# How Firebird transactions work

**Dmitry Kuzmenko**
**www.IBSurgeon.com**
**www.ib-aid.com**

# Thank you!


24-25 OCTOBER
PRAGUE
Firebird Conference 2014
Firebird Conference 2014

PLATINUM

# IBSurgeon


- Tools and consulting
- Platinum Sponsor of Firebird Foundation
- Founded in 2002: 12 years of Firebird and InterBase recoveries and consulting
- Based in Moscow, Russia, world-wide 24x7 support

# Agenda

What is transaction? Why we need it?

How we will present about transactions

Records and versions

Transactions and record versions

Transaction Inventory

Record visibility in transactions

Transaction Markers and their evaluation

Some conclusions

# What is transaction?

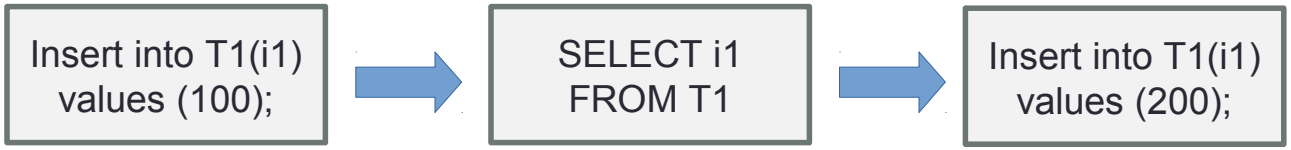
- Transaction as a general concept of any dynamic system
- “Classic” example
  - begin
    - -- move money from account1 to account2
    - Decrease account1
    - Increase account2
  - end – commit/rollback

  - Transaction Managers

# Database transaction definition

- a unit of work performed against a database, and treated in a coherent and reliable way independent of other transactions.
- A database transaction, by definition, must be **A**tomic, **C**onsistent, **I**solated and **D**urable

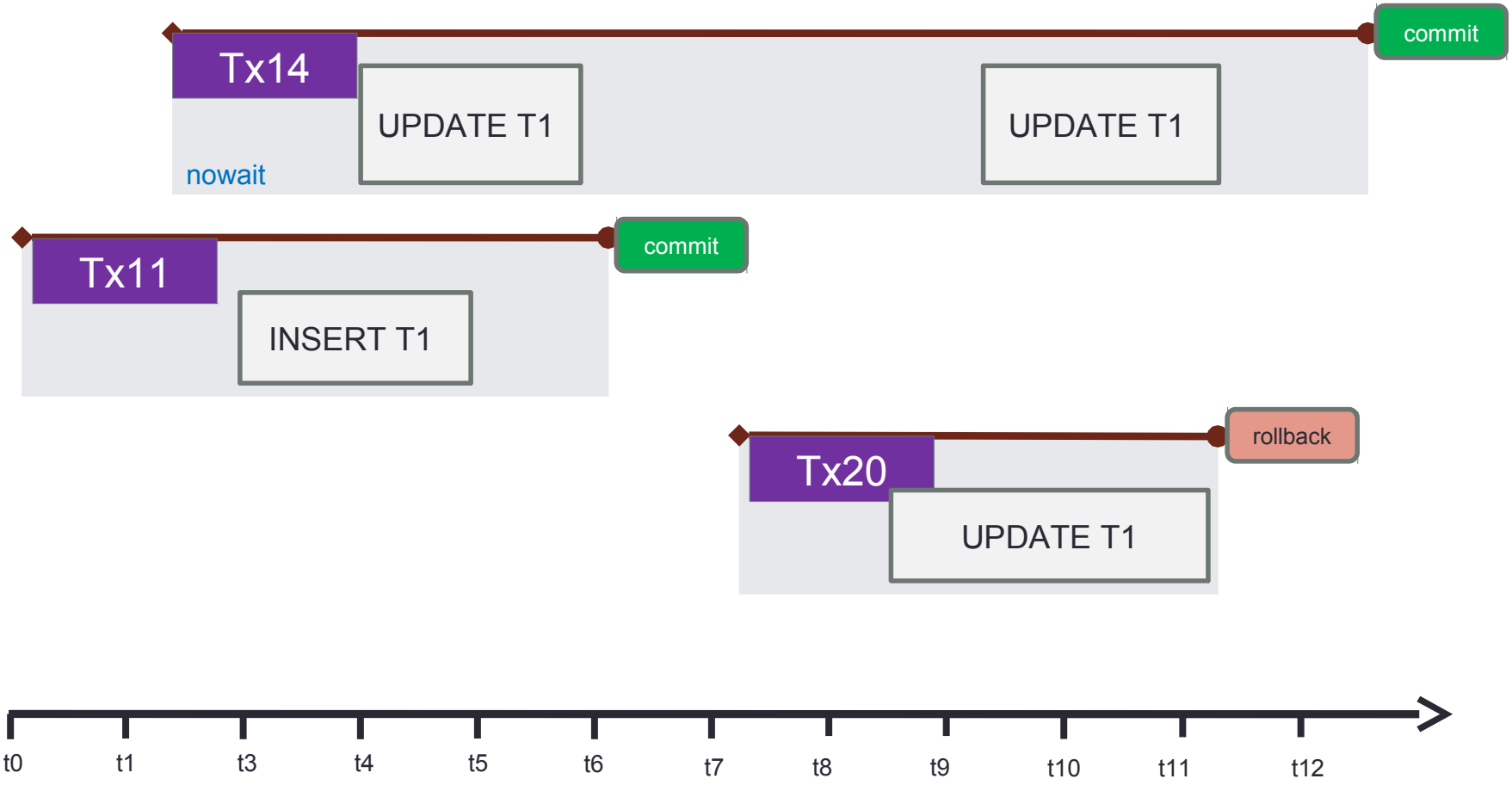
# In ideal world




only serial operations


ITM
WAX 267
LMJ 367
GOE 361
TXX 762
GAO 621
MHD 148

# In real world



# The ultimate purpose of transaction:

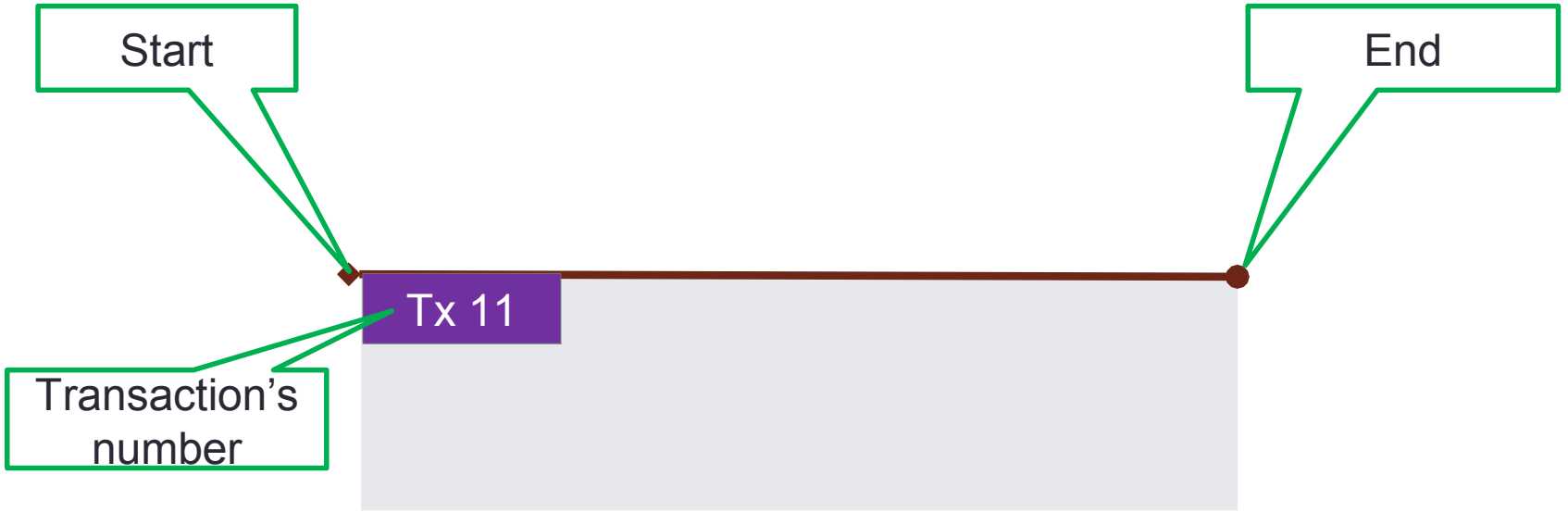
- Concurrent execution of operations should lead to the exactly the same result as sequental execution of operations.

*In simple words: each transaction should run as the only transaction.*

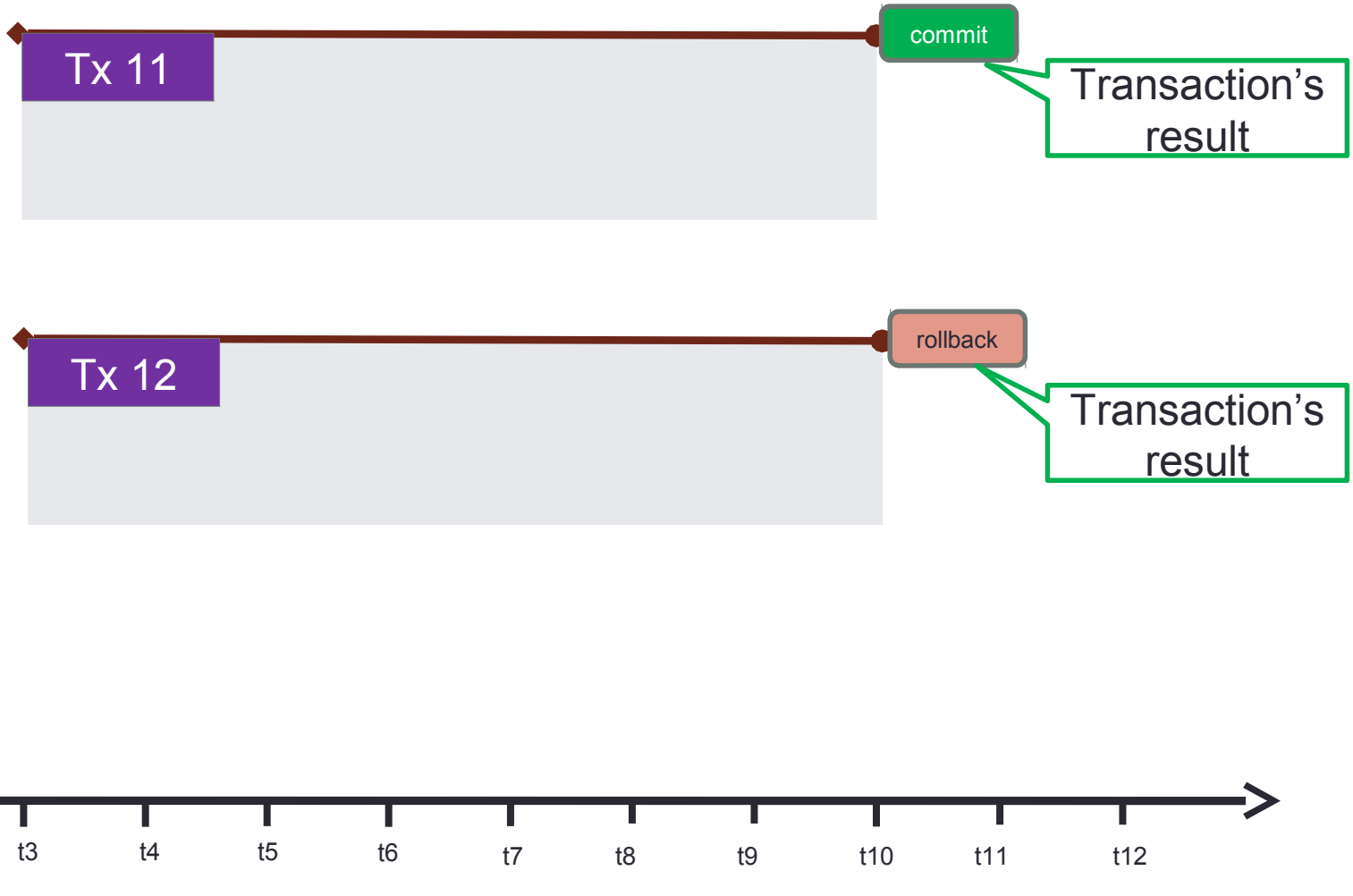
**For each [snapshot] transaction Firebird engine should maintain a stable view of the database.**

# How Firebird does implement stable view for each transactions?

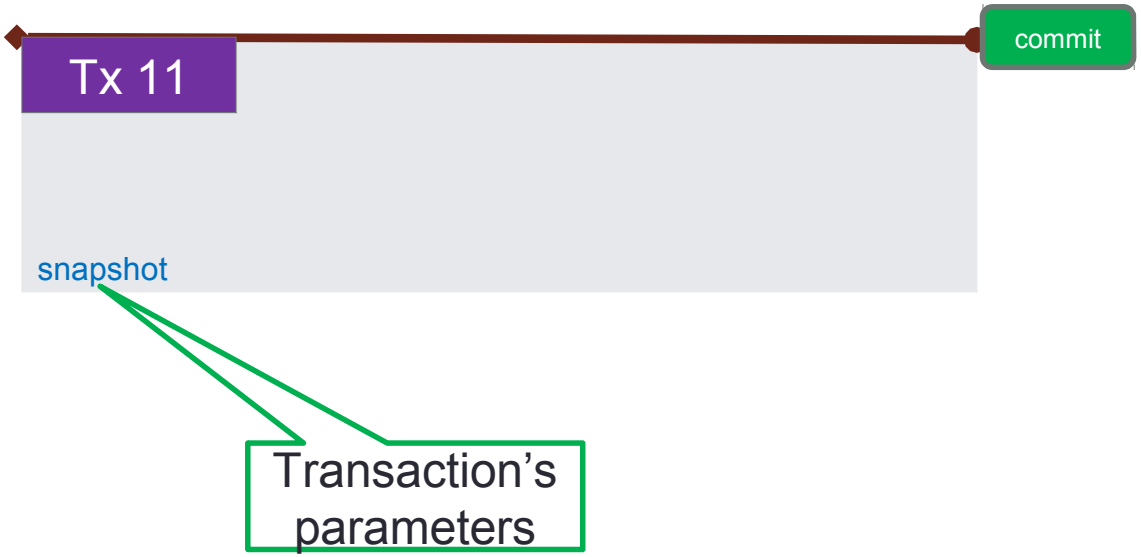
# How we will present about transactions



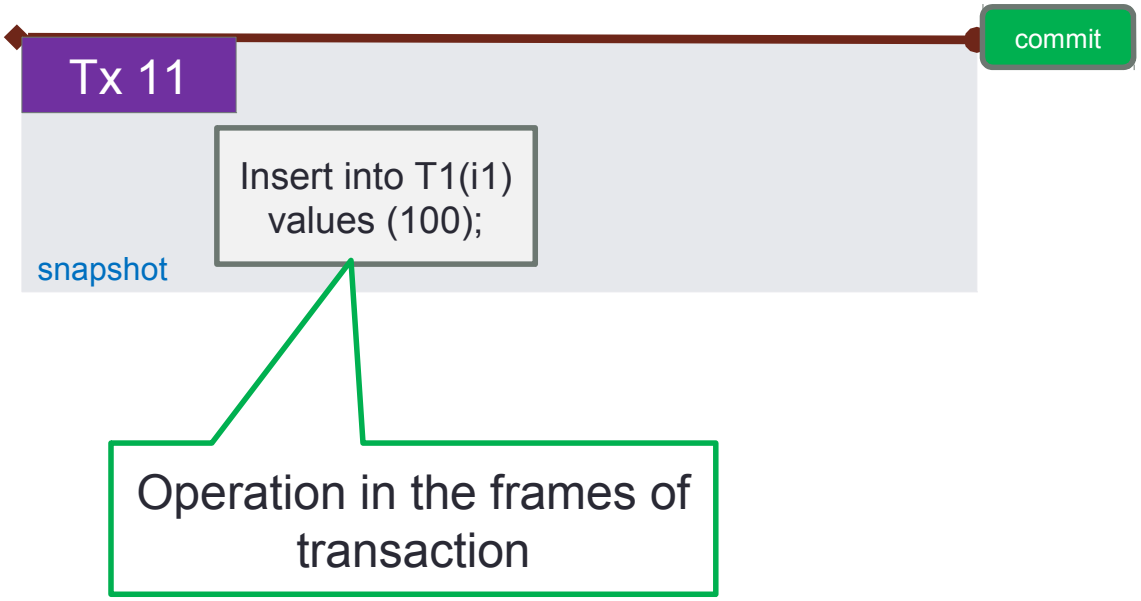

# How we will present about transactions



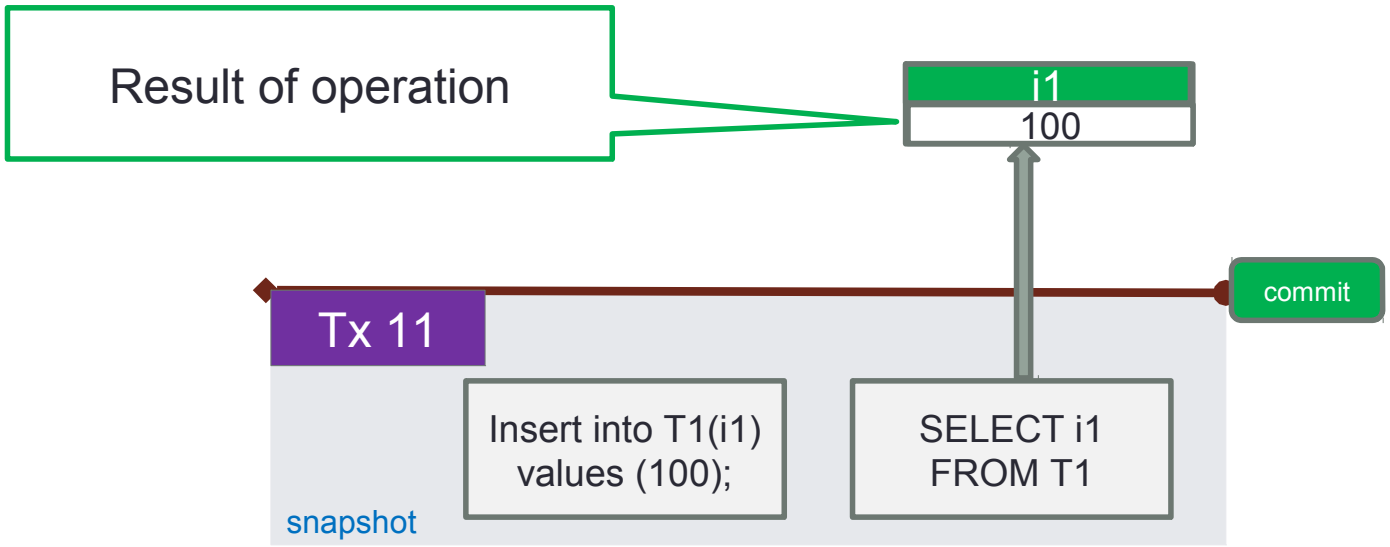
# How we will present about transactions




# How we will present about transactions




# How we will present about transactions




# Now let's start...

Basics your [probably] know:

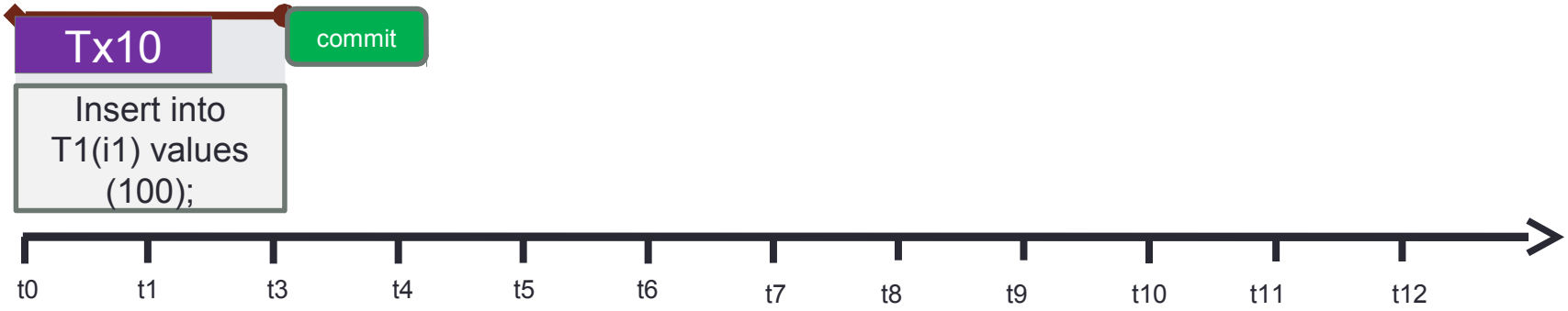
- Everything in the database is done within transaction

- Each transaction get it’s own incremented number 1, 2, 3, … etc

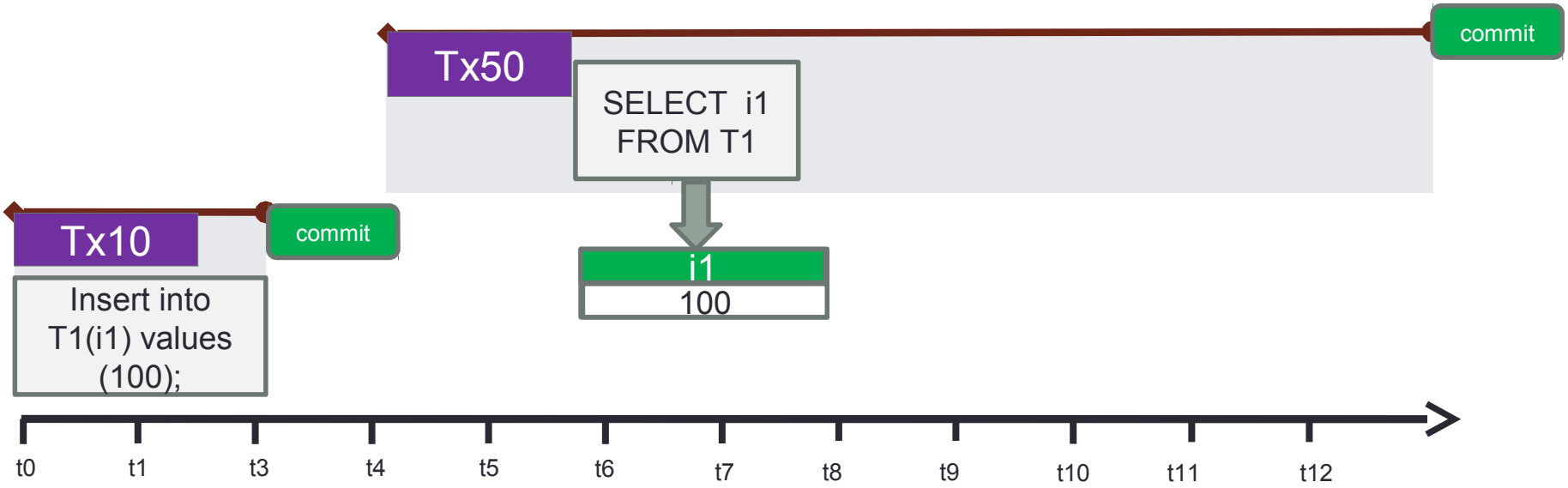
- Firebird is a multi-version engine (each record in *Firebird* can have versions)

Record versions is a key thing for understanding transactions' work in Firebird.

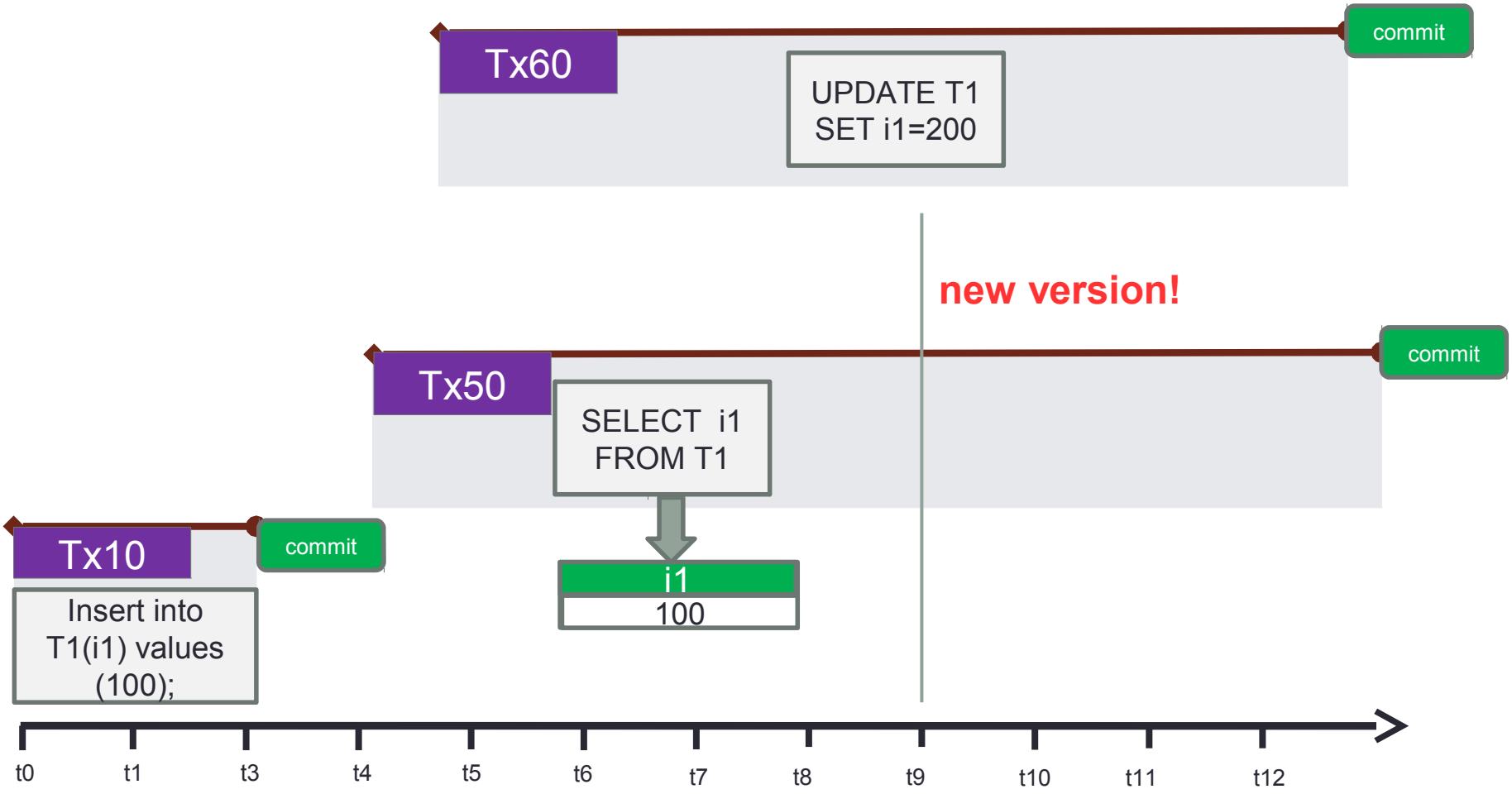
# How record versions appear



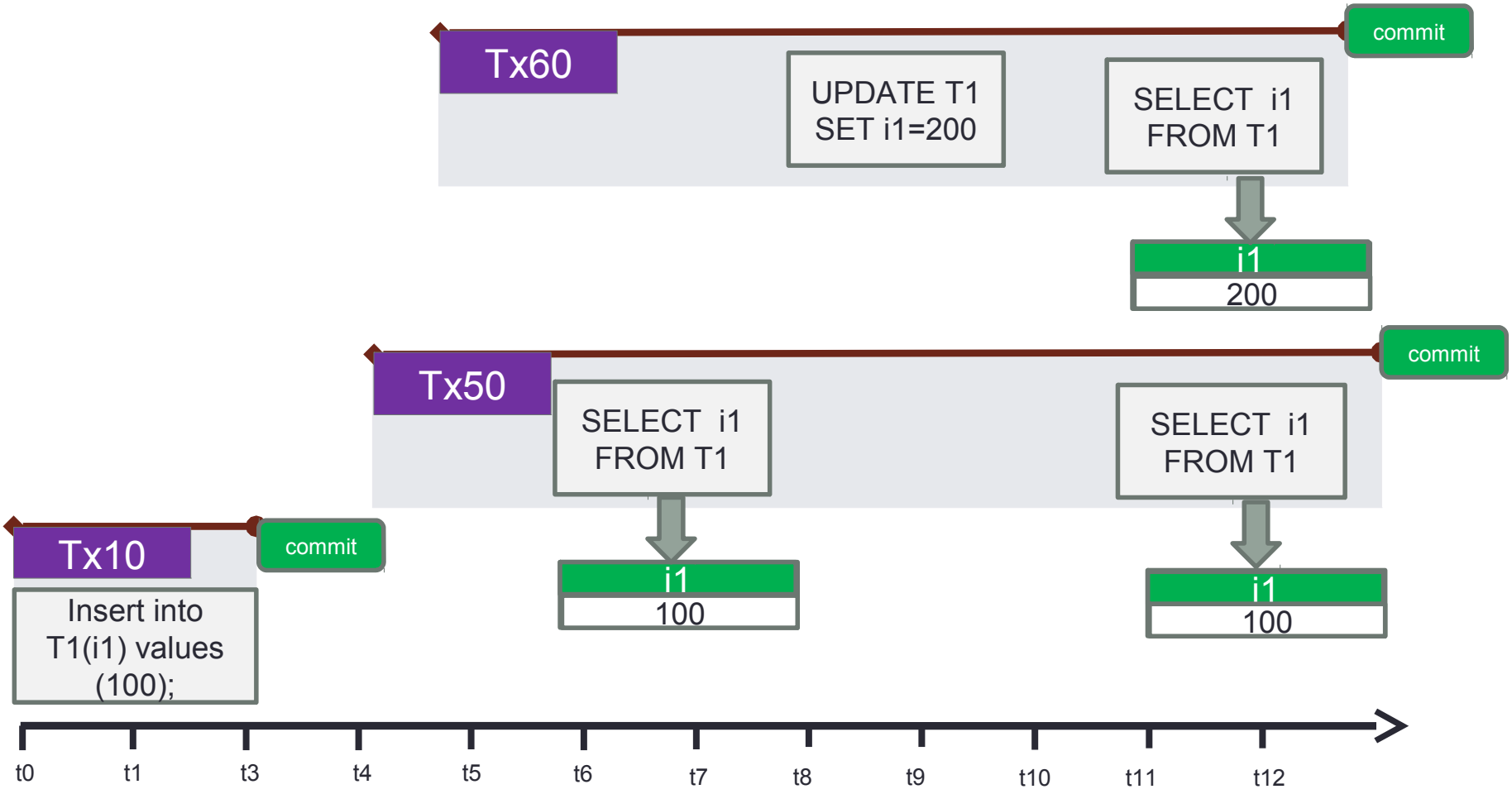
# How record versions appear



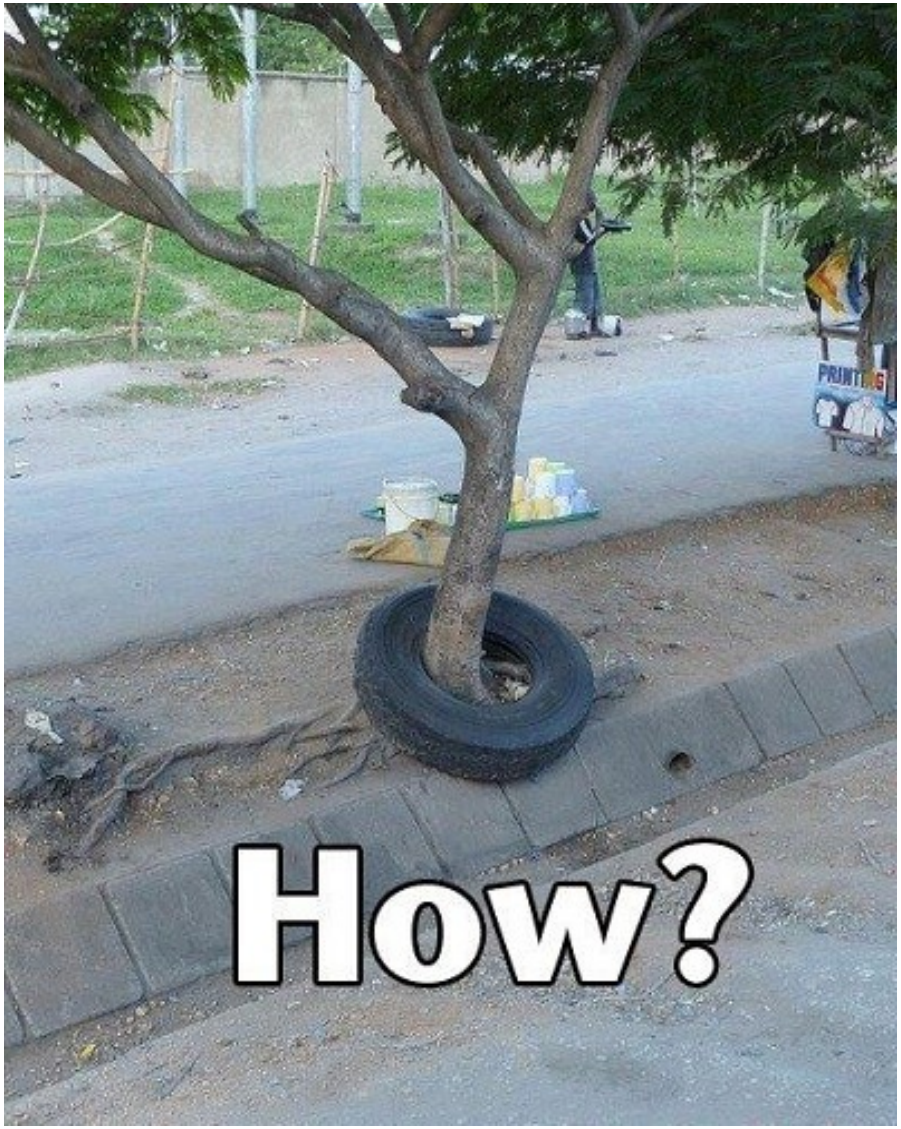
# How record versions appear



# How record versions appear

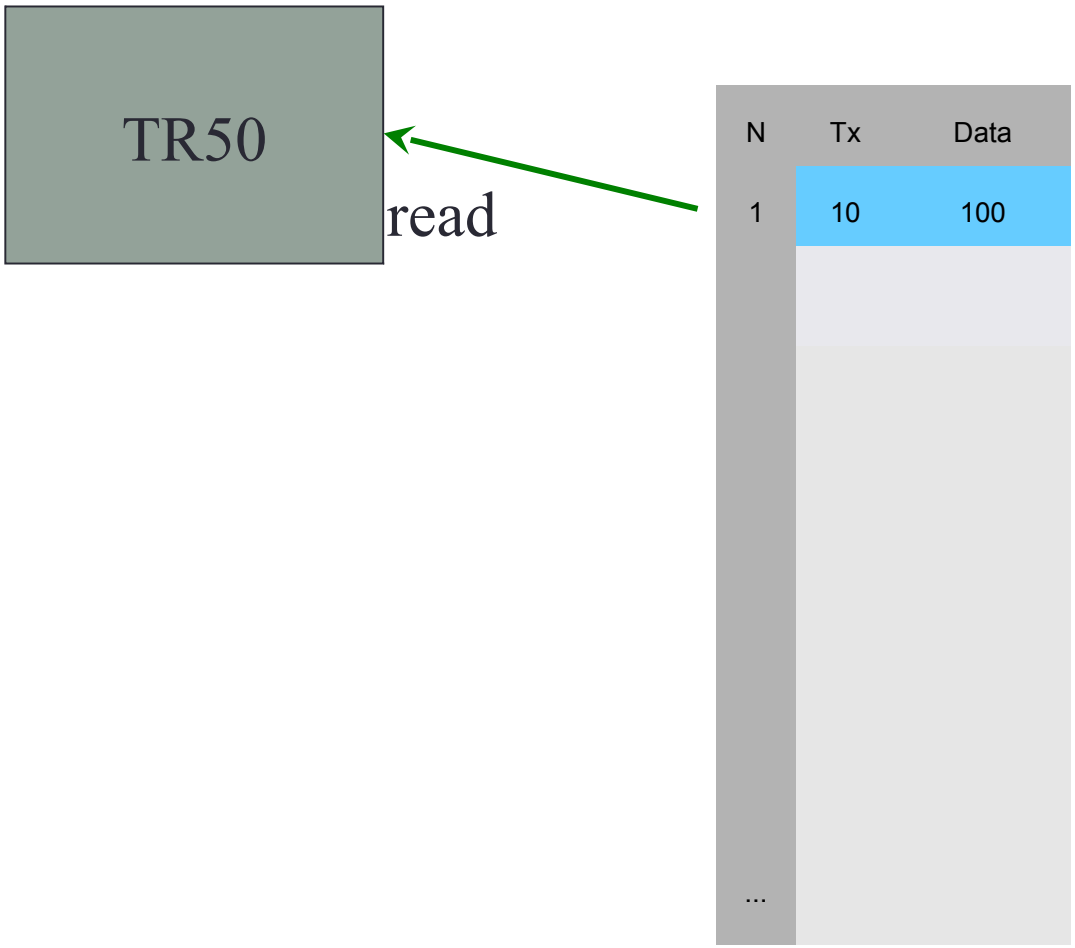


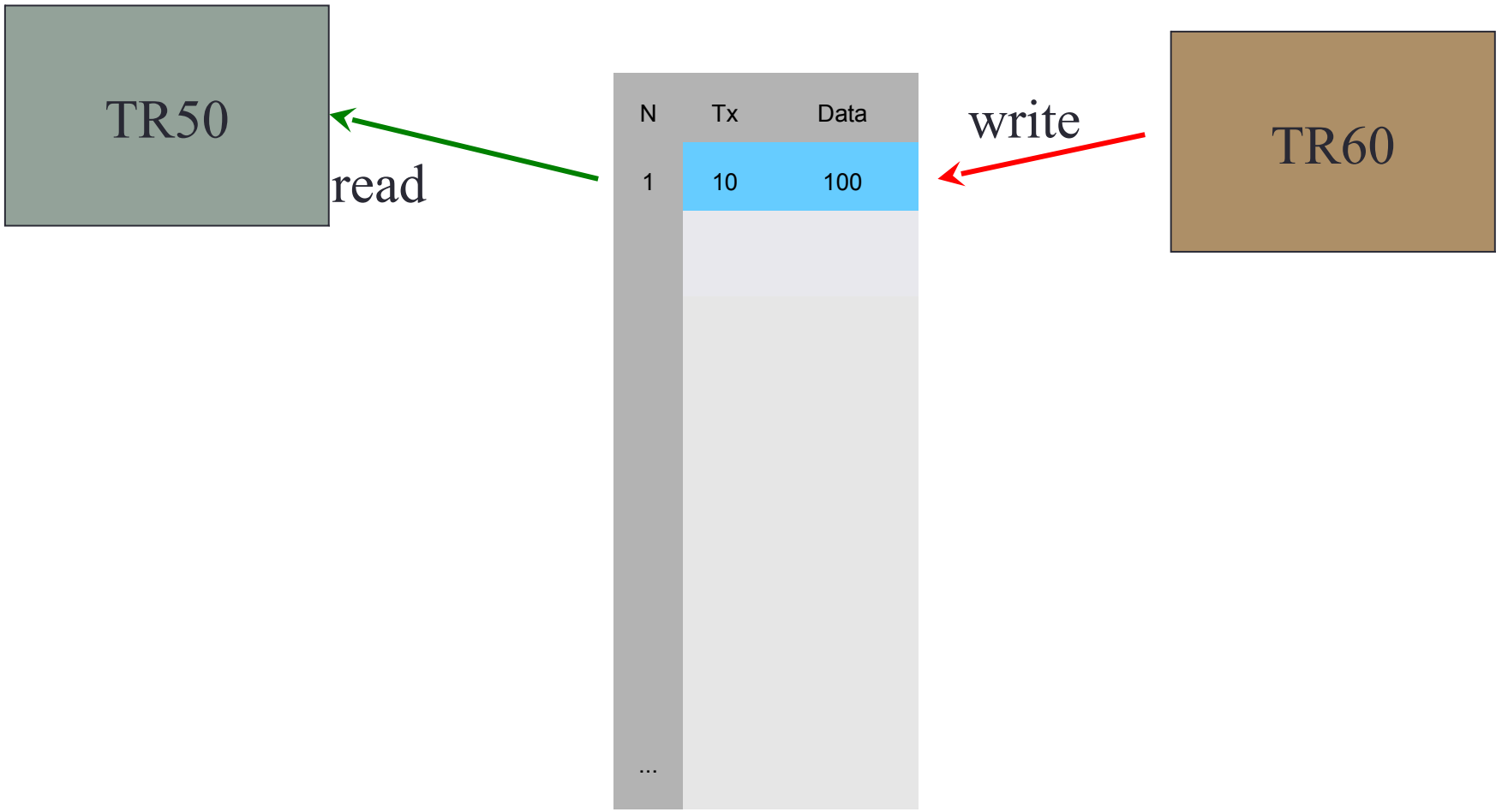
# How it works?

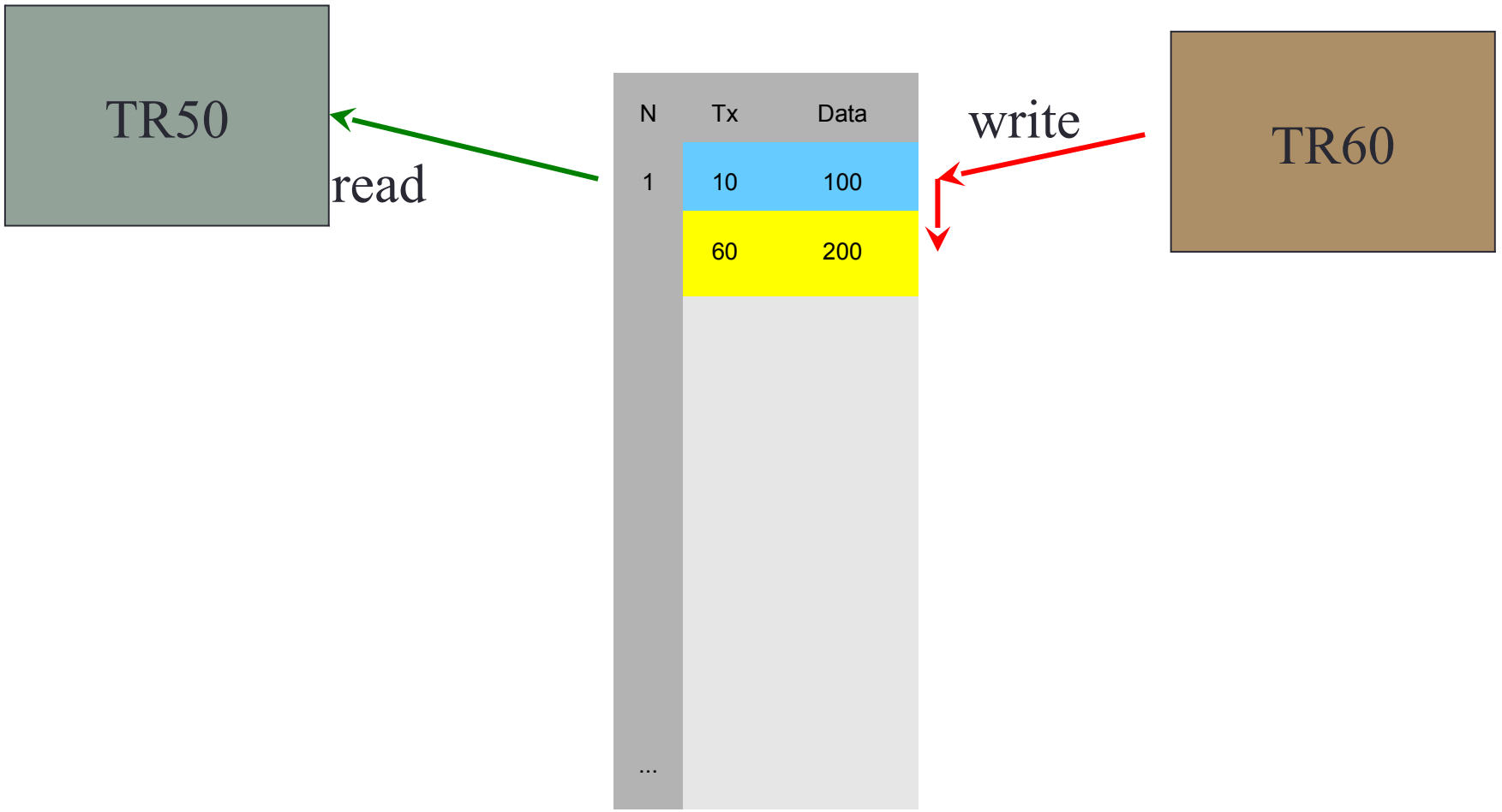


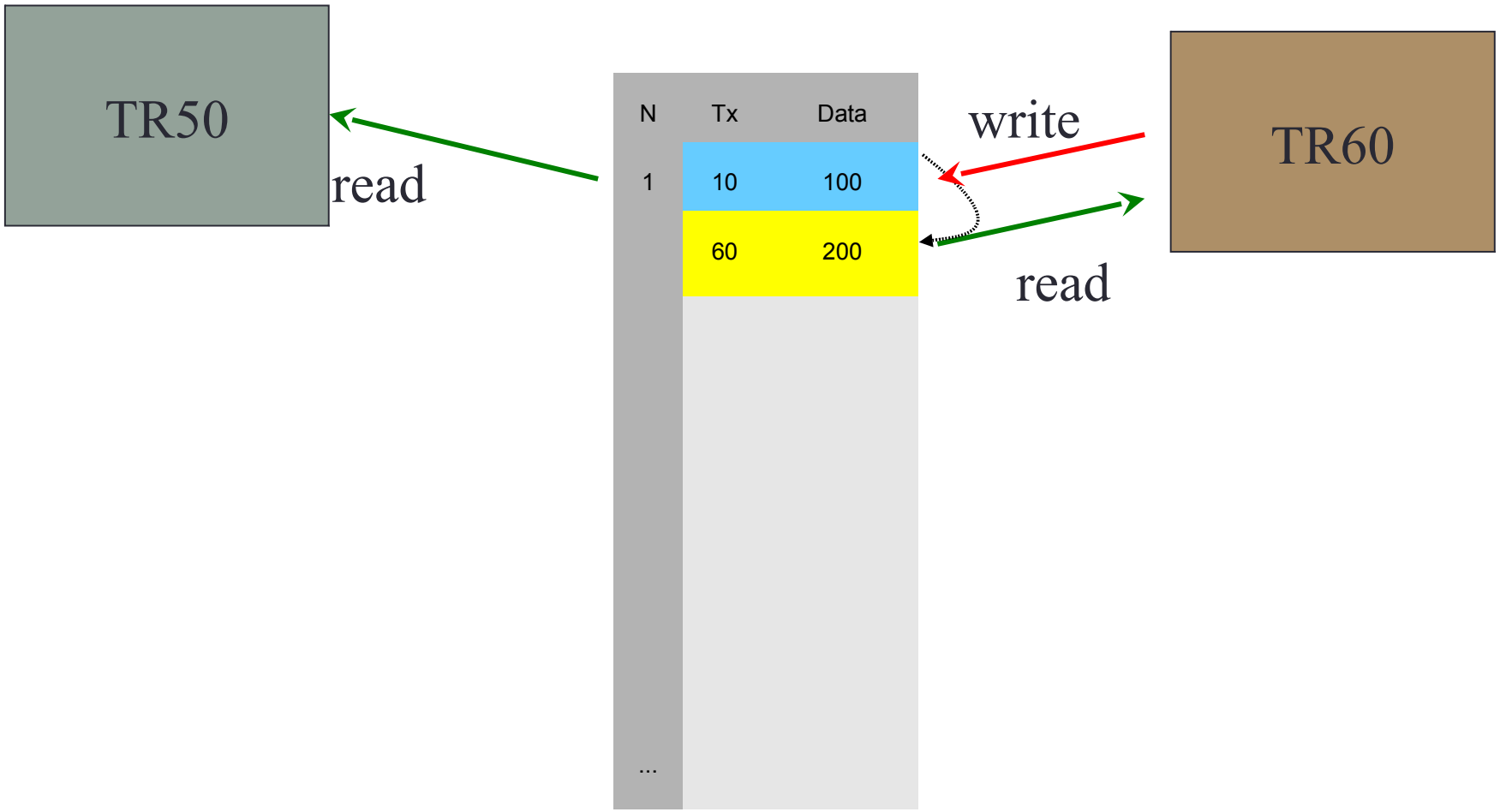
# Each record version has transaction #

| N on page | Transaction number | Datafield1, datafield2 |
|---|---|---|
| 1 | 50 | 100 |


TR50
read
N
Tx
Data
1
10
100
...


TR50
read
N
Tx
Data
1
10
100
...
write
TR60


TR50
read
N
Tx
Data
1
10
100
60
200
...
write
TR60


TR50
read
N
Tx
Data
1
10
100
60
200
...
write
read
TR60

# Some intermediate conclusions

1. No “locks” are placed on the record
2. There can be a lot of committed versions for one record
3. Versions may be needed or not. If not, they can be considered as “garbage”.
4. Only one non-committed version can exist for the record

(2 active transactions can’t update the same record)

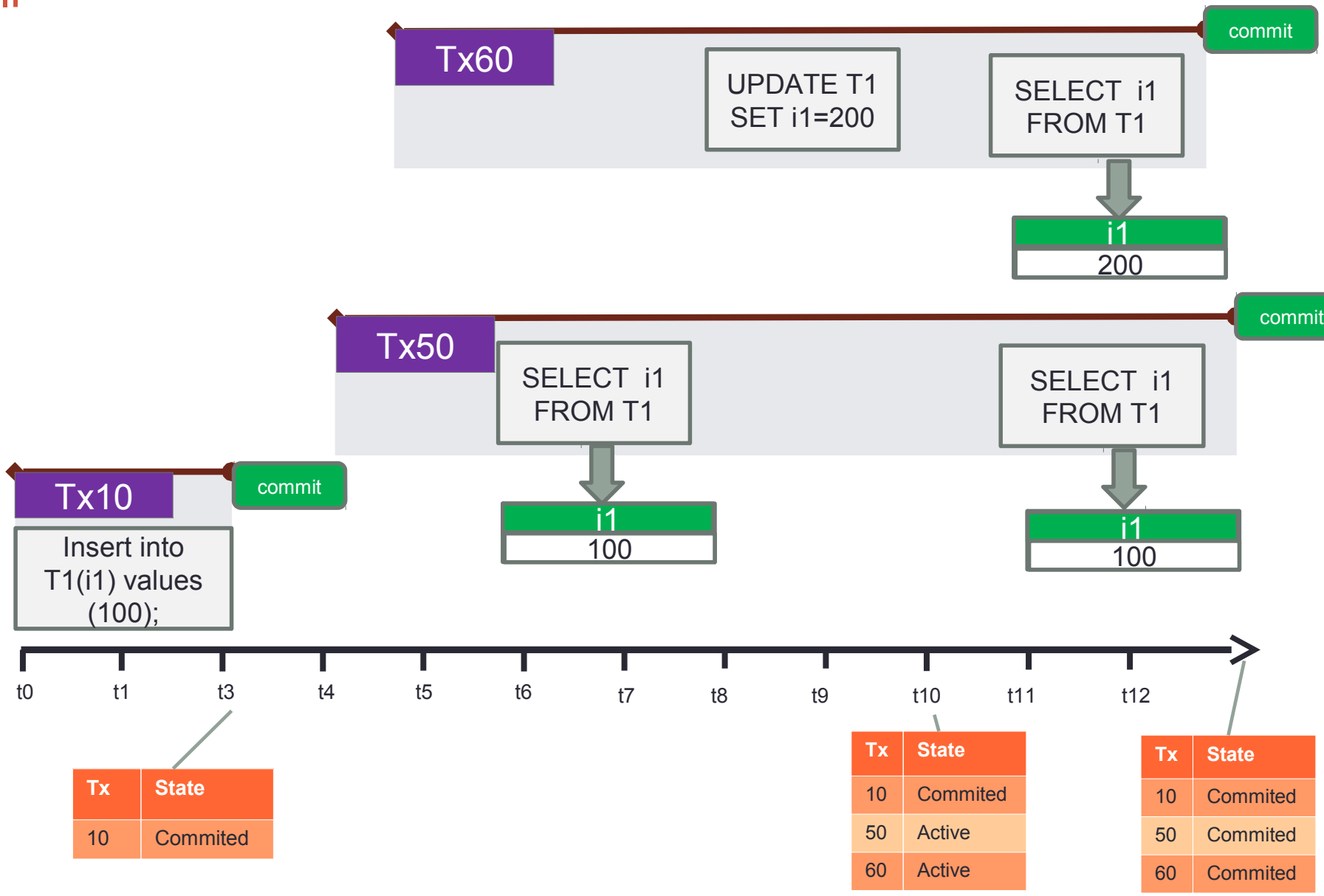
# How server knows about transactions states? Is transaction Active or not?

- TIP – Transaction Inventory Pages
  - Linear list of transaction states, from 1 to last transaction number
  - Stored in the database
  - Limitation — 2 billions of transactions

# Transaction states

- Each transaction is represented in Transactions Inventory by it's state
  - 00 – Active
  - 01 – Committed
  - 10 – Rolled back
  - 11 – Limbo (distributed 2-phase transactions)

# TIP

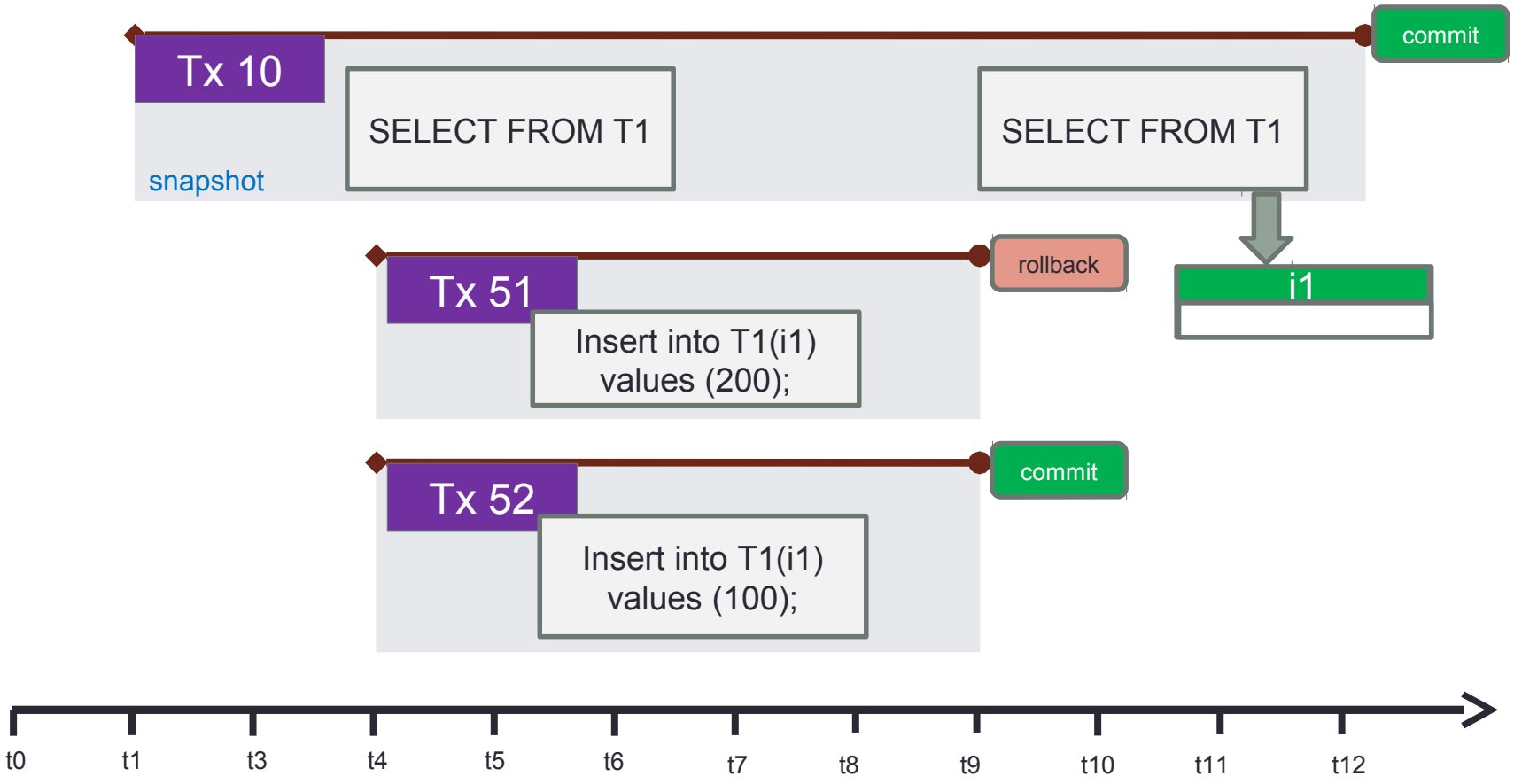

Tx60
UPDATE T1
SET i1=200
SELECT i1
FROM T1
commit
i1
200
Tx50
SELECT i1
FROM T1
SELECT i1
FROM T1
commit
i1
100
i1
100
Tx10
Insert into
T1(i1) values
(100);
commit
t0
t1
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12
Tx
State
10
Commited
Tx
State
10
Commited
50
Active
60
Active
Tx
State
10
Commited
50
Commited
60
Commited

# Transaction isolation levels

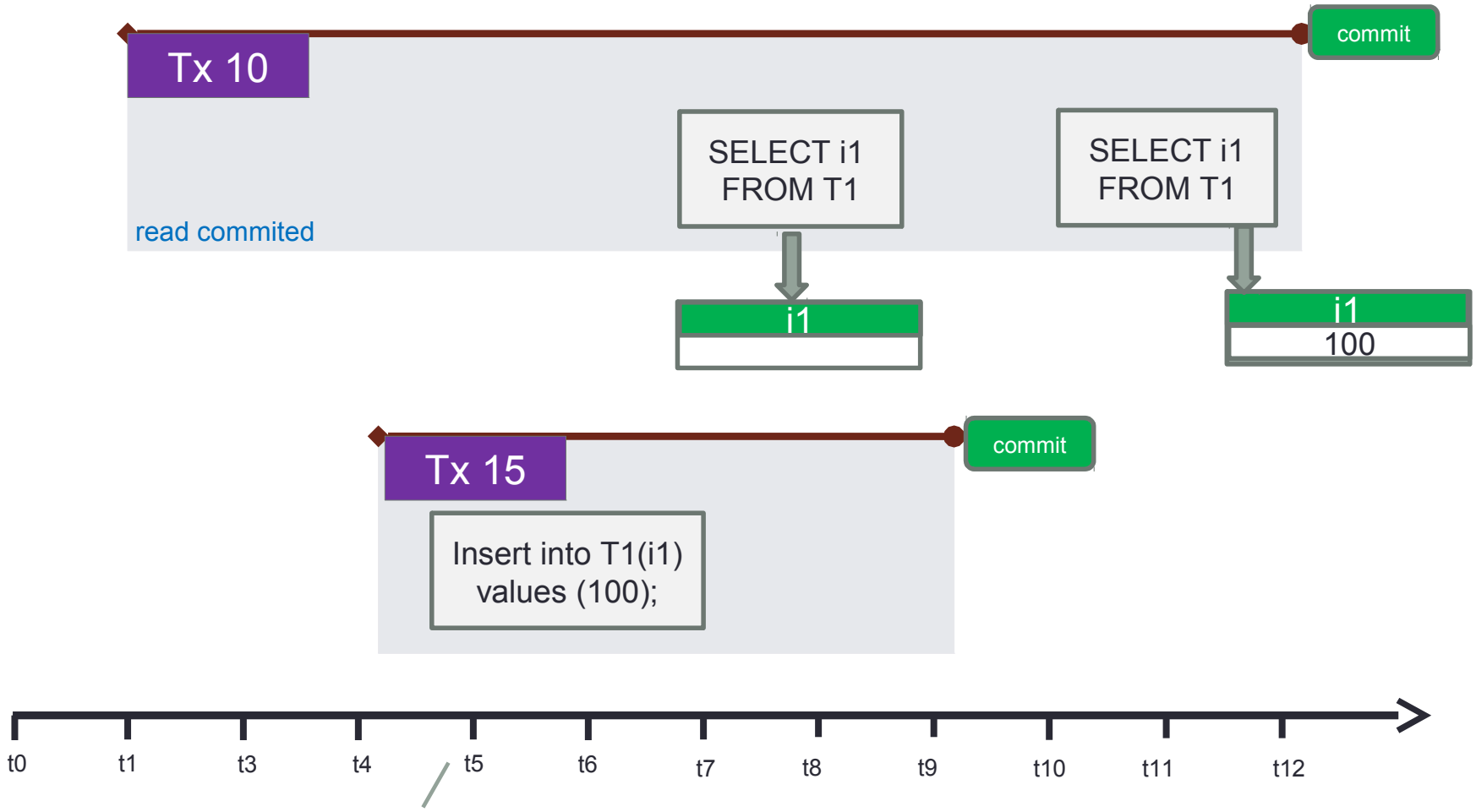
# Isolation levels in Firebird

| **Isolation levels in Firebird** |
| --- |
| READ COMMITED |
| SNAPSHOT |
| SNAPSHOT WITH TABLE STABILITY |

# Snapshot


Tx 10
SELECT FROM T1
SELECT FROM T1
commit
snapshot
Tx 51
rollback
Insert into T1(i1)
values (200);
i1
Tx 52
commit
Insert into T1(i1)
values (100);
t0
t1
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12

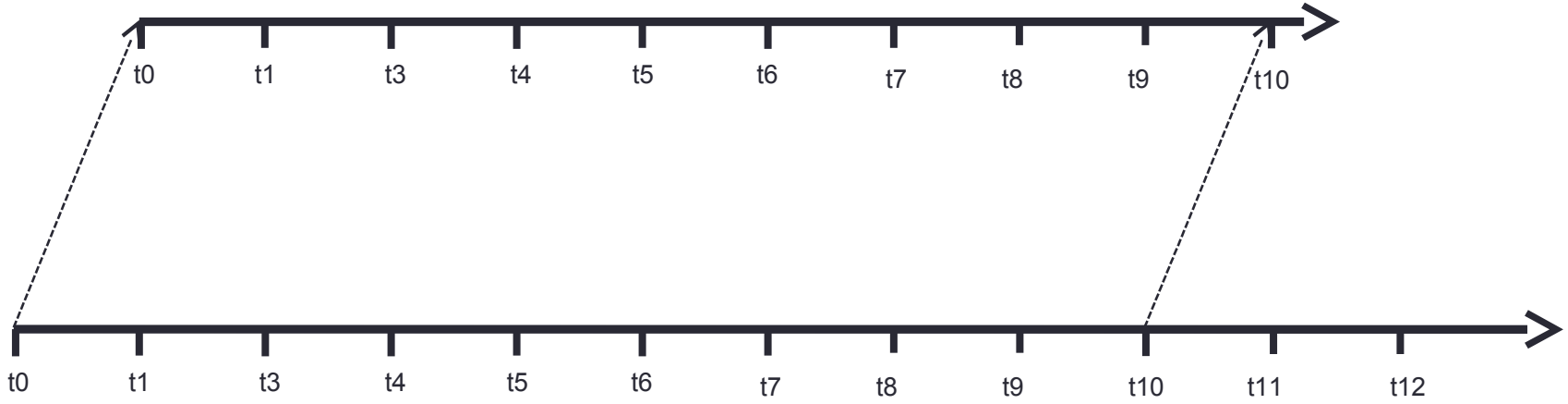
# Read Commited



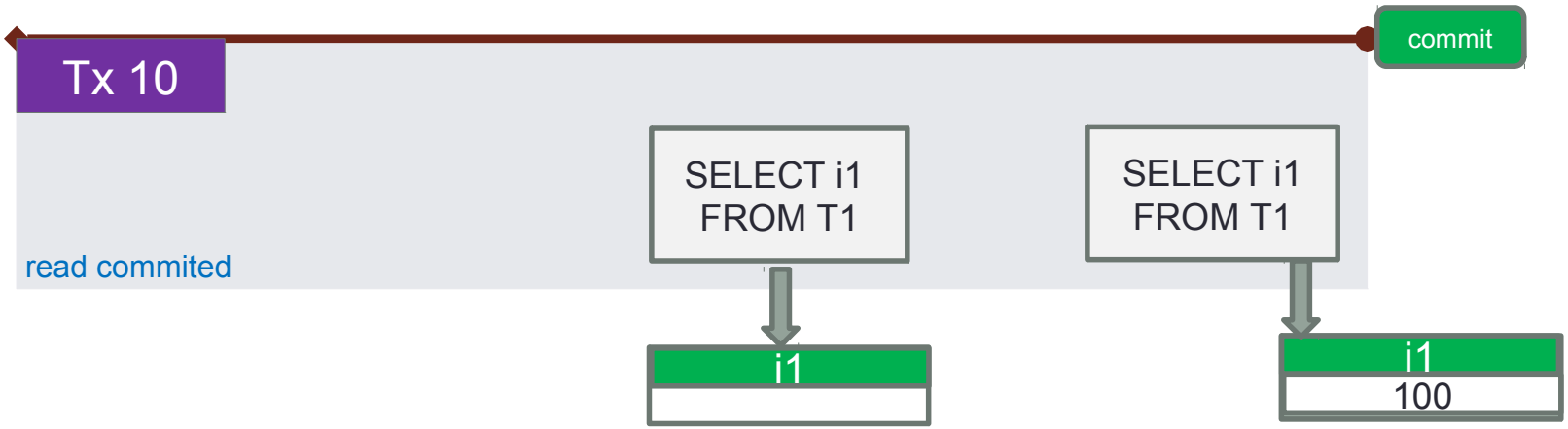
# Read Commited and Snapshot

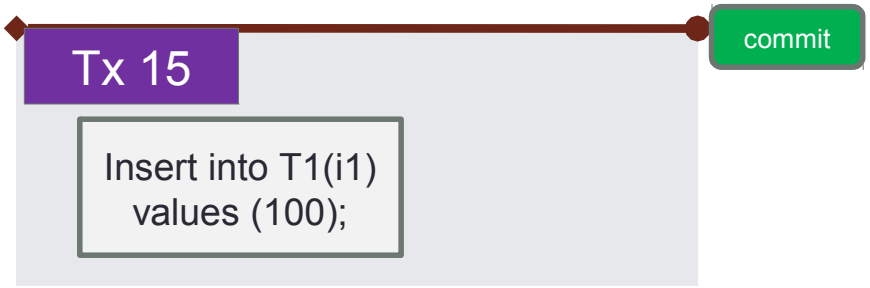
**Read Committed** transactions “see” global TIP.
That’s why they can read committed changes of other transactions


**Snapshot** copies TIP on it’s start. It does not see any changes made by other committed transactions after snapshot start



# TIP for Read Commited



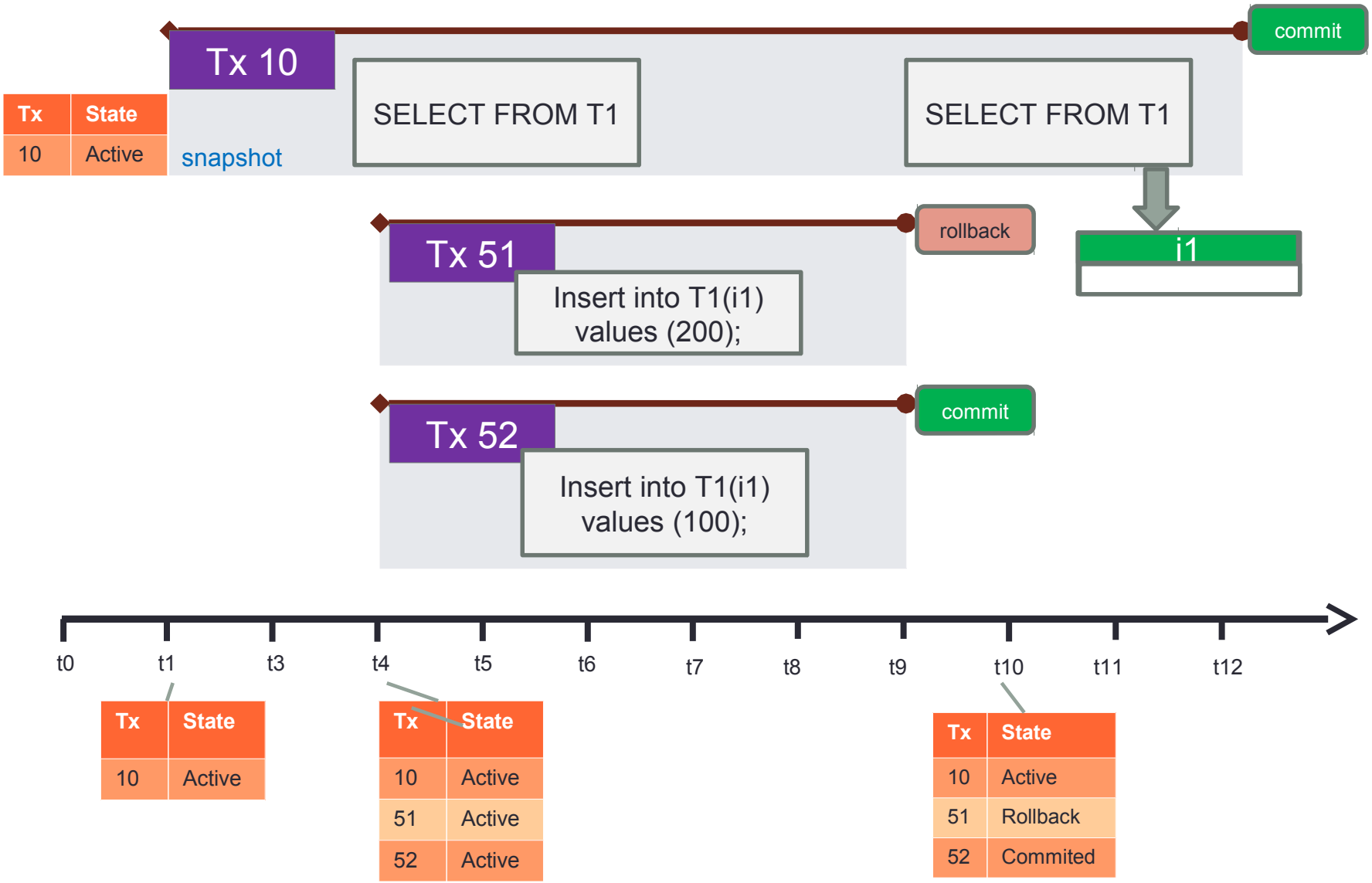


| Tx | State |
|---|---|
| 10 | Active |

| Tx | State |
|---|---|
| 10 | Active |
| 15 | Active |

| Tx | State |
|---|---|
| 10 | Active |
| 15 | Commited |

# TIP for snapshot




| Tx | State |
|---|---|
| 10 | Active |

| Tx | State |
|---|---|
| 10 | Active |
| 51 | Active |
| 52 | Active |

| Tx | State |
|---|---|
| 10 | Active |
| 51 | Rollback |
| 52 | Commited |

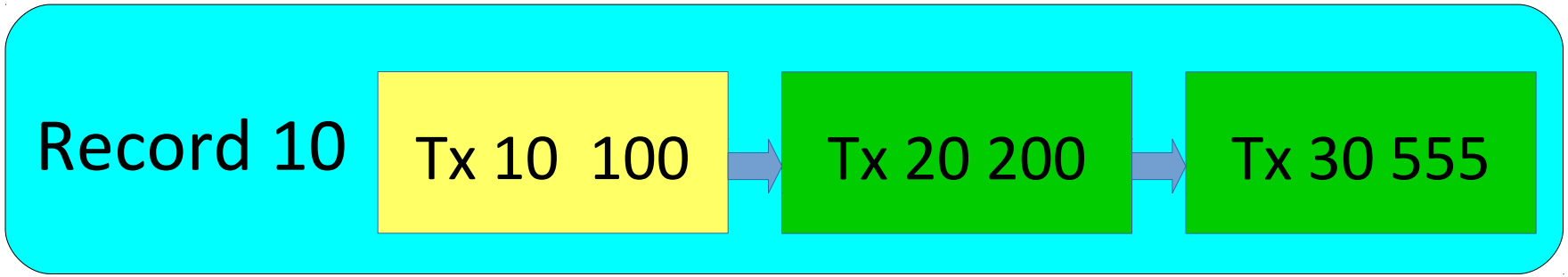
# Each transaction can see:

- Own created records and versions
  - Insert, Update, Delete
- If it is **Read Committed**, it can see every changes that was made by committed transactions, because it looks into global TIP
- If it is **Snapshot**, it can see own changes and record versions commited to the moment of its start, because it looks into it's own copy of TIP

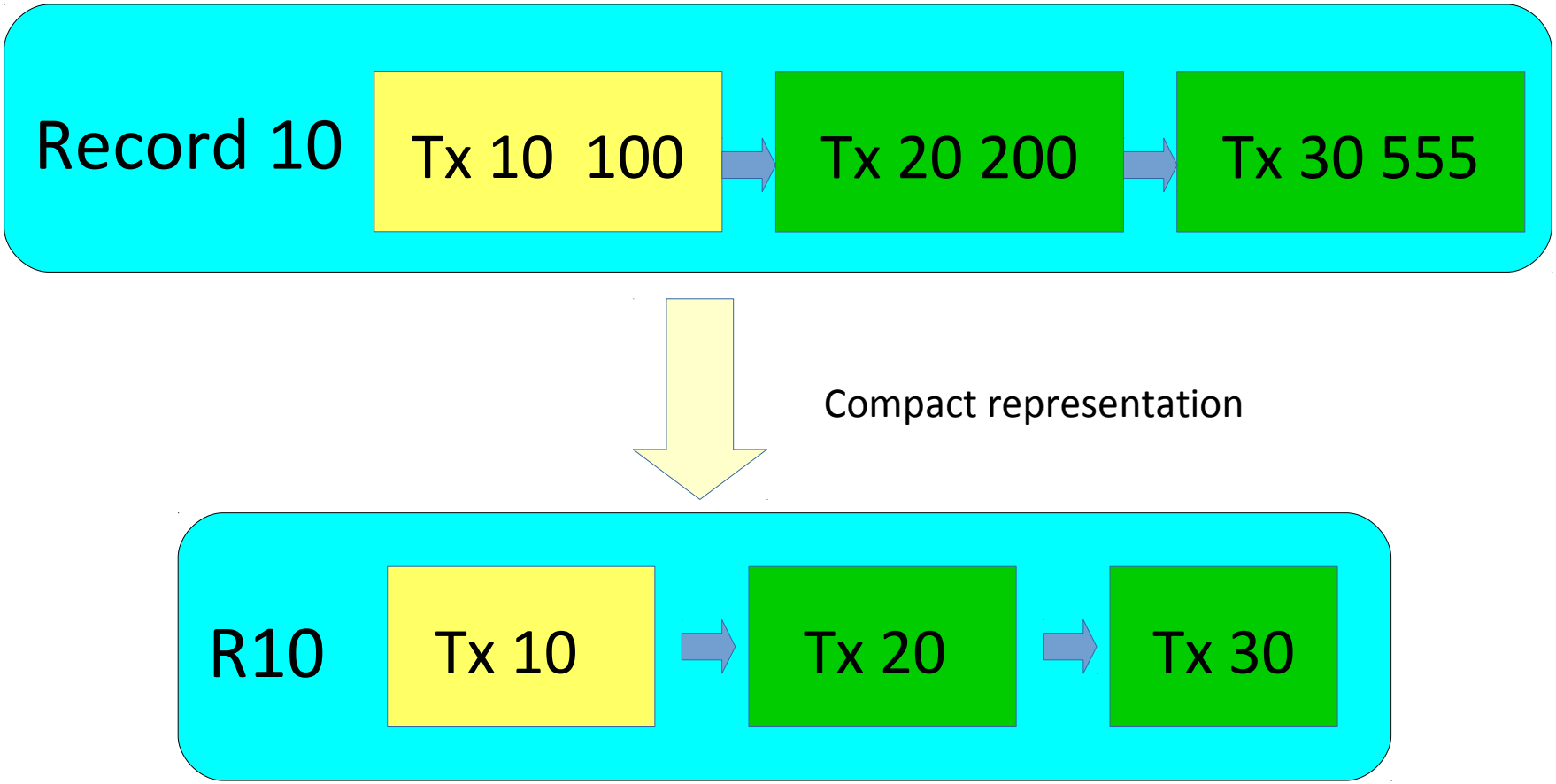
# Record versions visibility

# How we will present about records

Each record can have versions, created by different transactions



# How we will present about records



# 3 rules of record visibilty

1) For each snapshot transaction engine maintains stable view of database

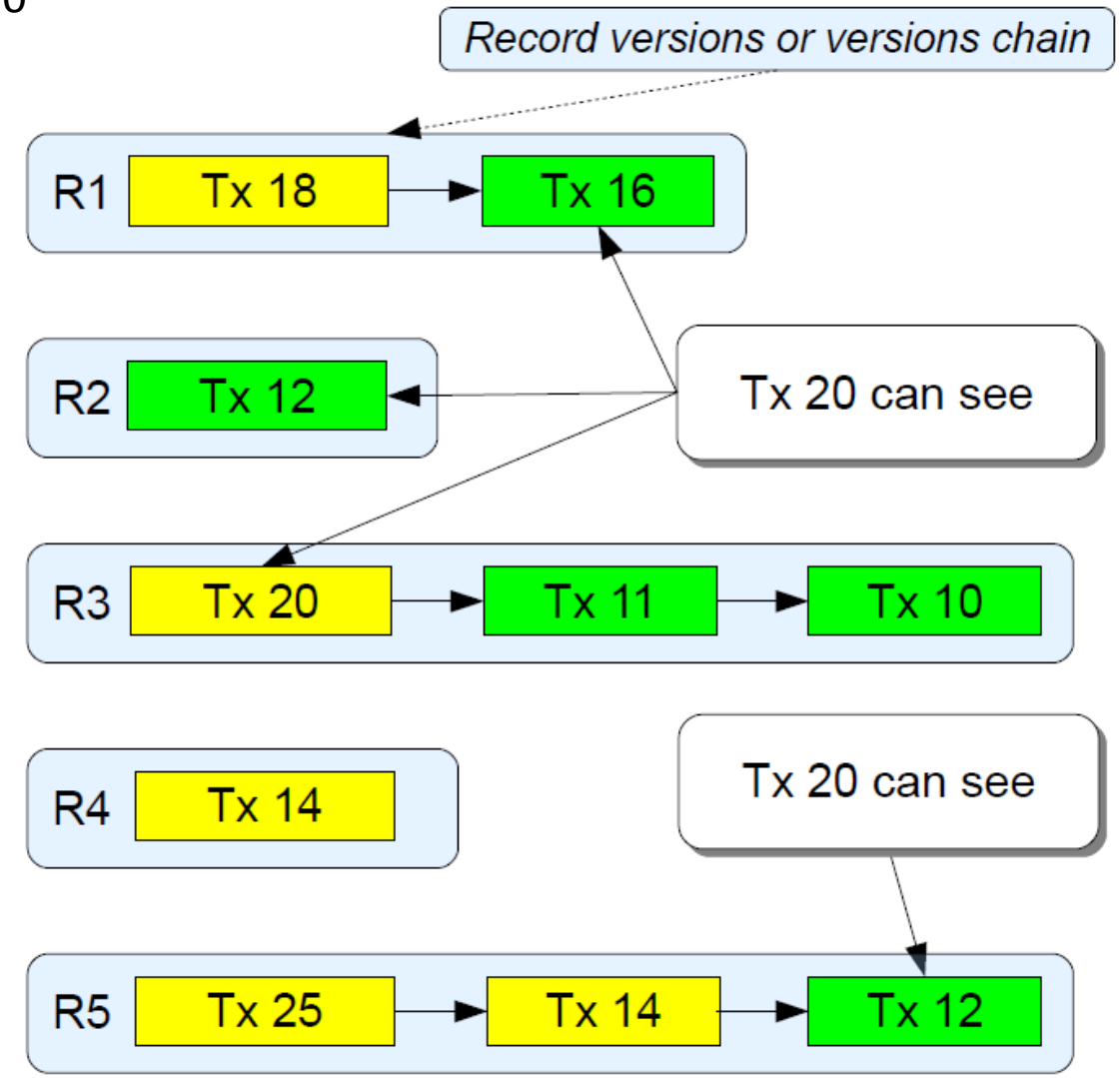
2) Transaction can not see record versions created by another active transaction

3) Transaction should walk backversions chain looking for commited backversion

# Ex: record versions visibility for Tx20

Snapshot isolation, copy of TIP for Tx20

| TIP contents for Tx 20 | |
|---|---|
| Tx № | Tx state |
| ... | committed |
| 11 | committed |
| 12 | committed |
| 13 | committed |
| 14 | active |
| 15 | committed |
| 16 | committed |
| 17 | rolled back |
| 18 | active |
| 19 | committed |
| 20 | active |
| ... | active |



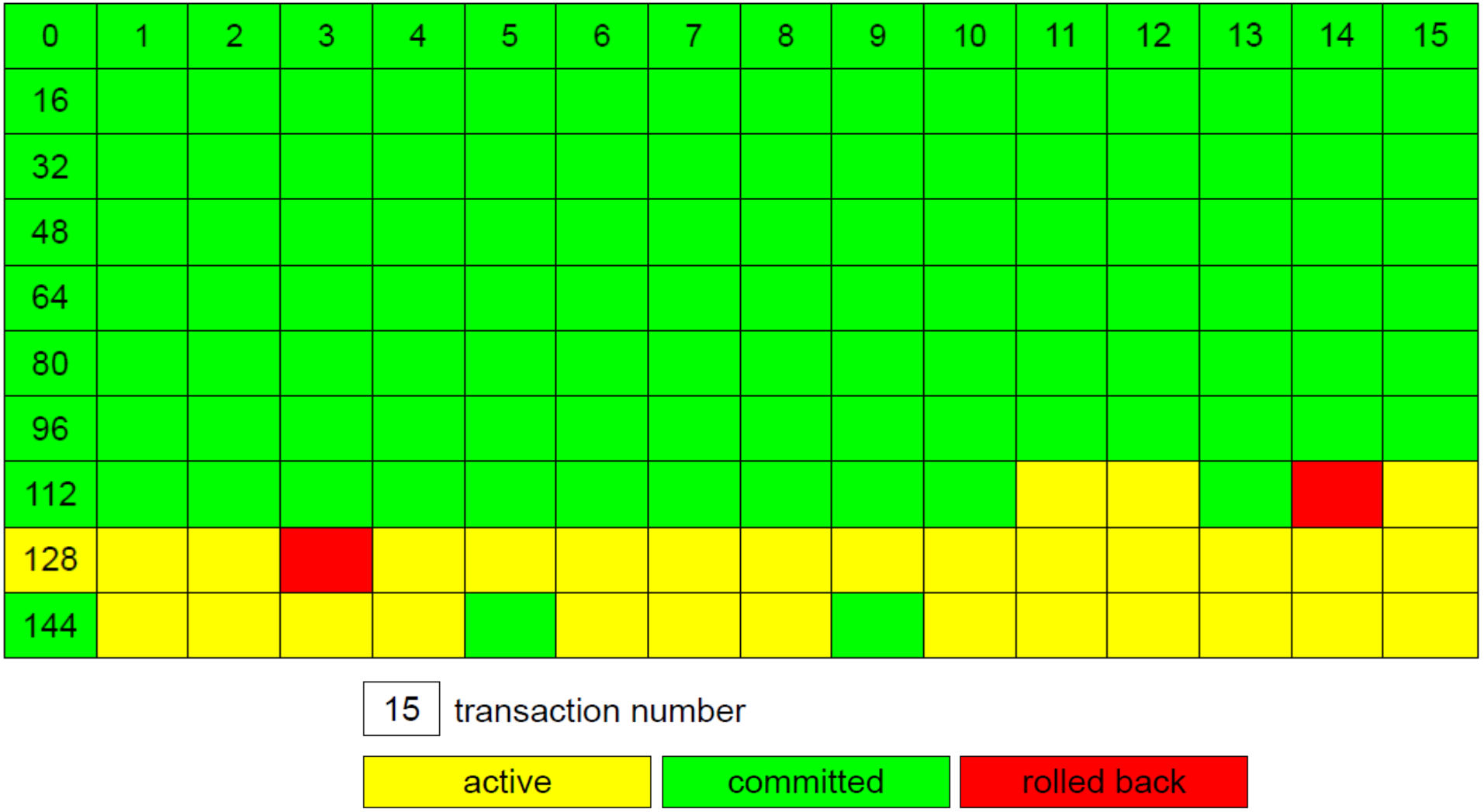
- In order to figure out which record version is visible, every transaction **must read** TIP
- TIP can contain up to 2 Billion transactions
- *So each transaction should read up to 2 billions of transactions! - Damn, that's why Firebird is slow!* (it's a joke)

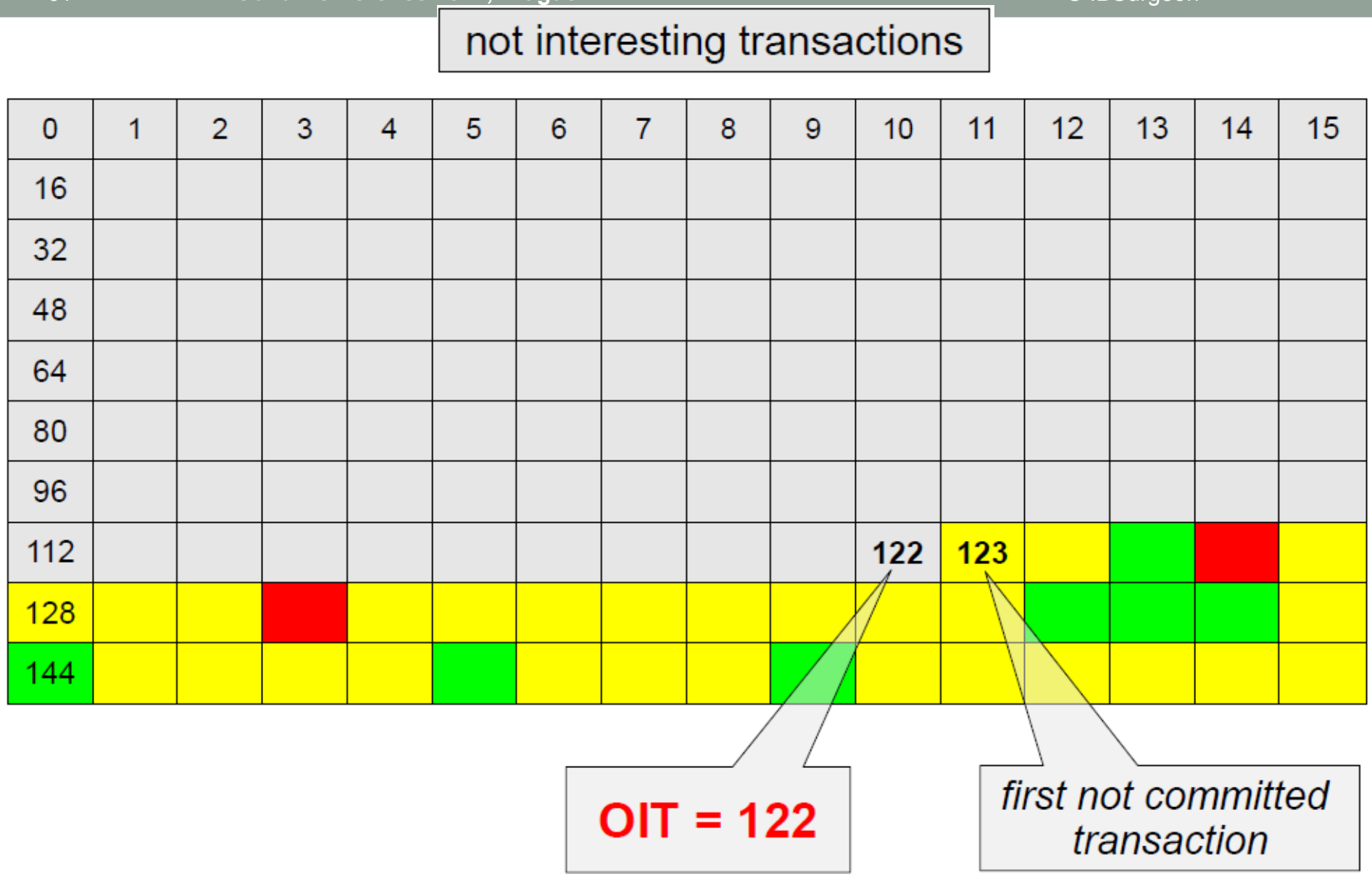
# TIP (example)

We need a way to **separate** old, not interesting transactions from currently active part of TIP

- For this purpose engine maintains ***Oldest Interesting Transaction*** marker, or ***OIT***

# TIP (example)


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16
32
48
64
80
96
112
128
144
15 transaction number
active
committed
rolled back


not interesting transactions
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
32
48
64
80
96
112
128
144
122
123
OIT = 122
first not committed transaction

# Transaction markers

```
firebird>gstat -h A.FDB
Database header page information:
Flags 0
Generation 6
System Change Number 0
Page size 4096
ODS version 12.0
Oldest transaction 1
Oldest active 2
Oldest snapshot 2
Next transaction 3
Sequence number 0
Next attachment ID 3
```

# 4 markers

- Transaction markers are key characterstics of TIP and transaction mechanism
  - Let's see what they mean and how they evaluated:
    - NEXT — next transaction
    - OAT — Oldest Active
    - OST — Oldest Snapshot
    - OIT — Oldest Interesting

# NEXT

- NEXT is the simplest — it's the most recent transaction
- NEXT number is written on header page

# OAT - Oldest Active Transaction

OAT is the first transaction in TIP which state is "active"

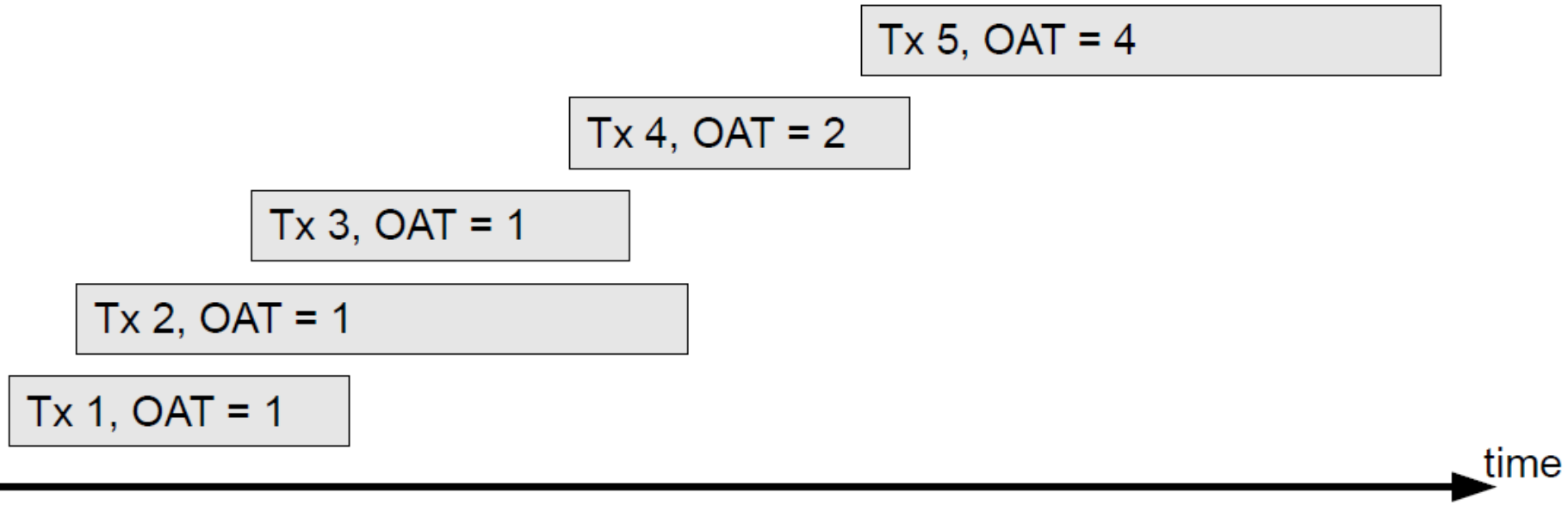
Evaluation:

- Scan TIP starting from current OAT value looking for "active" transaction
- Save found value in transaction's lock data
- Save found value as new OAT marker

**OAT is really an oldest active transaction**

# OAT evaluation example

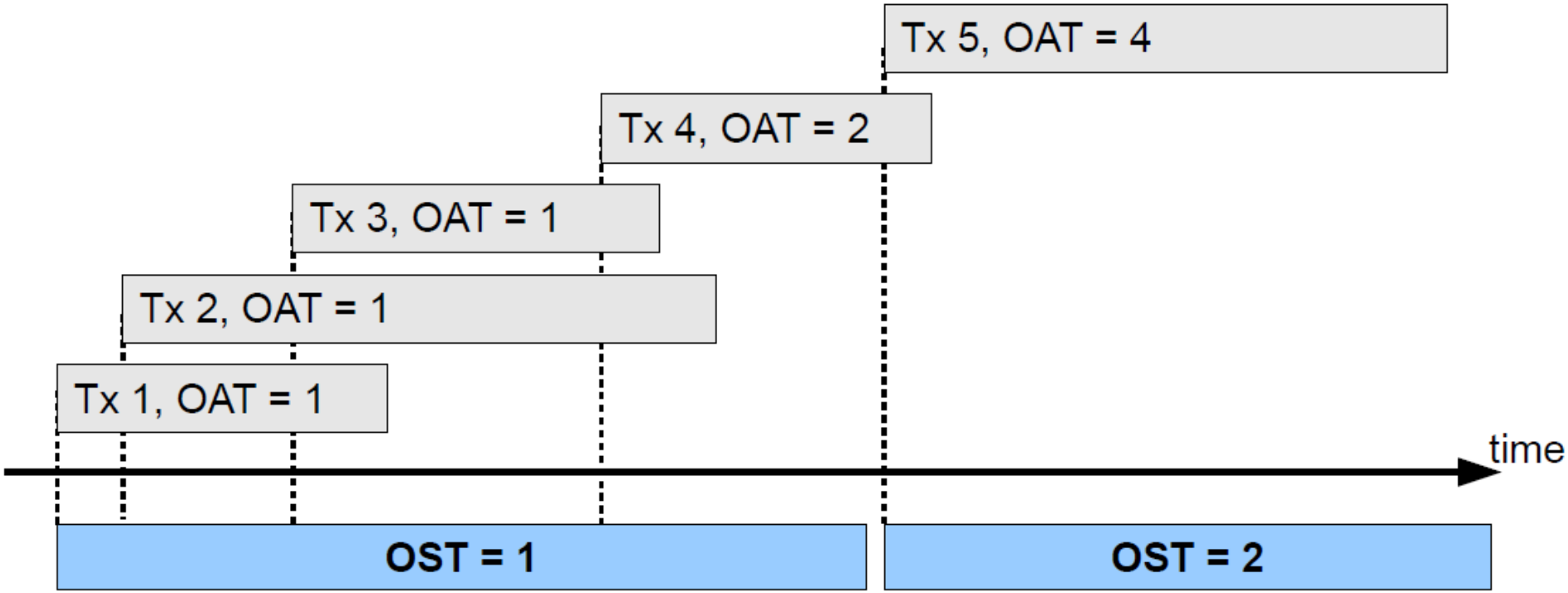
- Sample of transactions flow and evaluation of OAT



# Problems indicated by OAT

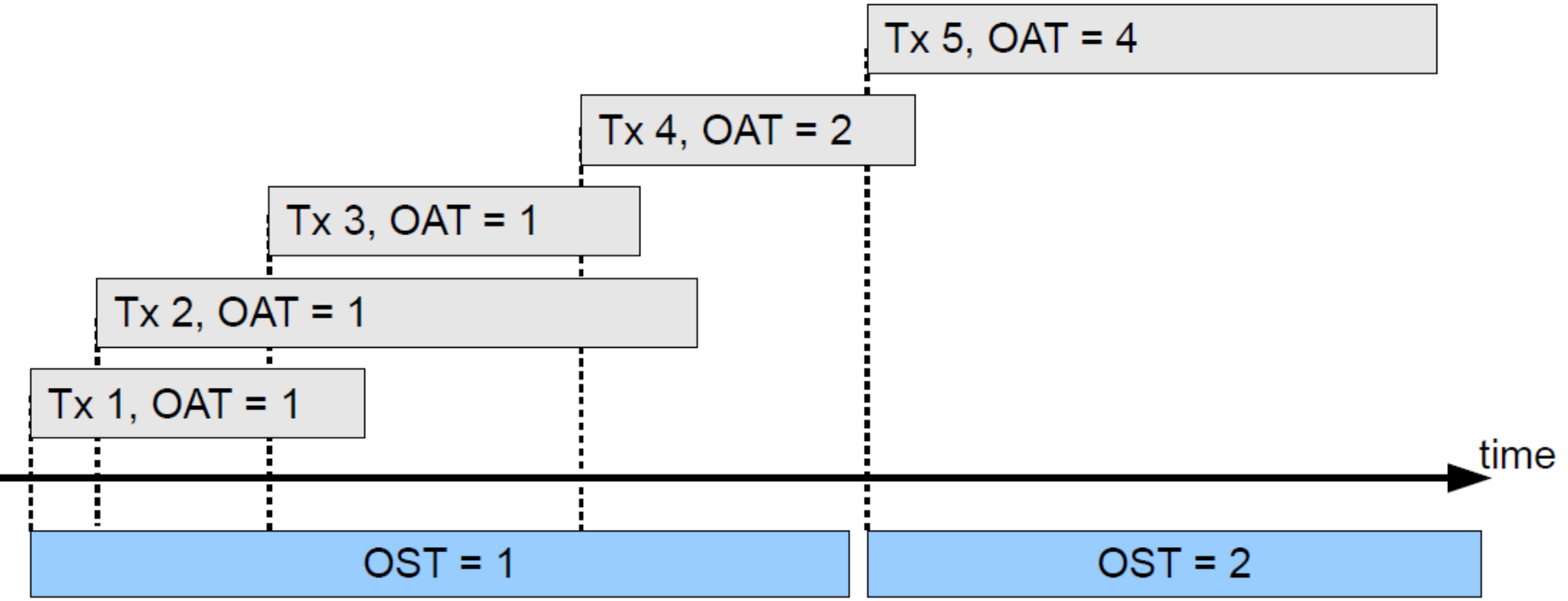
- Where to look?
  - NEXT — OAT > (number of connections * number of transaction)
- What it means?
  - Long running transaction which makes Firebird to think that record versions are still needed

- Oldest Snapshot Transaction (OST) marker is the value of the OAT recorded when oldest of currently active transactions was started
- Get min value of stored in transactions lock's data
- Save found value as new OST marker



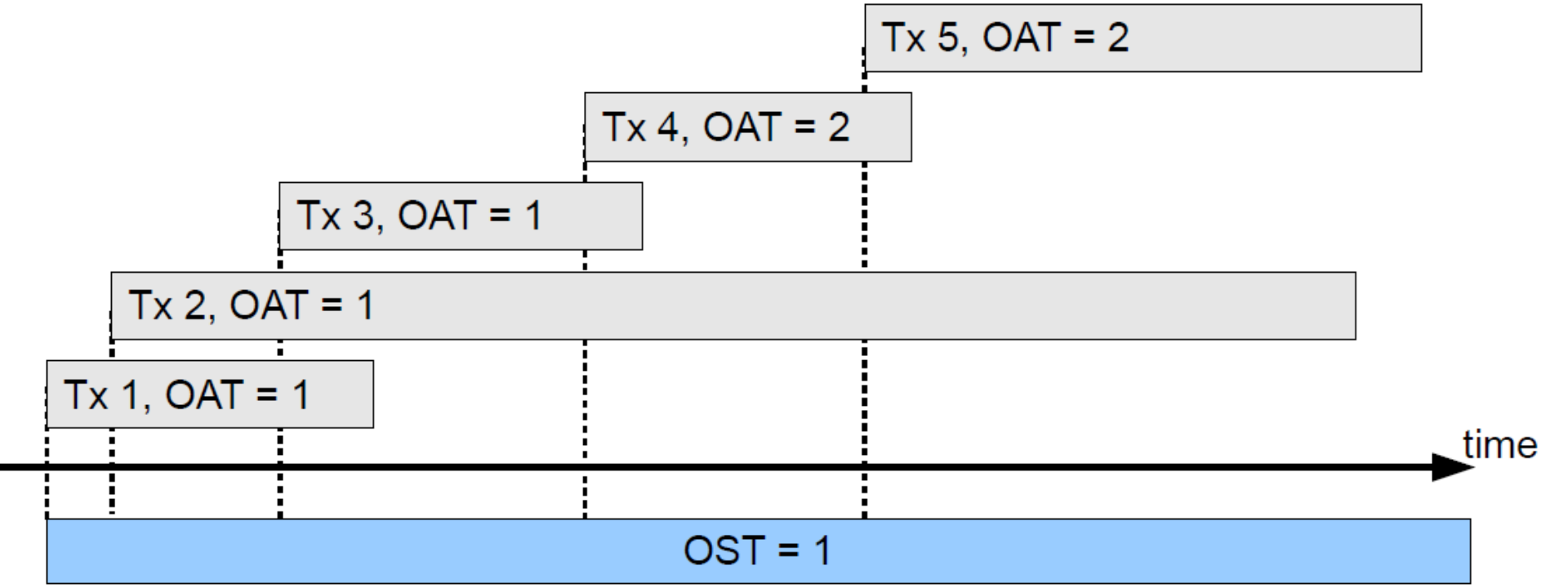
- Oldest Snapshot Transaction (OST) marker is the value of the OAT when oldest of currently active transactions was started

OST value often is not an alive transaction



- OST marker defines a garbage collection threshold: records, created by transactions >= OST can not be garbage collected

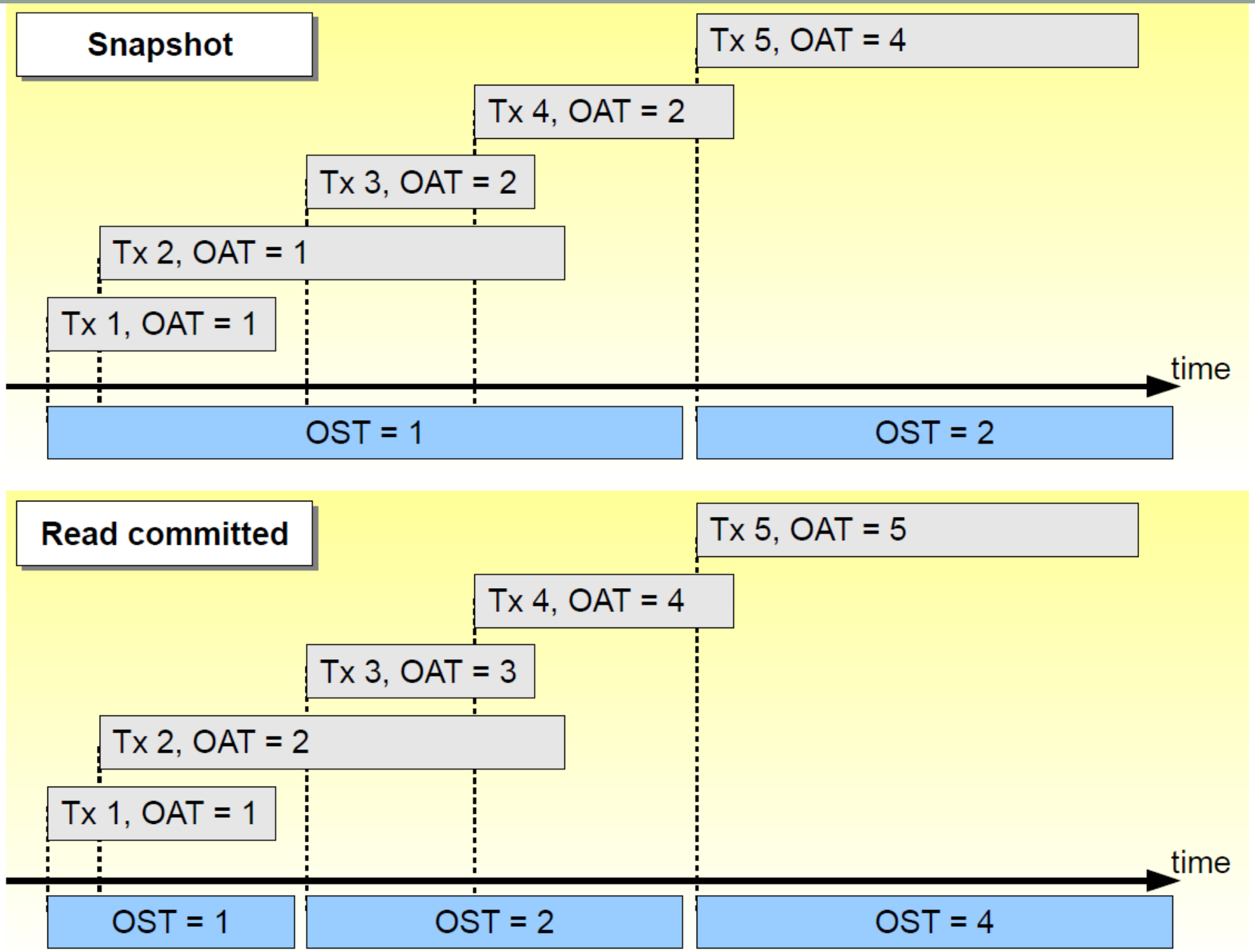
**Long running transactions will "stuck" OST and delay GC**

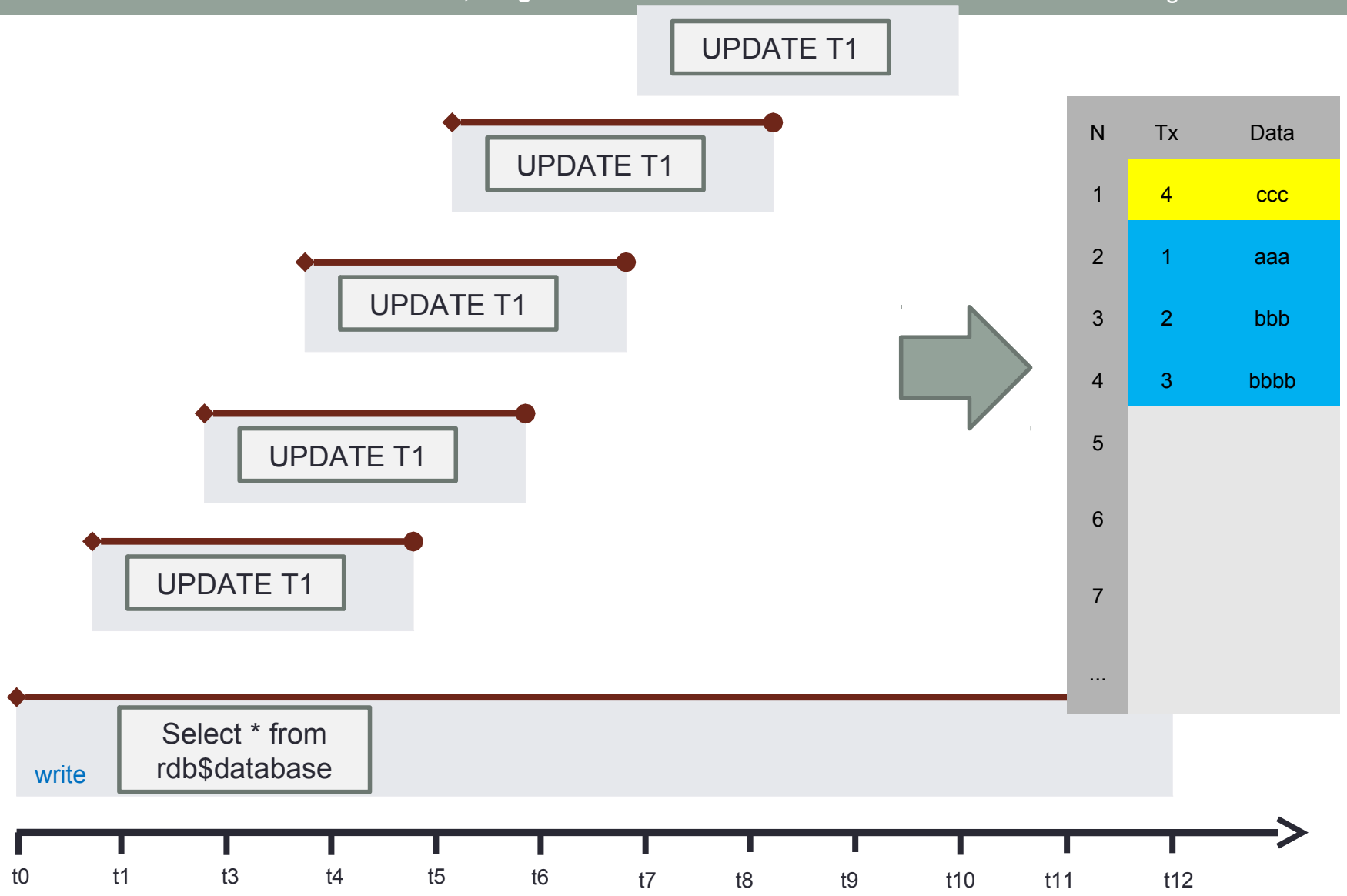


# OST and Read Commited transactions

- Read Committed transaction don't require stable snapshot of database
- Oldest Active value for Read Committed transaction is an own number of such transaction
- Read Committed Readonly transaction can't create record versions, is pre-committed at start and have no impact on OST

Read Committed Readonly transaction could run forever and do not delay garbage collection


Snapshot
Tx 5, OAT = 4
Tx 4, OAT = 2
Tx 3, OAT = 2
Tx 2, OAT = 1
Tx 1, OAT = 1
time
OST = 1
OST = 2
Read committed
Tx 5, OAT = 5
Tx 4, OAT = 4
Tx 3, OAT = 3
Tx 2, OAT = 2
Tx 1, OAT = 1
time
OST = 1
OST = 2
OST = 4




The longer transaction lasts, the higher chance to create potentially useless (potential garbage) versions

# Problems indicated by OST

Where to look

(OST-OIT) > sweep interval

What it means

- Autosweep does not work (if sweep interval >0)
- Some records need garbage collection

# Problems caused by long running transactions

- Direct
  - Loss of performance due to more record versions: i.e., queries become slower
    - More indexed reads
    - More data page reads
      - 1.5mln versions ~30mb per record
- Indirect
  - After transaction's end its versions become garbage, and garbage collection mechanism tries to gather it
  - Due to long transaction OST stuck, so autosweep (if it is not disabled) tries to start at unpredictable moment (and ends without success)
    - GC and sweep can consume a lot of resources
    - Unpredictable moment can occur at high load time

# Oldest Interesting Transaction

- Oldest Interesting Transaction (OIT) marker is necessary to know to separate old not active part of TIP from currently used active part
- OIT points before a first transaction in TIP which state is not committed
- Evaluation:
  - Scan TIP starting from current OIT value looking for first not committed transaction

# TIP size

- TIP to be copied is NEXT - OIT
- Size of active part of the TIP in bytes is **(Next – OIT) / 4**

| | |
|---|---|
| Page size | 4096 |
| Forced Write | ON |
| Dialect | 3 |
| OnDiskStructure | 11.2 |
| Attributes | force write |
| Sweep interval | 20000 |
| Oldest transaction | 2147483644 |
| Oldest snapshot | 2147483645 |
| Oldest active | 2147483645 |
| Next transaction | 2147483646 |
| Sweep gap (active - oldest) | 1 |
| TIP size | 131073 pages, 524292 kilobytes |

| Database info | |
|---|---|
| Database name | |
| Creation date | 05.06.2003 10:02:19 |
| Statistics date | 31.08.2006 18:11:32 |
| Page size | 8192 |
| Forced Write | ON |
| Dialect | 1 |
| OnDiskStructure | 10.0 |
| Attributes | force write |
| Sweep interval | 0 |
| Oldest transaction | 534249471 |
| Oldest snapshot | 429490176 |
| Oldest active | 534249472 |
| Next transaction | 534249481 |
| Sweep gap (snapshot - oldest) | -104759295 |
| TIP size | 16305 pages, 130440 kilobytes |
| Snapshot TIP size | 10 transactions, 8 kilobytes |
| Active transactions | 9, 0% of daily average |
| Transactions per day | 451224, for 1184 days |

# Problems indicated by OIT

Where to look

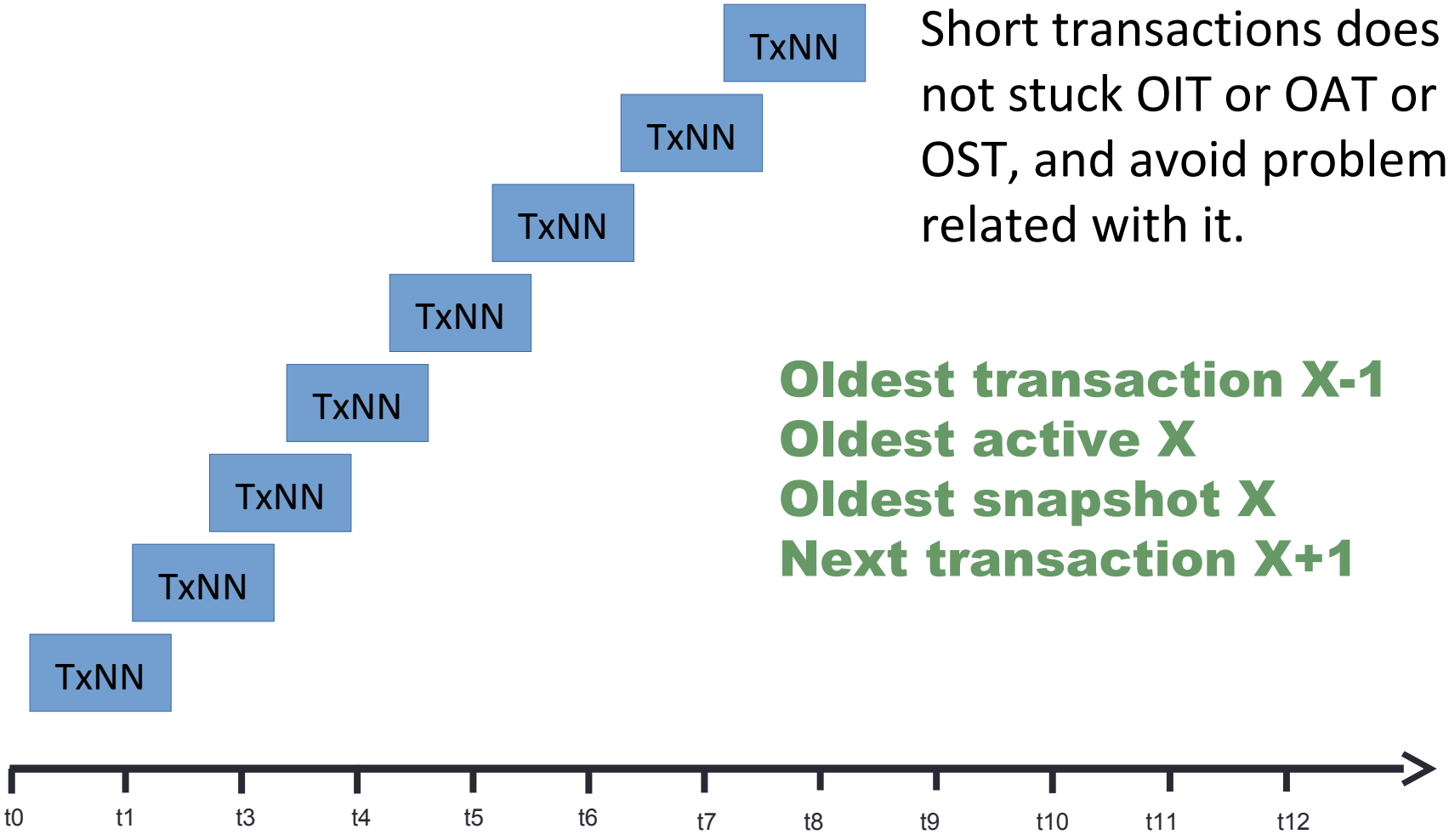
OIT- OST

Problem

Big size of TIP — Global, and, specifically copies of TIP for snapshots

| | |
|---|---|
| Creation date | 05.06.2003 10:02:19 |
| Statistics date | 22.06.2004 20:55:32 |
| Page size | 8192 |
| Forced Write | ON |
| Dialect | 1 |
| OnDiskStructure | 10.0 |
| Attributes | force write |
| Sweep interval | 0 |
| Oldest transaction | 839568 |
| Oldest snapshot | 112430561 |
| Oldest active | 112430625 |
| Next transaction | 112431441 |
| Sweep gap (snapshot - oldest) | 111590993 |
| TIP size | 3432 pages, 27457 kilobytes |
| Snapshot TIP size | 111591873 transactions, 27252 kilobytes |
| Active transactions | 816, 0% of daily average |
| Transactions per day | 292790, for 384 days |

# Ideal transactions flow




Short transactions does not stuck OIT or OAT or OST, and avoid problems related with it.

**Oldest transaction X-1**
**Oldest active X**
**Oldest snapshot X**
**Next transaction X+1**

# Summary

- Make write (for INSERT/UPDATE/DELETE) transactions as short as possible
- Use Read Commited Read-Only transactions for SELECTs
- Troubleshoot transactions with HQbird toolset: MonLogger, IBAnalyst and FBDataGuard

# Thank you!

- Questions? support@ib-aid.com