All About Transactions workshop

Vlad Khorsun, Firebird Project,
Dmitry Kuzmenko, IBSurgeon

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

Transaction

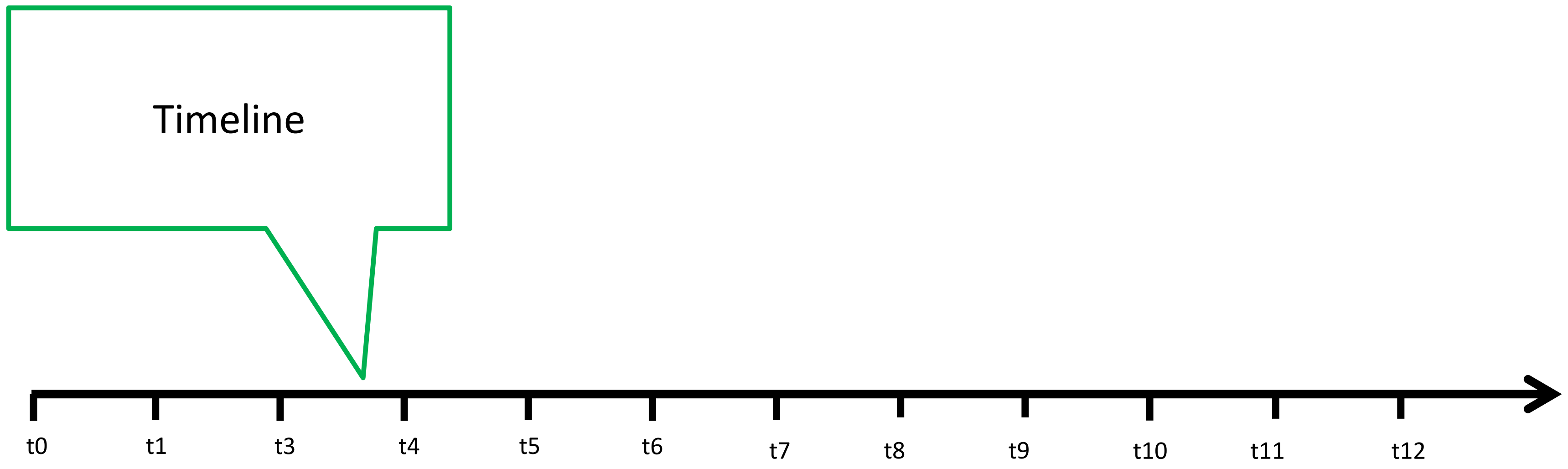
- Transaction as a general concept of dynamic system
- Classic example
  - begin
    - -- move money from account1 to account2
    - Decrease account1
    - Increase account2
  - end – commit/rollback

  - Transaction Managers

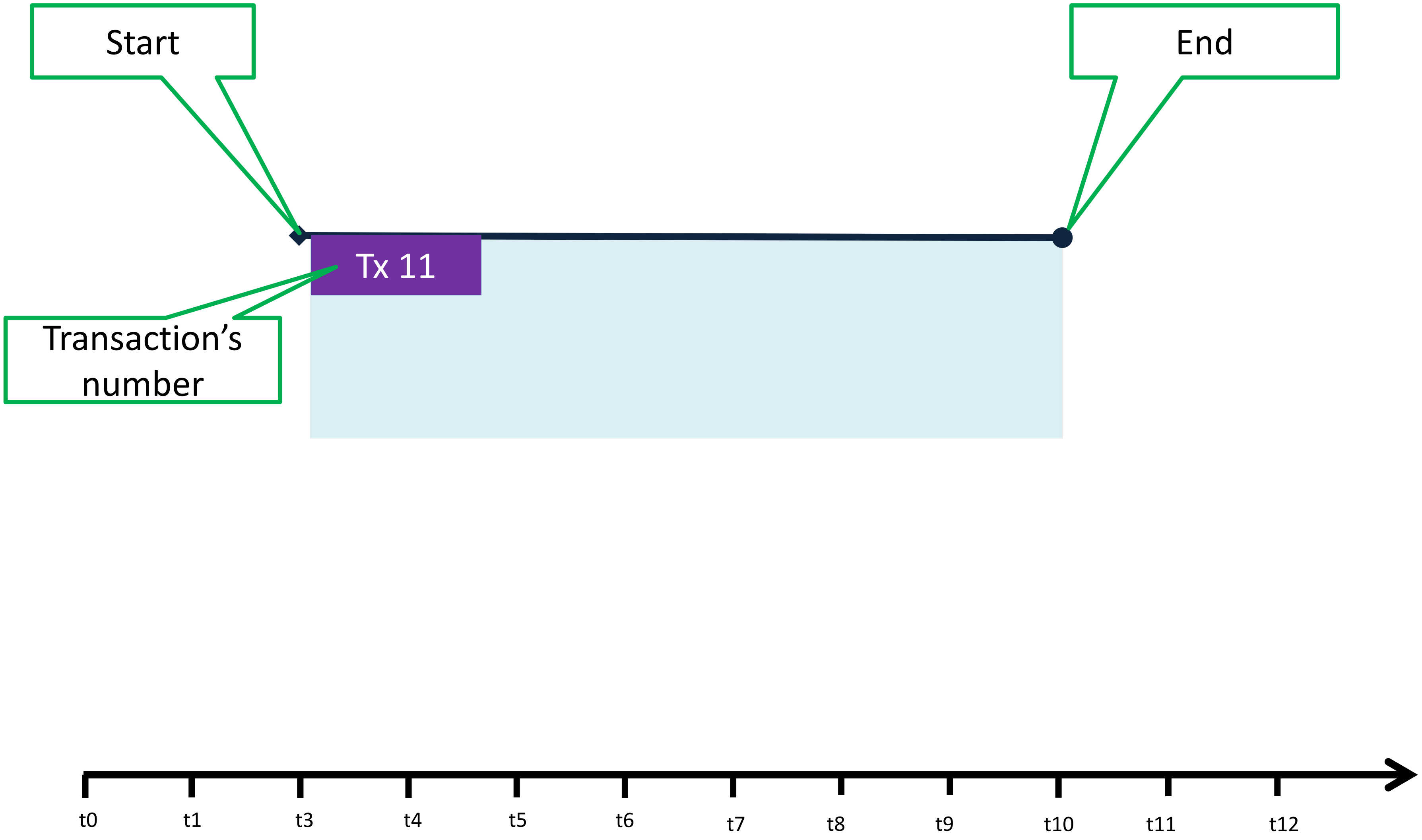
Database transaction

- 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 **atomic**, **consistent**, **isolated** and **durable**

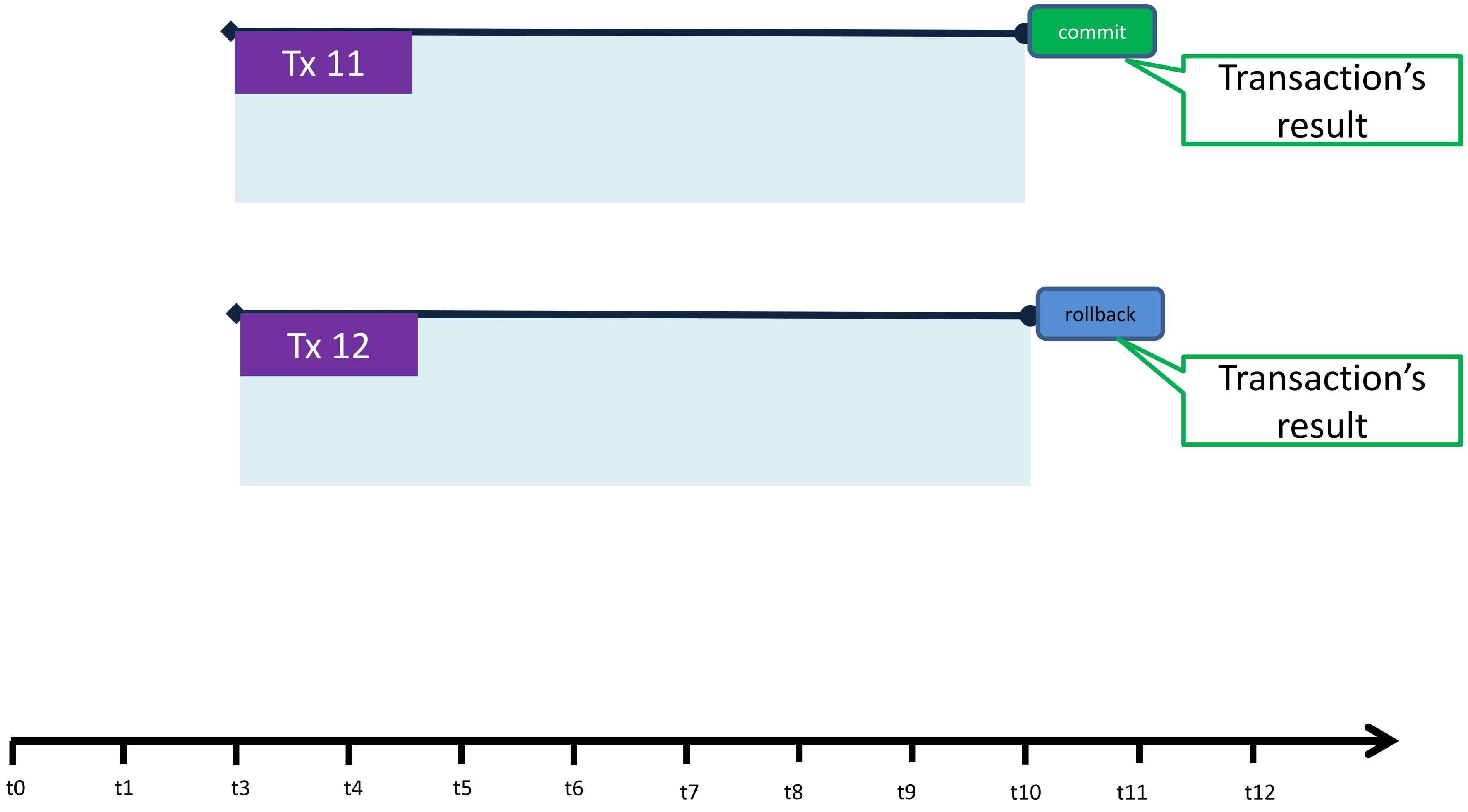
How we will present transactions



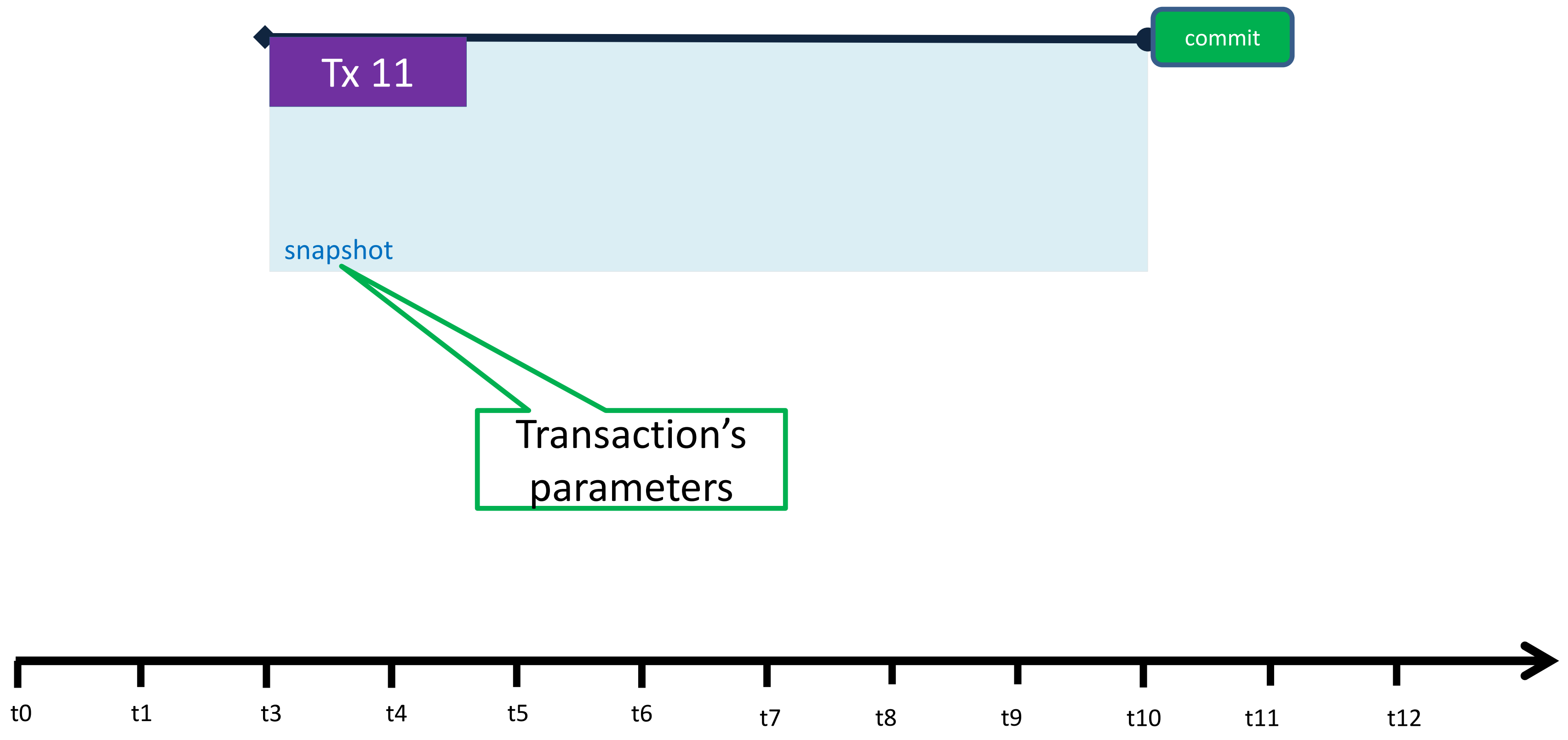
How we will present transactions



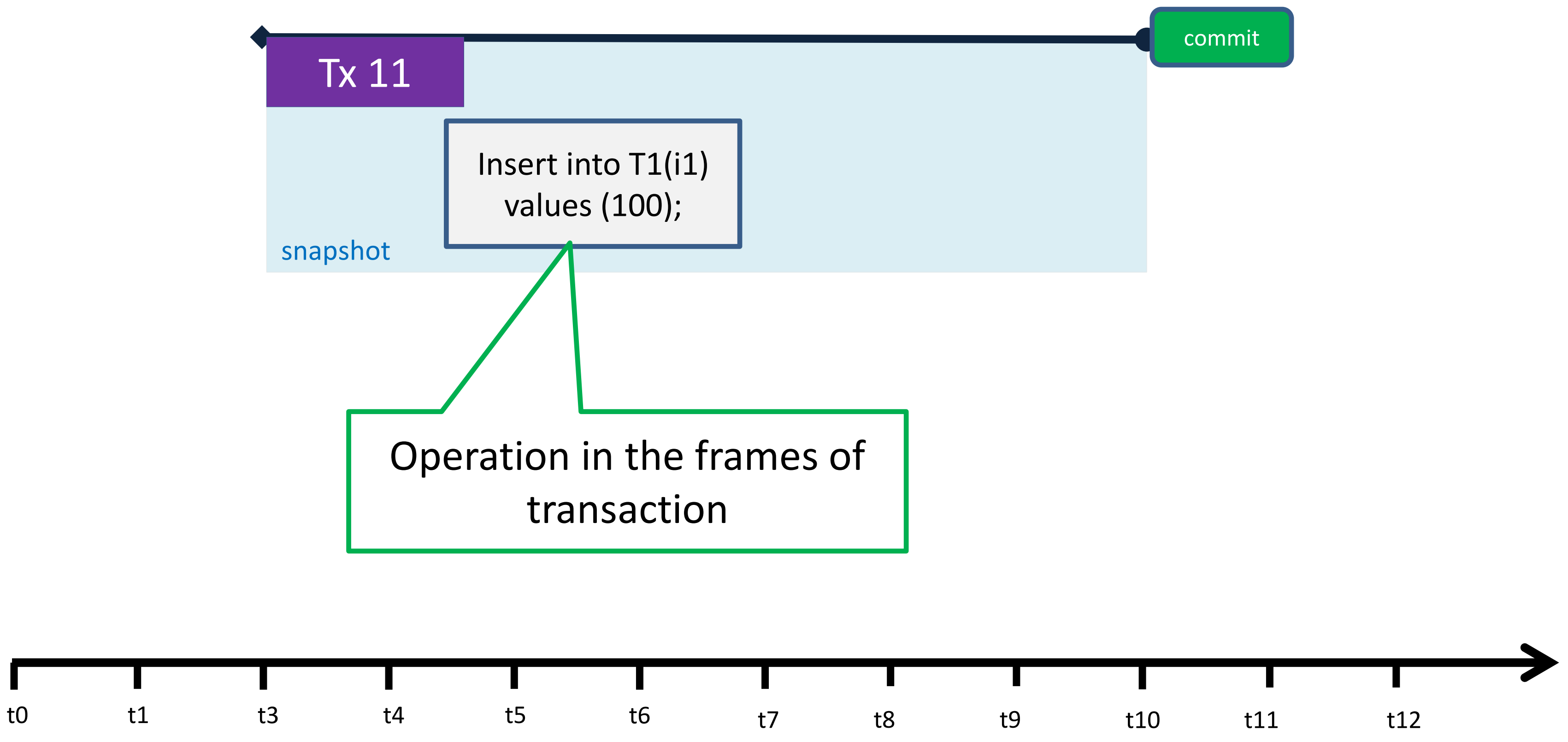
How we will present transactions



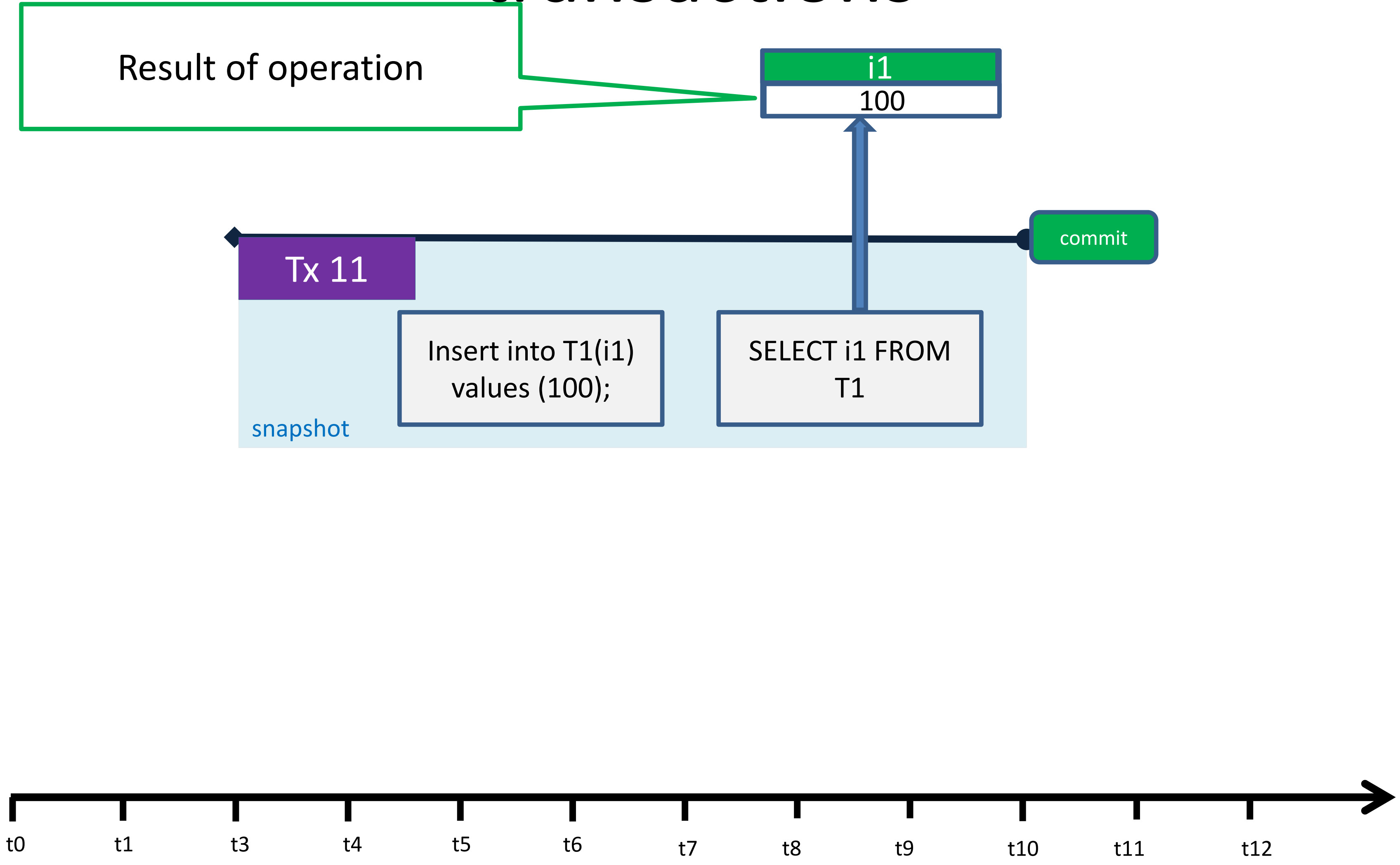
How we will present transactions



How we will present transactions


commit
Tx 11
Insert into T1(i1) values (100);
snapshot
Operation in the frames of transaction
t0
t1
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12

How we will present about transactions

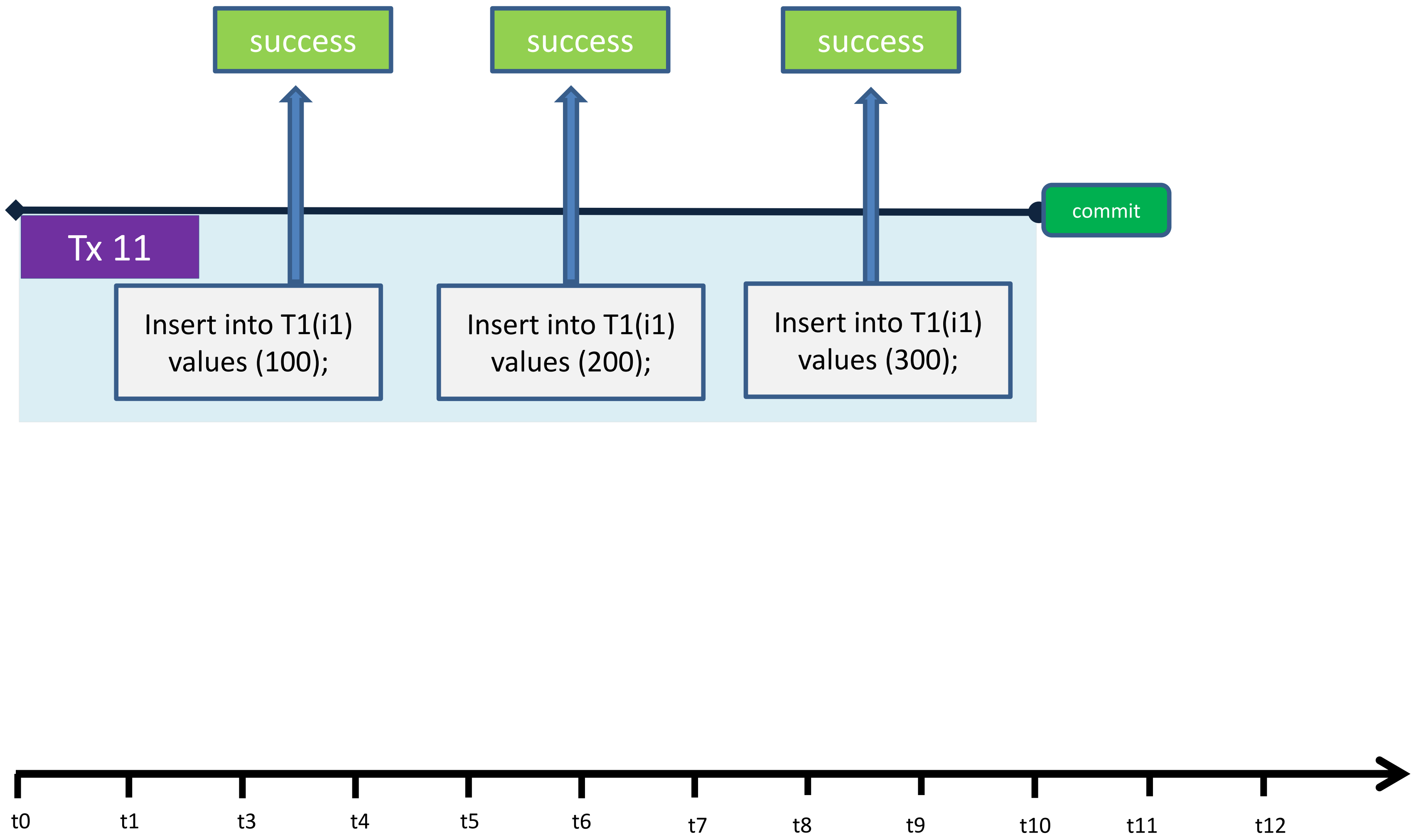


ACID

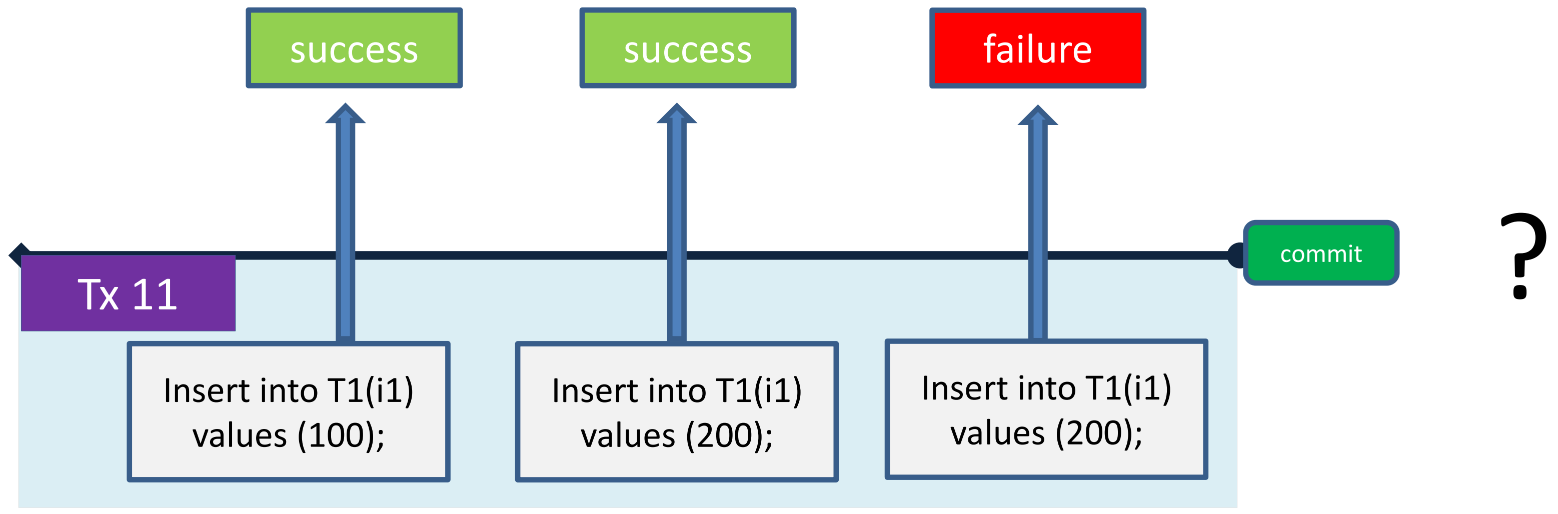
ACID properties are abstract constraints that any transaction must fulfill to comply with definition of transaction.

- A: Atomic
- C: Consistency
- I: Isolation
- D: Durability

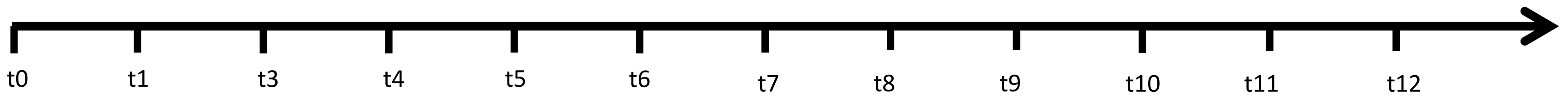
Transactions: Atomic


success
success
success
commit
Tx 11
Insert into T1(i1) values (100);
Insert into T1(i1) values (200);
Insert into T1(i1) values (300);
t0
t1
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12

Transactions: Atomic


success
success
failure
commit
?
Tx 11
Insert into T1(i1) values (100);
Insert into T1(i1) values (200);
Insert into T1(i1) values (200);

Most servers does not allow to commit, if any operator inside transaction returned an error. Firebird allows that, you may apply commit, it’s your decision.



Transaction: Atomic

- 2 levels of Atomic
- Atomic operator: always atomic
  - UPDATE t1 – update all or nothing
- Atomic group of operators (in the frame of transaction)
  - UPDATE t1
  - UPDATE t2
  - …depends on business logic and application developer

Transaction: Atomic

- Atomic means that all operations and their results will be processed together
- Atomic gives **an ability** to commit or rollback group of operations in the frames of transaction, according to the business logic you need to implement
- In wrongly designed system money transfer can be like this:
  - Begin transaction
    - Decrease money on account 1…. Success
    - Increase money on account 2… Failure
  - Commit

Transaction: Consistency

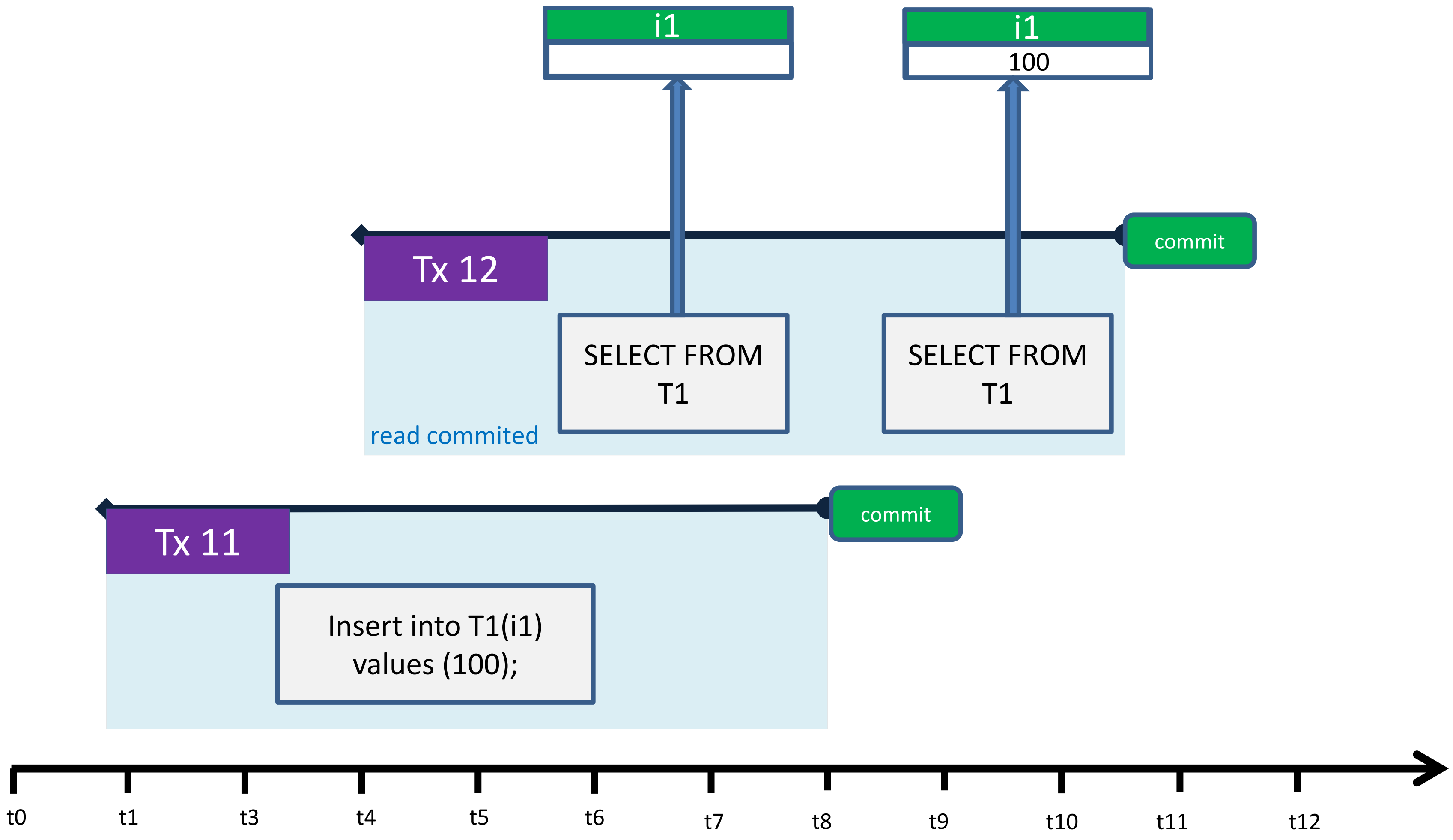
- A transaction enforces consistency of the system state by ensuring that at the end of any transaction the system is in a valid state.

- 2 levels of consistency:
  - Database level - enforced by database constraints
  - Application (business) level - enforced by application developer, with support from database engine

Transaction: Isolation

- **Isolation refers to the requirement that other operations cannot access or see the data in an intermediate state during a transaction.** This constraint is required to maintain the performance as well as the consistency between transactions in a database. Thus, each transaction is unaware of another transactions executing concurrently in the system.
- Supported by isolation levels concept

Transaction: Isolation (RC)



Transactions: Durability

- The concept of durability allows the developer to know that a completed (committed) transaction is a permanent part of the system, regardless of what happens to the system later on.

- Commit, then Reset.

ACID: Summary

- ACID are requirements for implementation of transactions in specific database engine
- Atomic
  - Operators are atomic
  - Group of operators can be atomic, supported by transactions
- Consistency
  - 2 levels of consistency: database constraints and application
- Isolation
  - Supported by transaction mechanism almost 1:1
- Durability
  - All commited data becomes permanent.

Transaction parameters: read/write, wait/nowait

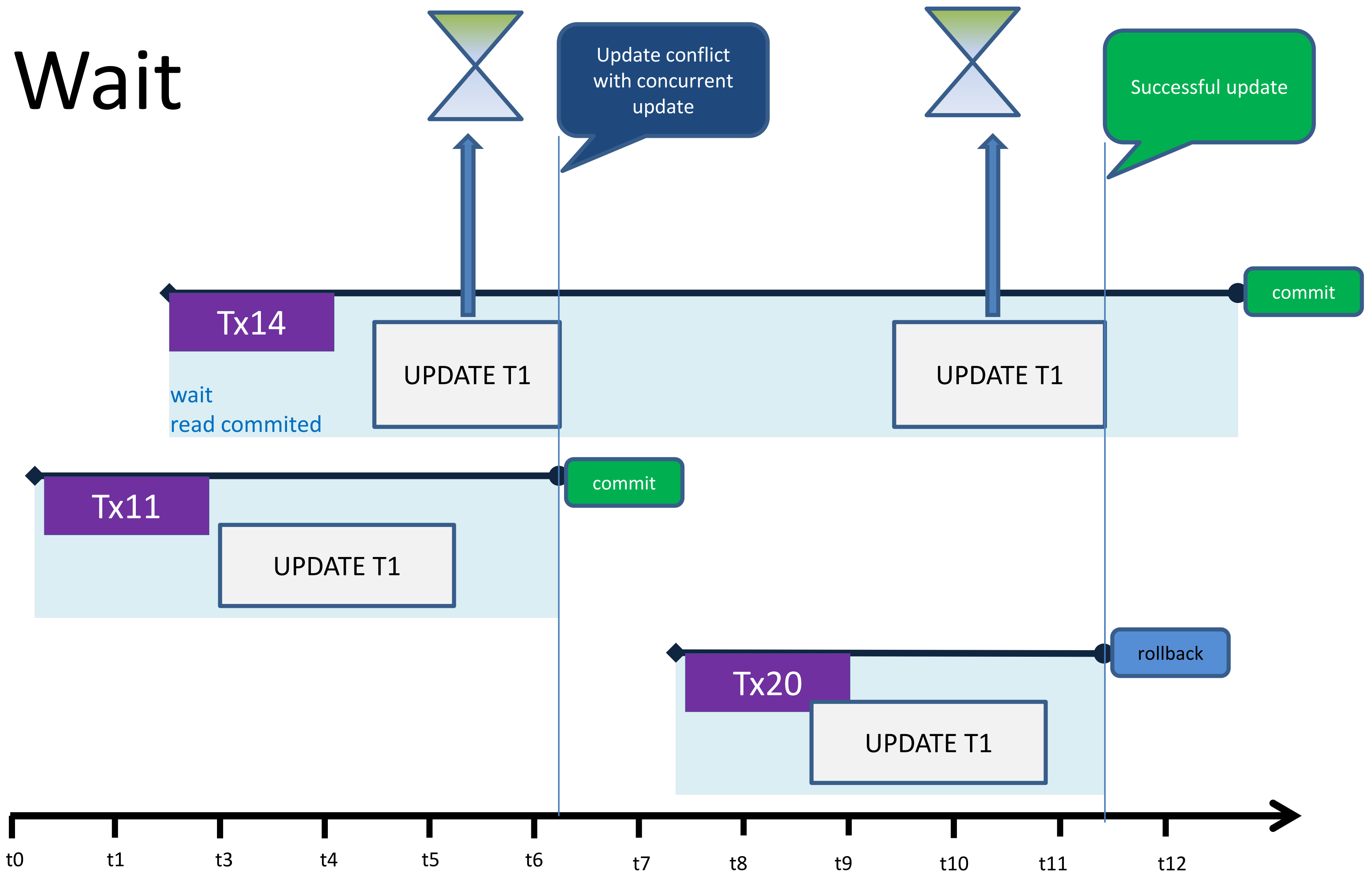
Write/Read-only

- Write is default
- Read-only
  - Cannot write
  - Read-only Read Commited is optimized to run eternally (see Firebird 4 notes)
  - Can write to temporary tables!
  - Can change generators
  - Can produce temporary blobs (concatenation, list function, etc)

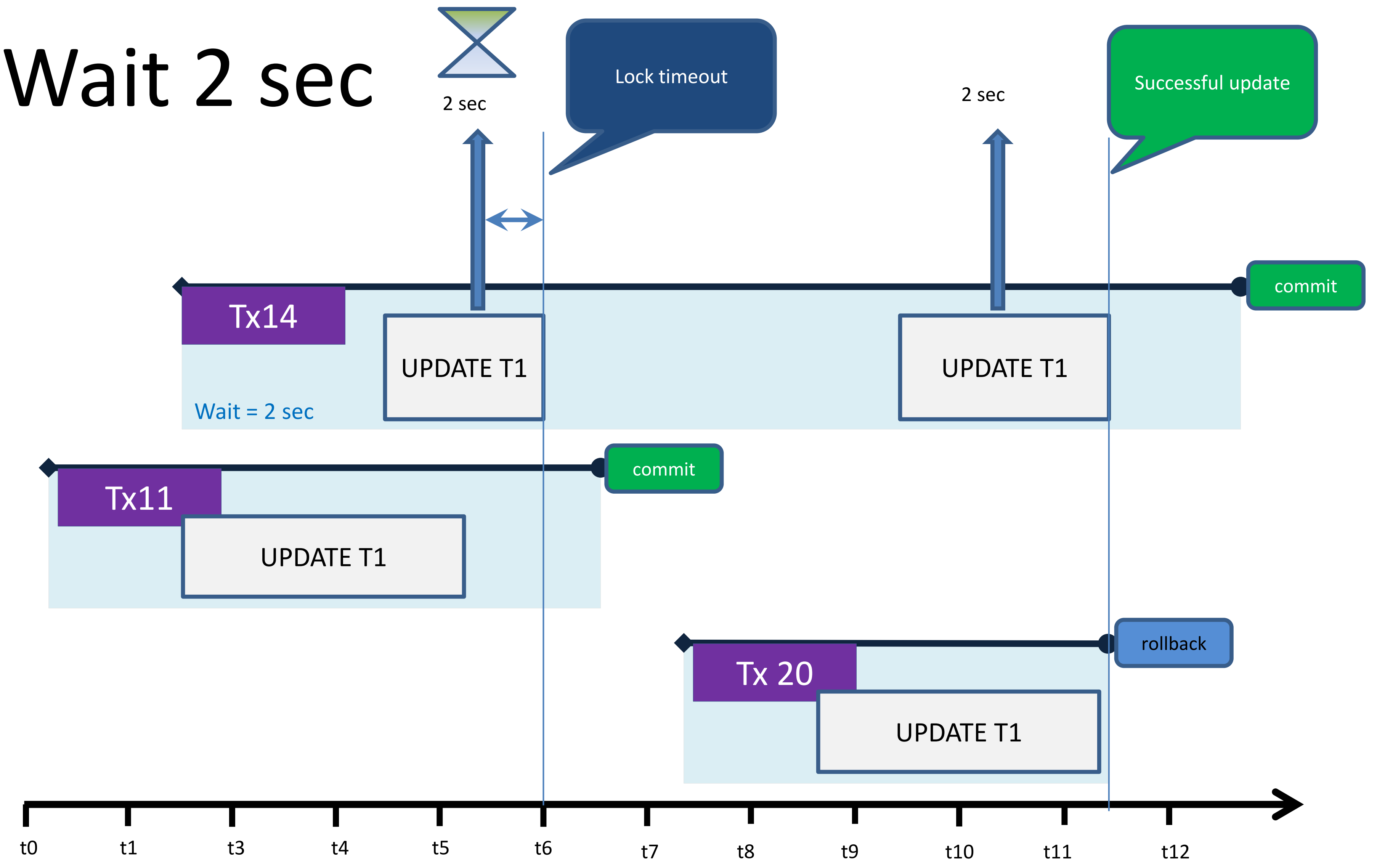
Wait

- Wait is default transaction mode
- Wait without parameter – endlessly wait
- Wait with parameter – wait till the timeout
- Nowait: timeout = 0

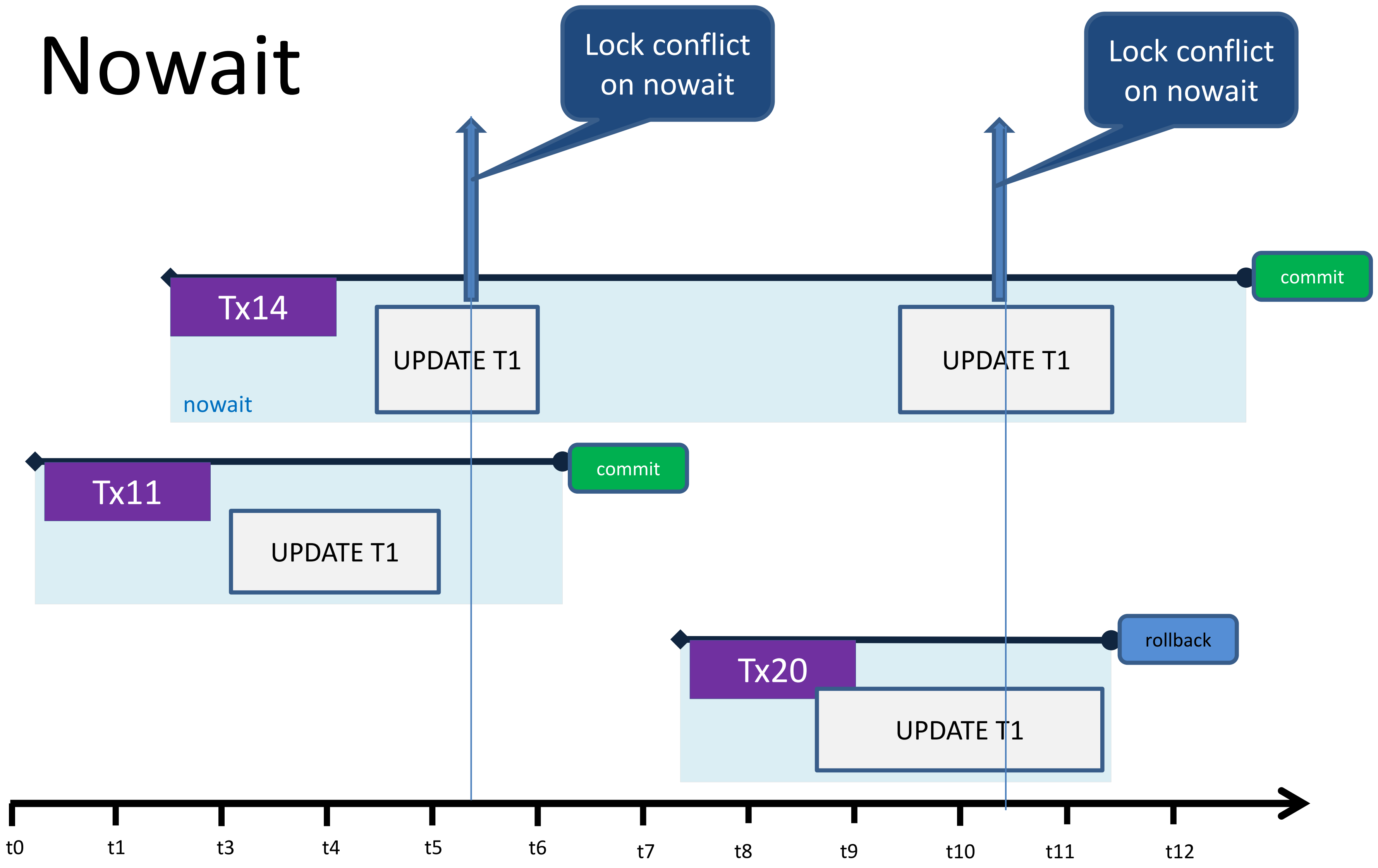
Wait


Update conflict with concurrent update
Successful update
Tx14
UPDATE T1
UPDATE T1
commit
wait
read commited
Tx11
UPDATE T1
commit
Tx20
UPDATE T1
rollback
t0
t1
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12

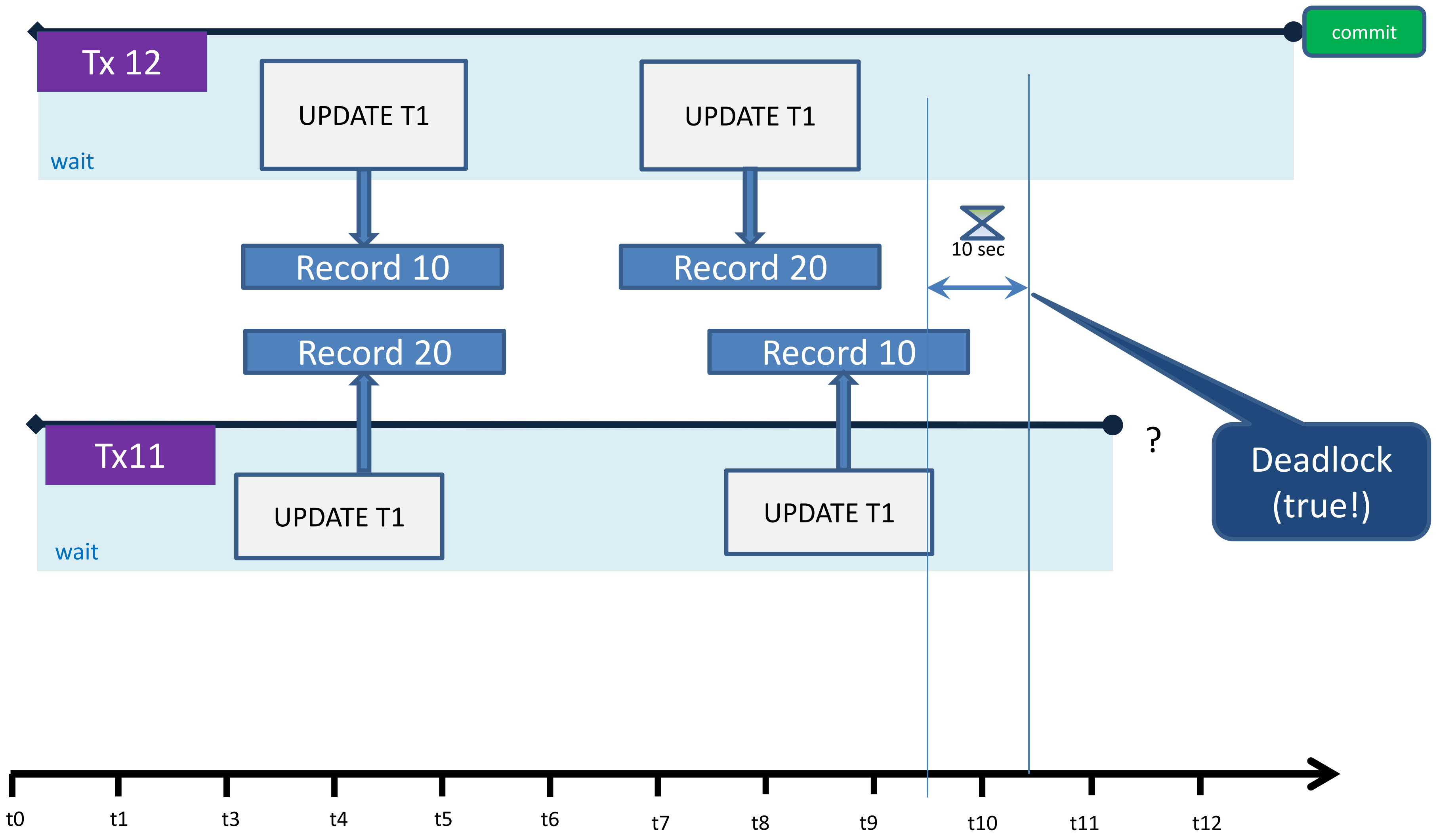
Wait 2 sec


2 sec
Lock timeout
2 sec
Successful update
Tx14
UPDATE T1
UPDATE T1
Wait = 2 sec
commit
Tx11
UPDATE T1
commit
Tx 20
UPDATE T1
rollback
t0
t1
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12

Nowait


Lock conflict on nowait
Lock conflict on nowait
Tx14
nowait
UPDATE T1
UPDATE T1
commit
Tx11
UPDATE T1
commit
Tx20
UPDATE T1
rollback
t0
t1
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12

True deadlock on records



After timeout server turns one of these transactions to nowait, allowing it to return an error

Isolation levels

Standard isolation levels

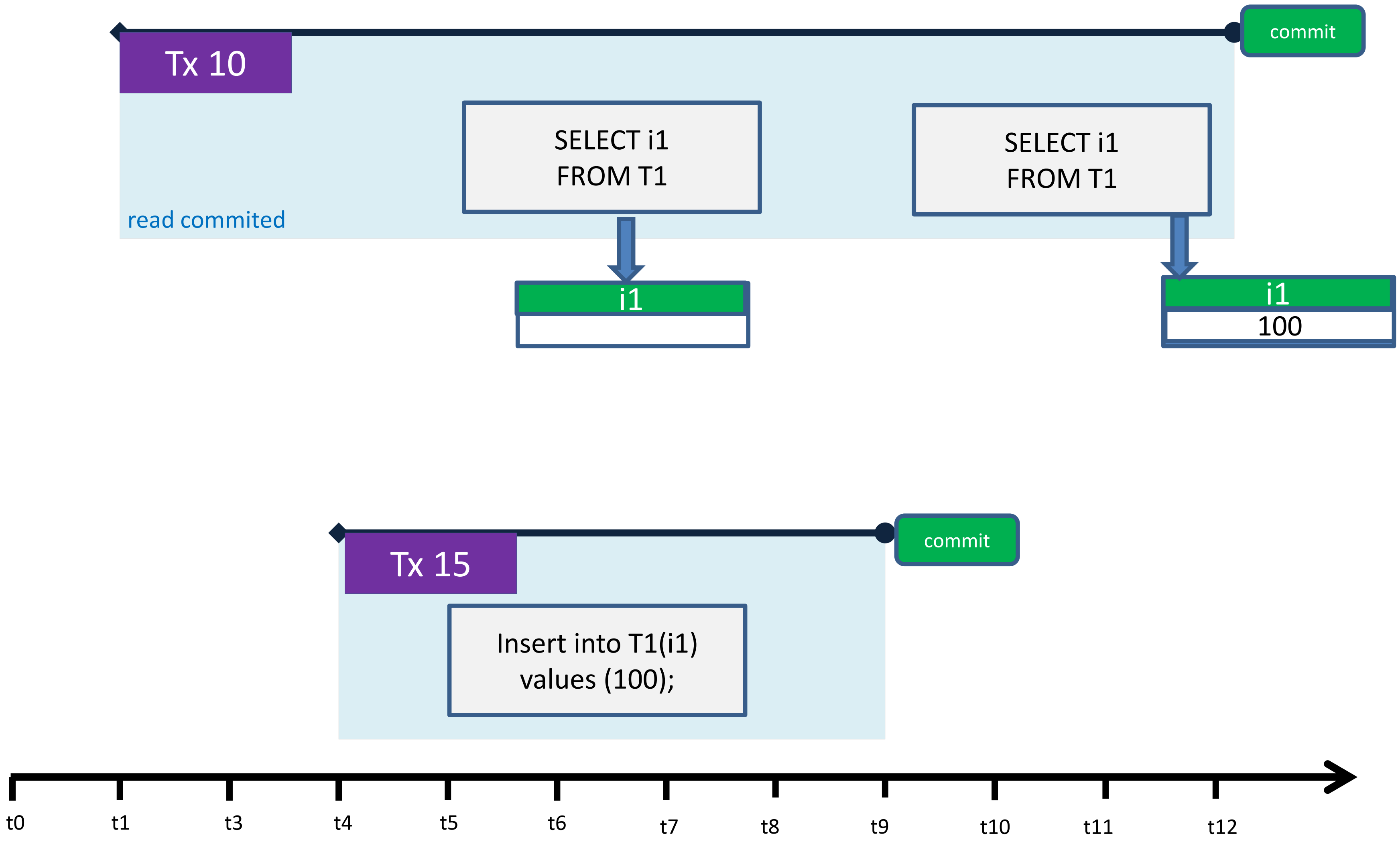
- Based on locking engines (too old)
- READ UNCOMMITTED
  - Or Dirty Read, like DBF
- READ COMMITTED
  - Reading new committed changes
- REPEATABLE READ
  - Allows phantoms – re-reading can show new committed changes
- SERIALIZABLE
  - All transactions goes serial, no conflicts

- A Critique of ANSI SQL Isolation Level – 1995
  - Repeatable read -> snapshot

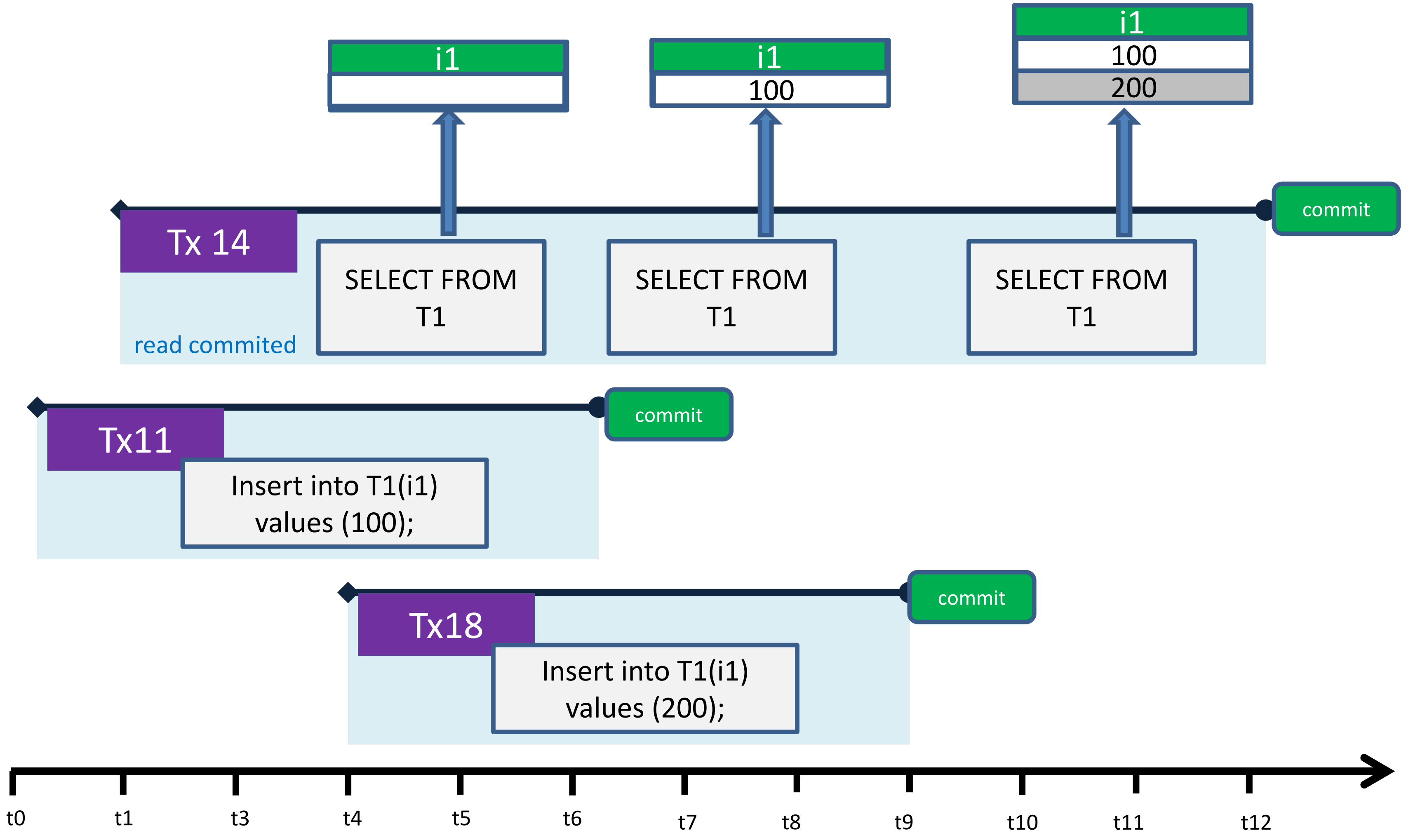
Firebird and Standard isolation levels

ANSI Isolation Levels	**Firebird Isolation Levels**
Read Uncommitted	n/a
Read Commited	Read Commited
Repeatable Read	Snapshot
Serializable	SNAPSHOT WITH TABLE STABILITY

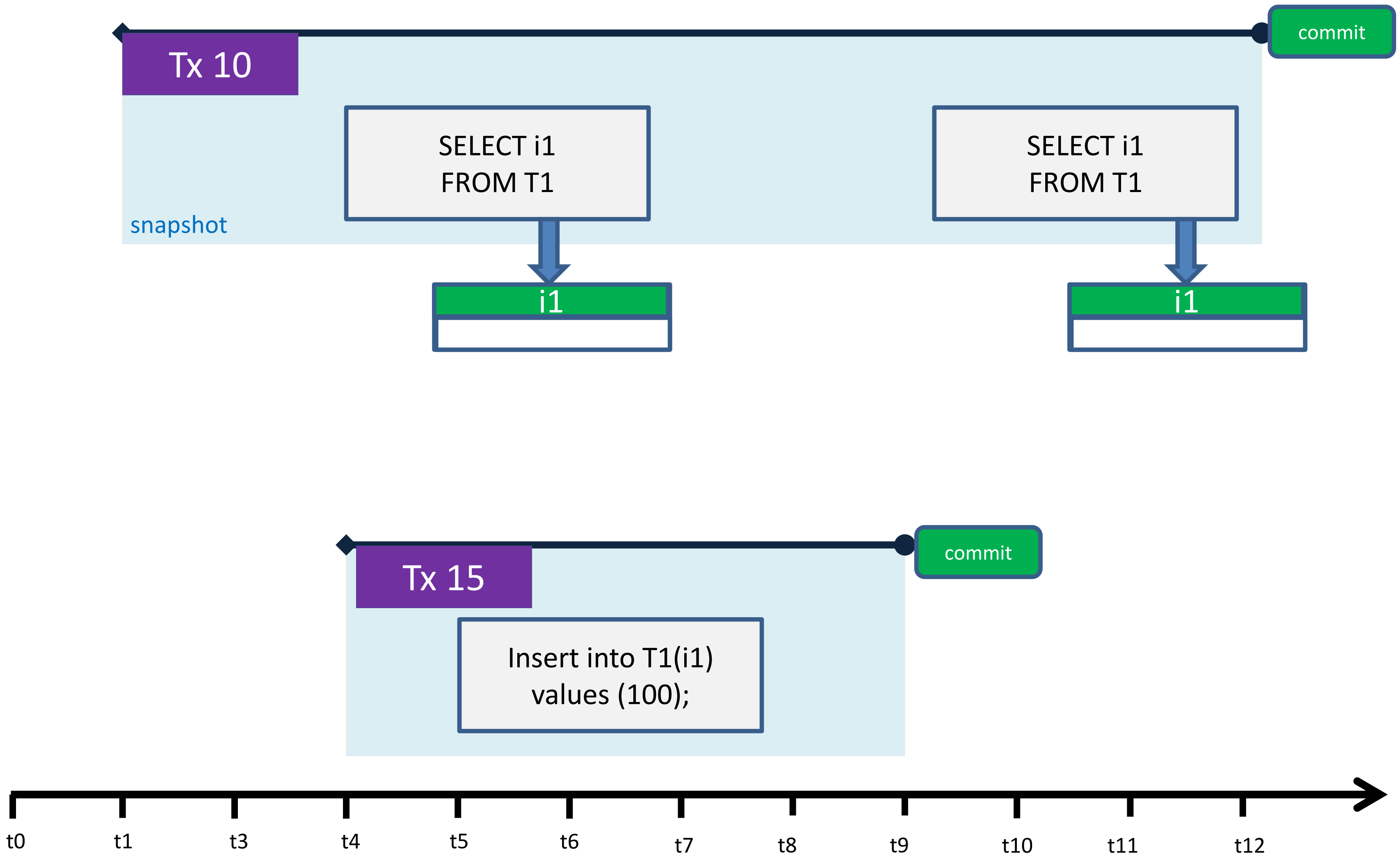
Read commited: simple example



Read commited: example with 2 transactions



Snapshot


commit
Tx 10
SELECT i1
FROM T1
SELECT i1
FROM T1
snapshot
i1
i1
commit
Tx 15
Insert into T1(i1)
values (100);
t0
t1
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12

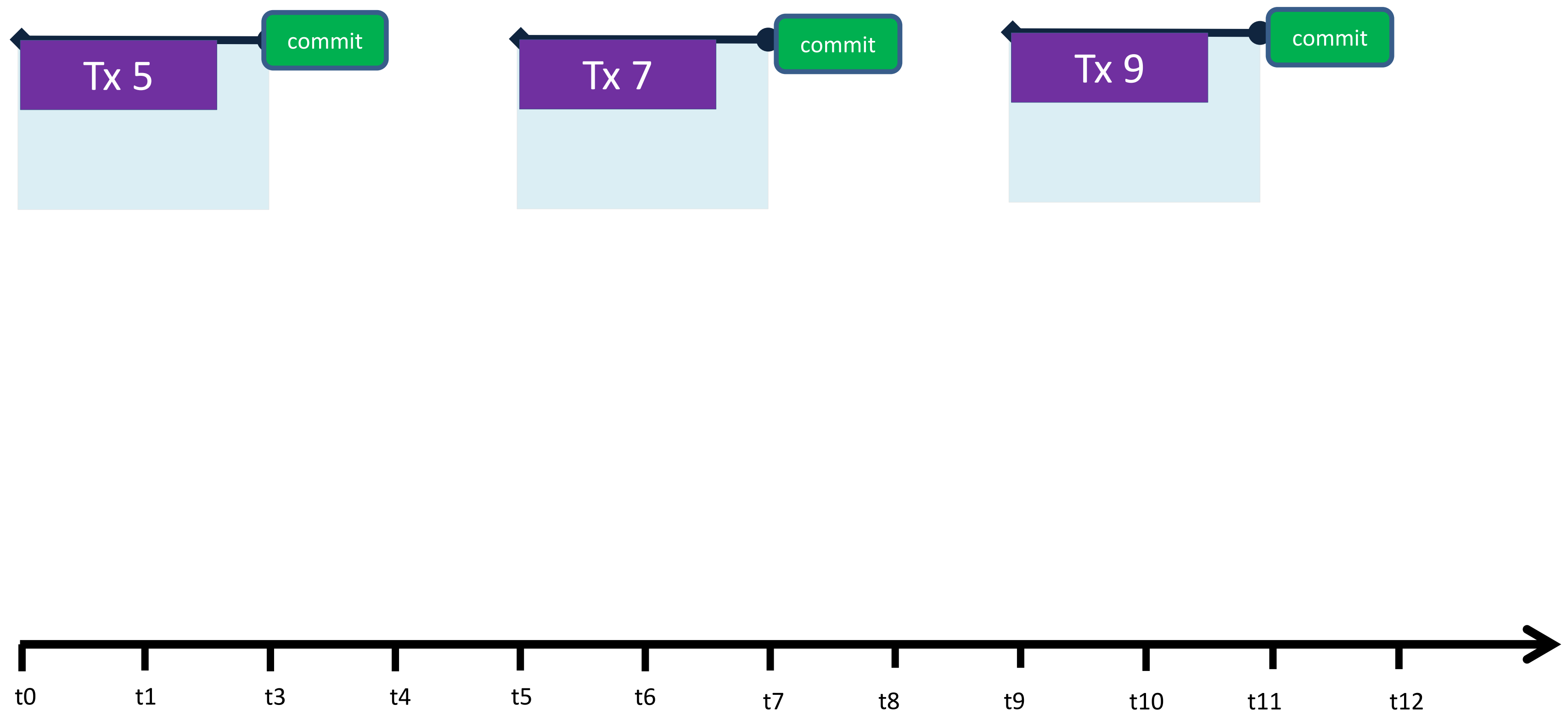
SNAPSHOT WITH TABLE STABILITY

- SNAPSHOT + Exclusive lock for table for read or write
- Let’s consider wait/nowait before going into details

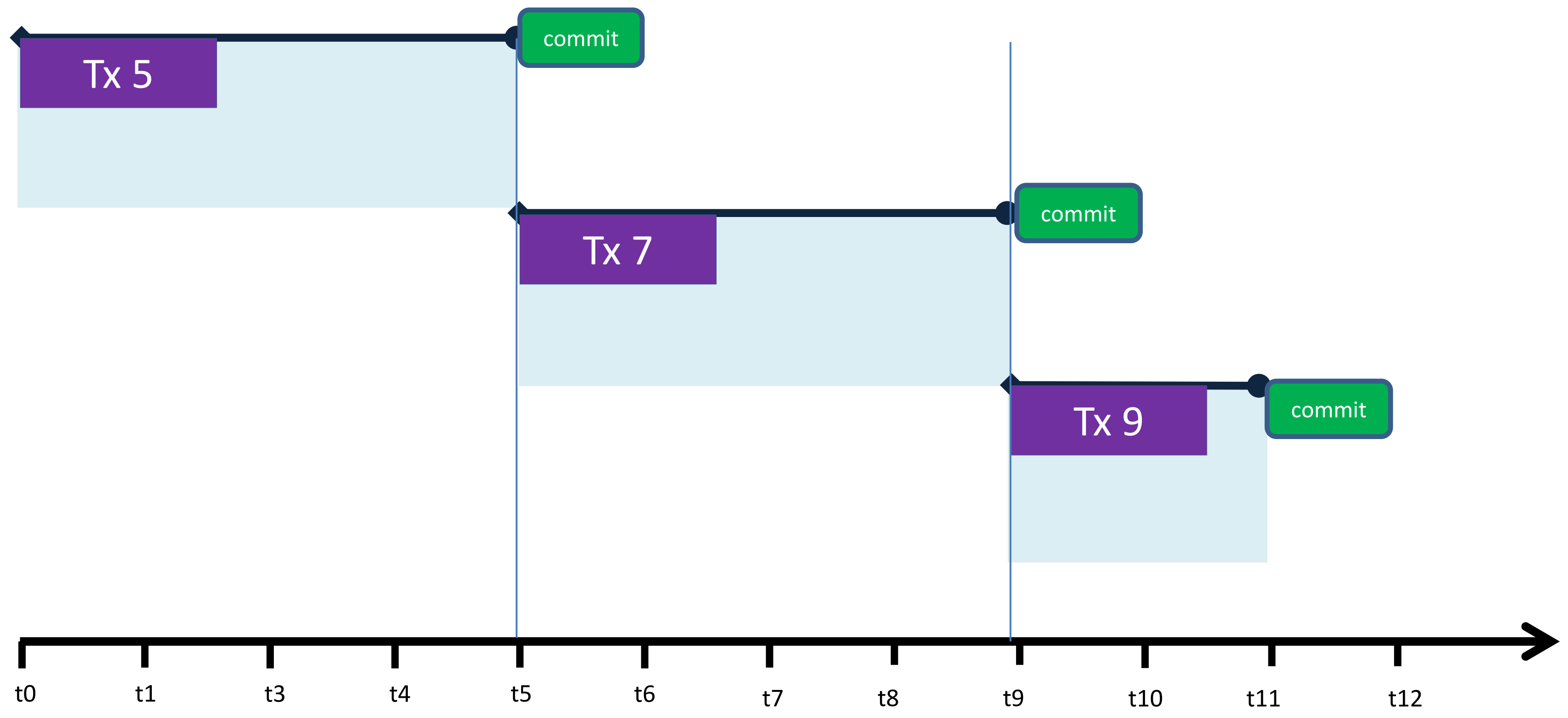
SNAPSHOT TABLE STABILITY

- SNAPSHOT TABLE STABILITY
  - Lock the whole table – prevent write or read access
- Without explicit table reservation:
  - Lock tables at first access, not at the start of transaction.
    - Deadlock!
- TABLE RESERVATION option
  - Locks specific table at the start of transaction
  - Wait option is recommended

Queue with Sequential transactions



SNAPSHOT WITH TABLE STABILITY WITH explicit TABLE RESERVATION



Snapshot Table Stability: Examples

- Queue implementation
  - With TABLE RESERVATION
  - Short “wait” transactions will be put in queue
  - Locks will be resolved on transaction level – i.e., there will be no lock conflict on record levels
- Tables as locks
  - Use locked table as flag for other transactions
- Rebuilding table in exclusive mode
  - Engine use Table Stability for when building indices

Summary

- ACID is requirement for implementation
- Transaction is a basis and great support of logic implementation
- Most useful isolation levels in Firebird are Read Commited and Snapshot
- Default parameters are “snapshot”, “wait”, “write”
- Defaults of the components/drivers may be different!

Next…

NEXT…