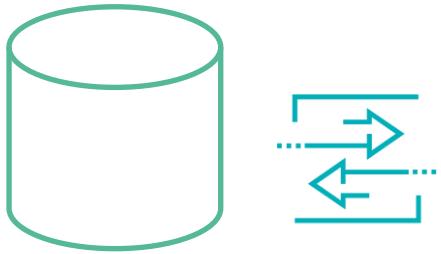
Transaction

A **transaction** is a unit of work performed within a database

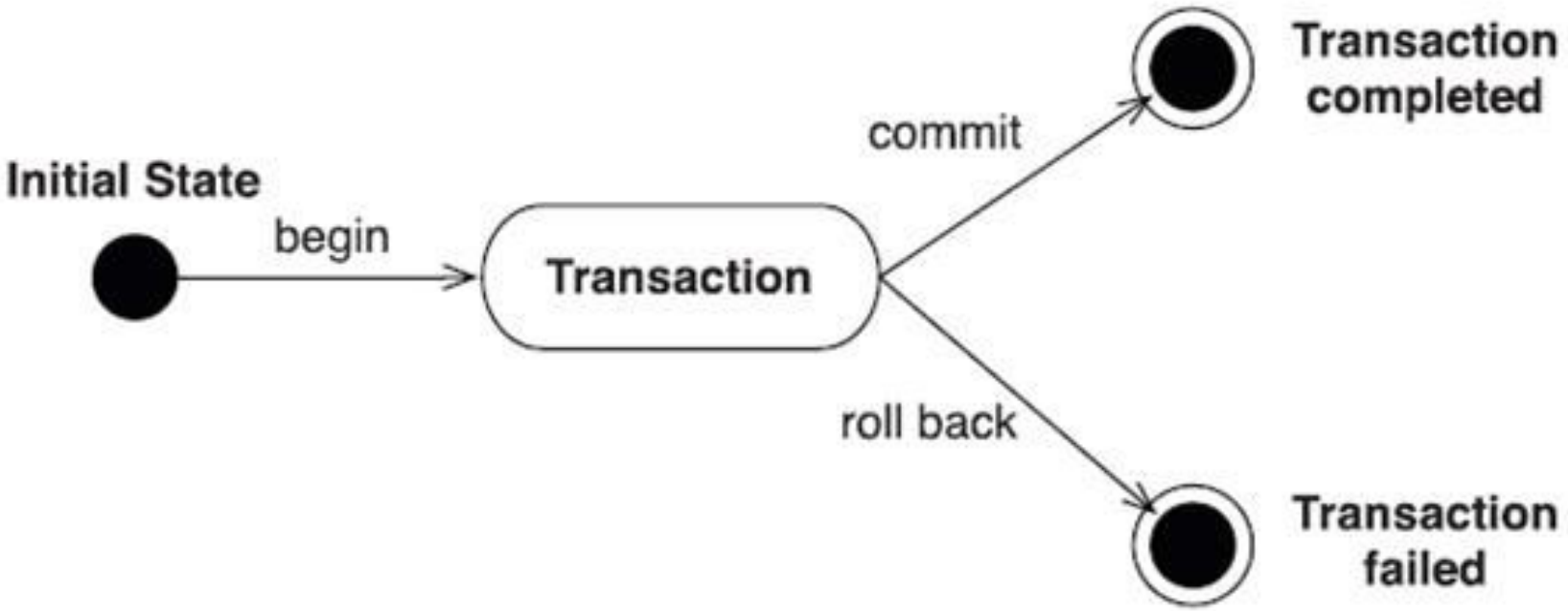


Transaction is a logical unit that is independently executed for data retrieval or updates

Transactions are represented as Schedules

	T_8	T_9
t_0	read (A)	
t_1	write (A)	
t_2		read (A)
t_3		commit
t_4	read (B)	

States of a Transaction

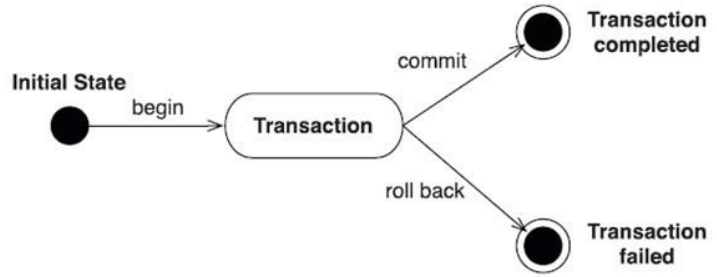


SQL Transaction

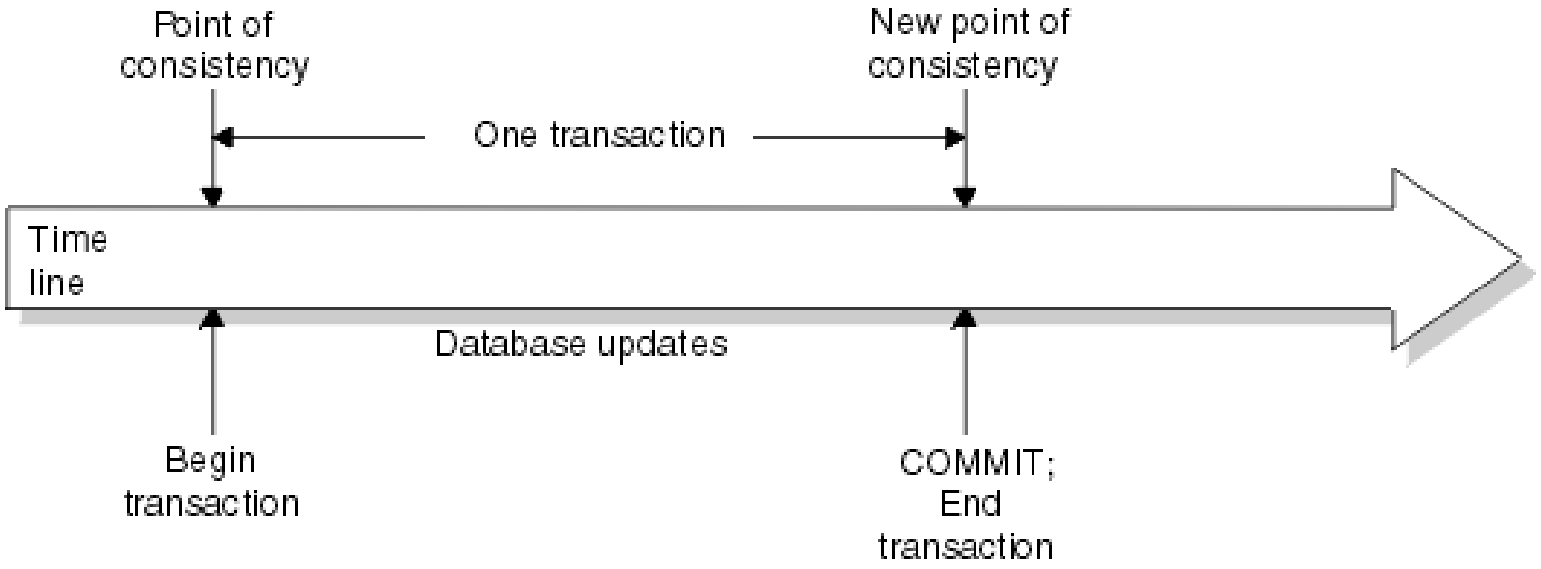
SQL **Transaction** is helpful to execute one more statements as a set.

```
BEGIN TRANSACTION

INSERT INTO [dbo].[EmployeeRecords] (
            [EmpID], [FirstName], [LastName], [Education], [Occupation], [YearlyIncome], [Sales])
     VALUES (5, 'SQL', 'Server', 'Education', 'Teaching', 10000, 200)

UPDATE [dbo].[EmployeeRecords]
    SET [Education] = 'Tutorials',
        [YearlyIncome] = 98000
    WHERE [EmpID] = 5

COMMIT TRANSACTION
```

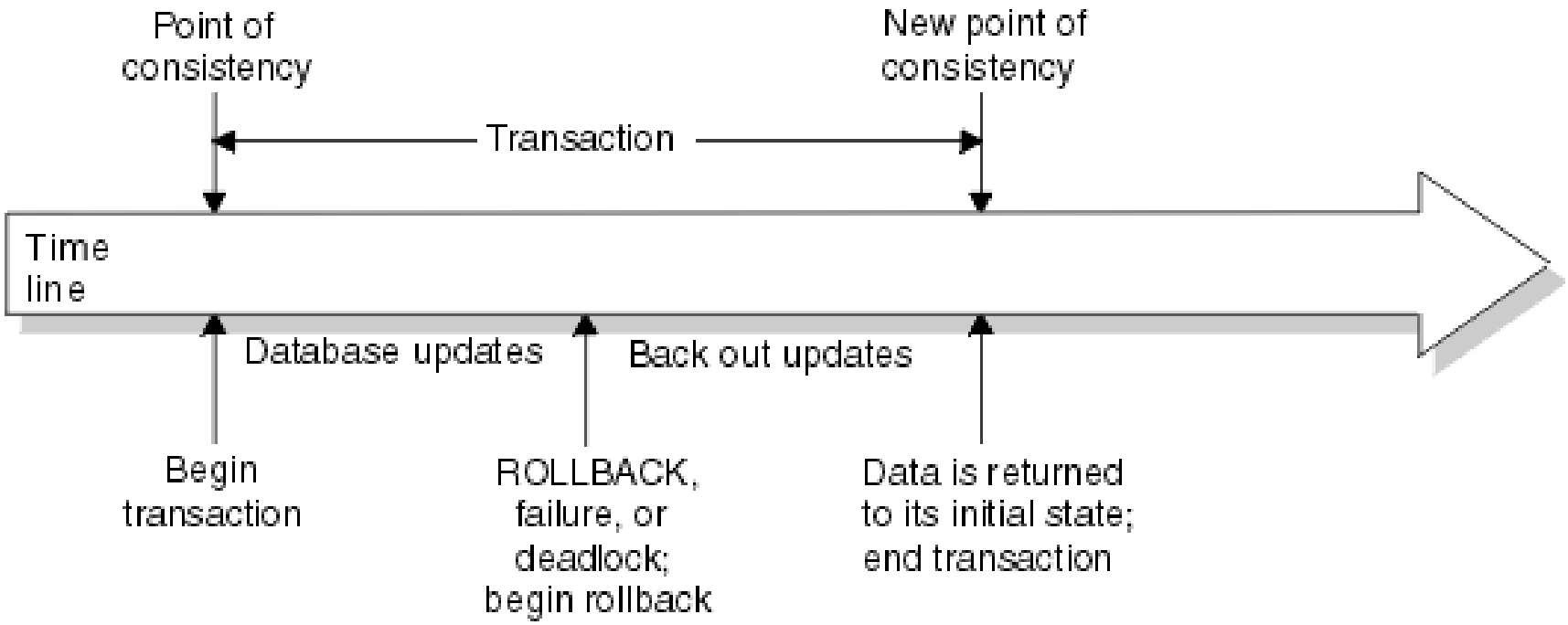
```
(1 row(s) affected)

(1 row(s) affected)
```



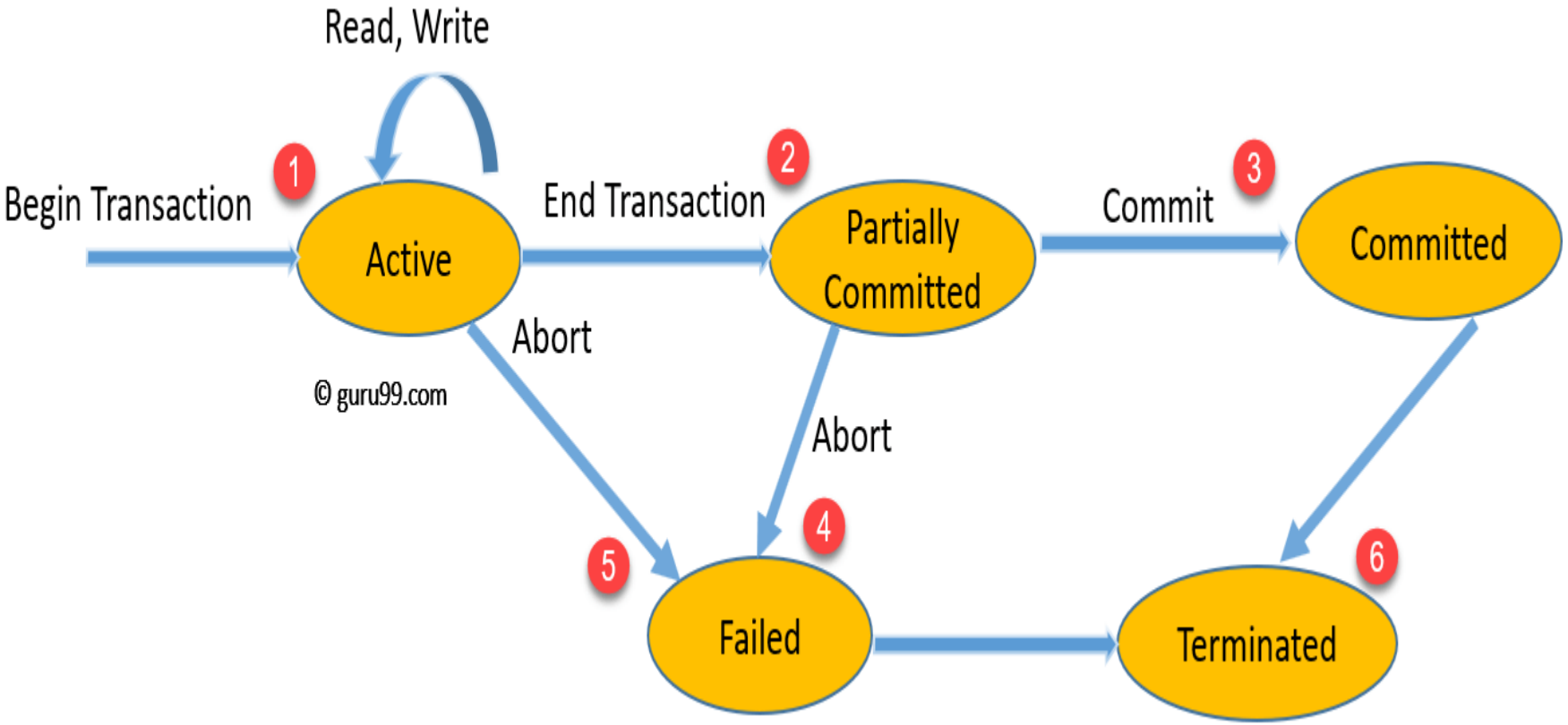
States of a Transaction: COMMIT



States of a Transaction: ROLLBACK

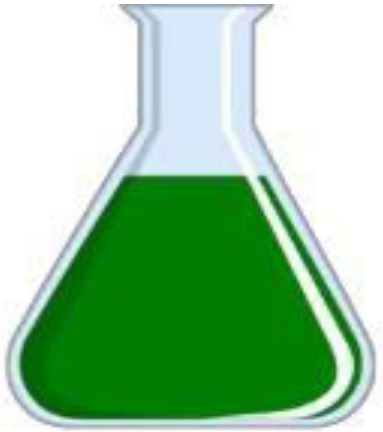


Complete life cycle of a Transaction



Properties of a Transaction

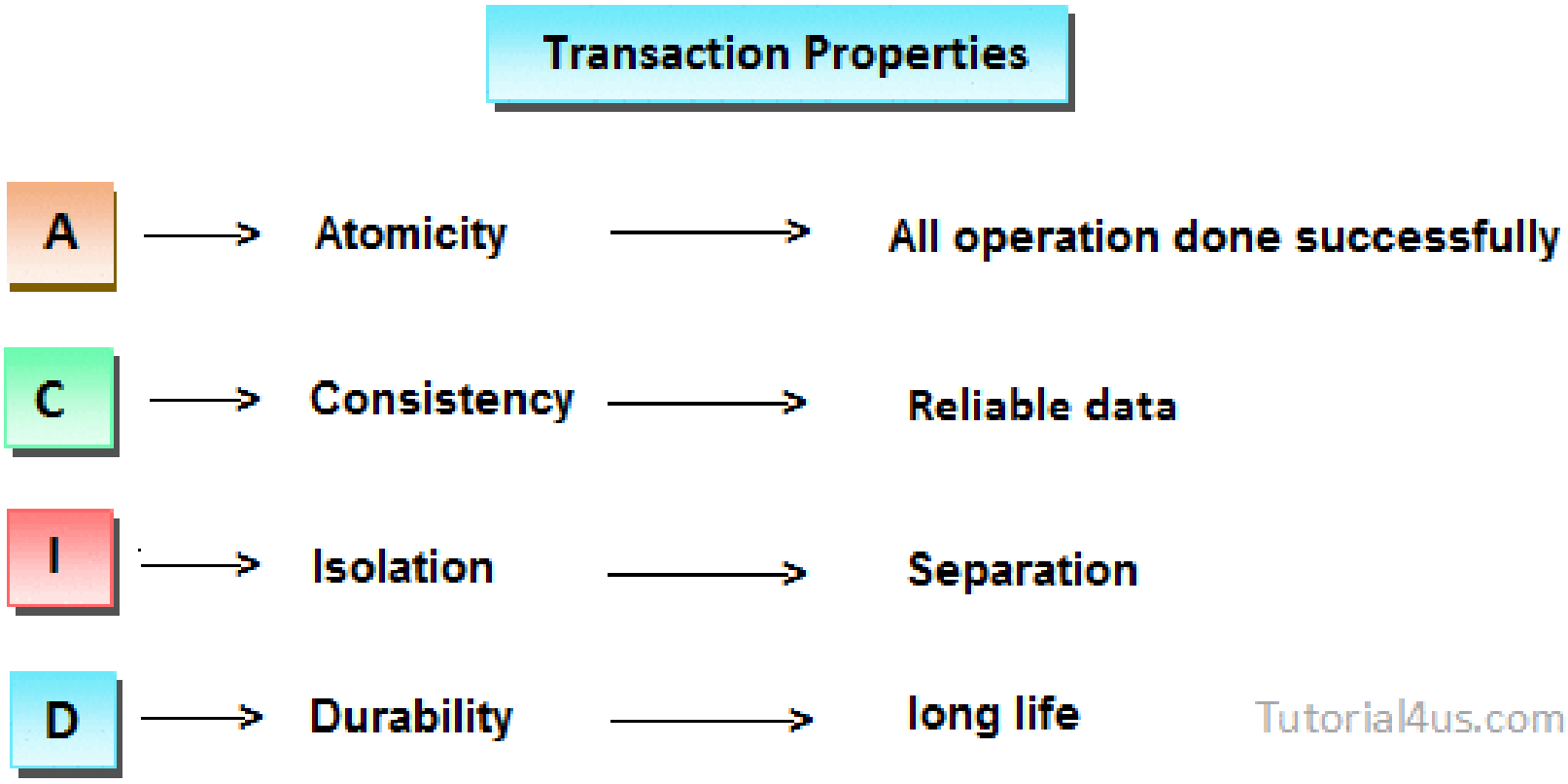
ACID PROPERTIES



To preserve the integrity of data the database system must ensure ACID properties

- **A** Atomicity
- **C** Consistency
- **I** Isolation
- **D** Durability

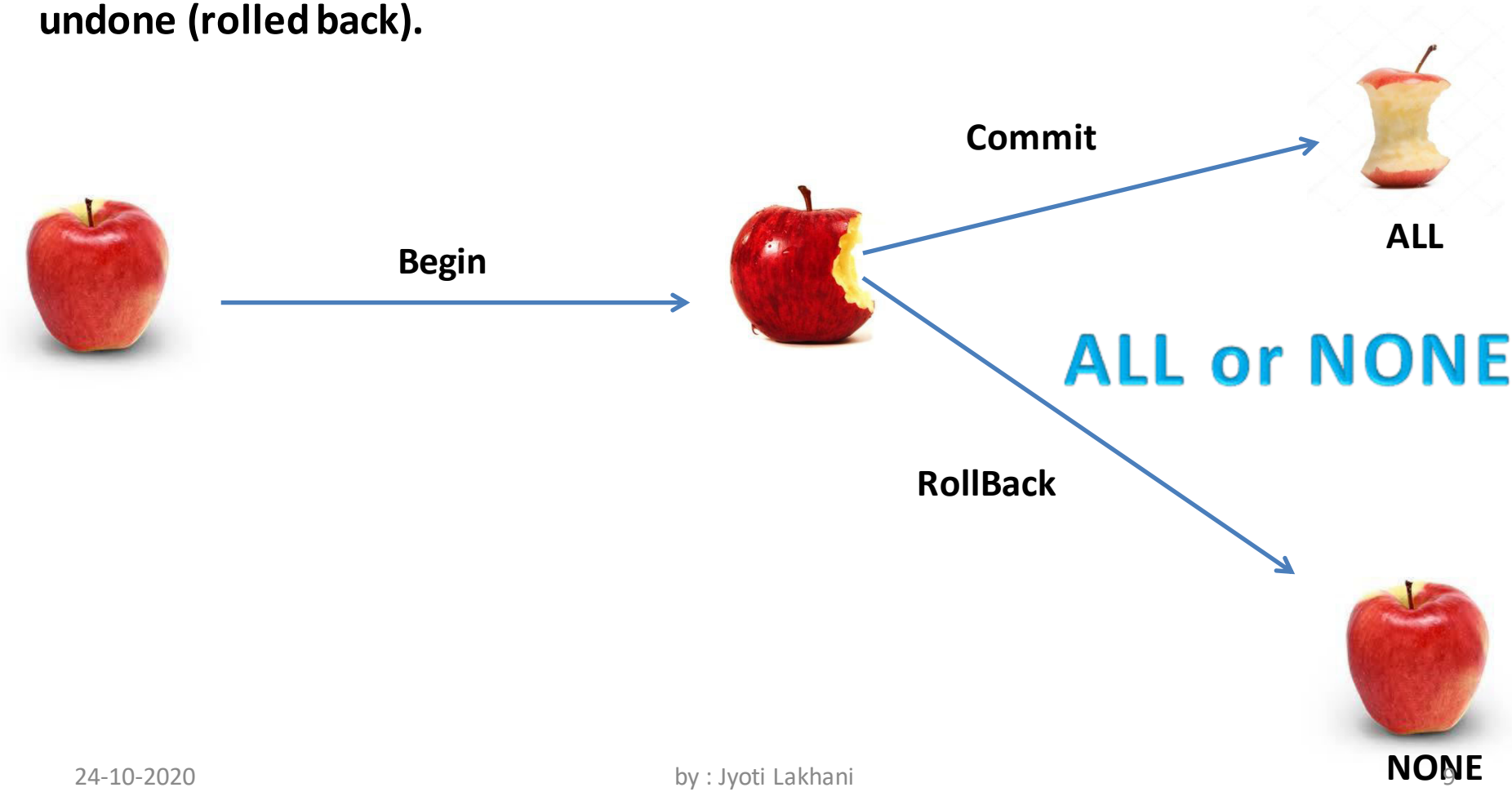
ACID Properties



ACID Properties: ATOMICITY

ATOMICITY

A transaction must be fully complete, saved (committed) or completely undone (rolled back).



CONSISTENCY

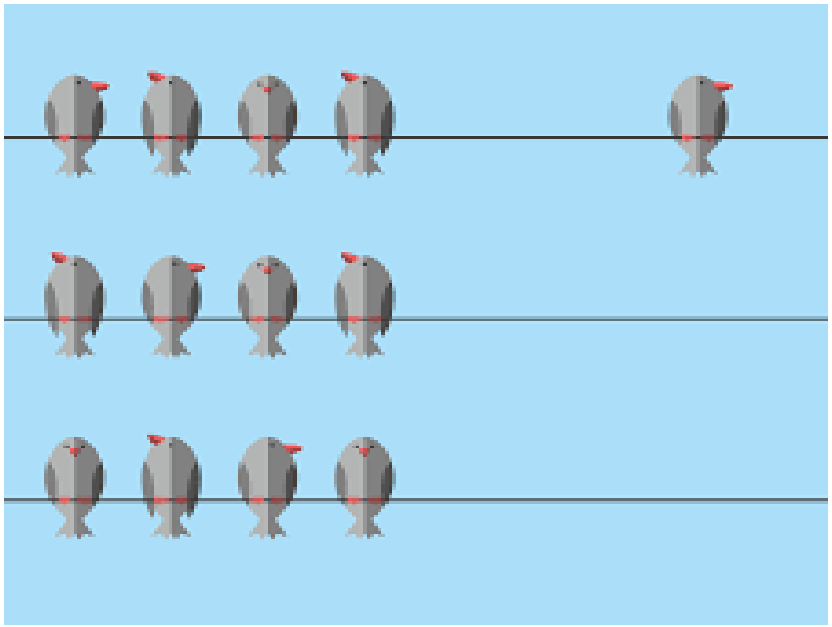
The transaction must be fully compliant with the state of the database as it was prior to the transaction

CONSISTENCY

INCONSISTENCY

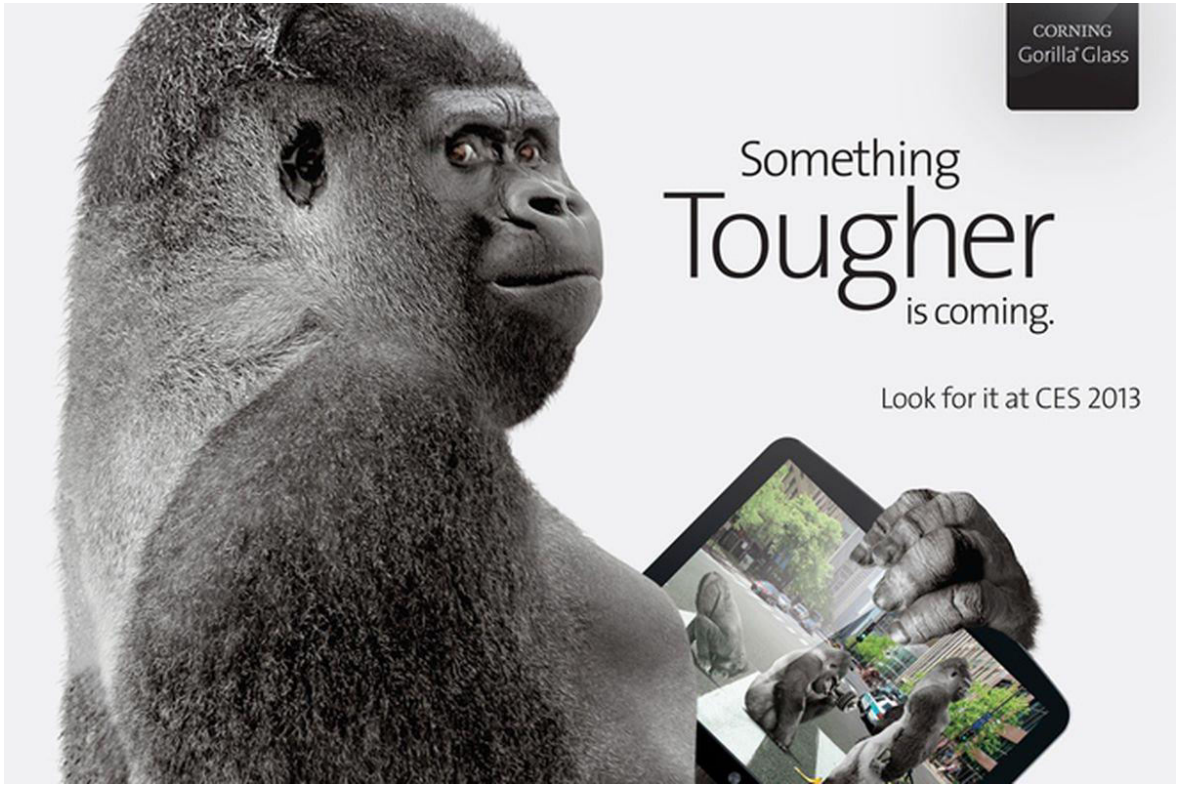
ISOLATION

Transaction data must not be available to other transactions until the original transaction is committed or rolled back.

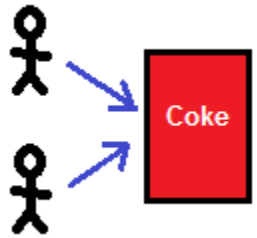


DURABILITY

Transaction data changes must be available, even in the event of database failure.

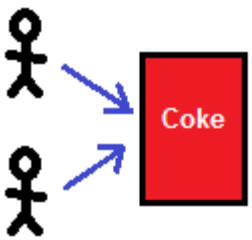


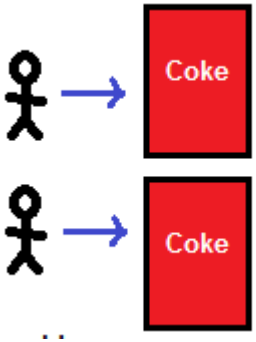
Concurrency



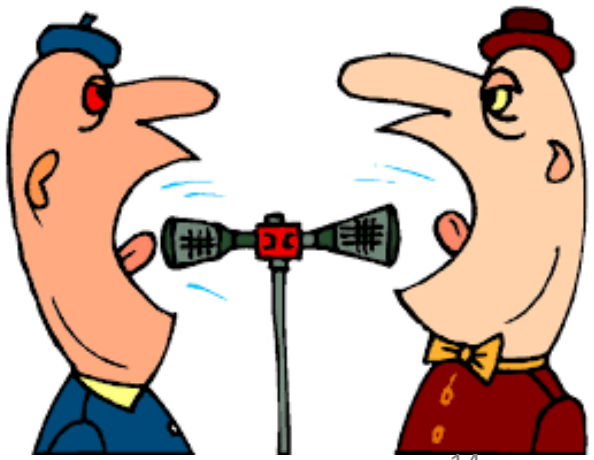
To perform **one task** at **same time** by **two objects**

Concurrent vs Parallel





CONCURRENCY → Sharing of Resources



Concurrent vs Parallel

Concurrency

Tasks start, run and complete in an interleaved fashion

Parallelism

Tasks run simultaneously

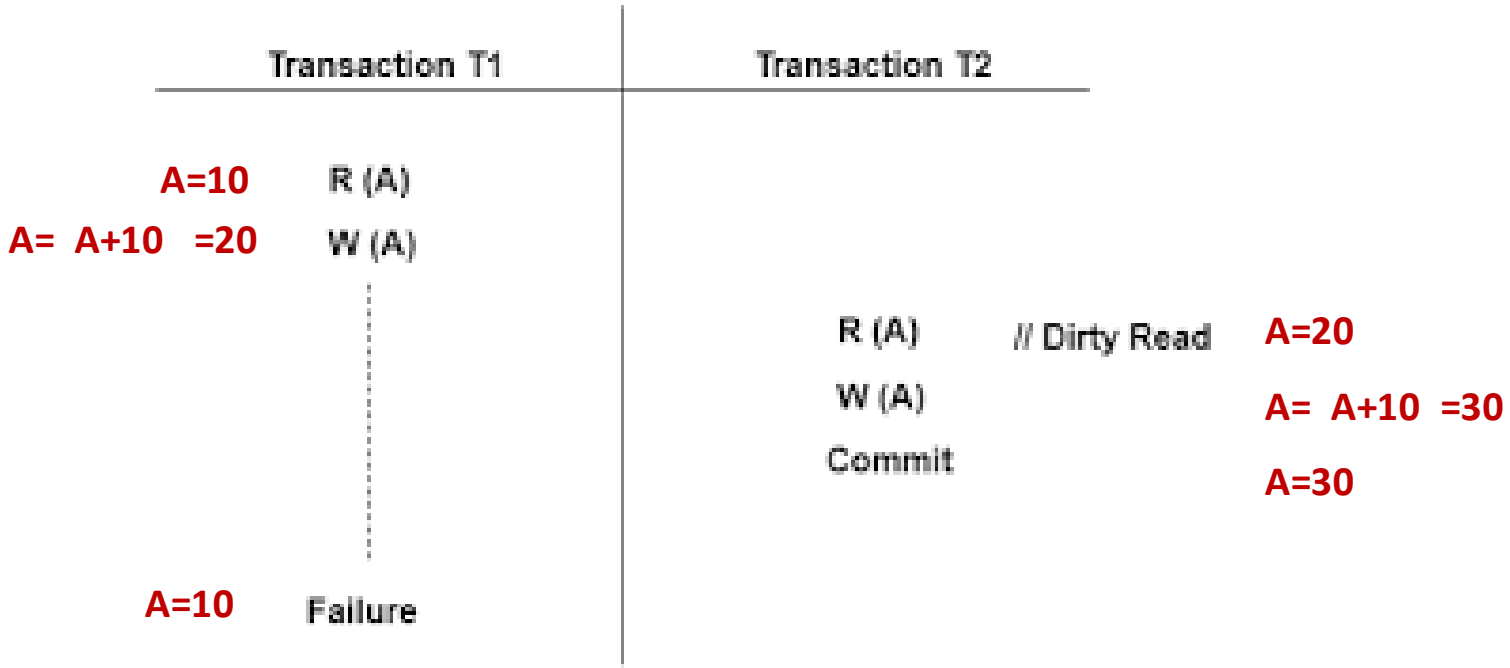
CONCURRENCY IS A PROBLEM

Problems of Concurrency

Problems of Concurrency: DIRTY READ

- **Dirty Read** (Temporary Update)
  - A transaction updates an item, then fails
  - The item is accessed by another transaction before rollback

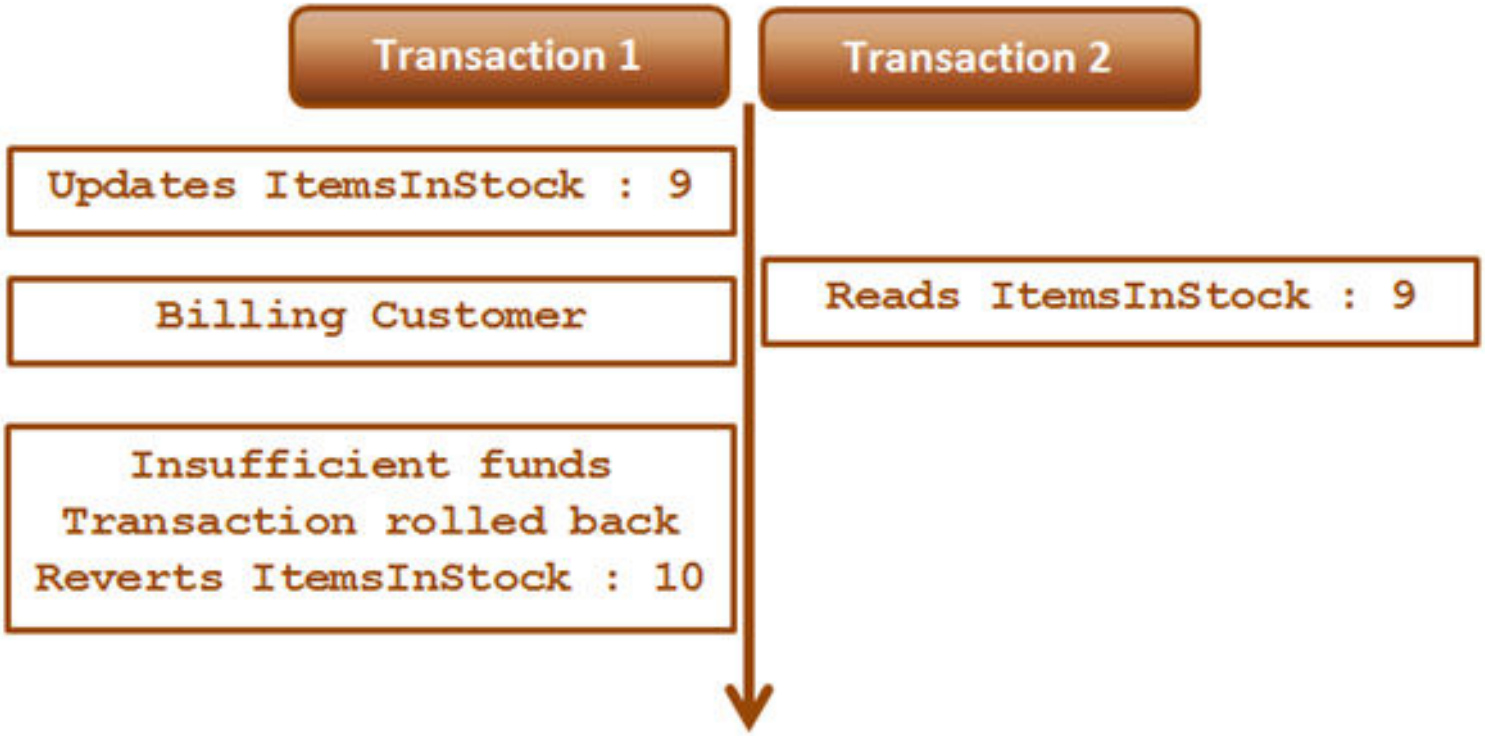
("Temporary Update Problem" or "Uncommitted Dependency Problem")



Problems of Concurrency: UNEPEATED READ

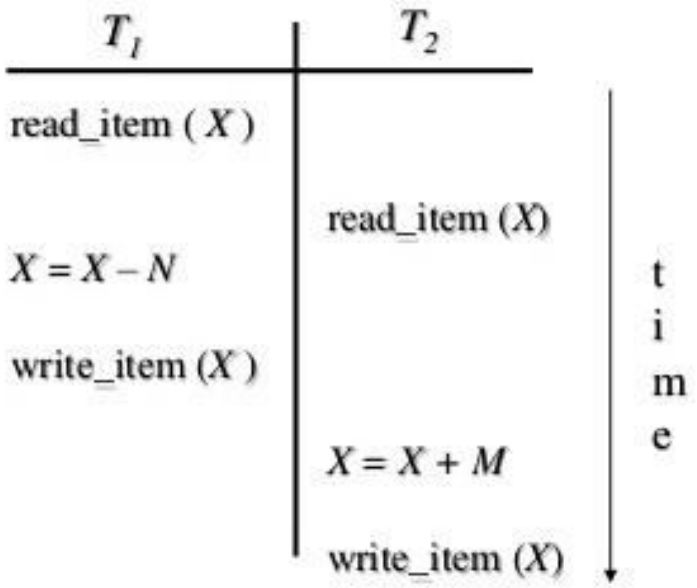
Non-Repeatable Read

- A transactions reads an item twice and gets different values because of concurrent change



Problems of Concurrency: LOST UPDATE

- **Lost update** problem:
  - Two concurrent transactions update same data element
  - One of the updates is lost
    - Overwritten by the other transaction

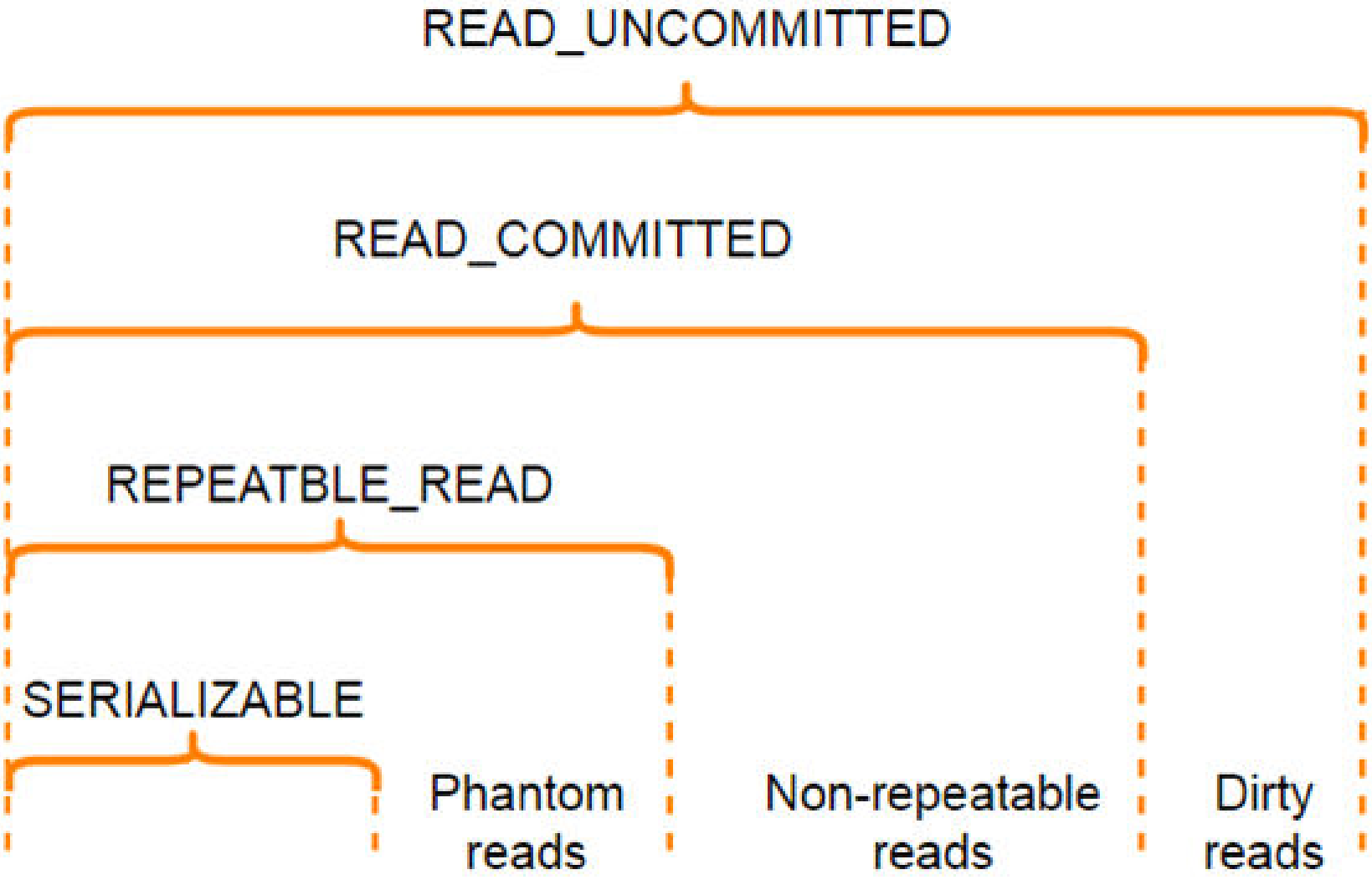


After termination of T_2, $X = X + M$.
T_1's update to X is lost because
T_2 wrote over X

Problems of Concurrency: PHANTOM READ

- **Phantom Read**
  - A transaction executes a query twice, and obtains a different numbers of rows because another transaction inserted new rows meantime

Problems of Concurrency



Problems of Concurrency: LOST UPDATE

CONCURRENCY CONTROL

Concurrency Control: WHAT IS?

Concurrency control is the procedure in DBMS for

managing simultaneous operations

without conflicting with each another

Concurrency Control Protocols

- Locking Protocol
- Two Phase Locking Protocol
- Time Stamp Protocol

Concurrency Control: LOCKING PROTOCOL

Lock Data Items

A lock is a data variable which is associated with a data item

Locks helps to –

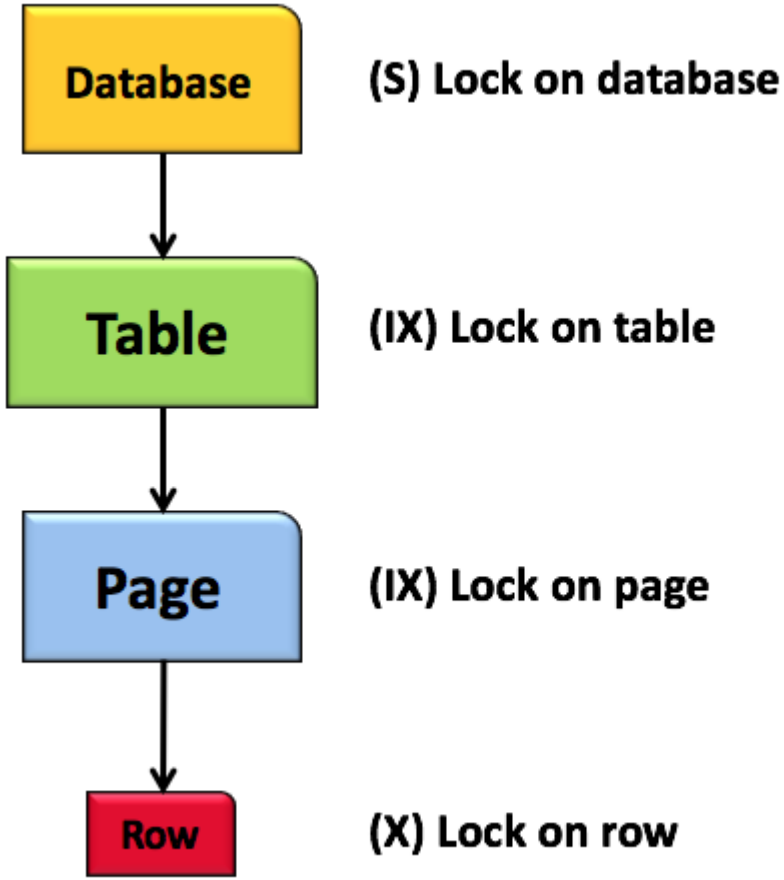
synchronize access to the database items

…. by concurrent transactions.

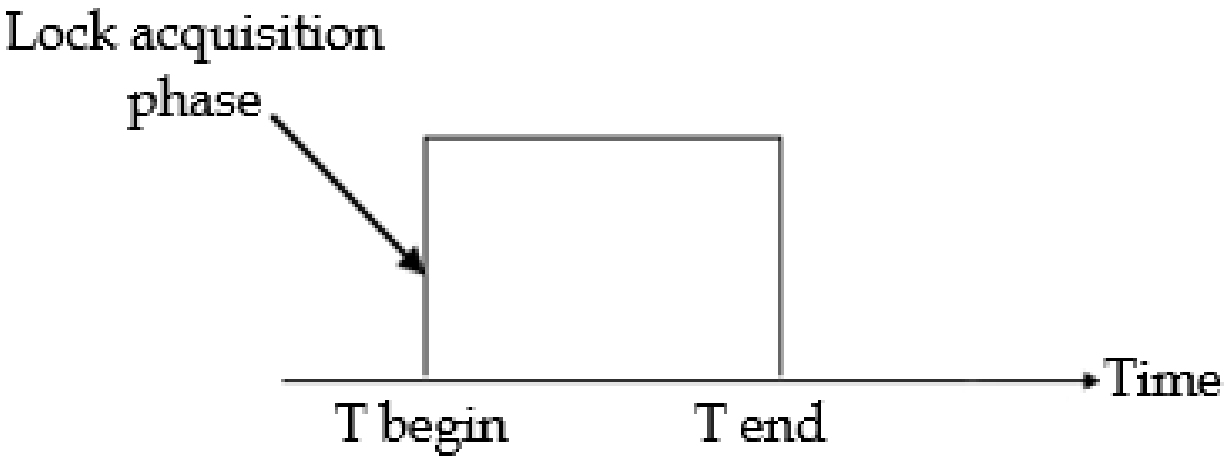
Transactions proceed only once the lock request is granted

Transactions proceed only once the lock request is granted

Concurrency Control: LOCKING PROTOCOL



Concurrency Control: LOCKING PROTOCOL



- **Binary Locks**
- **Shared/ Exclusive Locking**

Binary Locking

A Binary lock on a data item can either locked or unlocked states

An Example Of A Binary Lock

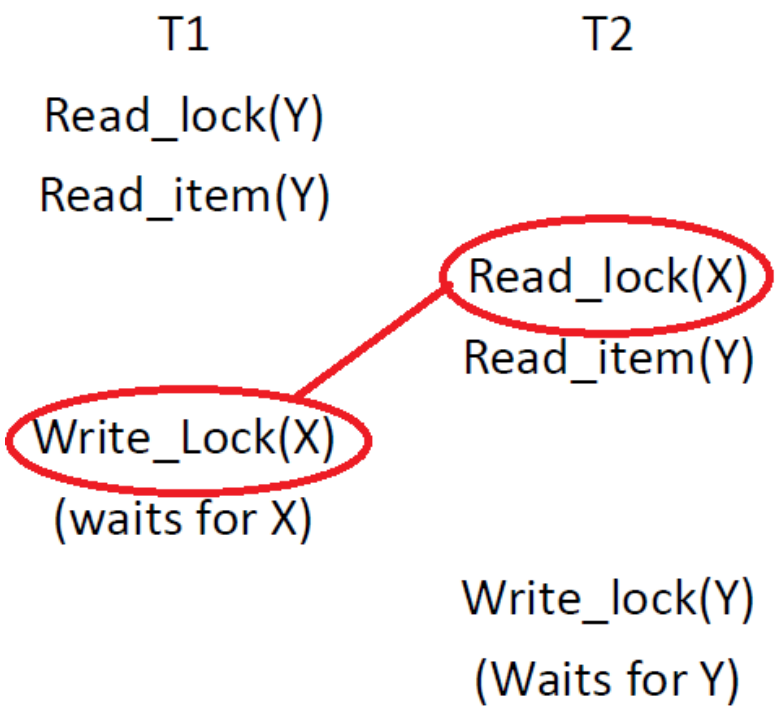
TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Lock PRODUCT	
2	T1	Read PROD_QOH	35
3	T1	PROD_QOH = 35 + 100	
4	T1	Write PROD_QOH	135
5	T1	Unlock PRODUCT	
6	T2	Lock PRODUCT	
7	T2	Read PROD_QOH	135
8	T2	PROD_QOH = 135 – 30	
9	T2	Write PROD_QOH	105
10	T2	Unlock PRODUCT	

Shared / Exclusive Locks

This type of locking mechanism separates the locks based on their uses.

Shared Lock (S)	Exclusive Lock (X)
Read-only lock	a data item can be read as well as written
data item can be shared between transactions	This is exclusive and can't be held concurrently on the same data item
you will never have permission to update data	Transactions may unlock the data item after finishing the 'write' operation

Concurrency Control: LOCKING PROTOCOL



Lock	Shared	Update	Exclusive
Shared (S)	✓	✓	X
Update (U)	✓	X	X
Exclusive (X)	X	X	X

Problem with Locking Protocol

Starvation

Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.

Deadlock

Deadlock refers to a specific situation where two or more processes are waiting for each other to release a resource or more than two processes are waiting for the resource in a circular chain.

Concurrency Control: Two Phase Locking Protocol

This locking protocol divides the execution phase of a transaction into three different parts

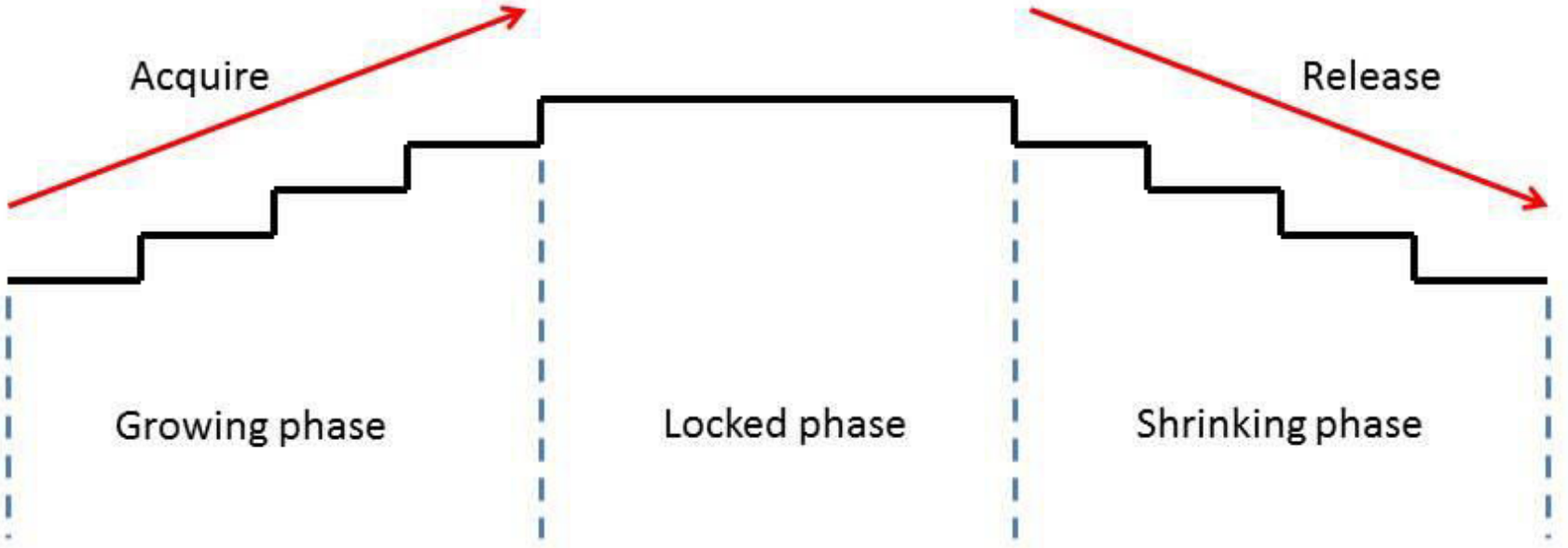
- **Growing Phase**
- **Locked Phase**
- **Shrinking Phase**

Growing Phase: In this phase transaction may obtain locks but may not release any locks.
Shrinking Phase: In this phase, a transaction may release locks but not obtain any new lock

Concurrency Control: Two Phase Locking Protocol

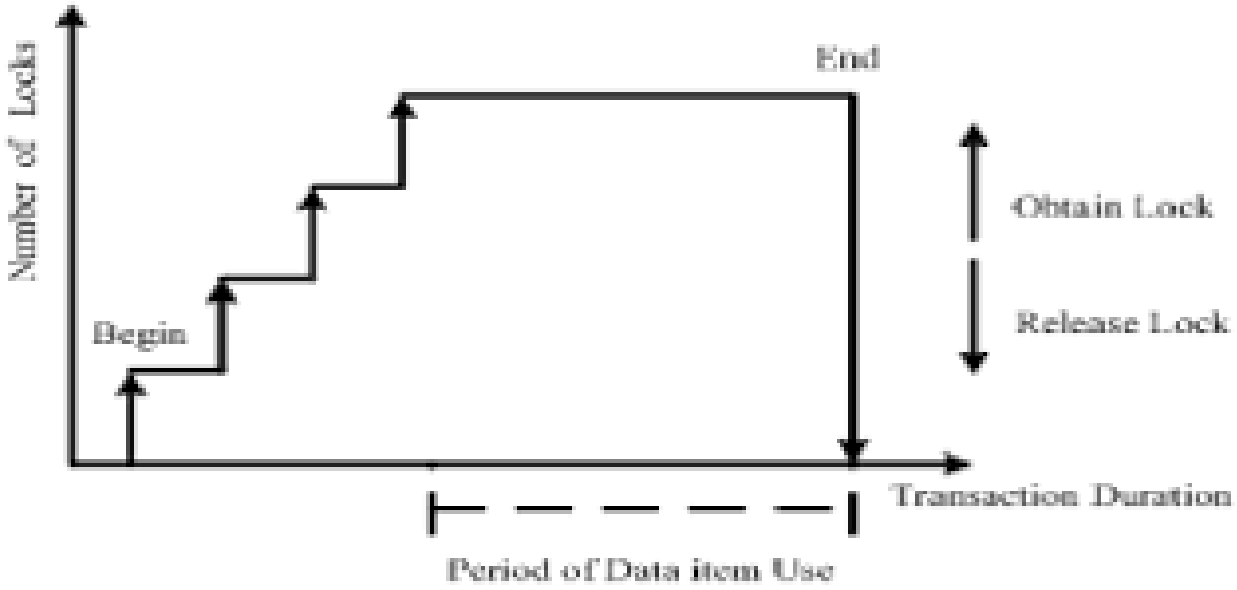
Phase 1: Growing phase –
Acquires all needed locks
No unlocking of data
When all locks aquired,
it's in its "locked pt."

Phase 2: Shrinking phase –
Transaction releases locks
Transaction is not allowed
to get new locks



Strict Two-Phase Locking Method

Transaction Hold Locks until end of the Transaction



Concurrency Control: Two Phase Locking Protocol

- Rule (1)
  - Ti locks tuple A before read/write
- Rule (2)
  - If Ti holds the lock on A, no other transaction is granted the lock on A
- Rule (3):
  - *Growing stage*: Ti may obtain locks, but may not release any lock
  - *Shrinking stage*: Ti my release locks, but may not obtain any new locks

Concurrency Control: Time Stamp based Protocol

Uses a timestamp to serialize the execution of concurrent transactions

This protocol ensures that every conflicting read and write operations are executed in timestamp order.

The protocol uses the System Time or Logical Count as a Timestamp.

Image: Precedence Graph for TS ordering

Timestamp-Based Protocols

- Each transaction is issued a timestamp when it enters the system. If an old transaction T_i has time-stamp TS(T_i), a new transaction T_j is assigned time-stamp TS(T_j) such that TS(T_i) <TS(T_j).
- The protocol manages concurrent execution such that the time-stamps determine the serializability order.
- In order to assure such behavior, the protocol maintains for each data Q two timestamp values:
  - **W-timestamp**(Q) is the largest time-stamp of any transaction that executed **write**(Q) successfully.
  - **R-timestamp**(Q) is the largest time-stamp of any transaction that executed **read**(Q) successfully.

Concurrency Control: Time Stamp based Protocol

In Time Stamp Protocol ensures that any conflicting read and write operations are executed in time stamp order
if not such an operation is rejected and transaction will be rolled back.
The rolled back transaction will be restarted with a new Time Stamp.

T1	T2
Read(A)	
	Write(A)
Write(A)	

Here you could see that conflict is occurring between T2->T1 and it is given that Time Stamp (T1) < Time Stamp (T2) which means it the generated conflict must be resolved in T1->T2. But which is not possible so we rollback transaction T1.

Concurrency Control: Time Stamp based Protocol

Thomos Write Rule

We allow write-write conflict by ignoring.

T1	T2
	Read(A)
Write(A)	
	Write(A)

Note The conflict occurred says T1->T2 and it is given that Time Stamp (T2) < Time Stamp (T1) which means it the conflict can't be resolved but

Thomos write rule says that we can ignore the write done by T1 as it has been overwritten by T2 later.

Concurrency Control: Time Stamp based Protocol

Advantages

1. Serializability
2. Ensures freedom from dead lock

Disadvantage

Starvation may occur due to continuously getting aborted and restarting the transaction.