DBTech VET / DBTechNet
SQL Transactions*

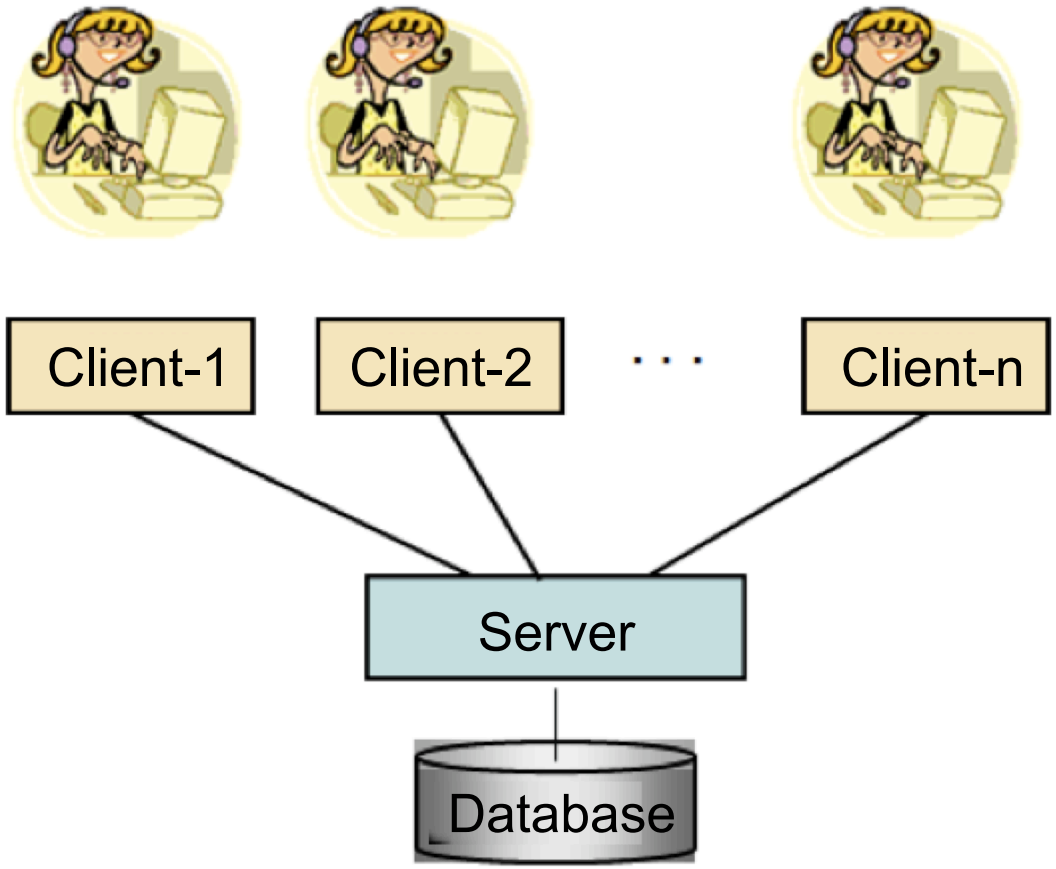
Database Transactions Summit 2013

HAAGA-HELIA University of Applied Sciences,
4 September 2013, Helsinki, Finland.

M. Laiho, D.A. Dervos, K. Silpiö
www.dbtechnet.org

* 

Concurrently executing transactions



IF ... THEN

IF

- The DBMS is lacking the support of basic concurrency control (CC) services, or
- The programmer is lacking the knowledge of how to make proper use of the DBMS supported CC services

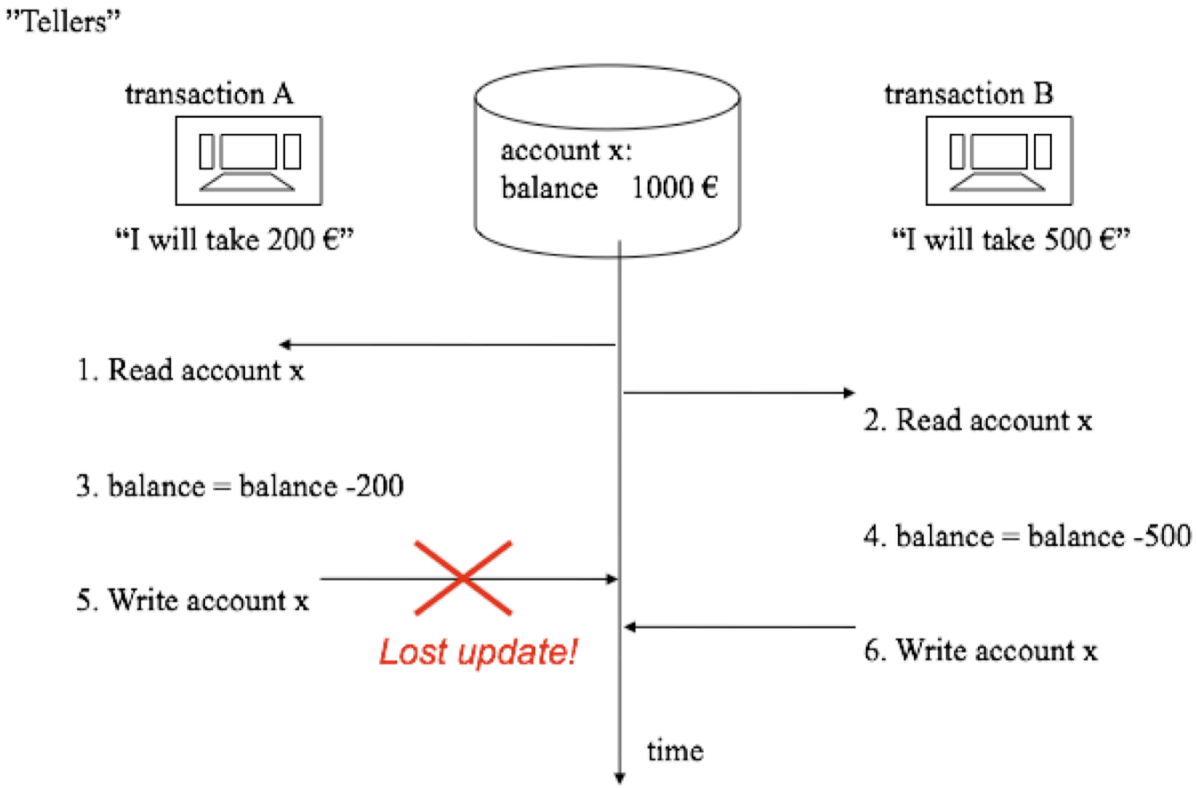
THEN

Data update operations may end up corrupting the DB data content

Concurrency problems (anomalies)

- Lost update
- Dirty read
- Non-repeatable read
- Phantom read

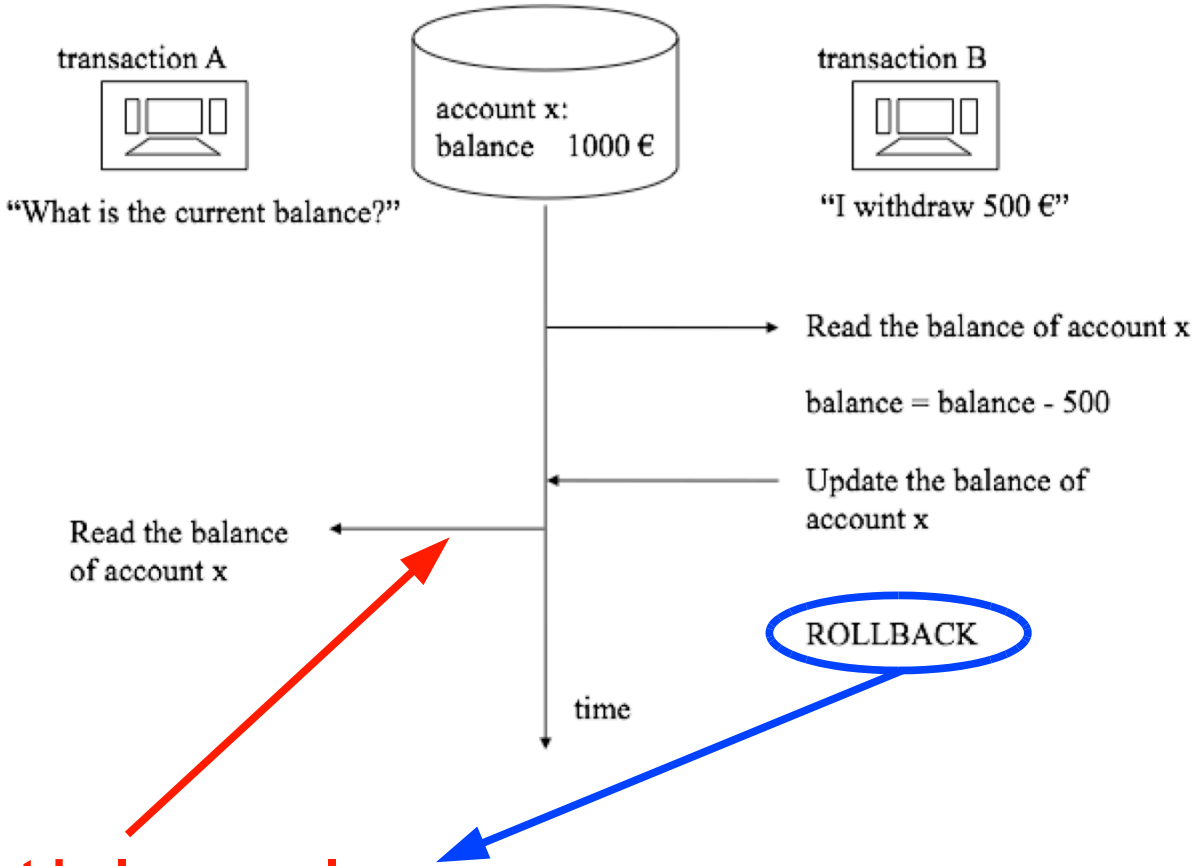
The lost update problem



Concurrency problems (anomalies)

- Lost update
- Dirty read
- Non-repeatable read
- Phantom read

The dirty read problem

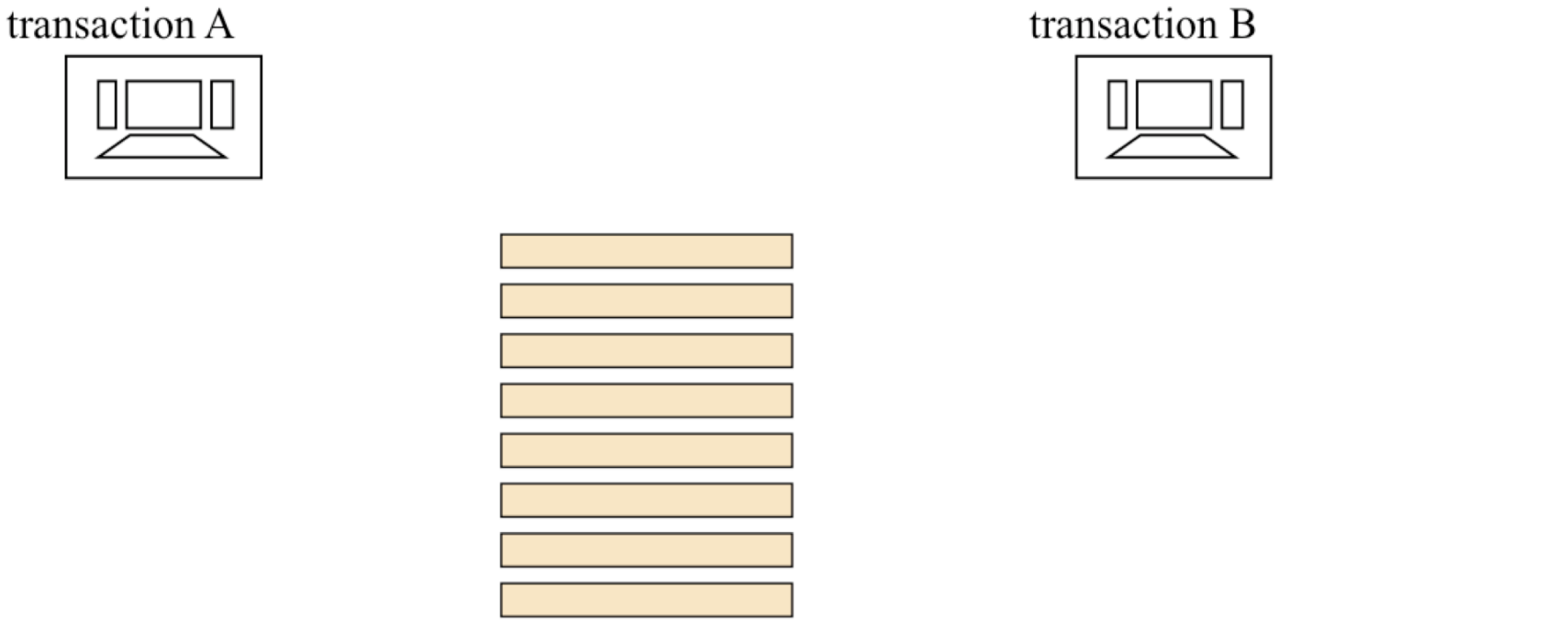


account balance value that never existed!

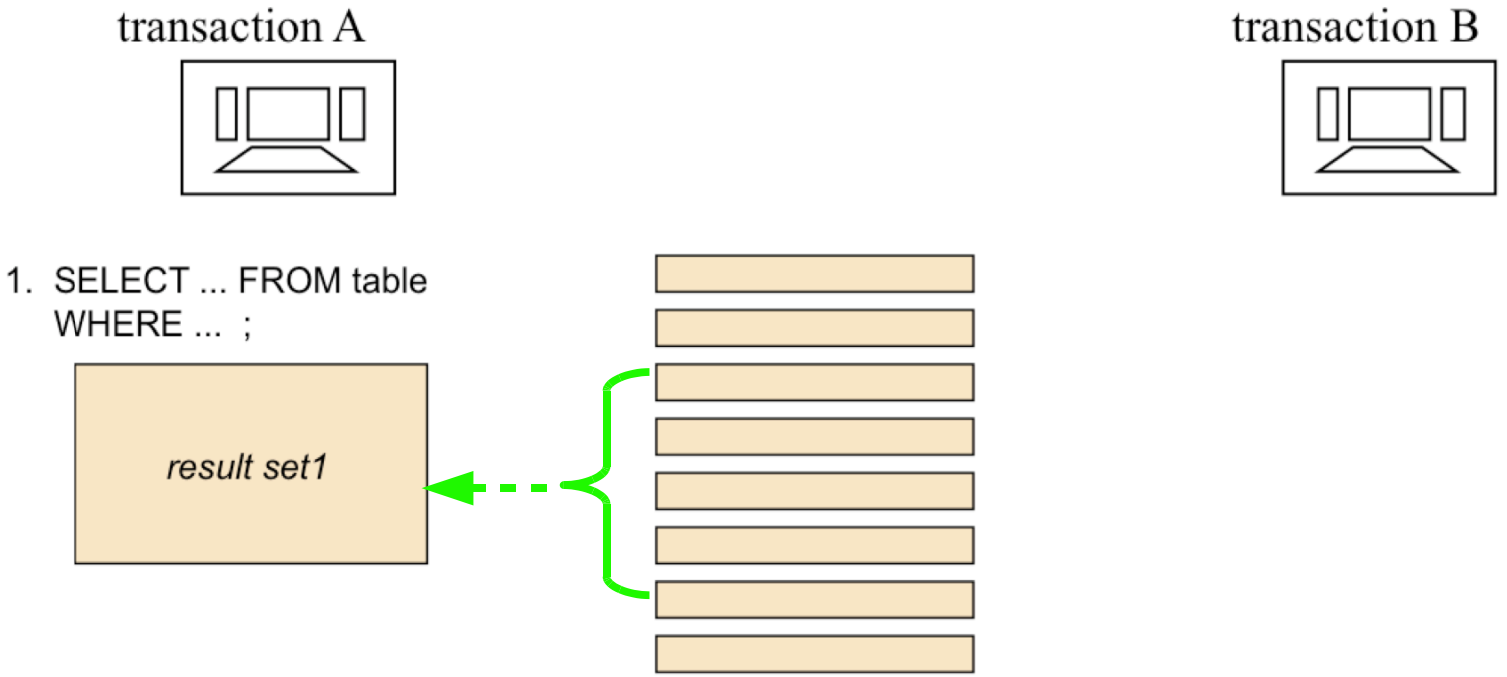
Concurrency problems (anomalies)

- Lost updates
- Dirty reads
- Non-repeatable reads
- Phantom reads

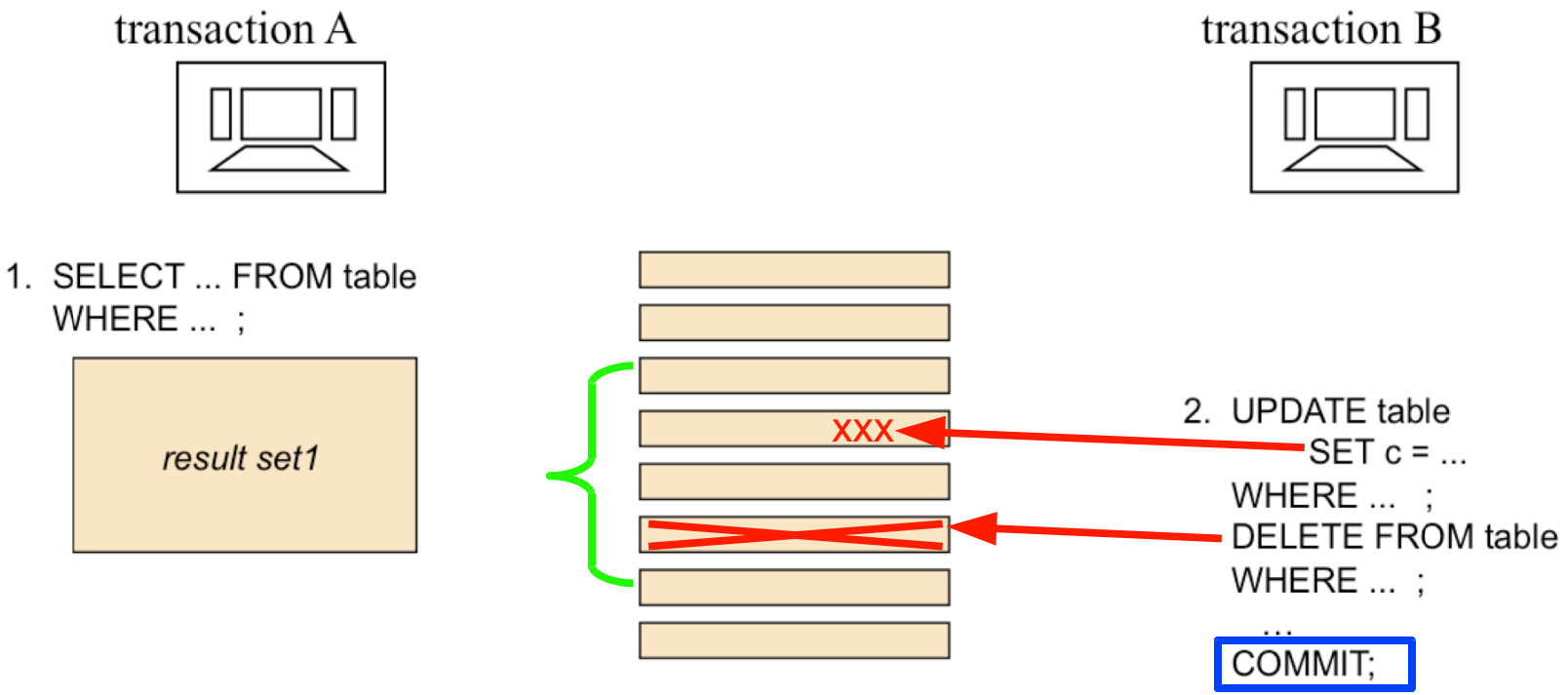
Non-repeatable reads



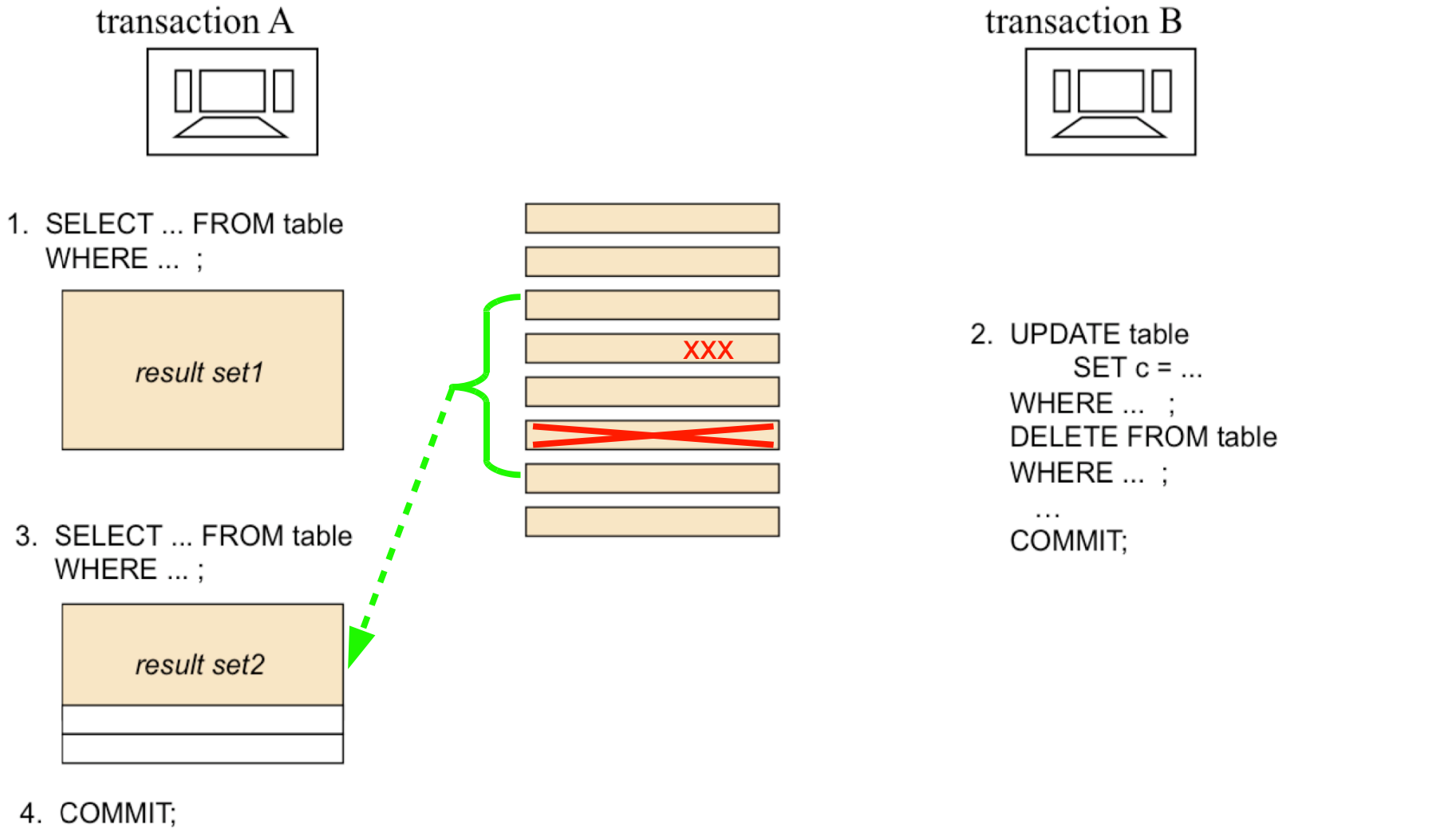
Non-repeatable reads



Non-repeatable reads



Non-repeatable reads



Non-repeatable reads (NRR) vs. dirty reads (DR)

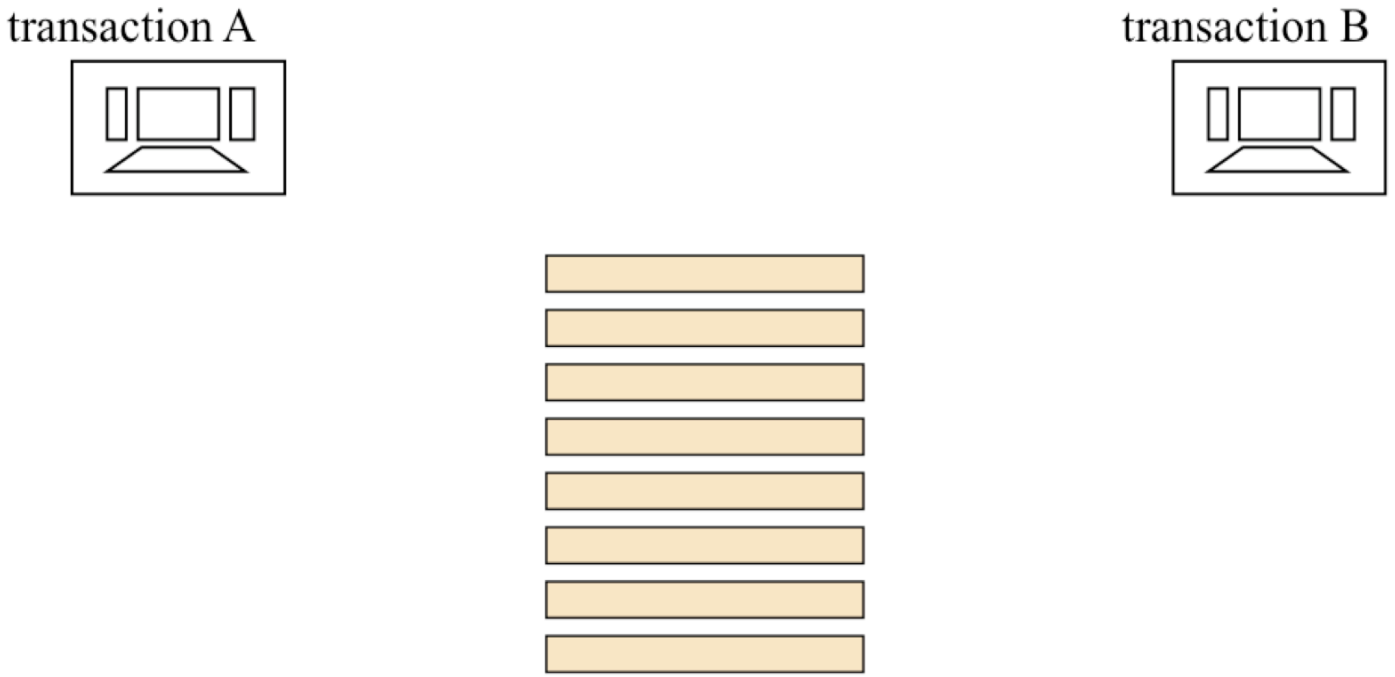
Lifelong Learning Programme

- The transaction “feels” changes made by other transactions (both NRR and DR)
- Repeating the same read operation may yield different results (both NRR and DR)
- Dirty reads (DR): the transaction “feels” changes made by other (concurrently running) transactions while the latter are still active (i.e. it is not yet known whether they will commit or rollback next)
- Non-repeatable reads NRR): the transaction “feels” changes made by other (concurrently running) transactions only after they commit

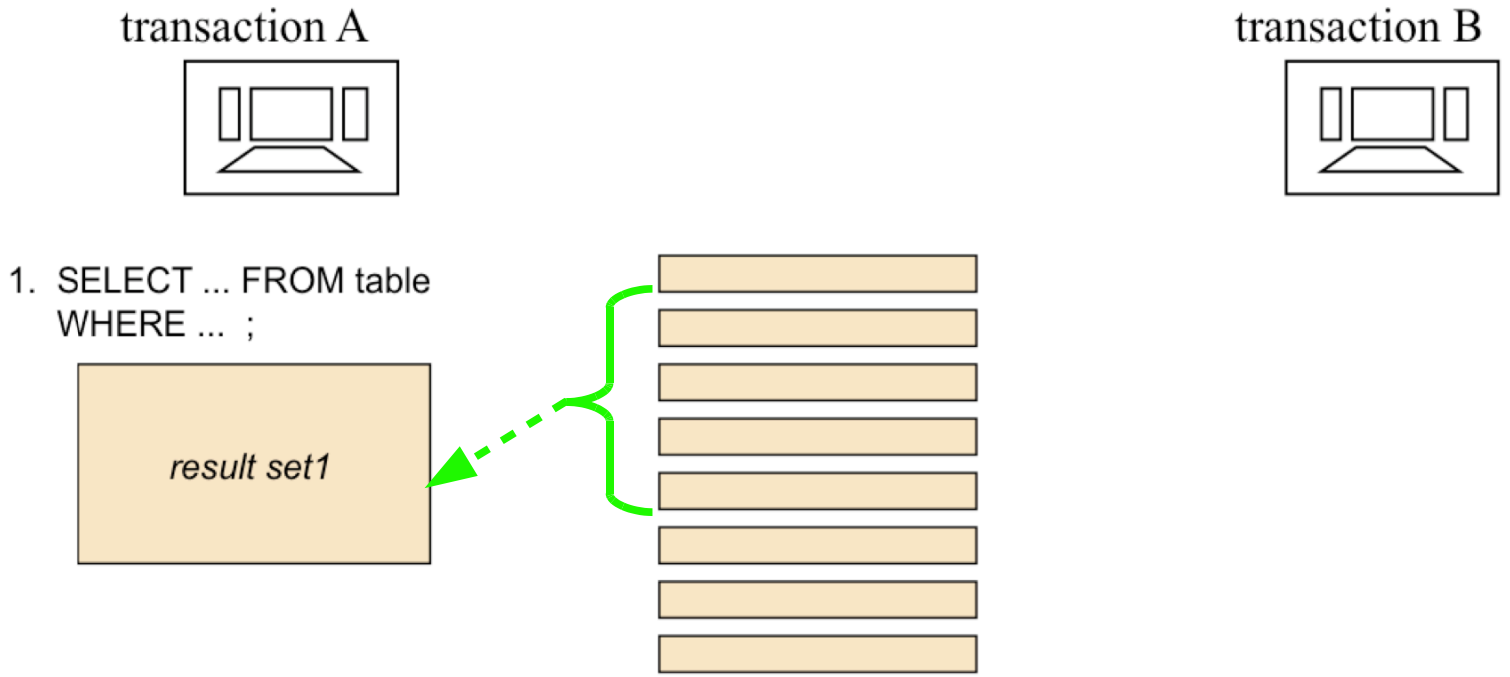
Concurrency problems (anomalies)

- Lost updates
- Dirty reads
- Non-repeatable reads
- Phantom reads

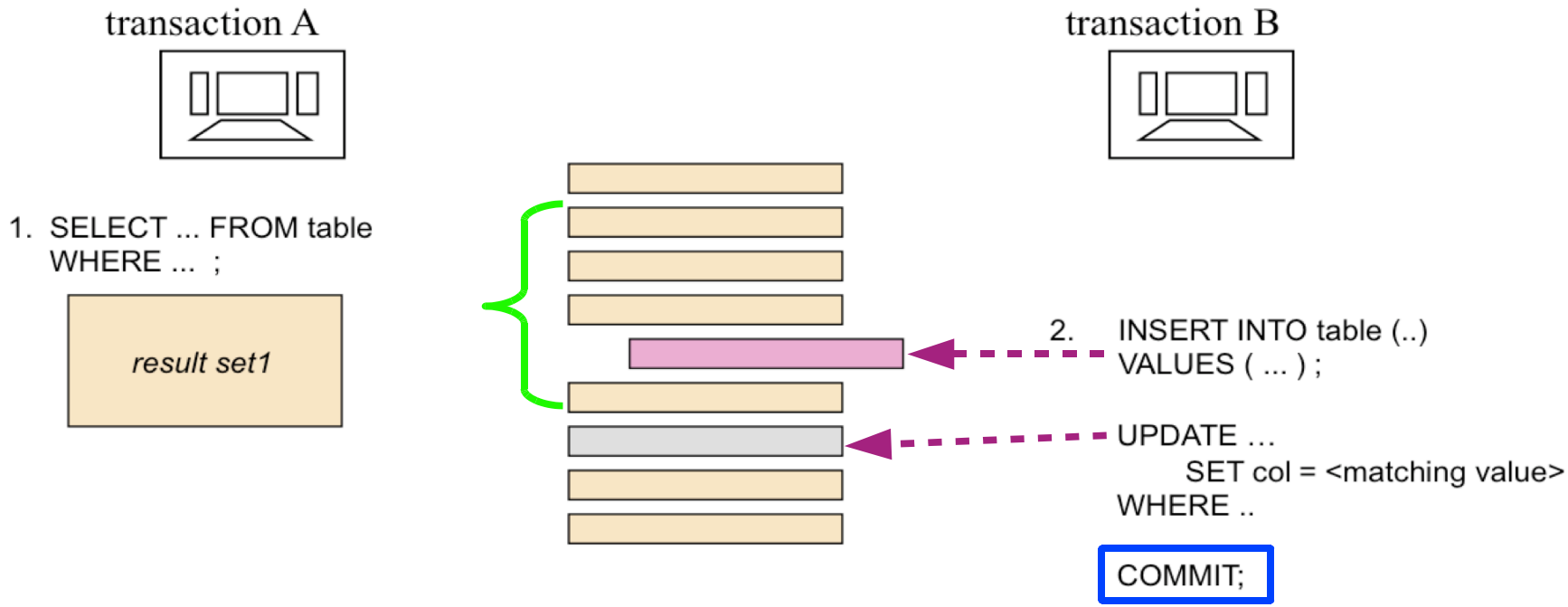
Phantom reads



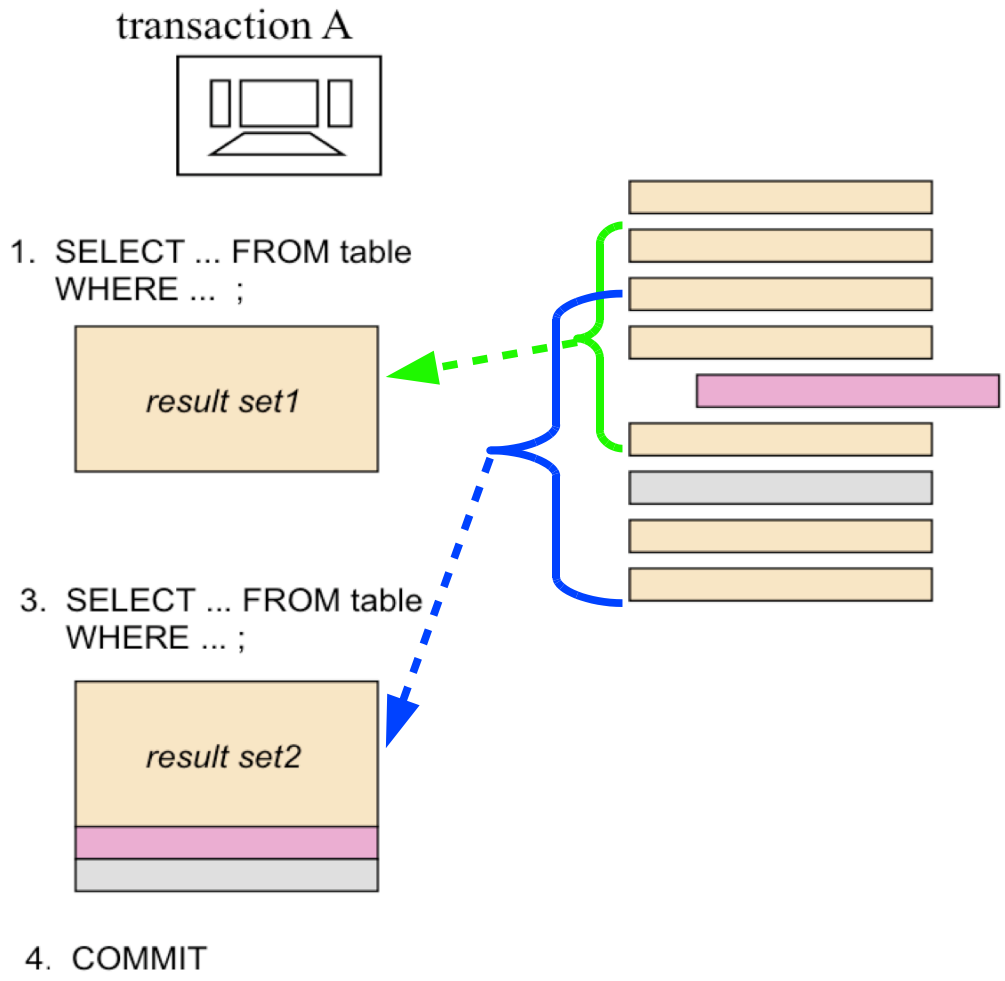
Phantom reads



Phantom reads



Phantom reads



transaction B

2. INSERT INTO table (..)
VALUES (...) ;

UPDATE …
SET col = <matching value>
WHERE ..

COMMIT;

Non-repeatable reads (NR) vs. phantom reads (PR)

Lifelong Learning Programme

- Rows out of nowhere (phantoms) do appear in both NRR and PR resultsets
- In NRR the affected transaction is assumed to be using the same search criterion (WHERE ...), repeatedly
- PR is more general: the affected transaction launches a new search criterion (WHERE ...) each one time.
- Phantom 'reads' because the targeted data/table regions may also involve 'ghost' rows (to be defined next)

A.C.I.D. properties

A transaction should execute in ...

Atomicity

... an ALL or NOTHING fashion

... Consistency

with regard to all the DBMS imposed data integrity rules

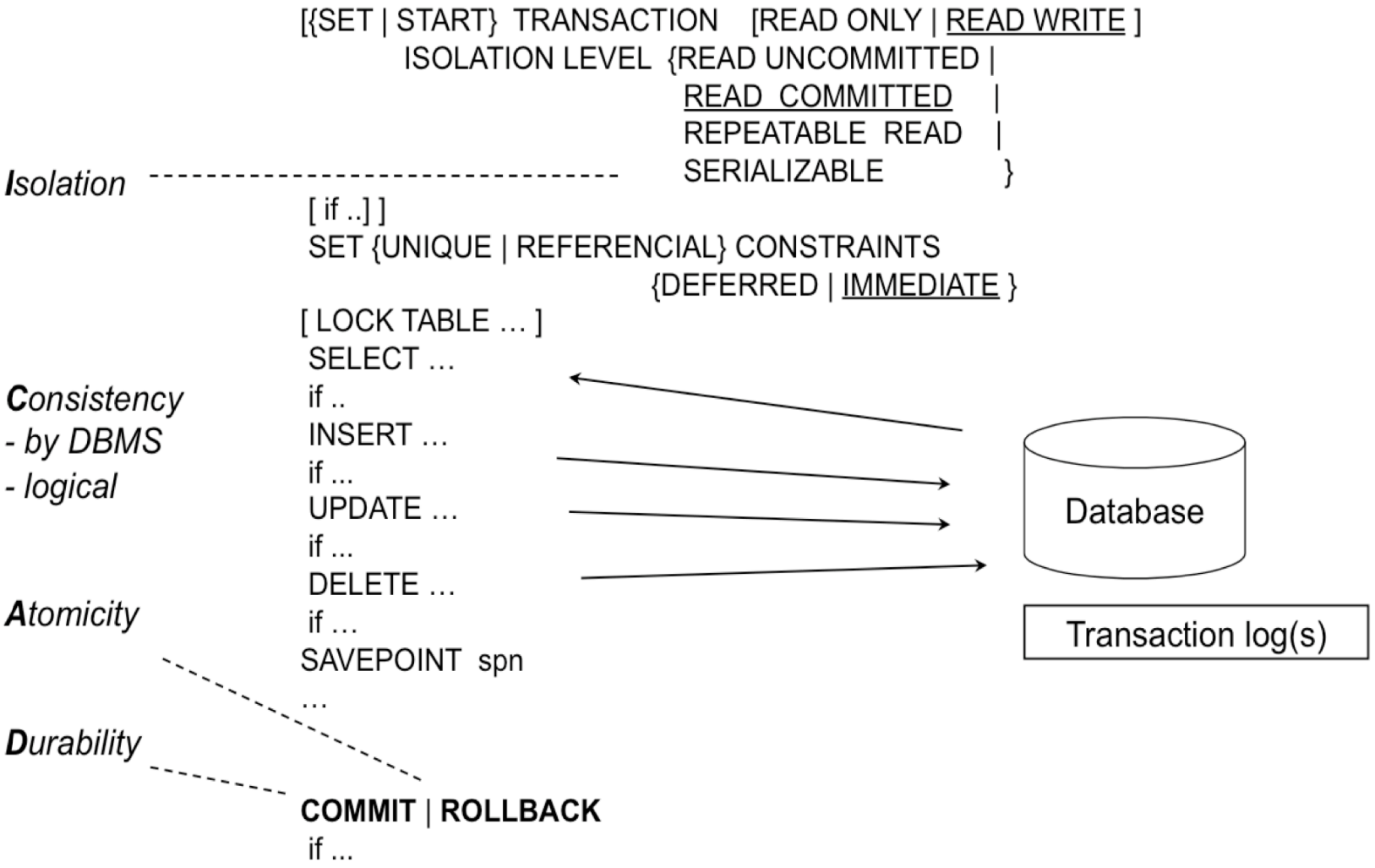
... Isolation

from what other concurrently running transactions do to the database content

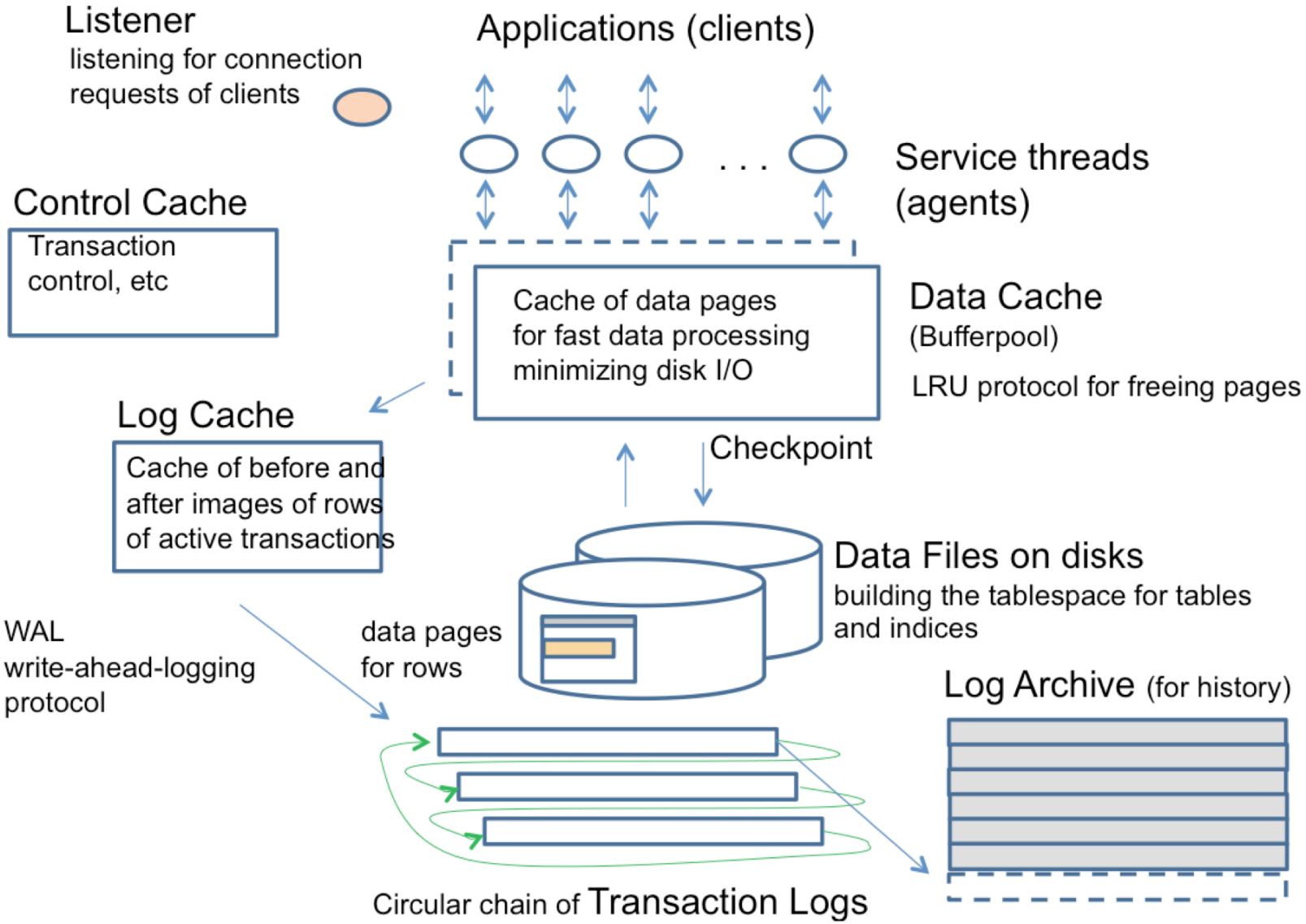
Durability

... a way in which its COMMIT is guaranteed to make persistent all the changes made to the database

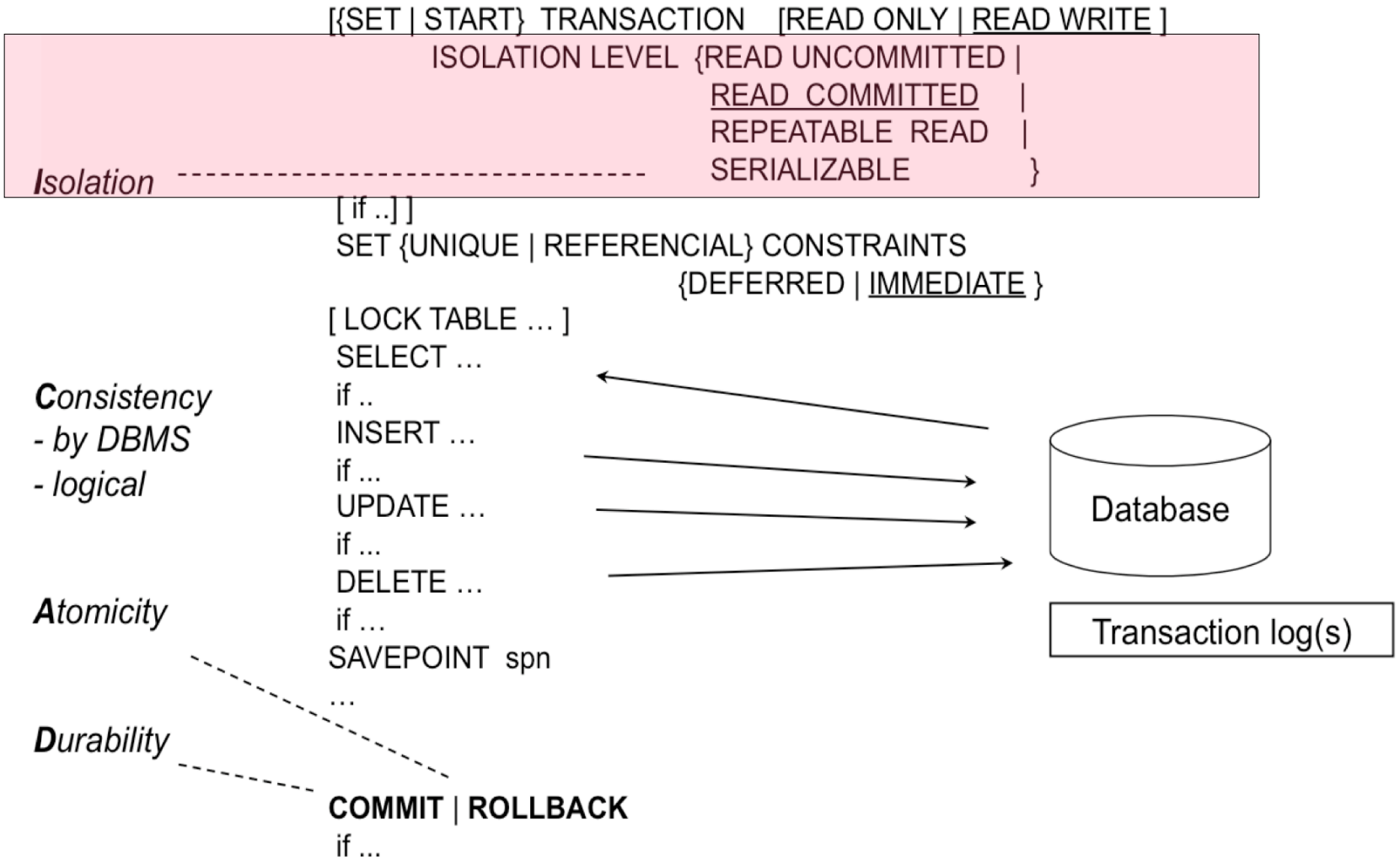
ISO SQL Transaction



A & D: the underlying technology



ISO SQL Transaction



ISO SQL isolation levels defined

Isolation Level: \ Anomalies:	Lost Update	Dirty Read	Nonrepeatable Reads	Phantom Reads
READ UNCOMMITTED	NOT possible	Possible !	Possible !	Possible !
READ COMMITTED	NOT possible	NOT possible	Possible !	Possible !
REPEATABLE READ	NOT possible	NOT possible	NOT possible	Possible !
SERIALIZABLE	NOT possible	NOT possible	NOT possible	NOT possible

Concurrency Control (CC): implementation of

- Multi-Granular Locking (MGL)
- Multi-Versioning (MVCC)
- Optimistic (OCC)

Basic S-(Read) and X-(Write) locking scheme

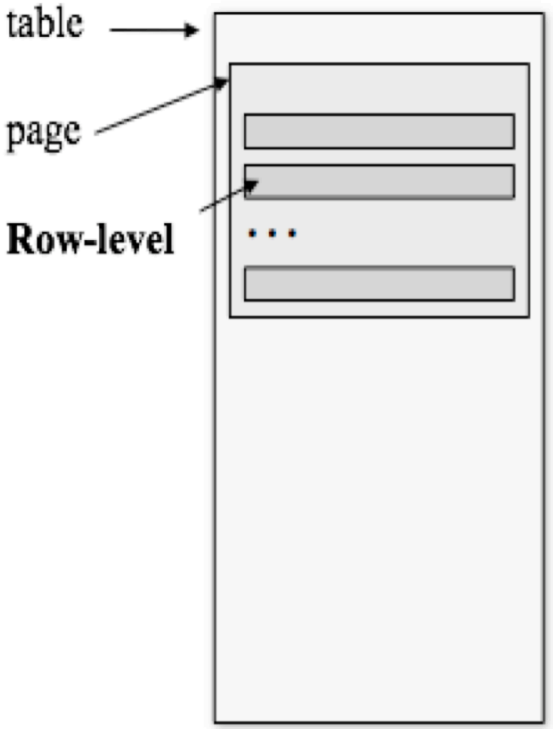
Compatibility of S and X locks:

Lock of transaction A to object o

Lock request of transaction B to object o	Shared	eXclusive
Shared	**Grant**	**Wait !**
eXclusive	**Wait !**	**Wait !**

- S-lock grants read access to object
- X-lock grants write access to object
- X-lock request after getting S-lock is called as lock promotion

Locking granularity:



CC: simplistic approach

Lifelong Learning Programme

READ UNCOMMITTED:

- No S-lock protection for reading, long duration X-locks

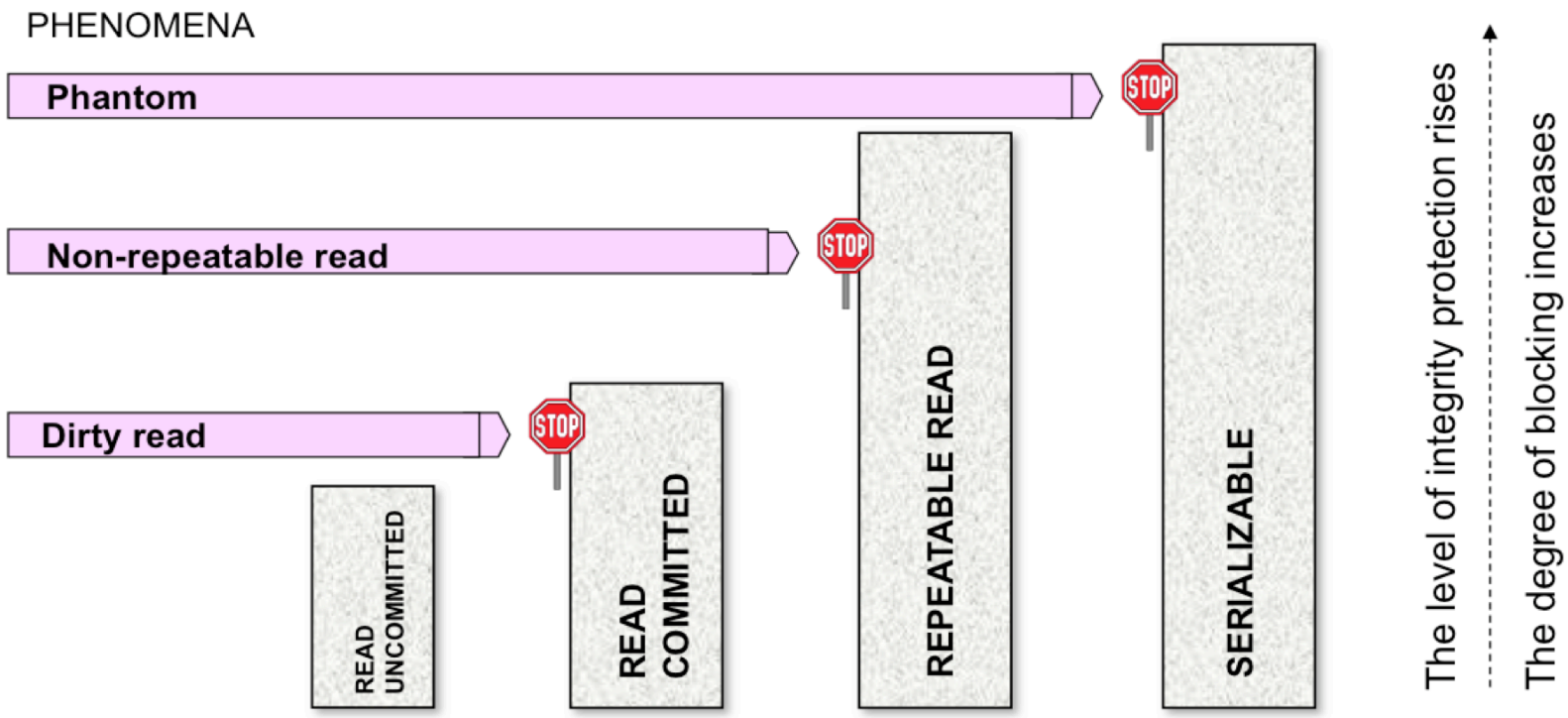
READ COMMITTED:

- Short duration S-locks, long duration X-locks

REPEATABLE READ and SERIALIZABLE:

- Long duration S- and X-locks

A picture's worth a thousand words

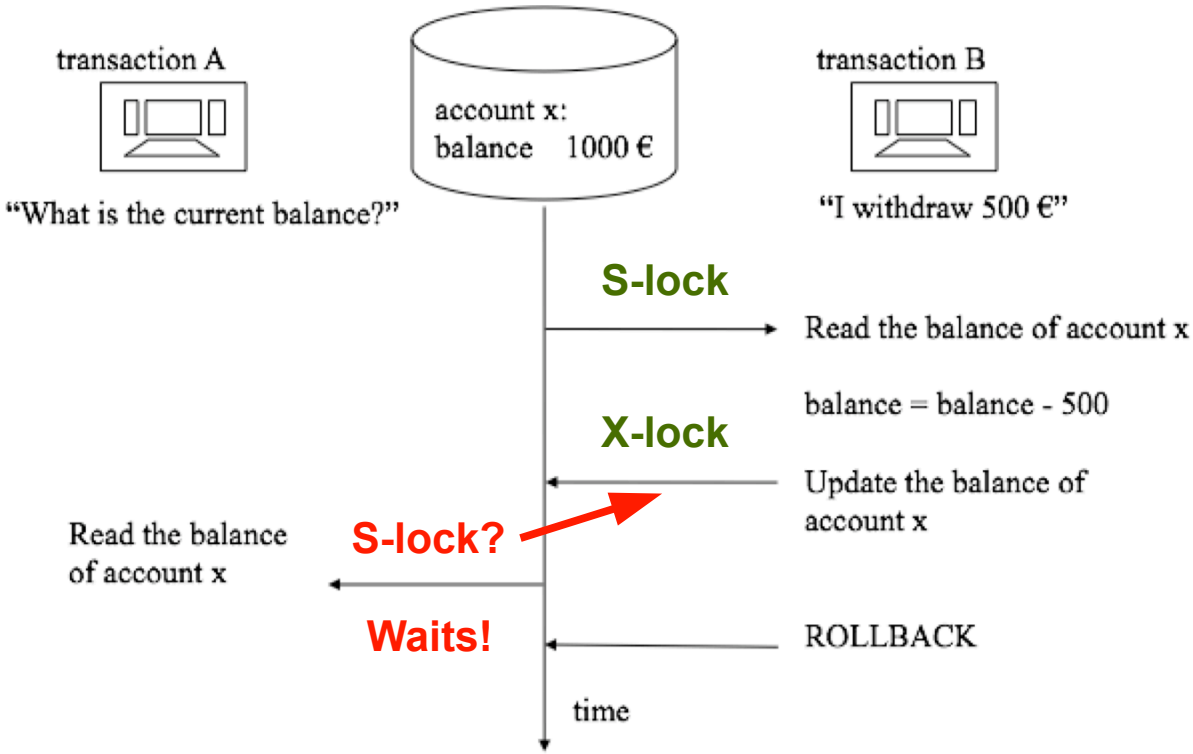


SQL-92 TRANSACTION ISOLATION LEVELS

Lock Durations

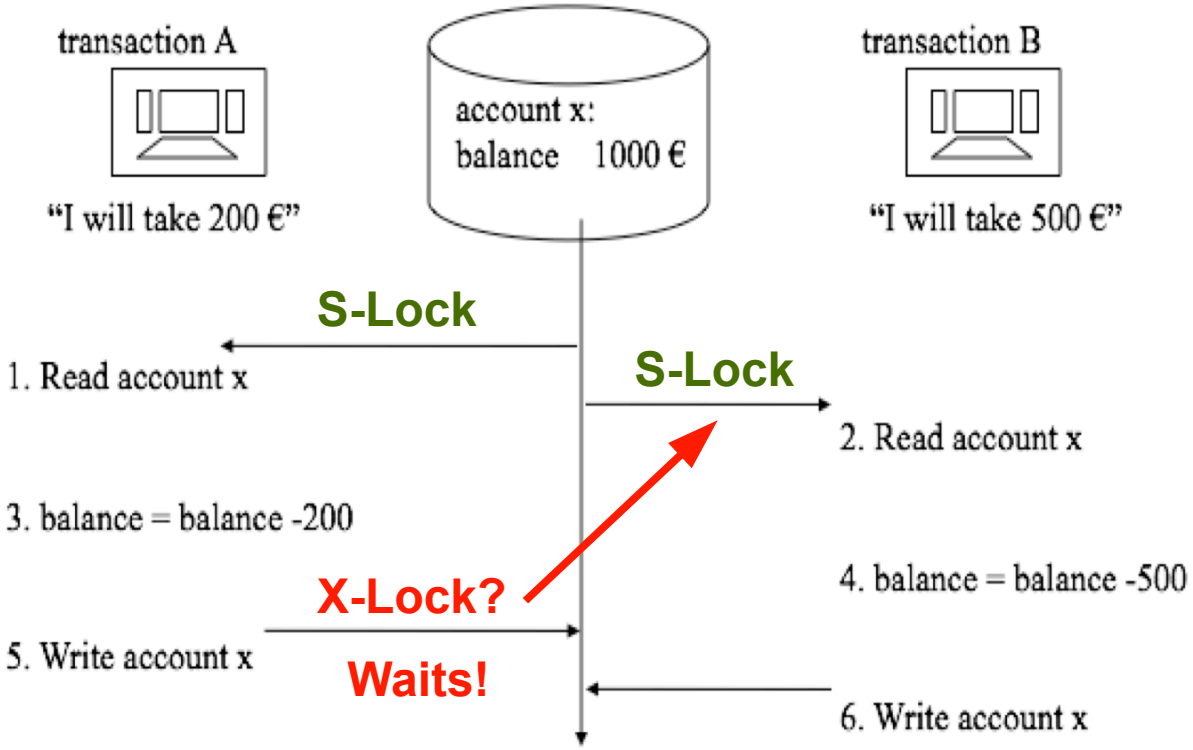
Exclusive Locks	long	long	long	long
Shared Locks (read locks)	not used	short	long	long (based on the predicate)

Dirty read with locks...

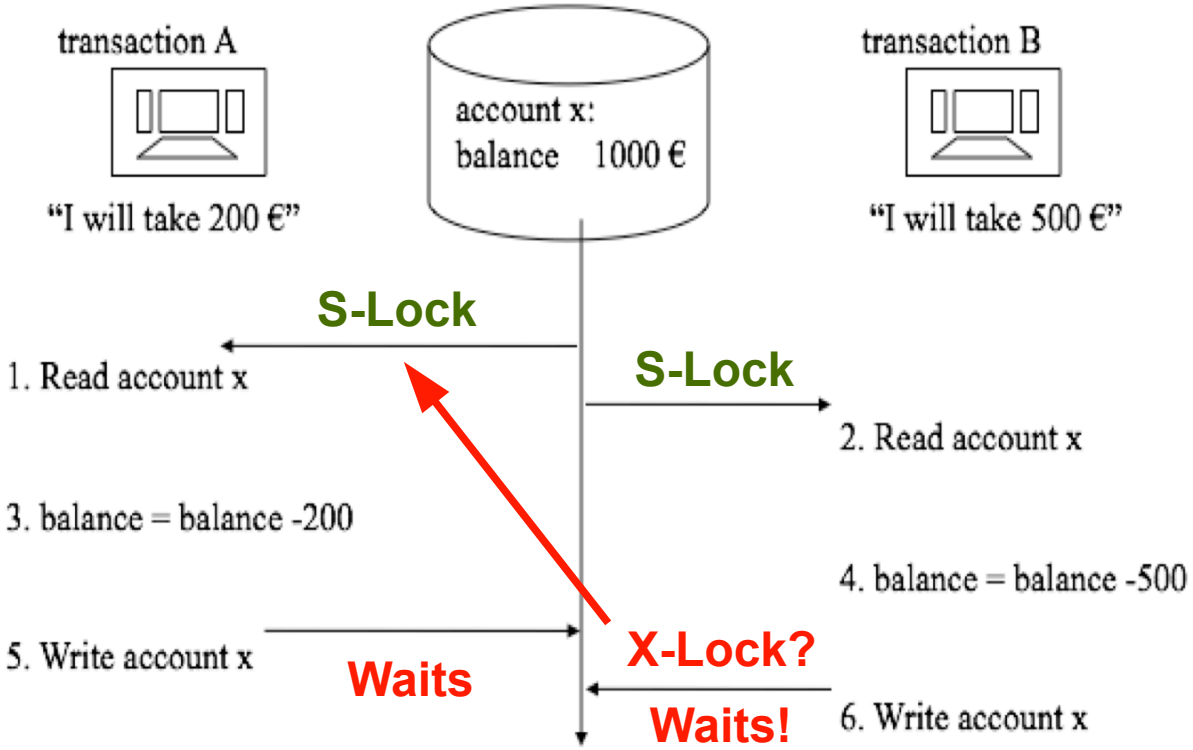


... problem resolved!

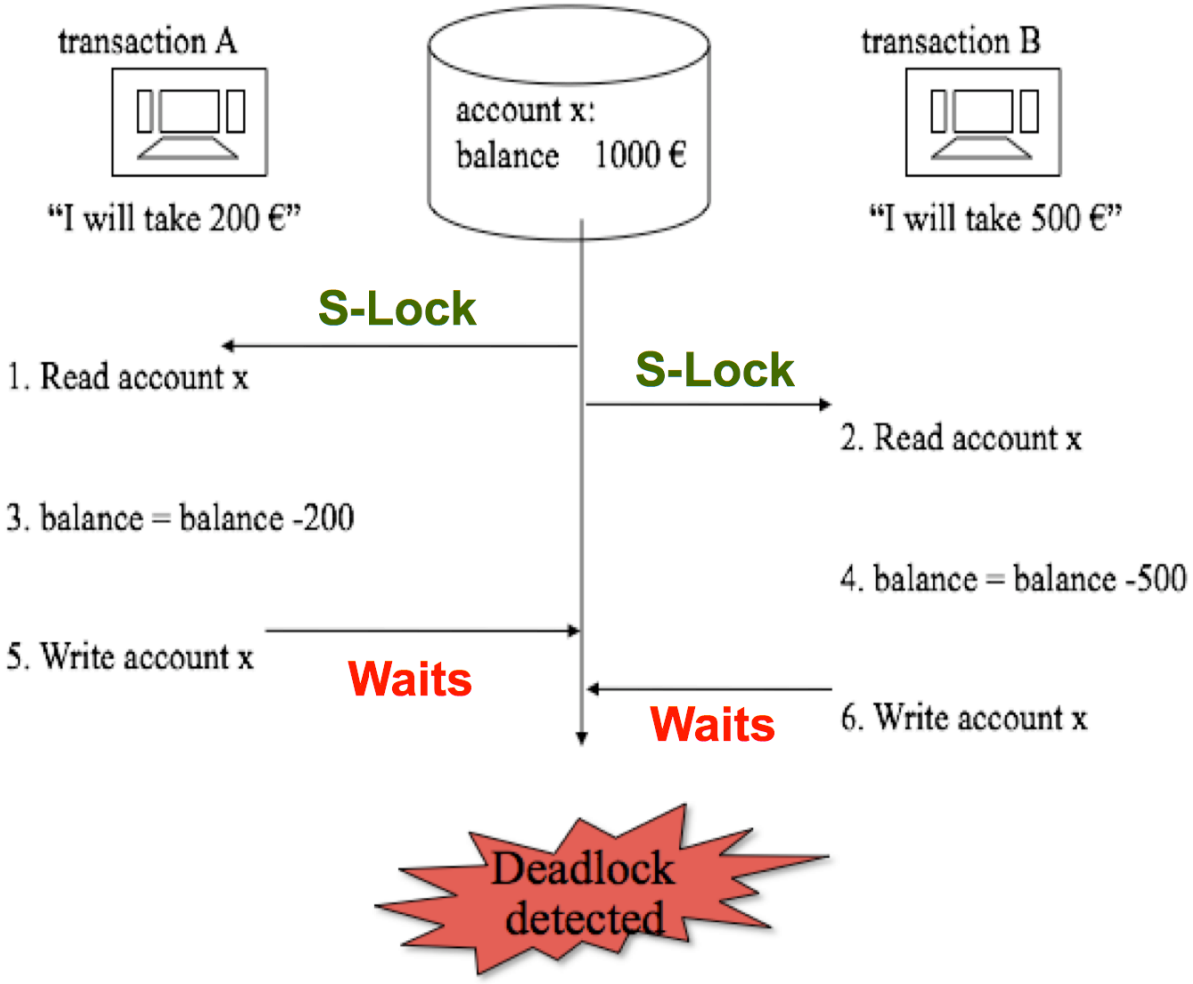
Lost update with locks



Lost update with locks



Lost update with locks



... one of (A,B) need be rolled back

Lost update with locks: sensitive updates

Using ANSI/ISO SQL, the “lost update” scenario is implemented with transactions A and B making use of (single) SQL UPDATE statements, instead of conducting (separate) READ and WRITE operations, e.g.:

```
UPDATE Accounts SET balance = balance – 200
WHERE acctID = 100;
```

In most of today's DBMS's: lock-based CC protection is in effect, and the above gets resolved in a deadlock-free manner...

...resolved?

Lost update with locks: problem resolved?

- Concurrent transactions involving more than one sensitive update SQL statements, plus
- Clumsy programming at the application side...

...may very well lead to having the lost update anomaly appear, even when lock-based concurrency control is implemented at the server side.

The solution to the problem: the system's state need be carefully inspected, in systematic way, following the execution of each one (any) SQL statement: GET DIAGNOSTICS

GET DIAGNOSTICS

Lifelong Learning Programme

- part of the ISO SQL Standard
- implemented in MySQL v5.6, and MariaDB

e.g. in MySQL v5.6 (to be considered during the HoL session):

```
GET DIAGNOSTICS @rowcount = ROW_COUNT;

GET DIAGNOSTICS CONDITION 1 @sqlstate = RETURNED_SQLSTATE,
@sqlcode = MYSQL_ERRNO ;

SELECT @sqlstate, @sqlcode, @rowcount;
```

Multi-granular locking scheme (MGL)

- *Sample variants of lock compatibility matrices*

Lock granules:

database

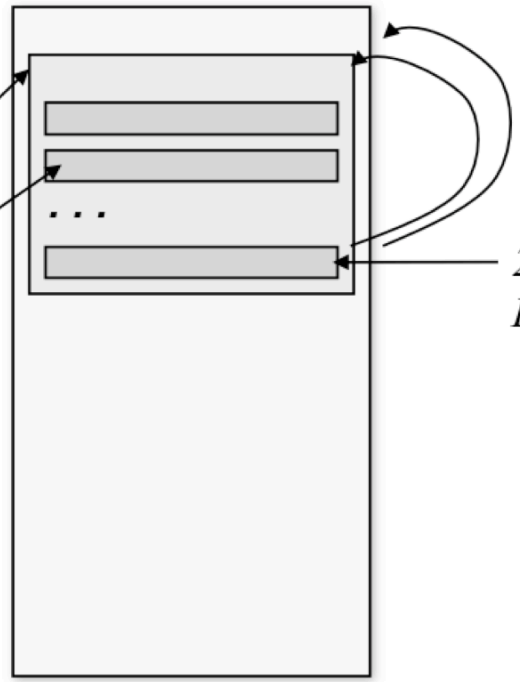
(tablespace)

table

(extent)

page

row



Other locks on index ranges, schemas

Lock requested:	Lock already granted to some other process				
	IS	IX	S	SIX	X
IS	grant	grant	grant	grant	wait
IX	grant	grant	wait	wait	wait
S	grant	wait	grant	wait	wait
SIX	grant	wait	wait	wait	wait
X	wait	wait	wait	wait	wait

SIX = S + IX

1. Intent locks
IS for S on row
IX for X on row

2.
Lock on row

Lock requested:	Lock already granted to some other process			
	none	S	U	X
S	grant	grant	grant[3]	wait
U	grant	grant	wait	wait
X	grant	wait	wait	wait

***S**hared locks (S) allow reading.*
*e**X**clusive locks (X) allow writing and are kept up to end of transaction eliminating lost updates.*

Concurrency control: implementation of

- Multi-Granular Locking (MGL)
- Multi-Versioning (MVCC)
- Optimistic (OCC)

MVCC: motivation

Lifelong Learning Programme

A database where several tables updated frequently:

- Want to read uncommitted, but dot not wish to compromise on data consistency
- Equivalently: wish to conduct data reading on a (logical) snapshot of the DB content, taken at begin time of the read-only transaction

MVCC: in simple terms

Lifelong Learning Programme

- The DB server makes use of timestamps to maintain history chains (versions), one for each row that is being updated
- Considering the above, any one transaction may either read the latest committed version of each row **at read time** (READ COMMITTED), or the latest committed version of each one row **at its (transaction) begin time** (SNAPSHOT)

Consequently, readers and writers do not block each other: improved performance

** extra overhead, or...*

Snapshot isolation (SI)

- Improved performance (readers and writers do not block each other, fewer deadlocks)

- Write-write conflicts are handled by policies that depend on the DBMS used:

  (a) ORACLE uses some type of locking; the second writer waits,
  (b) under SolidDB, the first writer is the only one who is allowed to commit

SI: ghosts and phantoms

- A transaction never accesses phantom rows, i.e. rows that have been inserted by other transactions after its begin time(stamp): compare/contrast with MGL
- A transaction may access ghost rows, i.e. rows that have been deleted or updated by other transactions after its begin time(stamp): compare/contrast with MGL

SI and serializability

Lifelong Learning Programme

- SI is not appropriate for implementing the ISO SERIALIZABLE isolation level
- MySQL uses it only for implementing the REPEATABLE READ isolation level

Example*

T1: copies x into y

T2: copies y into x

Initially: (x,y) = (8,10)

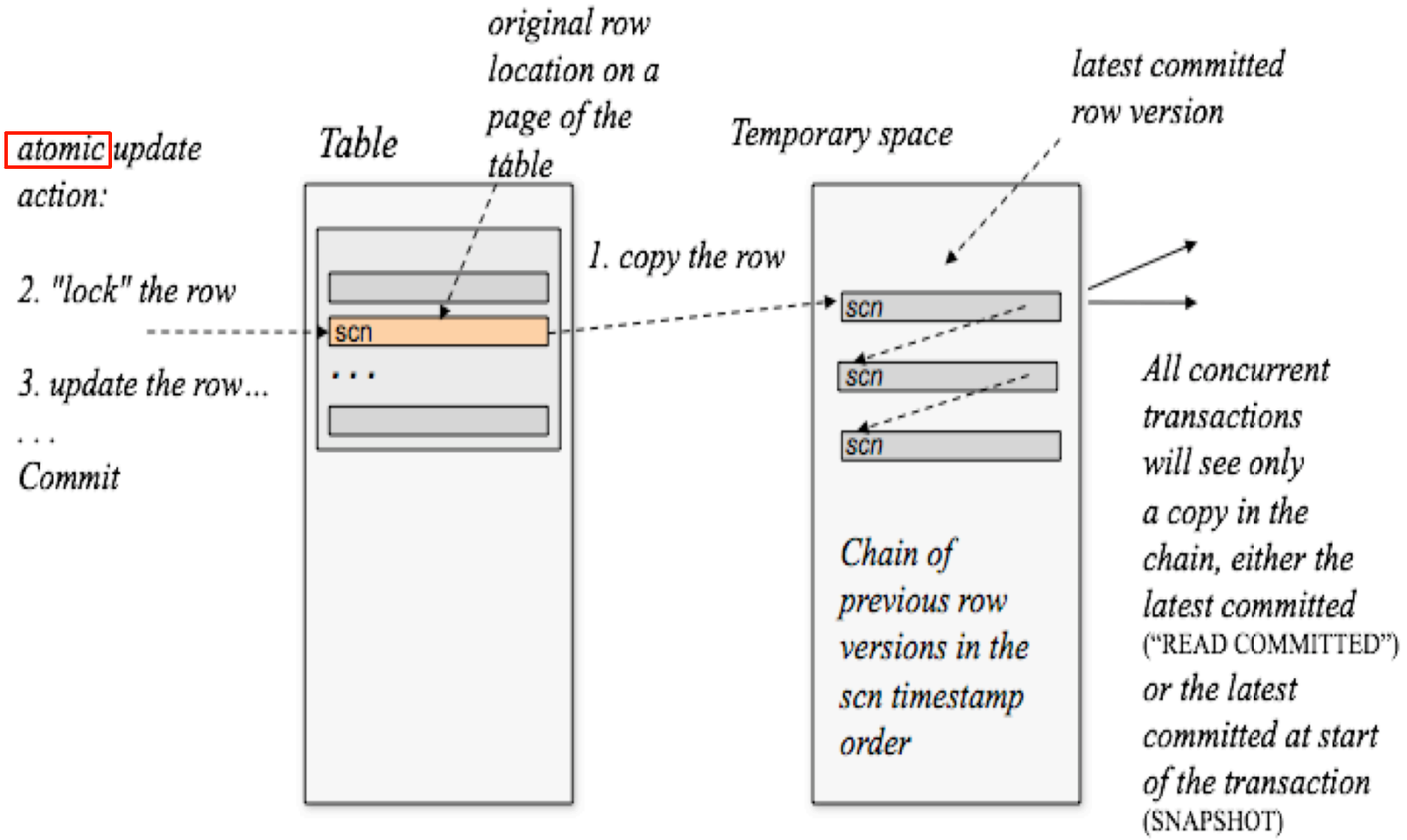
Possible outcomes

Under ISO SERIALIZABLE: (8,8), (10,10)

With SI: (8,8), (10,10), (10,8)

**Philip A. Bernstein and Eric Newcomer, Principles of Transaction Processing, 2nd Edition, Morgan Kaufmann, 2009*

MVCC implementation: Oracle



***scn**: system change number*

MySQL/InnoDB CC implementation

- **READ UNCOMMITTED**: MVCC (*read latest written*)
- **READ COMMITTED**: MVCC (*read latest committed*)
- **REPEATABLE READ**: MVCC (*snapshot isolation*)
- **SERIALIZABLE**: MGL (*MV with long locks on the latest version of the rows read/written*)

Optimistic Concurrency Control (OCC)

- Appropriate for situations where DB data are cached outside the DB server's cache memory
- The application reads data from the (remote) cache memory in an optimistic way (i.e. hoping that the corresponding DB server residing data have not been altered/updated in the meantime)
- Variations exist (e.g. MS-SQL Server's *optimistic with values*, and *optimistic with versions*)
- Original OCC (Phyrrho DBMS): all changes made to data by a transaction are kept separate from the database and they get registered to/synchronized with the latter at commit time

Questions

Lifelong Learning Programme

?

Getting ready for the HoL session

Lifelong Learning Programme

The DBTechNET DebianDB VM