CO527: Advanced Database Systems

Sampath Deegalla
dsdeegalla@pdn.ac.lk, sampath@eng.pdn.ac.lk

1 Introduction to Transaction Processing

1.1 Introduction

- 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.
- 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 a program.
- Transaction boundaries:
  - Begin and End transaction.
  - An application program may contain several transactions separated by the Begin and End transaction boundaries.

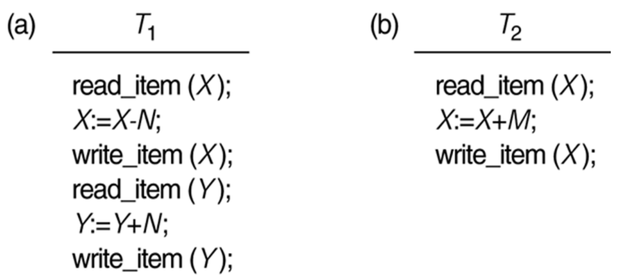
Simple Model of a Database (for purposes of discussing transactions):

- A database is a collection of named data items
- Granularity of data - a field, a record , or a whole disk block (Concepts are independent of granularity)
- Basic 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.

Read and Write Operations

- Basic unit of data transfer from the disk to the computer main memory is one block. In general, a data item (what is read or written) will be the field of some record in the database, although it may be a larger unit such as a record or even a whole 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 (if that disk block is not already in some main memory buffer).
  - Copy item X from the buffer to the program variable named X.
- 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 (if that disk block is not already in some main memory 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).

Two sample transactions



(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 1: Two sample transactions: (a) Transaction T1 (b) Transaction T2

Why Concurrency Control is needed

- *The Lost Update Problem:* See Figure 2
  - 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:* See Figure 3
  - This occurs when one transaction updates a database item and then the transaction fails for some reason.
  - The updated item is accessed by another transaction before it is changed back to its original value.
- *The Incorrect Summary Problem:* See Figure 4
  - 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.

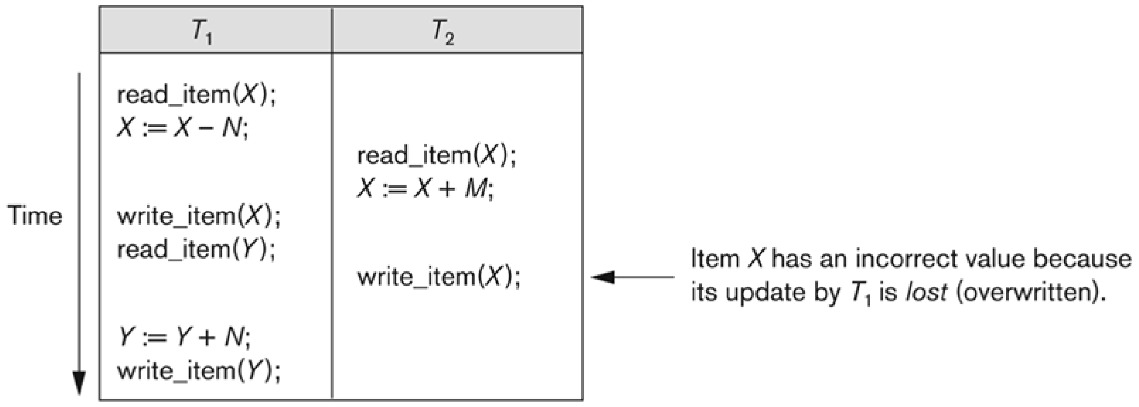


Figure 2: The lost update problem

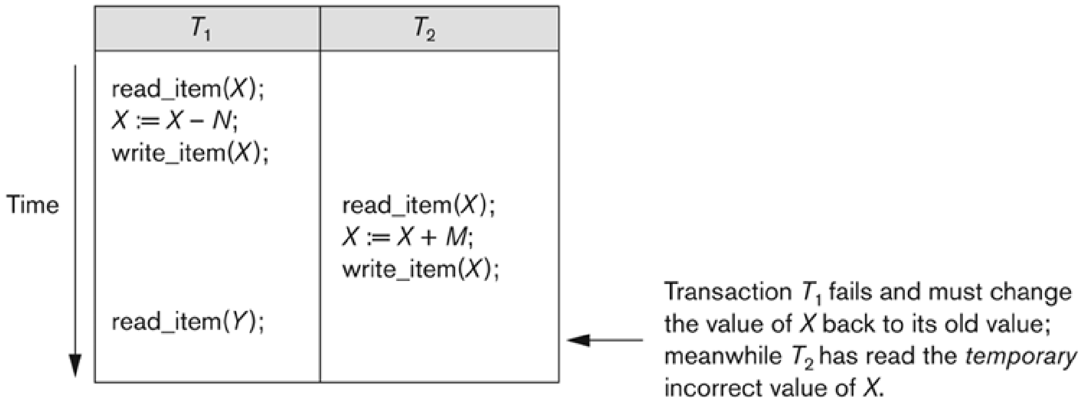
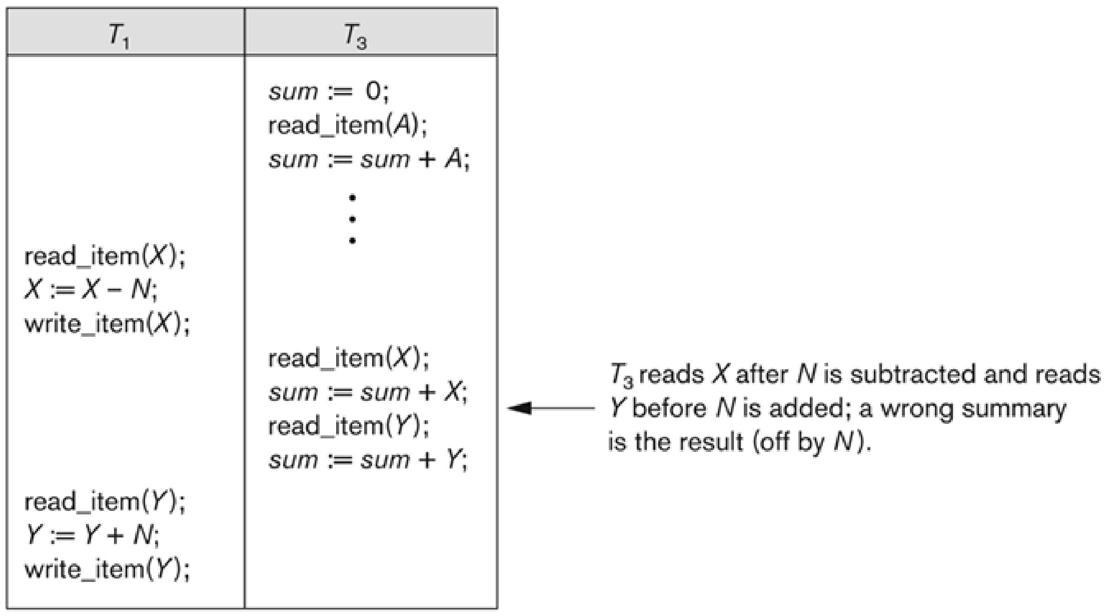

T_1
T_2
Time
read_item(X);
$X := X - N$;
write_item(X);
read_item(X);
$X := X + M$;
write_item(X);
read_item(Y);
Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the *temporary* incorrect value of X.

Figure 3: The temporary update problem

Why recovery is needed: (What causes a Transaction to fail)

- *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 computers internal memory may be lost.
- *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.
- *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.
- *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.
- *Disk failure:* Some disk blocks may lose their data because of a read or write malfunction or



T_1	T_3
	$sum := 0$;
	read_item(A);
	$sum := sum + A$;
	⋮
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).

Figure 4: The incorrect summary problem

because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.

- *Physical problems and catastrophes:* 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.

1.2 Transaction and System Concepts

- A transaction is an atomic unit of work that is either completed in its entirety or not done at all.
  - For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts.
- Transaction states: See Figure 5
  - Active state
  - Partially committed state
  - Committed state
  - Failed state
  - Terminated State

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.

- *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.

Recovery techniques use the following operators:

- *undo:* Similar to rollback except that it applies to a single operation rather than to a whole transaction.
- *redo:* This specifies that certain transaction operations must be redone to ensure that all the operations of a committed transaction have been applied successfully to the database.

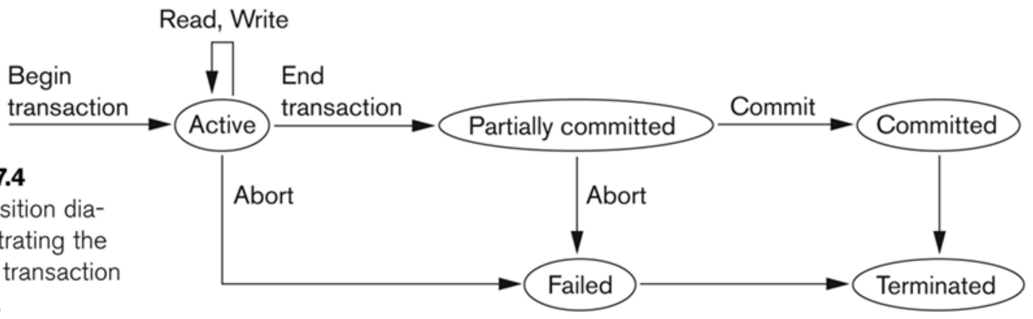


Figure 17.4
State transition diagram illustrating the states for transaction execution.

Figure 5: State transition diagram illustrating the states for transaction execution

The System Log (DBMS Journal)

- 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.
- 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.

Recovery using log records:

- If the system crashes, we can recover to a consistent database state by examining the log and using one of the Concurrency Control techniques discussed later.
  - Because the log contains a record of every write operation that changes the value of some database item, it is possible to undo the effect of these write operations of a transaction T by tracing backward through the log and resetting all items changed by a write operation of T to their old_values.
  - We can also redo the effect of the write operations of a transaction T by tracing forward through the log and setting all items changed by a write operation of T (that did not get done permanently) to their new_values.

Commit Point of a Transaction

- Definition a Commit Point:
  - A transaction T reaches its commit point when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database has been recorded in the log.
  - Beyond the commit point, the transaction is said to be committed, and its effect is assumed to be permanently recorded in the database.
  - The transaction then writes an entry [commit,T] into the log.
- Roll Back of transactions:
  - Needed for transactions that have a [start_transaction,T] entry into the log but no commit entry [commit,T] into the log.
- Redoing transactions:
  - Transactions that have written their commit entry in the log must also have recorded all their write operations in the log; otherwise they would not be committed, so their effect on the database can be redone from the log entries. (Notice that the log file must be kept on disk. At the time of a system crash, only the log entries that have been written back to disk are considered in the recovery process because the contents of main memory may be lost.)
- Force writing a log:
  - Before a transaction reaches its commit point, any portion of the log that has not been written to the disk yet must now be written to the disk.
  - This process is called force-writing the log file before committing a transaction.

1.3 Desirable Properties of Transactions

ACID properties:

- *Atomicity*: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
- *Consistency preservation*: A correct execution of the transaction must take the database from one consistent state to another.
- *Isolation*: A transaction should not make its updates visible to other transactions until it is committed; this property, when enforced strictly, solves the temporary update problem and makes cascading rollbacks of transactions unnecessary.
- *Durability or permanency*: Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.

1.4 Characterizing Schedules based on Recoverability

Transaction schedule or history:

- When transactions are executing concurrently in an interleaved fashion, the order of execution of operations from the various transactions forms what is known as a transaction schedule (or history).
- A schedule (or history) S of n transactions $T_1, T_2, \ldots, T_n$:
  - It is an ordering of the operations of the transactions subject to the constraint that, for each transaction T_i that participates in S, the operations of T_1 in S must appear in the same order in which they occur in T_1.
  - Note, however, that operations from other transactions T_j can be interleaved with the operations of T_i in S.

Notation:

- For the purpose of recovery and concurrency control, *read_item*, *write_item*, *commit* and *abort* oprations are considered.
- The following notation is used.

r	for read_item
w	for write_item
c	for commit
a	for abort

Example: a - Consider Figure 2

- $T_1 : r_1(X); w_1(X); r_1(Y); w_1(Y);$
- $T_2 : r_2(X); w_2(X);$
- $S_a : r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y);$

Example: b - Consider Figure 3

- $S_b : r_1(X); w_1(X); r_2(X); w_2(X); r_1(Y); a_1$

Conflicting operations

- Two operations in a schedule are said to be *conflict* if they satisfy all three of the following conditions:
  - they belong to *different transactions*
  - they access the *same item X*
  - at least one of the operations is a *write_item(X)*
- Example: consider schedule S_a
  - $r_1(X)$ $w_2(X)$, $r_2(X)$ $w_1(X)$, $w_1(X)$ $w_2(X)$ conflict operations
  - $r_1(X)$ $r_2(X)$, $w_2(X)$ $w_1(Y)$ do not conflict

Schedules classified on recoverability:

- Recoverable schedule:
  - One where no transaction needs to be rolled back.
  - A schedule S is recoverable if no transaction T in S commits until all transactions $T^{'}$ that have written an item that T reads have committed.
- Cascadeless schedule:
  - One where every transaction reads only the items that are written by committed transactions.

- Schedules requiring cascaded rollback:
  - A schedule in which uncommitted transactions that read an item from a failed transaction must be rolled back.
- Strict Schedules:
  - A schedule in which a transaction can neither read or write an item X until the last transaction that wrote X has committed.

Examples

- $S_a^{'} : r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); c_2; w_1(Y); c_1$
  - $S_a^{'}$ is recoverable, even though it suffers from the lost update problem.
- $S_c : r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); c_2; a_1;$
  - S_c is not recoverable because T_2 reads item X from T_1, and then T_2 commits before T_1 commits.
- $S_d : r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); w_1(Y); c_1; c_2;$
  - For the S_c schedule to be recoverable, the c_2 operation must be postponed until after T_1 commits as shown in S_d.
- $S_e : r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); w_1(Y); a_1; a_2;$
  - If T_1 aborts instead of commiting, then T_2 should also abort as shown in S_e.

1.5 Characterizing Schedules based on Serializability

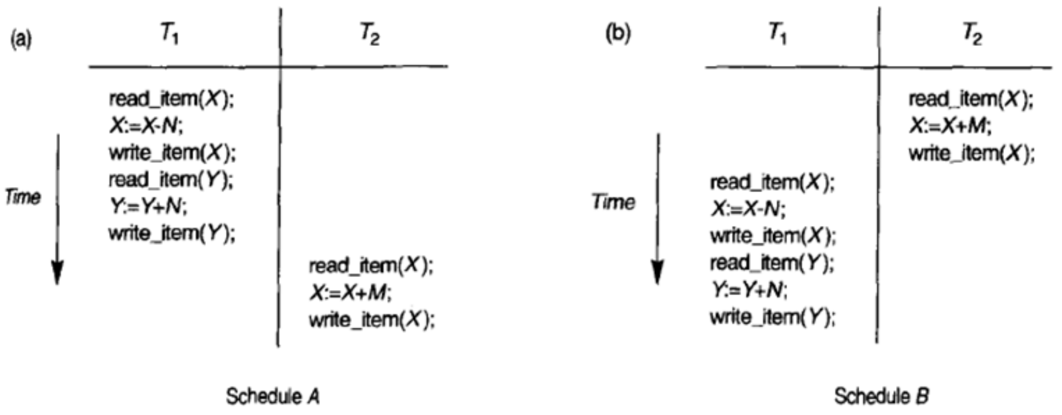
- Serial schedule:
  - A schedule S is serial if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule.
  - Otherwise, the schedule is called nonserial schedule.
- Serializable schedule:
  - A schedule S is serializable if it is equivalent to some serial schedule of the same n transactions.
- Result equivalent:
  - Two schedules are called result equivalent if they produce the same final state of the database.
- Conflict equivalent:
  - Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules.

- Conflict serializable:
  - A schedule S is said to be conflict serializable if it is conflict equivalent to some serial schedule S'.
- Being serializable is not the same as being serial
- Being serializable implies that the schedule is a correct schedule.
- It will leave the database in a consistent state.
- The interleaving is appropriate and will result in a state as if the transactions were serially executed, yet will achieve efficiency due to concurrent execution.
- Serializability is hard to check.
  - Interleaving of operations occurs in an operating system through some scheduler
  - Difficult to determine beforehand how the operations in a schedule will be interleaved.

Testing for conflict serializability:

- Algorithm:
  - 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 T_i to T_j if one of the operations in T_i appears before a conflicting operation in T_j
  - The schedule is serializable if and only if the precedence graph has no cycles.

Constructing the Precedence Graphs



(a) Schedule *A*

T_1	T_2

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);

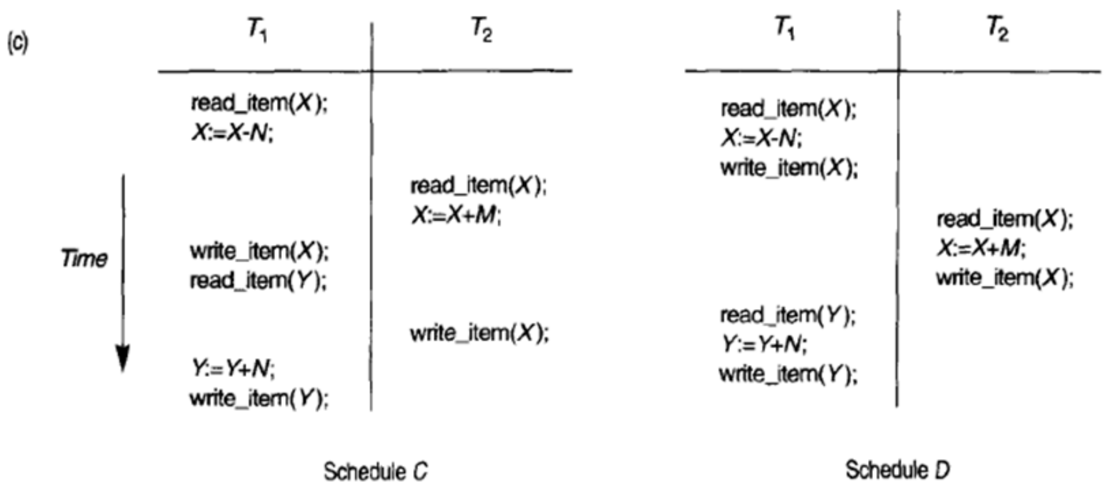
(b) Schedule *B*

T_1	T_2

	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);	

Figure 6: Examples of 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)

Time

T_1	T_2
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);	

Schedule C

T_1	T_2

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);	

Schedule D

Figure 7: Examples of schedules involving transactions T_1 and T_2 (c): Two nonserial schedules C and D with interleaving of operations

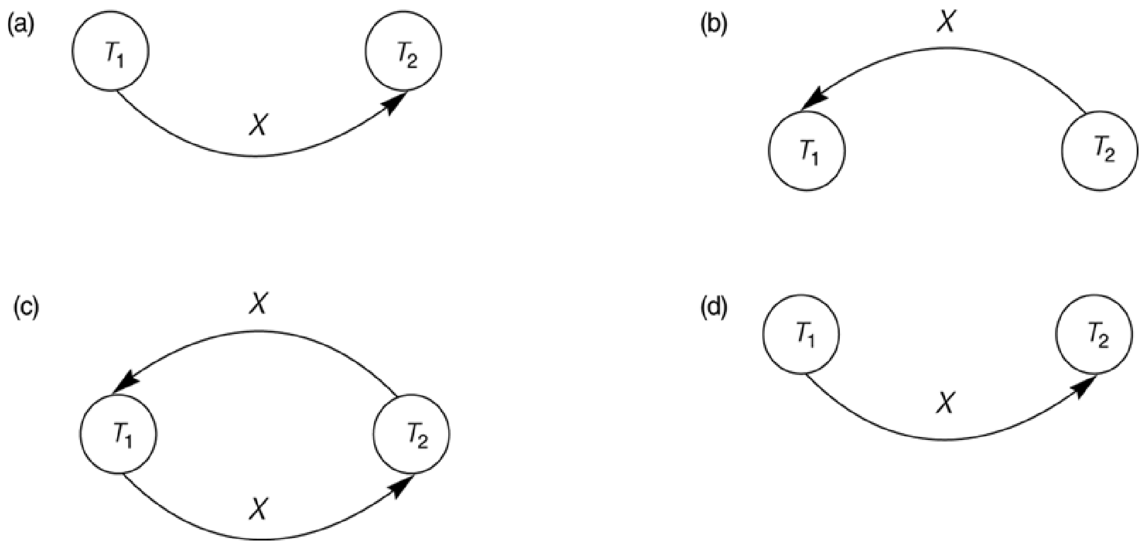


Figure 8: Constructing the precedence graphs for schedules A and D from Figure 17.5 to test for conflict serializability