W j X U i Y U j X L i X W i X this kind of sequence cannot occur if T i is actually following 2PL protocol. So, this leads to a contradiction about the assumption that we made that all the transactions follow 2PL protocol.

So, is that clear know, so, T i is not following to 2PL protocol is a contradiction. So, what actually it means now is that our assumption that there is some W j X that precedes W i X exists if we make this assumption that W j X precedes W i X, all these sequence of arguments coming into picture and leading us to a contradiction. So, we can now conclude that such a pair does not exist at all such a pair of operations does not exist at all, we cannot find such a pair of operation if we are able to find such an operation.

Then we will actually come to a contradiction saying that the transaction T i is not following 2PL protocol, but we made an assumption that everybody has to follow 2PL protocol. Now, actually for this particular pair W i and W j is not the nothing very particularly you know important about them, you can actually take any pair of conflicting operations W i and this could be R i of X and then that could be W j of X are as long as one of them is a right operation and both of them are doing the same database item. So, you can take any pair of conflictive.

(Refer Slide Time: 25:34)



Why 2PL works?

- The argument is same for any conflicting pairs of operations involving an operation of T_i
- Thus, for all ops of T_i, there are no conflicting ops that precede it.
- So, beginning with the first op of T_i,
  - We can swap ops of T_i with the previous ops and bring them all the way to the front of schedule S
- This can be repeated among the remaining *n*-1 txs
- S is conflict-equivalent to a serial schedule

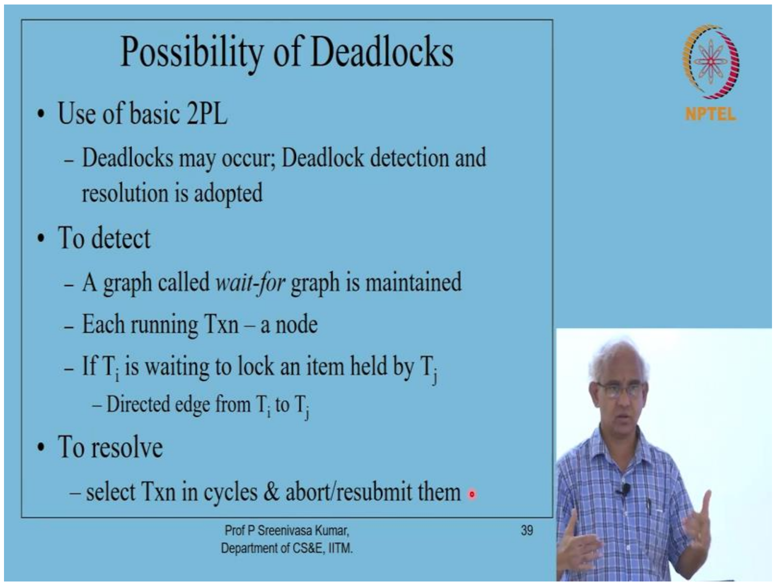
Prof P Sreenivasa Kumar, Department of CS&E, IITM. 38

So, argument is same for any conflicting pairs of operations involving an operation of T i. So, if you make such an assumption that some operation of T i conflicting you know, so some operation that conflicts with the operation of T i exists before that we see it is we have a contradiction. So we conclude that there is no such operation and if there is no such operation then for all these operations of T i, there are no conflicting operations that precede it.

If there are no operations that are precede it, then what I can do is to take this operations of T i and then actually swap them all the way, of the beginning of the schedule. So, beginning with the first operation of T i, we can swap operations of T i with the previous operations and bring them all the way to the front of the schedule S. So, because of that we can show that 2PL indeed always guarantees that we will end up conflict serializable schedules.

So, S, the schedule that we started of a is conflict equivalent to this particular serial schedule where operations of T i come first and the operations of the other transactions in which the second one will be the one that has issued the next unlock first things like that.

(Refer Slide Time: 27:42)



Possibility of Deadlocks

- Use of basic 2PL
  - Deadlocks may occur; Deadlock detection and resolution is adopted
- To detect
  - A graph called *wait-for* graph is maintained
  - Each running Txn – a node
  - If T_i is waiting to lock an item held by T_j
    - Directed edge from T_i to T_j
- To resolve
  - select Txn in cycles & abort/resubmit them

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 39

Now, so, this is good, this is the best thing that can happen. So, 2PL is the one that saves us, because it is the one that if you follow this protocol if all the individual transactions follow this protocol then the whatever sequence of operations that actually ensures that actually happens turns out to be conflict realize. So, and of course, I did not explicitly show here but we can see that from the definition itself that these conflict serializable schedules or really desirable schedules because they will avoid all those problems of last updates and repeatable waits and dirty waits and all that.

You can show that we can also work. Now, as you know, if you use locks, there is also a possibility of deadlocks. So deadlocks may occur. So what normally is done in database systems is to do a deadlock detection and resolution. So you can actually construct this, wait for graph as to the transaction processing subsystem, we allow admit transactions and then keep on checking whether who is waiting for whom. So you can construct a wait for graph.

And as long as a cycle comes into the picture, as long as a cycle does not come into the picture you are safe, so can keep on allowing the transactions to operate. If there is a cycle, then you know that that deadlock is happened. And so you can actually break that cycle. Take a particular arbitrary transaction from there and then simply aborted. So the transaction processing system is going to have a control on, who is going to succeed who is not going to succeed?

So it is good to just pick up one update that re admit resubmit the transaction, it will break the lock. Good. So with the combination of using two phase locking protocol and deadlock detection and resolution, we will be able to actually realize a concurrency control subsystem which will ensure that the transactions are happening in a safe way. There. So with that actually I will stop today and then I will take up the issues connecting with system failures crashes and how they influence is schedules.

So this schedules happening, but we also have to take into consideration the issues from system failure point of view. So what are the various considerations that come from system failure point of view and how they affect this transaction interleaving. We will discuss in the next few lectures where we will actually take up more detailed recovery procedures and things like that and then discuss how recovery actually happens.