
CS3402 Chapter 10: Transaction Management

Single-User versus Multiuser Systems

- A database system is single-user if at most one user at a time can use the system, and it is multiuser if many users can use the system—and hence access the database—concurrently.
- Single-user database system are mostly restricted to personal computer systems; most other DBMSs are multiuser, for example, an airline reservations system is used by hundreds of users and travel agents concurrently.
- In multiuser systems, hundreds or thousands of users are typically operating on the database by submitting **transactions** concurrently to the system. So we need **transaction** concurrency control to make the database system consistent.

What is a Transaction?

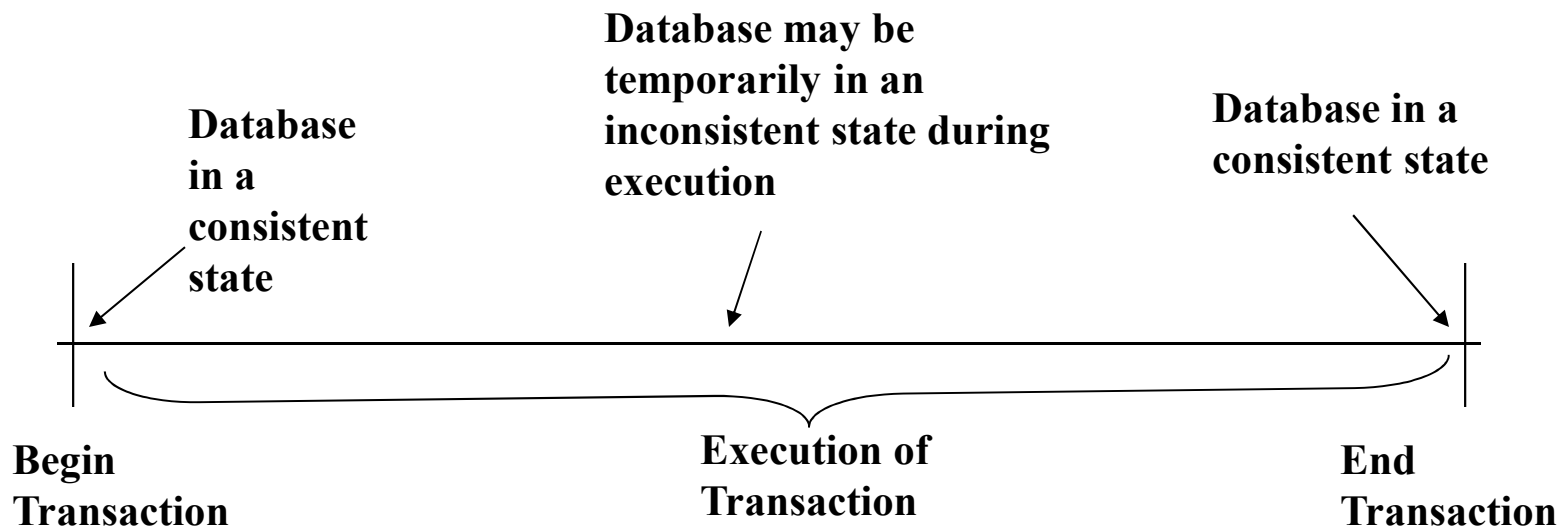
- A transaction is an executing program that forms a logical unit of database processing.
E.g. use of ATM and buy online tickets
- A transaction includes one or more database access operations—these can include insertion, deletion, modification (update), or retrieval operations.
- To specify the transaction boundaries, we use begin transaction and end transaction statements in an application program; in this case, all database access operations between the two are considered as forming one transaction.

What is a transaction?

- Structurally, each transaction is a process and consists of **atomic steps**. Each atomic step is called an **operation**.
- A transaction = database operations + transaction operations
- **Database operations**: read and write operations on a database
 - ◆ Read operation: to read the value of a data item or a group of items, e.g., SELECT
 - ◆ Write operation: to create a new value for a data item or a group of items, e.g., UPDATE
 - ◆ In between the read/write operations, there may be computation
- **Transaction operations**: a begin operation and an end operation (commit or abort)
 - ◆ For transaction management (where to start and where to finish)
 - ◆ The new values from a transaction will become permanent only if the transaction is **committed** successfully

Transaction Structure & Database Consistency

Example: T1: Begin; A:= R(a); B:= R(b) C:= A + B; W(c):= C; End



The whole transaction is considered as an **atomic unit**

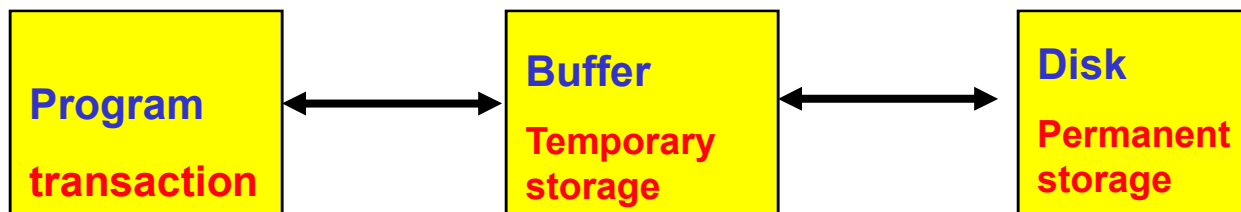
- ◆ Partial results are not allowed and is considered to be incorrect
- ◆ **Atomicity:** All or nothing

Read and Write Operations

- Data are resided on **disk** and the basic unit of data transferring from the disk to the main memory is **one disk block**
- In general, a data item (what is read or written) will be the field/fields of some records in the database (in a disk block)
- **Read (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 (if that disk block is not already in some main memory buffer)
 - ◆ Search for the required value in the buffer
 - ◆ Copy item X from the buffer to the program variable named X

Read and Write Operations

- **Write(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 (if that disk block is not already in some main memory buffer)
 - ◆ Search for the required value in the buffer
 - ◆ 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 (either immediately or at **some later** point in time)



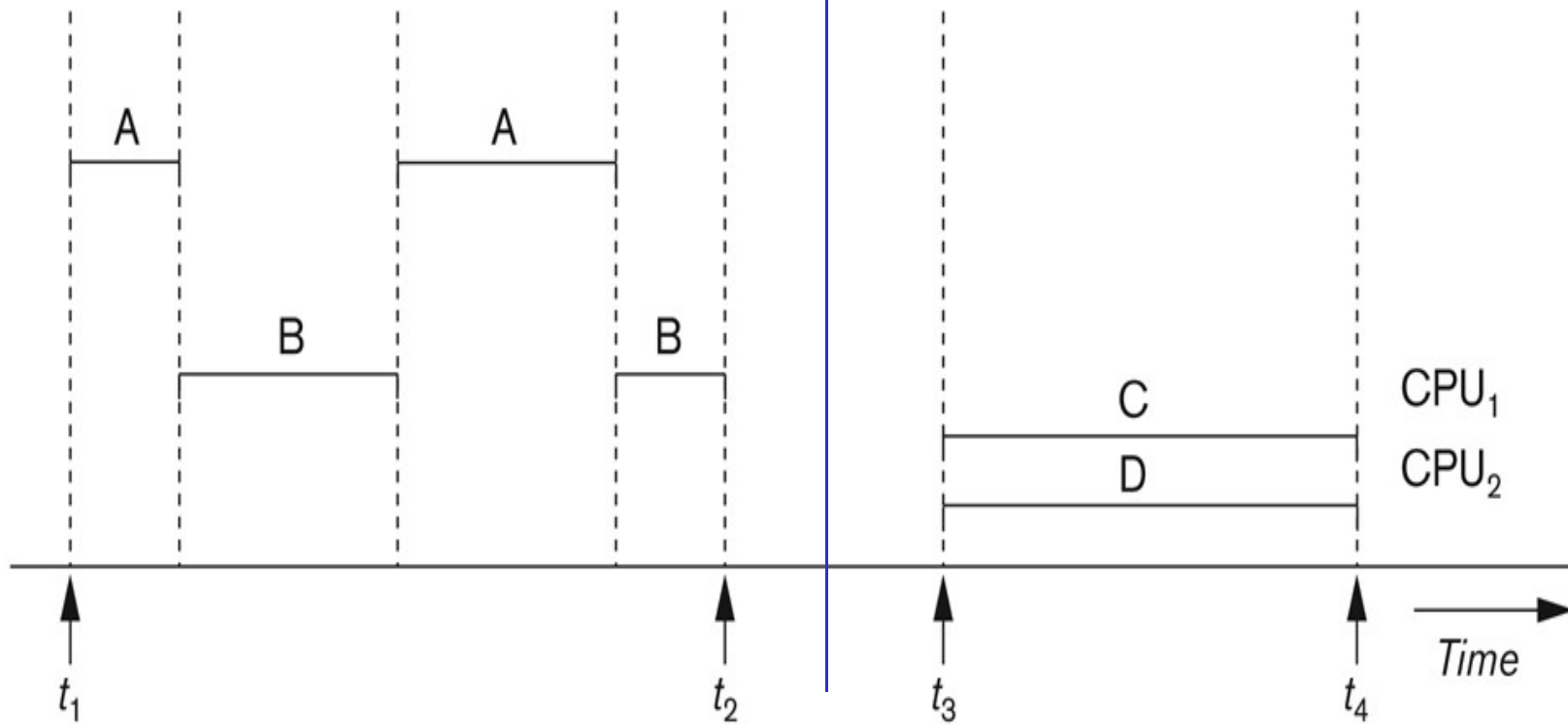
***How to improve transaction
processing performance?***

***Serial Schedule Vs. Serializable
Schedule***

Transaction Processing Performance

- Multiuser systems:
 - ◆ Many users (transactions) can access the database **concurrently** at the same time
- Serial execution: execute transitions one by one (slow)
- Concurrent execution:
 - ◆ **Interleaved processing:**
 - ◆ Concurrent execution of processes/transactions are interleaved in a single CPU system
 - ◆ Parallel processing:
 - ◆ Processes/transactions are concurrently executed in multiple CPUs system
- Higher Concurrency (more than one transaction is executing)
 - ◆ **Better performance**, i.e., lower response time
 - ◆ **Problem:** difficult to maintain database consistency if it is not well

Interleaved vs Parallel



While waiting disk data, the CPU processes another transaction

Figure 17.1


Interleaved processing versus parallel processing of concurrent transactions.

Transaction Schedule

- Transaction schedule
 - ◆ When transactions are executing **concurrently** in an **interleaved** fashion or **serially**, the order of execution of **operations** from all transactions forms a **transaction schedule**
- A schedule S of n transactions T_1, T_2, \dots, T_n is an ordering of the operations of the transactions subject to the constraint:
 - ◆ For each transaction T_i that participates in S , the operations of T_i in S must appear in the same order as in T_i (operations from other transactions T_j can be interleaved with the operations of T_i in S)
- A concurrent schedule: a new transaction starts **BEFORE** the completion of the current transaction
- A serial schedule: a new transaction only starts **AFTER** the current transaction is completed

Schedule Examples

- T1: Read(x), Write(y), Write(x)
- T2: Read(x), Read(y), Write(y)



T1	T2
Read(x) Write(y) Write(x)	 Read(x) Read(y) Write(y)

Execute
time

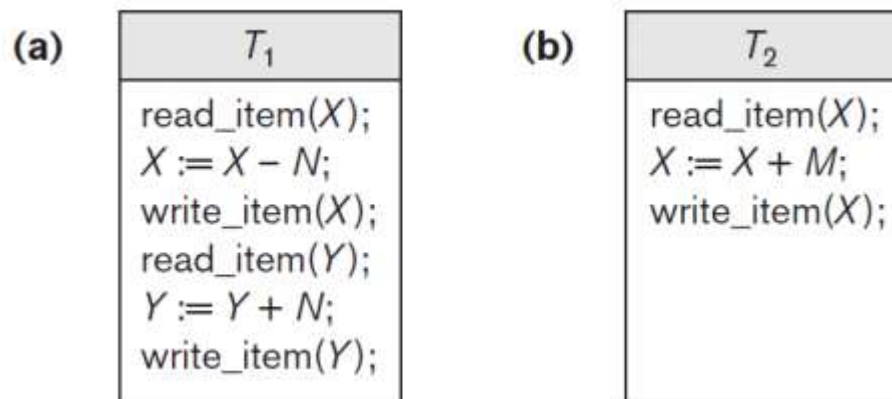
serial schedule

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

concurrent schedule

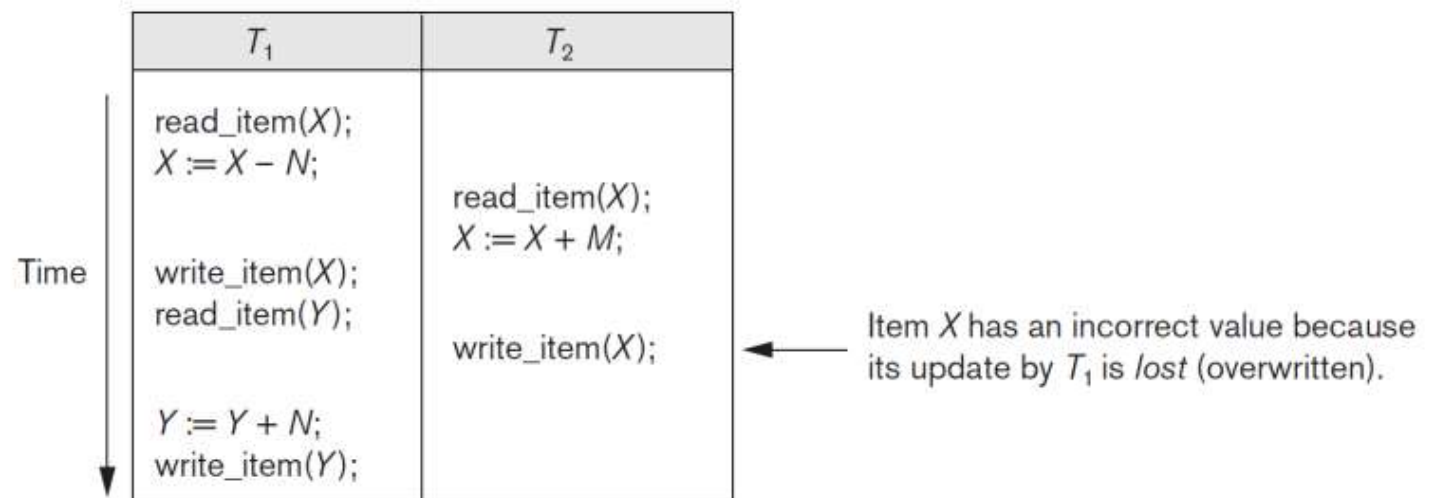
Consistency Problems in Concurrent Schedule

- Several problems can occur when concurrent transactions execute in an uncontrolled manner. We illustrate some of these problems by referring to a much simplified airline reservations database in which a record is stored for each airline flight.
- X , Y are database items to represent the number of reserved seats on each flight.
- Figure (a) shows a transaction T_1 that transfers N reservations from flight(X) to flight(Y). Figure (b) shows a simpler transaction T_2 that just reserves M seats on flight (X).



Consistency Problems in Concurrent Schedule

- **Lost Update Problem** (write/write conflicts)
 - ◆ 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 (inconsistent)
- For example, if $X = 80$, $Y = 40$ at the start (originally there were 80 reservations on the flight X, 40 on the flight Y), $N = 5$ (T_1 transfers 5 seat reservations from the flight X to the flight Y), and $M = 4$ (T_2 reserves 4 seats on X), the final result should be $X = 79$, $Y = 45$. However, in the interleaving of operations, it is $X = 84$ because the update in T_1 that removed the five seats from X was lost.



Consistency Problems in Concurrent Schedule

■ Incorrect Summary Problem (read/write conflicts)

- ◆ If one transaction is calculating an aggregate summary function on a number of database items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated.

If the interleaving of operations occurs, the result of T3 will be off by an amount N because T3 reads the value of X after N seats have been subtracted from it but reads the value of Y before those N seats have been added to it.

T_1	T_3
	$sum := 0;$ $read_item(A);$ $sum := sum + A;$ \vdots
$read_item(X);$ $X := X - N;$ $write_item(X);$	$read_item(X);$ $sum := sum + X;$ $read_item(Y);$ $sum := sum + Y;$
$read_item(Y);$ $Y := Y + N;$ $write_item(Y);$	

← T_3 reads X after N is subtracted and reads Y before N is added; a wrong summary is the result (off by N).

Read and write operation conflict rules

<i>Operations of different transactions</i>		<i>Conflict</i>	<i>Reason</i>
<i>read</i>	<i>read</i>	No	Because the effect of a pair of <i>read</i> operations does not depend on the order in which they are executed
<i>read</i>	<i>write</i>	Yes	Because the effect of a <i>read</i> and a <i>write</i> operation depends on the order of their execution
<i>write</i>	<i>write</i>	Yes	Because the effect of a pair of <i>write</i> operations depends on the order of their execution

Conflict equivalent:

- ◆ Two schedules are said to be conflict equivalent if the order of any two conflicting operations (RW,WW) is the same in both schedule

Schedules Classified on Serializability

- 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
- ◆ Serial schedules can maintain the database consistency
 - ◆ BUT, poor performance

- Serializable schedule:

- ◆ A concurrent schedule S which is conflict equivalent to a **serial** schedule.
- ◆ Can guarantee the database consistency and can have better performance.

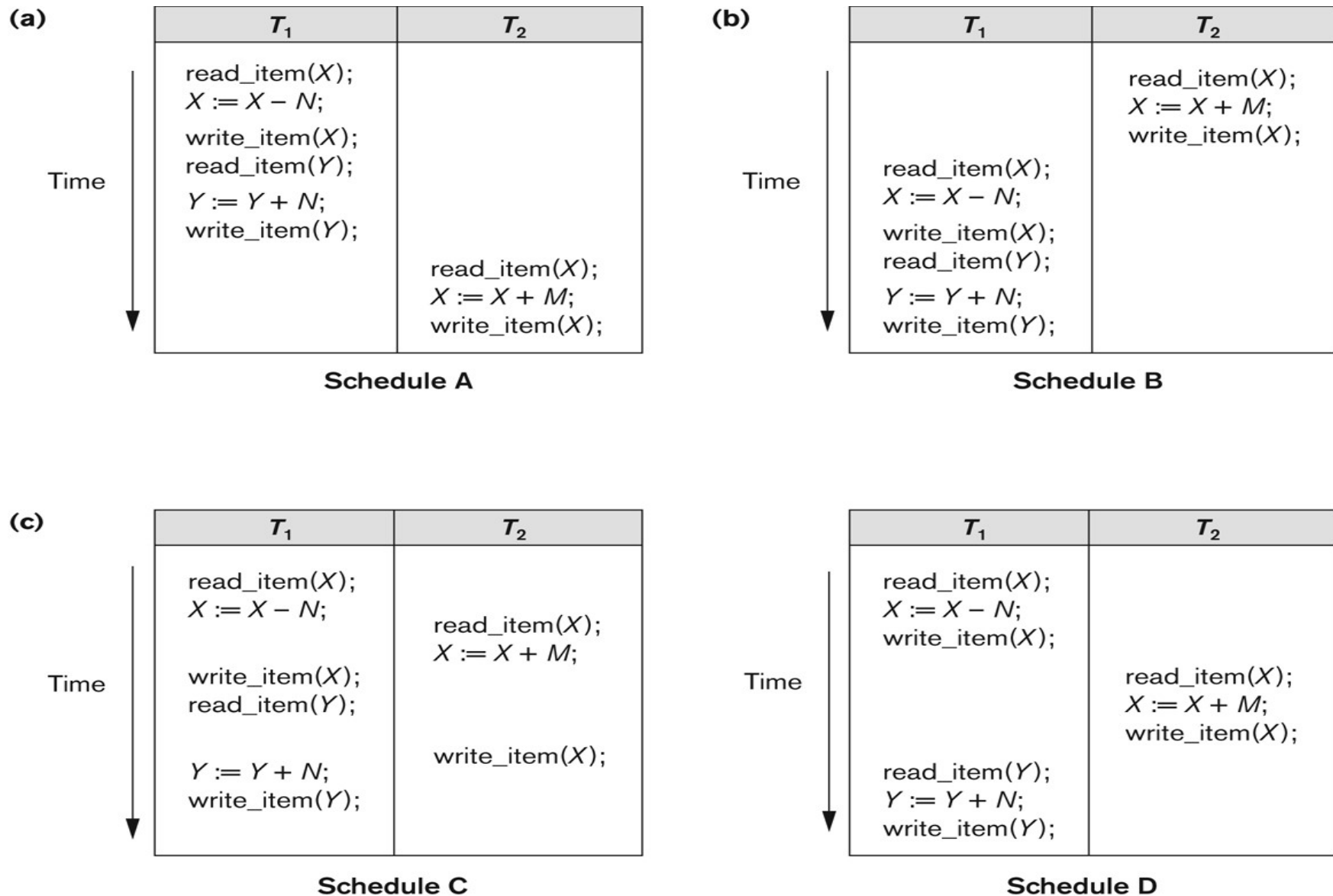
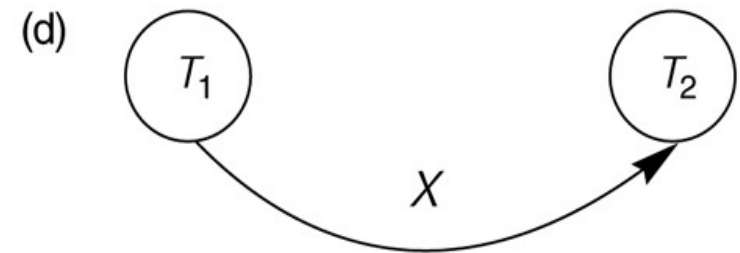
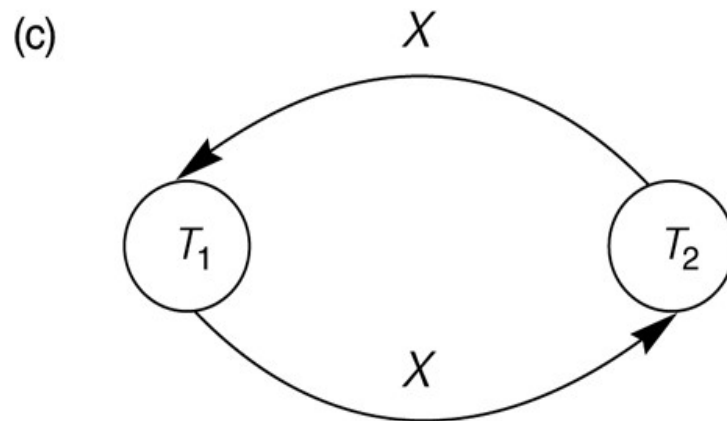
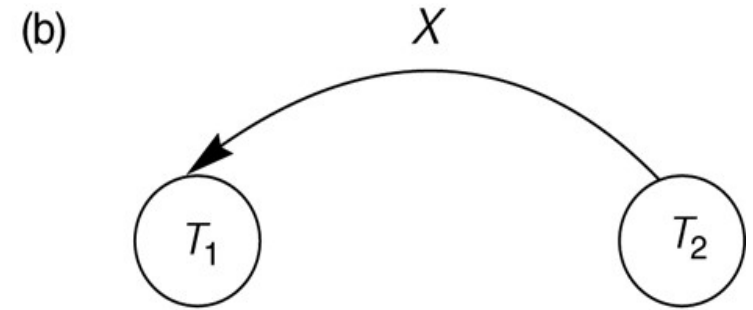
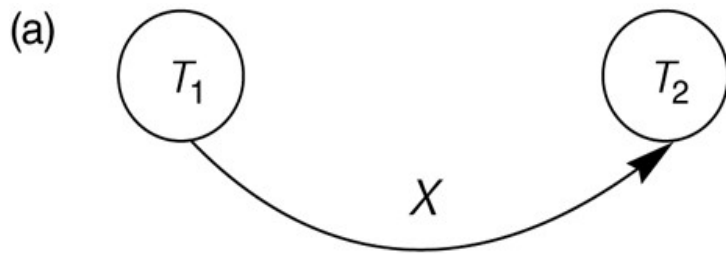


Figure 17.5

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.

Serialization Graphs

- The determination of a conflict serializable schedule can be done by the use of serialization graph (SG) or called precedence graph
- A serialization graph tells the effective execution order of a set of transactions
- The set of edges consists of all edges $T_i \rightarrow T_j$ for which one of the following three conditions holds:
 - ◆ W/R conflict: T_i executes write(x) before T_j executes read(x)
 - ◆ R/W conflict: T_i executes read(x) before T_j executes write(x)
 - ◆ W/W conflict: T_i executes write(x) before T_j executes write(x)
- Each edge $T_i \rightarrow T_j$ in a SG means that at least one of T_i 's operations precede and conflict with one of T_j 's operations
- Serializability theorem:
 - ◆ A schedule is serializable iff the SG is acyclic



- 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 conflict serializable).
 - ◆ (d) Precedence graph for schedule D (conflict serializable, equivalent to schedule A).

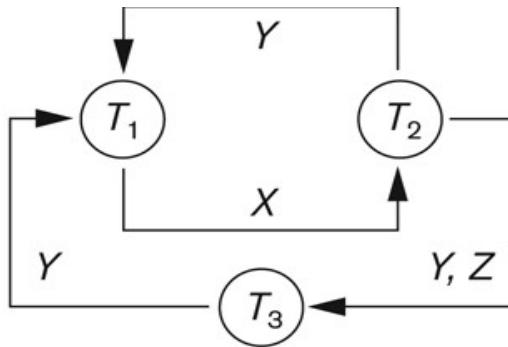
Another example of serializability testing.
(a) The read and write operations of three transactions T_1 , T_2 , and T_3 . (b) Schedule E. (c) Schedule F.

Time

Schedule E

Another Example of Serializability Testing

(d)



Equivalent serial schedules

None

Reason

Cycle $X(T_1 \rightarrow T_2), Y(T_2 \rightarrow T_1)$

Cycle $X(T_1 \rightarrow T_2), YZ(T_2 \rightarrow T_3), Y(T_3 \rightarrow T_1)$

Another Example of Serializability Testing

Figure 17.8

Another example of serializability testing. (a) The read and write operations of three transactions T_1 , T_2 , and T_3 . (b) Schedule E. (c) Schedule F.

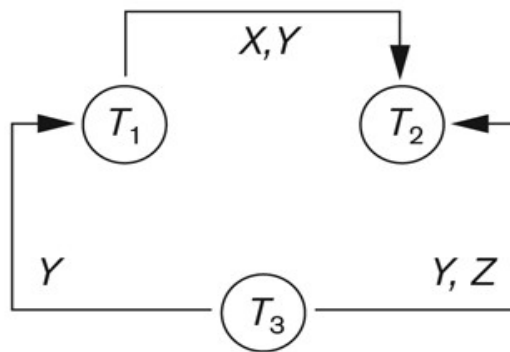
(c)

	Transaction T_1	Transaction T_2	Transaction T_3
Time ↓	read_item(X); write_item(X); read_item(Y); write_item(Y);	read_item(Z); read_item(Y); write_item(Y); read_item(X); write_item(X);	read_item(Y); read_item(Z); write_item(Y); write_item(Z);

Schedule F

Another Example of Serializability Testing

(e)



Equivalent serial schedules

$T_3 \rightarrow T_1 \rightarrow T_2$

References

- 6e
 - ◆ *Chapter 20, pages 721-750*