Chapter Two

Transaction

What is a Transaction?

- **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 an application program.
- **Transaction boundaries**:
  - Any single transaction in an application program is bounded with Begin and End statements.
- An **application program** may contain several transactions separated by the Begin and End transaction boundaries

Transaction operations

- the basic database access operations that a transaction can include are as follows:

■ read_item(*X*)*.*

✓ Reads a database item named X into a program variable

■ write_item(*X*)*.*

✓ Writes the value of program variable X into the database item named *X.*

- The basic access is one disk block from disk to memory

read_item(*X)*

- Executing a read_item(*X) command includes* the following steps:

1. Find the address of the disk block that contains x
2. Copy the disk block to memory buffer(if not in memory)
3. Copy the item from the buffer to program variable x

write_item(*X)*

- Executing a write_item(*X) command includes the following steps:*

1. Find the address of the disk block that contains x
2. Copy the disk block to memory buffer (if not in memory)
3. Copy the program variable x into buffer
4. Store the updated buffer to disk

A Transaction: An Informal Example

- E.g. Transfer transaction to transfer 500 birr from account A to account B: For a user it is one activity
- To database:
  - Read balance of A : read(A)
  - Read balance of B : read (B)
  - Subtract 500 birr from A
  - Add 500 birr to B
  - Write new value of A back to disk
  - Write new value of B back to disk

Cont'd

- Transaction to transfer 500 birr from account A to account B:
  1. **read**(*A*)
  2. *A* := *A* – 500
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 500
  6. **write**(*B)*

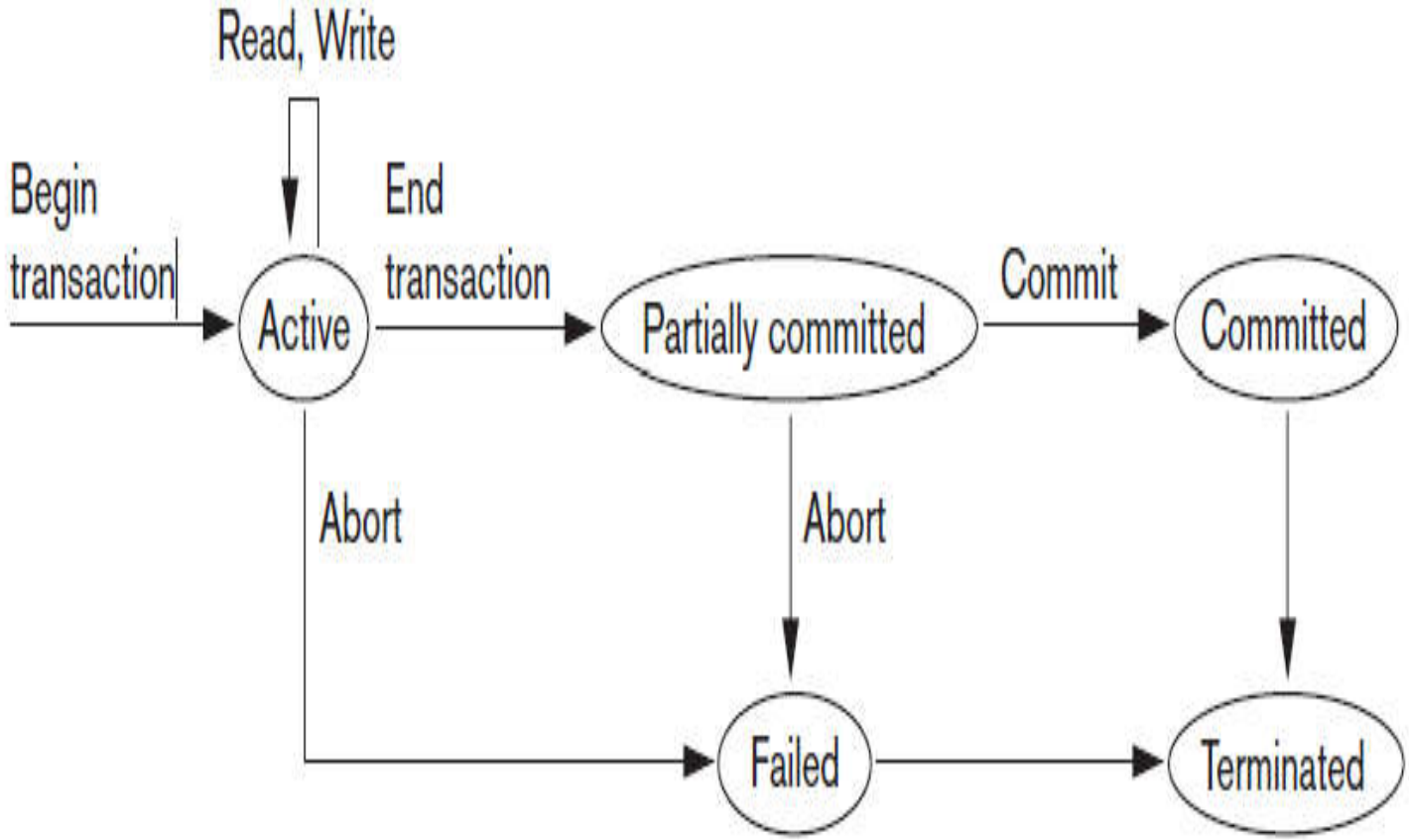
Transaction states and additional operations

- For recovery purposes, it keeps track of start, terminate and commits or aborts
- the recovery manager of the DBMS needs to keep track of the following operations :
  - ✓BEGIN_TRANSACTION
  - ✓ READ OR WRITE
  - ✓ END_TXN
  - ✓ COMMIT_TXN
  - ✓ ROLLBACK or ABORT
- Failed state need rollback operation

Transaction States

- transaction must be in one of several states:
  - **Active :** A transaction that has executed at least one action, but has not completed.
  - **Partially committed :** A transaction that has completed all actions, but has not been confirmed as completed.
  - **Failed :** A transaction that fails completion checks, or is aborted during the active state.
  - **Committed :** A completed and checked transaction
  - **Aborted :** A transaction that has failed and has had no effect on the database system

Cont'd


Read, Write
Begin transaction
Active
End transaction
Partially committed
Commit
Committed
Abort
Abort
Failed
Terminated

Desirable Properties of Transactions

- Transactions should possess several properties, often called the **ACID properties;**
- The following are the ACID properties
  - ✓**Atomicity**
  - ✓**Consistency preservation**
  - ✓**Isolation**
  - ✓**Durability or permanency**

Cont’d

- **Atomicity**

✓ A transaction is an atomic unit of processing;

✓ it should either be performed in its entirety or not performed at all

- **Consistency preservation.**

✓ A transaction should be consistency preserving,

✓ it is completely executed from beginning to end without interference from other transactions,

Cont’d

- **Isolation**

✓ A transaction should appear as though it is being executed in isolation from other transactions,

✓ even though many transactions are executing concurrently

- **Durability or permanency.**

✓ The changes applied to the database by a committed transaction must persist in the database.

✓ These changes must not be lost because of any failure.

Schedules

- Multiple transactions can be executed concurrently by interleaving their operations
- **Schedule S** – a sequences of instructions that specify the **chronological order** in which instructions of concurrent transactions T_1, T_2, … , T_n are executed
  - A schedule for a set of transactions **must consist** of all instructions of those transactions.
  - Must **preserve** the order in which the instructions appear in each individual transaction.

Cont'd

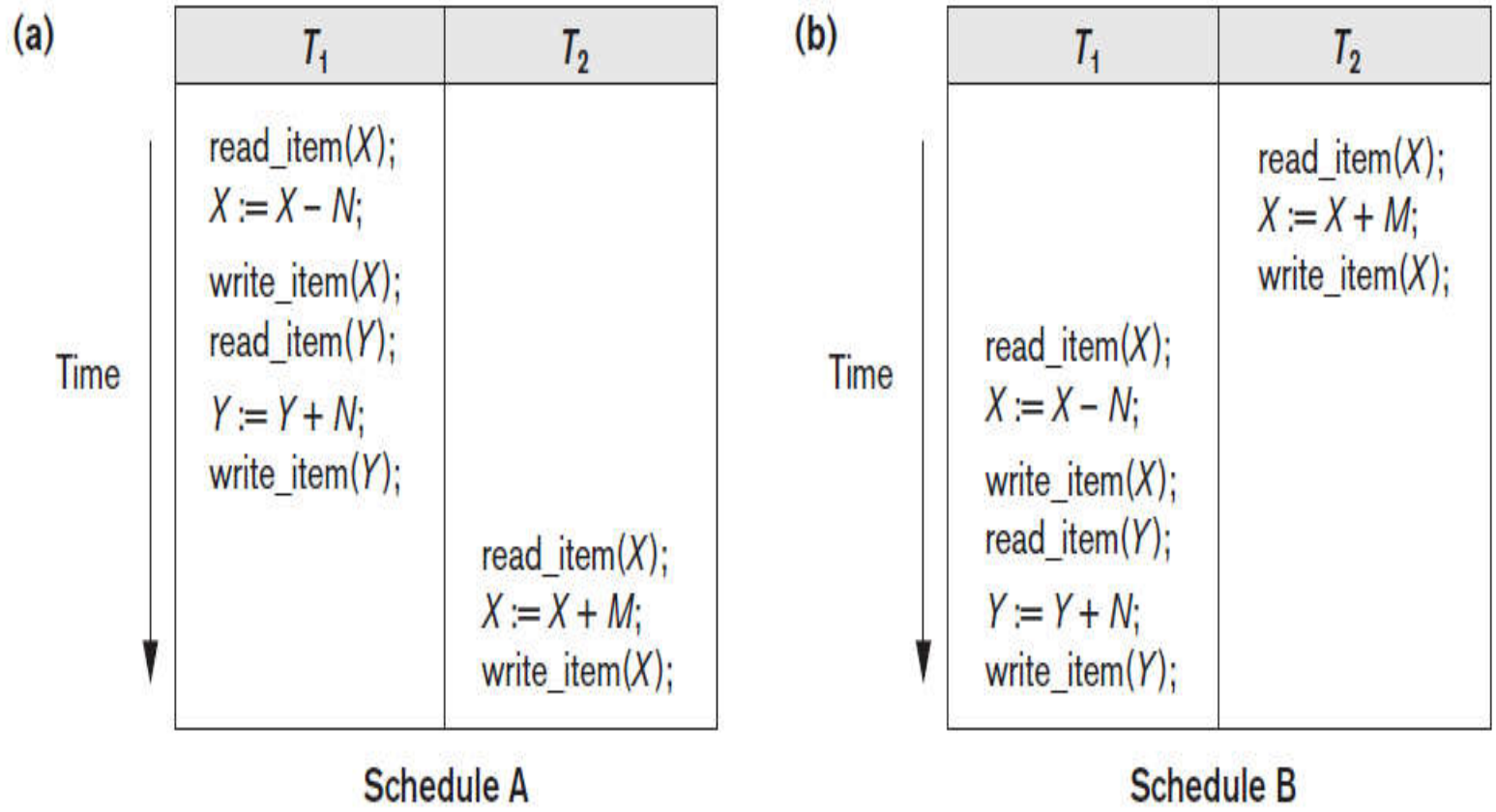
Serial schedule

- ✓ S is a schedule that executes all of the operations from one transaction T1, before moving on to the operations of another transaction T2.

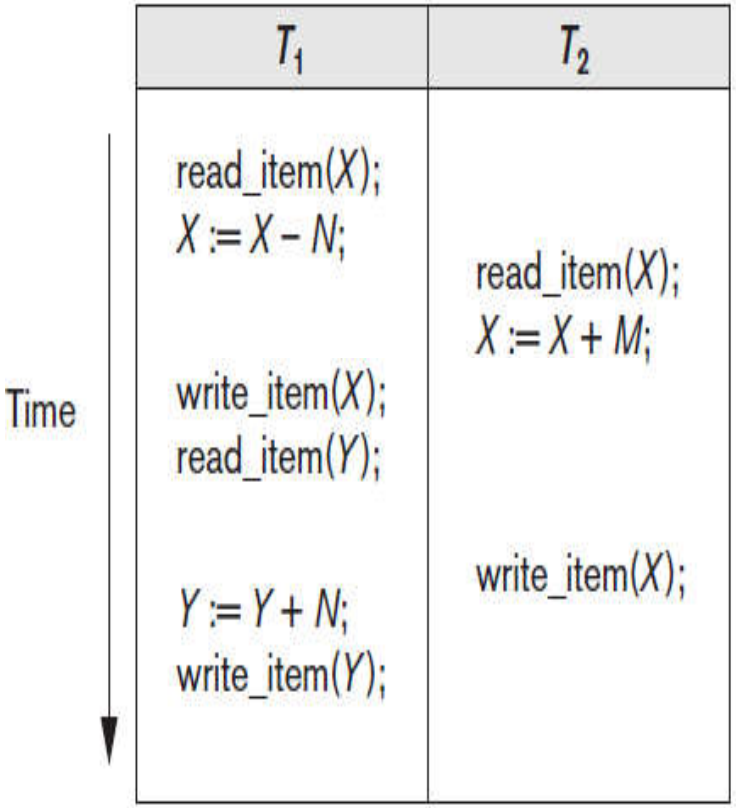
Non-serial schedule(*interleaved* schedule)

- ✓ An is a schedule in which the operations of an individual transaction are executed in order with respect to the same transaction,
- ✓ This is a type of Scheduling where the operations of multiple transactions are interleaved.
- ✓ This might lead to a rise in the concurrency problem

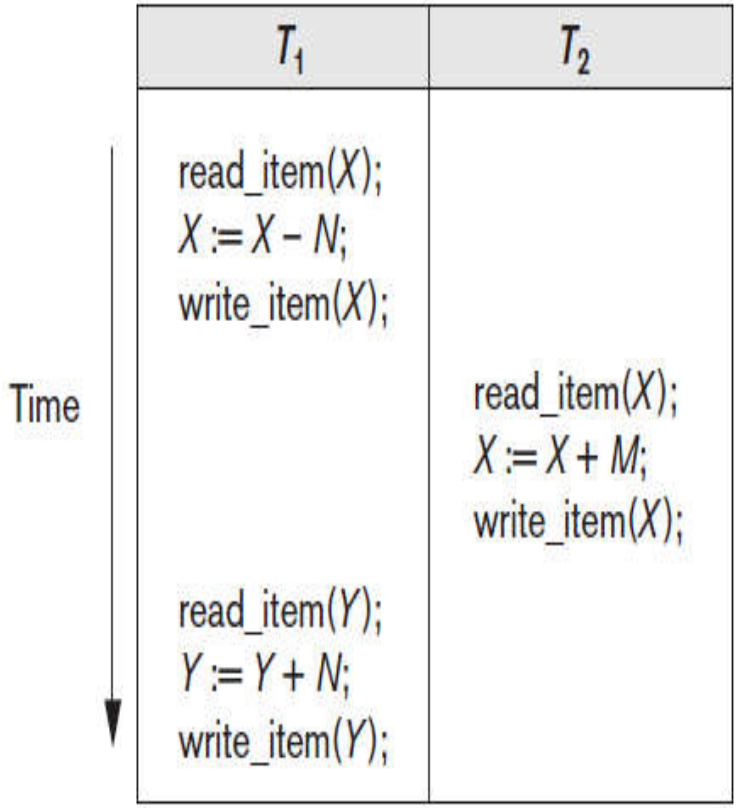
Serial schedule



Non-serial schedules



Schedule C



Schedule D

Result Equivalent Schedules

- Two schedules are result equivalent if they produce the same final state of the database
- Problem: May produce same result by accident!

S1	S2
read_item(X);	read_item(X);
X:=X+10;	X:=X*1.1;
write_item(X);	write_item(X);

Schedules S1 and S2 are result equivalent for X=100 but not in general

Characterizing Schedules based on Serializability

- The concept of Serializable of schedule is used to identify which schedules are correct when concurrent transactions executions have interleaving of their operations in the schedule
- Serial schedule:
  - A schedule S is serial if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule.
    - Otherwise, the schedule is called nonserial schedule.
- Serializable schedule:
  - a schedule whose effect on any consistent database instance is identical to that of some complete serial schedule over the set of *committed* transactions in *S*.
  - A nonserial schedule S is serializable is equivalent to say that it is correct to the result of one of the serial schedule

Serializability of Schedules

Serializability

- A schedule *S* of *n* transactions is **serializable** if it is *equivalent to some serial schedule* of the same *n* transactions
- **Kinds** of **serializability**
  - Conflict-serializable schedule
  - View-serializable schedule

Result Equivalent Schedules

- Two schedules are result equivalent if they produce the same final state of the database
- Problem: May produce same result by accident!

S1	S2
read_item(X);	read_item(X);
X:=X+10;	X:=X*1.1;
write_item(X);	write_item(X);

Schedules S1 and S2 are result equivalent for X=100 but not in general

Two types of equivalent schedule:

- Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules.
- Eg
  - S1: r1(x); w2(x) & S2: w2(x); r1(x)
  - S1: w1(x); w2(x); & S2: w2(x); w1(x);

Not conflict equivalent

- Conflict serializable:
  - A schedule S is said to be conflict serializable if it is conflict equivalent to some serial schedule S'.
  - Every conflict serializable schedule is serializable .

Conflict Serializable

- Schedule S is conflict serializable if it is conflict equivalent to some *serial* schedule S'
  - We can reorder the *non-conflicting* operations to improve efficiency
- Non-conflicting operations:
  - Reads and writes from same transaction
  - Reads from different transactions
  - Reads and writes from different transactions on different data items
- Conflicting operations:
  - Reads and writes from different transactions on same data item

- Two operations in a schedule are side to be conflict if they satisfy all the three of the following conditions.
  - They belongs to different transaction
  - They access the same data item X
  - At least one of the operation is a write_Item(X)

Eg. Sa: r1(X); r2(x); w1(X); r1(Y); W2(X); W1(Y);

- r1(X) and w2(X)
- r2(X) and w1(X);
- W1(X) and w2(X)

} Conflict operations

- r1(X) and r2(X)
- W2(X) and w1(Y)

} No Conflict

Cont'd

Testing for conflict serializability: Algorithm

- Looks at only read_Item (X) & 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.

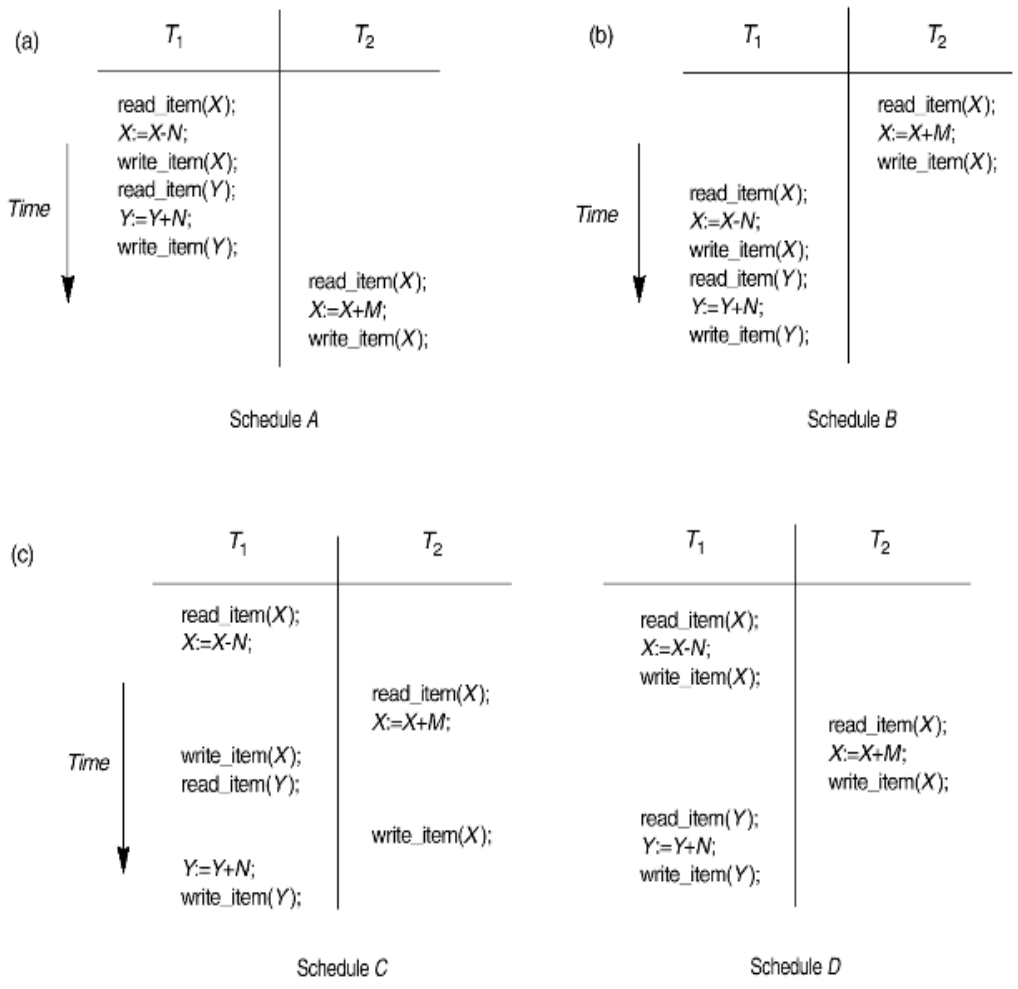
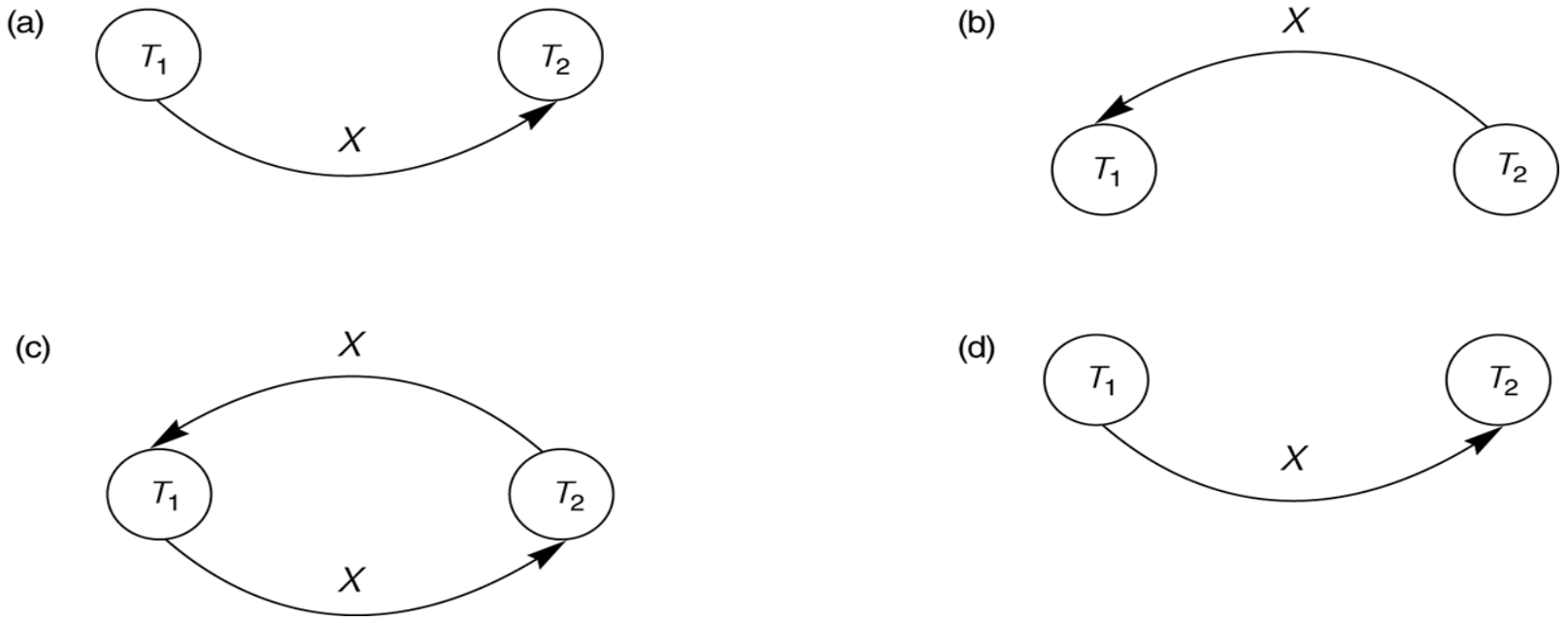


Fig 1: Examples of serial and nonserial schedules involving transactions T_1 and T_2.

Constructing the Precedence Graphs

- FIGURE 2: Constructing the precedence graphs for schedules A and D from Figure 1 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).



View Equivalence and View Serializability

- **View Serializability** is a process to find out that a given **schedule** is view serializable or not.
- To check whether a given schedule is view serializable,
- we need to check whether the given schedule is **View Equivalent** to its serial schedule.
  - **Definition of serializability based on view equivalence.**

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

view equivalence:

- As long as each read operation of a transaction reads the result of *the same write operation* in both schedules,
- the write operations of each transaction must produce the same results.
- **“The view”**: the read operations are said to see the *same view* in both schedules.

- Two schedules are said to be view equivalent if the following three conditions hold:
  1. The same set of transactions participates in S and S', and S and S' include the same operations of those transactions.
  2. If Ti reads a value **A** written by Tj in S1 , it must also read the value of **A** written by Tj in S2
  3. for each data object A, the transaction that perform the final write on x in S1 must also perform the final write on **A** in S2

S'

T1: R(A)	W(A)	
T2:	W(A)	
T3:		W(A)

view

S

T1: R(A),W(A)		
T2:	W(A)	
T3:		W(A)

- **Relationship between view and conflict equivalence**:
  - The two are same under **constrained write assumption** which assumes that if T writes X, it is constrained by the value of X it read;

    i.e., new X = f(old X)
  - Conflict serializability is **stricter** than view serializability.
  - a schedule that is view serializable is not necessarily conflict serializable.
  - Any conflict serializable schedule is also view serializable, but not vice versa.

Reading Assignment

Strict schedule