

J E P S E N

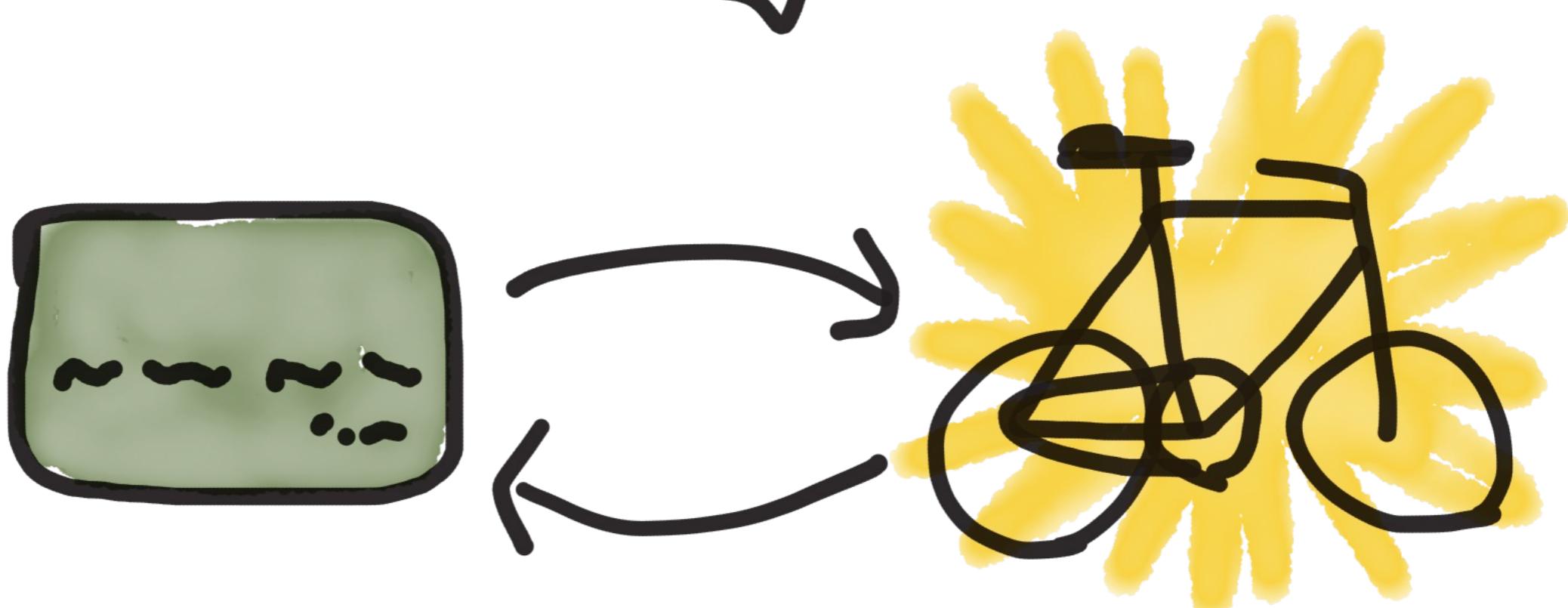
IV

Hope Springs
Eternal

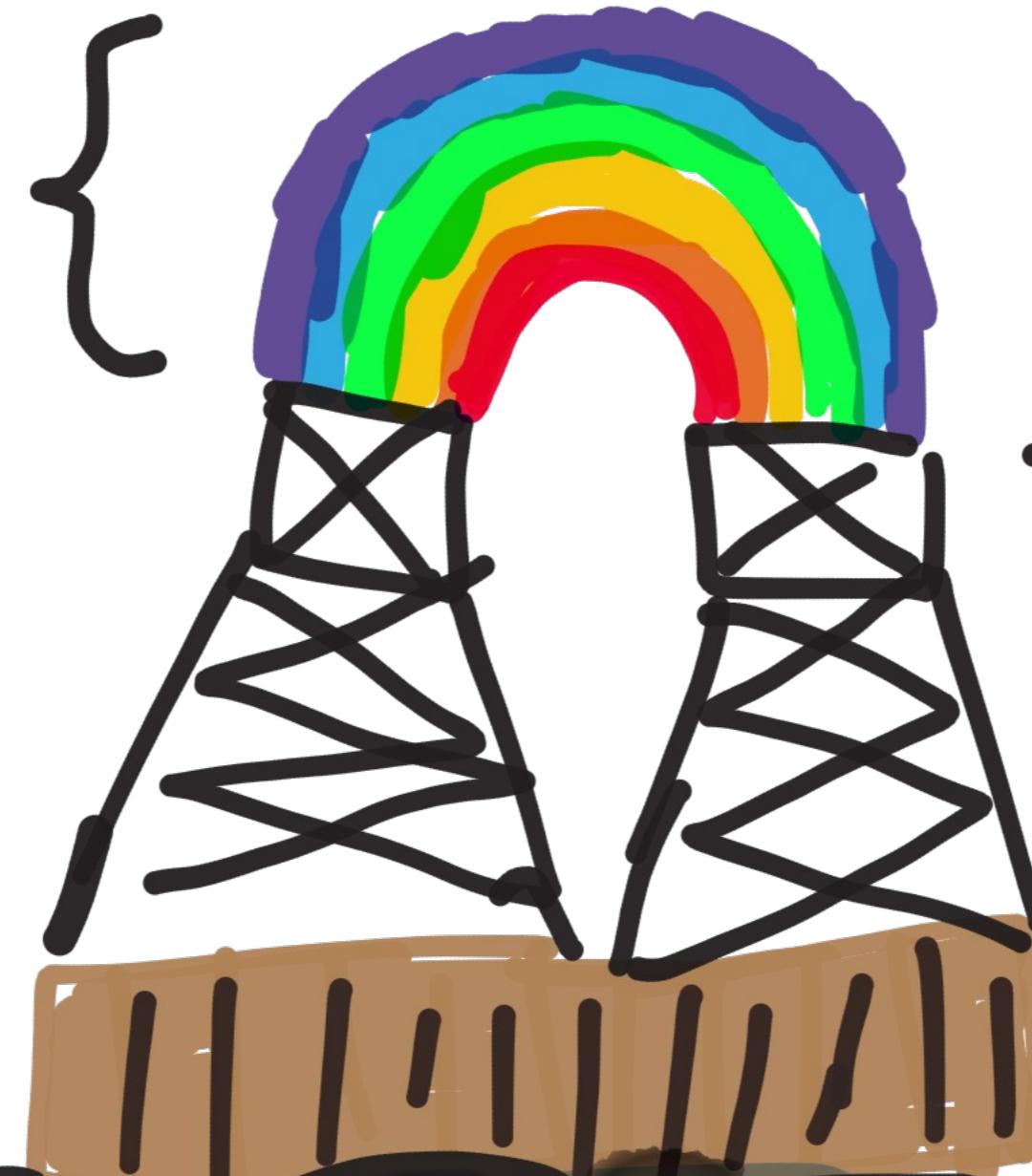


@aphyr
(Kyle Kingsbury)

straße



Public
API {



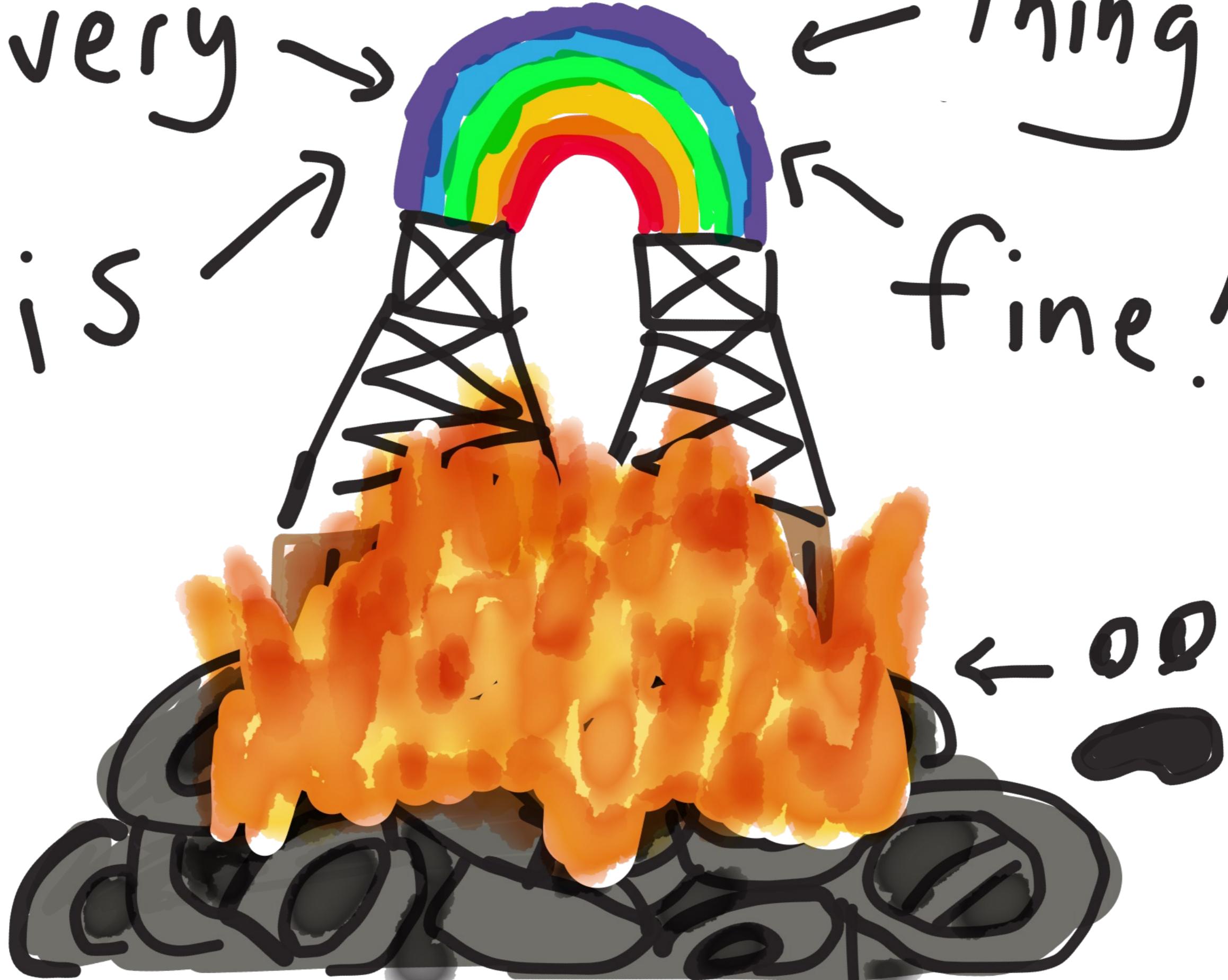
} API code

Ruby {

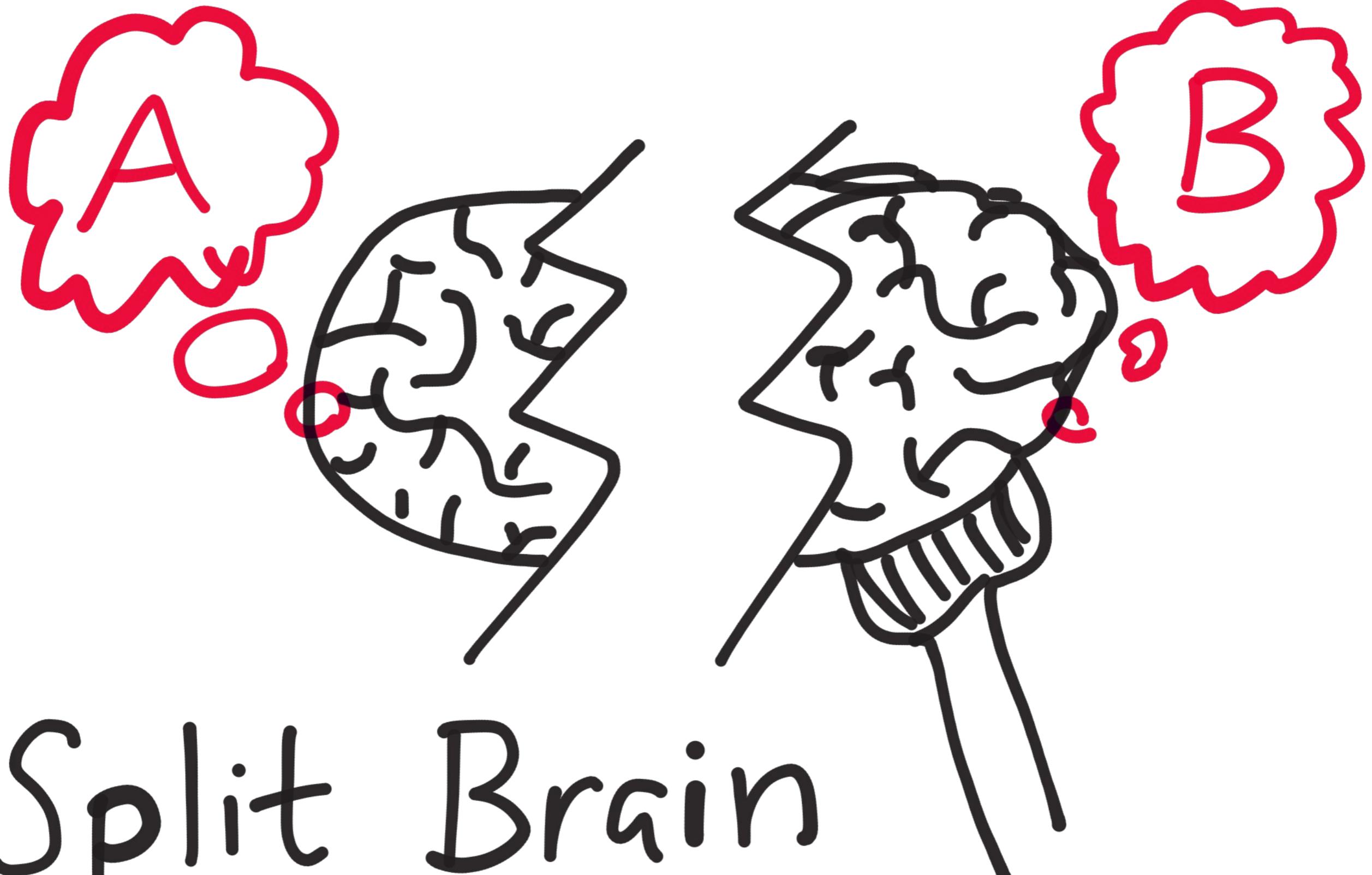


DBs

Every →
is →
thing ←
fine !

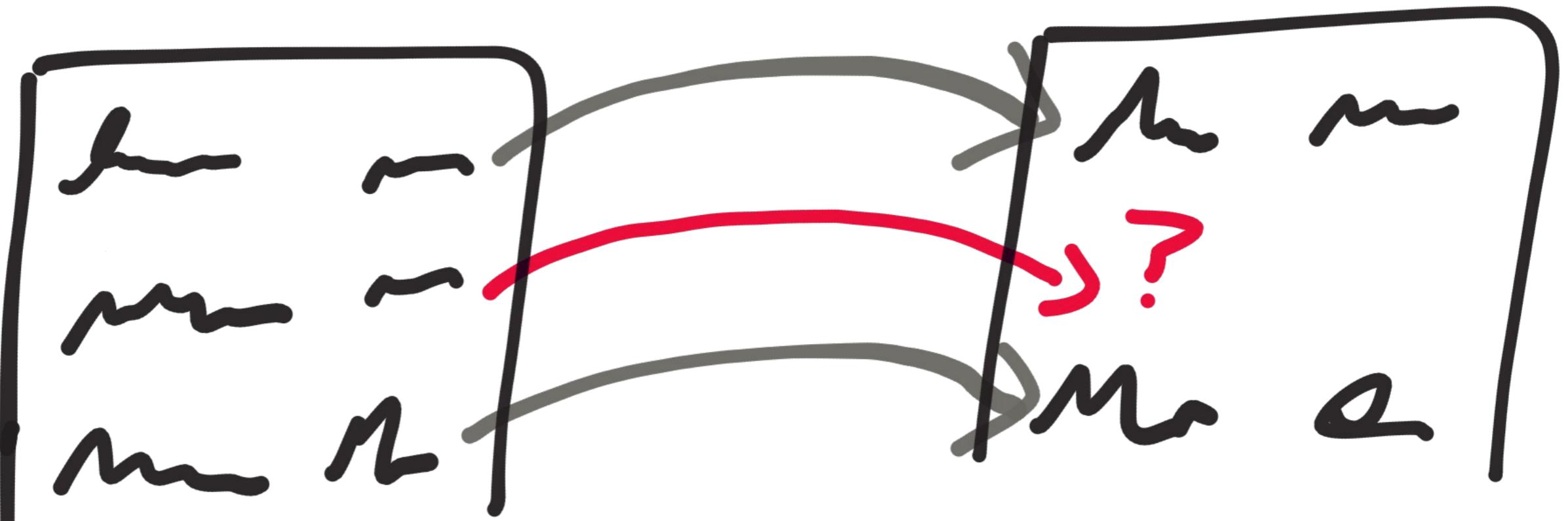


Databases! /
Queues! /
Discovery! /
THE HORROR

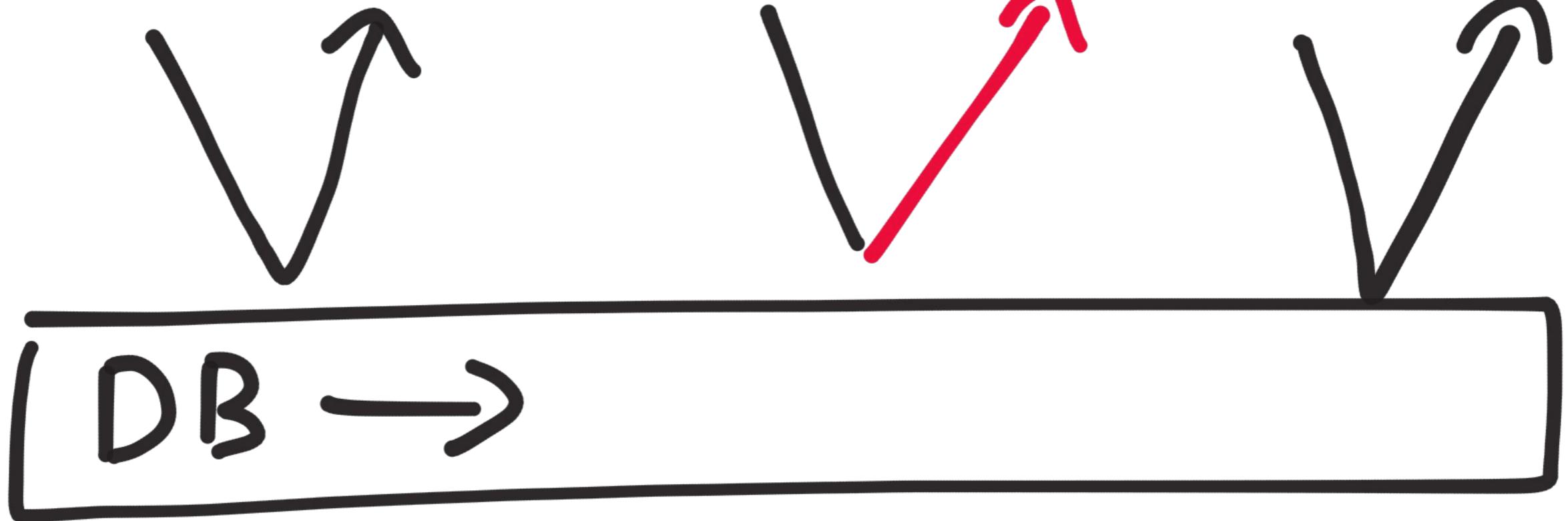


Split Brain

Broken Foreign Keys



write ok read \emptyset read ok



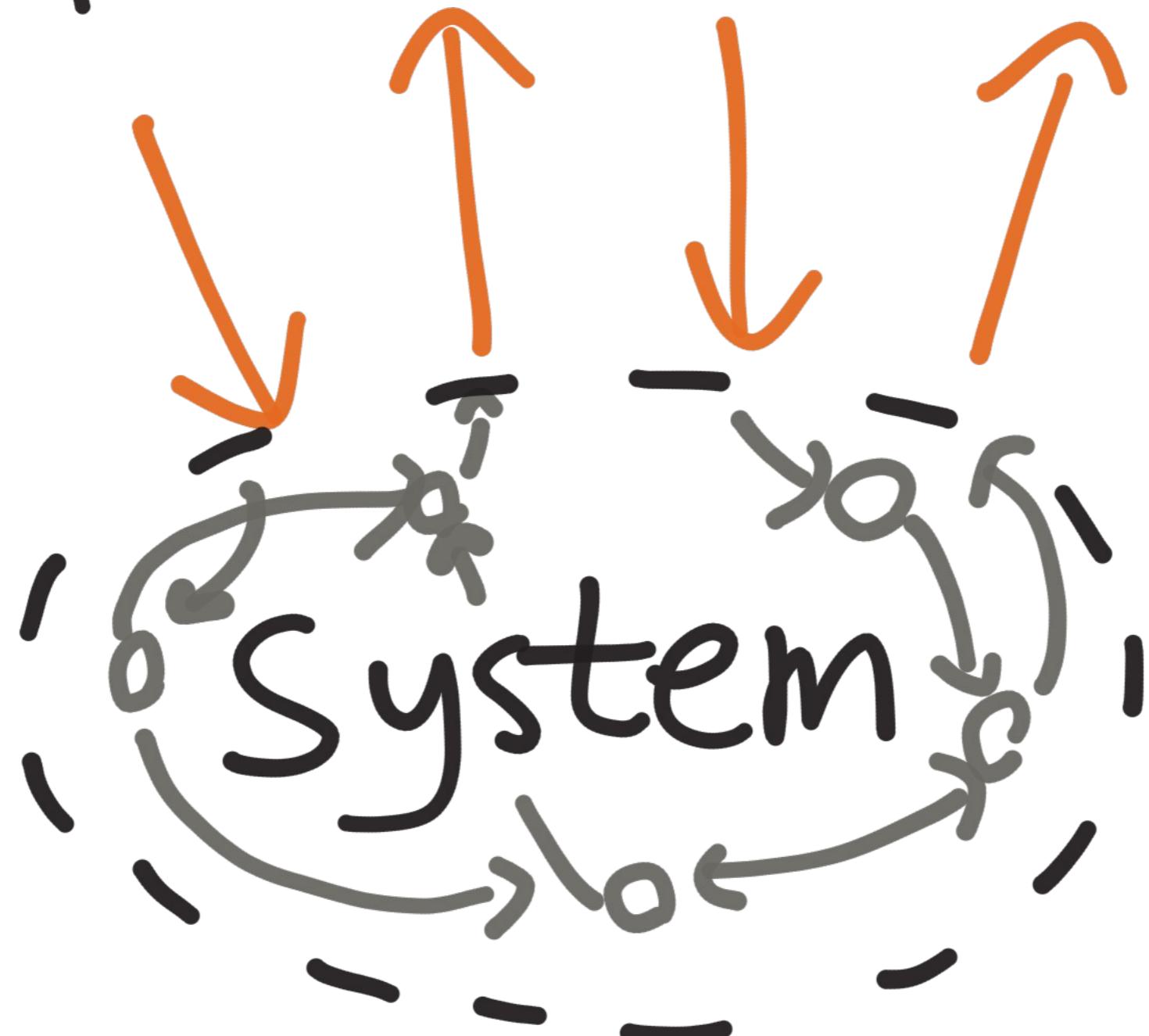
Anomalies

How do you
know if a
system is safe?

Jepsen

github.com/aphyr/jepsen

Environment





— INVARIANTS —

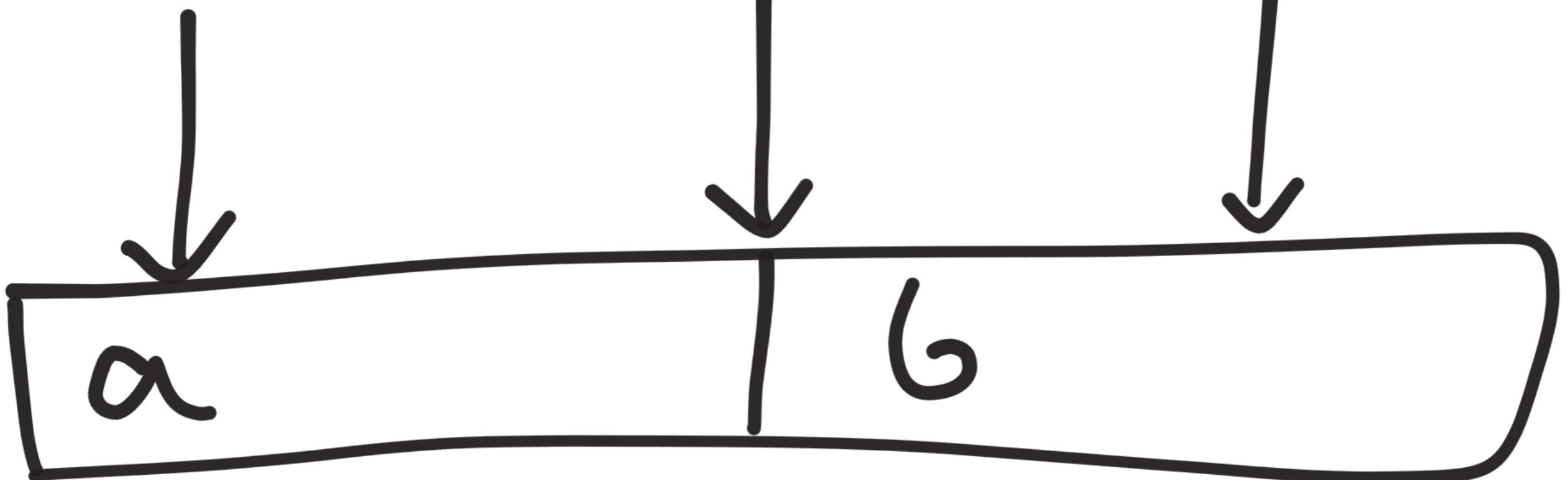


var $X = \alpha$

print X

$X = b$

print X

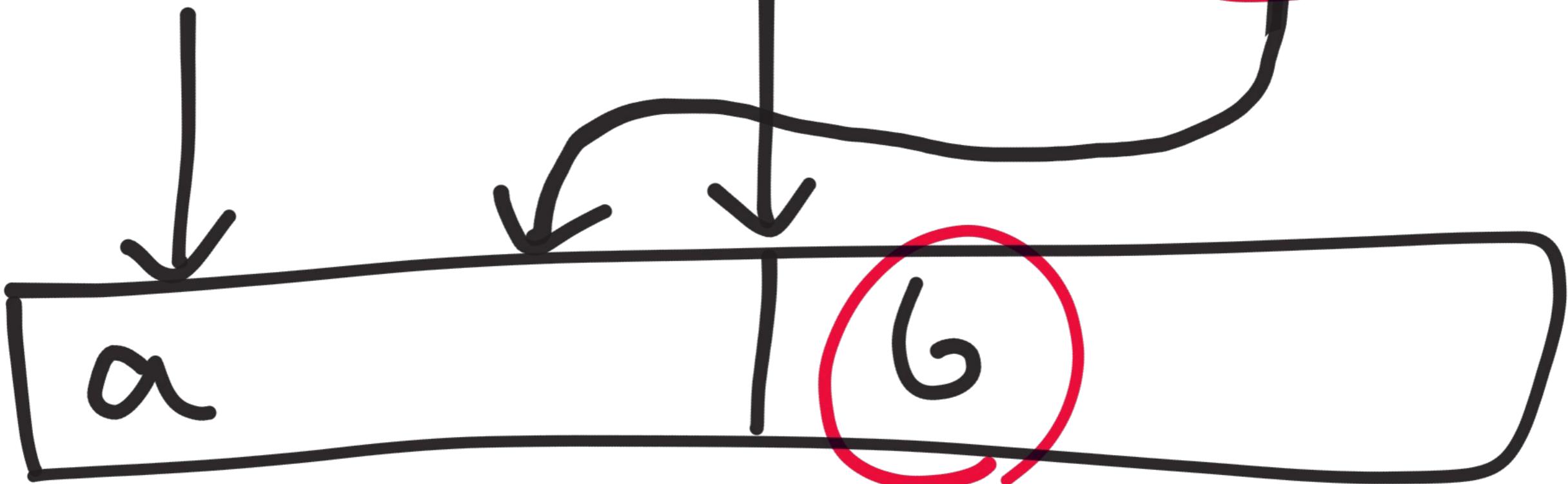
$r(a)$ $w(b)$ $r(b)$ 

time →

$r(a)$

$w(g)$

$r(a)$



time →

Invariants

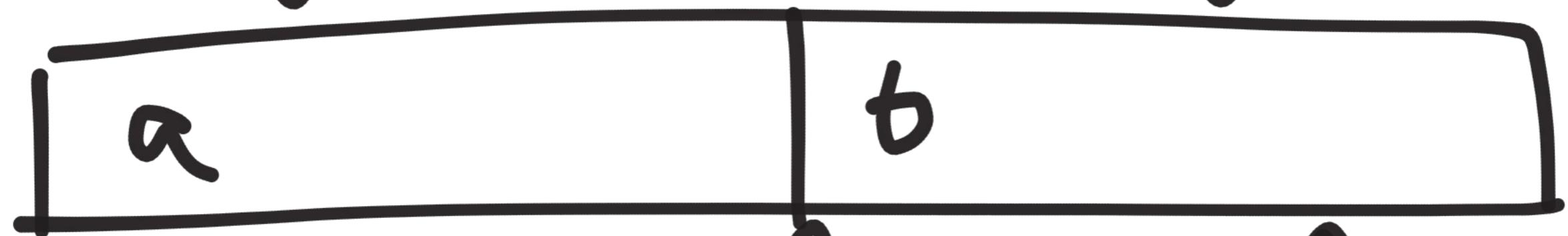
Constraint

Histories

What about

concurrency?

$r(a)$



$r(b)$



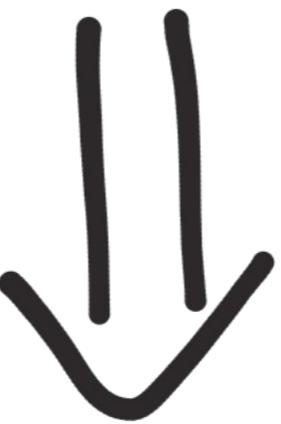
$w(b)$



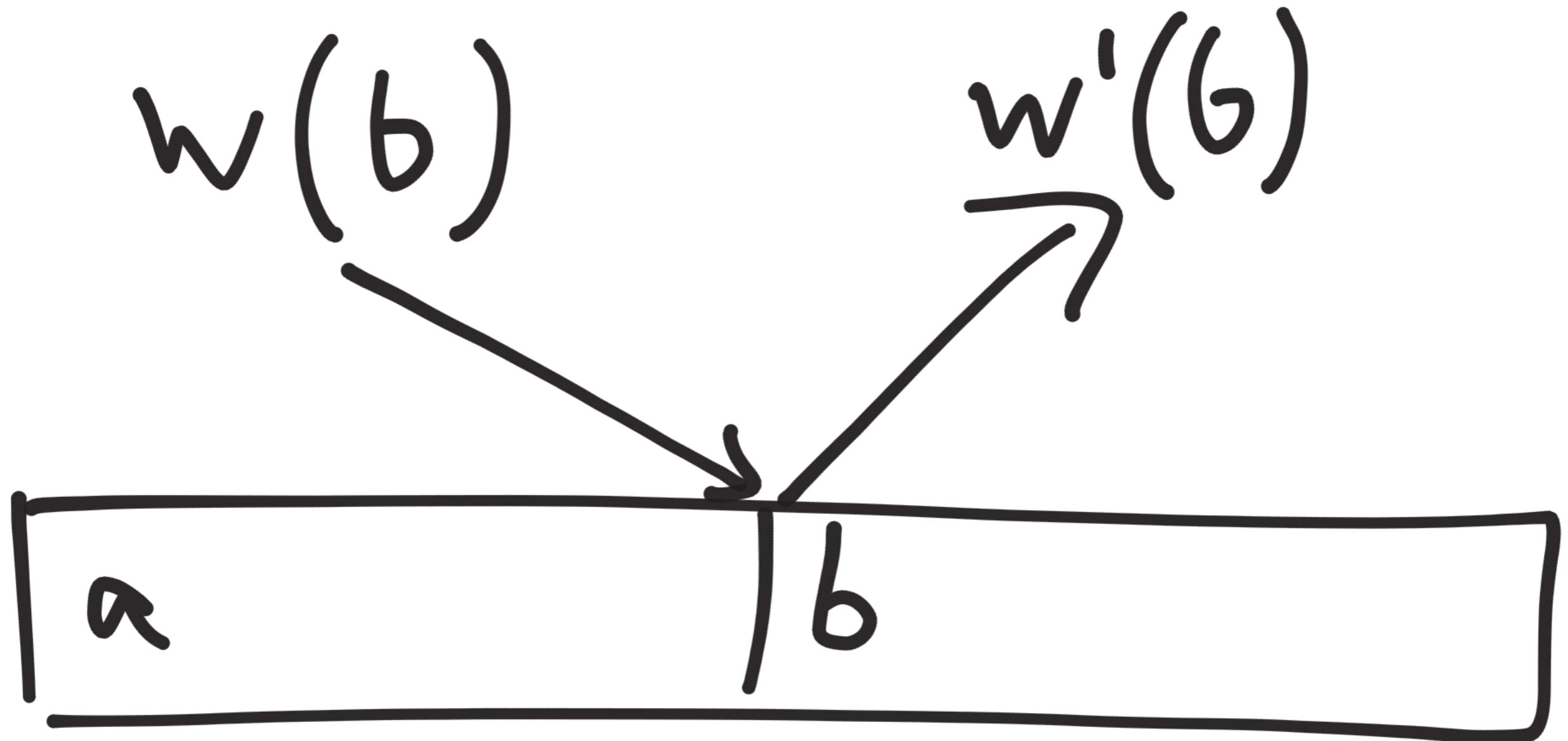
$r(b)$



Distributed

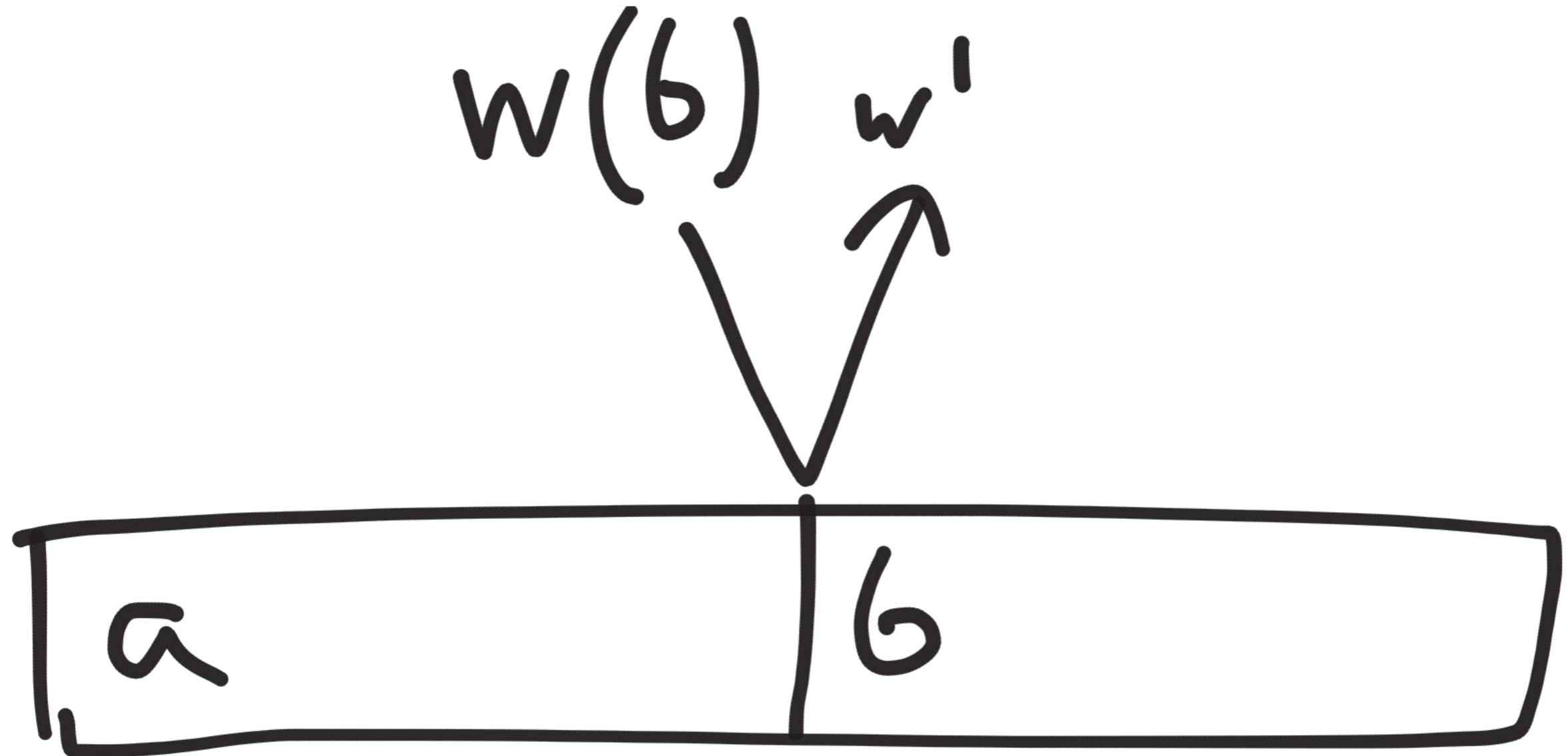


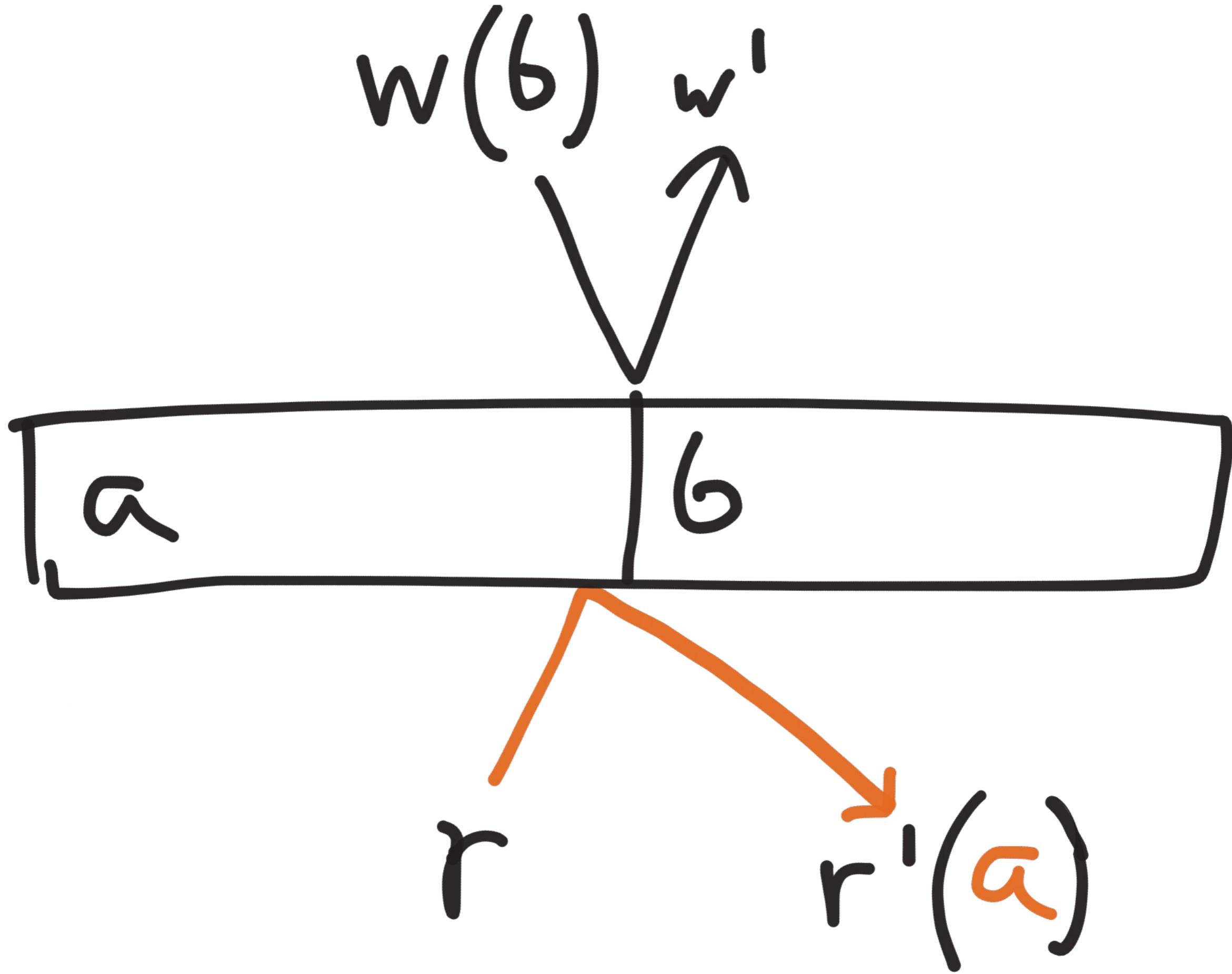
Distance



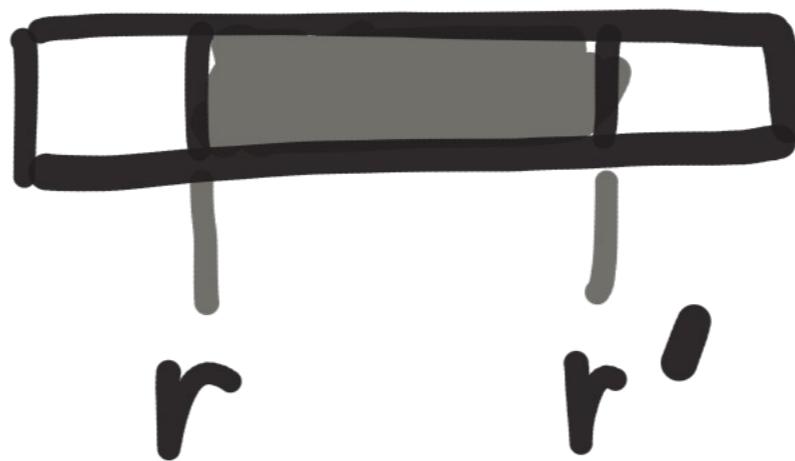
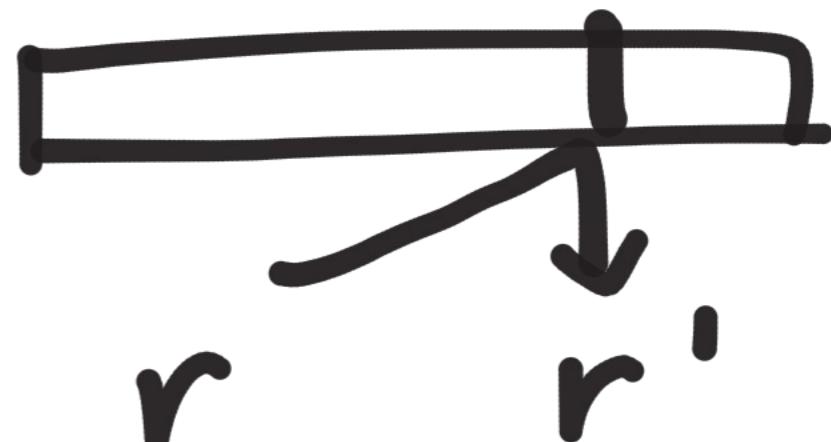
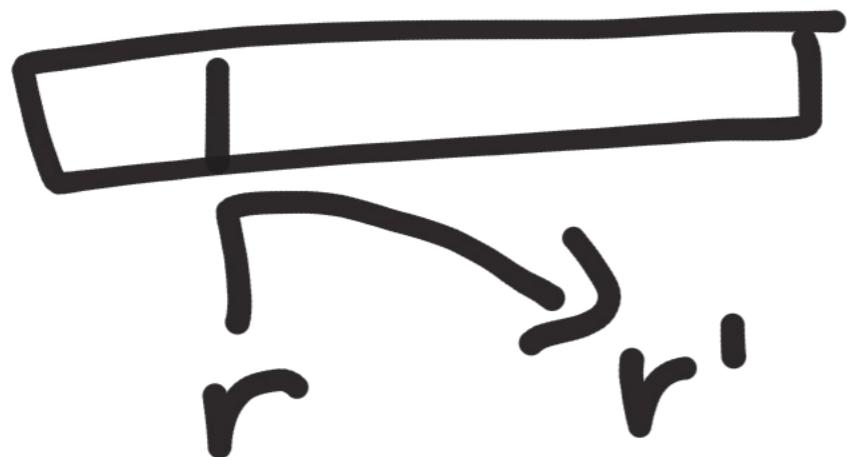
Mess@ger take time

Delays imply
ambiguous orders





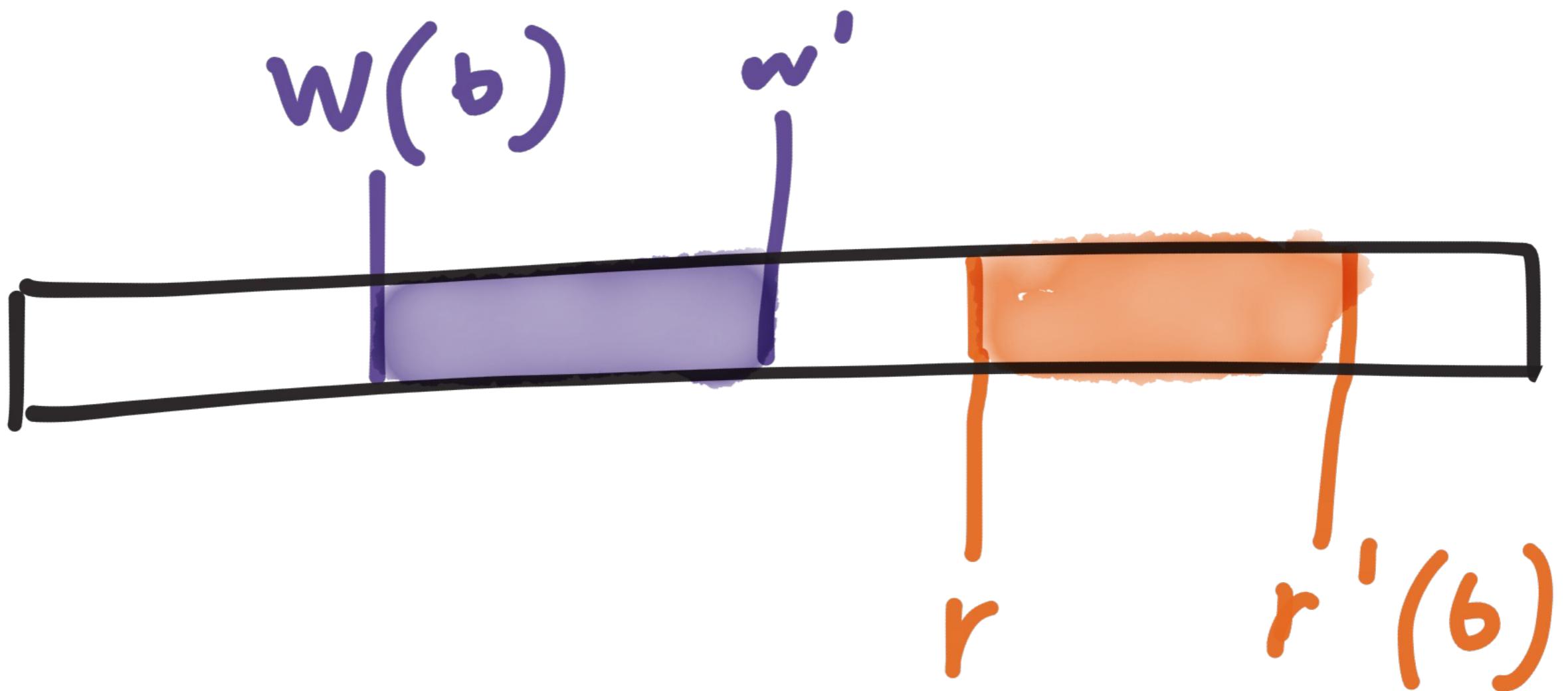
But there are finite
bounds on ambiguity!



Linearizability



After op is complete,
guaranteed visibility!

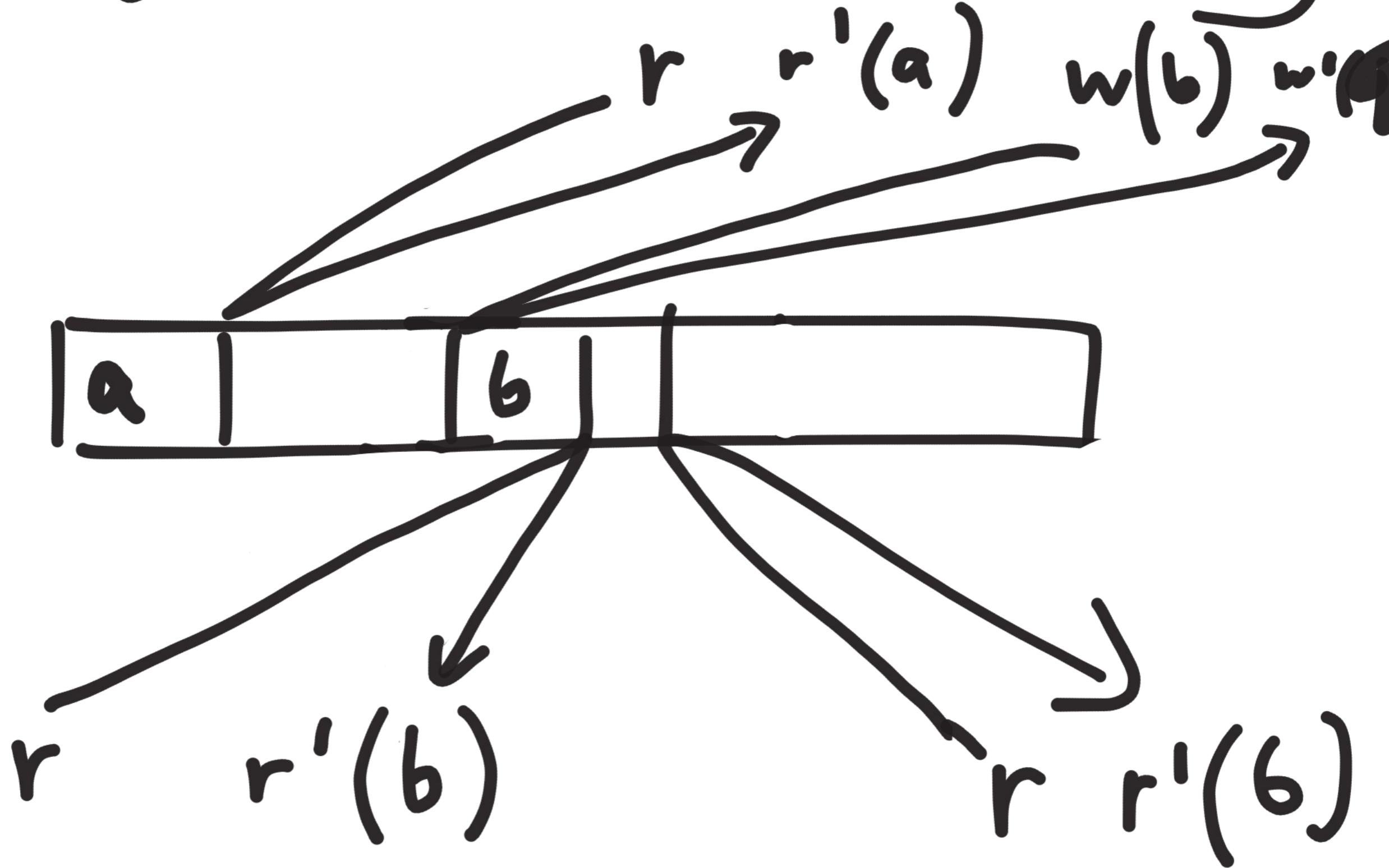


- Everyone agrees on order
- Ops are guaranteed to be visible to ALL clients once complete
- No stale reads

What if...
we relaxed the

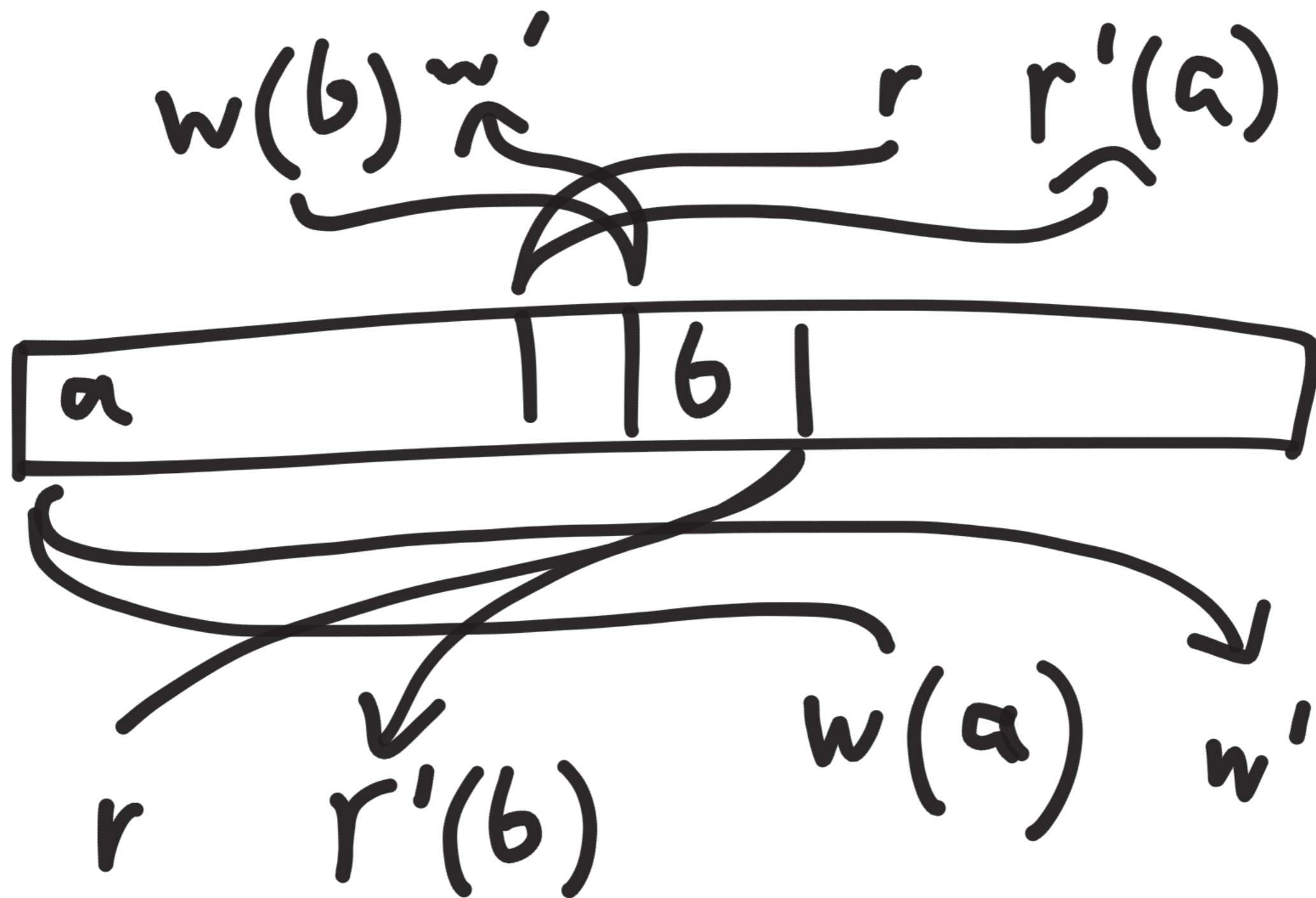
invariants?

Sequential Consistency



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

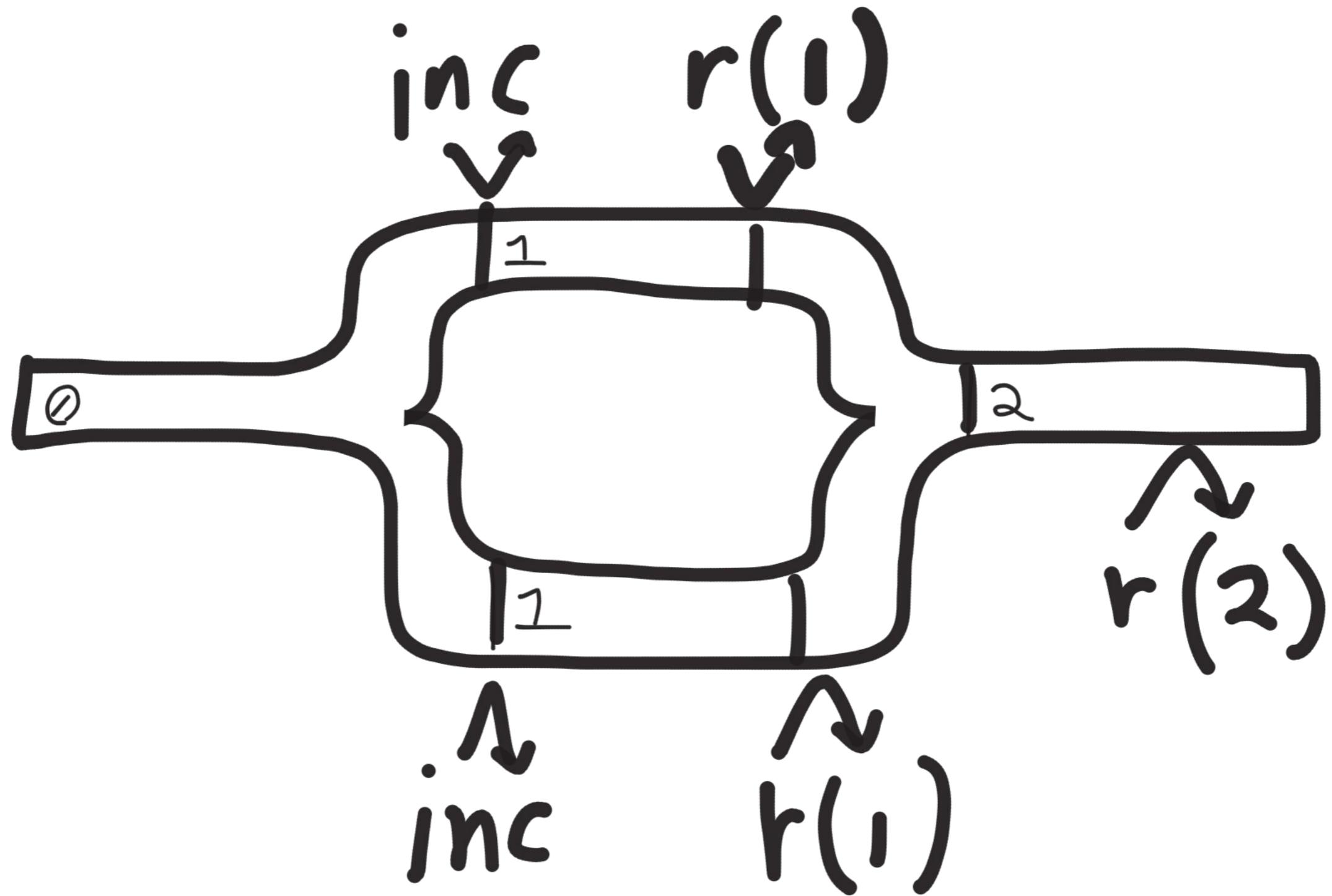
Serializable Consistency



Ops happen in SOME

Order, but nobody
has to agree on what
that was.

Eventual Consistency

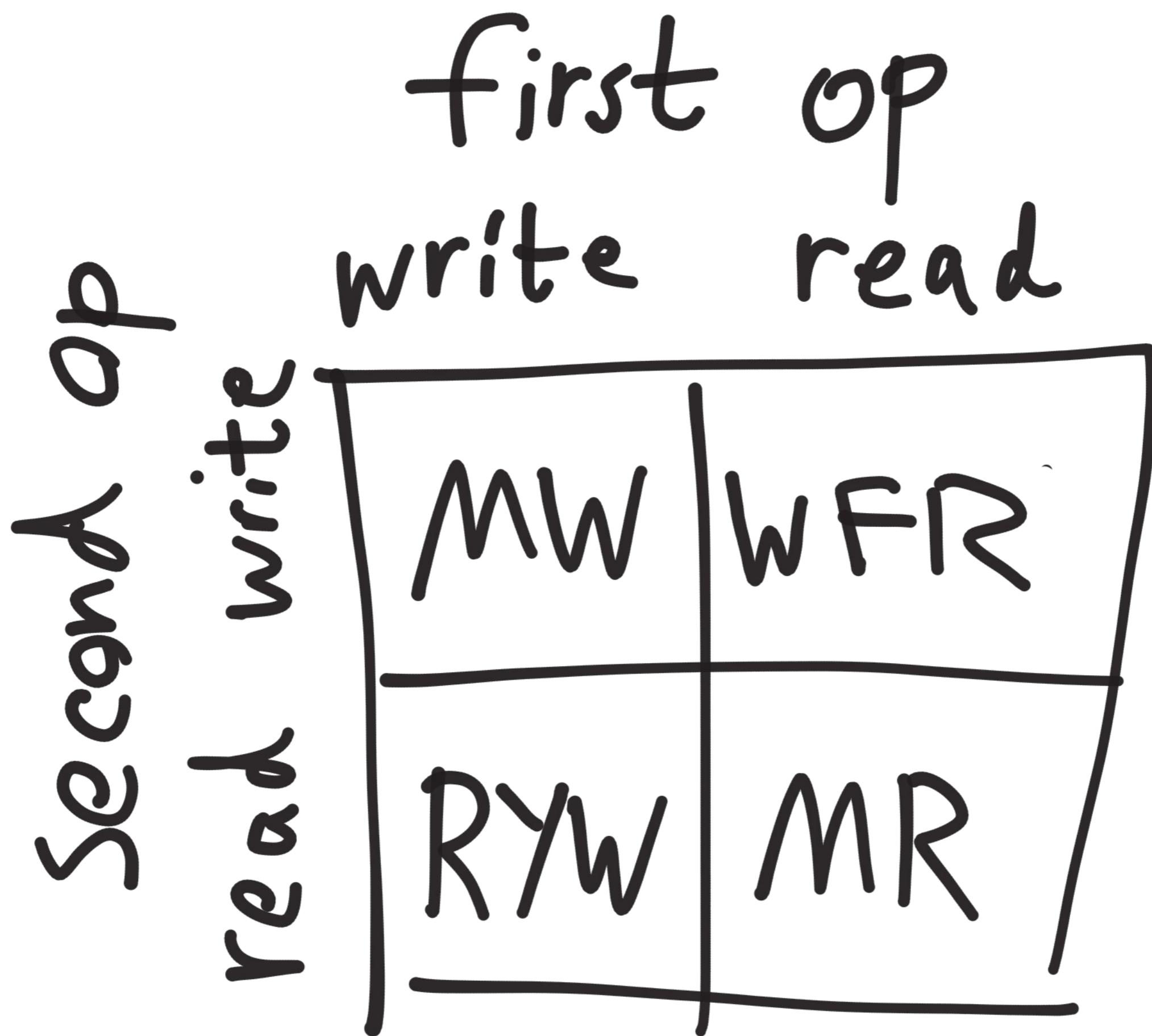


There doesn't have

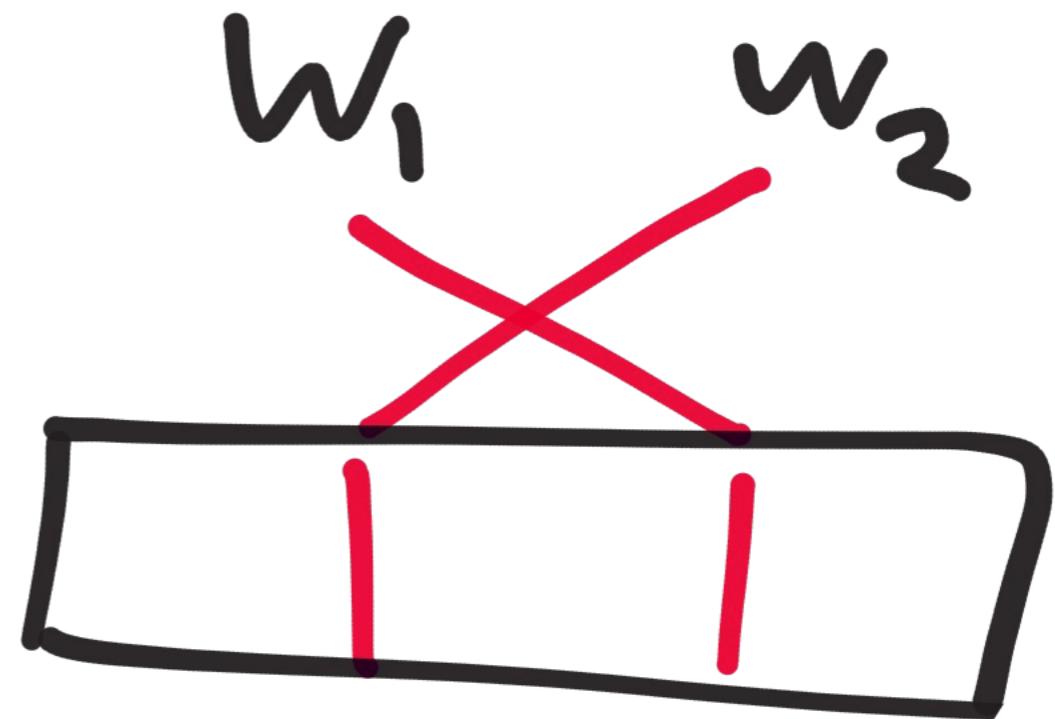
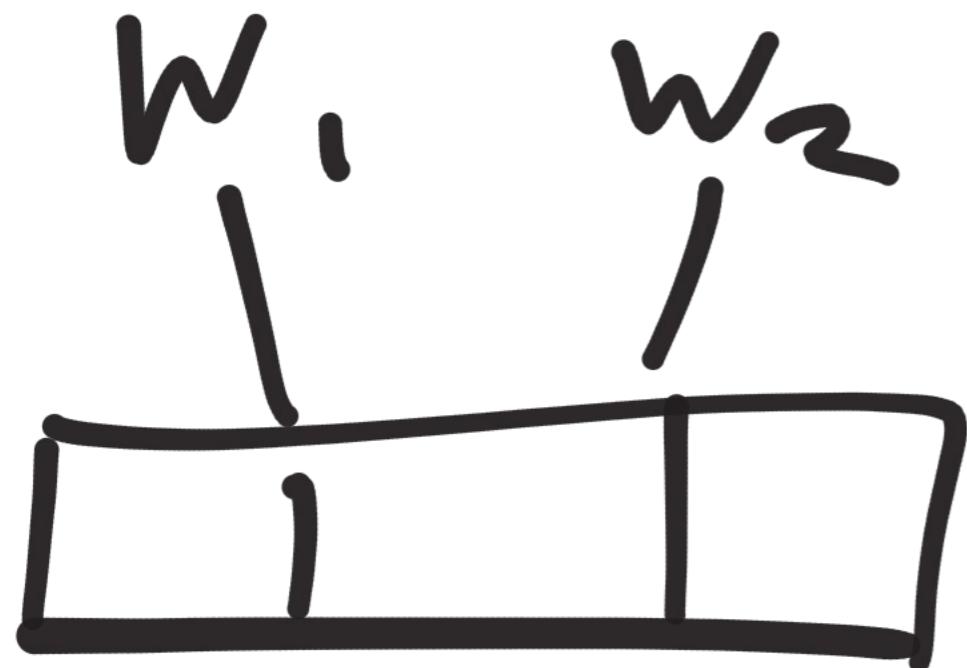
to be any single

order so long as

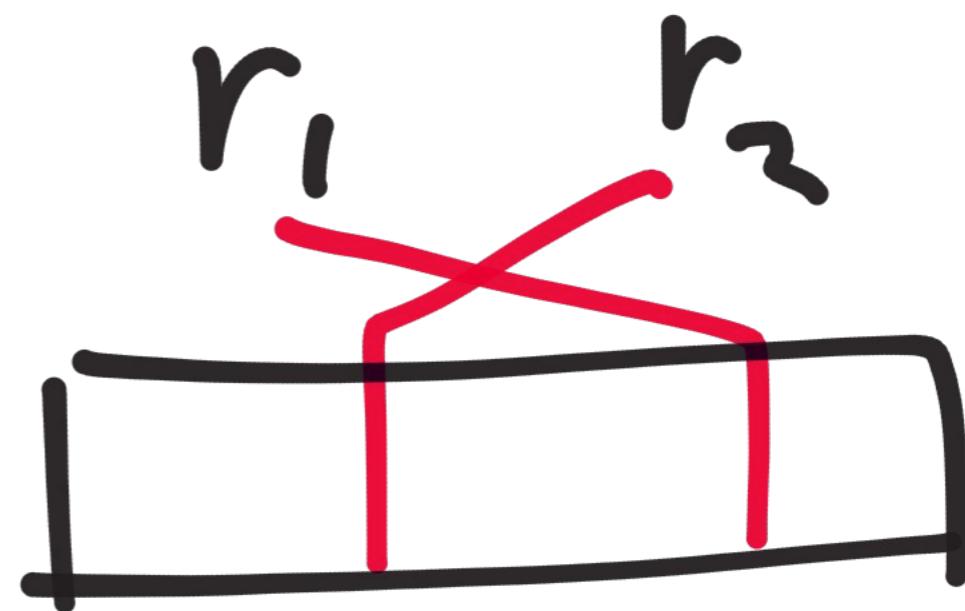
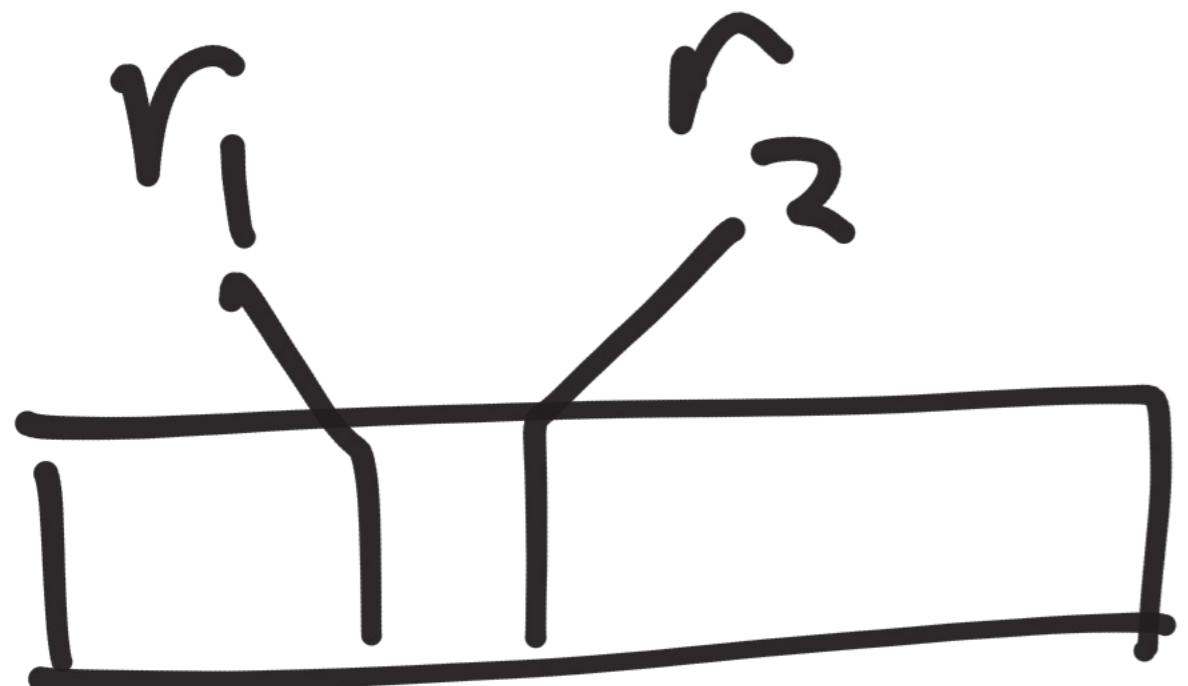
histories converge



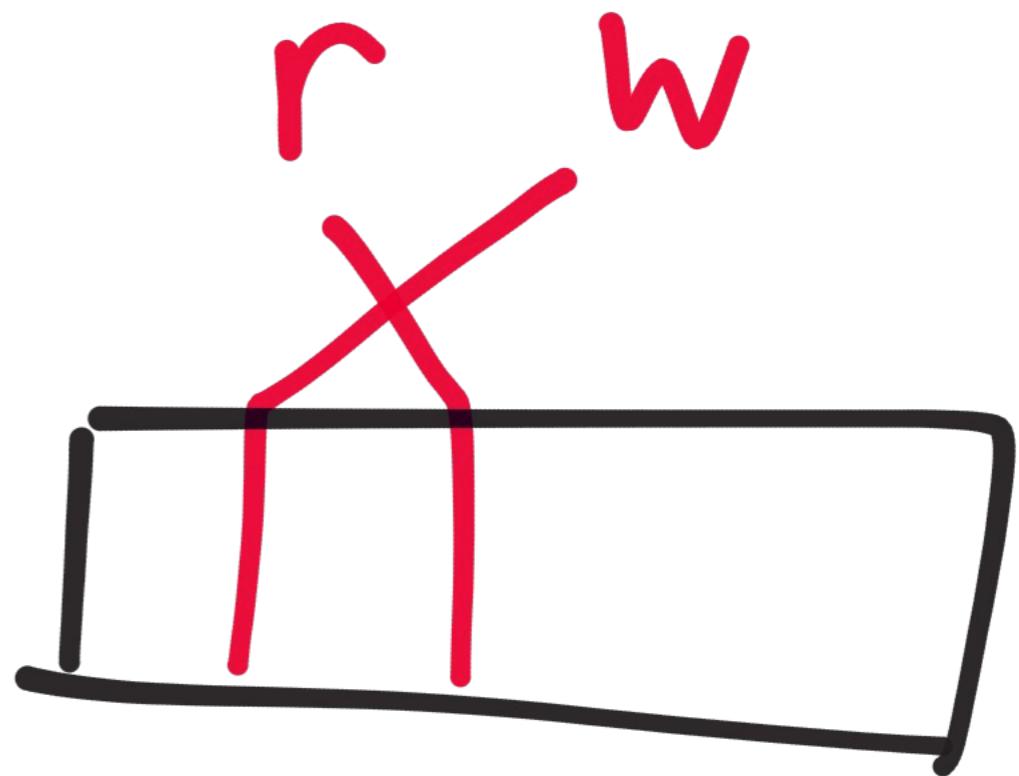
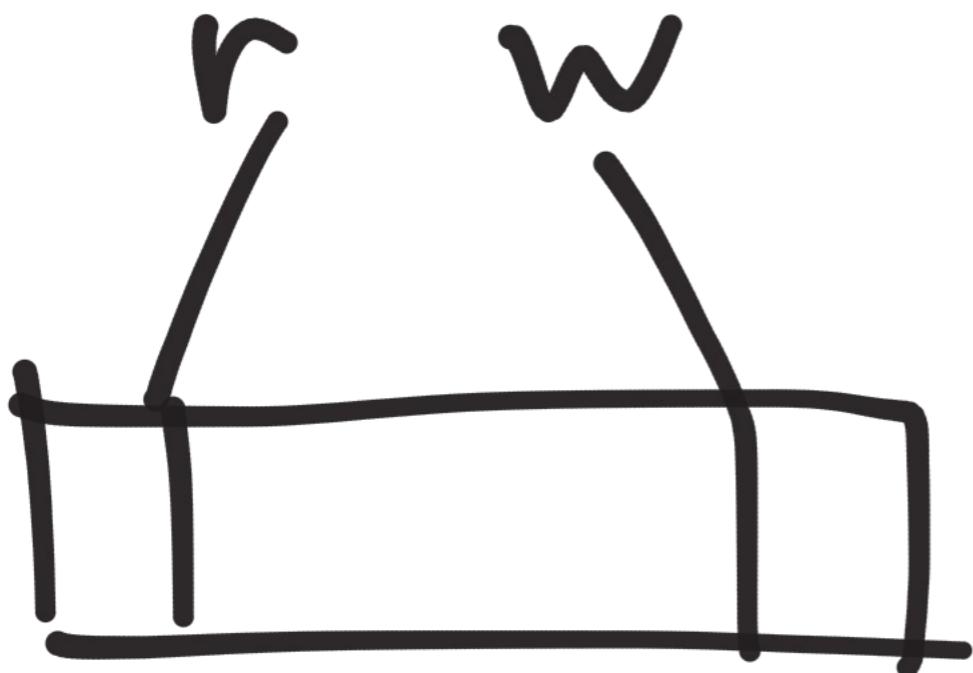
MW: Monotonic Write



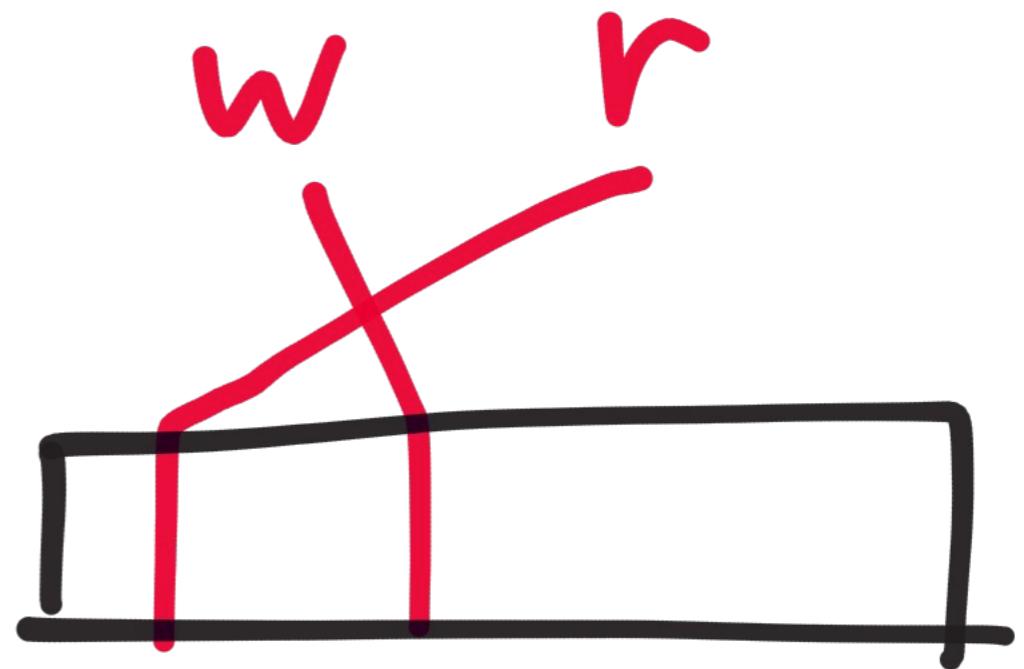
MR : Monotonic Read



WFR: Writes Follow Reads

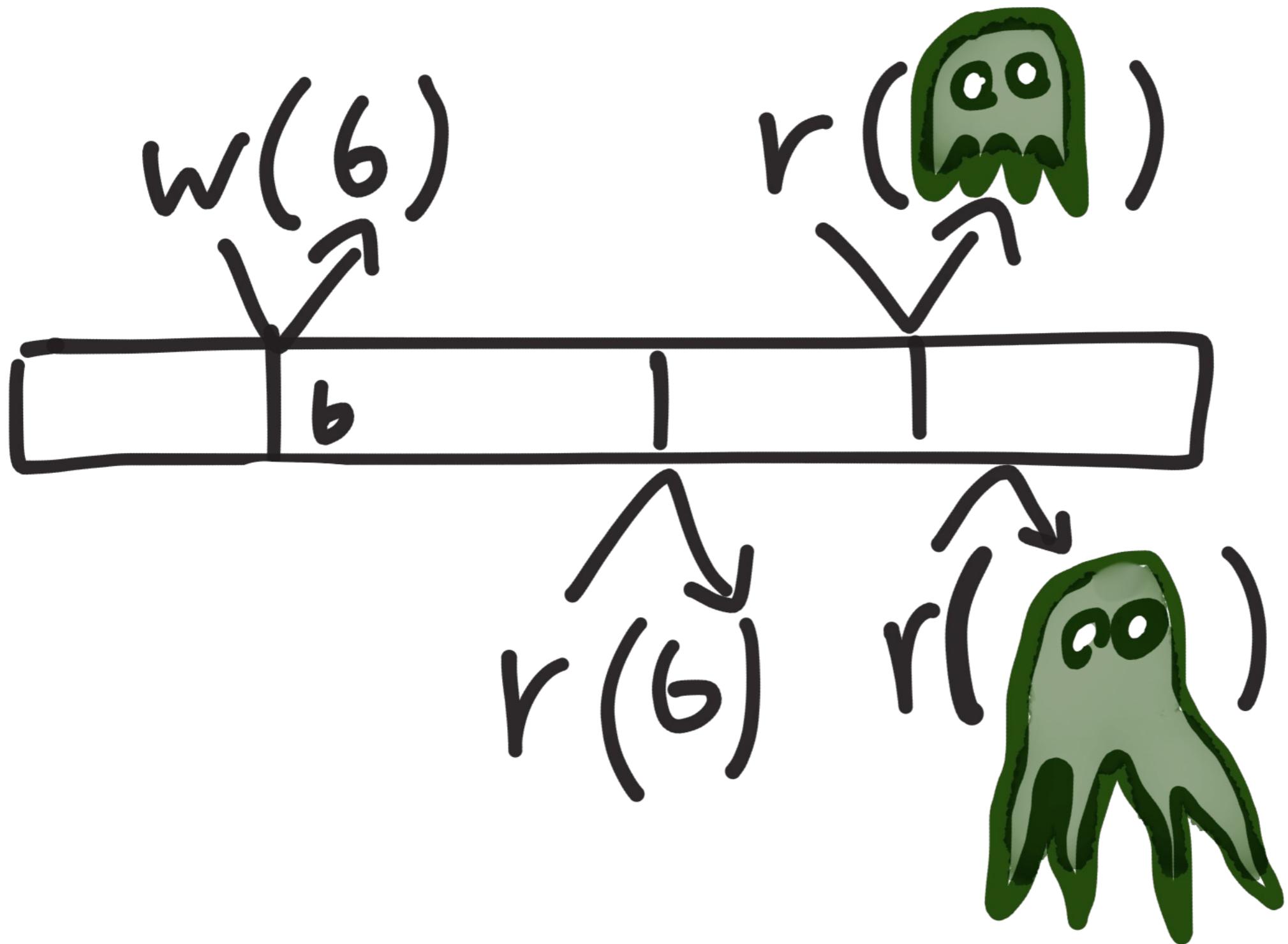


RYW: Read Your Writes



Infinitely many
Consistency classes

Cthulhu Consistency



strong
linearizable
sequential

serializable

serializable

RR

SI

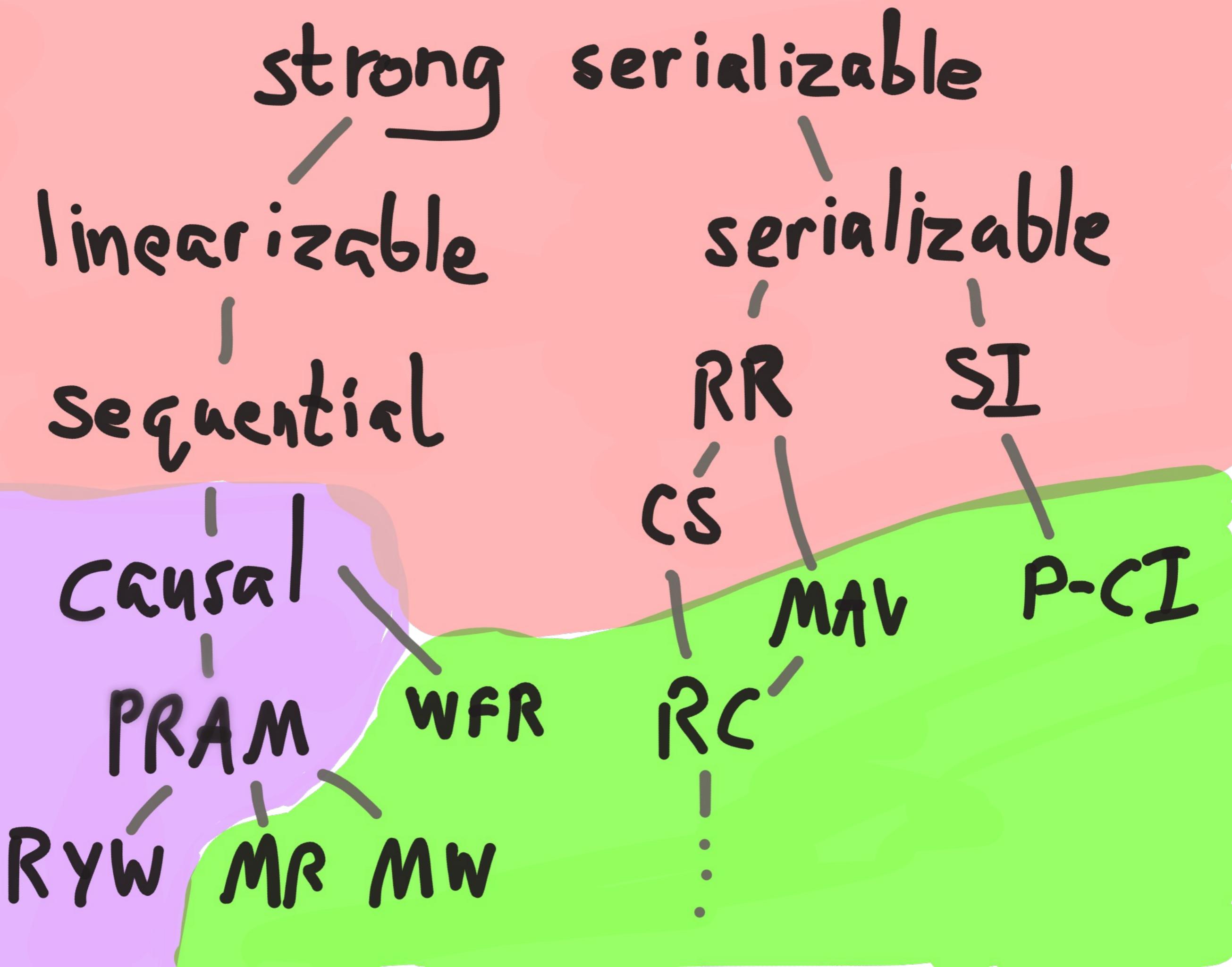
CS

P-CI

RC'

⋮

causal
PRAM
RYW MR MW
WFR



"weaker" consistency
models are more
available in failure

"Weaker" models

are less

intuitive

"weaker" models

need less coordination,
so they're faster!

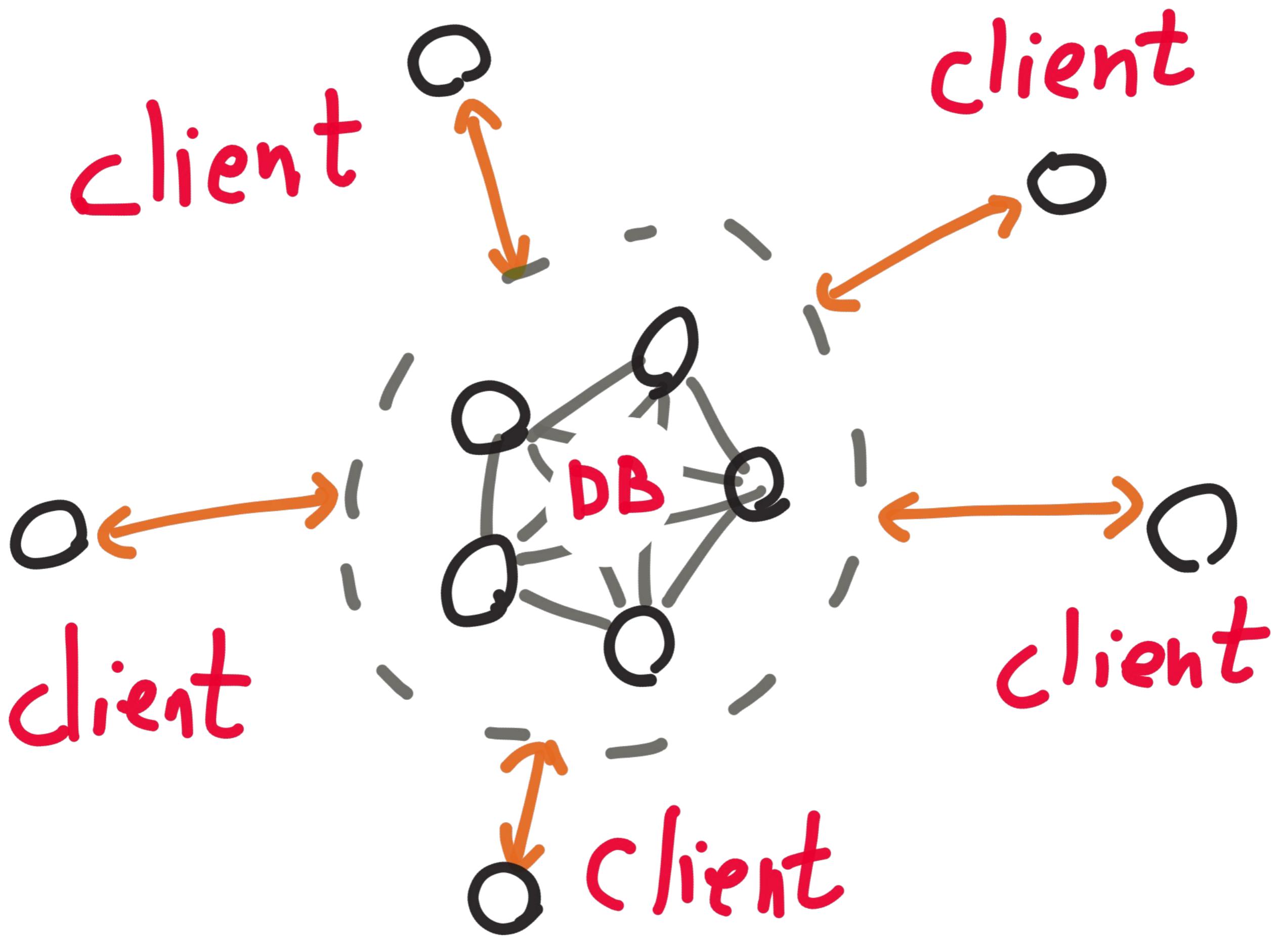
→ CPU memory model

"Weak" \neq unsafe

- Diff algos require
diff. levels of
consistency

SO: Consistency

models constrain
the system-environment
interactions.



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 segments are labeled $w-w'$, $r-r'$, and $w-w'$ from left to right. All three segments converge towards a central, blurred, cloud-like shape labeled "DB", representing the database system.

DB :

client : w — w' w — ...

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



invoke

ok

invoke

fail

invoke ?

? info

?

?

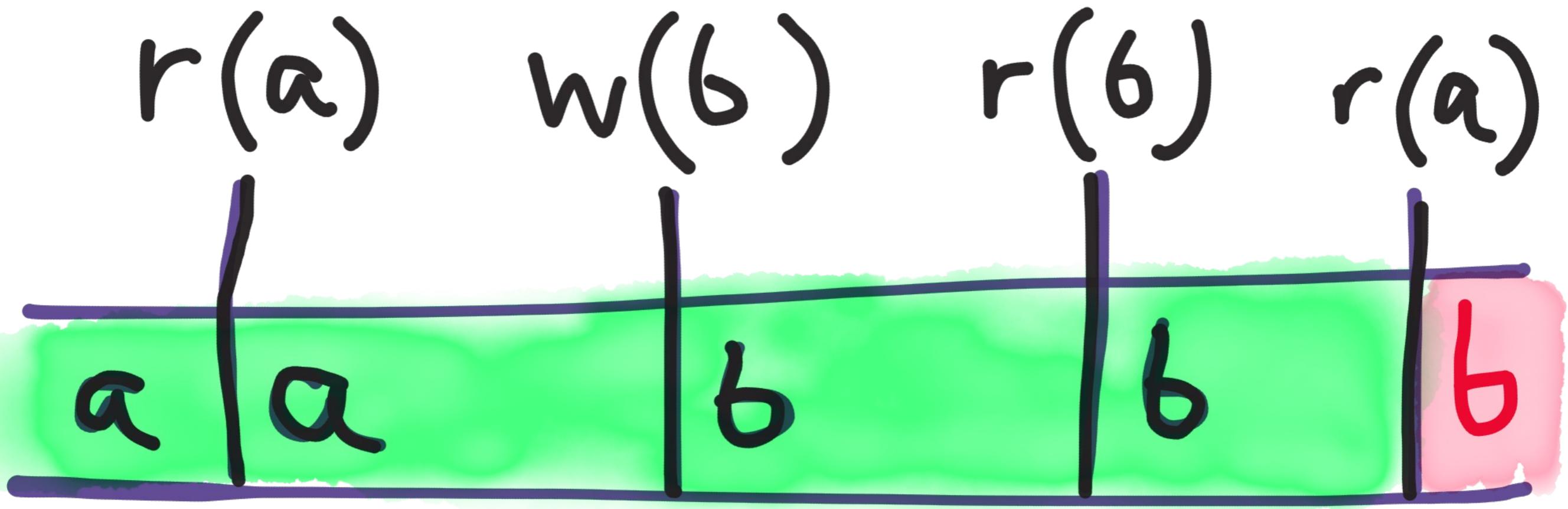
?

invoke ok

invoke fail

invoke ok

invoke info



Use a singlthreaded
model to verify path

invoke ok

invoke fail

invoke ok

invoke info

invoke ok

fail

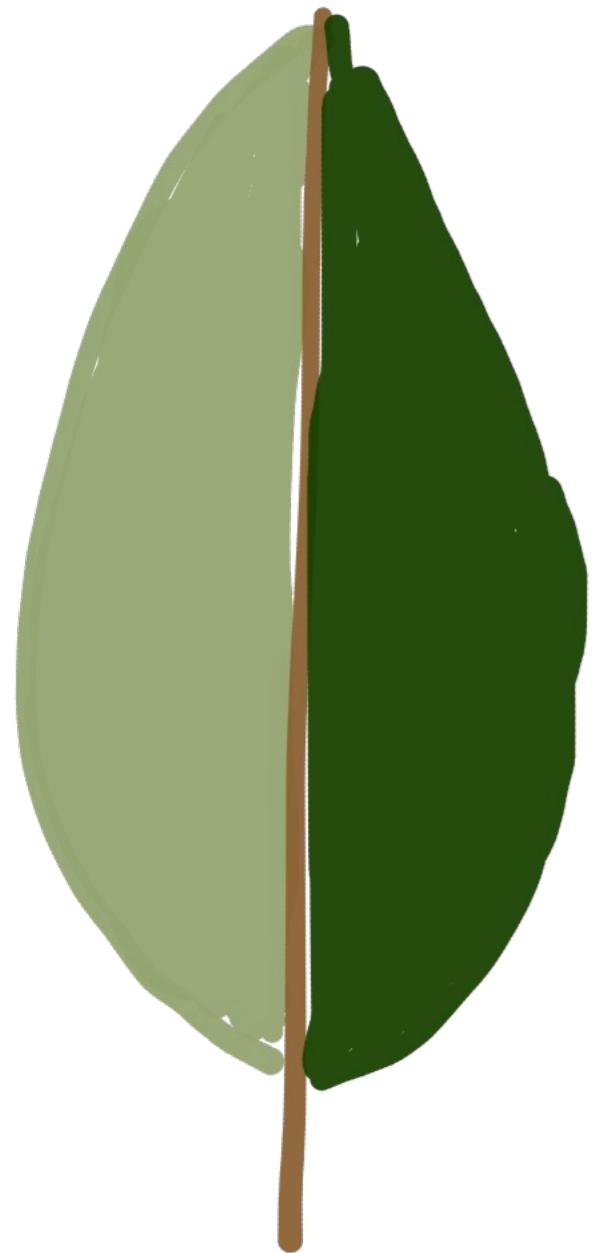
invoke ok

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

~~#~~

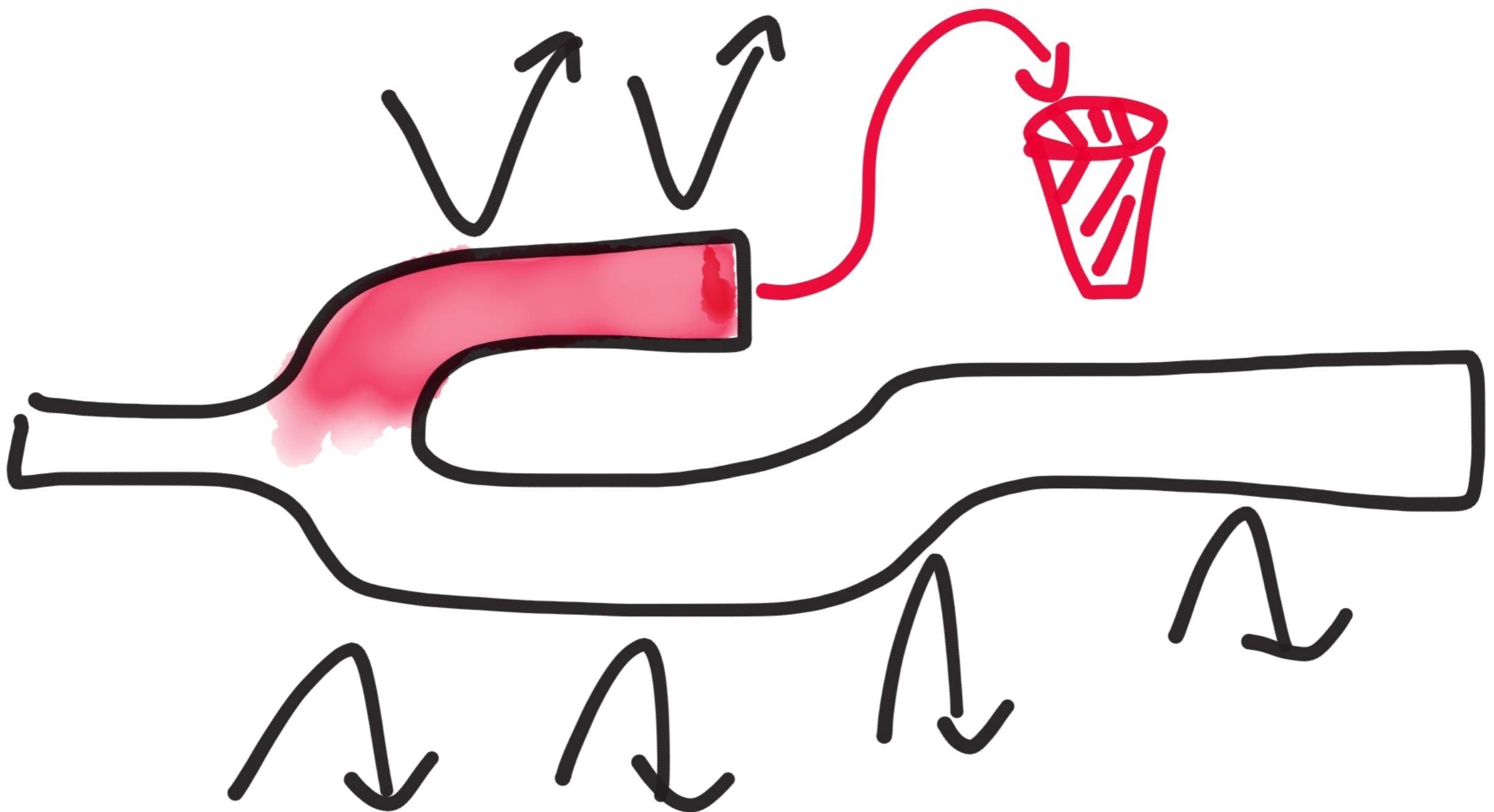
Data loss at all
WriteConcern levels

Unsafe defaults

- Some clients didn't 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 These bugs
are fixed now!



Ok! Let's

try 2.6.7

What kind of database is MongoDB?

MongoDB is a document-oriented DBMS. Think of MySQL but with JSON-like objects comprising the data model, rather than RDBMS tables. Significantly, MongoDB supports neither joins nor transactions. However, it features secondary indexes, an expressive query language, atomic writes on a per-document level, and fully-consistent reads.

Read Preference

Read preference describes how MongoDB clients route read operations to members of a replica set.

By default, an application directs its read operations to the primary member in a replica set. Reading from the primary guarantees that read operations reflect the latest version of a document. However, by distributing some or all reads to secondary members of the replica set, you can improve read throughput or reduce latency for an application that does not require fully up-to-date data.

IMPORTANT:

You must exercise care when specifying read preferences: modes other than `primary` can and will return stale data because the secondary queries will not include the most recent write operations to the replica set's primary.

Linearizable

CaS Registers

~~0~~ → write(1) → 1

$\emptyset \rightarrow \text{read}(\emptyset) \rightarrow \emptyset$

\emptyset $\rightarrow \text{read}(1) \rightarrow \boxed{\text{err}}$

0 → cas(0, 1) → 1

1 → cas(1, 2) → err

An Anomaly
Emerges!

47

CAS 0→4

0

1

r4

r3

CAS 3→1

2

r0

Write 4

3

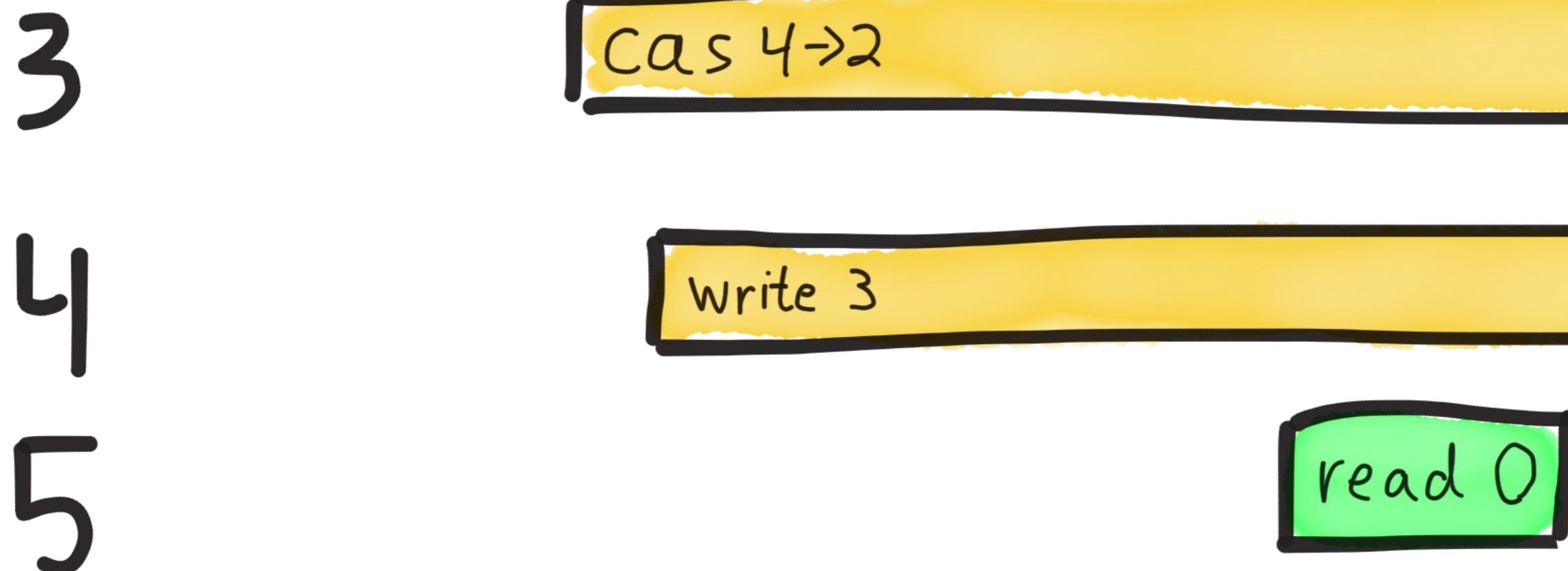
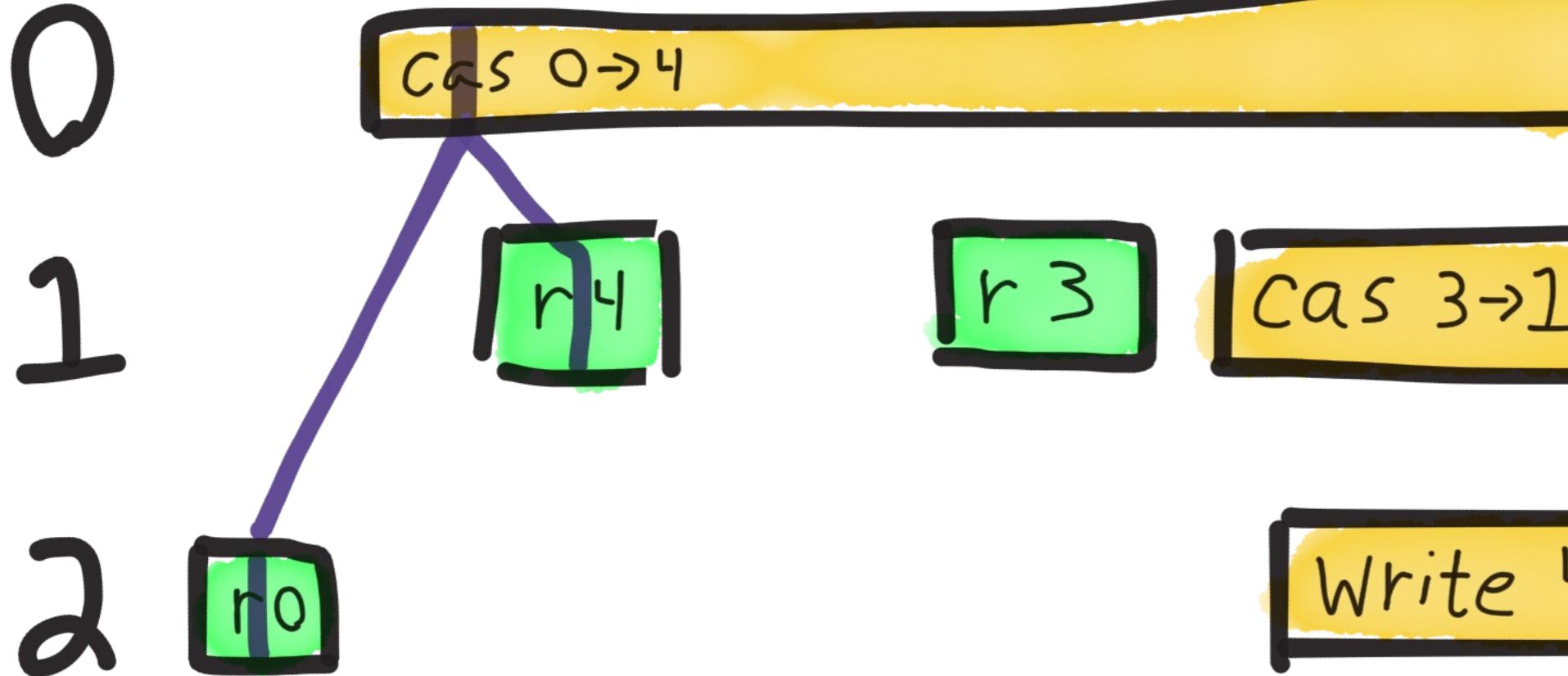
CAS 4→2

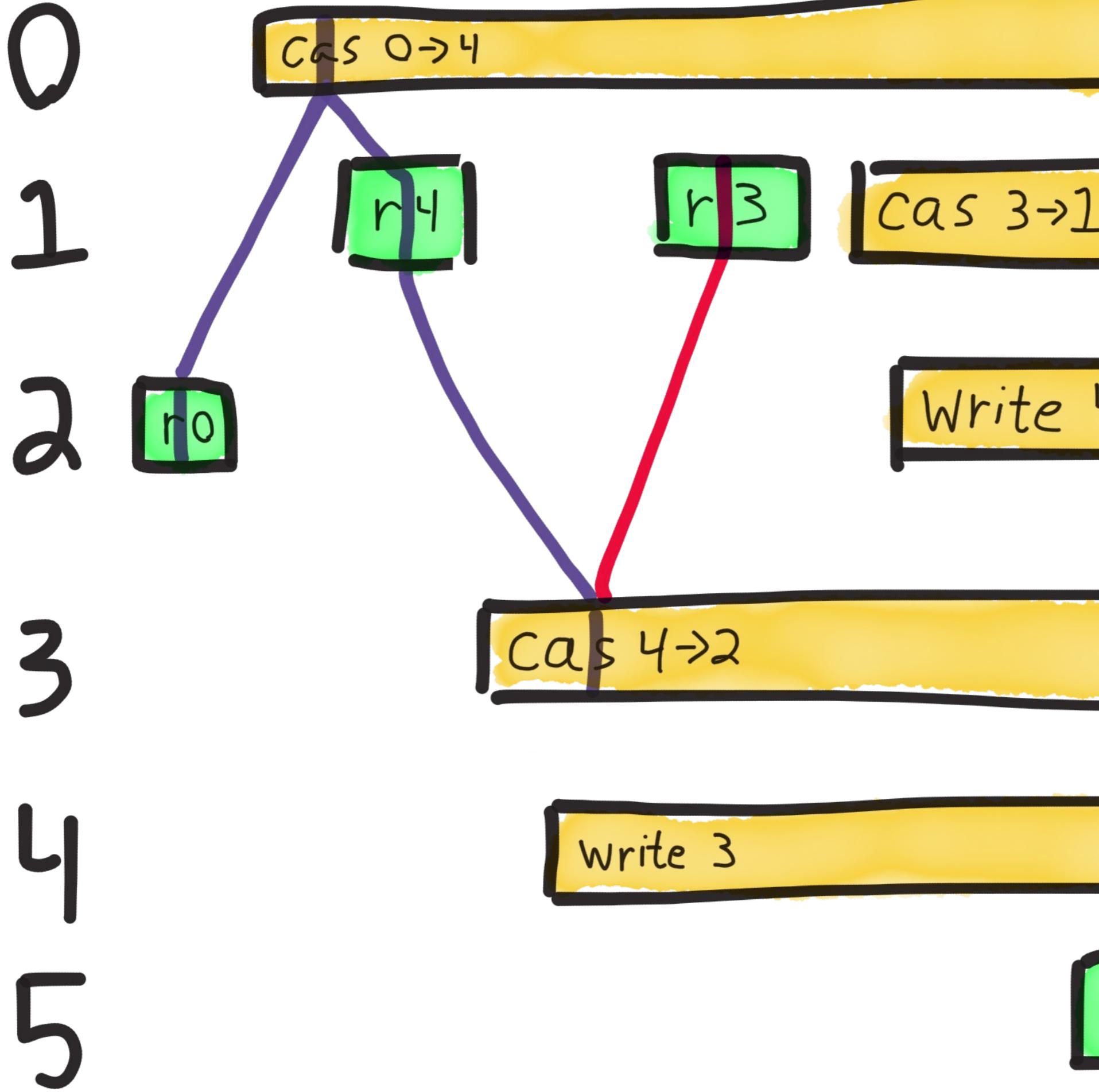
4

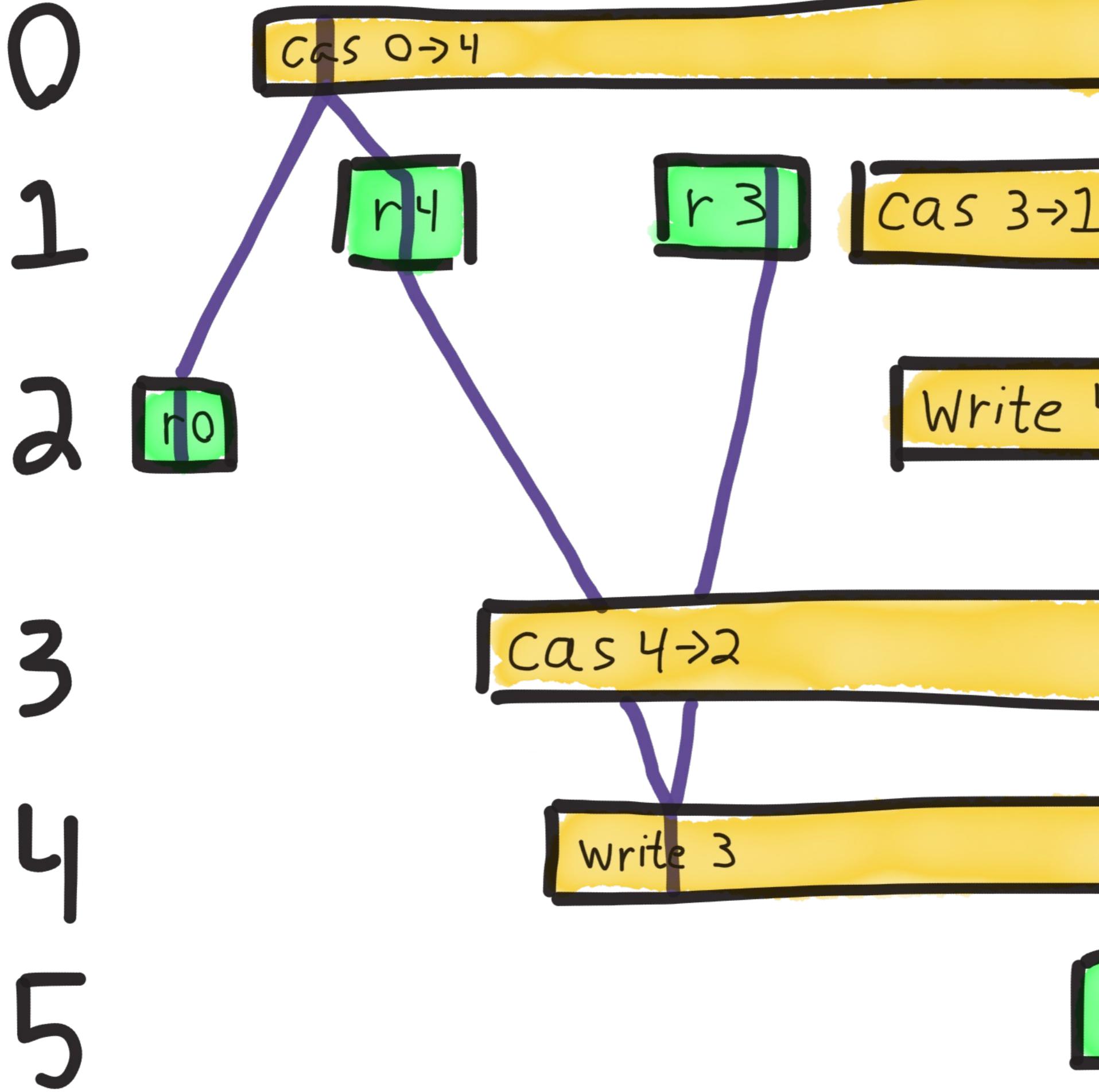
Write 3

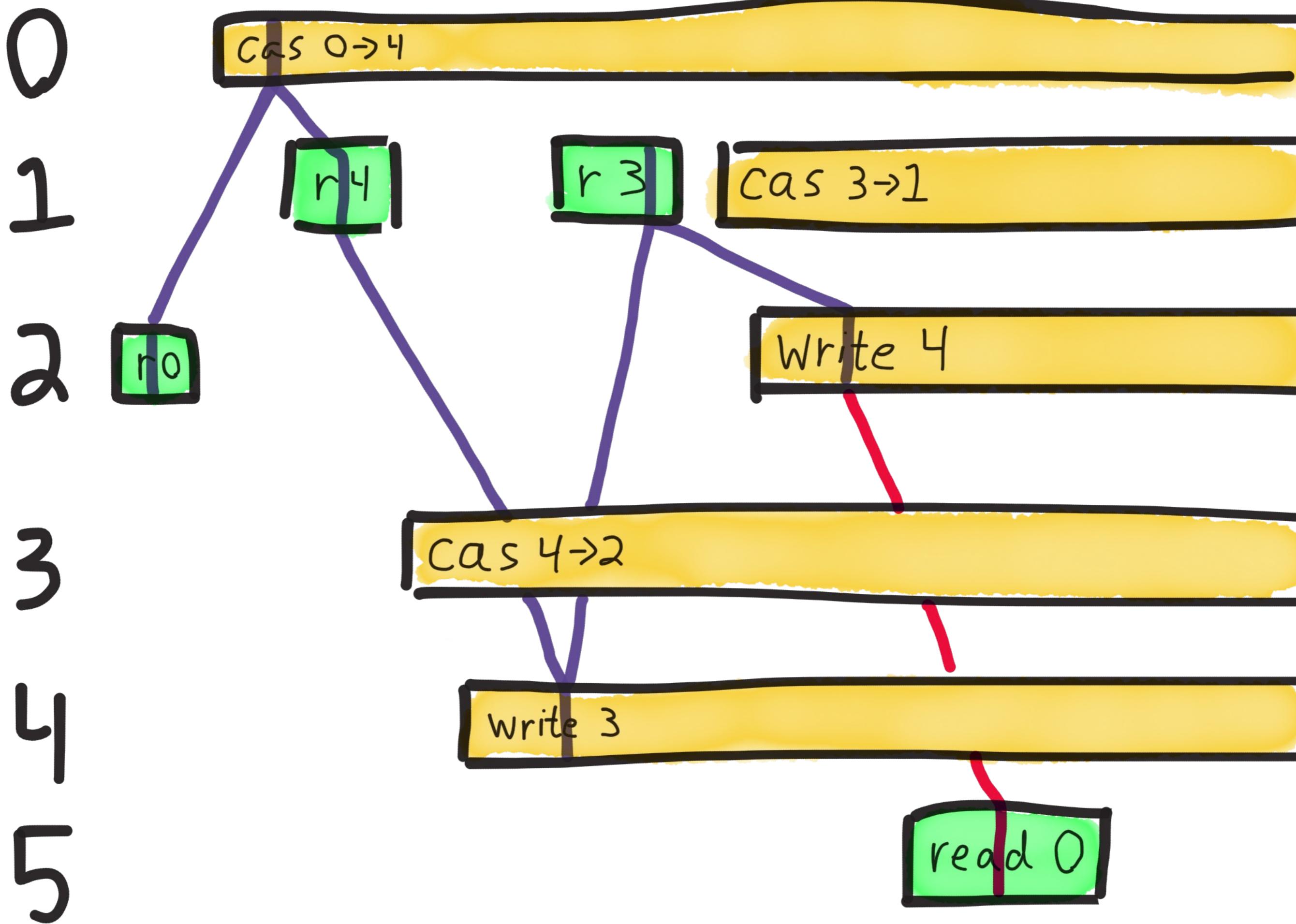
5

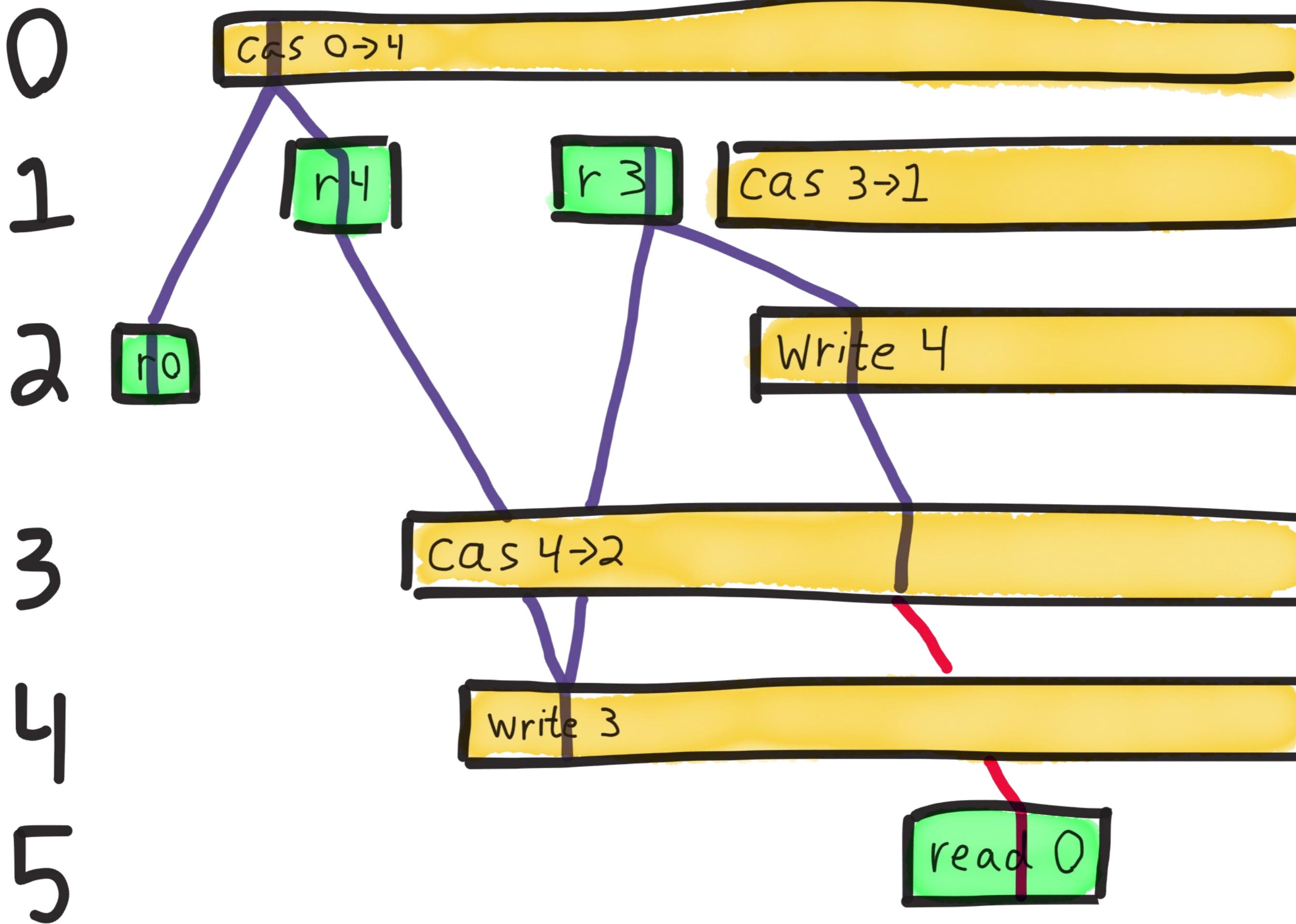
read 0

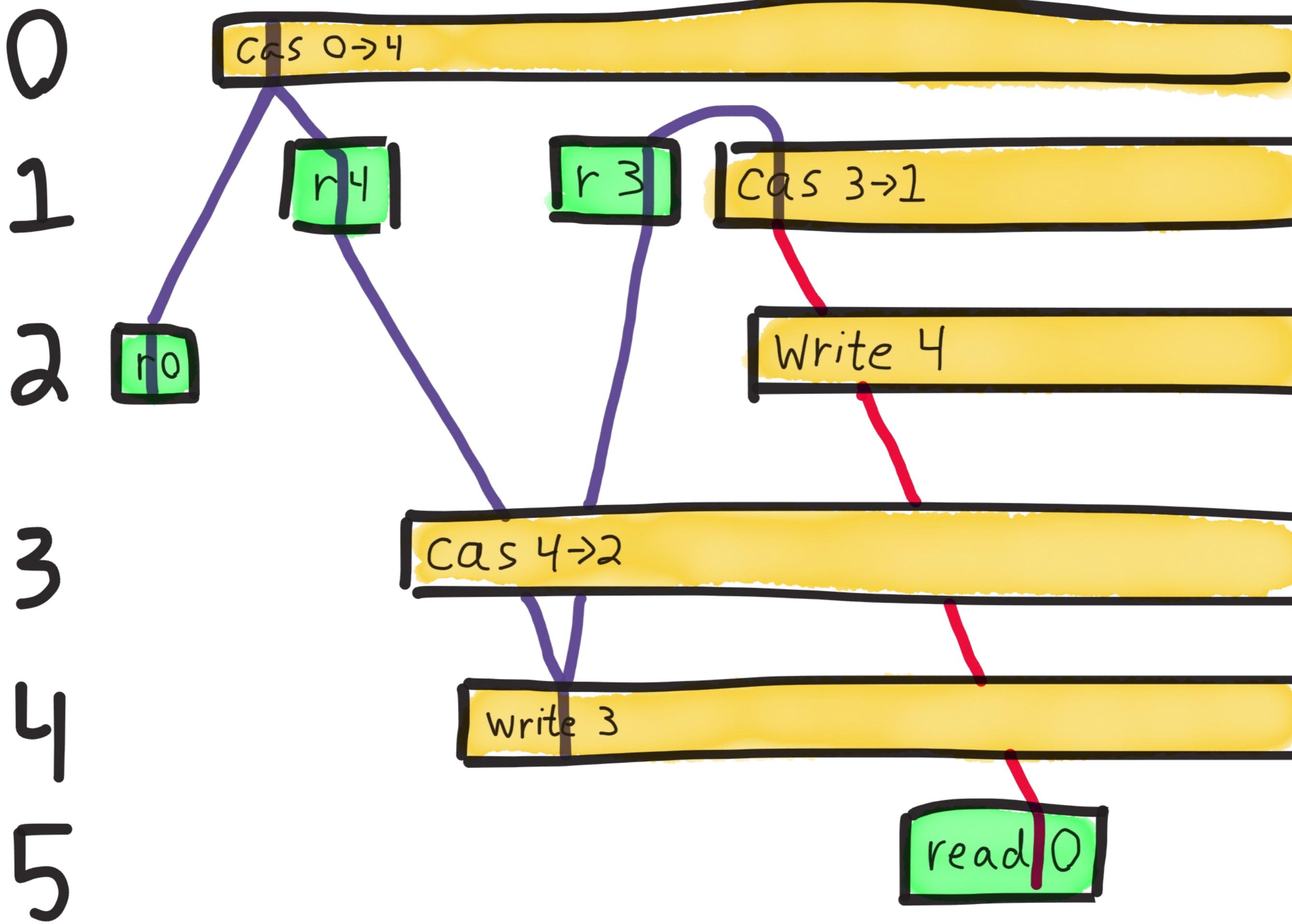


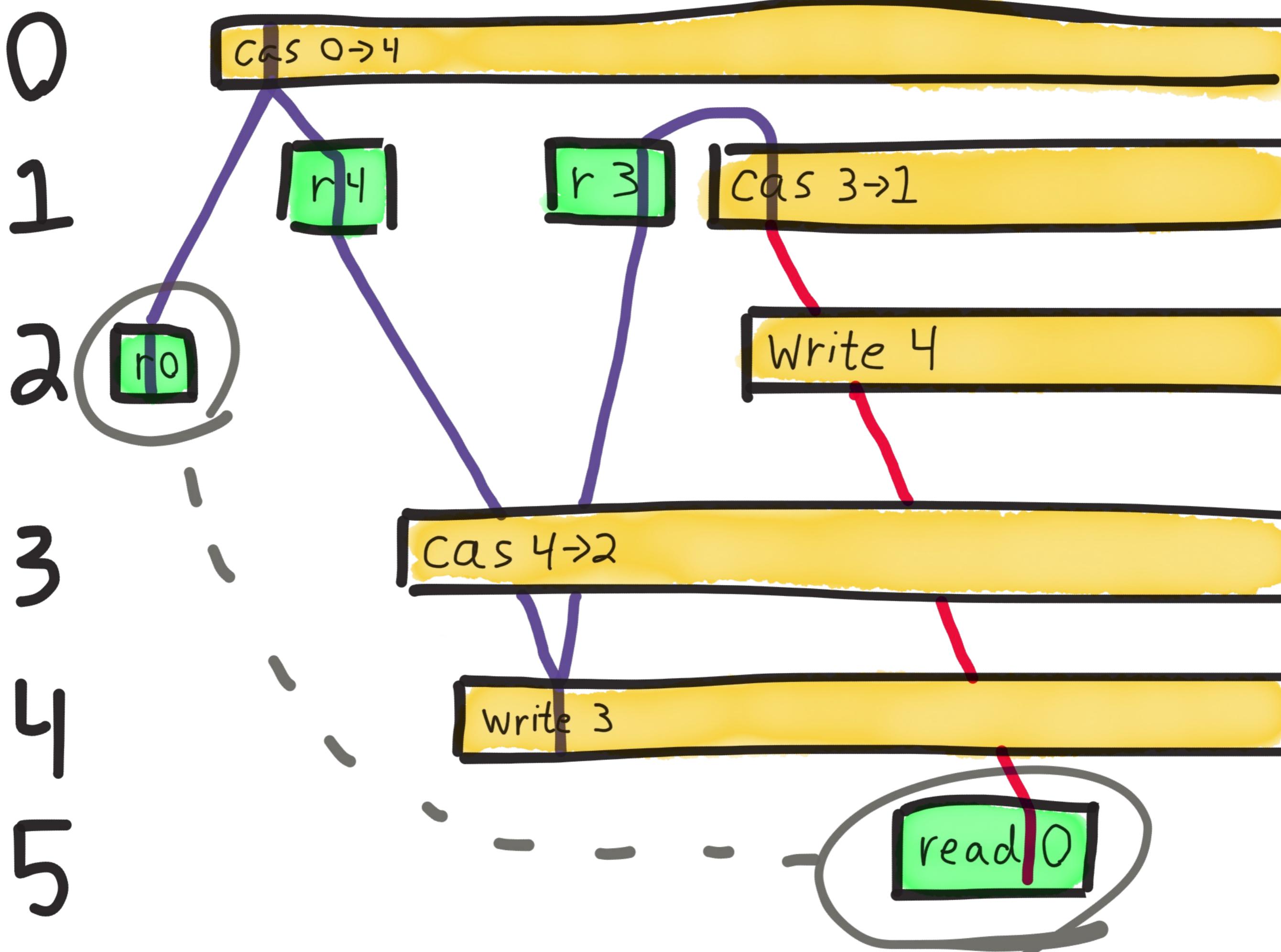




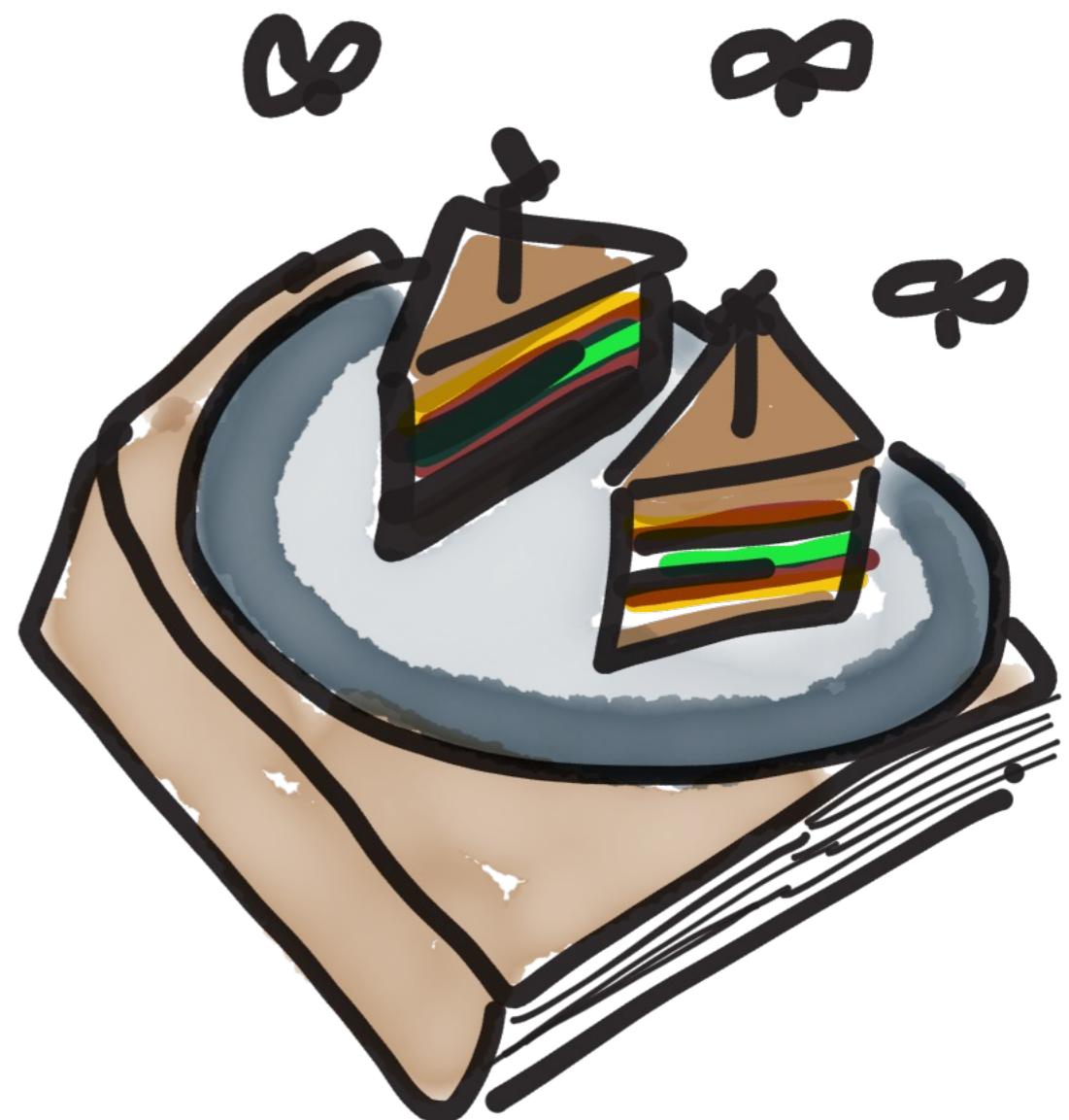








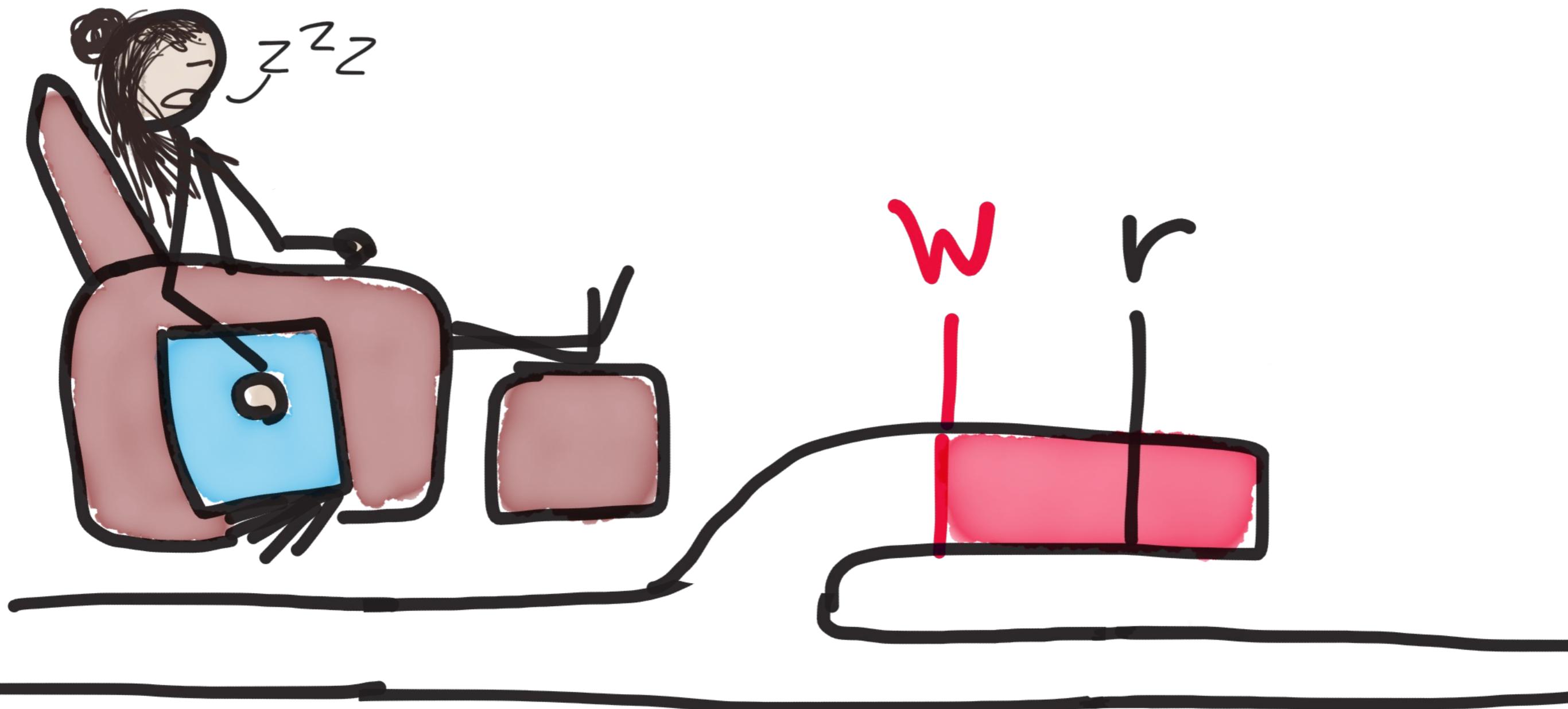
Stale
Reads



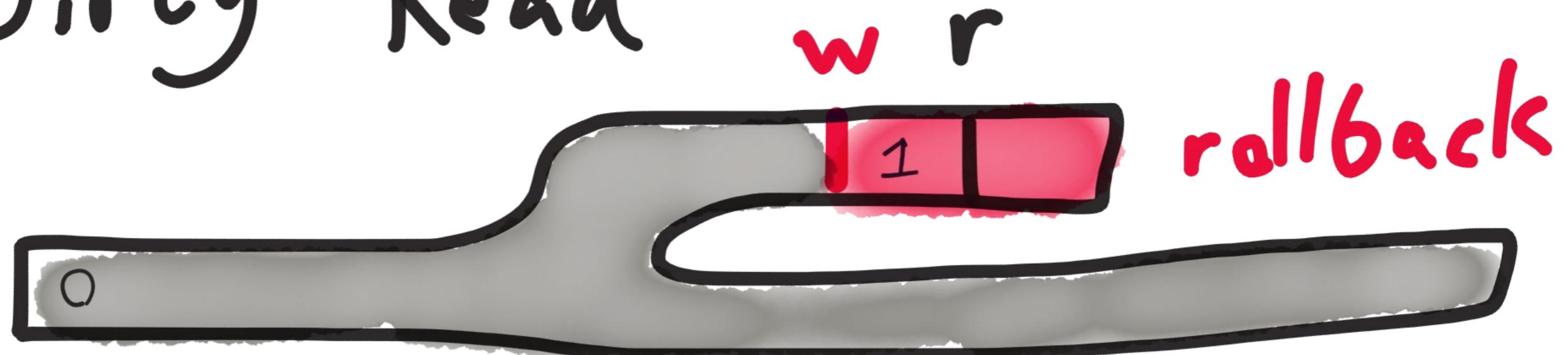
w

r

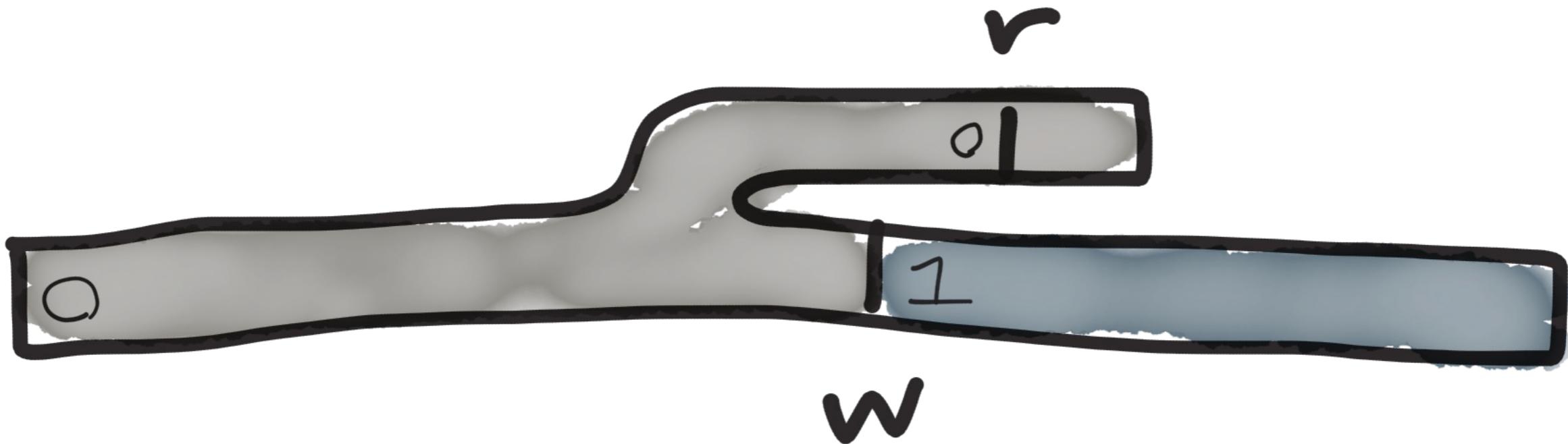
Read Uncommitted



Dirty Read



Stale Read

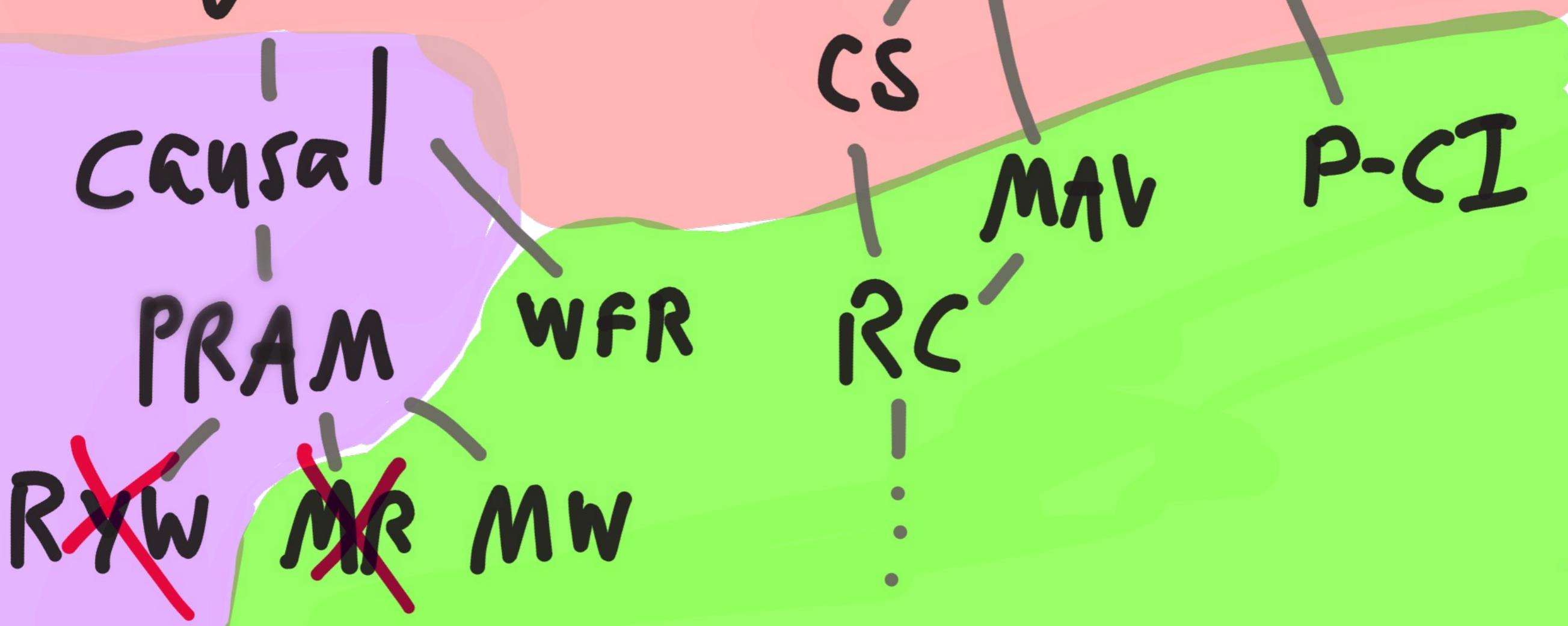


Server-17975

strong serializable

linearizable

sequential



~~strong~~ ~~serializable~~

~~linearizable~~

~~sequential~~

~~Causal~~

~~PRAM~~

~~RW~~ ~~MR~~ ~~MW~~

WFR

RR

CS

RC'

MAV

SI

P-CI

⋮

~~strong~~ ~~serializable~~

~~linearizable~~

~~sequential~~

~~Causal~~

~~PRAM~~

~~RW~~ ~~MR~~ ~~MW~~

~~serializable~~

~~RR~~

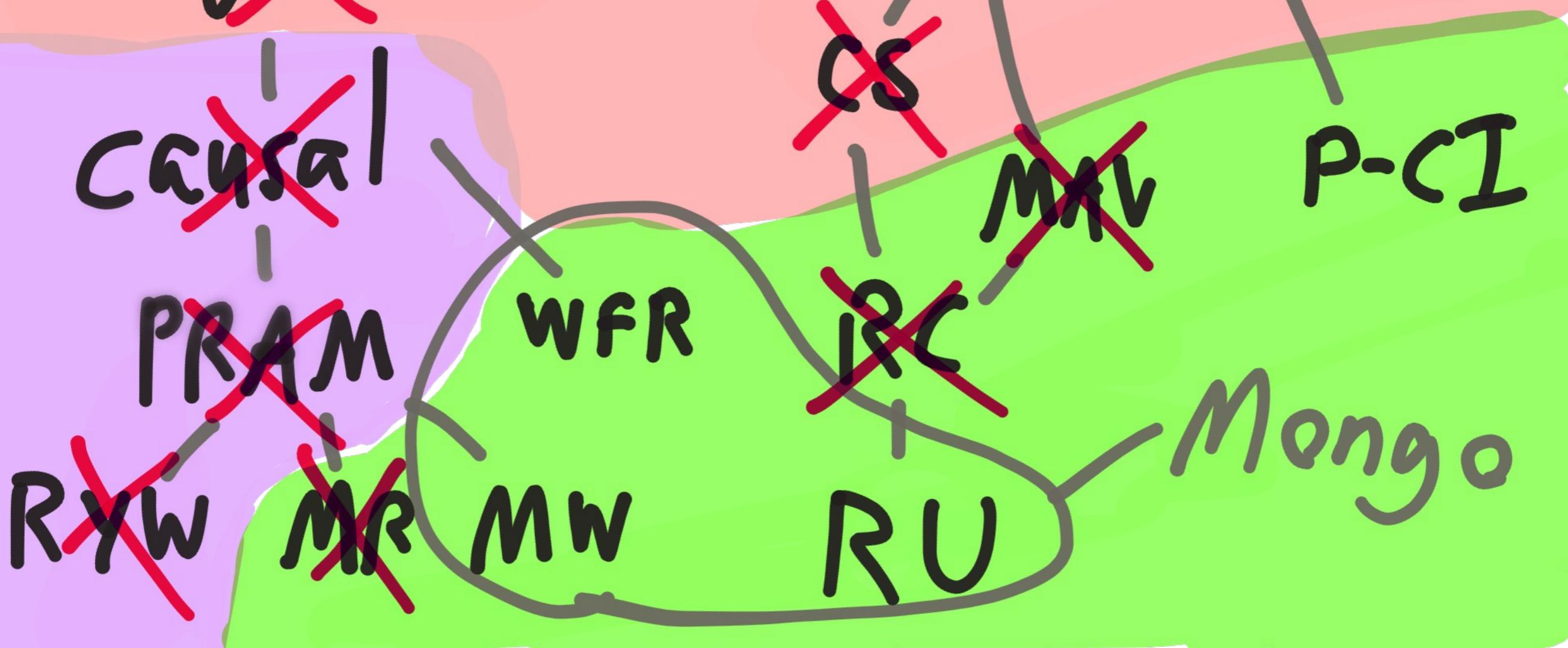
~~CS~~

~~MAV~~

~~RU~~

P-CI

~~strong~~ ~~serializable~~
~~linearizable~~
~~sequential~~



~~strong~~ ~~serializable~~

~~linearizable~~

~~sequential~~

~~Causal~~

~~PRAM~~

~~RXYW~~

~~MR~~

~~MW~~

WFR

~~serializable~~

~~RR~~

~~CS~~

~~MAV~~

~~RC~~

RU

P-CI

Mongo
vs. docs

Mongo Recommendations

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

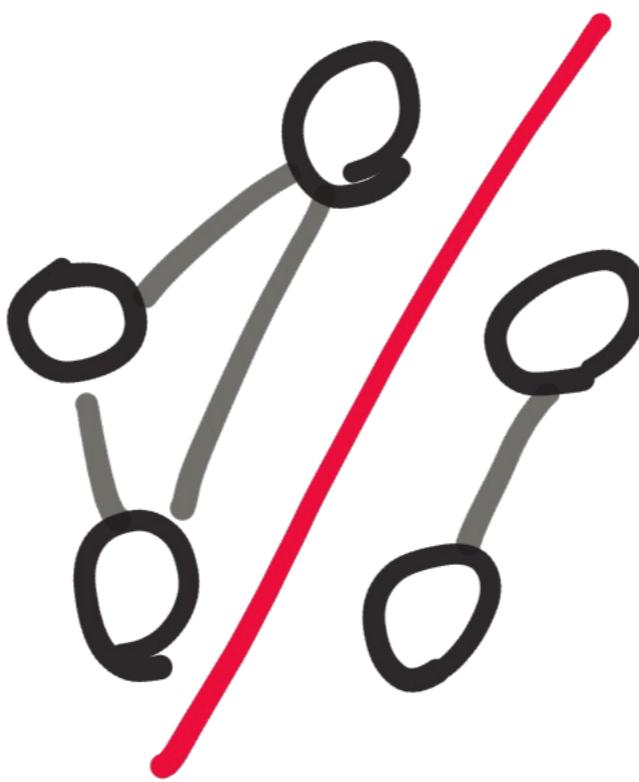


elasticsearch
[search]

First Elasticsearch
tests: June 2014

v 1.1.0

Split Brain from



Massive data loss: 90%.

ES added a
great page on
safety issues

Loss of documents during network partition (STATUS: ONGOING)



If a network partition separates a node from the master, there is some window of time before the node detects it. The length of the window is dependent on the type of the partition. This window is extremely small if a socket is broken. More adversarial partitions, for example, silently dropping requests without breaking the socket can take longer (up to 3x30s using current defaults).

If the node hosts a primary shard at the moment of partition, and ends up being isolated from the cluster (which could have resulted in split-brain before), some documents that are being indexed into the primary may be lost if they fail to reach one of the allocated replicas (due to the partition) and that replica is later promoted to primary by the master.

[#7572](#)

A test to replicate this condition was added in [#7493](#).



bleskes commented on Sep 1, 2014

Owner

I'm closing this issue, as it is solved, as specified, by the changes made in #7493. Of course, there is more work to be done and the effort continues.

Thx for all the input and discussion.



bleskes closed this on Sep 1, 2014



AeroNotix commented on Sep 1, 2014

@**bleskes** so it's 100% fixed?



bleskes commented on Sep 1, 2014

Owner

this issue (partial network splits causing split brain) is fixed now, yes.

Folks are
still referring
to the last talk



Mark Walkom

@warkolm



Follow

@falican durability in ES isn't a major problem these days, take a look at elasticsearch.org/guide/en/elast... for info on our view



9:36 PM - 20 Feb 2015



Reply to @warkolm @falican



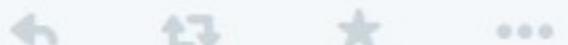
Rory Hart @falican · Feb 20

@warkolm yeah I've read through that, looks like you're all working very hard. I've got an abundance of caution today. Appreciate the time.



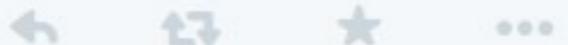
Mark Walkom @warkolm · Feb 20

@falican any time!



Mark Walkom @warkolm · Feb 20

@falican re-read that article, most of the complaints in it are pre 1.0 btw, given we're probably not far off 1.5, grains of salt needed ;)

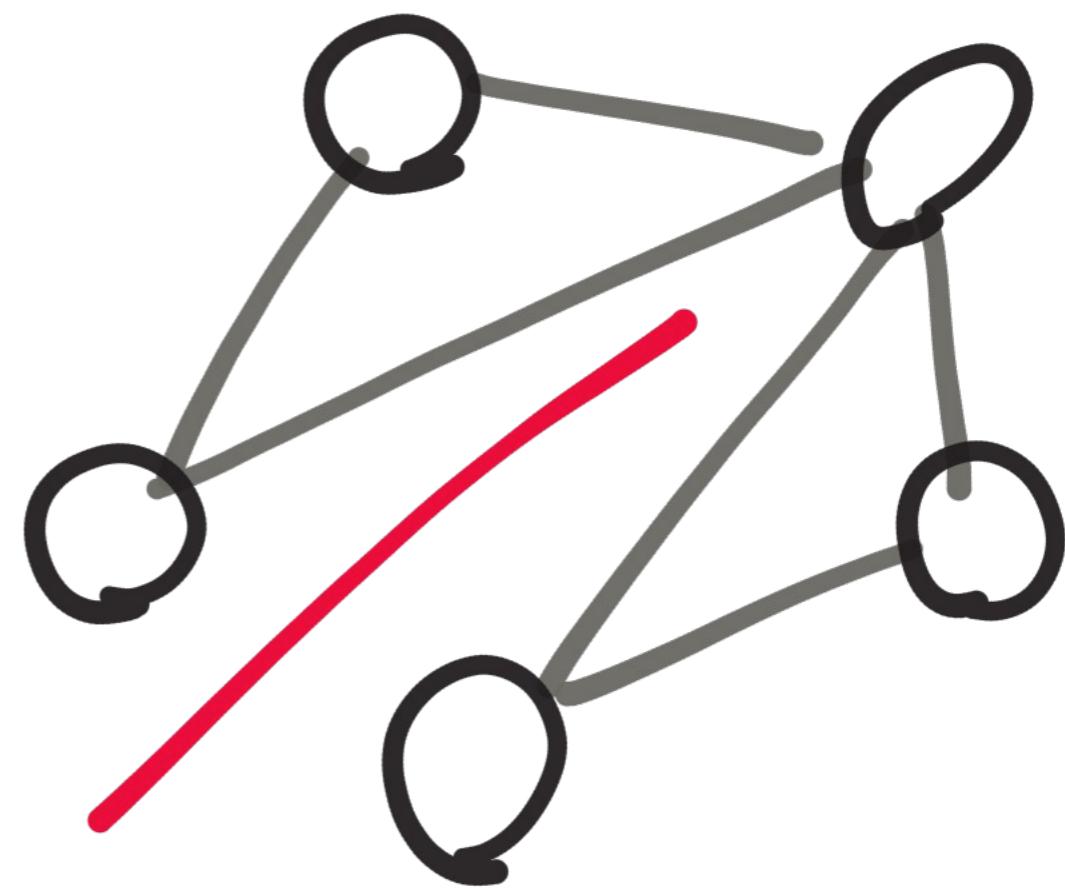


Retesting with

ES 1.50

Intersecting
partitions still cause

data loss



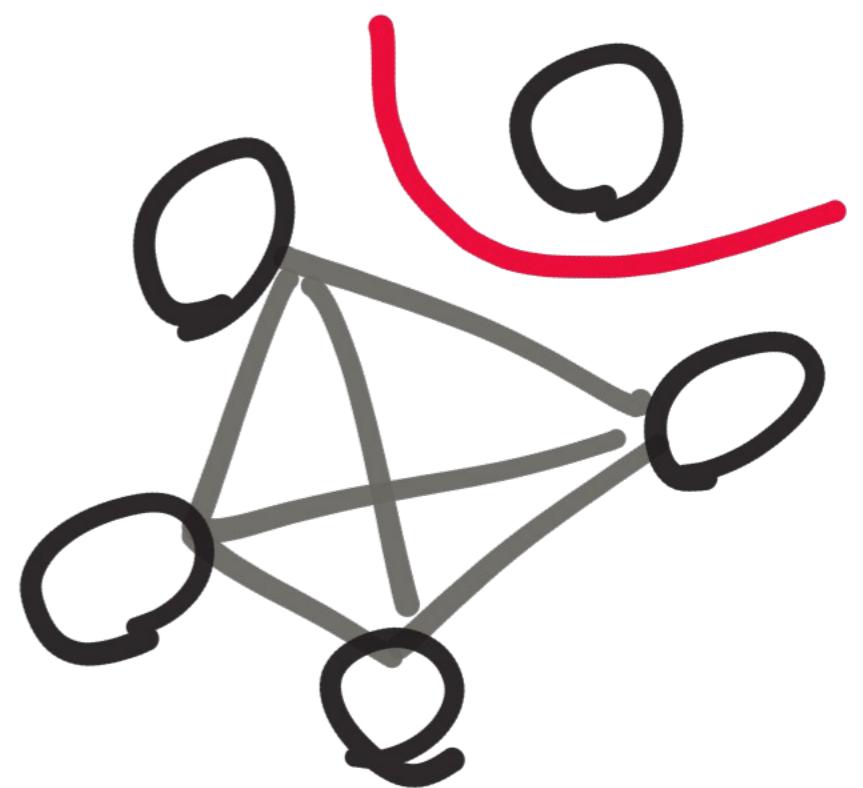
Long-lived split-

brain appears

fixed, gut docs

are still last

Single node
partitions still cause
data loss!



Electing a new
primary still takes

~90 seconds

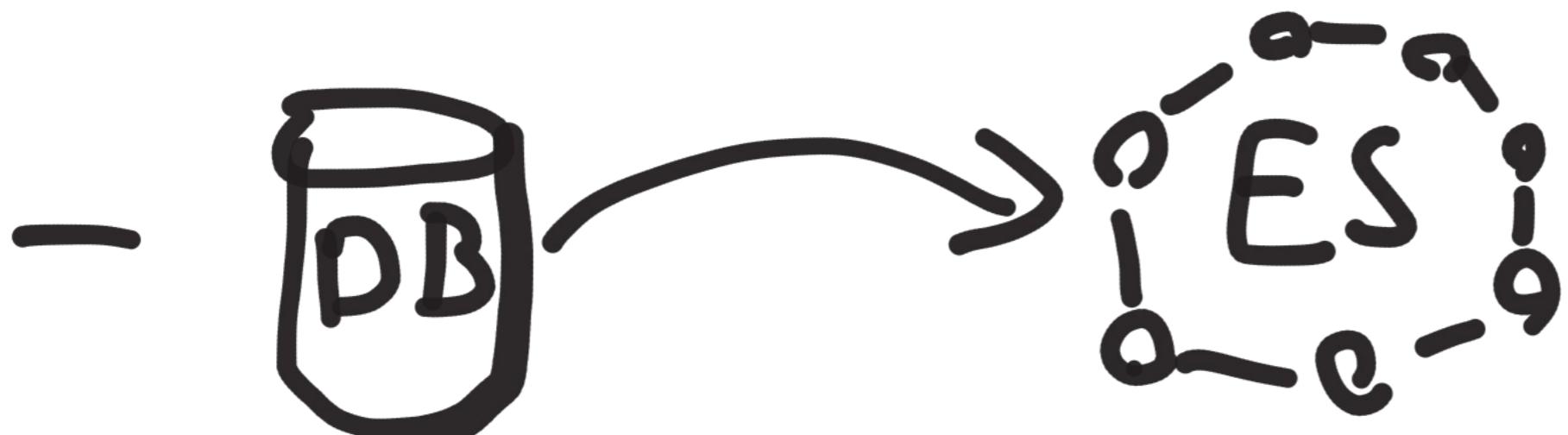
- Writers will stall

Elasticsearch is
doing better, esp.

with documentation !

ElasticSearch Recommendations

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





AEROSPIKE

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



RELIABILITY

100% Uptime
with strong consistency (ACID)

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

We have to go

Deeper

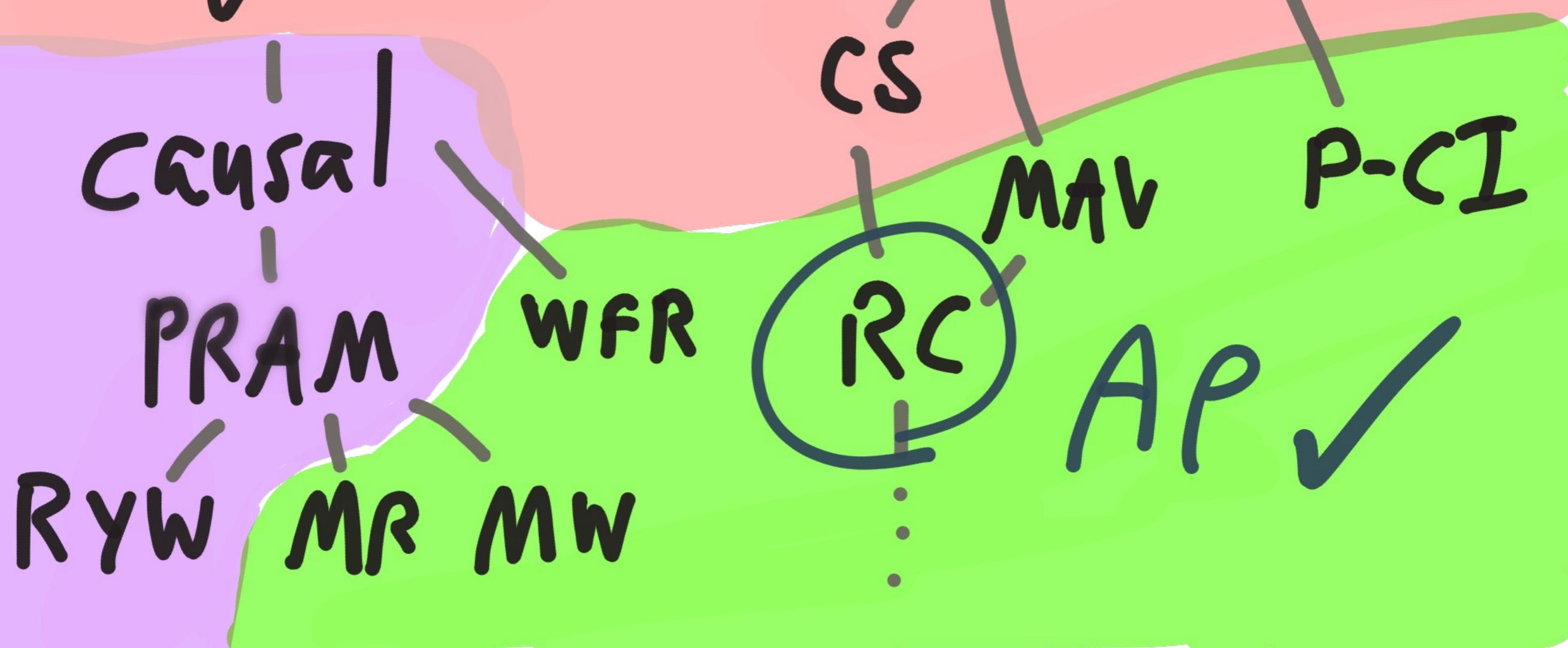
"AS is by and large an
AP system that provides
high consistency"

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

strong serializable

linearizable

sequential



"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

serializable

linearizable

sequential

causal

PRAM

RYW MR MW

serializable

RR

CS

SI

WFR

RC'

MAV

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"

[DISCOVER](#)[DOWNLOAD](#)[DEVELOP](#)[DEPLOY](#)[DOCUMENTATION](#)[COMMUNITY](#)[Get Started!](#)

DEPLOY

[TRAINING](#)

RESOURCES



[Aerospike is now available with Click-to-Deploy on Google Compute Engine](#)

You can quickly spin up an Aerospike cluster for development or production.



[Deploy Aerospike clusters easily in Amazon EC2](#)

Learn how to install on EC2 instances, get tips on capacity planning, configuration, benchmarking and using bcache for faster persistent storage.

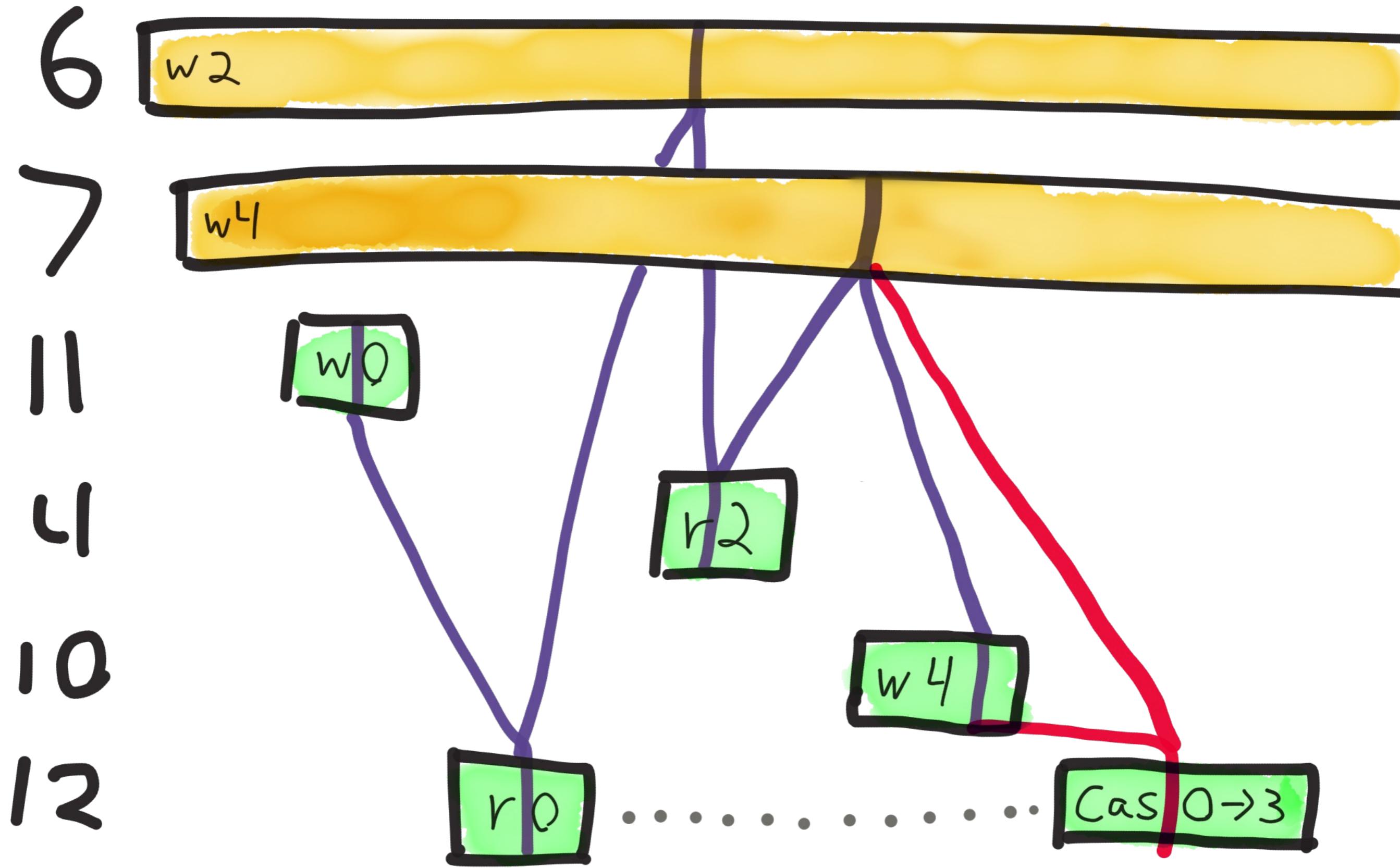


[1M TPS on 1 Server](#)

This can be achieved on a \$5k commodity server. Don't believe it? Follow this easy, 4 step recipe using YCSB or watch the recorded demo (3 min).

So for a CAS

Register...



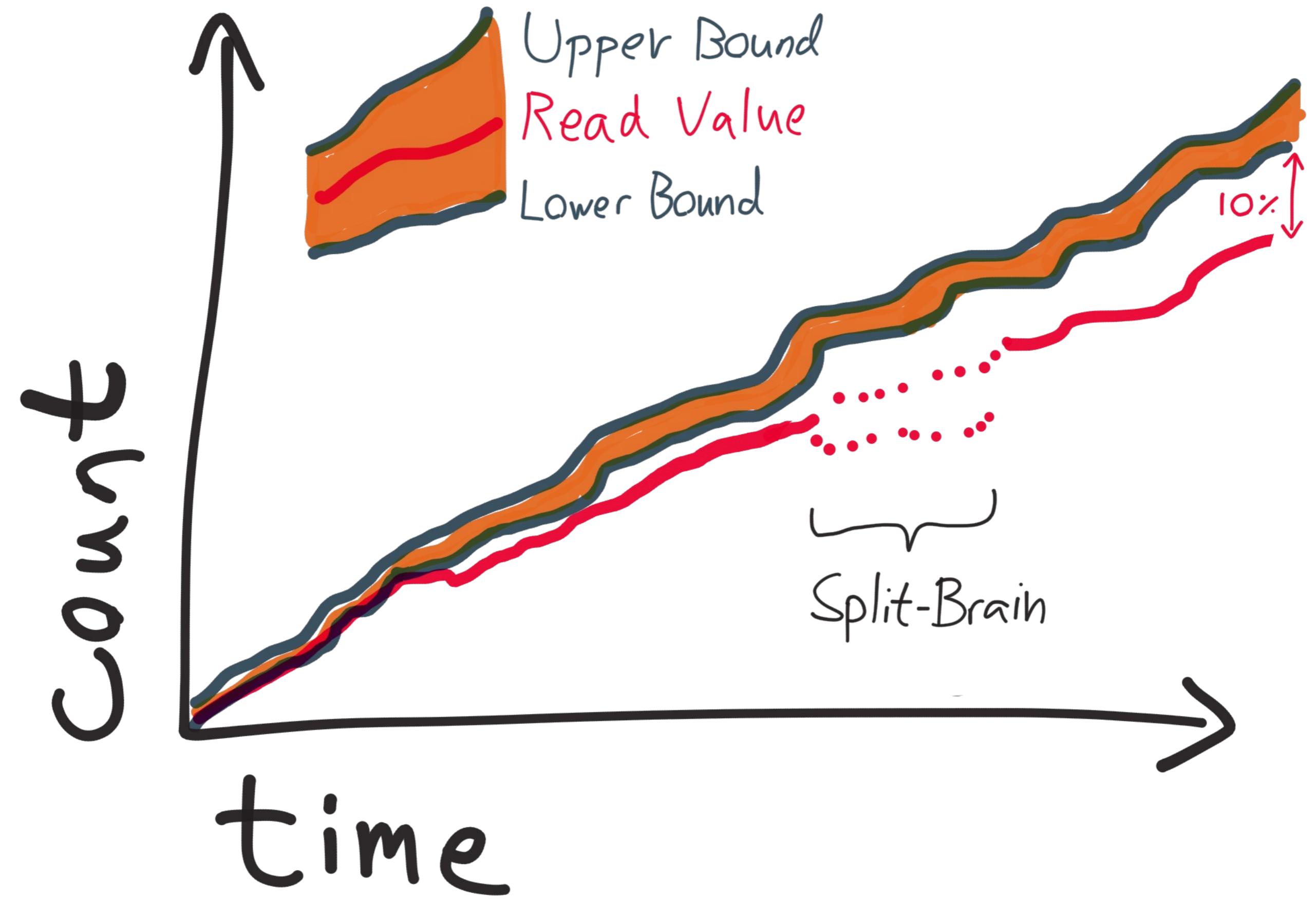
What about
Commutative ops?

$$a+b = b+a$$

Counters will

drop increments

during a partition



"Deployment modes
AP vs. CP"



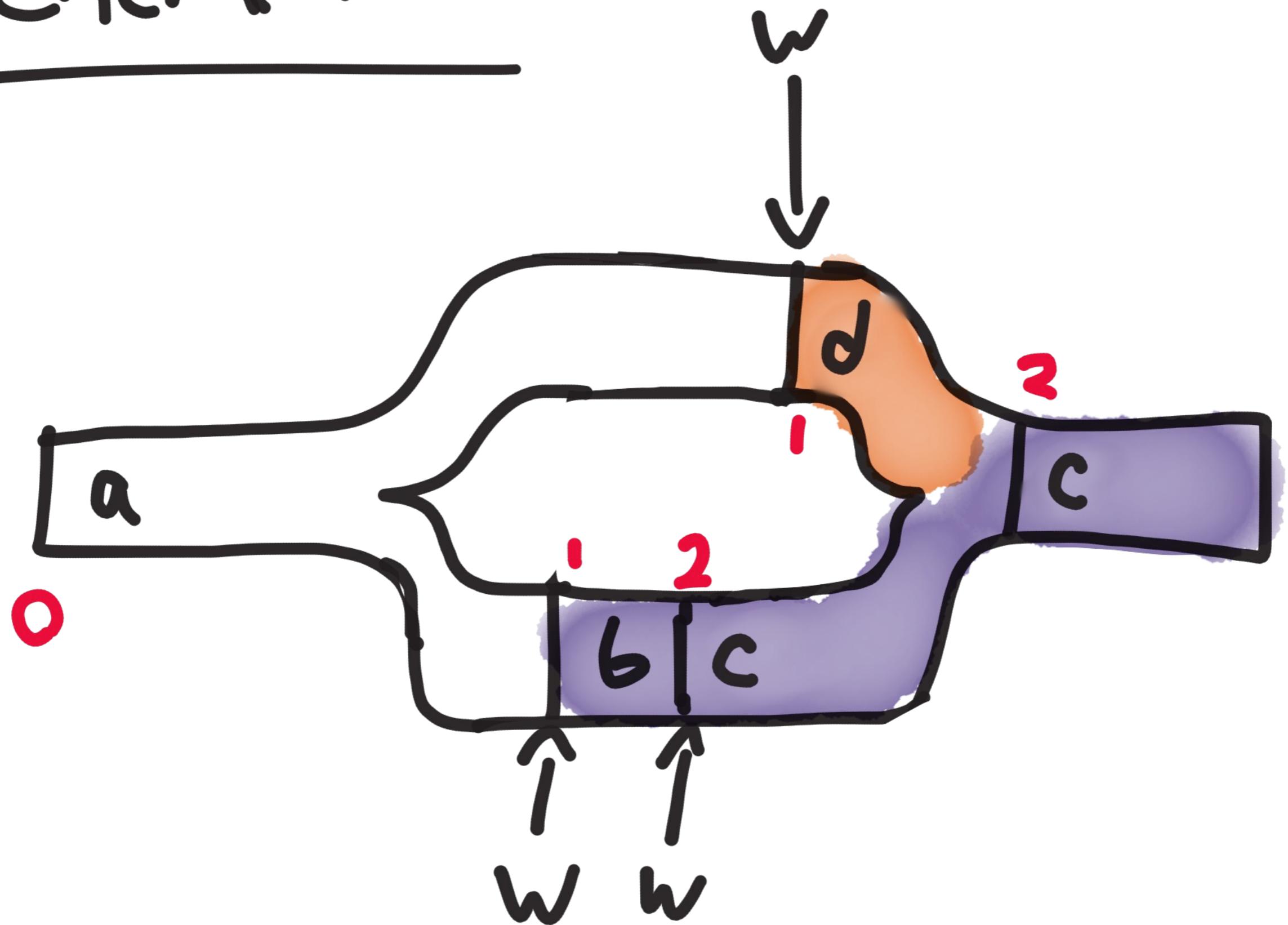
CP is vaporware

OK so how

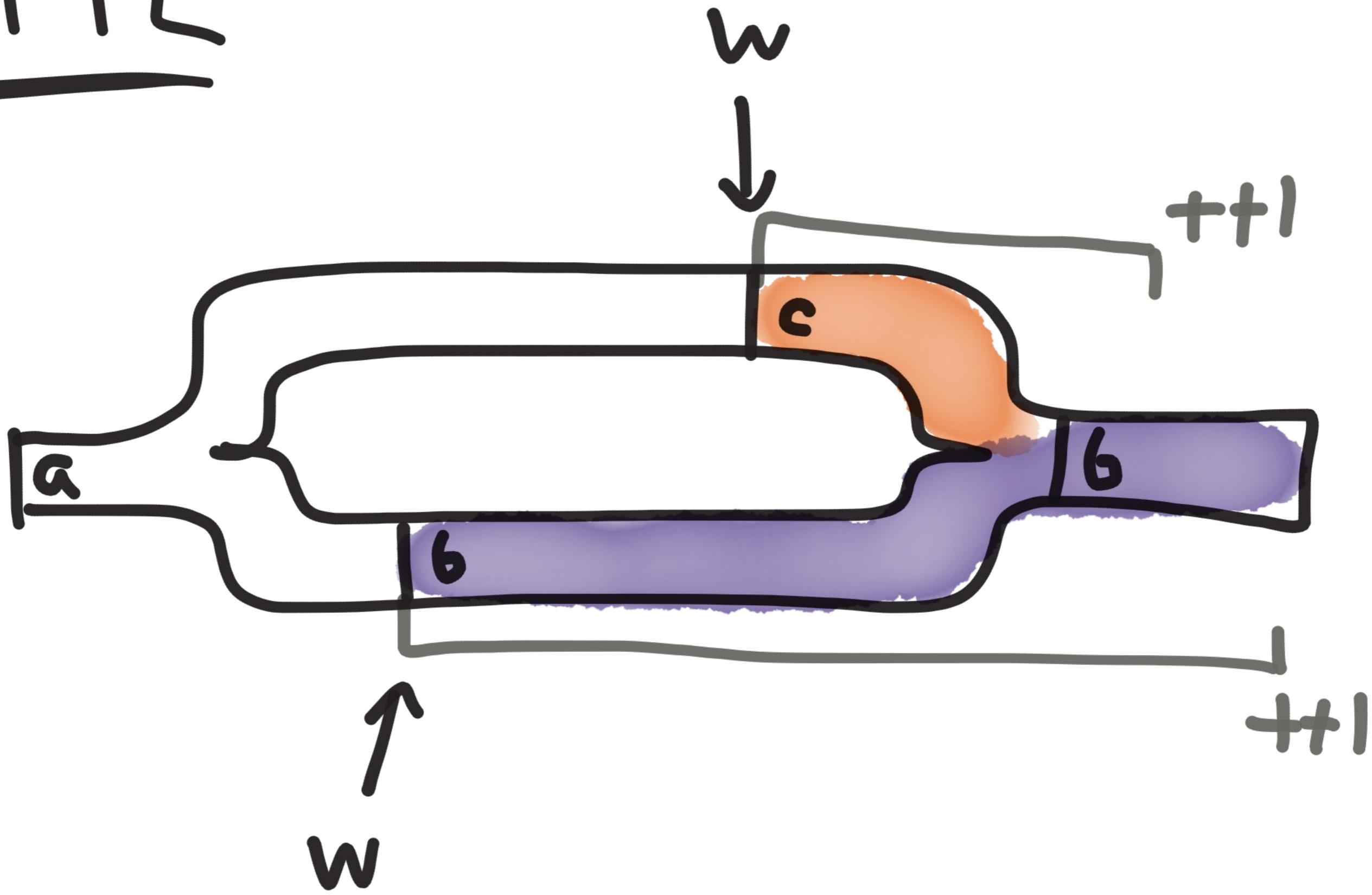
does AP mode

work?

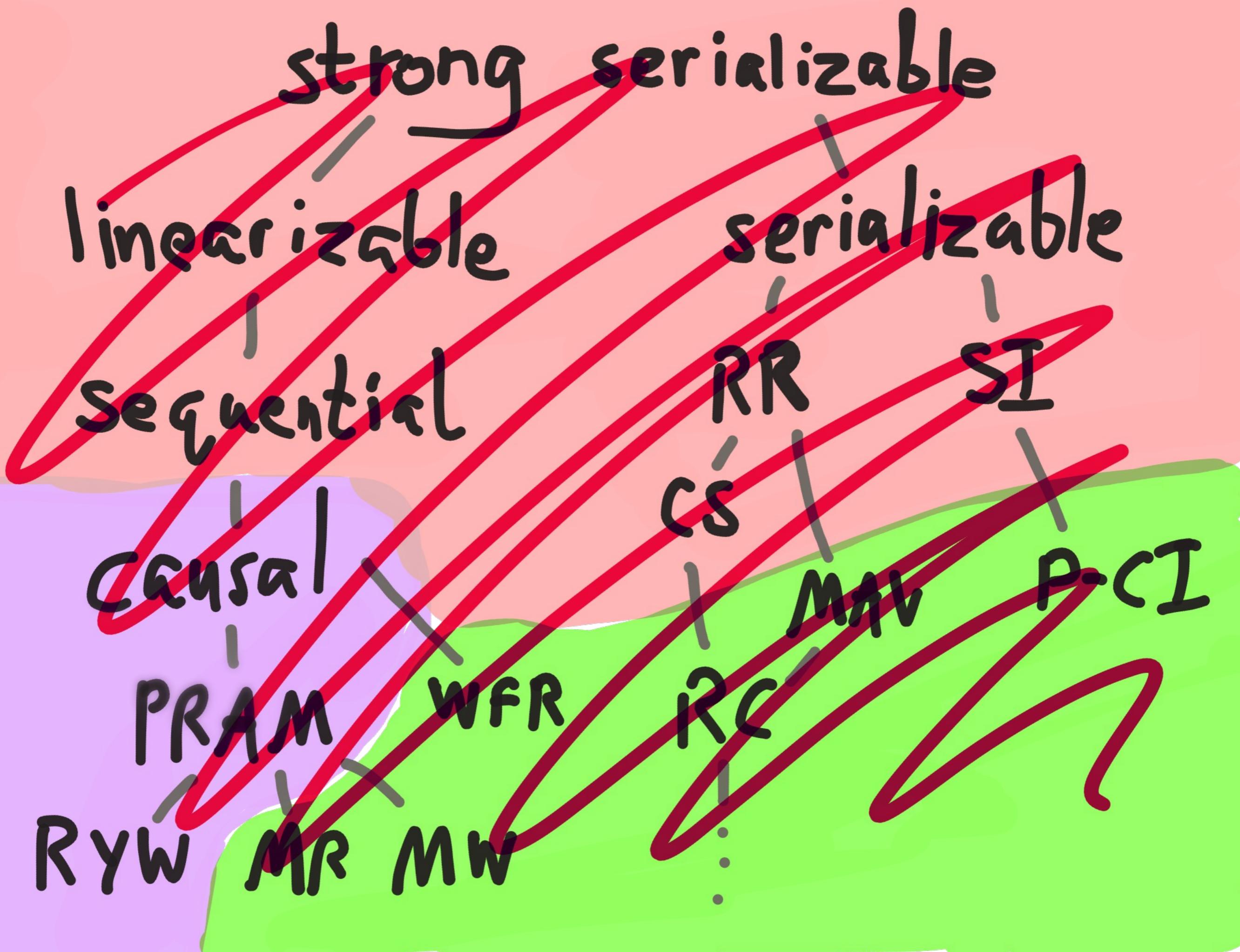
Generations



TTL



UPDATES



"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 & EJ

have made

improvements!



Still Surprises

Lurking in the



Read the docs!



Then
MEASURE
— far —
YOURSELF!



*Figure out the
invariants
your system needs*

How much



can you tolerate?

Consider
your
failure modes

Partitions

I/O/GC pauses

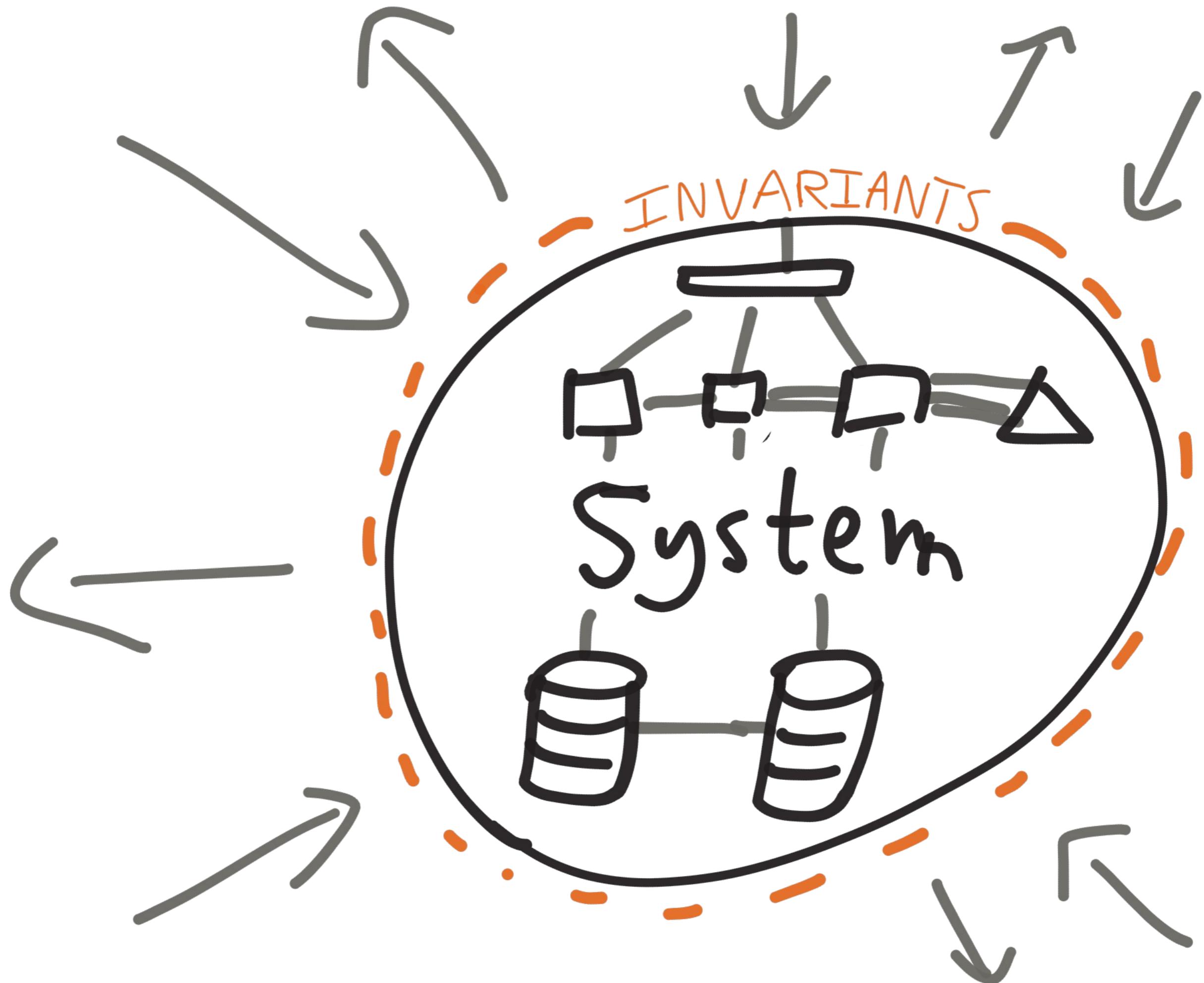
Node failure

Clock skew

Test your

Systems

end-to-end



Thanks!

Kevin Porter

Lucien Valmar

Boaz Leskes

Lee Hinman

Matt Kangas

Stripe

Evan Broder

Asya Kamsky

Kelly Stirman

Marc Hedlund