DATABASE MANAGEMENT SYSTEM

Subash Manandhar

Chapter 9 : Crash Recovery

- **Recovery System:**
  - DBMS is highly complex system with hundreds of transactions being executed every second.
  - Availability of DBMS depends on its complex architecture and underlying h/w or system s/w.
  - If it fails or crashes amid transactions being executed, it is expected that the system would follow some sort of algorithms or techniques to recover from crashes or failures.
  - There may be several causes of failure including power failure, disk crashed, s/w bugs etc.
  - Different types of failures are
    - Transaction failure
    - System crash
    - Disk failure

Chapter 9 : Crash Recovery

- **Transaction failure**
  - **Logical errors:**
  - Transaction can’t complete due to some internal condition.
  - **System errors:**
  - The database system must terminate an active transaction due to an error condition (e.g. deadlock)
- **System crash**
  - A power failure or other h/w or s/w failure causes the system to crash.
  - **Fail stop assumption:**
  - Non volatile storage contents are assumed not to be corrupted by system crash.
  - Db systems have numerous integrity checks to prevent corruption of disk data.
- **Disk failure**
  - a head crash or similar disk failure destroys all or part of disk storage.
  - Destruction is assumed to be detectable; disk drives use checksums to detect failures.
  - Multiple copies or archival tapes are solutions.

Chapter 9 : Crash Recovery

- Recovery algorithms are techniques to ensure database consistency and transaction atomicity and durability despite failures.
- Recovery algorithms have two parts:
  - Actions taken during normal transaction processing to ensure enough information exists to recover from failure.
  - Actions taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability.
- Modifying the database without ensuring that the transaction will commit may leave the database in an inconsistent state.
- To ensure atomicity despite failures, we first o/p information describing the modification to stable storage without modifying database itself.
- Two approaches:
  - Log Based Recovery
  - Shadow Paging
- Here, assume that transaction run serially.

Chapter 9 : Crash Recovery

- **Log Based Recovery:**
  - The log is a sequence of log records, and maintains a record of update activities on the database, which is kept in a stable storage.
  - It works as follows:
    - The log file is kept on stable storage media.
    - When a transaction T_i enters the system and starts execution, it writes a log about it $<T_i$, start$>$
    - When a transaction T_i modifies an item x, it write log as follows $<T_i$, x, v1, v2$>$; this means T_i has changed the value of x from v1 to v2.
    - When the transaction T_i finishes, it logs as $<T_i$, commit$>$
  - Two approaches using logs
    - Deferred database modification
    - Immediate database modification

Chapter 9 : Crash Recovery

- **Deferred Database Modification:**
  - It ensures transaction atomicity by recording all database modification in the log but deferring the execution of all write operations of a transaction until the transaction partially commits.
  - When a transaction partially commits the information on the log associated with the transaction is used in executing the deferred writes.
  - If the system crashes before the transaction completes its execution, then the information on the log is simply ignored.
  - Transaction starts by writing $<T_i$, start> record to log.
  - A write(X) operation results in a log record $<T_i$, X, V> being written where V is the new value of X. here old value is not needed.

Chapter 9 : Crash Recovery

- **Deferred Database Modification:**
  - **Deferred Update Steps:**
    - ‘write’ operation is not performed on X at this time but is deferred.
    - When T_i partially commits <T_i , commit> is written to log.
    - Finally, the log records are read and used to actually execute the previously deferred writes.
  - **Redo only scheme:**
    - During recovery after crash, a transaction needs to be redone if and only if both <T_i, start> and <T_i , commit> are there in log.
    - Redoing a transaction T_i (redo(T_i)) sets the value of all data items updated by the transaction to the new values.
  - Crashes can occur while the transaction is executing the original updates or while recovery action is being taken.

Chapter 9 : Crash Recovery

- **Deferred Database Modification:**
  - E.g. Transaction T0 and T1 such that T0 executes before T1.

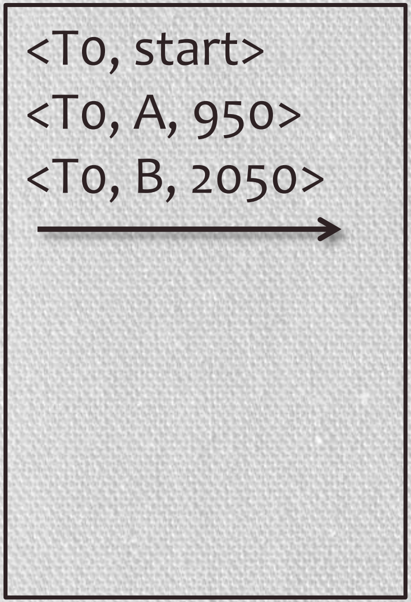
```
T0 : read(A)
     A = A - 50
     write(A)
     read(B)
     B=B+50
     write(B)
```

```
T1 : read(C)
     C = C - 100
     write(C)
```

LOG	DB
<T0, start>	
<T0, A, 950>	
<T0, B, 2050>	
<T0, commit>	
	A = 950
	B = 2050
<T1, start>	
<T1, C, 900>	
<T1, commit>	
	C = 900

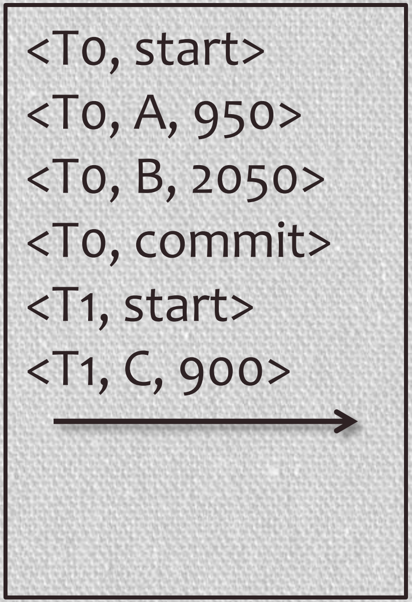
Chapter 9 : Crash Recovery

- **Deferred Database Modification:**
  - Recovery procedure redo(T_i) sets the value of all data items updated by transaction T_i to new values.
  - Below is the log as it appears at three instances of time.



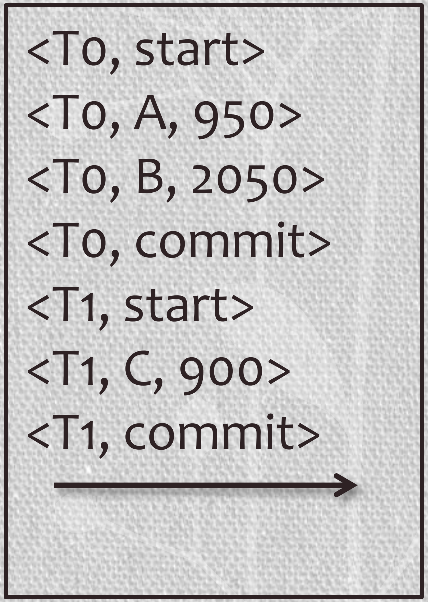
```
<T0, start>
<T0, A, 950>
<T0, B, 2050>
```

Fig: a



```
<T0, start>
<T0, A, 950>
<T0, B, 2050>
<T0, commit>
<T1, start>
<T1, C, 900>
```

Fig: b



```
<T0, start>
<T0, A, 950>
<T0, B, 2050>
<T0, commit>
<T1, start>
<T1, C, 900>
<T1, commit>
```

Fig: c

Chapter 9 : Crash Recovery

- **Deferred Database Modification:**
  - If log on stable storage at time of crash is as in fig. a, No redo action need to be taken
  - If log on stable storage at time of crash is as in fig. b, redo (T0) must be performed since <T0, commit> is present.
  - If log on stable storage at time of crash is as in fig. c, {redo(T0), redo(T1)} both must be performed since <T0, commit> and <T1, commit> are present.

Chapter 9 : Crash Recovery

- **Immediate Database Modification:**
  - Allows database updates of an uncommitted transaction to be made as the writes are issued.
  - Since, undoing may be needed, update logs must have both old value and new value.
  - Update log record must be written before database item is written.
    - We assume that the log record is output directly to stable storage.
    - Before execution of an output(B) operation for data block B, all log records corresponding to items B must be flushed to stable storage.
  - Output of updated blocks can take place at any time before or after transaction commit.
  - Order in which blocks are output can be different from the order in which they are written in buffer.

Chapter 9 : Crash Recovery

- **Immediate Database Modification:**
  - E.g. Transaction T0 and T1 such that T0 executes before T1.

```
T0 : read(A)
     A = A - 50
     write(A)
     read(B)
     B=B+50
     write(B)
```

```
T1 :  read(C)
      C = C - 100
      write(C)
```

LOG	**DB**
<T0, start>	
<T0, A, 1000, 950>	
<T0, B, 2000, 2050>	
	A = 950
	B = 2050
<T0, commit>	
<T1, start>	
<T1, C, 1000, 900>	
	C = 900
<T1, commit>	

Chapter 9 : Crash Recovery

- **Immediate Database Modification:**
  - Here, recovery procedure has two operations instead of one.
    - **Undo(T_i) :** restores the value of all data items updated by T_i to their old values going backwards from last log record for T_i.
    - **Redo(T_i) :** sets the value of all data items updated by T_i to the new values, going forward from the first log record for T_i .
  - Both operations must be idempotent
    - Undo(T_i) = Undo(Undo(T_i))=……………….
    - That is even if the operation is executed multiple times the effect is the same as if it is executed once. Needed since operations may get re-executed during recovery.
  - When recovering after failure transaction T_i needs to be redone if the log contains both the record <T_i, start> and record <T_i , commit>
  - Undo operations are performed first then only Redo operation.

Immediate Database Modification:

- Below is the log as it appears at three instances of time.

```
<T0, start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
  ⟶
```

Fig: a

```
<T0, start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0, commit>
<T1, start>
<T1, C, 1000, 900>
  ⟶
```

Fig: b

```
<T0, start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0, commit>
<T1, start>
<T1, C, 1000, 900>
<T1, commit>
  ⟶
```

Fig: c

Chapter 9 : Crash Recovery

- **Immediate Database Modification:**
  - If log on stable storage at time of crash is as in fig. a, Undo(T0) must be performed.
  - If log on stable storage at time of crash is as in fig. b, Undo(T1) and then Redo(T0) must be performed.
  - If log on stable storage at time of crash is as in fig. c, Redo(T0) and Redo(T1) must be performed.

Chapter 9 : Crash Recovery

- **Checkpoints :**
  - When more than one transactions are being executed in parallel, the logs are interleaved, at the time of recovery it would become hard for recovery system to backtrack all logs, and then start recovering.
  - To ease this situations most modern DBMS use concept of checkpoints.
  - Problems in recovery procedure as discussed earlier
    - Searching the entire log is time consuming.
    - We might unnecessarily redo transaction which have already output their updates to the database.
  - Streamline recovery procedure by periodically performing checkpointing
    - Output all log records currently residing in main memory on to stable storage.
    - Output all modified buffer blocks to disk.
    - Write a log record <checkpoint> on to stable storage
  - During checkpointing, other transactions cannot be processed.

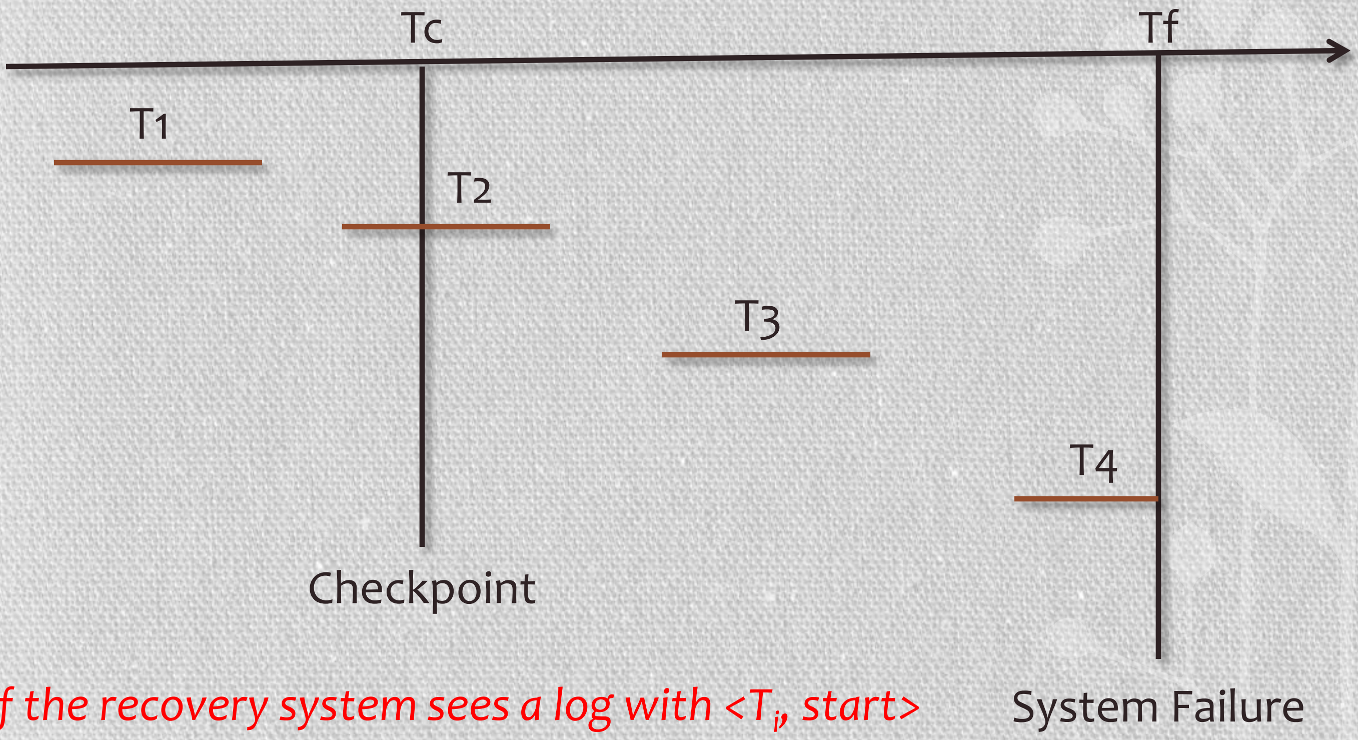
Chapter 9 : Crash Recovery

- **Checkpoints :**
  - During recovery we need to consider only the most recent transaction T_i that started before the checkpoint, and transactions that started after T_i
    - Scan backwards from end of log to find the most recent <checkpoint> record.
    - Continue scanning backwards till a record <T_i , start> is found.
    - Need only consider the part of log following above start record. Earlier part of log can be ignored during recovery and can be erased whenever desired.
    - For all transactions (starting from T_i or later) with no <T_i, commit>, execute Undo(T_i)
    - Scanning forward in the log, for all transactions starting from T_i or later with a <T_i, commit> , execute redo(T_i)

Chapter 9 : Crash Recovery

- **Checkpoints :**

If the recovery system sees a log with <T_i, start> but no commit or abort log found, it puts the transaction in undo list.



- *T1 can be ignored (updates already output to disk due to checkpoint)*
- *Undo T4*
- *Redo T2 and T3*

If the recovery system sees a log with <T_i, start> and <T_i, commit> or just <T_i, commit>, it puts the transaction in redo list.

Chapter 9 : Crash Recovery

- **Shadow Paging :**
  - Is an alternative to log based recovery.
  - Idea: maintain two page tables during the life time of a transaction; the current page table and the shadow page table.
  - Store the shadow page table in non volatile storage, such that state of the database prior to transaction execution may be recovered.
    - Shadow page table is never modified during execution.
  - To start with, both the page tables are identical
    - Only current page table is used for data item accesses during execution of the transaction.
  - Whenever any page is about to be written for first time
    - A copy of this page is made onto an unused page
    - The current page table is than made to point to the copy.
    - The update is performed on the copy.

Chapter 9 : Crash Recovery

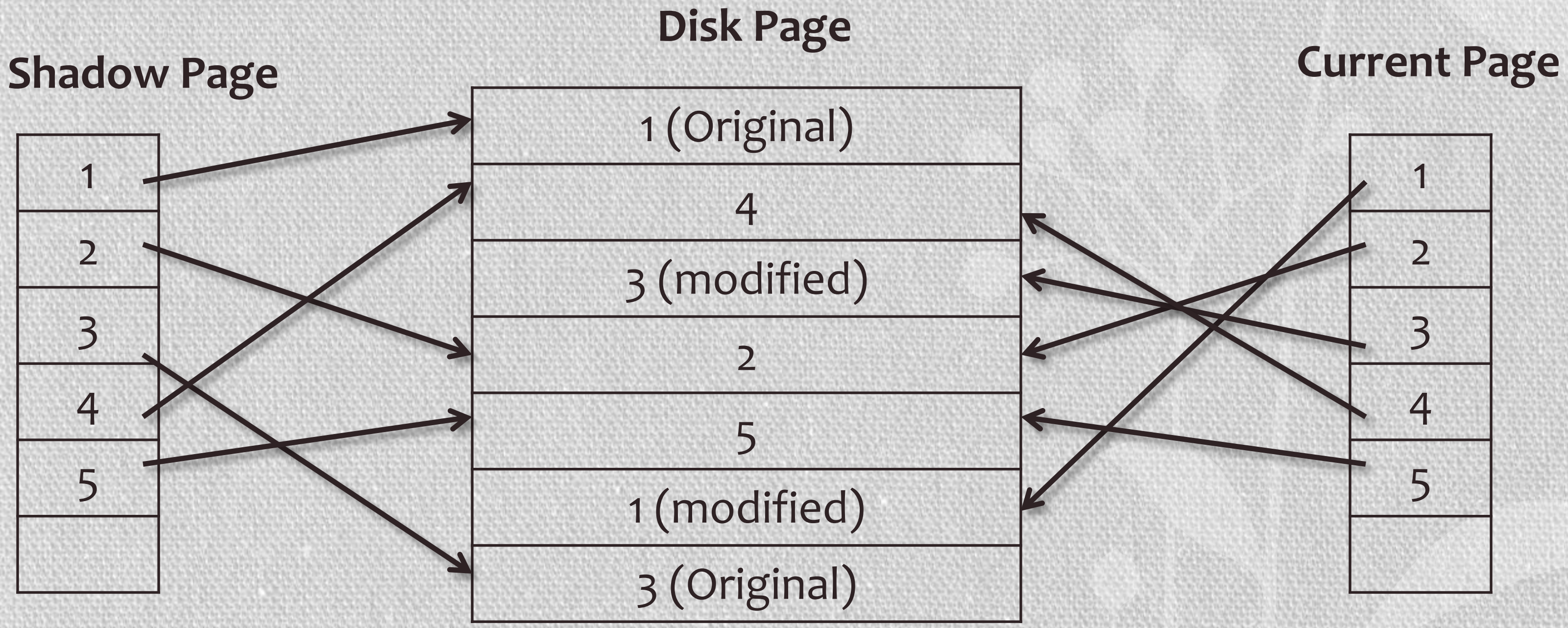
- **Shadow Paging :**
  - To commit a transaction
    - Flush all modified pages in main memory to disk.
    - Output current page table to disk
    - Make the current page table the new shadow page table as follows:
      - Keep a pointer to shadow page table at a fixed known location on disk.
      - Simply update the pointer to point to current page table on disk.
  - Once pointer to shadow page table has been written, transaction is committed.
  - Non recovery is needed after a crash, new transactions can start right away using shadow page table.
  - Pages not pointed to from current/shadow page table should be free
  - Advantages of shadow paging over log based
    - No overhead of writing log records
    - Recovery is trivial

Chapter 9 : Crash Recovery

- **Shadow Paging :**
  - Disadvantages of shadow paging
  - Copying the entire page table is very expensive
    - Can be reduced by using a page table structured like B+ tree. here, no need to copy entire tree, only need to copy paths in the tree that lead to update leaf nodes.
  - Commit overhead is high even with above extension. Need to flush every updated page and page table.
  - Data gets fragmented (related pages get separated on disk)
  - After every transaction completion old version of modified data need to be garbage collected.
  - Hard to extend algorithm To allow transactions to run concurrently.

Chapter 9 : Crash Recovery

- **Shadow Paging :**



Chapter 9 : Crash Recovery

- Is to protect data.
- Backup files can protect against accidental loss of data, database corruption and other failures.
- Backup tapes are stored in secured location.
- With backup and recovery plan, we can easily recover our important data from any type of disasters.
- To create backup plan, deal with following
  - How important is data on your system?
  - What type of information does the data contain?
  - How often does the data change?
  - How quickly do you need to recover the data?
  - Do you have equipment to perform backup?
  - Who will be responsible for backup and recovery plan?
  - What is best time to schedule backup?
  - Do you need to store backup off site?

Chapter 9 : Crash Recovery

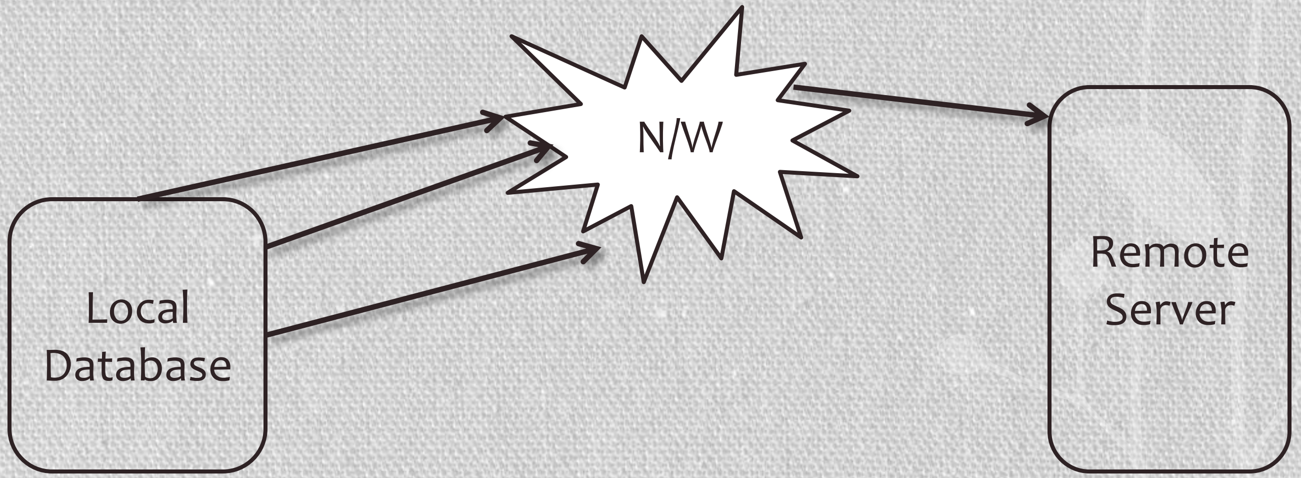
- Backup plan depends on factors like
  - Capacity
  - Reliability
  - Extensibility
  - Speed
  - Cost
- Common backup solutions
  - Tape drive
  - Digital audio tape (DAT) drives
  - Disk drives
  - Magnetic optical drive
  - Auto loader tape systems
  - Removable disks

Chapter 9 : Crash Recovery

- **Remote Backup / Online Backup**
- Here files, folders or entire contents of hard drive are regularly backed up on a remote server with n/w connection.
- Here risk of catastrophic data loss as a result of fire, theft, file corruption or other disaster is eliminated.
- Encryption and password protection system helps to ensure privacy and security.
- For large organization and for practically valuable data, online backup strategy is wise investment.
- In database, it is backup performed on data even though it is actively accessible to users and may currently be in state of being updated.
- Cloud computing, remote login method are for remote backup and recovery.

Chapter 9 : Crash Recovery

- **Remote Backup / Online Backup**
- Catastrophic failure is one where stable , secondary storage device get corrupted.
- To recover from catastrophic failure
  - Remote backup
  - Backup taken to magnetic tapes and stored in safer place.


Local Database
N/W
Remote Server