Agenda

- Background
- Design
- Contributions
- Limitations
Background: why a transactional key value store?

RDBMS

- SQL Support
- Concurrency Control
- Recovery
- Relational Data Model
- Storage Engine

Transactional KV Store

- Concurrency Control
- Recovery
- Simple Key Value Model
- Storage Engine

Data Processor

Transaction

Storage
Design: Architecture

- Coordinators: Paxos
- Transaction System: Stateless
- Log System: Stateful
- Storage System: Stateful
Design: Concurrency Control

- Strict Serializability
- Backward Validation OCC
- MVCC
- Commit Version:
  - prev version
  - current version
Design: Concurrency Control

**Algorithm 1:** Check conflicts for transaction $T_x$.

Require: $lastCommit$: a map of key range → last commit version

1. for each range $\in R_r$ do
2.  ranges = $lastCommit$.intersect(range)
3.  for each $r \in$ ranges do
4.      if $lastCommit[r] > T_x$.readVersion then
5.          return abort;
6.    // commit path
7.  for each range $\in R_w$ do
8.    $lastCommit[range] = T_x$.commitVersion;
9. return commit;
Design: Log System
Design: Recovery
Design: Tricky Problem

Assign versions in Sequencer?
- Strict Serializability
- SS Aggressive Fetching

Solution:
- Increasing Commit Versions
- Lasted Commited Version (as Read Version)
Contributions

- OCC vs Two Phase Locking
- Fast Recovery vs Always On
- Optimize For Happy Case
Limitations

- Transaction Size (OCC)
- 5s MVCC Window
- No RW Transactions in Recovery
Thanks!
And Questions?