NoSQL Database Systems

Presented by: Sai Vadlamudi and Tejbir Singh

NoSQL

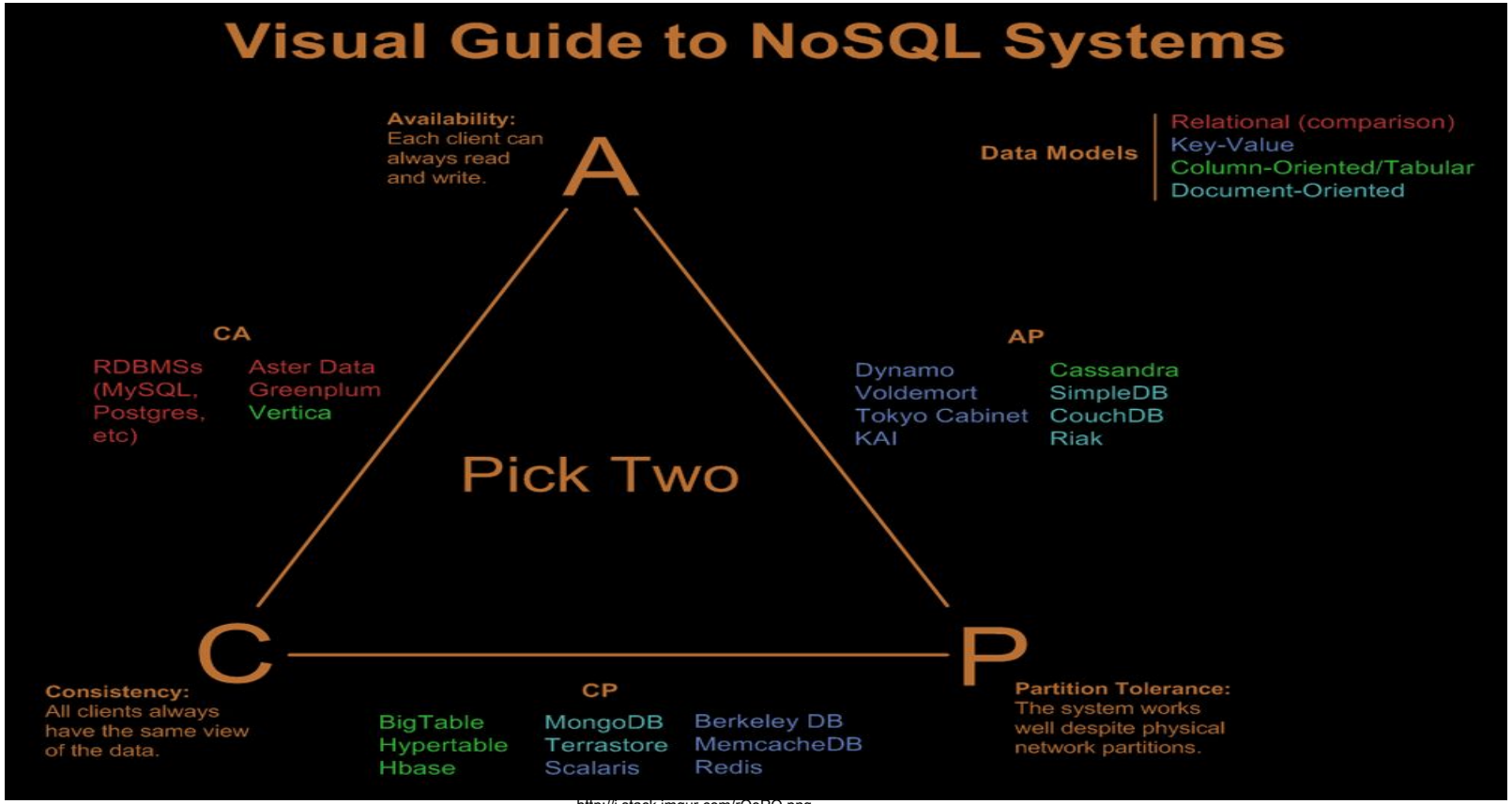
BASE

- Basically Available
- Soft State
- Eventual Consistency

ACID

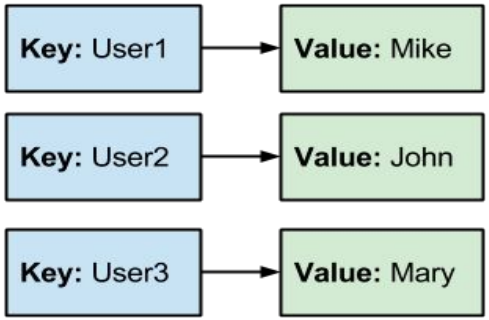
- Atomicity
- Consistency
- Isolation
- Durable

CAP Theorem



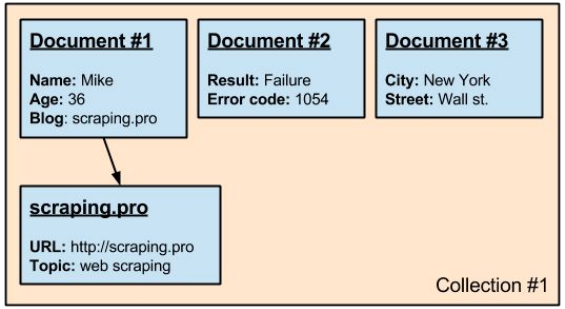
http://i.stack.imgur.com/rOeRQ.png

Data Models

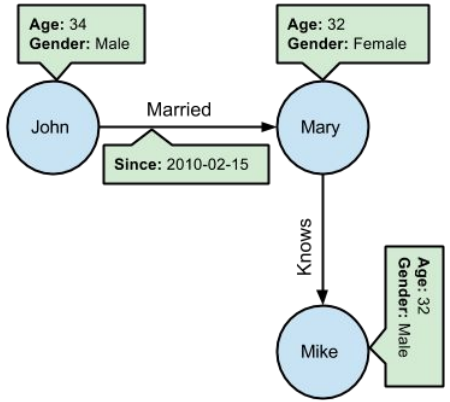


http://scraping.pro/res/nosql/keyvalue_database.png

http://scraping.pro/res/nosql/column_database.png



http://scraping.pro/res/nosql/document_database.png



http://scraping.pro/res/nosql/graph_database.pnghttp://scraping.pro/res/nosql/graph_database.png

Background: SQL vs. NoSQL

- Data modeling: schema-less

  -Relational: driven by the structure of available data

  -NoSQL: driven by application- specific pattern

- Query capability:

  - Relational: human user-oriented, query is simple

  - NoSQL: application-oriented, query is comparatively complex

- Scalability:

  - Relational: vertical

  - NoSQL: horizontal

Background: SQL vs. NoSQL

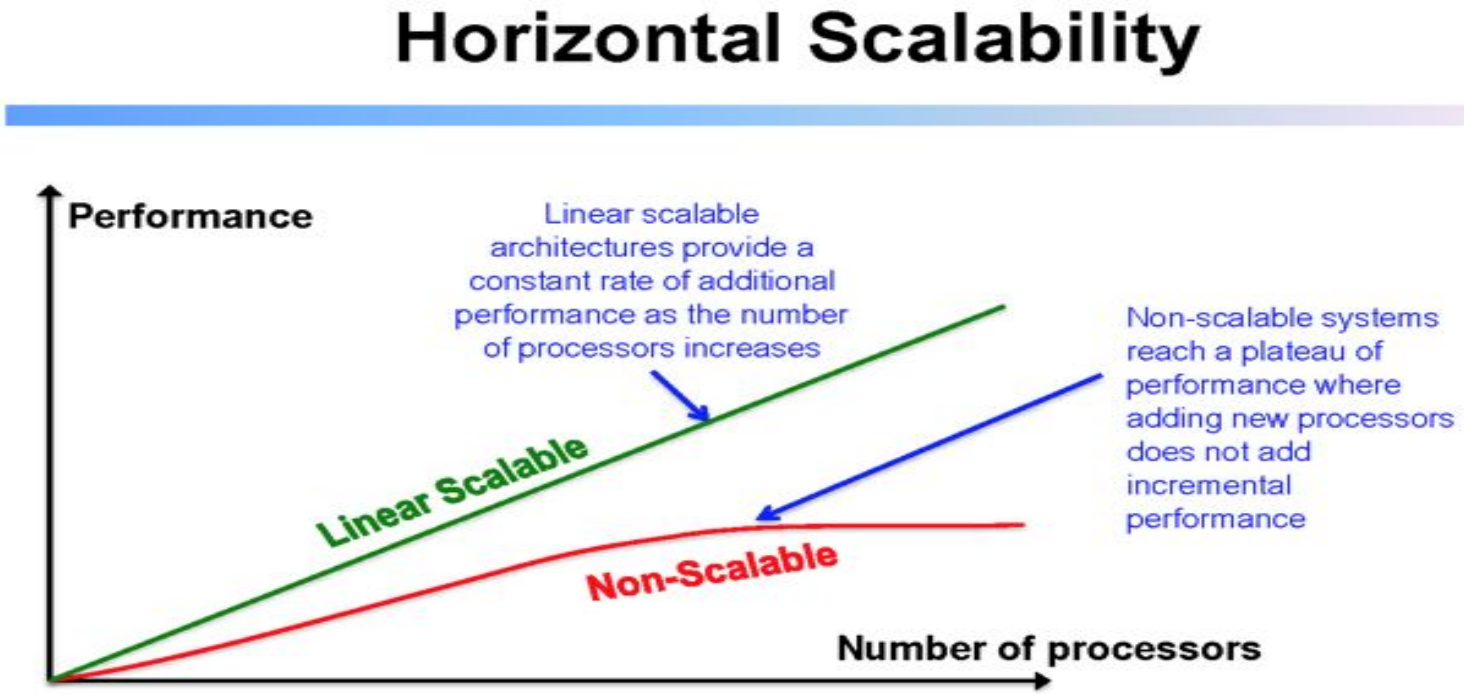


Figure 6.2

Background: SQL vs. NoSQL

- NoSQL is naturally fit for big data.

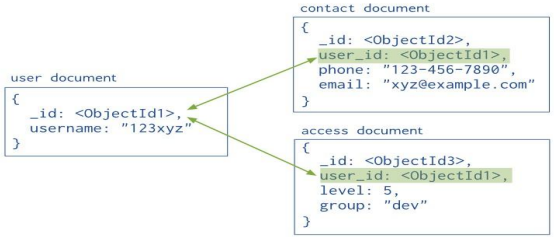
-Unstructured data with similar semantics but varied syntax

-Large volume of data for which scalability is becoming a must and consistency expensive

Background: A refresh on MongoDB

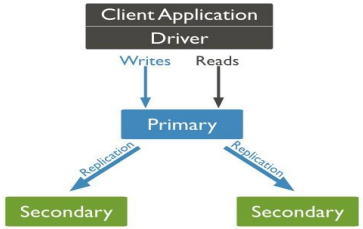
- Use collections to organize modules

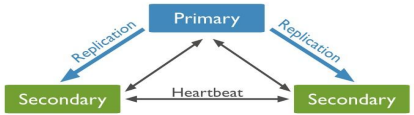
- Normalized (Reference) or denormalized (embedding)

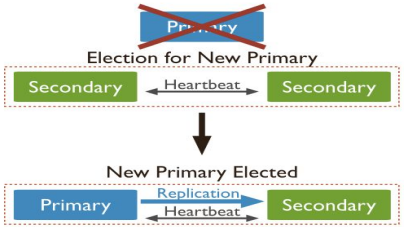


or

- Strict consistency (All writes must go to primary node)







CouchDB

A NoSQL DBMS that does not mimic SQL

Introduction: What is CouchDB?

- Name comes from:
  - Cluster Of Unreliable Commodity Hardware
  - Relax (in a couch)
- Written in Erlang, initial release in 2005
- Licence: Apache, Original author: Damien Katz, et al.

Introduction: What is CouchDB?

- An open source, document-oriented, NoSQL database that uses JSON to store data, JavaScript as its query language, and HTTP for an API.

- Instead of locking mechanism, CouchDB uses Multi-version Concurrency Control (MVCC) to resolve conflicts, and incremental replication to achieve eventual consistency.

Introduction: Why CouchDB?

- Availability, Locality and Scalability

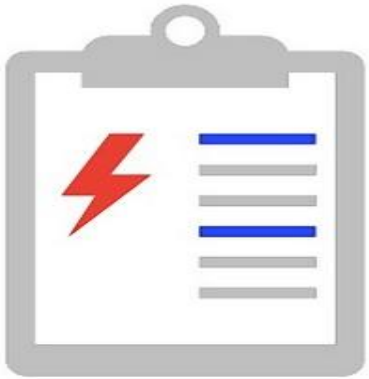
> Each node in a system should be able to make decisions purely based on local state. If you need to do something under high load with failures occurring and you need to reach agreement, you're lost. If you're concerned about scalability, any algorithm that forces you to run agreement will eventually become your bottleneck. Take that as a given.
>
> —Werner Vogels, Amazon CTO and Vice President

- "A database that completely embraces the web."

Data Modeling of CouchDB: JSON Format

- CouchDB: JSON
- MongoDB: BSON

BSON is binary JSON



BSON is a JSON that has been serialized as a binary document.

Data Modeling of CouchDB: Self-contained Data

- CouchDB:
  purely self-contained
  (Say Goodbye to SQL)

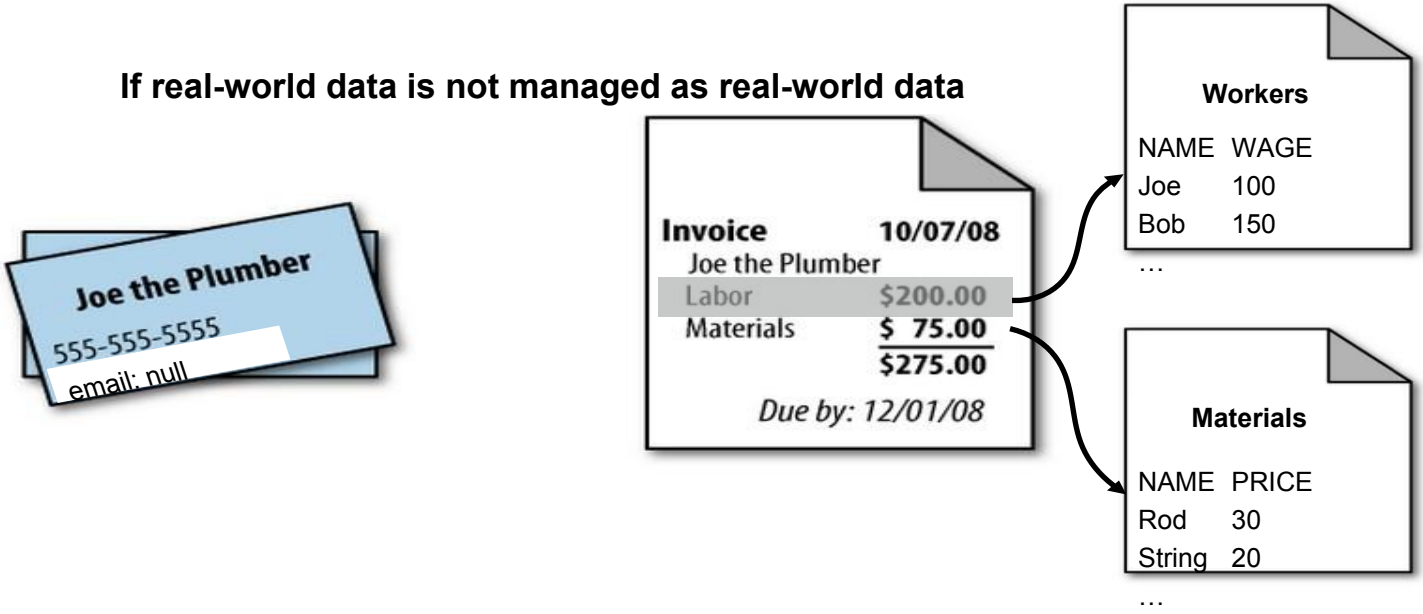
- MongoDB: embedded
  (*NoSQL*);
  or referenced (*SQL-like*)

Data Modeling of CouchDB: Self-contained Data

- CouchDB:
  purely self-contained
  (Say Goodbye to SQL)

- MongoDB: embedded
  (*NoSQL*);
  or referenced (*SQL-like*)

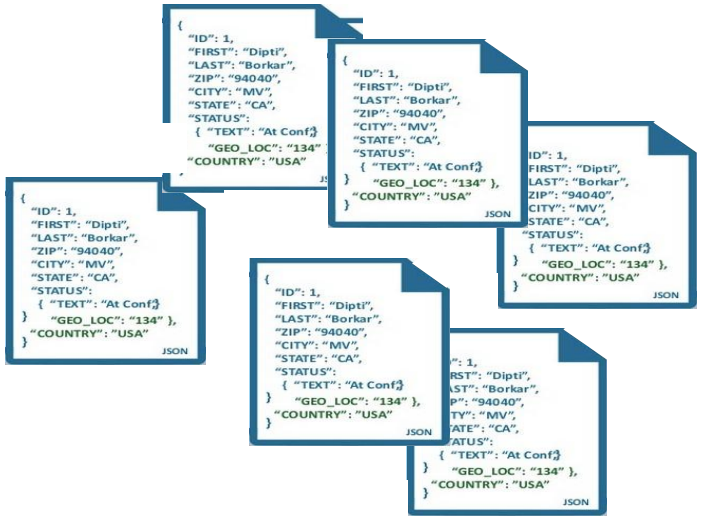
If real-world data is not managed as real-world data



Data Modeling of CouchDB: Data Storage

- CouchDB: one big warehouse

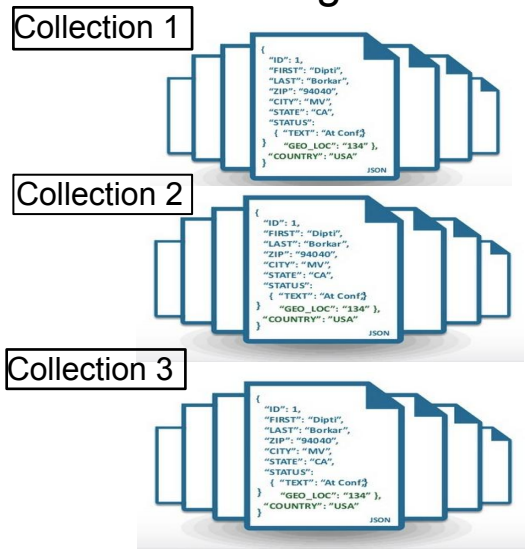
  No global indexes predefined on DB level, create a view to report results instead



MongoDB:

separated by collections

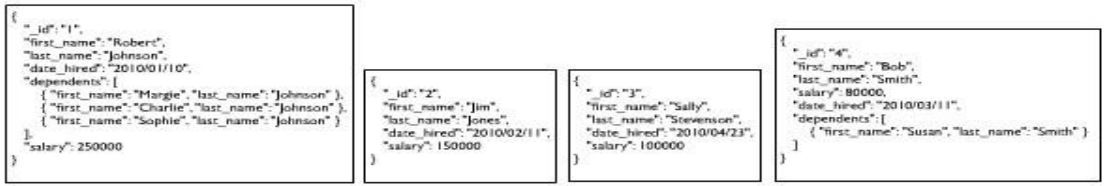
Can create index for any field of documents in a collection (identical to indexing in RDBMS)



Query Capabilities: How do you aggregate unstructured data?

- Define a view

  - Map takes documents and emits key/value pairs



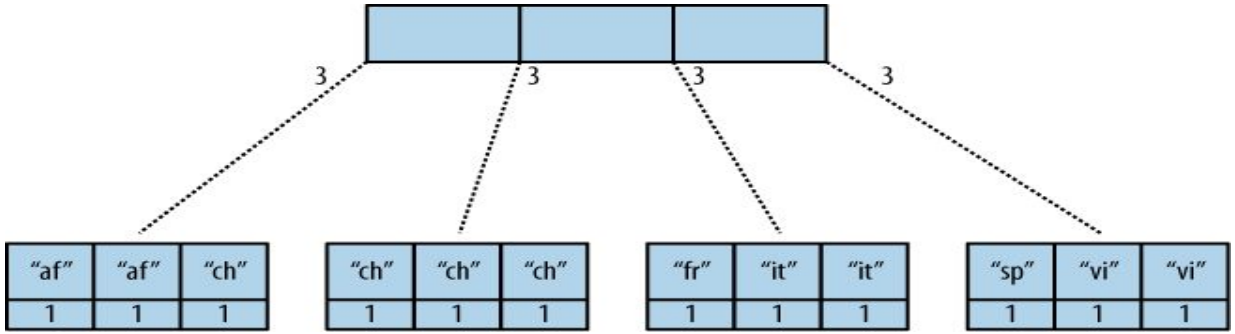
MapReduce

{"total_rows":4,"offset":0,"rows":[
{"id":"1","key":"1","value":{"first_name":"Margie","last_name":"Johnson"}},
{"id":"1","key":"1","value":{"first_name":"Charlie","last_name":"Johnson"}},
{"id":"1","key":"1","value":{"first_name":"Sophie","last_name":"Johnson"}},
{"id":"4","key":"4","value":{"first_name":"Susan","last_name":"Smith"}}
]}

Query Capabilities: How do you aggregate unstructured data?

- Construct B-tree index

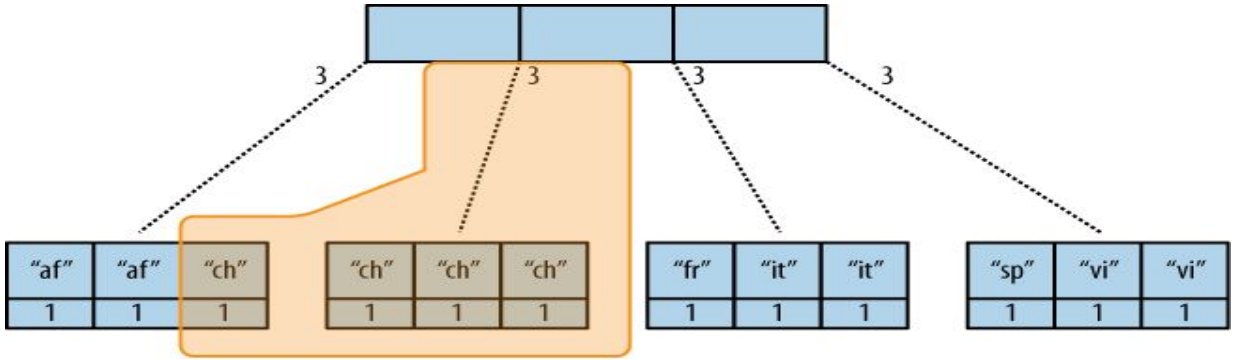
  - CouchDB storage engine constructs a B-tree index



Query Capabilities: How do you aggregate unstructured data?

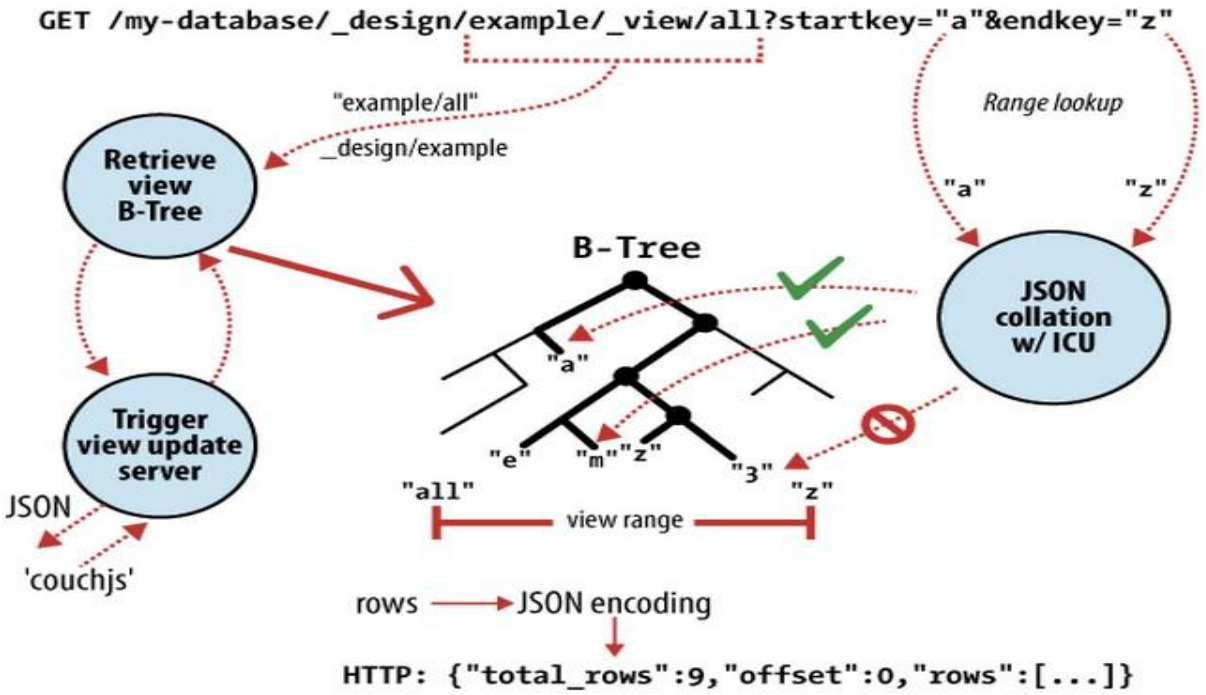
- Query the view

  - Reduce operates on the subtree to do aggregation



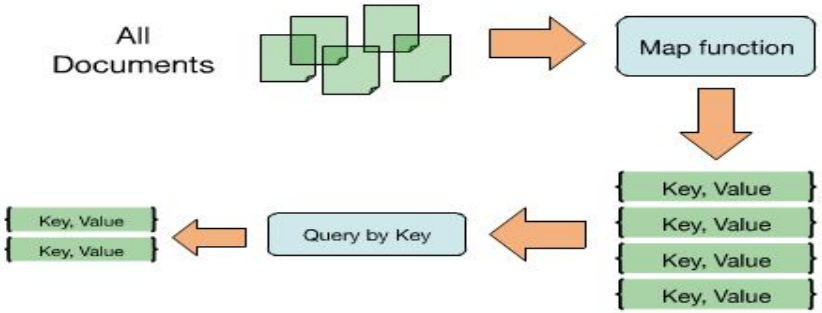
Query Capabilities: How do you aggregate unstructured data?

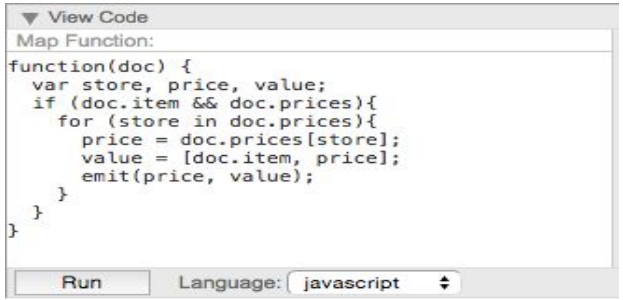
- MapReduce + B-tree = results of a view



Query Capabilities

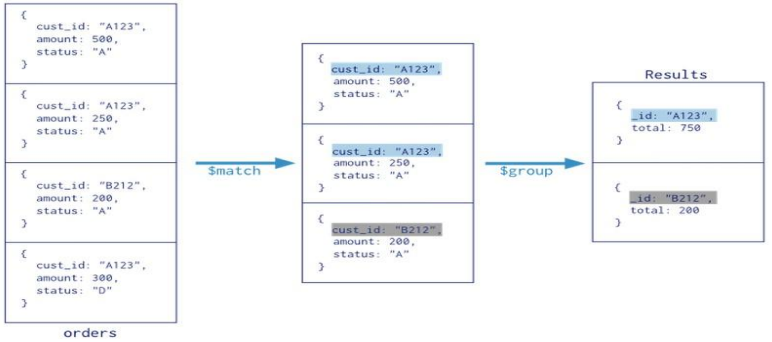
- CouchDB:
  MapReduce(complex queries)



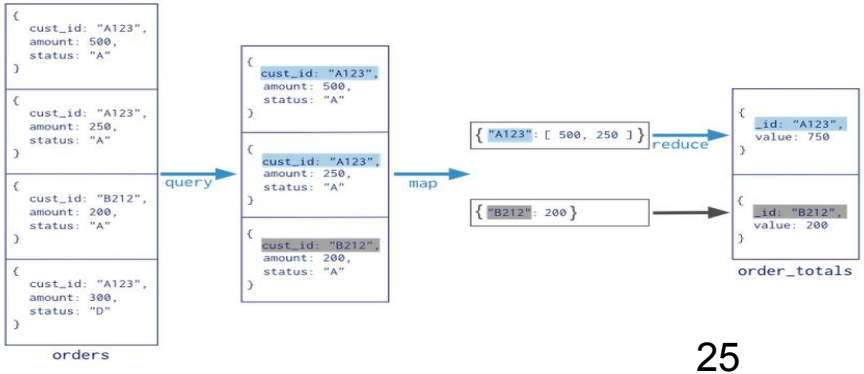


Well, comparatively complex…

- MongoDB:
  (1) Aggregation pipeline(SQL-like)



(2) MapReduce(complex queries)



Data Management

- REST API: a thin wrapper around the DB core

REST API

```
# Create
POST http://localhost:5984/employees

# Read
GET http://localhost:5984/employees/1

# Update
PUT http://localhost:5984/employees/1

# Delete
DELETE http://localhost:5984/employees/1
```

Data Management

- REST API: a thin wrapper around the DB core

Welcome:

```
pocoyang: ~ $: curl http://127.0.0.1:5984/
{"couchdb":"Welcome","uuid":"bd94a3f857e93302522f918d997cb706","version":"1.6.1"
,"vendor":{"version":"1.6.1-1","name":"Homebrew"}}
```

Add a new database:

```
pocoyang: ~ $: curl -X PUT http://127.0.0.1:5984/albums
{"ok":true}
```

Add a new document:

```
pocoyang: ~ $: curl -X PUT http://127.0.0.1:5984/albums/a688d8d20e17b5e87e47da6a
a8004eaa -d '{"title":"D Minor K466", "artist":"Mozart"}'
{"ok":true,"id":"a688d8d20e17b5e87e47da6aa8004eaa","rev":"1-d067700c88a3a78e5863
970ccad4f923"}
```

Get a new UUID:

(if don't have one)

```
pocoyang: ~ $: curl -X GET http://127.0.0.1:5984/_uuids
{"uuids":["a688d8d20e17b5e87e47da6aa8004eaa"]}
```

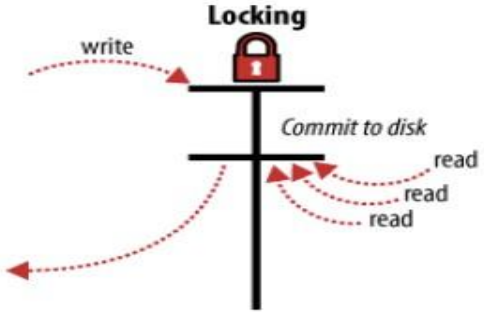
Read a document:

```
pocoyang: ~ $: curl -X GET http://127.0.0.1:5984/albums/a688d8d20e17b5e87e47da6a
a8004eaa
{"_id":"a688d8d20e17b5e87e47da6aa8004eaa","_rev":"1-d067700c88a3a78e5863970ccad4
f923","title":"D Minor K466","artist":"Mozart"}
```

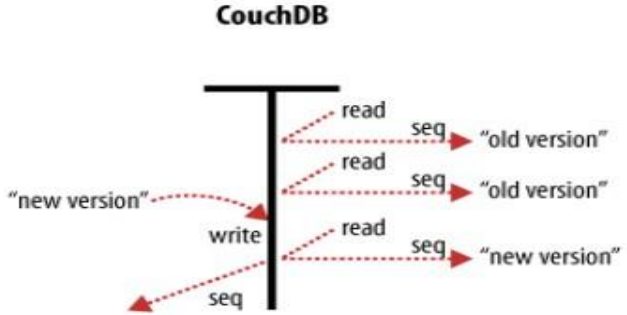
... ...

Concurrency control of CouchDB

- Multi-Version Concurrency Control:
  - Doesn't rely on global state, always available to readers;
  - Each reader is reading the latest visible snapshot



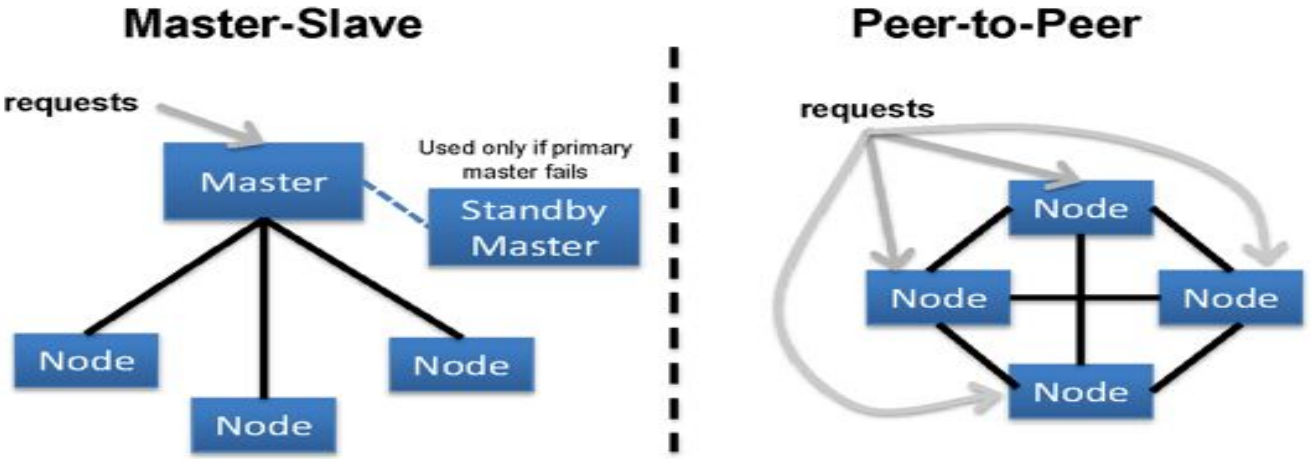
- MongoDB



- CouchDB

Distributed Architecture of CouchDB

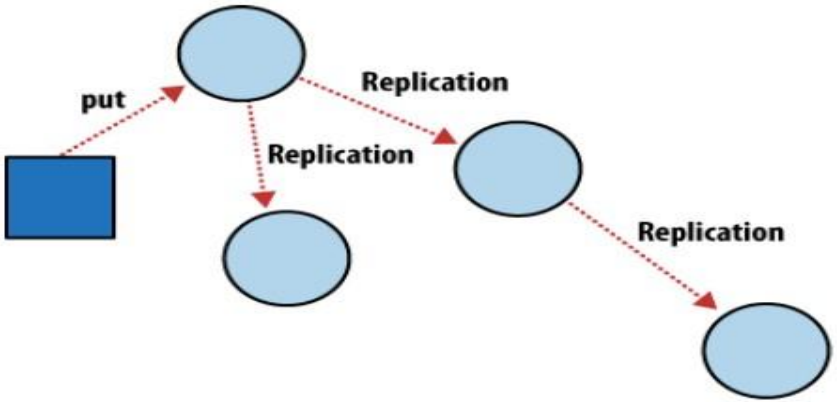
Master-Slave vs. Peer to Peer



- The Master node may become a bottleneck in large clusters
- Many newer NoSQL architectures are moving toward a true peer-to-peer system

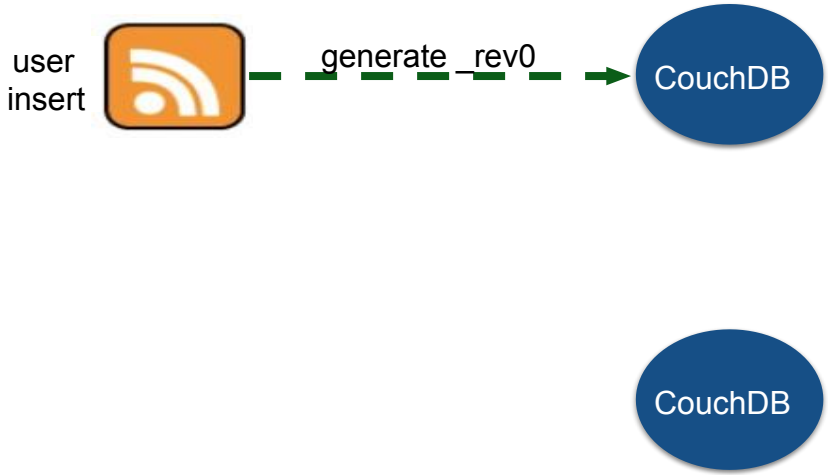
Distributed Architecture of CouchDB

- Eventual consistency by incremental replication:
  - Peer-to-peer rather than primary-secondary



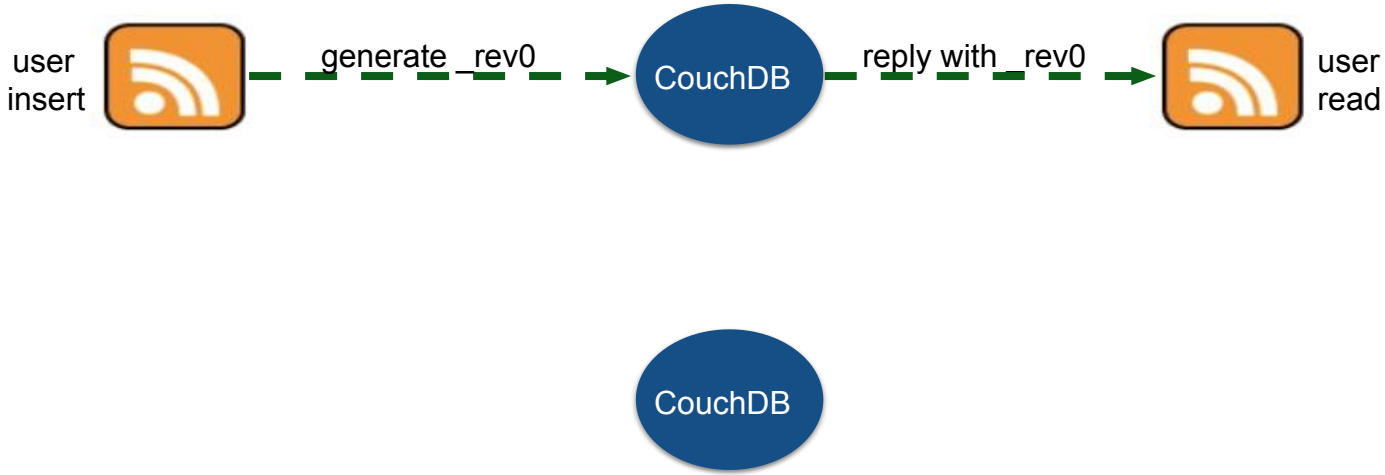
Distributed Architecture of CouchDB

- Eventual consistency by incremental replication:
  - Peer-to-peer rather than primary-secondary
  - Sites can go offline, DB will handle sync when back online



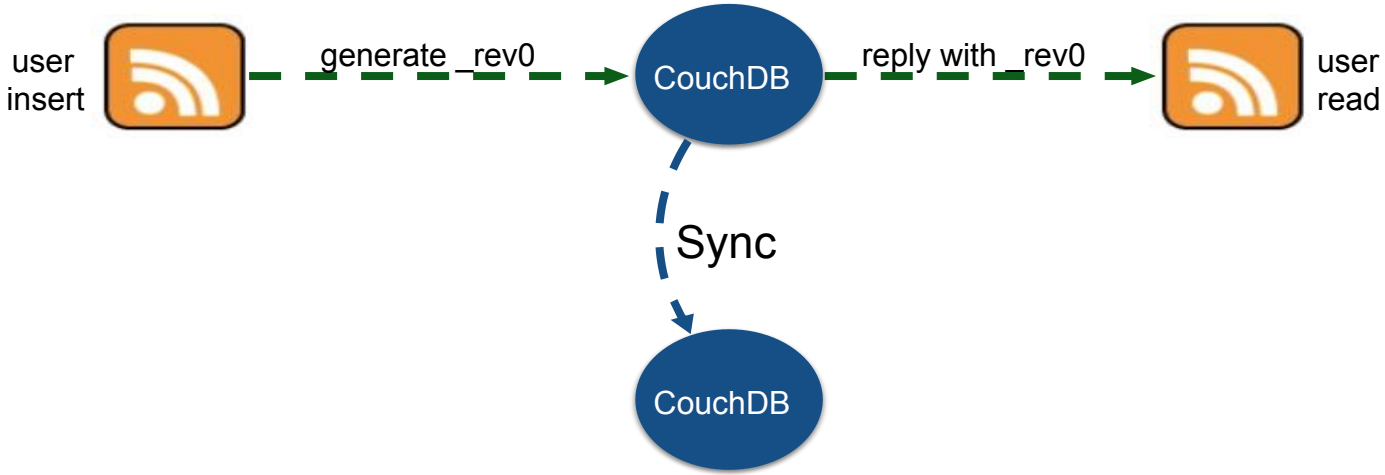
Distributed Architecture of CouchDB

- Eventual consistency by incremental replication:
  - Peer-to-peer rather than primary-secondary
  - Sites can go offline, DB will handle sync when back online



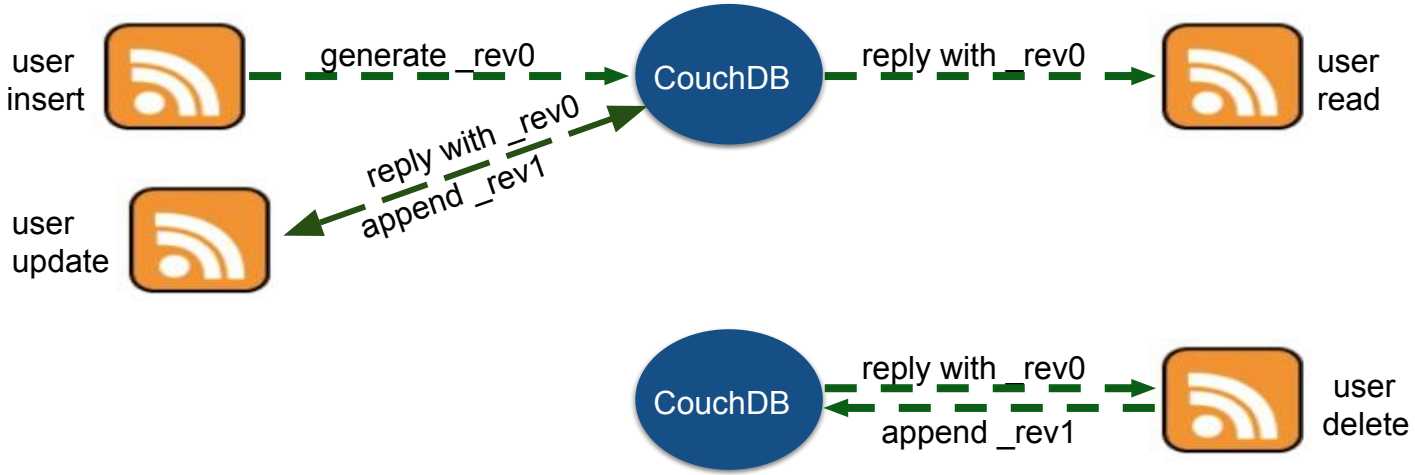
Distributed Architecture of CouchDB

- Eventual consistency by incremental replication:
  - Peer-to-peer rather than primary-secondary
  - Sites can go offline, DB will handle sync when back online



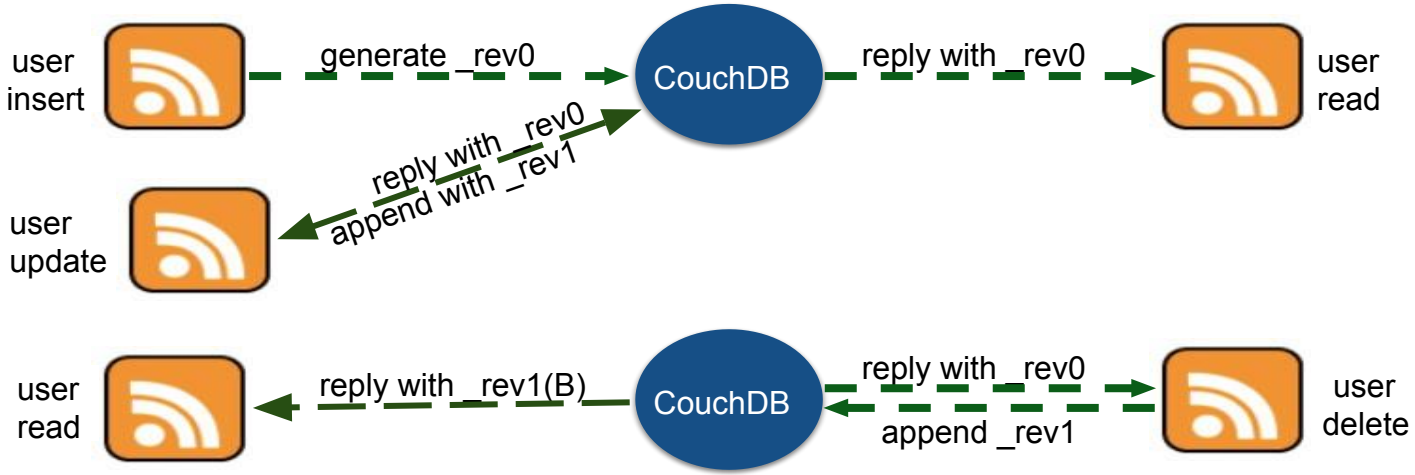
Distributed Architecture of CouchDB

- Eventual consistency by incremental replication:
  - Peer-to-peer rather than primary-secondary
  - Sites can go offline, DB will handle sync when back online
  - Automatic conflict detection and resolution



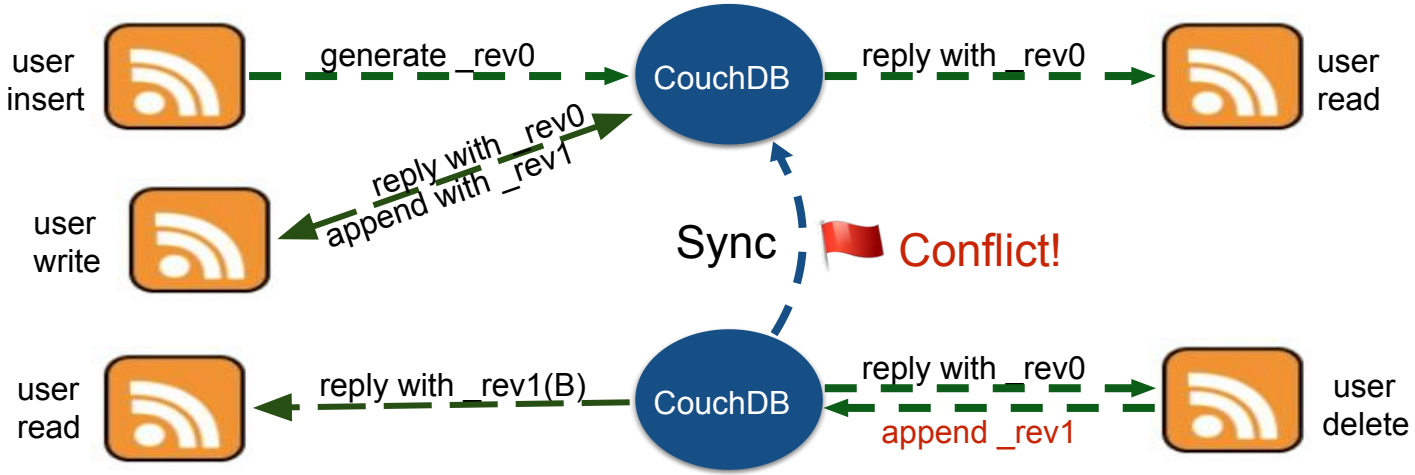
Distributed Architecture of CouchDB

- Eventual consistency by incremental replication:
  - Peer-to-peer rather than primary-secondary
  - Sites can go offline, DB will handle sync when back online
  - Automatic conflict detection and resolution



Distributed Architecture of CouchDB

- Eventual consistency by incremental replication:
  - Peer-to-peer rather than primary-secondary
  - Sites can go offline, DB will handle sync when back online
  - Automatic conflict detection and resolution



Conclusions

	MongoDB	CouchDB
Focus	**Consistency**	**Availability**
Distributed architecture	Primary-Secondary replication	Peer-Peer synchronization
Concurrency control	Update in-place (much like SQL)	MVCC
Document format	BSON	JSON
Data storage	Referenced or embedded	Self-contained
Data organization	One extra layer: collections	Everything piled together
Query capabilities	Aggregation pipeline or MapReduce	MapReduce views and indexes
CRUD syntax	SQL-like	HTTP methods

When to use what?

- You have some predefined queries upfront, want to run on occasionally changing data;
- Need to make sure that sites are always available, even if data center crashes;
- Need to replicate data bi-directionally between 2 or more data centers;
- If versioning is important;
- You are familiar with HTTP but not SQL;
- You are a geek and you believe RDBMS is outdated.

When to use what?

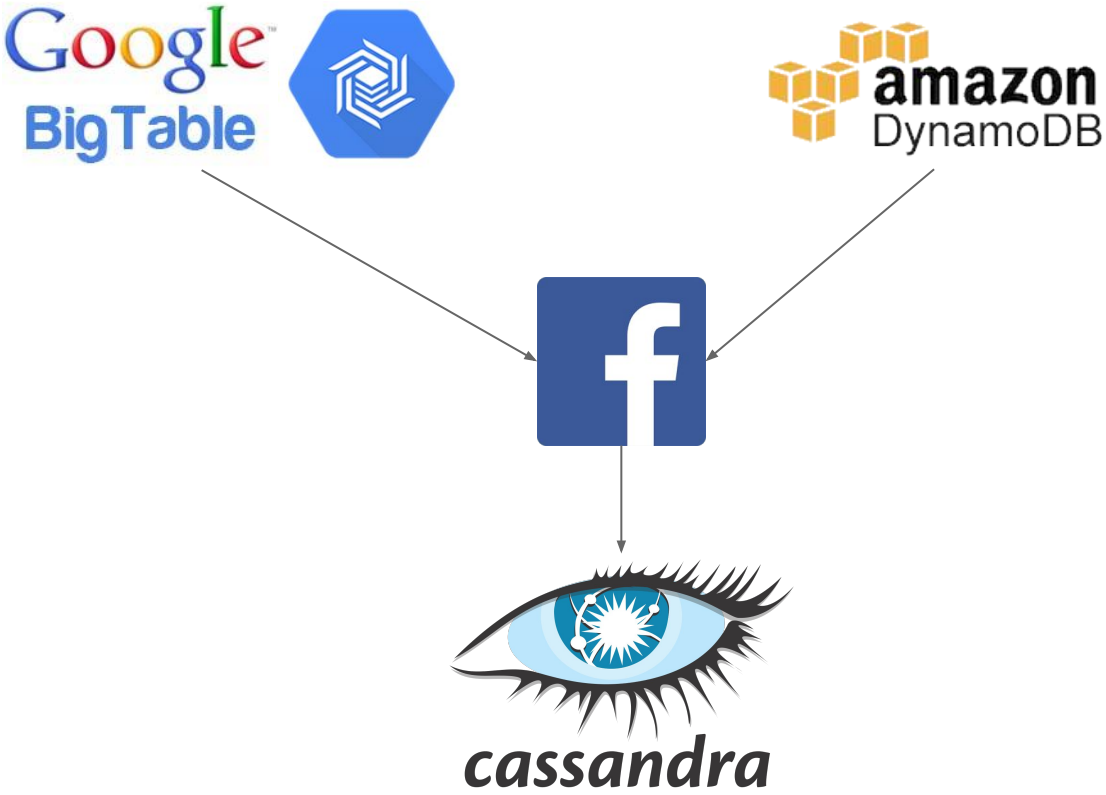
- All other cases when you need a distributed DBMS

Cassandra

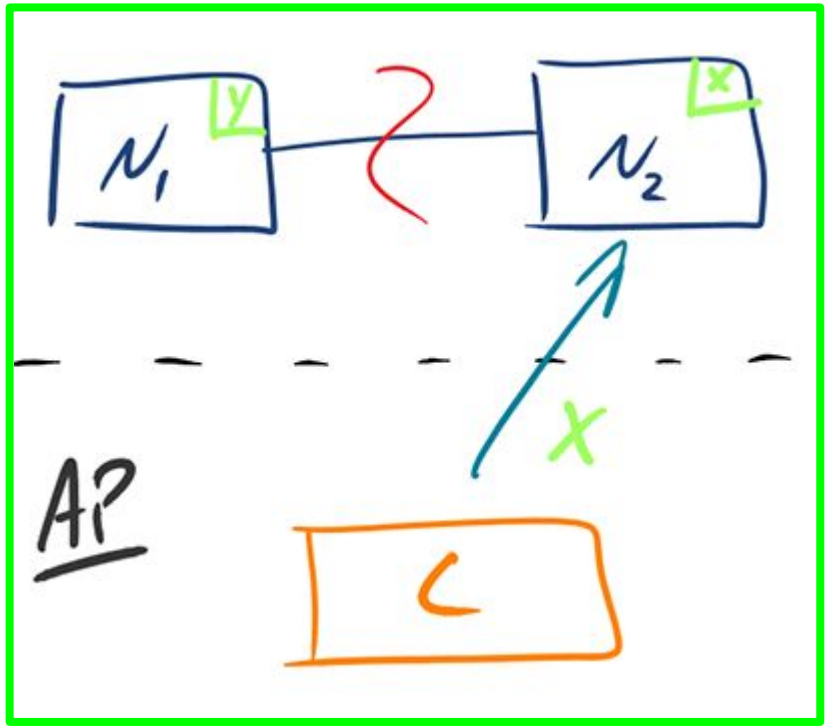
Overview

- Apache Cassandra is a free
  - Distributed
  - Performant
  - Scalable
  - Fault tolerant
- NoSQL Databases

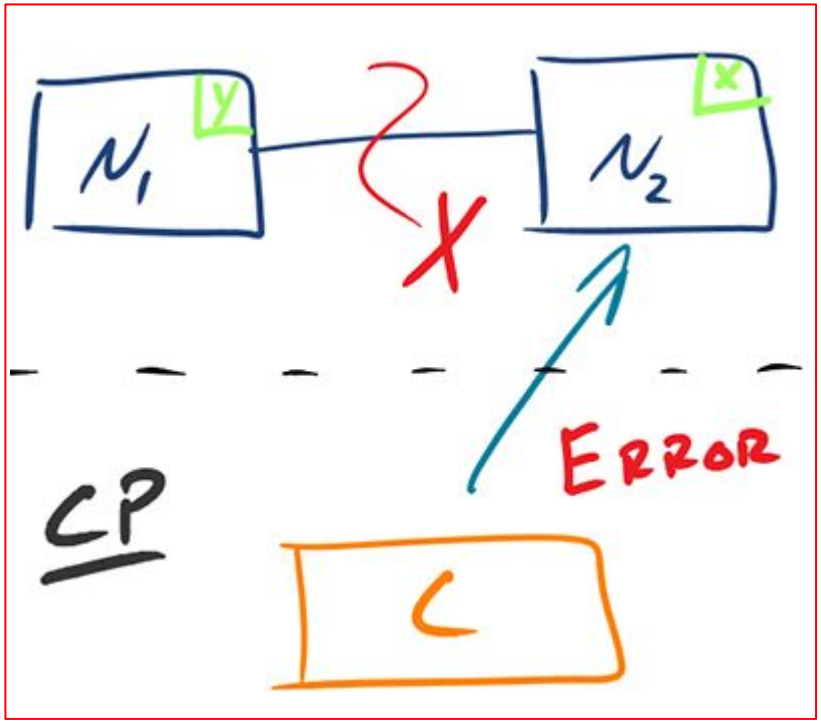
History



CAP Decision

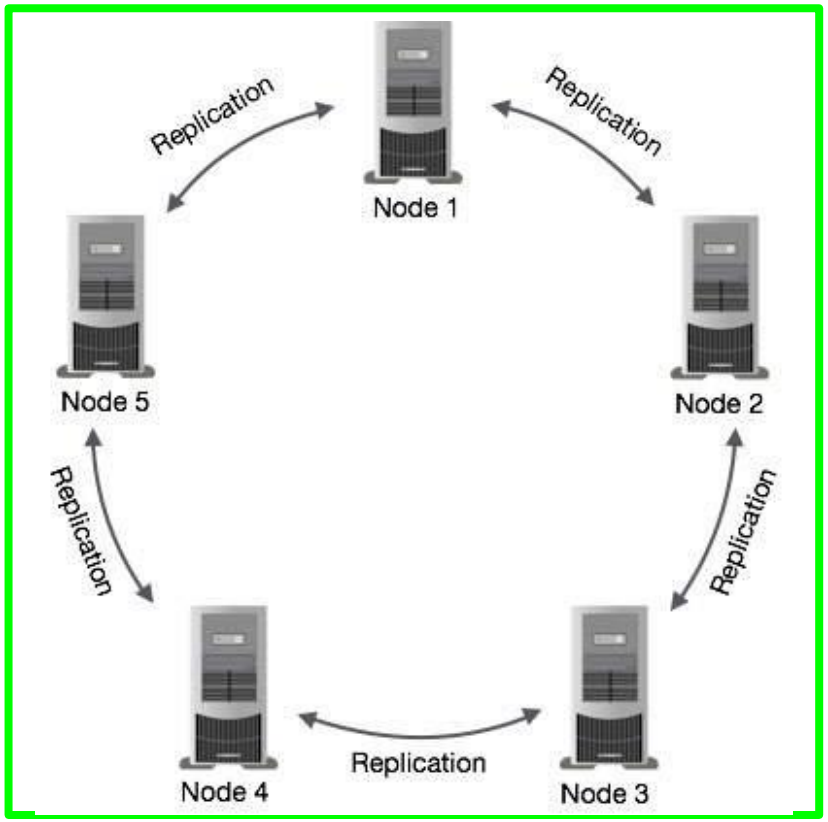


http://robertgreiner.com/uploads/images/2014/CAP-AP.png

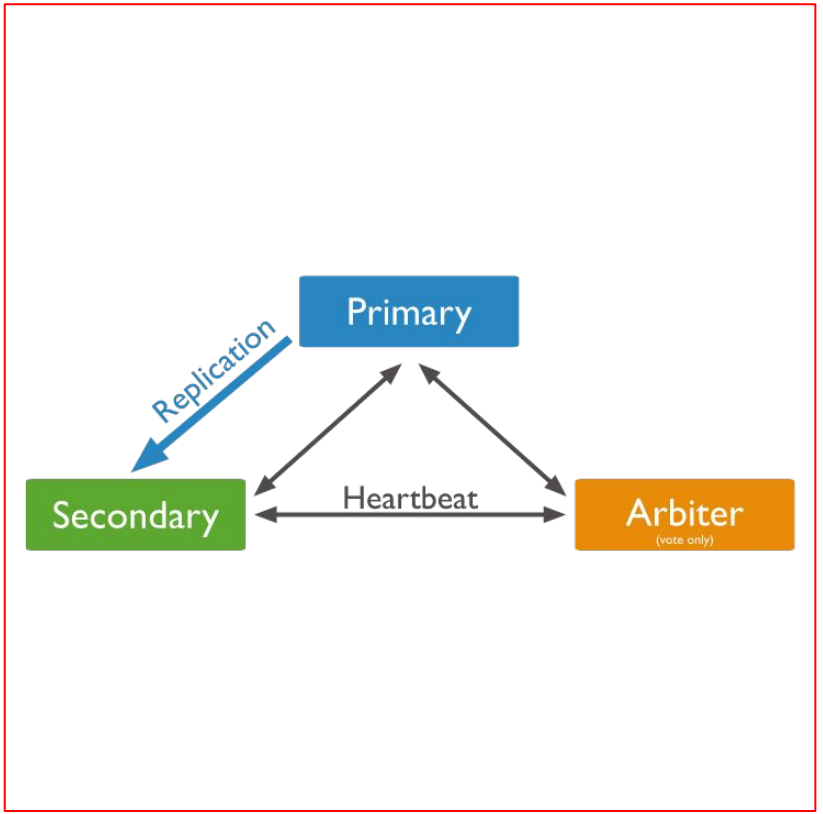


http://robertgreiner.com/uploads/images/2014/CAP-CP.png

Architecture Decision

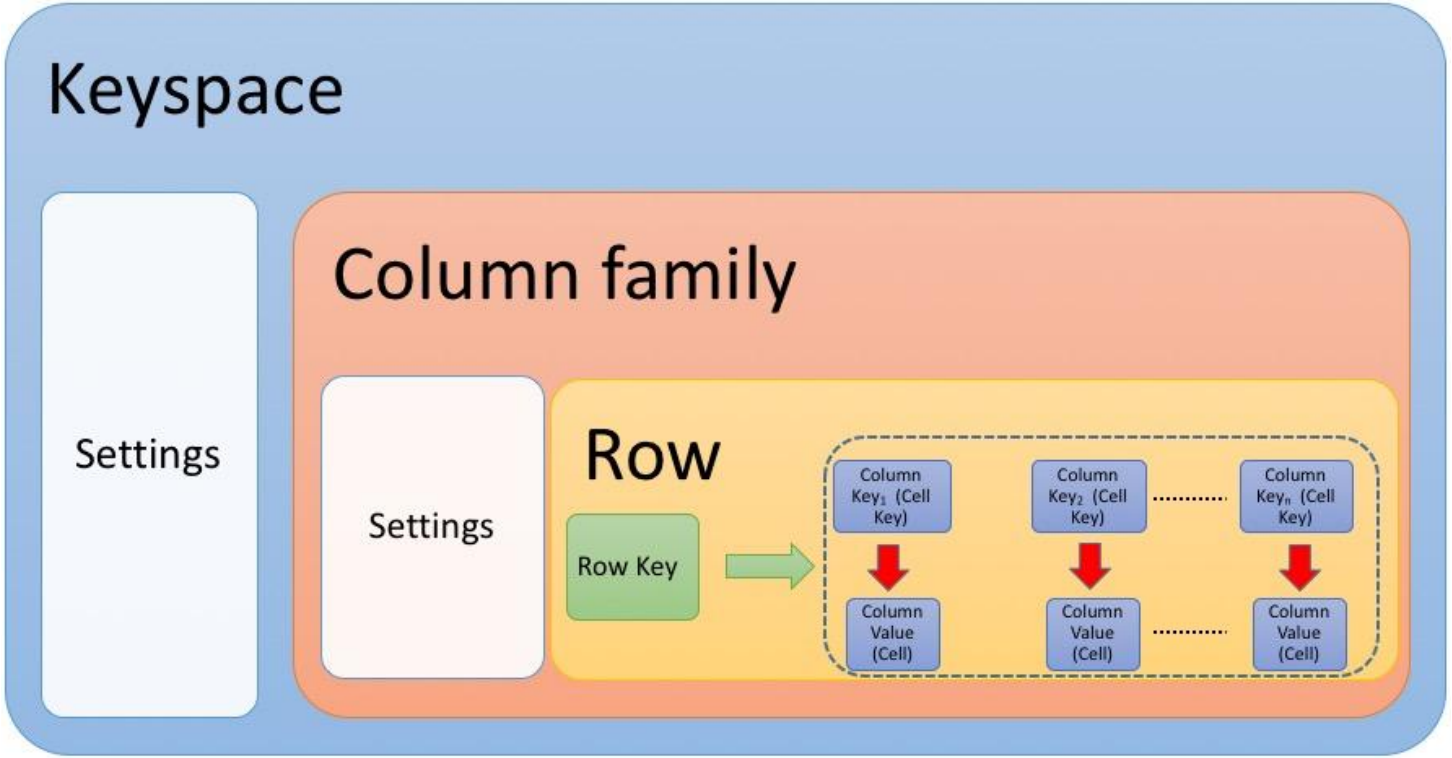


https://www.tutorialspoint.com/cassandra/images/data_replication.jpg

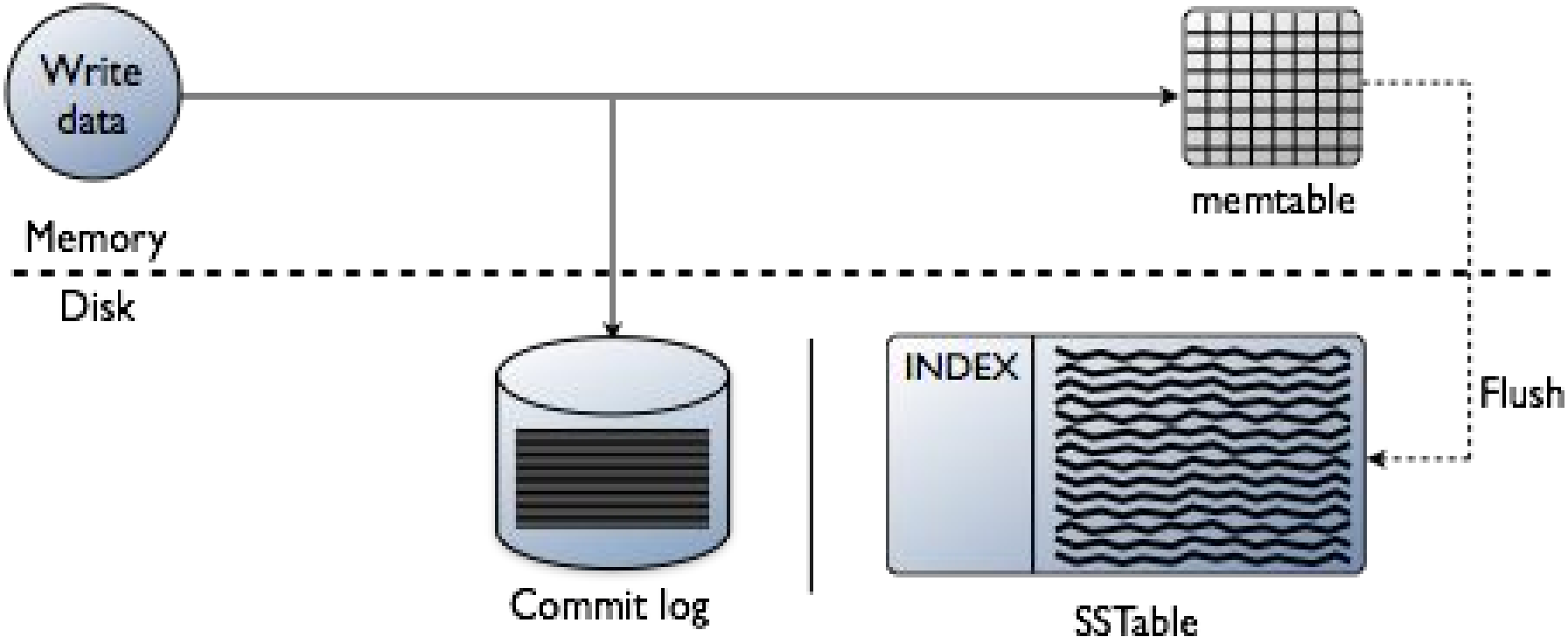


https://docs.mongodb.com/manual/replication/

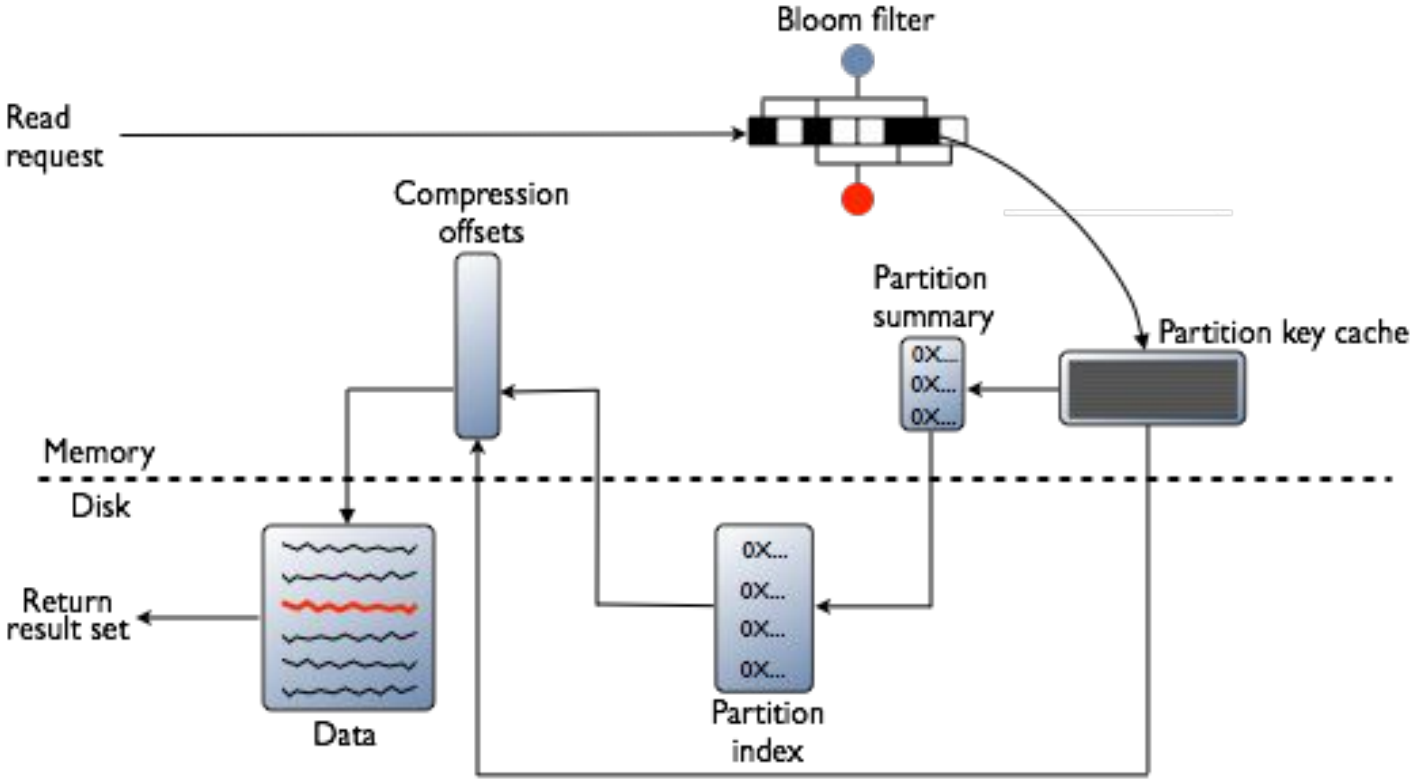
Data Model


Keyspace
Settings
Column family
Settings
Row
Row Key
Column Key1 (Cell Key)
Column Key2 (Cell Key)
Column Keyn (Cell Key)
Column Value (Cell)
Column Value (Cell)
Column Value (Cell)

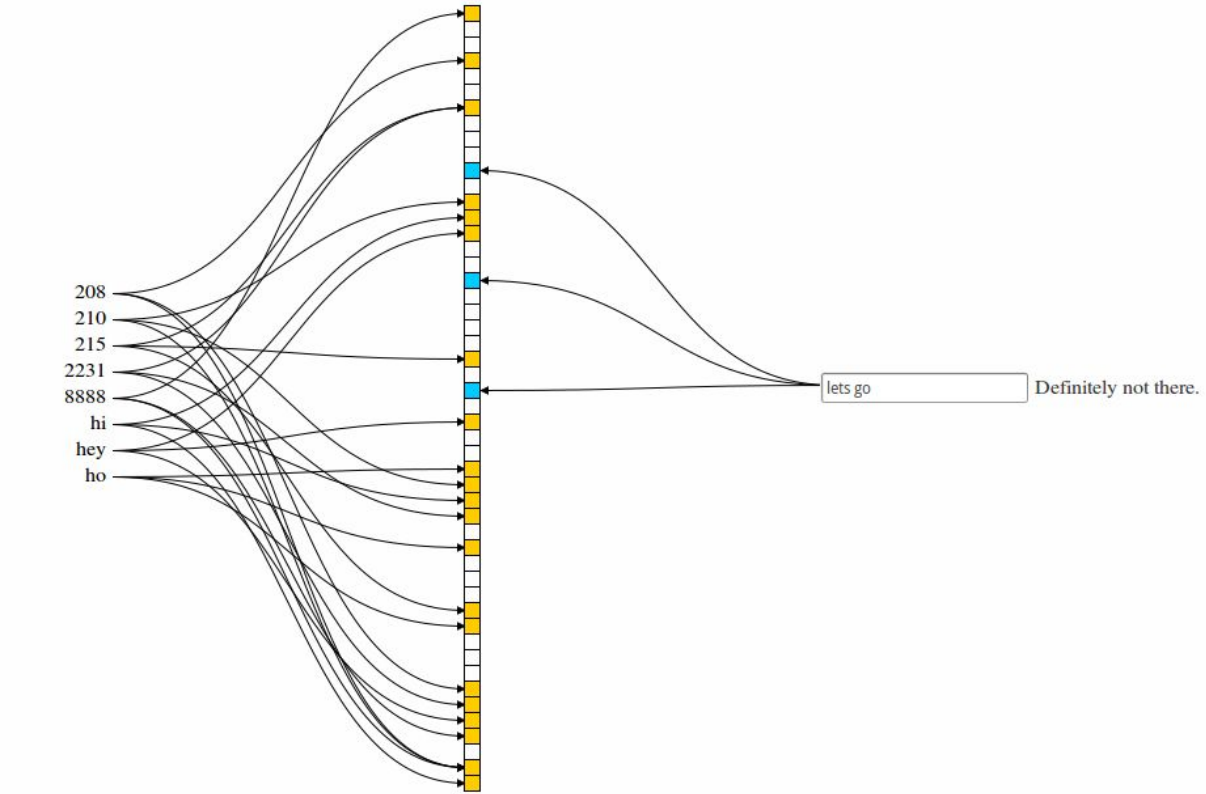
Writes


Write data
memtable
Memory
Disk
Flush
INDEX
Commit log
SSTable

Reads


Bloom filter
Read request
Compression offsets
Partition summary
0X...
0X...
0X...
Partition key cache
Memory
Disk
Return result set
0X...
0X...
0X...
0X...
Data
Partition index

Bloom Filter - Example



Query Model

- CQL
  - Data Definition
  - Data Manipulation
  - Secondary Indexes
  - Materialized Views
  - Data Security
  - Aggregate and User-defined functions
  - JSON Support
  - Triggers

Real Life

75,000 Nodes
10 PB

Apple

https://upload.wikimedia.org/wikipedia/commons/a/a5/Apple_gray_logo.png

2500 Nodes
420 TB
1 Trillion Daily Requests

Netflix

http://cdn.wegotthiscovered.com/wp-content/uploads/netflix-logo.jpg

>100 Nodes
250 TB

eBay

https://upload.wikimedia.org/wikipedia/commons/4/48/EBay_logo.png

Constant Contact
CERN
Comcast
GoDaddy
Hulu
Instagram
Intuit
Reddit
The Weather Channel
...

REFERENCES

CouchDB

- CouchDB: The Definitive Guide
- http://couchdb.apache.org
- https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling- techniques/
- http://blog.scottlogic.com/2014/08/04/mongodb-vs-couchdb.html
- http://openmymind.net/2011/10/27/A-MongoDB-Guy-Learns-CouchDB/

Cassandra

- https://docs.datastax.com/en/cassandra/3.0/index.html
- http://cassandra.apache.org/
- Abramova, Veronika, and Jorge Bernardino. "NoSQL databases: MongoDB vs cassandra." *Proceedings of the international C* conference on computer science and software engineering*. ACM, 2013.