Transaction Processing

SWEN304/SWEN439

Lecturer: Dr Hui Ma

Engineering and Computer Science

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Slides by: Pavle Morgan & Hui Ma

Outline

- Introduction to transaction processing
- How transactions influence database consistency
  - Lost update problem
  - Dirty read problem
  - Unrepeatable read problem
- Commit
- Transaction state transition diagram
- Desirable Properties of Transactions: ACID
  - *Readings from the textbook:*
    - *Chapter 21*

Introduction to Transaction Processing (1)

- The concept of **transaction** is used to describe logical units of a database processing
- **Transaction processing systems** are systems with large databases and hundreds of concurrent users executing database transactions
- Examples of these systems are:
  - Airline reservations,
  - Banking,
  - Credit card processing,
  - Supermarket checkout, and

Introduction to Transaction Processing (2)

- **Single-User System**: At most one user at a time can use the system

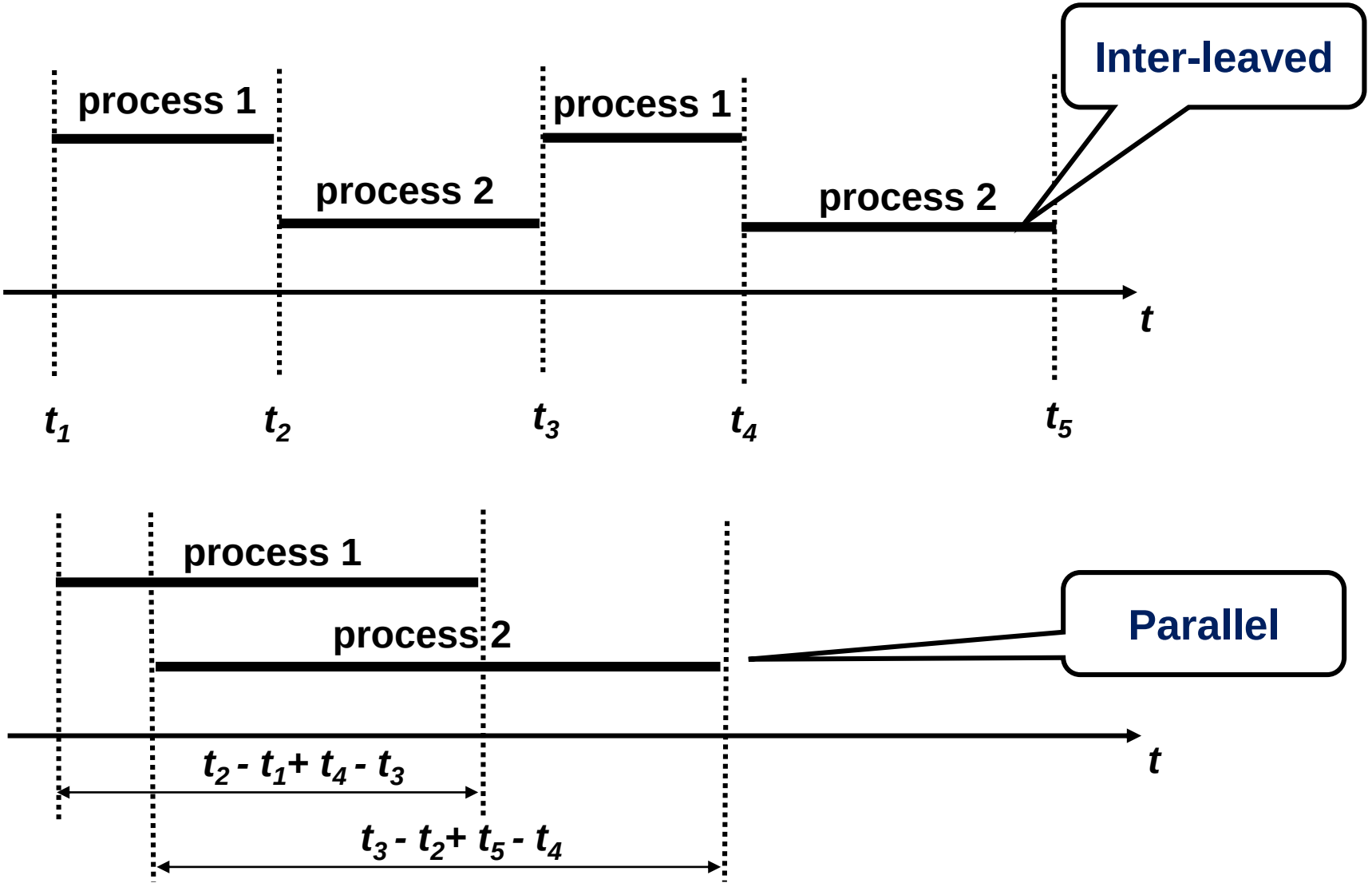
- **Multiuser System**: Many users can access the system concurrently because of multiprogramming
  - Multiprogramming operating systems execute some commands of one process, then suspend this process and execute some commands of another process

- Majority of database systems are of a multiuser type

Introduction to Transaction Processing (3)

- **Concurrency**
  - **Interleaved processing**: concurrent execution of processes is interleaved in a single CPU
  - **Parallel processing**: processes are concurrently executed in multiple CPUs

Interleaved and Parallel Processes



Introduction to Transaction Processing (4)

- **A Transaction**: logical unit of database processing that includes one or more access operations (read - retrieval, write - insert or update, delete)
- **A transaction (set of operations)** may be stand-alone specified in a high level language like SQL submitted interactively, or may be embedded within a program
- **Transaction boundaries**: Begin and End transaction
- An **application program** may contain several transactions separated by the Begin and End transaction boundaries

Introduction to Transaction Processing (5)

SIMPLE MODEL OF A DATABASE (for discussing transactions):

- **A database -** collection of named data items
- **Granularity of data** - a field, a record , or a whole disk block (Concepts are independent of granularity)
- Basic operations are **read** and **write**
  - **read_item(X)**: Reads a database item named X into a program variable. To simplify our notation, we assume that *the program variable is also named X.*
  - **write_item(X)**: Writes the value of program variable X into the database item named X.

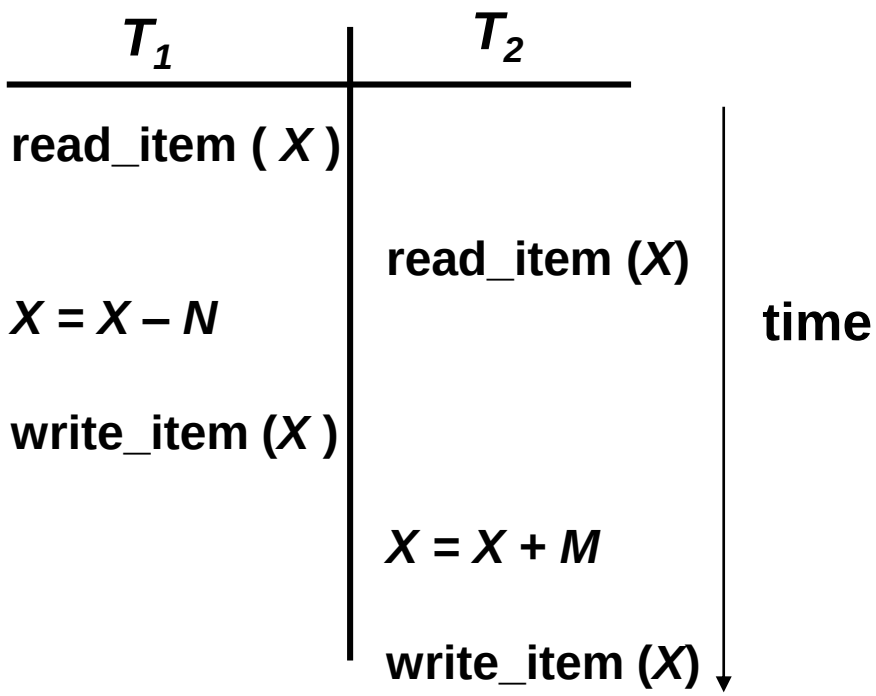
Introduction to Transaction Processing (6)

- In multiuser transaction processing systems, users execute database transactions **concurrently**
- Most often, concurrent means **interleaved**
- Users can attempt to modify the same database items at the same time, and that is potential source of database **inconsistency**
- Checking database integrity constraints is not enough to protect a database from threats induced by its concurrent use

Sources of Database Inconsistency

- Uncontrolled execution of database transactions in a multiuser environment can lead to database inconsistency:
  - lost update,
  - dirty read, and
  - unrepeatable read

Lost Update Problem

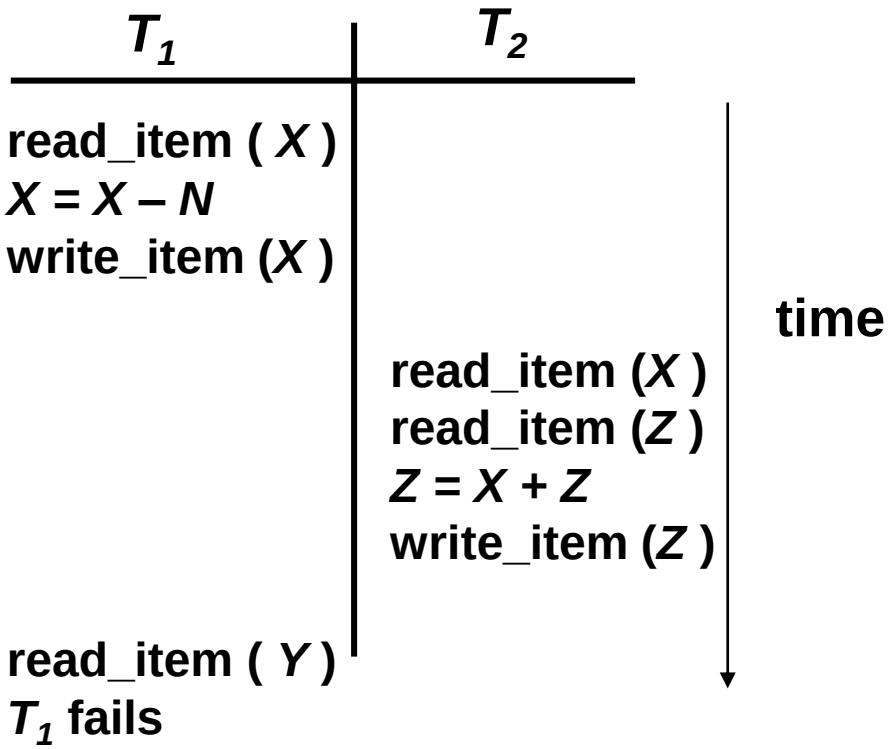


- Generally, lost update problem is characterized by:
  - T_2 reads X,
  - T_1 writes X, and
  - T_2 writes X

- After termination of T_2, $X = X + M$.
- T_1's update of X has been lost because T_2 has overwritten X

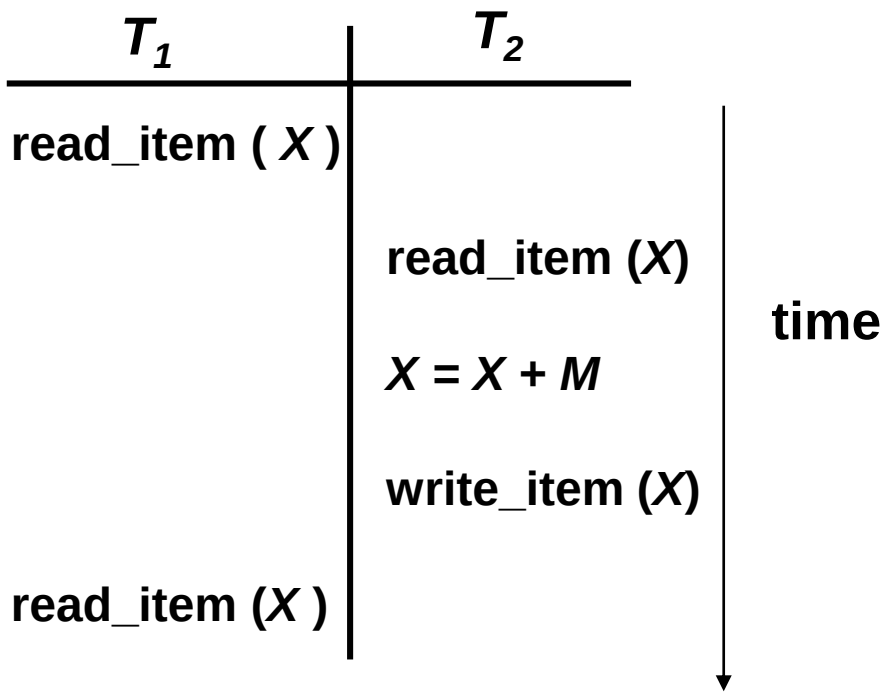
Note: In this example and all that follow, the pair of commands (read_item(X), write_item(X)) is a replacements for an SQL UPDATE statement

Dirty Read Problem



- Generally, dirty read problem is characterized by:
  - T_1 writes X,
  - T_2 reads X, and
  - T_1 fails
- Since T_1 failed, DBMS is going to undo the changes it made against the database
- T_2 has already read item X = X - N value, and that value is going to be altered by DBMS back to X

Unrepeatable Read Problem



- Generally, unrepeatable read problem is characterized by:
  - T_1 reads X,
  - T_2 writes X, and
  - T_1 reads X

- Transaction T_1 has got two different values of X in two subsequent reads, because T_2 has changed it in the meantime

A Question for You

- What is the difference between:
  - Dirty read and
  - Unrepeatable read

  problem?
- The difference is:
  - The dirty read is a consequence of reading updates made by a transaction before it has successfully finished (and has even failed later).
  - The unrepeatable read is a consequence of allowing a transaction to read and alter data that the other one has read

Prevention of Concurrency Anomalies

- Lost update, dirty read and unrepeatable read are called **concurrency anomalies**
- The concurrency control part of a DBMS has the task to prevent these problems
- DBMS is responsible to ensure that
  - either all operations of a transaction are successfully executed and their effect is permanently stored in the database,
  - or it happens as if the transaction were even not started
- The effect of a partially executed transaction has to be undone

Types of Failures

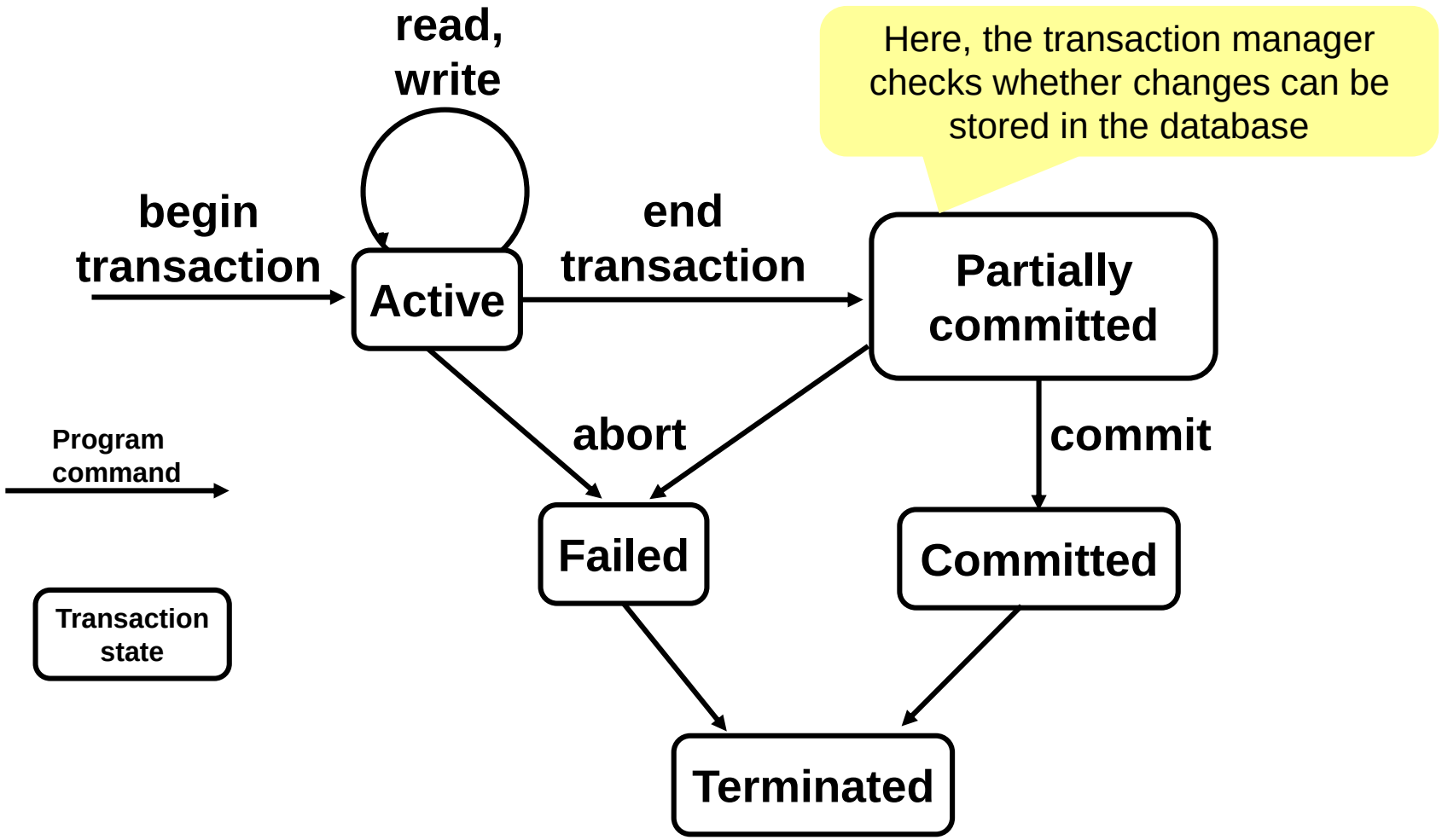
- A transaction can be partially executed due to:
  - A computer failure
    - E.g. hardware, software, network
  - A transaction error: some operation in the transaction may cause it to fail
    - E.g. integer overflow, division by zero, user interrupt
  - An exception condition: certain conditions necessitate cancellation of the transaction
    - E.g. data not found, condition not satisfied
  - A concurrency control enforcement
    - E.g. dead lock, timeout,…
  - An abort command in transaction program

Transaction and System Concepts (1)

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

- **Transaction states**:
  - Active state
  - Partially committed state
  - Committed state
  - Failed state
  - Terminated State

Transaction State Transition Diagram



Transaction and System Concepts (2)

- Recovery manager keeps track of the following operations:
  - **begin_transaction**: marks the beginning of transaction execution
  - **read** or **write**: specify *read* or *write* operations on the database items that are executed as part of a transaction
  - **end_transaction**: specifies that *read* and *write* transaction operations have ended and marks the end limit of transaction execution
    - At this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database or whether the transaction has to be aborted because it violates concurrency control or for some other reason

Transaction and System Concepts (3)

- Recovery manager keeps track of the following operations (cont):
  - **commit_transaction**: signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone
  - **rollback** (or **abort**): signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone

Commit and Abort

- A transaction reaches its commit point when all of its operations that access the database have been executed successfully and the effect of all transaction operations on the database have been stored somewhere permanently
- Beyond the commit point, the effect of a transaction is assumed to be permanently recorded in the database
- If a transaction does not reach its commit point it has to be **rolled back (aborted)**

Transaction and System Concepts (4)

- Recovery techniques use the following operators:
  - **undo**: similar to rollback except that it applies to a single operation rather than to a whole transaction.
  - **redo**: specifies that certain *transaction operations* must be *redone* to ensure that all the operations of a committed transaction have been applied successfully to the database.

Desirable Properties of Transactions (1)

ACID properties:

- **Atomicity**: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all

- **Consistency preservation**: A correct execution of the transaction must take the database from one consistent state to another

Desirable Properties of Transactions (2)

ACID properties (cont.):

- **Isolation**: A transaction should not make its updates visible to other transactions until it is committed;
  - this property, when enforced strictly, solves the temporary update problem and makes cascading rollbacks of transactions unnecessary

- **Durability or permanency**: Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure

Summary

- Executing transaction in an interleaved way may bring a database in an inconsistent state
- Transaction anomalies are:
  - Lost update,
  - Dirty read, and
  - Unrepeatable read
- A DBMS is responsible to ensure that either all operations of a transaction are successfully executed, or it is rolled back
- When a transaction reaches its commit point, everything is safely and permanently stored somewhere