Introduction to Database Systems IDBS - Spring 2024

Lecture 11 - Scale & NoSQL

Scaling Up/Out
NoSQL
Eventual Consistency
CAP Theorem

Readings: PDBM 11

Omar Shahbaz Khan

General Info

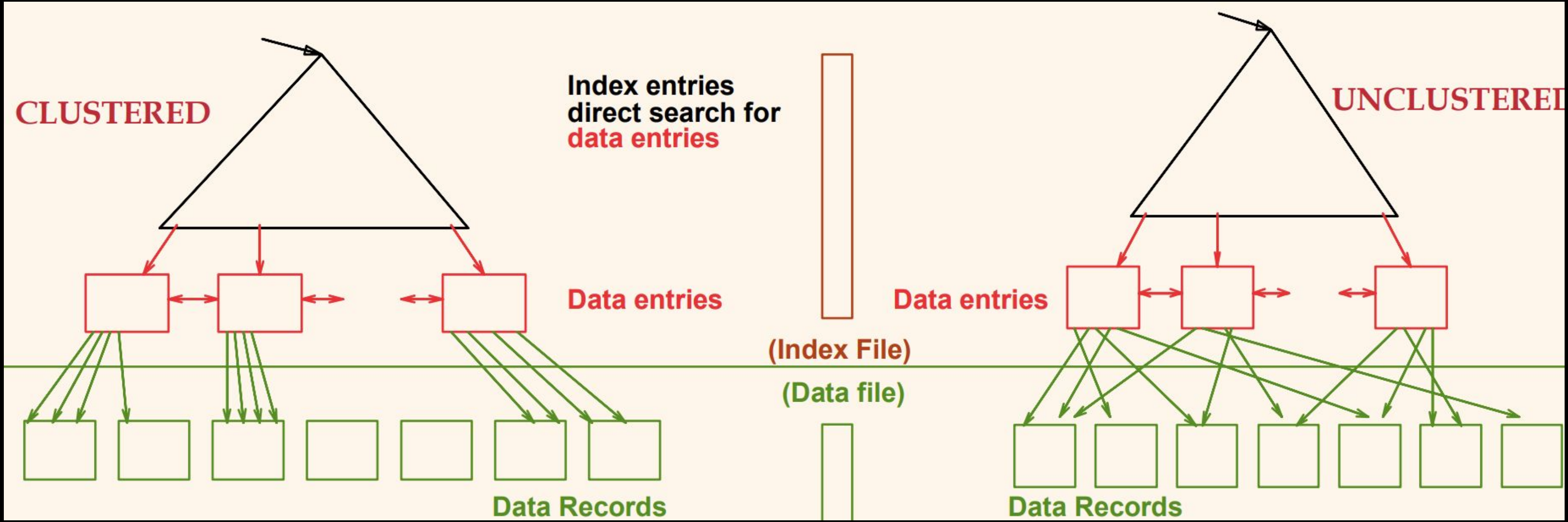
HOMEWORK 4 - OUT NOW!

- Deadline: May 7 2024 23:59
- Remember: 3/4 Homework Assignments need to be passed to be eligible for the exam!

FEEDBACK FOR HOMEWORK 3

- Will be out ASAP

Wake Up Task


CLUSTERED
Index entries
direct search for
data entries
UNCLUSTERED
Data entries
Data entries
(Index File)
(Data file)
Data Records
Data Records

Wake Up Task

- `OrderLines(ID, itemID, customerID, date, time, price, ...)`
- For each query, consider the choice of using a clustered index or a covering index.

Q1)
```
SELECT customerID, sum(price)
  FROM OrderLines
 GROUP BY customerID;
```

Q2)
```
SELECT max(date)
  FROM OrderLines
 WHERE customerID = 2200;
```

Q3)
```
SELECT *
  FROM OrderLines
 WHERE itemID = 1234;
```

Wake Up Task

- `OrderLines(ID, itemID, customerID, date, time, price, ...)`

- **Q1: Unclustered Index**
  - `OrderLines(customerID, price)`
- **Q2: Unclustered Index**
  - `OrderLines(customerID, date)`
- **Q3: Clustered Index**
  - `OrderLines(itemID)`

Q1)
```sql
SELECT customerID, sum(price)
  FROM OrderLines
 GROUP BY customerID;
```

Q2)
```sql
SELECT max(date)
  FROM OrderLines
 WHERE customerID = 2200;
```

Q3)
```sql
SELECT *
  FROM OrderLines
 WHERE itemID = 1234;
```

Last Time in IDBS…

-- TODO -> DONE

- ✓ DBMS History Lesson

- ✓ Transactions
  - ✓ ACID Properties
  - ✓ Buffer Management
  - ✓ Logging
  - ✓ Recovery
  - ✓ Locking

Transactions

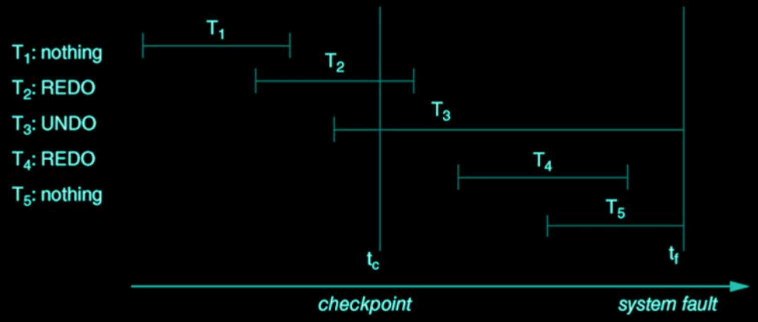
- Atomicity & Durability
  - Logging
  - Recovery
- Consistency
  - Constraints
  - Triggers
- Isolation
  - Locking
  - Isolation Levels

Transactions

- Atomicity & Durability
  - Logging
  - Recovery
- Consistency
  - Constraints
  - Triggers
- Isolation
  - Locking
  - Isolation Levels

- Write-Ahead-Logging (WAL)
  1. Before any changes are written to disk
     we force the corresponding log record to disk

     Is this necessary for MMDBs?
  2. Before a transaction is committed
     we force all log records for the transaction to disk

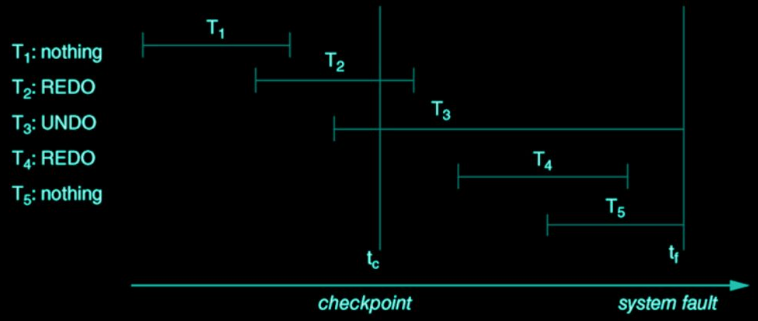


Transactions

- Atomicity & Durability
  - Logging
  - Recovery
- Consistency
  - Constraints
  - Triggers
- Isolation
  - Locking
  - Isolation Levels

- Recovery
  1. Analyze Log
  2. REDO Committed Transactions
  3. UNDO Uncommitted Transactions

Is this necessary for MMDBs?



Transactions

- Atomicity & Durability
  - Logging
  - Recovery
- Consistency
  - Constraints
  - Triggers
- Isolation
  - Locking
  - Isolation Levels

- Constraints
  - Primary Key, Foreign Key
  - NOT NULL, UNIQUE, CHECK[1]
  - IDENTITY[2], SERIAL
  - In DDL
- Triggers
  - Functions
  - Complex Requirements
  - Beyond DDL

1: https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-check-constraint/
2: https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-identity-column/

Transactions

- Atomicity & Durability
  - Logging
  - Recovery
- Consistency
  - Constraints
  - Triggers
- Isolation
  - Locking
  - Isolation Levels
- Serializability
- Two-Phase Locking (R2PL)
  - Acquire Locks DURING Transaction
  - Release Locks AFTER Transaction
- Deadlocks
  - Detected through wait-for-graphs
  - Resolved by picking a victim aka. aborting one of the transactions
- Phantoms

Transactions

- Atomicity & Durability
  - Logging
  - Recovery
- Consistency
  - Constraints
  - Triggers
- Isolation
  - Locking
  - Isolation Levels

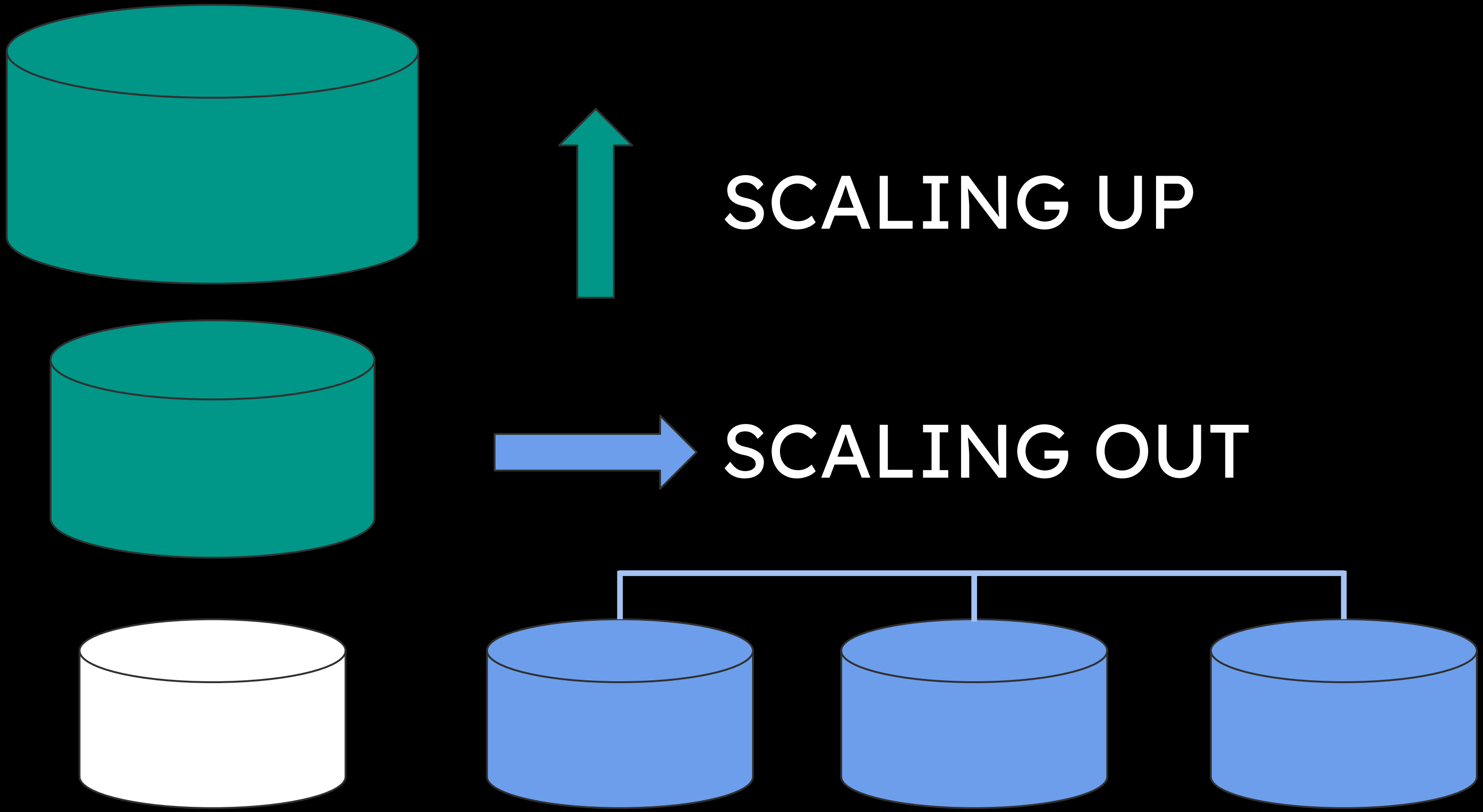
- Isolation Levels
  - Read Uncommitted (RU)
  - Read Committed (RC)
  - Repeatable Read (RR)
  - Serializable (S)
  - Snapshots (Si)

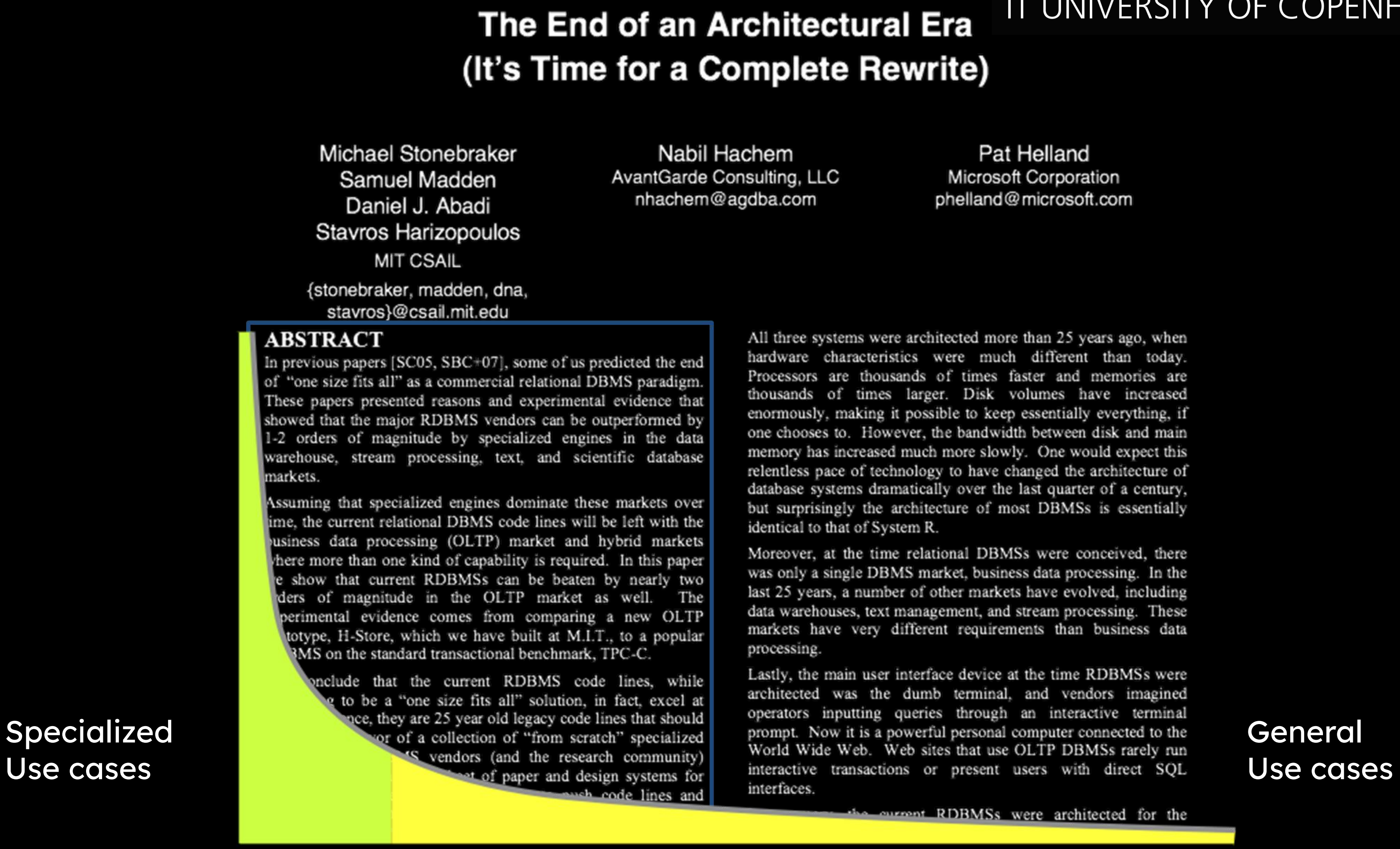
This Time…

-- TODO

- Scaling Up/Out

- NoSQL
  - Data Model
  - Architecture
    - Distributed Storage
  - Consistency/Availability trade-off
    - Eventual Consistency
    - CAP Theorem


SCALING UP
SCALING OUT



The End of an Architectural Era
(It's Time for a Complete Rewrite)

Michael Stonebraker
Samuel Madden
Daniel J. Abadi
Stavros Harizopoulos
MIT CSAIL
{stonebraker, madden, dna, stavros}@csail.mit.edu

Nabil Hachem
AvantGarde Consulting, LLC
nhachem@agdba.com

Pat Helland
Microsoft Corporation
phelland@microsoft.com

ABSTRACT

In previous papers [SC05, SBC+07], some of us predicted the end of "one size fits all" as a commercial relational DBMS paradigm. These papers presented reasons and experimental evidence that showed that the major RDBMS vendors can be outperformed by 1-2 orders of magnitude by specialized engines in the data warehouse, stream processing, text, and scientific database markets.

Assuming that specialized engines dominate these markets over ime, the current relational DBMS code lines will be left with the usiness data processing (OLTP) market and hybrid markets here more than one kind of capability is required. In this paper e show that current RDBMSs can be beaten by nearly two ders of magnitude in the OLTP market as well. The perimental evidence comes from comparing a new OLTP totype, H-Store, which we have built at M.I.T., to a popular MS on the standard transactional benchmark, TPC-C.

nclude that the current RDBMS code lines, while g to be a "one size fits all" solution, in fact, excel at nce, they are 25 year old legacy code lines that should or of a collection of "from scratch" specialized S vendors (and the research community) of paper and design systems for code lines and

All three systems were architected more than 25 years ago, when hardware characteristics were much different than today. Processors are thousands of times faster and memories are thousands of times larger. Disk volumes have increased enormously, making it possible to keep essentially everything, if one chooses to. However, the bandwidth between disk and main memory has increased much more slowly. One would expect this relentless pace of technology to have changed the architecture of database systems dramatically over the last quarter of a century, but surprisingly the architecture of most DBMSs is essentially identical to that of System R.

Moreover, at the time relational DBMSs were conceived, there was only a single DBMS market, business data processing. In the last 25 years, a number of other markets have evolved, including data warehouses, text management, and stream processing. These markets have very different requirements than business data processing.

Lastly, the main user interface device at the time RDBMSs were architected was the dumb terminal, and vendors imagined operators inputting queries through an interactive terminal prompt. Now it is a powerful personal computer connected to the World Wide Web. Web sites that use OLTP DBMSs rarely run interactive transactions or present users with direct SQL interfaces.

the current RDBMSs were architected for the

Major Trends

- More hardware
  - Better servers
  - More servers ➔ Cloud

- More data
  - More quantity of data
  - More types of data
  - Still want fast systems
  - Less structure ➔ less need for complexity

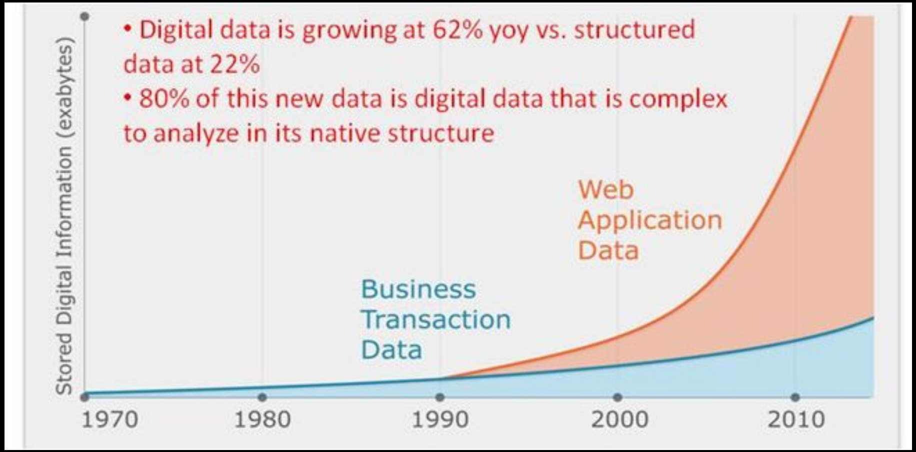


Image source: https://practicalanalytics.wordpress.com/2011/11/06/explaining-hadoop-to-management-whats-the-big-data-deal/

Scaling UP

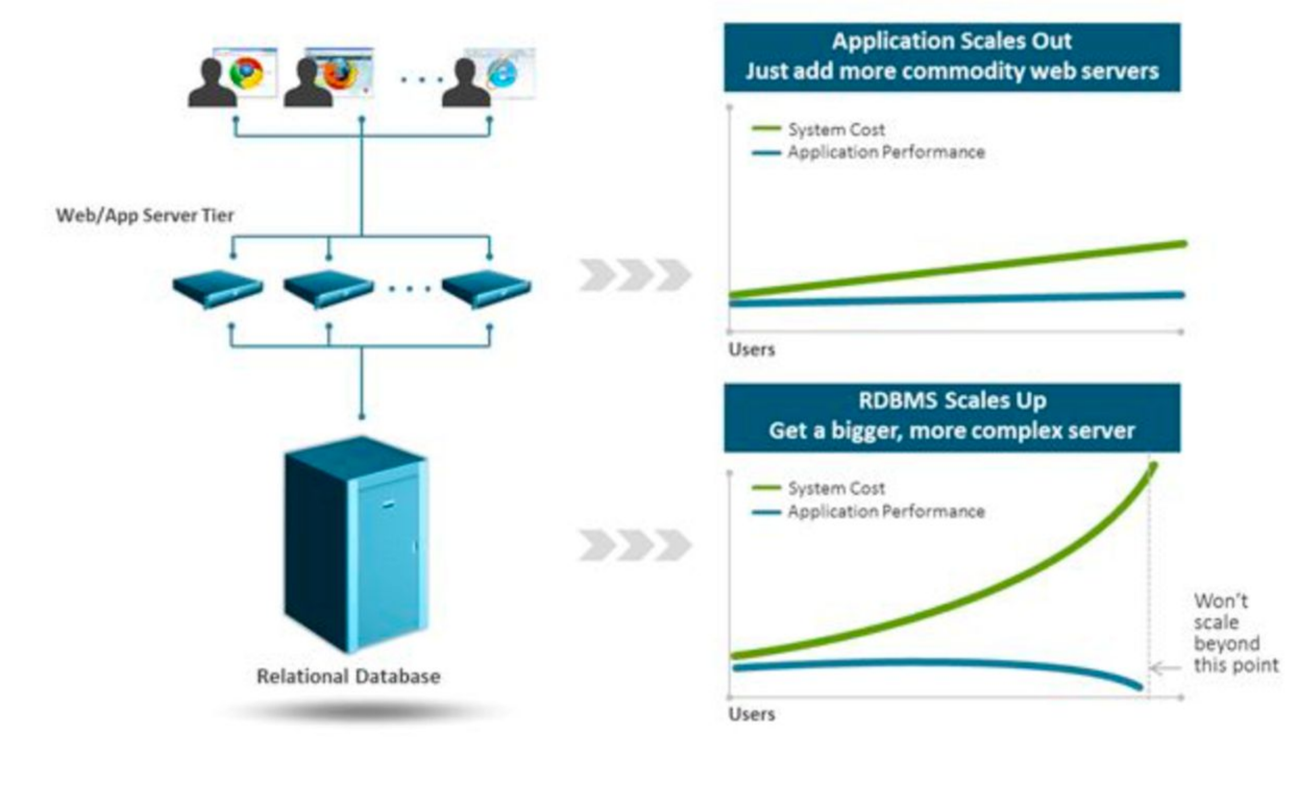


Image source: https://sungsoo.github.io/2014/07/08/nosql-background.html

Scaling OUT

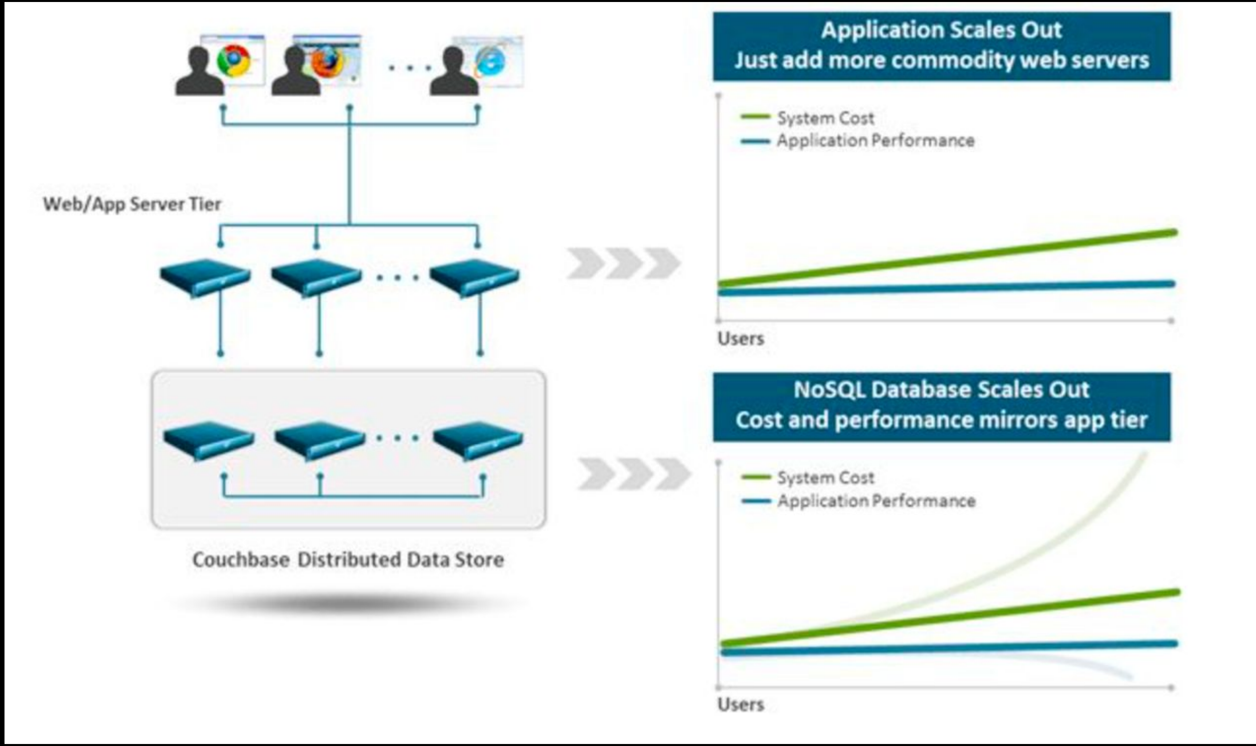
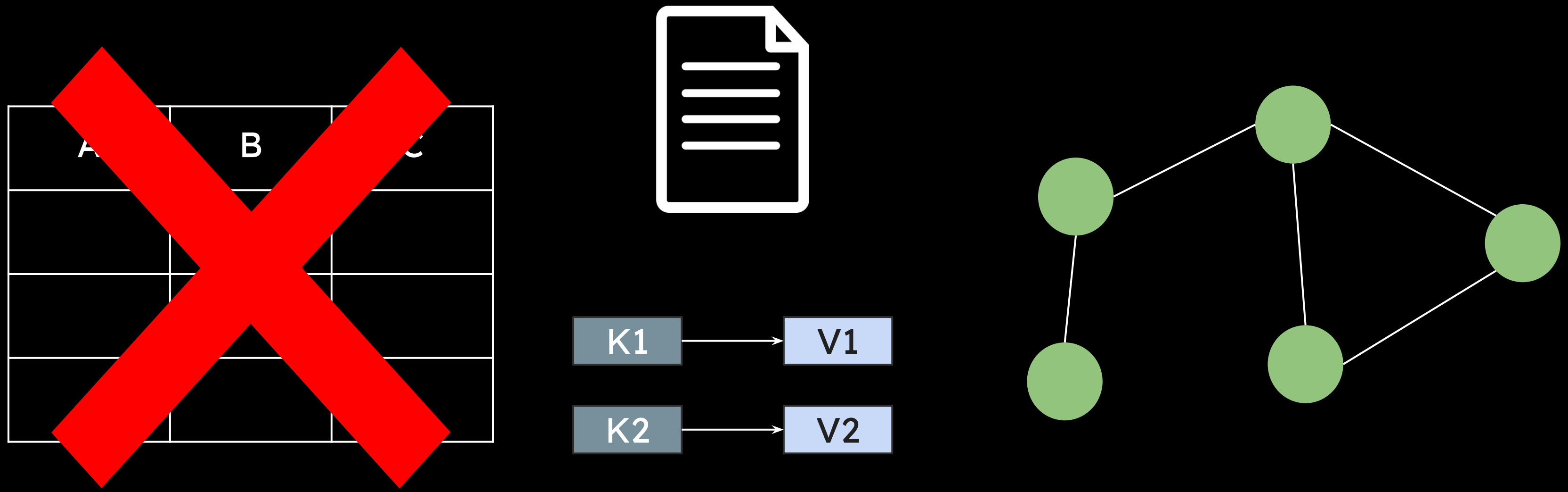


Image source: https://sungsoo.github.io/2014/07/08/nosql-background.html

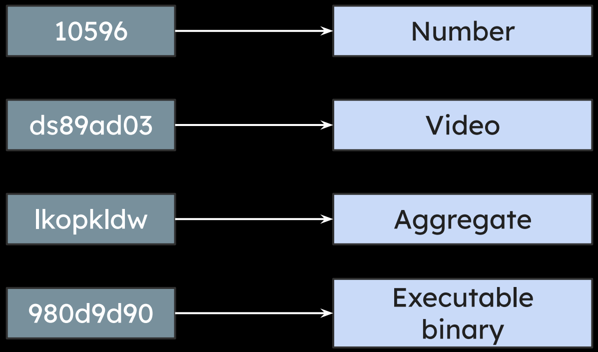

B
K1
V1
K2
V2
K3
V3

NoSQL in a Nutshell

- Data model
  - Not relational
  - No formally described schema
- Interface
  - Not only SQL (→ NoSQL name)
  - Proprietary, REST, CQL etc.
- Architecture
  - Usually distributed
- Mostly not ACID compliant
  - Consistency/Availability trade-off (CAP theorem)
- Mostly open source

Key-Value Stores

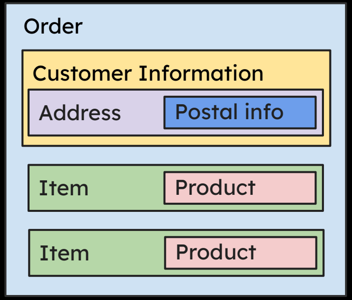
- Associative Array
  - Unique key points to a value
  - Value contents unknown
- Can not be queried
  - GET / PUT only
  - Value can be an aggregate structure
- Examples
  - Riak, Voldemort, MemcacheDB, Redis

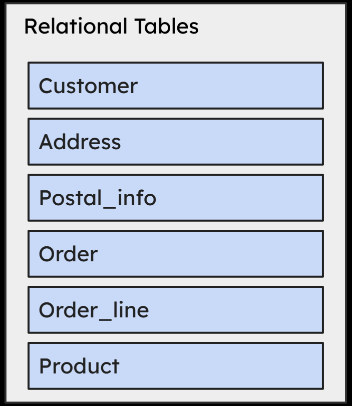


Document Stores

- Each value is a document
  - Most often JSON
  - Unique keys used for retrieval
- You can query into the document
  - More transparent than key-value stores
- The document is an aggregate structure
- Examples – MongoDB, Lotus Notes, CouchDB

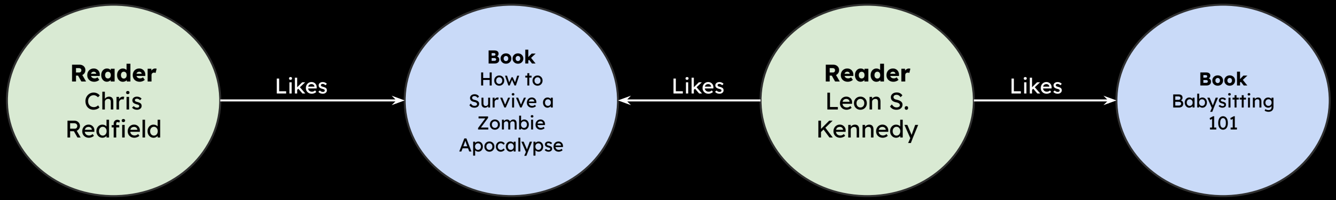
```
{
    "order_id": 3294,
    "customer": {
        "ssn": "123456789",
        "name": "John Doe",
        "address": "123 Random Street",
        "postal_code": "98765"
    },
    "line-items": [
        {"product": "Sofa", "price": 2500},
        {"product": "Table", "price": 1000},
    ]
}
```





Graph Stores

- Nodes = Entities
- Edges = Relationships, directional
- Properties = Entity descriptors
- Examples – neo4j, Allegro, InfiniGraph, OrientDB



-- TODO

✓ Scaling Up/Out

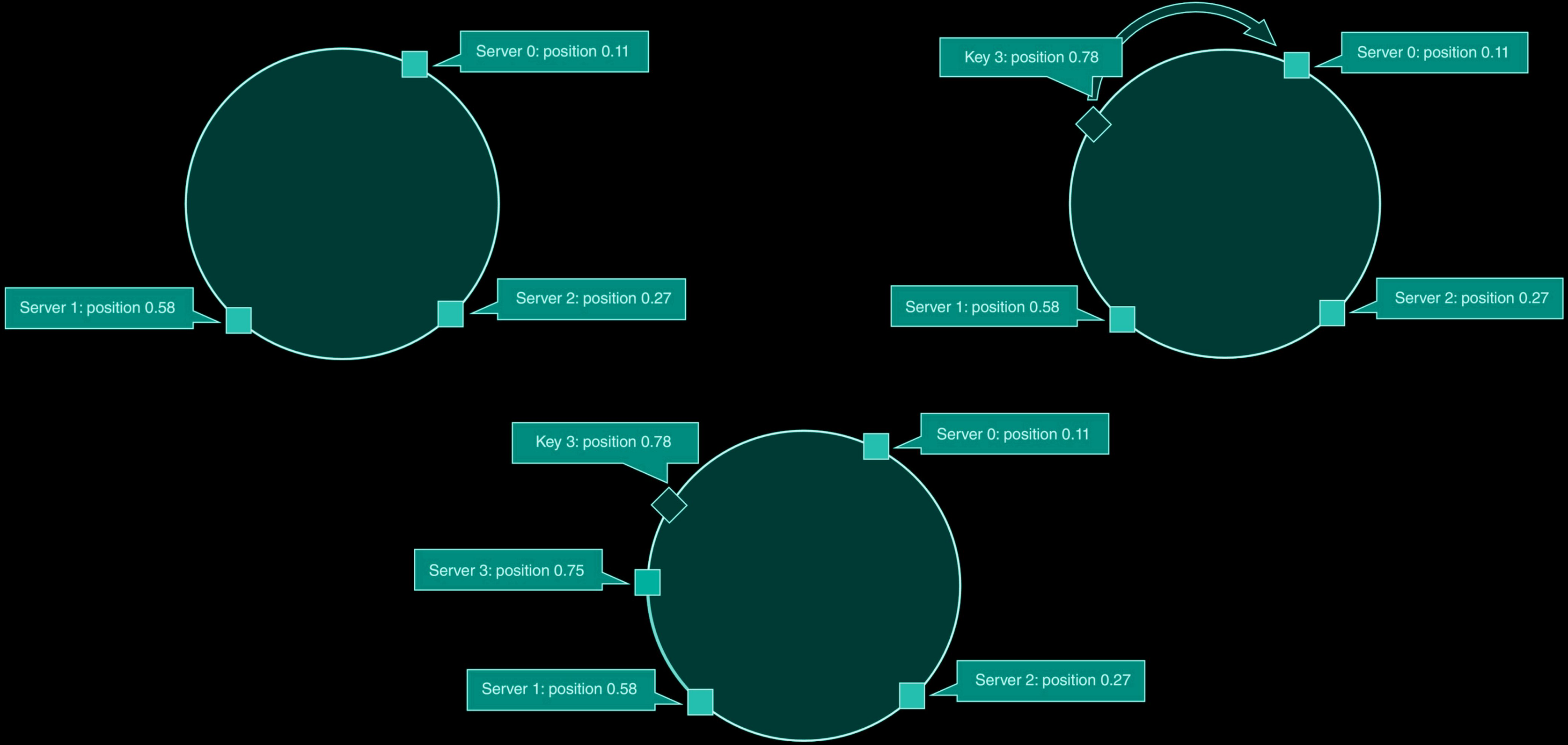
- NoSQL
  - ✓ Data Model
  - Architecture
    - Distributed
  - Consistency/Availability trade-off
    - Eventual Consistency
    - CAP Theorem

Goals of a Distributed Storage

- Workload sharing – balance
- Redundancy (replicas) – failure handling

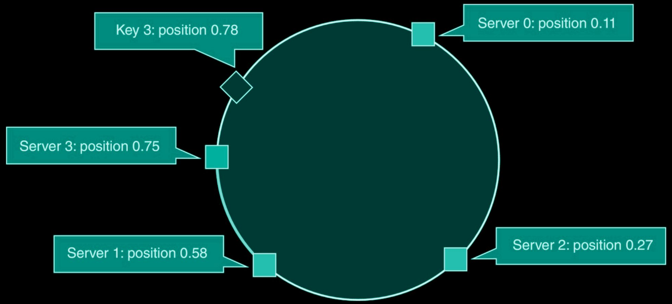
What does a "cloud" mean in this context?

Consistent Hashing: Balance



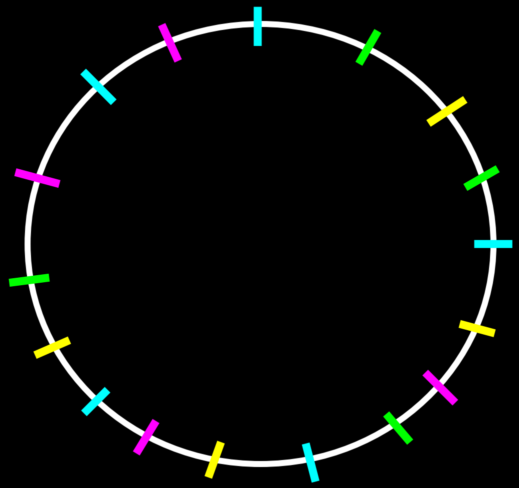
Consistent Hashing: Problems

- Random assignments can lead to skewed distributions
- Does not reflect a server's capabilities
  - Not all servers have the same hardware
- Adding new server only takes high load from 1 server



Consistent Hashing: Virtual Servers

- 1 Server = multiple smaller (virtual) servers
- Random assignments of virtual servers



Server 0 = 5

Server 1 = 4

Server 2 = 4

Server 3 = 4

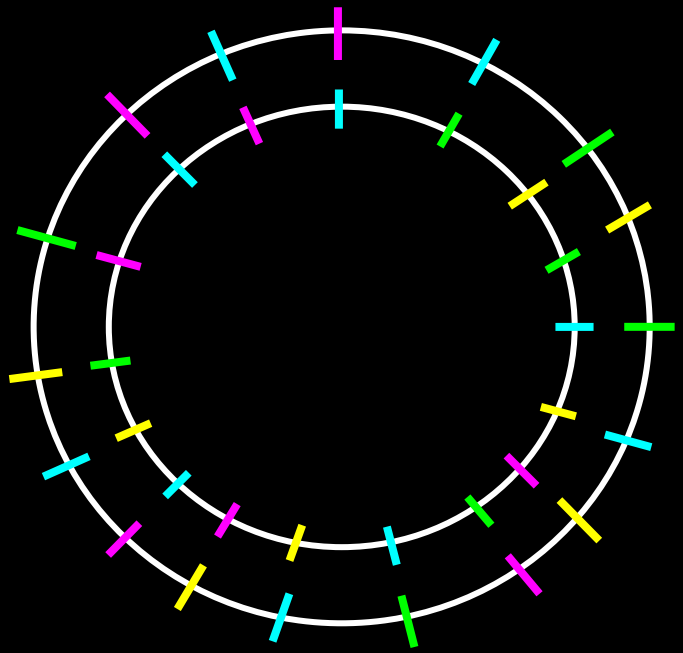
Consistent Hashing: Redundancy

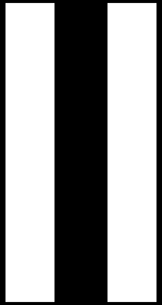
Server 0

Server 1

Server 2

Server 3





-- TODO

✓ Scaling Up/Out

- NoSQL
  - ✓ Data Model
  - ✓ Architecture
    - ✓ Distributed
  - Consistency/Availability trade-off
    - Eventual Consistency
    - CAP Theorem

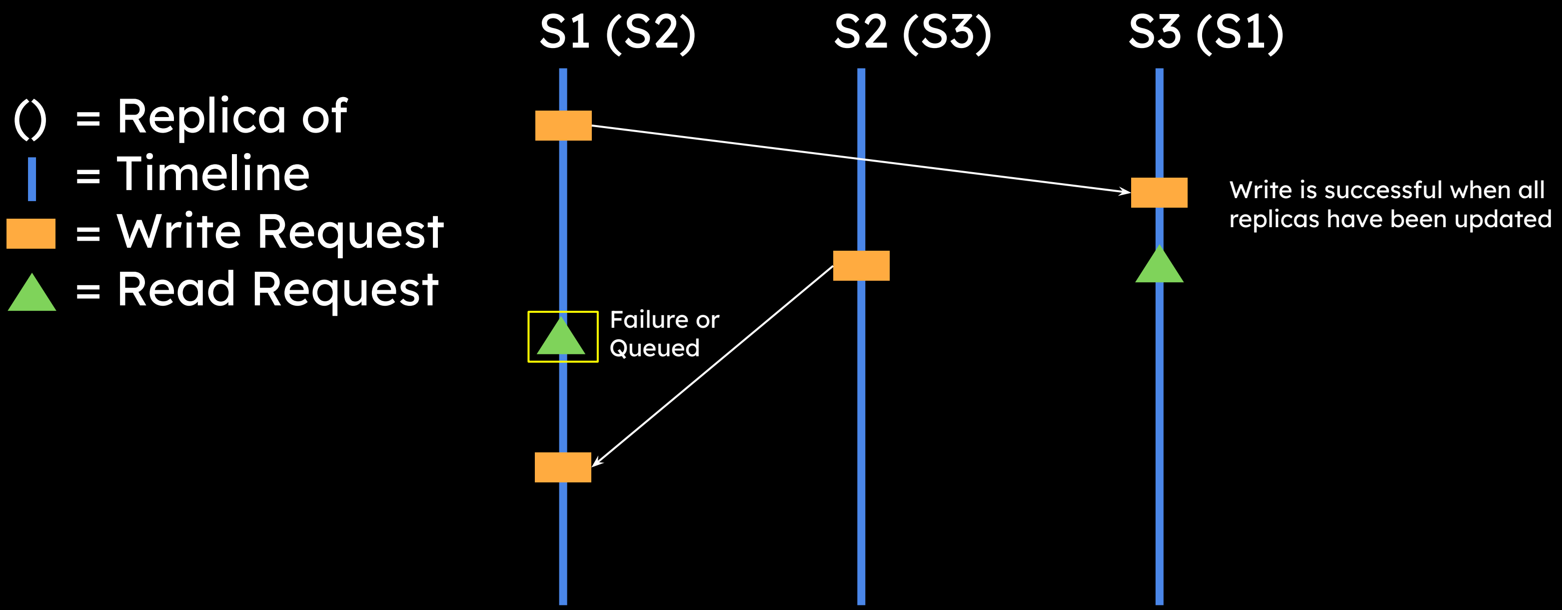
Replica Consistency

- Sequential (or strong) consistency: All updates are seen by all processes in the same order. As a result, the effects of an update are seen by all observers. There is no inconsistency.
  - Roughly the same as Isolation in ACID

- Weak consistency: Observers might see inconsistencies among replicas

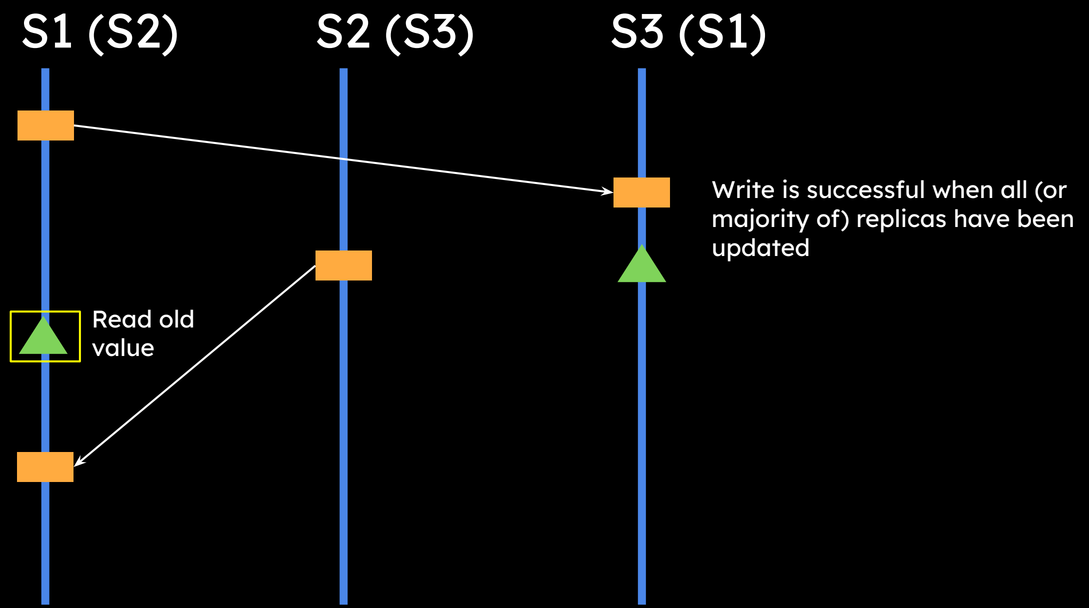
- Eventual consistency: A form of weak consistency, where at some point, in case there is no failure, all replicas will reflect the last update.

Replica Consistency: Strong Consistency



Replica Consistency: Eventual Consistency

() = Replica of
| = Timeline
■ = Write Request
▲ = Read Request



Tunable Consistency

- Not a binary decision
  - N replicas, R reads, W writes

- R = W = 1 gives eventual consistency

- R + W > N gives strong consistency

Example: if the replication factor is 3, then the consistency level of the reads and writes combined must be at least 4. Read operations using 2 out of 3 replicas to verify the value and write operations using 2 out of 3 replicas to verify the value will result in strong consistency.

CAP Theorem

- C = Consistency
  - Readers read the most recent update

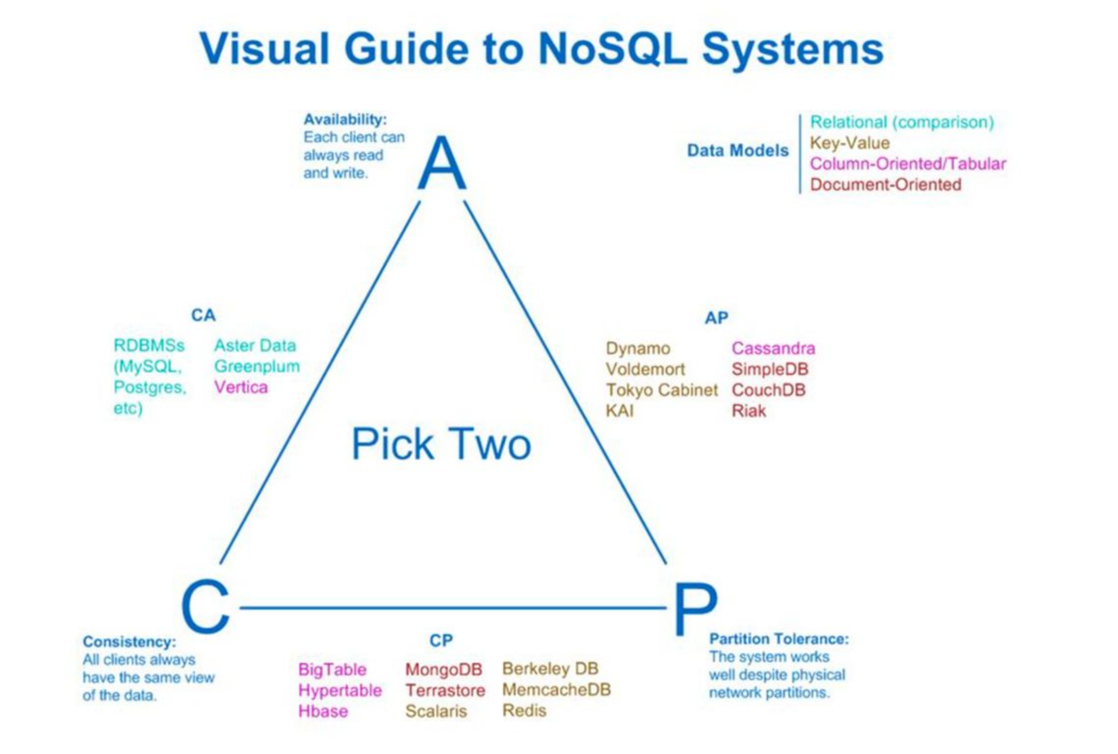
- A = Availability
  - A valid answer is returned, even if one or more nodes are down

- P = Partition Tolerance
  - Partition: The network becomes disconnected
  - Distributed system works despite the network failure

Incorrect (but Typical) Formulation

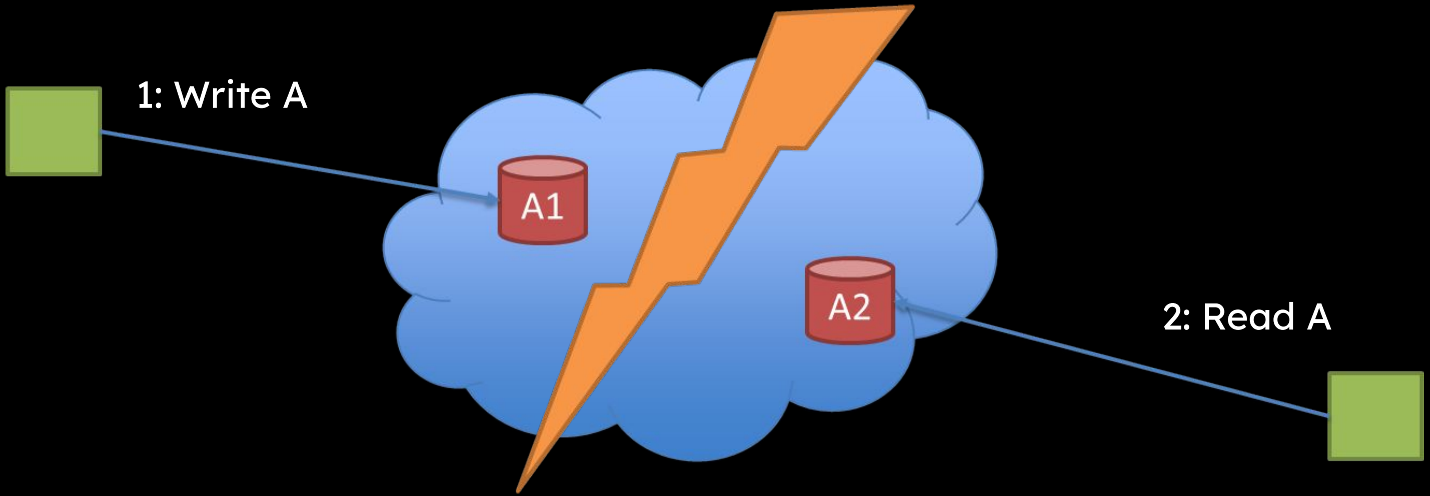
- You can only get **two** of
  Consistency,
  Availability, and
  Partition Tolerance

Resulting Classification



Correct (but Useless) Formulation

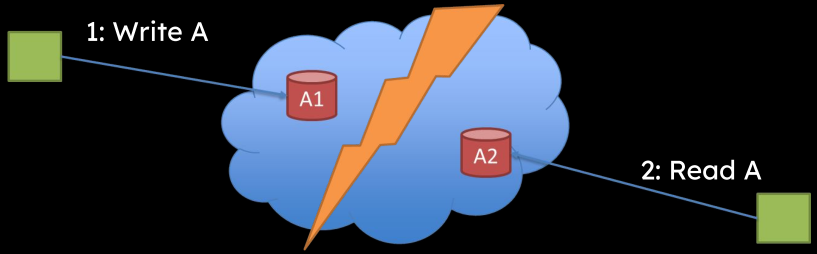
- In a **partitioned** network, choose between **Consistency** and **Availability**
- Proof: Simple thought experiment



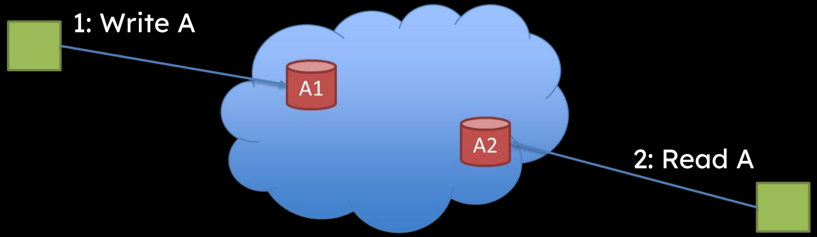
Seth Gilbert and Nancy Lynch, “Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services”, ACM SIGACT News, Volume 33 Issue 2 (2002)

PACELC (useful!) Formulation

- In a **partitioned** network, choose between **Availability** and **Consistency**



- **Else** (regular operation), choose between **Latency** and **Consistency**



Daniel Abadi, “Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story”, IEEE Computer 45(2): 37-42 (2012)

2012: Google Spanner

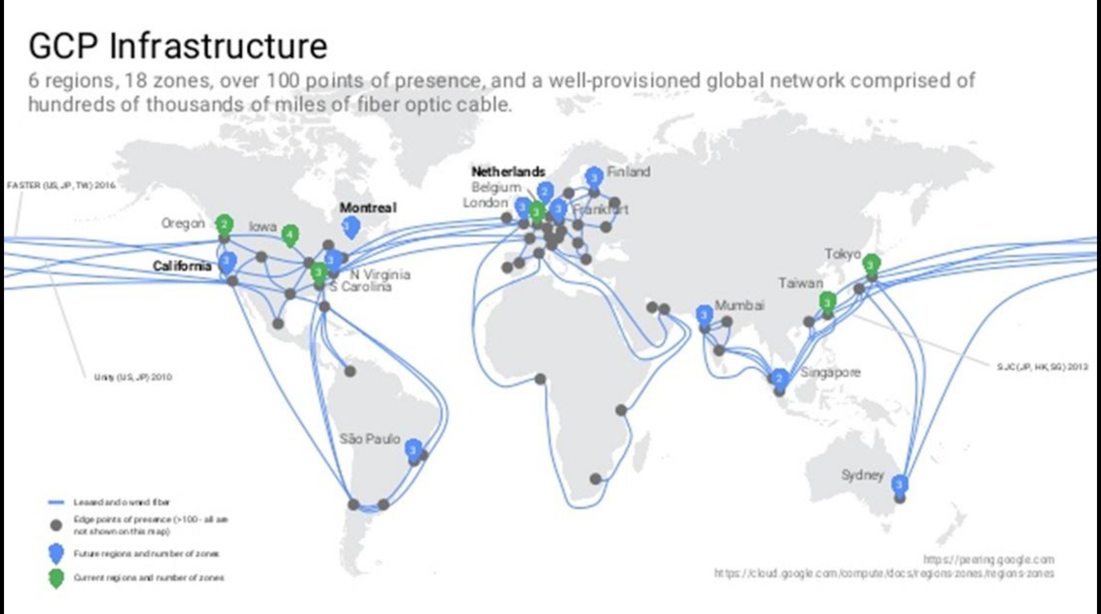
Cloud Spanner: The best of the relational and non-relational worlds

	CLOUD SPANNER	TRADITIONAL RELATIONAL	TRADITIONAL NON-RELATIONAL
Schema	✓ **Yes**	✓ Yes	✗ No
SQL	✓ **Yes**	✓ Yes	✗ No
Consistency	✓ **Strong**	✓ Strong	✗ Eventual
Availability	✓ **High**	✗ Failover	✓ High
Scalability	✓ **Horizontal**	✗ Vertical	✓ Horizontal
Replication	✓ **Automatic**	Configurable	Configurable

- Does it break the CAP theorem?

https://cloud.google.com/spanner/

What is High Availability?



- “Cloud Spanner […] serves data with low latency while maintaining transactional consistency and industry-leading 99.999% (five 9s) availability - 10x less downtime than four nines (<5 minutes per year).”

https://cloud.google.com/spanner/

Image source: https://www.slideshare.net/MichelleHolley1/google-cloud-networking-deep-dive

2012: Google Spanner

Cloud Spanner: The best of the relational and non-relational worlds

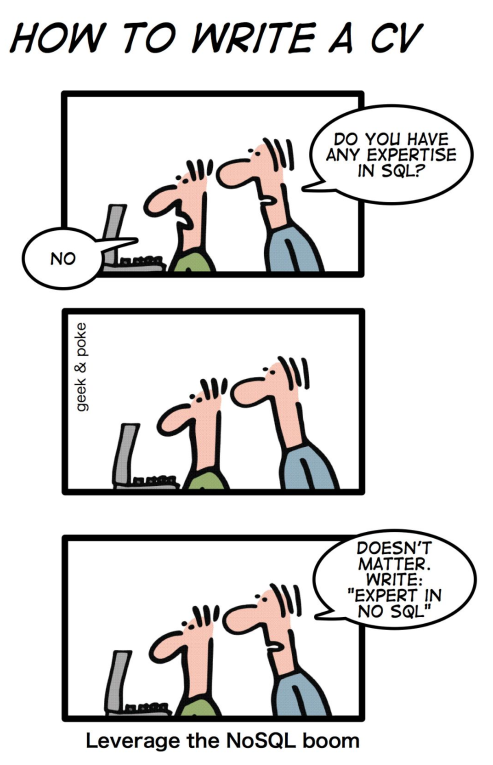
	CLOUD SPANNER	TRADITIONAL RELATIONAL	TRADITIONAL NON-RELATIONAL
Schema	**Yes**	Yes	No
SQL	**Yes**	Yes	No
Consistency	**Strong**	Strong	Eventual
Availability	**High**	Failover	High
Scalability	**Horizontal**	Vertical	Horizontal
Replication	**Automatic**	Configurable	Configurable

- Does it break the CAP theorem? NO!

https://cloud.google.com/spanner/

NoSQL

- A result of two trends:
  - More hardware
  - More data
- Data model
  - Not relational
  - No formally described schema
- Architecture
  - Usually distributed
- Mostly not ACID compliant
  - CAP = Consistency/Availability trade-off



-- TODO

- ✓ Scaling Up/Out

- ✓ NoSQL
  - ✓ Data Model
  - ✓ Architecture
    - ✓ Distributed
  - ✓ Consistency/Availability trade-off
    - ✓ Eventual Consistency
    - ✓ CAP Theorem

Takeaways

Why do NoSQL devs eat alone?

…

They don’t know how to join tables.

Scale Up vs <u>Scale Out</u>

NoSQL

- Non-relational Data Models
- Horizontal Scaling
- MongoDB, Redis, DynamoDB, Neo4j, etc.

CAP Theorem

- Consistency
- Availability
- Partitions
- Pick between Consistency and Availability in case of network partitions

Next Time in IDBS…

Introduction to Database Systems IDBS - Spring 2024

Lecture 12 - Big Data

Big Data Analytics
Distributed Computing Frameworks

Readings: PDBM 19.1-19.2, 19.4, 20.1-20.3

Eleni Tzirita Zacharatou