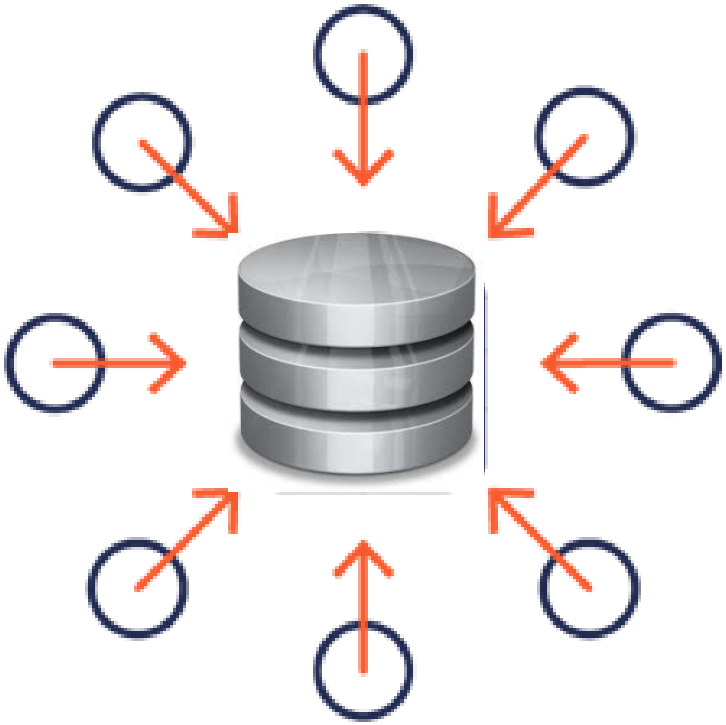
Transactions

What is DB Transactions?

Transaction is a process involving database queries and/or modification.



Why Transactions?

Database systems are normally being accessed by many users or processes at the same time

Example-ATM

You and your Friend each take Rs10000 from different ATM's at about the same time.

DBMS must make sure none of the account deduction is lost.

ACID Transactions

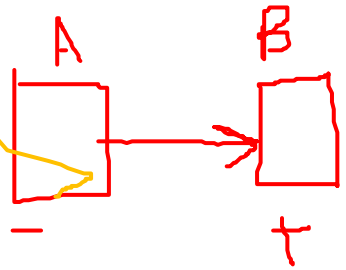
Atomic	• Whole transaction or none is done.
Consistent	• Database constraints preserved.
Isolated	• It appears to the user as if only one process executes at a time.
Durable	• Effects of a process survive a crash.

Optional: weaker forms of transactions are often supported as well.

Example of *Fund Transfer*

- Transaction to transfer $50 from account **A** to account **B**:

1. **read**(*A*)
2. *A* := *A* – 50
3. **write**(*A*)
4. **read**(*B*)
5. ***B* := *B* + 50**
6. **write**(*B*)



- **Atomicity requirement** :
  - if the transaction **fails** after step 3 and before step 6,
    - the **system** should **ensure** that :
      - its **updates** are *not reflected* in the database,
      - else an *inconsistency* will result.

Example of *Fund Transfer*

- Transaction to transfer $50 from account **A** to account **B**:
  1. **read**(A)
  2. $A := A - 50$
  3. **write**(A)
  4. **read**(B)
  5. $B := B + 50$
  6. **write**(B)

- **Consistency requirement** :
  - the **sum** of **A** and **B** is:
    - **unchanged** by the execution of the transaction.

Example of *Fund Transfer*

- Transaction to transfer $50 from account **A** to account **B**:
  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B*)
- **Isolation requirement** —
  - if between steps 3 and 6, another transaction is allowed to access the partially updated database,
    - it will see an **inconsistent database** (the sum *A* + *B* will be less than it should be).
  - Isolation can be **ensured** trivially by:
    - running transactions **serially**, that is **one** after the **other**.
  - *However*, executing multiple *transactions* **concurrently** has significant **benefits**.

Example of *Fund Transfer*

- Transaction to transfer $50 from account **A** to account **B**:
  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B*)

- **Durability requirement** :
  - once the user has been notified that the transaction has **completed** :
    - (i.e., the transfer of the $50 has taken place),
    - the **updates** to the database by the transaction **must persist**
      - despite *failures*.

T-SQL AND Transactions

SQL has following transaction modes.

- Autocommit transactions
  - Each individual SQL statement = transaction.

- Explicit transactions

  BEGIN TRANSACTION

  [SQL statements]

  COMMIT or ROLLBACK

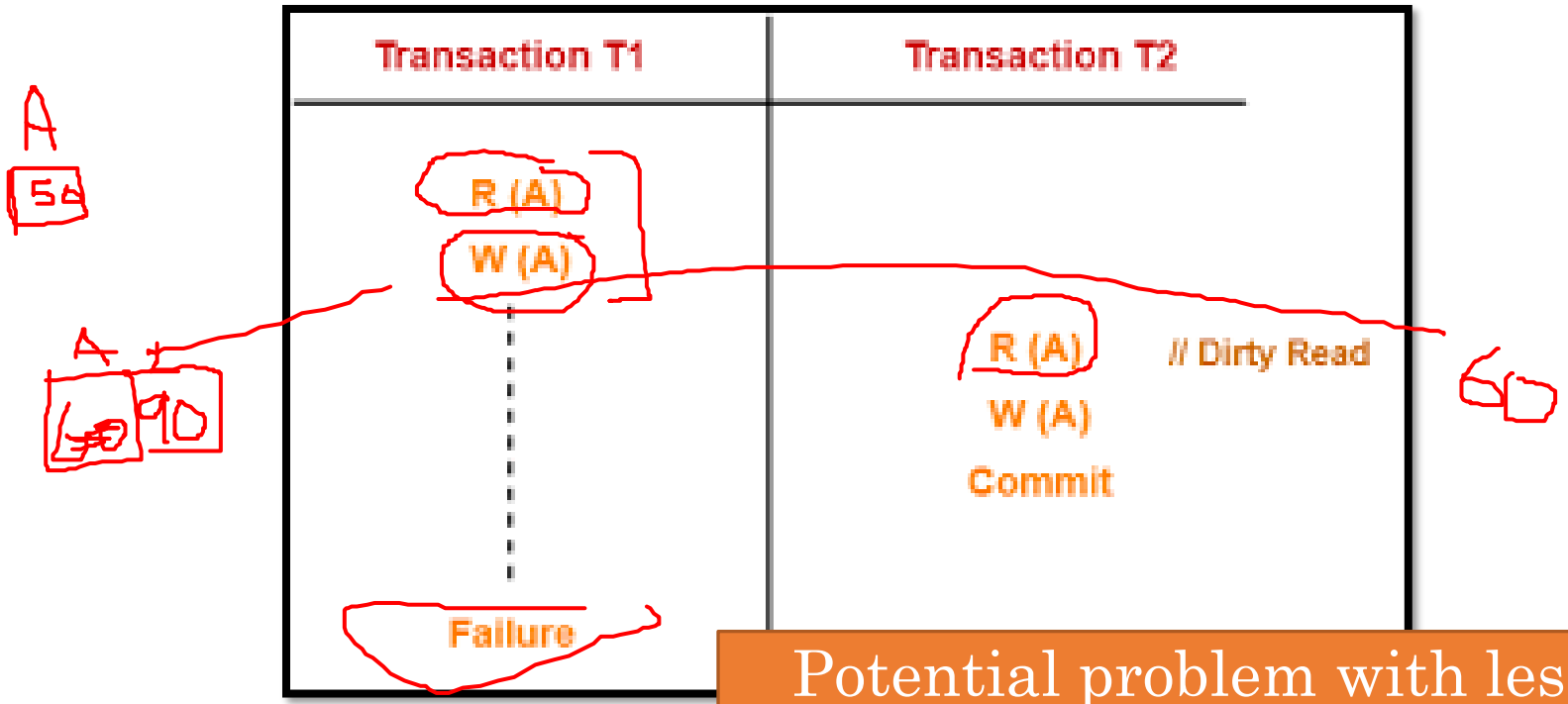
Transaction Support in TSQL

- BEGIN TRAN
- UPDATE Department
- SET Mgr_ssn = 123456789
- WHERE DNumber = 1
- UPDATE Department
- SET Mgr_start_date = '1981-06-19'
- WHERE Dnumber = 1
- COMMIT TRAN

Transaction Support in SQL

Dirty Read:

Reading a value that was written by a uncommitted transaction.

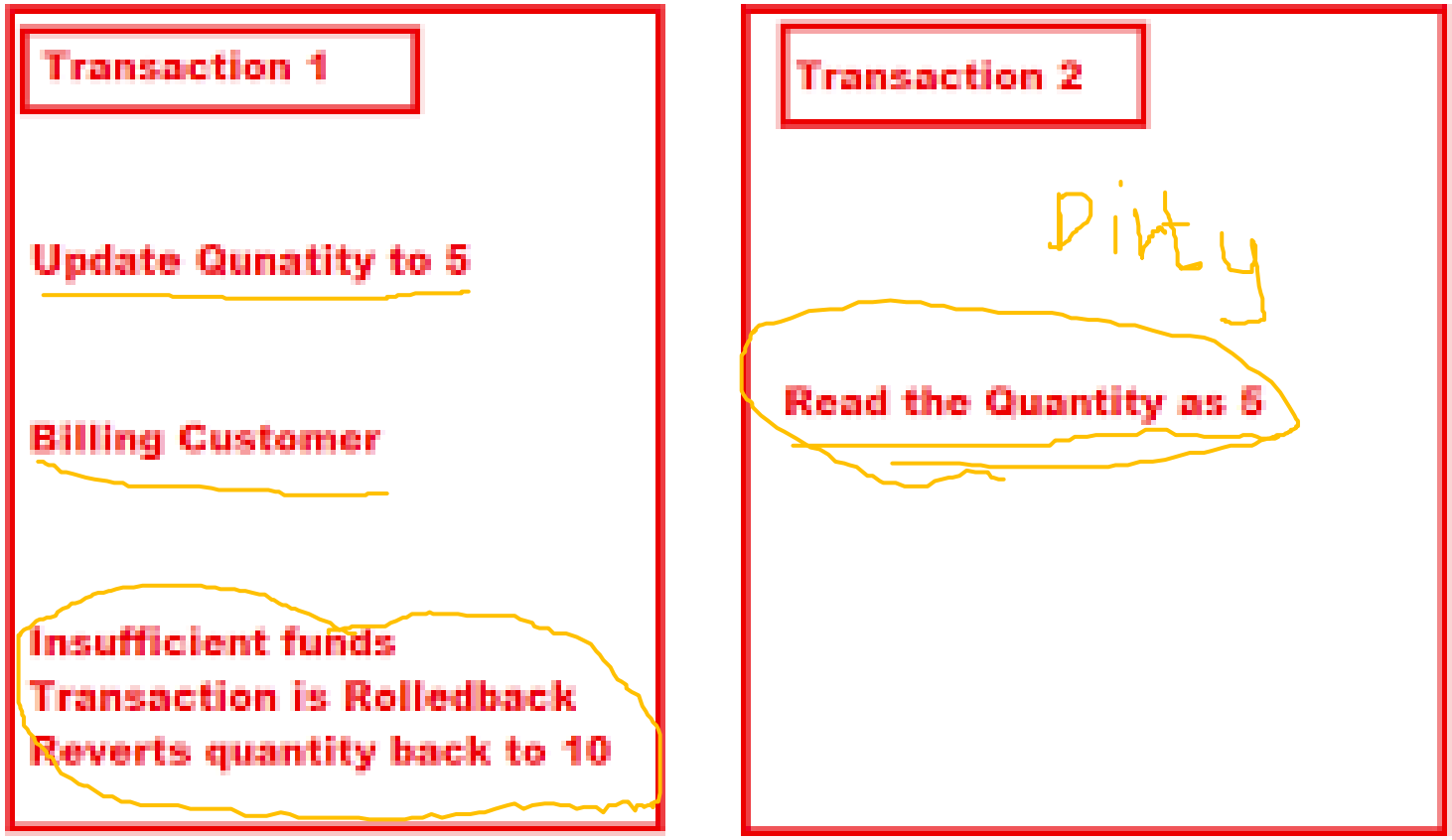


Potential problem with less restricted isolation levels

Transaction Support in SQL

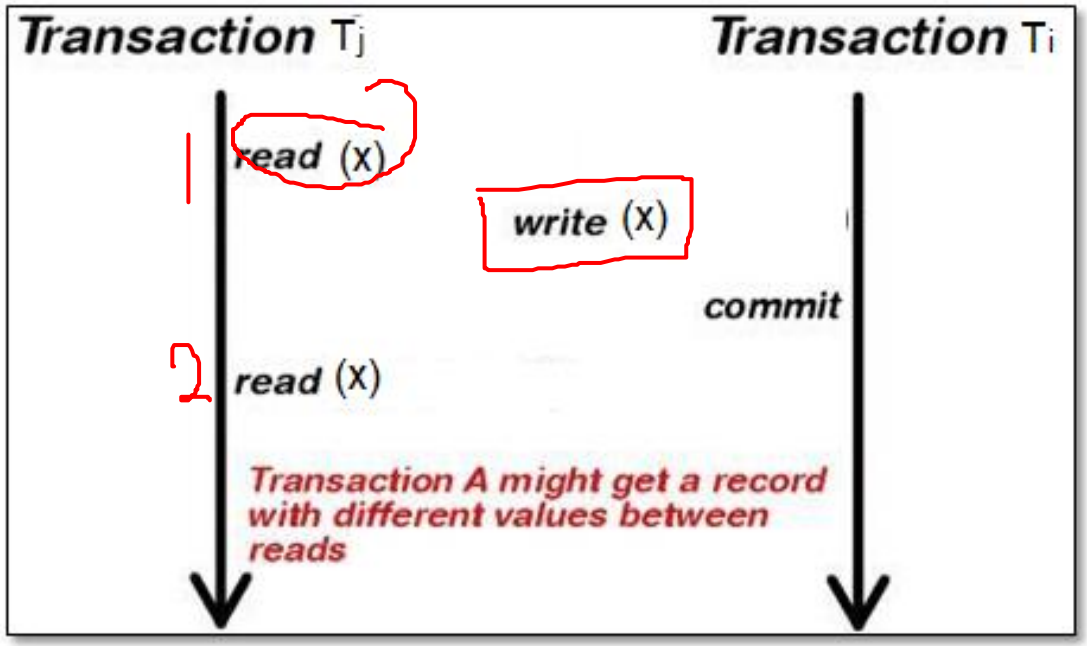
Dirty Read:

Reading a value that was written by a uncommitted transaction.

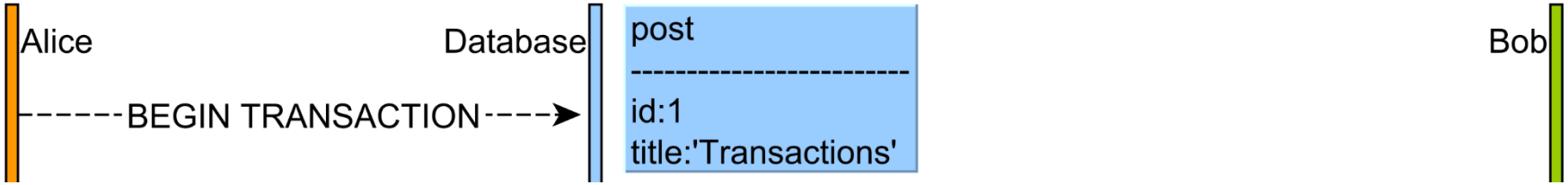


Transaction Support in SQL

- **Nonrepeatable Read**
  - A transaction T_i write a new value between multiple reads of transaction T_j.
    - T_j reads a given value X from a table.
    - T_i later updates X and T_j reads that value again, Tj will see a different value.

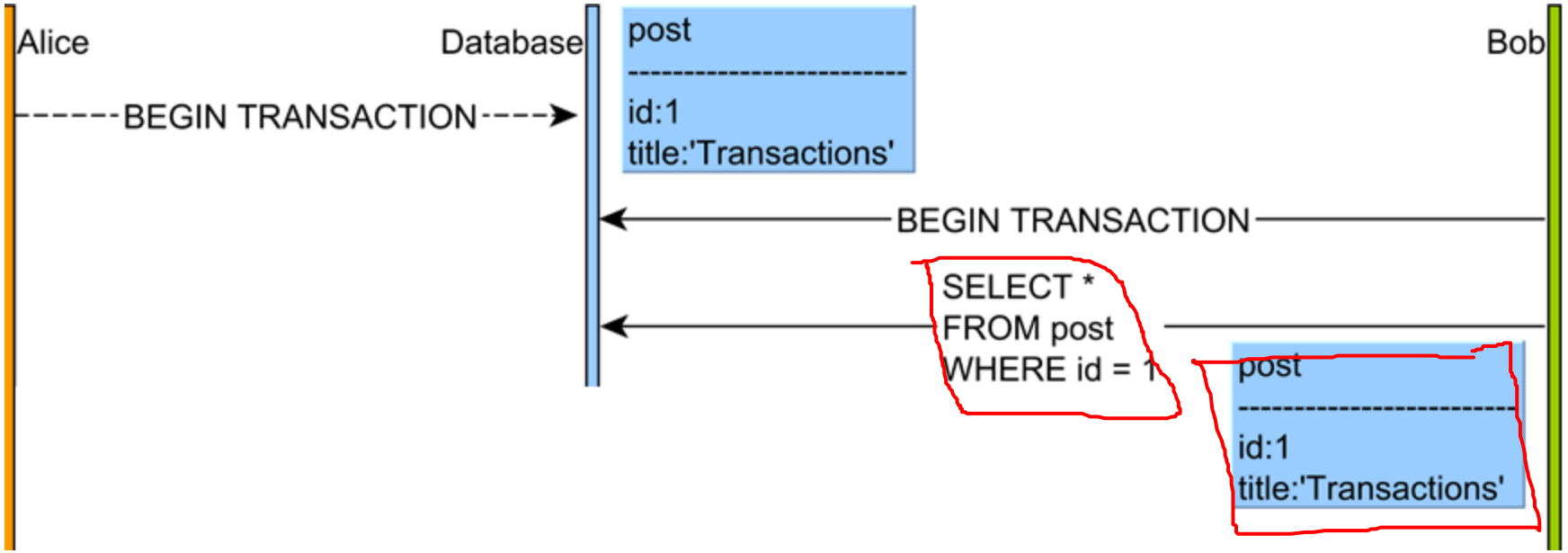


Nonrepeatable Read



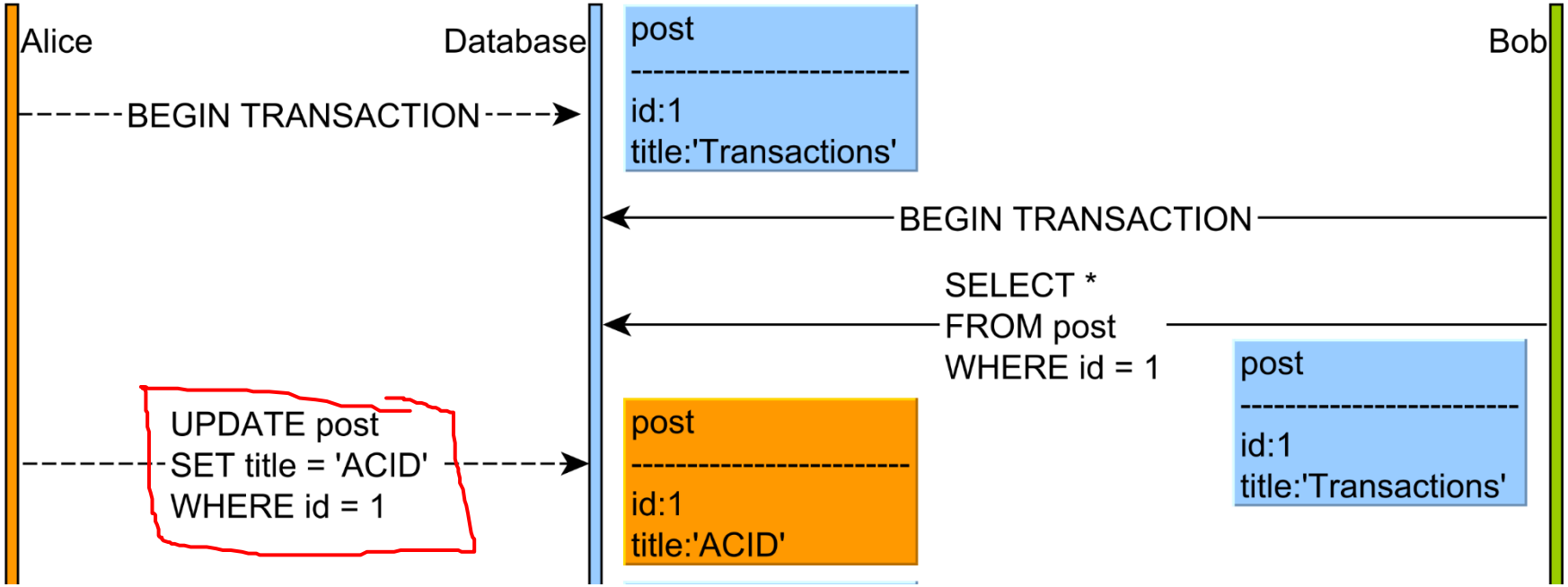
https://vladmihalcea.com/non-repeatable-read/

Nonrepeatable Read



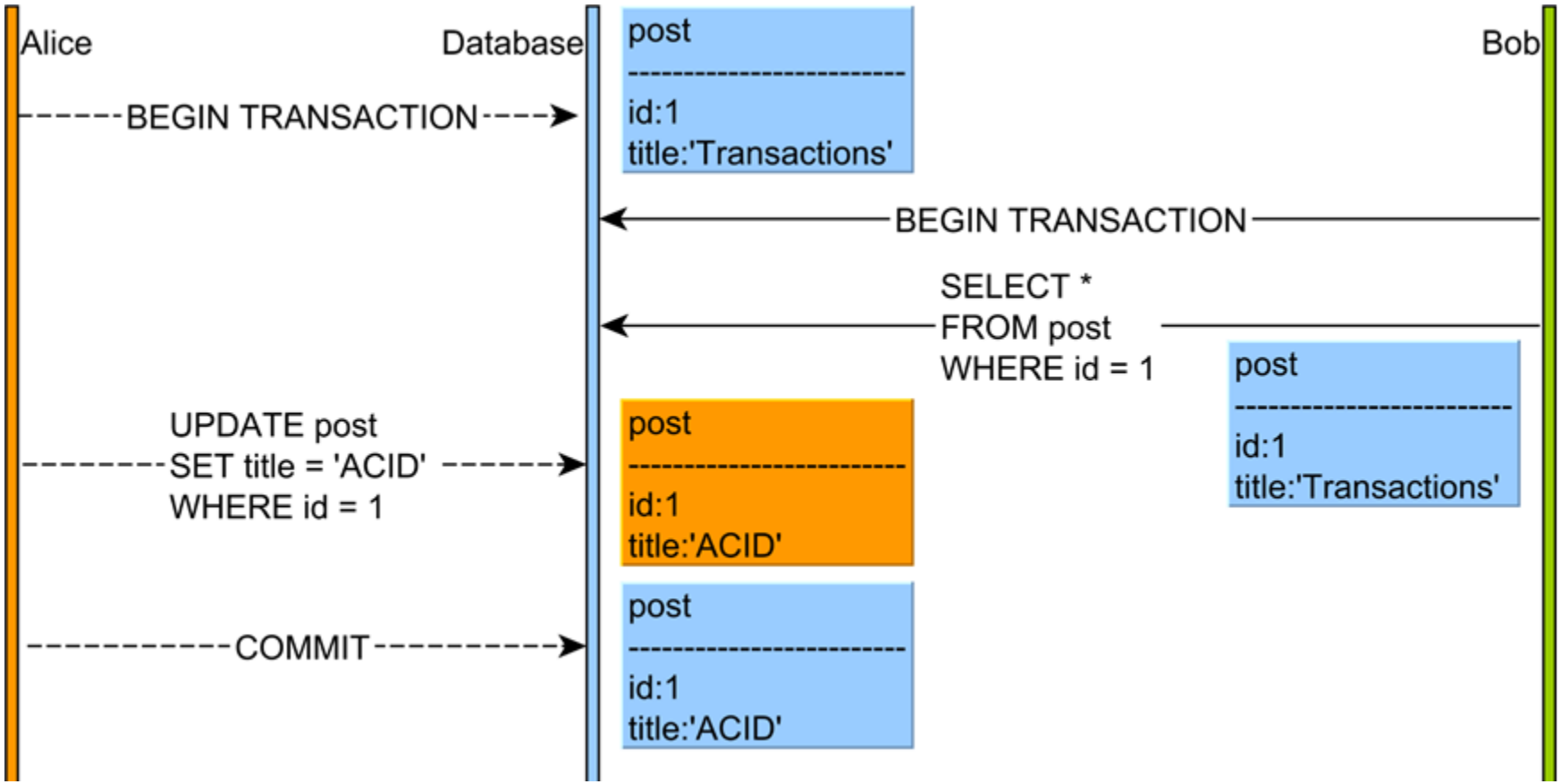
https://vladmihalcea.com/non-repeatable-read/

Nonrepeatable Read



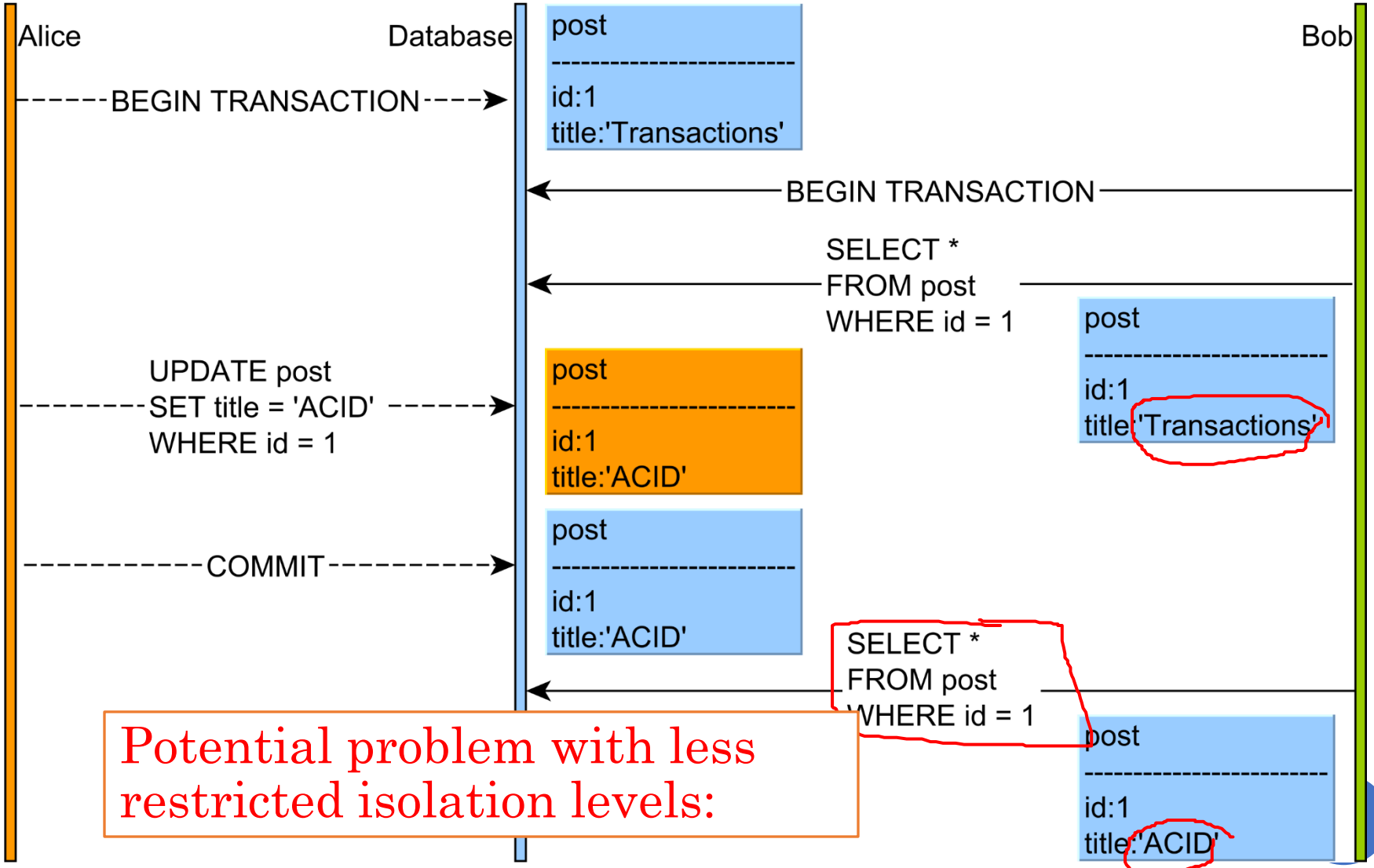
https://vladmihalcea.com/non-repeatable-read/

Nonrepeatable Read



https://vladmihalcea.com/non-repeatable-read/

Nonrepeatable Read



Potential problem with less restricted isolation levels:

https://vladmihalcea.com/non-repeatable-read/

Transaction Support in SQL

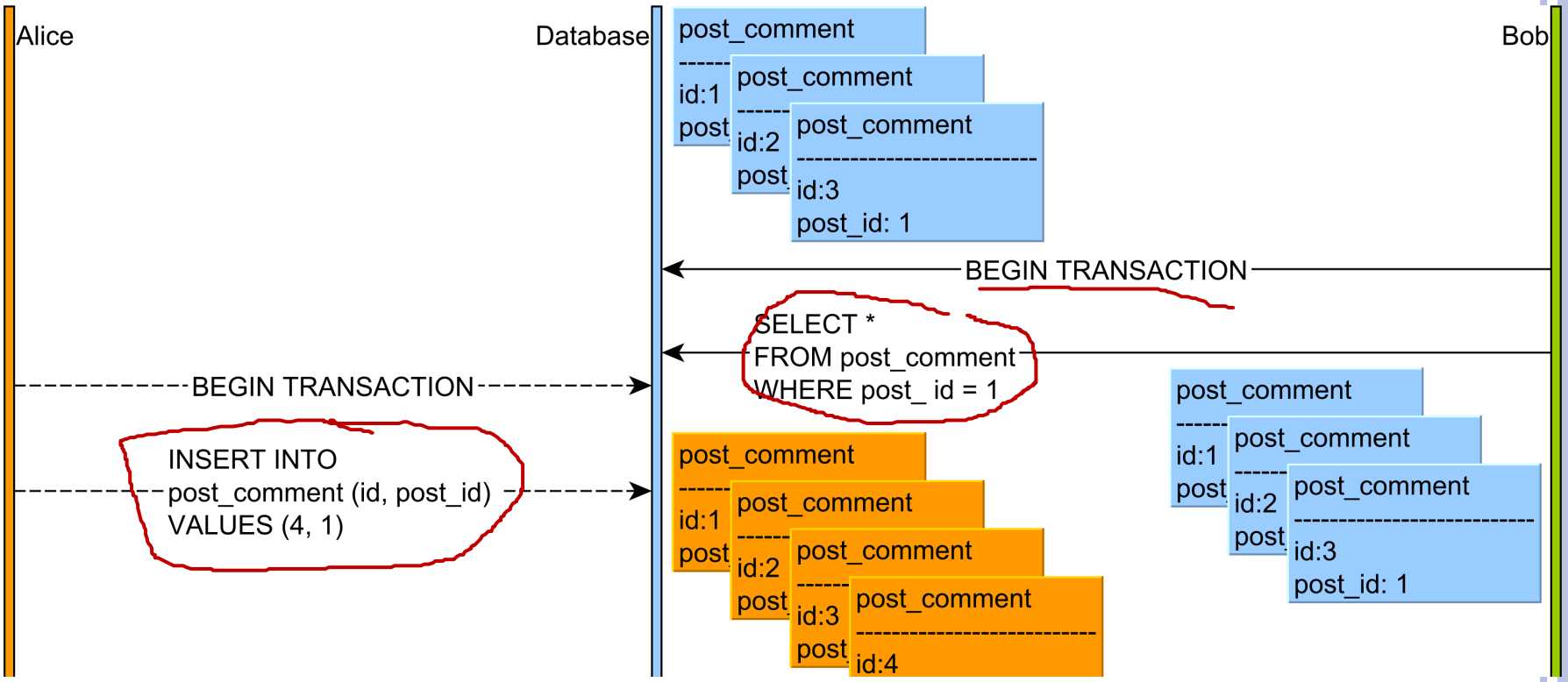
- **Phantoms**
  - New rows being read using the same read with a condition.

Potential problem with less restricted isolation levels

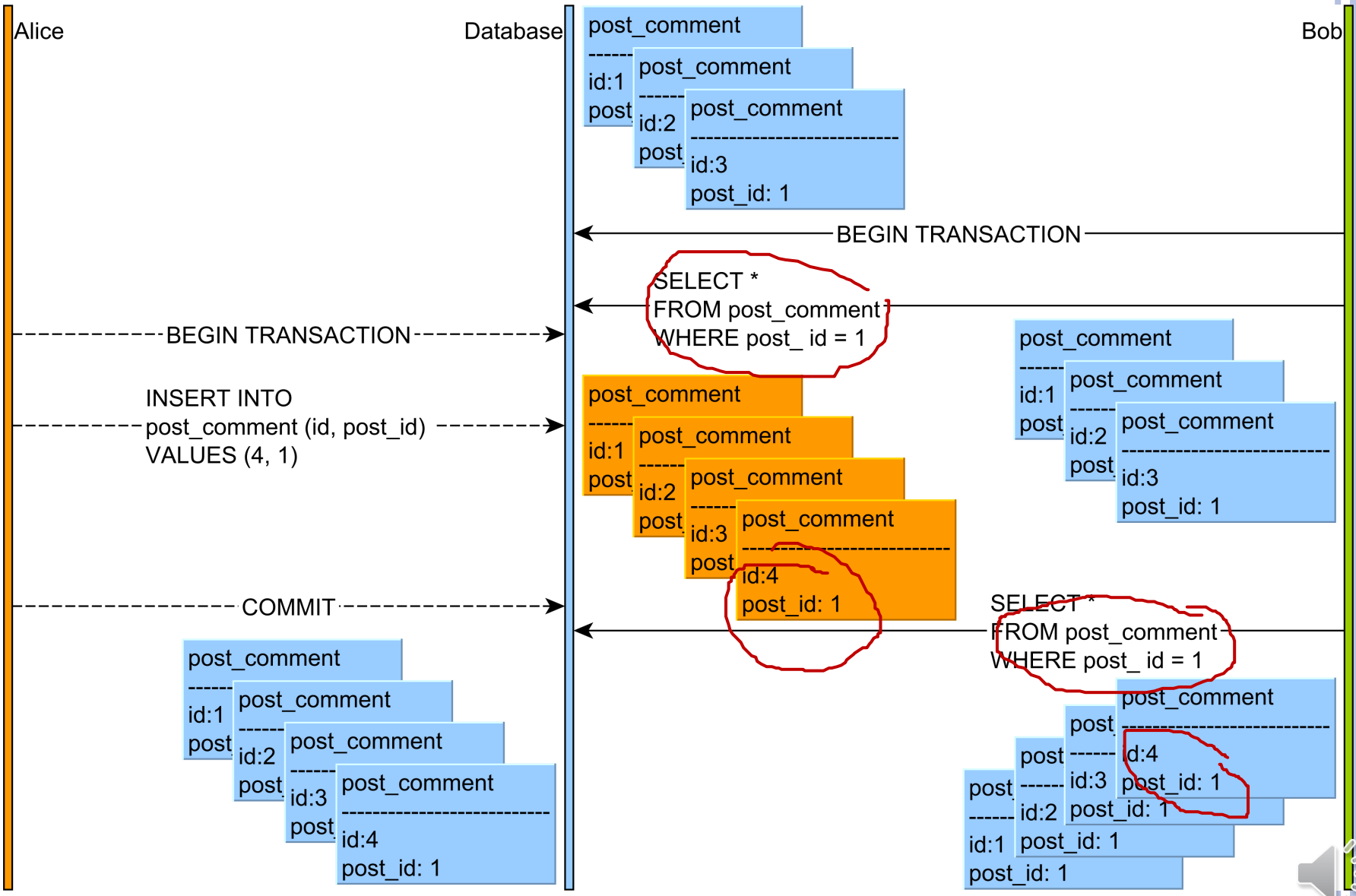
Phantom


Alice
Database
post_comment
---
id:1
post
post_comment
---
id:2
post
post_comment
---
id:3
post_id: 1
Bob

Phantom


Alice
Database
Bob
post_comment
id:1
post
post_comment
id:2
post
post_comment
id:3
post_id: 1
BEGIN TRANSACTION
SELECT *
FROM post_comment
WHERE post_ id = 1
BEGIN TRANSACTION
INSERT INTO
post_comment (id, post_id)
VALUES (4, 1)
post_comment
id:1
post
post_comment
id:2
post
post_comment
id:3
post
post_comment
id:4
post_comment
id:1
post
post_comment
id:2
post
post_comment
id:3
post_id: 1

Phantom


Alice
Database
Bob
post_comment
id:1
post_comment
id:2
post_comment
id:3
post_id: 1
BEGIN TRANSACTION
SELECT *
FROM post_comment
WHERE post_ id = 1
BEGIN TRANSACTION
INSERT INTO
post_comment (id, post_id)
VALUES (4, 1)
post_comment
id:4
post_id: 1
COMMIT
SELECT *
FROM post_comment
WHERE post_ id = 1

Transaction Support in SQL

- Potential problem with lower isolation levels (contd.):
  - **Phantoms**
    - New rows being read using the same read with a condition.
    - **Example:**
      - T_1 read a set of rows from a table,
        - on some condition specified in the SQL WHERE clause.
      - T_2 inserts a new row that also satisfies the WHERE clause condition of T_1, into the table used by T_1.
      - If T_1 is repeated, then T_1 will see a row that previously did not exist, called a **phantom**.

Transaction Support in TSQL

Table 21.1 Possible Violations Based on Isolation Levels as Defined in SQL

	Type of Violation		
Isolation Level	**Dirty Read**	**Nonrepeatable Read**	**Phantom**
READ UNCOMMITTED			
READ COMMITTED			
REPEATABLE READ			
SERIALIZABLE	No	No	No

Transaction Support in TSQL

Table 21.1 Possible Violations Based on Isolation Levels as Defined in SQL

	Type of Violation		
Isolation Level	**Dirty Read**	**Nonrepeatable Read**	**Phantom**
READ UNCOMMITTED			
READ COMMITTED			
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

Transaction Support in TSQL

Table 21.1 Possible Violations Based on Isolation Levels as Defined in SQL

	Type of Violation		
Isolation Level	**Dirty Read**	**Nonrepeatable Read**	**Phantom**
READ UNCOMMITTED			
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

Transaction Support in TSQL

Table 21.1 Possible Violations Based on Isolation Levels as Defined in SQL

	Type of Violation		
Isolation Level	**Dirty Read**	**Nonrepeatable Read**	**Phantom**
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

Transaction Support in TSQL

1. "Dirty reads"

   SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

2. "Committed reads"

   SET TRANSACTION ISOLATION LEVEL READ COMMITTED

3. "Repeatable reads"

   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

4. Serializable transactions (default):

   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

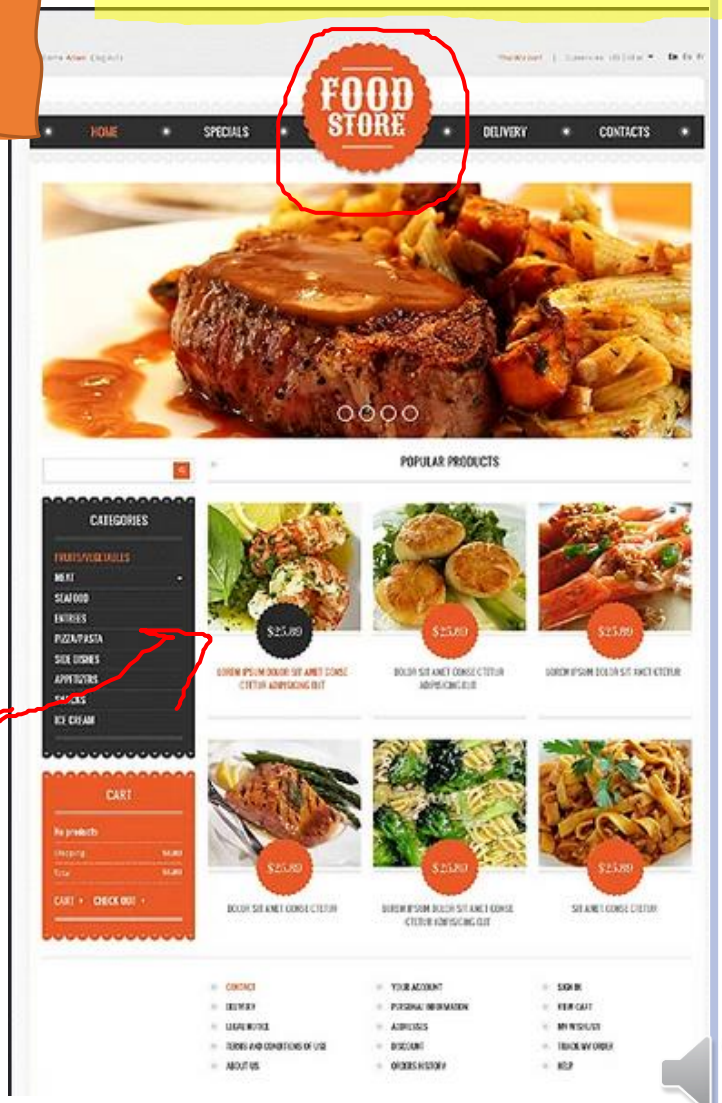
Transactions Example

Transactions Example

Web For FoOD CouRt

Relation

Sells(shop, item, price)

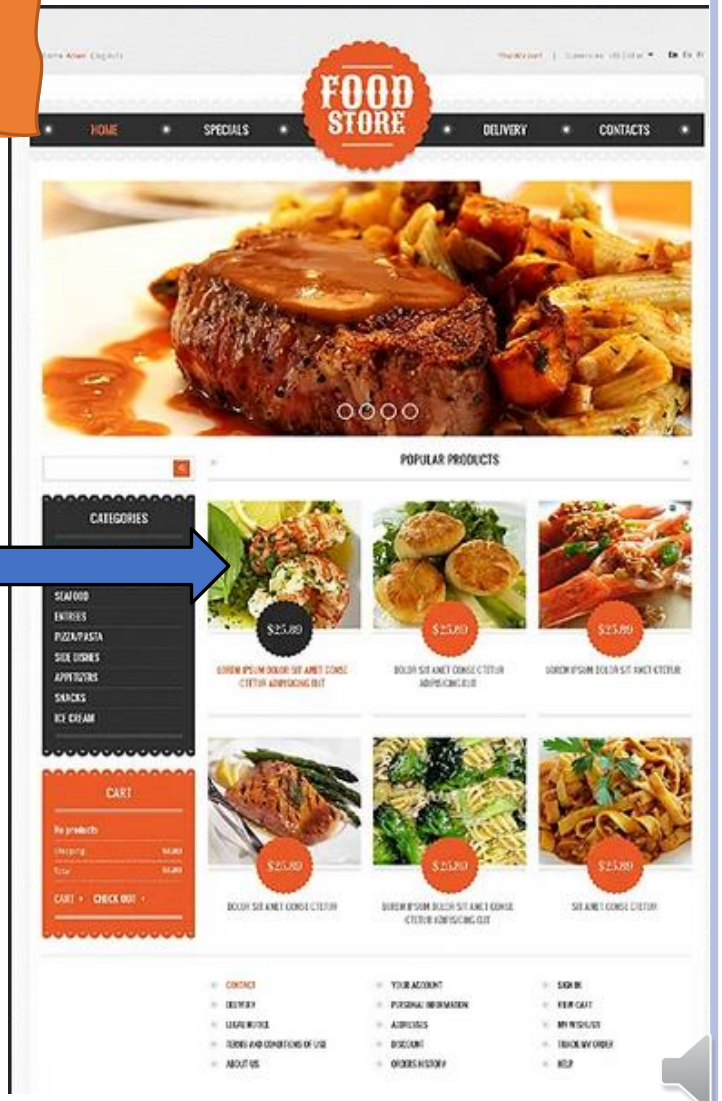


Transactions Example

Web For FoOD CouRt

Relation

Sells(shop, item, price)



Scenario

Ahmad has a shop in Food Store

Ahmad sells Pizza slice for Rs 200 and Sprite for Rs 50.

Transactions Example

Web For FoOD CouRt

Relation

Sells(shop, item, price)

Scenario

Tania is exploring Food Store's Website

Tania visits Ahmad's Shop Page

Tania checks highest & lowest price Ahmad charges.

Transactions Example

Web For FoOD CouRt

Relation

Sells(shop, item, price)

Scenario

Tania is exploring Food Store's Website

Tania visits Ahmad's Shop Page

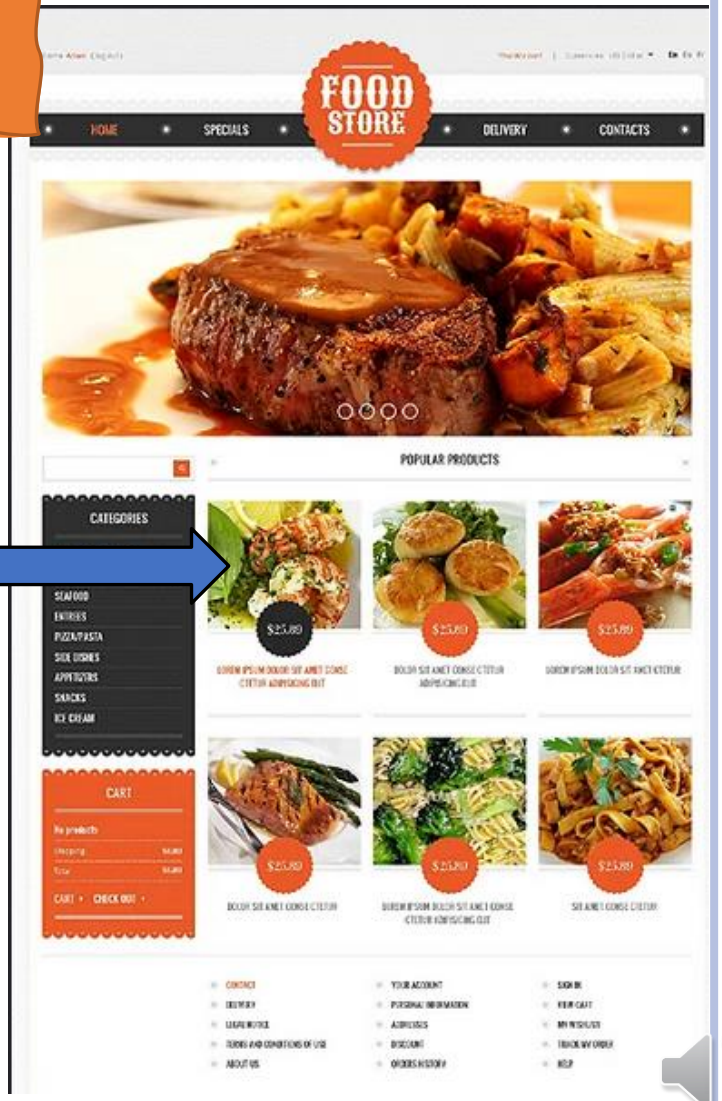
Tania query **Sells** for the highest & lowest price Ahmad charges

Transactions Example

Web For FoOD CouRt

Relation

Sells(shop, item, price)



Scenario

Ahmad shops is not performing well

Ahmad decides to stop selling Pizza slice for Rs 200 and sell Biryani instead !!! For 250Rs

Customer's Query

Tania(customer) executes the following two SQL statements called (min) and (max).

(max) SELECT MAX(price) FROM Sells
WHERE shop = 'Ahmad"s shop'

(min) SELECT MIN(price) FROM Sells
WHERE shop = 'Ahmad"s shop'

Shop Keeper's Query

At about the same time, Ahmad executes the following steps: (del) and (ins).

(del) DELETE FROM Sells
WHERE shop = 'Ahmad"s shop'

(ins) INSERT INTO Sells
VALUES('Ahmad"s shop', 'Biryani', 250.00)

Interleaving of Statements

- The statement (max) must come before (min), and
- The statement (del) must come before (ins),

Interleaving of Statements

- The statement (max) must come before (min), and
- The statement (del) must come before (ins),

- There are no other constraints on the order of these statements.
- Unless we group Tania's and/or Ahmad's statements into transactions.

Example: Strange Interleaving

- Suppose the steps execute in the order (max)(del)(ins)(min).

Ahmad's Prices:	{50,200}	{50,200}	{250}	{250}
Statement:	(max)	(del)	(ins)	(min)
Result:	200			250

Example: Strange Interleaving

- Suppose the steps execute in the order (max)(del)(ins)(min).

Ahmad's Prices:	{50,200}	{50,200}	{250}	{250}
Statement:	(max)	(del)	(ins)	(min)
Result:	200			250

- **Tania sees MAX < MIN!**

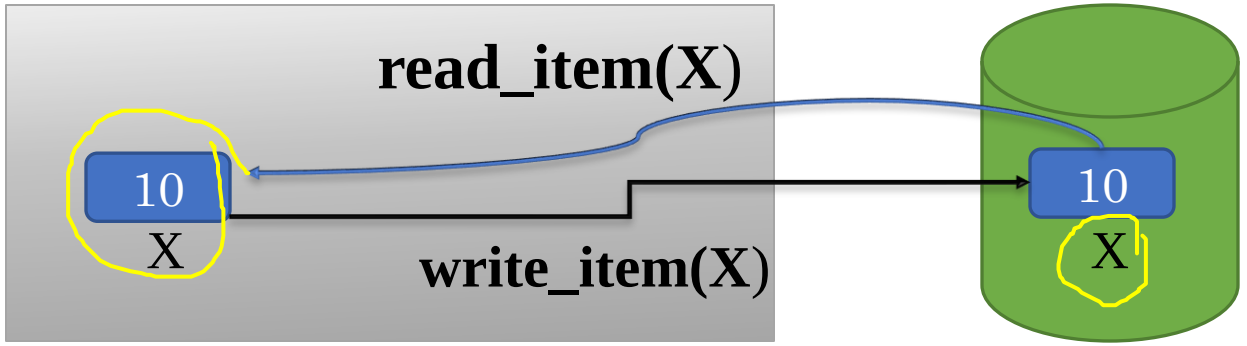
Fixing the Problem by Using Transactions

- Solution: Group Tania's statements (max)(min) into one transaction
  - Now, she cannot see this inconsistency.

- She sees Ahmad's prices at some fixed time.
  - Either before or after he changes prices, or in the middle, but the MAX and MIN are computed from the same prices.

Transaction Processing

- Basic operations in a DB are **read** and **write**
  - **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.



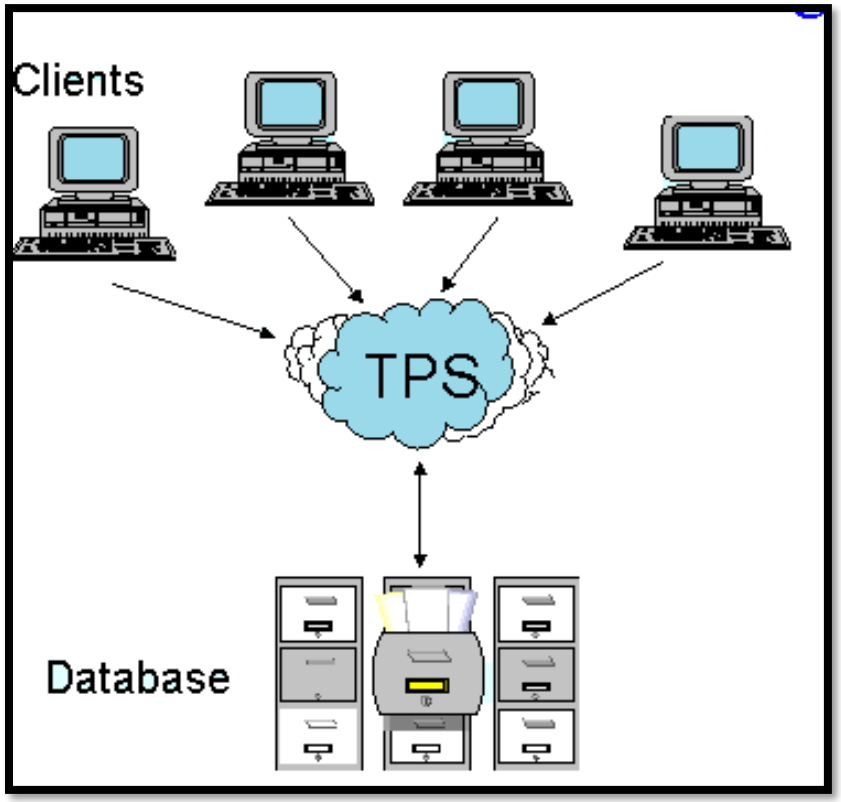
Sample Transactions

(a) T_1

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

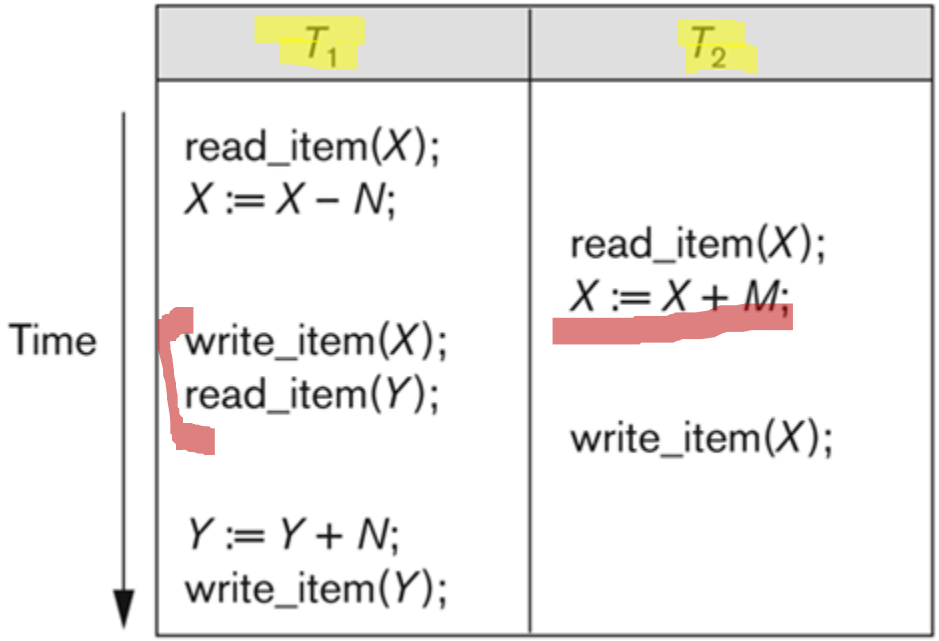
(b) T_2

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



Issues in Transaction Processing

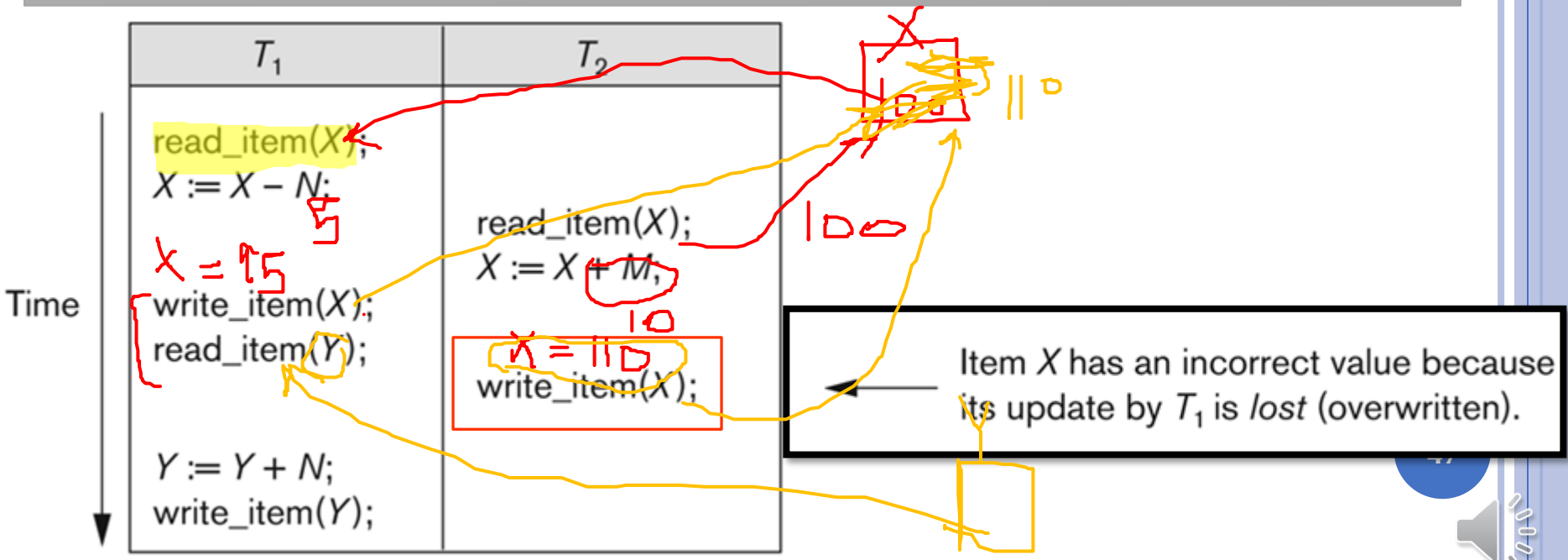
Why Concurrency Control is needed?



Issues in Transaction Processing

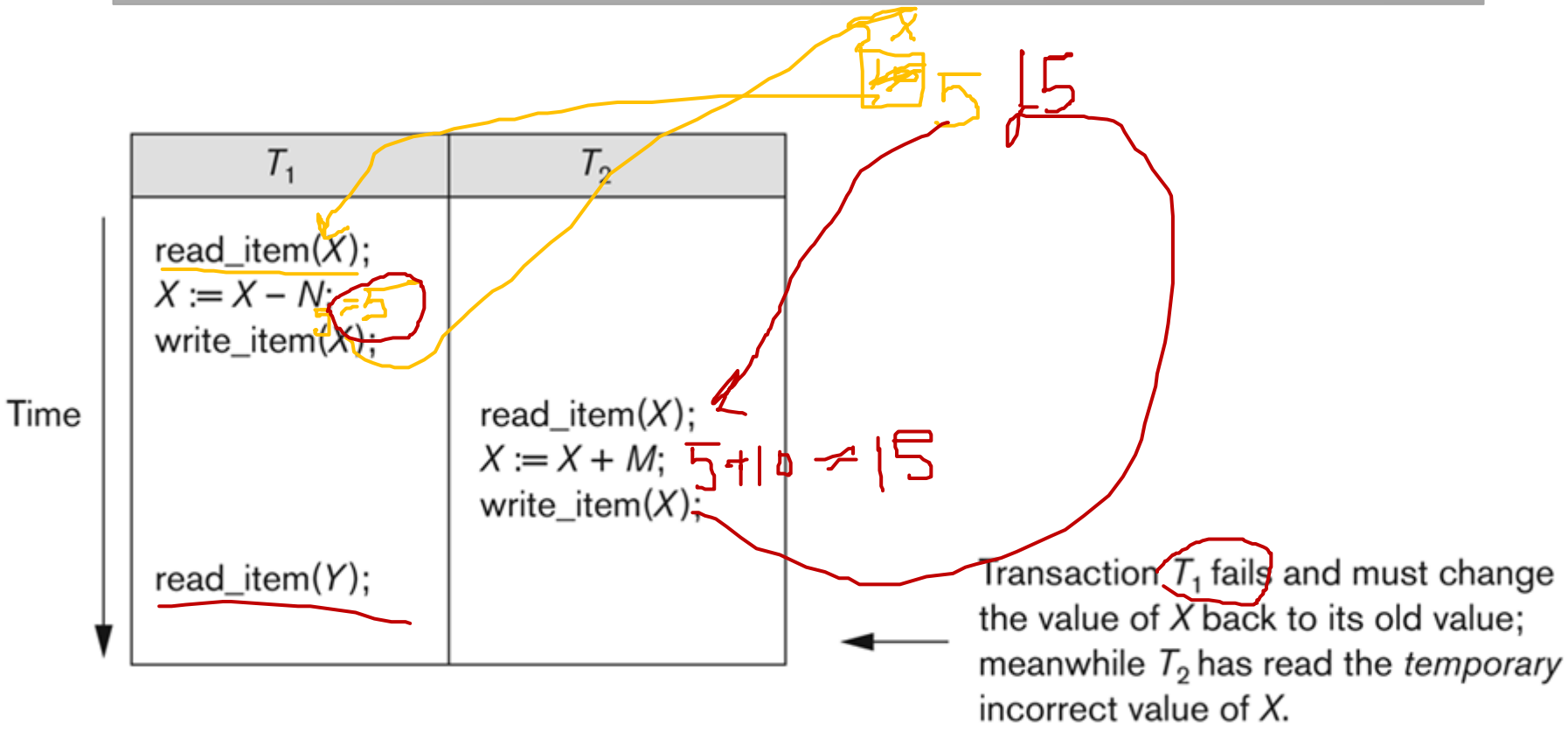
Why Concurrency Control is needed?

- **The Lost Update Problem**
  - Two transactions (that access the same DB items) have their operations interleaved in a way that makes the value of some database item incorrect.



Issues in Transaction Processing

Temporary Update (or Dirty Read) Problem



Issues in Transaction Processing

The Incorrect Summary Problem

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*);	

← T_3 reads *X* after *N* is subtracted and reads *Y* before *N* is added; a wrong summary is the result (off by *N*).

State transition diagram

Recovery manager keeps track of the following operations

- **Begin_transaction**
- **Read or Write**
- **End_transaction**
- **Commit**
- **Rollback (or abort)**
- **Undo**
- **Redo**

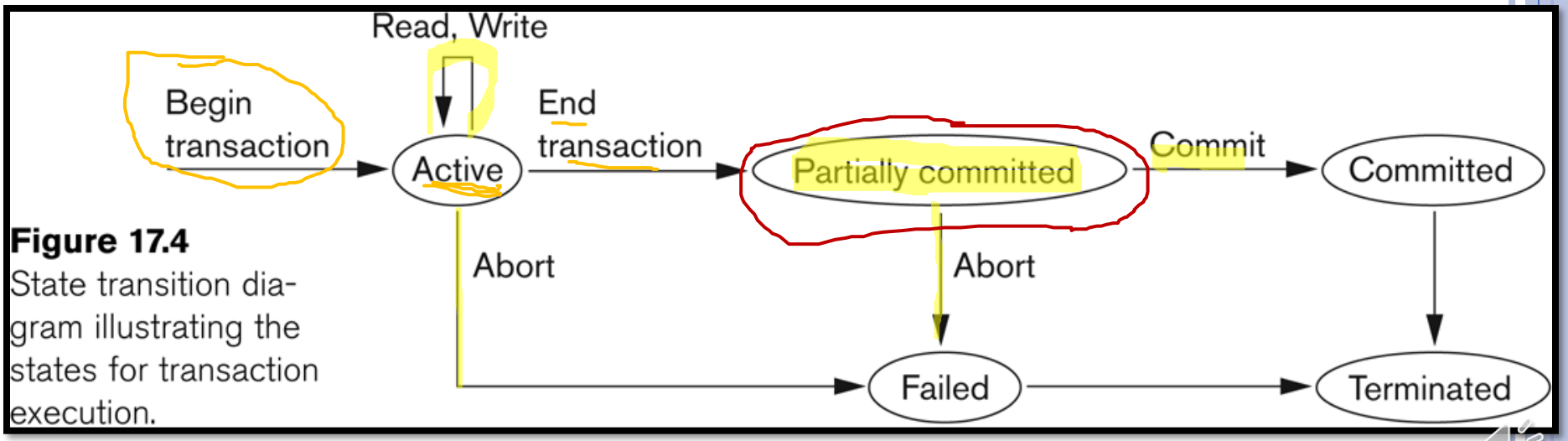


Figure 17.4
State transition diagram illustrating the states for transaction execution.

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

Transaction Processing & Recovery

- Why **recovery** is needed:
  - A computer failure (system crash):
  - A transaction or system error:
    - Integer overflow, division by zero, erroneous parameter values or the user may interrupt the transaction
  - Local errors or exception conditions
    - Data not found or
    - insufficient account balance may cause a fund withdrawal transaction to be canceled.
  - Concurrency control enforcement
    - Transaction violates serializability or several transactions are in a state of deadlock
  - Disk failure
  - Physical problems and catastrophes

Transactions