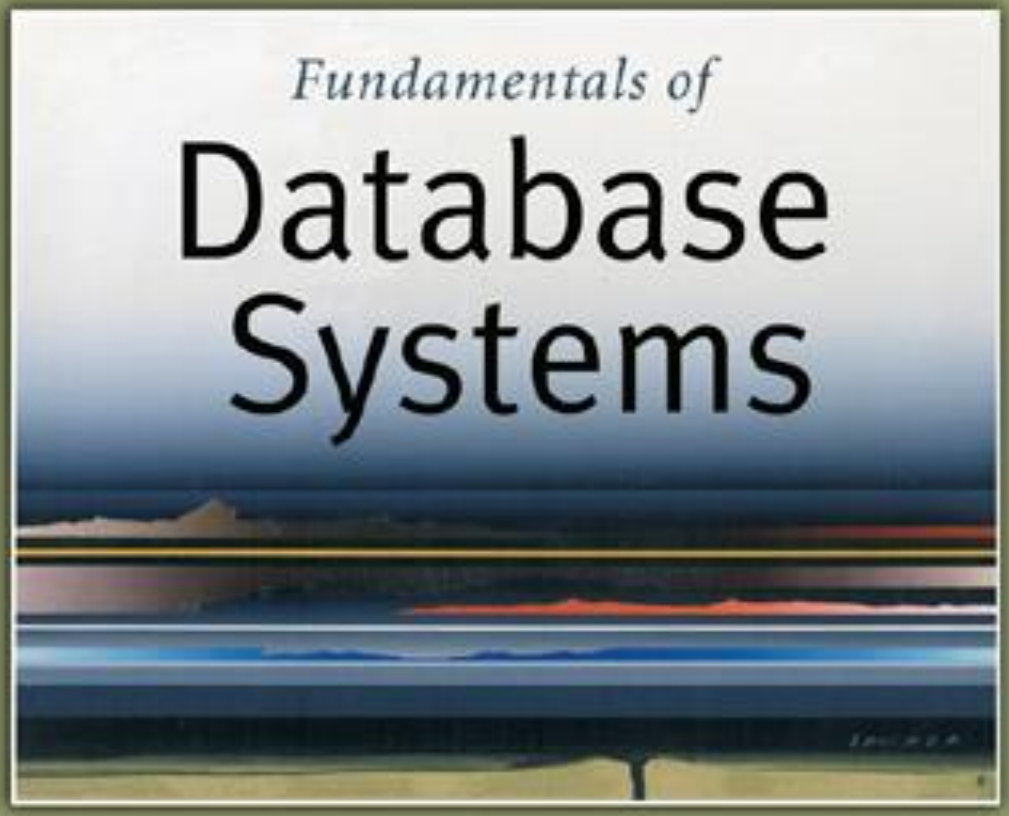
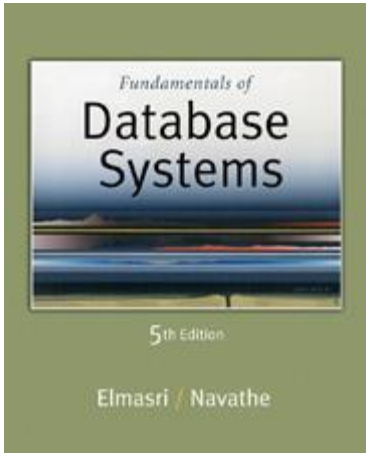


5th Edition

Elmasri / Navathe

Chapter 17

Introduction to Transaction Processing Concepts and Theory



1 Introduction to Transaction Processing (1)

- **Single-User System**:
  - At most one user at a time can use the system.
- **Multiuser System**:
  - Many users can access the system concurrently.
- **Concurrency**
  - **Interleaved processing**:
    - Concurrent execution of processes is interleaved in a single CPU
  - **Parallel processing**:
    - Processes are concurrently executed in multiple CPUs.

Introduction to Transaction Processing (2)

- A **Transaction**:
  - A transaction can be defined as a group of tasks or set of operations that includes one or more access operations (read - retrieval, write - insert or update, delete).
- **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 (3)

- Basic access 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 (4)

READ AND WRITE OPERATIONS:

- Basic unit of data transfer from the disk to the computer main memory is one block
- read_item(X) command includes the following steps:
  - Find the address of the disk block that contains item X.
  - Copy that disk block into a buffer in main memory
  - Copy item X from the buffer to the program variable named X.

Introduction to Transaction Processing (5)

READ AND WRITE OPERATIONS (contd.):

- **write_item(X)** command includes the following steps:
  - Find the address of the disk block that contains item X.
  - Copy that disk block into a buffer in main memory
  - Copy item X from the program variable named X into its correct location in the buffer.
  - Store the updated block from the buffer back to disk.

Two sample transactions

- FIGURE 17.2 Two sample transactions:
  - (a) Transaction T1
  - (b) Transaction T2

(a) T_1

```
read_item (X);
X:=X-N;
write_item (X);
read_item (Y);
Y:=Y+N;
write_item (Y);
```

(b) T_2

```
read_item (X);
X:=X+M;
write_item (X);
```

Introduction to Transaction Processing (6)

(Refer Problems of Concurrency Control document in Moodle)

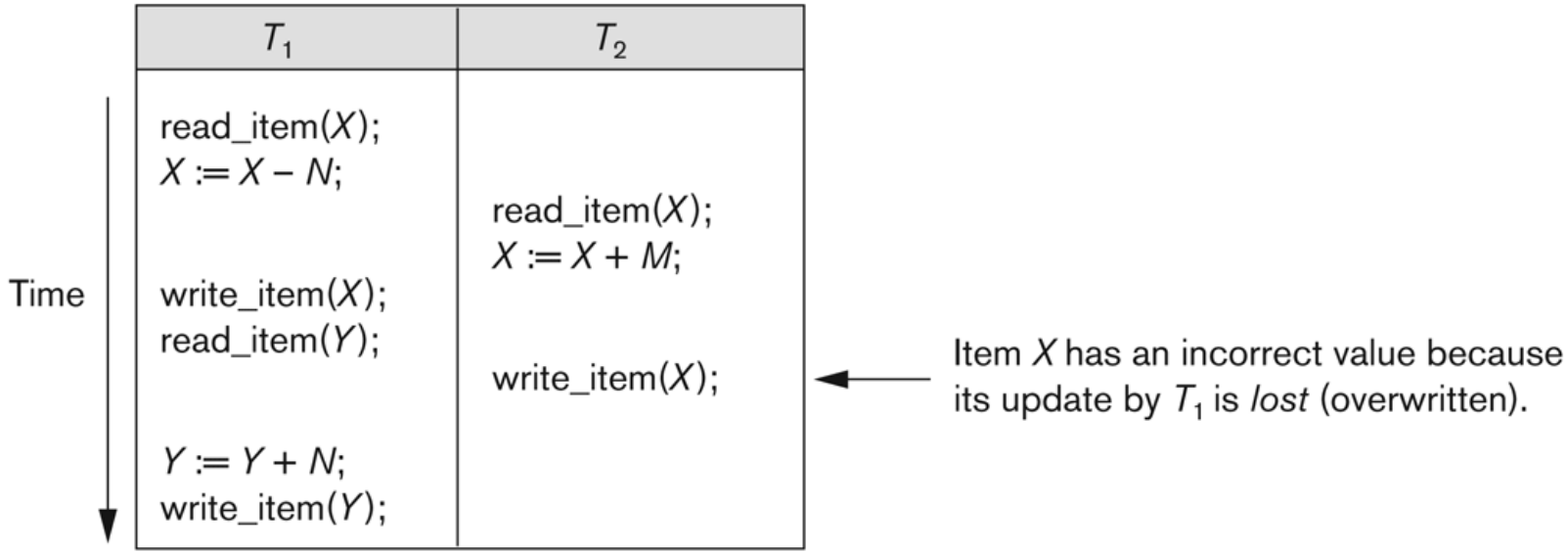
Types of problems in Concurrency Control:

- **The Lost Update Problem**
  - This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.
- **The Temporary Update (or Dirty Read) Problem**
  - This occurs when one transaction updates a database item and then the transaction fails for some reason (see Section 17.1.4).
  - The updated item is accessed by another transaction before it is changed back to its original value.
- **The Incorrect Summary Problem**
  - If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

Concurrent execution is uncontrolled: (a) The lost update problem.

Figure 17.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

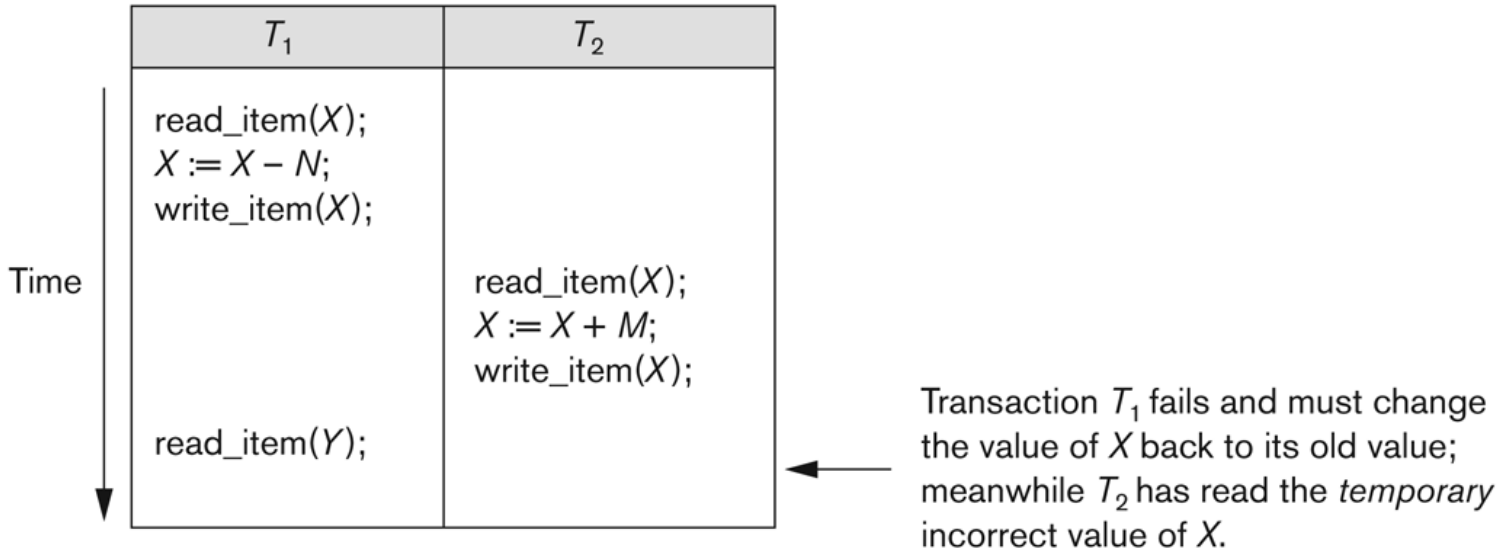
(a)



Concurrent execution is uncontrolled: (b) The temporary update problem.

Figure 17.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

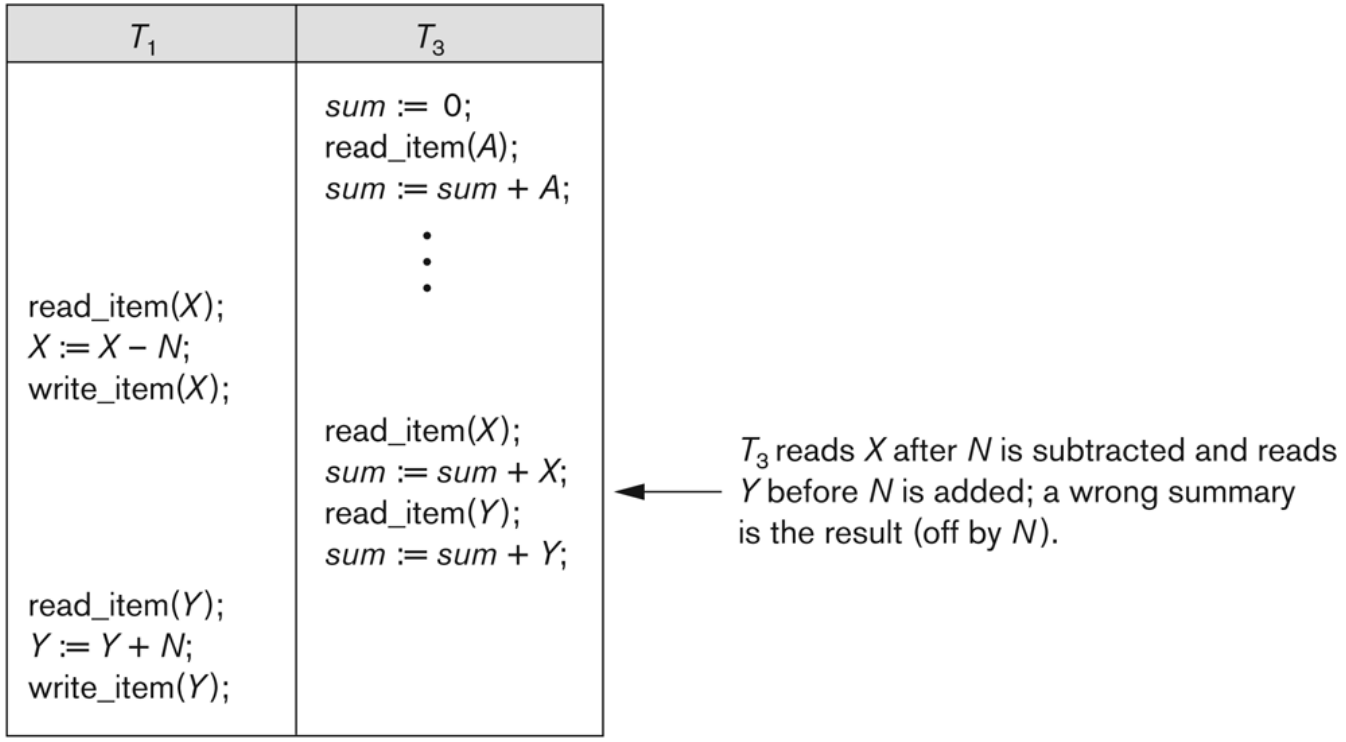
(b)



Concurrent execution is uncontrolled: (c) The incorrect summary problem.

Figure 17.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

(c)



Why **recovery** is needed:

Why **recovery** is needed:

(What causes a Transaction to fail)

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 or system 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.

Why **recovery** is needed:

Why **recovery** is needed (Contd.):

(What causes a Transaction to 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. A condition, such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal from that account, to be canceled.

   A programmed abort in the transaction causes it to fail.

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 (see Chapter 18).

Why **recovery** is needed:

Why **recovery** is needed (contd.):

(What causes a Transaction to 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 :

   This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

2 **Transaction states**:

- A **transaction** is an atomic unit of work that is either completed in its entirety or not done at all.
- **Transaction states**:
  - Active state
  - Partially committed state
  - Committed state
  - Failed state
  - Terminated State

Transaction states:

- Recovery manager keeps track of the following operations:
  - **begin_transaction**: This marks the beginning of transaction execution.
  - **read** or **write**: These specify read or write operations on the database items that are executed as part of a transaction.
  - **end_transaction**: This 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 states:

- Recovery manager keeps track of the following operations (cont):
  - **commit_transaction**: This 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**): This 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.

State transition diagram illustrating the states for transaction execution

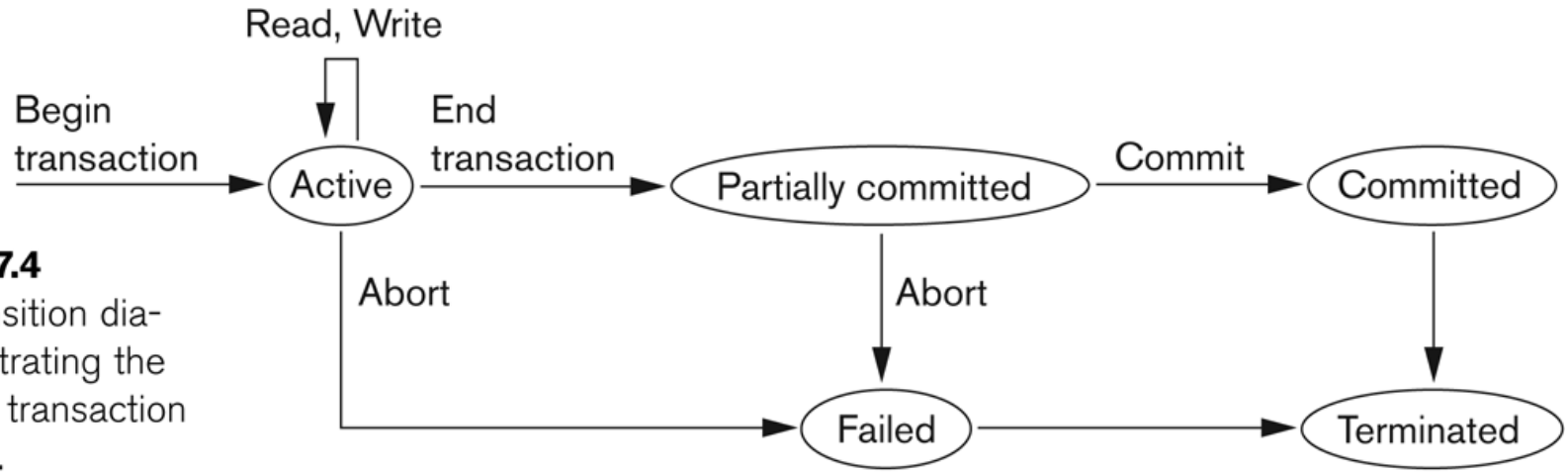


Figure 17.4
State transition diagram illustrating the states for transaction execution.

Transaction and System Concepts (6)

- The System Log
  - **Log** or **Journal**: The log keeps track of all transaction operations that affect the values of database items.
    - This information may be needed to permit recovery from transaction failures.
    - The log is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure.
    - In addition, the log is periodically backed up to archival storage (tape) to guard against such catastrophic failures.

Transaction and System Concepts (7)

- The System Log (cont):
  - T in the following discussion refers to a unique **transaction-id** that is generated automatically by the system and is used to identify each transaction:
  - Types of log record:
    - [start_transaction,T]: Records that transaction T has started execution.
    - [write_item,T,X,old_value,new_value]: Records that transaction T has changed the value of database item X from old_value to new_value.
    - [read_item,T,X]: Records that transaction T has read the value of database item X.
    - [commit,T]: Records that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
    - [abort,T]: Records that transaction T has been aborted.

Transaction and System Concepts (10)

Commit and rollback :

- **Definition a Commit : a commit** refers to the saving of data permanently after a set of changes.

Roll Back :

- restoring a database to a previous state by canceling a specific transaction

- Abort :

  abort means to terminate a process

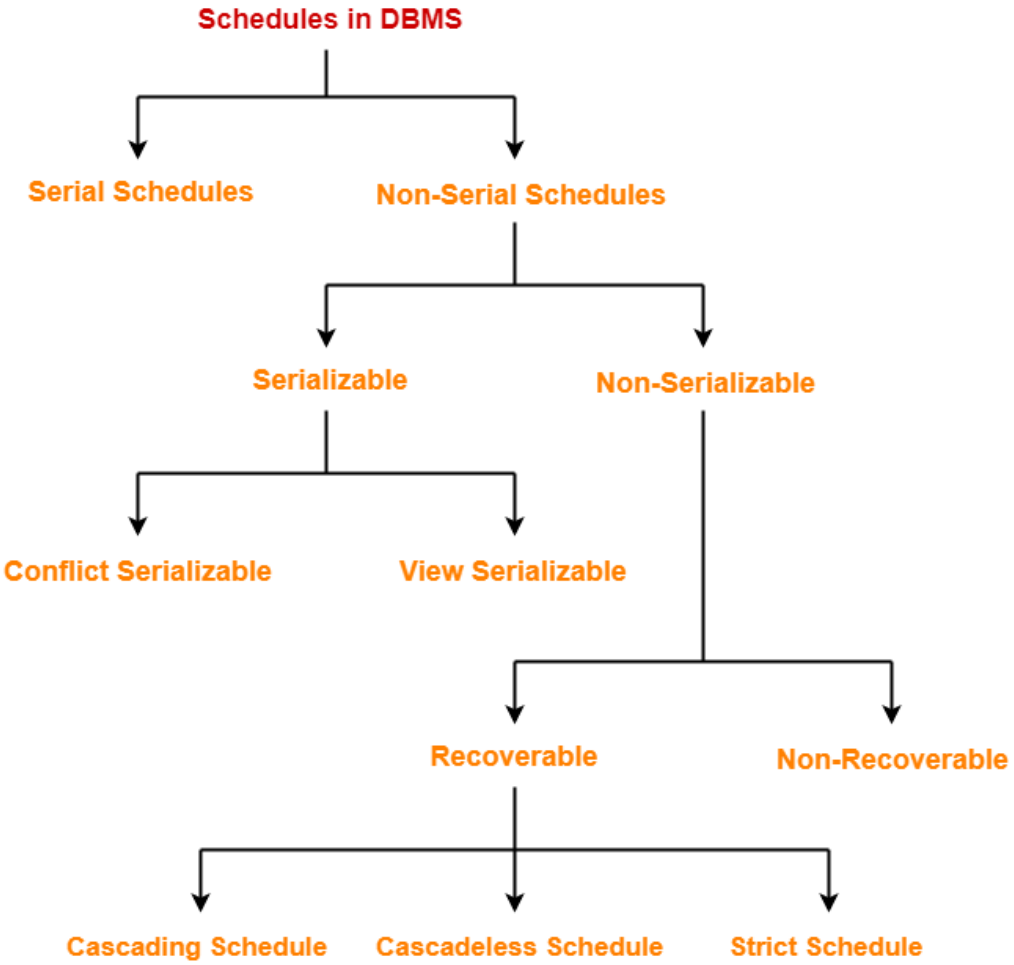
3 Desirable Properties of Transactions (1)

ACID properties:

- **Atomicity**: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
- **Consistency** : If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- **Isolation**: In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.
- **Durability** : Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.

4 Characterizing Schedules based on Recoverability (1)

- **Schedule:**
  - Set of transactions is know as schedule
  - A **schedule** S of n transactions T1, T2, …, Tn:



Non-serial schedule

T1	T2
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

Schedule S1

Serial Schedule

T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Schedule S2

Serializable schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

Non-Serializable Schedules-

- A non-serial schedule which is not serializable is called as a non-serializable schedule.
- A non-serializable schedule is not guaranteed to produce the the same effect as produced by some serial schedule on any consistent database.

Conflict Serializable Schedule

- A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.

Conflicting Operations

The two operations become conflicting if all conditions satisfy:

- Both belong to separate transactions.
- They have the same data item.
- They contain at least one write operation.

1. T1: Read(A) T2: Read(A)

T1	T2
Read(A)	
	Read(A)

Swapped

T1	T2
	Read(A)
Read(A)	

Schedule S1 **Schedule S2**

Here, S1 = S2. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)

T1	T2
Read(A)	
	Write(A)

Swapped

T1	T2
	Write(A)
Read(A)	

Here, S1 = S2. That means it is -conflict.

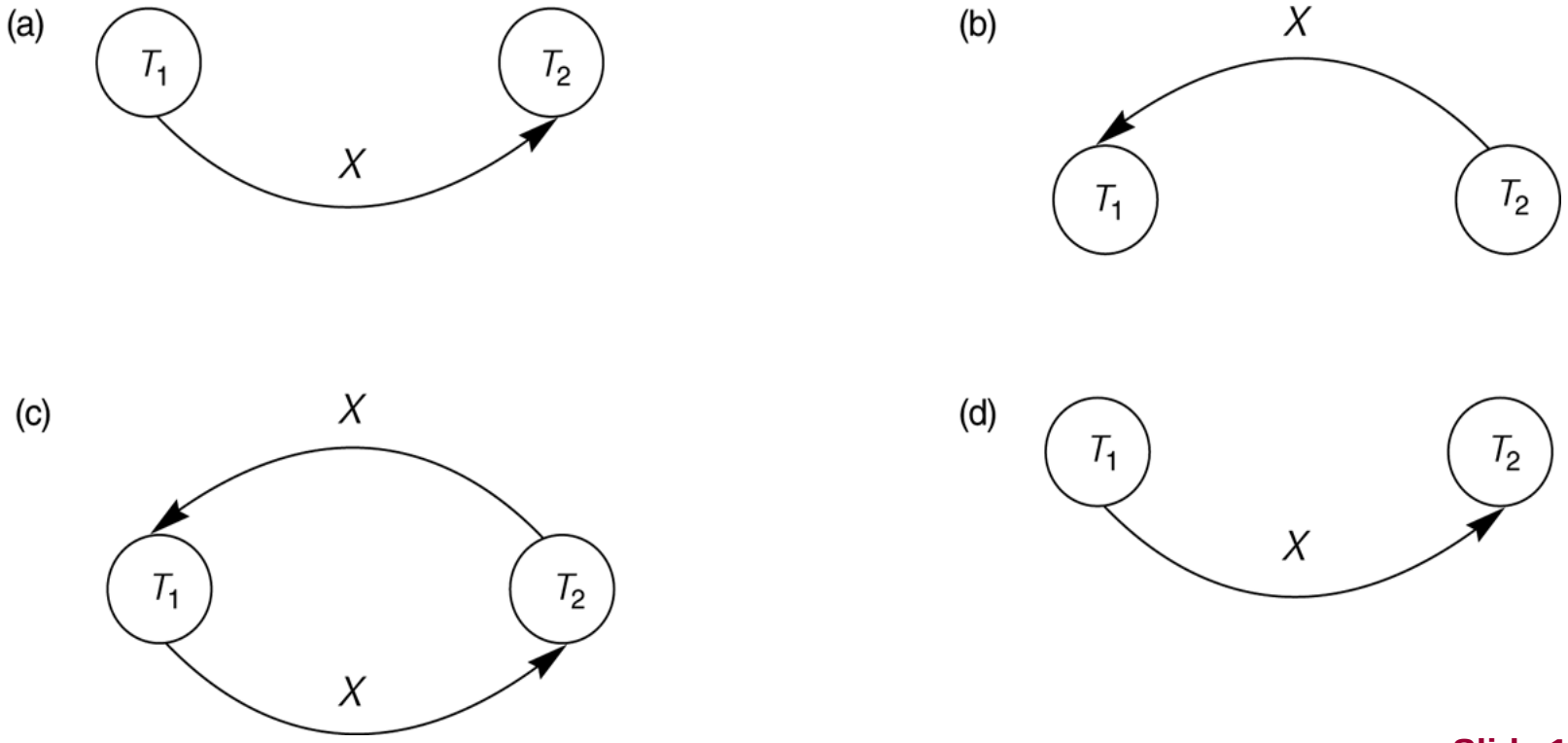
Find out conflict serializability: Algorithm

Testing for conflict serializability: Algorithm 17.1:

- Looks at only read_Item (X) and write_Item (X) operations
- Constructs a precedence graph (serialization graph) - a graph with directed edges
- An edge is created from Ti to Tj if one of the operations in Ti appears before a conflicting operation in Tj
- The schedule is serializable if and only if the precedence graph has no cycles.

Constructing the Precedence Graphs

- FIGURE 17.7 Constructing the precedence graphs for schedules A and D from Figure 17.5 to test for conflict serializability.
  - (a) Precedence graph for serial schedule A.
  - (b) Precedence graph for serial schedule B.
  - (c) Precedence graph for schedule C (not serializable).
  - (d) Precedence graph for schedule D (serializable, equivalent to schedule A).



Conflict Equivalent :

- **Conflict Equivalent:** Two schedules are said to be conflict equivalent when one can be transformed to another by swapping non-conflicting operations.

view Serializability

- A schedule will view serializable if it is view equivalent to a serial schedule.

View Equivalent

- Consider two schedules S1 and S2 each consisting of two transactions T1 and T2.
- Schedules S1 and S2 are called view equivalent if the following three conditions hold true for them-

- **Condition-01:**

For each data item X, if transaction T_i reads X from the database initially in schedule S1, then in schedule S2 also, T_i must perform the initial read of X from the database.

- **Condition-02:**

If transaction T_i reads a data item that has been updated by the transaction T_j in schedule S1, then in schedule S2 also, transaction T_i must read the same data item that has been updated by the transaction T_j.

Testing view Serializability algorithm

- **Condition-03:**

For each data item X, if X has been updated at last by transaction T_i in schedule S1, then in schedule S2 also, X must be updated at last by transaction T_i.

Recoverable Schedules

- **Recoverable schedule** — if a transaction T_j reads a data item previously written by a transaction T_i, then the *commit* operation of T_i appears before the *commit* operation of T_j.
- The following schedule (Schedule 9) is not recoverable if T_9 commits immediately after the read

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	