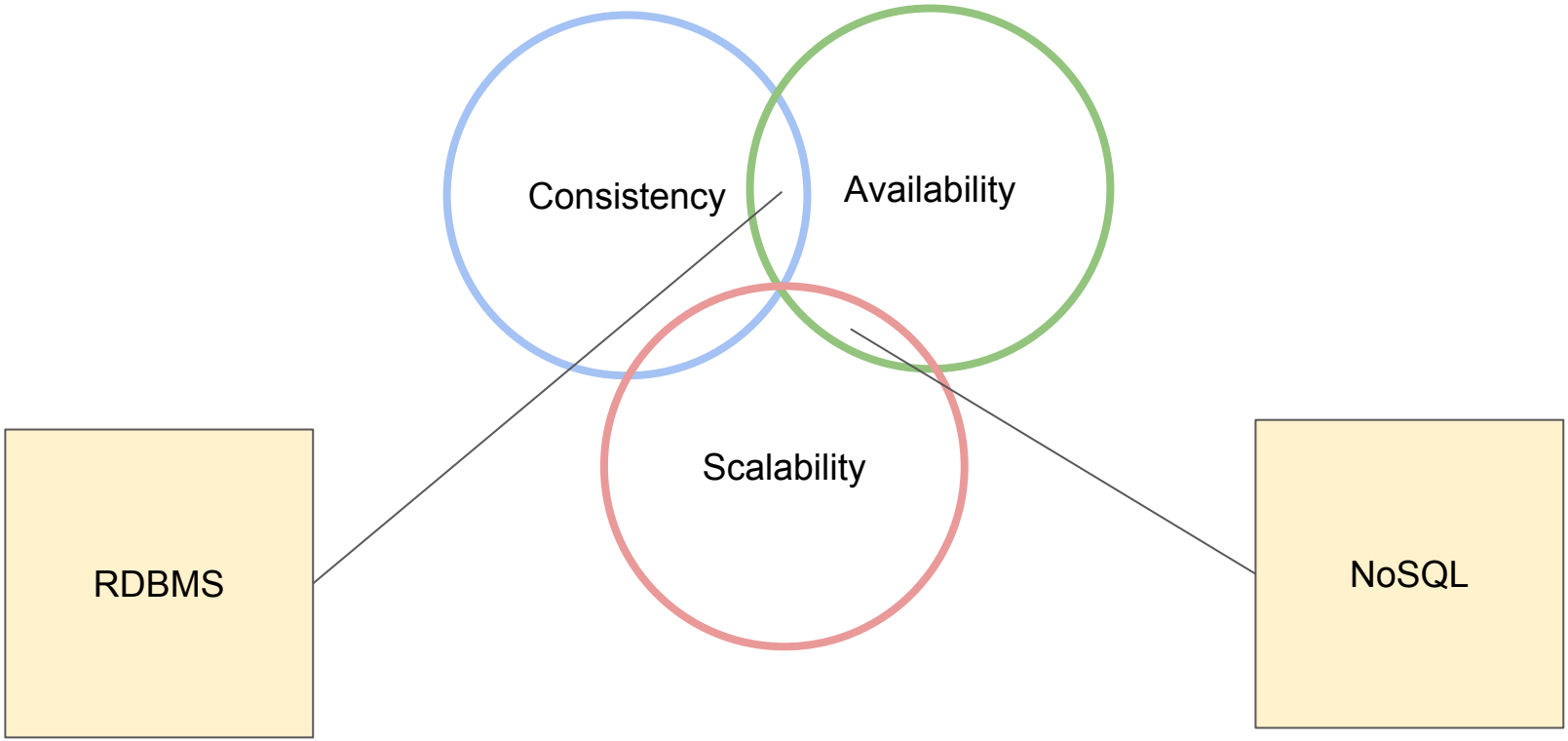
Google Megastore & Google Spanner

The Problem

I have a great app idea, but I don’t want to learn much about databases. Can’t it just be easy and scalable and reliable?

Because that’s impossible!

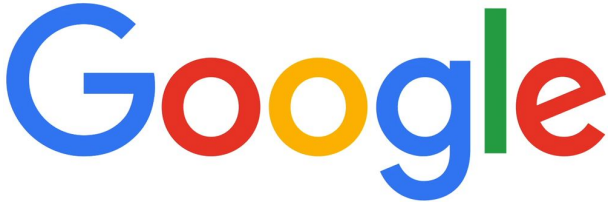



APACHE
HBASE


MySQL


cassandra


PostgreSQL

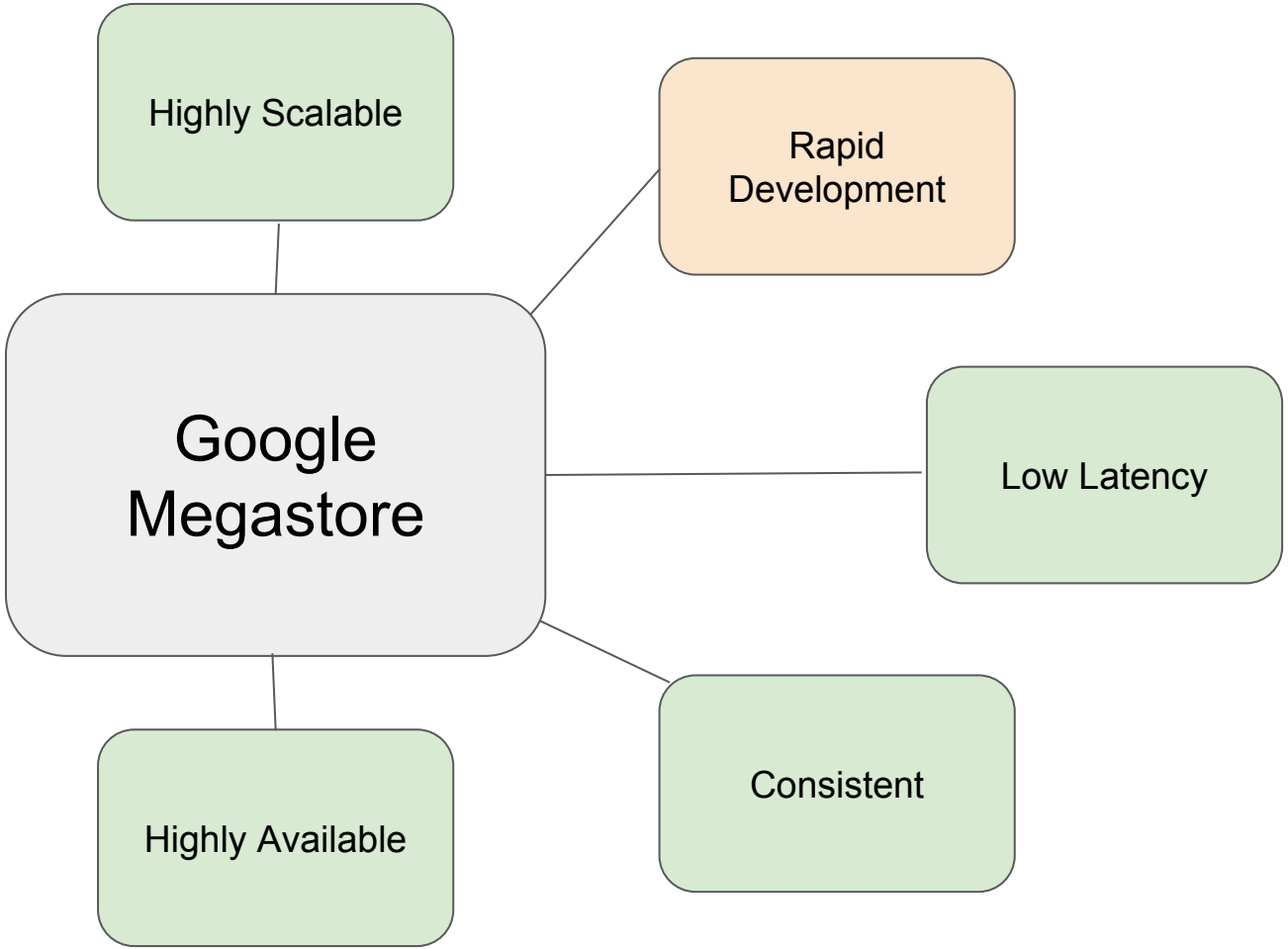

Google


APACHE
HBASE
cassandra
Google

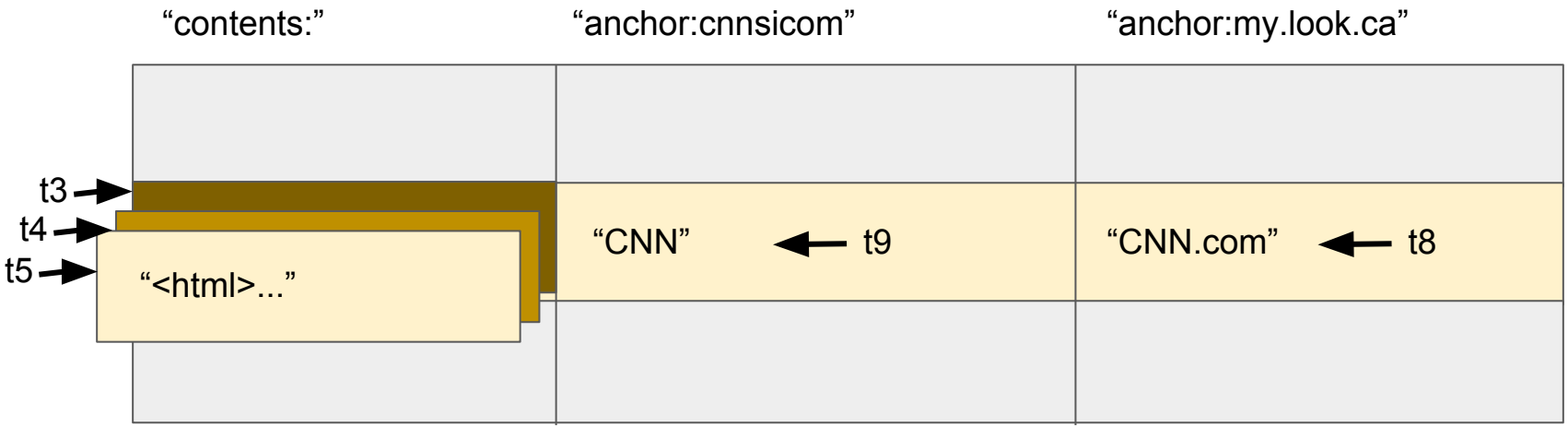

MySQL
PostgreSQL

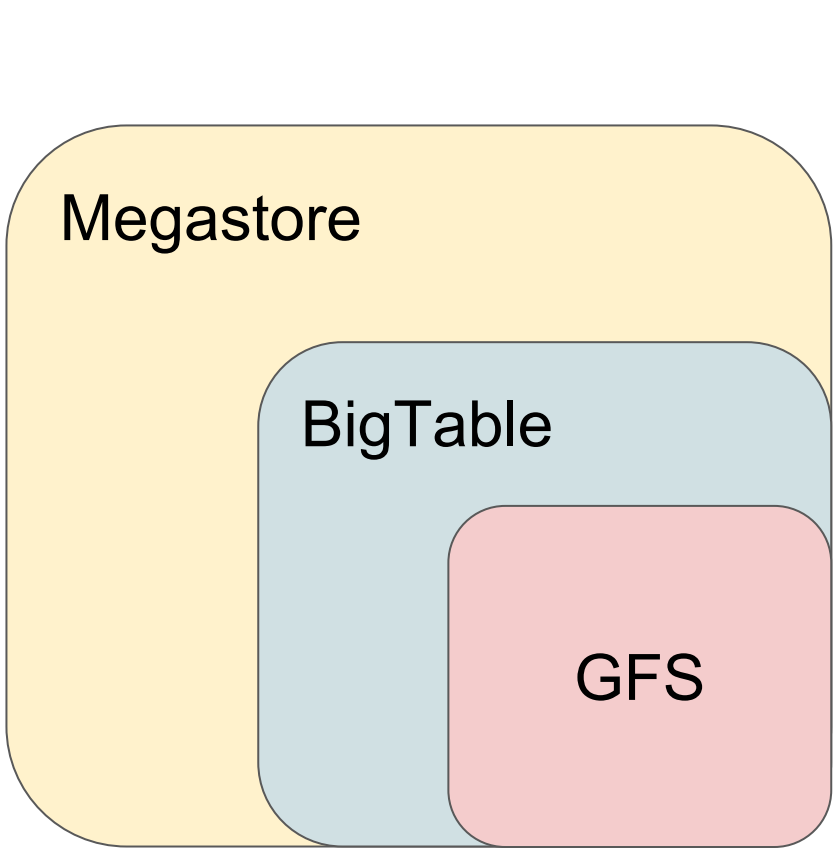

APACHE
HBASE
cassandra
Google

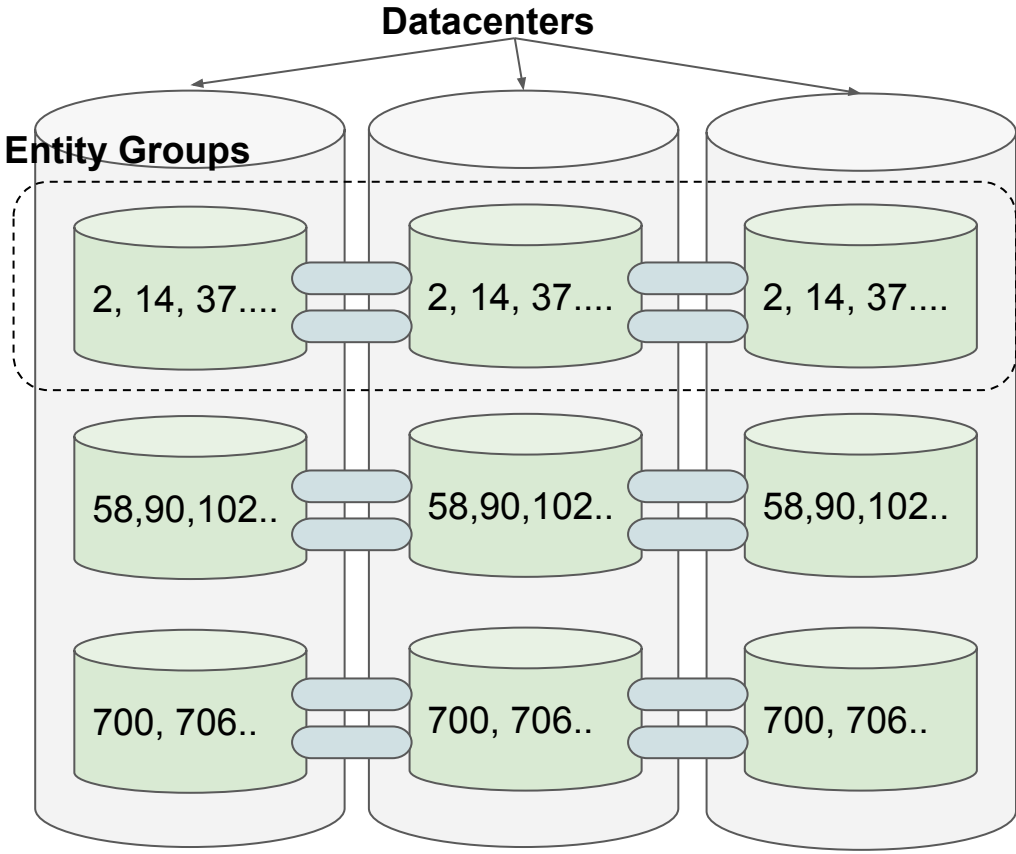

MySQL
PostgreSQL

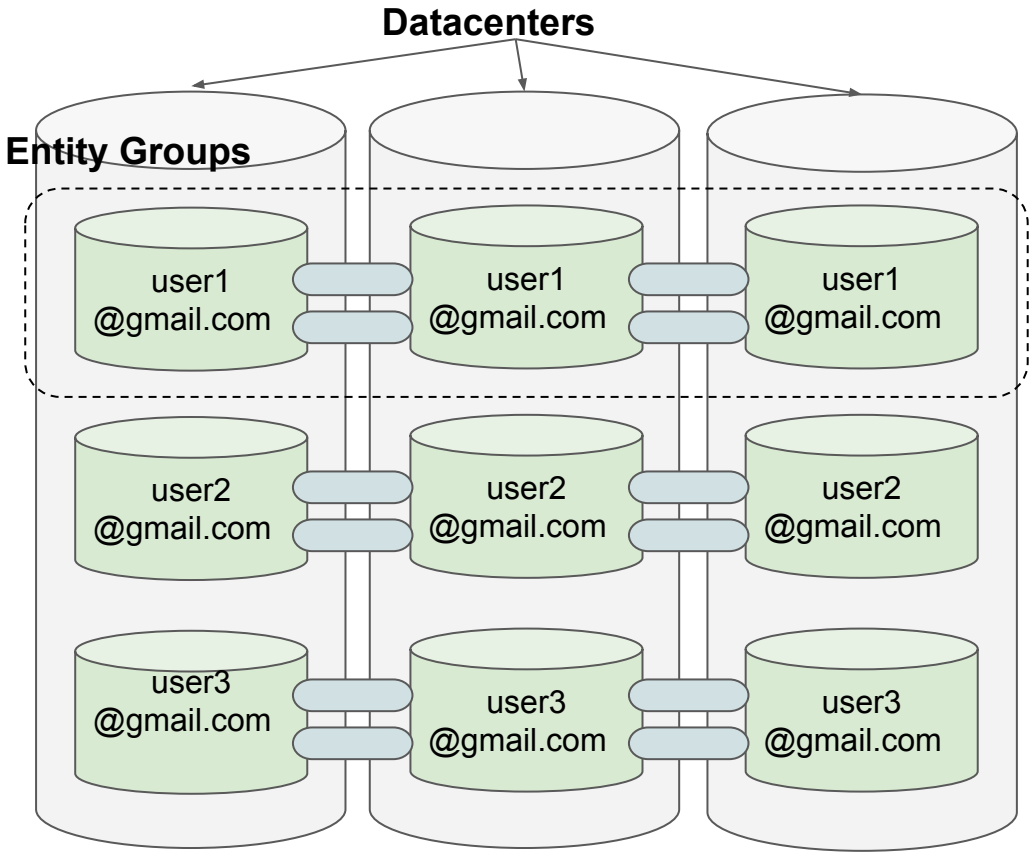

Highly Scalable
Rapid Development
Google Megastore
Low Latency
Highly Available
Consistent

BigTable, Google's distributed KV store



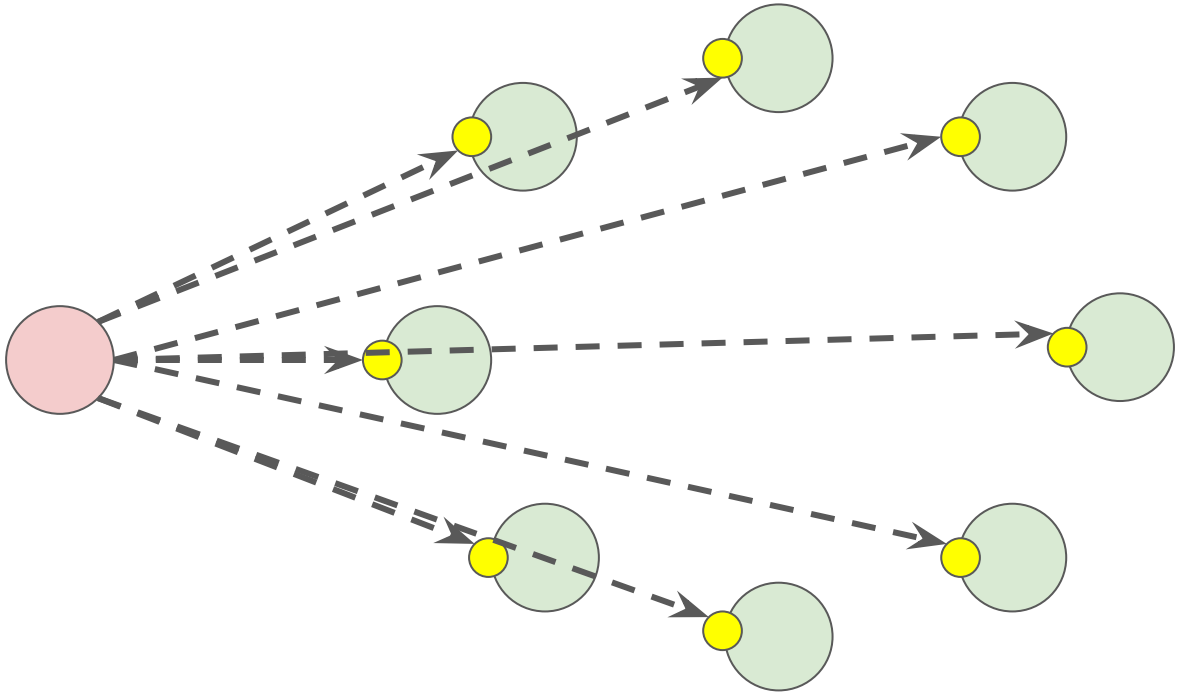

Megastore
BigTable
GFS


Datacenters
Entity Groups
2, 14, 37....
2, 14, 37....
2, 14, 37....
58,90,102..
58,90,102..
58,90,102..
700, 706..
700, 706..
700, 706..

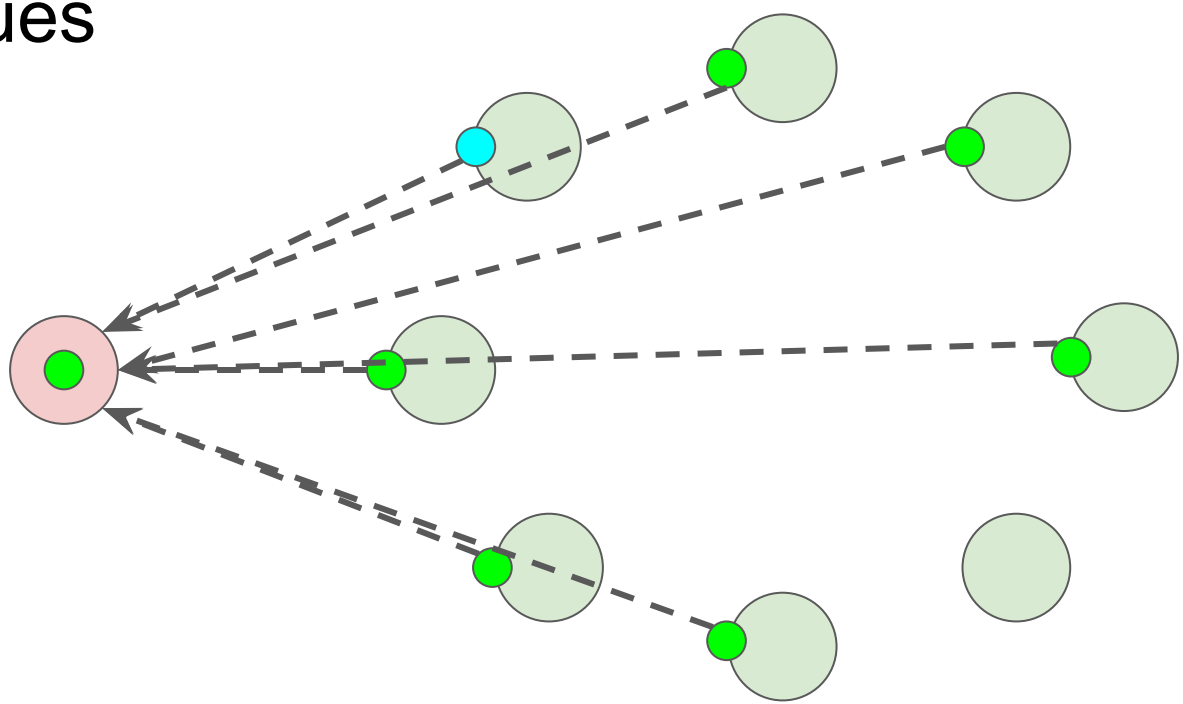

Datacenters
Entity Groups
user1
@gmail.com
user1
@gmail.com
user1
@gmail.com
user2
@gmail.com
user2
@gmail.com
user2
@gmail.com
user3
@gmail.com
user3
@gmail.com
user3
@gmail.com

Paxos Basics - Read and Write

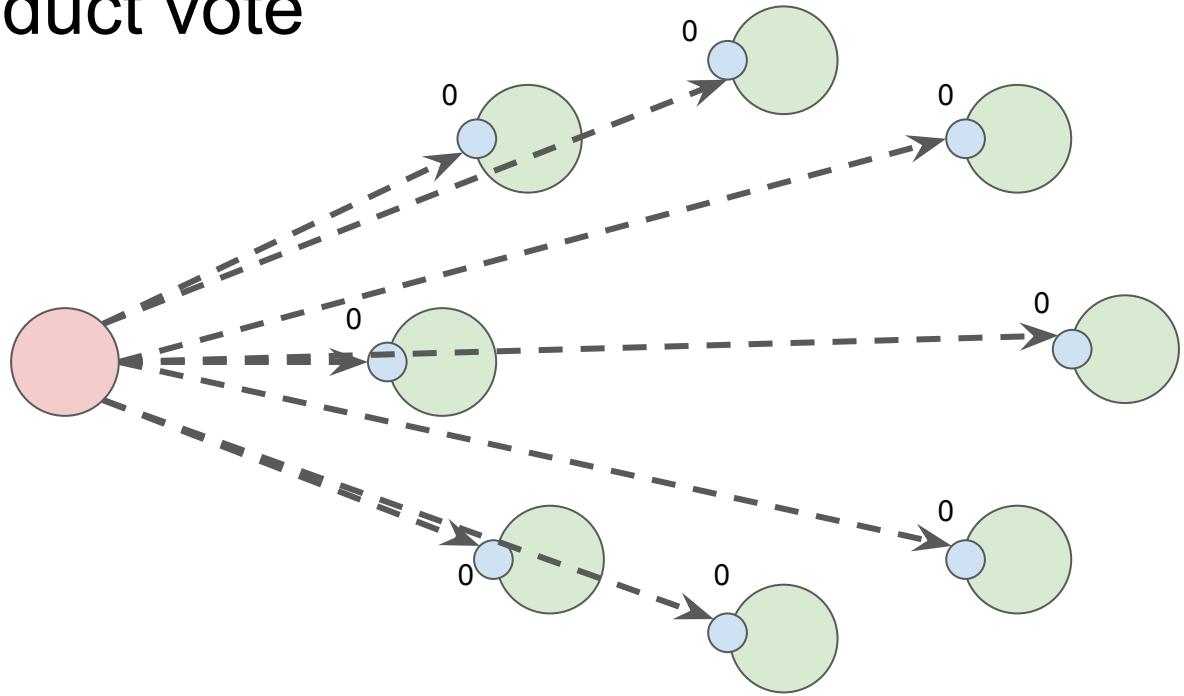
Read, Phase 1 - Reading replica polls other replicas



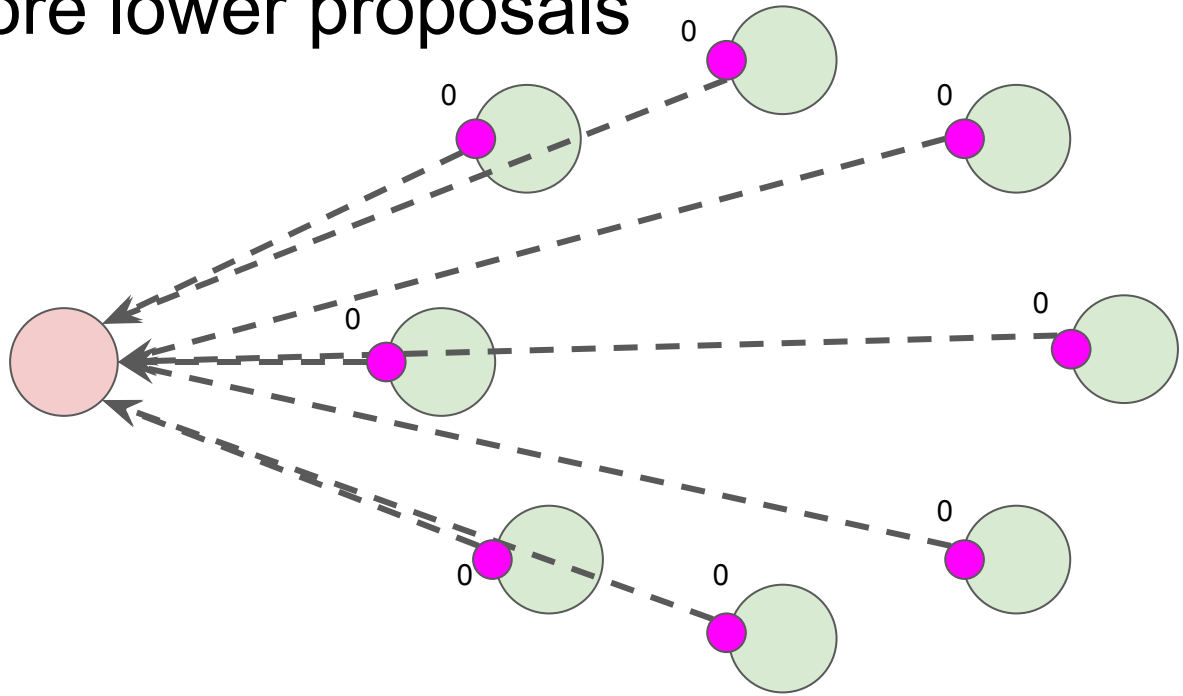
Read, Phase 2 - Available replicas respond with their values



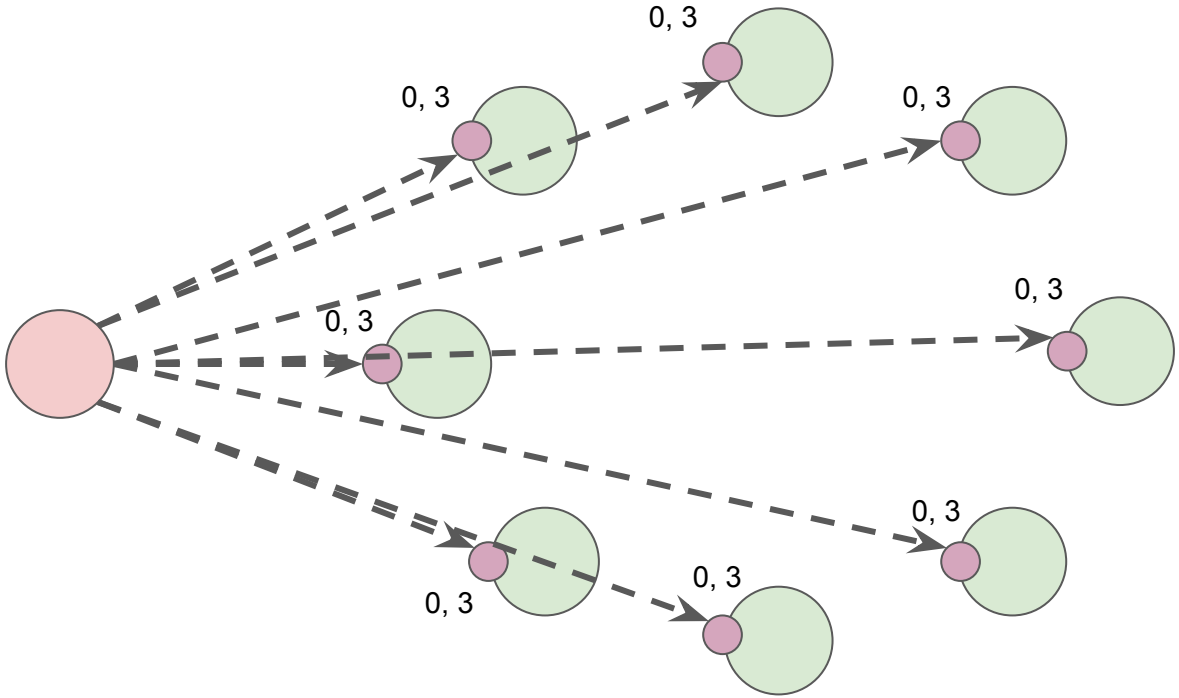
Write, Prepare - Writing replica asks other replicas to conduct vote



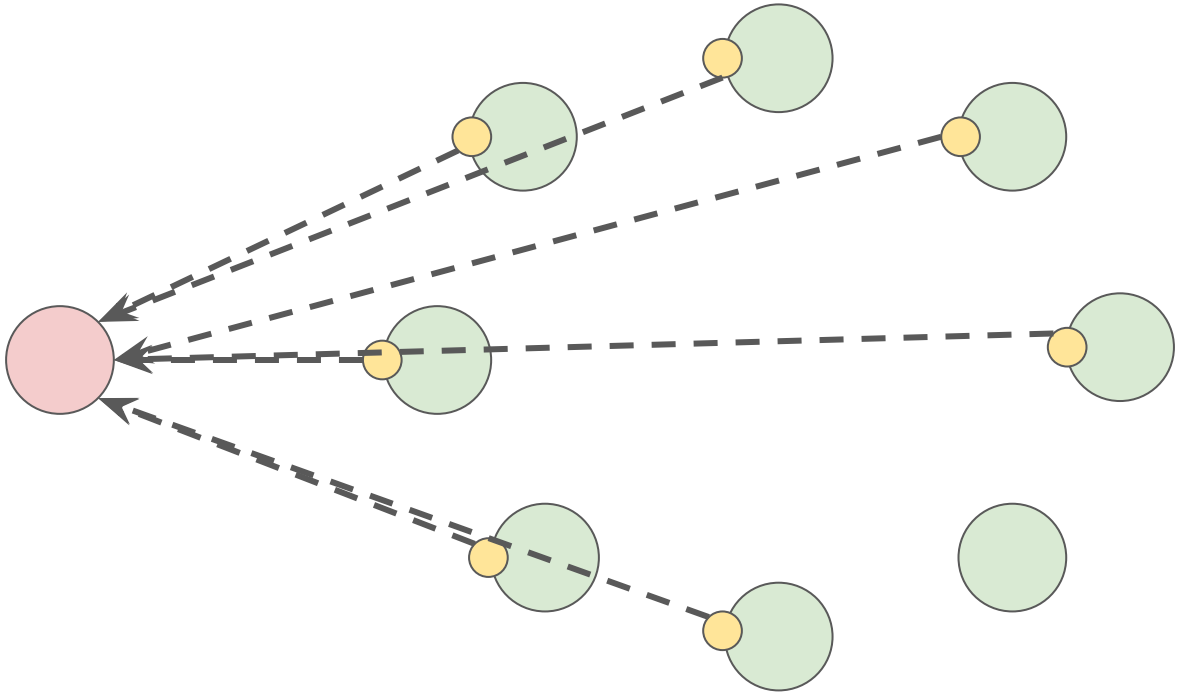
Write, Promise -- Available replicas promise to ignore lower proposals



Write, Accept - Writing replica proposes its value

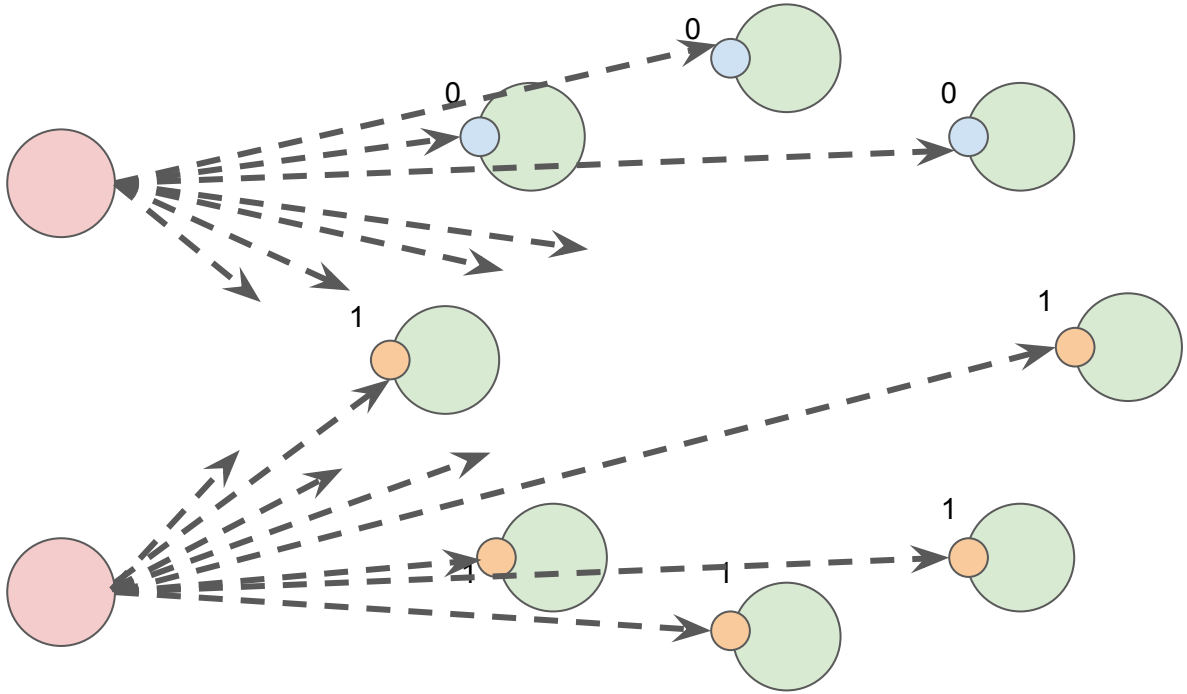


Write, Commit - Available replicas accept the value

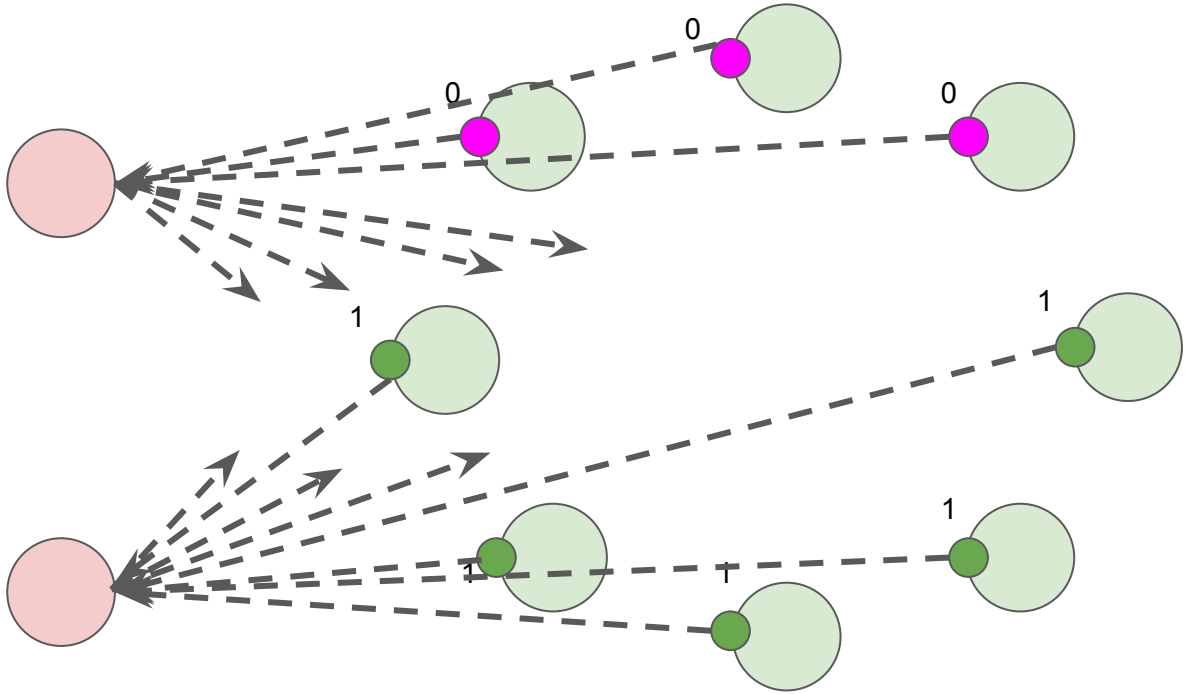


Paxos Basics -- Write Conflicts

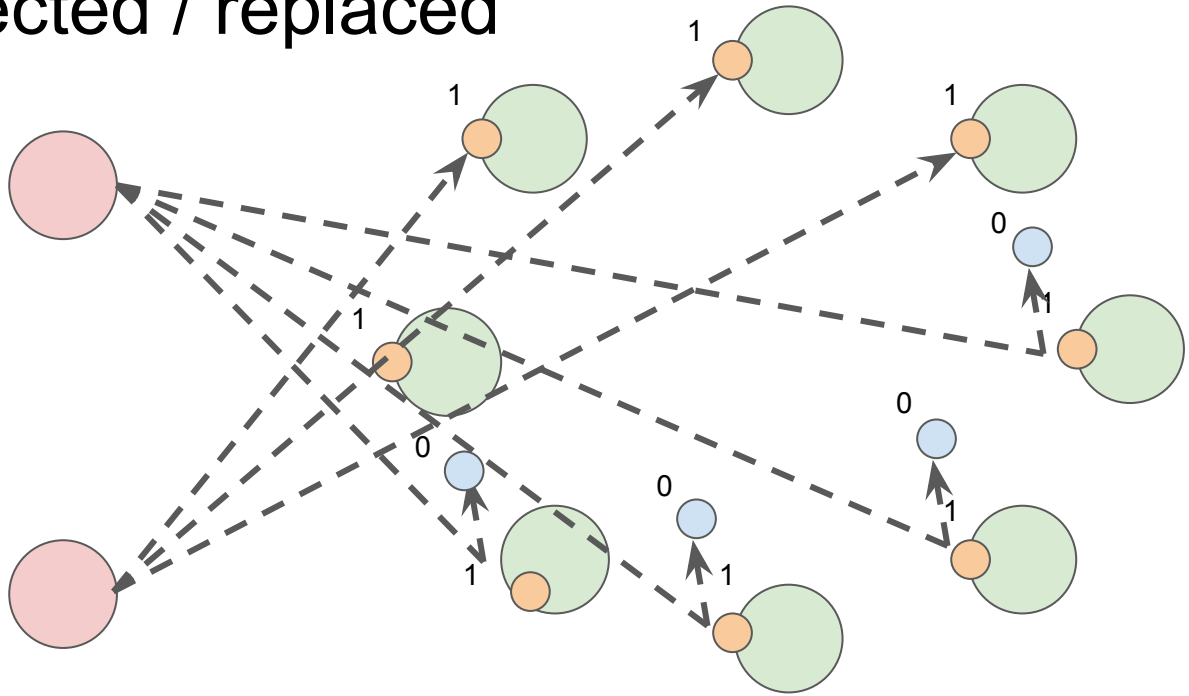
Prepare - 2 writing replicas want to make proposals



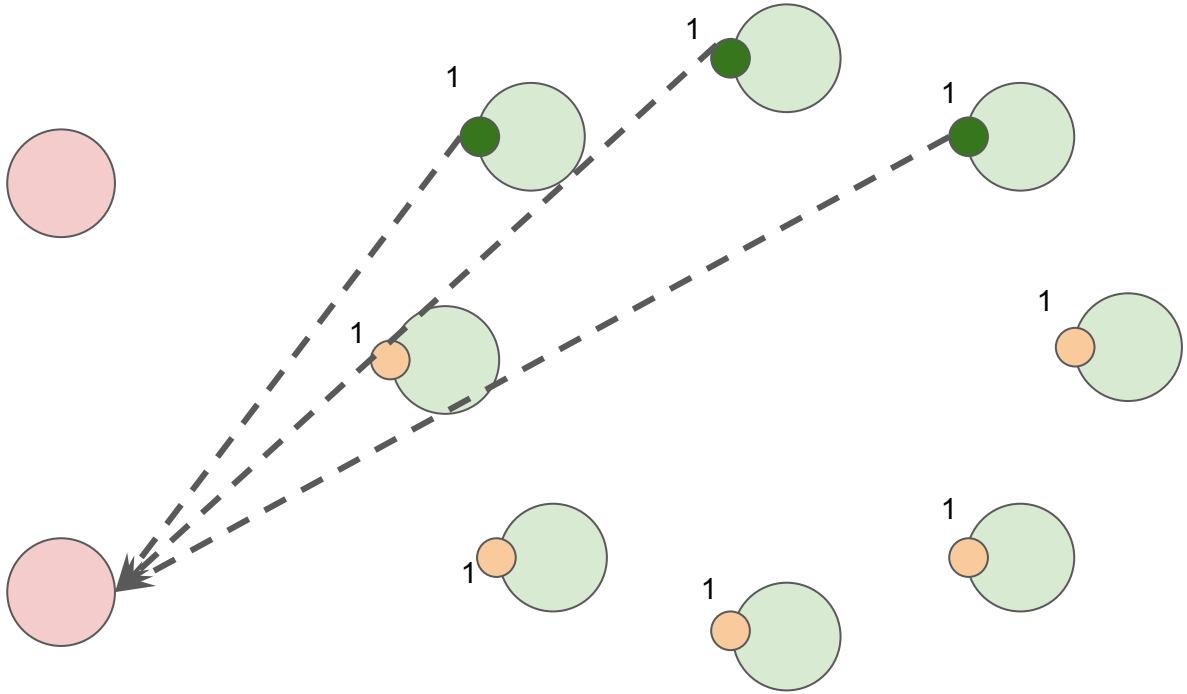
Promise



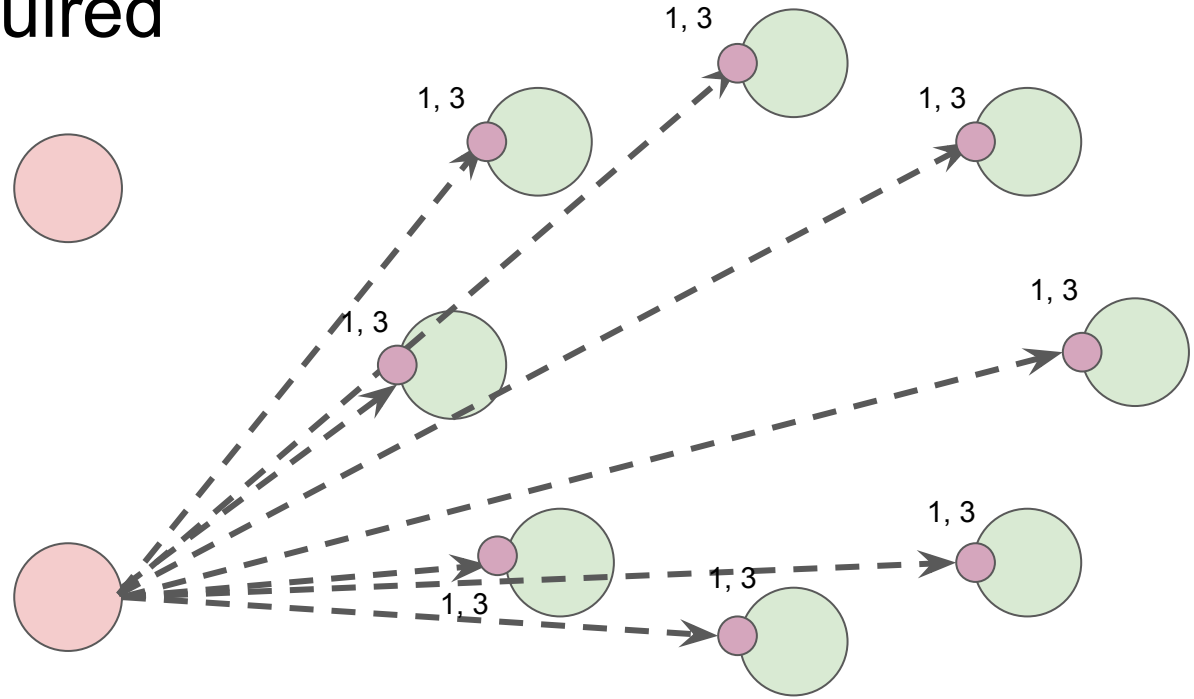
More prepares - Lower numbered proposals get rejected / replaced



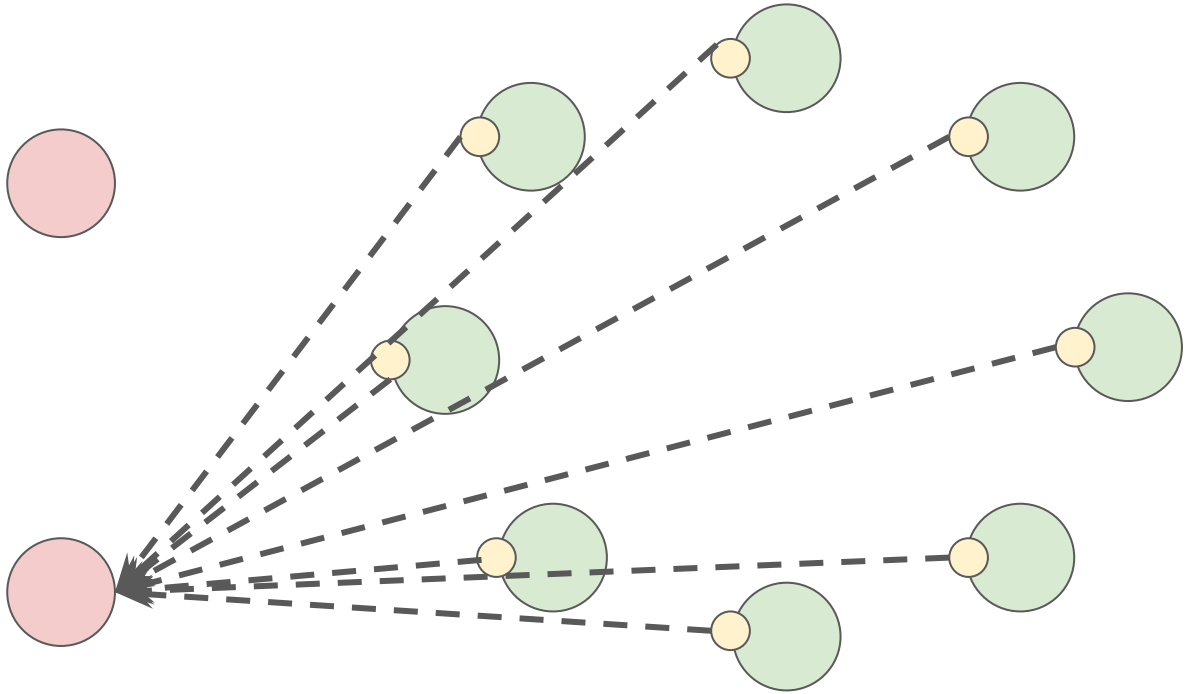
More Promises



Accept - Only 1 replica gets the majority of promises required

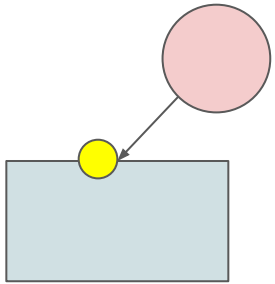


Commit

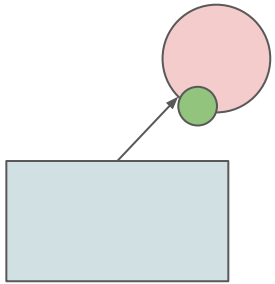


How could Paxos be made more efficient?

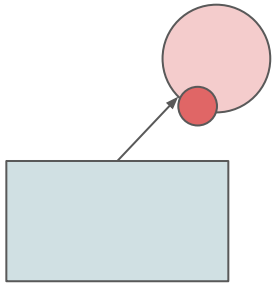
Local Reads via a “coordinator”



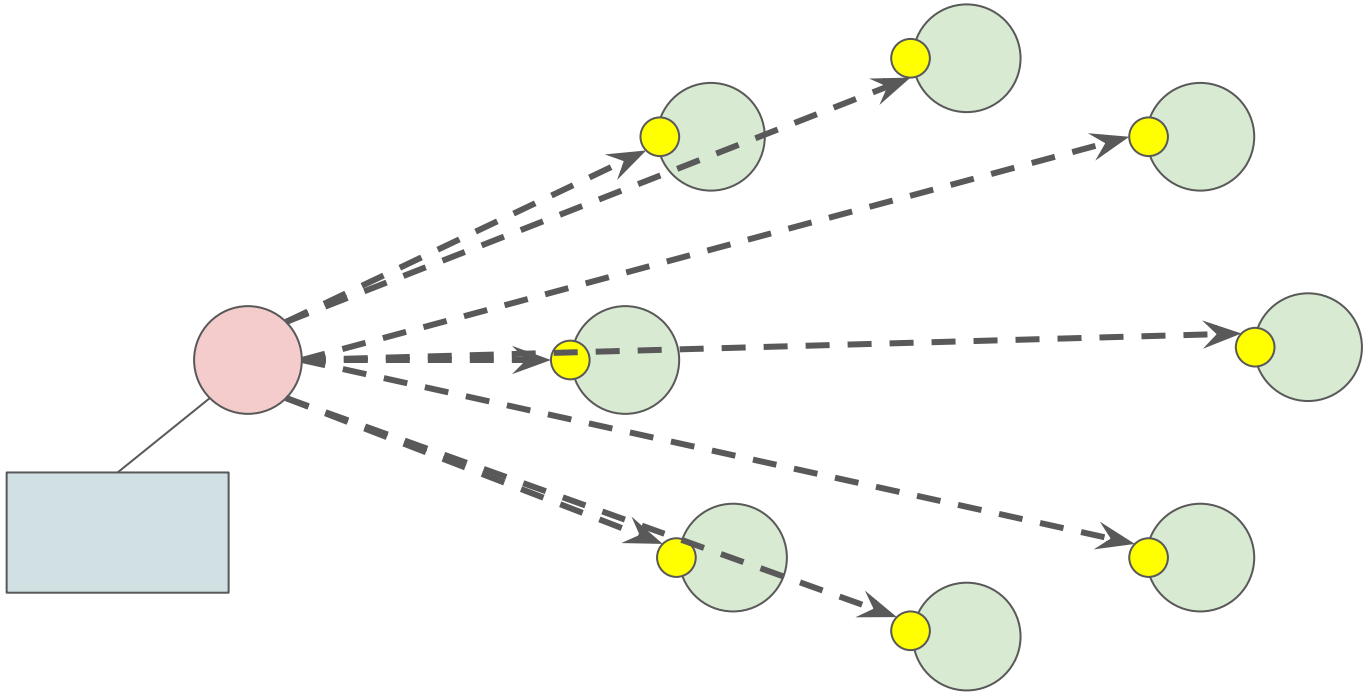
Local replica up-to-date



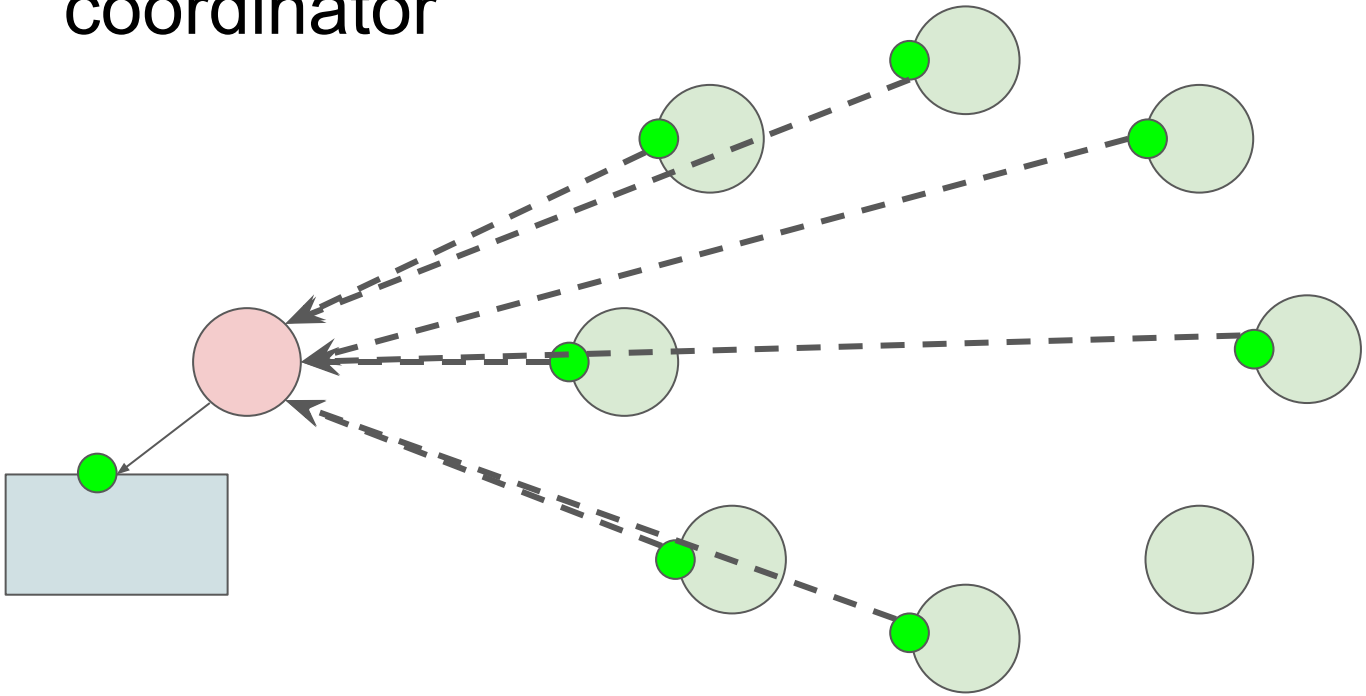
Local replica out-of-date



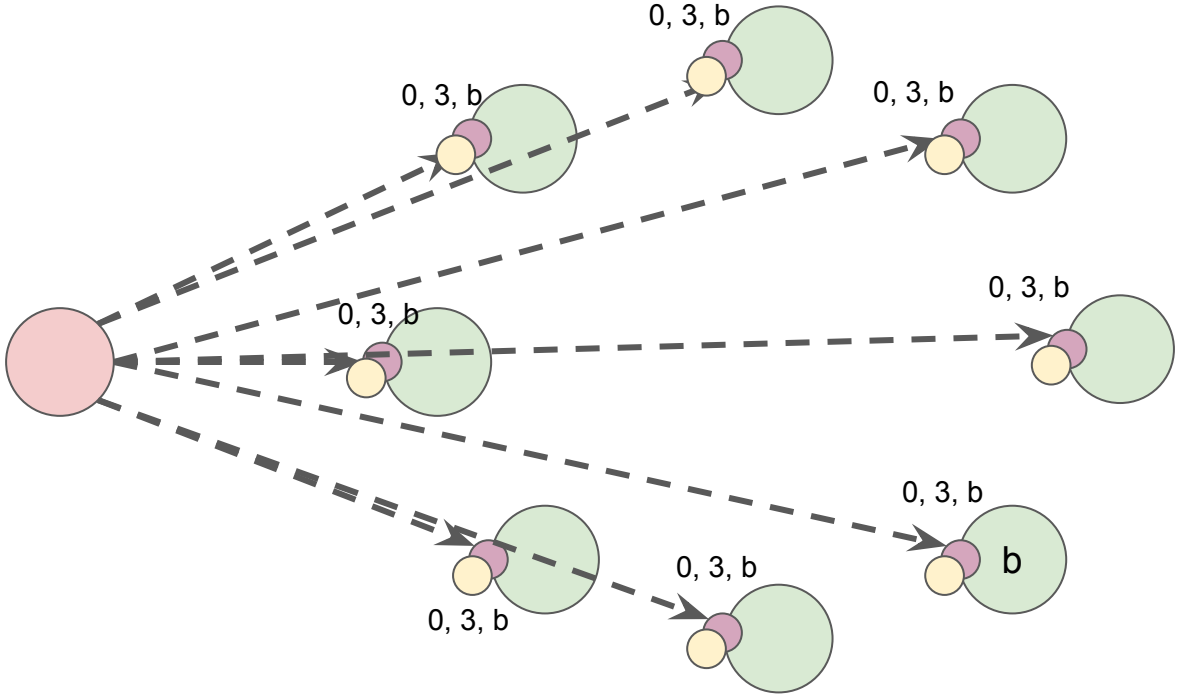
Local replica out-of-date - normal Paxos read



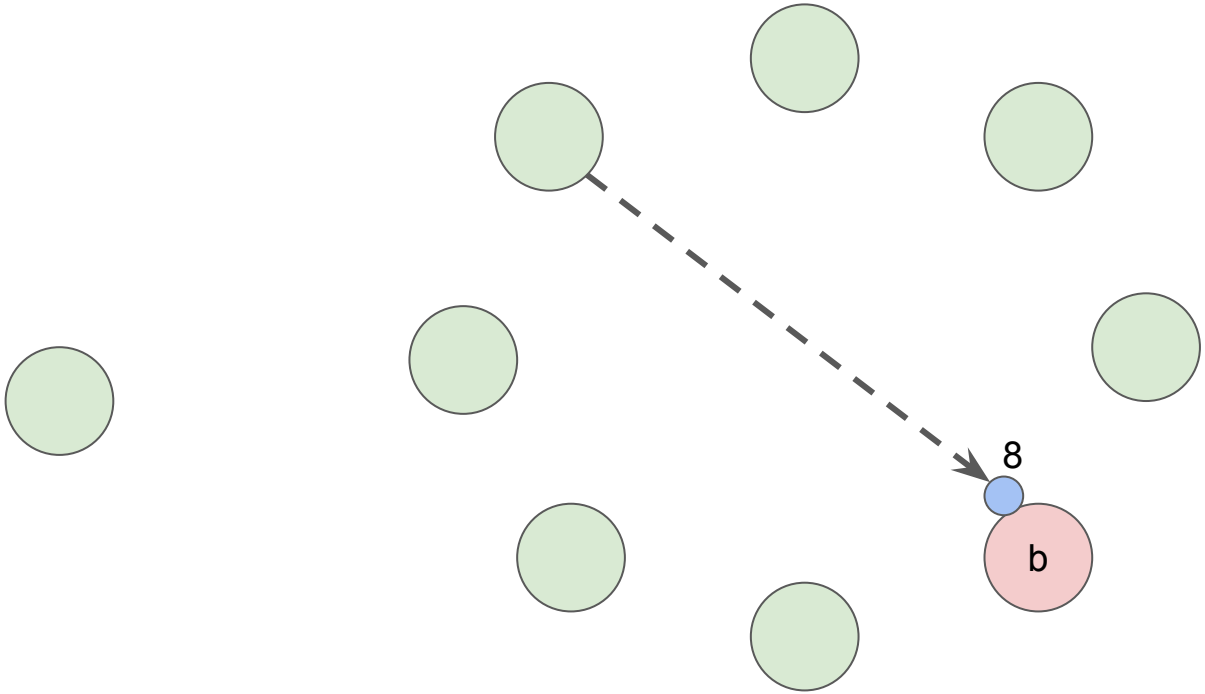
Local replica out-of-date - update replica and coordinator



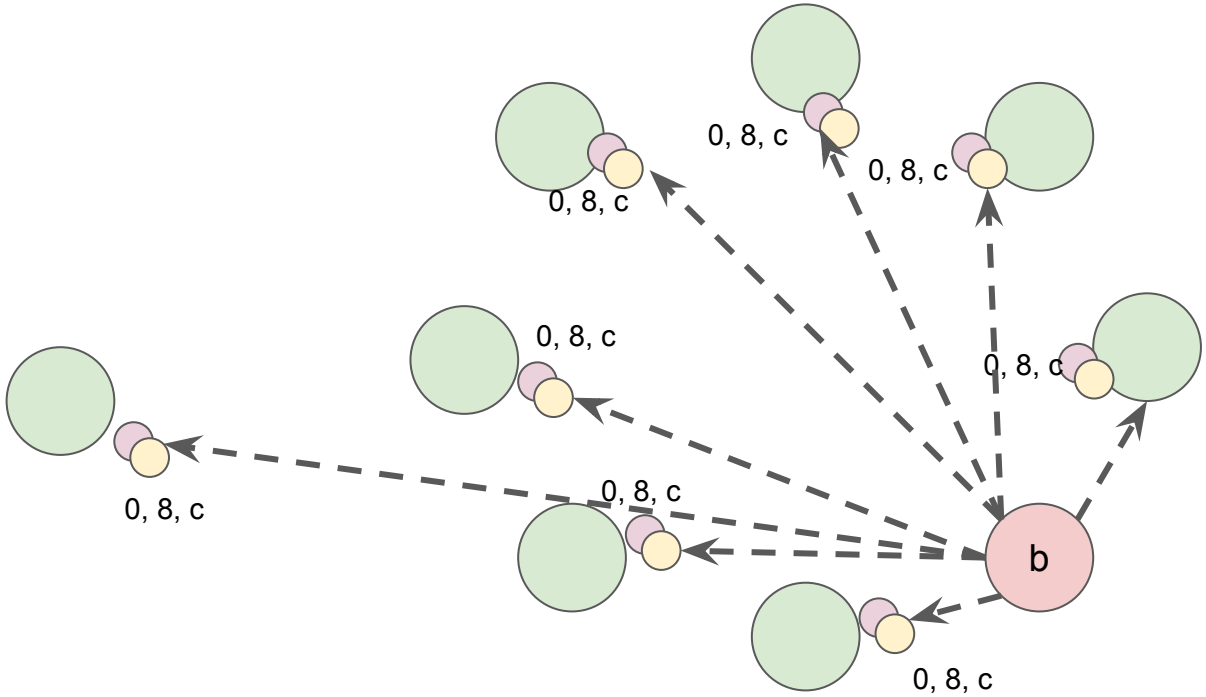
Faster Writes: Accept & Prepare in 1 communication



Writing replica requests to be proposal 0

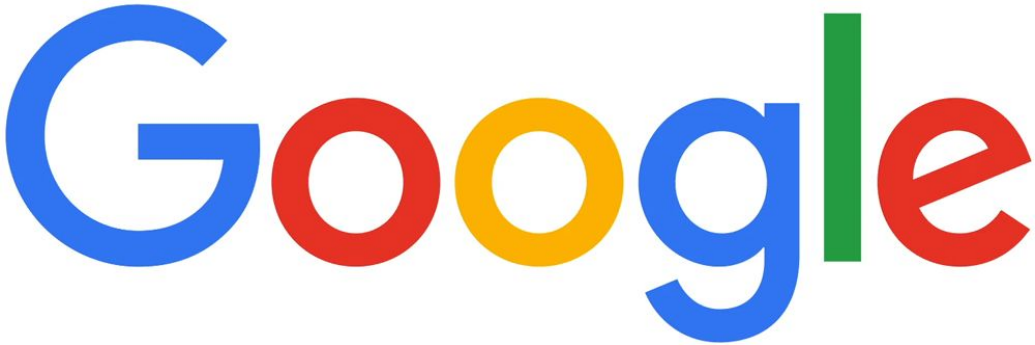


Immediate Accept & Prepare for next round



Megastore Performance Analysis

Performance Analysis



Our experimental setup is that we've used Megastore for 100+ applications for several years and it works!

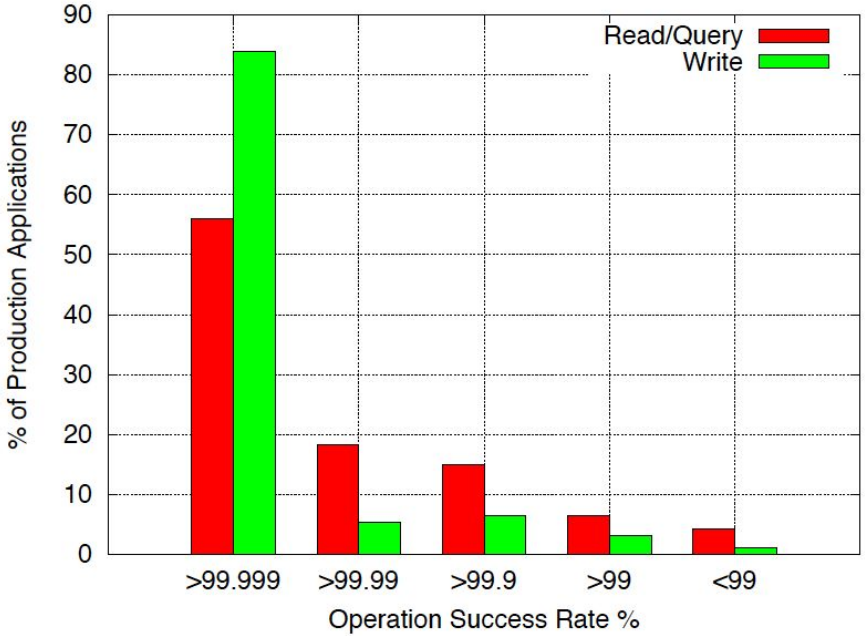


Figure 9: Distribution of Availability

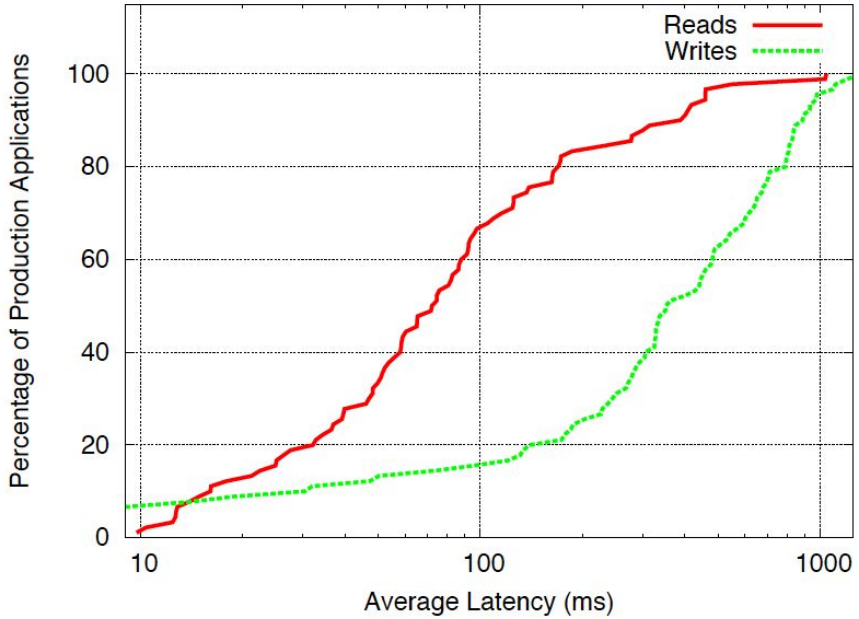
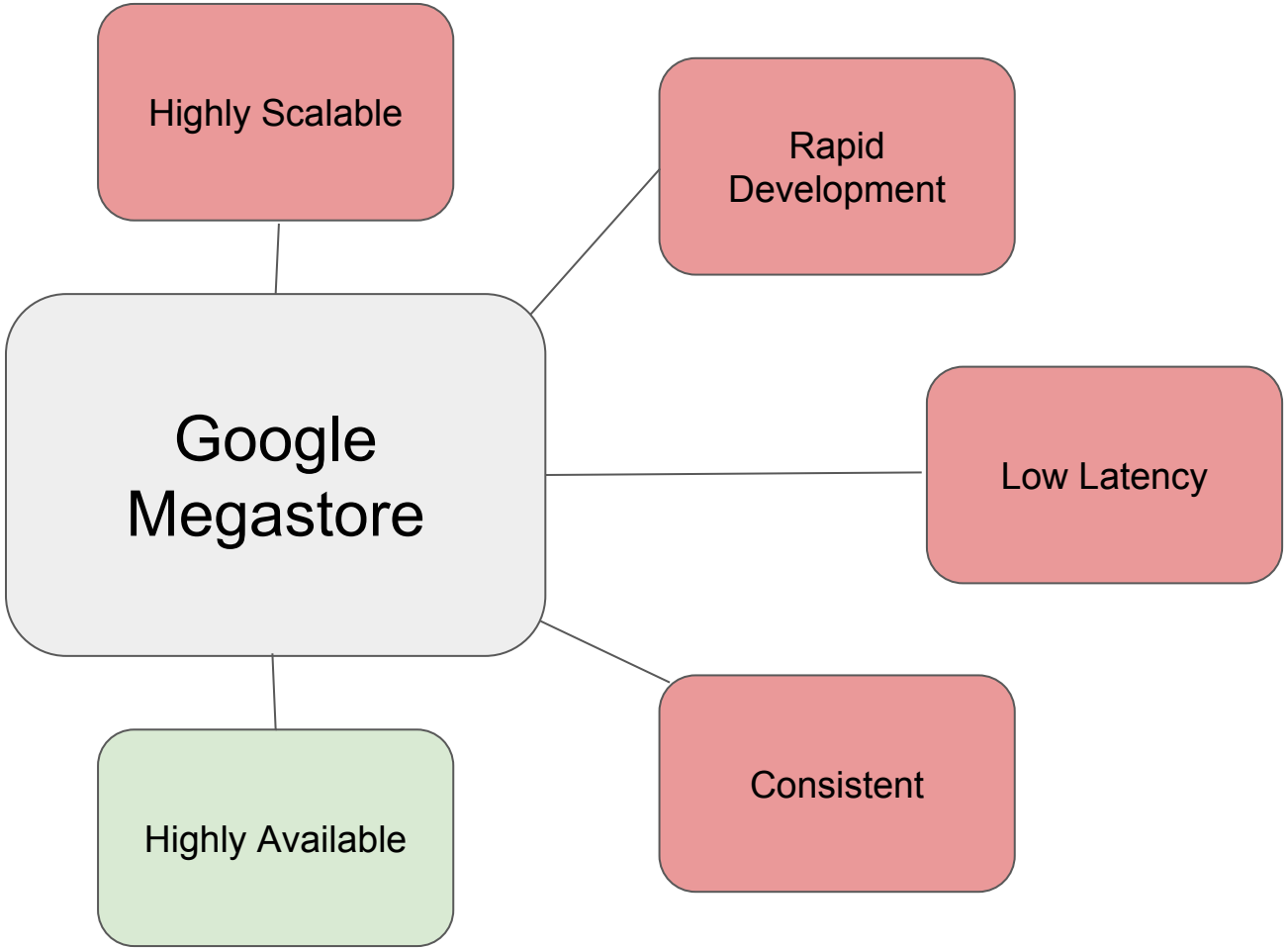
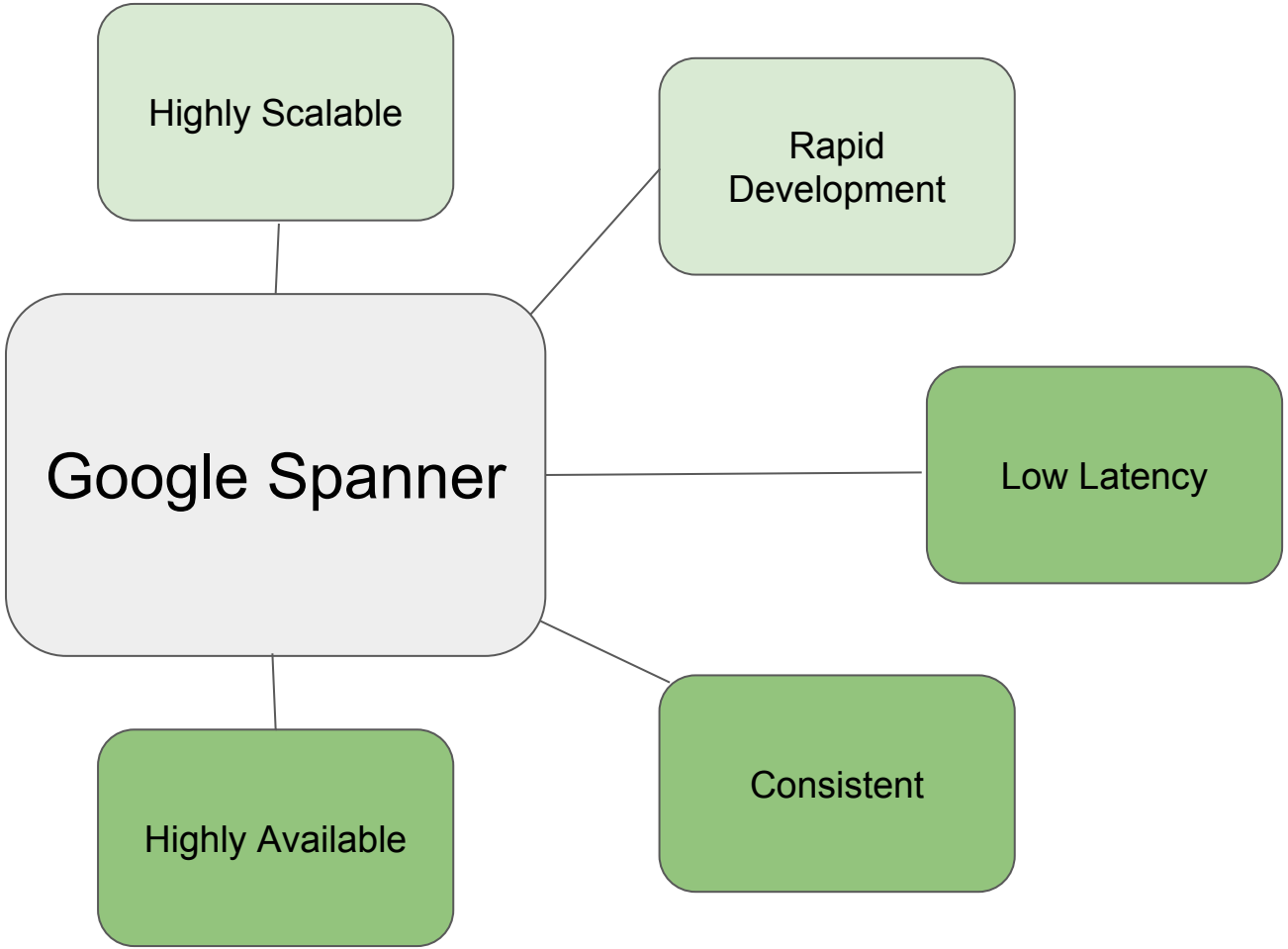


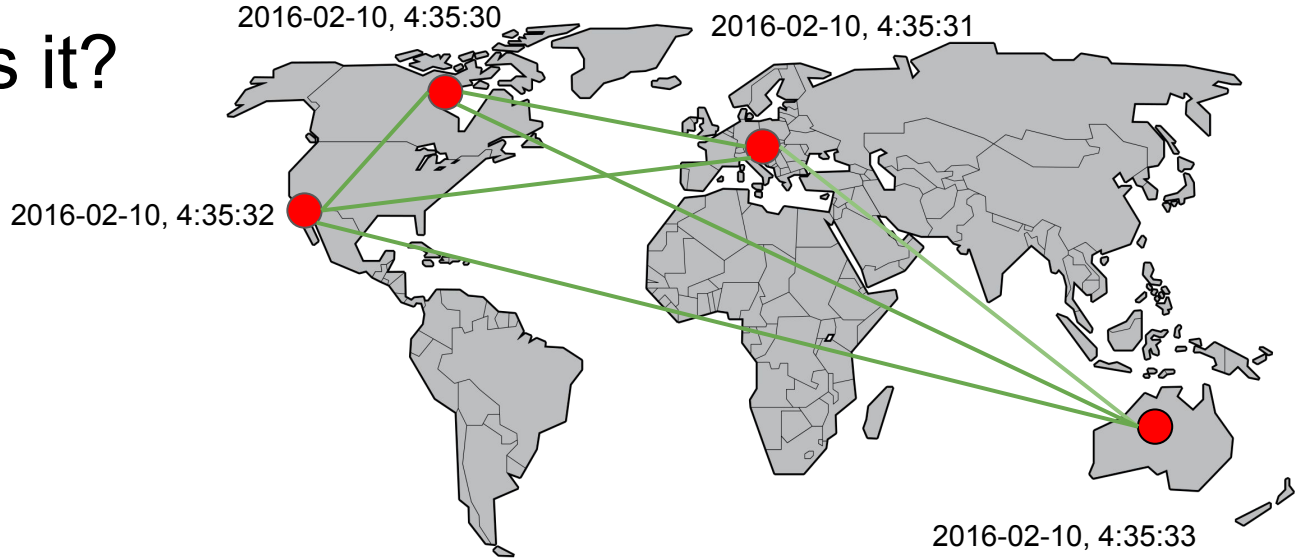
Figure 10: Distribution of Average Latencies

What are some drawbacks to this solution?

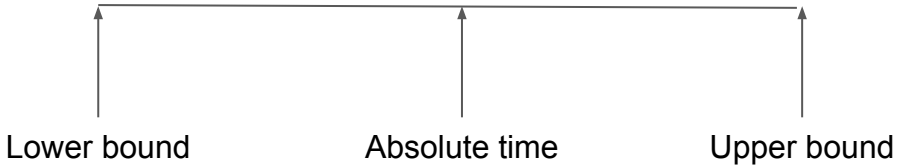

Highly Scalable
Rapid Development
Google Megastore
Low Latency
Consistent
Highly Available


Highly Scalable
Rapid Development
Google Spanner
Low Latency
Consistent
Highly Available

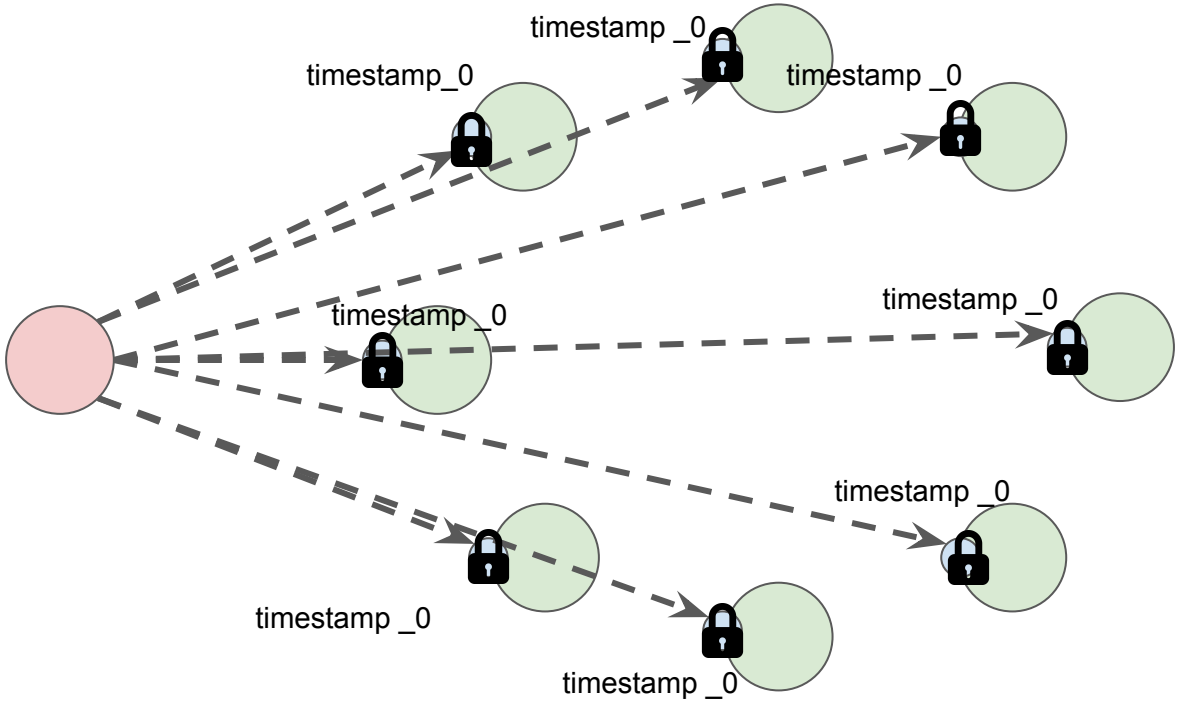
What time is it?



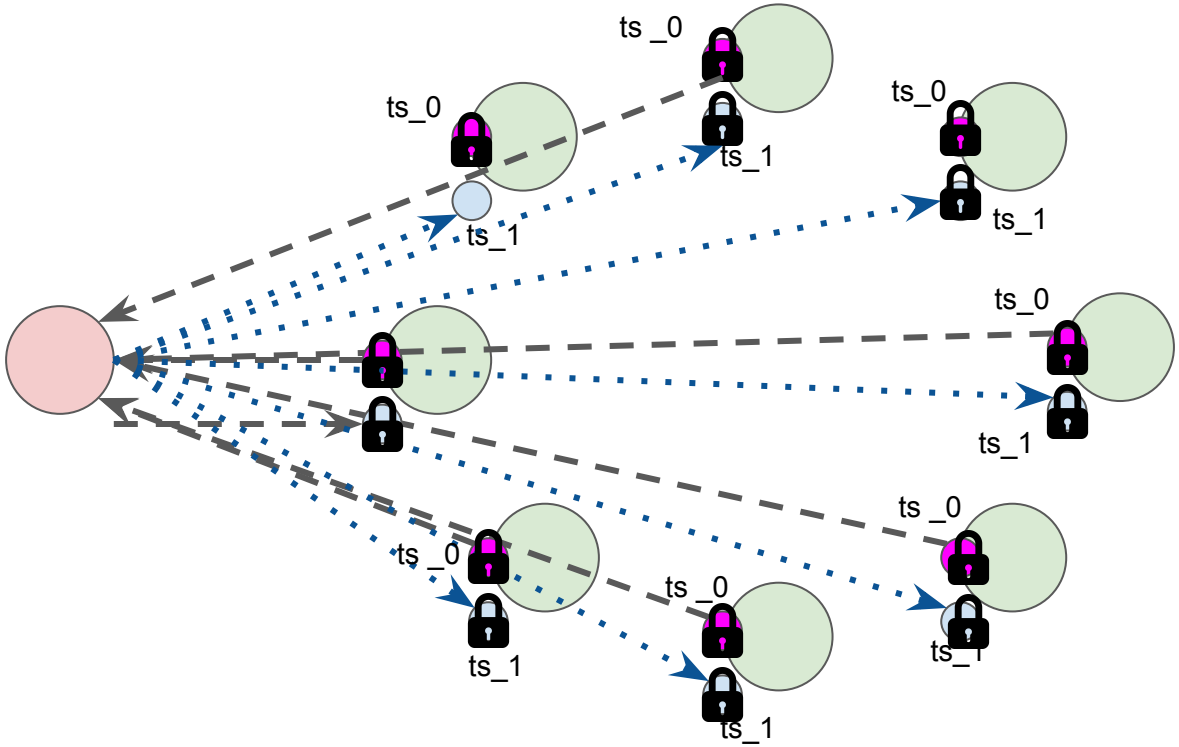
TT.now() → TTinterval:



Improved Paxos: Serialization with Locking



Improved Paxos: Long Leader Leases



Spanner Performance Analysis

Setup

replicas	latency (ms)			throughput (Kops/sec)		
	write	read-only transaction	snapshot read	write	read-only transaction	snapshot read
1D	9.4±.6	—	—	4.0±.3	—	—
1	14.4±1.0	1.4±.1	1.3±.1	4.1±.05	10.9±.4	13.5±.1
3	13.9±.6	1.3±.1	1.2±.1	2.2±.5	13.8±3.2	38.5±.3
5	14.4±.4	1.4±.05	1.3±.04	2.8±.3	25.3±5.2	50.0±1.1

Table 3: Operation microbenchmarks. Mean and standard deviation over 10 runs. 1D means one replica with commit wait disabled.

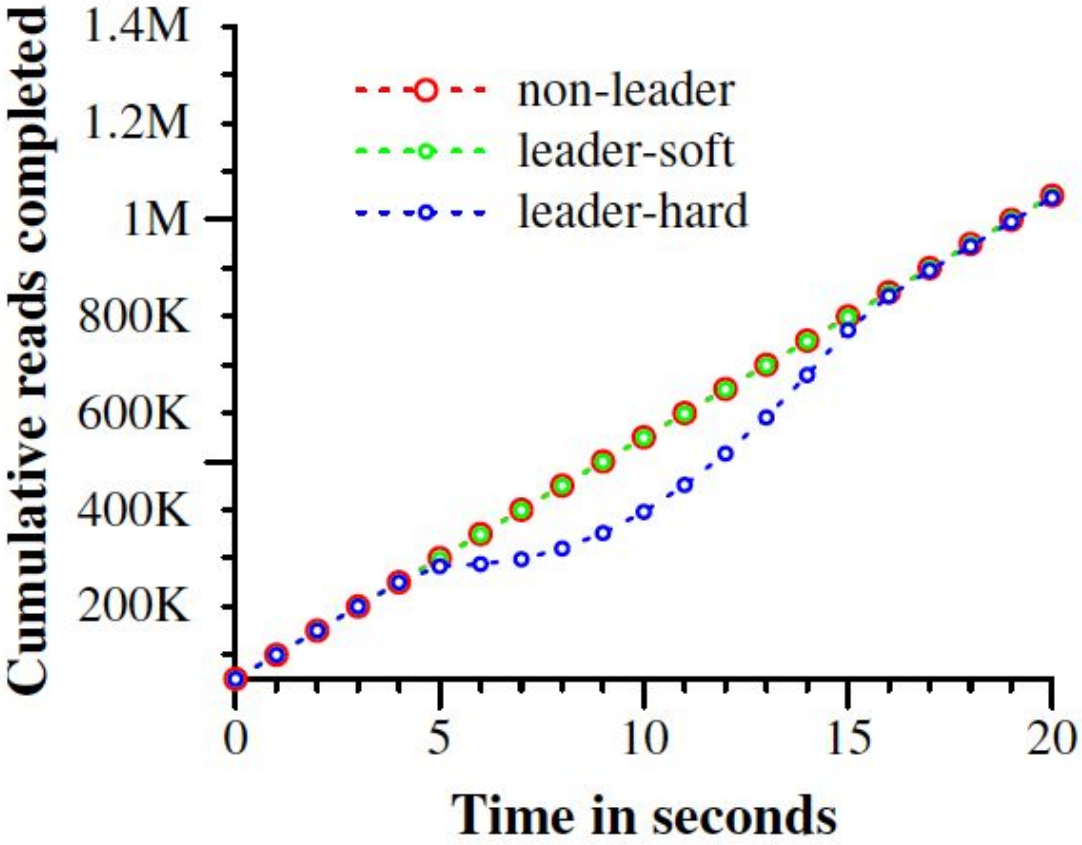
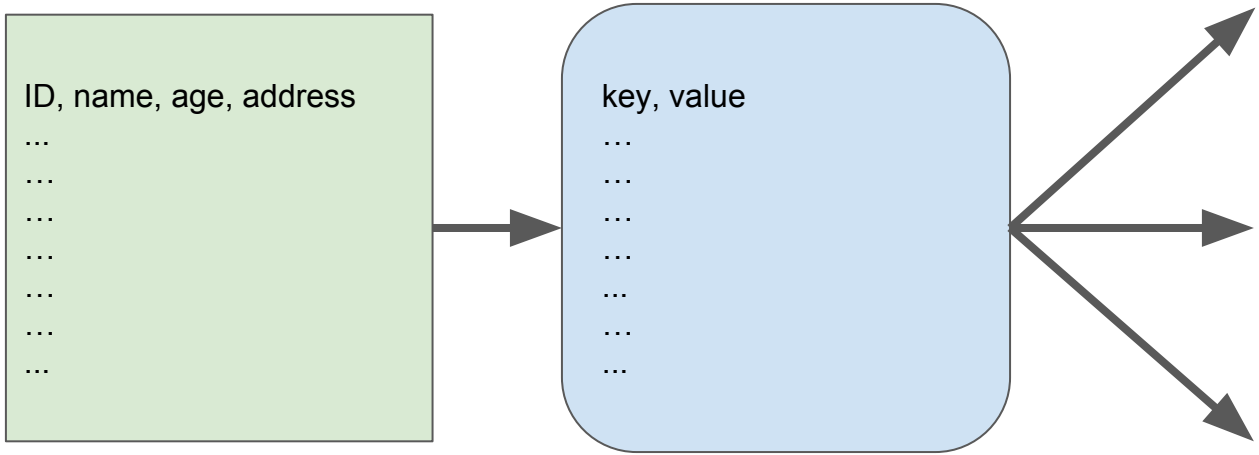


Figure 5: Effect of killing servers on throughput.

Next steps?

Supporting more complex SQL queries with an underlying key-value structure



The Future