Big Data
- NoSQL Databases

Dr. Qing “Matt” Zhang
ITU

Outline

- Introduction
- Key-Value stores
- Column stores
- Graph Databases
- Document stores

Introduction

Why NoSQL?

- ACID doesn't scale well
- Web apps have different needs (than the apps that RDBMS were designed for)
  - Low and predictable response time (latency)
  - Scalability & elasticity (at low cost!)
  - High availability
  - Flexible schemas / semi-structured data
  - Geographic distribution (multiple datacenters)

Why NoSQL?

- Web apps can (usually) do without
  - Transactions / strong consistency / integrity
  - Complex queries

Some NoSQL Use Cases

- Massive data volumes
  - Massively distributed architecture required to store the data
  - Google, Amazon, Yahoo, Facebook – 10-100K servers
- Extreme query workload
  - Impossible to efficiently do joins at that scale with an RDBMS

Some NoSQL Use Cases

- Schema evolution
  - Schema flexibility (migration) is not trivial at large scale
  - Schema changes can be gradually introduced with NoSQL

NoSQL Pros

- Massive scalability
- High availability
- Lower cost (than competitive solutions at that scale)
- (usually) predictable elasticity
- Schema flexibility, sparse & semi-structured data

NoSQL Cons

- Limited query capabilities (so far)
- Eventual consistency is not intuitive to program for
  - Makes client applications more complicated
- No standardization
  - Portability might be an issue
- Insufficient access control

Requirements to Distributed Systems

- *Consistency* – the system is in a consistent state after an operation
- Availability – the system is “always on”, no downtime
  - Node failure tolerance – all clients can find *some* available replica
- *Partition tolerance*
  - the system continues to function even when split into disconnected subsets (by a network disruption)

CAP Theorem

- You can satisfy **at most 2** out of the 3 requirements
  - *Consistency*
  - Availability
  - *Partition tolerance*

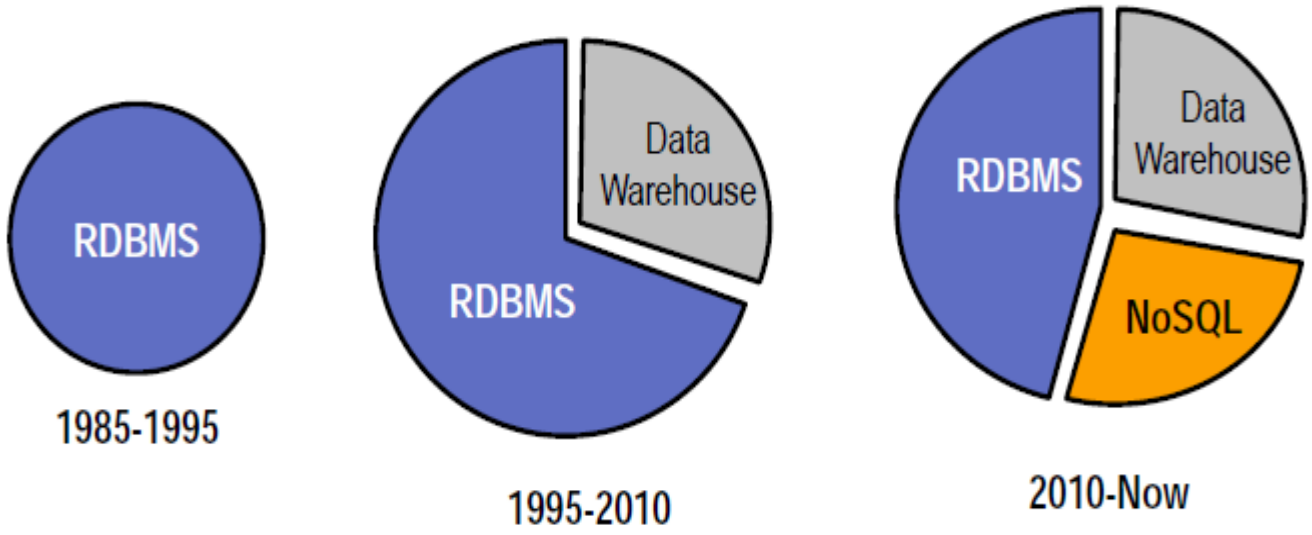
CAP In Production

- CP
  - *Some* data may be inaccessible (availability sacrificed), but the rest is still consistent / accurate
  - e.g. sharded database
- AP
  - System is still available under partitioning, but *some* of the data returned may be inaccurate
  - e.g. DNS, caches, Master/Slave replication
  - Need some conflict resolution strategy

NoSQL Taxonomy

- Key-Value stores
  - Simple K/V lookups (DHT)
- Column stores
  - Each key is associated with many attributes (columns)
  - NoSQL column stores are actually *hybrid* row/ column stores
    - Different from “pure” relational column stores!
- Document stores
- Graph databases

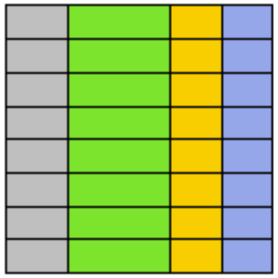
Three Eras of Databases



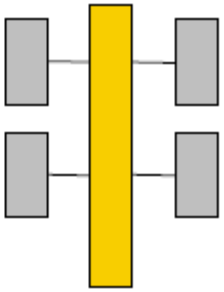
- RDBMS for transactions
- Data Warehouse for analytics

Before NoSQL

Relational

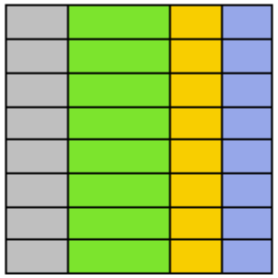


Analytical (OLAP)

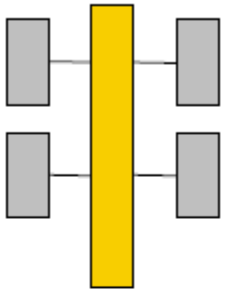


After NoSQL

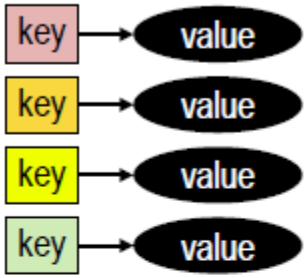
Relational



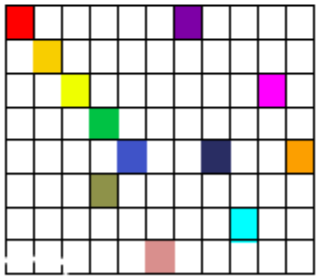
Analytical (OLAP)



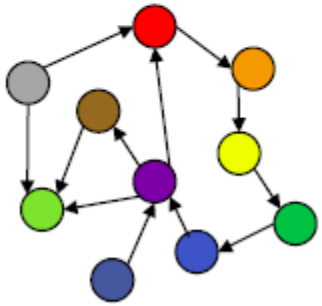
Key-Value



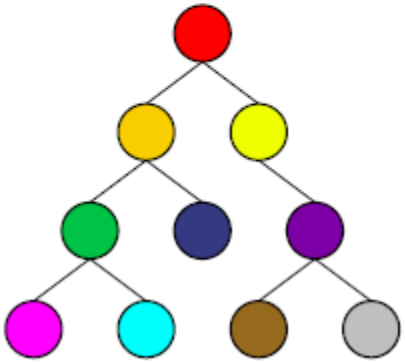
Column-Family



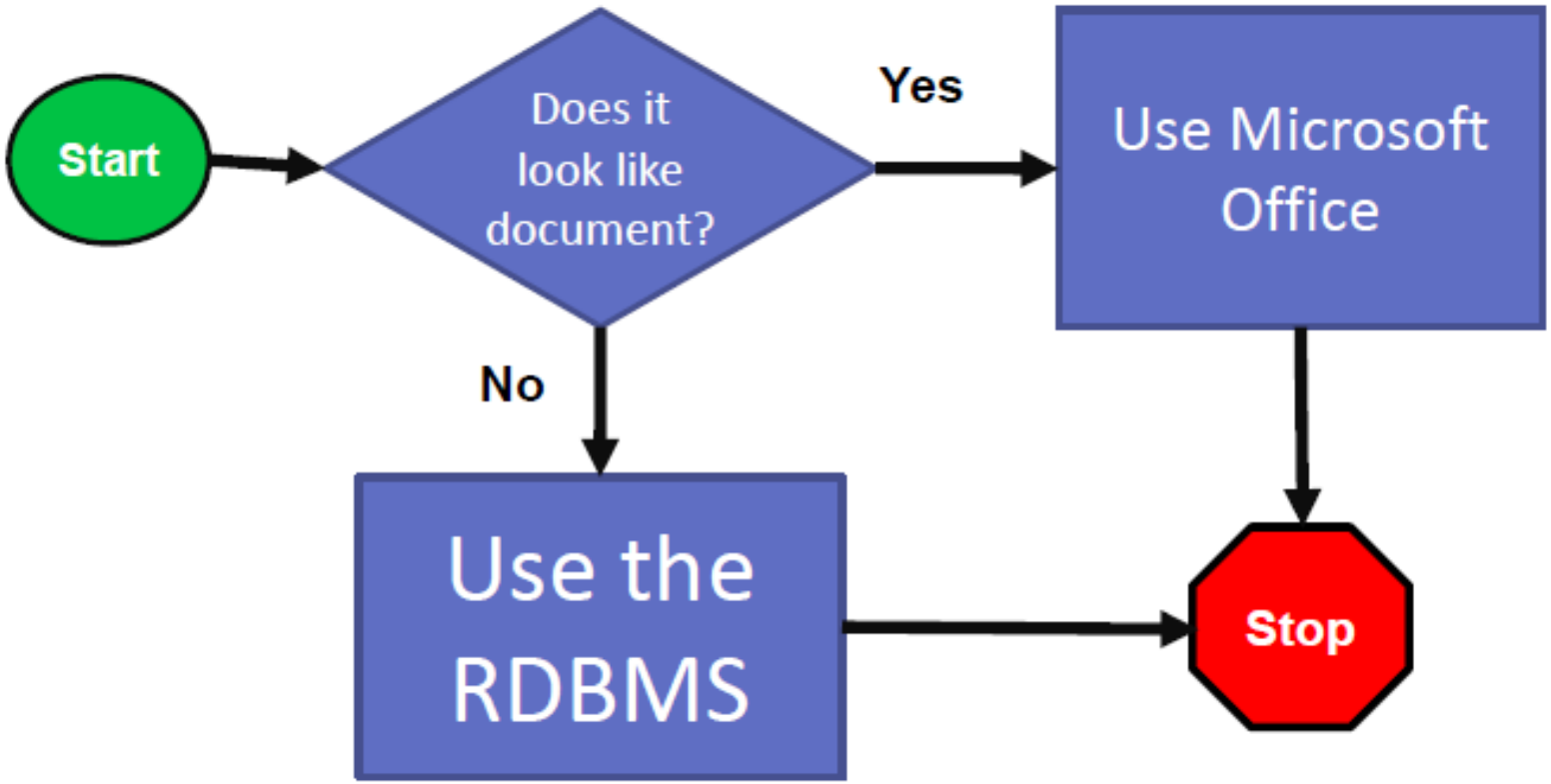
Graph



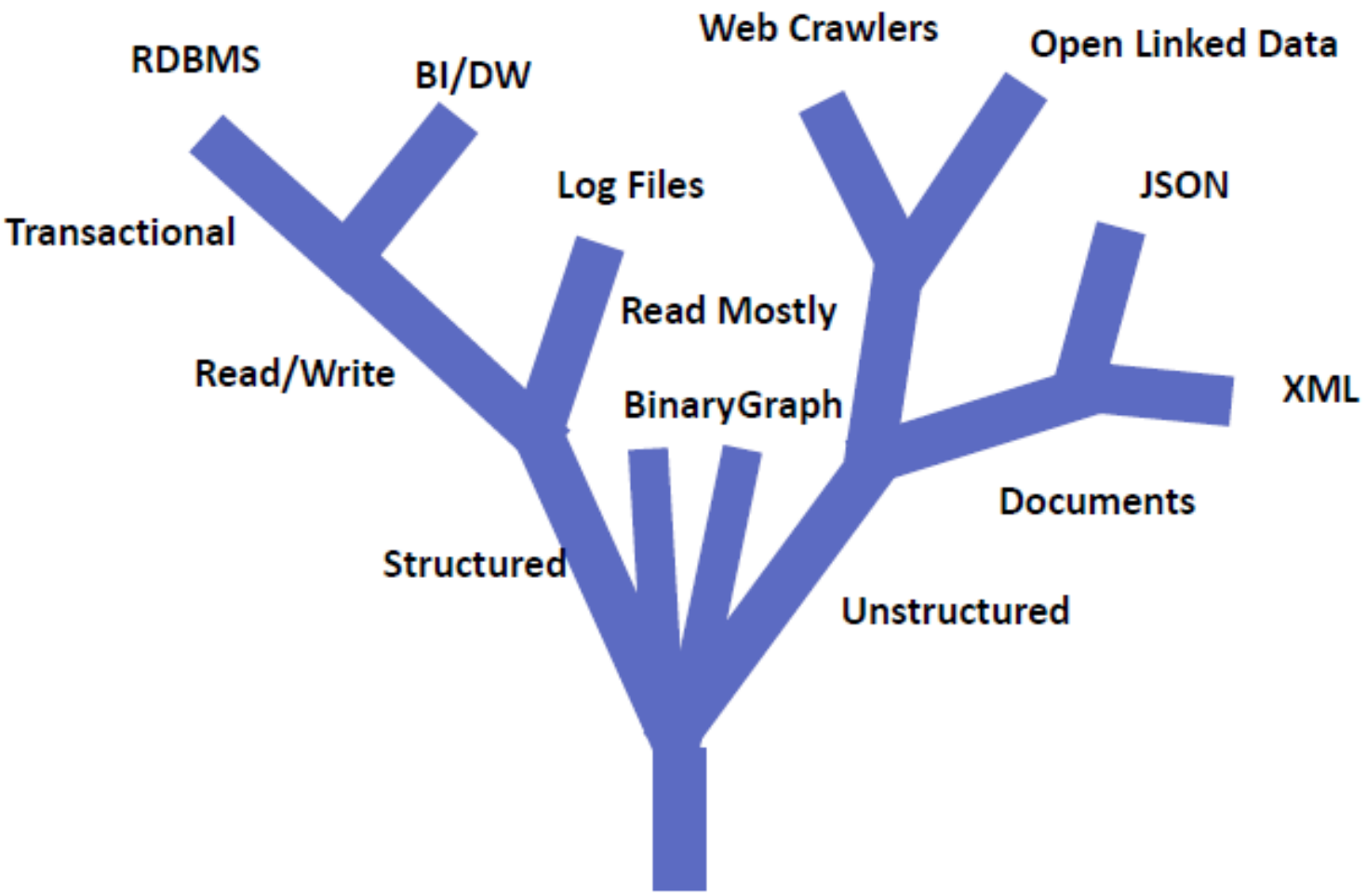
Document



Before NoSQL DB Selection was Easy



An Evolving Tree of Data Types

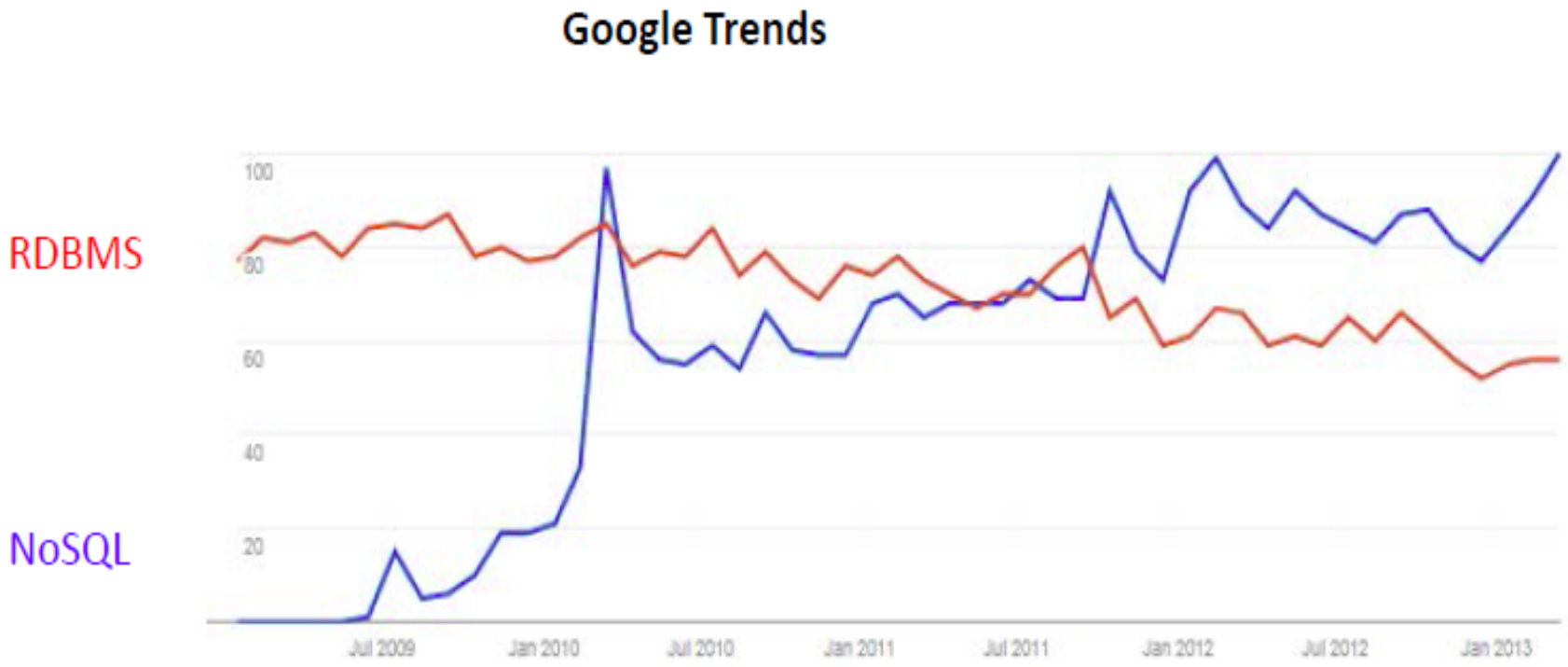


Simplicity is a Design Style

- Focus only on simple systems that solve many problems in a flexible way
- Examples
  - Touch screen interfaces
  - Key/Value data stores

RDBMS vs. NoSQL

- NoSQL is real and it's here to stay

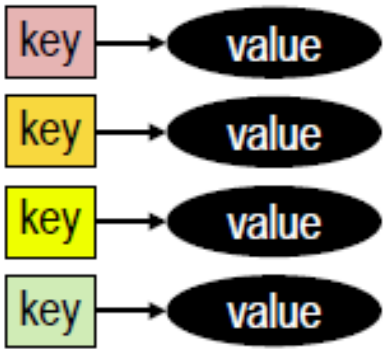


The NoSQL Universe

Key-Value Stores

Introduction

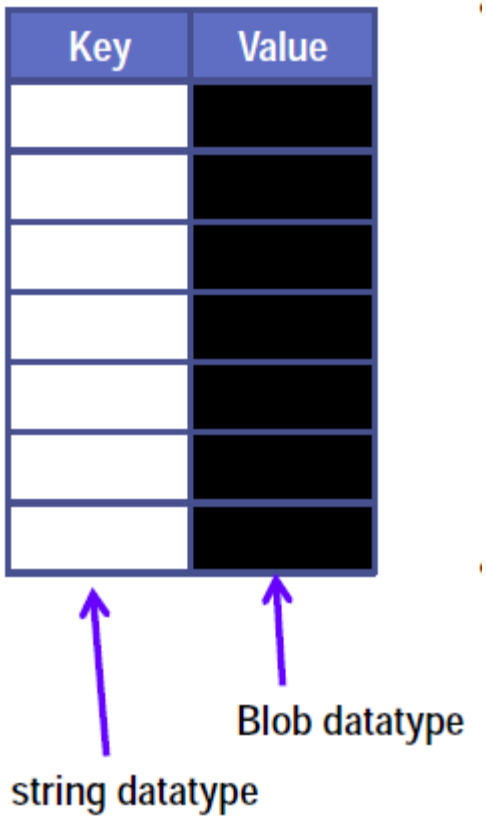
- Keys used to access opaque blobs of data
- Values can contain any type of data (images, video)
- **Pros:** scalable, simple API (put, get, delete)
- **Cons:** no way to query based on the content of the value



Examples:
Berkley DB, Memcache, DynamoDB, S3, Redis, Riak

Key Value Stores

- A table with two columns and a simple interface
  - Add a key-value
  - For this key, give me the value
  - Delete a key
- Blazingly fast and easy to scale (no joins)



The Locker Metaphor

Key-Values Stores are Like Dictionaries

The "key" is just the word "gouge"

The "value" is all the definitions and images

gouge |gouj|

noun

1 a chisel with a concave blade, used in carpentry, sculpture, and surgery.

2 an indentation or groove made by gouging.

gouge 1

verb [trans.]

1 make (a groove, hole, or indentation) with or as if with a gouge : *the channel had been* ***gouged out*** *by the ebbing water.*

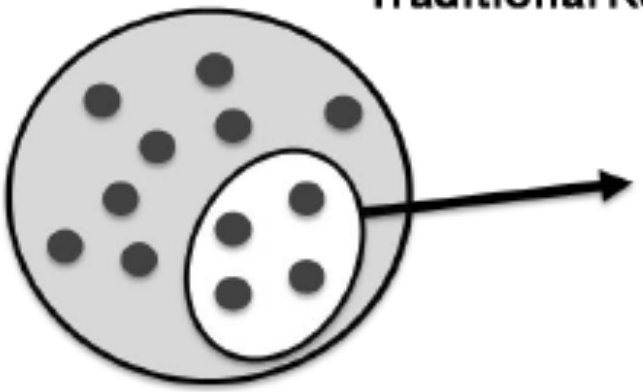
• make a rough hole or indentation in (a surface), esp. so as to mar or disfigure it : *he had wielded the blade inexpertly, gouging the grass in several places.*

• (**gouge something out**) cut or force something out roughly or brutally : *one of his eyes had been gouged out.*

2 informal overcharge; swindle : *the airline ends up gouging the very passengers it is supposed to assist.*

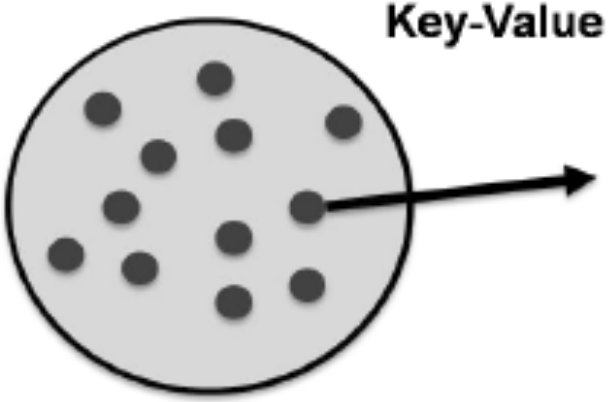
No Subset Queries in Key-Value Stores

Traditional Relational Model



- Result set based on row values
- Value of rows for large data sets must be indexed
- Values of columns must all have the same data type

Key-Value Store Model



- All queries return a single item
- No indexes on values
- Values may contain any data type

Types of Key-Value Stores

- Eventually-consistent key-value store
- Hierarchical key-value stores
- Key-Value stores in RAM
- Key-Value stores on disk
- High availability key-value store
- Ordered key-value stores
- Values that allow simple list operations

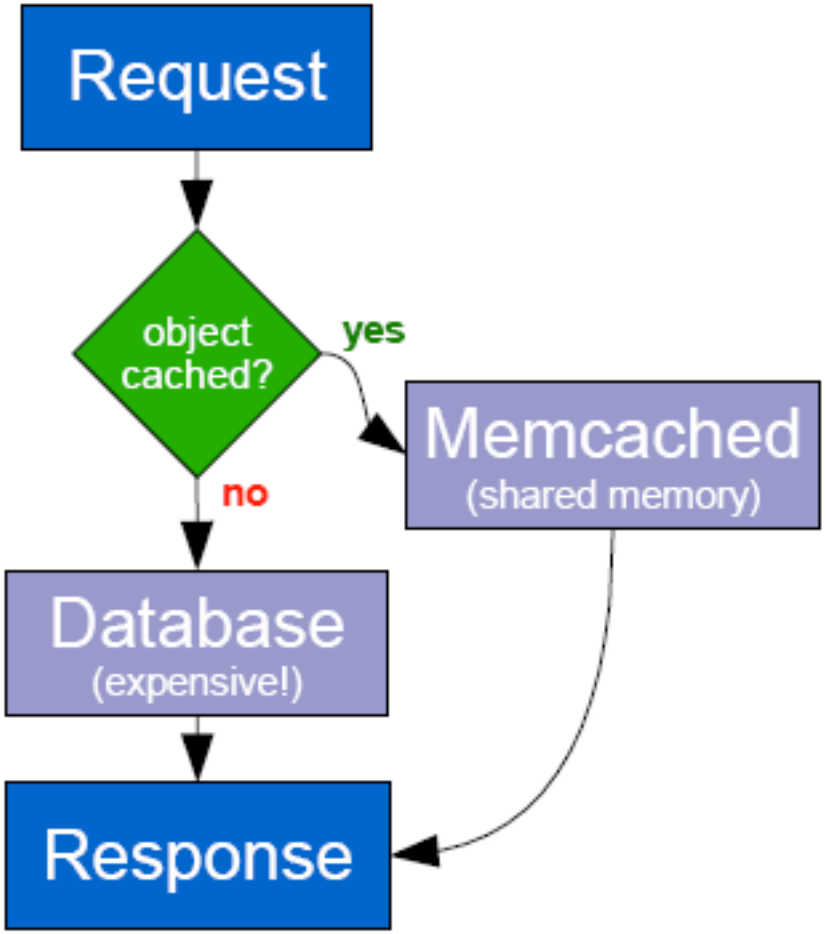
Memcached

- Open source in-memory key-value caching system
- Make effective use of RAM on many distributed web servers
- Designed to speed up dynamic web applications by alleviating database load

Memcached

- RAM resident key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering
- Simple interface for highly distributed RAM caches
- 30ms read times typical
- Designed for quick deployment, ease of development
- APIs in many languages

Memcached


Request
object cached?
yes
no
Memcached
(shared memory)
Database
(expensive!)
Response

riak

- Open source distributed key-value store with support and commercial versions by Basho
  - A "Dynamo-inspired" database
  - Focus on availability, fault-tolerance, operational simplicity and scalability
  - Support for replication and auto-sharding and rebalancing on failures
  - Support for MapReduce, fulltext search and secondary indexes of value tags
  - Written in ERLANG

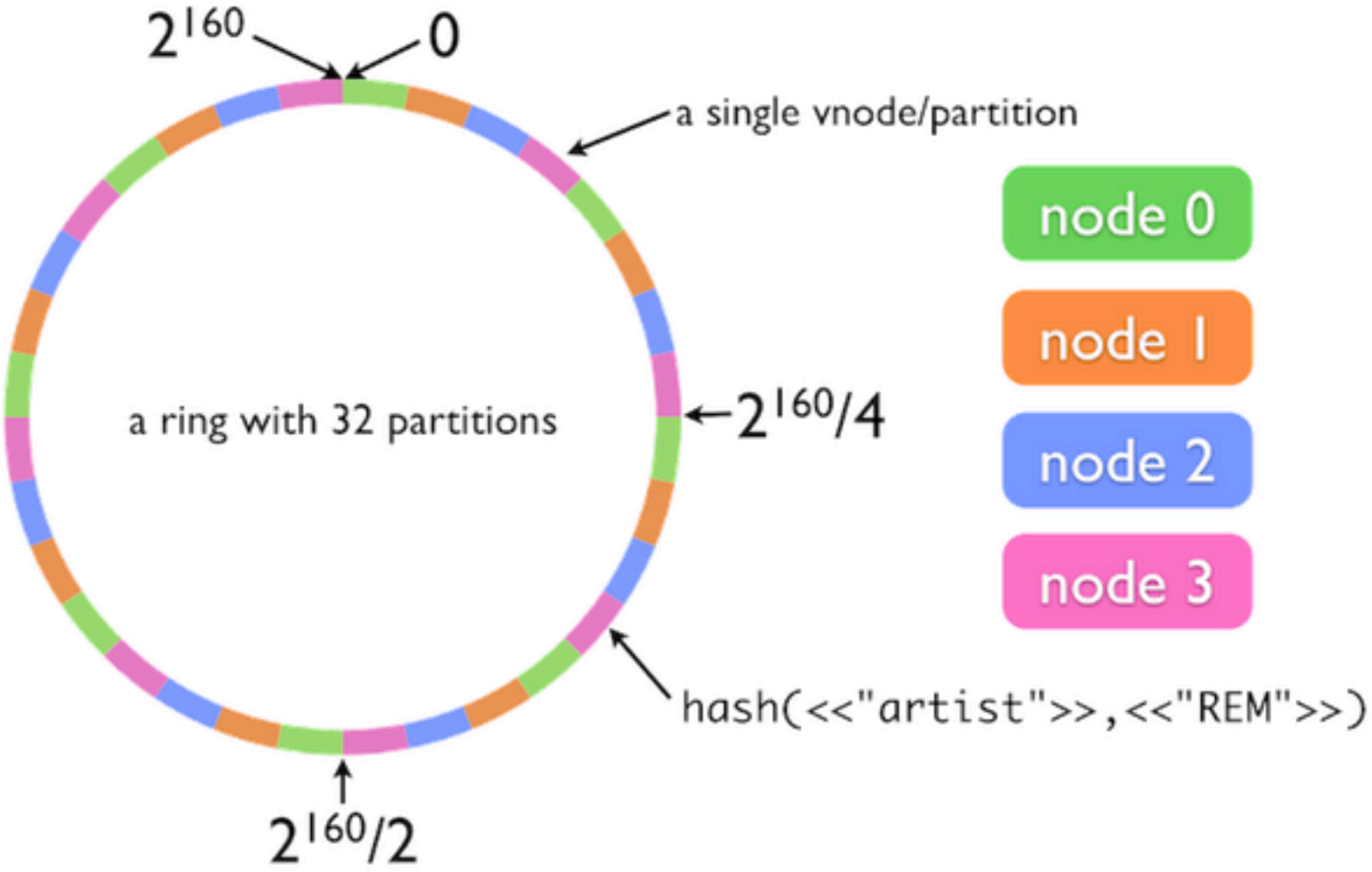
Riak Ring

- Riak clusters are often referred to as a "ring”
  - not a physical or network architecture
  - but rather a way to visualize how objects are handled
- Riak maps objects with a key hash, and hash value is divided into *partitions*
  - Riak cluster is a 160-bit integer space which is divided into equally-sized partitions.
  - Default value is 64: a ring divided into 64 sections.
  - Riak also maintains 3 copies of objects by default
    - so you can also find this object in 2 other partitions.

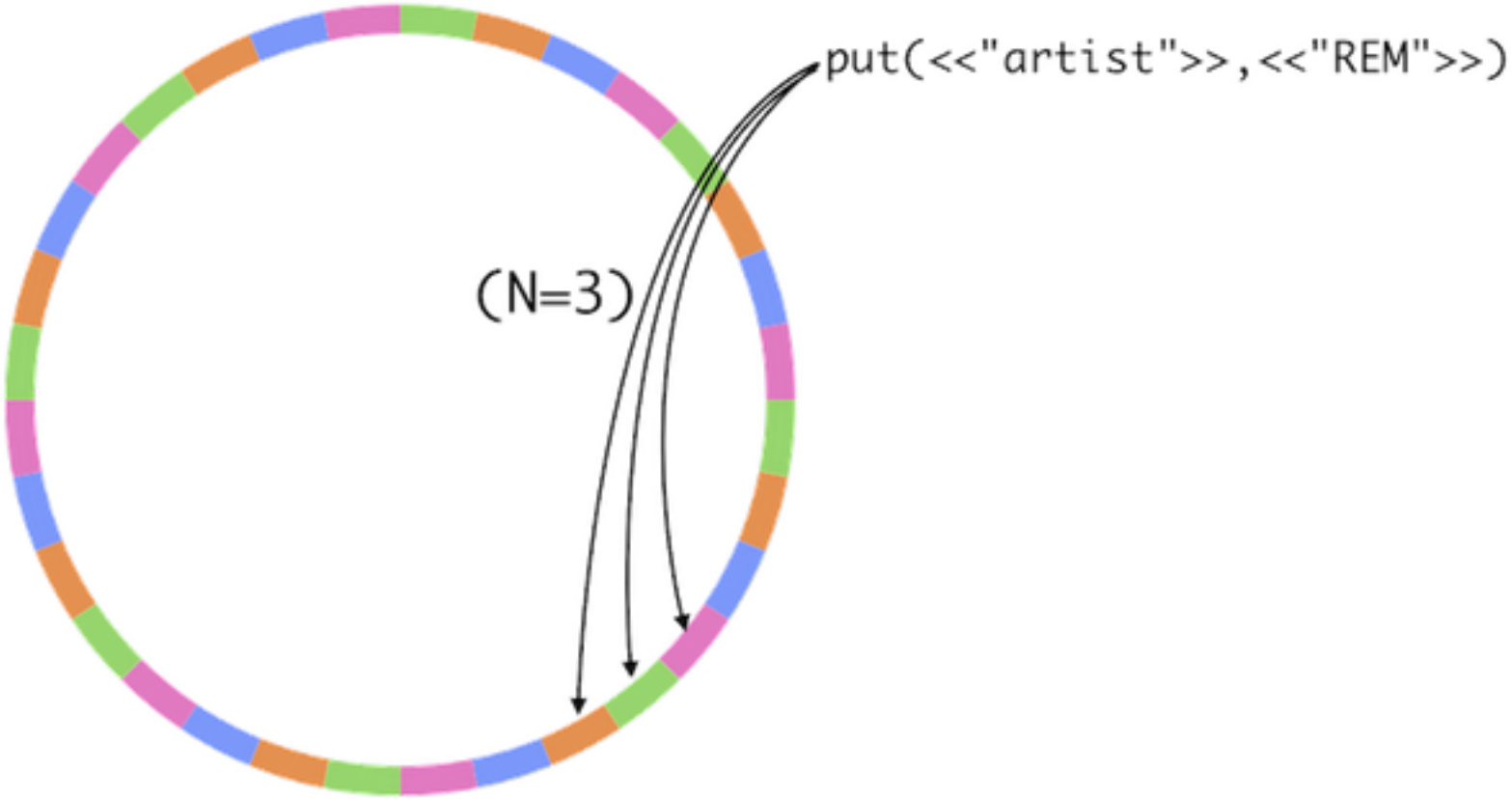
Riak Ring

- It's possible to have more than one partition per node
  - a concept Riak calls *vnodes* (virtual node).
  - A ring with 64 partitions deployed on 3 nodes, will have approximately 21 *vnodes* per node.
    - A deployment with 5 nodes will have approximately 13 *vnodes* per node.
- Riak is masterless, each node is aware of every other node in the cluster, their vnodes and other data.

Riak Ring


2^{160}
0
a single vnode/partition
node 0
node 1
node 2
node 3
a ring with 32 partitions
$2^{160}/4$
hash(<<"artist">>,<<"REM">>)
$2^{160}/2$

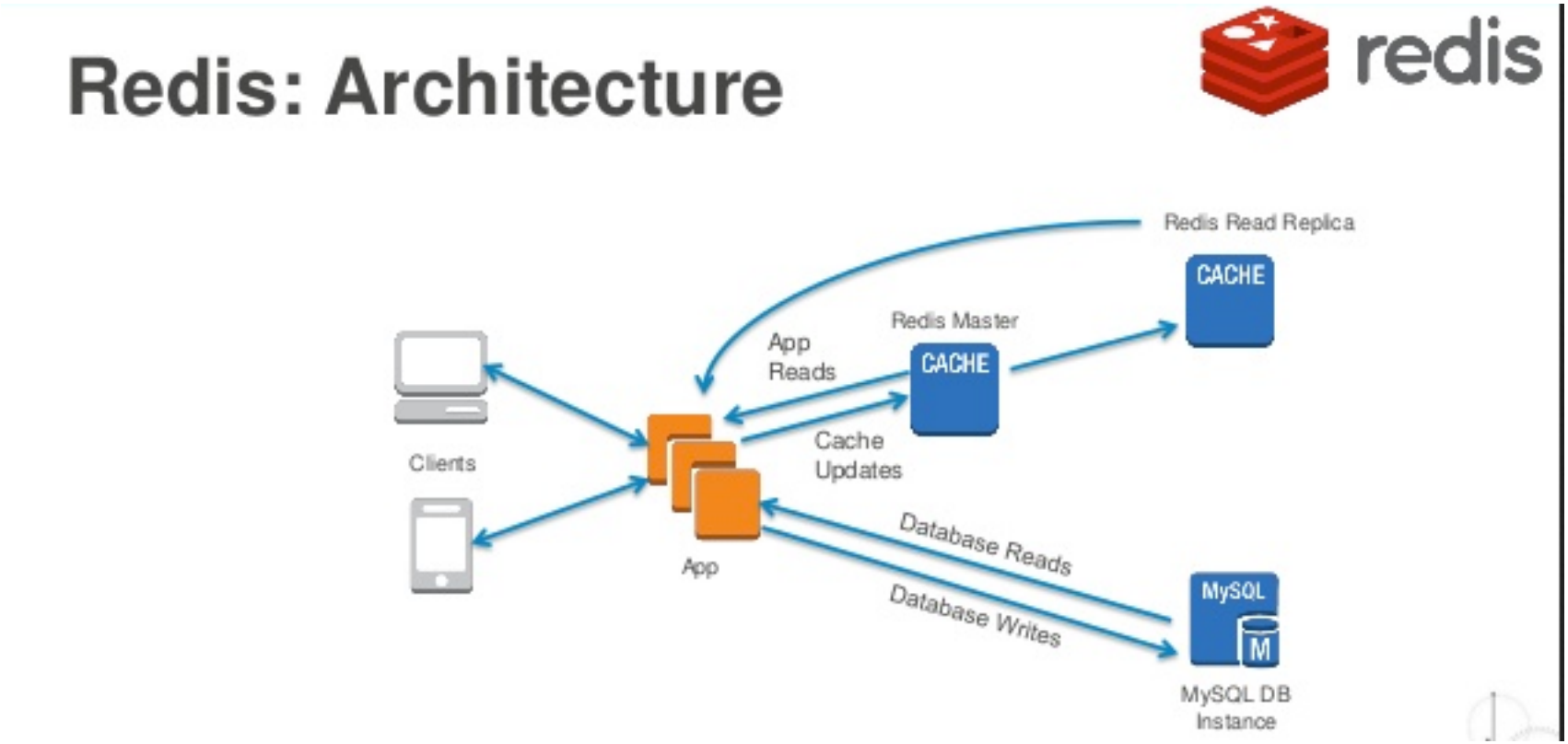
Riak Replication



redis

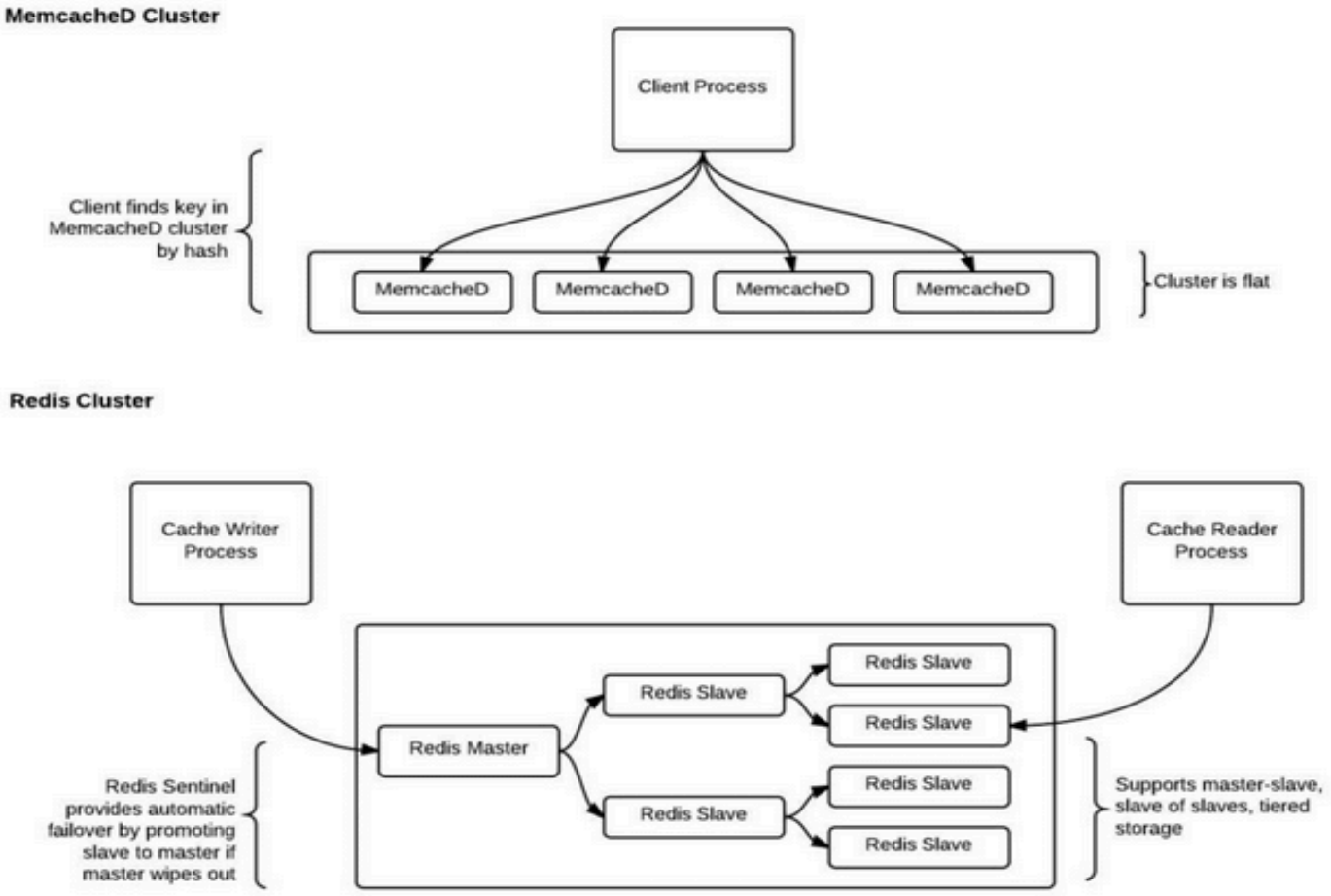
- Open source in-memory key-value store with optional durability
- Focus on high speed reads and writes of common data structures to RAM
- Allows simple lists, sets and hashes to be stored within the value and manipulated
- Many features that developers like
  - expiration, transactions, pub/sub, partitioning

Redis architecutre



Redis vs memcached

- Redis supports more data formats, and more complicate to manage (master-slave)



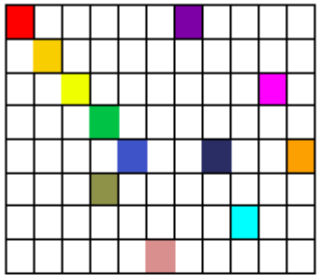
Amazon DynamoDB

- Based around scalable key-value store
- Fastest growing product in Amazon's history
- SSD **only** database service
- Focus on throughput not storage and predictable read and write times
- Strong integration with S3 and Elastic MapReduce

Column Store

Column-Family

- Key includes a row, column family and column name
- Store versioned blobs in one large table
- Queries can be done on rows, column families and column names
- Pros: Good scale out, versioning
- Cons: Cannot query blob content, row and column designs are critical

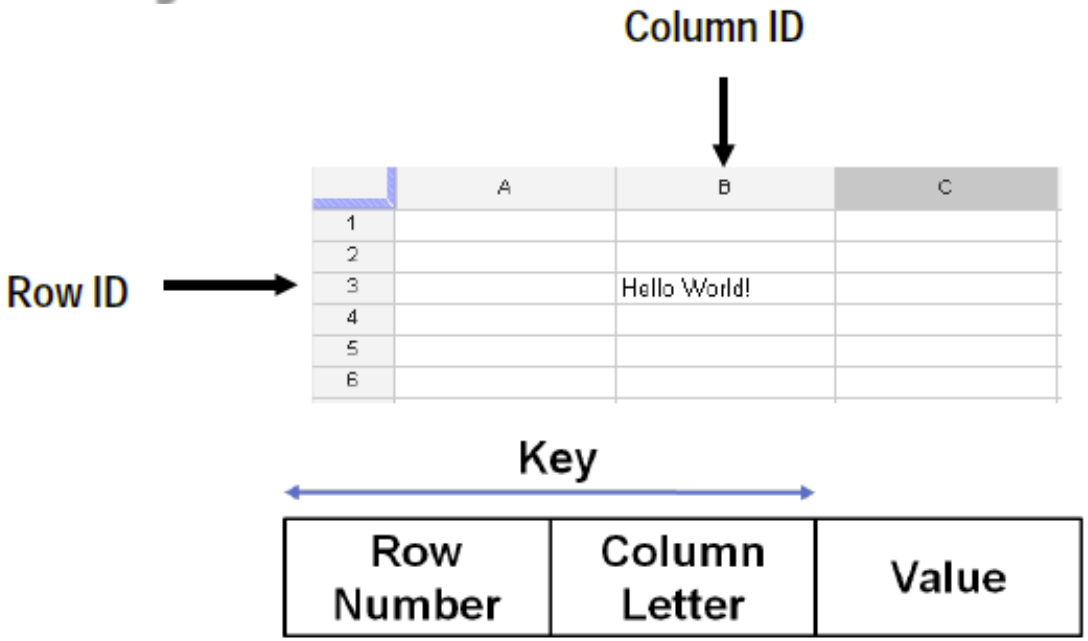


Examples:
Cassandra, HBase, Hypertable, Apache Accumulo, Bigtable

Column Family (Bigtable)

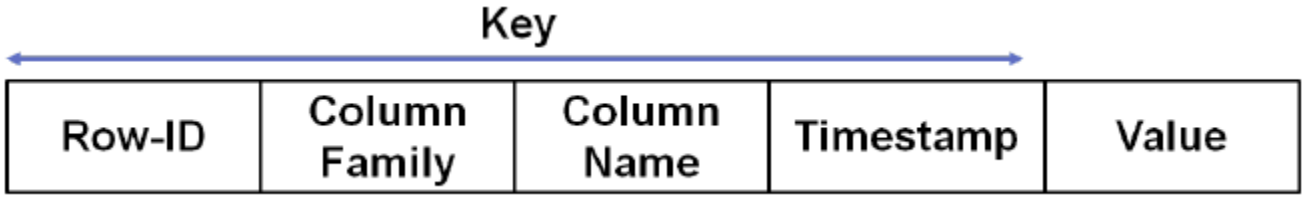
- The champion of "Big Data"
  - Excel at highly saleable systems
  - Tightly coupled with MapReduce
- Technically a "sparse matrix" where most cells have no data
  - Generating a list of all columns is non-trivial
- Examples:
  - Google Bigtable
  - Hadoop HBase

Spreadsheets Use a Row/Column as a Key



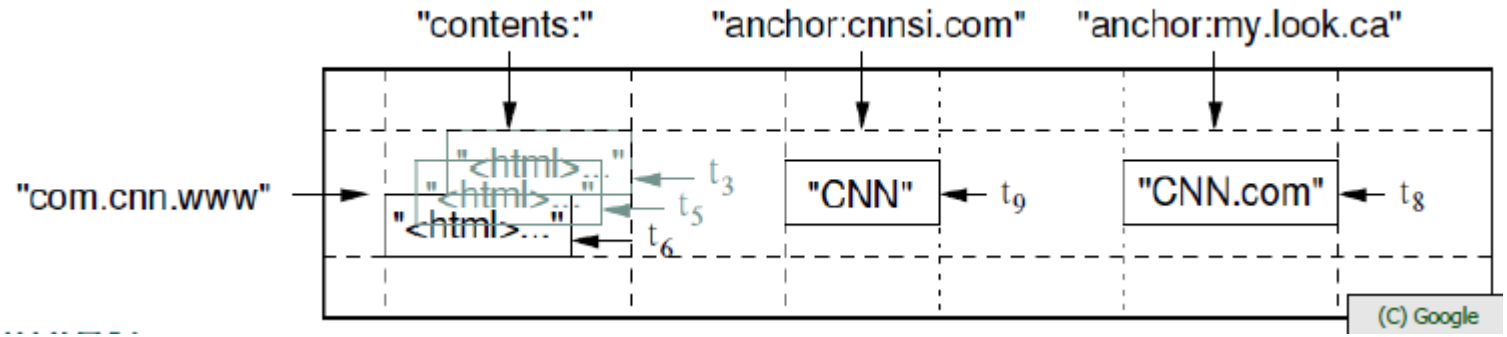
- Bigtable systems use a combination of row and column information as part of their key

Keys Include Family and Timestamps



- Bigtable systems have keys that include not just row and column ID but other attributes
  - Column Families are created when a table is created
  - Timestamps allows multiple versions of values
  - Values are just ordered bytes and have no strongly typed data system

BigTable



- Google, ~2006
- Sparse, distributed, persistent multidimensional sorted map
- *(row, column, timestamp)* dimensions, value is *string*

BigTable Data Model

- *Row*
  - Keys are arbitrary strings
  - Data is sorted by row key
- *Tablet*
  - Row range is dynamically partitioned into tablets (sequence of rows)
  - Range scans are very efficient
  - Row keys should be chosen to improve *locality* of data access

BigTable Data Model

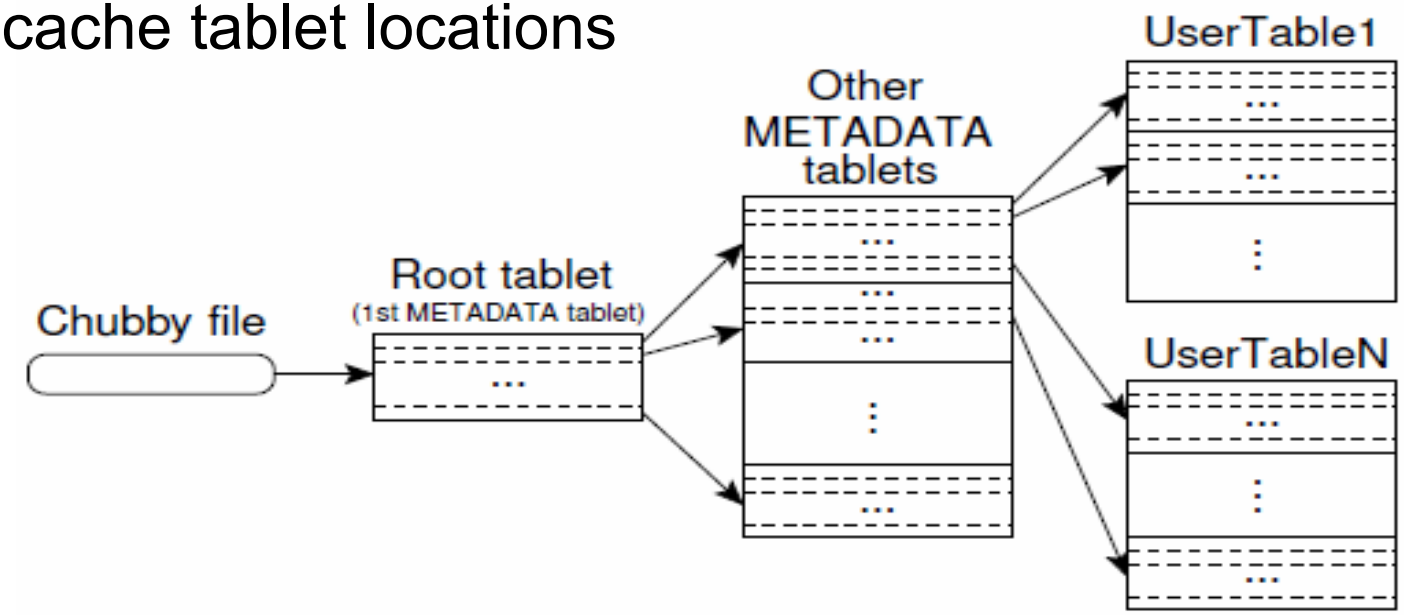
- *Column, Column Family*
  - Column keys are arbitrary strings, unlimited number of columns
  - Column keys can be grouped into families
  - Data in a CF is stored and compressed together (Locality Groups)
  - Access control on the CF level
- *Timestamps*
  - Each cell has multiple versions
  - Can be manually assigned

BigTable Architecture

- Data stored on *GFS*
- Relies on *Chubby* distributed lock service
  - based on Paxos protocol
- 1 *Master server*
- Thousands of *Tablet servers*

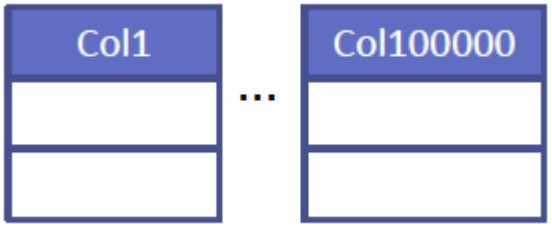
BigTable Tablet Location

- B+ tree index
- root (Chubby file) - location of the Root metadata tablet
- 1st level (Root Metadata tablet) – locations of all Metadata tablets
- 2nd level (Metadata tablet) – location of a set of tablets
- Clients cache tablet locations



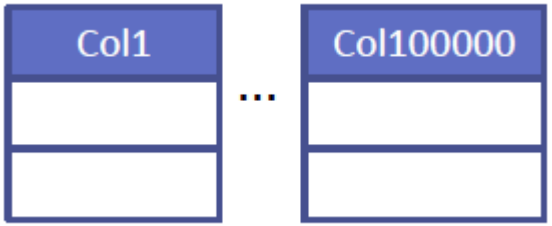
Column Store Concepts

- Preserve the table-structure similiar to RDBMS systems
- Not optimized for "joins"
- One row could have millions of columns but the data can be very "sparse"
- Ideal for high-variability data sets



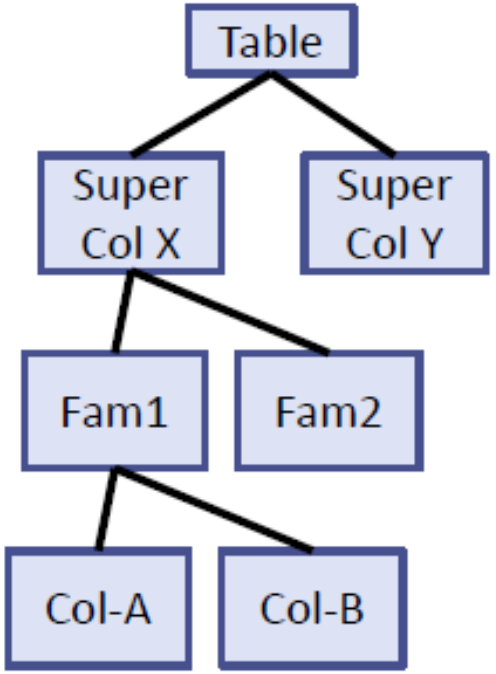
Column Store Concepts

- Colum families allow to query all columns that have a specific property or properties
- Allow new columns to be inserted without doing an "alter table"
- Trigger new columns on inserts



Column Families

- Group columns into "Column families"
- Group column families into "Super-Columns"
- Be able to query all columns with a family or super family
- Similar data grouped together to improve speed



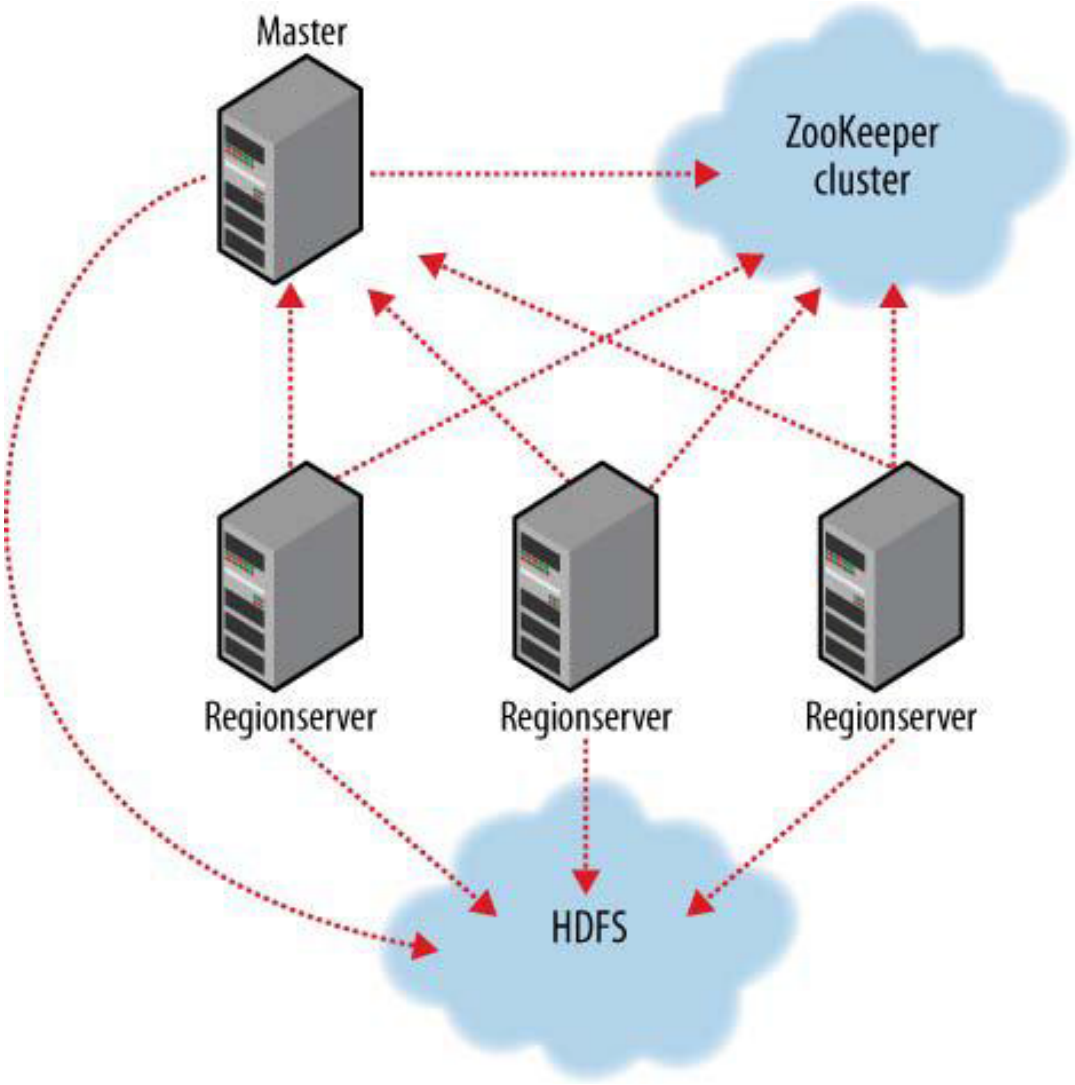
Apache HBase

- Open source implementation of MapReduce algorithm written in Java
- Initially created by Yahoo
  - 300 person-years development
- Column-oriented data store
- Java interface
- HBase designed specifically to work with Hadoop
  - Strong support by many vendors

Apache HBase

- Based on BigTable
  - *HDFS* (GFS), *ZooKeeper* (Chubby)
  - *Master Node* (Master Server), *Region Servers* (Tablet Servers)
  - *HStore* (tablet), *memcache* (memtable), MapFile (SSTable)
- Features
  - Data is stored sorted (no real indexes)
  - Automatic partitioning
  - Automatic re-balancing / re-partitioning
  - Fault tolerance (HDFS, 3 replicas)

Hbase - Architecture



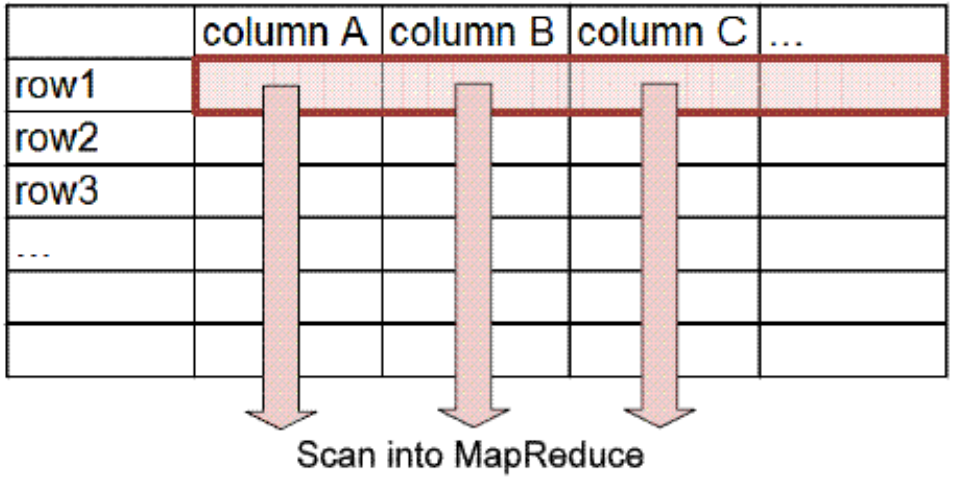
HBase Workflow

- Hstore/tablet location (== BigTable)
  - clients talk to ZooKeeper
  - root metadata, metadata tables, client side caching, etc
- Writes (== BigTable)
  - First added to the commit log, then to the *memcache* (memtable), eventually flushed to MapFile (SSTable)
- Reads (== BigTable)
  - First read from *memcache*, if info is incomplete then read the MapFiles

HBase MapReduce Jobs

- Can be a source/sink for M/R jobs in Hadoop

HBase as a MapReduce input



- Each row is an input record to MapReduce
- MapReduce jobs can sort/search/index/query data in bulk

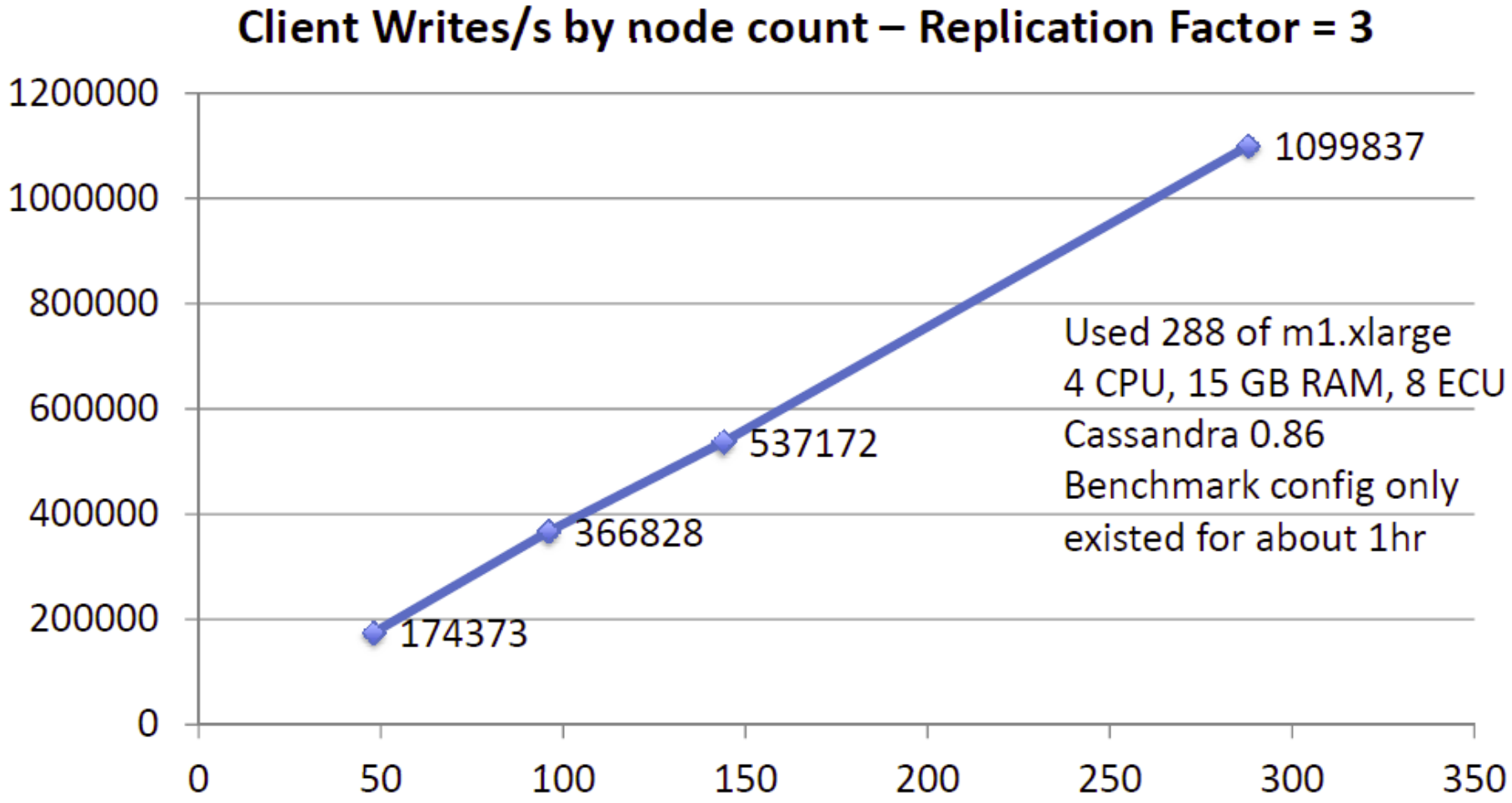
HBase vs. BigTable

Feature	BigTable	HBase
Access control	+	-
Master server	single	single/multiple*
Locality groups	+	-
Cell cache	+	-
Commit logs (WAL)	primary + secondary	single
Skip WAL for bulk loads	?	yes
Data isolation	+	-
replication	+	In progress
Everything else pretty much the same!		

Apache Cassandra

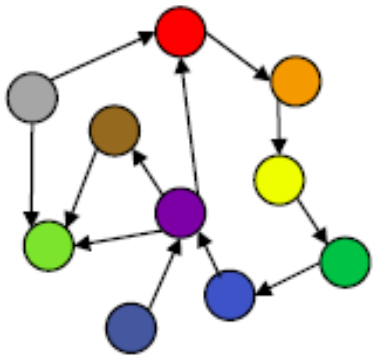
- Apache open source column family database supported by DataStax
- Peer-to-peer distribution model
- Strong reputation for linear scale out (millions of writes/second)
- Database side security
- Written in Java and works well with HDFS and MapReduce

Cassandra @Netflix



Graph Store

Graph Store

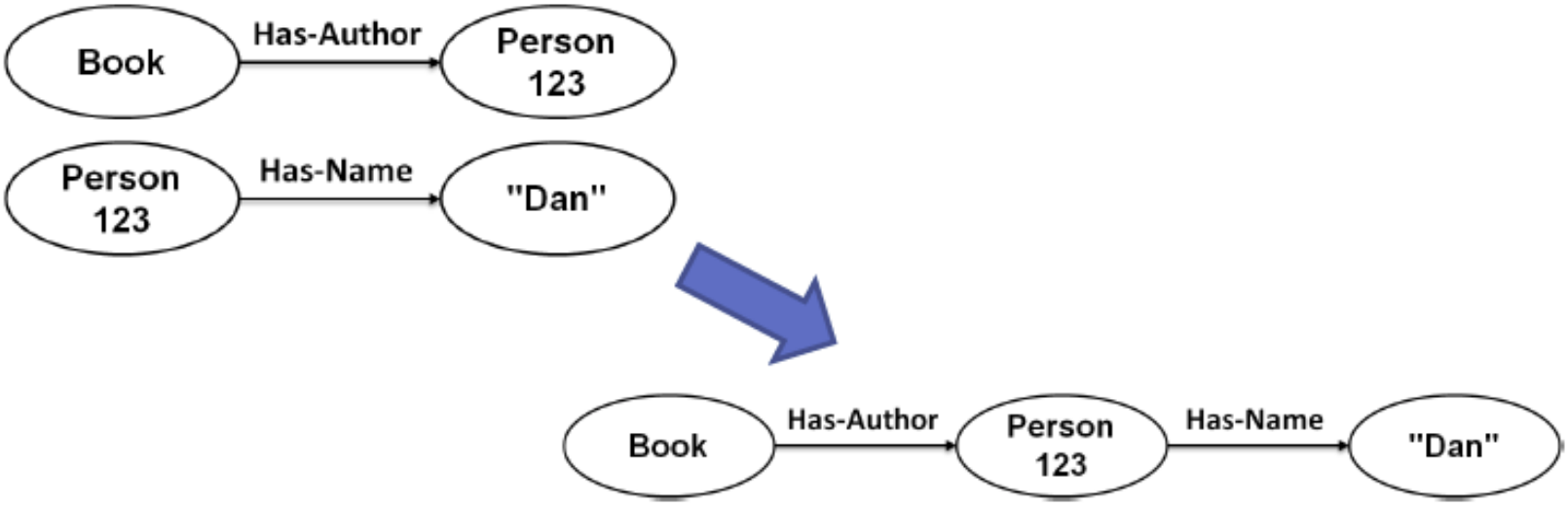


- Data is stored in a series of nodes, relationships and properties
- Queries are really graph traversals
- **Pros:**fast network search, works with public linked data sets
- **Cons:**Poor scalability when graphs don't fit into RAM, specialized query languages (RDF uses SPARQL)

Graph Store

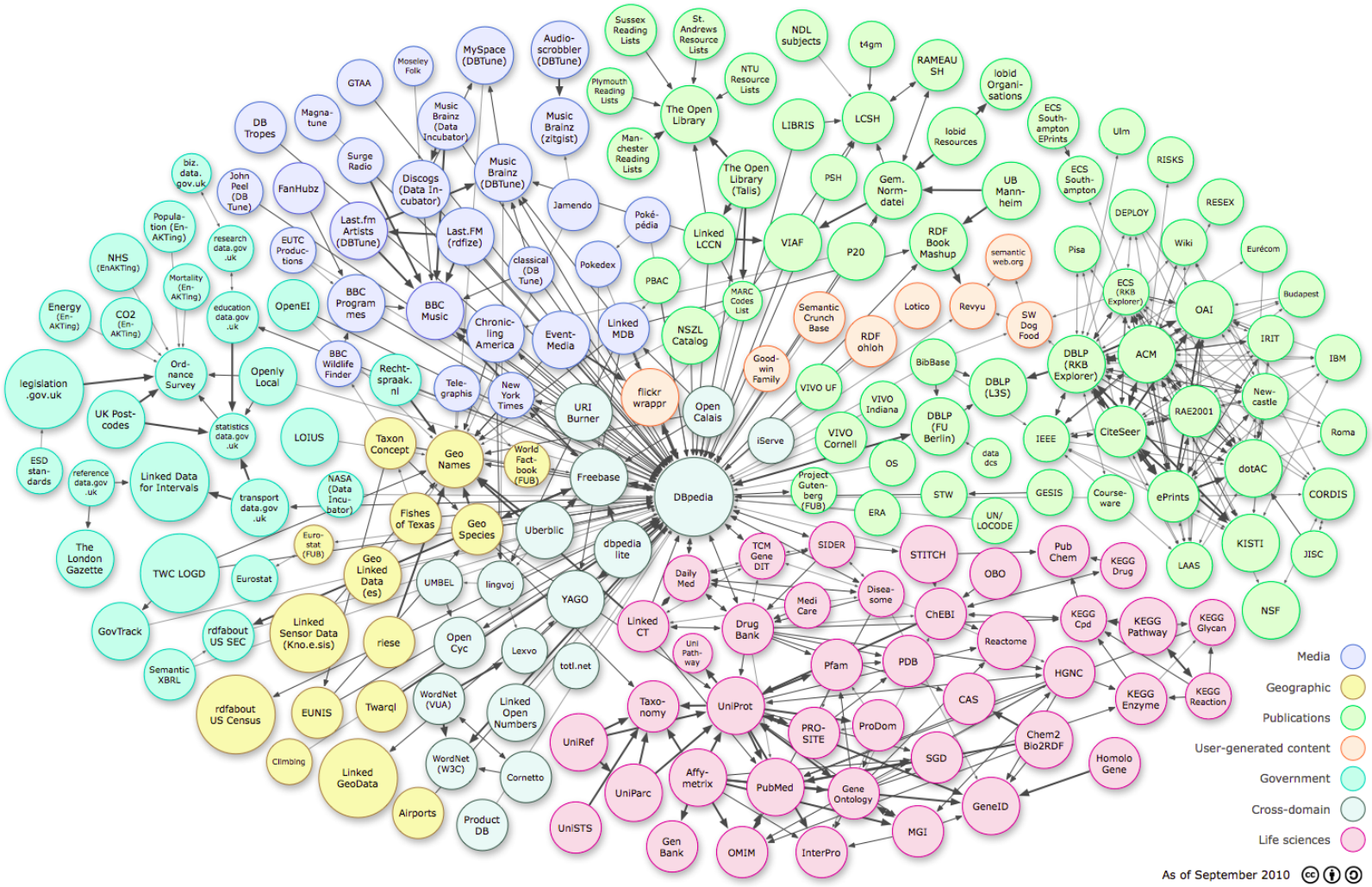
- Used when the relationship and relationships types between items are critical
- Used for
  - Social networking queries: "friends of my friends"
  - Inference and rules engines
  - Pattern recognition
- Automate "joins" of public data

Nodes are "joined" to create graphs



- How do you know that two items reference the same object?
- Node identification –URI or similar structure

Open Linked Data



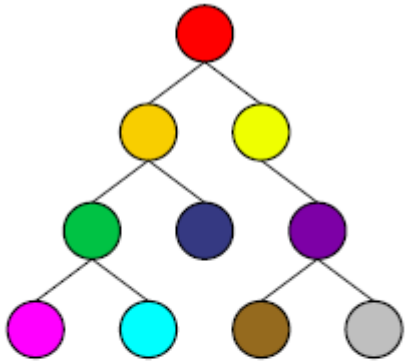
Neo4J

- Graph database designed to be easy to use by Java developers
- Dual license (community edition is GPL)
- Works as an embedded java library in your application
- Disk-based (not just RAM)
- Full ACID

Document Store

Document Store

- Data stored in nested hierarchies
- Logical data remains stored together as a unit
- Any item in the document can be queried
- **Pros:**No object-relational mapping layer, ideal for search
- **Cons:**Complex to implement, incompatible with SQL



Examples:
MarkLogic, MongoDB, Couchbase, CouchDB, eXist-db

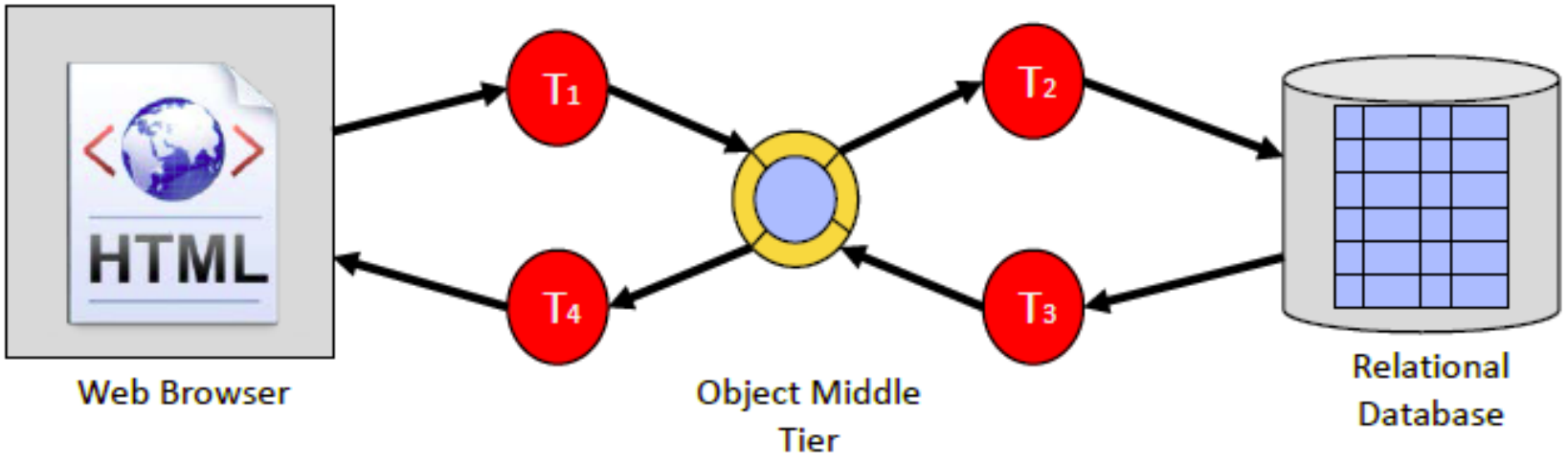
Document Stores

- Store machine readable documents together as a single blob of data
- Use JSON or XML formats to store documents
- Similar to "object stores" in many ways
- No shredding of data into tables
- Sub-trees and attributes of documents can still be queried XQuery or other document query languages

Document Stores

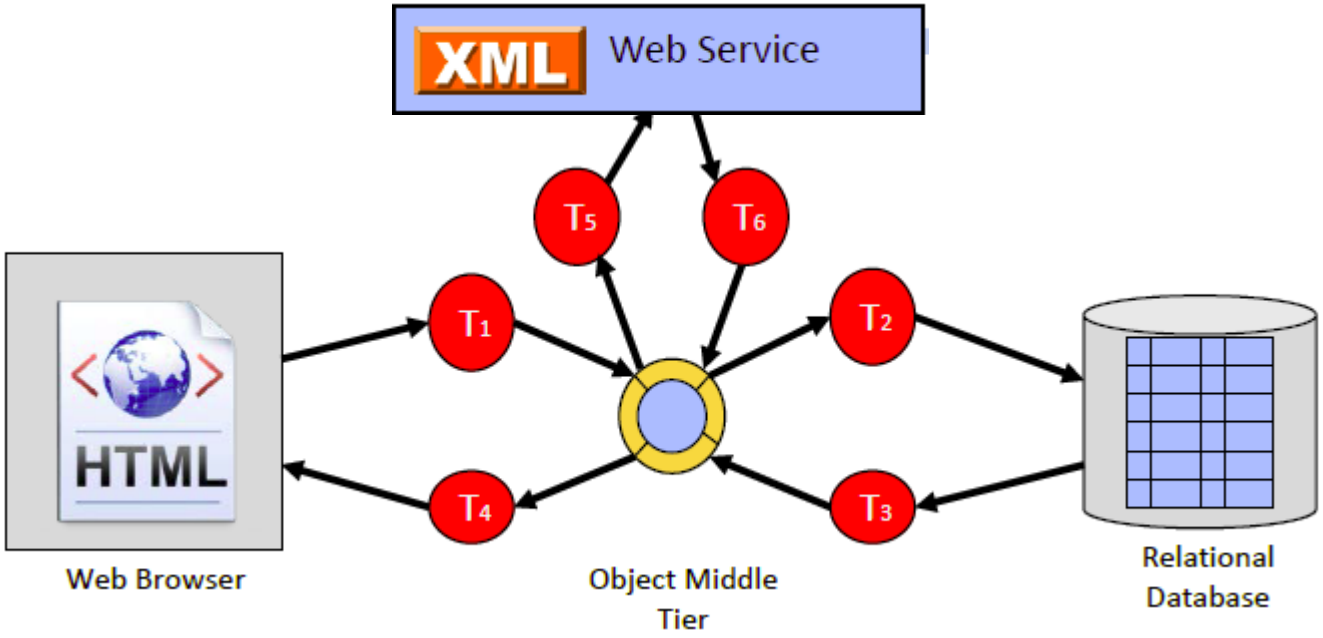
- Quickly maturing to include ACID transaction support
- Lack of object-relational mapping permits agile development
- Fastest growing revenues (MarkLogic, MongoDB, Couchbase)

Object Relational Mapping



- T1–HTML into Objects
- T2–Objects into SQL Tables
- T3–Tables into Objects
- T4–Objects into HTML

The Addition of XML Web Services

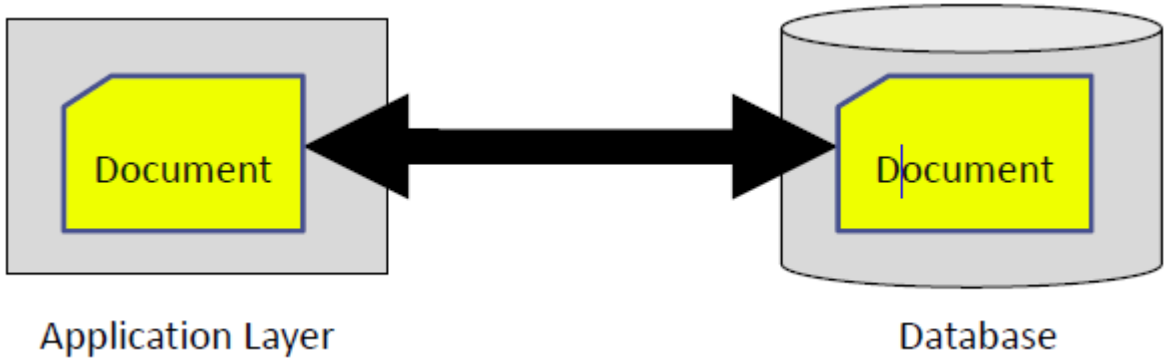


- T1–HTML into Java Objects
- T2–Java Objects into SQL Tables
- T3–Tables into Objects
- T4–Objects into HTML
- T5–Objects to XML
- T6–XML to Objects

Object-relational Mapping

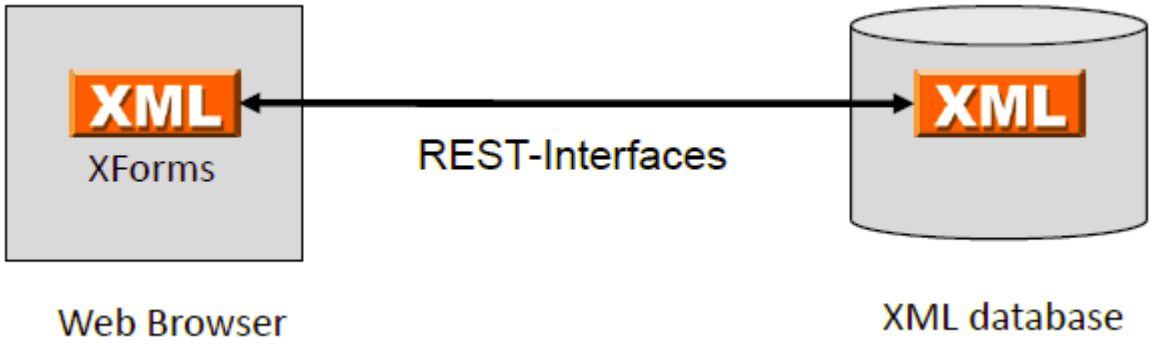
- Object-relational mapping has become one of the most complex components of building applications today
- A "Quagmire" where many projects get lost
- Sometimes the **best** way to avoid complexity is to keep your architecture very simple

Document Stores Need No Translation



- Documents in the database
- Documents in the application
- No object middle tier
- No "shredding"
- No reassembly
- Simple!

Zero Translation (XML)

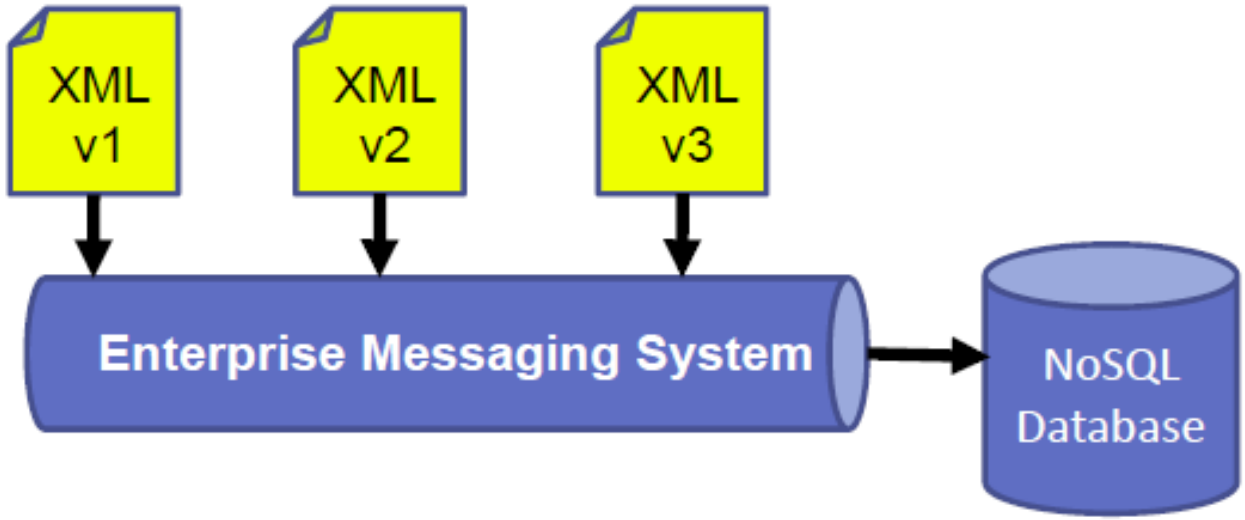


- XML lives in the web browser (**X**Forms)
- **R**EST interfaces
- XML in the database (Native XML, **X**Query)
- **XRX** Web Application Architecture
- No translation!

"Schema Free"

- Systems that automatically determine how to index data **as the data is loaded** into the database
- No ***a priori*** knowledge of data structure
- No need for up-front logical data modeling
- Adding new data elements or changing data elements is not disruptive
- Searching millions of records still has sub-second response time

Schema-Free Integration



"We can easily store the data that we **actually** get, not the data we **thought** we would get."

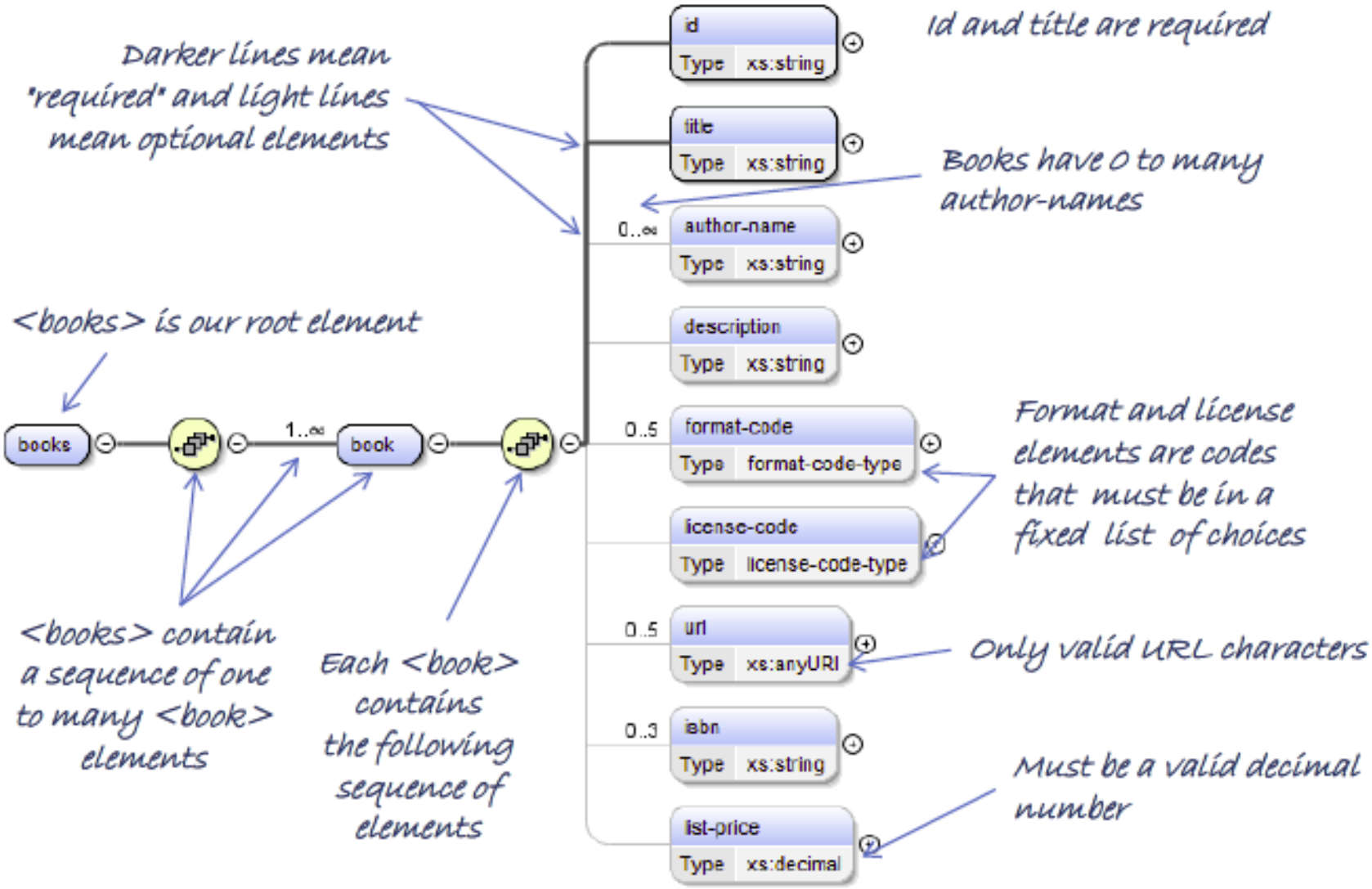
Upfront ER Modeling Not Required

- You do not have to **finish** modeling your data before you insert your first records
- No Data Definition Language "DDL" is needed
- Metadata is used to create indexes as data arrives

Upfront ER Modeling Not Required

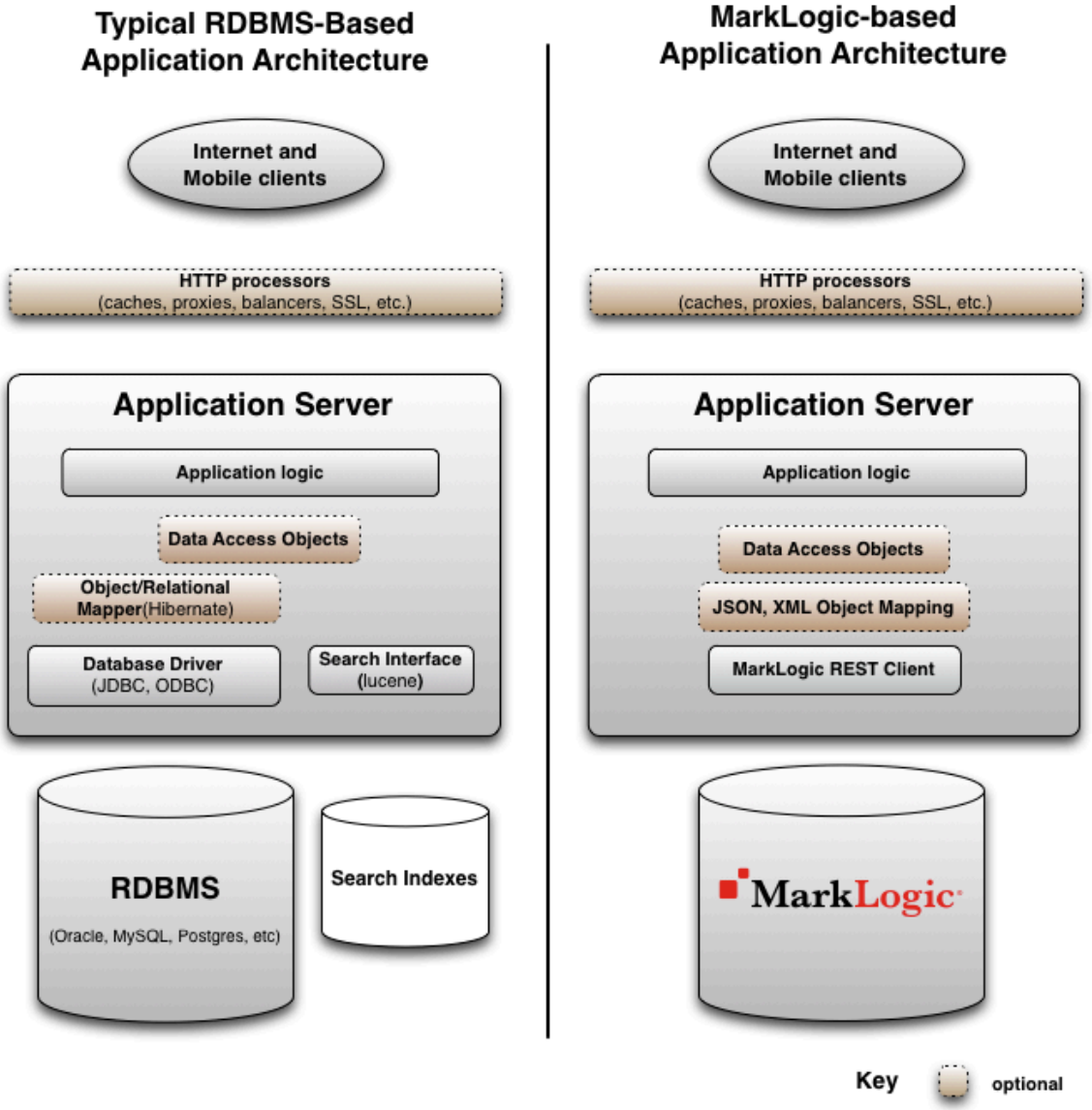
- Modeling becomes a statistical process –write queries to find exceptions and normalize data
- Exceptions make the rules but can still be used
- Data validation can still be done on documents using tools such as XML Schema and business rules systems like Schematron

Document Structure

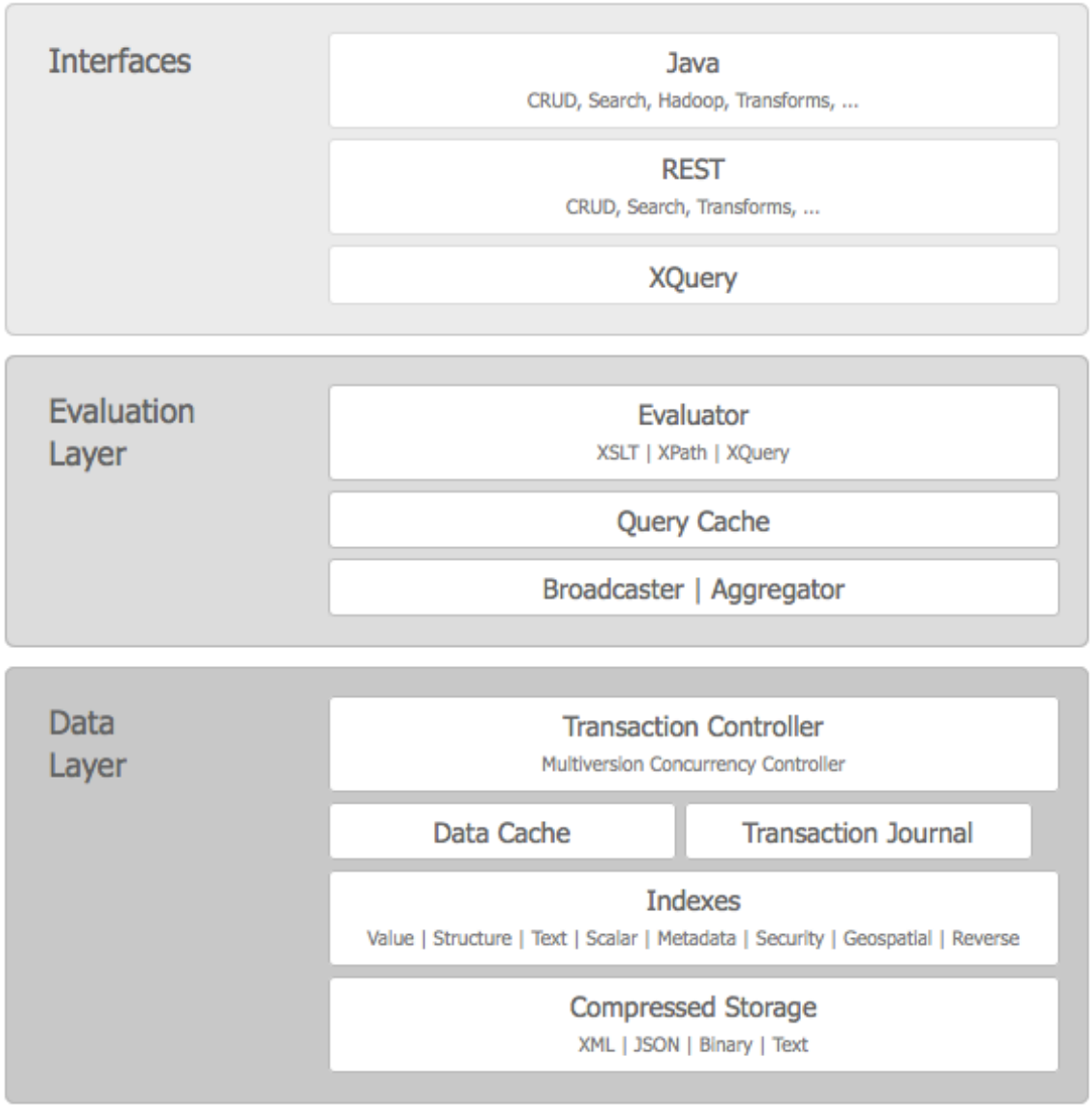


- Native XML database designed to scale to Petabyte data stores
- Leverages commodity hardware
- ACID compliant, schema-free document store
- Heavy use by federal agencies, document publishers and "high-variability" data

MarkLogic


Typical RDBMS-Based Application Architecture
Internet and Mobile clients
HTTP processors (caches, proxies, balancers, SSL, etc.)
Application Server
Application logic
Data Access Objects
Object/Relational Mapper(Hibernate)
Database Driver (JDBC, ODBC)
Search Interface (lucene)
RDBMS
(Oracle, MySQL, Postgres, etc)
Search Indexes
MarkLogic-based Application Architecture
Internet and Mobile clients
HTTP processors (caches, proxies, balancers, SSL, etc.)
Application Server
Application logic
Data Access Objects
JSON, XML Object Mapping
MarkLogic REST Client
MarkLogic
Key
optional

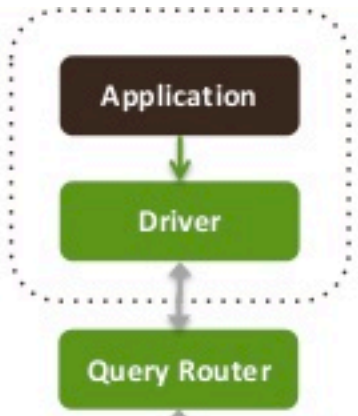
MarkLogic Architecture



- Open Source JSON data store created by 10gen
- Master-slave scale out model
- Strong developer community
- Sharding built-in, automatic
- Implemented in C++ with many APIs (C++, JavaScript, Java, Perl, Python etc.)

2. Native language drivers

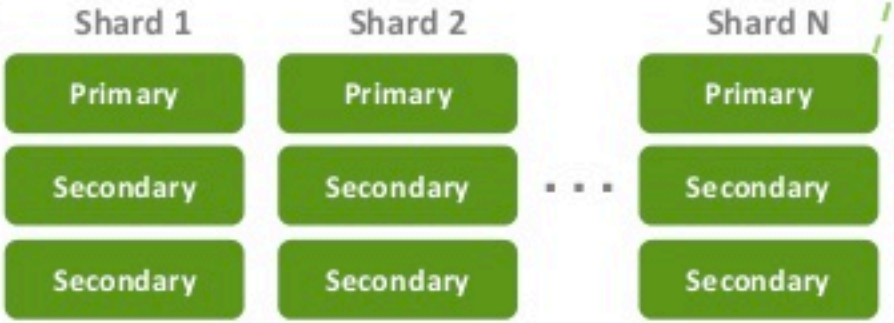
```
db.customer.insert({…})
db.customer.find({
 name: "John Smith"})
```



1. Dynamic Document Schema

```
{ name: "John Smith",
  date: "2013-08-01"),
  address: "10 3rd St.",
  phone: [
    { home: 1234567890},
    { mobile: 1234568138} ]
}
```

3. High availability
- Replica sets



4. High performance
- Data locality
- Rich Indexes
- RAM

5. Horizontal scalability
- Sharding

16

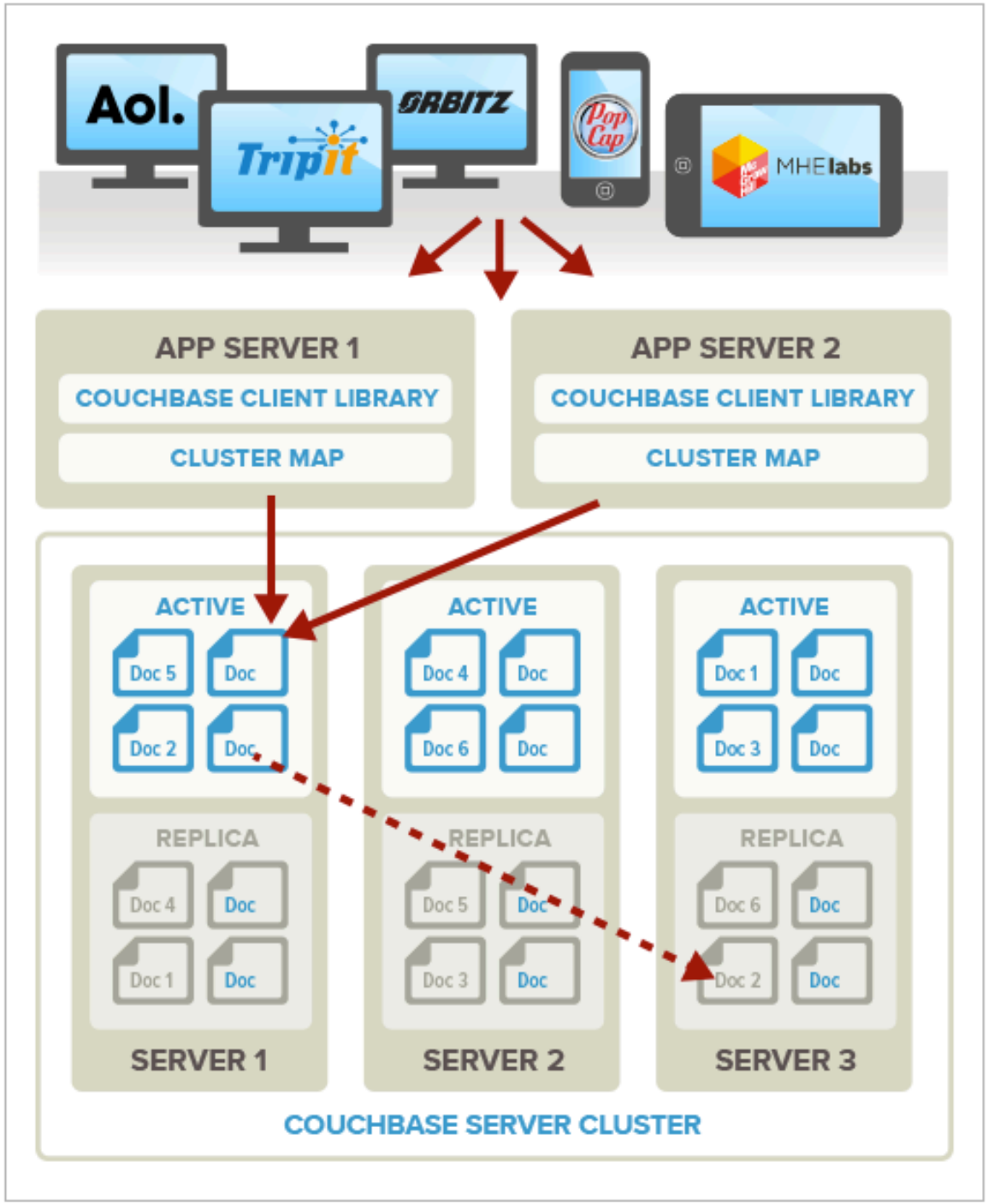
mongoDB

- Apache CouchDB
- Open source JSON data store
- Document Model
- Written in ERLANG
- RESTful JSON API
- B-Tree based indexing

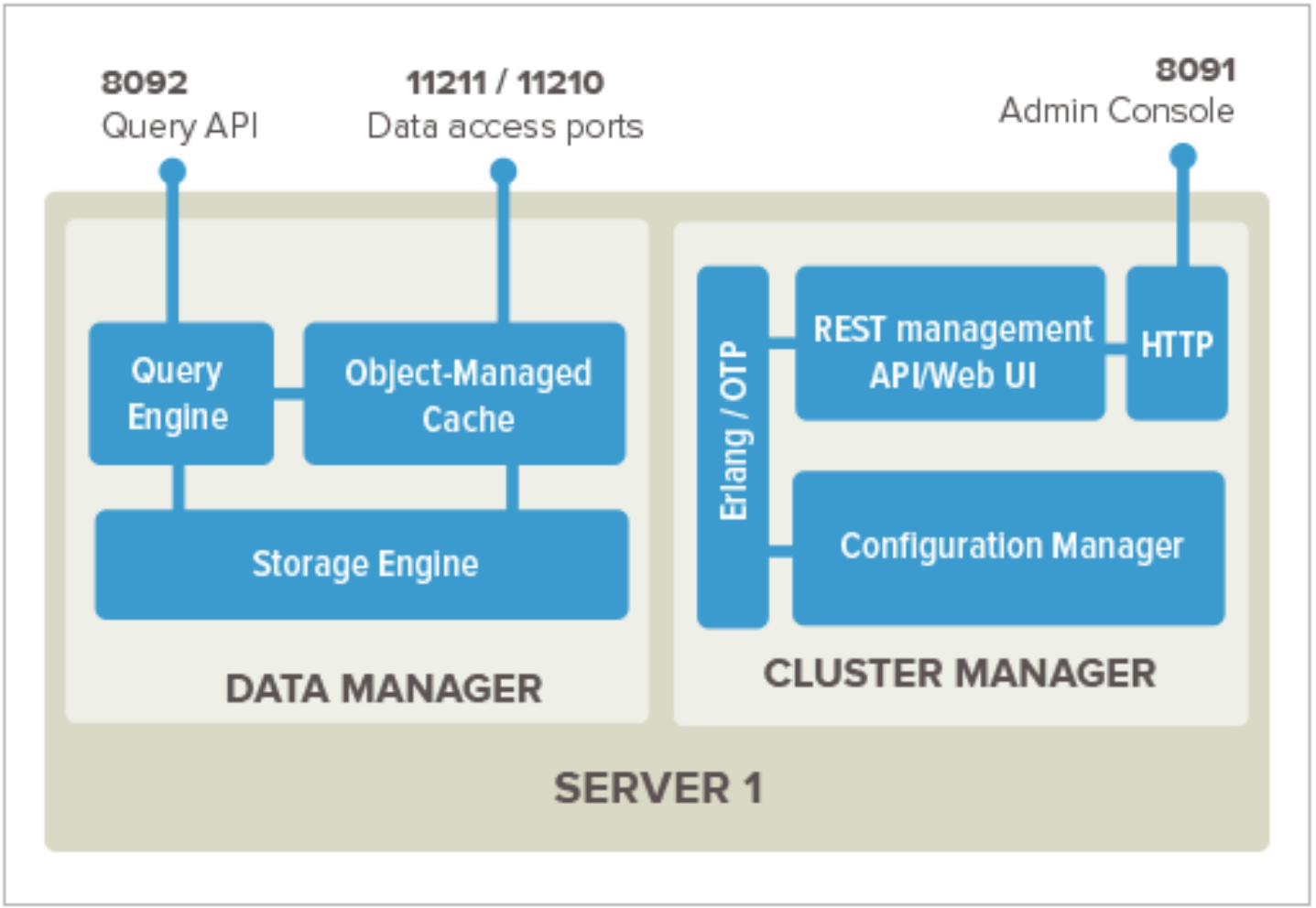
Couchbase

- Open source JSON document store
- Code base separate from CouchDB
- Built around memcached
- Peer to peer scale out model
- Written in C++ and Erlang
- Strengths in scale out, replication and high-availability

Couchbase

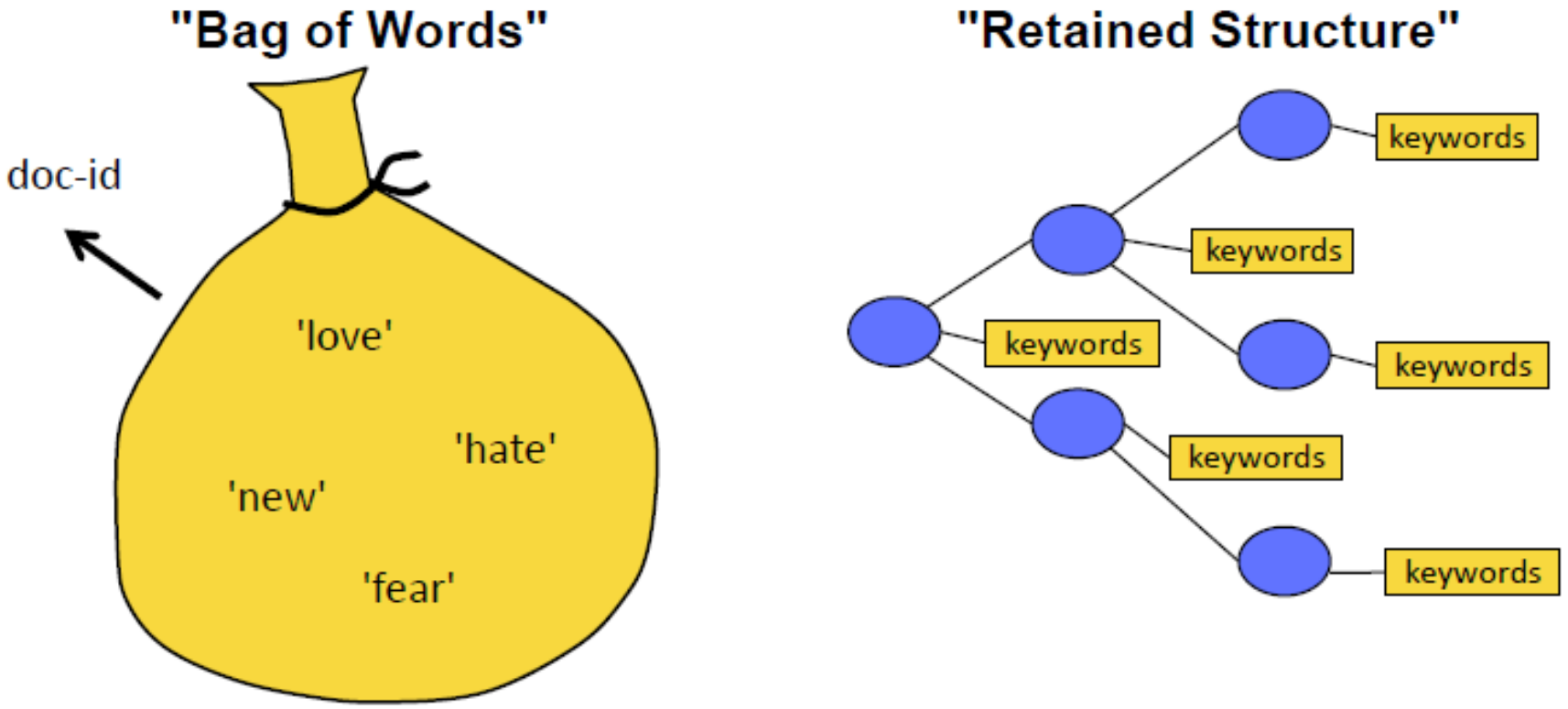

Aol.
TripIt
ORBITZ
PopCap
MHElabs
APP SERVER 1
COUCHBASE CLIENT LIBRARY
CLUSTER MAP
APP SERVER 2
COUCHBASE CLIENT LIBRARY
CLUSTER MAP
ACTIVE
Doc 5
Doc
Doc 2
Doc
REPLICA
Doc 4
Doc
Doc 1
Doc
SERVER 1
ACTIVE
Doc 4
Doc
Doc 6
Doc
REPLICA
Doc 5
Doc
Doc 3
Doc
SERVER 2
ACTIVE
Doc 1
Doc
Doc 3
Doc
REPLICA
Doc 6
Doc
Doc 2
Doc
SERVER 3
COUCHBASE SERVER CLUSTER

Couchbase


8092
Query API
11211 / 11210
Data access ports
8091
Admin Console
Query Engine
Object-Managed Cache
Storage Engine
DATA MANAGER
Erlang / OTP
REST management API/Web UI
HTTP
Configuration Manager
CLUSTER MANAGER
SERVER 1

- Open source native XML database
- Strong support for XQuery and XQuery extensions
- Heavily used by the Text Encoding Initiative (TEI) community and XRX/XForms communities
- Integrated Lucene search
- Collection triggers and versioning
- Extensive XQuery libs(EXPath)

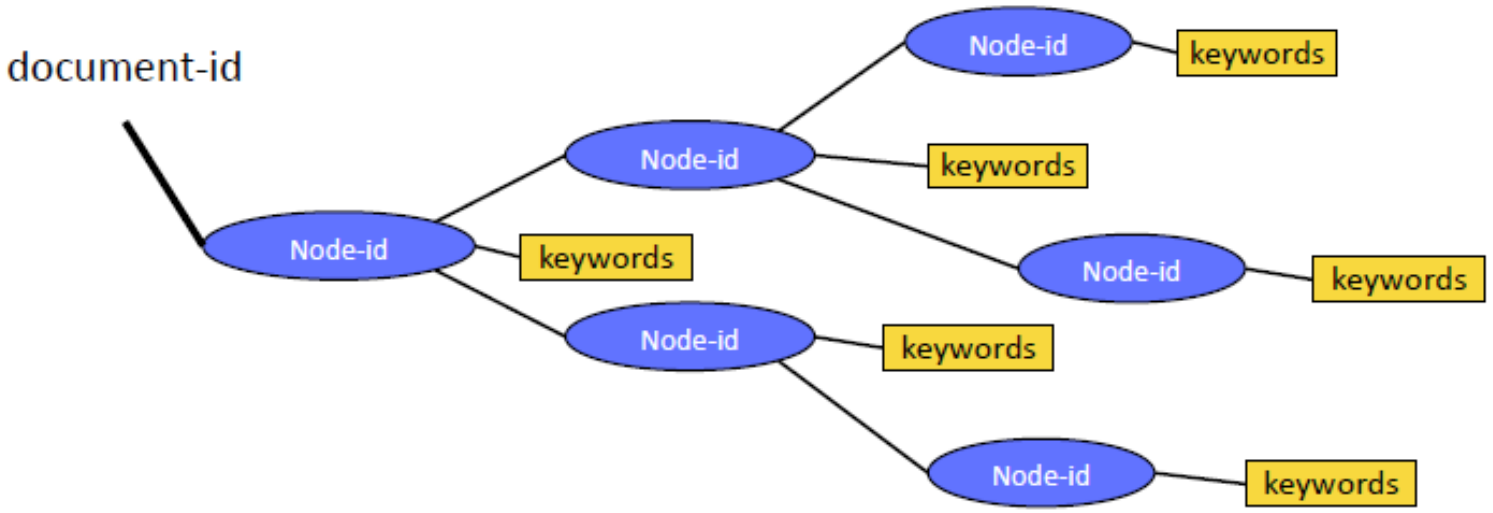
Two Models



- All keywords in a single container
- Only count frequencies are stored with each word

- Keywords associated with each sub-document component

Keywords and Node IDs



- Keywords in the reverse index are now associated with the **node-id** in every document

Hybrid Architectures

- Most real world implementations use some combination of NoSQL solutions
- Example
  - Use document stores for data
  - Use S3 for image/pdf/binary storage
  - Use Apache Lucene for document index stores
  - Use MapReduce for real-time index and aggregate creation and maintenance
  - Use OLAP for reporting sums and totals

Summary

Summary

- We covered the Pros and Cons of NoSQL
- We explored the major projects / implementations of NoSQL:
  - K-V stores
  - Column stores
  - Graph DB
  - Document stores

END