CASK

Transactions in HBase

Andreas Neumann
Gokul Gunasekaran
HbaseCon June 2017

gokul at cask.co
anew at apache.org
@caskoid

Goals of this Talk

- Why transactions?
- Optimistic Concurrency Control
- Three Apache projects: Omid, Tephra, Trafodion
- How are they different?

Transactions in noSQL?

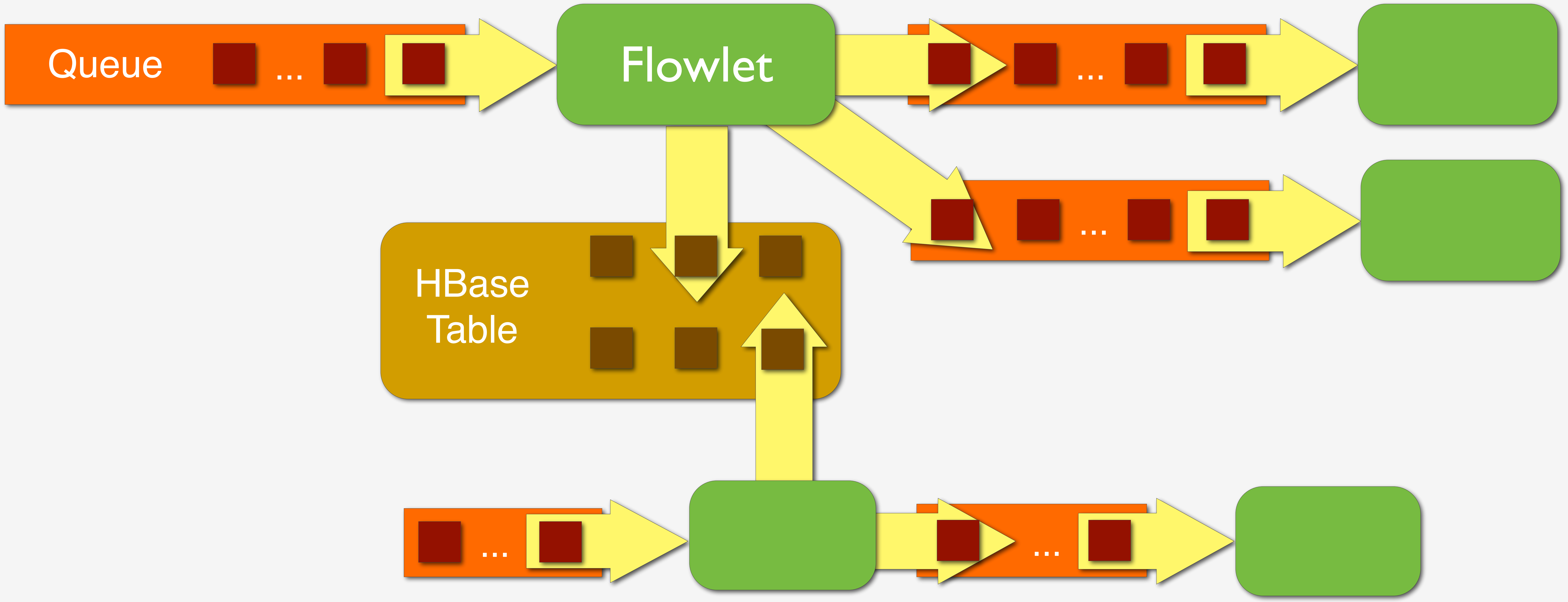
History

- SQL: RDBMS, EDW, …
- noSQL: MapReduce, HDFS, HBase, …
- n(ot)o(nly)SQL: Hive, Phoenix, …

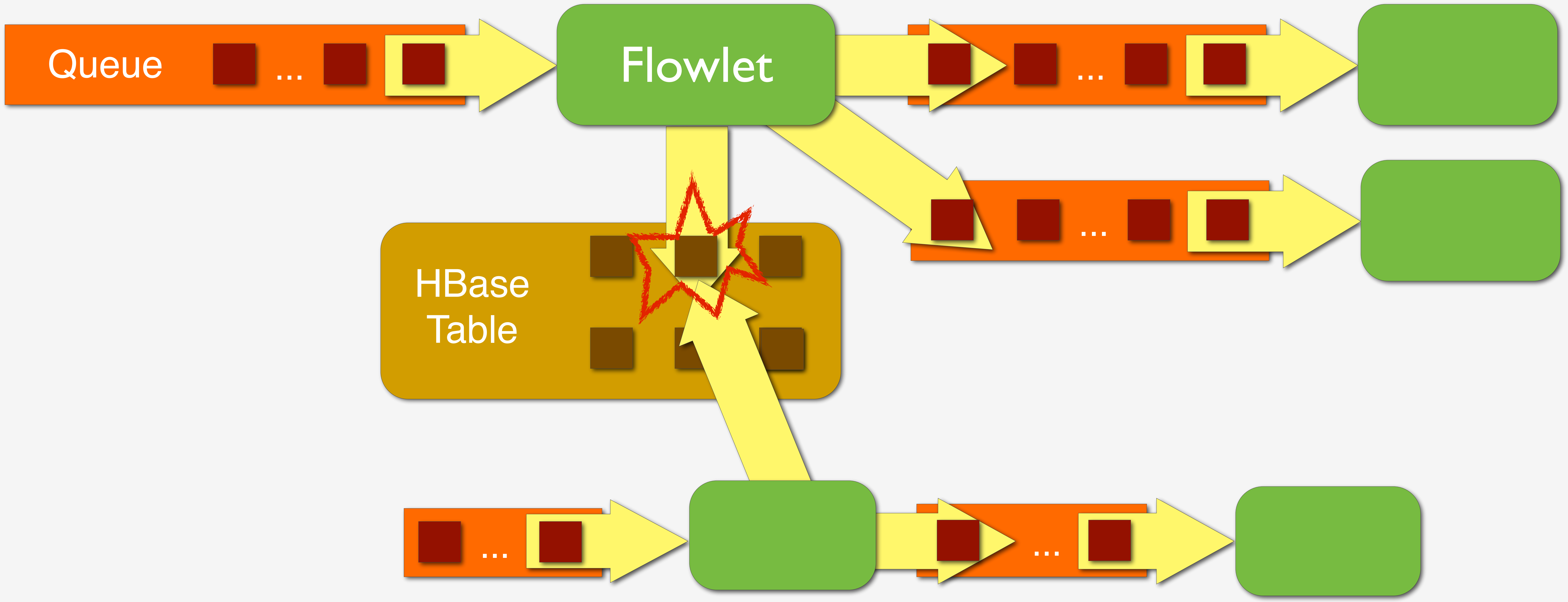
Motivation:

- Data consistency under highly concurrent loads
- Partial outputs after failure
- Consistent view of data for long-running jobs
- (Near) real-time processing

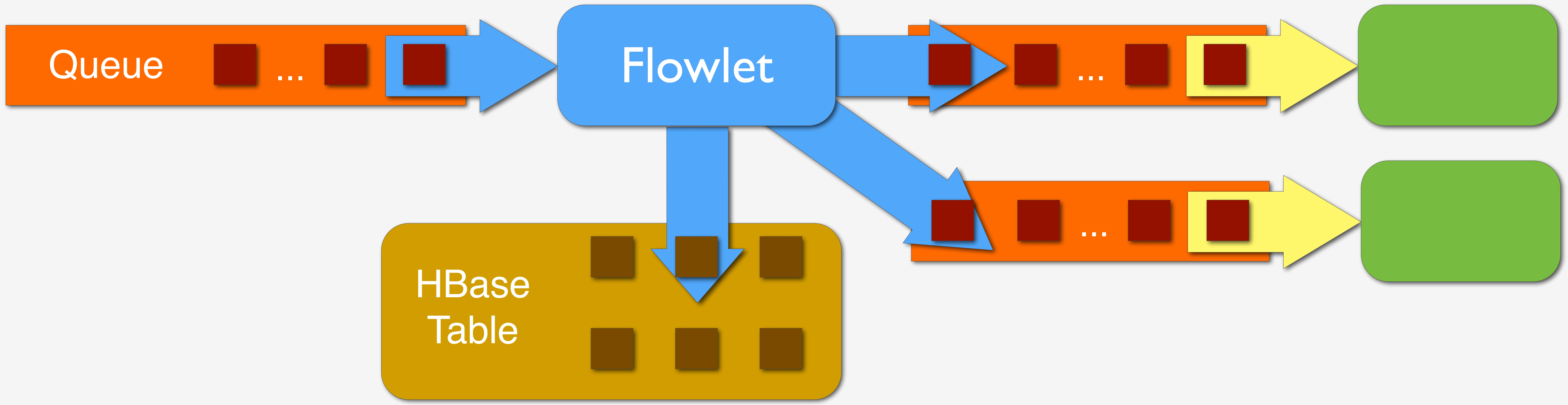
Stream Processing



Write Conflict!



Transactions to the Rescue



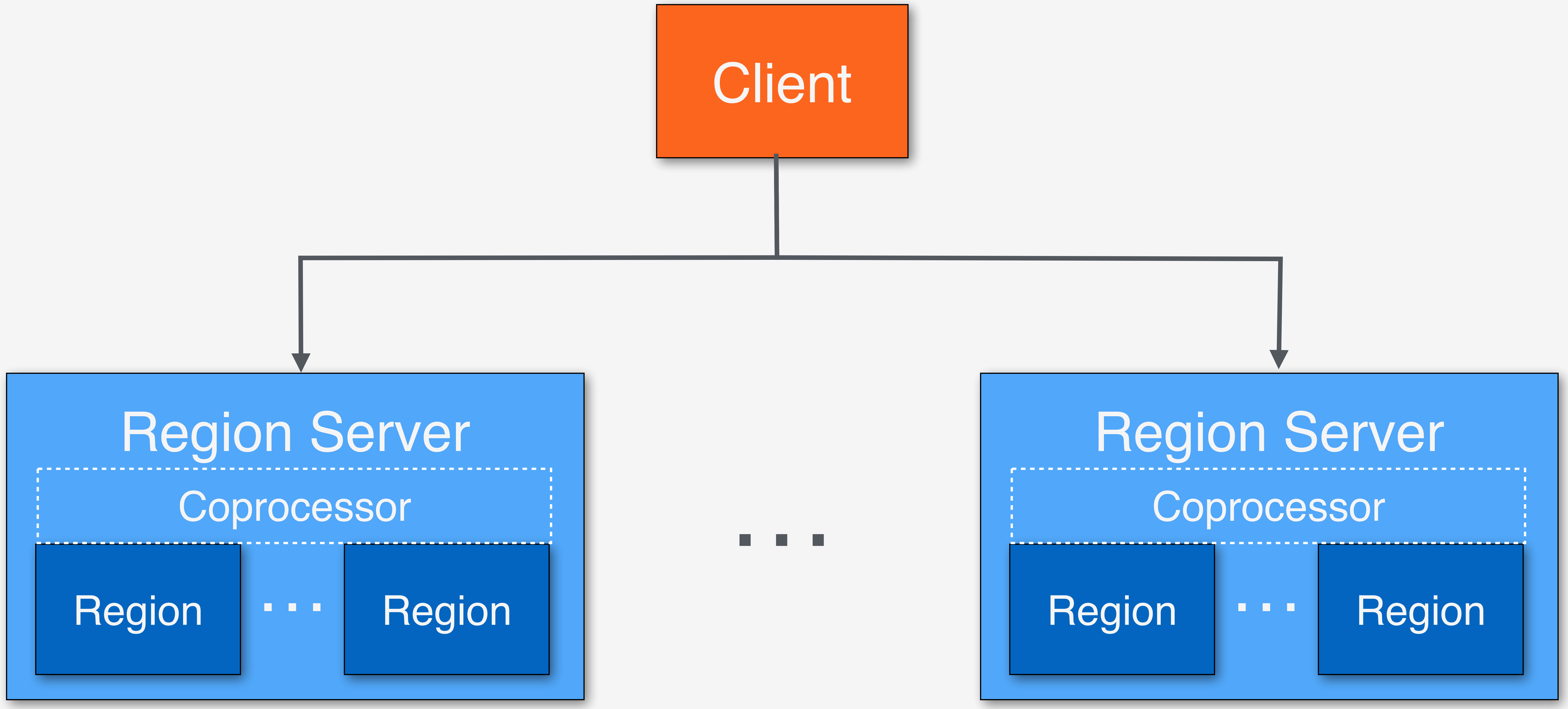
- Atomicity of all writes involved
- Protection from concurrent update

ACID Properties

From good old SQL:

- **Atomic** - Entire transaction is committed as one
- **Consistent** - No partial state change due to failure
- **Isolated** - No dirty reads, transaction is only visible after commit
- **Durable** - Once committed, data is persisted reliably

What is HBase?



What is HBase?

Simplified:

- Distributed Key-Value Store
- Key = <row>.<family>.<column>.<timestamp>
- Partitioned into Regions (= continuous range of rows)

- Each Region Server hosts multiple regions
- Optional: Coprocessor in Region Server

- Durable writes

ACID Properties in HBase

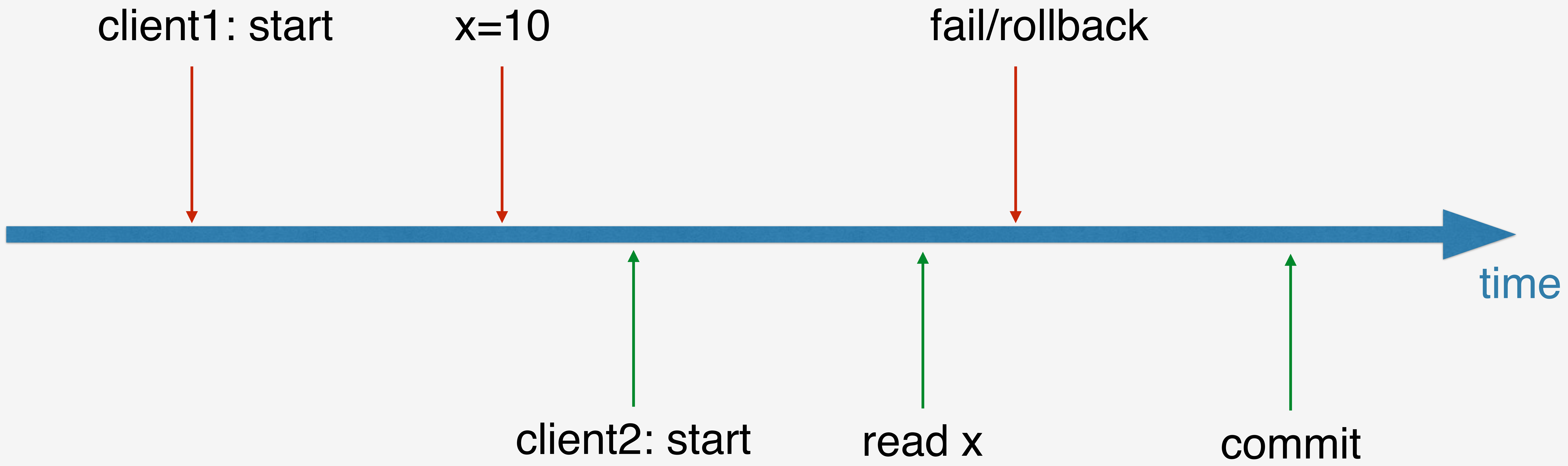
- **Atomic**
  - At cell, row, and region level
  - Not across regions, tables or multiple calls
- **Consistent** - No built-in rollback mechanism
- **Isolated** - Timestamp filters provide some level of isolation
- **Durable** - Once committed, data is persisted reliably

How to implement full ACID?

Implementing Transactions

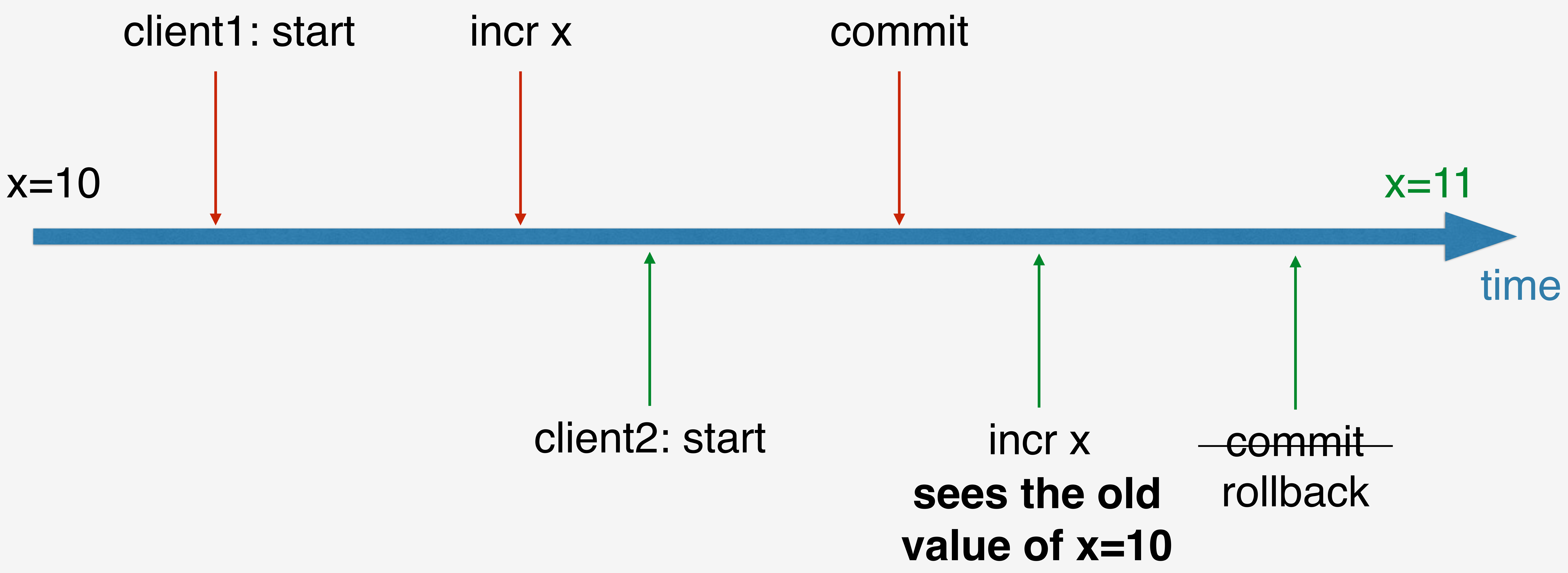
- Traditional approach (RDBMS): locking
    - May produce deadlocks
    - Causes idle wait
    - complex and expensive in a distributed env
- Optimistic Concurrency Control
    - lockless: allow concurrent writes to go forward
    - on commit, detect conflicts with other transactions
    - on conflict, roll back all changes and retry
- Snapshot Isolation
    - Similar to repeatable read
    - Take snapshot of all data at transaction start
    - Read isolation

Optimistic Concurrency Control

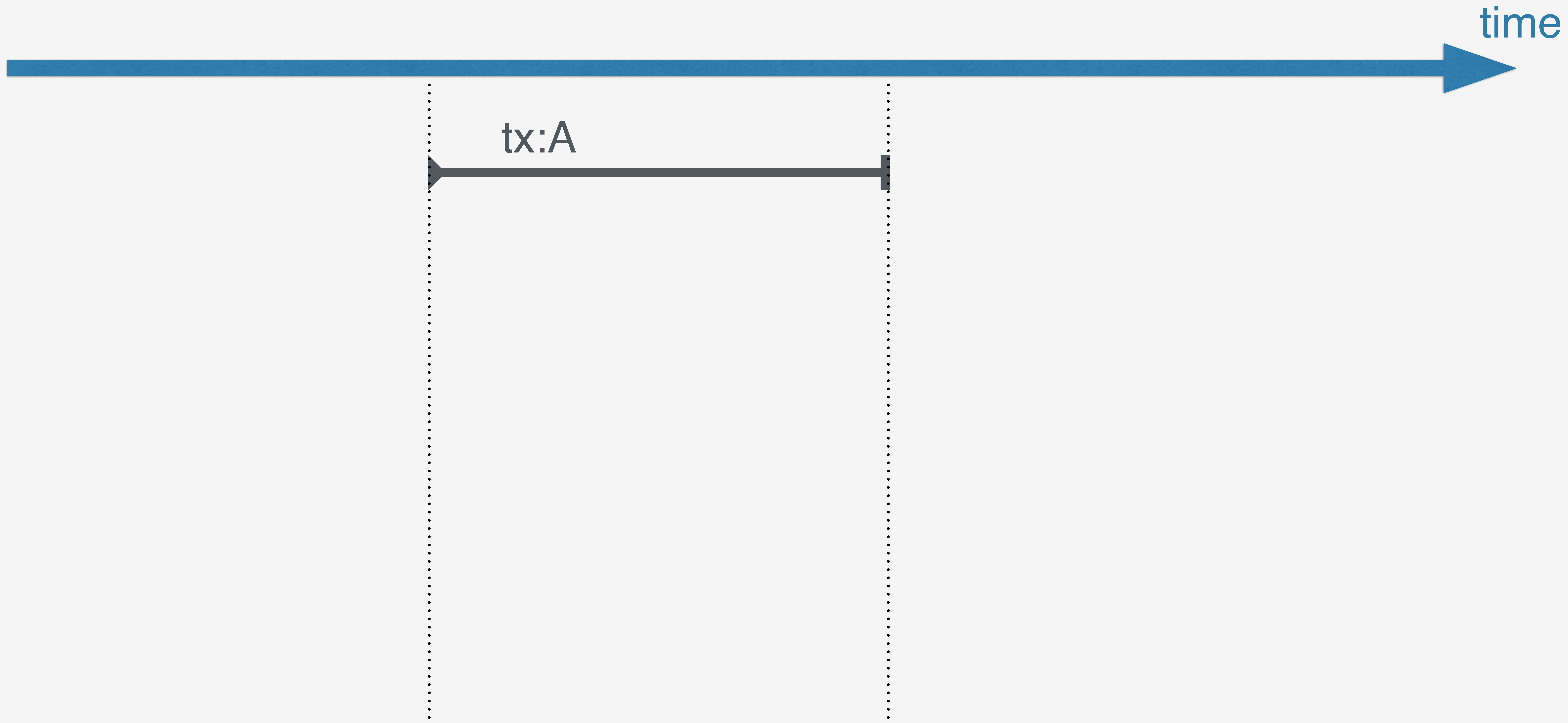


must see the old value of x

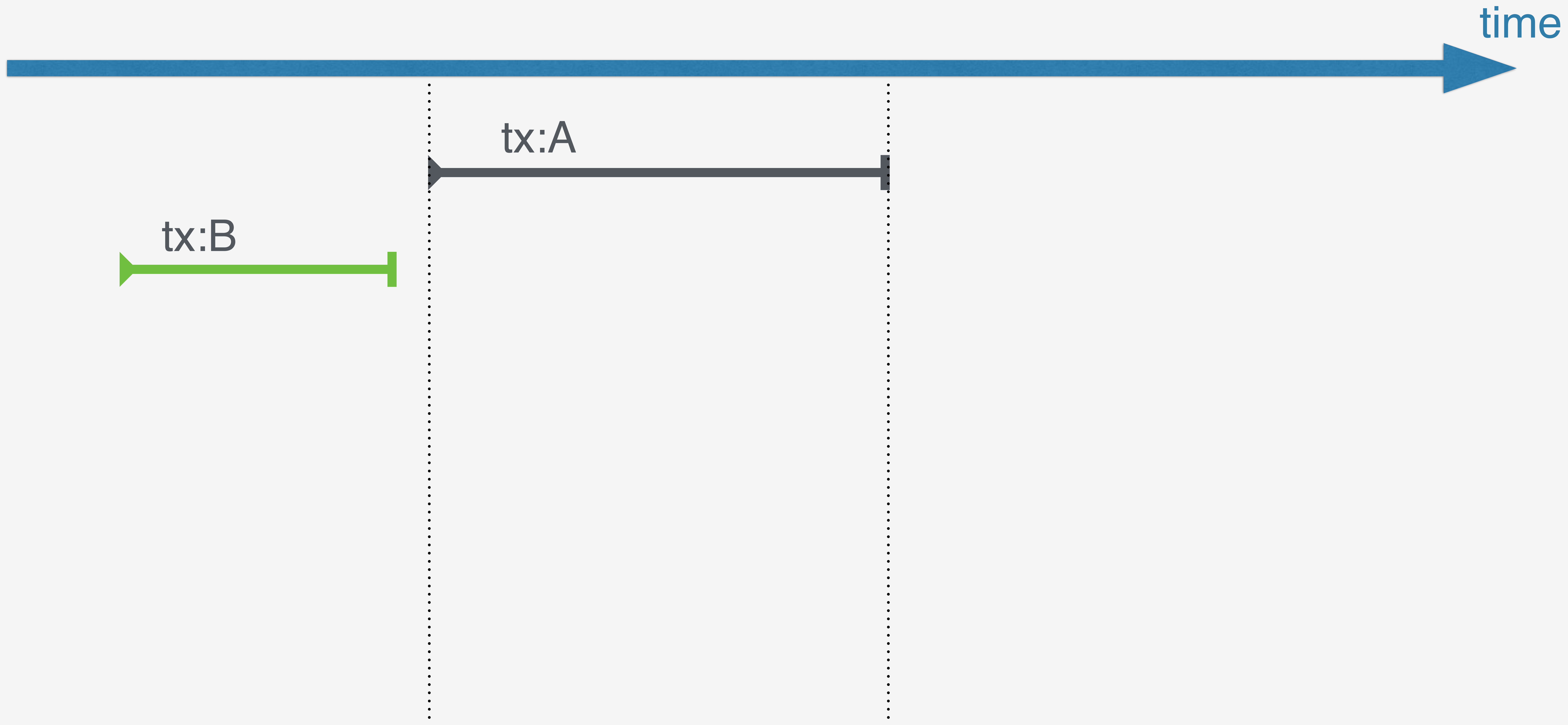
Optimistic Concurrency Control



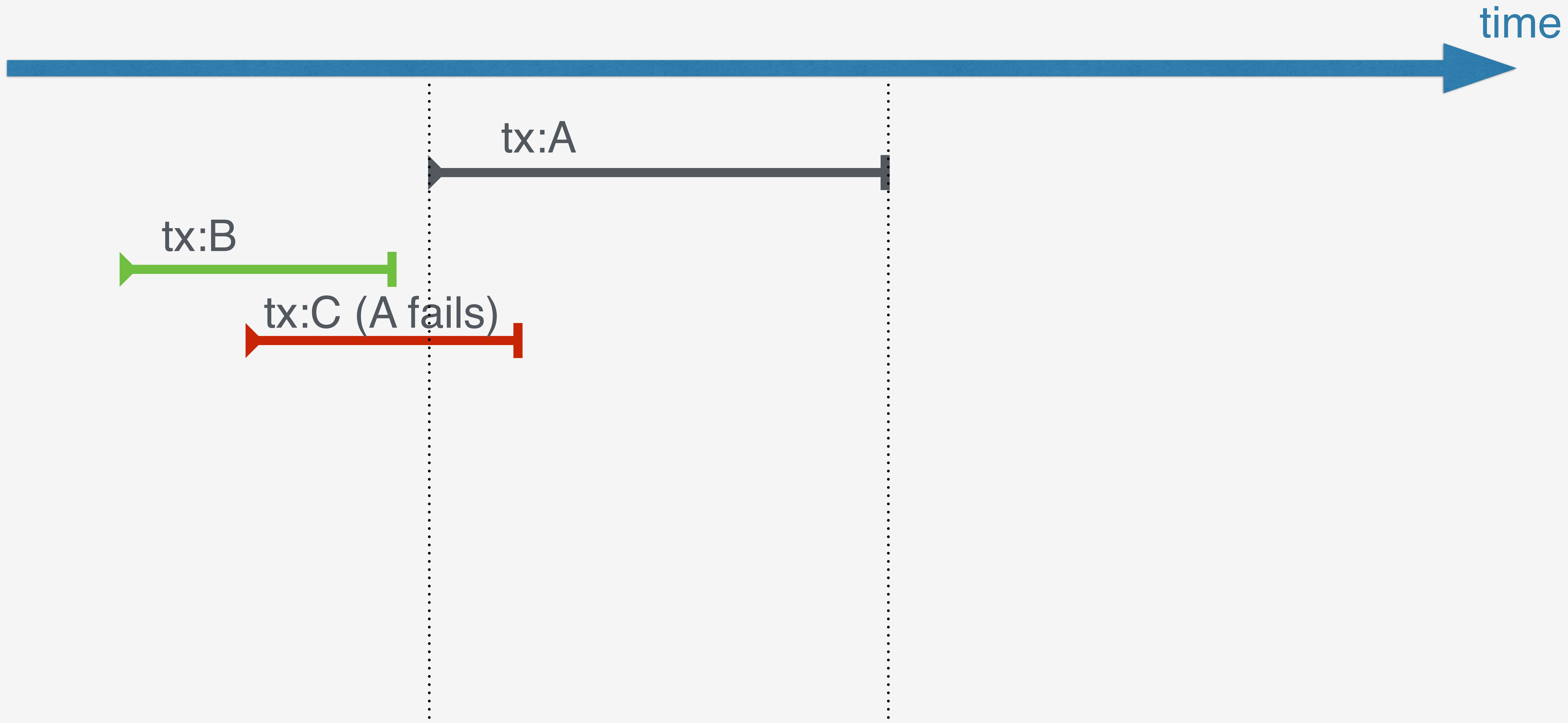
Conflicting Transactions



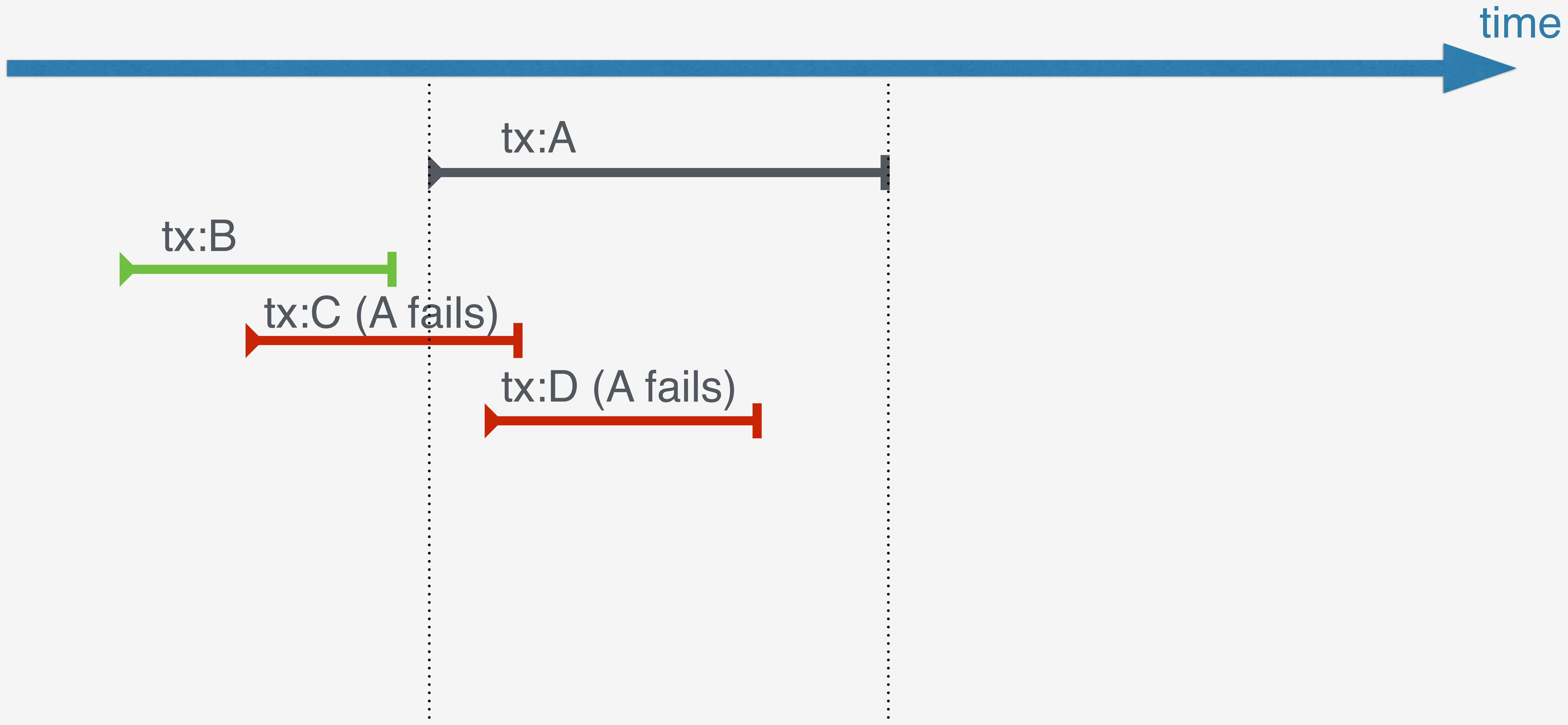
Conflicting Transactions



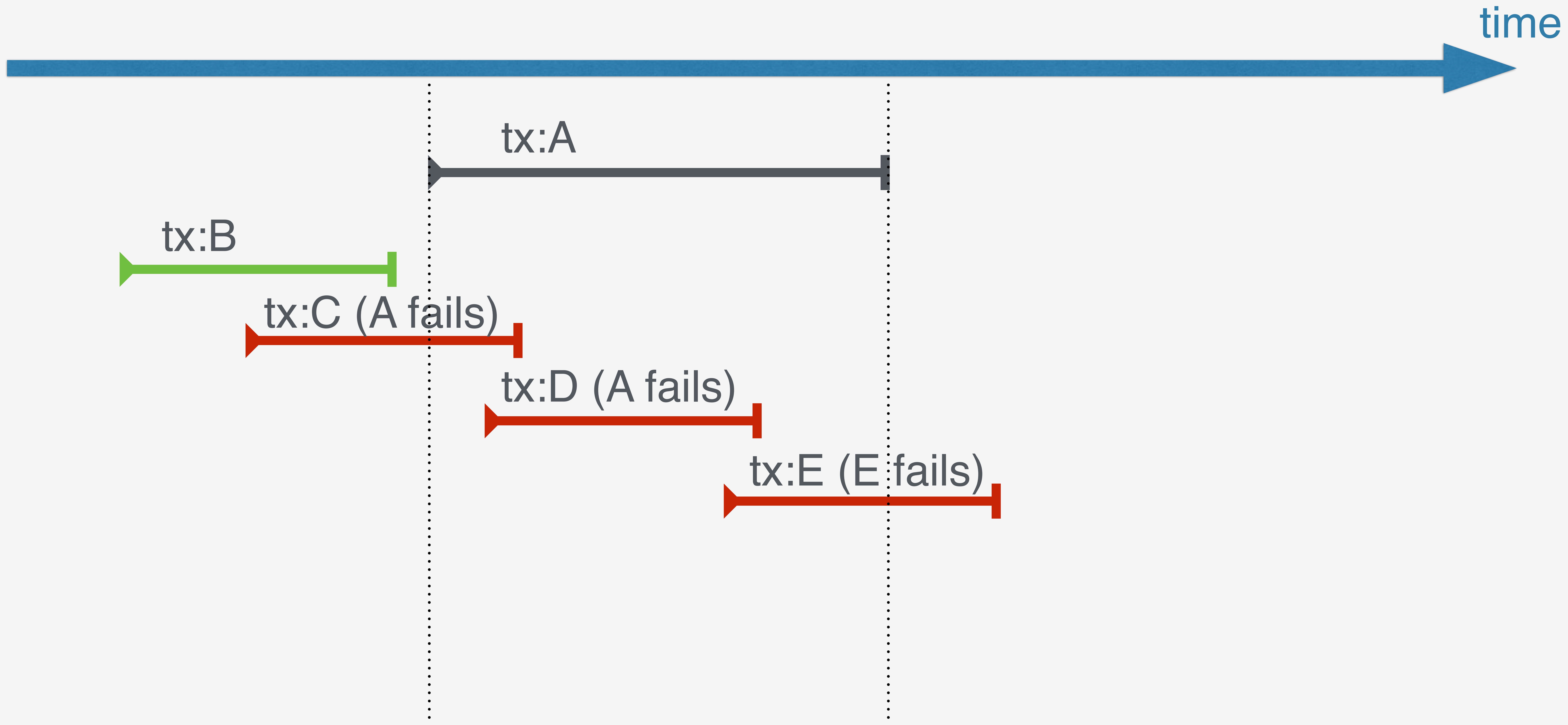
Conflicting Transactions



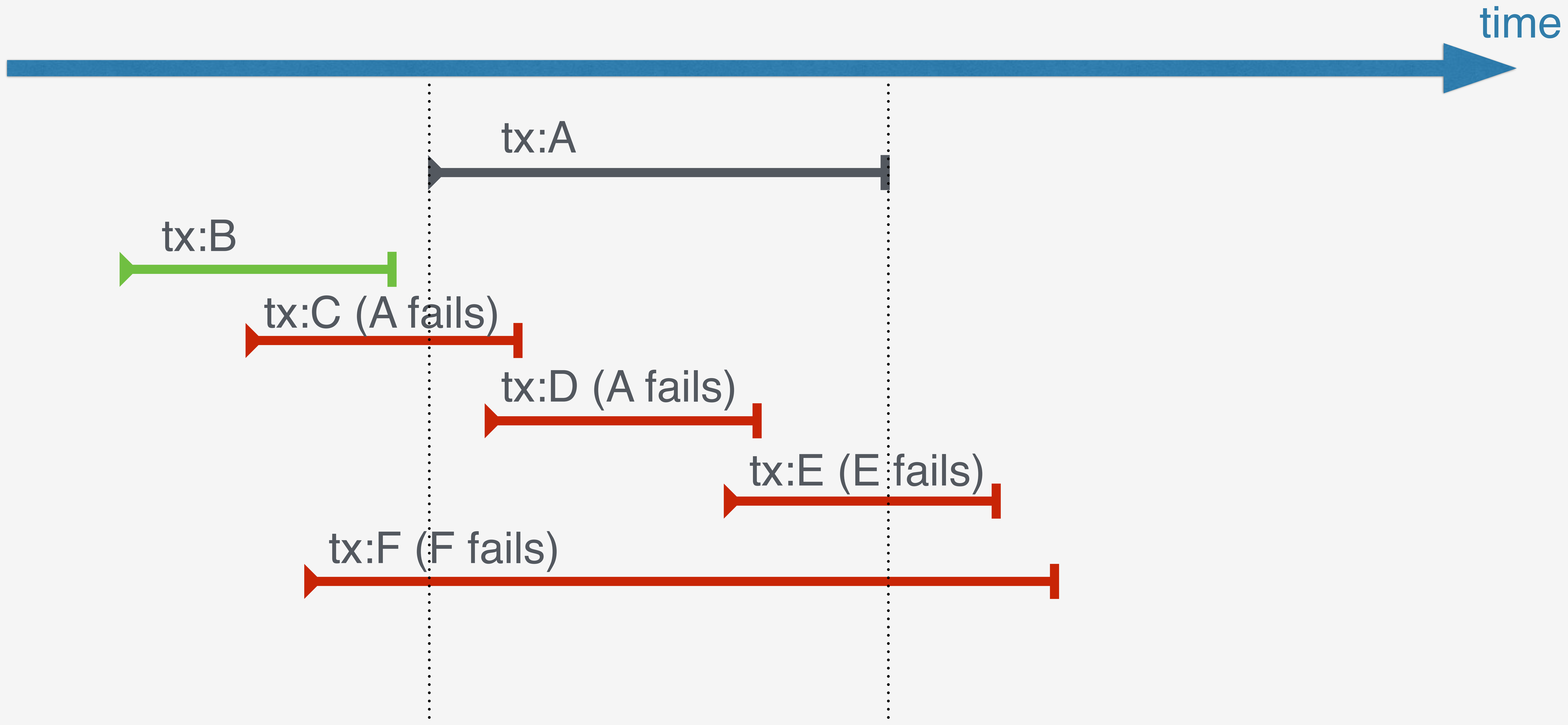
Conflicting Transactions



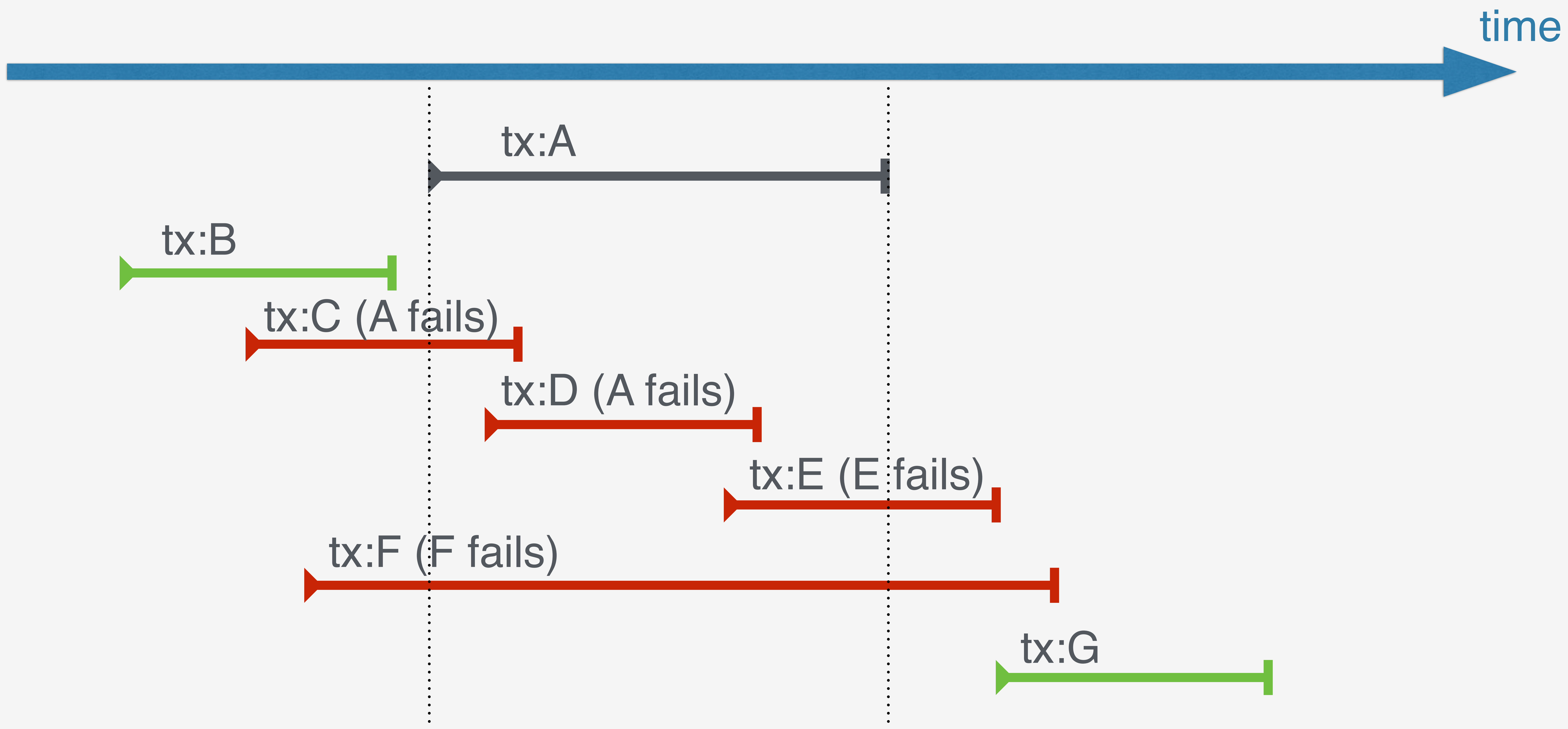
Conflicting Transactions



Conflicting Transactions



Conflicting Transactions



Conflicting Transactions

- Two transactions have a conflict if
  - they write to the same cell
  - they overlap in time
- If two transactions conflict, the one that commits later rolls back
- Active change set = set of transactions t such that:
  - t is committed, and
  - there is at least one in-flight tx t’ that started before t’s commit time
- This change set is needed in order to perform conflict detection.

HBase Transactions in Apache

Apache Tephra

(incubating)

Apache Omid (incubating)

Apache Trafodion

(incubating)

In Common

- Optimistic Concurrency Control must:
  - maintain Transaction State:
    - what tx are in flight and committed?
    - what is the change set of each tx? (for conflict detection, rollback)
    - what transactions are invalid (failed to roll back due to crash etc.)
  - generate unique transaction IDs
  - coordinate the life cycle of a transaction
    - start, detect conflicts, commit, rollback

- All of { Omid, Tephra, Trafodion } implement this
  - but vary in how they do it

Apache Tephra

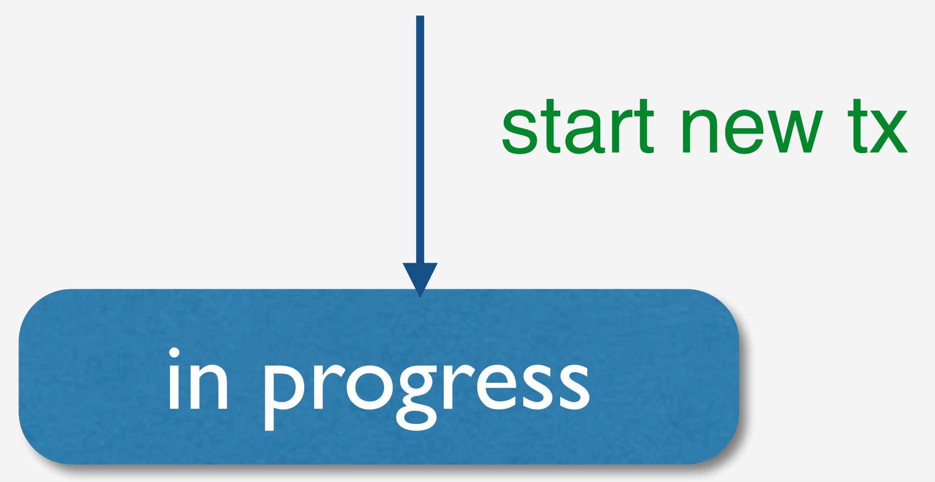
- Based on the original Omid paper:

  Daniel Gómez Ferro, Flavio Junqueira, Ivan Kelly, Benjamin Reed, Maysam Yabandeh: *Omid: Lock-free transactional support for distributed data stores*. ICDE 2014.

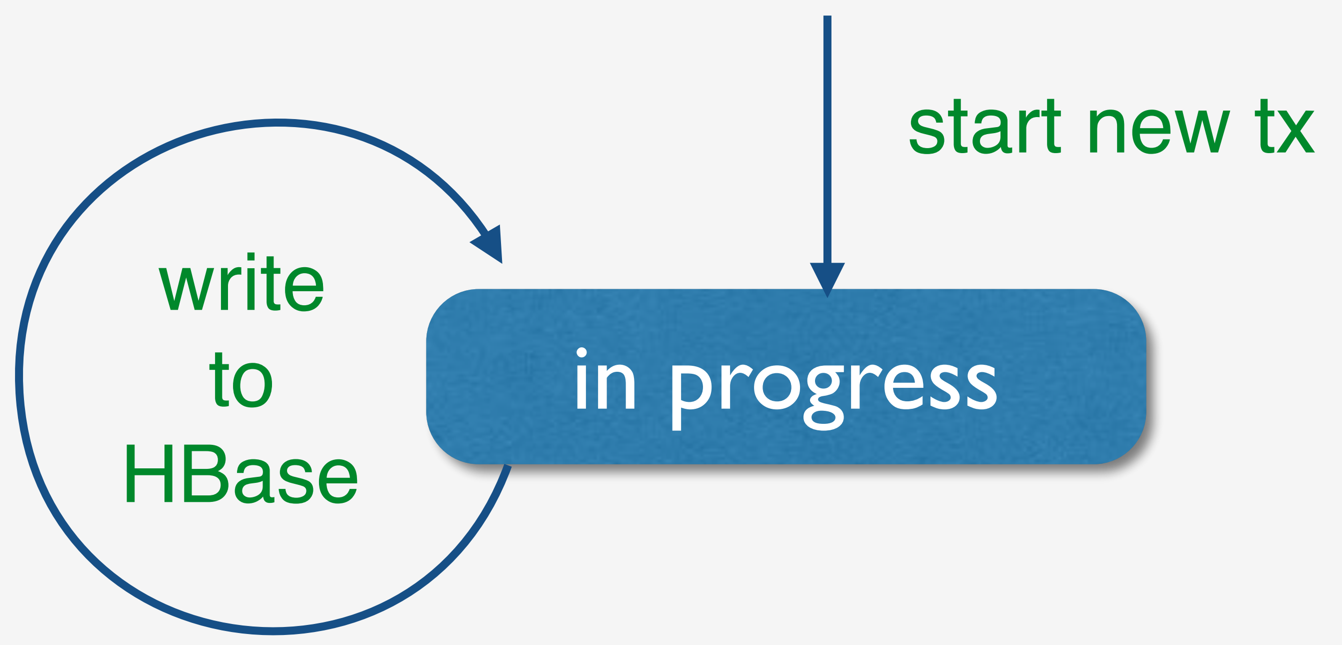
- Transaction Manager:
  - Issues unique, monotonic transaction IDs
  - Maintains the set of excluded (in-flight and invalid) transactions
  - Maintains change sets for active transactions
  - Performs conflict detection
- Client:
  - Uses transaction ID as timestamp for writes
  - Filters excluded transactions for isolation
  - Performs rollback

Transaction Lifecycle

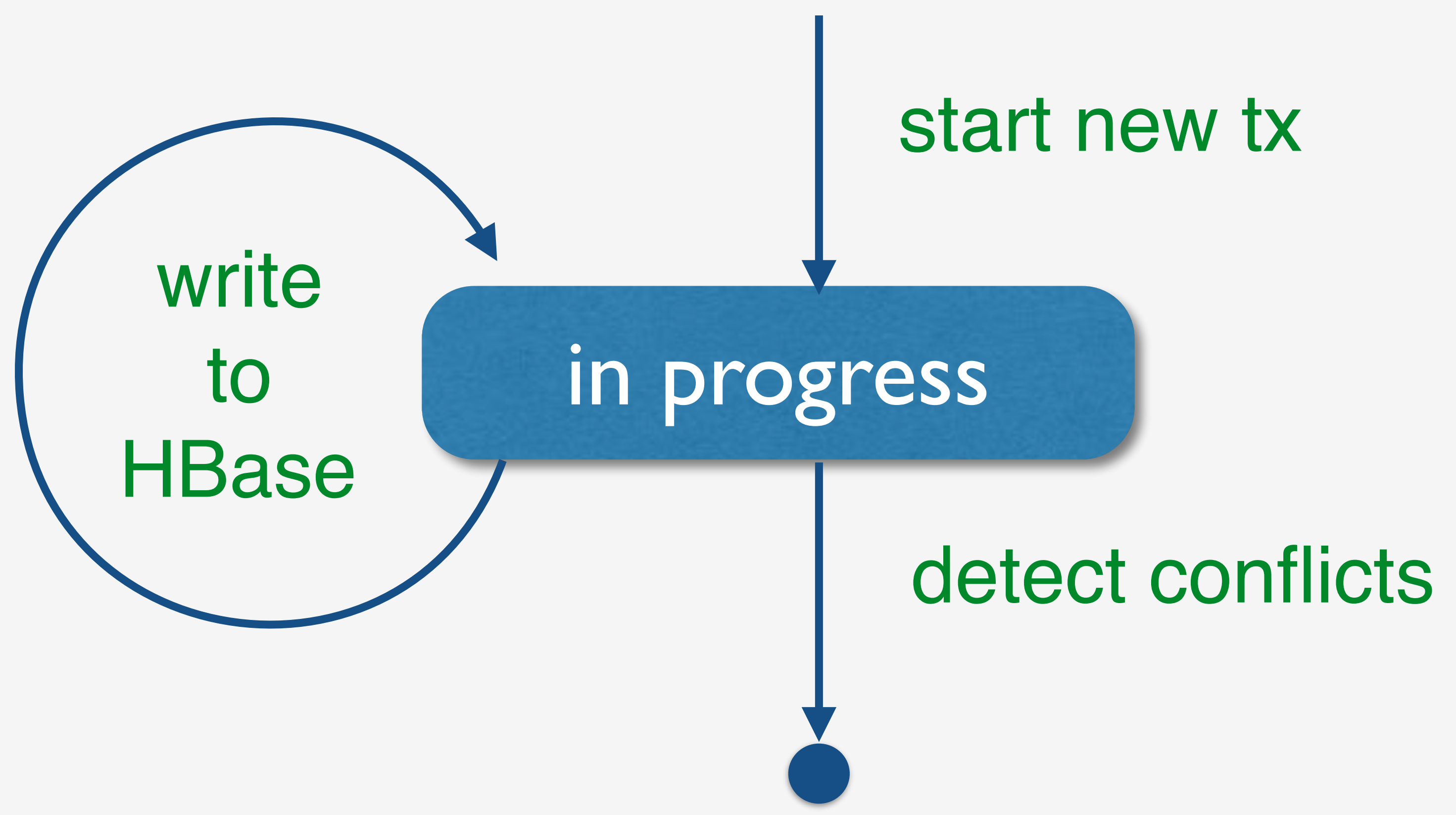
Transaction Lifecycle



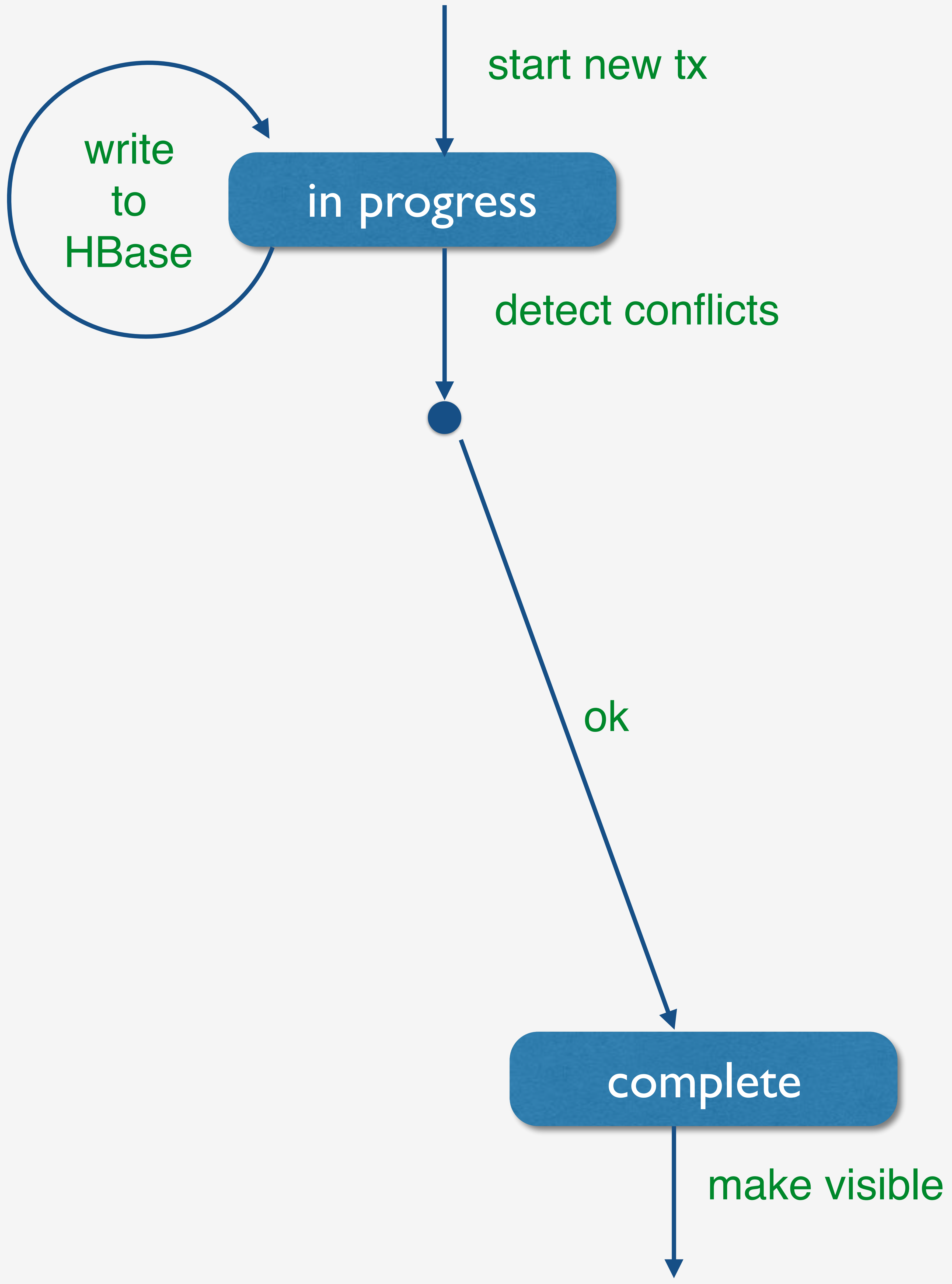
Transaction Lifecycle



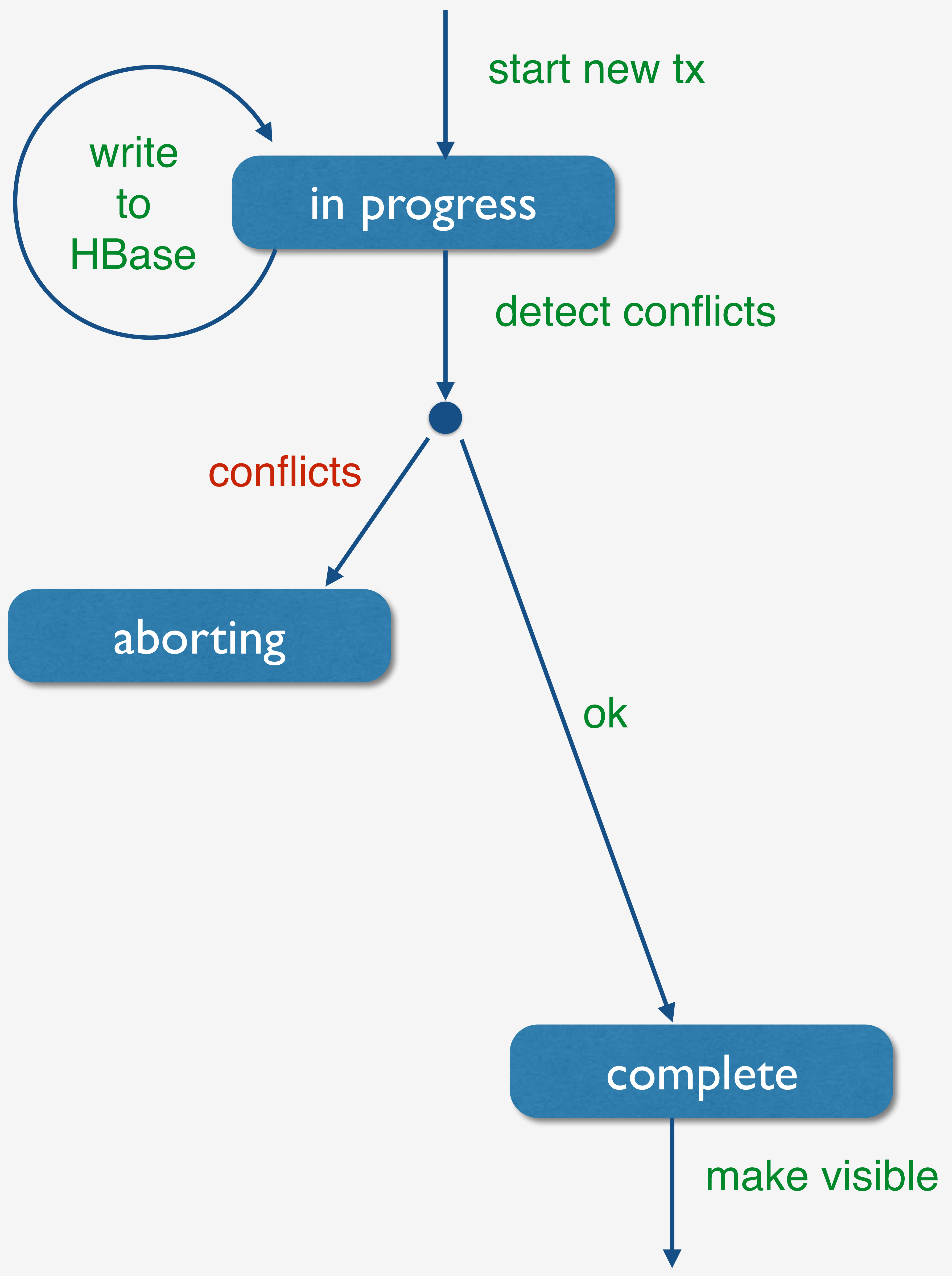
Transaction Lifecycle



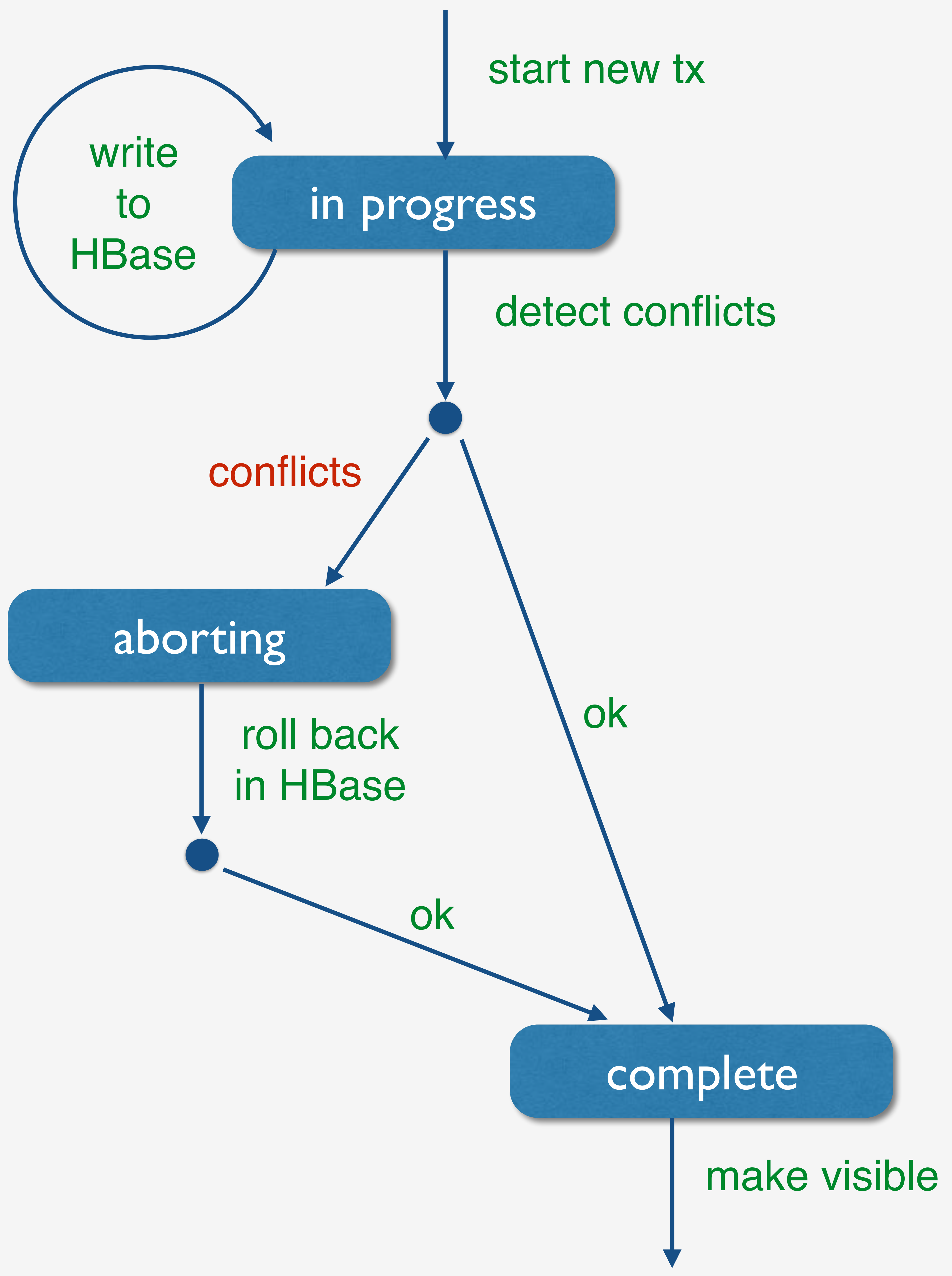
Transaction Lifecycle



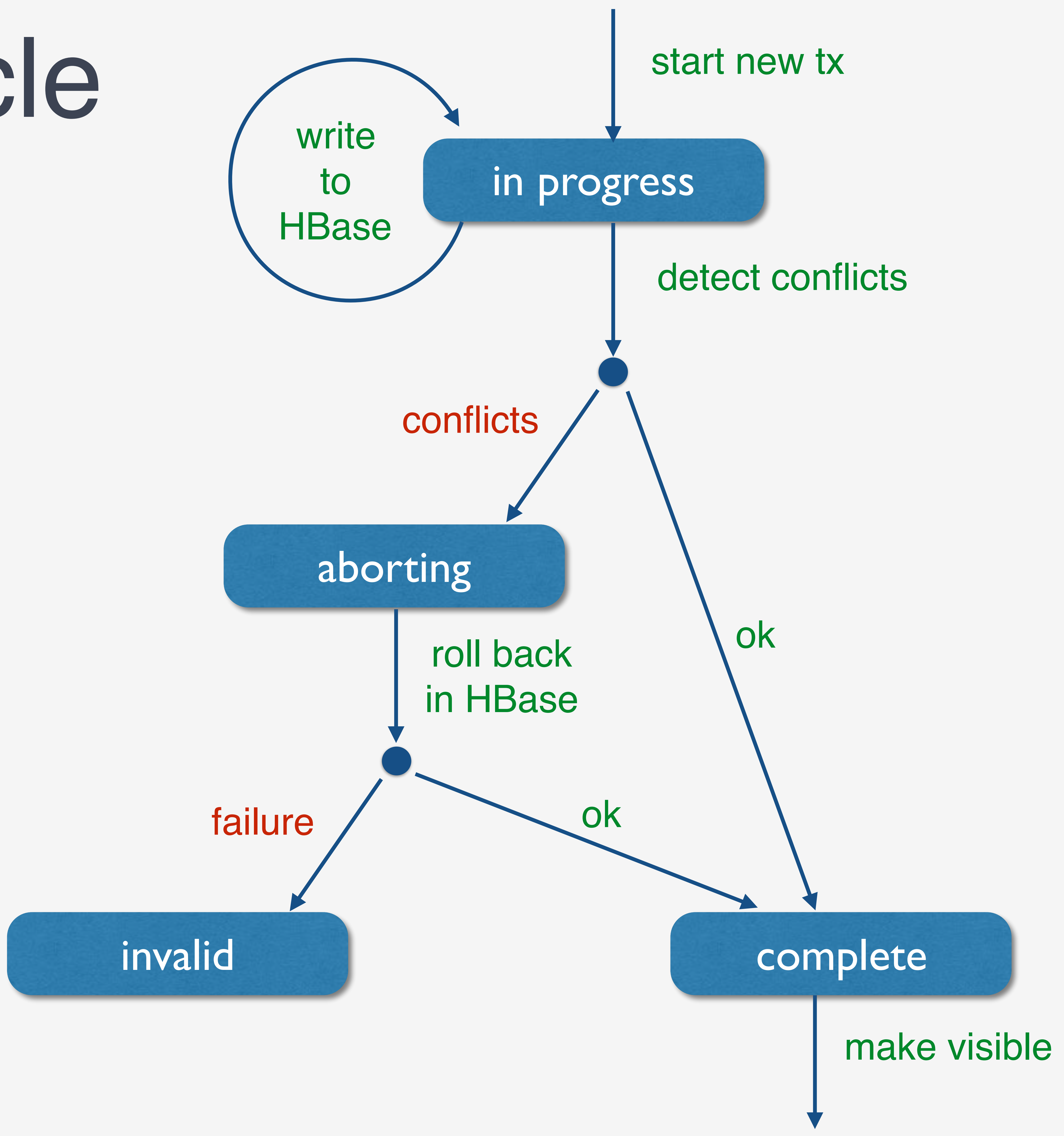
Transaction Lifecycle



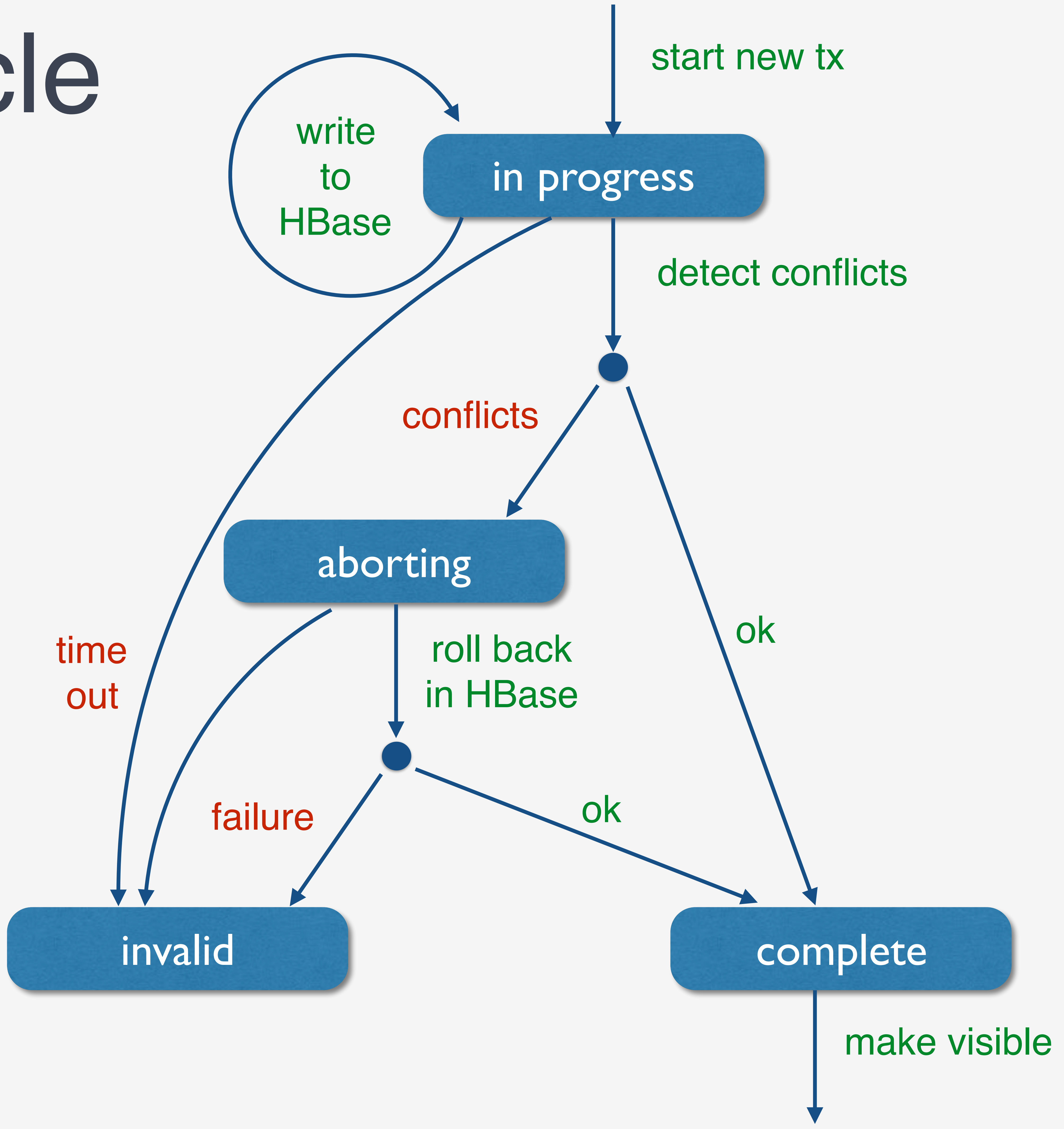
Transaction Lifecycle



Transaction Lifecycle

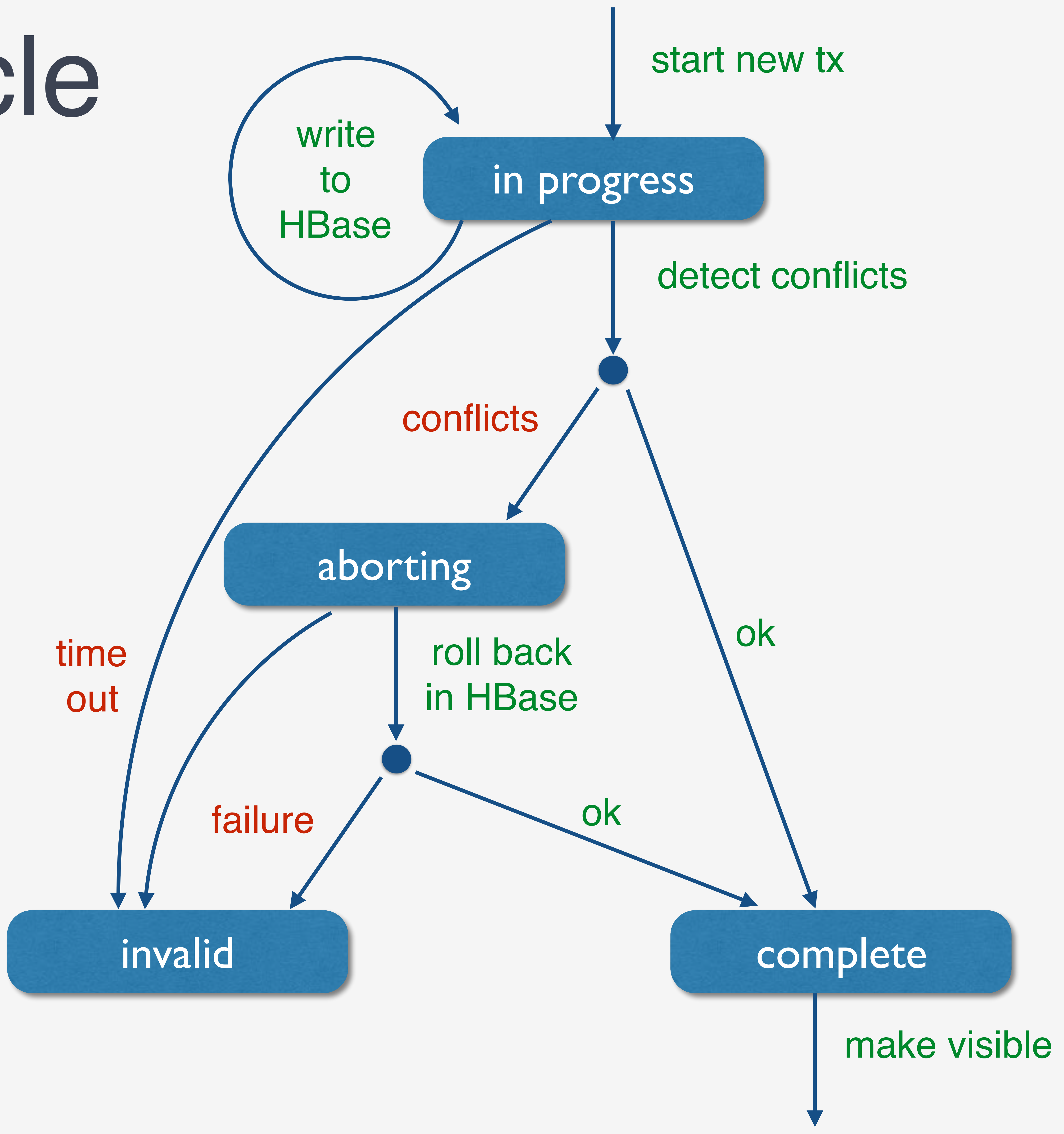


Transaction Lifecycle

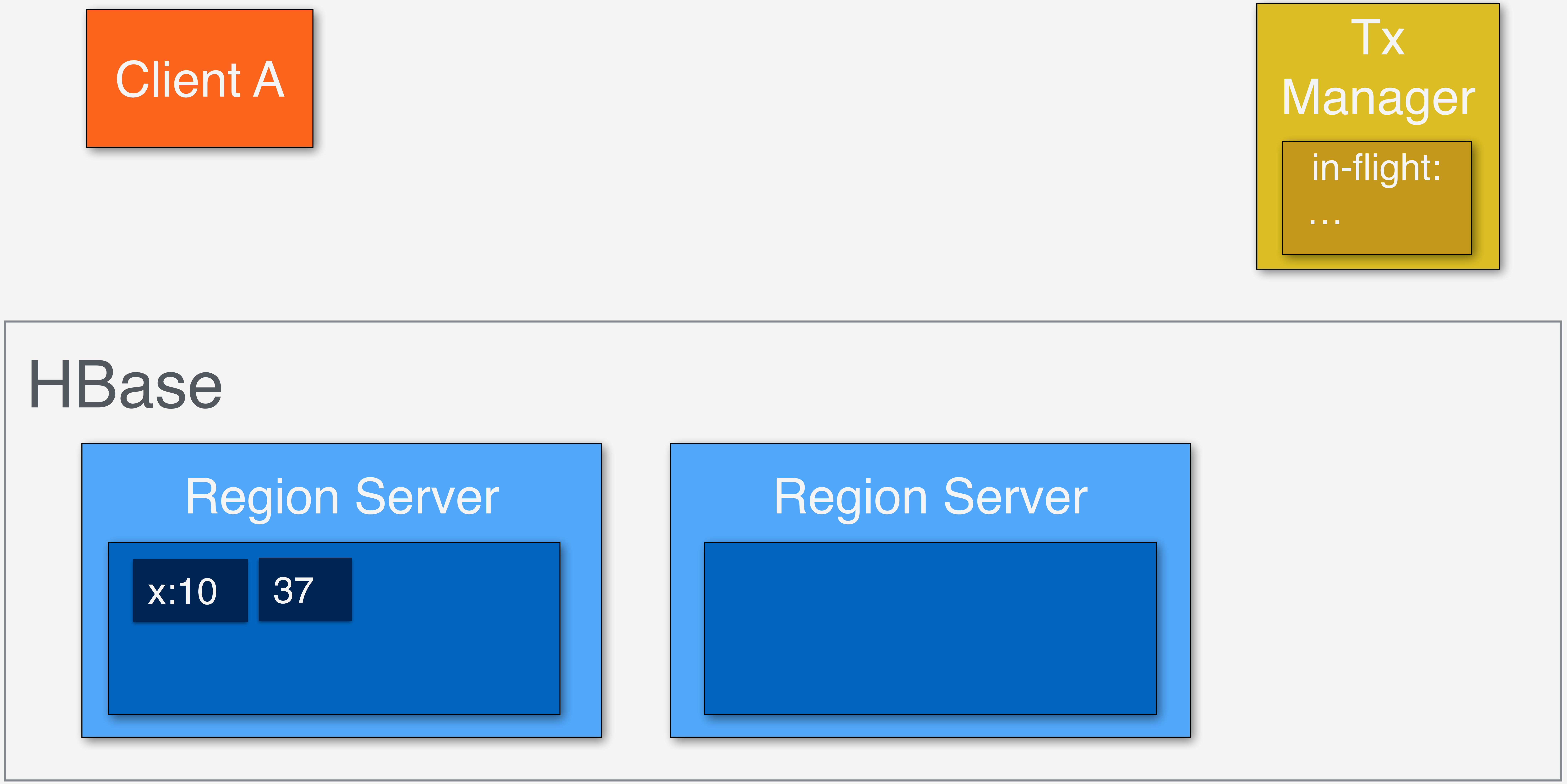


Transaction Lifecycle

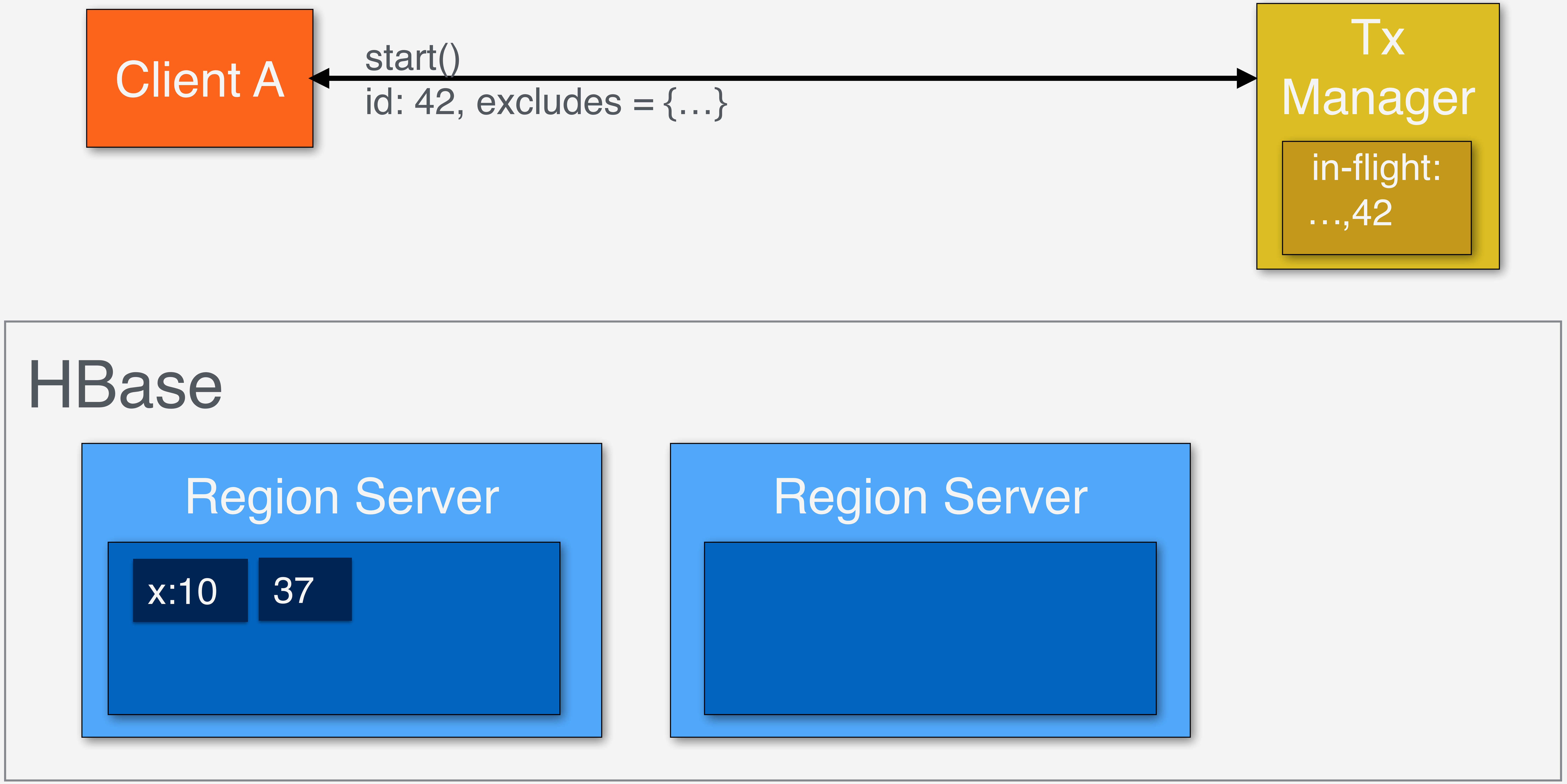
- Transaction consists of:
    - transaction ID (unique timestamp)
    - exclude list (in-flight and invalid tx)
- Transactions that do complete
    - must still participate in conflict detection
    - disappear from transaction state when they do not overlap with in-flight tx
- Transactions that do not complete
    - time out (by transaction manager)
    - added to invalid list



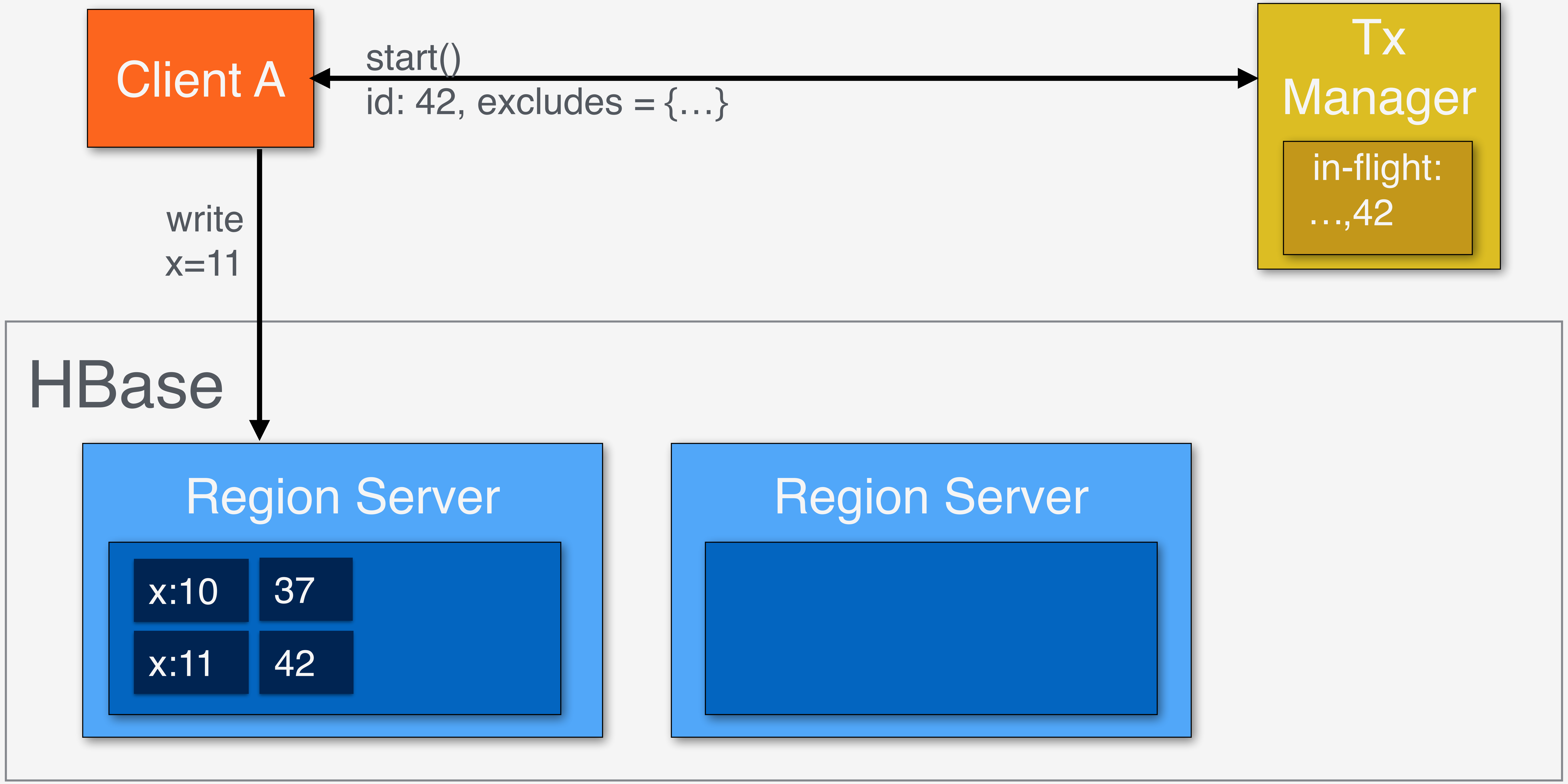
Apache Tephra



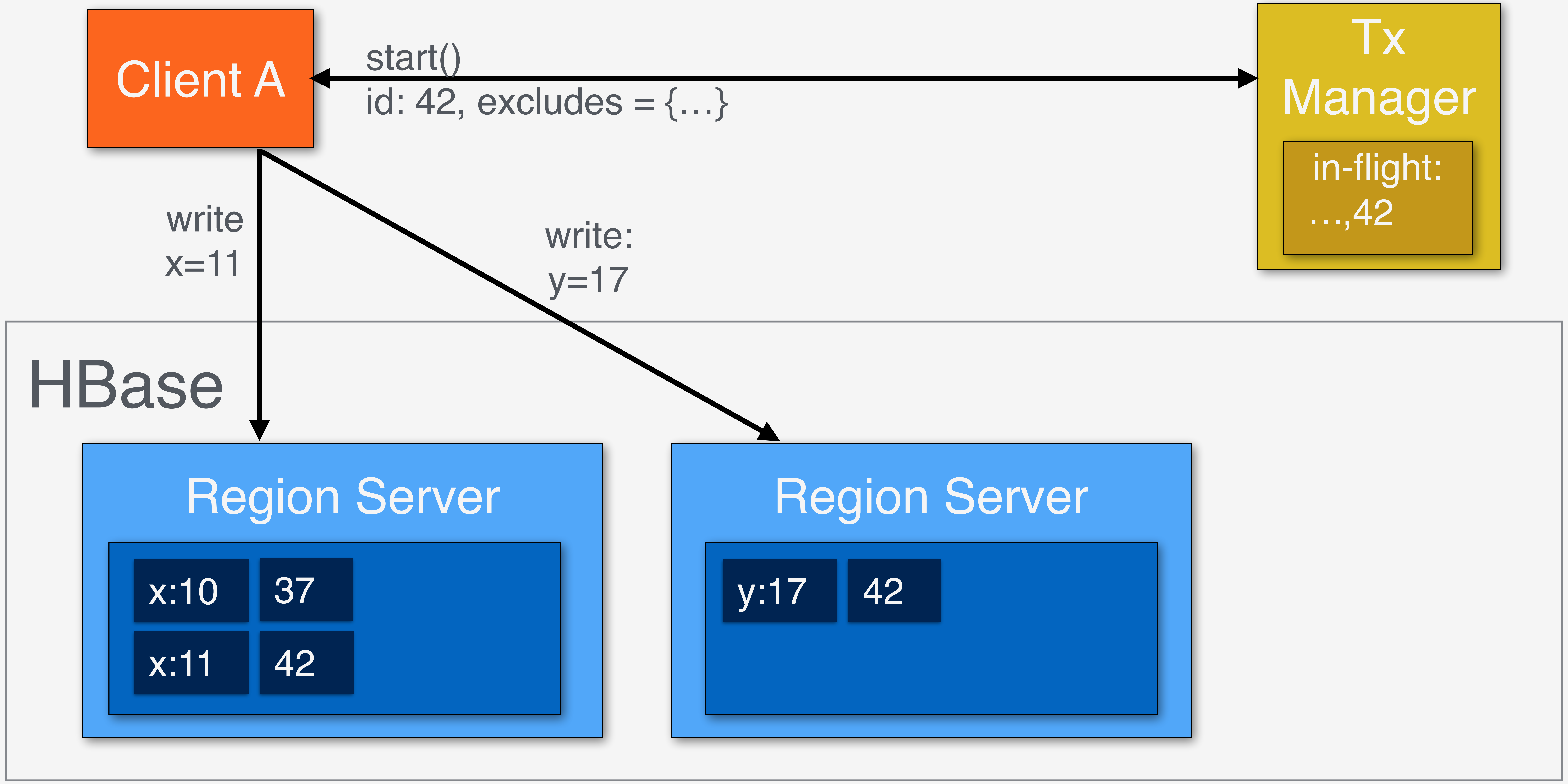
Apache Tephra



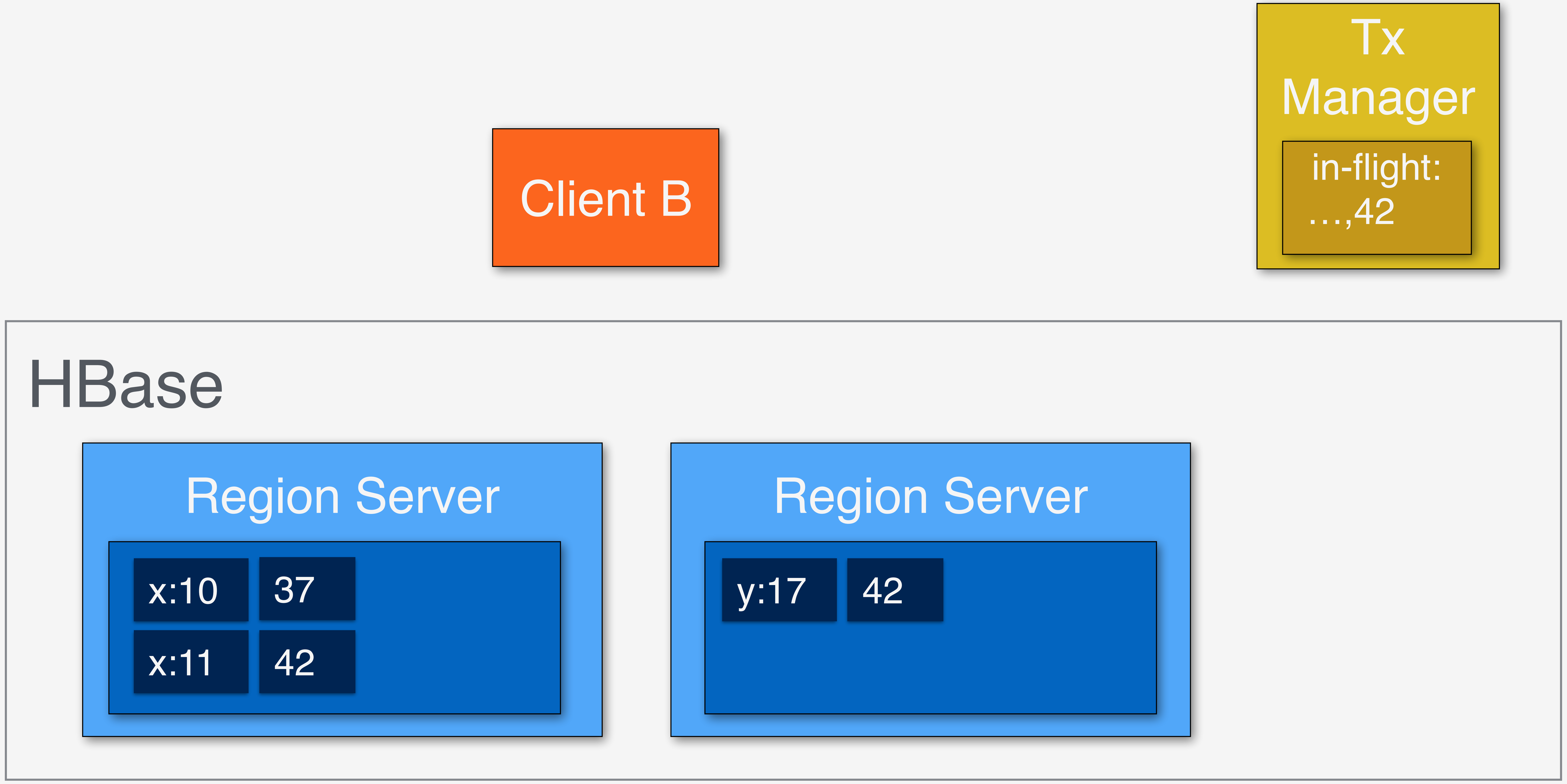
Apache Tephra



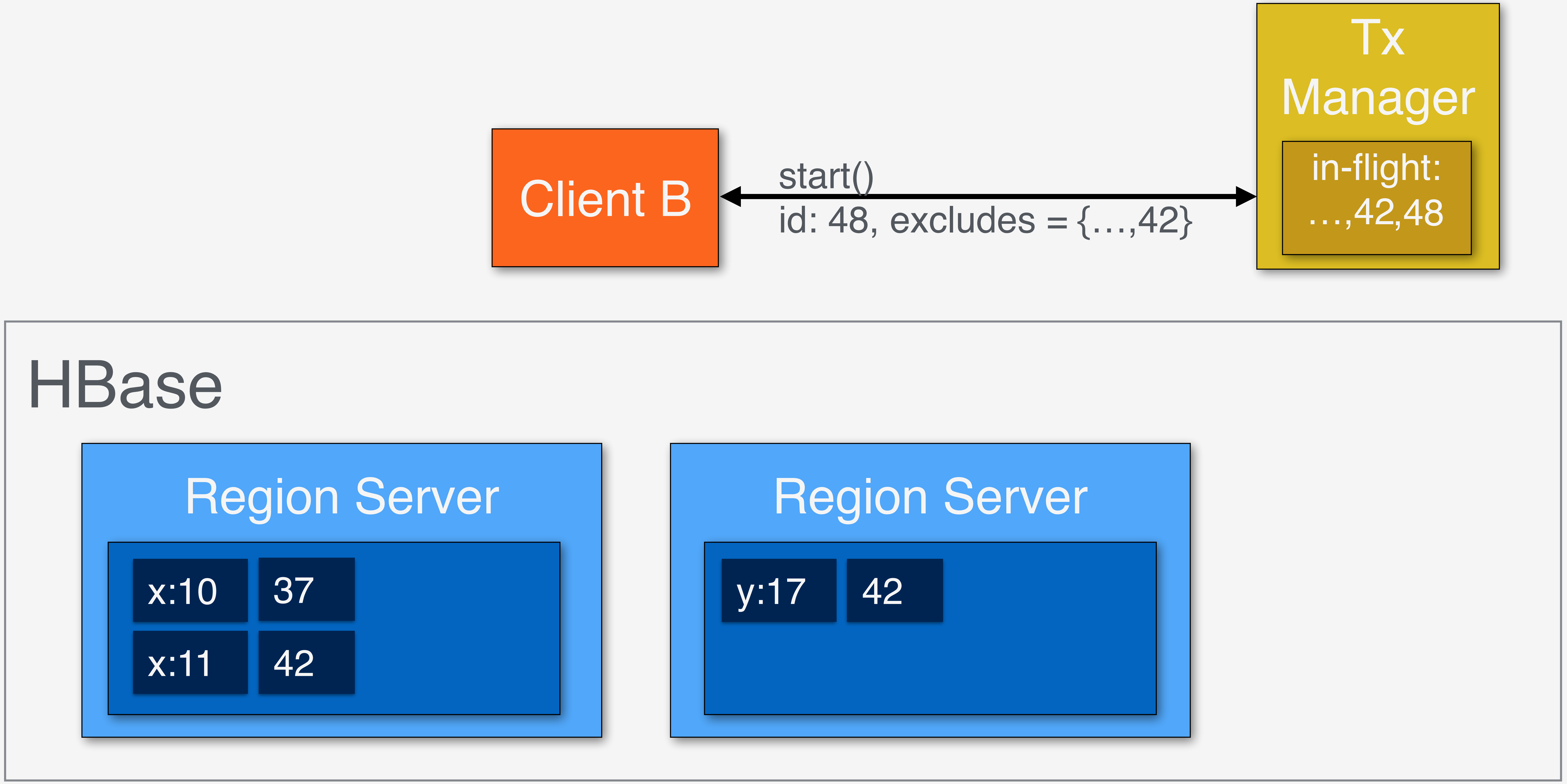
Apache Tephra



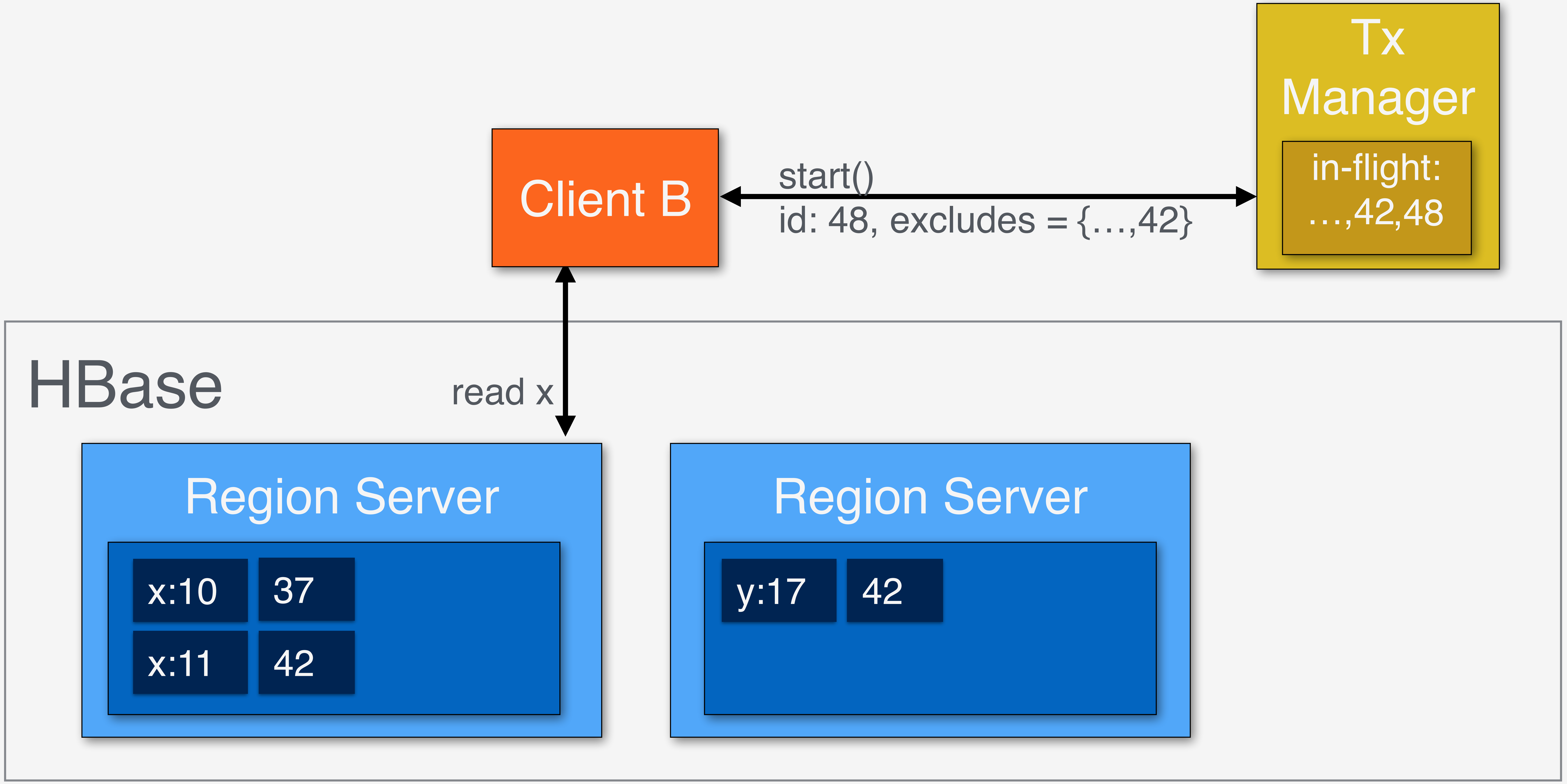
Apache Tephra



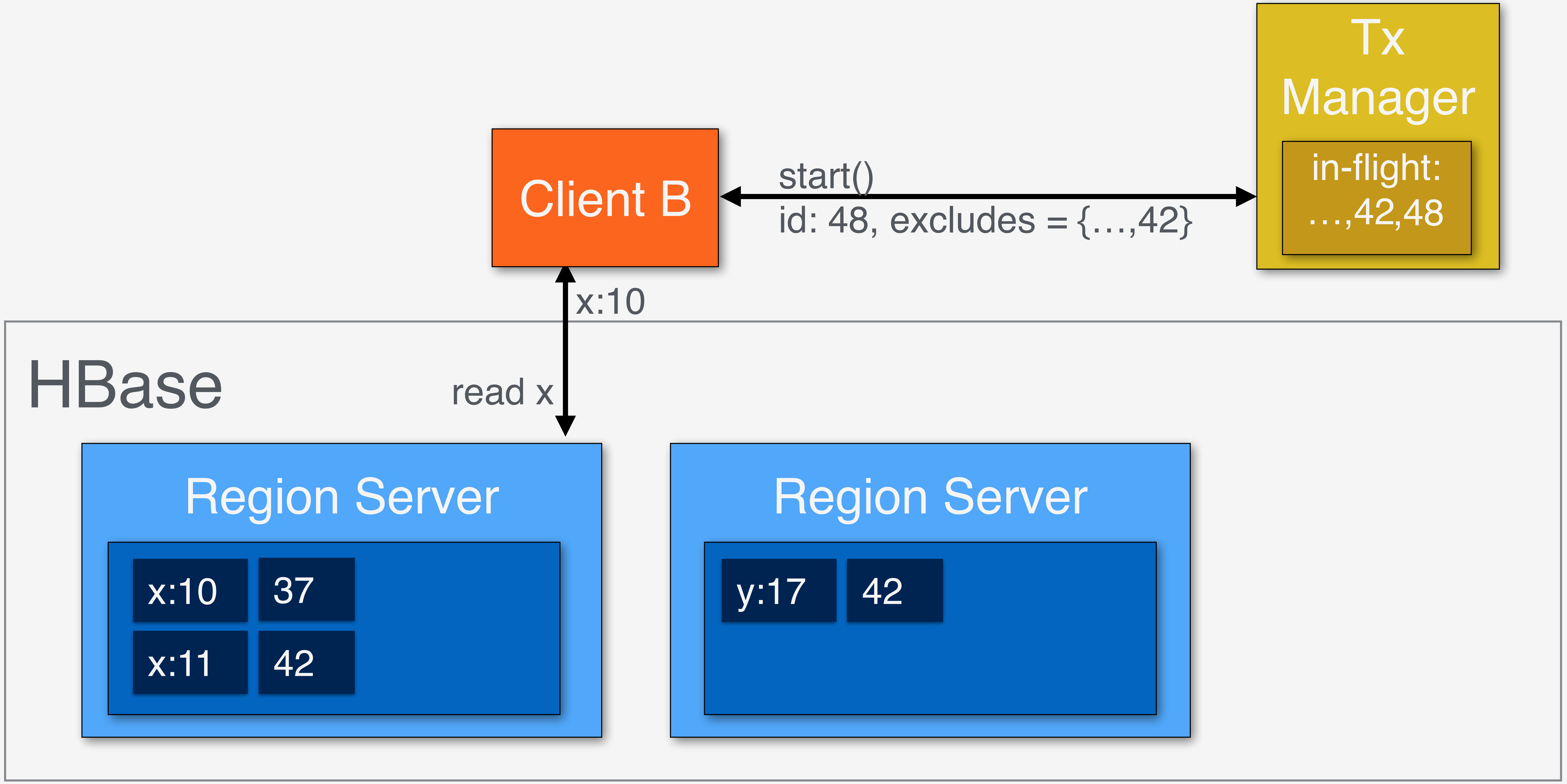
Apache Tephra



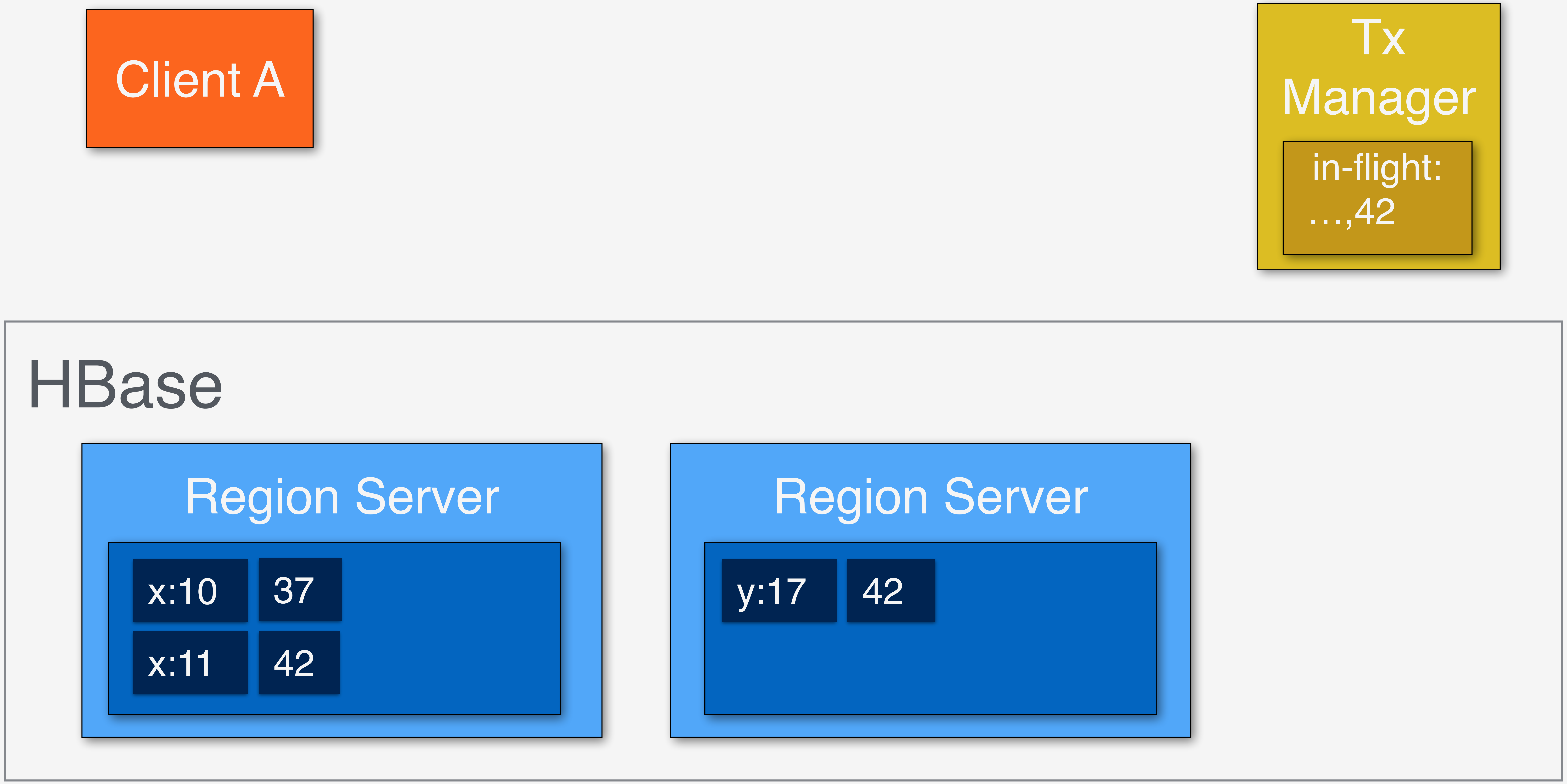
Apache Tephra



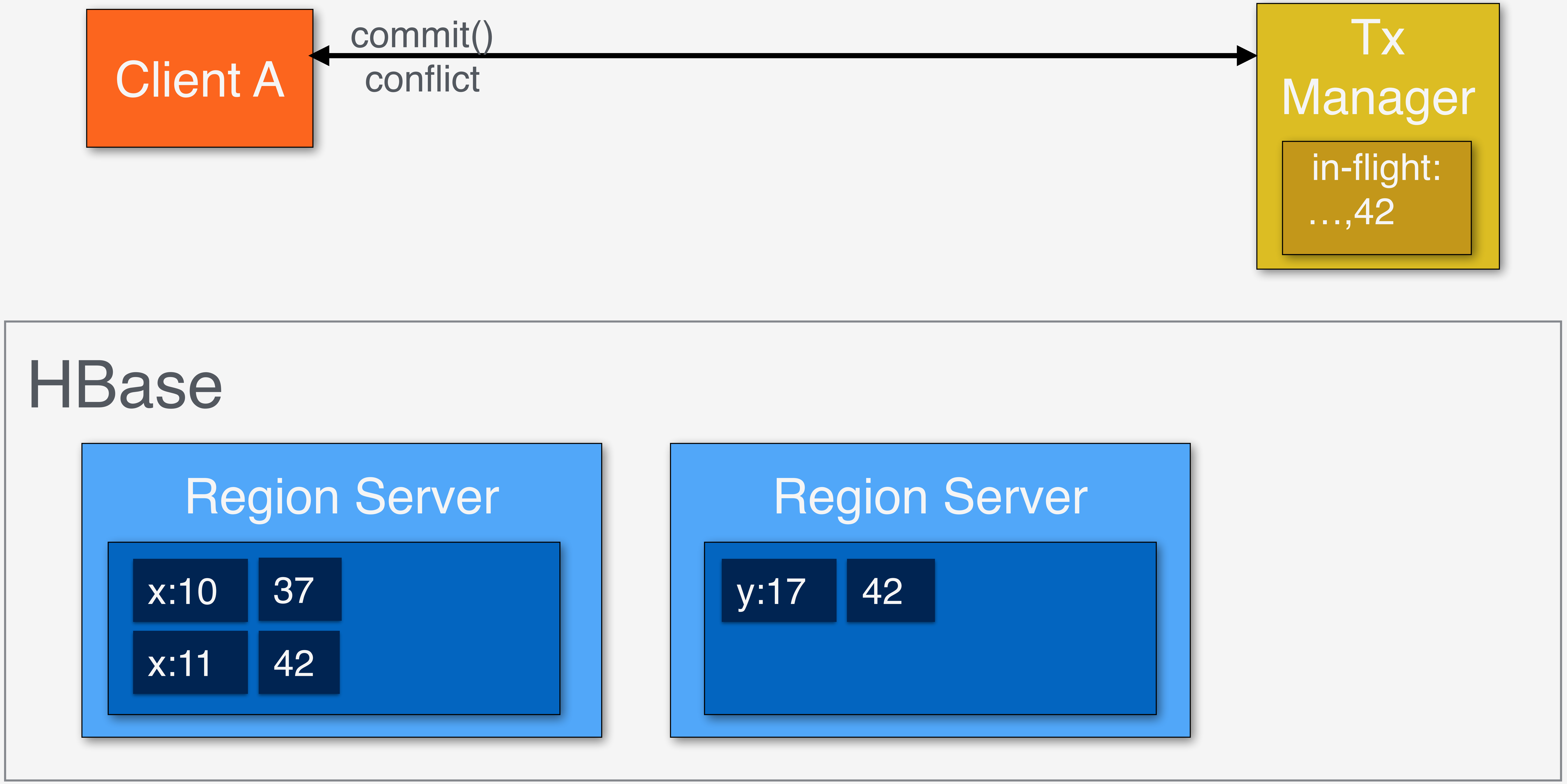
Apache Tephra



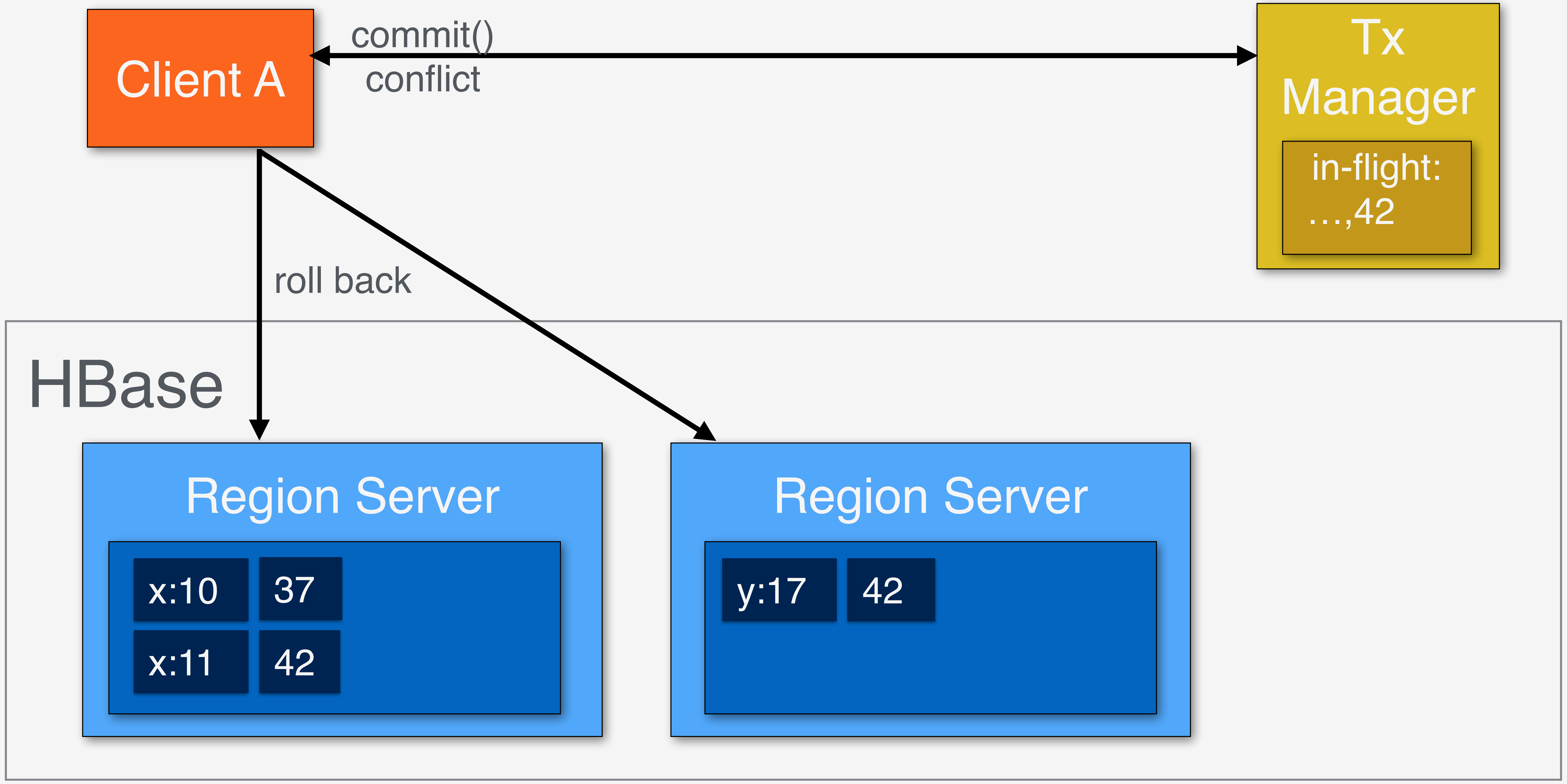
Apache Tephra



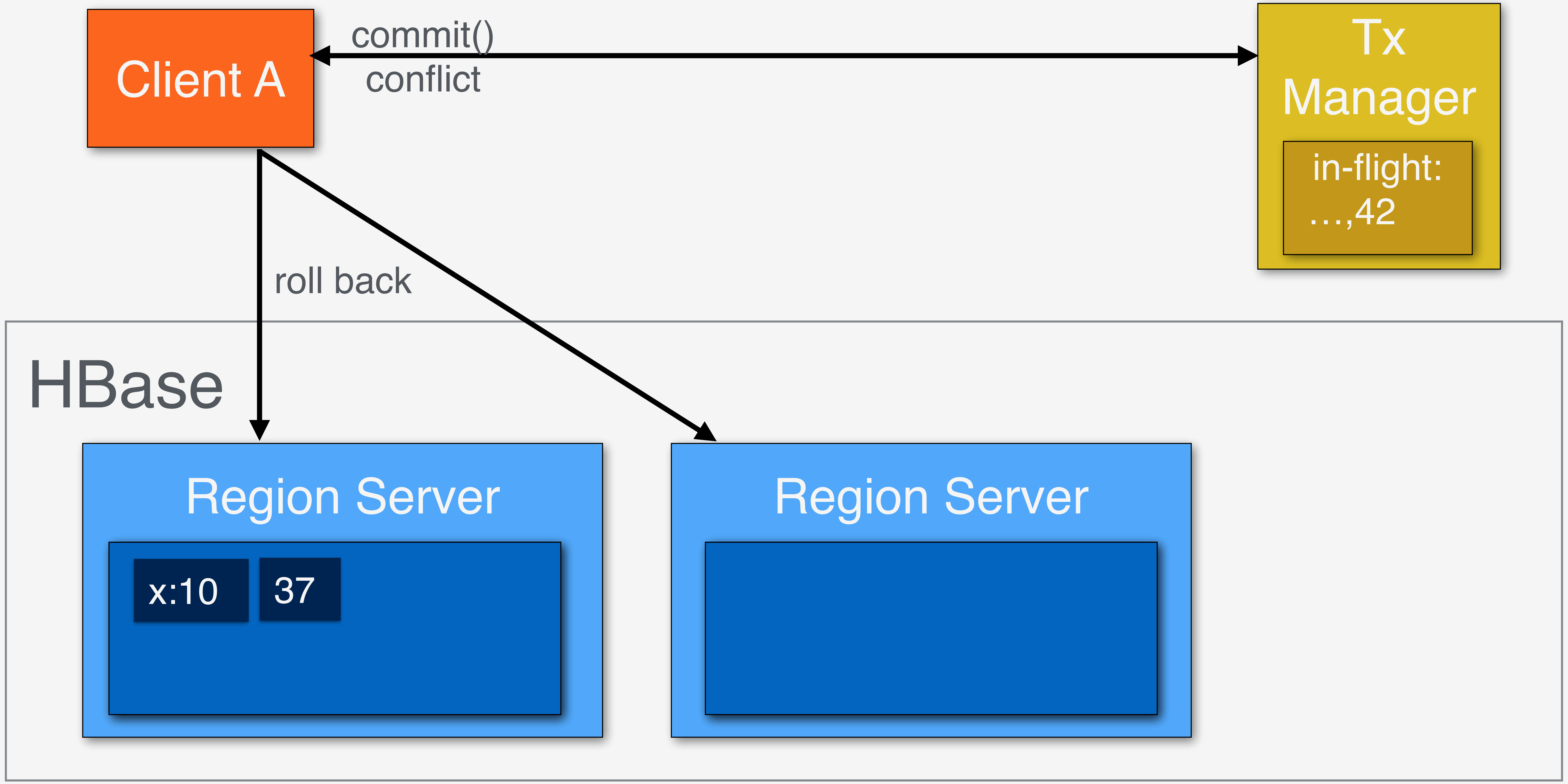
Apache Tephra



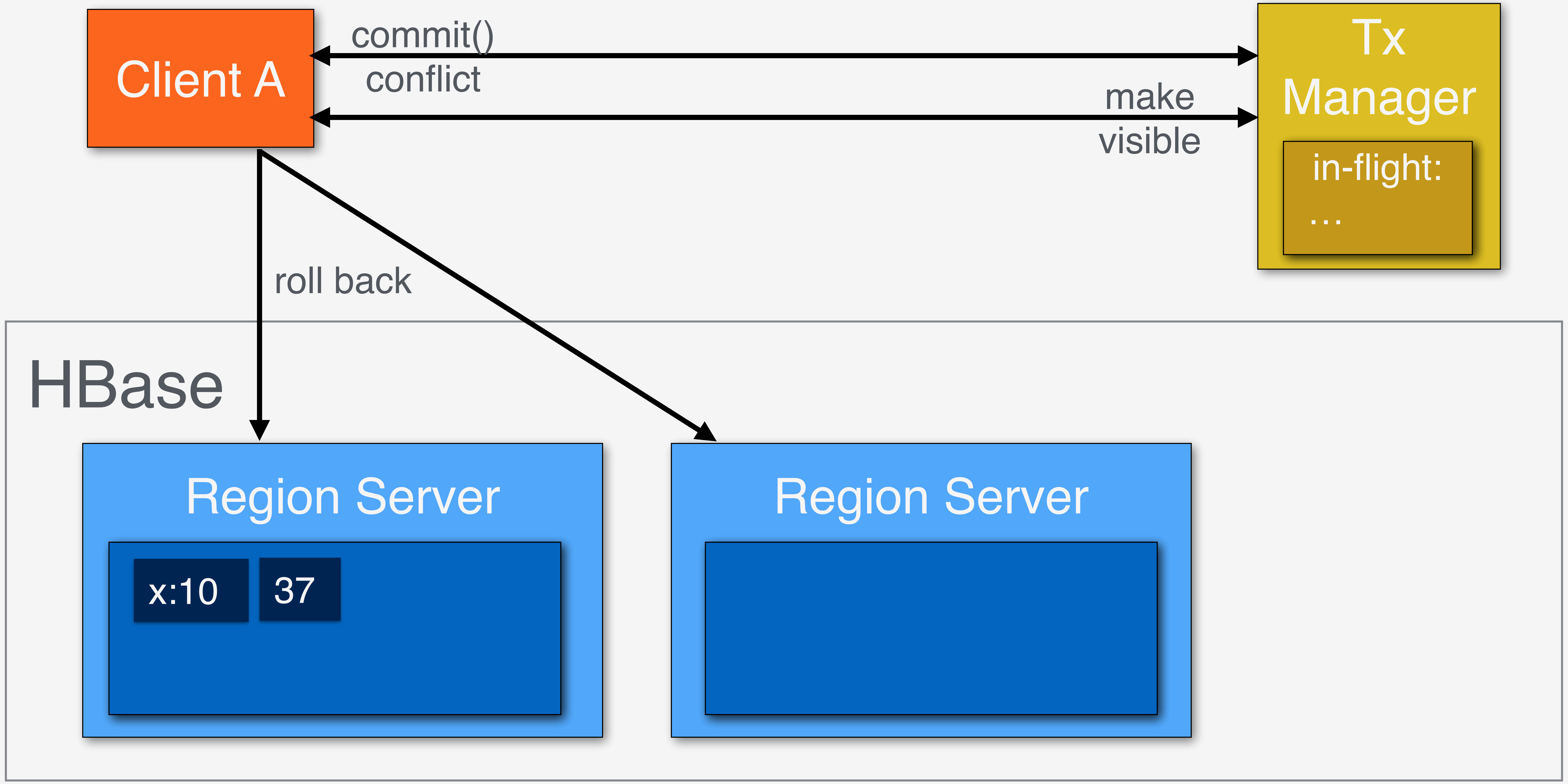
Apache Tephra



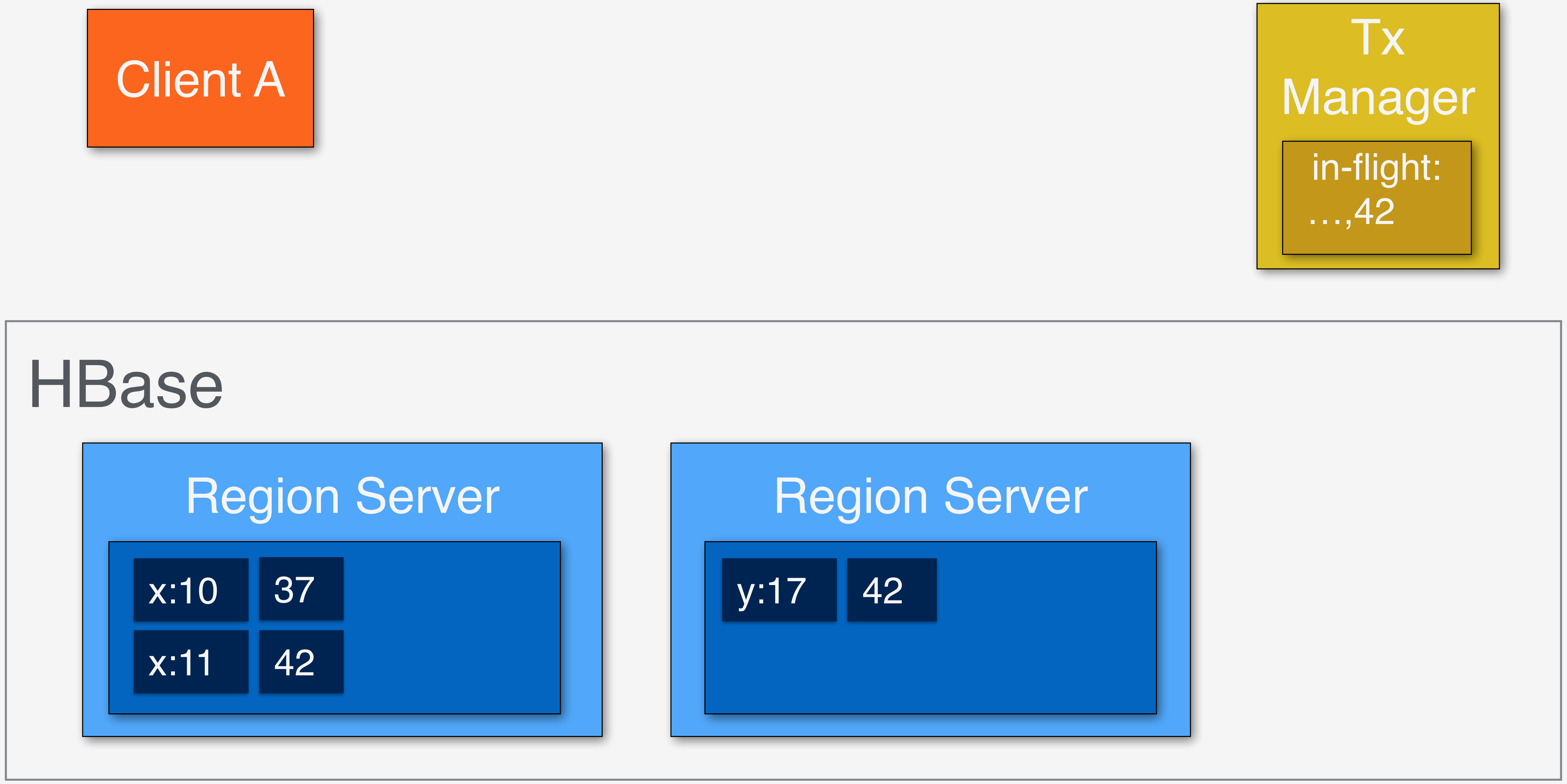
Apache Tephra



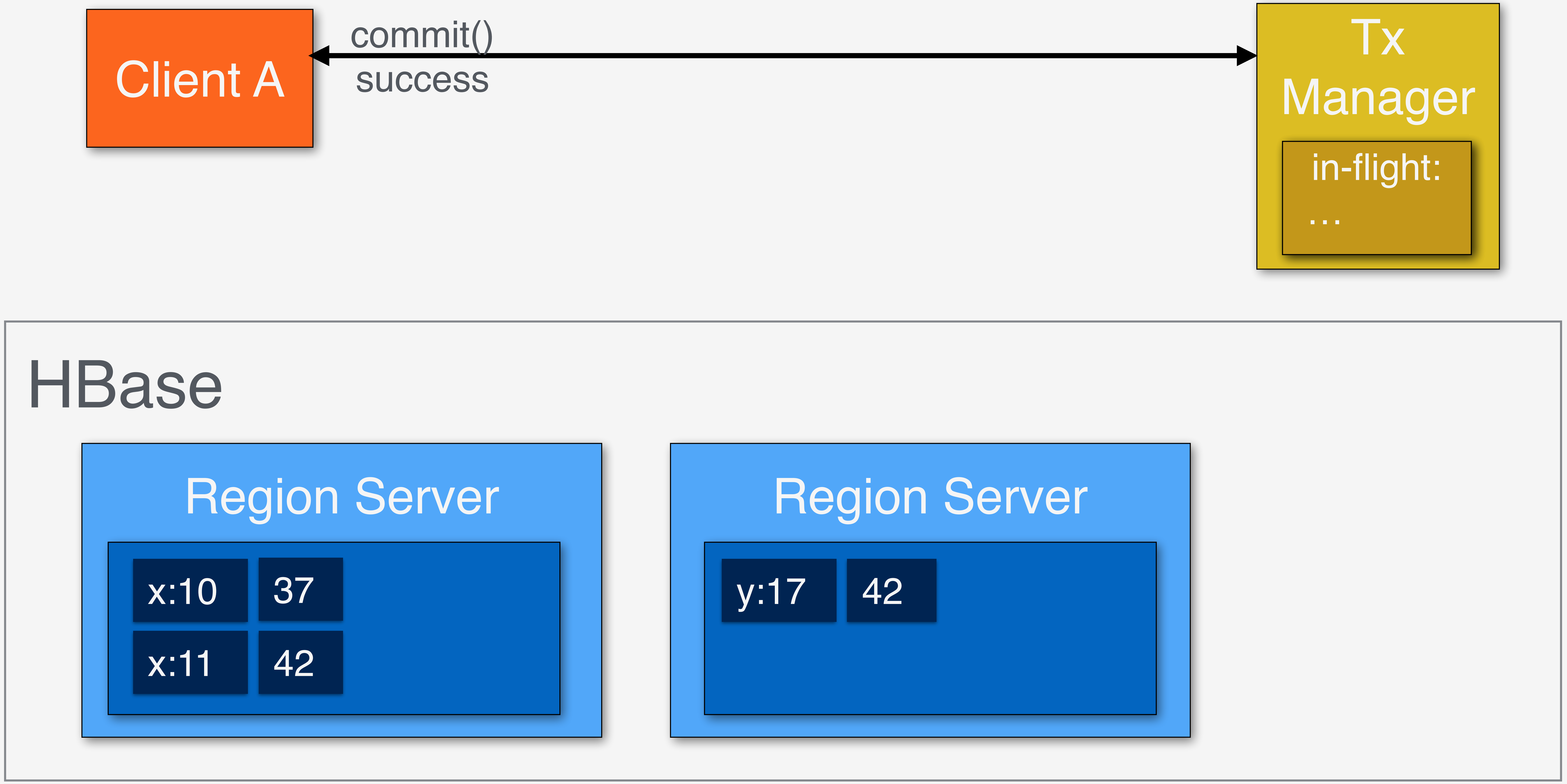
Apache Tephra



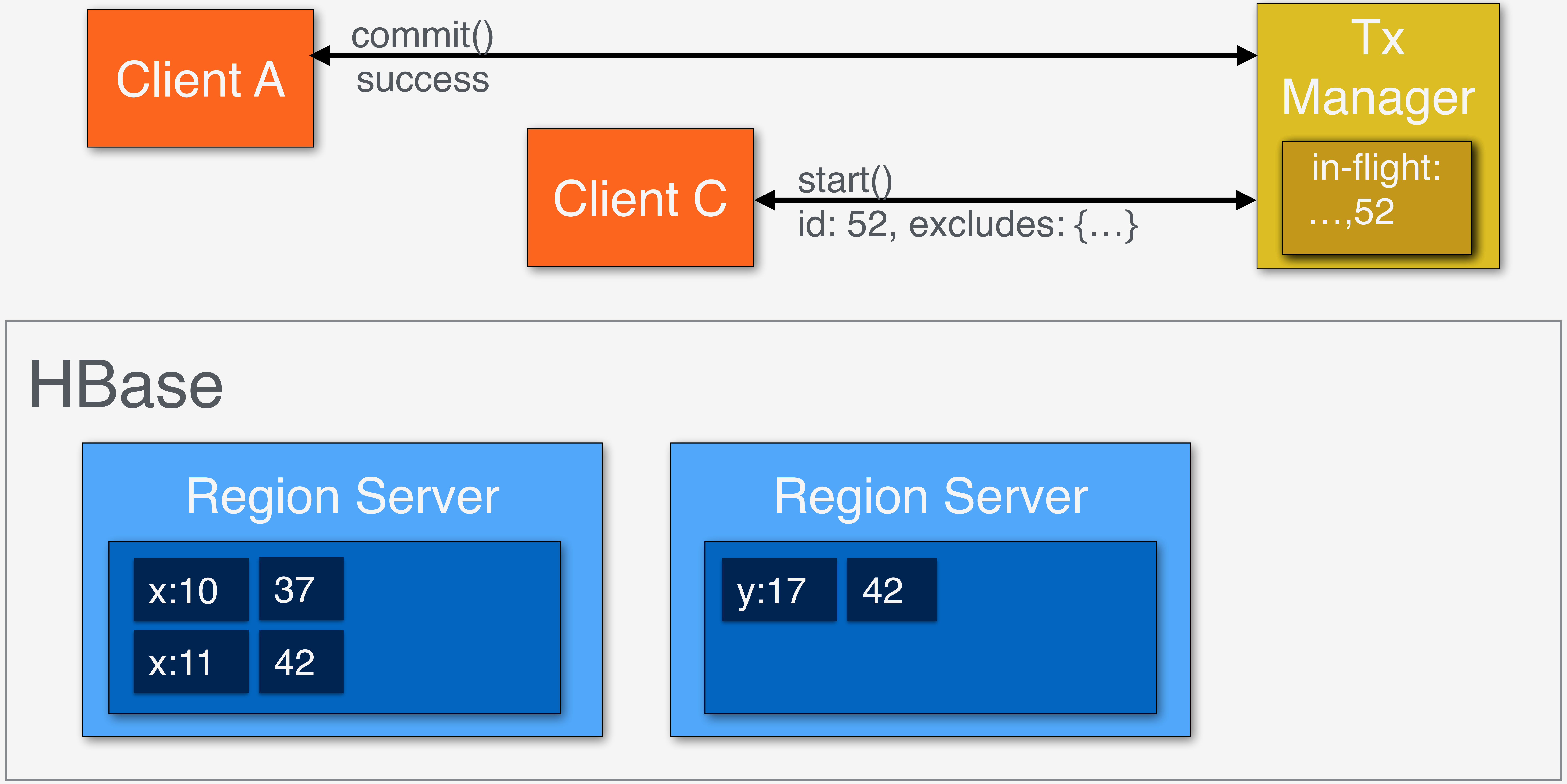
Apache Tephra



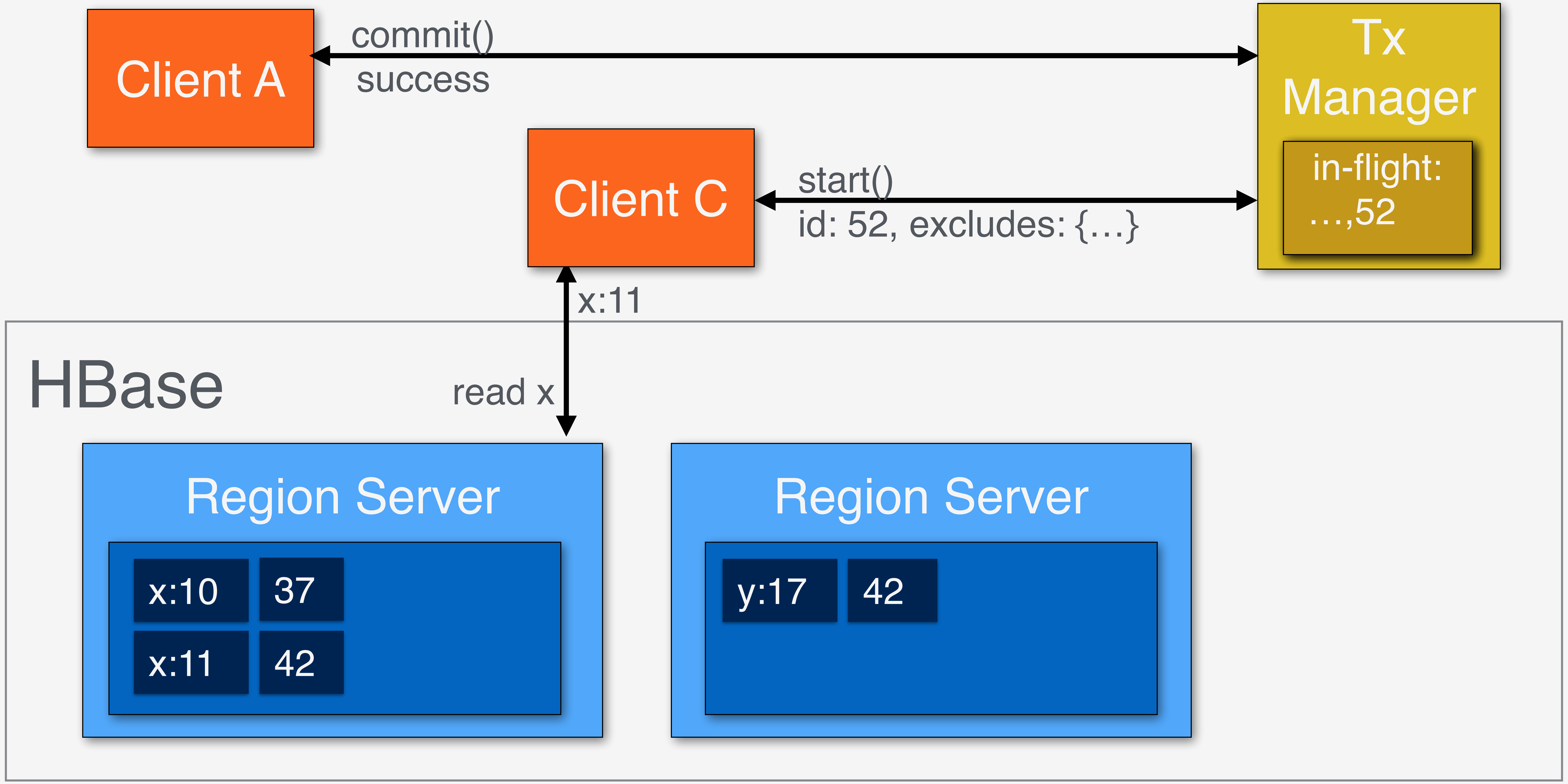
Apache Tephra



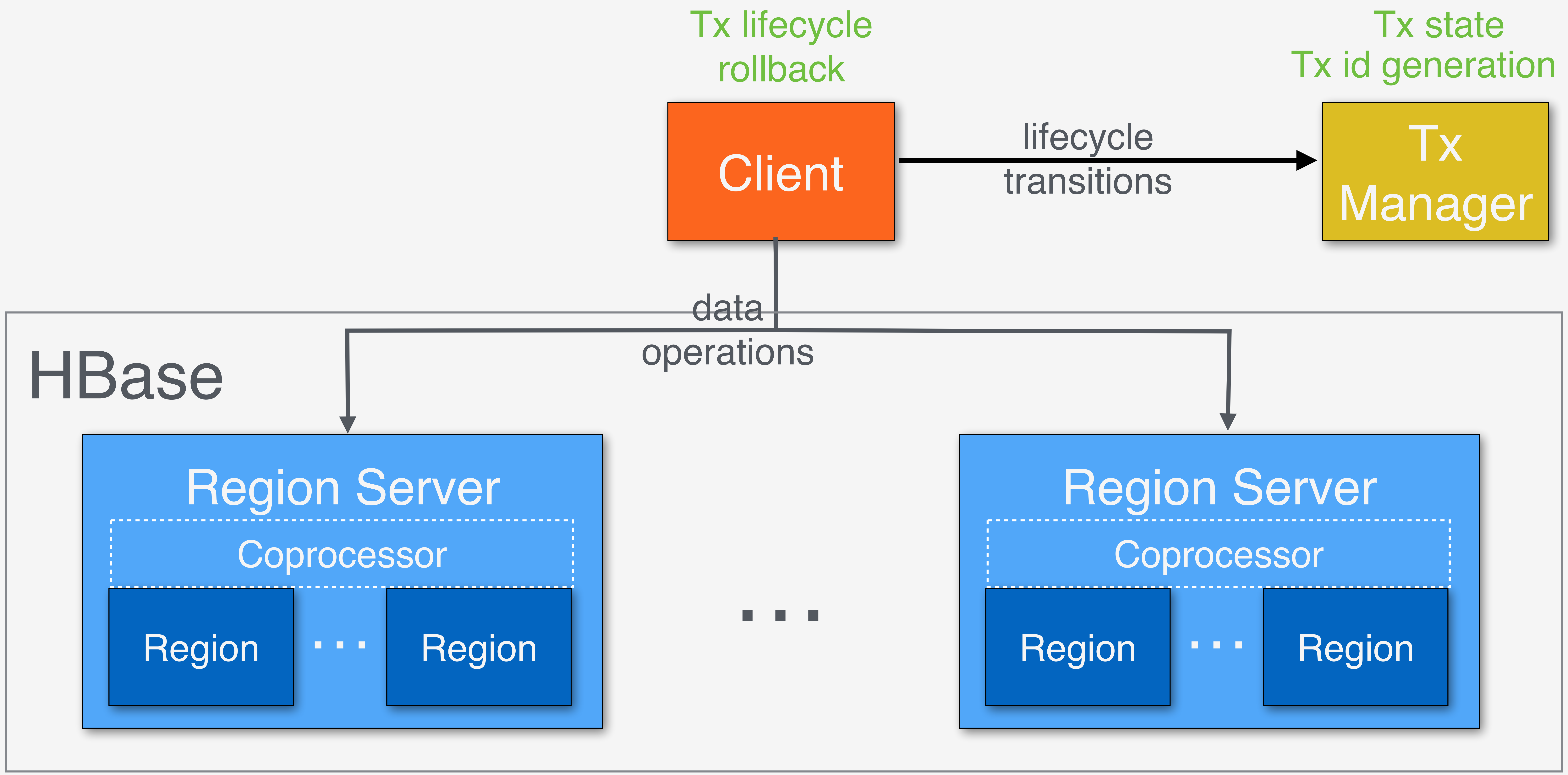
Apache Tephra



Apache Tephra



Apache Tephra



Apache Tephra

- HBase coprocessors
  - For efficient visibility filtering (on region-server side)
  - For eliminating invalid cells on flush and compaction

- Programming Abstraction
  - TransactionalHTable:
    - Implements HTable interface
    - Existing code is easy to port
  - TransactionContext:
    - Implements transaction lifecycle

Apache Tephra - Example

```
txTable = new TransactionAwareHTable(table);
txContext = new TransactionContext(txClient, txTable);
txContext.start();
try {
  // perform Hbase operations in txTable
  txTable.put(…);
  ...
} catch (Exception e) {
  // throws TransactionFailureException(e)
  txContext.abort(e);
}
// throws TransactionConflictException if so
txContext.finish();
```

Apache Tephra - Strengths

- Compatible with existing, non-tx data in HBase
- Programming model
    - Same API as HTable, keep existing client code
- Conflict detection granularity
    - Row, Column, Off
    - Special "long-running tx" for MapReduce and similar jobs
- HA and Fault Tolerance
    - Checkpoints and WAL for transaction state, Standby Tx Manager
- Replication compatible
    - Checkpoint to HBase, use HBase replication
- Secure, Multi-tenant

Apache Tephra - Not-So Strengths

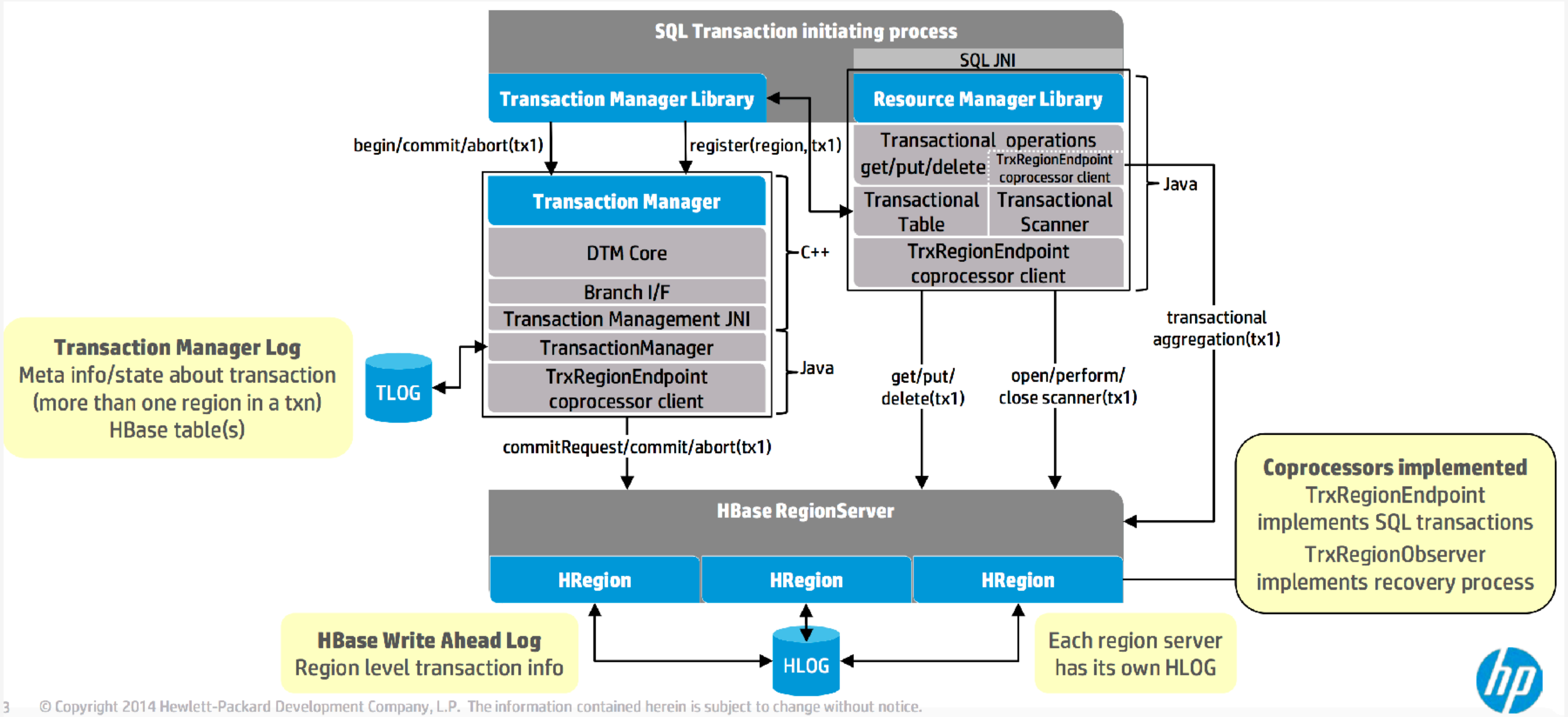
- Exclude list can grow large over time
    - RPC, post-filtering overhead
    - Solution: Invalid tx pruning on compaction - complex!
- Single Transaction Manager
    - performs all lifecycle state transitions, including conflict detection
    - conflict detection requires lock on the transaction state
    - becomes a bottleneck
    - Solution: distributed Transaction Manager with consensus protocol

Apache Trafodion

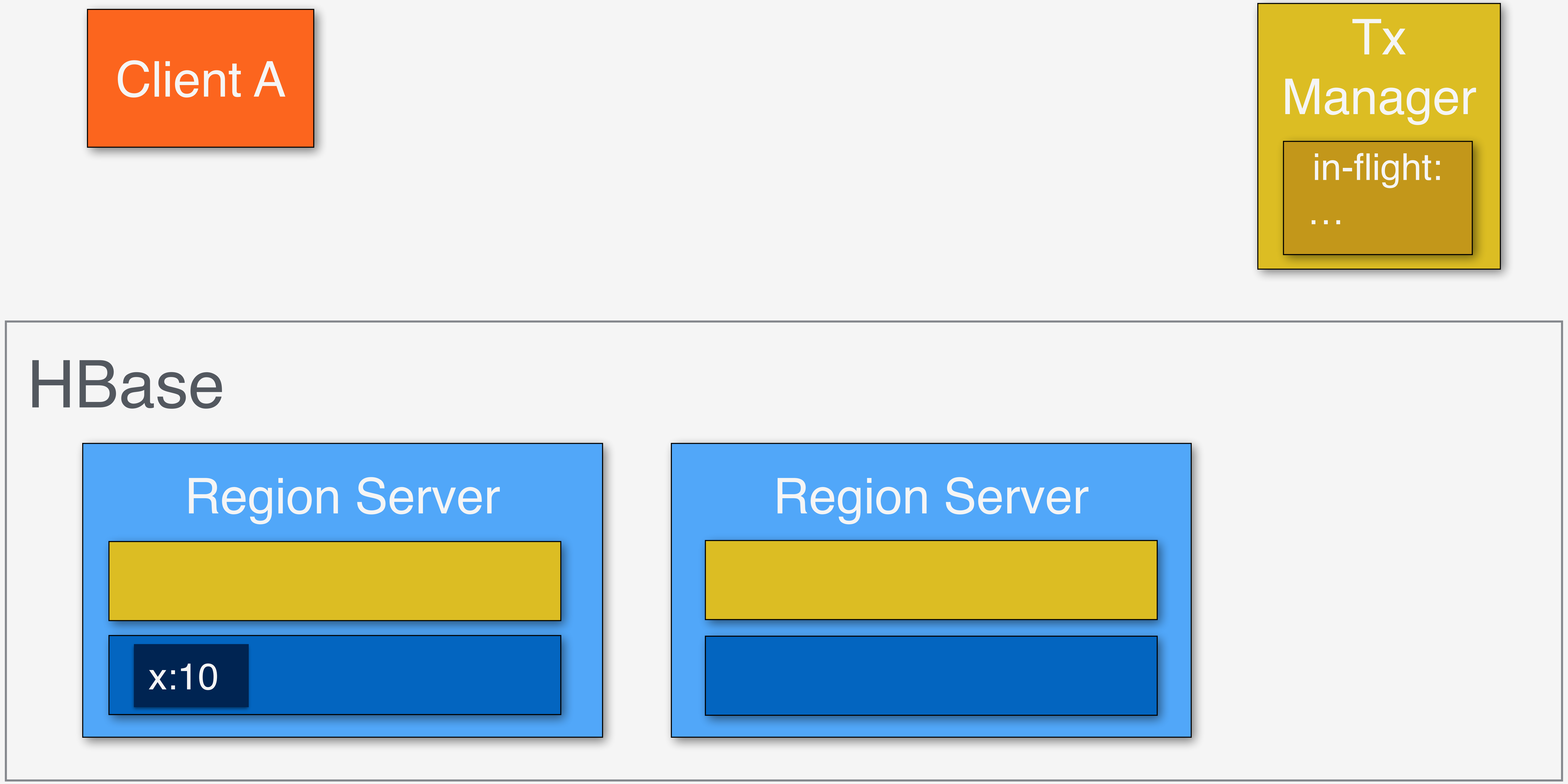
- A complete distributed database (RDBMS)
  - transaction system is not available by itself
  - APIs: jdbc, SQL
- Inspired by original HBase TRX (transactional region server
  - migrated transaction logic into coprocessors
  - coprocessors cache in-flight data in-memory
  - transaction state (change sets) in coprocessors
  - conflict detection with 2-phase commit
- Transaction Manager
  - orchestrates transaction lifecycle across involved region servers
  - multiple instances, but one per client

(incubating)

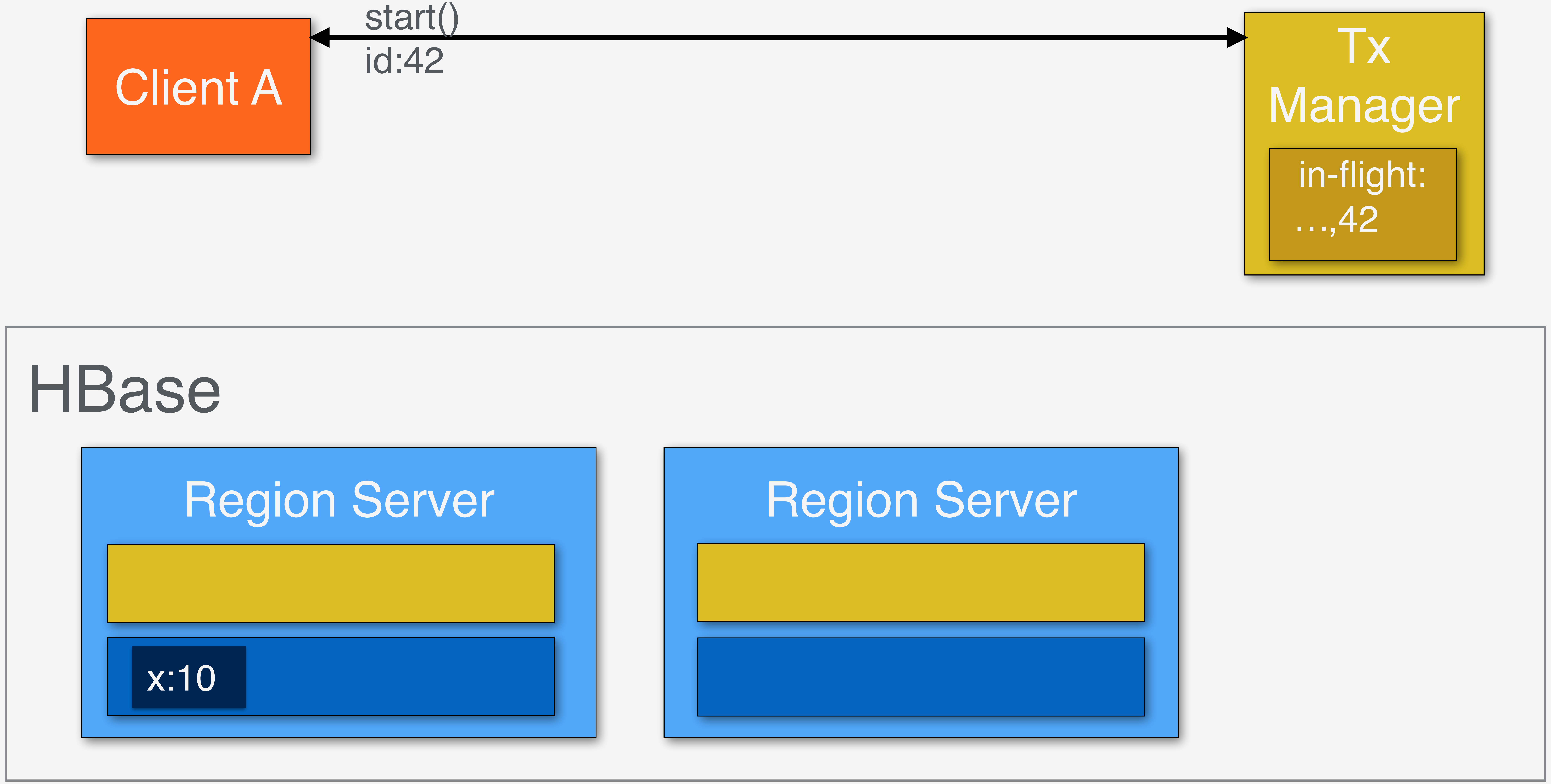
Apache Trafodion


SQL Transaction initiating process
SQL JNI
Transaction Manager Library
Resource Manager Library
begin/commit/abort(tx1)
register(region, tx1)
Transactional operations
get/put/delete
TrxRegionEndpoint coprocessor client
Transactional Table
Transactional Scanner
TrxRegionEndpoint coprocessor client
Java
Transaction Manager
DTM Core
Branch I/F
Transaction Management JNI
TransactionManager
TrxRegionEndpoint coprocessor client
C++
Java
Transaction Manager Log
Meta info/state about transaction (more than one region in a txn)
HBase table(s)
TLOG
transactional aggregation(tx1)
get/put/ delete(tx1)
open/perform/ close scanner(tx1)
commitRequest/commit/abort(tx1)
HBase RegionServer
HRegion
HRegion
HRegion
Coprocessors implemented
TrxRegionEndpoint implements SQL transactions
TrxRegionObserver implements recovery process
HBase Write Ahead Log
Region level transaction info
HLOG
Each region server has its own HLOG

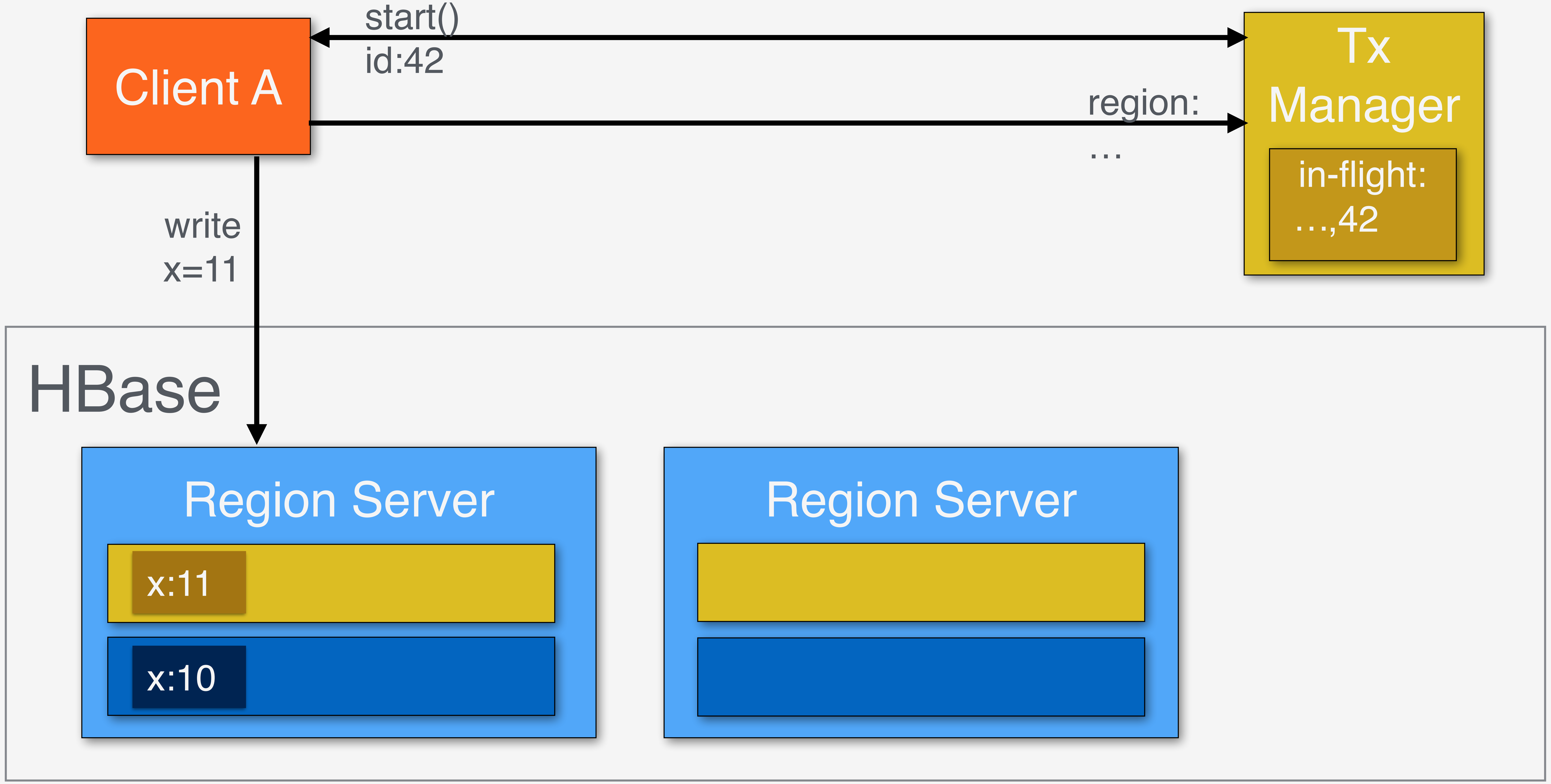
Apache Trafodion



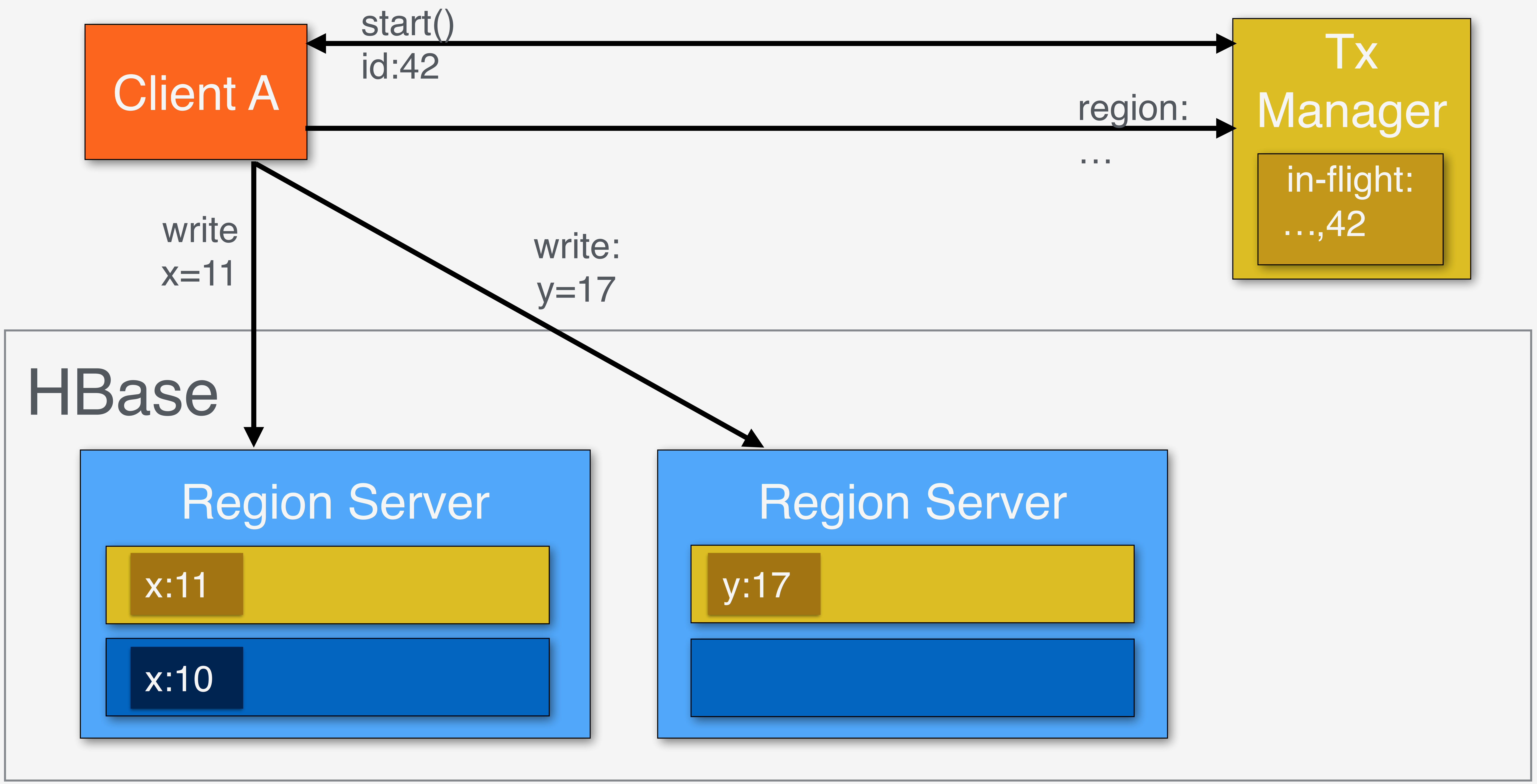
Apache Trafodion



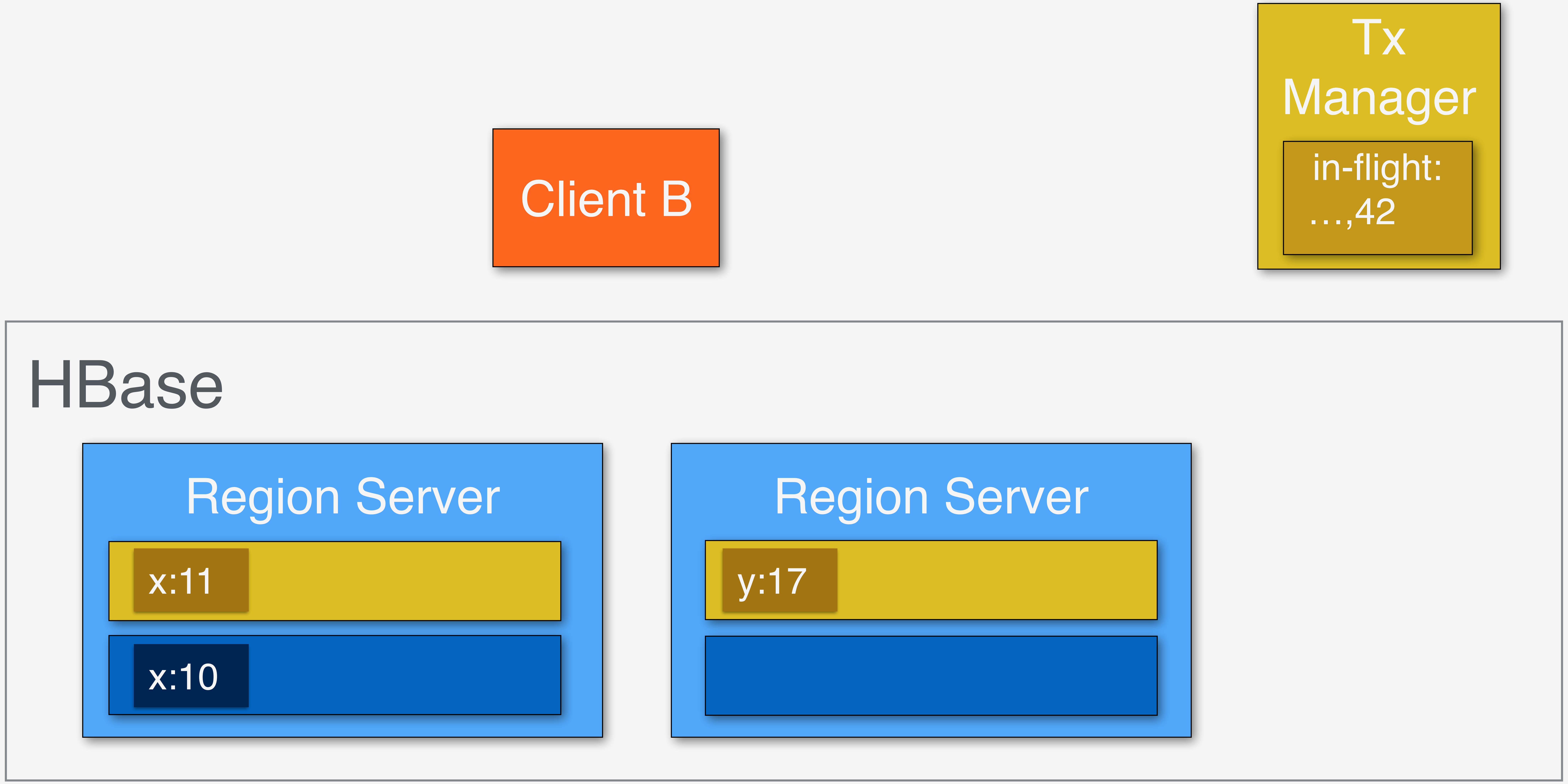
Apache Trafodion



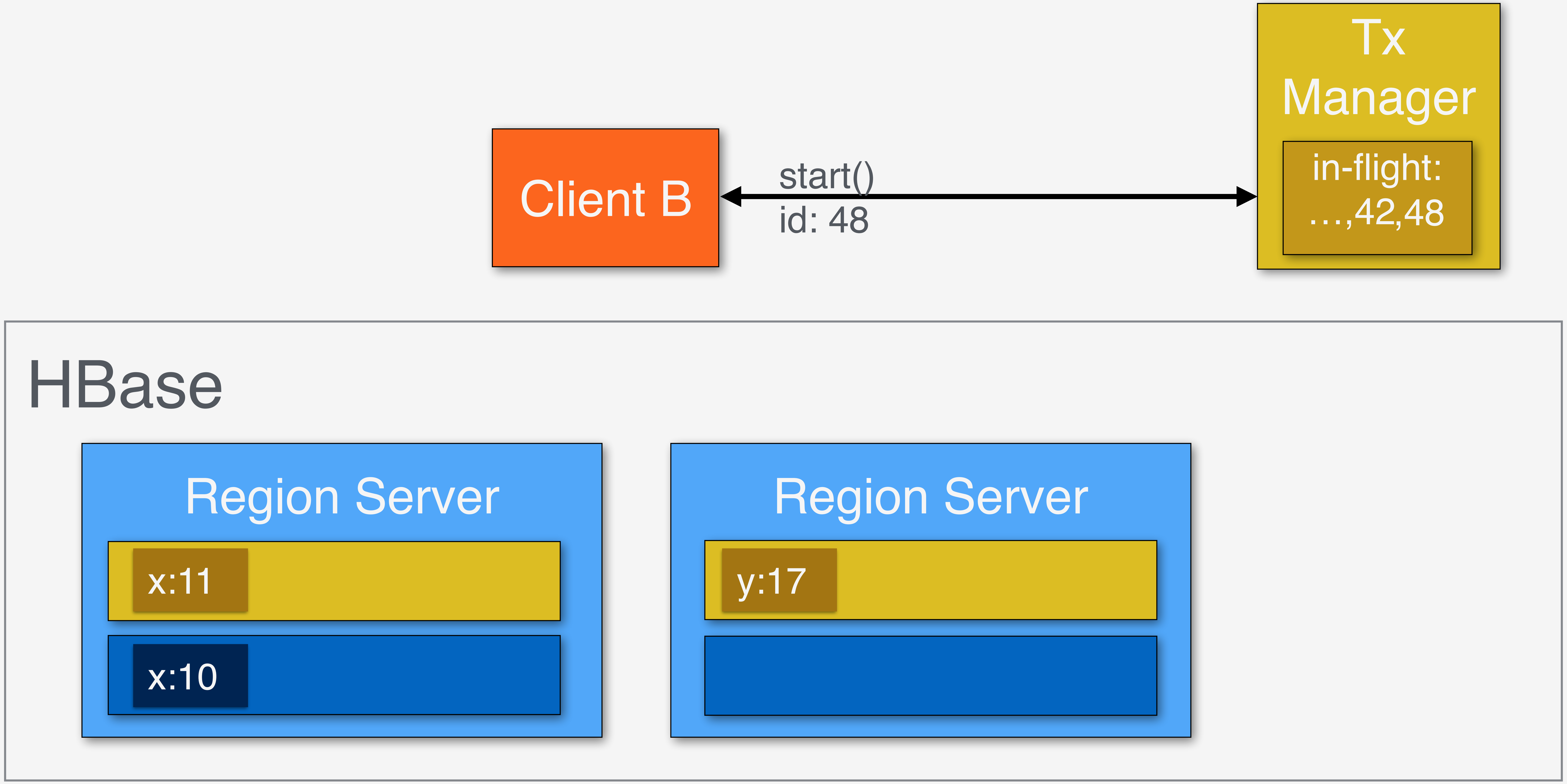
Apache Trafodion



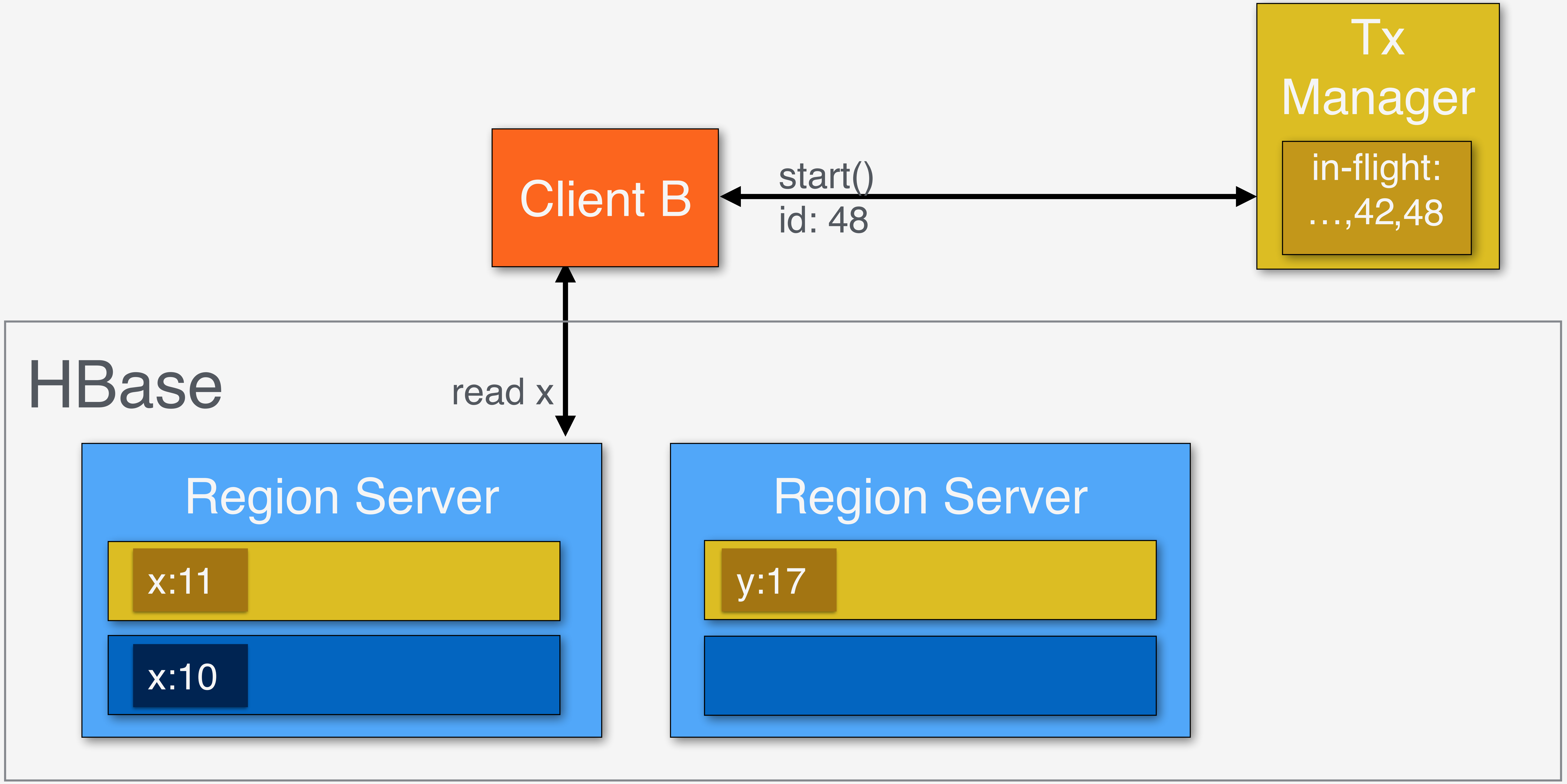
Apache Trafodion



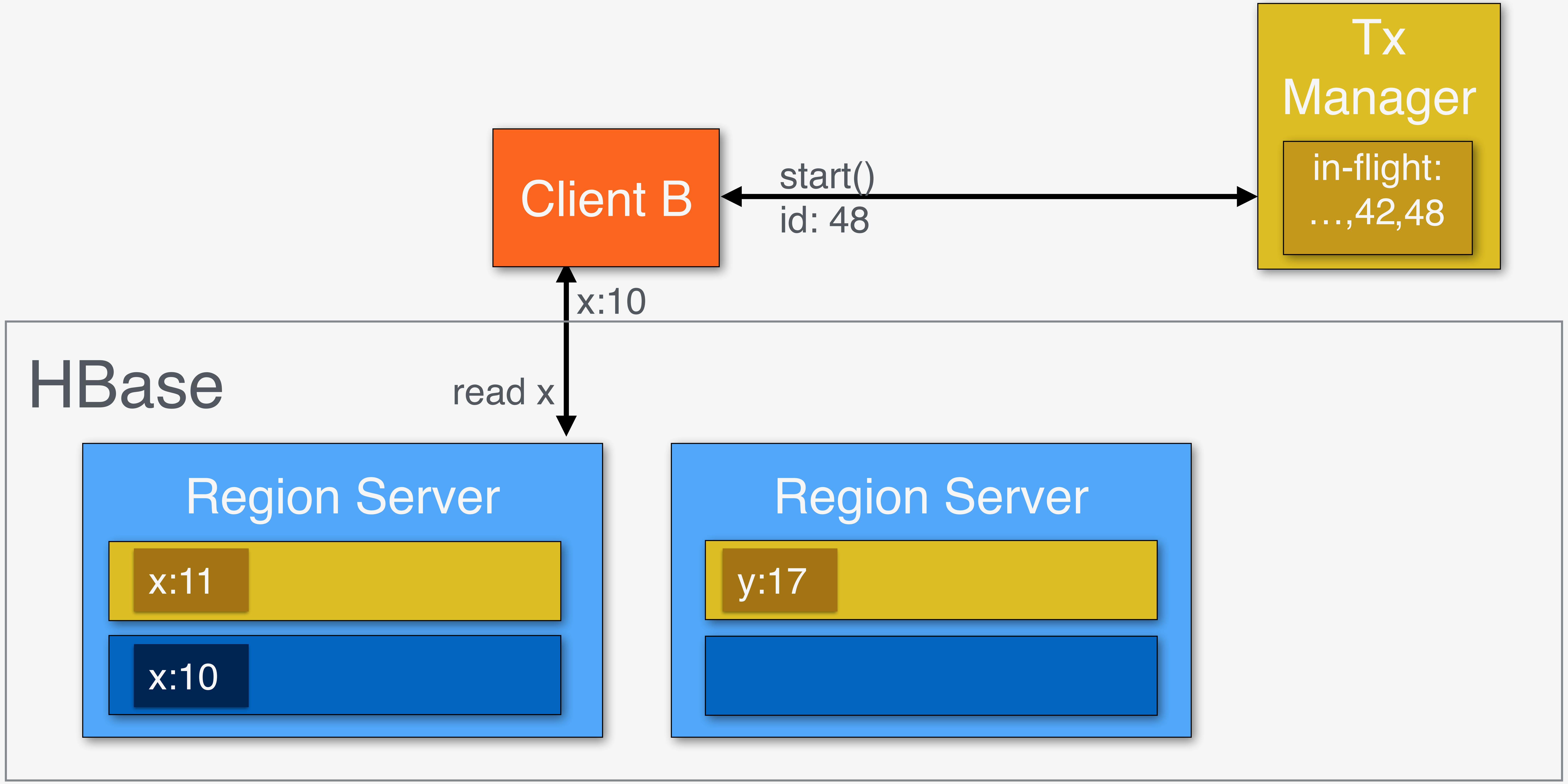
Apache Trafodion



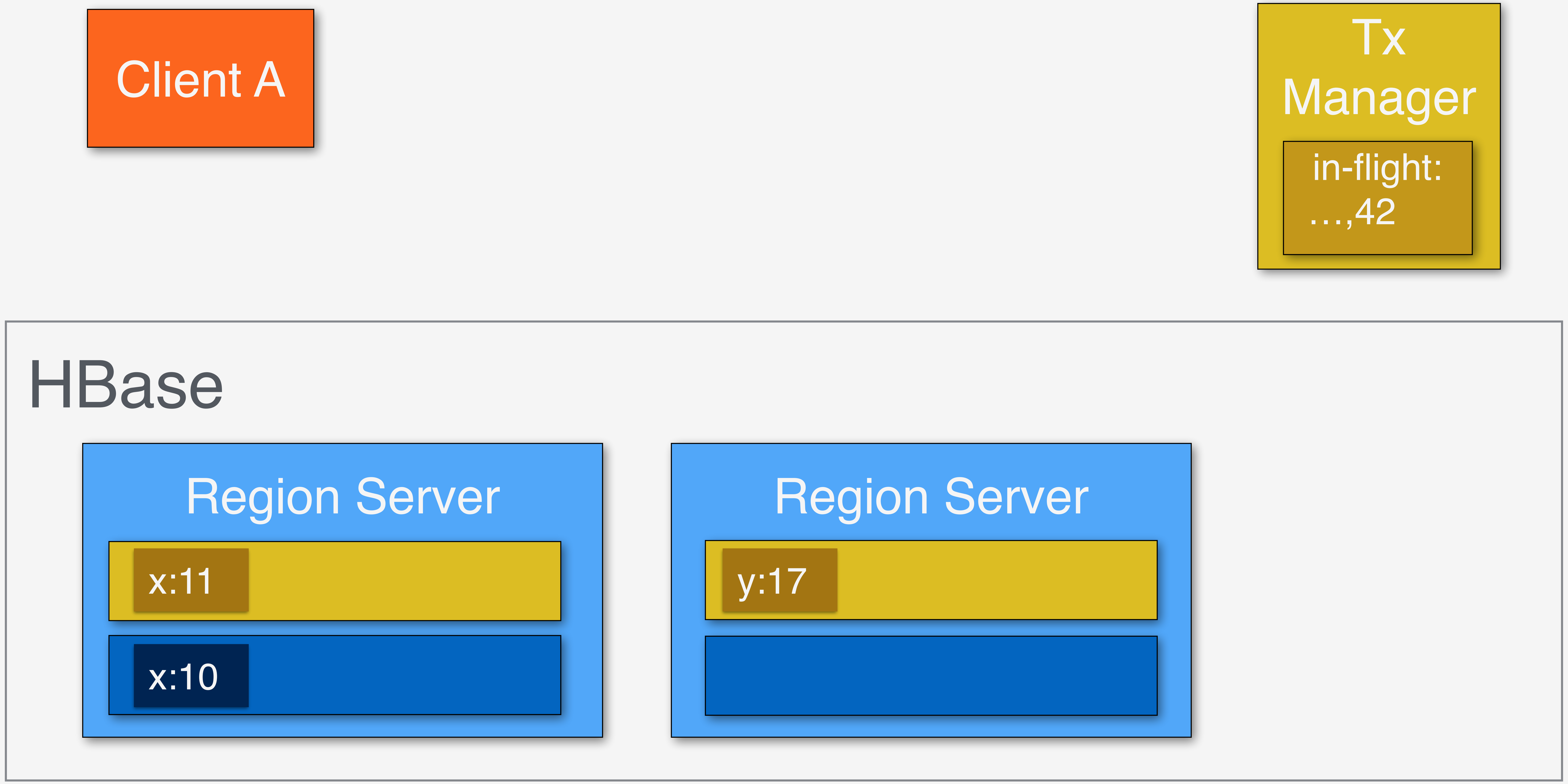
Apache Trafodion



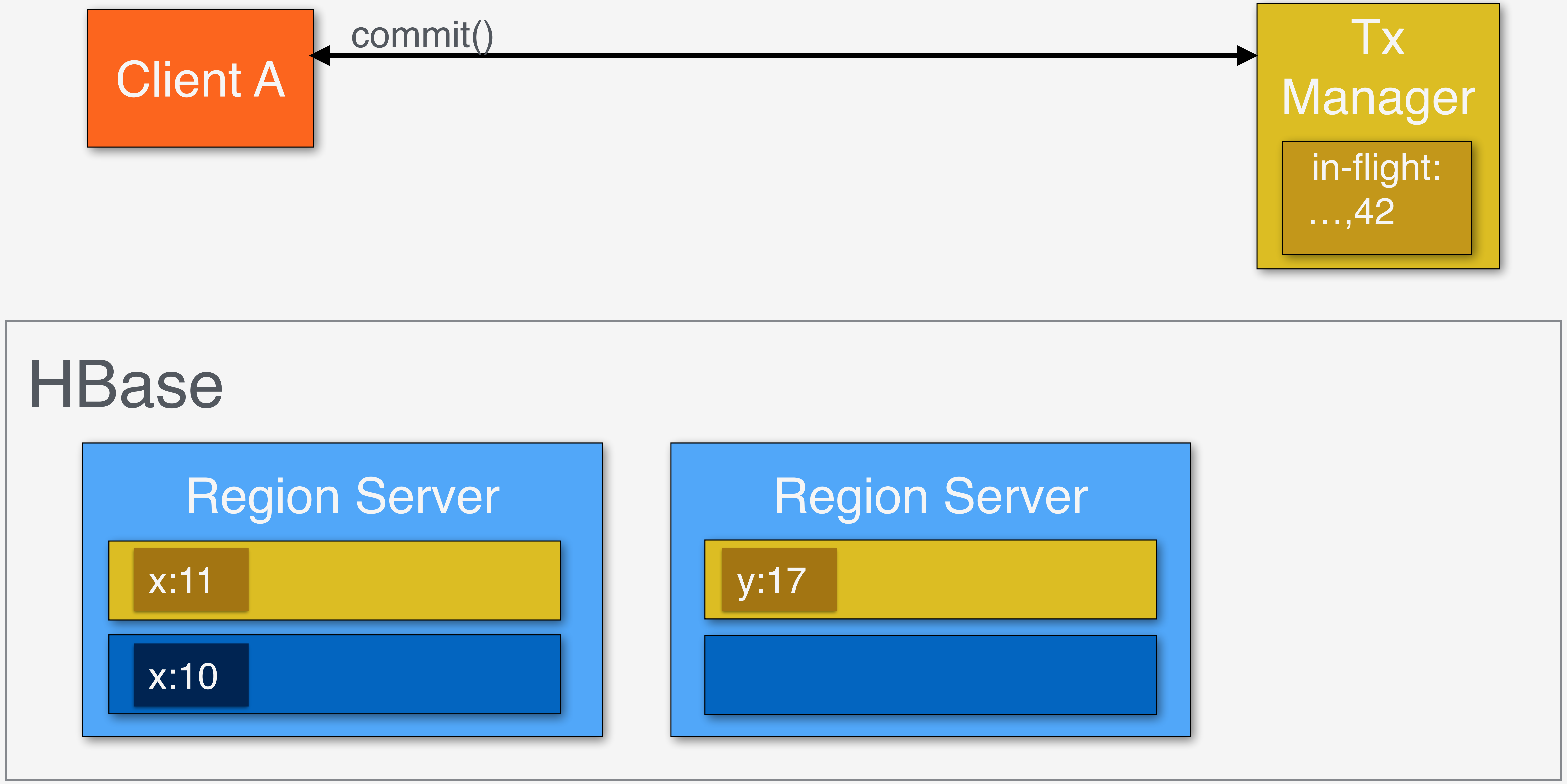
Apache Trafodion



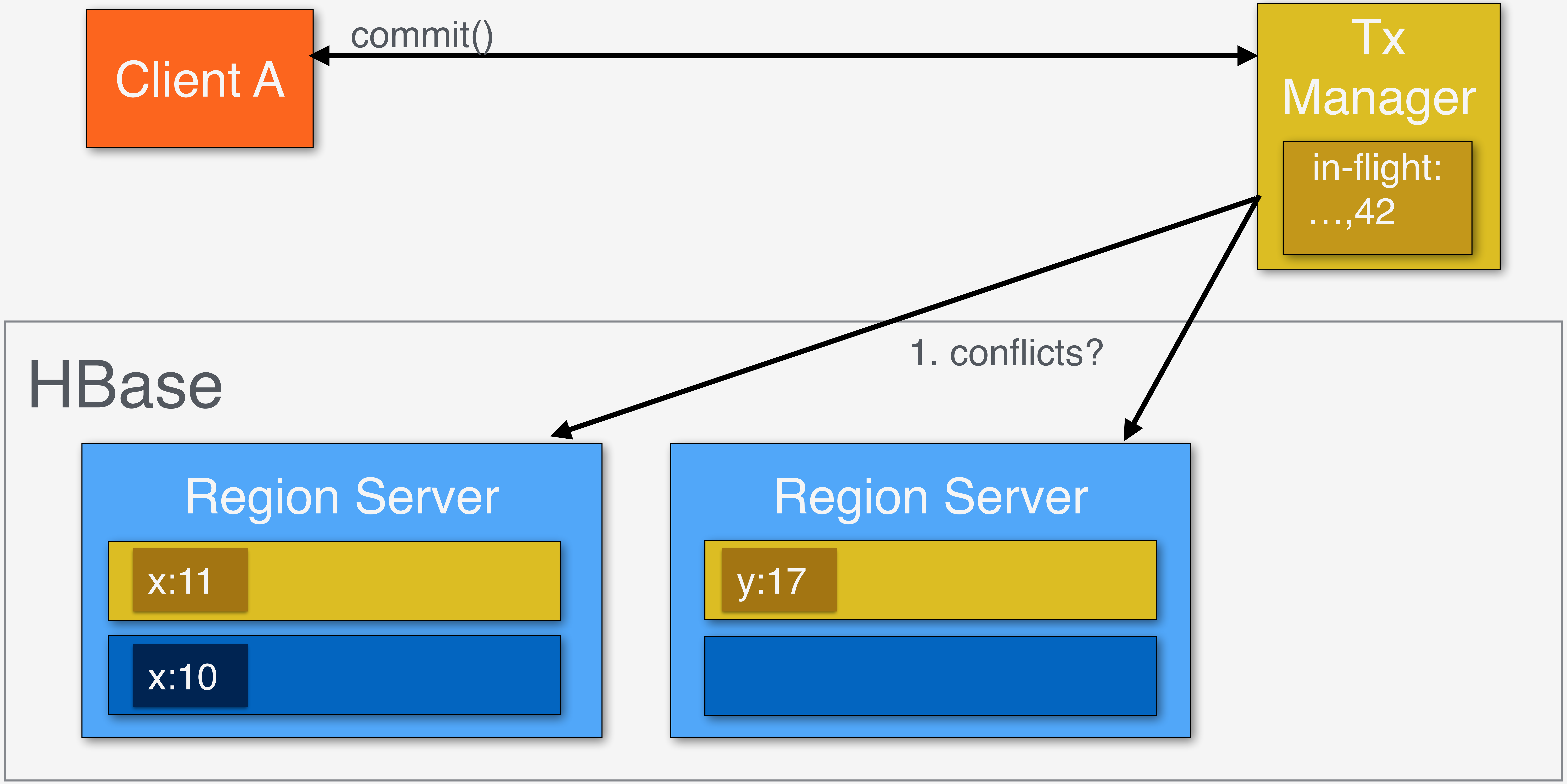
Apache Trafodion



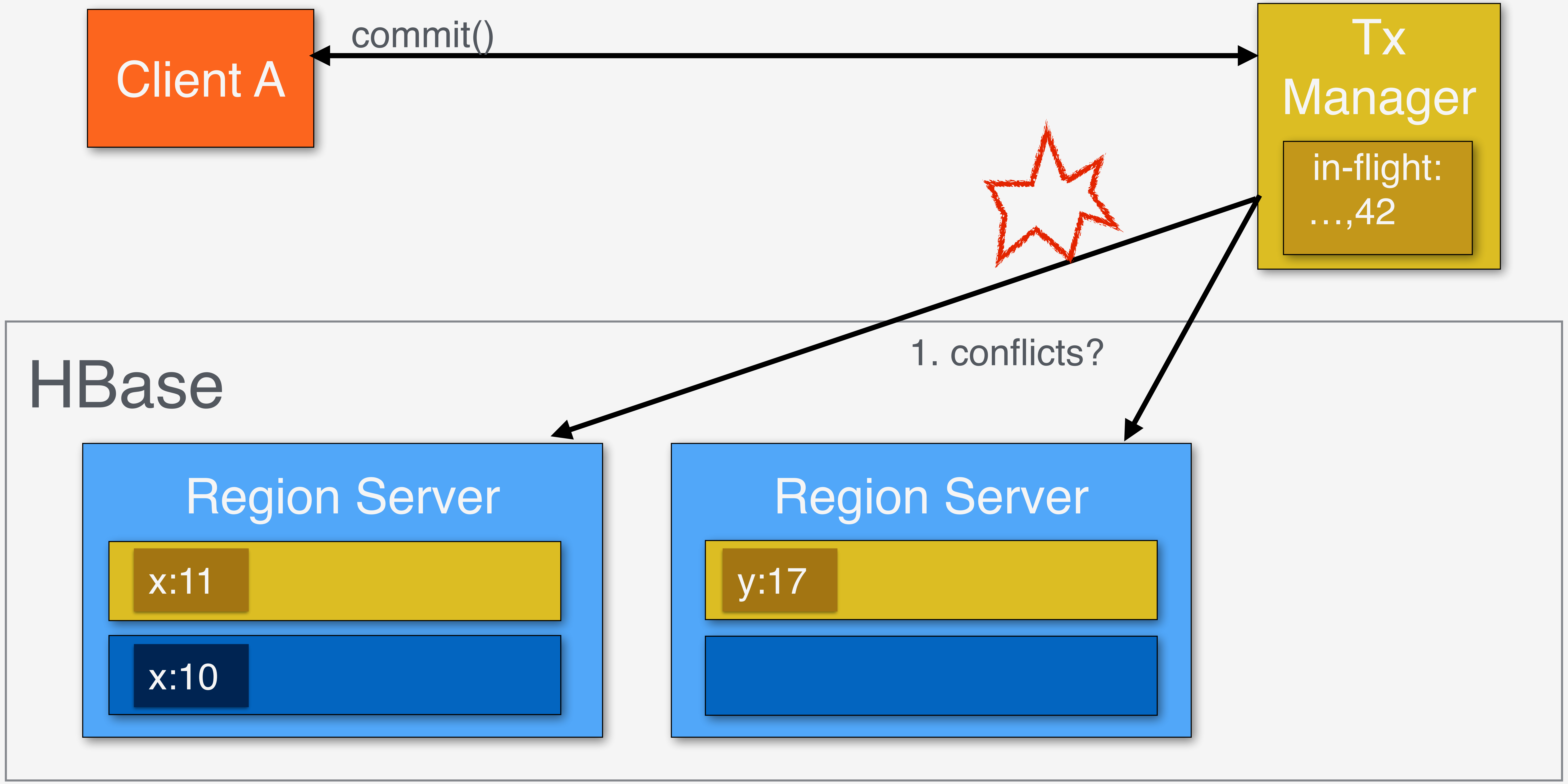
Apache Trafodion



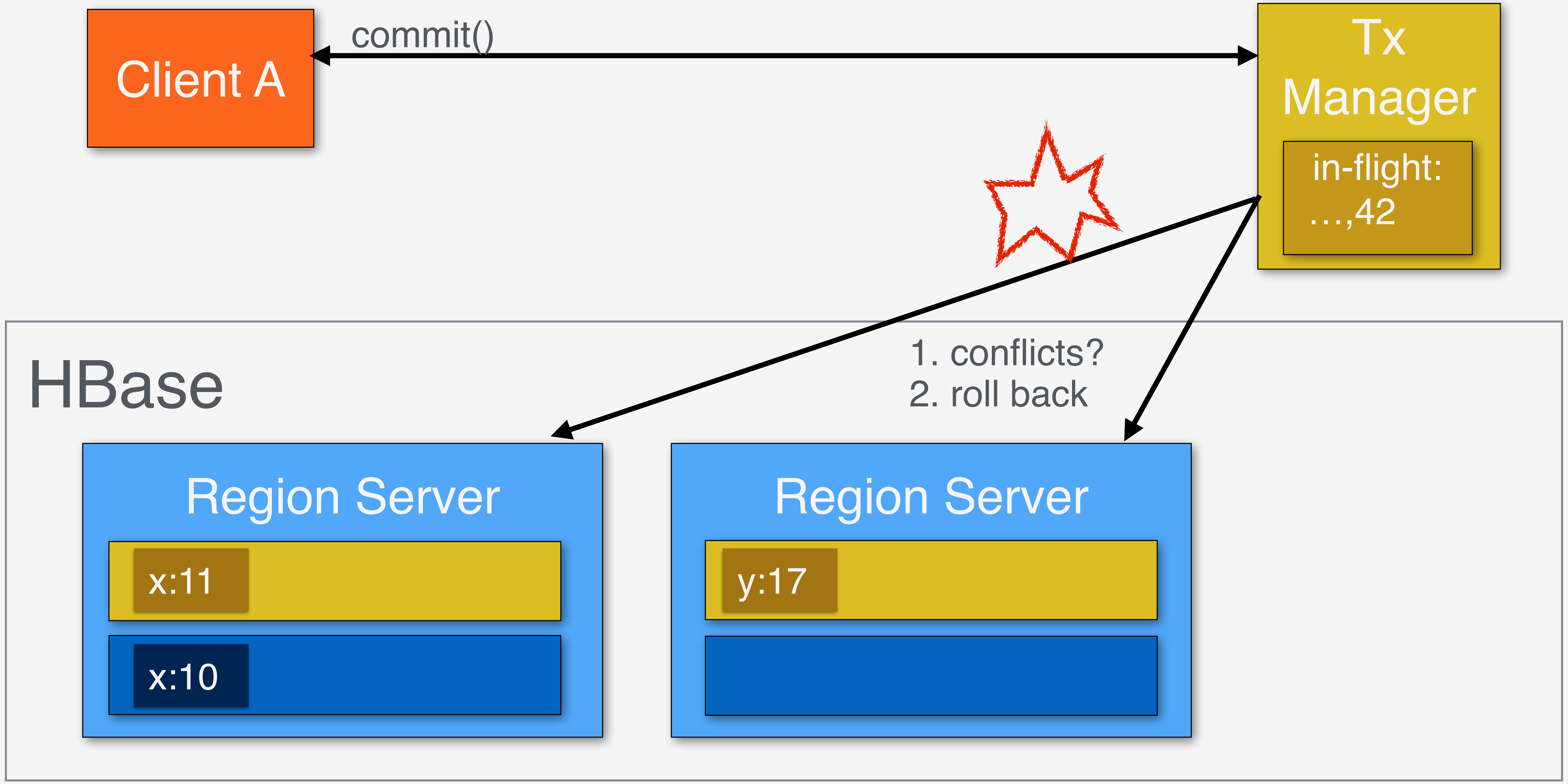
Apache Trafodion



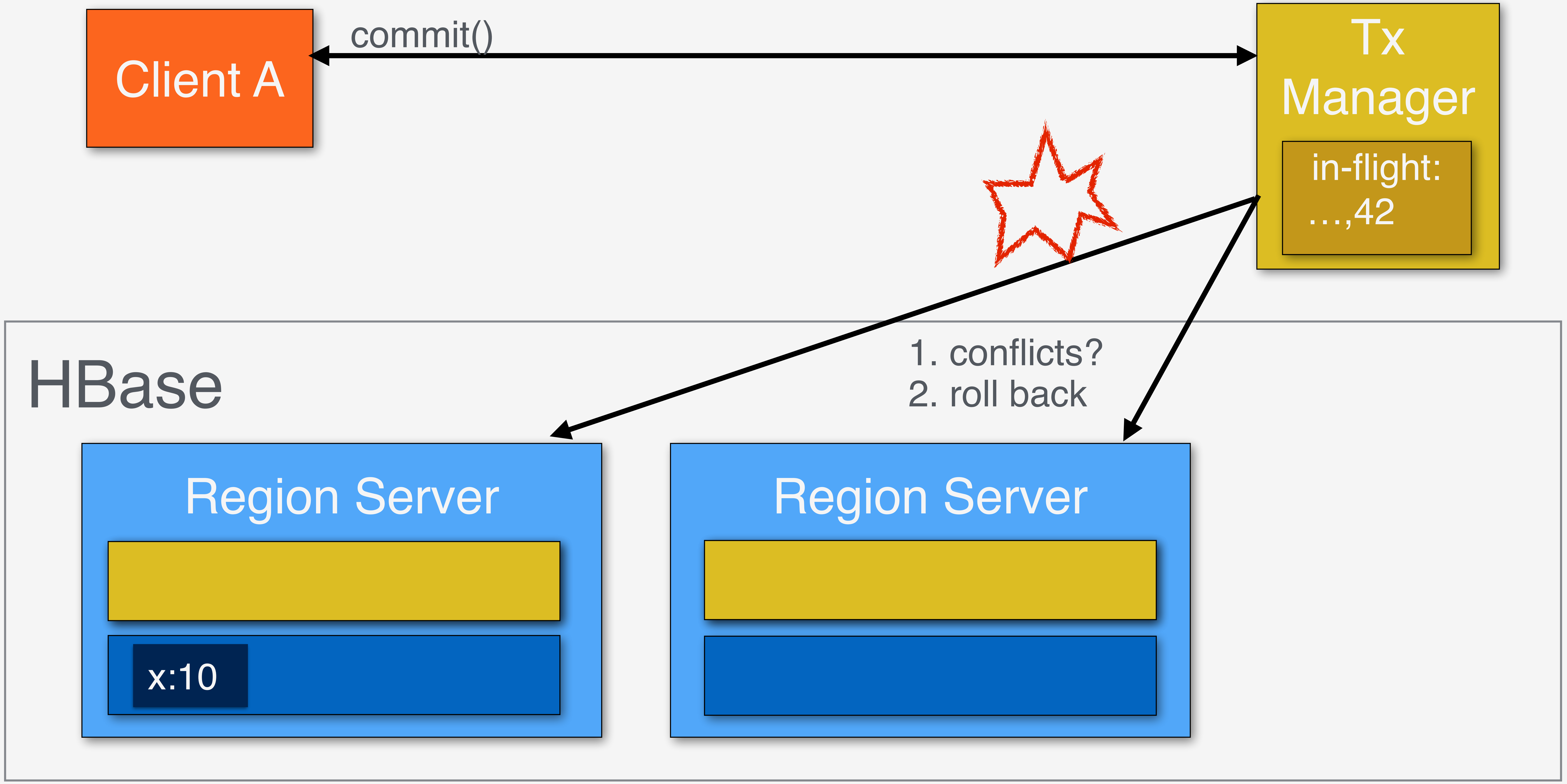
Apache Trafodion



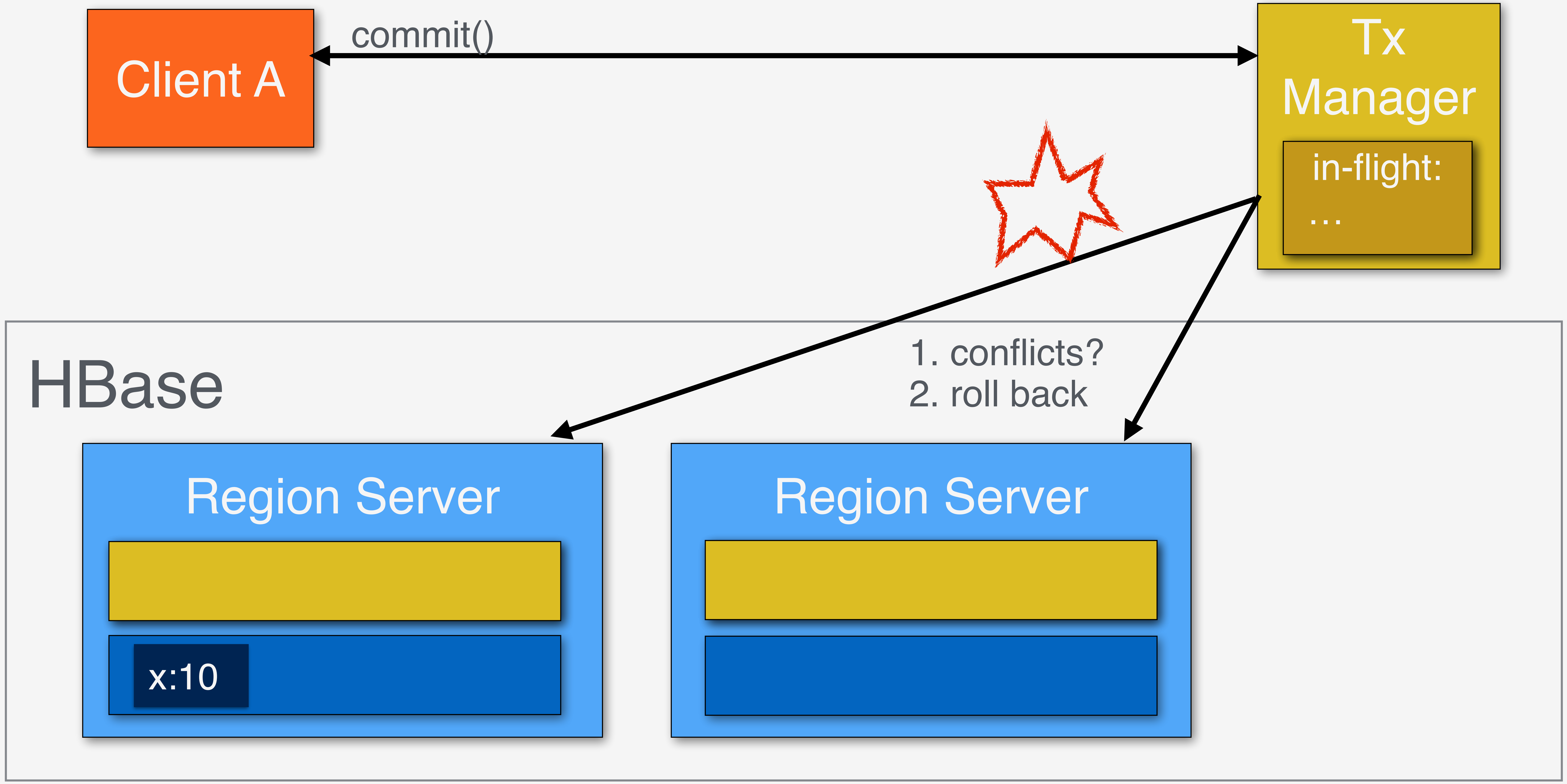
Apache Trafodion



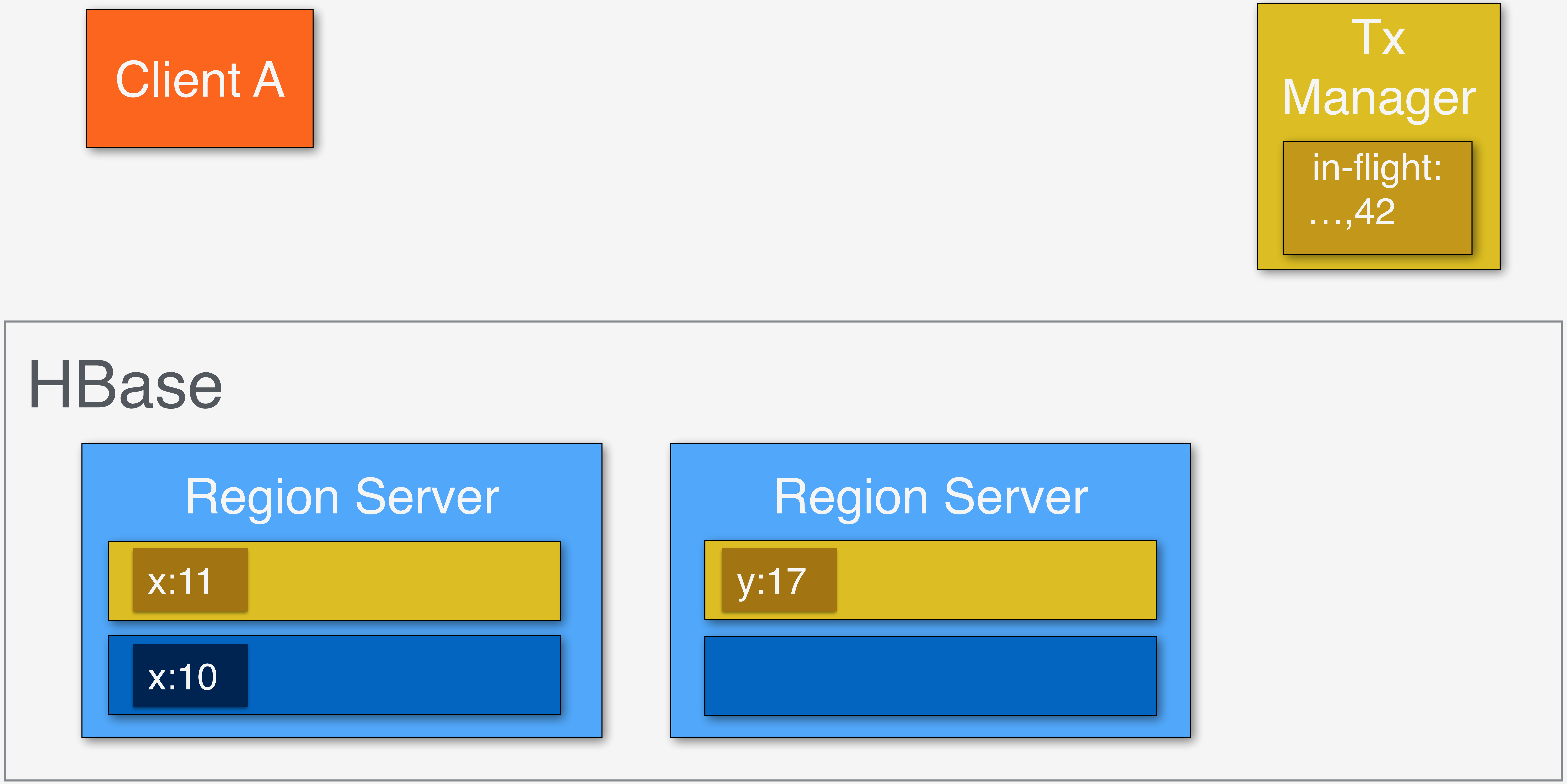
Apache Trafodion



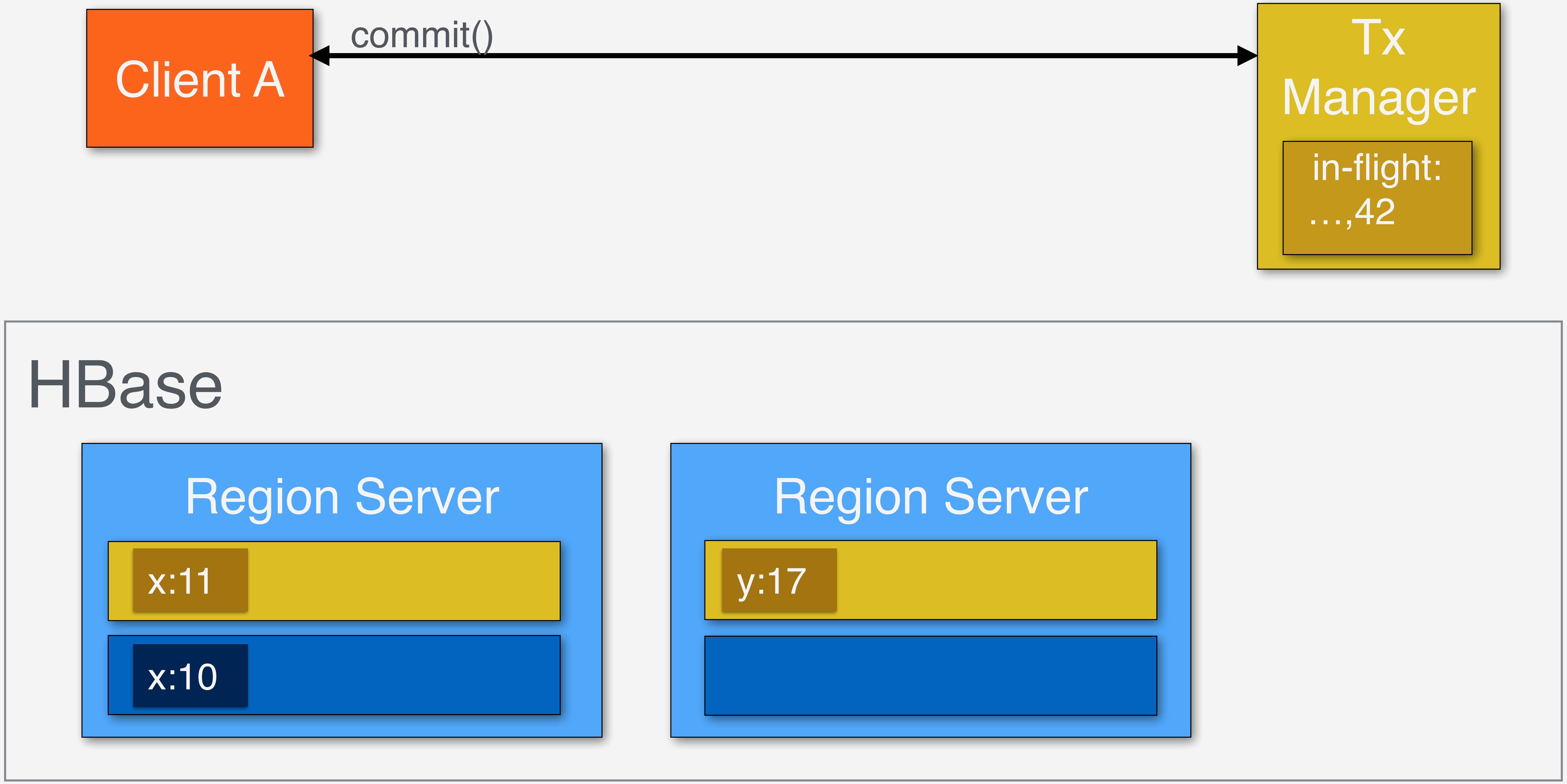
Apache Trafodion



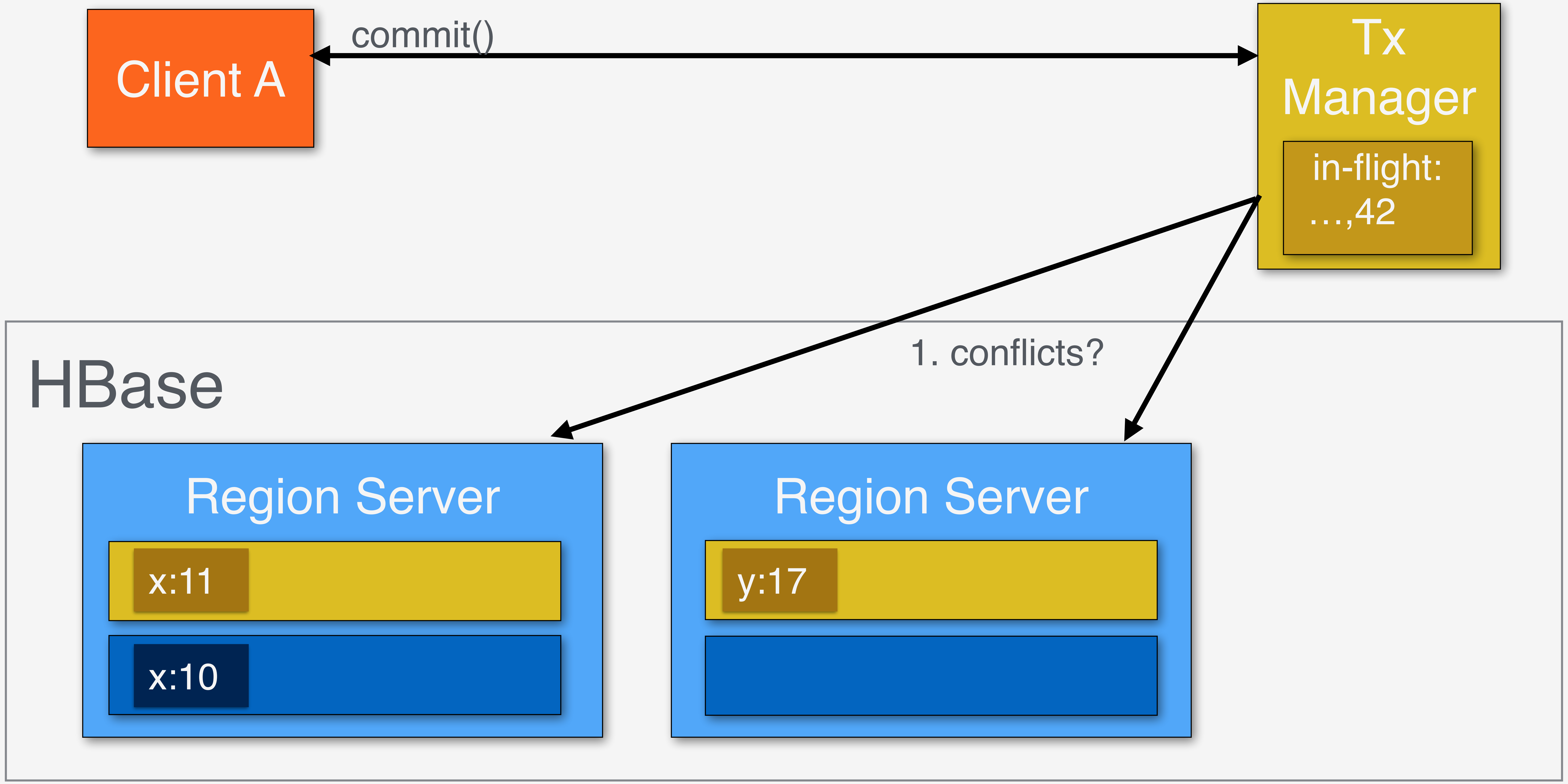
Apache Trafodion



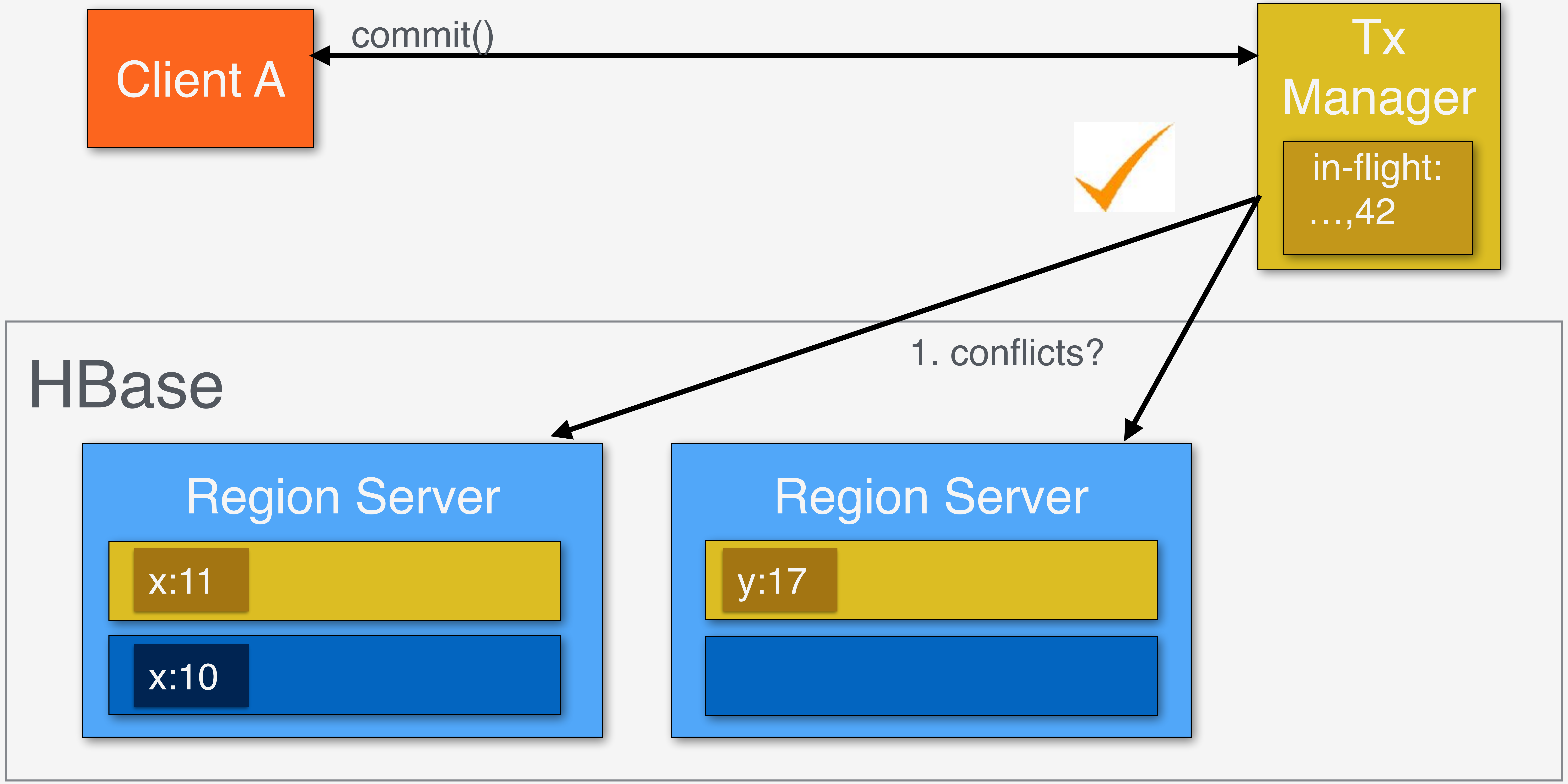
Apache Trafodion



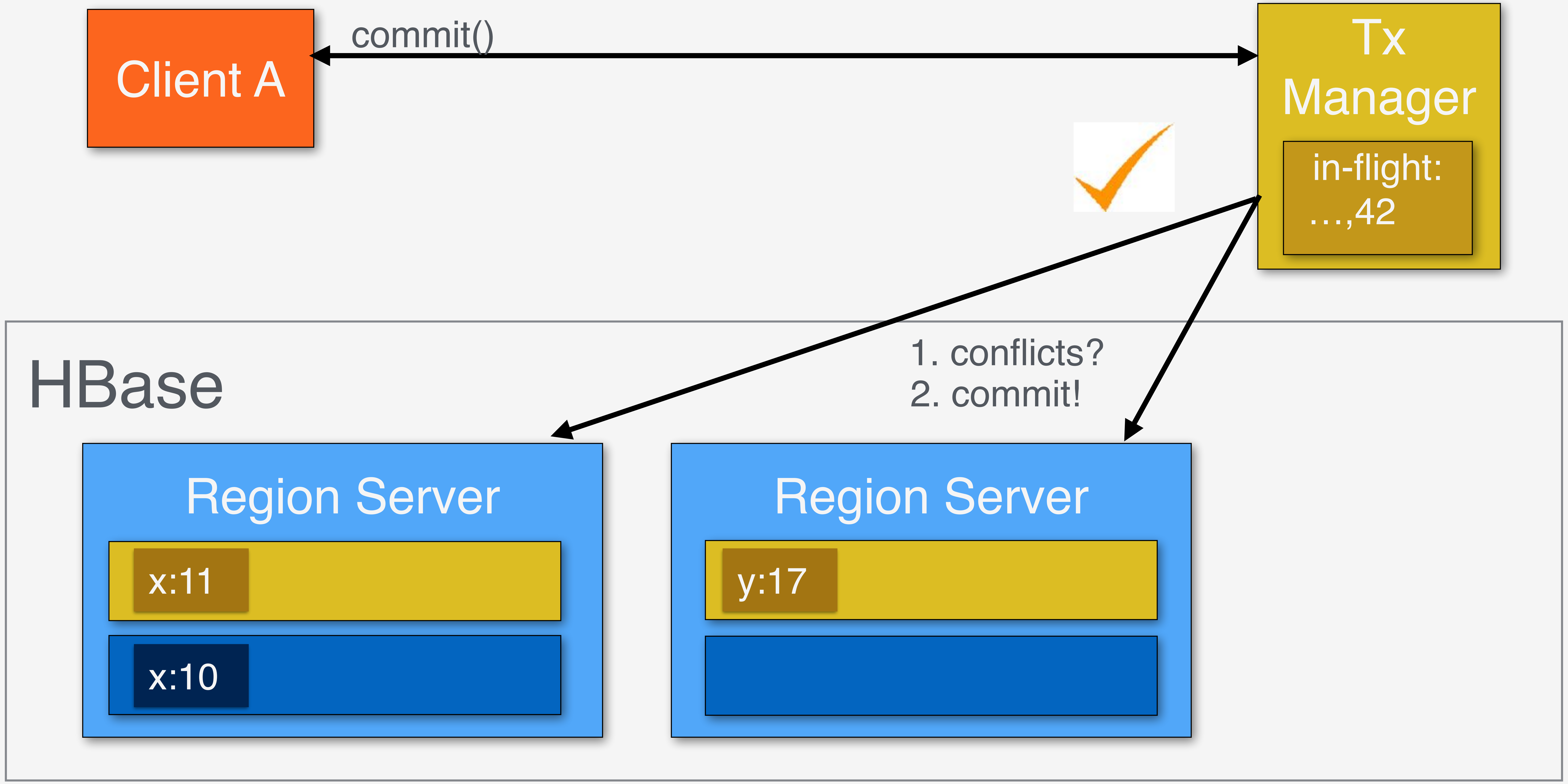
Apache Trafodion



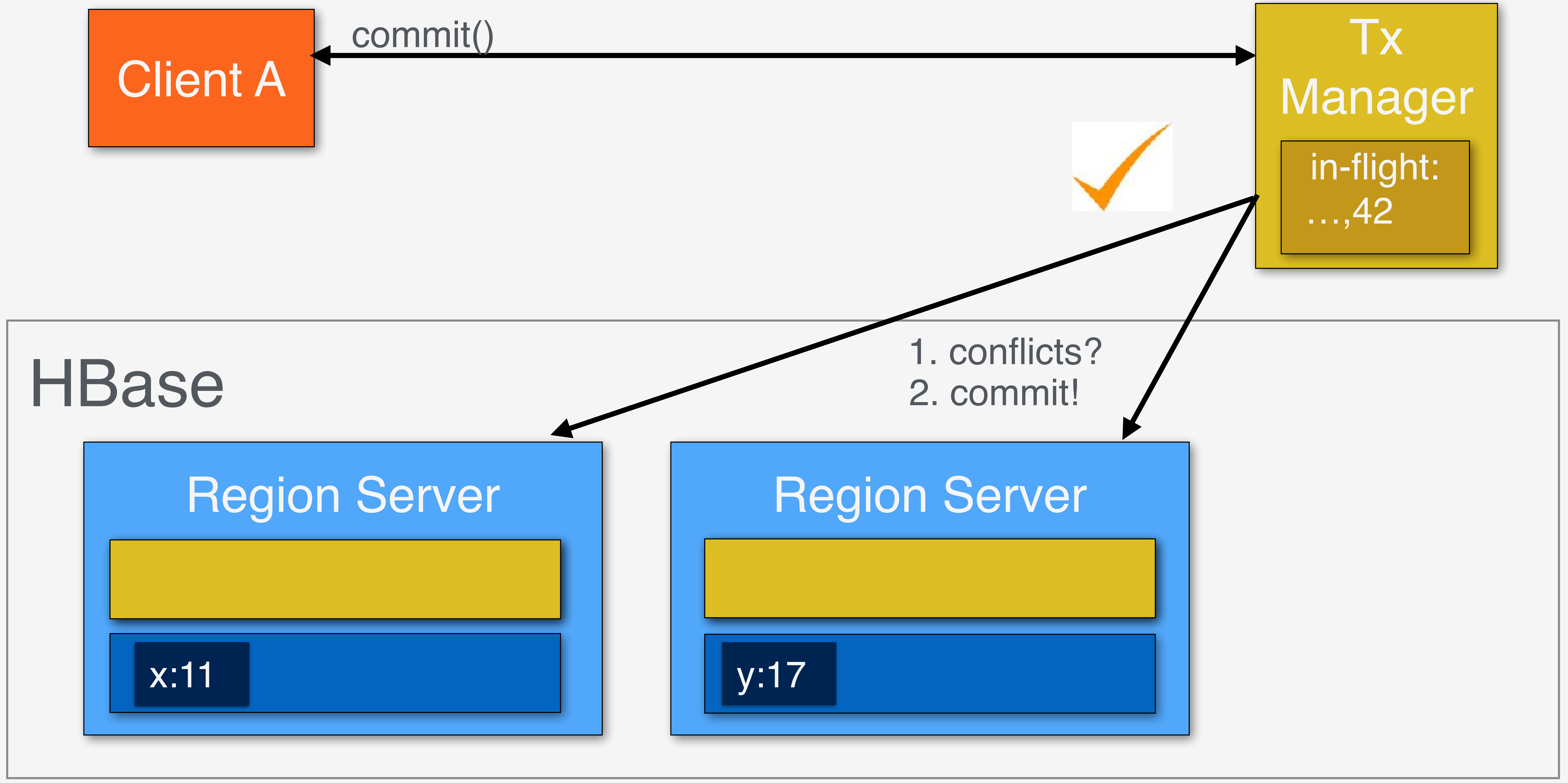
Apache Trafodion



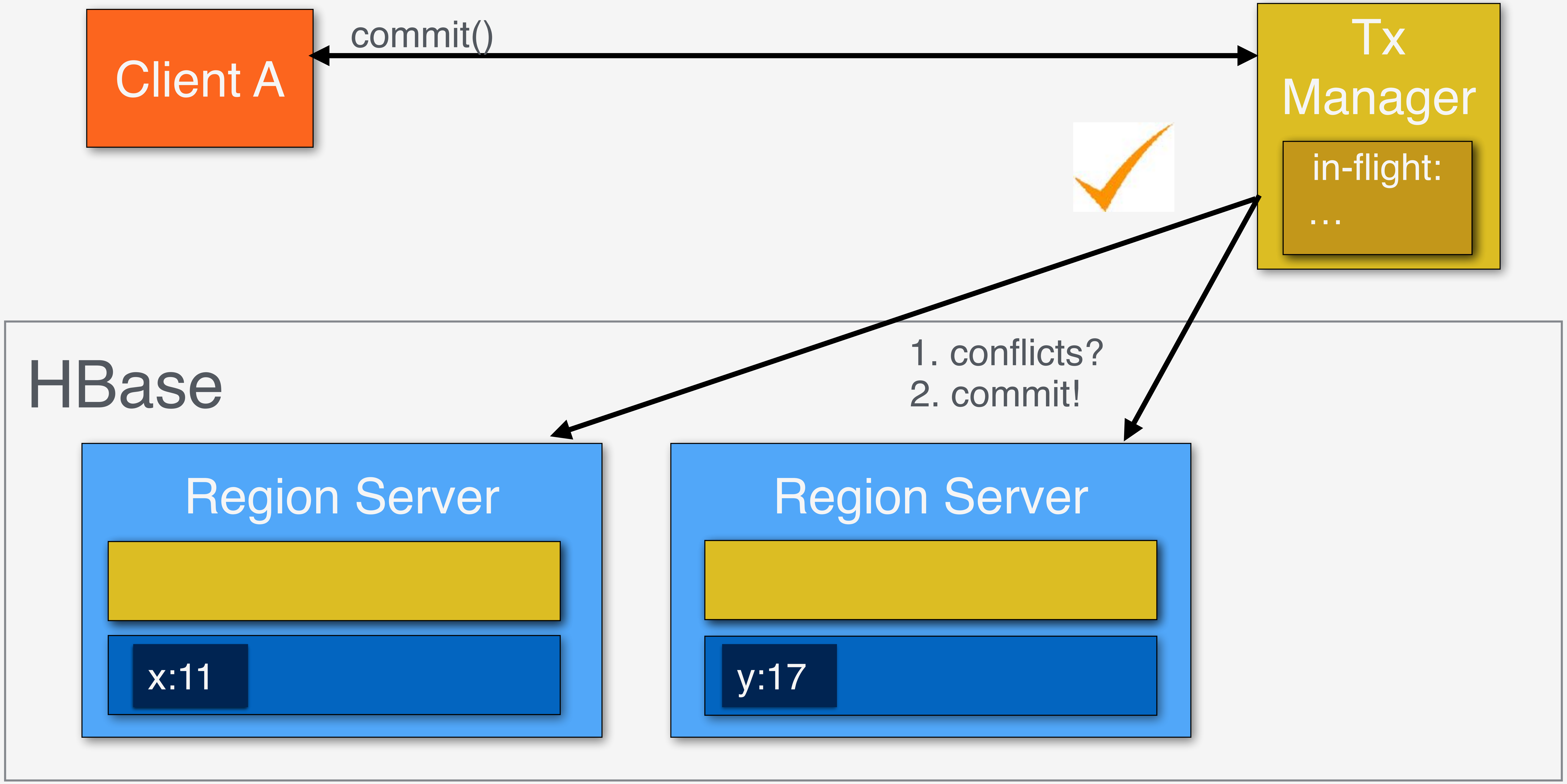
Apache Trafodion



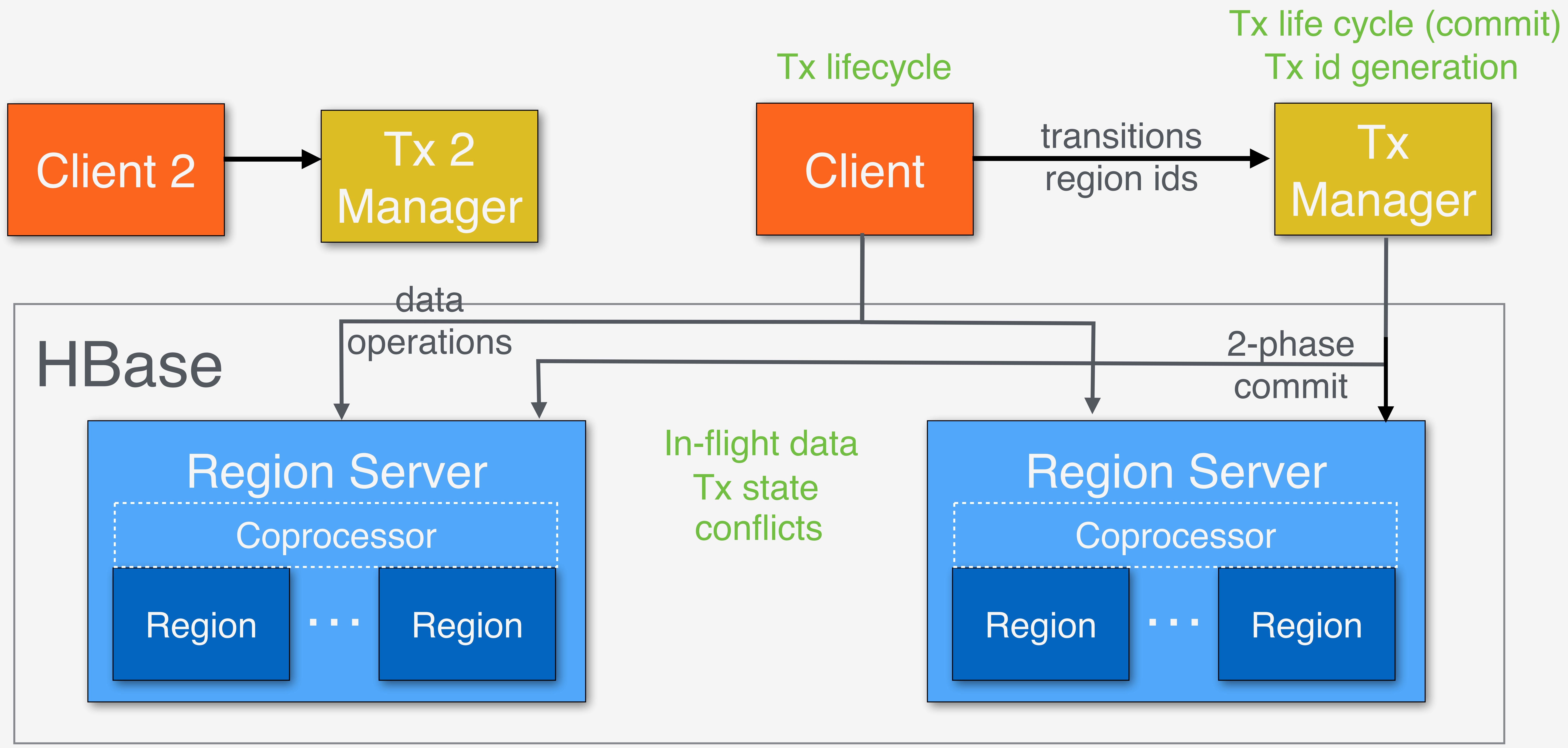
Apache Trafodion



Apache Trafodion



Apache Trafodion



Apache Trafodion

- Scales well:
    - Conflict detection is distributed: no single bottleneck
    - Commit coordination by multiple transaction managers
    - Optimization: bypass 2-hase commit if single region
- Coprocessors cache in-flight data in Memory
    - Flushed to HBase only on commit
    - Committed read (not snapshot, not repeatable read)
    - Option: cause conflicts for reads, too
- HA and Fault Tolerance
    - WAL for all state
    - All services are redundant and take over for each other
- Replication: Only in paid (non-Apache) add-on

Apache Trafodion - Strengths

- Very good scalability
    - Scales almost linearly
    - Especially for very small transactions
- Familiar SQL/jdbc interface for RDB programmers
- Redundant and fault-tolerant
- Secure and multi-tenant:
    - Trafodion/SQL layer provides authn+authz

Apache Trafodion - Not-So Strengths

- Monolithic, not available as standalone transaction system
- Heavy load on coprocessors
    - memory and compute
- Large transactions (e.g., MapReduce) will cause Out-of-memory
    - no special support for long-running transactions

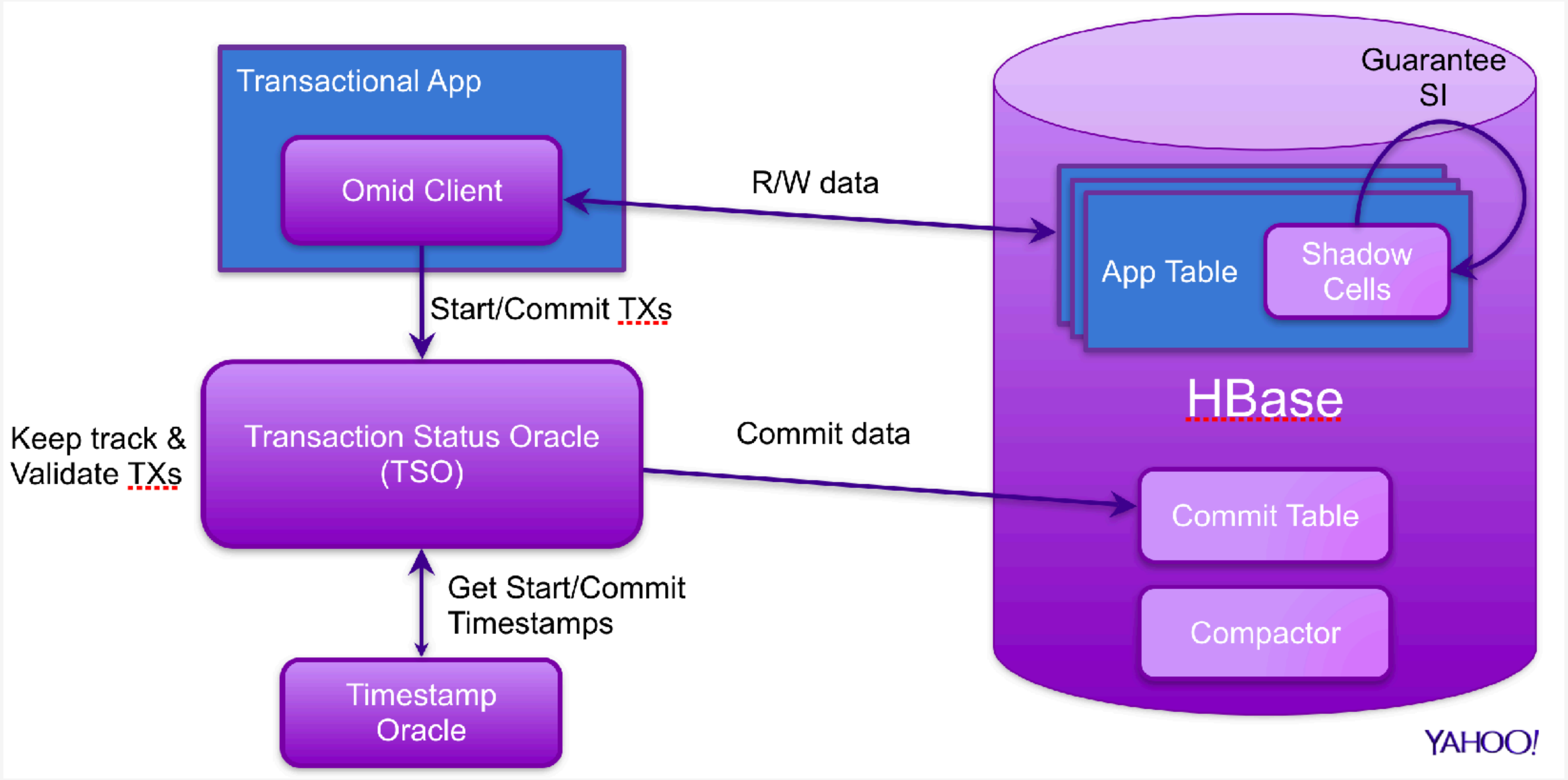
Apache Omid

- Evolution of Omid based on the Google Percolator paper:

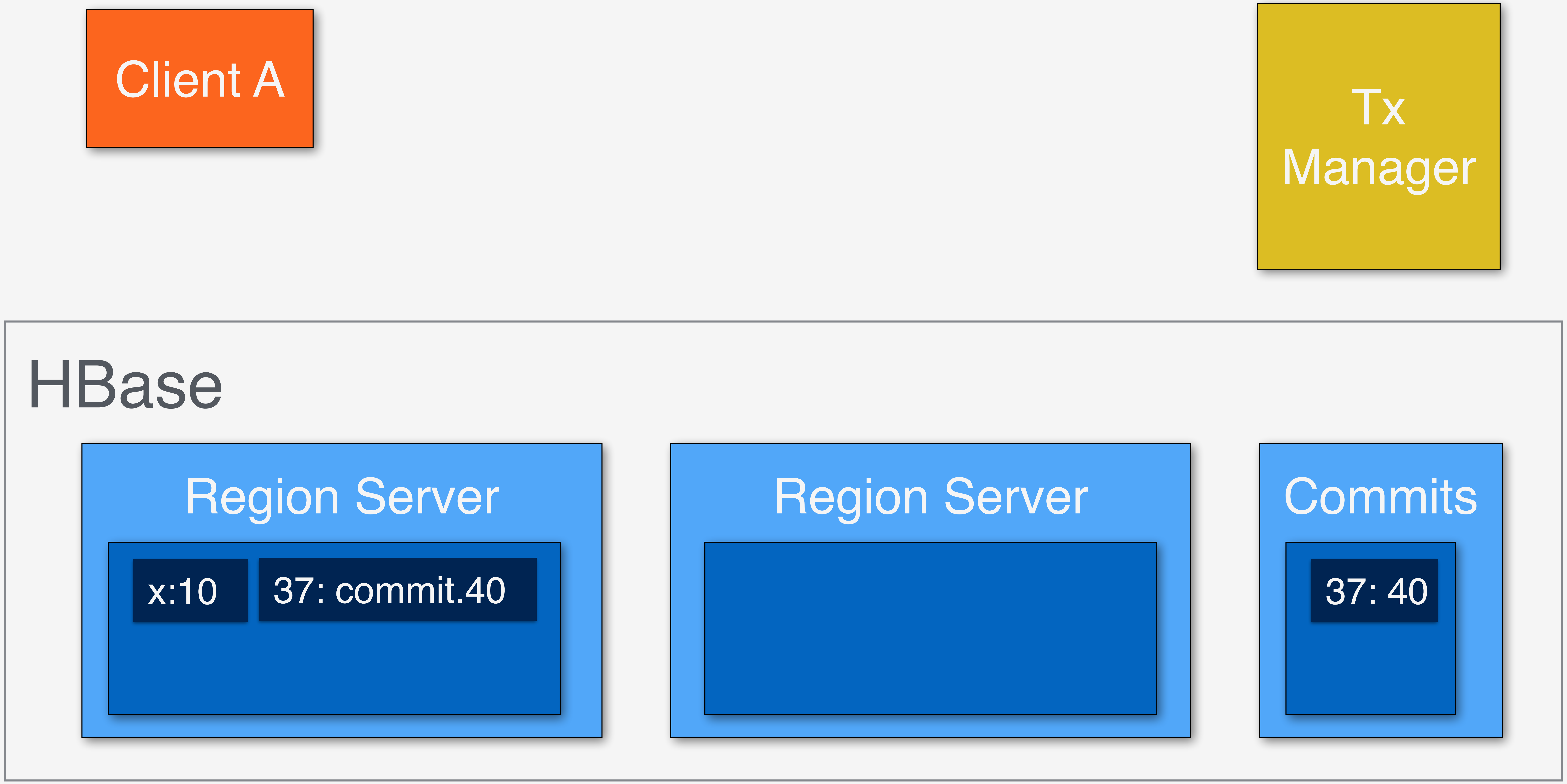
  Daniel Peng, Frank Dabek: *Large-scale Incremental Processing Using Distributed Transactions and Notifications,* USENIX 2010.

- Idea: Move as much transaction state as possible into HBase
  - Shadow cells represent the state of a transaction
  - One shadow cell for every data cell written
  - Track committed transactions in an HBase table
  - Transaction Manager (TSO) has only 3 tasks
    - issue transaction IDs
    - conflict detection
    - write to commit table

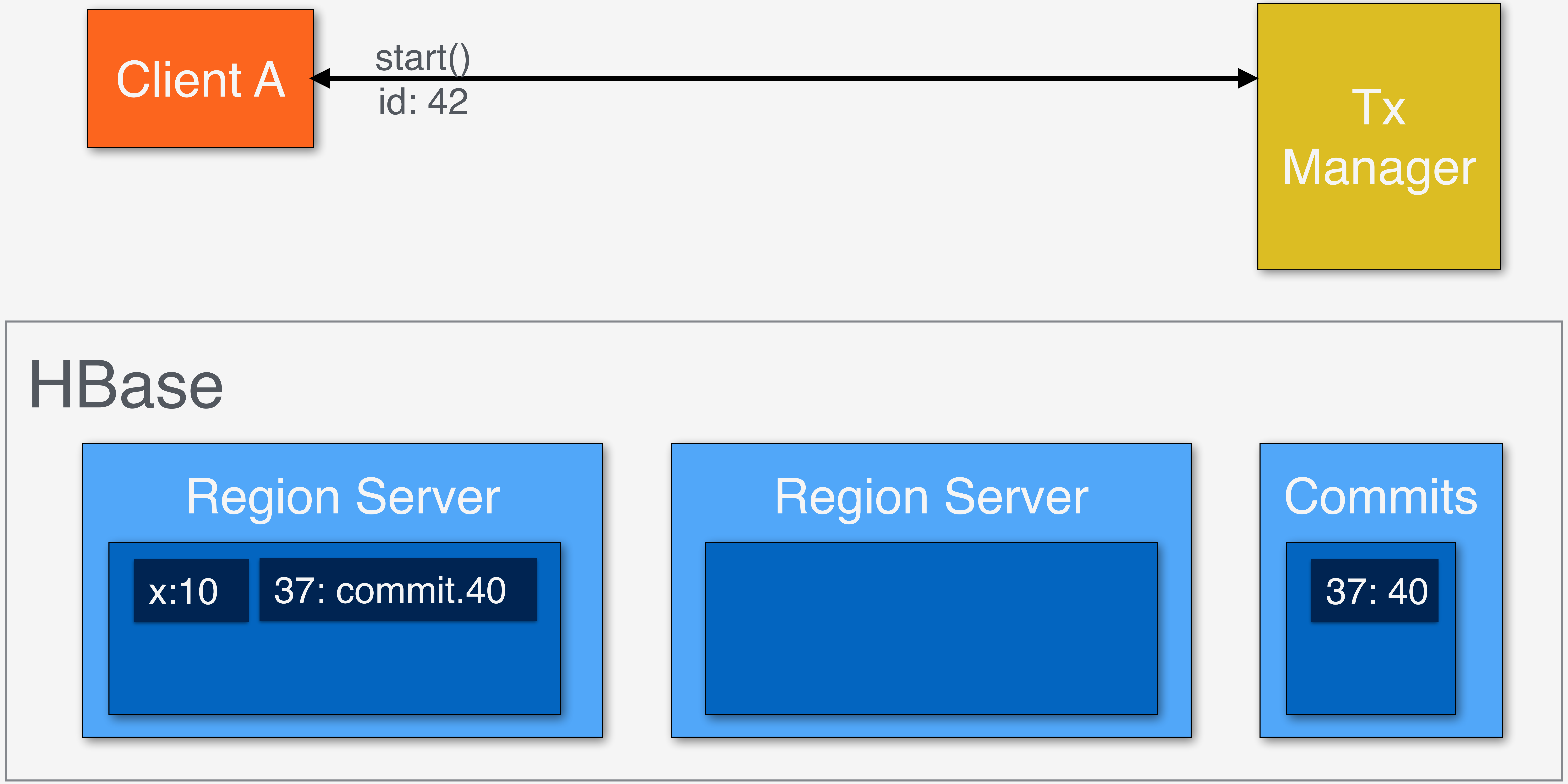
Apache Omid


Transactional App
Omid Client
R/W data
Start/Commit TXs
Guarantee SI
App Table
Shadow Cells
HBase
Keep track & Validate TXs
Transaction Status Oracle (TSO)
Commit data
Commit Table
Get Start/Commit Timestamps
Compactor
Timestamp Oracle
YAHOO!

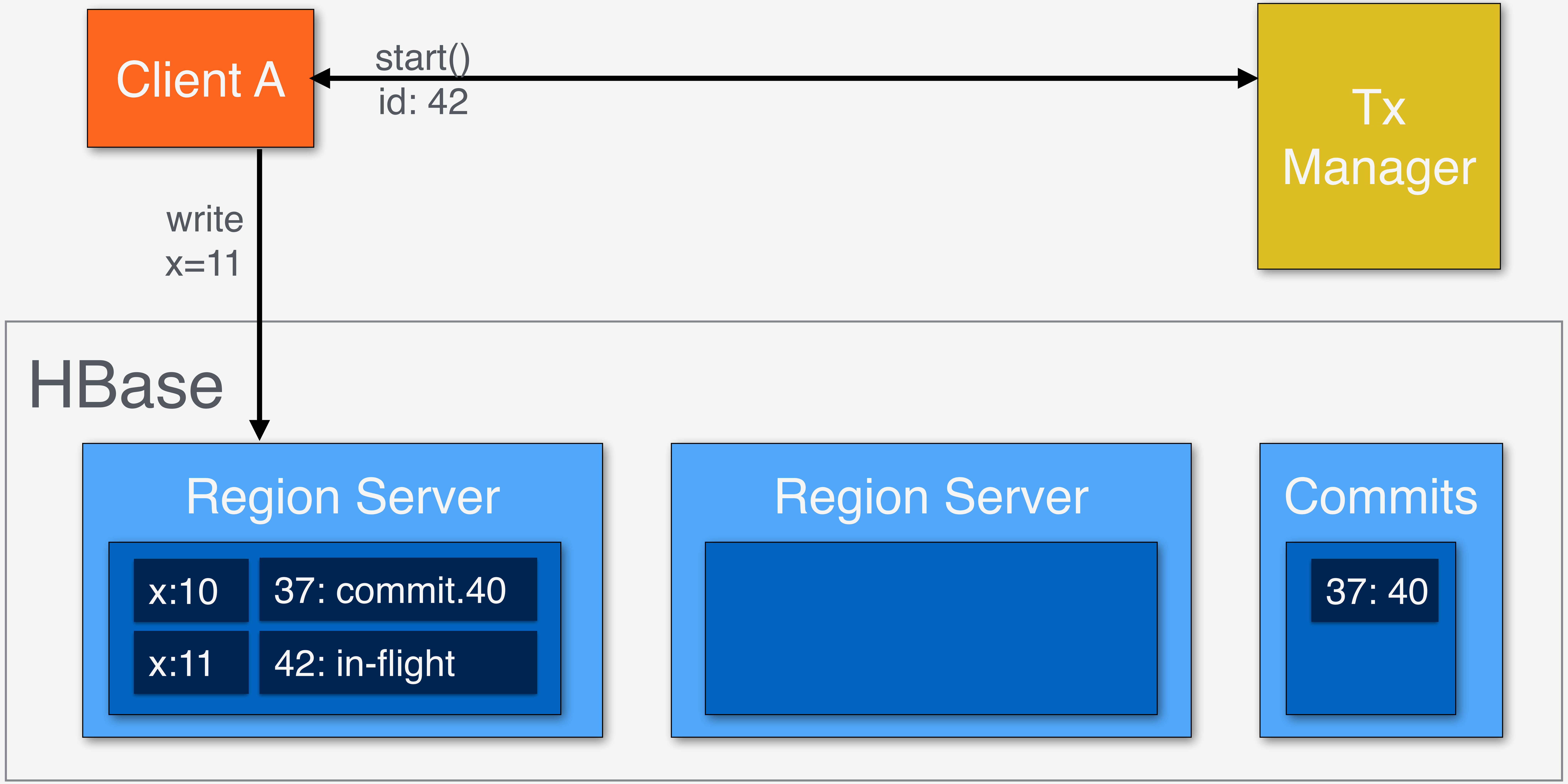
Apache Omid


Client A
Tx Manager
HBase
Region Server
x:10
37: commit.40
Region Server
Commits
37: 40

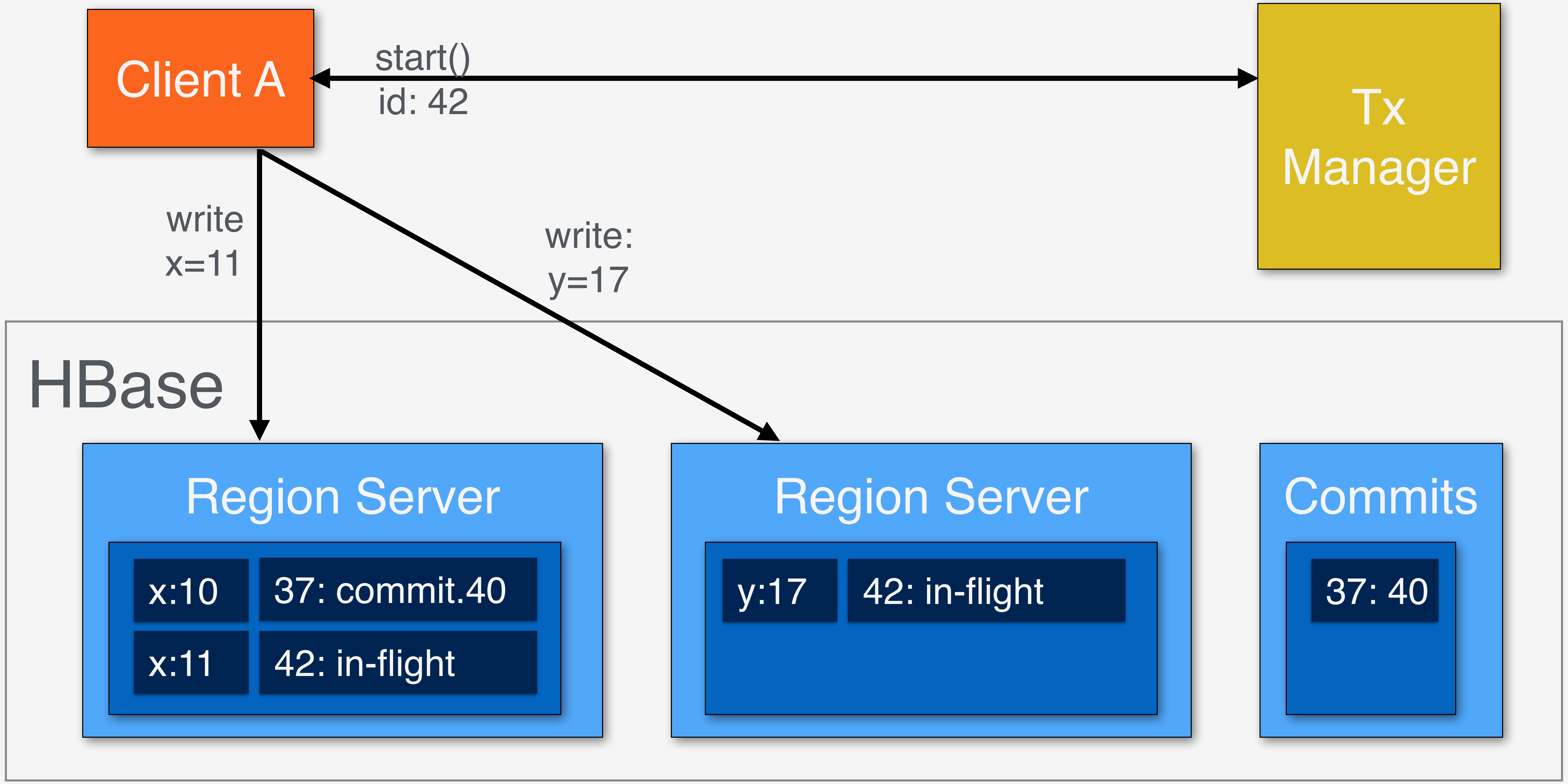
Apache Omid


Client A
start()
id: 42
Tx Manager
HBase
Region Server
x:10
37: commit.40
Region Server
Commits
37: 40

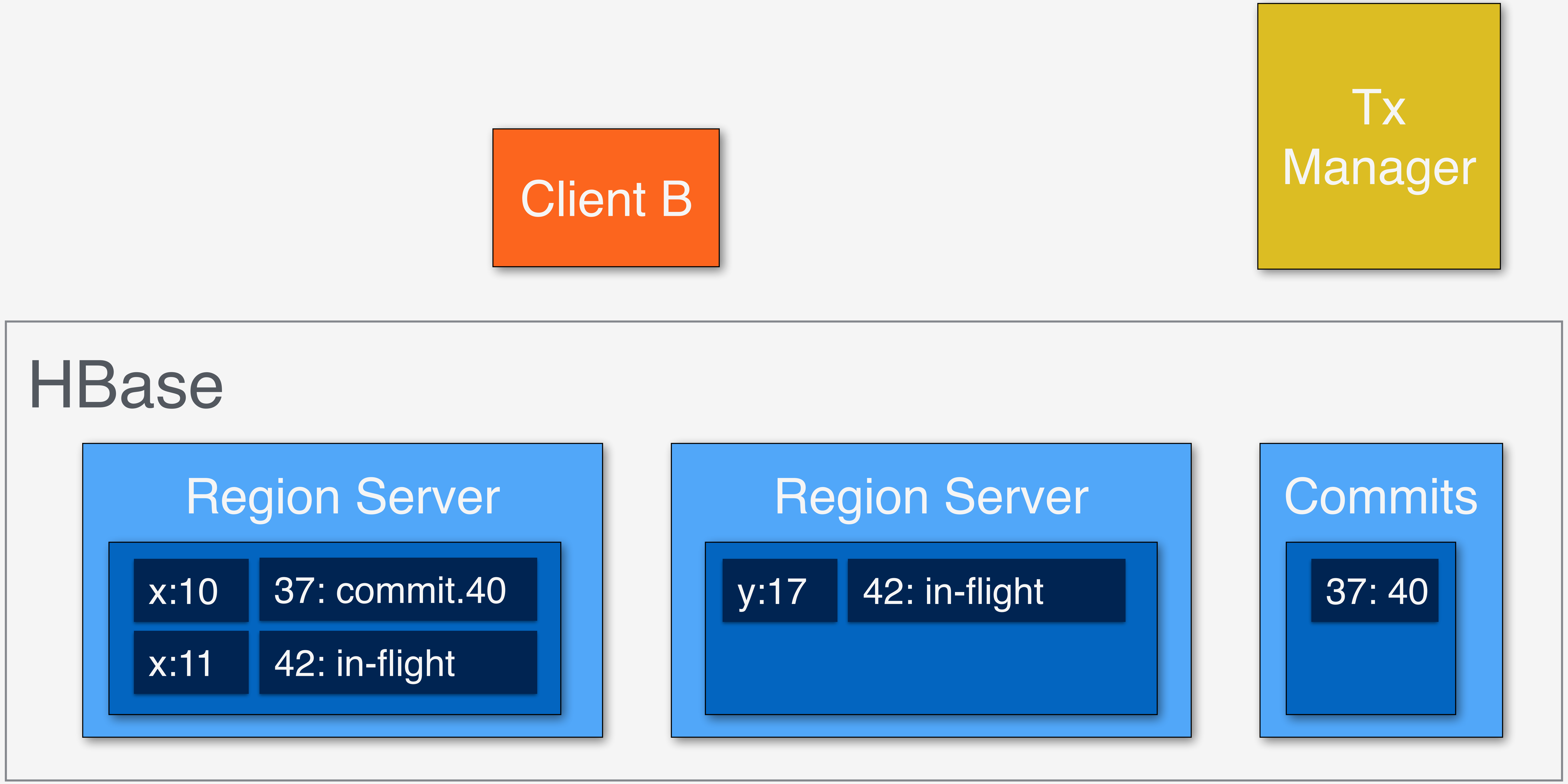
Apache Omid


Client A
start()
id: 42
Tx Manager
write
x=11
HBase
Region Server
x:10
37: commit.40
x:11
42: in-flight
Region Server
Commits
37: 40

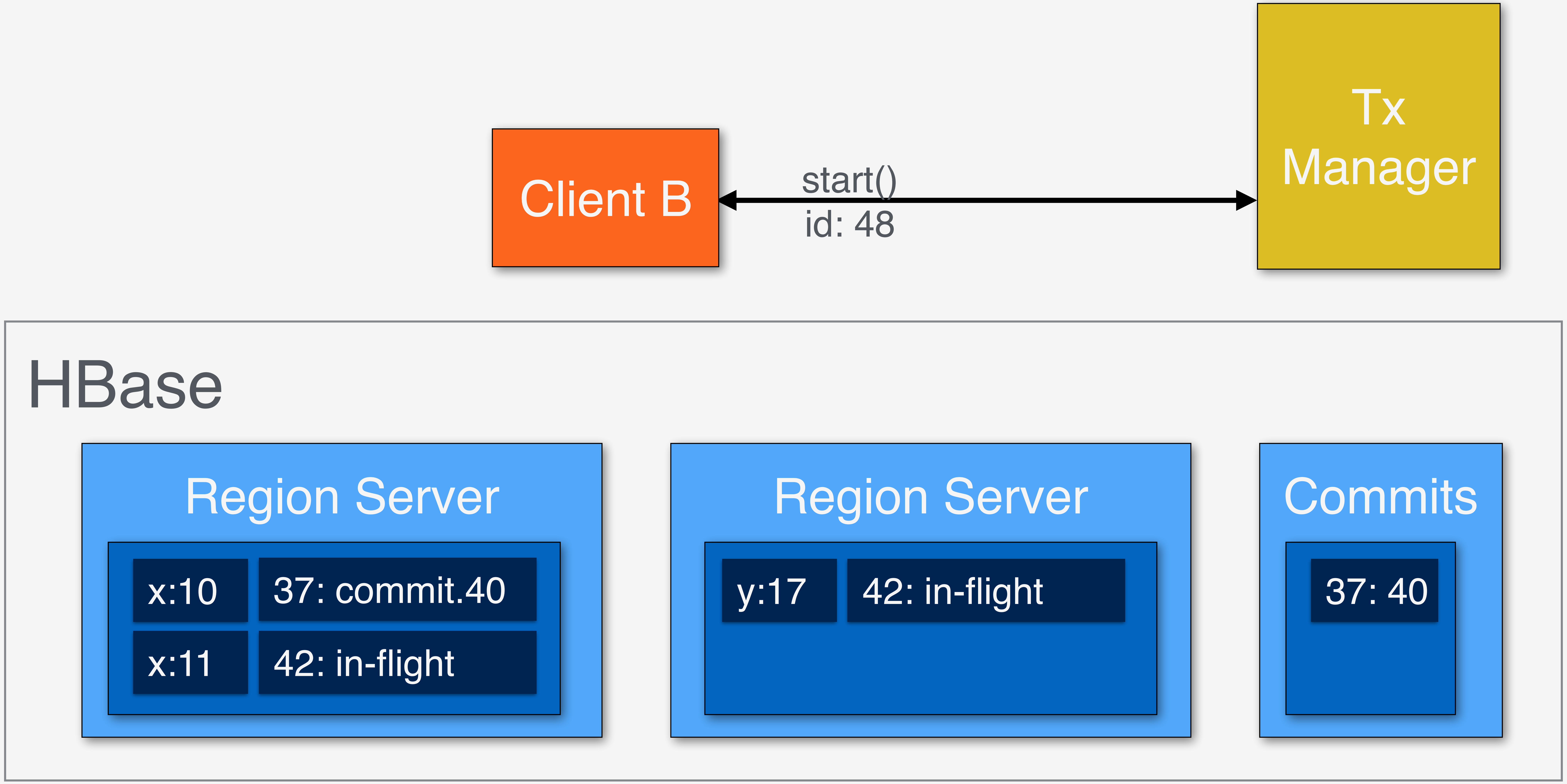
Apache Omid


Client A
start()
id: 42
Tx
Manager
write
x=11
write:
y=17
HBase
Region Server
x:10
37: commit.40
x:11
42: in-flight
Region Server
y:17
42: in-flight
Commits
37: 40

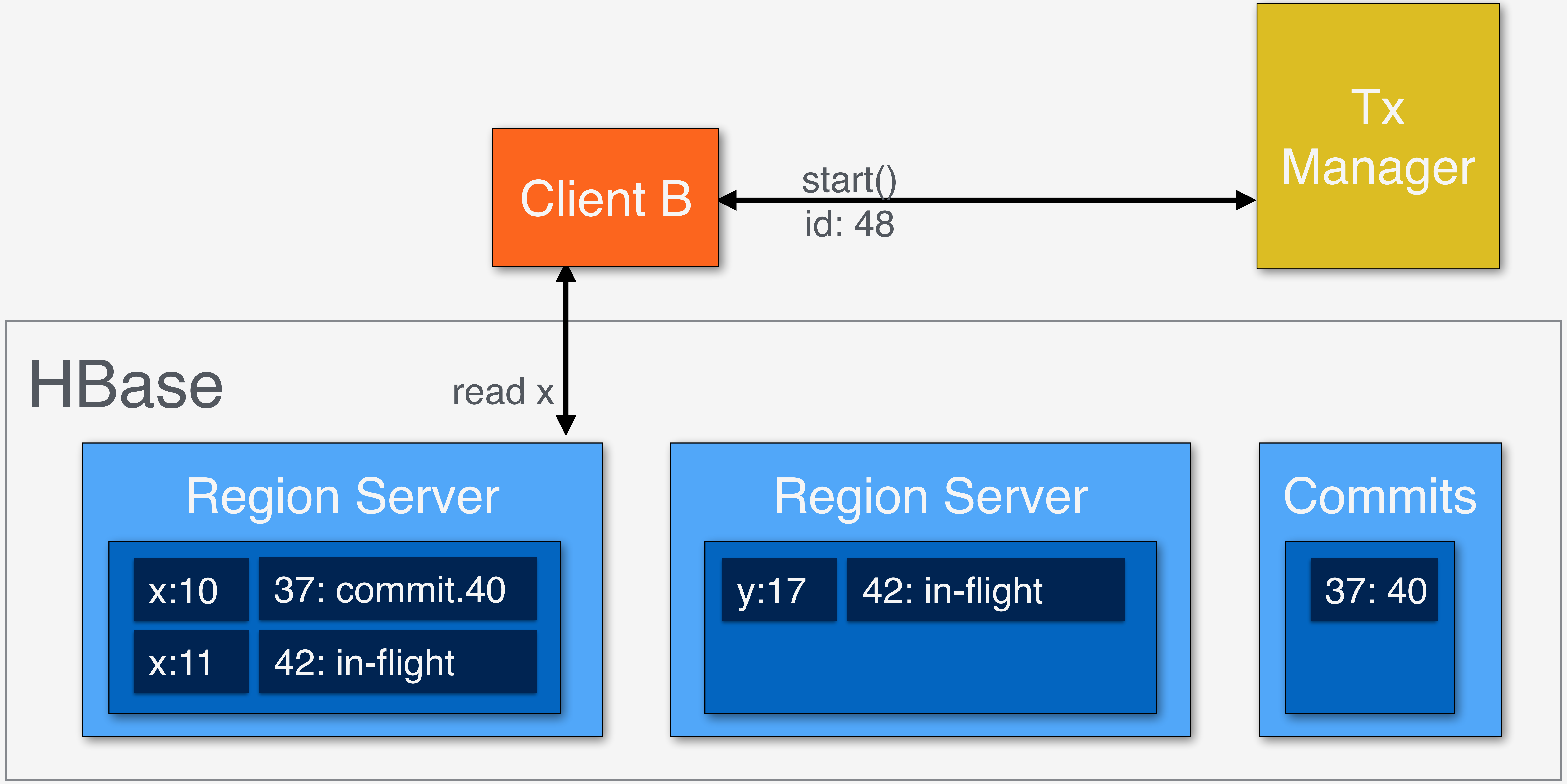
Apache Omid


Client B
Tx Manager
HBase
Region Server
x:10
37: commit.40
x:11
42: in-flight
Region Server
y:17
42: in-flight
Commits
37: 40

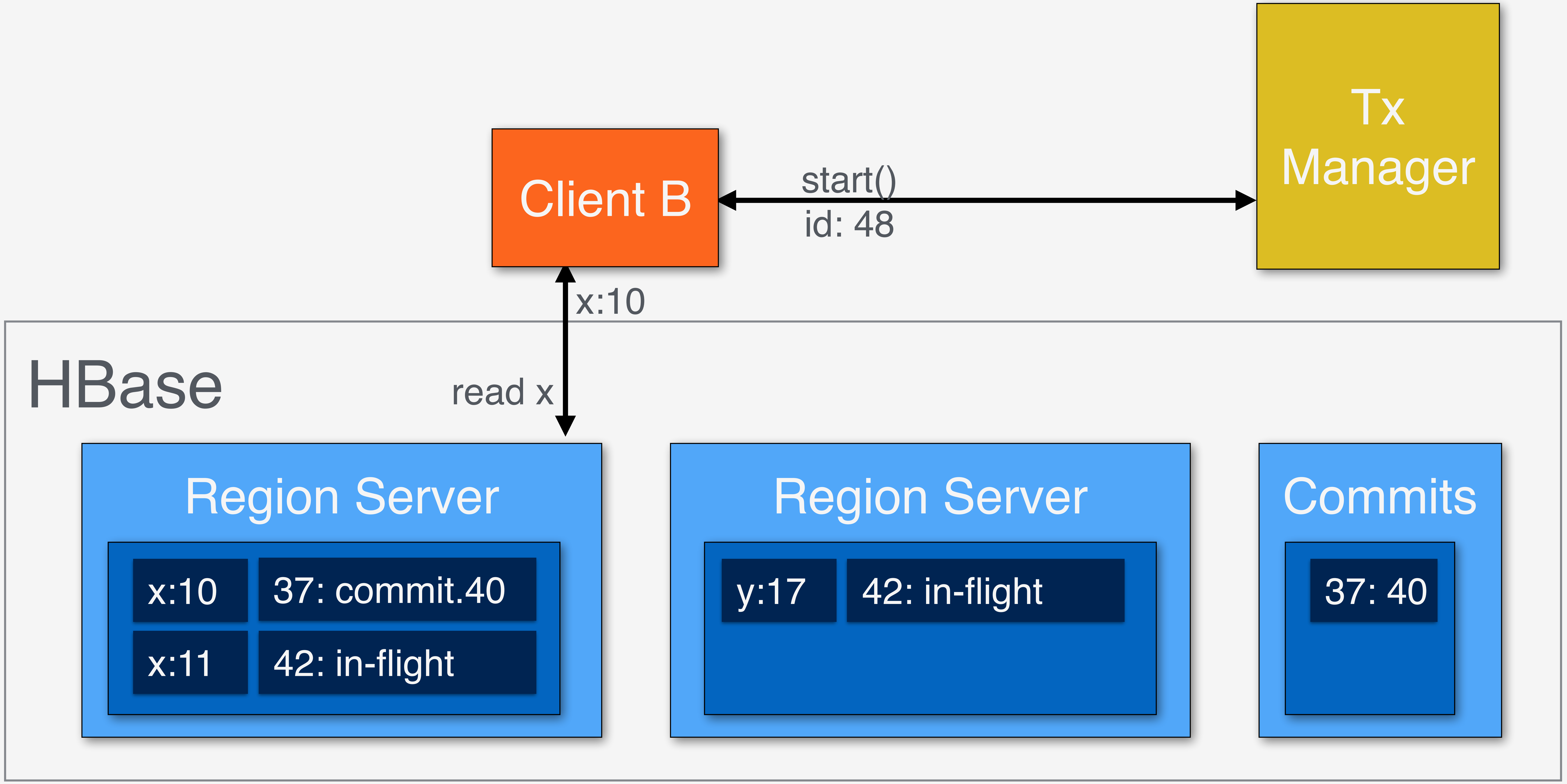
Apache Omid


Client B
start()
id: 48
Tx Manager
HBase
Region Server
x:10
37: commit.40
x:11
42: in-flight
Region Server
y:17
42: in-flight
Commits
37: 40

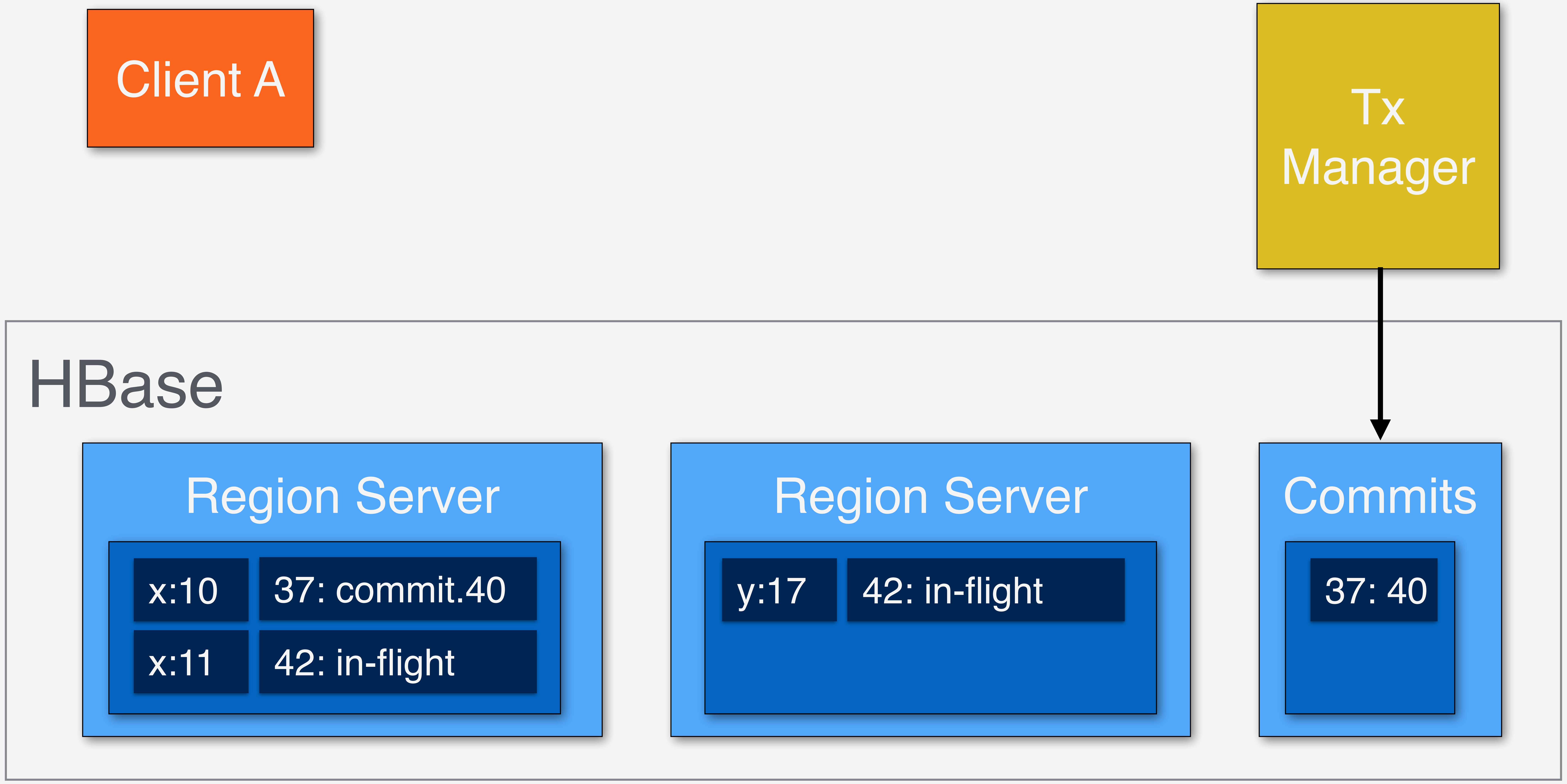
Apache Omid


Client B
start()
id: 48
Tx
Manager
HBase
read x
Region Server
x:10
37: commit.40
x:11
42: in-flight
Region Server
y:17
42: in-flight
Commits
37: 40

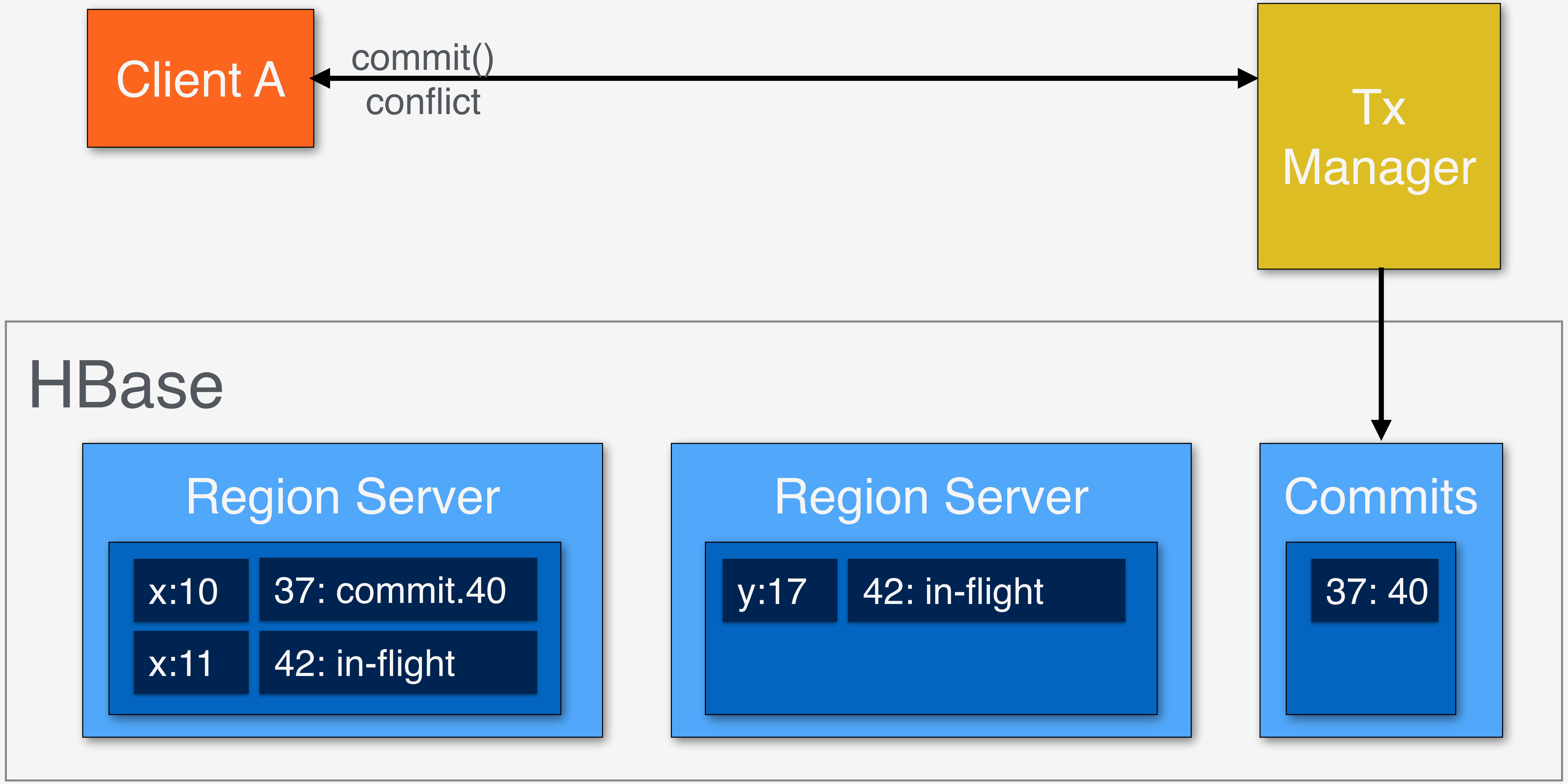
Apache Omid


Tx Manager
Client B
start()
id: 48
x:10
read x
HBase
Region Server
x:10
37: commit.40
x:11
42: in-flight
Region Server
y:17
42: in-flight
Commits
37: 40

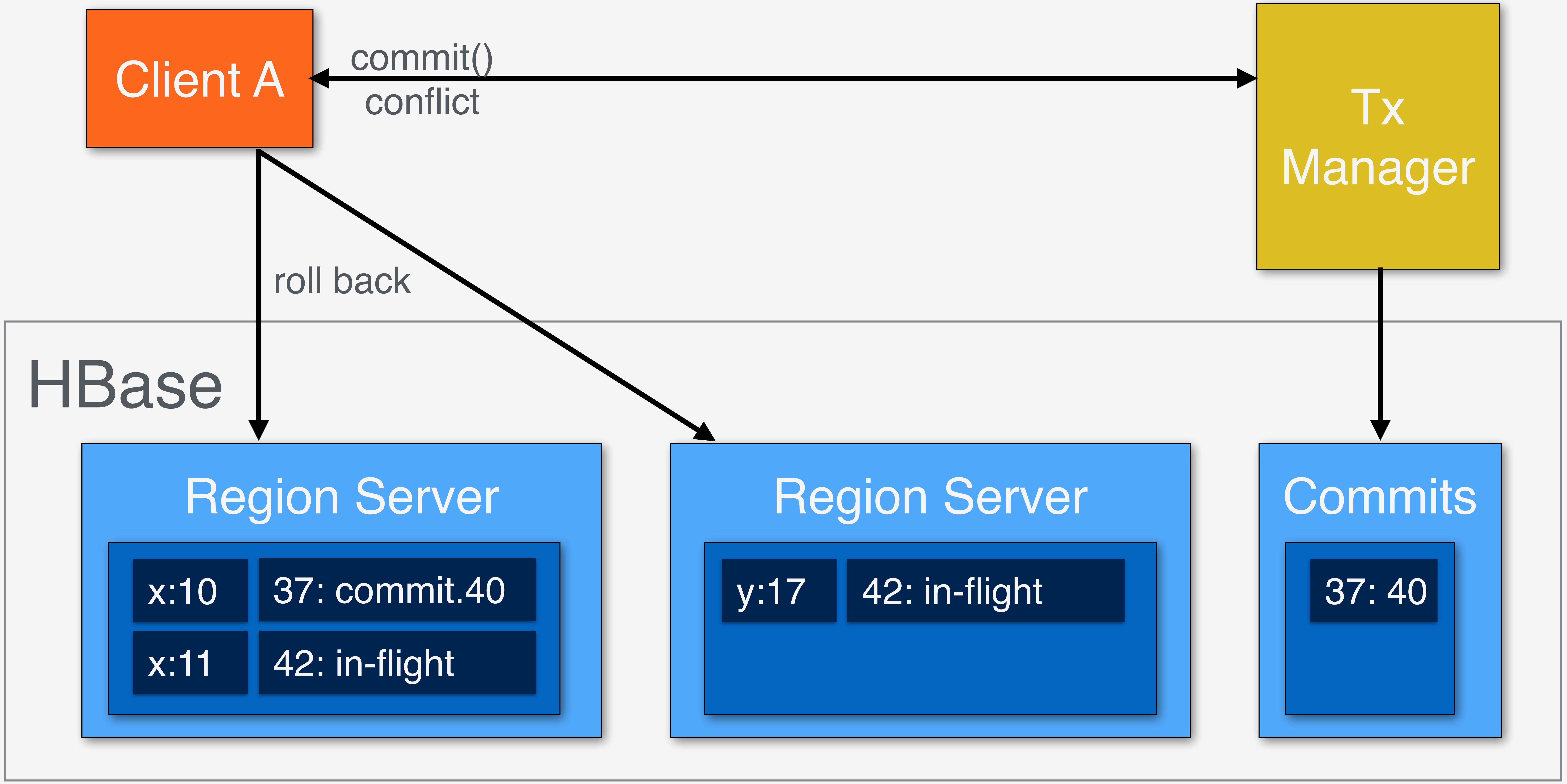
Apache Omid


Client A
Tx Manager
HBase
Region Server
x:10
37: commit.40
x:11
42: in-flight
Region Server
y:17
42: in-flight
Commits
37: 40

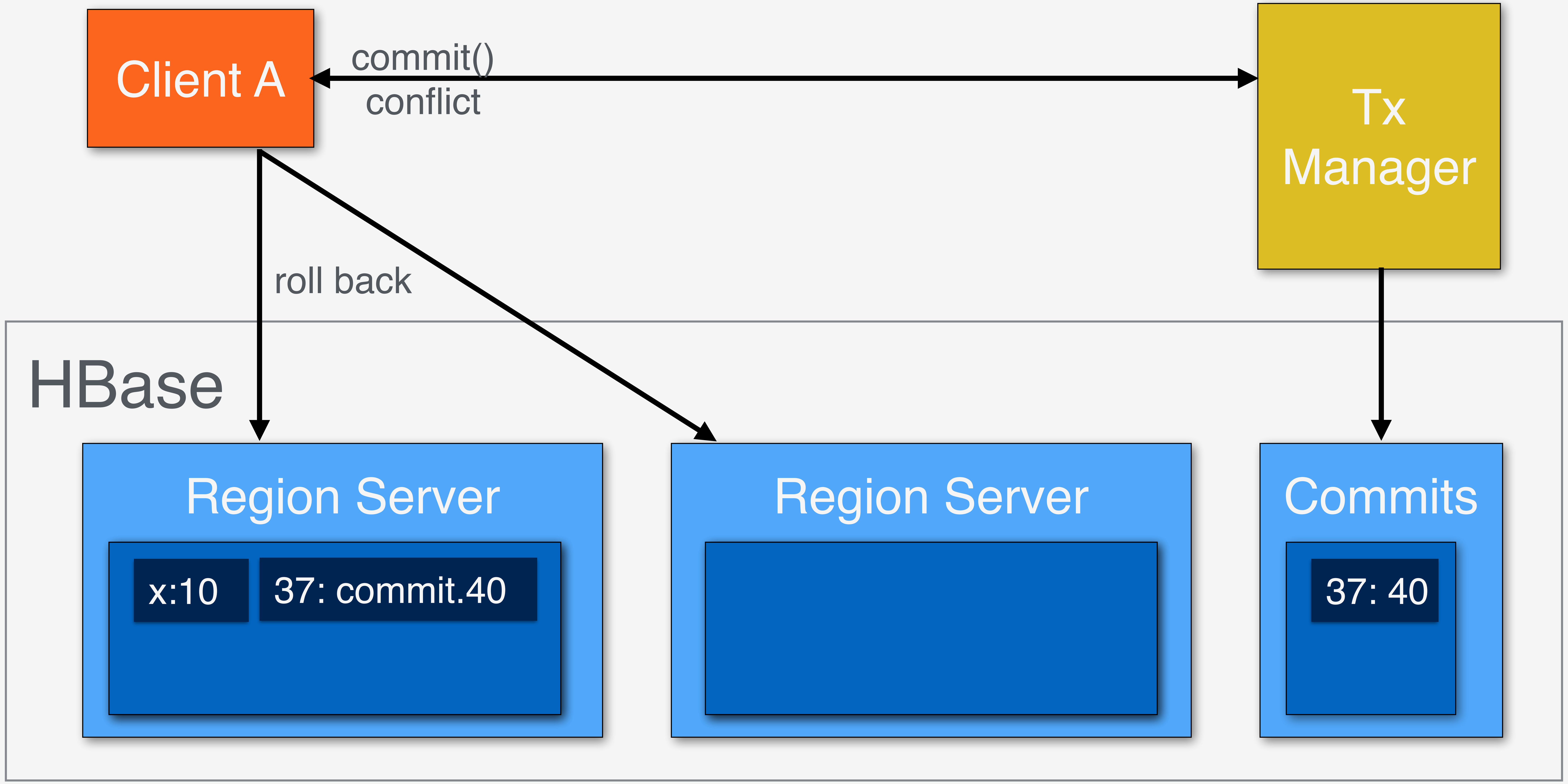
Apache Omid


Client A
commit()
conflict
Tx Manager
HBase
Region Server
x:10
37: commit.40
x:11
42: in-flight
Region Server
y:17
42: in-flight
Commits
37: 40

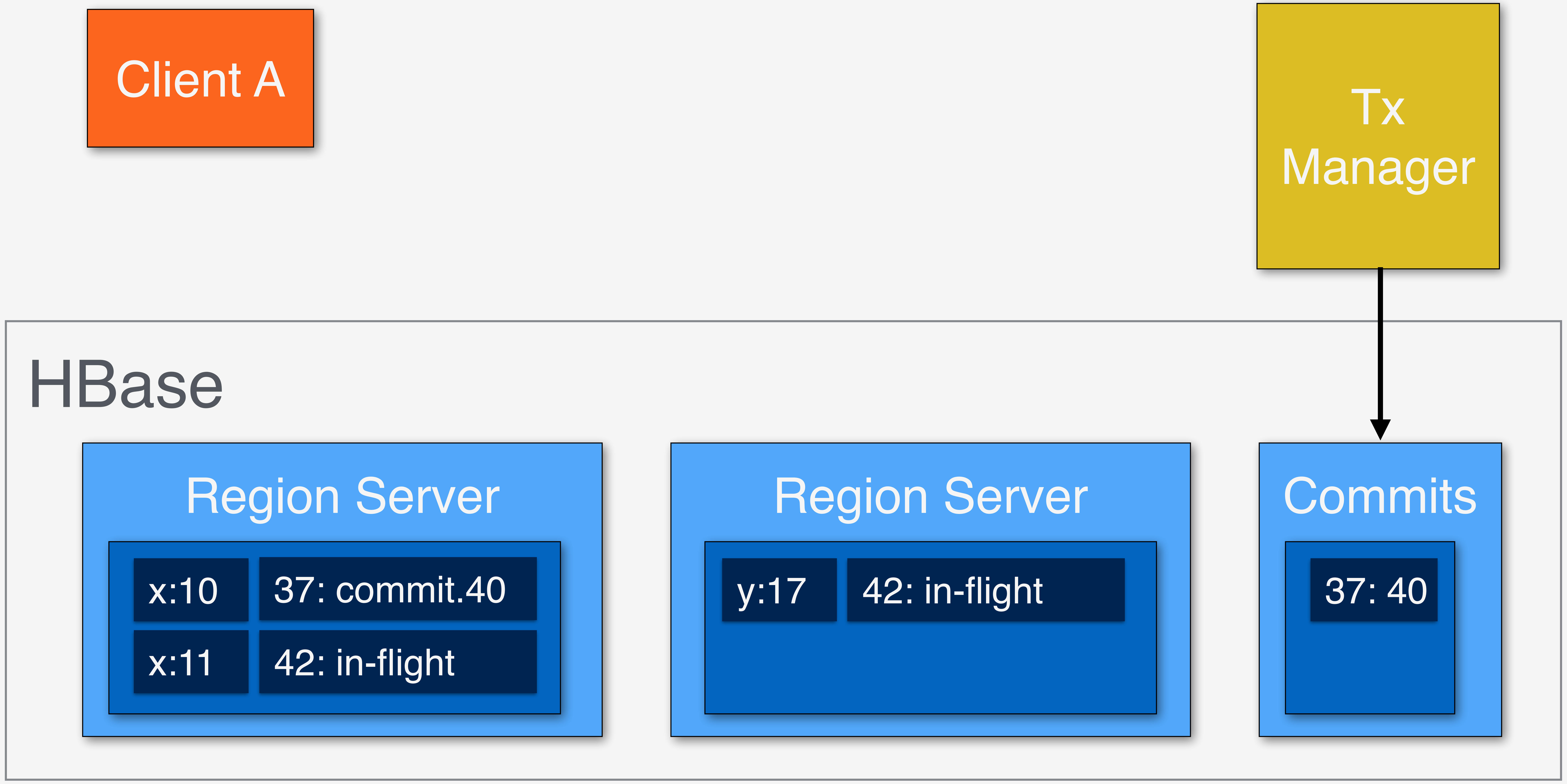
Apache Omid


Client A
commit()
conflict
Tx Manager
roll back
HBase
Region Server
x:10
37: commit.40
x:11
42: in-flight
Region Server
y:17
42: in-flight
Commits
37: 40

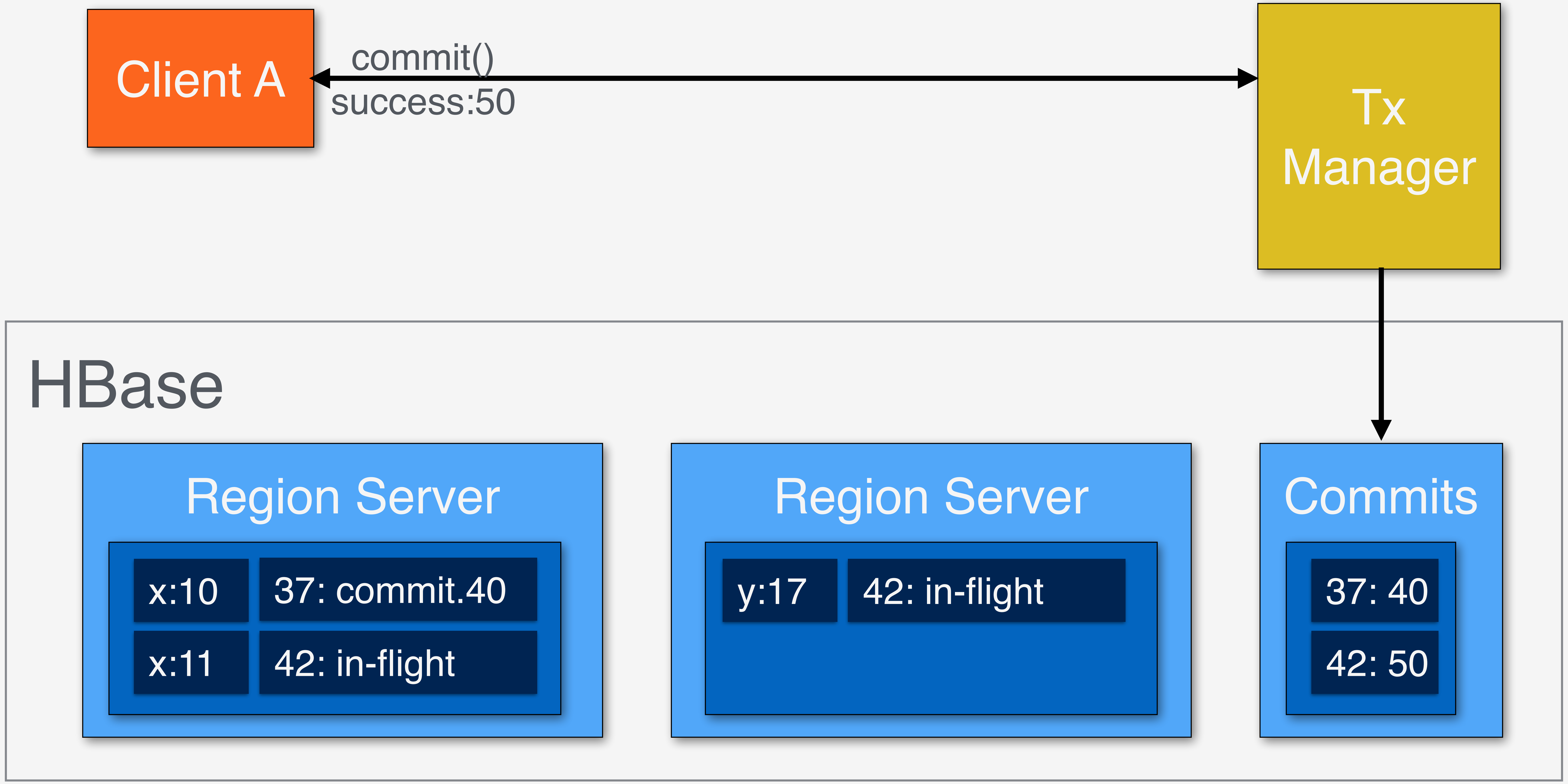
Apache Omid


Client A
commit()
conflict
Tx Manager
roll back
HBase
Region Server
x:10
37: commit.40
Region Server
Commits
37: 40

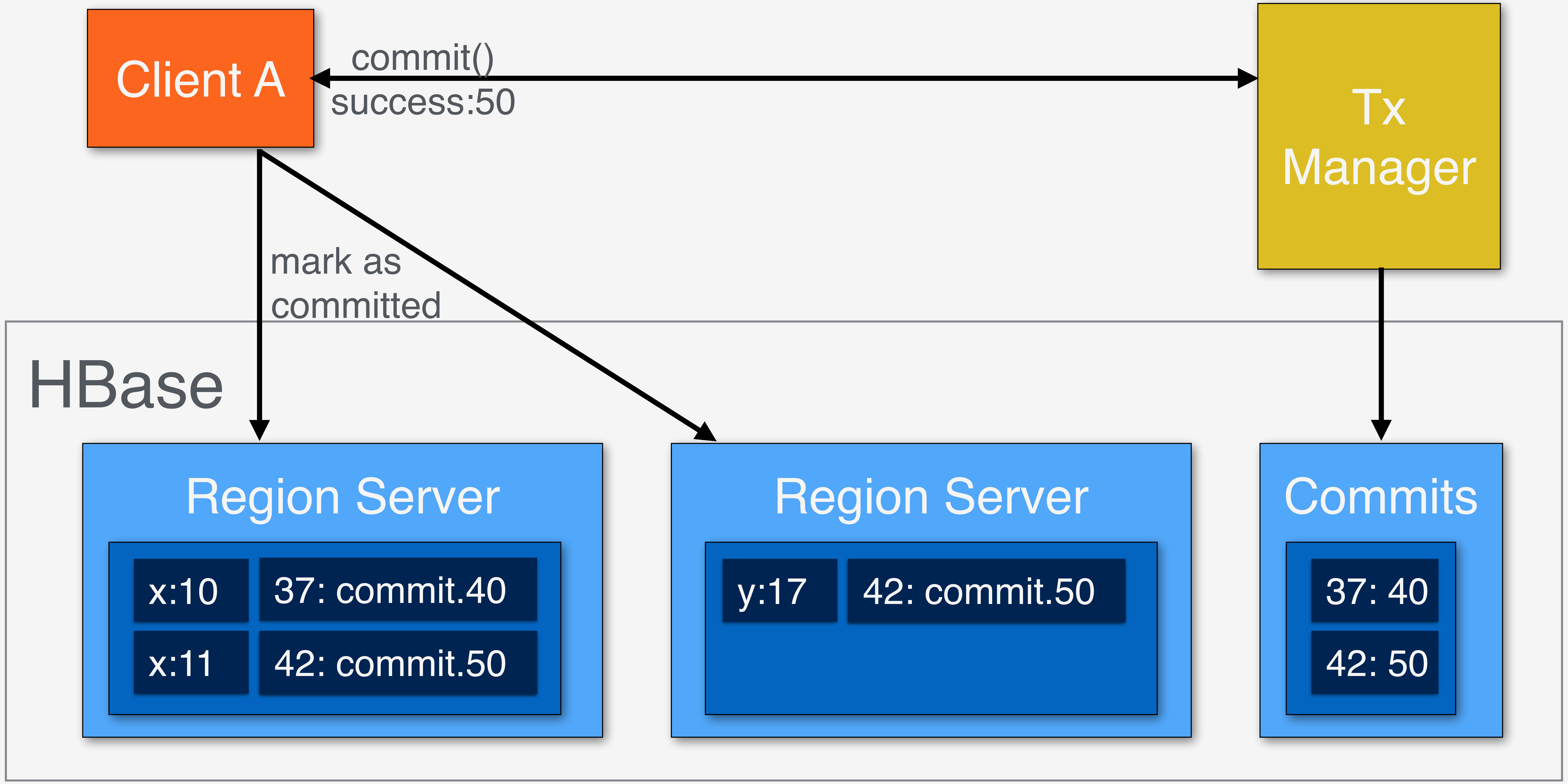
Apache Omid


Client A
Tx Manager
HBase
Region Server
x:10
37: commit.40
x:11
42: in-flight
Region Server
y:17
42: in-flight
Commits
37: 40

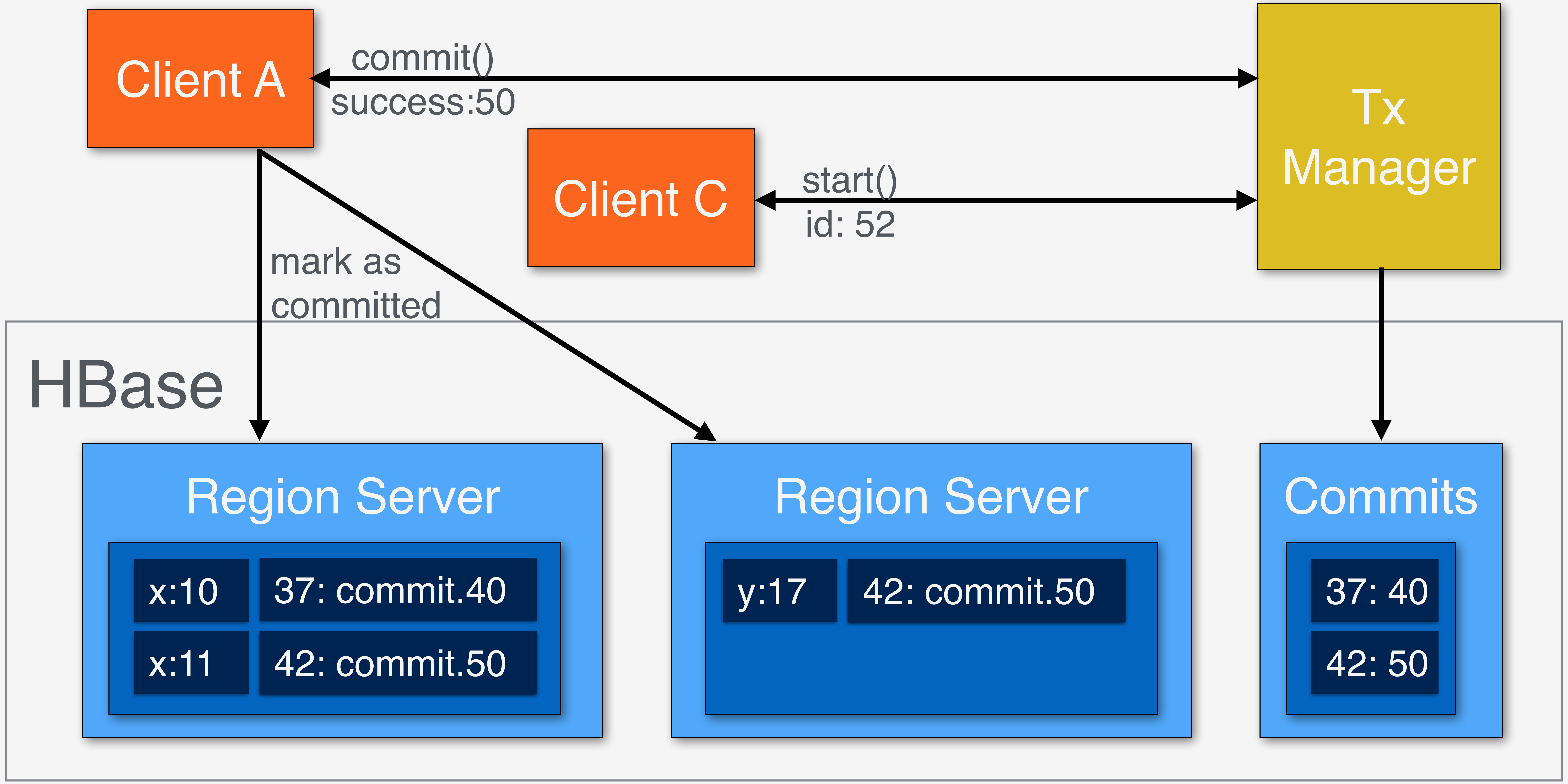
Apache Omid


Client A
commit()
success:50
Tx Manager
HBase
Region Server
x:10
37: commit.40
x:11
42: in-flight
Region Server
y:17
42: in-flight
Commits
37: 40
42: 50

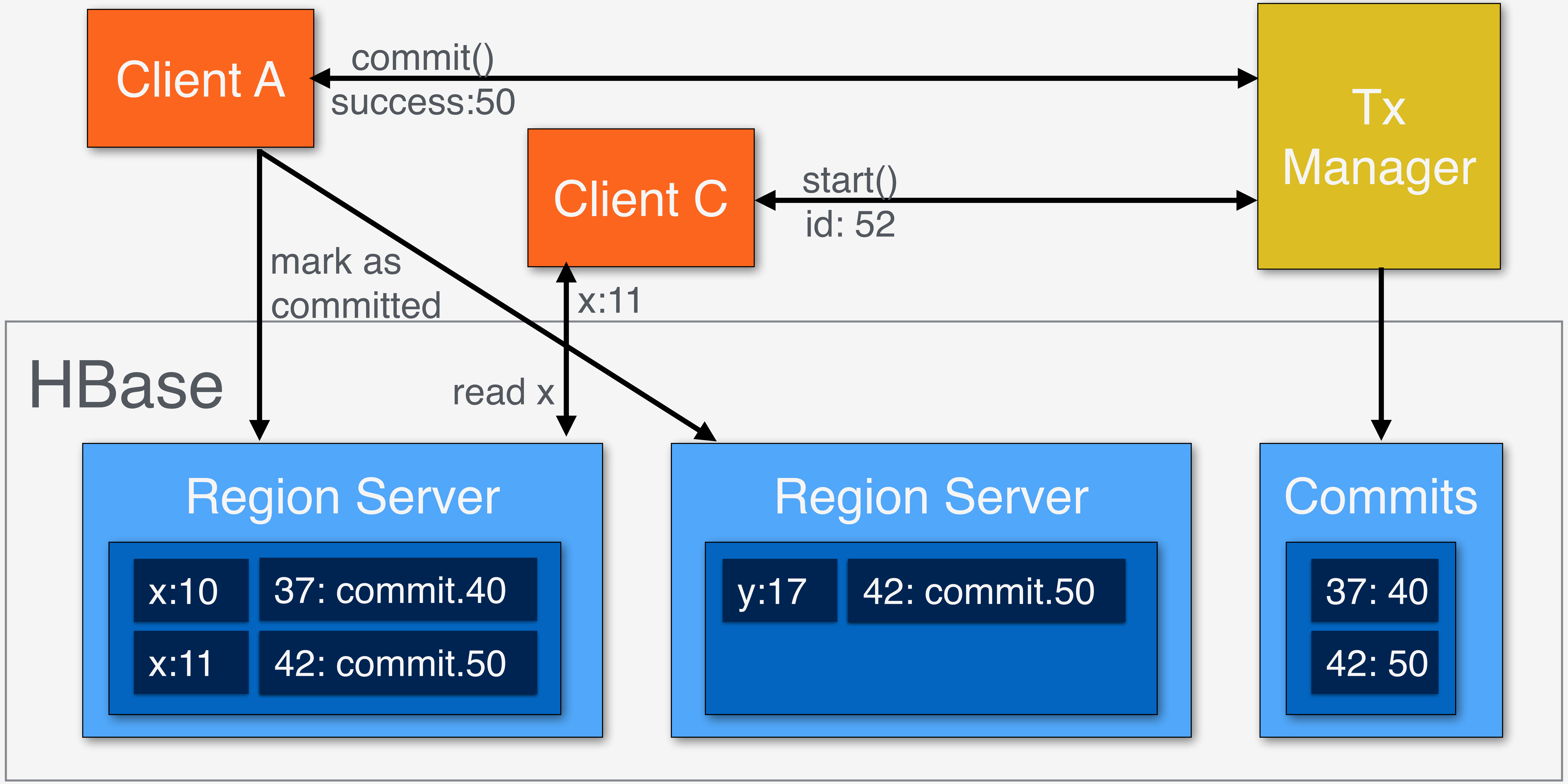
Apache Omid


Client A
commit()
success:50
Tx Manager
mark as committed
HBase
Region Server
x:10
37: commit.40
x:11
42: commit.50
Region Server
y:17
42: commit.50
Commits
37: 40
42: 50

Apache Omid

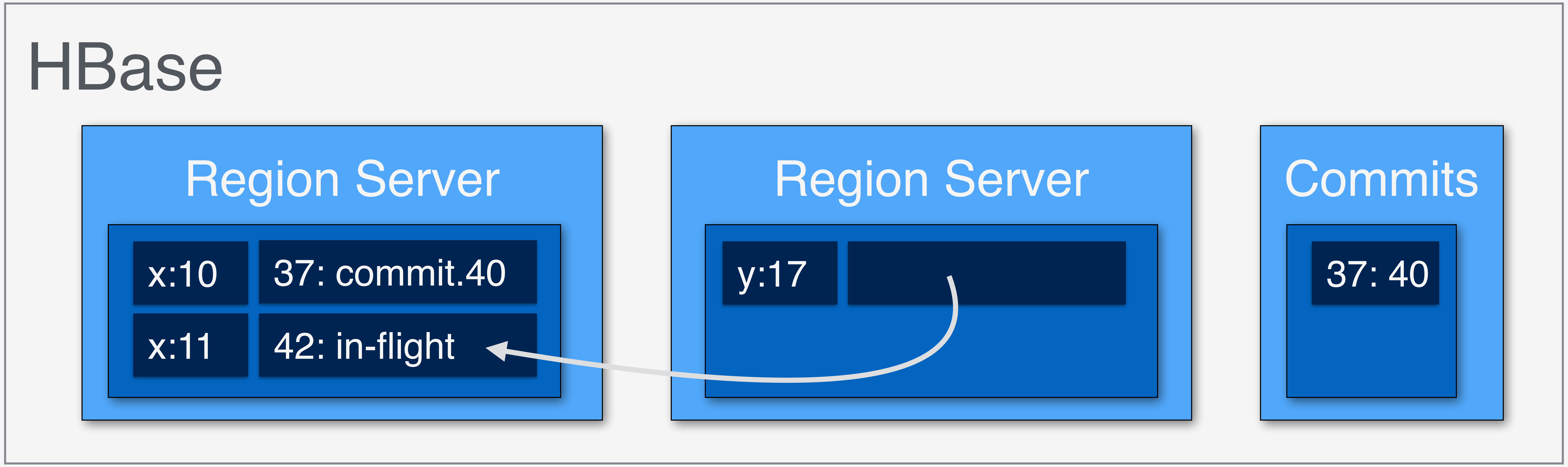

Client A
commit()
success:50
Client C
start()
id: 52
Tx Manager
mark as committed
HBase
Region Server
x:10
37: commit.40
x:11
42: commit.50
Region Server
y:17
42: commit.50
Commits
37: 40
42: 50

Apache Omid

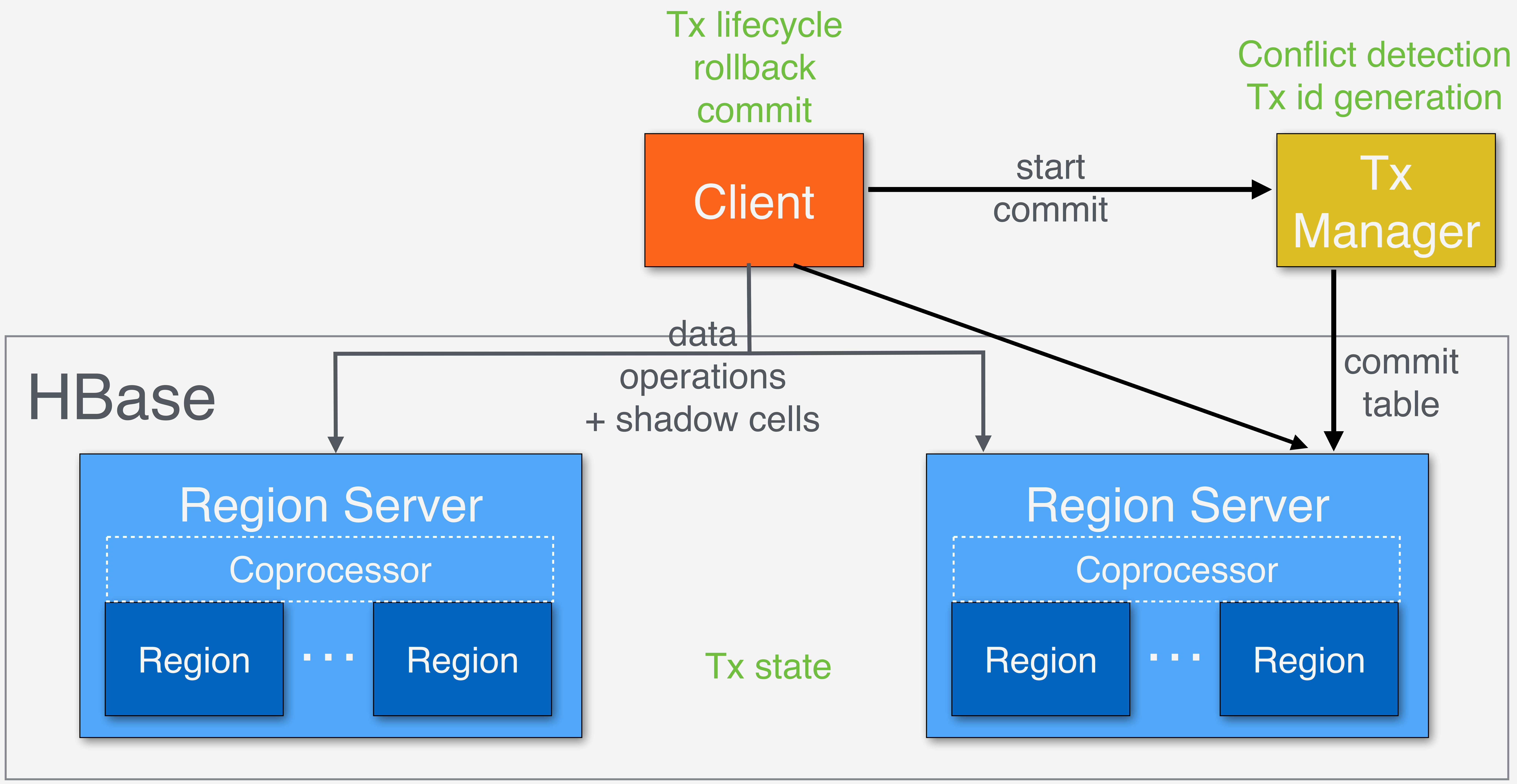

Client A
commit()
success:50
Client C
start()
id: 52
Tx Manager
mark as committed
x:11
read x
HBase
Region Server
x:10
37: commit.40
x:11
42: commit.50
Region Server
y:17
42: commit.50
Commits
37: 40
42: 50

Apache Omid - Future

- Atomic commit with linking?
    - Eliminate need for commit table



Apache Omid



Tx lifecycle
rollback
commit
Conflict detection
Tx id generation
Client
start
commit
Tx Manager
data operations + shadow cells
commit table
HBase
Region Server
Coprocessor
Region
. . .
Region
Tx state
Region Server
Coprocessor
Region
. . .
Region

Apache Omid - Strengths

- Transaction state is in the database
    - Shadow cells plus commit table
    - Scales with the size of the cluster
- Transaction Manager is lightweight
    - Generation of tx IDs delegated to timestamp oracle
    - Conflict detection
    - Writing to commit table
- Fault Tolerance:
    - After failure, fail all existing transactions attempting to commit
    - Self-correcting: Read clients can delete invalid cells

Apache Omid - Not So Strengths

- Storage intensive - shadow cells double the space
- I/O intensive - every cell requires two writes
    1. write data and shadow cell
    2. record commit in shadow cell
- Reads may also require two reads from HBase (commit table)
- Producer/Consumer: will often find the (uncommitted) shadow cell
    - Scans: high througput sequential read disrupted by frequent lookups
- Security/Multi-tenancy:
    - All clients need access to commit table
    - Read clients need write access to repair invalid data
- Replication: Not implemented

Summary

	Apache Tephra	Apache Trafodion	Apache Omid
Tx State	Tx Manager	Distributed to region servers	Tx Manager (changes) HBase (shadows/commits)
Conflict detection	Tx Manager	Distributed to regions, 2-phase commit	Tx Manager
ID generation	Tx Manager	Distributed to multiple Tx Managers	Tx Manager
API	HTable	SQL	Custom
Multi-tenant	Yes	Yes	No
Strength	Scans, Large Tx, API	Scalable, full SQL	Scale, throughput
So so	Scale, Throughput	API not Hbase, Large Tx	Scans, Producer/Consumer

Links

Join the community:

Apache Tephra

(incubating)
http://tephra.apache.org/

Apache Omid (incubating)
http://omid.apache.org/

Apache Trafodion

(incubating)
http://trafodion.apache.org/

Thank you

… for listening to my talk.

Credits:

- Sean Broeder, Narendra Goyal (Trafodion)
- Francisco Perez-Sorrosal (Omid)

Thank you

… for listening to my talk.

Credits:

- Sean Broeder, Narendra Goyal (Trafodion)
- Francisco Perez-Sorrosal (Omid)

Questions?