SCHOOL OF COMPUTING
THE UNIVERSITY OF UTAH

Klemen Simonic
Robert Christensen

NoSQL: How to Manage Big Data?

Scenario

Size of data >> disk or memory space on a single machine

Store data across many machines

Retrieve data from many machines

Machine = Commodity machine

Facebook 2009

Facebook storage system includes 4000 MySQL servers

Database servers not enough for large data demand from the users

Facebook adds additional 2000 memcached servers:

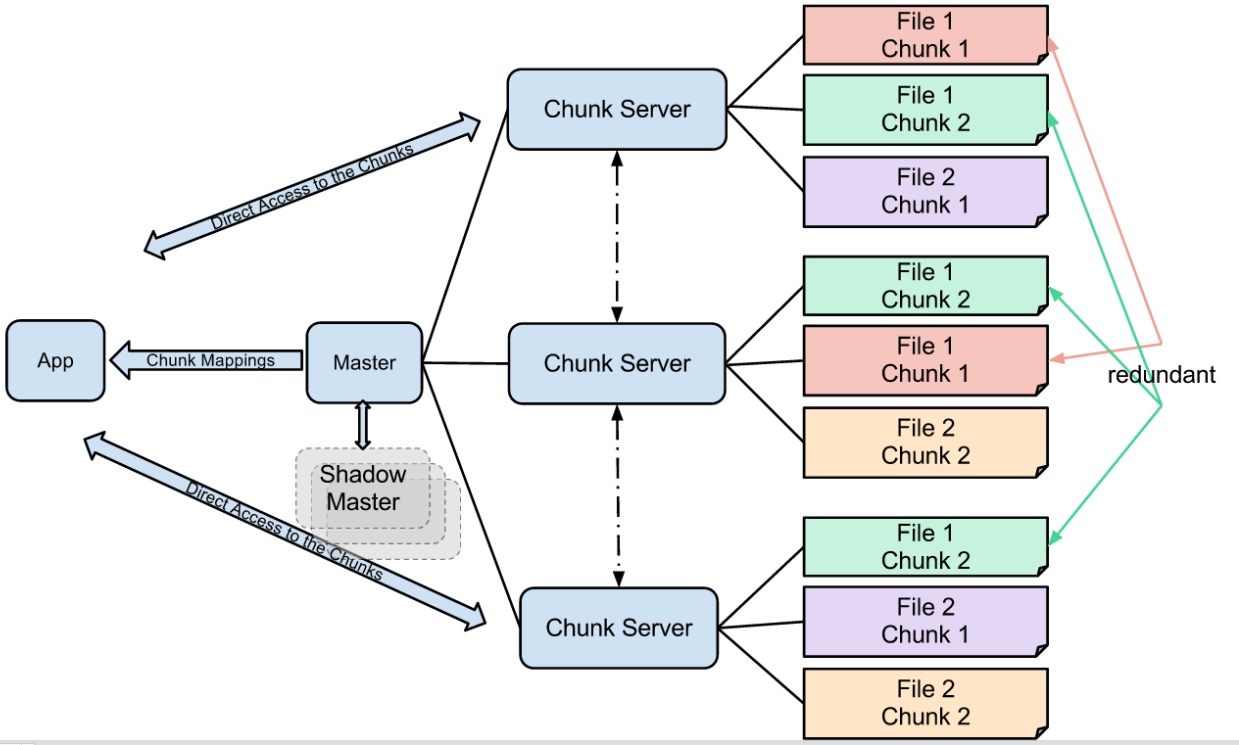
cache recently used query results in key-value stores kept in memory

Example of a cluster of commodity machines:

#servers	1000
RAM Capacity / Server	64 GB
Total RAM Capacity	64 TB

Each machine has a few TBs of disk space

Distributed File System



- File system stored across many commodity hardware
- Fault tolerant (hardware failure is a standard)
- High throughput access (streaming access) and NOT low latency data access
- Simple coherency access model: Write-once-read many
- Write Model: Permission is granted to a process; modifies the primary chunk and other chunks; acknowledge from all the chunk servers (GFS)
- Enables to store large dataset

Hypertable

Overview

- **Database system** based on Google's Big Table
- Runs on top of DFS (e.g. Hadoop, GlusterFS)
- Maximum performance
- Scalability
- Written in C++
- Comprehensive Language Support
  (Java, PHP, Python, Perl, Ruby, C++, …)
- Ease of Use
- Good Documentation

Companies / Organizations

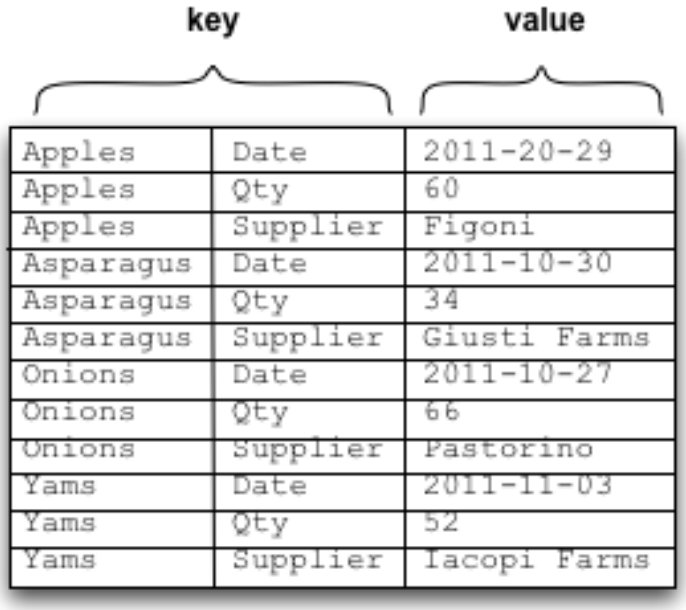
- Baidu
- eBay
- IBM
- Yelps
- Groupon
- Insparx
- …

Physical Layout

RDBMS

Item	Date	Qty	Supplier
Apples	2011-20-29	60	Figoni
Asparagus	2011-10-30	34	Giusti Farms
Bananas	\N	\N	\N
Cantelope	\N	\N	\N
Grapes	\N	\N	\N
Onions	2011-10-27	66	Pastorino
Oranges	\N	\N	\N
Peaches	\N	\N	\N
Pears	\N	\N	\N
Pineapples	\N	\N	\N
Plums	\N	\N	\N
Strawberries	\N	\N	\N
Yams	2011-11-03	52	Iacopi Farms

Hypertable



Apples	Date	2011-20-29
Apples	Qty	60
Apples	Supplier	Figoni
Asparagus	Date	2011-10-30
Asparagus	Qty	34
Asparagus	Supplier	Giusti Farms
Onions	Date	2011-10-27
Onions	Qty	66
Onions	Supplier	Pastorino
Yams	Date	2011-11-03
Yams	Qty	52
Yams	Supplier	Iacopi Farms

Flattens out the table structure into a **sorted list of key/value** pairs, each one representing a **cell** in the table.

Cells that are NULL are simply not included (good for **sparse data**)

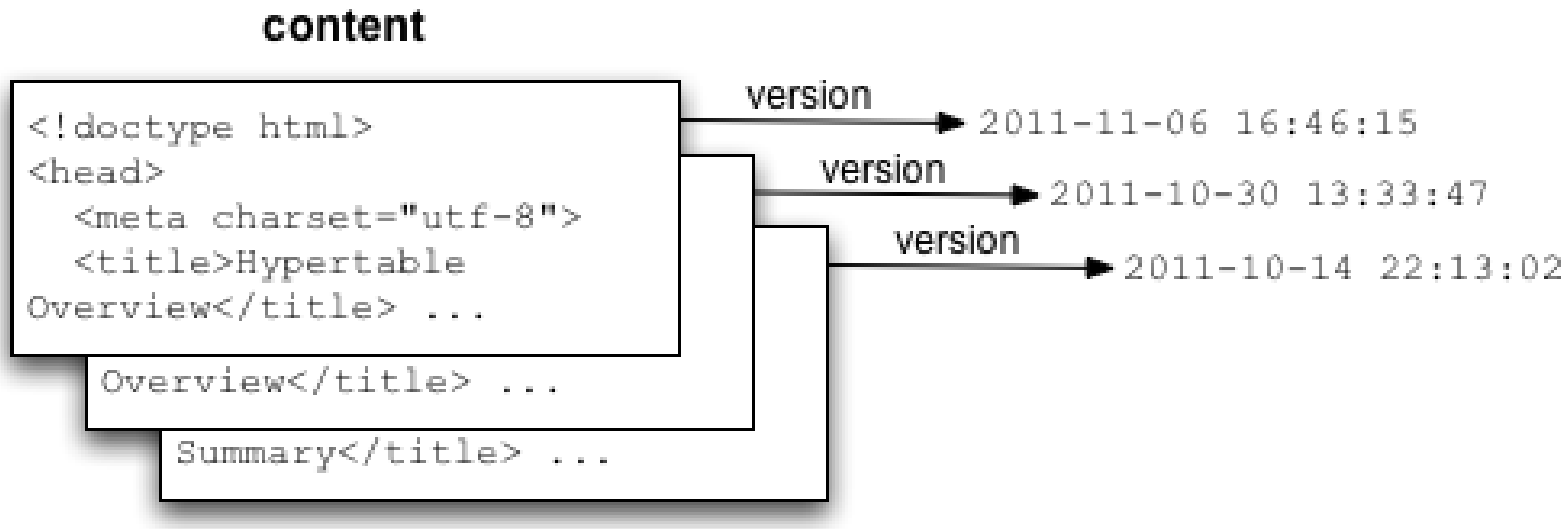
Redundancy in the row keys and column identifiers is minimal due to the block data compression.

Cell Versions

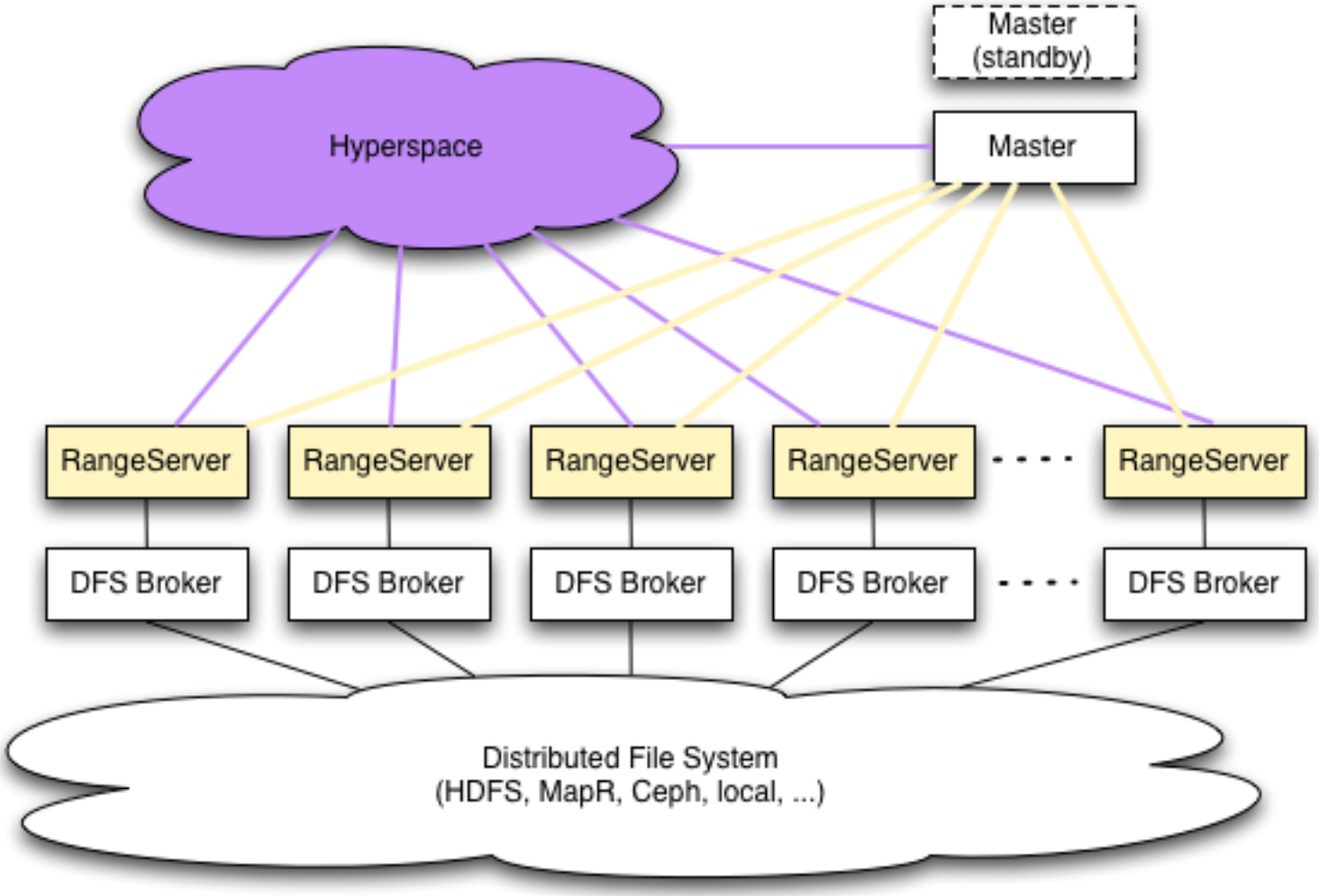
Extends the traditional two-dimensional table model by adding a third dimension: **timestamp**

Representing different versions of each table cell.

Specified by MAX_VERSIONS parameter.

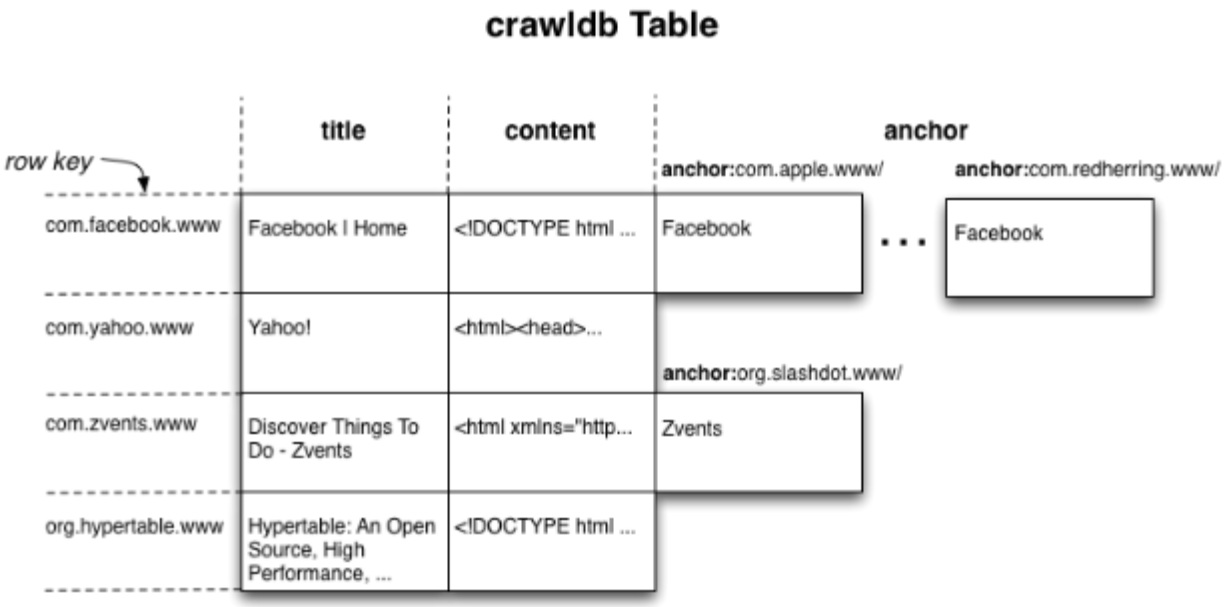


System Overview



Data Modelling: Logical View

Each cell is identified by a **row key** and **column name**



Two additional features:

- **Timestamp**
- **Column qualifier**

  A column actually defines a set of related columns knows as a column family

  family:qualifier

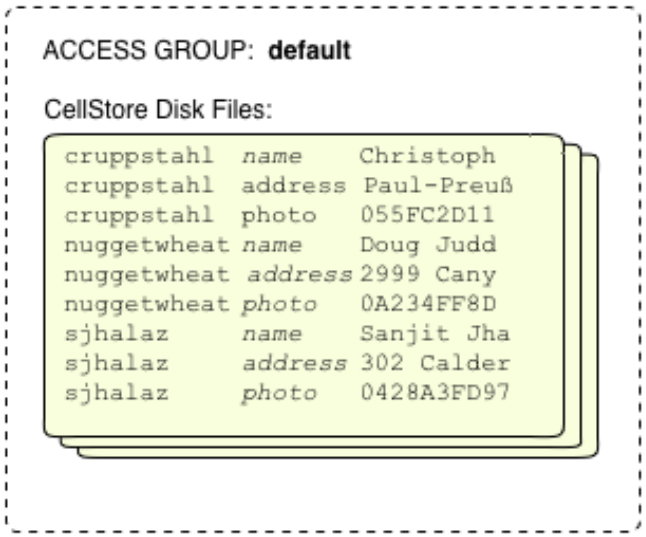
Data Modelling: Physical View

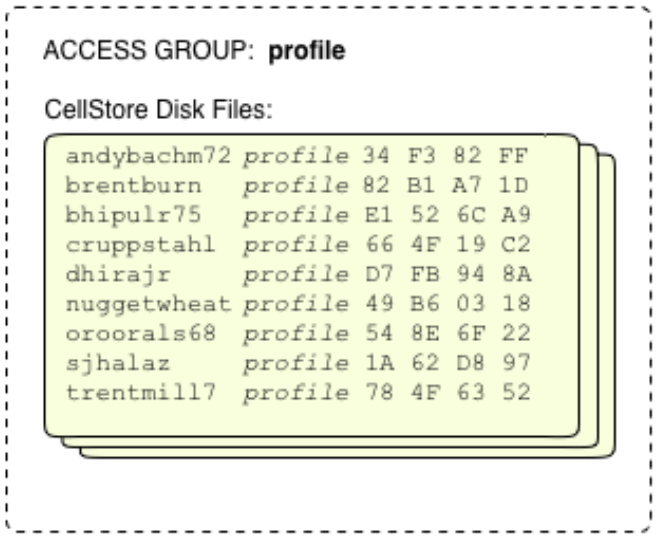
crawldb Table

key	value
com.facebook.www title *2008-02-11 15:14:01*	Facebook	Home
com.facebook.www title *2008-02-03 19:27:57*	Facebook	Home
com.facebook.www title *2008-01-22 08:46:28*	Facebook	Home
com.facebook.www content *2008-02-11 15:14:01*	<!DOCTYPE html PUBLIC "-//W3C//DTD...
com.facebook.www content *2008-02-03 19:27:57*	<!DOCTYPE html PUBLIC "-//W3C//DTD...
com.facebook.www content *2008-01-22 08:46:28*	<!DOCTYPE html PUBLIC "-//W3C//DTD...
com.facebook.www anchor:com.apple.www/ *2008-02-11 15:14:01*	Facebook
com.facebook.www anchor:com.apple.www/ *2008-02-03 19:27:57*	Facebook
com.facebook.www anchor:com.apple.www/ *2008-01-22 08:46:28*	Facebook
com.facebook.www anchor:com.redherring.www/ *2008-02-11 15:14:01*	Facebook
com.facebook.www anchor:com.redherring.www/ *2008-02-03 19:27:57*	Facebook
com.yahoo.www title *2008-02-10 21:12:09*	Yahoo!
com.yahoo.www title *2008-02-04 03:46:22*	Yahoo!
com.yahoo.www title *2008-01-22 08:46:28*	Yahoo!
com.yahoo.www content *2008-02-10 21:12:09*	<html><head><meta http-equiv="Content-...
com.yahoo.www content *2008-02-04 03:46:22*	<html><head><meta http-equiv="Content-...
...	...

Access Groups

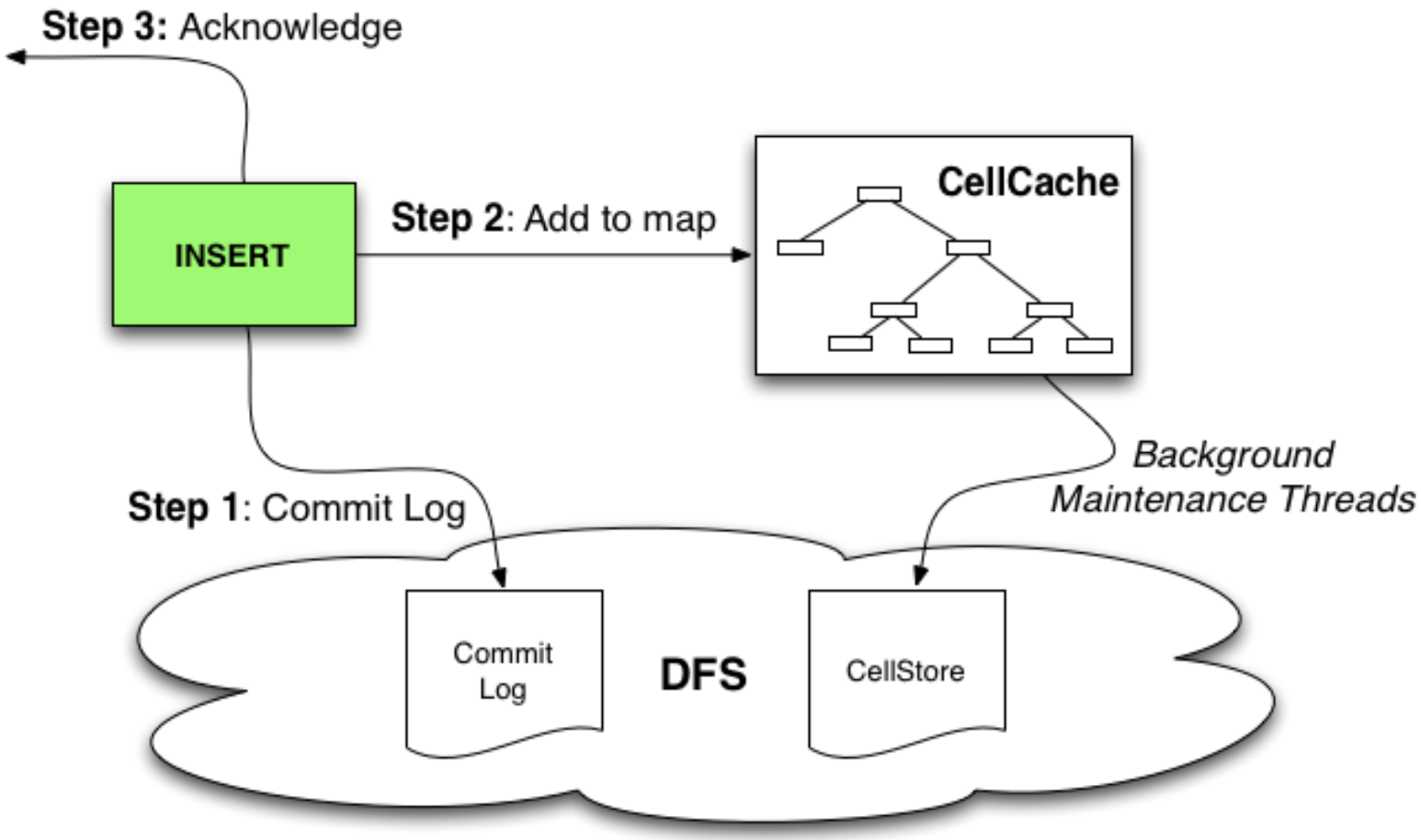
```
CREATE TABLE User (
  name,
  address,
  photo,
  profile,
  ACCESS GROUP default (name, address, photo),
  ACCESS GROUP profile (profile)
);
```



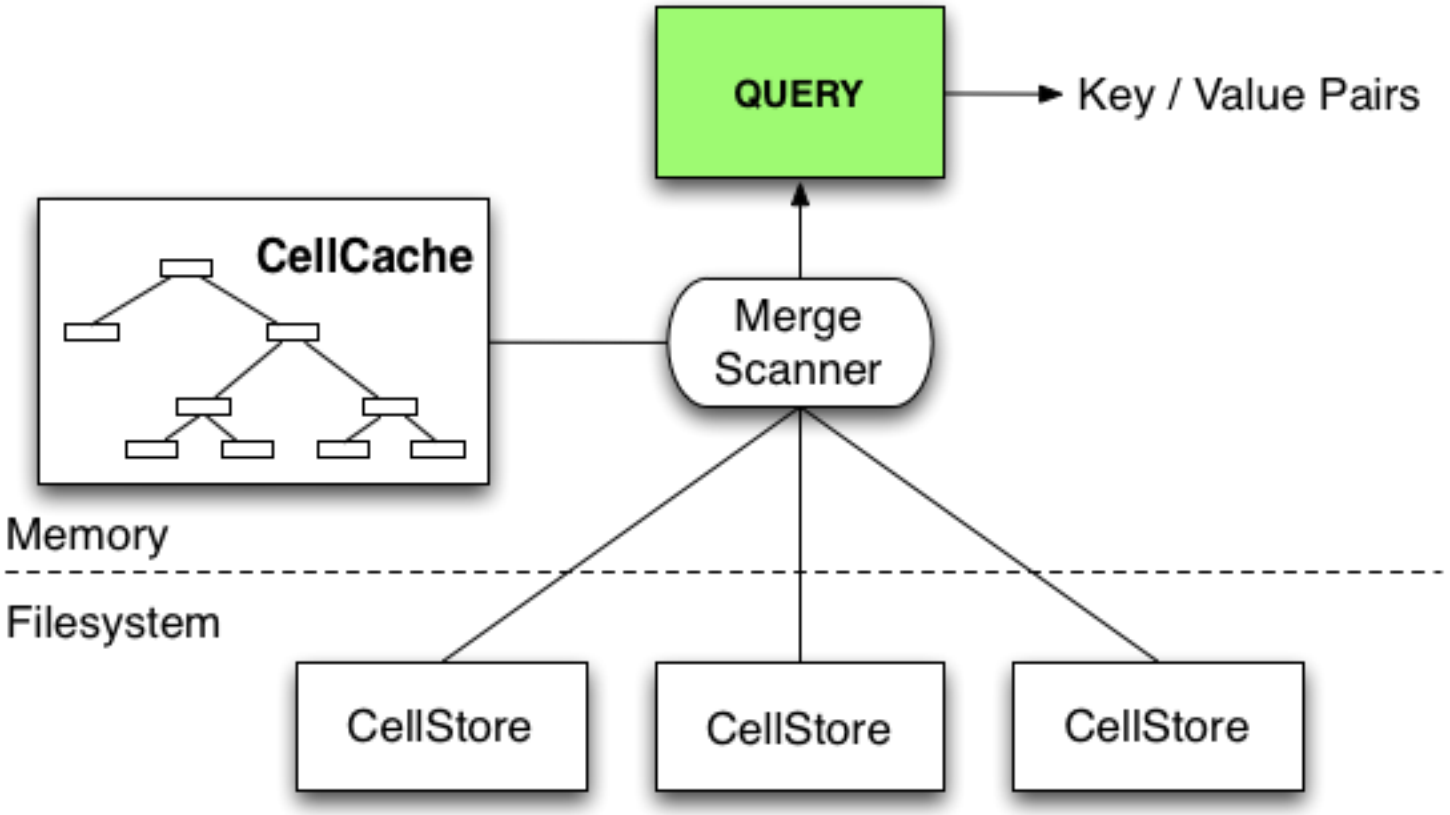


SELECT profile from User;

Range Server: Insert



Range Server: Query



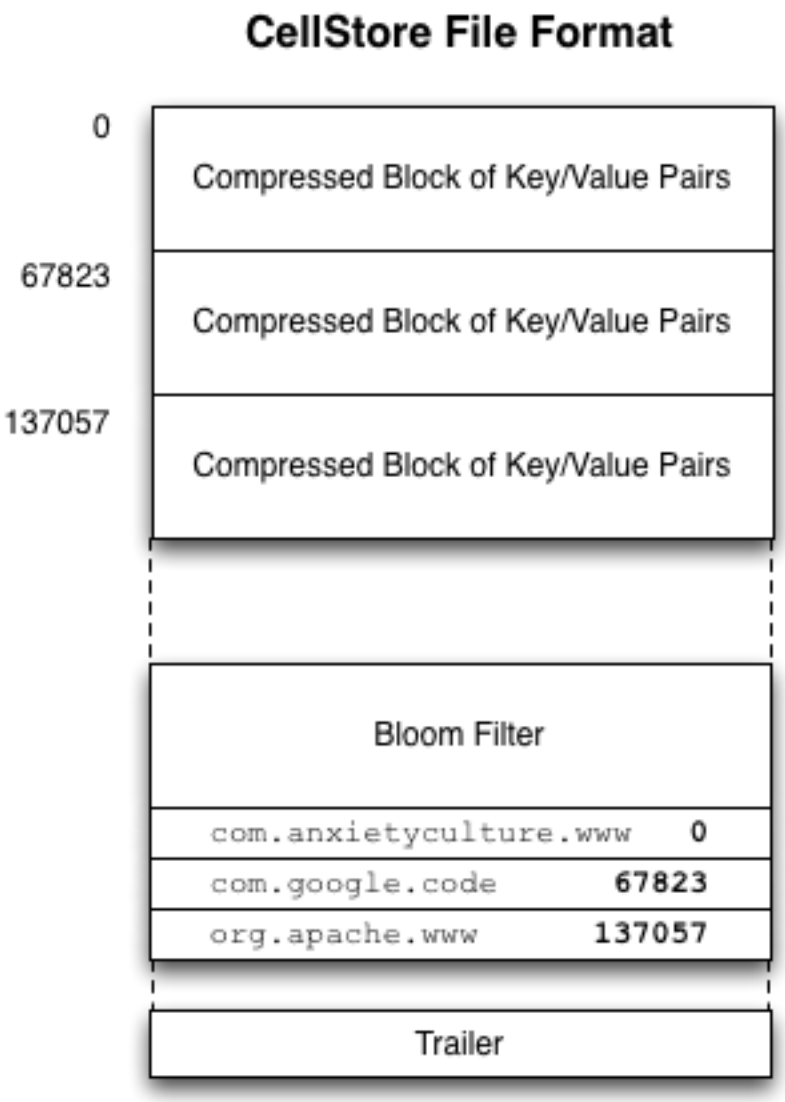
Cell Store Format

Compressed blocks of cells:
Series of sorted blocks of compressed sorted key/value pairs.

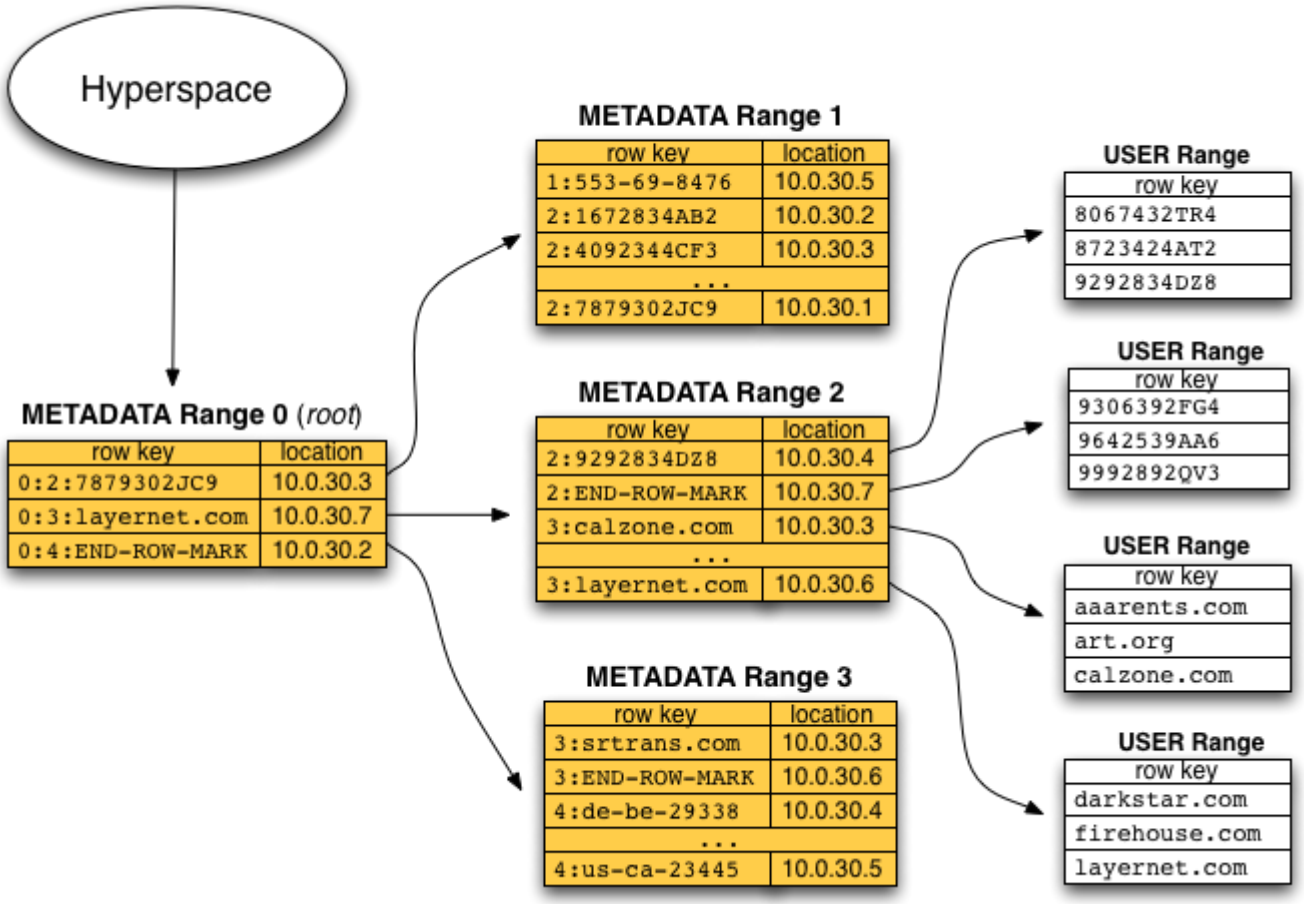
Bloom Filter:
Describes the keys that exist (with high likelihood) in the CellStore.

Stores the information if a key is not present

Avoid unnecessary block transfer and decompression.



Query Routing



METADATA table that contains a row for each range in the system.
Two-level hierarchy is overlaid on top of the METADATA table

=> **Client library includes a METADATA cache** (avoid walking through METADATA)

MongoDB

Overview

- Document-Oriented database system
  - JSON-style documents
- Full Index Support
- Replication & High Availability
  - Mirror access across LAN or WAN.
- Auto-Sharding
- Fast In-Place Updates
- GridFS file
  - specialized data storage optimized for large documents
- Implemented in C++

NoSQL

- Document Model
  - **MongoDB**
  - CouchDB
- Graph Model
  - Neo4j
  - HyperGraphDB
- Key-Value/Wide Column Models
  - Riak
  - Redis
  - HBase
  - Cassandra

Partners

- Cloud Partners
  - Amazon Web Services
  - Windows Azure
  - VMware
- Hardware Partners
  - Fusion-IO
  - Intel

Companies using MongoDB

- CERN
  - primary back-end for Data Aggregation System
- Forbes
  - articles and company data
- Shutterfly
  - photo storage platform with 7 million users
- Foursquare
  - store venue and user check-ins
- Source Forge

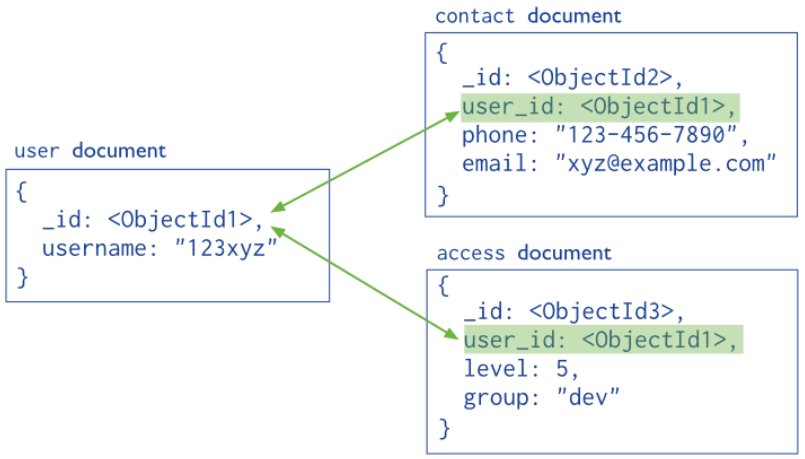
(information from Wikipedia)

Data Modelling

MongoDB has a *flexible schema*

References

store relationships between data by including links or references

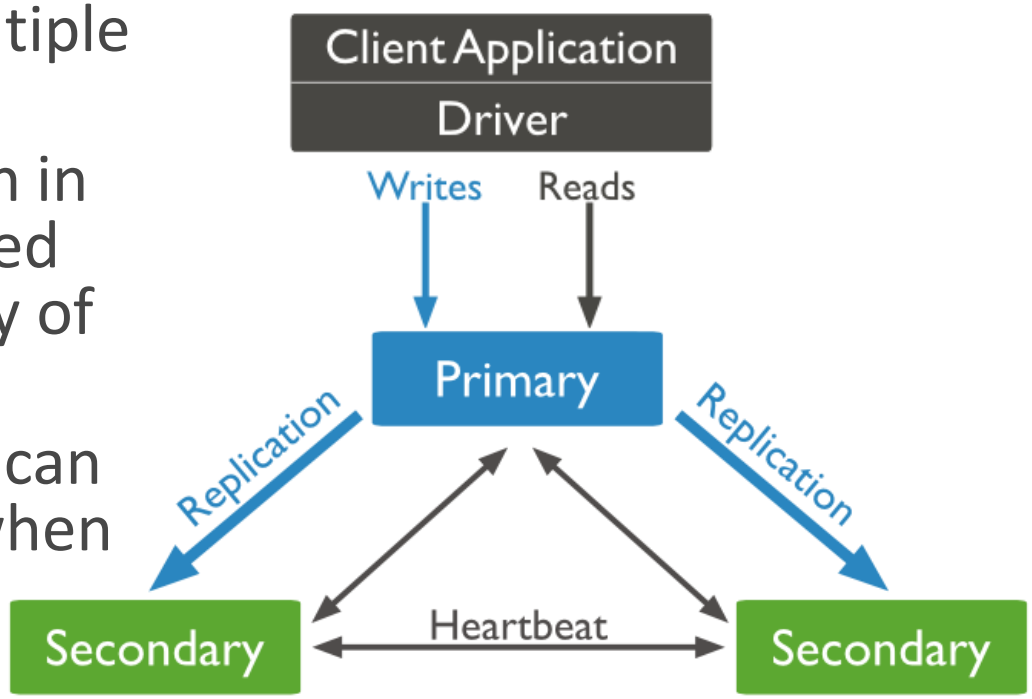


Embedded Data

denormalized data model

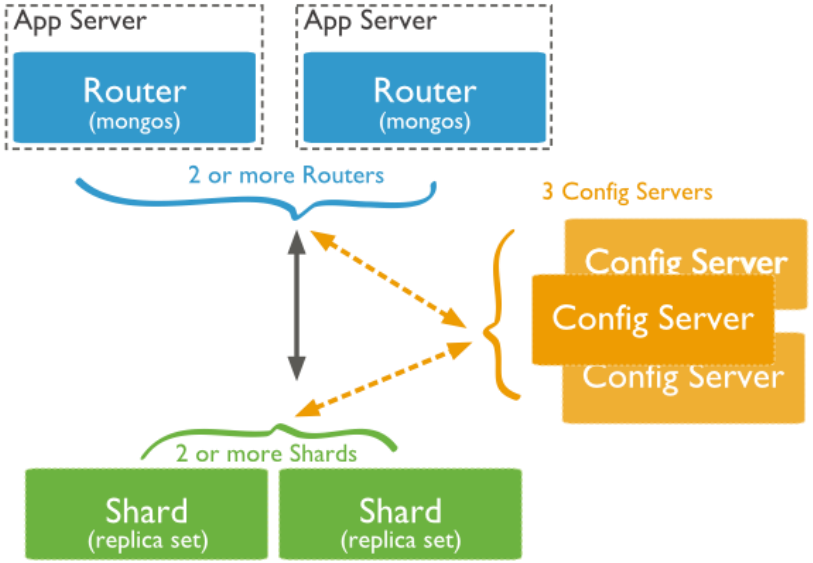
Replication and Sharding

- Data is replicated in a replica set
  - data copied to multiple servers
  - Each server session in a replica is supposed to be an exact copy of the primary.
  - Secondary servers can become primary when primary fails.



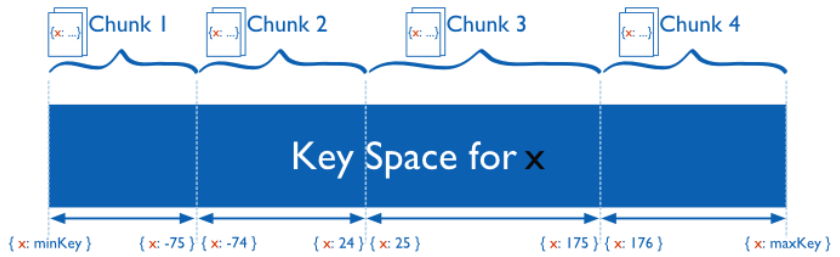
Replication and Sharding -- system setup

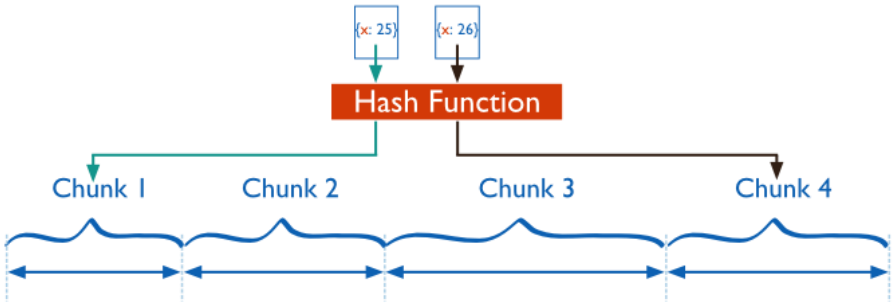
- Multiple servers for sharding
  - App Server
    - processes queries
  - Config Server
    - manages meta data for shards
  - Shard servers
    - contains section of data



Replication and Sharding -- distributing data

- Range Based Sharding
  - separate into small chucks. Each chuck has a continues section of primary key
- Hash Based Sharding
  - Primary key is hashed before being placed into chunk





Replication and Sharding -- extra notes

- Load Balancing is done with chunks (64MB blocks)
  - Migration happens when number of chucks in unbalanced
- Shards are replica sets
  - replica sets are often >3 machines with duplicate data
- Queries always pass through App server. If a query does not contains primary key, all shards will be queried.

Queries -- JSON data

```
{
   "firstName": "John",
   "lastName": "Smith",
   "age": 25,
   "address": {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": "10021"
   },
   "phoneNumber": [
      {
         "type": "home",
         "number": "212 555-1234"
      },
      {
         "type": "fax",
         "number": "646 555-4567"
      }
   ],
   "gender":{
      "type":"male"
   }
}
```

JSON records describe the data they hold

New data elements in a records can be added or removed as needed.

Queries -- simple data extraction

- An empty document query selects all documents in the collection:
  - db.inventory.find({ })
- Equality conditions use { <field>: <value> } to select all documents that have that field with the specific value
  - db.inventory.find({ type: "snacks" })

Query and Projection Operators

- Query Operations
  - Comparison (!=, >, in, . . .)
  - Logical (and, or, not, nor)
  - Element (exists, check type of field)
  - Evaluation (mod, regular expression, where)
  - Geospacial (geoWithin, near, geoIntersects)
  - Array (element match, query size)
- Projection

Queries can be combined in various ways using these operators

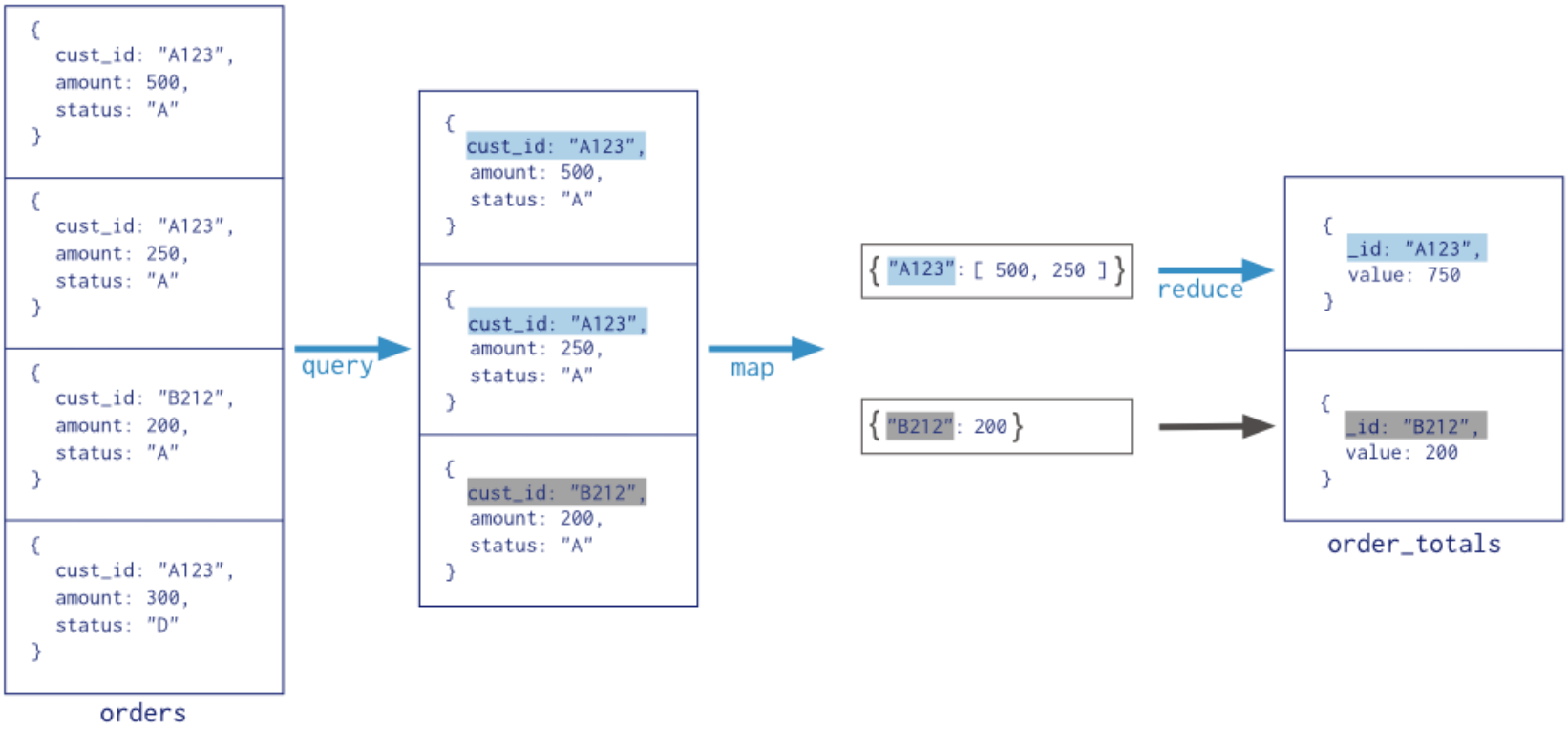
Map-Reduce

- Map-reduce is provided for processing aggregated results
  - use a custom JavaScript function for map and reduce operations
  - map-reduce operations can be saved as a collection for iterative map-reduce operations

Map Reduce example

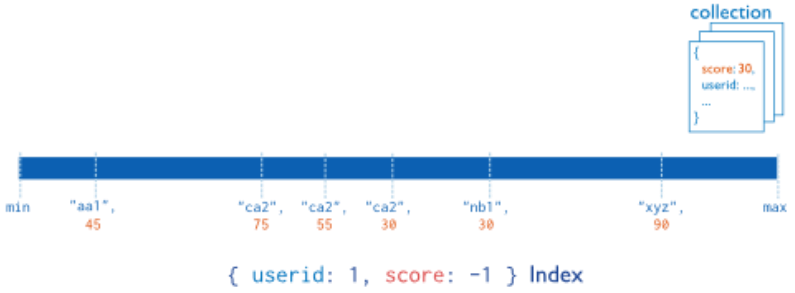
Collection

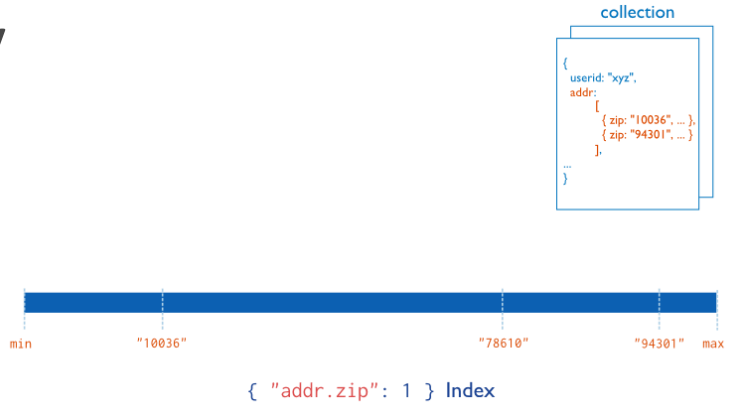
```
db.orders.mapReduce(
          map      →  function() { emit( this.cust_id, this.amount ); },
          reduce   →  function(key, values) { return Array.sum( values ) },
                      {
          query    →    query: { status: "A" },
          output   →    out: "order_totals"
                      }
                   )
```



Indexes -- 1

- Support for multiple index types on the database
  - Single Field
  - Compound Index
  - Multikey Index
    - index contents of arrays. Each array element has its own index entry





Indexes -- 2

- Support for multiple index types on the database
  - Geospatial Index
    - only 2D.
    - Planar geometry or spherical geometry
  - Text Indexes (beta)
    - Language specific stop words removed
  - Hash Based Indexing
    - Only supports equality join
- Range indexes use B+ tree

Conclusion

Distributed Document storage database

Indexing options allow for fast query of many data element

dynamic schema

Generic operations can be usable in many scenarios, but may not be optimized for those scenarios.

Many More Platforms / Frameworks

- **Cassandra:**
  - Distributed **database** management system

- **Spark:**
  - Distributed In-Memory **Cluster Computing Engine**
  - **Iterative Algorithms**
  - **Interactive Data Analysis**

- **RAM Cloud:**
  - Distributed In-Memory storage system

- **…**

Questions?