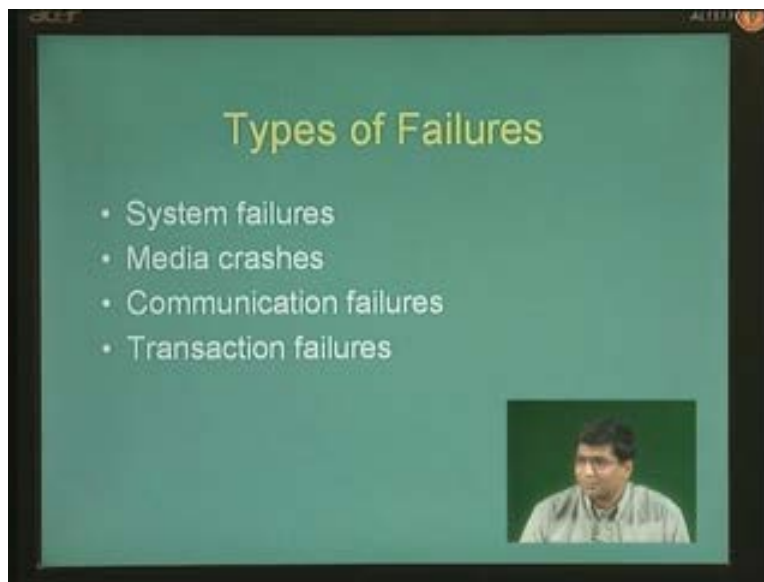


Database Management System
Dr.S.Srinath
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 29

Recovery Mechanisms III

Hello and welcome. In the previous session we have been talking about recovery techniques in database management systems. How do we recover a database especially when the database was running and the system was subjected to some kind of a failure. And we saw that there are different varieties of recovery techniques each having their own advantages and disadvantages. Let us continue with this topic further today and bring it to a logical conclusion by looking at all the other issues that entails database recovery.

(Refer Slide Time: 02:02)



Let us briefly summarize what we have studied until now. Whenever you are thinking ... (Refer Slide Time: 2:00) given computer system whether it is databases or otherwise is subjected to frequent or frequent or non-frequent kind of failures, different kinds of failures. And we could classify failures into different types like say system failures or media crashes or communication failures or transaction failures.

System failures and transaction failures are by far the most frequent or the most common kinds of failures. What is a system failure? System failure is something like suppose you are running your dbms on a machine and let's say the power goes off and your machine crashes. The characteristics of such a system failure is that all data that are there in the volatile memory like ram are lost. However data that are present on persistent storage like

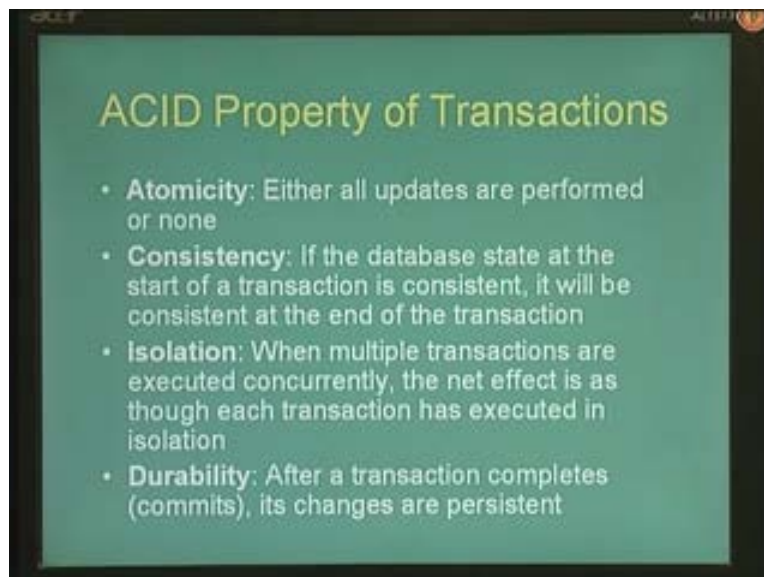
disk are still retained. So whatever has made it to the disk can still be retained, however whatever was still in the ram is lost.

Similarly there are failures like transaction failures which are again quite common. Transactions failures entail failures where for a variety of reasons given transaction is not able to complete its operation. This reasons could be something like too many processes running in this system or not enough privileges for running this transaction or the transaction trying to do illegal ... (Refer Slide Time: 03:34) transaction and so on and so forth.

And these are again quite frequent kind of failures and here again the characteristic is that whatever the transaction was doing and whatever data that the transaction had in the main memory is lost, however the data that are present on persistent storage like disk are still retained.

Then there are communication failures where especially if your database is distributed and you had to communicate between two or more different geographically distributed databases then you might encounter communication failures which need to abort a transaction or leave a transaction midway and there are somewhat infrequent failures called media crashes like a disk crash, like a disk developing bad sectors or some kind of a damage, physical damage to the disk where not only data that are there in the physical memories, in the primary memory is lost but the data that are present on the disk are also lost. That is you ... (Refer Slide Time: 04:47).

(Refer Slide Time: 4:52)



Now what is the main property that we have to consider or that we have to maintain when we are talking about recovery techniques. As we all know transactions in databases follow this well-known property of acid which stands for atomicity, consistency, isolation and durability. So atomicity essentially says that either all updates that are to be

performed by a transaction are performed or none of them are performed. This is obvious because suppose you are transferring money between two accounts, let us say you have a debit in one account and a credit in another account either both of them have to be performed or none of them have to be performed. So either the transaction should run fully or it should not run at all and so on. It's not sufficient if money is debited but not credited or vice versa.

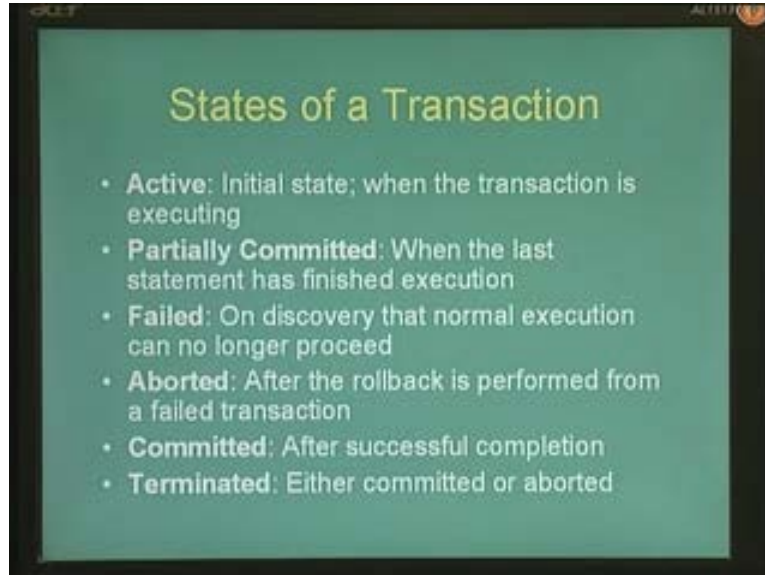
Similarly, consistency. A transaction when it completes cannot leave the database in an inconsistent state. We have seen some integrity constraints and notions of triggers where suppose data database consistency is violated, automatically triggers are enforced which will cause the transactions to roll in many commercial databases. So if the database starts from a consistent state, at the end of a transaction it should remain in a consistent state.

And of course the third property is of isolation where when multiple transactions are executed simultaneously, for efficiency reasons they are executed in a concurrent fashion that is the updates made by each of the transaction are performed concurrently. However the net effect that is not all possible concurrency is okay, essentially the kind of concurrency that is permitted is that the net effect of these multiple transaction should be such that it should be as though the transactions have been executed in some kind of serial fashion.

And the last property is about durability where once a transaction commits, once a transaction says that I have committed then whatever changes that it has made is persistent. After commit it should never be case that let us say system crashes after a transaction has committed, it should never be case that the transaction is now aborted or rolled back. Once the transaction is committed, it means that it has performed some kind of physical operation that is some, it has performed some kind of an operation that is beyond the purview of the database itself.

For example when in an atm transaction, let us say in an automatic teller machine transaction once a transaction commits it means that it is safe for the atm to dispense money... (Refer Slide Time: 07:49) after the atm has dispensed money we decide to actually abort the transaction and then say okay it has been not committed and so on. So once changes that are made by transactions are committed they have to be durable that is it has to be persistent.

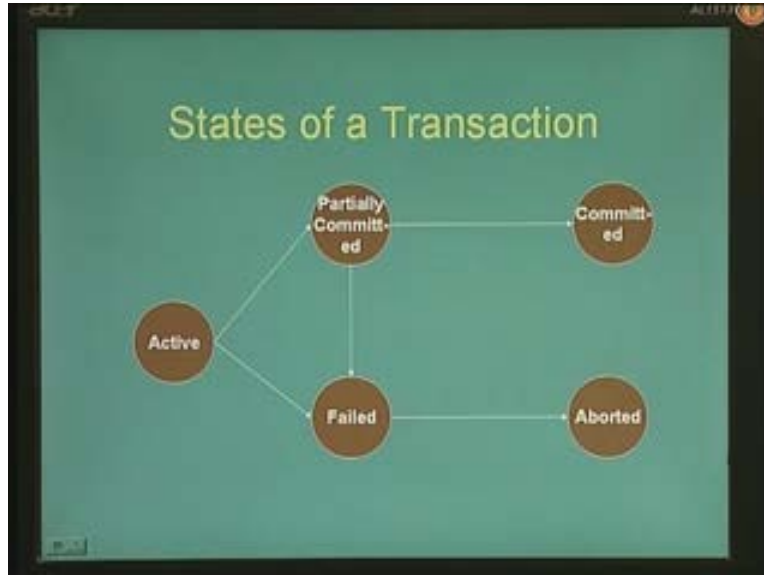
(Refer Slide Time: 8:07)



And we also saw that transactions can be in different states like active or partially committed or failed, aborted, committed or terminated. So an active state is the initial state when the transaction is executing, it's performing some operation, it feeds some data elements and made some changes and so on. And partially completed is a state when it has performed all its operations and it is ready to commit, it's not yet committed.

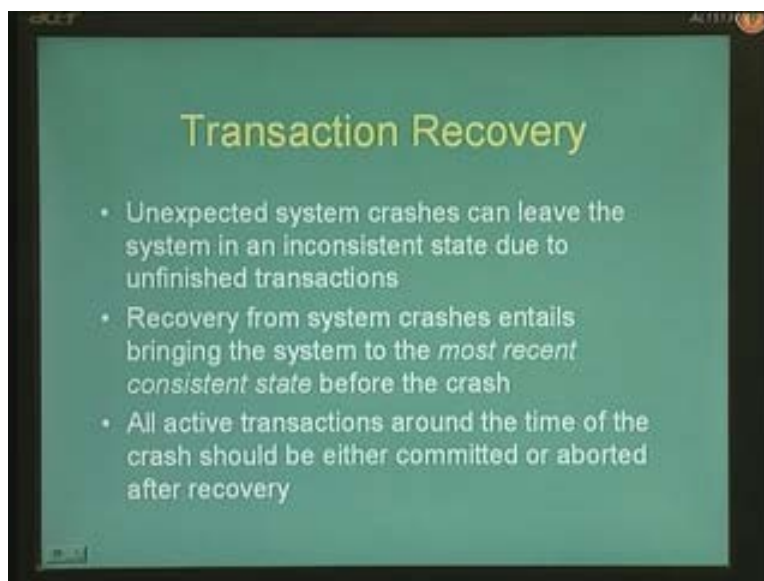
And once it discovers that it is not able to proceed further, whether it's not able to commit whether it's not able to calculate or perform its computation further we say that the transaction has failed. Once a transaction has failed it has to roll back any changes that it has made to the database has to be reverted, has to be undone and once a roll back is complete we say that the transaction has aborted. And on the other hand if the transaction successfully commits, it's a committed transaction and a terminated transaction which has something either aborted or committed.

(Refer Slide Time: 9:15)



So this slide which is now familiar shows the different states of a transaction in a schematic fashion. That is an active transaction can become either partially committed or failed transaction depending on what it has, whether it has been able to perform all its operations or it's not able to perform all its operations. And from a partially committed transaction, you can still go to a failed state when you see that it's not possible to commit the transaction. But once you see that it is possible to commit the transaction then you go into a committed state and once you come to a fail state, you perform a roll back operation where you go to the aborted state in the transaction.

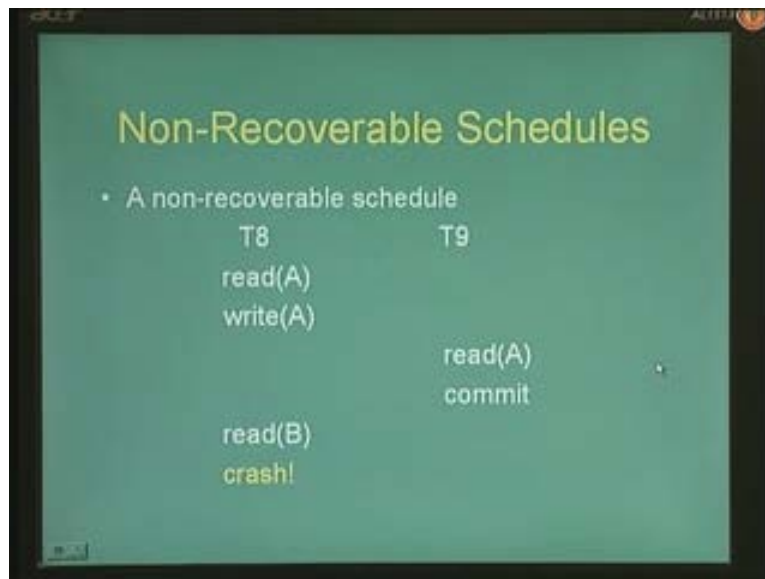
(Refer Slide Time: 9:57)



What is meant by transaction recovery once again? A transaction recovery is a process by which we recover the database system which is performing several different transactions into the most recent consistent state. That is at any given point in time, any ... (Refer Slide Time: 10:21) forming several transactions simultaneously. Now during such a time when there are some kinds of system crashes let us say some kind of failures like say system crashes or transaction failures or media crashes or whatever, it basically leaves the system in an consistent state, half of some transactions have been performed some of them have been committed, some of them have just started and some of them were shown to be committed but may not have been committed and so on and so forth.

It's essential that when we recover from this system crash, we should recover to a state that is obviously consistent. And it has to be not just any consistent state, it has to be the most recent consistent state that was there before the crash. and all active transactions around the time of the crash that is when the crash happened, there were let us say 10 different active transactions that were running, once we recover from the crash of all these active transactions there should be either in a committed state or an aborted state after recovery. That is either all the operations that are performed by the transactions are persistently stored in the database or they are rolled back so that they can be invoked once again buy the dbms system.

(Refer Slide Time: 11:46)



We also saw that there are certain pre-requisites for recovery to happen. That is there are certain kinds of schedules by which transaction activities are performed which are not recoverable at all. For example this slide shows two different transactions T8 and ... (Refer Slide Time: 12:07) following operations T8 performs a right operation on an element called A. So it reads an element A and make some changes and writes it back to the element.

Now let us say T9 reads the new element of A which has been return back by T8 and commits that means let us say displace. So let us say something like you have just updated the bank account in your bank or updated your stock prize or something like that and there is another transaction which has read the new stock prize or the new balance account and displayed it or printed a statement let us say, commit is a some kind of physical operation. So let us say it printed a statement.

However the transaction T8 has not yet completed here and while it is still going on, it encounters a system crash. And once we recover we say that T8 has to be rolled back or has to be aborted and restarted again. But what has happened here is that this transaction t nine has already said that the new value of A is so and so and it has already printed the statement or displayed it or done something of that sort. So such a kind of schedule is not a recoverable schedule that is we will not be able to recover from such a schedule.

(Refer Slide Time: 13:29)

Cascading Rollbacks

- Even if a schedule is recoverable, to recover from the failure of a transaction, there is a need to rollback several transactions.

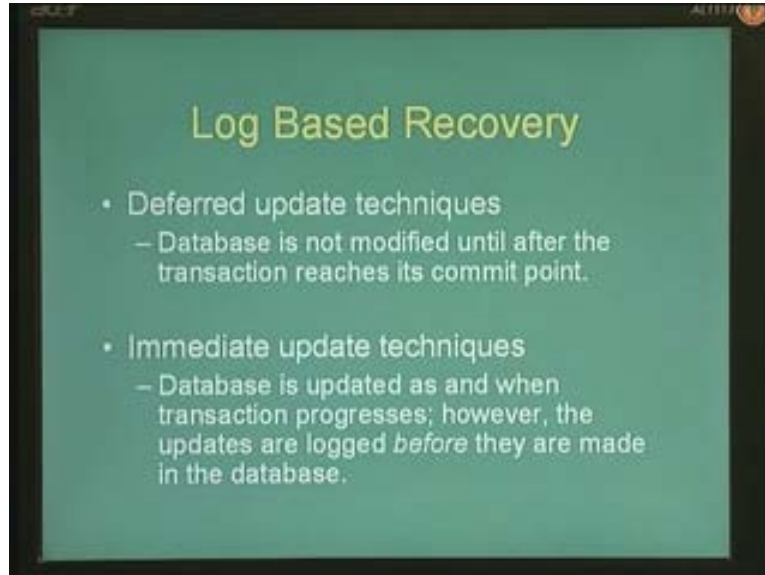
T	T1	T2
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

crash!

There is also another kind of problem with recovery namely that of cascading roll backs. This slide shows such a example here that is even when a schedule is recoverable, sometimes whenever we roll back a transaction there might be several other transaction that are waiting on it that have to be rolled back. And all of them have to roll back one after the other which causes a cascade of different roll backs.

So here there are three different such transactions T, T1 and T2 where T has read element A and return something back to here. Now whatever has been returned back by TA or by T is now read by T1 and its return back and whatever has been return back by T1 is now read by T2. But transaction T has not yet committed here and it's still going on and then it crashes. Now after we recover from the crash, suppose we decide that transaction T has to be aborted, it means that transactions T1 as well as transaction T2 have to be aborted because they have read some changes or they are basically depending upon the changes that are made by T itself. So this causes the problem of cascading roll backs.

(Refer Slide Time: 14:49)



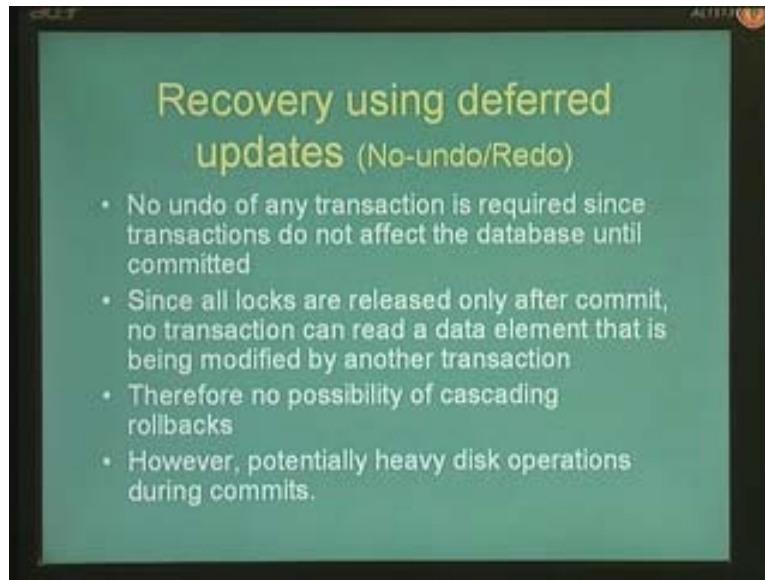
So we saw how to prevent cascading roll backs and how to prevent non recoverable schedules in the previous session which I will not go into more detail here. Essentially in order to prevent non recoverable schedules, you need to ensure that no transaction reads a data element that is written by another transaction unless it has committed, only after committing a particular operation another transaction can read this transaction. We also saw two kinds of log based recovery. What is a log based recovery? Log based recovery uses what are called as write ahead logging or what are also abbreviated as wla write ahead logging addresses, wal rather write ahead logging.

Now what is a write ahead logging? Write ahead logging essentially means that before performing any operation on the database, you perform a corresponding operation on to a log file and then perform the operation on to the database. Now whenever the database crashes, you have to use your log file in order to be able to recover from your system failures. We saw essentially two kinds of techniques based on write ahead logging. What are called as ... (Refer Slide Time: 16:16). In deferred update techniques database is not modified until after the transaction reaches its commit point. that is whatever changes that a transaction makes to the database is first just returned to the log and only when the transaction is ready to commit, will the log be return on to the or the changes be return onto the database itself.

So, until after the commit point transaction or the database is not modified and once the commit succeeds, you also enter a commit or committed entry into the log. So essentially whenever the system crash, you just need to redo your transaction so that it reaches the point where it has been. And there are other kinds of techniques called immediate update techniques where databases updated as and when the transaction progresses. However as the name write ahead logging suggests that the updates are first written onto the log before they are written onto the database. And immediate update techniques requires both

undo and redo operation that is you need to undo all active transactions which have not committed and you have to redo all transactions which have committed actually.

(Refer Slide Time: 17:41)

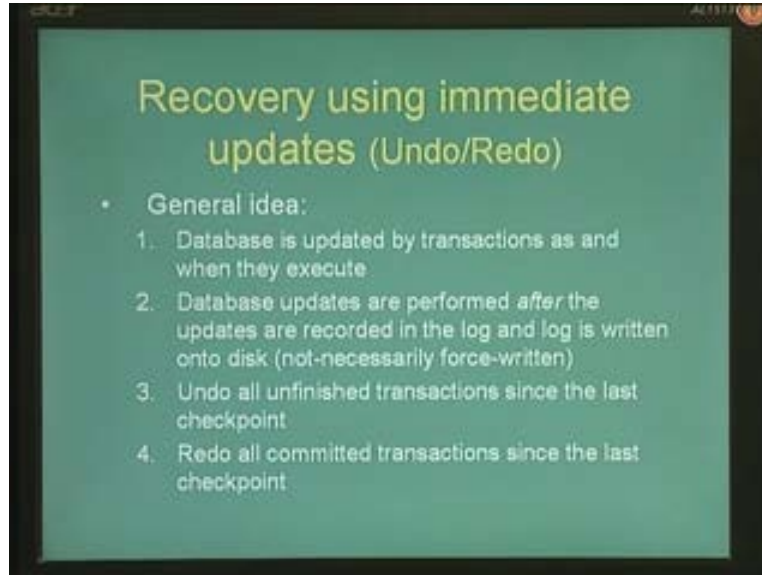


The general idea behind recovery using deferred updates is shown in these slides and such kinds of ... (Refer Slide Time: 17:48) that is there is no undo operation as part of this database, as a part of this recovery technique however there is a redo operations. Because, why is there no undo operations? Because the database is not touched at all that is the database is not modified at all until the transaction is ready to ... (Refer Slide Time: 18:12) multi user system. In a multi user database system, you have to perform let's say some kinds of two phase locking in order to prevent ... (Refer Slide Time: 18:28) in order to enforce isolation constraints. And all locks that is whenever you are holding such a lock on a database item all such locks are released only after the commit.

Therefore no transaction can read any other, can read the values written by any other transaction before it has committed. Therefore there is no possibility of cascading roll backs and also there is no possibility of a non-recoverable schedule occurring as part of this recovery mechanism.

However the problem with deferred update techniques is that suppose a transaction performs let us say some 500 different operations. Now none of these 500 different operations are written on to the database until after the transaction is ready to commit. therefore once a transaction is ready to commit, suddenly there is a huge spurt of activity, suddenly there is all 500 different operations have to be performed on to the database at one instance of time rather than being distributed over the entire life span of the transaction. Therefore there could be potentially heavy disk operations during commits which could bring down performance or which could cause the system to thrash or lead to some other such problems.

(Refer Slide Time: 19:53)



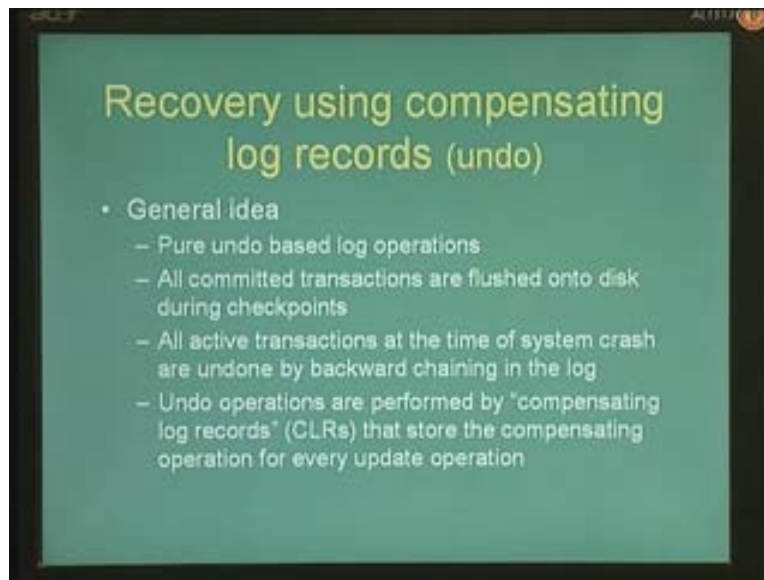
With immediate update techniques that is where locks or where the database is updated as and when the transactions are written onto logs, we require both undo and redo operations. So such kinds of recovery techniques are also called undo slash redo transaction or undo slash redo recovery techniques. Now what is the general idea behind immediate recovery techniques? The general idea is that as and when the transaction is executing, the database is updated even before committed which is very important here. That is the database is updated as and when the transaction is executing and the transaction may not commit at all, the transaction may become a fail transaction it may abort, it may have to roll back and so on. However as and when the transaction is executing the database is getting updated.

What it means is suppose the transaction cannot commit, suppose the transaction is not able to successfully perform its commit operation then we need to roll back the transaction. Now because we are changing the database as and when we are performing the transaction, we have to perform corresponding undo operation that is we have to undo whatever the changes that the transaction has made to the database.

So database updates are perform after the updates are recorded in the log and the logs are written onto disks. And of course it may not necessarily be force written onto disk that is you just write it onto log and then write it to the database. Now once you know that the log has been written to disks, it's safe for you to write to the database. Now suppose there is crash at any point in time and ... (Refer Slide Time: 21:46) to do two different operations, one is you have to undo all unfinished transactions since the last check point and you have to redo all committed transactions since (Refer Slide Time: 21:55) as you know our different stages in transaction logs where we say that it is safe to throw away all other historical data that is there in the log.

Because, the log is a monotonically increasing file where every operation keeps on adding to the log. Suppose we don't truncate the log, it could well be the case that the log becomes far more bigger than the database itself and becomes far more difficult to manage than the database itself. Therefore it is important for us to be able to conclusively decide, when at time what parts of the logs can be safely deleted or can be safely thrown away. So that is where the notion of check points also comes in to the picture.

(Refer Slide Time: 22:44)



There is a third kind of recovery techniques which we have not seen in detail but I am including this here for the sake of completeness which are called undo based of transaction, undo variety of transaction recovery protocols, we saw that is undo slash no redo. we saw that there was the deferred update technique was no redo slash undo and the immediate tech update techniques were undo slash redo and this is undo slash no redo.

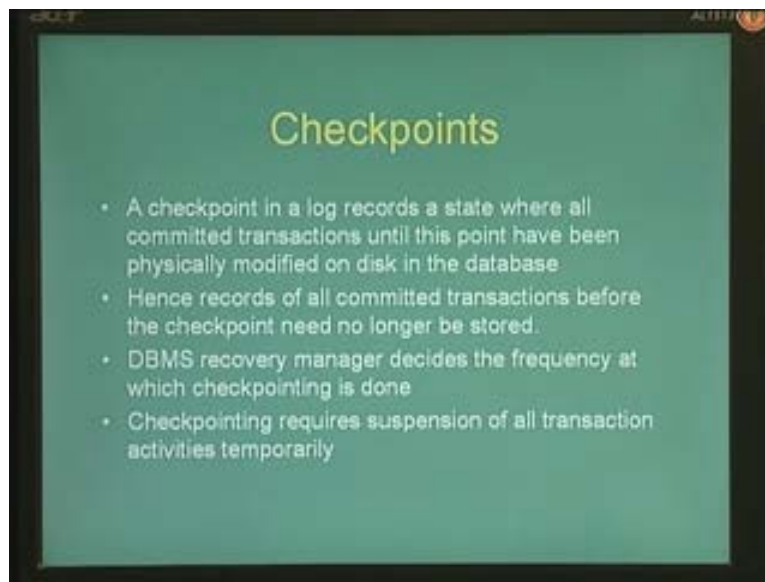
The general idea behind undo based transactions is that it's a recovery technique that is based on pure undo operation. what this means is all committed transactions are flushed onto disks during check points that is at every check point in order to safely throw away certain transactions, we flush all committed transactions ... (Refer Slide Time: 23:46) transactions are maintained that is the chain of all activities by an active transaction is maintained by what is called as backward chaining in the log.

That is whenever I enter an entry for a particular transaction in the log, I also maintain a pointer to the previous operation that it performed. therefore the way that these log entries are maintained are what are called as compensating log records or also called as CLRs where for every update operation, you perform you store what is called as a compensating operation. For example if you let us say debit 5000 rupees from an account A, the compensating transaction or the compensating operation rather would be to credit 1000 rupees or 5000 rupees to this particular account.

So for every kind of update operation let us say you deleted a particular tuple, the compensating transaction would be to insert the same tuple back to the database, the compensating operation. So you maintain the log of compensating operations and at every check point you see which are the transactions that are committed and flush them onto disk so that you don't have to undo them once again. And at any time the system crashes or we encounter a system crash, what we need to do is simply start from the state where the system has crashed and follow the backward chain of logs and perform all the compensating log records that is compensating operations that are performed.

So this is the pure undo based operation where there is no need to redo any transaction or **there is no need to redo any transaction or** there is no need to redo any particular transaction activity. The redo is performed or the flushing of transactions data are performed only at check points and not after any system crashes

(Refer Slide Time: 25:53)



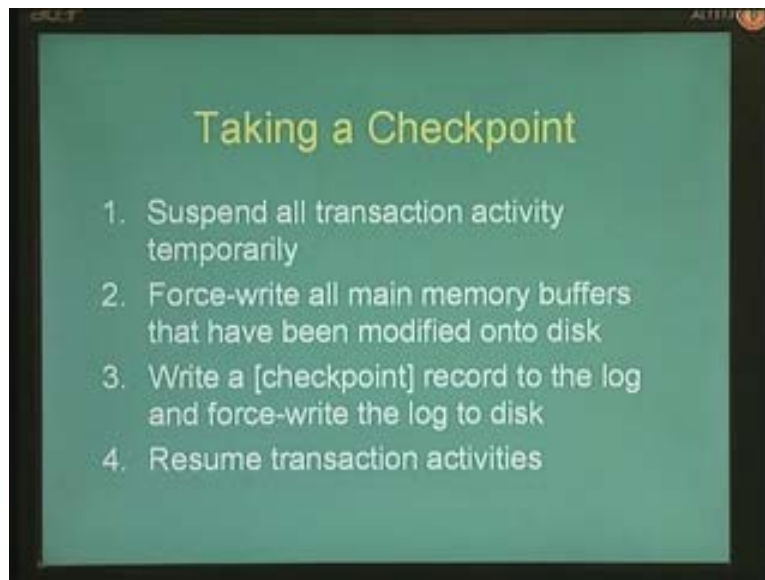
Now let us look into the notion of check points itself in a little more detail. As we said before, a check point is some point in a log where we say that it is safe to throw away everything before this check point. That is all committed transactions until this check point have been are now durable that means they have been physically committed to the database. Therefore the records of all committed transaction need no longer be stored as part of this transaction log. And it is safe to throw away all the data about committed transactions. and the frequency at which we take a check point decides or some kind of a balancing factor between what is the amount of log that you are going to store versus what is the performance over head that you are going to incur.

Because at every check ... (Refer Slide Time: 26:55) you need to bring the database to what is called as the quiescent state that means you have to suspend all the activities all transactional ... (Refer Slide Time: 27:05) transaction that have committed and ensure

that they are physically modified in the database and ... (Refer Slide Time: 27:13) restart the database system that is restart all the transaction activities.

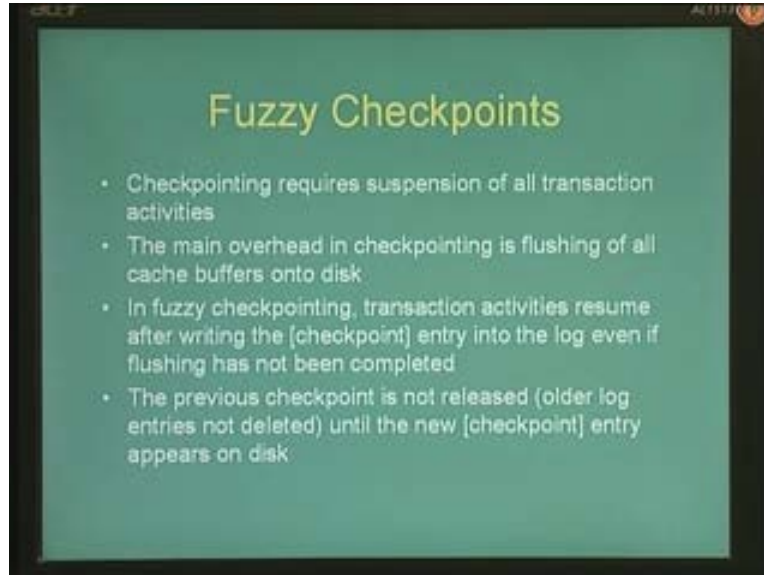
Therefore if check pointing is done at two frequent an interval, if check pointing is done too frequently then it impedes performance while if check pointing is done at too widen interval then it starts increasing the amount of log records that you have to store.

(Refer Slide Time: 27:45)



So what is the general algorithm for taking a check point? We have to first suspend all transaction activities temporarily that is we need to take the dbms to a quiescent state and then we have to force write or flush all main memory buffers that have been modified of committed transactions actually on to disk. And then write a check point record into the log and force write the log onto disk as well. And then now it is safe to resume transaction activities.

(Refer Slide Time: 28:18)



But then as we saw that bringing a database system to a quiescent state is a costly operation just imagine a large database system like let us say the database system for railway reservation let us say for a given railway zone like say southern railway zone. At any given point in time there are possibly hundreds or even thousands of transactions that are performing, so many requests coming from different places for reservations and cancellations or some other kinds of activities not just reservations and cancellations let's say rescheduling and so on.

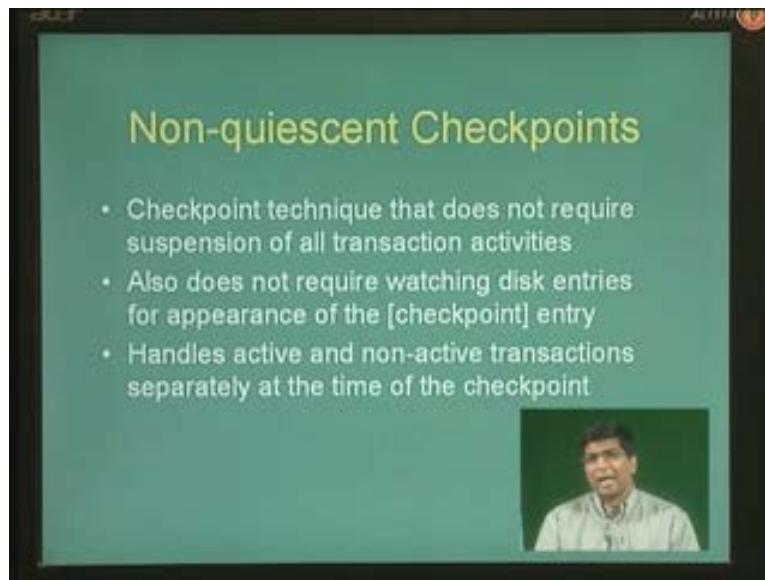
There are several different transactions that are running at any given point of time and even if we stop the database system let us say for a few seconds or even if or sometimes even a few minutes because the size of the database is so huge, it actually impedes or impairs the activities of a large number of requests that is there could be a large number of people waiting in different reservation centers waiting to reserve their tickets and bringing the dbms to a quiescent state will actually stall all of them and so on.

So there are other technique that are used which can try to obviate the need for bringing the database on to a quiescent state whenever a check point is being taken. Remember taking a checkpoint is quite important because otherwise the transaction log itself becomes too difficult to handle but ... (Refer Slide Time: 30:00) it doesn't impedes the performance too much.

That is where the notion of fuzzy check point comes into the picture. That is fuzzy check points essentially does not require, normal check pointing requires suspension of all transaction activities but fuzzy check point does not require in bringing the database onto a quiescent state. Essentially what we do in fuzzy check pointing is that we write the check point entry in to the log and we resume operations but we keep watching the disk itself that is at the physical layer.

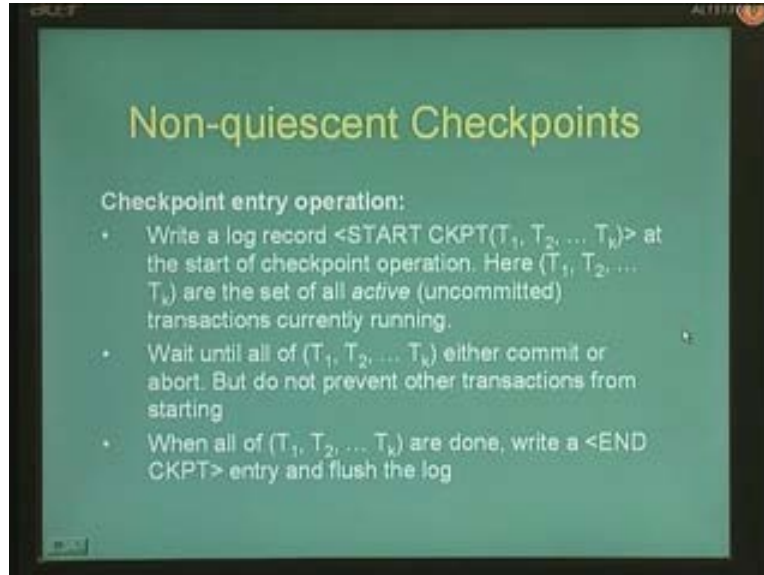
For example note that I mean note that when we write something onto disk, the operating system may not actually physically write it on to disk. It would actually write it onto a buffer cache within the main memory, so it's still volatile and it will write it onto disk at some later point in time. So we wait until that point where the check point log actually comes onto the disk and then we start removing all the old entries that is whatever has to be safely removed are then removed.

(Refer Slide Time: 31:21)



A second kind of check pointing which does not require the database to be brought to a quiescent state is also called a non-quiescent check pointing operation. Here unlike fuzzy check pointing, there is also no need to watch the physical disk sector or physical disk block to see whether the check point entry has actually appeared. But the idea here in non-quiescent check pointing is that active and non-active transactions at the time of the check point are handled separately.

(Refer Slide Time: 31:56)

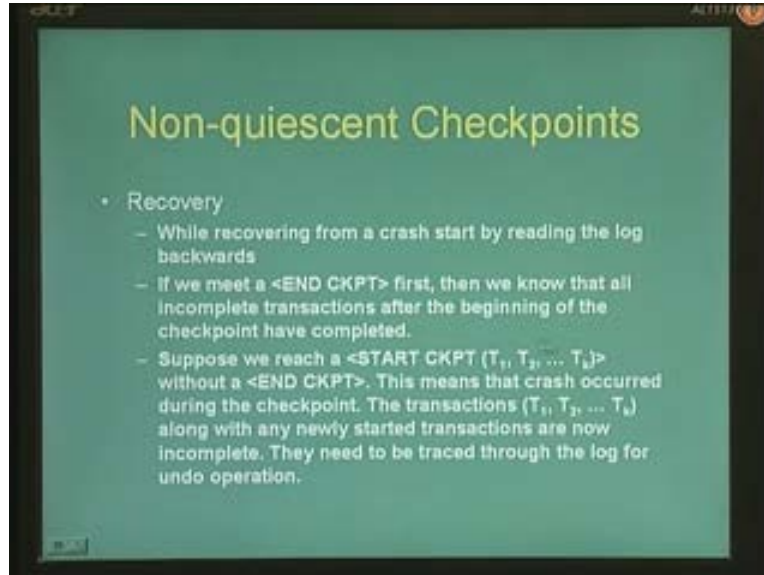


Let us look at how non quiescent check pointing actually performs. There are essentially three different steps in this non quiescent check pointing. At any time if I have to take a check point, what I do is I first determine which are all the active transactions at this point in time. That is these are all the active uncommitted transactions that are currently running and then make an entry called start check point with even to T_k which are all the active transactions running at this point.

Now this entry will go into the buffer cache and so on and so forth and finally at some point in time it will reach the disk. Now wait until and we are not concerned when they are going to reach the disk and we are not going to watch the disk at all, we just wait until each of this transaction either commit or abort. But we do not prevent other transaction from starting that is other activities could be resuming by themselves but we simply have to wait until any of these commit or abort.

When all of them are done that is they are finished or terminated, we write a end check point entry into the log and then force write the log onto disk. That is we flush the log onto disk. Now what is the use of such a check pointing? That is how can we recover from such a check pointing?

(Refer Slide Time: 33:18)



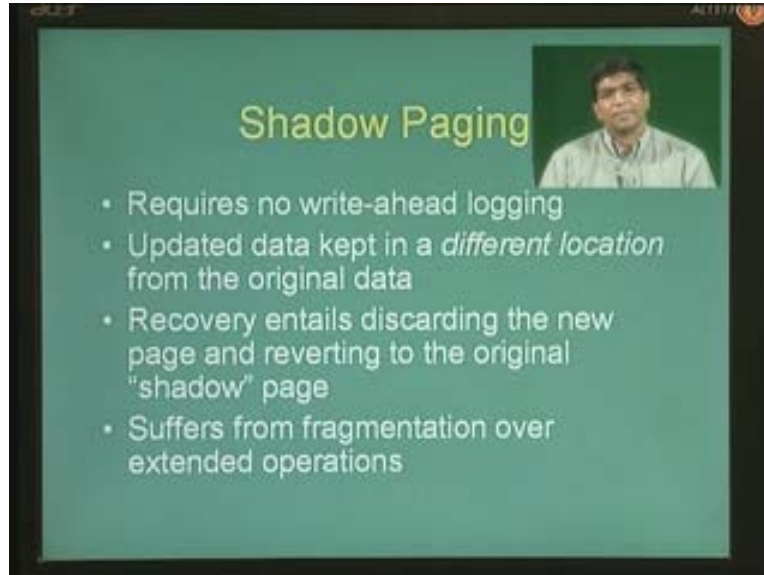
Recovery from such a non quietest check pointing is also quite simple. Let us say at some point in time there is a crash and we recover from such a crash or we have started with the recovery process after the crash and we start to read the log backwards that is from the last entry onwards. Now suppose we meet a end check point entry first, what does that means that the previous check pointing operation has successfully completed. That is we know all incomplete transactions that were there at the time of the check point have been completed. Therefore there is no need to do anything that is it is safe, we just have to restart whatever transactions were running after this time.

But suppose we reach a start check point without an end check point. What does this mean? This means that the crash has occurred during the check pointing operation itself. That is these were the set of transactions that were running and all of them have not yet terminated, they have not yet committed or aborted and the crash has happened. So therefore what is that we need to do? We have to undo all these operations that is all the transactions, all the operations that have been performed by these transactions.

There again we can use some kind of backward like we saw in the undo only kind of recovery algorithm by which we undo all of these operations. And until how far behind in the log we need to go? We need to go just until the previous check point because we know that the previous check point would not have completed until all the active transactions at that point in time have not yet completed.

So therefore we never need to go beyond one check point that is beyond the previous checkpoint. That is we can safely throw everything else beyond the previous check point and we just have to trace them through the log till the previous check point and undo all these operations. So that's another kind of check pointing operations that are used in addition to or in complement to the normal quietest based check pointing where you need to suspend all activities of the database system.

(Refer Slide Time: 35:41)

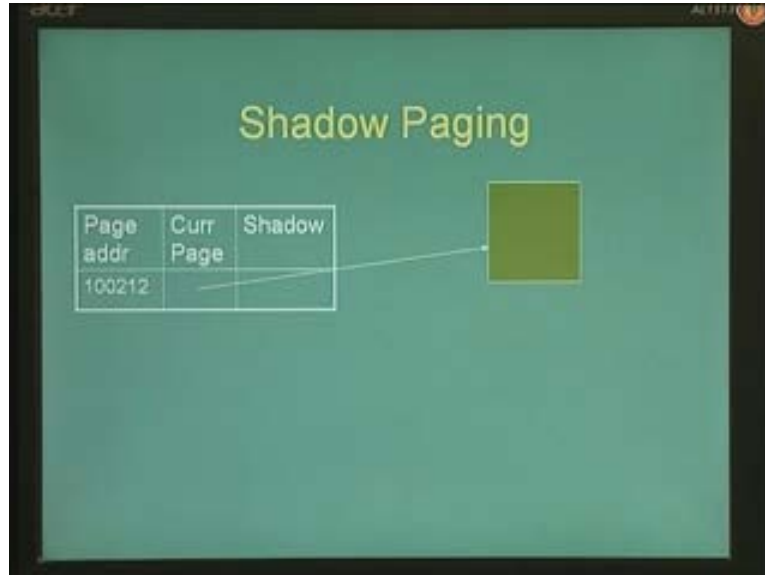


There is another... (Refer Slide Time: 35:42) which is called as shadow paging, it was ... (Refer Slide Time: 35:50) Part of sys ... (Refer Slide Time: 35:53) in this but where the technique of recovery is not based on logging but on maintaining different copies of database pages. That is the database is managed in terms of different pages or logical entities of disk of operation which pages could be just let's say disk blocks or disk sectors or set of sectors and so on.

And we basically make different copies of these pages in order to maintain the log. So there is no need for write ahead logging and all updated data are kept in a different location from the original data, they are not kept on the main or on the same data set. And recovery basically entails that we discard the new page whatever is the updated page and revert back to the shadow original page. That is the shadow of the original page which was still there that is we don't delete the original page until it is safe to do so which we will see shortly.

Shadow paging is quite efficient in terms, in the sense that there is no need to perform multiple write operations. In log based recovery you need to first do a write ahead logging and then write on to the database system. So there is a need to perform multiple write operations for every given multiple update operations for every update operation that is involved. But in shadow paging there is no need to do that however shadow paging suffers from fragmentation of pages over an extended period of time and usually we need to defragment the set of database pages by an offline operation after a number of recovery techniques have been executed.

(Refer Slide Time: 37:54)

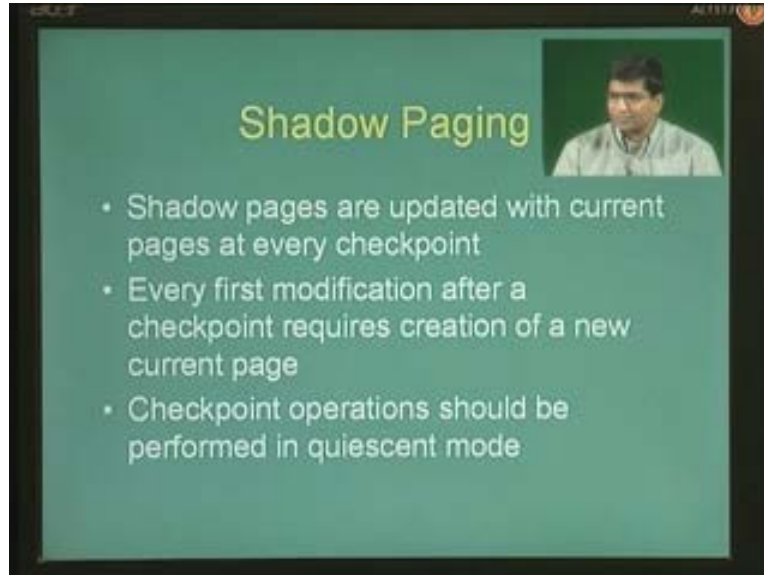


So how does the shadow paging technique work? Here is a small illustration. We usually keep a page table that is each page has a particular address where each page can correspond to a set of tuples for a given table for example and each page is given a particular address. And there two different point of for each page address what is called as the current page pointer and the shadow page pointer.

Initially we just have the current page pointer here and which is pointing to a particular page. Now let us say some tuple in this page has to be modified. So there is a modification request for this page. now we see that this page has no shadow page. So therefore what we do is we create a copy of this page and point the current page pointer to the copy. The original will now become the shadow that is the shadow of the original page that existed.

And we are perform the modifications, whatever update operations that we need to do will be done on this page here. Now suppose somewhere down the line when we are using this page there is a system crash. Now when is a system crash all that we need to do is to discard this page and go back to the shadow original page. And then change the current page pointer to the shadow pointer here and set the shadow pointer to null. So we just discard whatever was there and we have gone back to a consistent state. Now you might be wondering at what time are we going to start, how do we know that this is consistent that is how do we know at what time do we start creating a shadow page.

(Refer Slide Time: 39:41)



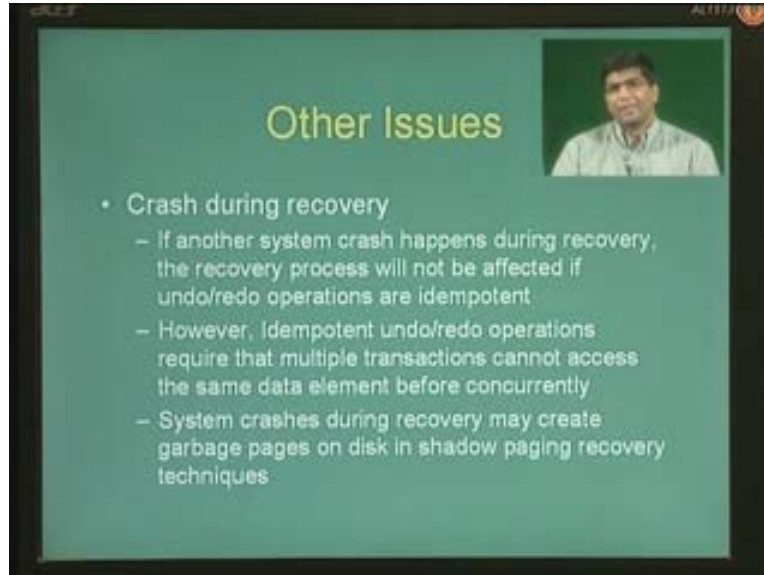
The slide is titled "Shadow Paging" in a yellow font. It features a small video inset in the top right corner showing a man in a light-colored shirt speaking. The main content consists of three bullet points in white text on a dark green background:

- Shadow pages are updated with current pages at every checkpoint
- Every first modification after a checkpoint requires creation of a new current page
- Checkpoint operations should be performed in quiescent mode

Essentially shadow pages are updated with current pages at every check point. That is whenever there is any check point or check pointing operation not just crash recovery operation, we perform this following operation, we perform the following set of activities. That is we replace the current page with a shadow page, make the shadow page pointer to null and the current page pointer to the previous shadow page pointer. So at every check point we are flushing the shadow page or we are updating the shadow page with the new current page.

And every first modification after a check point requires creation of a new current page. And as this kind of evident here, the check pointing operation that has to be used here should be quiescent mode check pointing operation that is we need to suspend a database activities for performing such a update operation that is replacing the shadow page with the new consistent current page.

(Refer Slide Time: 40:46)



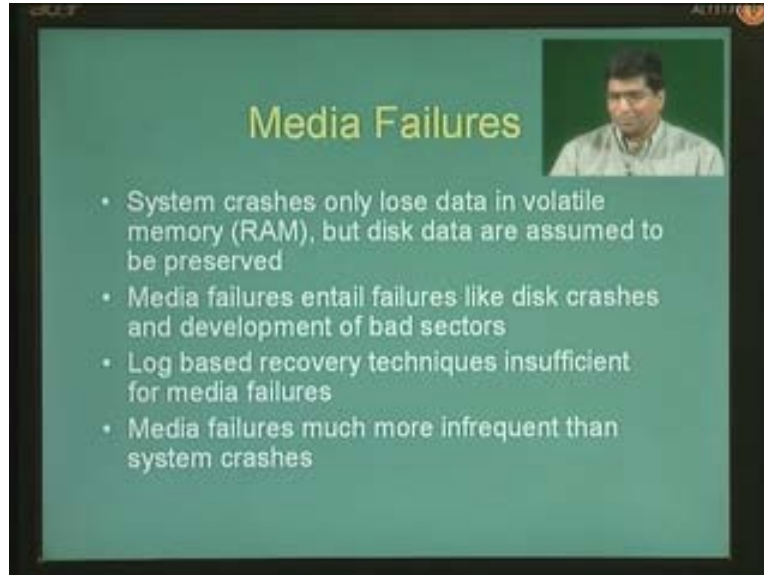
There are also other issues during that we have to content with for crash recovery. What happens that is the first, the main issue is what happens if there is crash during the recovery process itself. That is there is a system crash and we are trying to recover the database system and then there is again a crash.

Now in order to be able to handle such issues that is so that crashes during recovery do not affect the database, we need to be or we need to ensure that the undo and redo operations are idempotent. Remember even in the previous session we had used this word idempotent. Idempotent essentially means that no matter how many times you perform the operation, the net effect would be as though you have performed the operation only once. And it doesn't matter how many times you perform this operation that is let us say copying an element a to element b is idempotent. No matter, how many times you copy it, it is equivalent to saying that we have copied it only once

But there is a problem with idempotent operations as well. That is if you have to make undo and redo operations idempotent then we have to ensure that multiple transactions or multiple active transactions do not access the same element at the same time. That is one transaction has to wait until previous transaction has committed before it is safe for it to access a data element and that actually impedes performance in terms of performance.

And ofcourse in shadow paging technique there is other problem which occurs when there is a crash during recovery, especially when we are manipulating pointers. That is suppose you have discarded the new current page and we are changing the pointers to the to the old shadow page and then there is a system crash. It may so happen that all pointers are lost and the page has just become garbage that is there is no way to access the page and basically place it in the context of the larger database.

(Refer Slide Time: 43:14)



Media Failures

- System crashes only lose data in volatile memory (RAM), but disk data are assumed to be preserved
- Media failures entail failures like disk crashes and development of bad sectors
- Log based recovery techniques insufficient for media failures
- Media failures much more infrequent than system crashes

Until now we have essentially talked about system crashes or transaction failures where the fundamental criteria was that we can lose or we may lose data in the main memory but not in the persistent storage. What happens if there are media failures that is there is a disk crash, let us say or development of bad sectors. In this case we cannot even rely upon the data to be safe in the persistent storage as well that is even the data are there in the persistent storage are gone. Fortunately media failures are much more ... (Refer Slide Time: 43:53) disk crashes happen far more in frequent fashion than let us say power offer or operating system hanging and so on.

(Refer Slide Time: 44:06)



Protecting against media failures

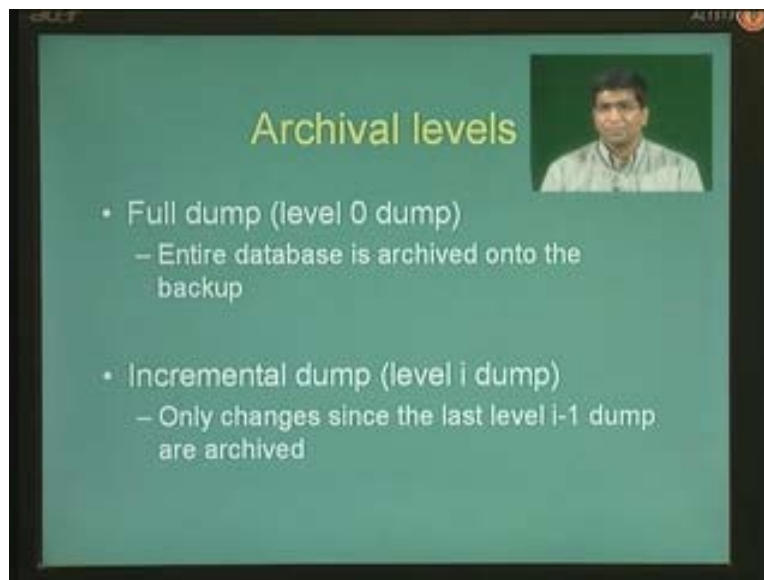
- Archiving
 - Maintaining a copy of the database separate from the database itself
 - Archiving or "backup" operations need to be performed online. It may be too costly to suspend operations of the database for backup
 - Relatively infrequent activity compared to logging

And ... (Refer Slide Time: 44:08) crashes what are called as archiving or taking backups and I am sure you know the concept of taking backups. It's simply taking a copy of a entire set of database on to another offline media which is stored physically in a different location. And however when you are talking about large database systems, again when we talk about database system always imagine a large database system because most of the problems of database management systems come from their size and not from anything else.

So imagine a large database system like a railway reservation or a bank or whatever. Now the problem is ... (Refer Slide Time: 44:55) itself takes a huge amount of time. Suppose I have one giga byte of data in my database which is quite common, in fact it could sometimes we even have more than one giga byte of data, we have several giga bytes of data in any operational database and possibly even tera bytes of data. Now copying them to a separate let us say tape media or optical storage takes huge amount of time in the order of hours or possibly sometimes even days to copy the entire set of data elements onto backup storage and therefore there is a need to perform back up operations in an online fashion.

That is as and when database activities are going on, we cannot obviously suspend all railway reservations let us say for one day. We cannot say tomorrow there is you cannot reserve any train tickets and so on. It has to be on everyday 24 hours a day or 7 days a week, so everyday or every instance of time the reservation activities are going on. You can book train tickets over the net and you can book them over the counter so on and so forth.

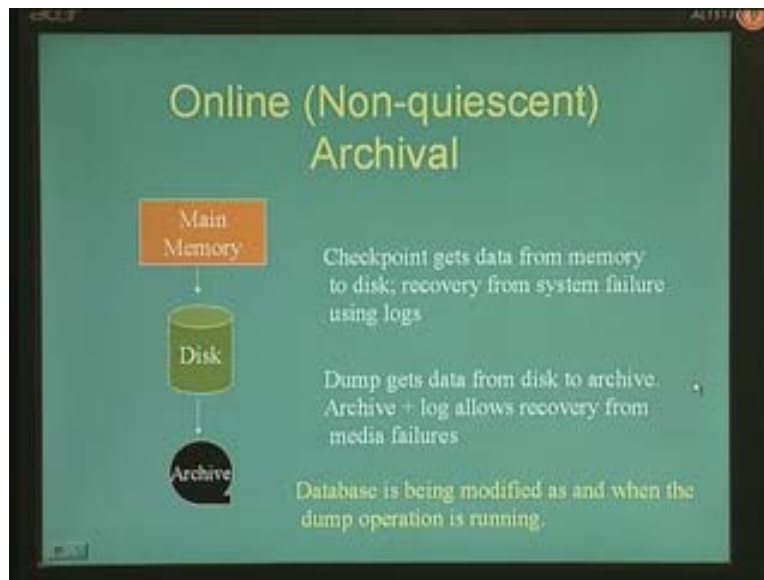
(Refer Slide Time: 46:14)



So there is a need to perform online backup operations. So let us briefly see what is the basic idea behind online operations and which can help us appreciate the need and the complexities involved in a online backup operations. Before we do that we need to take

up certain definitions, we distinguish between different levels of archivals. So what to call as level zero archivals and level i archival. Level zero archival or what is called as a full dump is an entire archival or a copy of the entire database onto the archival storage. And a level ... (Refer Slide Time: 47:04) incremental dump is essentially copying of only the changes that the database has under gone before the, rather after the last archival. That is only changes since the last i minus 1 level i minus 1 are archived.

(Refer Slide Time: 47:25)

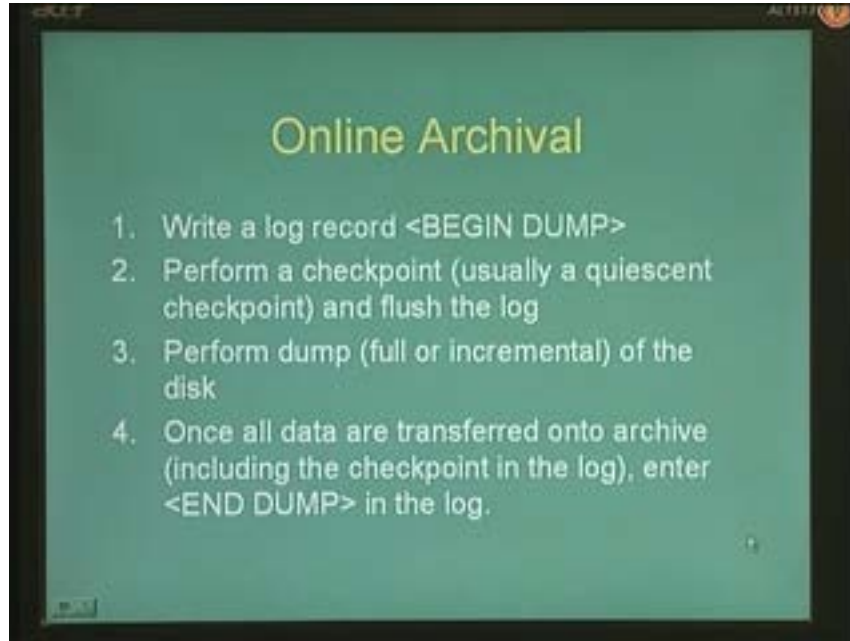


Now the basic idea behind online archival is shown in this figure. Between main memory and disk there is this check pointing operation. As you might have noticed similarity between archival and check pointing. Check pointing requires that the dbms system to be brought in to a quiescent state and archival requires a suspension of all database activities in order to copy them onto the archive.

So check pointing gets data from memory onto disk and whenever we need to recover, we recover based on logs. And the archival process gets data from disk onto archive and because it is unrealistic to say that we suspend all dbms activities during archival. We usually allow that the dbms to keep updating its database operations as and when the archival process is going on.

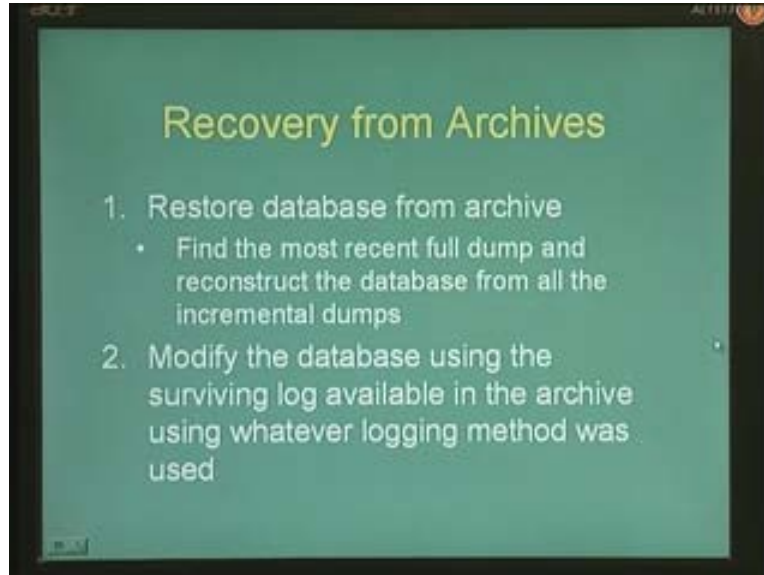
However in addition to the database, we also store the log from the previous check point until the previous check point in addition to the database onto the archive. Therefore whenever they need to recover, we need to recover from archive plus log to bring back the database onto the last consistent state before a media crash and the database is being modified as and when the dump operation is running.

(Refer Slide Time: 48:47)



So a simple algorithm for online archival is given here. We first begin by writing a begin dump record to the log that is starting from the logging operation itself we begin the archival process. We then perform a check point usually a quiescent check point but not necessarily and then we flush the log. That is we have begin the dump and began the check point that is we have flushed all committed transactions onto the database and we have a log of all the active transactions here. Then perform the dump, full or incremental or whatever kind of dump from the disk on to the archival data. Now once all data are transferred onto archive including the check point that is there in the log then enter end dump in the log.

(Refer Slide Time: 49:47)



So what is the advantage? The advantage is shown in this slide that is during recovery that is then we have to restore the database. Now the recovery from media crashes is called restore rather than recovery that is we are restoring the database from the archive. So when we restore the database from the archive, we find the most recent full dump and reconstruct the archive based on all the incremental dumps and then we write it back on to the database.

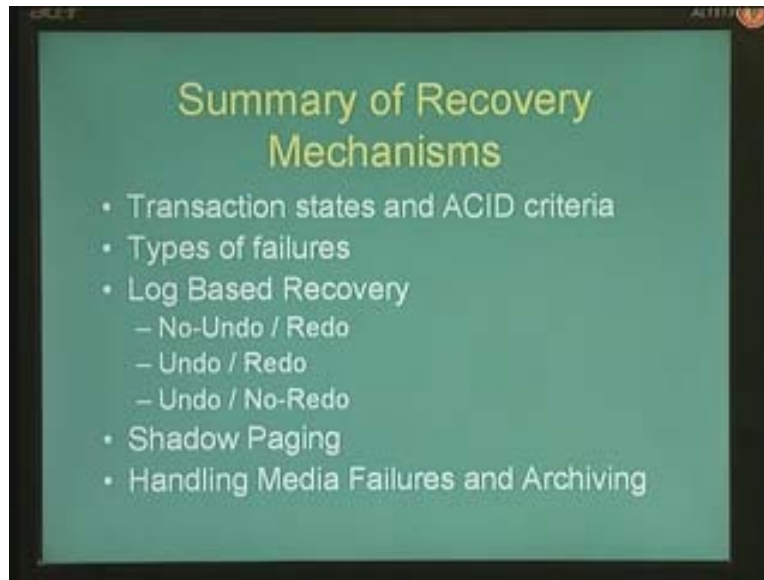
And then we take the database log that is the surviving log entry that was also archived in the archives, in the archival storage and there we have a check point which shows what all has been done to the database and whatever was there after the check point we start redoing those database. That is whatever kind of logging method is used that is suppose we have an undo logs then we perform the corresponding undo operations and suppose there were it was a redo log we perform the corresponding redo operations depending upon lets say it's a deferred updates or an immediate kind of logging operations.

So that kinds of summarize ... (Refer Slide Time: 51:05) there are number of other topics in database recovery which we have not touched upon, especially perhaps most significantly the kind of recovery techniques that are used in many of the commercial dbms systems. Most of the commercial dbms systems today use a combination of undo and redo logging operation that is this is to get the right tradeoff between performance and recovery over rates because in a pure redo operations you have a performance overhead in the sense that there is a huge spurt of activity during commits for every transaction.

Therefore they use a combination of these undo and redo operations and there is a kind of a famous series of protocols called aries, a r I e s which can possibly search on the internet which was proposed it IBM unverdant research centre which was kind of used

quite, which is kind of very popular and used in several different commercial database systems.

(Refer Slide Time: 52:23)



We have not covered the details of the aries protocols here for the reasons of brevity but most of that concepts used in aries depend upon or based upon several of the concepts that we have studied here like deferred updates or immediate updates and check pointing and undo and redo operations and idempotent operations and so on and so forth.

So let us briefly summarize what we studied in database recovery under the topic of database recovery. We looked into the idea of transaction states and what are the different states that a particular transaction can exist and the acid criteria for transactions which determines how or which determines how our recovery protocol should run. And we looked at different kinds of failures like say system crashes media crashes transaction failures and so on.

And for the most frequent kind of failures like say system crashes or transaction failures we saw three different kinds of recovery mechanisms. That is no undo slash redo operations that is deferred update recovery and undo slash redo which is immediate update and there is undo slash no redo operations which are based on compensating log records or clr's.

We also saw the notion of shadow paging where you don't have the need for a log therefore it's much more faster, the recovery mechanisms or the dbms itself is much more faster because there are no multiple write operation overheads that are involved during every update but which suffers from a possible fragmentation. That is so many pages created at different places in the disk are possibly when suppose there is system crash during recovery itself, there is a possibility of encountering garbage system pages as part

of the disk itself. So shadow paging has its own advantages and disadvantages with respect to write ahead logging.

We also saw the notion of archiving or the idea of taking online backups for handling media failures where we take a incremental backup and we recover or we restore from backup based on not just the archives but based also on the log that ... (Refer Slide Time: 54:57). That brings us to the end of the session.