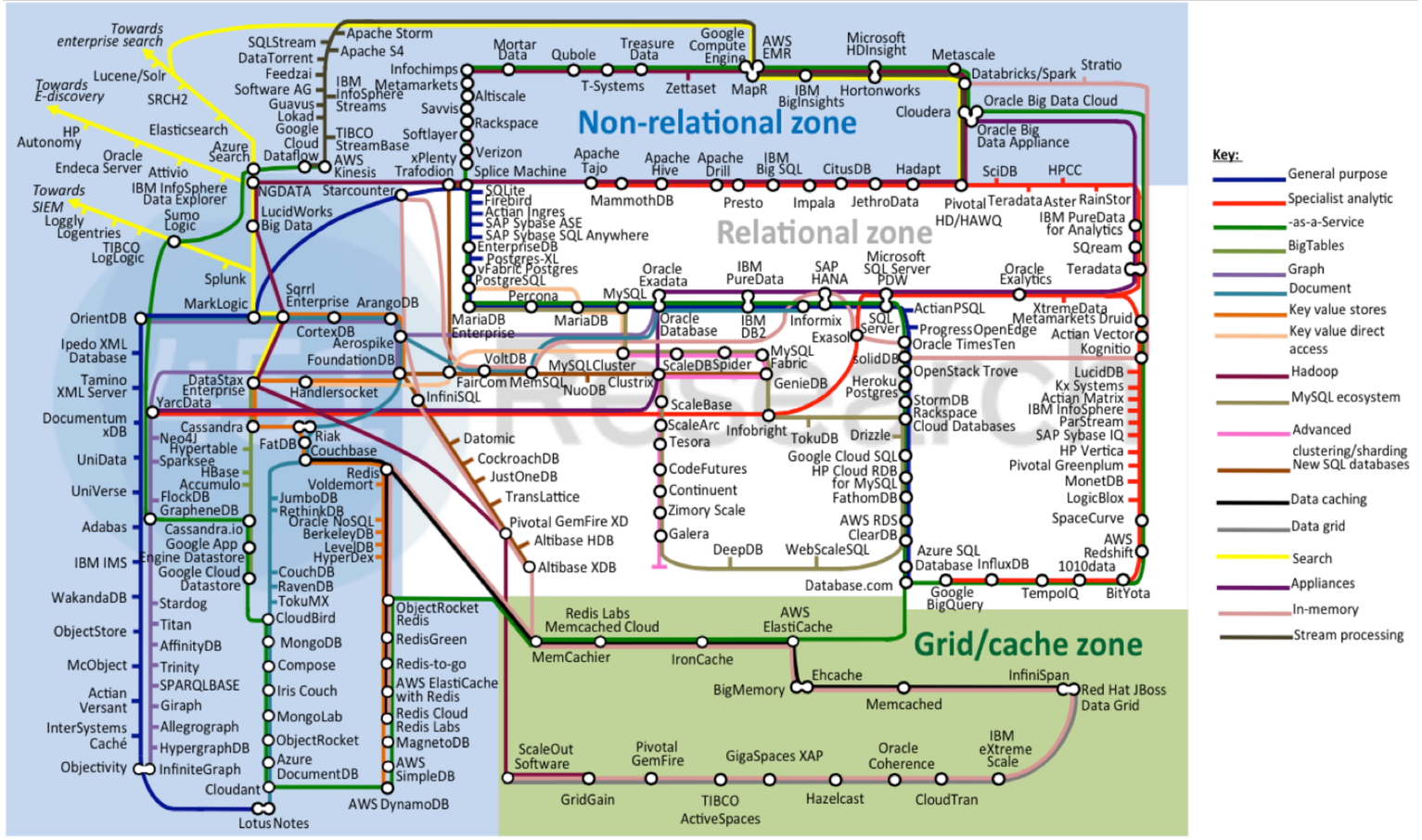
NoSQL and MongoDB:

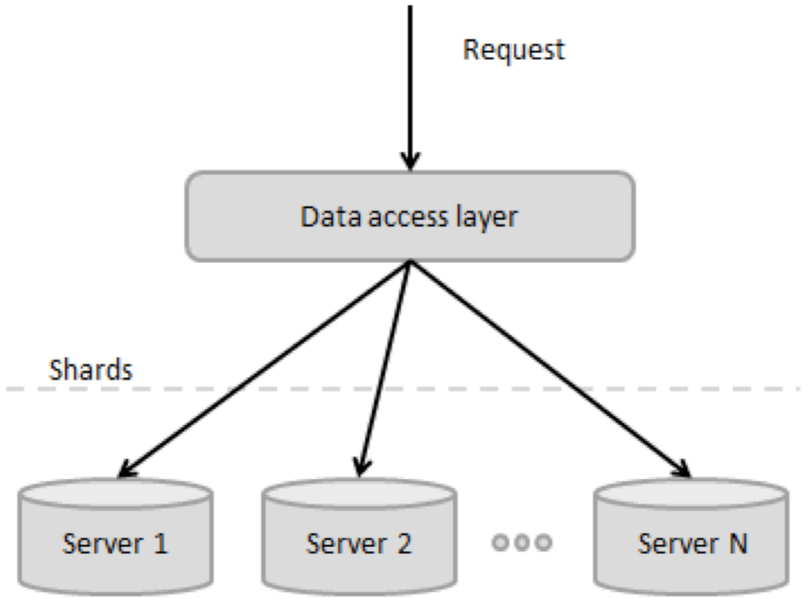
JADS Master: Data Engineering



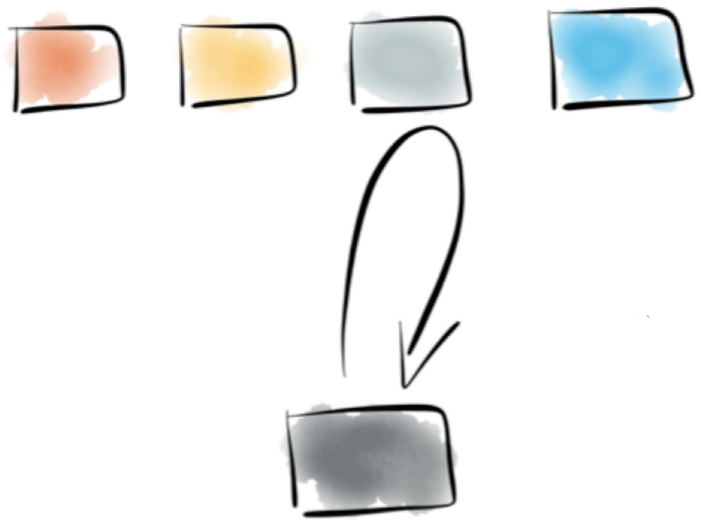
Part II: Partitioning, CAP, NewSQL

- Partitioning
- CAP
- NewSQL (if time permits)

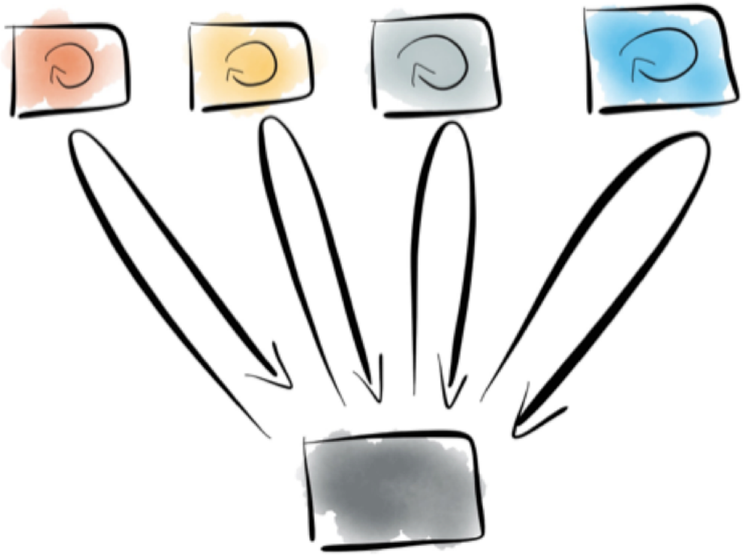
Scaling: often means partitioning



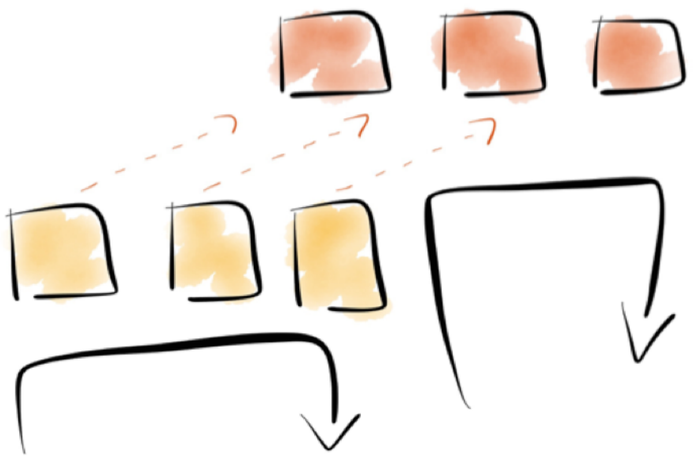
Partitioning



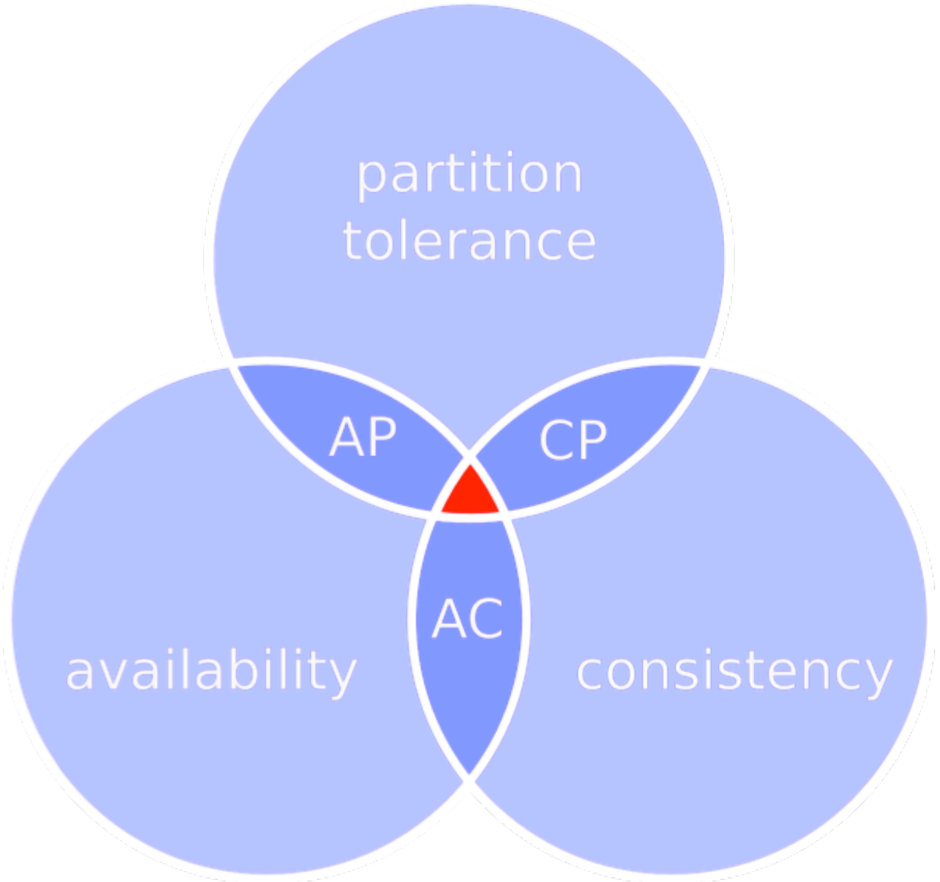
Partitioning



Partitioning



Partitioning and CAP

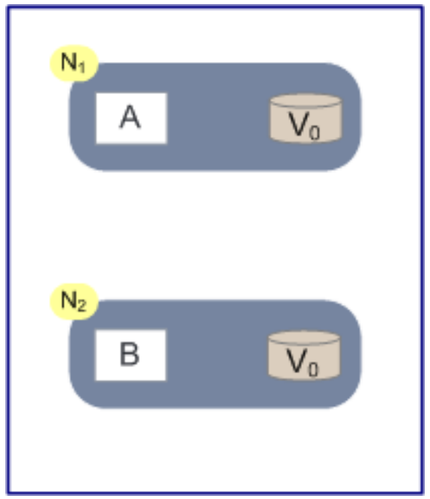


Consistency: All nodes see the same data at the same time

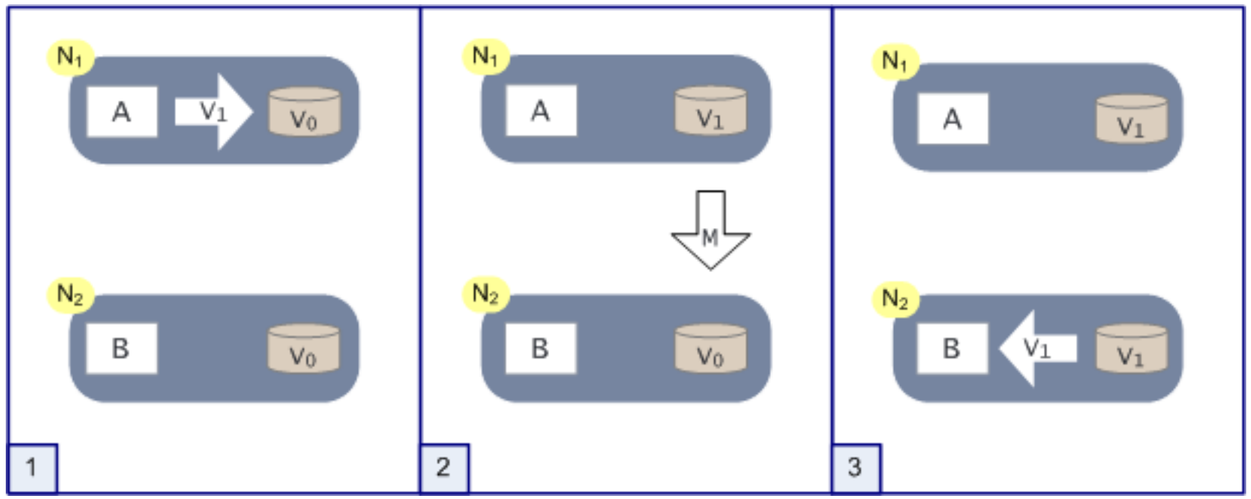
Availability: A guarantee that every request receives a response about whether it succeeded or failed

Partition tolerance: Ability to cope with a partitioned network of system nodes

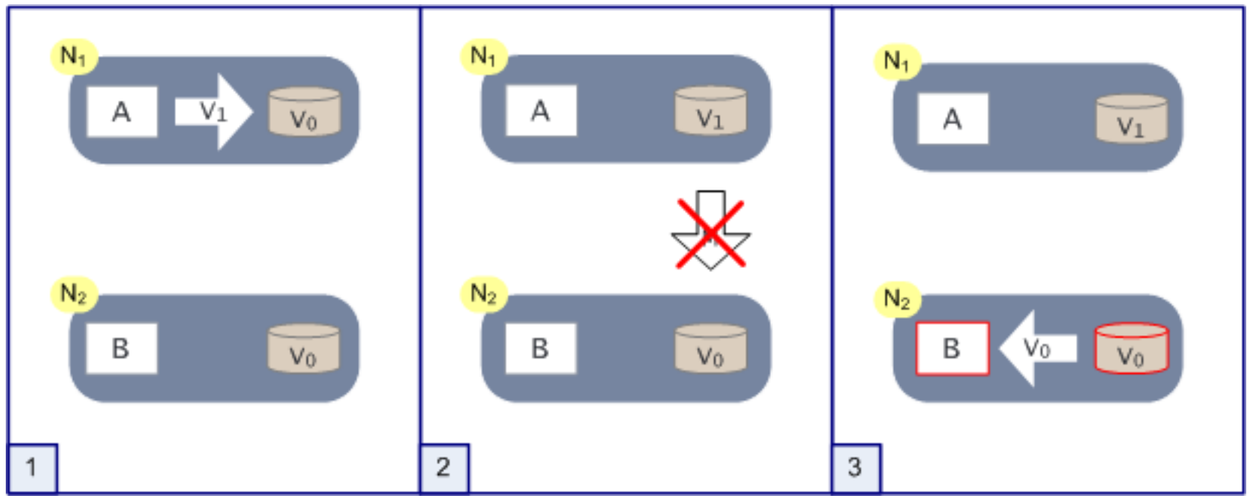
Partitioning and CAP



Partitioning and CAP

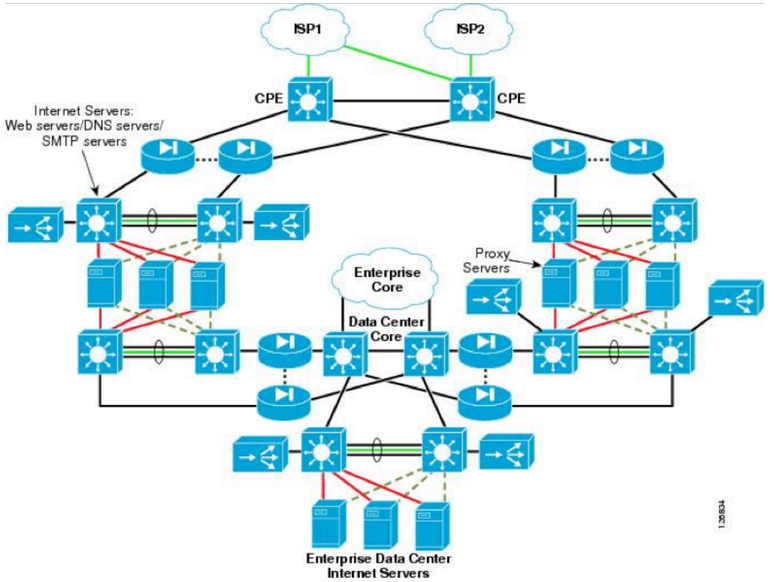


Partitioning and CAP



Partition tolerance

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous



CAP Misconceptions

1. CA?
2. No CA when P?
3. C is all or nothing
4. CAP is all about eventual consistency

PACELC

If Partioned:

Tradeoff Availability and Consistency

Else:

Tradeoff Latency and Consistency

Types of systems:

- PC/EC -> Most consistent
- PA/EL -> No consistency but low latency
- PA/EC -> Give up consistency when partitioned
- PC/EL -> Madness? See PNUTS

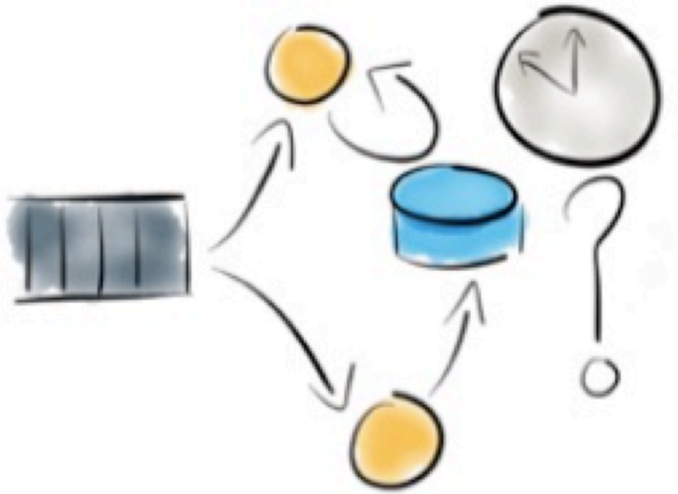
Nodes must exchange information on writes:

- Inform replicas
- Inform client

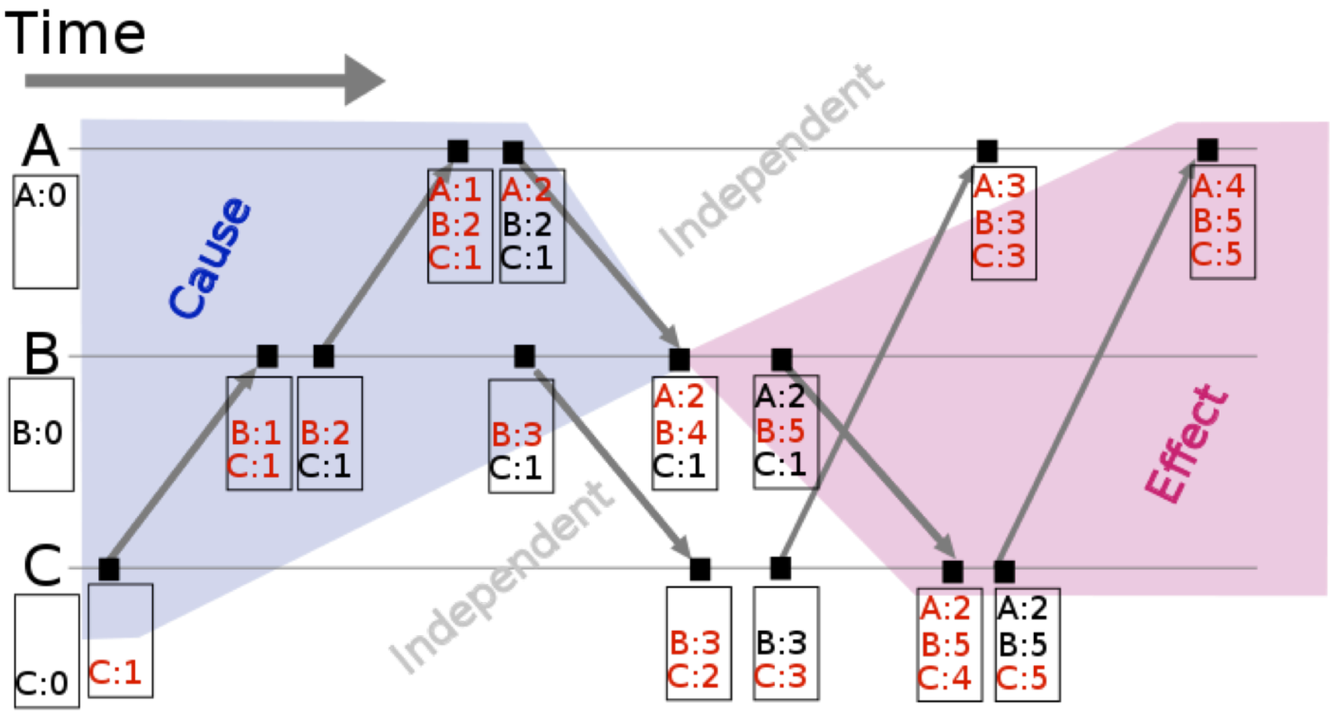
How to deal with conflicts?:

- Last write win?
- Vector clocks
- Multiversion storage
- Hardware clocks

Consistency: atomicity and ordering



Consistency: vector clocks



Consistency: tunable CAP

Riak:

N: number of total copies

R: minimal number of responding clients when reading

W: minimal number of responding clients when writing

Cassandra:

Tunable write consistency

Tunable read consistency

MongoDB:

Write result setting

Read result setting

Tunable CAP: Amazon shopping Basket

Amazon uses DynamoDB for shopping baskets. DynamoDB, like Riak is a distributed key-value store where N-R-W can be set for operation.

If you were Amazon, how would you choose N-R-W for a shopping basket?

Eventual consistency

Key property of non-ACID

- If no further changes
- All nodes will end up consistent

Weak guarantee

- When is eventually? It doesn’t say..
- In practice: expect inconsistency; always!

In practice:

- Stronger guarantees: predicting/measuring behavior
- Systems often appear strongly consistent

Eventual consistency

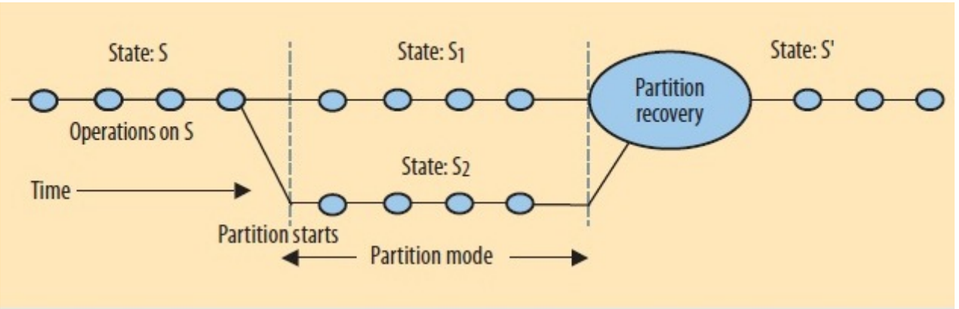


Figure 1. The state starts out consistent and remains so until a partition starts. To stay available, both sides enter partition mode and continue to execute operations, creating concurrent states S_1 and S_2, which are inconsistent. When the partition ends, the truth becomes clear and partition recovery starts. During recovery, the system merges S_1 and S_2 into a consistent state S′ and also compensates for any mistakes made during the partition.

Questions?

Meanwhile at

NewSQL

This backend was originally based on **a MySQL database that was manually sharded** many ways. The uncompressed dataset is tens of terabytes, which is small compared to many NoSQL instances, but was large enough to cause difficulties with sharded MySQL. The MySQL sharding scheme assigned each customer and all related data to a fixed shard. This layout enabled the use of indexes and complex query processing on a per-customer basis, but **required some knowledge of the sharding in application business logic**. Resharding this revenue-critical database as it grew in the number of customers and their data was extremely costly. **The last resharding took over two years of intense effort**, and involved coordination and testing across dozens of teams to minimize risk.

NewSQL

We store financial data and have **hard requirements on data integrity and consistency**. We also have a lot of experience with eventual consistency systems at Google. In all such systems, we find developers spend a significant fraction of their time **building extremely complex and error-prone mechanisms to cope with eventual consistency** and handle data that may be out of date. We think this is an unacceptable burden to place on developers and that consistency problems should be solved at the database level.

NewSQL

Some Requirements:

Scalable
- By adding hardware
- No manual sharding

Available
- No downtime, ever

Consistent
- Full ACID

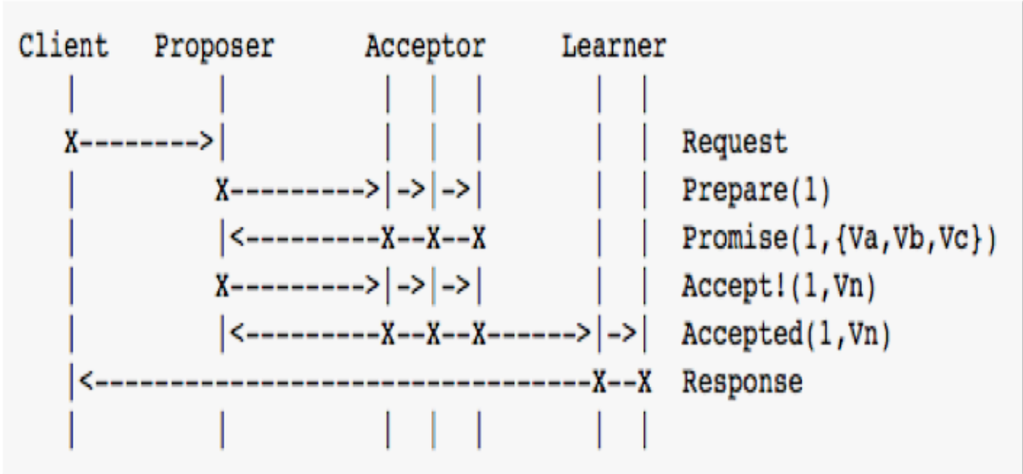
Usable
- Full SQL with indexes

At least 300 applications within Google use Megastore (despite its relatively low performance) because its **data model is simpler to manage** than Bigtable's, and because of its **support for synchronous replication** across datacenters. (Bigtable only supports eventually-consistent replication across datacenters.) Examples of well-known Google applications that use Megastore are **Gmail, Picasa, Calendar, Android Market, and AppEngine**.

NewSQL

Spanner:

- Semi-relational distributed DB
  - SQL queries
  - Versioned data
- Consistent reads/writes
- Atomic schema updates
- High availability
- Removing nodes has no effect except on throughput (PC/EC)

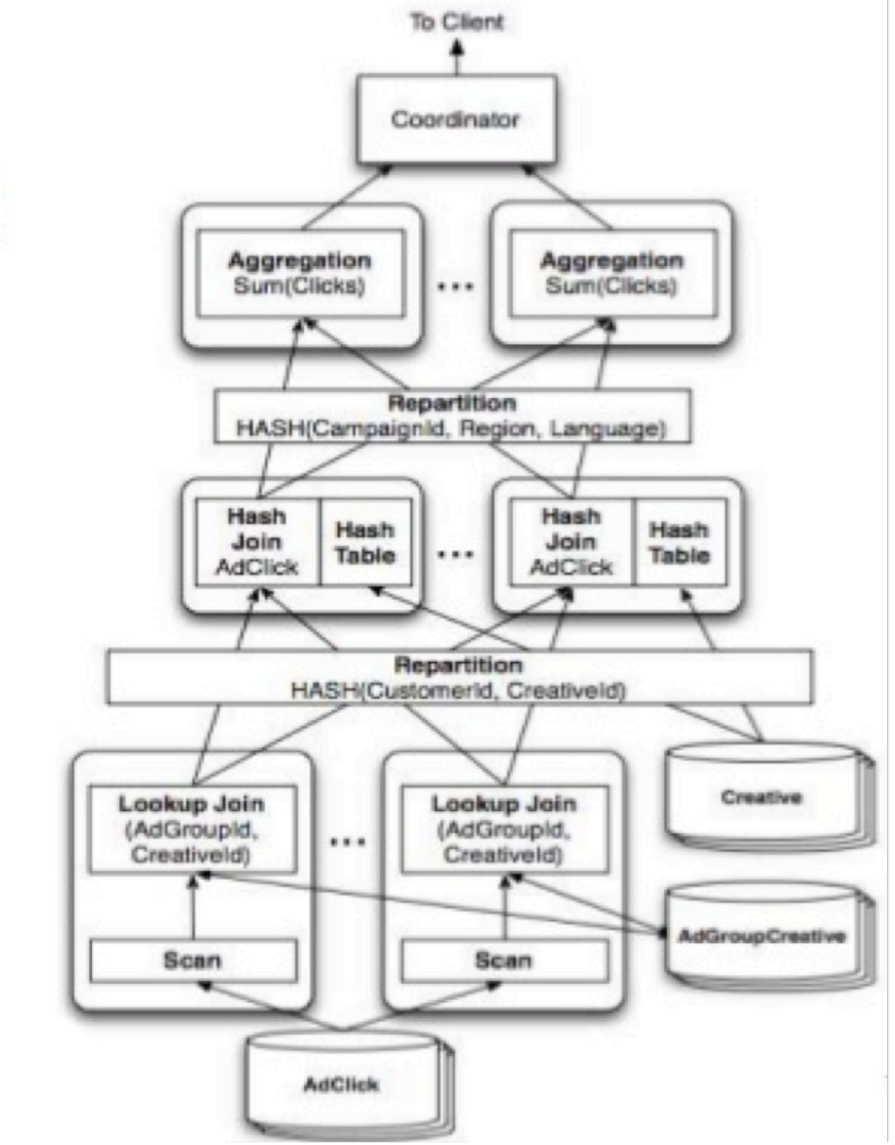


TrueTime:

- Timestamp: consistent ordering on transactions
- Uses: GPS and atomic clocks

F1:

- Distributed SQL queries
- Consistent indexes
- Automatic change history (triggers)



NewSQL

F1:

- 100 TB of uncompressed data
  - Over 5 data centers
  - Five nines uptime

— Hundreds of thousands of request/second

— SQL queries scan trillions of rows/day

— No observable increase of latency compared to MySQL

— NoSQL (key -> row) and full SQL

operation	latency (ms)		count
	mean	std dev	
all reads	8.7	376.4	21.5B
single-site commit	72.3	112.8	31.2M
multi-site commit	103.0	52.2	32.1M

Questions?