

All happy databases
are alike; each
unhappy database is
unhappy in its own

way - Leo Tolstoy?

Anna

Concurenha





Kyle Kingsbury

@aphyr



I break
databases!

Public
API {



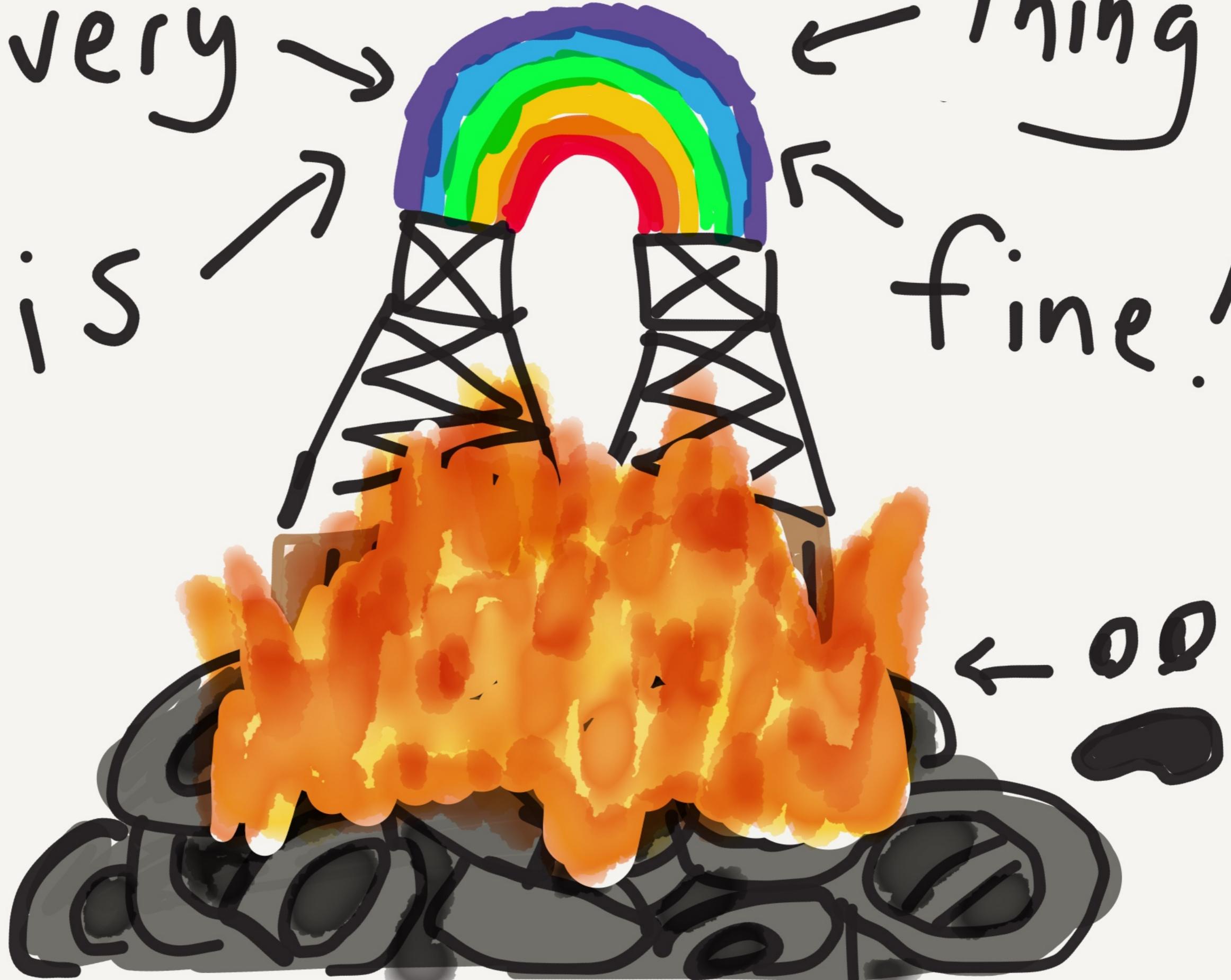
} API code

Ruby {

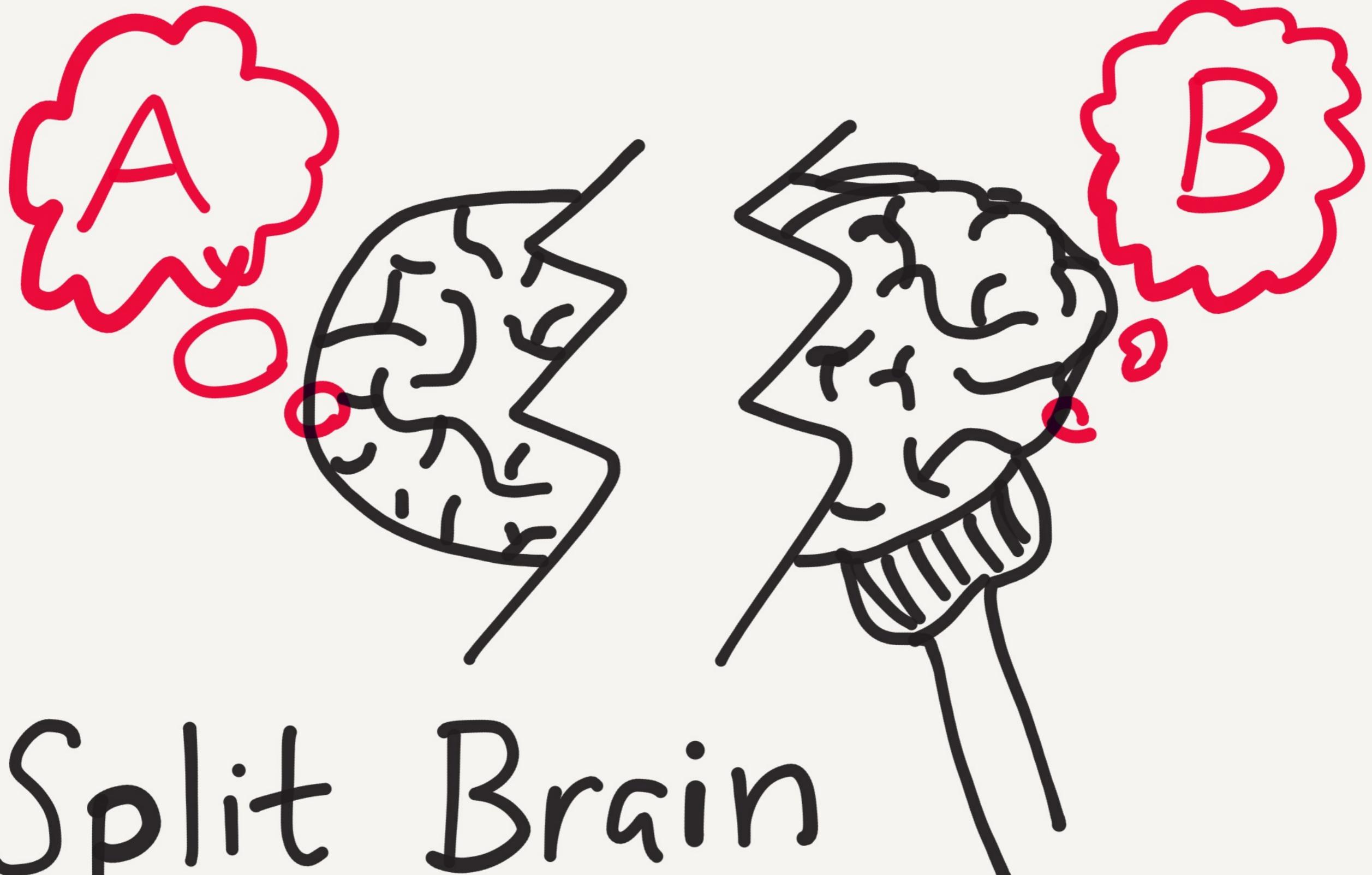


DBs

Every →
is →
← Thing
fine !

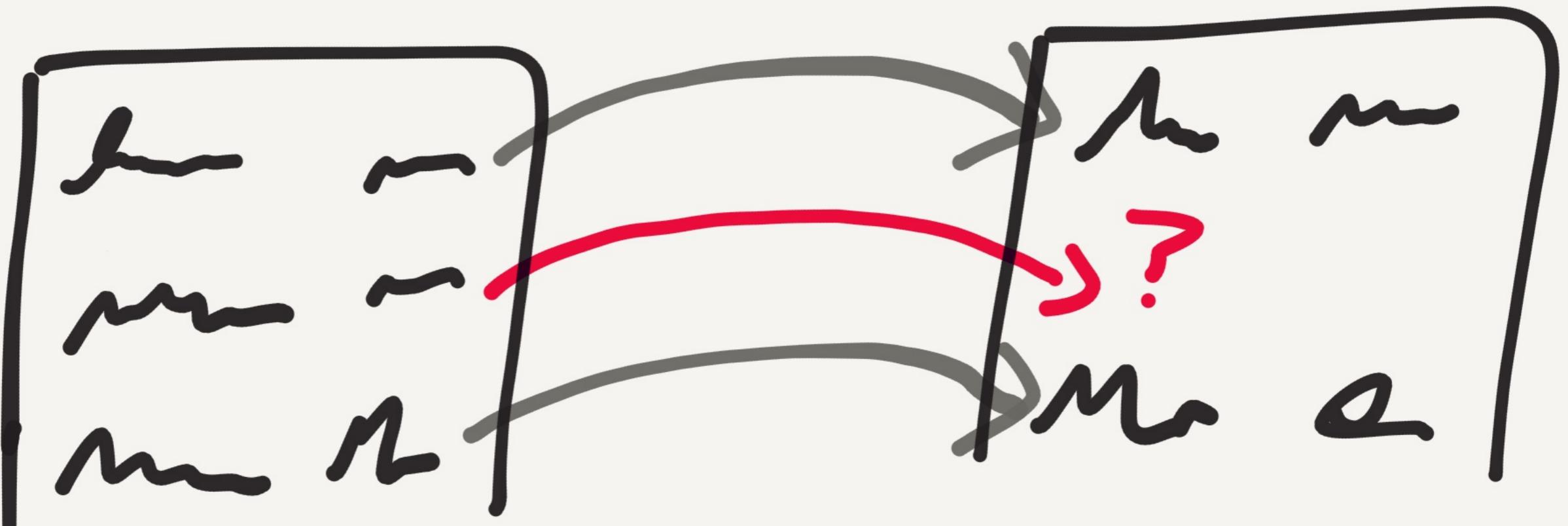


Databases! /
Queues! /
Discovery! /
THE HORROR

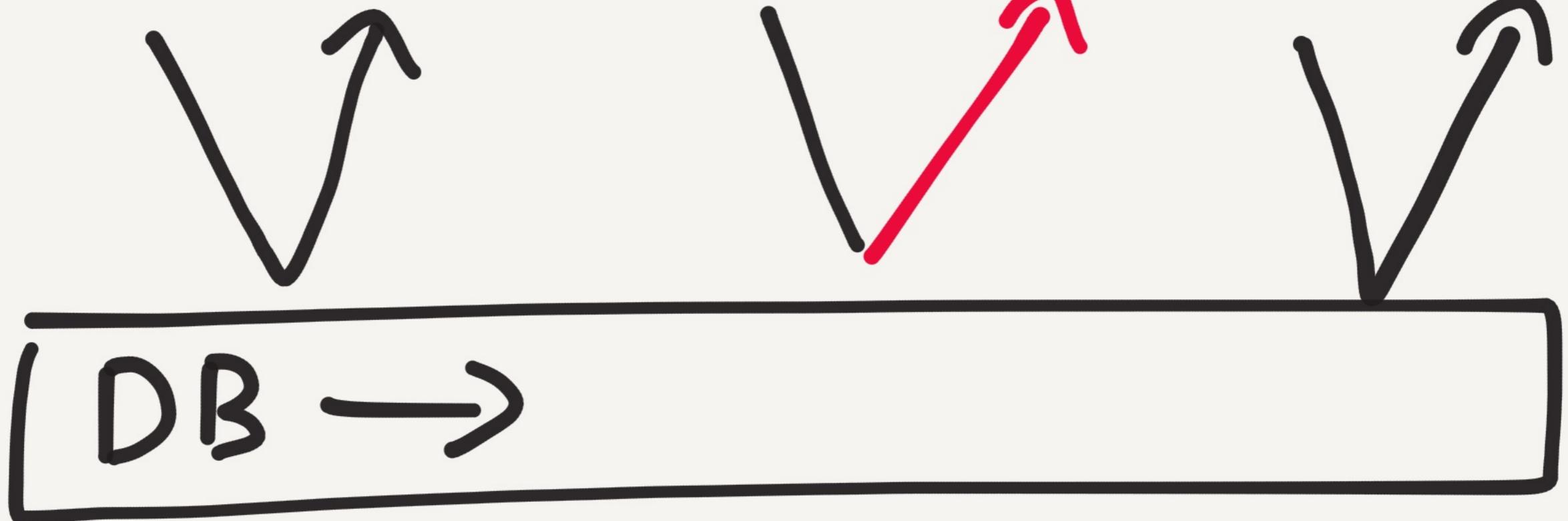


Split Brain

Broken Foreign Keys



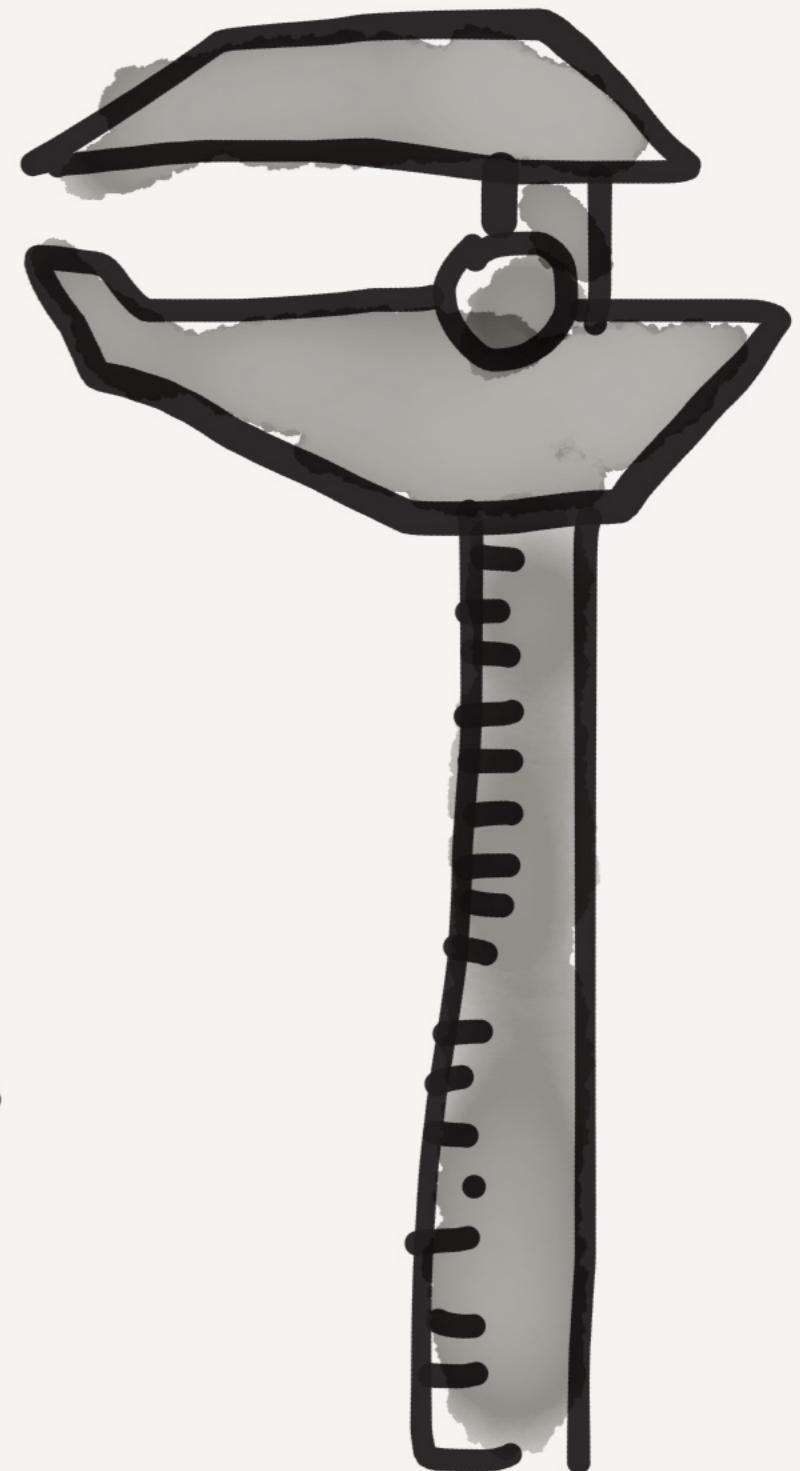
write ok read \emptyset read ok



Anomalies

How do you
know if a
system is safe?

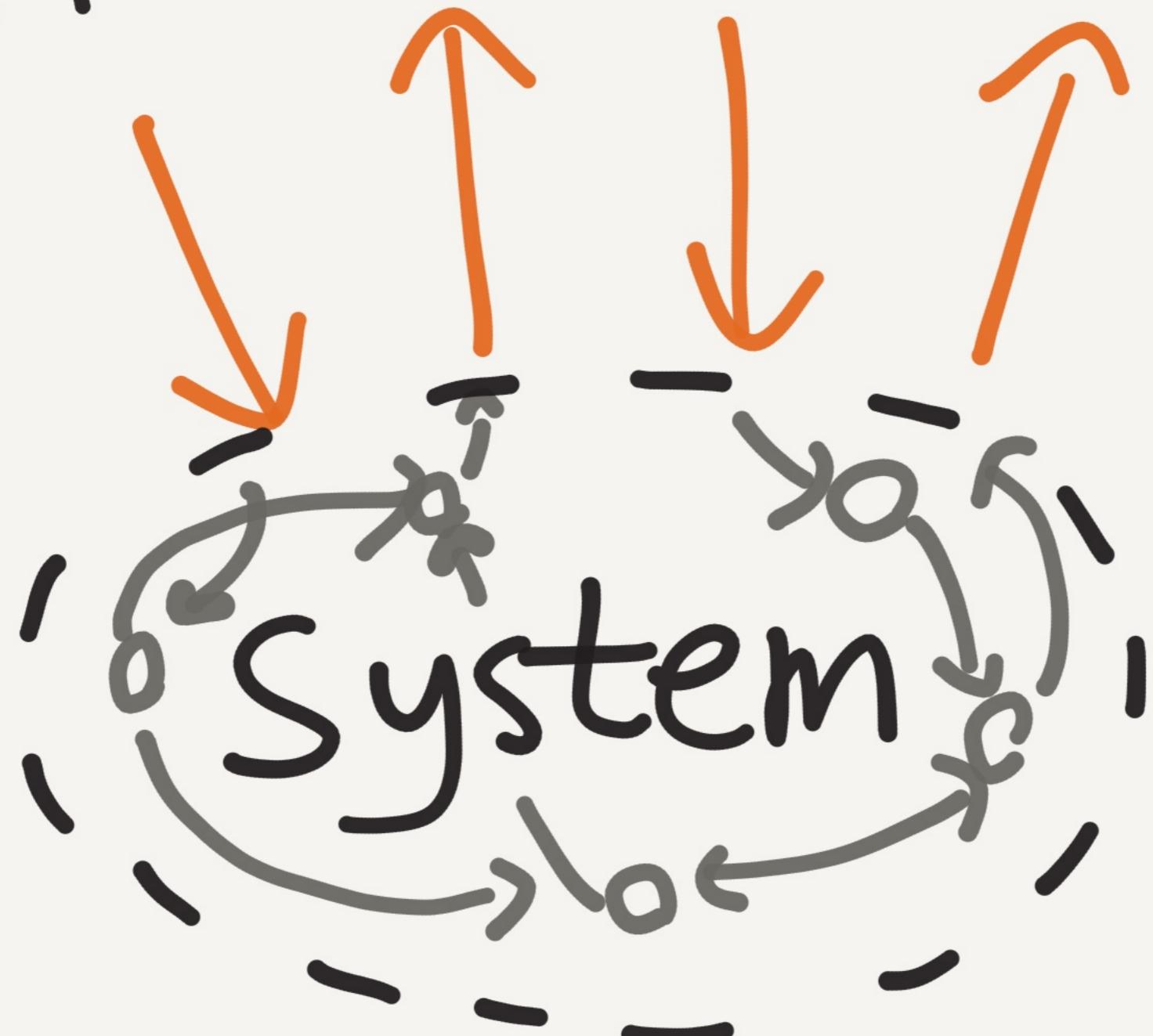
Measure
your
Systems



Jepsen

github.com/aphyr/jepsen

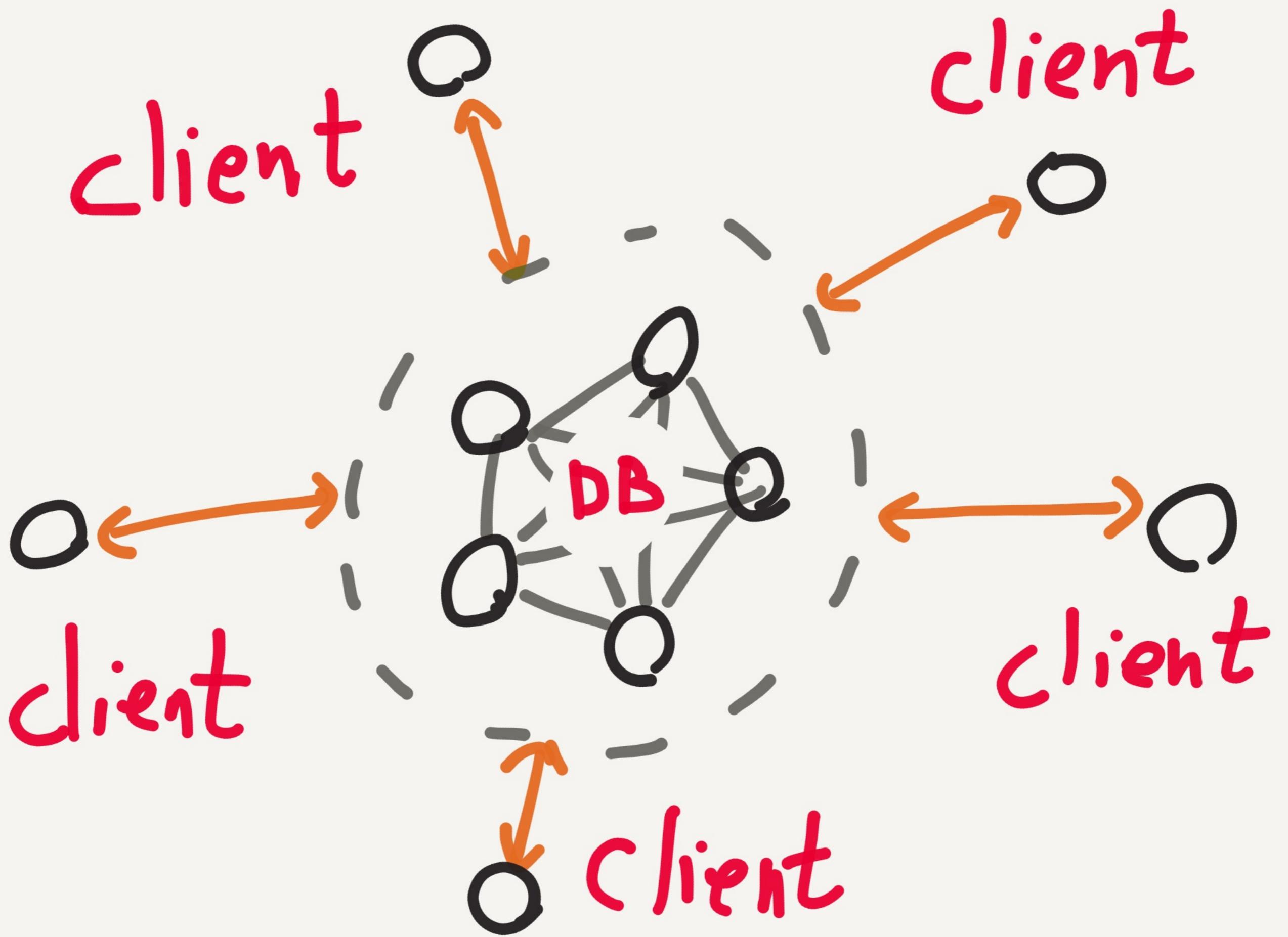
Environment





— INVARIANTS —





client: $w-w'$ $r-r'$ $w-w'$

The diagram illustrates three client requests to a central database system. Each request is shown as a horizontal line segment with arrows at both ends, indicating a transaction or query. The first request is labeled $w-w'$, the second $r-r'$, and the third $w-w'$. All three requests converge on a central, blurred area representing the database.

DB :

client : w — w' w — ... -

The diagram shows the internal state of the database after the client requests have been processed. The database is represented by a central cloud-like shape. Inside, there are two nodes labeled w and w' . A horizontal line connects them. A vertical arrow points downwards from the center of this line to a dashed line below, representing a log or history of operations.

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



invoke

ok

invoke

fail

invoke ?

? info

?

?

?

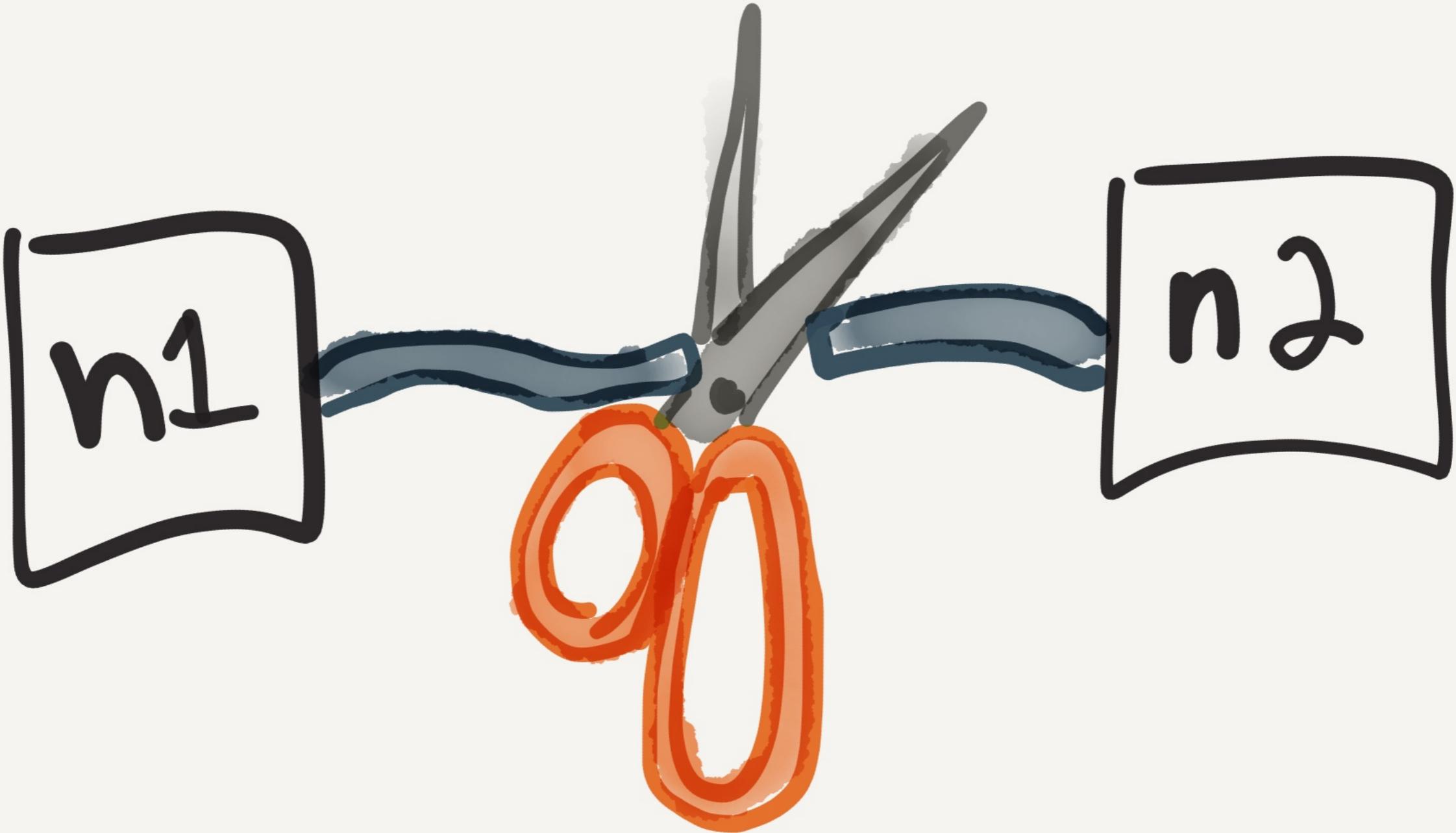
invoke ok

invoke fail

invoke ok

invoke info

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



Partitions!

So, what
have you
found?



Riak

LWW \rightarrow lost writes

CRDTs \rightarrow safe

5/13

Mongo

Data loss at all

Write Concerns

Redis Sentinel

5/13

Split brain,
massive write loss

Cassandra

9/13

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

NuoDB

Beat CAP by buffering all requests in IRAM during partition

9/13

Kafka

In-sync Replica Set

Could shrink to 0

nodes, causing msg

loss.

9/13

Zookeeper

Works.

etcd / Consul

6/14

Stale reads

Elastic search

Loses documents in
every class of partition
tested.

6/14

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

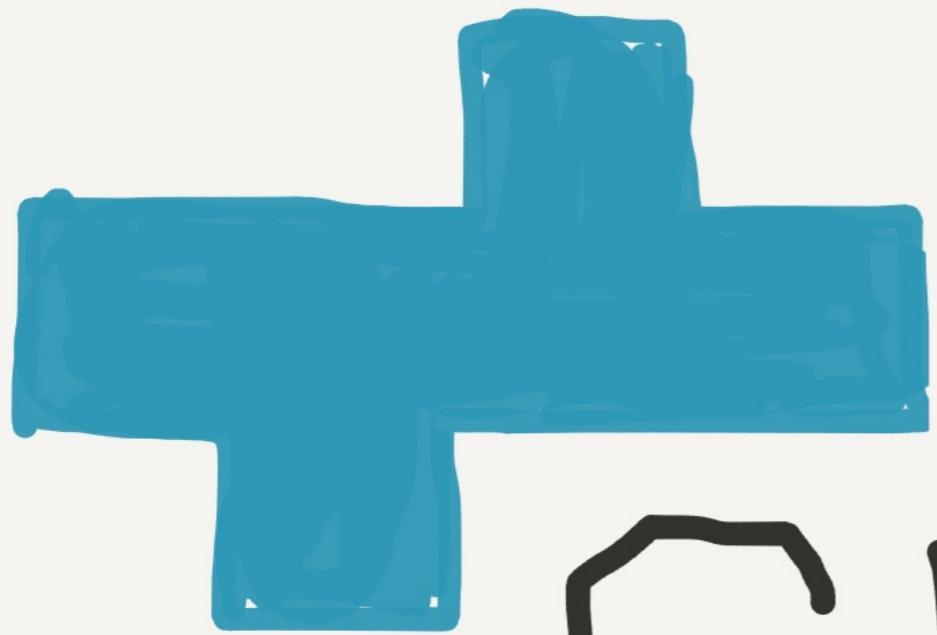
1/16

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

- Stale reads
- Dirty reads
- Lost writes

Be bad, but at
least don't be
a *liar*,
a *deceiver*!





CRATE. | 0

0.54.9

Distributed SQL

DB for IoT,

Analytics, etc

"after having written its record, the record will be consistent and immediately available when selecting this primary key across the cluster."

APP



CREATE

ElasticSearch



ES 1.1.0: extensive
data loss

ES 1.5.0: less extensive
data loss

Crate uses 1.7.

Elasticsearch 1.7

- stale reads
- dirty reads
- lost updates

Ok but how bad
are dirty reads

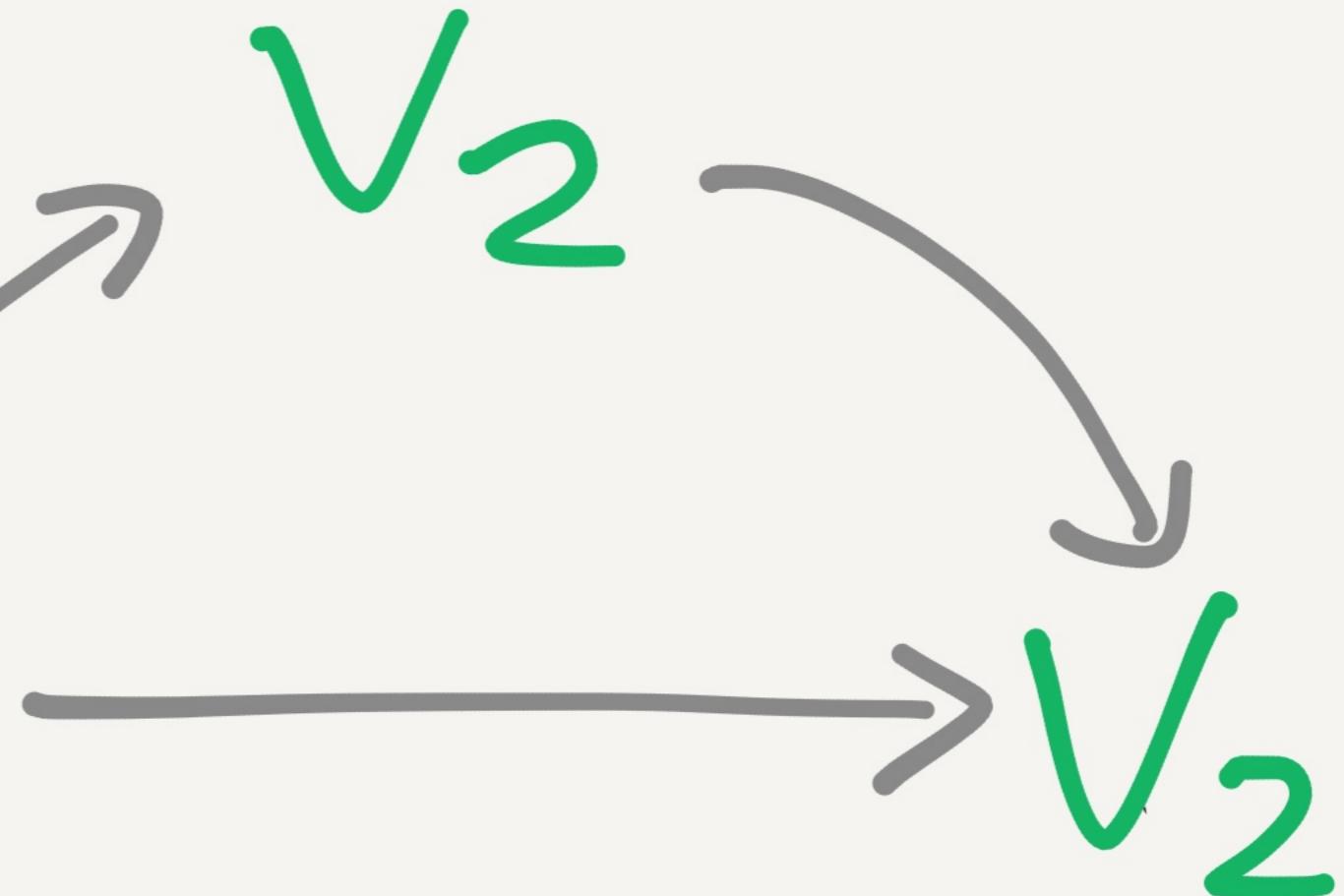
anyway?



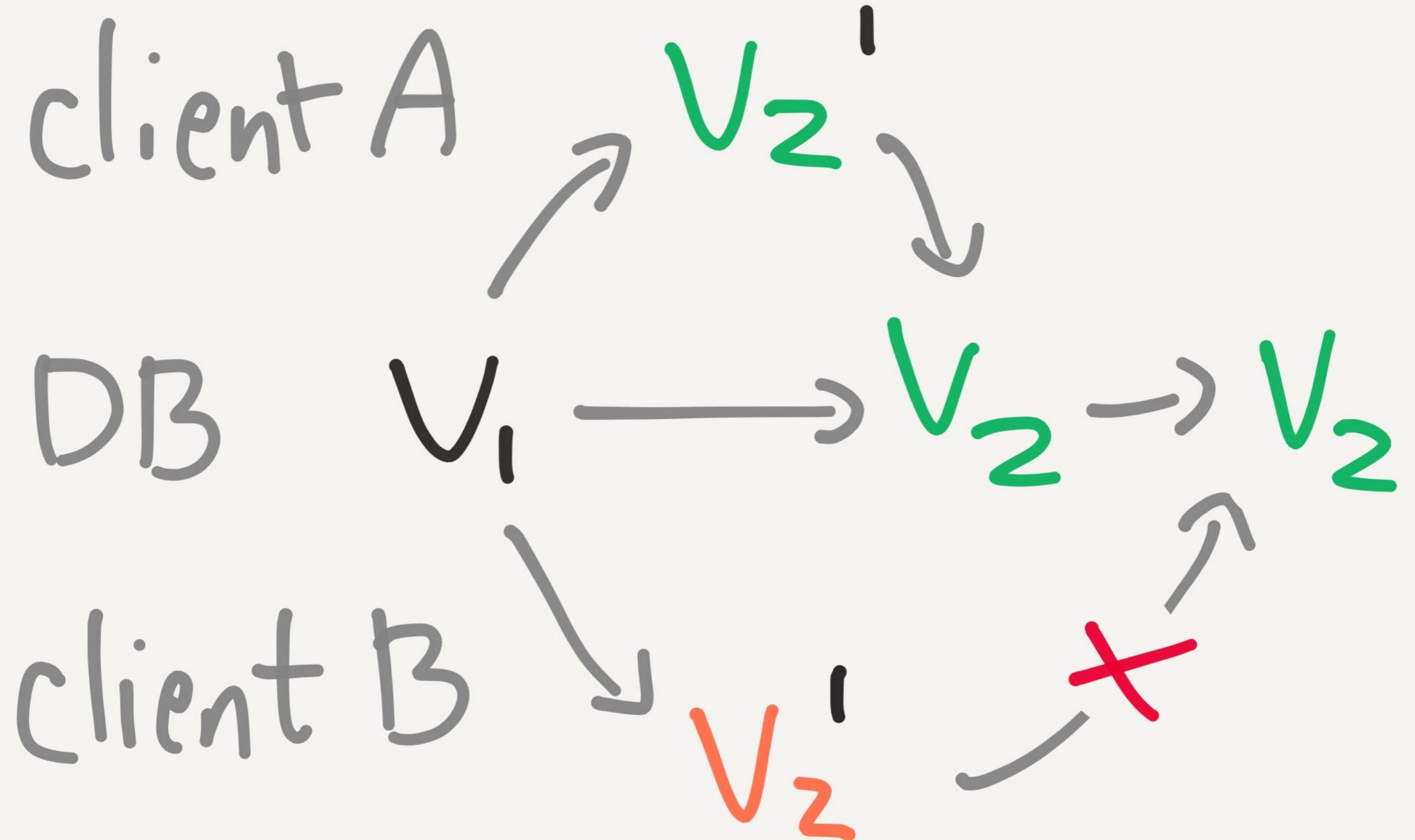
client

DB

v_1



Sequential updates



Are versions
actually
distinct?

an test

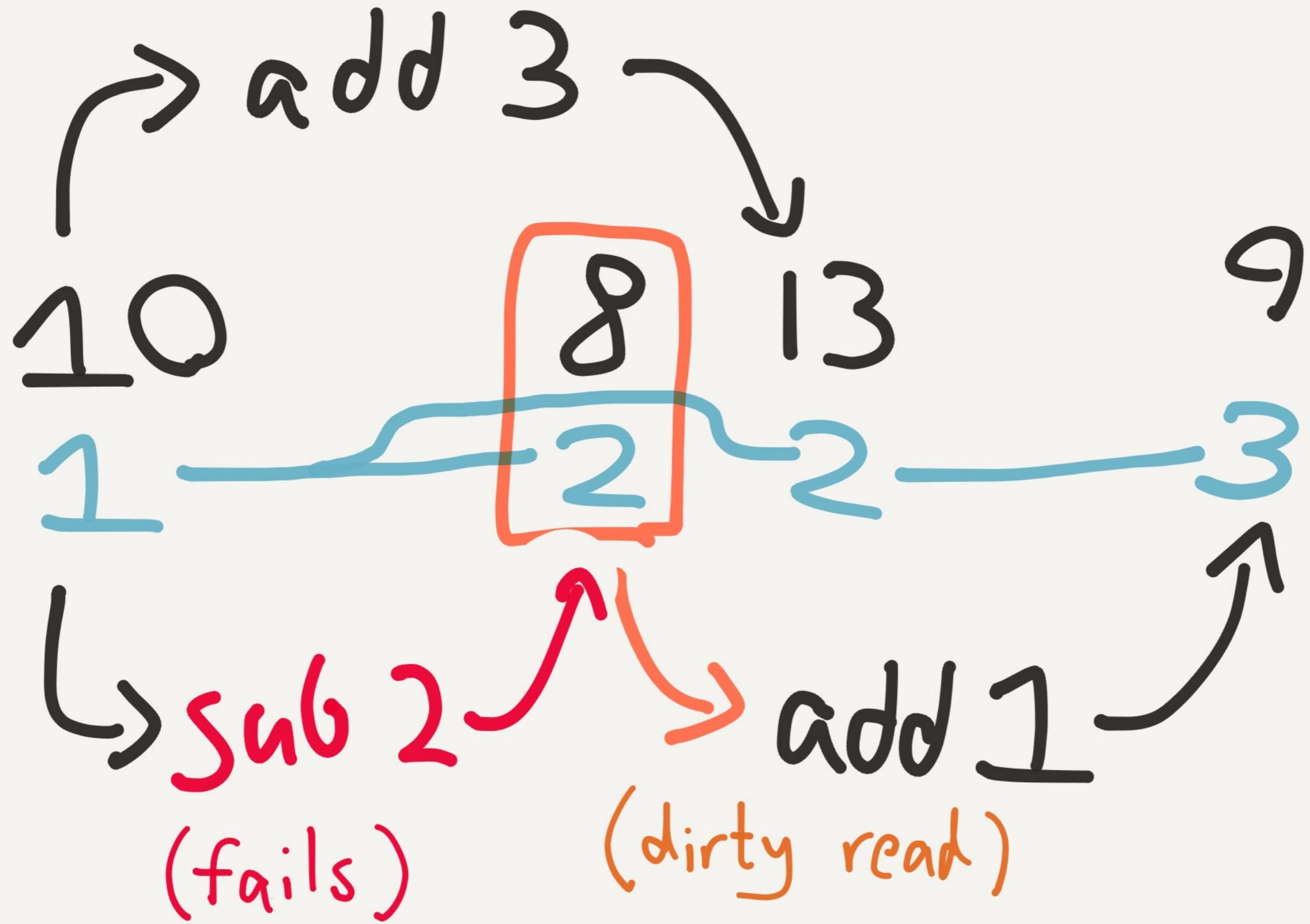
version: 1 2 3 4 5 ...
value: 1 2 3 8 9 ...

- write unique values
- read value, version

{:value 1, :version 2}

{:value 3, :version 2}

value
version



$$10 + 3 + 1 = 14 \neq 9$$

$$10 - 2 + 3 + 1 = 12 \neq 9$$

Preserves garbage - 2

Loses ok update + 3

If you read,
modify, write,
use Read Committed

~~It gets~~
worse!

It turns out
that Crate will
use plain old
inserts, too.

Not Cratesí

fault! It's an

[ES issue ...

What about

using a newer

Elasticsearch?

Elastic closed

#7572 : "network
partitions can cause
... documents to be lost"

Resiliency page:

Lost documents is

fixed in 5.0.0

5.0.0 - alpha 5

- dirty reads
- lost updates
- replica divergence

#20031

Dirty reads :

no plan to fix

⇒ no safe updates

Lost updates:

#20384: can promote
stale primaries. Partial
fix in 5.0.0. Full: 6.0?

Replica Divergence

Unacked docs aren't resynchronized until the next replica recovery

still unaddressed

as of Dec 2016

(5.0.2)

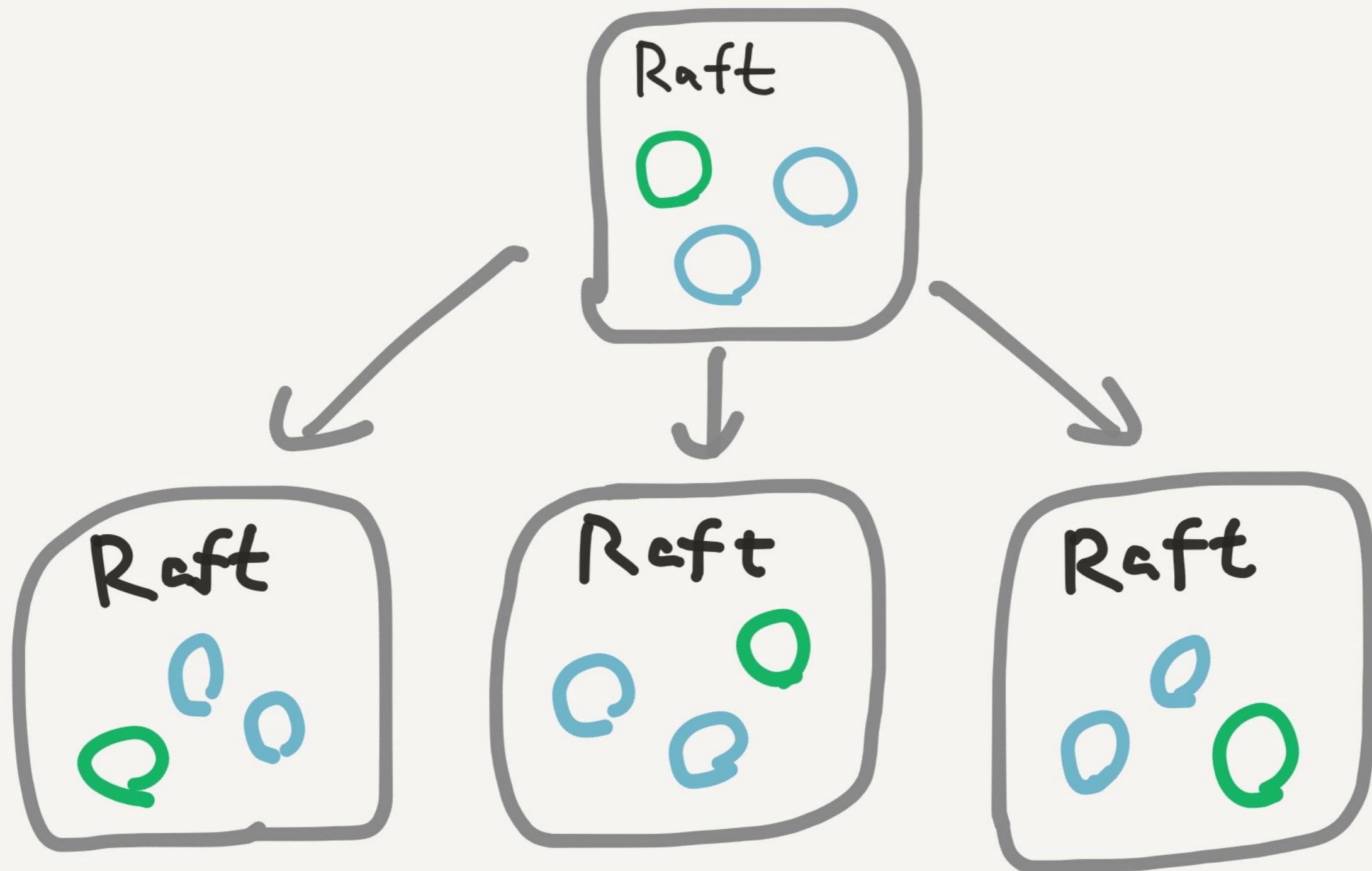
Recommendations

- Don't use ES as a system of record.
- Don't make up your own consensus algo.

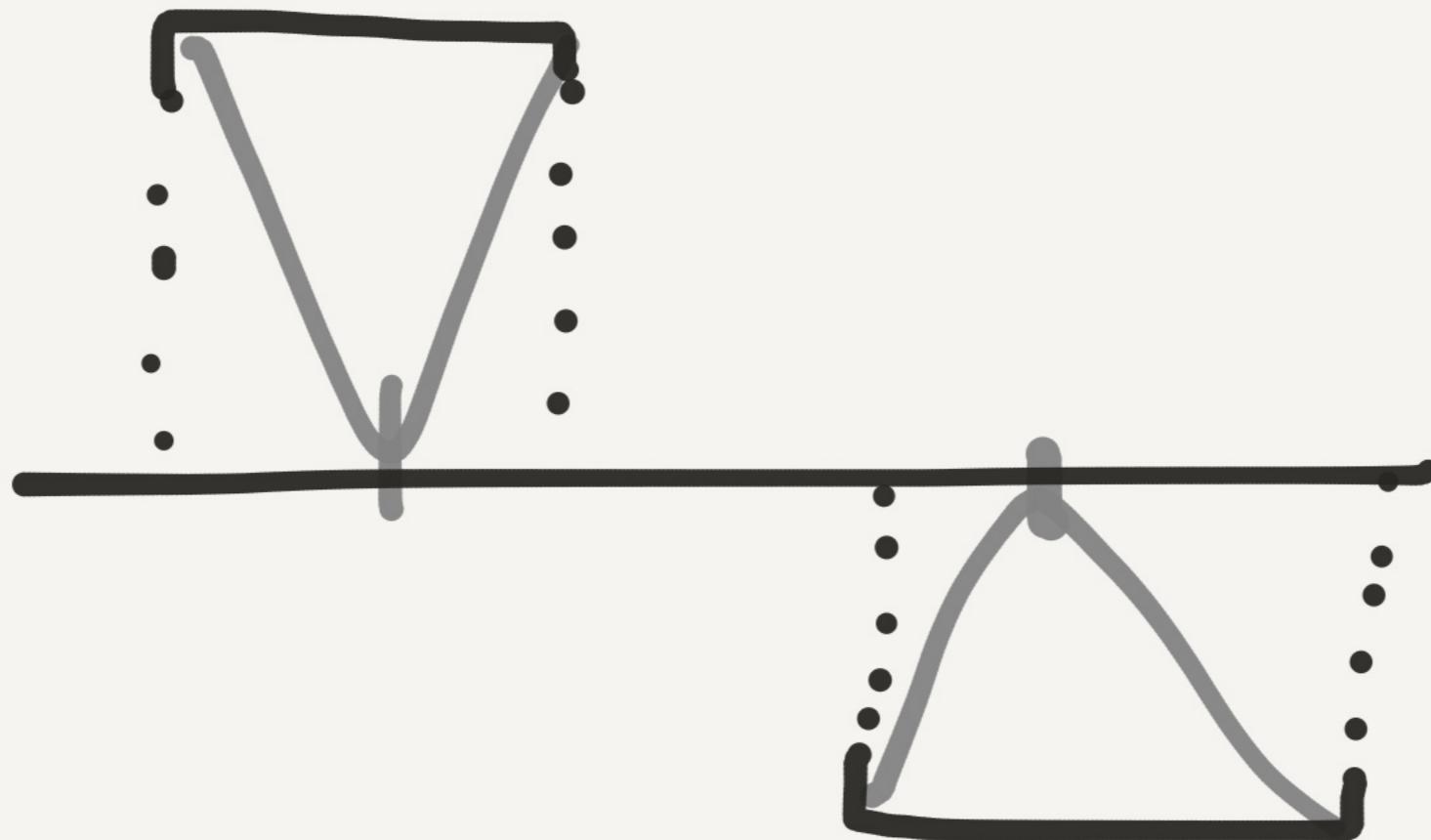
Crate & ES are
well suited for
machine data where
loss is OK!



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



Txns in a Raft
cluster are linearizable



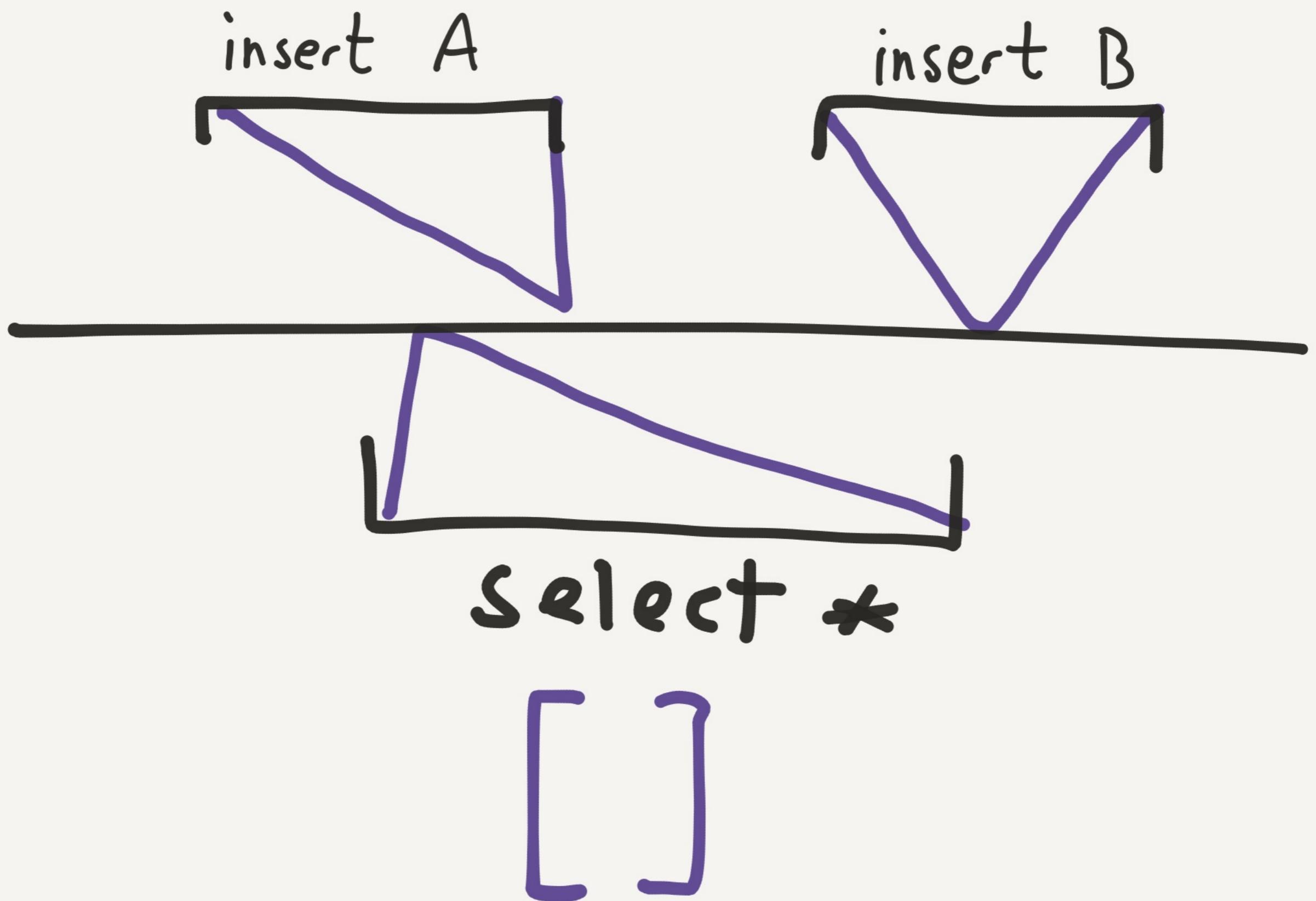
Cockroach cheats on
reads with time-based
leader leases

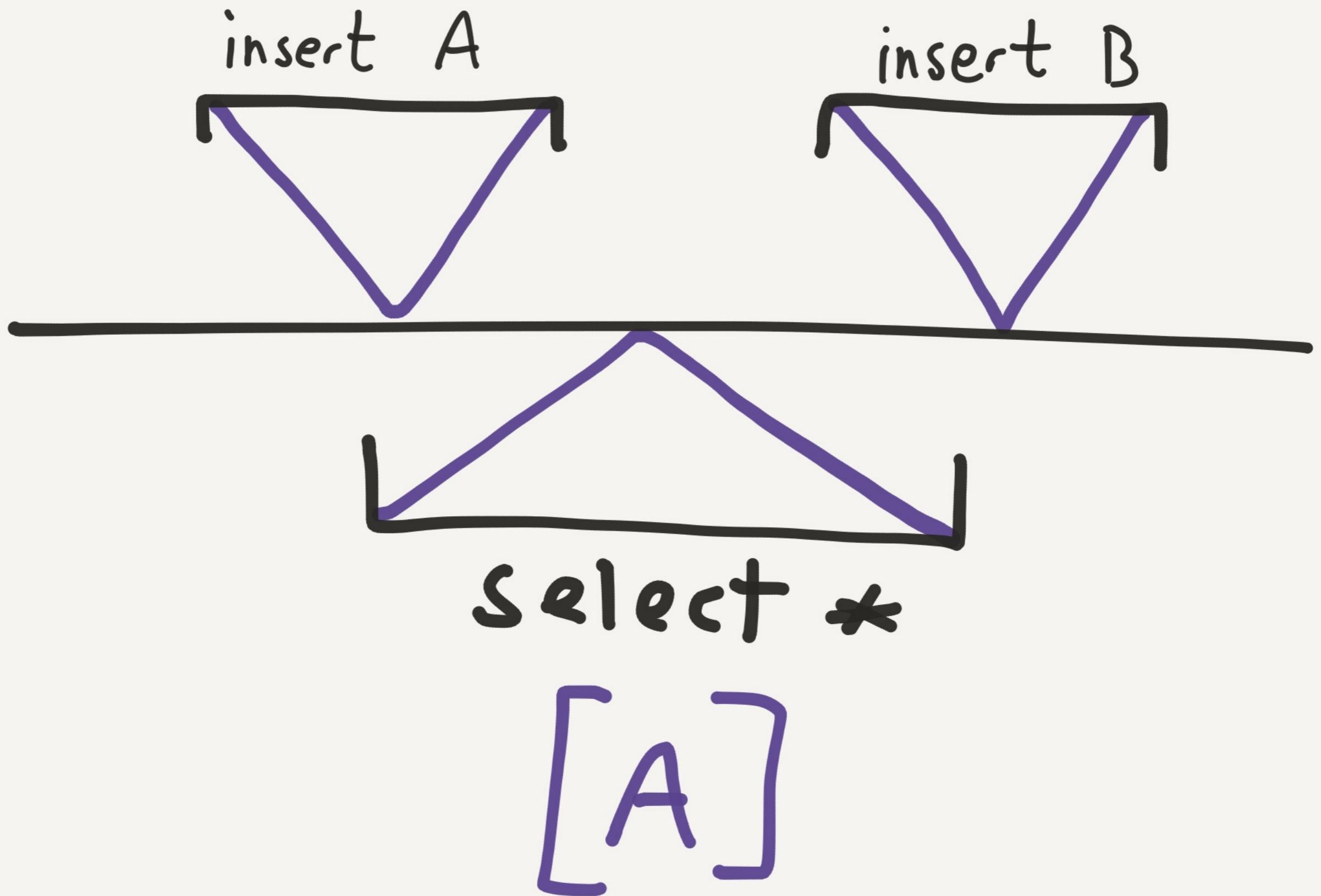
⇒ Needs reliable timeouts

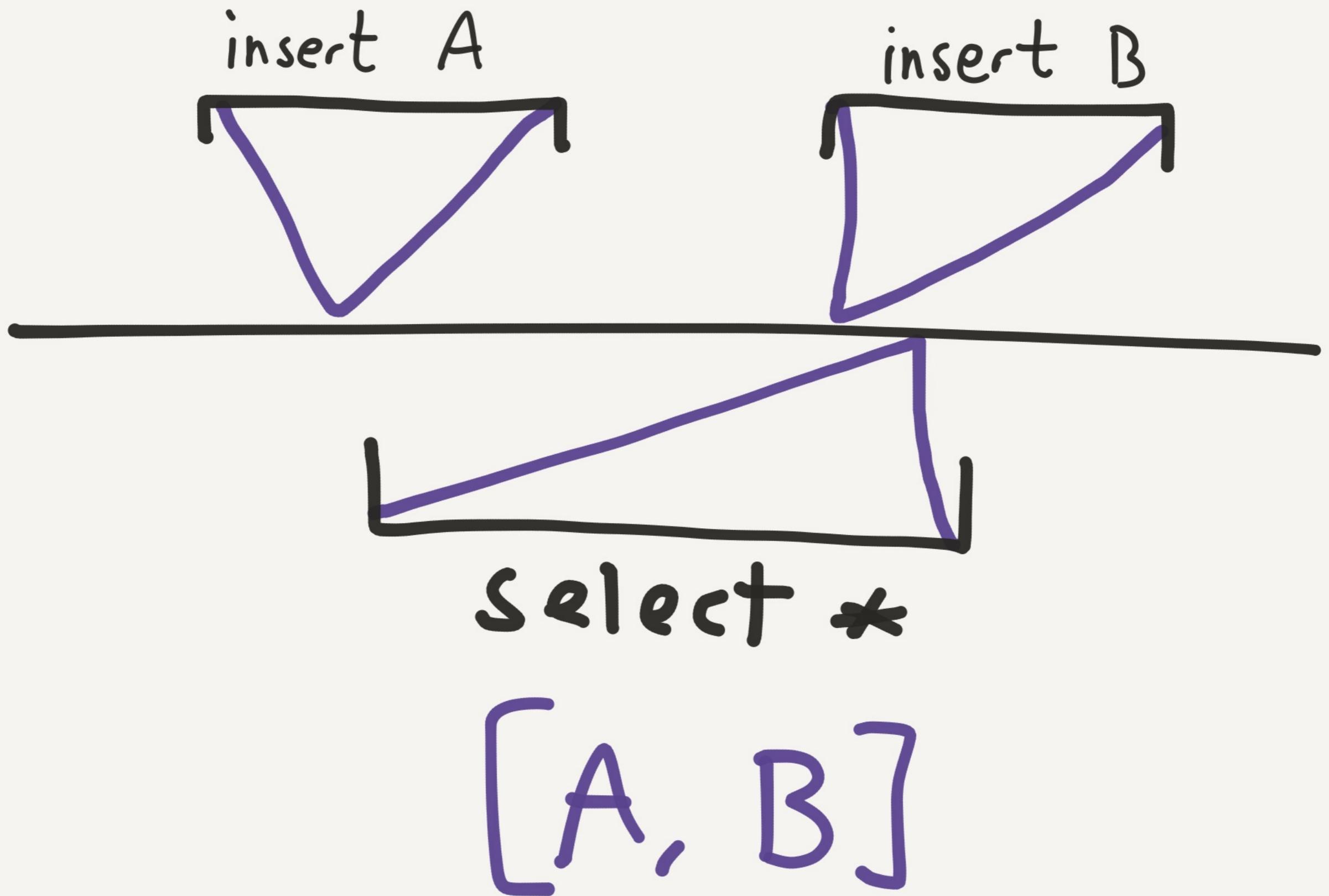
Txns across n Raft
clusters use a custom
protocol w/HLC

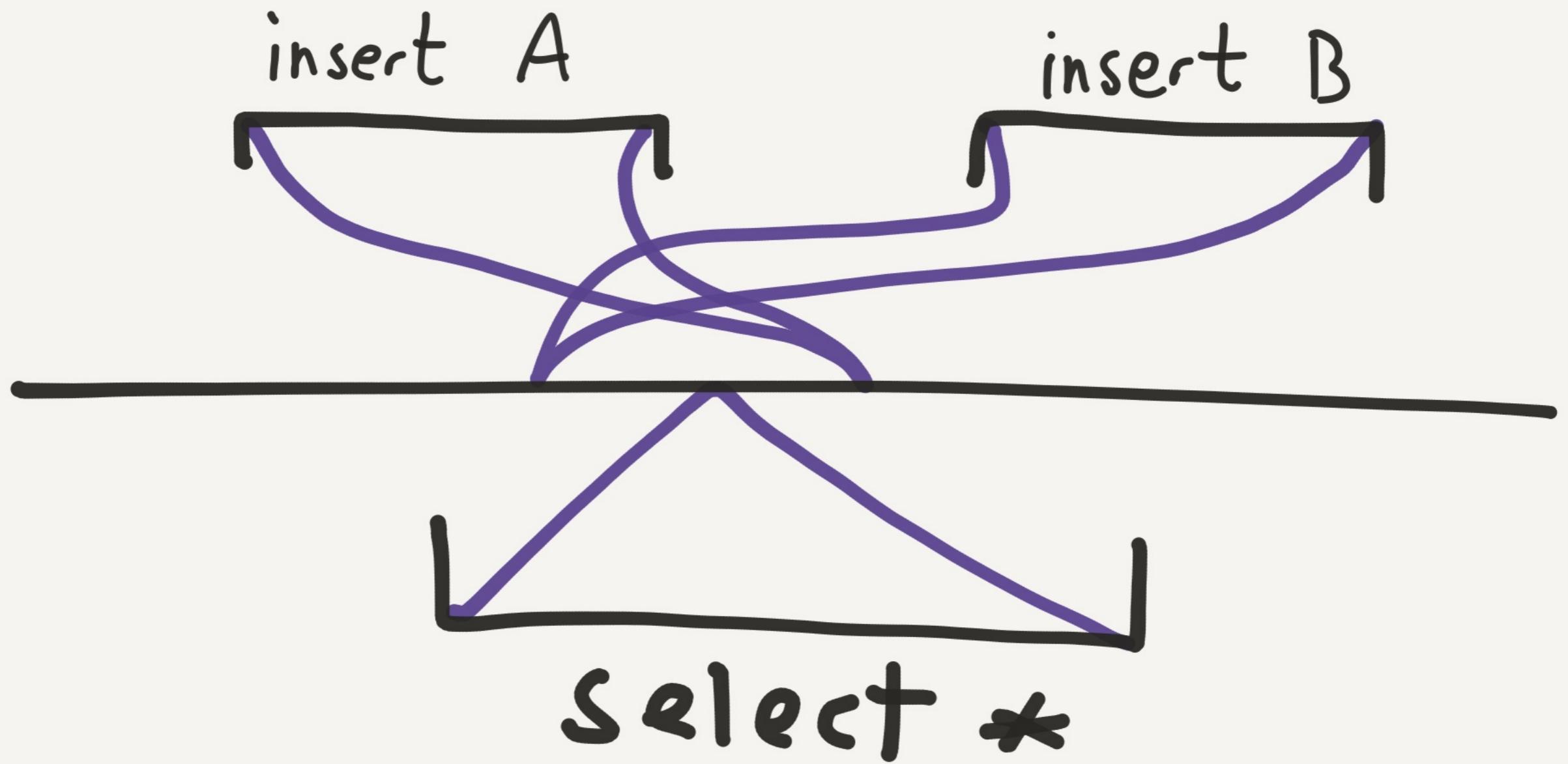
⇒ Needs synced clocks

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









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!

table 1

1
2

table 2

0
3

table 1

1
2

table 2

0
3
4

Select max;
insert max + 1

table 1

1
2
4

table 2

0
3
4

2 copies!

Time stamp Cache

- Stores set of keys read by a txh
- Prevents read keys from modification

Txns with same
timestamp can steal

each other's ts cache
entries



Fix: clear txn id for
Conflicting reads

⇒ Allows read-read, but
subsequent writes fail

Problem #2

insert 1;

insert 2;

insert 3;

...

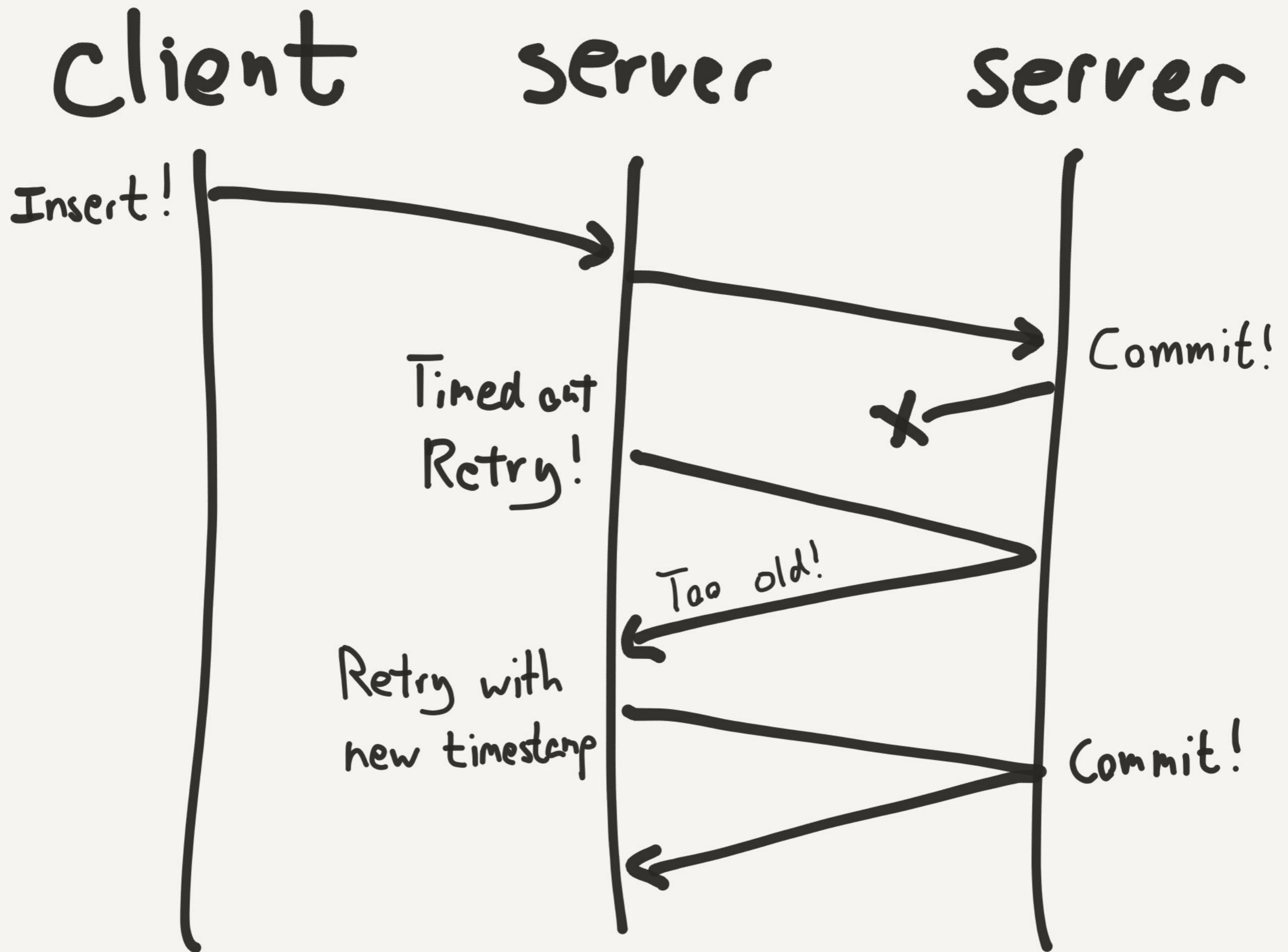
Select *;

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

whaaaaat.



- Single-stmt txns can be executed in 1 phase

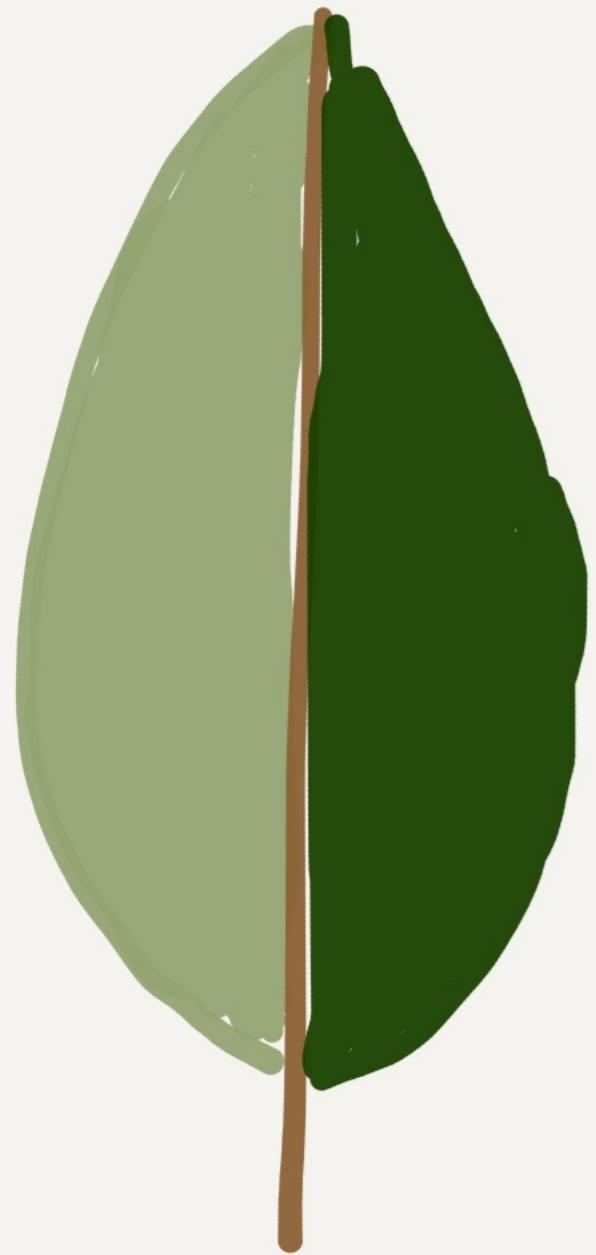


Now distinguishes
between retryable &
non-retryable errors

Cockroach DB 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

Last writes

2015

Safe writes

Dirty/stale reads



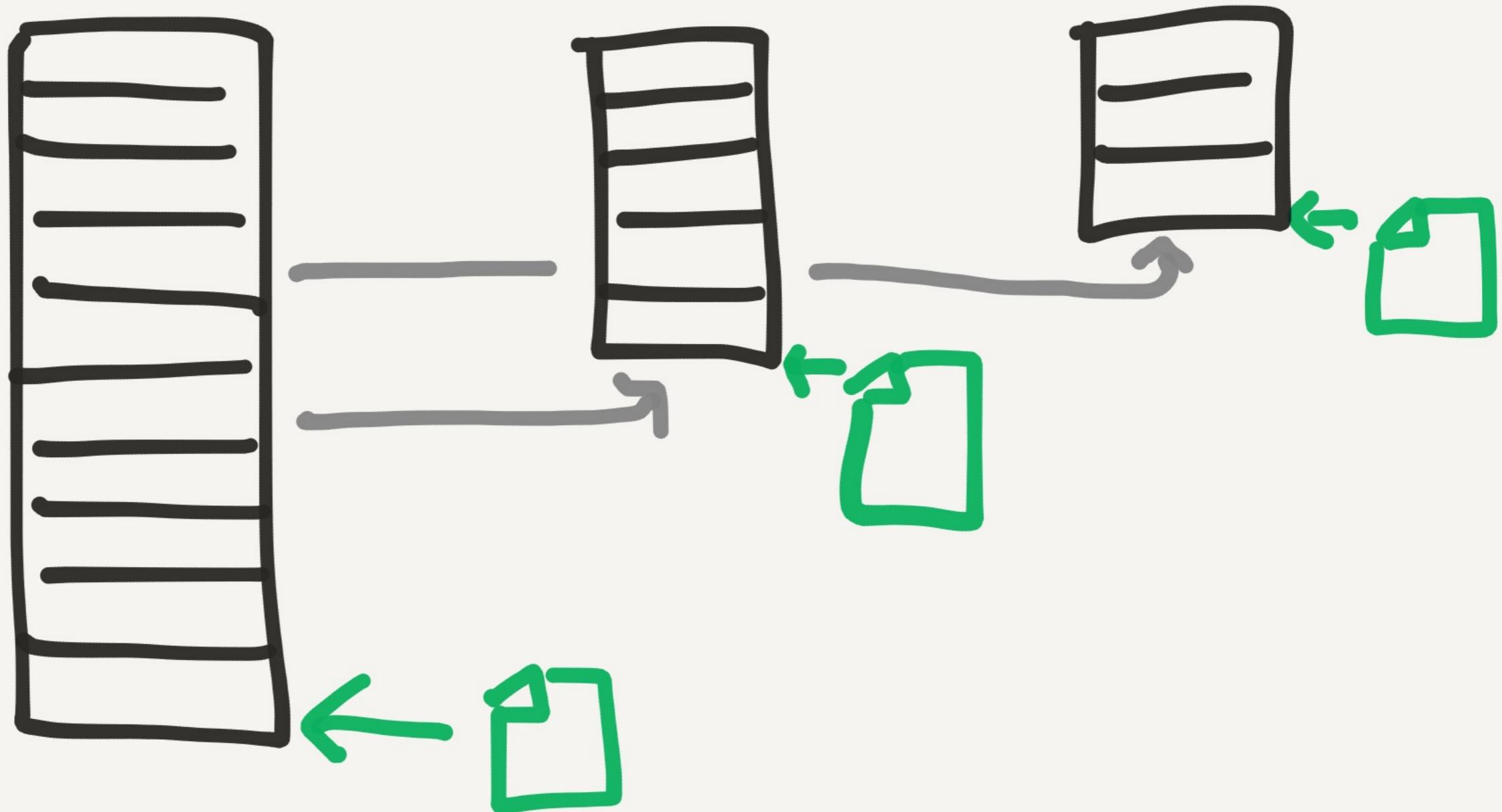
3.0: Wired Tiger
Replication v1

- 3.2: Majority reads
- 3.4: Linearizable reads

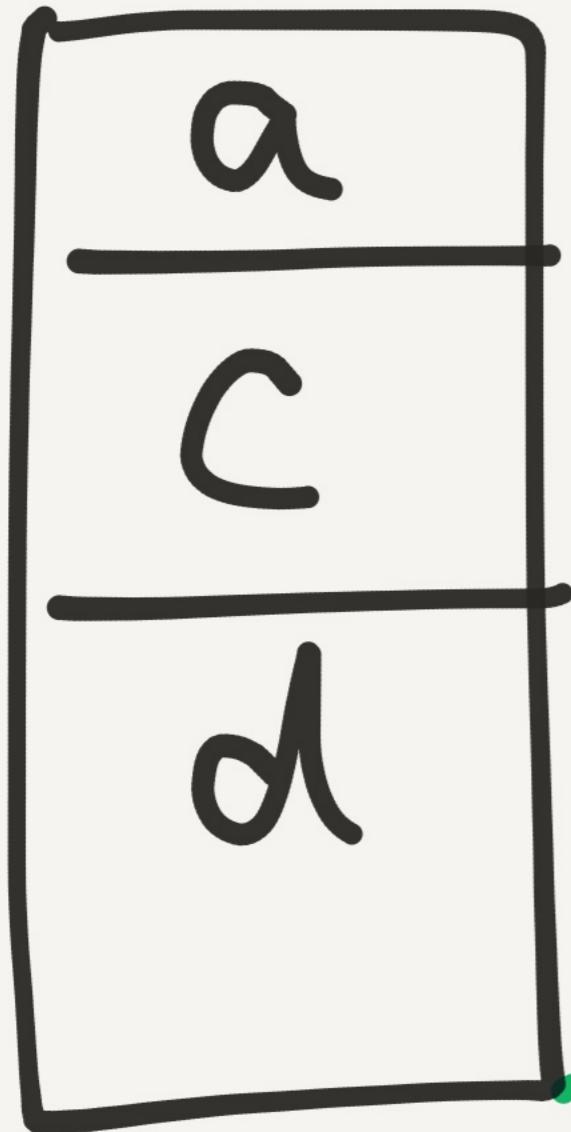
primary

sec.

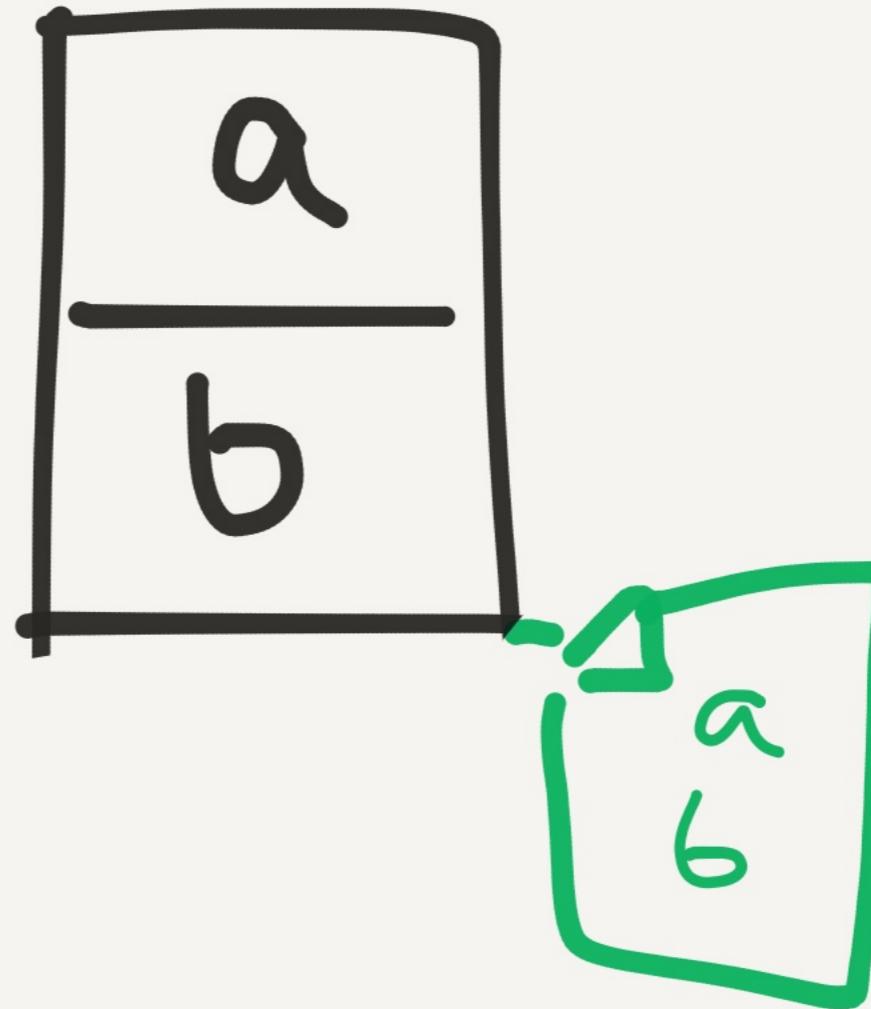
sec.



primary

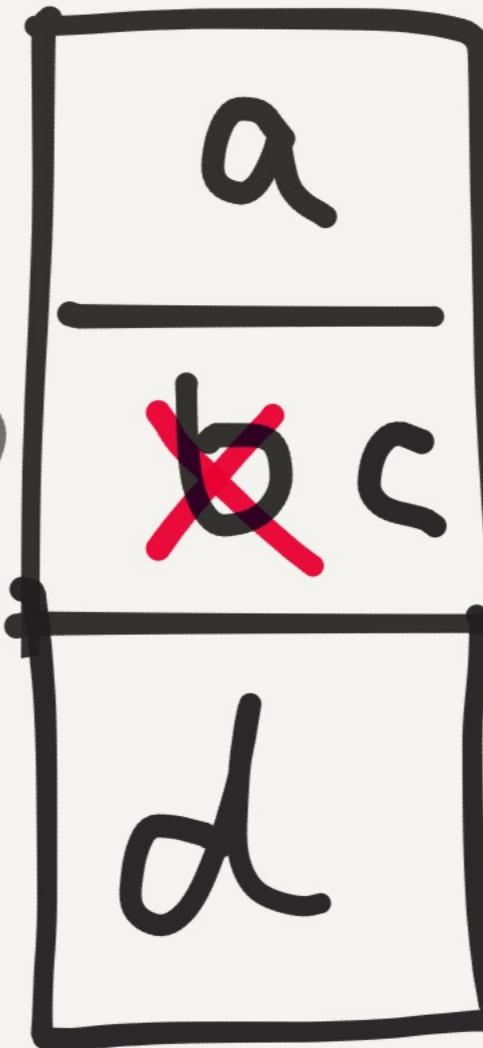
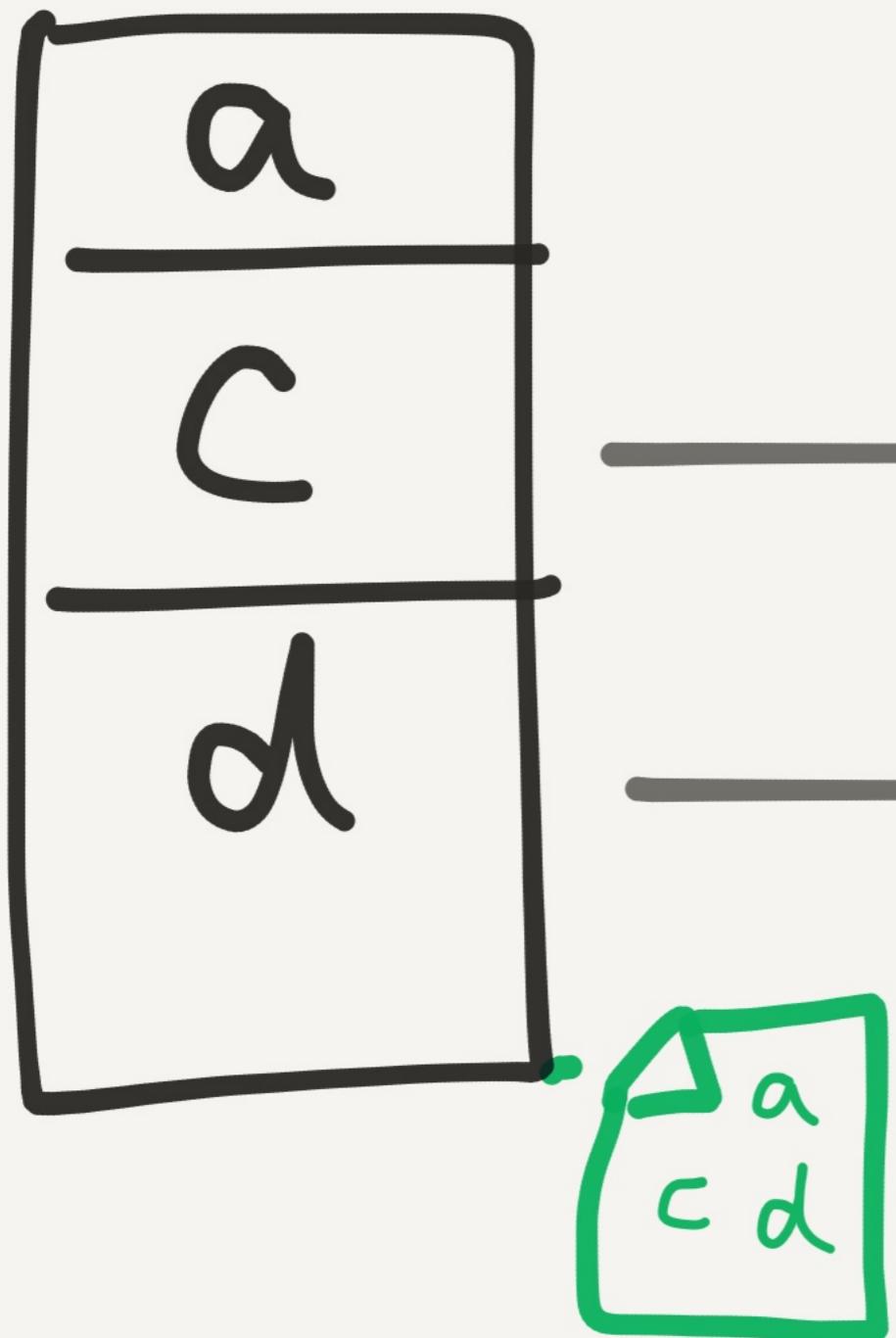


secondary

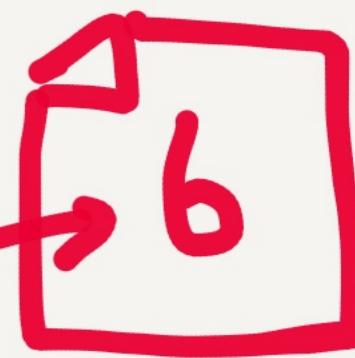


primary

secondary

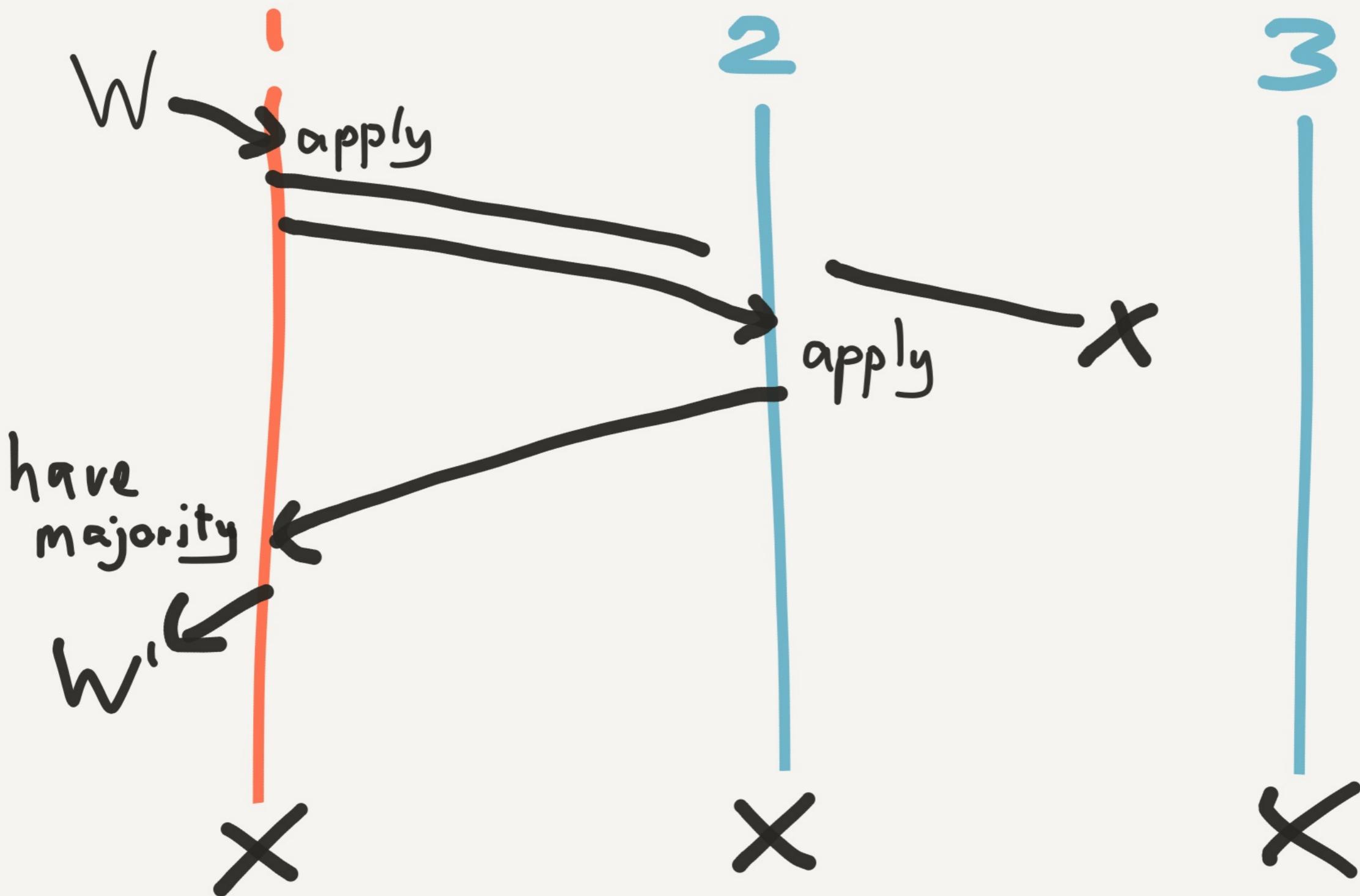


rollback



v0 protocol:

- Dirty reads
- Lost writes?



???

..

Optime:

Max { last optime + 1
local wall clock

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

Nade
w/ highest
optime
wins

w_1

w_2

$t = 10$

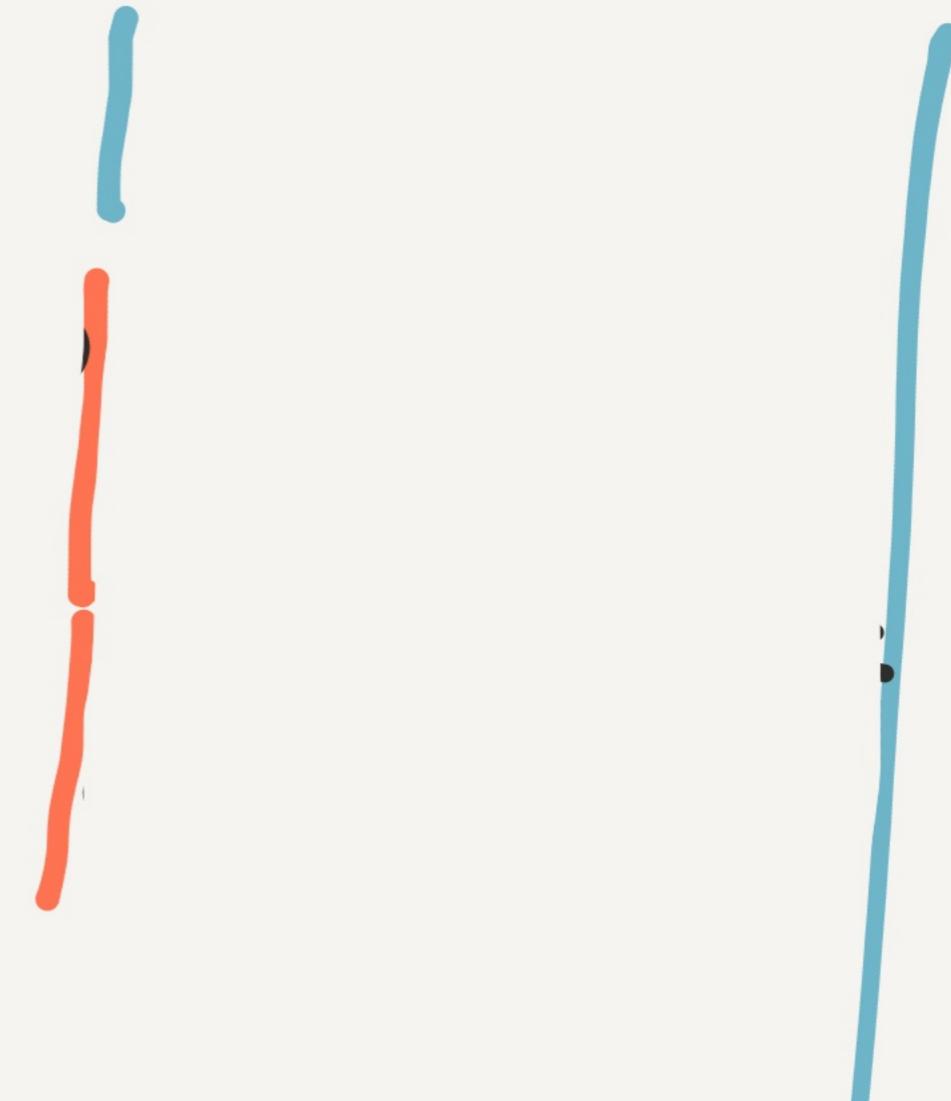
$t = 11$

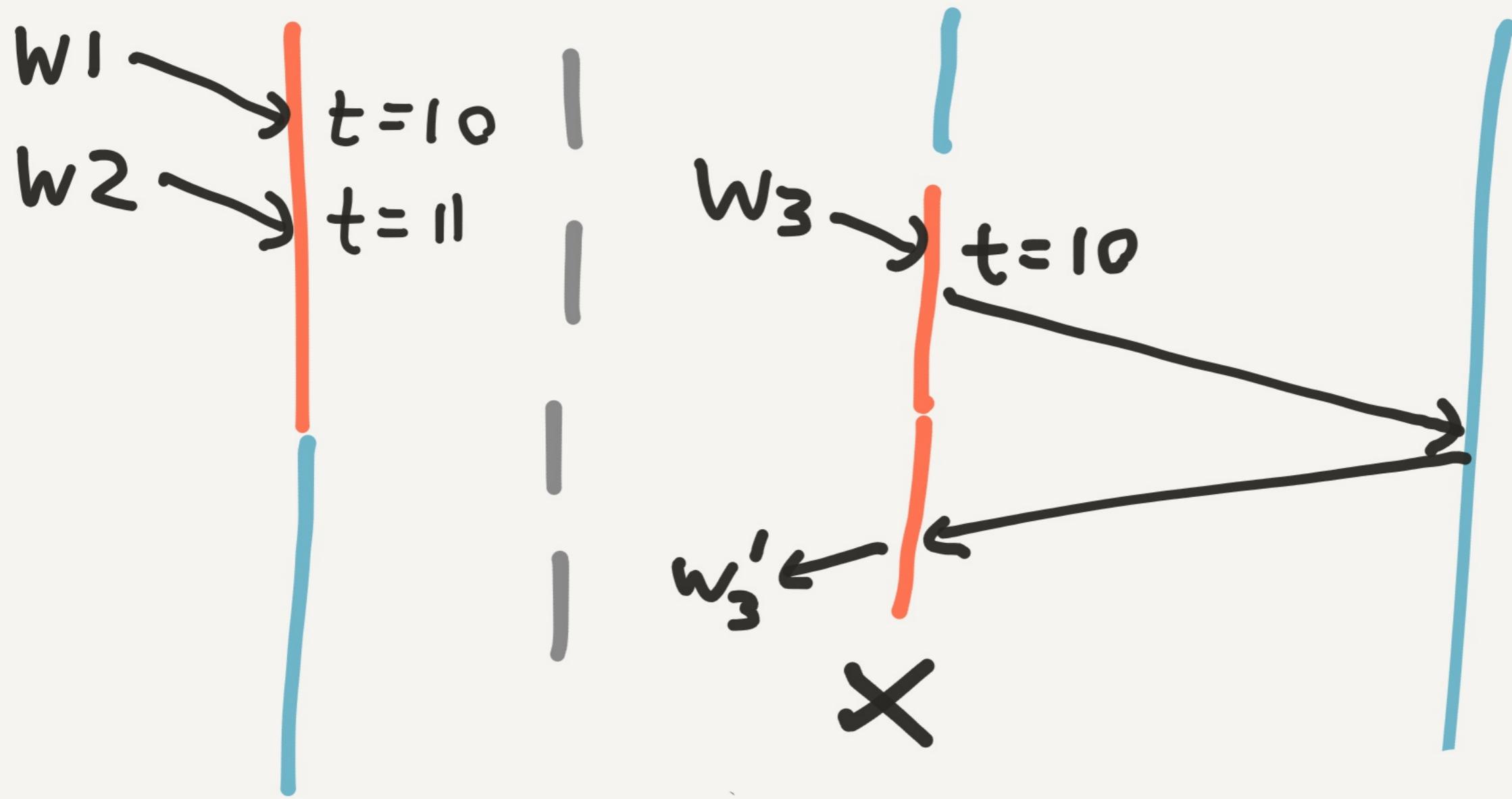
A diagram illustrating a sequence of events or states over time. Two vertical lines are shown: a red line on the left and a grey line on the right. The red line is labeled $t = 10$ and the grey line is labeled $t = 11$. Above the red line, there are two arrows pointing towards it, labeled w_1 and w_2 , indicating inputs or initial conditions at time $t = 10$.

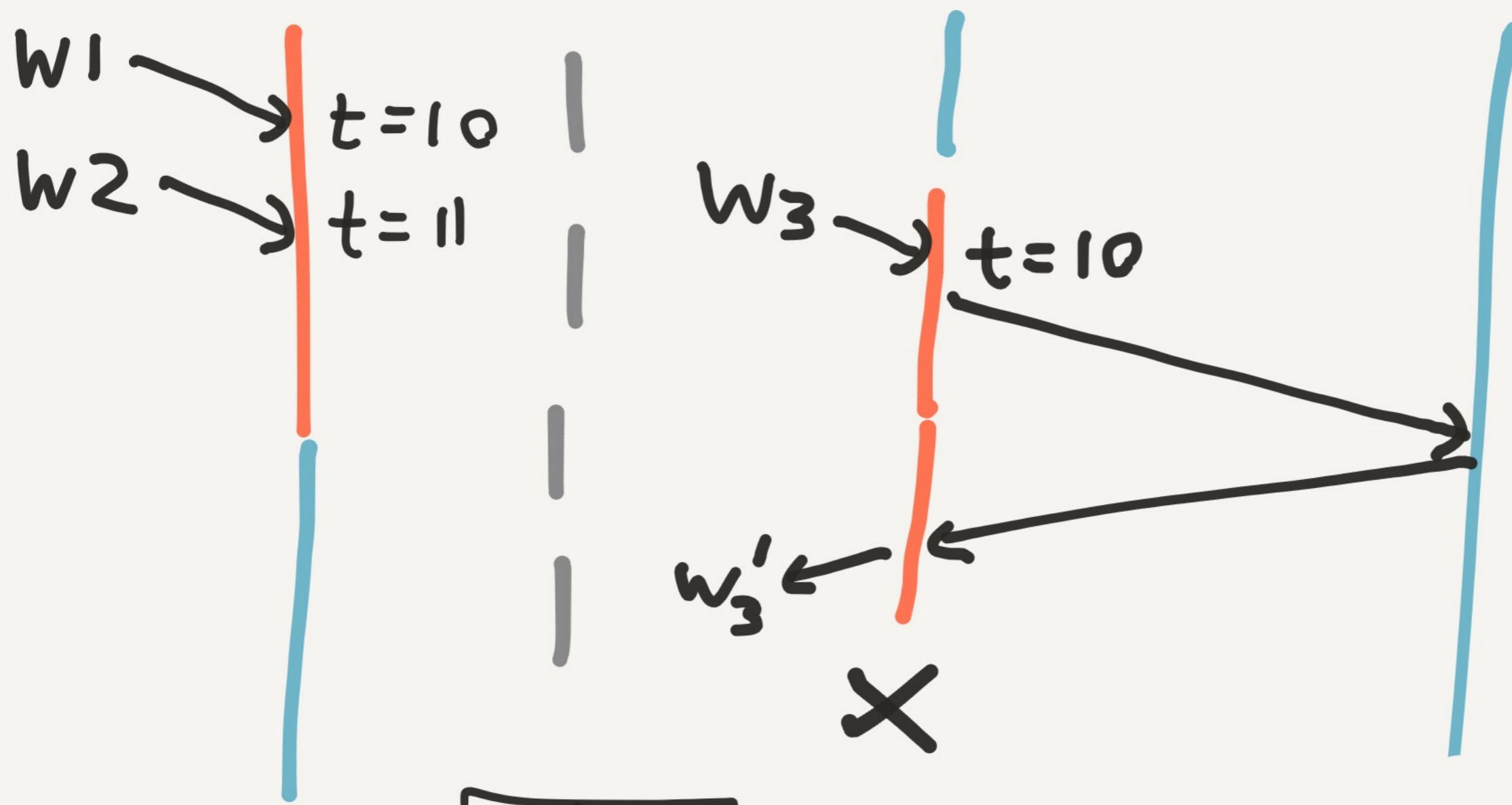
—

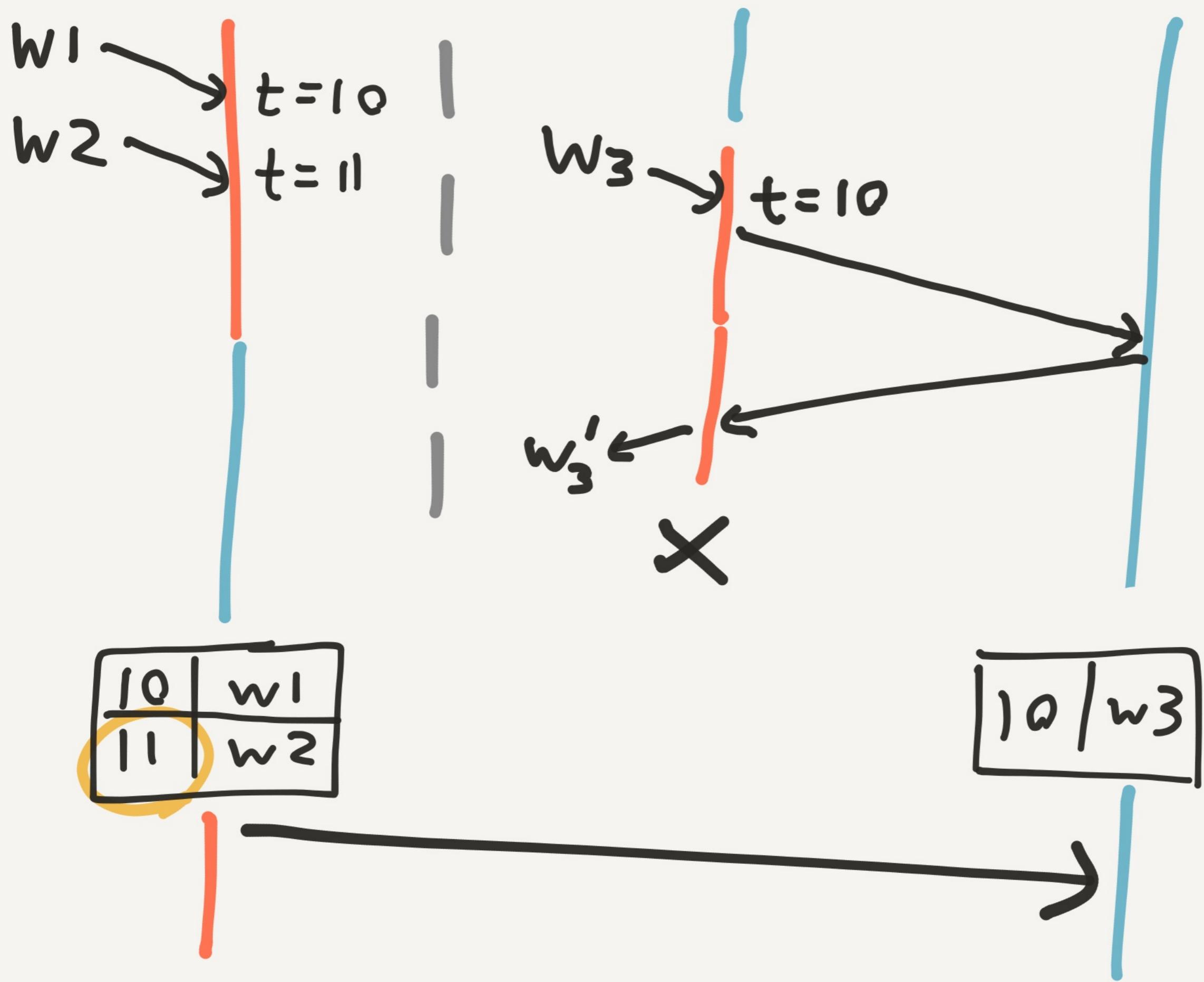
—

—



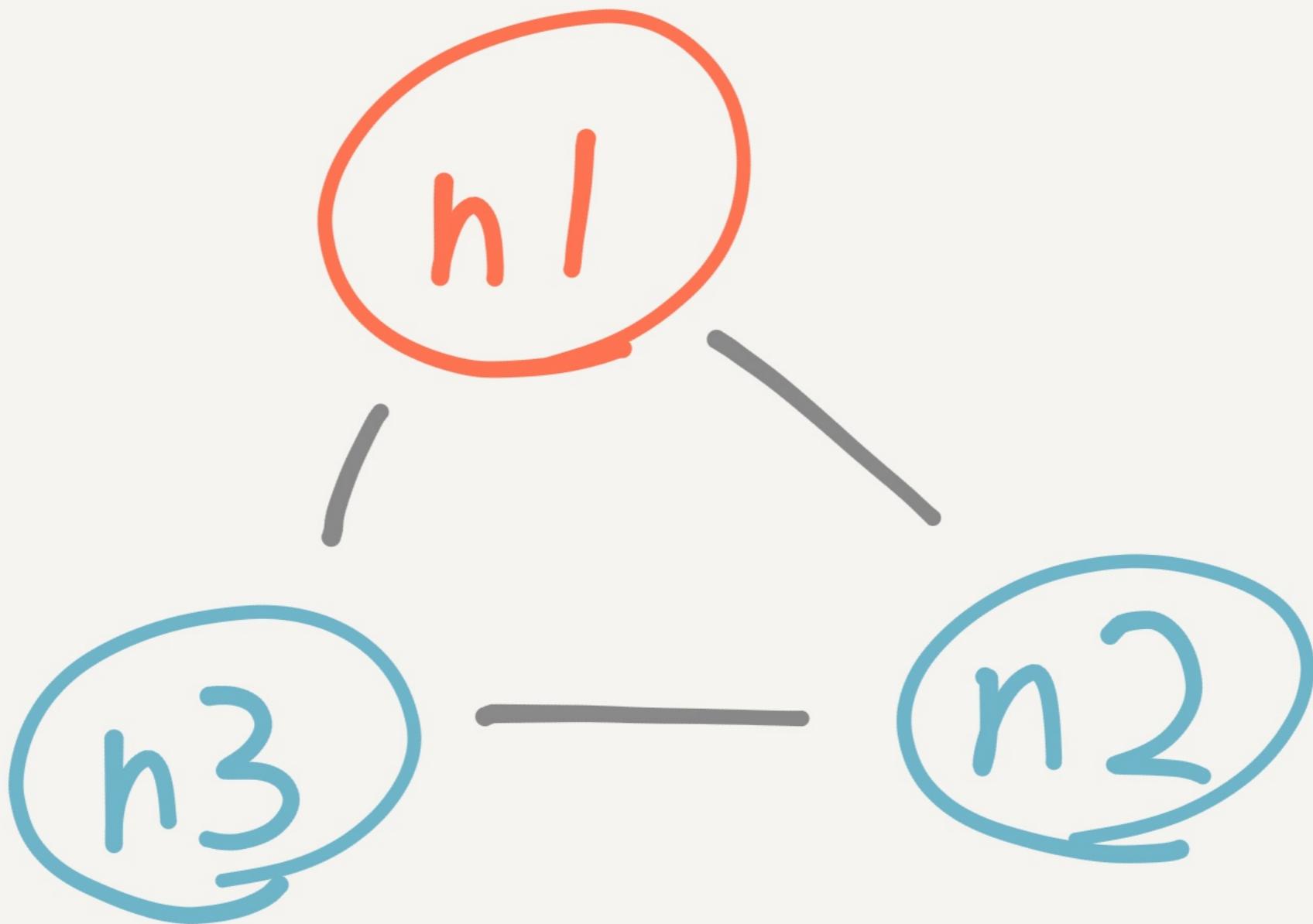


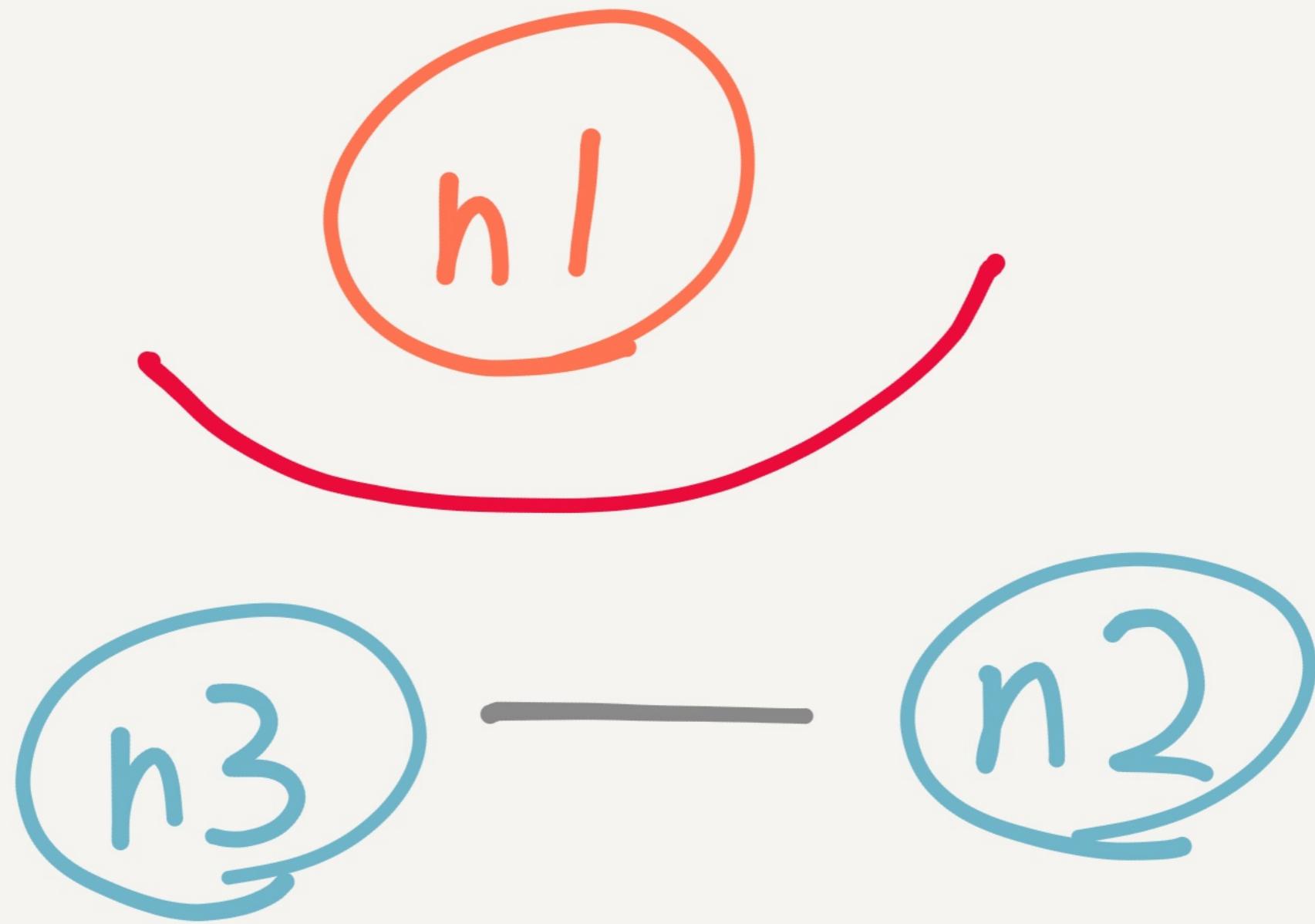


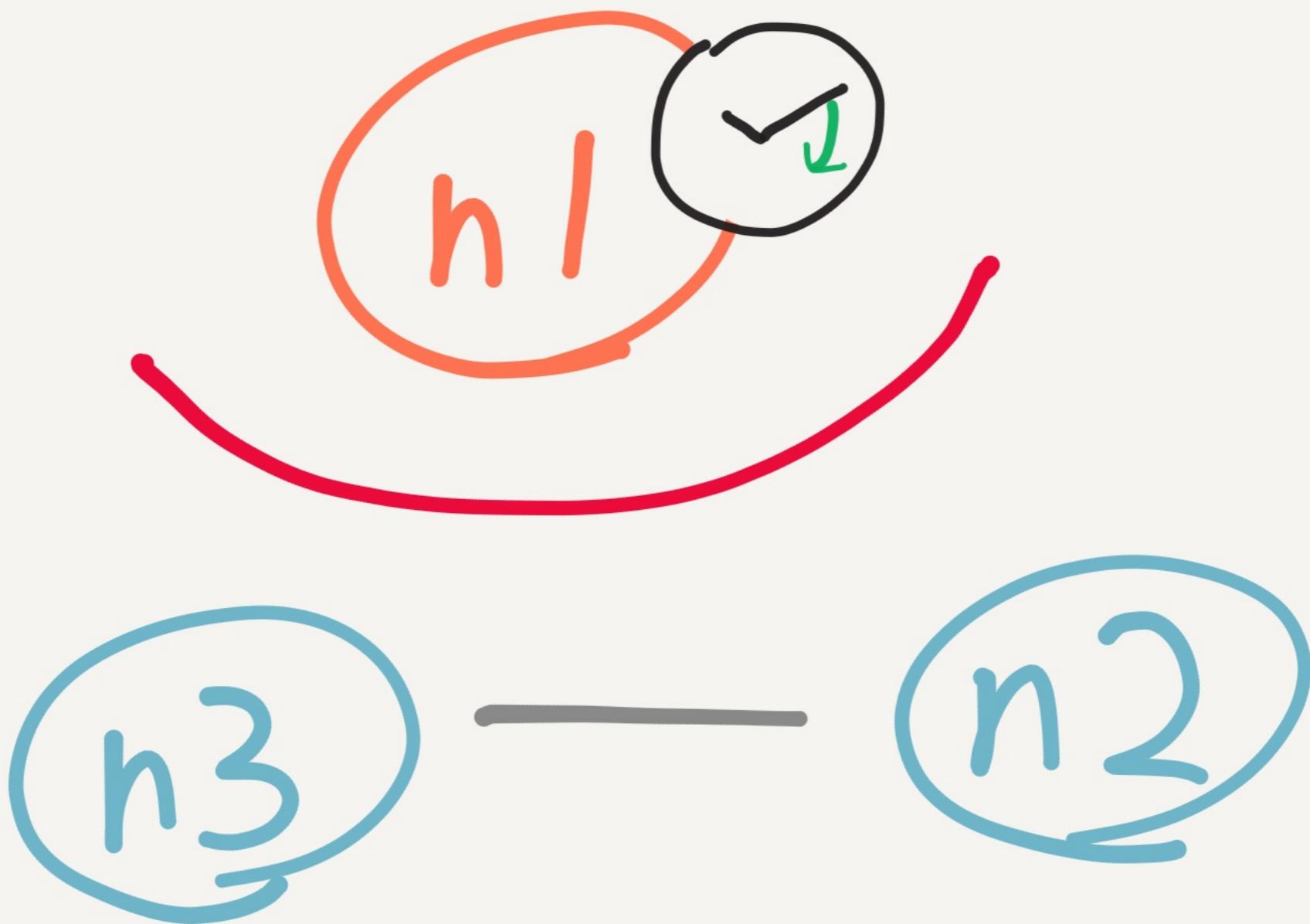


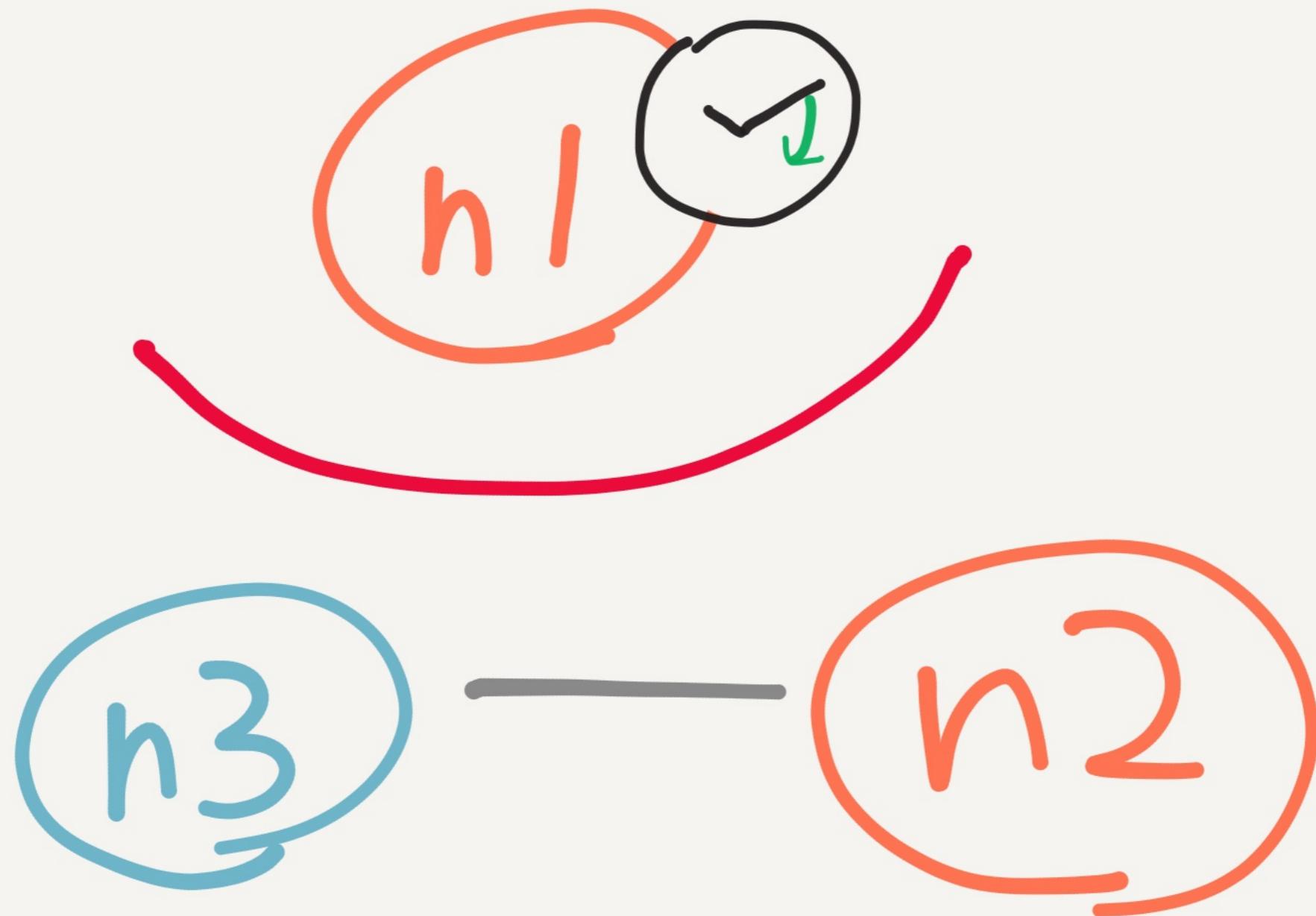
Requires precise timing

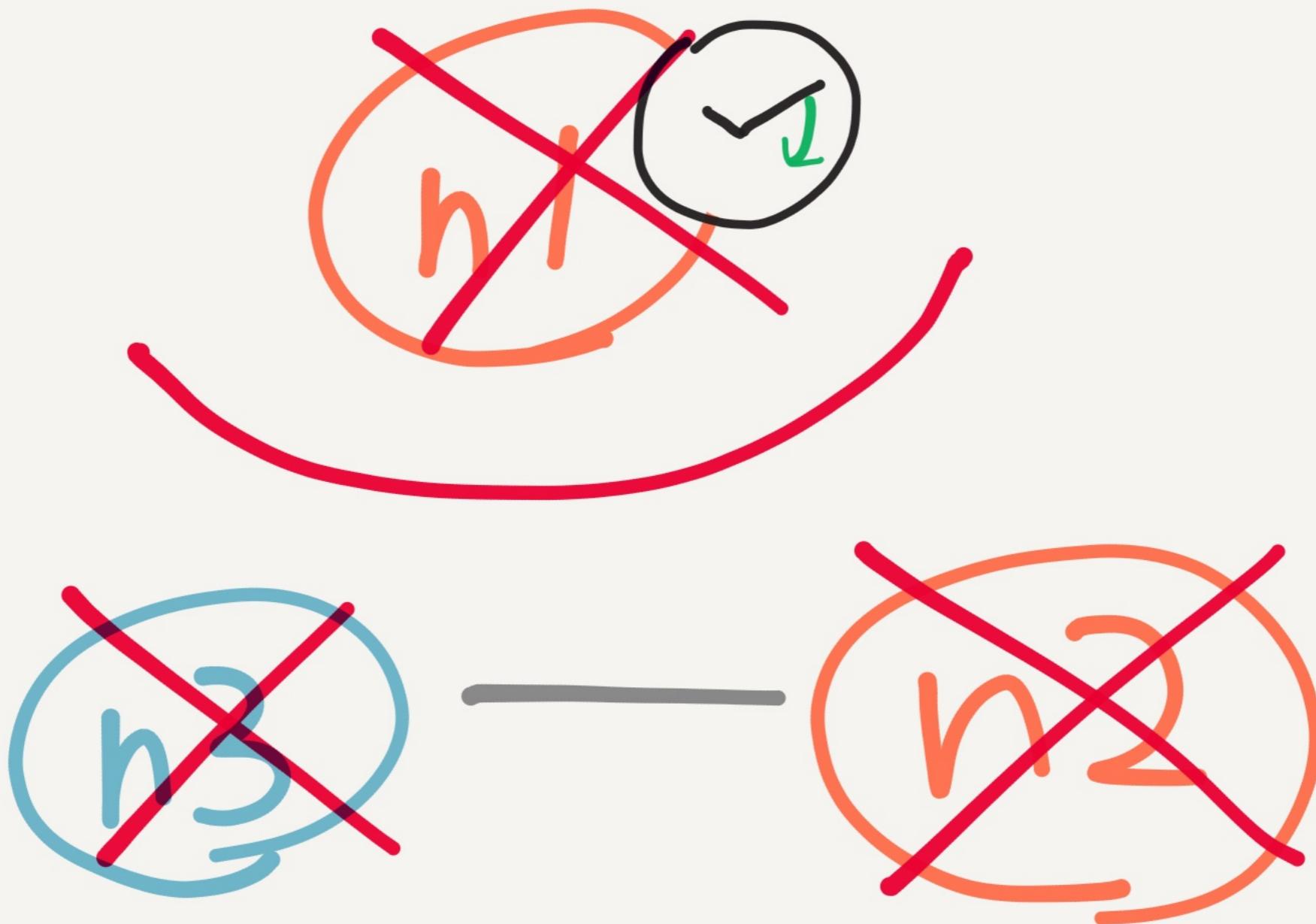
- But window grows
with clock skew!

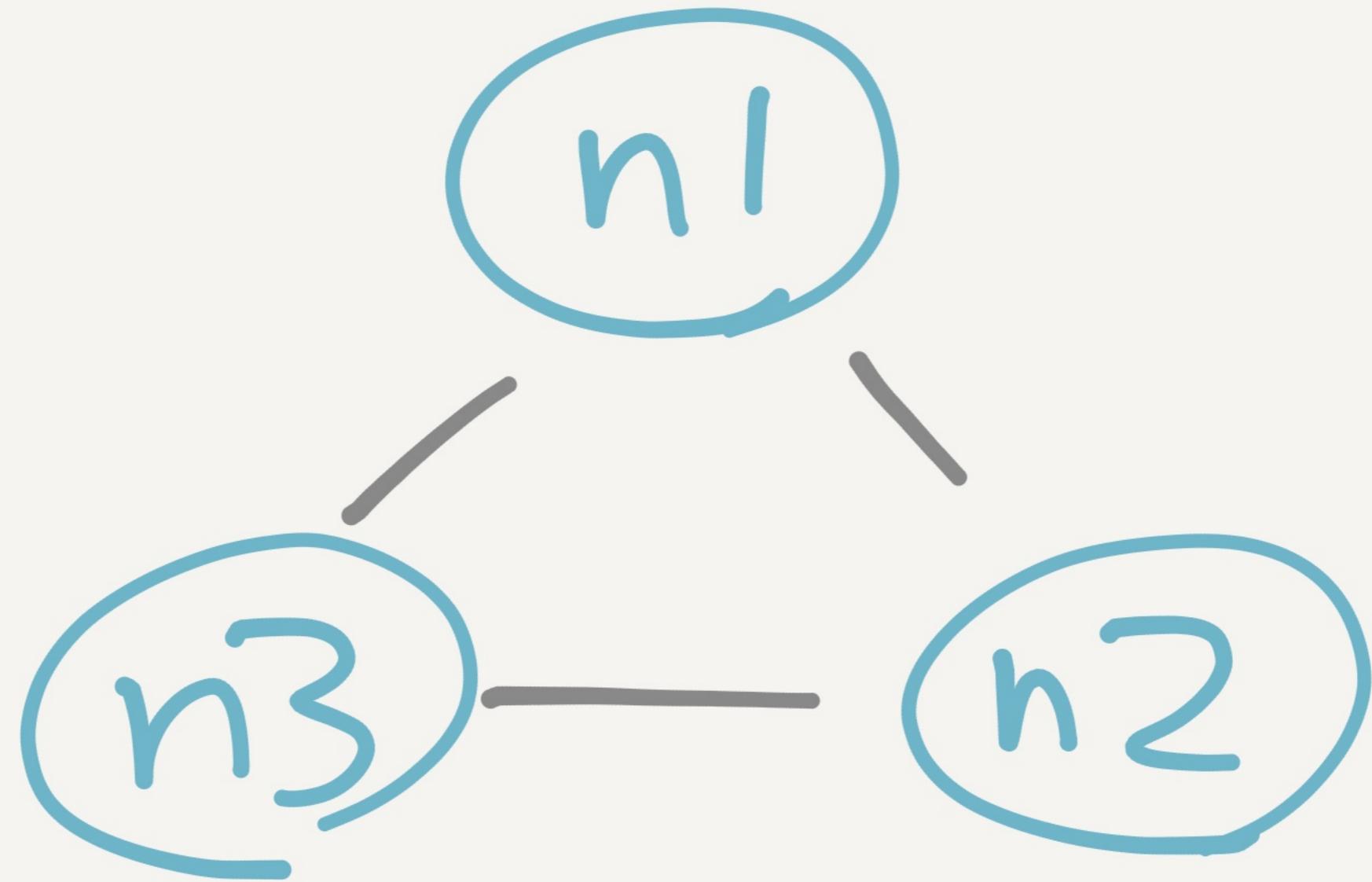


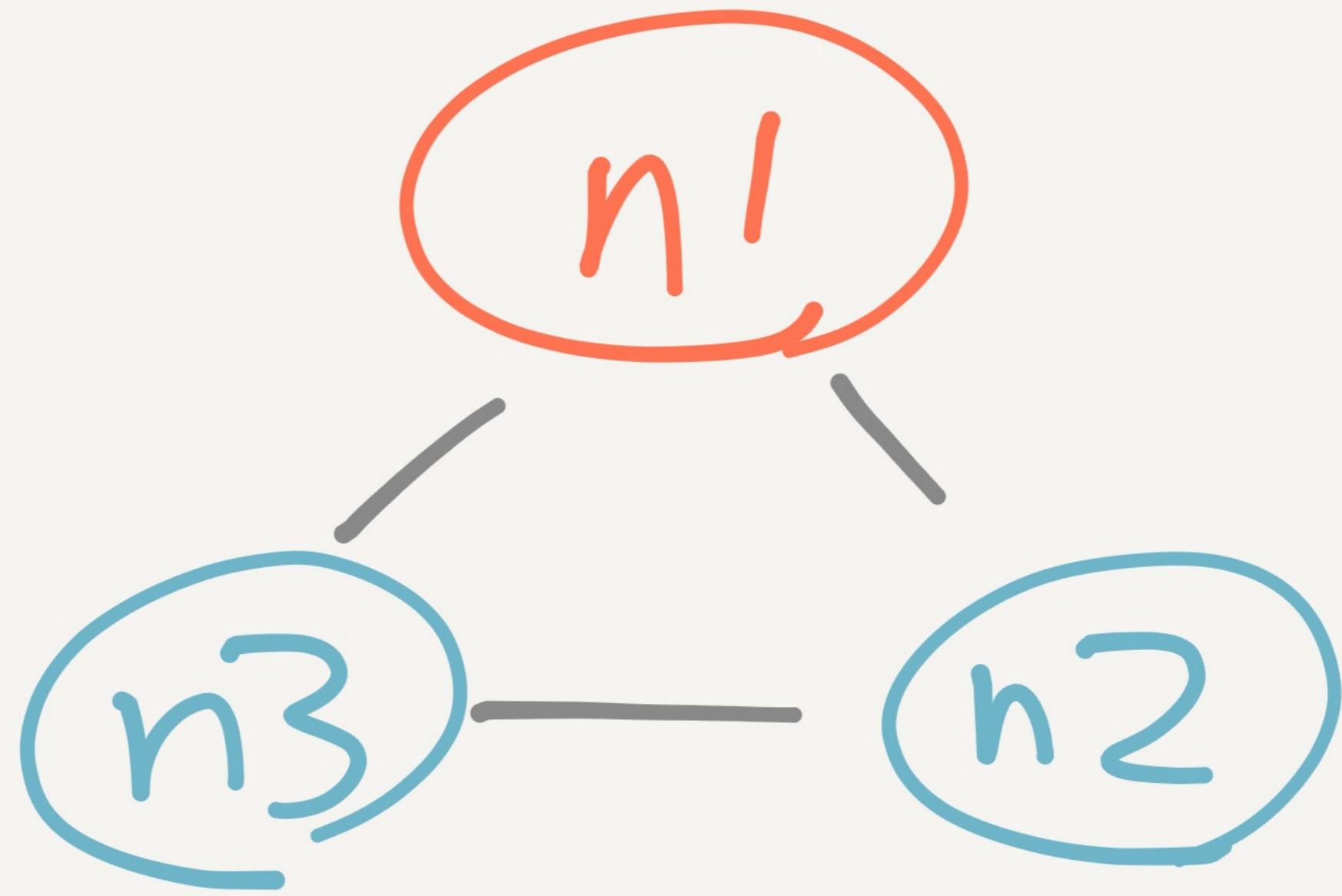












ok

lost

$$\begin{array}{r} 322 \\ + 93 \\ \hline \end{array}$$

4525

attempted inserts

v○ protocol

is fundamentally

broken 😞

Protocol v1

Default in 3.2
for new replsets

Optime

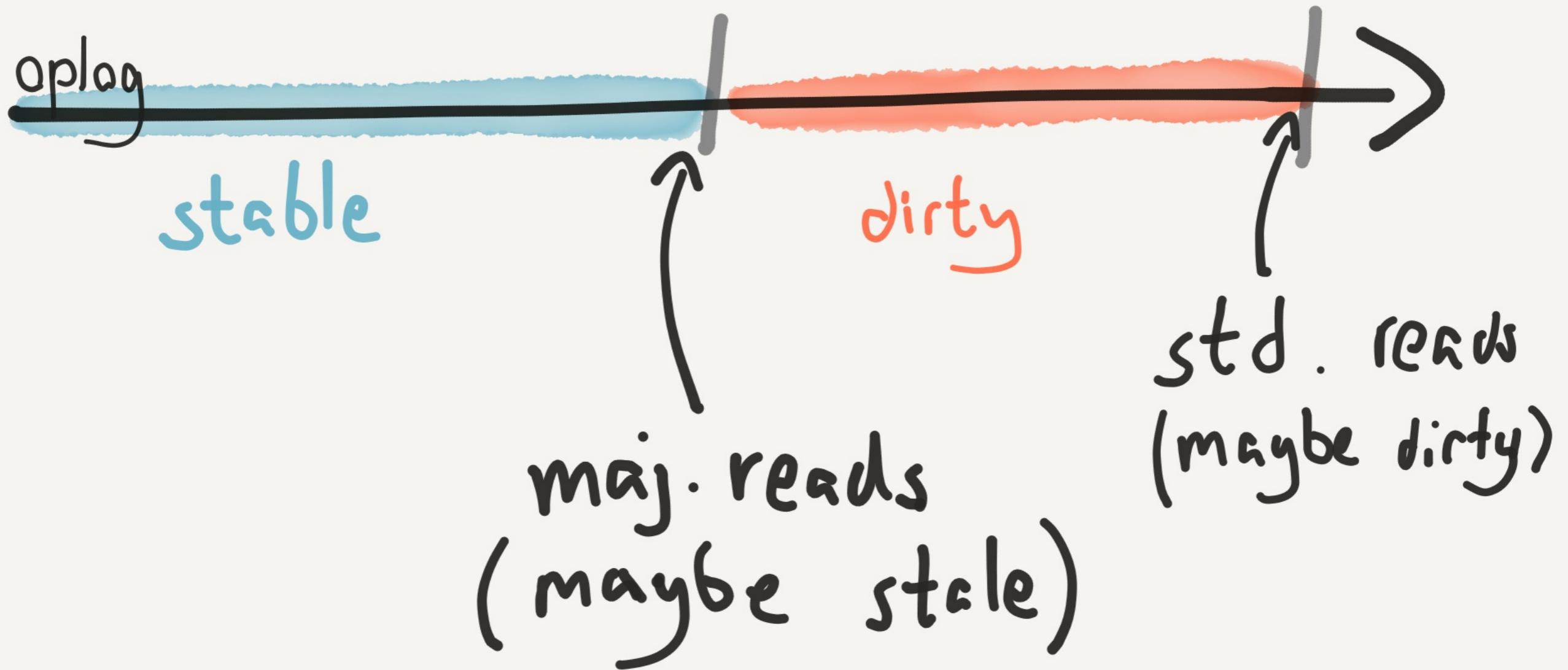
[term, time]

logical clock

old optime

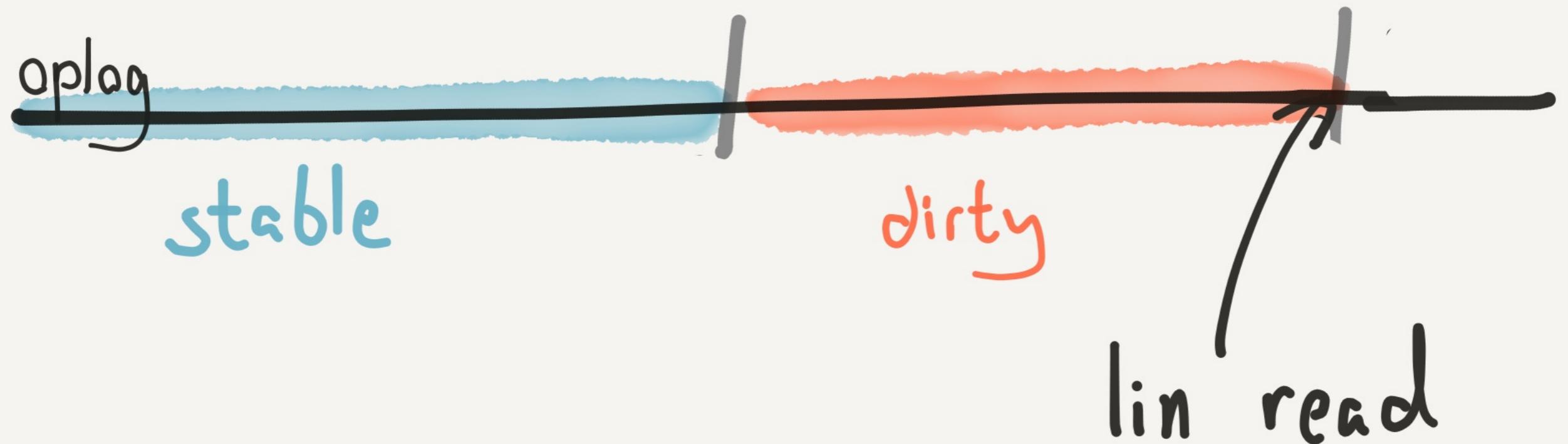
majority
state

Current
state



majority
state

Current
state



majority
state



OK

lost

515 + 417

} acked

6078

attempted inserts

Server - 27053

primaries could ack
writes from prior
terms

Heartbeats received

by outdated primary

could advance Commit

index

3.4.0 - rc4

Server - 27149

Secondaries ignored
terms & took ops
from old leaders

THE FEDERAL TRADE COMMISSION



Fixed in

3.2.12

3.4.0

3.5.1

Recommendations

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

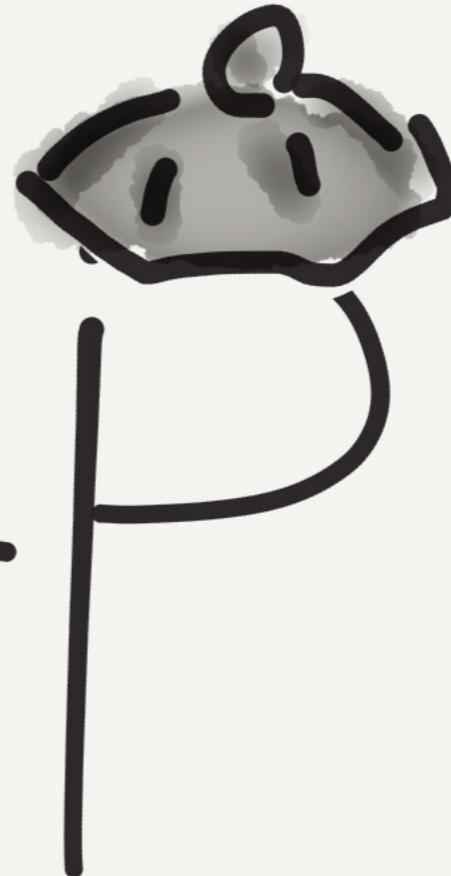
Use Write Concern

MAJORITY

Read Concern

MAJ/LIN

Recap



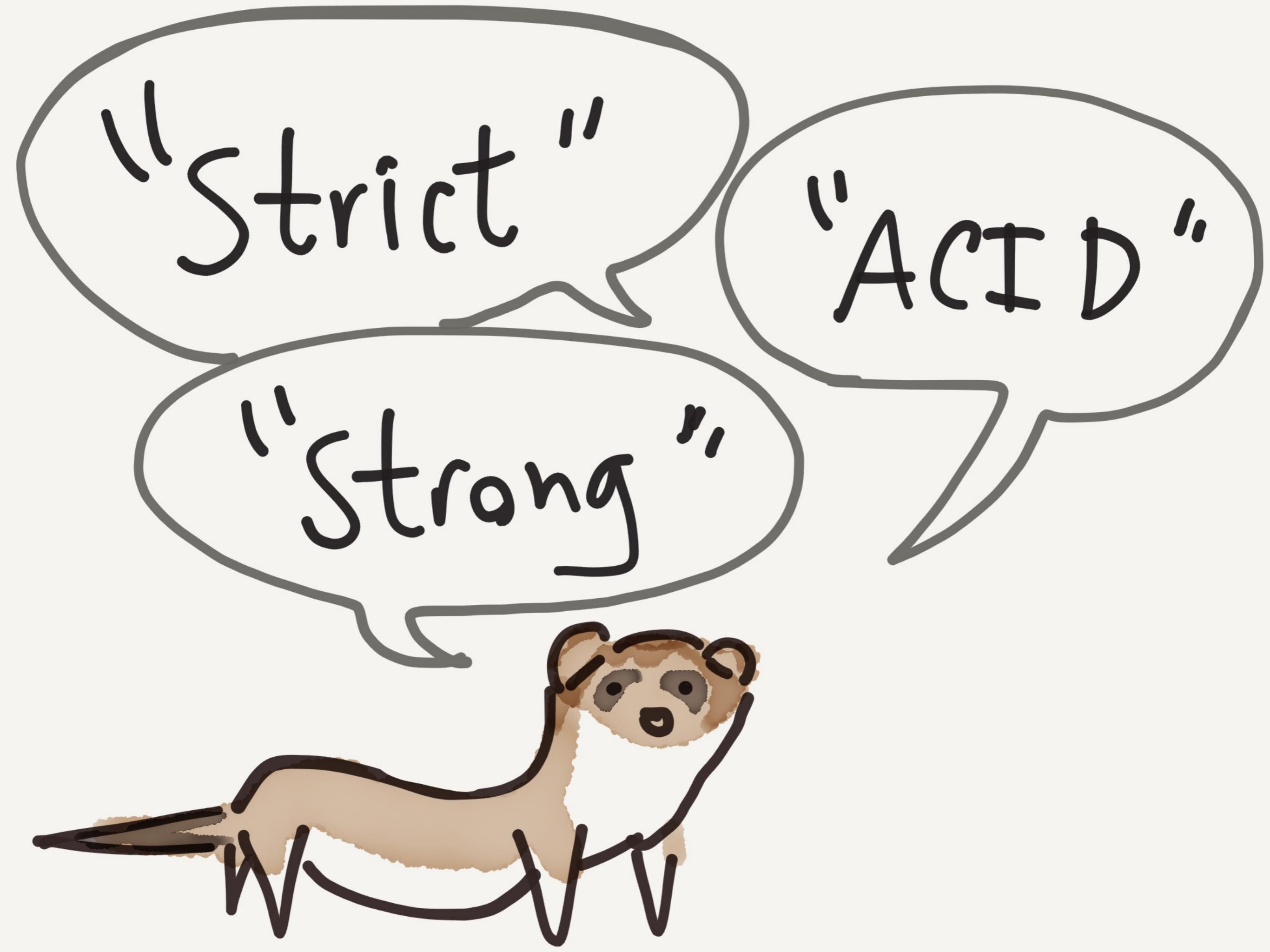
Read the docs!



Then test it

— for —

YOURSELF



"Strict"

"ACID"

"Strong"

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 1028000

fake time ...

GC/TQ Pause

killall -s STOP foo

killall -s CONT foo

Network Partition

iptables -j DROP

tc qdisc ... delay...
drop...

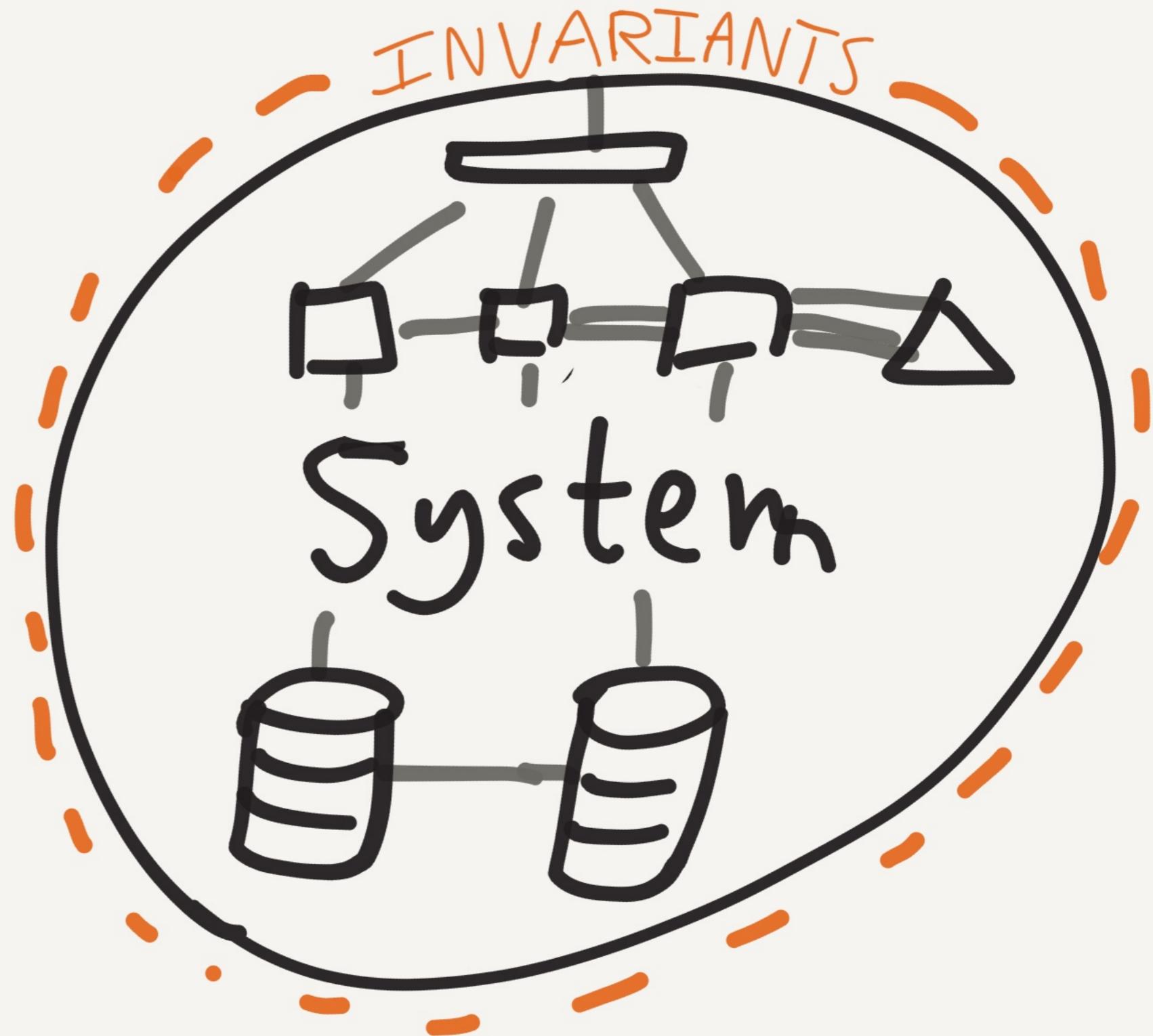
Test your

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

Crate.io

Cockroach Labs

MongoDB

Boaz Leskes

Peter Alvaro



Funded
Research

<http://jepsen.io>