Jepsen 8

the Ocho!

Kyle Kingsbury

@aphyr

I break databases!

Public API {

} API code

Ruby {

DBs

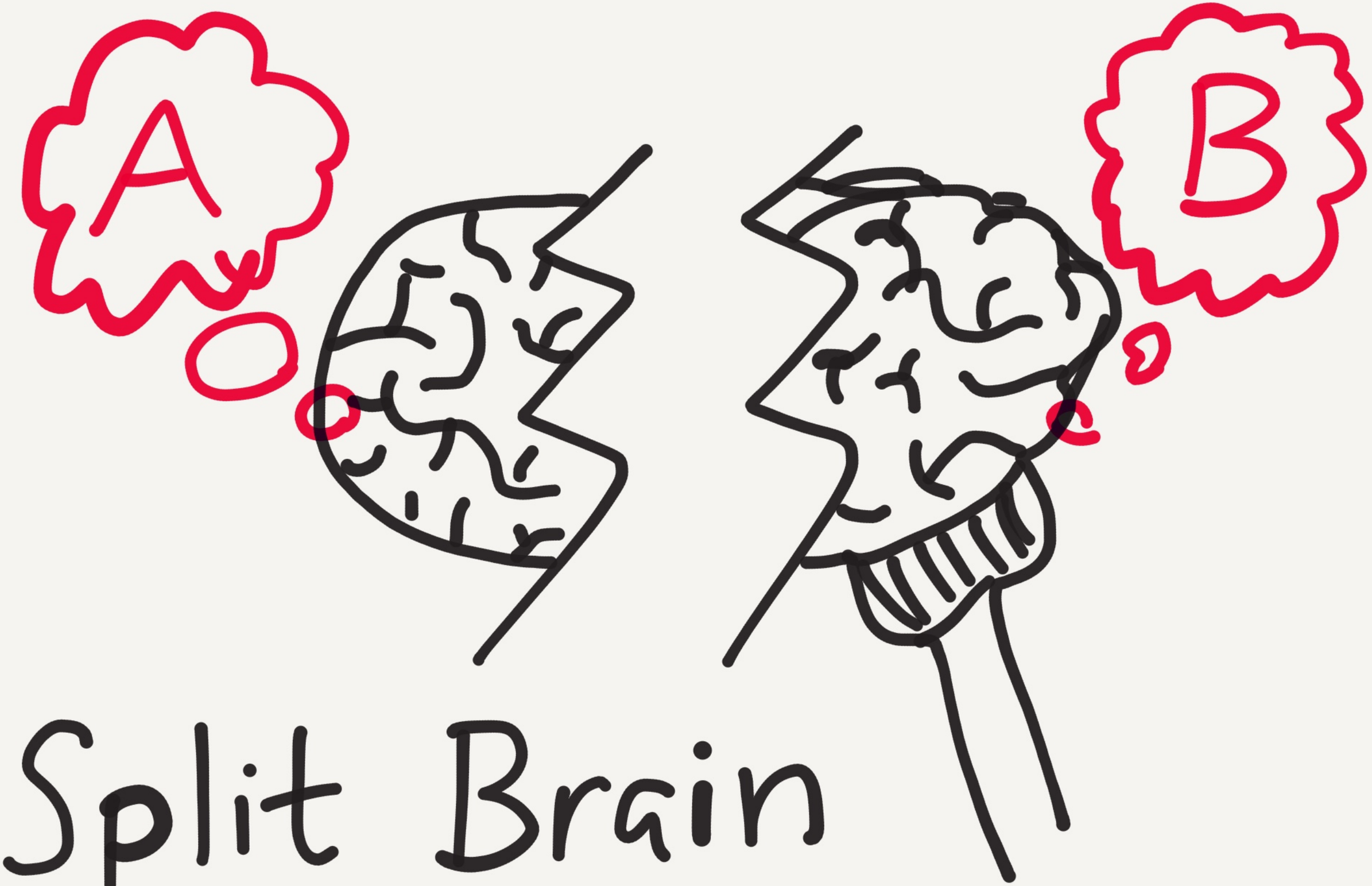
Every thing is fine!

Databases!

Queues!

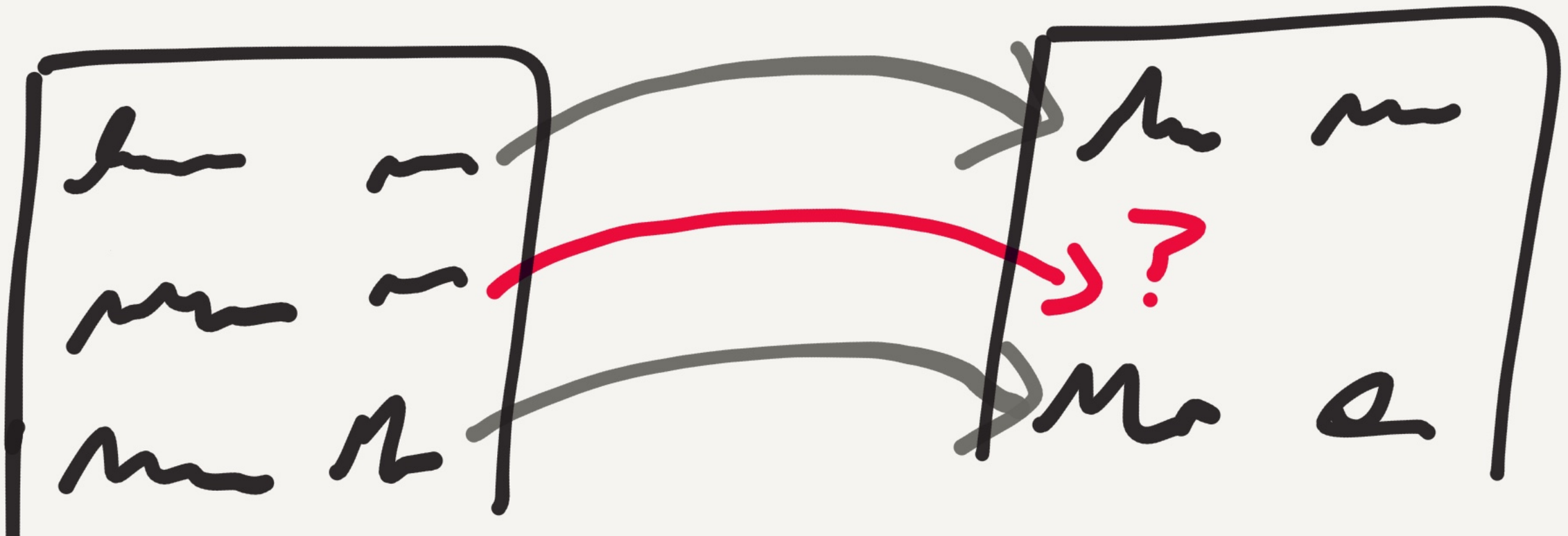
Discovery!

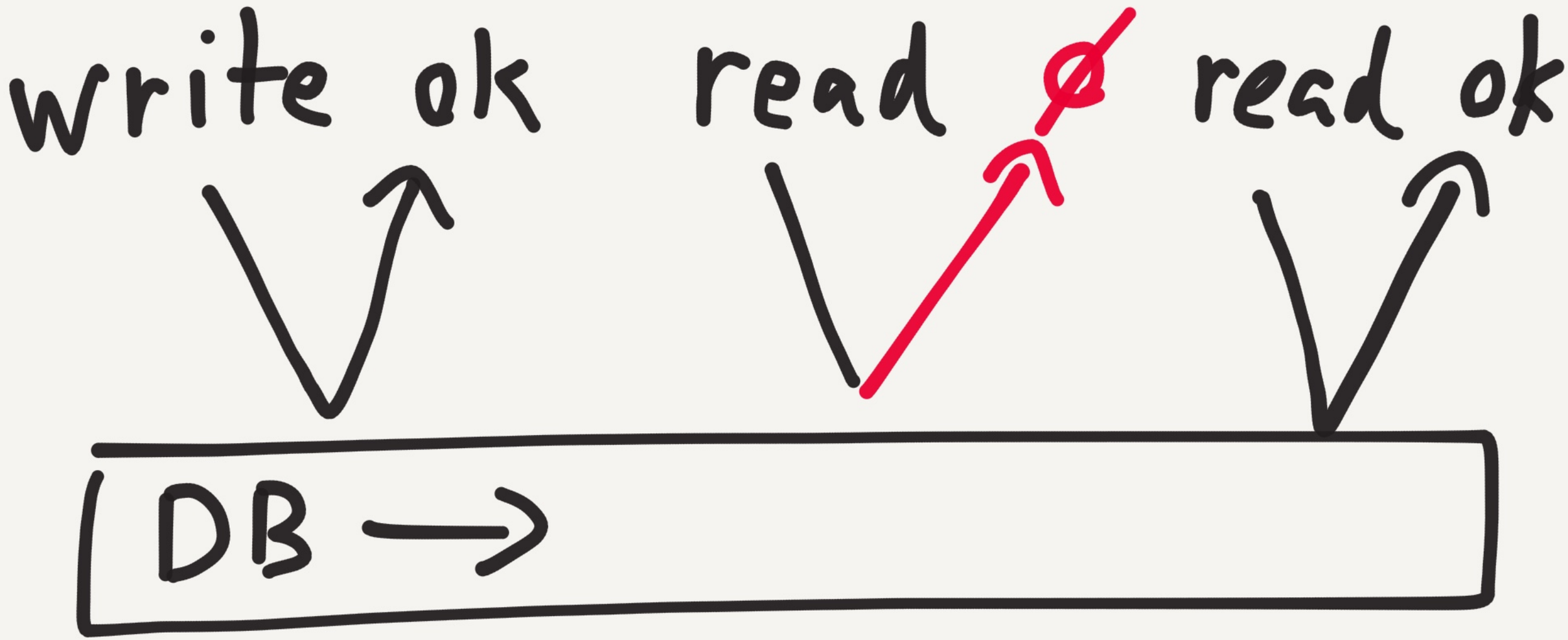
THE HORROR



Split Brain

Broken Foreign Keys

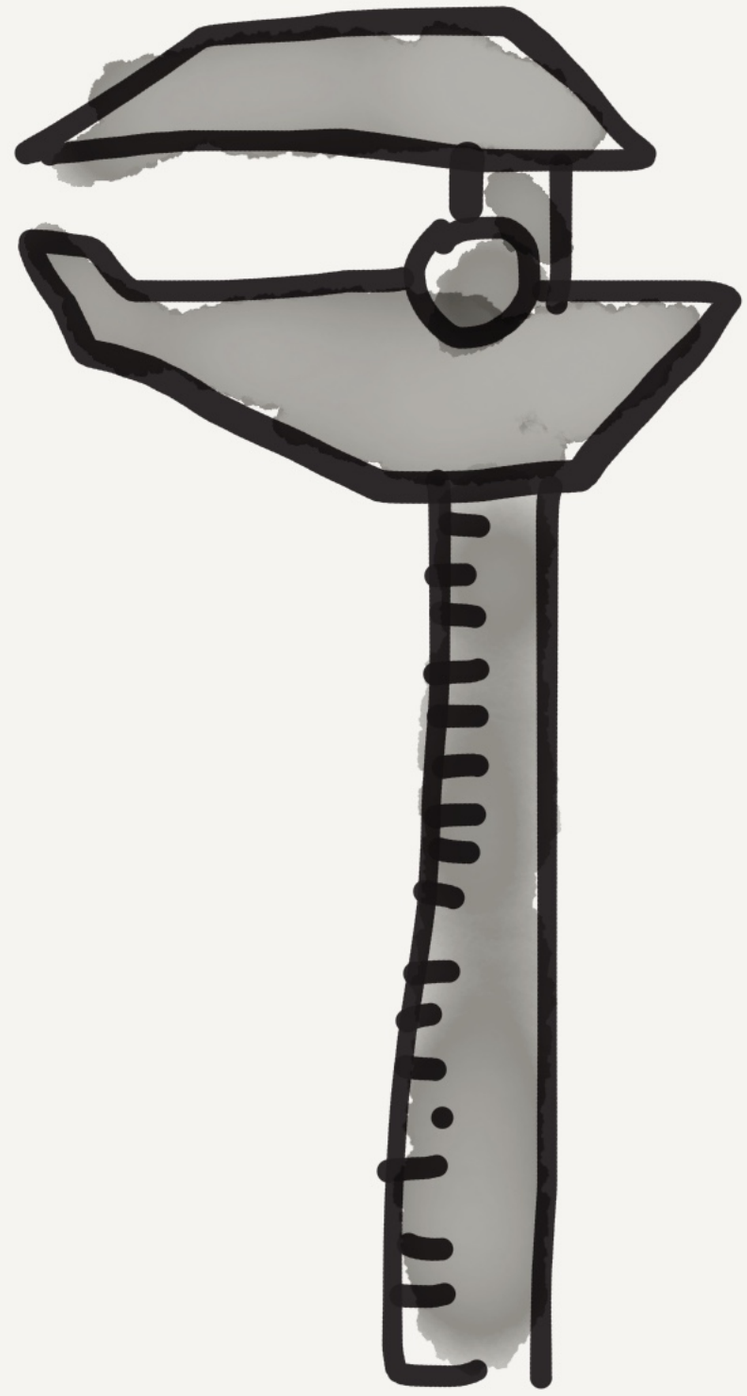




Anomalies

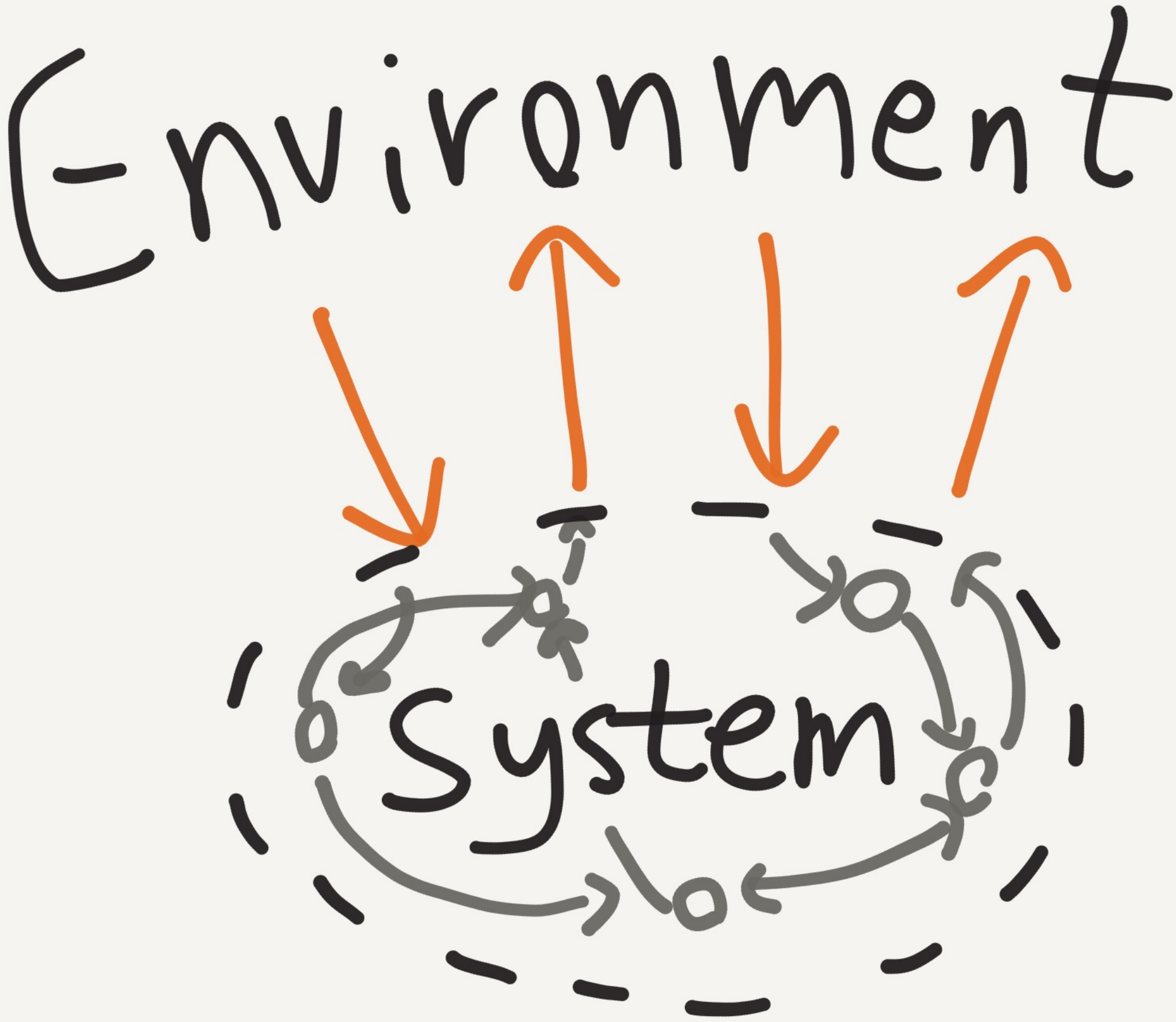
How do you
know if a
system is safe?

Measure your Systems

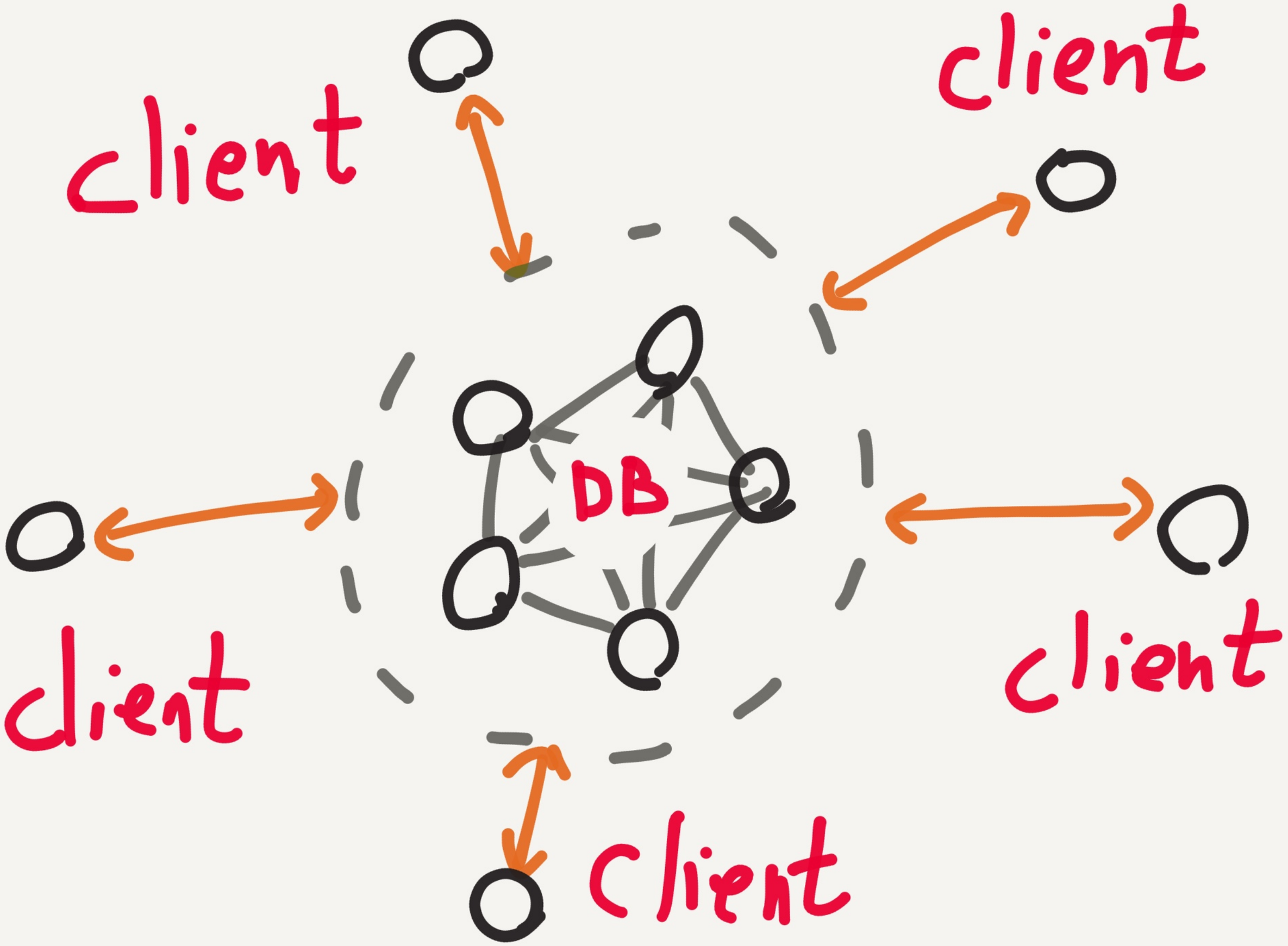


Jepsen

github.com/aphyr/jepsen


Environment
System

-INVARIANTS-

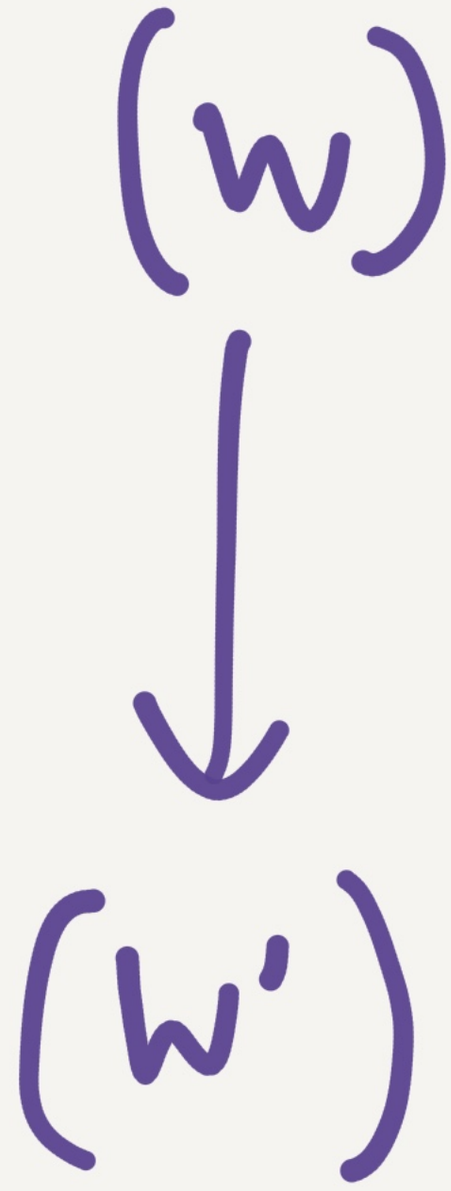

client
client
DB
client
client
client

client: w—w' r—r' w—w'

DB:

client : w —— w' w —— - - -

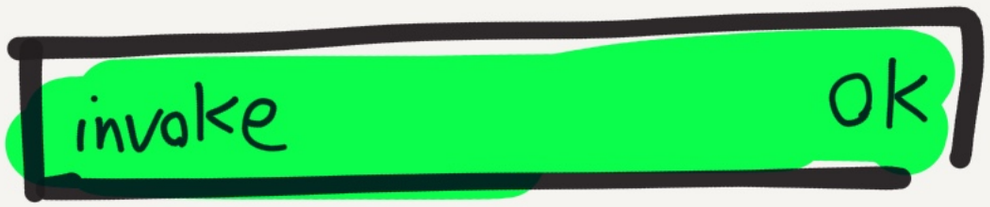
Clients Generate random operations (w) and apply them to the system



invoke ok

invoke fail

involce ? ? info ? ? ?

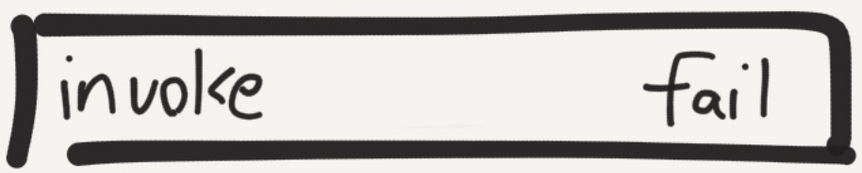

invoke
ok


invoke
fail


invoke
ok


invoke
info


invoke
ok


invoke
fail


invoke
ok


invoke
info

invoke ok

invoke fail

invoke ok

invoke info

invoke ok

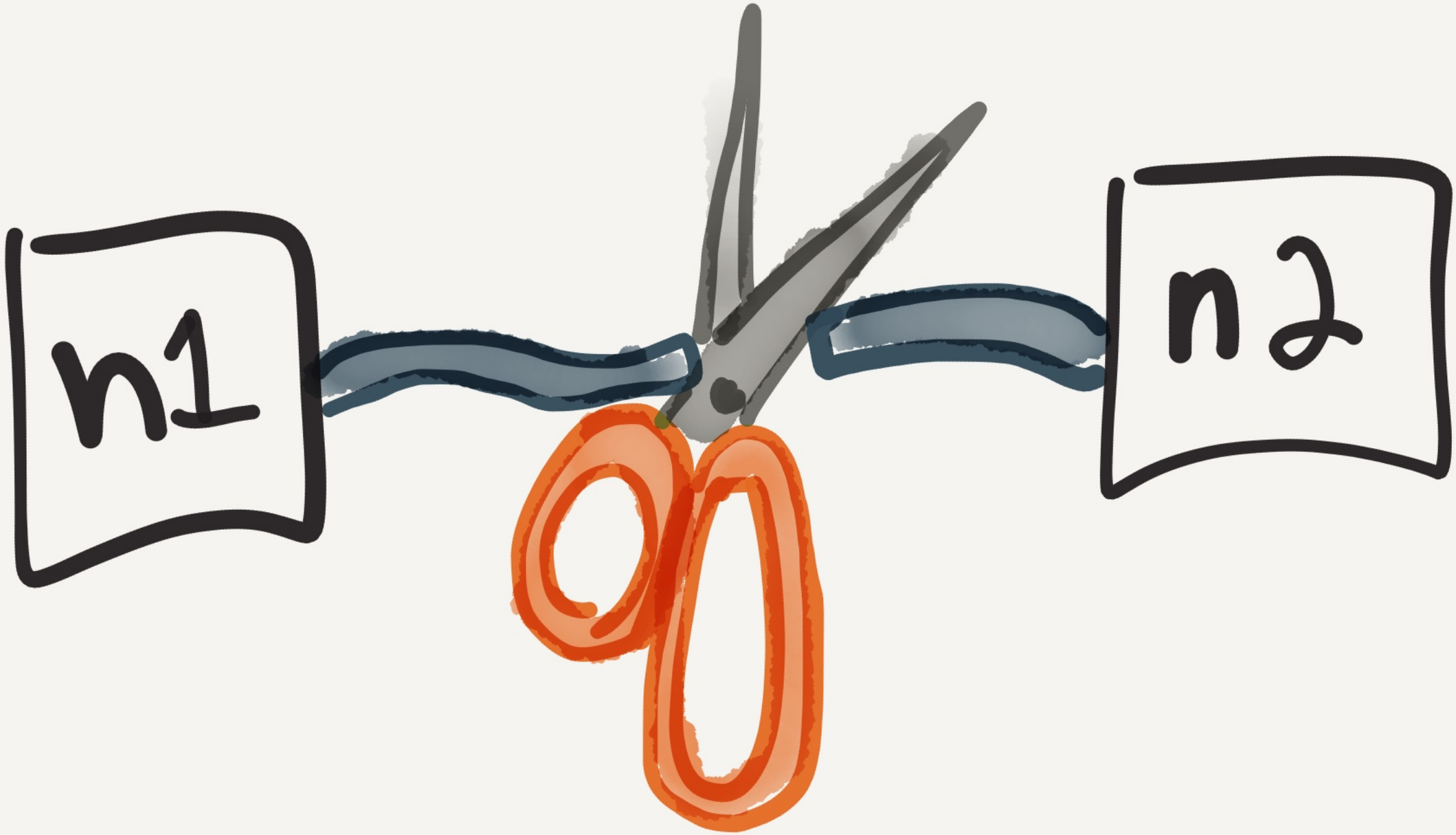
invoke fail

invoke ok

invoke ok

invoke info

1. Generate ops
2. Record history
3. Verify history is consistent w/ model



Partitions!

“So, what have you found?”

Riak

5/13

LWW → lost write

CRDTs → safe

5/13

Mongo

Data loss at all

Write Concerns

Redis Sentinel

Split brain,

massive write loss

Cassandra

9/13

- LWW write loss
- Row isolation broken
- Transaction deadlock, data loss

NuoDB

9/13

Beat CAP by buffer-ing all requests in RAM during partition

Kafka

In-sync Replica Set could shrink to 0 nodes, causing msg loss.

Zookeeper

Works.

etcd / Consul

6/14

stale reads

Elasticsearch

Loses documents in every class of partition tested.

RabbitMQ

Split brain, massive

message loss

Aerospike

Claims "ACID", was really LWW.

Elasticsearch 1.5.0

5/15

Still loses data

in every test case

MongoDB 2.6.7

5/15

stale reads

dirty reads

Chronos

8/15

- Breaks forever after losing quorum

Percona XtraDB/Galera 9/15

- "Snapshots" weren't
- First-committer-wins not preserved
- Read locks broken

RethinkDB

- Basic tests passed
- Reconfiguration could destroy cluster in rare cases

VoltDB 6.3

- Stale reads
- Dirty reads
- Lost writes

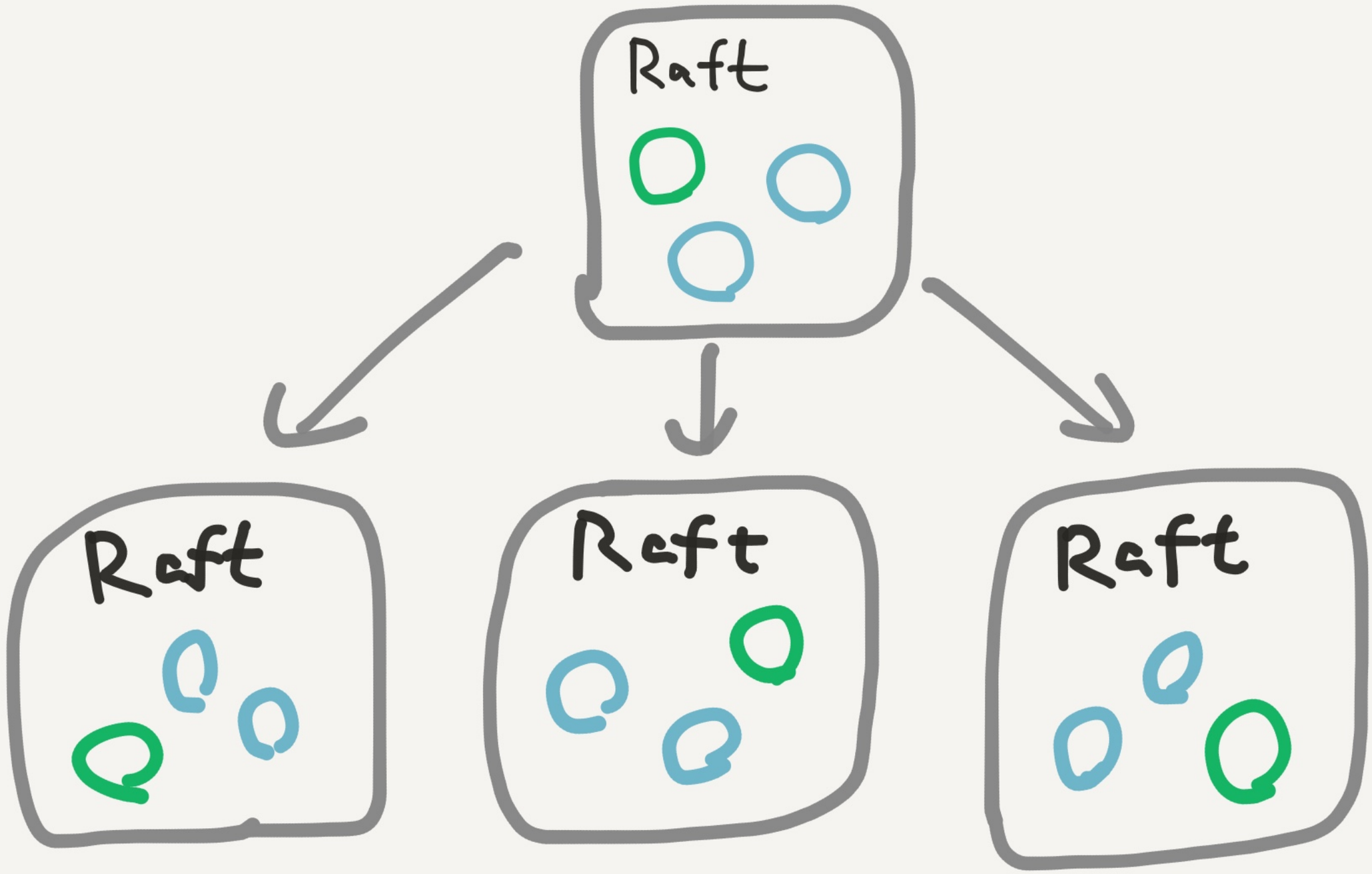
Crate.io

2016

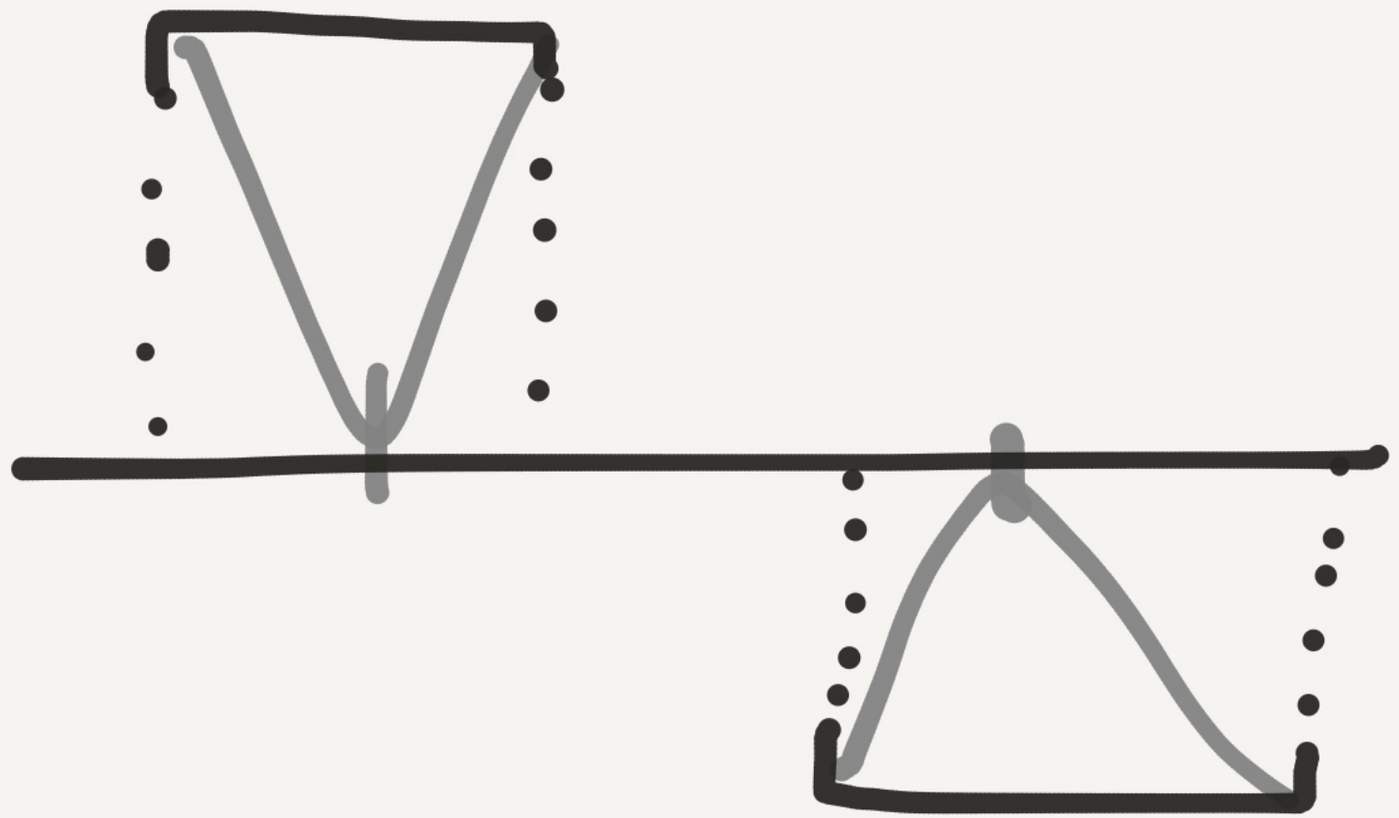
- Stale reads
- Dirty reads
- Lost/corrupt updates
- Lost inserts


CockroachDB

- Scale-out SQL
- Postgres wire protocol
- Like Spanner*
- Hybrid Logical Clocks


Raft
Raft
Raft
Raft

Txns in a Raft cluster are linearizable



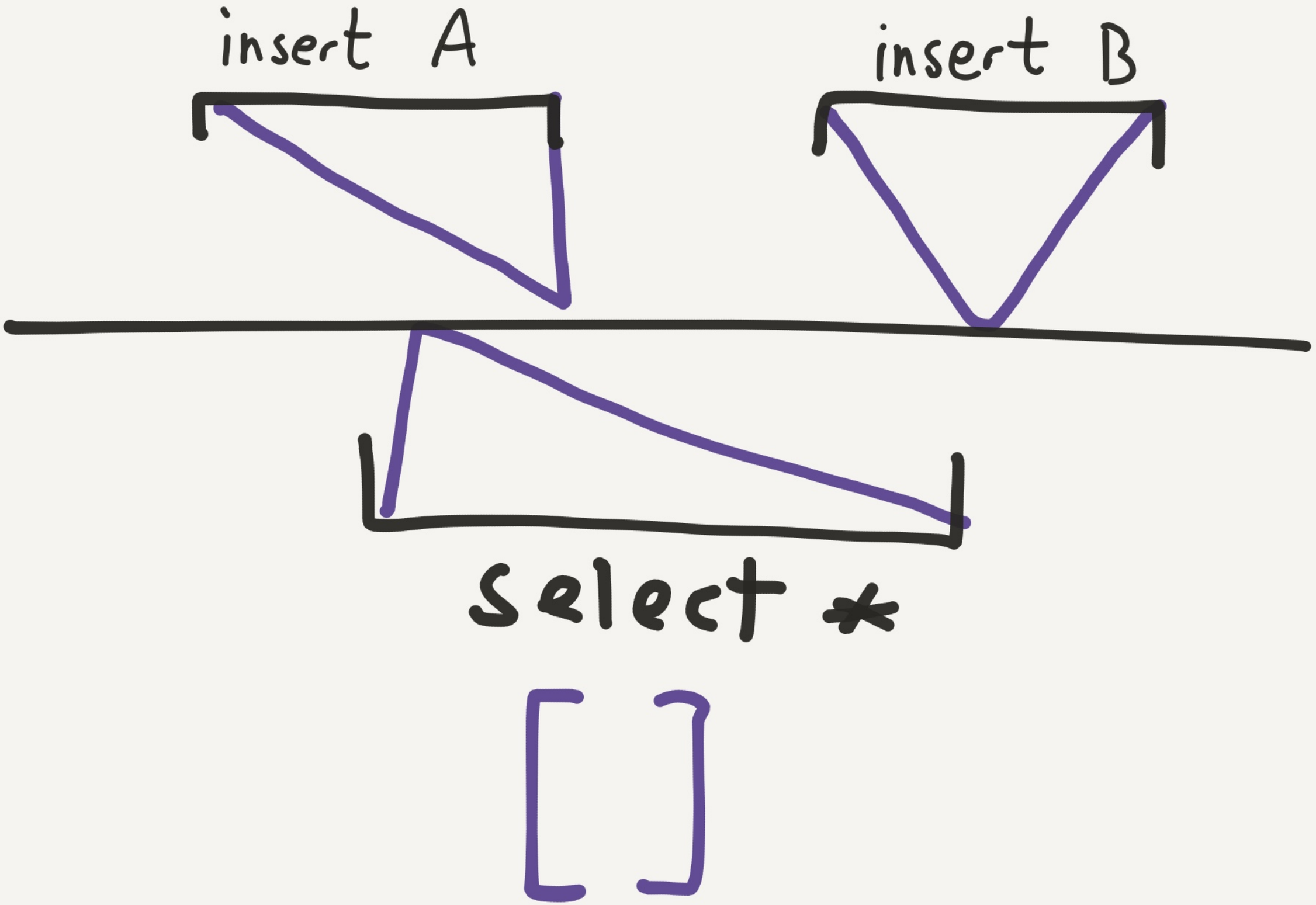
Cockroach cheats on reads with time-based leader leases

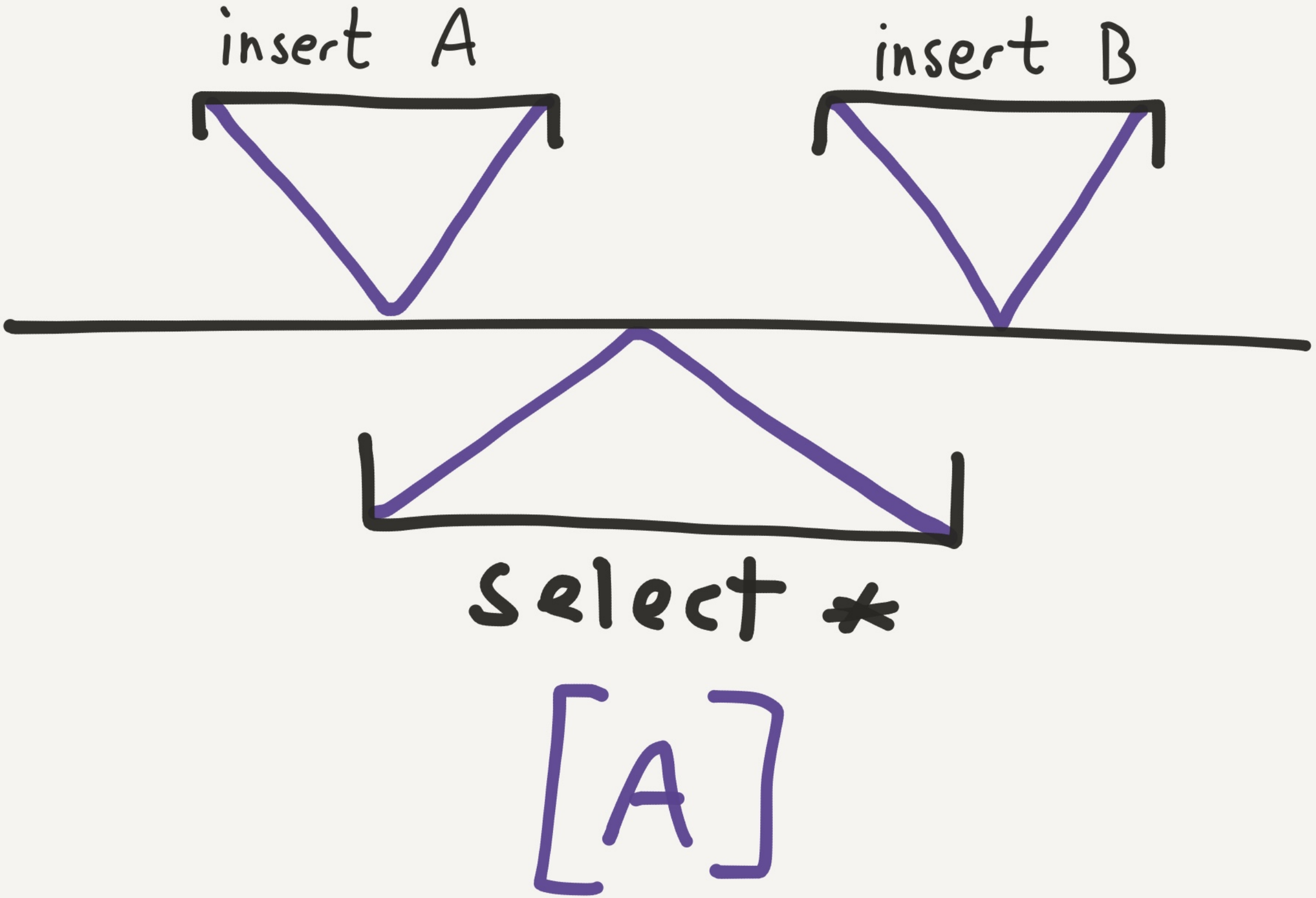
⇒ Needs reliable timeouts

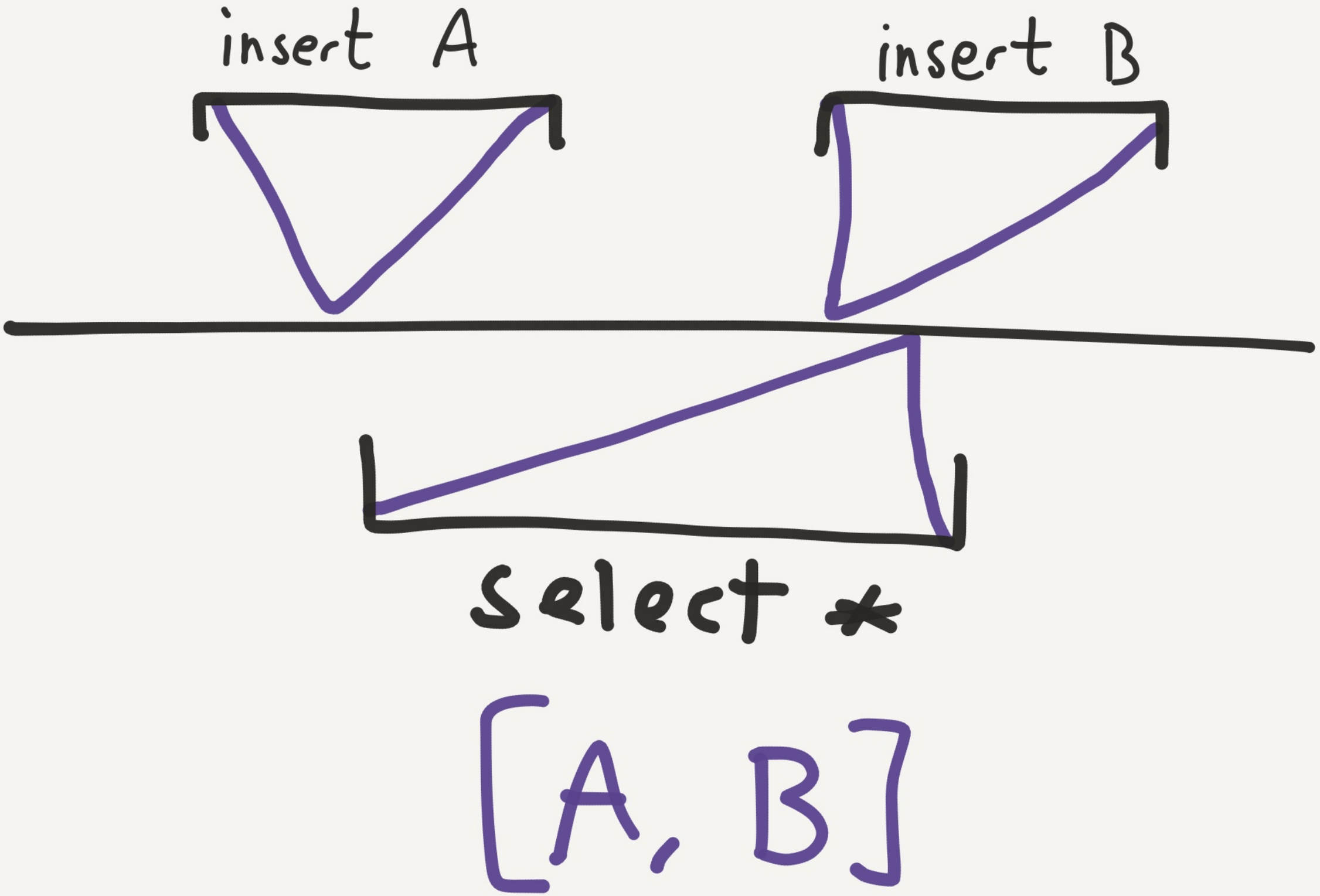
Txns across n Raft clusters use a custom protocol w/HLC

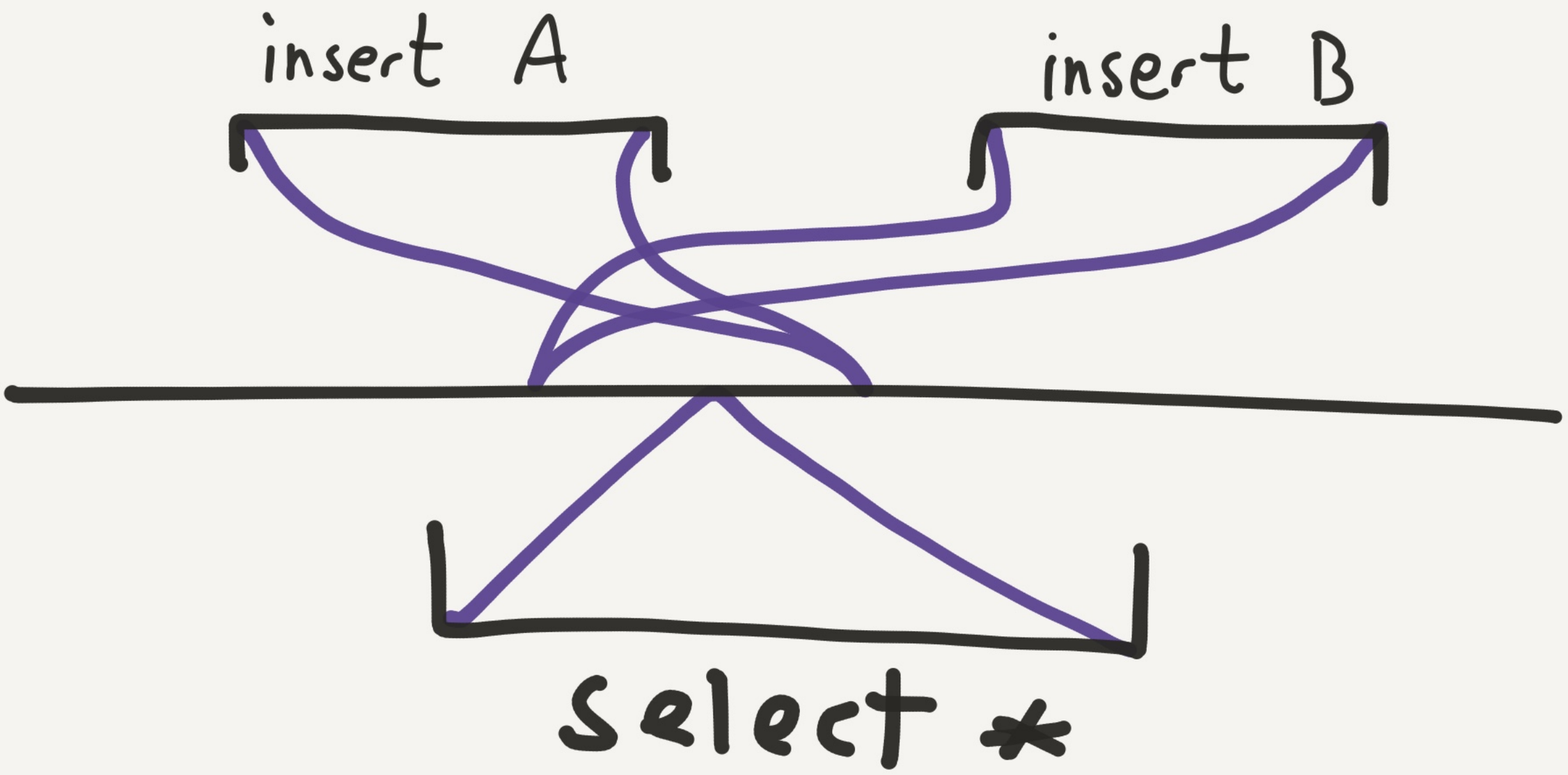
⇒ Needs synced clocks

- Writes commit once replicated.
- Reads on contested keys may block up to clock skew threshold.


insert A
insert B
select *
[ ]


insert A
insert B
select *
[A]


insert A
insert B
select *
[A, B]



Serializable!

[B]

Nonlinearizable

Cockroach Labs tested with Jepsen

— Clock-skew detector needed to be stricter

— sql timestamps derived from KV timestamps

We have to go Deeper

We found two new issues together!

table1

1
2

table2

0
3

table 1

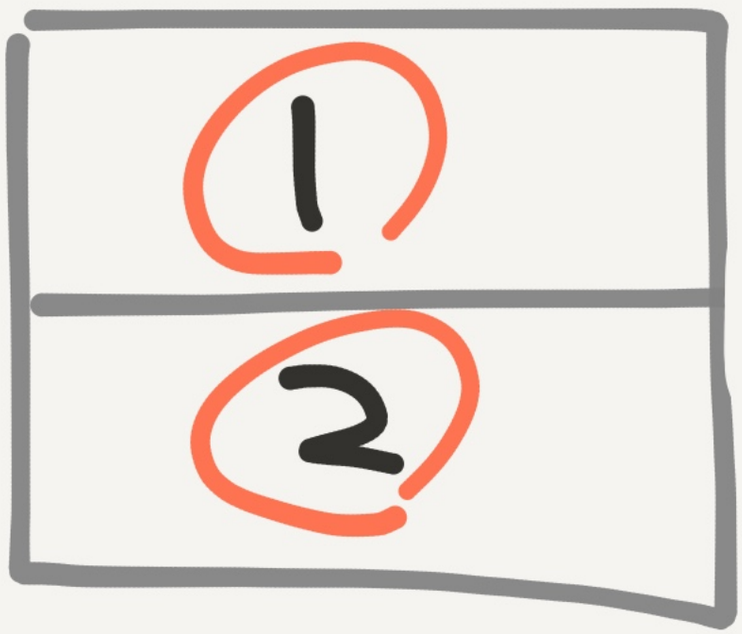
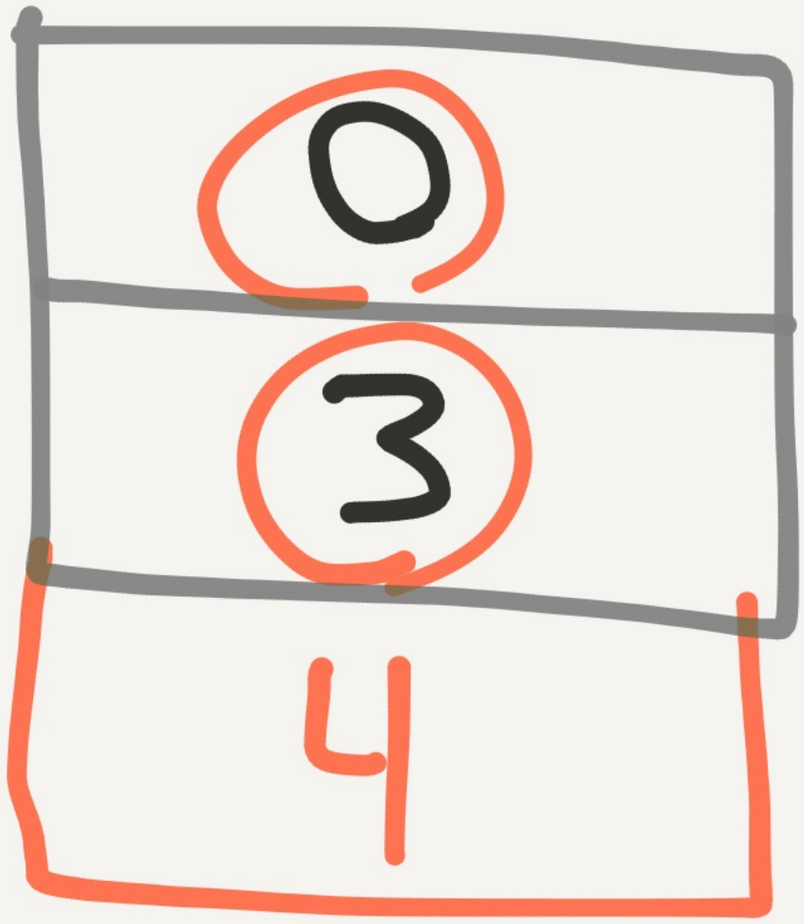


table 2



select max;
insert max+1

table 1

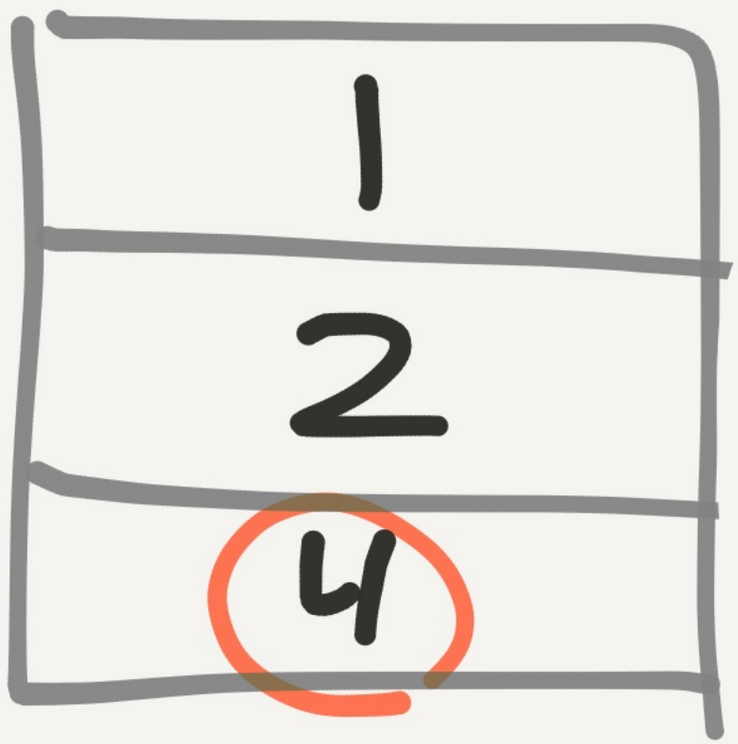
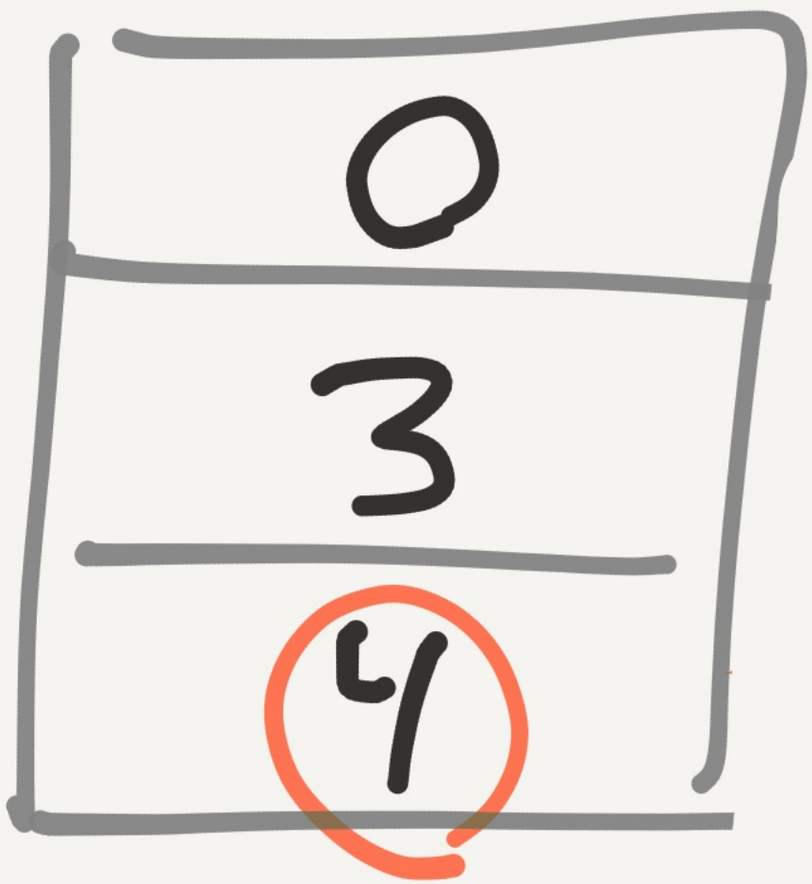


table 2



2 copies!

Time stamp Cache

- Stores set of keys read by a txn
- prevents read keys from modification

Txns with same timestamp can steal each others ts cache entries

Fix: clear txnid for conflicting reads

⇒ Allows read-read, but subsequent writes fail

Problem #2

insert 1;

insert 2;

insert 3;

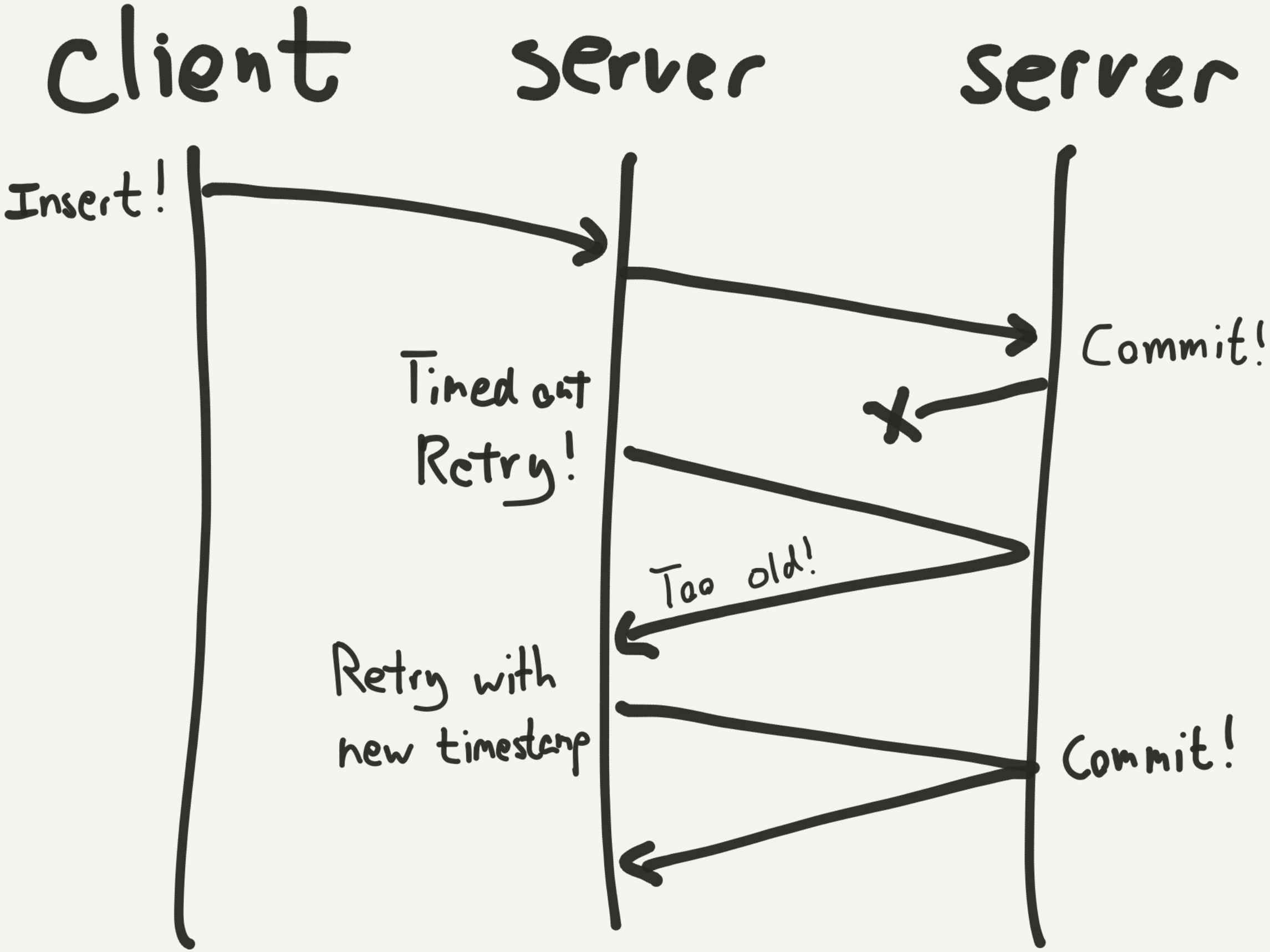
...

select *;

{0, 1, 2, 3, 4, 5, ..., 3}

whaaaaat.

- Single-stmt txns can be executed in 1 phase


client
server
server
Insert!
Commit!
Timed out
Retry!
Too old!
Retry with
new timestamp
Commit!

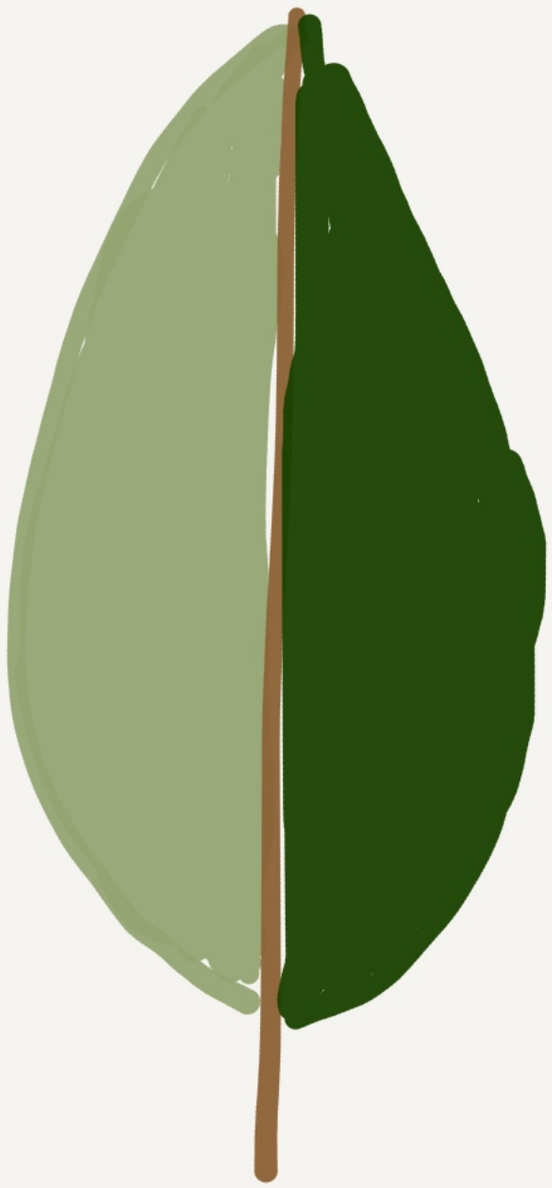
Now distinguishes between retryable & non-retryable errors

CockroachDB now passes its test suite!

— The world needs more serializable SQL DBs

- But remember, clocks MUST be synchronized

- Performance still a sticking point - it's β



mongoDB

3.4.0-rc3

- Document Store
- BSON docs
- Custom replication
- Sharding layer

2013

Lost writes

2015

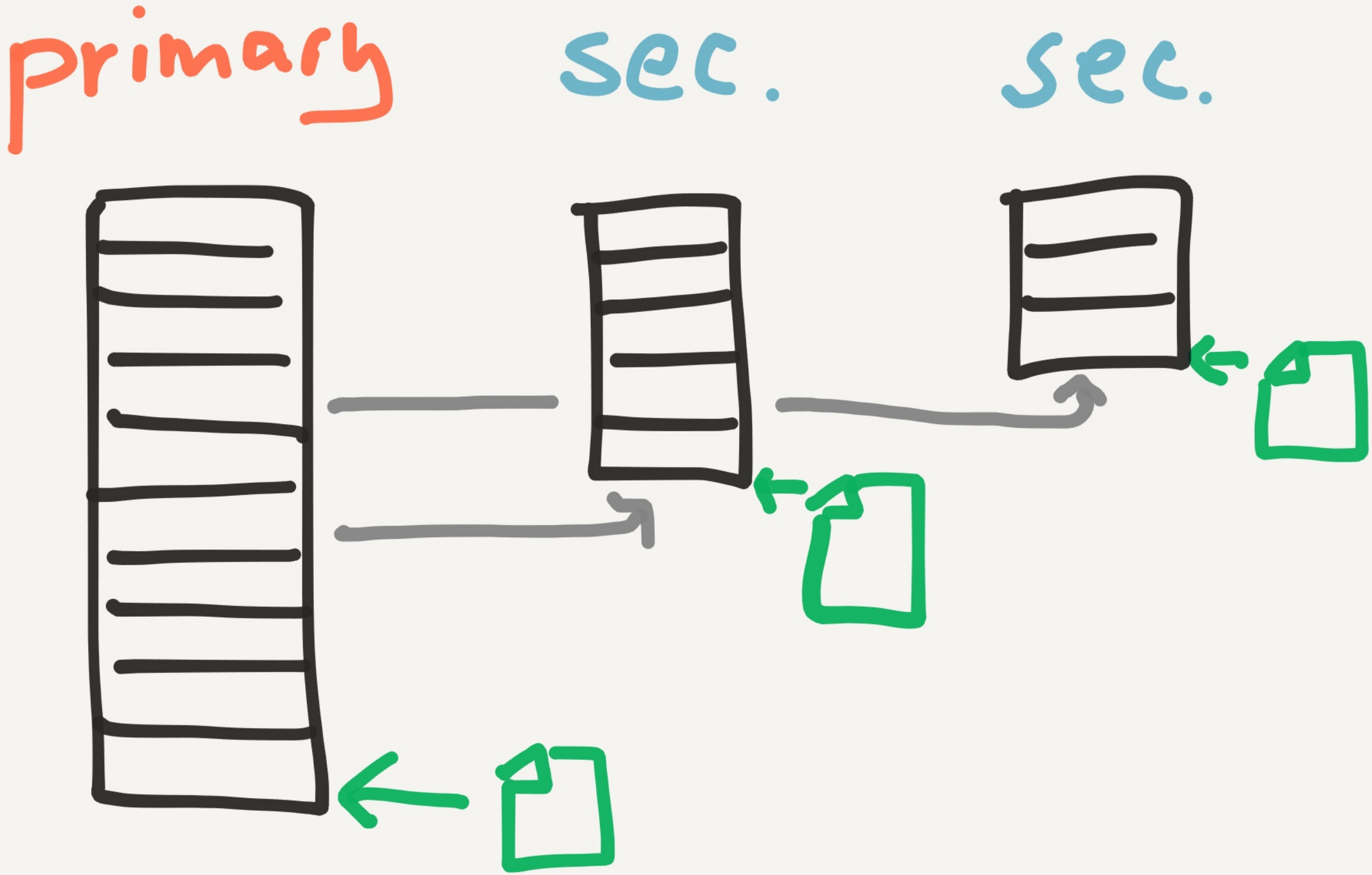
Safe writes

Dirty/stale reads

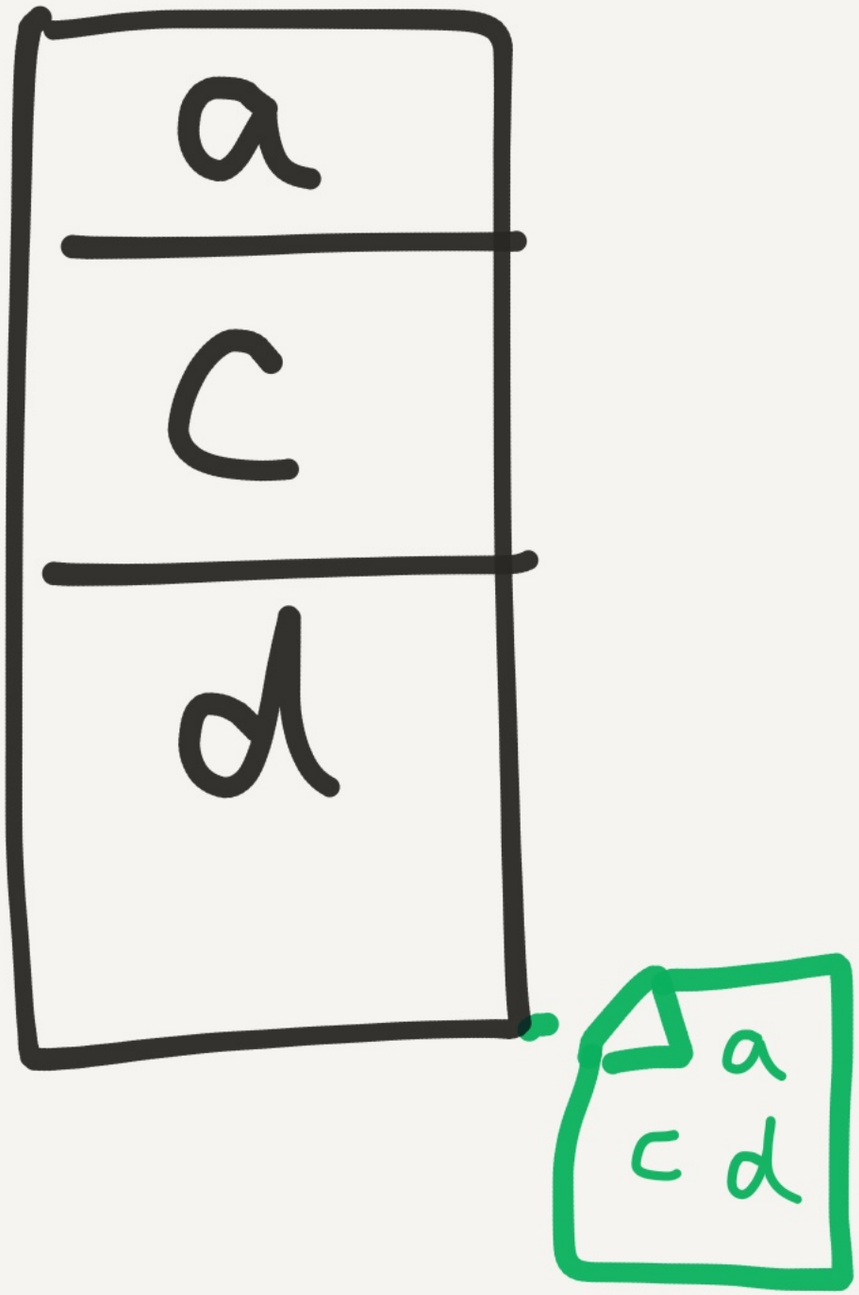
3.0: WiredTiger Replication v1

3.2: Majority reads

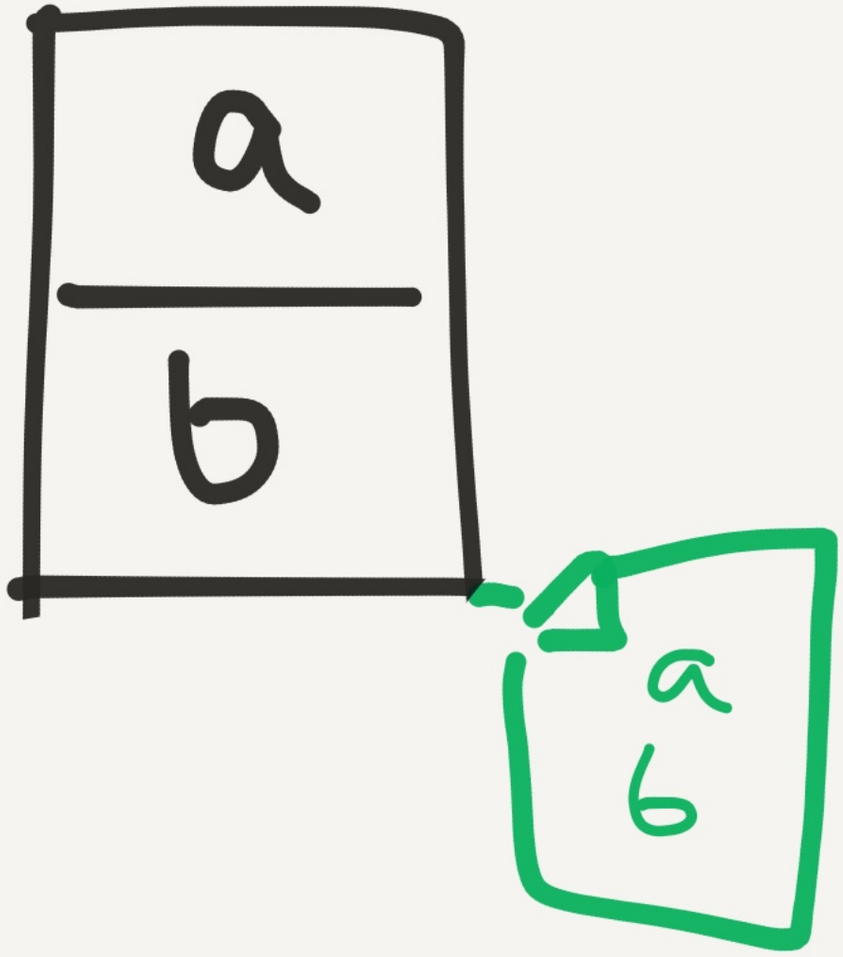
3.4: Linearizable reads


primary
sec.
sec.

primary

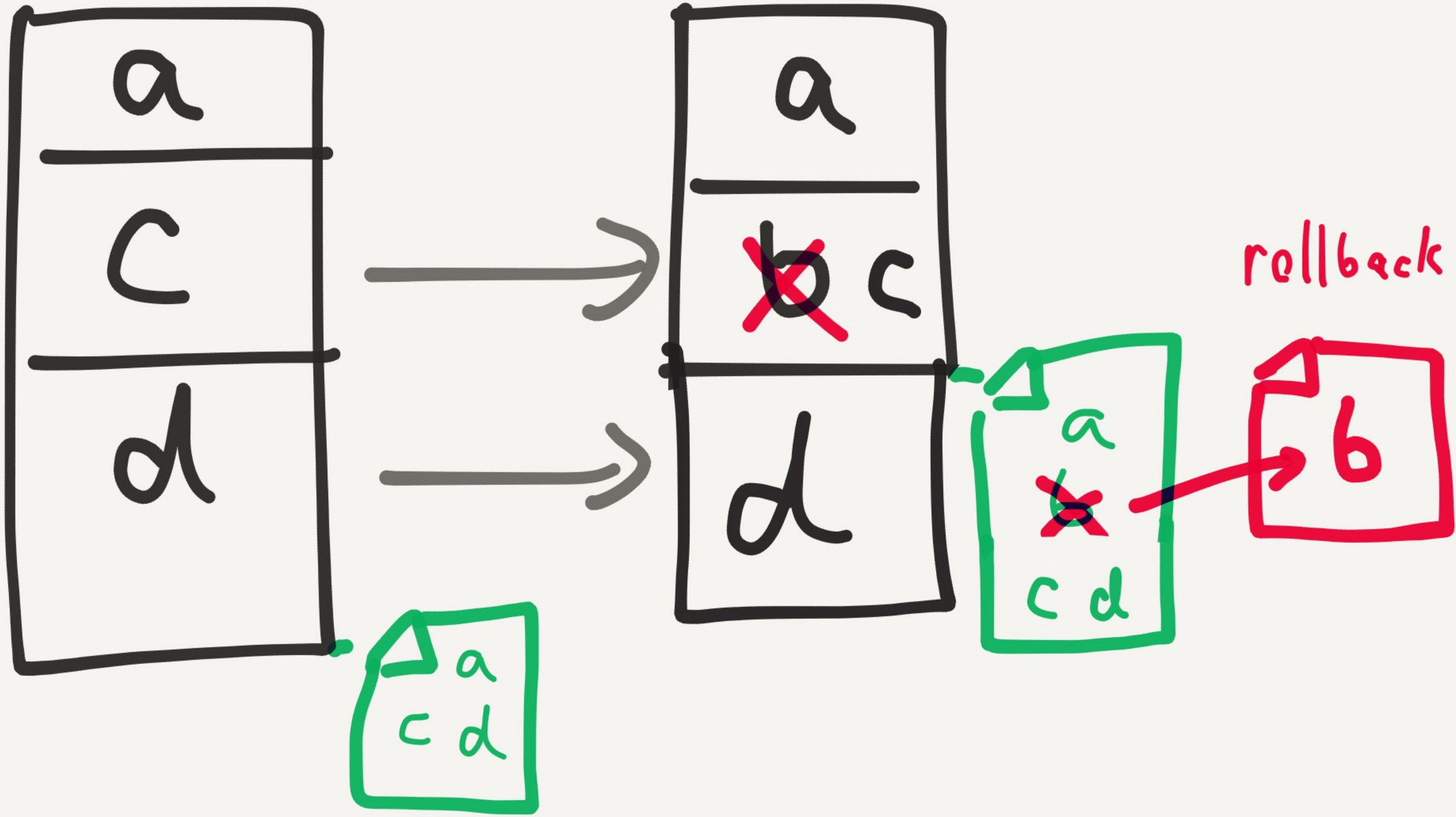


secondary



primary

secondary

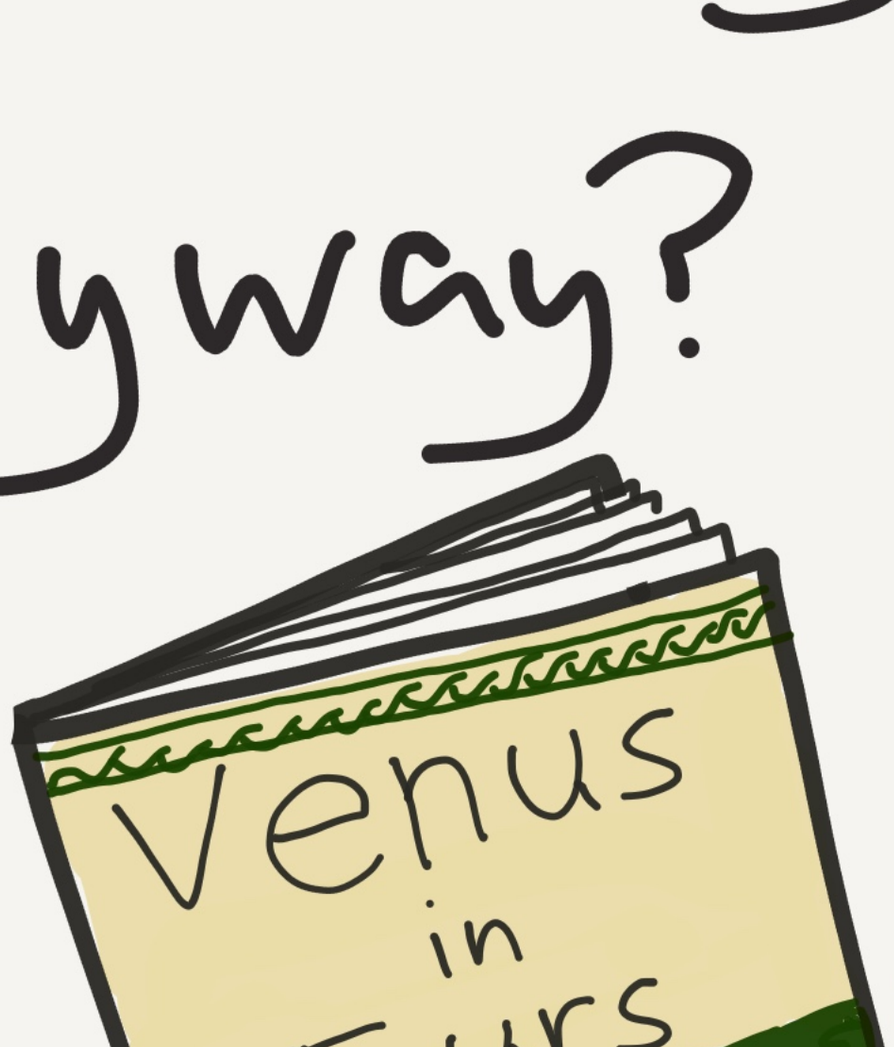


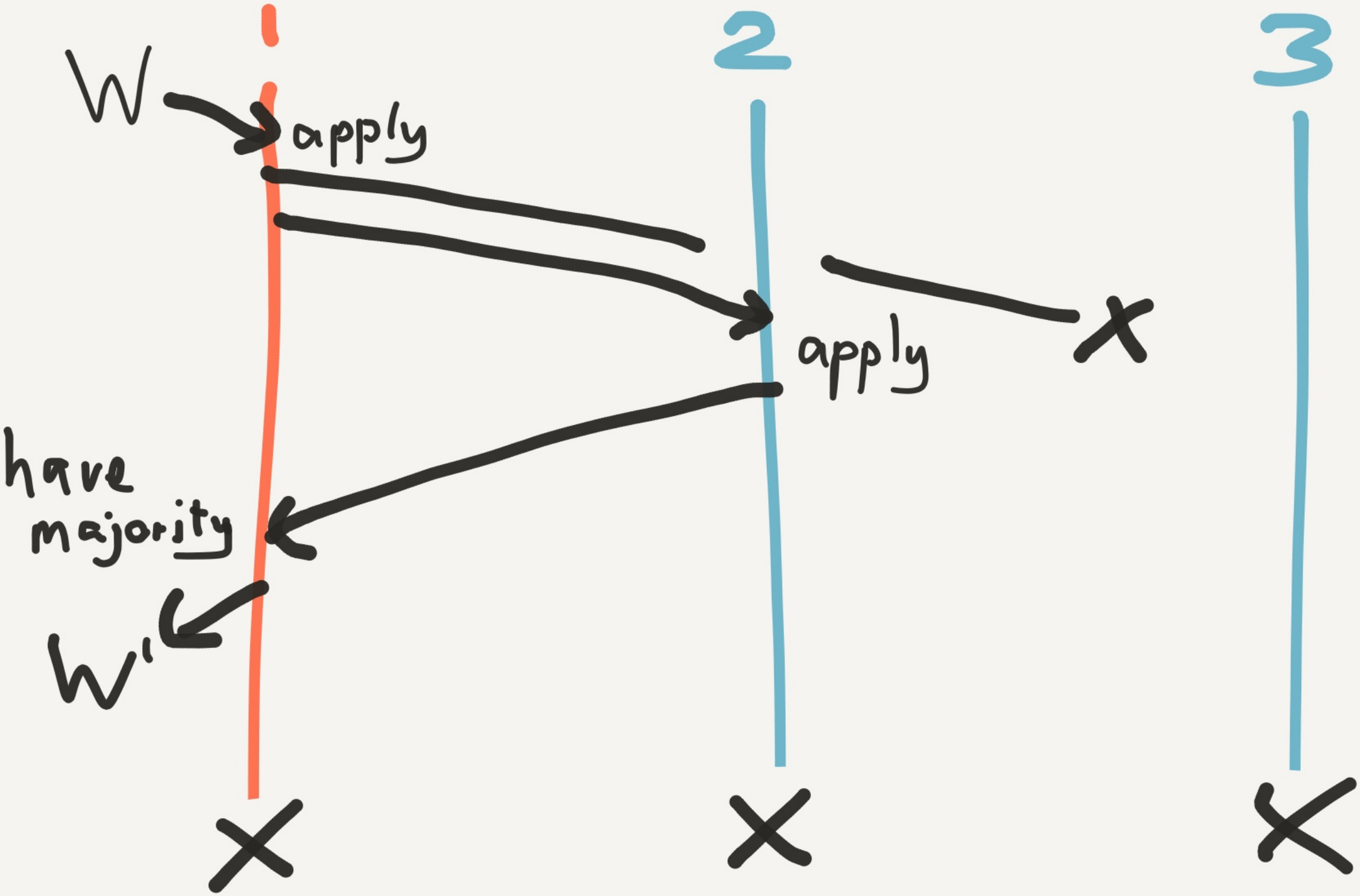
v0 protocol:

- Dirty reads

- Lost writes?

Ok but how bad are dirty reads anyway?




1
2
3
W
apply
apply
X
have majority
W'
X
X
X

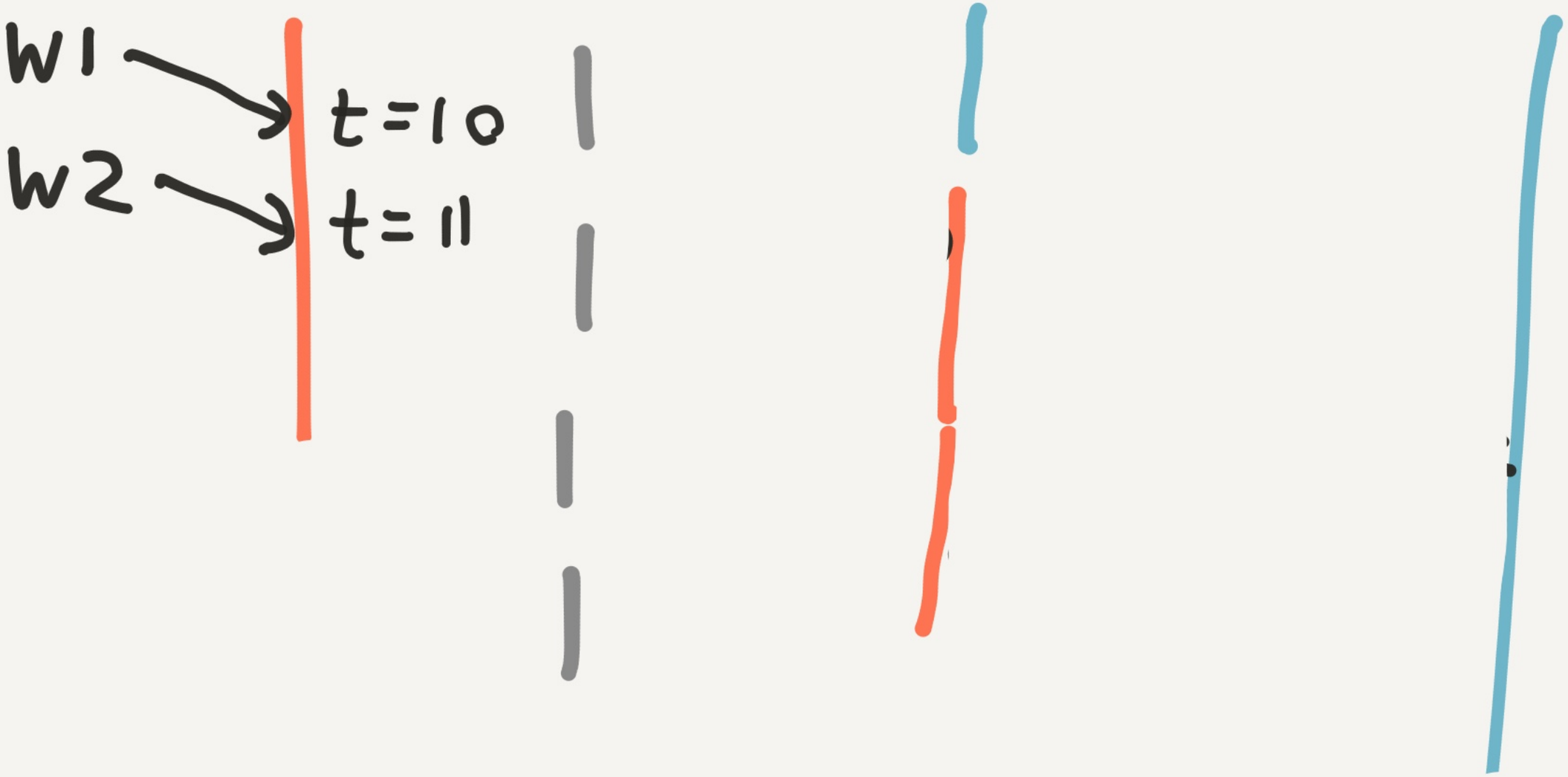
???

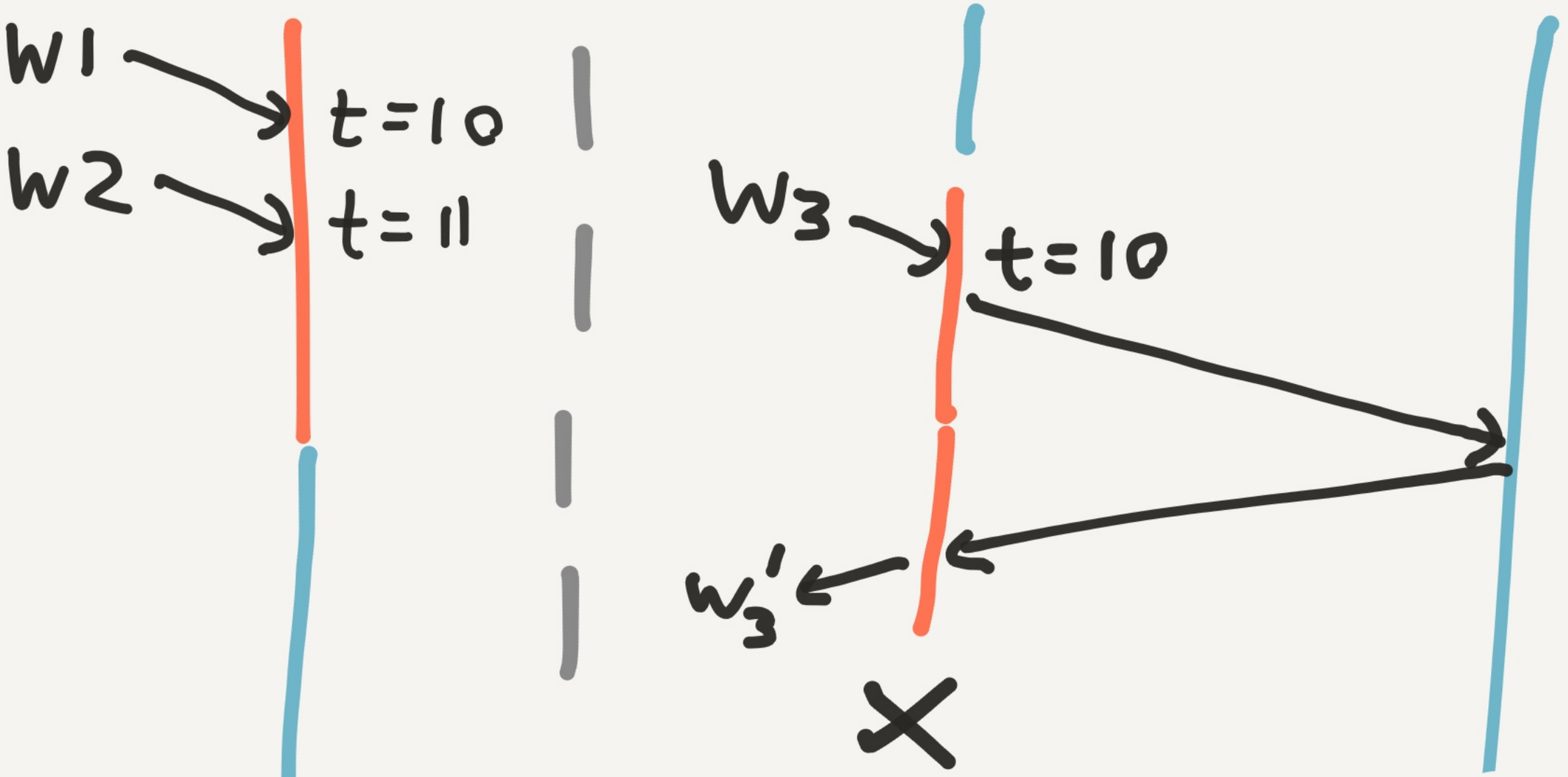
Optime:

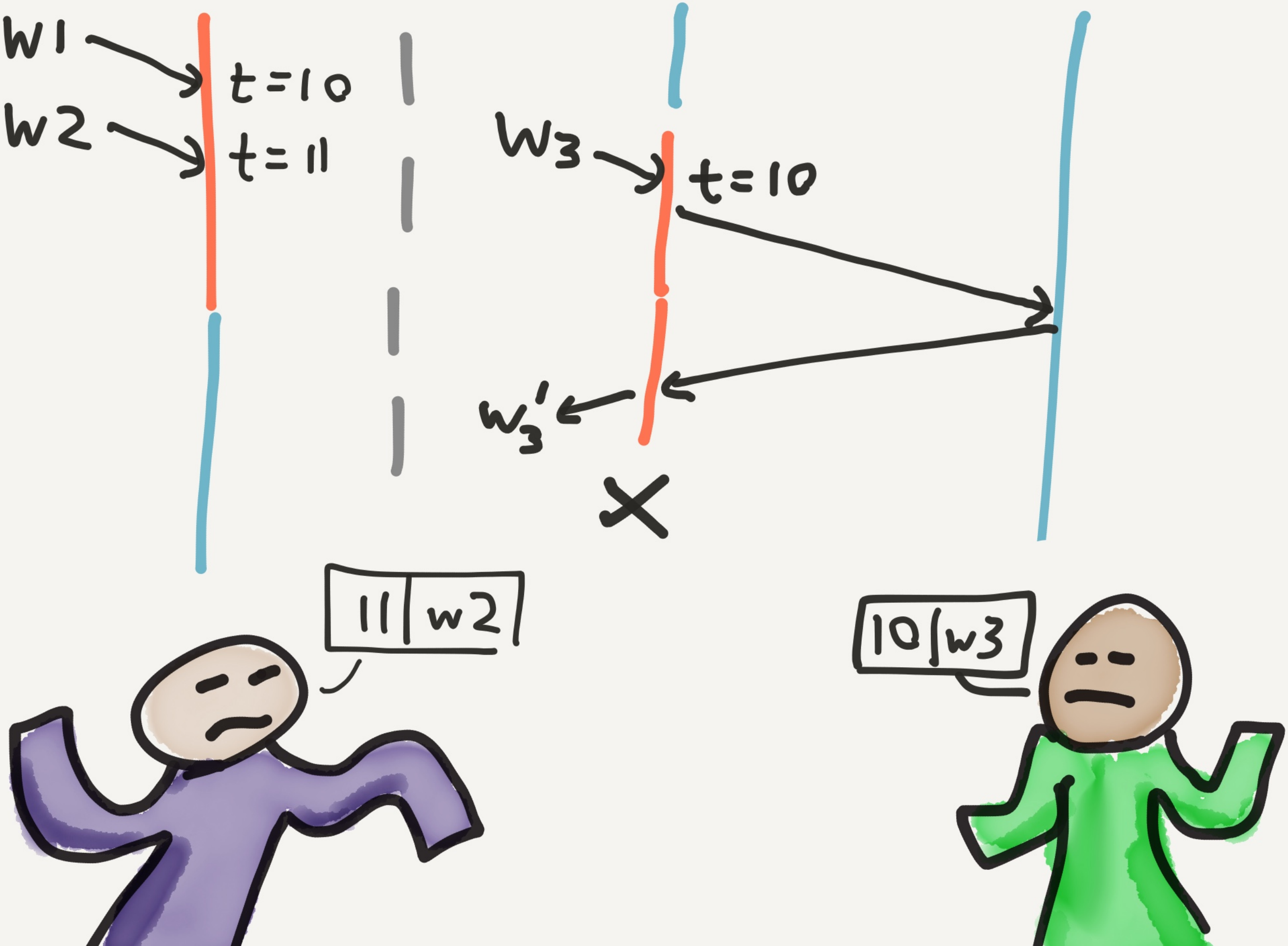
$$\text{Max}\begin{cases}\text{last optime} + 1 \\ \text{local wall clock}\end{cases}$$

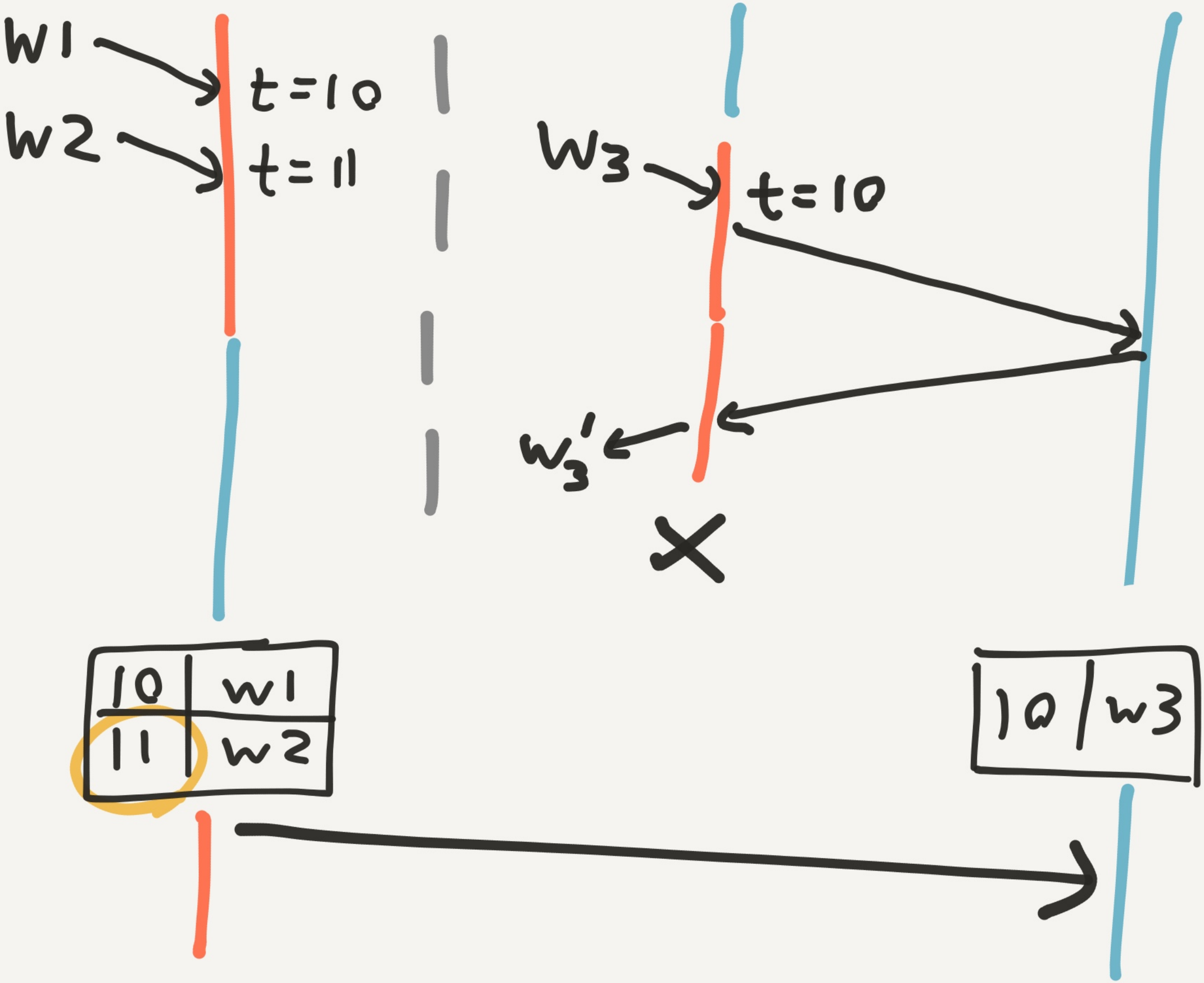
Optime	Operation
10	~~~
11	~~~
12	~~~
15	~~~
20	~~~

Node w/ highest optime wins


w1
w2
t=10
t=11

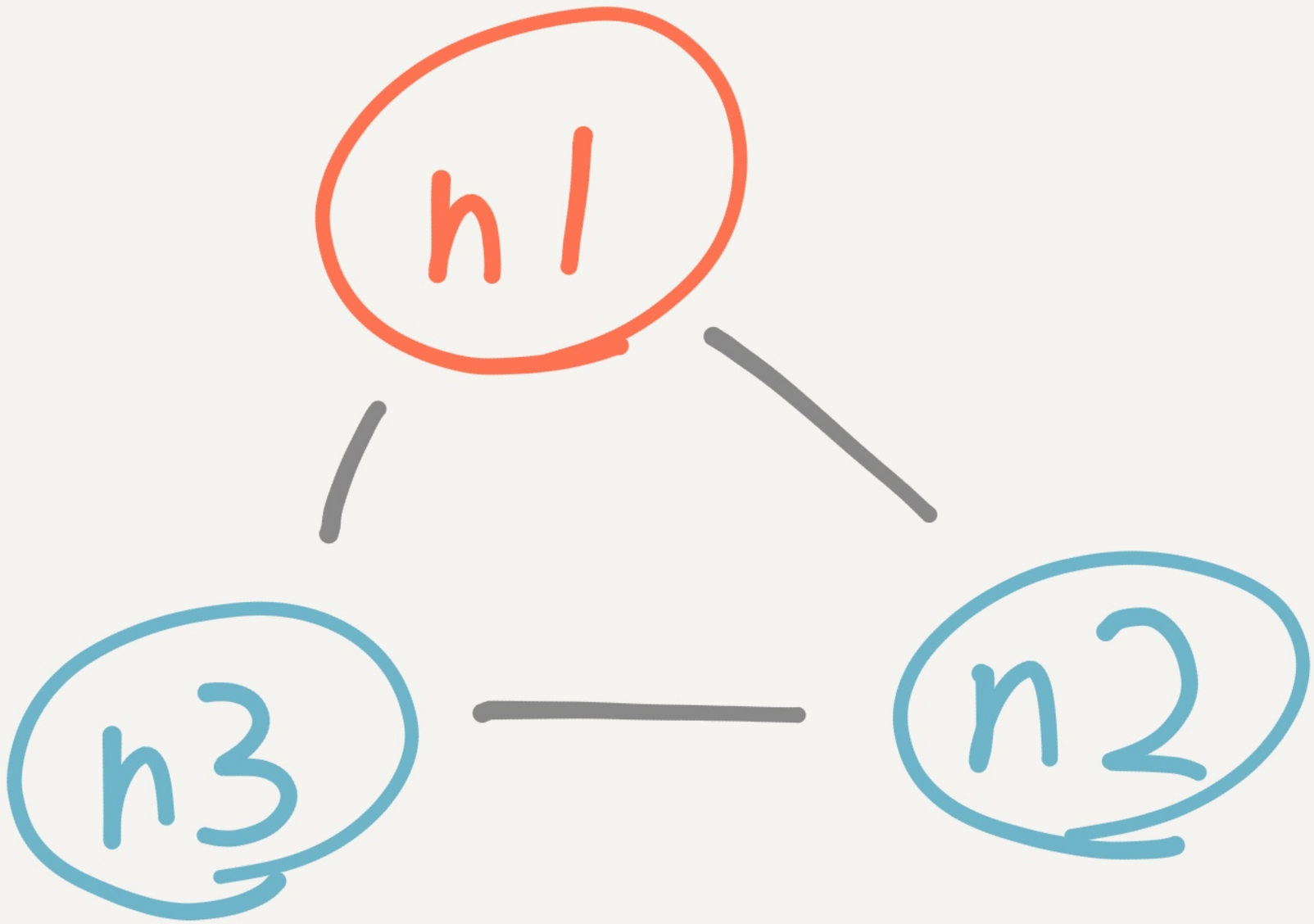

w1
w2
t=10
t=11
w3
t=10
w3'

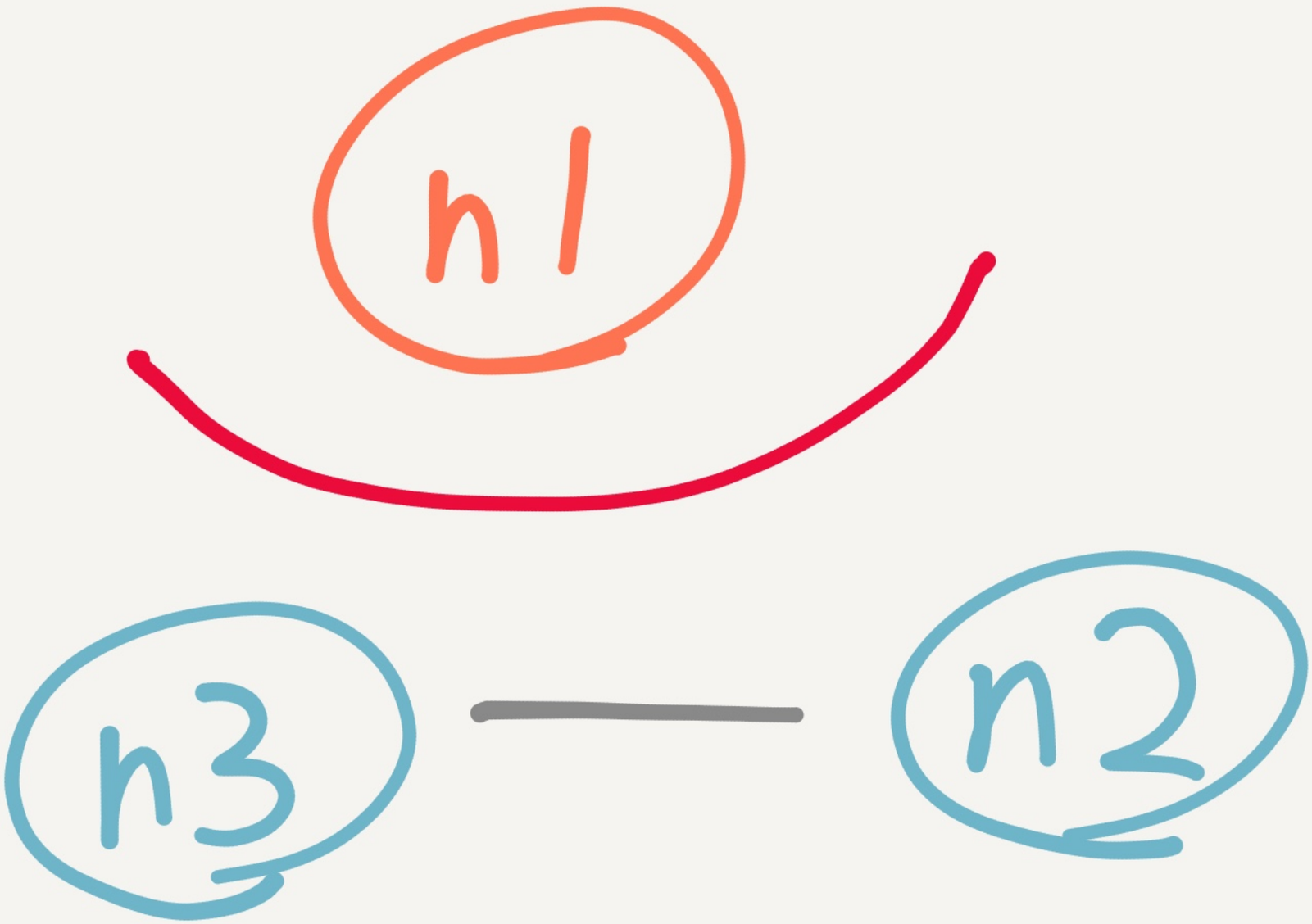

w1
w2
t=10
t=11
w3
t=10
w3'
11|w2
10|w3

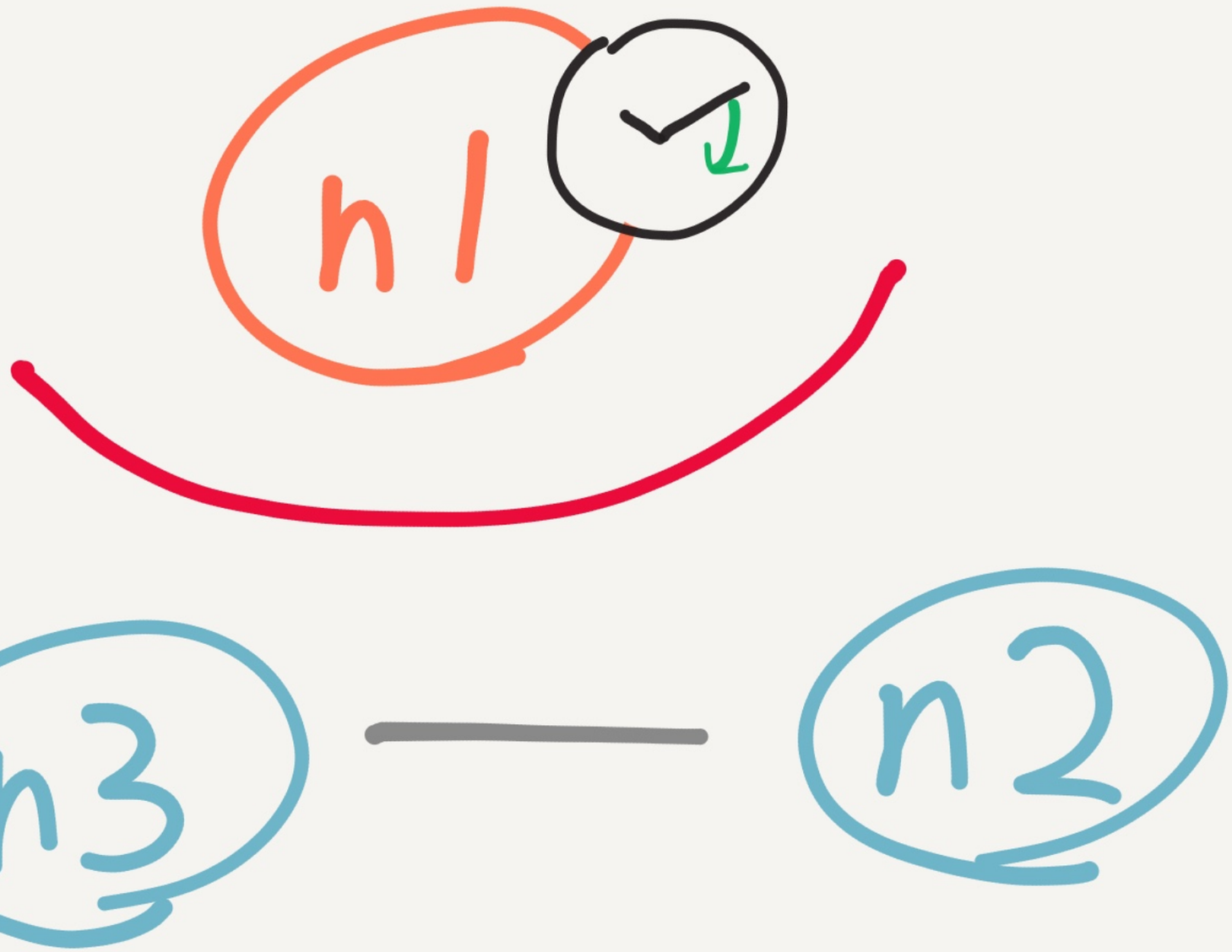

w1
w2
t=10
t=11
w3
t=10
w3'
10 | w1
11 | w2
10 | w3

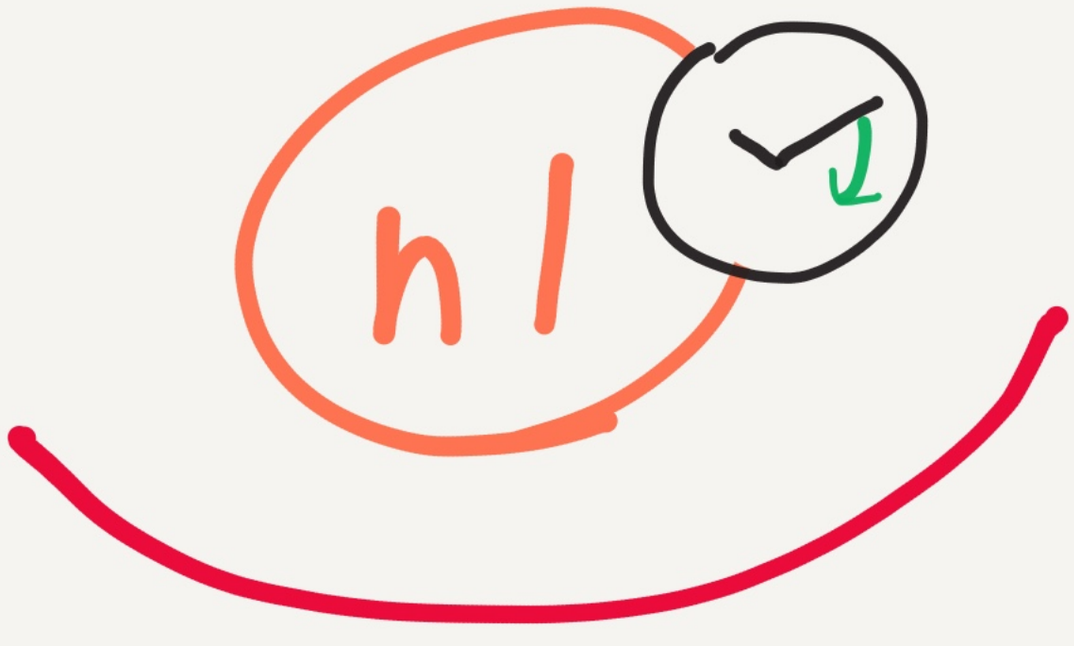
Requires precise timing

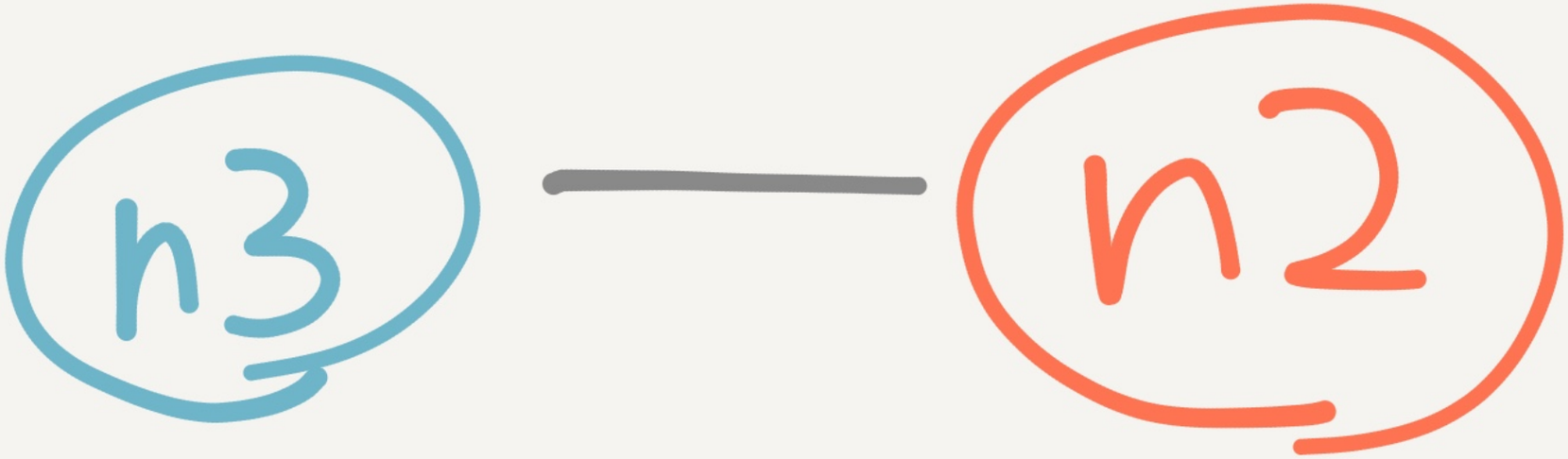
– But window grows with clock skew!


n1
n3
n2

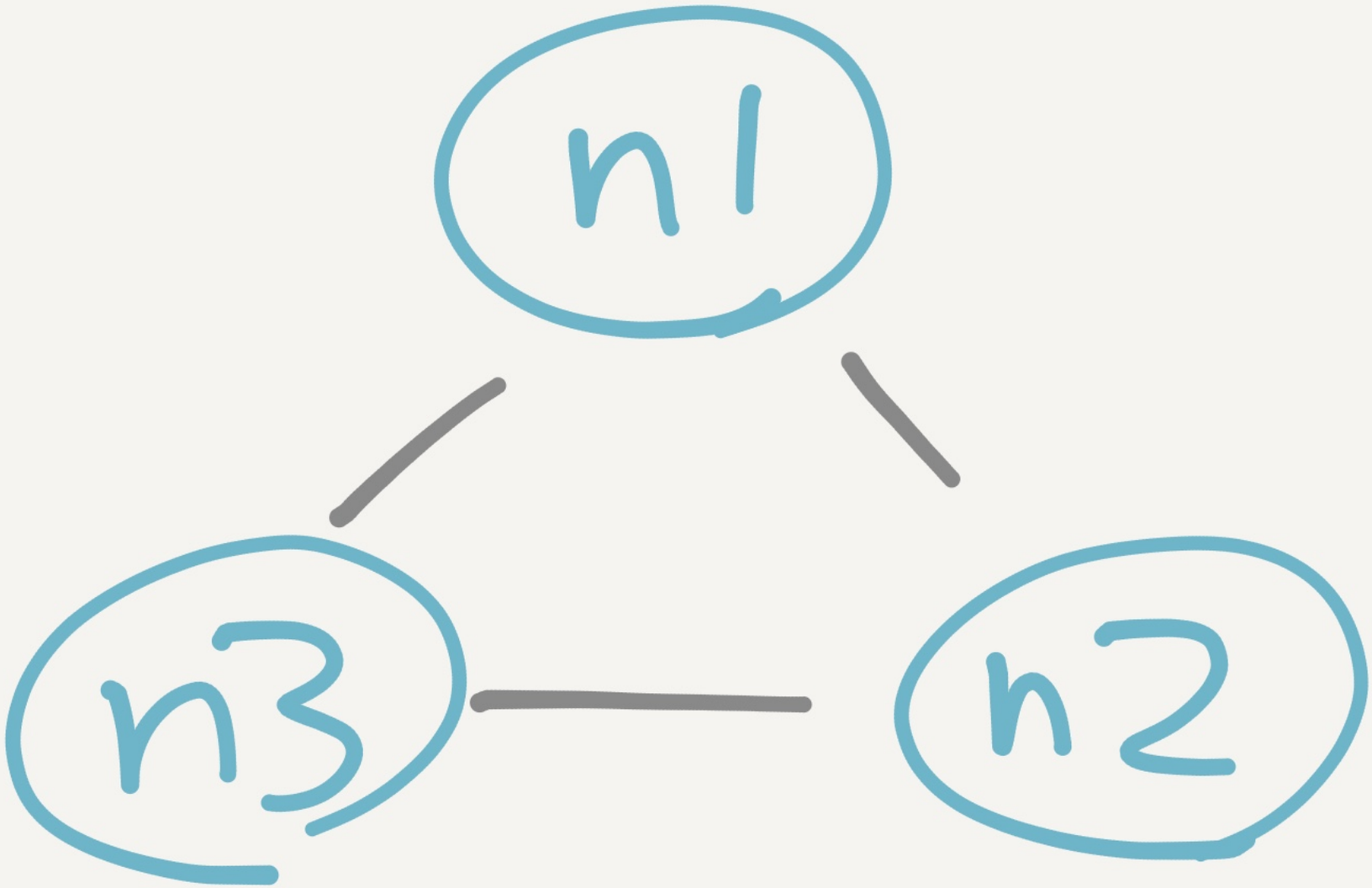

n1
n3
n2

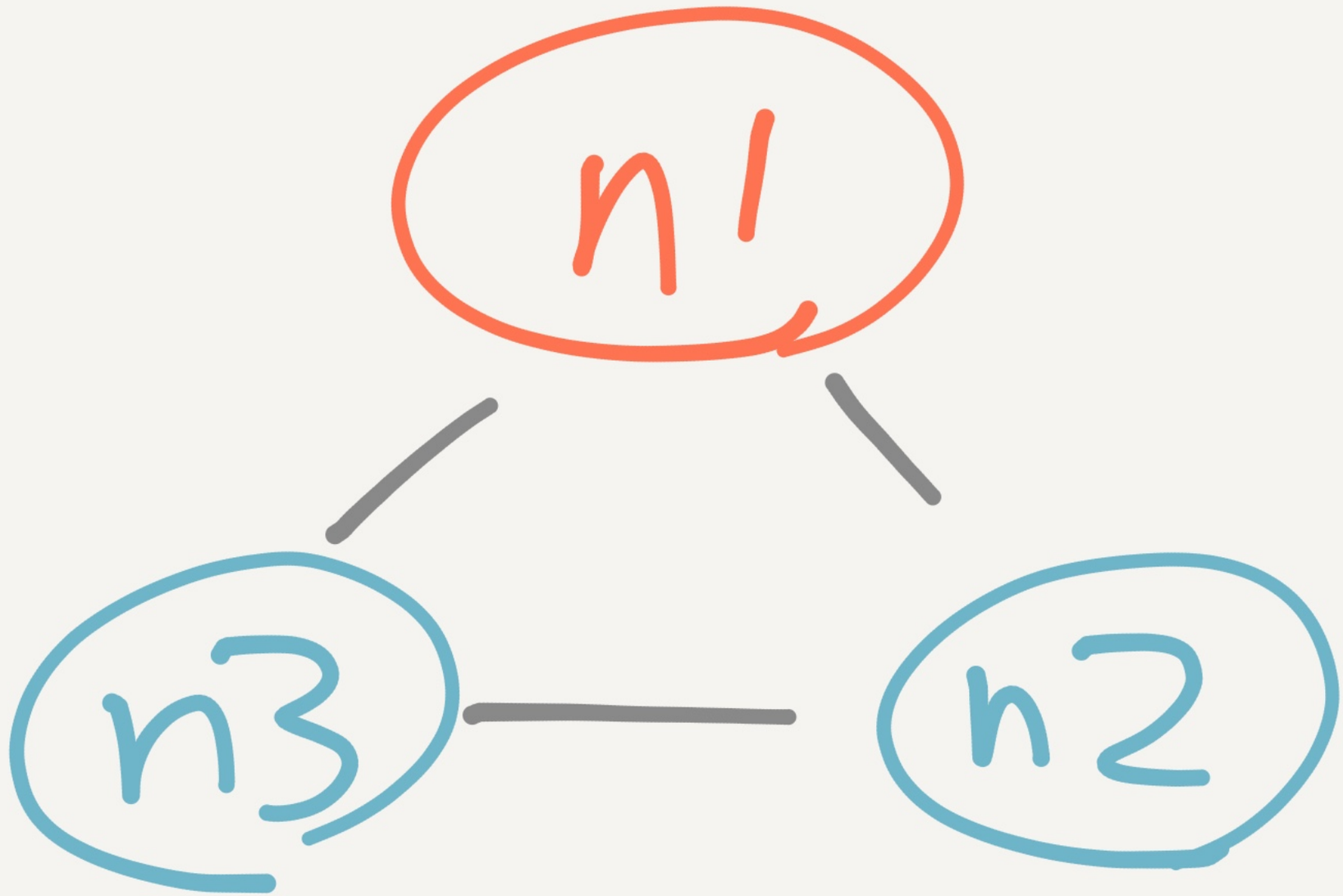

n1
n3
n2


n1


n3
n2


n1
n3
n2


n1
n3
n2


n!
n3
n2

ok

lost

322 + 93 } acked

4525

attempted inserts

v0 protocol is fundamentally broken :(

Protocol v1

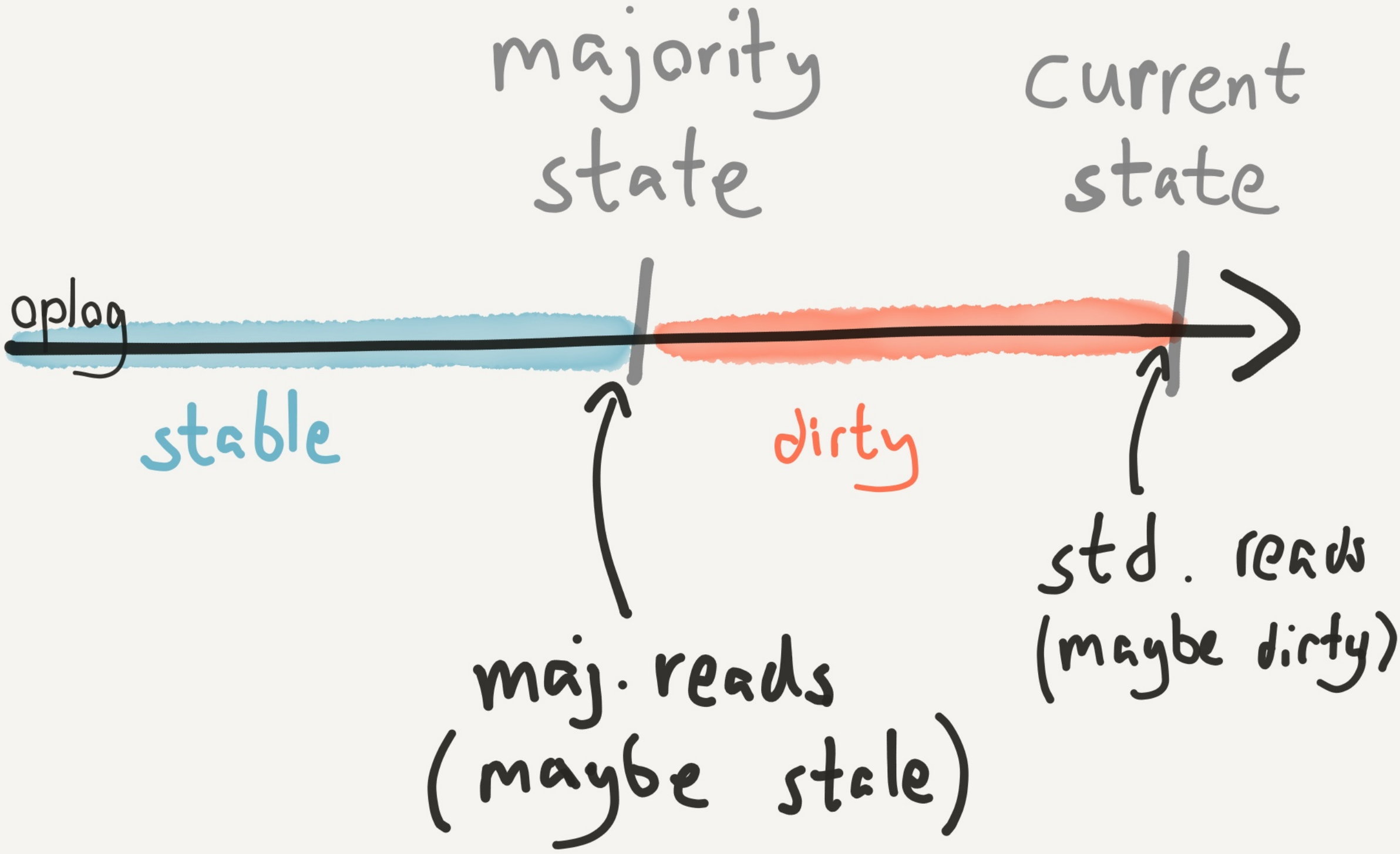
Default in 3.2 for new replsets

Optime

[term, time]

logical clock

old optime


majority state
current state
oplog
stable
dirty
maj. reads
(maybe stale)
std. reads
(maybe dirty)

majority state

current state

oplog

stable

dirty

lin read

majority state

oplog

stable

lin read ✓

ok

lost

515 + 417 } acked

6078

attempted inserts

Server-27053

primaries could ack writes from prior terms

Heartbeats received by ontdated primary could advance commit index

3.4.0-rc4

Server-27149

Secondaries ignored terms & took ops from old leaders

LEGACY

Fixed in

3.2.12

3.4.0

3.5.1

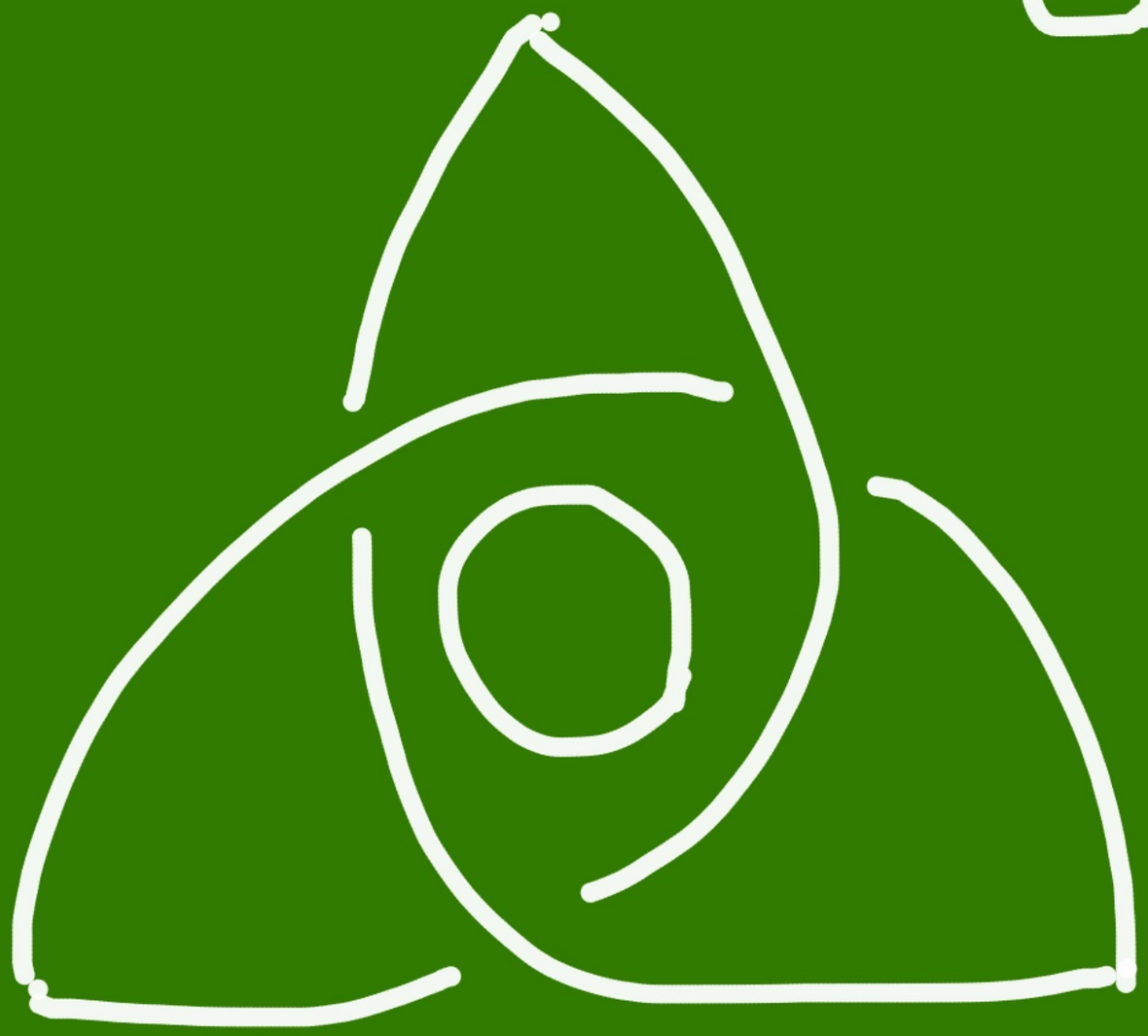
Recommendations

- Get off v0
- Don't use arbiters
- Upgrade to 3.4.x

Use Write Concern MAJORITY

Read Concern MAJ/LIN

0.10.2

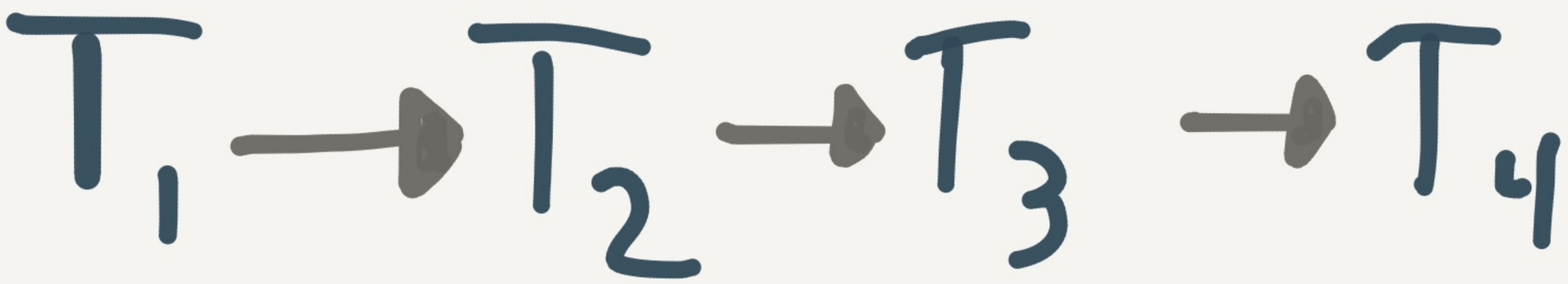


Tendermint

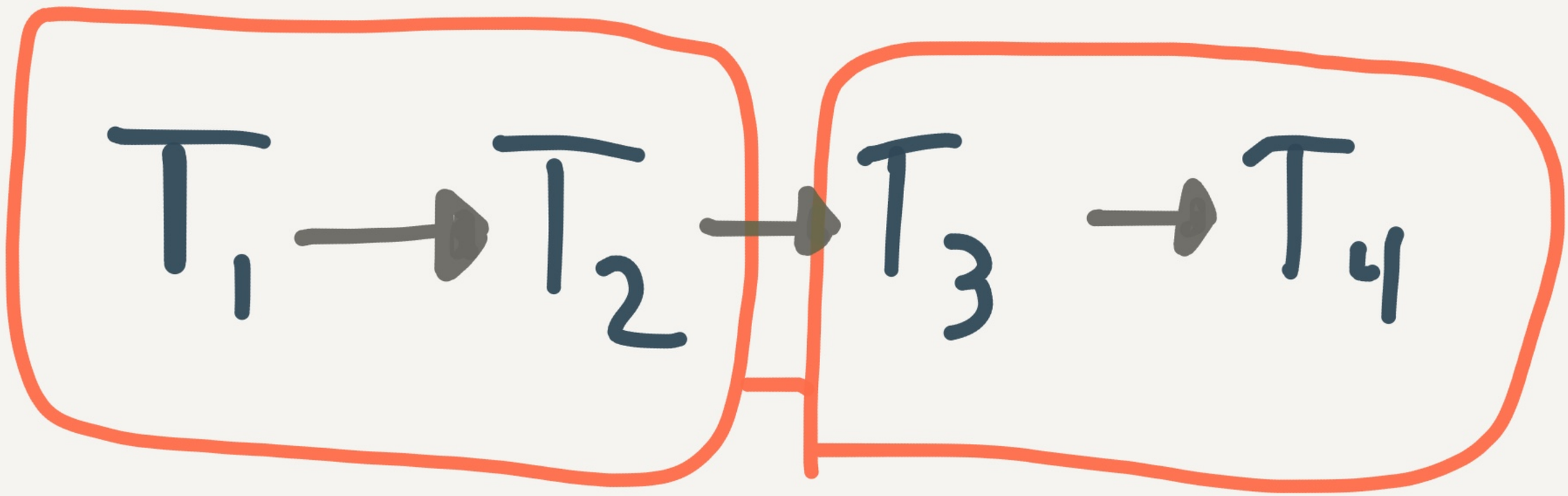
- Transaction replication
- Pluggable FSMs
- Byzantine fault tolerant

"Raft Meets Blockchain"

Like Raft, provides a linearizable order of txns



Like Raft, provides a linearizable order of txns



blockchain!

Byzantine Fault Tolerant!

shun!

shun!

shun!

shun!

Pluggable State Machines

validator

validator

state

state

merkle
eyes

cas-register

- read(2)
- write(5)
- cas(1,4)

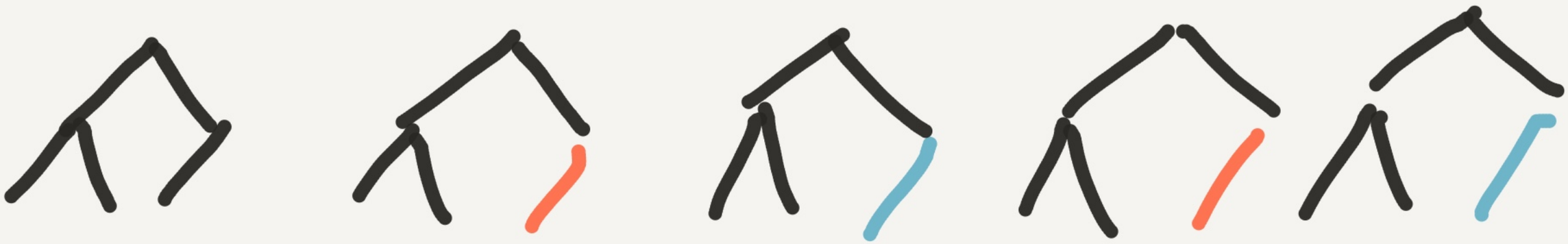
set

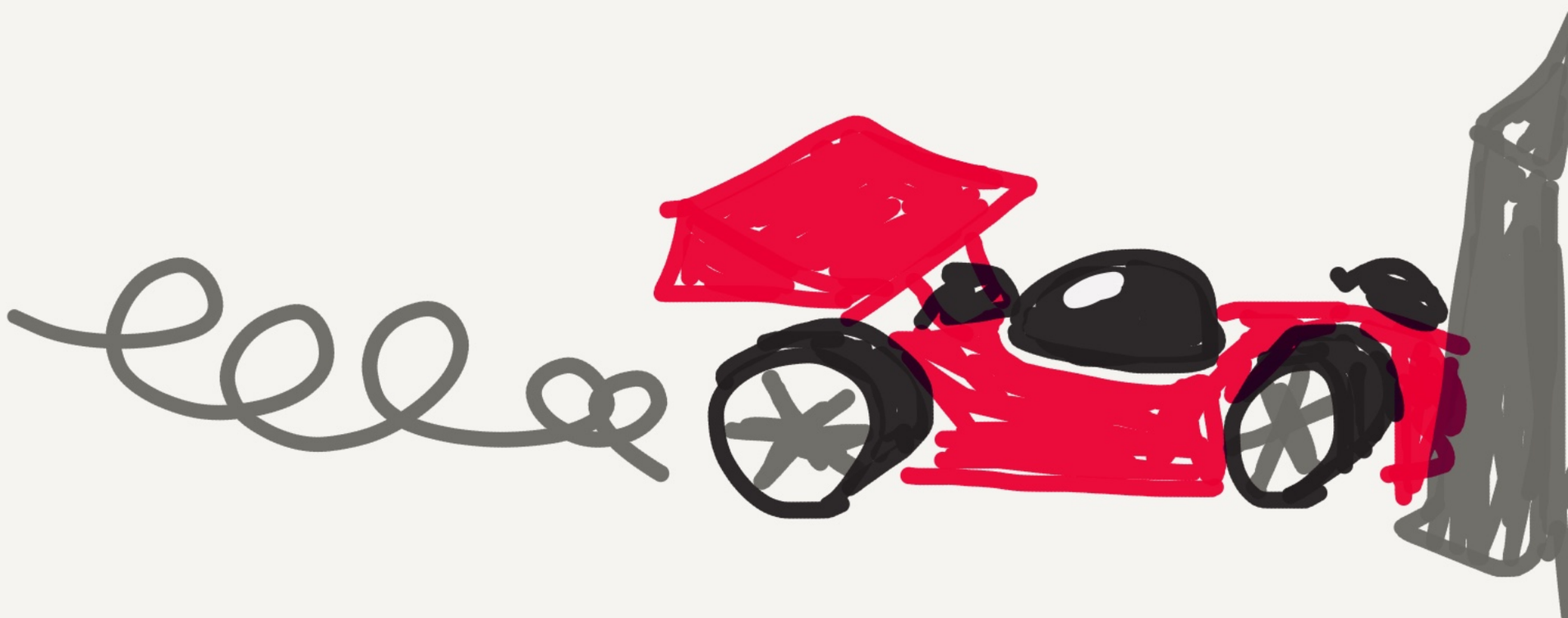
- add(6)

  [read({1,2,3})
  cas({1,2,3}, {1,2,3,6})]

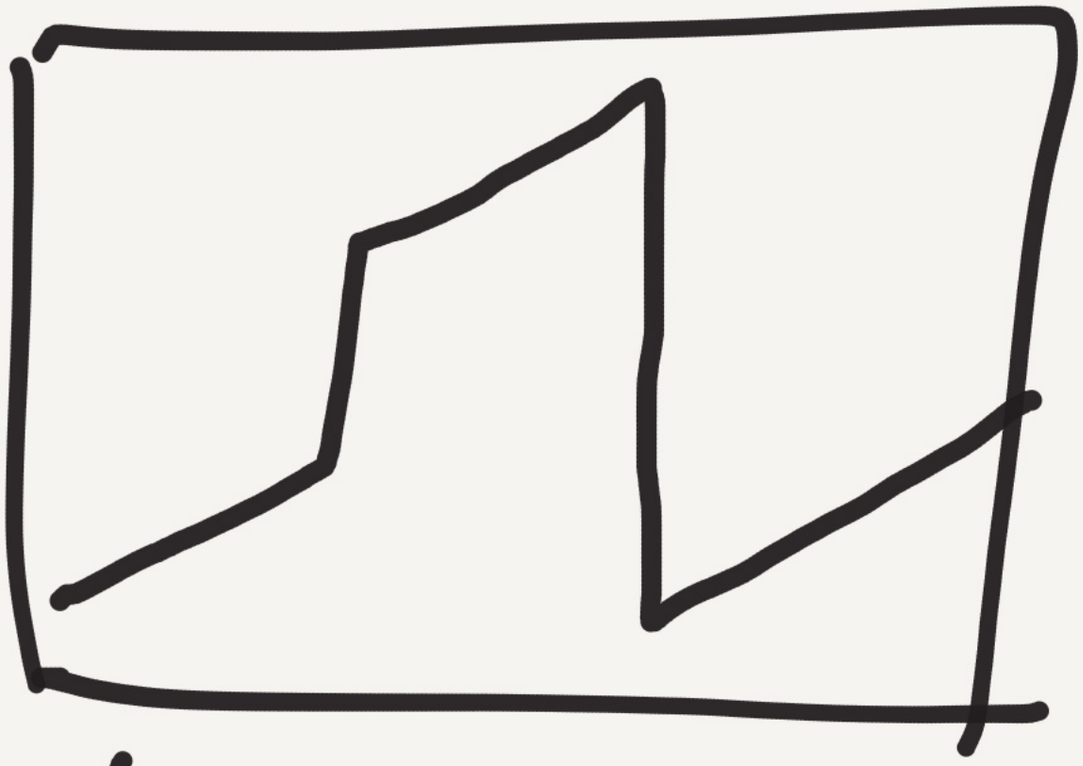
- read({1,2,3, ...})

Merkleeyes Race

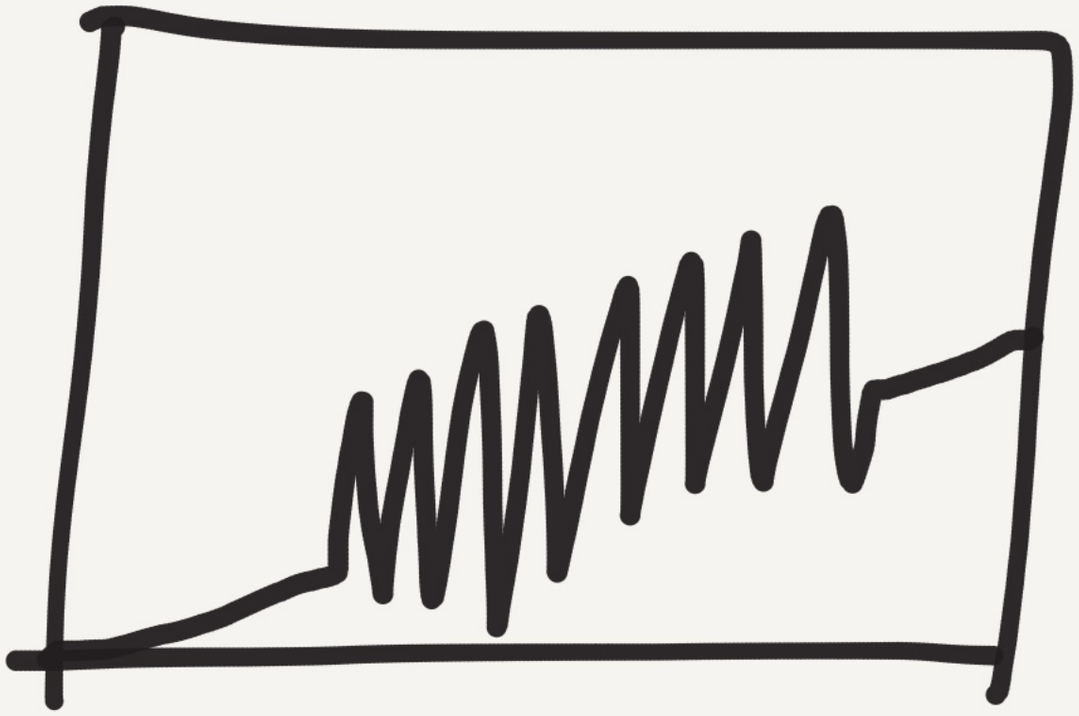




clock skew

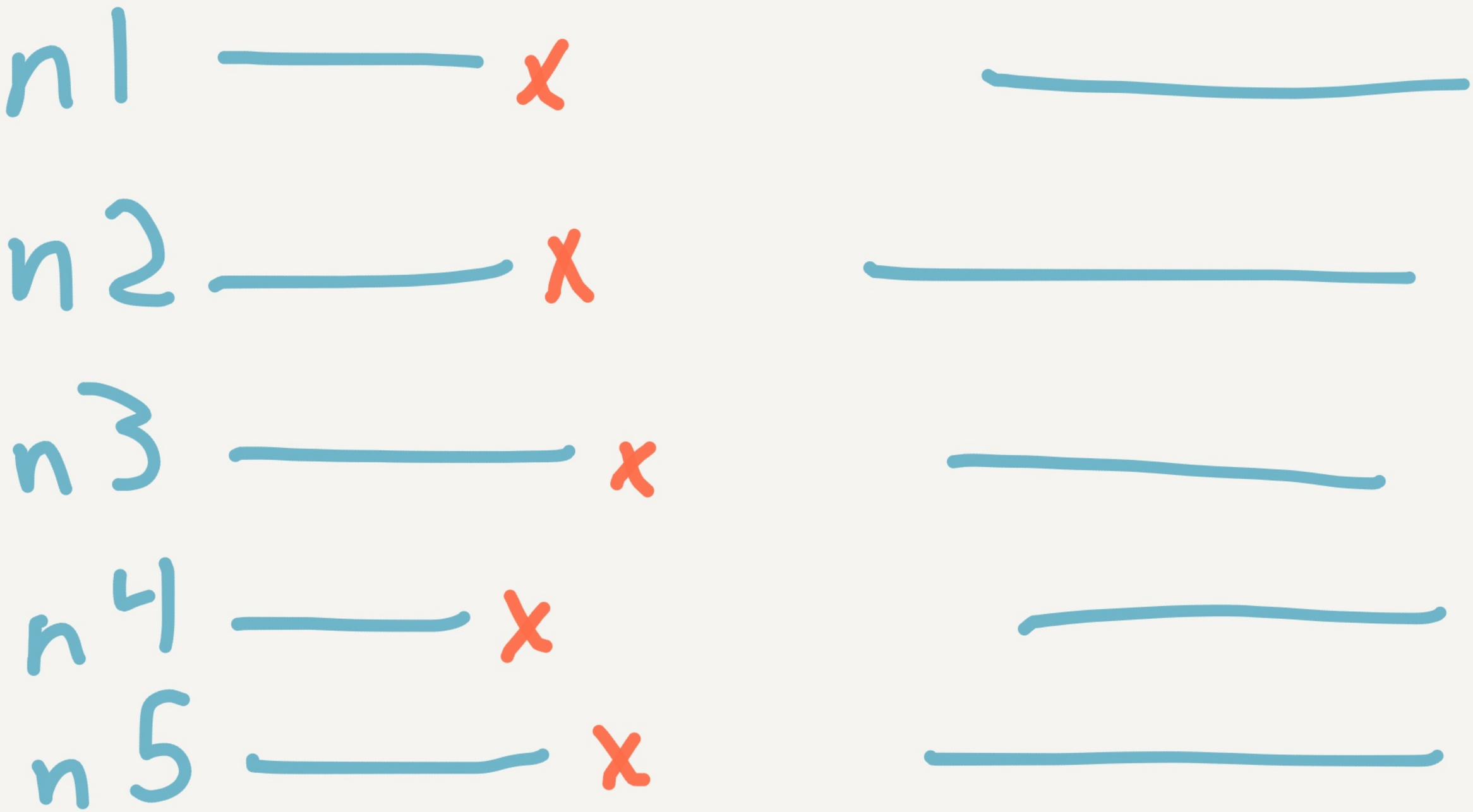


bump

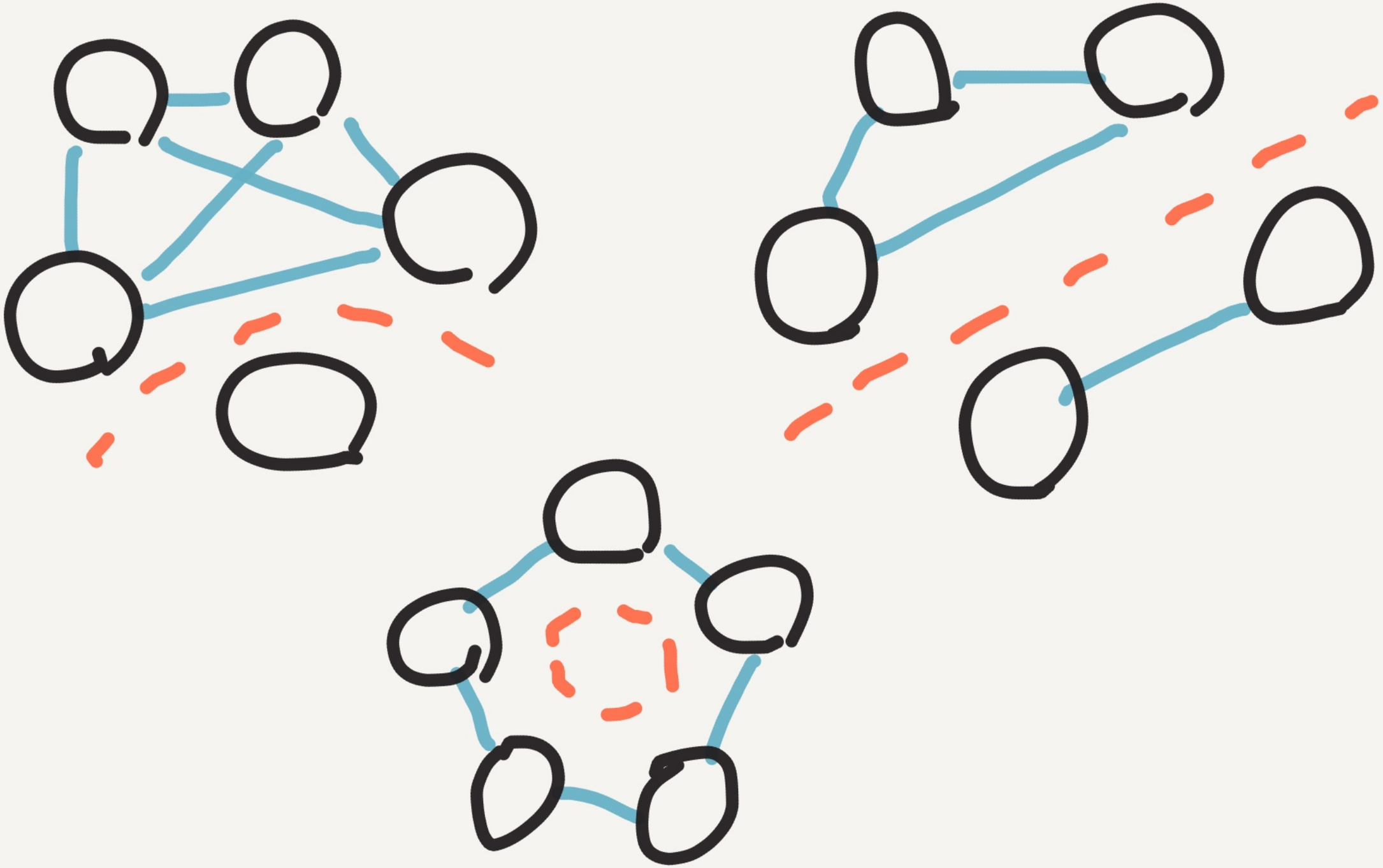


strobe

crashes



partitions



Duplicate Validators

$7A\ B^2$

$7A^1\ C^2$

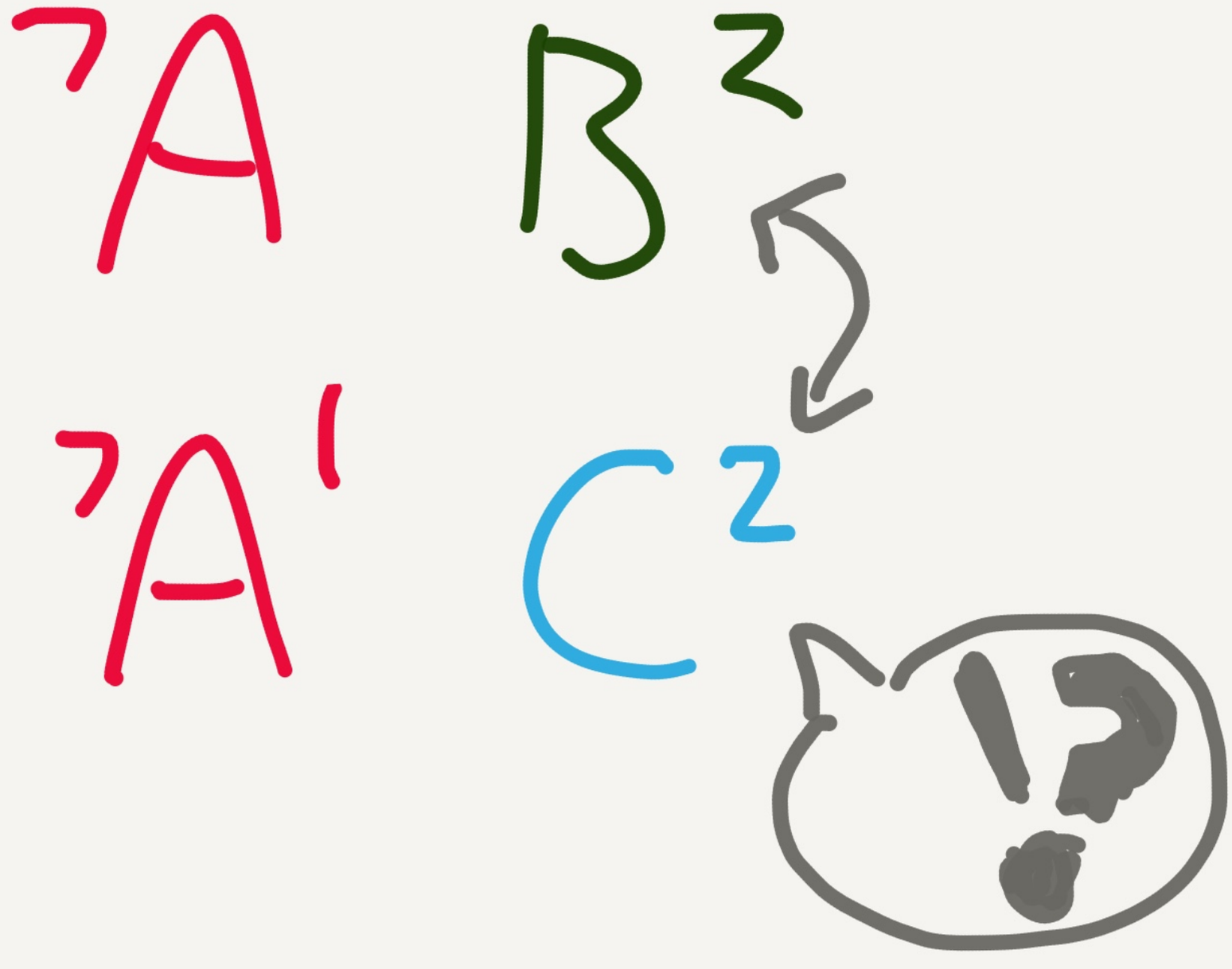
$7 + 2 + 2 = 11$

Duplicate Validators

$^{7}A\ B^{2}\qquad 9/11$

$^{7}A^{1}\ C^{2}\qquad 9/11$

Duplicate Validators



Lost Documents

iff

$\geq \frac{1}{3}$

of votes

file truncation

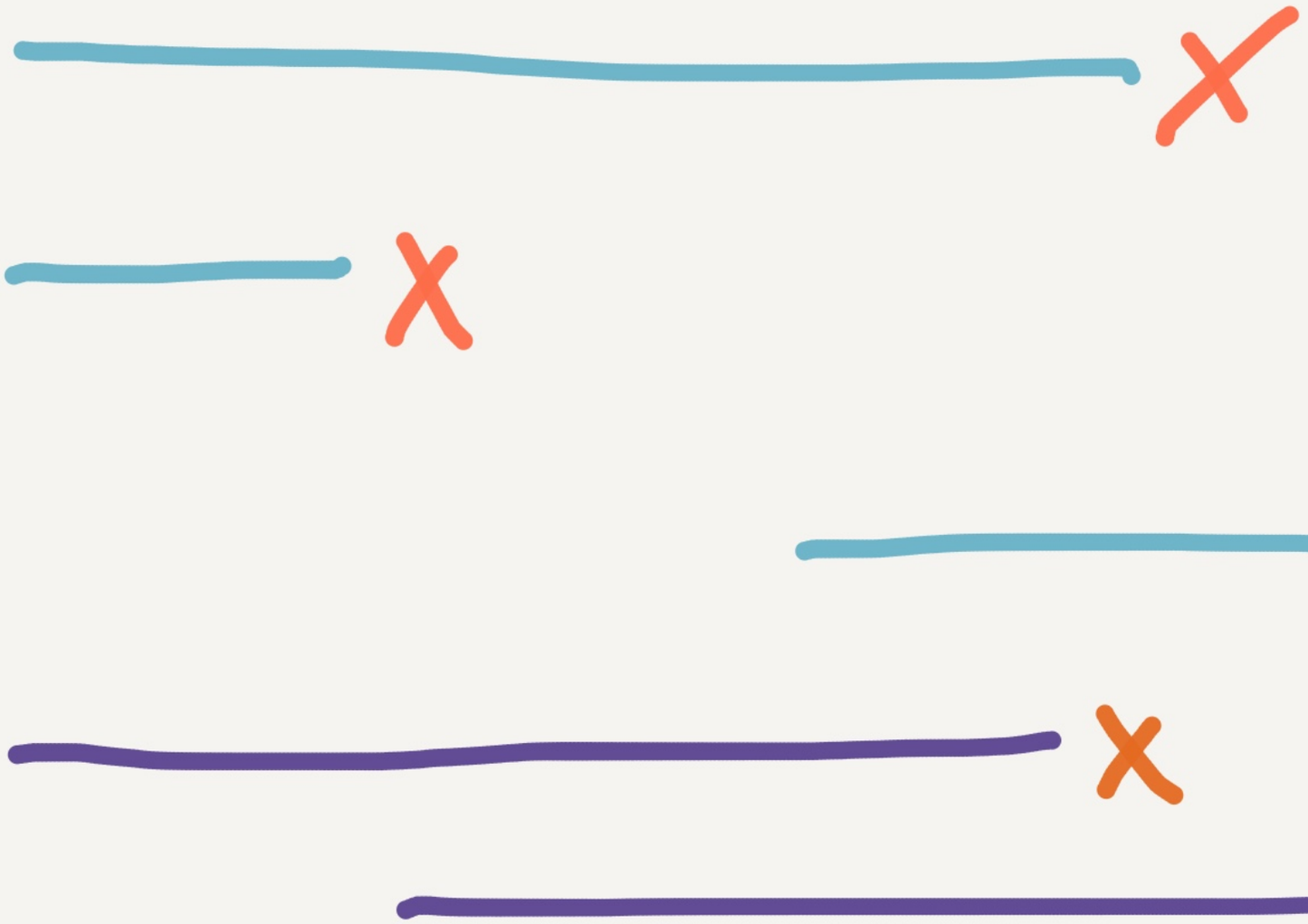
merkleeyes crash!

(goleveldb bug)

Tendermint

Doesn't fsync to disk!

Dynamic Reconfig



Data corruption & crashes are to be expected — Tendermint is still in beta

Team working to fix these issnes

Recap

Read the docs!

CAREFULLY

Then test it

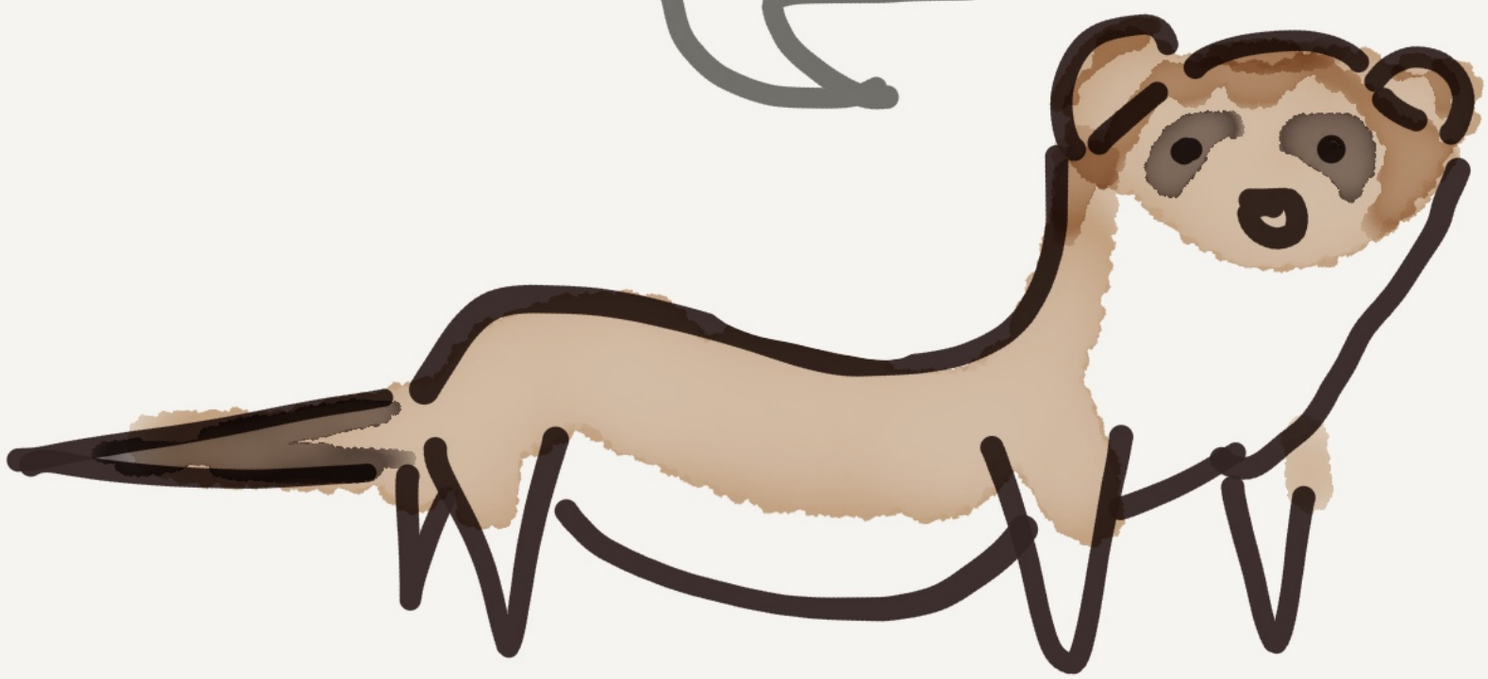
— for —

YOURSELF

"Strict"

"Strong"

"ACID"



Be Formal

&

Be Specific!

Figure out the invariants your system needs

Consider your failure modes

Process Crash

#kill -9 1234

Node failure

- AWS terminate
- Physical power switch

Clock Skew

```
# date 10280000
# faketime ...
```

SC/IO Pause

```
# killall -s STOP foo
# killall -s CONT foo
```

Network Partition

```
# iptables -j DROP
```

```
# tc qdisc ... delay ...
```

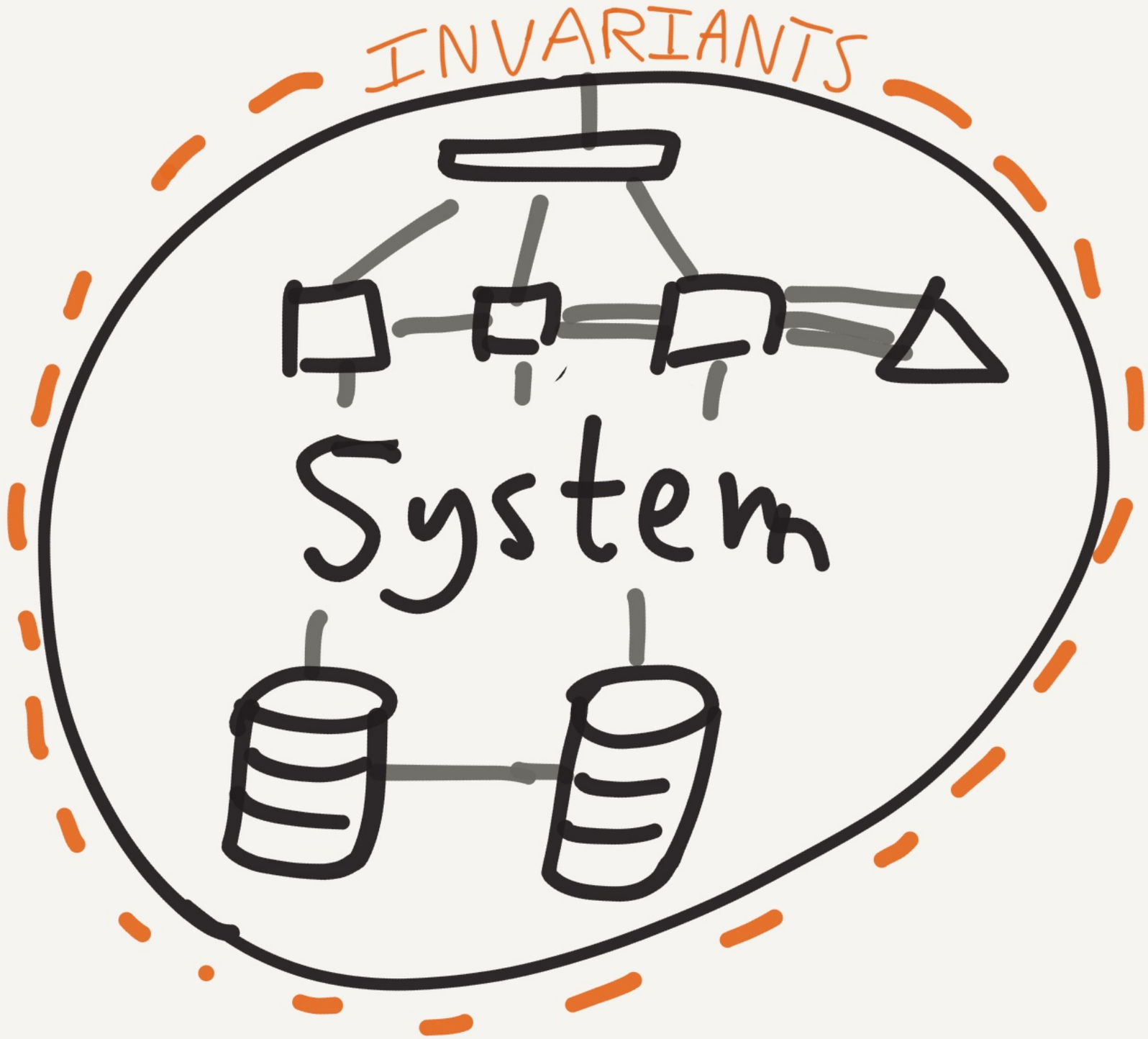
```
drop ...
```

Test your

Systems

end ——— to ——→ end

vs.



If you look for

perfection

you will never be

content

- Property testing
- High-level invariants
- With distsys failure modes

Thanks

Cockroach Labs
Mongo DB
All in Bits
Peter Alvaro

} Funded Research

http://jepsen.io