09 - Transaction-Concurrency Control

School of Computer Science
University of Windsor

Dr Shafaq Khan

Agenda

- **Lecture**
  - Concurrency control
  - Serializability and Recoverability
  - Locking
- **Lab 4**

Announcements

- **Assignment 3 - Spark certificate**

  Submission deadline:: Sec 1 & 4: Jul 16; Sec 2: Jul 17; Sec 3: Jul 18

- **Final Report**

  Submission deadline: Sec 1 & 4: Jul 30; Sec 2: Jul 31; Sec 3: Aug 1

Introductory Questions

What is the meaning of serializability and how it applies to concurrency control?

How locks can be used to ensure serializability?

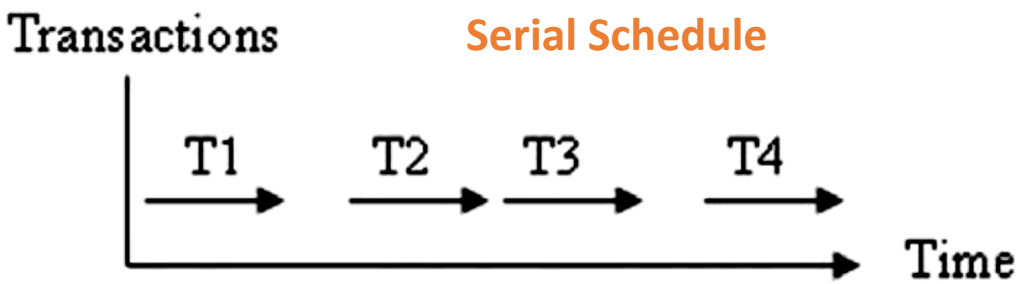
Schedule

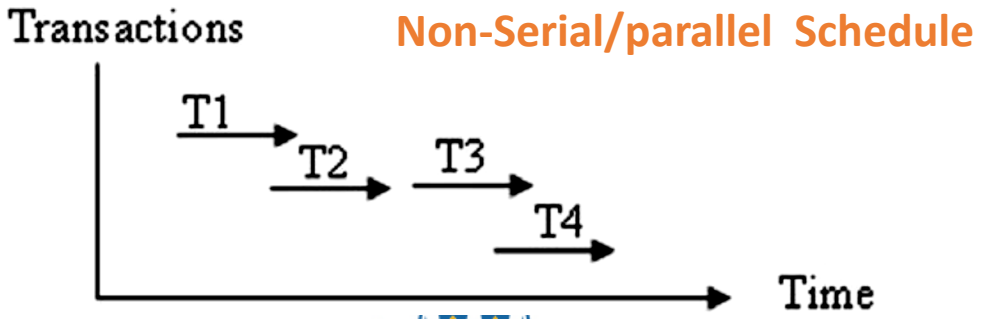
Transaction comprises a sequence of operations consisting of read and/or write actions to the database, followed by a commit or abort action.

Schedule: A schedule S is a set of transactions (T1, T2, T3,….Tn)

Serial Schedule: A schedule where the operations of each transaction are executed consecutively **without any interleaved operations** from other transactions.

Non serial/parallel schedule: A schedule where the operations from a set of concurrent transactions are interleaved.





Serializability and Recoverability

Most concurrency control techniques ensure serializability of schedules

Concurrency control techniques are used to ensure the "**isolation**" property (of the ACID properties of transactions)

- Even though transactions may be scheduled concurrently with interleaved operations, their **net effect** on the database must be **equivalent** to scheduling the transactions in some serial order

Objective of a concurrency control protocol is to schedule transactions in such a way as to avoid any interference.

Obvious solution?

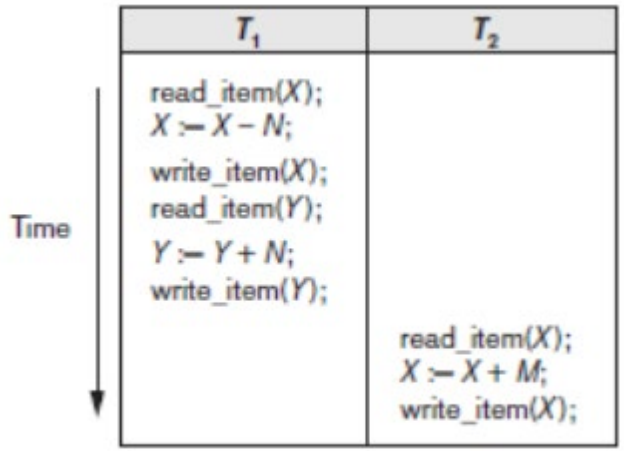
allow only one transaction to execute at a time: one transaction is committed before the next transaction is allowed to begin.

Aim of a multi-user DBMS: to maximize the degree of concurrency or parallelism in the system.

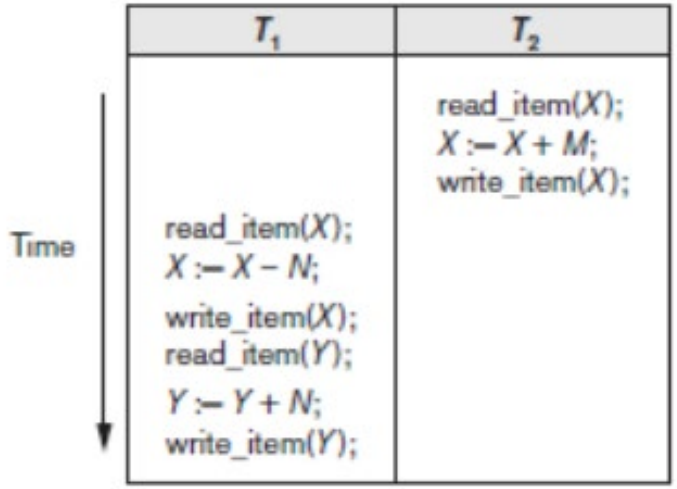
transactions that can execute without interfering with one another can run in parallel.

How?

Serial Schedule

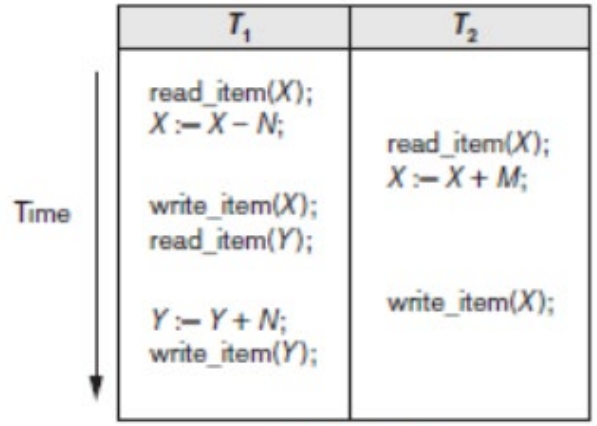


Schedule A

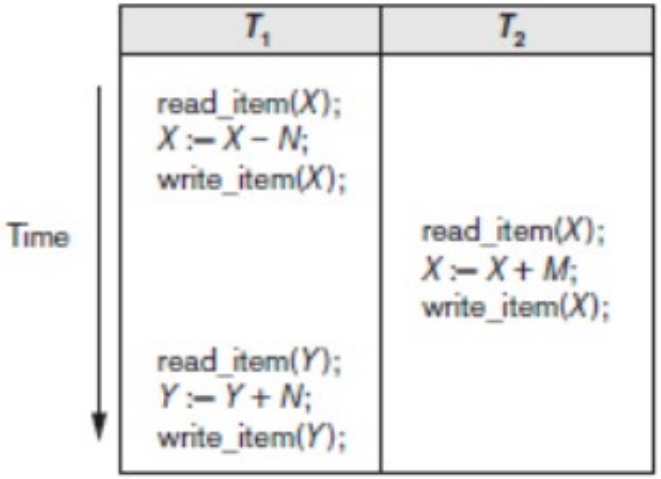


Schedule B

Non-Serial/parallel Schedule

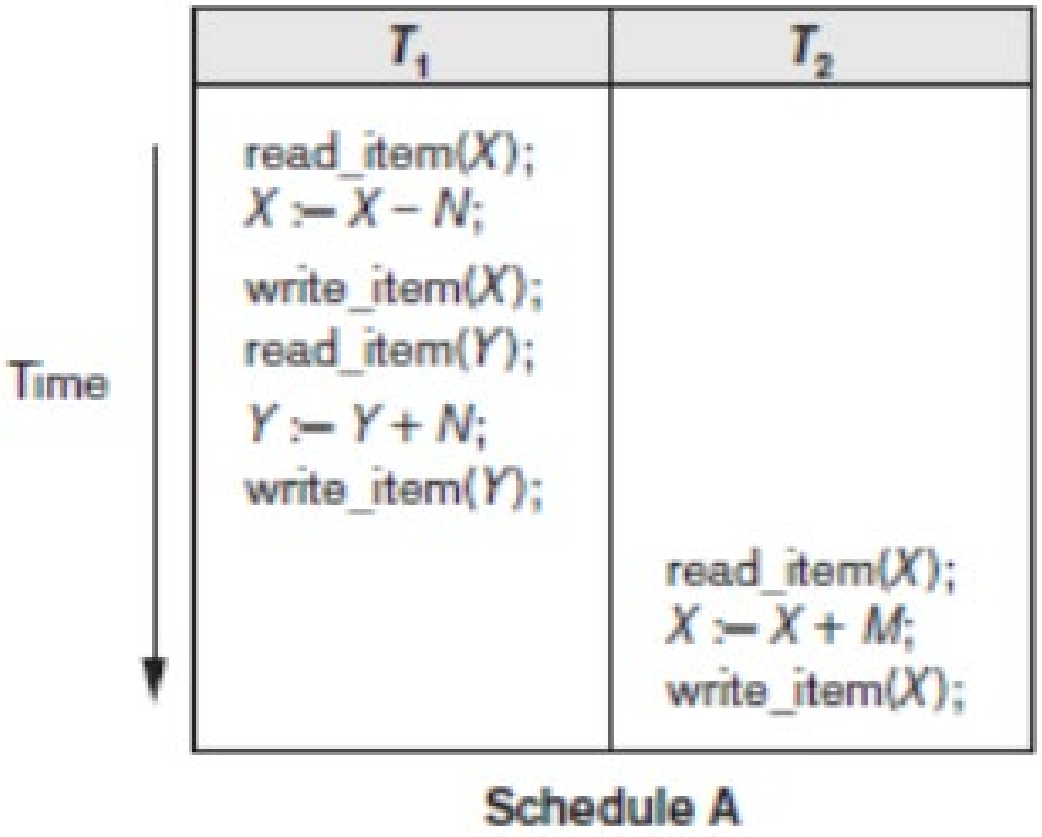


Schedule C



Schedule D

Serial Schedule



Schedule A

Assume
X=100, Y=50, M=10, N=20

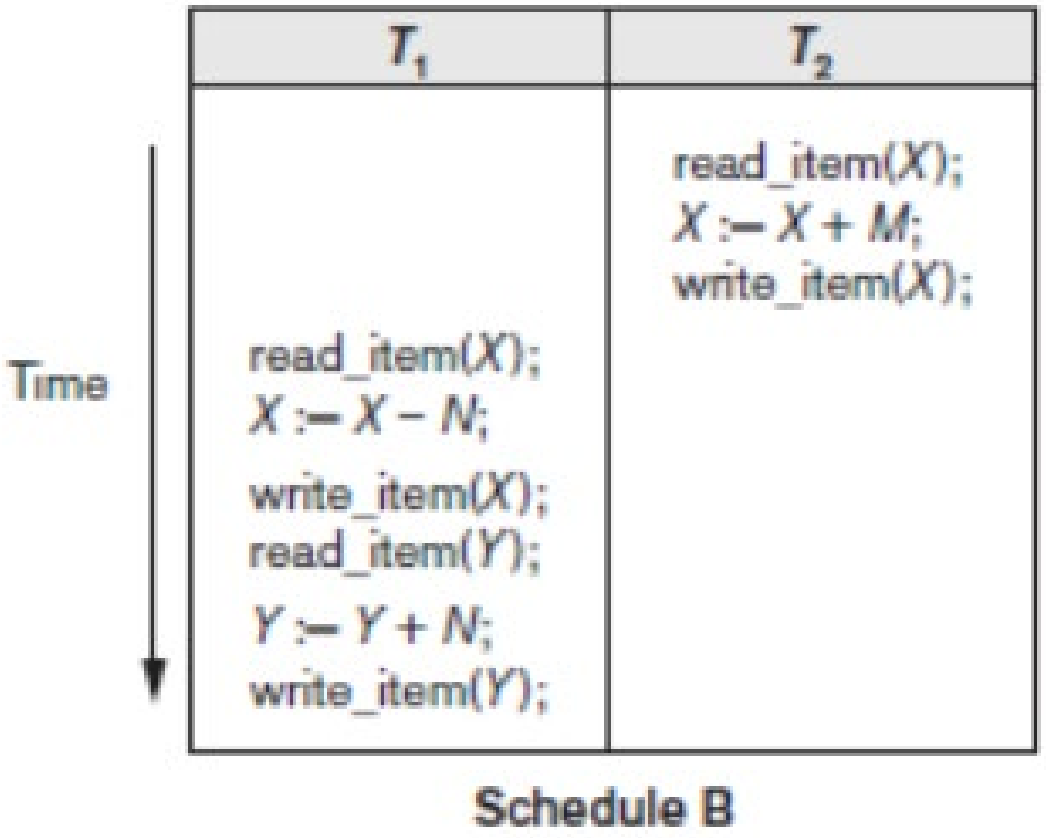
X=100
X=100-20=80

X=80
Y=50
Y=50+20
Y=70

X=80
X=80+10
X=90

Serial Schedule

Assume
X=100, Y=50, M=10, N=20



Schedule B

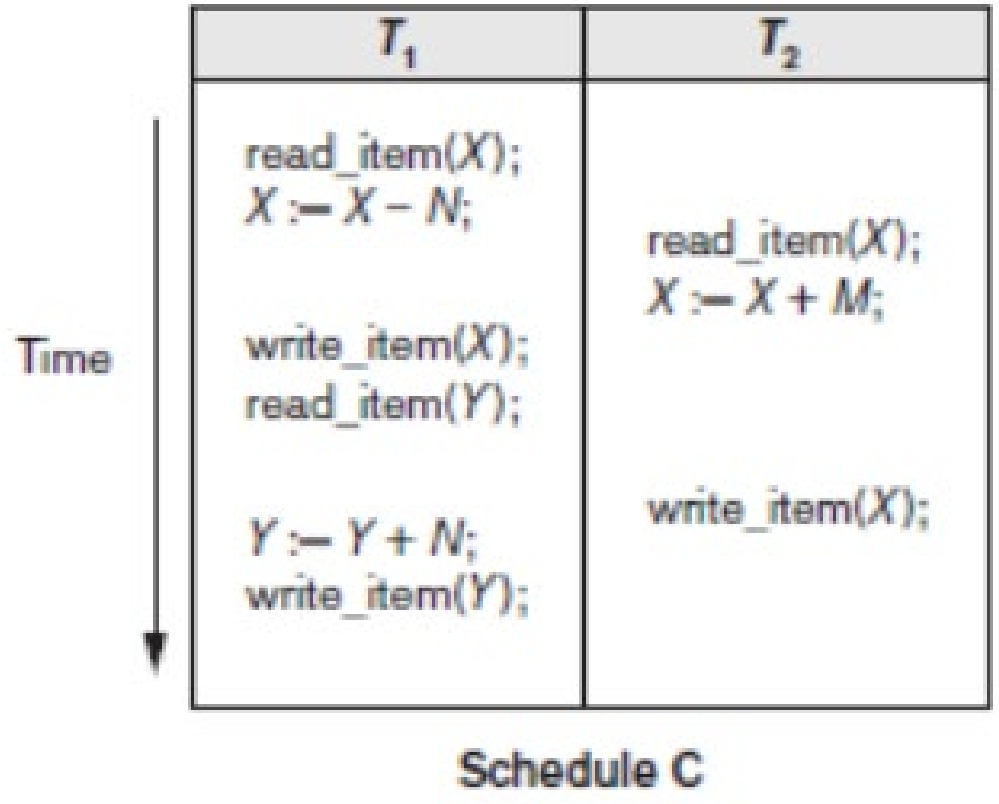
X=100
X=100+10=110
X=110

X=110
X=110-20=90

X=90
Y=50
Y=50+20=70
Y=70

Non-serial Schedule

Assume
X=100, Y=50, M=10, N=20



Schedule C

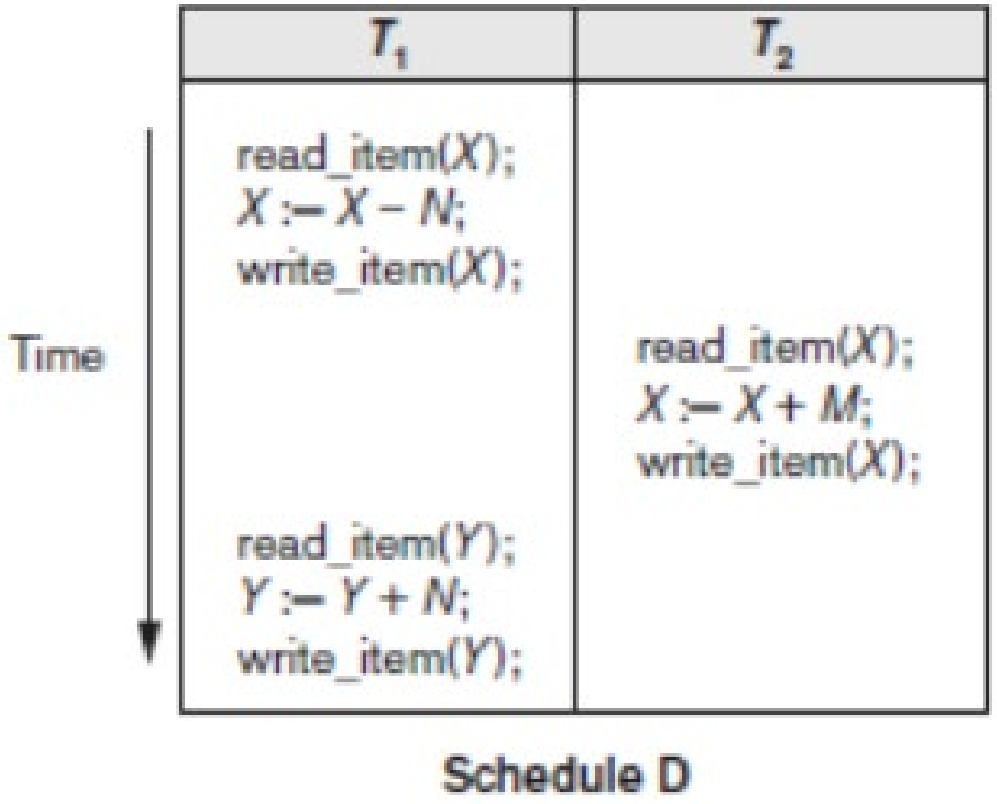
X=100
X=100-20=80
X=100 //Lost Update
X=100+10=110
X=110
Y=50

X=110
Y=50+20
Y=70

Not equivalent to Serial Schedule A or/and Serial Schedule B)

Non-Serial Schedule

Assume
X=100, Y=50, M=10, N=20



Schedule D

X=100
X=100-20=80
X=80

X=80
X=80+10=90
X=90

Y=50
Y=50+20
Y=70

Equivalent to Serial Schedule A or/and Serial Schedule B)

Serializable Schedule

- A schedule *S* of *n* transactions is serializable:
  - If it is equivalent to some **serial schedule** of the same *n* transactions
- With n transactions, n! serial schedules are possible
- A number of non-serial schedules are also possible and can be divided into two disjoint groups
  - Group 1: Non-serial schedules which are equivalent to one or more of the(n!) serial schedules and are thus **serializable**
  - Group 2: Non-serial schedules which are **not equivalent** to any of the (n!) serial schedules and are thus **non-serializable**

Serializability

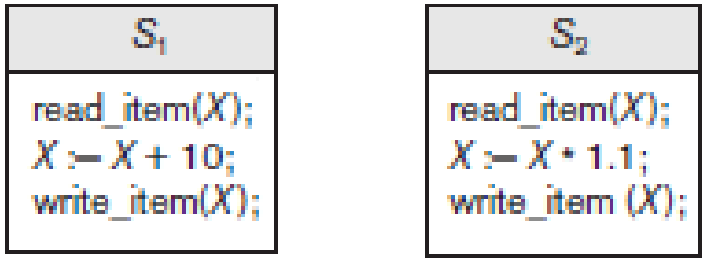
Objective of serializability is to find non-serial schedules that allow transactions to execute concurrently without interfering with one another.

If a set of transactions executes concurrently, we say that *the (nonserial) schedule is correct if it produces the same result as some serial execution.* Such a schedule is called **serializable**. To prevent inconsistency from transactions interfering with one another, it is essential to guarantee serializability of concurrent transactions.

In serializability, ordering of read/writes is important:

- ✓If two transactions only read a data item, they do not conflict, and order is not important.
- ✓If two transactions either read or write separate data items, they do not conflict, and order is not important.
- ✓If one transaction writes a data item and another reads or writes same data item, order of execution is important

Equivalence of Two Schedules



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

- Result Equivalence may not always be reliable
- The results may be equivalent for some/few initial values, but not in the general case

- Conflict Equivalence
- View Equivalence

Conflicts

A conflict occurs when two running transactions perform noncompatible operations on the same data item of the database.

A conflict occurs when one transaction writes an item that another transaction is reading or writing.

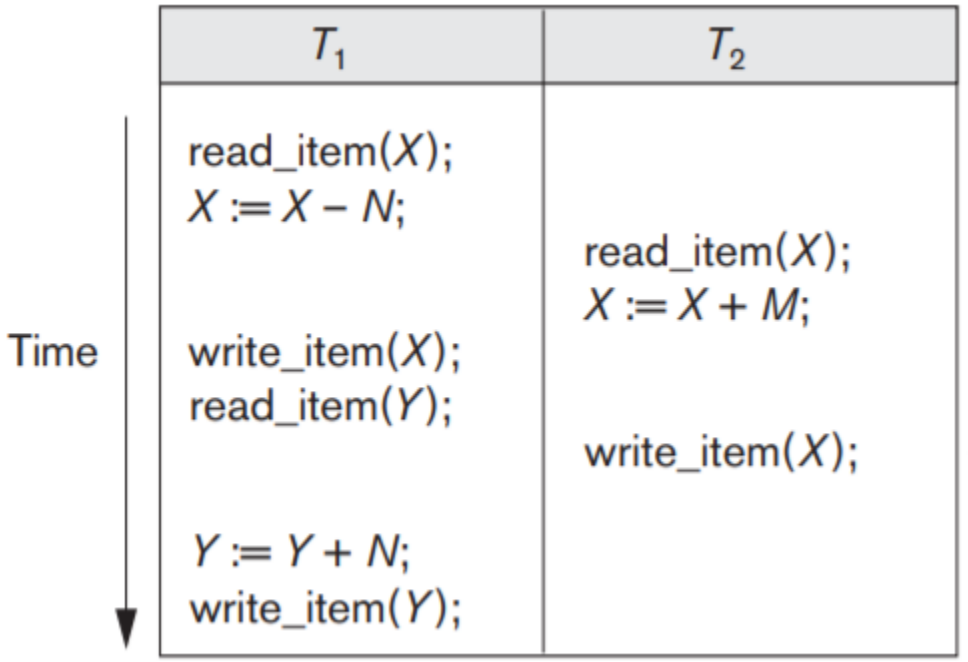
	READ T_2	WRITE T_2
READ T_1	No Conflict	Conflict
WRITE T_1	Conflict	Conflict

CONFLICT MATRIX.

Conflicts: Example

Conflict operations:

- ✓the operations $r_1(X)$ and $w_2(X)$,
- ✓the operations $r_2(X)$ and $w_1(X)$,
- ✓and the operations $w_1(X)$ and $w_2(X)$



Do not Conflict:

- ✓the operations $r_1(X)$ and $r_2(X)$ -- since they are both read operations.
- ✓the operations $w_2(X)$ and $w_1(Y)$ -- they operate on distinct data items X and Y.
- ✓operations $r_1(X)$ and $w_1(X)$ -- they belong to the same transaction.

Conflict serializability

Conflict serializable schedule orders any conflicting operations in same way as some serial execution.

Testing for conflict serializability:

Under constrained write rule (transaction updates data item based on its old value, which is first read), use **precedence graph** to test for serializability.

Precedence Graph:

1. Create node for each transaction;
2. Check conflict pair in other transactions and Draw edges
   - ✓ a directed edge $T_i \rightarrow T_j$, if T_j reads the value of an item written by T_i;
   - ✓ a directed edge $T_i \rightarrow T_j$, if T_j writes a value into an item after it has been read by T_i.
   - ✓ a directed edge $T_i \rightarrow T_j$, if T_j writes a value into an item after it has been written by T_i.

Conflict Pairs

$R_i(A) \rightarrow W_j(A)$
$W_i(A) \rightarrow R_j(A)$
$W_i(A) \rightarrow W_j(A)$

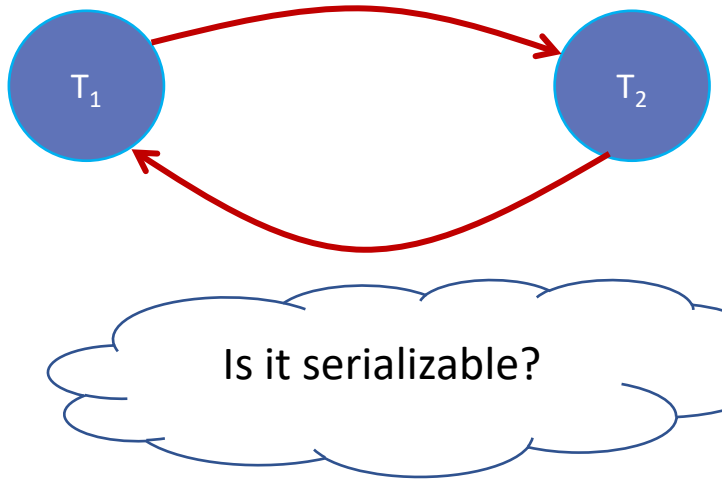
3. Check for loop /cycle

If there is a cycle in the precedence graph, schedule S is not (conflict) serializable; if there is no cycle, S is serializable, and hence consistent schedule.

Example - Non-conflict serializable schedule

$R_i(A) \rightarrow W_j(A)$
$W_i(A) \rightarrow R_j(A)$
$W_i(A) \rightarrow W_j(A)$

T_1 is transferring $100 from one account with balance X to another account with balance Y.
T_2 is increasing balance of these two accounts by 10%.



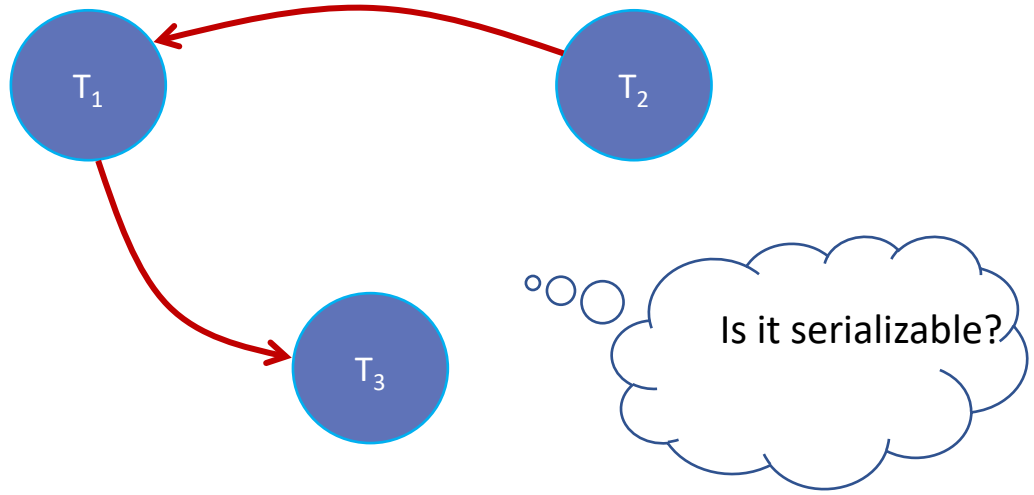
Precedence graph has a **cycle** and so is **not** serializable.

Time	T_1	T_2
t_1	BEGIN	
t_2	READ(X)	
t_3	X=X+100	
t_4	WRITE(X)	BEGIN
t_5		READ(X)
t_6		X=X*1.1
t_7		WRITE(X)
t_8		READ(Y)
t_9		Y=Y*1.1
t_{10}		WRITE(Y)
t_{11}	READ(Y)	COMMIT
t_{12}	Y=Y-100	
t_{13}	WRITE(Y)	
t_{14}	COMMIT	

Two concurrent update transactions that are **not** conflict serializable.

Example

Time	T_1	T_2	T_3
t_1	Read(A)		
t_2	Write(A)		
t_3			Read(A)
t_4			Write(A)
t_5			Commit
t_6		Read(B)	
t_7		Write(B)	
t_8		Commit	
t_9	Read(B)		
t_{10}	Write(B)		
t_{10}	Commit		



Precedence graph does **not** have a cycle and so it is serializable and hence consistent schedule.

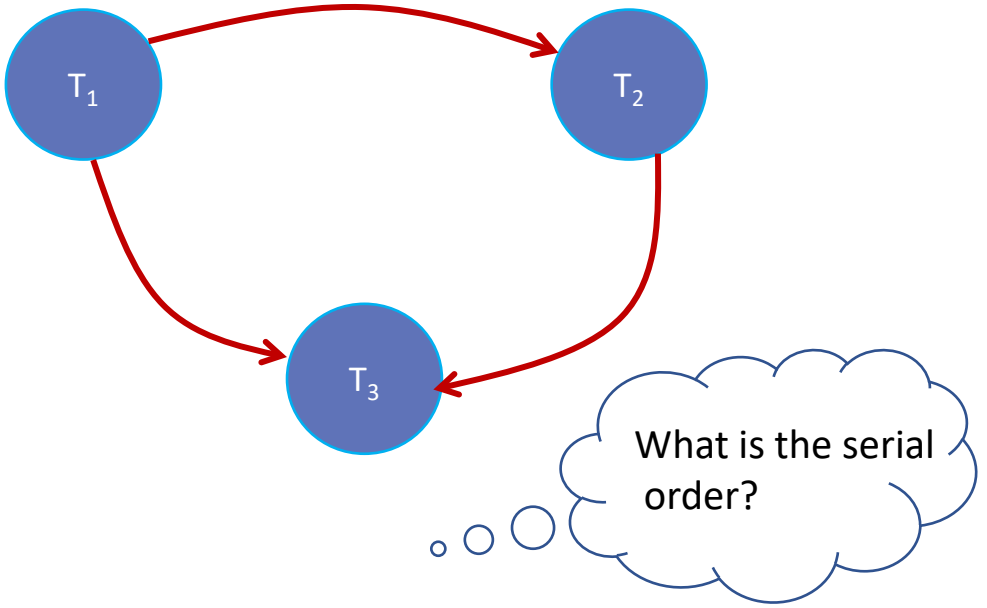
To check the serial order:

- Check for vertex where indegree=0. Any vertex which has no edge. Remove that vertex and related edges. Keep repeating this process.

Which gives us the serial schedule: T2->T1->T3

Time	T_1	T_2	T_3
t_1	Read(A)		
t_2	Read(C)		
t_3	Write(A)		
t_4		Read(B)	
t_5	Write (C)		
t_6		Read(A)	
t_7			Read (C)
t_8		Write(B)	
t_9			Read (B)
t_{10}			Write(C)
t_{11}		Write(A)	
t_{12}			Write(B)

Example



Precedence graph has no cycle that's why it is serializable and hence consistent schedule.

Concurrency Control

The purpose of concurrency control is to prevent two different users (or two different connections by the same user) from trying to update the same data at the same time.

Concurrency control can also prevent one user from seeing out-of-date data while another user is updating the same data.

Concurrency Control Techniques

Serializability can be achieved in several ways;

However, the two main concurrency control techniques that allow transactions to execute safely in parallel.

Locking,
Timestamping

Both are conservative approaches (or pessimistic): **delay transactions** in case they conflict with other transactions.

Optimistic methods assume conflict is rare and allow transactions to proceed unsynchronized and check for conflicts only at the end, when a transaction commits.

Locking Methods

Lock: A procedure used to control concurrent access to data. When one transaction is accessing the database, a lock may deny access to other transactions to prevent incorrect results.

- ✓ Most widely used approach to ensure serializability.
- ✓ Generally, a transaction must claim a **shared** (read) or **exclusive** (write) lock on a data item before read or write.
- ✓ Lock prevents another transaction from modifying item or even reading it, in the case of a write lock.

Locking - Basic Rules

Shared lock: If a transaction has a shared lock on a data item, it can read the item but not update it S(A). Other transactions can also read but not write on this data item.

Exclusive lock: If a transaction has an exclusive lock on a data item, it can both read and update the item X(A). No other transaction can read or update this item.

- ✓ Because read operations cannot conflict, it is permissible for more than one transaction to hold shared locks simultaneously on the same item.
- ✓ Exclusive lock gives transaction exclusive access to that item.
  - As long as a transaction holds the exclusive lock on the item, no other transactions can read or update that data item.

Compatibility Matrix

	Shared	Exclusive
Shared	✓	✗
Exclusive	✗	✗

Locking - Basic Rules

Locks are used in the following way:

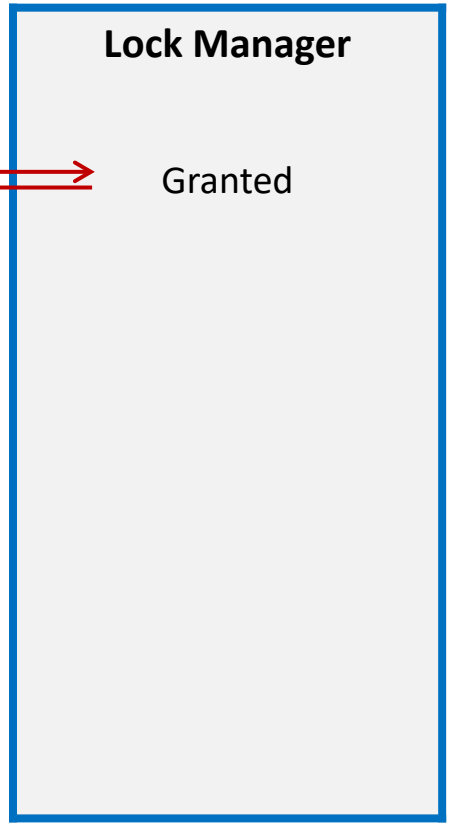
- ✓ Any transaction that needs to access a data item must **first lock the item**, requesting a shared lock for read-only access or an exclusive lock for both read and write access.
- ✓ If the item is not already locked by another transaction, the lock will be **granted**.
- ✓ If the item is currently locked, the DBMS determines whether the request is compatible with the existing lock. If a shared lock is requested on an item that already has a shared lock on it, the request will be granted; otherwise, the transaction must **wait** until the existing lock is released.
- ✓ A transaction continues to hold a lock until it explicitly releases it either during execution or when it terminates (aborts or commits). It is only when the exclusive lock has been released that the effects of the write operation will be made visible to other transactions

Some systems allow transaction to **upgrade** read lock to an exclusive lock, or **downgrade** exclusive lock to a shared lock.

Locks

Schedule A

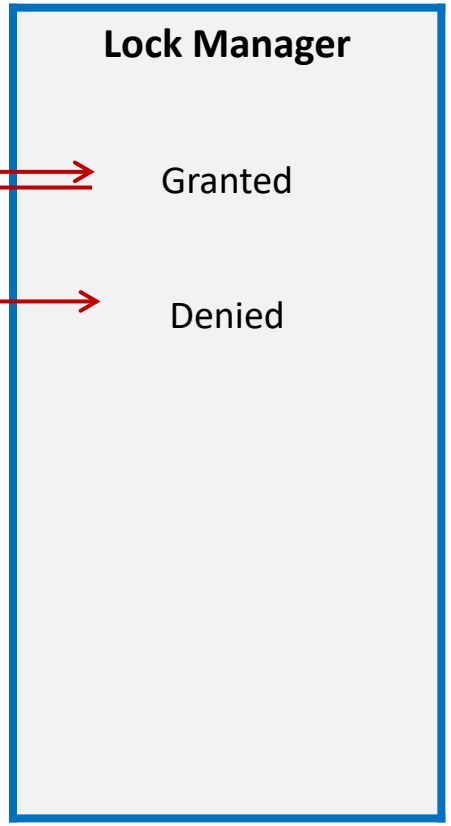
Time	T_1	T_2
t_1	LOCK(A)	
t_2	Read(A)	
t_3		LOCK(A)
t_4	Write(A)	
t_5	Read(A)	
t_6	UNLOCK(A)	
t_7	Read(C)	Read(A)
t_8		Write(A)
t_9	Commit	UNLOCK(A)
t_{10}		Commit



Locks

Schedule A

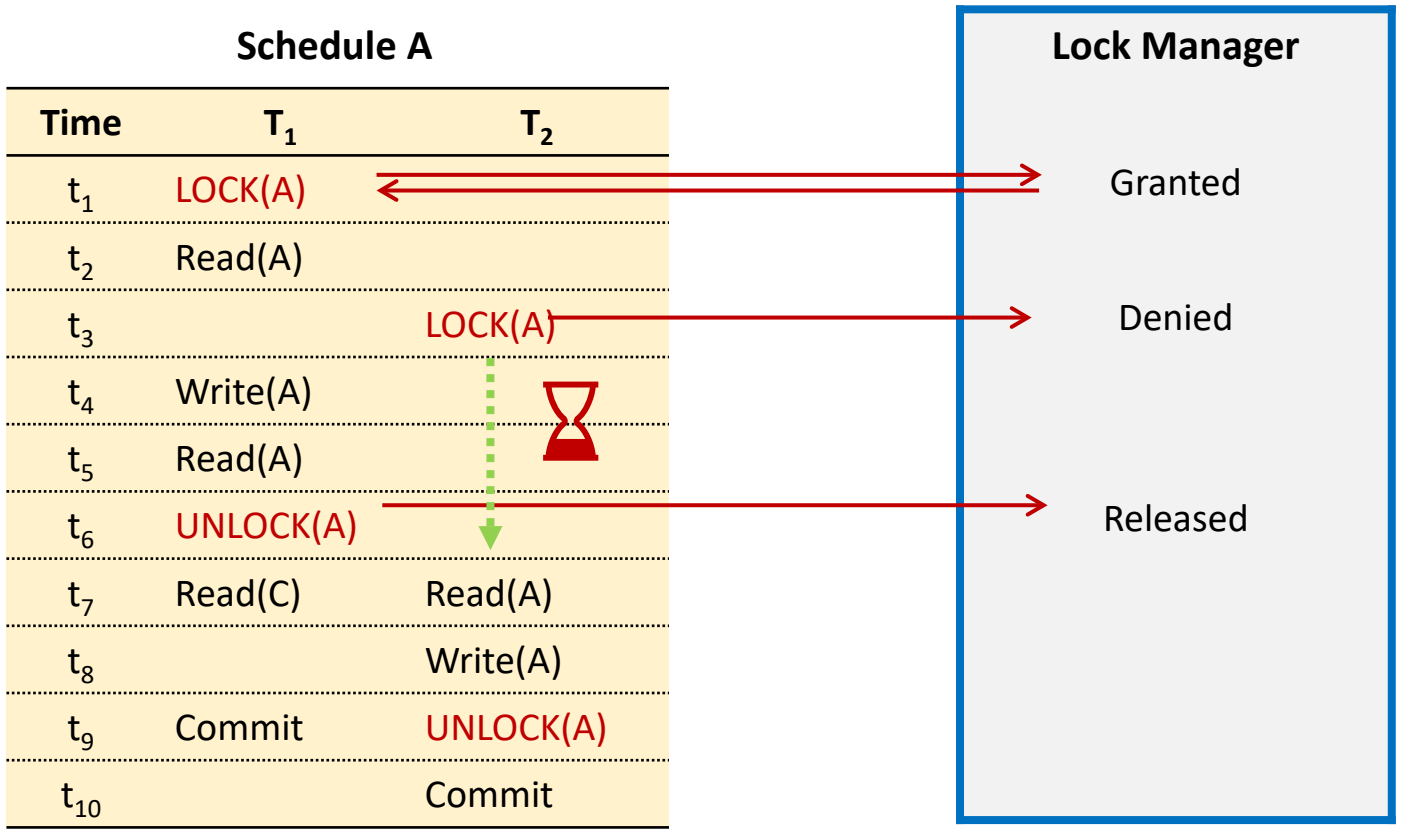
Time	T_1	T_2
t_1	LOCK(A)	
t_2	Read(A)	
t_3		LOCK(A)
t_4	Write(A)	
t_5	Read(A)	
t_6	UNLOCK(A)	
t_7	Read(C)	Read(A)
t_8		Write(A)
t_9	Commit	UNLOCK(A)
t_{10}		Commit



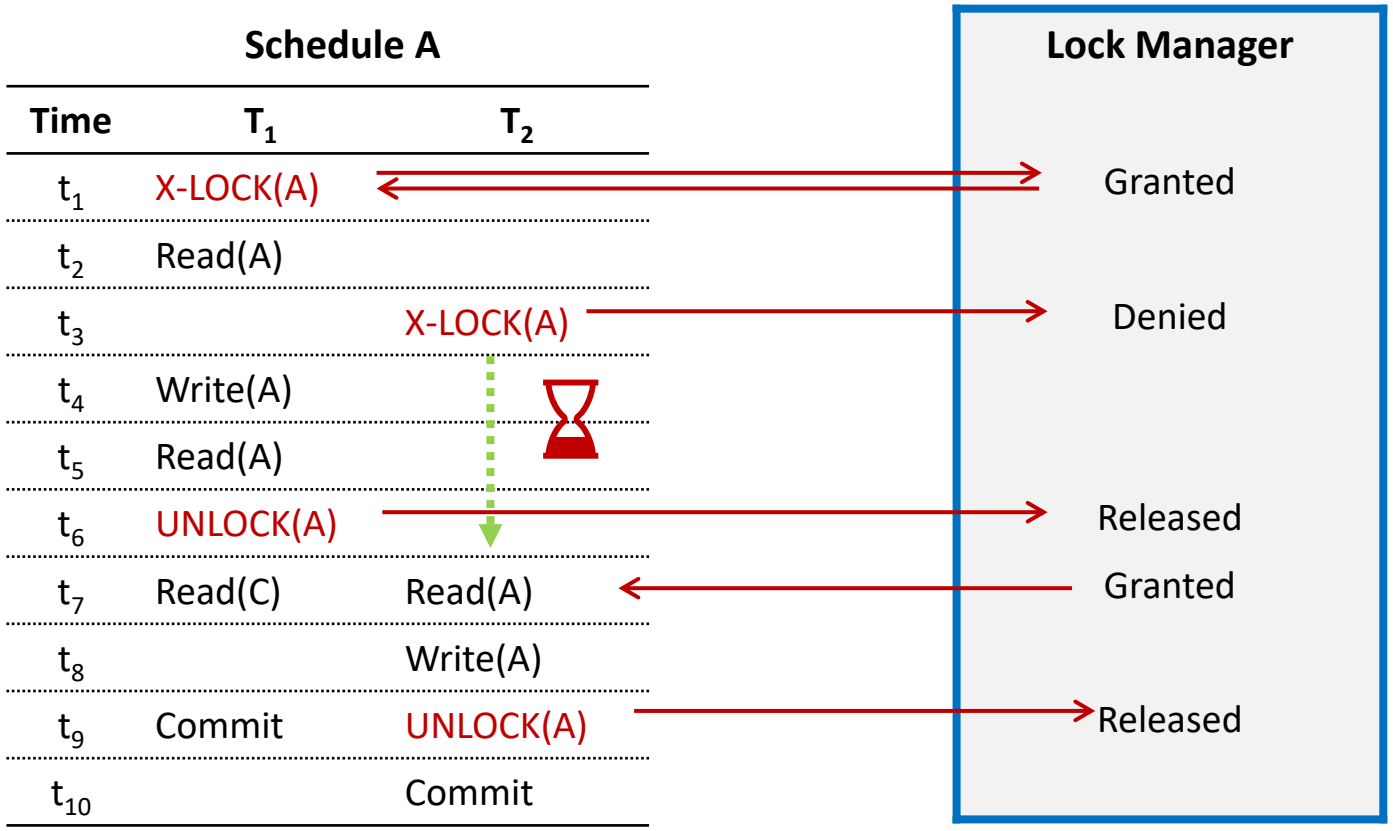
Locks

Schedule A

Time	T_1	T_2
t_1	LOCK(A)	
t_2	Read(A)	
t_3		LOCK(A)
t_4	Write(A)	
t_5	Read(A)	
t_6	UNLOCK(A)	
t_7	Read(C)	Read(A)
t_8		Write(A)
t_9	Commit	UNLOCK(A)
t_{10}		Commit



Locks


Schedule A
Time
T_1
T_2
t_1 X-LOCK(A)
t_2 Read(A)
t_3 X-LOCK(A)
t_4 Write(A)
t_5 Read(A)
t_6 UNLOCK(A)
t_7 Read(C) Read(A)
t_8 Write(A)
t_9 Commit UNLOCK(A)
t_{10} Commit
Lock Manager
Granted
Denied
Released
Granted
Released

Summary

Serializability helps for the transactions that can execute without interfering with one another can run in parallel.

Various types of Serializability: Conflict Serializability and View Serializability

Serializability can be achieved: Locking and Timestamping

Lock: When one transaction is accessing the database, a lock may deny access to other transactions to prevent incorrect results.

Any Questions