DBTech VET

www.DBTechNet.org

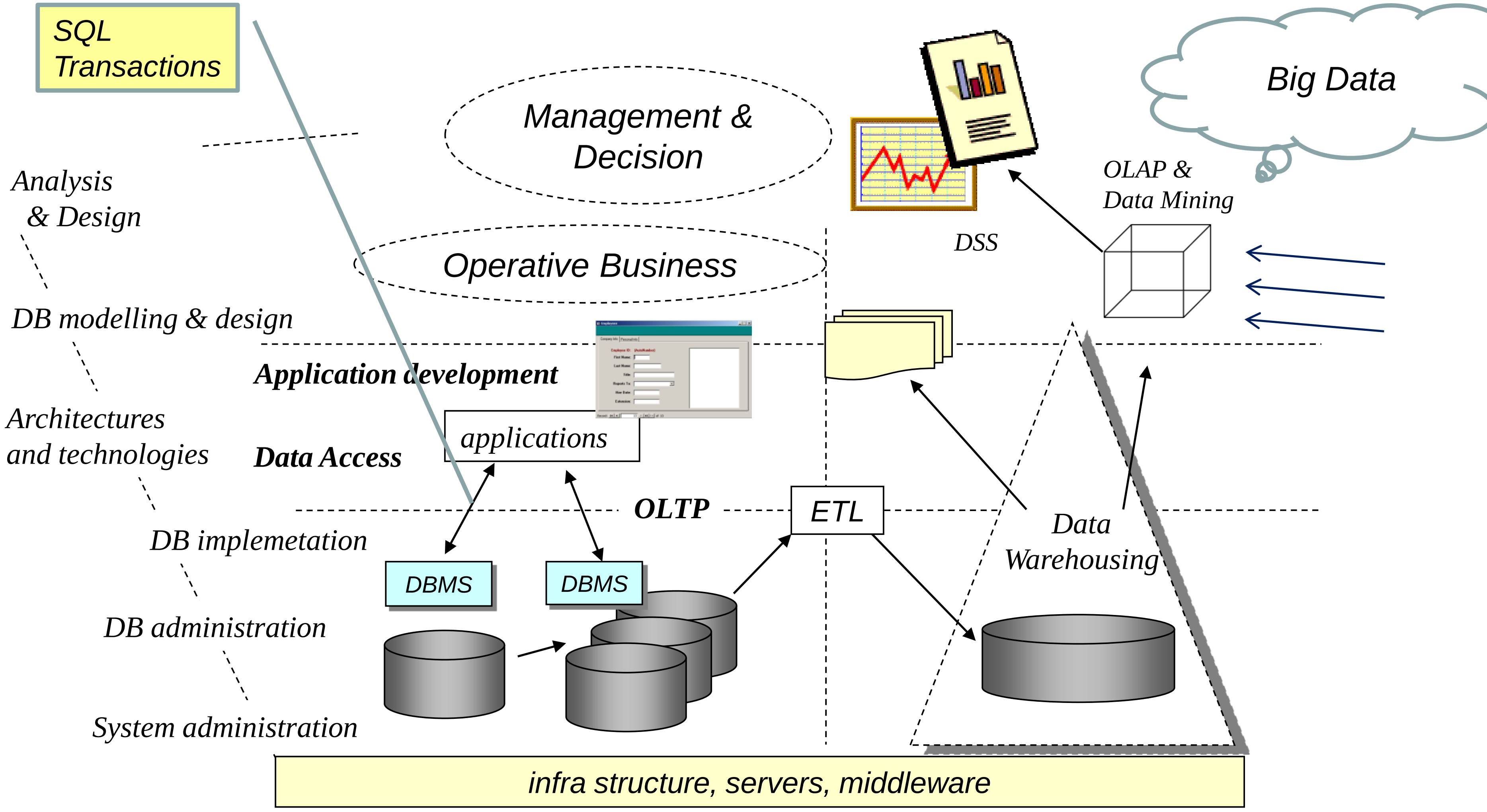
Introduction to SQL Transactions

for teachers, trainers and application developers

Martti Laiho
martti.laiho@haaga-helia.fi

Disclaimer
This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Areas in Database Technologies



SQL
Transactions

Analysis
& Design

DB modelling & design

Architectures
and technologies

DB implemetation

DB administration

System administration

Management &
Decision

Operative Business

Application development

Data Access

applications

OLTP

ETL

DBMS

DBMS

DSS

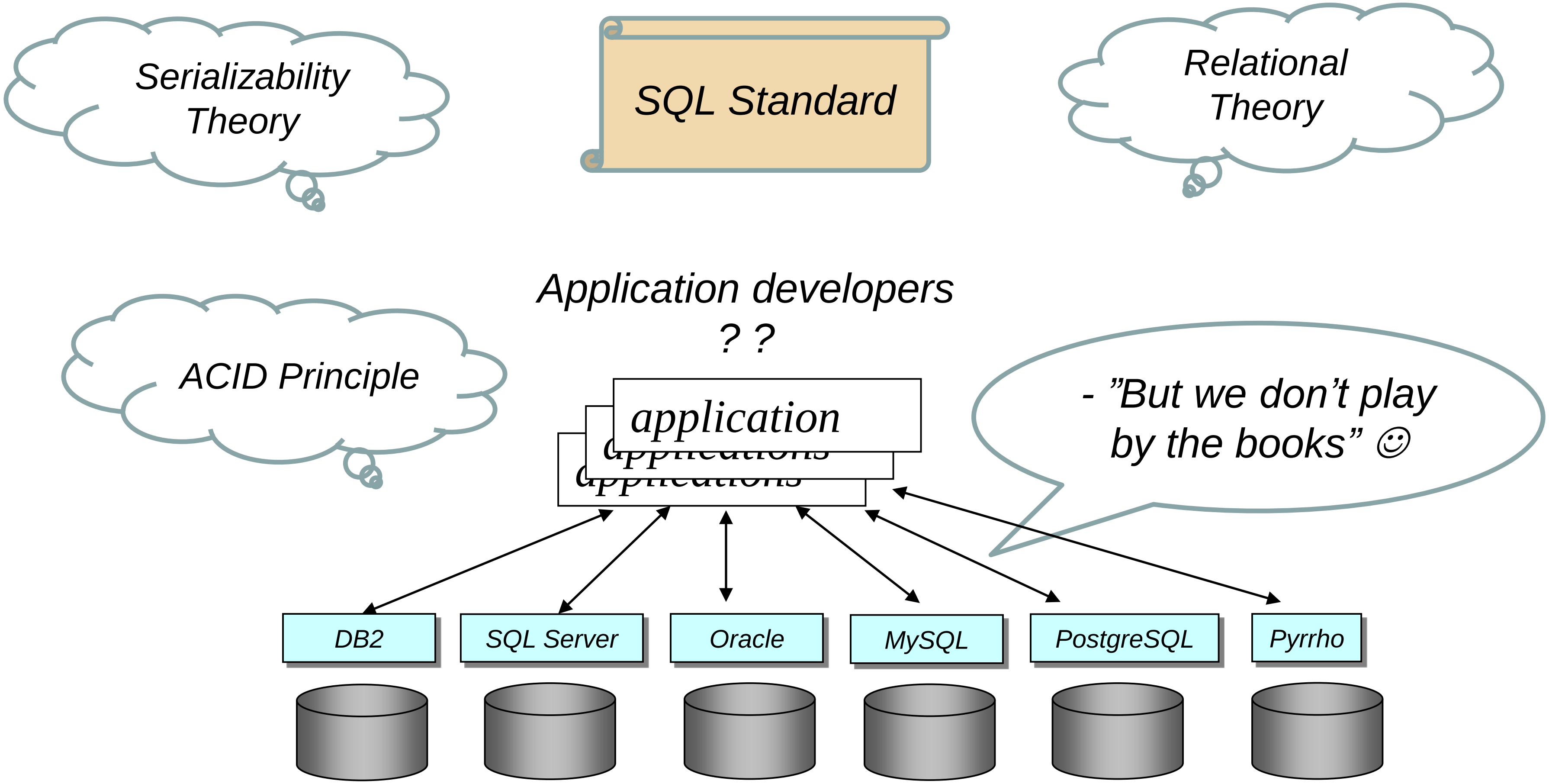
OLAP &
Data Mining

Big Data

Data
Warehousing

infra structure, servers, middleware

OLTP - Theories and Practice?



Serializability Theory

SQL Standard

Relational Theory

Application developers

? ?

ACID Principle

application

applications

applications

- ”But we don’t play by the books” ☺

DB2

SQL Server

Oracle

MySQL

PostgreSQL

Pyrrho

Contents

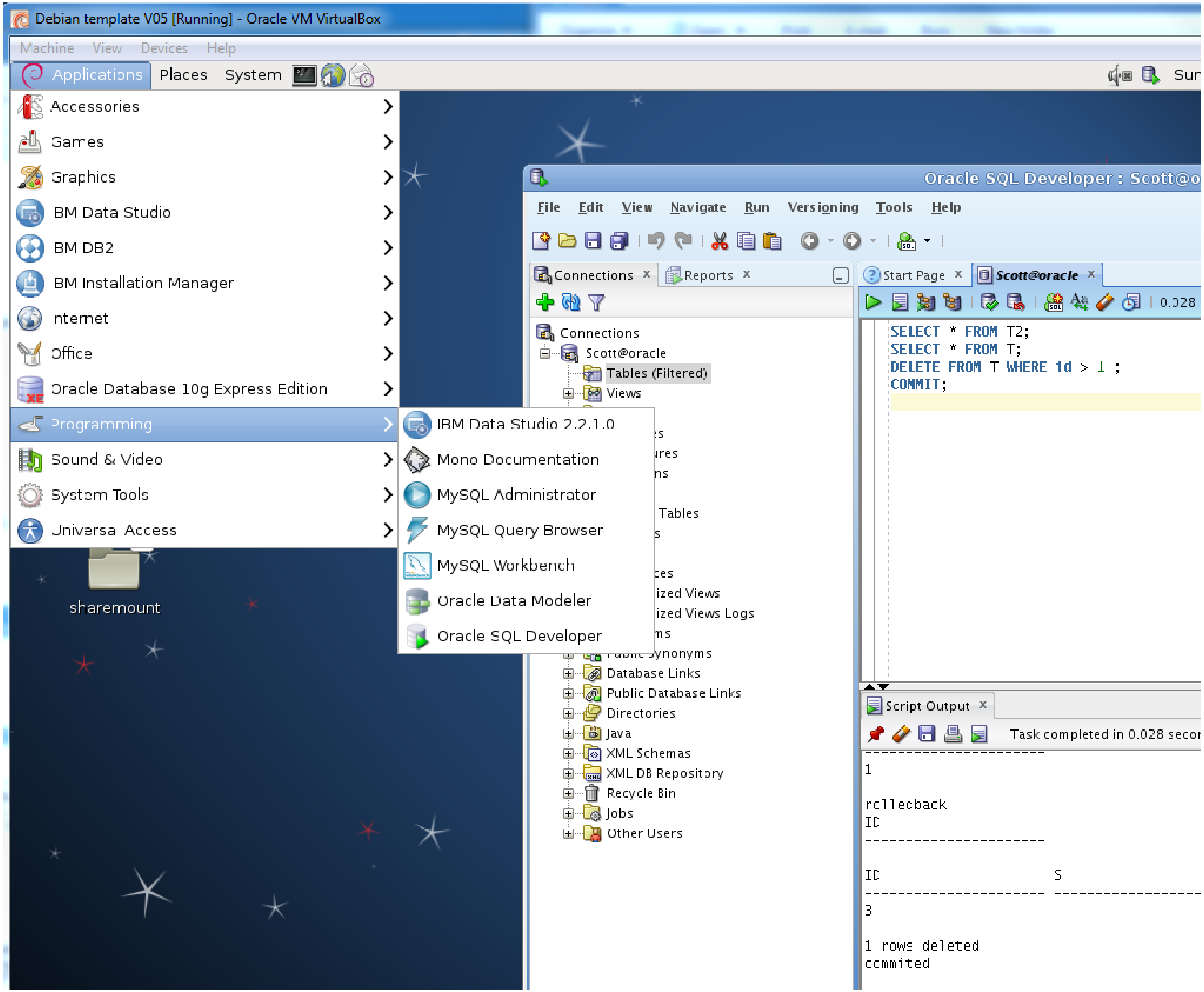
Lesson 1

- Database laboratory: DB2, Oracle, MySQL/InnoDB, PostgreSQL,...
- Concepts:
  - SQL-server, SQL-client, SQL-session
  - Client/Server dialogue: request, result, diagnostics
- SQL transaction
  - Autocommit mode, Implicit/explicit start of transaction
  - Commit: new consistent state, durability
  - Rollback: atomicity, transaction recovery
  - Consistency: constraints, diagnostics, exception handling
  - Diagnostics: SQLcode, SQLSTATE, ..
- Single-user experiments

Lesson 2

- Concurrency: anomalias
- ACID principle: isolation?
- Isolation levels
- Concurrency Control Mechanisms: MGL, MVCC
- Multi-user experiments
- Some ”Best Practices”

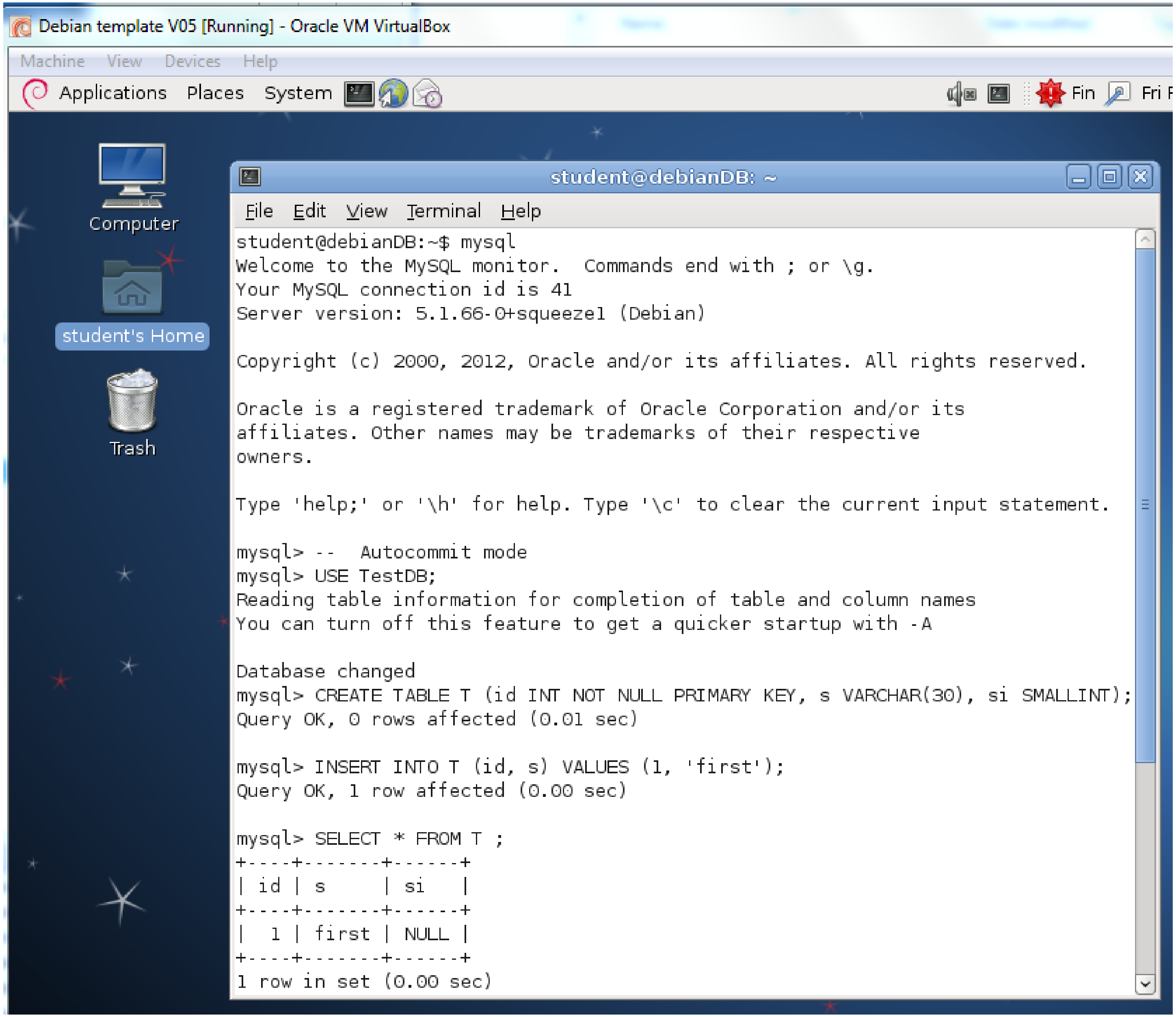
VirtualBox DebianDB



```
SELECT * FROM T2;
SELECT * FROM T;
DELETE FROM T WHERE id > 1 ;
COMMIT;
```

```
---------------------
1

rolledback
ID
---------------------

ID                    S
--------------------- ------------------
3

1 rows deleted
commited
```

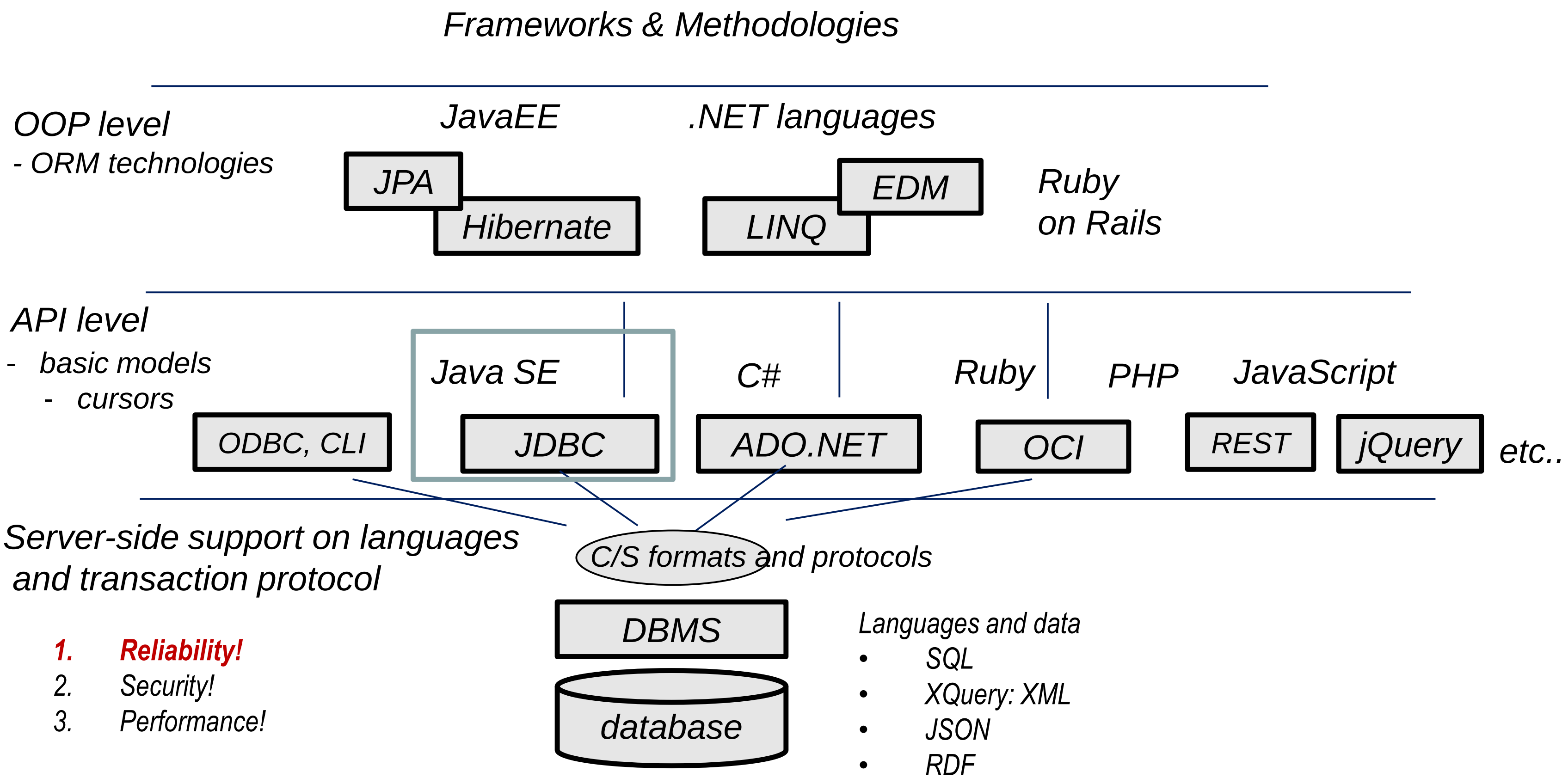
A sample MySQL test



Debian template V05 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Applications Places System
Fin
Fri
Computer
student's Home
Trash
student@debianDB: ~
File Edit View Terminal Help

```
student@debianDB:~$ mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 41
Server version: 5.1.66-0+squeeze1 (Debian)

Copyright (c) 2000, 2012, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> --  Autocommit mode
mysql> USE TestDB;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> CREATE TABLE T (id INT NOT NULL PRIMARY KEY, s VARCHAR(30), si SMALLINT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO T (id, s) VALUES (1, 'first');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM T ;
+----+-------+------+
| id | s     | si   |
+----+-------+------+
|  1 | first | NULL |
+----+-------+------+
1 row in set (0.00 sec)
```

A Map on Data Access Technologies



Frameworks & Methodologies

OOP level
- ORM technologies

JavaEE

JPA

Hibernate

.NET languages

LINQ

EDM

Ruby
on Rails

API level
- basic models
  - cursors

ODBC, CLI

Java SE

JDBC

C#

ADO.NET

Ruby

OCI

PHP

JavaScript

REST

jQuery

etc..

Server-side support on languages
and transaction protocol

C/S formats and protocols

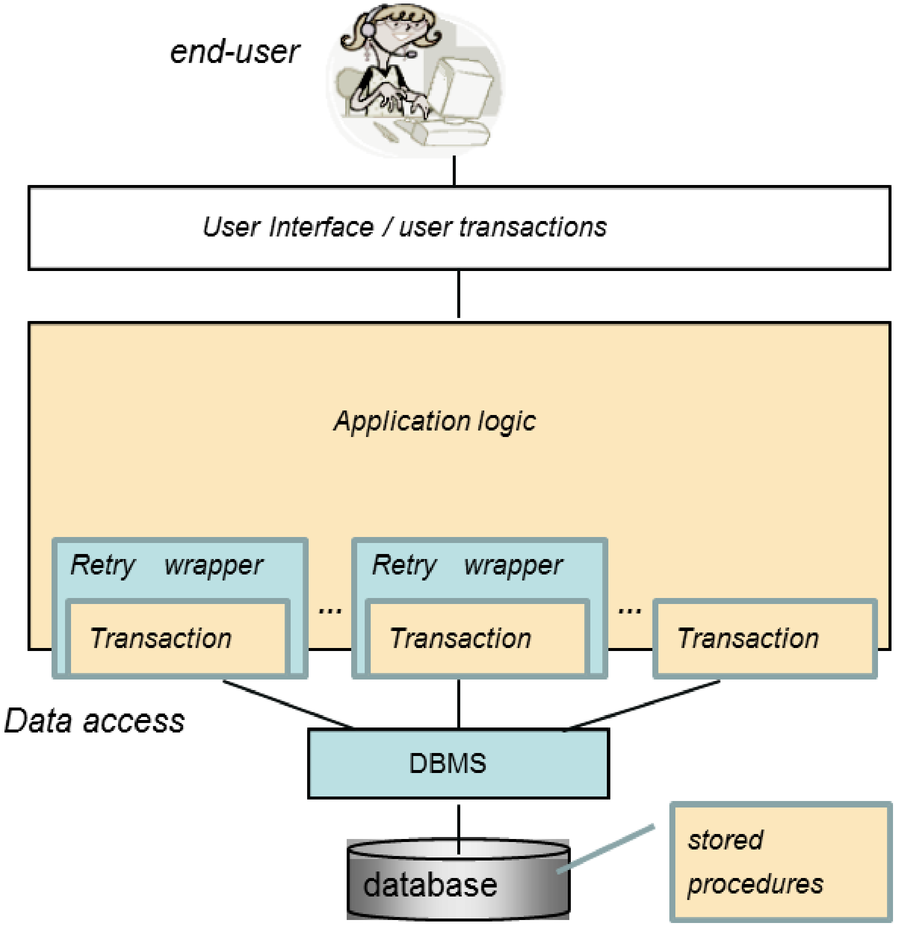
DBMS

database

1. **Reliability!**
2. Security!
3. Performance!

Languages and data
- SQL
- XQuery: XML
- JSON
- RDF

SQL Transactions in Reliable Applications

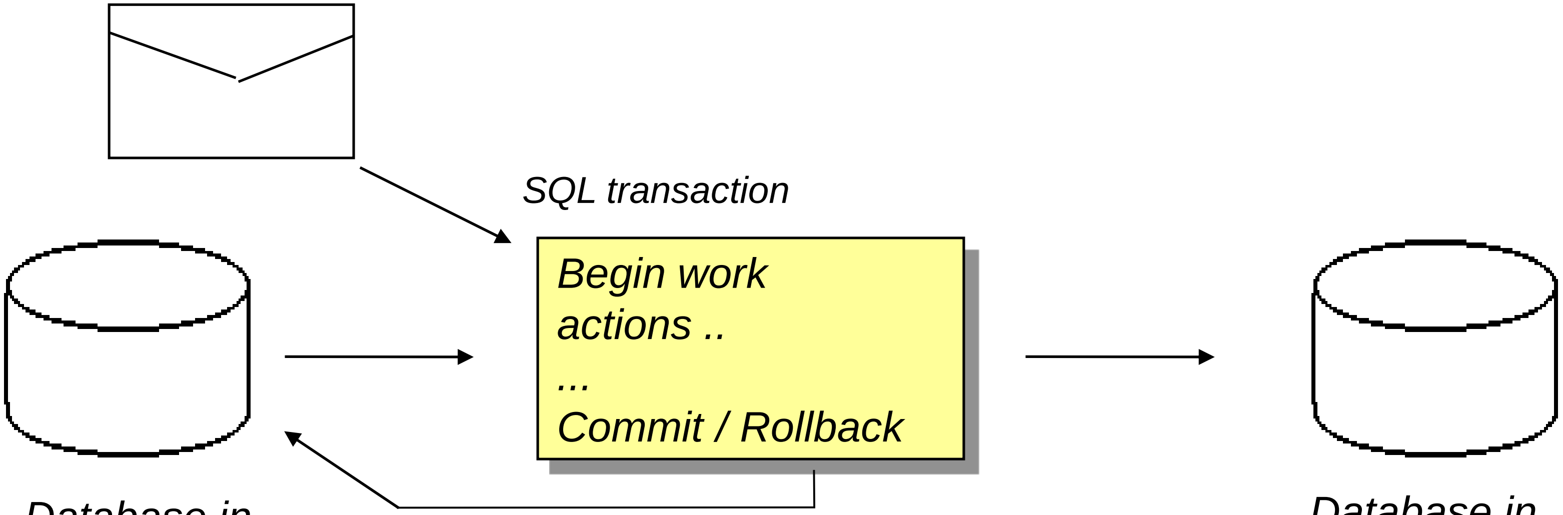


Martti Laiho

SQL Transaction

Context:

“Business transaction
=> Use case
=> User transaction => Sequence of SQL transactions



SQL transaction

Begin work
actions ..
...
Commit / Rollback

Database in consistent state

Database in consistent state

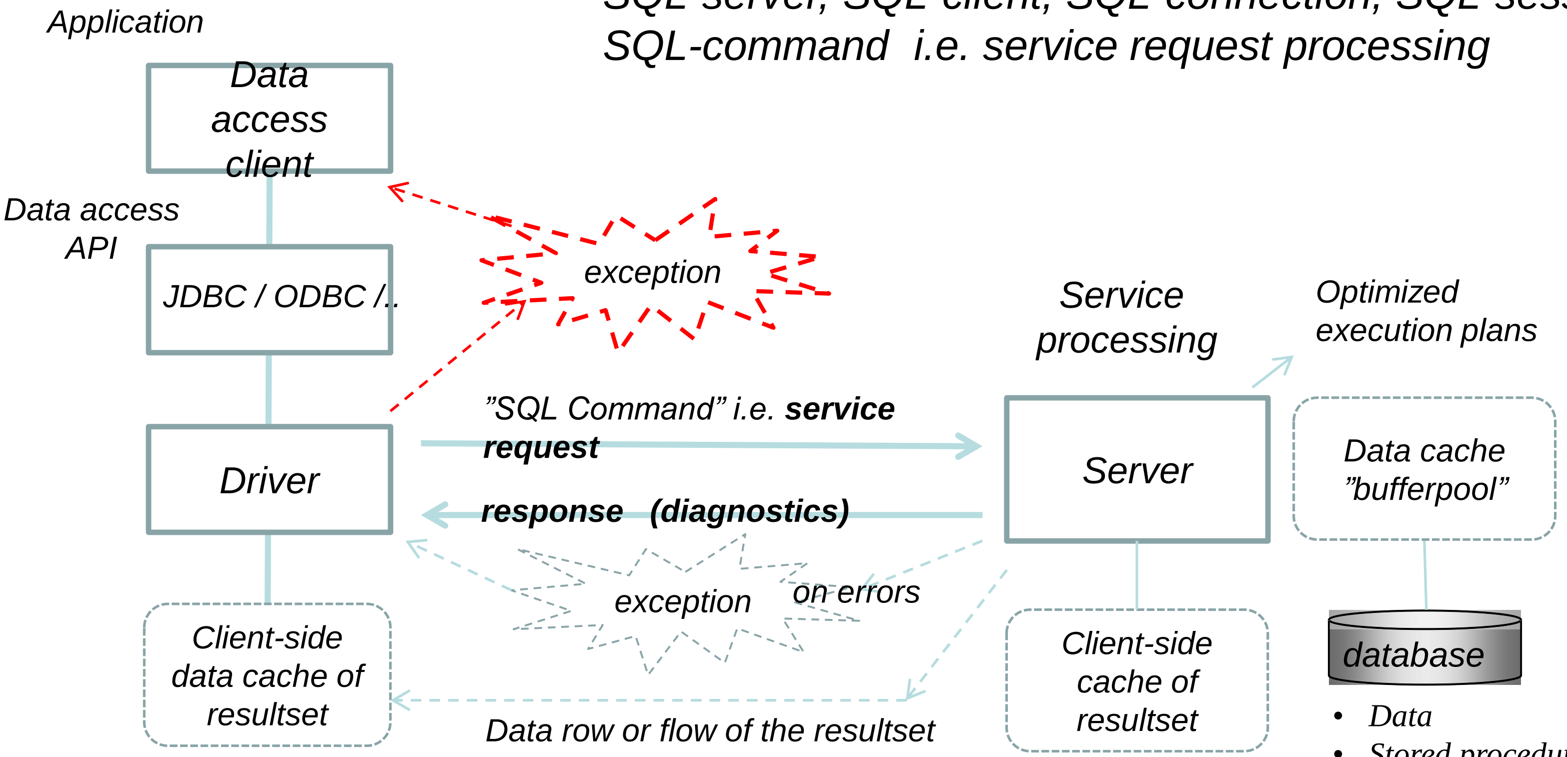
Is a Logical Unit of Work (LUW)
Ideally with the ACID properties
- Atomicity
- Consistency
- Isolation
- Durability

Problems and need for Transactions

- Today society, infra structures, business, and every day life of citizens are dependant on ICT and software using OLTP databases, which provide the most reliable services for storing and retrieving the needed data
- However, inproper access to database services results in erroneous or missing data causing difficulties, lost business, etc
  - Missing orders, shipments, payments, ..
  - Double-bookings, double-invoicing, ..
  - Delays, erroneous information, ..
  - even catastrophes
- Professional use of database services avoids these problems accessing database only by well-designed SQL transactions which are the basic building blocks of fault-tolerant applications

Client / Server Dialogues

SQL-server, SQL-client, SQL-connection, SQL-session
SQL-command i.e. service request processing



Application
Data access client
Data access API
JDBC / ODBC /..
Driver
exception
Service processing
Optimized execution plans
"SQL Command" i.e. **service request**
Server
Data cache "bufferpool"
response   (diagnostics)
exception
on errors
Client-side data cache of resultset
Client-side cache of resultset
database
Data row or flow of the resultset

- *Data*
- *Stored procedures*
- *Execution plans*

Client protocols:

- *Shared Memory*
- *TCP/IP*
- *named pipes*

Diagnostics: SQLcode, SQLSTATE

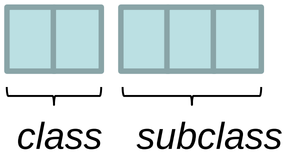
ISO SQL-89 SQLcode:

Integer:

100	No data
0	successful execution
< 0	errors

ISO SQL-92 SQLSTATE:

String of 5 characters:



Successful execution	0 0	0 0 0	
Warning	0 1	n n n	
No data	0 2	0 0 0	
. . .			
Transaction rollback	4 0	0 0 0	
		0 0 1	Serialization failure
		0 0 2	Integrity constraint violation
		0 0 3	Statement completion unknown
		0 0 4	Triggred action exception

etc - lots of standardized and implementation dependent codes

ISO SQL:1999 Get Diagnostics …

List of diagnostic items, including SQLSTATE and number of rows. Only few implementations this far

Structures for using Diagnostics

DB2 SQL:

```
<SQL statement>
IF (SQLSTATE <> '00000') THEN
     <error handling>
END IF;
```

compare with Java:

```
... throws SQLexception {
...
try {
     ...
     <JDBC statement(s)>
}
catch (SQLException ex) {
     <exception handling>
}
```

Oracle PL/SQL:

```
BEGIN
     <processing>
EXCEPTION
WHEN <exception name> THEN
     <exception handling>;
...
WHEN OTHERS THEN
     err_code := sqlcode;
     err_text := sqlerrm;
     <exception handling>;
END;
```

Transact-SQL of SQL Server:

```
BEGIN TRY
          <T-SQL statement(s)>
END TRY
BEGIN CATCH
       <exception handling based on
                    ERROR_NUMBER(),
                    ERROR_SEVERITY(),
                    ERROR_STATE(),
                    ERROR_PROCEDURE(),
                    ERROR_LINE(),
                    ERROR_MESSAGE()> ;
END CATCH;
```

ISO SQL: SET TRANSACTION

Source: Melton & Simon ”SQL:1999”

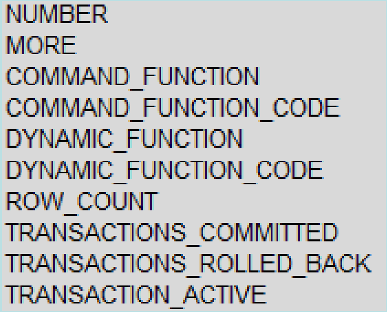
```
SET [LOCAL] TRANSACTION <mode>, …
<mode> ::= [READ ONLY | READ WRITE ] |
                  [ READ UNCOMMITTED |
                    READ COMMITTED  |
                    REAPEATABLE READ |
                    SERIALIZABLE ] |
                   [DIAGNOSTICS SIZE <integer>]
```

SET TRANSACTION tunes the attributes for following transaction.
It cannot be used in an active transaction.

Diagnostics per SQL command consists of ***header*** *and condition* ***details****.*
Diagnostics size defines for how many condition details per SQL command the server will reserve space in the diagnostics area in the transaction context.

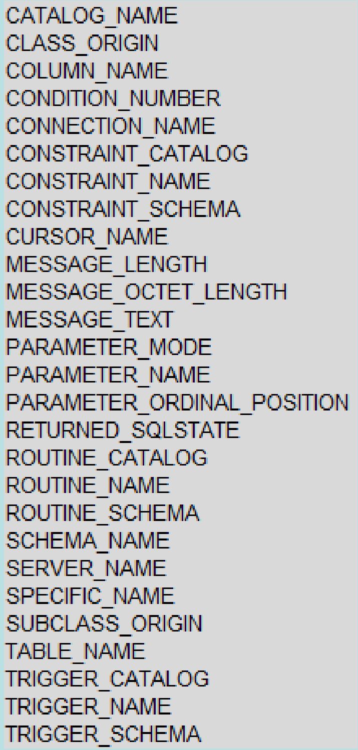
DIAGNOSTICS Items

<header>



NUMBER
MORE
COMMAND_FUNCTION
COMMAND_FUNCTION_CODE
DYNAMIC_FUNCTION
DYNAMIC_FUNCTION_CODE
ROW_COUNT
TRANSACTIONS_COMMITTED
TRANSACTIONS_ROLLED_BACK
TRANSACTION_ACTIVE

<detail>



CATALOG_NAME
CLASS_ORIGIN
COLUMN_NAME
CONDITION_NUMBER
CONNECTION_NAME
CONSTRAINT_CATALOG
CONSTRAINT_NAME
CONSTRAINT_SCHEMA
CURSOR_NAME
MESSAGE_LENGTH
MESSAGE_OCTET_LENGTH
MESSAGE_TEXT
PARAMETER_MODE
PARAMETER_NAME
PARAMETER_ORDINAL_POSITION
RETURNED_SQLSTATE
ROUTINE_CATALOG
ROUTINE_NAME
ROUTINE_SCHEMA
SCHEMA_NAME
SERVER_NAME
SPECIFIC_NAME
SUBCLASS_ORIGIN
TABLE_NAME
TRIGGER_CATALOG
TRIGGER_NAME
TRIGGER_SCHEMA

(1) .. (<max diagnostics detail count>)

<SQL statement> ;
GET DIAGNOSTICS <target> = <item> [, . . .]
If SQLSTATE = . . .

SQL GET DIAGNOSTICS

Example of getting diagnostics in MySQL 5.6:

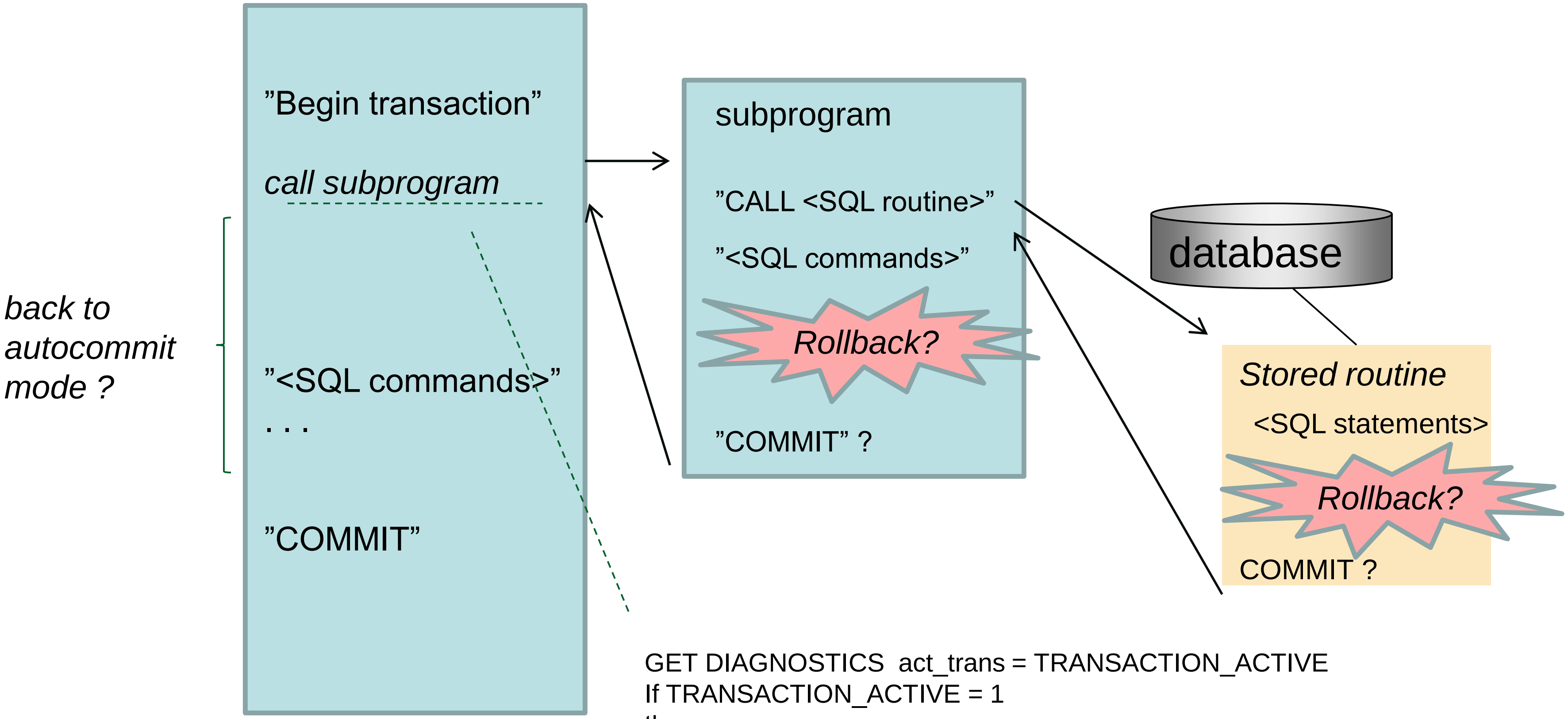
```
INSERT INTO T (id, s) VALUES (2, NULL);
INSERT INTO T (id, s) VALUES (2, 'Hi, I am a duplicate');
```

```
mysql> INSERT INTO T (id, s) VALUES (2, 'Hi, I am a duplicate');
ERROR 1062 (23000): Duplicate entry '2' for key 'PRIMARY'
```

```
GET DIAGNOSTICS @rowcount = ROW_COUNT;
GET DIAGNOSTICS CONDITION 1
        @sqlstate = RETURNED_SQLSTATE,
        @sqlcode = MYSQL_ERRNO ;
SELECT @sqlstate, @sqlcode, @rowcount;
```

```
mysql> SELECT @sqlstate, @sqlcode, @rowcount;
+-----------+----------+-----------+
| @sqlstate | @sqlcode | @rowcount |
+-----------+----------+-----------+
| 23000     |     1062 |        -1 |
+-----------+----------+-----------+
1 row in set (0.00 sec)
```

Potential errors



back to autocommit mode ?

”Begin transaction”

call subprogram

”<SQL commands>”

. . .

”COMMIT”

subprogram

”CALL <SQL routine>”

”<SQL commands>”

Rollback?

”COMMIT” ?

database

Stored routine

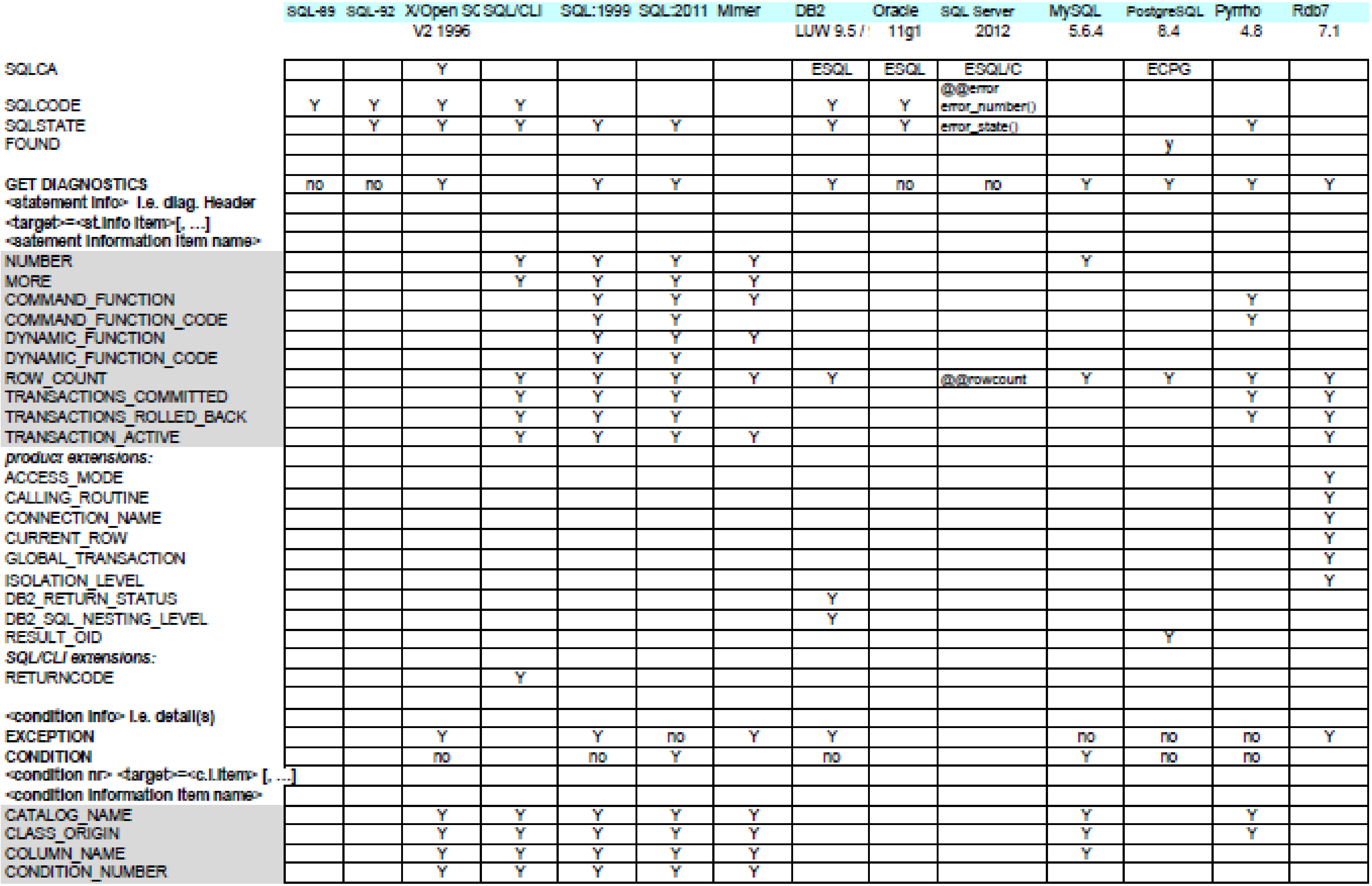
<SQL statements>

Rollback?

COMMIT ?

```
GET DIAGNOSTICS  act_trans = TRANSACTION_ACTIVE
If TRANSACTION_ACTIVE = 1
then …
else
AutoCommit mode ?
```

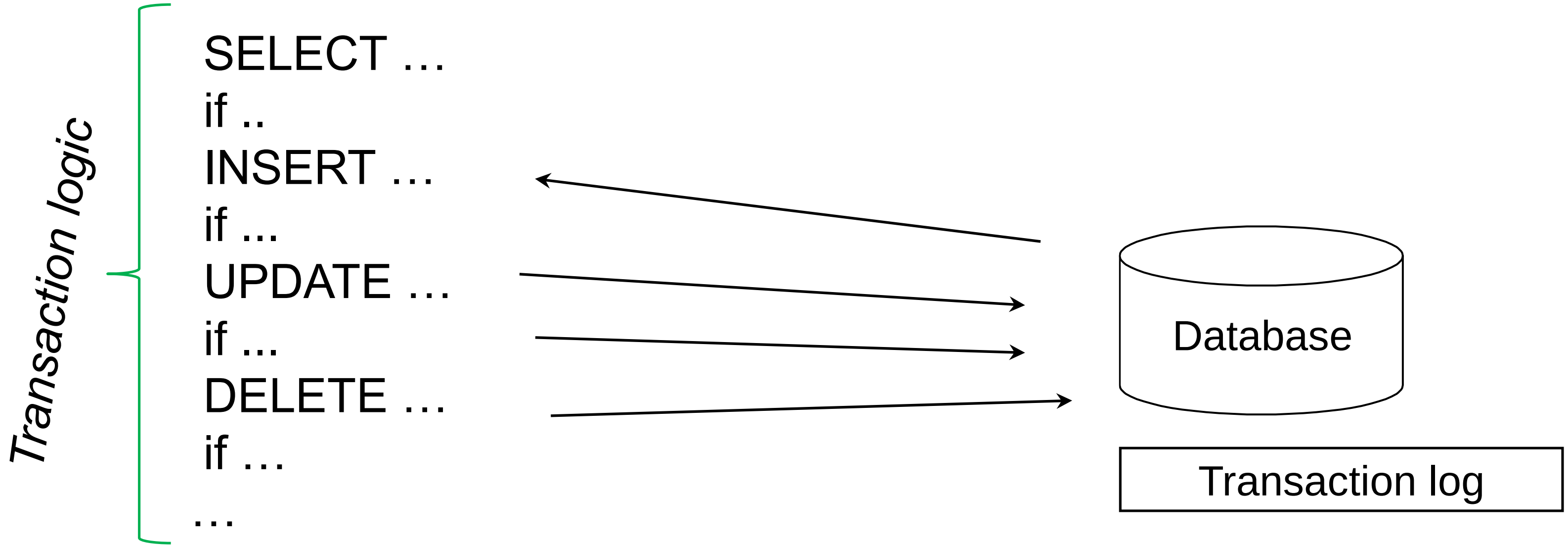
Diagnostics



	SQL-89	SQL-92	X/Open SC	SQL/CLI	SQL:1999	SQL:2011	Mimer	DB2	Oracle	SQL Server	MySQL	PostgreSQL	Pyrrho	Rdb7
			V2 1996					LUW 9.5 /	11g1	2012	5.6.4	8.4	4.8	7.1
SQLCA			Y					ESQL	ESQL	ESQL/C		ECPG		
SQLCODE	Y	Y	Y	Y				Y	Y	@@error error_number()				
SQLSTATE		Y	Y	Y	Y	Y		Y	Y	error_state()			Y	
FOUND												y		
GET DIAGNOSTICS	no	no	Y		Y	Y		Y	no	no	Y	Y	Y	Y
<statement info> i.e. diag. Header														
<target>=<st.info item>[, ...]														
<satement information item name>														
NUMBER				Y	Y	Y	Y				Y			
MORE				Y	Y	Y	Y							
COMMAND_FUNCTION					Y	Y	Y						Y	
COMMAND_FUNCTION_CODE					Y	Y							Y	
DYNAMIC_FUNCTION					Y	Y	Y							
DYNAMIC_FUNCTION_CODE					Y	Y								
ROW_COUNT				Y	Y	Y	Y	Y		@@rowcount	Y	Y	Y	Y
TRANSACTIONS_COMMITTED				Y	Y	Y							Y	Y
TRANSACTIONS_ROLLED_BACK				Y	Y	Y							Y	Y
TRANSACTION_ACTIVE				Y	Y	Y	Y							Y
product extensions:														
ACCESS_MODE														Y
CALLING_ROUTINE														Y
CONNECTION_NAME														Y
CURRENT_ROW														Y
GLOBAL_TRANSACTION														Y
ISOLATION_LEVEL														Y
DB2_RETURN_STATUS								Y						
DB2_SQL_NESTING_LEVEL								Y						
RESULT_OID												Y		
SQL/CLI extensions:														
RETURNCODE				Y										
<condition info> i.e. detail(s)														
EXCEPTION			Y		Y	no	Y	Y			no	no	no	Y
CONDITION			no		no	Y		no			Y	no	no	
<condition nr> <target>=<c.i.item> [, ...]														
<condition information item name>														
CATALOG_NAME			Y	Y	Y	Y	Y				Y		Y	
CLASS_ORIGIN			Y	Y	Y	Y	Y				Y		Y	
COLUMN_NAME			Y	Y	Y	Y	Y				Y			
CONDITION_NUMBER			Y	Y	Y	Y	Y							

SQL Transaction

<implicit start> or <explicit start>



Transaction logic

```
SELECT …
if ..
INSERT …
if ...
UPDATE …
if ...
DELETE …
if …
…
```

Database

Transaction log

COMMIT | ROLLBACK

```
<explicit start> ::=  BEGIN WORK
                   |  BEGIN TRANSACTION
                   |  START TRANSACTION
```

ACID SQL transaction

```
[{SET | START}  TRANSACTION    [READ ONLY | READ WRITE ]
          ISOLATION LEVEL  {READ UNCOMMITTED |
                            READ  COMMITTED     |
                            REPEATABLE  READ    |
                            SERIALIZABLE             }
```

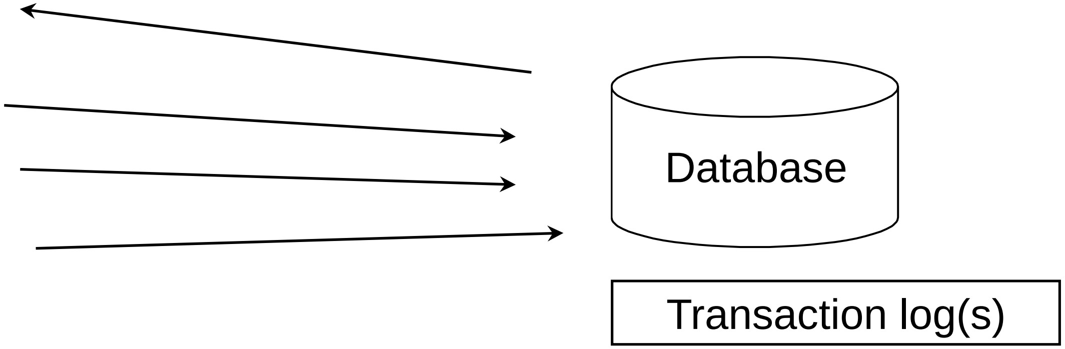
**I**solation

```
 [ if ..] ]
 SET {UNIQUE | REFERENCIAL} CONSTRAINTS
                             {DEFERRED | IMMEDIATE }
[ LOCK TABLE … ]
 SELECT …
 if ..
 INSERT …
 if ...
 UPDATE …
 if ...
 DELETE …
 if …
SAVEPOINT  spn
…
```

**C**onsistency

- by DBMS

- logical



**A**tomicity

**D**urability

```
COMMIT | ROLLBACK
 if ...
```

ROLLBACK

- i.e. automatic transaction recovery is based on use of transaction history which saves addresses and ”before images” of all changed / deleted rows
- For inserted rows the ”before image” is empty
- In ROLLBACK operation the server simply restores the before images of all rows affected by the transaction back to the original addresses

- For more details, see the presentation ”Basics of SQL Transactions”

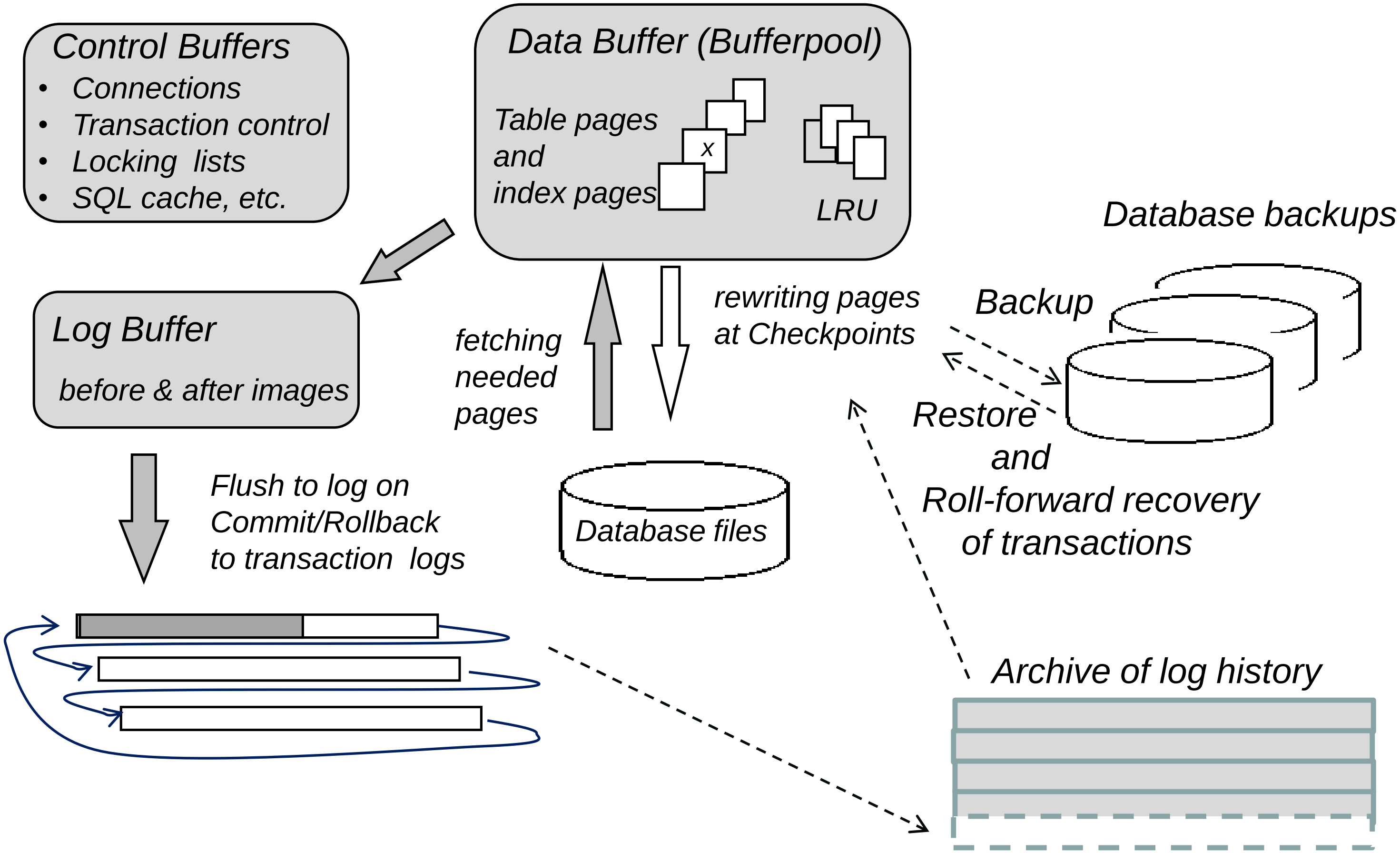
A generic overview of a database server

Database server (instance)

- Processes, threads and caches

DBMS services:
Listener
Server agents
Transaction manager
SQL engine (parser)
Security manager
Query Optimizer
Concurrency manager
(Lock manager)
Deadlock detector
Recovery manager
Relational engine
Memory manager

OpSys:
File manager
Disk manager



Control Buffers

- *Connections*
- *Transaction control*
- *Locking lists*
- *SQL cache, etc.*

Data Buffer (Bufferpool)

Table pages and index pages

x

LRU

Log Buffer

before & after images

fetching needed pages

rewriting pages at Checkpoints

Flush to log on Commit/Rollback to transaction logs

Database files

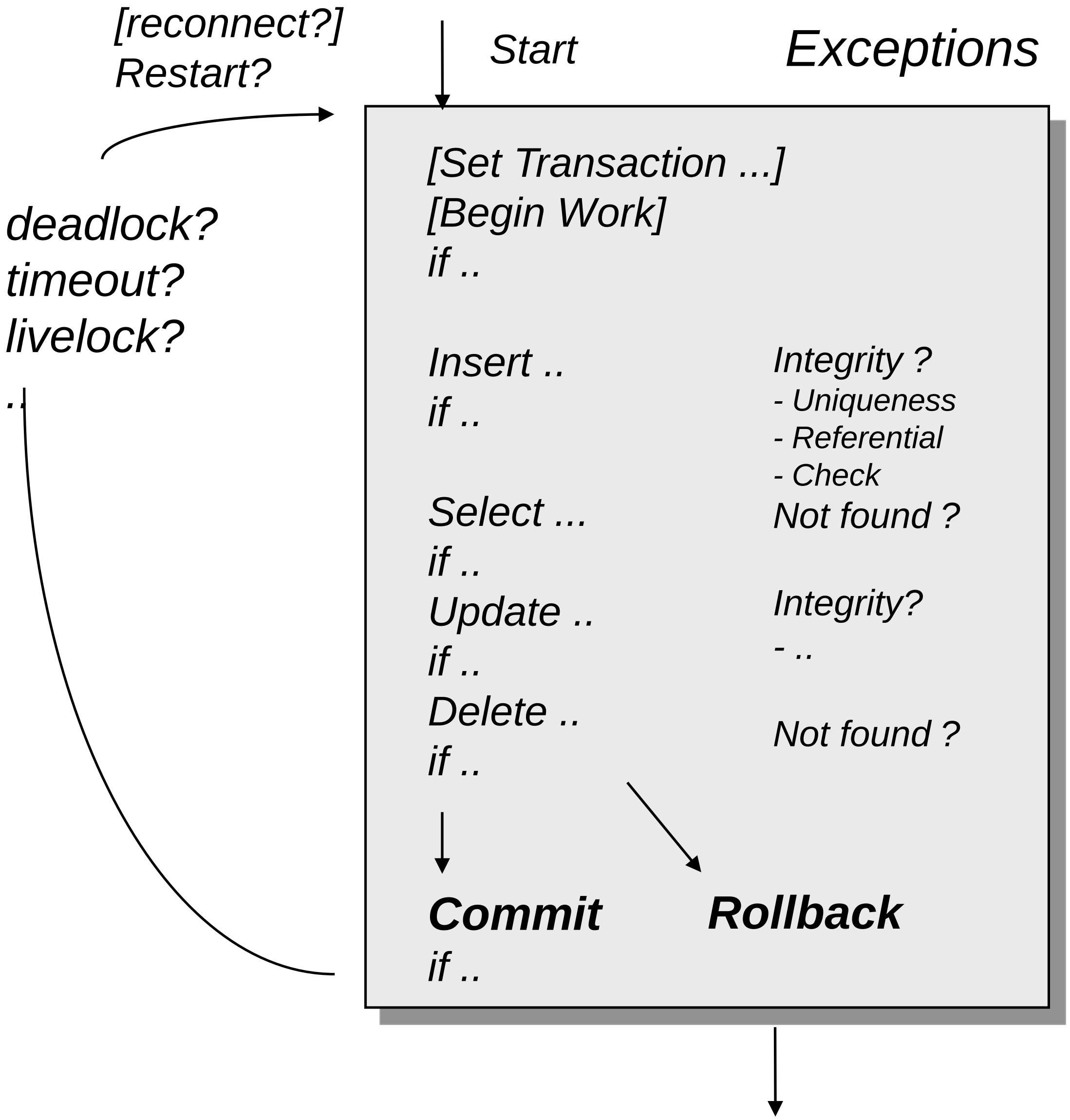
Database backups

Backup

Restore and Roll-forward recovery of transactions

Archive of log history

Diagnostics needed after every SQL command



[reconnect?]
Restart?

deadlock?
timeout?
livelock?
..

Start

Exceptions

[Set Transaction ...]
[Begin Work]
if ..

Insert ..
if ..

Integrity ?
- Uniqueness
- Referential
- Check

Select ...
if ..

Not found ?

Update ..
if ..

Integrity?
- ..

Delete ..
if ..

Not found ?

Commit
if ..

Rollback

Errors

Restart?

multiprogramming
- limit, timeout

YES

syntax error in dynamic SQL
On plan re-optimizing
- Invalid objects/privileges

Serializability
- conflict
- deadlock
- timeout

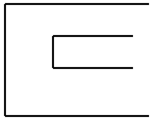
YES

Services of
- DBMS buffers, etc
- OpSys
- data communication
- HW problems

Martti Laiho 1998-2009

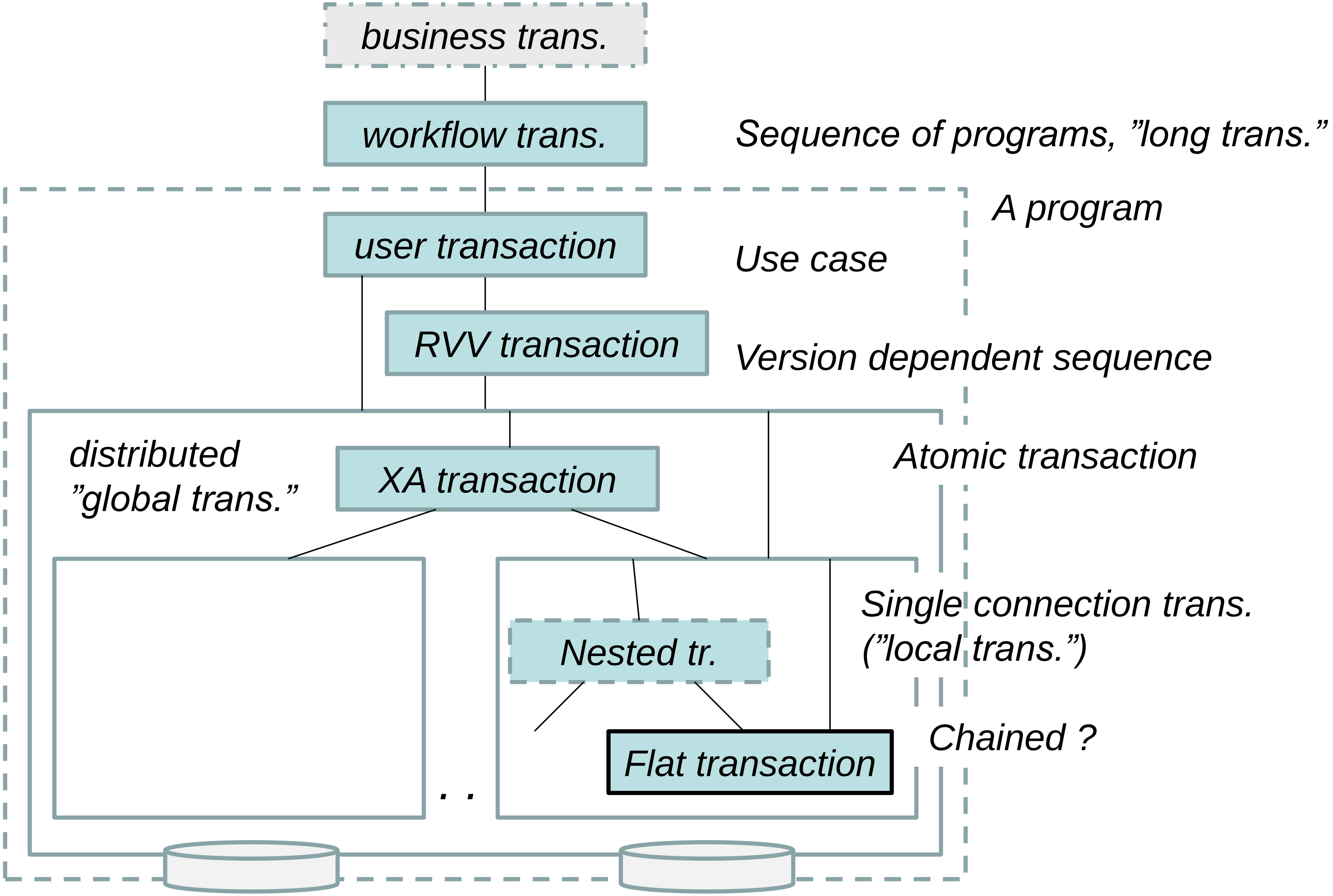
SQL Transaction Models

- Flat transaction
  - ACID properties
    - Atomicity (all or nothing!)
    - Consistency (integrity constraints)
    - Isolation (based on MGLCC, MVCC, or OCC concurrency control)
    - Durability (persistency)
  - Savepoints
    - Atomicity in parts
  - Isolation levels
    - AC(I-)D - compromizing for performance
    - Default for commands in the transaction
    - Can be defined differently for cursors and single commands
- Nested transactions



- Chained transactions

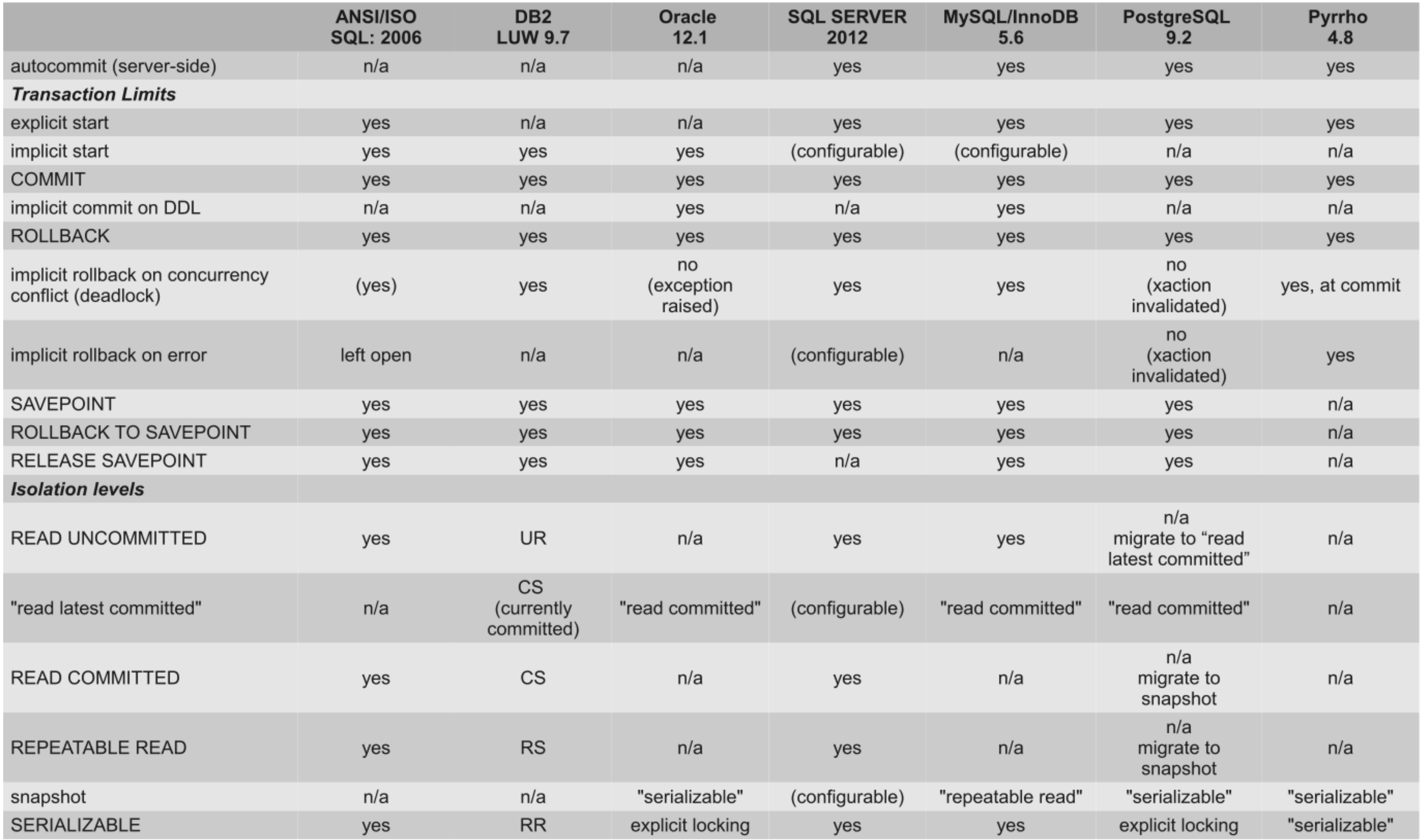
Hierarchy of Transaction Concepts



Differently behaving products

- As default in AUTOCOMMIT mode ?
- Implicit or explicit starts of transactions
- Implicit COMMIT on DDL ?
- Default isolation
- What is considered as error or Warning ?
  - Value truncation, value overflow, …
- Error in command
  - Rolls back the command
  - Rolls back the command and discards commands until end of transaction
  - Rolls back the transaction
- Concurrency control mechanism

ISO/SQL xacts and product implementations



	ANSI/ISO SQL: 2006	DB2 LUW 9.7	Oracle 12.1	SQL SERVER 2012	MySQL/InnoDB 5.6	PostgreSQL 9.2	Pyrrho 4.8
autocommit (server-side)	n/a	n/a	n/a	yes	yes	yes	yes
Transaction Limits							
explicit start	yes	n/a	n/a	yes	yes	yes	yes
implicit start	yes	yes	yes	(configurable)	(configurable)	n/a	n/a
COMMIT	yes	yes	yes	yes	yes	yes	yes
implicit commit on DDL	n/a	n/a	yes	n/a	yes	n/a	n/a
ROLLBACK	yes	yes	yes	yes	yes	yes	yes
implicit rollback on concurrency conflict (deadlock)	(yes)	yes	no (exception raised)	yes	yes	no (xaction invalidated)	yes, at commit
implicit rollback on error	left open	n/a	n/a	(configurable)	n/a	no (xaction invalidated)	yes
SAVEPOINT	yes	yes	yes	yes	yes	yes	n/a
ROLLBACK TO SAVEPOINT	yes	yes	yes	yes	yes	yes	n/a
RELEASE SAVEPOINT	yes	yes	yes	n/a	yes	yes	n/a
Isolation levels							
READ UNCOMMITTED	yes	UR	n/a	yes	yes	n/a migrate to “read latest committed”	n/a
"read latest committed"	n/a	CS (currently committed)	"read committed"	(configurable)	"read committed"	"read committed"	n/a
READ COMMITTED	yes	CS	n/a	yes	n/a	n/a migrate to snapshot	n/a
REPEATABLE READ	yes	RS	n/a	yes	n/a	n/a migrate to snapshot	n/a
snapshot	n/a	n/a	"serializable"	(configurable)	"repeatable read"	"serializable"	"serializable"
SERIALIZABLE	yes	RR	explicit locking	yes	yes	explicit locking	"serializable"

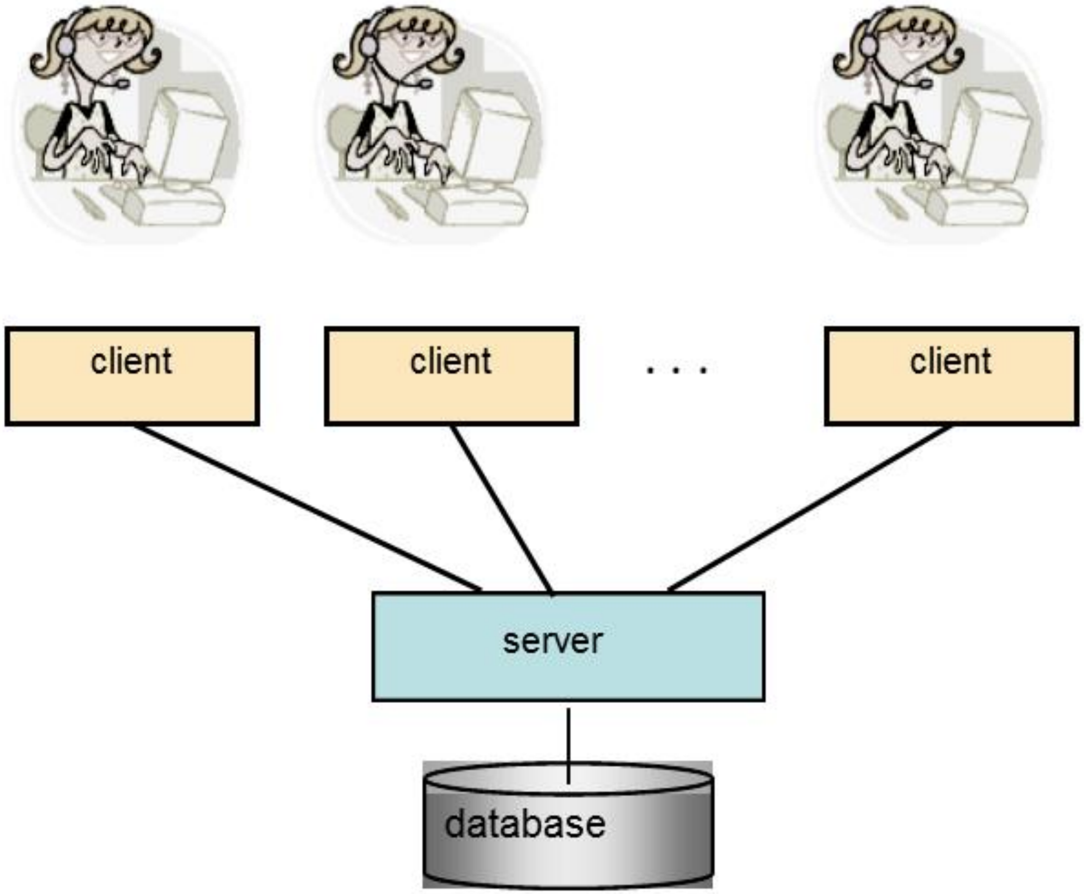
Single-user Transaction Experiments

- Students start their private copies of DebianDB
- Teacher demonstrates the first steps
  making sure that all students can repeat every step
  getting started with the experiment
- The same DBMS product is selected to be studied,
  - for example MySQL/InnoDB
- A single SQL session is started in a terminal window
- Students make notes of the transaction experiments or experiences are discussed

Experiments with help of the instructor

- 1.1
- 1.2
- 1.3
- 1.4
- 1.5
- 1.6
- 1.7

Competing Transactions in Multi-user Environment



Concurrency Control Technologies

- SQL standard defines Isolation Levels for transaction context based on **anomalies**, without concerning the technologies

- Concurrency Control Implementations tuned by Isolation Levels:
  - Optimistic Concurrency Control (OCC) 100% isolated
  - Locking Schemes (MGL, LSCC) 0% ..100%
  - Multi-Versioning (MVCC) %?
  - Cursor level concurrency control,

    SELECT .. FOR UPDATE

- Client-side
  - Row Version Verification (RVV) aka. "Optimistic Locking"

Concurrency Problems

Typical anomalies (C J Date, Milton, SQL-92)

1 Lost Update Problem (solved?)

2 Uncommitted Dependency Problem (Dirty Read)

3 Inconsistent Analysis Problems

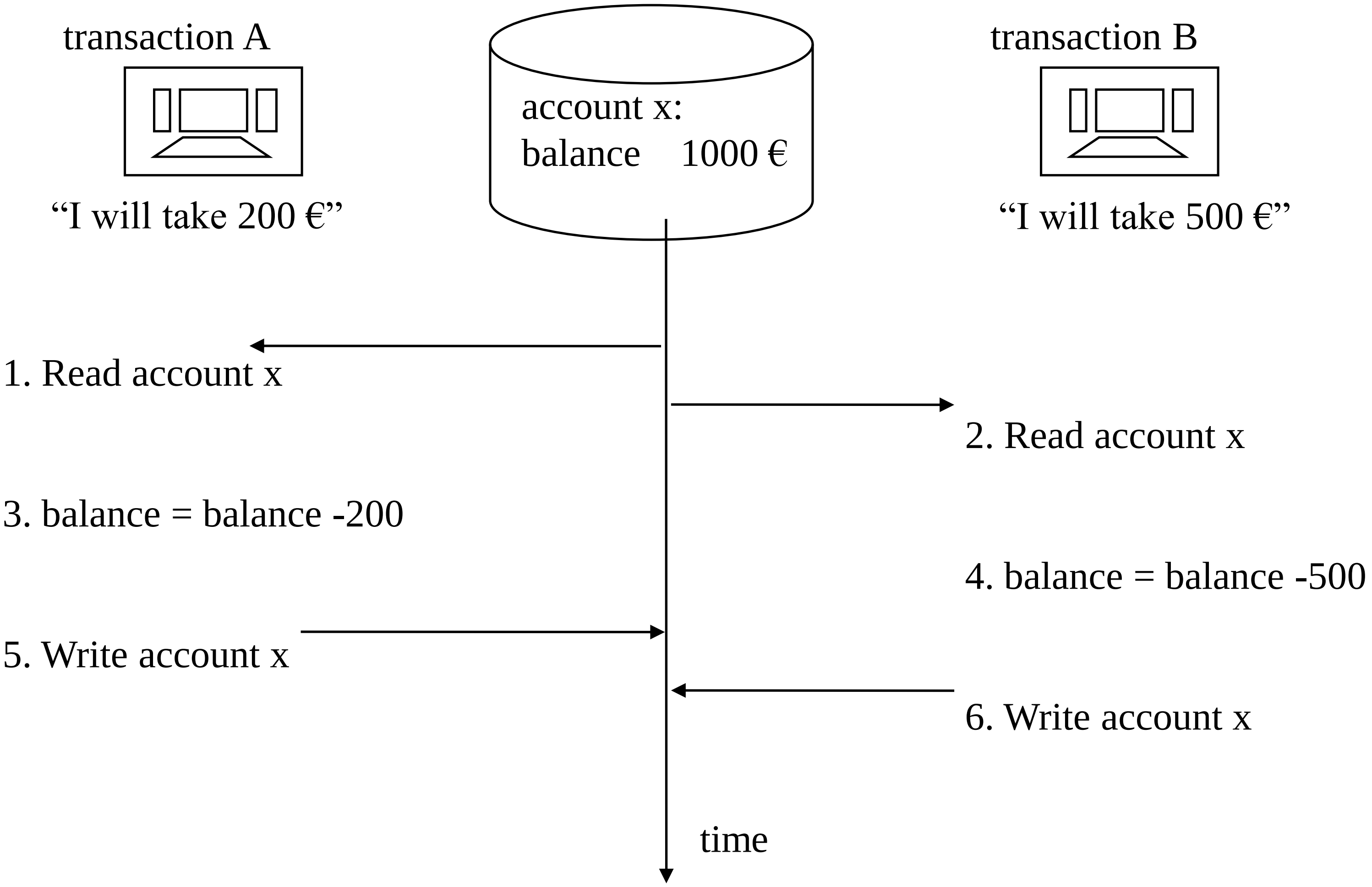
a) Decreasing Read Set (Non-repeatable Read)

b) Increasing Read Set (Phantoms)

1. The Lost Update Problem

C. J. Date: Lost Update

”Tellers”



transaction A

“I will take 200 €”

account x:
balance 1000 €

transaction B

“I will take 500 €”

1. Read account x

2. Read account x

3. balance = balance -200

4. balance = balance -500

5. Write account x

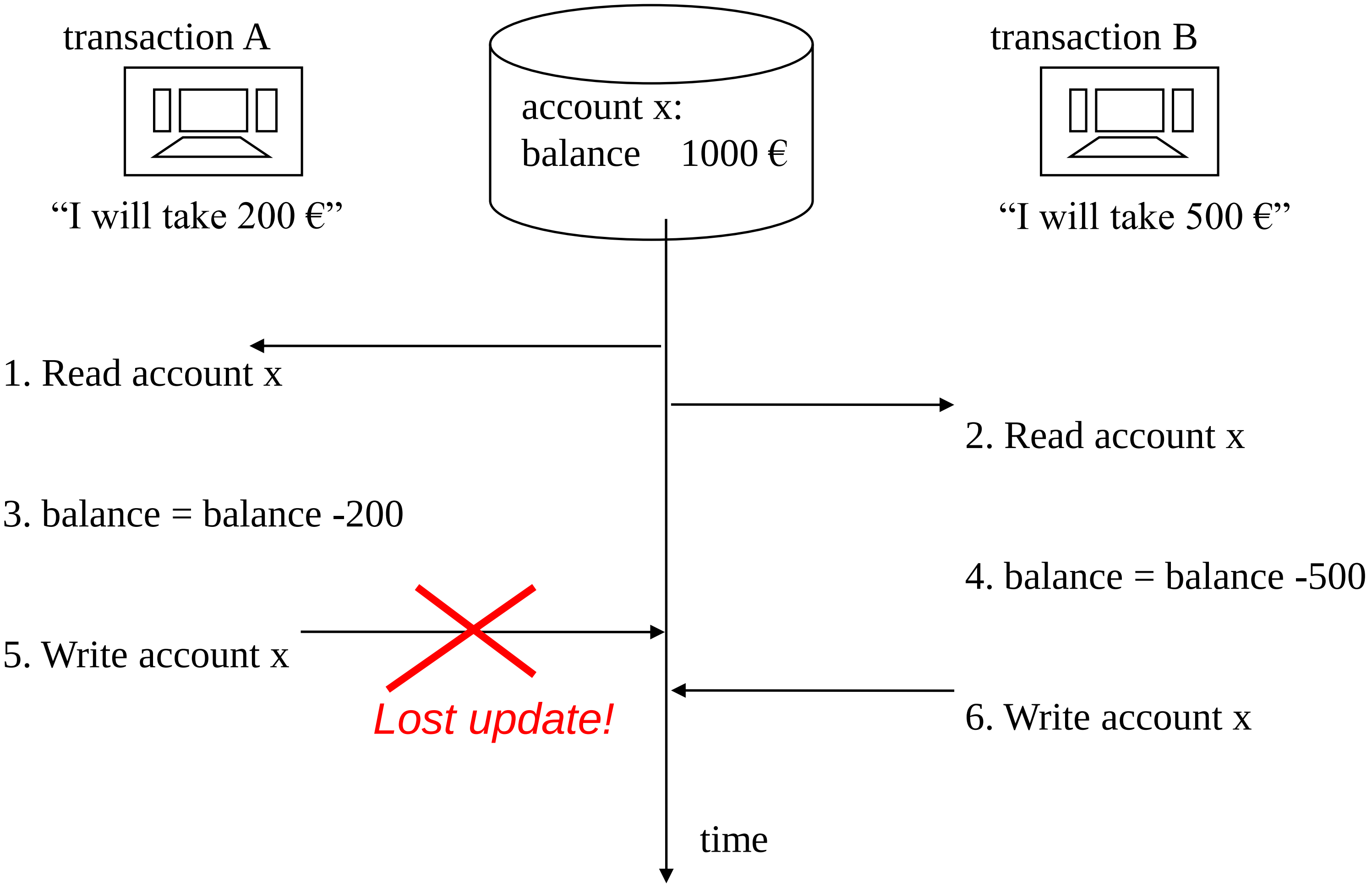
6. Write account x

time

1. The Lost Update Problem

C. J. Date: Lost Update

”Tellers”



transaction A

“I will take 200 €”

account x:
balance 1000 €

transaction B

“I will take 500 €”

1. Read account x

2. Read account x

3. balance = balance -200

4. balance = balance -500

5. Write account x

Lost update!

6. Write account x

time

Concurrency Control by S- and X-locks

Compatibility of S and X locks:

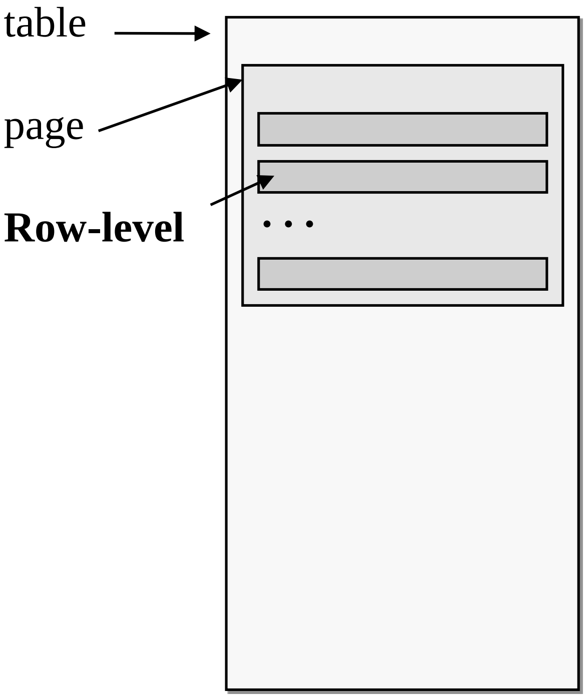
Lock of transaction A to object o

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:

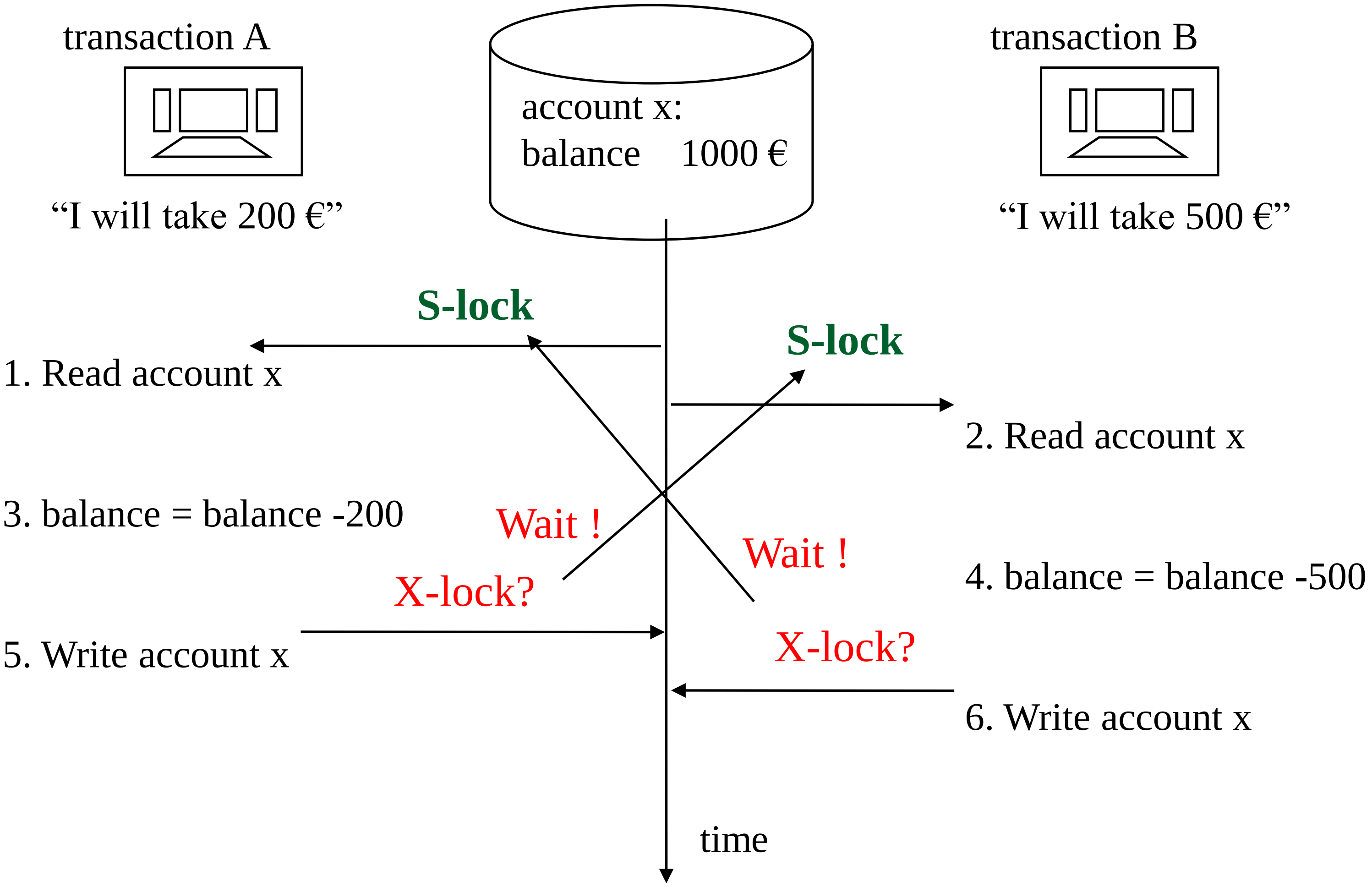


1. The Lost Update Problem

C. J. Date: Lost Update

- Applying the locking scheme:

”Tellers”



transaction A

“I will take 200 €”

account x:
balance 1000 €

transaction B

“I will take 500 €”

S-lock

1. Read account x

S-lock

2. Read account x

3. balance = balance -200

Wait !

Wait !

4. balance = balance -500

X-lock?

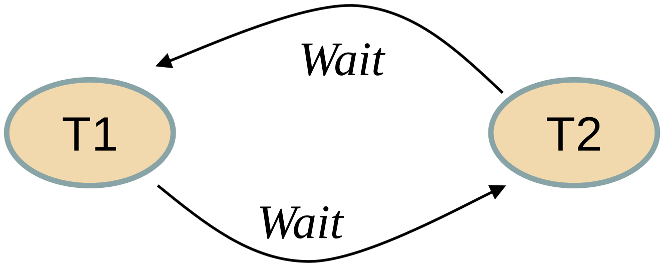
5. Write account x

X-lock?

6. Write account x

time

Deadlock

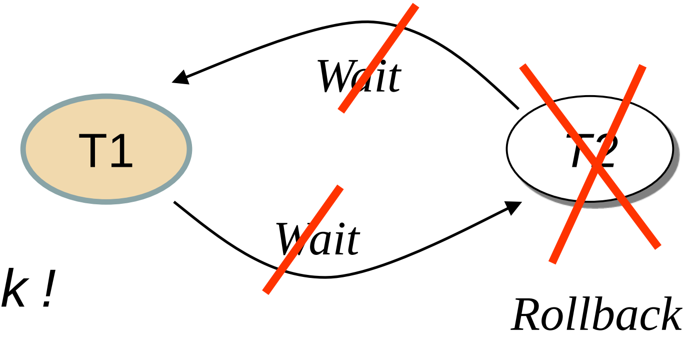


A Cycle of Lock Waits

Modern DBMS systems will detect the deadlock in some seconds (deadlock detection) and solve the waiting cycle

- *selecting the victim*
- *making automatic Rollback (not Oracle)*
- *send error message to the application*

=> *Application must react on the deadlock !*

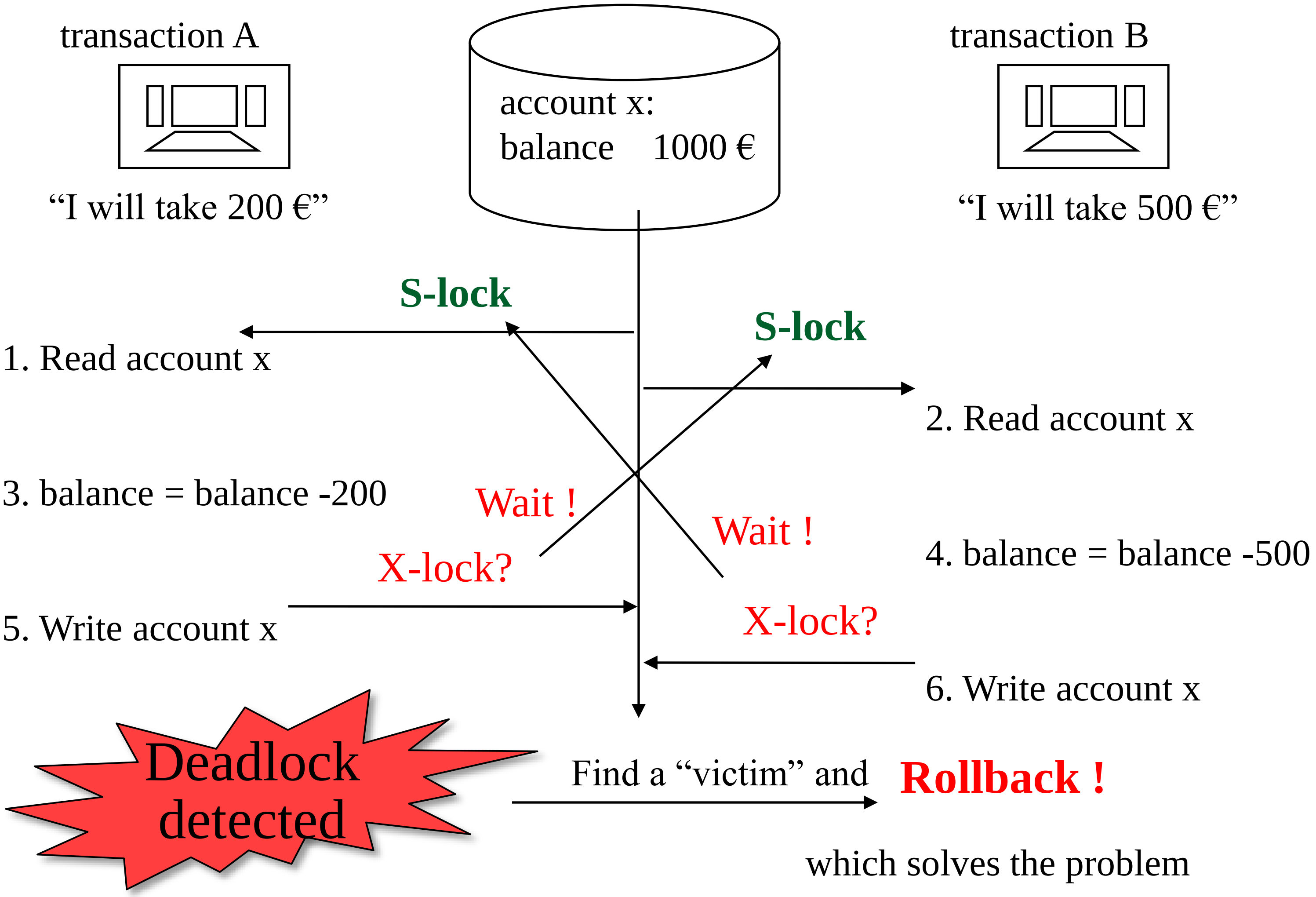


1. The Lost Update Problem

C. J. Date: Lost Update

- Applying the locking scheme:

”Tellers”



transaction A

“I will take 200 €”

account x:
balance 1000 €

transaction B

“I will take 500 €”

S-lock

1. Read account x

S-lock

2. Read account x

3. balance = balance -200

Wait !

Wait !

4. balance = balance -500

X-lock?

5. Write account x

X-lock?

6. Write account x

Deadlock detected

Find a “victim” and **Rollback !**

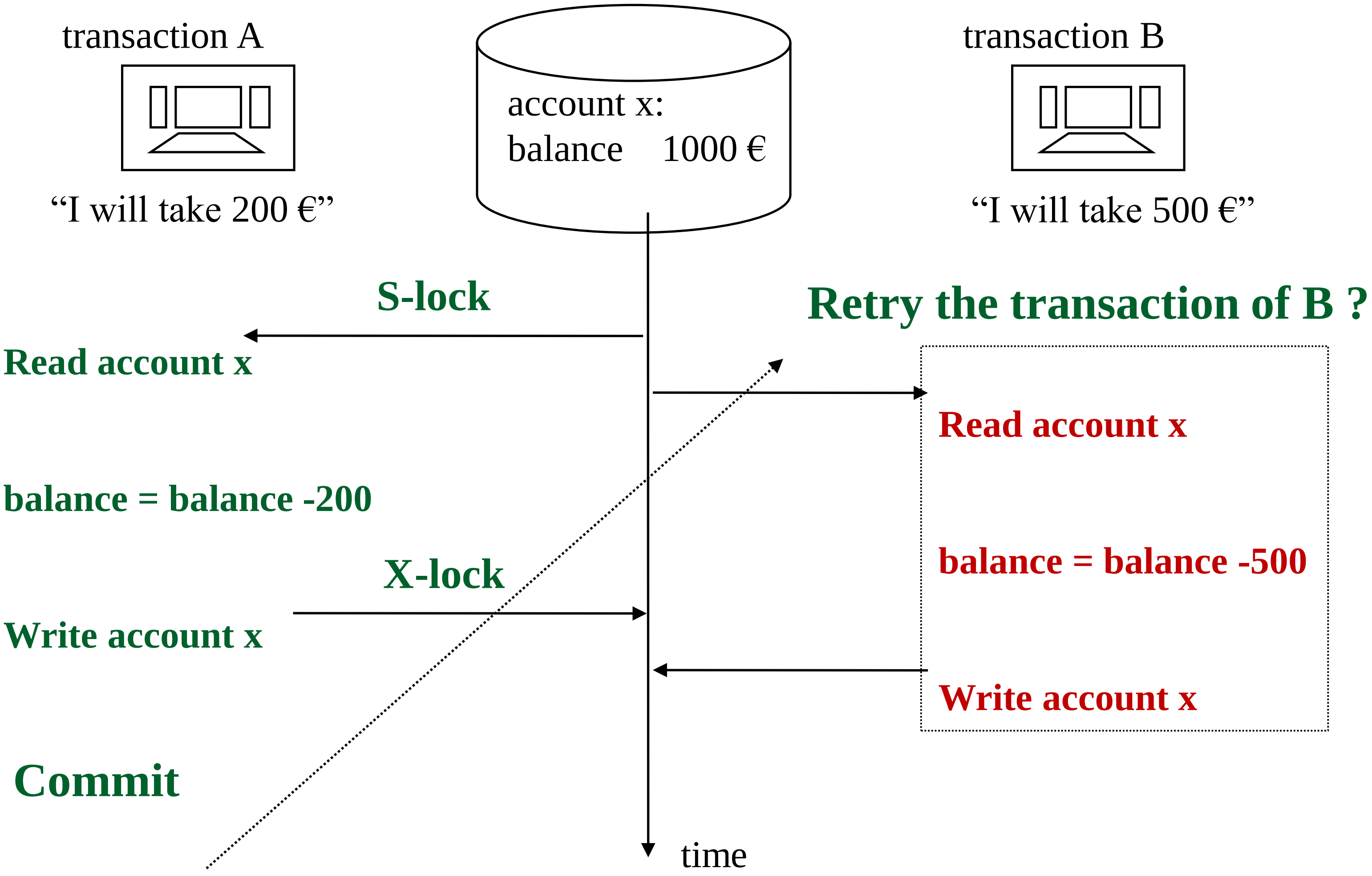
which solves the problem

1. The Lost Update Problem

- solved by locking scheme:

C. J. Date: Lost Update

”Tellers”



transaction A

“I will take 200 €”

account x:
balance 1000 €

transaction B

“I will take 500 €”

S-lock

Read account x

balance = balance -200

X-lock

Write account x

Commit

Retry the transaction of B ?

Read account x

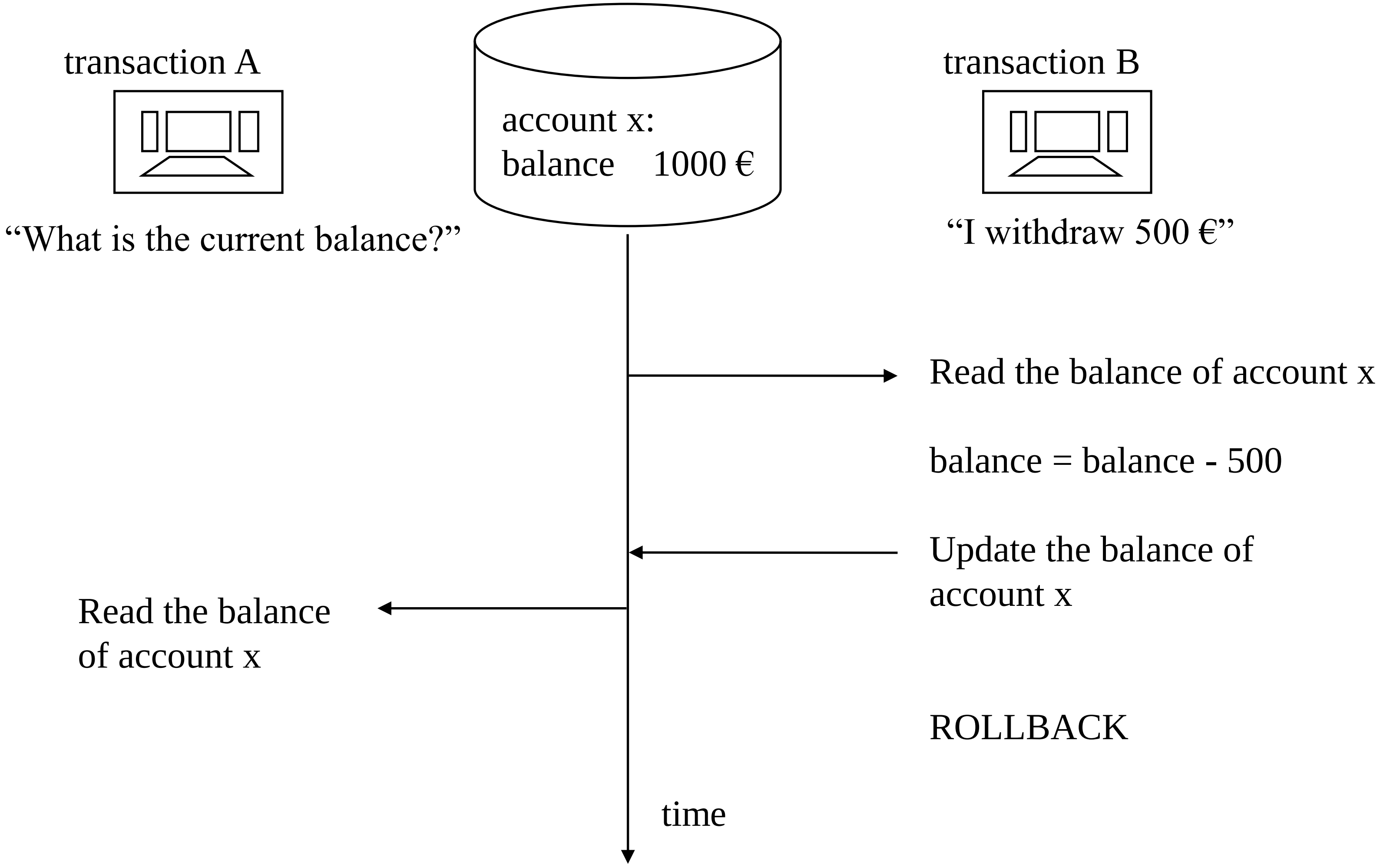
balance = balance -500

Write account x

time

Dirty Read

C.J. Date



transaction A

“What is the current balance?”

account x:
balance 1000 €

transaction B

“I withdraw 500 €”

Read the balance of account x

balance = balance - 500

Update the balance of account x

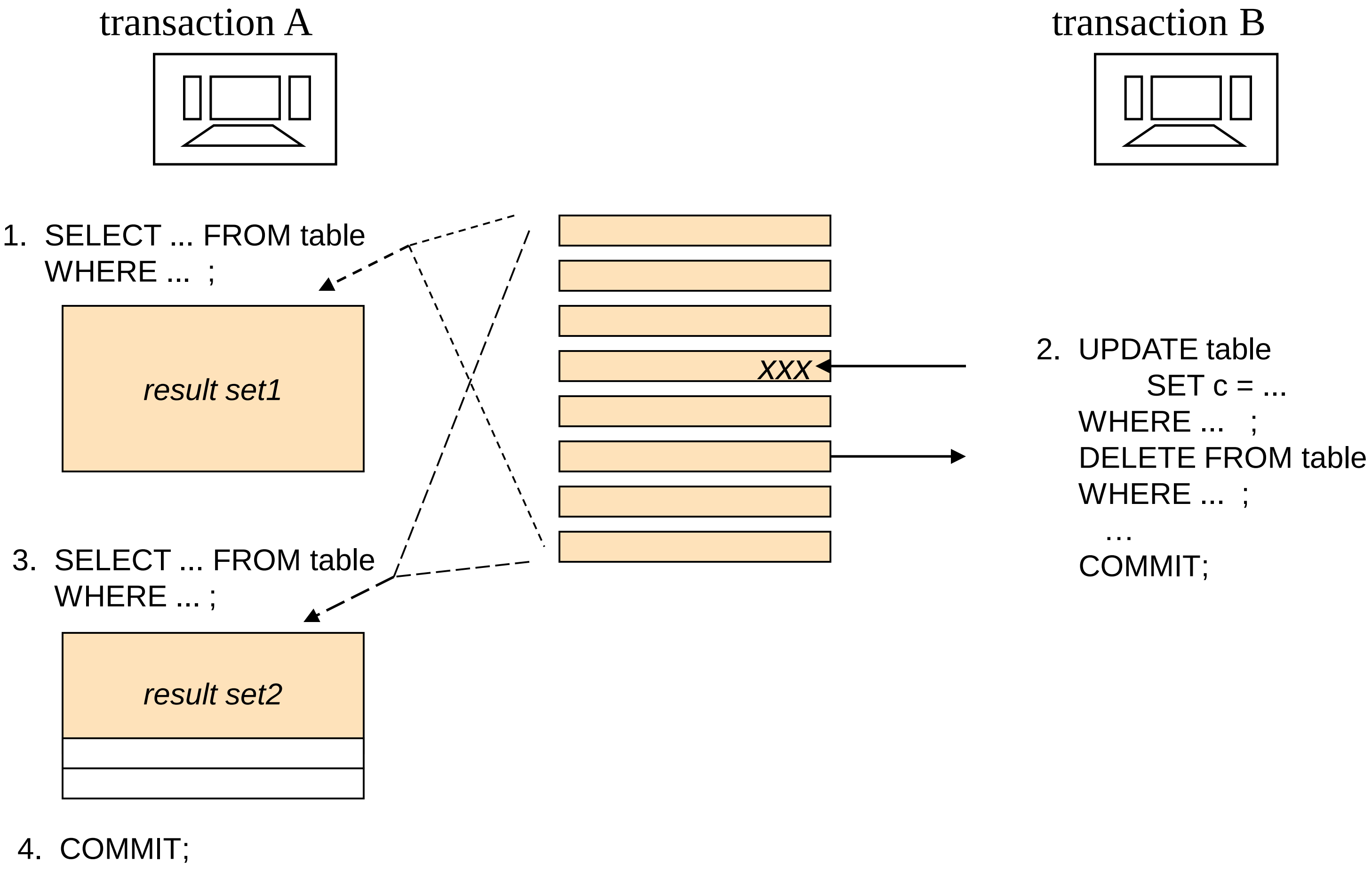
Read the balance of account x

ROLLBACK

time

Non-Repeatable Read

C.J. Date



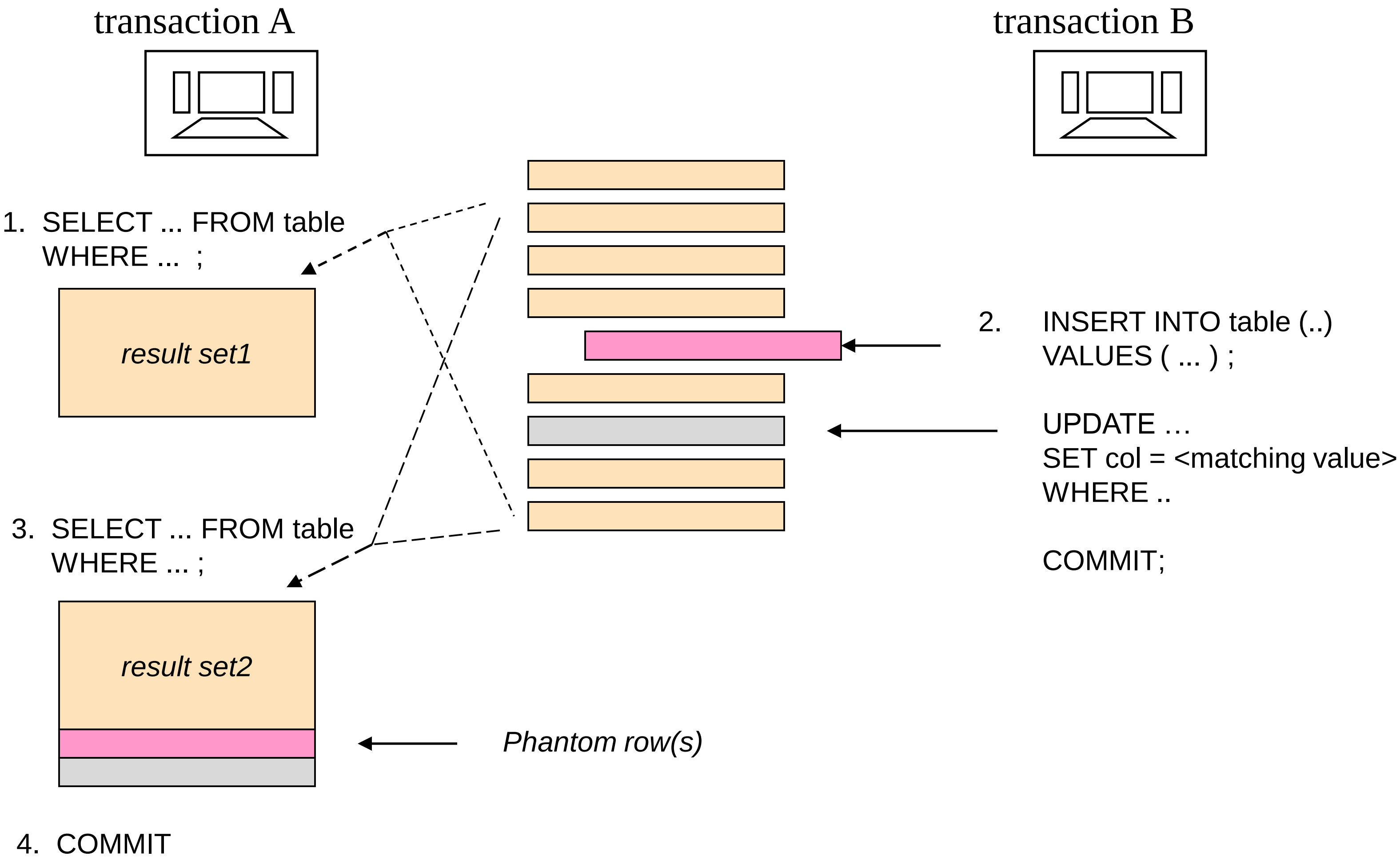
1. SELECT ... FROM table
   WHERE ... ;

2. UPDATE table
   SET c = ...
   WHERE ... ;
   DELETE FROM table
   WHERE ... ;
   …
   COMMIT;

3. SELECT ... FROM table
   WHERE ... ;

4. COMMIT;

Phantom Read



transaction A

transaction B

1. SELECT ... FROM table
   WHERE ... ;

result set1

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

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

   COMMIT;

3. SELECT ... FROM table
   WHERE ... ;

result set2

Phantom row(s)

4. COMMIT

Martti Laiho

ACID SQL transaction

[{SET | START} TRANSACTION   [READ ONLY | READ WRITE]
ISOLATION LEVEL {READ UNCOMMITTED |
READ COMMITTED |
REPEATABLE READ |
SERIALIZABLE   }

***I**solation*

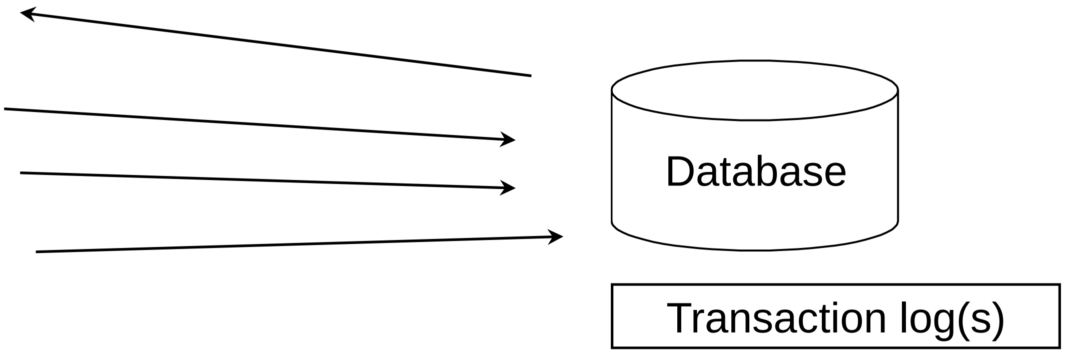
[if ..]]
SET {UNIQUE | REFERENCIAL} CONSTRAINTS
{DEFERRED | IMMEDIATE }
[LOCK TABLE …]
SELECT …
if ..
INSERT …
if ...
UPDATE …
if ...
DELETE …
if …
SAVEPOINT spn
…

***C**onsistency*
- by DBMS
- logical

***A**tomicity*

***D**urability*

COMMIT | ROLLBACK
if ...



Isolation Levels of ISO SQL

Anomalies: *Isolation Level:*	***Lost Update***	***Dirty Read***	***Nonrepeatable Read***	***Phantoms***
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*

Isolation levels can be explained by objects and duration in S-locking
preventing only the transaction itself against certain anomalies,
but can’t prevent concurrent transactions from dirty reads, etc
i.e. can’t provide strict isolation as defined by Haerder and Reuter

Locking Scheme Concurrency Control (LSCC)

Compatibility of S and X locks

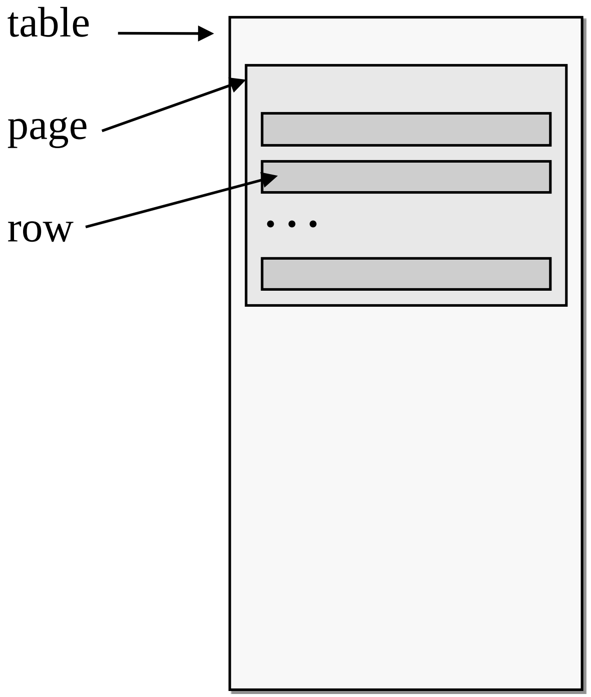
Lock of transaction A to object o

Lock request of transaction B to object o

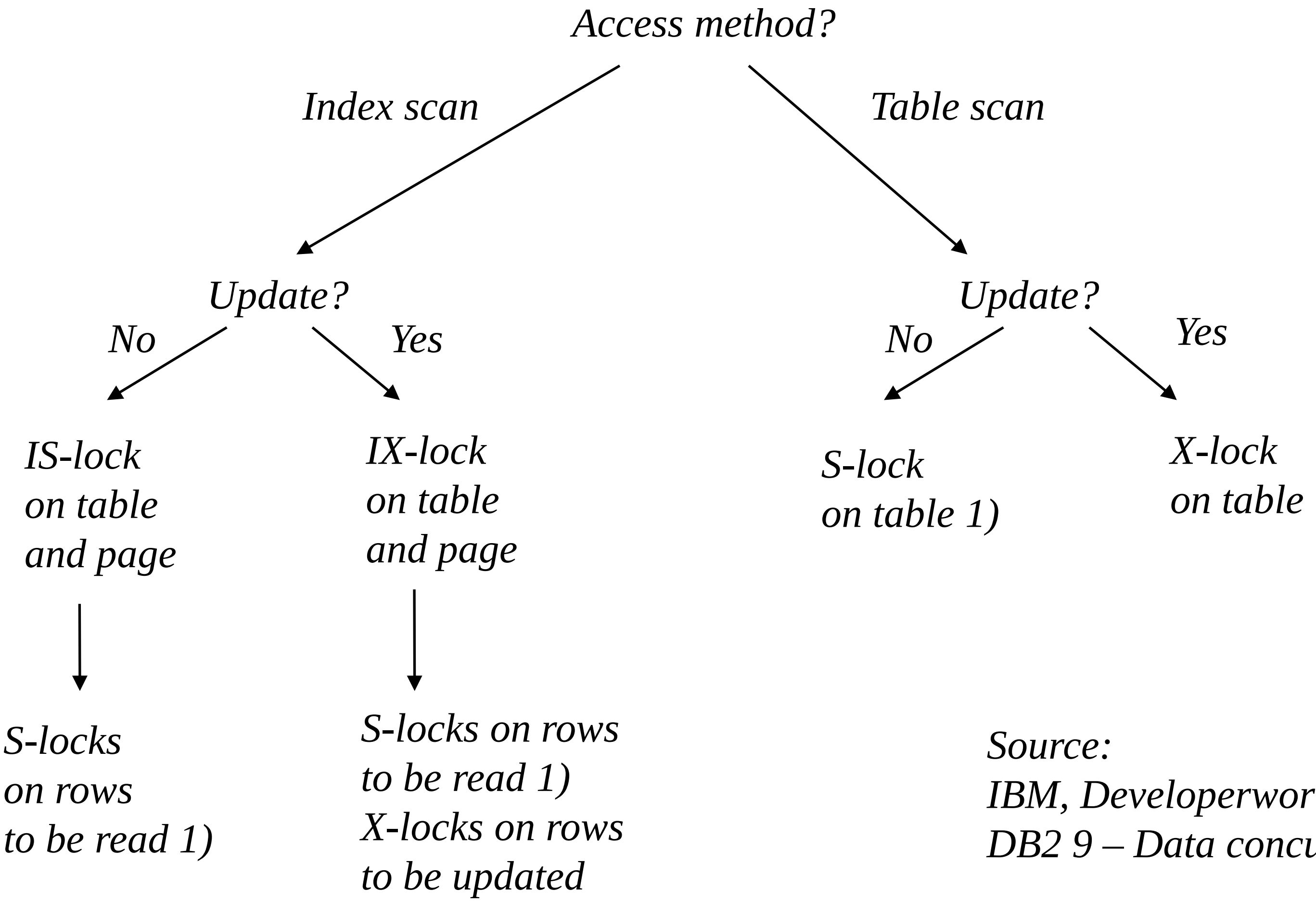
	Shared	e**X**clusive
Shared	**Grant**	**Wait !**
e**X**clusive	**Wait !**	**Wait !**

- 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:



Locking Mode is selected by Optimizer



Access method?

Index scan

Table scan

Update?

No

Yes

IS-lock
on table
and page

IX-lock
on table
and page

S-locks
on rows
to be read 1)

S-locks on rows
to be read 1)
X-locks on rows
to be updated

Update?

No

Yes

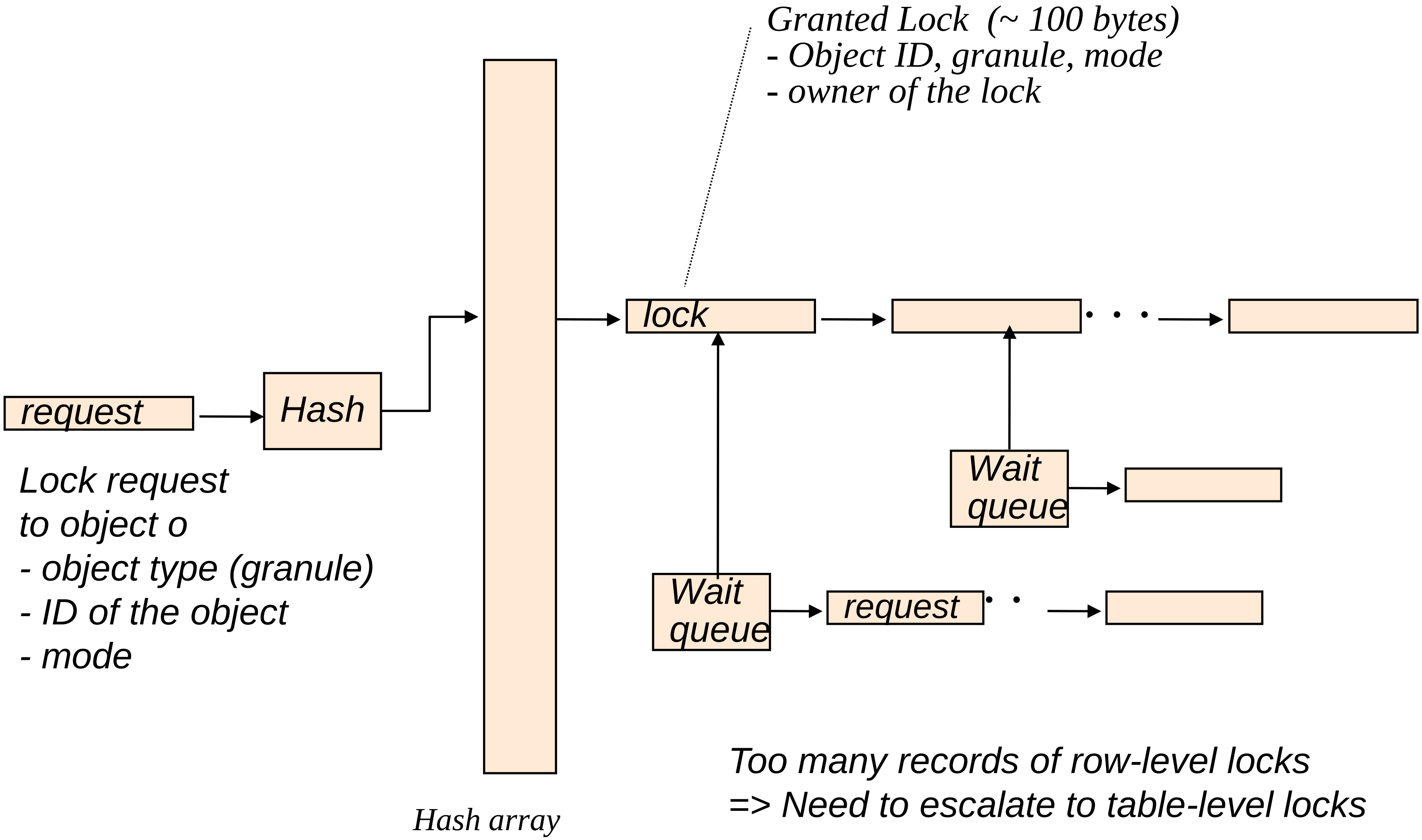
S-lock
on table 1)

X-lock
on table

Source:
IBM, Developerworks, Sanders
DB2 9 – Data concurrency

1) depending on the isolation level

Management of Lock Records and Requests



Lock request to object o

- *object type (granule)*
- *ID of the object*
- *mode*

Too many records of row-level locks
=> Need to escalate to table-level locks

Multi-Granular Locking (MGL) scheme

- Sample variants of lock compatibility matrices

Lock granules:

database

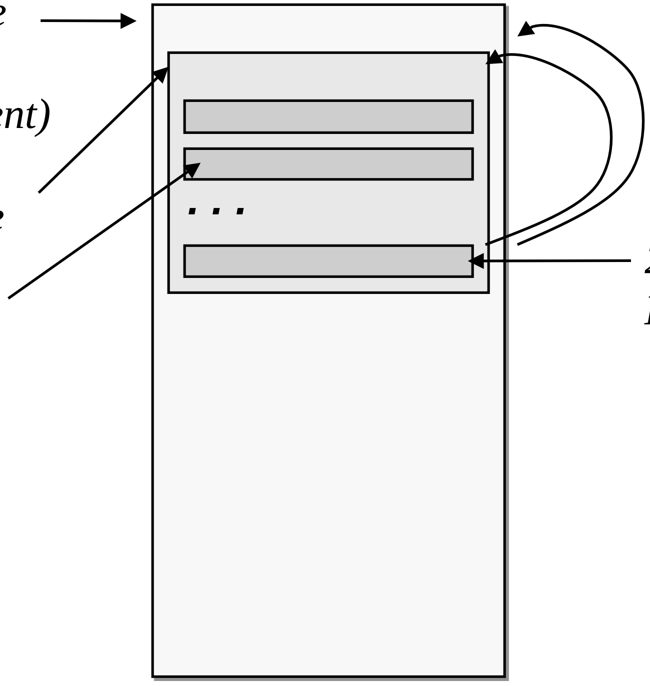
(tablespace)

table

(extent)

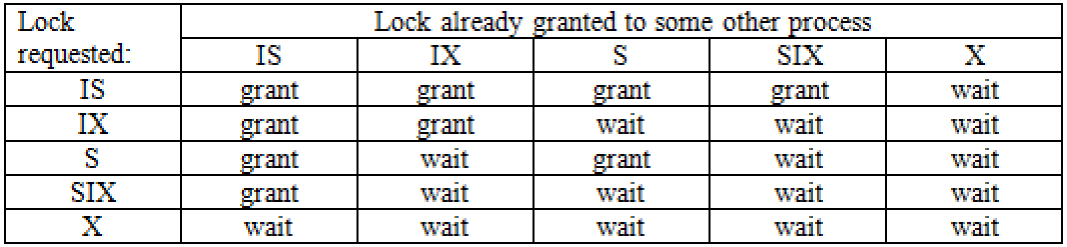
page

row



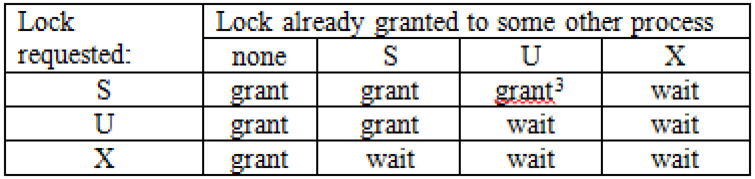
1. Intent locks
IS for S on row
IX for X on row

2.
Lock on row



Lock requested:	Lock already granted to some other process				
	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



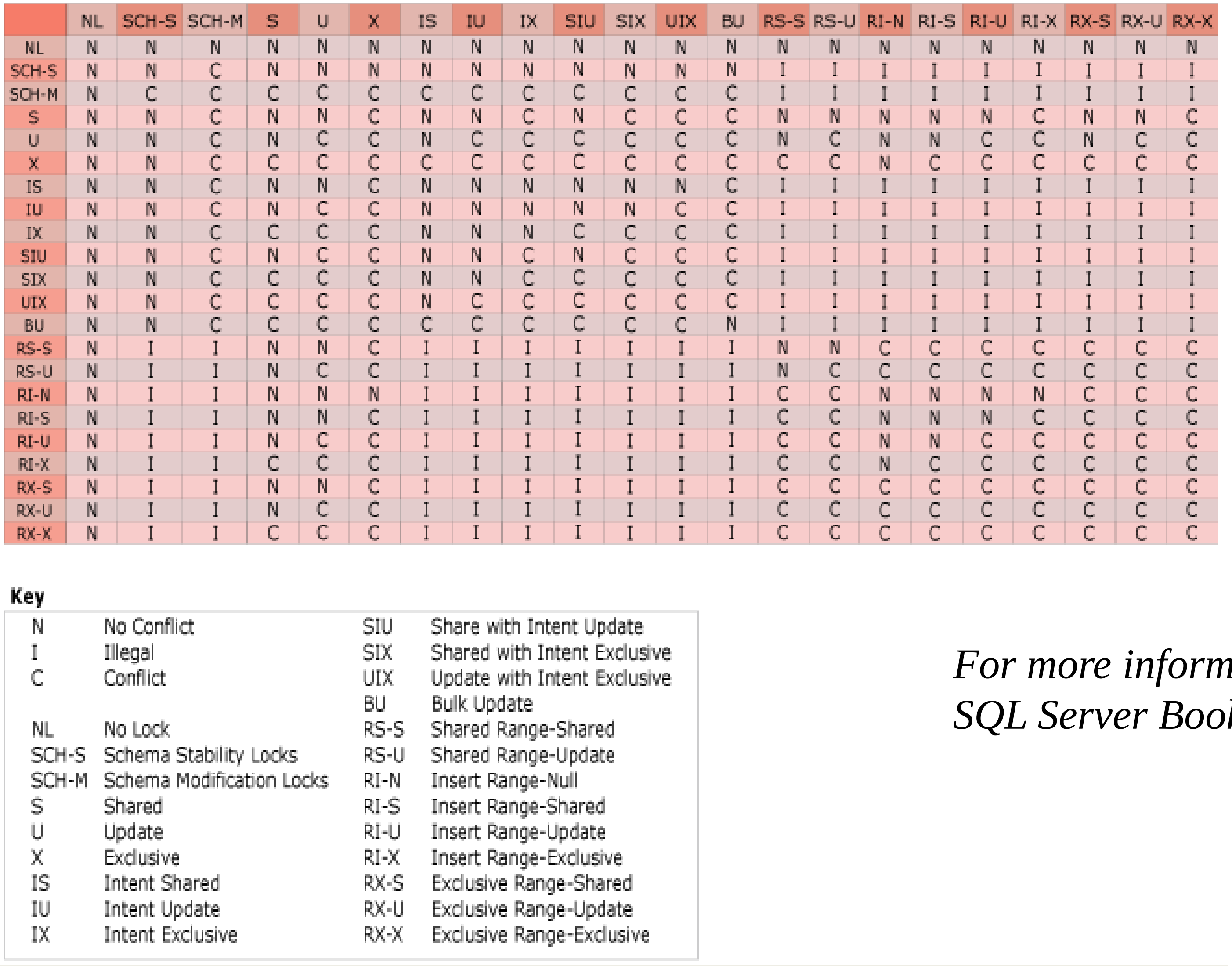
Lock requested:	Lock already granted to some other process			
	none	S	U	X
S	grant	grant	grant[3]	wait
U	grant	grant	wait	wait
X	grant	wait	wait	wait

__S__hared locks (S) allow reading.
e__X__clusive locks (X) allow writing and are kept up to end of transaction eliminating lost updates.

Other locks on index ranges, schemas

either GRANT or CNVT

Compatibility Matrix of SQL Server Locks



	NL	SCH-S	SCH-M	S	U	X	IS	IU	IX	SIU	SIX	UIX	BU	RS-S	RS-U	RI-N	RI-S	RI-U	RI-X	RX-S	RX-U	RX-X
NL	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SCH-S	N	N	C	N	N	N	N	N	N	N	N	N	N	I	I	I	I	I	I	I	I	I
SCH-M	N	C	C	C	C	C	C	C	C	C	C	C	C	I	I	I	I	I	I	I	I	I
S	N	N	C	N	N	C	N	N	C	N	C	C	C	N	N	N	N	N	C	N	N	C
U	N	N	C	N	C	C	N	C	C	C	C	C	C	N	C	N	N	C	C	N	C	C
X	N	N	C	C	C	C	C	C	C	C	C	C	C	C	C	N	C	C	C	C	C	C
IS	N	N	C	N	N	C	N	N	N	N	N	N	C	I	I	I	I	I	I	I	I	I
IU	N	N	C	N	C	C	N	N	N	N	N	C	C	I	I	I	I	I	I	I	I	I
IX	N	N	C	C	C	C	N	N	N	C	C	C	C	I	I	I	I	I	I	I	I	I
SIU	N	N	C	N	C	C	N	N	C	N	C	C	C	I	I	I	I	I	I	I	I	I
SIX	N	N	C	C	C	C	N	N	C	C	C	C	C	I	I	I	I	I	I	I	I	I
UIX	N	N	C	C	C	C	N	C	C	C	C	C	C	I	I	I	I	I	I	I	I	I
BU	N	N	C	C	C	C	C	C	C	C	C	C	N	I	I	I	I	I	I	I	I	I
RS-S	N	I	I	N	N	C	I	I	I	I	I	I	I	N	N	C	C	C	C	C	C	C
RS-U	N	I	I	N	C	C	I	I	I	I	I	I	I	N	C	C	C	C	C	C	C	C
RI-N	N	I	I	N	N	N	I	I	I	I	I	I	I	C	C	N	N	N	N	C	C	C
RI-S	N	I	I	N	N	C	I	I	I	I	I	I	I	C	C	N	N	N	C	C	C	C
RI-U	N	I	I	N	C	C	I	I	I	I	I	I	I	C	C	N	N	C	C	C	C	C
RI-X	N	I	I	C	C	C	I	I	I	I	I	I	I	C	C	N	C	C	C	C	C	C
RX-S	N	I	I	N	N	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C
RX-U	N	I	I	N	C	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C
RX-X	N	I	I	C	C	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C

Key

N	No Conflict	SIU	Share with Intent Update
I	Illegal	SIX	Shared with Intent Exclusive
C	Conflict	UIX	Update with Intent Exclusive
		BU	Bulk Update
NL	No Lock	RS-S	Shared Range-Shared
SCH-S	Schema Stability Locks	RS-U	Shared Range-Update
SCH-M	Schema Modification Locks	RI-N	Insert Range-Null
S	Shared	RI-S	Insert Range-Shared
U	Update	RI-U	Insert Range-Update
X	Exclusive	RI-X	Insert Range-Exclusive
IS	Intent Shared	RX-S	Exclusive Range-Shared
IU	Intent Update	RX-U	Exclusive Range-Update
IX	Intent Exclusive	RX-X	Exclusive Range-Exclusive

For more information see:
SQL Server Books Online

Multi-Version Concurrency Control (MVCC)

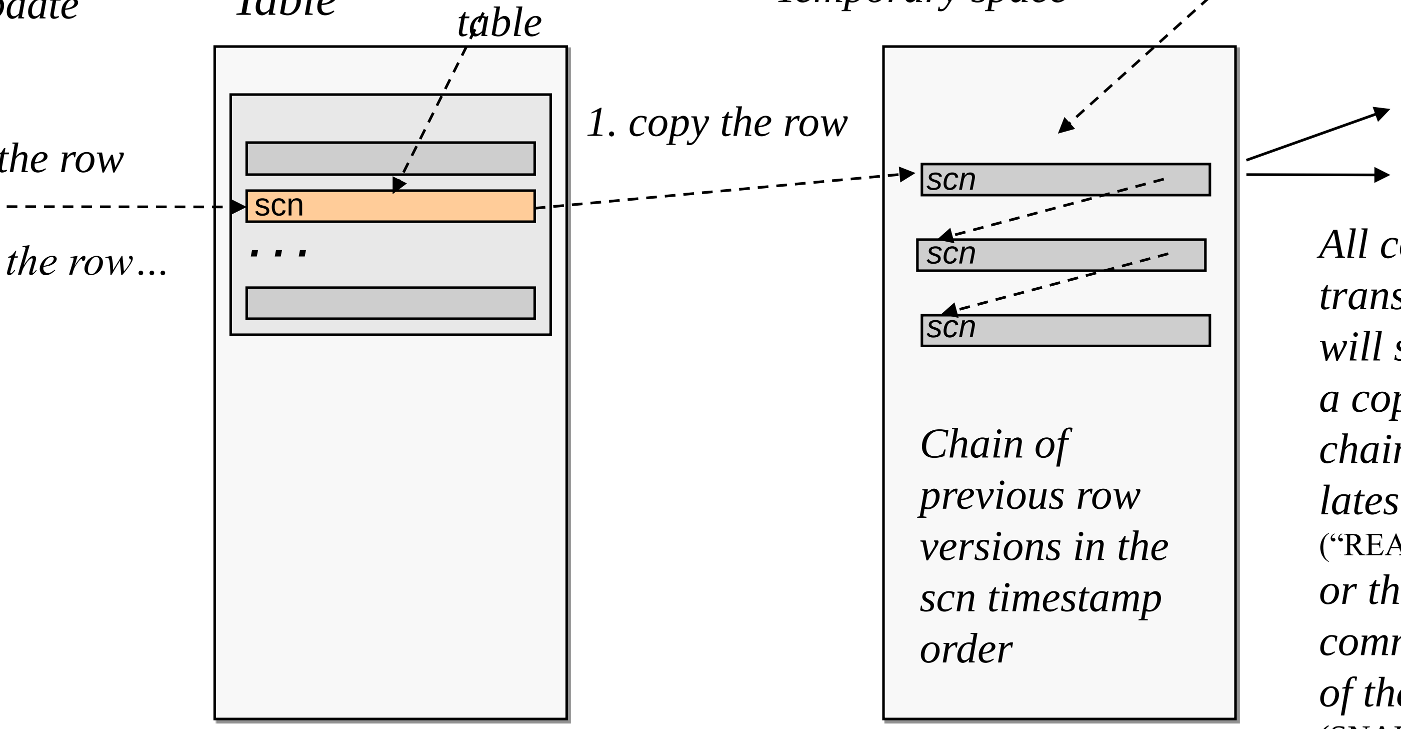
atomic update action:

2. "lock" the row

3. update the row…

. . .

Commit



Table

original row location on a page of the table

1. copy the row

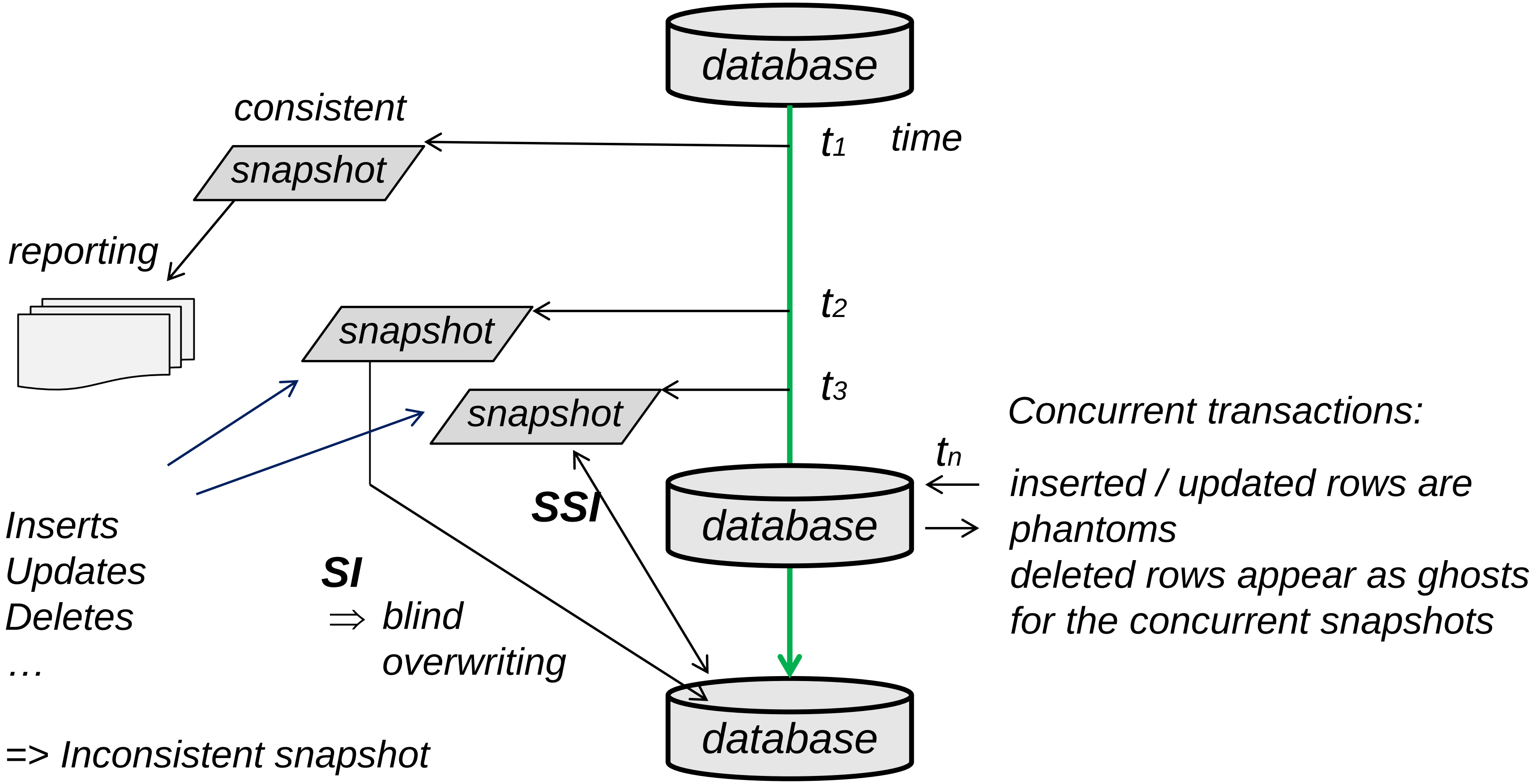
Temporary space

latest committed row version

Chain of previous row versions in the scn timestamp order

All concurrent transactions will see only a copy in the chain, either the latest committed ("READ COMMITTED") *or the latest committed at start of the transaction* (SNAPSHOT)

Phantoms & ghosts in snapshot isolation (SI)



Concurrent transactions:

inserted / updated rows are phantoms
deleted rows appear as ghosts
for the concurrent snapshots

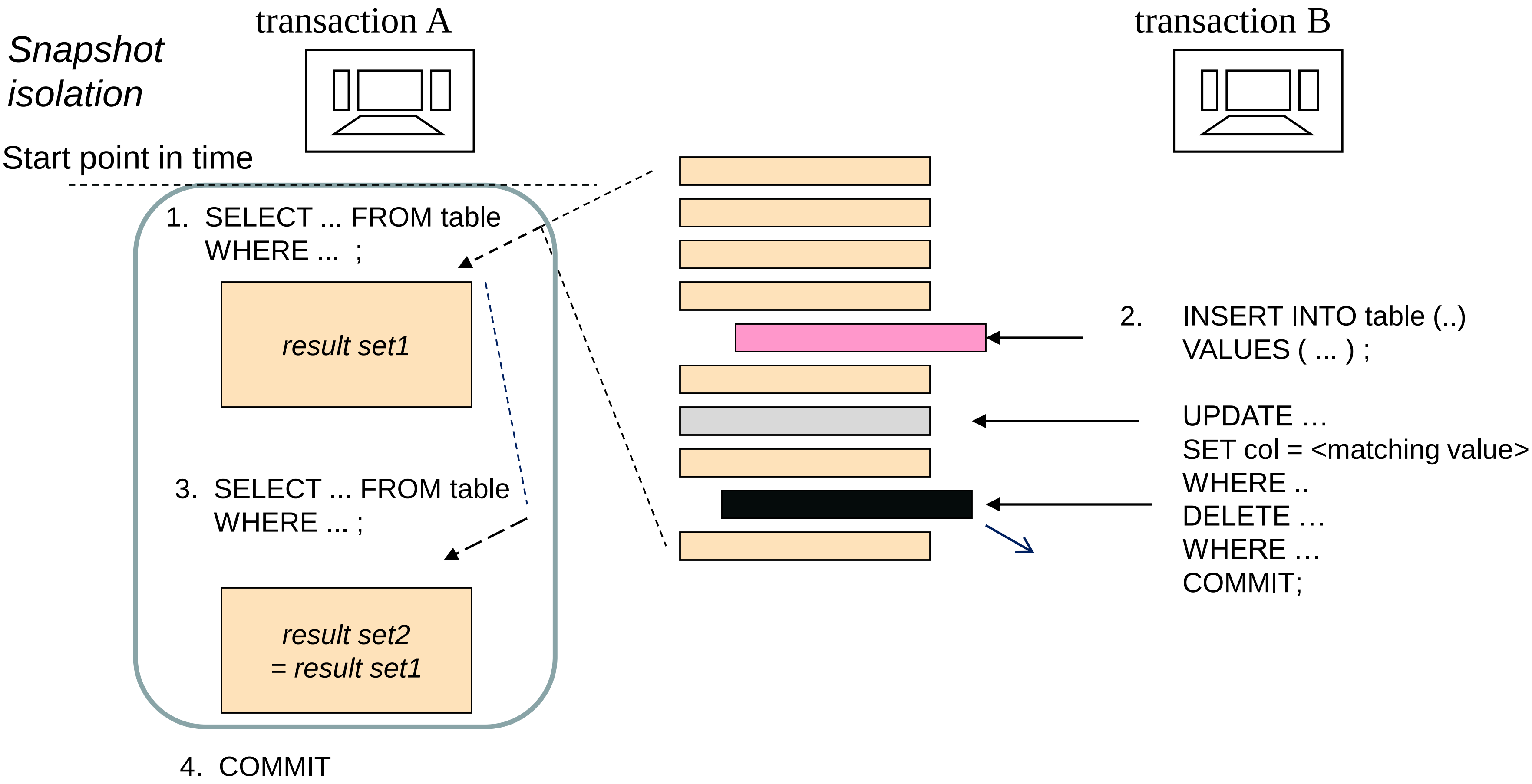
Inserts
Updates
Deletes
…

SI
⇒ *blind overwriting*

=> Inconsistent snapshot

SI = snapshot isolation
SSI = ”serializable” snapshot isolation (using version verification)

Snapshot (start point in time)



Snapshot isolation

Start point in time

transaction A

1. SELECT ... FROM table
   WHERE ... ;

result set1

3. SELECT ... FROM table
   WHERE ... ;

result set2
= result set1

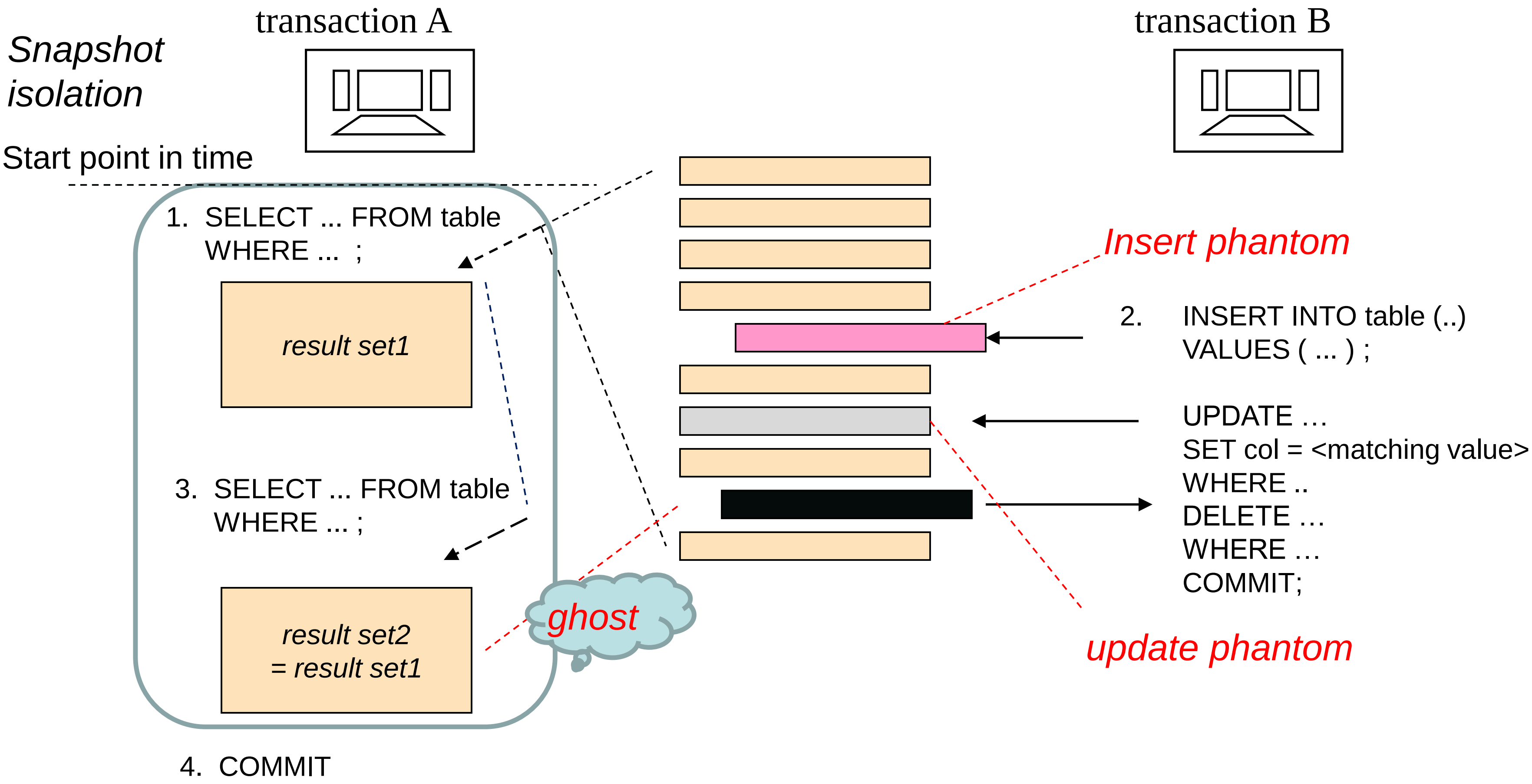
4. COMMIT

transaction B

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

   UPDATE …
   SET col = <matching value>
   WHERE ..
   DELETE …
   WHERE …
   COMMIT;

Phantoms and Ghosts of Snapshot



Snapshot isolation

Start point in time

transaction A

1. SELECT ... FROM table WHERE ... ;

result set1

3. SELECT ... FROM table WHERE ... ;

result set2 = result set1

ghost

4. COMMIT

transaction B

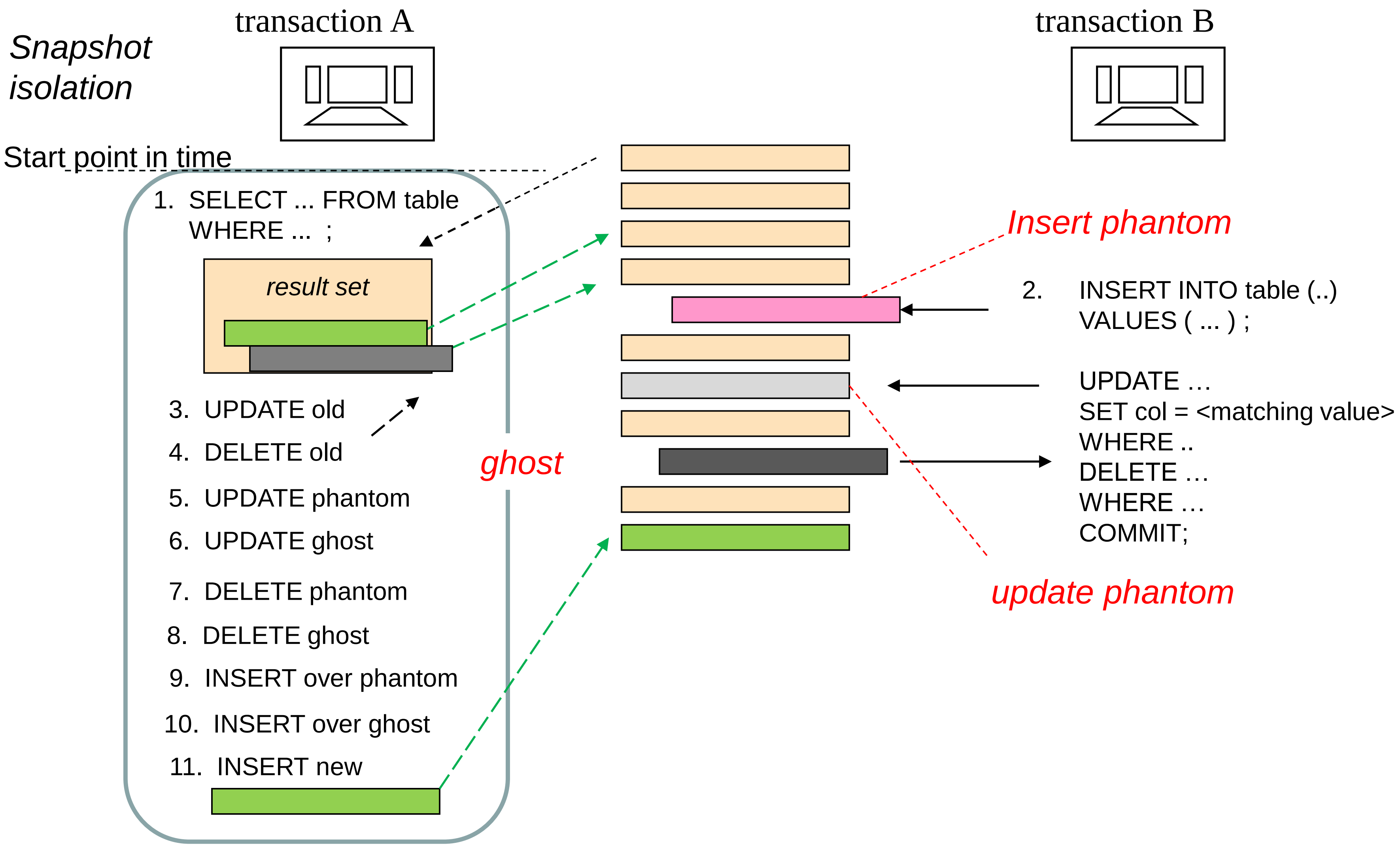
Insert phantom

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

UPDATE …
SET col = <matching value>
WHERE ..
DELETE …
WHERE …
COMMIT;

update phantom

Inconsistencies of Snapshot



Snapshot isolation

transaction A

transaction B

Start point in time

1. SELECT ... FROM table WHERE ... ;

result set

3. UPDATE old
4. DELETE old
5. UPDATE phantom
6. UPDATE ghost
7. DELETE phantom
8. DELETE ghost
9. INSERT over phantom
10. INSERT over ghost
11. INSERT new

ghost

Insert phantom

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

UPDATE …
SET col = <matching value>
WHERE ..
DELETE …
WHERE …
COMMIT;

update phantom

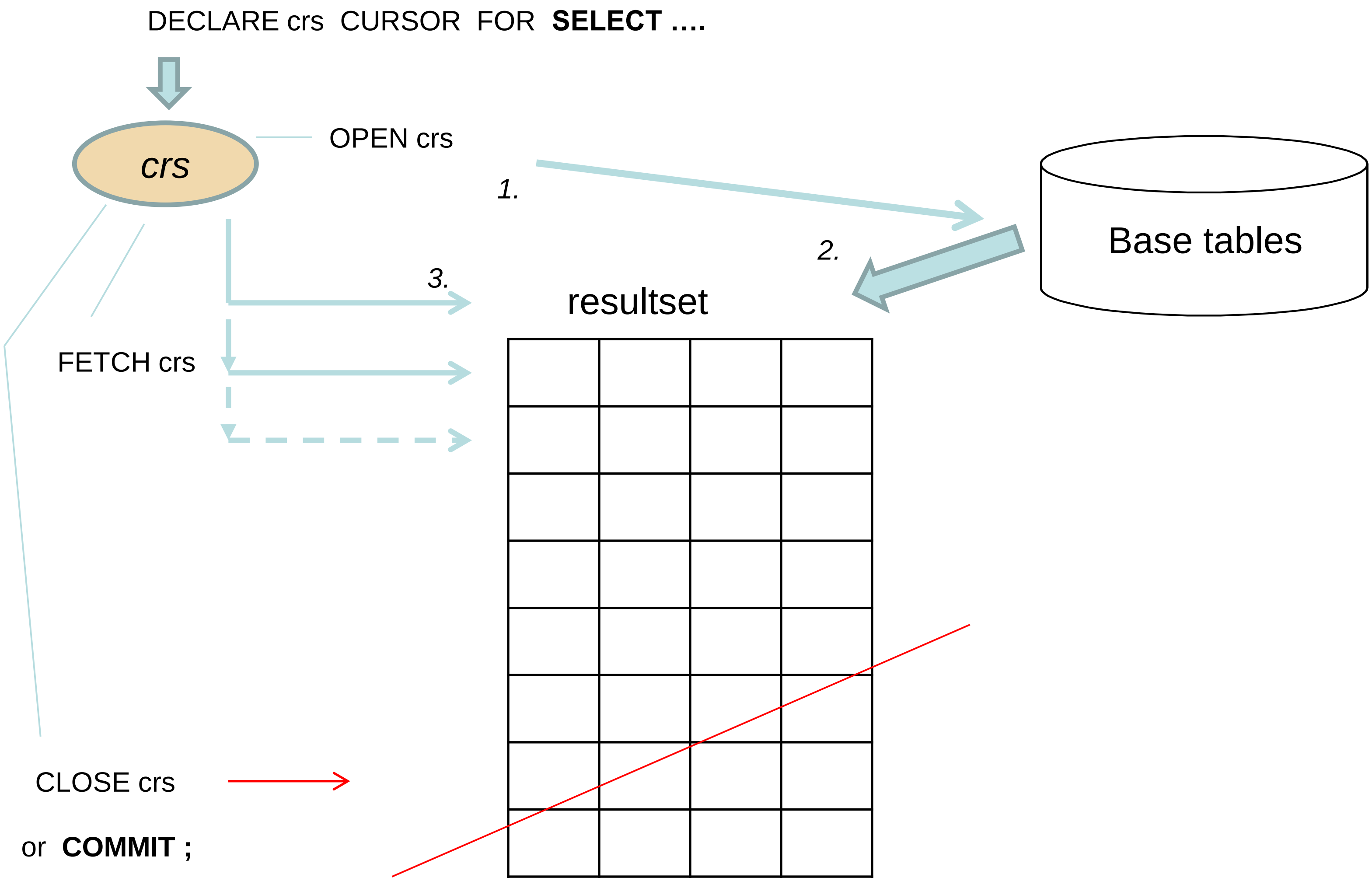
12. COMMIT

Cursor Processing

- Solves the paradigm mismatch between
  - Procedural Programming and
  - (”Relational”) SQL databases

- Scrolling / Forward only
- Sensitive / insensitive (snapshot)
- Server-side / client-side cache
- Optimistic concurrency
- Scope: transaction / (holdable) multiple transactions
- Options (hints)

..Cursor Processing



DECLARE crs CURSOR FOR **SELECT ....**

crs

OPEN crs

1.

Base tables

2.

3.

resultset

FETCH crs

CLOSE crs

or **COMMIT ;**

Multi-user Transaction Experiments

- Students start their private copies of DebianDB
- Teacher demonstrates the first steps
  making sure that all students can repeat every step
  getting started with the experiment
- The same DBMS product is selected to be studied,
  - for example MySQL/InnoDB
- Two concurrent SQL sessions are started in separate terminal windows
- Students make notes of the transaction experiments or experiences are discussed

Experiments on concurrency

- 2.2b
- 2.3
- 2.4
- 2.5
- 2.6
- 2.7

A Well-designed SQL Transaction

- Is an atomic, logical unit of work that either transfers the database from a consistent state to another consistent state – or all its actions need to be rolled back
- Is a short dialogue with the database server performing data retrieval and/or data update task of some use case
- Does not contain any user intervention during the transaction
- Checks carefully diagnostics of the received data access services
- Handles the generated data access exceptions
- May contain transaction logic which depends on the received data or diagnostics
- Is restarted on concurrency or connection failures but avoiding livelocks