Jepsen 6

a year to Forget

Kyle Kingsbury

@aphyr

I break databases

Public API {

} API code

Ruby {

DBs

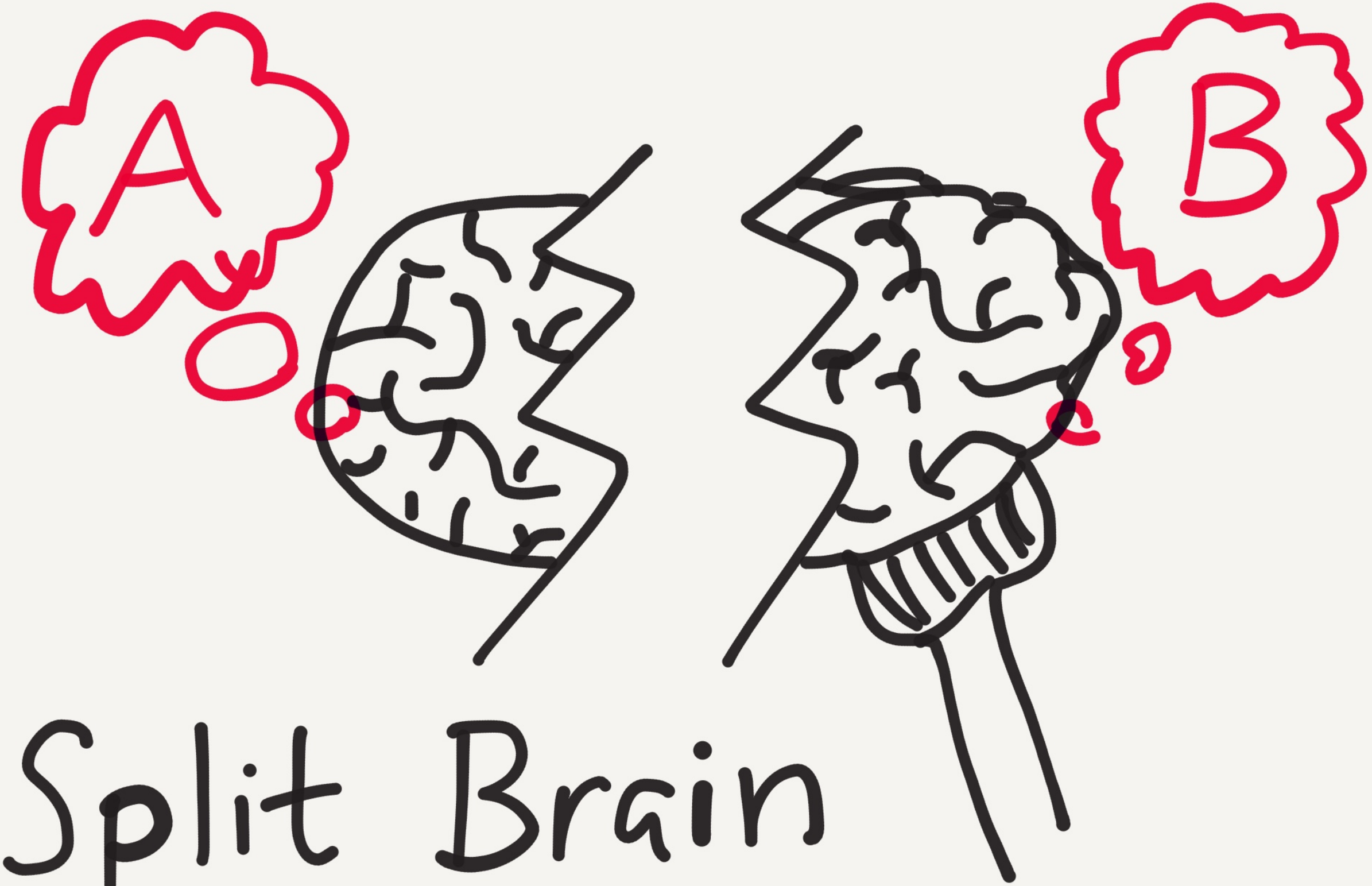
Every thing is fine!

Databases!

Queues!

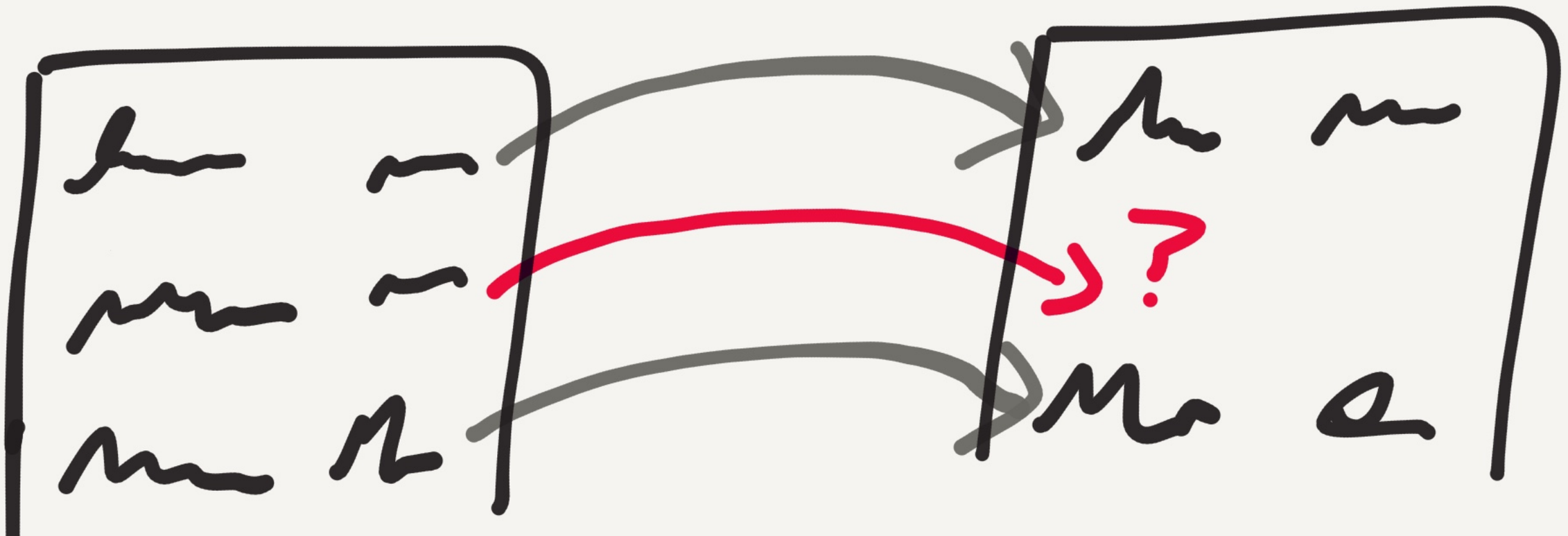
Discovery!

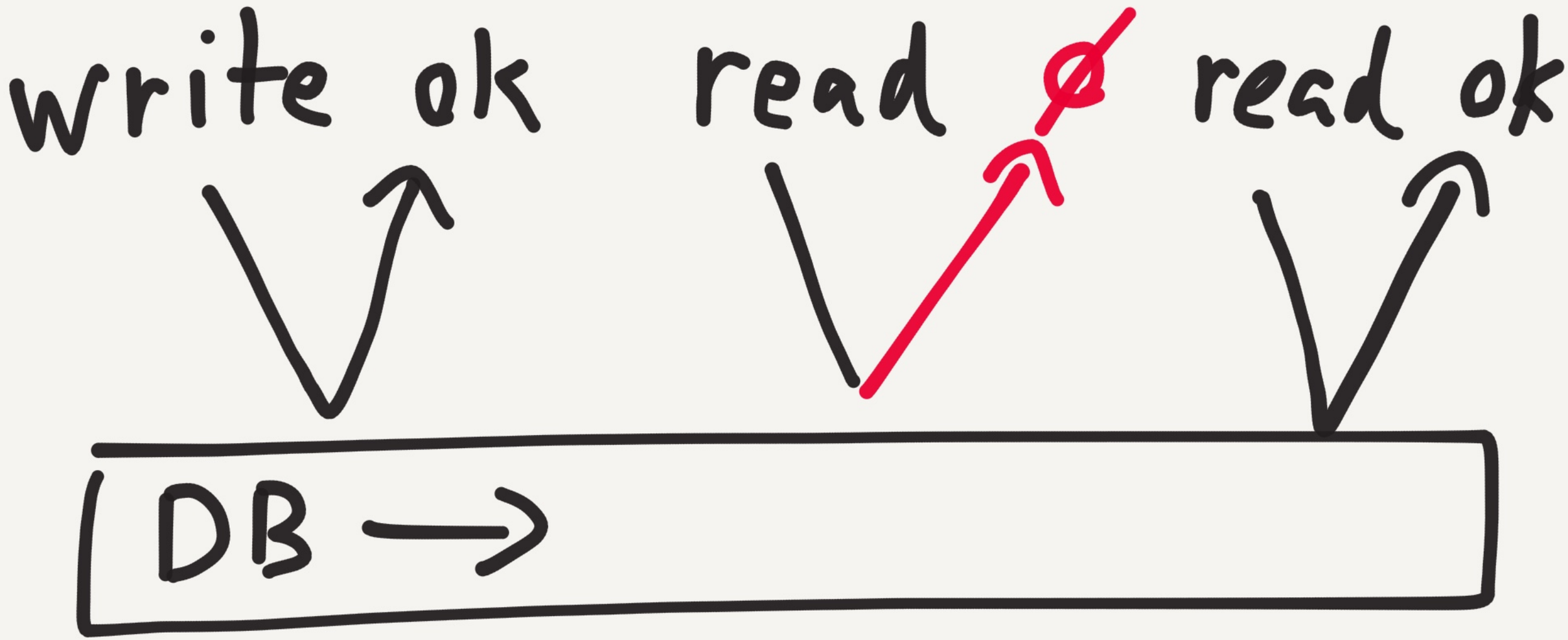
THE HORROR



Split Brain

Broken Foreign Keys

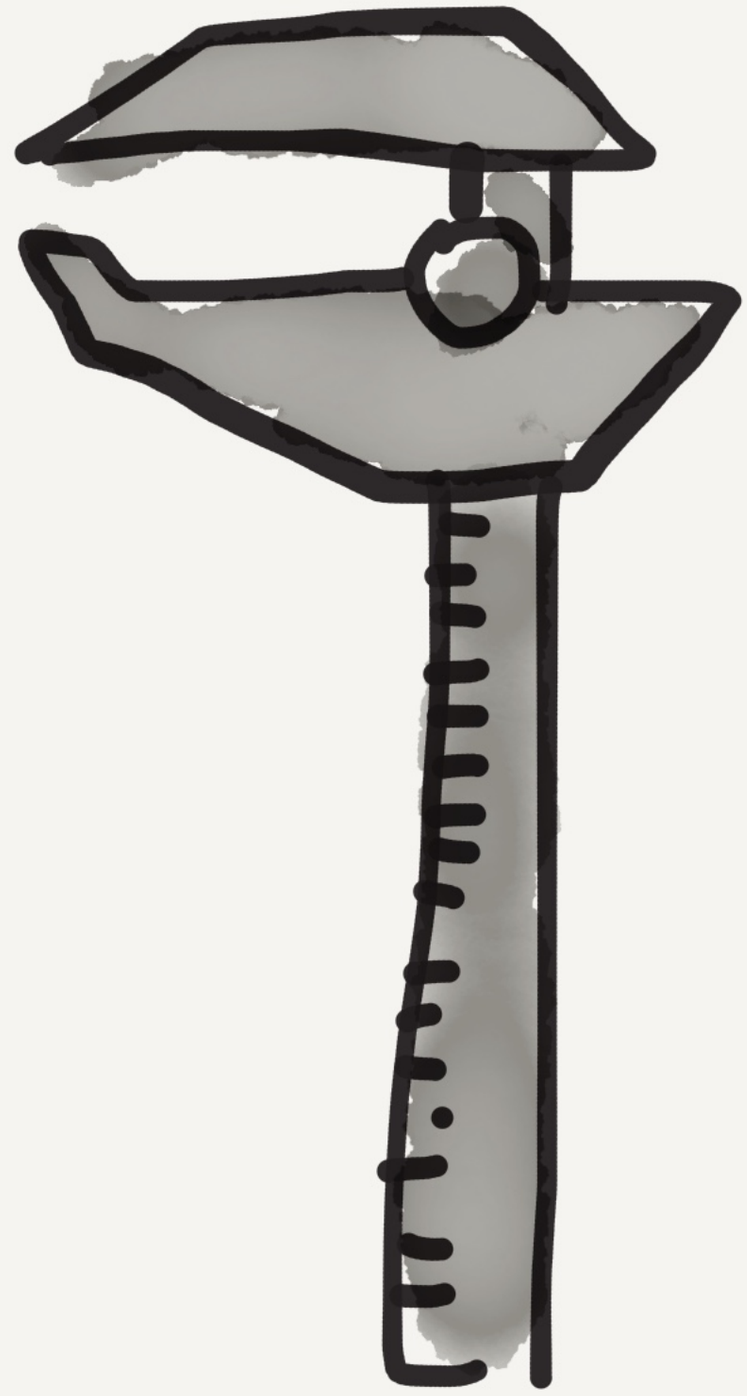




Anomalies

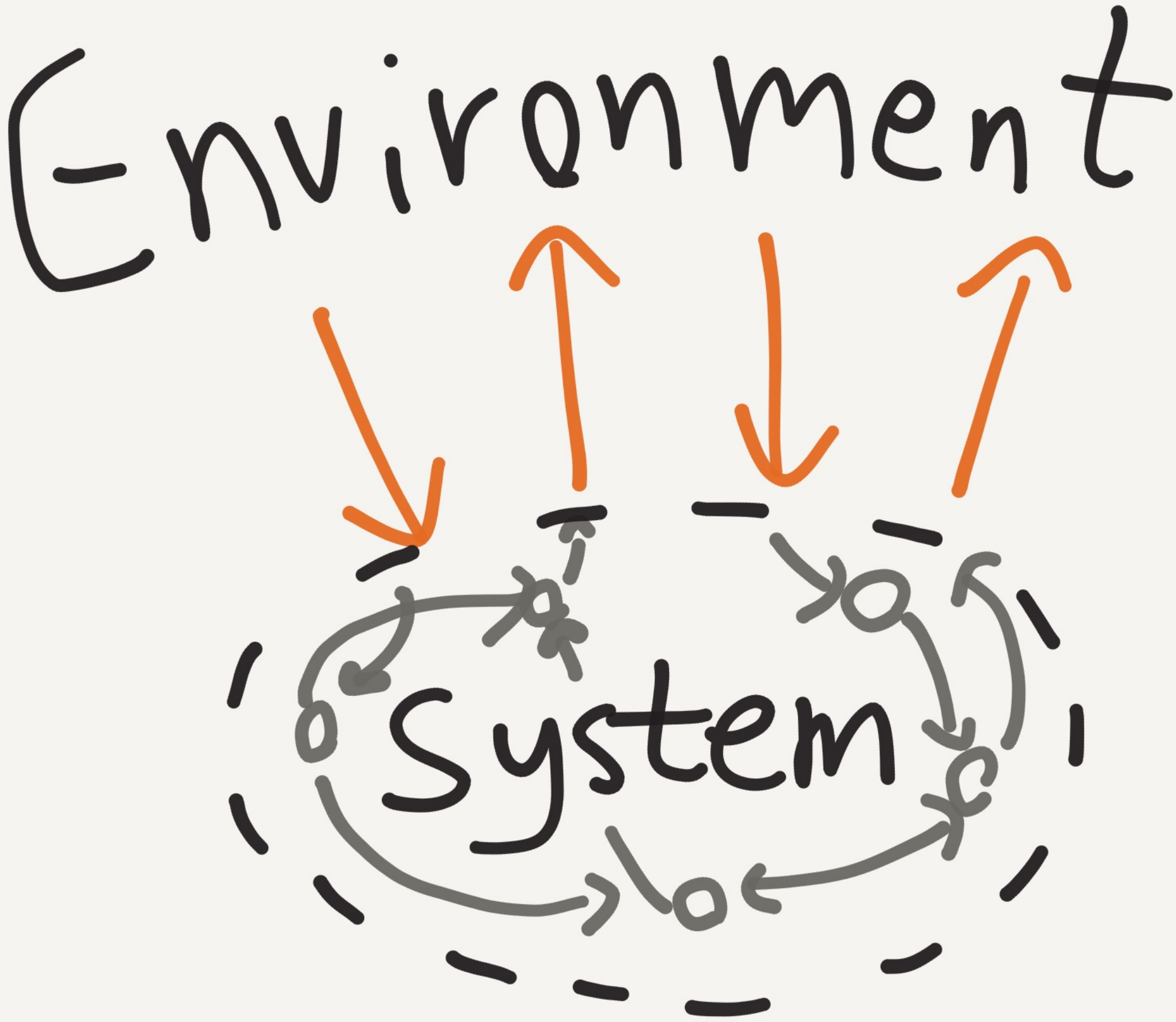
How do you
know if a
system is safe?

Measure your Systems

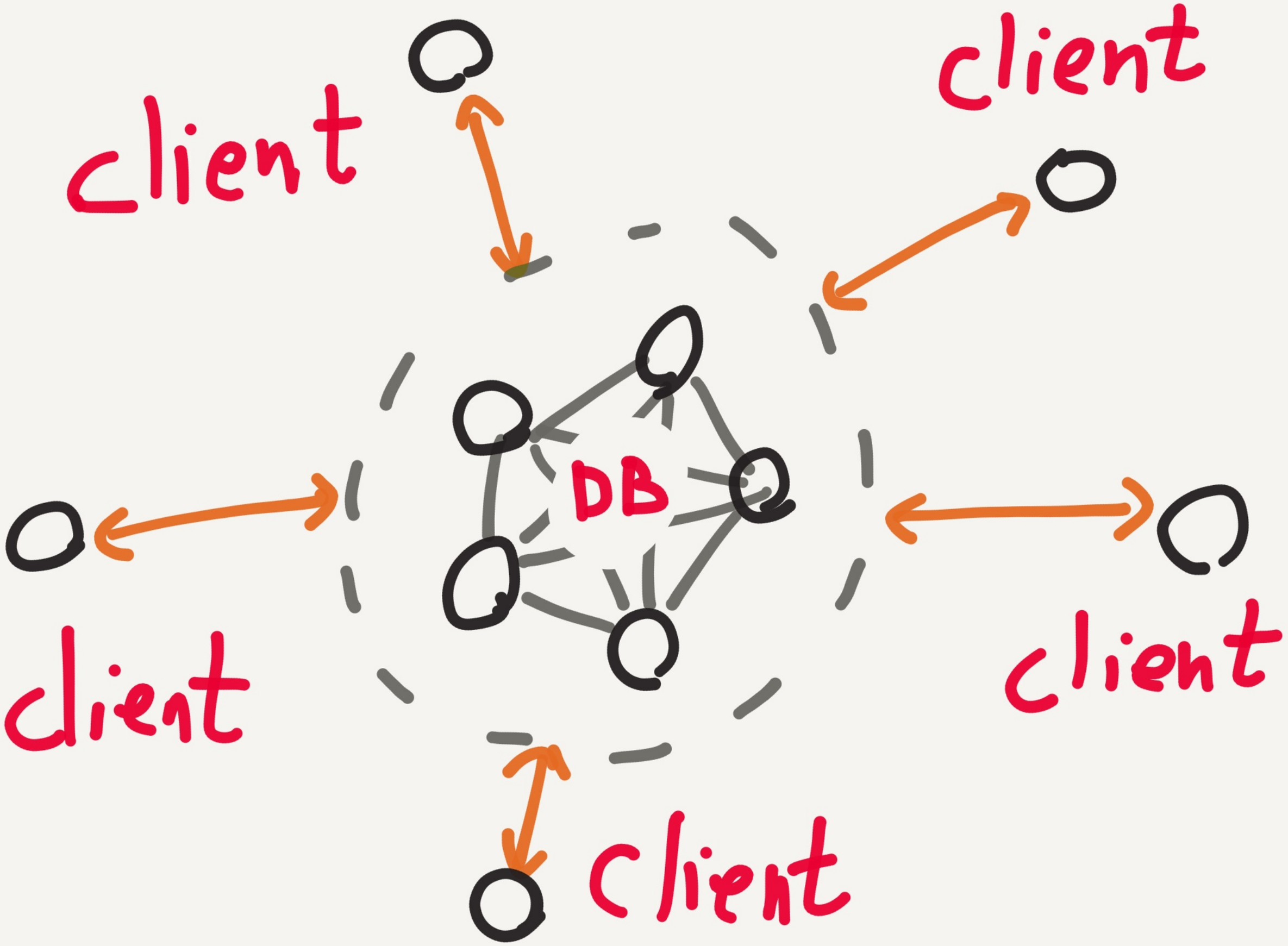


Jepsen

github.com/aphyr/jepsen


Environment
System

-INVARIANTS-

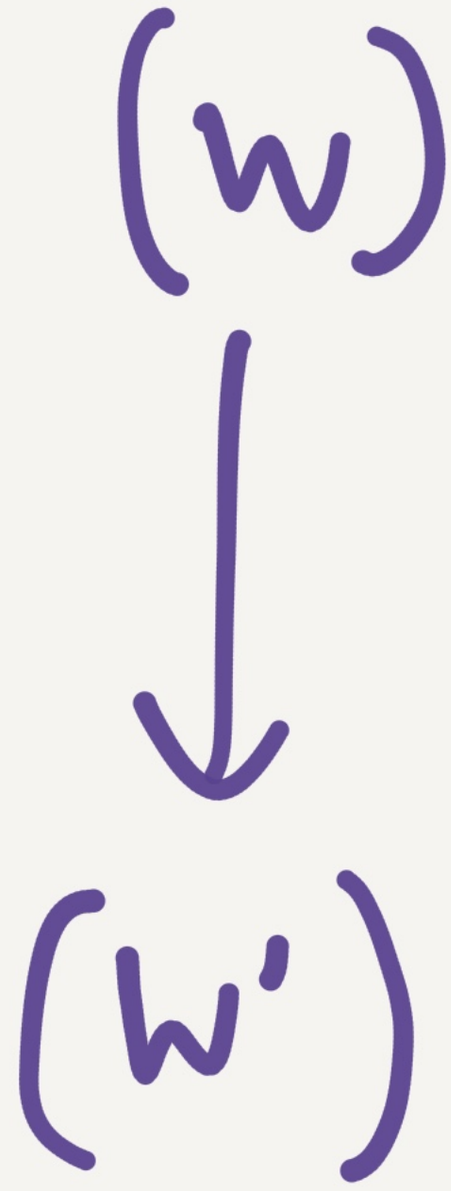

client
client
DB
client
client
client

client: w—w' r—r' w—w'

DB:

client : w —— w' w —— - - -

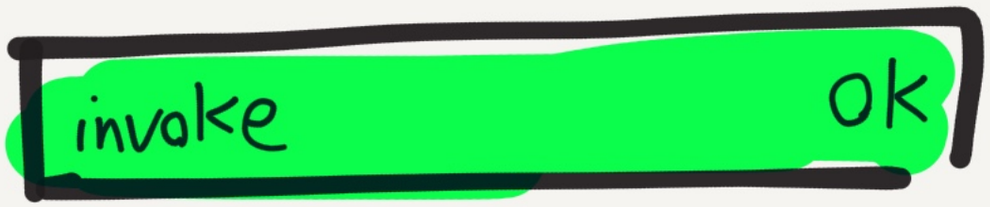
Clients Generate random operations (w) and apply them to the system



invoke ok

invoke fail

involce ? ? info ? ? ?

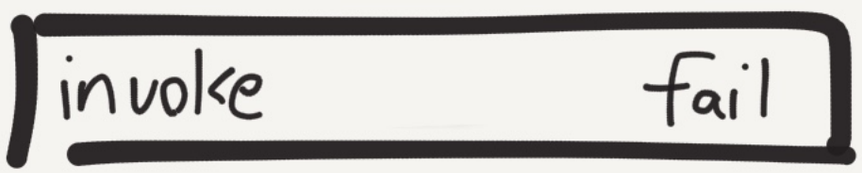

invoke
ok


invoke
fail


invoke
ok


invoke
info


invoke
ok


invoke
fail


invoke
ok


invoke
info

invoke ok

invoke fail

invoke ok

invoke info

invoke ok

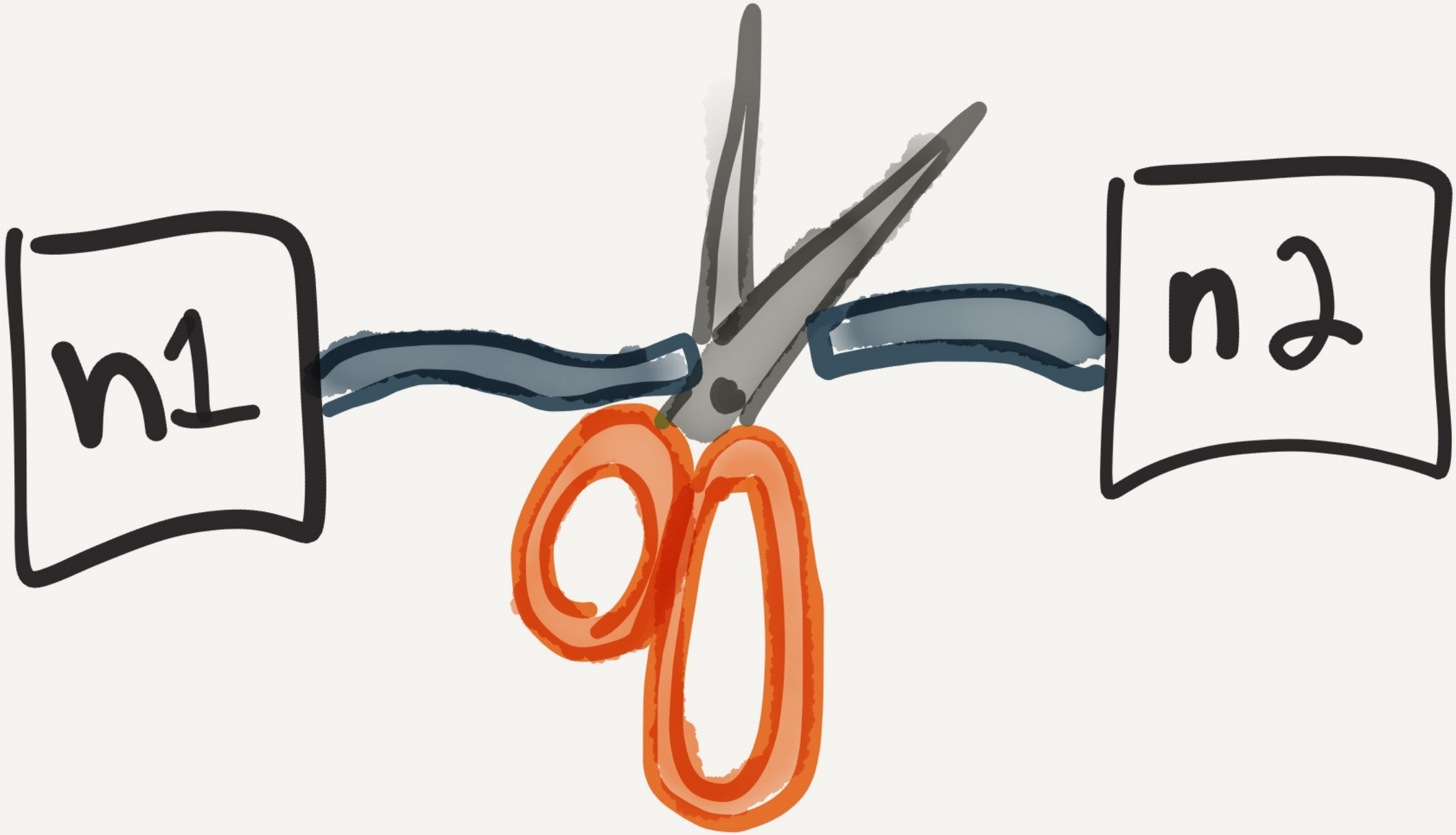
invoke fail

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

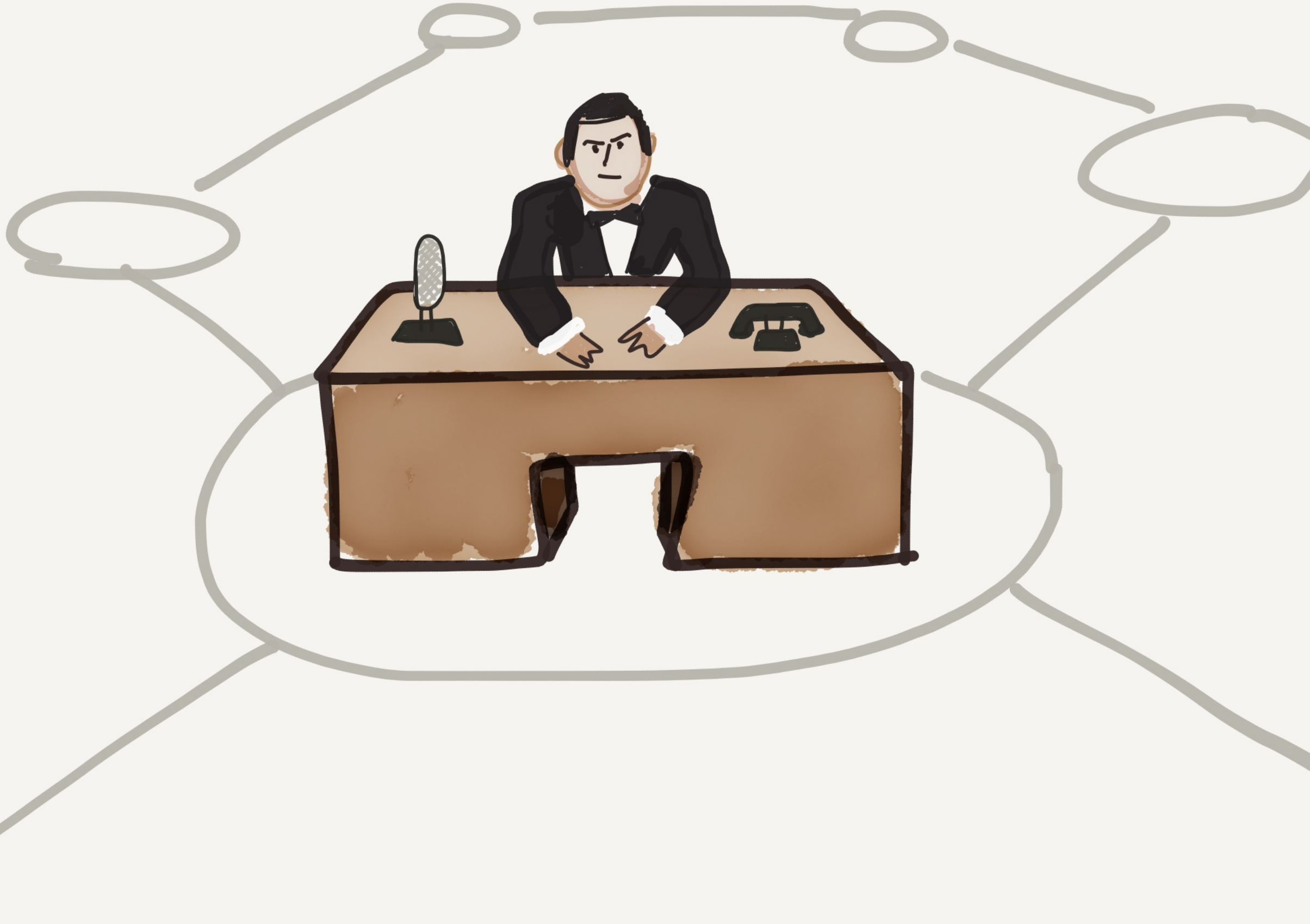
Chronos

8/15

- Breaks forever after losing quorum

Percona XtraDB/Galera 9/15

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



RethinkDB

- Document Store
- Like MongoDB
  - Linearizable on 1 Key
  - No multi-doc txns

Now uses

to offer

consensus!

reads

writes	majority	single	outdated
majority	Linear	Dirty	Reads
single	LOST UPDATES	Stale Reads	

During Partitions,

- brief latency spike
- brief downtime
- Recovery in ~10s

Outdated reads always available

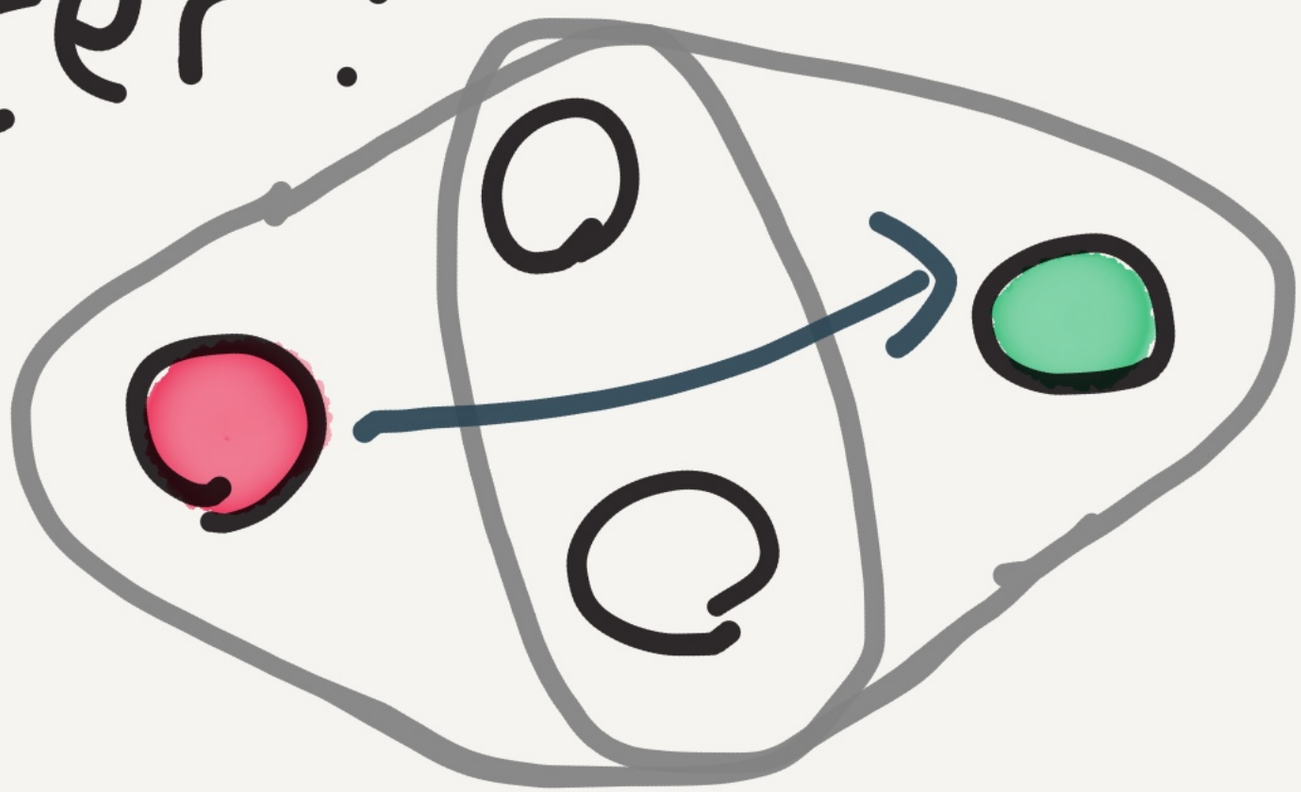
All other ops require a majority connected

Majority/Majority

PASSED

We have to go Deeper

What if... we rebalanced the cluster?



still passes. Ok.

Let's rebalance

during partitions

lost update!

3

write 0

17

cas 3→0

time ⟶

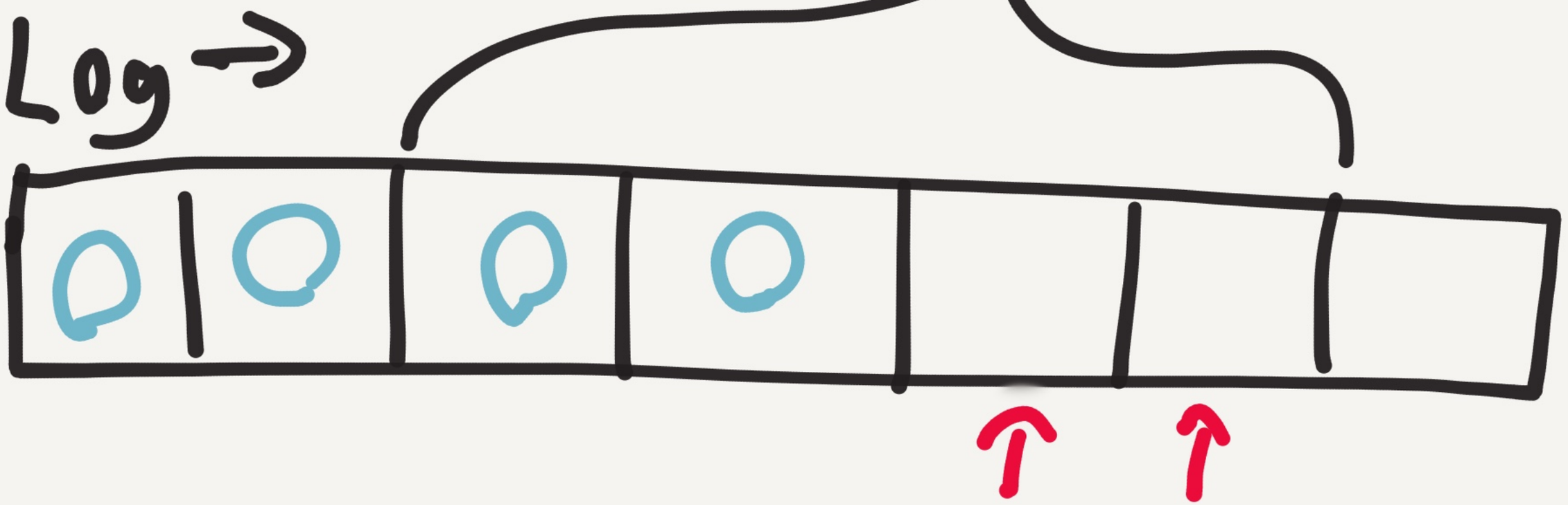
"Guarantee failed:

[index <= get_latest_index]

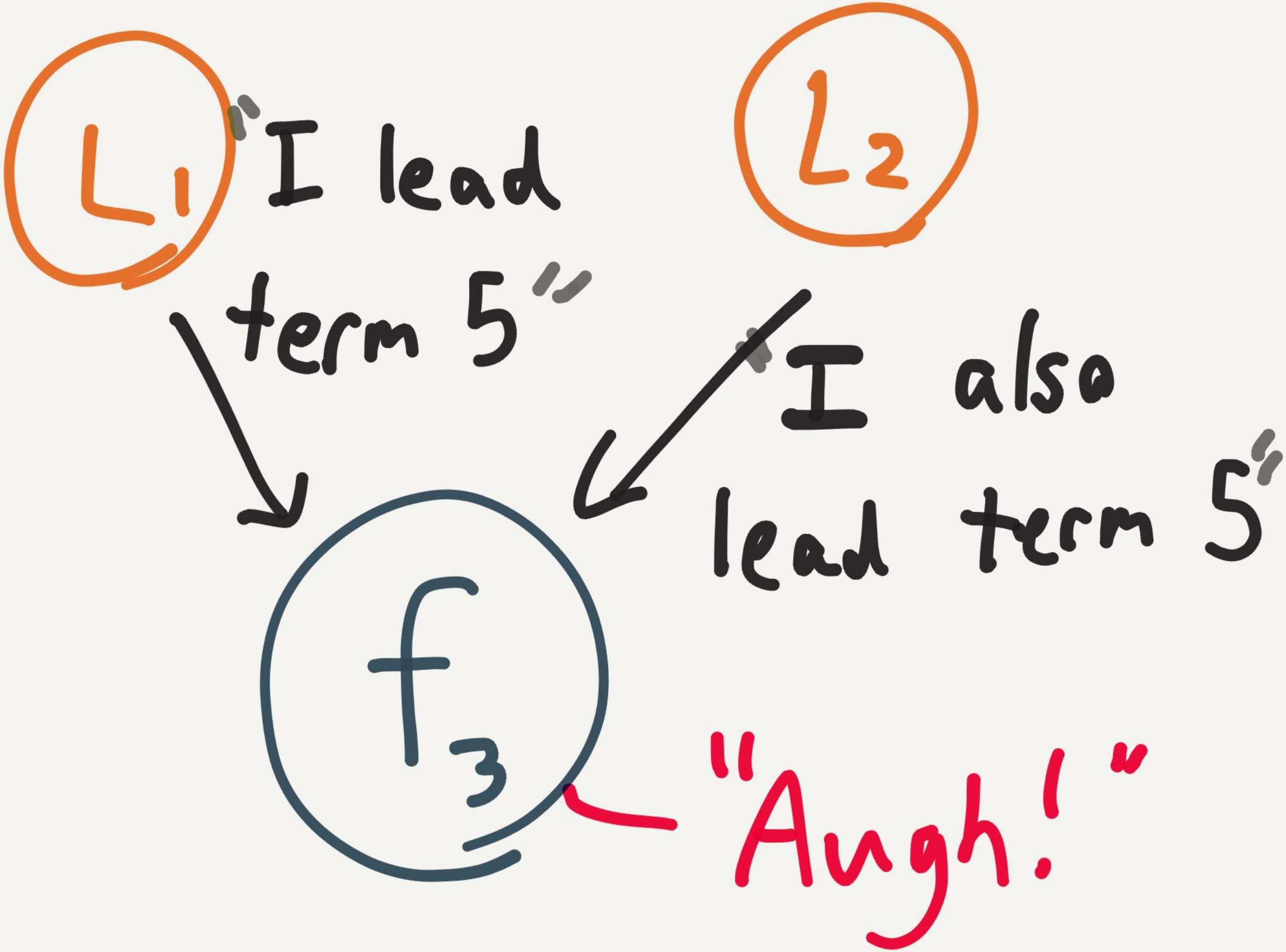
the log doesn't go

forward this far"

Leader: "please apply this range to your SM"



"Guarantee failed:
[current_term_leader_id
== request_leader_id]"


L1
"I lead term 5"
L2
"I also lead term 5"
f3
"Augh!"

seq #

raft ID

START	1	foo
STOP	2	foo

START | 1 | foo

STOP | 2 | foo

dup

START | 1 | foo

START | 1 | foo

STOP | 2 | foo

dup

START | 1 | foo

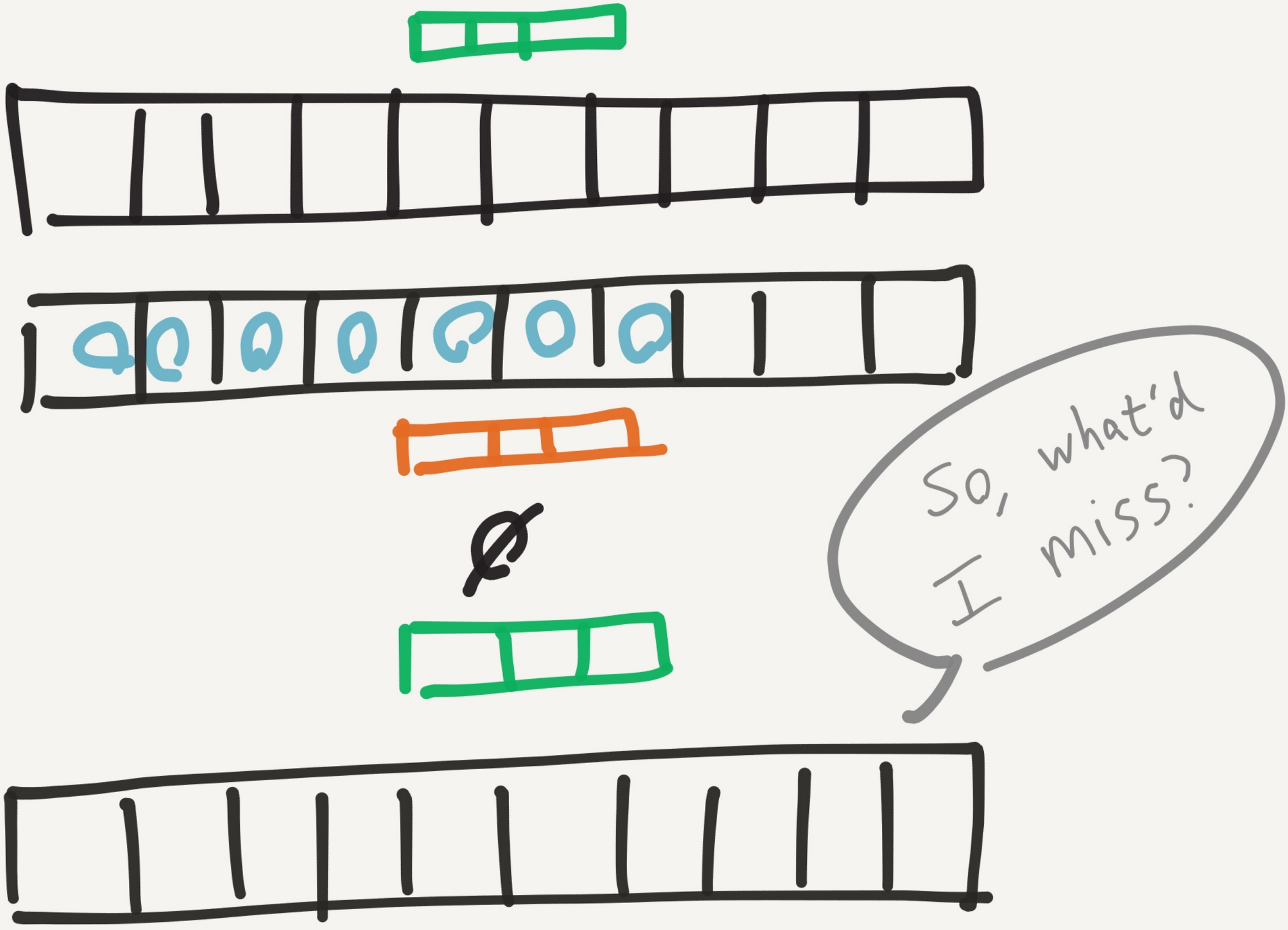
. . . mostly .

#4668

In version 2.1.0,
seq #s set to
MAX_INT.

Workaraund:

Always process ACTIVE messages


So, what'd I miss?

Patches in

2.2.4 &

2.1.6

6.3

Distributed SQL Database

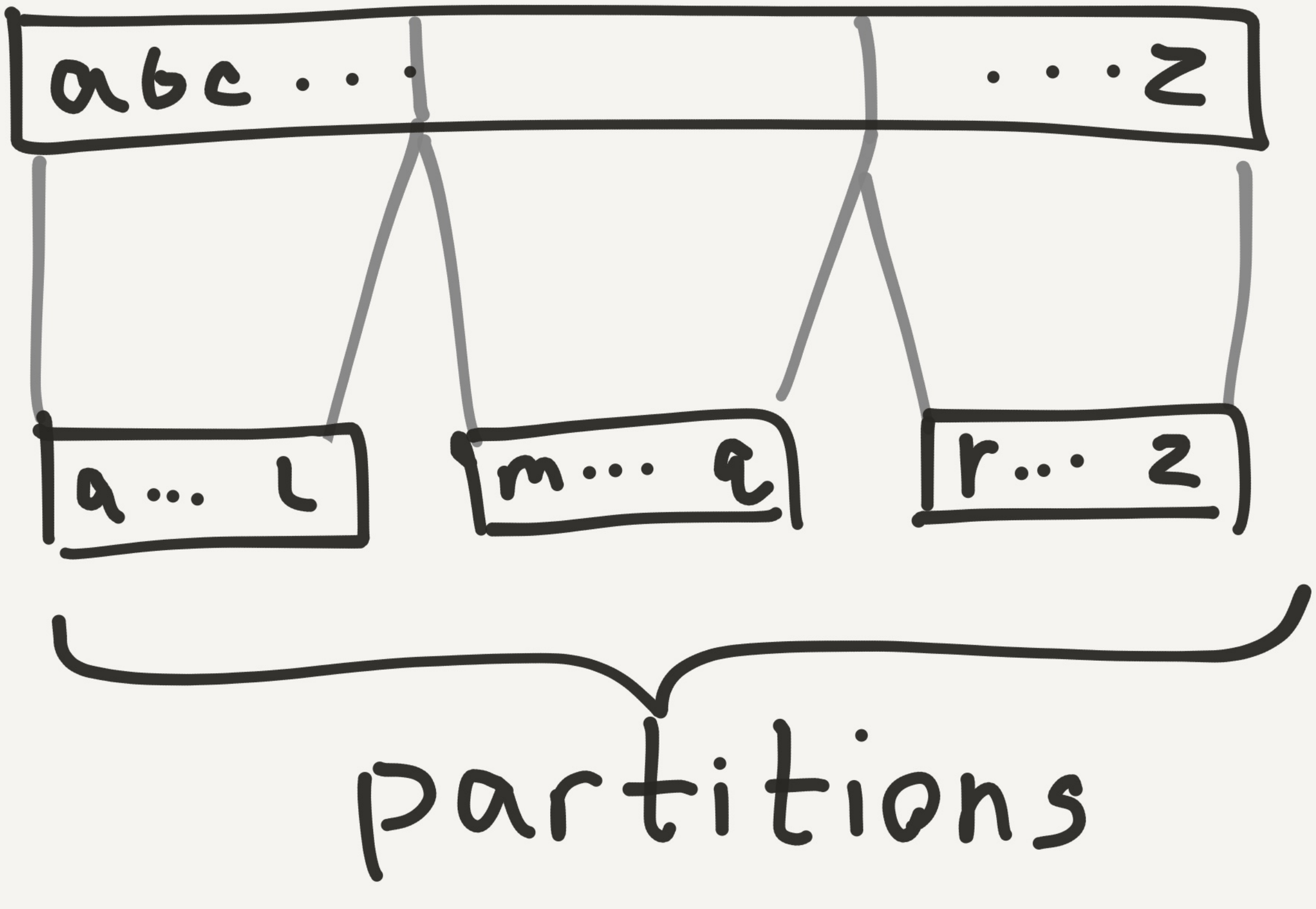
All data in RAM
(but persistent)

Basic SQL queries

Txns as Java stored procedures

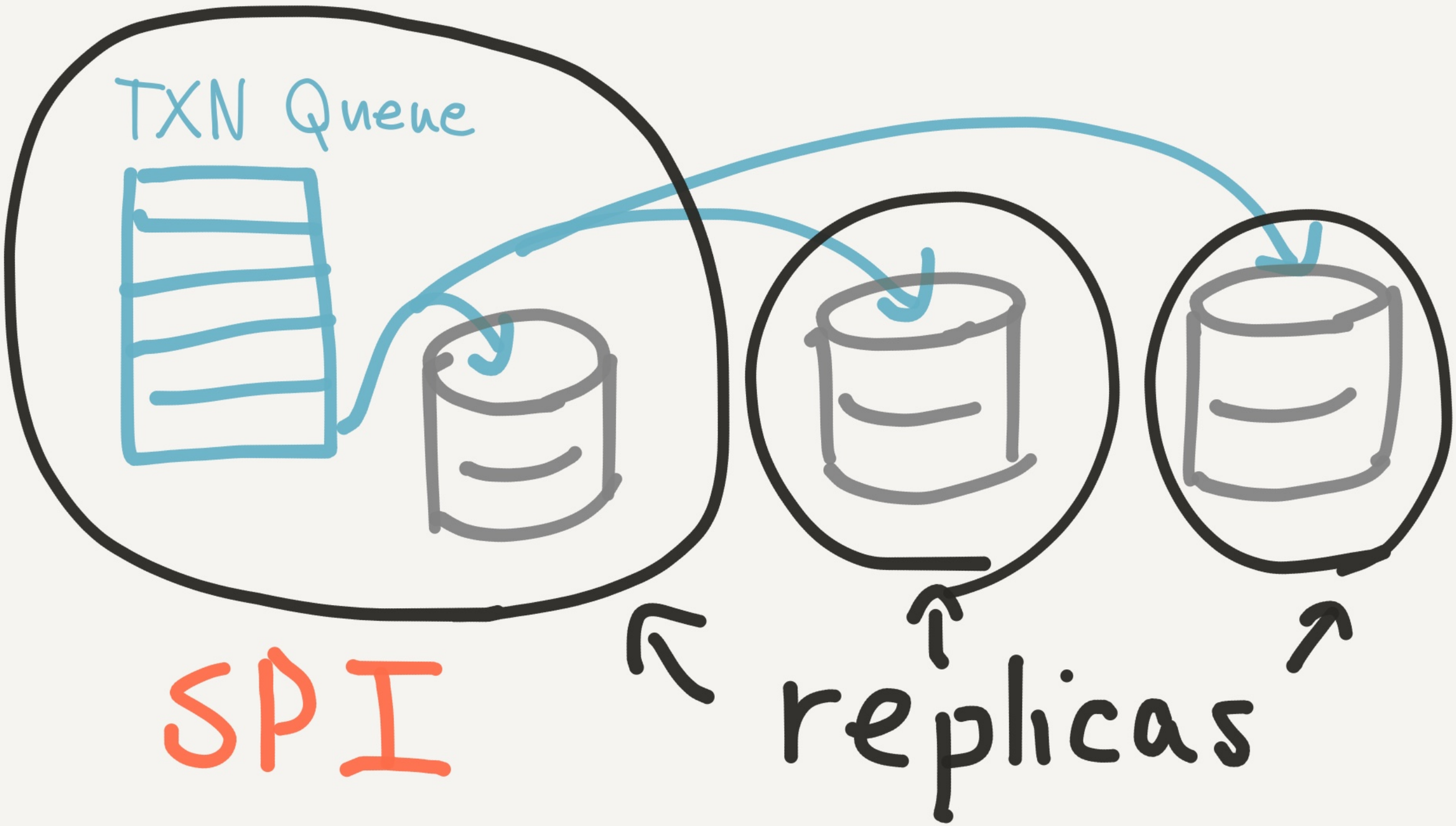
Intended for high-throughput, mostly sharded transactions

Keyspace

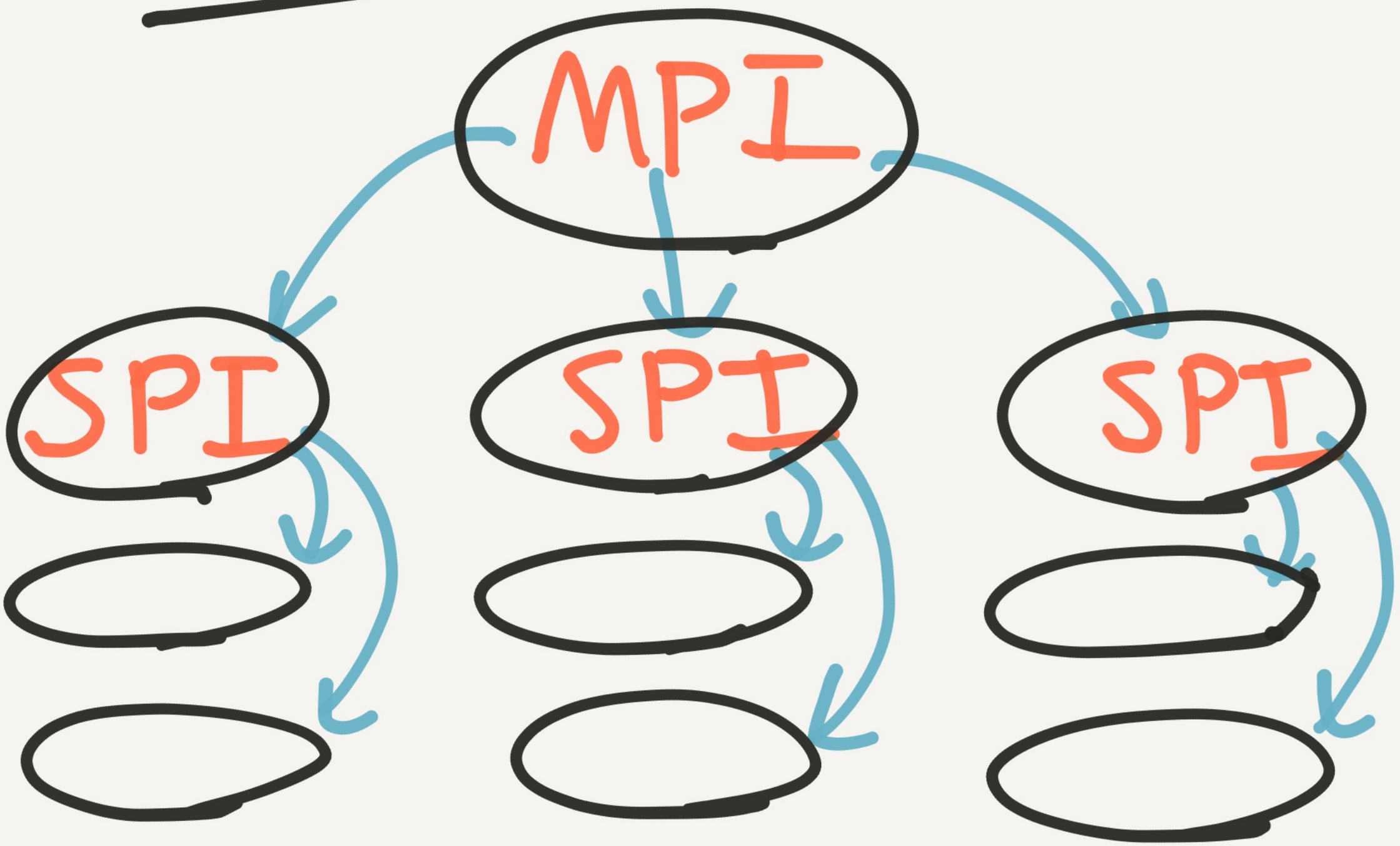


partitions

A partition



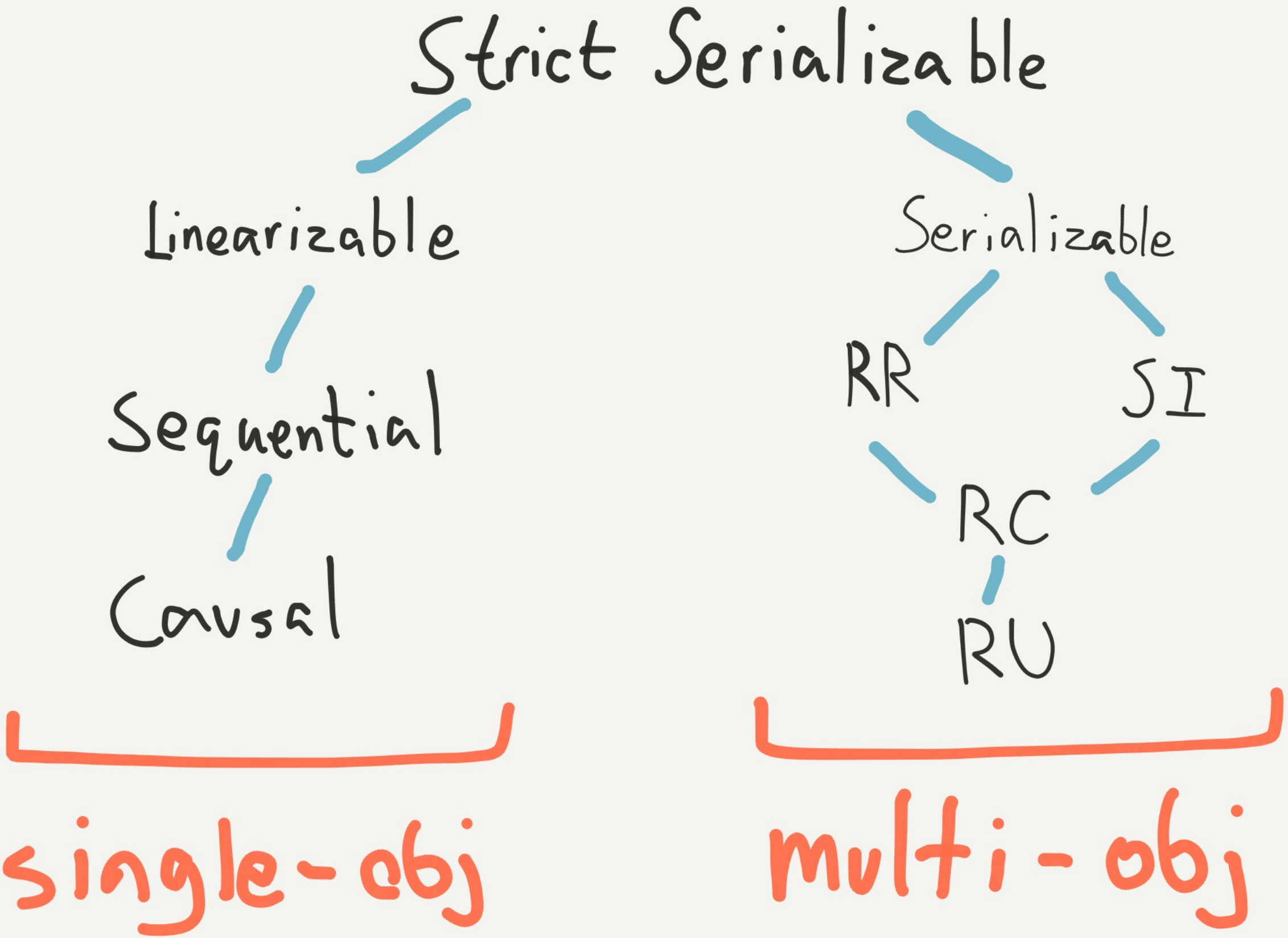
Across Partitions



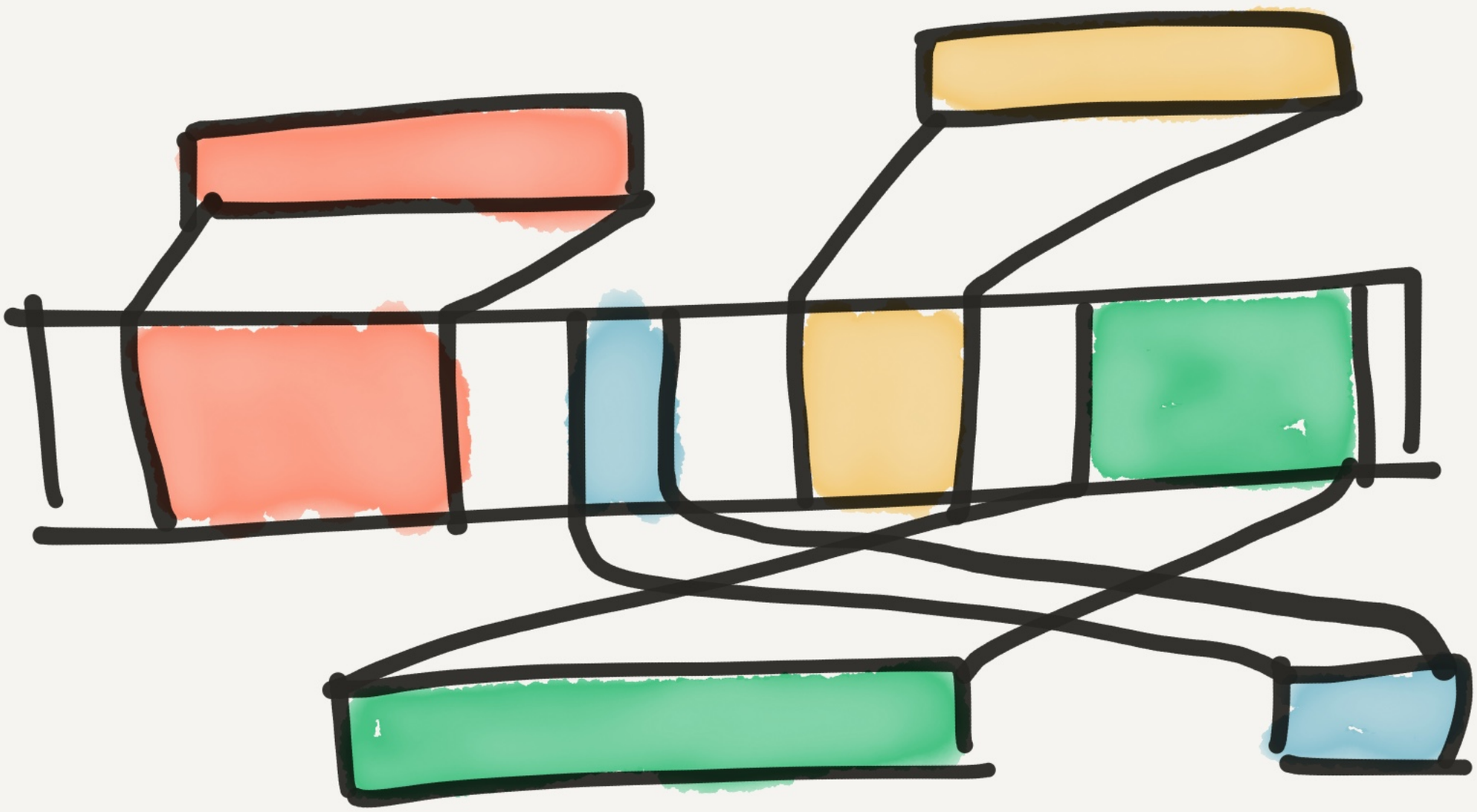
- Txns in 2 partition scale ~linearly

- Txns across partitions have const throughput

Claim: all txns are strict serializable!

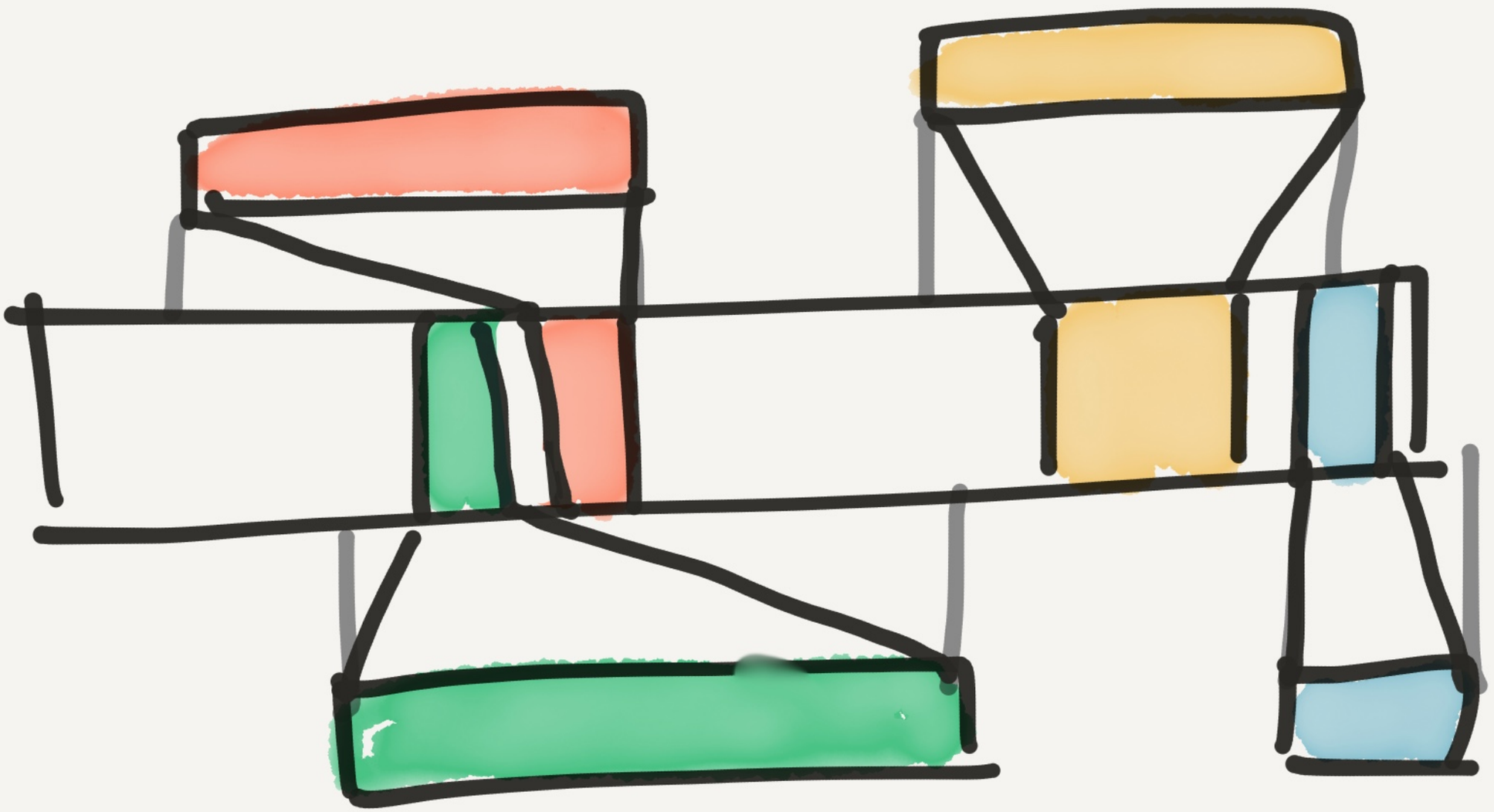

Strict Serializable
Linearizable
Sequential
Causal
Serializable
RR
SI
RC
RU
single-obj
multi-obj

Serializability



time →

Linearizability



time →

Strict Serializable

- Real-time
- Multi-object

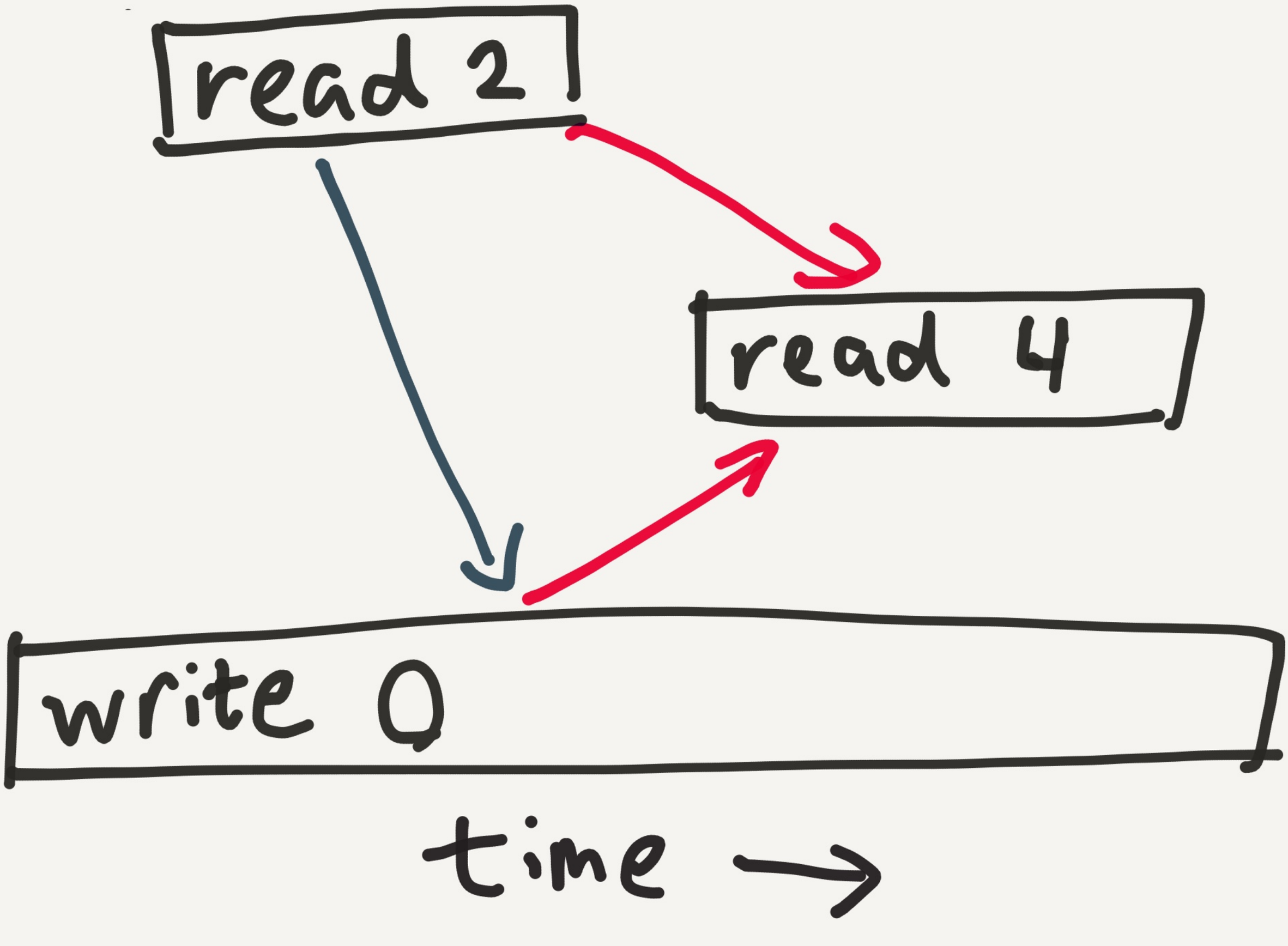
VoltDB 6.3 exhibits several violations of strict serializability

Stale Reads

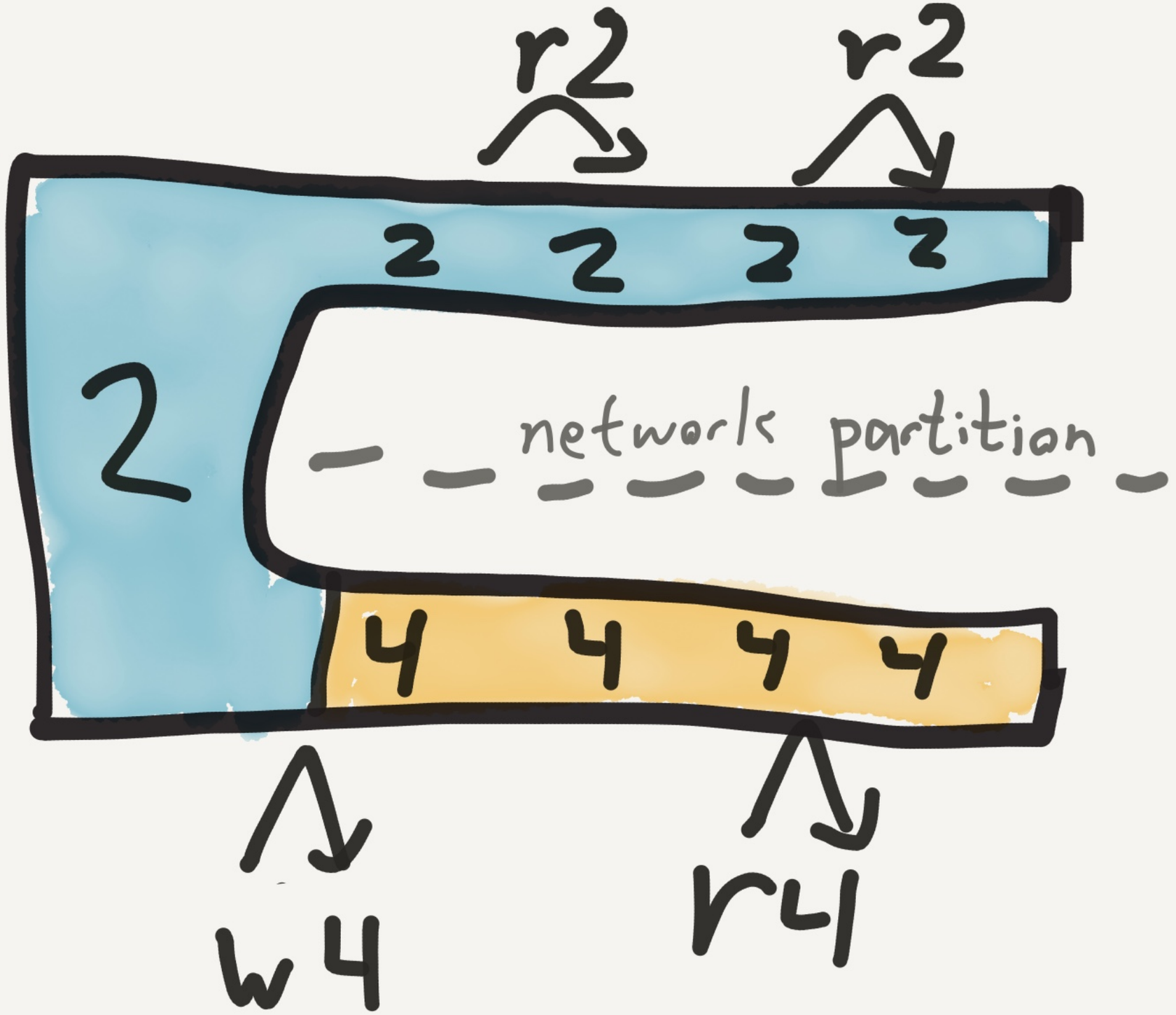
Dirty Reads

Lost Updates

Stale Reads

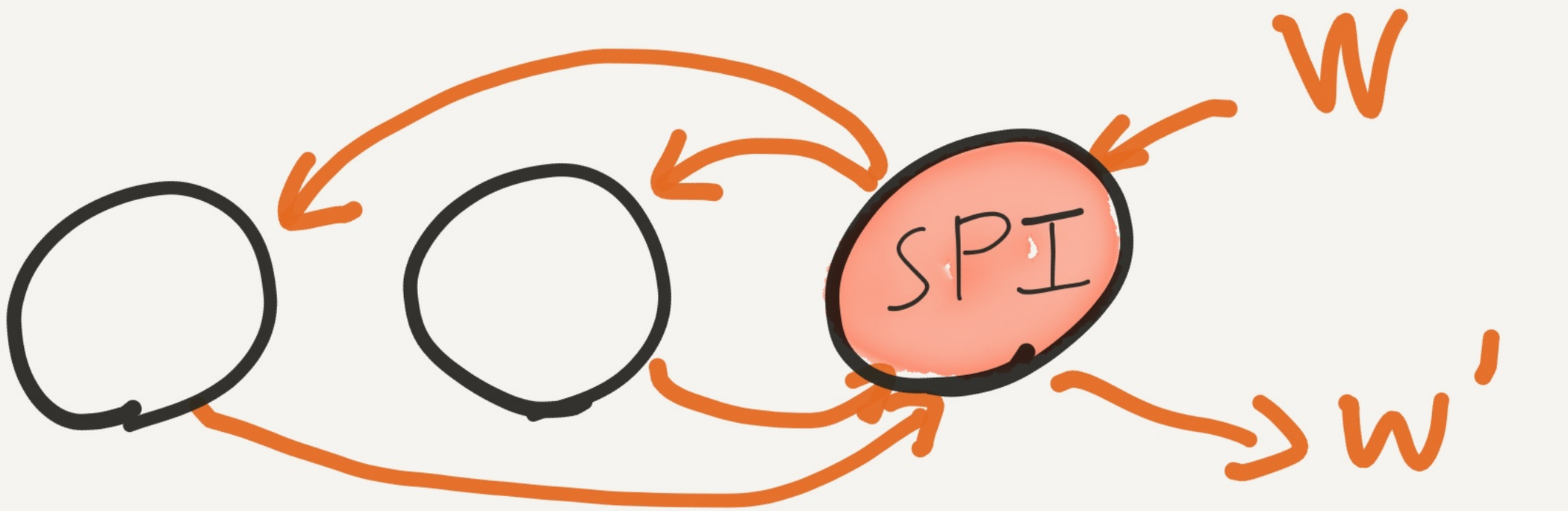

read 2
read 4
write 0
time →

node 1



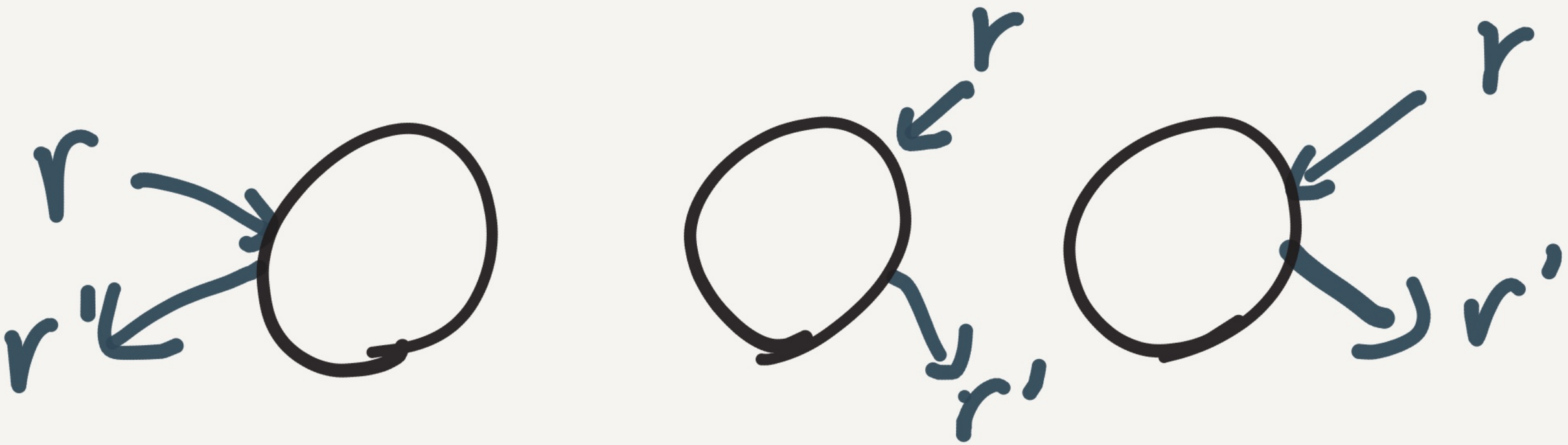
node 2

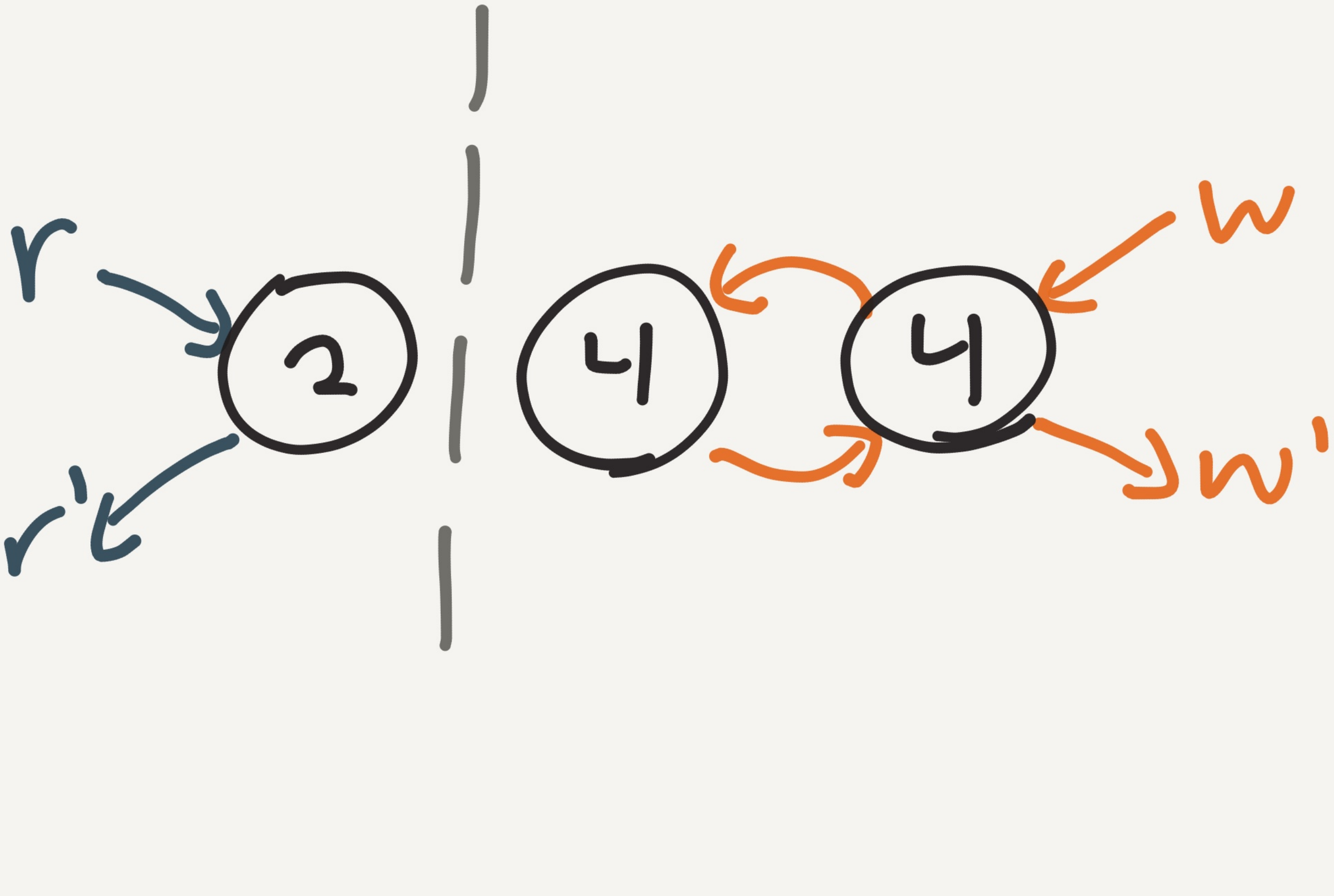
For a given partition



SPI orders updates

...but not reads




r
2
4
4
w
r'
w'

Solution:

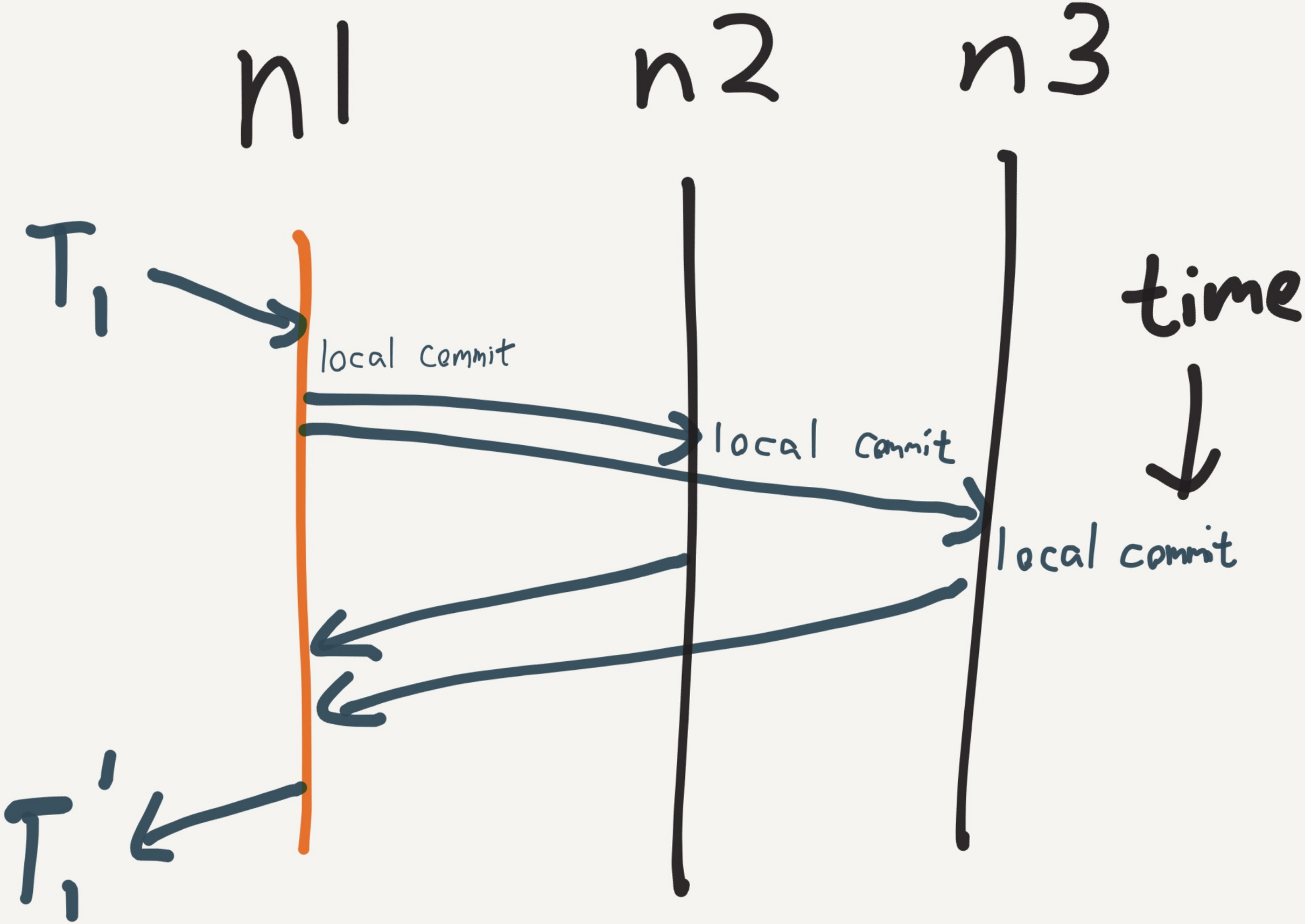
Push all txns through the SPI ordering path

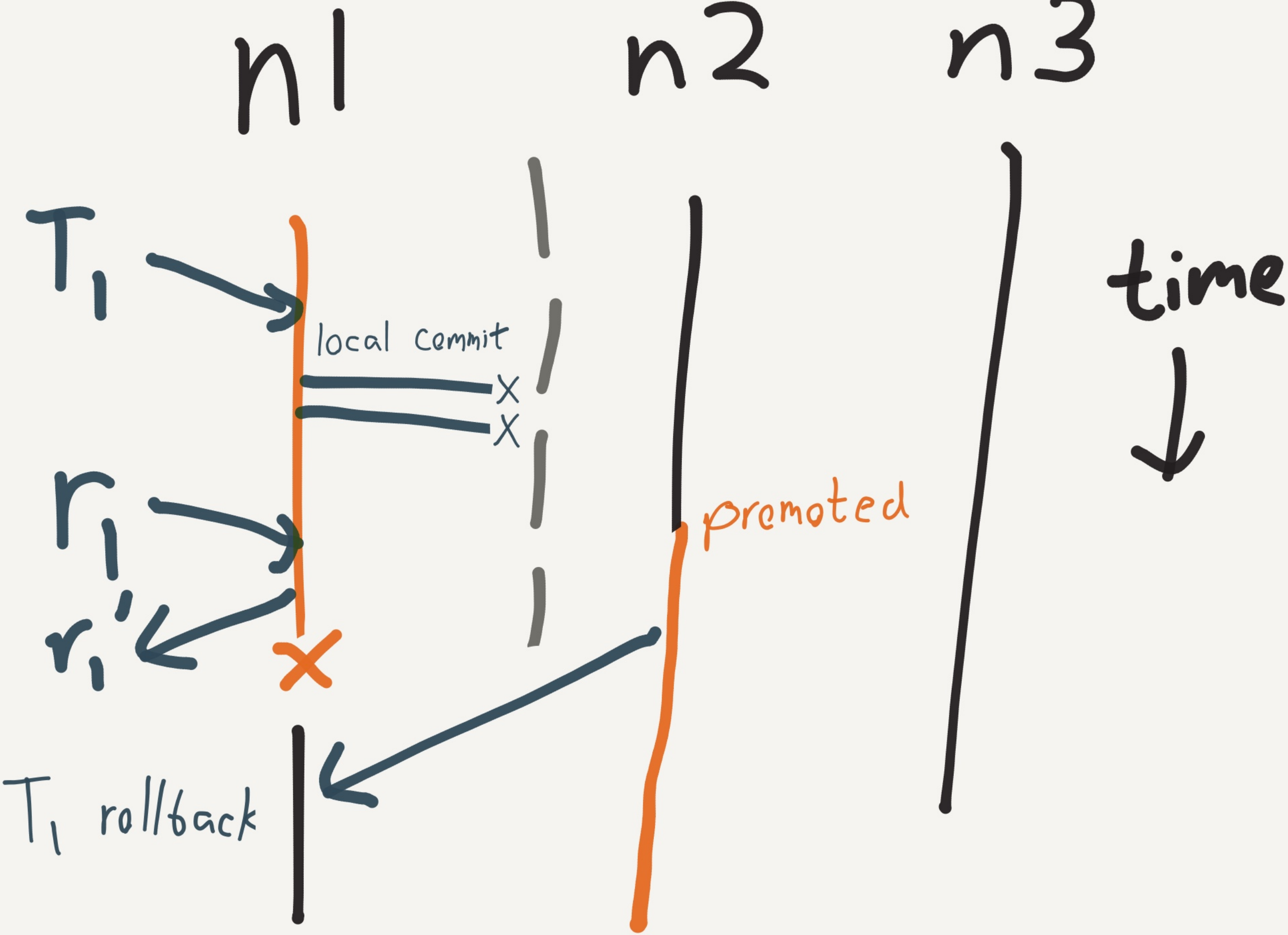
Dirty

Reads

T_1 w3... abort

T_2 r3

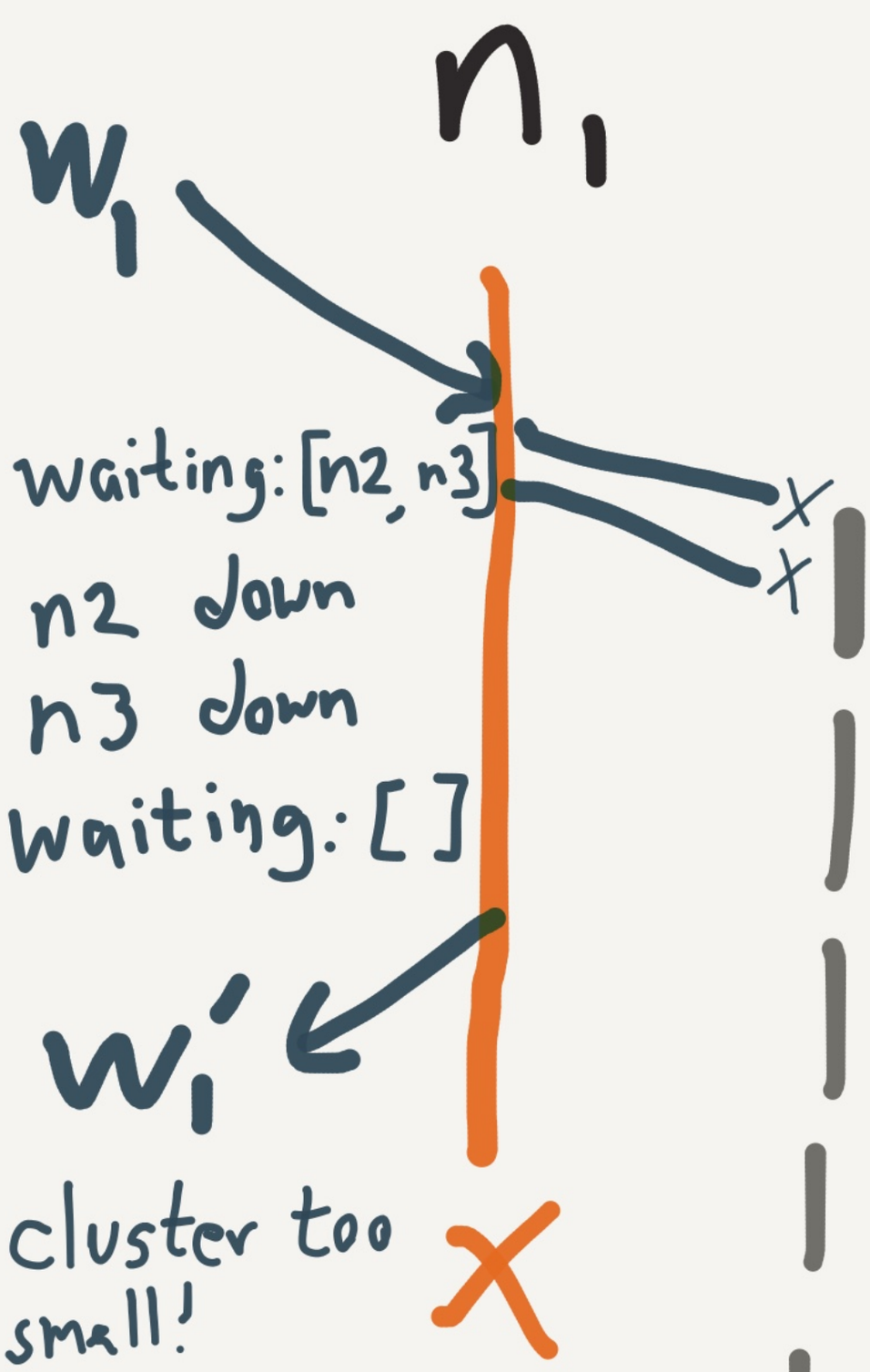

n1
n2
n3
T1
local commit
local commit
time
local commit
T1'

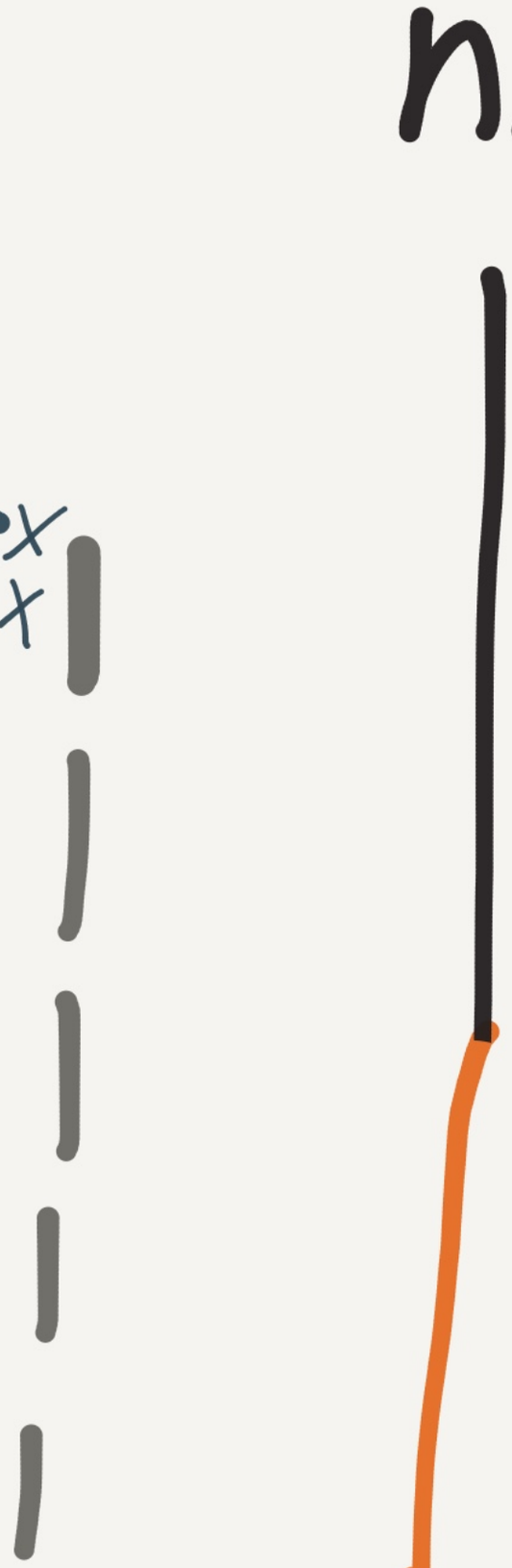

n1
n2
n3
time
T1
local commit
X
X
r1
r1'
promoted
T1 rollback

Lost Updates

T_1 : W_1 ← lost!

T_2 : r_0


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


n_2


n_3

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

Solution:

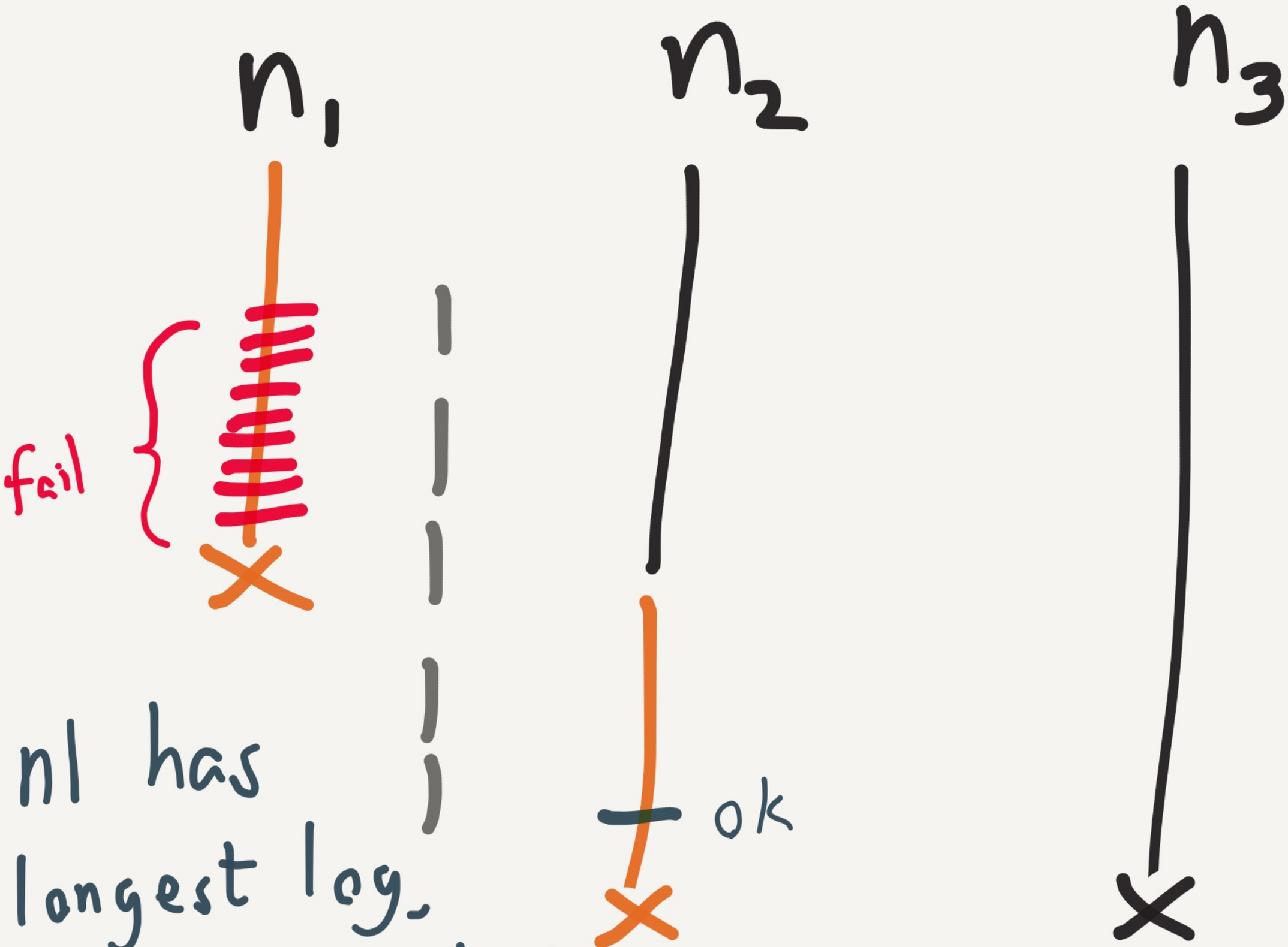
Make cluster shrink

by 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

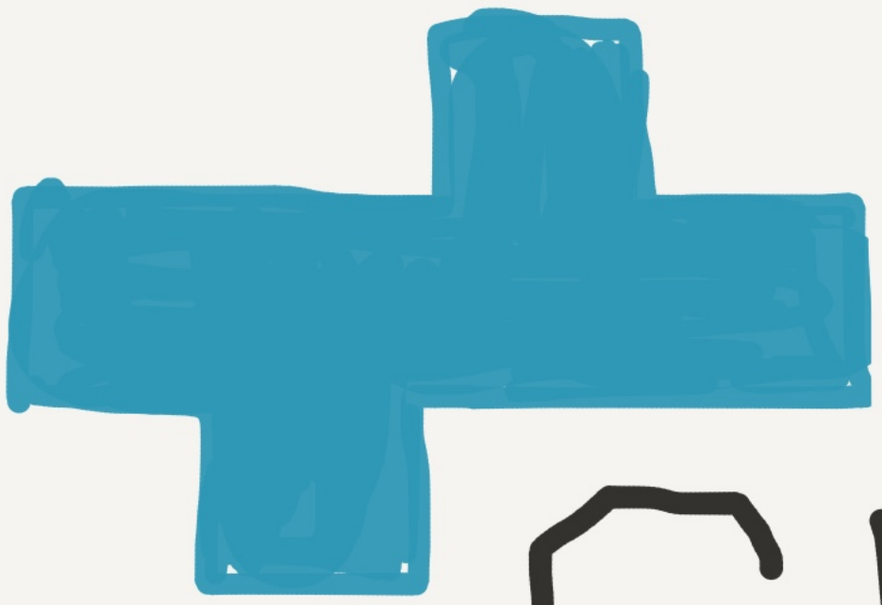
MPI fully serializes read and write txns, may prevent divergence

VoltDB 6.4

Passes all Jepsen tests for strict serializability

Future works

- formal models
- Partial failure
- Clocks skew



CRATE.IO

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

CRATE

ElasticSearch

ES 1.1.0: extensive data loss

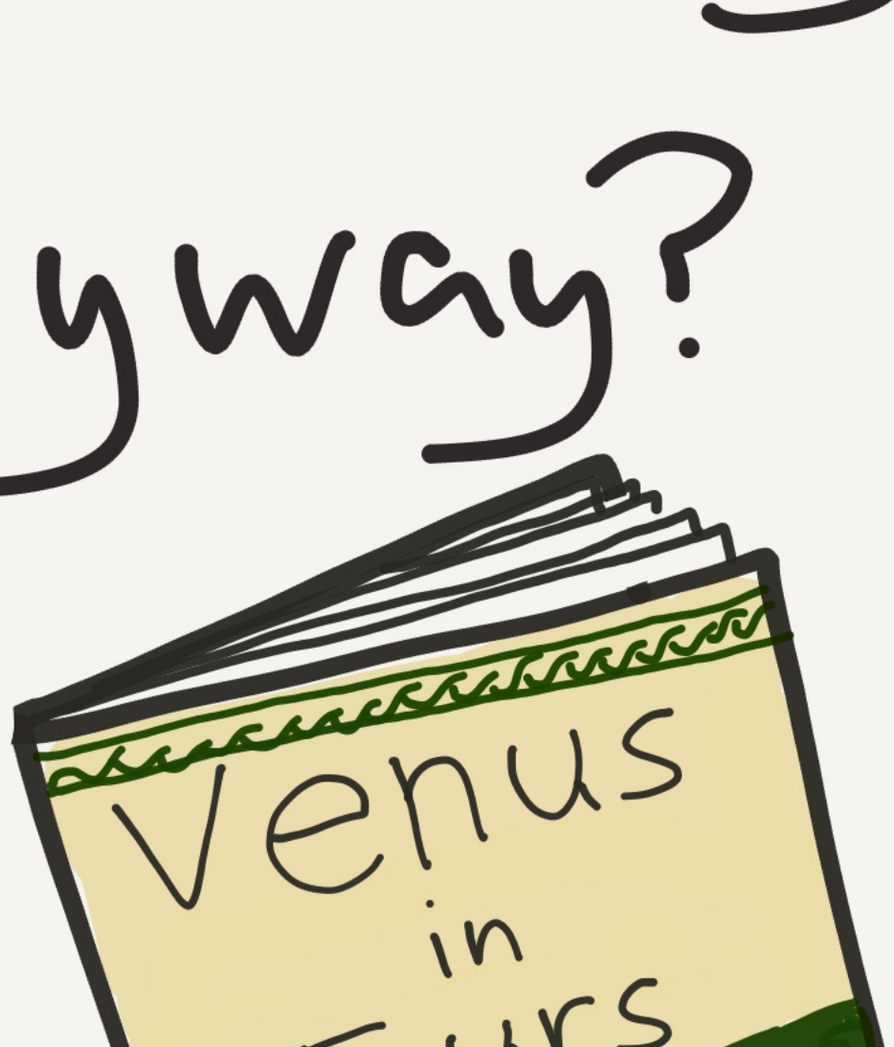
ES 1.5.0: less extensive data loss

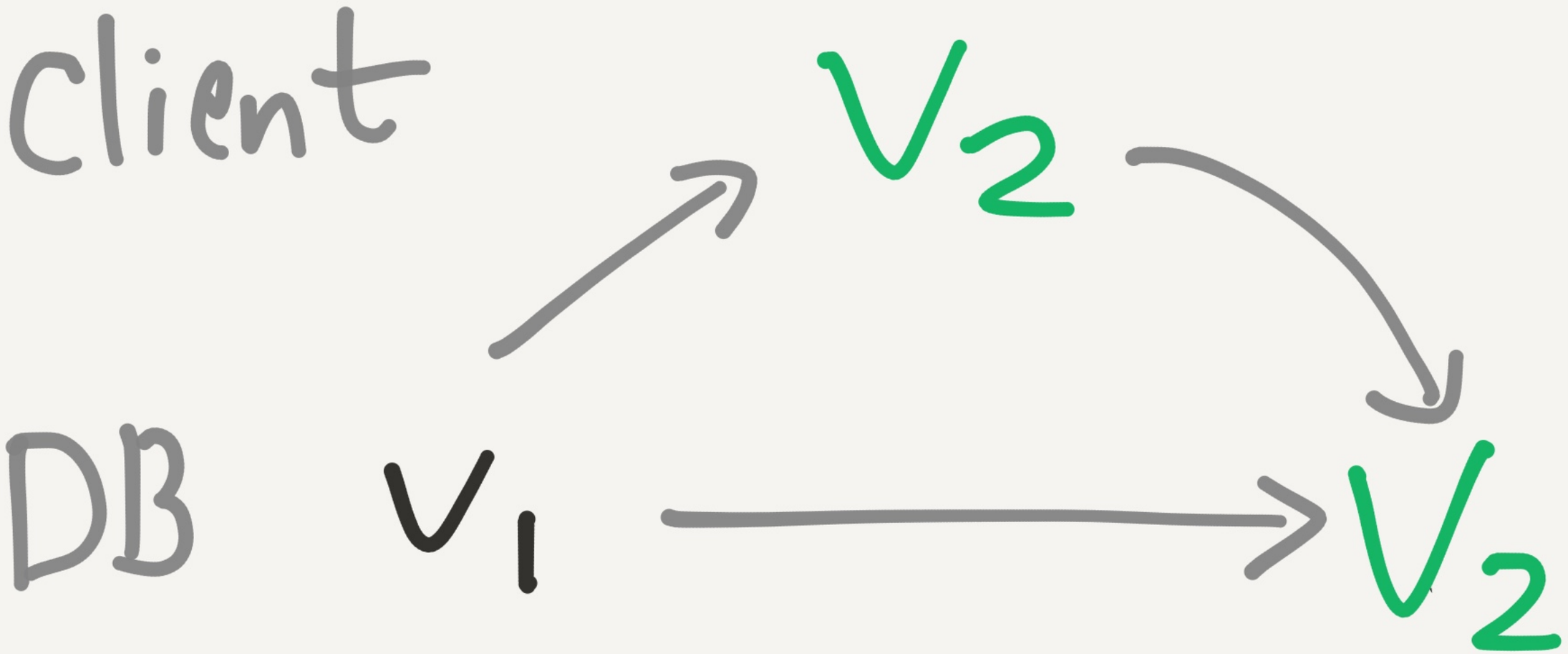
Crate uses 1.7.

Elasticsearch 1.7

- stale reads
- dirty reads
- lost updates

Ok but how bad are dirty reads anyway?





Sequential updates

client A

V_2'

DB V_1 → V_2 → V_3

client B

V_2'

X

Are versions actually distinct?

an test

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

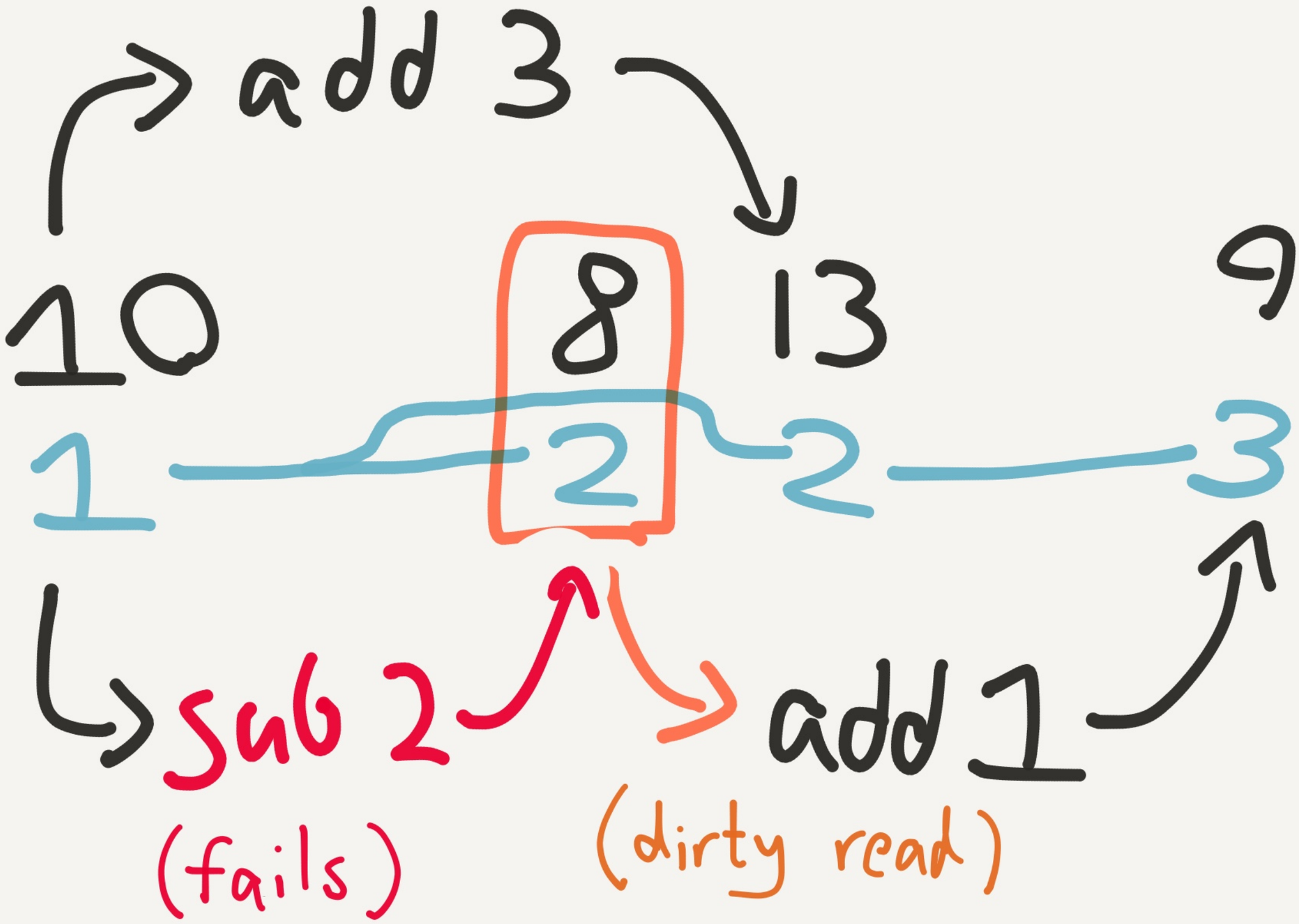
- write unique values
- read value, version

{:value 1, :version 2}

{:value 3, :version 2}

value

Version


add 3
10
8
13
9
1
2
2
3
sub 2
(fails)
(dirty read)
add 1

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

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

Preserves garbage -2

Loses ok update +3

If you read, modify, write, use Read Committed

It gets

worse!

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

Not Crate's 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-alpha5

- dirty reads
- lost updates
- replica divergence

#20031

Dirty reads:

no plan to fix

⇒ no safe update

Lost updates:

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

Replica Divergence

Unacked docs arent 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!

Recap

Read the docs!

CAREFULLY

Then test it

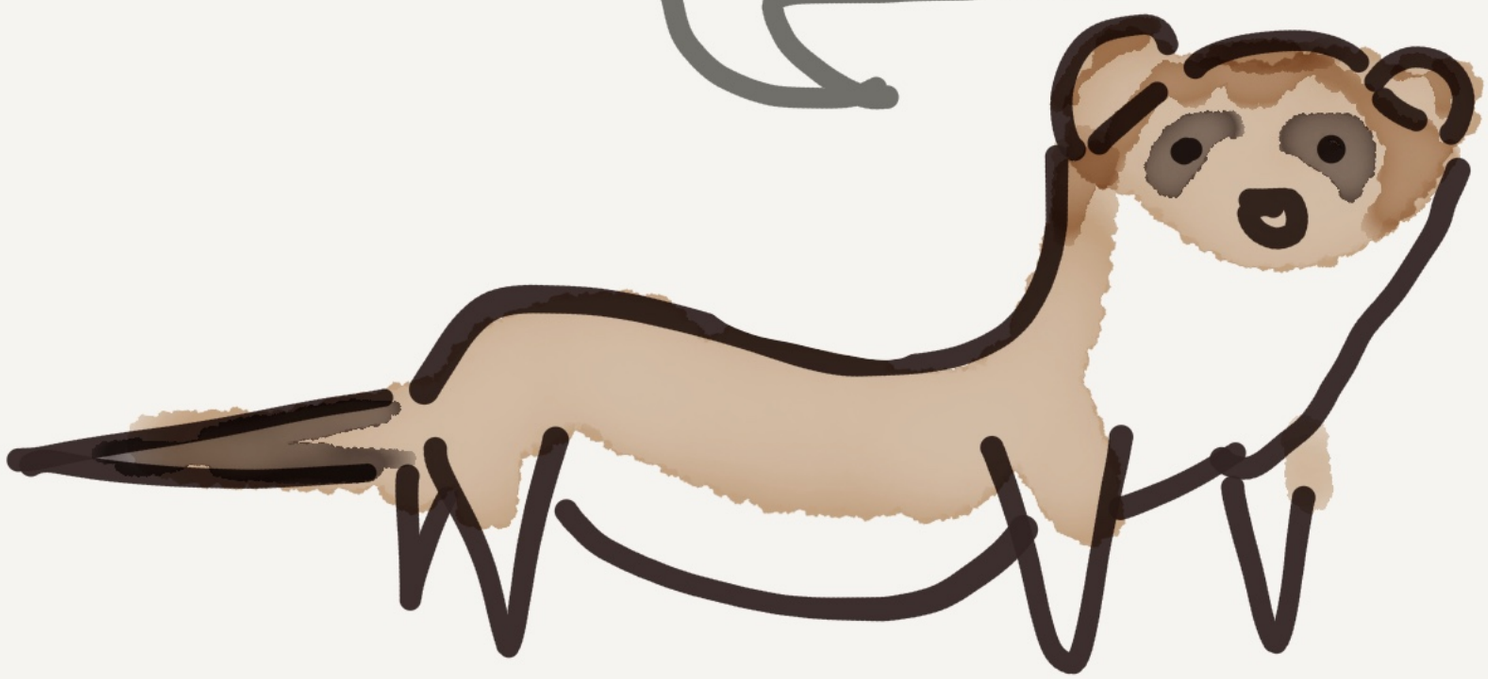
— for —

YOURSELF

"Strict"

"Strong"

"ACID"



Be Formal

&

Be Specific!

Figure out the invariants your system needs

How much

can yan tolerate?

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 ...
```

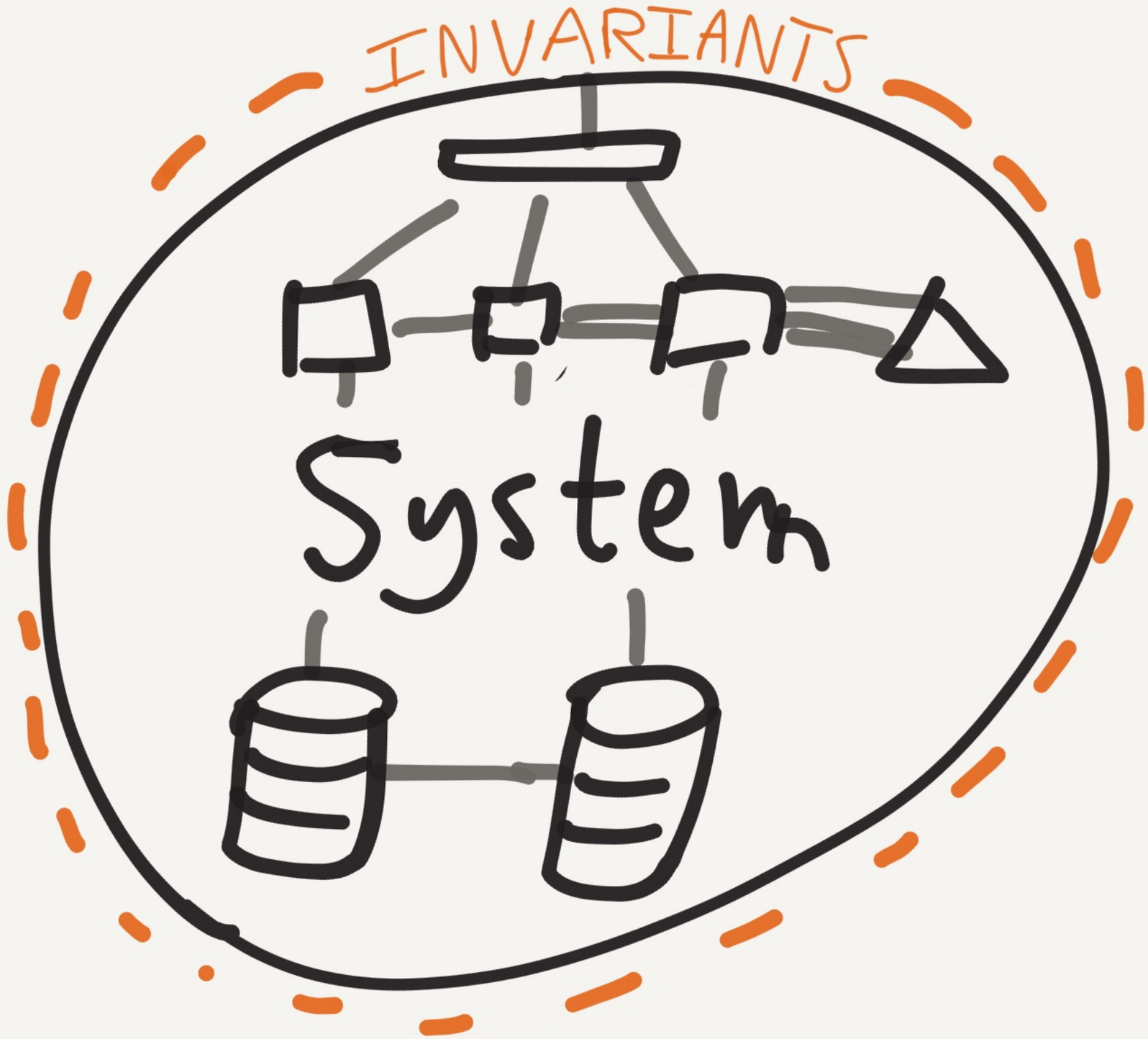
```
drop ...
```

Test your

Systems

end ——— to ——→ end

vs.



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

Thanks

RethinkDB
VoltDB
Crate.io
Boaz Leskes
Peter Alvaro

} Funded Research

http://jepsen.io