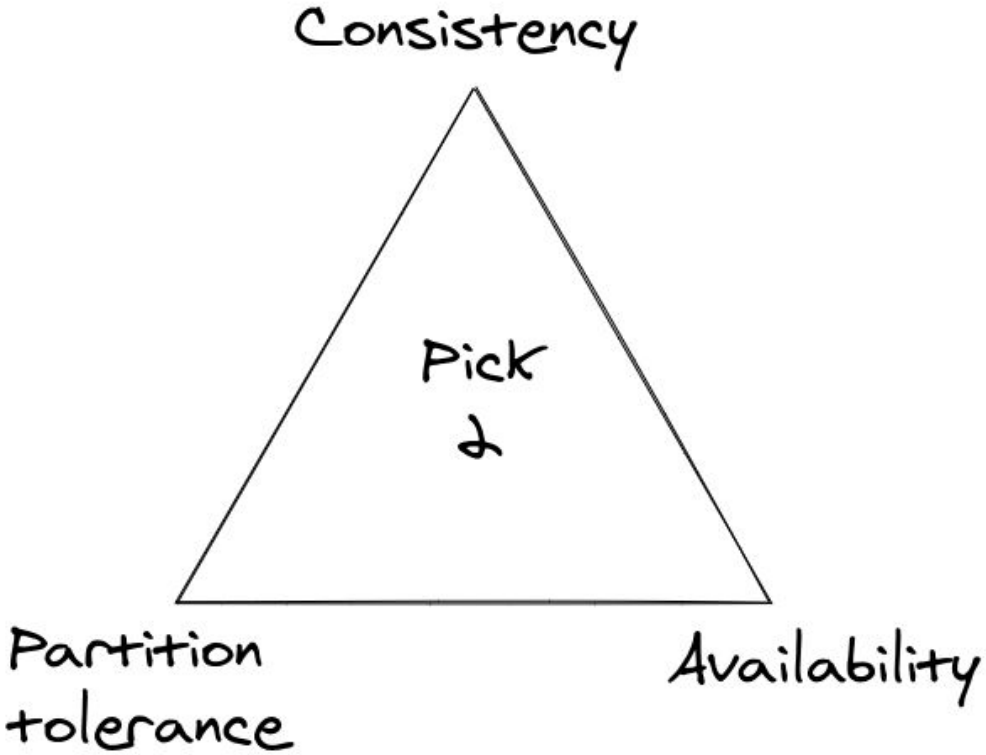
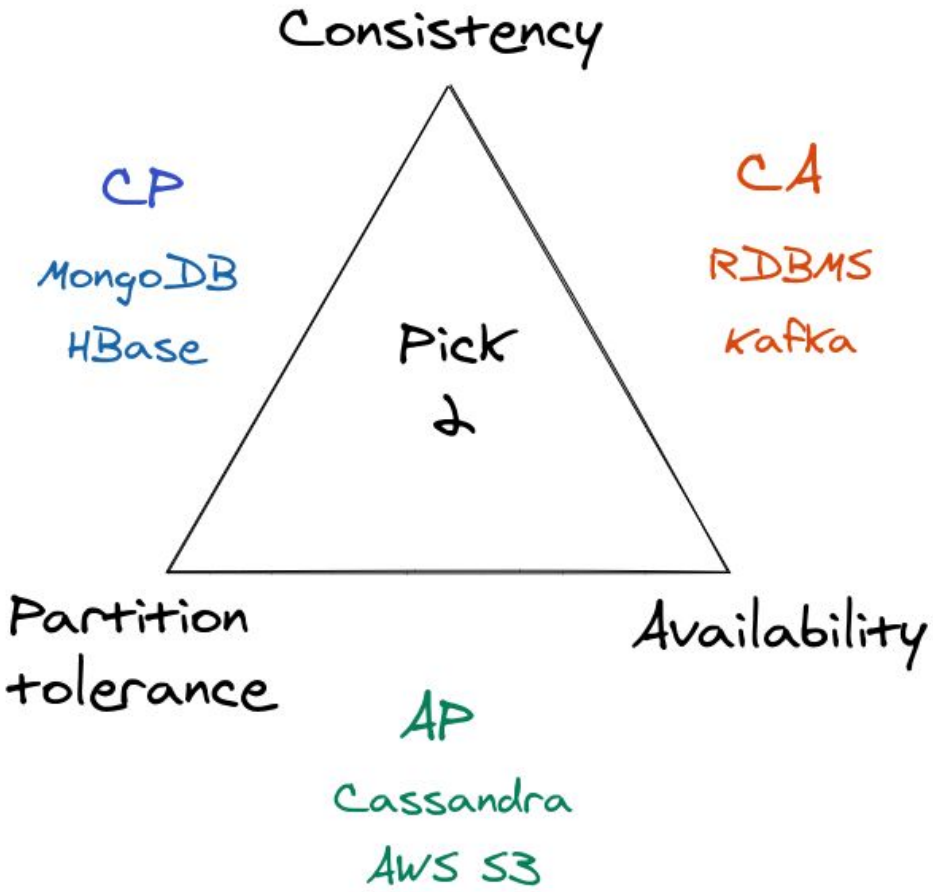
CAP Theorem - two decades and few clouds later

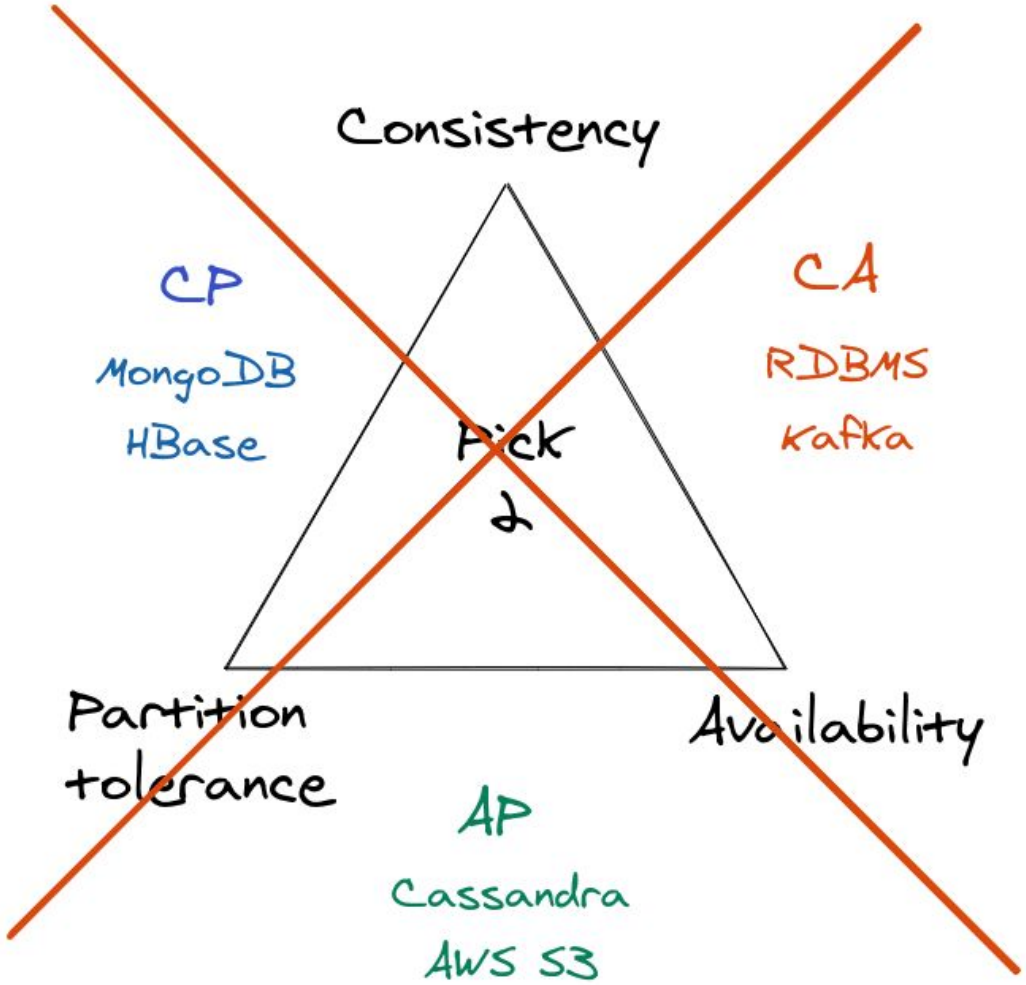
@mikemybytes


Consistency
Pick
2
Partition
tolerance
Availability

Triangle makes CAP easy to remember

Triangle has as much CAP as CAPpuccino


Consistency
CP
MongoDB
HBase
CA
RDBMS
Kafka
Pick
2
Partition
tolerance
Availability
AP
Cassandra
AWS S3


Consistency
CP
MongoDB
HBase
CA
RDBMS
Kafka
Pick
2
Partition
tolerance
Availability
AP
Cassandra
AWS S3

aws

Contact Us Support English My Account Create an AWS Account

Products Solutions Pricing Documentation Learn Partner Network AWS Marketplace Customer Enablement

Amazon S3

Overview Features Storage classes Pricing Security Resources FAQs

Products / Storage / Amazon S3 / Amazon S3 Features / ...

Amazon S3 Strong Consistency

Amazon S3 delivers strong read-after-write consistency automatically for all applications, without changes to performance or availability, without sacrificing regional isolation for applications, and at no additional cost. With strong consistency, S3 simplifies the migration of on-premises analytics workloads by removing the need to make changes to applications, and reduces costs by removing the need for extra infrastructure to provide strong consistency.

After a successful write of a new object, or an overwrite or delete of an existing object, any subsequent read request immediately receives the latest

Introducing strong consistency for Amazon S3, featuring Dropbox (4:36)

https://aws.amazon.com/s3/consistency/

CAP: take two
AWS: take them all!

Has the cloud changed the rules?

CAP is proven

CAP is misleading

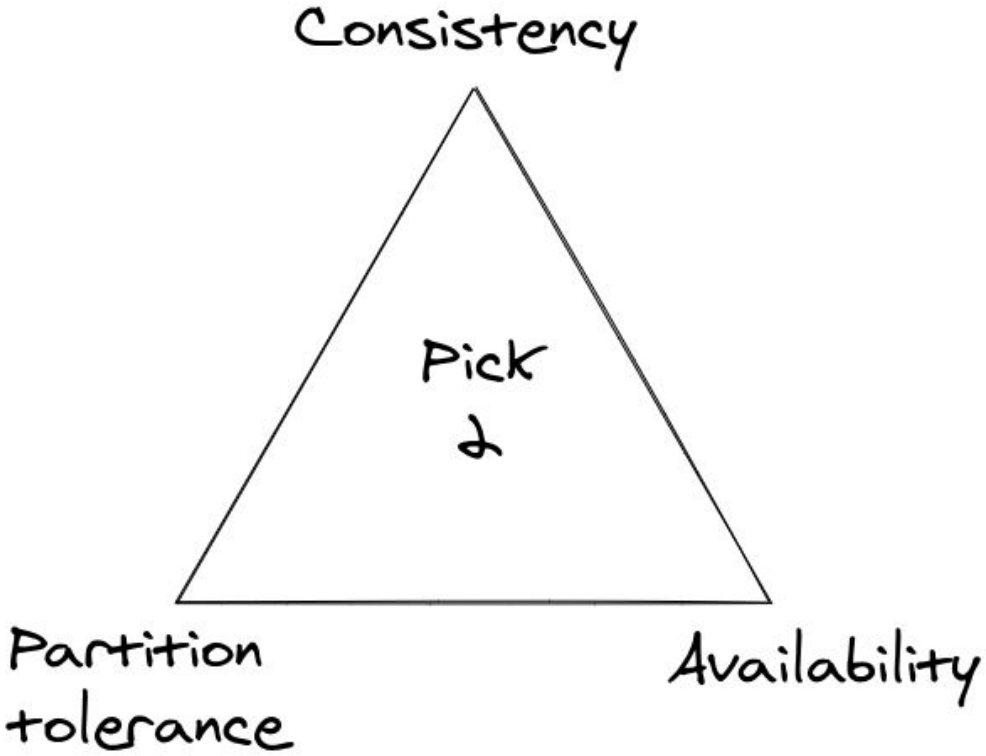
What’s left?

Should we still care?

What about the tools?

That’s what this talk is all about!

Quick reCAP

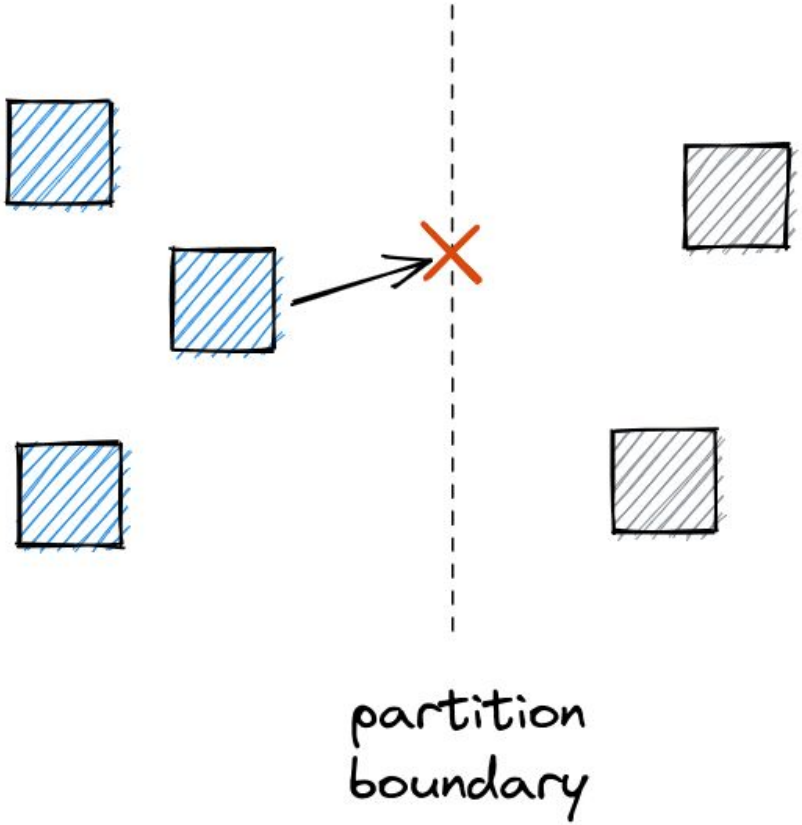

Consistency
Pick
2
Partition
tolerance
Availability

Availability

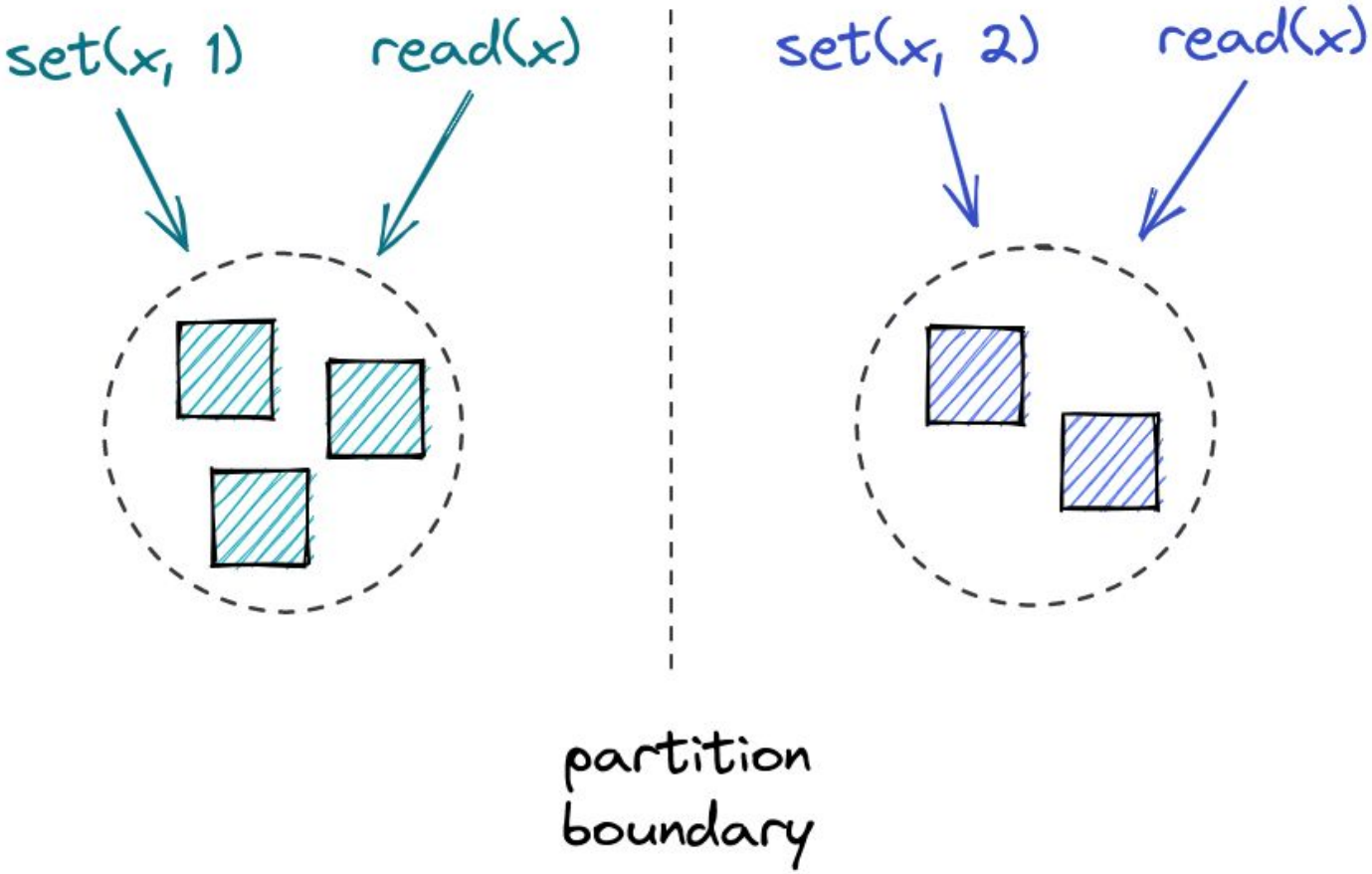
Every request received by a non-failing node...

...must result in
a (non-error) response

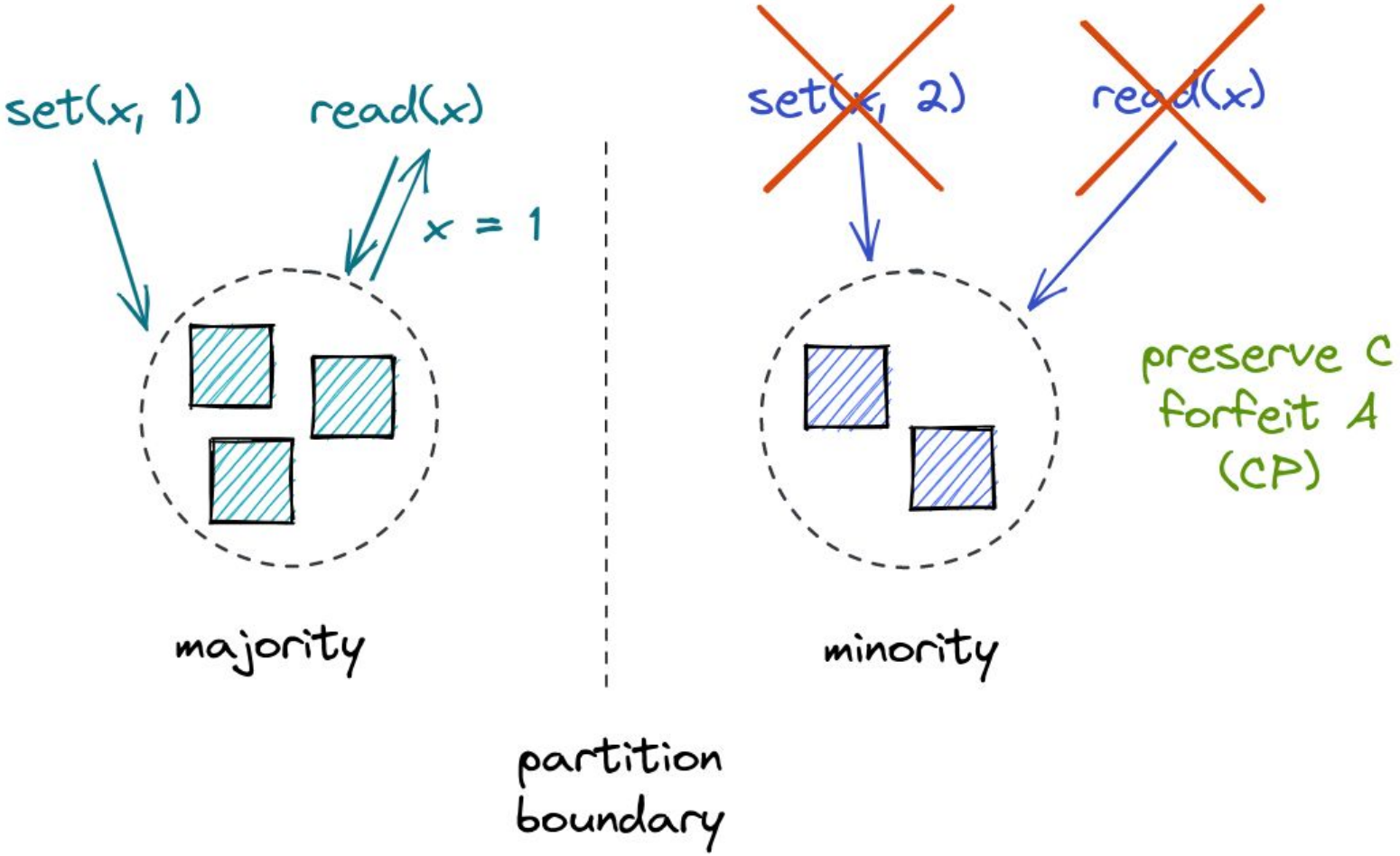
Network partition


partition
boundary

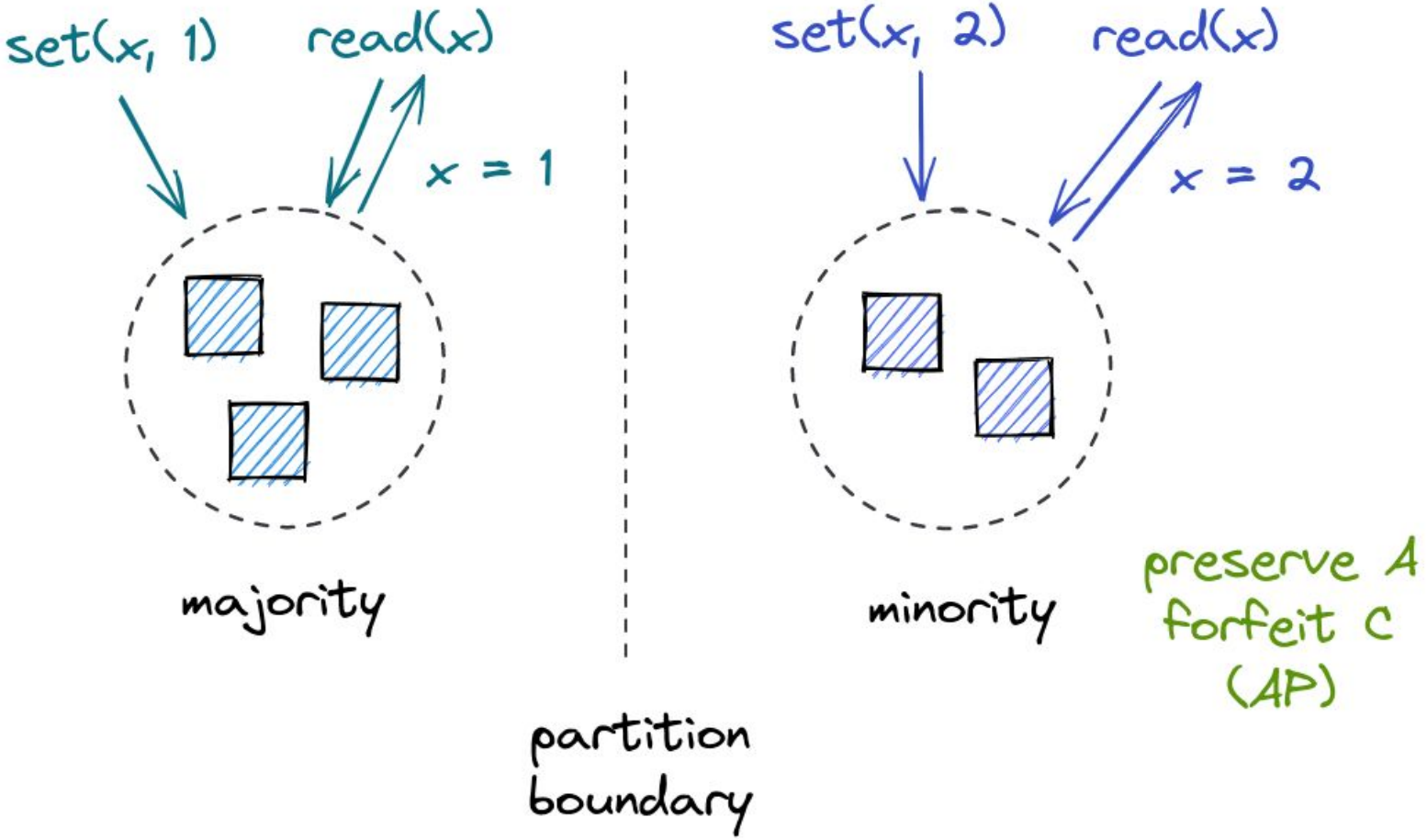
Partitions are not something we choose!


set(x, 1)
read(x)
set(x, 2)
read(x)
partition
boundary

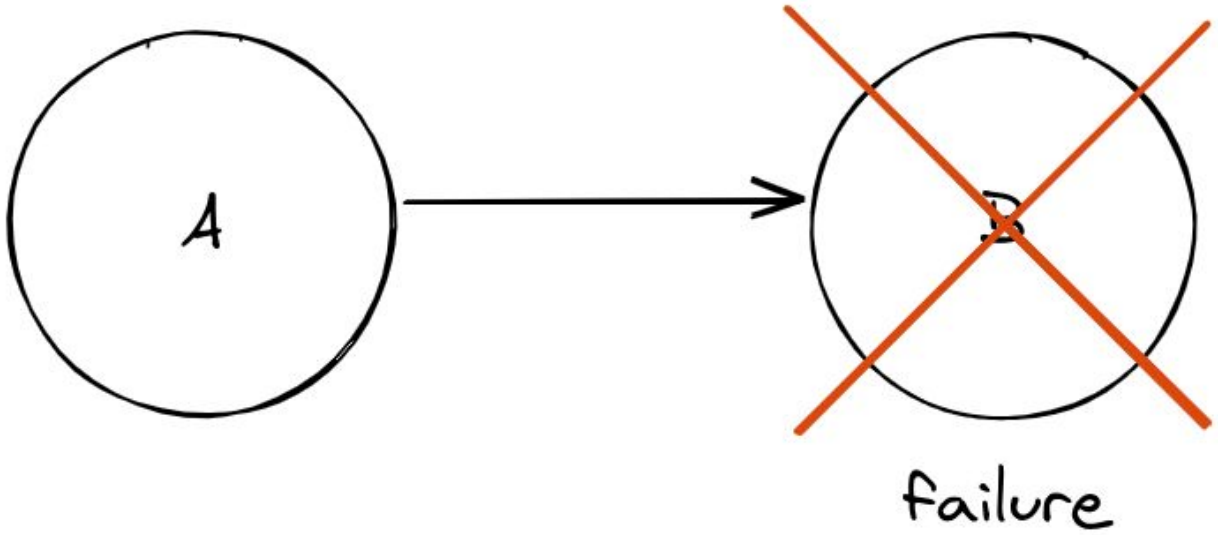
CAP is about the decision making


set(x, 1)
read(x)
x = 1
majority
partition
boundary
set(x, 2)
read(x)
minority
preserve C
forfeit A
(CP)

Single inconsistent read invalidates C

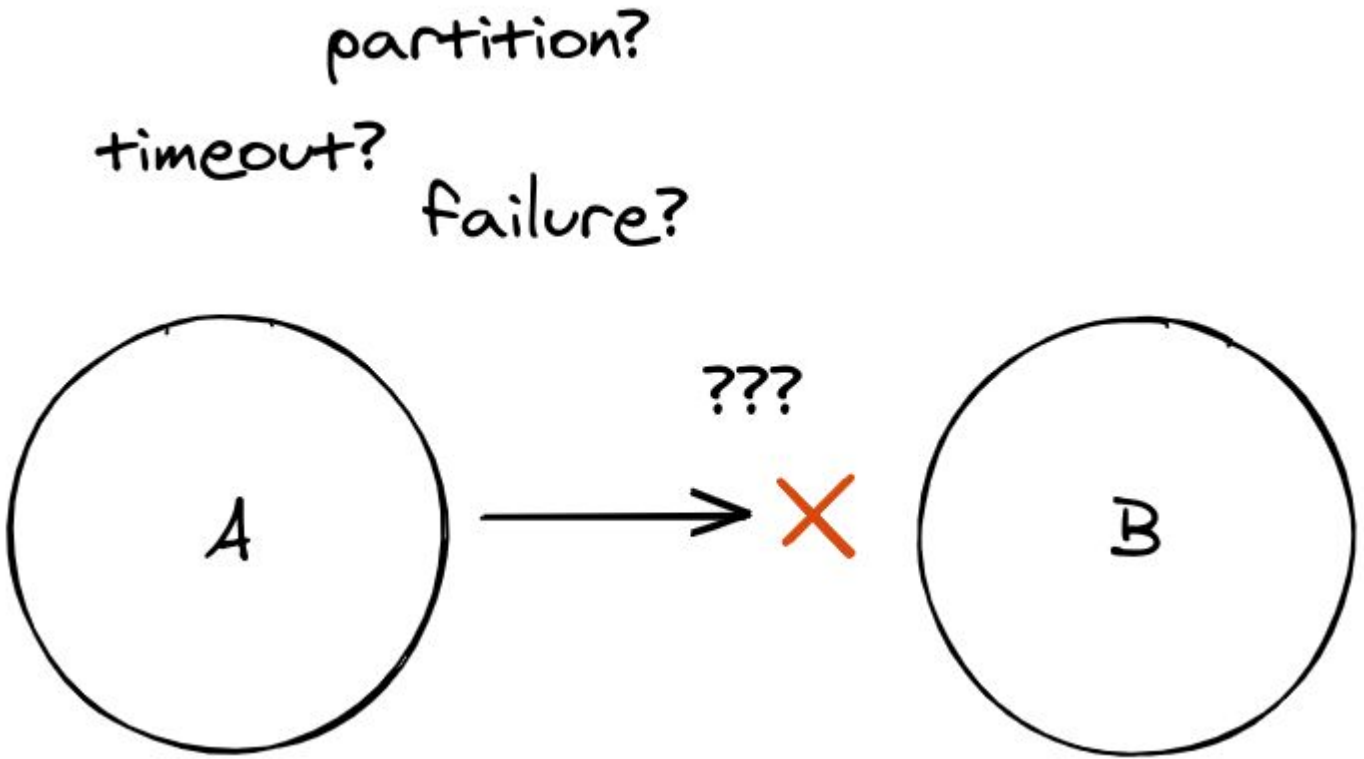

set(x, 1)
read(x)
x = 1
majority
partition
boundary
set(x, 2)
read(x)
x = 2
minority
preserve A
forfeit C
(AP)

What about node failures?


A
B
failure

Node failure is not a network partition!

Failures/downtimes happen frequently


partition?
timeout?
failure?
???
A
B

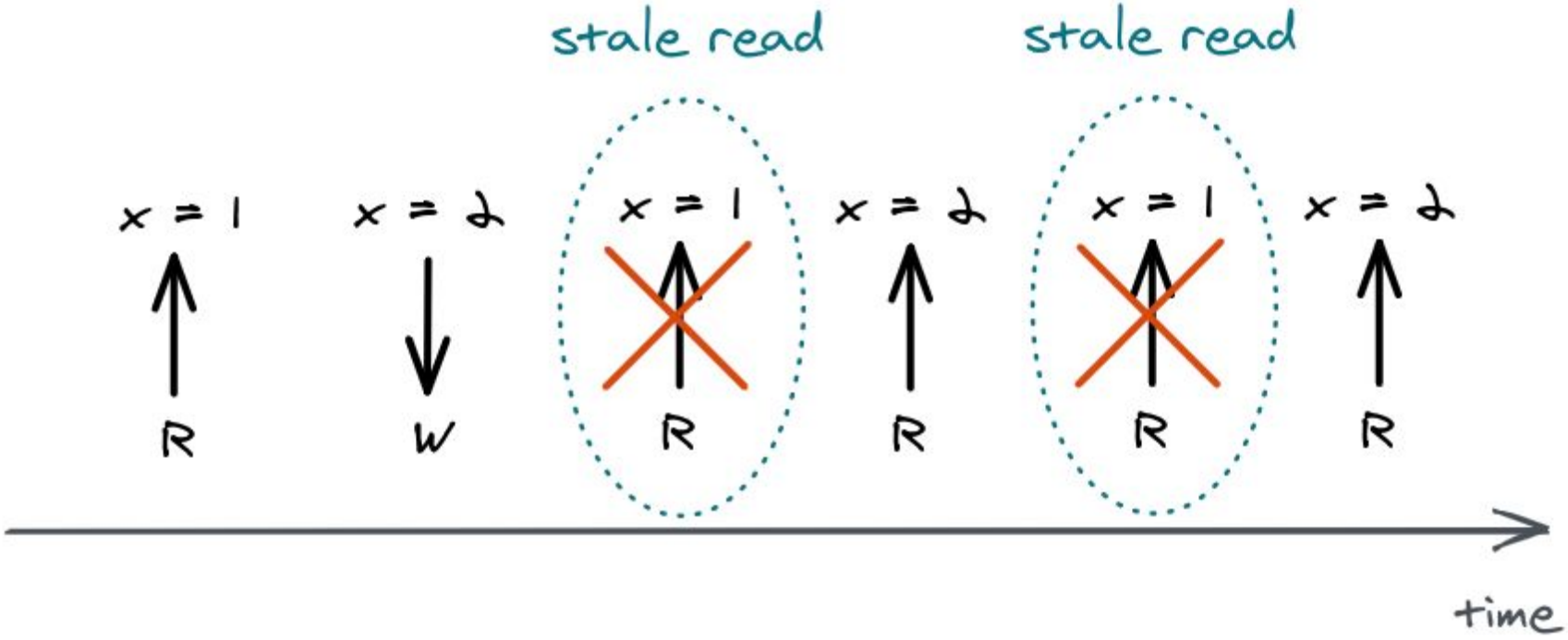
“P strategy” has consequences!

Consistency

Linearizability
(strict consistency)

All operations behave as if they were executed atomically...

…on a single copy of data (single node)

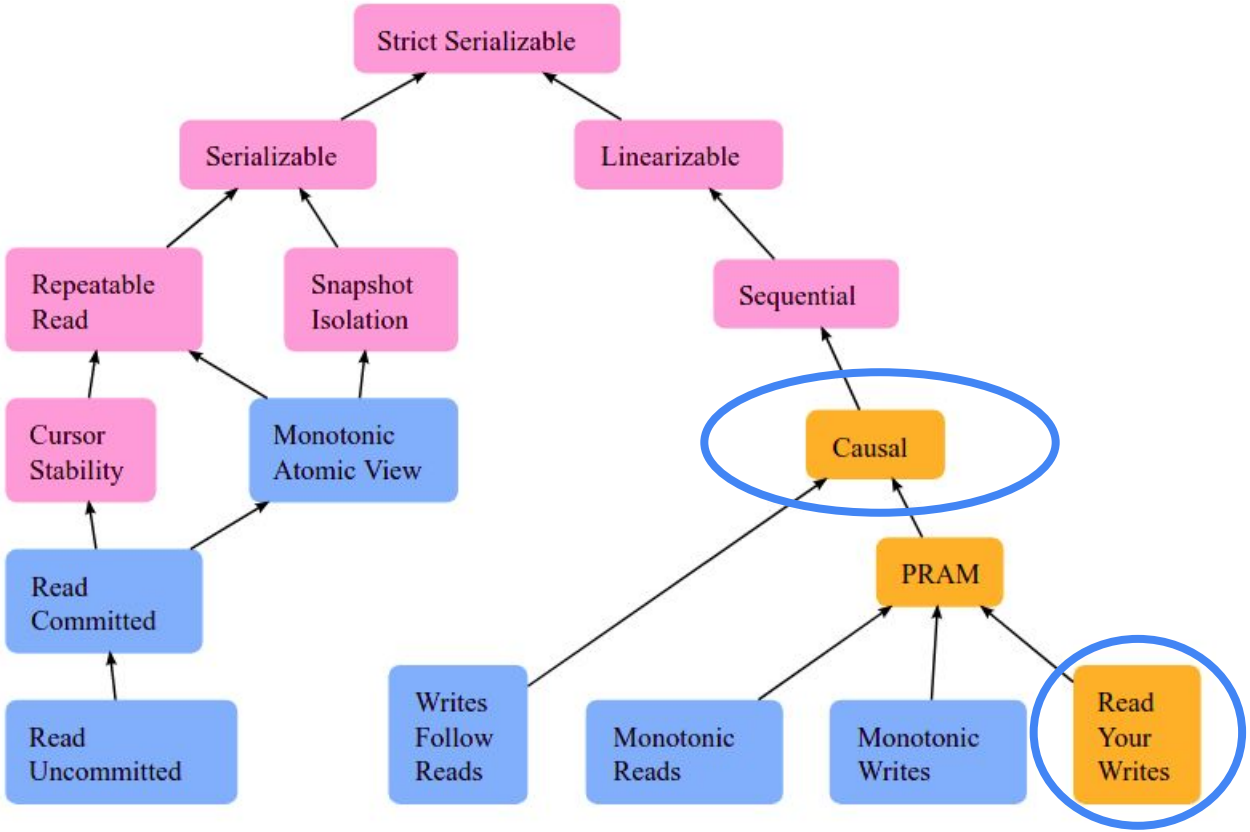

stale read
stale read
x = 1
x = 2
x = 1
x = 2
x = 1
x = 2
R
W
R
R
R
R
time

Linearizability is a strong guarantee

What about non-C systems?

Consistency is a spectrum of models and guarantees

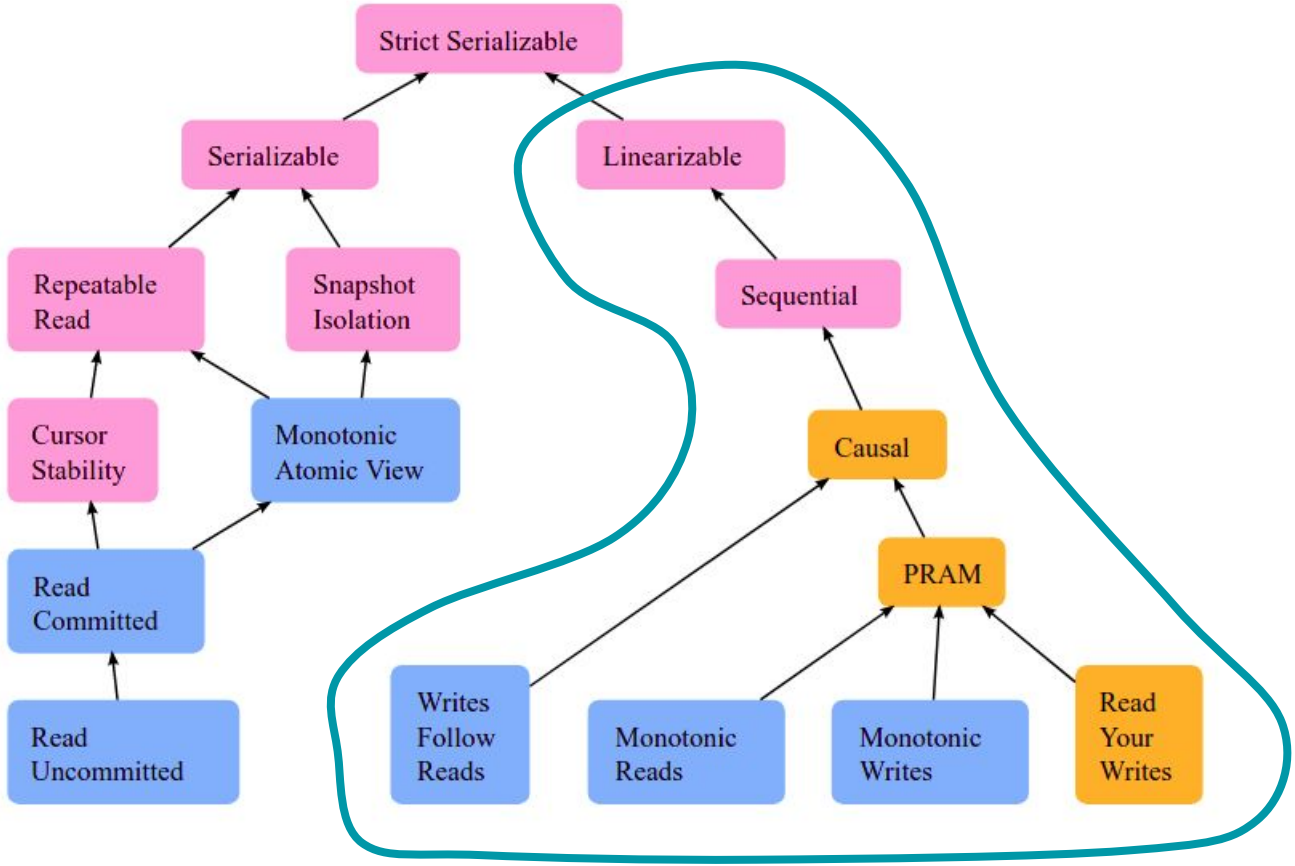
Performance & fault tolerance trade-offs



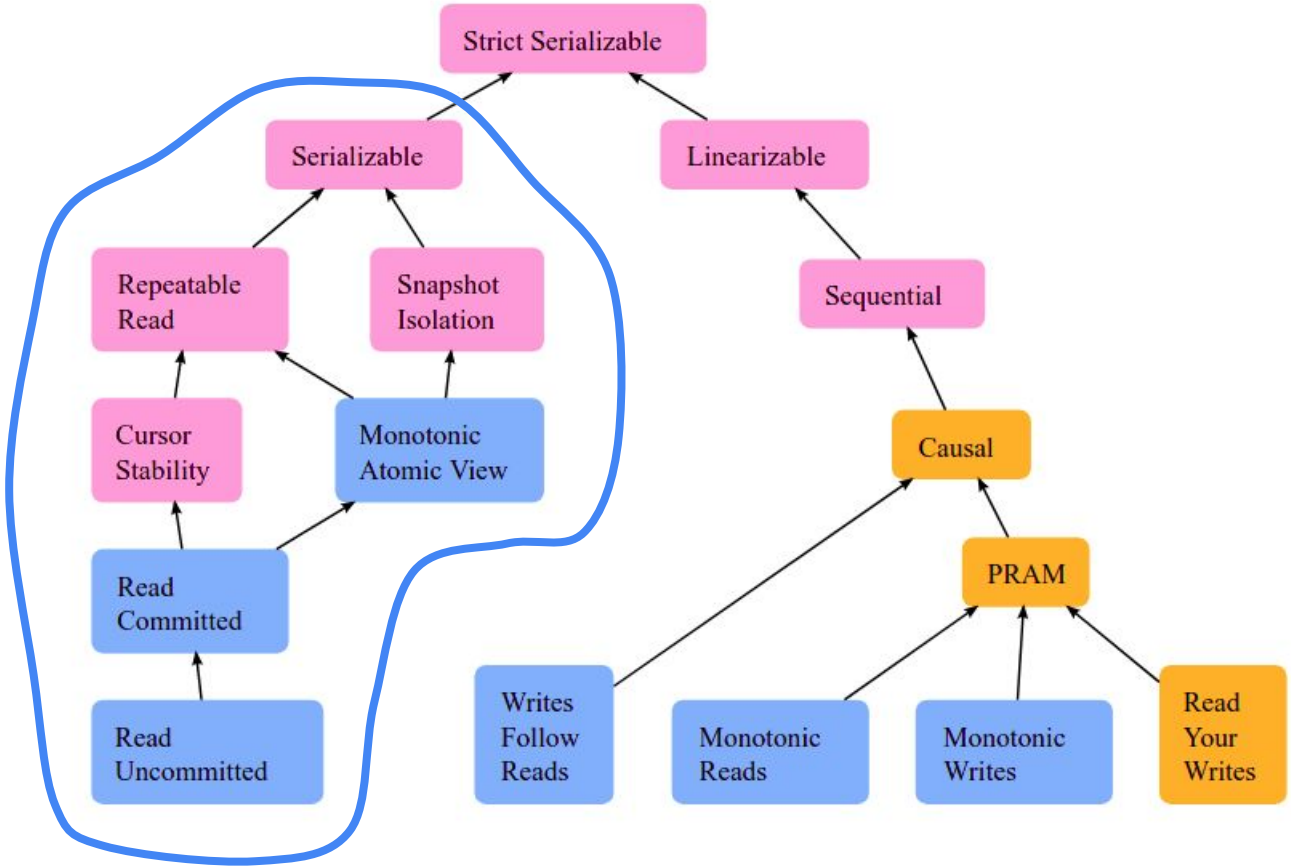
https://jepsen.io/consistency

Eventual consistency is not “no consistency”

What about ACID?

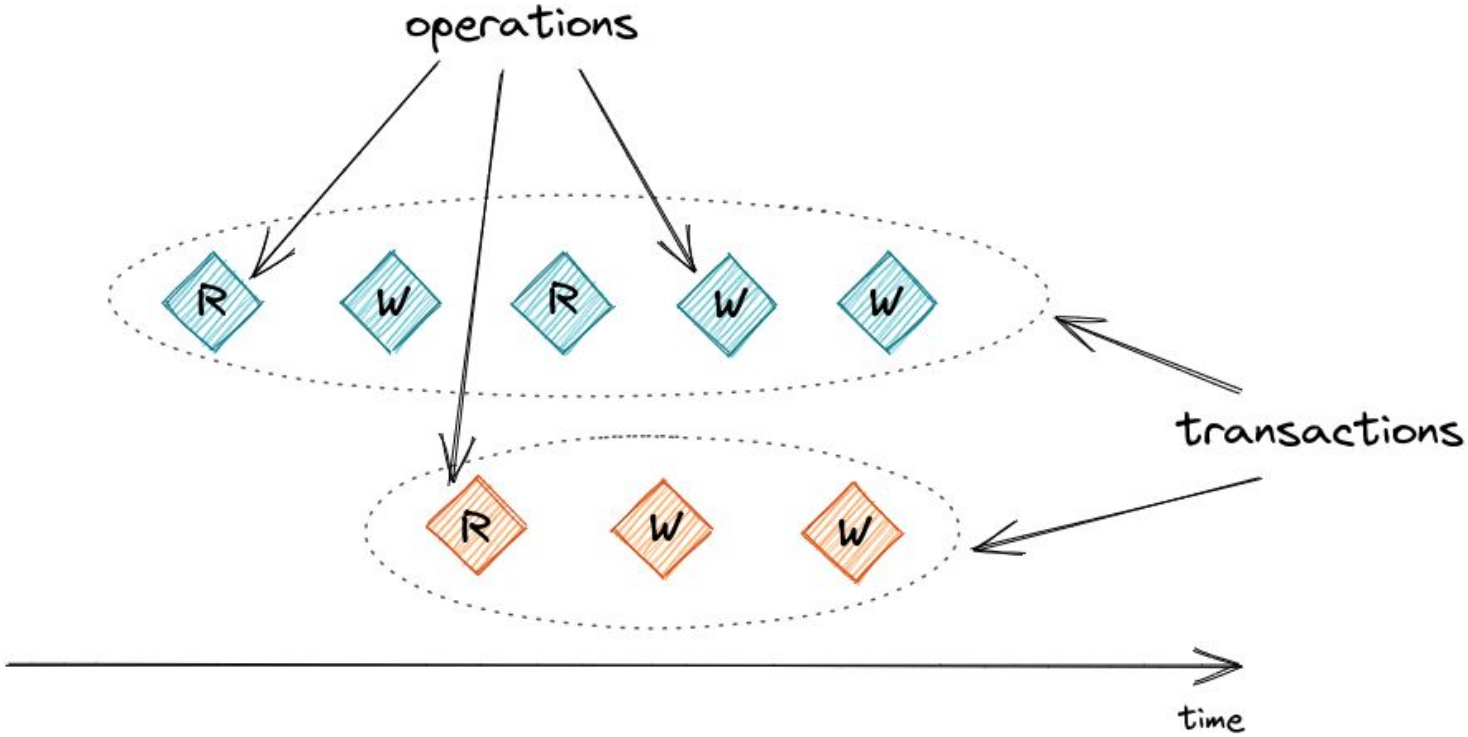


https://jepsen.io/consistency



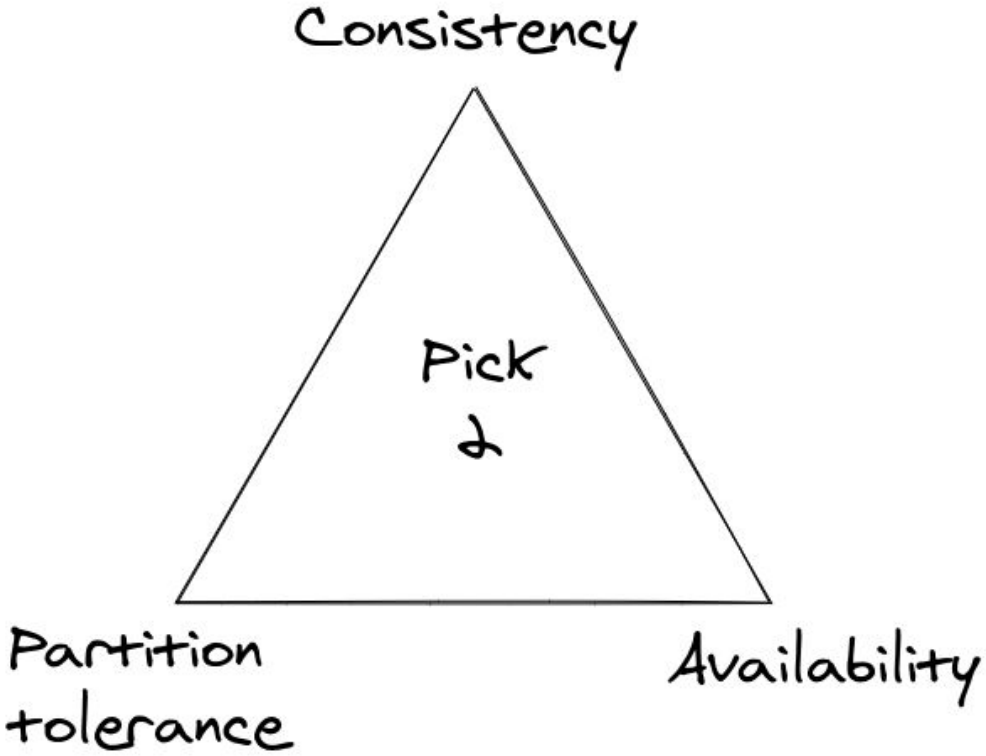
https://jepsen.io/consistency

ACID: Transactions
CAP: Operations


operations
R
W
R
W
W
R
W
W
transactions
time

ACID Consistency is about the invariants

Do not confuse ACID with CAP!


Consistency
Pick
2
Partition
tolerance
Availability

CAP: When P, system must choose between A & C (linearizability)

AWS S3

Cloud hash-table for blobs (key-value)

Buckets & objects

Redundant by default

Before 2020: eventually consistent

aws

Contact Us Support English My Account Create an AWS Account

Products Solutions Pricing Documentation Learn Partner Network AWS Marketplace Customer Enablement

Amazon S3

Overview Features Storage classes Pricing Security Resources FAQs

Products / Storage / Amazon S3 / Amazon S3 Features / ...

Amazon S3 Strong Consistency

Amazon S3 delivers strong read-after-write consistency automatically for all applications, without changes to performance or availability, without sacrificing regional isolation for applications, and at no additional cost. With strong consistency, S3 simplifies the migration of on-premises analytics workloads by removing the need to make changes to applications, and reduces costs by removing the need for extra infrastructure to provide strong consistency.

After a successful write of a new object, or an overwrite or delete of an existing object, any subsequent read request immediately receives the latest

Introducing strong consistency for Amazon S3, featuring Dropbox (4:36)

https://aws.amazon.com/s3/consistency/

How much CAP does it have now?

Amazon S3 data consistency model

Amazon S3 provides strong read-after-write consistency for PUTs and DELETEs of objects in your Amazon S3 bucket in all AWS Regions. This applies to both writes to new objects as well as PUTs that overwrite existing objects and DELETEs. In addition, read operations on Amazon S3 Select, Amazon S3 Access Control Lists, Amazon S3 Object Tags, and object metadata (e.g. HEAD object) are strongly consistent.

https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html#ConsistencyModel

Note

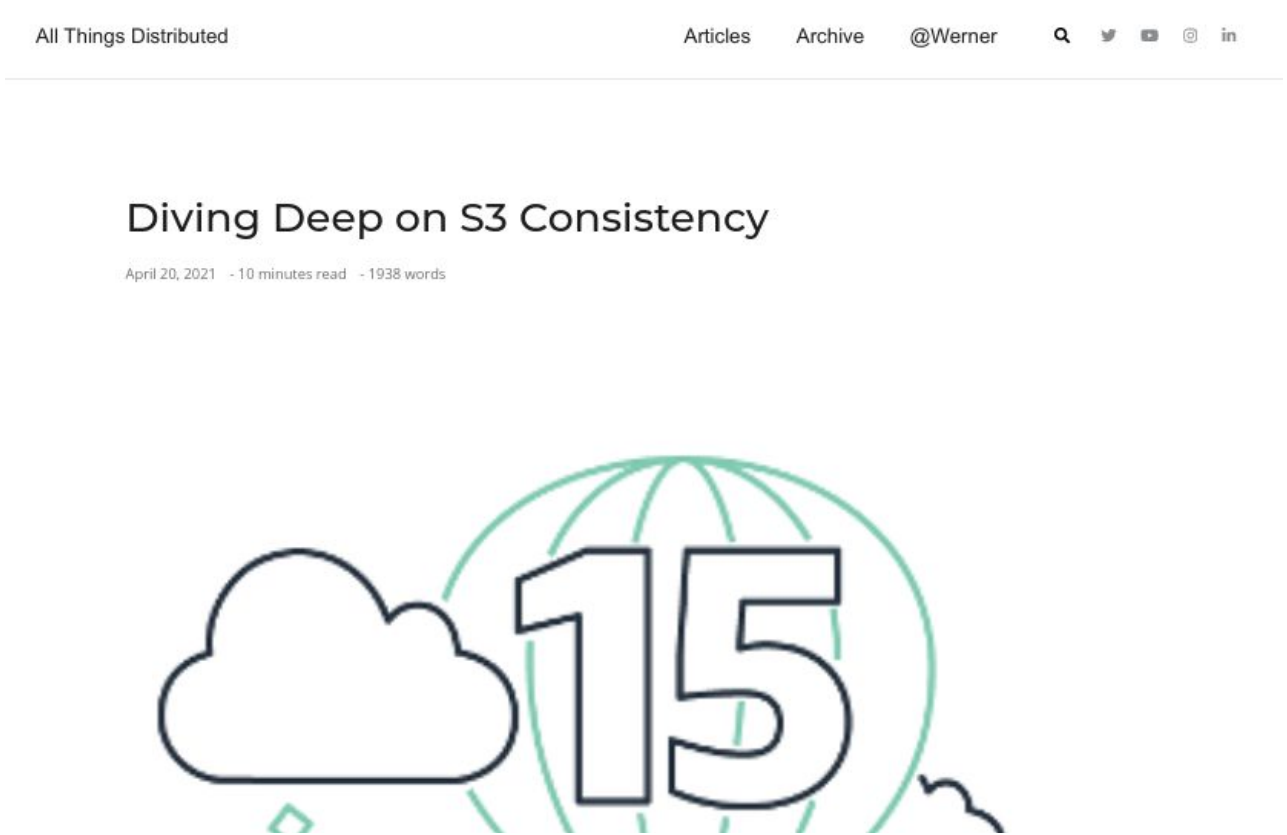
- Amazon S3 does not support object locking for concurrent writers. If two PUT requests are simultaneously made to the same key, the request with the latest timestamp wins. If this is an issue, you will need to build an object-locking mechanism into your application

https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html#ConsistencyModel

Bucket configurations have an eventual consistency model. Specifically:

- If you delete a bucket and immediately list all buckets, the deleted bucket might still appear in the list.
- If you enable versioning on a bucket for the first time, it might take a short amount of time for the change to be fully propagated. We recommend that you wait for 15 minutes after enabling versioning before issuing write operations (PUT or DELETE) on objects in the bucket.

https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html#ConsistencyModel



https://www.allthingsdistributed.com/2021/04/s3-strong-consistency.html

It’s still unclear...

...but feels quite strong

Magician usually does not reveal its secrets...

Google Spanner

Spanner, TrueTime & The CAP Theorem

Eric Brewer
VP, Infrastructure, Google

February 14, 2017

Spanner is Google's highly available global SQL database [CDE+12]. It manages replicated data at great scale, both in terms of size of data and volume of transactions. It assigns globally consistent real-time timestamps to every datum written to it, and clients can do globally consistent reads across the entire database without locking.

https://research.google/pubs/pub45855/

“Technically a CP”

“Effectively CA”

What does that even mean?

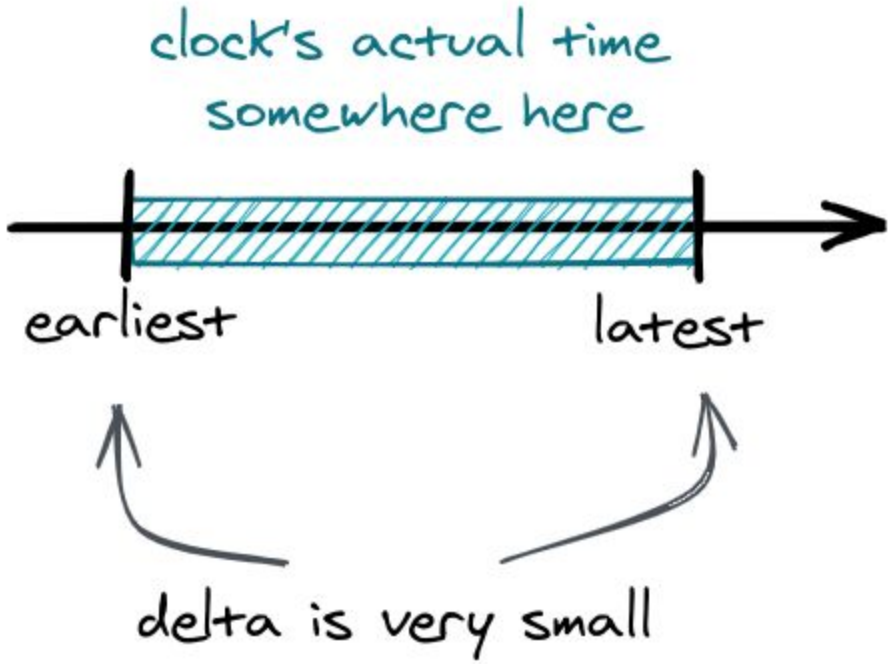
If you're rich, the rules apply a little less to you...

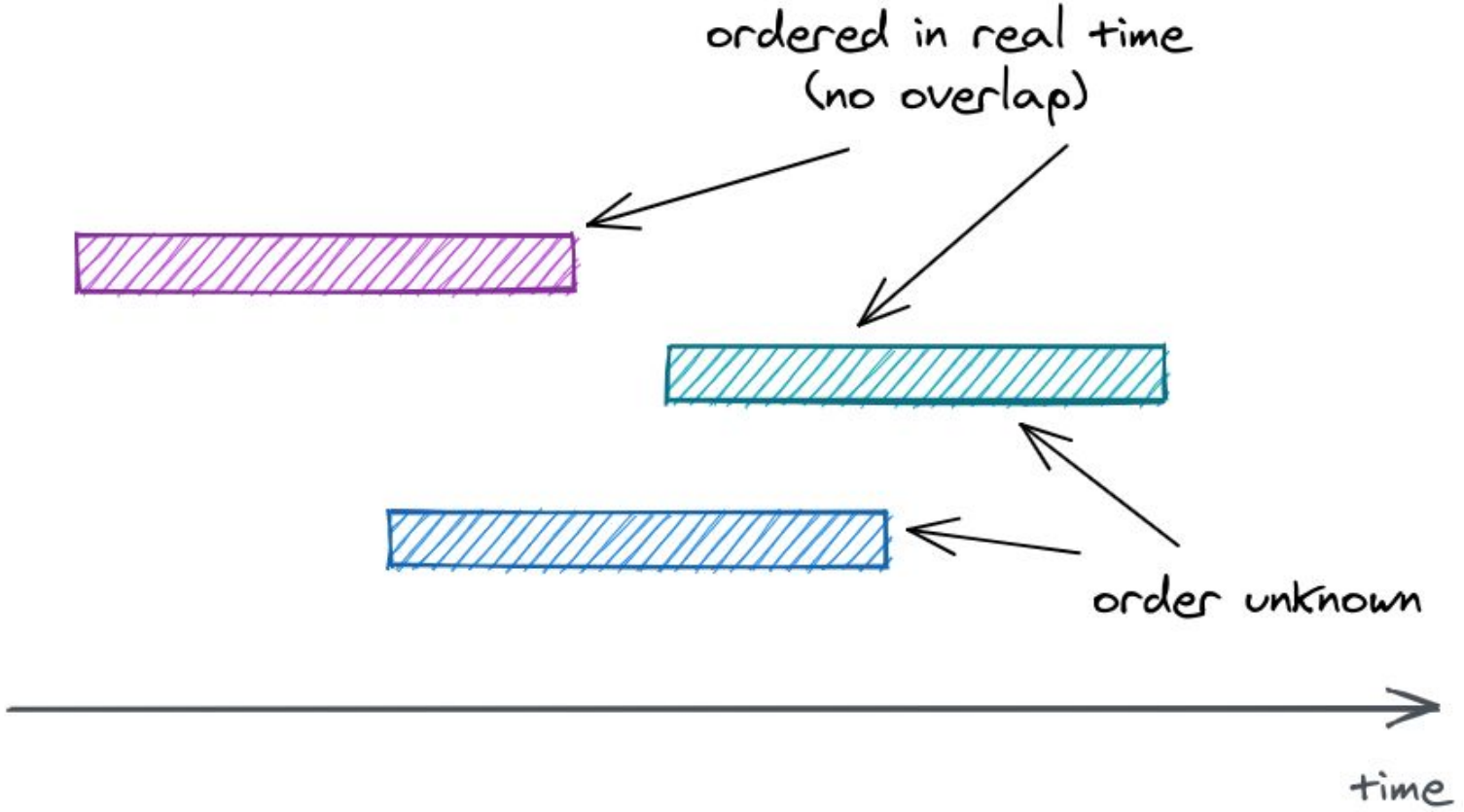
…but you have to be clever too!

2PC + 2PL + Paxos

TrueTime

t = [earliest, latest]




ordered in real time
(no overlap)
order unknown
time

Linearizability from TrueTime

“Effectively CA”

C always guaranteed

Availability: 99.999%
(5 min 15 sec / year)

Differential availability

It’s not a crime if you don’t get caught!

Spanner squeezes a lot out of CAP

AWS S3 SLA

Monthly Uptime Percentage	Service Credit Percentage
Less than 99.9% but greater than or equal to 99.0%	10%
Less than 99.0% but greater than or equal to 95.0%	25%
Less than 95.0%	100%

https://aws.amazon.com/s3/sla/

No discount:
1m 26s per day!

Replication Time Control (RTC)

Monthly 15-minute Replication Percentage	Service Credit Percentage
Less than 99.9% but greater than or equal to 98.0%	10%
Less than 98.0% but greater than or equal to 95.0%	25%
Less than 95.0%	100%

"Monthly 15-minute Replication Percentage" is calculated by subtracting from 100% the percentage of the objects replicated by the RTC Feature that did not successfully complete replication within 15 minutes in each region pair per account during the monthly billing cycle in which the replication was initiated.

https://aws.amazon.com/s3/sla-rtc/

SLA is the new CAP

CAP hidden under the availability guarantees

Availability issues easy to recover from

Unpredictable consistency is the worst nightmare

Consistency must be predictable!

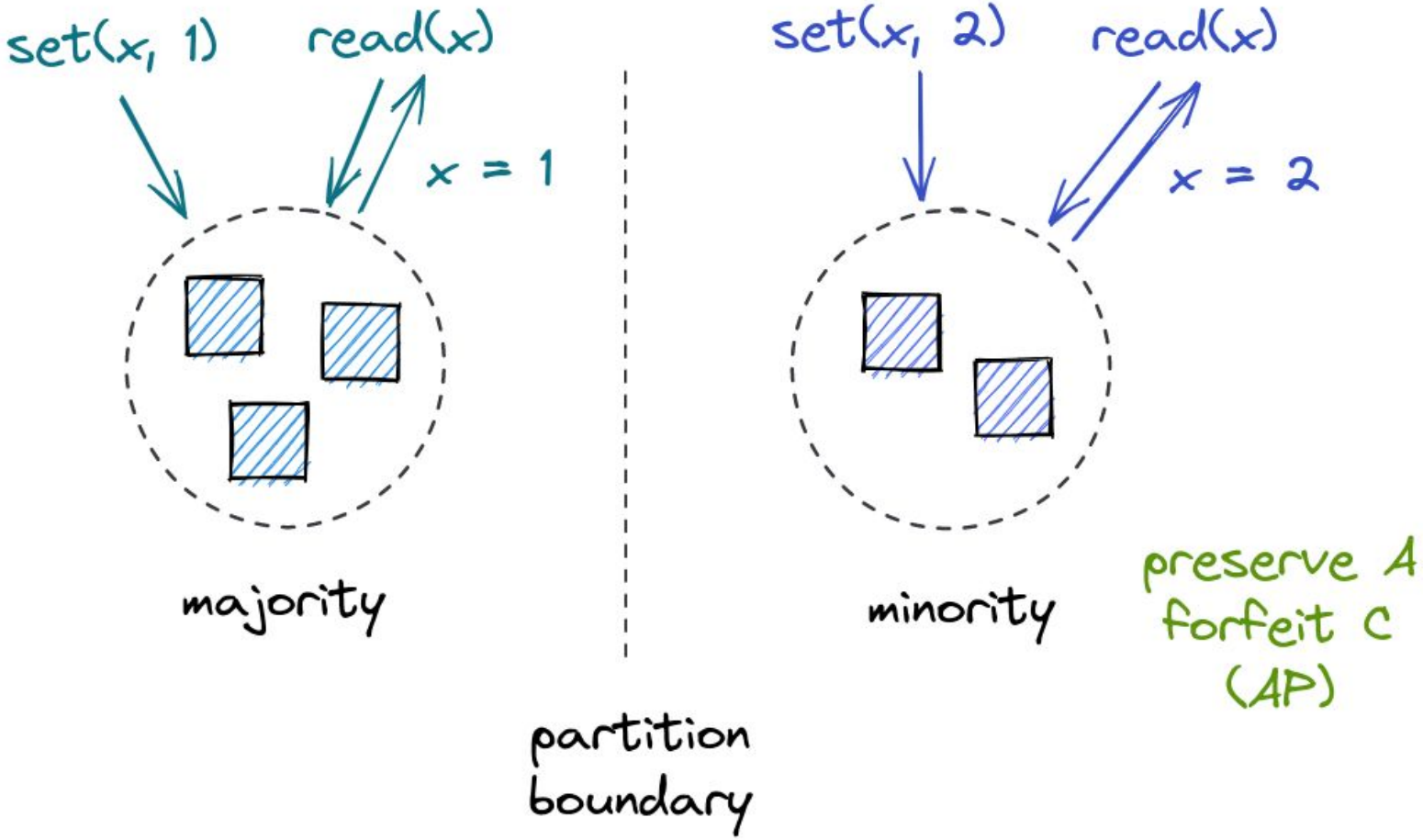
Availability: a rug to sweep under

Is CAP only for DB/service providers?

Cassandra

“Partitioned row store”

Designed to be always writable...


set(x, 1)
read(x)
x = 1
majority
set(x, 2)
read(x)
x = 2
minority
preserve A
forfeit C
(AP)
partition
boundary

...and writing things really fast!

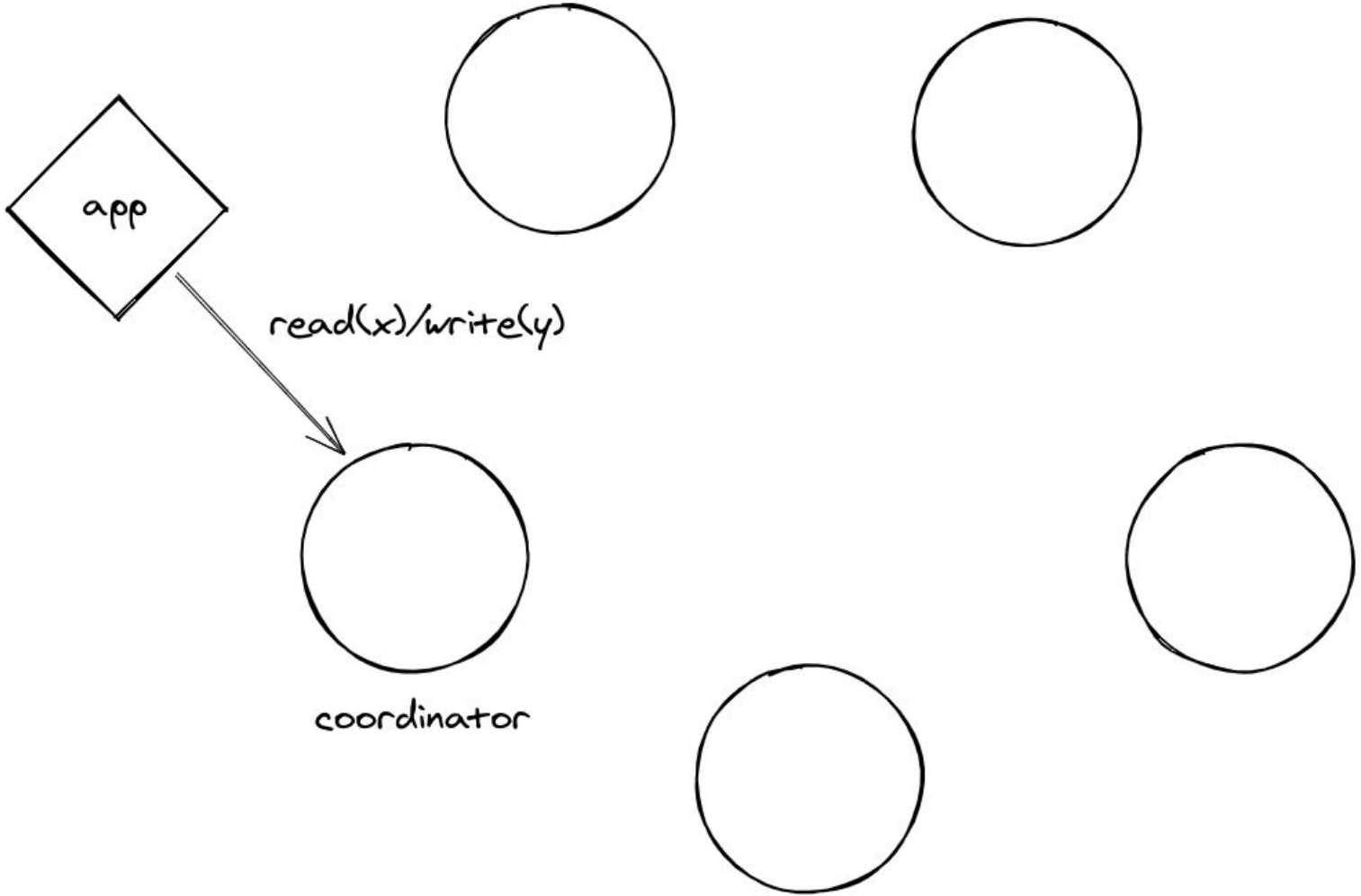
Nodes democracy

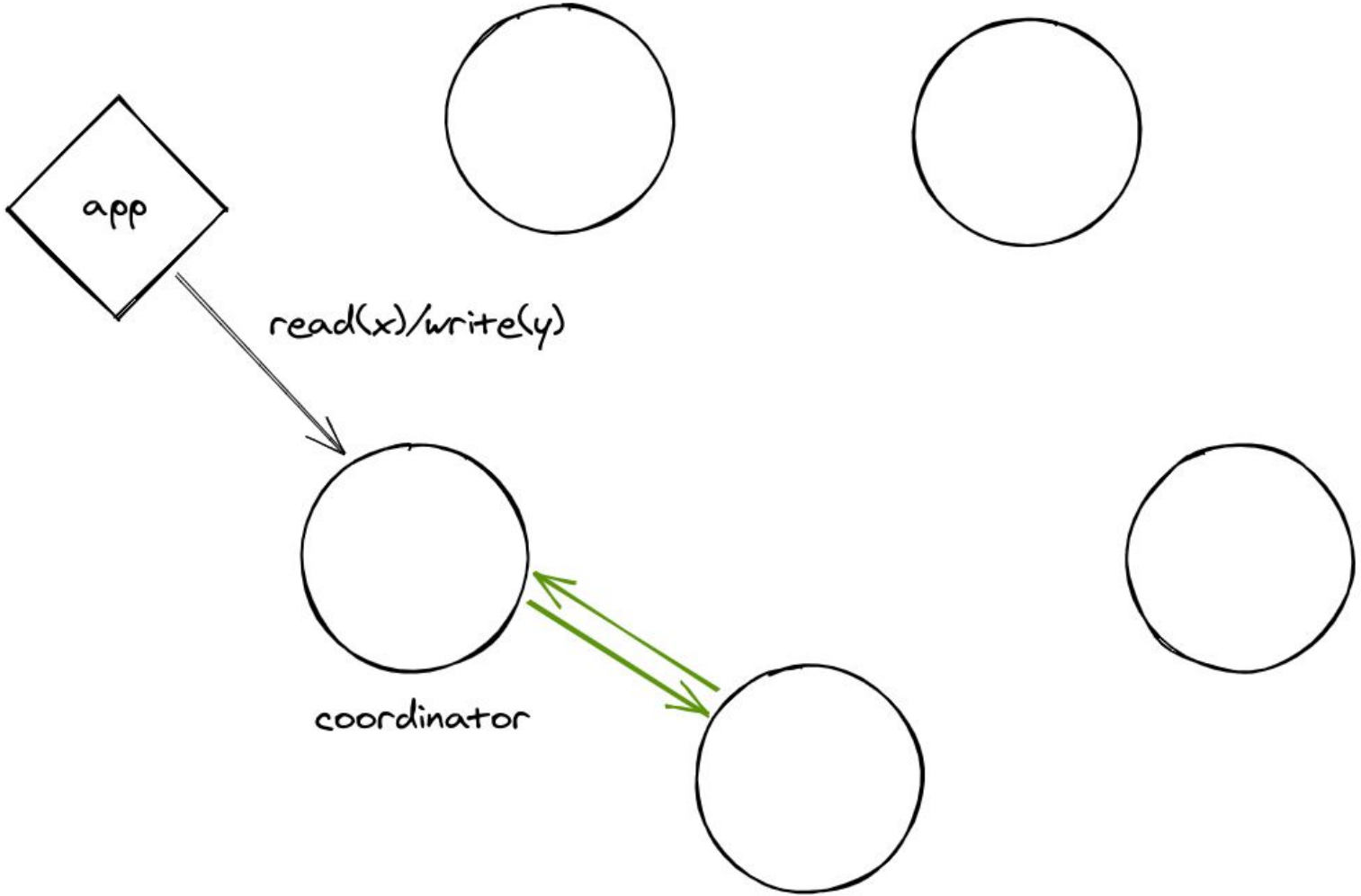
Gossiping

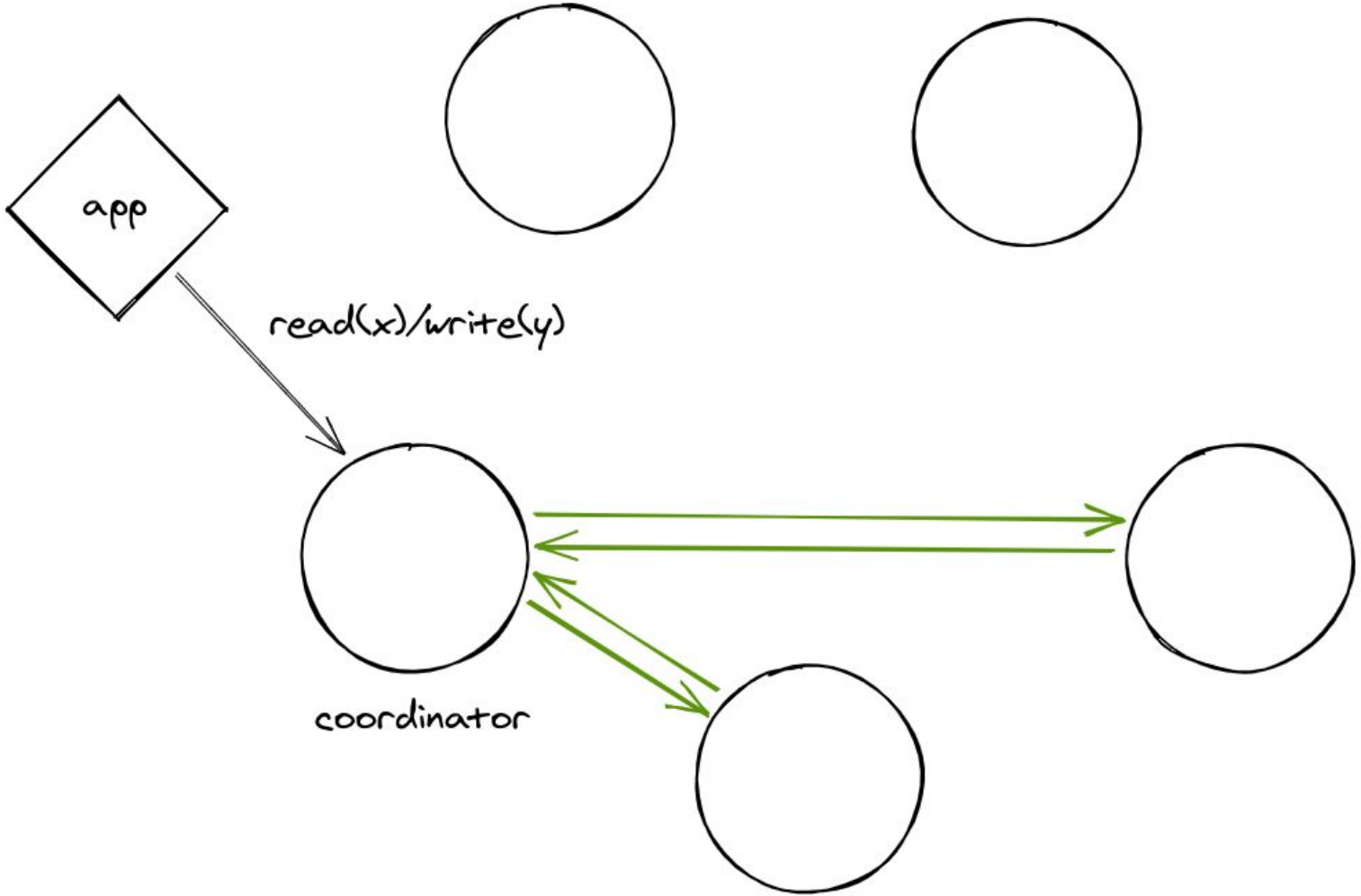
Defer conflict resolution to the reads

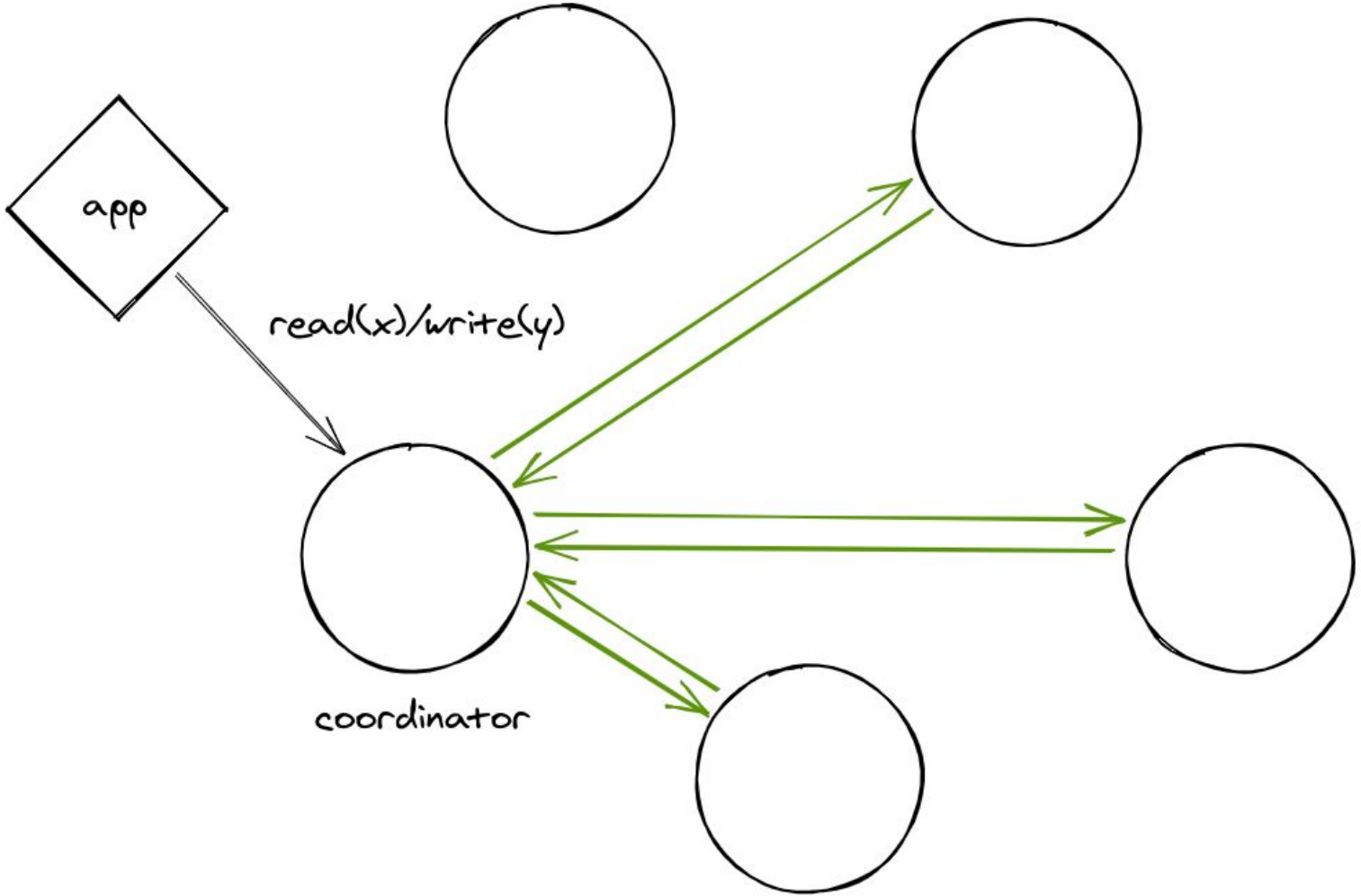
Last write wins

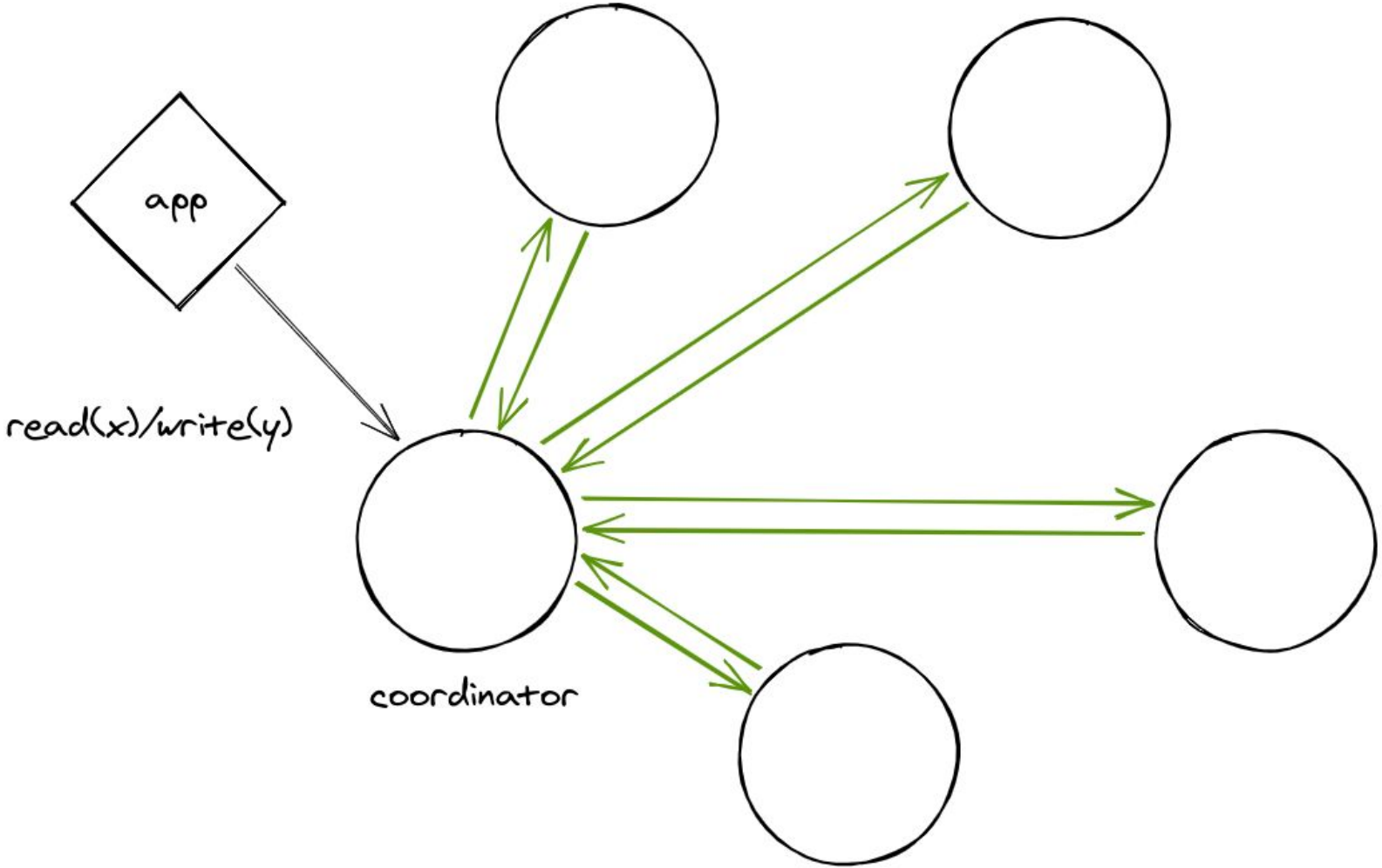
“Tunable consistency”

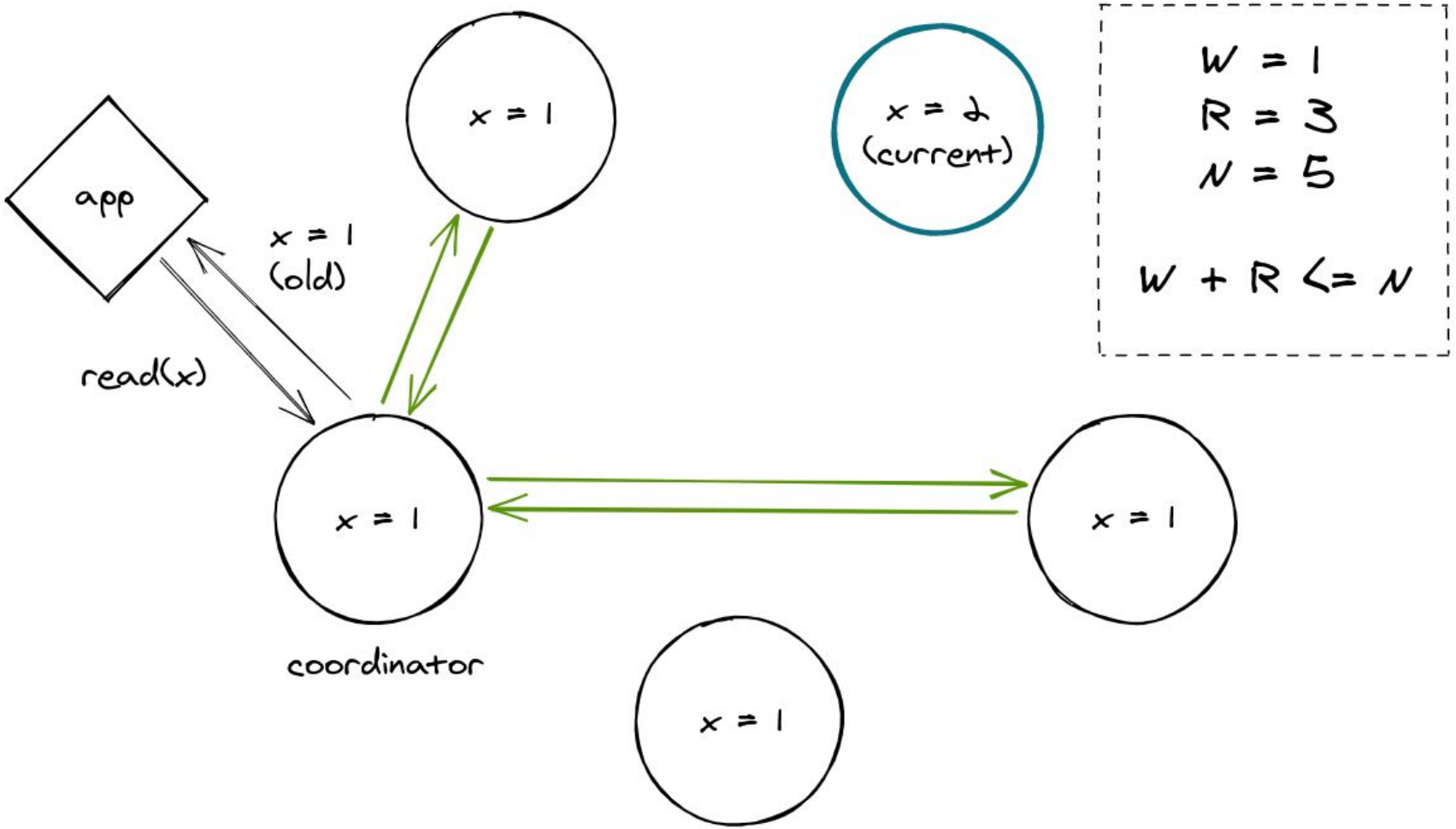

app
read(x)/write(y)
coordinator

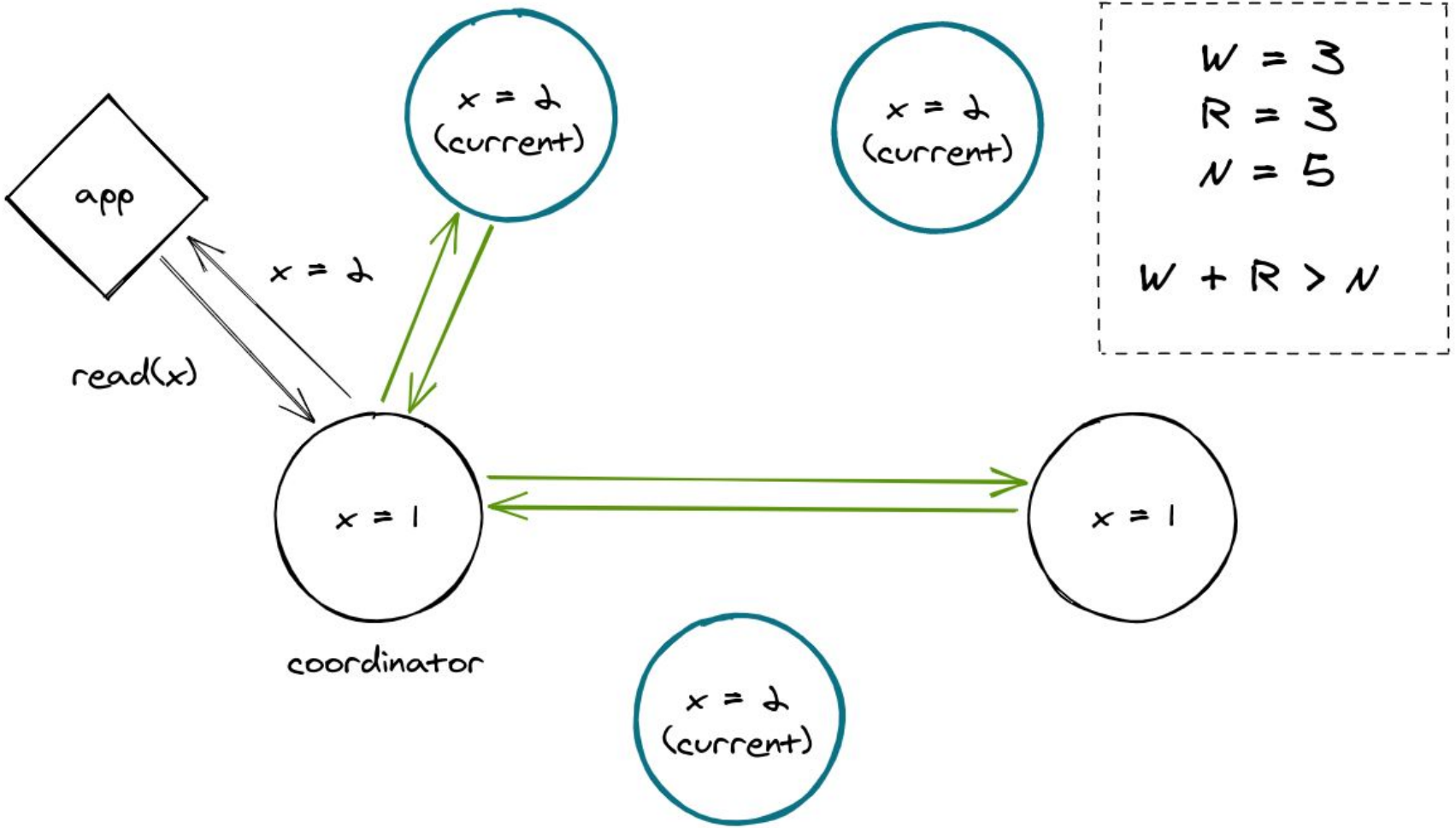

app
read(x)/write(y)
coordinator


app
read(x)/write(y)
coordinator


app
read(x)/write(y)
coordinator


app
read(x)/write(y)
coordinator


app
x = 1
x = 1
(old)
read(x)
x = 1
(current)
W = 1
R = 3
N = 5
W + R <= N
x = 1
x = 1
coordinator
x = 1


app
read(x)
x = 2
x = 2
(current)
x = 2
(current)
x = 1
coordinator
x = 1
x = 2
(current)
W = 3
R = 3
N = 5
W + R > N

More like a CP...

We can make different A vs C choices per operation!

CAP not only on the DB side

Lightweight transactions (Paxos)

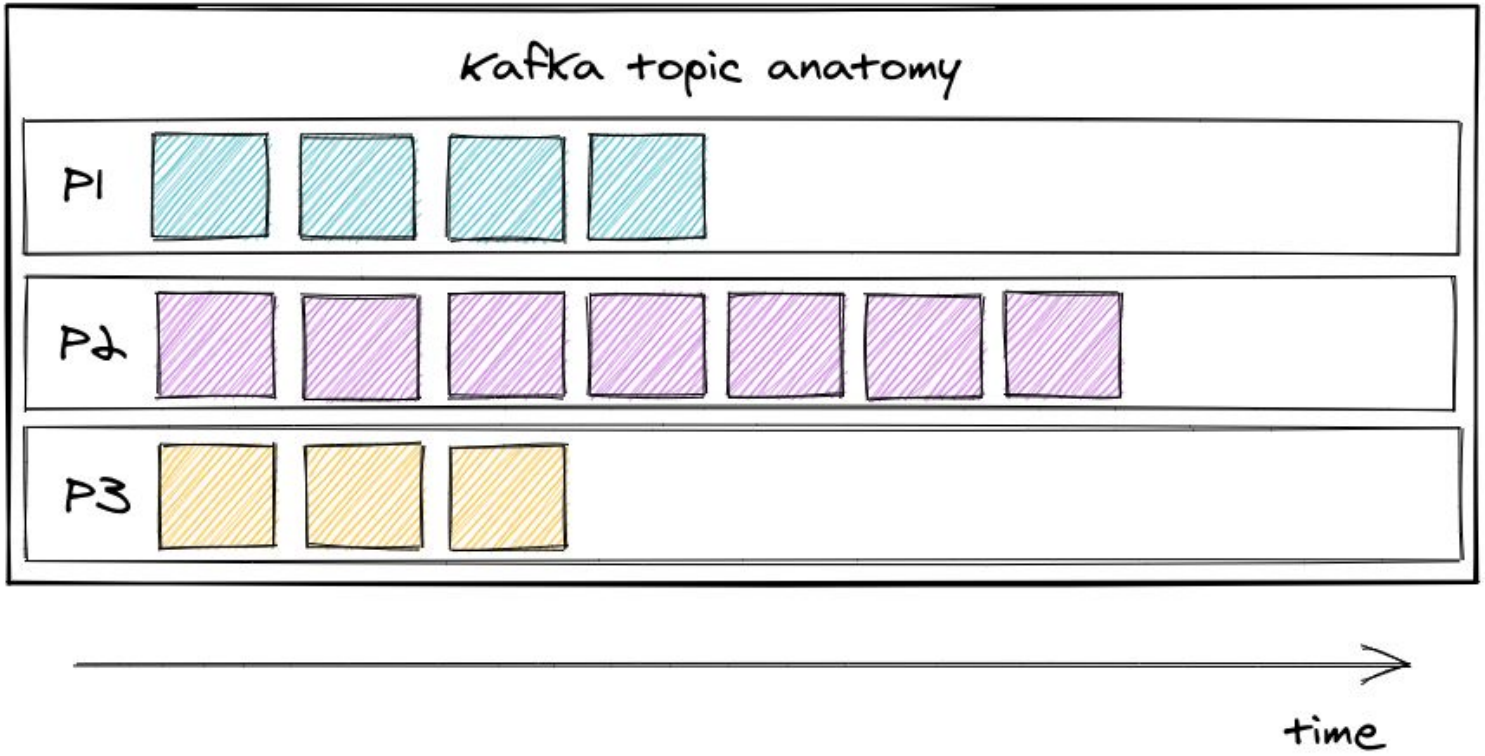
Known for (not only) performance issues...

Flexibility does not come for free

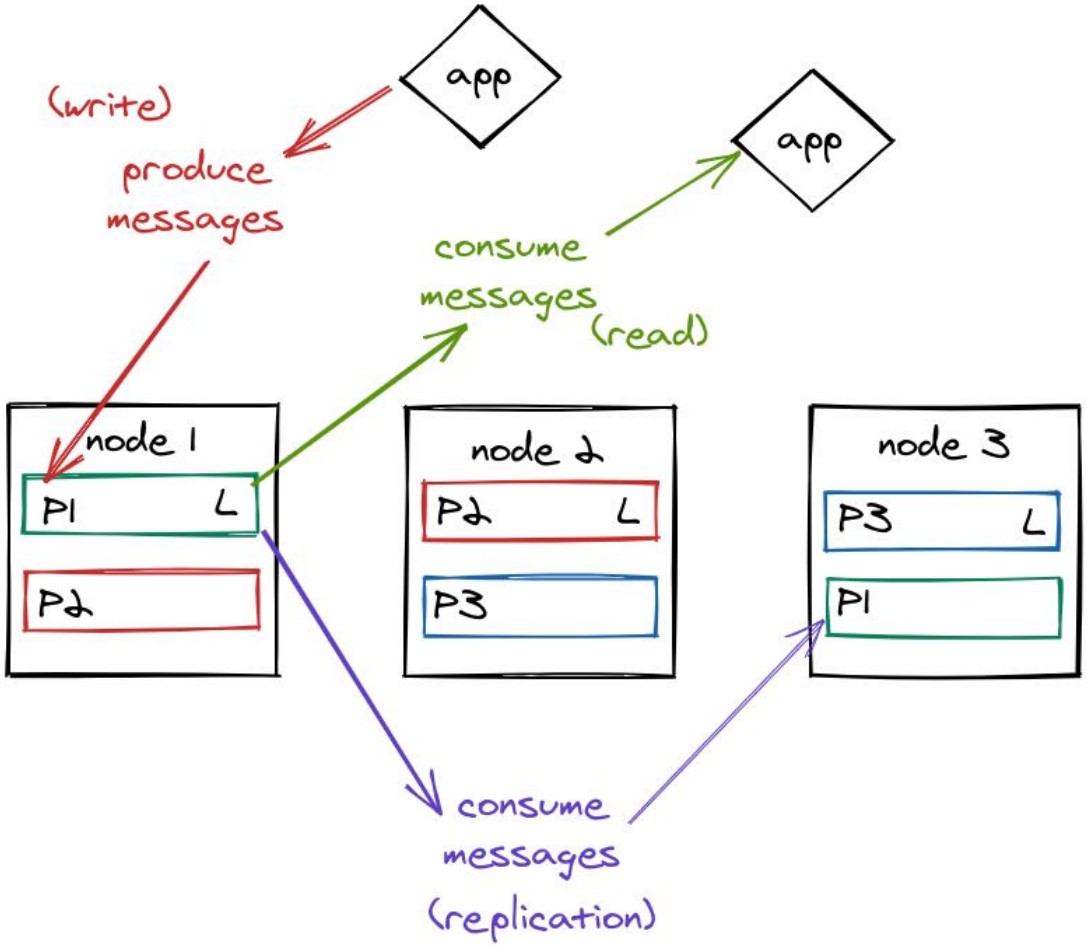
Kafka

Distributed append-only data log (and beyond!)

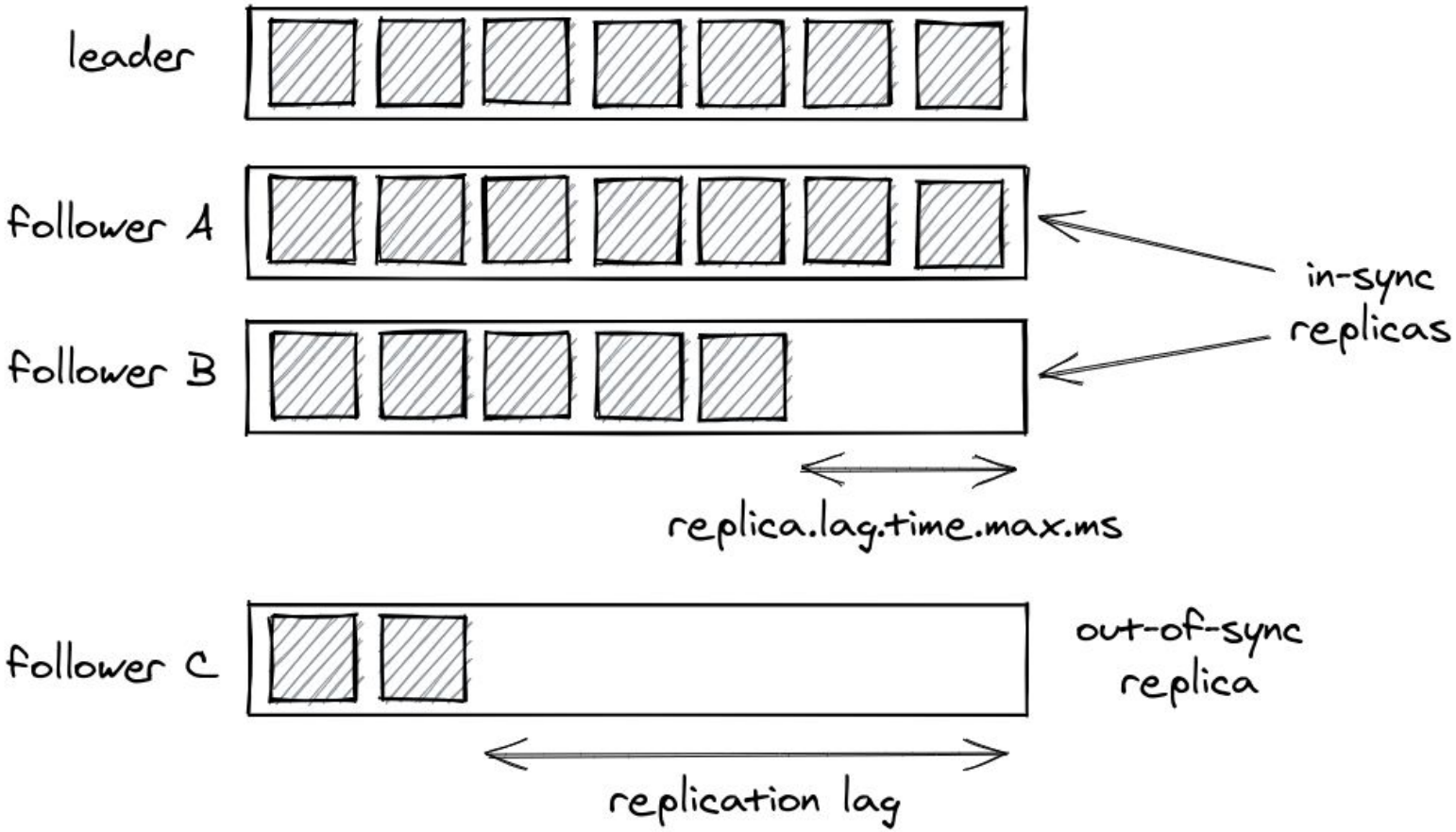
Kafka was build for scalability


kafka topic anatomy
P1
P2
P3
time

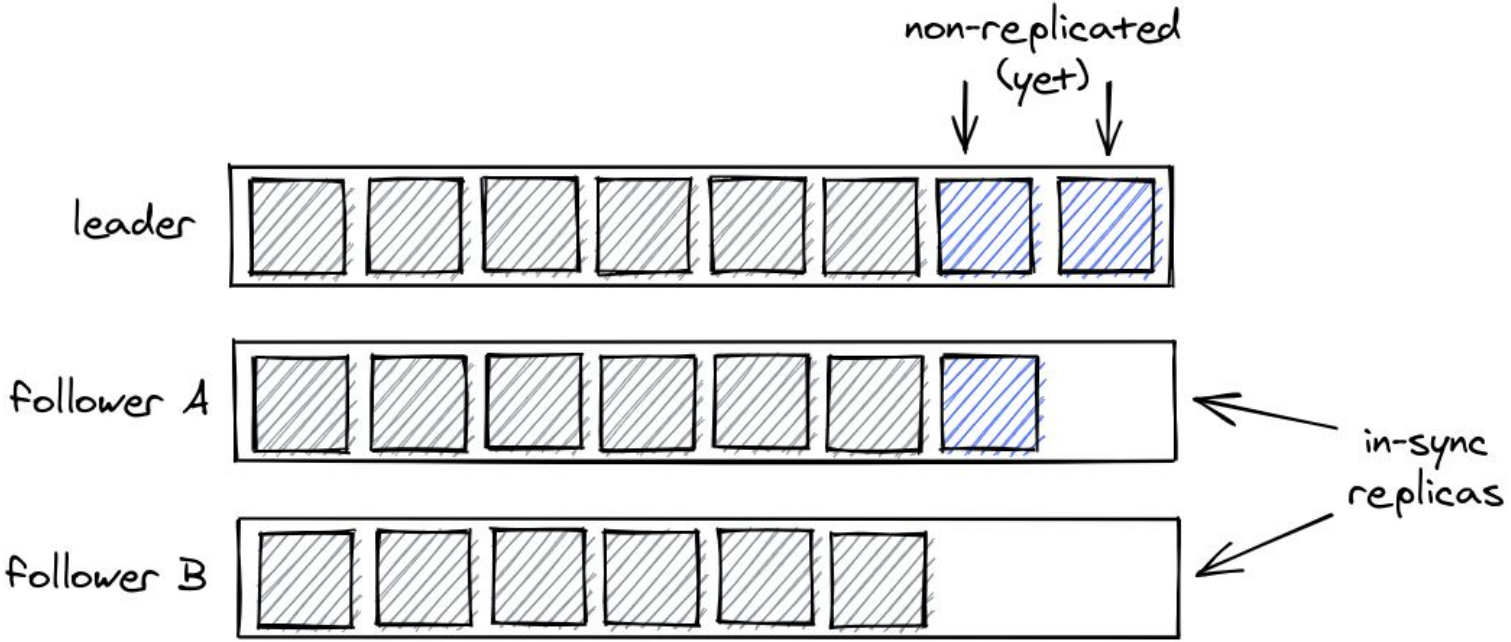
Consistency recipe: leader & follower(s)

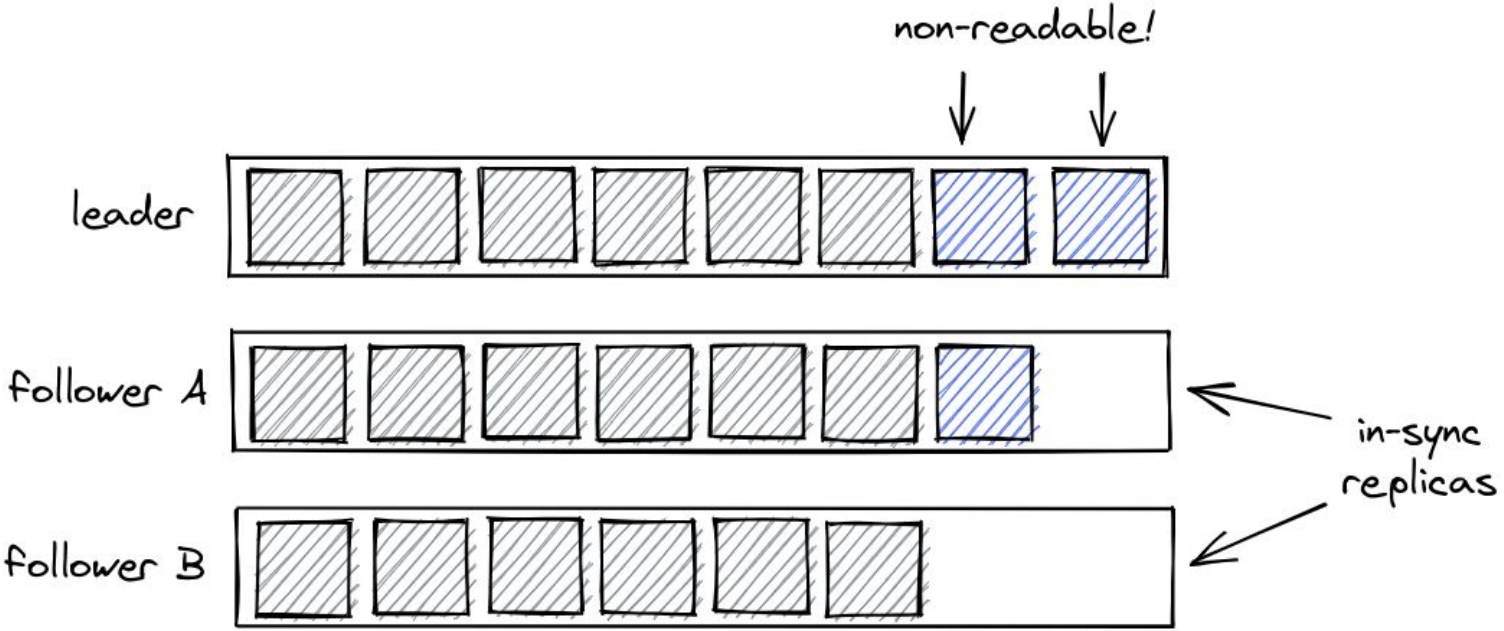

app
(write)
produce
messages
app
consume
messages
(read)
node 1
P1
L
P2
node 2
P2
L
P3
node 3
P3
L
P1
consume
messages
(replication)

Followers may lag


leader
follower A
follower B
in-sync
replicas
replica.lag.time.max.ms
follower C
out-of-sync
replica
replication lag

ISR may become a new leader

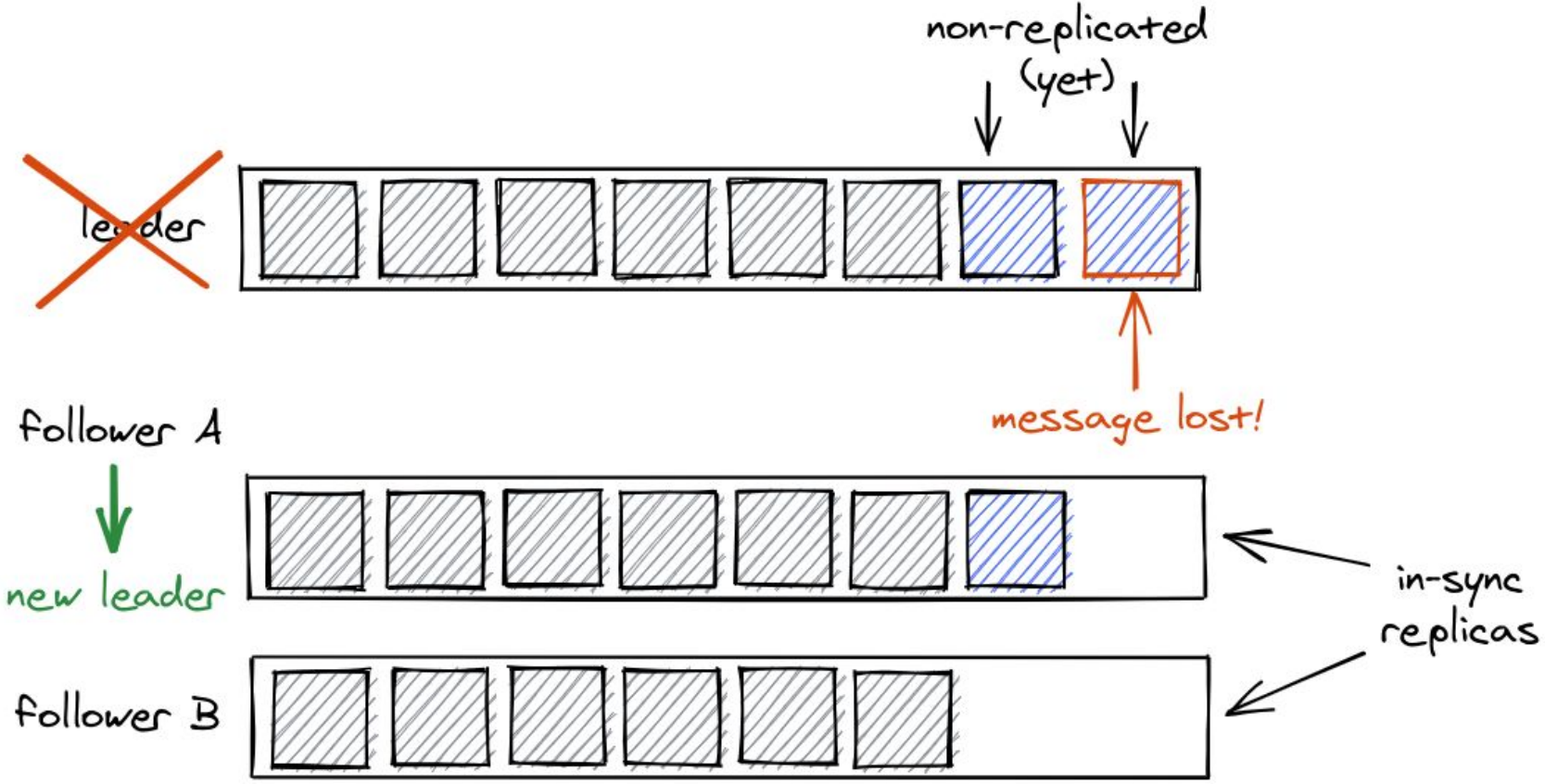

non-replicated
(yet)
leader
follower A
follower B
in-sync
replicas


non-readable!
leader
follower A
follower B
in-sync
replicas

No “read your writes”?


producer's
wait time
acks=0 : "fire & forget"
acks=1 : wait for leader
acks=all: wait for all ISRs
min.insync.replicas
durability
read own writes

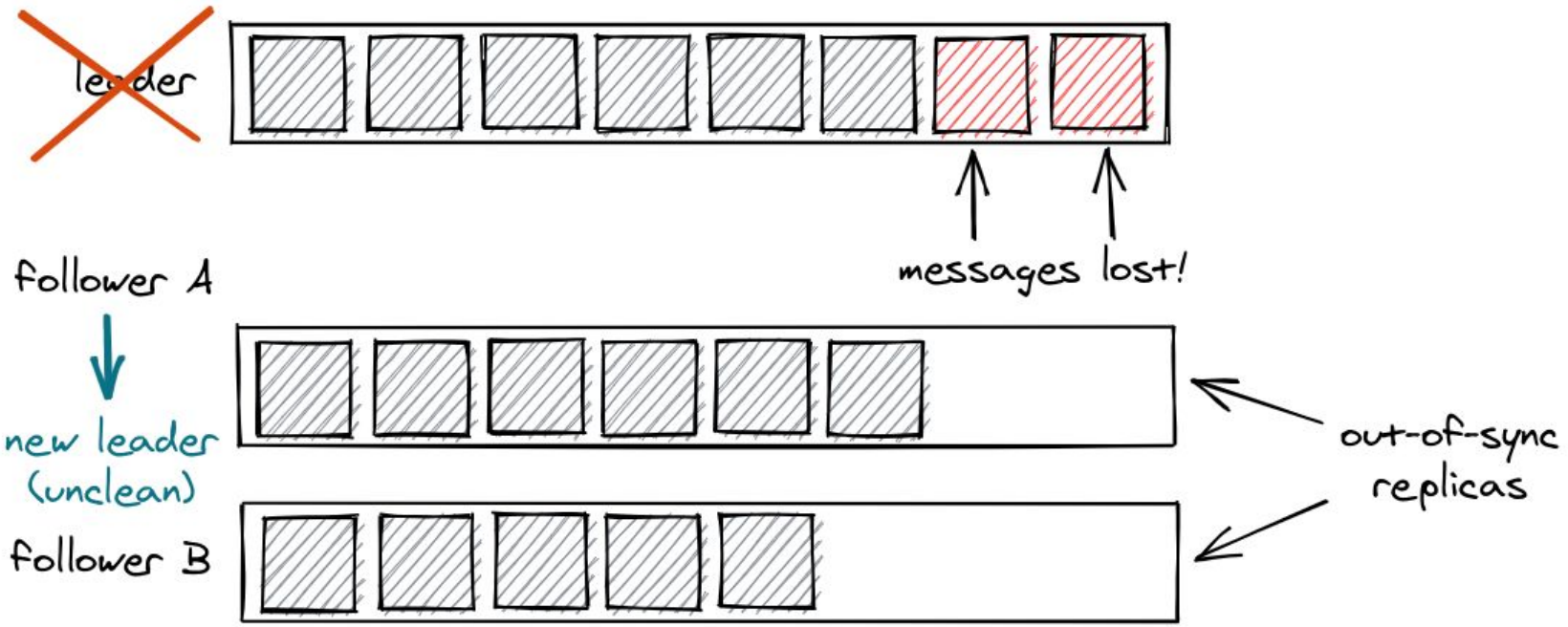
Leader unavailable?


non-replicated
(yet)
leader
message lost!
follower A
new leader
in-sync
replicas
follower B

Consistency does not guarantee durability!

Improving availability

Unclean leader election


leader
messages lost!
follower A
new leader
(unclean)
follower B
out-of-sync
replicas

KIP-392: Follower fetching

There is more!
(flushes, transactions)

Config & params define CAP trade-offs

Summary

CAP theorem is not going to retire

Consistency and availability are spectrums

CAP’s C/A defined precisely

C means linearizability

A is about being always available

High availability is not enough for A

Eventual consistency is a spectrum too

Partition handling strategy matters

SaaS: SLA beats CAP

Read the fine print carefully!

Availability trade-offs are the easiest

Consistency must be predictable!

Not all operations have to choose the same (C vs A)

CAP not only for DB designers

CAP missing parts: durability, latency, ...

Tools can be more flexible than we think

The devil is in the (impl/conf) details

Know your tools, read the docs!

References

- Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services, S. Gilbert, N. Lynch, 2002
- CAP Twelve Years Later: How the “Rules” Have Changed, E. Brewer, 2012
- A Critique of the CAP Theorem, M. Kleppmann, 2015
- Spanner, TrueTime and the CAP Theorem, E. Brewer, 2017
- Designing Data-Intensive Applications, M. Kleppmann, 2017
- Cassandra: The Definitive Guide, J. Carpenter, E. Hewitt, 2020
- Diving Deep on S3 Consistency, W. Vogels, 2021

mikemybytes.com

Twitter: @mikemybytes

Thank you!