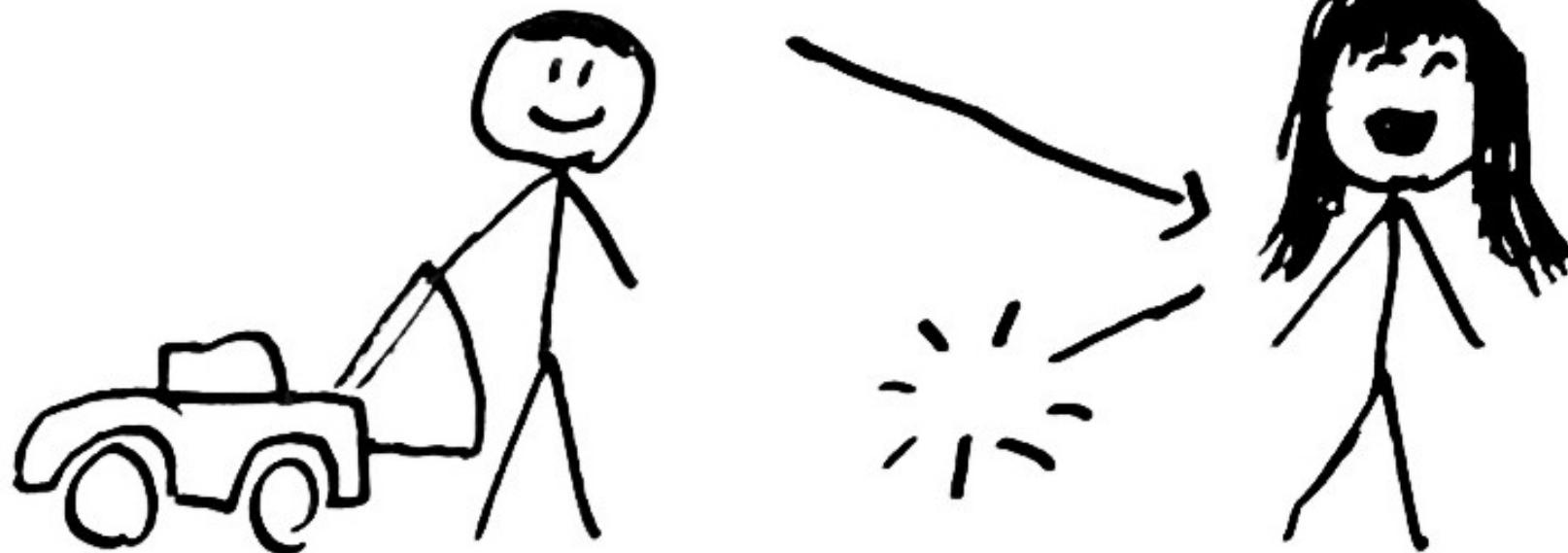


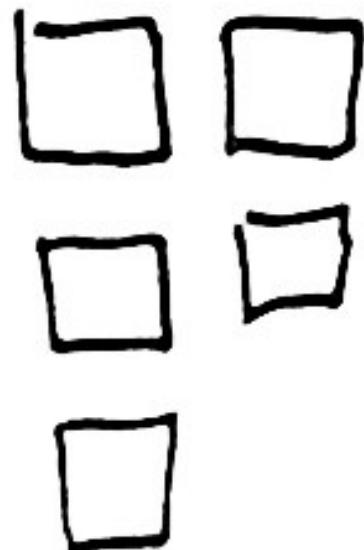
Call Me Maybe!



H

@aphyr
Kyle Kingsbury
"Jepsen Networks"

Backend engineer at



Factual

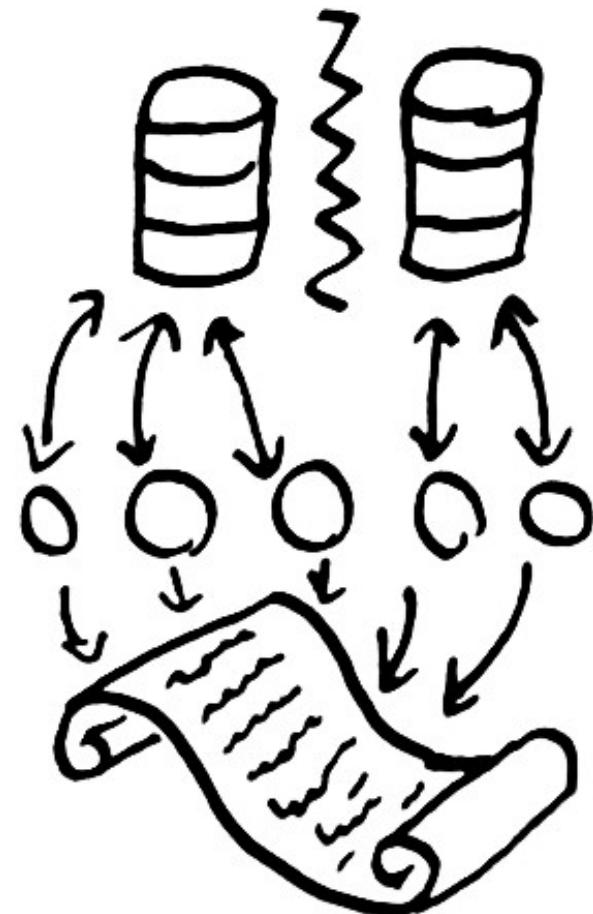
Jepsen II

Linearizable

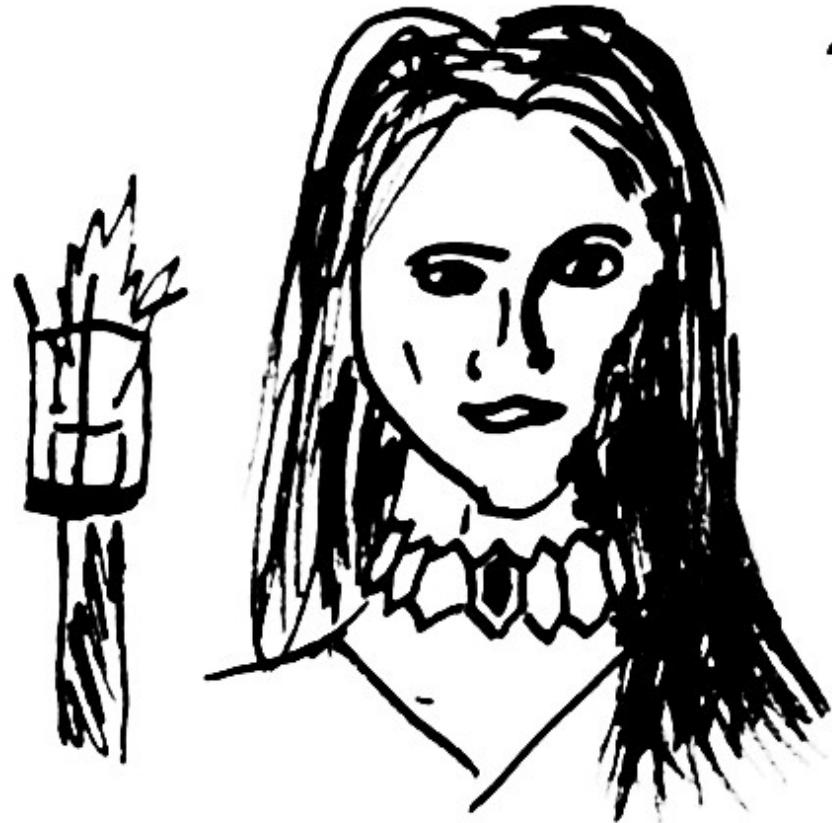
Boogaloo

Jepsen

Measures
Systems
Under
Stress

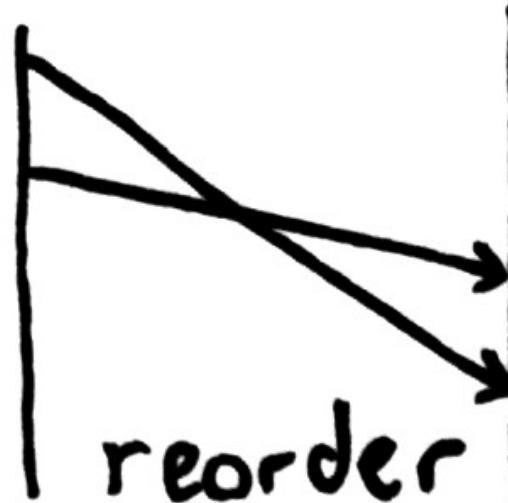
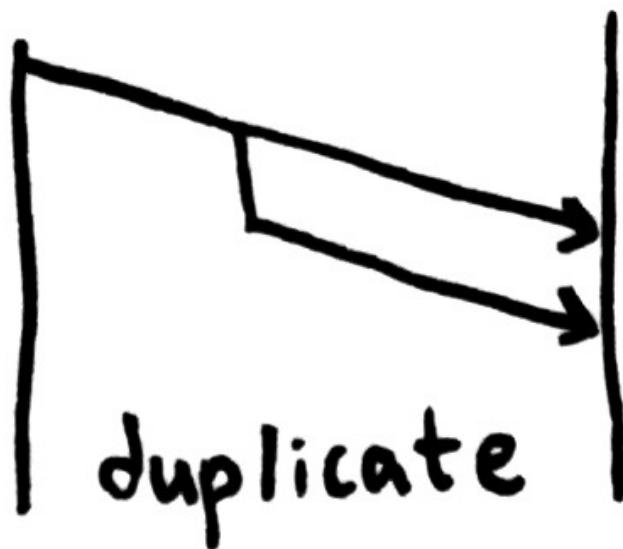
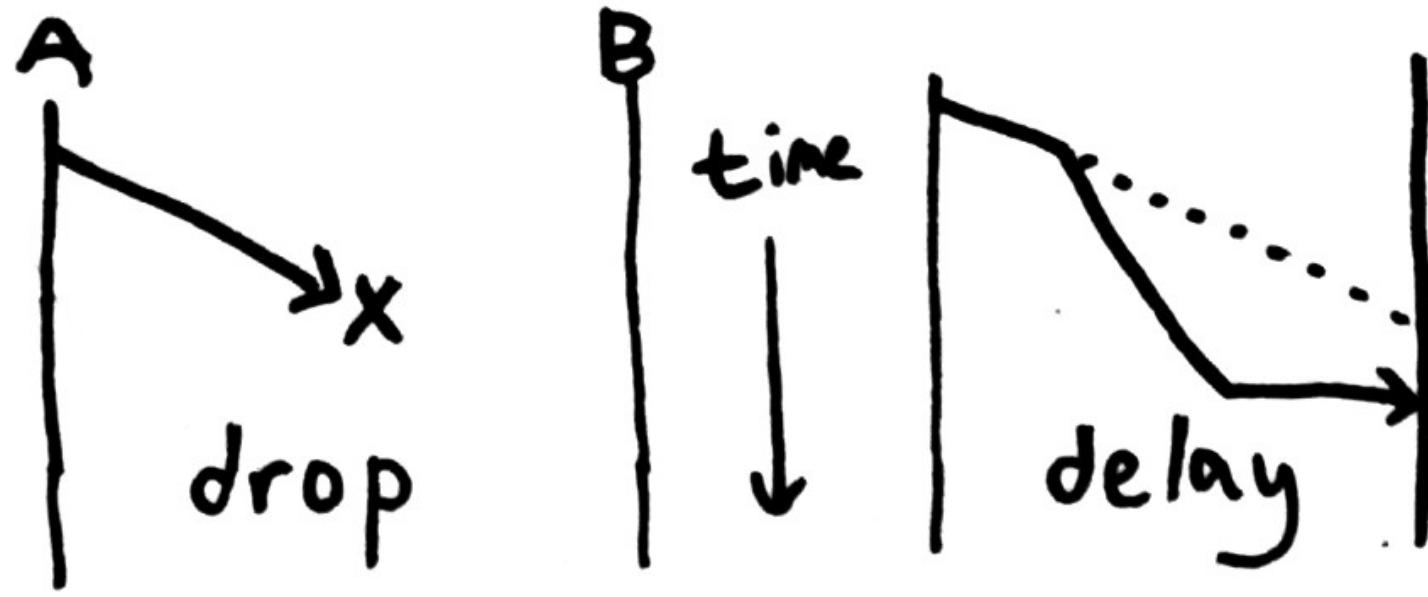


Why Care?



The net is
dark, and full of
TERAHS

Formally:



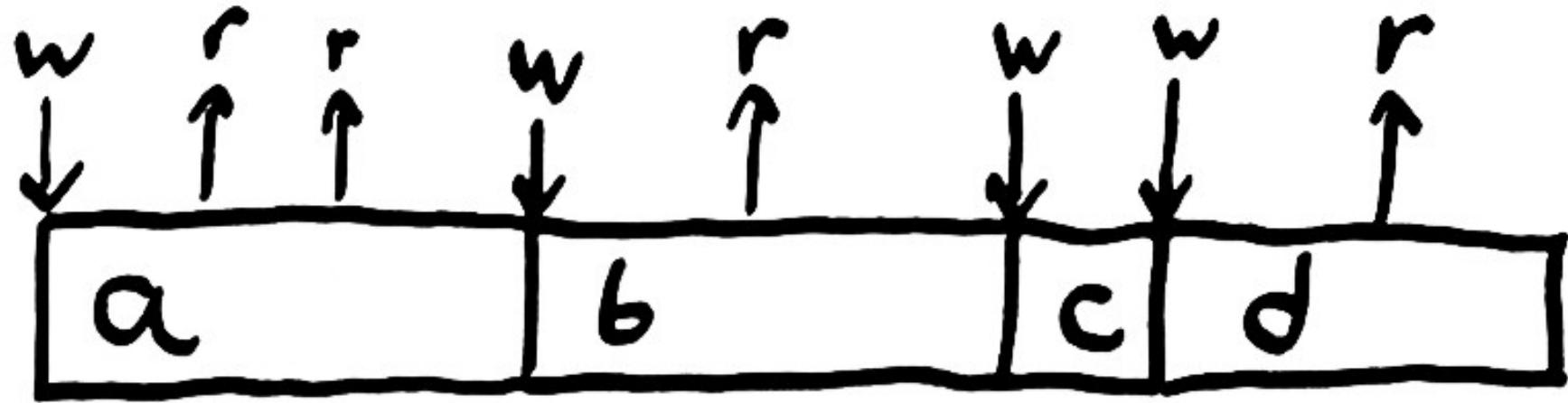
Causes

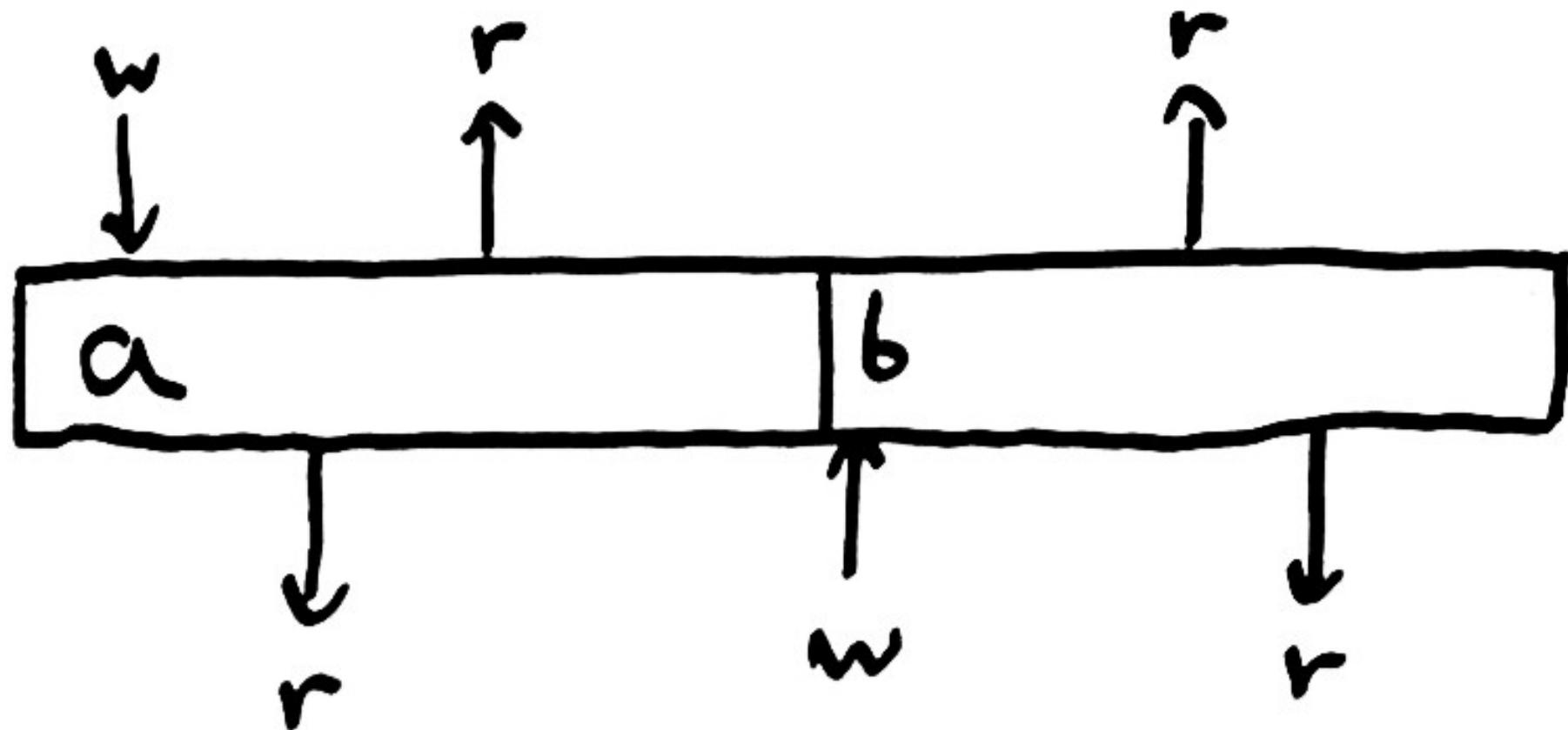
- GC pauses
- Network maint
- Segfaults & crashes
- Faulty NICs
- Bridge loops

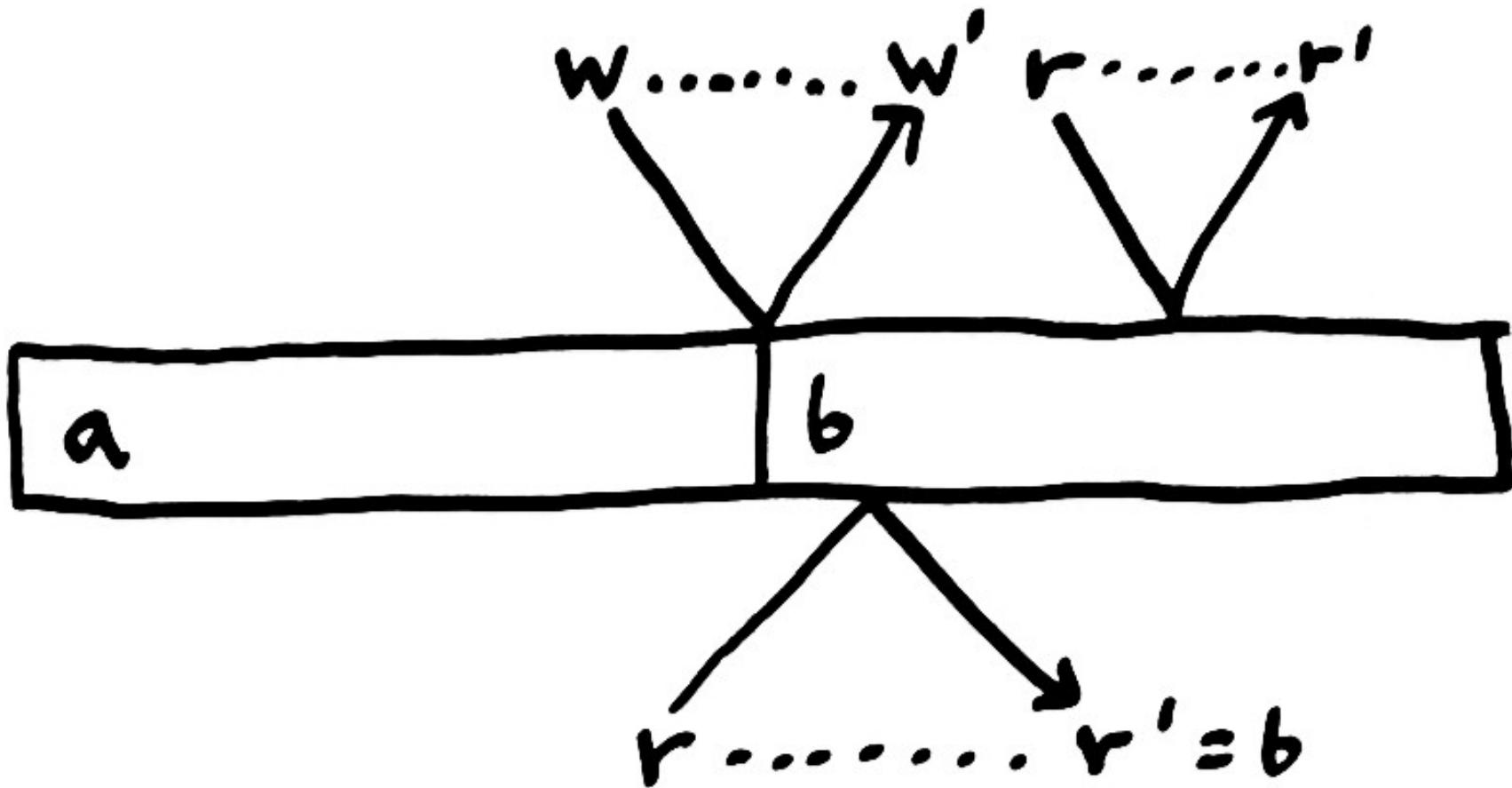
... continued

- Spanning tree
- Hosted networks
- The cloud
- WAN links & Backhoes

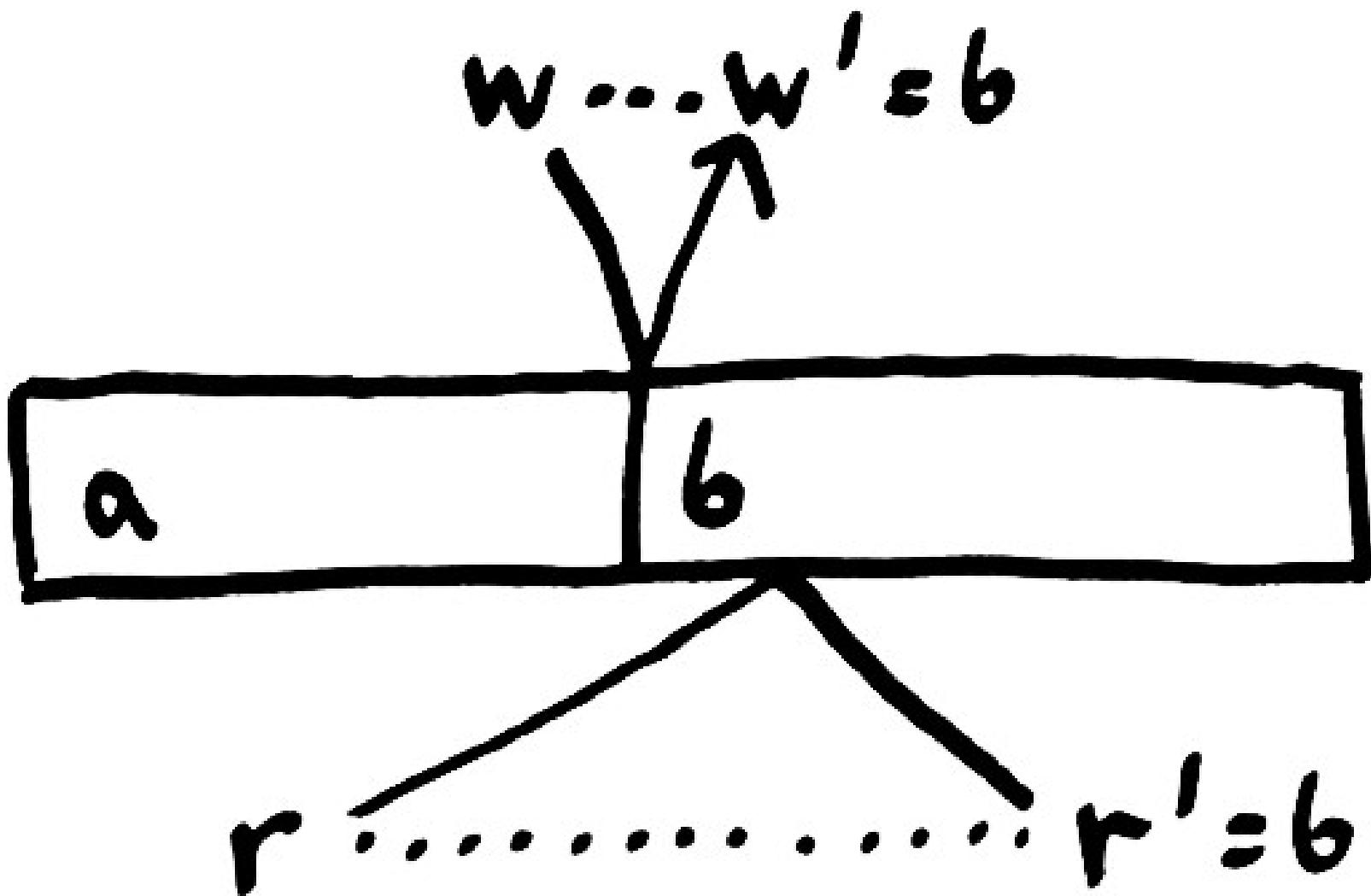
Intuitive Correctness



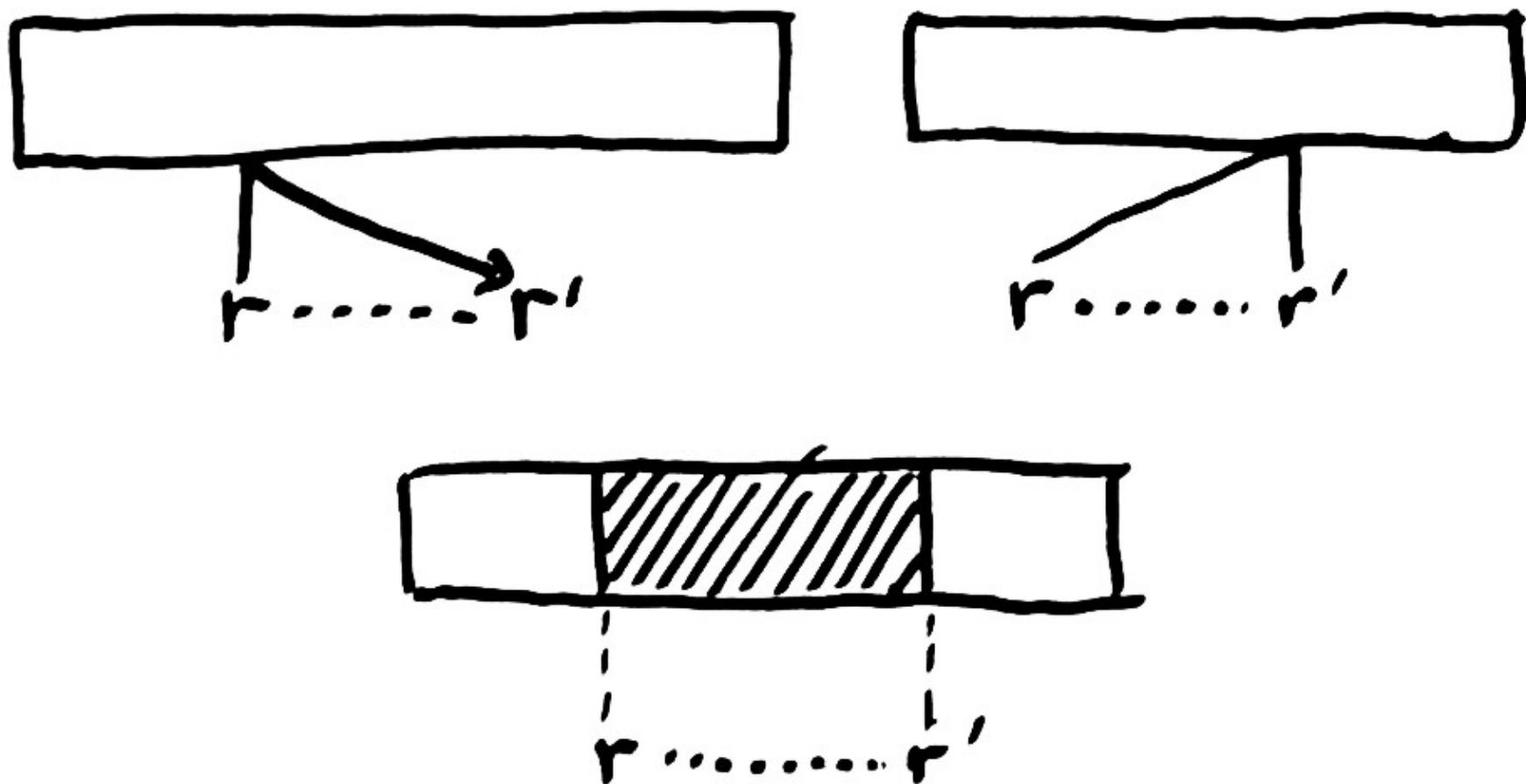




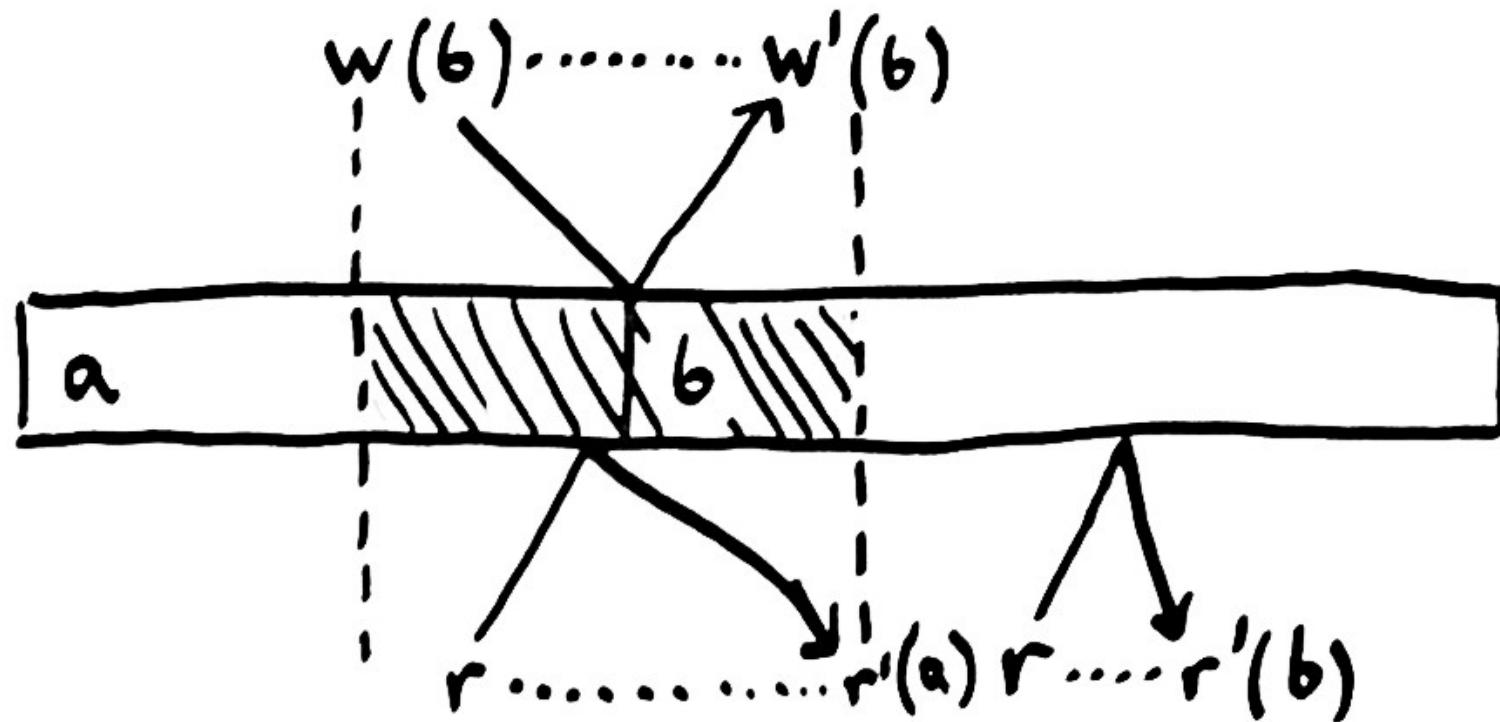
Synchronization
delay implies
ambiguous orders



But there are
finite bounds!



Once operation is complete,
it will be visible to all.



No stale reads

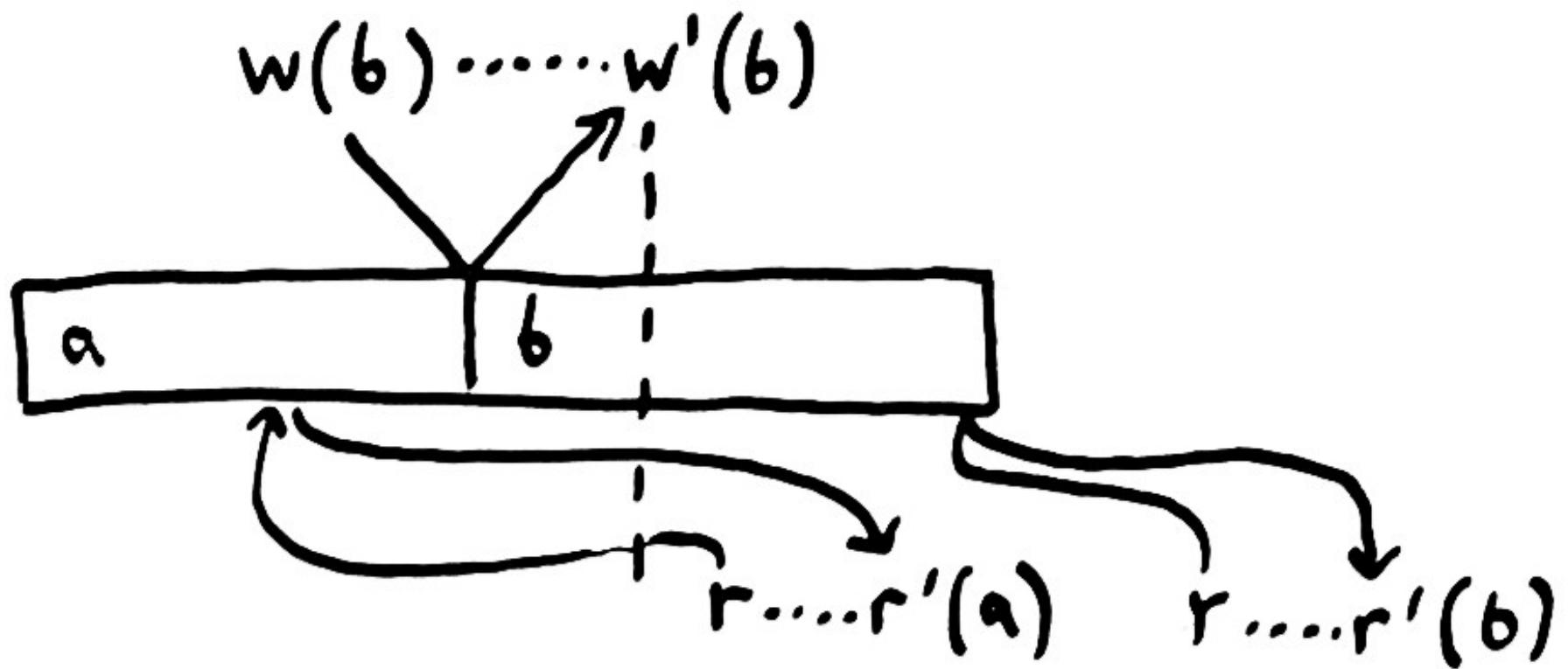
Safe state mutation

Everyone sees the
Same order of states.

Same properties as
atomic ops in a
multithreaded lang.

Sequential Consistency:

Everyone agrees on
order, but not on
time.

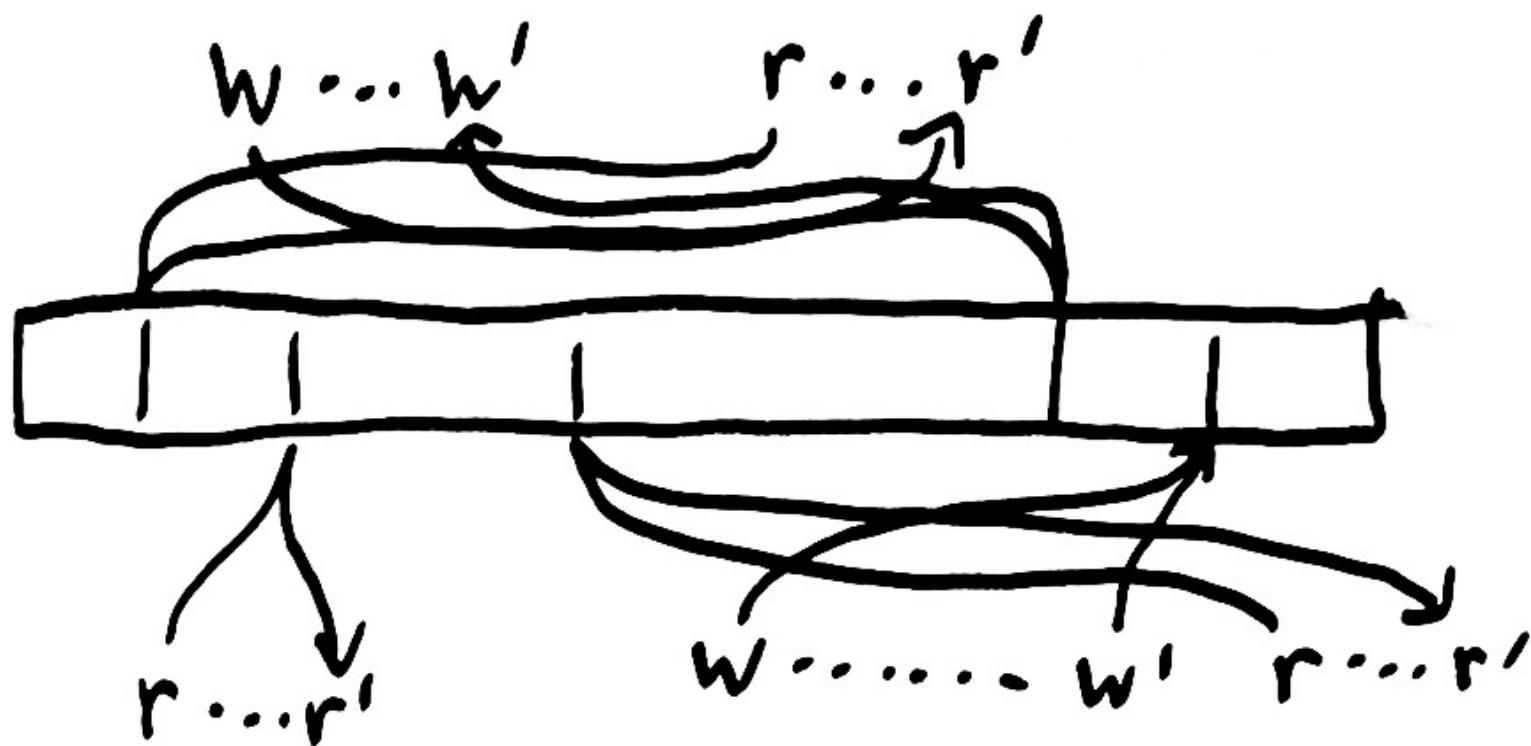


"lines can't cross"

Serializability

"Everything happens in some order, but nobody agrees on what that order was!"

Serializability



"ACID consistency/isolation"

"Acronyms are easy
to make shit up about."

-MRB

CAP:

C = Linearizability

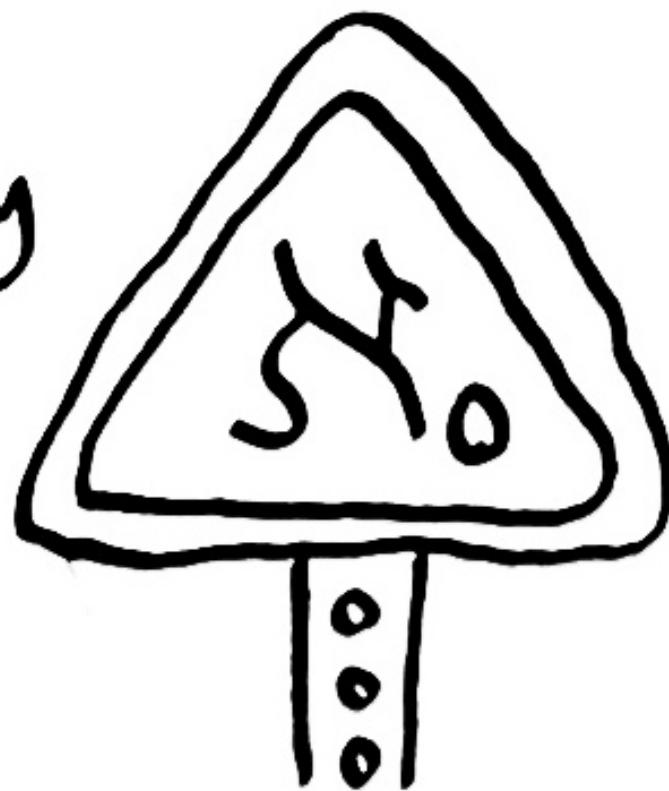
A = All up nodes can
satisfy all requests

P = Mandatory

Pick AP or CP.

Anyone who claims CA
has no idea what they're
doing.

Infinitely many
models!



Strong Serializable

Linearizable

Sequential

Causal

WFR

PRAM

MR

RYW

MW

Serializable

RR

SI

CS

RC

MAV

P-CI

Weaker consistency

models are more

available in failure.

Weaker consistency

models are less
intuitive.

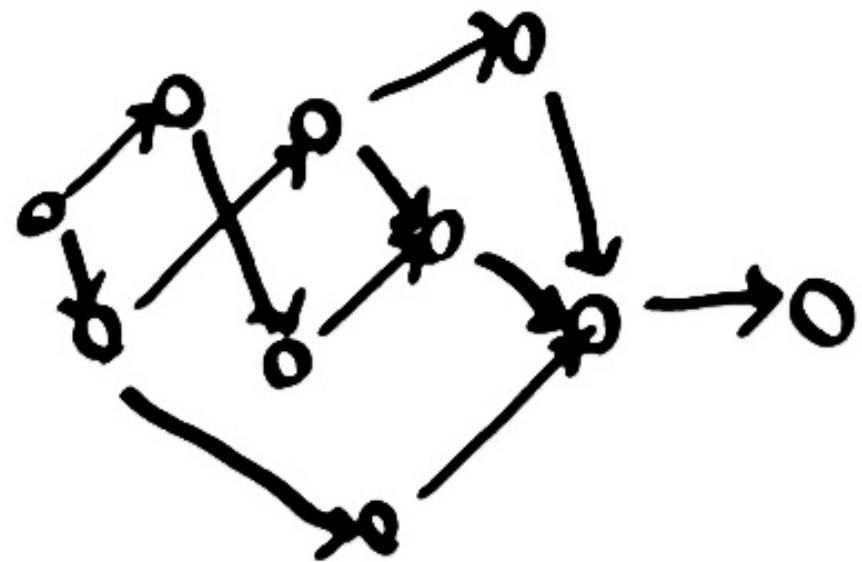
Weaker consistency

models are faster.

- CPU memory model

"Weak" \neq Unsafe

CRDTs



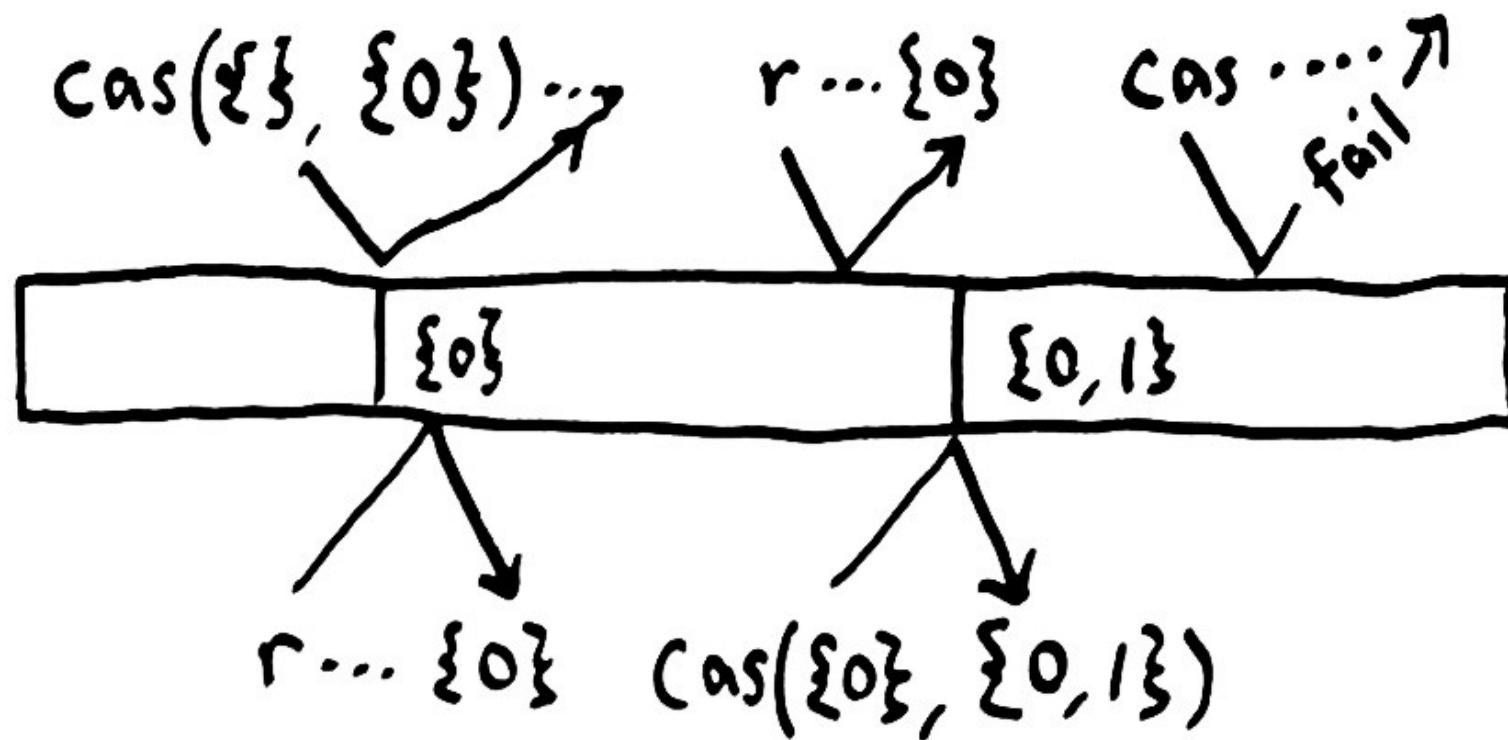
ZooKeeper

NuoDB

Kafka

Cassandra

Jepsen failed to find problems
in etcd, and other strongly
consistent systems. Why?



CAS, CAS, CAS, CAS, ... read final
 $+0 +1 +2 +3 \uparrow \{0, 1, 2, 3, \dots\}$
?

Jepsen: DB automation, testing,
failures, scheduling ops.



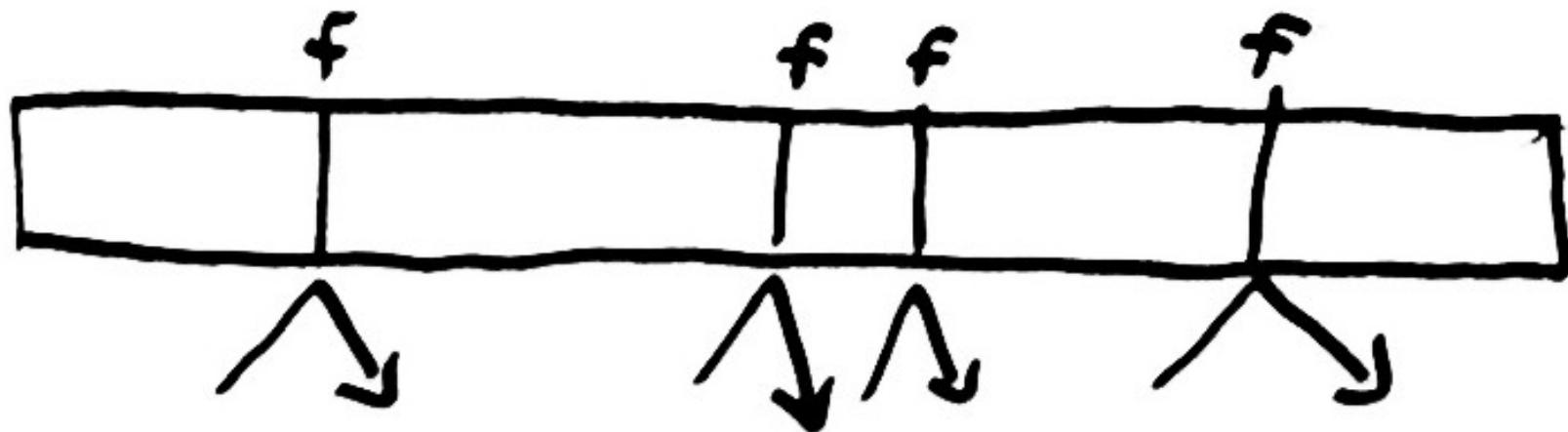
History of Ops



Knossos: Verifies Linearizability

Model

$f(\text{state}, \text{op}) \rightarrow \text{state}'$



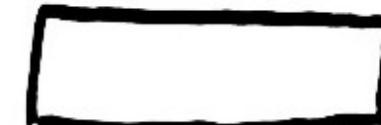
Model verifies that a
single threaded history is OK.



But we don't know the
singlethreaded history...

Histories

Client 1



2



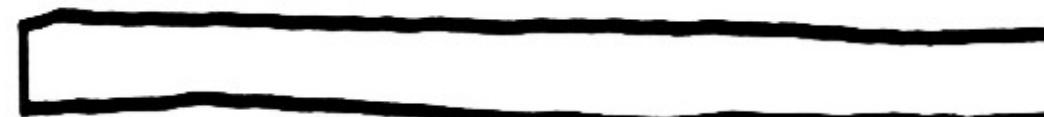
3



4



5



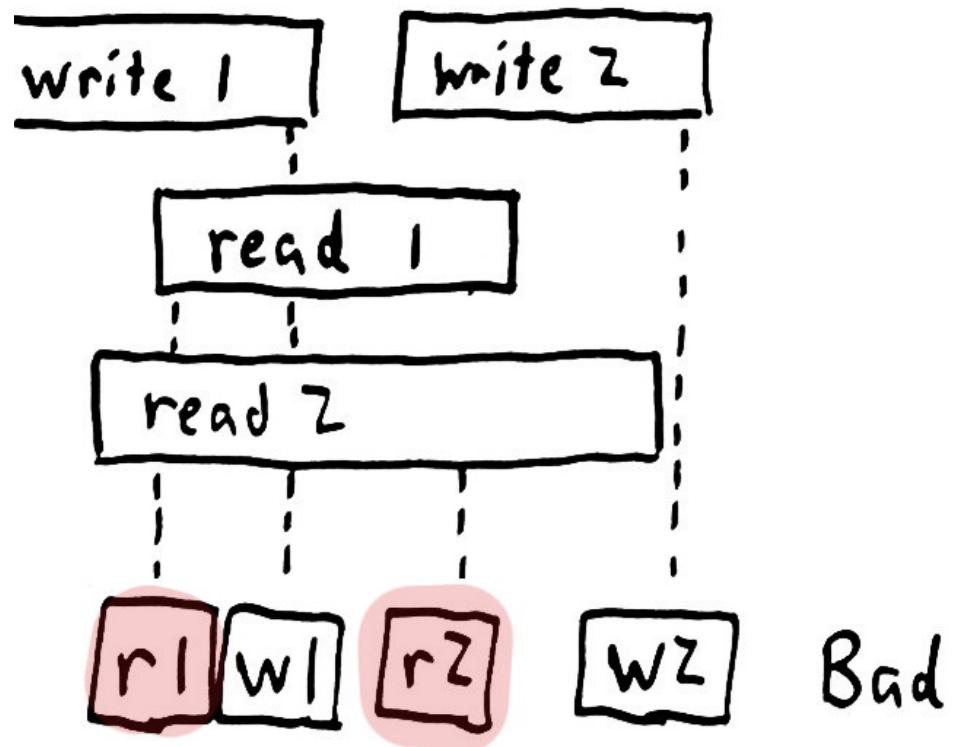
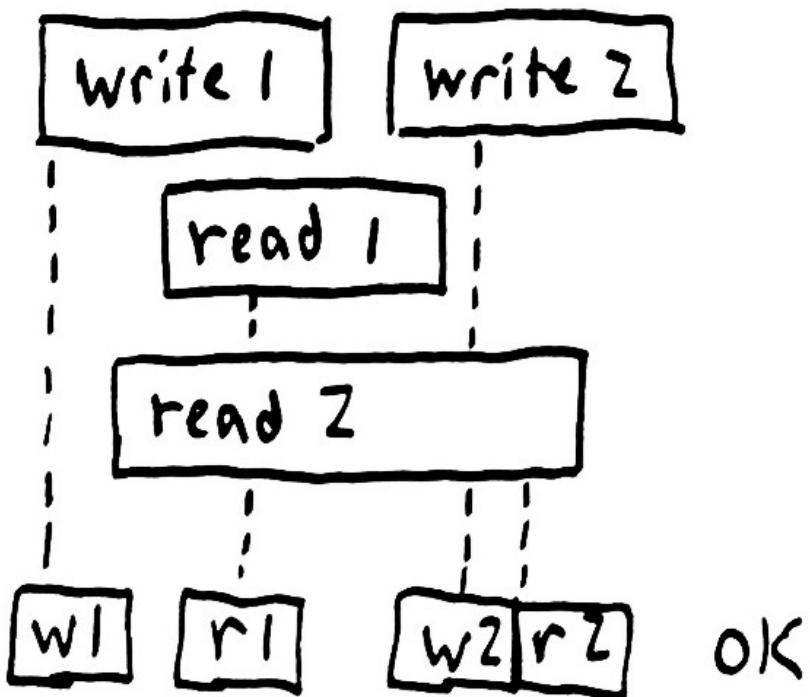
||||

Nemesis



t





$O(n!)$

(oh,)

Techniques:

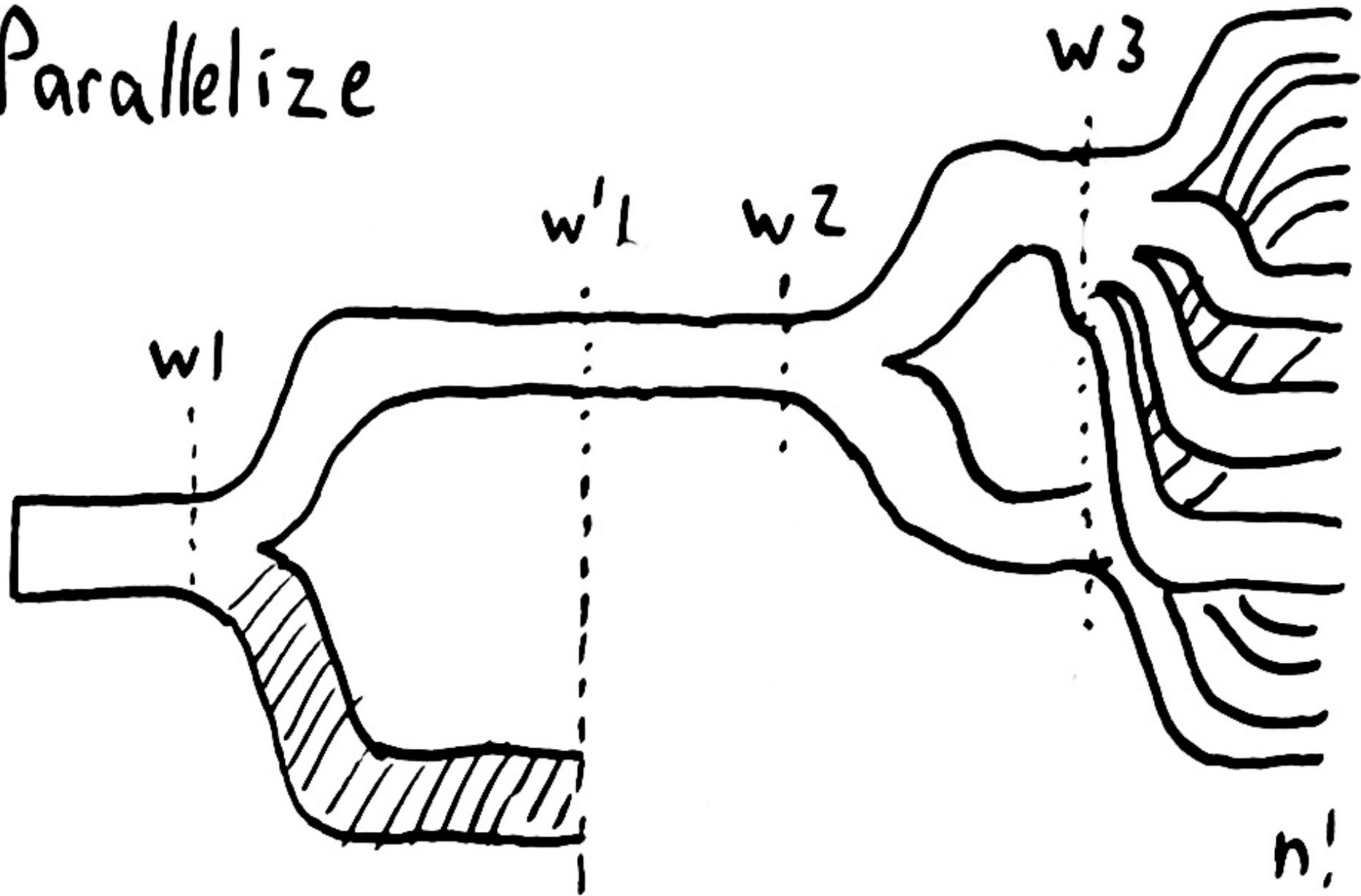
- Exploit degeneracy
- Parallelize
- Laziness + memoization
- Immutable, shared-structure representation of each world

Degeneracy

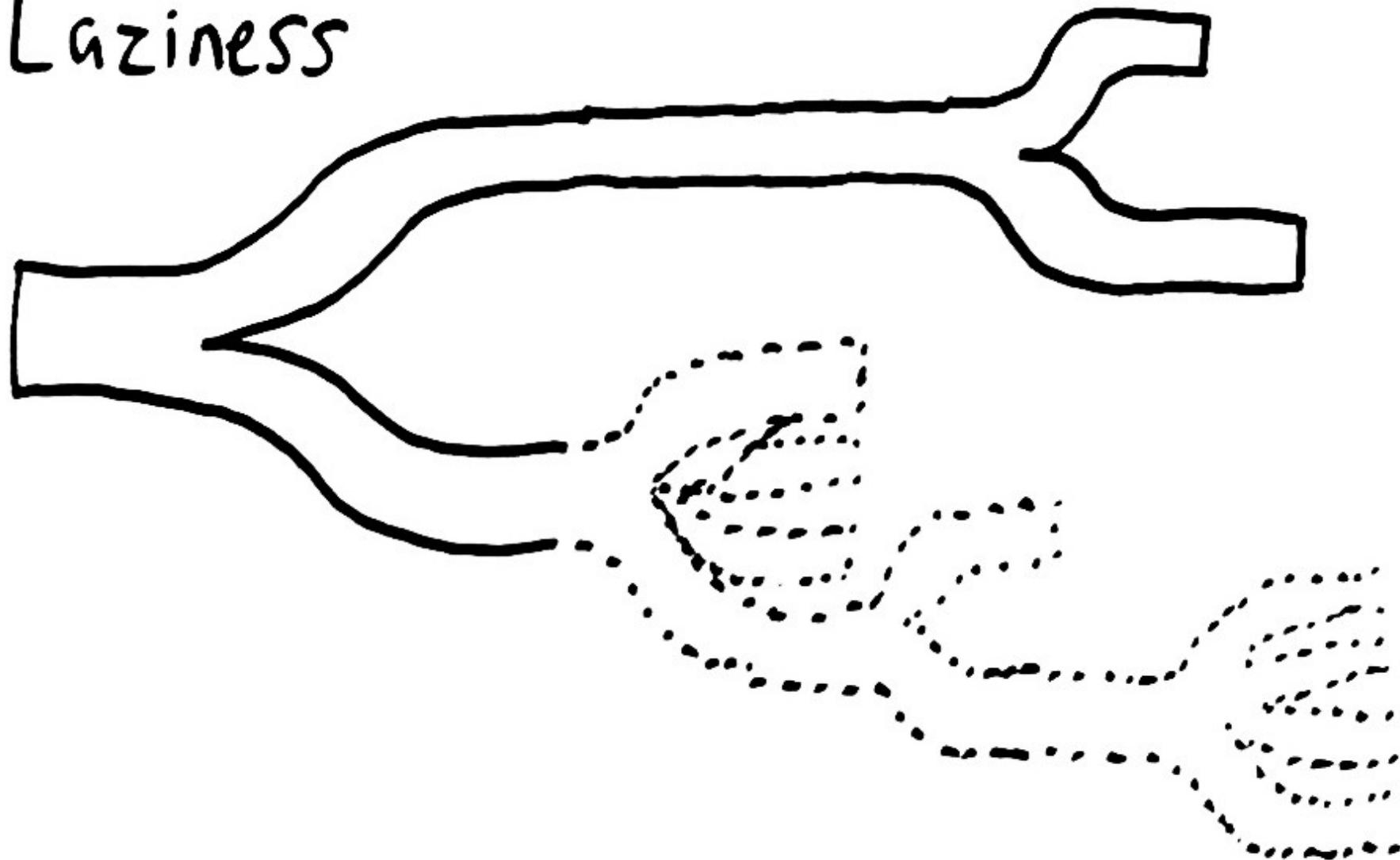
history	state	future
w'l, r'l	1	r'l, w'z, r'l, w'z
w'l	1	r'l, w'z, r'l, w'z
w'l, w'z	2	r'l, w'z, r'l, w'z
w'z	2	w'z, r'l, w'z, r'l, w'z

The past is a fiction.

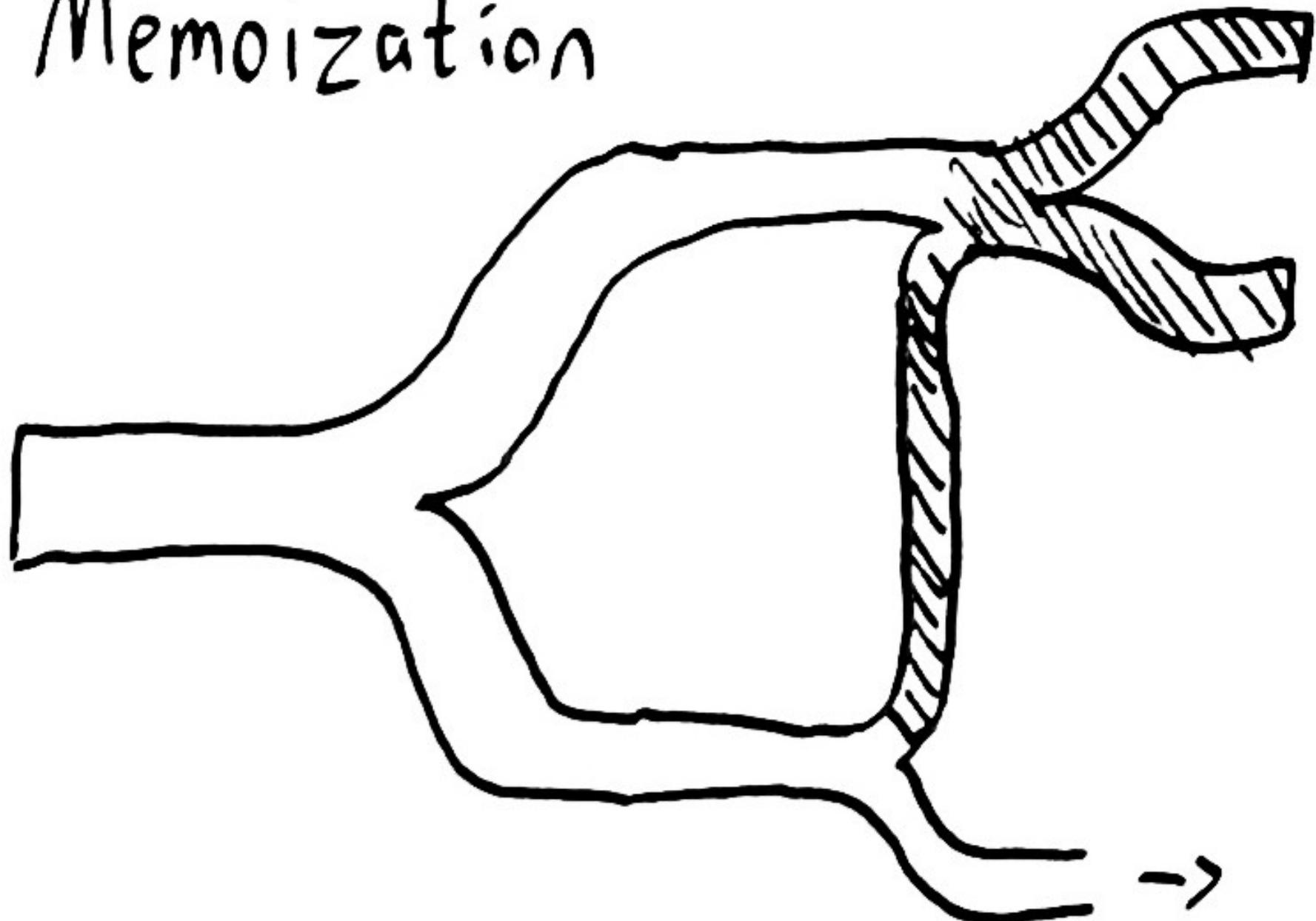
Parallelize

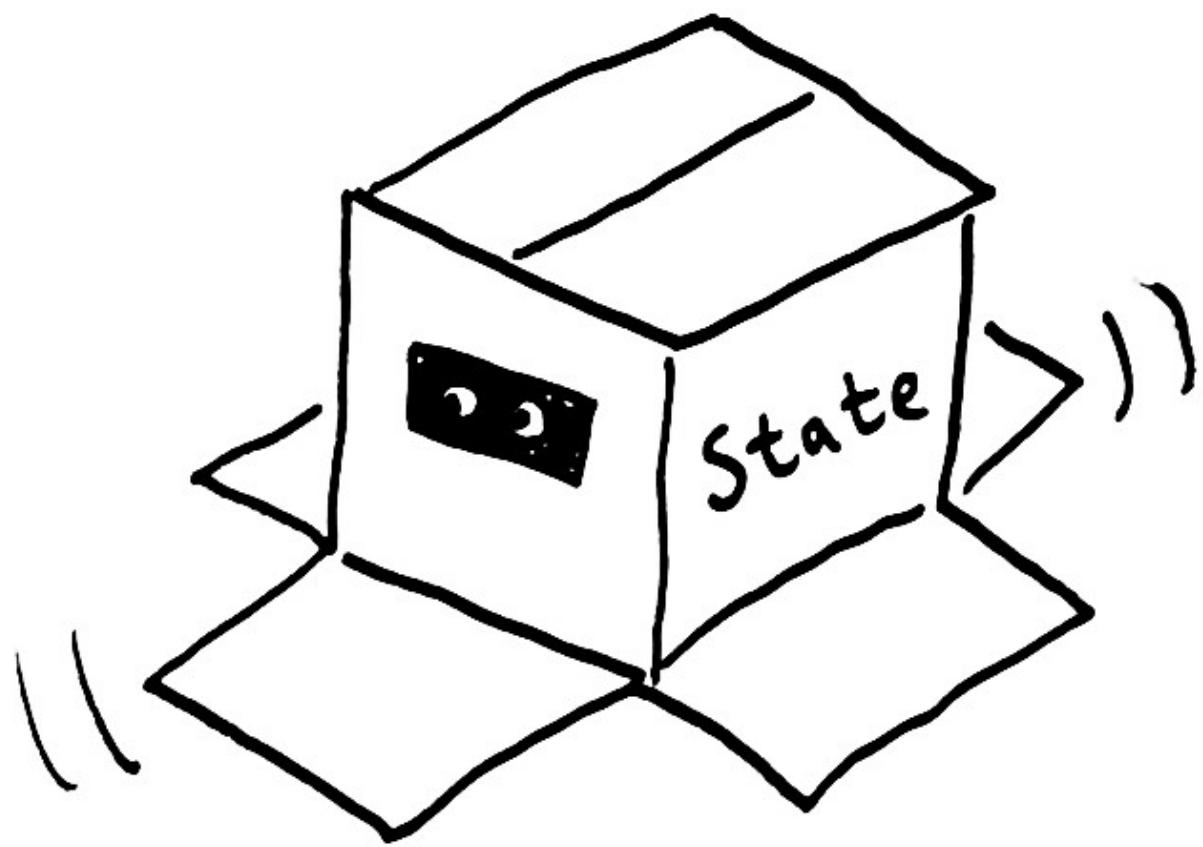


Laziness

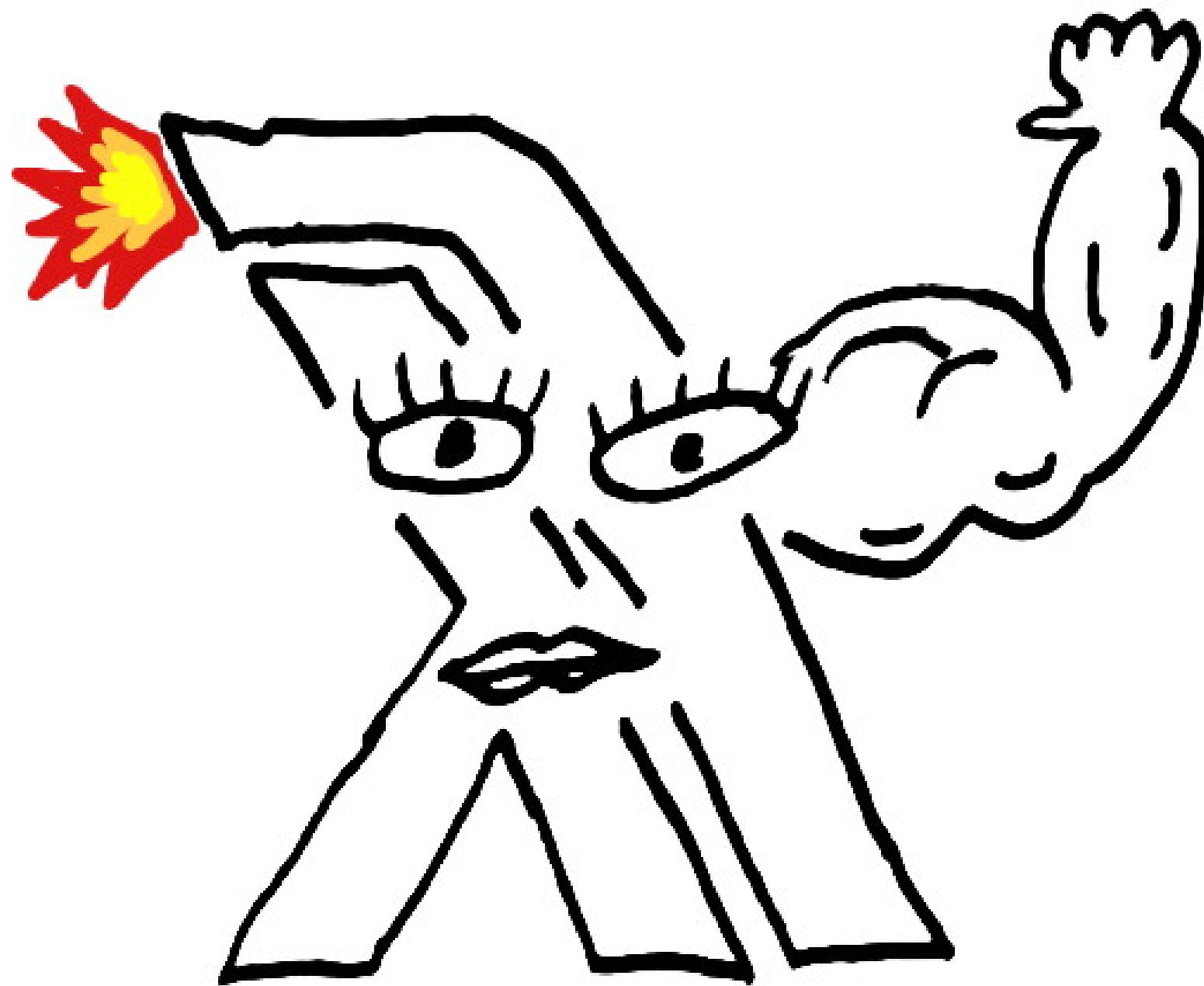


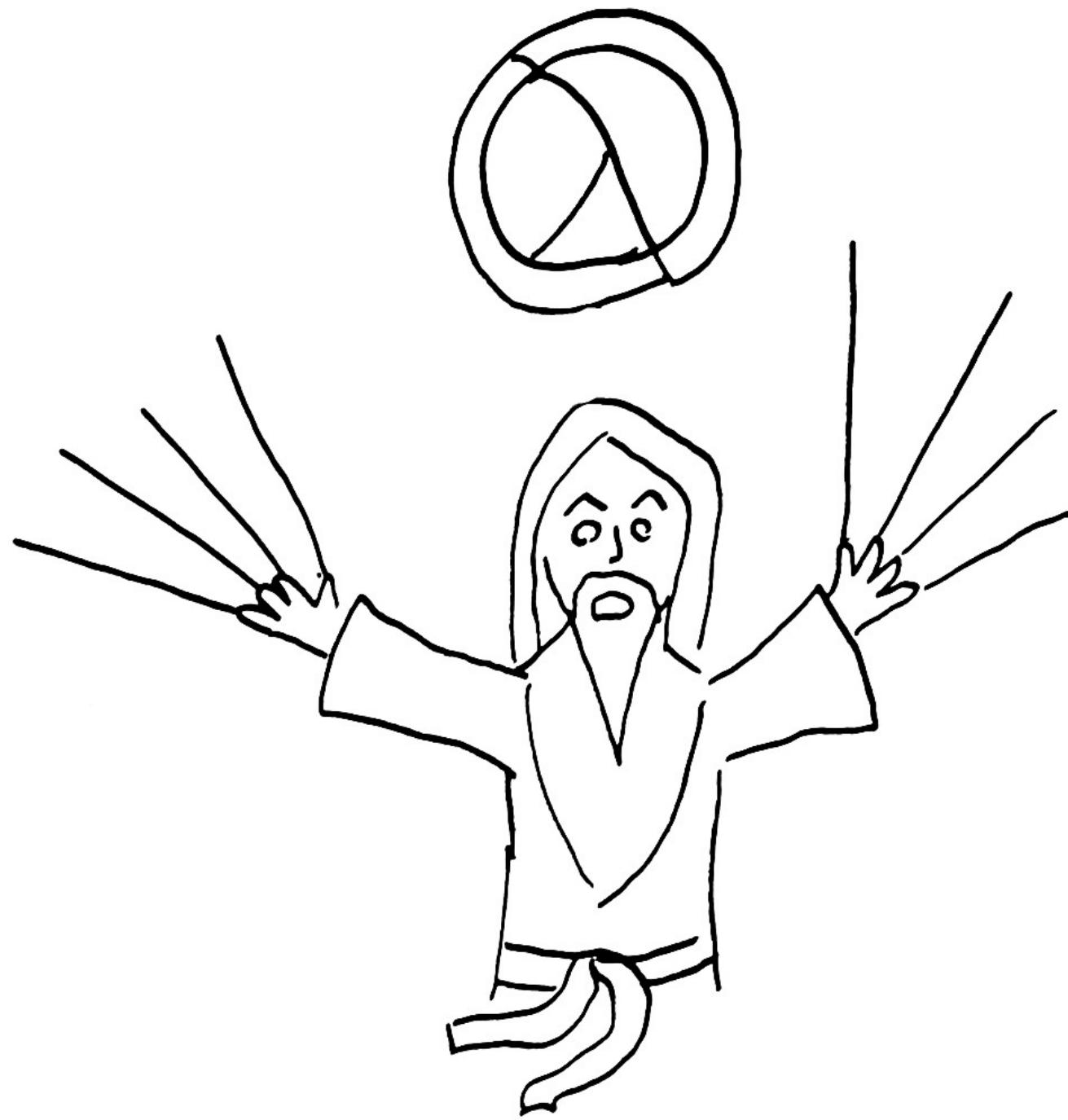
Memoization





- All parallel-safe
- Execution as data
- Adapted from Margo Seltzer's talk on autoparallelizing x86





Shared structure

Clojure's maps/vectors only store
deltas; reduces memory overhead
from world forks by orders of
magnitude.

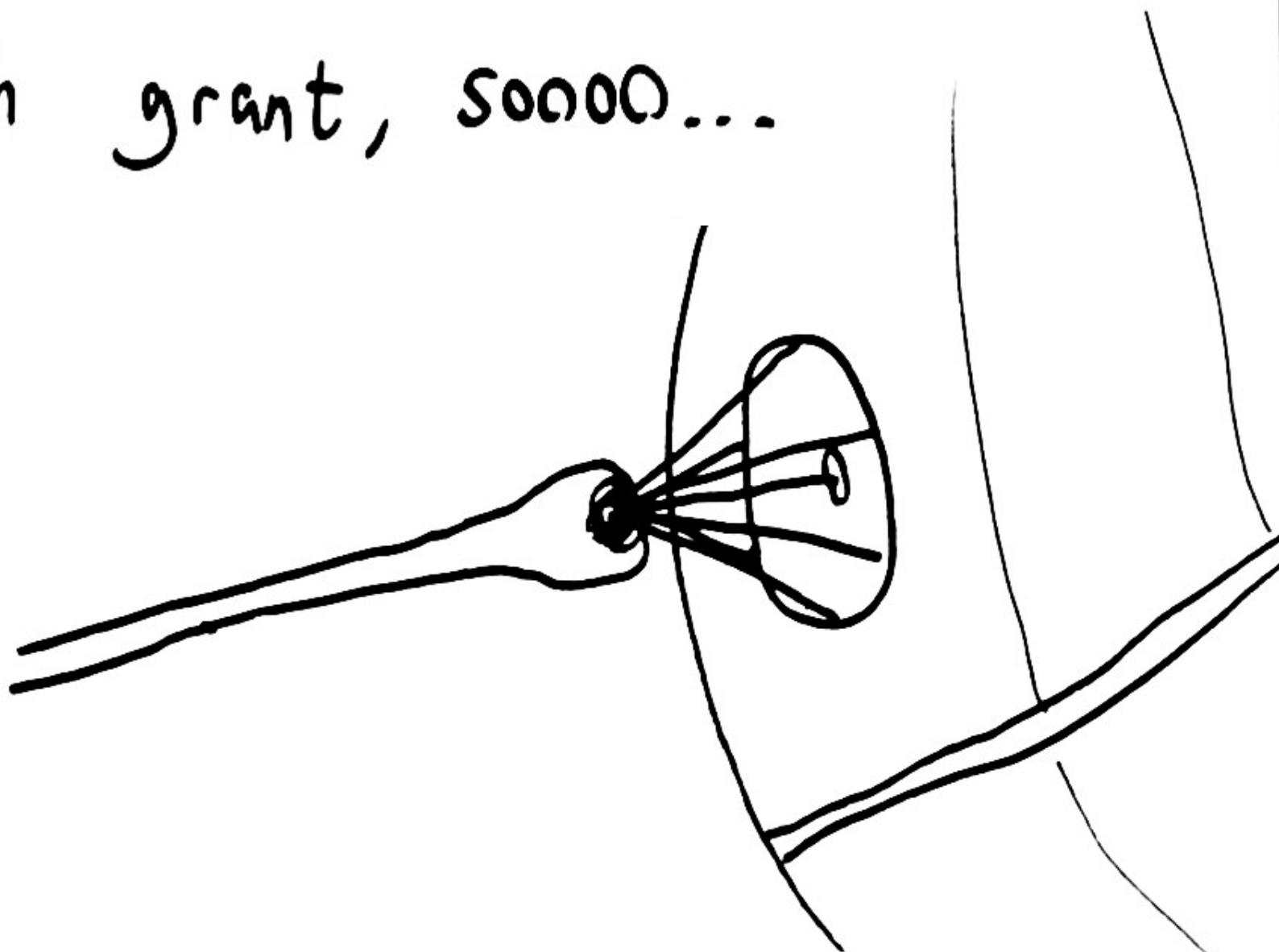


PUT A (pmap) ON IT!

Controlled use of JVM mutable
types where perf > determinism

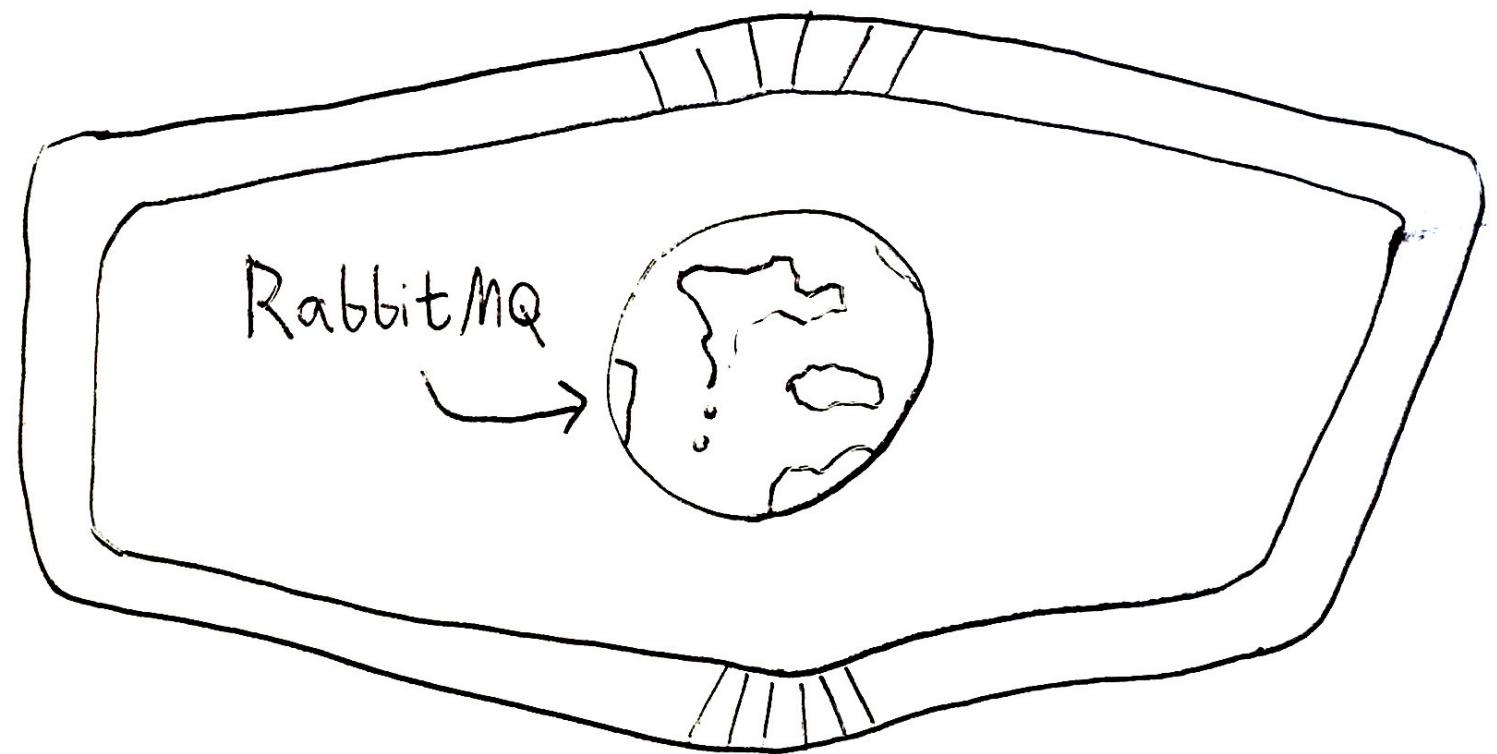
- Cliff Click's high-scale-lib
- Striped wrapper for j.u.c. PrioQueue
- j.u.c. atomics for CAS state
- Clojure atoms & futures too

Comcast gave me a
research grant, sooo...



1	[97.0%]	26	[97.6%]
2	[97.0%]	27	[97.6%]
3	[97.0%]	28	[97.0%]
4	[97.6%]	29	[97.6%]
5	[97.0%]	30	[97.6%]
6	[97.0%]	31	[97.0%]
7	[97.6%]	32	[97.6%]
8	[97.0%]	33	[97.6%]
9	[97.0%]	34	[97.6%]
10	[97.0%]	35	[97.6%]
11	[97.0%]	36	[97.0%]
12	[96.4%]	37	[96.4%]
13	[97.0%]	38	[97.0%]
14	[97.0%]	39	[96.4%]
15	[97.0%]	40	[97.6%]
16	[97.6%]	41	[97.0%]
17	[97.6%]	42	[97.6%]
18	[96.4%]	43	[96.4%]
19	[97.0%]	44	[97.0%]
20	[97.0%]	45	[96.4%]
21	[97.6%]	46	[96.4%]
22	[97.6%]	47	[97.0%]
23	[96.4%]	48	[97.6%]
24	[96.4%]	Mem	[41769/129152MB]
25	[96.4%]	Swp	[0/5119MB]

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1702	aphyr	20	0	43.9G	1007M	27148	S	4511	0.8	33:11.91	java -classpath /home/aphyr/j
1868	aphyr	20	0	43.9G	1007M	27148	R	93.2	0.8	0:40.60	java -classpath /home/aphyr/j
2430	aphyr	20	0	43.9G	1007M	27148	R	94.4	0.8	0:38.89	java -classpath /home/aphyr/j
2432	aphyr	20	0	43.9G	1007M	27148	R	78.7	0.8	0:43.57	java -classpath /home/aphyr/j
2436	aphyr	20	0	43.9G	1007M	27148	R	93.8	0.8	0:43.06	java -classpath /home/aphyr/j
2803	aphyr	20	0	43.9G	1007M	27148	R	93.8	0.8	0:39.03	java -classpath /home/aphyr/j
2804	aphyr	20	0	43.9G	1007M	27148	R	90.2	0.8	0:38.59	java -classpath /home/aphyr/j
2805	aphyr	20	0	43.9G	1007M	27148	R	84.8	0.8	0:37.92	java -classpath /home/aphyr/j
2809	aphyr	20	0	43.9G	1007M	27148	R	80.6	0.8	0:38.95	java -classpath /home/aphyr/j



RabbitMQ

- Message Queue
- Knows about worker tasks
- Let's make a look out of it!

« [Preventing Unbounded Buffers with RabbitMQ](#)

[Breaking things with RabbitMQ 3.3](#) »

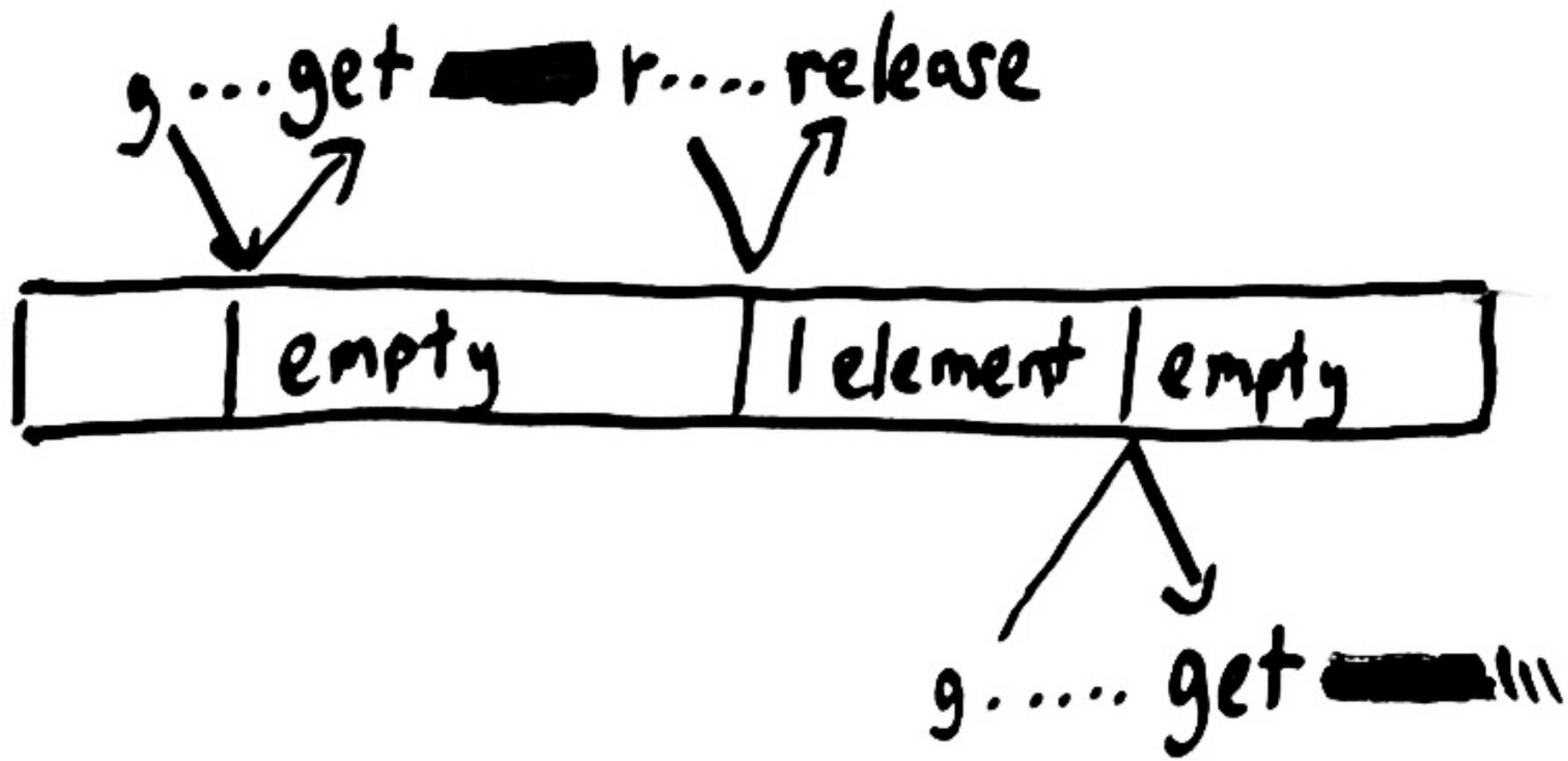
Distributed Semaphores with RabbitMQ

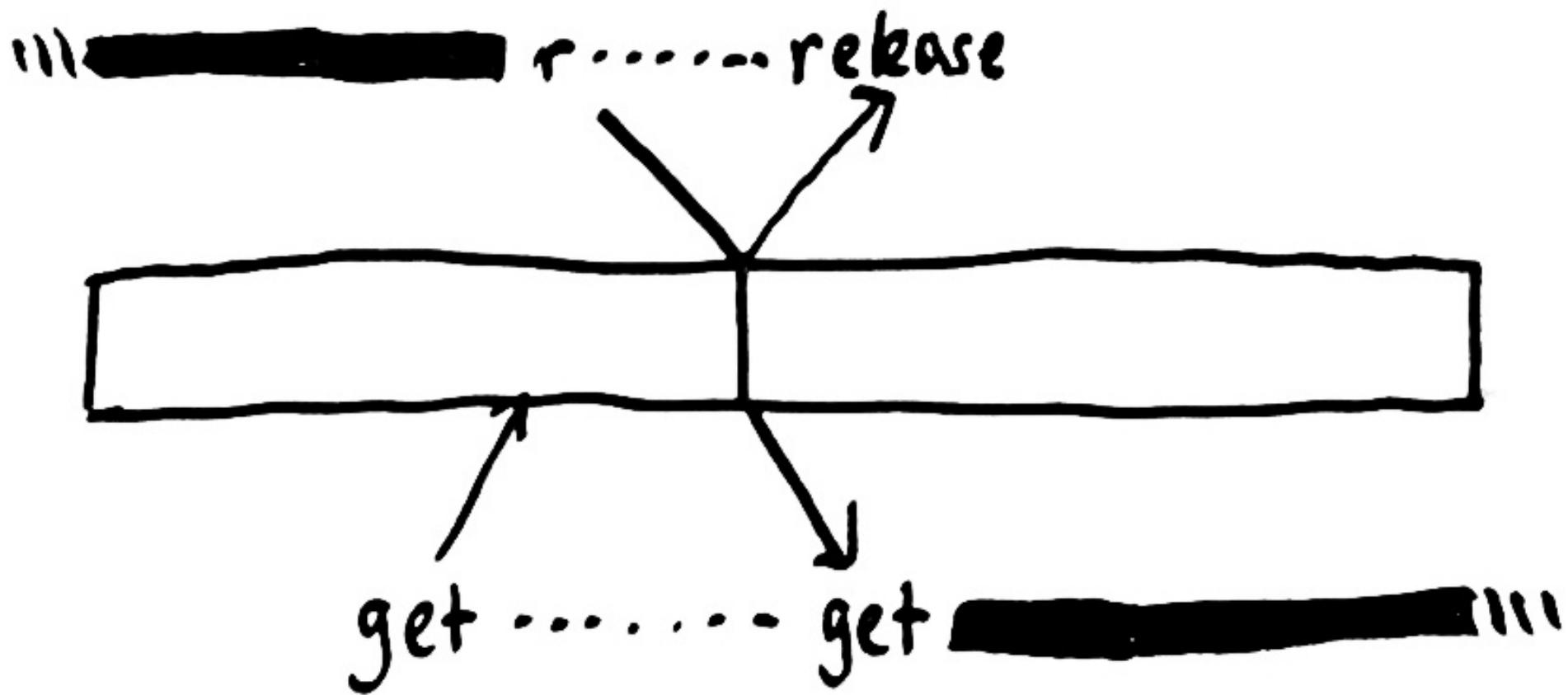
In this blog post we are going to address the problem of controlling the access to a particular resource in a distributed system. The technique for solving this problem is well known in computer science, it's called Semaphore and it was invented by Dijkstra in 1965 in his paper called "Cooperating Sequential Processes". We are going to see how to implement it using AMQP's building blocks, like consumers, producers and queues.

The Need for Semaphores

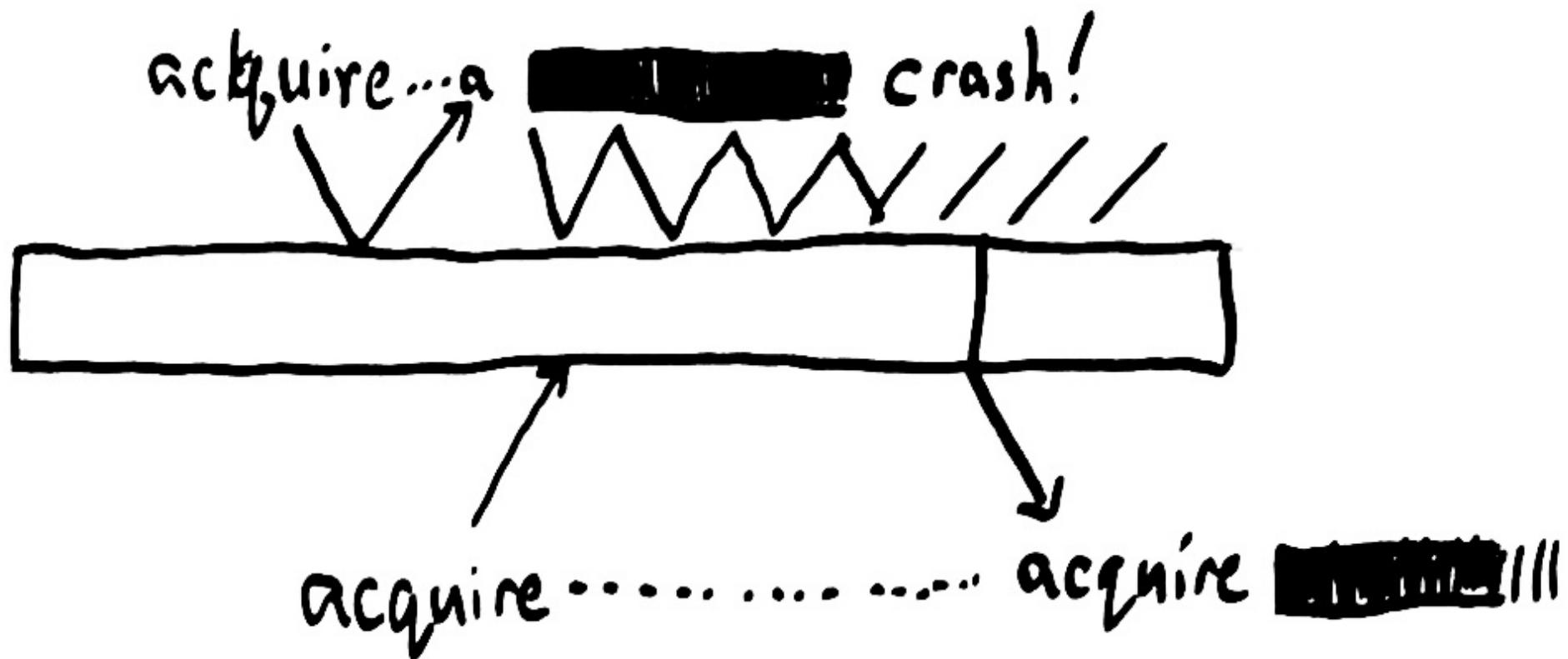
Before going into the actual solution, let's see when we might actually need something like this:

Let's say our application has many processes taking jobs from a queue and then inserting records to a database, we might need to limit how many of them do it concurrently.





But Rabbit is not
a linearizable system!

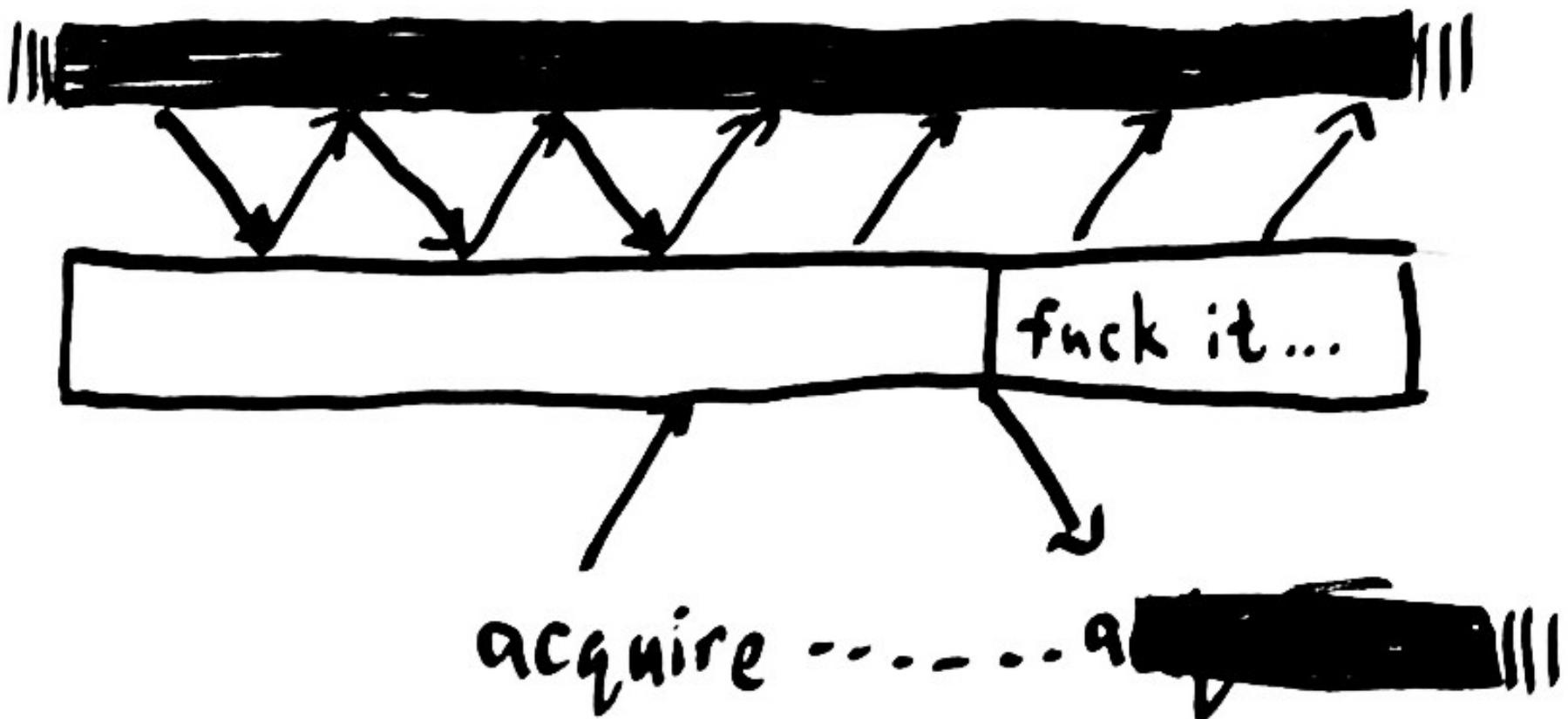


- Rabbit docs are very clear: partitions ruin all guarantees

=ERROR REPORT==== 10-Apr-2014::13:16:08 ===
** Node rabbit@n3 not responding **
** Removing (timedout) connection **

=INFO REPORT==== 10-Apr-2014::13:16:29 ===
rabbit on node rabbit@n5 down

=ERROR REPORT==== 10-Apr-2014::13:16:45 ===
Mnesia(rabbit@n1): ** ERROR ** mnesia_event got
{inconsistent_database,
running_partitioned_network, rabbit@n3}



```
(defrecord Mutex [locked?])
```

Model

```
(step [r op]
  (condp = (:f op)
    :acquire (if locked?
      (inconsistent "already held")
      (Mutex. true)))
    :release (if locked?
      (Mutex. false)
      (inconsistent "not held")))))
```

Not linearizable. Linearizable prefix was:

...

:nemesis	:info	:stop	nil
:nemesis	:info	:stop	"fully connected"
:nemesis	:info	:start	nil
:nemesis	:info	:start	"partitioned into [(:n1 :n4) (:n3 :n5 :n2)]"
3	:invoke	:acquire	nil
2	:invoke	:acquire	1
2	:ok	:acquire	1
1	:invoke	:acquire	2

Followed by inconsistent operation:

1	:ok	:acquire	2
---	-----	----------	---

Last consistent worlds were: -----

World from fixed history:

```
1 :invoke :acquire 1
1 :invoke :release nil
2 :invoke :acquire 1
```

and current state #jepsen.model.Mutex{:locked true}

with pending operations:

```
3 :invoke :acquire nil
1 :invoke :acquire 2
0 :invoke :acquire nil
4 :invoke :acquire nil
```

Inconsistent state transitions:
([{:locked true} "already held"])

We expect this from any
queue-client crashes
make exactly once impossible!

Queues often sacrifice
linearizability to provide
fault tolerance.

Perfectly fine queue.

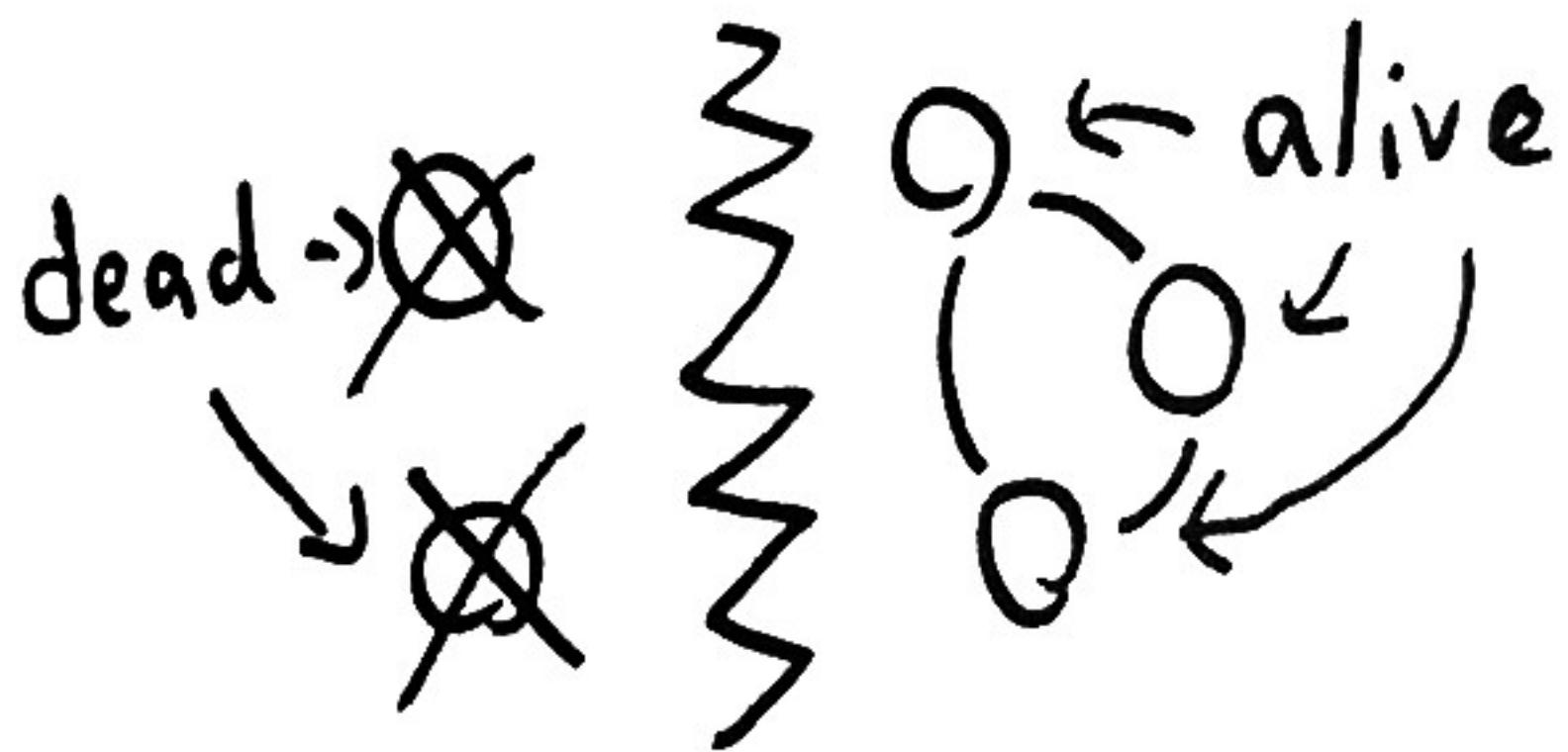
Not a lock service.

"It's Fine."

	Clustering
ave different	A cluster forms a single logical broker.
RabbitMQ and Erlang.	Nodes must run the same version of RabbitMQ, and frequently Erlang.
WAN links. ecured by SSL), ions to be set up.	Brokers must be connected via reliable LAN links. Communication is via Erlang internode messaging, requiring a shared Erlang cookie.
opology you	All nodes connect to all other nodes in both directions.
nce from the CAP	Chooses Consistency and Availability (or optionally Consistency and Partition Tolerance) from the CAP theorem.
erated while some	Clustering is all-or-nothing.
ly see queues in	A client connecting to any node can see queues on all nodes.

CP-mode docs say
Pause - Minority
mode is safest.

"At most the nodes
in a single partition
will run."



```
INFO jepsen.util - 15 :invoke :dequeue nil
INFO jepsen.util - 15 :ok      :dequeue 2935
INFO jepsen.util - 2551 :invoke :enqueue 3519
INFO jepsen.util - 2551 :ok      :enqueue 3519
INFO jepsen.util - 2501 :invoke :dequeue nil
WARN jepsen.core - Process 2501 indeterminate
java.util.concurrent.ExecutionException: java.io.IOException
    at java.util.concurrent.FutureTask.report(FutureTask.java:122)
    at java.util.concurrent.FutureTask.get(FutureTask.java:202)
    ...
Caused by: com.rabbitmq.client.ShutdownSignalException: channel error; reason:
{#method<channel.close>(reply-code=404, reply-text=NOT_FOUND - home node
'rabbit@n1' of durable queue 'jepsen.queue' in vhost '/' is down or
inaccessible, class-id=60, method-id=70), null, ""}
```

```
:total-queue
{:valid? false,
 :lost
#{2558 3359 3616 3173 1858 2030 2372 3135 3671 3534 3358 2855 3251
 3429 2615 3248 2479 1976 2430 3478 3693 2388 3174 3484 3638 2813
 3280 2282 2475 3239 2973 1984 3630 2264 2523 2565 2462 3278 3425

... lots more lines ...

3313 3413 3443 2048 3513 2705 3392 3001 2215 3097 3364 3531 2605
2411 2220 2042 1923 2314 3592 3538 3128 2801 3636 1861 3500 3143
3276 1991 3343 3656 3233 3611 3244 3717 3314 2922 3404 3708},

:unexpected
#{487 497 491 510 493 490 508 504 505 502 495 506 500 496 501 498 507
 494 489 492 503 509 499 488},

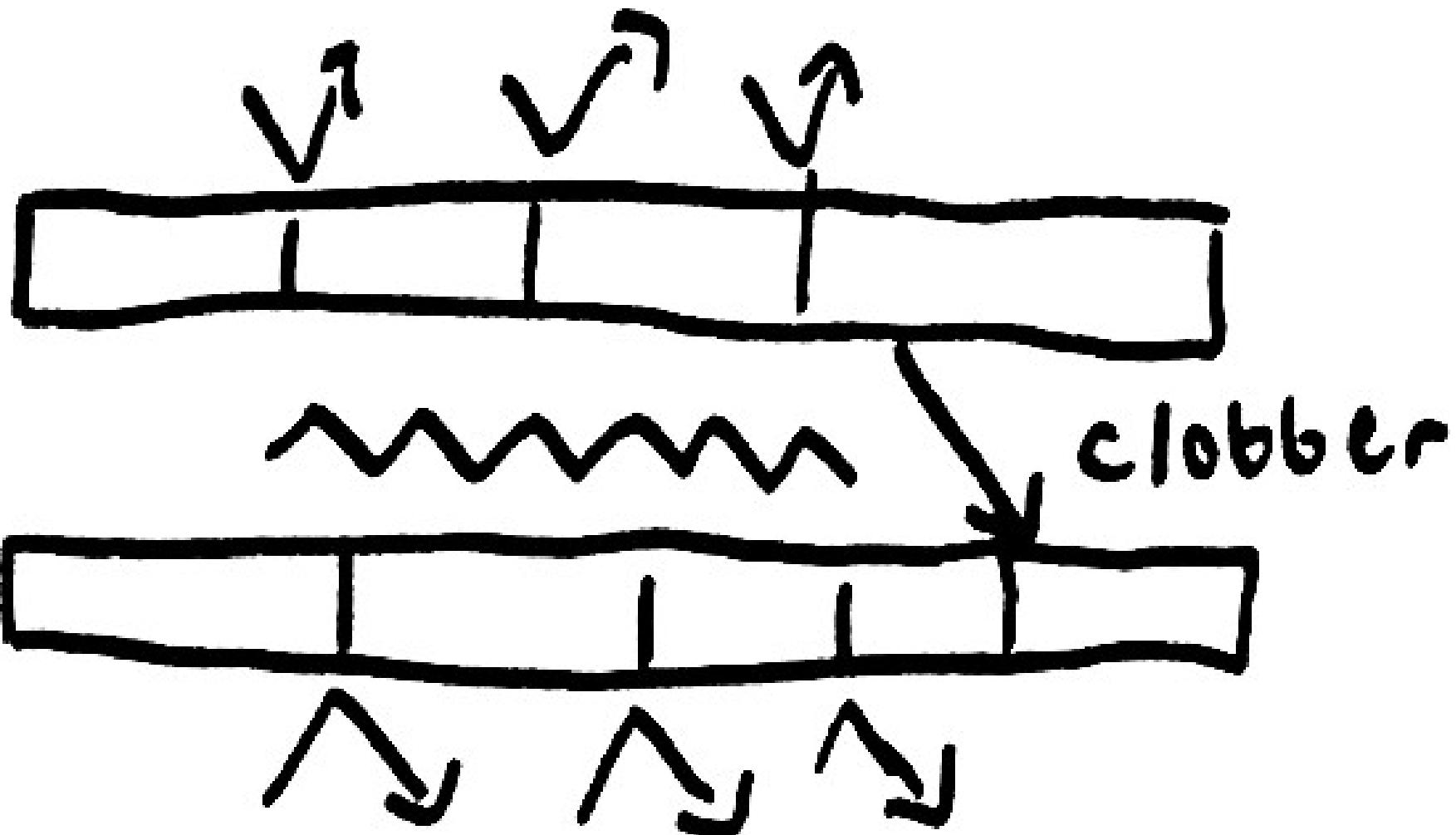
:recovered
#{519 521 529 510 527 518 512 517 516 515 523 531 525 528 522 1398
 520 524 513 509 511},

:ok-frac 786/1249,
:unexpected-frac 8/1249,
:lost-frac 1312/3747,
:recovered-frac 7/1249}}
```

35% of acknowledged
messages LOST.

Not a theoretical issue.
Real deployments hit this.

How does Rabbit
recover from a
partitioned cluster?



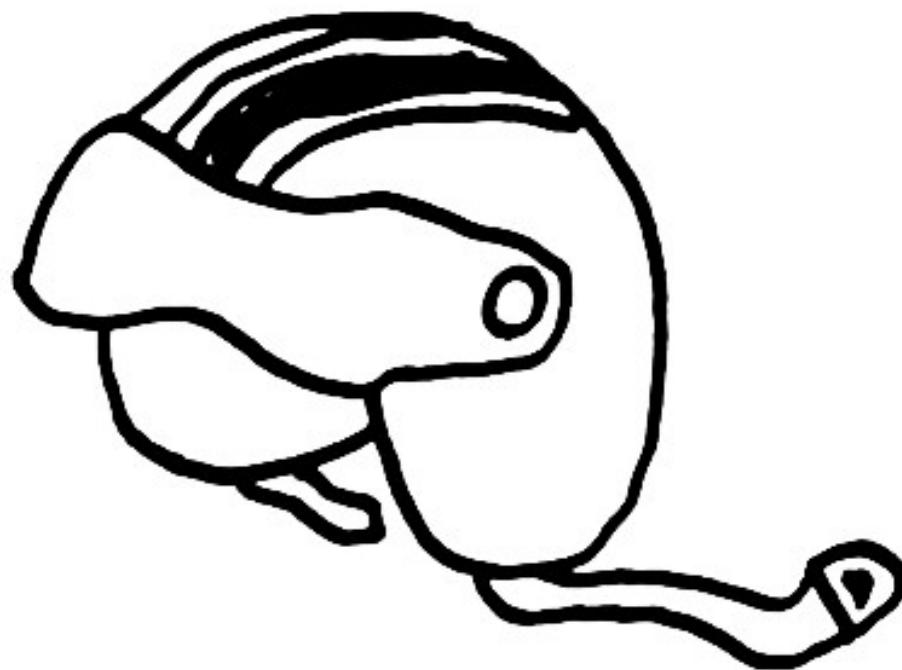
Rabbit tips:

- Use "ignore" partition mode
- Manually dump/recover isolated nodes
- Don't use as a mutex
- Shovel is an SPOF
- Federation ???

e t c d

- Based on Raft
- HTTP API
- Configuration & Coordination
- Reads, writes, value &
index-based CAS

Race Conditions in Cluster join!



FAIL in (register-test) (etcd_test.clj:45)
expected: (:valid? (:results test))
actual: false
Not linearizable. Linearizable prefix was:
...
:nemesis :info :start "Cut off {:n5 #{:n4 :n1}, :n2
#{:n4 :n1}, :n3 #{:n4 :n1}, :n1 #{:n3 :n2 :n5}, :n4
#{:n3 :n2 :n5}}"
2 :invoke :cas [1 4]
4 :invoke :read 1
4 :ok :read 1
2 :ok :cas [1 4]

Followed by inconsistent operation:
0 :invoke :read 1

World with fixed history:

```
...
4 :invoke :write 1
0 :invoke :read 1
3 :invoke :read 1
1 :invoke :read 1
4 :invoke :read 1
2 :invoke :cas [1 4]
```

led to state:

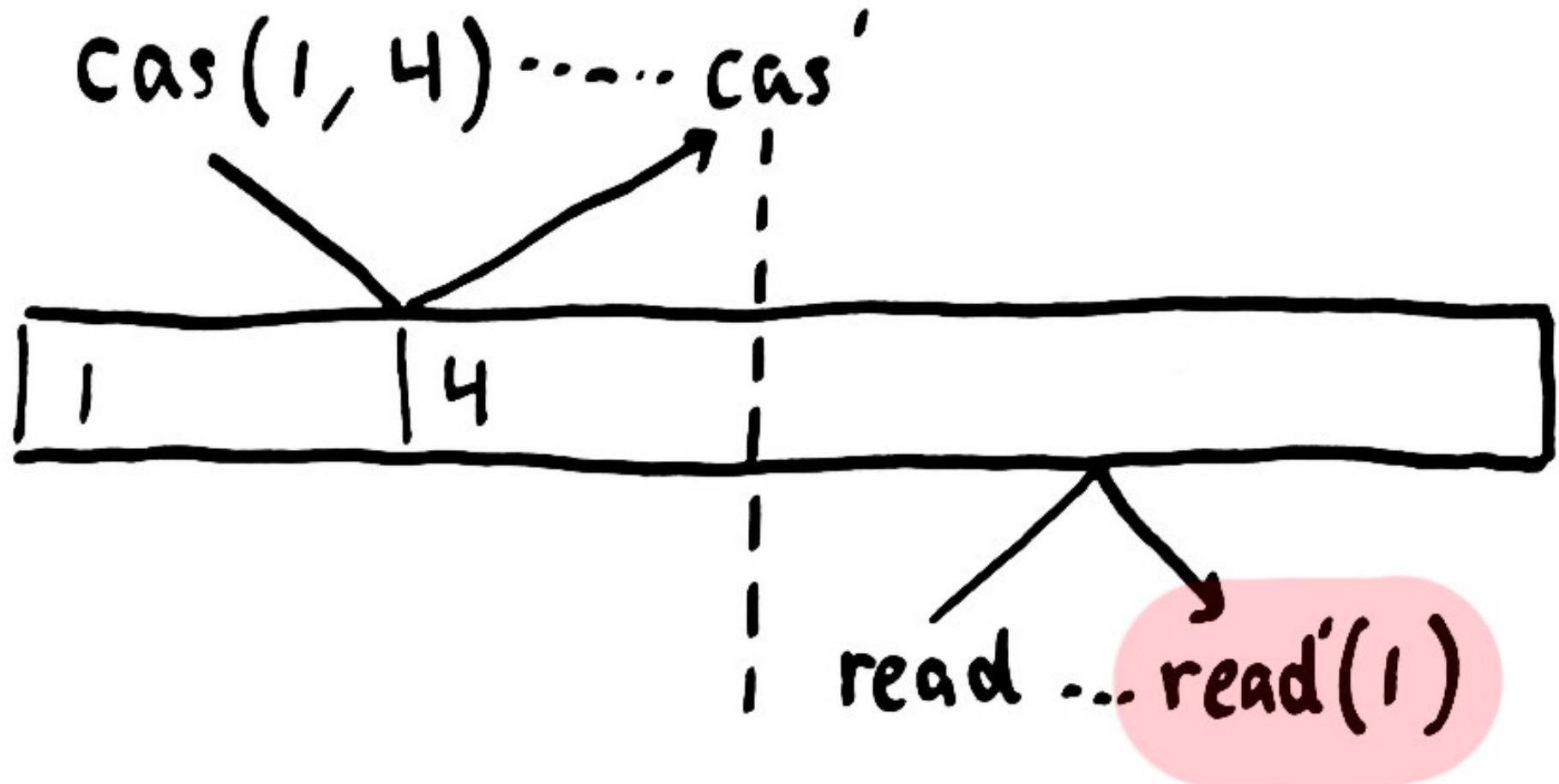
```
{:value 4}
```

with pending operations:

(and 12928 more worlds, elided here

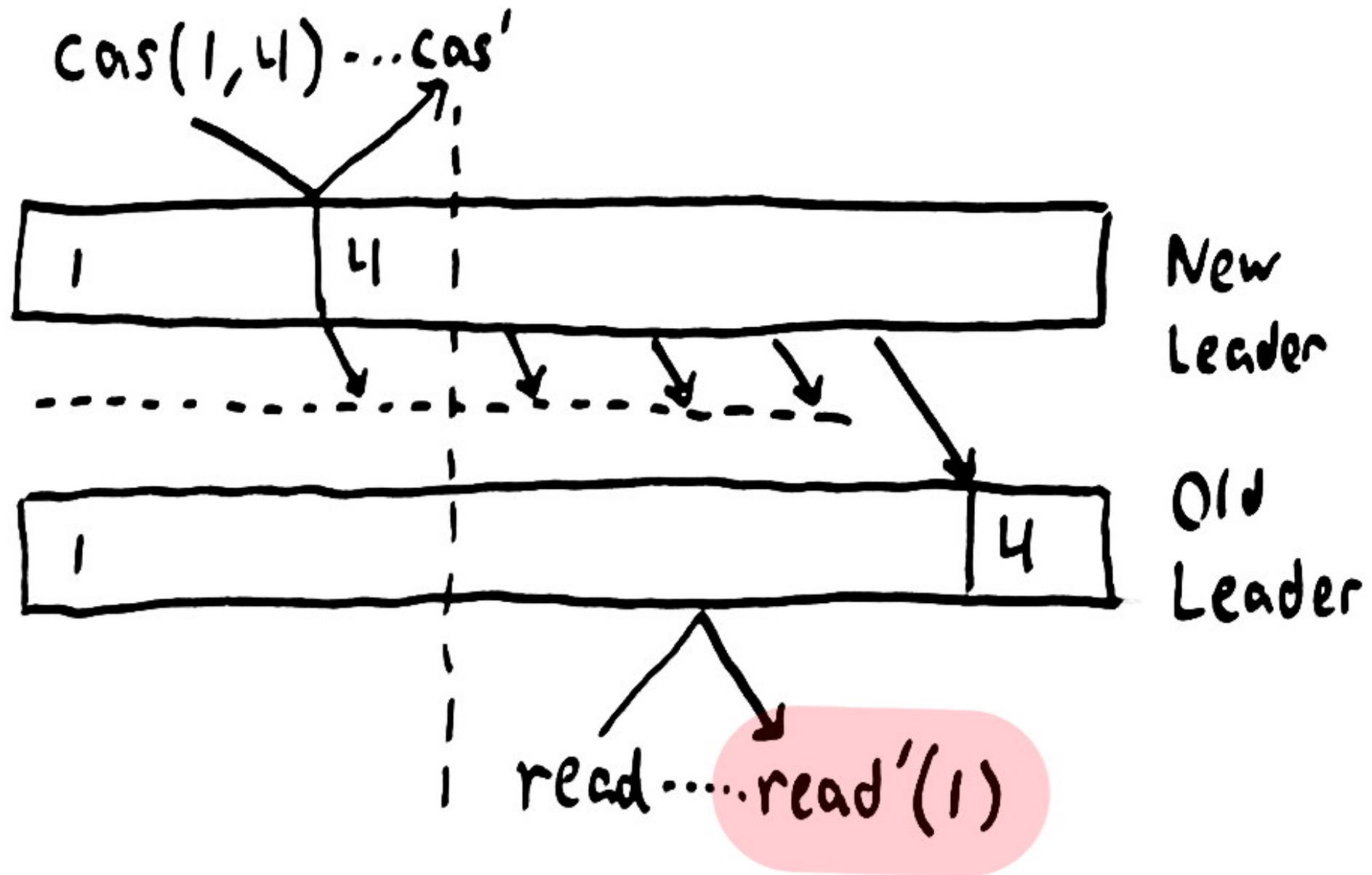
Inconsistent state transitions:

```
([{:value 4} "can't read 1 from register 4"])
```



Raft guarantees committed
log entries are linearizable.

But etcd "consistent" reads
don't go through the Raft
log! They read current
primary state & return!



Stale reads violate linearizability,
sequential, & causal consistency.

Also RYW. Also MW. Also MR.

As part of our Consul testing, we ran it against Jepsen to determine if any consistency issues could be uncovered. In our testing, Consul gracefully recovered from partitions without introducing any consistency issues.

As part of our Consul testing, we ran it against Jepsen to determine if any consistency issues could be uncovered. In our testing, Consul gracefully recovered from partitions without introducing any consistency issues.

```
LeaderLeaseTimeout: time.Second,
```

```
LeaderLeaseTimeout: 300 * time.Millisecond,
```

Suggestion: timeouts in a non-realtime system can't guarantee correctness, only improve odds.

→ Elasticsearch GC partitions...

Etcd & Consul have

now added support for

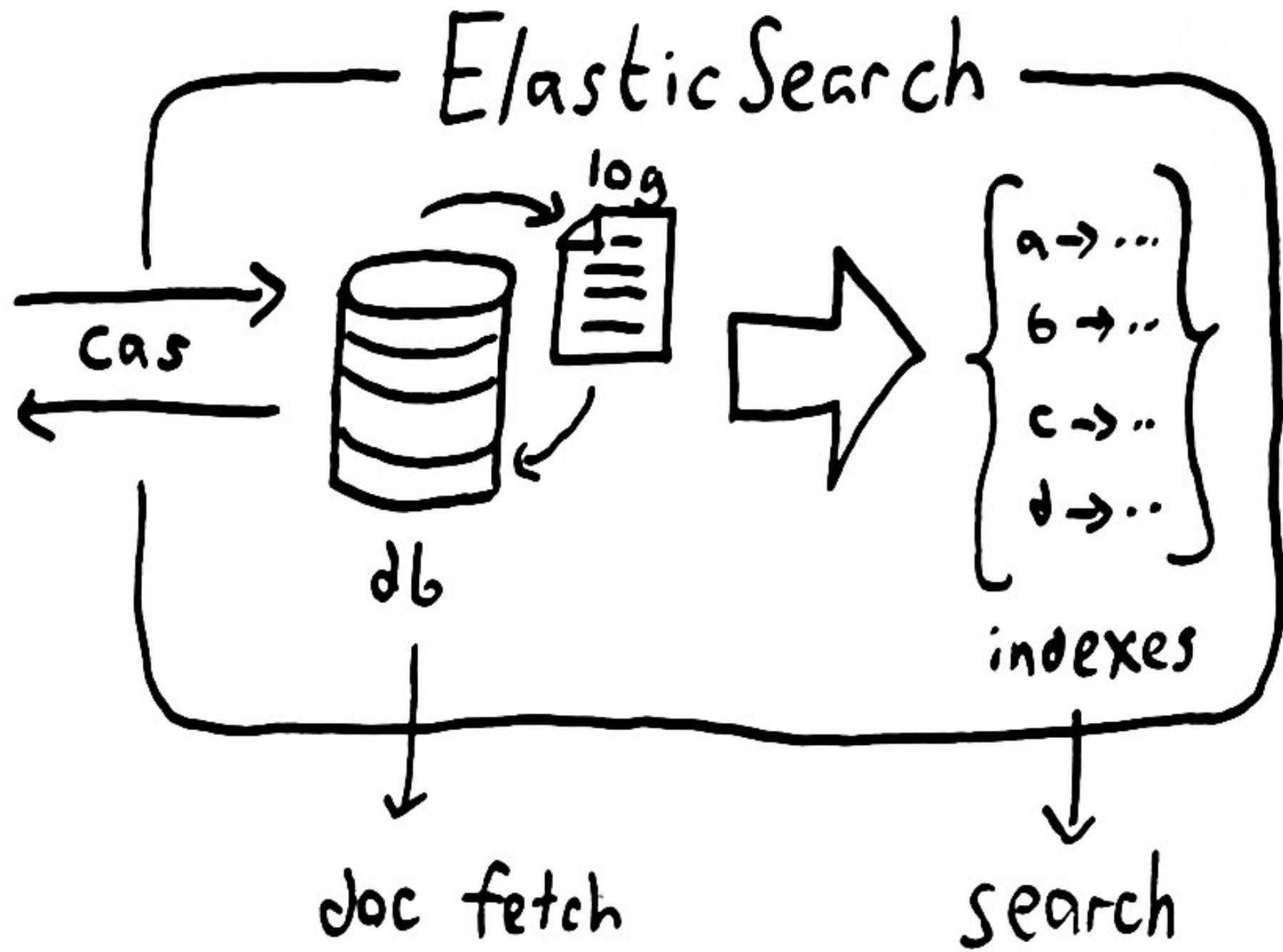
linearizable reads!



Rock on!

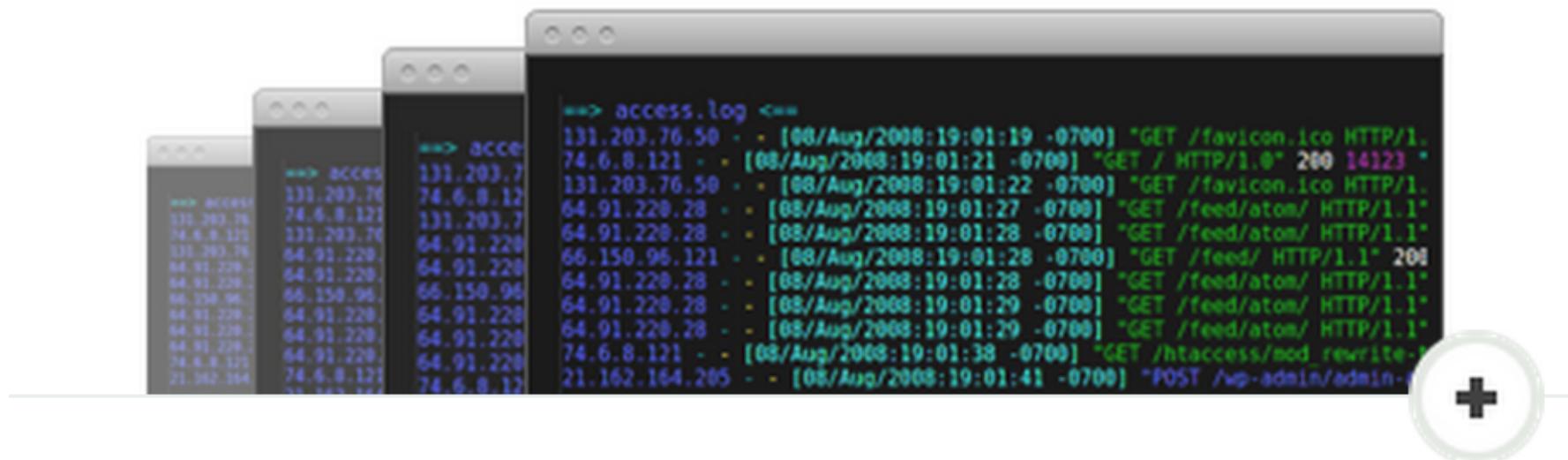
ElasticSearch!





per-operation persistence

Elasticsearch puts your data safety first. Document changes are recorded in transaction logs on multiple nodes in the cluster to minimize the chance of any data loss.



```
if (result.hasConflicts()) {  
    // TODO: What should we do???
```

Elasticsearch offers

atomic compare-and-set.

⇒ linearizability

write consistency



To prevent writes from taking place on the "wrong" side of a network partition, by default, index operations only succeed if a quorum ($>\text{replicas}/2+1$) of active shards are available. This default can be overridden on a node-by-node basis using the `action.write_consistency` setting. To alter this behavior per-operation, the `consistency` request parameter can be used.

[kimchy](#)

Jun 14, 2010;

[Reply](#) | [Threaded](#) | [More](#) ▾



Administrator
6265 posts

Hi,

When it comes to CAP, in a very high level, elasticsearch gives up on partition tolerance. This is for several reasons:

1. I personally believe that *within the same data center*, network partitions very rarely happen, and when they do, its a small set (many times single) machine that gets "partitioned out of the network". When a single machine gets disconnected from the network, then that's not going to affect elasticsearch. When it comes to cross data centers, a solution that gives up on consistency can be built on top of elasticsearch, either by elasticsearch (in the future), or now, by using messaging in some form to replicate changes between two data centers.



Jörg Prante

Jan 3



ES gives up on partition tolerance, it means, if enough nodes fail, cluster state turns red and ES does not proceed to operate on that index.

ES is not giving up on availability. Every request will be responded, either true (with result) or false (error). In a system being not available, you would have to expect the property of having some requests that can no longer be answered at all (they hang forever or the responder is gone).

"Instantaneous" cluster

Convergence \sim 90 seconds.

Cluster status
endpoint is
full of lies

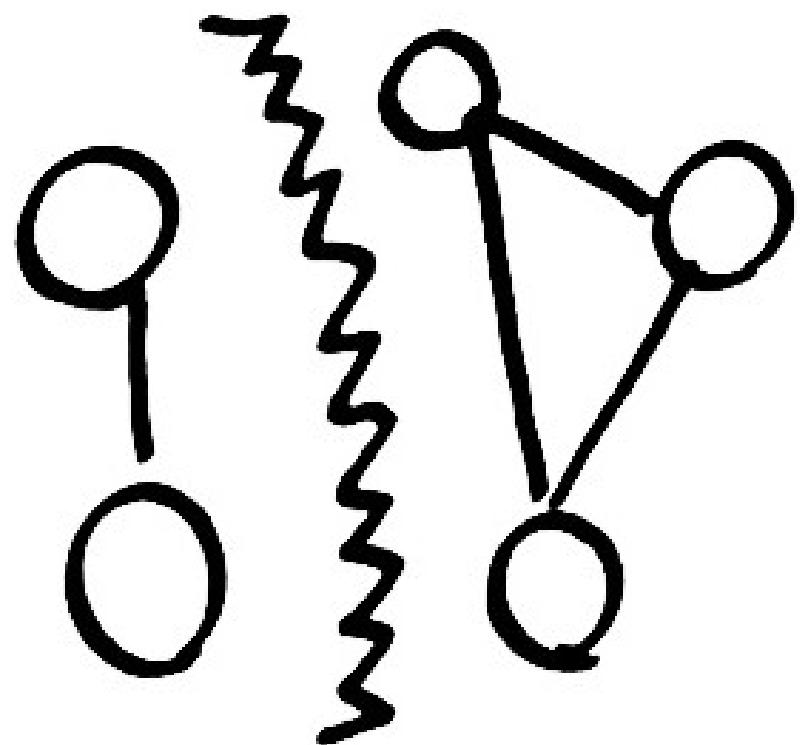
Short windows of write

loss during cluster transitions.

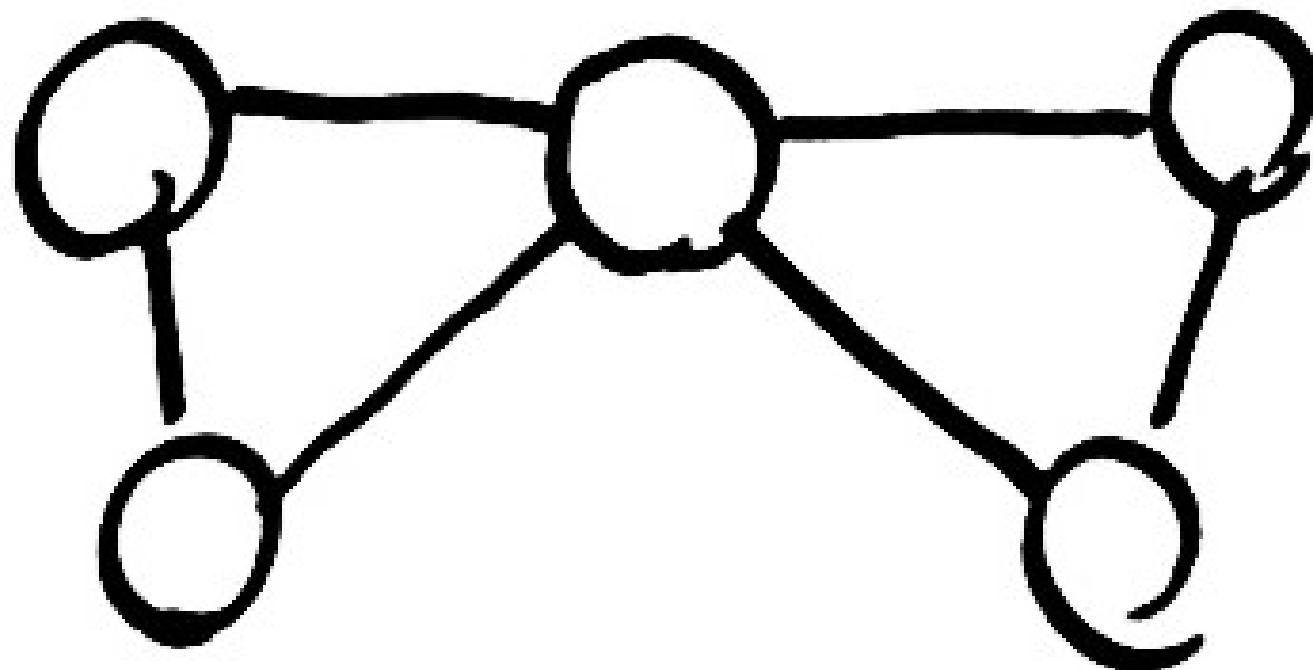
```
{:valid? false,
:ok      #{0..6 9..24 26..33 35..46 48 50..52 55..57 60..62 65
           67..68 71..73 76..78 81..83 86..88 91..93 96..98
           101..103 106..108 111..113 116..118 121..123
           126..128 131..133 136..138 141..143 146..148
           151..164 166 168..171 173..175 270..272 274},
:recovered #{50..52 55..57 60..62 65 67..68 71..73 76..78 81..83
            86..88 91..93 96..98 101..103 106..108 111..113
            116..118 121..123 126..128 131..133 136..138
            141..143 146..147},
:lost       #{176..178 250..251},
:unexpected #{},
:ok-fraC    26/55,
:recovered-fraC 59/275,
:lost-fraC   1/55,
:unexpected-fraC 0}
```

But wait –
There's More!

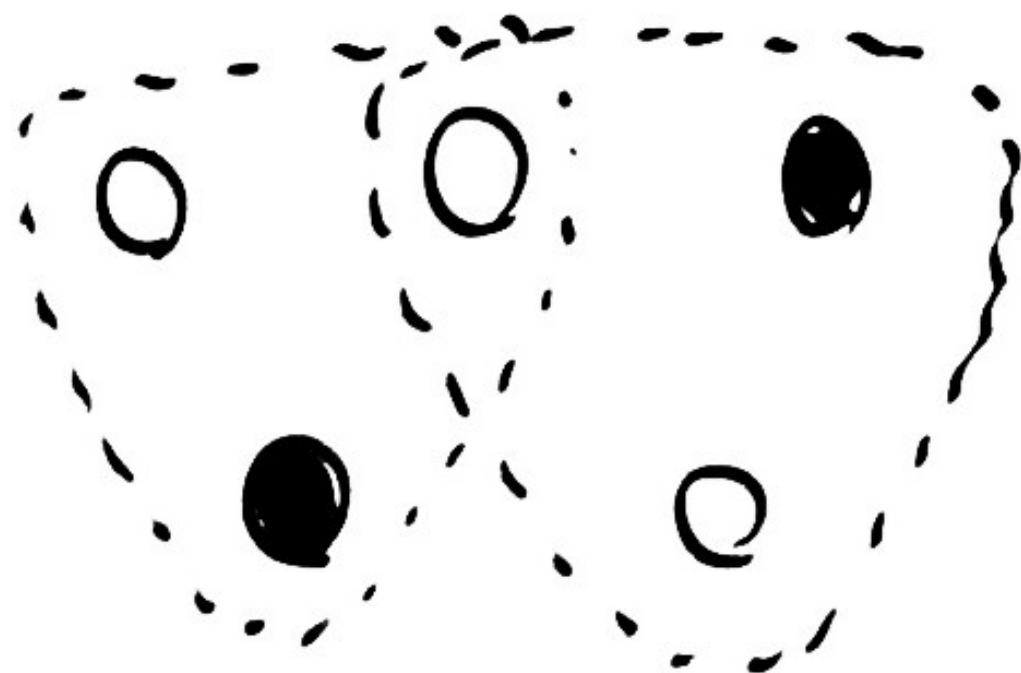
Assumes partitions are
symmetric & complete



What about..



Every node thinks it
has majority.



2 primaries

```
FAIL in (register-test) (elasticsearch_test.clj:49)
expected: (:valid? (:results test))
actual: false
{:valid? false,
 :ok ...,
 :recovered #{420 438 457 475},
 :lost    #{418..419 421..427 429..437 439..444 446..456 458..463 465..474
476..481 483..493 495..500 502..503 505..506 509..510 514 516 520..521 524 527 529
531 534 536 539 542 544 547..548 552 554 557..558 562 564 567..568 572 574 577 579
582..583 587..588 592..593 597 599 602 604 607 609 612 614 617 619 622 624 626 629
631 634 636 638 642 644 647 649 652..653 657 659 662 664 667 669 671 674 676 679 681
684 686 689 691 694 696 698 701 704 706 709 711 715..716 720..721 725 727 730 732
735 737 741..742 746..747 749 752 754 757..758 762..763 767 769 772 775 777 779
782..783}
```

...

```
1706..1707 1710 1712 1716..1717 1721..1722 1726..1727 1731..1732 1736..1737
1741..1742 1746..1747 1751..1752 1756 1761 1766 1771..1772 1776..1777 1781..1782
1786..1787 1791..1792 1796..1797 1801 1806..1807 1811..1812 1816..1817 1821
1826..1827 1831..1832 1836..1837 1841..1842 1846..1847 1851..1852 1856..1857
1861..1862 1866..1867 1871..1872 1877..1878 1882..1883 1887..1888 1892..1893
1897..1898 1902..1903 1907 1912..1913 1917..1918 1922..1923 1926 1931..1932
1936..1937 1941 1946..1947 1951 1953 1957},
:unexpected      #{},
:ok-frac        1103/1961
:recovered-frac 4/1961,
:lost-frac      645/1961,
:unexpected-frac 0})
```

But inserts are
still safe, right?

```
: recovered-frac 11/289,  
: unexpected-frac 0,  
: unexpected "#{}",  
: lost-frac 688/2023,  
: ok-frac 1265/2023}
```

```
"RemoteTransportException[[Cadaver]
[inet[/192.168.122.12:9300]][index]]; nested:
RemoteTransportException[[Death Adder]
[inet[/192.168.122.11:9300]][index]]; nested:
DocumentAlreadyExistsException[[jepsen-index][1]
[number][EpVU56YERB0fRqyVc-_hAg]: document already
exists];
```

The response is similar to what we saw before, except that the `_id` field has been generated for us:

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "wM00SFhDQXGZAWDf0-drSA",
  "_version": 1,
  "created": true
}
```

Auto-generated IDs are 22 character long, URL-safe, Base64-encoded string *universally unique identifiers, or UUIDs*.

flow about total
partitions?



```
:total 461,  
:recovered-count 73,  
:unexpected-count 0,  
:lost-count 17,  
:ok-count 319
```

Ok but isolating
a single node
is safe, right?

```
: recovered-count 66,  
: unexpected-count 0,  
: unexpected "#{}",  
: lost-count 8,  
: ok-count 541}}}
```

"reason [do not exists
on master, act as
master failure]"

wedged GLISTER

STATE

Even a single, one-node failure, can cause data loss.

We've seen data loss
in prod continuously.

Reboot a node? Loss.

Shard rebalance? Loss.

Multiple production user
reports - caused by
network failure or GC.

“We’ve had some bad networking, some Virtual Machine stalls (result of SAN issues, or VMWare doing weird stuff), or even heavy GC activity can cause enough pauses for aspects of the split brain to occur.”

--Paul Smith



tallpsmith

11 months ago

suffered from this the other day when an accidental provisioning error had a 4GB ES Heap instance running on a 4GB O/S memory, which was always going to end up in trouble. The node swapped, process hung, and the intersection issue described here happened.

I am experiencing at random time spike in heap usage followed by long GC.

after the nodes finished the long GC cluster is getting into split brain situation with weird states where one of the cluster nodes is member of both sides of the split. `minimum_master_nodes` option does not help in this case since the node exists in two of the different cluster states.



trevorreeves

6 months ago

We have been frequently experiencing this 'mix brain' issue in several of our clusters - up to 3 or 4 times a week. We have always had dedicated master eligible nodes (i.e. `master=true, data=false`), correctly configured `minimum_master_nodes` and have recently moved to 0.90.3, and seen no improvement in the situation.

Our situation appears to be IOWait related, in that a master node (also a data-node) hits an issue that causes extensive IOWait (a scroll based search can trigger this, we already cap the # streams and Mb/second recovery rate through settings), the JVM becomes unresponsive. The other nodes that are doing the Master Fault Detection are configured with 3 x 30 second ping timeouts, all of which fail, and then they give up on the master.

For some reason, node A and B cannot talk to each other... but both can still talk to C and C can talk to A and B i.e. a ‘on the fence’ network partition as C can still see all...

--Mark Tinsley

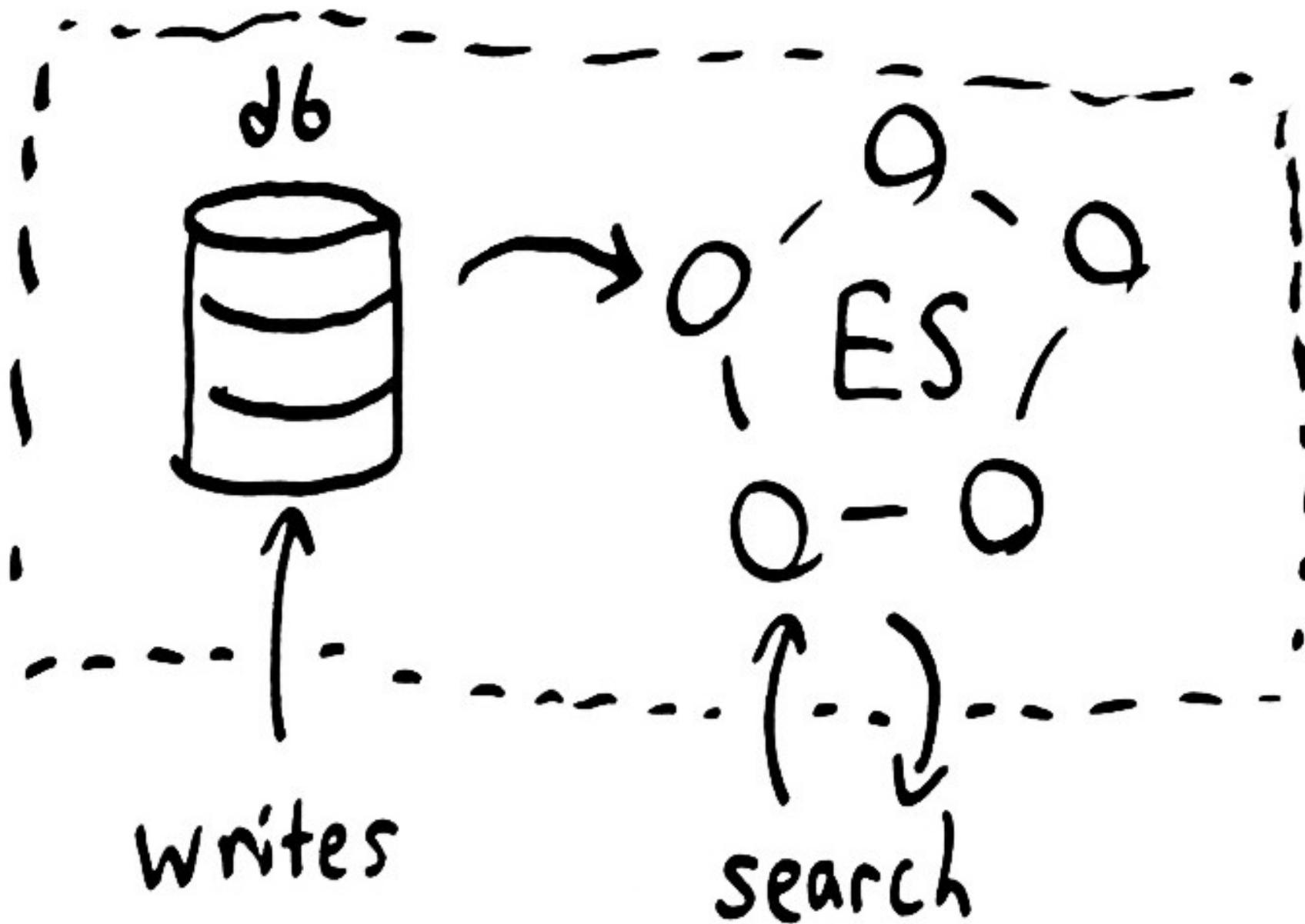
Multiple-principles ticket
is now closed, but
data-loss issues in ES
are pervasive.

ES is focusing on single-node stability improvements, choose not to adopt a real consensus algorithm.

Zendisco is terrible.

Use zk instead!

Always treat ES as a
derivative datastore. Keep
Primary data somewhere
safe, & periodically resync.



In summary...

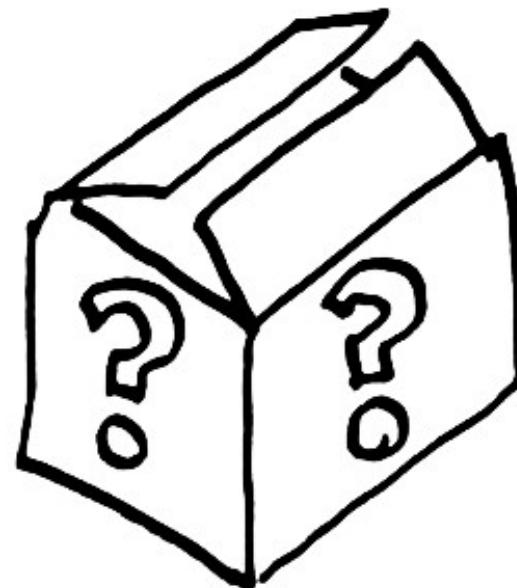
Jepsen tells you about

1 Particular

SYSTEM

Is YOUR system

safe? I dunno!



Consider...

- Runtime pauses
- Network slowness
- One-way partitions

"How do we rely
on clocks?"



Clients are
a part of the
distributed system!

Reason carefully about state
transitions with critical impact

How much safety do
you really need?

Data loss or
inconsistency can
be OK!

Balance

Complexity

— with —

Correctness

Measure
your
Systems

Use iptables to simulate
failures in prod, and record
results!

- Most systems have gone untested under failure
- You'd be surprised what well-designed systems will do during a network partition
- Test your systems.

Thanks!

Shay Banoh

Camille Fournier

Coda Hale

Shikhar Bhushan

Jeff Hodges

Evan Phoenix

Alex P

Cliff Moon

JD Maturan

Jared Morrow

Ryan Zeeski

Blake Mizerany

Michael Klishnin

Thanks!

Peter Bailis

Aaron France

Bob Poekert

Alvaro Videla

Kelly Sommers

Jacques Fuentes

Armon Dadgar

Xiang Li

Joseph Blomstedt

Tyler Schuett

Comcast + Factual + et al

