Chapter 20

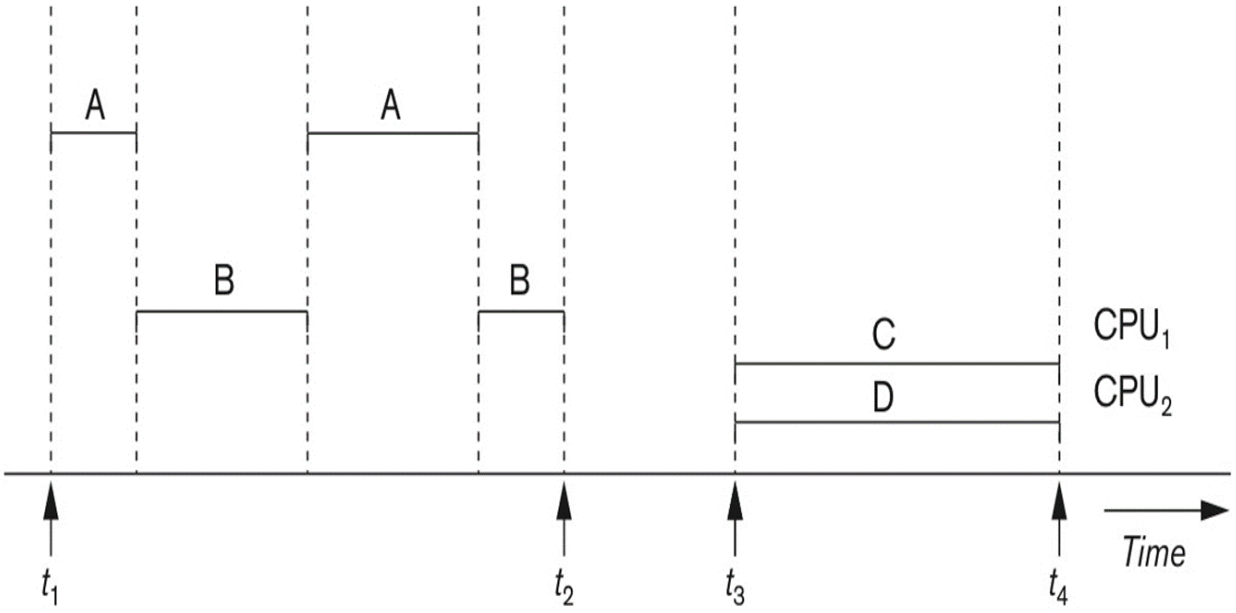
Introduction to Transaction Processing Concepts and Theory

- Logical units of DB processing
- Large database and hundreds of transactions
- Ex. Stock market, super market, banking, etc…
- High availability and fast response
- Hundreds of concurrent users
- Must be completed in its entirety to ensure correctness
- A transaction (Txn) includes database commands; retrievals, insertion, deletion and updates

1. Transaction Processing

- Number of users who can use the system concurrently
- Single user or multiuser
- Single user PC systems
- Most DBMSs are multiuser (airline reservation)
- Multiusers can use DBMS because of multiprogramming
- Multiple programs or processes running at the same time allowed by OS
- CPU can only execute at most one program at a time
- Multiprogramming: suspend one process and execute another
- Concurrent execution is interleaved (most of the DBMS theory is based on this)
  Fig. 20.1

Figure 20.1 Interleaved processing versus parallel processing of concurrent transactions.



Transactions, Database items, Read and Write operations and DBMS buffers

- A transaction is an executing program, forms a logical unit of database processing
- Txn includes one or more database operations
- Txn can be embedded in an application program (or it can be a command line query)
- Txn boundary; begin Txn …..end Txn
- A single application program can contain many Txns

- If a Txn is retrieve and no updates, it is called a read only Txn, otherwise read-write
- Data item can be a record or an entire block (granularity) or could be a single attribute
- Each data item has a unique name
- The basic database operation that a Txn can include: read_item(x), write_item(x); x is a program variable
- The basic access is one disk block from disk to memory

Read_item(x):

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):

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

- Underlying OS and recovery manager stores the buffer on the disk
- Database cache maintains many data buffers
- Each buffer is usually one block
- Buffer replacement policy is used to replace buffers (LRU), specific to DBMS
- Concurrency control and recovery mechanisms are concerned with database commands in the Txn
- Txns submitted by various users may execute concurrently and may access same items

Why concurrency control is needed?

- Several problems can occur when Txns run in an uncontrolled manner
- Ex. Airline reservation (each record includes the number of reserved seats in a flight)

1. The Lost Update
2. The temporary update (or dirty read)
3. The incorrect summary

The Lost Update: Fig 20.3.(a)

- Two Txns accessing the same database item
- The Txn operations are interleaved
- T1 write is lost

X=80

N=5 Read(X)

Write(75)

M=4

Write(80+4)

(5 seats transferred by T1) 4 seats reserved by T2

(Intended operation: 80-5+4=79 seats)

Figure 20.2 Two sample transactions. (a) Transaction **T1.** (b) Transaction **T2.**

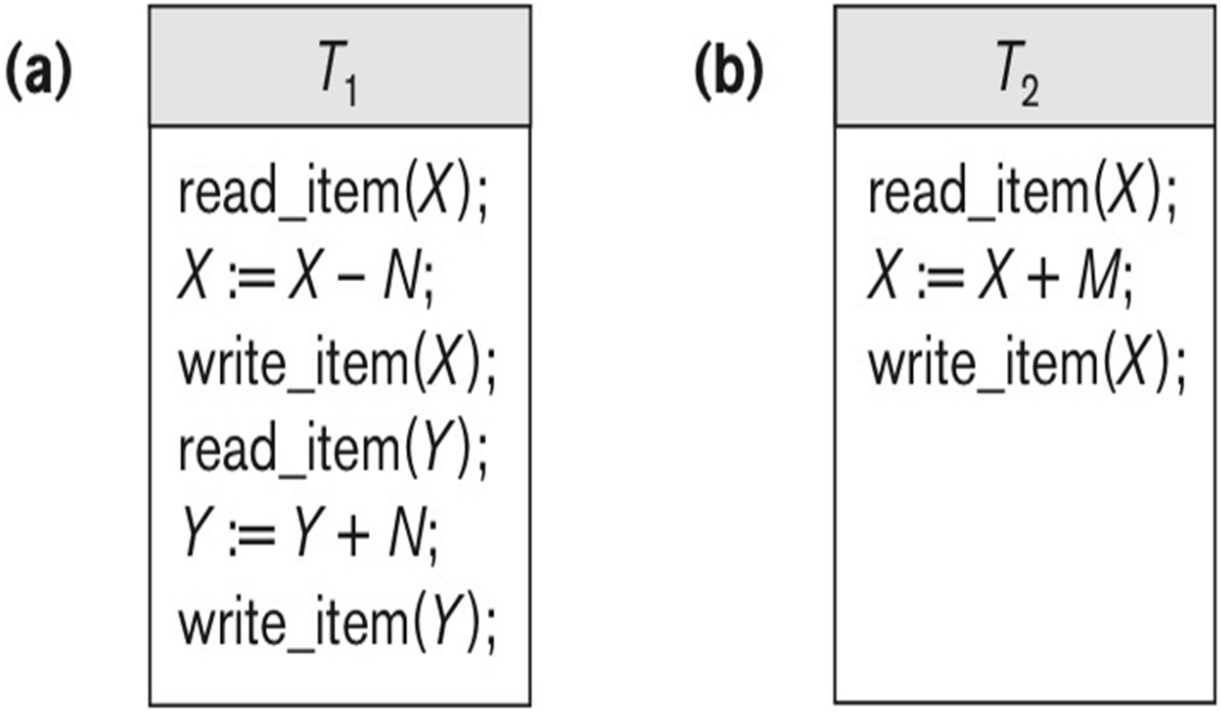


Figure 20.3a Some problems that occur when concurrent execution is uncontrolled. The lost update problem.

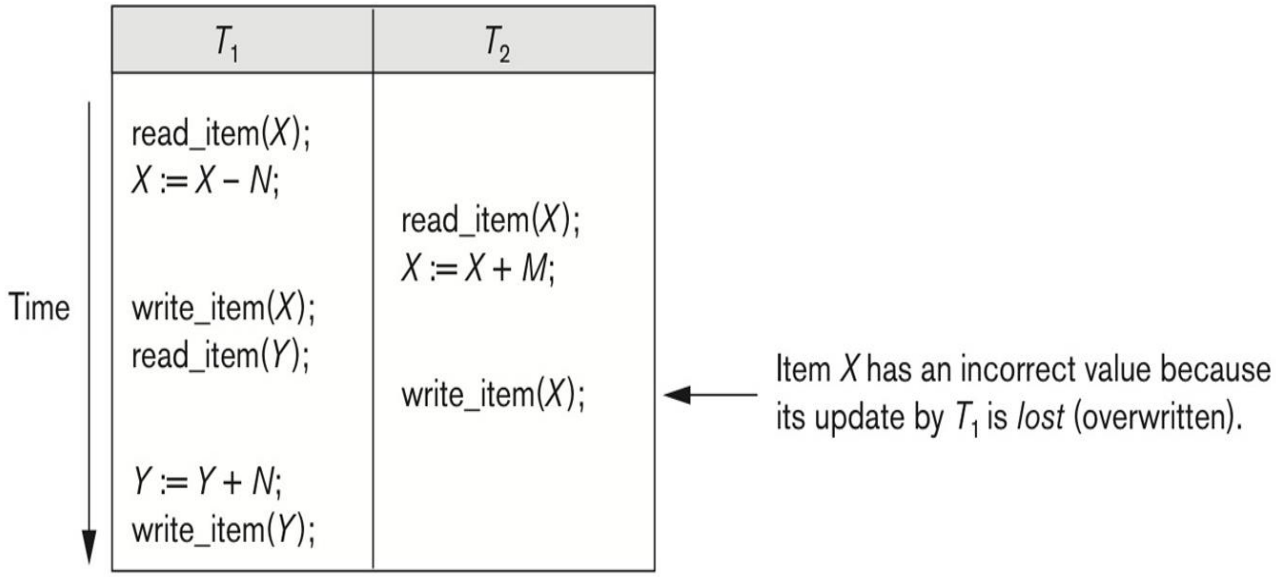
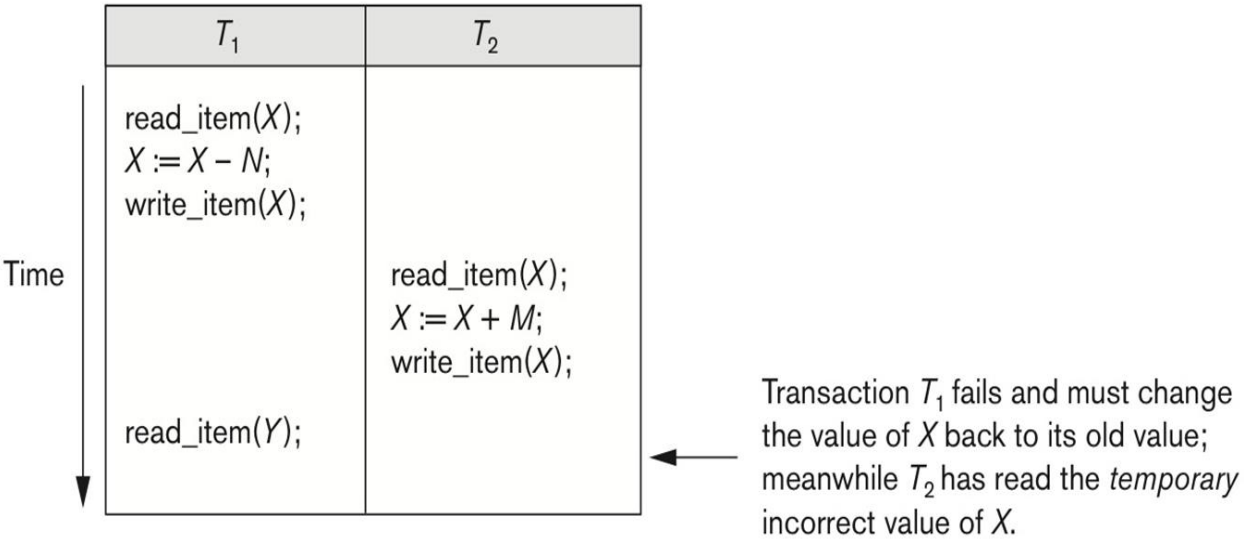


Figure 20.3b Some problems that occur when concurrent execution is uncontrolled. The temporary update problem.



The Dirty Read

Fig. 20.3(b)

- When a Txn updates a database item and fails
- Meanwhile, another Txn reads

T1 fails and it should change the value back to old X value (T1s write occurred but its Txn failed)

T2 read incorrect value as T1 was not complete

The Incorrect Summary

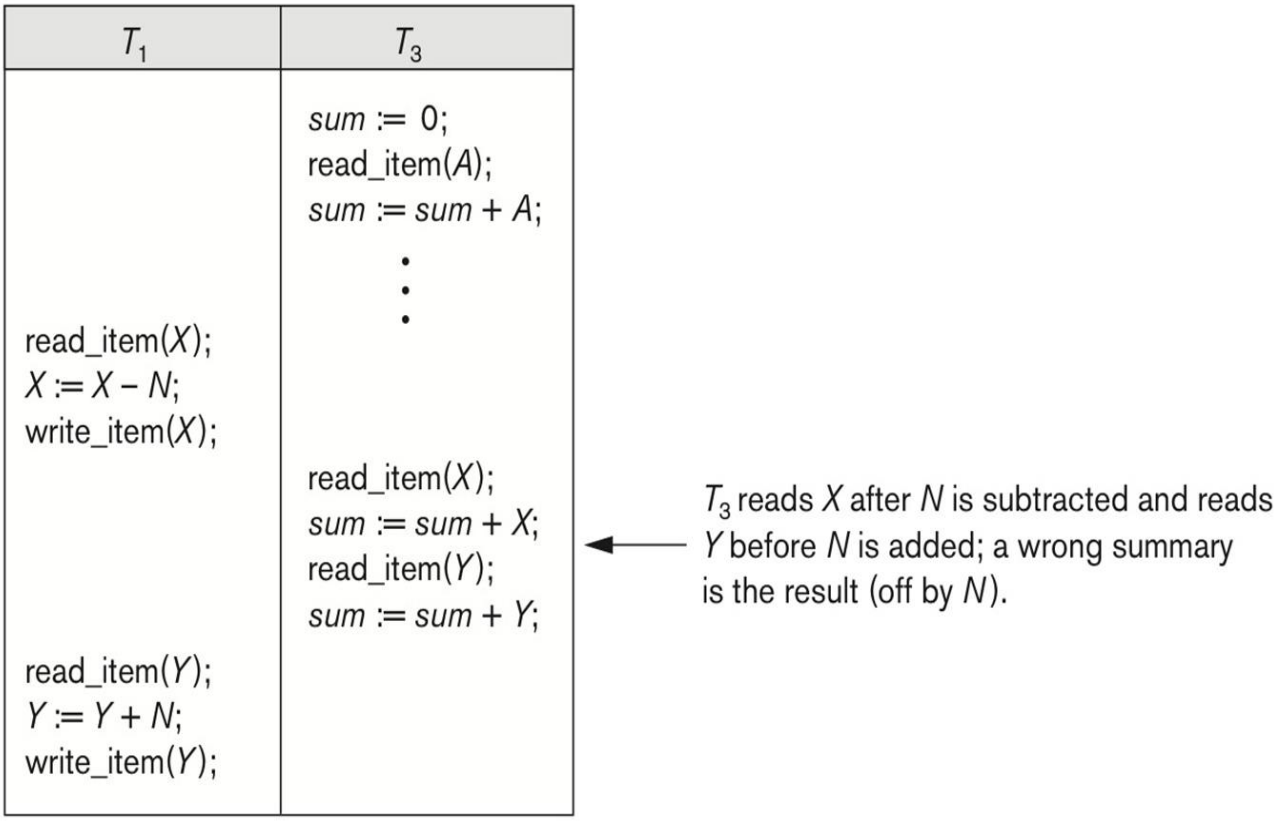
- One Txn is calculating an aggregate summary function on a number of database items, other Txn is updating the same items
- Aggregate function may use values before or after update
  Fig20-3 (c)

Unrepeatable Read Problem:

- A Txn reads an item twice and gets a different data
- Some other Txn modified the item

Ex. Airline reservation – checks a seat, not confirmed, later checks second time, it is gone

Figure 20.3c Some problems that occur when concurrent execution is uncontrolled. The incorrect summary problem.



Why recovery is needed?

- Many database operations run and after that a Txn is committed
- Or Txn may be aborted and has no effect on the database
- The DBMS must not allow some operations to be complete and other operations are not---Txn is a whole operation and it is a logical unit
- If a Txn fails after performing some operations, then those operations already executed must be undone and have no lasting effect

Types of failures

- System crash, media failure, memory failure
- A Txn or system error; integer overflow, division by 0, erroneous parameters, logical programming errors
- Local errors and exception conditions detected by Txns (data for Txn missing, insufficient account balance)
- Concurrency control mechanism; may abort Txn due to serializability violation; may abort one or more Txns due to deadlocks
- Disk failures; read or write malfunction
- Physical problems and catastrophes; power, A/C, fire, thelft, overwriting disks

2. Transactions and System Concepts

Transaction states and additional operations

- A Txn is atomic, either done or aborted
- For recovery purposes, it keeps track of start, terminate and commits or aborts
- Recovery manager keeps track of the following:
    - BEGIN_TRANSACTION
    - READ OR WRITE
    - END_TXN
    - COMMIT_TXN
    - ROLLBACK or ABORT

Failed state need rollback operations

Fig. 20.4

The System Log

- To recover from failures, the system maintains a log file to keep track of all Txns
- Log is a sequential appended file kept on disk (disk or catastrophic failures is a problem)
- Log buffers in memory (filled goes to disk)
- Log records are written to log files
- Txn has a unique Txn id
- For each Txn log records are kept
- All permanent changes occur within a Txn
- Recovery from a failed Txn amounts to either undoing or redoing Txn operations individually from a log file
- Write entry in log file helps to UNDO operations

- REDO is required if the log files are not written to disk due to failures

1. [start_transaction, T]
2. [write_item, T, X, old-value, new-value]
3. [read_item, T, X]
4. [commit, T]
5. [abort, T]

Commit point of Transaction

- A Txn reaches commit point if all operations are recorded on disk and the log file is updated
- The Txn then writes commit record in the log file after commit point
- If a Txn fails without a commit record in the log file, Txn may have to be rolled back
- Log file must be kept on disk
- Force-writing log files to disk often

DBMS-specific Buffer Replacement Policies

- DBMS cache maintains buffers for Txns that are running
- A page replacement algorithm is used to select a particular buffer to be replaced
- Some algorithms:
    - Domain specific method (DS): based on domains of buffers; index pages; data file pointers; log buffers etc…; separate LRU for each domain buffers; one can also design group LRU with a given priority for each domain

- Hot set method: determines disk pages needed repeatedly and keeps them in the buffers (nested queries; outer query records needed in the buffer)
- The DBMIN method: page replacement policy uses a method known as QLSM (query locality set method); predetermines the pattern of page references for each algorithm for a particular type of database operation; calculate a locality set of each operation (SELECT, JOIN, …)

3. Desirable properties of transactions

ACID properties:

Atomicity: permitted in its entirety or not permitted at all

Consistency: Txn should be consistency preserving; Txn with no interference from other Txns while executing

Isolation: A Txn should appear to be executed in isolation, although many Txns are executing concurrently

Durability or persistency: The changes applied to the database by a committed Txn must persist in the database. These changes must not be lost due to any failure.

4. Characterizing schedules based on recoverability

- When Txns are executing concurrently in interleaved fashion, then the order of execution of operations from all various transactions is known as a Schedule (or history)
- Schedules (or histories) of transactions:
  - Ordering of the operations from different transactions

- The ordering of operations for a given Txn Ti must be preserved in S
- For recovery purposes, we are only interested in read, write, and commit operations in the schedule

Fig 20.3(a): S

Sa: r1(X); r2(X); w1(X); r1(Y); w2(X); w1(Y)

Fig 20.3(b): S

Sb: r1(X); w1(X); r2(X); w2(X); r1(Y); a1

<u>Conflicting operations in a schedule</u>

- Two operations in a schedule are said to conflict if they satisfy:
  - They belong to different Txns
  - They access the same item X
  - At least one of the operation is a write (X)

Sa:

r1(x)	r2(X)
w1(X)	w2(X)
r1(Y)	
w1(Y)	

Two operations are conflicting, if changing their order can result in a different outcome

---

r1(X)

w2(X)

value is read before change

---

w2(X)

r1(X)

value is read after change

---

read-write conflict

---

w1(X)

w2(X)

---

w2(x)

w1(X)

---

write-write conflict

Two read operations are not in conflict..

Schedule S of n transactions T1, T2, ….Tn is said to be a complete schedule if the following conditions hold:

1. The operations in S are exactly those operations in T1, T2, …Tn including a commit or abort
2. For any pair of operations from the same Txn Tp, their relative order of operations in S is same as their order of appearance in Tp
3. For any two conflicting operations one of the two must occur before the other in the schedule (does not define which occurs first, it is called a partial order)
4. All operations must appear in the schedule

In general, it is difficult to achieve complete schedules as new transactions enter into S while old transactions are executing.

Characterizing Schedules Based on Recoverability

For some schedules it is easy to recover from failure and for some schedules it is not possible to recover or not recoverable at all.

It is possible to identify recoverable schedules.

Assume a Txn T is committed, there is no need to rollback. The schedules that meet this condition is called recoverable schedules.

A schedule where a committed transaction have to be rolled back is called a non-recoverable schedule

A schedule S is recoverable, if no transactions T in S commits until all transaction T’ that have written some item in X that T reads have committed.

Write_item(X)

read_item(X)

c2

c1

---not recoverable

---

r1(X)

r2(X) (read X)

w1(X) (write X)

r1(Y)

w2(X)

c2 (committed)

w1(Y)

c1 (committed)

---- recoverable, however it was lost update

---

r1(X)

w1(X) (writes X)

r2(X) (reads X)

r1(Y)

w2(X)

c2 (committed early)

a1

(committed late)

-----NOT recoverable

Sd: {r1(X); w1(X); r2(X); r1(Y); w2(X); w1(Y); c1; c2 }

---

r1(X)

w1(X) (write X)

r2(X) (read X)

r1(Y)

w2(X)

w1(Y)

C1 (committed first)

C2 (committed later)

---- recoverable

It is recoverable schedule, no committed transactions ever needs to be rolled back, so durability of a committed Txn is not violated.

It is possible for a phenomenon called cascading rollback (or cascading abort) to occur in some recoverable schedules. A uncommitted Txn has to be rolled back because it read an item from a Txn that failed.

Se: {r1(X); w1(X); r2(X); r1(Y); w2(X); w1(Y); a1; a2}

---

r1(X)

w1(X)

r2(X) (read an item from T1)

r1(Y)

w2(X)

w1(Y)

a1

a2

T2 has to be rolled back

To avoid rollback, r2(X) should occur after T1 is committed…

Strict Schedule

In which Txns can neither read nor write an item X until the last Txn that wrote X has committed or aborted.

Strict schedules simplify the recovery process.

Any strict schedule is cascade less. Any cascade less schedule is recoverable. Most recovery protocols only allow strict schedules so that the recovery is simple.

5. Characterizing schedules based on serializability

- Characterize schedules based on correctness; such schedules are known as serializable
- If no interleaving is permitted then schedules follow operations in T1 and T2 as serial operations. T1; T2 or T2; T1 (serial schedules); Fig 20.5 (correct schedules)
- If interleaving is allowed, there are many possibilities of operations in S
- The concept of serializability is used to find correct schedule

A schedule S is serial, if for every transaction T participating in S, all operations of T are executed consequently, without any interleaved operations from other Txns, otherwise it is not serial.

Every serial schedule is correct (no concurrency).

A schedule S of n transactions is serializable if it is equivalent to some serial schedule of the same n transactions. Serializable implies it is correct.

Figure 20.4 State transition diagram illustrating the states for transaction execution.

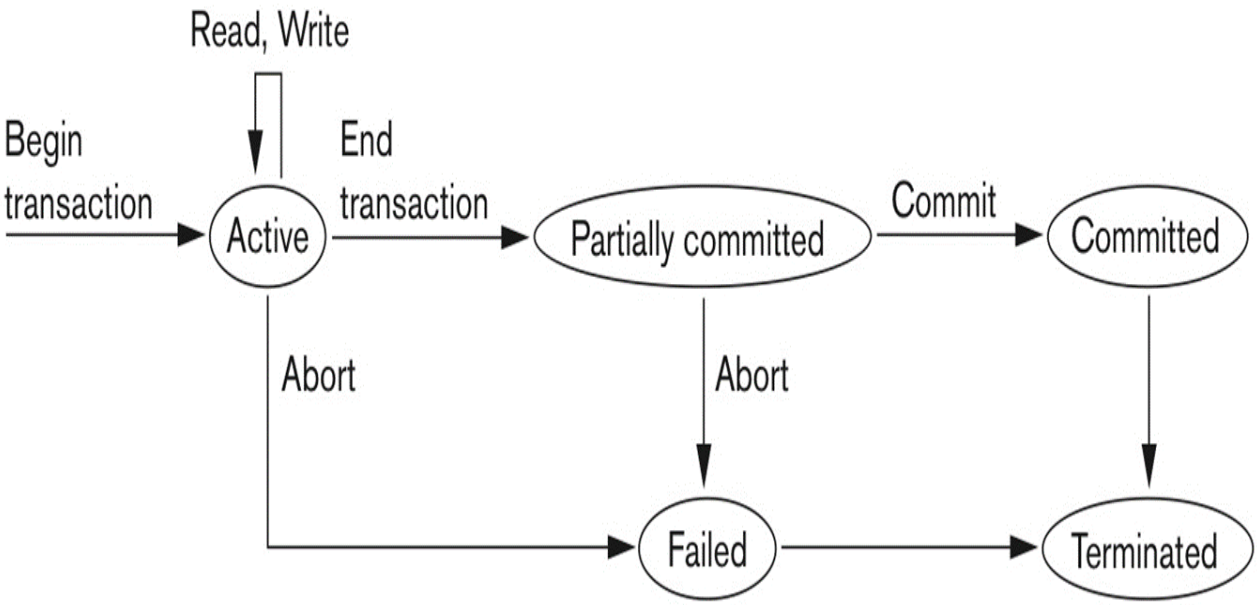
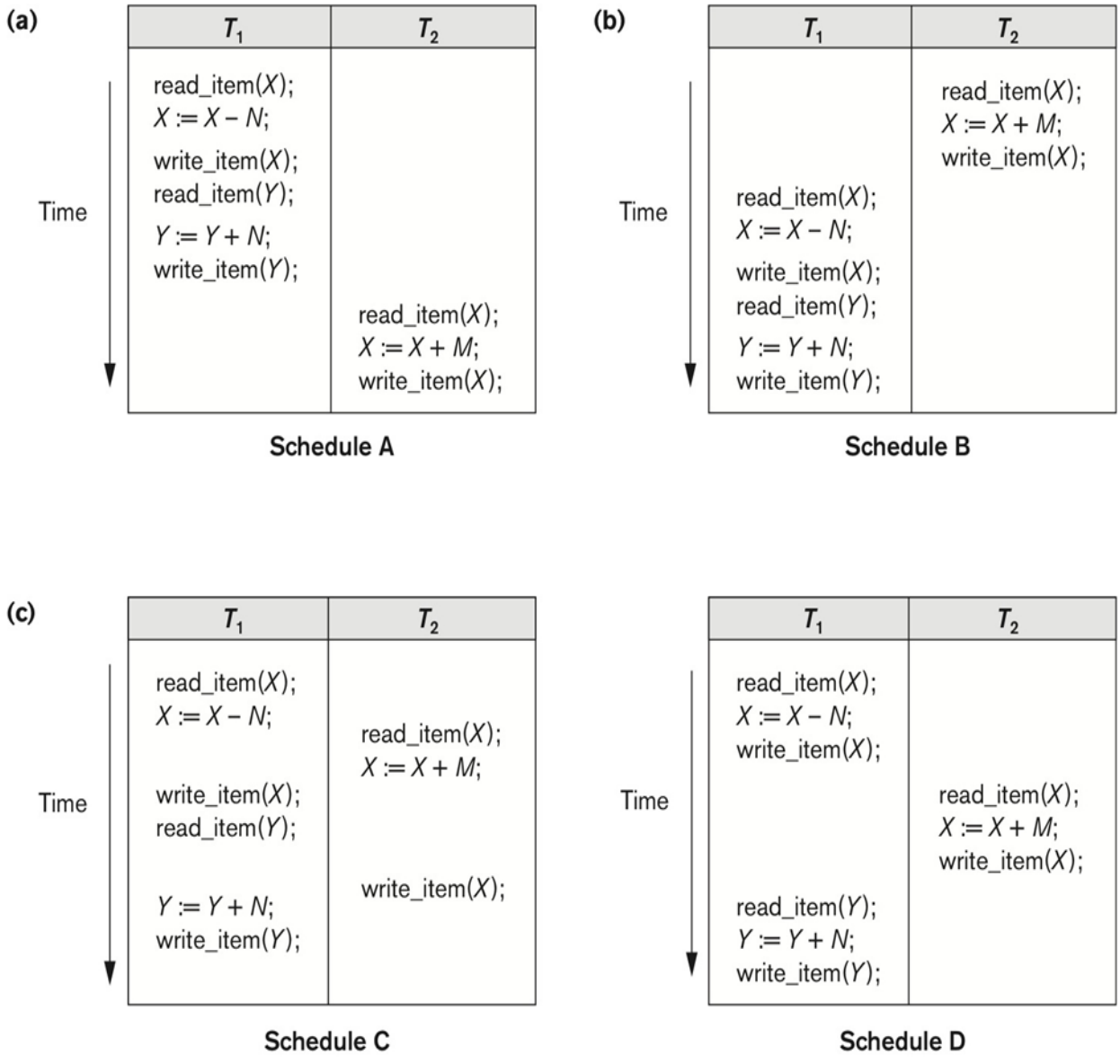


Figure 20.5 Examples of serial and nonserial schedules involving transactions T_1 and T_2. (a) Serial schedule A: T_1 followed by T_2. (b) Serial schedule B: T_2 followed by T_1. (c) Two nonserial schedules C and D with interleaving of operations.



Testing for a Serializability of a Schedule

Uses a precedence graph (directed graph)

1. For each Ti in S create a node Ti
2. For each case in S, where a Tj executed a read_item(X) after Ti executes a write_item(X), create an edge Ti->Tj in the graph
3. For each case in S, where a Tj executed a write_item(X) after Ti executes a read_item(X), create an edge Ti->Tj in the graph
4. For each case in S, where a Tj executed a write_item(X) after Ti executes a write_item(X), create an edge Ti->Tj in the graph
5. The S is serializable iff the precedence graph has no cycles.

(Ti->Tj means, transaction I should come before transaction j)

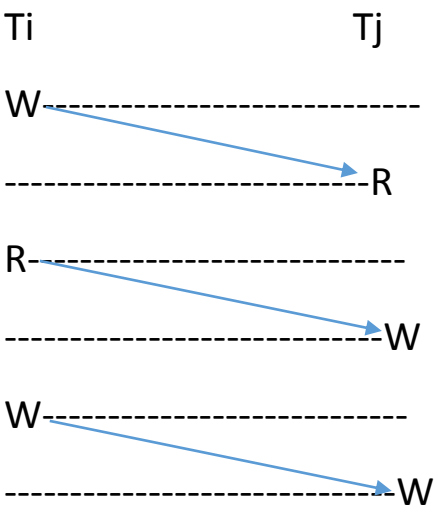


Fig. 20.5(a)

T1	**T2**
read_item(X)	
X = X – N	
write_item(X)	
read_item(Y)	
Y = Y+N	
write_item(Y)	
	read_item(X)
	X = X + M
	Write_item(X)

i j

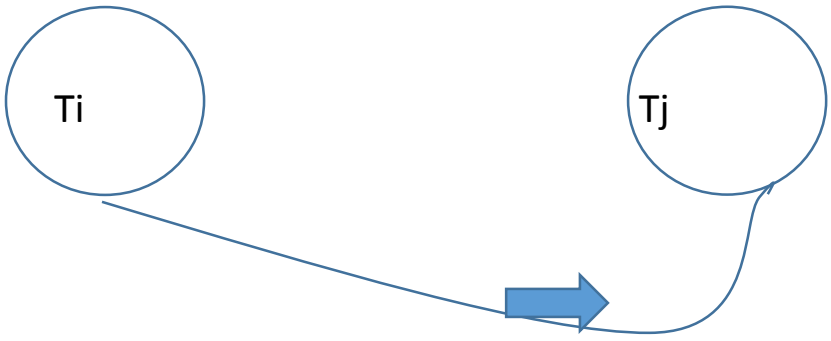


Fig. 20.5(b)

T1	**T2**
	read_item(X)
	X = X + M
	Write_item(X)
read_item(X)	
X = X – N	
write_item(X)	
read_item(Y)	
Y = Y+N	
write_item(Y)	

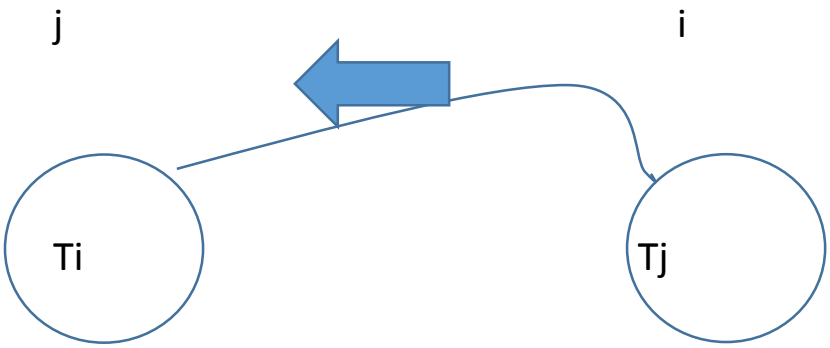
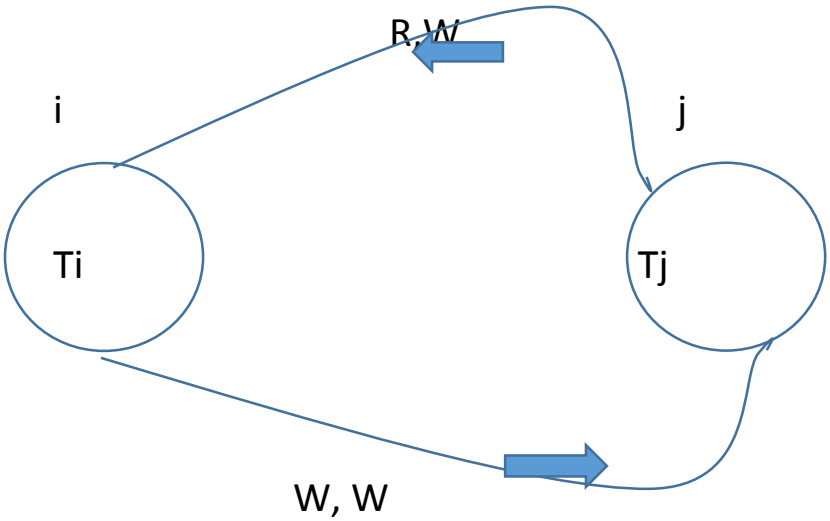


Fig. 20.5(c)

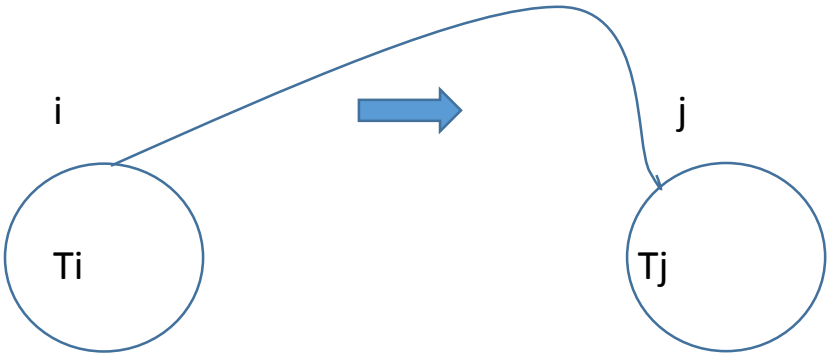
T1	**T2**
read_item(X)	
X = X – N	
	read_item(X)
	X = X + M
write_item(X)	
read_item(Y)	
	Write_item(X)
Y = Y+N	
write_item(Y)	



Not recoverable

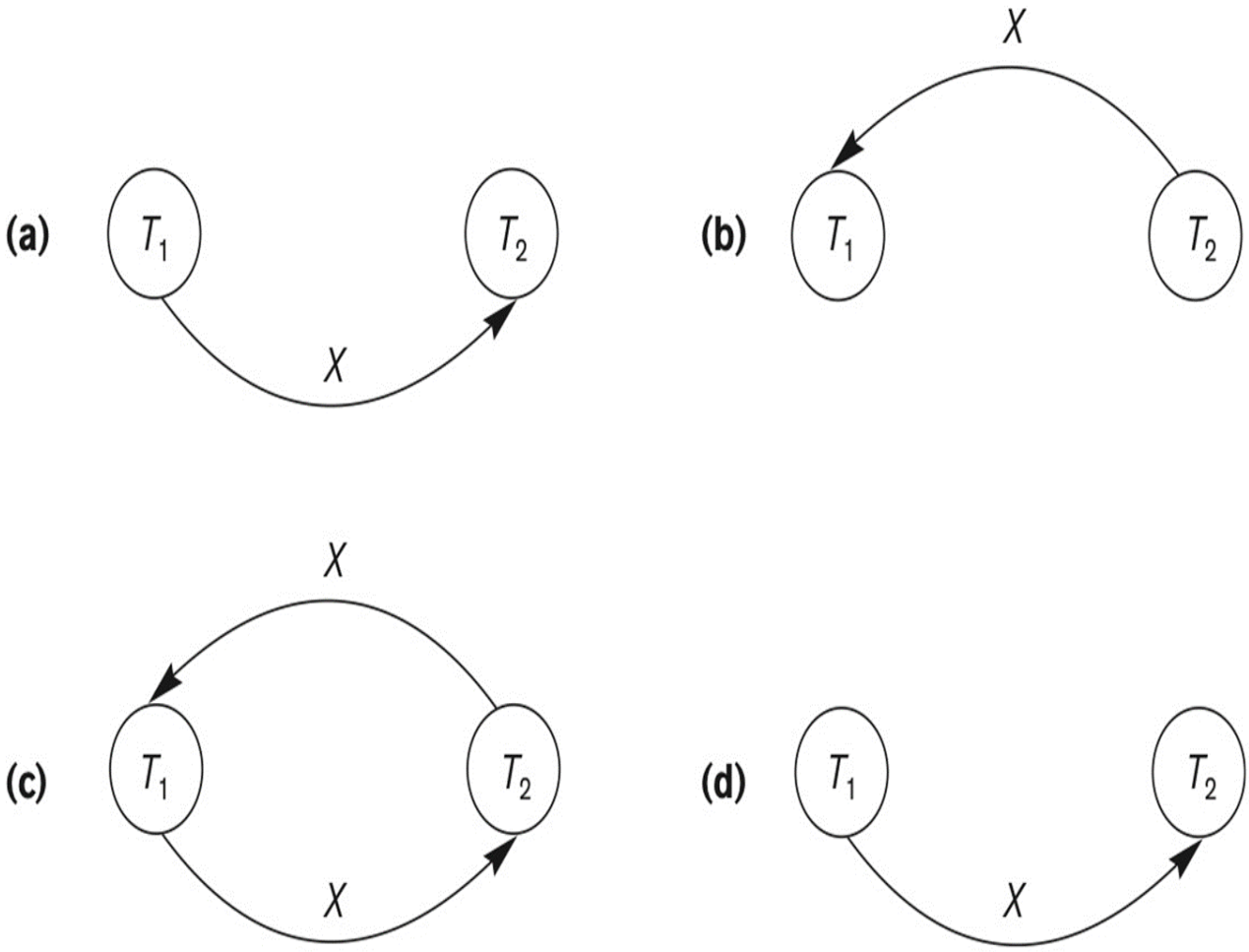
Fig. 20.5(d)

T1	**T2**
read_item(X)	
X = X – N	
write_item(X)	
	read_item(X)
	X = X + M
	Write_item(X)
read_item(Y)	
Y = Y+N	
write_item(Y)	



recoverable

Figure 20.7 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).



SKIP rest of this section

6. **Transaction Support in SQL**

A transaction is a logical unit of work and guaranteed to be atomic. A single SQL statement is always considered to be atomic. A transaction is done implicitly when an SQL statement is encountered. Every Txn must have an explicit end statement, either COMMIT or a ROLLBACK. Every Txn has certain characteristics attributed to it. SET TRANSACTION statement specifies these characteristics. The characteristics are : access mode, diagnostics area size and isolation level.

Access mode: READ ONLY or READ WRITE (default)

Diagnostic area size: n number of conditions that can be held in diagnostics (errors, etc..)

Isolation level: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE (default)

If a Txn runs at a serializable level, then one or more of the following three violations may occur:

(1) Dirty read: T1 read a value updated by T2 which is not committed; if T2 fails and aborted, then T1 has incorrect value

(2) Nonrepeatable read: T1 read a value, T2 may update this value; later if T1 reads, it will see a different value

(3) Phantoms: T1 may read a set of rows with a condition; now T2 inserts a new row r with the same conditions as in T1; r is called a phantom record. T1 do not see r. In serializable option, there are no phantom records.

A simple SQL Transaction:

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION READ WRITE
        DIAGNOSTIC SIZE 5
        ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT INTO EMPLOYEE (Fname, Lname, Ssn, Dno,
     Salary) VALUES ('Robert', 'Smith', '991004329'35000);
EXEC SQL UPDATE EMPLOYEE
     SET Salary = Salary * 1.1 WHERE Dno = 2;
EXEC SQL COMMIT; GOTO THE_END;
UNDO: EXEC SQL ROLLBACK;
THE_END: …;
//inserts a new row in the EMPLOYEE table, updates the salary of
//employees who work in the department 2.
//If an error occurs in any statement, then it will roll back
//
```

Snapshot Isolation: A transaction starts with the current state of the database and it does not see changes in the database while executing.