Distributed Transaction Architectures

Kyle Kingsbury

@aphyr

I Break Databases!

Jepsen.io

This work was funded by

Fauna

Previous work funded by

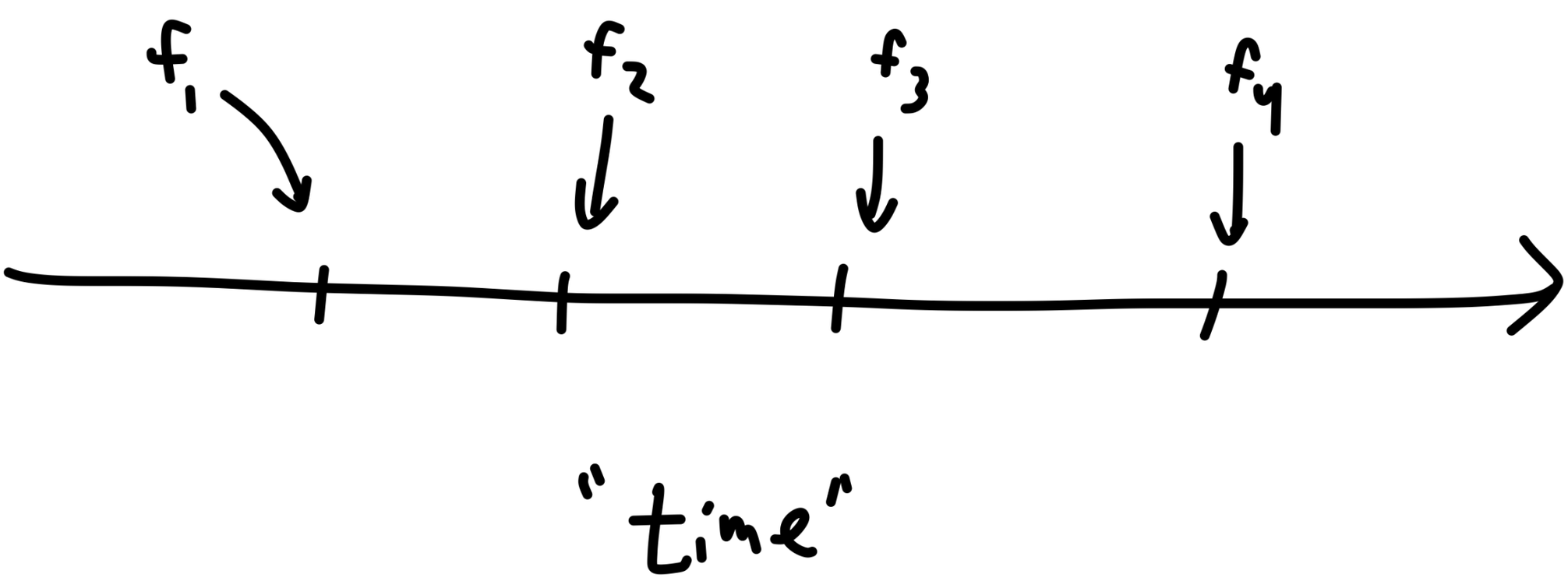
- Mongo
- Cockroach Labs
- Yugabyte
- PingCAP
- VoltDB

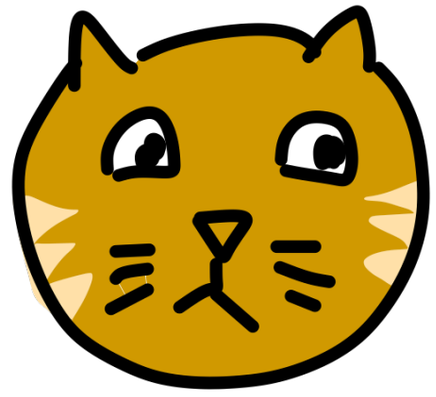
What
– do we –
Want?

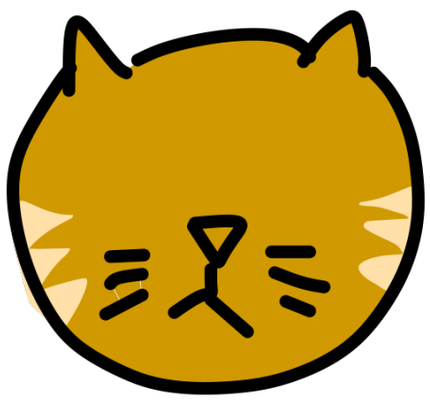
Mutation is

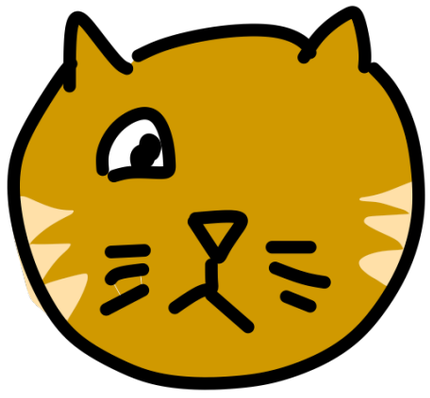
HARD

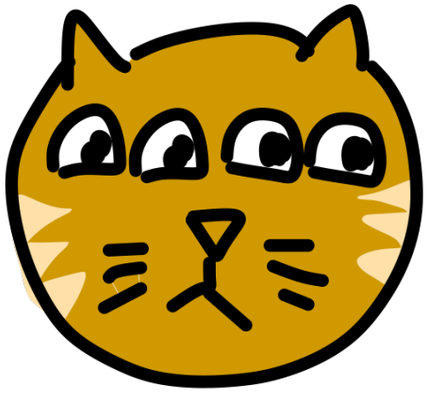
$$f(x, a) \rightarrow (x', r)$$



f( , ◠ ◠) →  ✓

 ✗

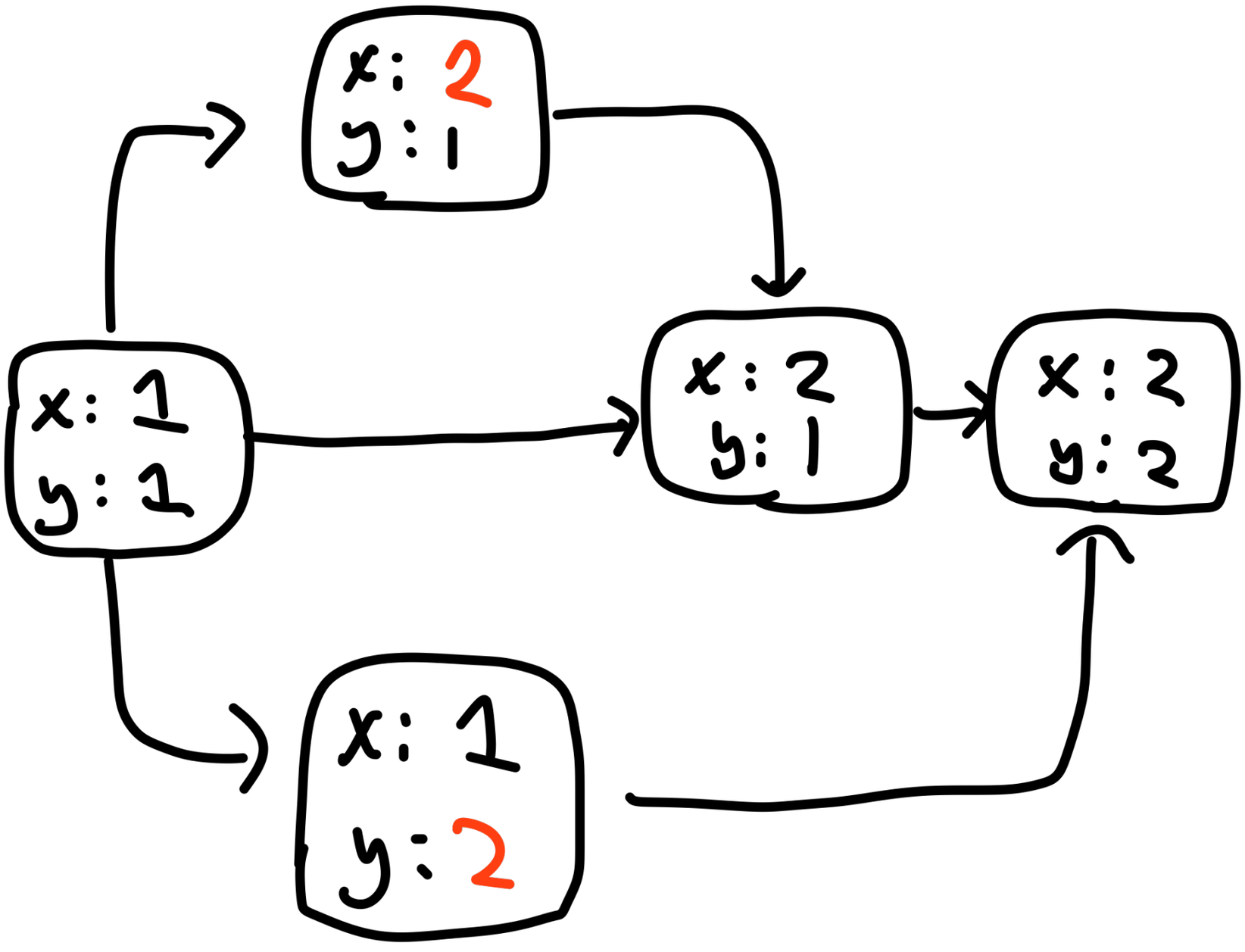
 ✗

 ✗

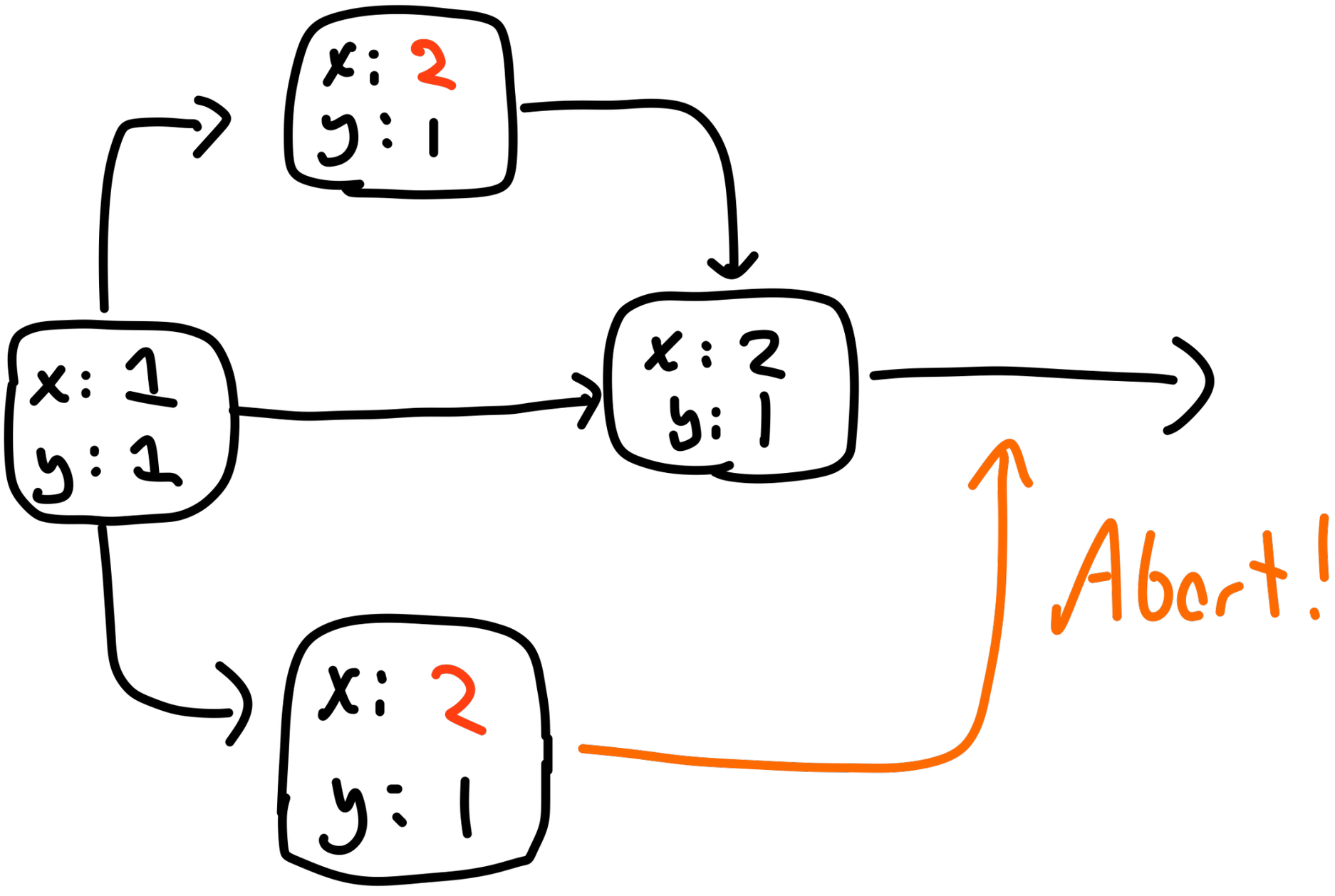
$$f(x, a) = f_3(f_2(f_1(x, a), a), a)$$

```
x = get("ms fluffles")
y = get("mx cuddlekins")

y.mice = x.mice.first

put(y)
```

Snapshot Isolation



Snapshot Isolation



Write Skew

x: 1
y: 0

"write x if both are 0"

x: 0
y: 0

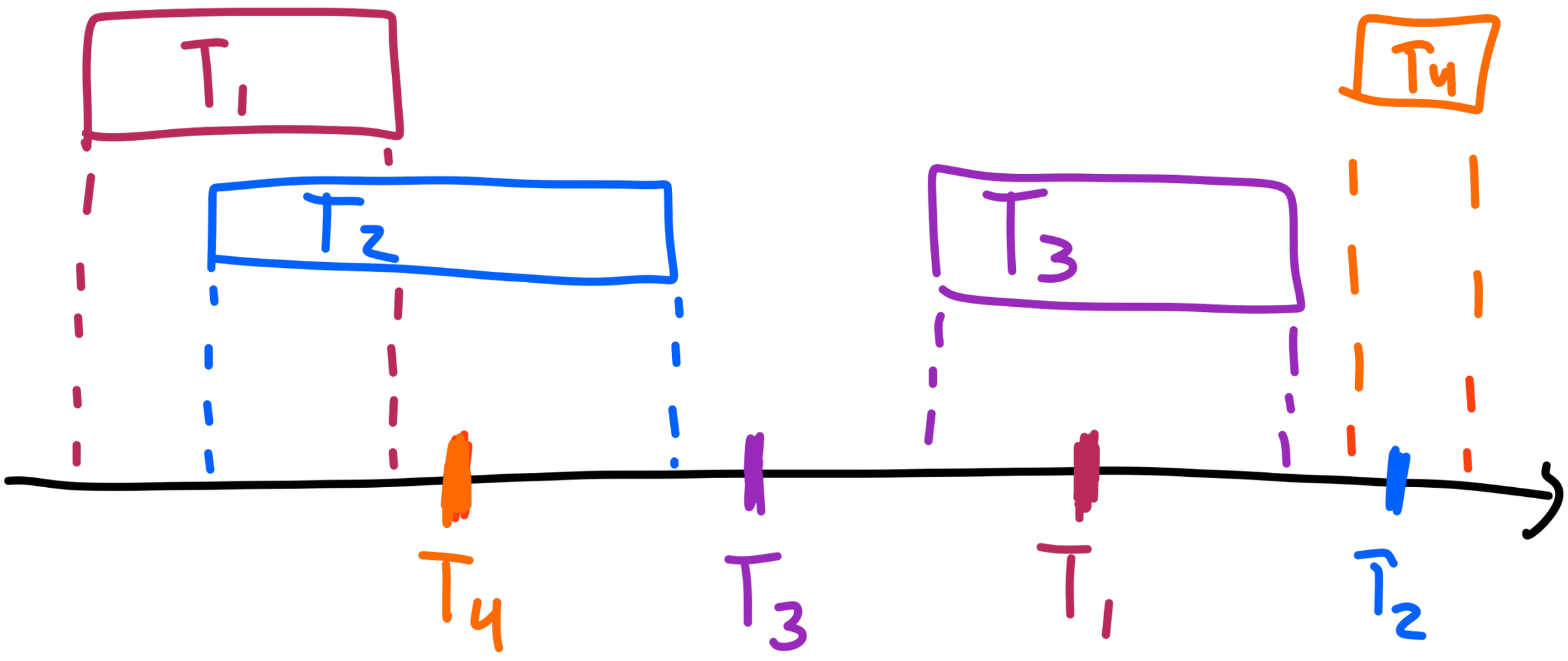
x: 1
y: 0

x: 1
y: 1

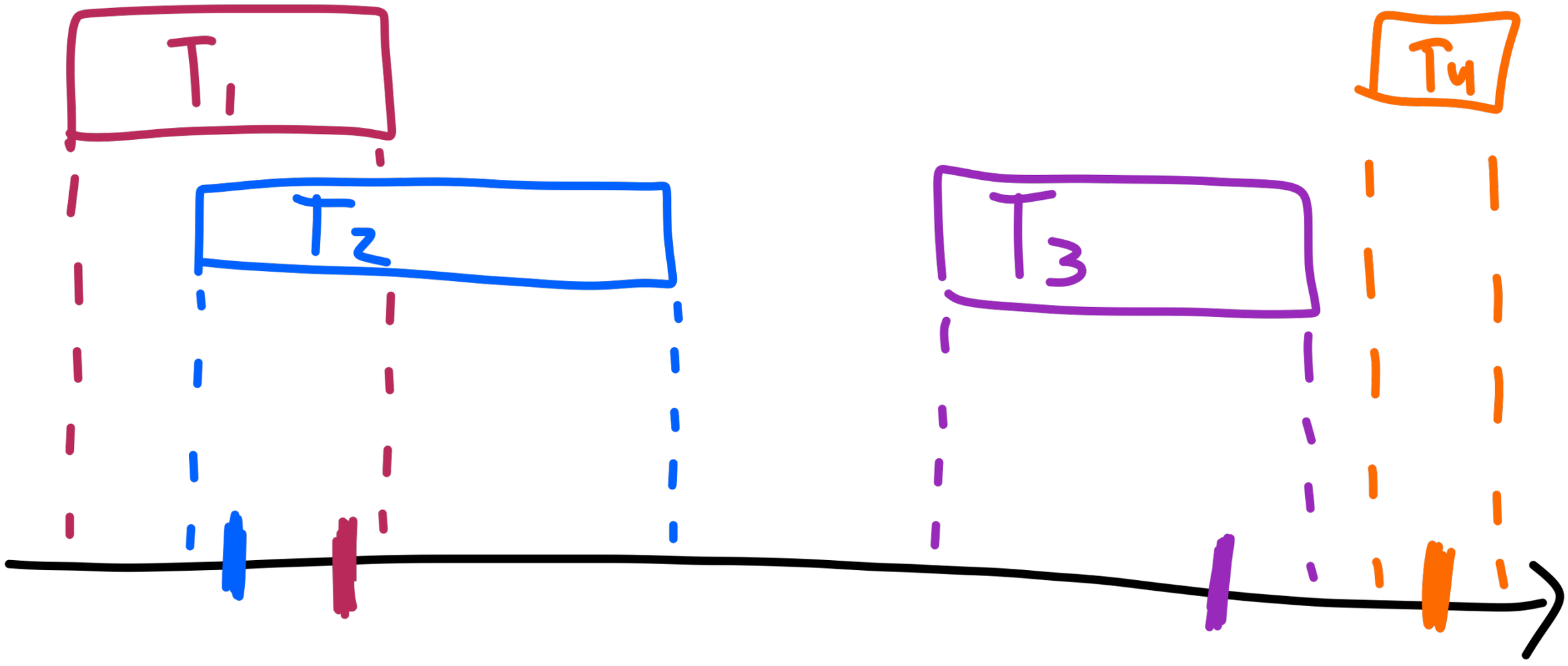
"write y if both are 0"

x: 0
y: 1

Serializable



Strict Serializable



Where

— do we —

Want it?

- Distributed
- Acrass Datacenters
- Waddwide

When

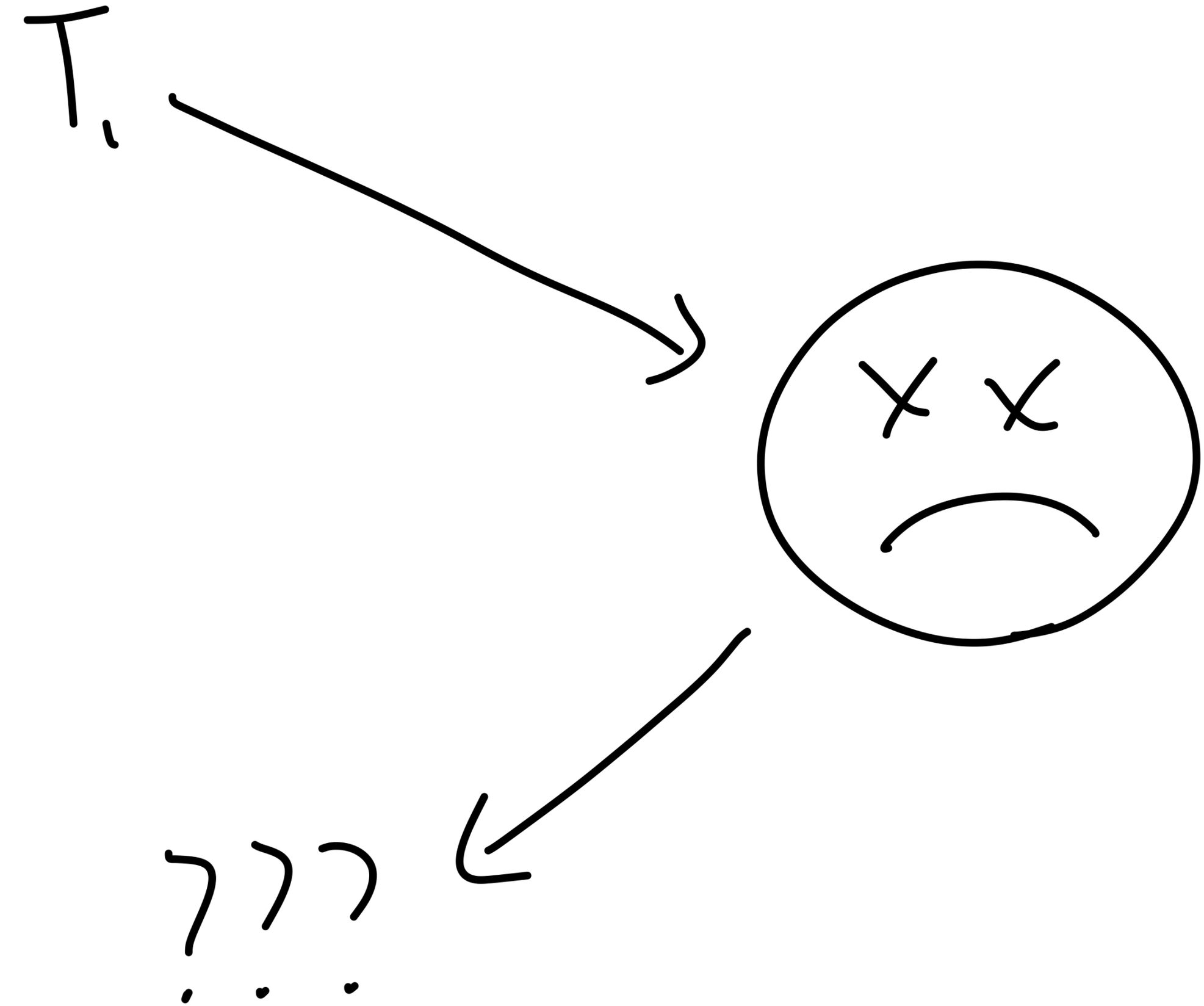
— do we —

Want It?.

Now!

Always!

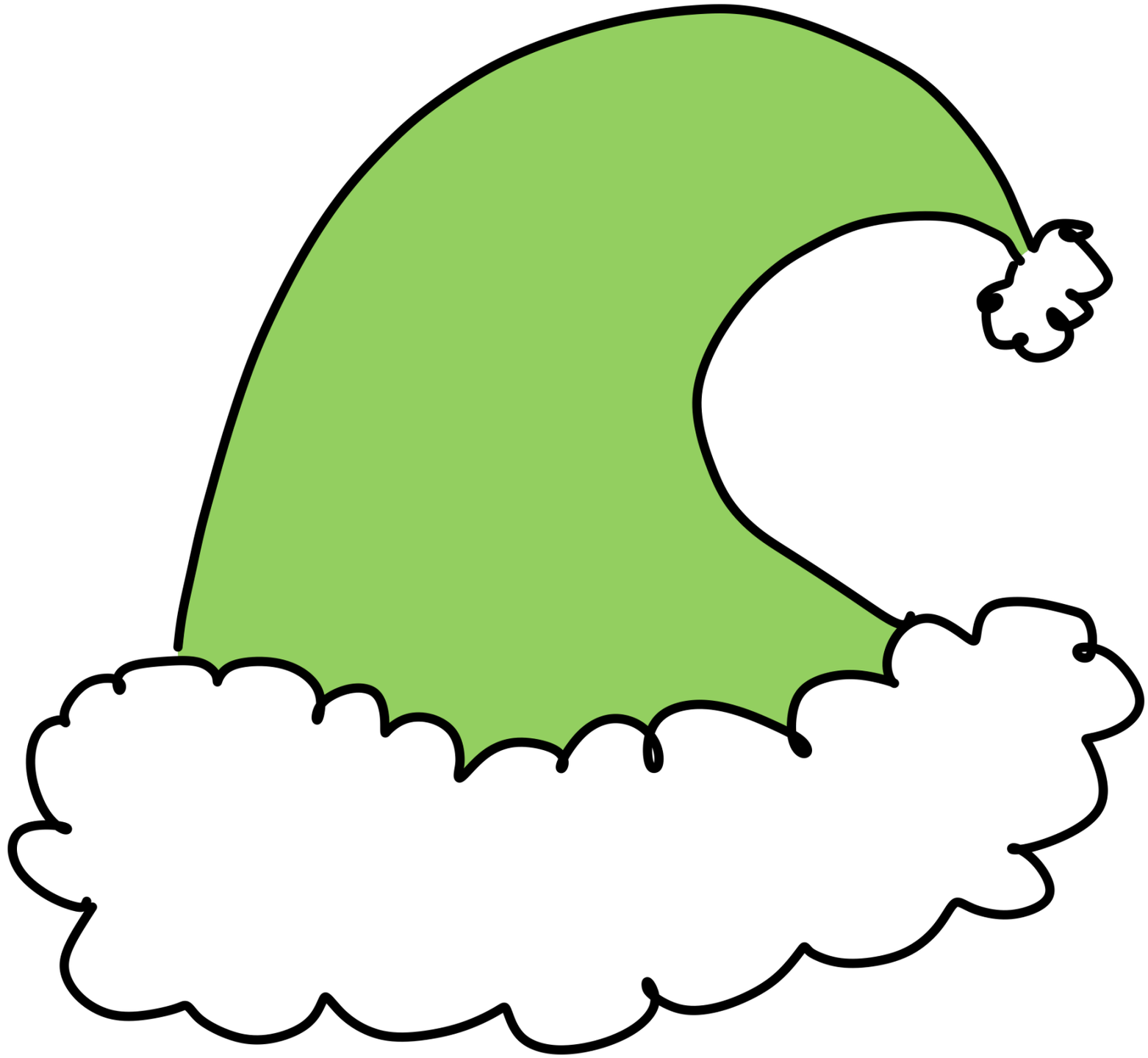
Forever!



Total Availability

"Every request to a non-faulty node succeeds"

the CAP theorem



Strict - 1SR

Serializable

Repeatable Read

Cursor stability

Read

Read

Snapshot Isolation

Monotonic Atomic View

Committed

Uncommitted

Linearizable

Sequential

Causal

WFR

PRAM

MR

MW

RYW

If we can't have total or sticky available... what CAN we have for strong consistency?

Consensus

Algorithms

- Nontriviality

- Safety

- Liveness

Lampat, 2002

- $n > 2f$

- Minimum 2 msg delays (w/ conflicting requests)

Practically Achievable

ZAB

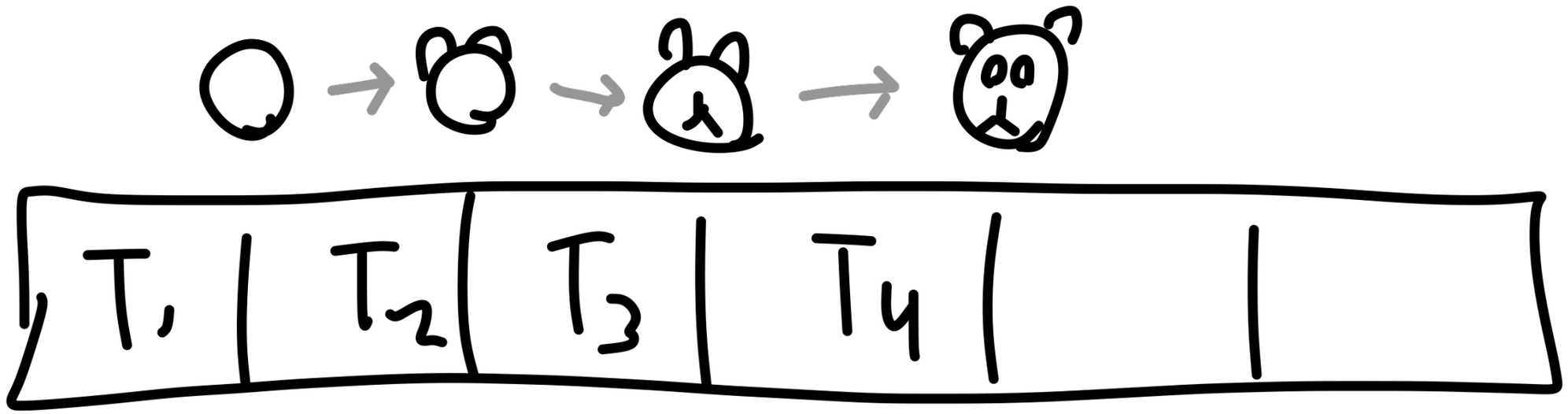
Raft

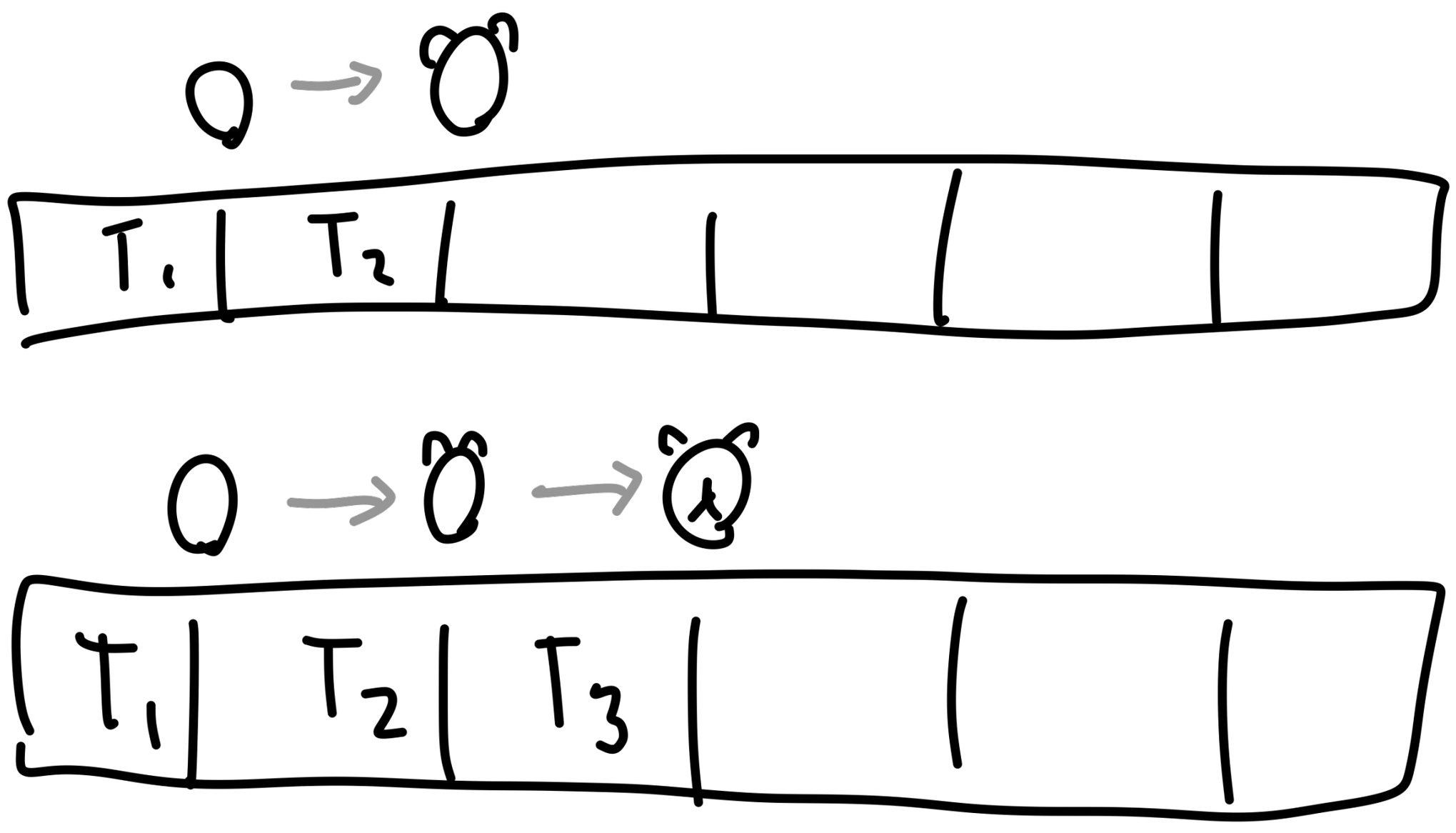
Paxos

- Model system as a state machine

- Consensus on transitions

- And their order


T1
T2
T3
T4


T1
T2
T1
T2
T3

- Linearizable in 2 hops

- Sequential reads if desired, in 0 hops

- MongoDB

- Cassandra LWT

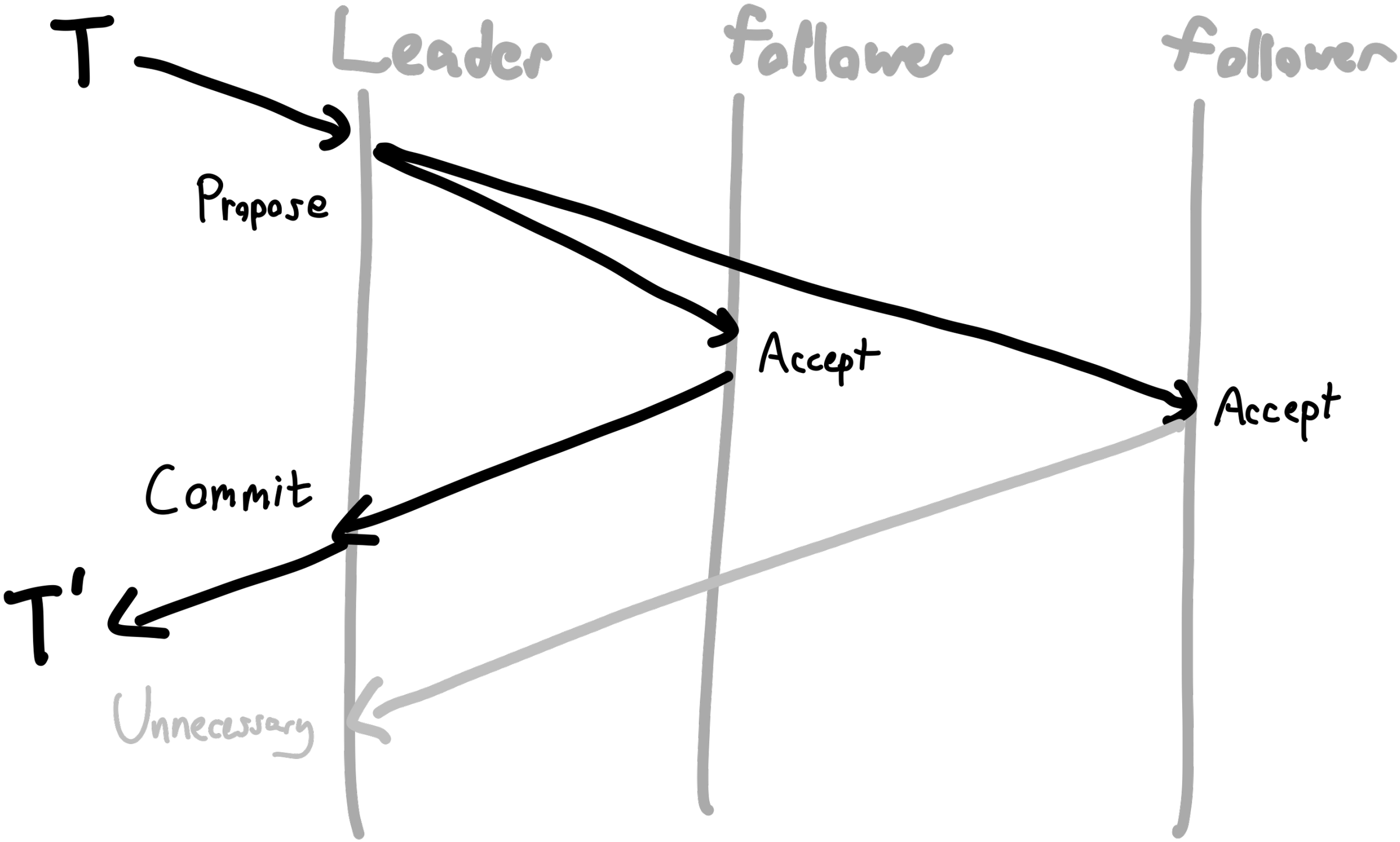
- Riak SC

- Aerospike

Majority Consensus

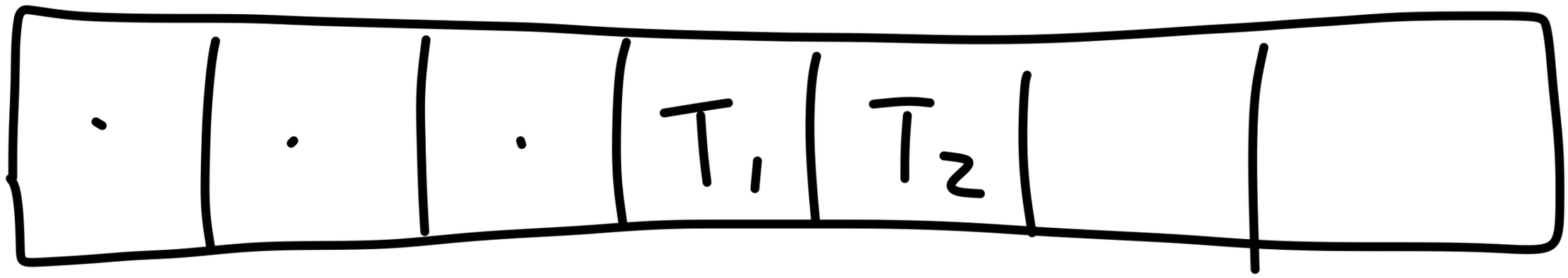
1 round trip

$\lfloor n/2 \rfloor + 1$ nodes to commit

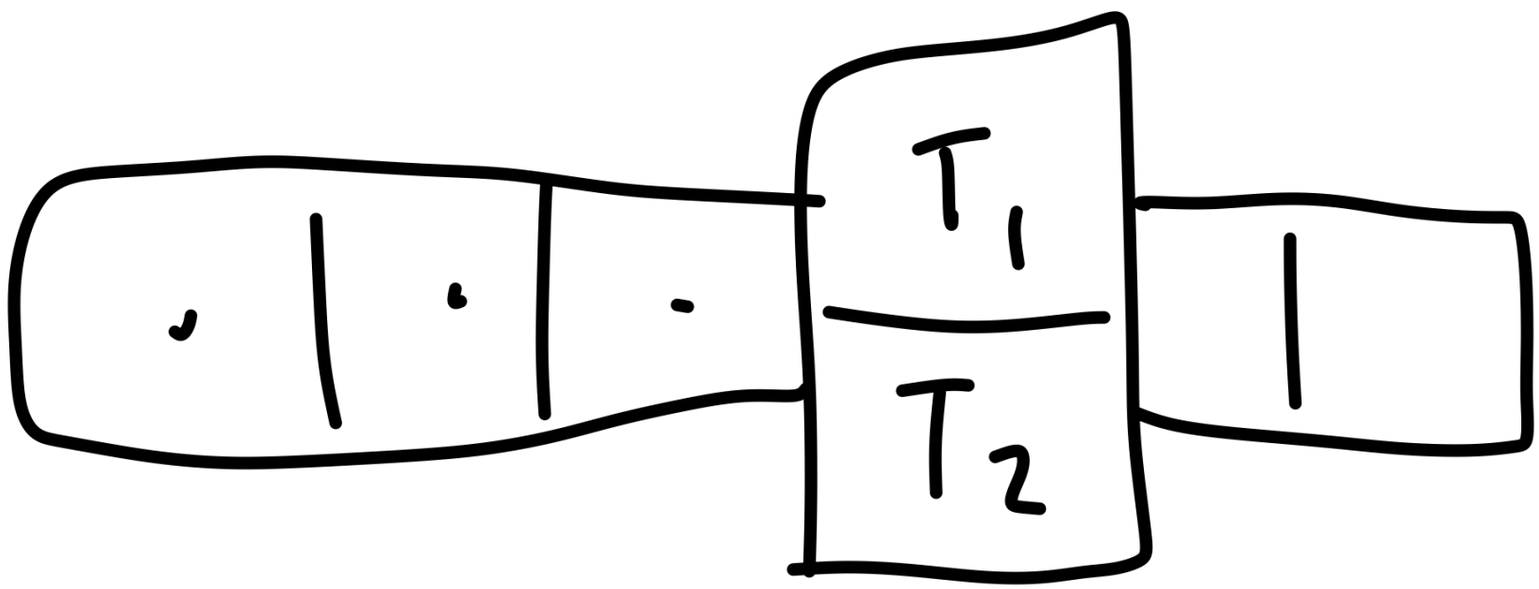

T
Leader
follower
follower
Propose
Accept
Accept
Commit
T'
Unnecessary

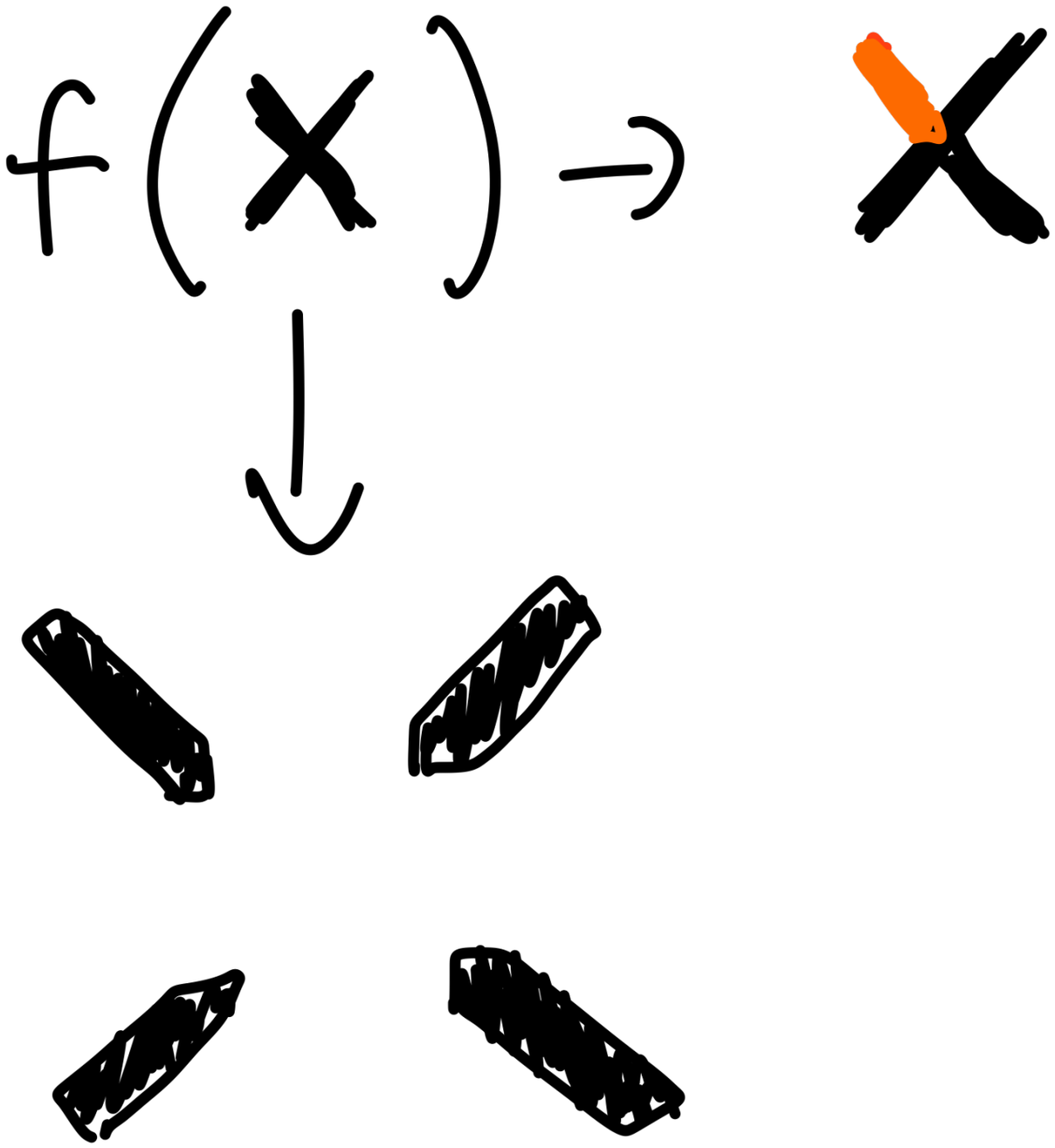
The Consensus

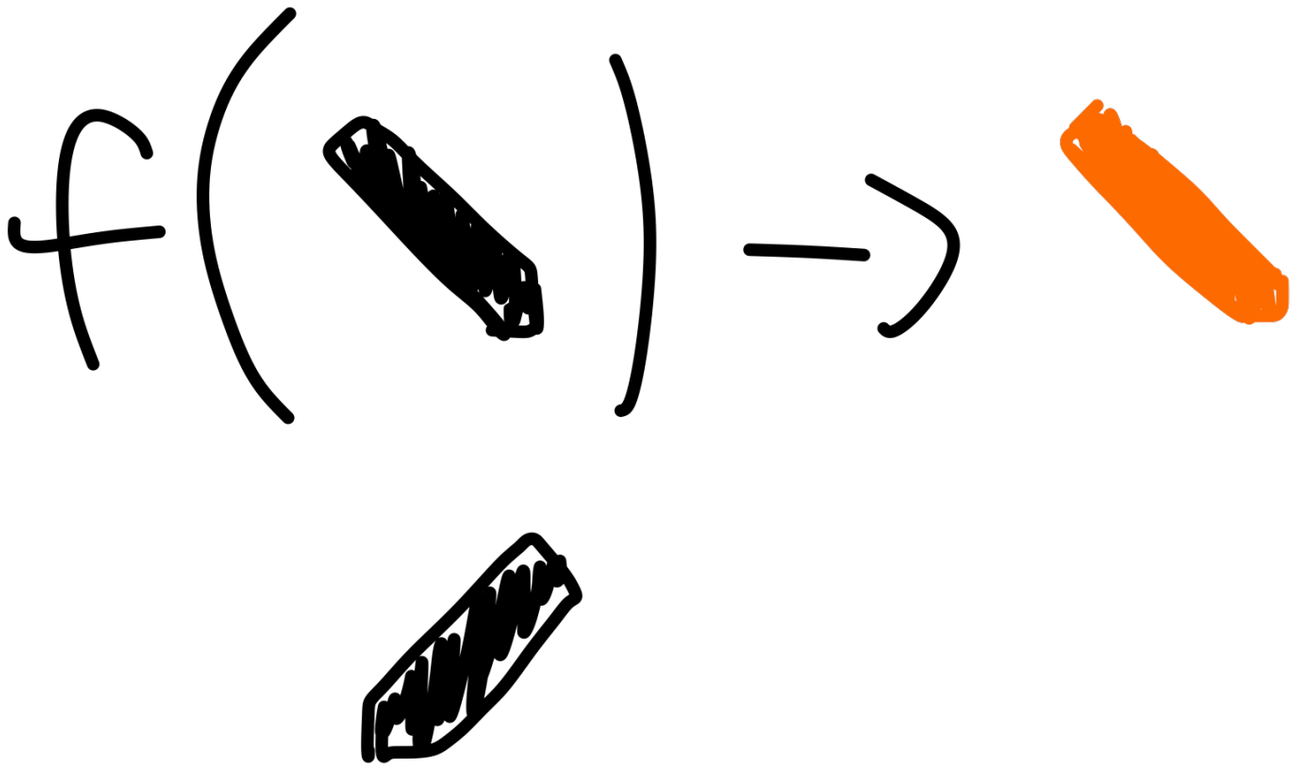
State machines are single-threaded... so our DB is single-threaded!

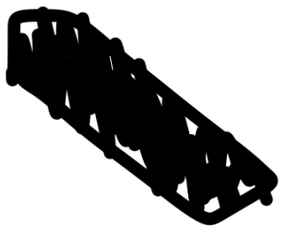


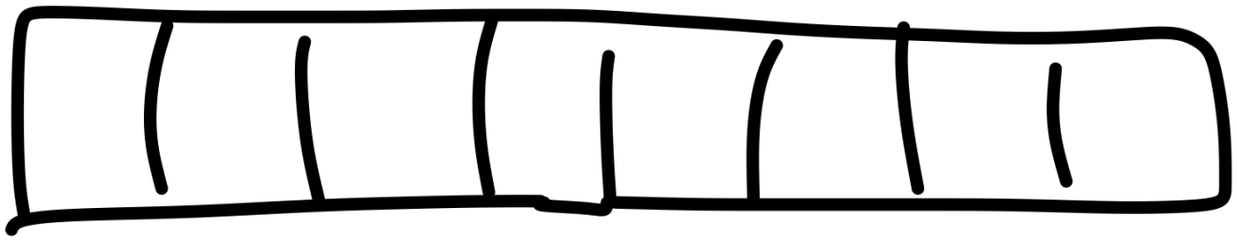
$$T_1 \circ T_2 = T_2 \circ T_1 \quad ?$$

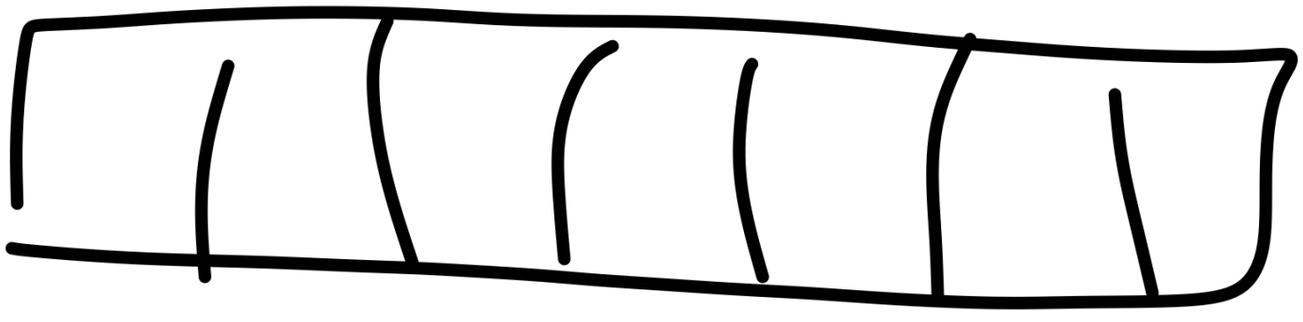


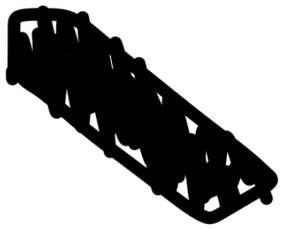


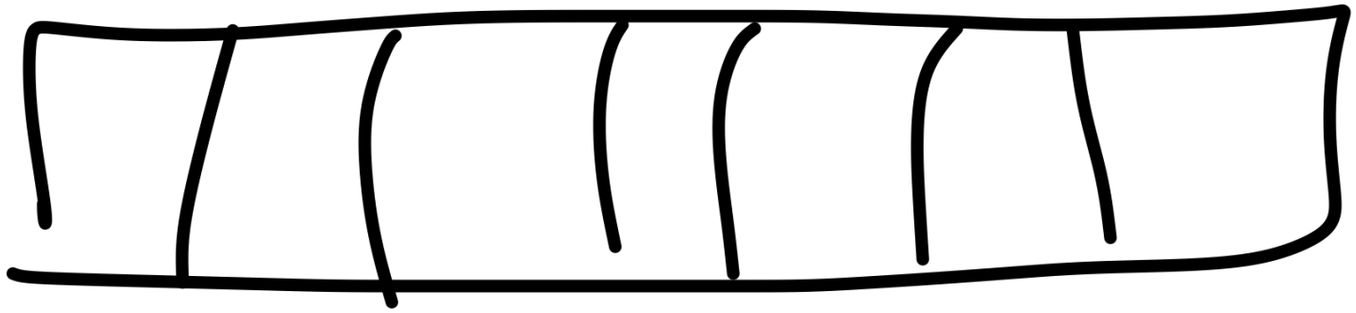


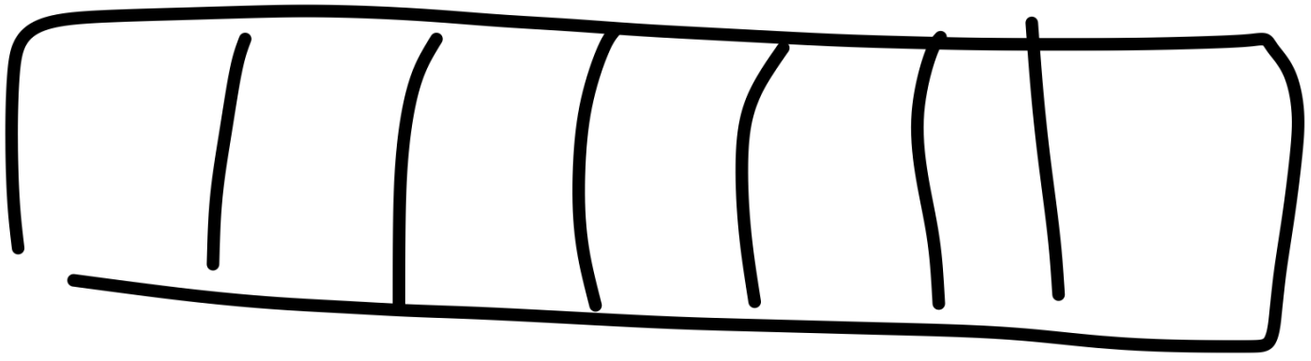








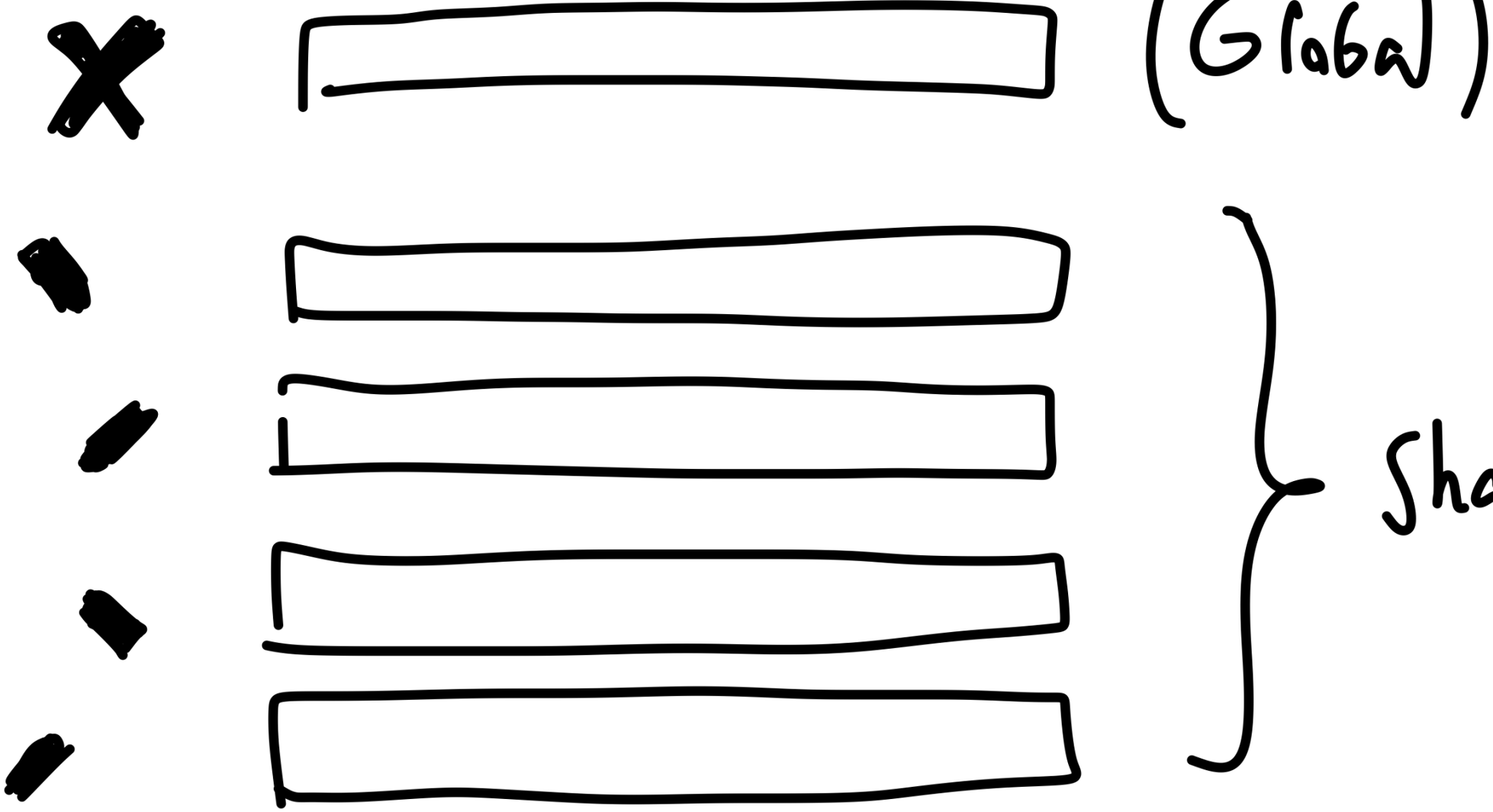




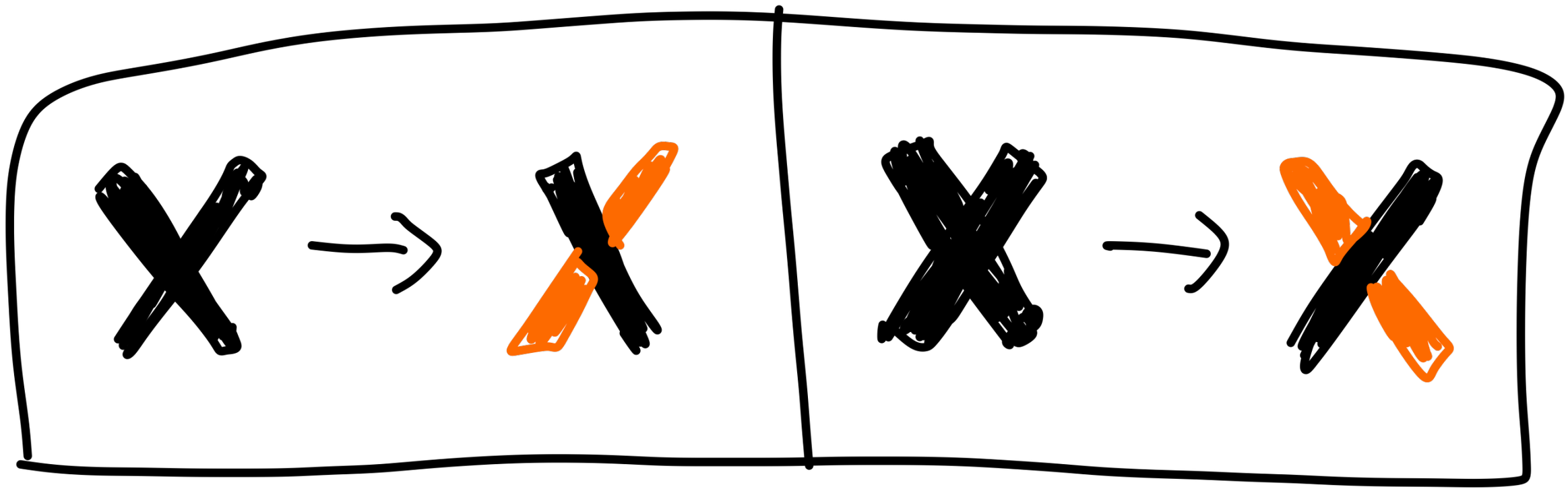
Sharding

(e.g. VoltDB)

What about cross-shard transactions?



Leaves a single point of contention for cross-shard transactions!



Let's Talk

WORST

Case

- Cross-Shard Transactions
- Strong (SI — strict-1SR)
- Globally Distributed
  - Site redundancy
  - Access everywhere

You CAN do Getter.!

- Stay inside shard
- Relax consistency
- Relax durability
- Colocate clients w/data
- Colocate shard leaders

Percolator

Google 2010

Peng & Dabek

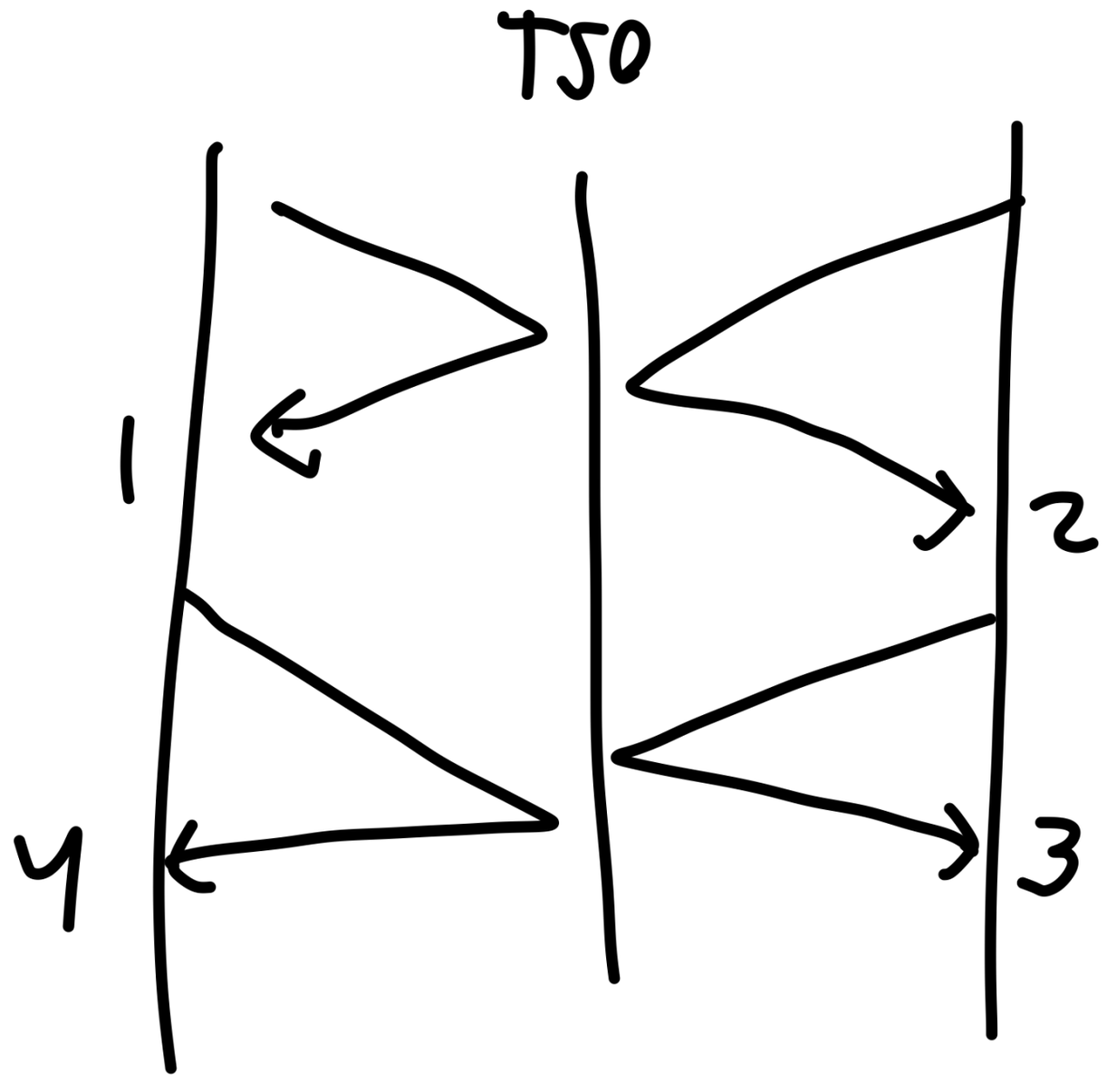
Alternative to MapReduce for txnal index building

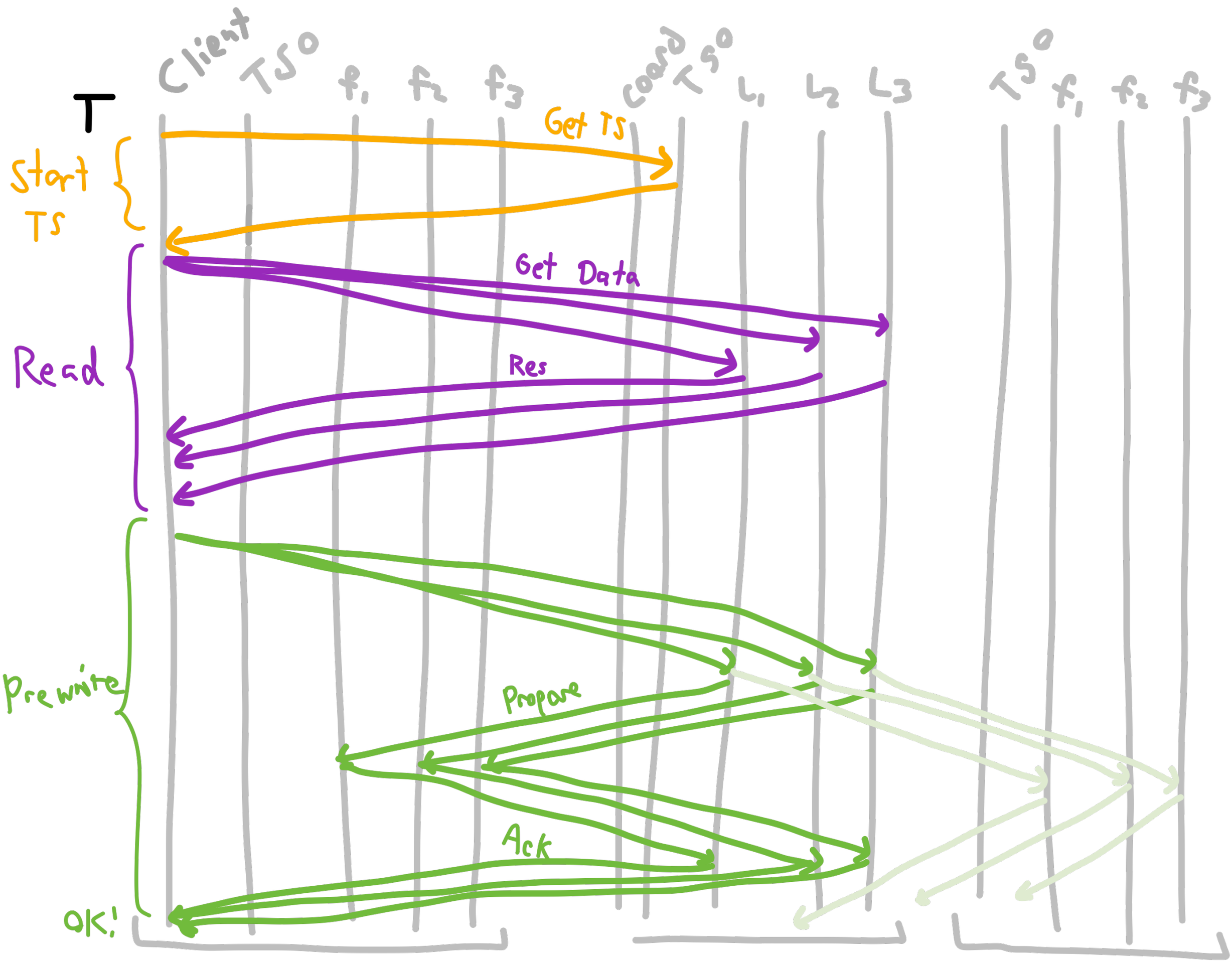
- "ACID Txns" (really SI)
- Fast read only txns
- 2PC for updates
- Relaxed latency

Built on BigTable

- k/v store
- Atomic read-modify-write
- No multi-key transactions
- Percolator is a client lib!

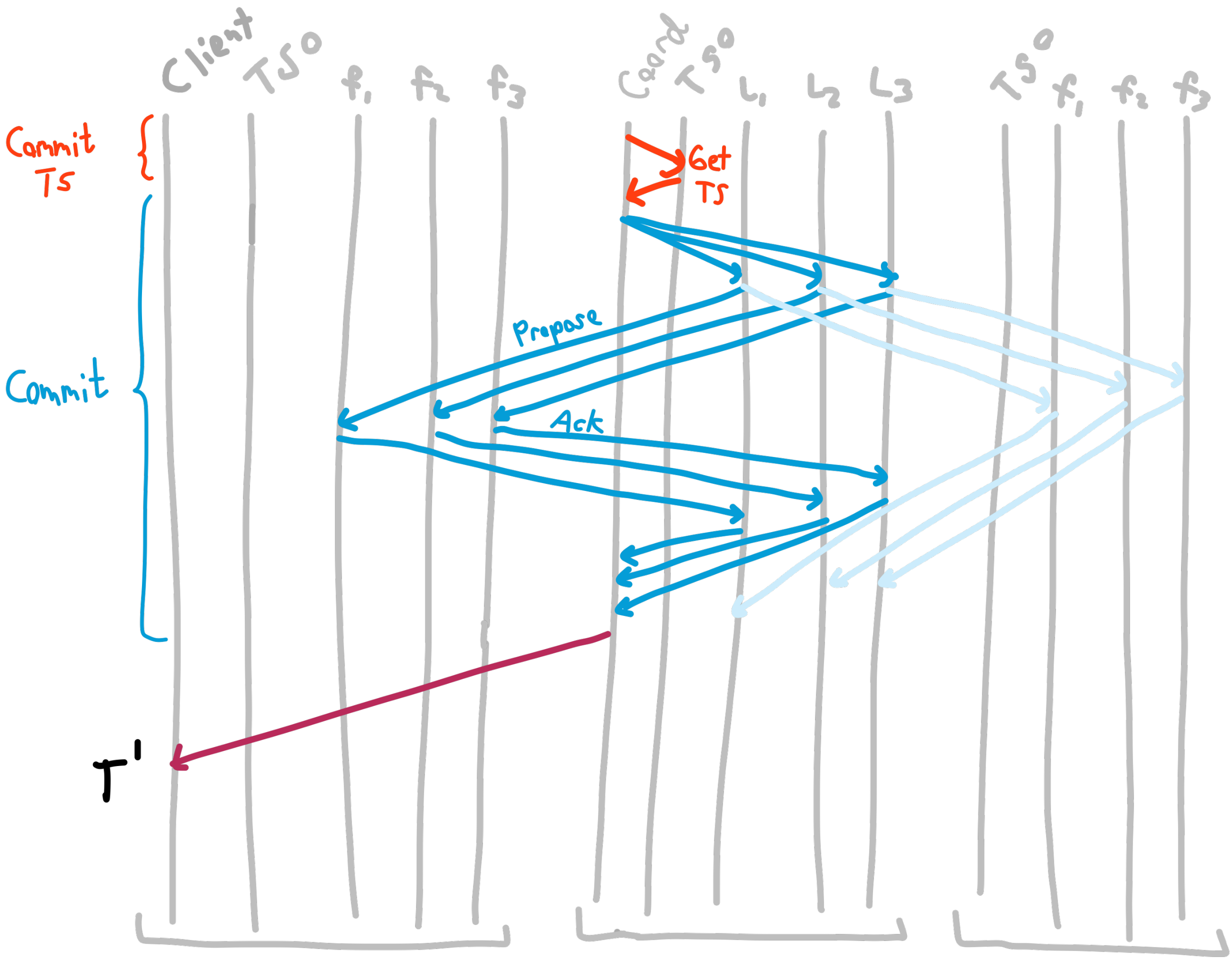
Time Stamp Oracle




Client
TSO
f1
f2
f3
coord
TSO
L1
L2
L3
TSO
f1
f2
f3
T
Get TS
Start TS
Get Data
Read
Res
Prewrite
Prepare
Ack
ok!

Client
TSO
f_1
f_2
f_3
Coord
TSO
L_1
L_2
L_3
TSO
f_1
f_2
f_3
Commit TS
Get TS
Commit Primary Record
Commit
Prepare
Ack
Done
T'
Commit
Prepare
Ack
Done
Commit Secondary Records (Async)

Client
TS^0
f_1
f_2
f_3
coord
TS^0
L_1
L_2
L_3
TS^0
f_1
f_2
f_3
T
Start
TS
Reads
TS
Read
Read
Res
TS
Reads
Prewrite
Writes
Prewrites
Prepare
Ack
OK!


Client
TSO
f_1
f_2
f_3
Coord
TSO
L_1
L_2
L_3
TSO
f_1
f_2
f_3
Commit
Ts
Commit
Get
TS
Propose
Ack
T'

R/W	Txns	
14	cross-DC	haps
0	local	haps

TiDB

- Commercial/OSS implementation
- SI K/V store + MySQL layer
- Raft for each shard
- Timestamp via Raft + leader leases

Spanner

Google 2012

- Corbett, Dean, Epstein, ...

- Globally replicated SQL-style OLTP DB

- Versioned
- Fast snapshot reads
- "External Consistency"
  (Strict-1SR?)

TSO → TrueTime

- service
- Paxos
- latency insensitive?

- library
- magic clocks
- latency sensitive?

TrueTime is Unusual

- GPS + atomic clocks
- Voting out bad nodes
- Clock sync algo
- Dedicated network & nodes
  - Instrument GC/IO pauses?

Basically 2PC over Paxos

- TS ⇒ lock intervals
- Locks taken on Paxos leaders
- Coordinator Paxos group commits
- Block to ensure TS passed
- Clean up locks async

Client
TT
f_1
f_2
f_3
TT
L_1
L_2
L_3
TT
f_1
f_2
f_3
T
TS
Reads
Read
Read Values
Writes
Choose TS
prepare
Write
Ack

Client
TT
f_1
f_2
f_3
TT
L_1
L_2
L_3
TT
f_1
f_2
f_3
Notify Coordinator of prepare TS
Choose Commit TS
Commit
Commit
Ack
Wait
Wait TS
Commit TS
Commit TS
T'
Apply Txn
Release locks

R/W Txns

8 cross-DC hops

1 local hop

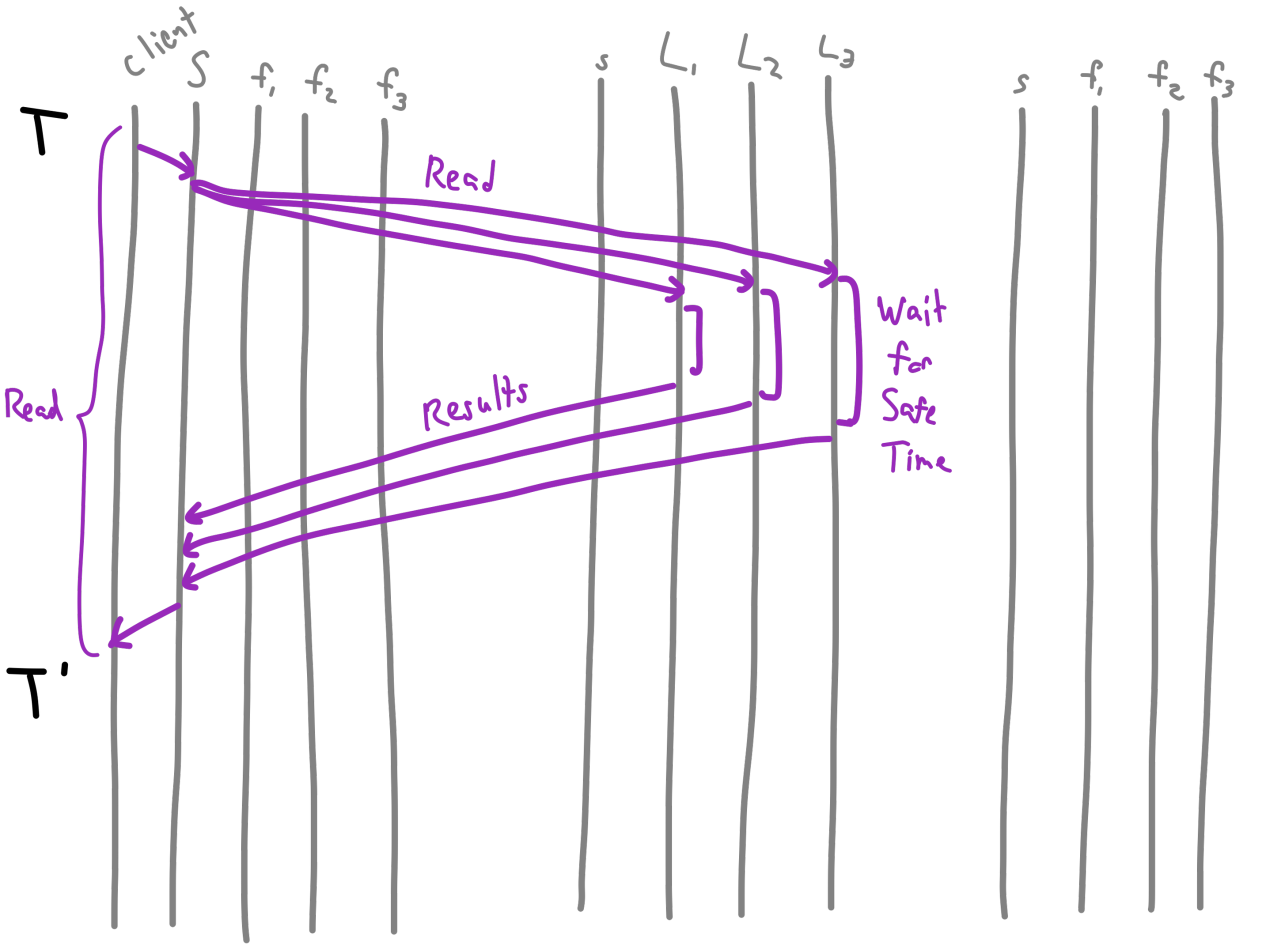
R/O Txns

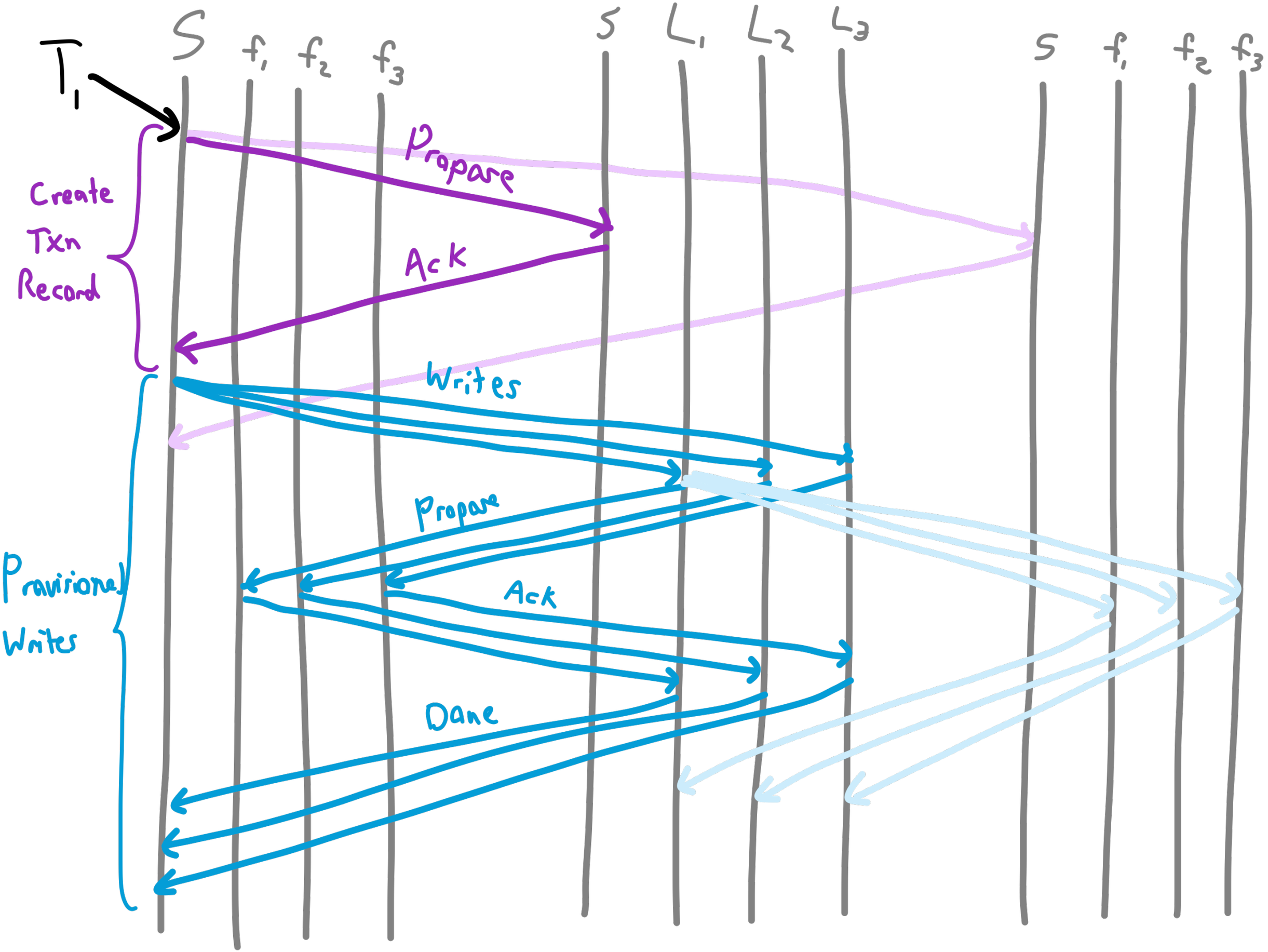
- Pick timestamp locally, read locally
- Linearizable in 0 round trips?

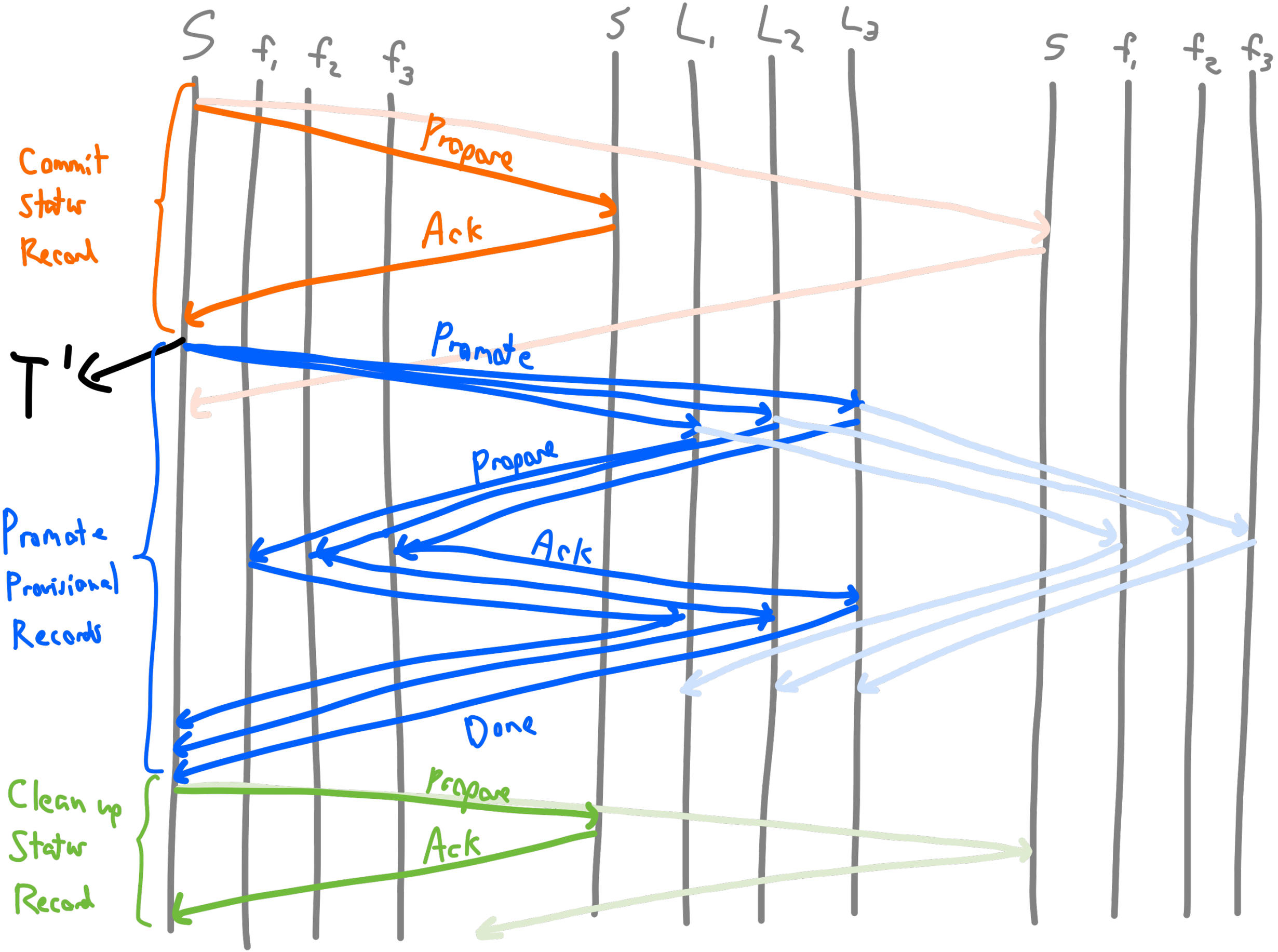
YugaByte DB

- Another commercial/oss impl
- Sort of Spanner-inspired
- HLCs coupled to Raft logs
- Leader leases
- "Safe times"
- 2PC, txn record, async cleanup

- SI CQL
  - Serializable $ SQL coming
- Linearizable single-key ops
- No general txns
  - One read
  - Batch write


client
S
f_1
f_2
f_3
s
L_1
L_2
L_3
s
f_1
f_2
f_3
T
Read
Read
Results
Wait for Safe Time
T'


S
f1
f2
f3
S
L1
L2
L3
S
f1
f2
f3
T1
Prepare
Ack
Create Txn Record
Writes
Prepare
Ack
Done
Provisional Writes


S
f1
f2
f3
S
L1
L2
L3
S
f1
f2
f3
Commit
Status
Record
Prepare
Ack
T'
Promote
Prepare
Ack
Promote
Provisional
Records
Done
Clean up
Status
Record
Prepare
Ack

R/W Transactions

2+8 cross-DC haps

2+2 local haps

Cockroach DB

- Commercial/OSS Spanner
- HLCs
- Serializable (lin. on keys?)
- Raft per shard
  - Leader lease

- No TrueTime
- Relies on your clocks!
  - And your network
- 500 ms clock skew tolerance

Spanner blocks ~> ms

on every write

CockroachDB blocks contended

reads up to 500 ms

1 PC ???

T

Client

f_1 f_2 f_3

GW

L_1 L_2 L_3

f_1 f_2 f_3

Read

Write

T'

?

6 cross-DC haps

4 local haps

Calvin

Yale 2012

Ren, Shao, Abadi

Globally distributed

serializable transactions

Key Insights

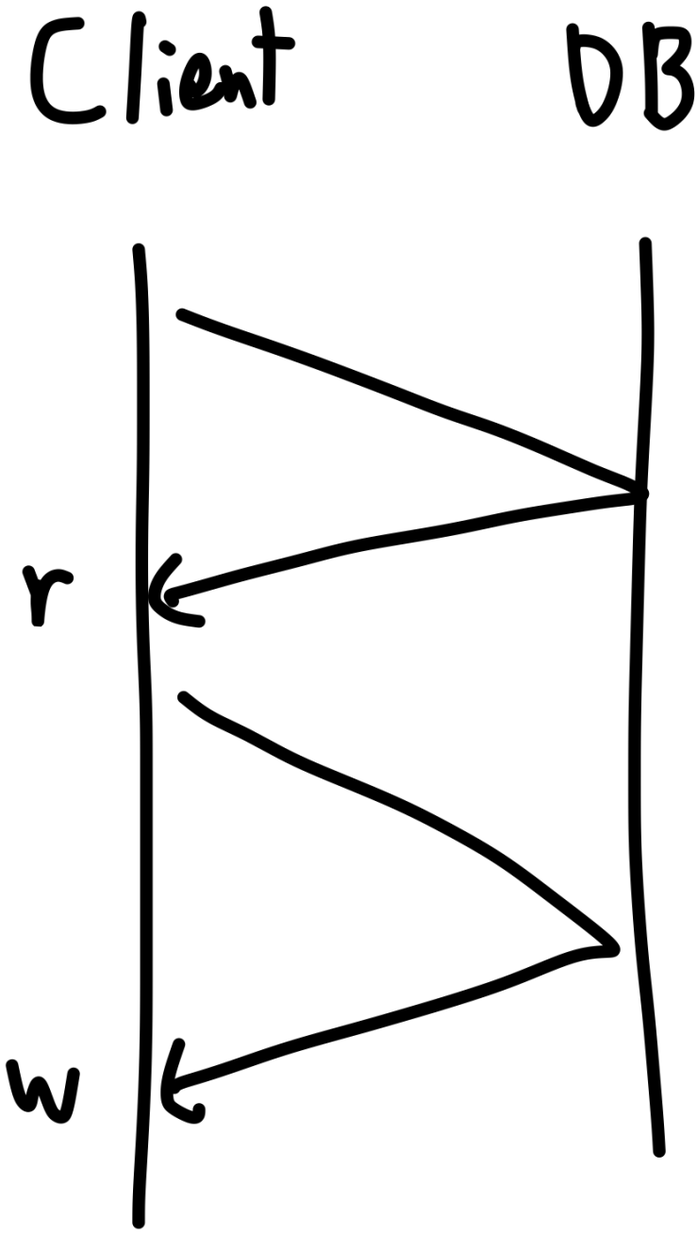
- Locks/OCC → Predetermined order

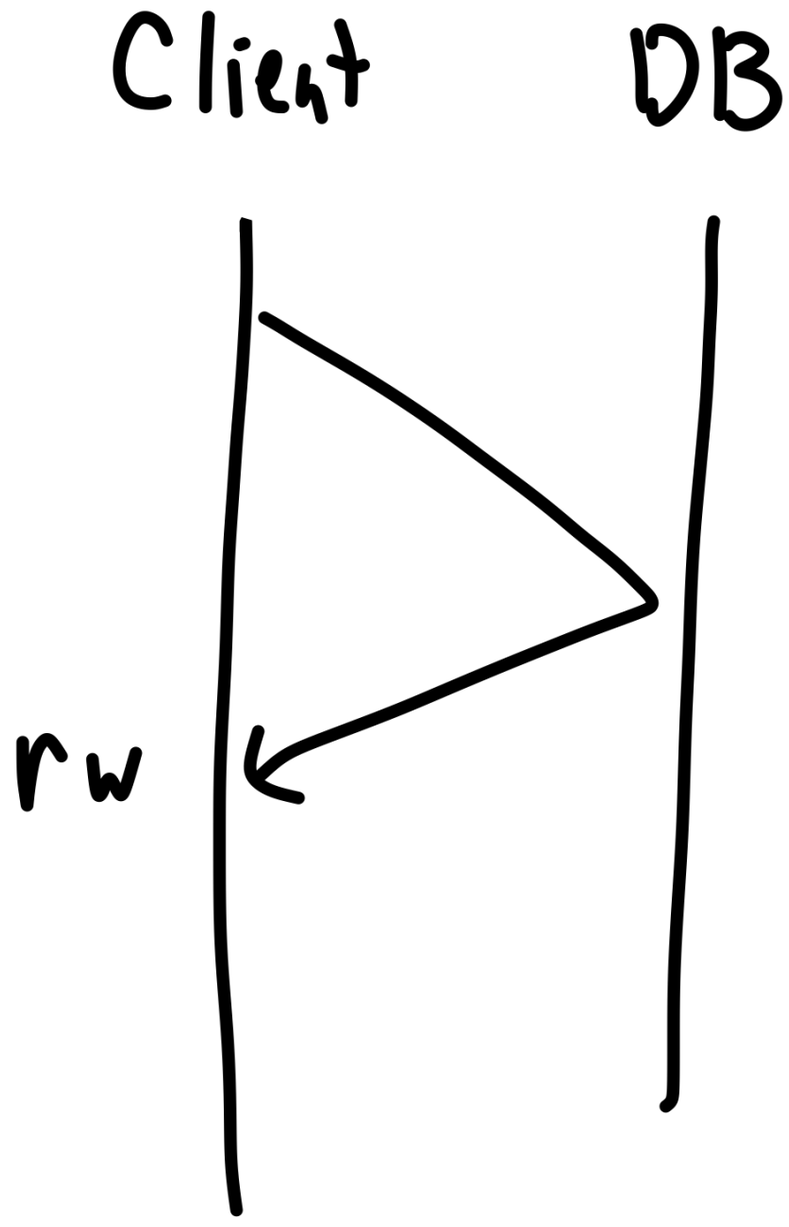
- Txns must be pure

- Txns known up front

- No interactive sessions!

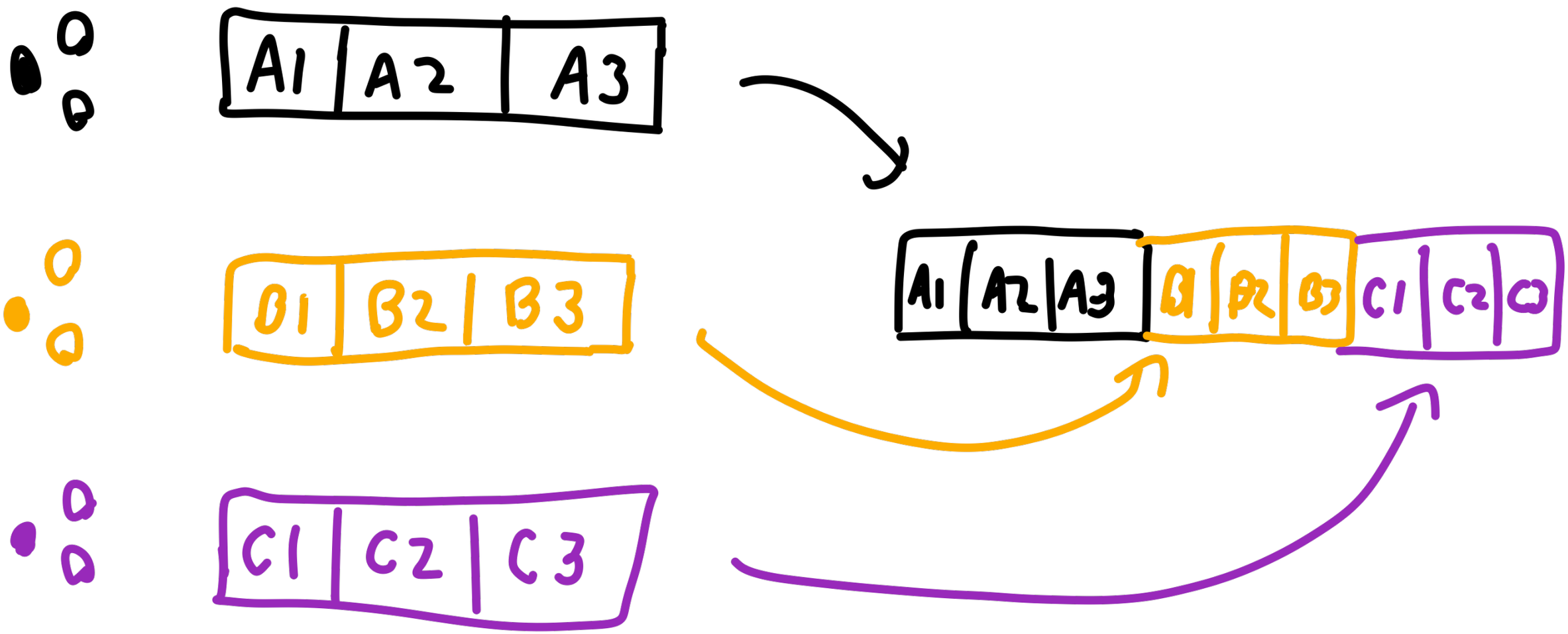
Key Insights





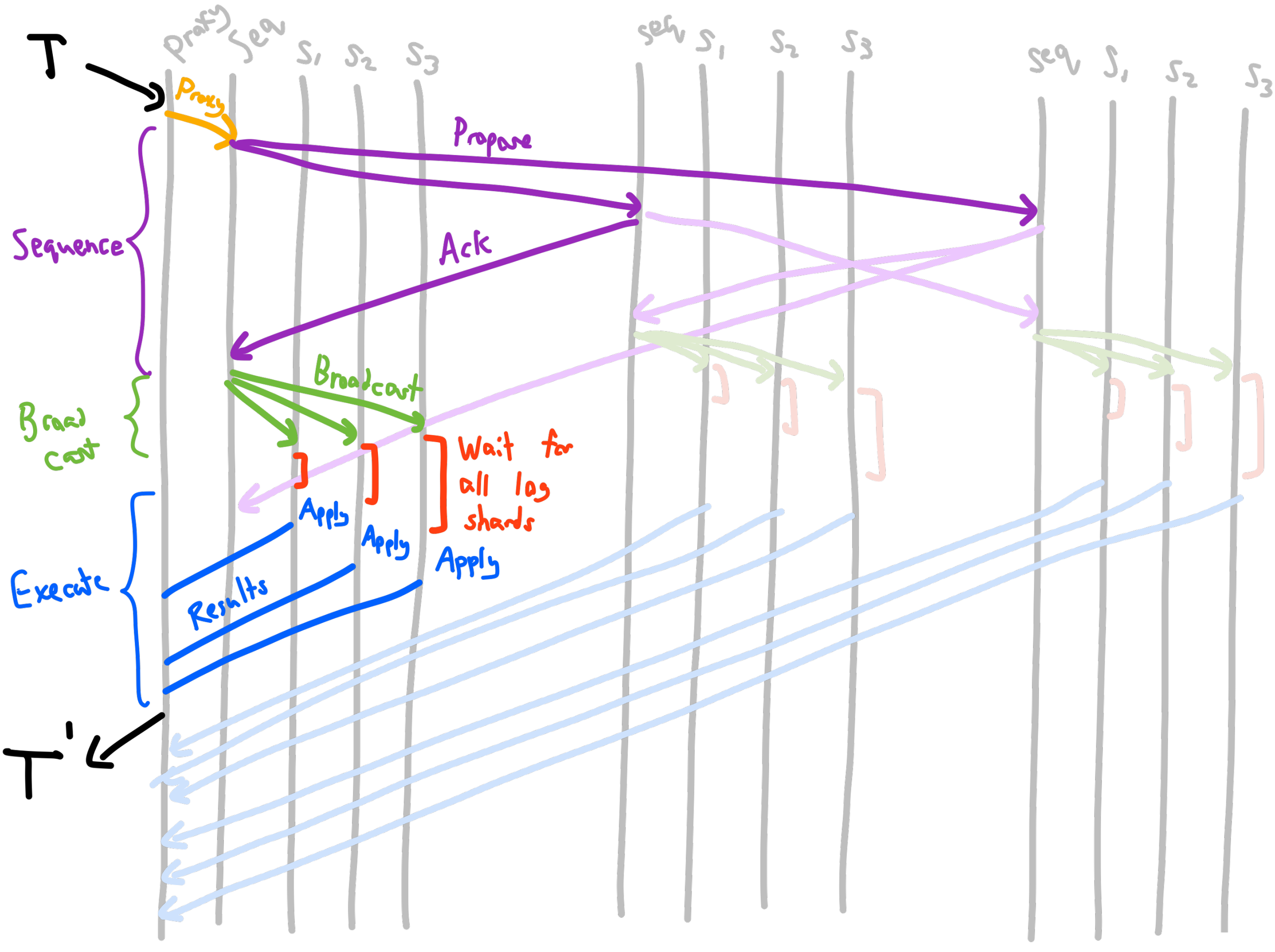
Key Insights

- You don't NEED Paxos for order...



epoch

A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9


T
Proxy
Seq
S1
S2
S3
Seq
S1
S2
S3
Seq
S1
S2
S3
Proxy
Prepare
Sequence
Ack
Broadcast
Broadcast
Wait for all log shards
Apply
Apply
Apply
Execute
Results
T'

2 cross-DC hops

5 local hops

No clock dependency
(for safety)

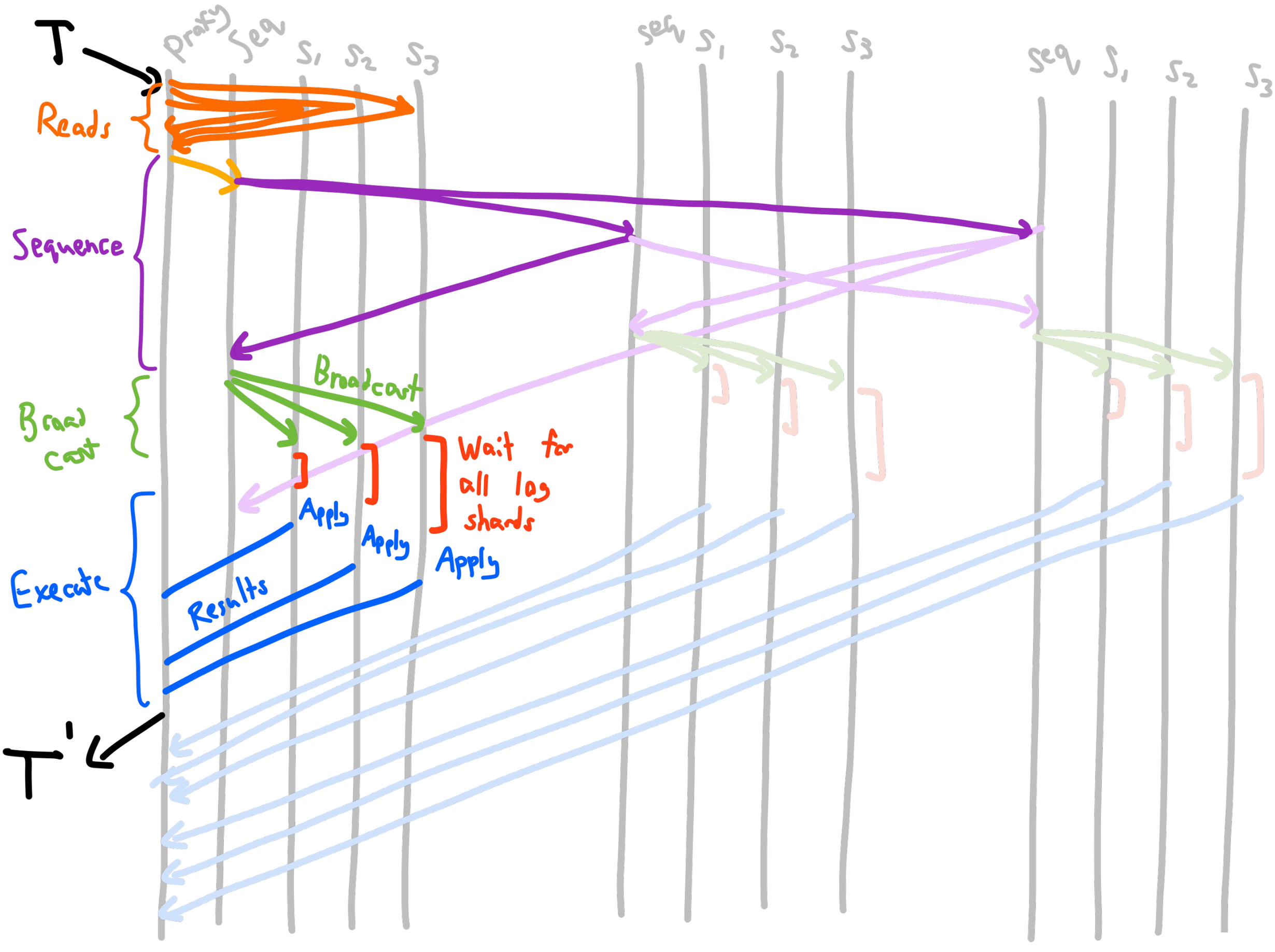
Need to block for epoch
to complete

No Interactive Transactions

- Some queries inexpressible
- Back off to r + CAS
- Can extend Calvin w/extra round trips to support interactive transactions?

FaunaDB

- Commercial impl of Calvin
- Raft for consensus
- SI, Serializable, Strict-1SR
- Dynamic R/W sets


T
Reads
Sequence
Broad cast
Execute
T'
Proxy
Seq
S1
S2
S3
Broadcast
Wait for all log shards
Apply
Apply
Apply
Results
seq
S1
S2
S3
seq
S1
S2
S3

2 cross-DC hops

7 local hops

to Review

Latency	Cross-DC	Local
Percolator	8-14?	10-0?
TiDB	14	0
Spanner	8	1
YugaByteDB	2+8	2+2
Cockroach DB	6	4
Calvin	2	5
Fauna DB	2	7

Linearizable Read Cost (xDC)
Percolator	2?
TiDB	2?
Spanner	0
YugaByte DB	2?
Cockroach DB	N/A?
Calvin	2

Clock Errors

- TiDB: Possibly? Not observed
- Spanner: Stale reads, ???
- Cockroach: Caused reverse, ???
- YB: Stale reads, ???
- Calvin: No safety issues

Interactive Transactions

- ✓ Percolator
- ✓ TiDB
- ✓ Spanner
- ✓ Cockroach DB
- ✗ YugaByte DB
- ✗ Calvin
- ✗ FaunaDB

Isolation	Min	Max
Percolator	SI	SI
TiDB	SI	SI
Spanner	SI (r)	Strict-1SR
Cockroach	SI (r)	Serializable+
YugaByte	SI	SI
FaunaDB	SI	Strict-1SR

Thanks

- Fauna $
- Cockroach $
- Mongo DB $
- Yugabyte $
- Volt DB $
- PingCAP $

Thanks

- Camille Fournier
- Kena (@lknz)
- Tang Lin
- Nathan VanBenschoten
- Karthik Ranganathan

Thanks

- @sergei
- Matt Freels
- Evan Weaver


Questions???