Inside Firebird transactions

Vlad Khorsun,
Firebird developers team

Firebird Conference 2019

Berlin, 17-19 October

IBPhoenix

YOUR PREMIER SOURCE OF FIREBIRD SUPPORT

IBSurgeon

MOSCOW EXCHANGE

Fast Reports

Reporting must be fast!

IB Expert

REDSOFT

What we will speak about:

- Transaction start
  - transactions markers
- COMMIT
- ROLLBACK
- Savepoints and undo log
- Retaining transaction context
- Resource usage by transaction

Transaction start actions:

- Get own unique number
- Create own copy of TIP
  - snapshot transactions
- Update shared TIP cache
  - read-committed transactions
- Evaluate markers OAT, OST and OIT
- Run auto-sweep if necessary

Transactions 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
```

Get own unique number

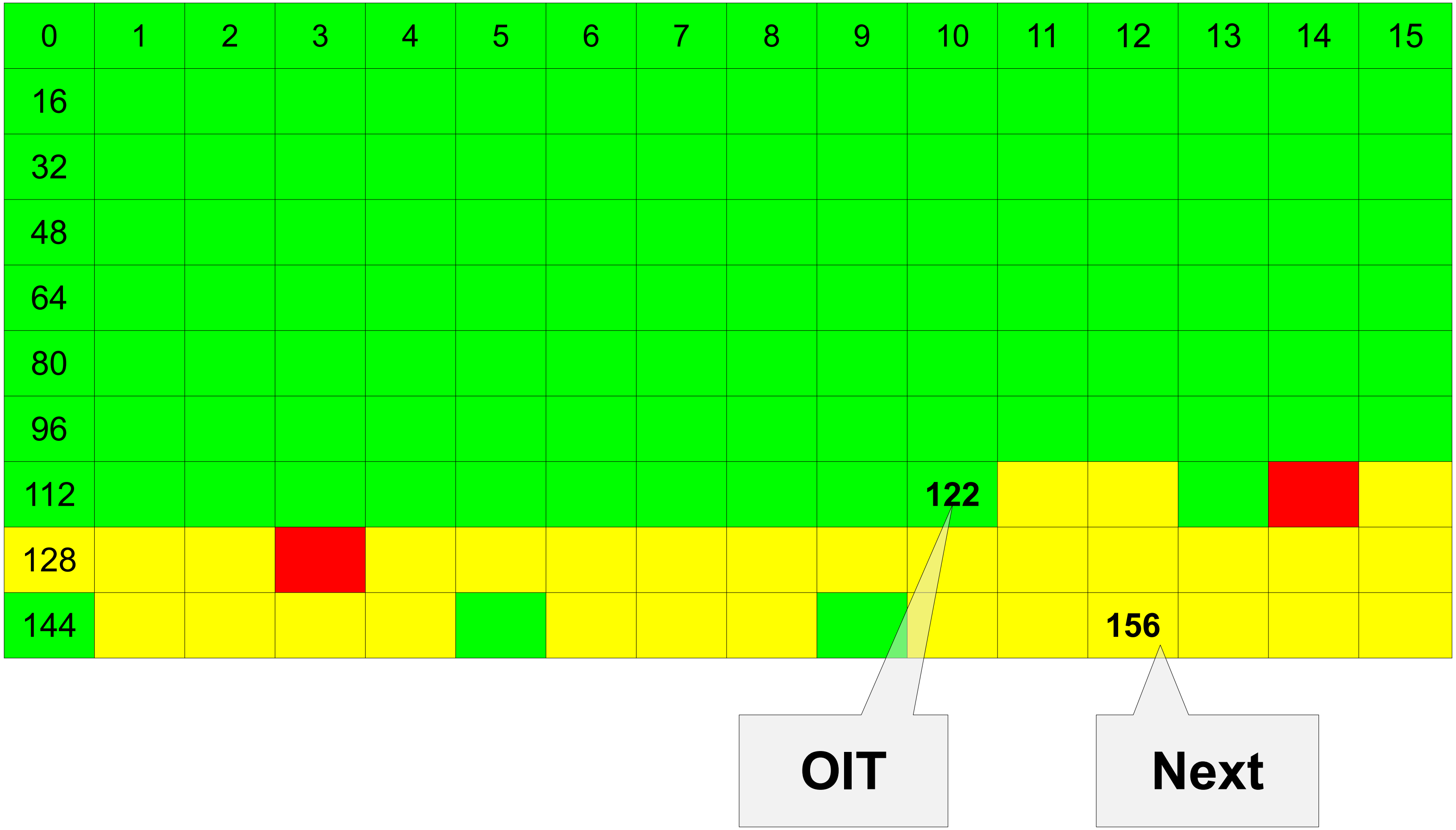
- Lock Header page for write
- Read and increment Next transaction marker
- Extend TIP if necessary
- Write new value of Next back to the Header page
- Release Header page

Create own copy of transactions inventory (TIP)

- Snapshot transactions uses its own copy of TIP
- Only active part of TIP contents is copied
- The low bound is current OIT value
  - All transactions below OIT considered committed
- The high bound is Next value
  - All transactions above Next considered active

Size of active part of the TIP in bytes is
(Next – OIT) / 4
It affects memory usage !

Create own copy of transactions inventory (TIP)



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16 🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
32 🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
48 🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
64 🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
80 🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
96 🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
112 🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	122 🟩	🟨	🟨	🟩	🟥	🟨
128 🟨	🟨	🟨	🟥	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
144 🟩	🟨	🟨	🟨	🟨	🟩	🟨	🟨	🟨	🟩	🟨	🟨	156 🟨	🟨	🟨	🟨

Active part of TIP contains transactions 123 – 156
Its size is 10 bytes

Update shared cache of transactions inventory

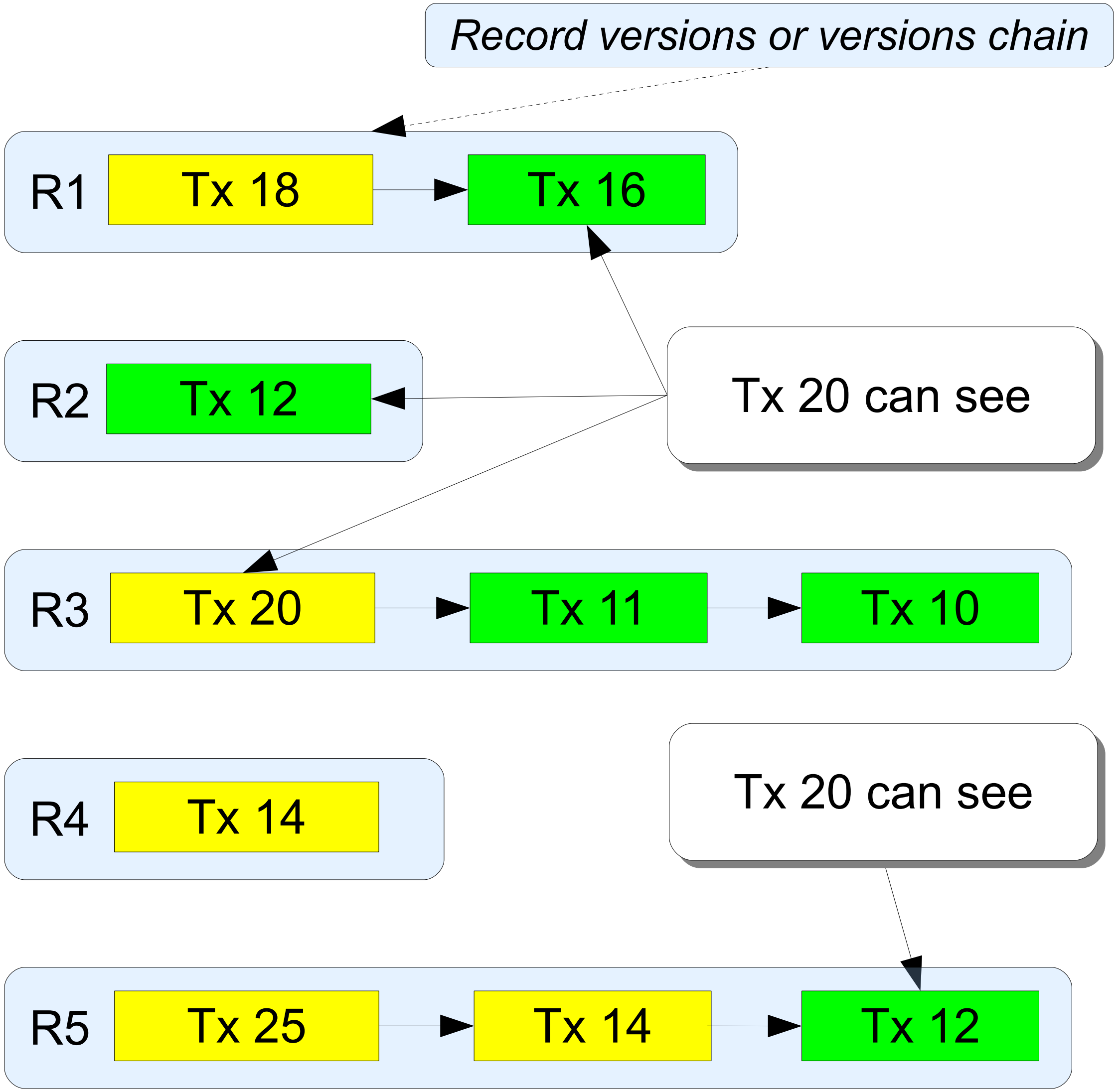
- Read-committed transactions uses common shared cache of transactions inventory (TIP cache):
  - Committed and rolled back transactions can't change its state, no verification is required
  - Active transaction is verified additionally:
    - If transaction is alive – it is really active
    - Else fetch state from disk and update TIP cache

Read-committed transaction could be less efficient when reads many records updated by active transactions

Record versions visibility

- For each snapshot transaction engine maintains stable view of database
- Transaction can not see record versions created by another active transaction
- Transaction should walk backversions chain looking for committed backversion

Record versions visibility



Record versions or versions chain

R1 Tx 18 → Tx 16

R2 Tx 12

R3 Tx 20 → Tx 11 → Tx 10

R4 Tx 14

R5 Tx 25 → Tx 14 → Tx 12

Tx 20 can see

Tx 20 can see

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

Transaction's private snapshot of database

- Engine should not remove backversions if primary record version I see is active
- Records, which primary version is created by any active transaction I know, must be preserved for me
- Records, which primary version is created by transaction younger then oldest active transaction I know, must be preserved for me
- Oldest Active Transaction defines transaction's private snapshot

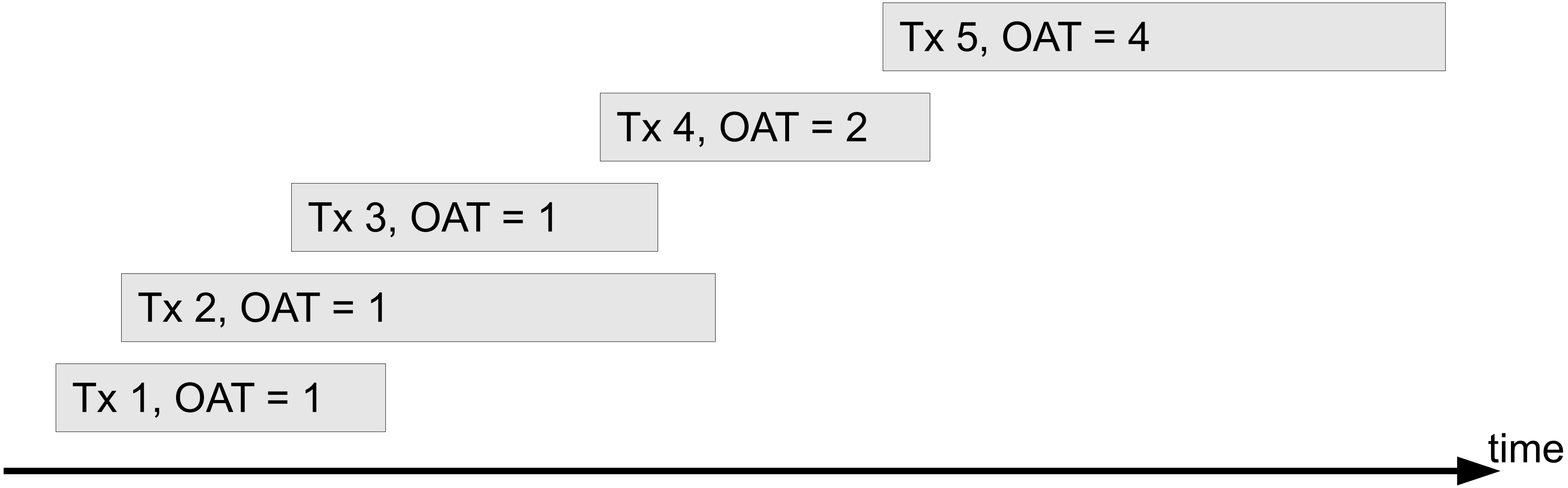
OAT - Oldest Active Transaction

- 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 - Oldest Active Transaction

- Sample of transactions flow and evaluation of OAT

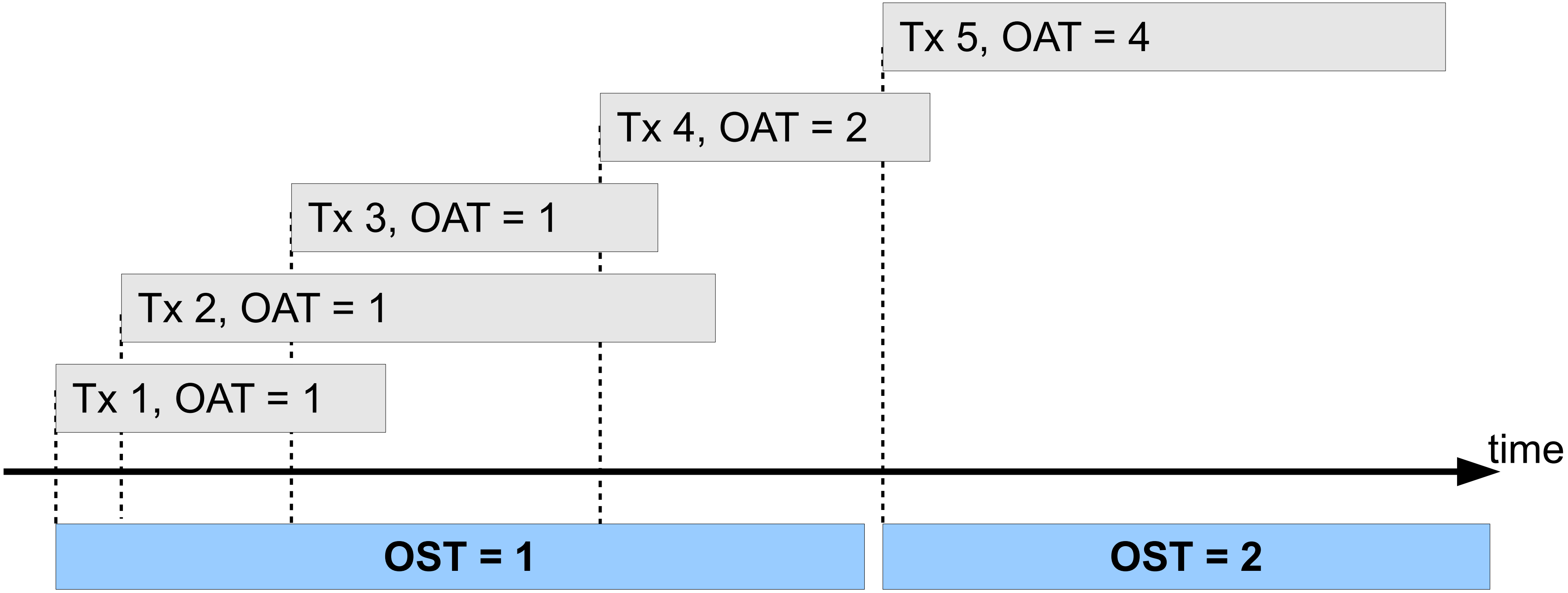


Oldest snapshot in database

- Engine maintains snapshots for every active snapshot transaction
- Snapshot of oldest of currently active transactions is an oldest snapshot in database

OST - Oldest Snapshot Transaction

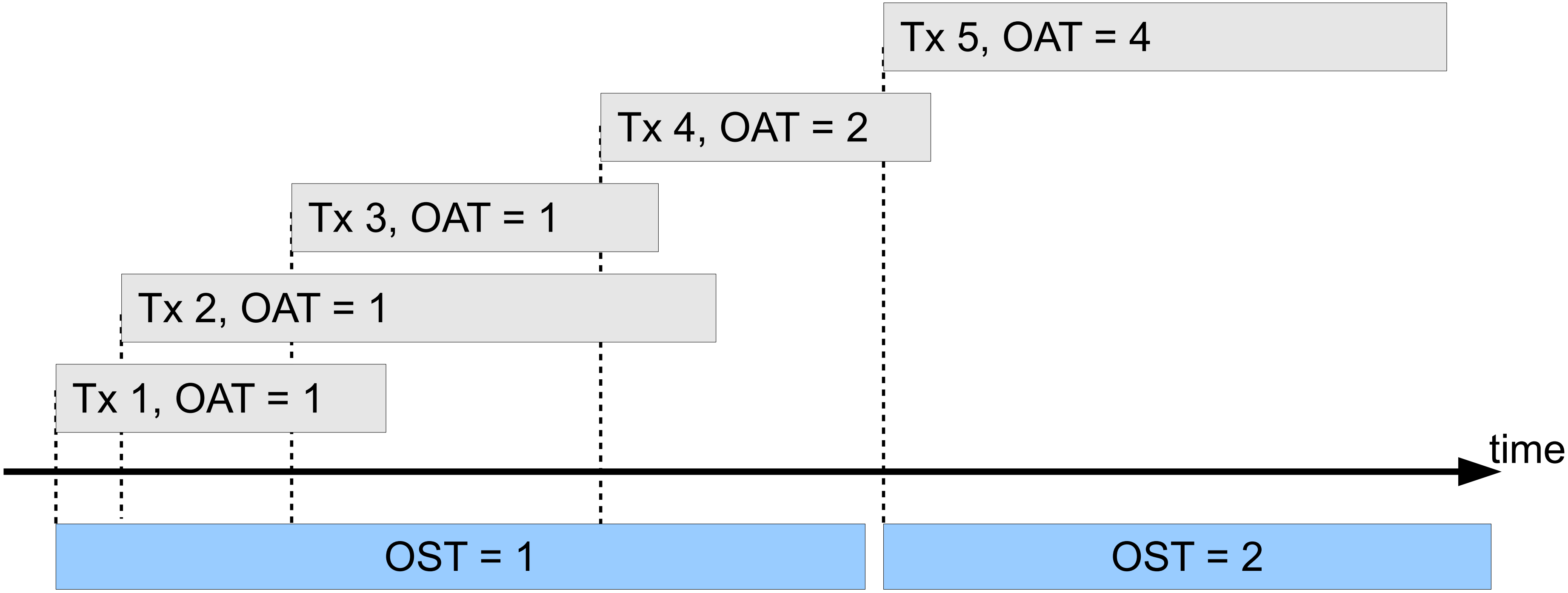
- Oldest Snapshot Transaction (OST) marker is the value of the OAT recorded when oldest of currently active transactions was started
- Get min value of stored in transactions lock's data
- Save found value as new OST marker



OST - Oldest Snapshot Transaction

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

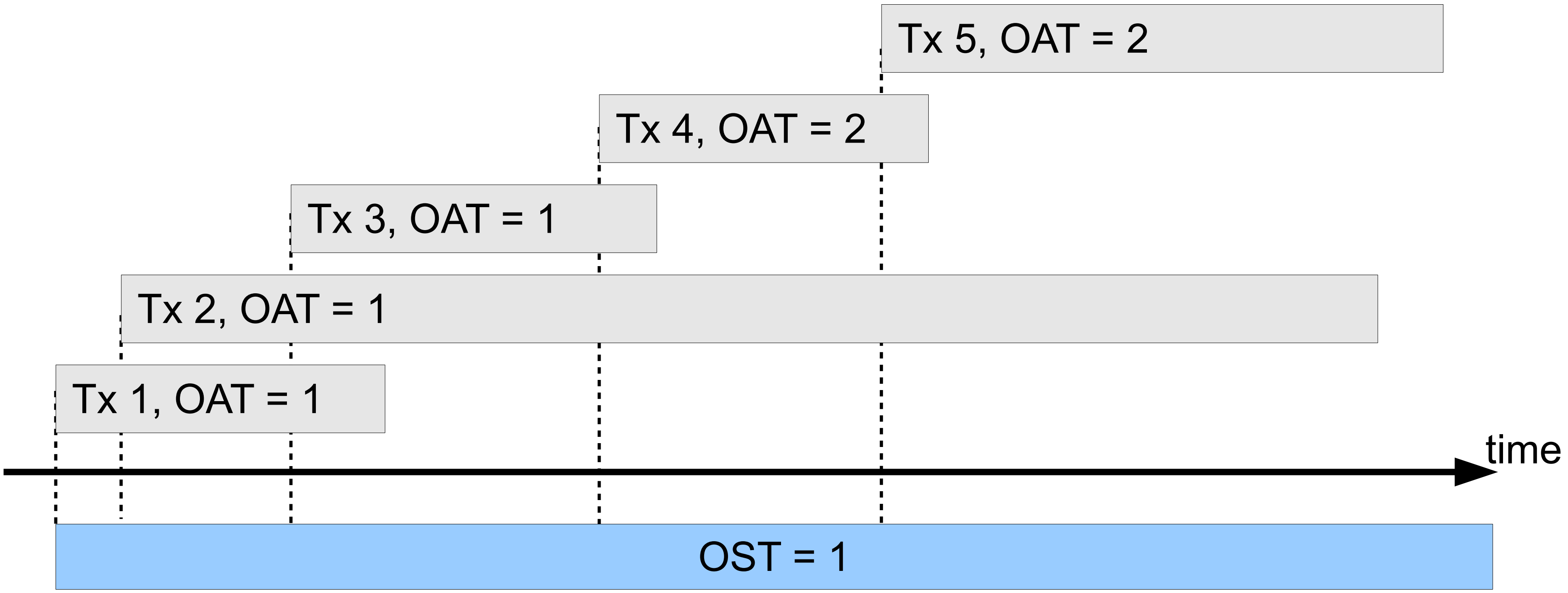
OST value often is not an alive transaction



OST - Oldest Snapshot 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

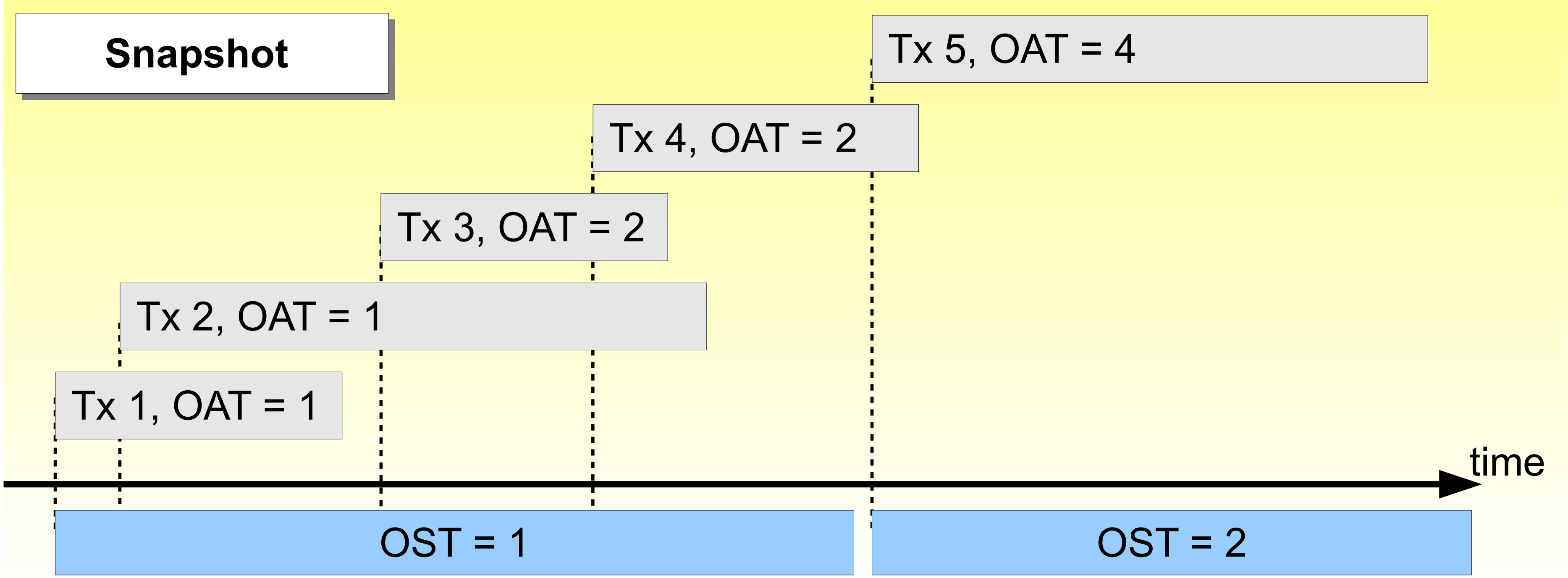


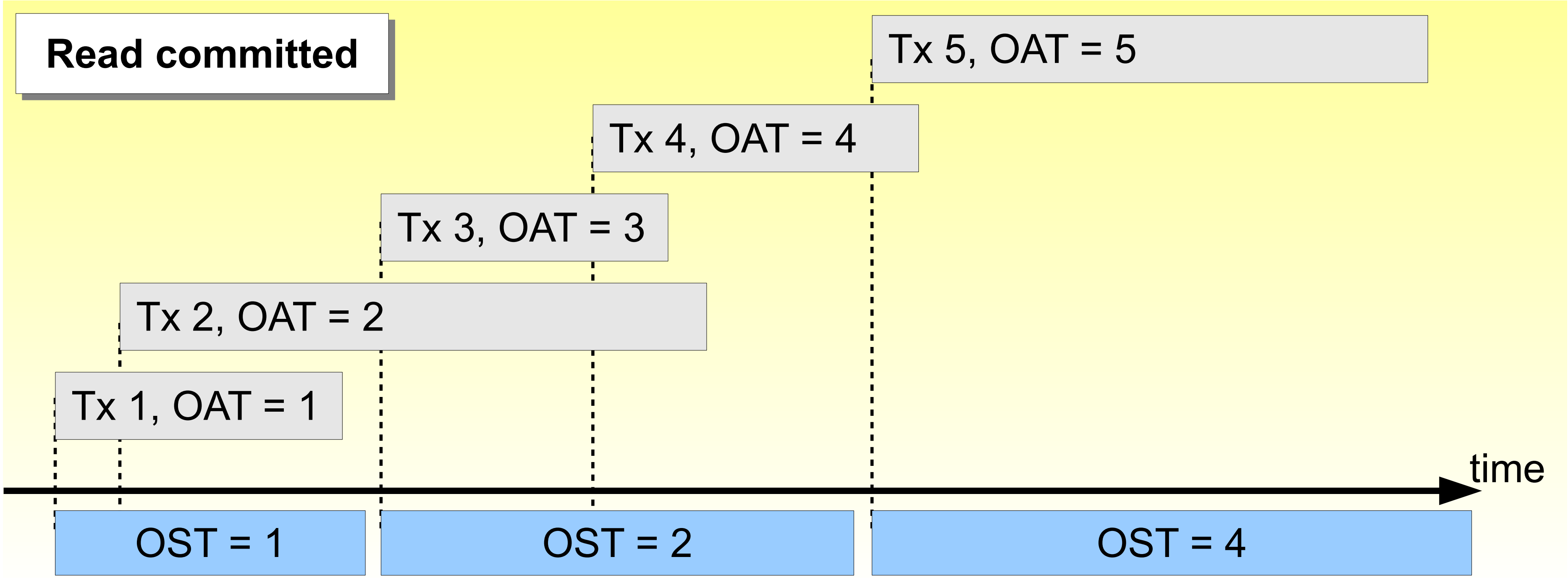
OST and Read Committed

- 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

Transaction start





OIT - 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

Transaction start: final actions

- Run sweep if necessary:
  - sweep_interval > 0, and
  - sweep_interval < OST – OIT
- Mark READ_COMMITTED READONLY transaction as committed in TIP
- Run ON TRANSACTION START triggers
  - For user transactions (including autonomous) only

Commit actions

- Run ON TRANSACTION COMMIT triggers
  - User and autonomous transactions
- Commit EXTERNAL transaction's
- Finish DDL work items
- Flush dirty pages to disk
- Set transaction state in TIP to committed
- Post EVENTS
- Release transaction and its resources

DDL statements execution

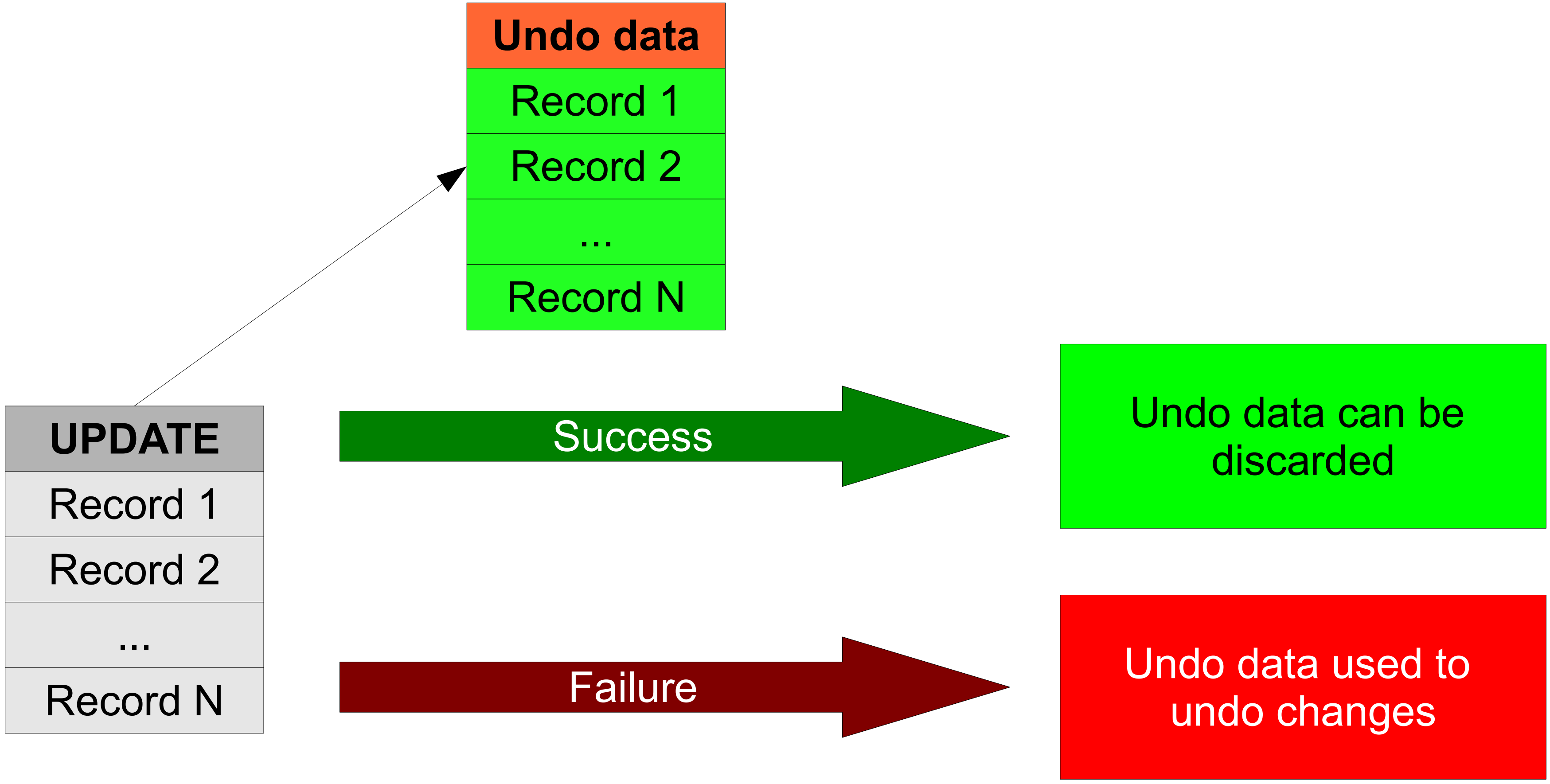
- User: *CREATE INDEX*
  - Engine:
    - Generate index name, if needed
    - Checks logical validity
    - *INSERT INTO RDB$INDICES*
    - *INSERT INTO RDB$INDEX_SEGMENTS*
- User: *COMMIT*
  - Engine:
    - Reads all metadata necessary to build index
    - Checks logical validity of metadata entered
    - Acquires Protected Read lock(s) for table(s)
    - Build an index B-Tree

Rollback actions

- Run ON TRANSACTION ROLLBACK triggers
  - User and autonomous transactions
- Rollback EXTERNAL transaction's
- If rollback is forced or there are many data to undo:
  - Set state in TIP to rolled back
- Not forced and no or few data to undo:
  - Undo changes
  - Write dirty pages to disk and flush OS file cache
  - Set state in TIP to committed
- Release transaction and its resources

Savepoints

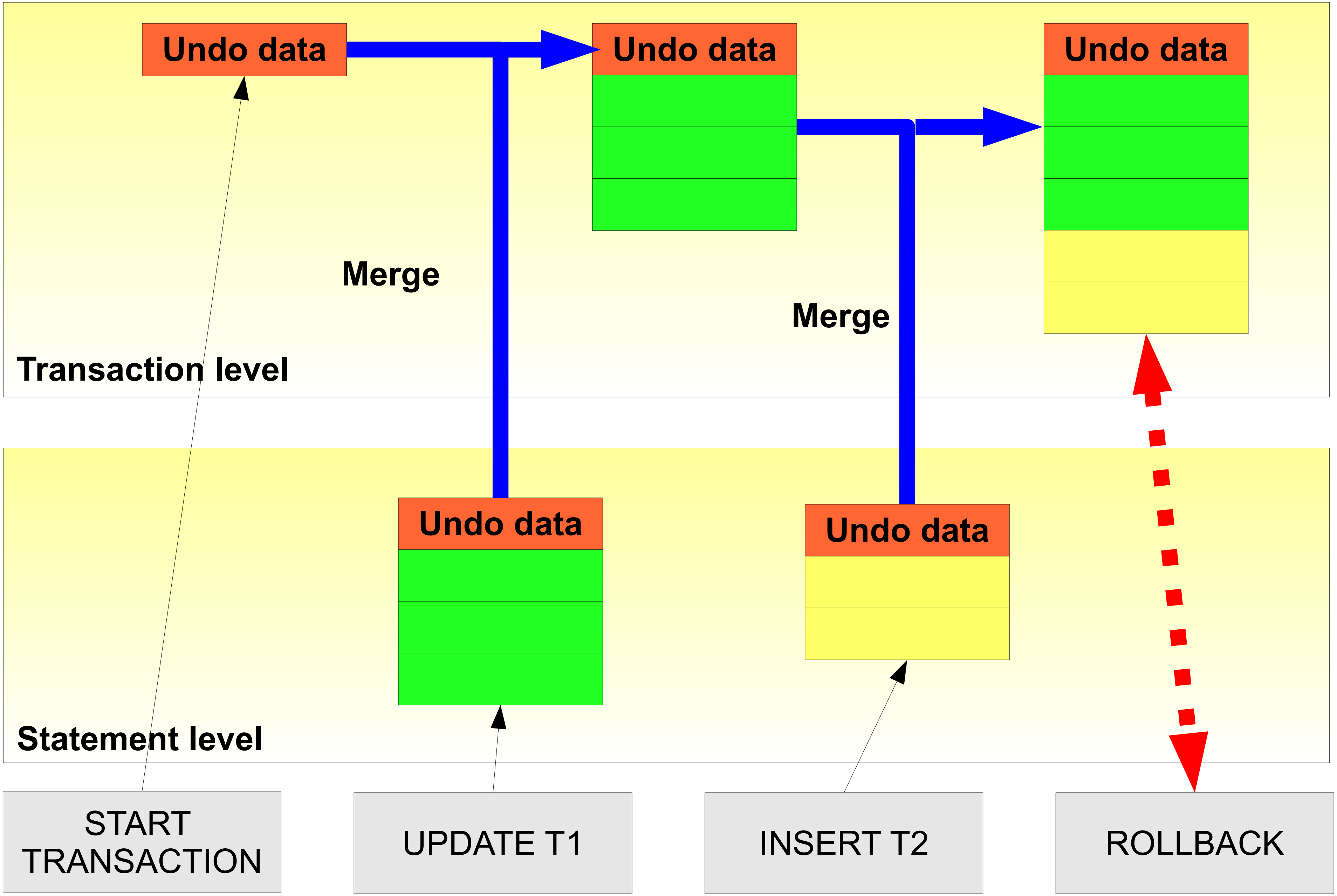
- Every statement is enclosed into own savepoint which contains data used to undo statement changes



Savepoints

- Group of statements could be enclosed into common savepoint:
  - BEGIN … END
- User also could set savepoints and rollback all work done after savepoint was set:
  - SAVEPOINT <name>
  - RELEASE SAVEPOINT <name>
  - ROLLBACK TO [SAVEPOINT] <name>
- Transaction also could have savepoint

Transaction Savepoint



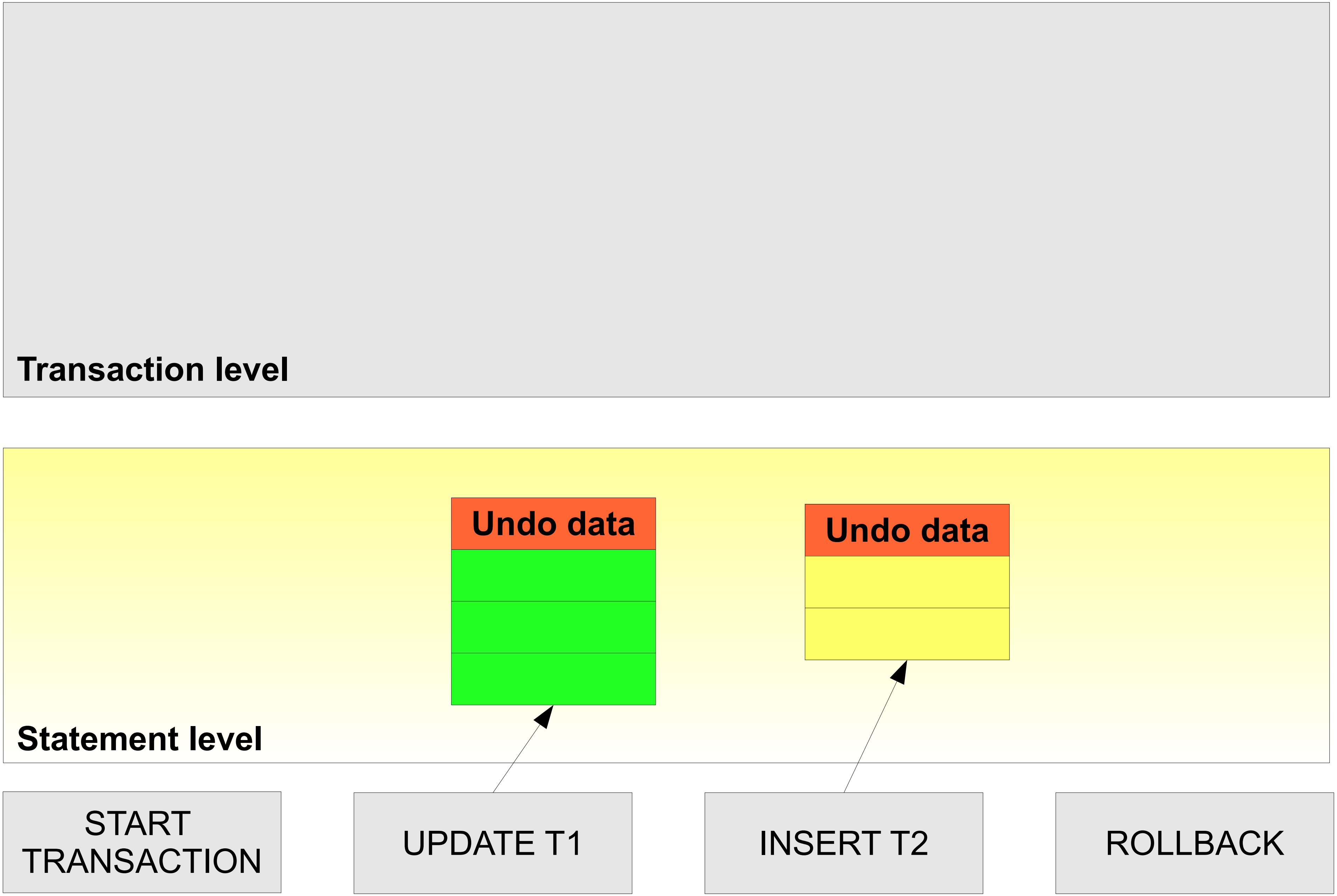
Rollback using undo log

- When undo log have reasonable small size it could be used to undo all changes in transaction:
  - Pluses
    - No garbage is left in database
    - Transaction state in TIP set to “committed”
    - OIT is not stuck
  - Minuses
    - Longer time of rollback
- Undo log reside in memory and overflows to disk
  - TempCacheLimit setting in firebird.conf

“No undo log” option

- *isc_tpb_no_auto_undo*
  - It does not cancel usage of undo log !
  - It just cancel accumulation of changes by different statements at transaction level
  - Allows to use less memory when transaction run more than one DML statement
  - Makes rollback using undo log impossible

No transaction savepoint



Rollback “via TIP”

- Used when engine process exits and there is no time for long actions (forced), or
- When there are too much work to undo, or
- There is no undo data at transaction level

Rollback “via TIP”

- The fastest way to do rollback
- No changes is undone by rollback itself
  - Someone after me should undo my changes
- Dirty pages are not flushed to disk
  - Often leads to “orphan” pages
- Transaction state in TIP is set to “rolled back”

Rollback “via TIP” make OIT stuck

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

Resources used by transaction

- Metadata locks
  - Object existence locks
  - Relation locks
- Memory
  - Private copy of transactions inventory
  - Undo log data
  - Temporary blobs data

Object existence locks

- Used to prevents deletion (DROP) of interesting object
  - Tables
  - Views
  - Indices
  - Stored procedures
  - Text collations
- Acquired when statement starts its execution
- Released at hard commit or rollback

Relation locks

- Used to implement consistency isolation mode (protects read\write access to the whole table)
- Acquired:
  - When DML statement executed:
    - SELECT
      - None *(read-committed, snapshot)*
      - Protected Read *(serializable)*
    - INSERT\UPDATE\DELETE\MERGE
      - Shared Write *(read-committed, snapshot)*
      - Exclusive *(serializable)*
  - When transaction starts - if explicit table reservation is used at Transaction Parameters Block
- Released at hard commit or rollback

Temporary blobs

- Persistent (or materialized) blobs
  - Stored within some table
  - All blob data already at some data pages
- Temporary blobs
  - Not assigned to any table
  - Some blob data (up to DB page size) is kept in memory
  - Released at first event:
    - Statement close
    - Transaction hard commit or rollback

Temporary blobs

Bad sample: create a lot of temporary blobs

```
DECLARE Str VARCHAR(255);
DECLARE Blb BLOB;
DECLARE I   INTEGER;
BEGIN
  Blb = '';
  FOR SELECT StrField FROM T1 INTO :Str DO
    Blb = Blb || Str;
END
```

Any “change” of blob creates a new one !

Temporary blobs

Much better: just one blob is created

```
DECLARE Blb BLOB;
BEGIN
  SELECT LIST(StrField) FROM T1 INTO :Blb;
END
```

Temporary blobs

Could create many temporary blobs, one per group

```
SELECT Fld, LIST(StrField)
  FROM T1
GROUP BY Fld
```

Blobs and autonomous transactions

Sample: create blob in autonomous transaction

```
CREATE PROCEDURE ProcA
   RETURNS(Blb BLOB)
AS
BEGIN
   IN AUTOMONOUS DO
      SELECT LIST(StrField) FROM T1 INTO :Blb;

   SUSPEND;
END
```

- Autonomous transaction should not release its temporary blobs to be able to pass it “outside”
- Blobs, created by autonomous transaction, are bound to a “parent” transaction

Temporary blobs

- Blob memory usage:
  - Until materialisation engine keeps in memory up to page_size part of the blob
    - Other blob data stored in database pages
  - A lot of temporary blobs could use a lot of memory !
  - This memory could be overflow to temporary file at disk (Firebird 2.5)
    - TempCacheLimit at firebird.conf

THANK YOU FOR ATTENTION

Firebird official web site **http://www.firebirdsql.org**

Firebird tracker **http://tracker.firebirdsql.org**

hvlad@users.sf.net