Transactions interaction and best practices for application development

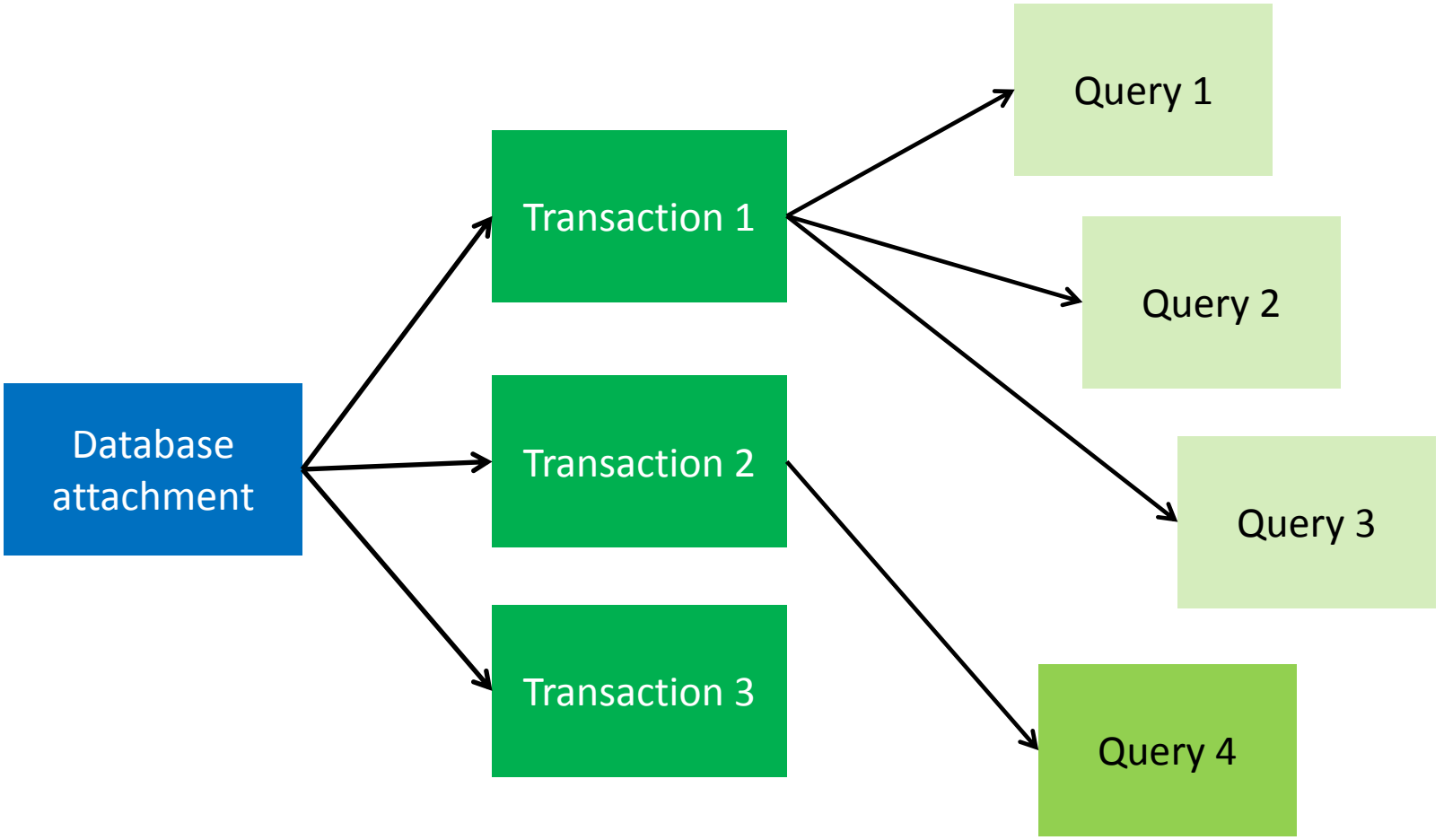
Firebird Conference 2019

Berlin, 17-19 October

IBSurgeon

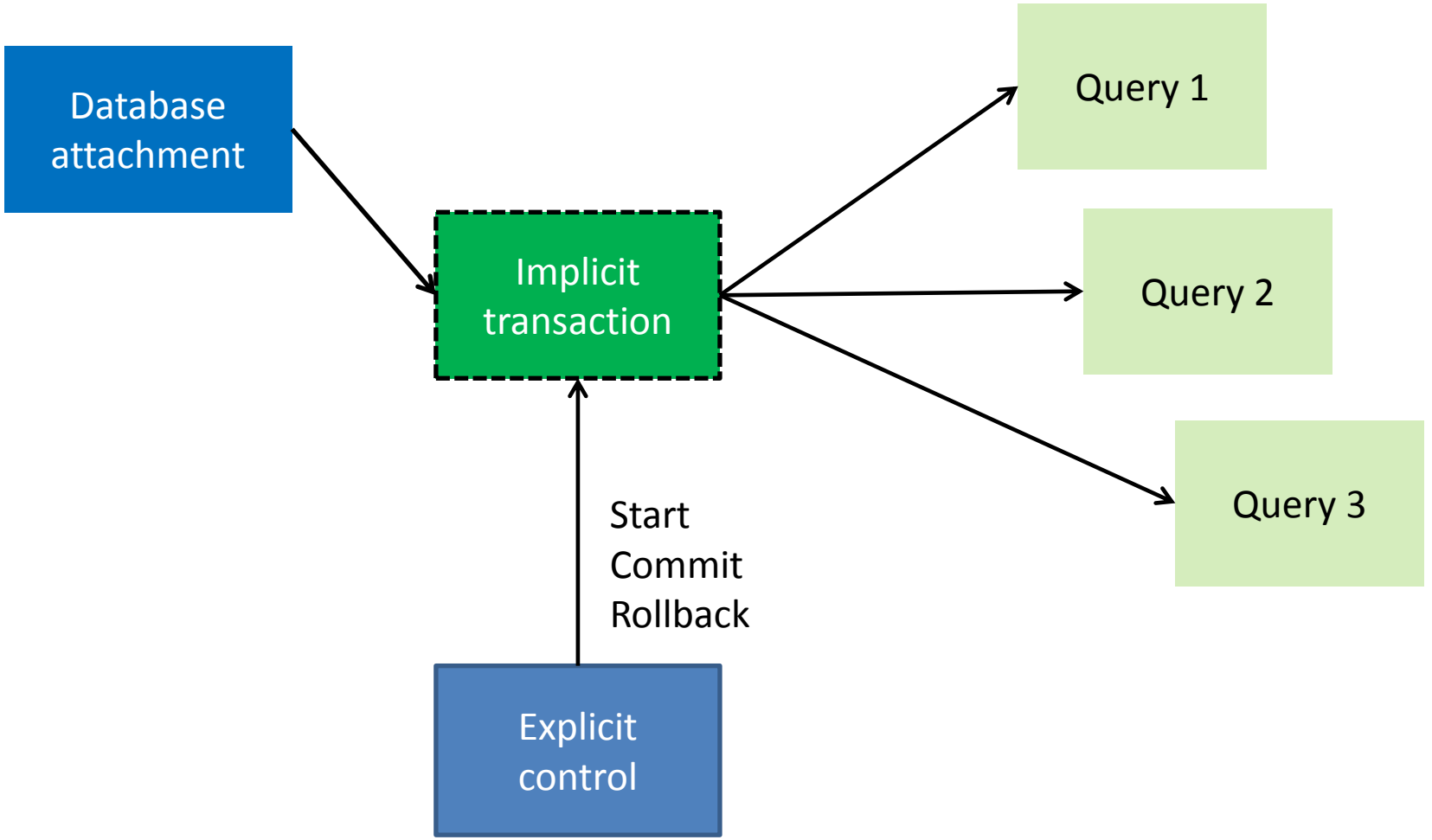
TRANSACTIONS LIFETIME IN VARIOUS DRIVERS

InterBase and Firebird API



Handles!

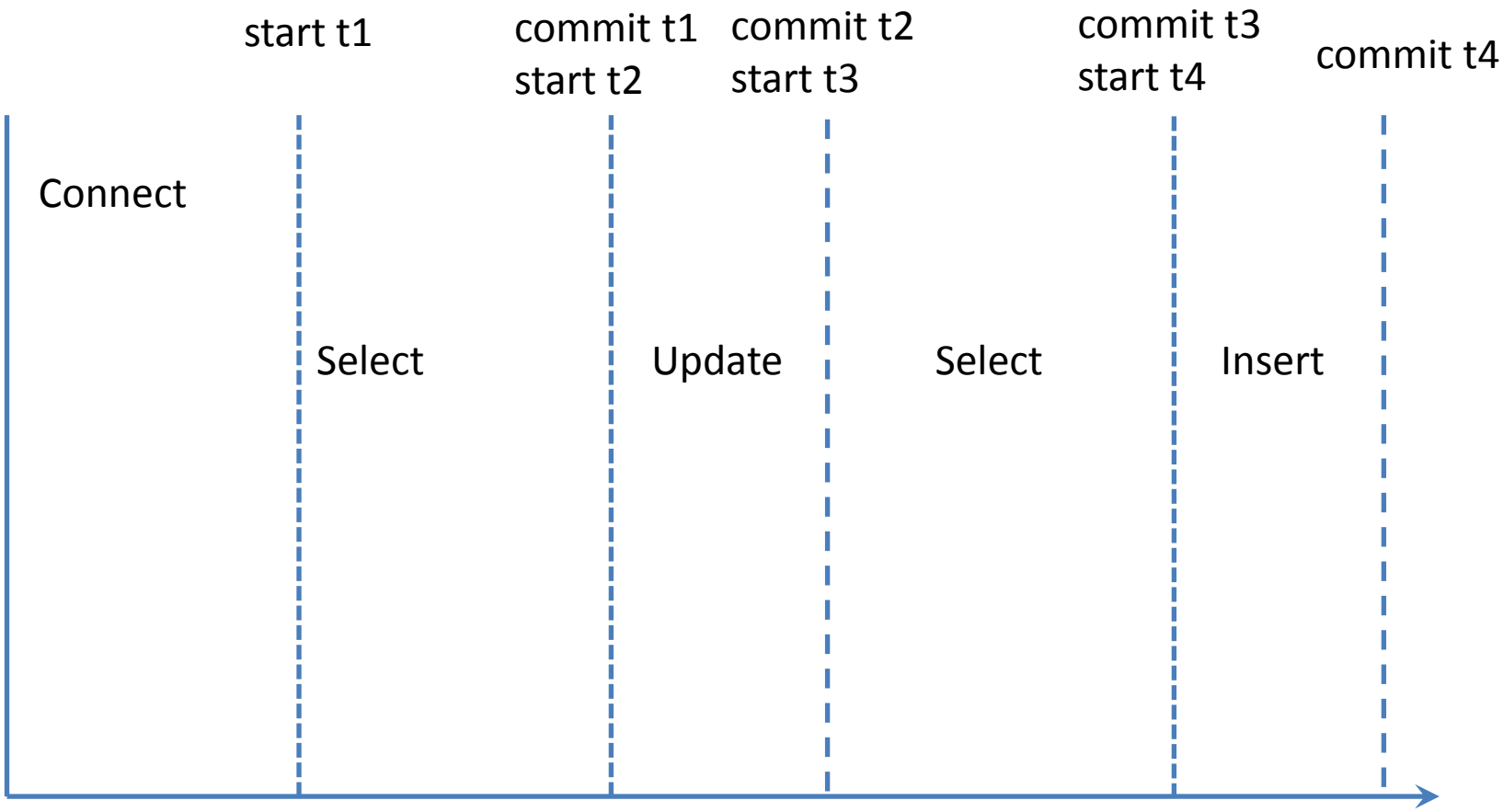
Most standard drivers



Implicit transactions

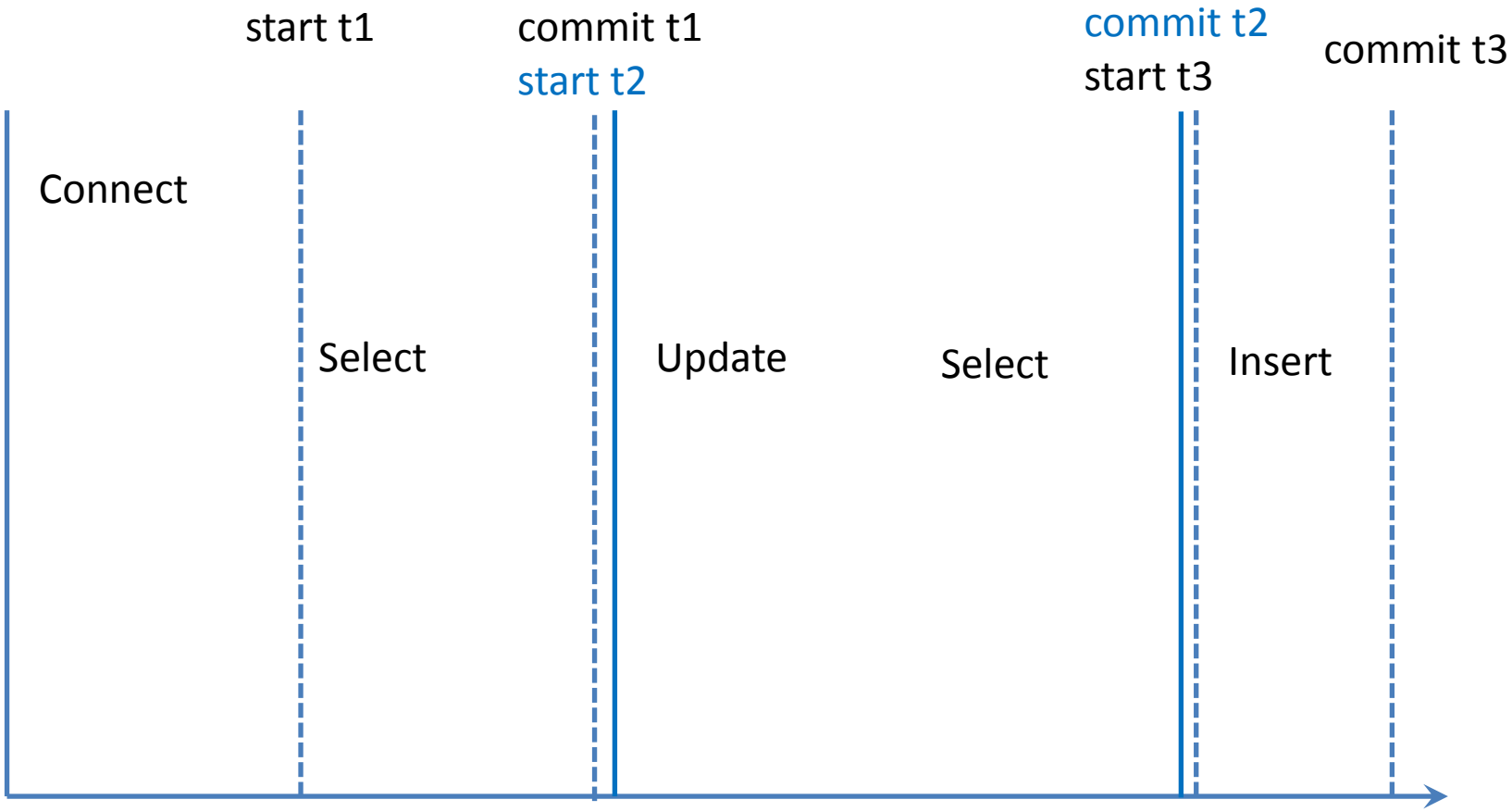
- Hidden. You do not see them, you do not control them
- Any SQL statement causes transaction start
- Autocommit mode
  - Each successful INSERT, UPDATE, DELETE and Execute Procedure causes automatic Commit. Any error causes Rollback.
  - SELECT statements may not be committed, until Insert, Update, Delete or Execute procedure.

- You cannot define set of SQL DML statements as a real transaction
- Transaction with Select statements can run forever
- Transaction may be ended by Retaining
- Connect may start transaction immediately


start t1
commit t1
start t2
commit t2
start t3
commit t3
start t4
commit t4
Connect
Select
Update
Select
Insert

Explicit transactions

- You may (or not) call StartTransaction
- All SQL statements will execute in that transaction
- You must end transaction with explicit Commit or Rollback


start t1
commit t1
start t2
commit t2
start t3
commit t3
Connect
Select
Update
Select
Insert

One transaction per connect

- Common driver architecture
  - BDE
  - ODBC
  - JDBC
  - DBExpress
  - .Net driver
  - …
- Implicit transactions by default
- You may start explicit transactions
- Transactions can live (be active) for a very long time

BDE

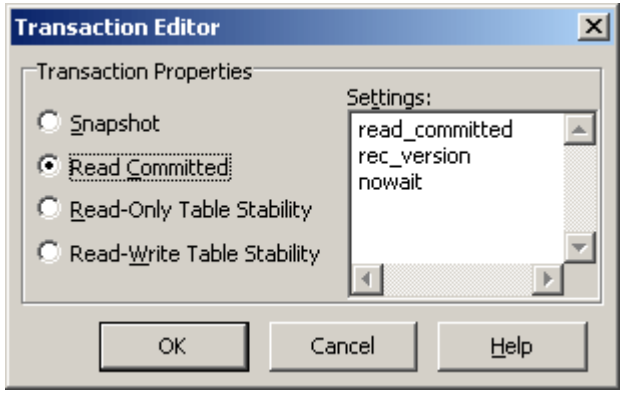
- DataSet.Open;
- Query.ExecSQL;
- Database.StartTransaction
  - Query1.ExecSQL;
  - DataSet1.Open;
- Database.Commit;

Many transactions per connect

- IBX, FIBPlus, FireDAC (AnyDAC), UIB, …

- Total control on transaction parameters
- Many transactions per connect
- Ability to use DataSets for read in one transaction, and write in another

IBX



- Transaction1.StartTransaction;
- IBQuery1.ExecSQL;

- Transaction2.StartTransaction;
- IBQuery2.ExecSQL;

- Transaction1.Commit;
- Transaction2.Rollback;

Long reading transactions

- from InterBase 6.0
- read
  nowait
  read_committed
  rec_version

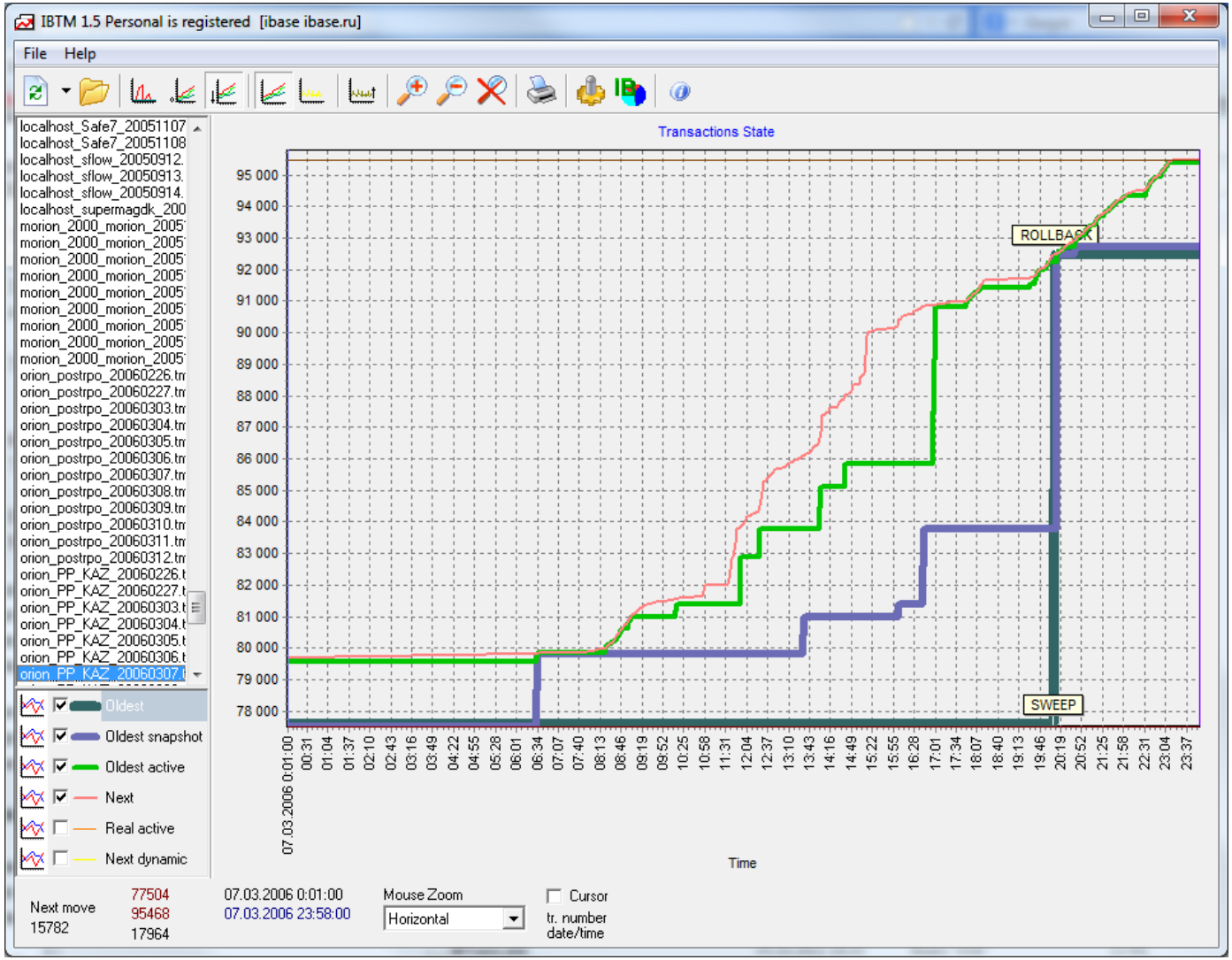
- This can be running forever

dbExpress – handles?

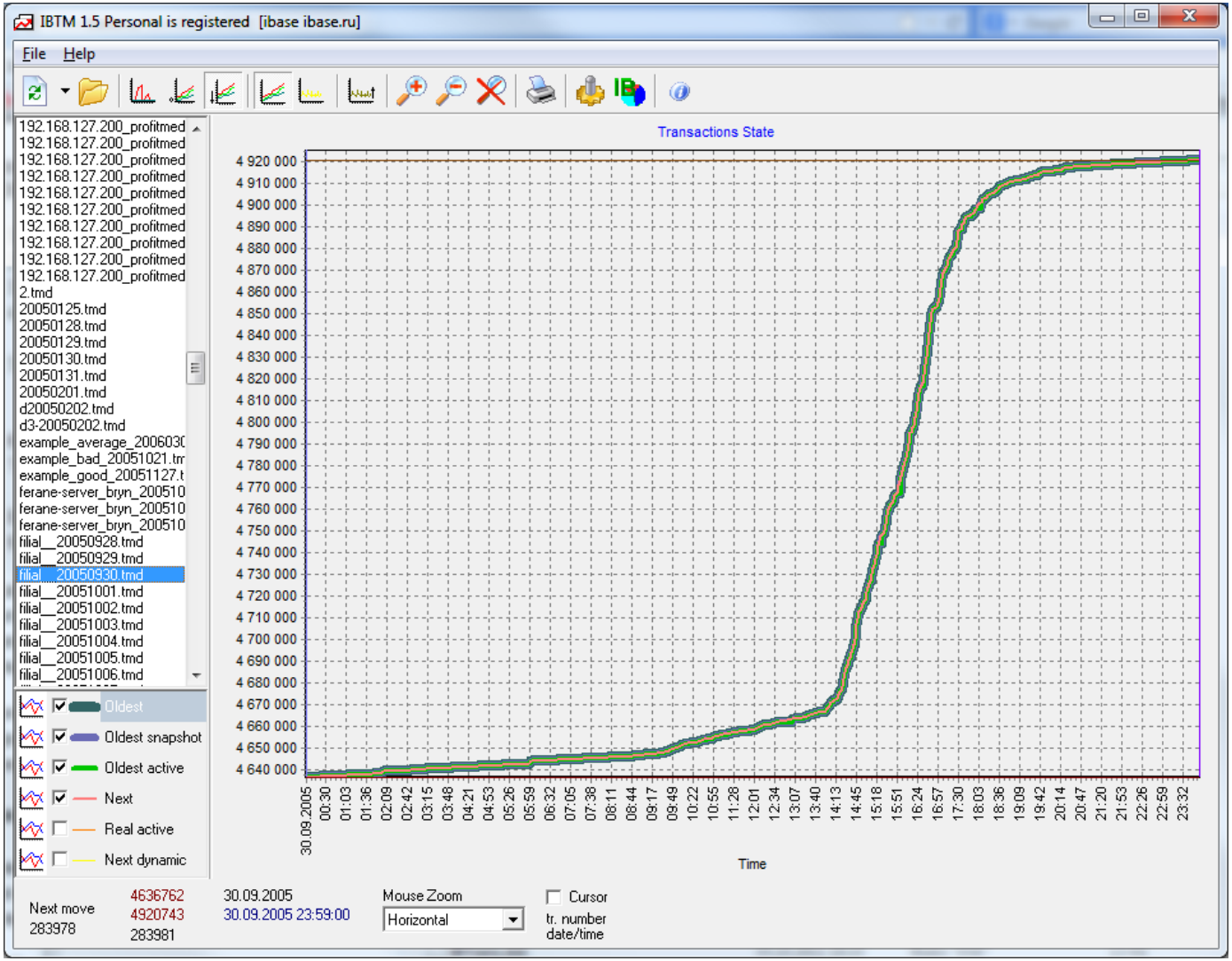
- Cannot set parameters
- TDBXTransaction exists, but useless
- Cannot switch between transactions

- transaction1.BeginTransaction;
- …
- transaction2.BeginTransaction;
- … here you can not return to transaction1 context, you can only call it’s commit/rollback.

Bad transaction control



Perfect transaction control



Rules for native components/drivers

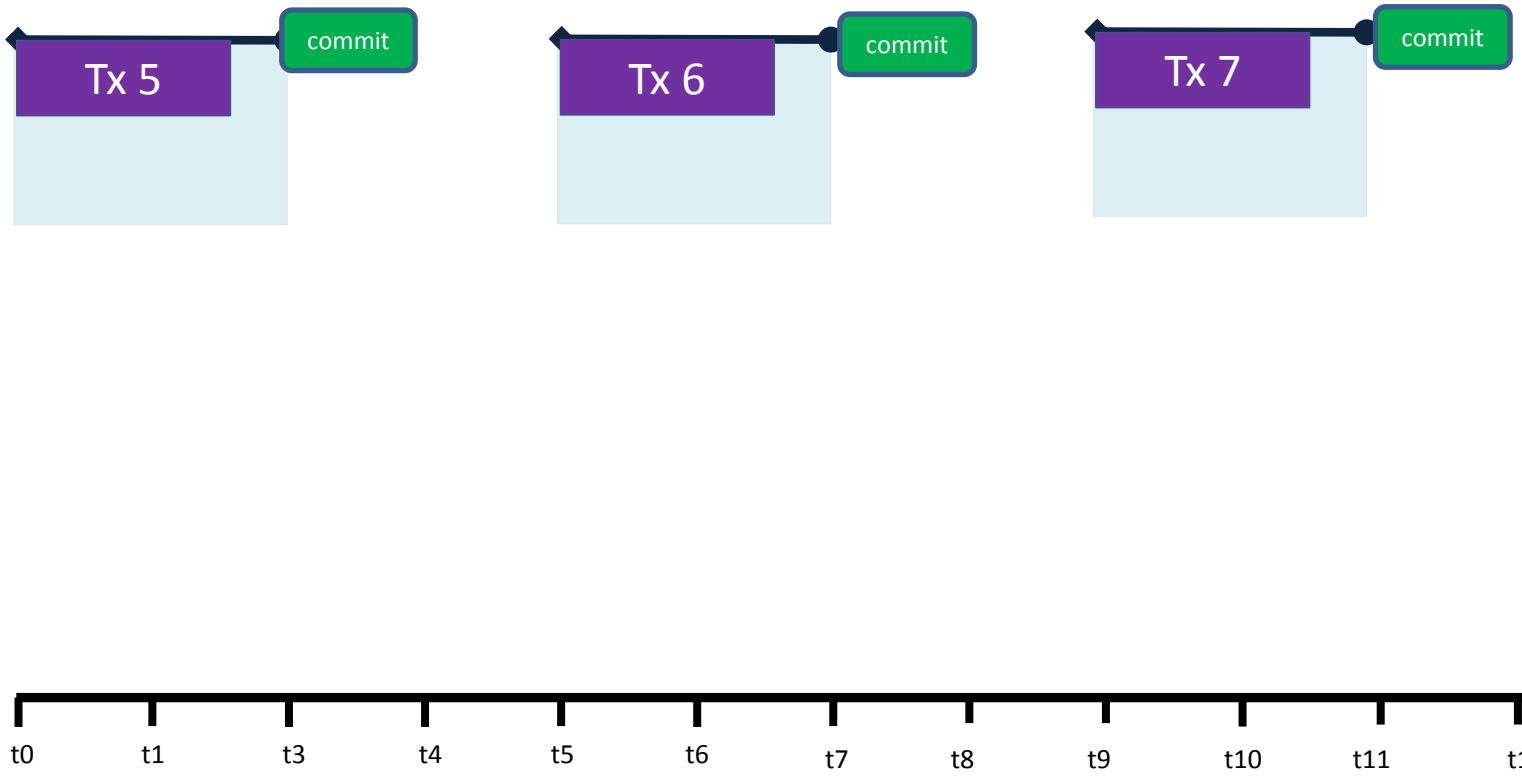
- Do not use “default” transaction. Always use explicit transaction control.
- Do not use “default transaction parameters”
  - Default may be ReadCommitted or Snapshot, you will never know
- Do not allow to live transactions for a long time.
  - Keep transactions short
  - Use read read_committed for long reading
- Do not use retaining (CommitRetaining, RollbackRetaining)
- Even if there AutoCommit option, check it not to use Retaining mode.

Rules for ‘single transactional’

- Do not use these drivers/components ☺
- All you can do, is from time to time call
  - Database.StartTransaction;
  - Database.Commit;
- to end default transaction lifecycle

TRANSACTIONS INTERACTION TYPES

Sequential transactions

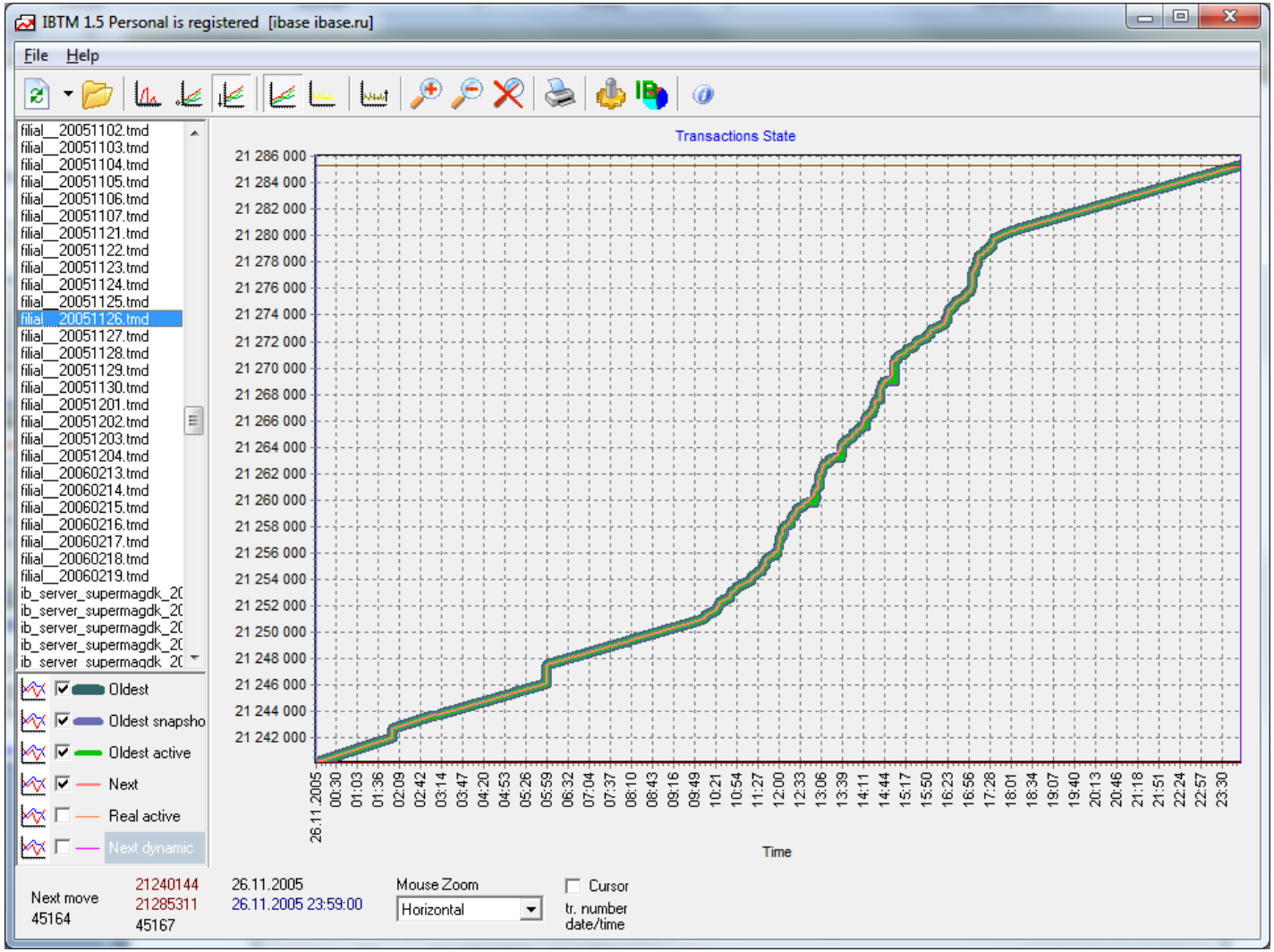


Gstat -h

```
Database header page information:
        Flags                   0
        Checksum                12345
        Generation              112431494
        Page size               8192
        ODS version   11.1
        Oldest transaction      100         x-1
        Oldest active           101         x
        Oldest snapshot         101         x
        Next transaction        102         x+1
        Bumped transaction      1
        Sequence number         0
        Next attachment ID      0
        Implementation ID       16
        Shadow count            0
        Page buffers   256
        Next header page        0
        Database dialect        1
        Creation dateJun 5, 2011 10:02:19
        Attributes              force write

    Variable header data:
        Sweep interval:         20000
        *END*
```

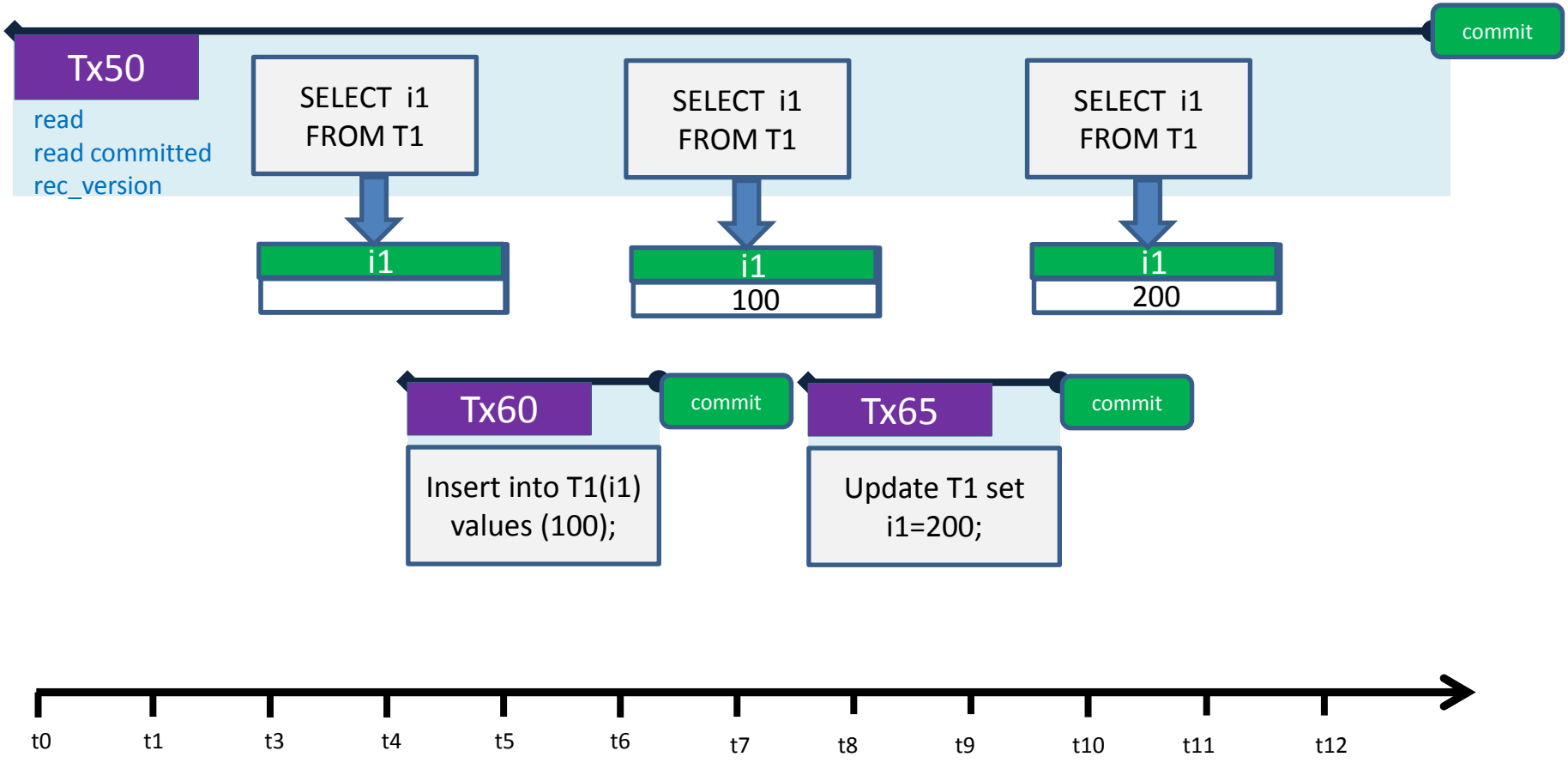
Ideal transaction control



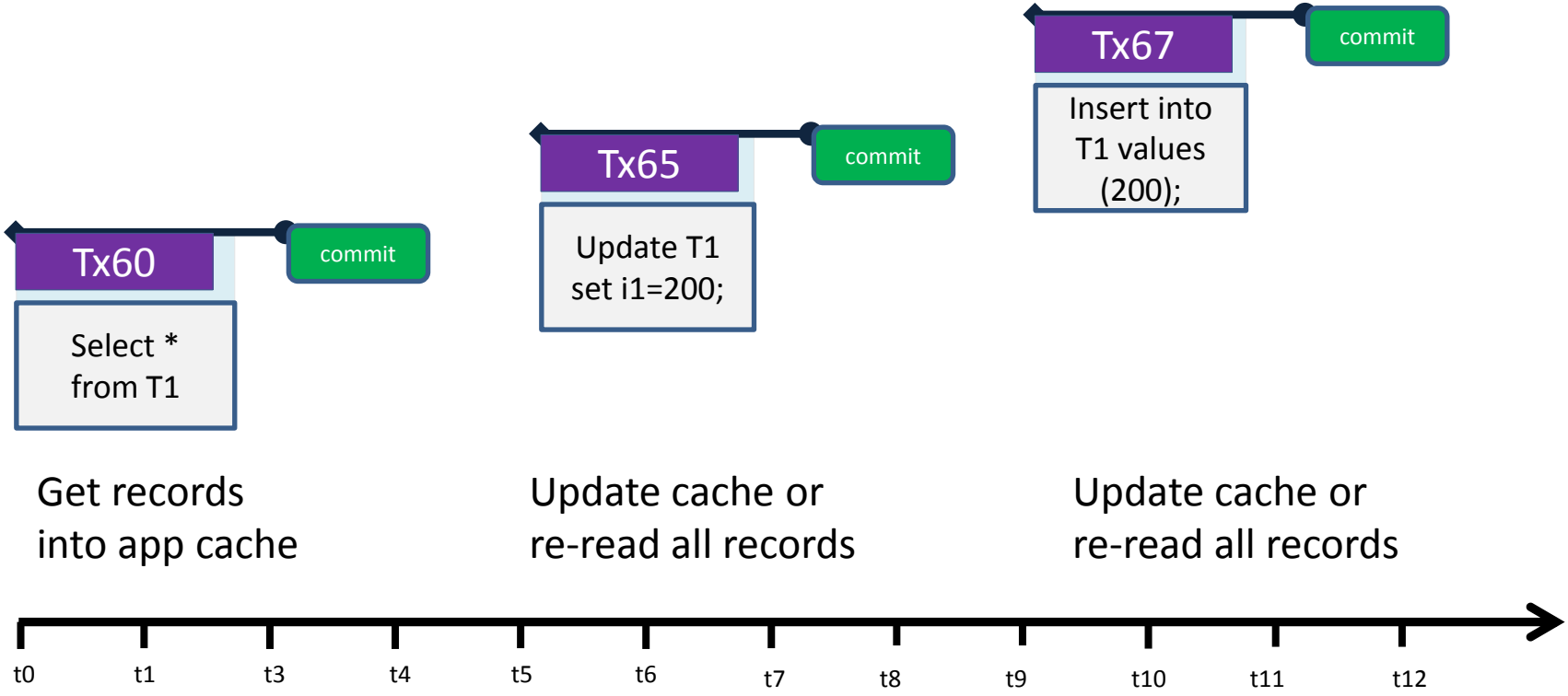
Two ways to almost ideal transaction control

1. Long read-only read-committed and short write
2. Short read and write

Long read-only RC and short write



Short read and write



Pro & Contra

Long read-only RC and short writes

- + easy to implement read and update logic
- - requires support from drivers/components (2 transactions or 2 connections)
- + more convenient for client-server
- - less convenient for multi-tier and stateless applications

Short read and writes

- - hard to implement sophisticated caching
- + works with any data access drivers/components
- - less convenient for client-server
- + more convenient for multi-tier and stateless applications

HOW TO IMPLEMENT EDIT DIALOGS IN AN EFFECTIVE AND SAFE WAY

Data editing

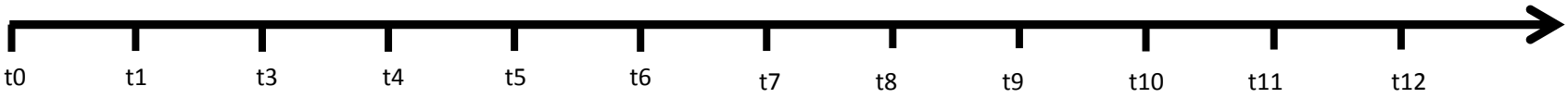
- Application is used by operator not in the way developer designed it
- Badly designed data editing can be a problem

Data editing: wrong scenario

Open form
StartTransaction;
IBDataSet.Open;
IBDataSet.Edit;
// entering edit mode

Wait for user input

on Save button
Commit;
on Cancel button
Rollback;



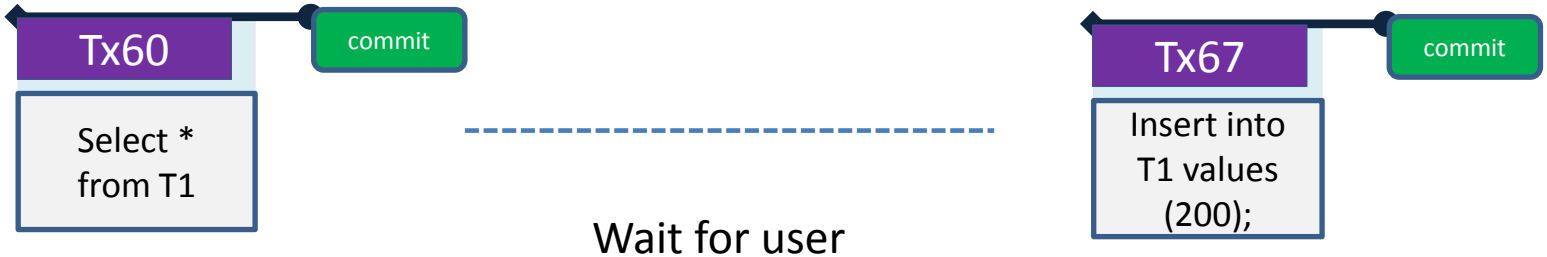

IBTM 1.5 Personal is registered [ibase ibase.ru]
File Help
192.168.0.105_forward_zakup_filial__20051026.tmd
morion_2000_morion_2005111
orion_PP_KAZ_20060226.tmd
orion_PP_KAZ_20060307.tmd
Transactions State
Oldest
Oldest snapshot
Oldest active
Next
Real active
Next dynamic
What is that?
Lunch time!
Time
Next move
71279
4152
798703
794551
10.11.2005 12:58:03
10.11.2005 14:39:51
Mouse Zoom
Horizontal
Cursor
tr. number
date/time

Data editing: Solution

- Open form
- StartTransaction;
- Fill controls
- Commit;
- Wait for user

- User presses Save button:
- StartTransaction;
  - IBDataSet.Edit; or IBQuery1.Prepare
- Fill data from controls
  - IBDataSet.Post; or IBQuery1.ExecSQL;
- Commit;

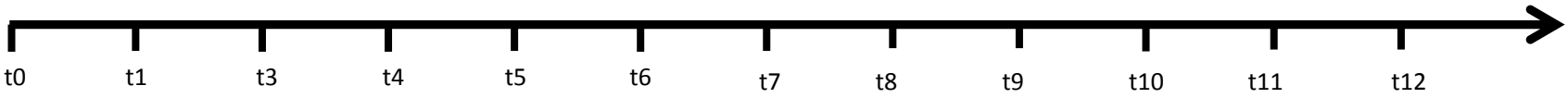
Data editing: Solution



Open form
StartTransaction;
Fill controls
Commit;

on Save button:

StartTransaction;
IBDataSet.Edit; or IBQuery1.Prepare
Fill data from controls
IBDataSet.Post; or IBQuery1.ExecSQL;
Commit;



Retaining transaction context

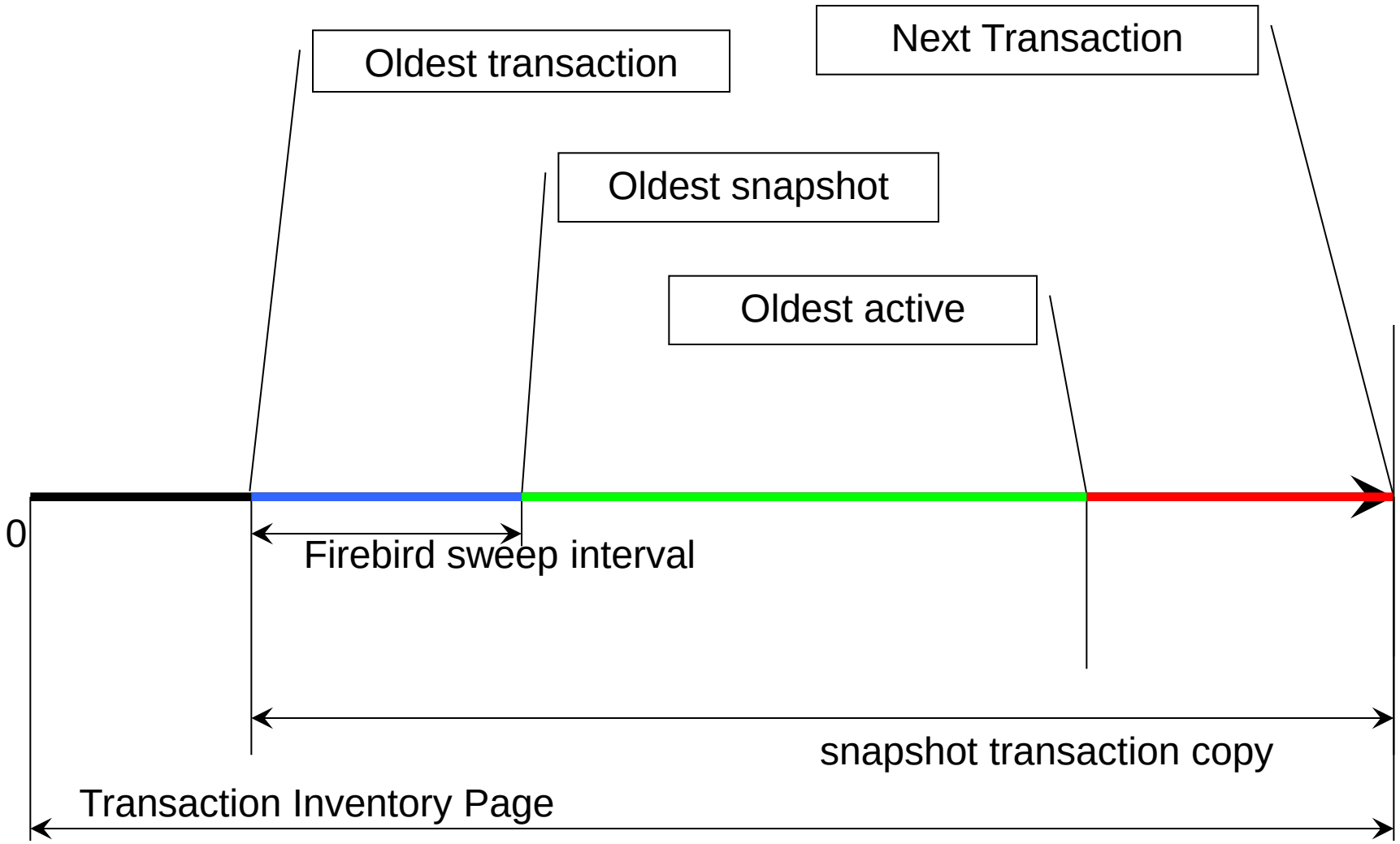
- Retaining ends transaction and starts a new one
  - Old transaction is marked in TIP as committed\rolled back
  - New transaction keeps context of old transaction
  - **Old snapshot is preserved, i.e. new transaction have the same OAT value as the old one**
  - New transaction will see changes of the old one as committed

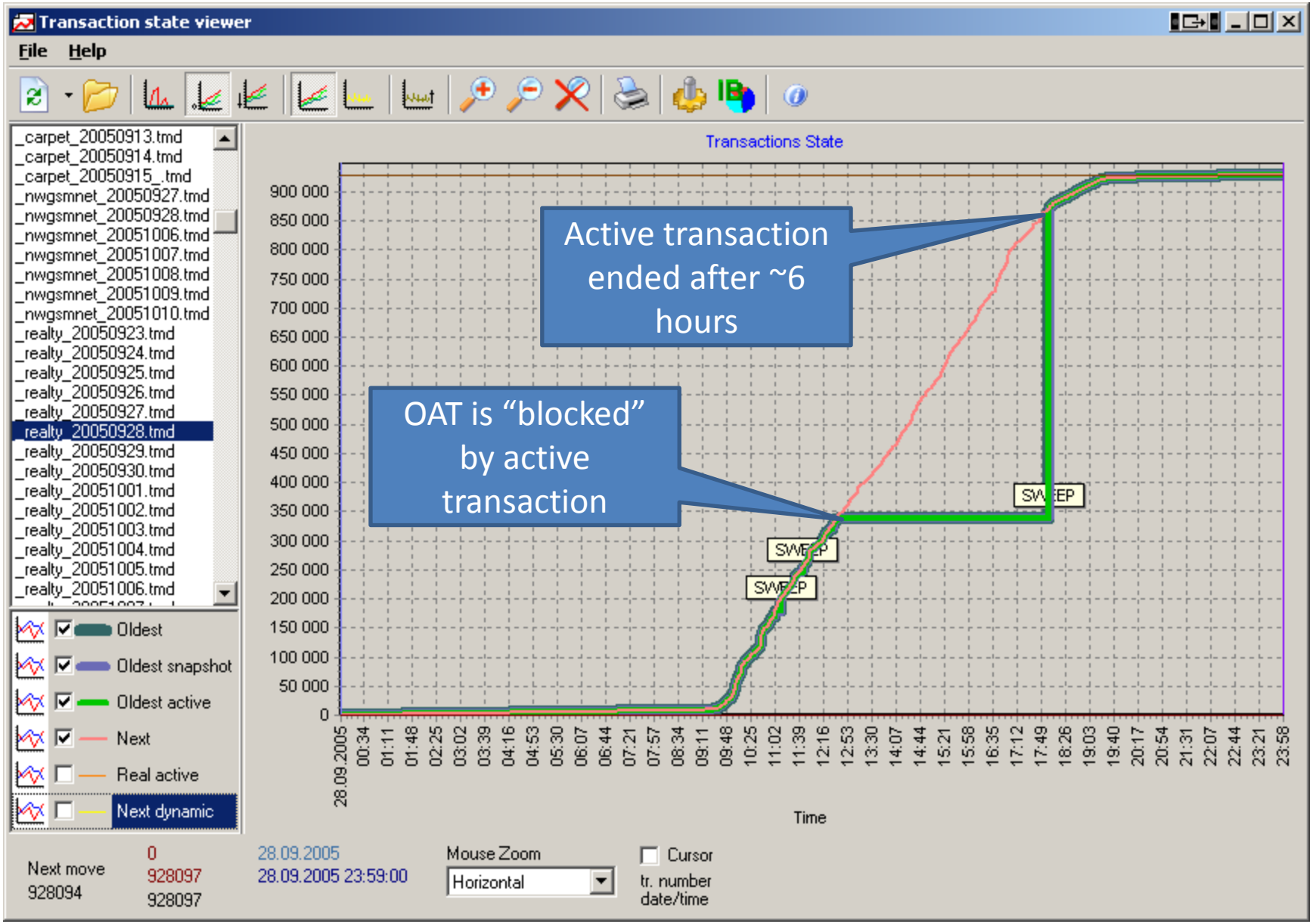
Hard commit\rollback vs retaining

- Pluses
  - One network roundtrip instead of two
  - Client recordsets survive transaction end
- Minuses
  - Open cursors are not closed
  - Temporary blobs are not released
  - Metadata locks are not released

WHEN GARBAGE COLLECTION DOES NOT WORK?

TIP markers


Oldest transaction
Next Transaction
Oldest snapshot
Oldest active
0
Firebird sweep interval
snapshot transaction copy
Transaction Inventory Page


Transaction state viewer
File Help
_carpet_20050913.tmd
_carpet_20050914.tmd
_carpet_20050915_.tmd
_nwgsmnet_20050927.tmd
_nwgsmnet_20050928.tmd
_nwgsmnet_20051006.tmd
_nwgsmnet_20051007.tmd
_nwgsmnet_20051008.tmd
_nwgsmnet_20051009.tmd
_nwgsmnet_20051010.tmd
_realty_20050923.tmd
_realty_20050924.tmd
_realty_20050925.tmd
_realty_20050926.tmd
_realty_20050927.tmd
realty_20050928.tmd
_realty_20050929.tmd
_realty_20050930.tmd
_realty_20051001.tmd
_realty_20051002.tmd
_realty_20051003.tmd
_realty_20051004.tmd
_realty_20051005.tmd
_realty_20051006.tmd
Oldest
Oldest snapshot
Oldest active
Next
Real active
Next dynamic
Transactions State
900 000
850 000
800 000
750 000
700 000
650 000
600 000
550 000
500 000
450 000
400 000
350 000
300 000
250 000
200 000
150 000
100 000
50 000
0
28.09.2005
00:34
01:11
01:48
02:25
03:02
03:39
04:16
04:53
05:30
06:07
06:44
07:21
07:57
08:34
09:11
09:48
10:25
11:02
11:39
12:16
12:53
13:30
14:07
14:44
15:21
15:58
16:35
17:12
17:49
18:26
19:03
19:40
20:17
20:54
21:31
22:07
22:44
23:21
23:58
Time
Active transaction ended after ~6 hours
OAT is “blocked” by active transaction
SWEEP
SWEEP
SWEEP
Next move
928094
0
928097
928097
28.09.2005
28.09.2005 23:59:00
Mouse Zoom
Horizontal
Cursor
tr. number
date/time

HOW TO IDENTIFY SWEEP

In the firebird.log !

- SRV-250 Mon May 18 21:00:01 2015
  Sweep is started by SYSDBA
  Database “----"
  OIT 25963894, OAT 26340734, OST 26340734, Next 27458805

- SRV-250 Mon May 18 21:46:25 2015
  Sweep is finished
  Database “----“
  OIT 26340733, OAT 26340734, OST 26340734, Next 27499132

What sweep could do

- Sweeping took 46 minutes (356gb database)
- OIT moved up by 376 839
- OST went up by 0
- OAT went up by 0
- Next went up by 40 327
  - 77k transactions per hour
- Next-OAT = 1 158 398
  - div 77k = oat stuck ~14 hours ago

HOW TO IDENTIFY PROBLEMATIC TRANSACTIONS (TOO LONG, WRONG ISOLATION LEVEL) WITH MON$

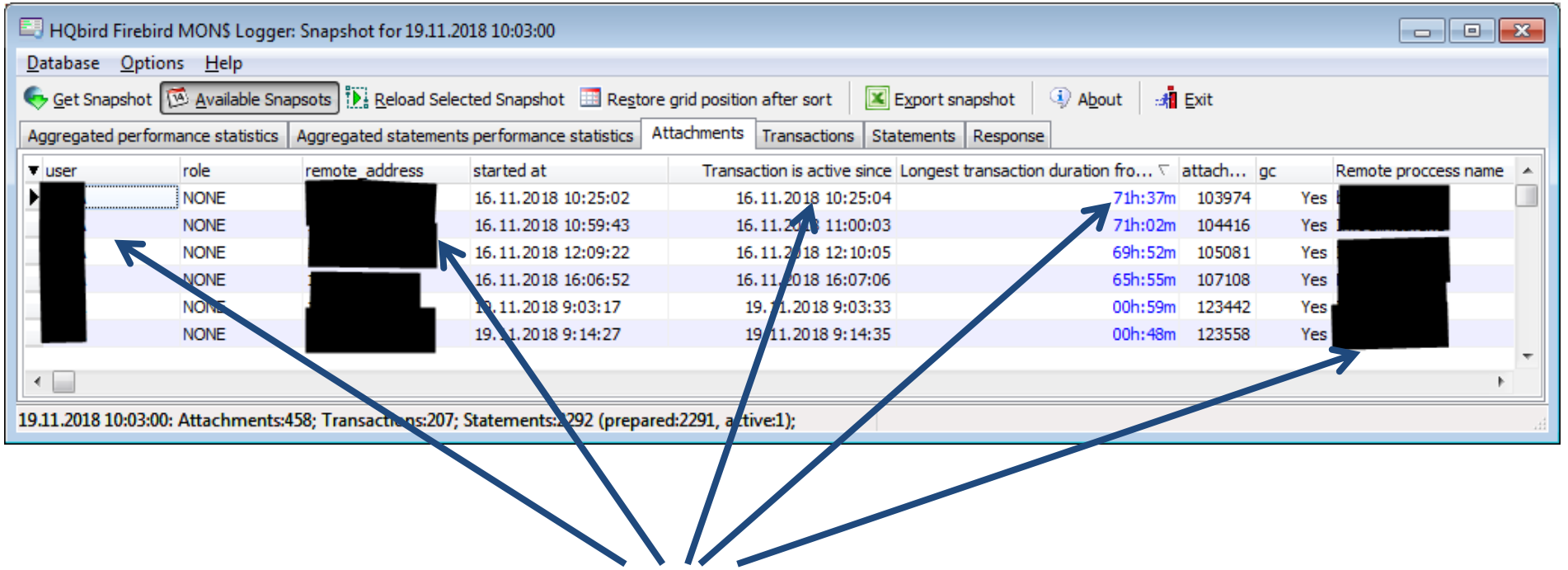
mon$transactions

- MON$TRANSACTION_ID - transaction ID
- MON$ATTACHMENT_ID - attachment ID
- MON$STATE - transaction state
  - 0: idle
  - 1: active
- MON$TIMESTAMP - transaction start date/time
- MON$TOP_TRANSACTION top transaction MON$OLDEST_TRANSACTION - local OIT number
- MON$OLDEST_ACTIVE - local OAT number
- MON$ISOLATION_MODE - isolation mode
  - 0: consistency
  - 1: concurrency
  - 2: read committed record version
  - 3: read committed no record version
- MON$LOCK_TIMEOUT - lock timeout
  - 0: no wait
  - 1: infinite wait
  - N: timeout N
- MON$READ_ONLY - read-only flag 0/1
- MON$AUTO_COMMIT - auto-commit flag
- MON$AUTO_UNDO - auto-undo flag
- MON$STAT_ID - statistics ID

- select * from mon$transactions
  order by mon$timestamp desc

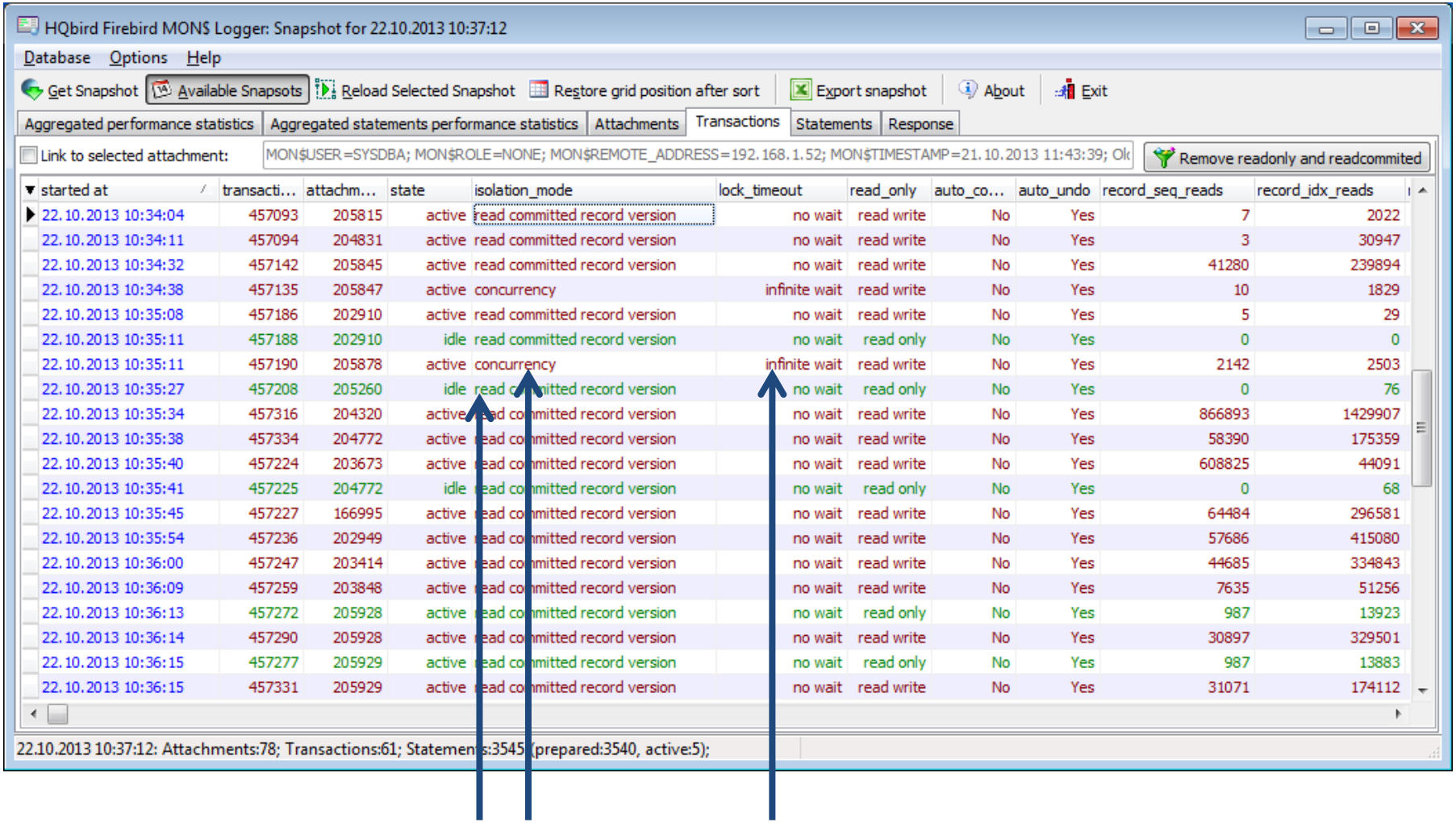
- select a.*, t.*
  from mon$attachments a, mon$transactions t
  where a.mon$attachment_id =
  t.mon$attachment_id
  order by t.mon$timestamp desc

MonLogger



who, where, when, how long, what application

read/write, RC & snapshot, wait



How to track conflicts and deadlocks

- Set trace config
- Run trace session
- Analyze trace log

- https://ib-aid.com/en/how-to-track-deadlocks-in-firebird/

LEGACY APPLICATIONS: WORKAROUNDS FOR ERROR IN TRANSACTIONS MANAGEMENT

Oops...

- If you do not have sources, or you do not understand sources, or ….
  The only way is to **terminate** these applications by at/cron schedule, for example, each hour or two.
- Active transactions in these applications decrease performance by accumulating record versions, blocking sweep, etc.
- ! rewrite these applications

BEST PRACTICES

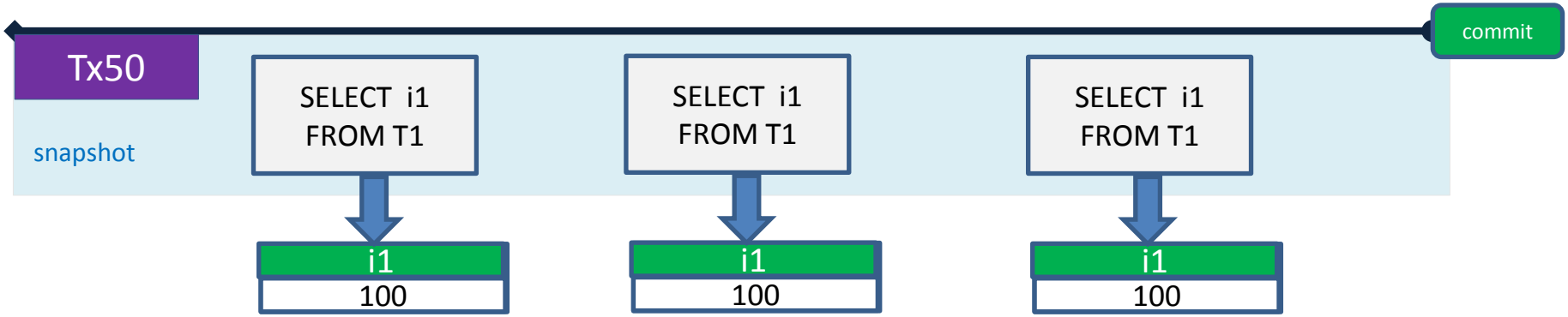
Exceptions from ideal transaction control

- Reports
- Goods balance
- Explicit record locking
- Robots

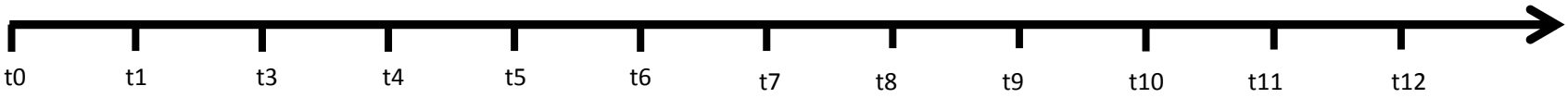
Reports

- Need data consistency
- Long queries
- Complex reports read the same data several times

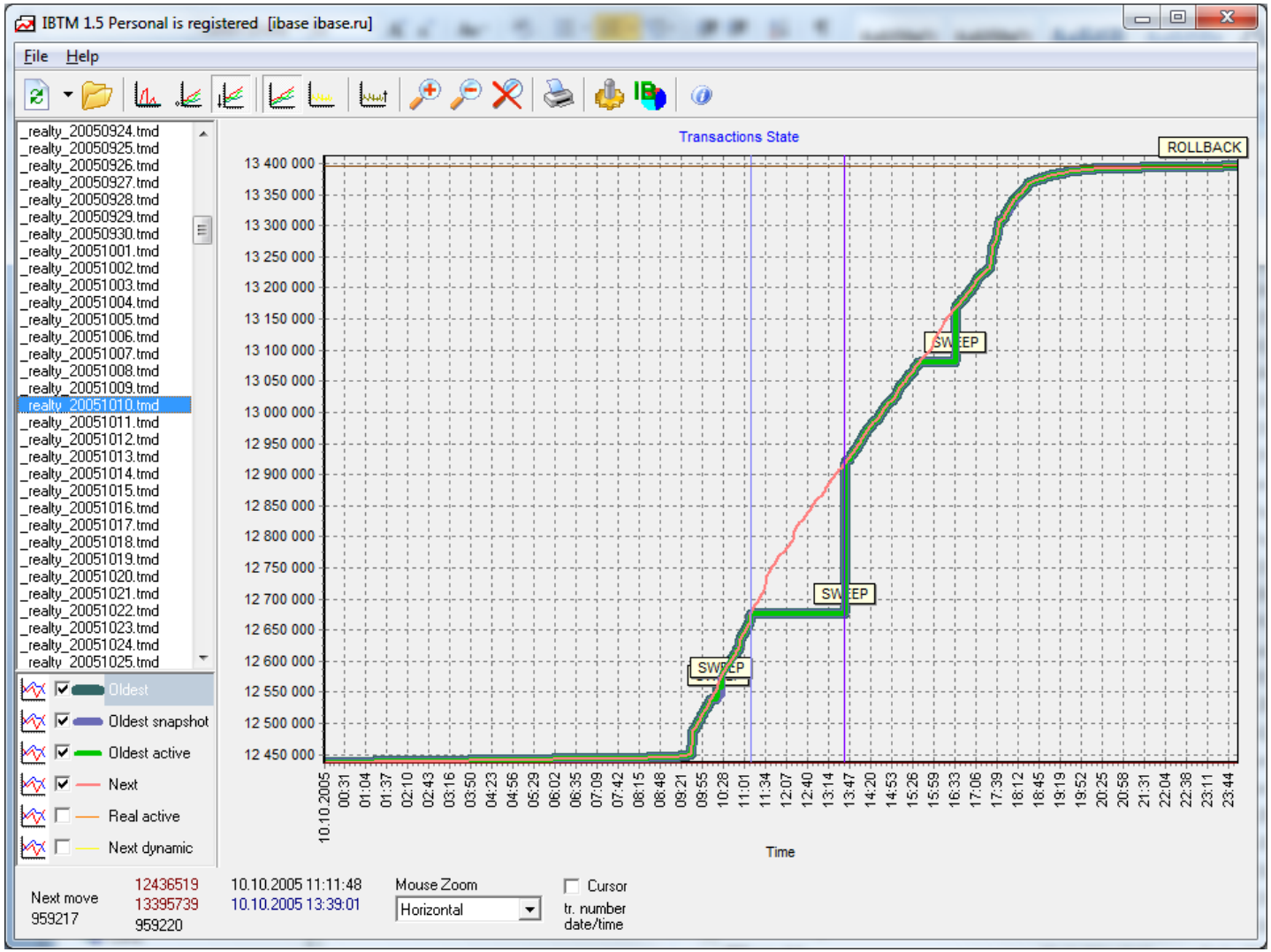
Reports - snapshot



No difference – wait/nowait (except concurrency), read/write



Heavy report example – OAT stuck



How to workaround long report problem

- Most reports does not need real-time data

- Change logic of data processing
- Scaling
  - Replication
  - Transferring data to another DB with Execute Statement On External
  - Nbackup

Change logic of data processing – Stored aggregates

- 1 order - ~10 goods
- 100 orders per day
- 100*10 = 1000 records per day
- 365000 records per year

- Store “order_total” in ORDERS table – 10 times less records
- Pro: less records, faster queries
  - No update conflicts if there is no concurrent order editing
- Con: additional field in ORDERS

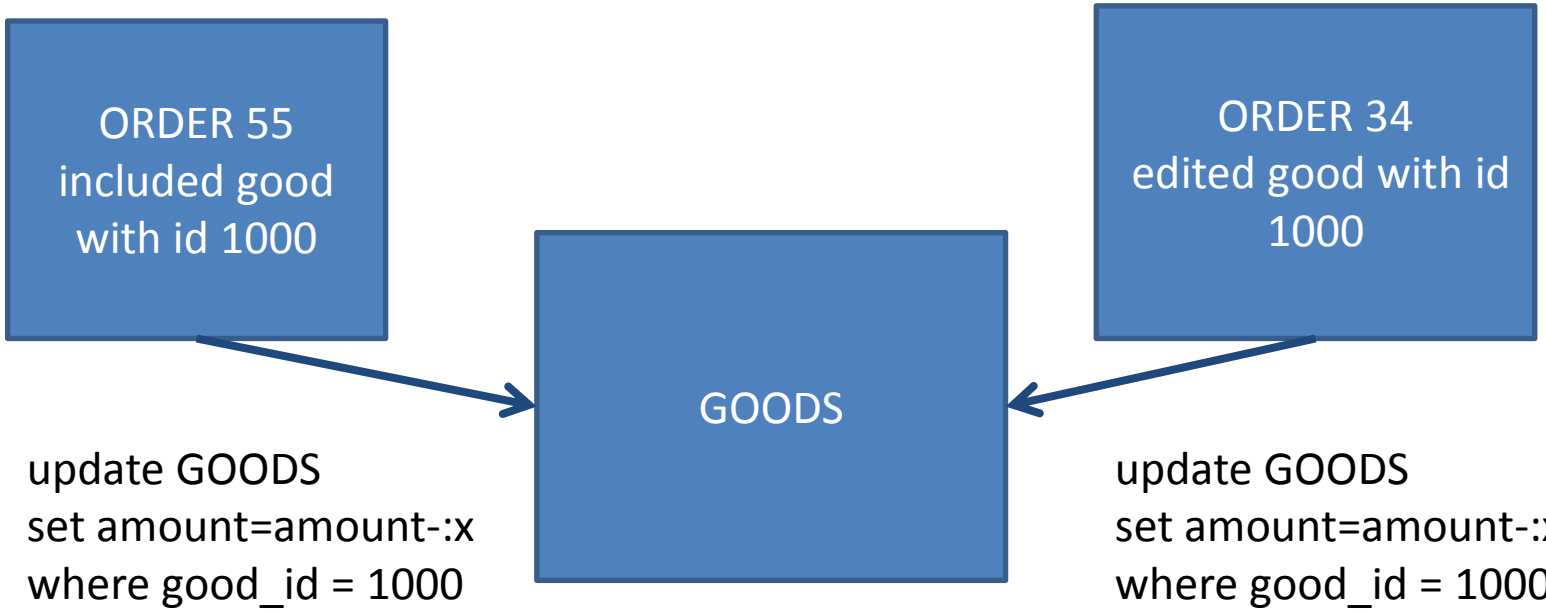
If you want to go further…

- To store sum by day, month, …
- Updates by triggers “in place” won’t work – too high possibility of lock conflicts
- Solution? Routine updates
  - Routine procedure must be run in exclusive mode
    - Using generator
    - Using consistency isolation mode
  - By schedule (at night)

Goods balances – update locks

- Change goods AMOUNT while order is processed
-
- Insert - set AMOUNT = AMOUNT - new.INORDER
- Delete - set AMOUNT = AMOUNT + new.INORDER
- Update - set AMOUNT = AMOUNT + new.INORDER – old.INORDER

- There may be conflicts when 2 people sell same good_id
  - Long transaction will lock all concurrent order processing
  - Short transactions have less chances to get update conflict, and may be retried



Editing one order by 2 users is a rare case, but using same item is not rare

Goods balances- solution

- CREATE TABLE MOVEMENTS(
  GOOD INTEGER NOT NULL REFERENCES GOODS, AMOUNT INTEGER NOT NULL)

- CREATE TABLE GOODS_AMOUNTS_AGG(
  GOOD INTEGER NOT NULL REFERENCES GOODS, AMOUNT INTEGER NOT NULL)

- On insert update and delete MOVEMENTS do
- **INSERT** INTO GOODS_AMOUNT_AGG
  (GOOD, AMOUNT) VALUES
  - (NEW.GOOD, NEW.AMOUNT);
  - (NEW.GOOD, NEW.AMOUNT-OLD.AMOUNT);
  - (OLD.GOOD, -OLD.AMOUNT);

- CREATE VIEW GOODS_AMOUNT
  (GOOD, AMOUNT) AS
  SELECT GOOD, SUM(AMOUNT)
  FROM GOODS_AMOUNT_AGG
  GROUP BY GOOD

- CREATE PROCEDURE GOODS_AMOUNT_ROLL_UP AS
  DECLARE GOOD INTEGER;
  DECLARE TOTAL INTEGER;
  BEGIN
  FOR SELECT GOOD, SUM(AMOUNT)
  FROM GOODS_AMOUNT_AGG
  GROUP BY GOOD
  HAVING COUNT(*)>1 – *interested of 2 or more records*
  INTO :GOOD, :TOTAL
  DO
  BEGIN
  DELETE FROM GOODS_AMOUNT_AGG
  WHERE GOOD=:GOOD;
  INSERT INTO GOODS_AMOUNT_AGG
  (GOOD, AMOUNT) VALUES(:GOOD, :TOTAL);
  END
  END
- Run procedure in concurrency (or consistency)

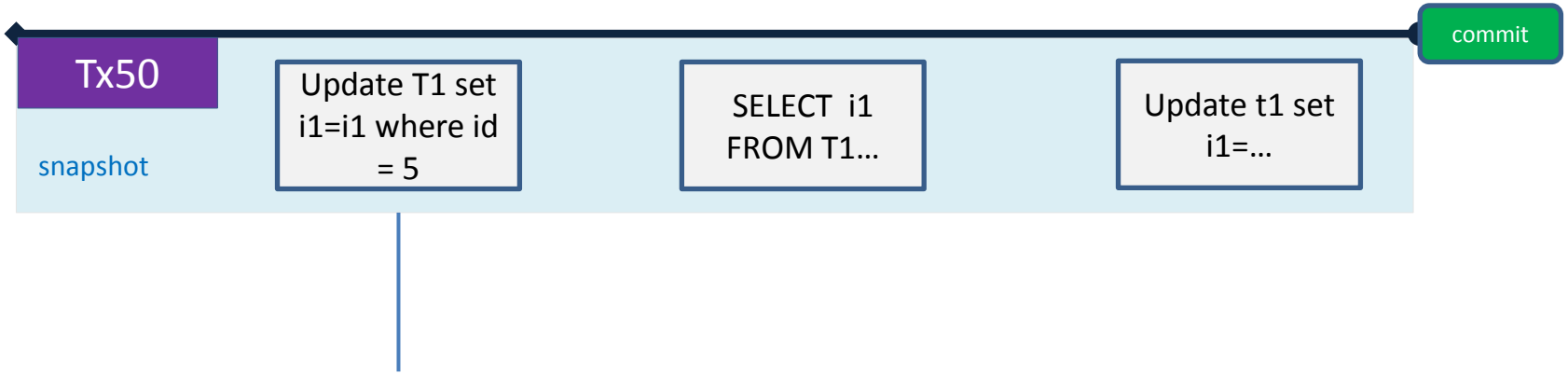
Exclusive document editing

- Goal – implement exclusive changes
- Rollbacks are not welcome
- Need explicit record locking

How to implement explicit record locking

- Blank update in long transaction
  - Or SELECT … FOR UPDATE WITH LOCK
- Flags at business logic level

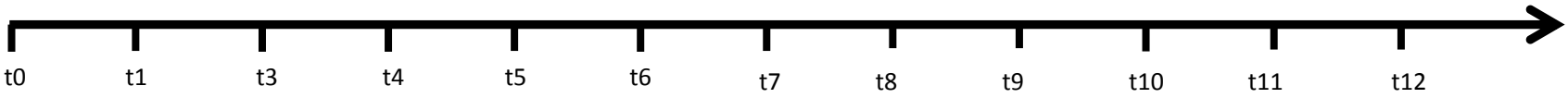
Blank update



First update creates record version,
preventing other transactions to update this record

Pro: Easy to implement

Con: Trigger fires on update



SELECT … FOR UPDATE WITH LOCK

- Same as blank update
- Can lock several records
- Locks record on fetch
  - Result returns one record per one fetch (no buffering)
- Useless for aggregates (SUM, AVG, COUNT, …)

- Locking in the versioning server is not normal
- It maybe not enough to choose appropriate transaction isolation level

Flags at business logic level

- Add User and TimeStamp fields, or create additional table
- When you want to “lock”, write USER and CURRENT_TIMESTAMP in short transaction

- if user <> myself then
  if TimeStamp is far then
  UPDATE set User, TimeStamp
  else Fail(“locked by user User at TimeStamp”)
  else
  UPDATE set TimeStamp

- Additional table need to be cleared (disconnected apps)

Robot rules

Reading robots

- Use read-only ReadCommitted
- Try to do work in one transaction, if possible
- Multi-tier - connection and transaction pooling
- Goals
  - Do not stuck OAT
  - Do not advance Next too much

Writing robots

- Do not keep attachment open
  - attach, do work, close;
- Keep transactions short
- Try to do work in one transaction, if possible
- Goals
  - Do not stuck OAT

- Thank you!

- www.firebirdsql.org
- www.ib-aid.com
- support@ib-aid.com