COS221 - L31 - Transaction processing 2

Linda Marshall

19 May 2023

Transactions and System Concepts

- Remember, a transaction is an atomic unit of work. It is either to be completed in its entirety or not at all.
- The recovery manager of a transaction therefore needs to keep tabs on when a transaction starts, terminates, and commits or aborts.
- The DBMS there keeps track of the following operations:

```
BEGIN_TRANSACTION
READ or WRITE
END_TRANSACTION
COMMIT_TRANSACTION
ROLLBACK (or ABORT)
```

- A transaction that goes into the failed state will need to undo any WRITE operations to the database.

Transactions and System Concepts

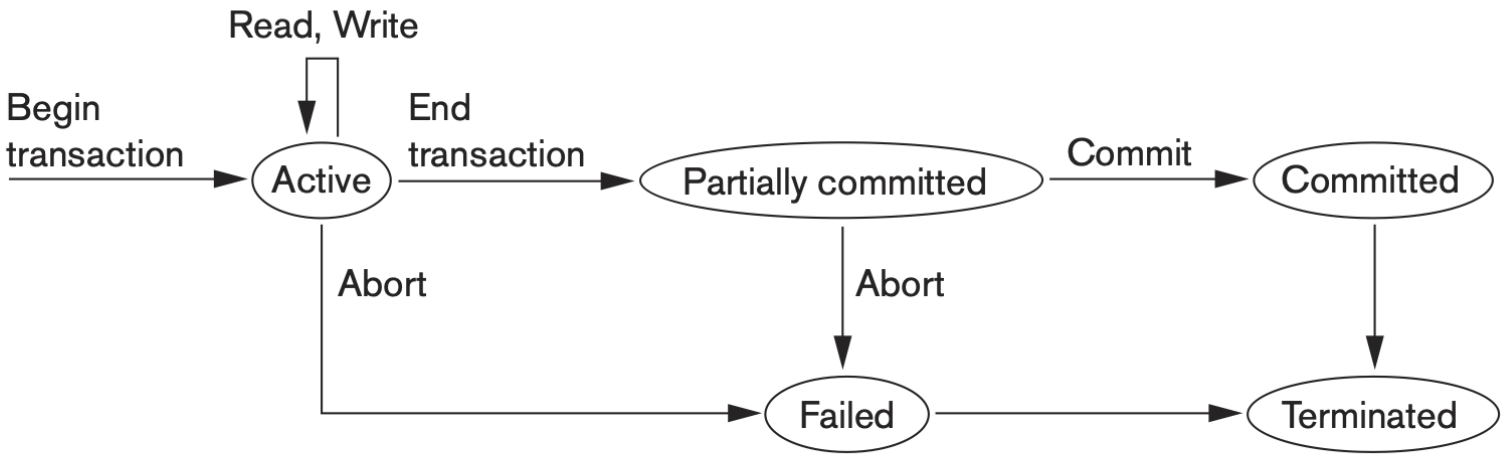


Figure 20.4
State transition diagram illustrating the states for transaction execution.

Transactions and System Concepts

The effects of all transactions are kept in a system log. Each transaction has a unique id, T. The operations are therefore each linked to this unique id.

1. [start_transaction, T]. Indicates that transaction T has started execution.
2. [write_item, T, X, old_value, new_value]. Indicates that transaction T has changed the value of database item X from old_value to new_value.
3. [read_item, T, X]. Indicates that transaction T has read the value of database item X.
4. [commit,T]. Indicates that the transaction completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
5. [abort, T]. Indicates that transaction T has been aborted.

Transactions and System Concepts

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 (ACID properties)

- **Atomicity**
  Enforced by the recovery protocol. A transaction must execute to completion.
- **Consistency preservation**
  Specifies that each transaction does a correct action on the database on its own. Application programmers and DBMS constraint enforcement are responsible for this.
- **Isolation**
  Responsibility of the concurrency control protocol. Attempts have been made to define the level of isolation - from 0 to 3. Other isolation type is called snapshot isolation.
- **Durability or permanency**
  Enforced by the recovery protocol.

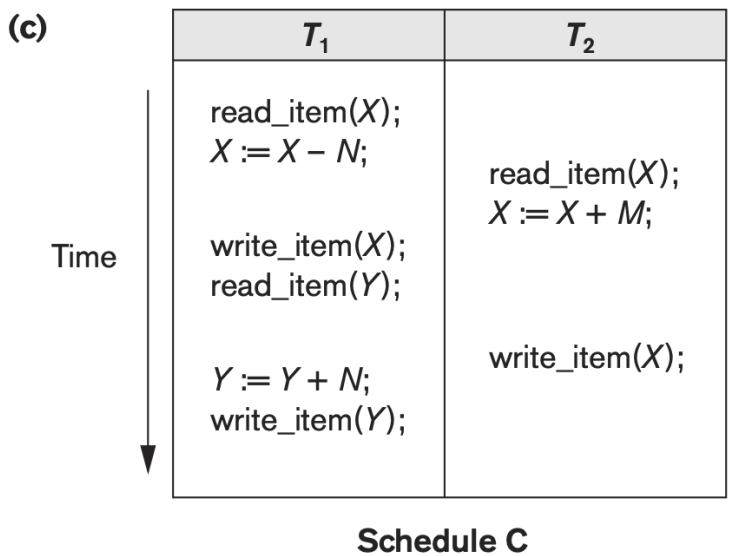
Characterising Schedules

A schedule (order of operations of a transaction) can be characterised by:

- Recoverability
- Serialisability

Characterising Schedules Based on Recoverability

- The order of the operations of transactions executing concurrently in an interleaved fashion is known as a schedule (or history), S. S of n transactions T_1, T_2, ... , T_n is an ordering of the operations of the transactions.
- The order of the operations of S is a *total ordering* if for any two operations in the schedule, one must occur before the other.
- The schedule for the transactions below is given by:
  S_C: $r_1(X)$; $r_2(X)$; $w_1(X)$; $r_1(Y)$; $w_2(X)$; $w_1(Y)$;



Schedule C

Characterising Schedules Based on Recoverability

Two operations are said to conflict if they satisfy the following three conditions:

- they belong to different transactions
- they access the same item (X)
- at least one of the operations is a write_item(X)

For example, in S_C: $r_1(X)$; $r_2(X)$; $w_1(X)$; $r_1(Y)$; $w_2(X)$; $w_1(Y)$; , the operations $r_1(X)$ and $w_2(X)$ are in conflict as are $r_2(X)$ and $w_1(X)$, and $w_1(X)$ and $w_2(X)$. Operations $r_1(X)$ and $r_2(X)$ do not conflict - they are both read operations. $w_2(X)$ and $w_1(Y)$ do not conflict because they operate on distinct data items X and Y.

Generally speaking, two operations are conflicting if changing the order results in different outcomes.

Characterising Schedules Based on Recoverability

A schedule S of n transactions, $T_1, T_2, \ldots, T_n$, is a complete schedule if:

1. The operations in S are exactly those operations in $T_1, T_2, \ldots, T_n$, including a commit or abort operation as the last operation for each transaction in the schedule.
2. For any pair of operations from the same transaction T_i, their relative order of appearance in S is the same as their order of appearance in T_i.
3. For any two conflicting operations, one of the two must occur before the other in the schedule.

A committed projection, C(S), of S includes only the operations in S that belong to transactions T_i whose commit operation C_i is in S.

Characterising Schedules Based on Recoverability

- For some schedules it is easy to recover from failures (either transaction or system). To move towards an algorithm for recovery, characterisation of schedules is necessary. Characteristics include:
  1. Once a transaction T is committed it should never be rolled back thereby ensuring the durability property of transactions. Schedules that meet this criteria are referred to as recoverable schedules.
  2. Cascadeless (avoid cascading rollback)
  3. Strict schedule
- There is no order specified for the operations. The schedule is therefore seen as a *partial order*.
- A *total order* must be specified for any pair of conflicting operations (condition 3) and any pair of operations from the same transaction (condition 2).

Characterising Schedules Based on Serialisability

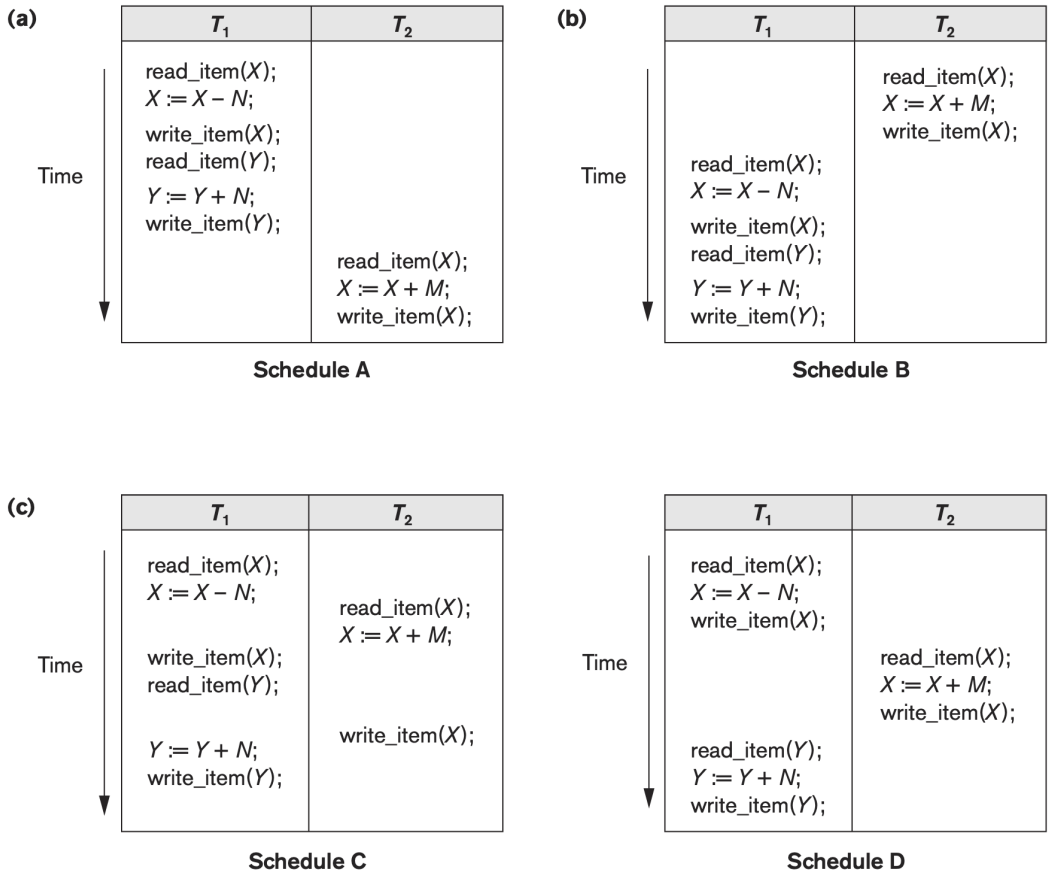
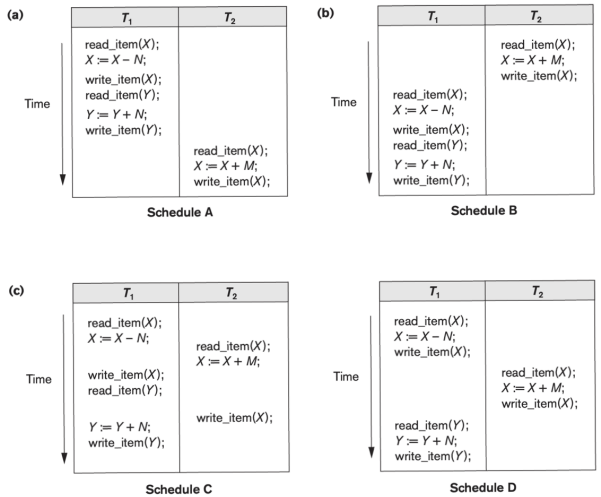


Figure 20.5
Examples of serial and nonserial schedules involving transactions T_1 and T_2. (a) Serial schedule A: T_1 followed by T_2. (b) Serial schedule B: T_2 followed by T_1. (c) Two nonserial schedules C and D with interleaving of operations.

Characterising Schedules Based on Serialisability



- Schedules in Figure 20.5 in Schedule Notation
  - Schedule A: $r_1(X)$; $w_1(X)$; $r_1(Y)$; $w_1(Y)$; $r_2(X)$; $w_2(x)$;
  - Schedule B: $r_2(X)$; $w_2(X)$; $r_1(X)$; $w_1(X)$; $r_1(Y)$; $w_1(Y)$;
  - Schedule C: $r_1(X)$; $r_2(X)$; $w_1(X)$; $r_1(Y)$; $w_2(X)$; $w_1(Y)$;
  - Schedule D: $r_1(X)$; $w_1(X)$; $r_2(X)$; $w_2(X)$; $r_1(Y)$; $w_1(Y)$;
- Schedule Notation
  - Order of operations from left to right
  - Include only read (r) and write (w) operations, with transaction id (1, 2, ?) and item name (X, Y, ?)
  - Can also include other operations such as b (begin), e (end), c (commit), a (abort)

Characterising Schedules Based on Serialisability

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*.

- Schedules A and B are serial - each operation is executed consecutively and the entire transaction is performed before the next one begins.
- For Schedule A, T_1 is performed and then T_2. For Schedule 2 the order of the transactions is reversed.
- Serial schedules limit concurrency and therefore waste CPU time. If some transactions are long, then other shorter transactions may wait where they could have completed execution.

Characterising Schedules Based on Serialisability

- Schedules C and D are nonserial, because the sequence of operations for T_1 and T_2 are interleaved.
- Schedule C produces an incorrect result due to a lost update .
- Schedule D results in a correct result. Schedules that give the correct results are referred to as *Serialisable* schedules.
- A schedule S of *n* transactions is serialisable if it is equivalent to some serial schedule of the same *n* transactions.

Characterising Schedules Based on Serialisability

- Schedules are said to be **result equivalent** if they produce the same final state of the database.
  Note: Two schedules may accidentally produce the same result. For example, for X = 100.

Figure 20.6
Two schedules that are result equivalent for the initial value of $X = 100$ but are not result equivalent in general.

S_1

read_item(X);
$X := X + 10$;
write_item(X);

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

- Schedules are said to be **conflict equivalent** if the relative order of any two conflicting operations is the same in both schedules.
- A schedule S to be **serialisable** if it is (conflict) equivalent to some serial schedule S'.

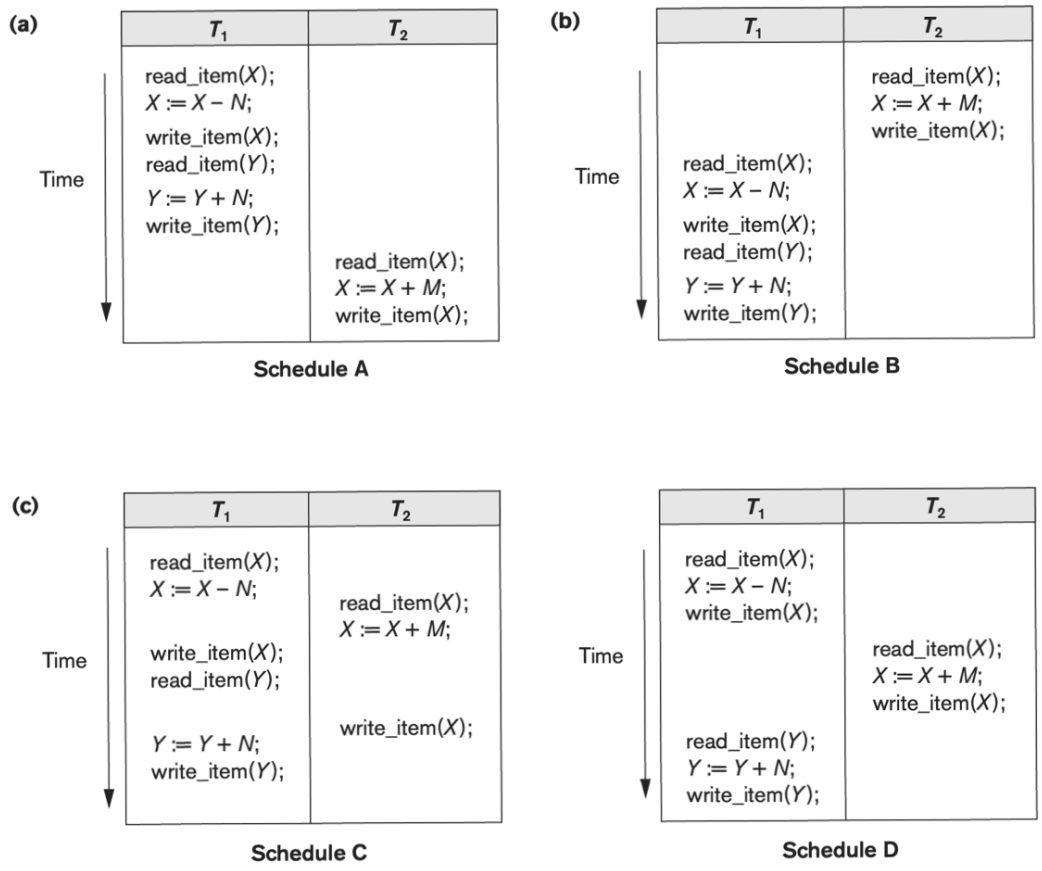
Characterising Schedules Based on Serialisability

- According to the definition of serialisability, Schedule D is equivalent to serial Schedule A.
- Algorithm for determining if two schedules are conflict equivalent or not.

Algorithm 20.1. Testing Conflict Serializability of a Schedule S

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.

Characterising Schedules Based on Serialisability



Characterising Schedules Based on Serialisability

Precedence graph for Schedules A to D:

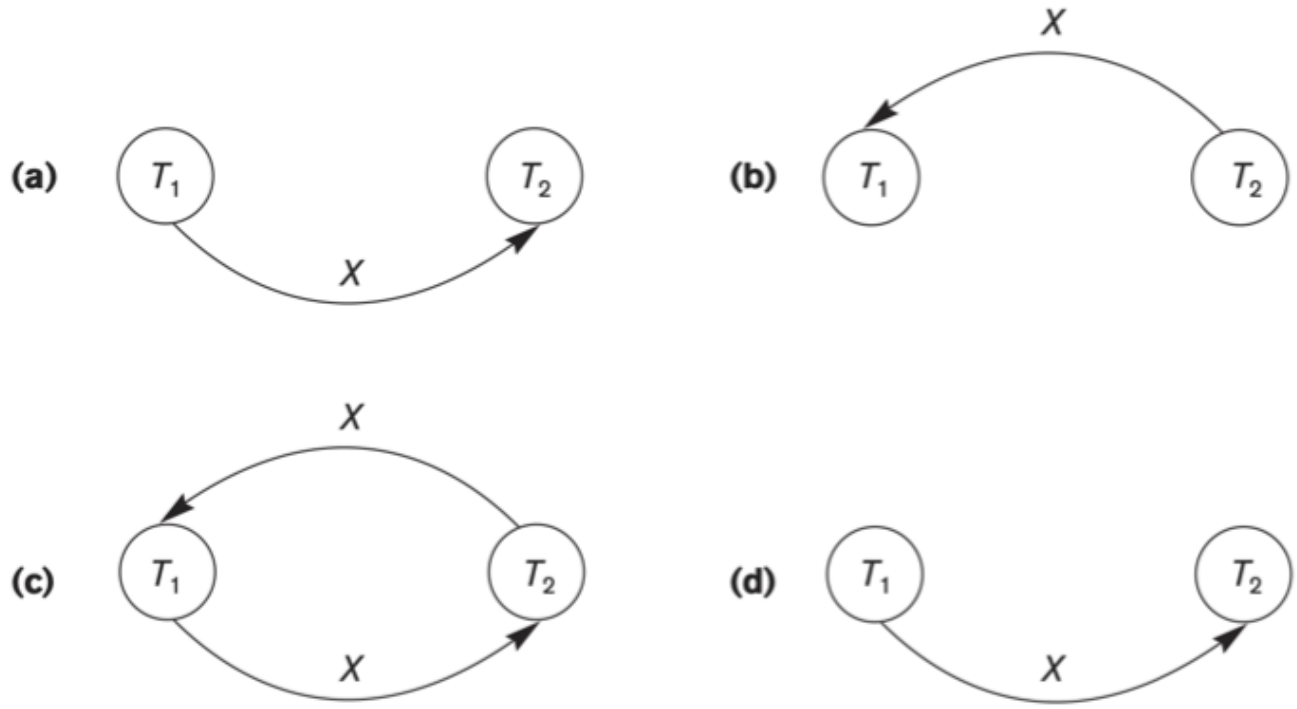


Figure 20.7
Constructing the precedence graphs for schedules A to D from Figure 20.5 to test for conflict serializability. (a) Precedence graph for serial schedule A. (b) Precedence graph for serial schedule B. (c) Precedence graph for schedule C (not serializable). (d) Precedence graph for schedule D (serializable, equivalent to schedule A).

Characterising Schedules Based on Serialisability

- Two schedules S and S' are said to be **view equivalent** if the following three conditions hold:
  1. The same set of transactions participates in S and S', and S and S' include the same operations of those transactions.
  2. For any operation $r_i(X)$ of T_i in S, if the value of X read by the operation has been written by an operation $w_j(X)$ of T_j (or if it is the original value of X before the schedule started), the same condition must hold for the value of X read by operation $r_i(X)$ of T_i in S'.
  3. If the operation $w_k(Y)$ of T_k is the last operation to write item Y in S, then $w_k(Y)$ of T_k must also be the last operation to write item Y in S'.

Characterising Schedules Based on Serialisability

- The premise of view equivalence:
  - Each read operation of a transaction reads the result of *the same write operation* in both schedules
  - The **view**, the read operations are said to see the *same view* in both schedules
  - The final write operation on each operation is the same on both schedules resulting in the same final database states in the case of blind writes.
- That is, the same set of transactions participate in S and S', and S and S' include the same operations of those transactions.

Characterising Schedules Based on Serialisability

- Serializability is generally hard to check at run-time:
  - Difficult to determine beforehand how the operations in a schedule will be interleaved
  - Interleaving of operations is generally handled by the operating system through the process scheduler
  - Transactions are continuously started and terminated
- Practical approach:
  - Come up with methods (concurrency control protocols) to ensure serialisability
  - DBMS concurrency control subsystem will enforce the protocol rules and thus guarantee serialisability of schedules
- Current approach used in most DBMSs - make use of locks with two phase locking