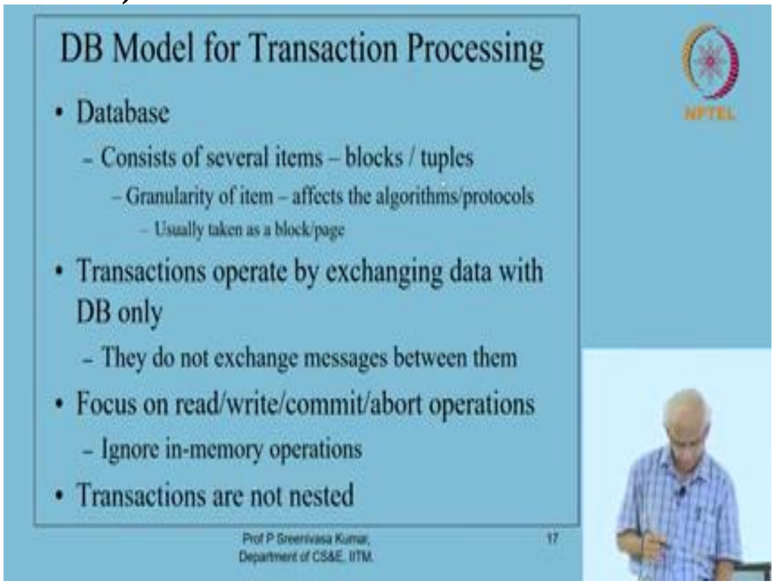
Database Systems
Prof. Dr. Sreenivasa Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 37
Schedules

(Refer Slide Time: 00:29)



DB Model for Transaction Processing

- Database
  - Consists of several items – blocks / tuples
    - Granularity of item – affects the algorithms/protocols
      - Usually taken as a block/page
- Transactions operate by exchanging data with DB only
  - They do not exchange messages between them
- Focus on read/write/commit/abort operations
  - Ignore in-memory operations
- Transactions are not nested

Prof P Sreenivasa Kumar,
Department of CS&E, IITM. 17

I gave a introduction at the area of this transaction processing and error recovering. So, we looked at what exactly is a transaction and then what are the various responsibilities of the programmers involved, what are important properties the atomicity, consistency, isolation and durability, properties of the transaction systems in general and then you know which of these things is the responsibility of what subsystem of the system.

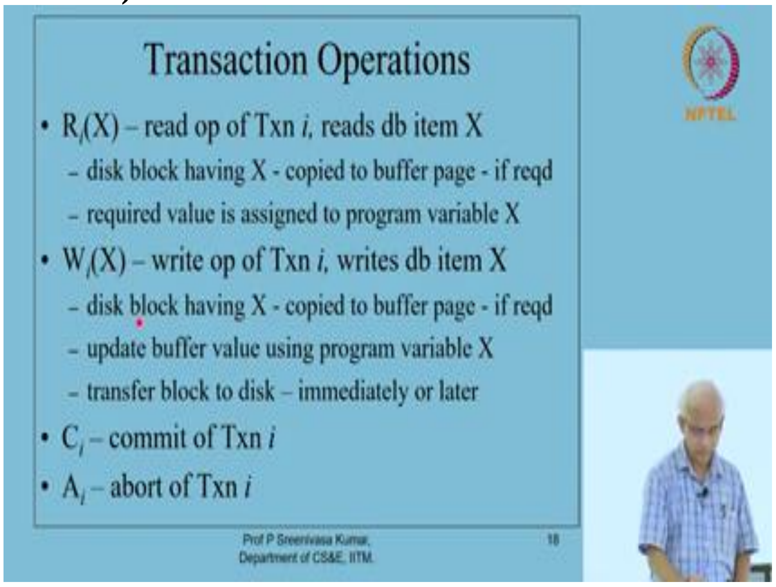
So we looked at all that. So today we will focus first on the issue of managing concurrency appropriately. So, concurrency is an important aspect we have to allow these multiple processes. Each process is an incubation of one of these transaction programs. So, multiple processes to run concurrently and in the, in this context, we should also ensure that the system maintains consistency of the database.

So, how does so what are the various how do we approach this problem? How do we model it? And how do we look at the solution? What are the kind of solutions that are possible in this space? So we will broadly look at the concurrency controls of module. So let me go to the last slide that we are looking at in the last lecture, basically, we just said that the database model for transaction processing is basically is that we will think of the database as consisting of several items and each of these items is independently accessible.

And then the transactions will ask for these items, modify them and write them back. So usually we make an assumption that this item, the granularity of this item is that it is a block or a page and then transactions. Transactions only operate by exchanging data with the data base server only. They do not exchange messages between them. And we will focus on these read, write, commit and abort operations, plus, plus.

They will gave you as for what those things stand for we actually will not be able to at this stage will not be able to fully appreciate what should happen in commit and what will happen in abort unless we also study this recovery procedures. So, we will come to that in the next week actually. And we also said that the transactions should not be nested.

(Refer Slide Time: 03:27)



Transaction Operations

- $R_i(X)$ – read op of Txn *i*, reads db item X
  - disk block having X - copied to buffer page - if reqd
  - required value is assigned to program variable X
- $W_i(X)$ – write op of Txn *i*, writes db item X
  - disk block having X - copied to buffer page - if reqd
  - update buffer value using program variable X
  - transfer block to disk – immediately or later
- C_i – commit of Txn *i*
- A_i – abort of Txn *i*

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 18

So, with this background let me go a little more closely at this operations of this transactions. So let us develop some notation for representing these things. So R i of X this symbol will stand for the read operation of transaction i. So since we have we are going to mix the operations of the transactions, several of them a bunch of them together and then start talking about them. So we need to distinguish between the various operations of each of these.

So, we use this transaction number as a subscript for these operations. So R subscript i indicates that i is the transaction number and the transaction is showing a read and the db item is X. So what exactly happens when you do a read i read X is that the disk block having this particular item X is copied to the buffer page from the disk if required. So, it is possible that some other transaction as you know, requested for the same item a little earlier and so, the page is actually lying on the memory in the memory.

So if it is required, it will be brought from the disk to the buffer space. So, copy to the buffer pages. So we will think of the buffer as something which has a number of these pages which are all equivalent to the disk block size. So, the disk box will be first brought to the buffer page and then they will be made accessible to the transactions and then the required value whatever this value is assigned to some program variable X. So, even though the page is brought the entire page may not be actually used.

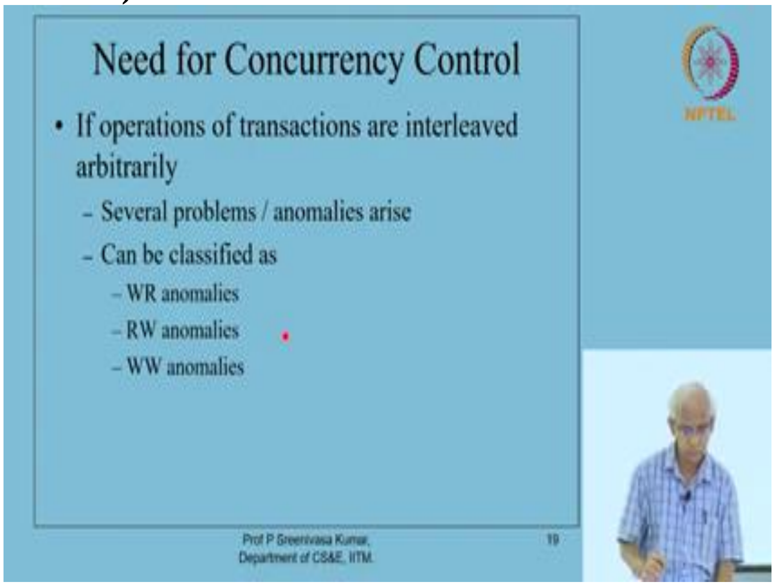
So, whatever is required by the program it will be assigned to X. So read is that and then similarly, W i X i means again the transaction i. So, a particular transaction i is doing this write operation. So, it writes the db item X. So, again if you update the db item X, so, you need to have the disk block containing this item X. So if it is required if it is not already present in the buffer pages, then it has to be brought from the disk and then made available to the transaction.

So, the transaction will update the buffer value using this program variable whatever you see, we have some internal the program space is there a program has its own memory space. So, it is making some values as it is running in memory. So, those values will be transferred to these buffer pages and then they will be transfer block to the disk. So, when you do a write, they transfer of the block to disk. So, this might happen immediately or might happen later. So, this actually depends on the kind of recovery protocol that we are going to adopt.

So, for the moment we will think of it as the transfer of the block to the disk will happen at some later point of time either immediately or later it will happen. So, unless it happens actually the write operation is not fully successful done we can see that. So that is write and then we have C of i to say the transaction wants to commit. So, the transaction commits and A of i indicate that a transaction wants to abort.

So, these are the transaction operations and we just thought of them as read, write, commit, abort via this lecture, but now we are giving more details to that. So the transaction number comes as the subscript and the item that is actually being read or write comes inside the brackets.

(Refer Slide Time: 07:55)



Need for Concurrency Control

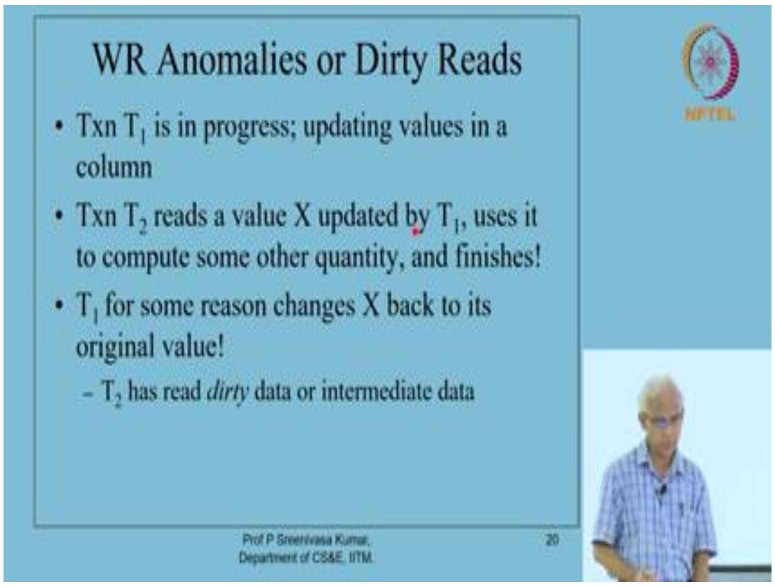
- If operations of transactions are interleaved arbitrarily
  - Several problems / anomalies arise
  - Can be classified as
    - WR anomalies
    - RW anomalies
    - WW anomalies

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 19

So, with this notation, we will be able to set up what is called a transaction the schedule of transactions, we will set up. So before we go into that, let us briefly discuss the need for concurrency control, I think we can actually individually feel the need for the concurrency control because it is more multiple processes try to access the same item, then we know that there is there are problems they are going to be problem.

So, let us give some clarity to that. So, these anomalies are the problems that are going to occur when you interleave operations of multiple transactions in an arbitrary way. It can be kind of classified into the write read anomalies or read write anomalies of write write anomalies. I will show you these things in shorter time.

(Refer Slide Time: 08:55)



WR Anomalies or Dirty Reads

- Txn T_1 is in progress; updating values in a column
- Txn T_2 reads a value X updated by T_1, uses it to compute some other quantity, and finishes!
- T_1 for some reason changes X back to its original value!
  - T_2 has read *dirty* data or intermediate data

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 20

So, write read anomalies are they are also called dirty reads in our context in the context of databases, we call them a dirty reads, it becomes clear now, why it is. So the transaction T i, let us say T 1 is in progress, so it is updating values in a particular column. And then so, the system allows transaction T 2 kind of read a value X that has been already updated by T 1 make use of that value compute some other quantity and also finish.

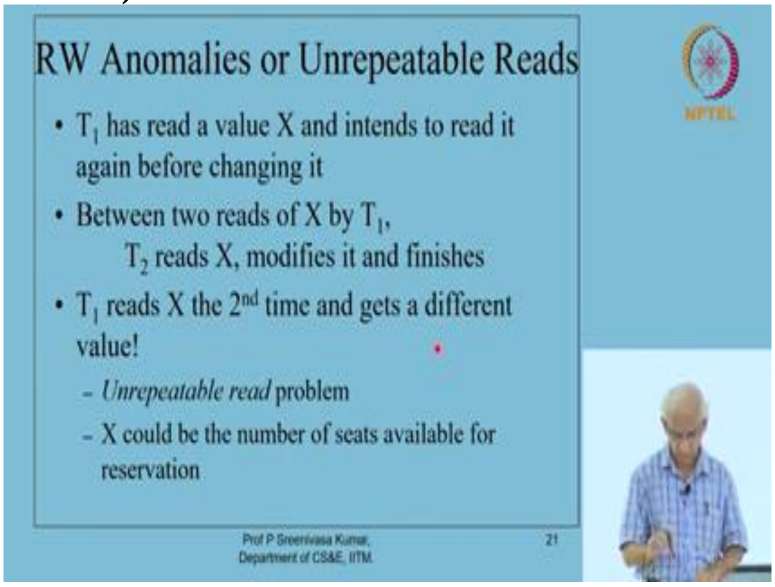
By finish it says that I am committing and quitting. So, suppose this system allows this kind of a thing then the transaction T 1 is still in progress. So after this entire T 2 has read the value X and then make use of it and then finish T 1 for some reason might actually decides saying that, no, no I did a mistake in updating this X value, I will have to go back and then change it back to the original value.

Notice that as a programmer, I have the, I have the freedom to do many things with the database. So I might, you know, programmatically decide at a later point of time that there is a have been some mistake. And so I will have to I may want to do a change back to the original value. Also this might happen because due to some reason we detected that this transaction T 1 is not I know following rules properly and the things like that the system might itself, decide to roll back the operations of the transaction T 1.

In, which case the value that it has changed has to be brought back to the original value. So in this context, what happens is that there is a write that happened and there is a read that happened. And there is a cancellation of the write is happening. And so, essentially what happened is that the T 2 has read some dirty data, it is read some intermediate data, or it is also called dirty data.

It is not correct data and so got into trouble, actually the solution for this issue T 2 actually might having read X having make use of this X, he also actually wait for a while, see what exactly happens to T 1 and then decide to do whether to finish or not. So but without doing all without doing all such kind of things T 2 in a hurry, as you know, updated and claimed that it has completed its work and quits. And suppose the system allows this then this kind of dirty read problem will arise. So this is a dirty read kind of situation called, like that. It is not a anomaly situation we would like to avoid that.

(Refer Slide Time: 12:20)



RW Anomalies or Unrepeatable Reads

- T_1 has read a value X and intends to read it again before changing it
- Between two reads of X by T_1, T_2 reads X, modifies it and finishes
- T_1 reads X the 2nd time and gets a different value!
  - *Unrepeatable read* problem
  - X could be the number of seats available for reservation

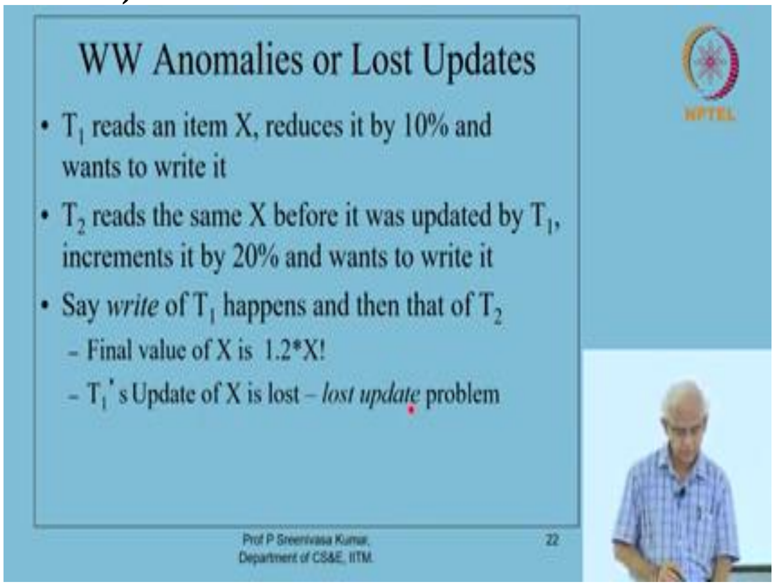
Prof P Sreenivasa Kumar, Department of CS&E, IITM. 21

Here are the other kind of unrepeatable reads we call them as unrepeatable reads. So this is a read write anomaly. Your T 1 has read a value X and intends to actually read it again before changing it. Between these 2 reads of X by the transaction T 1 some of the transaction T 2 reads X modifies it and finishes, quits. So, when T 1 reads the value of X the second time it gets a different.

So, the read that you have done is not a repeatable way. But you get 2 different values when you read the same thing without you updating that means somebody else has interfering your work basically, right? So you, read a item from the database and then you again read it, you have not modified it yourself and you try to read it after a while, you expect the same value to be there because you guarantee that isolation.

But that does not happen here, because something else has come and then change. So this kind of an issue, again, is because the system allowed T 2 to read X before bothering about what is happening here. So this kind of thing is what is called unrepeatable read problem and it causes issues. And so we should try to avoid this kind of anomaly situation. So, all this activity is happening, because 2 transactions are trying to access the same database item in some order of that.

(Refer Slide Time: 14:14)



WW Anomalies or Lost Updates

- T_1 reads an item X, reduces it by 10% and wants to write it
- T_2 reads the same X before it was updated by T_1, increments it by 20% and wants to write it
- Say *write* of T_1 happens and then that of T_2
  - Final value of X is 1.2*X!
  - T_1's Update of X is lost – *lost update* problem

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.
22

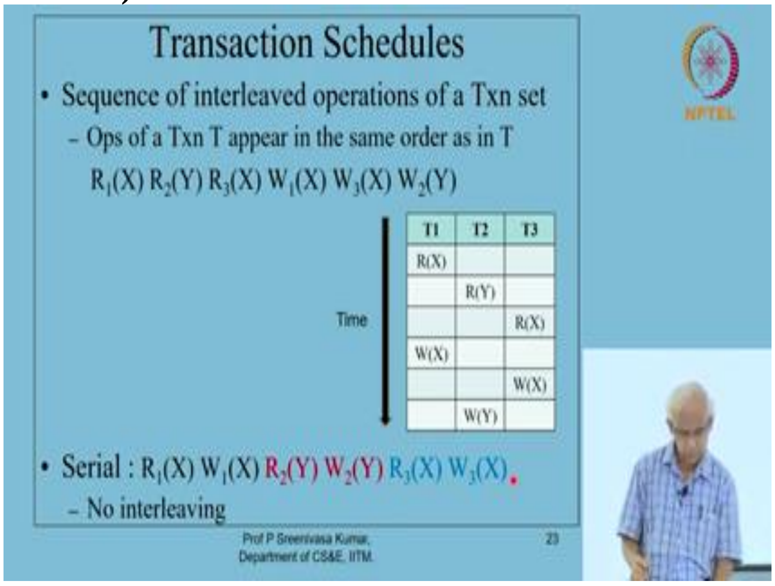
This problem called WW anomaly or lost update problem was called lost update problem also happens in a kind of similar way. So T 1 reads an item X and reduces by 10% and wants to write it. It does not yet write it, it has done a 10% reduction. T 2 reads the same value X before it was actually updated by X. That means the same original value X and increments it by 20% and it wants to write.

So, T 1wants to increase it by reduces it by 10% and T 2 wants to increase the same value by 20%. Now, both of them are attempting to write. So whichever write happens first will actually win so in some sense. Say write of T 1 happens then and after that the write of T 2 happens, let us say. So, the final value is actually 1.2 times X only. So, whoever actually has written first has actually lost something. You can see that, so, write of T 1 happened and then that of T 2 happen.

But then the final value is what T 2 wrote. So what is the final whatever the value that was written by T 1 was overwritten by T 2. And so the update that was made by T 1, it is a 10% reduction was actually lost. It lost. So, this issue is what called a lost update problem. So, you might have actually studied these kind of issues in you know, in processes trying to access memory variables, you know, simultaneously like that in an operating system, course context also.

And you would have seen this you know, memory traces and similar kind of issues come up there. Anyway, so, these things are very real problems in the in the context of our transactions also. And basically these are the various things that we should avoid actually, as when we are controlling the concurrency.

(Refer Slide Time: 16:44)



Transaction Schedules

- Sequence of interleaved operations of a Txn set
  - Ops of a Txn T appear in the same order as in T

  $R_1(X)\ R_2(Y)\ R_3(X)\ W_1(X)\ W_3(X)\ W_2(Y)$

Time

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

- Serial : $R_1(X)\ W_1(X)\ R_2(Y)\ W_2(Y)\ R_3(X)\ W_3(X)$.
  - No interleaving

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 23

So, let us move ahead. Here is an important notation that we make use of in order to model the interleaved operations of a bunch of transactions are currently running, kind of admitted and they are all running. So we want to take a peek into how they have actually happened. What is the exact interleaving of the operations of these transactions, so we need a model for that; we need a notation for that.

So let us represent it like this. So it is actually easier to view it in a table or kind of form. So time is progressing this way. These are the individual operations of the transaction T 1. So since there they are the column T 1, we did not put the subscripts here. So then the column T 2, so that means they are all of transaction T 2, so we do not put the subscripts, in this similar way these are.

This is the time line. You can see that each time line, there is only one event one operation. So, at any point of time it is the uniprocessor model at any point of time one of these operations is happening. So, this is the interleaving of operations of transactions that happening. Now, since each time writing this picture is difficult, so we have a notation for that. So what we will do is we will write this as R 1 X.

Then the next thing is R 2 Y. Transaction 2 is writing reading Y. So R 2 Y then, so next, it will go to the next time step is R 3 X. So we will write it R 3 X here and then W 1 X. Then, W 3 X and then finally W 3 Y. So, we write this in this particular sequence and we call this as the transaction schedule or list of transaction schedule. So, we assume that there are a bunch of 3 transactions running and this is a particular way in which the operations of the transactions are interleaved by the system and run by the system.

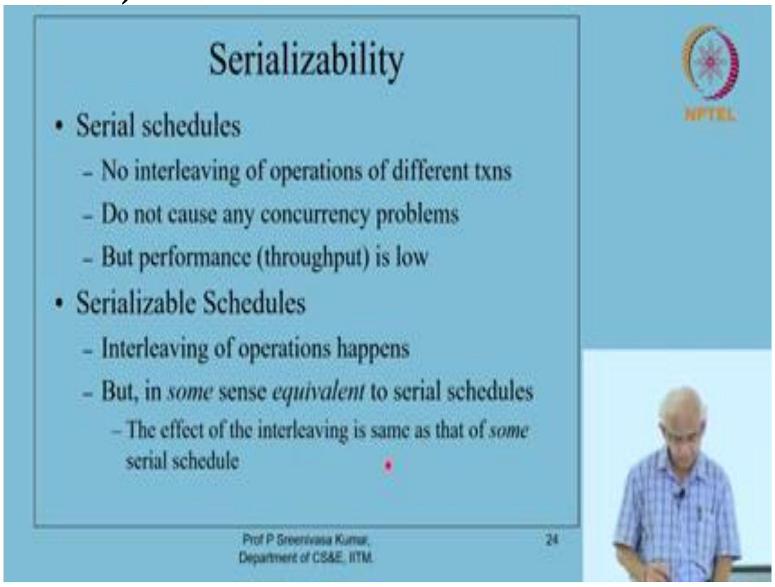
So, we have a notation. Now we use this notation to kind of study the properties of the schedules and then actually define what is a desirable property or schedule and then see how exactly that kind of schedules will actually have in the system. So this is just a notation for representing the histories of their running. It is also not fully complete, because we have actually not put the commit, abort, commit operations of these transactions also, it is kind of incomplete schedules.

Now, in the context of these transaction histories of schedules, this is another important thing, we call something as a serial schedule. A serial schedule is where you know no such interleaving actually happens, no interleaving happens. So, all the operations of one particular transaction come together. In this case, they all come in one T 1 T 2 T 3 in that order, but the order that actually does not matter, you may put operations of T 3 first and then operations of T 1 first and then the operations of T 2.

So as long as all the operations of a particular transaction all together in the schedule, we call such a schedule as a serial schedule essentially, its operating there is no interleaving. So, we call that as a serial schedule. Notice that we have you know discussed earlier that if you have serial schedule then there is no issue of any problems arising at all because we are doing one transaction at a time essentially, there is no interleaving but then the transaction throughput will be very slow.

Because, you are kind of waiting for these serial operations to finish and all that. So serial schedules are they do not pose any concurrency problems at all, but then they are practically not useful because they are they will slow down the system. But here is the notation for serial schedule. So all the operations of run a particular transaction all together.

(Refer Slide Time: 22:11)



Serializability

- Serial schedules
  - No interleaving of operations of different txns
  - Do not cause any concurrency problems
  - But performance (throughput) is low
- Serializable Schedules
  - Interleaving of operations happens
  - But, in *some* sense *equivalent* to serial schedules
    - The effect of the interleaving is same as that of *some* serial schedule

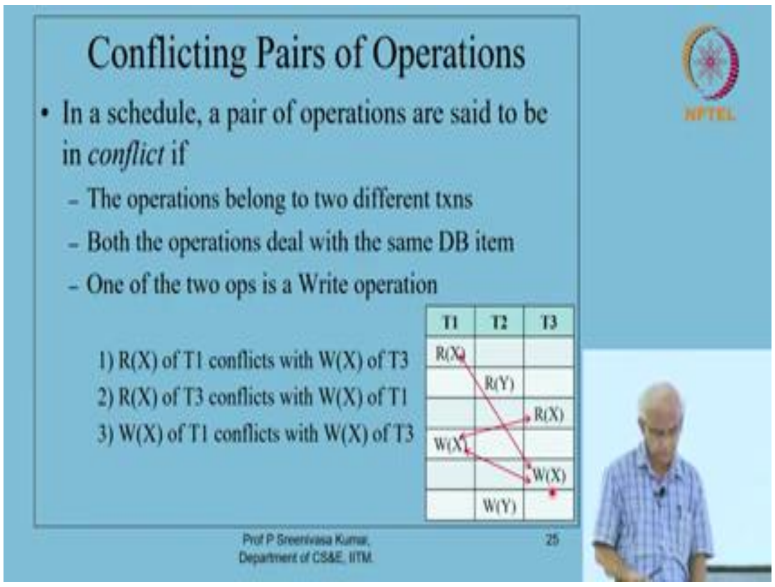
Prof P Sreenivasa Kumar, Department of CS&E, IITM. 24

So here comes a very important new term called serializability type it in word it will give a spelling error actually. We use this term often in database context, this is called serializability. Serializability is defined. So right now so serial schedules, you will see that there are no interleaving of operations of different transactions and they do not cause concurrency problems, but then the performance is low.

A question arises is that is there some interleaving of these all of these operations, which in some sense, equivalent to a serious schedule. If the interleaving actually happens, but somehow the interleaving is in some sense equivalent to a serial schedule. If such a thing can be set up, then we will call such things as serializable schedules. They are not actually serial schedule but then they are, in some sense equivalent to serial schedule.

So we will call such kind of schedule serializable schedule. And if we somehow can realize this serializable schedules in practice, then we will feel better off compared to having serial schedules. So what is this sense of equivalence that is what we will now define and set up. So the effect of this interleaving is same as that of some serial schedules. If that is the case, we will call them as serializable schedules. So but this one is vague because we have not really told as to what is that equivalence and things like that. So let me make this much more precise now.

(Refer Slide Time: 24:44)



Conflicting Pairs of Operations

- In a schedule, a pair of operations are said to be in *conflict* if
  - The operations belong to two different txns
  - Both the operations deal with the same DB item
  - One of the two ops is a Write operation

1) R(X) of T1 conflicts with W(X) of T3
2) R(X) of T3 conflicts with W(X) of T1
3) W(X) of T1 conflicts with W(X) of T3

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

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 25

Now, in order to make this precise, we need to have a definition calling what are called conflicting pairs of operations within a schedule. Because it is all these conflicting pairs of operations that are going to actually cause problems for us, so let us set them up and then discuss them, how to handle them. So, a pair of operations in the schedule. So there is this kind of, so we are picking up some pair of operations and they said to be in conflict, if the operations belong to 2 different transactions.

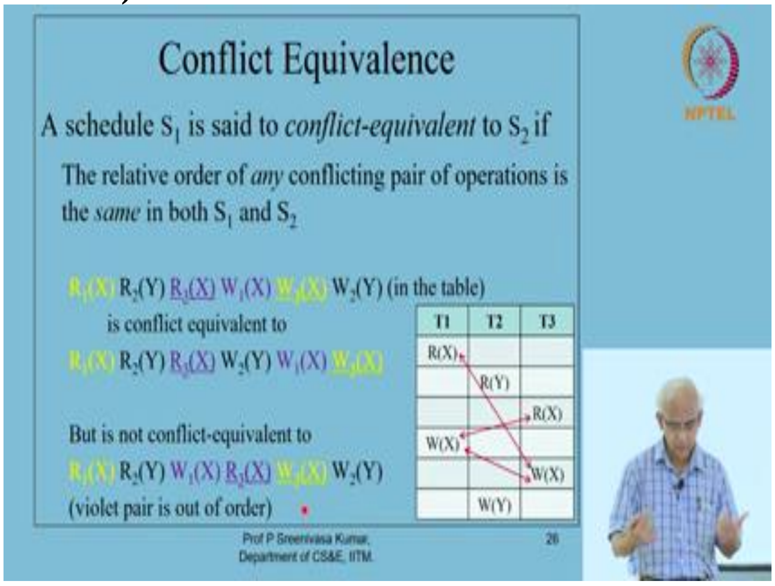
Operations belong to 2 different transactions. Both the operations are dealing with the same database item. And one of them is a write operation. This is precisely the kind of, you know, pairs of problem pairs of operations that are actually was causing a lot of these issues like the lost update problems and repeat all those problems. So, let me show you in this schedule, let us identify the conflicting pairs of operations.

So, this transaction T 2 is actually all dealing with Y, whereas T 1 and T 3 are dealing with X. So T 2 operations actually do not conflict with anybody. Because they do not share the same variable and that is clear. So T 1 operations of course any of the operations of the transaction do not conflict to themselves. They are in sequence order. So the R X conflicts with it does not conflict with this R X because both of them are read operations one of them has to be write operation.

So R X conflicts with W X. You can see R X conflicts with W X and this R X conflicts with that W X because they are both represent the same item one of it is write operation they belong 2 different transactions same is the case this W X. Let 3 pairs of conflicting operations here in this particular schedule. So, given any schedule, we will be able to figure out as what are the conflicting pairs of operations? Is it clear, what are conflicting pairs of operations? I am repeat.

There are operations that belong to 2 different transactions or different transactions, both of them reading the same database item and one of them is write operation. So now that we understood what are the conflicting pairs of operations?

(Refer Slide Time: 28:09)



Conflict Equivalence

A schedule S_1 is said to *conflict-equivalent* to S_2 if

The relative order of *any* conflicting pair of operations is the *same* in both S_1 and S_2

$R_1(X)$ $R_2(Y)$ $R_3(X)$ $W_1(X)$ $W_3(X)$ $W_2(Y)$ (in the table)

is conflict equivalent to

$R_1(X)$ $R_2(Y)$ $R_3(X)$ $W_2(Y)$ $W_1(X)$ $W_3(X)$

But is not conflict-equivalent to

$R_1(X)$ $R_2(Y)$ $W_1(X)$ $R_3(X)$ $W_3(X)$ $W_2(Y)$

(violet pair is out of order)

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

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 26

And the position defined a notion of equivalence and this notion of equivalence is specifically called conflict equivalence because it focuses on a certain aspect. So it is called conflict equivalence not a general it is called conflict equivalence. So a schedule S 1 now we will give names for these schedules, because we have to talk about the multiple schedules and then see which are reliable which are not reliable.

So schedule S 1 is said to be conflict equivalent to some schedule S 2. So if the relative order among any pair of conflicting operations, is actually the same in both S 1 and S 2. If you are having 2 schedules and then you focus on this conflicting pairs of operations, each conflicting pair operations you will have certain you know one read occurring and one write occurring like that. So you look at them, if the conflicting pair occurs in the same order in both S 1 and S 2 for all of the conflicting pairs, then we call them as conflict equivalent.

Let me show you an example. Let us take the same schedule. So, this is what is there. So, R 1 X, this is what is there in the table R 1, R 2, R 3, W 3, W 2, W 2 1finally W 1. Now, this is conflict equivalent to this one, you can notice that if you take this the yellow ones, there is a conflicting pair of operations, R 1 X, W 3 X, they are occurring in the same order in here and also here. And in a similar way, if you take these violet ones R 3 X, W 1 X, they are also occurring in the same order R 3 X, W 1 X.

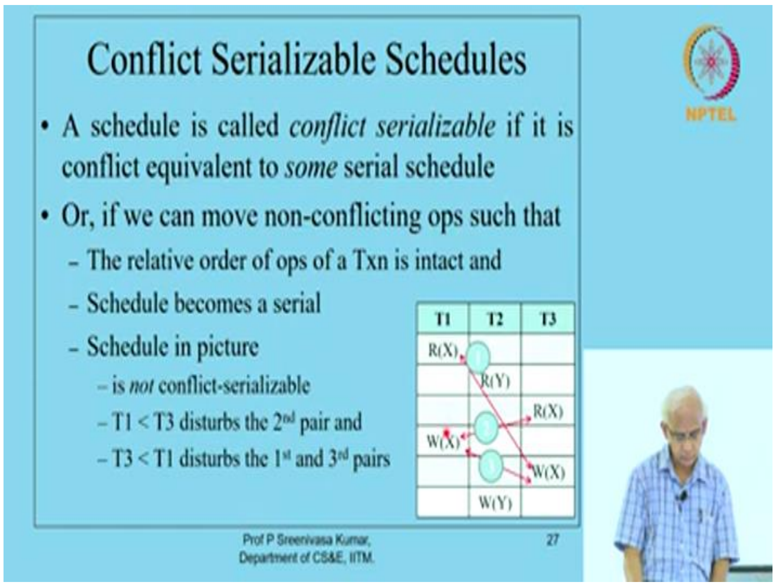
And then if you take this underlined one R 3 X and W 3 X that is something here. Yeah, I think there is some one of them subscript has to be corrected. So let us look at all the 3 so this R 1 X, W 3 X, they are in the same order. And then this R 3, W 3 , R 3, W 3 and so and this one finally W 1 X and W 3 X. So, W 1 X and W 3 X even these are actually in the same order. Now, let us see what exactly happened between this and this.

What we just did is to actually position this W 2 Y bring inside. So W 2 Y, we have pushed it up, push it to this position and then push this, these 2 operations down in the same order and we got a schedule, we got a slight different schedule, but then it is equivalent to the conflict equivalent to the upper side. Now, if you do something similar, but if you change the order among these conflicting pairs of operations obviously, you will not have conflict equivalence.

So, now the, I have done something like this. So, simply swap what is it that got swapped here you can just see R 1 X, R 2 Y. So, W 3 and W 1 X compared to this, I swapped them and got into a different table. So, this obviously is not equivalent to the other one conflict-equivalent the other one. Because, you can see that, yeah, in this for this conflicting pair of operations the relative order in which they are occurring is different in both cases. Now why is this useful? What is this use of this conflict-equivalence? Let us see.

Now, our idea here is, that having developed this notion of equivalence and make use of it to say that, if there is some schedule, if it is conflict equivalent to a serial schedule, some serial schedule that is conflict equivalent to some serial schedule then is going to be useful for us.

(Refer Slide Time: 33:40)



Conflict Serializable Schedules

- A schedule is called *conflict serializable* if it is conflict equivalent to *some* serial schedule
- Or, if we can move non-conflicting ops such that
  - The relative order of ops of a Txn is intact and
  - Schedule becomes a serial
  - Schedule in picture
    - is *not* conflict-serializable
    - T1 < T3 disturbs the 2nd pair and
    - T3 < T1 disturbs the 1st and 3rd pairs

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

Prof P Sreenivasa Kumar,
Department of CS&E, IITM. 27

So, that is why we bring in a specific notion of serializability. The serializability is a bit more general notion, saying that it is useful for you know, it is useful for realizing the concurrency control but here is a more specific you know, property called conflict serializable. So, we defined some schedule as conflict serializable schedule. If it is conflict equivalent to some serial schedule and how many serial schedules are there.

There are some 10 number of transactions there will be factorial 10 number of serial schedules because you can position them in factorial 10 number of different ways. So, depending on how you position them, the ops will finish, you know, different times and all that but we are not concerned much about how when they finish but we are concerned about correctness and all that consistency.

So what we are doing here is that now we are saying that interleaving of operations is ok as long as the particular interleaving that you have done is actually equivalent to some serial schedule. What is that equivalent schedule is conflict-equivalent. So, what is it mean? So, in this particular schedule, which we are calling it as a conflict serializable schedule the conflicting pairs of operations are actually occurring you know in the same order as they are occurring in some serial schedule.

So, in effect, the effect of this particular conflict serializable schedule is in some sense equivalent to that serial schedule going to conflict-equivalent. So, that is what we are going to make use of another way in which we can actually define this conflict serializability is to say that if you focus on non-conflicting pairs of operations and then somehow I know move them around, move them .

So, that the schedule in question becomes a serial schedule. Let me illustrate that. So let us take this. Now this schedule in picture is definitely not conflict serializable. Because what are the possibilities here to kind of realize conflict-serializable. So, the question that we are asking is that is this particular schedule conflict equivalent to some serial schedule. What are the serial schedules are possible.

So, I can put among the T 1 and T 2, I can either put all the operations of T 1 before the operations of T 3. And the operations of T 2 actually do not matter because the T 2 is all operating on Y and does not really conflict with anybody. So, I can put the operations of T 2 all the way you know, before or all the way later, etc. Whereas, if you focus on the operations of T 1 and T 3, if you try and you know, check if this particular schedule is conflict equivalent to a serial schedule in which T 1 comes before T 3 if we try to do that.

Then so, what does it mean? So, this W X should come before this R X. Only then this T 1 operation will be before the operations of T 3 in that serial schedule. So, if you try making this particular schedule that is given in the picture conflict-equivalent to a schedule in which the operations of T 1 come before the operations of T 3 then obviously, this second pair will be out of order. This pair will be out of order because they are right now here like this.

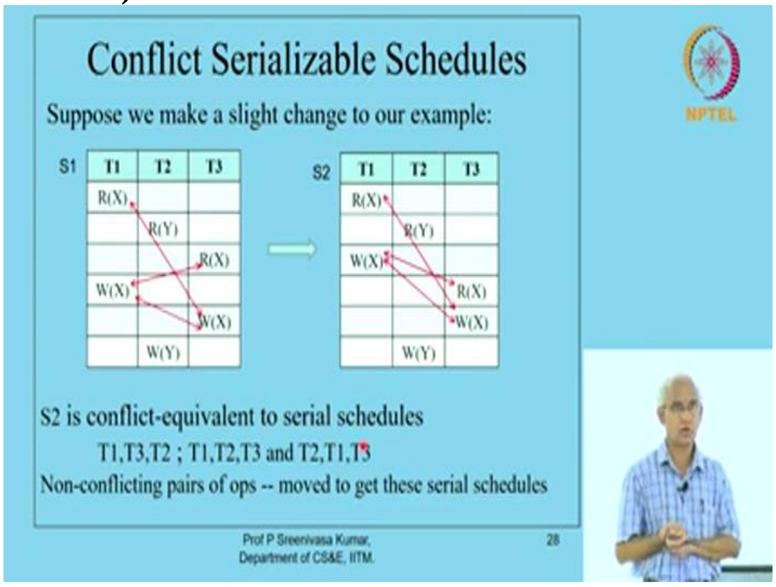
R X comes first and then W X comes. But then if you are trying to put operations of T 1 before the operations of T 3, then the relative order among this pair will get swap. So I cannot try showing that this particular schedule is conflict-equivalent to some schedule in which operations have T 1 come first and then the operations of T 3. In a similar way, if you try to argue that T 3 operations, I have let me try to put before the operations of T 1.

Then also the other pairs will get destroyed. So, let us try let me so trying to put a T 3 before T 1, what it means is that I should actually bring this or should I take these things all the way up here. If you try to take all these things all the way up here, then this pair and this pair will actually get disturbed; because we are trying to take this W X all the way above.

So that is the only way you can realize a serial schedule involving T 3 operations before the T 1 operations. So either way and these are the only 2 possibilities. Either T 1 should come first or T 3 should come or T 3 should come first T 1 come later. They are the only 2 possibilities and in both possibilities, you can see that this particular schedule that we are considering is not conflict-equivalent either of these.

And that is why this schedule is not a conflict serializable schedule there is some inherent problem here. There are some inherent problem you can actually feel that there is some kind of acyclicity problem. So we will make that a little bit more conflict.

(Refer Slide Time: 41:12)



Conflict Serializable Schedules

Suppose we make a slight change to our example:

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)	

S2 is conflict-equivalent to serial schedules

T1,T3,T2 ; T1,T2,T3 and T2,T1,T3

Non-conflicting pairs of ops -- moved to get these serial schedules

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 28

Now to illustrate the situation as slightly a better situation where we can realize complex serializability for this particular schedule, let me change it slightly. Let us focus on this example and change it slightly. A slight change for example what is the change I have done is to simply swap these ops, swap this and this. I put R X here and W X above the same conflicting pairs of operations are occurring.

But then, you can actually immediately see, I suppose if you are following me, you will be able to immediately see that this particular schedule in which I have R X, R Y, W X, R X, W X, W Y is equivalent to conflict equivalent to a schedule in which operations of T 1 come before the operations of T 3. In any schedule in which the operations of T 1 come before the operations of T 3, this is equivalent and the operations of T 2 do not matter.

So, just by swapping these 2 things, you are able to show that this is in the conflict-equivalent. So, S 2 is conflict-equivalent to serial schedule. In fact, it is equivalent to all these serials T 1, T 3, T 2; T 1, T 2, T 3, T 2, T 1, T 3. Of course, it is not conflict-equivalent to T 3, T 1, T 2. Because, if you try to get T 3 T 1 T 2 then you will basically these are conflicting pairs of operations you will actually all you know swap them all of them you will be trying to reverse which is not possible.

This is the order in which they have been given to us in the initial schedule. So, this notion of conflict equivalence is very important one, I hope you understood the importance of I hope you understood the definition properly. So, the definition just says that focus on conflicting pairs of operation the relative order in which they come should be the same. In some serial schedule it does not matter what serial schedule it is, but it as long as that is a serial schedule then we call the given schedule as a conflict serializable schedule.

Serializable is not serial schedule it is serializable. Now you can also see that. So, if you focus on non-conflicting pairs of operations here for example, this R X this R X is not conflicting, this R Y and this R Y are not conflicting. So, this W Y is actually not conflicting with anybody. So, what we can do is to swap it with this. So, this W X comes down, this W Y will go up. That is what I mean by swapping.

So if you swap it, then the swapped schedule is actually conflict-equivalent to the given schedule that you can see, right? Suppose I do a one swap, I take this W Y here and then put W X here. This W Y is not involved in any conflicting pair of operations. And so, it does not disturb any relative order of any conflicting pairs of operations in a given schedule. And so, it will be equivalent to the given schedule.

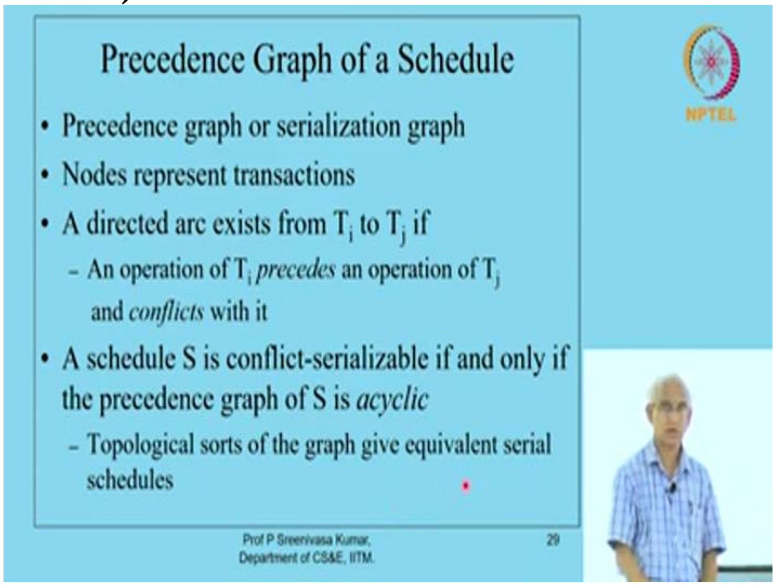
So like that, as long as this does not conflict with the immediate operation, to which I am swapping, I can actually swap it all the way up like this. And continue to maintain equivalence of the schedule before and after swap. You get the point. So like that now you can see that I can actually swap R Y with this R X, I can bring the R X down and then take R Y up. It does not change the relative order of any conflicting operation.

So I will continue to do. So what I can do is actually I can do many things like this. So for example, I can take this R Y, swap it down all the way here. I can move it all the way down here, because it is not conflicting with this. So I can move this up and this down. It is not conflicting with this so I can move this down and this up like that. I can actually swap it all the way.

Then what actually happens? So, you can see that this schedule has become a serial schedule. Why? All the operations of T 1 coming then the operations of T 3 coming, I have brought this down all the way here. So, operations of T 2 are coming later. So it is become equivalent to it has been turned into equivalent schedule. So, why I am telling all this event one way of checking whether a given schedule is in fact, equivalent to conflict-equivalent to some conflict-equivalent and conflict serializable.

Given schedule is conflict serializable is to try and focus on non-conflicting pairs of operations within the schedule and swap them around to ensure that the given schedule can be turned into some serial convenient serial schedule. So, in this case actually you can later verify that you can turn this S 2 into either T 1,T 3,T 2 or T 1, T 2, T 3 or T 2, T 1, T 3 all these things you can do by just moving the non-conflicting pairs of operations swapping. So, that is the notion of conflict-equivalence.

(Refer Slide Time: 48:38)



Precedence Graph of a Schedule

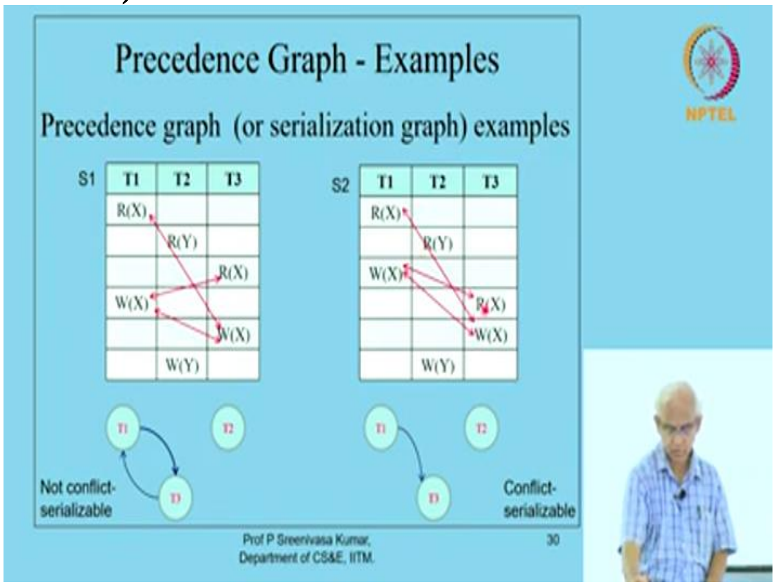
- Precedence graph or serialization graph
- Nodes represent transactions
- A directed arc exists from T_i to T_j if
  - An operation of T_i *precedes* an operation of T_j and *conflicts* with it
- A schedule S is conflict-serializable if and only if the precedence graph of S is *acyclic*
  - Topological sorts of the graph give equivalent serial schedules

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 29

So, in order to actually check whether something is conflict-equivalent or not, we can actually construct what is called a precedence graph of a schedule. Precedence graph is also called serialization graph. So, the nodes represent transactions here and there is a directed arc from the T i to T j, if an operation of T i precedes the operation of T j in the schedule, precedes means comes before and conflicts with it.

If that is the case, we can put a arc. So essentially we are actually again representing the conflicting pairs of operation by arcs. And then we can say that we can redefine this conflict-serializability in terms of this precedence graph saying that S schedule S is conflict-serializable if and only if precedence graph is acyclic. The precedence graph is acyclic then we can define that the schedule is conflict-schedule. In fact, you can see that the different topological sorts of the acyclic graph will give you equivalent schedule.

(Refer Slide Time: 50:08)



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)	

Not conflict-serializable

Conflict-serializable

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

So, I can show you the picture for these 2, just know we consider these 2 diagrams right. So, you can take this diagram T 1, T 2, T 3, this is the original diagram that I gave you and this is the slightly modified diagram. So, we noticed that this slightly modified diagram is actually conflict-serializable whereas, the original was not conflict-serializable. So, you can also check it by constructing the precedence graph here.

So, the precedence graph here has T 1, T 2, T 3 as nodes representing transactions. Now, if there is a conflicting pair of operations and one precedes the other then there is an arc from T 1 to T 3 you can verify that later in the definition. So, because of this T 1 to T 3 arc comes into picture. Because of this, this conflicting operations and this precedes that T 3 precedes the operations of T 3 precedes that of T 1.

So, because of this, this arc comes. So, the cycle comes into the picture. T 2 is of course cyclic independent. Whereas here you can see that because of this swap that we have done this all these things contribute to exactly one arc. There are conflicting pair of operations, but this T 1 precedes T 3 in that this also T 1 precedes T 3 and here also T 1 precedes T 3. And so actually all of them contribute to this arc but it is like this.

So you can see them. So using precedence graph or serialization graph, one can actually check whether a particular schedule is in fact serializable or not. So there are multiple ways of doing this serializability conflict and Serializability check. One is by constructing the precedence. Now, there are one other one is by checking whether it is actually equivalent to a serial schedule. And that you can do by actually swapping non conflicting pairs of operations and then reducing the given schedule to some serial schedule. So, these are the ways in which you can check serializable.