# Transaction Management on Cassandra

10 Sep, 2019 at ApacheCon NA 2019

Hiroyuki Yamada
CTO/CEO at Scalar, Inc.

# Who am I ?

- Hiroyuki Yamada
  - Passionate about Database Systems and Distributed Systems
  - Ph.D. in Computer Science, the University of Tokyo
  - IIS the University of Tokyo, Yahoo! Japan, IBM Japan

# Agenda

- What is Scalar DB
- Transaction Management on Cassandra
- Benchmark and Verification Results

# Maybe You Don't Need ACID Transactions

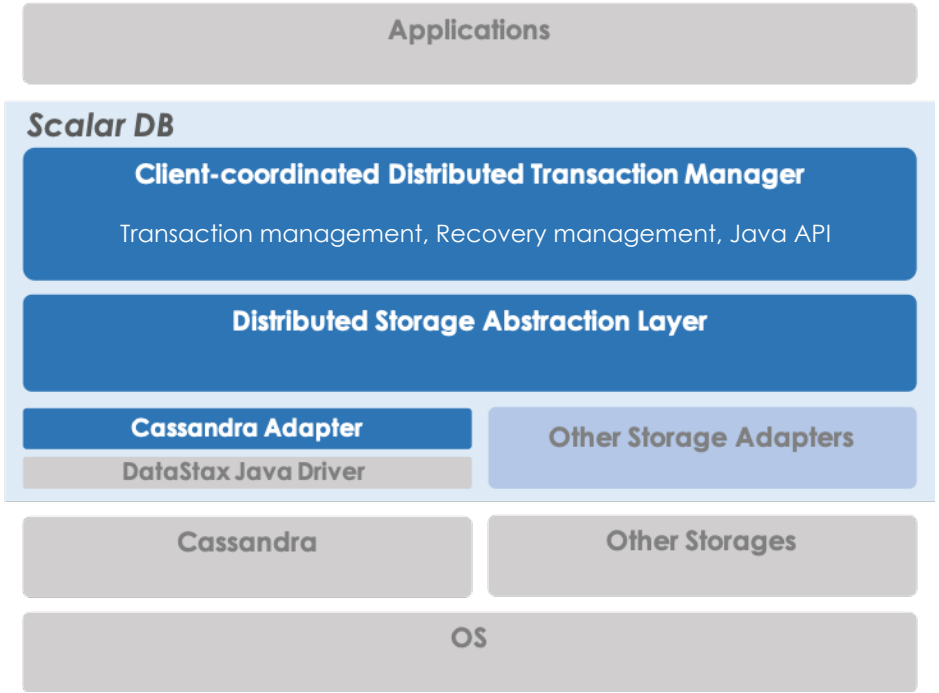
- ACID transactions are heavy
  - Especially when data is distributed

- One of other solutions:
  - Make operations idempotent and retry them if atomicity is not required

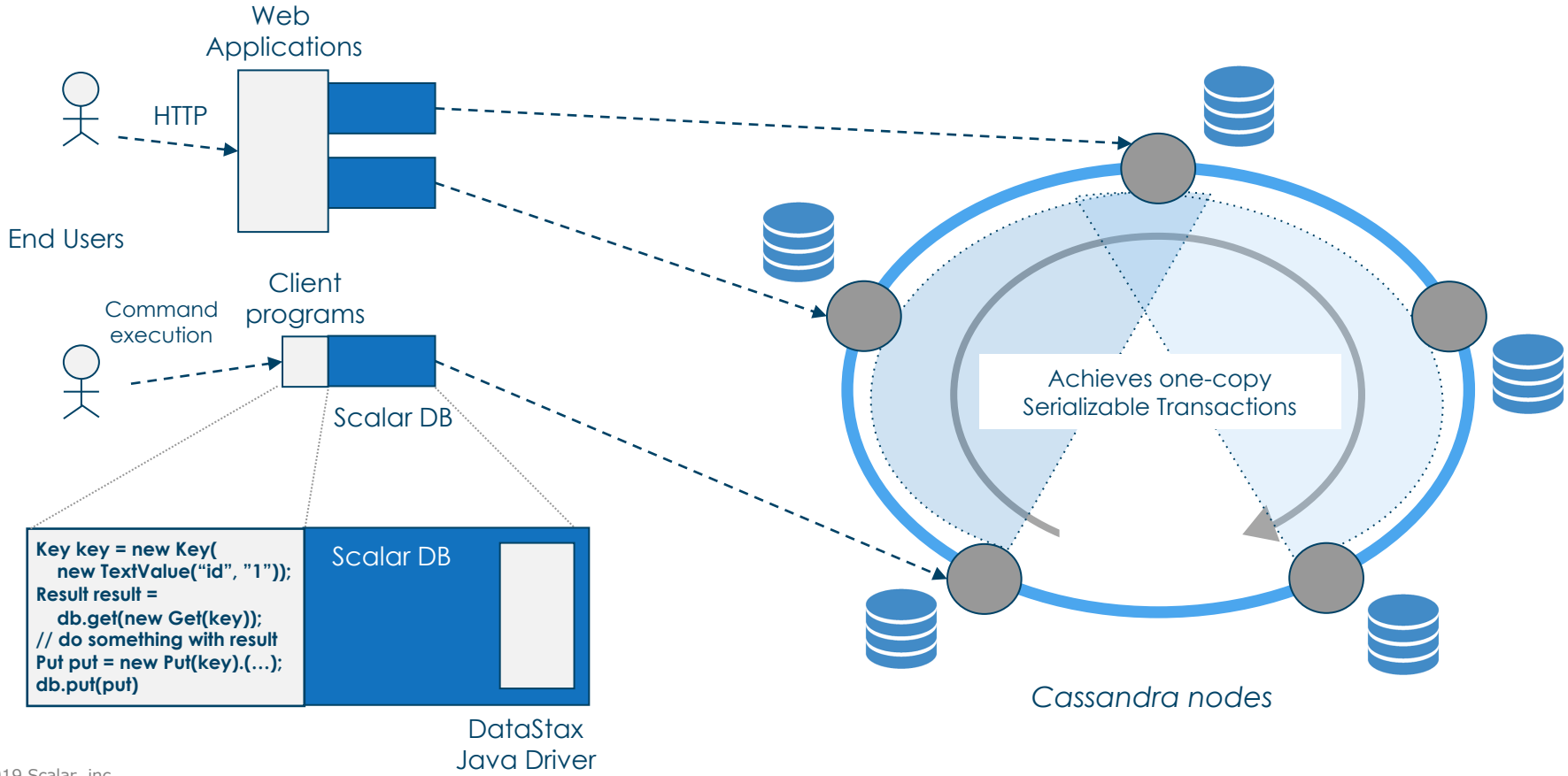
# What is Scalar DB

https://github.com/scalar-labs/scalardb

- A library that makes non-ACID distributed databases ACID-compliant
  - Cassandra is the first supported distributed database



# System Architecture with Scalar DB and Cassandra



# Key Characteristics

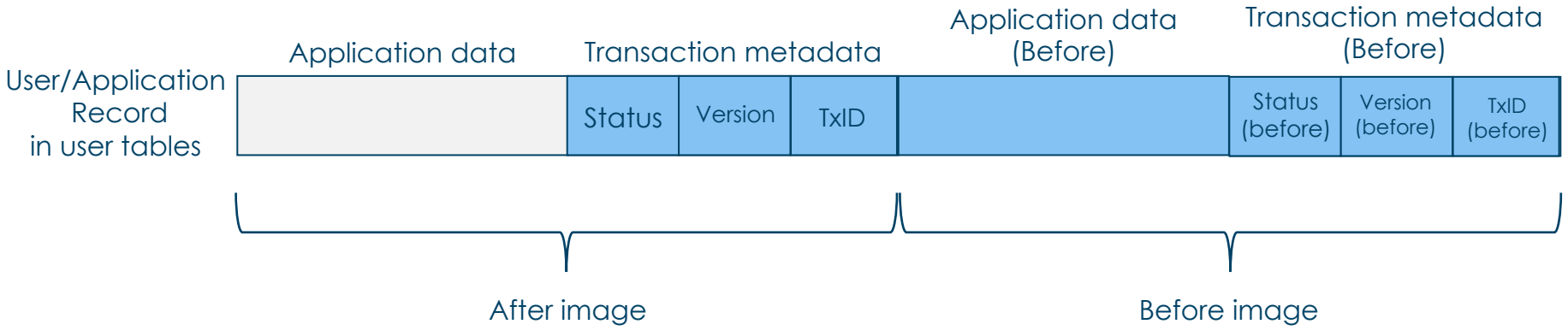
- Non-invasive approach
  - Any modifications to the underlying database are not required
- High availability
  - Available as long as quorum of replicas are up
  - C* high availability is fully sustained by the client-coordinated approach
- Horizontal scalability
  - Throughput scales linearly
  - C* high scalability is fully sustained by the client-coordinated approach
- Strong Consistency
  - Replicas updated by transactions are always consistent and up-to-date

# Transaction Management on Cassandra - Introduction

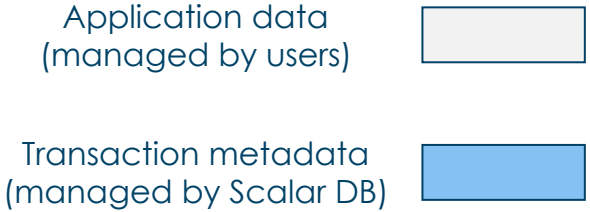
- Based on Cherry Garcia protocol [ICDE'15]
  - Requires minimum set of features such as linearizable conditional update and the ability to store metadata
- Scalar DB is one of the applications of the protocol
  - Use LWT for Linearizability
  - Manage transaction metadata in user record space
- Implement enhancements
  - Protocol correction
  - No use of TrueTime API
  - Serializable support (SI is the default isolation level)

# Transaction Metadata Management

- WAL (Write-Ahead Logging) records are distributed







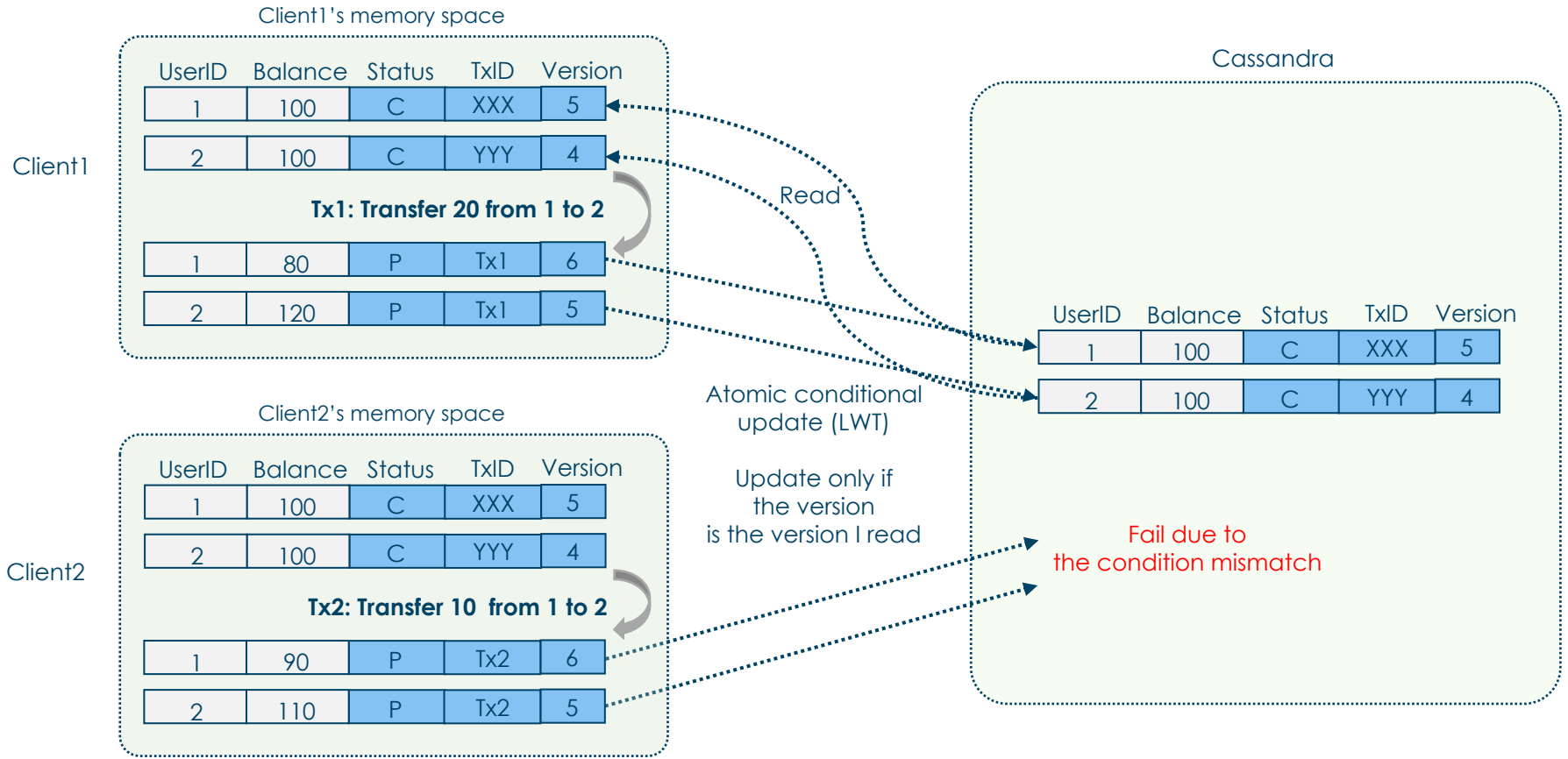
# Transaction Protocol - Overview

- Optimistic concurrency control

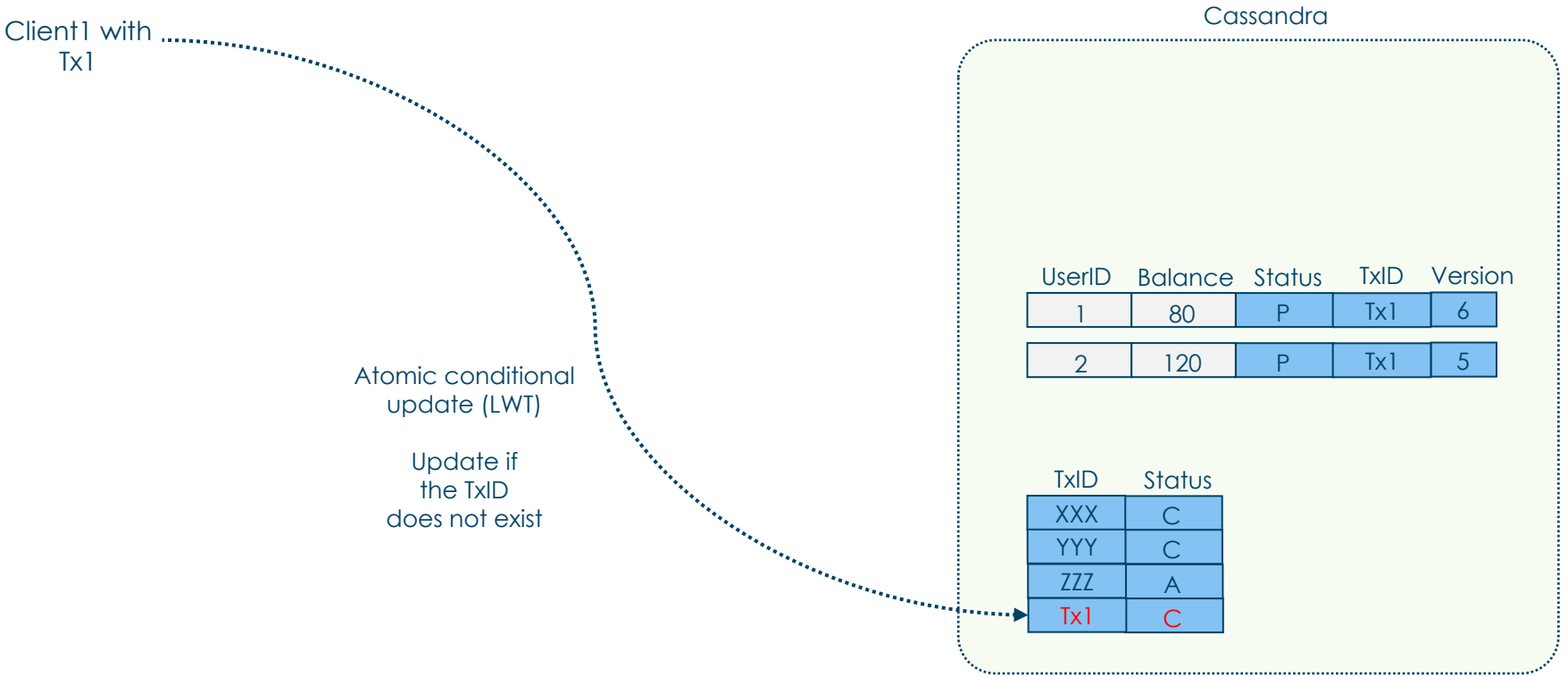
- Similar to 2 phase commit protocol
  - Prepare phase: prepare records
  - Commit phase 1: commit status
    - This is where a transaction is regarded as committed or aborted in normal cases
  - (Commit phase 2: commit records)

- Lazy recovery
  - Uncommitted records will be rollforwarded or rollbacked based on the status of a transaction when the records are read

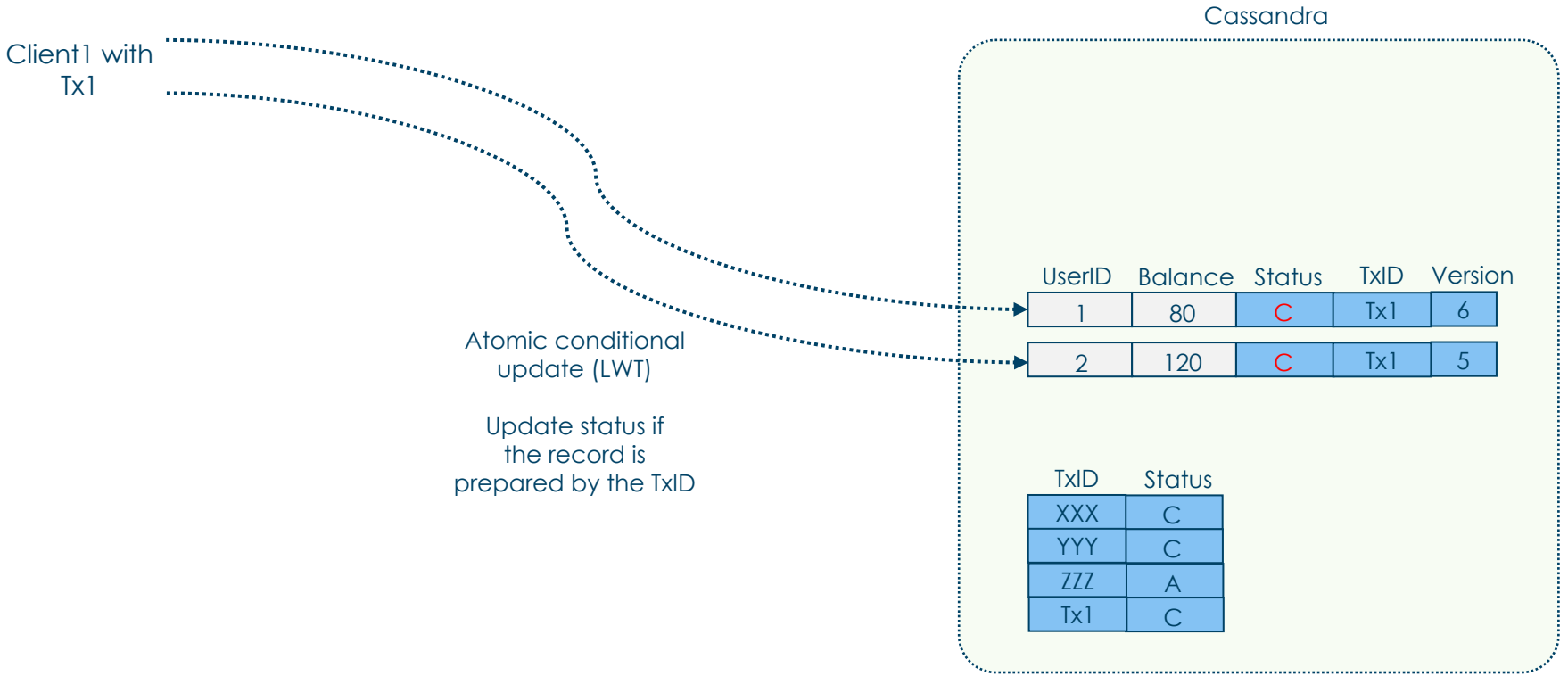
# Transaction Protocol By Examples – Prepare Phase



# Transaction Protocol By Examples – Commit Phase 1



# Transaction Protocol By Examples – Commit Phase 2

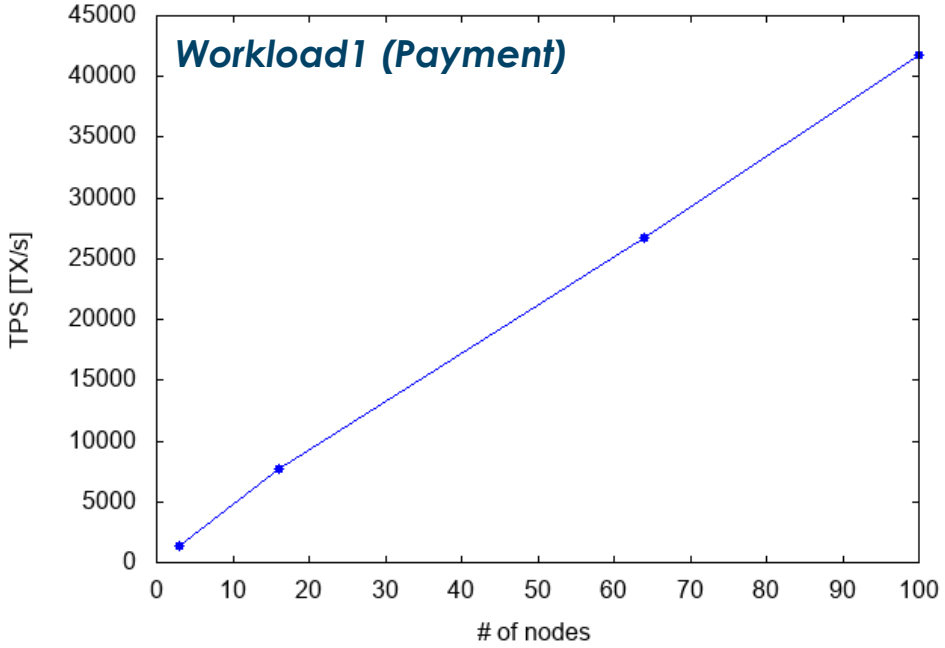


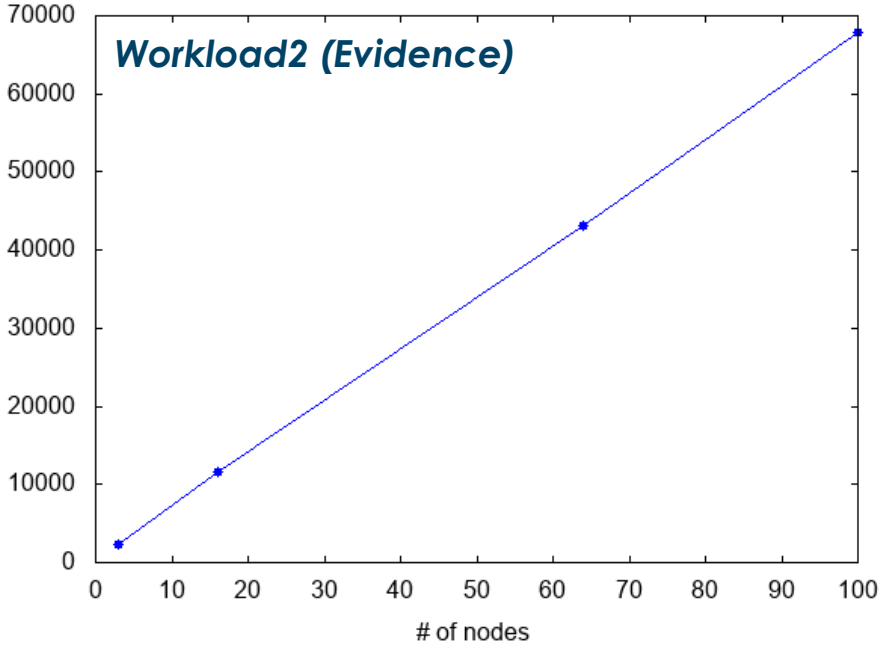
# Failure Handling by Examples

- If TX1 fails before prepare phase
  - Just clear the memory space for TX1
- If TX1 fails after prepare phase and before commit phase 1 (no status is written in Status table)
  - Another transaction (TX3) reads the records and notices that the records are prepared and there is no status for it
  - TX3 tries to abort TX1 (TX3 tries to write ABORTED to Status with TX1's TXID and rolls back the records)
  - TX1 might be on it's way to commit status, but only one can win, not both
- If TX1 fails (right) after commit phase 1
  - Another transaction (TX3) tries to commit the records (rollforward) on behalf of TX1 when TX3 reads the same records as TX1
  - TX1 might be on it's way to commit records, but only one can win, not both

# Benchmark Results

- Achieved 90 % scalability in 100-node cluster
  (Compared to the Ideal TPS based on the performance of 3-node cluster)




Each node: i3.4xlarge (16 vCPUs, 122 GB RAM, 1900 GB NVMe SSD * 2), RF: 3

# Verification Results

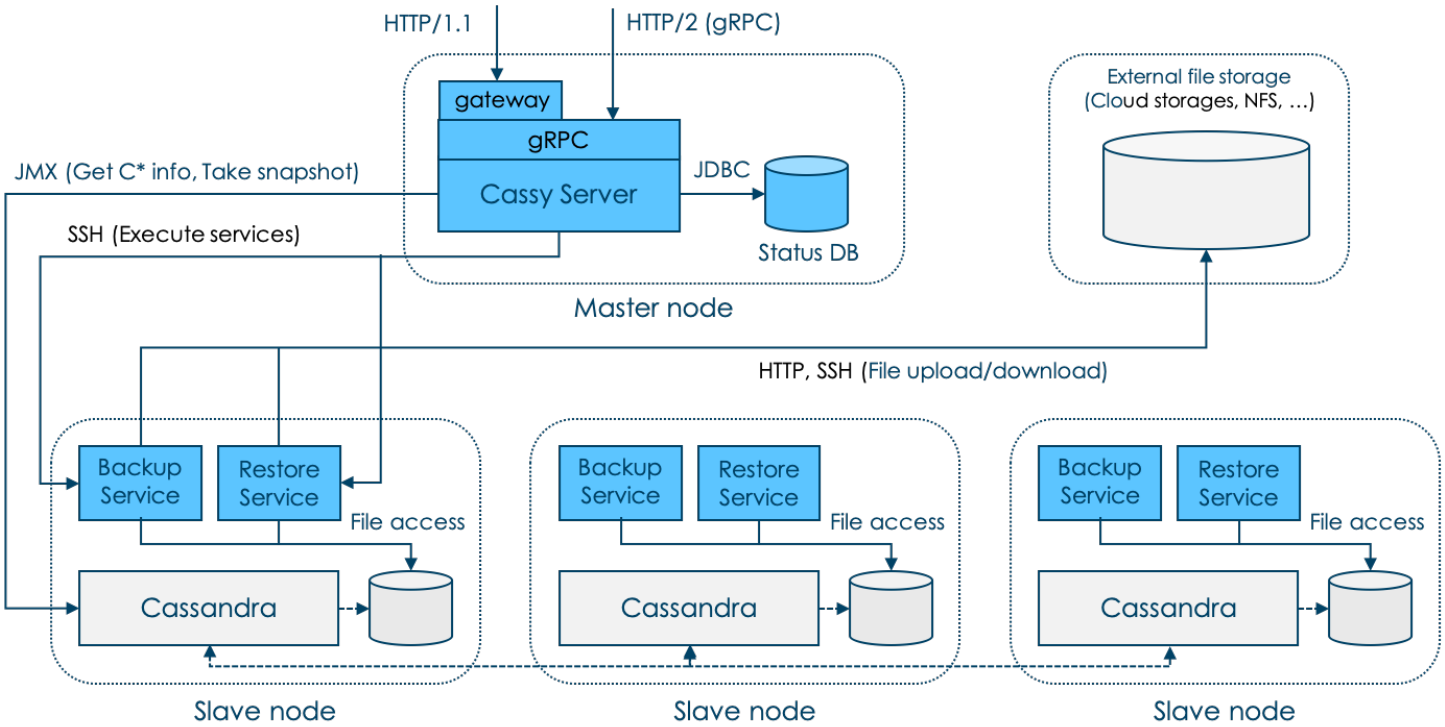
- Scalar DB has been heavily tested with Jepsen and our destructive tools
  - Note that Jepsen tests are created and conducted by Scalar
- It has passed both tests for a long time
- See https://github.com/scalar-labs/scalar-jepsen for more detail

# Other Contributions for Apache Cassandra from Scalar

- GroupCommitlogService – Yuji Ito
  - Group multiple commitlog writes at once
  - CASSANDRA-13530
- Jepsen tests for Cassandra – Yuji Ito, Craig Pastro
  - Maintain with the latest Jepsen
  - Rewrite with Alia clojure driver
  - https://github.com/scalar-labs/scalar-jepsen
- Cassy
  - A simple and integrated backup tool
  - Just released under Apache 2
  - https://github.com/scalar-labs/cassy

# Cassy: A simple and integrated backup tool

- Required to take a transactionally consistent backup



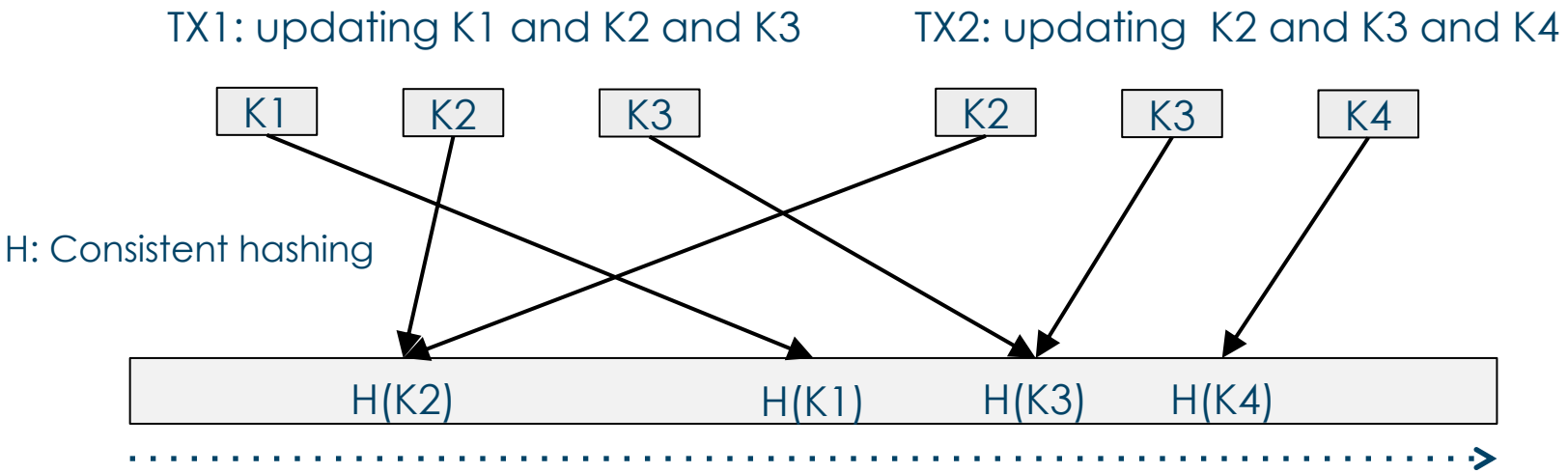
# Future Work

- DataStax driver 4.x support
- Cassandra 4.x support
  - Hopefully nothing needs to be done
- Other C* compatible databases integration
  - Scylla DB (waiting for LWT)
  - Cosmos DB (waiting for LWT)
- (HBase adapter)

# Questions ?

# Optimization

- Prepare in deterministic order
  - => First prepare always wins




Always prepare in this order !!!

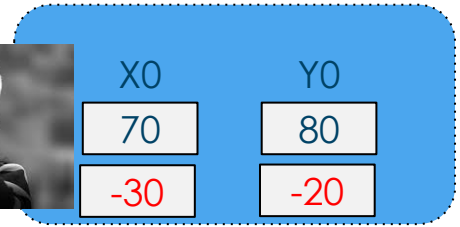
(otherwise, all Txs might abort.
Ex. TX1 prepares K1,K2 and TX2 prepares K4,K3 in this order)

# Snapshot Isolation

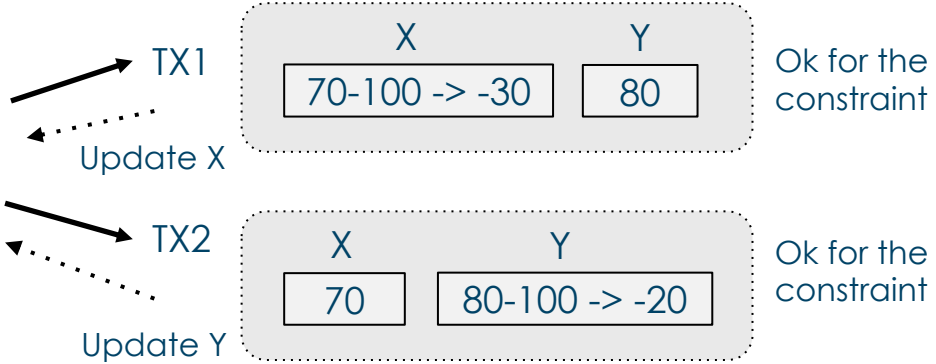
- Strong isolation level but weaker than Serializable
  - Similar to “MVCC”
  - Oracle’s most strict isolation level (it’s called “Serializable”)
- Read only sees a snapshot (=> non blocking reads)
- Mostly strong enough but there are still some anomalies

# Anomalies in Snapshot Isolation

- Write Skew, Read-Only Transaction
- Write skew example:
  - Account balances: *X* and *Y* (assume family account)
    - Initial state: X=70, Y=80
  - Constraint: X + Y > 0
  - TX1: X = X – 100, TX2: Y = Y - 100
  - *H*: *R*1(*X*0, 70) *R*2(*X*0, 70) *R*1(*Y*0, 80) *R*2(*Y*0, 80)*W*1(*X*1, −30)*C*1 *W*2(*Y*2, −20)*C*2




Update succeeds without conflict in Snapshot Isolation



# Serializable Support

- Convert all reads into writes (writing the same value) in a transaction

# Protocol Correction

- Commit records by non-atomically
  - => NO!!!
- Someone else have already did it and have started and prepared a new transaction