JEPSEN 8

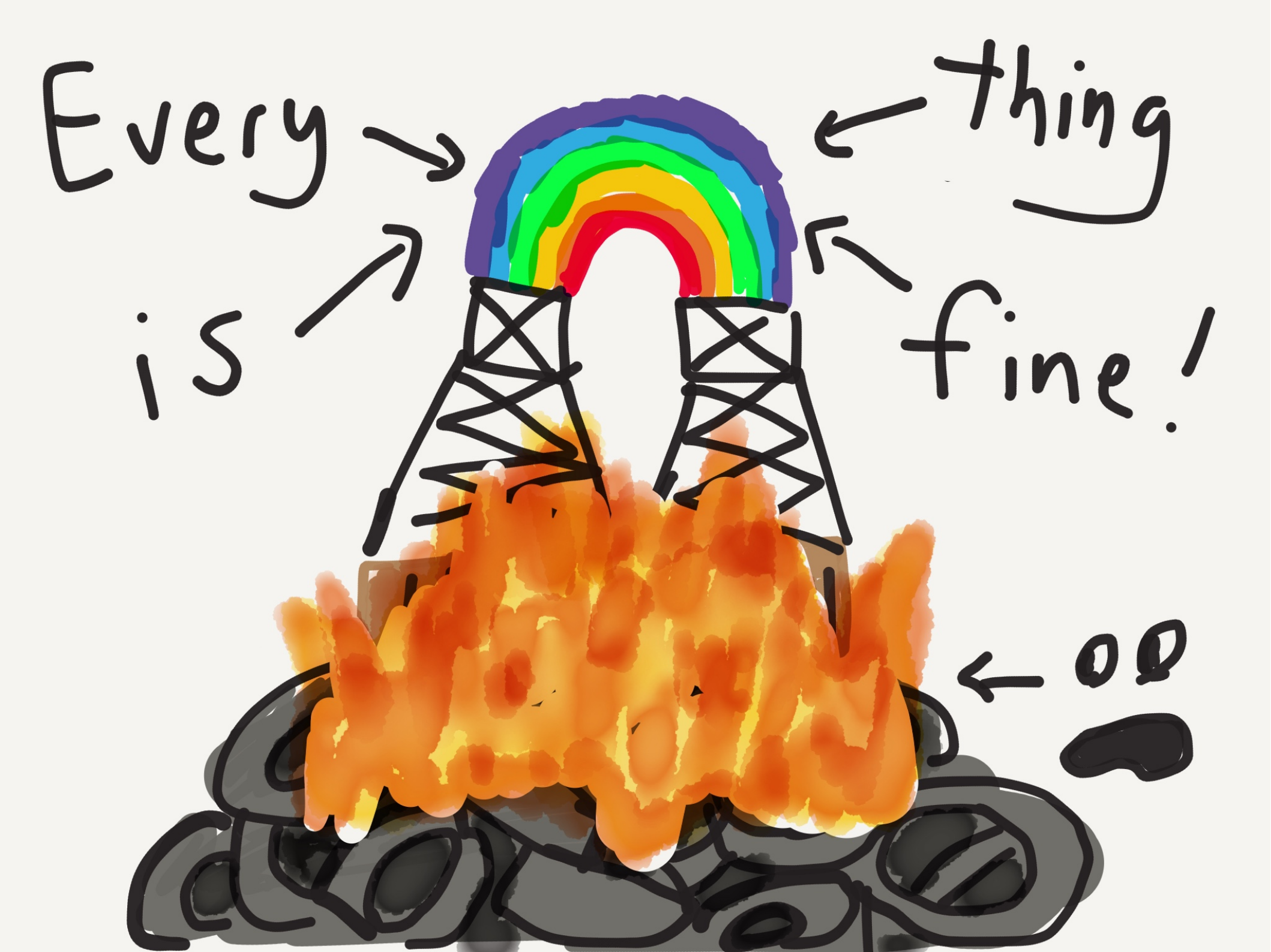
HPTS

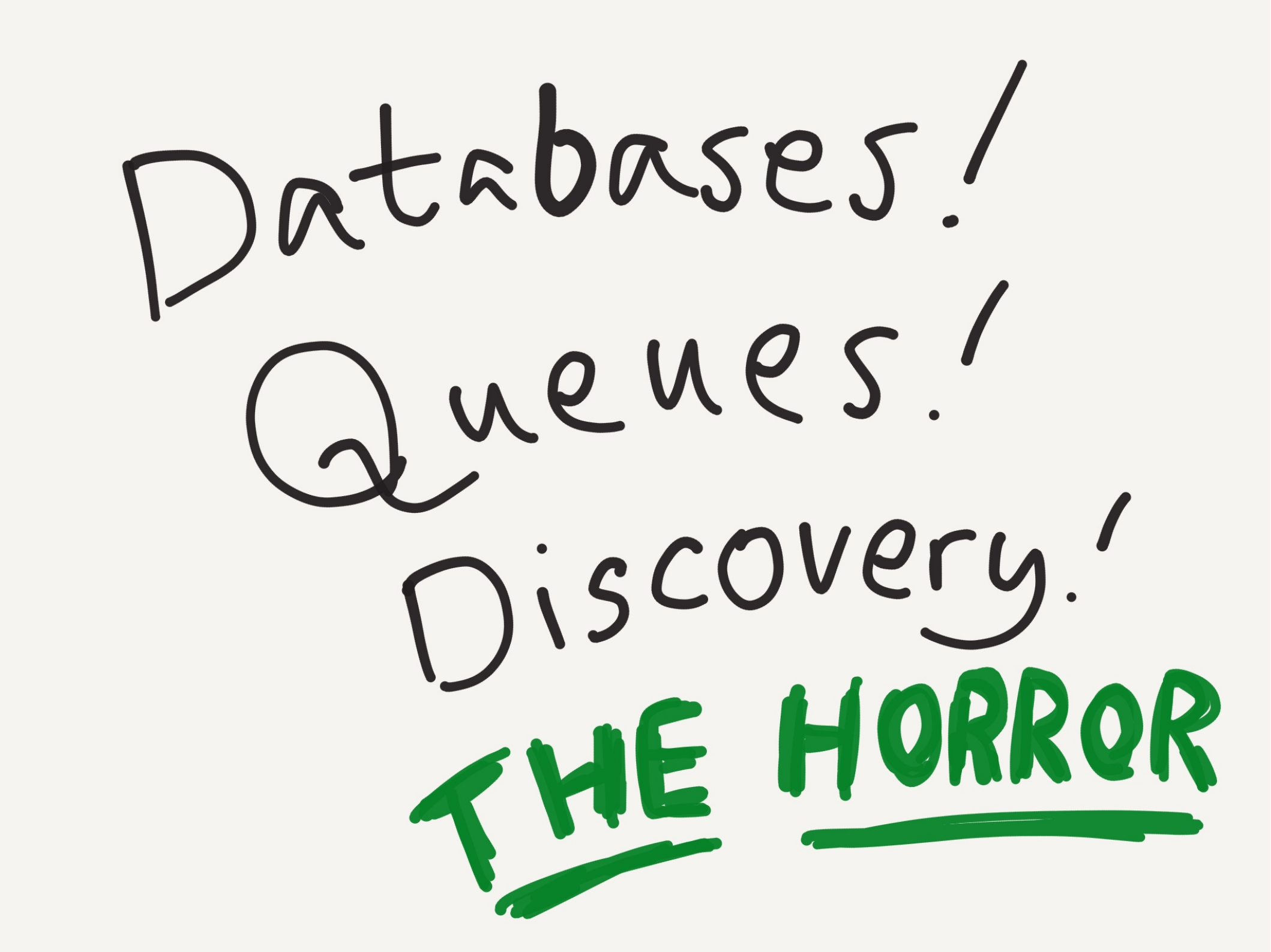
Kyle Kingsbury

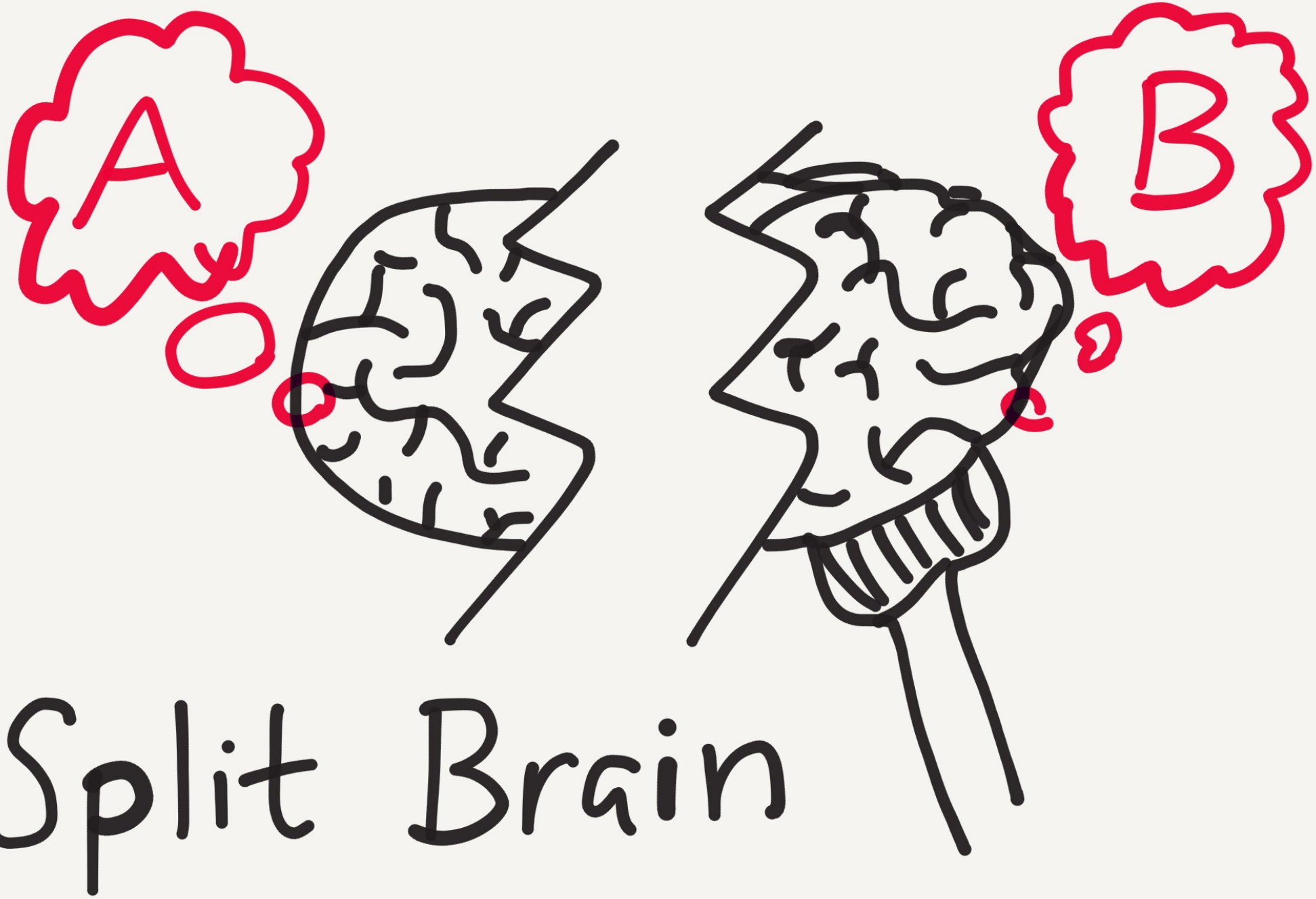
@aphyr

I break databases!


Public API
API code
Ruby
DBs

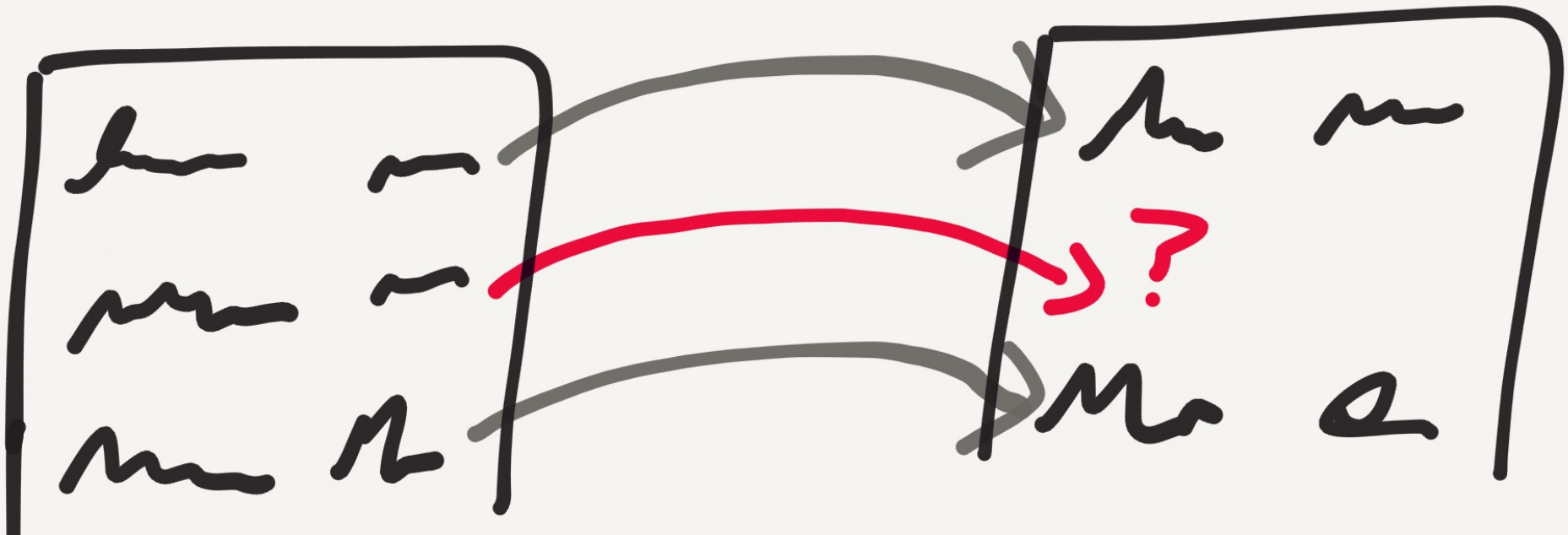

Every
thing
is
fine!
oo

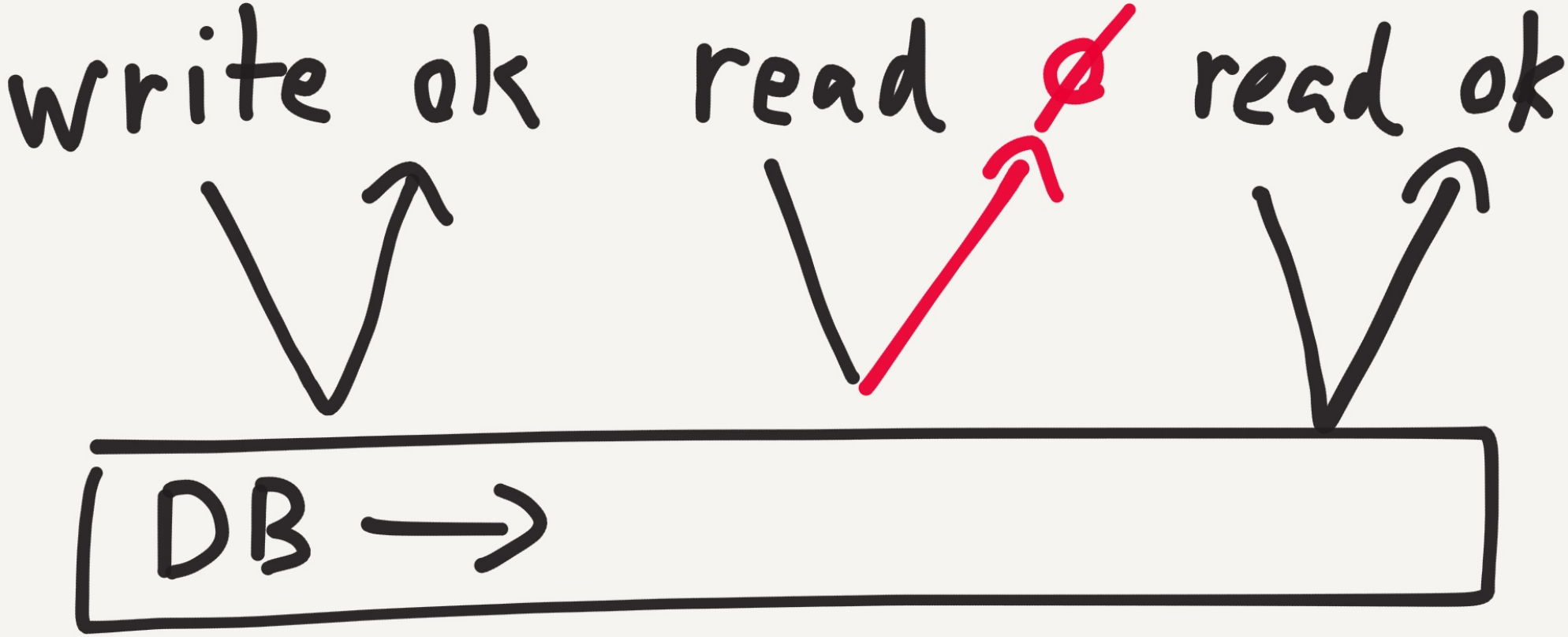

Databases!
Queues!
Discovery!
THE HORROR



Split Brain

Broken Foreign Keys

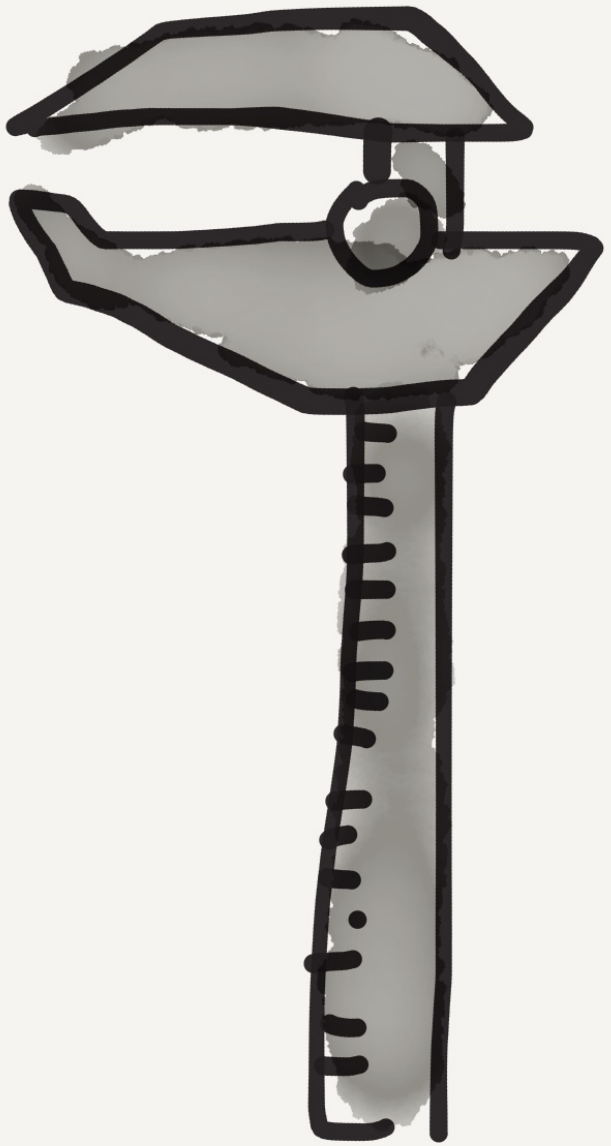




Anomalies

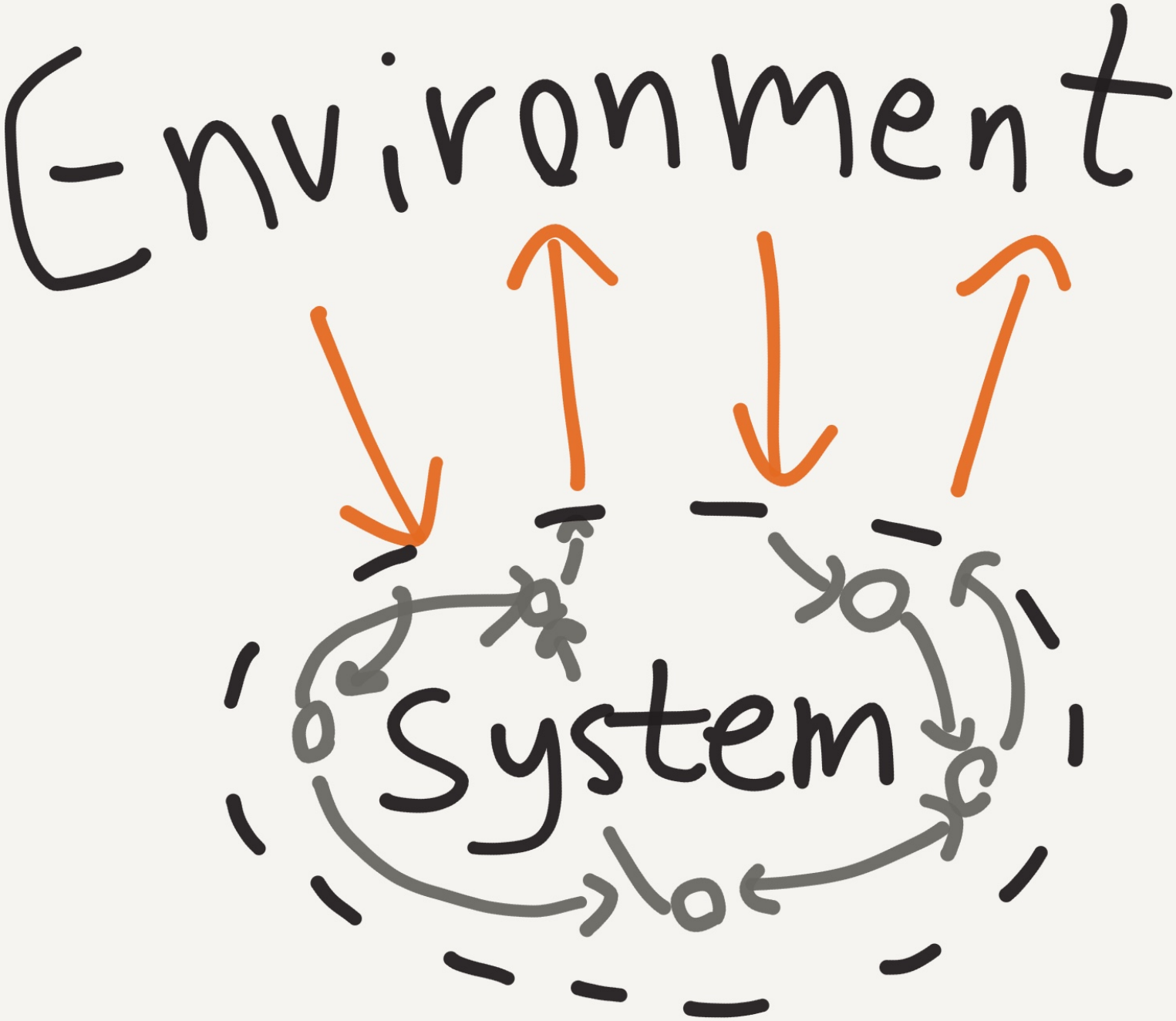
How do you
know if a
system is safe?

Measure your Systems

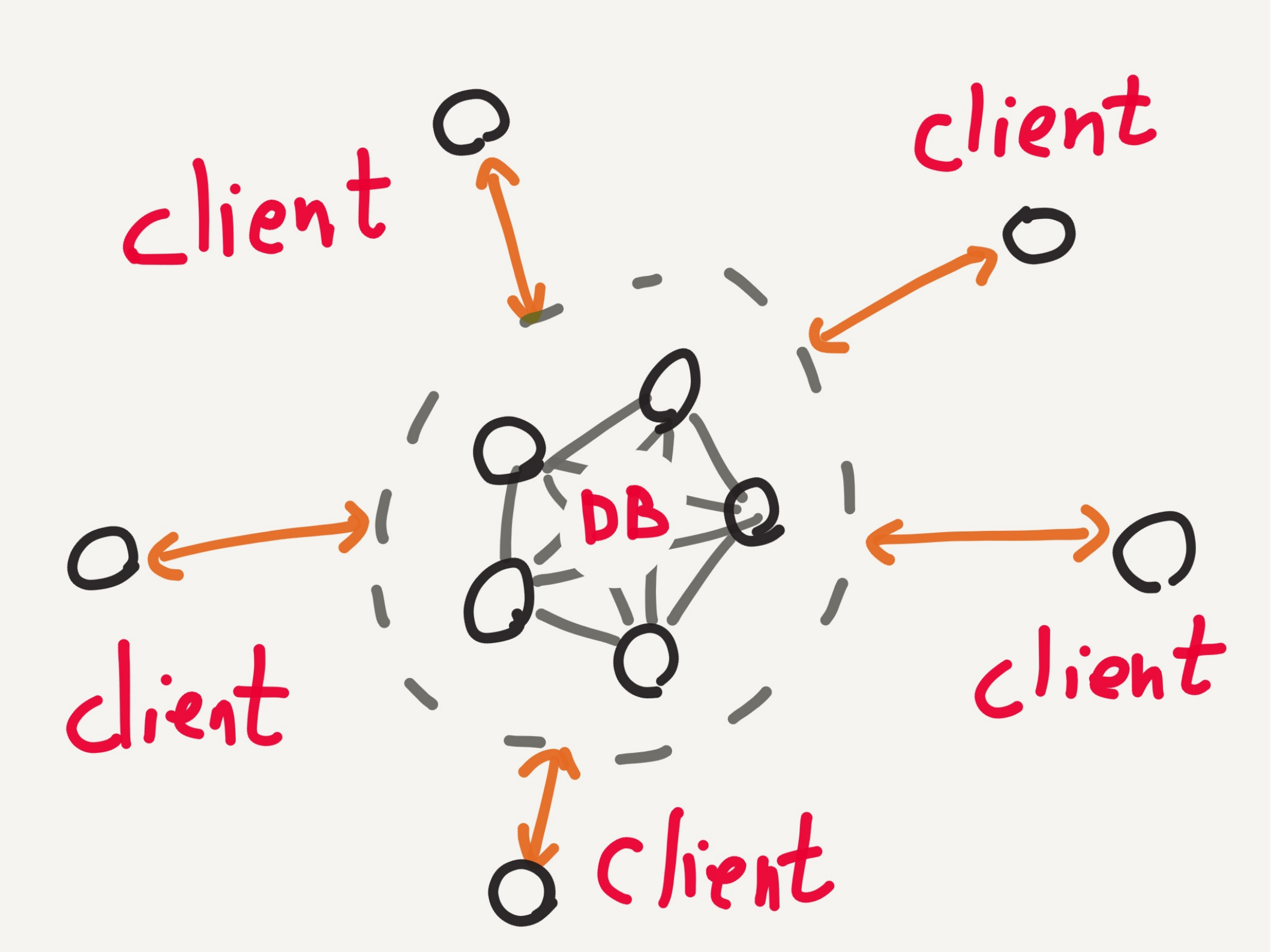


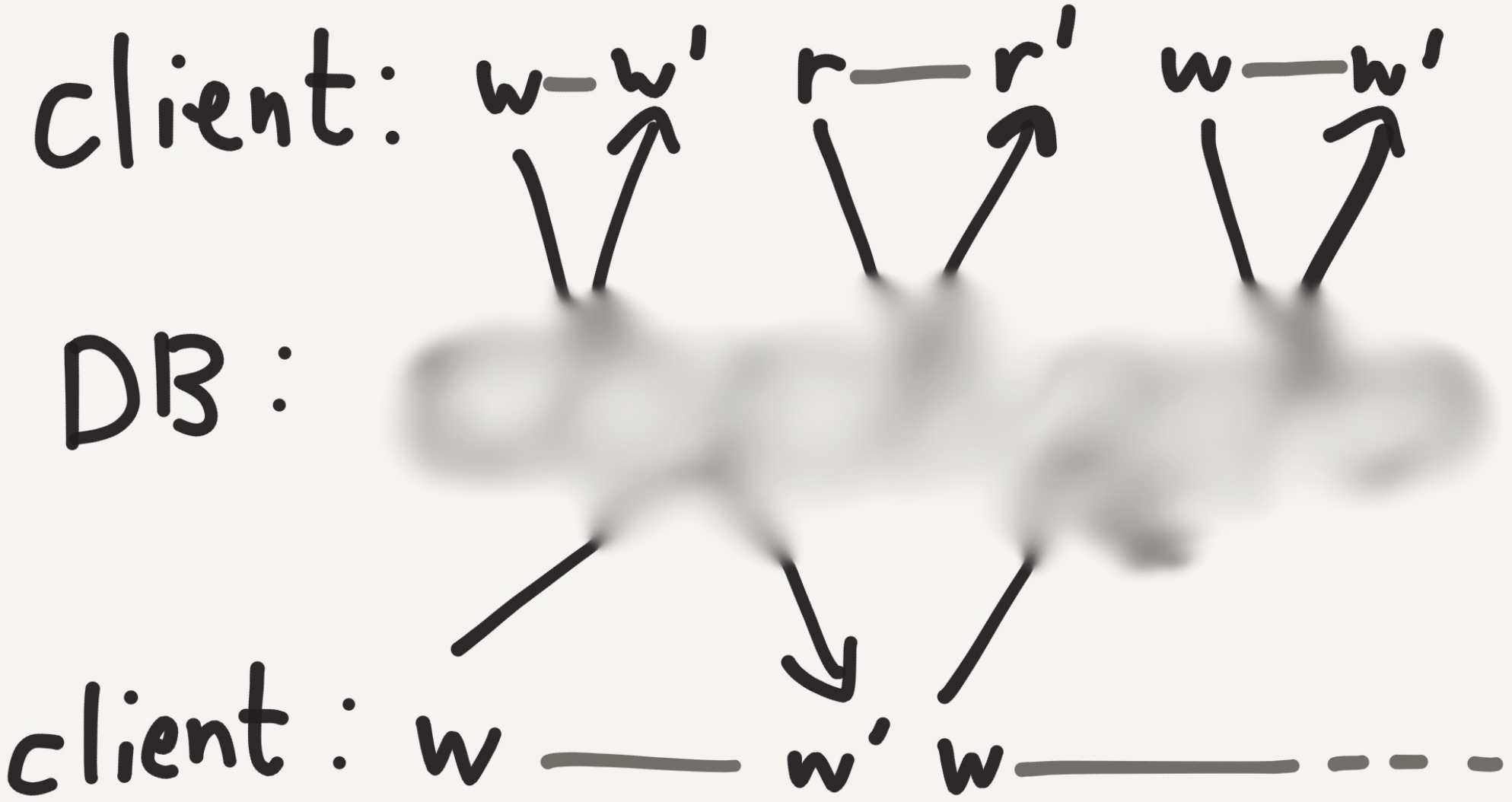
Jepsen

github.com/aphyr/jepsen


Environment
System

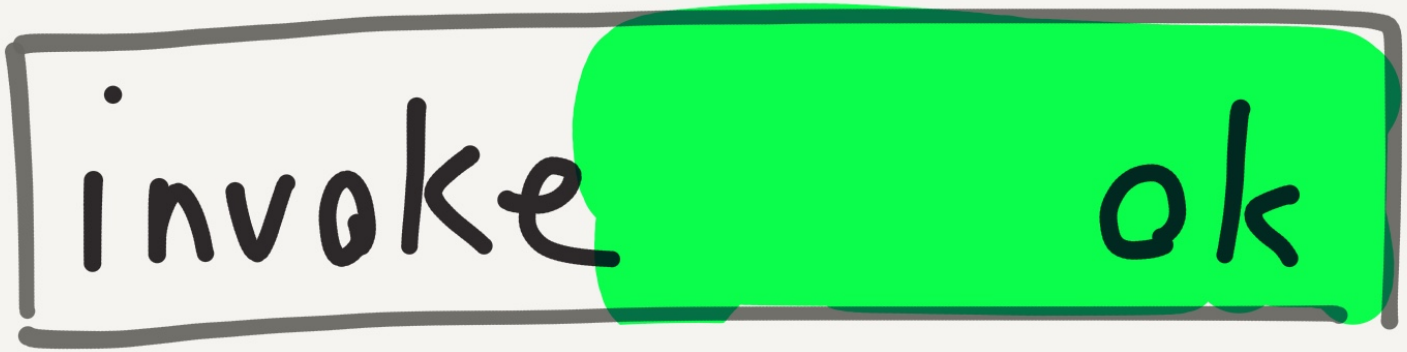

INVARIANTS

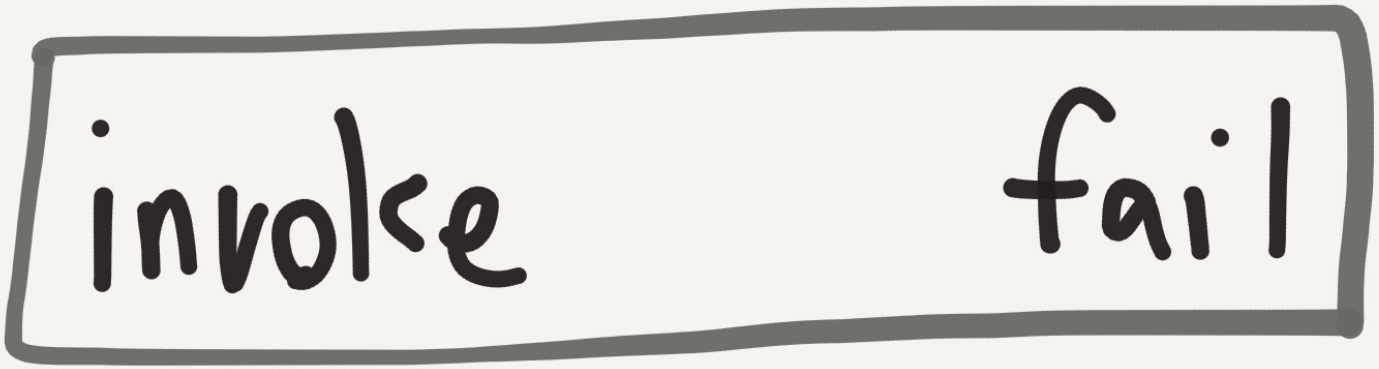

client
client
client
DB
client
client
client


client: w w' r r' w w'
DB:
client: w w' w

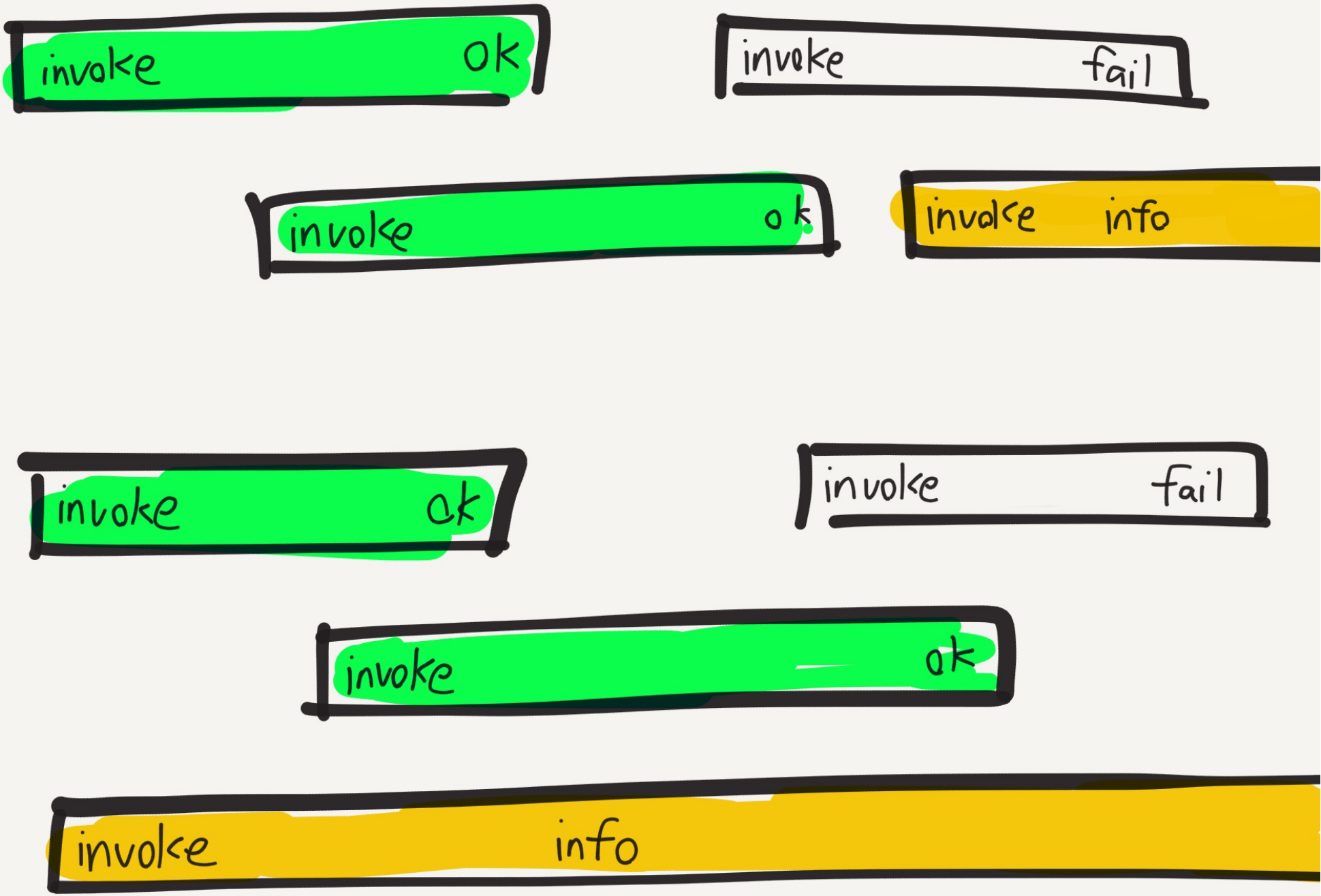
Clients Generate random operations (w) and apply them to the system

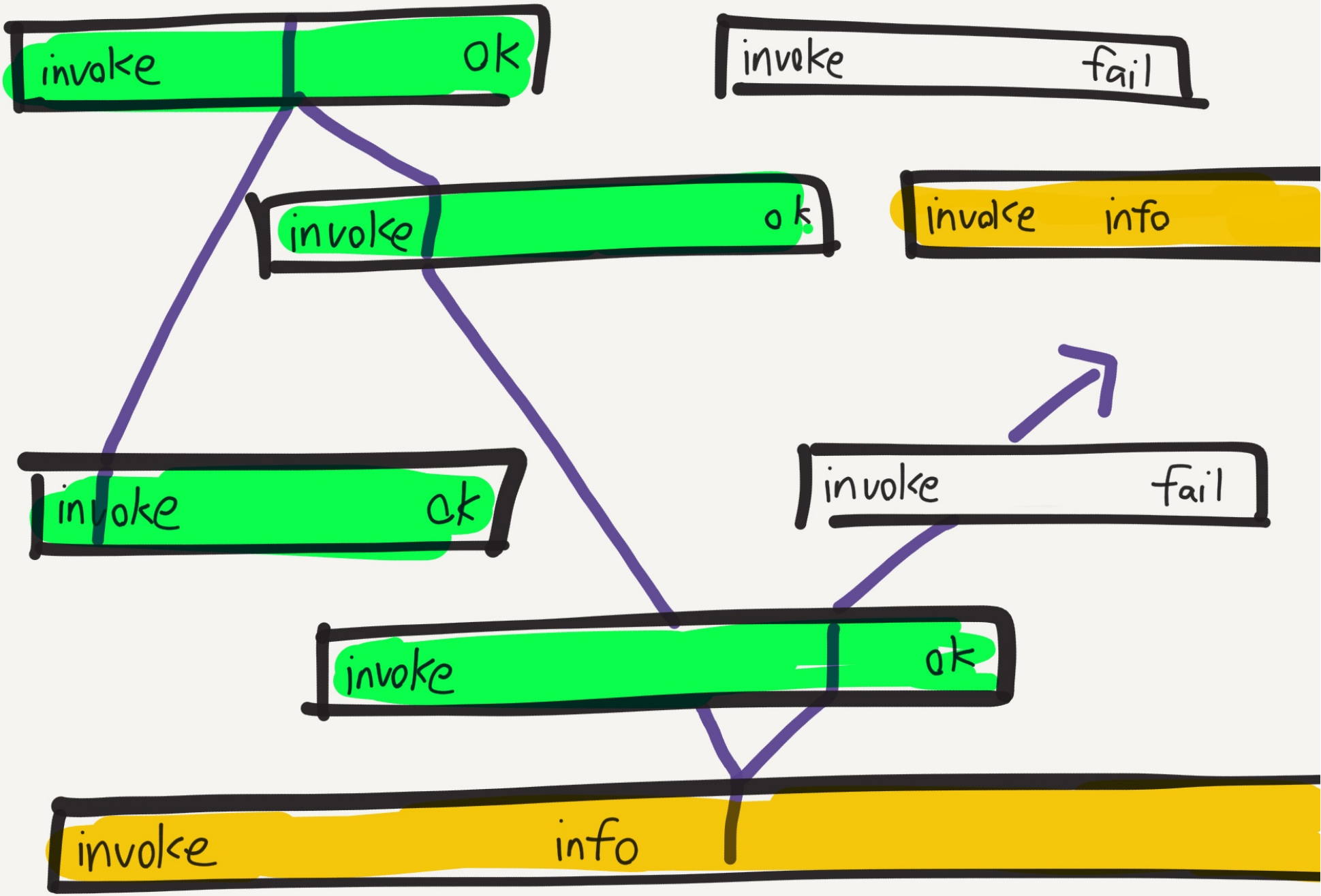
(w')


invoke
ok

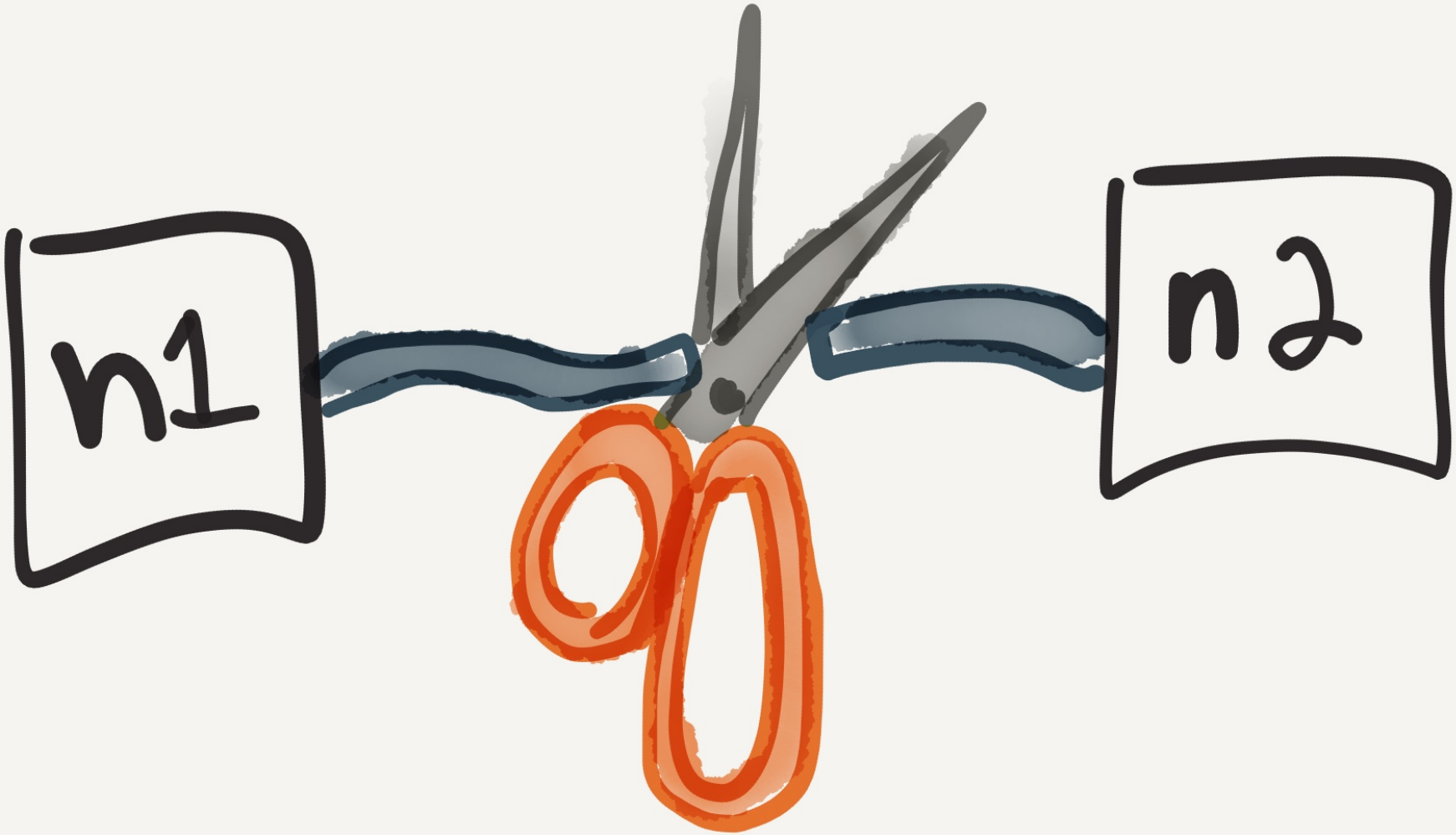

invoke
fail


invoke
info

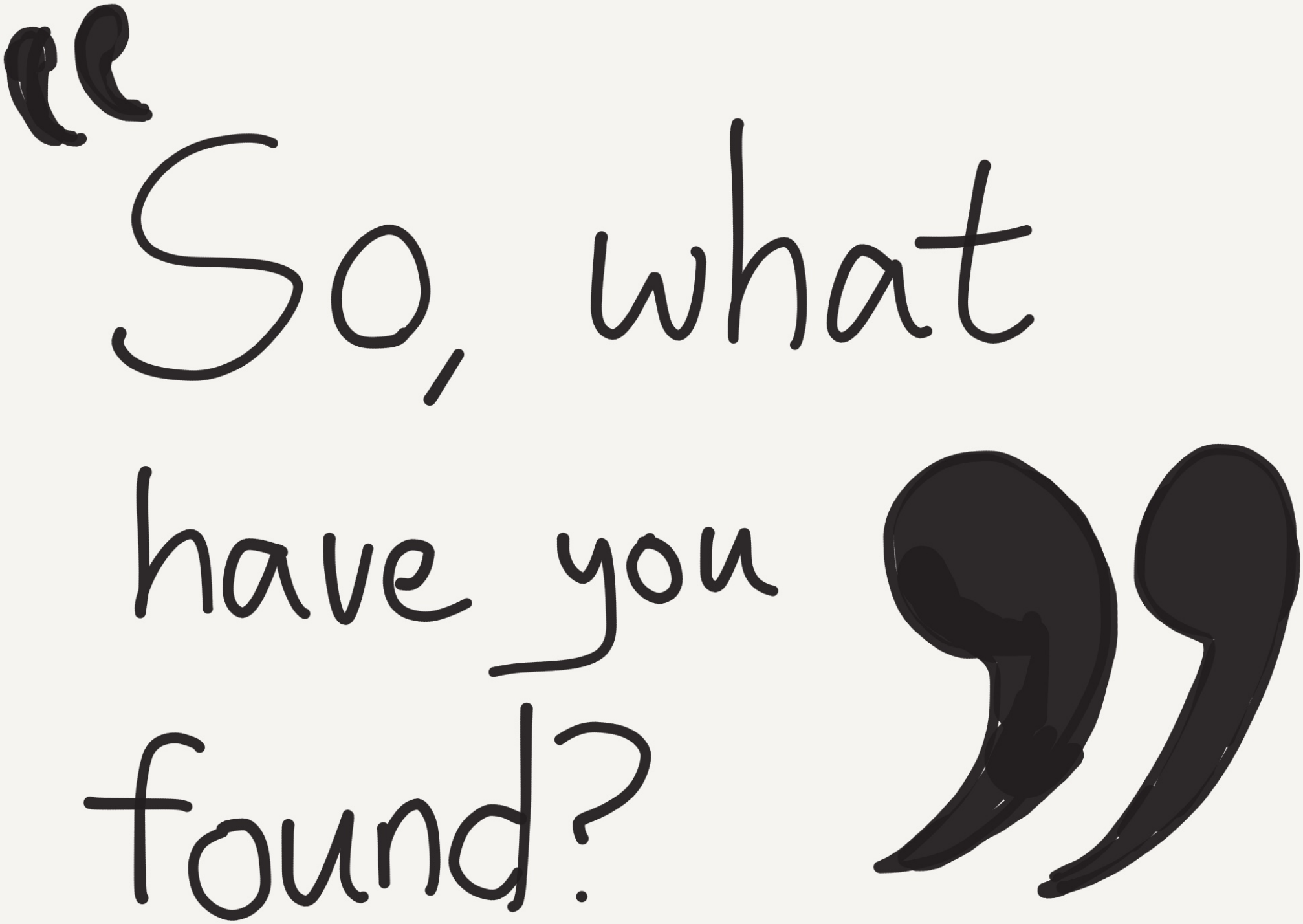

invoke
ok
invoke
fail
invoke
ok
invoke
info
invoke
ok
invoke
fail
invoke
ok
invoke
info


invoke
ok
invoke
fail
invoke
ok
invoke
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 → lost writes

CRDTs → safe

Mongo

Data loss at all

write concerns

Redis Sentinel

Split brain,
massive write loss

Cassandra

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

NuoDB

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

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

Still loses data

in every test case

MongoDB 2.6.7

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

Crate.io

2016

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

Cockroach DB

10/2016

- Double-insert
- Phantom Anomaly

6.3

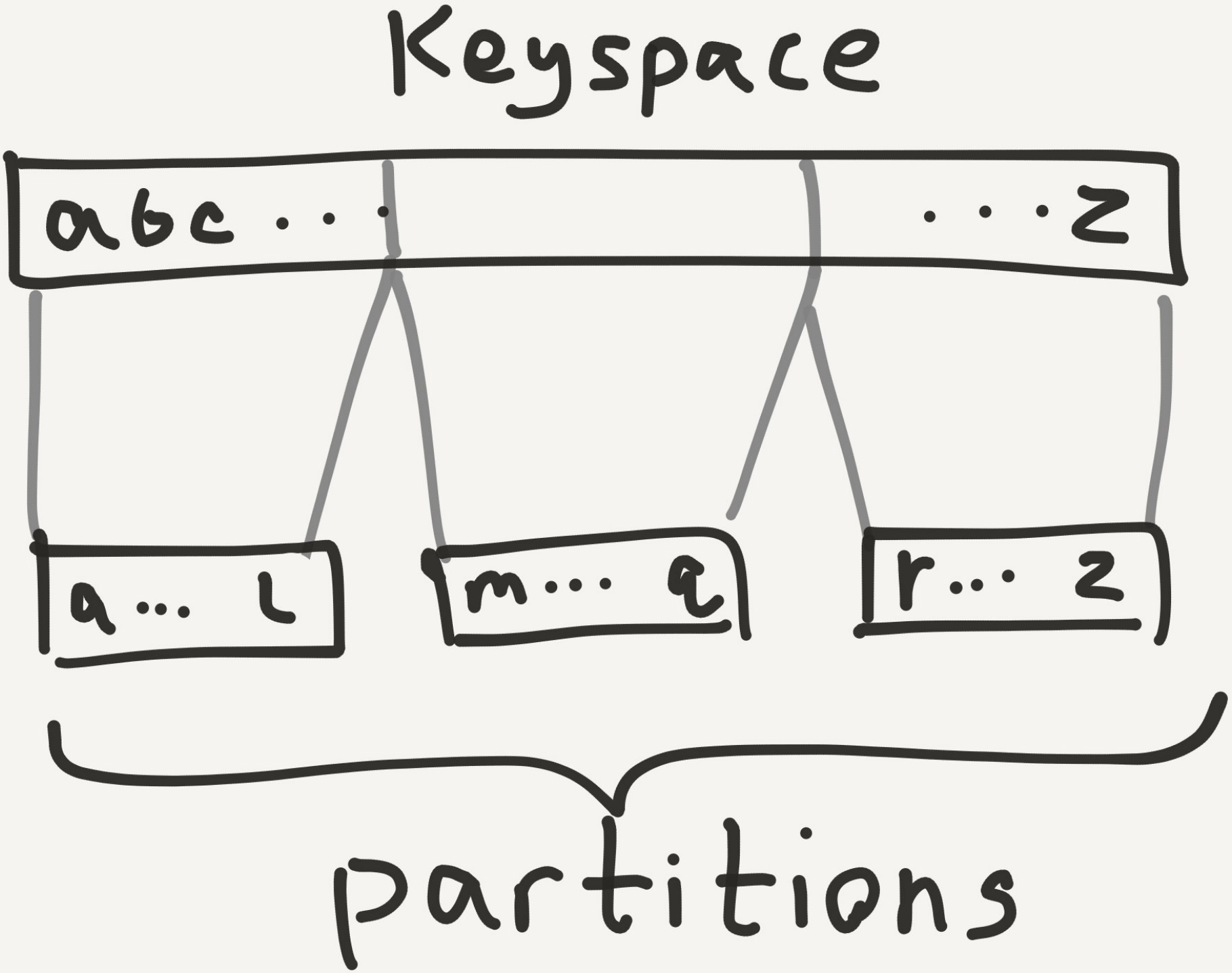
Distributed SQL Database

All data in RAM (but persistent)

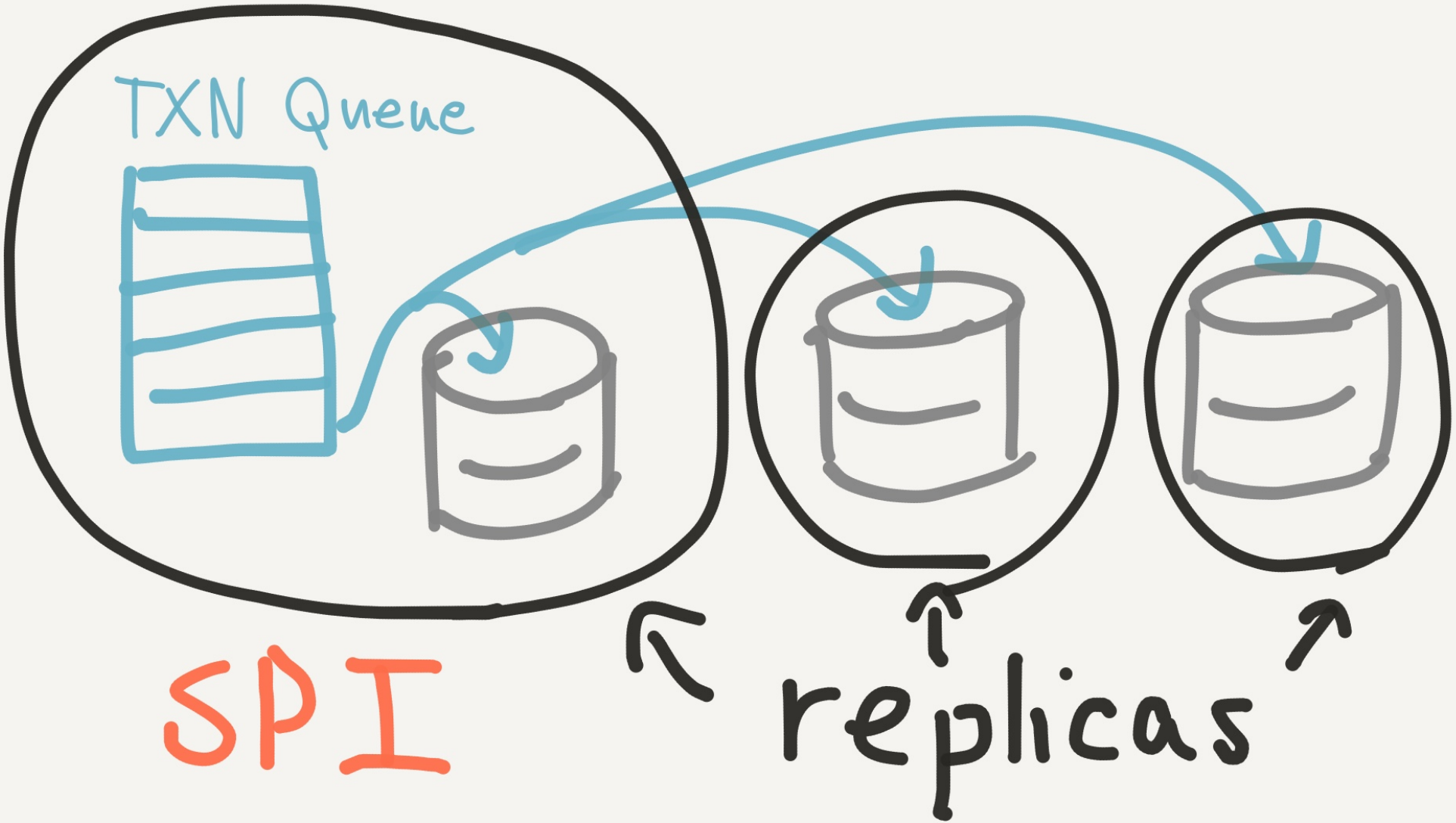
Basic SQL queries

Txns as Java stored procedures

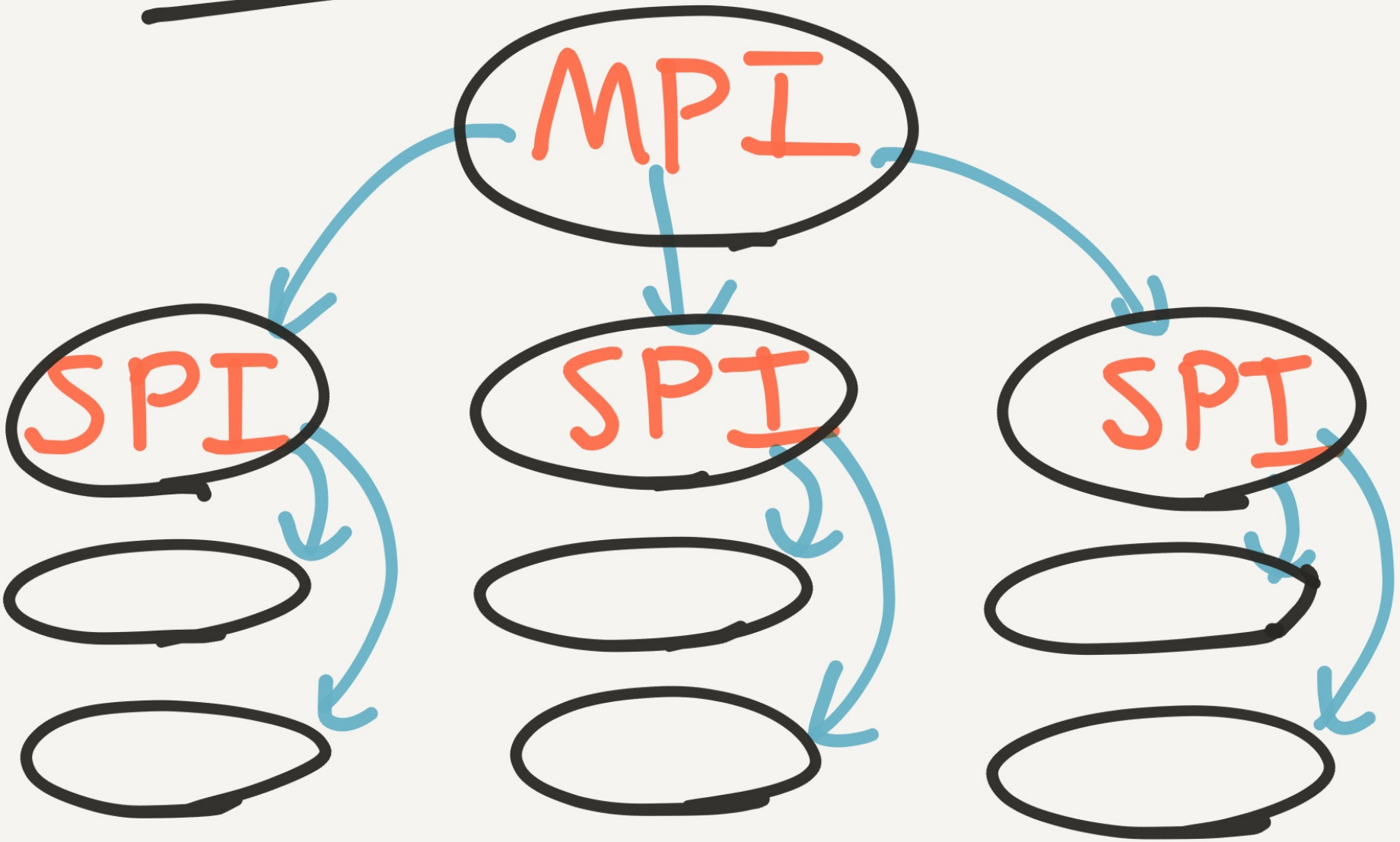
Intended for high-throughput, mostly sharded transactions


Keyspace
abc . . .
. . . z
a ... l
m ... q
r ... z
partitions

A partition

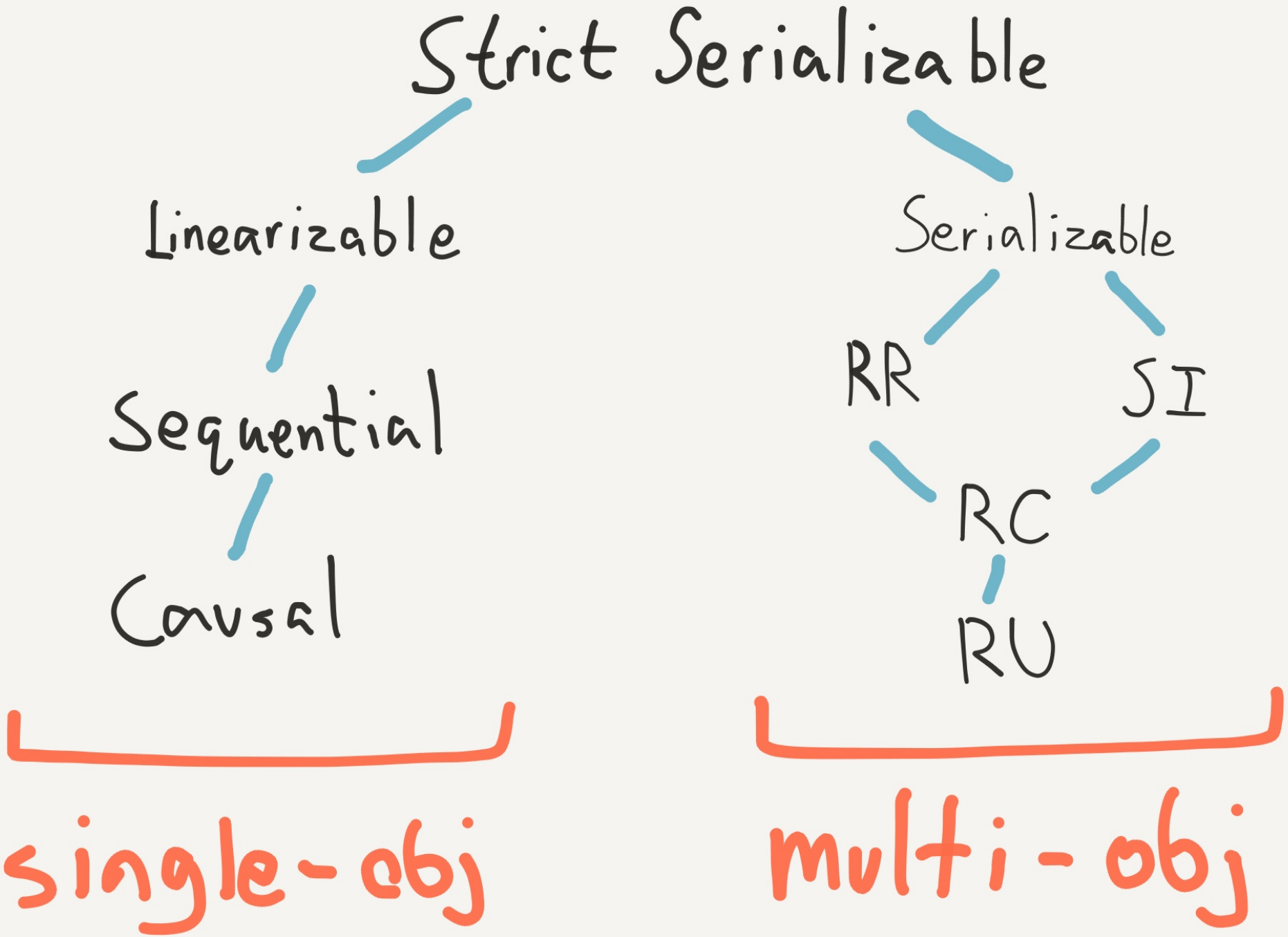


Across Partitions

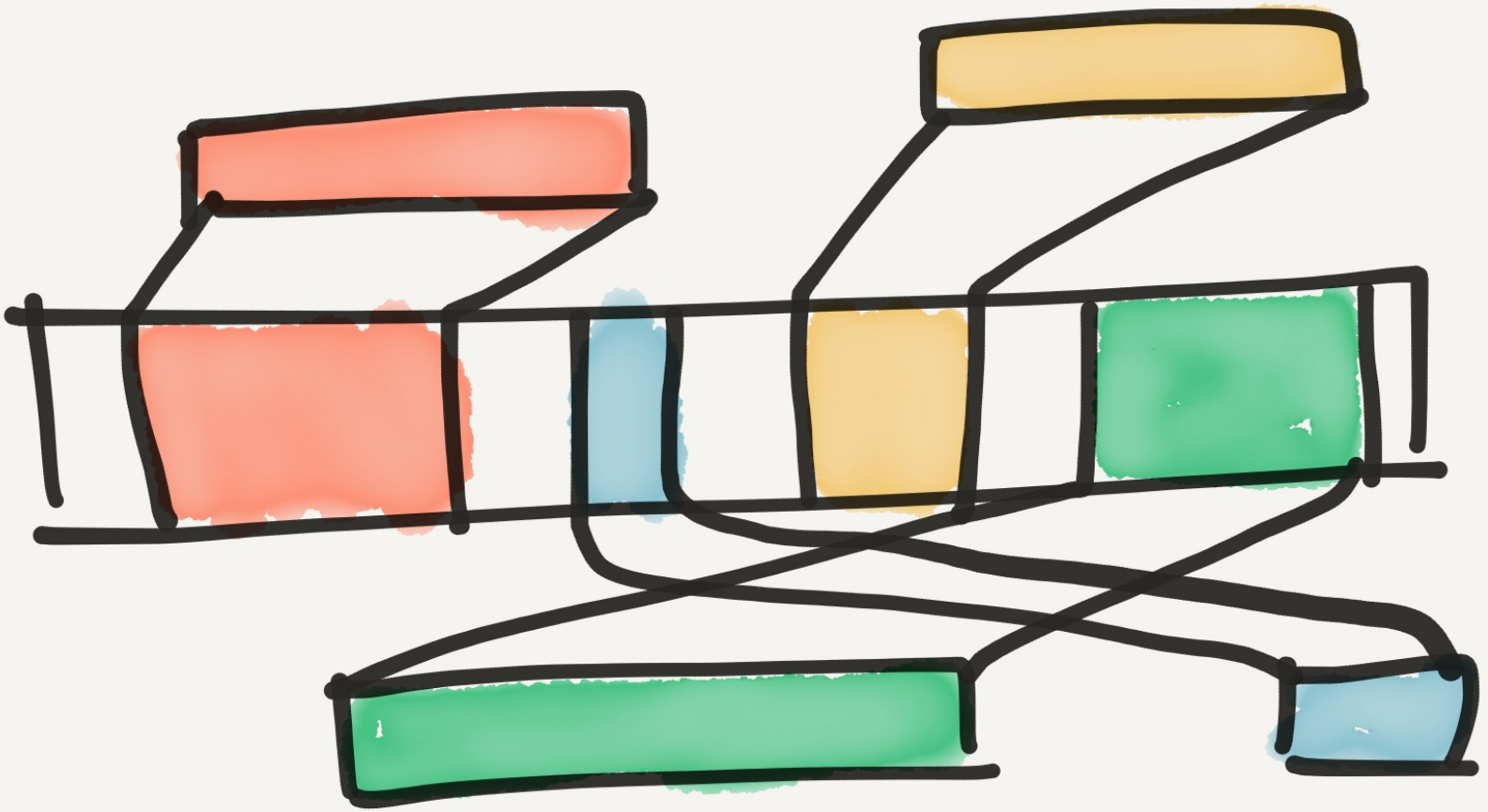


- Txns in 1 partition scale ~linearly
- Txns across partitions have const throughput

Claim: all txns are strict serializable!

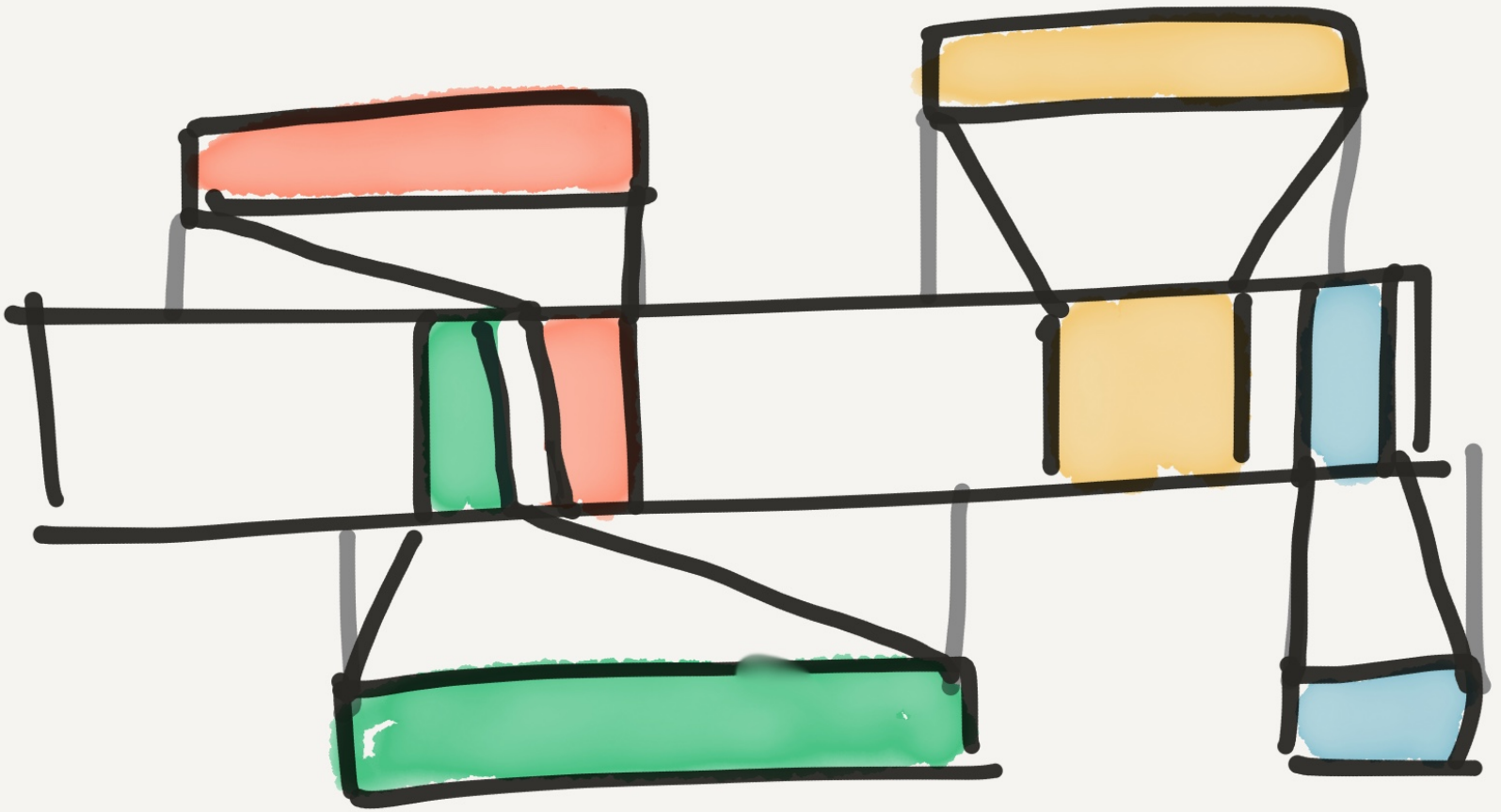

Strict Serializable
Linearizable
Sequential
Causal
Serializable
RR
SI
RC
RU
single-obj
multi-obj

Serializability



time →

Linearizability



time →

Strict Serializable

- Real-time
- Multi-object

VoltDB 6.3

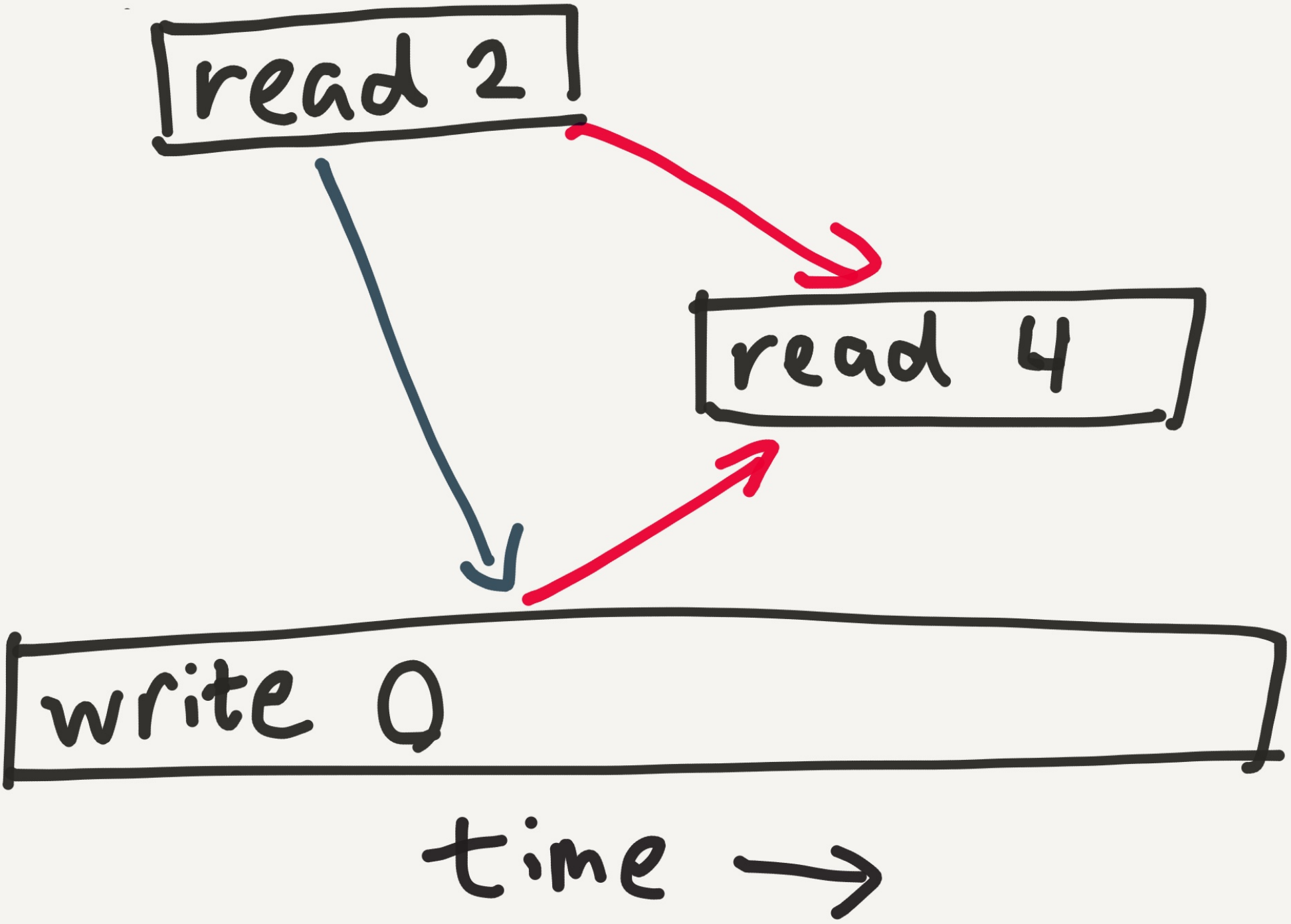
exhibits several violations of strict serializability

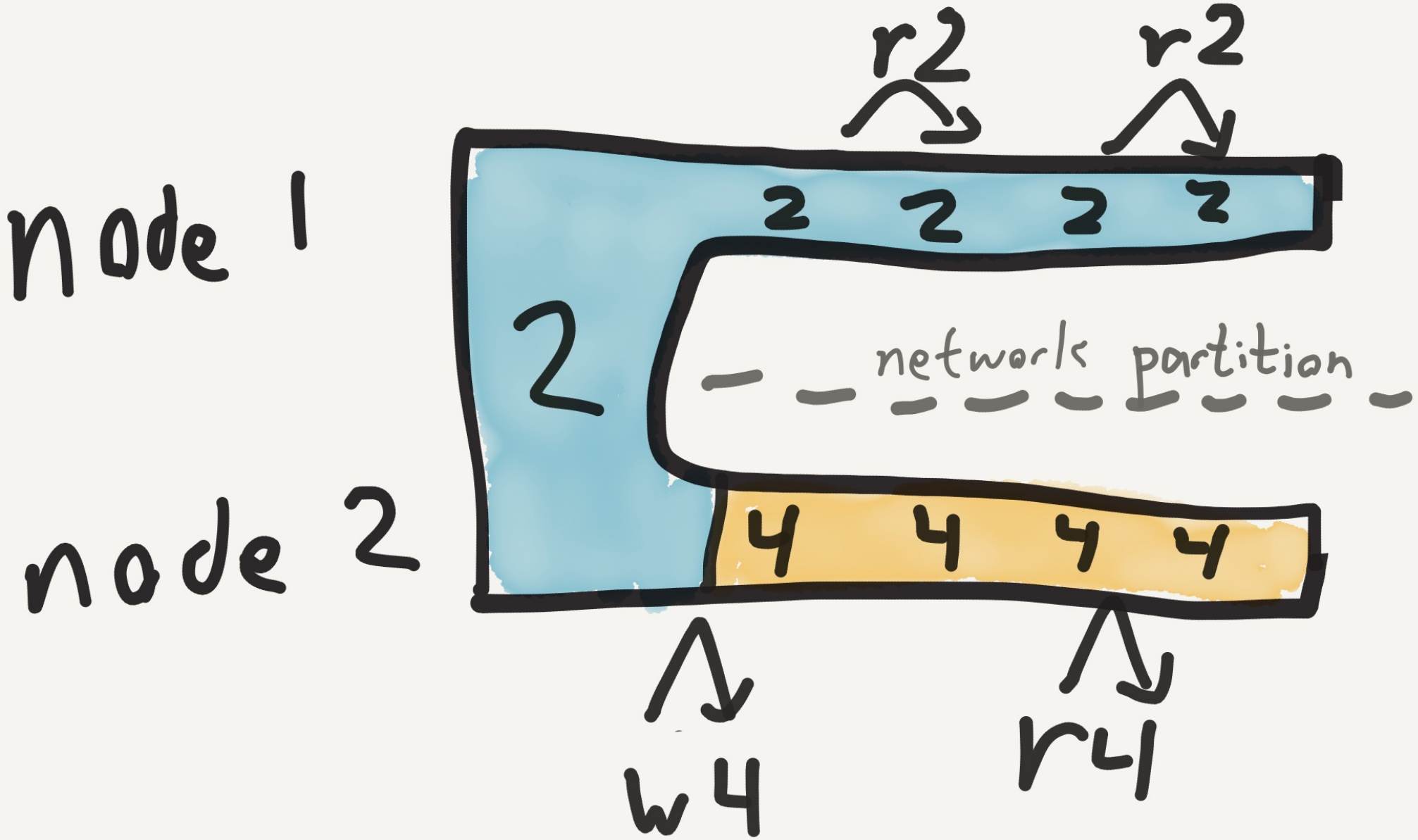
Stale Reads

Dirty Reads

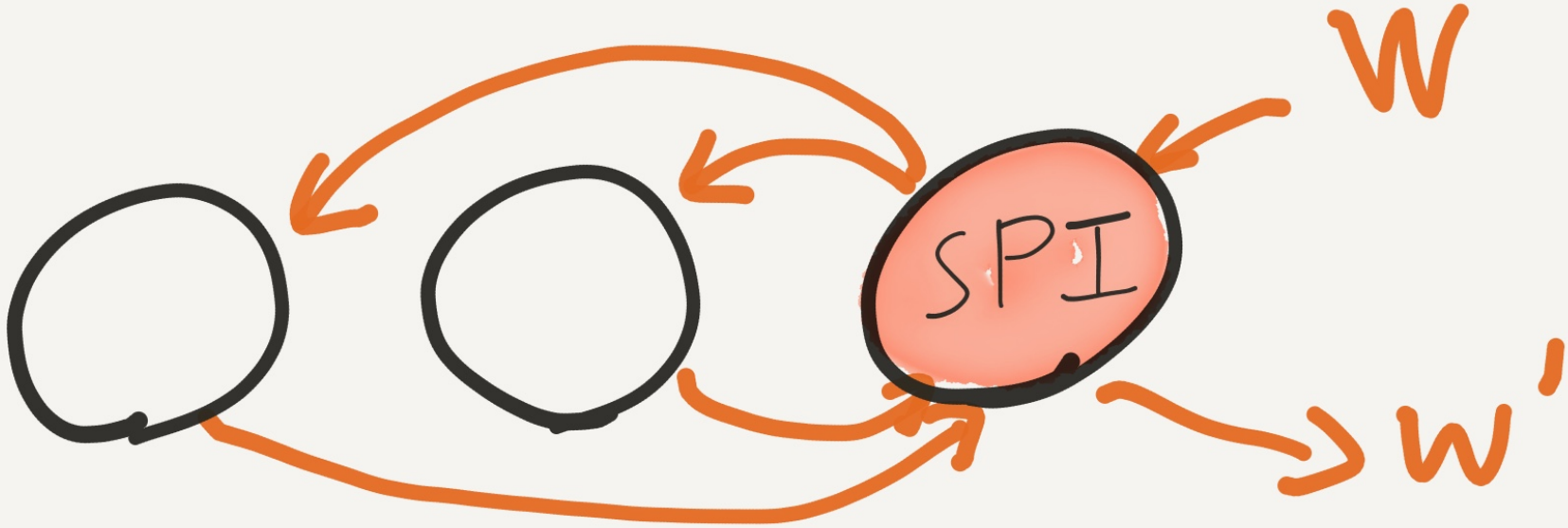
Lost Updates

Stale Reads


read 2
read 4
write 0
time

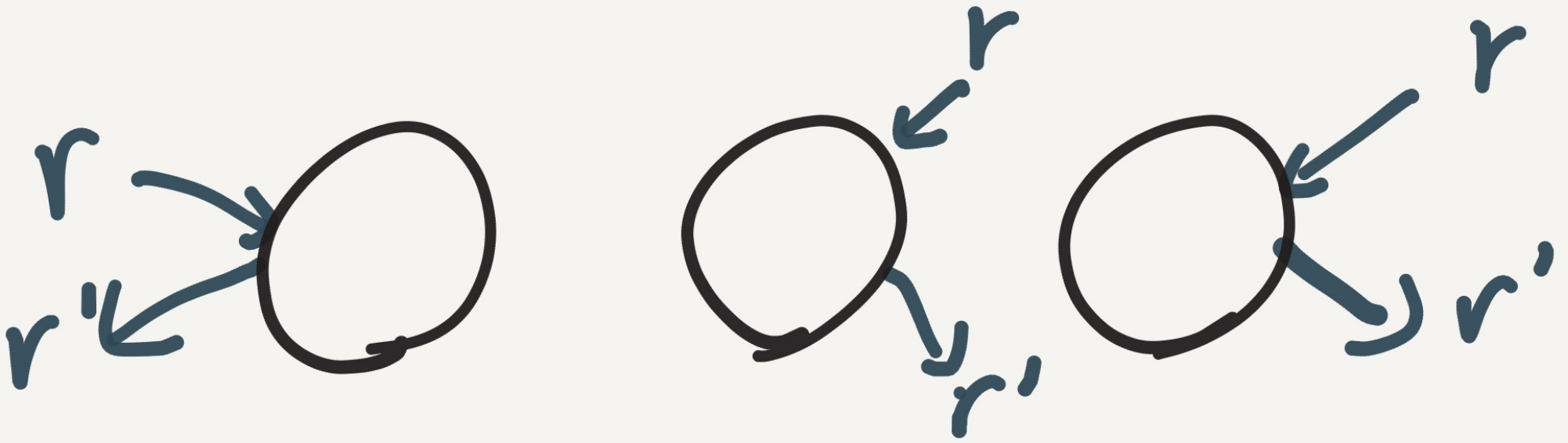

node 1
node 2
r2
r2
2 2 2 2
2
network partition
4 4 4 4
w4
r4

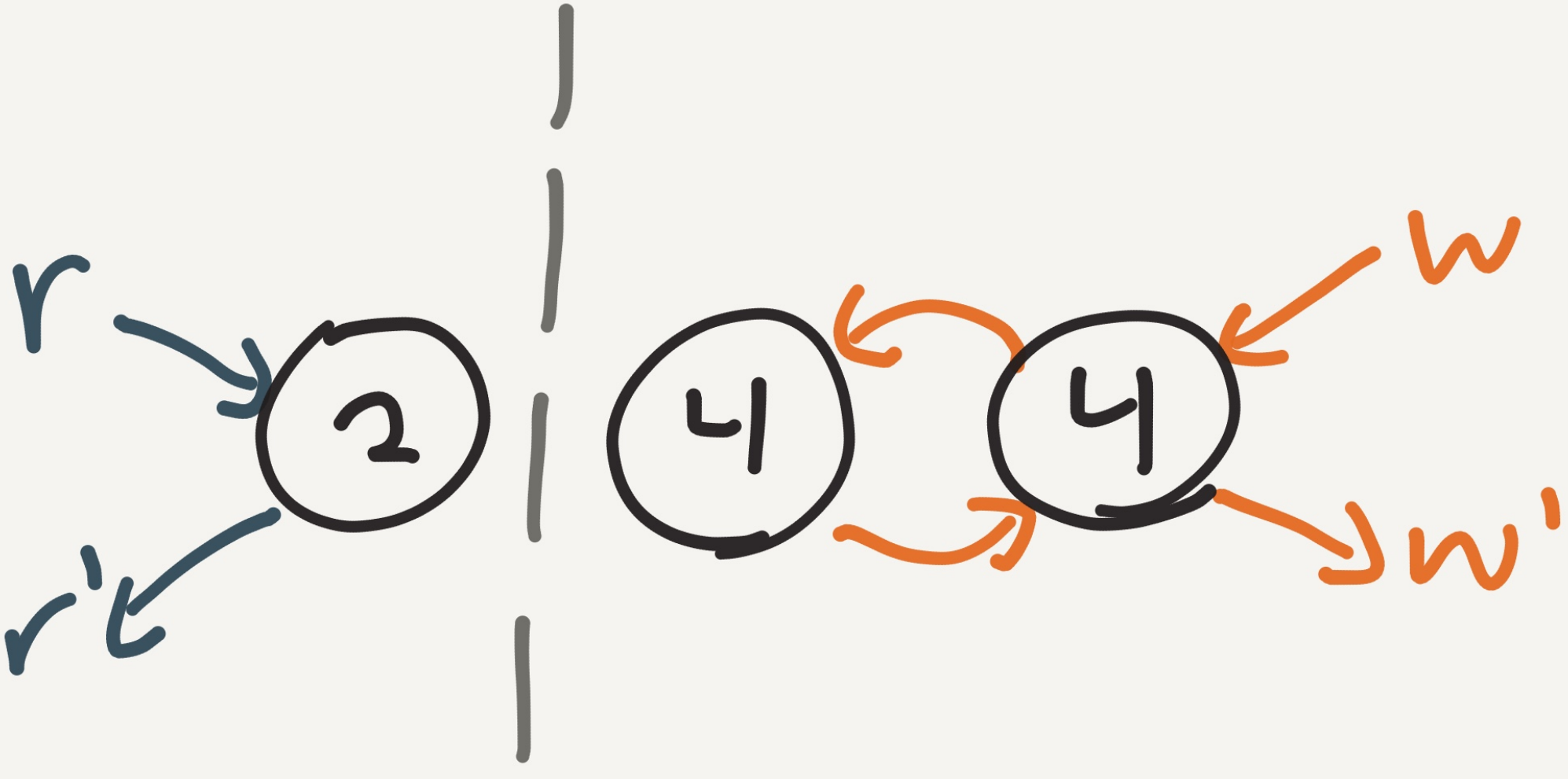
For a given partition



SPI orders updates

...but not reads

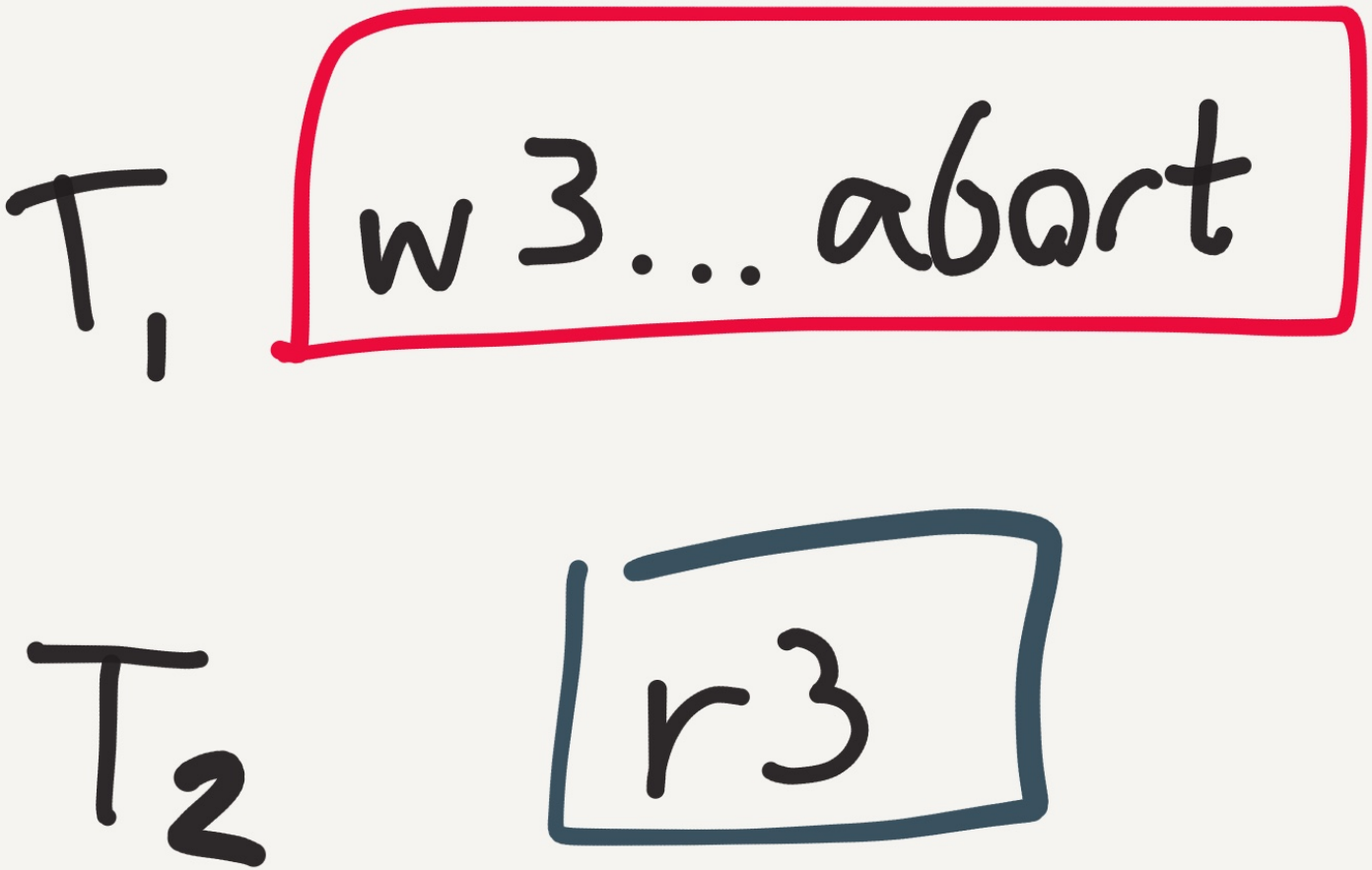


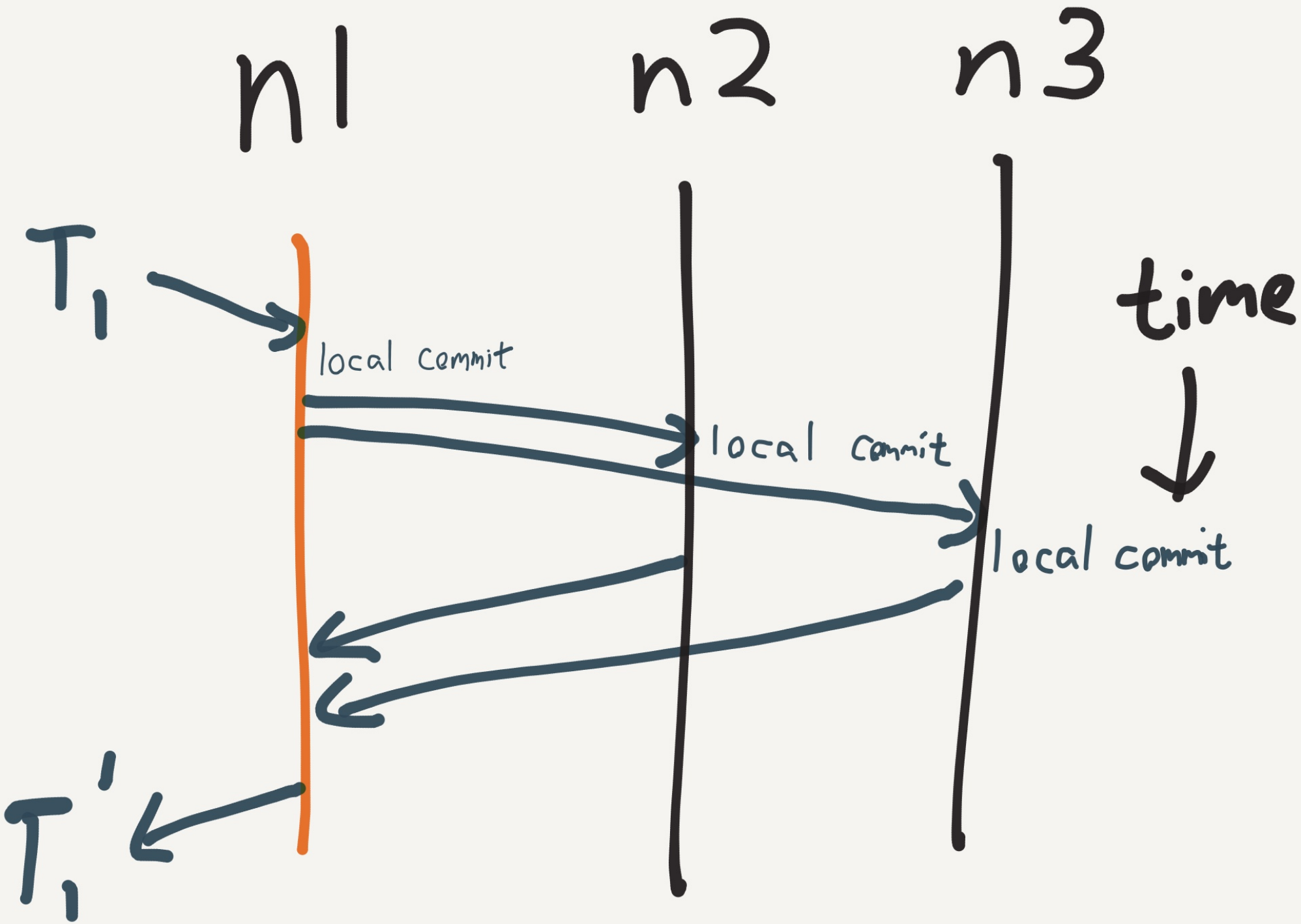

r
2
4
4
w
r'
w'

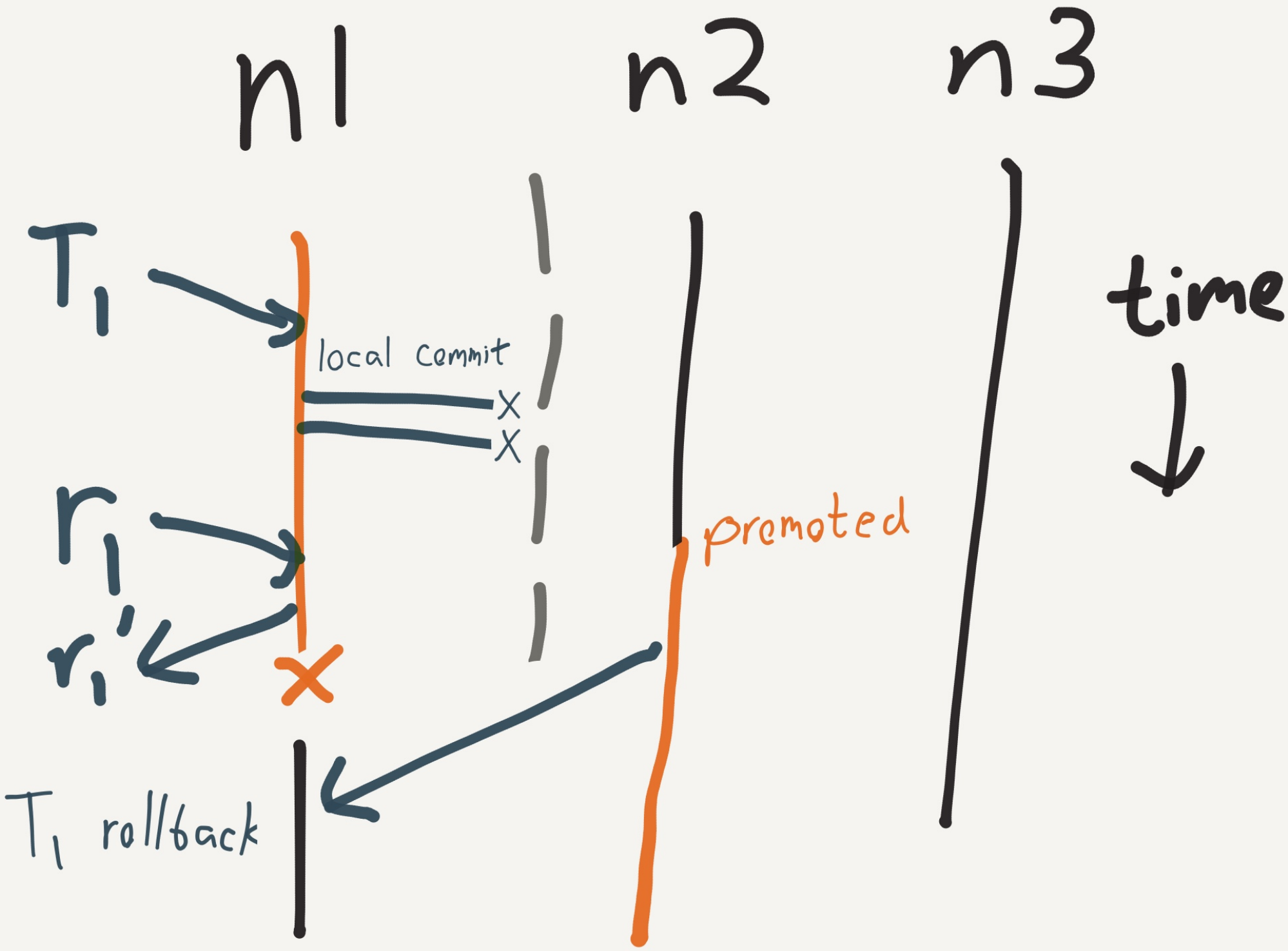
Solution:

Push all txns through the SPI ordering path

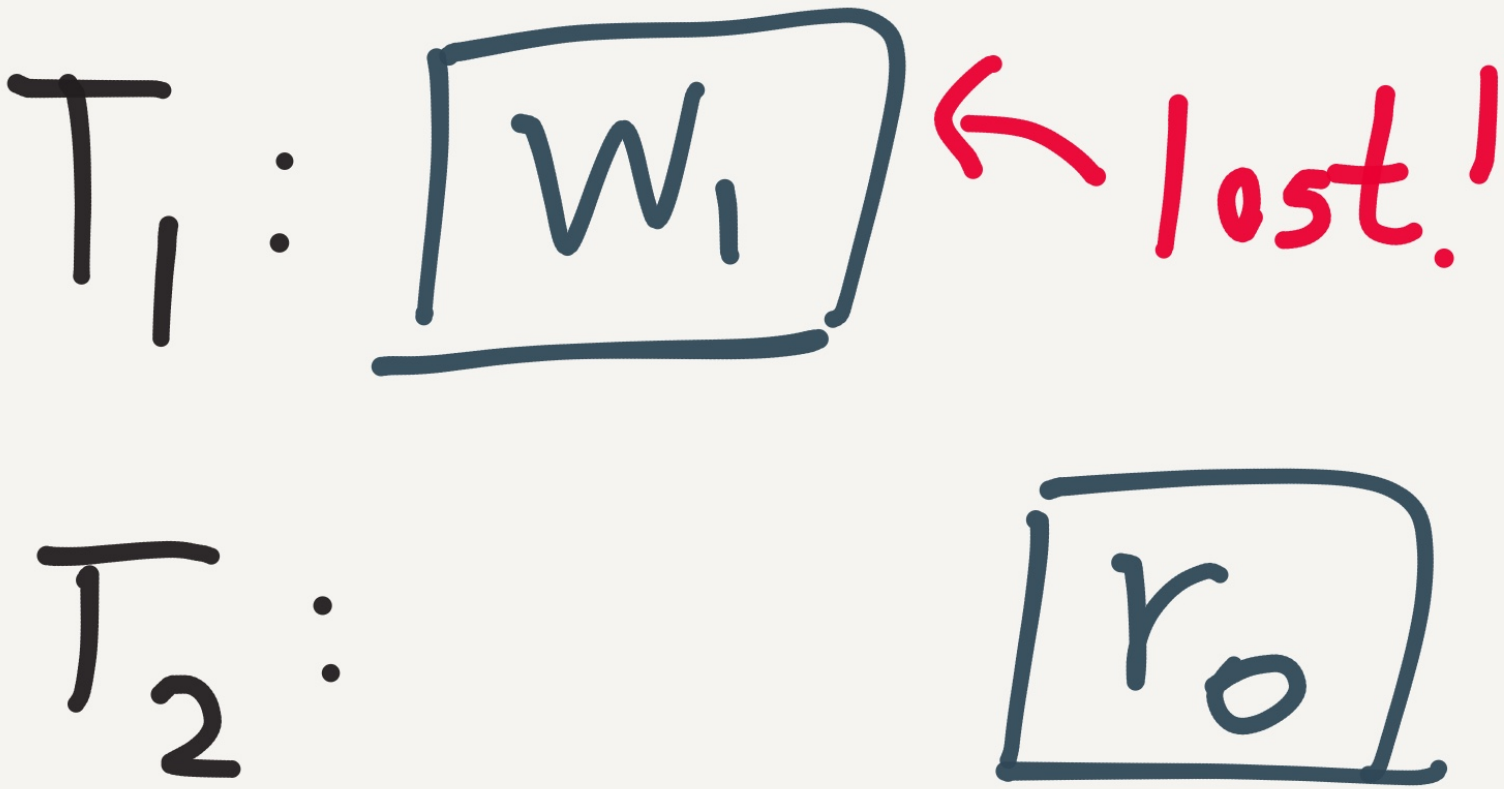

Dirty
Reads

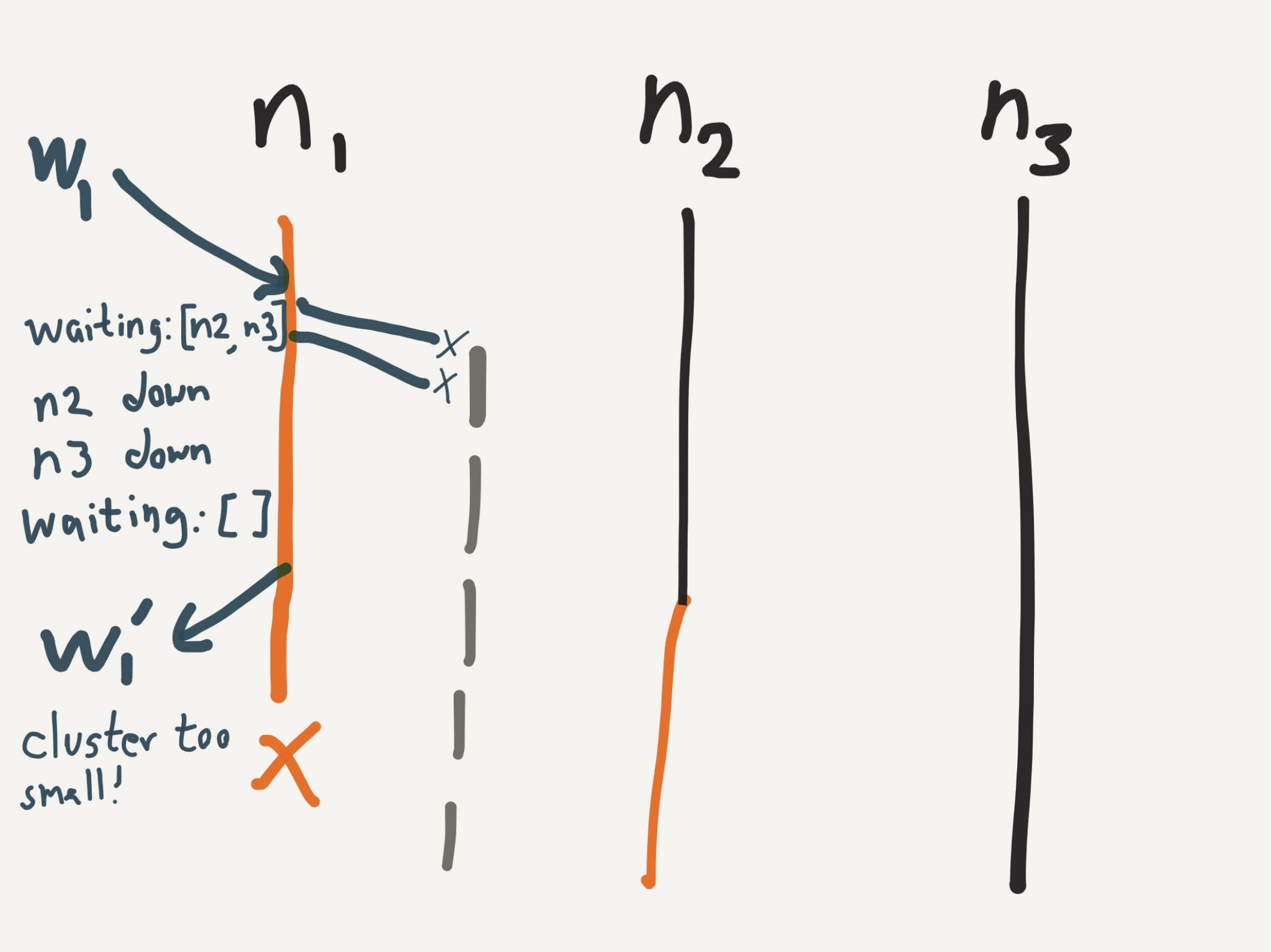

T1
w3... abort
T2
r3


n1
n2
n3
T1
local commit
local commit
local commit
time
T1'


n1
n2
n3
time
T_1
local commit
X
X
r_1
r_1'
promoted
T_1 rollback

Lost Updates


T_1 :
w_1
← lost!
T_2 :
r_0


n_1
n_2
n_3
w_1
waiting: [n2, n3]
n2 down
n3 down
waiting: [ ]
w_1'
cluster too small!

"Zookeeper" watch
are asynchronous;
race condition
between ack & shutdown

Solution:

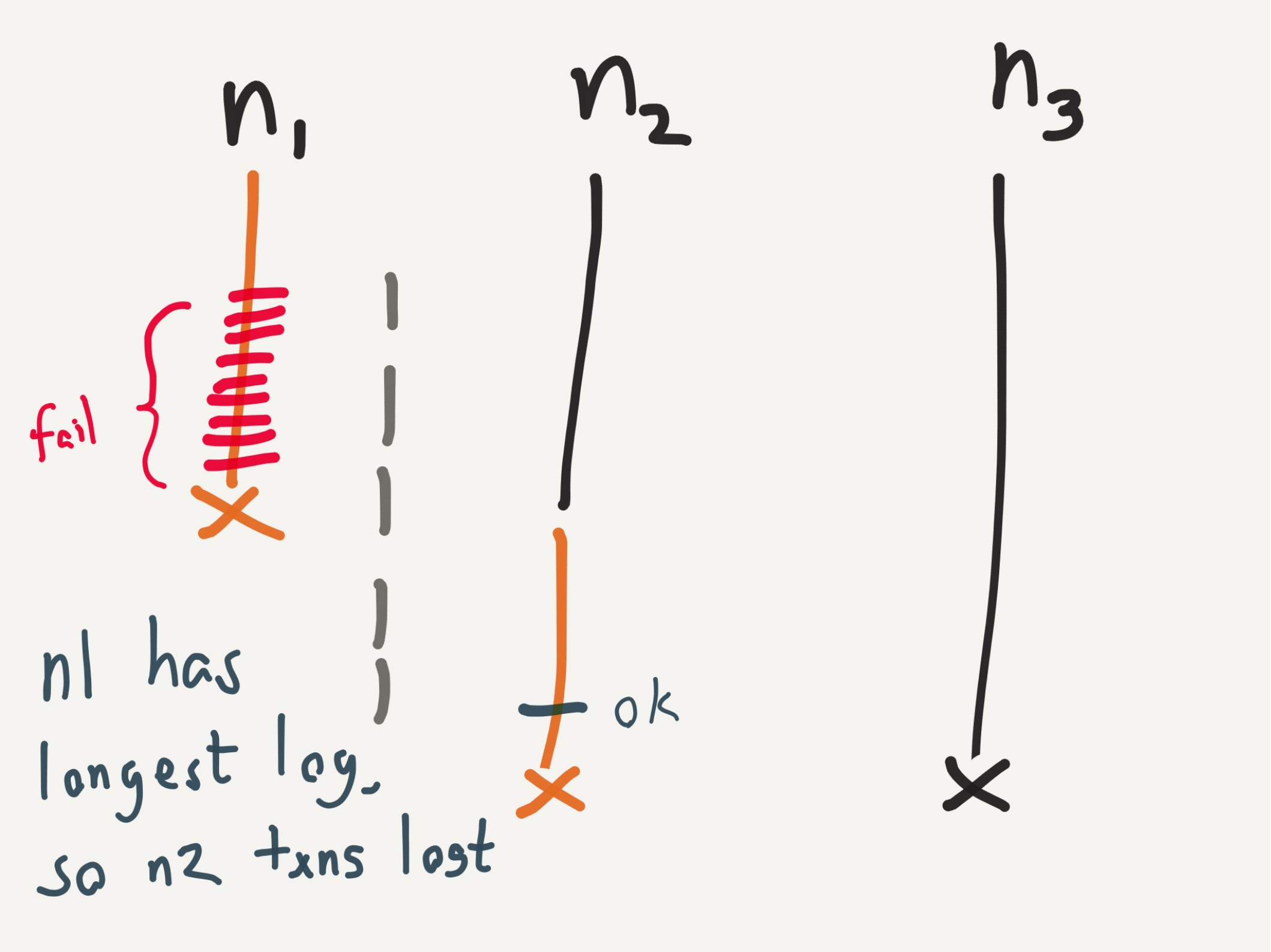
Make cluster shrink

฿ shutdown one

atomic step

Still loses
acknowledged
writes...

On crash recovery, Volt picks longest log as authoritative.


n1
n2
n3
fail
ok
n1 has
longest log.
so n2 txns lost

Solution:

Reconstruct cluster state from fault logs

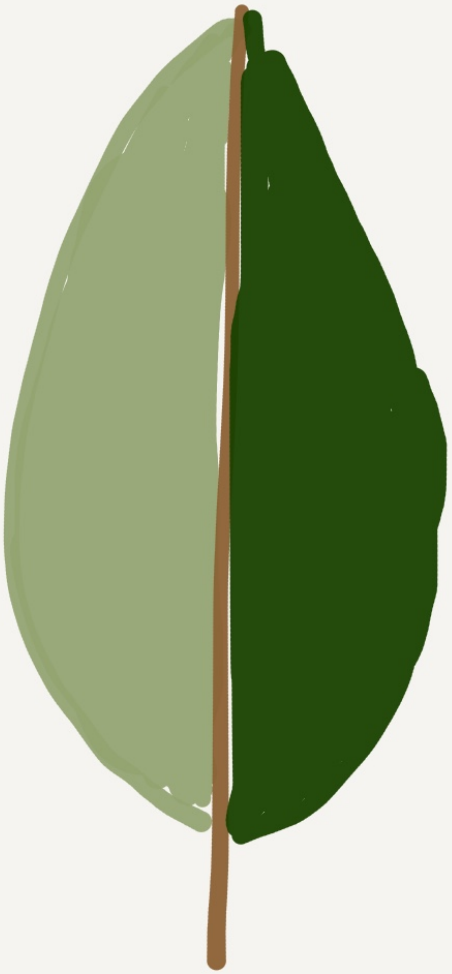
Multi-Partition Transactions

No faults found

MPI fully serializes read and write txns, may prevent divergence

VoltDB 6.4

Passes all Jepsen tests for strict serializability



mongoDB

3.4.0-rc3

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

2013

Lost writes

2015

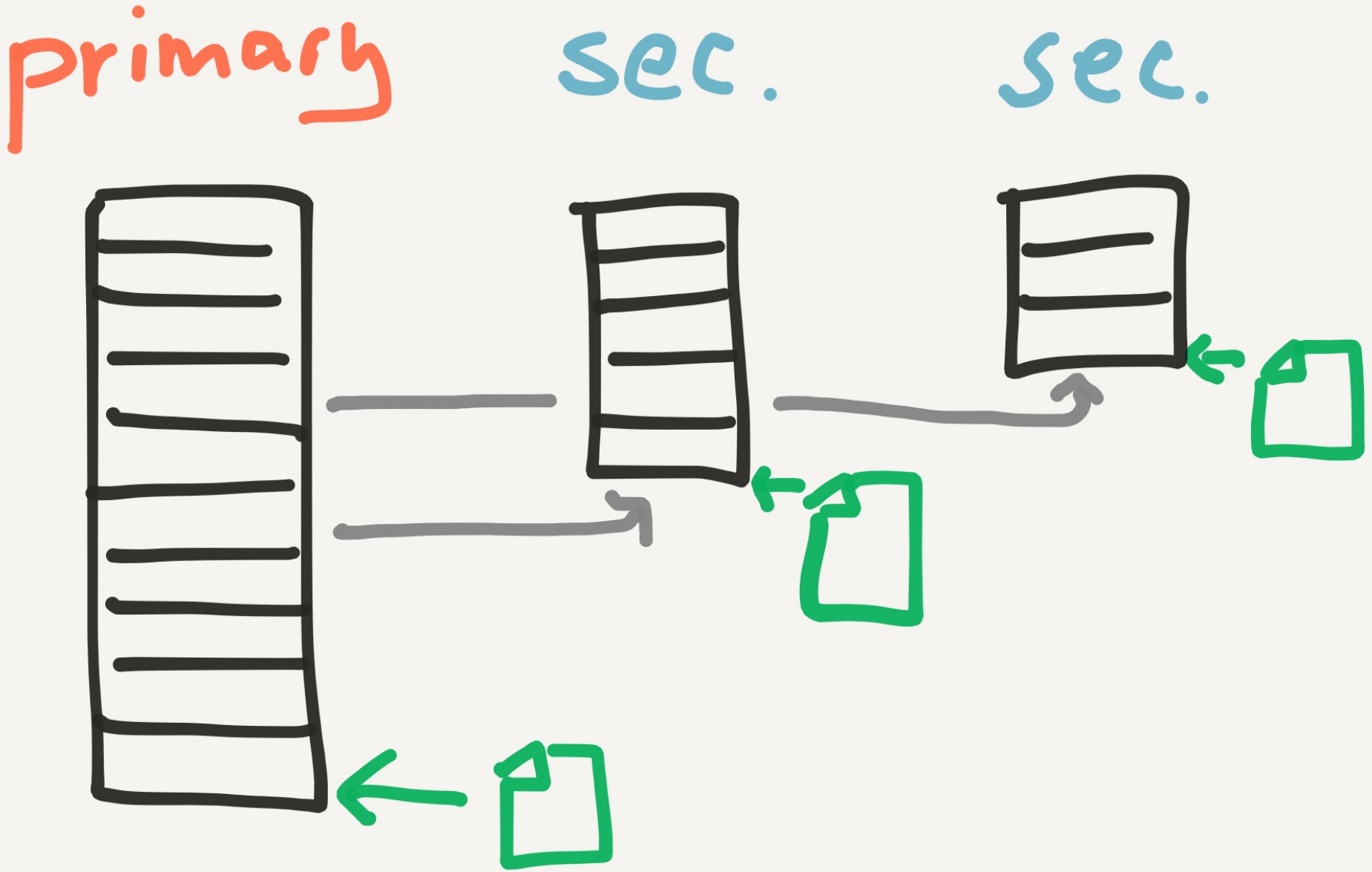
Safe writes

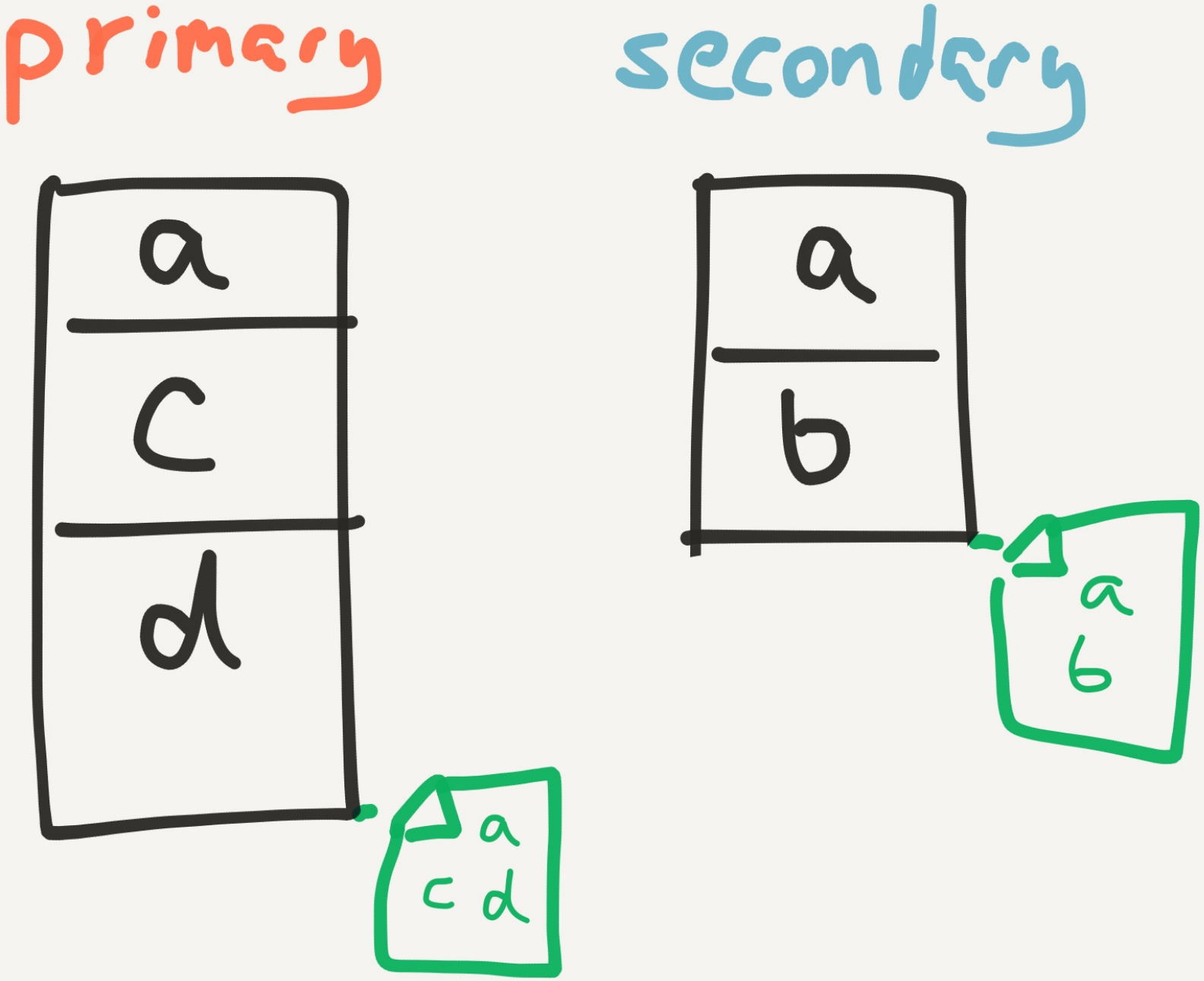
Dirty/stale reads

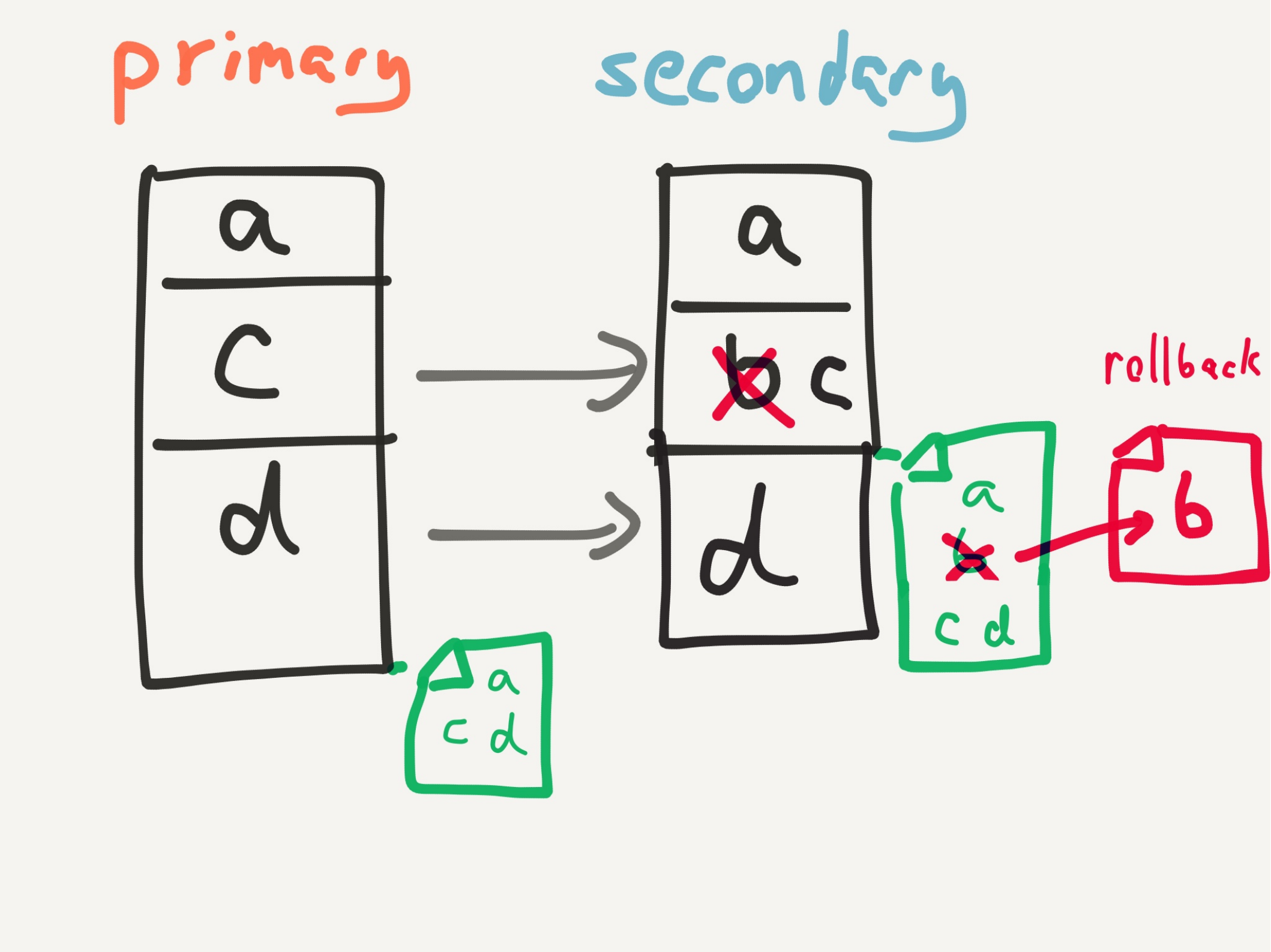
3.0: WiredTiger Replication v1

3.2: Majority reads

3.4: Linearizable reads


primary
sec.
sec.

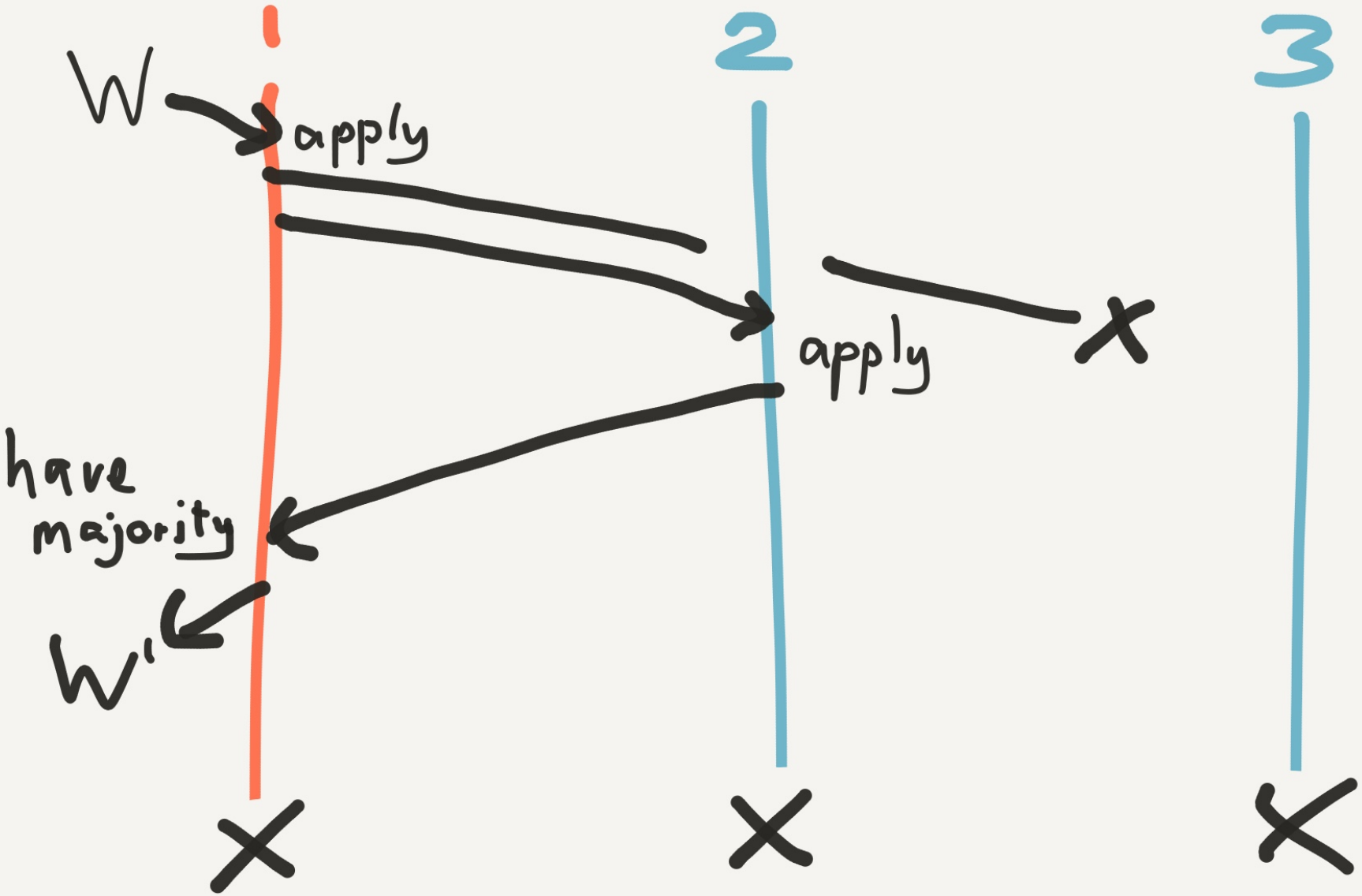

primary
a
c
d
a
c d
secondary
a
b
a
b


primary
secondary
a
c
d
a
b c
d
rollback
a
c d
a
c d
b

v0 protocol:

- Dirty reads
- Lost writes?

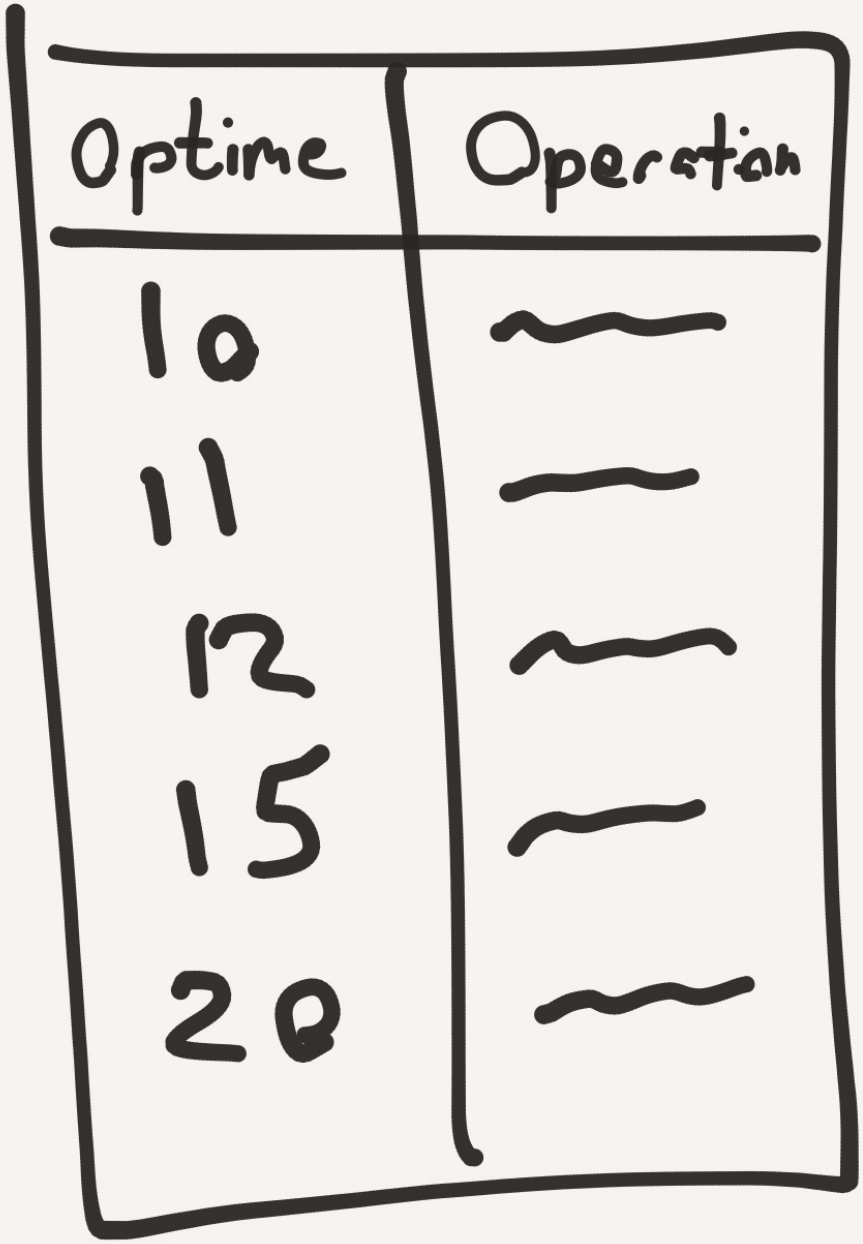
Ok but how bad are dirty reads anyway?


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

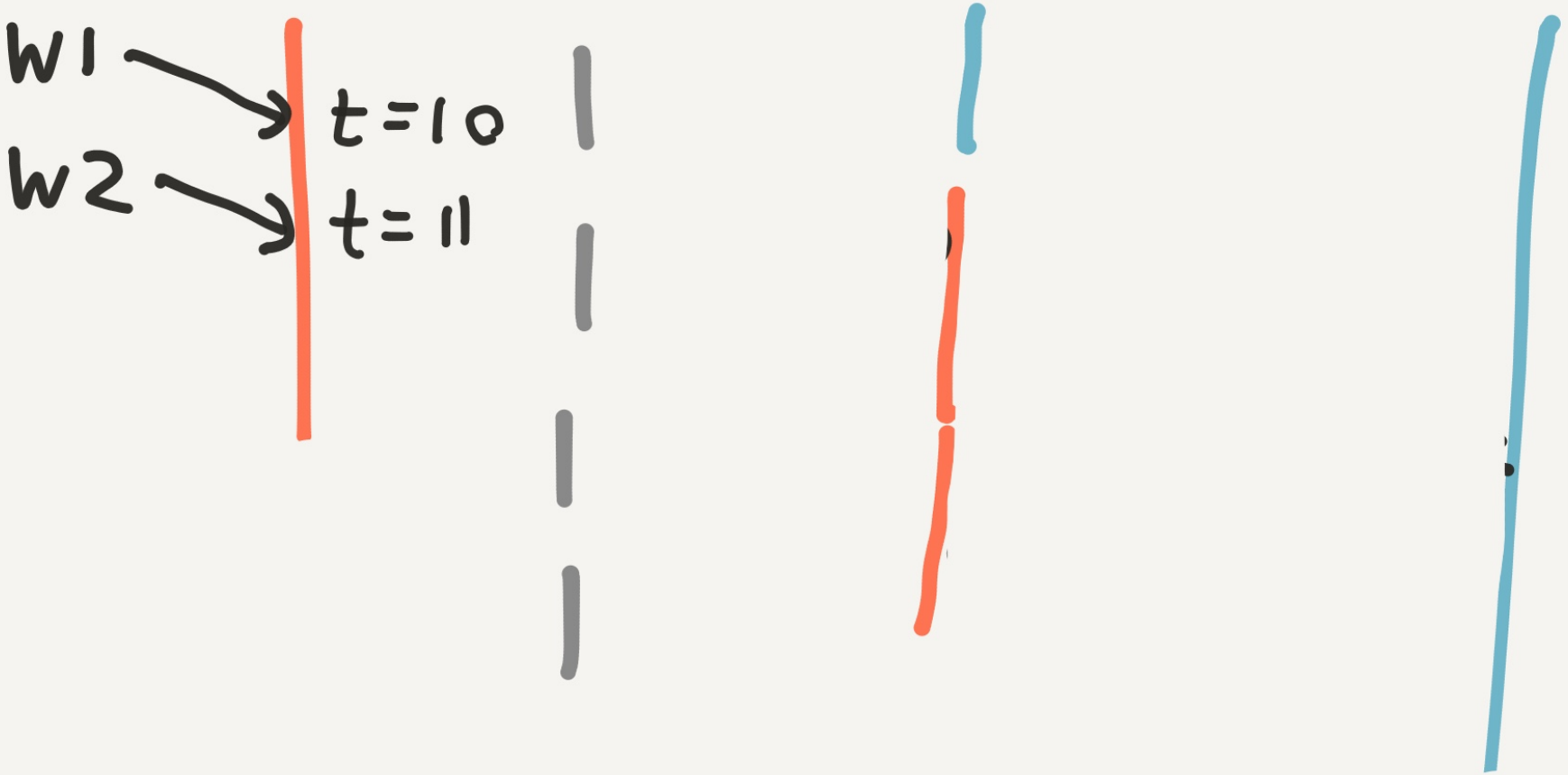
???

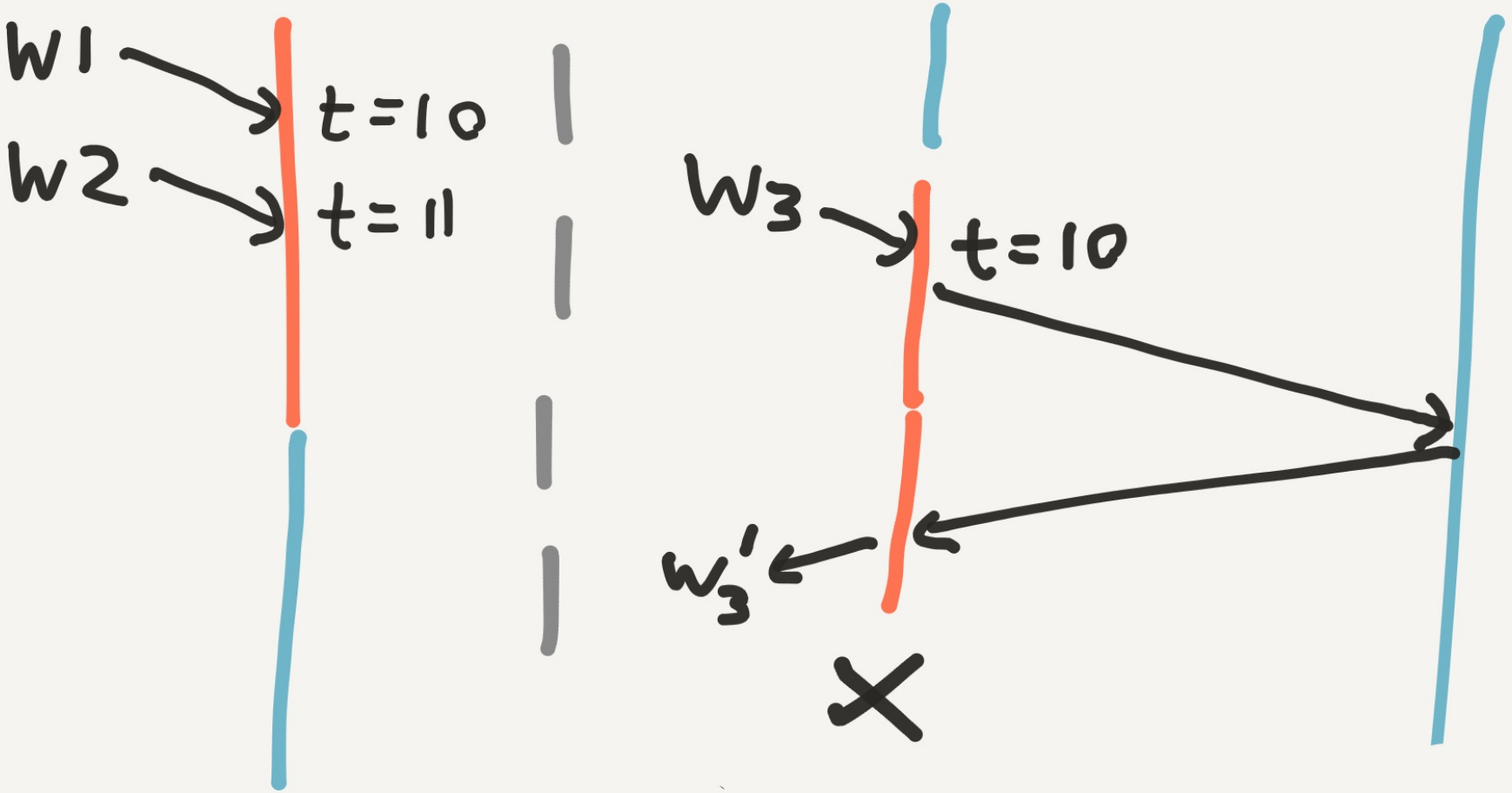
Optime:

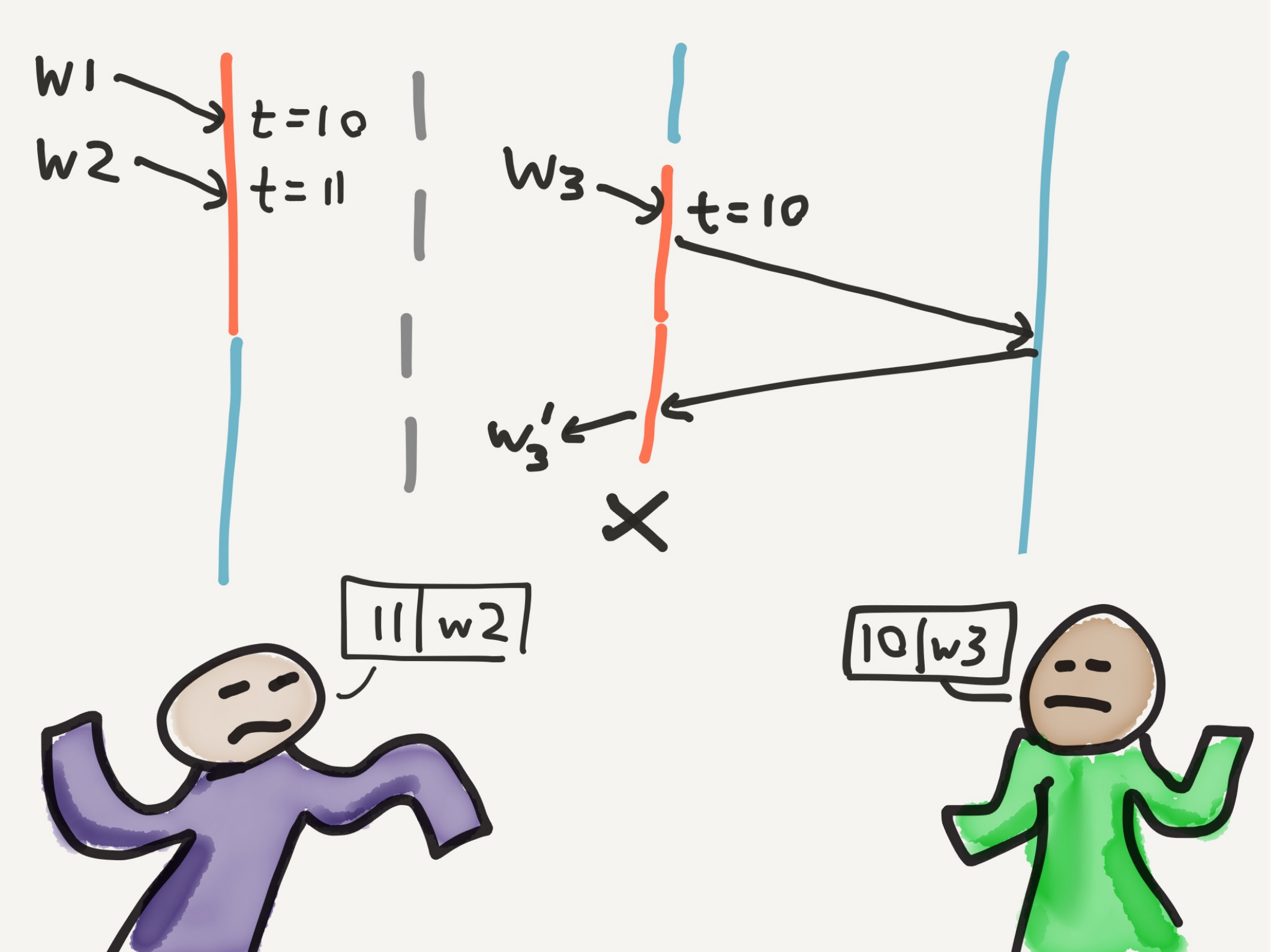
$$\text{Max}\begin{cases} \text{last optime} + 1 \\ \text{local wall clock} \end{cases}$$

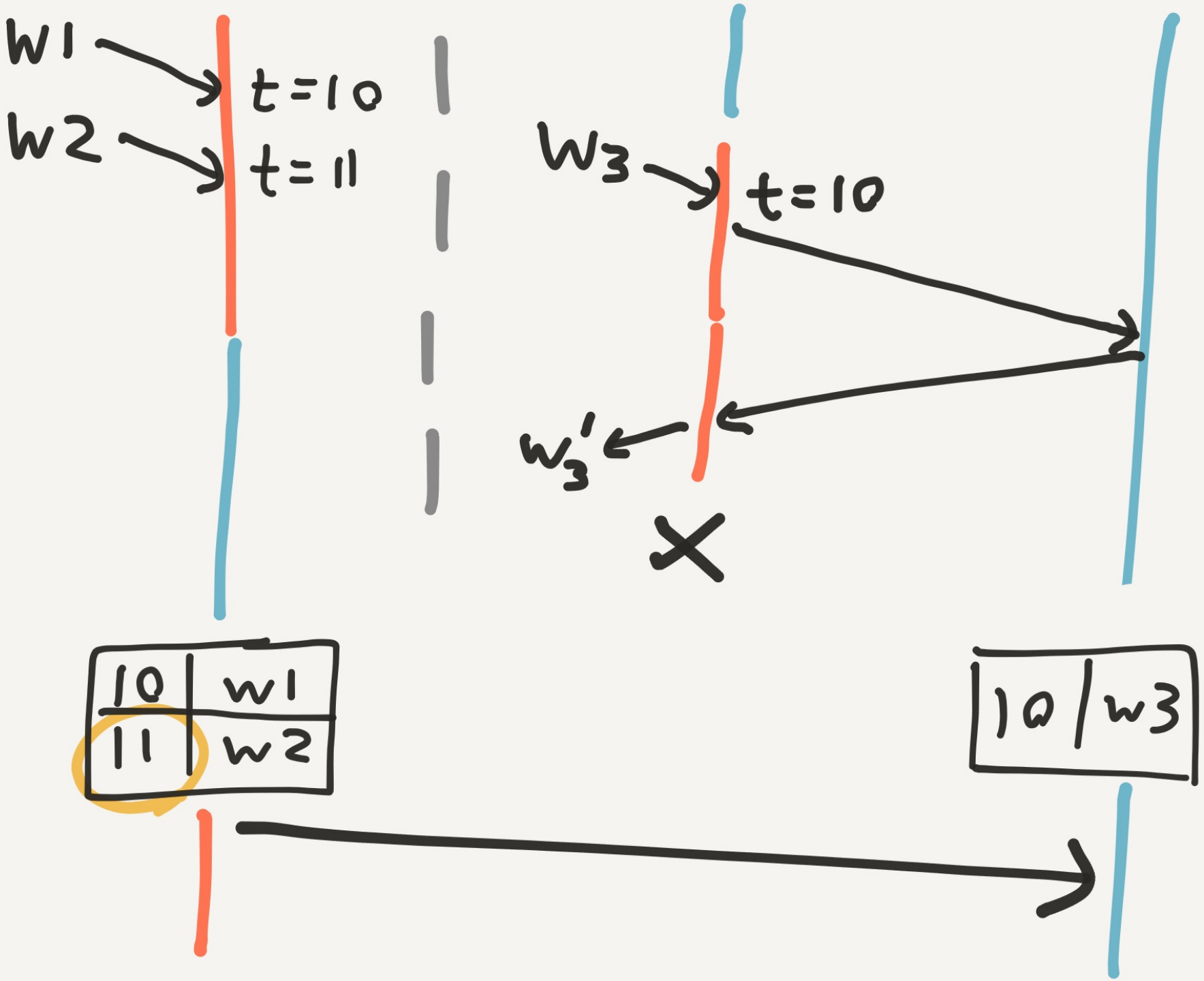


Node w/ highest optime wins


W1
W2
t=10
t=11

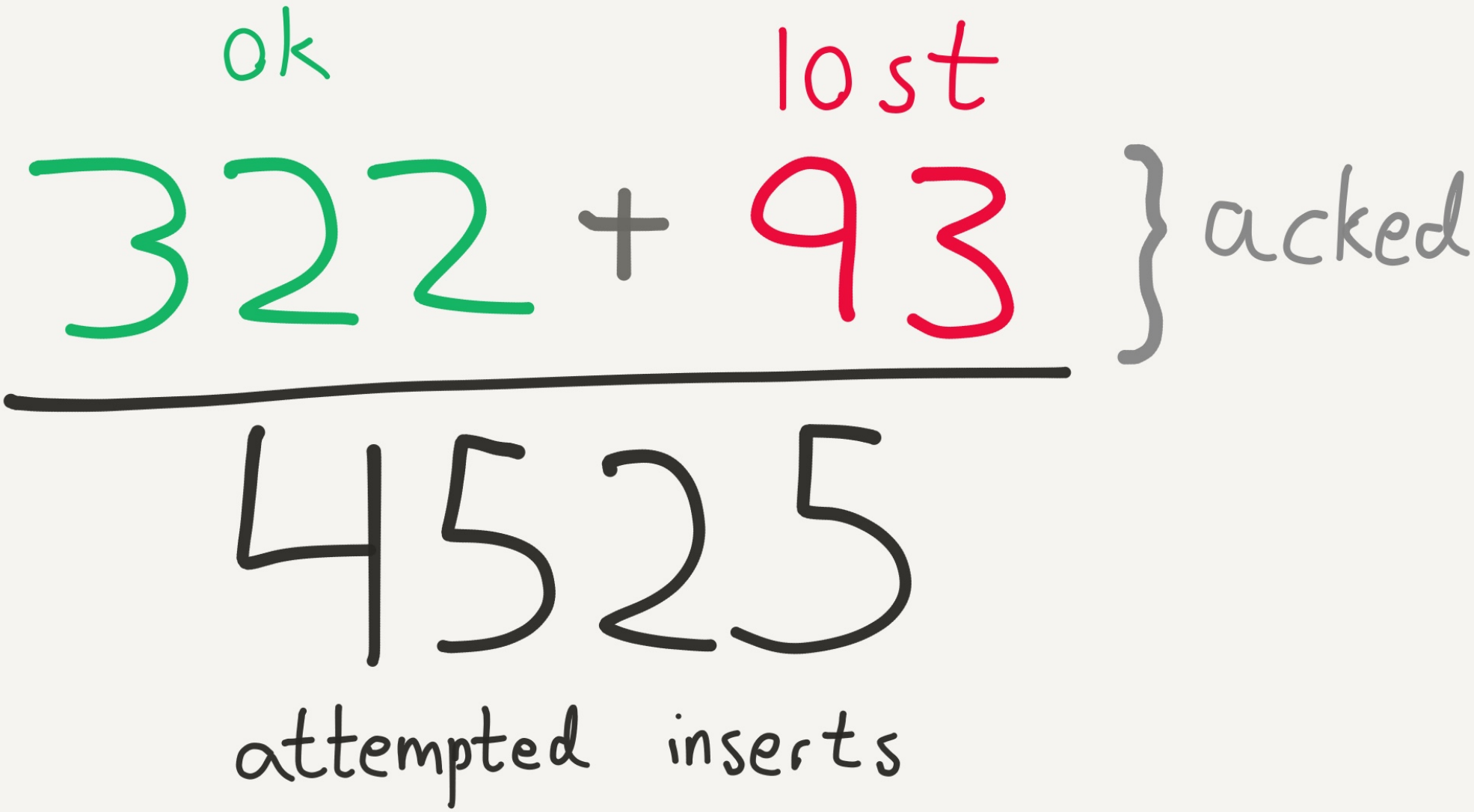

W1
t=10
W2
t=11
W3
t=10
W3'
X

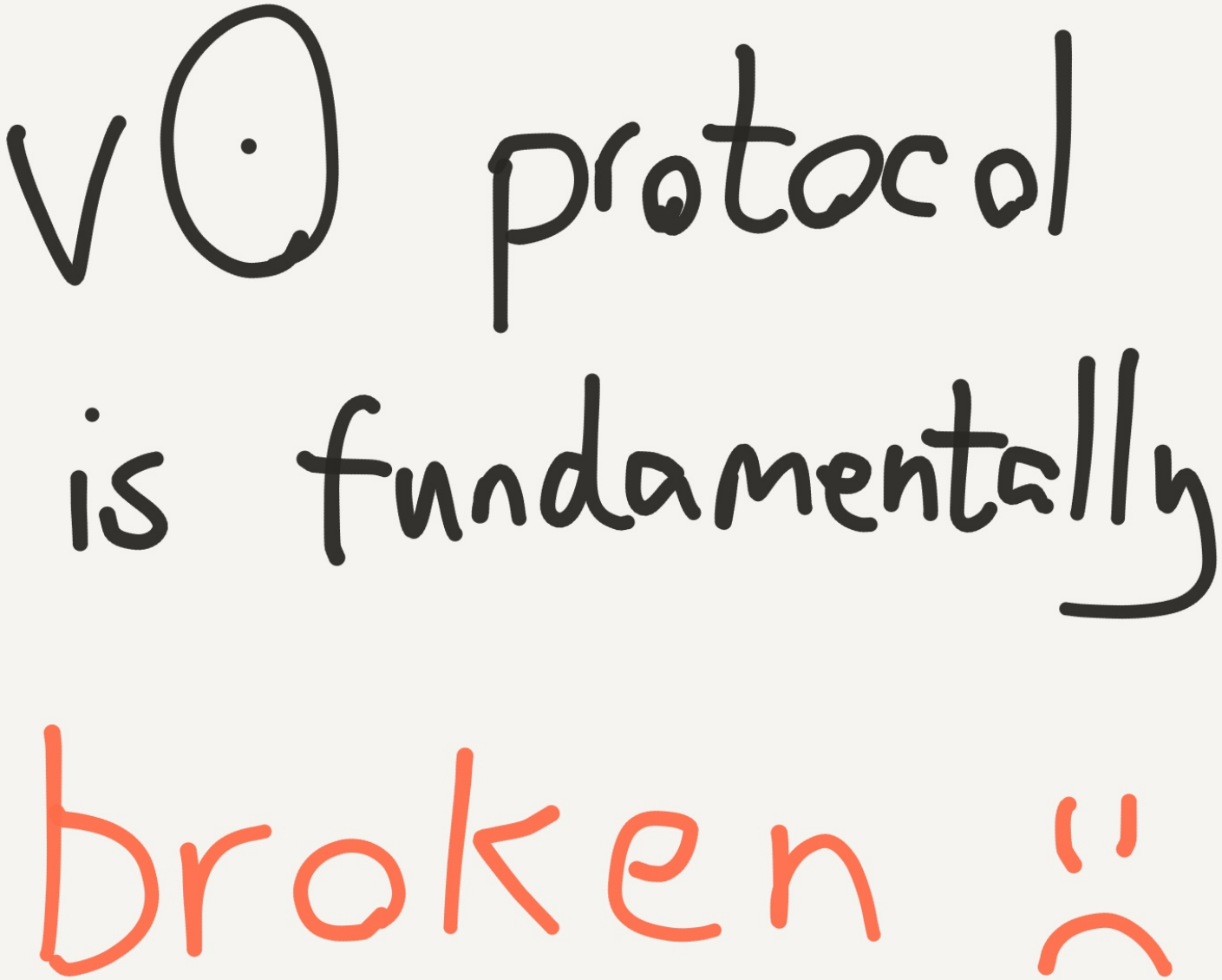

w1
t=10
w2
t=11
w3
t=10
w3'
X
11 | w2
10 | w3


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!


ok
lost
322 + 93
acked
4525
attempted inserts

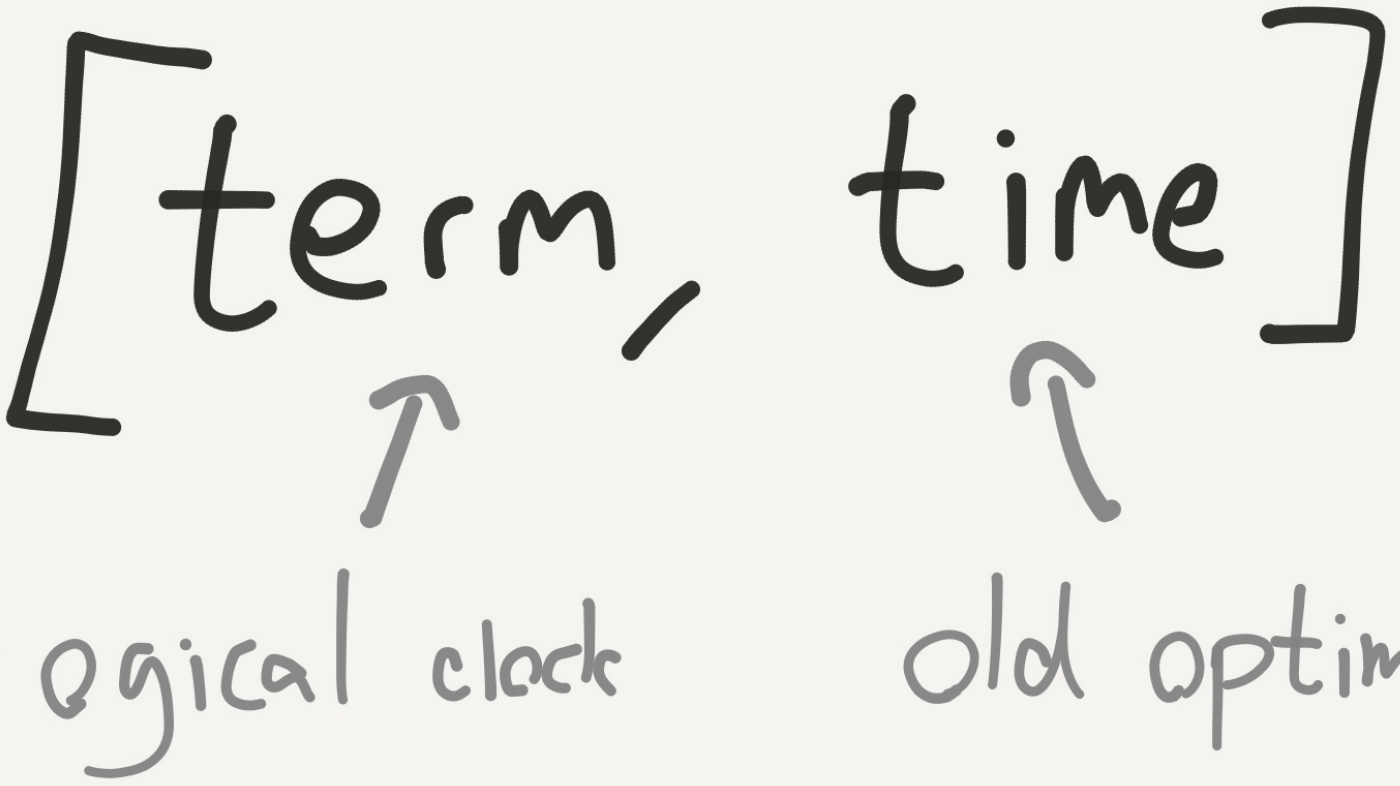

v0 protocol
is fundamentally
broken

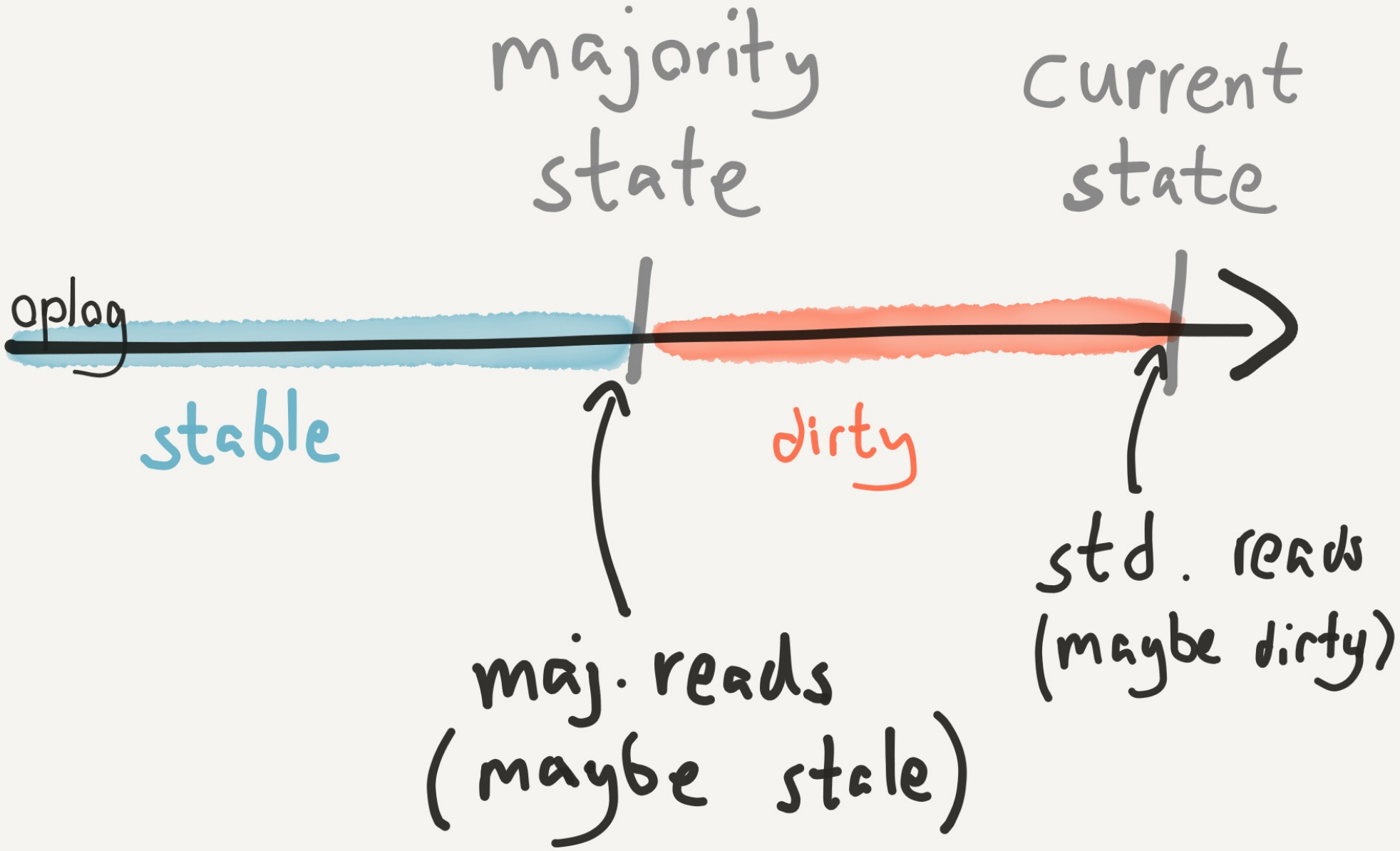
Protocol v1

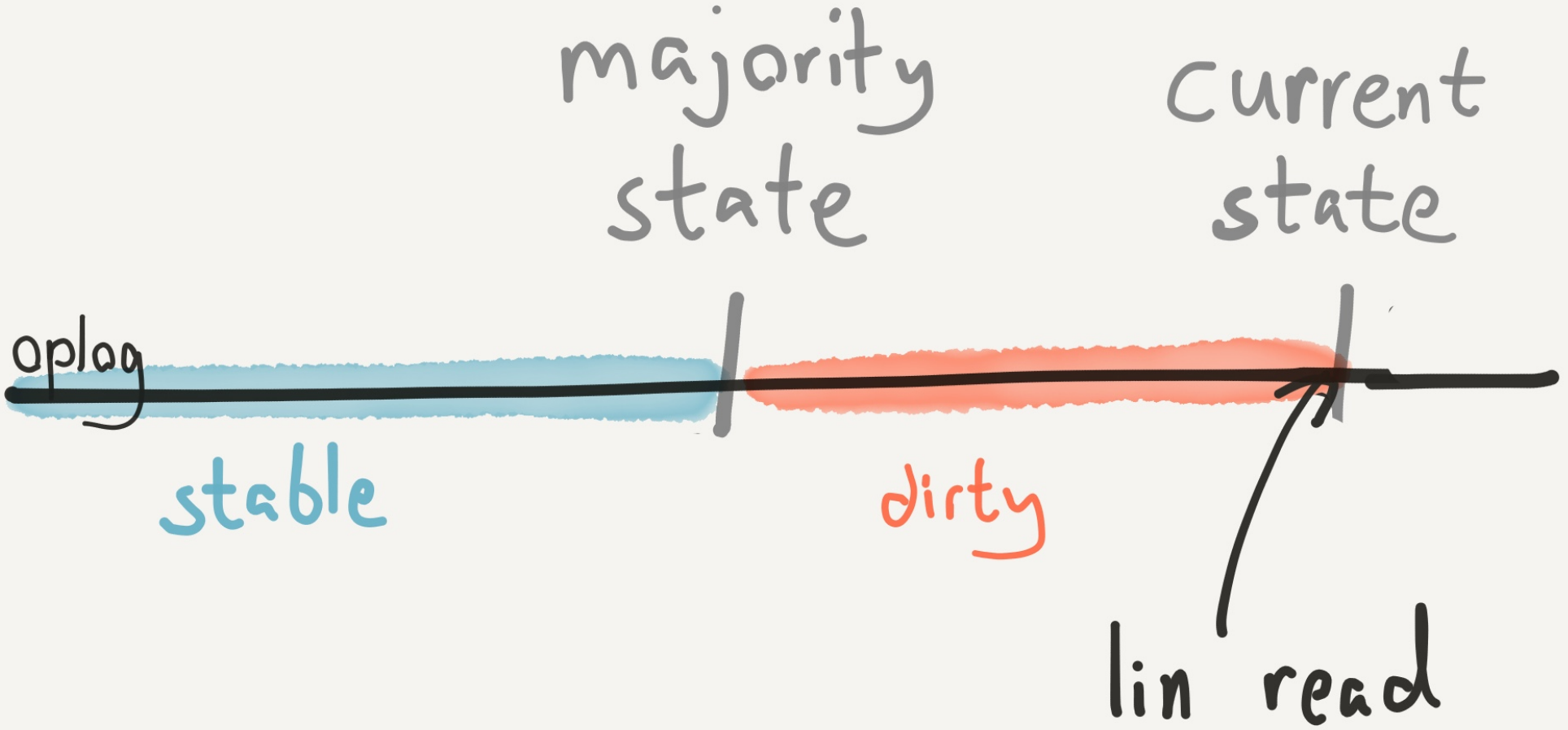
Default in 3.2

for new replsets

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

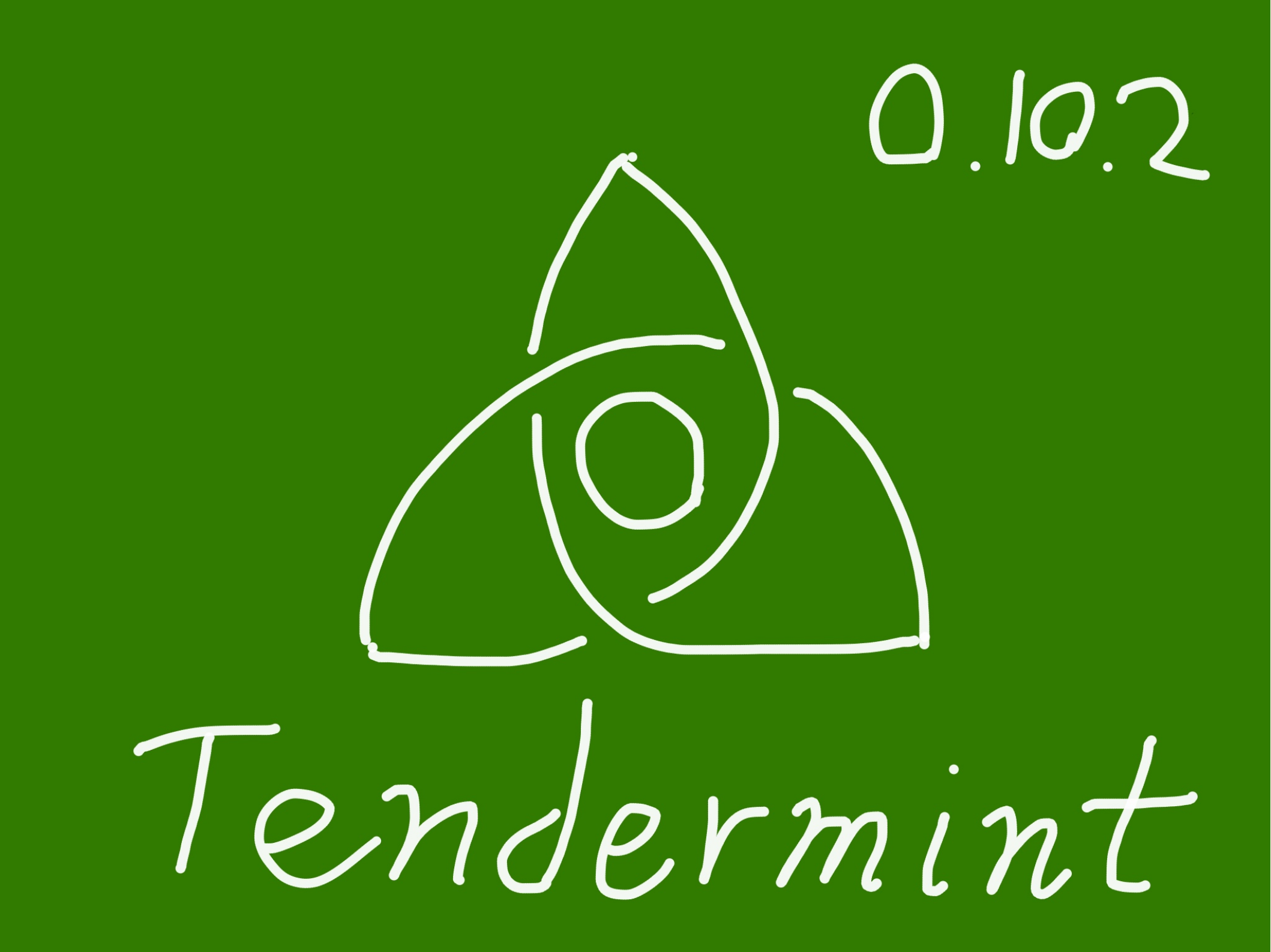
3.5.1

Recommendations

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

Use Write Concern MAJORITY

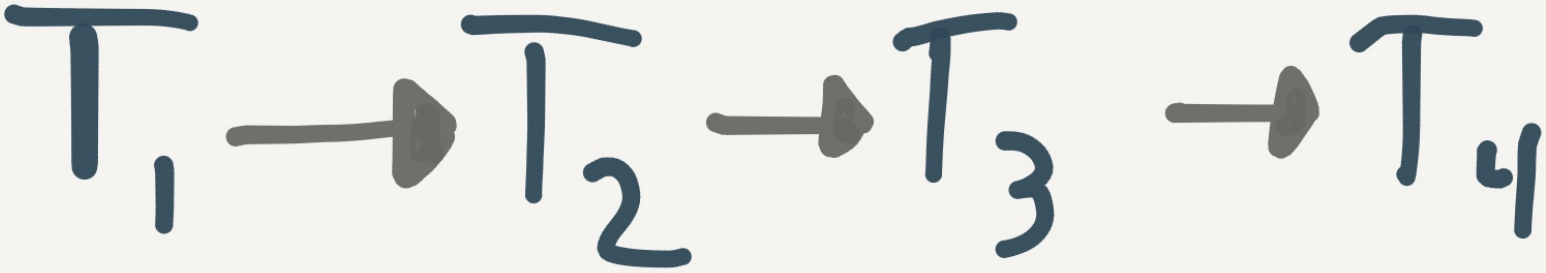
Read Concern MAJ/LIN


0.10.2
Tendermint

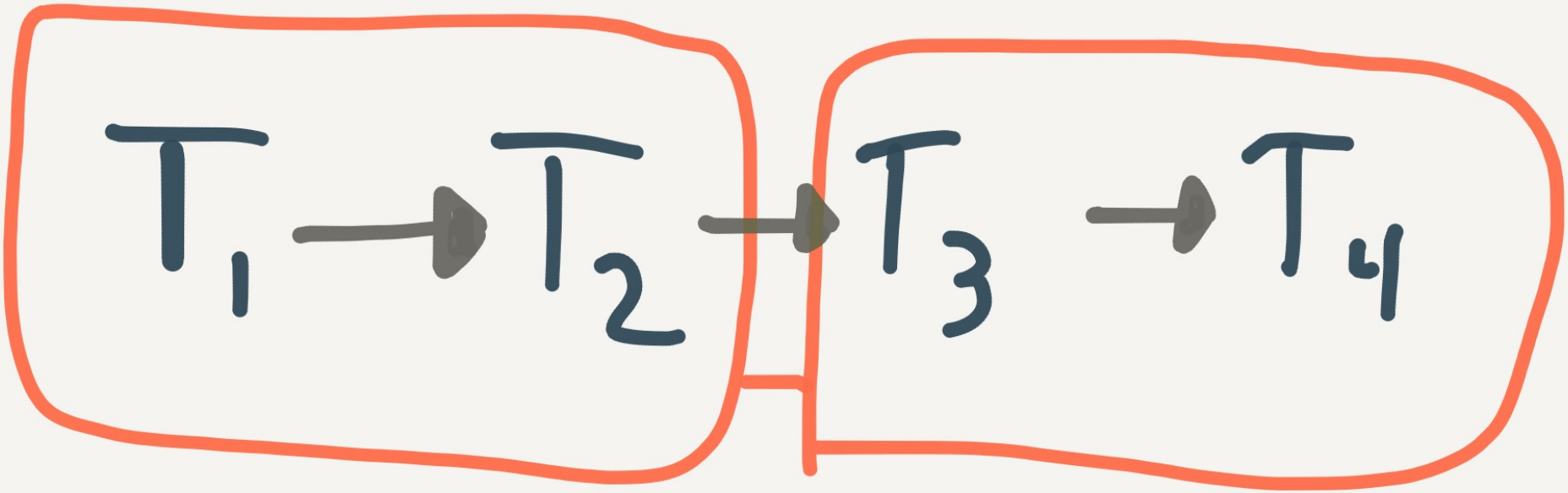
- Transaction replication
- Pluggable FSMs
- Byzantine fault tolerant

"Raft Meets Blockchain"

Like Raft, provides a linearizable order of txns

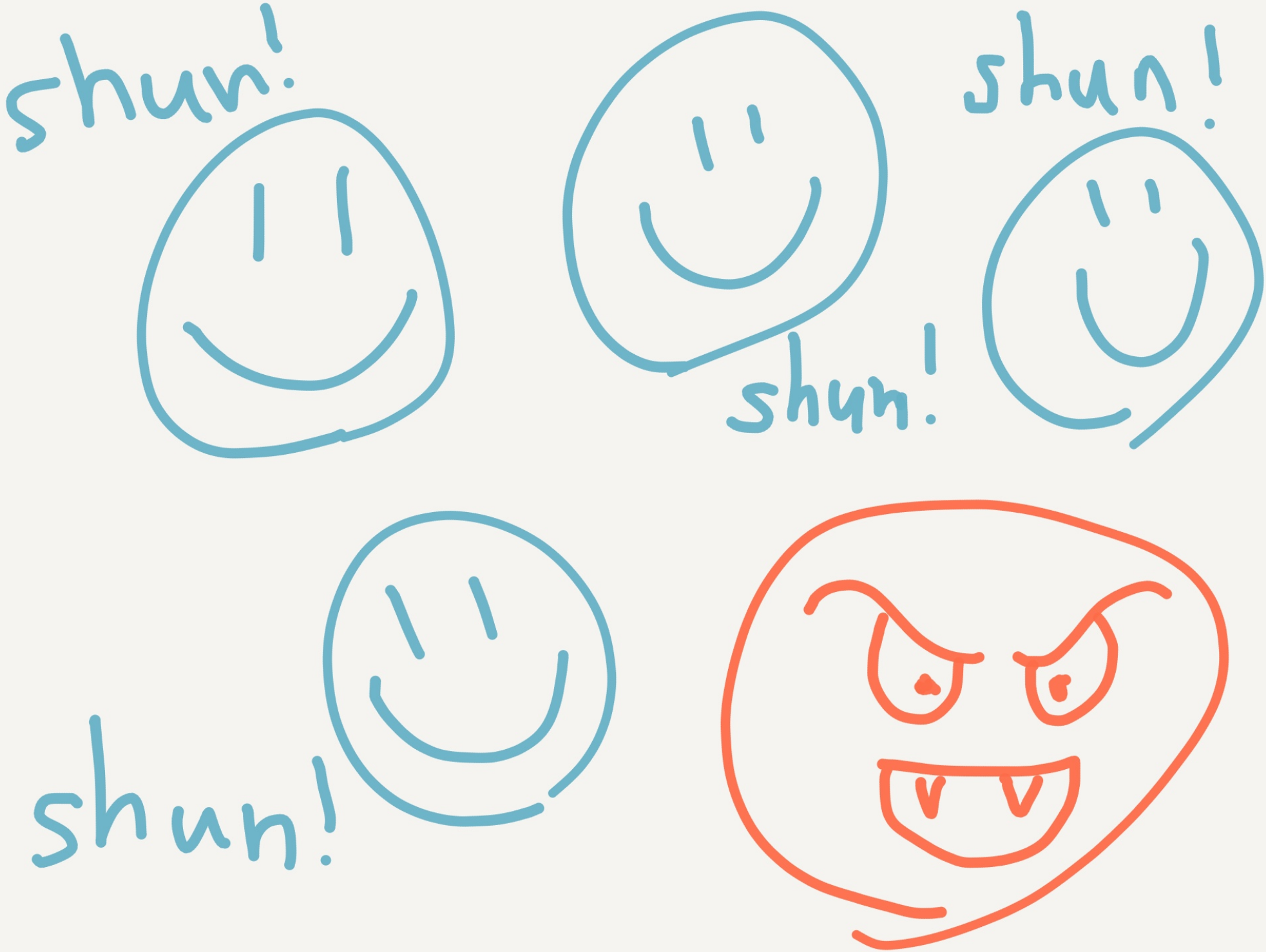


Like Raft, provides a linearizable order of txns

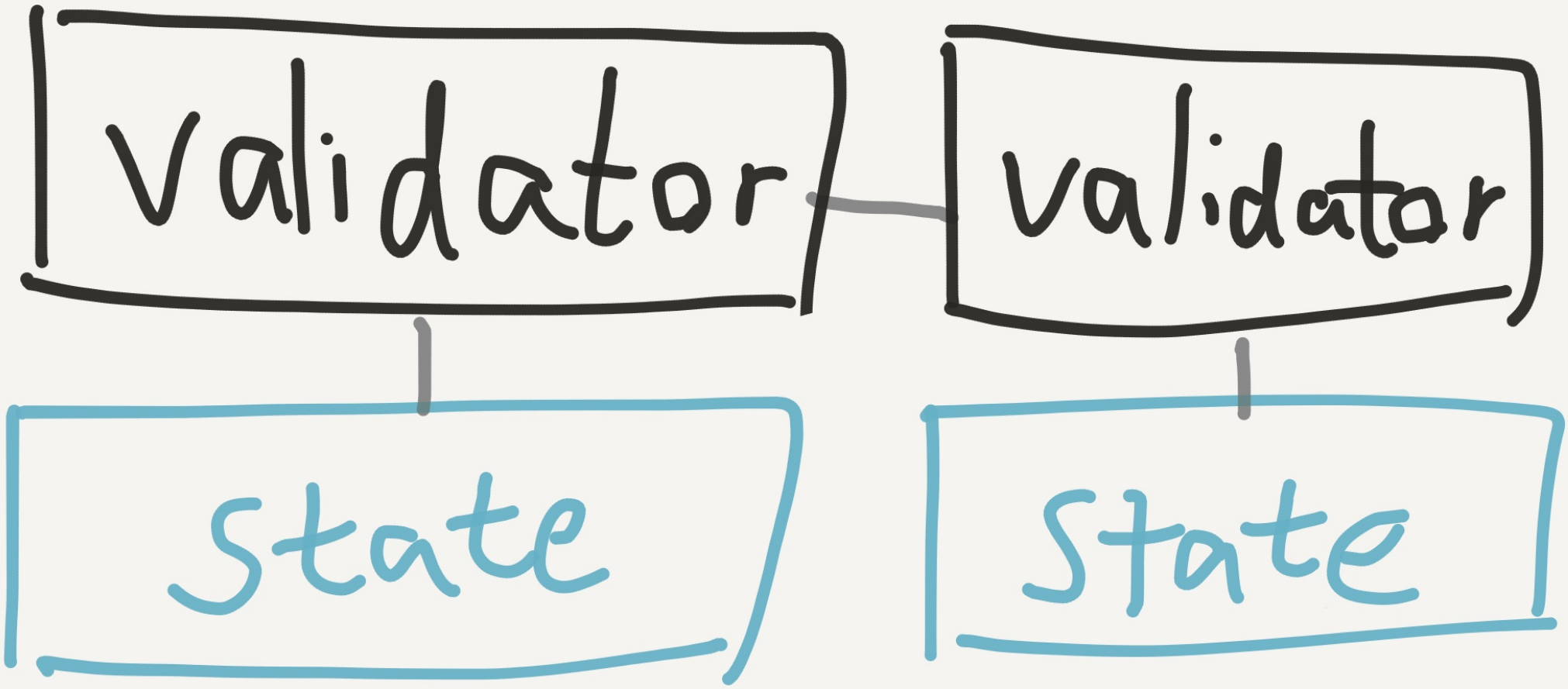


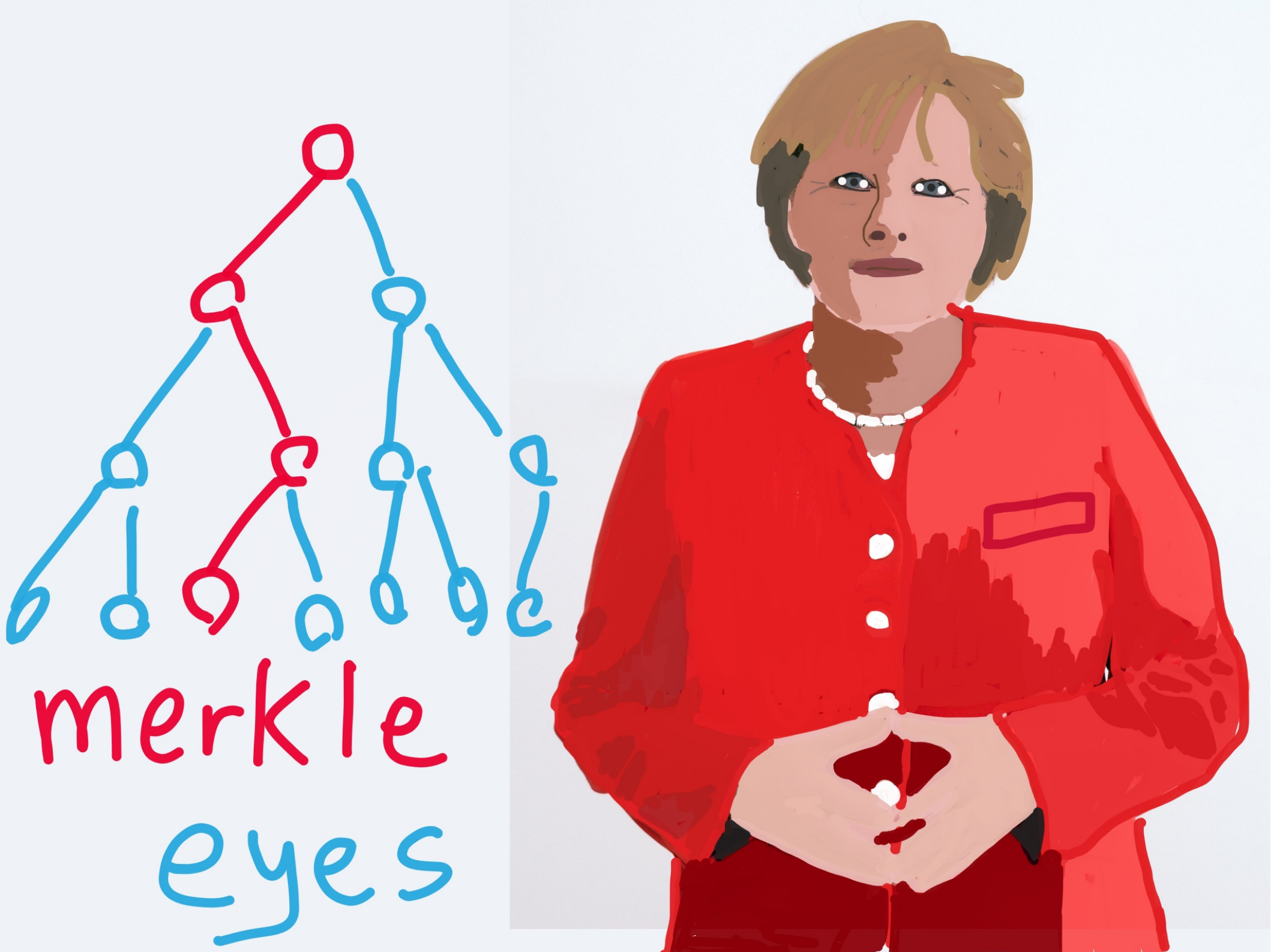
blockchain!

Byzantine Fault Tolerant!


shun!
shun!
shun!
shun!
shun!

Pluggable State Machines


validator
validator
state
state


merkle
eyes

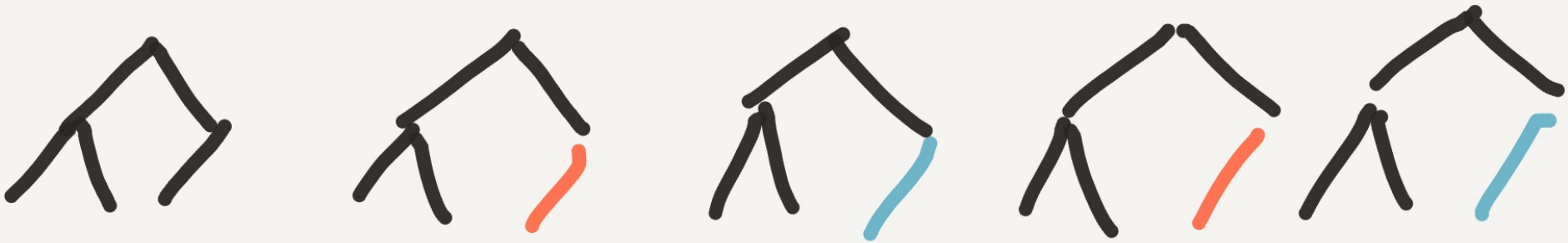
cas-register

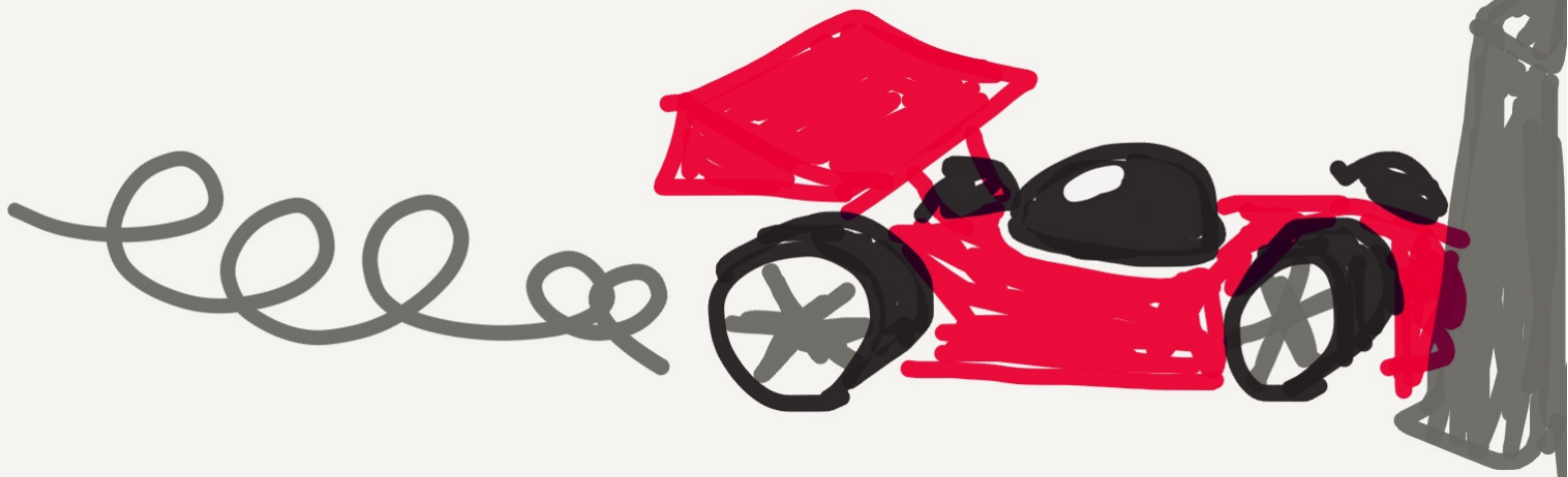
- read(2)
- write(5)
- cas(1,4)

set

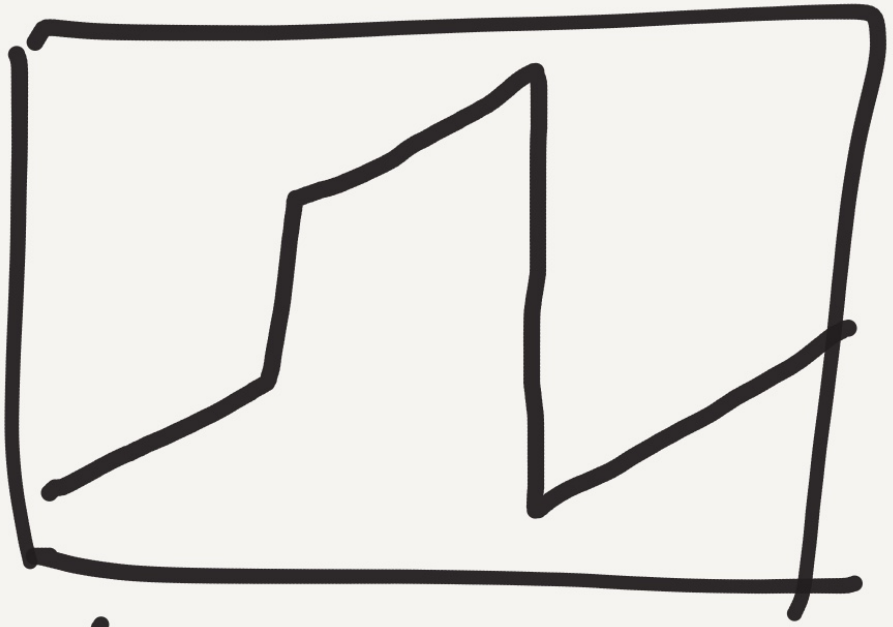
- add(6)
  - [read({1,2,3})
  - cas({1,2,3}, {1,2,3,6})]
- read({1,2,3,...})

Merkleeyes Race

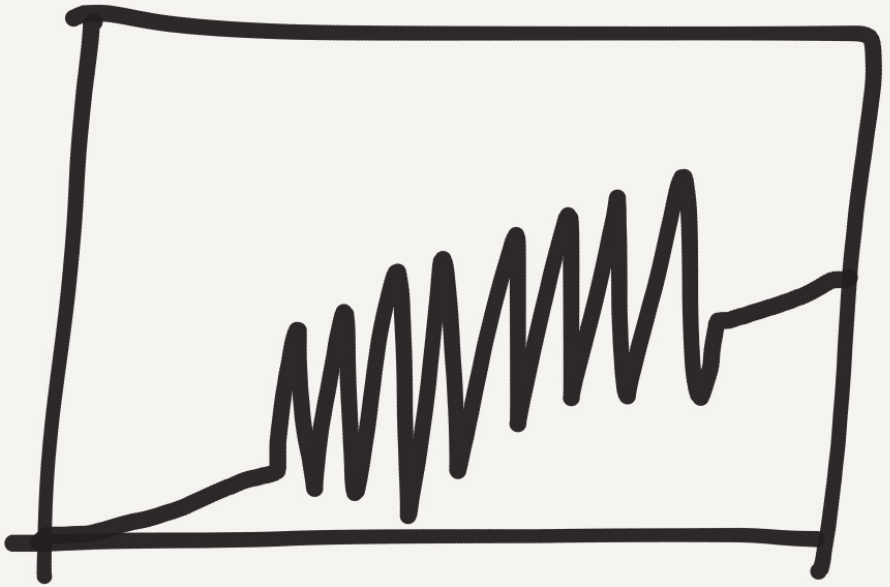




clock skew

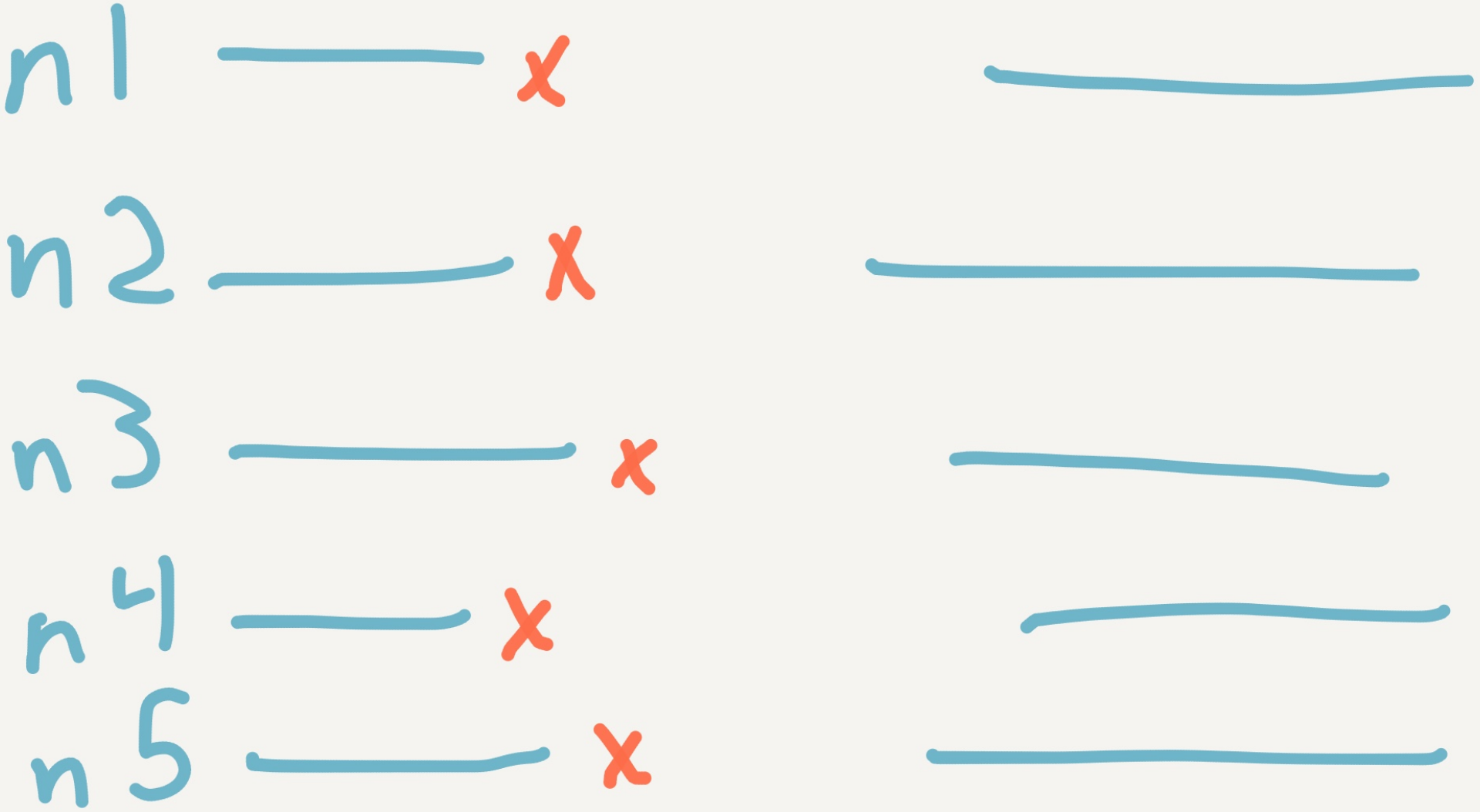


bump

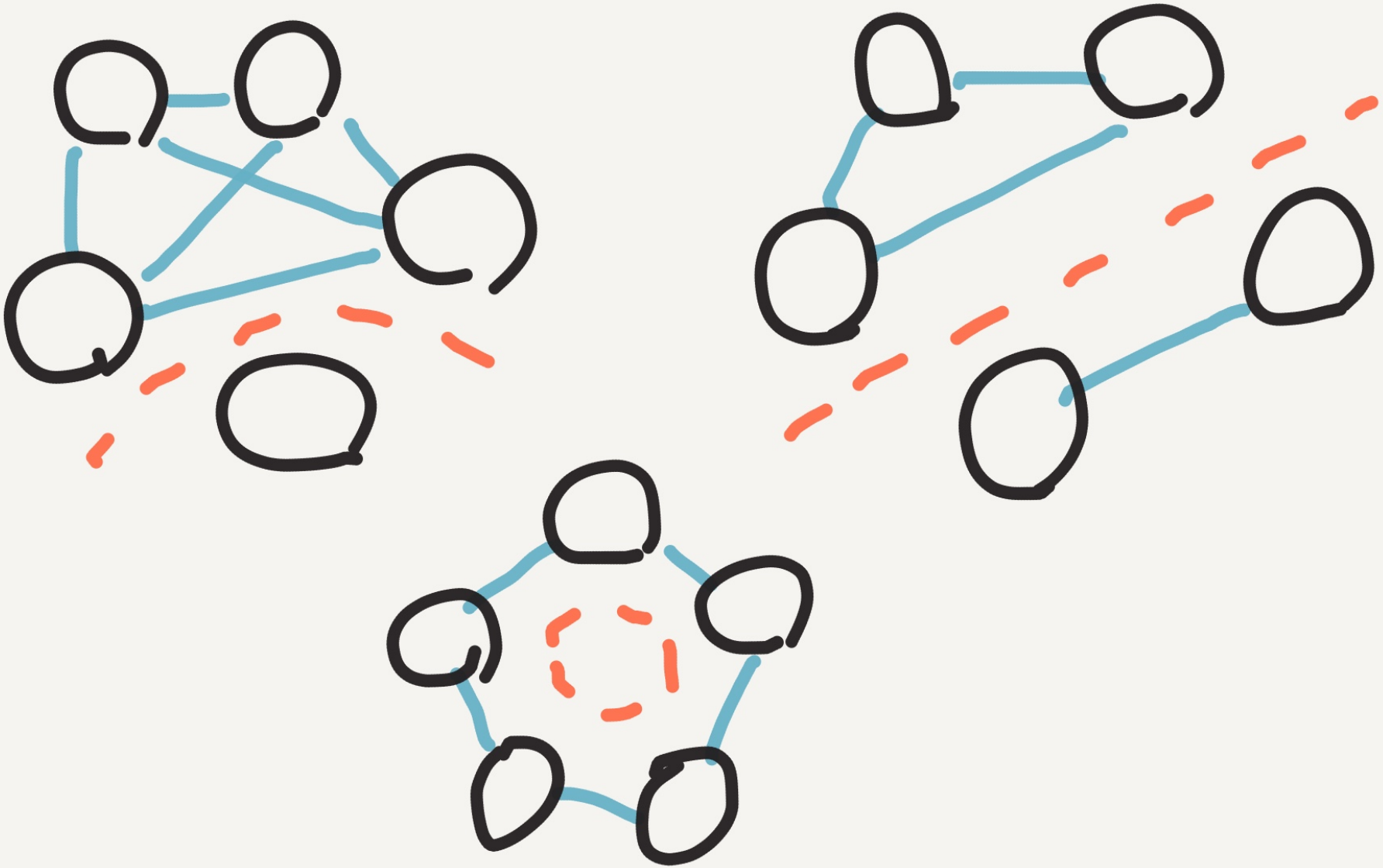


strobe

crashes



partitions



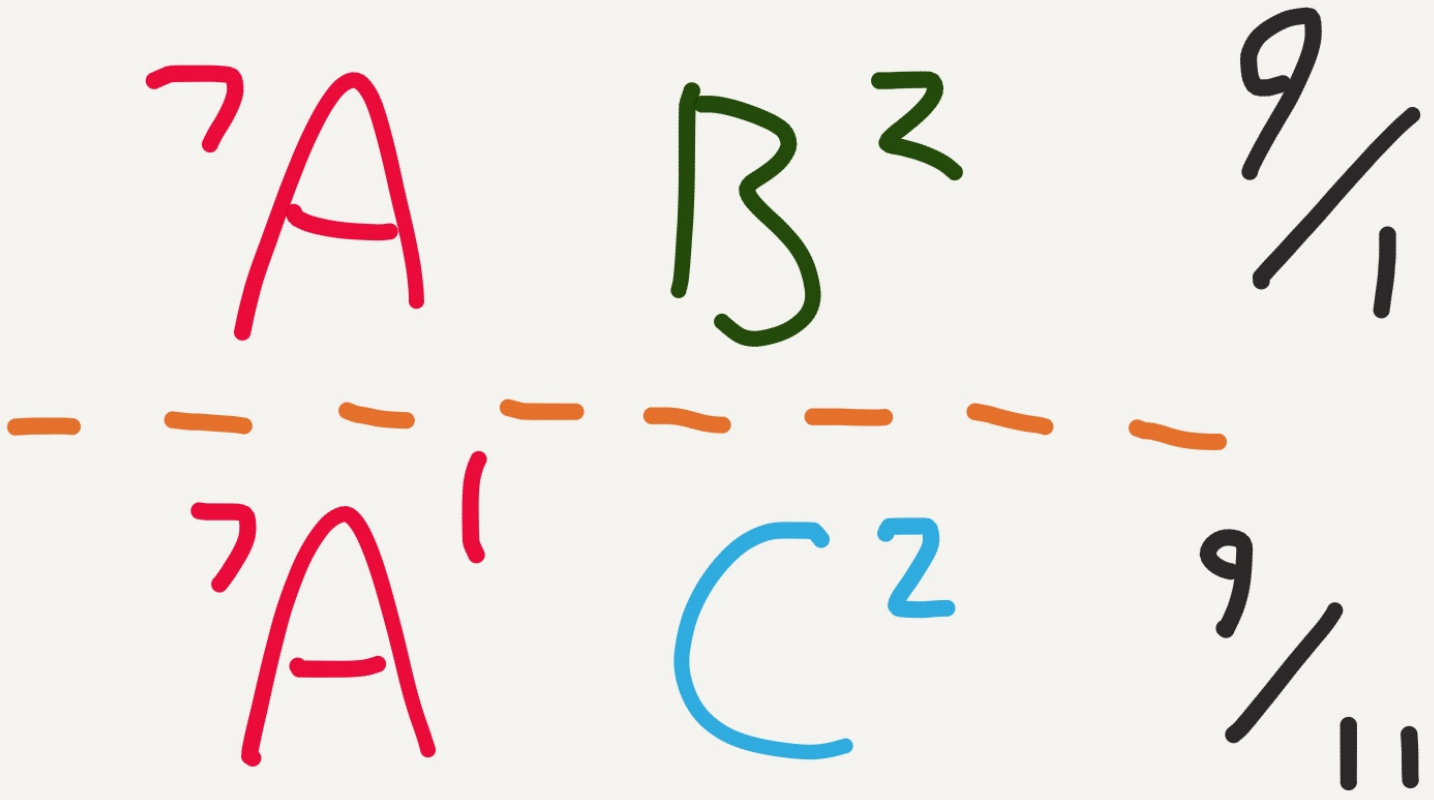
Duplicate Validators

$7A^2$ B^2

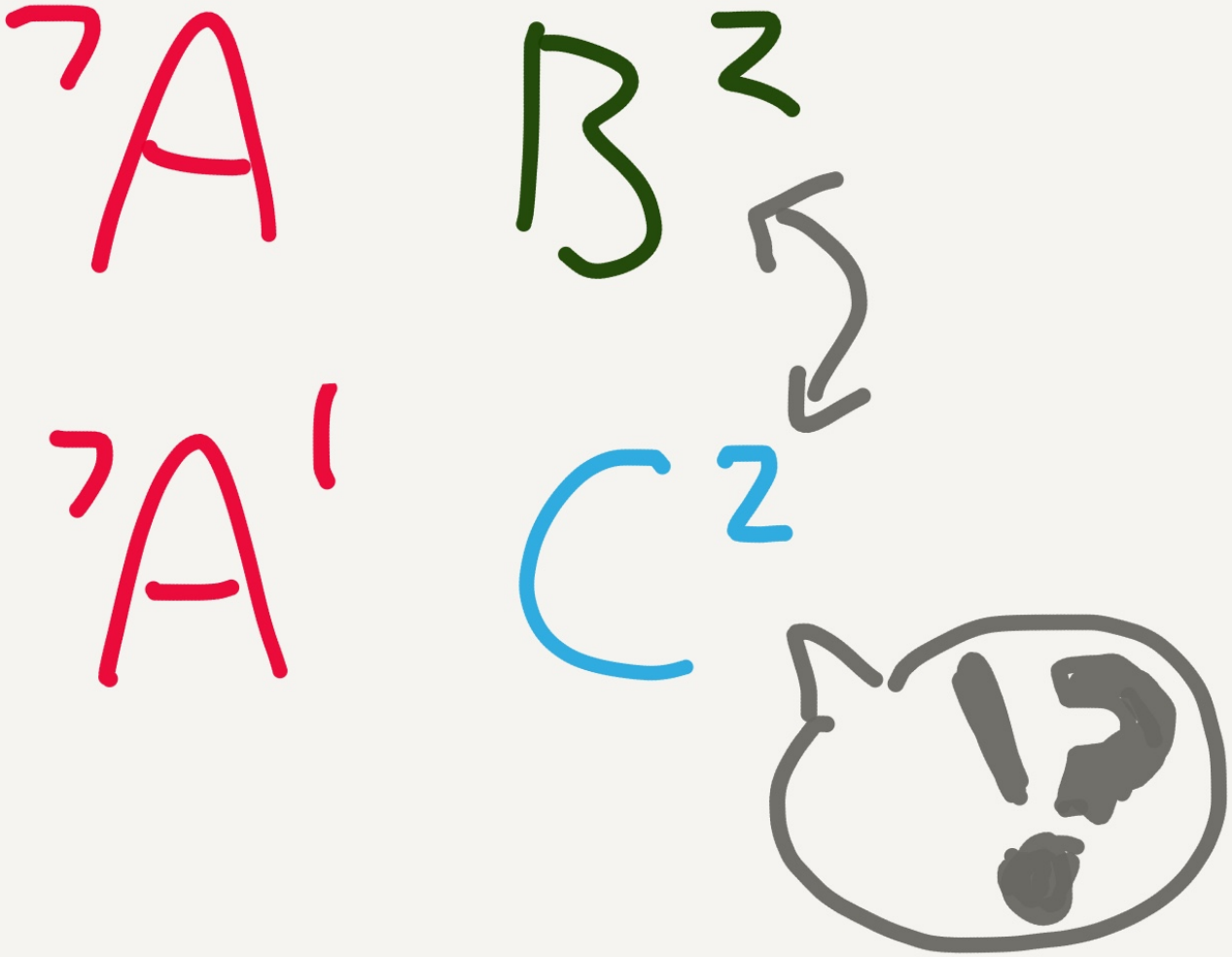
$7A'^1$ C^2

$7 + 2 + 2 = 11$

Duplicate Validators



Duplicate Validators



Lost Documents

iff

$\geq 1/3$

of votes

file truncation

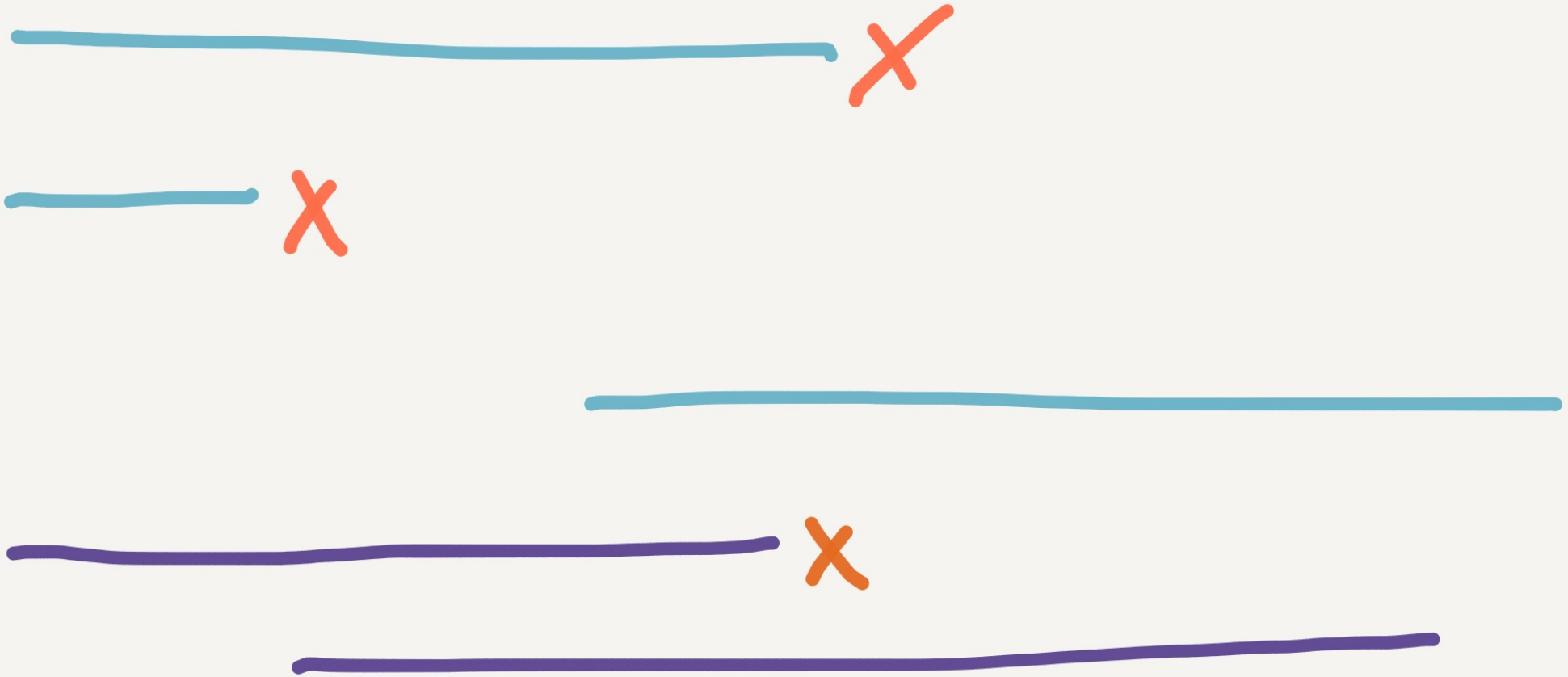
Merkleeyes crash!

(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

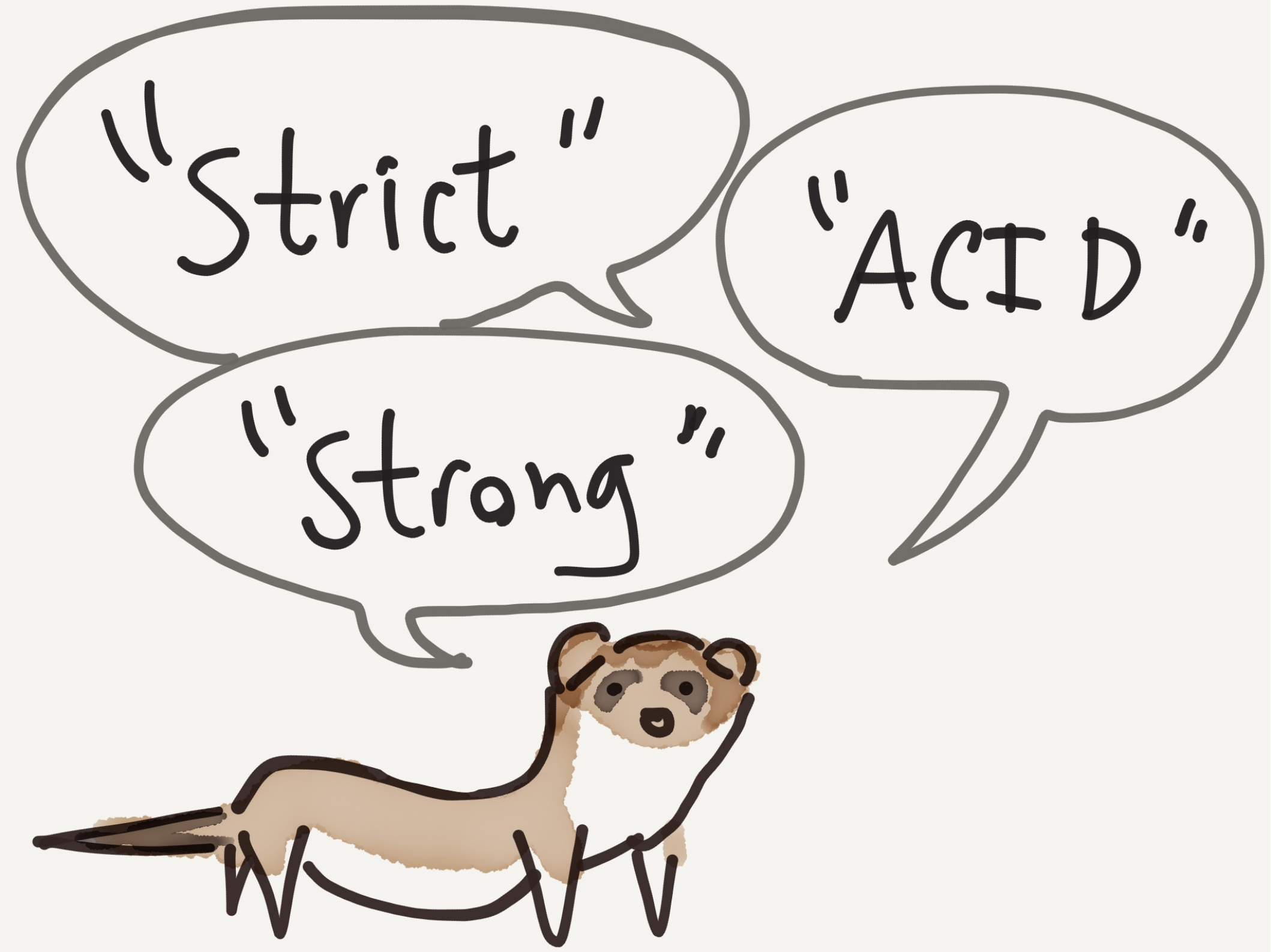
Recap


Read the docs!
CAREFULLY

Then test it

— for —

YOURSELF


"Strict"
"ACID"
"Strong"

Be Formal

&

Be Specific!

Figure out the

Invariants

your system needs

Consider your failure modes

Consider your failure modes

Process Crash

```
#kill -9 1234
```

Node failure

- AWS terminate
- Physical power switch

Clock Skew

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

GC/IO Pause

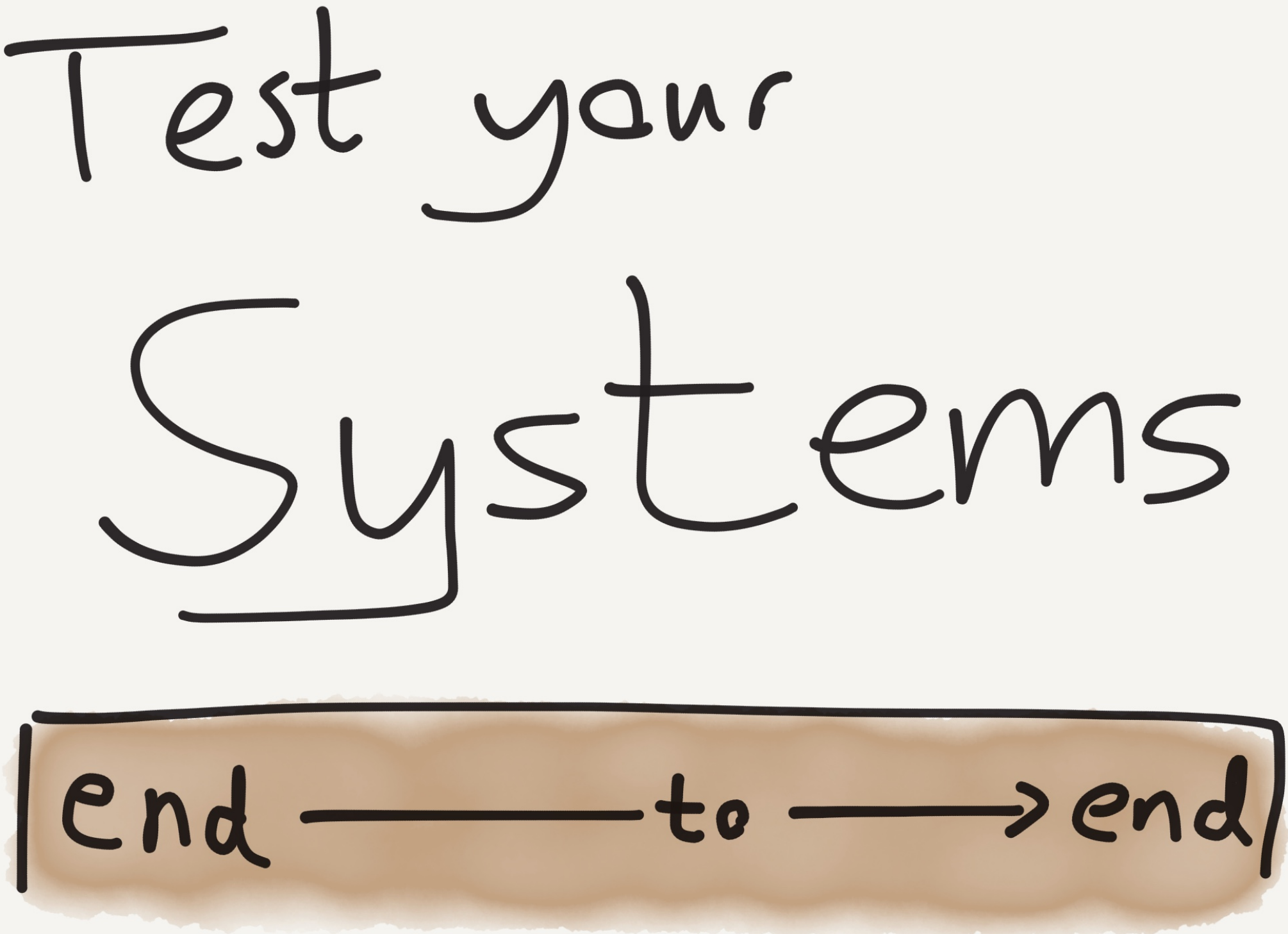
```
# killall -s STOP foo
# killall -s CONT foo
```

Network Partition

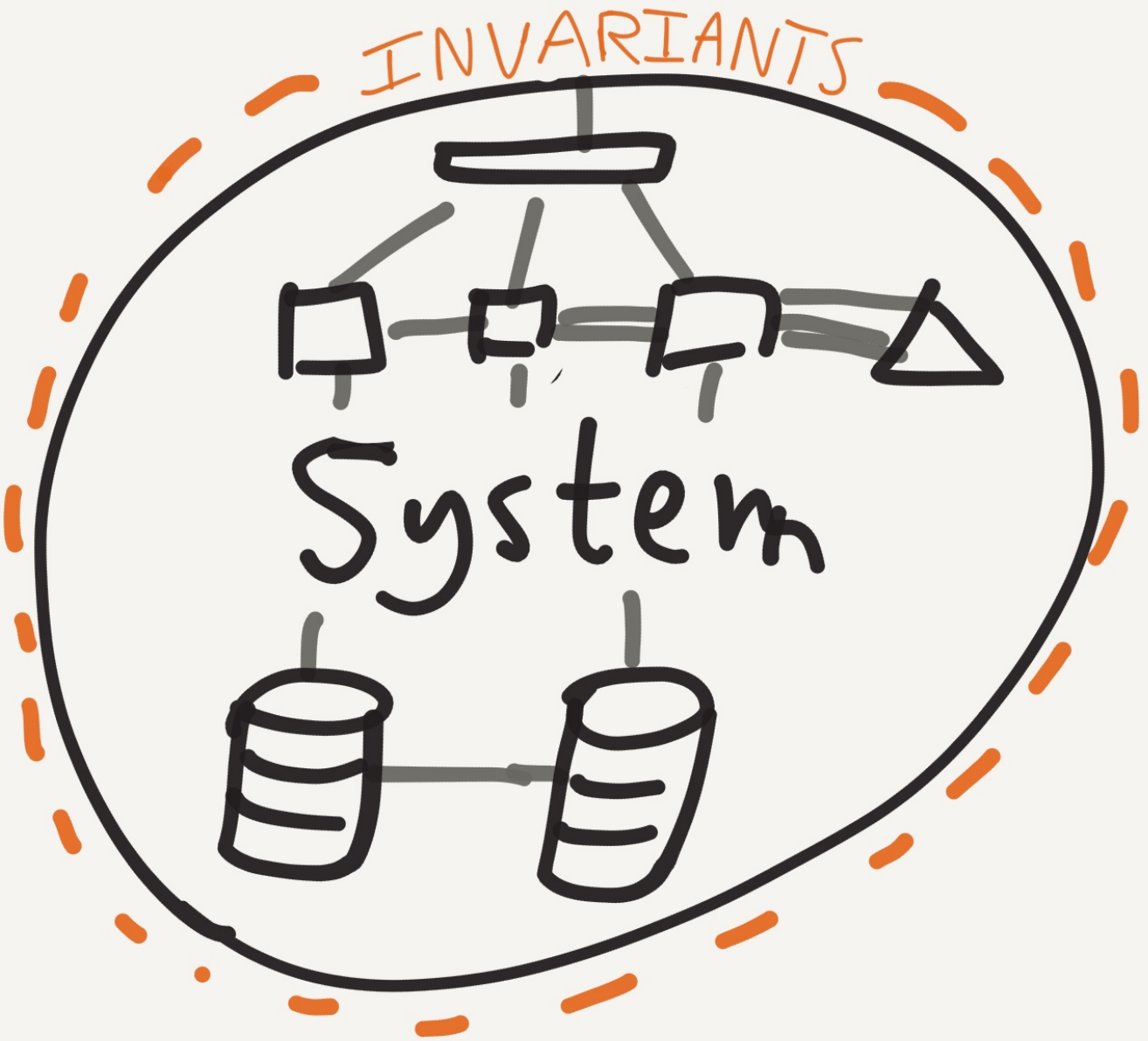
iptables -j DROP

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

- VoltDB
- MongoDB
- Tendermint
- Peter Alvaro

http://jepsen.io