Transaction Management

(Concurrency Control)

Deadlock Prevention Scheme

Lock-based concurrency control cannot prevent deadlocks.

We need an active solution, a solution not based on locks maybe?

A different approach that guarantees serializability involves using transaction timestamps to order transaction execution for an equivalent serial schedule.

Concurrency Control with Timestamps

- Not based on locks!
- Assign each transaction $\mathbf{T_i}$ a timestamp T $\mathbf{TS(T_i)}$
- unique identifier to identify a transaction

Implementing Timestamps

Timestamps can be generated in several ways.

- Possibility 1:
  - A simple **counter** (e.g., int counter)
  - (Increment value each time its value is assigned to a transaction.)
- Possibility 2:
  - Use the current **date/time value** of the system clock.

Timestamp Ordering

Concurrency control techniques based on timestamp ordering do not use locks; hence, *deadlocks cannot occur*.

For two transactions T_i and T_j that may be involved in a deadlock.
Assume T_i wants a lock that T_j holds, two policies are possible:

- Policy 1 (**Wait-Die Protocol**):
  - If T_i is older, T_i waits for T_j
  - If T_i is younger, T_i aborts*
- Policy 2 (**Wound-wait Protocol**):
  - If T_i is younger, T_i waits for T_j
  - If T_i is older, T_j aborts*

Note:

1. T_i older than T_j if $\mathbf{TS(T_i) < TS(T_j)}$
2. * If a transaction re-starts, it retains its original timestamp

Cyclic Restart

Notice: both the schemes end up aborting the younger of the two transaction. Why?

Tip: If a transaction re-starts, it retains its original timestamp

It could lead to **cyclic restart**

- A kind of “live lock” can occur - transactions may be constantly aborted and restarted.

Deadlock Prevention Scheme

- Concurrency control via timestamp ordering:
  - Ensures that the result final schedule recorded is equivalent to executing the transactions serially in timestamp order
  - Therefore, ensuring serializability

- Compare this with concurrency control via 2PL:
  - In 2PL, a schedule is serializable by being equivalent to some serial schedule allowed by the locking protocols.
  - In timestamp ordering, however, the schedule is equivalent to the serial order corresponding to the order of the transaction timestamps.

2PL vs. TSO

A Comparison among two-phase locking (2PL), timestamp ordering (TSO) concurrency control techniques.

1. 2PL can offer the greatest concurrency degree (in average); but will result in deadlocks. To resolve the deadlocks, either
   - need additional computation to detect deadlocks and to resolve the deadlocks, or
   - reduce the concurrency degree to prevent deadlocks by adding other restrictions.

2PL vs. TSO (cont.)

A Comparison among two-phase locking (2PL), timestamp ordering (TSO) concurrency control techniques.

1. If most transactions are very short, we can use 2PL + deadlock detection and resolution.
2. TSO has a less concurrency degree than that of 2PL if a proper deadlock resolution is found. However, TSO does not cause deadlocks. Other problems, such as cyclic restart and cascading rollback, will appear in TSO.

Transaction Management

(Database Recovery)

Transaction Failures

Up to now, we discuss schedules assuming implicitly that there are no transaction failures.

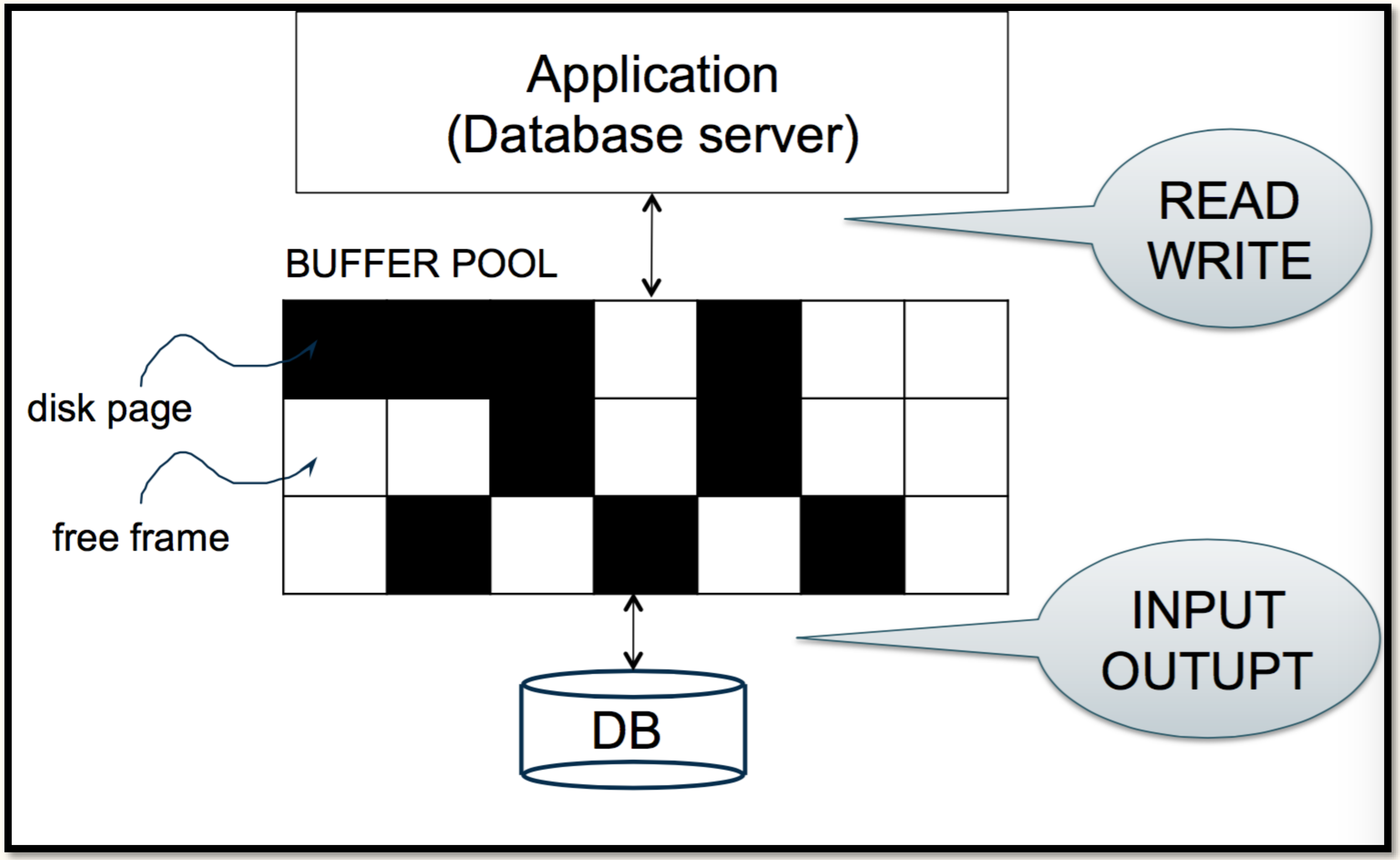
1. We need to consider the effect of transaction failures during concurrent execution.
2. Why? If a transaction T_i fails, we need to undo the effect of this transaction to ensure the atomicity property.
3. By atomicity, it requires that any transaction T_j that is dependent on T_i (i.e., T_j has read data written by T_i) is also aborted.

Recall Failures

1. **A computer failure (system crash).** A hardware, software, or network error occurs in the computer system during transaction execution.
2. **A transaction or system error.** Some operation in the transaction may causeit to fail.
3. **Local errors or exception conditions detected by the transaction.** During transaction execution, certain conditions may occur that necessitate cancellation of the transaction
4. **Concurrency control enforcement.** The concurrency control method may decide to abort a transaction because it violates serializability or it may abort one or more transactions to resolve a state of deadlock among several transactions.

   (noncatastrophicfailures)

Buffer Management in a DBMS



Database Recovery

- The recovery strategy is to identify any changes that may cause an inconsistency in the database.
- We need to have information to *rollback* an unsuccessful transaction (undo any partial updates).

System Log

To be able to recover from failures that affect transactions, the system maintains a **log** to keep track of all transaction operations that affect the values of database items.

1. The system needs to record the states information to recover failures correctly.
2. The information is maintained in a log (also called journal or audit trail).
3. The system log is kept in hard disk but maintains its current contents in main memory.

System Log

Log records

1. [start transaction, *T*]: Start transaction marker, records that transaction *T* has started execution.
2. [read item, *T*, *X*]: Records that transaction *T* has read the value of database item *X*.
3. [write item, *T*, *X*, old value, new value]: Records that *T* has changed the value of database item *X* from old value to new value.

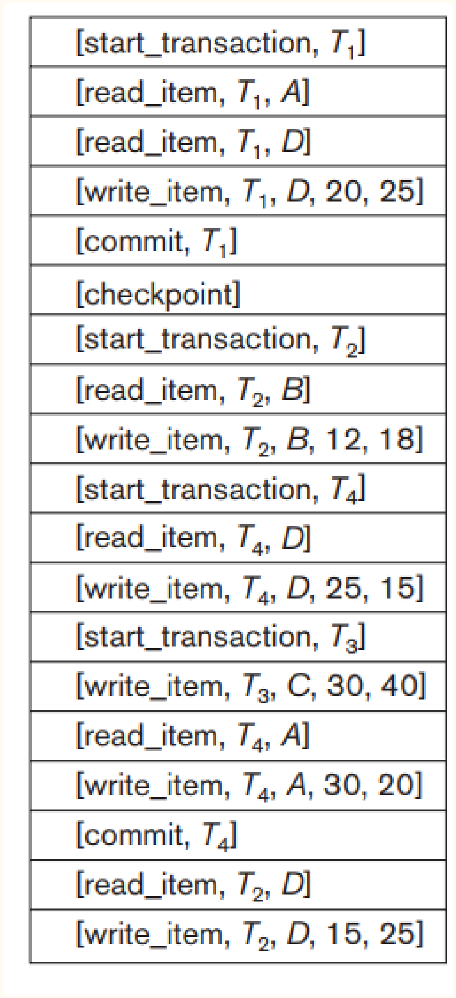
System Log

Log records (Cont.)

1. [commit, *T*]: Commit transaction marker, records that transaction *T* has completed successfully, and confirms that its effect can be committed (recorded permanently) to the database.
2. [abort, *T*]: Records that transaction *T* has been aborted.

System Log

Sample log



[start_transaction, T_1]
[read_item, T_1, A]
[read_item, T_1, D]
[write_item, T_1, D, 20, 25]
[commit, T_1]
[checkpoint]
[start_transaction, T_2]
[read_item, T_2, B]
[write_item, T_2, B, 12, 18]
[start_transaction, T_4]
[read_item, T_4, D]
[write_item, T_4, D, 25, 15]
[start_transaction, T_3]
[write_item, T_3, C, 30, 40]
[read_item, T_4, A]
[write_item, T_4, A, 30, 20]
[commit, T_4]
[read_item, T_2, D]
[write_item, T_2, D, 15, 25]

Immediate Update Techniques

In the **immediate update** techniques, the database *may be updated* by some operations of a transaction *before* the transaction reaches its commit point.

However, these operations must also be recorded in the log *on disk* by force-writing *before* they are applied to the database on disk, making recovery still possible.

If a transaction fails after recording some changes in the database on disk but before reaching its commit point, the effect of its operations on the database must be undone; that is, the transaction must be rolled back. In the general case of immediate update, both *undo* and *redo* may be required during recovery.

Transaction Roll Back

If a transaction fails for whatever reason after updating the database, but before the transaction commits, it may be necessary to **roll back** the transaction. If any data item values have been changed by the transaction and written to the database, they must be restored to their previous values

Undo And Redo

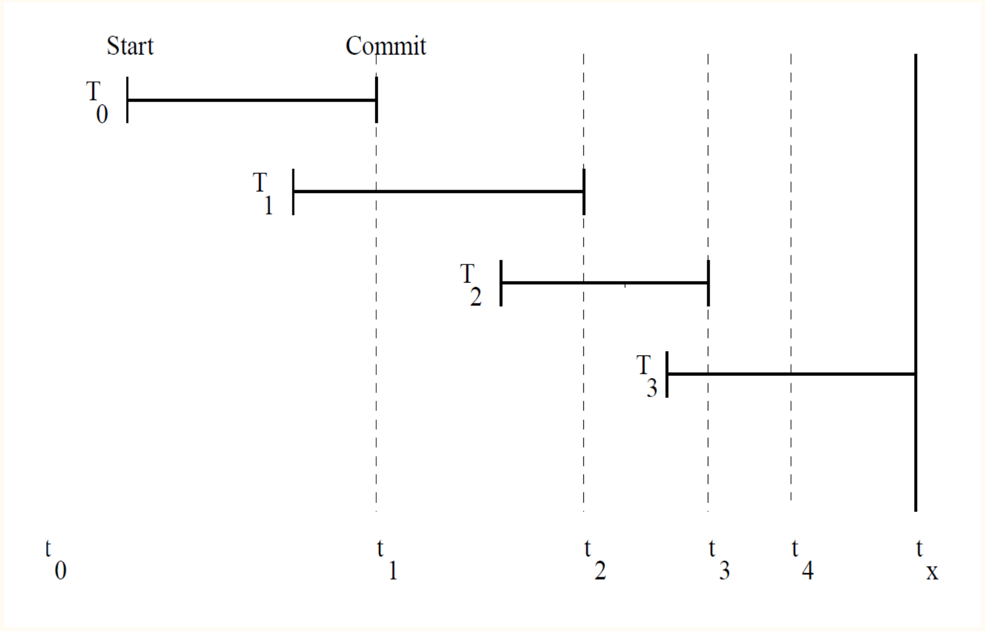
Procedure UNDO (WRITE_OP):

- Undoing a write_item operation WRITE_OP
- Consists of eaxamining its log entry [write_item, *T*, *X*, old_value, new_value] and setting the value of item *X* in the database to old_value.
- Undoing a number of write_item operations from one or more transactions from the log must proceed in the *reverse order* from the order in which the operations were written in the log.

Procedure REDO (WRITE_OP)**:**

- Redoing a write_item operation WRITE_OP
- Consists of examining its log entry [write_item, *T*, *X*, new_value] and setting the value of item *X* in the database to new_value.

Log-based Recovery



Assume that the database was recently created, the diagram shows transactions up until a crash.

The database state will be somewhere between that at t_0 and the state at t_x (also true for log entries)

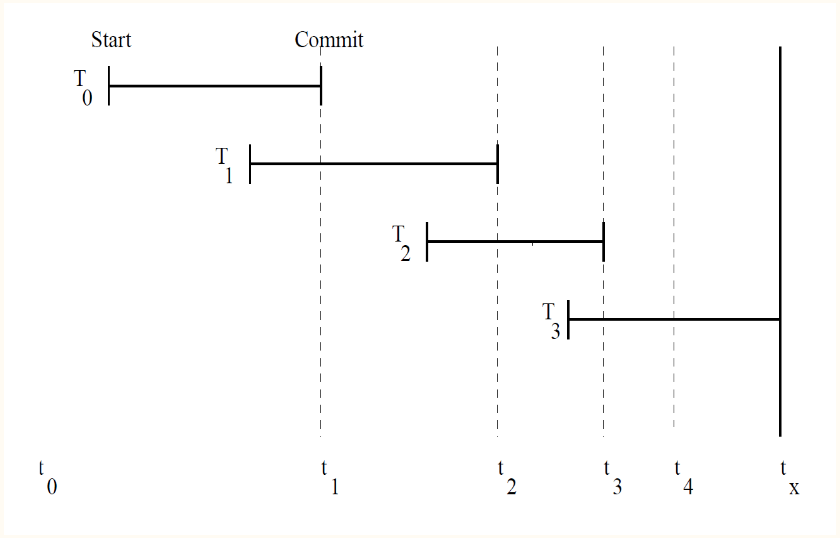
Write-Ahead Logging

Write-ahead log strategy

This means that before a series of transactions is handled

1. old data values must be force-written to the log (i.e. the buffer must be copied to log) before any change can be made to the database, and
2. the transaction is regarded as committed when the new data values and the commit marker have been force-written to the log.

Log-based Recovery



Suppose the log was last written to disk at t_4 (shortly after t_3)

We would know that T_0, T_1 and T_2 have committed and their effects **should be** reflected in the database after recovery.

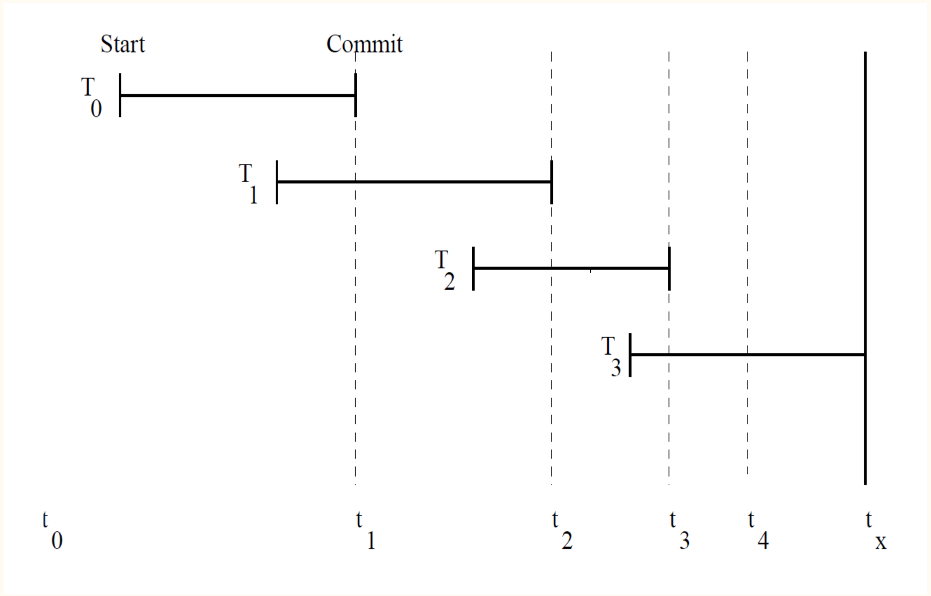
We also know that T_3 has started, may have modified some data, but is not committed. Thus T_3 should be undone.

Rolling back (Undo) T_3

With a write-ahead strategy, we would be able to make some recovery by rolling back T_3

Step 1:

- Undo the values written by T_3 to the old data values from the log



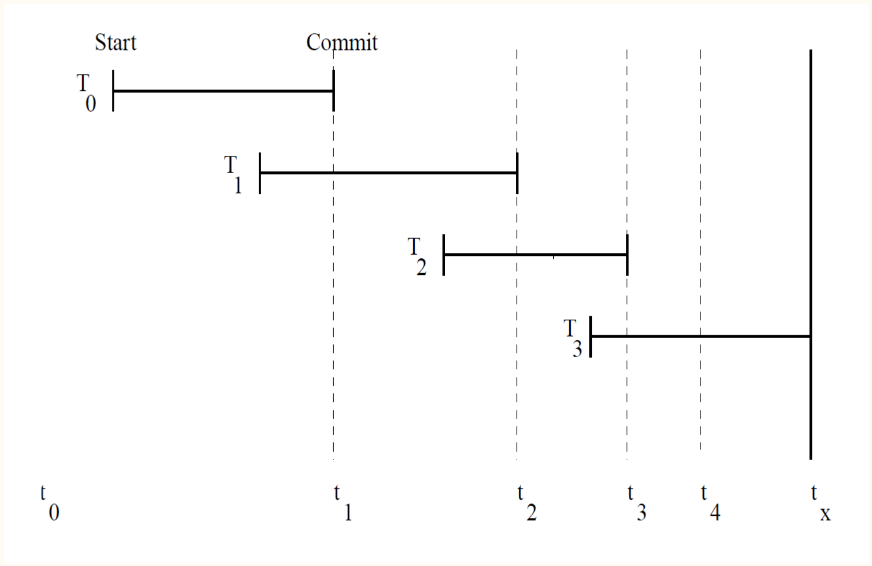
- Undo helps guarantee **atomicity** in **ACID**

Redoing T_0 … T_2

With a write-ahead strategy, we would be able to make some recovery by rolling back T_3

Step 2:

- Redoing the changes made by T_0 … T_2 using the new data values (for these committed transactions) from the log.



- Redo gaurentees **durability** in **ACID**

Checkpoints

Notice also that using this system, the longer the time between crashes, the longer recovery may take.

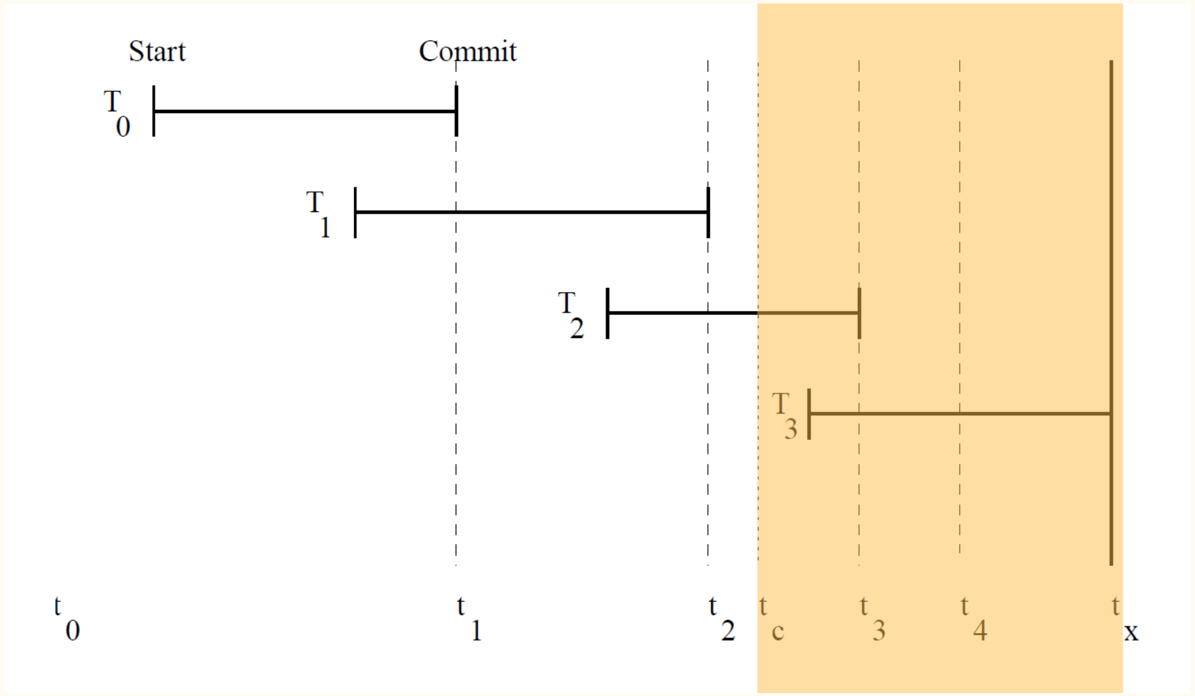
Checkpoints

To reduce this problem, the system could take *checkpoints* at regular intervals.

Taking a checkpoint consists of the following actions:

1. Suspend execution of transactions temporarily.
2. Force-write all main memory buffers that have been modified to disk.
3. Write a [checkpoint] record to the log, and force-write the log to disk.
4. Resume executing transactions.

Log Checkpoints



In our example, suppose a checkpoint is taken at time t_c. Then on recovery we only need redo T_2.

Recall Failures

1. **A computer failure (system crash).** A hardware, software, or network error occurs in the computer system during transaction execution.
2. **A transaction or system error.** Some operation in the transaction may cause it to fail.
3. **Local errors or exception conditions detected by the transaction.** During transaction execution, certain conditions may occur that necessitate cancellation of the transaction
4. **Concurrency control enforcement.** The concurrency control method may decide to abort a transaction because it violates serializability or it may abort one or more transactions to resolve a state of deadlock among several transactions.

Catastrophic Failures

1. **Disk failure.** Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash.
2. **Physical problems and catastrophes.** This refers to an endless list of problems that includes power or fire, theft, overwriting disks by mistake…

Catastrophic Failures

So far, all the techniques we have discussed apply to noncatastrophic failures

A key assumption has been that the system log is maintained on the disk and is not lost as a result of the failure.

The recovery techniques we have discussed use the entries in the system log to recover from failure by bringing the database back to a consistent state.

The recovery manager of a DBMS must also be equipped to handle more catastrophic failures such as disk crashes.

Database Backup

The main technique used to handle such crashes is a **database backup**

- (1) whole database and (2) the log are periodically copied onto a cheap storage medium such as magnetic tapes or other large capacity offline storage devices.
- The latest backup copy can be reloaded from the tape to the disk, and the system can be restarted.

To not lose all transactions they have performed since the last database backup. We also back up the system log at more frequent intervals than full database backup by periodically copying it to magnetic tape.