Database Systems II

Lecture 9

Transaction Processing

Introduction to Transaction Processing

- **Transaction:** A process that includes one or more database access operations
  - Read operations (database retrieval, such as SQL SELECT)
  - Write operations (modify database, such as SQL INSERT, UPDATE, DELETE)
- A transaction may appear to the user as a single step while it encompasses several steps
- Example: Bank balance transfer of $100 dollars from one account to another account in a BANK database

Introduction to Transaction Processing

- A transaction (set of operations) may be:
  - stand-alone, specified in a high-level language like SQL submitted interactively, or
  - consists of database operations embedded within a program (most transactions)
- **Transaction boundaries**: Begin and End transaction.
  - Note: An **application program** may contain several transactions separated by Begin and End transaction boundaries

Introduction to Transaction Processing

- **Transaction Processing Systems:** Large multi-user database systems supporting thousands of *concurrent transactions* per minute
- **Two Modes of Concurrency**
  - **Interleaved processing**: concurrent execution of processes is interleaved in a *single CPU*
  - **Parallel processing**: processes are concurrently executed in *multiple CPUs*
  - Basic transaction processing theory assumes interleaved concurrency

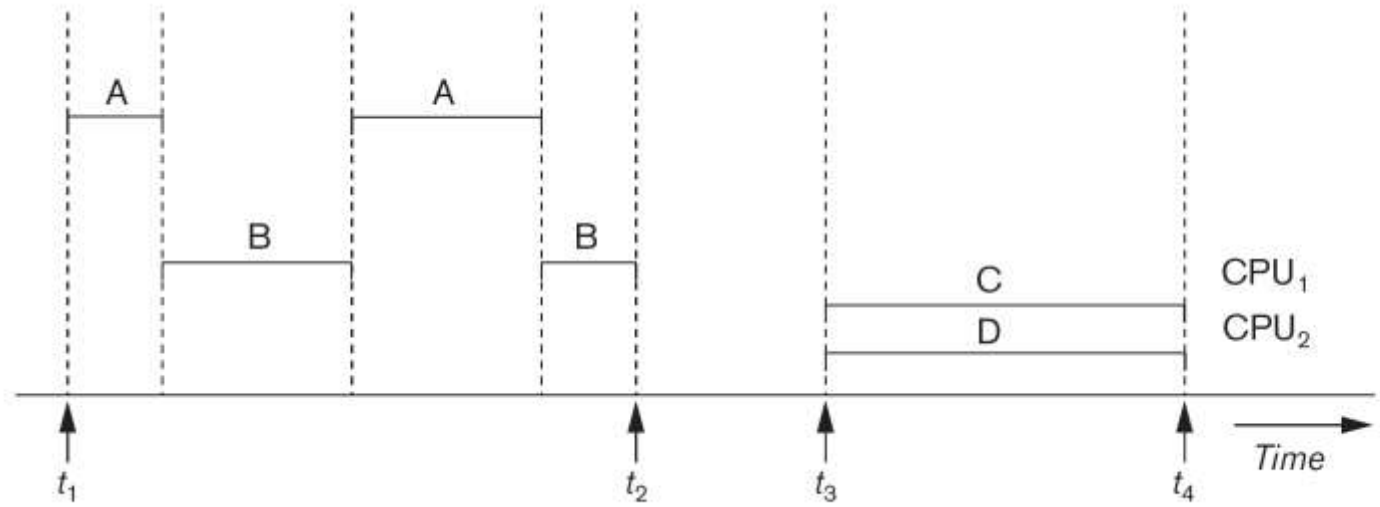


Figure 21.1 Interleaved processing versus parallel processing of concurrent transactions.

Interleaved Processing Parallel Processing

Read and Write Operations

- A transaction is called *read only transaction* if operations included in the transaction don't update the database
- A transaction is called *read/write transaction* otherwise
- Basic operations on an item X:
  - **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.

Transaction Example

(a)

T_1

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

(b)

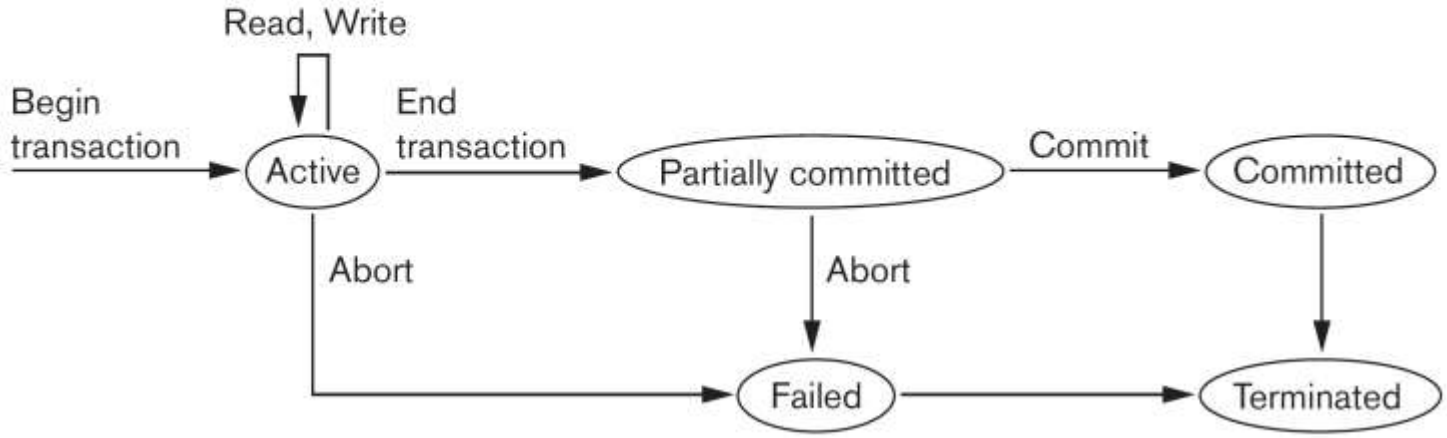
T_2

read_item(X);
$X := X + M$;
write_item(X);

Figure 21.2
Two sample transactions. (a) Transaction T_1. (b) Transaction T_2.

Transaction States

Figure 21.4
State transition diagram illustrating the states for transaction execution.



Transaction States

- **Begin transaction:** This marks the beginning of transaction execution
- **Read/write:** These specify read or write operations on the database items that are executed as part of a transaction
- **End transaction:** This marks the end of transaction execution. At this point it is important to check whether to permanently save changes to database or not
- **Commit transaction:** This signals successful end of the transaction so any changes can be safely saved to the database
- **Rollback (abort):**This signals that the transaction has ended unsuccessfully and any changes to the database must be undone

Transaction States

Transaction passes through several states:

- Active state: after the execution of transaction where it execute read, write operations
- Partially committed state: At this point, the transaction has completed its execution, but it is still possible that it may have to be aborted, since the actual output may still be temporarily residing in main memory, and thus a hardware failure may prevent its successful completion
- Committed state: When a transaction is committed, it has concluded its execution successfully and all its changes must be recorded permanently in the database

Transaction States

- Failed state: A transaction enters the failed state after the system determines that the transaction can no longer proceed with its normal execution (for example, because of hardware or logical errors). Transaction will be rolled back
- Terminated State: Transaction leaves system

The System Log File

- To be able to recover from failures that affect transactions, the system maintains a log to keep track of all transaction operations that affect the values of database items, as well as other transaction information that may be needed to permit recovery from failures.

- It is an append-only file to keep track of all operations of all transactions *in the order in which they occurred*. This information is needed during recovery from failures

- Log is kept on disk - not affected except for disk or catastrophic failure

The System Log File

Types of records (entries) in log file:

- [start_transaction,T]: Records that transaction T has started execution.
- [write_item,T,X,old_value,new_value]: T has changed the value of item X from old_value to new_value.
- [read_item,T,X]: T has read the value of item X (not needed in many cases).
- [end_transaction,T]: T has ended execution
- [commit,T]: T has completed successfully, and committed.
- [abort,T]: T has been aborted.

Commit Point of a Transaction

Definition:

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 file (on disk). The transaction is then said to be **committed**.

Desirable Properties of Transactions

ALL transactions in RDBs follow the ACID Properties

Atomicity, Consistency, Isolation, Durability

- **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.

Desirable Properties of Transactions

- **Isolation**: Even though transactions are executing concurrently, they should appear to be executed in isolation – that is, their final effect should be as if each transaction was executed in isolation from start to finish.

- **Durability or Permanency**: Once a transaction is committed, its changes (writes) applied to the database must never be lost because of subsequent failure.

Why we need concurrency control

Without Concurrency Control, problems may occur with concurrent transactions:

- **Lost Update Problem:**

  Occurs when two transactions update the same data item, but both read the same original value before update.

- **The Temporary Update (or Dirty Read) Problem:**

  This occurs when one transaction T1 updates a database item X, which is accessed (read) by another transaction T2; then T1 fails for some reason.

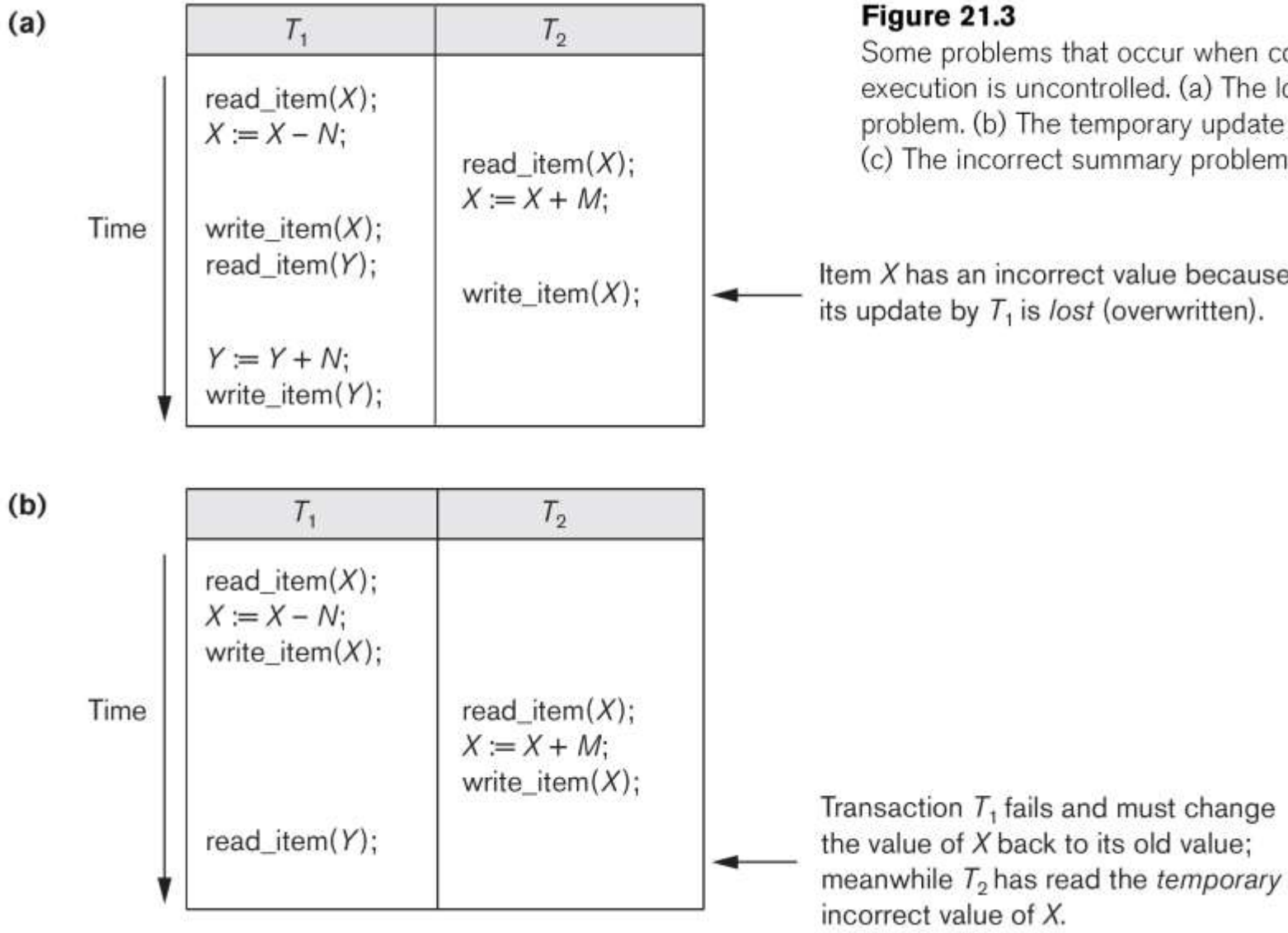


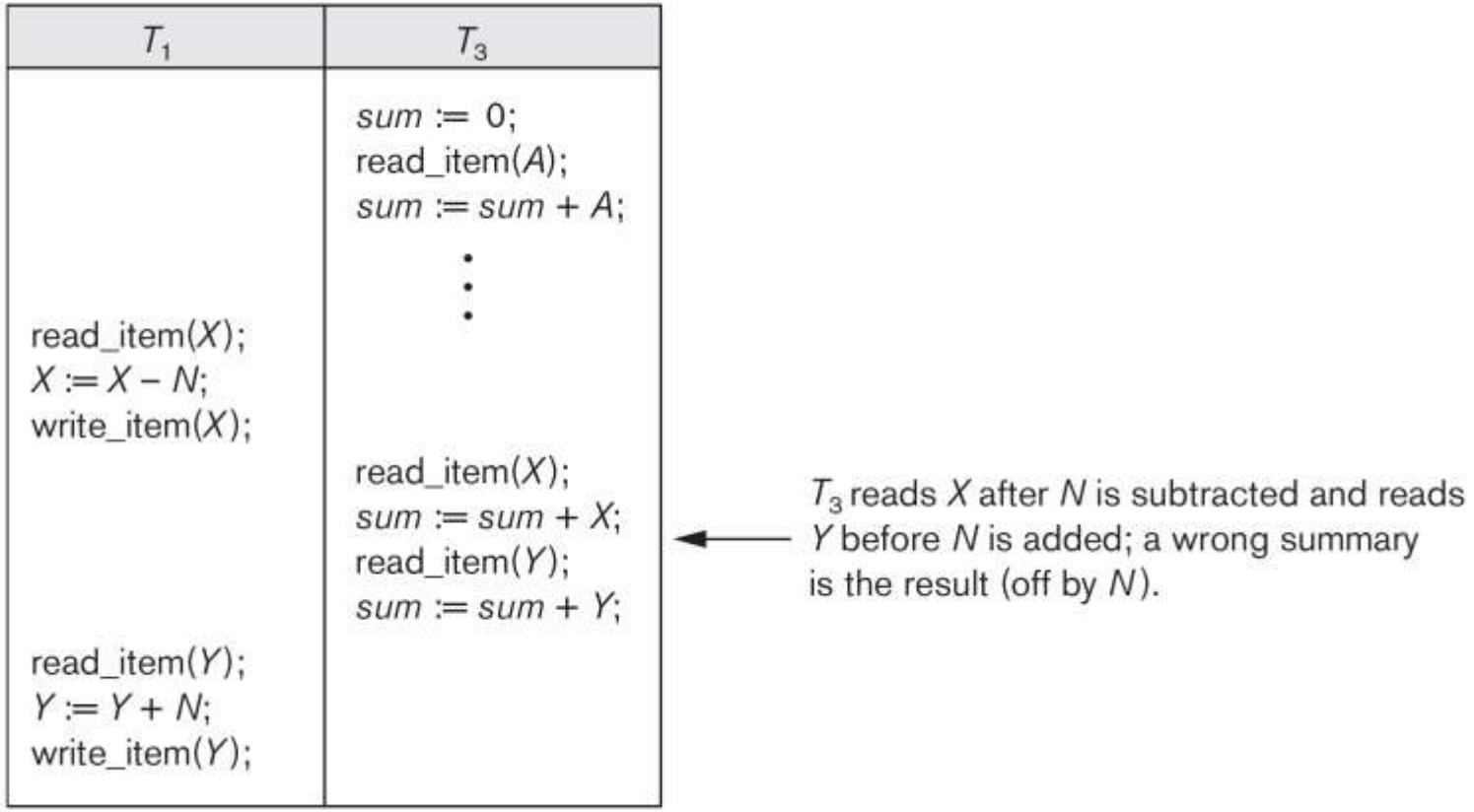
Figure 21.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

Why we need concurrency control

- **The Incorrect Summary Problem:**

  One transaction is calculating an aggregate summary function on a number of records (for example, sum (total) of all bank account balances) while other transactions are updating some of these records (for example, transferring a large amount between two accounts); the aggregate function may read some values before they are updated and others after they are updated.

(c)



Why we need concurrency control

- **The Unrepeatable Read Problem:**

  A transaction T1 may read an item (say, available seats on a flight); later, T1 may read the same item again and get a different value because another transaction T2 has updated the item (reserved seats on the flight) between the two reads by T1

Why recovery is needed

A computer failure (system crash):

- A hardware or software error occurs during transaction execution.
- If the hardware crashes, the contents of the computer's internal main 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, erroneous parameter values, or a logical programming error.
- In addition, the user may interrupt the transaction during its execution.

Local errors or exception conditions:

- Certain conditions necessitate cancellation of the transaction. For example, data for the transaction may not be found.

Why recovery is needed (cont.)

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 because of a disk read/write head crash. This kind of failure and item 6 are more severe than items 1 through 4.

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.