# CS3402 Chapter 11: Database Recovery

## A Transaction may Fail

- Why recovery is needed? (What causes a Transaction to abort)
  1. A computer failure (system crash):
     - A hardware or software error occurs in the computer system during transaction execution
     - If the hardware crashes, the contents of the computer's internal memory may be lost.
  2. A transaction error:
     - Some operation in the transaction may cause it to fail, such as integer overflow or division by zero
     - Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, the user may interrupt the transaction during its execution.

## A Transaction may Fail

3. Local errors or exception conditions detected by the transaction:
   - Certain conditions necessitate cancellation of the transaction. For example, data for the transaction may not be found
4. Concurrency control enforcement:
   - The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock

## A Transaction may Fail

5. 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. This may happen during a read or a write operation of the transaction.

6. Physical problems and catastrophes:

- ◆ This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, overwriting disks

## Transaction State

- A transaction is an atomic unit of work that is either completed in its entirety or not done at all (atomicity)
  - For recovery purposes, the system needs to keep track of when a transaction starts, terminates, and commits or aborts

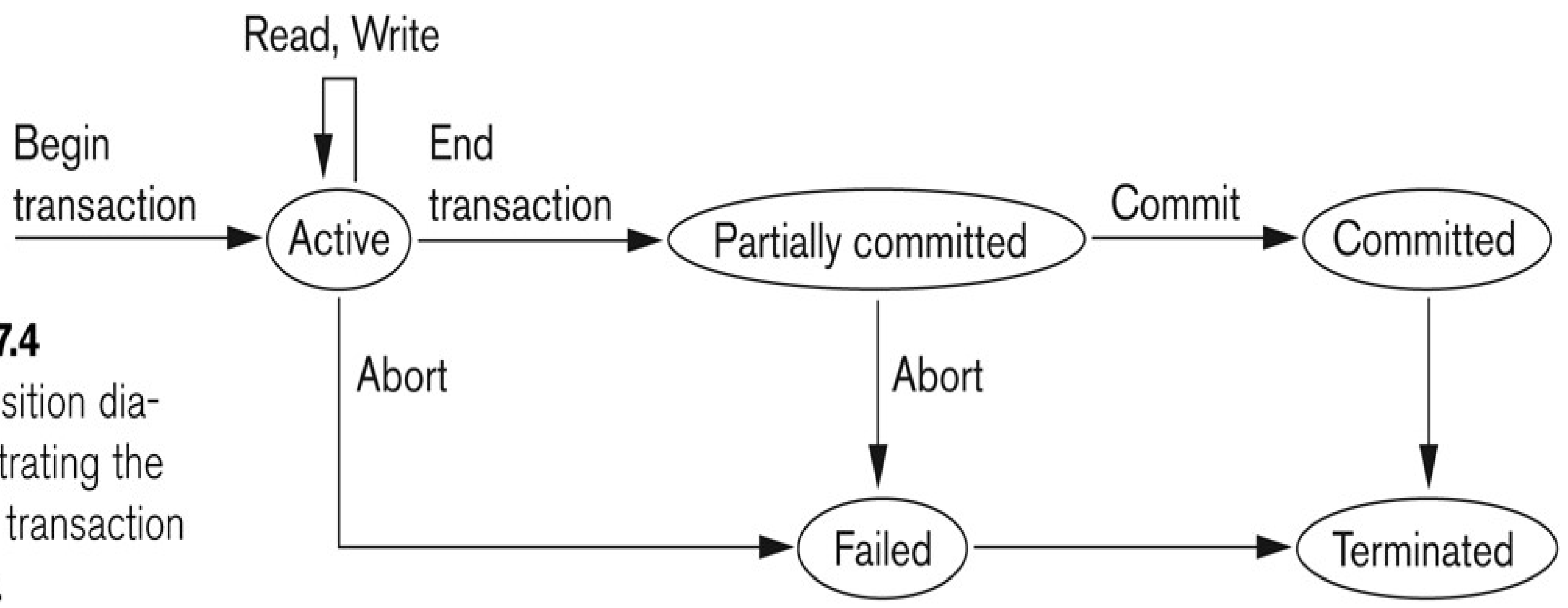


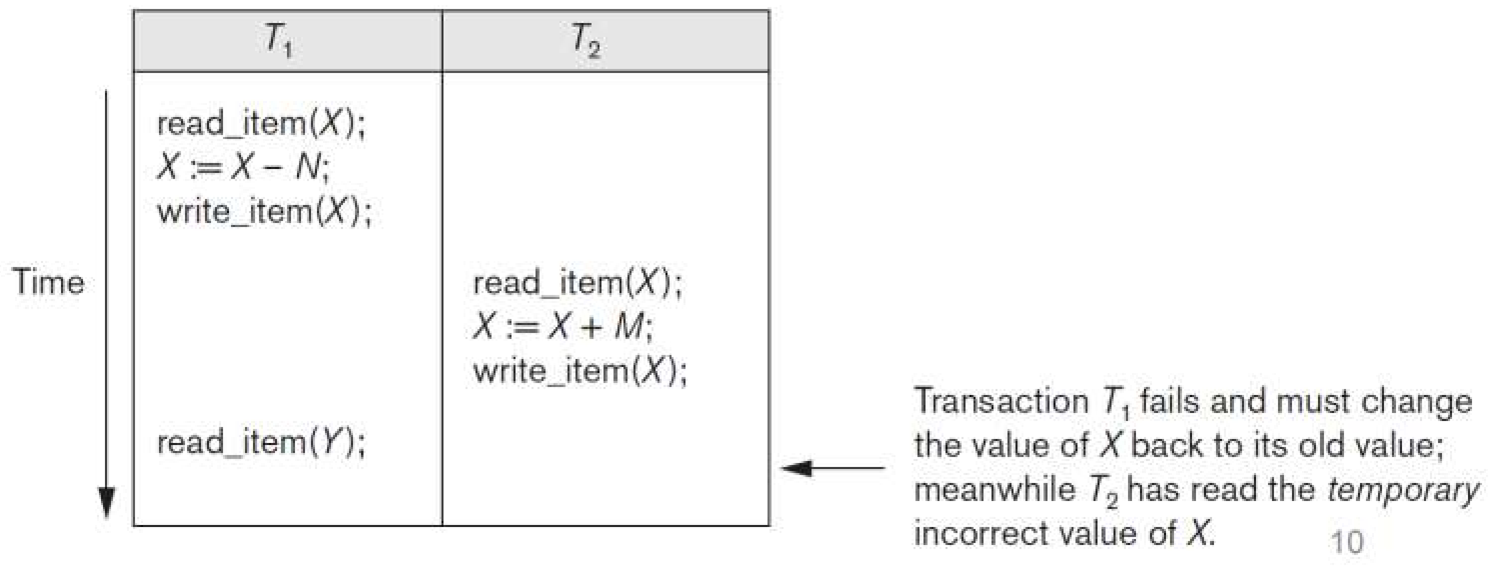
**Figure 17.4**
State transition diagram illustrating the states for transaction execution.

## *Undo Logging for Recovery*

- A log is a file of log records, each telling something about what some transaction has done
- As transactions execute, the log manager records in the log each important event (e.g., write operations)
- Logs are initially created in main memory and are allocated by the buffer manager
  - Why not on disk? Disk I/O takes a lot of time
- Logs are periodically copied to disk by "flush-log" operation
- If log records appear in nonvolatile storage (disk), we can use them to restore the database to a consistent state after a system crash
- After a system failure, all data in volatile storage (memory) will lose but the data in nonvolatile storage remain

# *Recoverability Problems in Concurrent Schedule*

- Dirty Read Problem (write/read conflicts)
  - This problem occurs when one transaction updates a database item and then the transaction fails for some reason. Meanwhile, the updated item is accessed (read) by another transaction before it is changed back (or rolled back) to its original value.



| $T_1$ | $T_2$ |
|---|---|
| read_item($X$); <br> $X := X - N$; <br> write_item($X$); | |
| | read_item($X$); <br> $X := X + M$; <br> write_item($X$); |
| read_item($Y$); | |

Transaction $T_1$ fails and must change the value of $X$ back to its old value; meanwhile $T_2$ has read the *temporary* incorrect value of $X$.

## *Schedules Classified on Recoverability*

- Non-Recoverable schedule:
  - The committed transaction with dirty-read problem cannot be rolled back.

- Recoverable schedule:
  - No committed transaction needs to be rolled back.
  - A schedule S is recoverable if

    For all Ti and Tj where Ti read an item written by Tj,

    Tj commits before Ti

## Example 1 (Non-Recoverable vs Recoverable)

| T1 | T2 |
|---|---|
| Read(A) | |
| A=A+1 | |
| Write(A) | |
| | Read(A) |
| | Read(B) |
| | B=B+A |
| | … |
| | Commit/Abort |
| Commit/Abort | |

Non-recoverable

If T2 commits and then

T1 Aborts…

| T1 | T2 |
|---|---|
| Read(A) | |
| A=A+1 | |
| Write(A) | |
| | Read(A) |
| | Read(B) |
| | B=B+A |
| | … |
| Commit/Abort | |
| | Commit/Abort |

Recoverable

T2 commits after T1

## Schedules Classified on Recoverability

- Cascaded rollback:
  - A single rollback leads to a series of rollback
  - All uncommitted transactions that read data items from a failed (aborted) transaction must be rolled back
- Cascadeless schedule:
  - Every transaction reads only the items that are written by committed transactions.
  - In other words

    Before Ti reads an item written by Tj, Tj is already committed
- Strict Schedules:
  - A schedule in which a transaction can neither read or write an item X until the last transaction that wrote X has committed.

## Example 2 (Non-Cascadeless vs Cascadeless)

| T1 | T2 |
|---|---|
| Read(A) | |
| A=A+1 | |
| Write(A) | |
| | Read(A) |
| | Read(B) |
| | B=B+A |
| | …. |
| Commit/Abort | |
| | Commit/Abor |

**Non-cascadeless**

T1 rollbacks, T2 also has to roll back

| T1 | T2 |
|---|---|
| Read(A) | |
| A=A+1 | |
| Write(A) | |
| Commit/Abort | |
| | Read(A) |
| | Read(B) |
| | B=B+A |
| | … |
| | Commit/Abort |

**Cacadeless**

T2 read A after T1 commits

# *Example 3 (Cascadeless vs Strict )*

- **S: w1 (X,5); w2 (X,8); a1 (T1 aborts)**
- Initially, X=9
- T1 writes a value 5 for X (keeping 9 as "before image")
- T2 writes a value 8 for X (keeping 5 as "before image")
- Then T1 aborts

- Cascadeless schedule:
  - **If the system restore according to the "before image" kept in T1, then 9 will be the value for X now, which means, the effect of T2 is lost**

- Strict Schedules:
  - **w2 can not happen until T1 commits**

## ACID Properties of Transactions

- Atomicity: A transaction is an atomic unit of processing. It is either performed completely or not performed at all (all or nothing)
- Consistency: A correct execution of a transaction must take the database from one consistent state to another (correctness)
- Isolation: A transaction should not make its updates visible to other transactions until it is committed (no partial results)
- Durability: Once a transaction changes the database state and the changes are committed, these changes must never be lost because of subsequent failure (committed and permanent results)

## References

- 6e
  - *Chapter 20, pages 721-750*