JEPSEN

IV

Hope Springs Eternal

@aphyr

(Kyle Kingsbury)

strip

Public API {

} API code

Ruby {

DBs

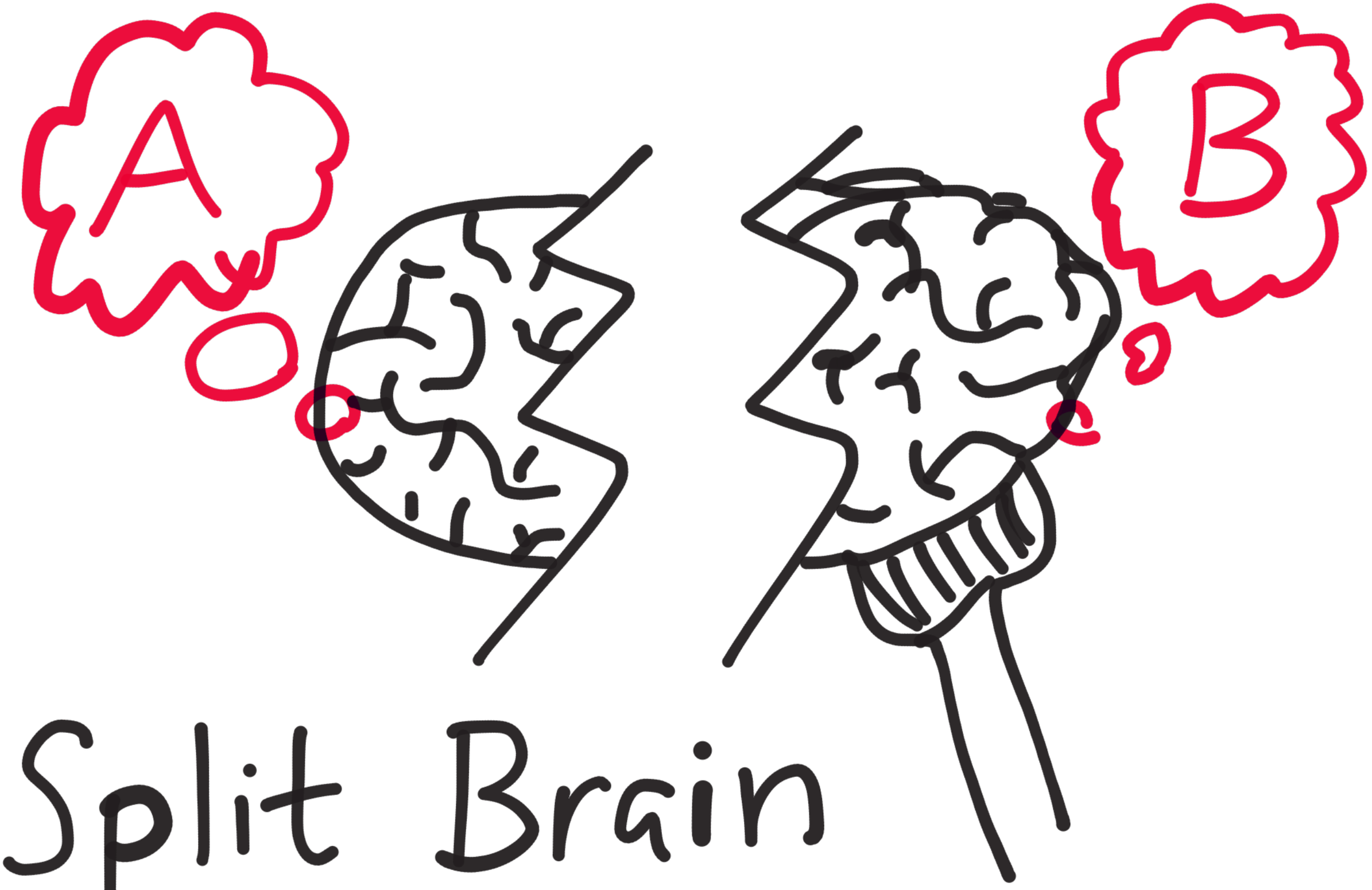
Every thing is fine!

Databases!

Queues!

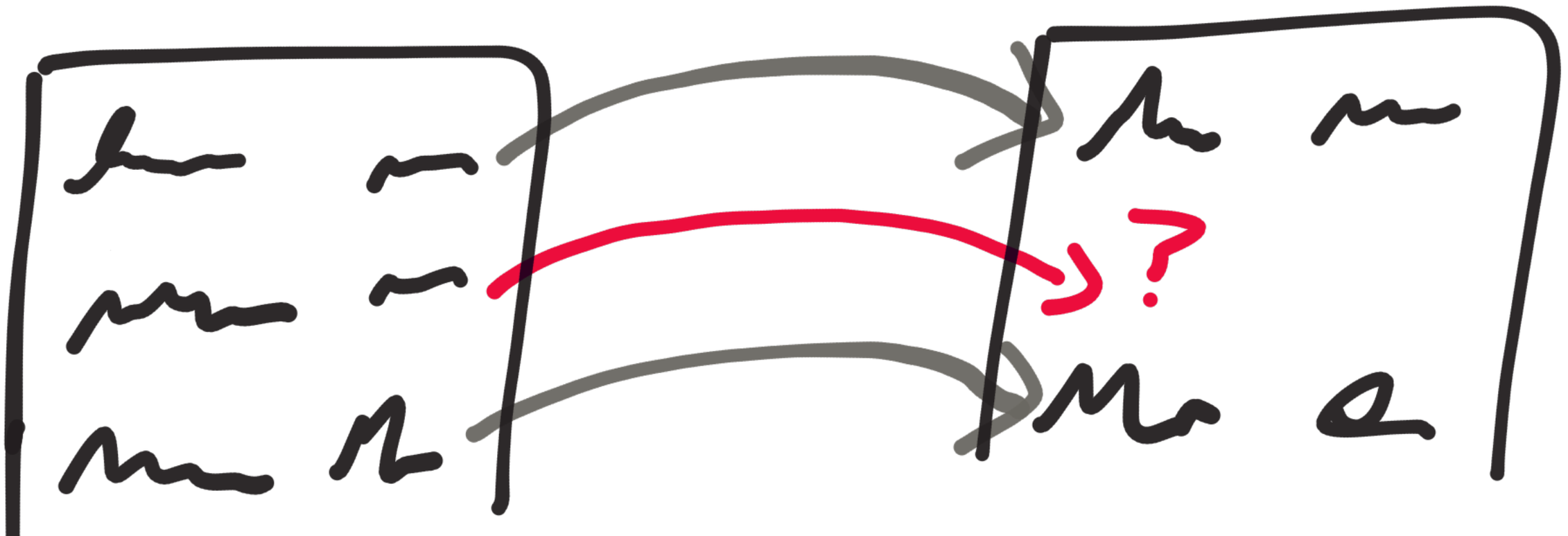
Discovery!

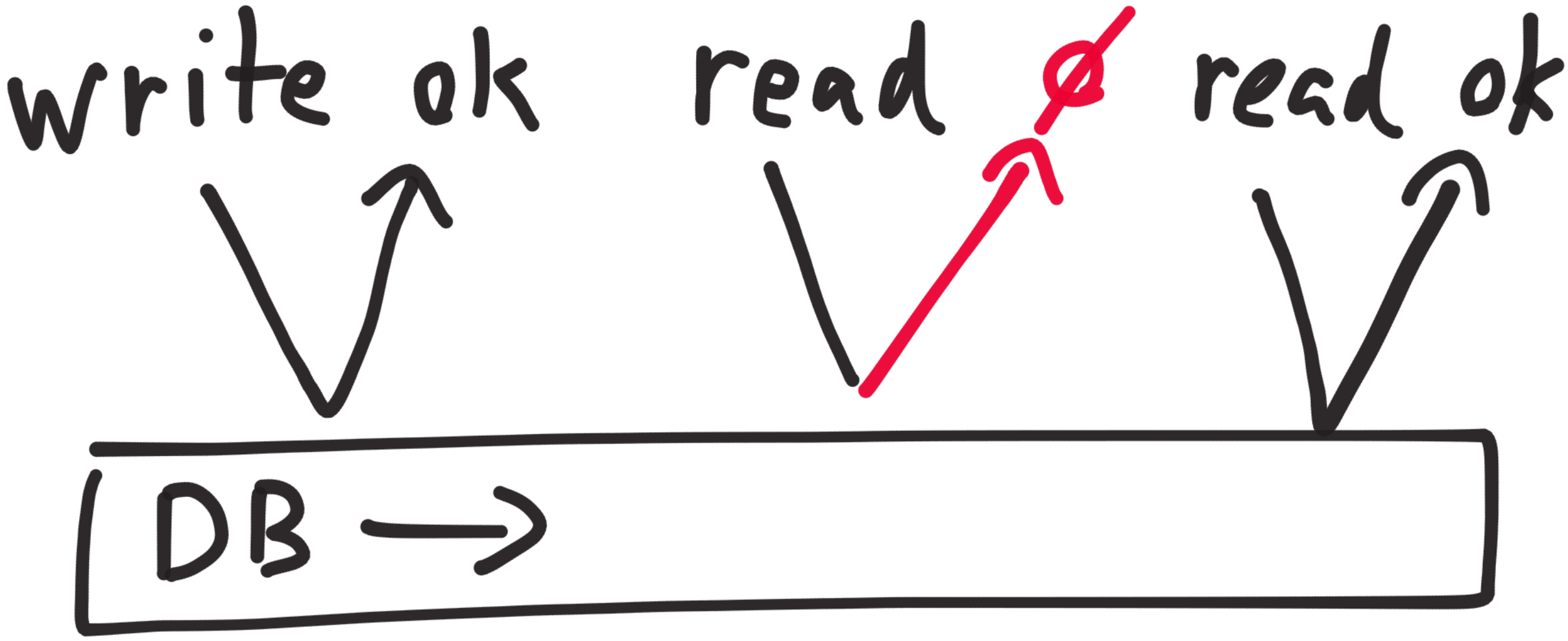
THE HORROR



Split Brain

Broken Foreign Keys



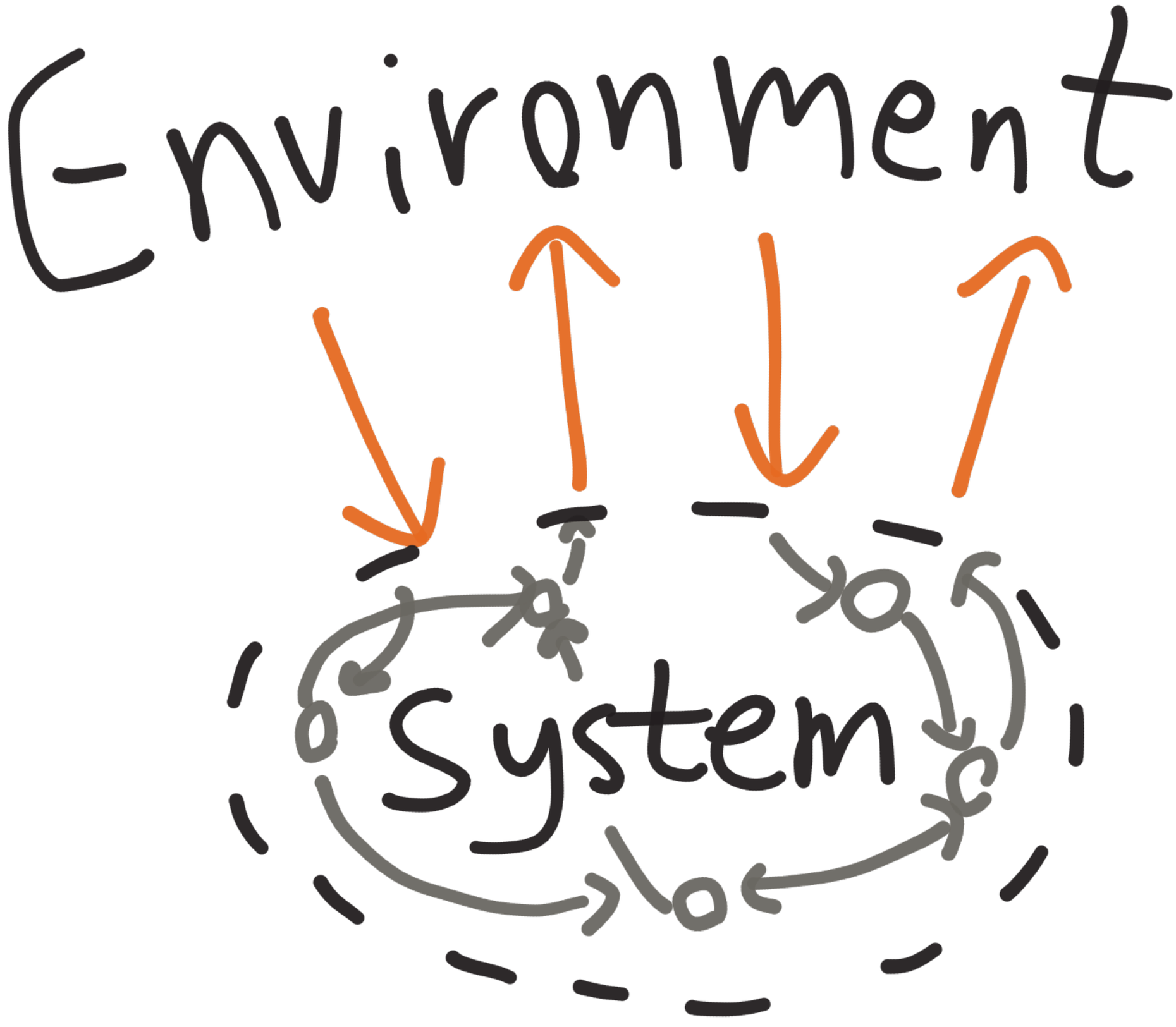


Anomalies

How do you
know if a
system is safe?

Jepsen

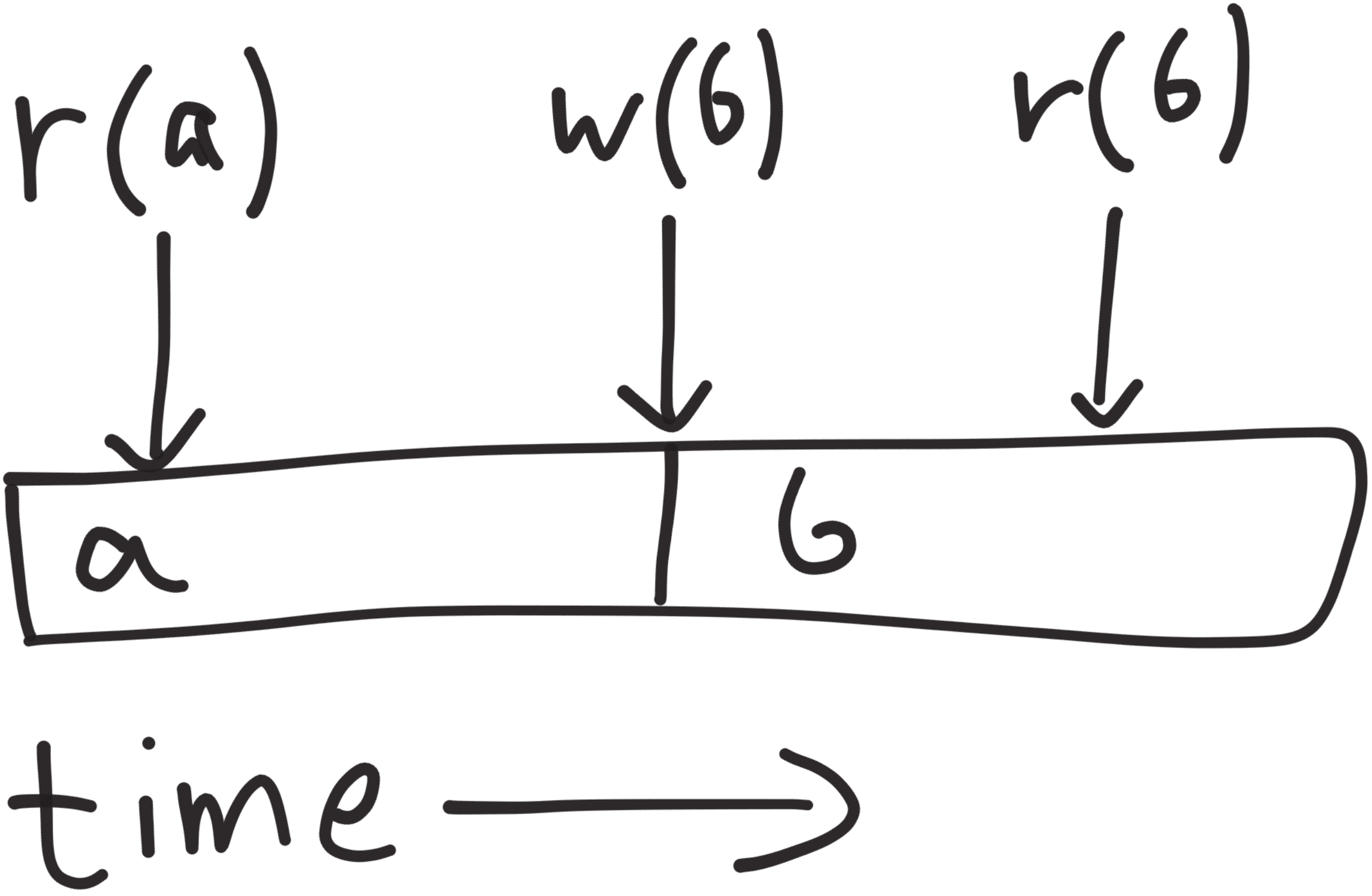
github.com/aphyr/jepsen

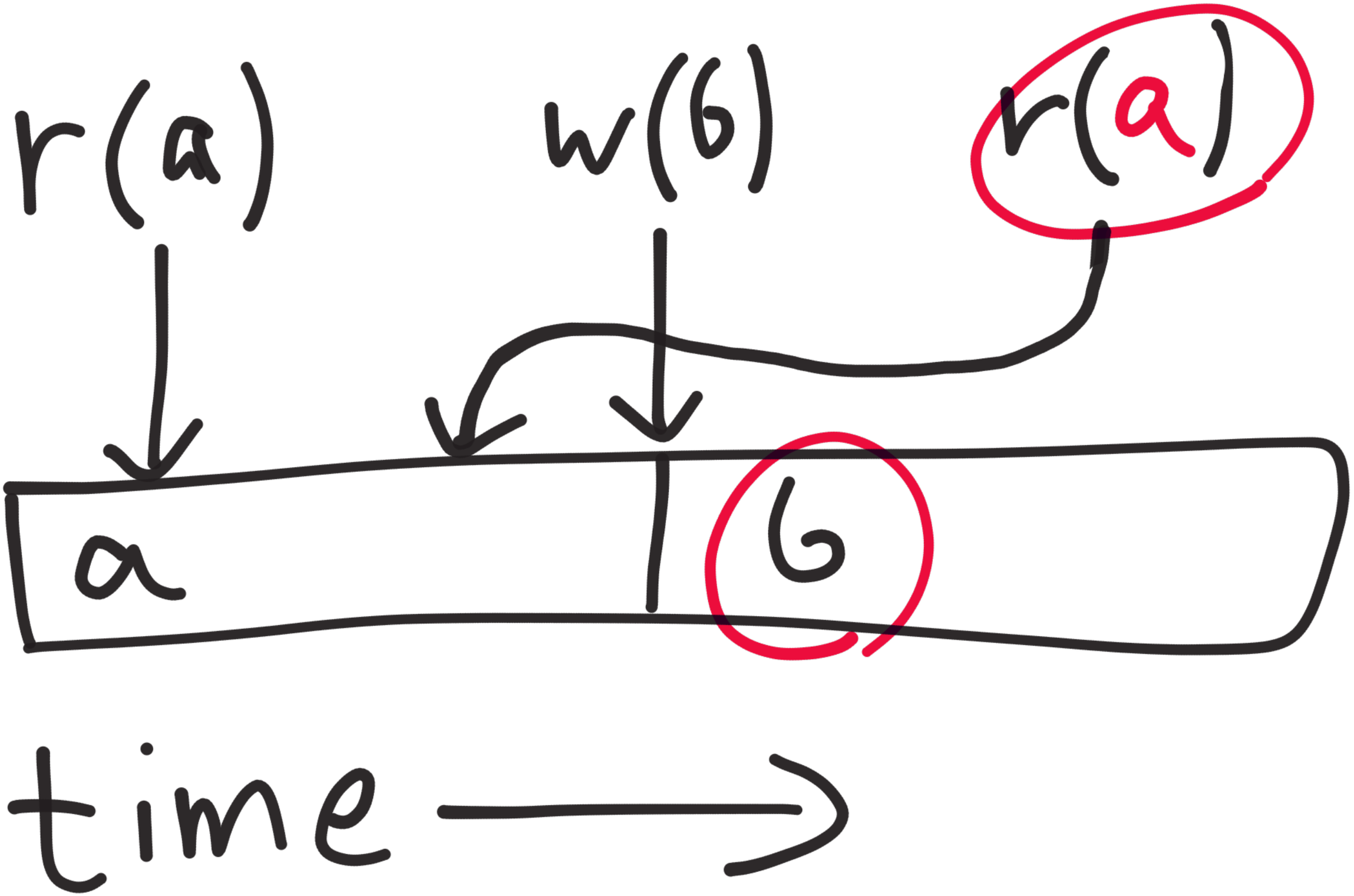

Environment
System

-INVARIANTS-

```
var x = a
print x
```

```
x = b
print x
```


r(a)
w(6)
r(6)
a
6
time


r(a)
w(b)
r(a)
a
b
time

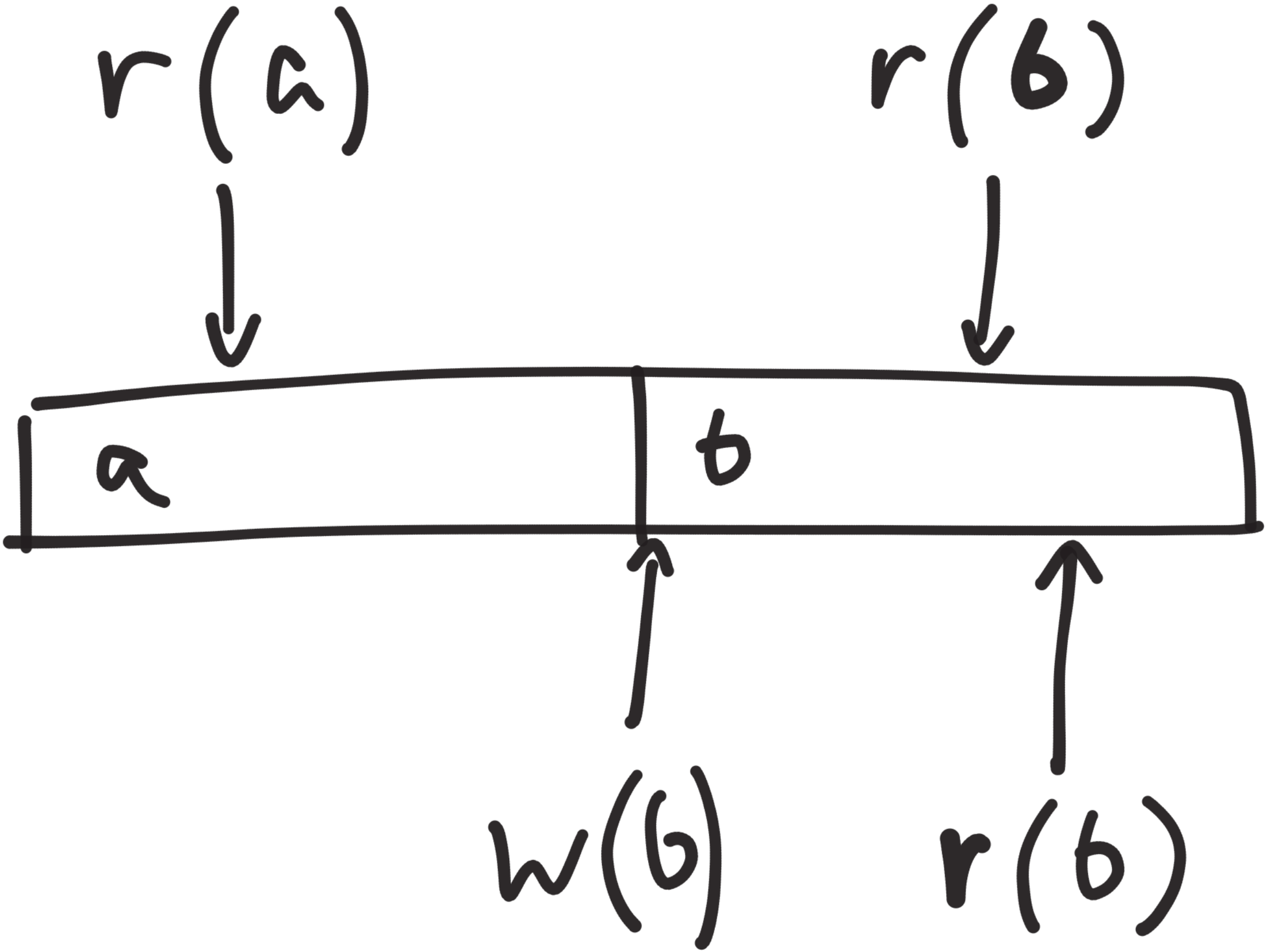
Invariants

Constrain

Histories

What about

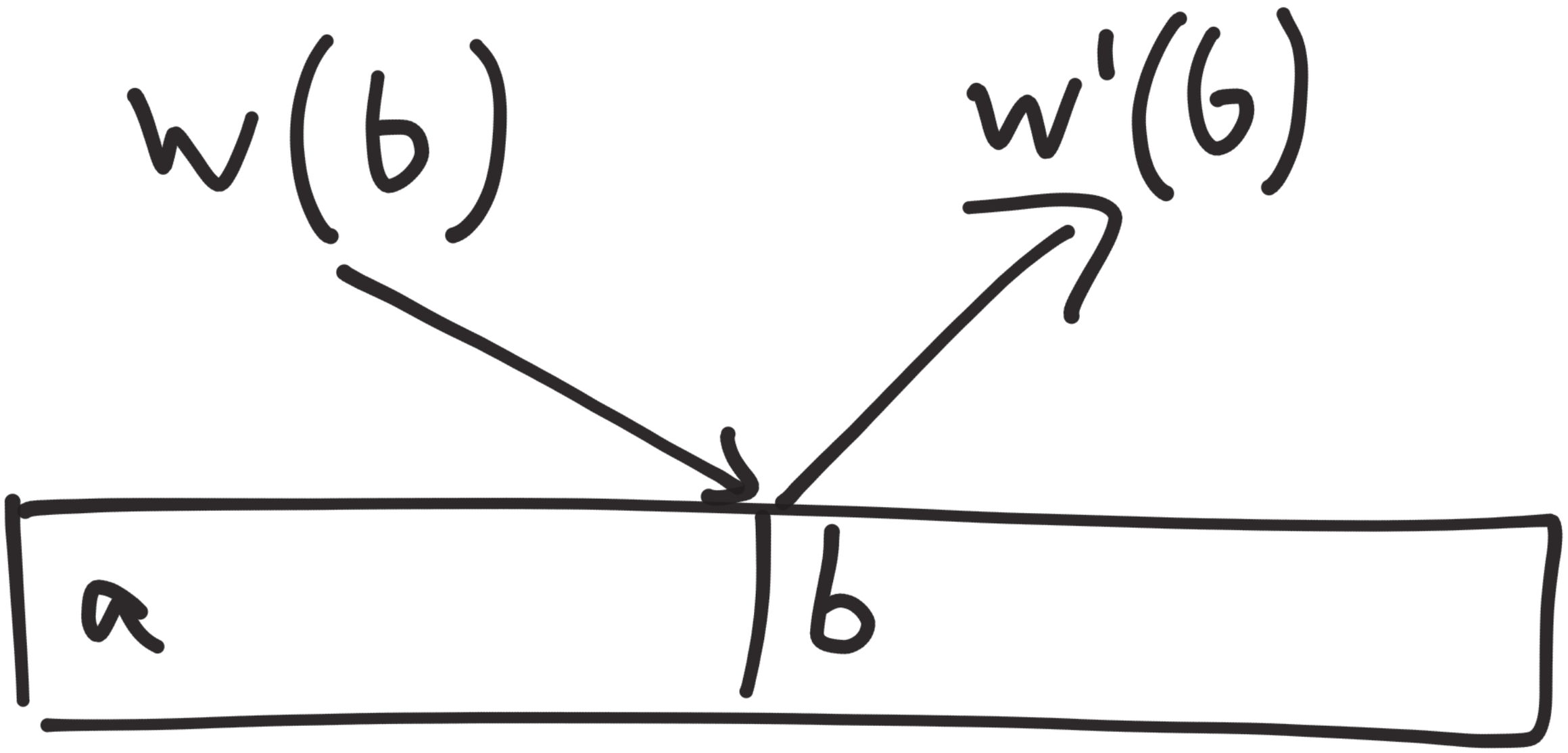
Concurrency?


r(a)
r(b)
a
b
w(b)
r(b)

Distributed

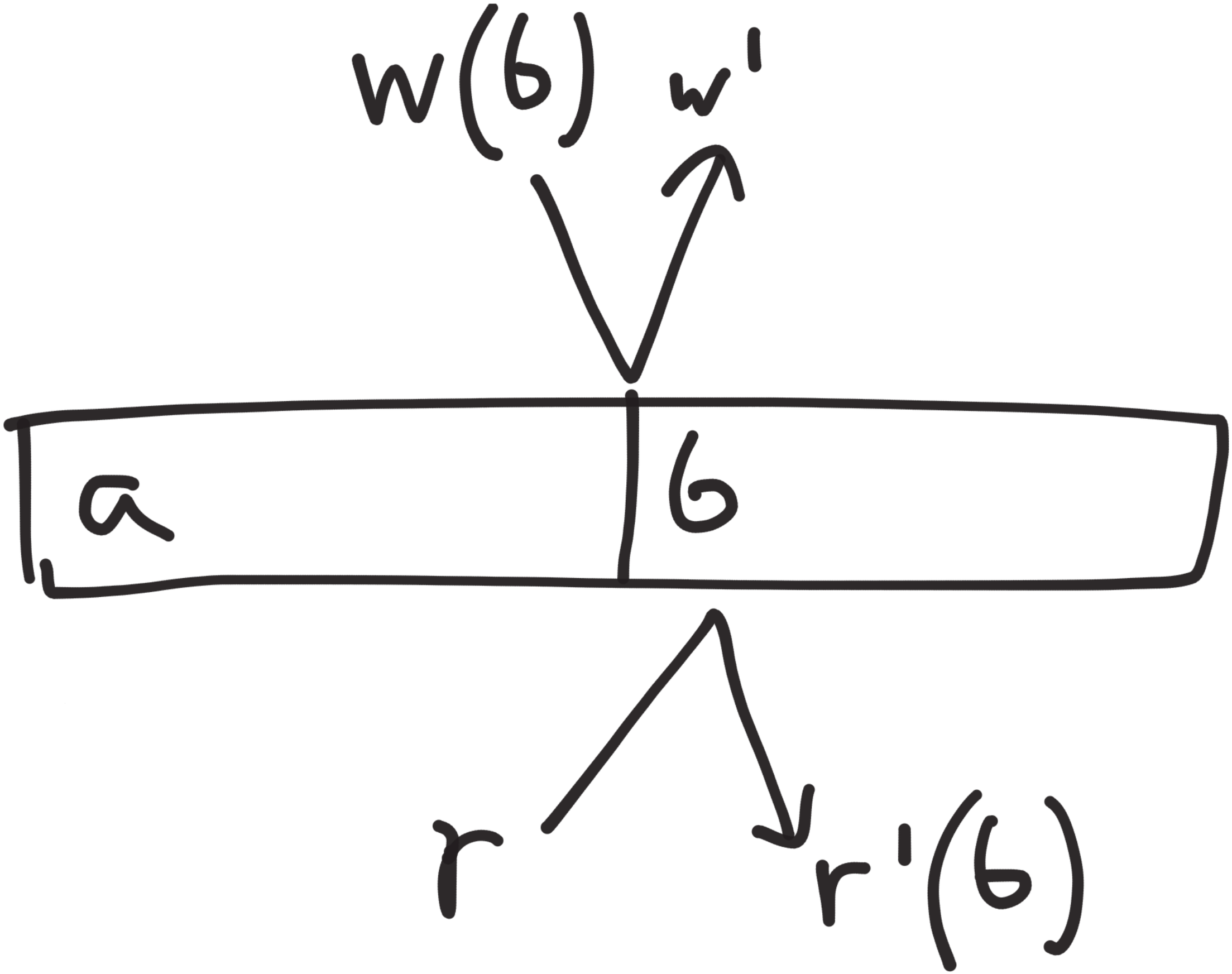
⇓

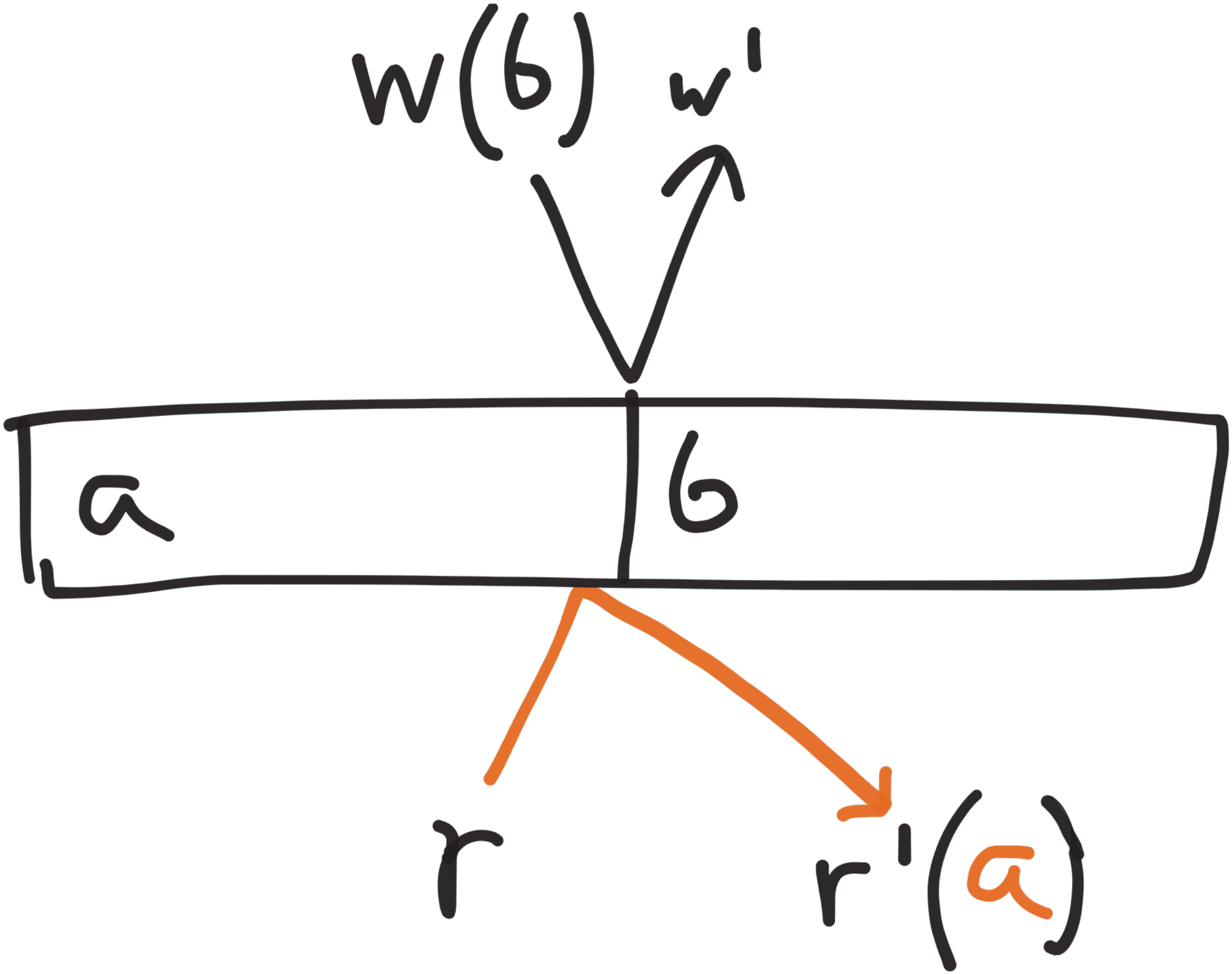
Distance



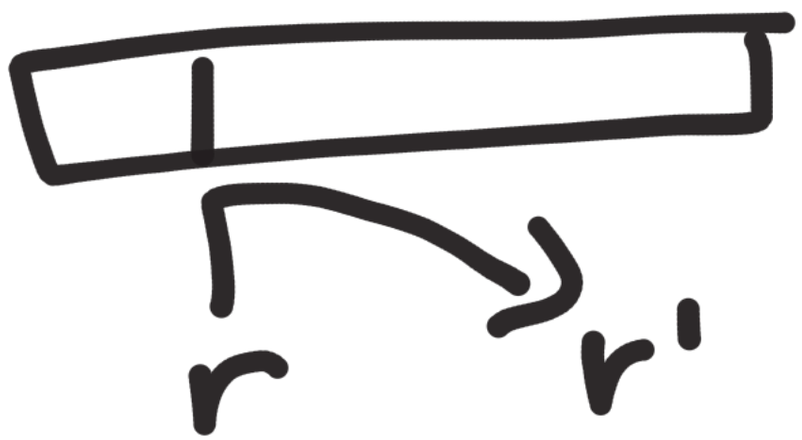
Messages take time

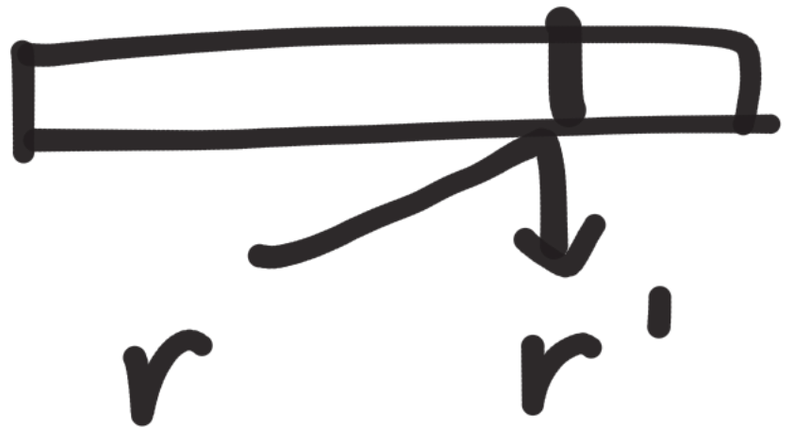
Delays imply ambignous orders

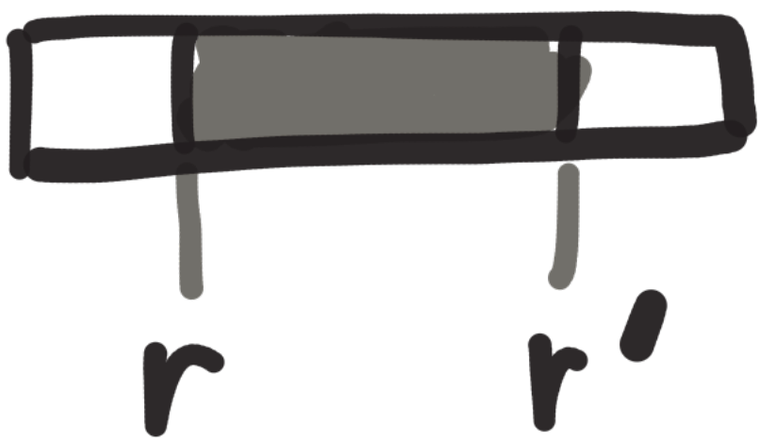

w(b)
w'
a
b
r
r'(b)


w(b)
w'
a
b
r
r'(a)

But there are finite bounds on ambiguity!

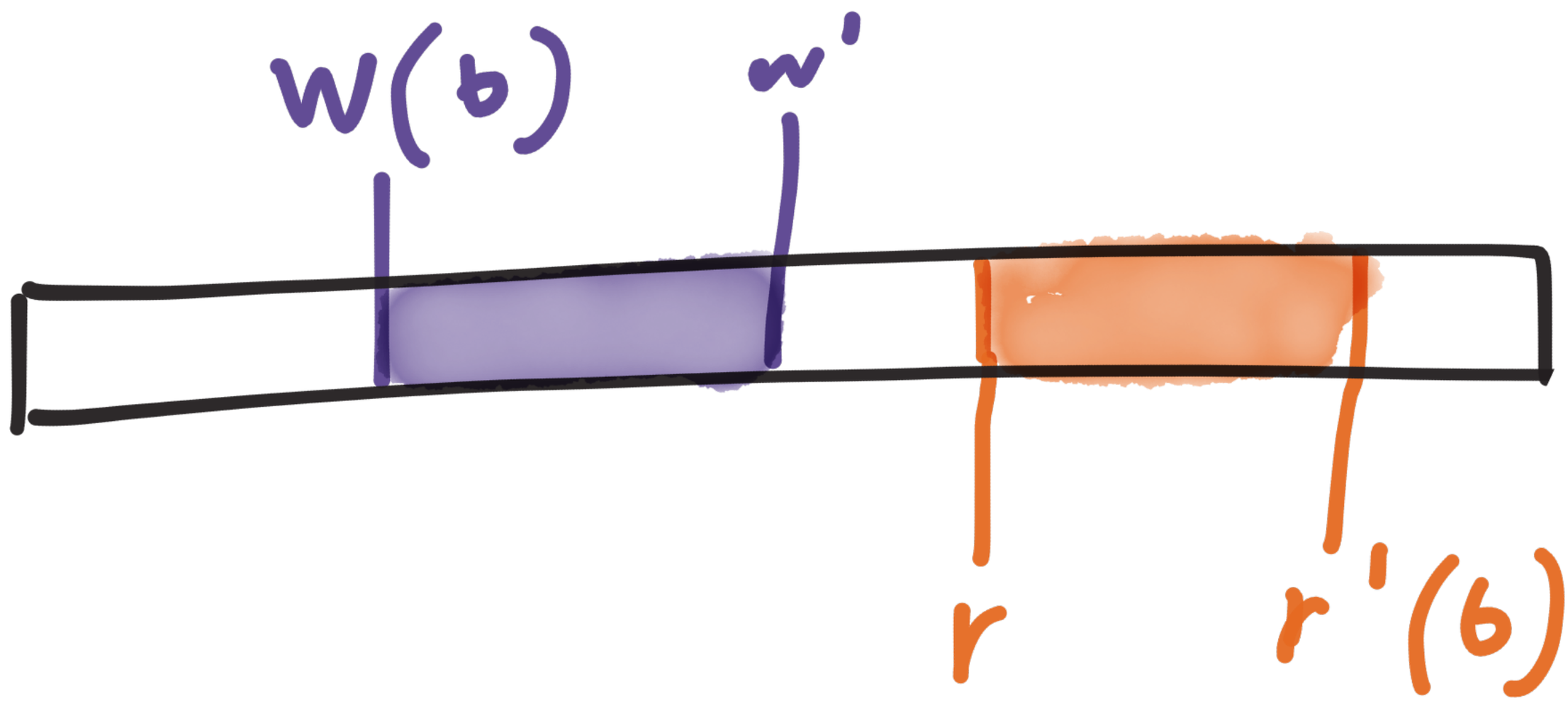






Linearizability

After op is complete, guaranteed visibility!



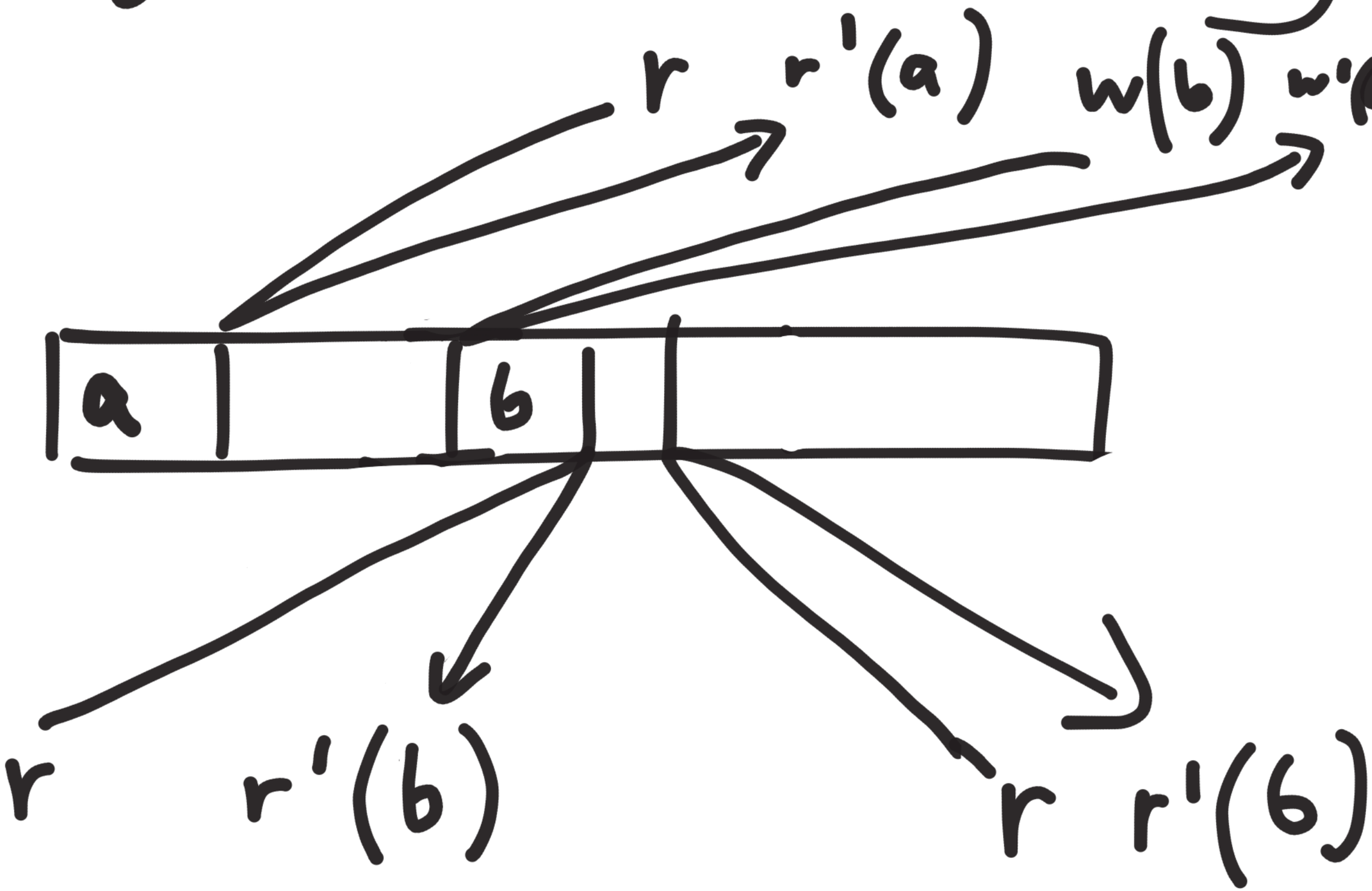
- Everyone agrees on order
- Ops are guaranteed to be visible to ALL clients once complete
- No stale reads

What if...

we relaxed the

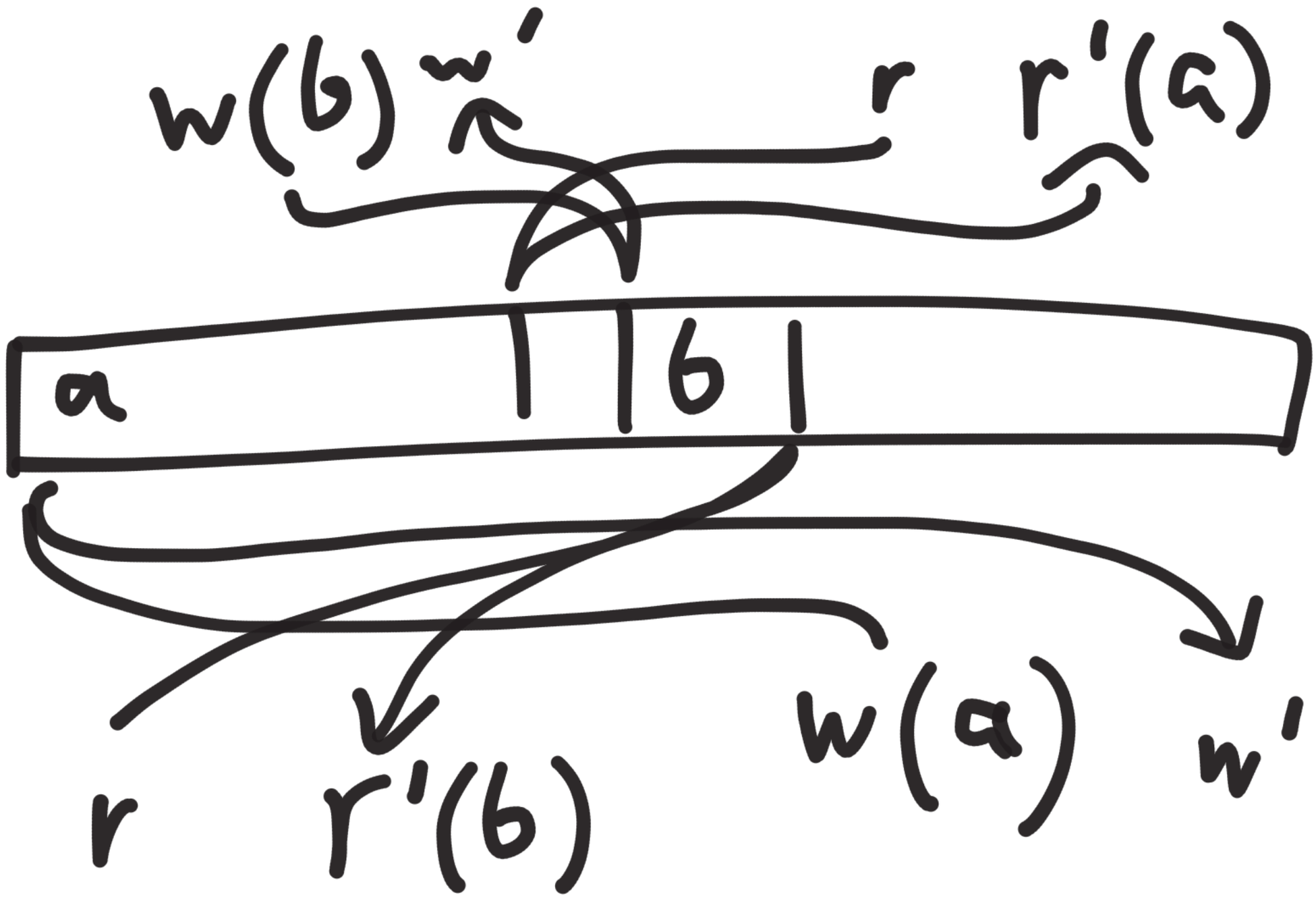
invariants?

Sequential Consistency



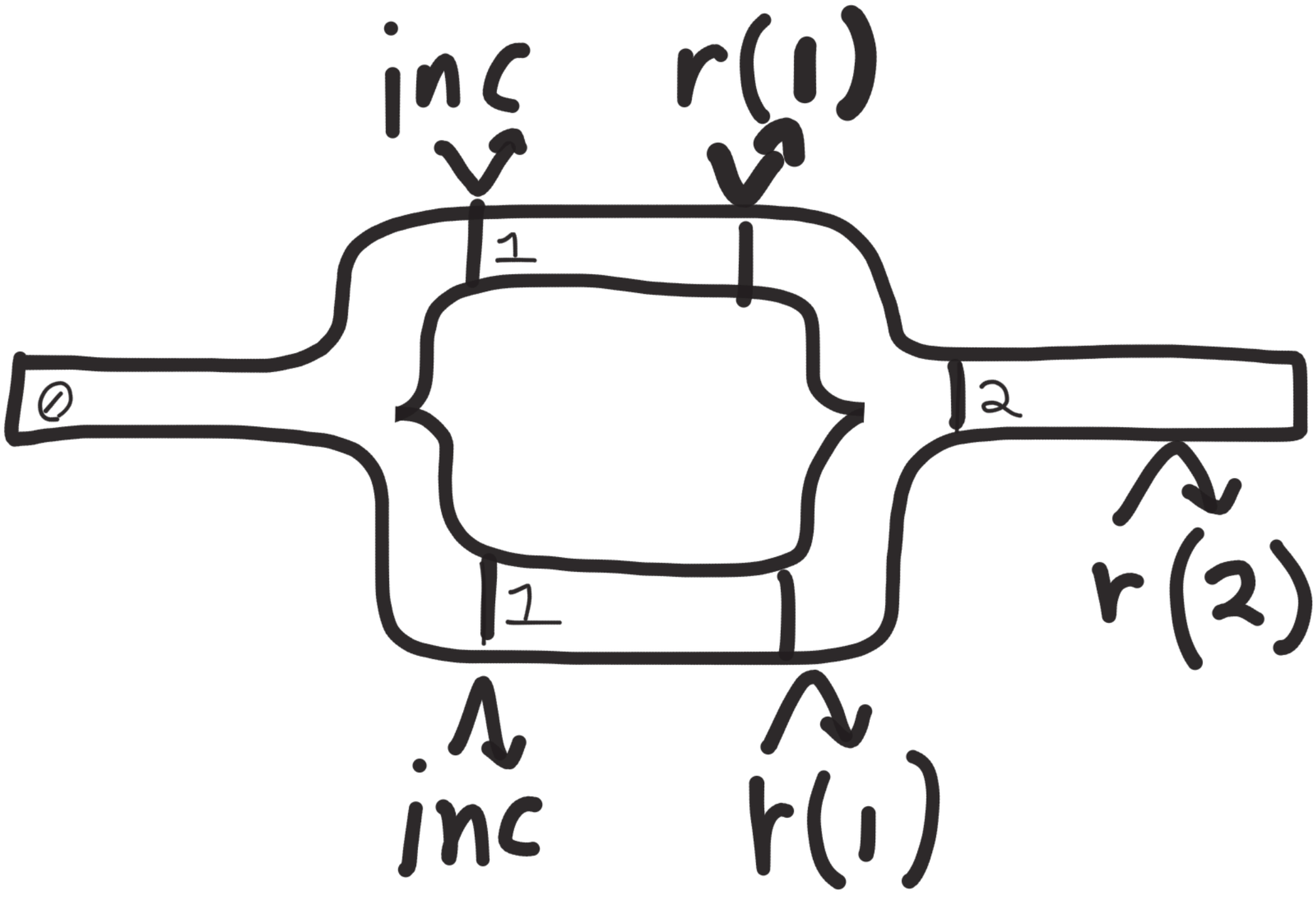
All processes agree on order of ops, but may be delayed in time.

Serializable Consistency



Ops happen in SOME order, but nobody has to agree on what that was.

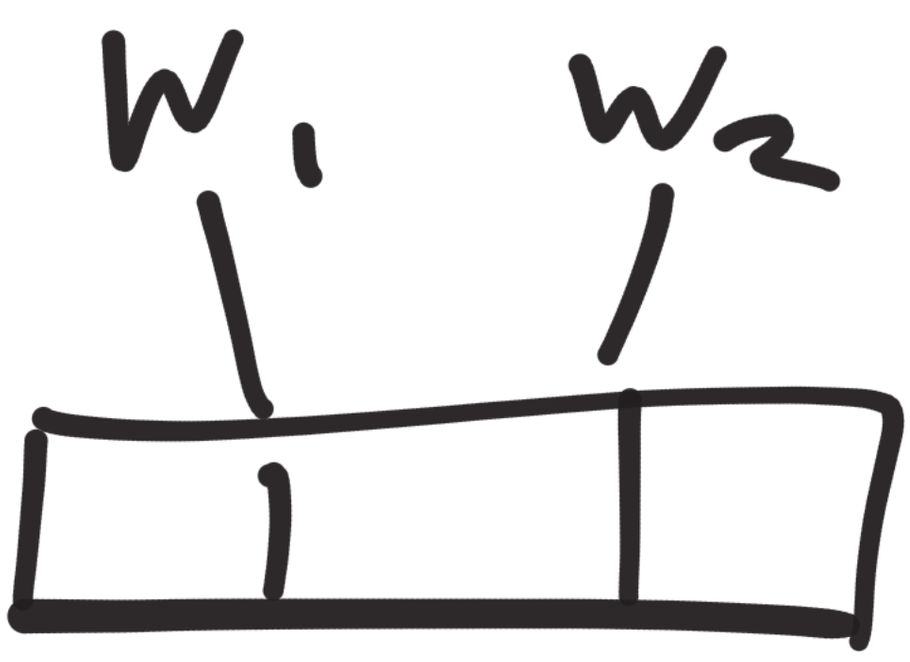
Eventual Consistency

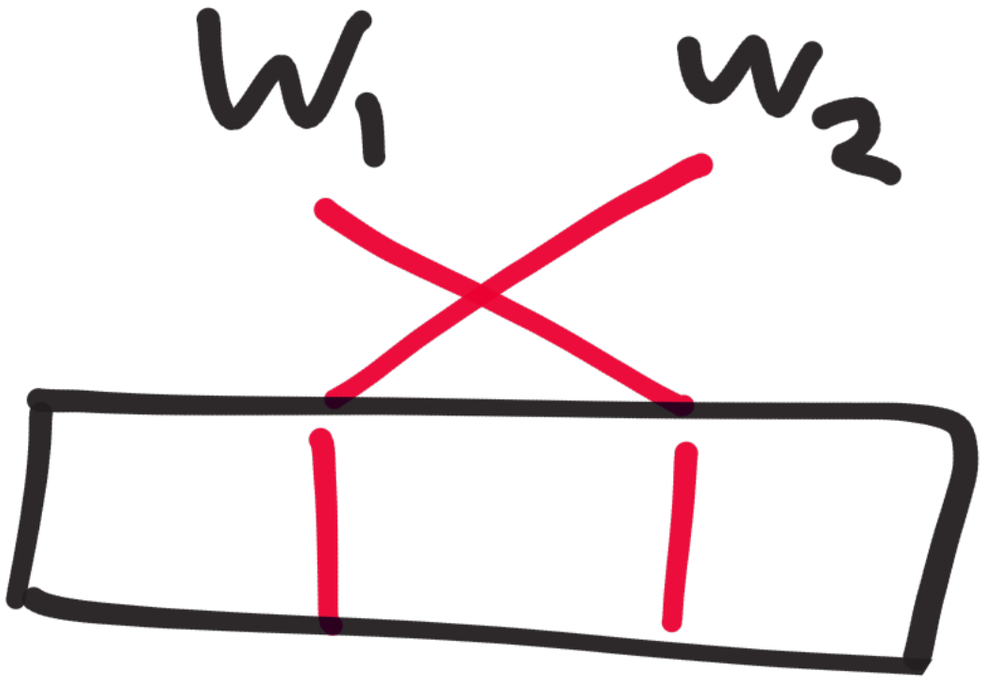


There doesnt have to be any single order so long as histories converge

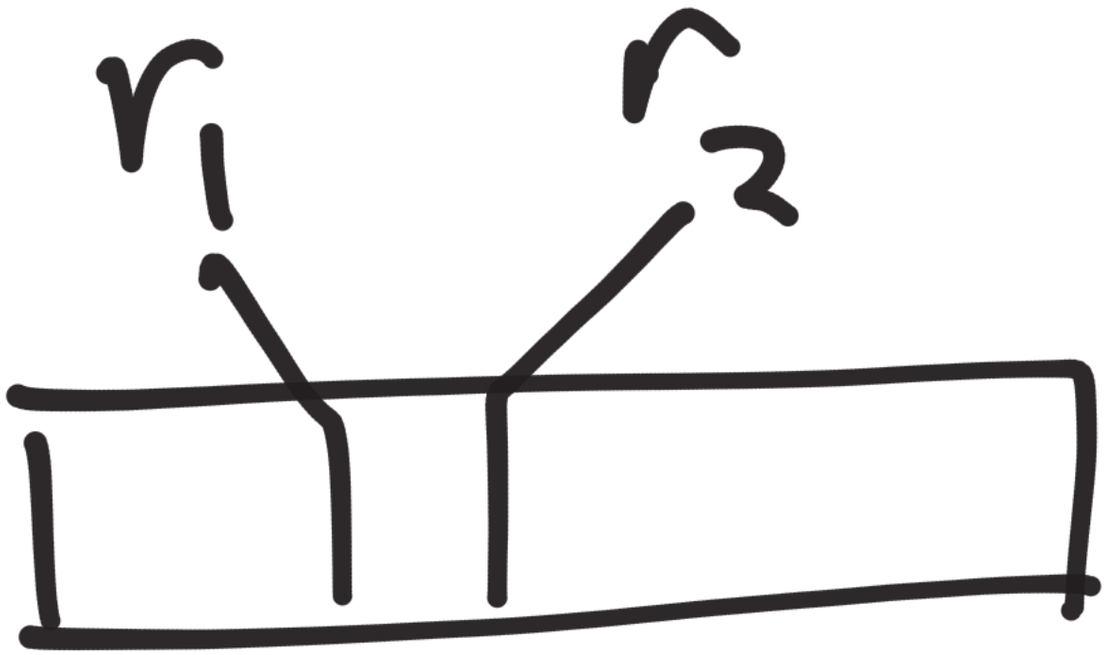
second op: write	first op: write	first op: read
read write	MW	WFR
read	RYW	MR

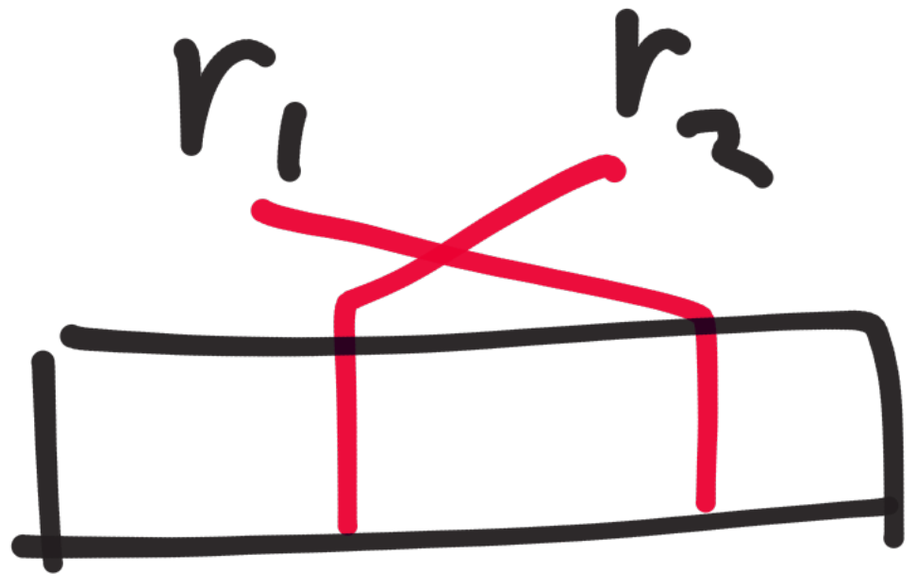
MW: Monotanic Write



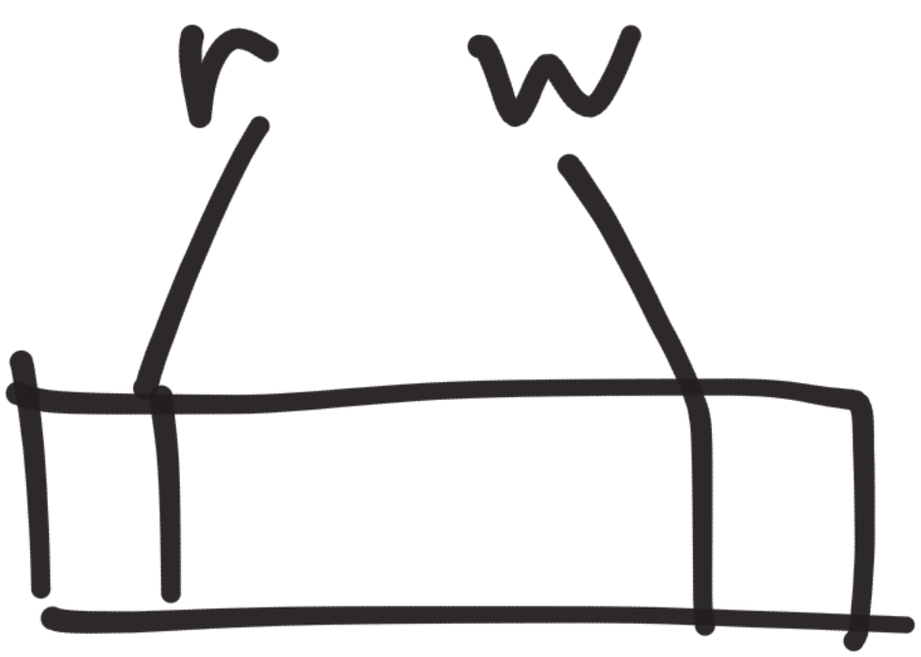


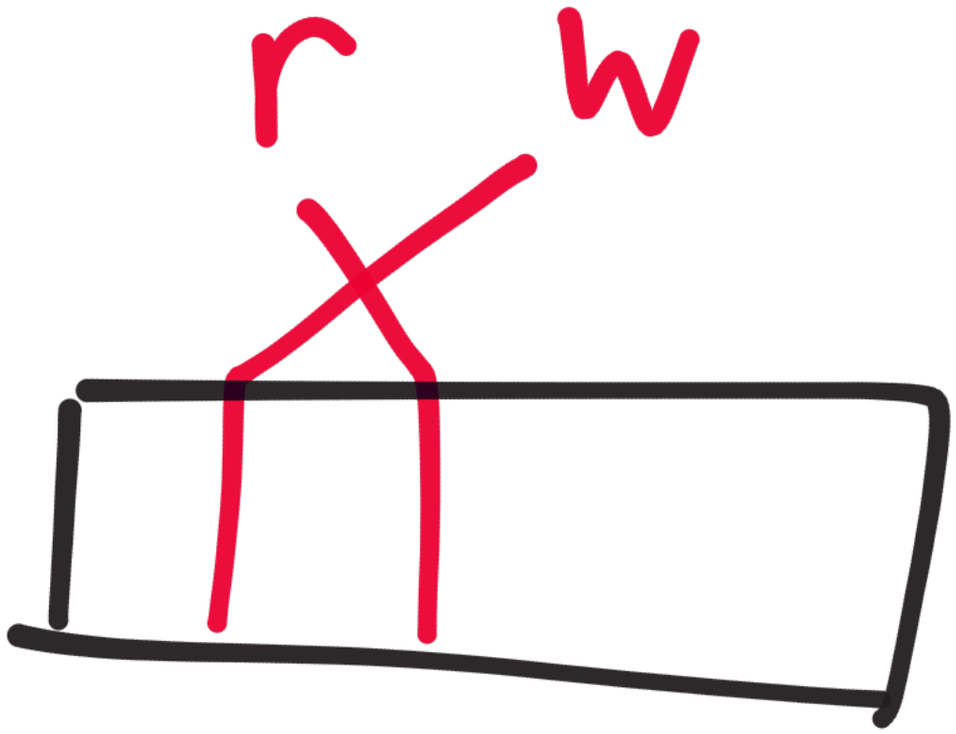
MR: Monotonic Read



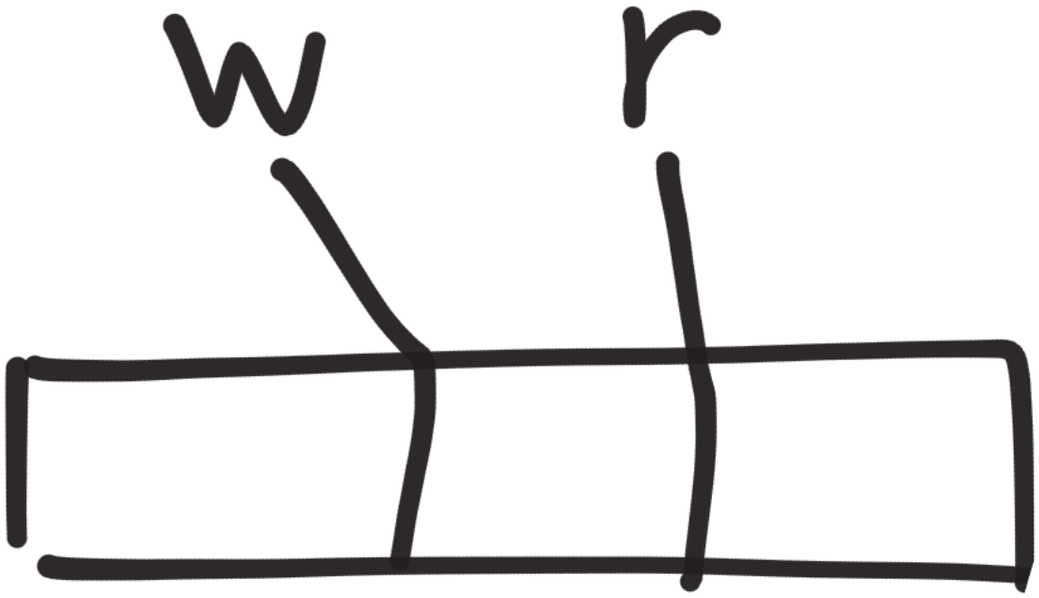


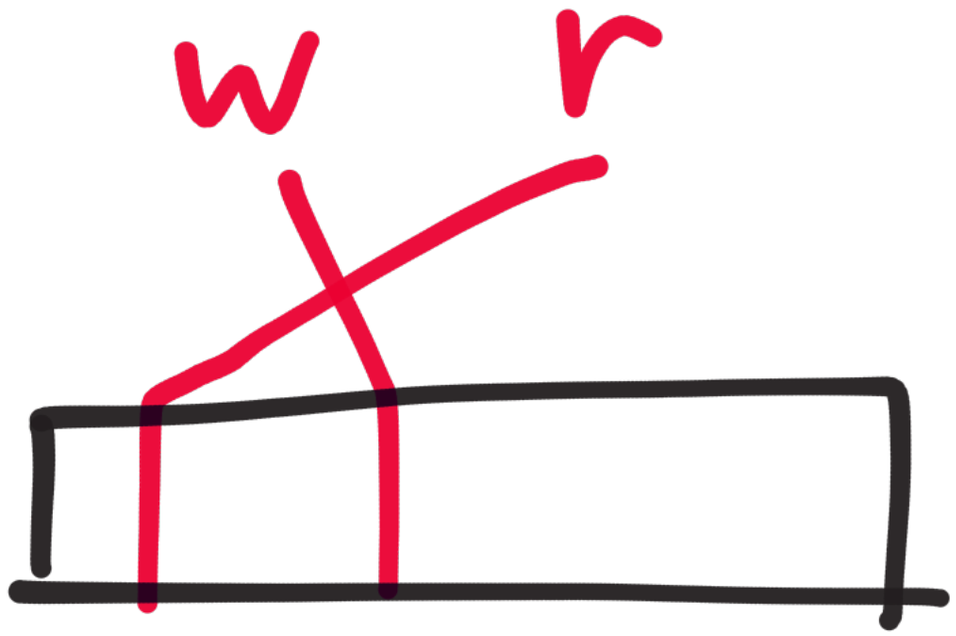
WFR: Writes Follow Reads





RYW: Read Your Writes

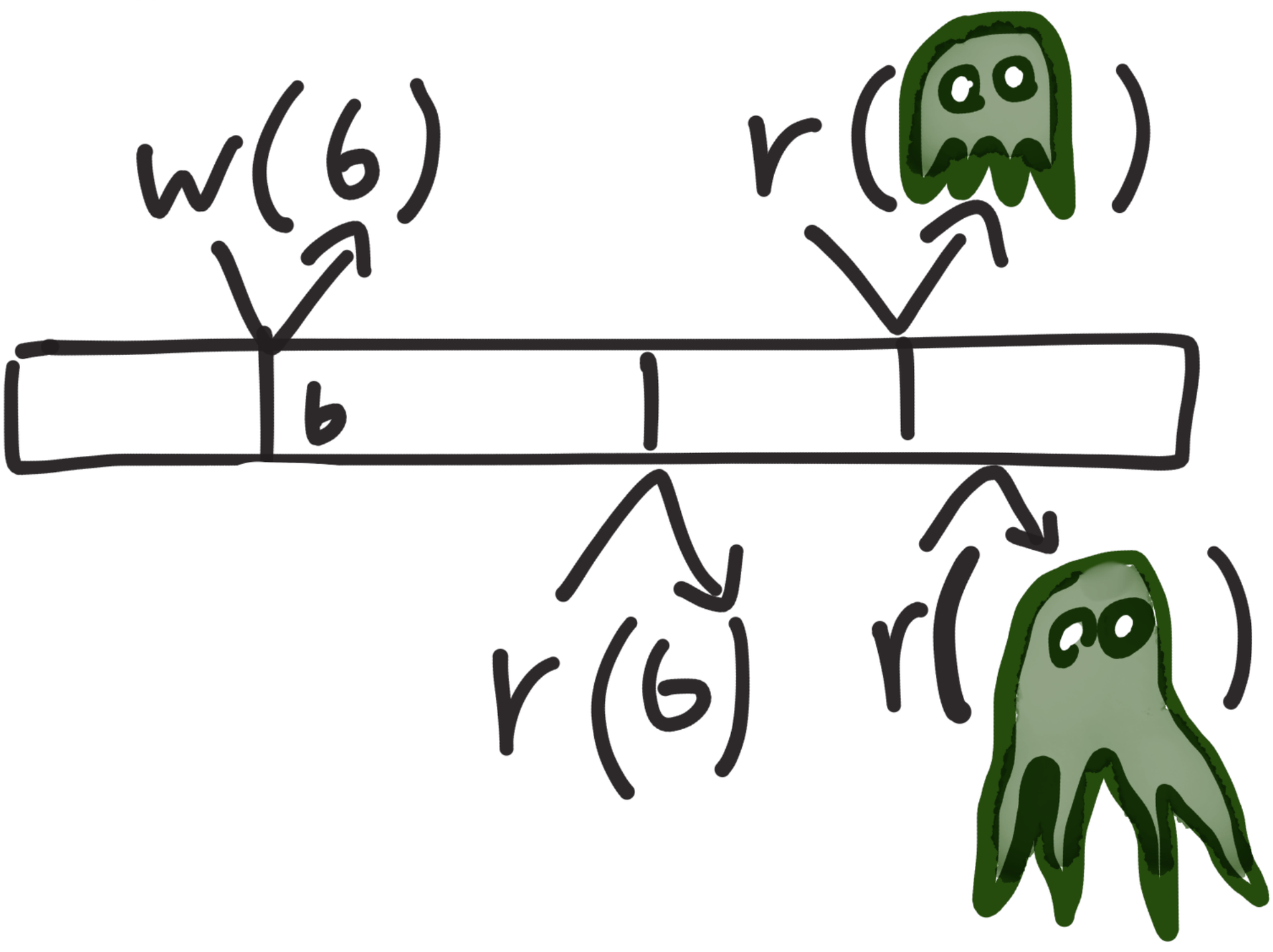




Infinitely many

Consistency classes

Cthuln Consistency



strong

linearizable

sequential

causal

PRAM

RYW

MR

MW

WFR

serializable

serializable

RR

CS

SI

MAV

P-CI

RC

...

"weaker" consistency models are more available in failure

"weaker" models are less intuitive

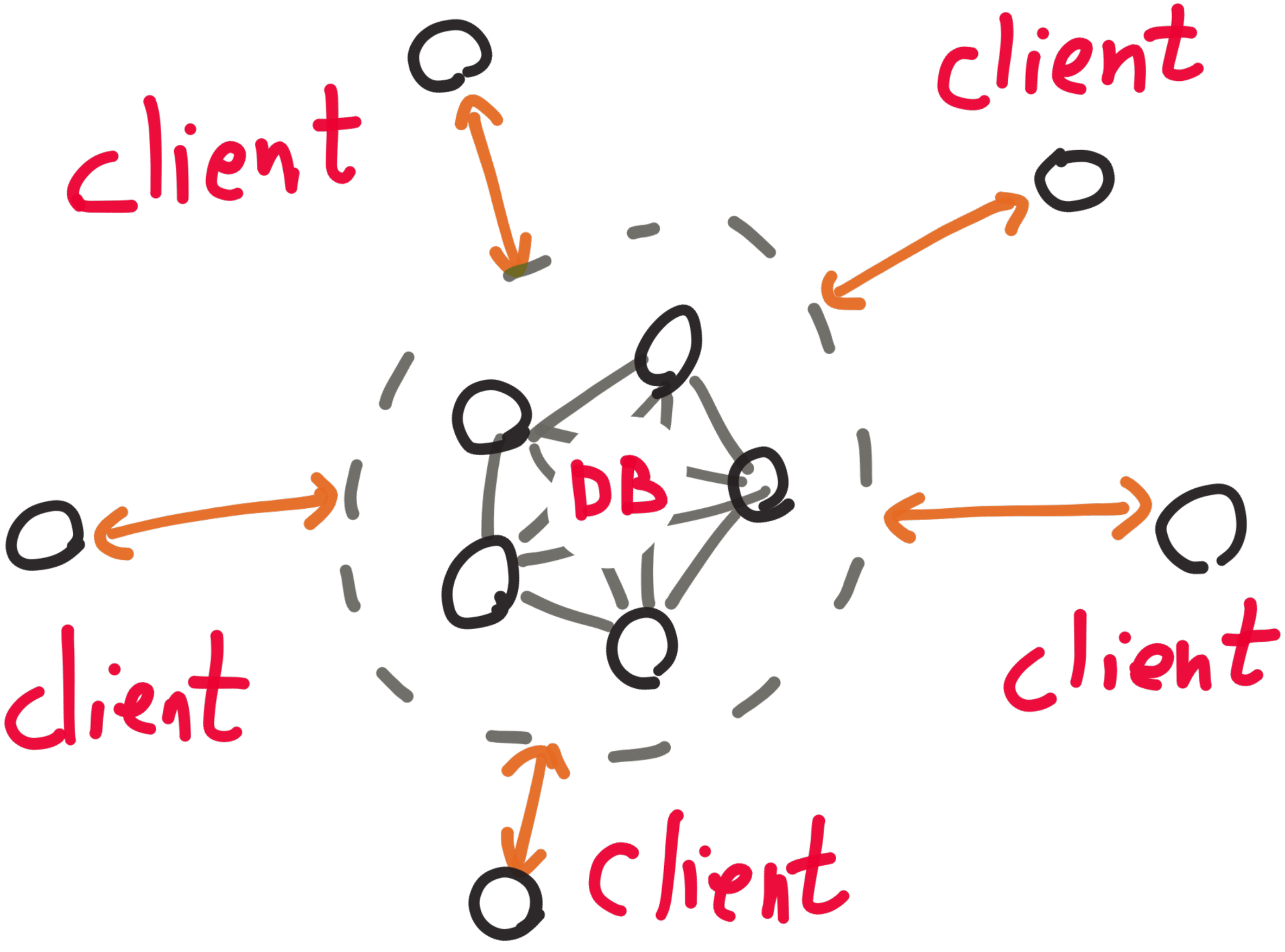
"weaker" models need less coordination, so they're faster!

→ CPU memory model

"weak" ≠ unsafe

- Diff algos require diff. levels of consistency

SO: Consistency models constrain the system-environment interactions.

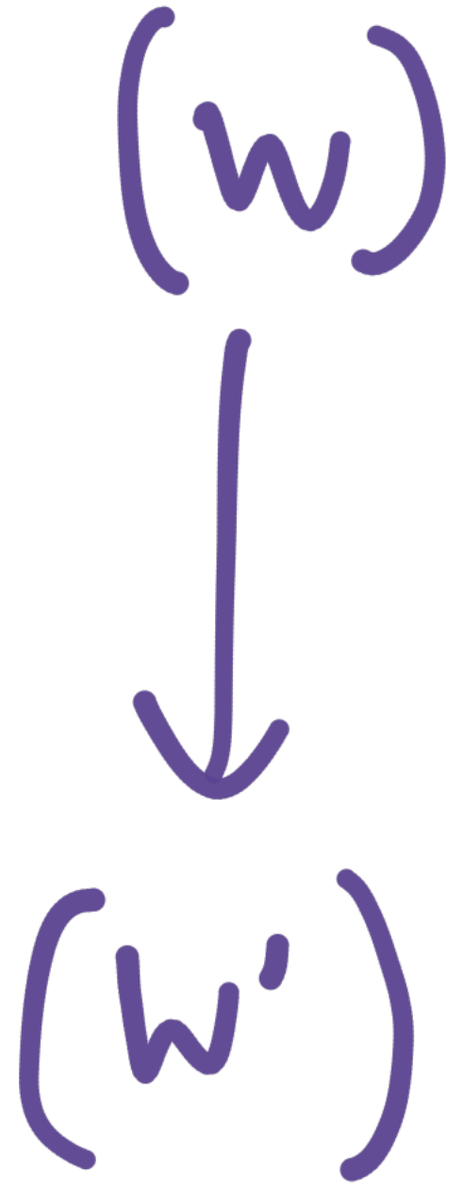

client
client
DB
client
client
client

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

invoke ? ? info ? ? ?

invoke ok

invoke fail

invoke ok

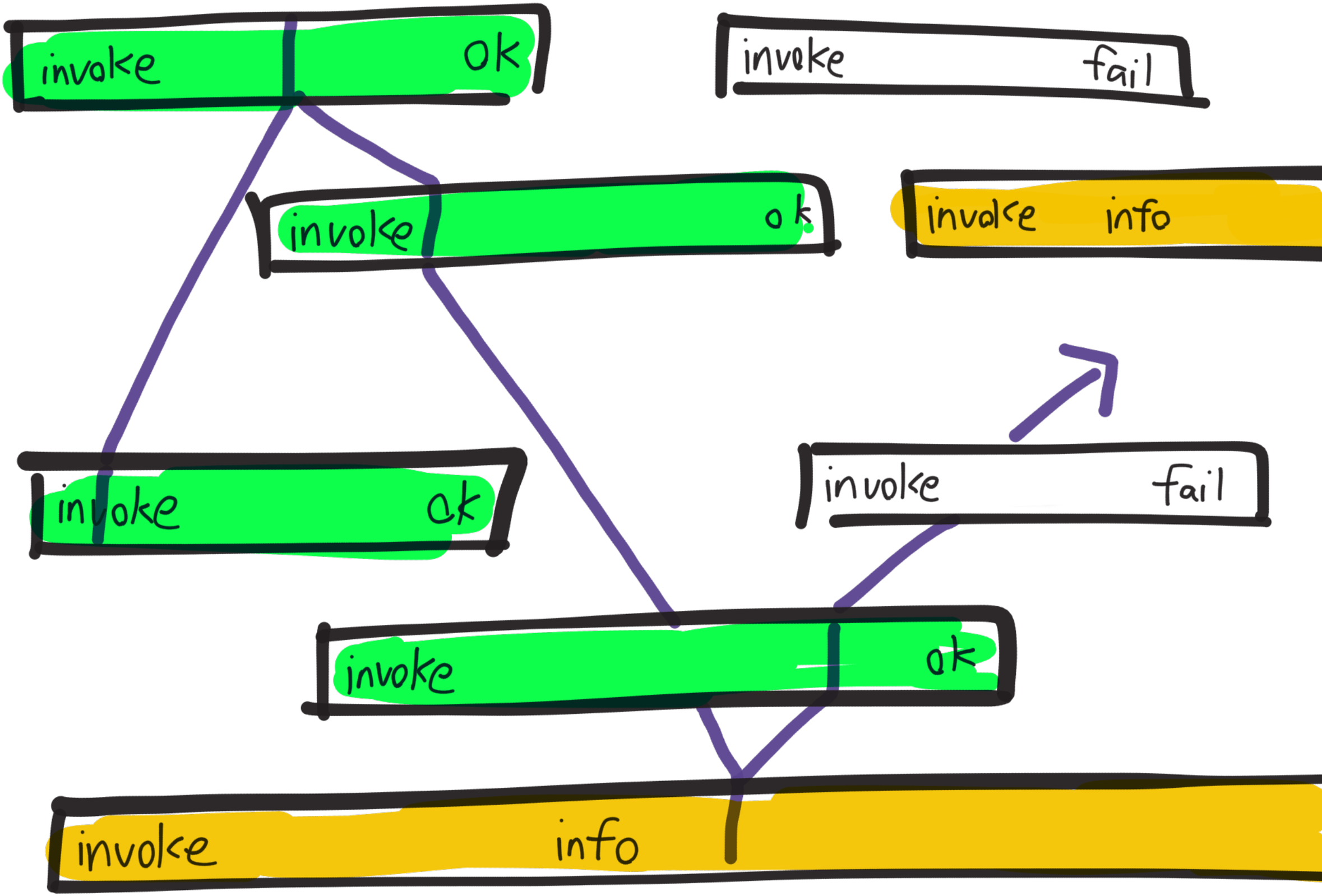
invoke info

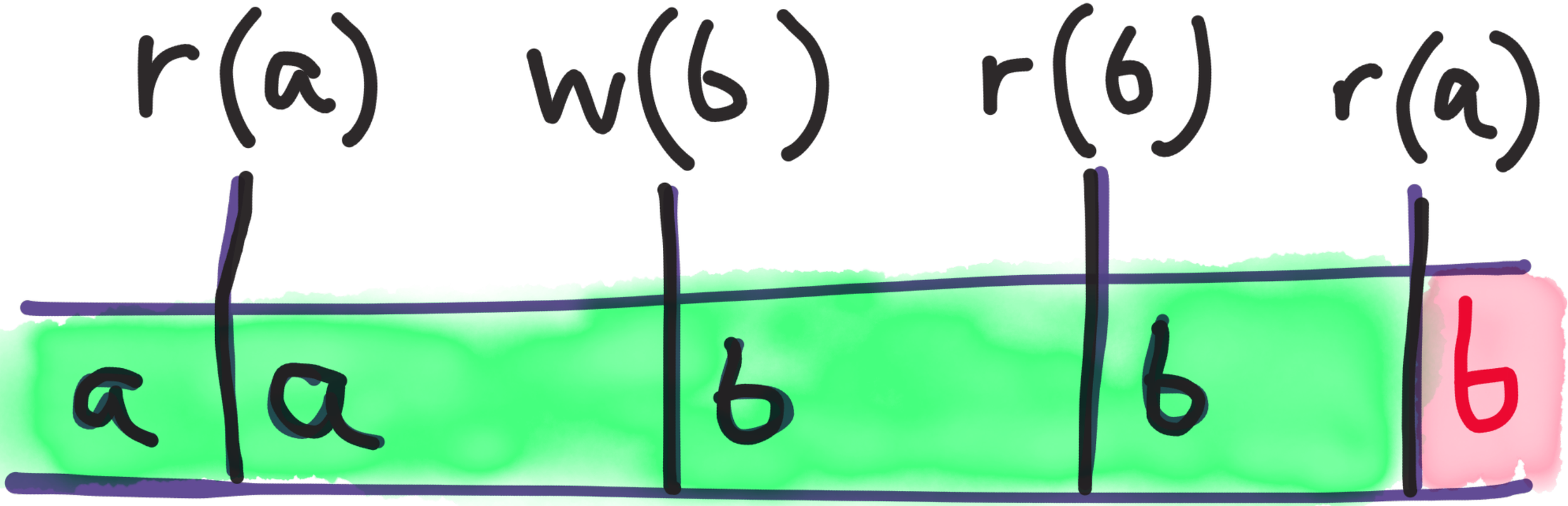
invoke ok

invoke fail

invoke ok

invoke info


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



Use a singlethreaded model to verify path

invoke ok

invoke fail

invoke ok

invoke info

invoke ok

invoke fail

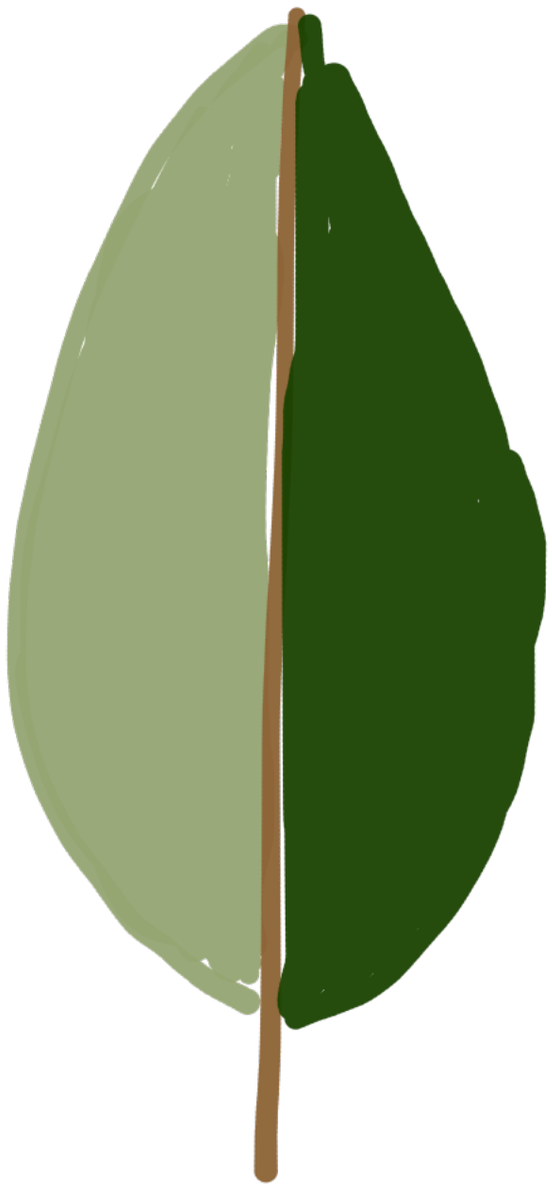
invoke ok

invoke info

invoke info

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

“So, what have you found?”



mongoDB

First Jepsen test in May 2013

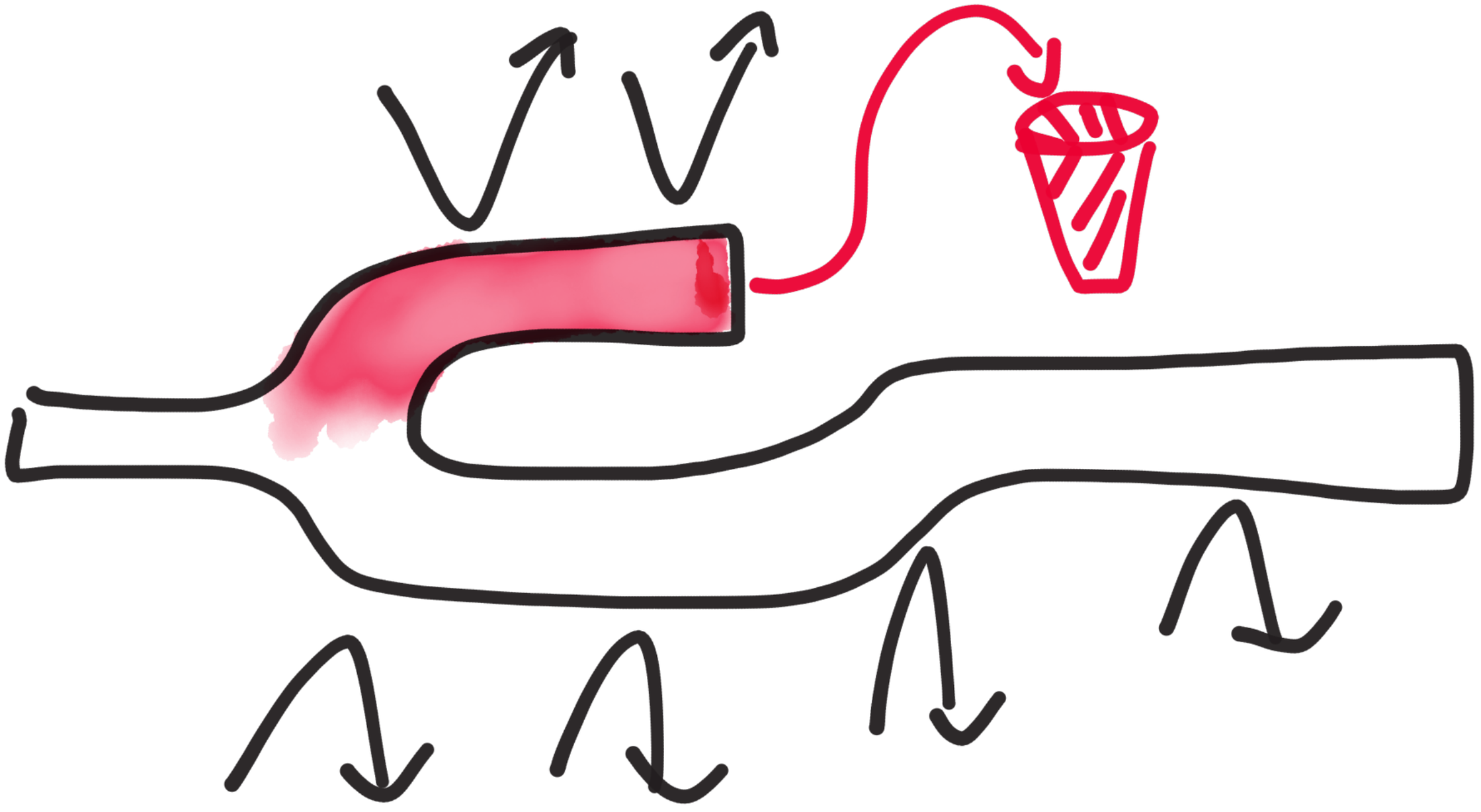
Data loss at all WriteConcern levels

Unsafe defaults

- Some clients didnt check for errors at all

Unsafe defaults

- Better clients only wait for 1 or nodes, instead of a majority!



Rollbacks

Even Majority unsafe!

Network failures were considered successful responses

But those bugs are fixed now!

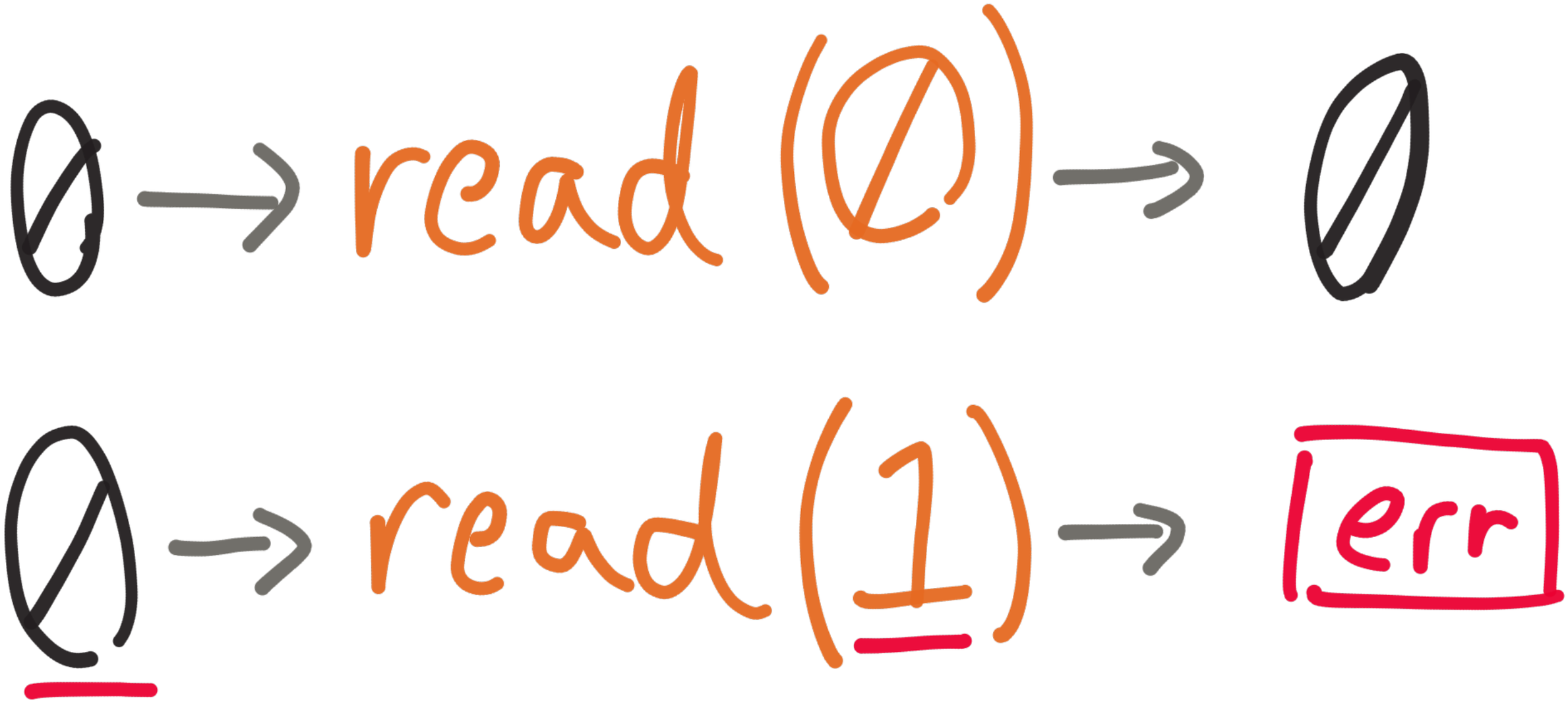
Ok! Let's

try 2.6.7

Linearizable

CaS Registers


∅ → write(1) → 1


∅ → read(∅) → ∅
∅ → read(1) → err

$$\emptyset \to cas(\emptyset, 1) \to 1$$

$$\underline{\emptyset} \to cas(\underline{1}, 2) \to \boxed{err}$$

An Anomaly Emerges!

0

cas 0→4

1

r4

r3

cas 3→1

2

r0

Write 4

3

cas 4→2

4

write 3

5

read 0

0

cas 0→4

1

r4

r3

cas 3→1

2

r0

Write 4

3

cas 4→2

4

write 3

5

read 0

0

cas 0→4

1

r4

r3

cas 3→1

2

r0

Write 4

3

Cas 4→2

4

write 3

5

read 0

0

cas 0→4

1

r4

r3

cas 3→1

2

r0

Write 4

3

cas 4→2

4

write 3

5

read 0

0

cas 0→4

1

r4

r3

cas 3→1

2

r0

Write 4

3

cas 4→2

4

write 3

5

read 0

0

cas 0→4

1

r4

r3

cas 3→1

2

r0

Write 4

3

cas 4→2

4

write 3

5

read 0

0

cas 0→4

1

r4

r3

cas 3→1

2

r0

Write 4

3

Cas 4→2

4

write 3

5

read 0

0

cas 0→4

1

r4

r3

cas 3→1

2

r0

Write 4

3

Cas 4→2

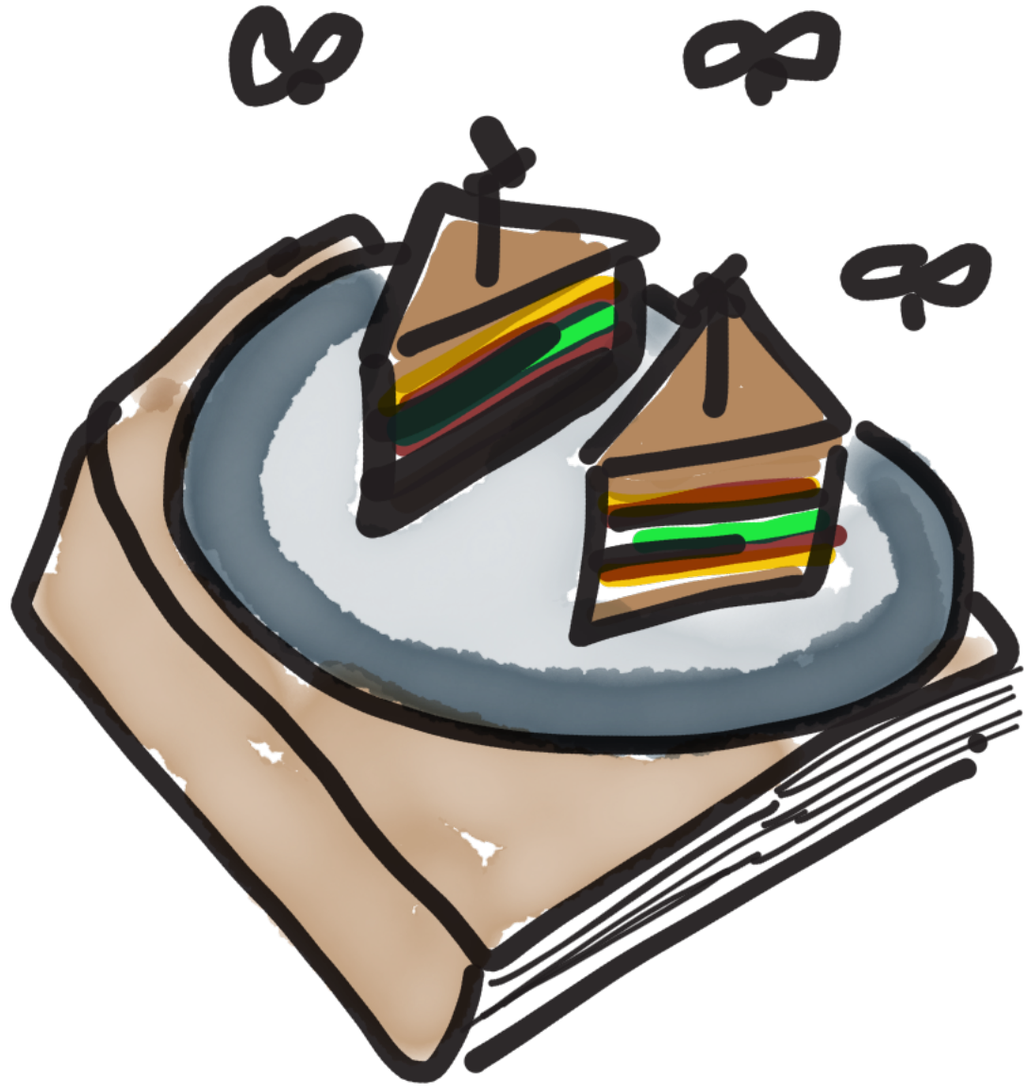
4

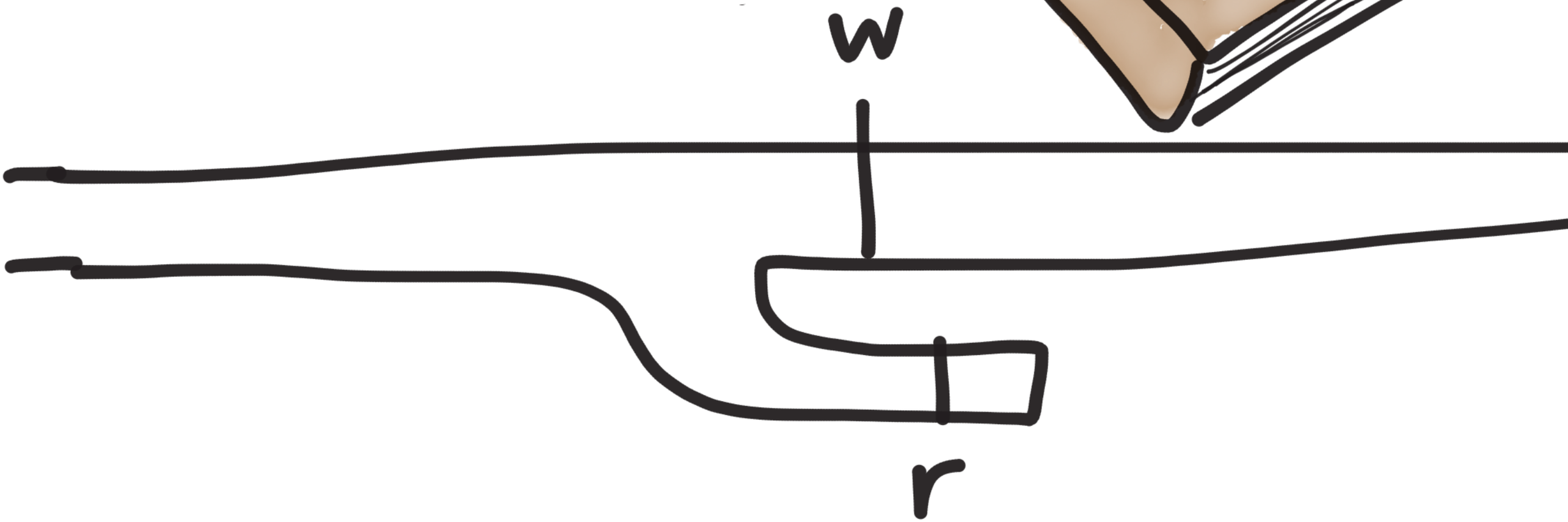
write 3

5

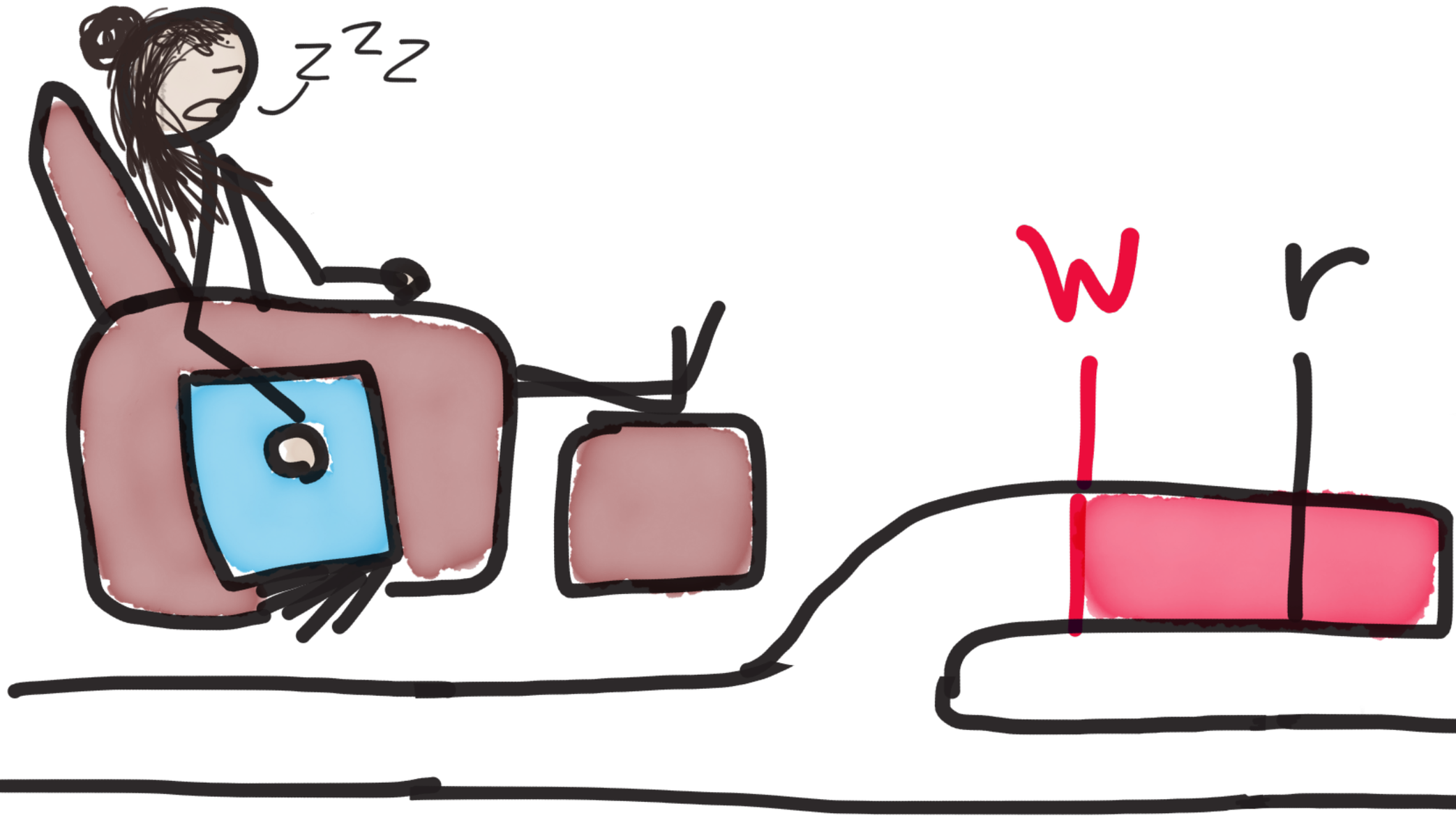
read 0

Stale Reads

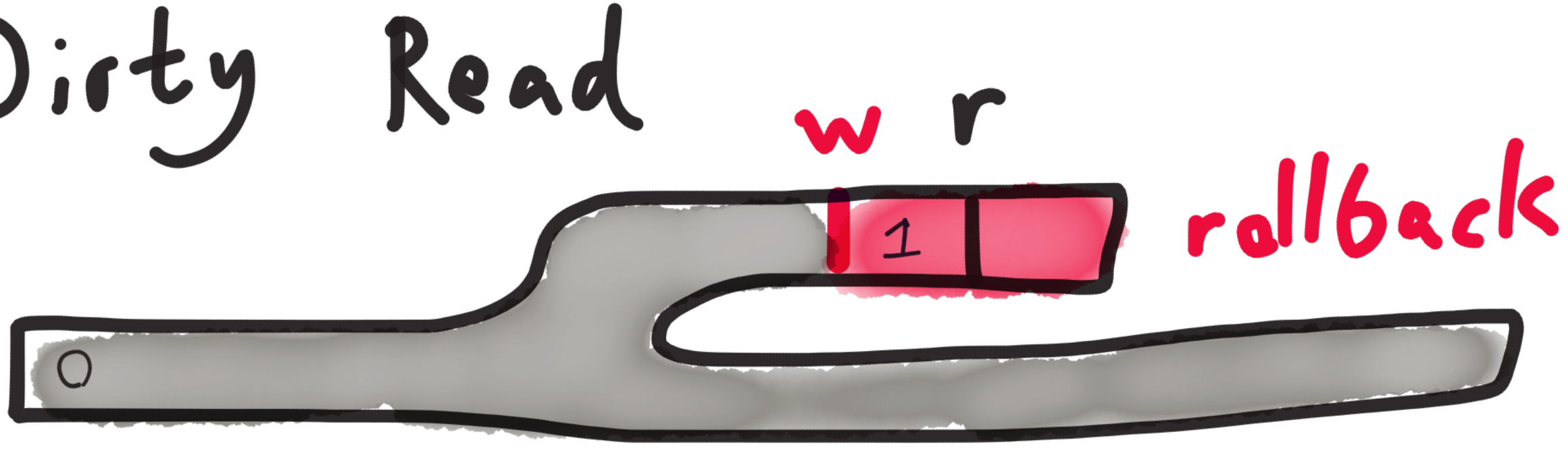




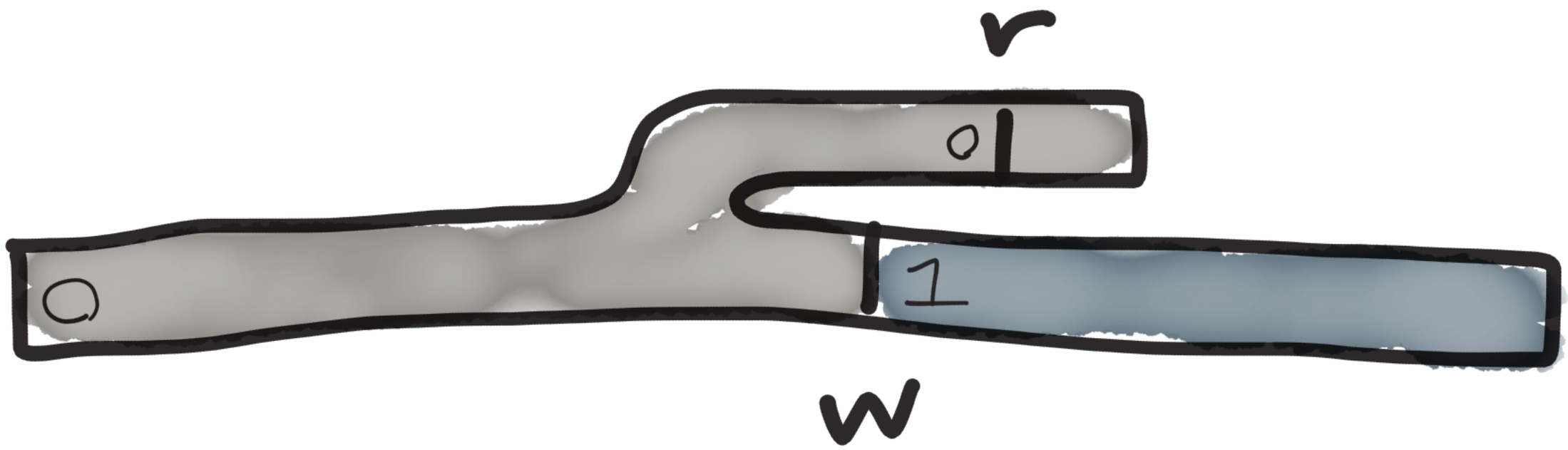
Read Uncommitted



Dirty Read



Stale Read



Server-17975

strong

linearizable

sequential

causal

PRAM

~~R~~ ~~W~~

~~M~~ ~~R~~

MW

WFR

serializable

serializable

RR

CS

SI

MAV

P-CI

RC

strong

~~serializable~~

linearizable

serializable

~~sequential~~

RR

SI

CS

~~causa~~

MAV

P-CI

PRAM

WFR

RC

~~RYW~~

~~MR~~

MW

strong

serializable

linearizable

serializable

sequential

RR

SI

causal

CS

MAV

P-CI

PRAM

WFR

RC

RYW

MR

MW

RU

strong

serializable

linearizable

serializable

sequential

RR

SI

causal

CS

MAV

P-CI

PRAM

WFR

IRC

RYW

MR

MW

RU

Mongo

strong

serializable

linearizable

sequential

causal

PRAM

RYW

MR

MW

WFR

serializable

RR

SI

CS

MAV

P-CI

IRC

RU

Mongo vs. docs

Mongo Recommendations

- Plan for stale reads
- Use CaS to verify read values are legitimate

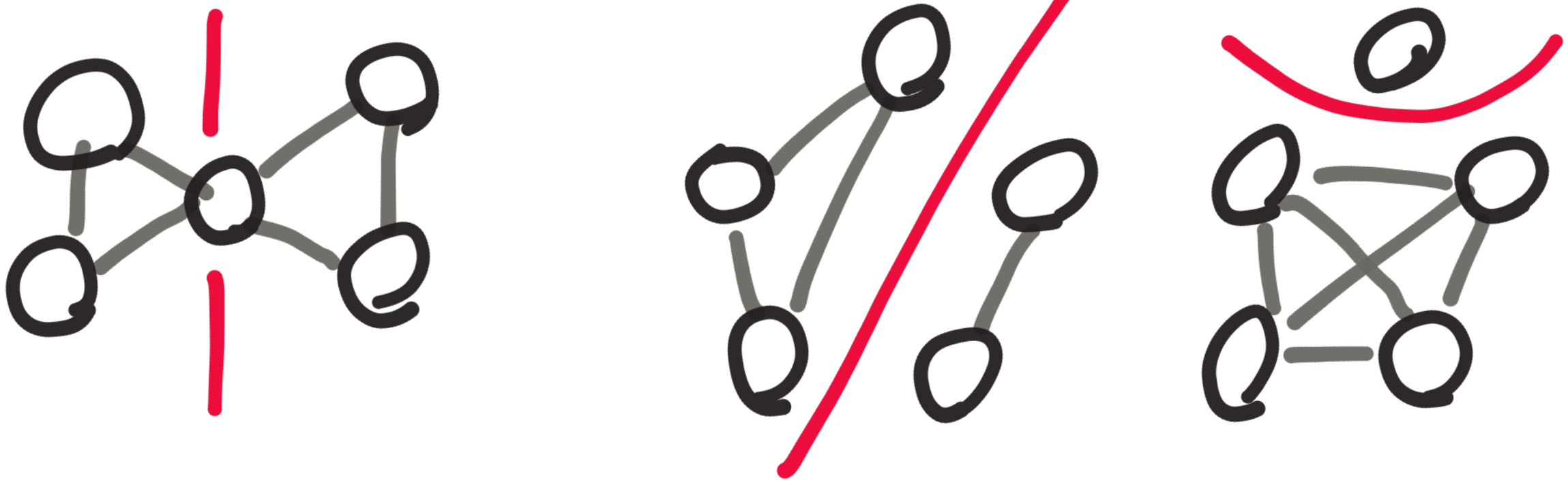
elastic

[search]

First Elasticsearch tests: June 2014

v1.1.0

Split Brain from



Massive data loss: 90%

ES added a great page on safety issues

Screenshot of safety docs

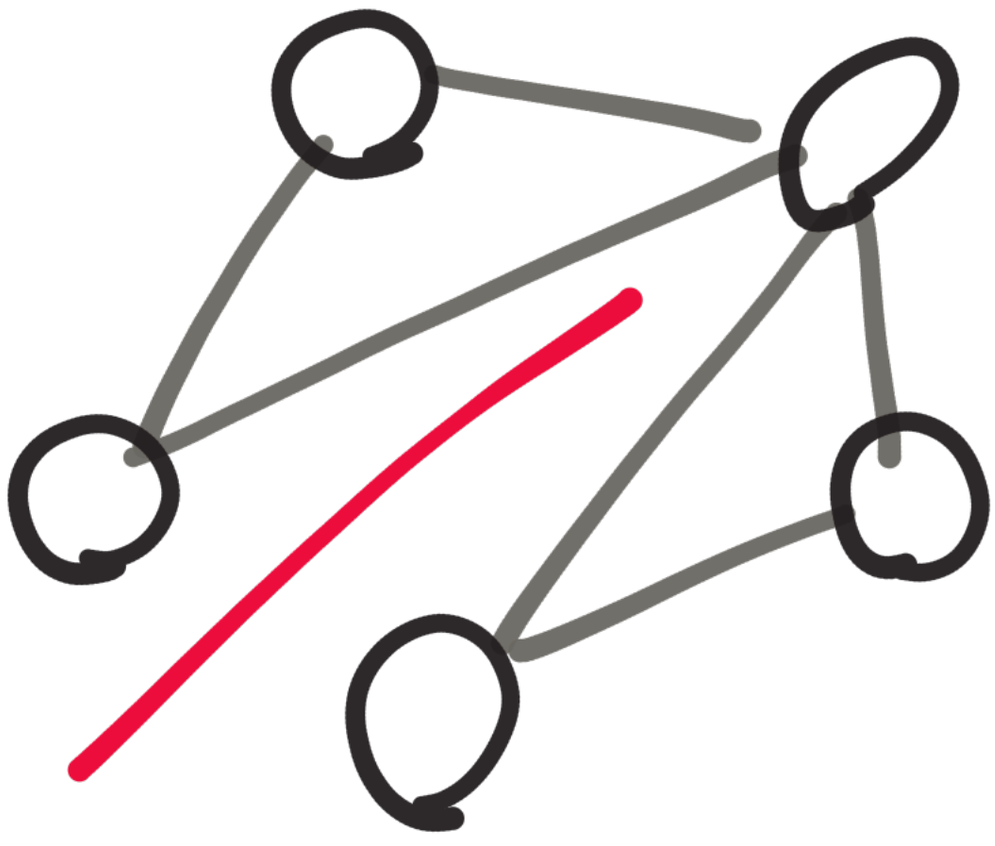
Screenshot of closed issue

Folks are still referring to the last talk

Screenshot of "not a big deal"

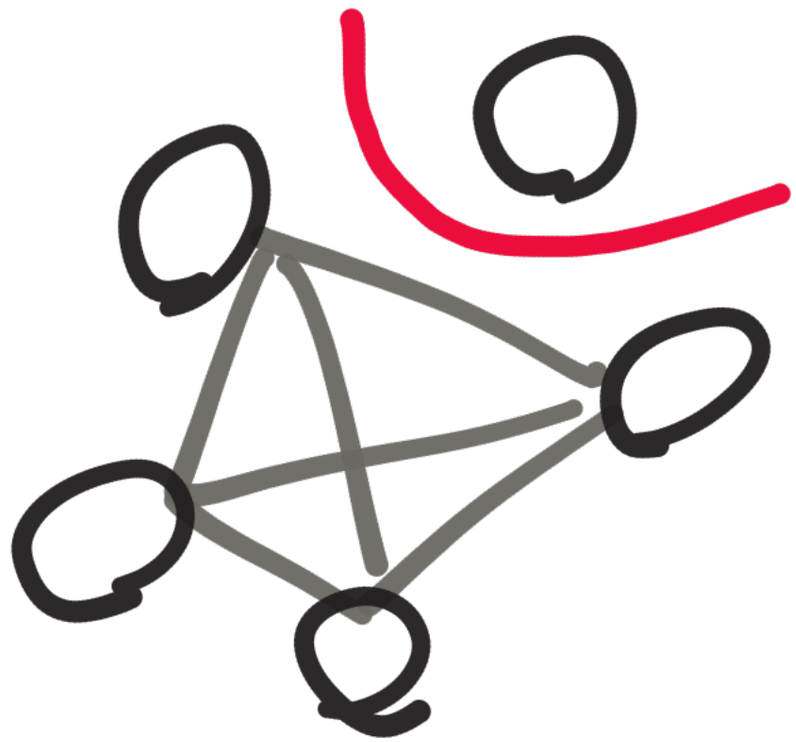
Retesting with ES 1.5.0

Intersecting partitions still cause data loss



Long-lived split-brain appears fixed, but docs are still last

Single node partition still cause data loss!



Electing a new primary still takes

~90 seconds

- Writes will stall

Elasticsearch is doing better, esp. with documentation!

Elastic Search Recommendations

- No longer catastrophic
- But, still loses data in all failure modes
- 


KEROSPIKE

High-performance 5-D KV store for online analytics (esp. in adtech)

"ACID reliability with no downtime - ever!"

Screwlit: no data loss

"No data loss. Row-level locking... immediate consistency (ACID), with synchronous replication"

We have to go

Deeper

"AS is by and large an AP system that provides high consistency"

"Aerospike provides read-committed isolation level using record locks to ensure isolation between [txns]"

strong

linearizable

sequential

causal

PRAM

RYW MR MW

WFR

serializable

serializable

RR

CS

MAV

SI

P-CI

RC

AP ✓

"for ops. on single keys with replication, Aerospike provides immediate consistency using synchronous writes to replicas"

"...read requests are guaranteed to find the newly written data..."

strong

linearizable

sequential

serializable

serializable

RR

SI

CS

causal

PRAM

RYW

MR

MW

WFR

MAV

RC

P-CI

"Virtually eliminate partition formation proven by years of deployment in DC and cloud environments."

"A key design point of As is to setup cluster nodes that are tightly coupled so that partitions are virtually impossible to create"

Screenshot:

Deploy page

So for a CaS

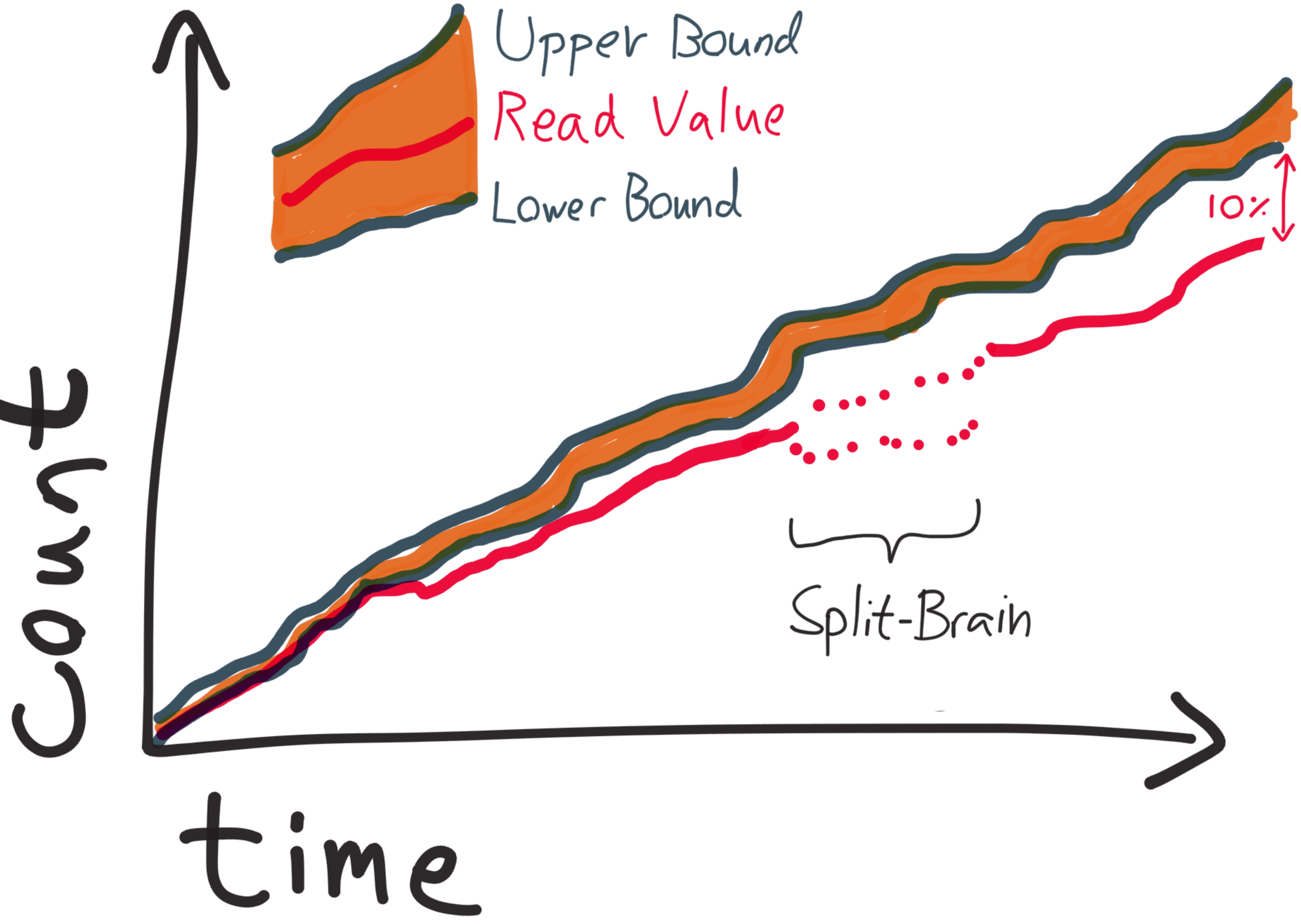
Register...

6
w2
7
w4
11
w0
4
r2
10
w4
12
r0
CaS 0→3

What about commutative ops?

$$a + b = b + a$$

Counters will drop increments during a partition

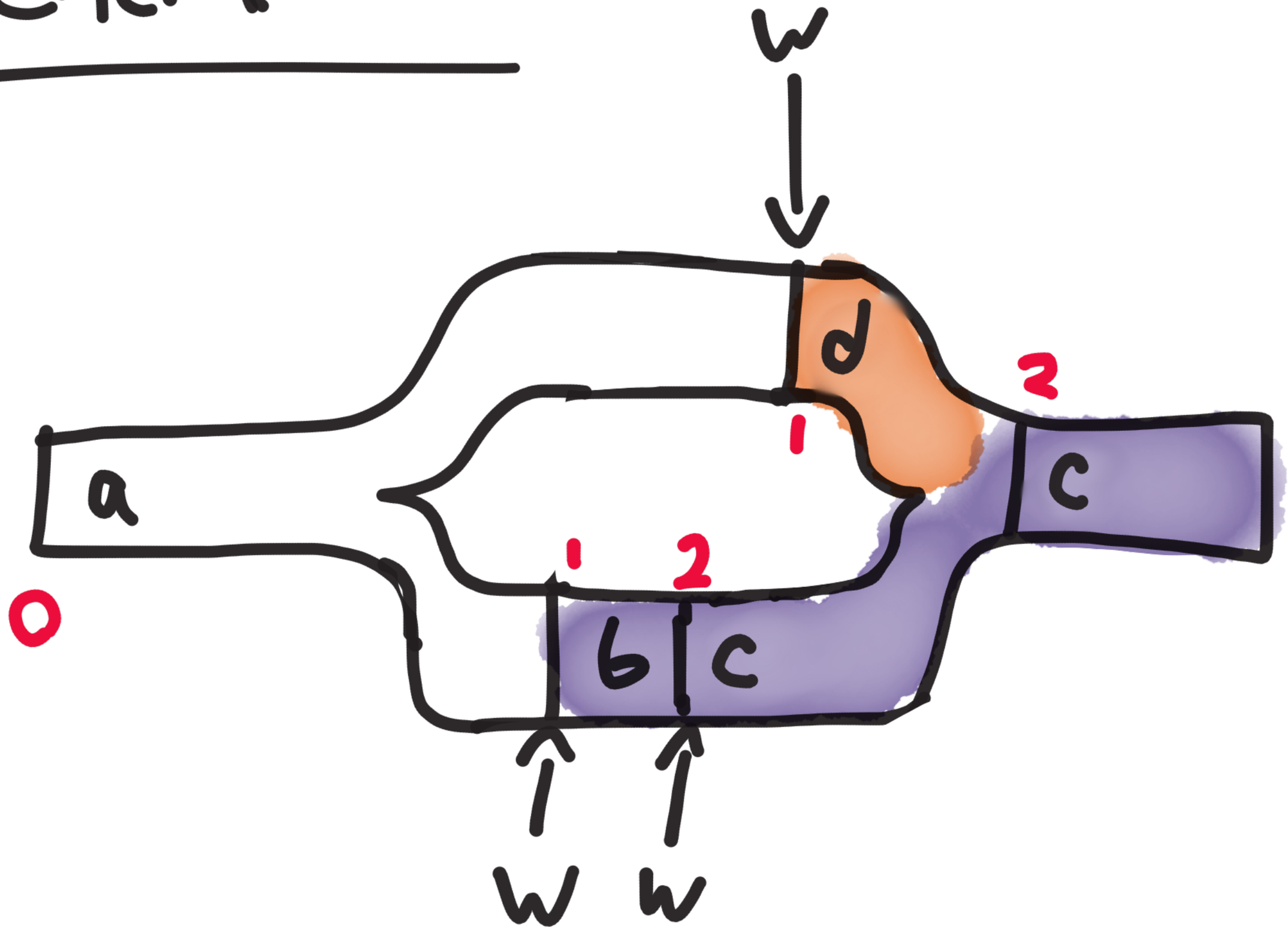

Upper Bound
Read Value
Lower Bound
10%
Split-Brain
count
time

"Deployment modes
AP vs. CP"

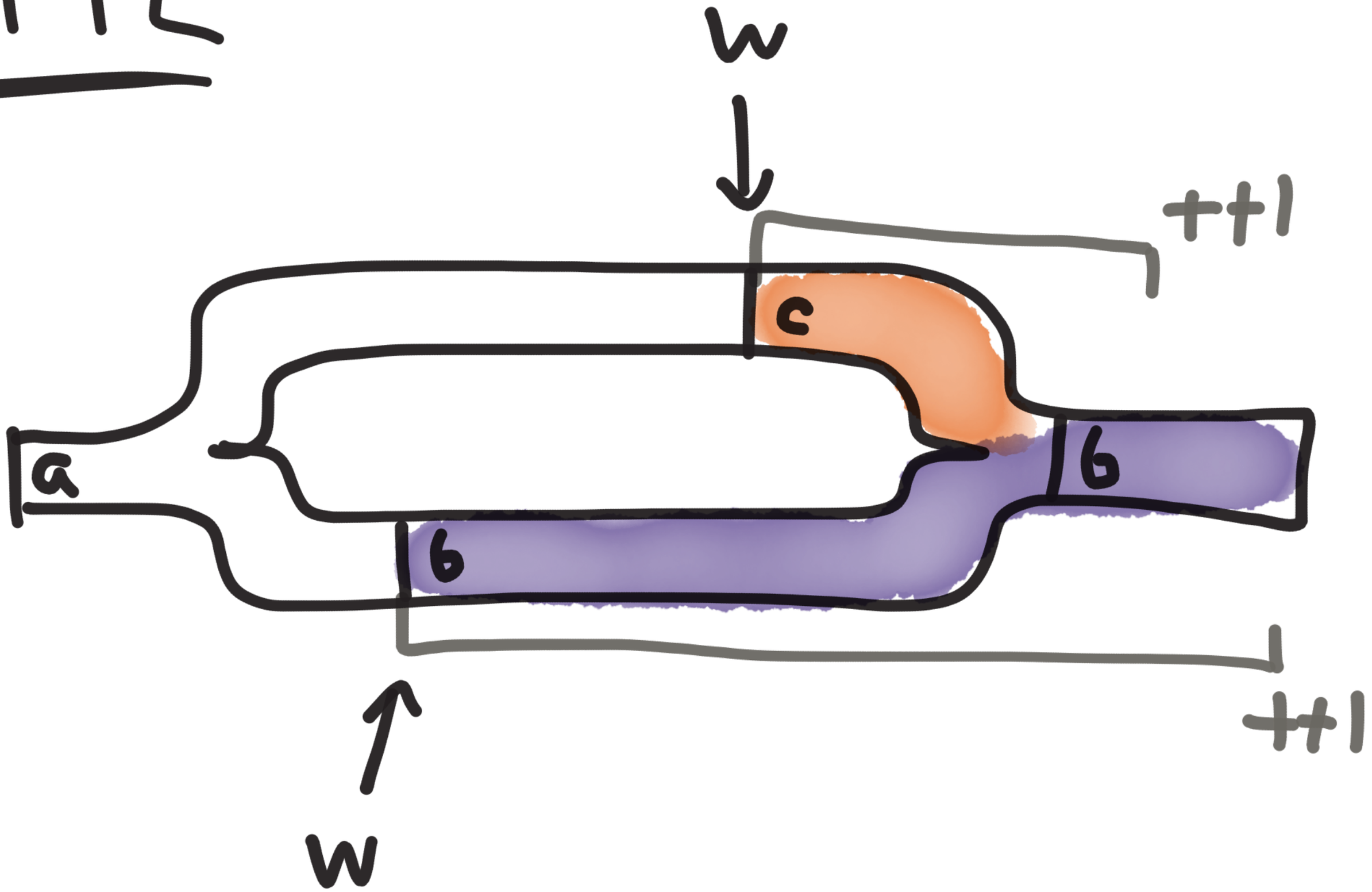
CP is vaporware

OK so how does AP mode work?

Generation



TTL



LOST

UPDATES

strong

linearizable

sequential

causal

PRAM

RYW MR MW

WFR

serializable

RR

CS

RC

serializable

SI

MAV

P-CI

"Why not use your PAXOS implementation?"

Probably fine for stats data, but expect some loss

Recap

Absence of evidence

— is not —

Evidence of correctness

Mongo ❤ ES

have made

improvements!

Still Surprises

lurking in the

DEEP

Read the docs!

Then

MEASURE

- for -

YOURSELF!

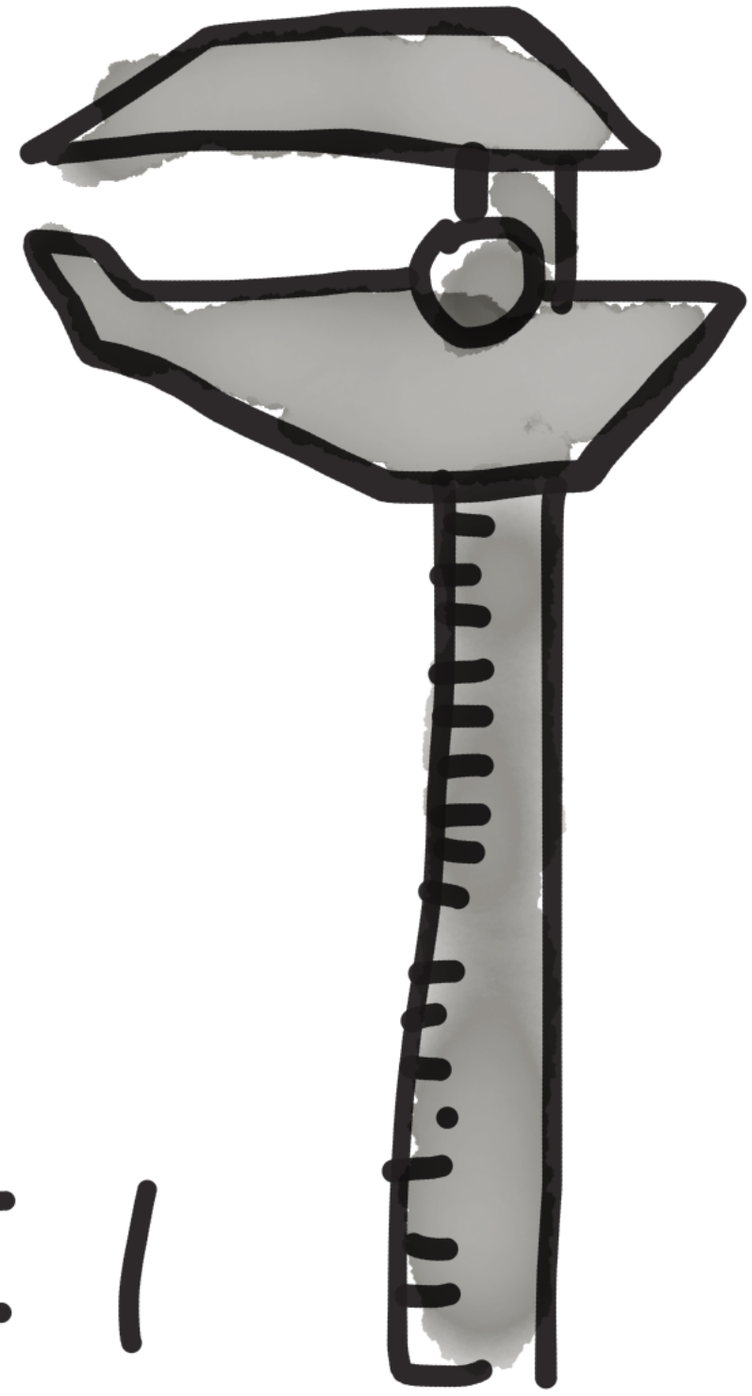


Figure out the invariants your system needs

How much

can you tolerate?

Consider your failure modes

Partitions

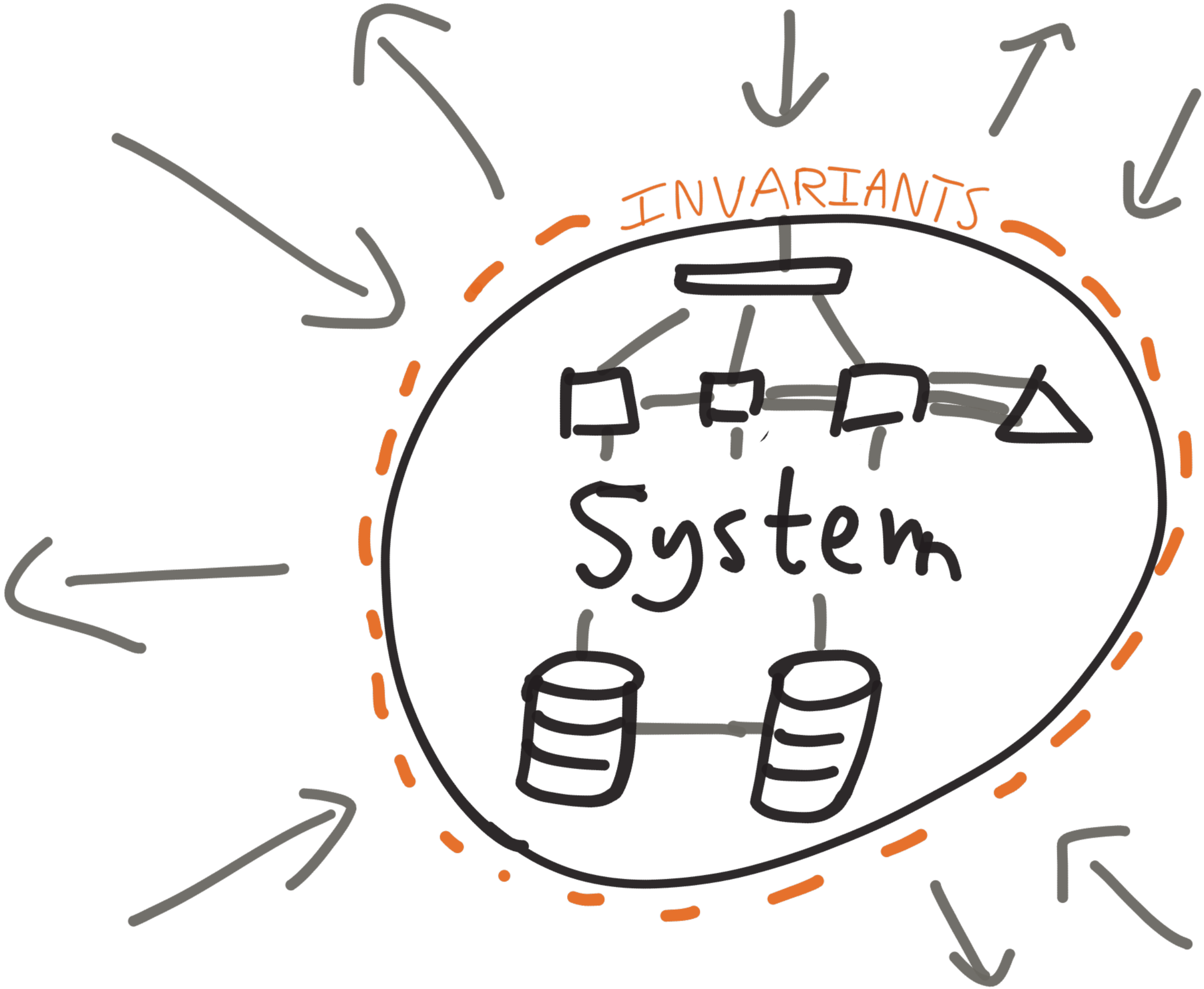
IO/GC pauses

Node failure

Clock skew

Test your Systems

end ——— to ——→ end


INVARIANTS
System

Thanks!

Kevin Porter
Lucien Valmar
Stripe
Evan Broder
Boaz Leskes
Asya Kamsky
Lee Hinman
Kelly Stirman
Matt Kangas
Marc Hedlund