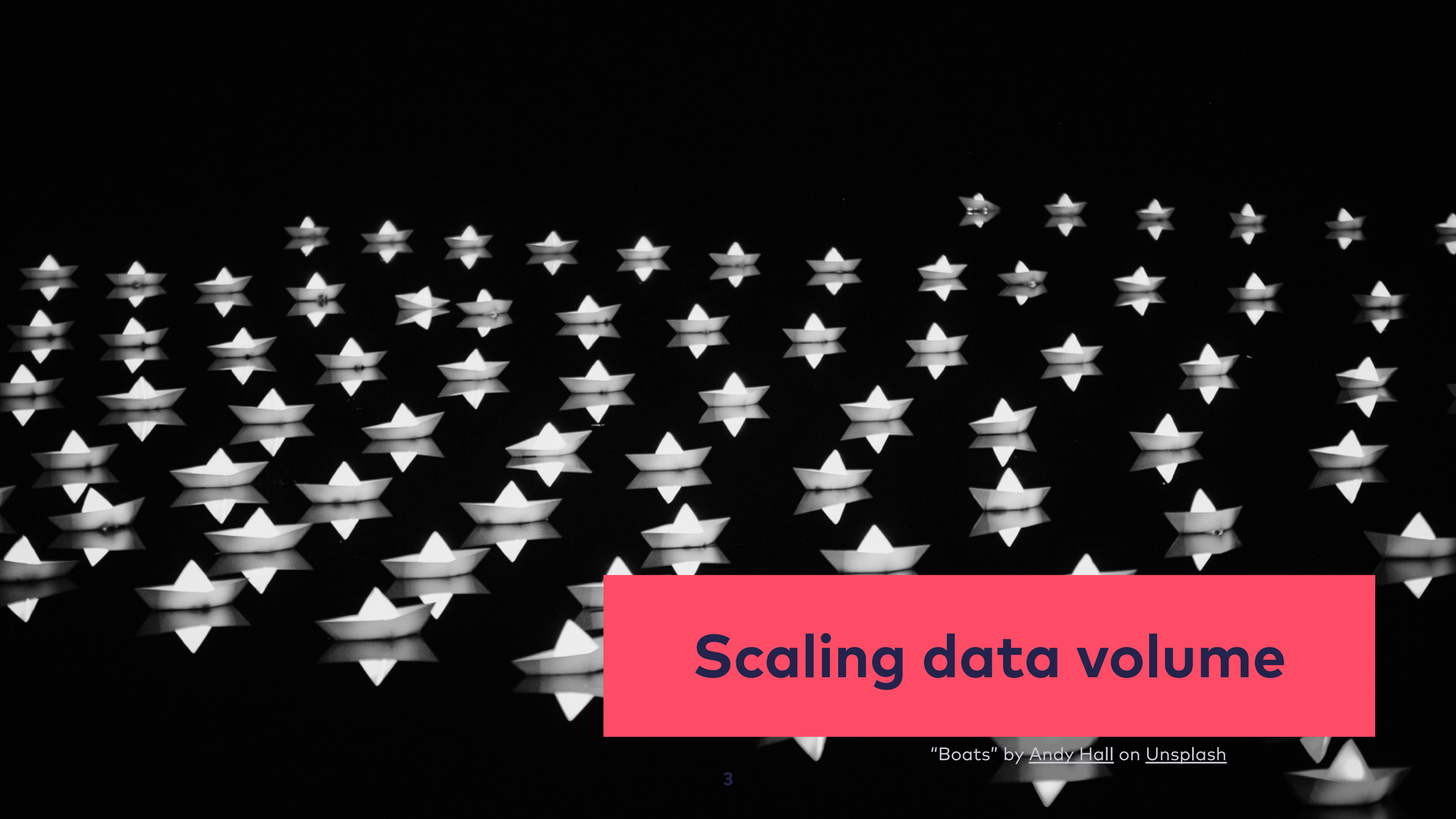
INNOQ Technology Lunch
2021/09/08

Scaling Data

INNOQ

LUCAS DOHMEN
@moonbeamlabs

What is your goal?



Scaling data volume

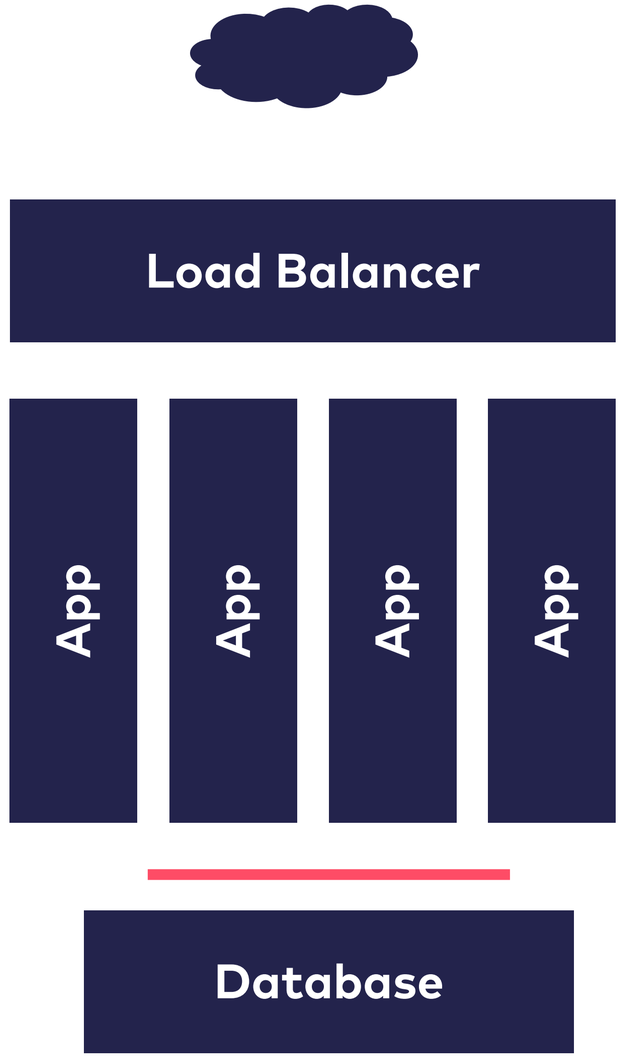
"Boats" by Andy Hall on Unsplash

How do we scale web applications?

Share nothing between application servers

Put behind a load balancer

Add servers



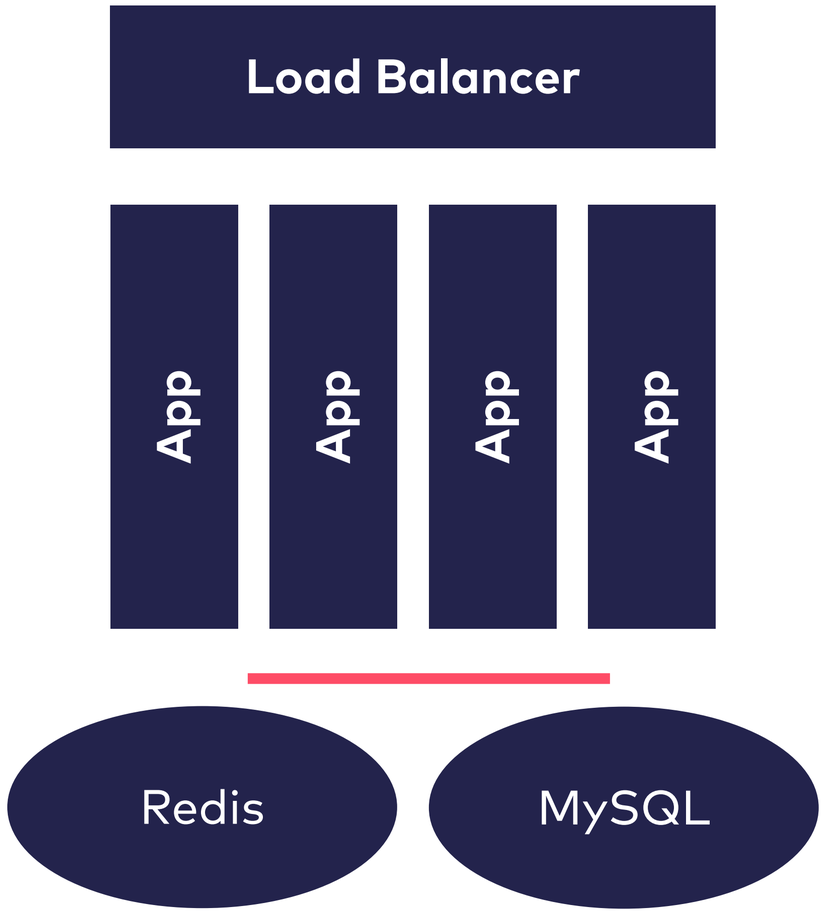
Share Nothing for Databases?

Possible & underused

Separate databases for separate data

- for different tenants
- for different countries
- for different use cases
- …

If we need to join data, we need to join in the app



But what if we can't do that manually?

Can the database do it for us?

Can it even scale automatically?

Maybe without us even noticing?

Sharding
=
Each node has only part of the data

Simple example: memcached

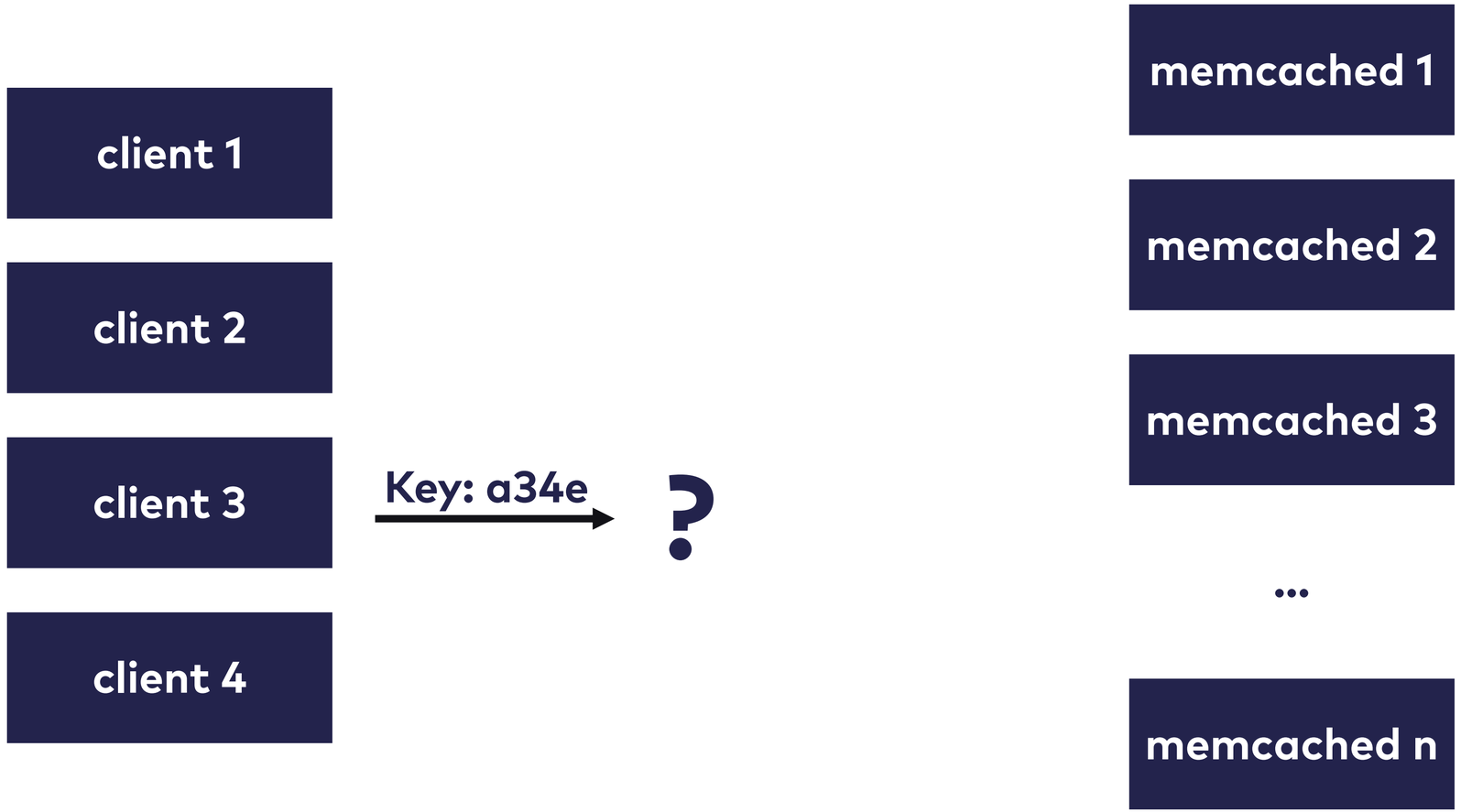
Cache (data may be lost)

Stores/retrieves a value for a key

More memcached nodes mean more memory is available

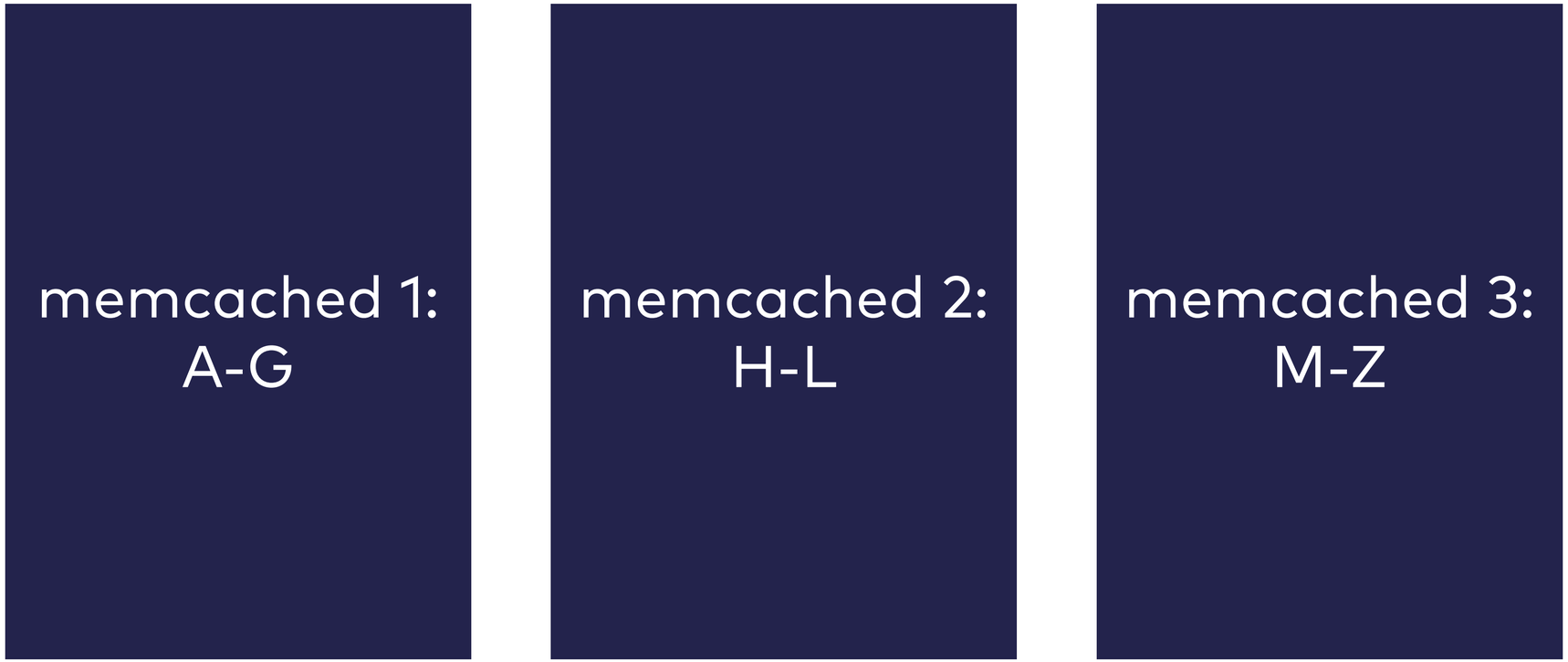
Sharding is done completely in the memchached-clients

memcached

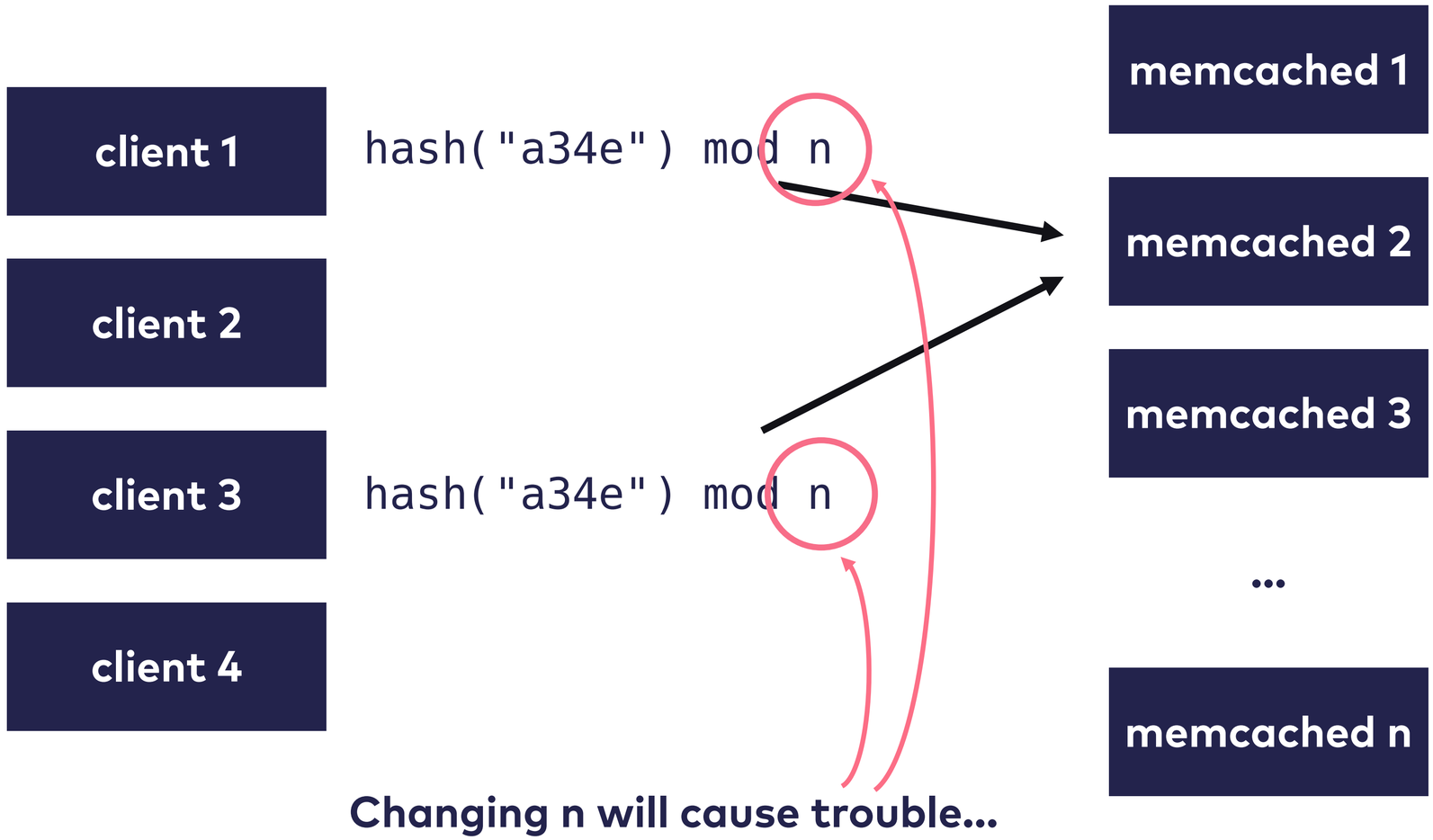

client 1
client 2
client 3
client 4
Key: a34e
?
memcached 1
memcached 2
memcached 3
...
memcached n

Sharding by Key

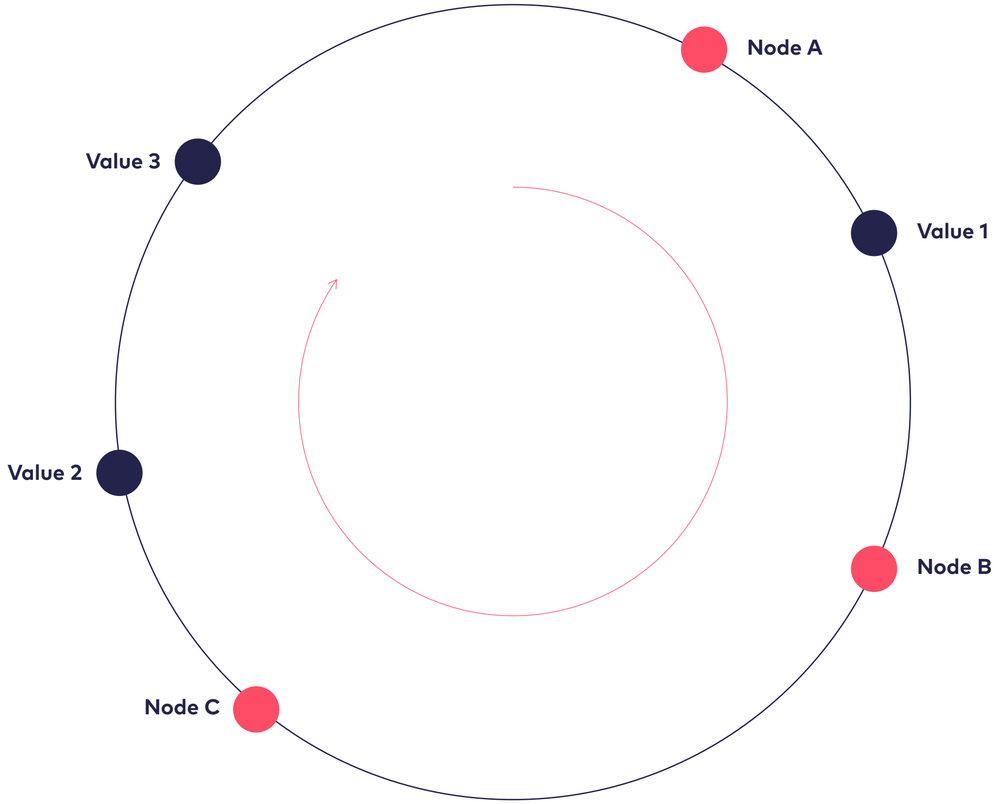
Hashed! ⇒ Equal distribution to all shards!



Equally distributed sharding

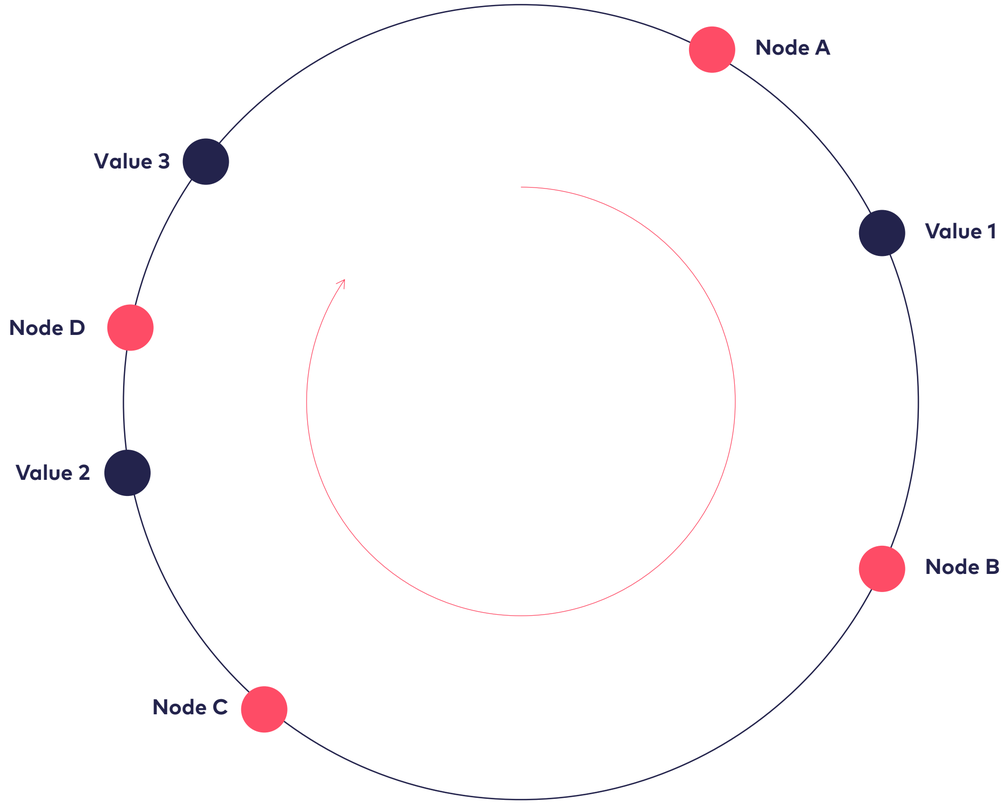


Consistent Hashing



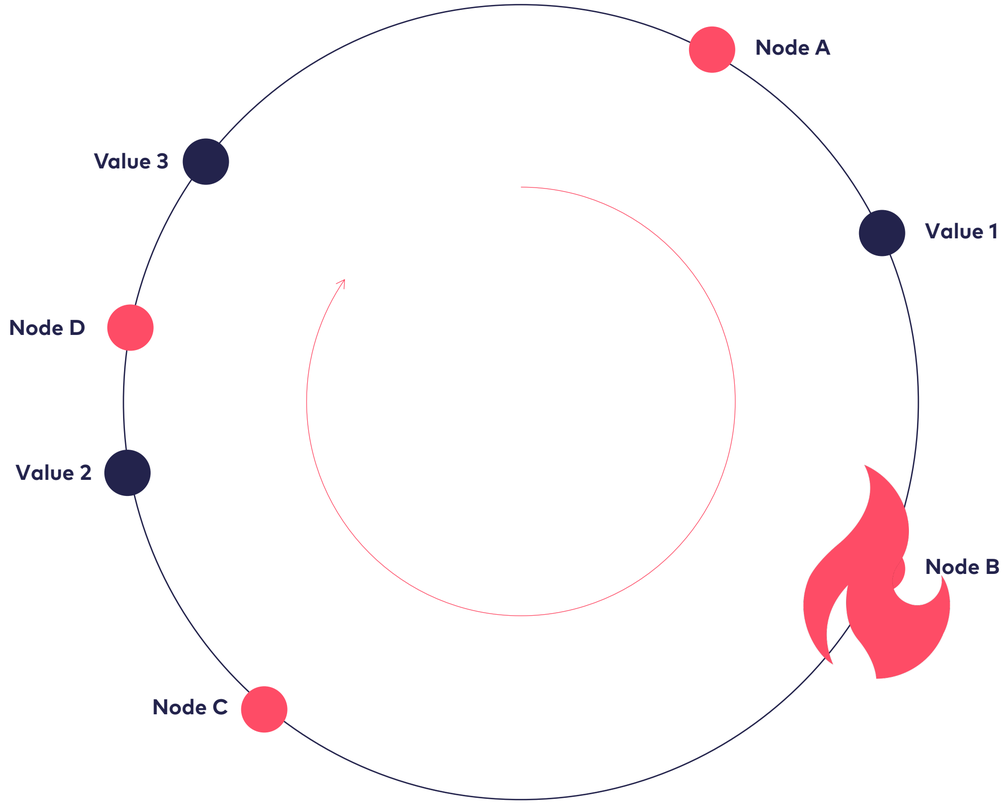
1 is stored in B
2 & 3 are stored in A

Consistent Hashing



1 is stored in B
3 is stored in A
2 is stored in D

Consistent Hashing



Only 1 is lost
And will be placed on C
next time

Sharding

Usable for scaling data volume

And a bit for reads and writes (since we have more machines)

But:

- what about really scaling reads and writes?
- what about geographical distribution?
- what about failure resistance?

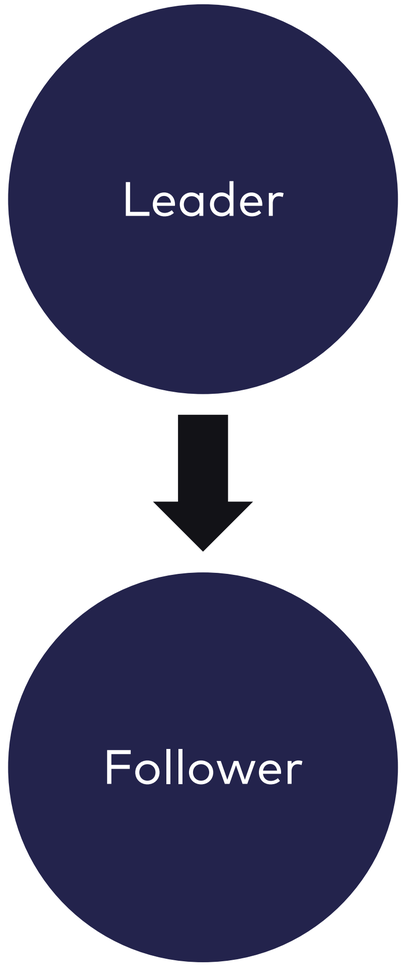
Replication
=
Same data on multiple nodes

Single Leader

Failover

Read scaling

No write scaling

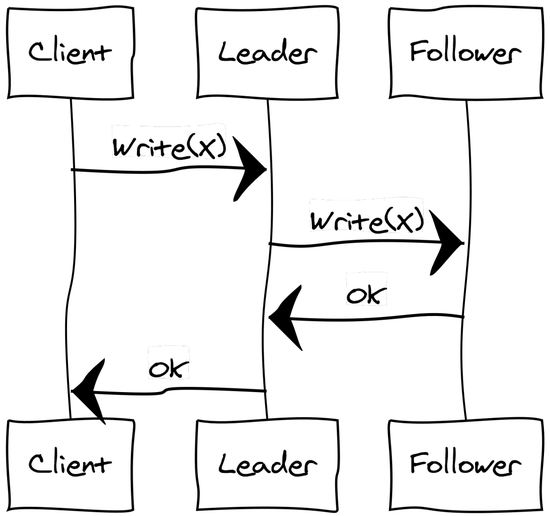


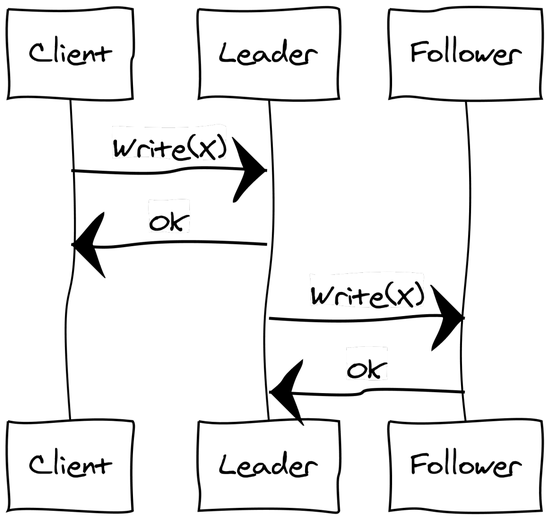
Sync or Async Replication?

Trade-off between consistency & speed

Sync: Every follower we add decreases performance

Async: If our leader dies and the replication is not done, we have lost acknowledged data. Also: Consistency is at risk.





Examples

Redis

MySQL/MariaDB

PostgreSQL

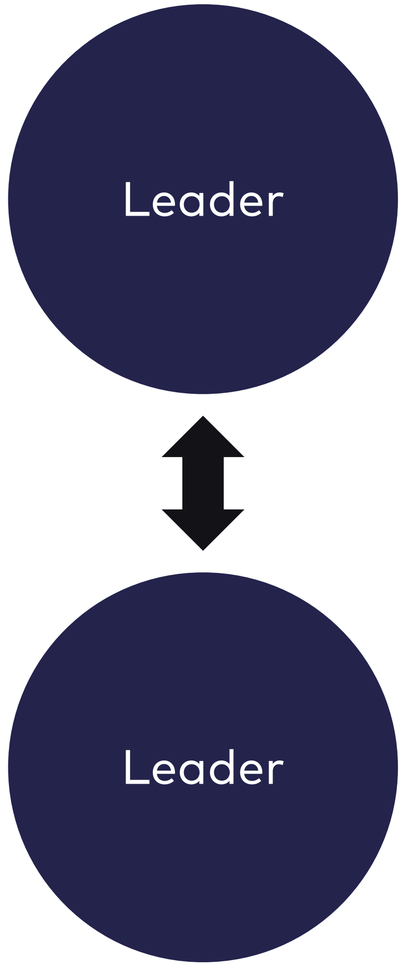
MongoDB

Multi Leader

Failover

Read & write scaling

Always async replication



Write Conflicts

Two leaders can accept a conflicting write

We usually resolve them when reading

Do we have all information to resolve the conflict at read time?

Examples

CouchDB

(git)

Paxos

Some database vendors call their database “multi leader”

when they in fact are closer to single leader

these algorithms are mostly based on Paxos (or Raft)

In these systems, the nodes do leader elections

And you always write to the leader

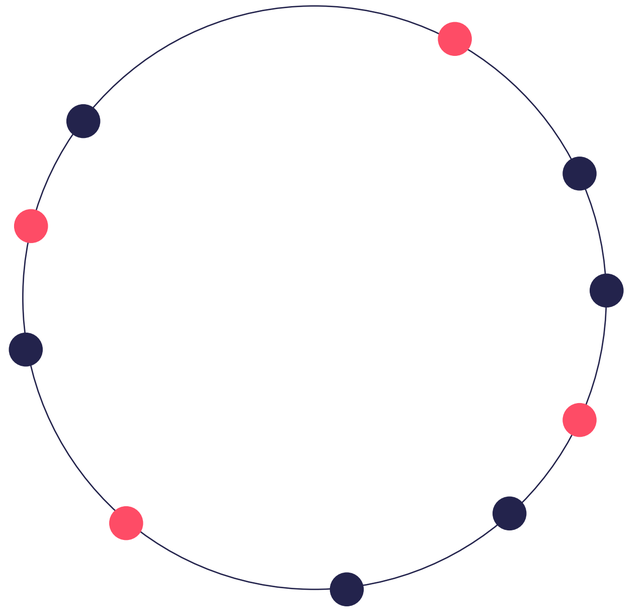
This leads to consistent systems

Explaining Paxos definitely goes beyond what we can cover here

Leaderless

Failover

Read & write scaling

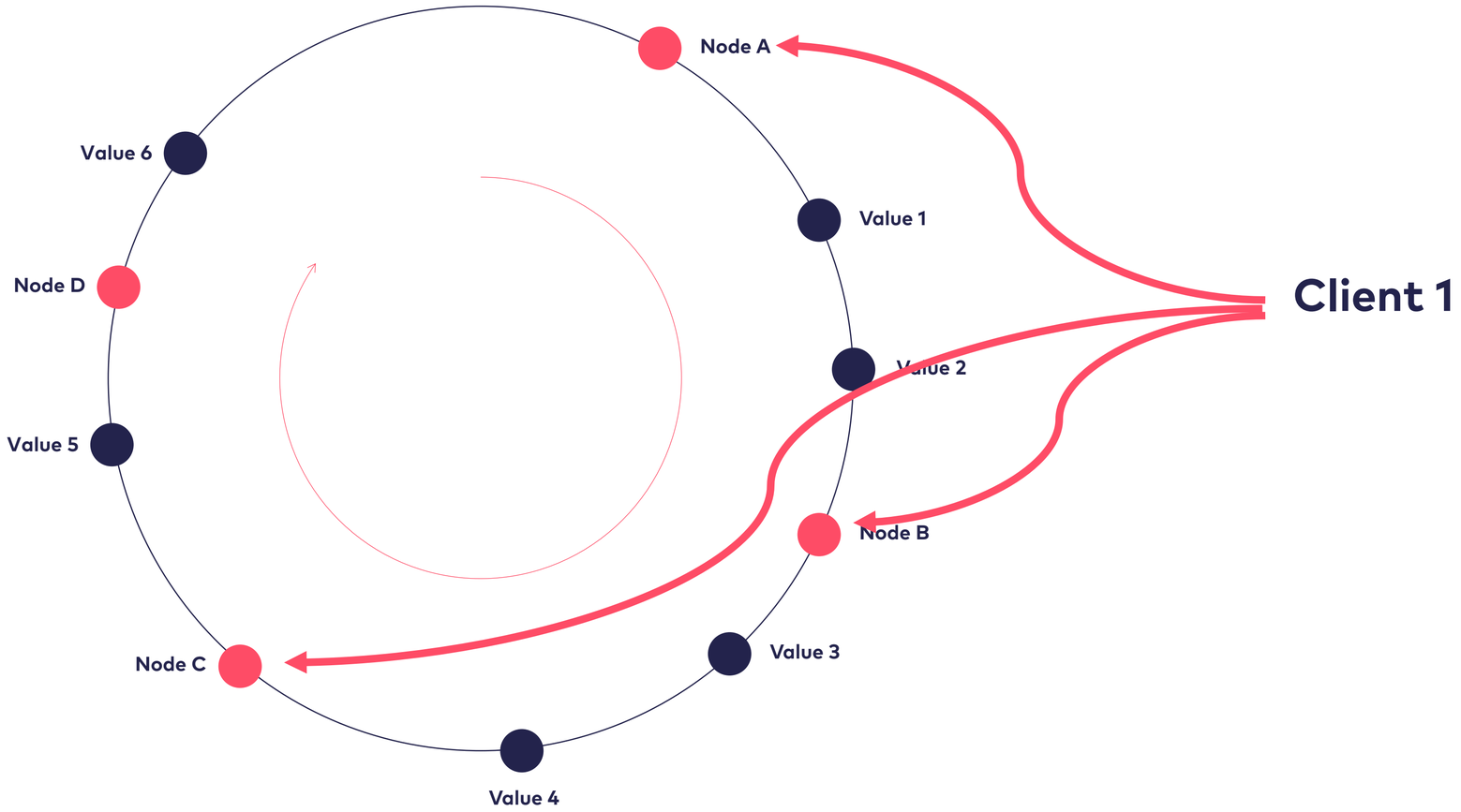


Examples

Riak

Cassandra

Back to our hash ring



Quorum

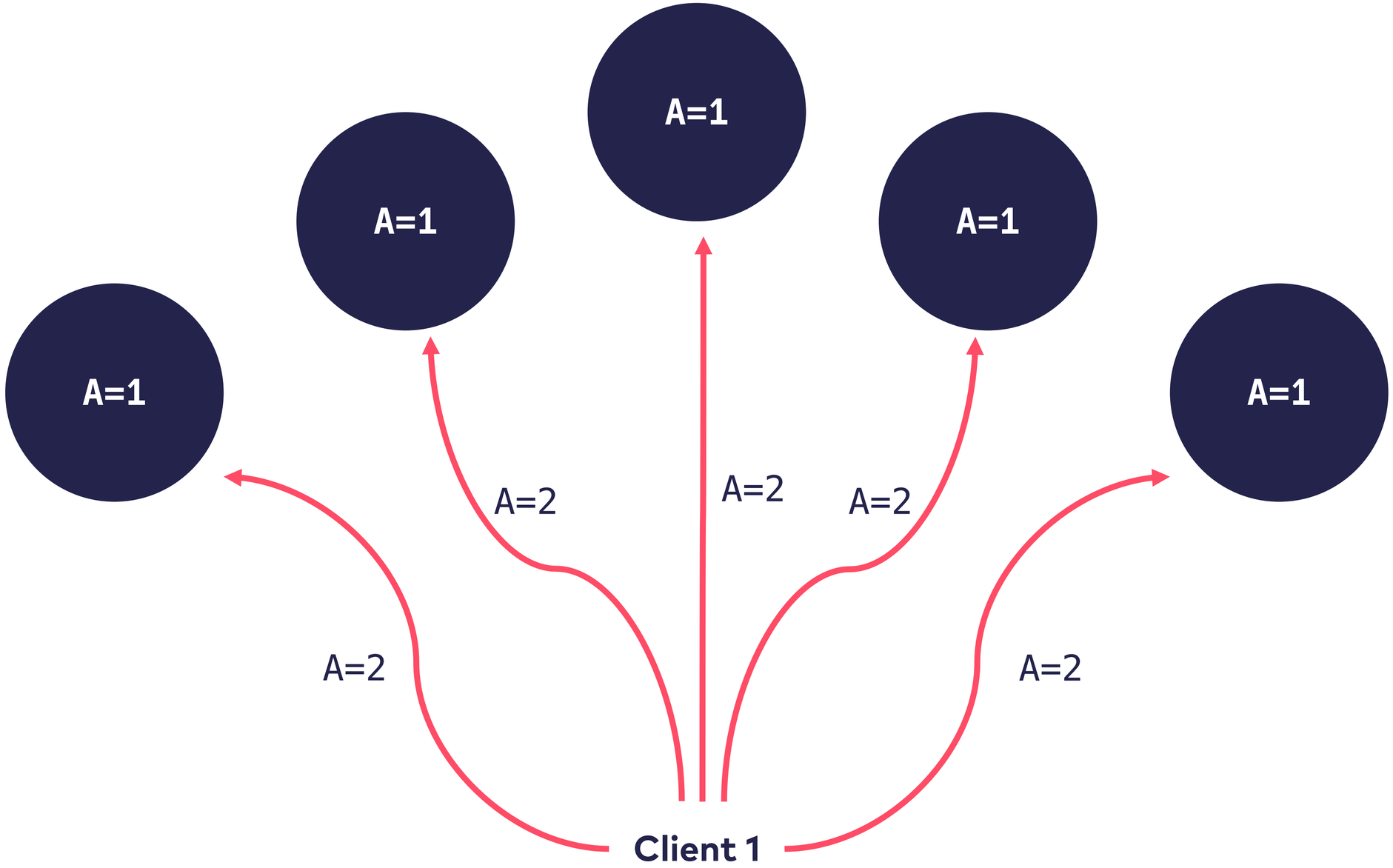
Clients write to n nodes at once

When more than w nodes acknowledged the write, the write is successful

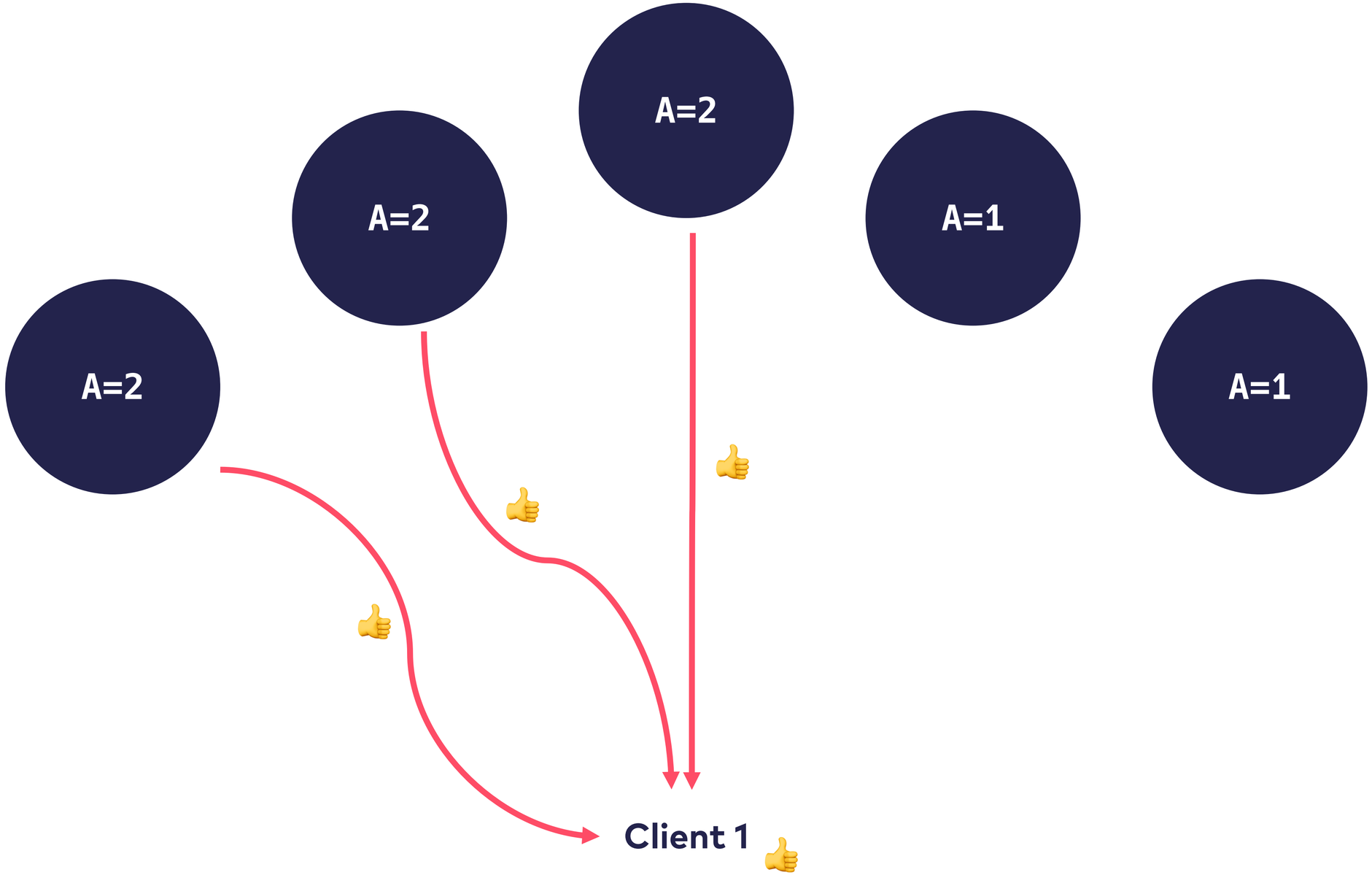
w is the write quorum

When we read, we wait for the result of r nodes

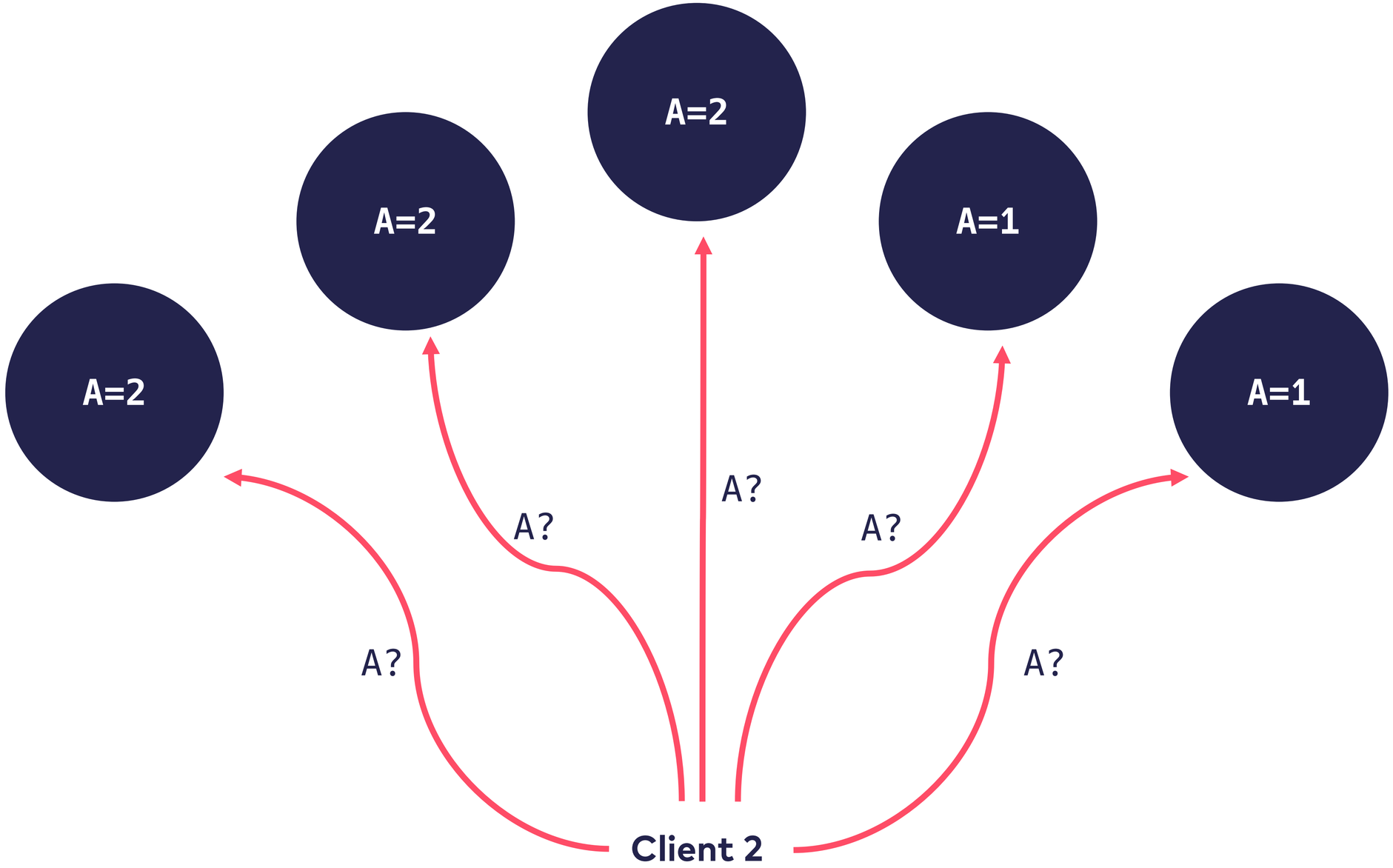
r is the read quorum



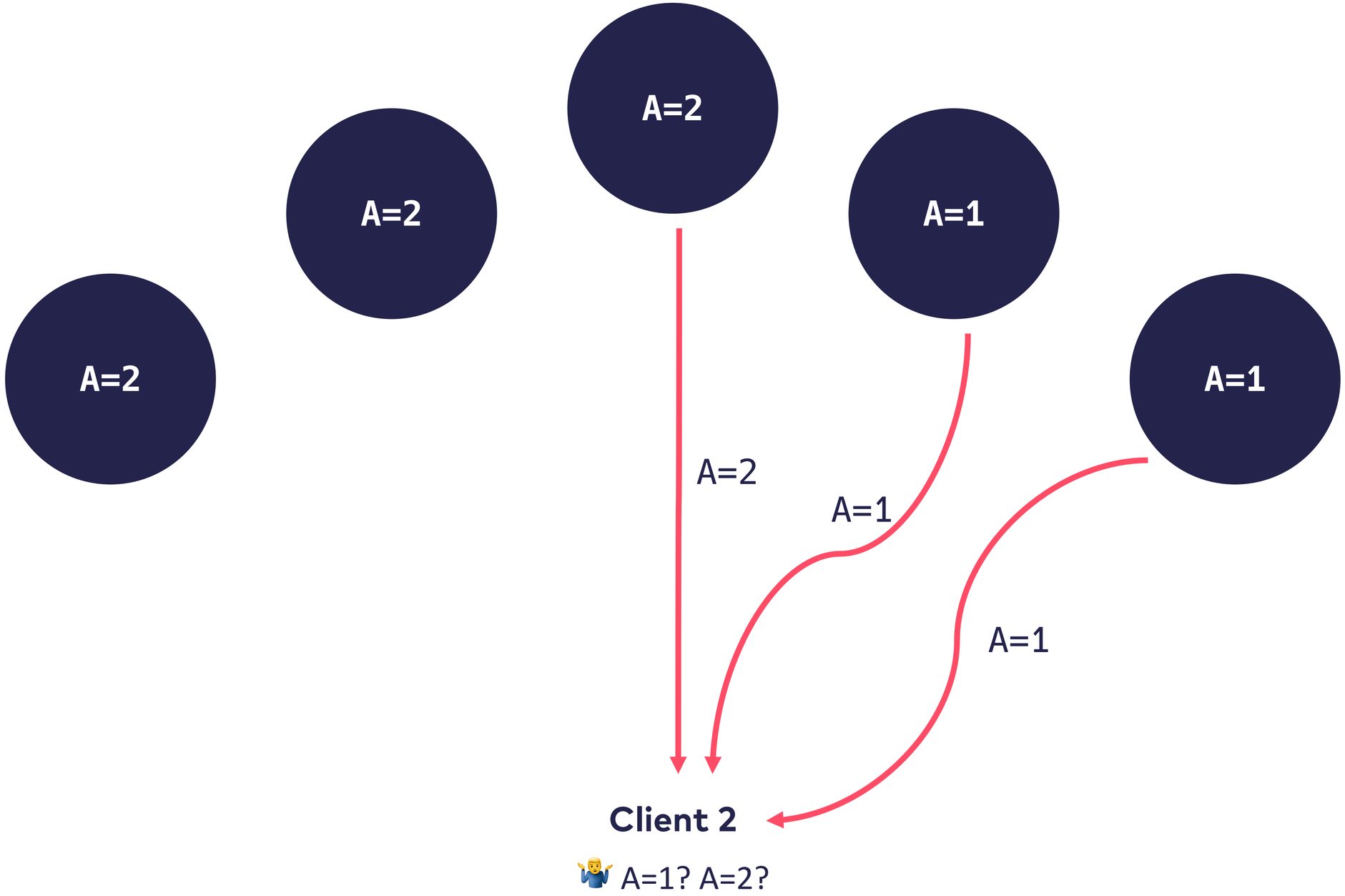
n=5
w=3
r=3



n=5
w=3
r=3



n=5
w=3
r=3



n=5
w=3
r=3

Given W+R>N

Do we always receive the correct result?

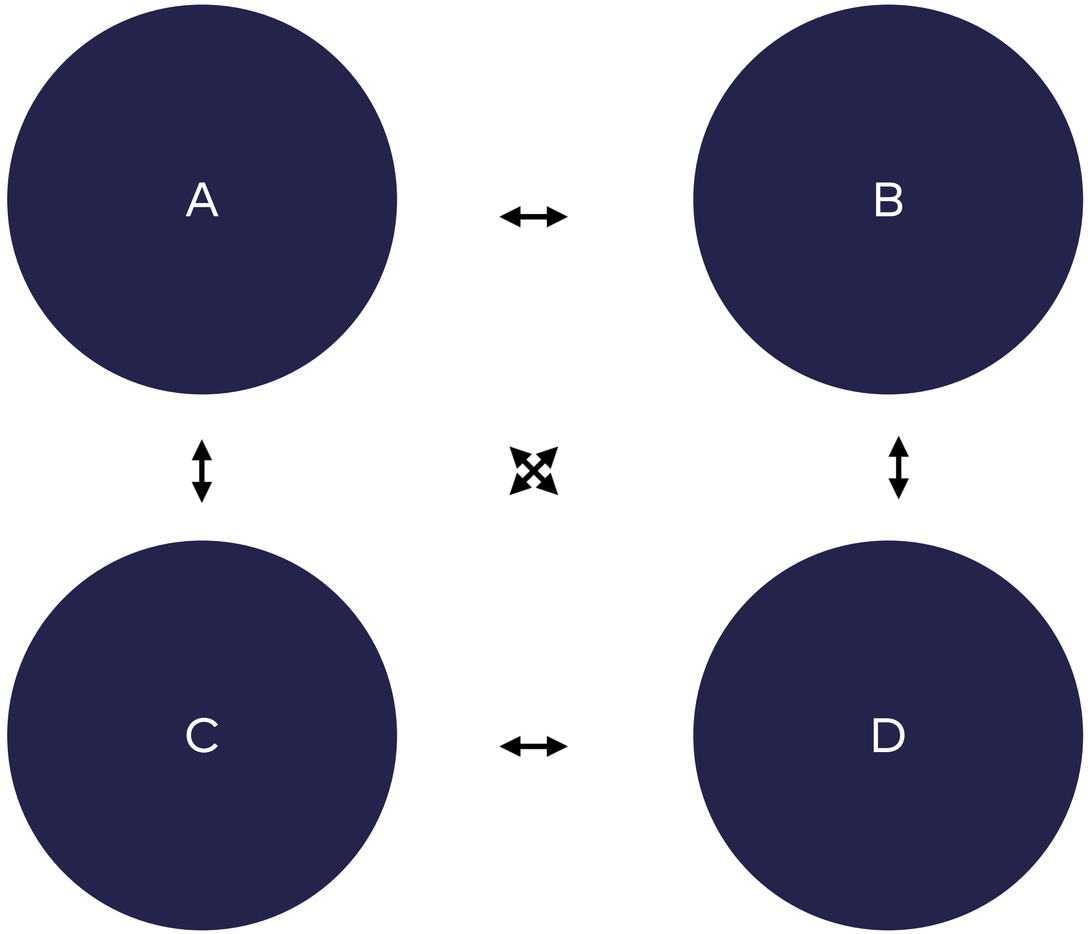
When the node comes back up,
Is it restored with A=1 or A=2?

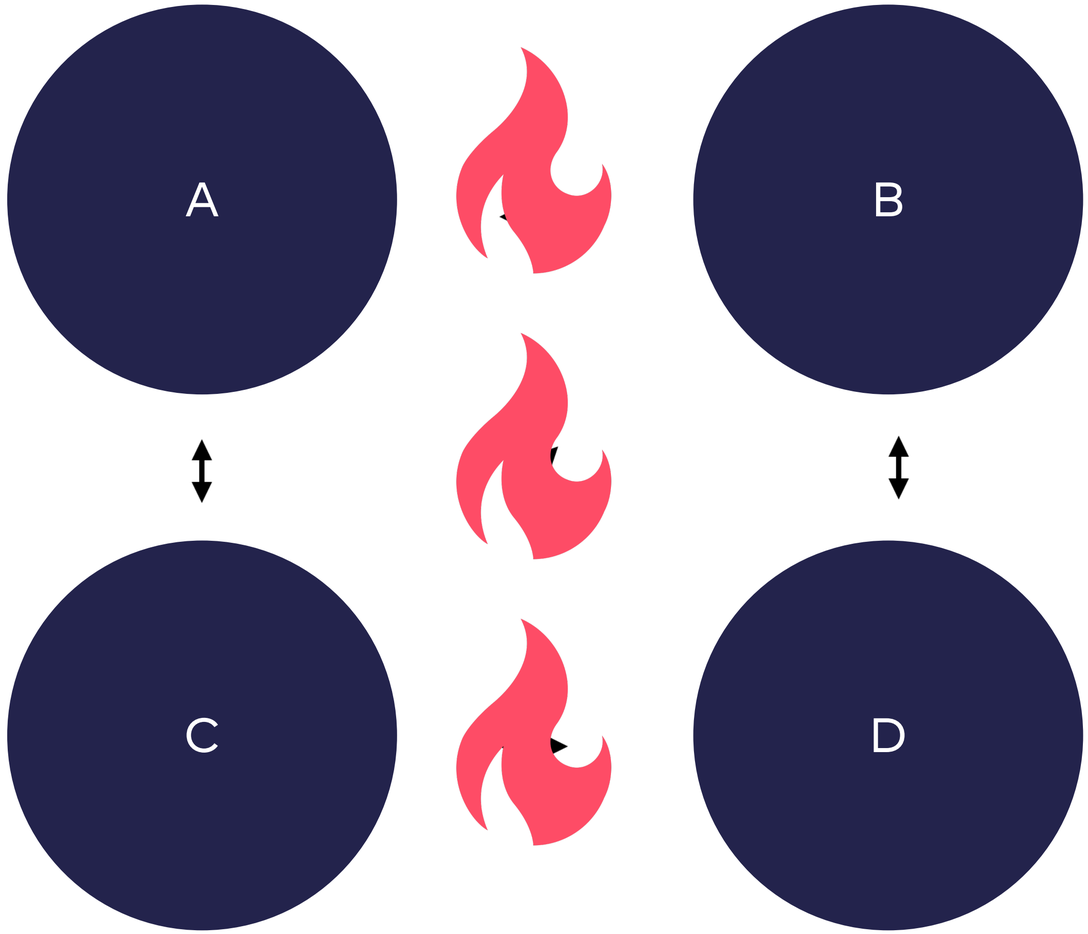
Client

`w=3`
`r=3`

Availability

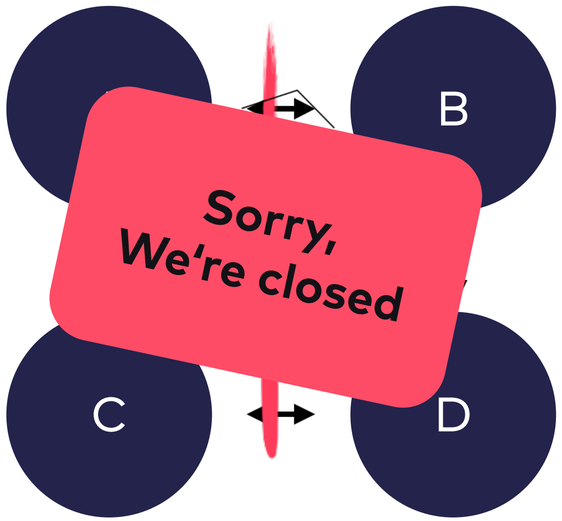
Photo by Ken Treloar on Unsplash


A
B
C
D



Latency is only distinguishable from failure by a timeout

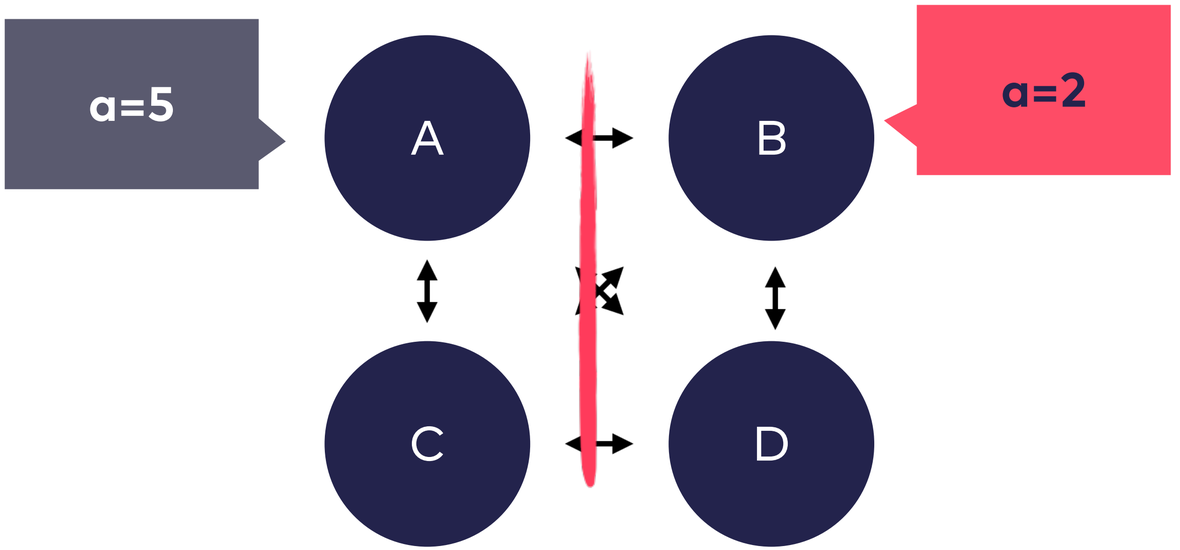
You have two choices



Stop taking requests

Not available, but consistent under partition

CP

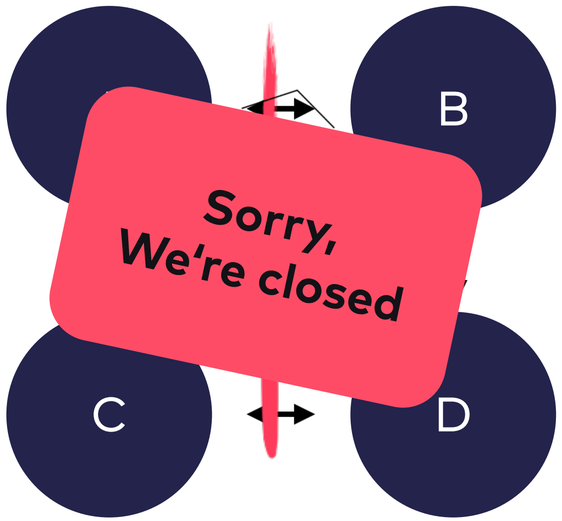


Continue taking requests

Available, but not consistent under partition

AP

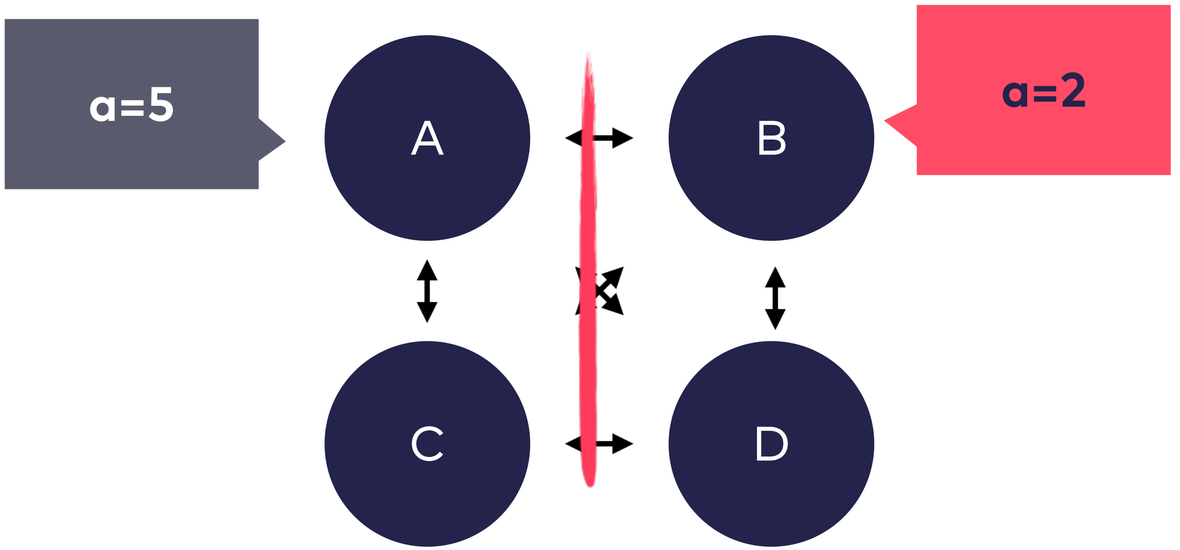
CP



Not available, but consistent under partition

~Single Leader, Paxos

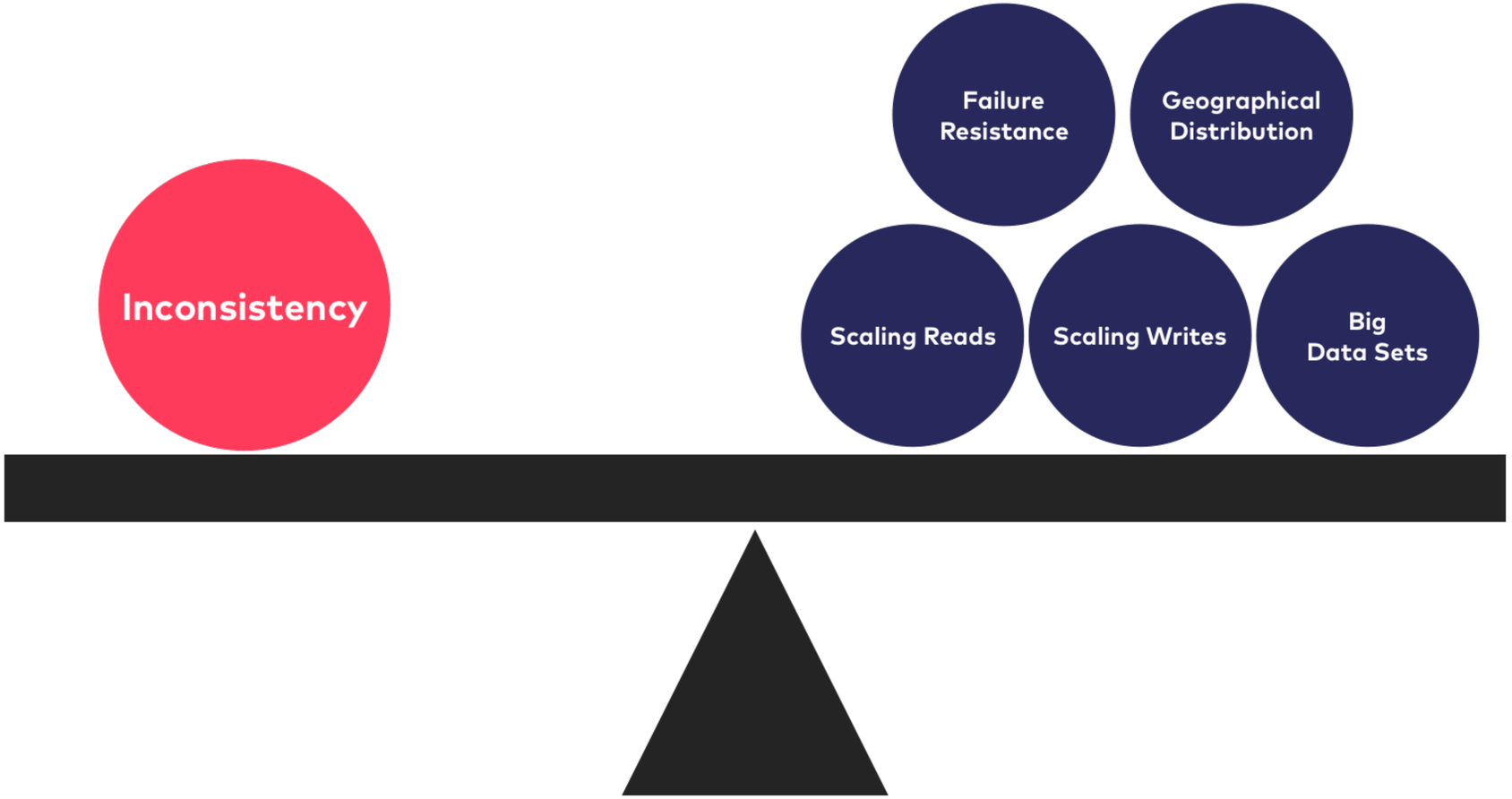
AP



Available, but not consistent under partition

~Leaderless, Multi Leader

What are your requirements?



https://leanpub.com/datenbanken