CMPT 506

Database Recovery Techniques

Read Chapters 23

Dr. Abdelkarim Erradi

Department of Computer Science and Engineering

QU

Recovery

< Back

Purpose of Database Recovery

- To bring the database into a consistent state after a failure occurs.
- To ensure the transaction properties of **Atomicity** (a transaction must be done in its entirety otherwise, it has to be rolled back) and **Durability** (a committed transaction cannot be canceled and all its updates must be applied permanently to the database).
- The **DBMS recovery manager** is responsible for bringing the system into a consistent state before transactions can resume.

Types of Failure

1. Transaction failure: Transactions may fail because of incorrect input, deadlock, etc.

2. System failure: System may fail because of addressing error, application error, operating system fault, RAM failure, etc.

- For 1 & 2 Data on disk still there on restart

  => Solution: atomicity via logging

3. Media failure: Disk head crash, etc.

- Data on disks lost! **=> Solution: Restore from backup**

4. Catastrophic failure: fire, earthquake, etc.

- Data on disks lost and local backup lost!

  => Solution: Restore data from geographically distributed backup

Recovery in DBMS

- **Example (to illustrate consistency issues that can be introduced by failures)**

Balance transfer

```
decrement the balance of account X
    by $100;
 increment the balance of account Y
    by $100;
```

- Scenario 1: Power failure after the first instruction
  - **Such failures may leave database in an inconsistent state with partial updates carried out**
  - Transfer of funds from one account to another should either complete or not happen at all

=> Database transactions come to the rescue!

Purpose of Database Recovery

- To bring the database into the last consistent state, which existed prior to the failure
  - To preserve transaction properties (Atomicity, Consistency, Isolation and Durability)
- Example:
  - If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value

=> Thus, the database must be restored to the state before the transaction modified any of the accounts

Log file

- Holds the information that is necessary for the recovery process
- Records all relevant operations in the order in which they occur
- Is an append-only file.
- Holds various types of log records (or log entries).

Log file entries

Types of records (entries) in log file:

- [start_transaction,T]: Records that transaction T has started execution.
- [write_item,T,X,old_value,new_value]: T has changed the value of item X from old_value to new_value.
- [commit,T]: T has completed successfully, and committed.
- [abort,T]: T has been aborted.

Log file entries (Cont.)

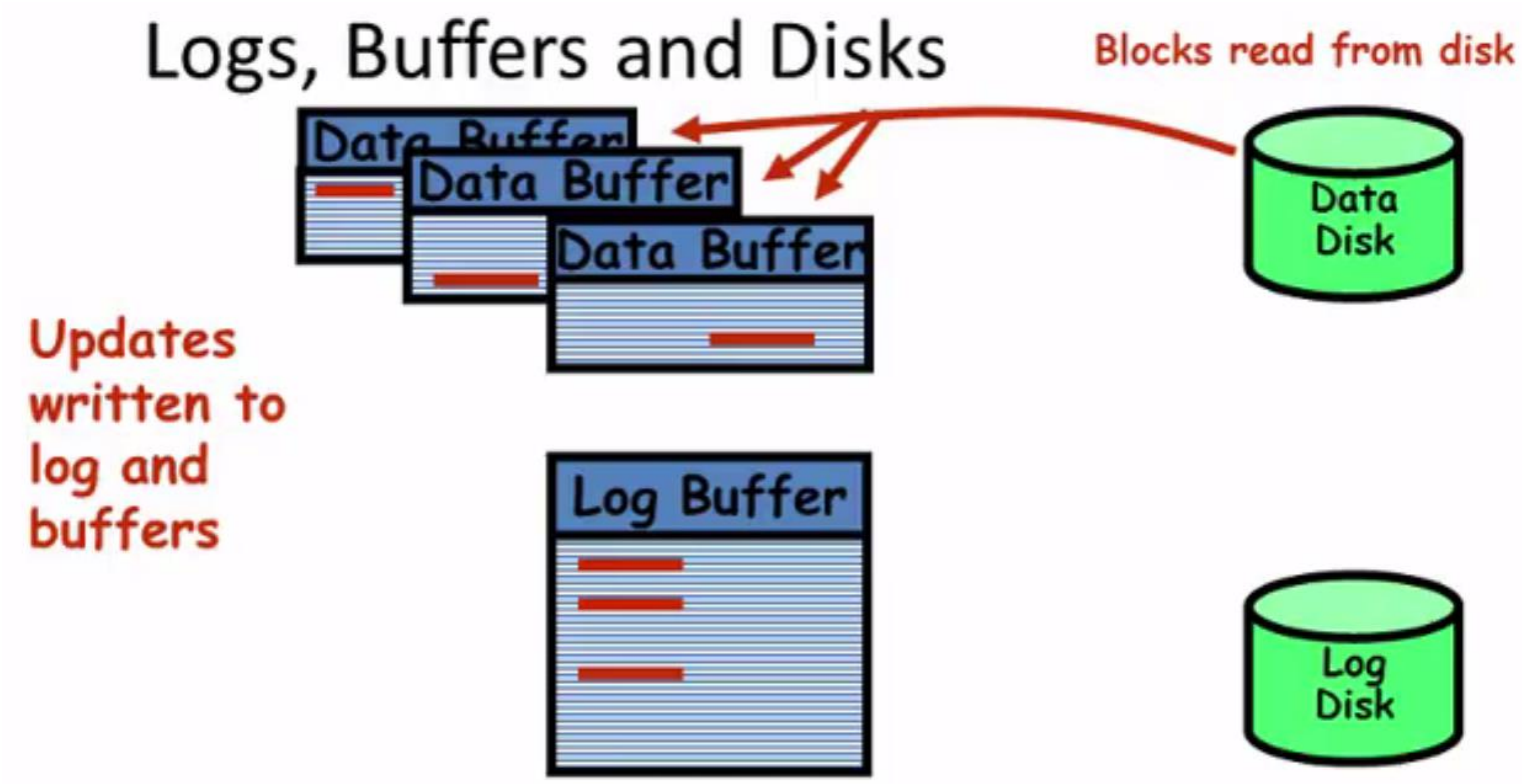
For **write_item** log entry, *old value* of item before modification (**BFIM** - Before Image) and the *new value* after modification (**AFIM** – After Image) are stored.

BFIM needed for UNDO, AFIM needed for REDO

A sample log

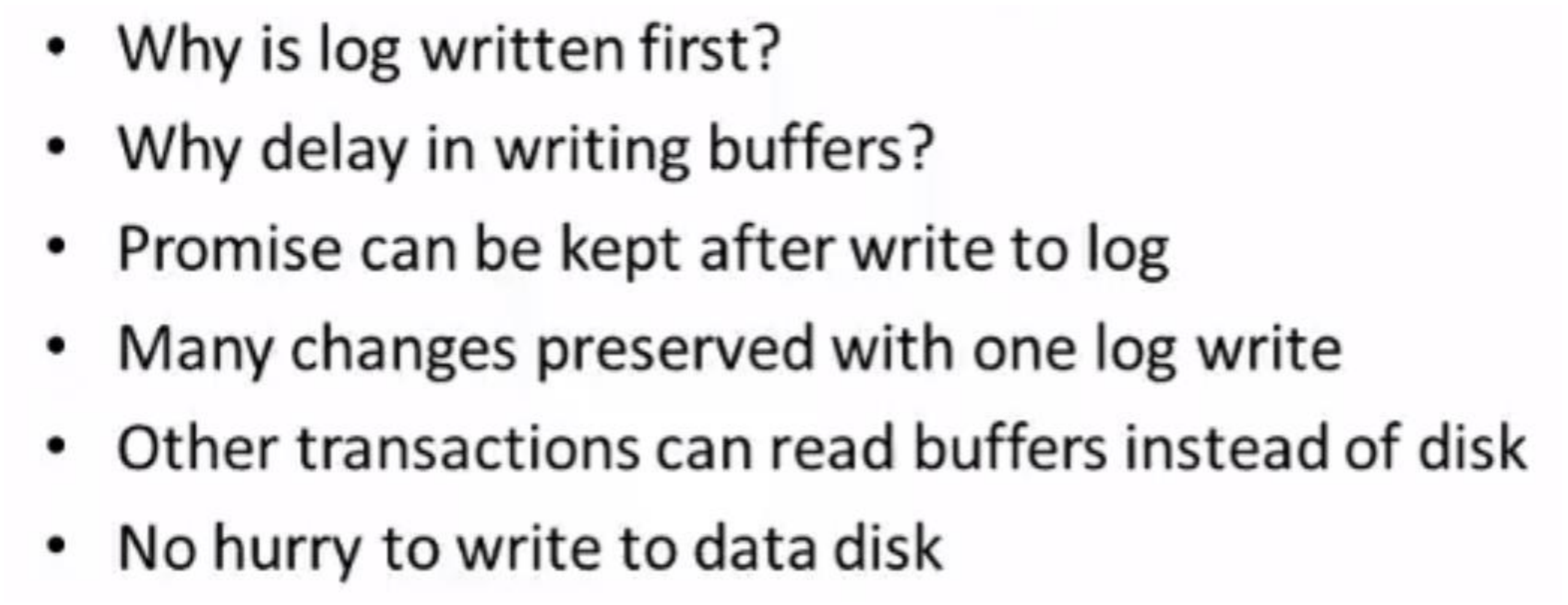
```
< START T1 >
<T1, A, 50, 25>
<T1, B, 250, 25>
<START T2>
<T1, A, 75, 50>
<T2, C, 35, 25>
<COMMIT T1>
<START T3>
<T3, E, 55, 25>
<T2, D, 45, 25>
```

Why do we need the log?



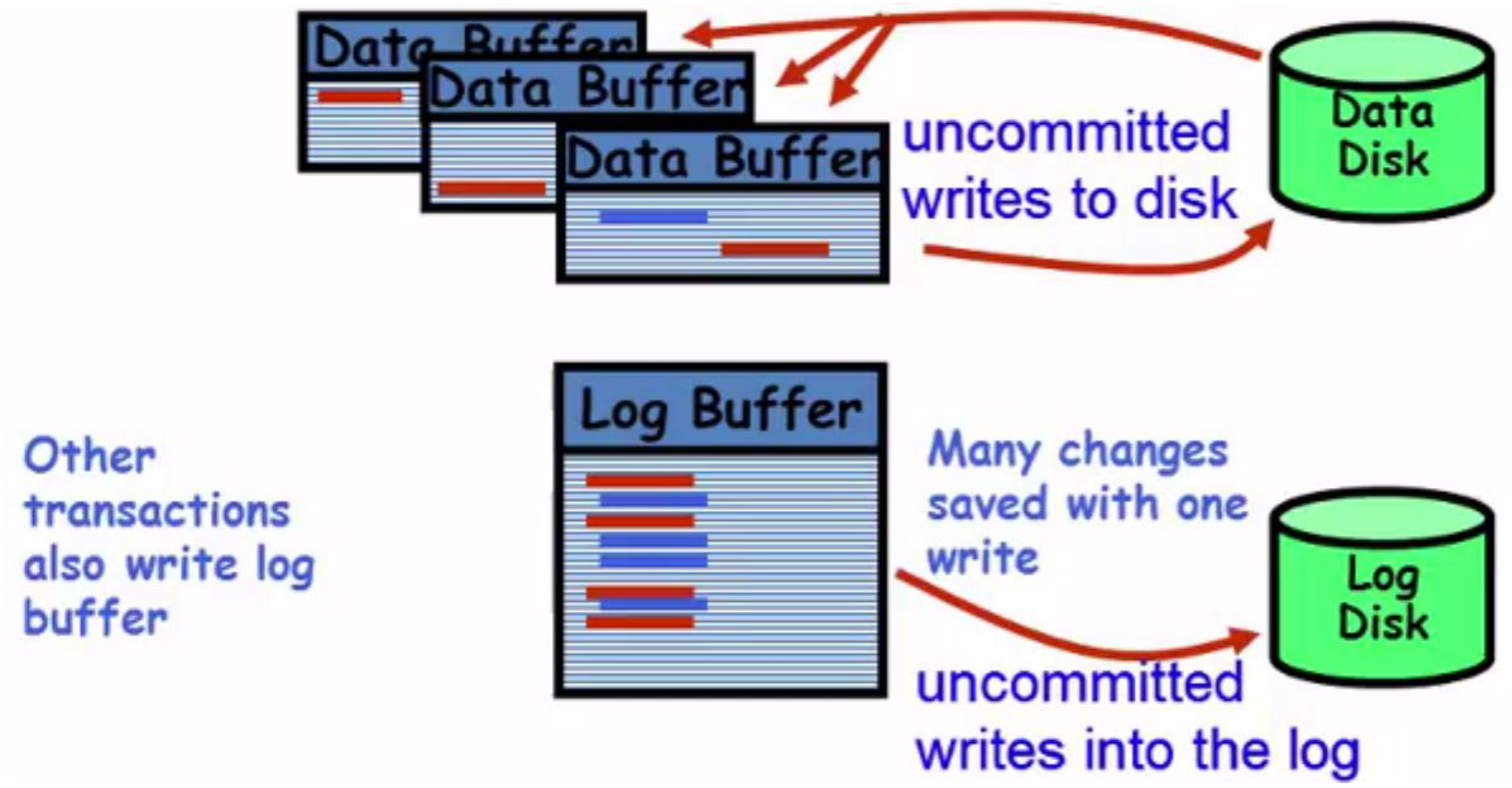
Data items to be modified are first read into Data Buffers by the Buffer Manager and after modification they are later flushed (written) back to the disk

Forcewriting Log



- Why is log written first?
- Why delay in writing buffers?
- Promise can be kept after write to log
- Many changes preserved with one log write
- Other transactions can read buffers instead of disk
- No hurry to write to data disk

Forcewriting Log


Data Buffer
Data Buffer
Data Buffer
uncommitted writes to disk
Data Disk
Other transactions also write log buffer
Log Buffer
Many changes saved with one write
Log Disk
uncommitted writes into the log

Undo/Redo Logging

Update X ➡ **<T_i, X, old X value, New X value>**

- Undo/Redo Logging Rules

(1) Element X can be flushed before or after Ti commit

(2) Before modifying X on disk, all corresponding log records appear on disk

(3) Flush log before commit

Recovery Process

- The undo/redo recovery policy
  - Redo committed transactions
  - Undo uncommitted transactions.
- **Backward pass** (end of log to the start)
  - Construct set C of transactions that committed
  - Undo all actions of transactions not in C
- **Forward pass** (start of log to the end)
  - Redo all actions of transactions in C

UNDO, REDO RECOVERY ACTIONS

- **Undo**: Restore all BFIMs from log to database on disk. UNDO proceeds backward in log
- **Redo**: Restore all AFIMs from log to database on disk. REDO proceeds forward in log
- UNDO (roll-back) is needed for transactions that are not committed yet
- REDO (roll-forward) is needed for committed transactions whose writes may have not yet been flushed from cache to disk

Checkpointing

To minimize the **REDO** operations during recovery checkpointing can be used. The following steps define a checkpoint operation:

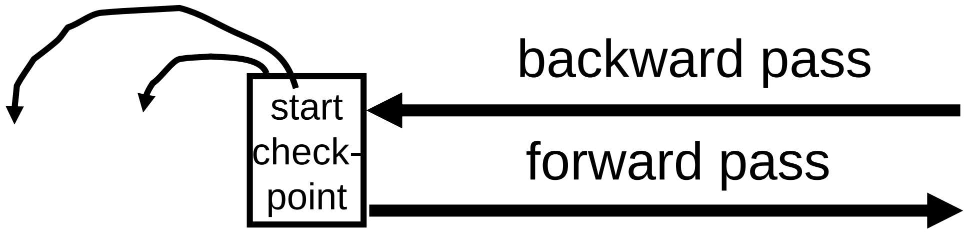
- Do not accept new transactions
- Write "start checkpoint" listing all active transactions to log
- Flush log to disk
- Write to disk all dirty **Data** Buffers whether or not their transactions have committed
- Write "end checkpoint" to log
- Flush log to disk
- Resume normal transaction execution

End Checkpoint

- If the system successfully performed the checkpointing (i.e., wrote **END CHECKPOINT** log entry to disk) then this means that all Dirty Data Buffers before the **START CHECKPOINT** must have been forced written to disk
  - If the crash happens <u>after</u> the **END CHECKPOINT** then we only redo actions of completed transactions <u>after</u> START CKPT (NOT from the start of the log file)
- This is basically the benefit of checkpointing i.e., avoid the need to redo ALL successful actions from the beginning of the log in order to reduce the recovery time

Recovery process (1 of 2)

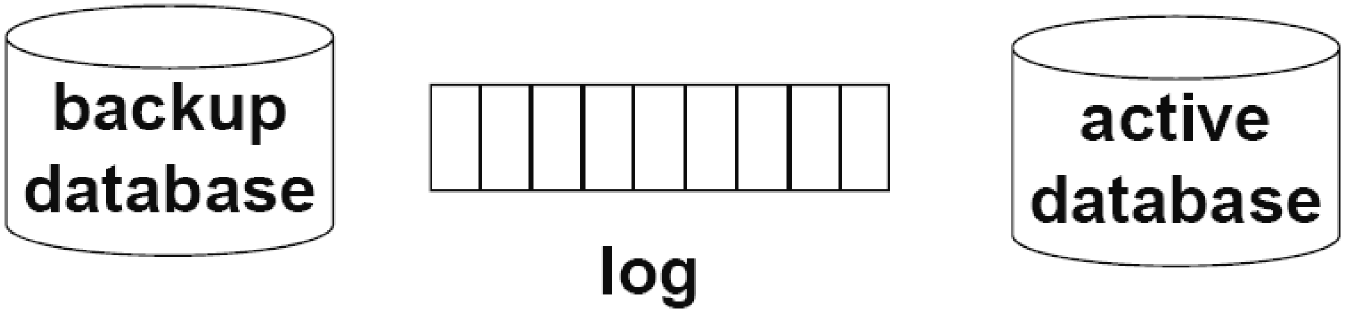
- Backwards pass (end of log ➲ most recent checkpoint start)
  - construct set S of committed transactions
  - undo actions of transactions not in S
- Forward pass (latest checkpoint start ➲ end of log)
  - redo actions of S transactions



Recovery Process (2 of 2)

- First for **uncommitted transactions we always undo them** (regardless of checkpointing)
- If the crash happens <u>before</u> the **END CHECKPOINT** then we redo completed transactions from the latest successful **START CHECKPOIT** or from the start of the log file
- If the crash happens <u>after</u> the **END CHECKPOINT** then we only redo actions of completed transactions <u>after</u> **START CHECKPOIT** (NOT from the start of the log file)

Recovery from Media Failure using Backup + Log

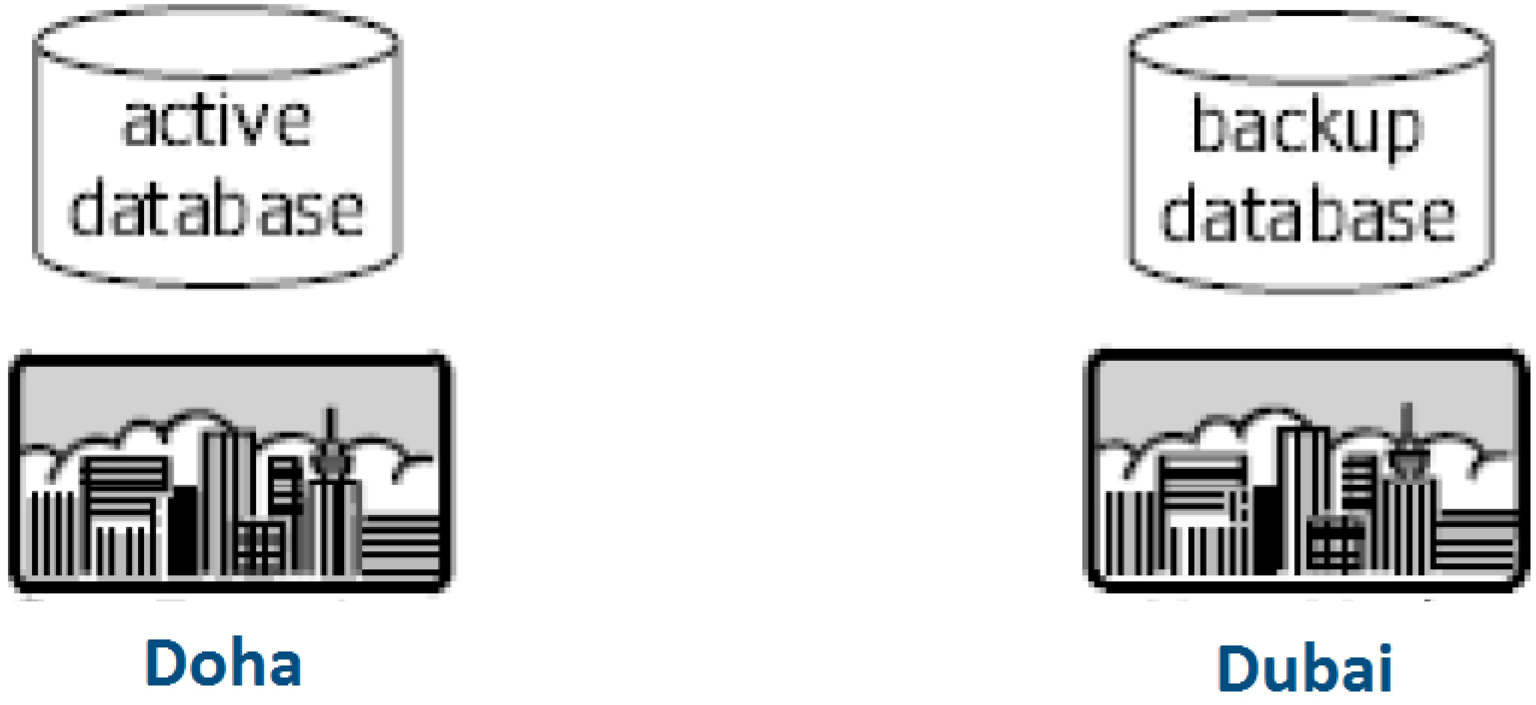


If active database is lost:
(1) Restore active database from backup
(2) Bring up-to-date using redo entries in log

Catastrophic Failure

- Array of disks will not help in case of: fire, earthquake, vandalism, viruses

=> Solution: Geographically distributed copies!


active database
backup database
Doha
Dubai