# CHAPTER 13

# Introduction to Transaction Processing Concepts and Theory

# Introduction

- Transaction
  - Describes local unit of database processing
- Transaction processing systems
  - Systems with large databases and hundreds of concurrent users
  - Require high availability and fast response time

# 13.1 Introduction to Transaction Processing

- Single-user DBMS
    - At most one user at a time can use the system
    - Example: home computer
- Multiuser DBMS
    - Many users can access the system (database) concurrently
    - Example: airline reservations system

# Introduction to Transaction Processing (cont'd.)

- Multiprogramming
  - Allows operating system to execute multiple processes concurrently
  - Executes commands from one process, then suspends that process and executes commands from another process, etc.

# Introduction to Transaction Processing (cont'd.)

▶ Interleaved processing

▶ Parallel processing

   ▶ Processes C and D in figure below

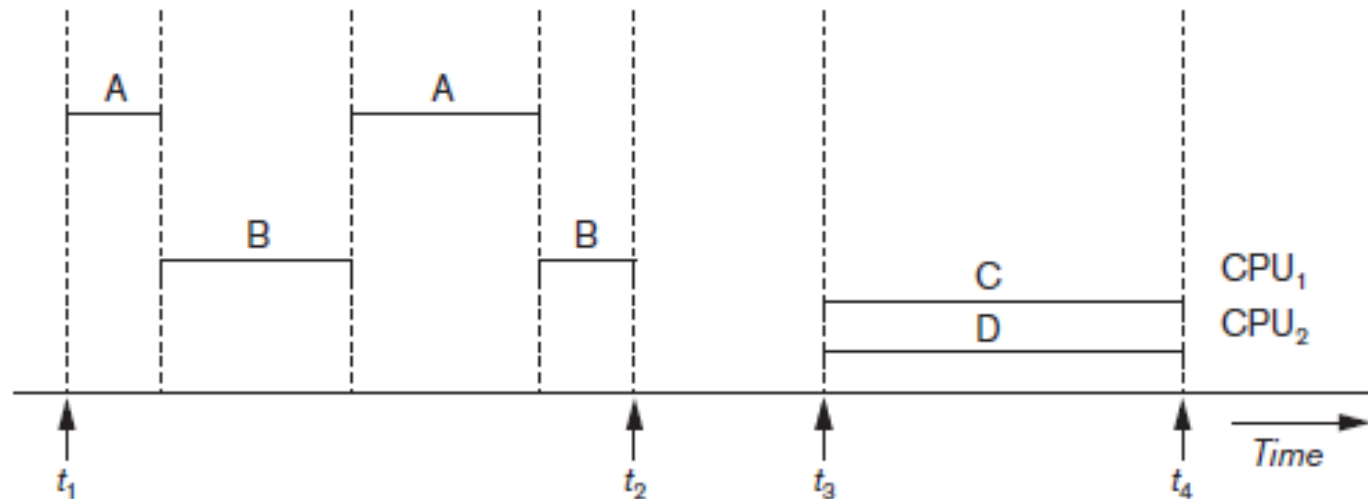

Figure 13.1 Interleaved processing versus parallel processing of concurrent transactions

# Transactions

- Transaction: an executing program
  - Forms logical unit of database processing
- Begin and end transaction statements
  - Specify transaction boundaries
- Read-only transaction
- Read-write transaction

# Database Items

- Database represented as collection of named data items

- Size of a data item called its granularity

- Data item

  - Record

  - Disk block

  - Attribute value of a record

# Read and Write Operations

- read_item(X)

  - Reads a database item named X into a program variable named X

  - Process includes finding the address of the disk block, and copying to and from a memory buffer

- write_item(X)

  - Writes the value of program variable X into the database item named X

  - Process includes finding the address of the disk block, copying to and from a memory buffer, and storing the updated disk block back to disk

# Read and Write Operations (cont'd.)

▶ Read set of a transaction

  ▶ Set of all items read
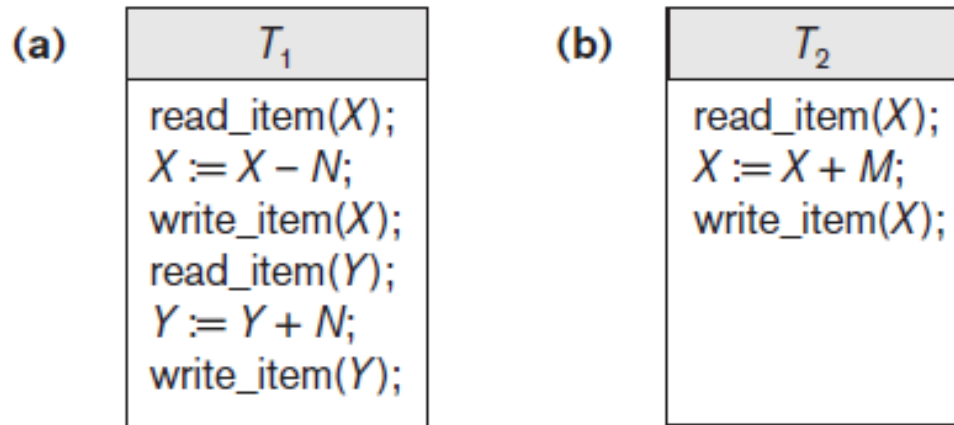
▶ Write set of a transaction

  ▶ Set of all items written

| (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($X$); |

Figure 13.2 Two sample transactions (a) Transaction $T1$ (b) Transaction $T2$

# DBMS Buffers

▶ DBMS will maintain several main memory data buffers in the database cache

▶ When buffers are occupied, a buffer replacement policy is used to choose which buffer will be replaced

  ▶ Example policy: least recently used

# Concurrency Control

▶ Transactions submitted by various users may execute concurrently

  ▶ Access and update the same database items

  ▶ Some form of concurrency control is needed

▶ The lost update problem

  ▶ Occurs when two transactions that access the same database items have operations interleaved

  ▶ Results in incorrect value of some database items
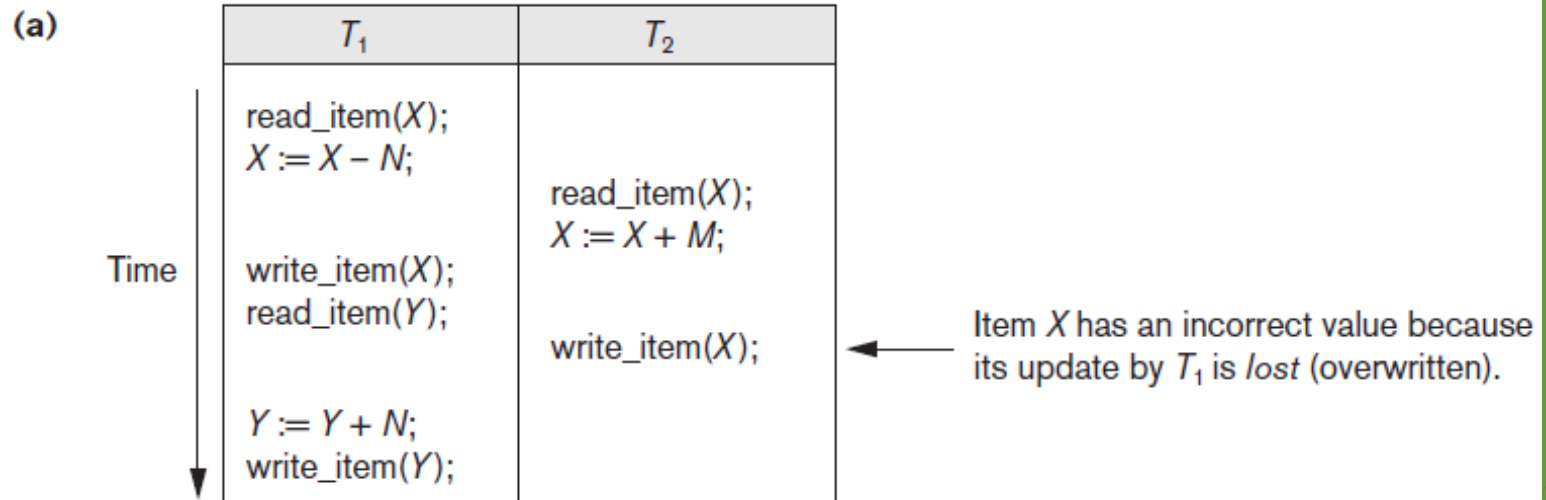
# The Lost Update Problem



**(a)**

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

Time →

Item $X$ has an incorrect value because its update by $T_1$ is *lost* (overwritten).

Figure 13.3 Some problems that occur when concurrent execution is uncontrolled (a) The lost update problem

# The Temporary Update Problem



**(b)**

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

Time ↓

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 $X$.
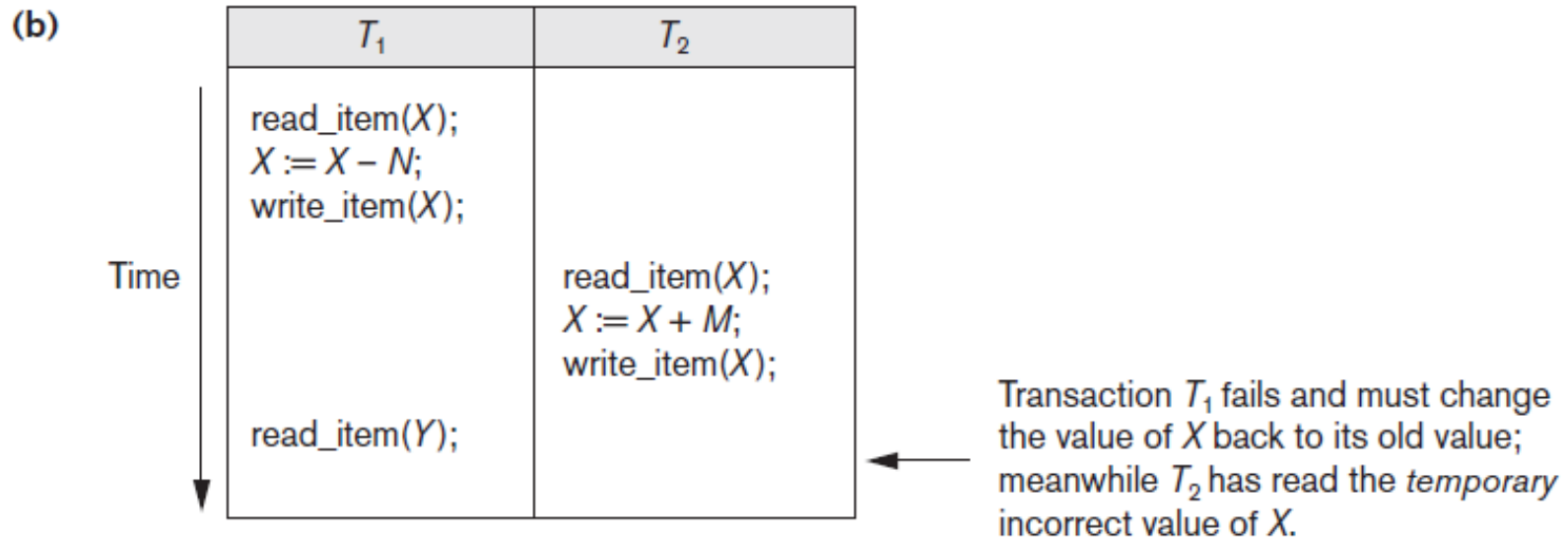
Figure 13.3 (cont'd.) Some problems that occur when concurrent execution is uncontrolled (b) The temporary update problem

# The Unrepeatable Read Problem

▶ Transaction T reads the same item twice

▶ Value is changed by another transaction T' between the two reads

▶ T receives different values for the two reads of the same item

# Why Recovery is Needed

- Committed transaction

  - Effect recorded permanently in the database

- Aborted transaction

  - Does not affect the database

- Types of transaction failures

  - Computer failure (system crash)

  - Transaction or system error

  - Local errors or exception conditions detected by the transaction

# Why Recovery is Needed (cont'd.)

- Types of transaction failures (cont'd.)
  - Concurrency control enforcement
  - Disk failure
  - Physical problems or catastrophes
- System must keep sufficient information to recover quickly from the failure
  - Disk failure or other catastrophes have long recovery times

# 13.2 Transaction and System Concepts

▶ System must keep track of when each transaction starts, terminates, commits, and/or aborts

- ▶ BEGIN_TRANSACTION

- ▶ READ or WRITE

- ▶ END_TRANSACTION

- ▶ COMMIT_TRANSACTION

- ▶ ROLLBACK (or ABORT)

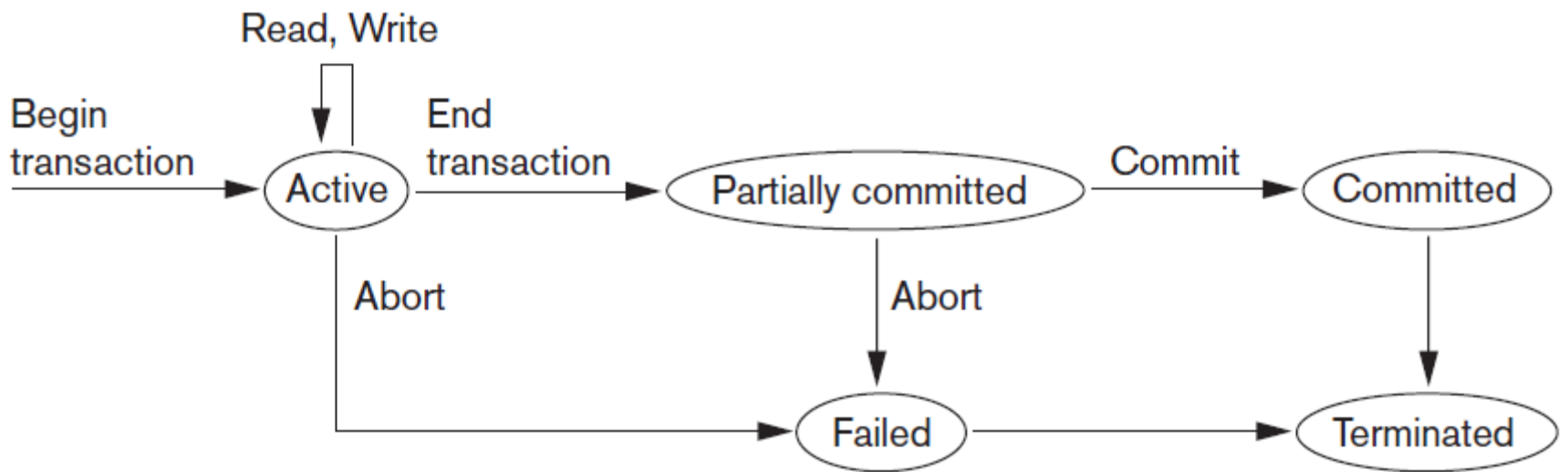# Transaction and System Concepts (cont'd.)



Figure 13.4 State transition diagram illustrating the states for transaction execution

# The System Log

- ▶ System log keeps track of transaction operations

- ▶ Sequential, append-only file

- ▶ Not affected by failure (except disk or catastrophic failure)

- ▶ Log buffer

    - ▶ Main memory buffer

    - ▶ When full, appended to end of log file on disk

- ▶ Log file is backed up periodically

- ▶ Undo and redo operations based on log possible

# Commit Point of a Transaction

- Occurs when all operations that access the database have completed successfully

  - And effect of operations recorded in the log

- Transaction writes a commit record into the log

  - If system failure occurs, can search for transactions with recorded start_transaction but no commit record

- Force-writing the log buffer to disk

  - Writing log buffer to disk before transaction reaches commit point

# 13.3 Desirable Properties of Transactions

- ACID properties
  - Atomicity
    - Transaction performed in its entirety or not at all
  - Consistency preservation
    - Takes database from one consistent state to another
  - Isolation
    - Not interfered with by other transactions
  - Durability or permanency
    - Changes must persist in the database

# Desirable Properties of Transactions (cont'd.)

- Levels of isolation

  - Level 0 isolation does not overwrite the dirty reads of higher-level transactions

  - Level 1 isolation has no lost updates

  - Level 2 isolation has no lost updates and no dirty reads

  - Level 3 (true) isolation has repeatable reads

    - In addition to level 2 properties

  - Snapshot isolation

- Snapshot isolation

  - Used in some commercial DBMSs

  - Transaction sees data items that it reads based on the committed values of the items in the database snapshot when transaction starts

  - Ensures phantom record problem will not occur

# 13.4 Characterizing Schedules Based on Recoverability

▶ Schedule or history

   ▶ Order of execution of operations from all transactions

   ▶ Operations from different transactions can be interleaved in the schedule

▶ Total ordering of operations in a schedule

   ▶ For any two operations in the schedule, one must occur before the other

# Characterizing Schedules Based on Recoverability (cont'd.)

▶ Two conflicting operations in a schedule

　▶ Operations belong to different transactions

　▶ Operations access the same item X

　▶ At least one of the operations is a write_item(X)

▶ Two operations conflict if changing their order results in a different outcome

▶ Read-write conflict

▶ Write-write conflict

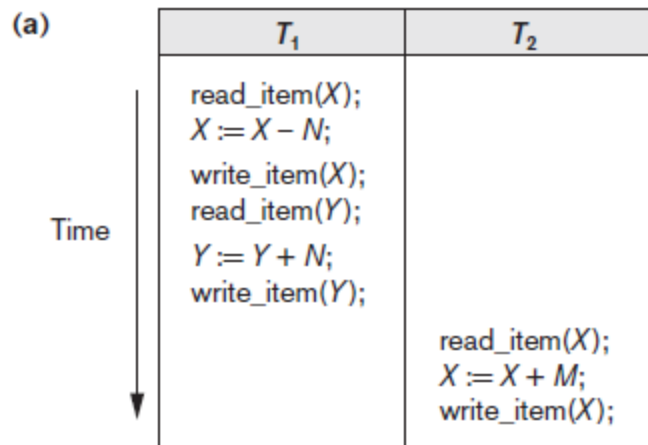# Characterizing Schedules Based on Recoverability (cont'd.)

- ▶ Recoverable schedules
    - ▶ Recovery is possible
- ▶ Nonrecoverable schedules should not be permitted by the DBMS
- ▶ No committed transaction ever needs to be rolled back
- ▶ Cascading rollback may occur in some recoverable schedules
    - ▶ Uncommitted transaction may need to be rolled back

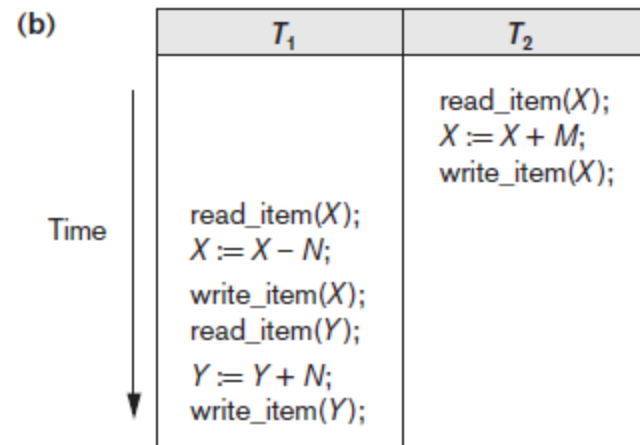# Characterizing Schedules Based on Recoverability (cont'd.)

▶ Cascadeless schedule

    ▶ Avoids cascading rollback

▶ Strict schedule

    ▶ Transactions can neither read nor write an item X until the last transaction that wrote X has committed or aborted

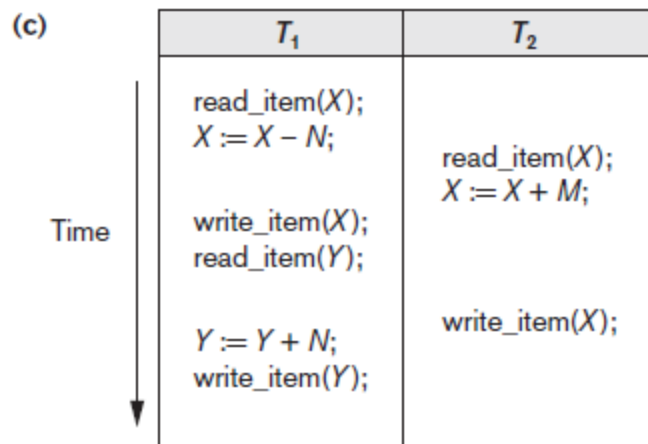# 13.5 Characterizing Schedules Based on Serializability

▶ Serializable schedules

  ▶ Always considered to be correct when concurrent transactions are executing

  ▶ Places simultaneous transactions in series

    ▶ Transaction $T_1$ before $T_2$, or vice versa
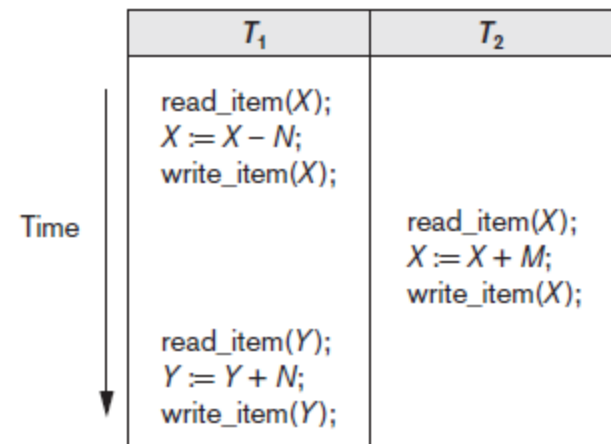
| (a) | $T_1$ | $T_2$ |
|-----|-------|-------|
| Time | read_item($X$);<br>$X := X - N$;<br>write_item($X$);<br>read_item($Y$);<br>$Y := Y + N$;<br>write_item($Y$); | read_item($X$);<br>$X := X + M$;<br>write_item($X$); |

Schedule A

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

Schedule B

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

Schedule C

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

Schedule D

Figure 13.5 Examples of serial and nonserial schedules involving transactions $T1$ and $T2$ (a) Serial schedule A: $T1$ followed by $T2$ (b) Serial schedule B: $T2$ followed by $T1$ (c) Two nonserial schedules C and D with interleaving of operations

# Characterizing Schedules Based on Serializability (cont'd.)

▶ Problem with serial schedules

　　▶ Limit concurrency by prohibiting interleaving of operations

　　▶ Unacceptable in practice

　　▶ Solution: determine which schedules are equivalent to a serial schedule and allow those to occur

# Characterizing Schedules Based on Serializability (cont'd.)

▶ Result equivalent schedules

  ▶ Produce the same final state of the database

    ▶ May be accidental

  ▶ Cannot be used alone to define equivalence of schedules

| $S_1$ |
| --- |
| read_item($X$); |
| $X := X + 10$; |
| write_item($X$); |

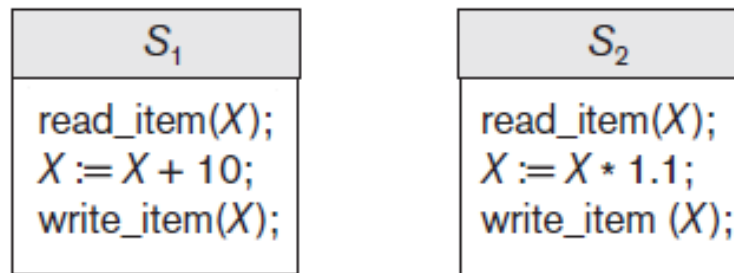| $S_2$ |
| --- |
| read_item($X$); |
| $X := X * 1.1$; |
| write_item ($X$); |

Figure 13.6 Two schedules that are result equivalent for the initial value of $X = 100$ but are not result equivalent in general

# Characterizing Schedules Based on Serializability (cont'd.)

► Conflict equivalence

  ► Relative order of any two conflicting operations is the same in both schedules

► Serializable schedules

  ► Schedule S is serializable if it is conflict equivalent to some serial schedule S'.

# Characterizing Schedules Based on Serializability (cont'd.)

▶ Testing for serializability of a schedule

1. For each transaction $T_i$ participating in schedule $S$, create a node labeled $T_i$ in the precedence graph.

2. For each case in $S$ where $T_j$ executes a read_item($X$) after $T_i$ executes a write_item($X$), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.

3. For each case in $S$ where $T_j$ executes a write_item($X$) after $T_i$ executes a read_item($X$), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.

4. For each case in $S$ where $T_j$ executes a write_item($X$) after $T_i$ executes a write_item($X$), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.

5. The schedule $S$ is serializable if and only if the precedence graph has no cycles.

Algorithm 20.1 Testing conflict serializability of a schedule S

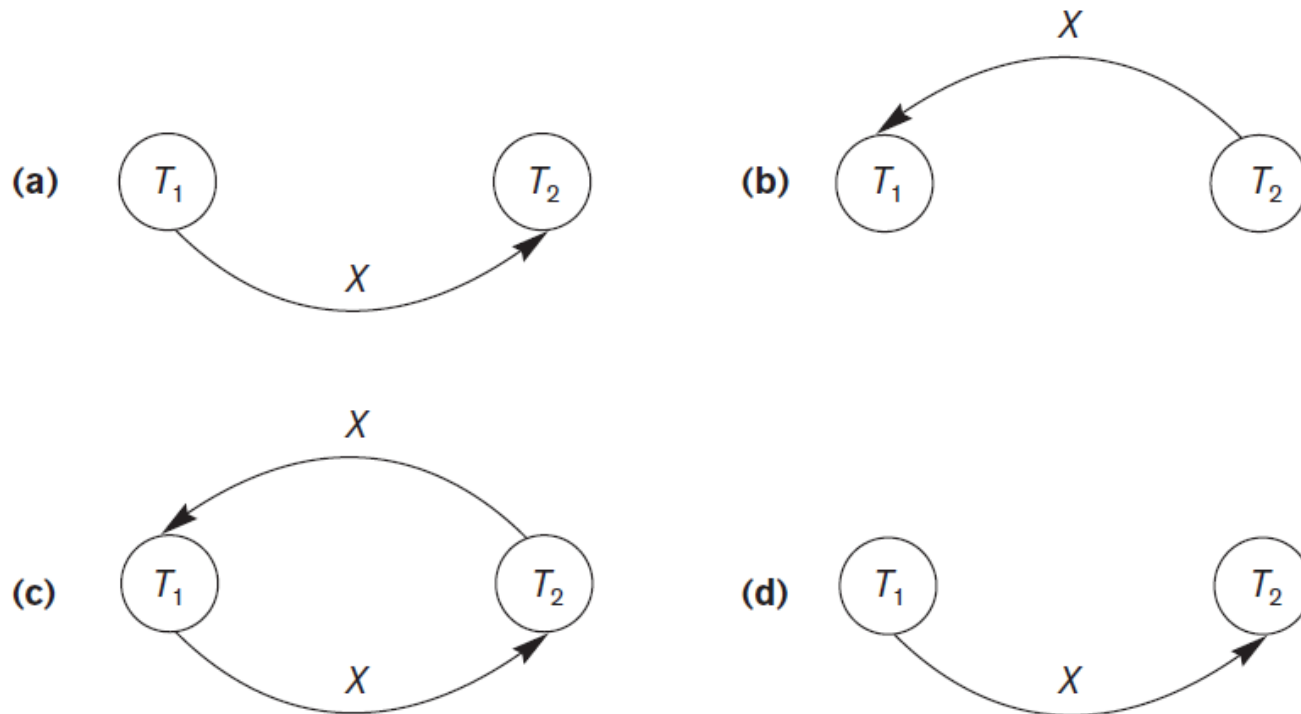# Characterizing Schedules Based on Serializability (cont'd.)



Figure 13.7 Constructing the precedence graphs for schedules A to D from Figure 20.5 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)

# How Serializability is Used for Concurrency Control

- Being serializable is different from being serial

- Serializable schedule gives benefit of concurrent execution
  - Without giving up any correctness

- Difficult to test for serializability in practice
  - Factors such as system load, time of transaction submission, and process priority affect ordering of operations

- DBMS enforces protocols
  - Set of rules to ensure serializability

# View Equivalence and View Serializability

▶ View equivalence of two schedules

  ▶ 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

  ▶ Read operations said to see the same view in both schedules

▶ View serializable schedule

  ▶ View equivalent to a serial schedule

# View Equivalence and View Serializability (cont'd.)

- ▶ Conflict serializability similar to view serializability if constrained write assumption (no blind writes) applies

- ▶ Unconstrained write assumption
  - ▶ Value written by an operation can be independent of its old value

# 13.6 Transaction Support in SQL

▶ No explicit Begin_Transaction statement

▶ Every transaction must have an explicit end statement

  ▶ COMMIT

  ▶ ROLLBACK

▶ Access mode is READ ONLY or READ WRITE

▶ Diagnostic area size option

  ▶ Integer value indicating number of conditions held simultaneously in the diagnostic area

# Transaction Support in SQL (cont'd.)

▶ Isolation level option

  ▶ Dirty read

  ▶ Nonrepeatable read

  ▶ Phantoms

| Isolation Level | Type of Violation | | |
|---|---|---|---|
| | Dirty Read | Nonrepeatable Read | Phantom |
| READ UNCOMMITTED | Yes | Yes | Yes |
| READ COMMITTED | No | Yes | Yes |
| REPEATABLE READ | No | No | Yes |
| SERIALIZABLE | No | No | No |

Table 13.1 Possible violations based on isolation levels as defined in SQL

# 13.7 Summary

- ▶ Single and multiuser database transactions
- ▶ Uncontrolled execution of concurrent transactions
- ▶ System log
- ▶ Failure recovery
- ▶ Committed transaction
- ▶ Schedule (history) defines execution sequence
  - ▶ Schedule recoverability
  - ▶ Schedule equivalence
- ▶ Serializability of schedules