Mansoura University
Faculty of Computers and Information
Department of Information System

[IS313P] Database System II

Grade: 3 rd. IS & IT

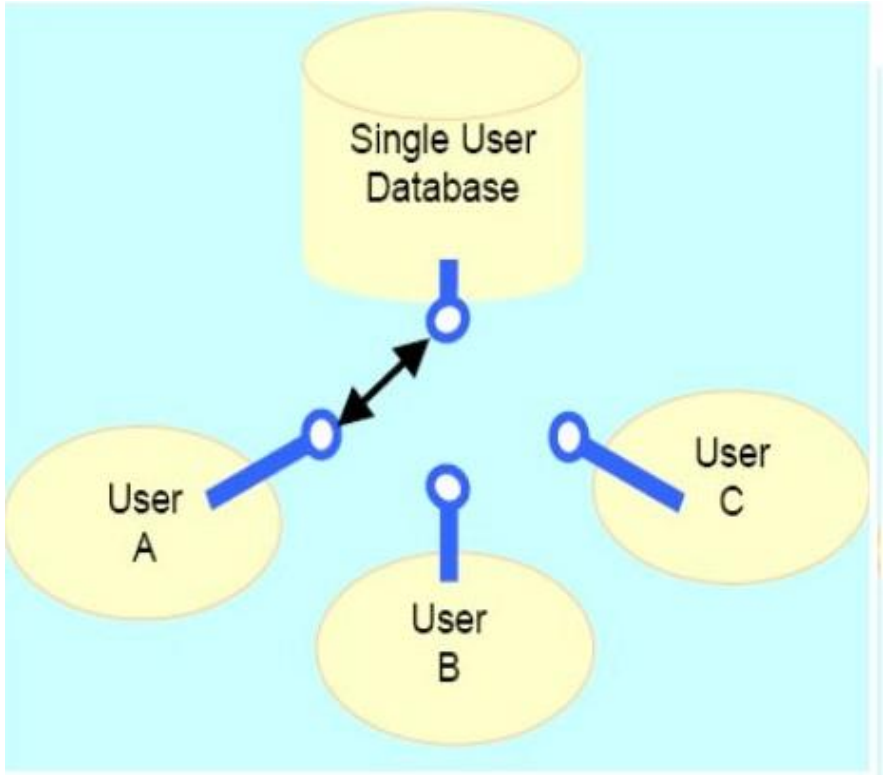
Dr. Amira Rezk

TRANSACTION PROCESSING

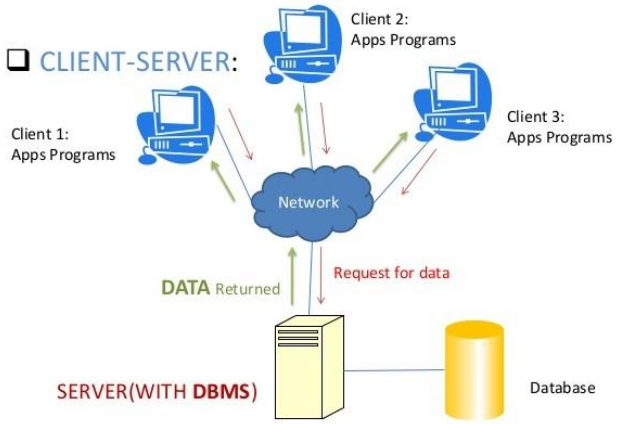
AGENDA

- Introduction to Transaction Processing
- Transaction and System Concepts
- Desirable Properties of Transactions
- Characterizing Schedules Based on Recoverability
- Characterizing Schedules Based on Serializability

INTRODUCTION TO TRANSACTION PROCESSING SINGLE USER VS. MULTIUSER SYSTEMS

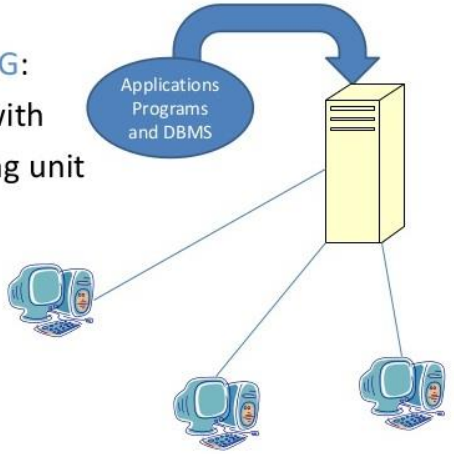


❑ CLIENT-SERVER:

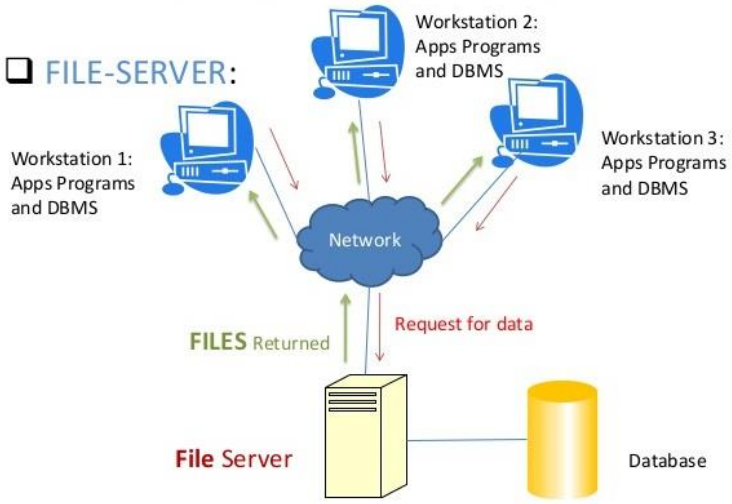


❑ TELEPROCESSING:

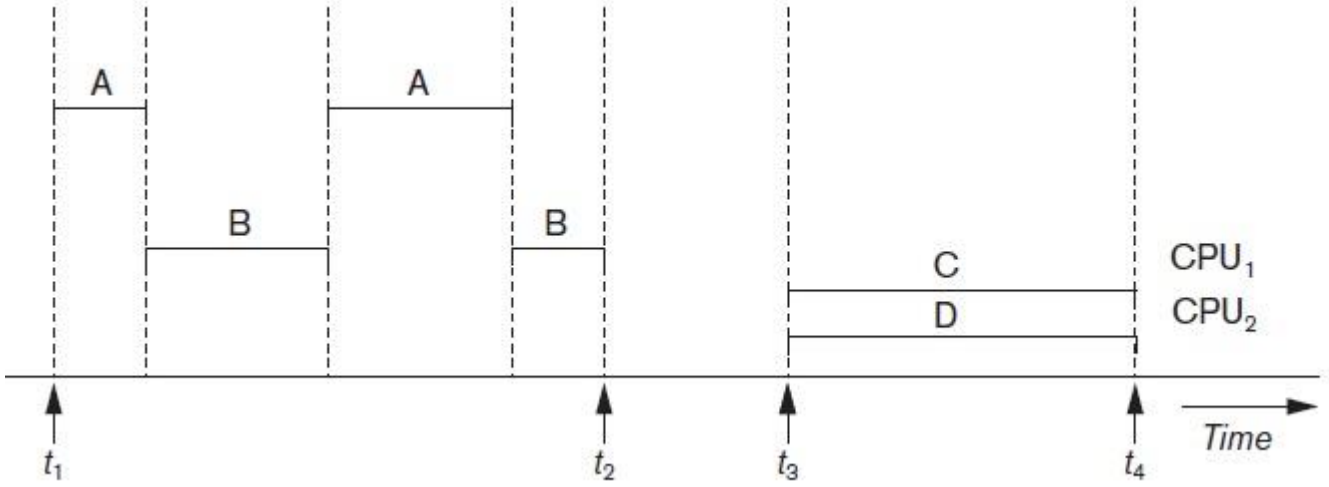
- One Computer with central processing unit
- N Terminals



❑ FILE-SERVER:



INTRODUCTION … CONT.
INTERLEAVED VS. PARALLEL PROCESSING



Interleaved processing versus parallel processing of concurrent transactions.

INTRODUCTION … CONT.

- A transaction is a logical unit of database processing that includes one or more database access operations (read : retrieval or write: insertion, deletion, modification)
- The transaction boundaries
  - begin transaction and end transaction
- An application program may contain several transactions separated by the transaction boundaries.
- The basic database access operations
  - read_item(X). Reads a database item named X into a program variable.
  - write_item(X). Writes the value of program variable X into the database item named X.

TWO SAMPLE TRANSACTIONS

- Two sample transactions.
- (a) Transaction T1. (b) Transaction T2.

(a)

T_1

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

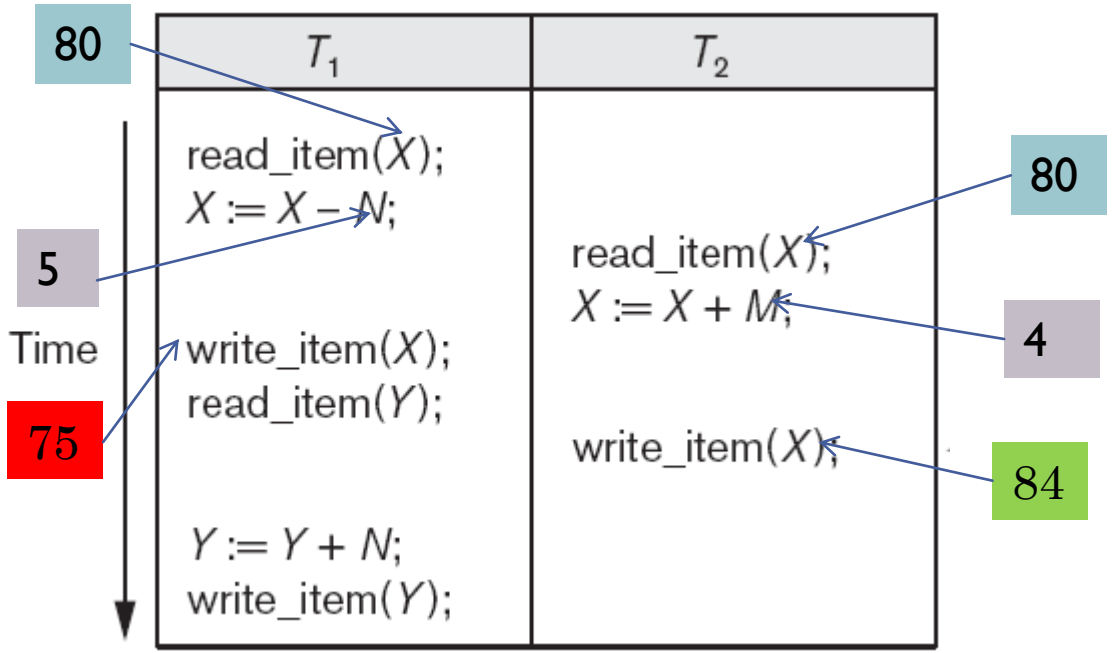
(b)

T_2

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

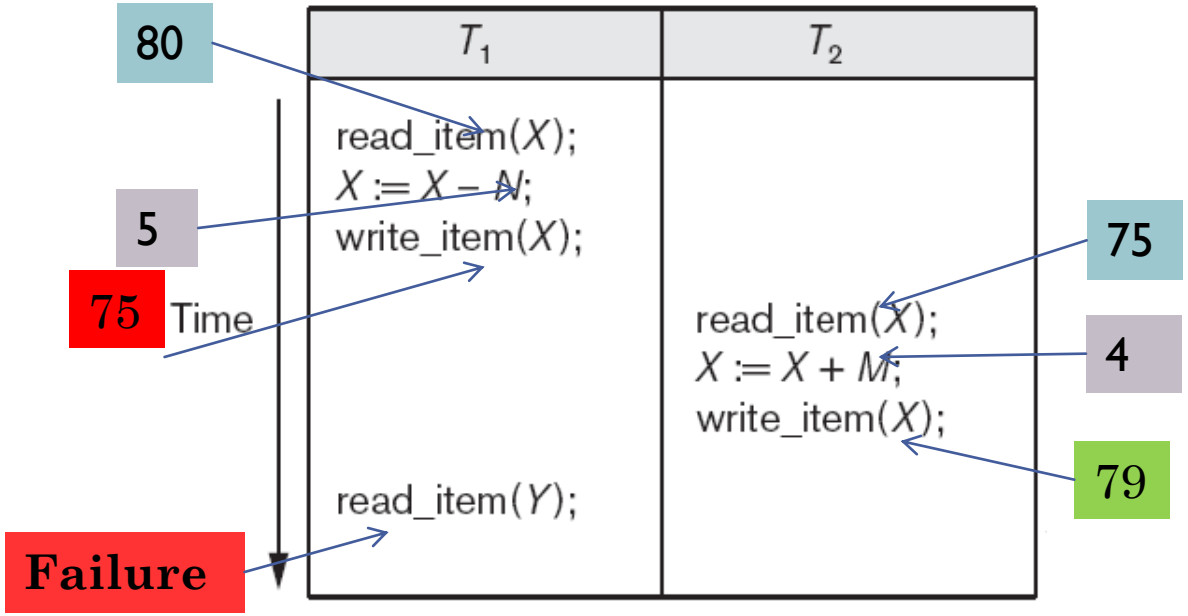
WHY CONCURRENCY CONTROL IS NEEDED

- The Lost Update Problem. This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect.



WHY CONCURRENCY CONTROL IS NEEDED… CONT.

- The Temporary Update (or Dirty Read) Problem.
- This problem occurs when one transaction updates a database item and then the transaction fails for some reason. Meanwhile, the updated item is accessed (read) by another transaction before it is changed back to its original value.



WHY CONCURRENCY CONTROL IS NEEDED… CONT.

- The Incorrect Summary Problem. 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.

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*);	

WHY CONCURRENCY CONTROL IS NEEDED… CONT.

- The Unrepeatable Read Problem.
- Another problem that may occur is called unrepeatable read, where a transaction T reads the same item twice and the item is changed by another transaction T between the two reads. Hence, T receives different values for its two reads of the same item.

QUESTION ?

- The schedule S: r1(x), w1(x), r2 (x), w2(x), A1, C2 causes …….. problem

a-lost update b- dirty read

c-incorrect summary d- Unrepeatable read

- True or false
- The schedule S: r1(x), w1(x), r2 (x), w2(x), A1, C2 causes dirty read problem

WHY RECOVERY IS NEEDED

- The system is responsible for making sure that
  - all the operations in the transaction are completed successfully and their effect is recorded permanently in the database, (committed)
  - The transaction does not have any effect on the database or any other transactions. (aborted)

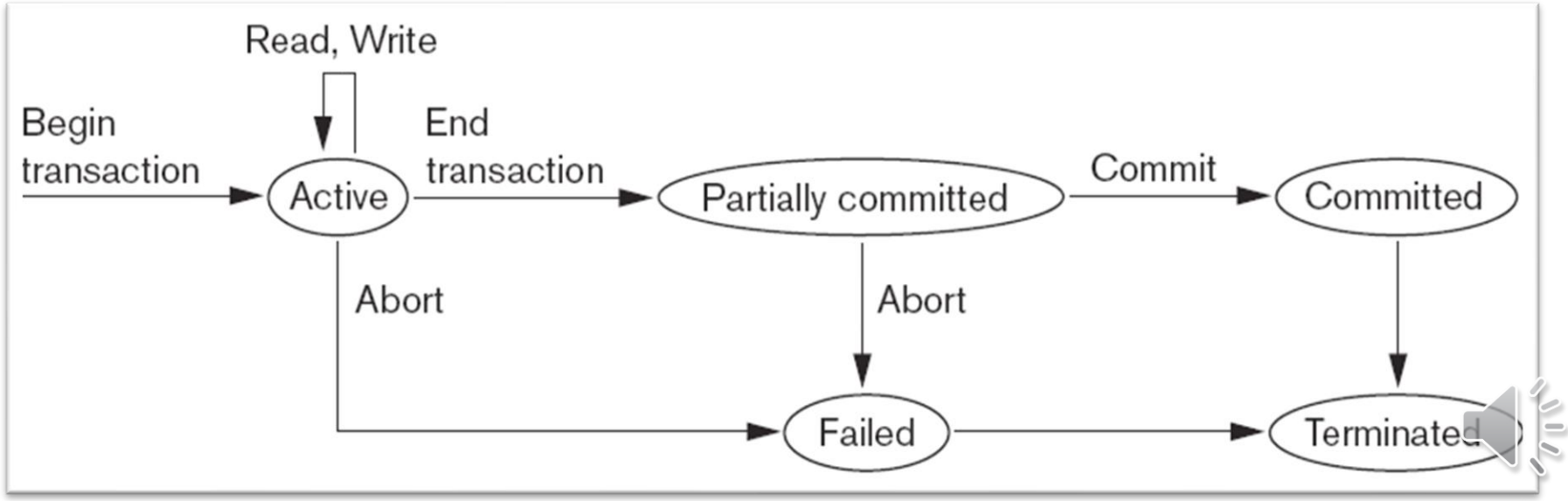
WHY RECOVERY IS NEEDED … CONT.

- What causes a Transaction to fail)
  - 1. A computer failure (system crash):
    - If the hardware crashes, the contents of the computer's internal memory may be lost.
  - 2. A transaction or system error:
    - Transaction operation error (integer overflow or division by zero).
    - erroneous parameter values or a logical programming error.
    - The user may interrupt the transaction during its execution.
  - 3. Local errors or exception conditions detected by the Trans. :
    - Ex. data for the transaction may not be found.
    - A programmed abort in the transaction causes it to fail.
  - 4. Concurrency control enforcement:
    - The concurrency control method may decide to abort the transaction, to be restarted later
  - 5. Disk failure:
    - Ex. disk read/write head crash.
  - 6. Physical problems and catastrophes:
    - Ex. power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, … etc

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:
  - Active state
  - Partially committed state
  - Committed state
  - Failed state
  - Terminated State

State transition diagram



TRANSACTION AND SYSTEM CONCEPTS CONT…

- 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.

TRANSACTION AND SYSTEM CONCEPTS. CONT…

- Recovery manager keeps track of the following(cont):
  - 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.

TRANSACTION AND SYSTEM CONCEPTS CONT…

- The System Log
  - 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 refers to a unique transaction-id that is generated automatically by the system and is used to identify each transaction:

TRANSACTION AND SYSTEM CONCEPTS CONT…

- 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.

TRANSACTION AND SYSTEM CONCEPTS CONT…

- 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.

DESIRABLE PROPERTIES OF TRANSACTIONS

- ACID properties:
- **A**tomicity: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
- **C**onsistency preservation: A correct execution of the transaction must take the database from one consistent state to another.
- **I**solation: 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
- **D**urability or permanency: Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.

QUESTION2

- ACID properties refer to all the following except …..

a-Atomicity b- Consistency

c- Integrity d- Durability

- True or false

Properties of transactions (ACID) refer to Automated, Correct, Integrated, Defined.

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 T1, T2, …, Tn:
  - It is an ordering of the operations of the transactions subject to the constraint that, for each transaction Ti that participates in S, the operations of T1 in S must appear in the same order in which they occur in T1.
  - Note, however, that operations from other transactions Tj can be interleaved with the operations of Ti in S.

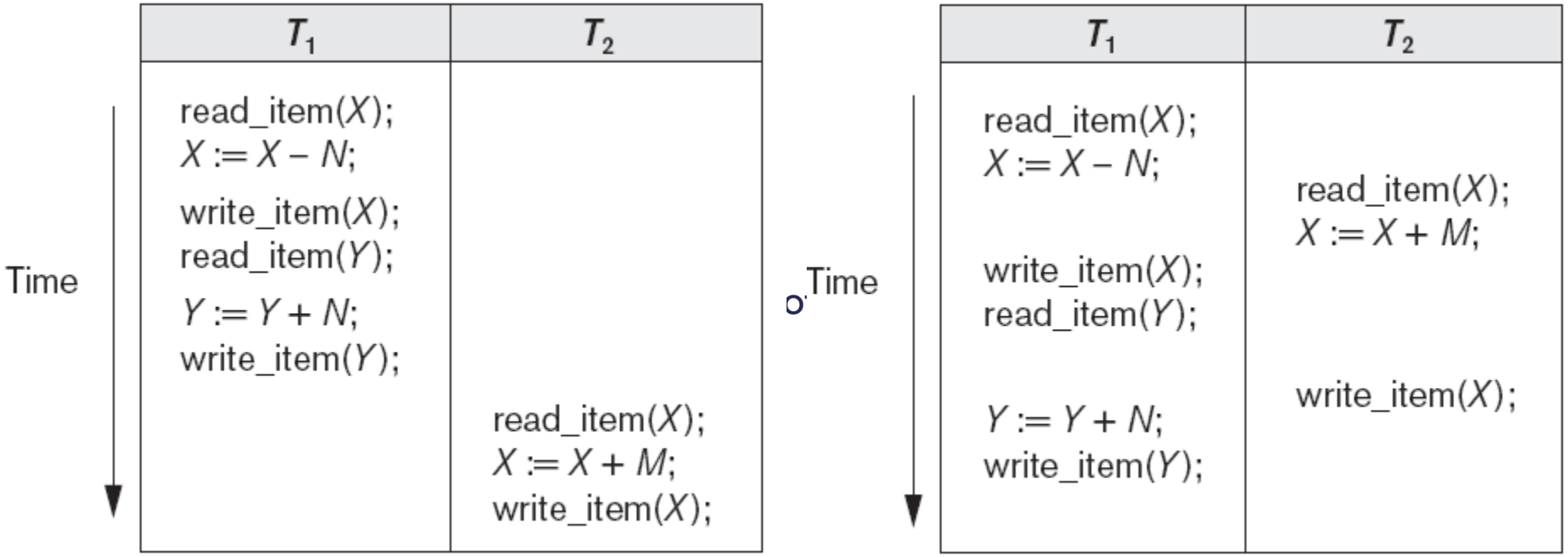
CHARACTERIZING SCHEDULES BASED ON RECOVERABILITY

- 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.

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.

EXAMPLES OF SERIAL AND NONSERIAL SCHEDULES



Schedule A

Schedule C

CHARACTERIZING SCHEDULES BASED ON SERIALIZABILITY

- 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'.

CHARACTERIZING SCHEDULES BASED ON SERIALIZABILITY

- 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.

CHARACTERIZING SCHEDULES BASED ON SERIALIZABILITY

- Practical approach:
- Come up with methods (protocols) to ensure serializability.
- It's not possible to determine when a schedule begins and when it ends.
  - Hence, we reduce the problem of checking the whole schedule to checking only a committed project of the schedule (i.e. operations from only the committed transactions.)
- Current approach used in most DBMSs:
  - Use of locks with two phase locking

CHARACTERIZING SCHEDULES BASED ON SERIALIZABILITY

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

- **Conflict occurs if two operations satisfy with**
  1. They belong to different transactions
  2. They access the same item X
  3. At least one of the operation is **write_item(X)**

r2(X) and w1(X): conflict r1(X) and r2(X): Not conflict

w2(X) and w1(Y): Not conflict r1(X) and w1(X): Not conflict

EXAMPLE OF SERIALIZABILITY TESTING

The read and write operations of three transactions $T1$, $T2$, and $T3$.

Transaction T_1
read_item(X);
write_item(X);
read_item(Y);
write_item(Y);

Transaction T_2
read_item(Z);
read_item(Y);
write_item(Y);
read_item(X);
write_item(X);

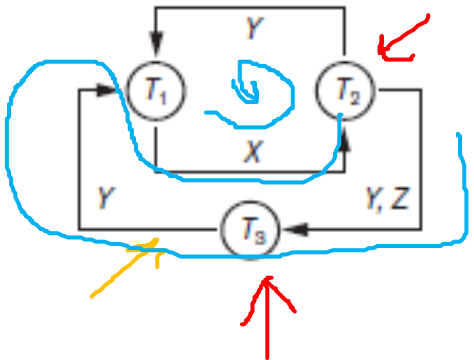
Transaction T_3
read_item(Y);
read_item(Z);
write_item(Y);
write_item(Z);

EXAMPLE OF SERIALIZABILITY TESTING

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

Time

Schedule E



Equivalent serial schedules

None

Reason

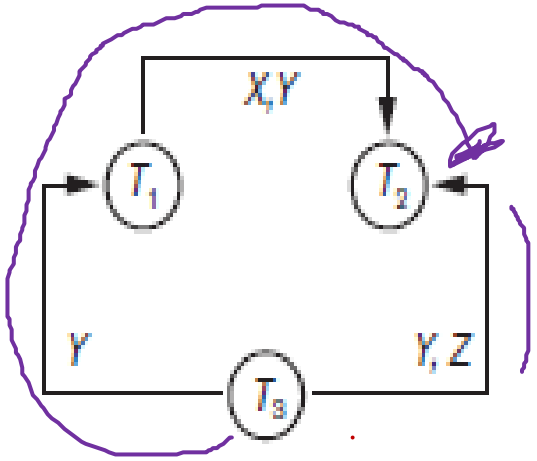
Cycle $X(T_1 \rightarrow T_2), Y(T_2 \rightarrow T_1)$
Cycle $X(T_1 \rightarrow T_2), YZ\ (T_2 \rightarrow T_3), Y(T_3 \rightarrow T_1)$

EXAMPLE OF SERIALIZABILITY TESTING

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

Time

Schedule F



Equivalent serial schedules

$T_3 \rightarrow T_1 \rightarrow T_2$

CHARACTERIZING SCHEDULES BASED ON SERIALIZABILITY

- View equivalence:
  - A less restrictive definition of equivalence of schedules
- View serializability:
  - Definition of serializability based on view equivalence.
  - A schedule is view serializable if it is view equivalent to a serial schedule.

CHARACTERIZING SCHEDULES BASED ON SERIALIZABILITY

- Two schedules are said to be view equivalent if the following three conditions hold:
  - The same set of transactions participates in S and S', and S and S' include the same operations of those transactions.
  - For any operation Ri(X) of Ti in S, if the value of X read by the operation has been written by an operation Wj(X) of Tj (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 Ri(X) of Ti in S'.
  - If the operation Wk(Y) of Tk is the last operation to write item Y in S, then Wk(Y) of Tk must also be the last operation to write item Y in S'.

CHARACTERIZING SCHEDULES BASED ON SERIALIZABILITY

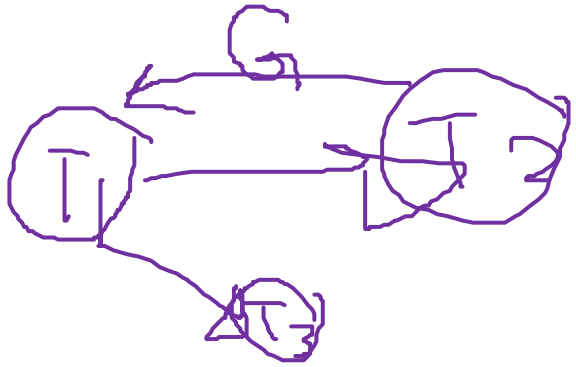
- The premise behind view equivalence:
  - As long as each read operation of a transaction reads the result of the same write operation in both schedules, the write operations of each transaction must produce the same results.
  - "The view": the read operations are said to see the same view in both schedules.

CHARACTERIZING SCHEDULES BASED ON SERIALIZABILITY

- Relationship between view and conflict equivalence:
  - The two are same under constrained write assumption which assumes that if T writes X, it is constrained by the value of X it read; i.e., new X = f(old X)
  - Conflict serializability is stricter than view serializability. With unconstrained write (or blind write), a schedule that is view serializable is not necessarily conflict serializable.
  - Any conflict serializable schedule is also view serializable, but not vice versa.

CHARACTERIZING SCHEDULES BASED ON SERIALIZABILITY

- Relationship between view and conflict equivalence (cont):
  - Consider the following schedule of three transactions
    - T1: r1(X), w1(X); T2: w2(X); and T3: w3(X):
  - Schedule Sa: r1(X); w2(X); w1(X); w3(X); c1; c2; c3;



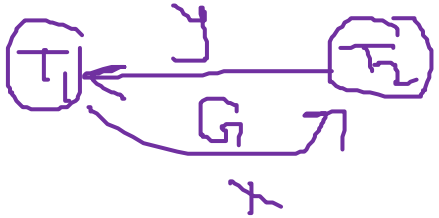
- In Sa, the operations w2(X) and w3(X) are blind writes, since T2 and T3 do not read the value of X.
  - Sa is view serializable, since it is view equivalent to the serial schedule T1, T2, T3.
  - However, Sa is not conflict serializable, since it is not conflict equivalent to any serial schedule.

CHARACTERIZING SCHEDULES BASED ON SERIALIZABILITY

- Other Types of Equivalence of Schedules
- Under special semantic constraints, schedules that are otherwise not conflict serializable may work correctly.
  - Using commutative operations of addition and subtraction (which can be done in any order) certain non-serializable transactions may work correctly

CHARACTERIZING SCHEDULES BASED ON SERIALIZABILITY

- Other Types of Equivalence of Schedules (contd.)
- Example: bank credit / debit transactions on a given item are separable and commutative.
  - Consider the following schedule S for the two transactions:
  - Sh : r1(X); w1(X); r2(Y); w2(Y); r1(Y); w1(Y); r2(X); w2(X);
  - Using conflict serializability, it is not serializable.
  - However, if it came from a (read, update, write) sequence as follows:
    - r1(X); X := X – 10; w1(X); r2(Y); Y := Y – 20; w1(Y);
    - Y := Y + 10; w1(Y); r2(X); X := X + 20; w2(X);
  - Sequence explanation: debit, debit, credit, credit.
  - It is a correct schedule for the given semantics



QUESTION

- The schedule S: r1(x), r1(y), w1(x), r2(x), r2 (z), w2(z) is …….

a-serial b- not serial

c- serializable d- conflict serializable

- The schedule S: r1(x), r2(y), w1(x), r2(y), r2(x), w2(y), C2. C1 is …..

a-recoverable b- unrecoverable

c- cascadeless d- cascaded rollback

- The schedule S: r1(x), r2(y), r3 (z), r2 (z), w1(x), w2(y), w3 (z), w2 (z) is …..

a-serial b- strict

c- conflict serializable d- not conflict serializable

QUESTION

- Consider schedules S1, S2, and S3 below. Determine whether each schedule is serializable, cascadeless, recoverable, or nonrecoverable.
- S1: r1 (X), r2 (Z), r1 (Z), r3 (X), r3 (Y), w1 (X), c1, w3 (Y), c3, r2 (Y), w2 (Z), w2 (Y), c2,
- S2: r1 (X), r2 (Z), r1 (Z), r3 (X), r3 (Y), w1 (X), w3 (Y), r2 (Y), w2 (Z), w2 (Y), c1, c2, c3,
- S3: r1 (X), r2 (Z), r3 (X), r1 (Z), r2 (Y), r3 (Y), w1 (X), c1, w2 (Z), w3 (Y), w2 (Y), c3, c2,

QUESTION

- For the schedule S: = [r1 (Z), r2 (Y), w2 (Y), r3 (Y), r1 (X), w1 (X), w1 (Z), w3 (Y), r2 (X), r1 (Y), w1 (Y), w2 (X), r3 (W), w3 (W)]
- Draw a precedence graph for this schedule, is it conflict serializable?

- Thank You