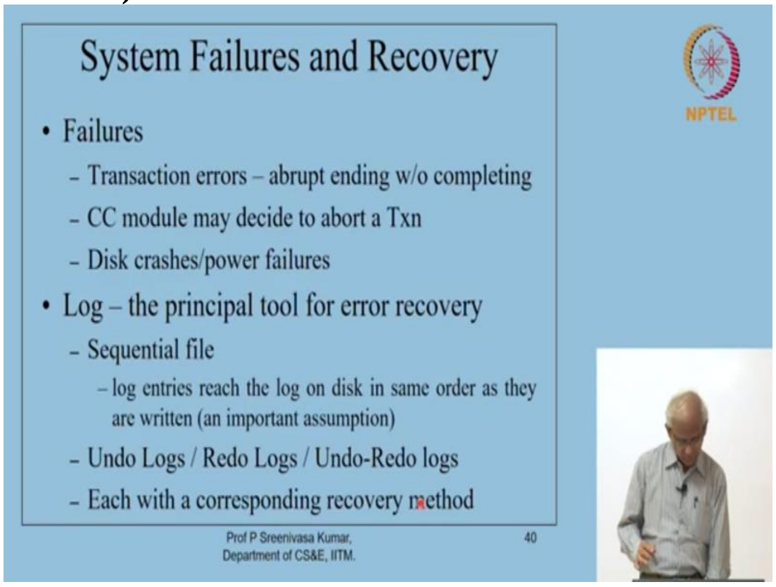
Database Systems
Prof. Dr. Sreenivasa Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 39
Recovery using Undo Logging Method

(Refer Slide Time: 00:26)


System Failures and Recovery

- Failures
  - Transaction errors – abrupt ending w/o completing
  - CC module may decide to abort a Txn
  - Disk crashes/power failures
- Log – the principal tool for error recovery
  - Sequential file
    - log entries reach the log on disk in same order as they are written (an important assumption)
  - Undo Logs / Redo Logs / Undo-Redo logs
  - Each with a corresponding recovery method

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 40

So in the last lecture, we were looking at the issues in concurrency control. And we established a notation for representing the interleaved operations of a bunch of transactions, we will call them as schedule and then we looked at various kinds of schedules and then define what is called a serializable schedule and in particular, what are called conflict serializable schedules.

And conflict serializable schedules are actually desirable schedules because they are in sometimes equivalent to a serial schedule and then we looked at how do we achieve a conflict serializable schedule in practice if you are using lock based concurrency control. So a very important concept in this context was this 2 phase locking protocol and if we are using 2 phase locking protocol.

It can be guaranteed that a bunch of transactions that are following this protocol will be such that the interleaved operations of all these transactions will need be a conflict serializable schedule and so, the concurrency can does be controlled to desirable need. Now in this lecture beginning this lecture, we will focus on how do we handle system failures, failures of various kinds of transaction errors.

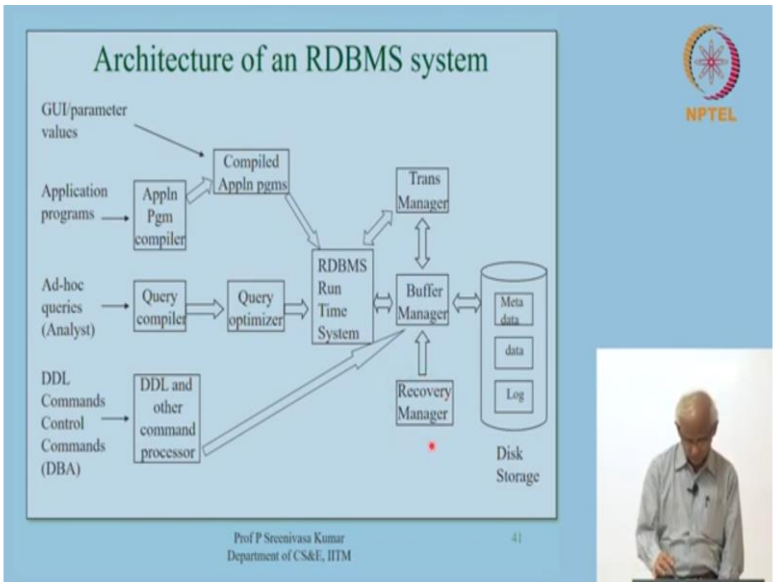
And then the concurrency control module might sometimes decide that a particular transaction is not proceeding as per the requirements. So, it might actually abort a transaction and there could be disk crashes and power failures and all that. So, in the face of all these things, remember that we have a commitment in the transaction system that we have something called atomicity and durability.

Atomicity means we should ensure that all the operations of the transactions or all the operations transaction are done in their entirety. And durability means that if the transaction is completed and how exactly declares that it is completed, we will see as we proceed in this chapter so if the transaction is admitted as completed, then whatever the changes that it has done should be permanently recorded in the database.

So there is our durability requirement. So, no, the error recovery module is what if in charge for this particular task and let us see how exactly this is done. So, the principal tool for achieving always is what is called a log. A log is basically a sequential file, log event sequential file, where new log entries are actually appended to the end of the file. So you have a sequence of log entries, we keep on appending these log entries to the end of the file.

And then that some part of the file will actually be on the in the memory. Some part of the file will be on the disk and the log entries are made first in the memory, but they reach the log on the disk in the same order in which they are written in the memory. So that an important assumption we make. And so, we will examine various kinds of case logs, which are actually called undo logs, redo logs and also called undo-redo log and in all these contexts. We will study how exactly error recovery can be achieved. So let us proceed with first giving you an idea about what are these log entries that are there.

(Refer Slide Time: 04:45)

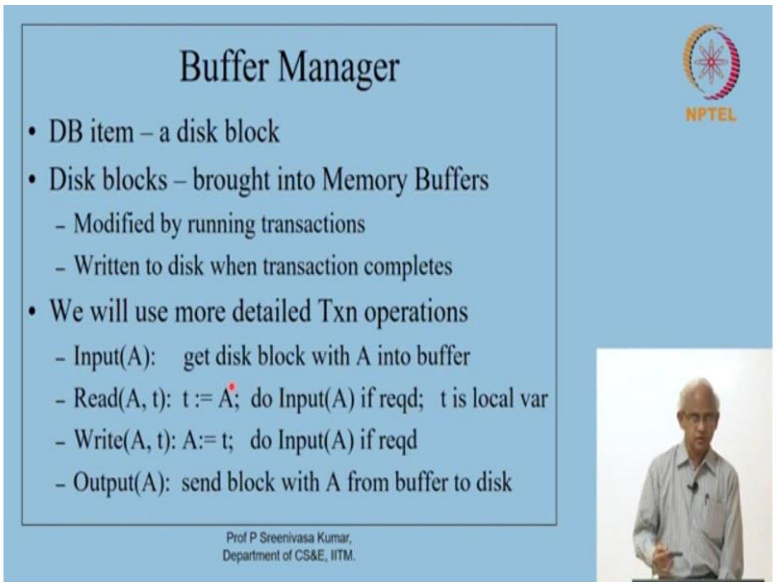


So recall that here is our architecture of the RDBMS system. So we have this part of the was all done in the early part of the course. Now we are actually focusing on the back end, where transaction manager, buffer manager, recovery manager, they are all interacting with this the data on the disk and the data on the disk is there is data there is metadata and there is a log. So it is this the log is actually maintained here in stable storage.

So we will see exactly how so the data the database data, which is available in logs, which is the buffer and is made available to the RDBMS runtime system, which is actually running all these transactions etc. So there is a transaction the transaction manager is the one that is actually watching all the schedules and then will enforce the concurrency control. So, the recovery manager is the one that that is watching are as to happen and if errors happen what to do?

So, recovery manager is the first subsystem of the RDBMS entire RDBMS system that gets the control of the whole system when the system is restarted after some error or a crashes occur, so this is the first system that gets hold of it and then it restores the database system to a consistent state. And then start submitting transactions to be performed. So recovery manager is the one that gets the first control of the whole system.

(Refer Slide Time: 06:46)



Buffer Manager

- DB item – a disk block
- Disk blocks – brought into Memory Buffers
  - Modified by running transactions
  - Written to disk when transaction completes
- We will use more detailed Txn operations
  - Input(A): get disk block with A into buffer
  - Read(A, t): t := A; do Input(A) if reqd; t is local var
  - Write(A, t): A:= t; do Input(A) if reqd
  - Output(A): send block with A from buffer to disk

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

So now let us look at so in our discussion in this set of lectures, we will assume that the DB item is actually a disk block. So this also affects as to what is the lockable item. So we will also assume that the lockable item, minimum thing that you will lock is actually a disk block. So notice that this is actually a lot of data because disk block will contain a lot of data. So if you are locking that, unless you release the lock on nobody else will be able to access that.

So but so we will, there are larger issues to be there are other issues to be considered when we have even smaller units than in disk block that are lockable. But, that introductory set of lectures is difficult to go into all those details. So we will make a simple assumption here that we have a DB item which is a disk block and disk blocks are locked in a shell. So they are brought into the memory buffers and they are modified by these running transactions and then they are going to be written to the disk when the transaction completes.

So, here in this particular lecture, when we are discussing the specific recovery methods, we will see exactly when transactions when transaction when these disk blocks reach the disk. When the blocks in the memory buffers actually reach the disk will see that exactly. So, in order to model and understand these intricacies, let us introduce a little bit more detail transaction operations. So, let me call this input A is an operation where disk block item A, whatever we are trying to actually modify is brought into the buffer space, memory buffers.

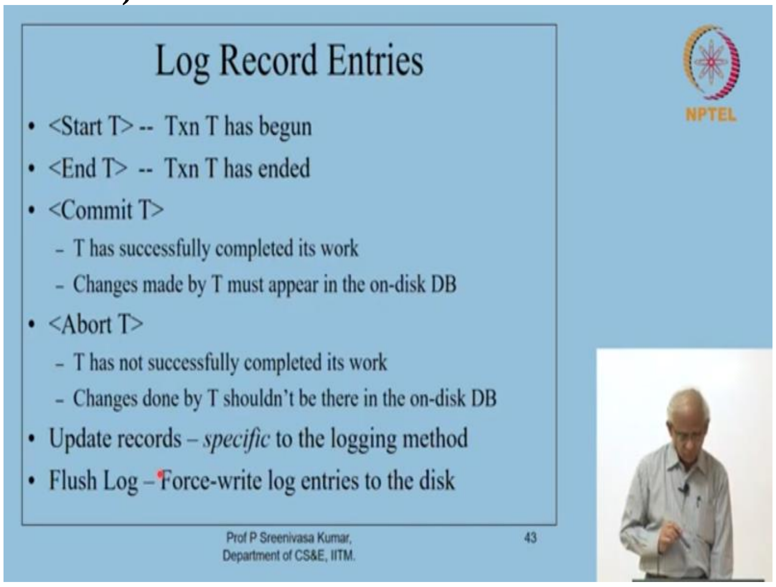
Then, read A t t is a local variable that is a variable of the transaction program. So let us assume that there is an operation called read A t where the t the local variable is assigned value A whatever is there on the in the database disk database block. So, if you issue a read A t, then in case the block that contains A is not there in the memory buffers, then input A will be required to be done that will be automatically done.

Let us assume that so input A will be done. If you are if you simply issue a read A t, then if A is not there, in the pages that are there in the memory buffer then input A will be done. When you do for input A, that particular disk block that contains this item A will be brought into the memory buffers. In a similar way, we will say write A t so, basically what this says is that whatever the change so all the changes by the transaction are at happening of the local variables by memory the variables of the program the program variables.

So when we say write A t that what we mean to say is that now the memory buffer item A can now be updated with the value of the t t has been changed by the transaction. So and in case this is actually not there, you may have to input do an input A again in case this item is not available in memory buffers will have to do input A. And then finally, we have output A that means send the block that contains this item A to the disk.

This is a very crucial operation this is where exactly data is actually reaching the disk now you are watched these output operations carefully. So, with this little bit more detail model, let us now try to understand what exactly happens when we do logging and when we use various recovery procedures.

(Refer Slide Time: 11:19)



Log Record Entries

- <Start T> -- Txn T has begun
- <End T> -- Txn T has ended
- <Commit T>
  - T has successfully completed its work
  - Changes made by T must appear in the on-disk DB
- <Abort T>
  - T has not successfully completed its work
  - Changes done by T shouldn't be there in the on-disk DB
- Update records – *specific* to the logging method
- Flush Log – Force-write log entries to the disk

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 43

So here are the log record entries. In this slide I will give you log entries that are common to all the recovery procedures, there are a few there is one important log entry that is specific to a particular logging method and we will examine that as we consider those methods. So, start T so T is some transaction id so start of T is a log entry that says that so and so transaction has begun its operations.

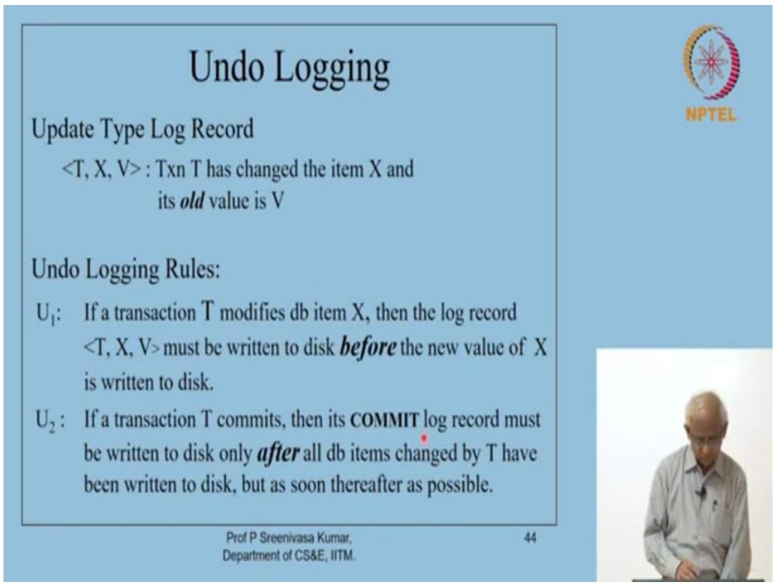
And End T is the so and so transaction has ended its operation. Commit T, T has successfully completed its work on changes made by the T must appear on the disk. That is what Commit T means. So, this is also a log entry now, it will be a specific log entry. Abort T so, T has not successfully completed its work and whatever the changes done by T should not be present in the database on the disk.

So, this is the that is our intended meaning for abort T, but abort T is basically a log entry. And then there are what we called update records. So, these update records basically are made while the transaction is actually updating the variables. So I will show you exactly what those these are specific to the particular logging method that they are using and so we will exactly see them as we look at the specific methods.

And then finally, there is called Flush log. What this says is that remember, the log also is a file. And so it also has blocks and some of the blocks of the logs with the last part of the log, are there in the memory buffers? But then it is important for us to ensure that this log is permanently available on the disk. So, at some crucial points, we will actually force write all the log entries to the disk.

That means we will transfer this log blocks that are there in the memory buffers to the stable storage, your secondary storage. So that operation is what is called flush log. So we will use all these operations and illustrate how exactly logging I mean error recovery logs. So let me first take you to one of the logging methods that we will use that can be used.

(Refer Slide Time: 14:12)


Undo Logging

Update Type Log Record

<T, X, V> : Txn T has changed the item X and its *old* value is V

Undo Logging Rules:

U_1: If a transaction T modifies db item X, then the log record <T, X, V> must be written to disk ***before*** the new value of X is written to disk.

U_2: If a transaction T commits, then its **COMMIT** log record must be written to disk only ***after*** all db items changed by T have been written to disk, but as soon thereafter as possible.

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 44

Now it is called undo logging. the name of the logging method is undo logging. So from the name, it is kind of clear as to what will going to the principal thing that this what particular method does is undoing. That means if some transaction has not completed its work, whatever it has done will be undone. So, that is the basic thing. We will see how exactly, we will do that.

So the now we can talk about all the log entries that are there, so all those log entries that have shown begin T and you know, can start T end T and commit T abort T the all those things are there. In addition, there is what is called an update type log record that update type log record the under this undo logging record will be like this, there is a transaction id and X is the database item and V is its old value.

So we make a note about the old value. Transaction T has changed the item X and its old value is V. So that is the meaning of this update log record. Now, with this, we have a bunch of rules for actually doing the undo logging. So the undo logging rules the 2 of them. So if a transaction T in at any point of time modifies the db item X, then the log record T, X, V must be written to the disk.

Before the new value of X is written to be disk, this is the principle that will happen. So, this who enforces this? It is the concurrency the recovery module concurrency control; the transaction processing models have ensured this thing. So this is called undo logging rule U 1 so what it basically says is that, keep this log entry, ensure that the log entries written to the disk before you change the item on the disk.

Change the actual database on the disk. So this transaction these entries, log entries will be first made in the memory buffers. And so, but let us suppose you have come to a state where you can actually start updating the database on the disk. So before you do an update on the database, you must do ensure that all the log entries have actually reached the disk. So that is what this U 1 means.

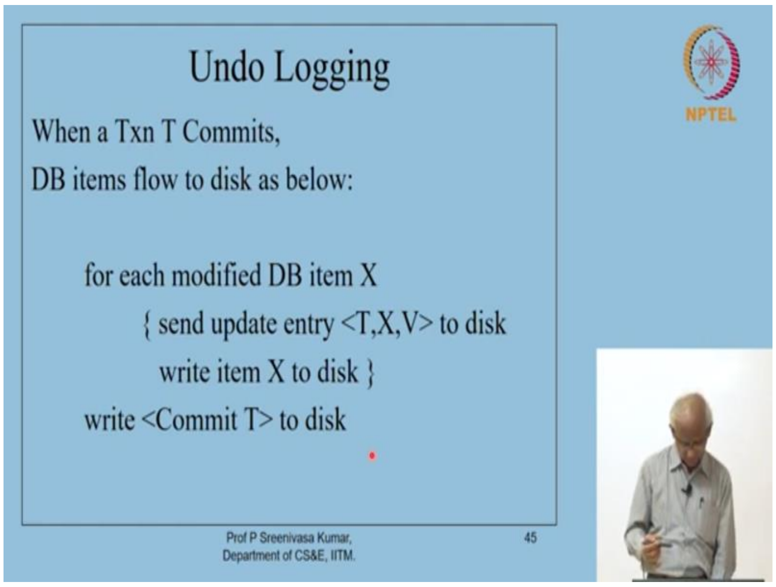
Now, so this is about commit, if a transaction T commits, a transaction T has requested for a commit, then what we do is this commit log record must be written to the disk only after all the db items that have been changed by this particular transaction have already been written to the disk. First, ensure that all the changes that are done by this transaction to the database items have all been written to disk.

Before you say that before you actually write this commit T log record on the disk. So, what after you do all those changes, then soon thereafter, you can write this log record. Now actually, it is very crucial that the log record of this particular transaction actually reaches the disk. Because, when a crash occurs, what the error recovery module will see is only the log and from that log it will try to figure out what exactly has happened just before the crash.

So, if the commit T log record is actually there on the disk, then it knows that this particular transaction actually has committed otherwise it will treat it as incomplete transaction. So is these principles clear now, the undo logging method, the update log got will maintain old values. The 2 rules here that before you actually transfer the this block that contains the modified value of X to the disk.

You ensure that the log record is written to the disk and also before you before you agree for the commit of this particular transaction, ensure that all the db items have been actually written to the disk. So that is what these 2 rules mean. So, as you can see here committed transactions are actually perfectly safe because you have already written whatever the work that has been done by a committed transaction to the disk. And only then you know, written this commit T log record to the disk.

(Refer Slide Time: 19:58)



Undo Logging

When a Txn T Commits,
DB items flow to disk as below:

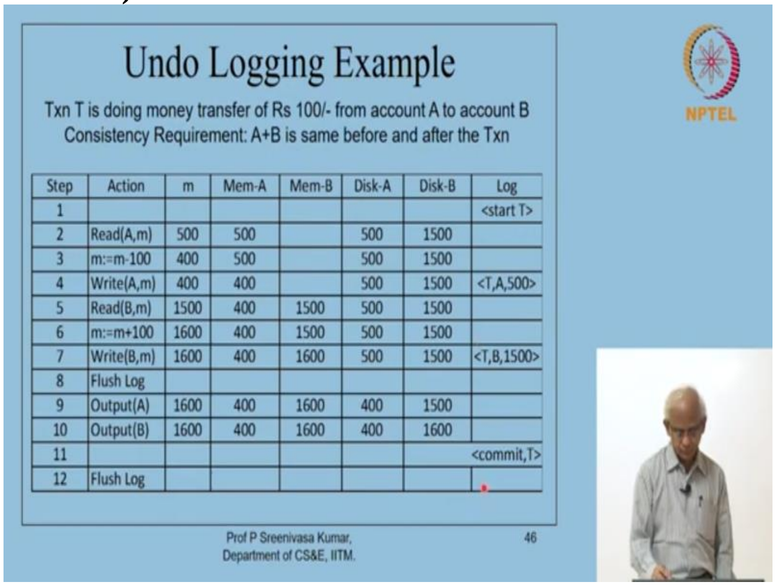
```
for each modified DB item X
    { send update entry <T,X,V> to disk
      write item X to disk }
write <Commit T> to disk
```

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

45

So, in some sense the when a transaction T commits the database items, you know, it changes would flow to the disk like this right for each modified db item, the update entry T, X, V is sent to the disk and then the write item will be done and then commit T will be done, after all these things have been the commit T will be written. So, let us know take a concrete example and then look at the actual sequence of operations and then also kind of look very carefully as to what should be done if a crash occurs at what point of time in the sequence of operations.

(Refer Slide Time: 20:49)



Undo Logging Example

Txn T is doing money transfer of Rs 100/- from account A to account B
Consistency Requirement: A+B is same before and after the Txn

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,500>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1500>
8	Flush Log						
9	Output(A)	1600	400	1600	400	1500	
10	Output(B)	1600	400	1600	400	1600	
11							<commit,T>
12	Flush Log						

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

46

So, for this let me take a running example, where we have a money transfer happening of some hundred rupees from A's account to the B's account so, the consistency requirement that we have is A + B should be same before and after the transaction is over. So that is our consistency requirement if both of them together how in this case, so here is a big table, so, watch it carefully.

So on the disk, disk A has 500 box. So, we have 1500 box and we are transferring 100 box from this fellow to that fellow. So, ultimately this fellow will have 400 box and that fellow will have 1600 box the sum of these 2 things will remain as 2000 at the beginning and at the end of the transaction, now actually this the transaction operations are happening here. So you can see this is the transaction. So m is a memory variable, m is a transaction variable local variable.

So it is reading A into m, m is being reduced by 100. And then it is writing A to m and then reading B from m reading B into the same variable m and then incrementing m by 100 and writing it as B and then it is doing a flush log and then outputting A outputting B into the database. So and then finally, then actually after, so, remember, the undo logging principle requires you to ensure that the log entries so as it is doing, log entries will be made,

We will see exactly how the log entries are done. So, all these log entries are actually happening. So before it does any change to the data on the disk, through this output A output B output A is remember the only place where actually the block reaches to the disk. So before it does this it does a flush log to ensure that all these log entries reach the disk and then it changes these disk items and then does a commit T. That is the protocol for commit T.

So if a transaction commits, what it should do is it should, write this commit T record to the disk, but before that it must do the changes to the database items and ensure that they are done in the disk. That this is the protocol that is the U 2. We talked about U 1, U 2, U 1 said before you do the changes, ensure that the log entries into the disk and U 2 said before you do a commit, and show that the changes done by the transaction or actually written to the disk.

So now let us look at the little bit detail actually when what are the log entries that made?
So step 1 before you start this, I am obviously just focusing on one transaction, but in general, there will be multiple transactions running, but let us just focus on one transaction first to understand how exactly recording should happen. So, step 1, we just recording that transaction T has started and it has read. So you say read A m so obviously, it will cause an input to happen.

So, whatever it is there in the disk will come to the memory buffers. So this mem-A mem-B are memory buffers, disk-A and disk-B are disk of the database, database on the disk, the database on the disk. So disk-A disk-B are database logs on the disk. So they will be brought to the memory and now m = m - 100. So, this change happens in memory. So it becomes 400 and then you say write, so this 400 will be written to this memory buffers.

The disk page on the memory. Notice that the disk database has not changed yet. Now at this stage, since the transaction has changed this item A db item A. We now make a log entry saying that transaction has changed this db item A and its old value is 500. We make a log entry now changing saying that the transaction T has changed this A and this old value is 500 new value we are not bothered about.

The new values, they are in the memory right now. Now, then the similar kind of things will happen with B, so we read B, so that will cause this disk page containing B to be brought to the memory buffer. I am made available and then it will be assigned to the same variable m and all that. So m is now updated. So this becomes 1600 and this continues to be 1500 and then the transaction says write.

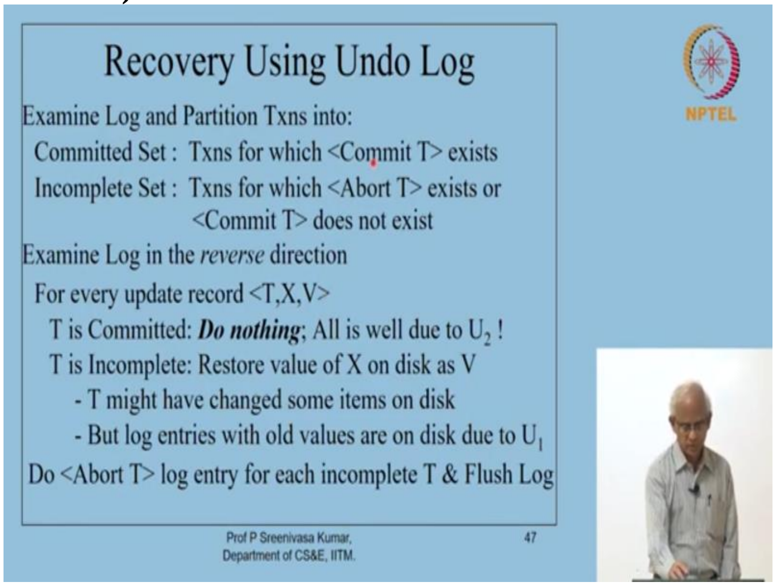
Write B using m. So now this one the buffer actually we changed the 1600. So the moment that happens we make a log entry again saying that the transaction T has changed this db item B and its old values. Now, the transaction actually is more or less done its work in memory it has done its work. So it is now ready to update the database on the disk. But, in the undo logging principle before we do any changes to the database items on the disk, we must ensure that the log entries are reach the database.

So you do a flush log. So, if we do a flush log, so, we know that these are the entries done by on behalf of this transaction and appended to the log record log buffers. So all these along with this of course, there are some other items that might be there done on behalf some other transactions also. All of them now will reach the disk the log file on the disk now we do an output A so where so this output A so that means this mem-A we will now be transferred to this B.

And then we will do the output B, so the 1600 will be transferred to disk B. And then the transaction says commit. So we make a commit log entry. And as soon as we do a commit log entry, it is in our interest to ensure that commit T log record actually reaches the disk. So we do have flush log again, to ensure that this commit T at least gone database. So this is how we do undo logging based work, transaction work will be done like this.

Of course, I am ignoring locking and all that because that is there is a separate concern. So before you do all these things, you should acquire logs and all those things. So that is the constraint from the conflict serializability point of view. Acquiring locks, releasing locks and all those things will temporarily kept that aside. We are just looking at recovery one way. Now let us is it clear as to what we have done. Now, let me take the same example and illustrate exactly what happens when a crash occurs and what are the various possibilities that we will see when a crash occurs.

(Refer Slide Time: 29:36)



Recovery Using Undo Log

Examine Log and Partition Txns into:
- Committed Set : Txns for which <Commit T> exists
- Incomplete Set : Txns for which <Abort T> exists or <Commit T> does not exist

Examine Log in the *reverse* direction
- For every update record <T,X,V>
  - T is Committed: ***Do nothing***; All is well due to U_2 !
  - T is Incomplete: Restore value of X on disk as V
    - T might have changed some items on disk
    - But log entries with old values are on disk due to U_1

Do <Abort T> log entry for each incomplete T & Flush Log

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 47

Now let us see so after the crash occurs or some system failure occurs recovery module gets the control. So what it does is to examine the log, it will examine the log that is the only thing that is available today everything in the worldwide up. So then looking at the log it will partition the transactions into 2 sets. One is the called committed set the transactions for which the commit T log without exists on the log.

And the incomplete set the transactions for which the start of the transaction exists but either abort T exist or commit T does not exist. So, we know that transaction has started but commit T does not exist. So, we treated as an incomplete transaction and suppose abort T exists then we also we treated as incomplete transaction. It is possible the transaction itself, decided to abort. So, in which case it will make an abort T log entry.

So, in either case we will treat that as an incomplete transaction. So, a set of all incomplete transaction ids and the set of completed a committed transaction ids, we will formulate by looking at the log, the log is on the disk. So we will assume that disk has not crashed, the system has crashed, the disk has crashed. So in the worst case of disk crashing, we should ensure that the log file actually is available to us safely.

So, that is another level of, you know, precautions that we have to take. So, while writing the disk onto the file, we probably should make multiple copies so that they are not actually in a graphically same position, things like that there are probably 2 different places. So, we should ensure that a copy of the disk copy of the log is always available along the disk copy. Now, we will examine the log in the reverse direction why in the reverse direction.

Because we are doing undoing. So, undoing work of some bunch of transactions has been already done. So, we are undoing. So, we should proceed in the reverse, we remember the log is in a sequential file, the changes made by all transactions are available in sequential order in a time sequence order. So, we must undo them in the reverse direction. So, we will start seeing this update log records.

So, every update log records a PC, we will see whether the transaction T is committed or not. If the transaction is committed, that means commit T log record is actually there on nothing. So, we actually have to do anything about it. You do nothing about that transaction because all is well. Why? We had U 2 there like that the abort the undo logging principle to U 2 said that you make all the changes to the database and only then write commit T.

So, if we have already found a commit T record and classified this transaction as a committed transaction, then we are very sure that it has actually updated the database and then made everything I know as per requirements. So we should we do not need to do anything about the transactions. Now, suppose T is incomplete, then what we should do is restore the value of X on the disk as this V remember.

What are this V these are old values. So what is the case here, the transaction is incomplete. Then how will it be possible? The commit T log record is not there. So it is possible that the transaction actually has finished all its work and started writing. It changes onto the disk. And some of the changes might have read the disk. Some of the changes might not have read the disk.

We do not know exactly what happened, we will never know exactly what happened. But fortunately for us, because of our principle, saying that if before you update anything on the disk, write the log entries into the disk, the log entries will contain the old values are available to us. So using these old values will basically restore the database. We will assume that the transaction has actually changed.

The items and we will simply restore them to the old values because we do not know how many of them it has changed how many of them it has not existing in the memory buffers and they all vanished. So we do not know the new values we know the old values. So we will restore the value of X on the disk as this old value V and all this old values are available to us because the log entries have reached the disk and then for each of the T transactions.

We will do an abort T log entry and then and do a flush log, so that, we know that the transaction has been successful aborted. And so, once we complete all that we can actually then resume our operations, admitting new transactions we started. So we, at the end of this, we will be sure that the databases restore to a consistent state and so we will start admitting new transactions to come into this. Notice one thing that while you are recovering, the system again might fail, the system might fail while you are recovering also.

But fortunately for us these recovery methods are item potent, that means you can actually do the same thing again without getting the same effect. So if in case you are done, some part of this does not matter. You come up and then again do this division of committed transactions and incomplete transactions and carry over this entire process. Of course we made, I am crossing over many final points about that in sense that we are actually examining the whole log till to the beginning of the log entries.

It is also possible to kind of introduce what are called some checkpoints on the logs. And then ensure that you do not do too much work while recovering. You go to the last checkpoint that those logs are called additional features will be there they are called checkpoint logs. We are not considering checkpoint or logs here. We can send some simple logs, not logs. Now let us look at the same example and then let us see exactly so there it is the same thing, as we have seen in the previous slide, but I just made this flush log entries red.

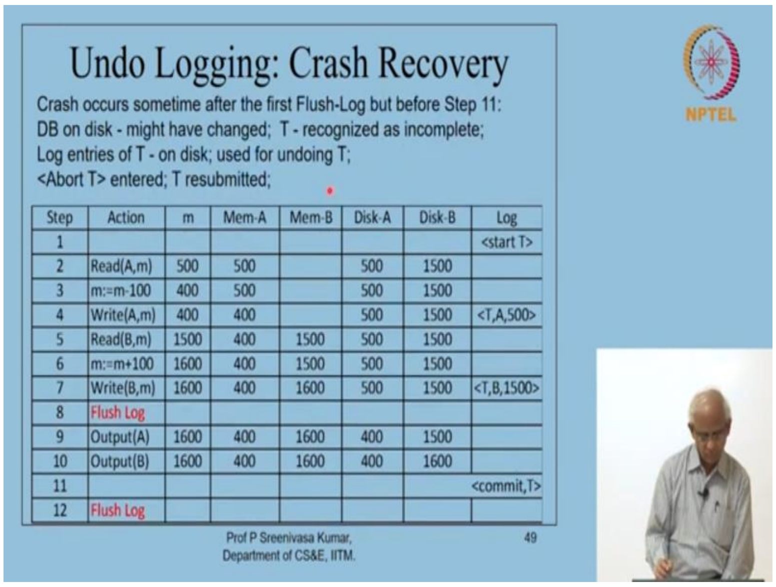
So that so let us say let us consider the cases like crash occurs sometime before the first flush log before the first flush log, which is this sometime somewhere here that you know crash occurs. And so when we recover, we do not see anything about this transaction T. But fortunately, nothing on the disk has changed, because the crash occurred before the first log. So, the output A is occurring is actually happening after this flush log.

So if somewhere here the crash has occurred. The database on the disk is basically not changed at all. This 500, 1500 will continue to be there. So T has actually been recognized as incomplete and we are not really sure whether the log entries of T have reached the disk or not. Because he crash occurred somewhere here, even before the crash log. So, these things might have reached the disk, because some other transaction was actually doing a flush log.

And so, these things might have reached the disk in case these things have reached to the disk, we can make use of them and simply restore the old values anyway. If you write back 1500 on 1500, nothing happens, so we can always write that. So the old value need not be changed actually here because nothing actually happened on disk. But in any case, we will simply restore the old values using this log in case the log entries are available.

So you ensure of course, that T is resubmitted because this work has to happen transfer has to happen. So, you enter abort T and then T. So, you enter a abort T and then resubmit T. Now let us assume the I am changing the slide.

(**Refer Slide Time: 40:13)**



Undo Logging: Crash Recovery

Crash occurs sometime after the first Flush-Log but before Step 11:
DB on disk - might have changed; T - recognized as incomplete;
Log entries of T - on disk; used for undoing T;
<Abort T> entered; T resubmitted;

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,500>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1500>
8	Flush Log						
9	Output(A)	1600	400	1600	400	1500	
10	Output(B)	1600	400	1600	400	1600	
11							<commit,T>
12	Flush Log						

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

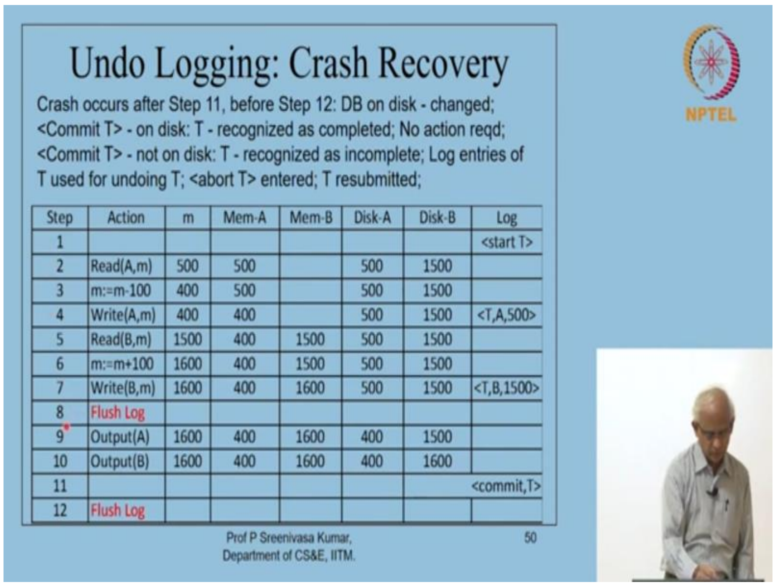
49

Say these slides are so alike each other, I tell you among change the slide. So I have changed the slide. So this first case we have considered, I have now changed the slide. So crash occurs sometime after the flush log after the first flush log, but before step 11, or step 11. Step 11 is commit T. So unfortunately, commit T record has not reach the disk. This is a really from a transaction point of view.

Which is a really unfortunate situation because it has finished all its work and in fact has even successfully updated the we do not know what that actually whether it is a updated the database on the disk or not, we really do not know. But if it does actually, if the crash has just occurred after step 10, then it has probably changed the disk also, but we are really unlucky for the transaction T.

Because we do not see the commit T log record on the log. And so, we will create this transaction T as incomplete. We will treat the transaction T as incomplete. And of course log entries of T are there on the disk because it has the crashes occurred after the first crash log but before the step 11. So, we will anyway whether these changes have actually happened on the disk or not. We are going to restore the old values and then resubmit the transaction T. So that is now let me change the slide again.

(Refer Slide Time: 42:11)



Undo Logging: Crash Recovery

Crash occurs after Step 11, before Step 12: DB on disk - changed;
<Commit T> - on disk: T - recognized as completed; No action reqd;
<Commit T> - not on disk: T - recognized as incomplete; Log entries of T used for undoing T; <abort T> entered; T resubmitted;

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,500>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1500>
8	Flush Log						
9	Output(A)	1600	400	1600	400	1500	
10	Output(B)	1600	400	1600	400	1600	
11							<commit,T>
12	Flush Log						

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

50

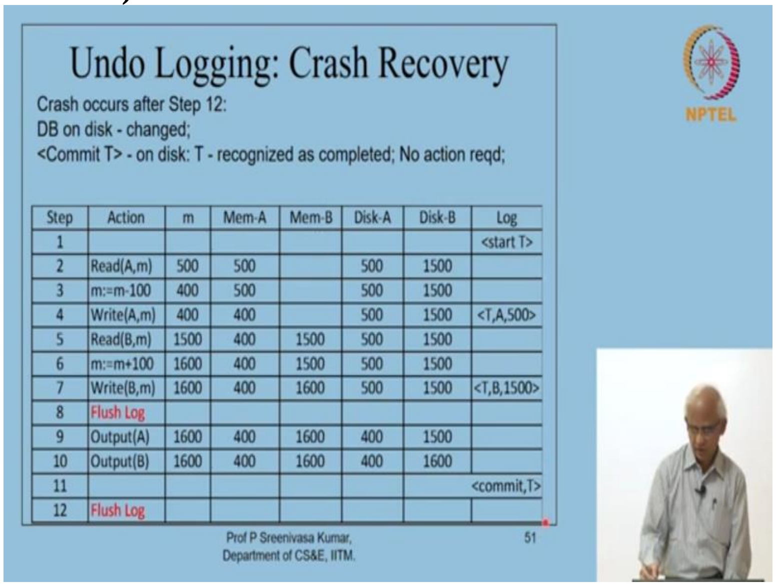
Crash occurs after step 11 just after the step 11. So, again, there are 2 cases here after step 11 and that means from the transaction point of view, it has asked for commit T because it has actually changed this and so it is asked for commit T. It will be lucky if it has commit T record actually has reached the disk. Then it will be recognized as a transaction that has actually completed and nothing will be done.

But if this commit T record has actually not reached the disk for some reason, then we will again treat these transactions with T as incomplete transaction and then use this log entries to undo the operations of the transaction and then get it resubmitted. But if the commit T record is on the disk then the T is recognized as completed and no action will be taken. So we can see now only you can actually appreciate as when the transaction actually committed.

So it all depends on this kind of login method that we are using and how the log entries are made and when the log entries reach the disk all this issues are connected. So if the log entry commit T has reach the disk, then the action is treated as completed. And then of course, from a system point of view, this is the safest thing, we need to ensure that the database is consistent.

So we cannot take any risks about these things. So we put in very strict rules saying that unless this commit T record is actually write to the disk we cannot create this transaction and we will simply undo its operations with the help of log entries. Good. So these are the various cases. So we have considered cases as to where the crash occurs before this, between this after this etc. So, I hope it is all clear now as to how to use undo logging.

(**Refer Slide Time: 44:30)**



Undo Logging: Crash Recovery

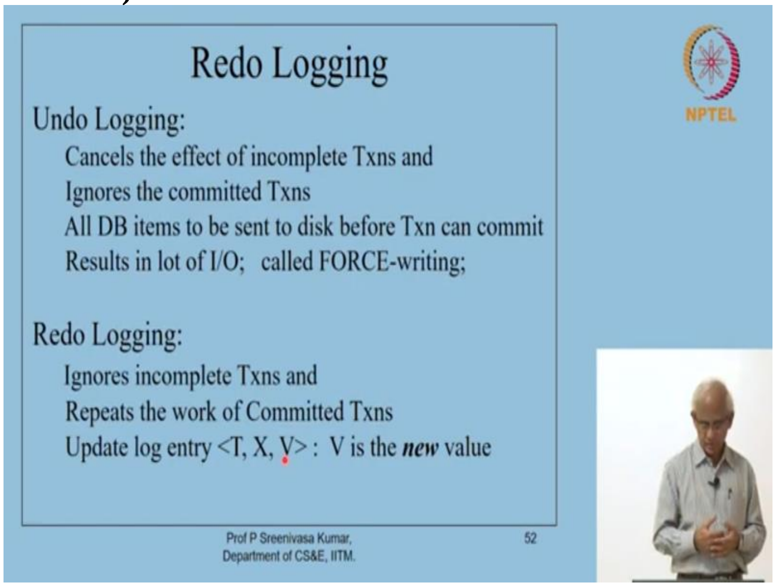
Crash occurs after Step 12:
DB on disk - changed;
<Commit T> - on disk: T - recognized as completed; No action reqd;

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,500>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1500>
8	Flush Log						
9	Output(A)	1600	400	1600	400	1500	
10	Output(B)	1600	400	1600	400	1600	
11							<commit,T>
12	Flush Log						

Prof P Sreenivasa Kumar,
Department of CS&E, IITM. 51

Crash occurs after step 12 that means flush log is commit T is already there. So commit T is on the disk the T is recognized as completed no action done for other as far as this transaction is concerned no action will be done. They will of course other transactions for which we have to do some undoing. Undoing always happens remember the undoing have happens in the reverse direction log. So, this is the forward direction log. We will start examining the update record in the reverse direction log and then start doing and undoing.

(**Refer Slide Time: 45:14)**



Redo Logging

Undo Logging:
Cancels the effect of incomplete Txns and
Ignores the committed Txns
All DB items to be sent to disk before Txn can commit
Results in lot of I/O; called FORCE-writing;

Redo Logging:
Ignores incomplete Txns and
Repeats the work of Committed Txns
Update log entry <T, X, V> : V is the ***new*** value

Prof P Sreenivasa Kumar,
Department of CS&E, IITM. 52

So that is undo logging so basically, in summary, what undo logging does is to cancel the effects of all the incomplete transactions and basically does nothing about committed transactions because nothing needs to be done about committed. So it ignores the committed transactions. Now, but one major issue about using undo logging is that all these db items that the transaction has changed.

Have to be sent to the disk before the transaction can make this log committed. So, obviously in a transaction I have finished all my work. So I wanted do commit so whatever be I will be requesting that send this page to the disk or when I make a commit T the transactions manager has to ensure that all the disk blocks that I have changed are actually written to the disk before it allows me to enter.

So a lot of IO happens at the commit time. So this can be called as force write in some sense, there always db always disk blocks have been written by force in the disk. As recognize this there is a another method which does exactly the reverse of this method. It is called redo logging. What it does is to ignore the incomplete transactions and repeat the work off all the committed transactions.

And it uses a different update log entry where it will store instead of old values, it will store the new values. It will start storing the new values instead of old values in the update log entries, then it is possible for us to redo some of the work of the redo the work of a committed transaction. So that is how redo logging works. So I think we will stop here. I will talk about redo logging in the next lecture. And then finally we will close with you know we have another method called both undoing and redoing can be adopted as principles for recording method. We will discuss that also.