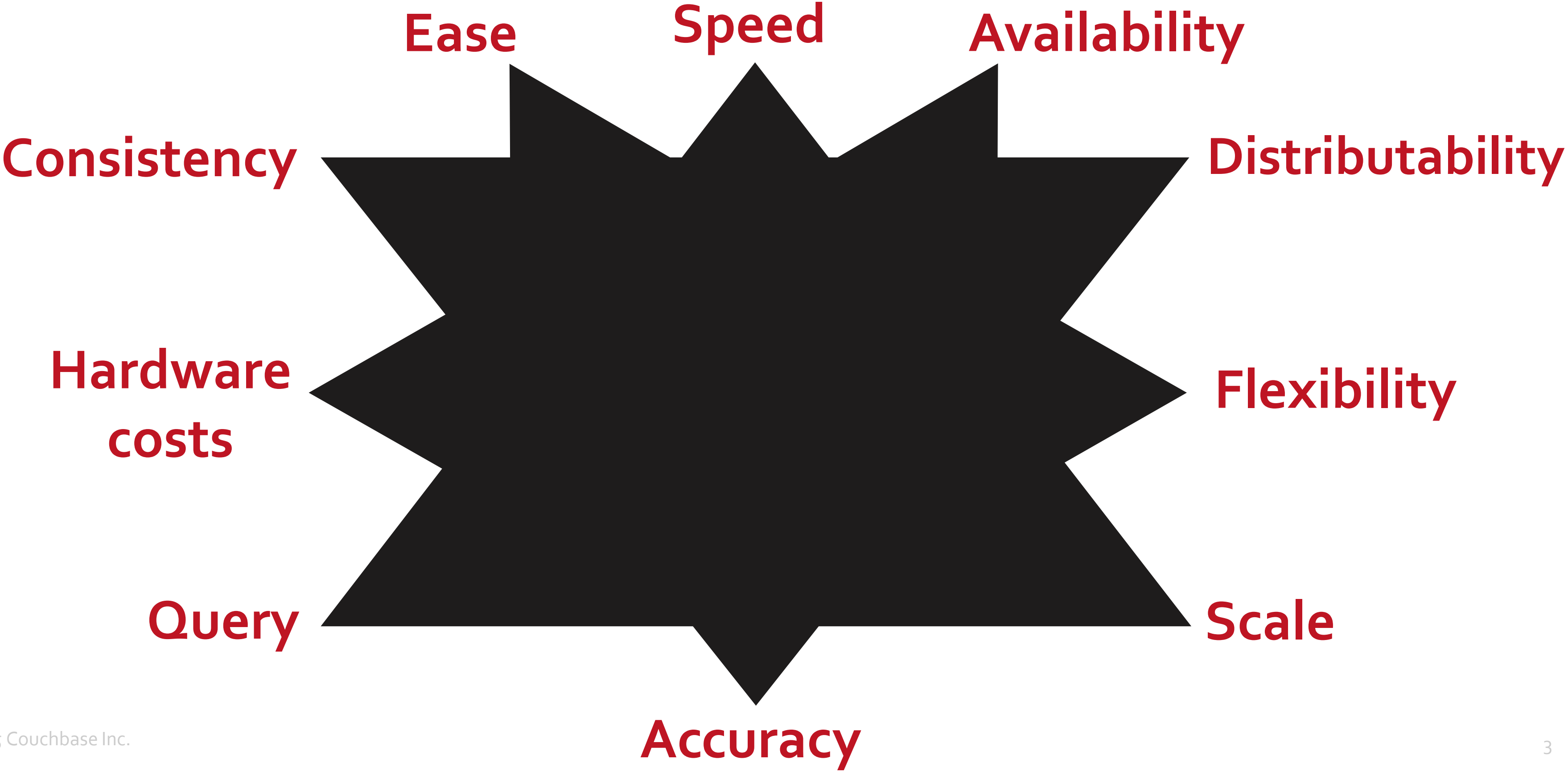
Speed, scale, query:
can NoSQL give us all three?

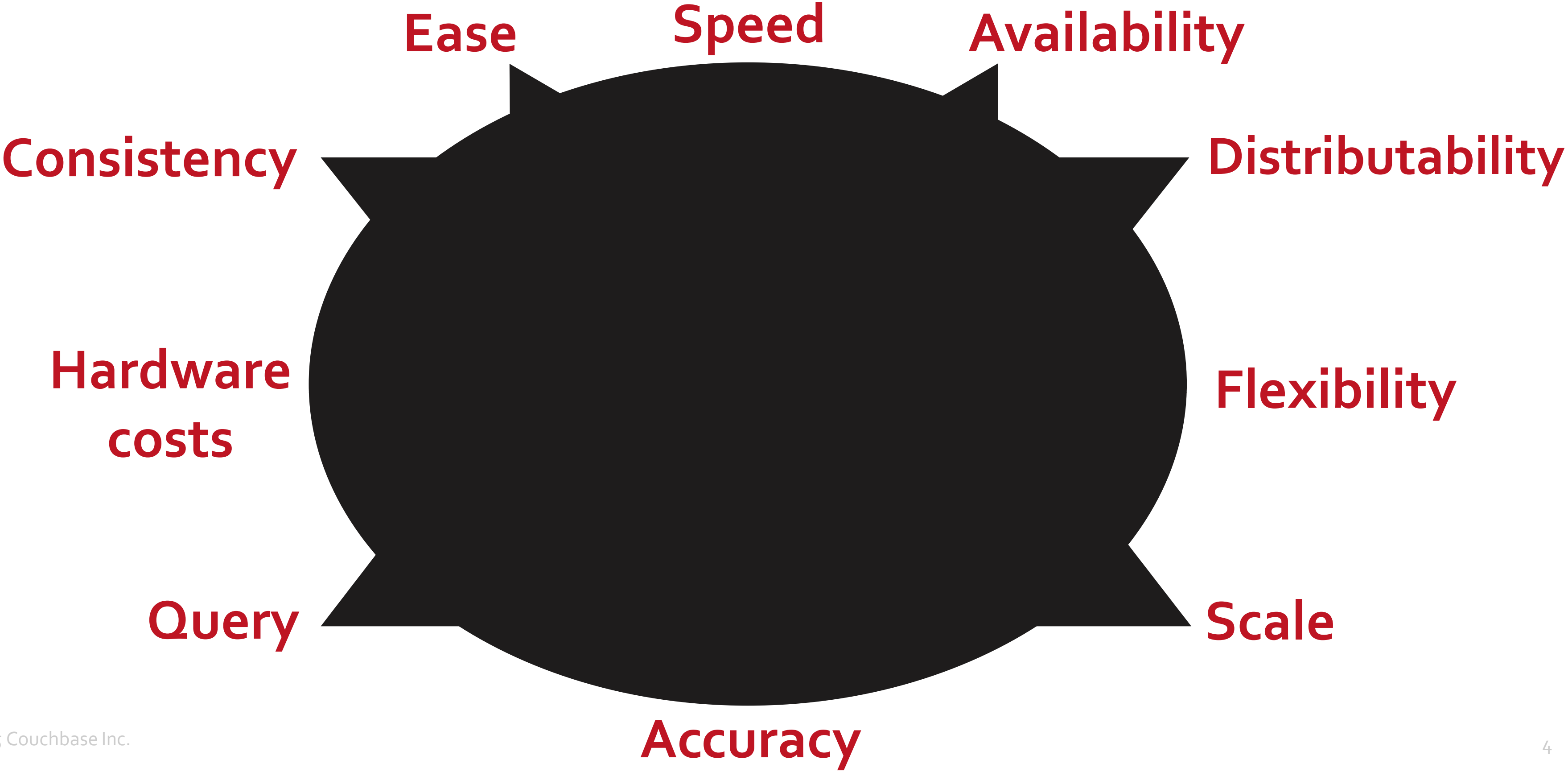
Arun Gupta, @arungupta
Matthew Revell, @matthewrevell
Couchbase

The project management triangle

The data storage star triangle



The data storage circle



What affects speed, scale and query?

First up: data models

What do we mean by NoSQL?

~~NoSQL~~

Non-Relational

Four popular NoSQL data models

Key-value

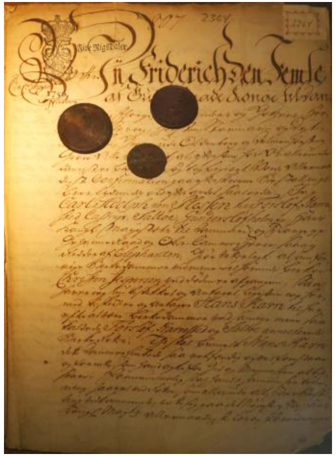
Key	Value
Logo	Couchbase
Email	advocates@couchbase.com
Profile	`{ "name": "A Person", "location": "Someplace" }`

Four popular NoSQL data models

Document

Key

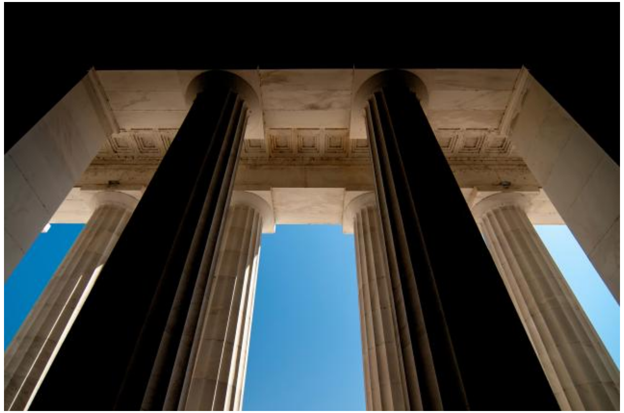
```
{
    "name": "A Person",
    "location": "Place",
    "team": "Team A",
    "interests": "music"
}
```



Four popular NoSQL data models

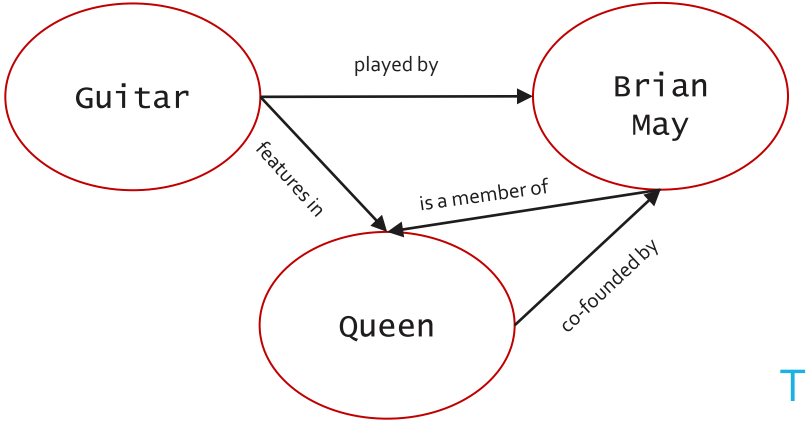
Columnar

Author Row	Title	Year of release
JK Rowling	Harry Potter and the Philosopher's Stone	1997
	Harry Potter and the Chamber of Secrets	1998
	Harry Potter and the NoSQL database	2016



Four popular NoSQL data models

Graph



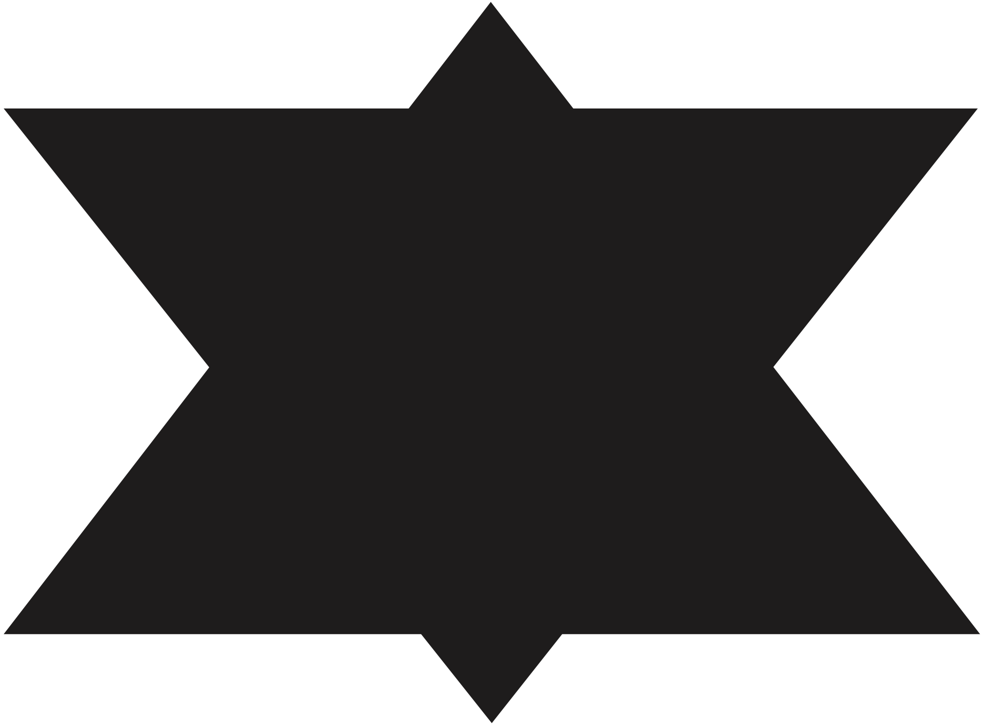
This is Euler

Data model is the first consideration

Relational

Object

Key-value



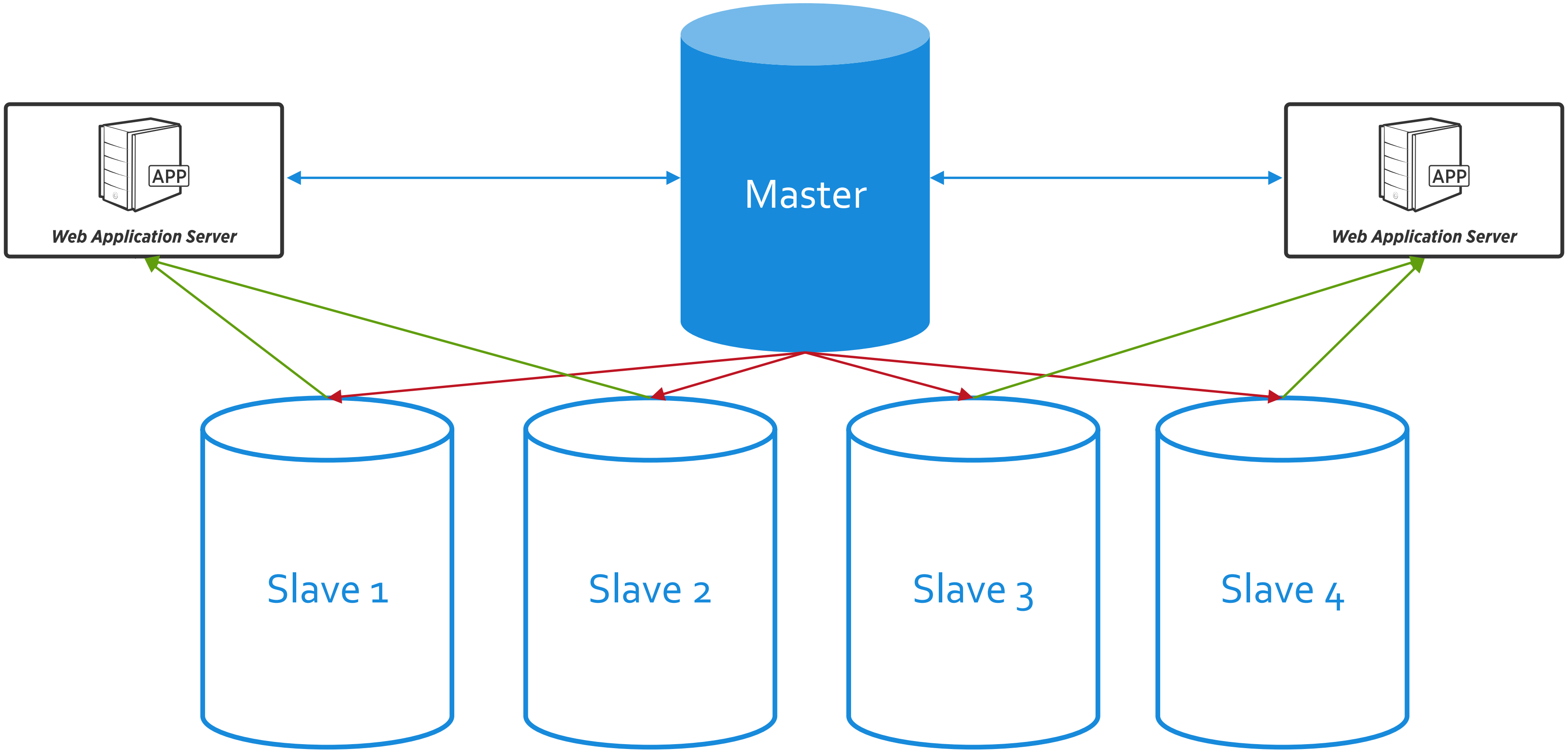
Graph

Document

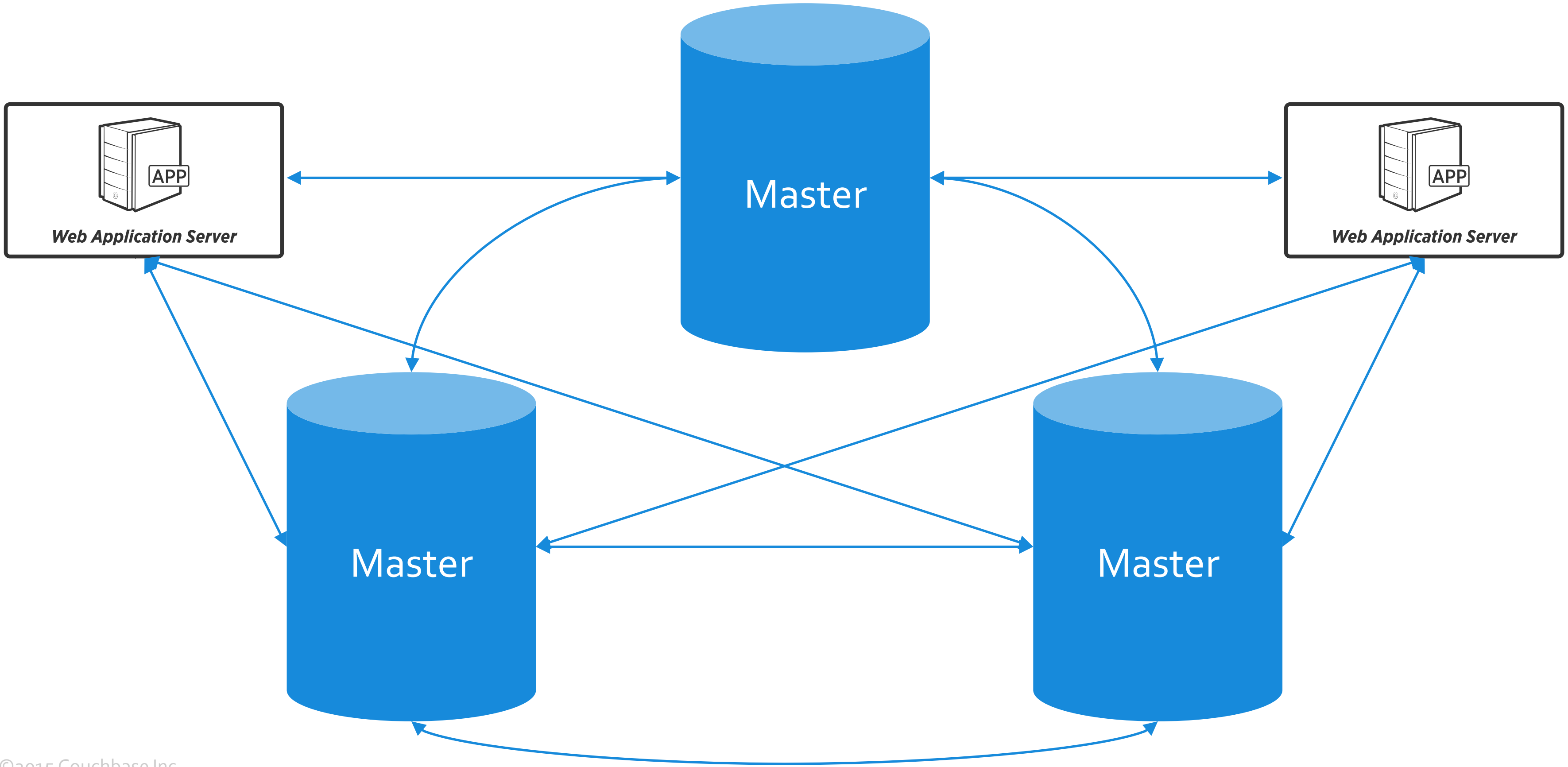
Columnar

Next up: architecture

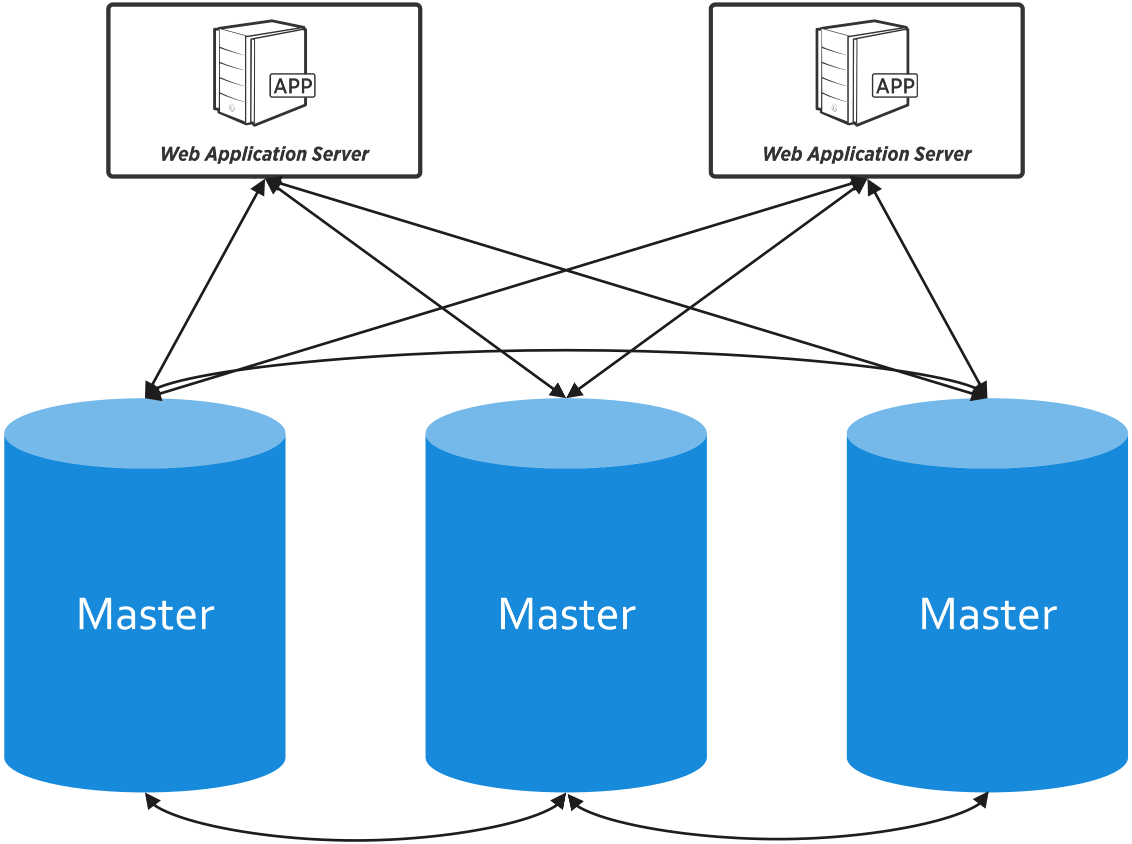
Master-slave



Master-master: replicated topology



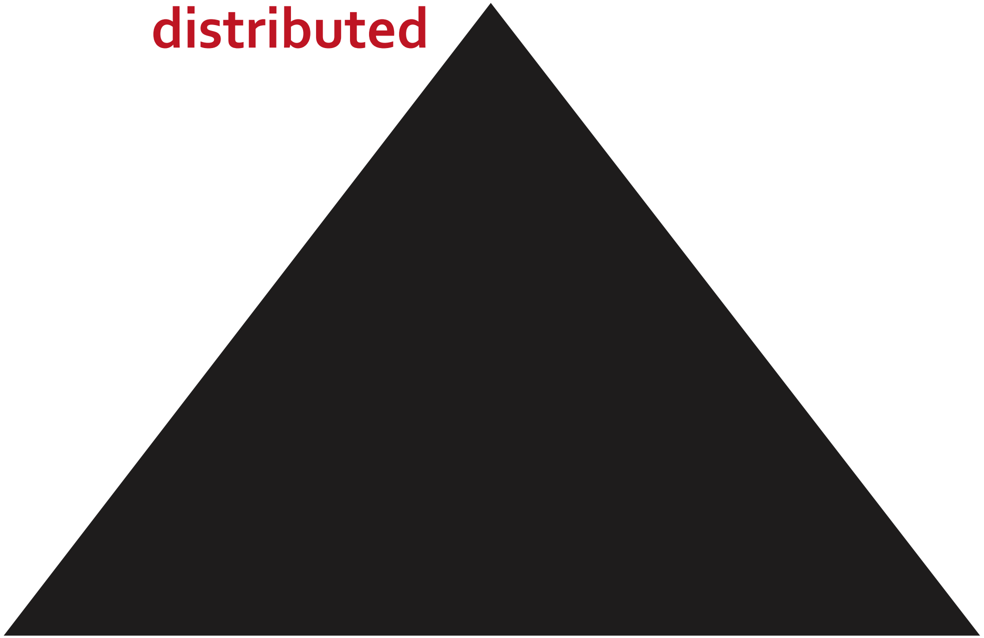
Master-master: distributed topology



Master-master: replicated v distributed

Replicated	Distributed
Dataset must fit on one machine	Dataset is sharded across machines: can be huge
Write to/read from any machine	
Eventually consistent	CP or AP

Architecture is the second consideration



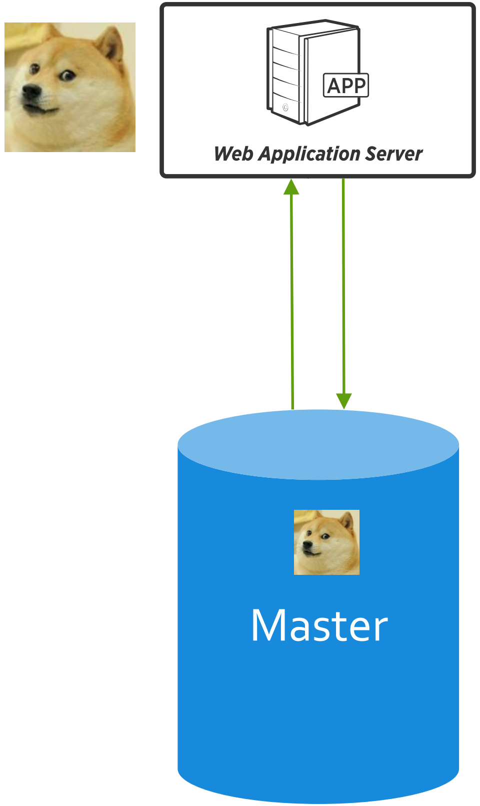
How do data model and architecture influence speed, query and scalability?

Four examples

- Single server key-value store
- Master-slave document store
- Multi-master eventually consistent column store
- Multi-master strongly consistent document store

Single server, key-value

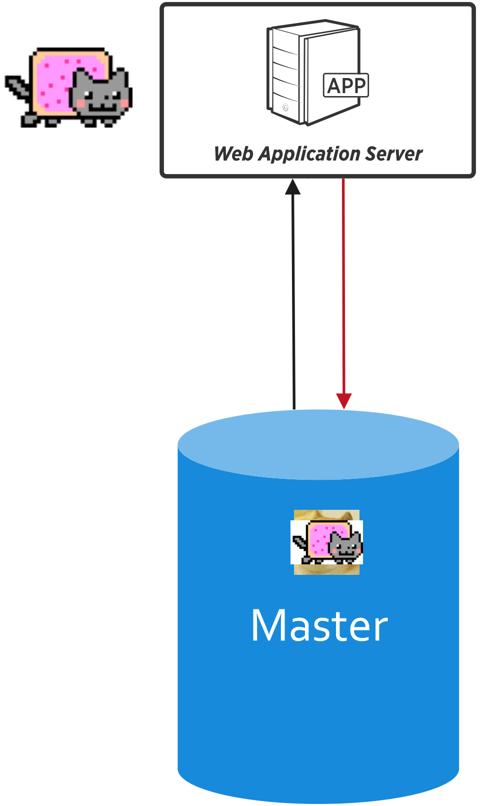
GET



Single server

GET

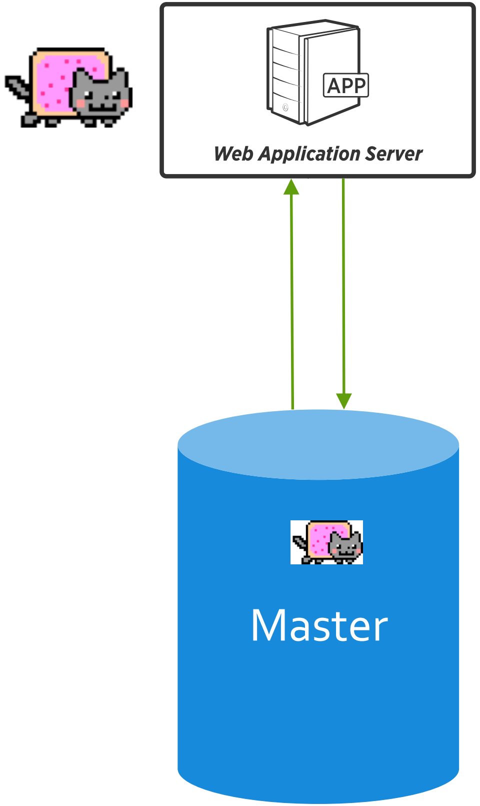
SET



Single server, key-value

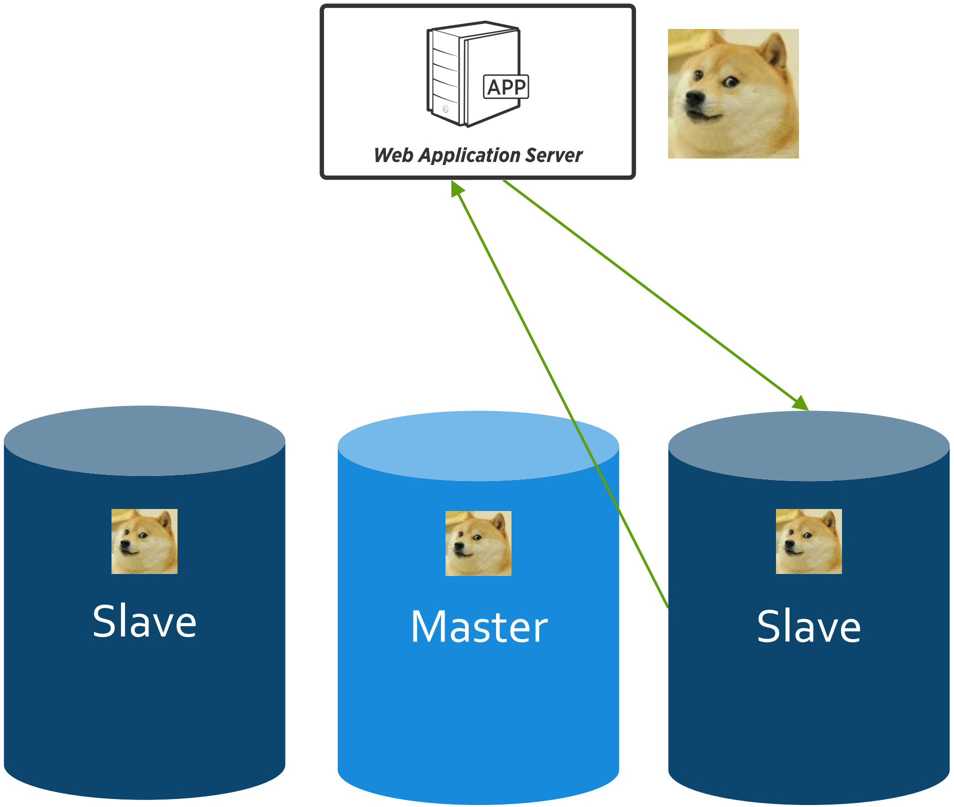
Impact

	Data model	Architecture
Speed	None	None
Query	None	None
Scale	Need to choose AP or CP	Manual sharding on the application layer



Master-slave, document

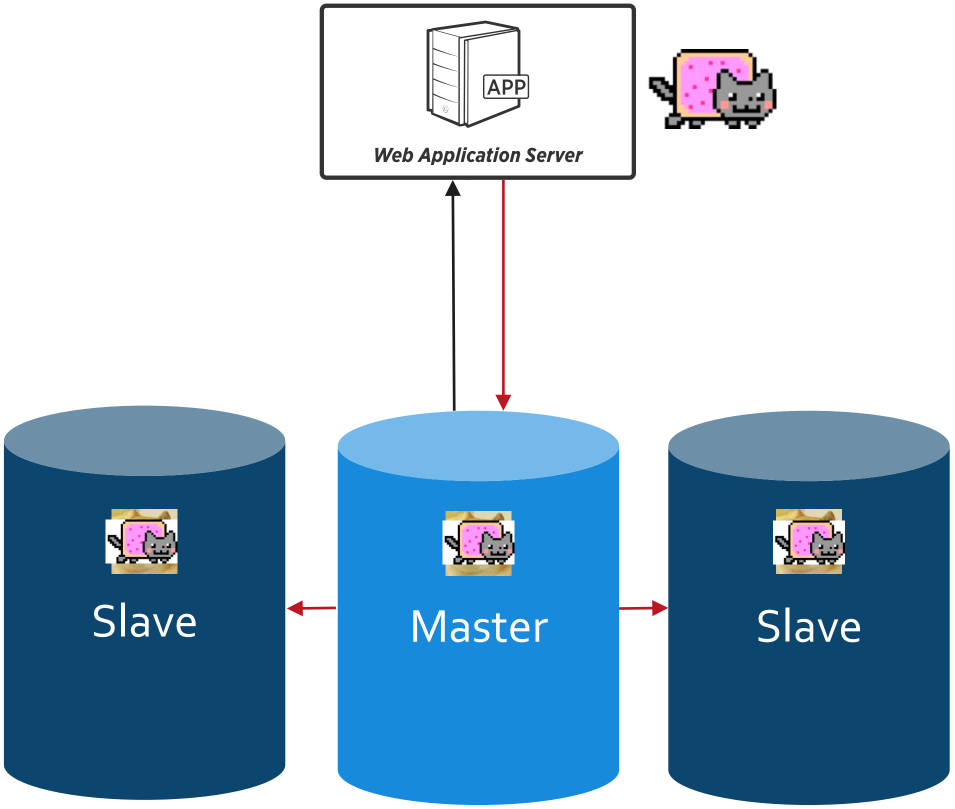
GET



Master-slave, document

GET

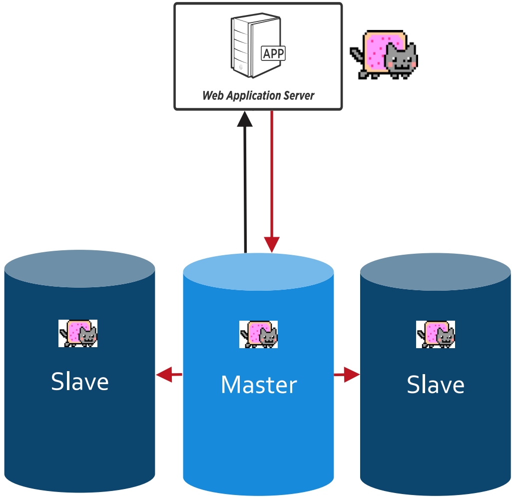
SET



Master-slave, document

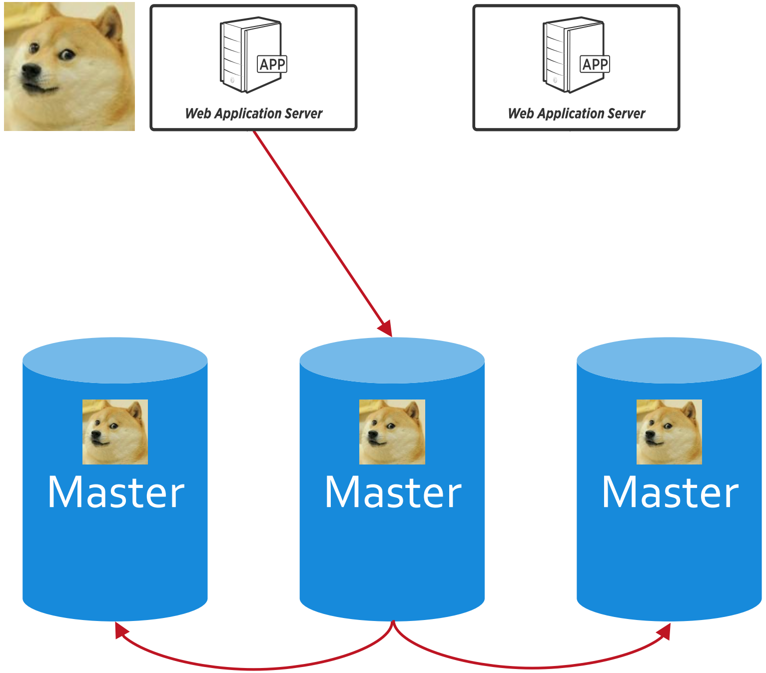
Impact

	Data model	Architecture
Speed	None	Favours reads, over writes
Query	Ad-hoc query possible	Eventual consistency
Scale	Distinct documents are easily distributed	Master is a SPOF



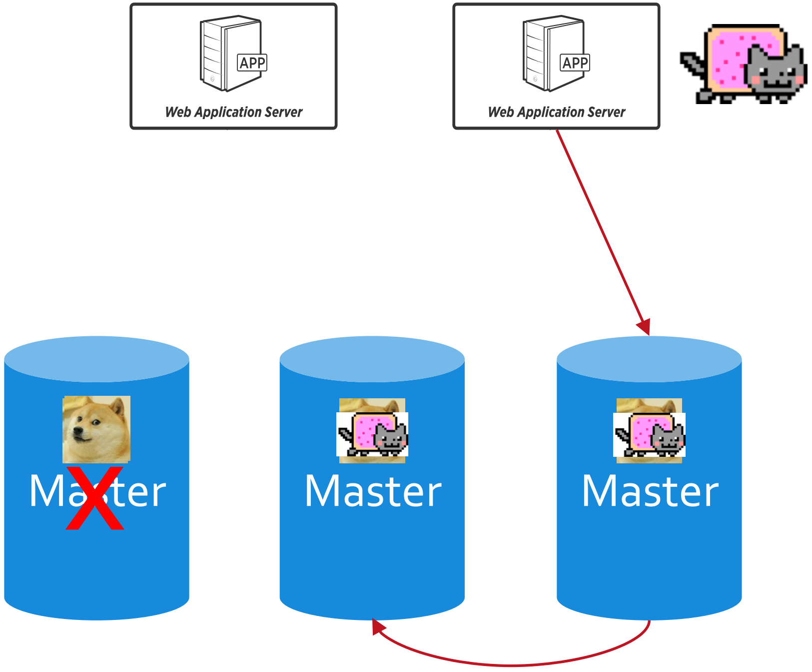
Dynamo-like distributed master-master column store

SET



Dynamo-like distributed master-master column store

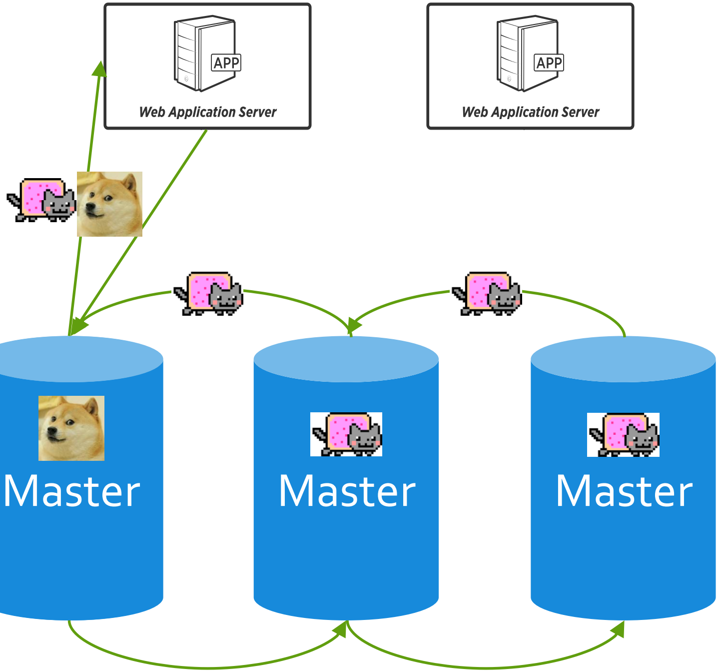
SET



Dynamo-like distributed master-master column store

GET

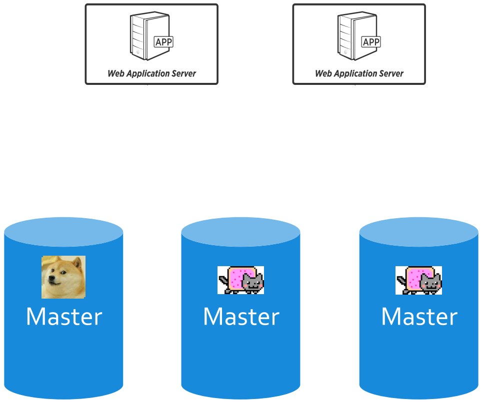
EVENTUAL CONSISTENCY!



Dynamo-like distributed master-master column store

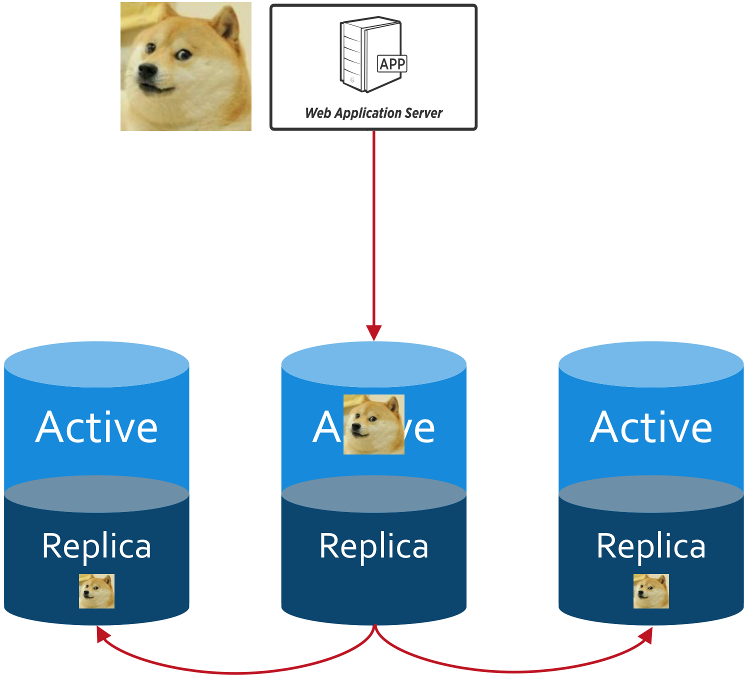
Impact

	Data model	Architecture
Speed	Favours writes, over reads	Favours reads, over writes
Query	Favours range-queries Ad-hoc not so easy	Eventual consistency complicates queries
Scale	None	No SPOFs High write availability Linear scalability



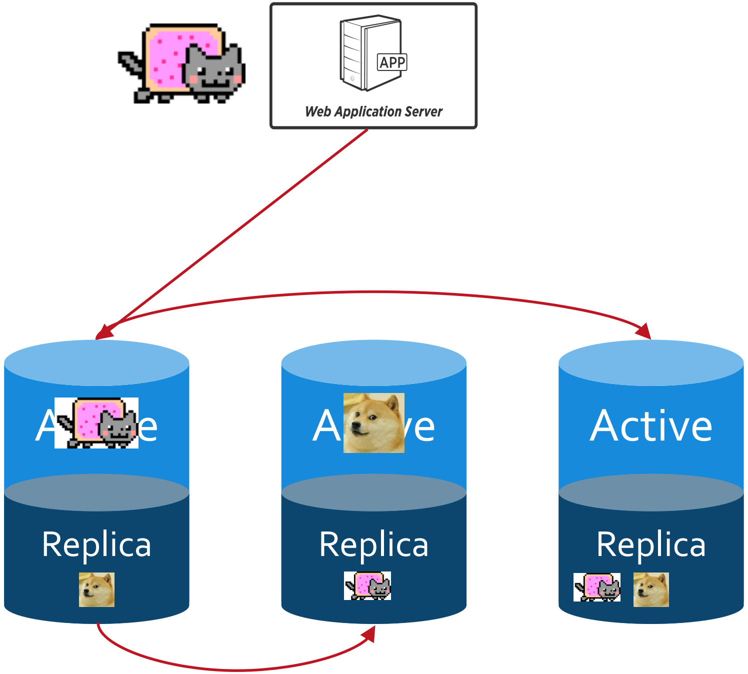
Distributed master-master, strong consistency, document

SET



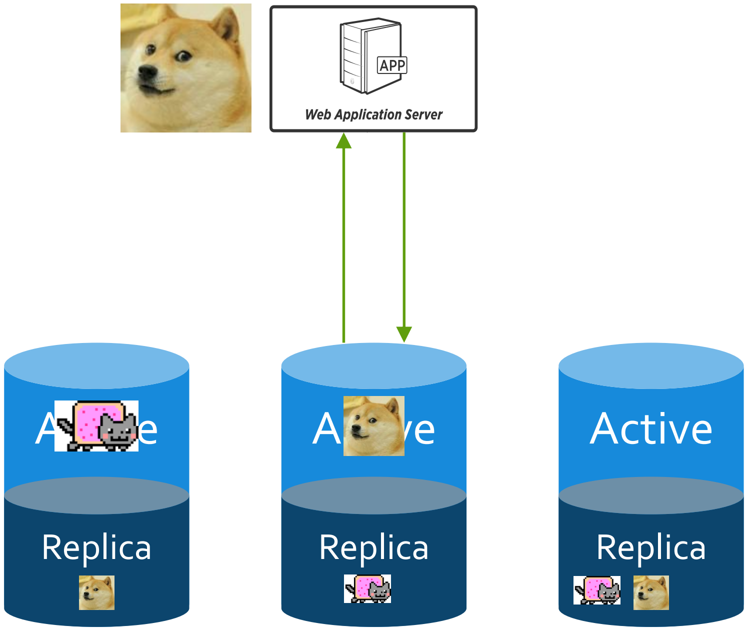
Distributed master-master, strong consistency, document

SET



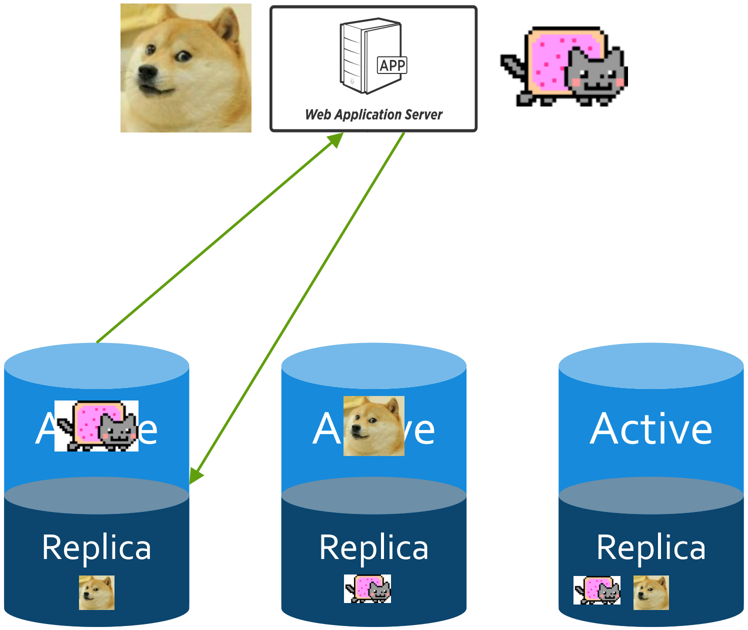
Distributed master-master, strong consistency, document

GET



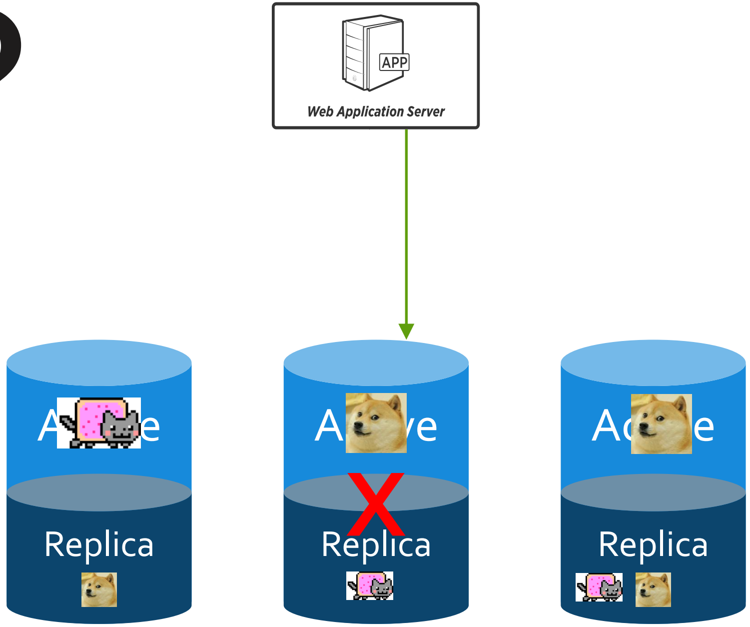
Distributed master-master, strong consistency, document

GET



Distributed master-master, strong consistency, document

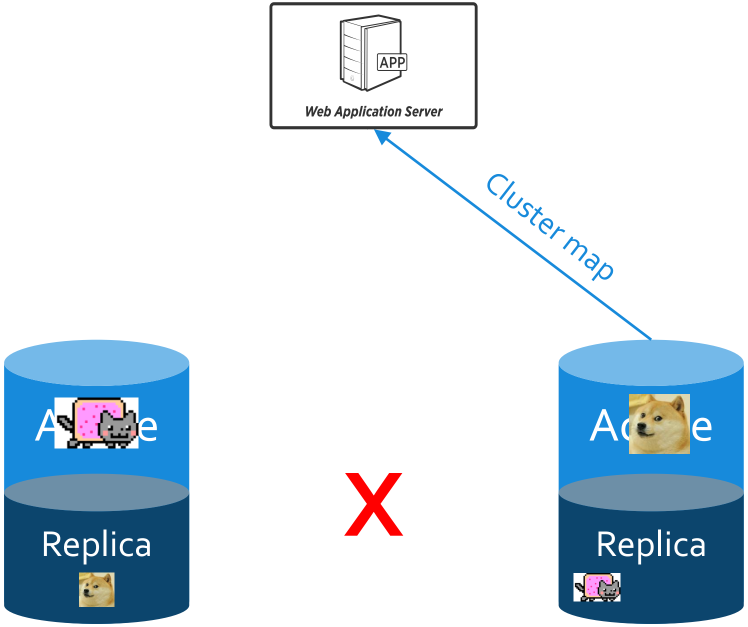
FAILURE?



Distributed master-master, strong consistency, document

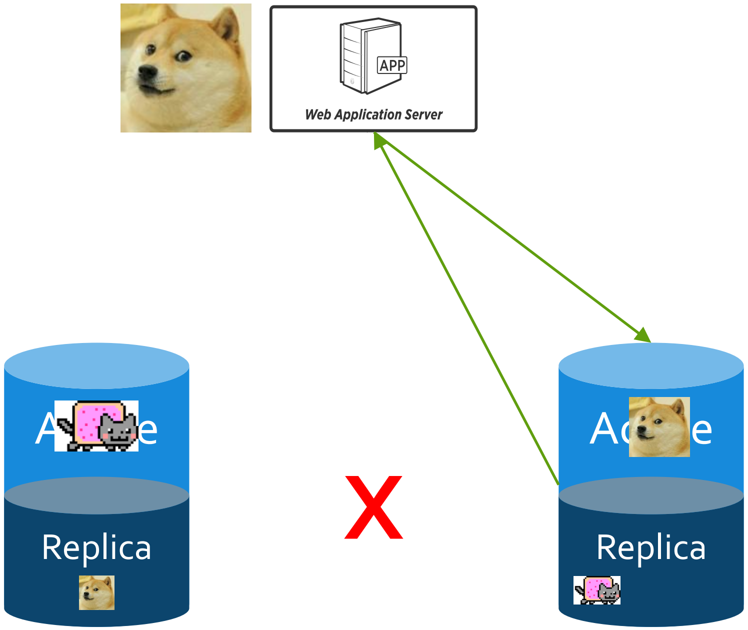
FAIL

OVER

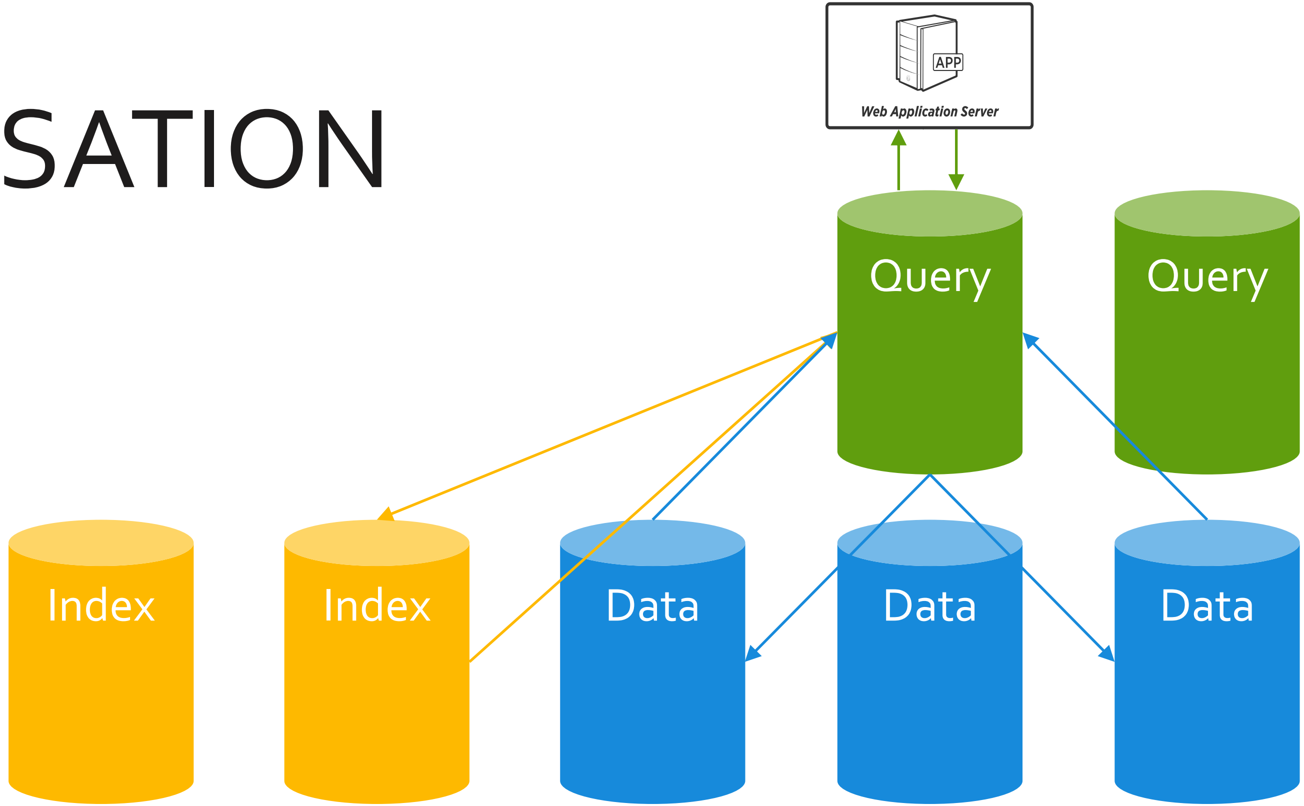


Distributed master-master, strong consistency, document

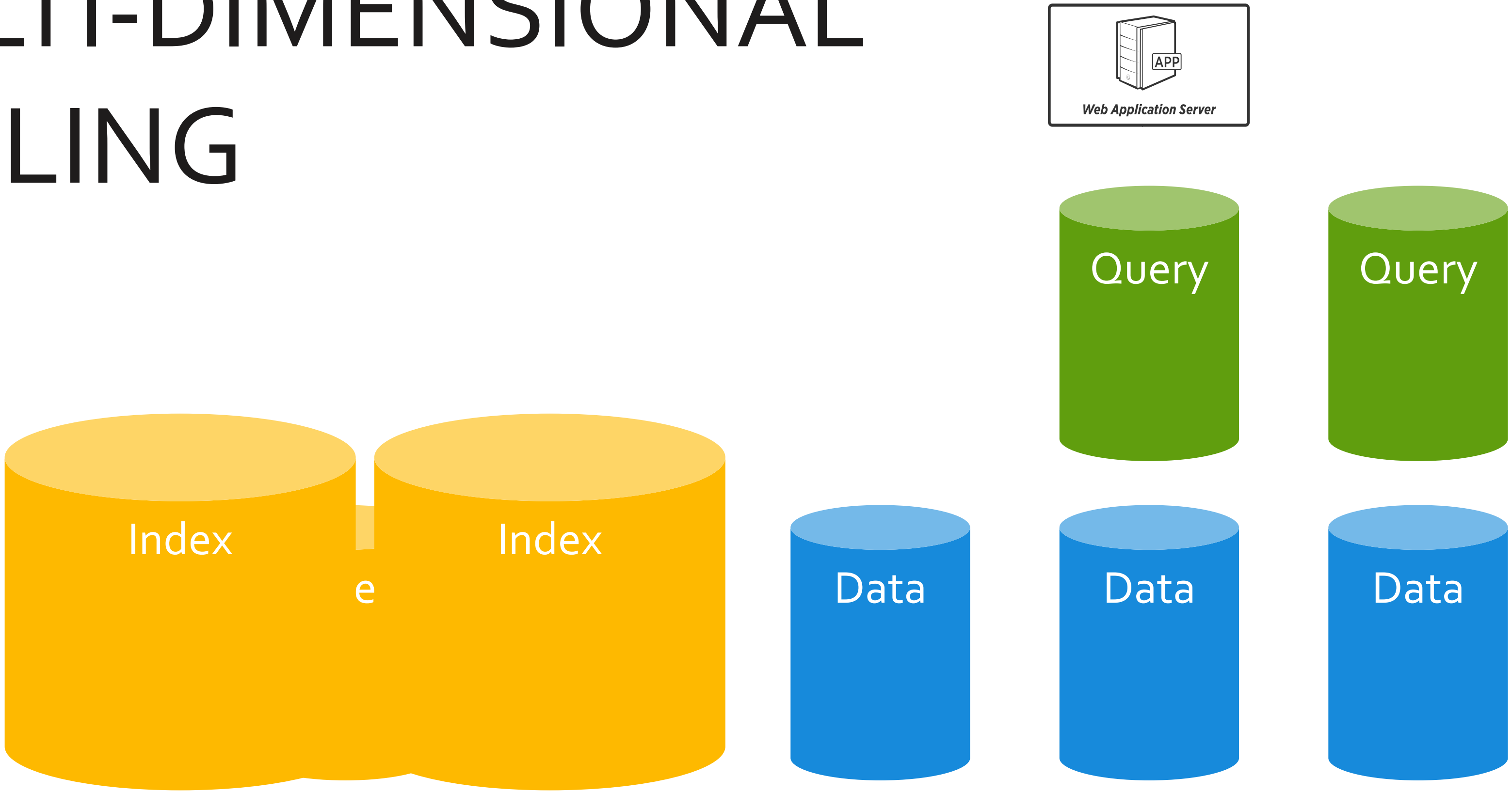
FAIL
OVER



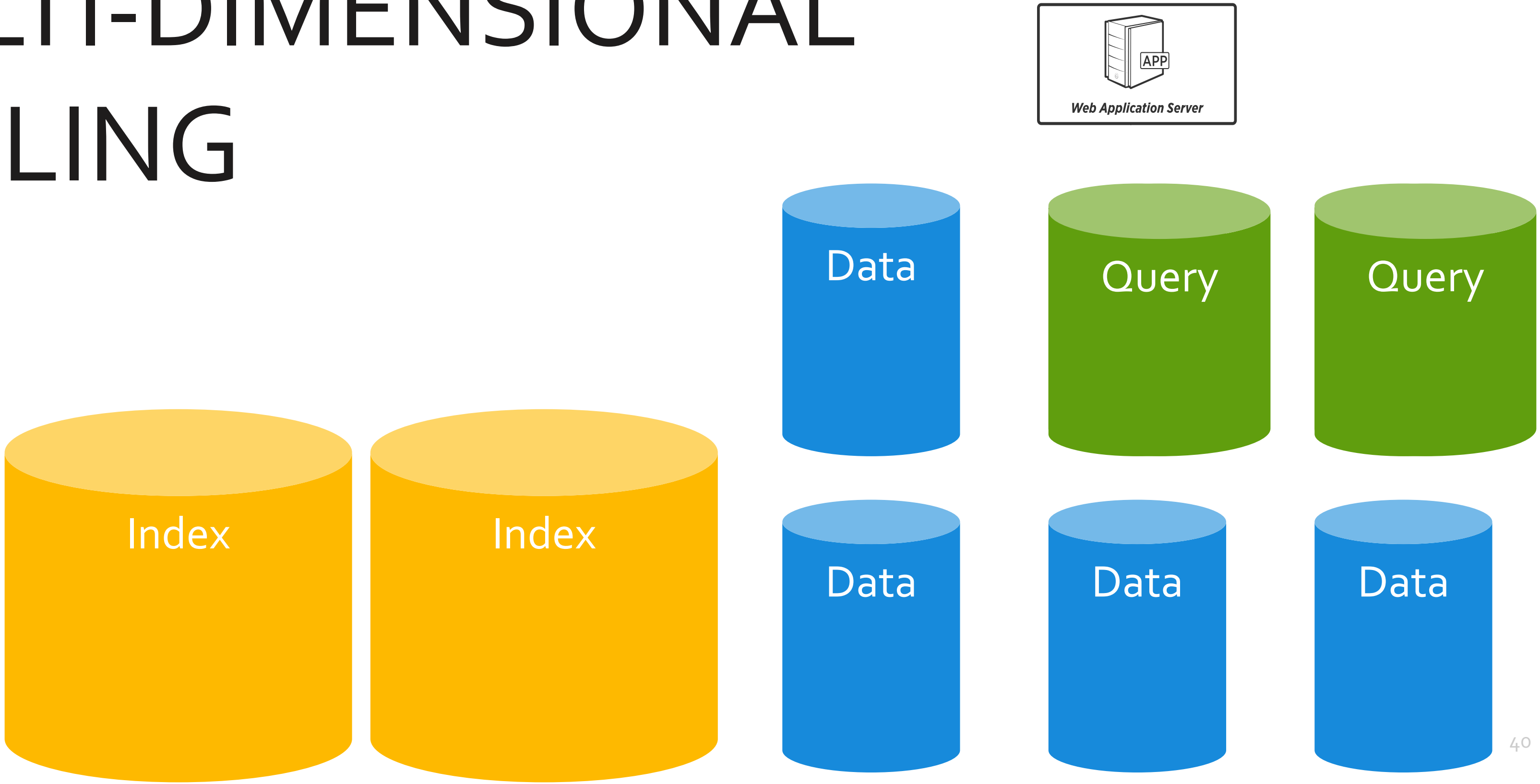
NODE SPECIALISATION



MULTI-DIMENSIONAL SCALING



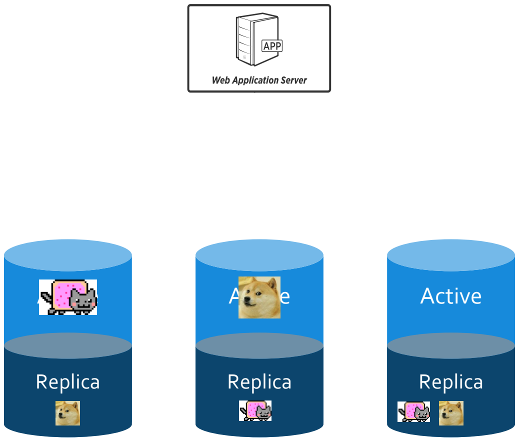
MULTI-DIMENSIONAL SCALING



Distributed master-master, strong consistency, document

Impact

	Data model	Architecture
Speed	Simple GETs and SETs	Single read, optionally single write
Query	Simplifies complex ad-hoc query	Strong consistency makes complex query easier
Scale	Distinct documents are easily distributed	Linear scalability No SPOFs No conflicts



Diving deeper into query

The first NoSQL approach to query

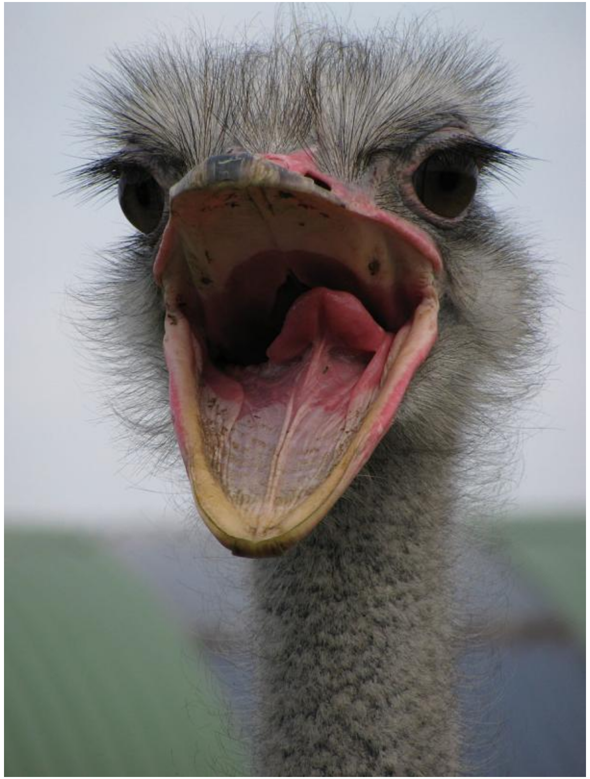
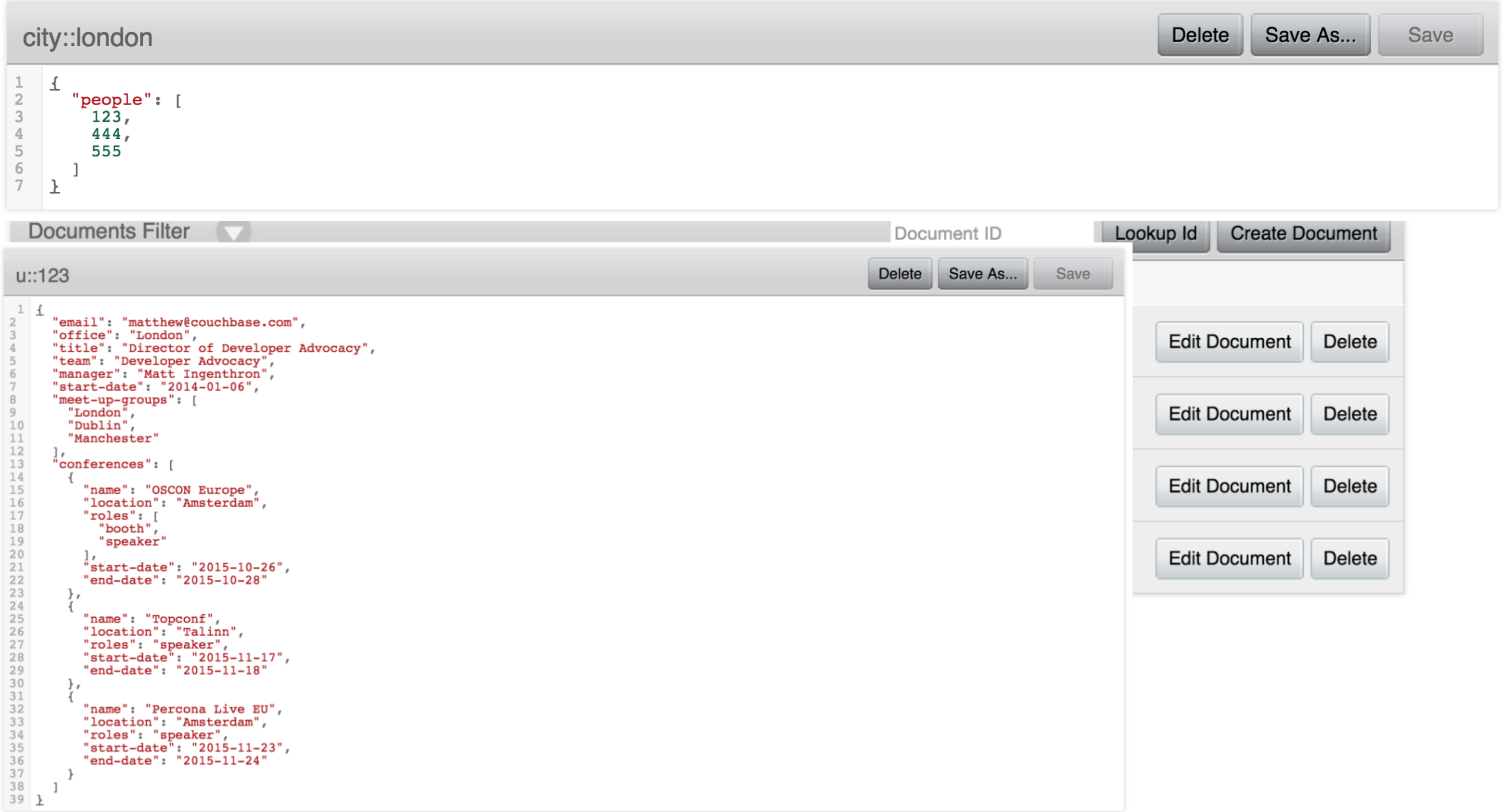
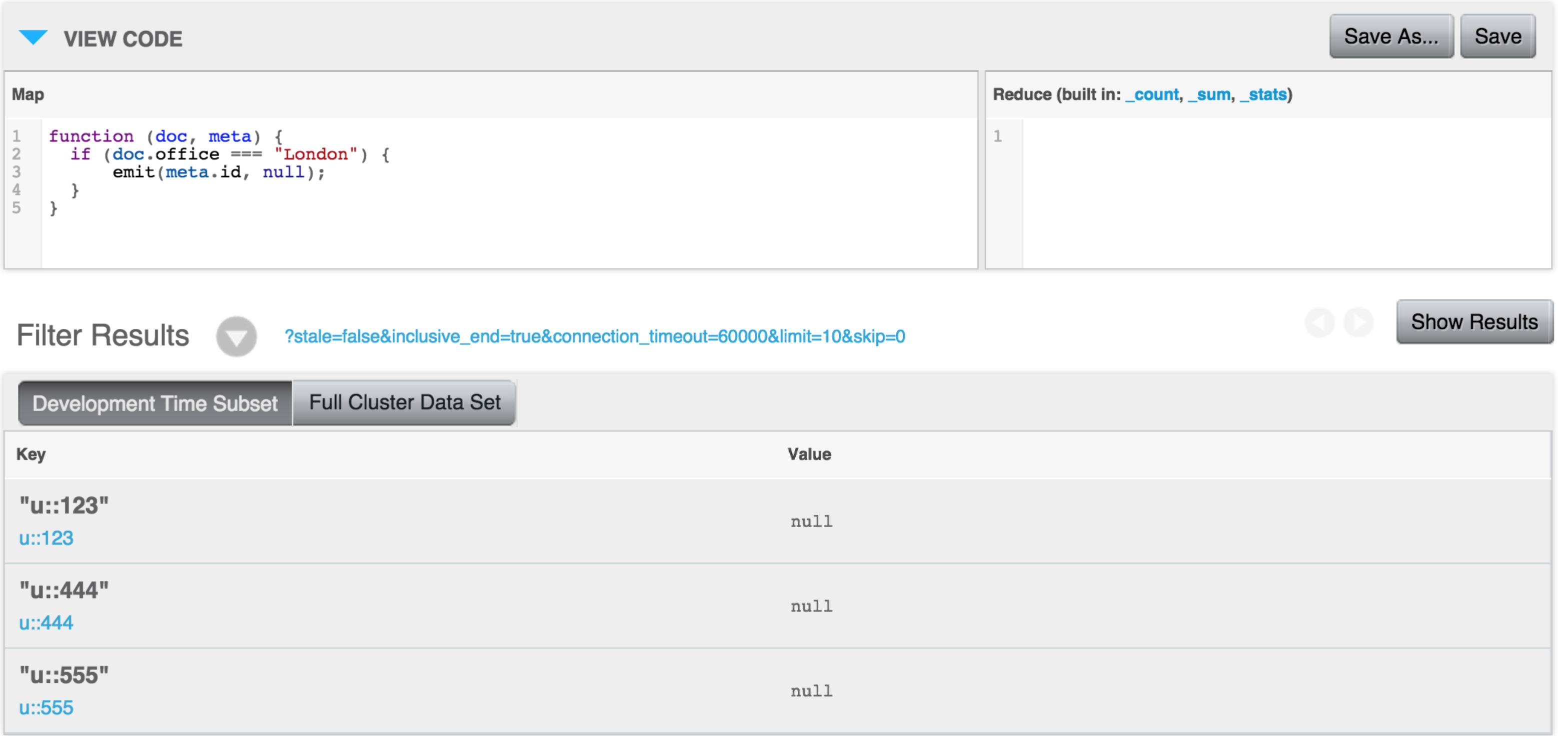


Photo by Donarreiskoffer. CC-by-3.0

Manual secondary indexes



Map-Reduce was one of the first steps towards query



Declarative query for NoSQL

Declarative query

- DB-specific: Neo4J's Cypher or MongoDB's query
- Attempts at standardisation: Jsoniq
- SQL reworked for a non-relational model

DB-specific query: MongoDB

```
db.staff.find({office: 'London'})

db.staff.find({office: {$in:['London', 'Amsterdam']}})

db.staff.insert({name: 'Matthew Revell', office:
'London'})

db.staff.update({name: 'Matthew Revell',
                 office: 'Amsterdam'})
```

Attempt at standardisation: JSONiq

- Based on XQuery
- Functional language
- Works with sets, rather than tuples

Attempt at standardisation: JSONiq

```
for $p in collection('staff')
where $p.serviceyears gt 2
let $name := $p.firstname || " " || $p.lastname
group by $p.office
order by $p.serviceyears
return { $name, $p.office, $p.serviceyears }
```

SQL for NoSQL: what needs to change?

- Data is nested
- Schema is unenforced, so data is heterogenous
- Data is not normalised

SQL for NoSQL: SQL++

The SQL++ Query Language: Configurable, Unifying and Semi-structured

Kian Win Ong, Yannis Papakonstantinou, Romain Vernoux
{kianwin,yannis,rvernoux}@cs.ucsd.edu

arXiv:1405.3631v7 [cs.DB] 29 Apr 2015

ABSTRACT

NoSQL databases support semi-structured data, typically modeled as JSON. They also provide limited (but expanding) query languages. Their idiomatic, non-SQL language constructs, the many variations, and the lack of formal semantics inhibit deep understanding of the query languages, and also impede progress towards clean, powerful, declarative query languages.

This paper specifies the syntax and semantics of SQL++, which is applicable to both JSON native stores and SQL databases. The SQL++ semi-structured data model is a superset of both JSON and the SQL data model. SQL++ offers powerful computational capabilities for processing semi-structured data akin to prior non-relational query languages, notably OQL and XQuery. Yet, SQL++ is SQL backwards compatible and is generalized towards JSON by introducing only a small number of query language extensions to SQL. Indeed, the SQL capabilities are most often extended by removing semantic restrictions of SQL, rather than inventing new features.

Recognizing that a query language standard is probably premature for the fast evolving area of NoSQL databases, SQL++ includes configuration options that formally itemize the semantics variations that language designers may choose from. The options often pertain to the treatment of semi-structuredness (missing attributes, heterogeneous types, etc), where more than one sensible approaches are possible.

SQL++ is unifying: By appropriate choices of configuration options, the SQL++ semantics can morph into the semantics of existing semi-structured database query languages. The extensive experimental validation shows how SQL and four semi-structured database query languages (MongoDB, Cassandra CQL, Couchbase N1QL and AsterixDB AQL) are formally described by appropriate settings of the configuration options.

Early adoption signs of SQL++ are positive: Version 4 of Couchbase's N1QL is explained as syntactic sugar over SQL++. AsterixDB will soon support the full SQL++ and Apache Drill is in the process of aligning with SQL++.

1. INTRODUCTION

Numerous databases marketed as SQL-on-Hadoop, NewSQL and NoSQL support Big Data applications. These databases generally support the 3Vs [7]. (i) Volume: amount of data (ii) Velocity: speed of data in and out (iii) Variety: semi-structured and heterogeneous data. Due to the Variety requirement, they have adopted semi-structured data models, which are generally different subsets of enriched JSON.[1]

Their evolving query languages fall short of full-fledged semi-structured query language capabilities[2] and have many variations. Some variations are due to superficial syntactic differences. However, other variations are genuine differences in query language capabilities and semantics. The lack of succinct, formal syntax and semantics inhibits a deep understanding of the various systems. It also impedes progress towards declarative languages for querying semi-structured data.

SQL++ is a semi-structured query language that is *backwards compatible* with SQL, in order to be easily understood and adopted by SQL programmers. The described semi-structured SQL++ data model is a superset of JSON and the SQL data model. The SQL++ model expands JSON with bags (as opposed to having JSON arrays only) and enriched values, i.e., atomic values that are not only numbers and strings (vendors have already adopted this extension [5]). Vice versa, one may think of SQL++ as expanding SQL with JSON features: arrays, heterogeneity, and the possibility that any value may be an arbitrary composition of the array, bag and tuple constructors, hence enabling arbitrary nested structures, such as arrays of arrays. The SQL++ query language inputs and outputs SQL++ data. It makes the following contributions towards the evolution of query languages for JSON databases.

Full-fledged semi-structured language Many commercial JSON databases started as key-value and document-oriented databases. Others started with SQL as their base. In either case, they grow towards full-fledged JSON databases. SQL++ provides a full-fledged target language whose semantics pick the salient features of past full-fledged *declarative* query languages for non-relational data models: OQL [2], the nested relational model and query languages [8, 15, 1] and XQuery (and other XML-based query languages) [14, 6, 4]. Importantly, in the spirit of XQuery and OQL, SQL++ is a fully composable and semi-structured language, hence being able to input and output nested and heterogeneous

9[1] As explained below, the SQL data model itself is a subset of enriched JSON.

9[2] They also fall short of full-fledged SQL capabilities also.

1

SQL for NoSQL: SQL++

- Superset of SQL for semi-structured data
- Handles missing data gracefully and/or explicitly
- Can query inside nested data
- Nests and unnests data in results
- JOINs between documents

Introducing N1QL: SQL++ in action

Finding all the airports

2. cbq

```
SELECT * FROM `travel-sample`
WHERE type='airport';
```

Limiting and ordering our results

2. cbq

```
SELECT * FROM `travel-sample`
WHERE type='airport'
ORDER BY COUNTRY
LIMIT 10;
```

Working with documents

```
{
  "id": 3469,
  "type": "airport",
  "airportname": "San Francisco Intl",
  "city": "San Francisco",
  "country": "United States",
  "faa": "SFO",
  "icao": "KSFO",
  "tz": "America/Los_Angeles",
  "geo":
  {
    "lat": 37.618972,
    "lon": -122.374889,
    "alt": 13
  }
}
```

Working with nested data

2. cbq

```
SELECT a.name, s.flight, s.utc, r.sourceairport,
r.destinationairport, r.equipment
FROM `travel-sample` r
UNNEST r.schedule s
JOIN `travel-sample` a
ON KEYS r.airlineid
WHERE r.sourceairport="LHR"
AND r.destinationairport = "SFO"
AND s.day=1
ORDER BY s.utc;
```

Flying to and from SFO

```
{
        "callsign": "UNITED",
        "country": "United States",
        "iata": "UA",
        "icao": "UAL",
        "id": 5209,
        "name": "United Airlines",
        "type": "airline"
}
```

```
{
  "id": 3469,
  "type": "airport",
  "airportname": "San Francisco Intl",
  "city": "San Francisco",
  "country": "United States",
  "faa": "SFO",
  "icao": "KSFO",
  "tz": "America/Los_Angeles",
  "geo":
  {
    "lat": 37.618972,
    "lon": -122.374889,
    "alt": 13
  }
}
```

```
{
        "airline": "UA",
        "airlineid": "airline_5209",
        "destinationairport": "SFO",
        "equipment": "777",
        "id": 57047,
        "schedule": [
        {
                "day": 0,
                "flight": "UA894",
                "utc": "02:32:00"
},
                ...
        ],
        "sourceairport": "LHR",
        "stops": 0,
        "type": "route"
}
```

Prepared statements

- Optimise frequently-run queries
- Execution plan happens once, query is run multiple times

2. cbq

```
PREPARE LonSanFran FROM
SELECT airline FROM `travel-sample`
WHERE sourceairport="LHR"
AND destinationairport = "SFO";
```

Creating indexes

```
CREATE IND
ON `travel
WHERE type
AND countr
USING GSI;
```

Indexes

Bucket	Node	Index Name	Status	Initial Build Progress
default	127.0.0.1:8091	#primary	Ready	100%
travel-sample	127.0.0.1:8091	def_airportname	Ready	100%
travel-sample	127.0.0.1:8091	def_city	Ready	100%
travel-sample	127.0.0.1:8091	def_faa	Ready	100%
travel-sample	127.0.0.1:8091	def_icao	Ready	100%
travel-sample	127.0.0.1:8091	def_name_type	Ready	100%
travel-sample	127.0.0.1:8091	def_primary	Ready	100%
travel-sample	127.0.0.1:8091	def_sourceairport	Ready	100%
travel-sample	127.0.0.1:8091	def_type	Ready	100%
travel-sample	127.0.0.1:8091	ukairports	Ready	100%

Definition: CREATE INDEX ukairports ON travel-sample(`type`) WHERE ((`type` = "airport") and (`country` = "UK")) USING GSI

Mutating data

- DELETE: provide the key to delete the document
- INSERT: provide a key and some JSON to create a new document
- UPSERT: as INSERT but will overwrite existing docs
- UPDATE: change individual values inside existing docs

Recapping NoSQL speed

Speed and NoSQL

Data model considerations	
Key-value	No CPU load, minimal disk seeks
Document	Largely single ops, minimal disk seeks, often relatively simple query
Column	Rapid writes
Graph	Simplifies otherwise expensive queries

Architecture considerations	
Master-slave	Speeds up reads, slows writes
Master-master replicated	Speeds up reads and writes (with consistency lag)
Master-master distributed	Speeds up reads and writes

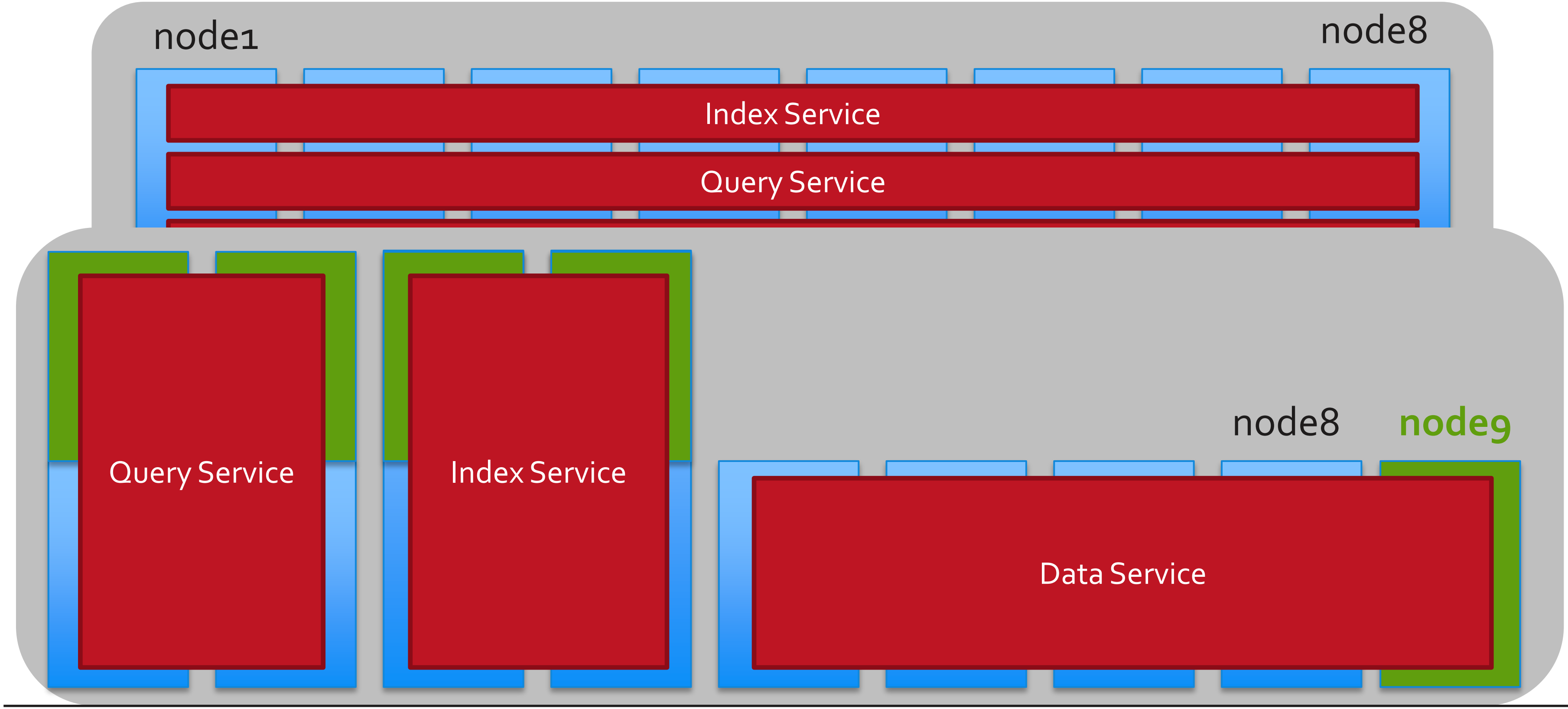
Recapping NoSQL scale

Scale and NoSQL

Data model considerations	
Key-value	Each item is independent, easily distributed
Document	Item independence, easily distributed. Indexes might bring cross-node dependencies.
Column	Distribute column families, hashed sharding
Graph	Shard based on data

Architecture considerations	
Master-slave	Speeds up reads, slows writes
Master-master replicated	Speeds up reads and writes (with consistency lag)
Master-master distributed	Speeds up reads and writes

Scale out, scale up or both: multi-dimensional scaling



Next steps

What next?

- Developer portal: developer.couchbase.con
- Forums: forums.couchbase.com
- Free online training: training.couchbase.com/online
- Join your local Couchbase meet-up: bit.ly/couchbasemeetups
- Follow the Couchbase developer community on Twitter: @couchbasedev

Thank you

Q&A