NoSQL Databases

Amir H. Payberah
Swedish Institute of Computer Science

amir@sics.se
April 10, 2014

Database and Database Management System

- Database: an organized collection of data.

Database and Database Management System

- Database: an organized collection of data.

- Database Management System (DBMS): a software that interacts with users, other applications, and the database itself to capture and analyze data.

History of Databases

- 1960s
  - Navigational data model: hierarchical model (IMS) and network model (CODASYL).
  - Disk-aware

History of Databases

- 1960s
  - Navigational data model: hierarchical model (IMS) and network model (CODASYL).
  - Disk-aware
- 1970s
  - Relational data model: Edgar F. Codd paper.
  - Logical data is disconnected from physical information storage.

History of Databases

- 1960s
  - Navigational data model: hierarchical model (IMS) and network model (CODASYL).
  - Disk-aware
- 1970s
  - Relational data model: Edgar F. Codd paper.
  - Logical data is disconnected from physical information storage.
- 1980s
  - Object databases: information is represented in the form of objects.

History of Databases

- 1960s
  - Navigational data model: hierarchical model (IMS) and network model (CODASYL).
  - Disk-aware
- 1970s
  - Relational data model: Edgar F. Codd paper.
  - Logical data is disconnected from physical information storage.
- 1980s
  - Object databases: information is represented in the form of objects.
- 2000s
  - NoSQL databases: BASE instead of ACID.
  - NewSQL databases: scalable performance of NoSQL + ACID.

Relational Databases Management Systems (RDMBSs)

- RDMBSs: the dominant technology for storing structured data in web and business applications.

- SQL is good
  - Rich language
  - Easy to use and integrate
  - Rich toolset
  - Many vendors

- They promise: ACID

ACID Properties

- Atomicity
  - All included statements in a transaction are either executed or the whole transaction is aborted without affecting the database.

ACID Properties

- Atomicity
  - All included statements in a transaction are either executed or the whole transaction is aborted without affecting the database.
- Consistency
  - A database is in a consistent state before and after a transaction.

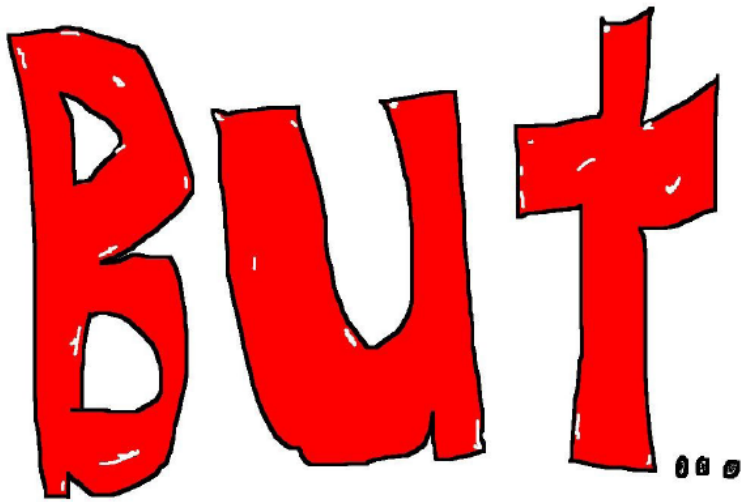
ACID Properties

- Atomicity
  - All included statements in a transaction are either executed or the whole transaction is aborted without affecting the database.
- Consistency
  - A database is in a consistent state before and after a transaction.
- Isolation
  - Transactions can not see uncommitted changes in the database.

ACID Properties

- Atomicity
  - All included statements in a transaction are either executed or the whole transaction is aborted without affecting the database.
- Consistency
  - A database is in a consistent state before and after a transaction.
- Isolation
  - Transactions can not see uncommitted changes in the database.
- Durability
  - Changes are written to a disk before a database commits a transaction so that committed data cannot be lost through a power failure.

RDBMS is Good ...



RDBMS Challenges

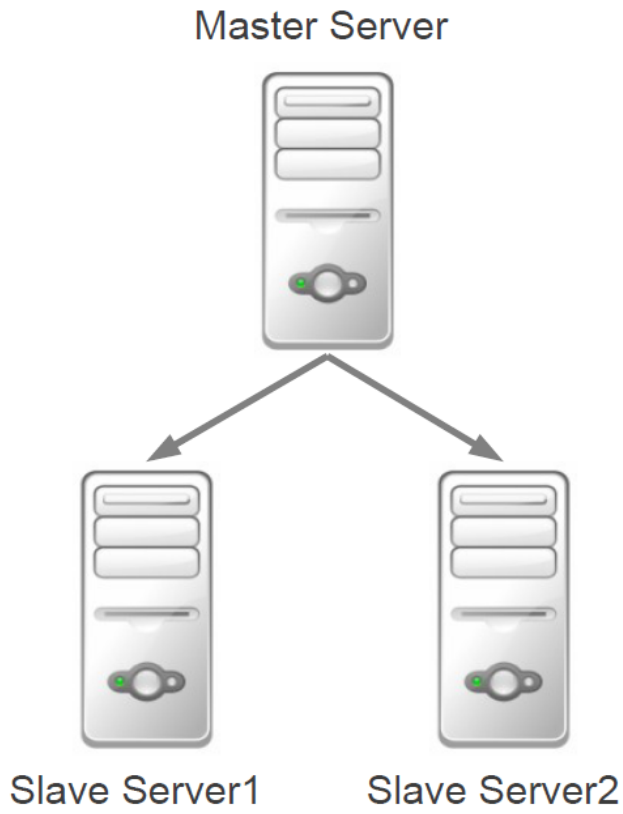
- Web-based applications caused spikes.
  - Internet-scale data size
  - High read-write rates
  - Frequent schema changes

Let's Scale RDBMSs

- RDBMS were not designed to be distributed.
- Possible solutions:
  - Replication
  - Sharding

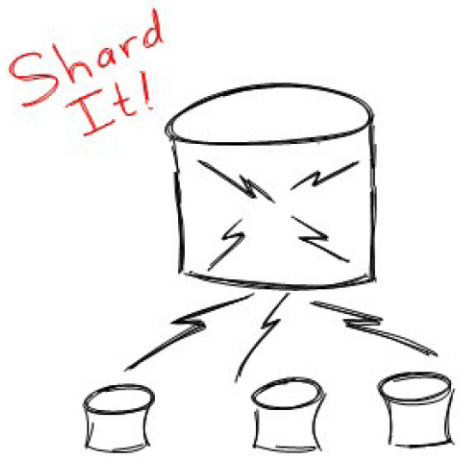
Let's Scale RDBMSs - Replication

- Master/Slave architecture
- Scales read operations

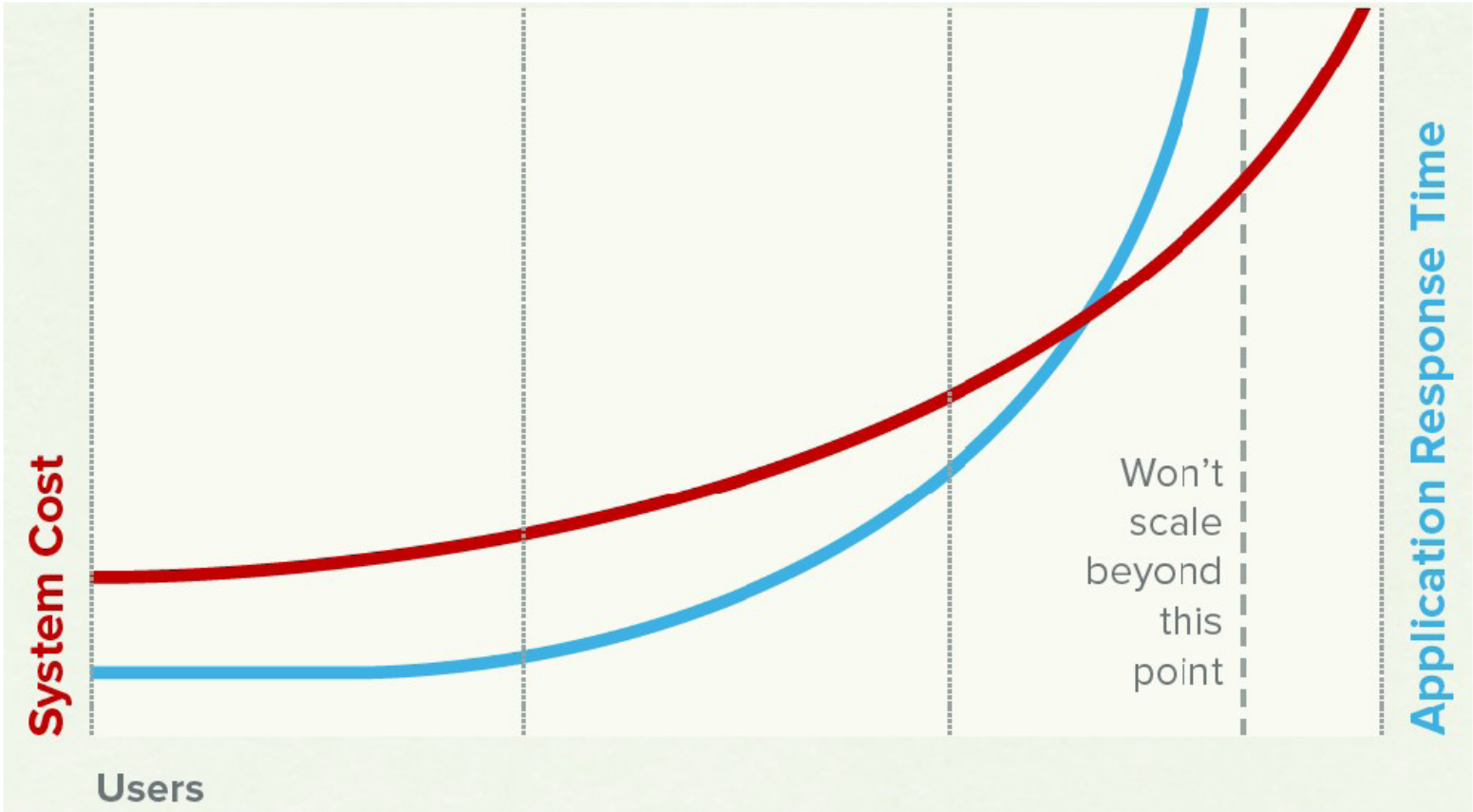


Let's Scale RDBMSs - Sharding

- Dividing the database across many machines.
- It scales read and write operations.
- Cannot execute transactions across shards (partitions).



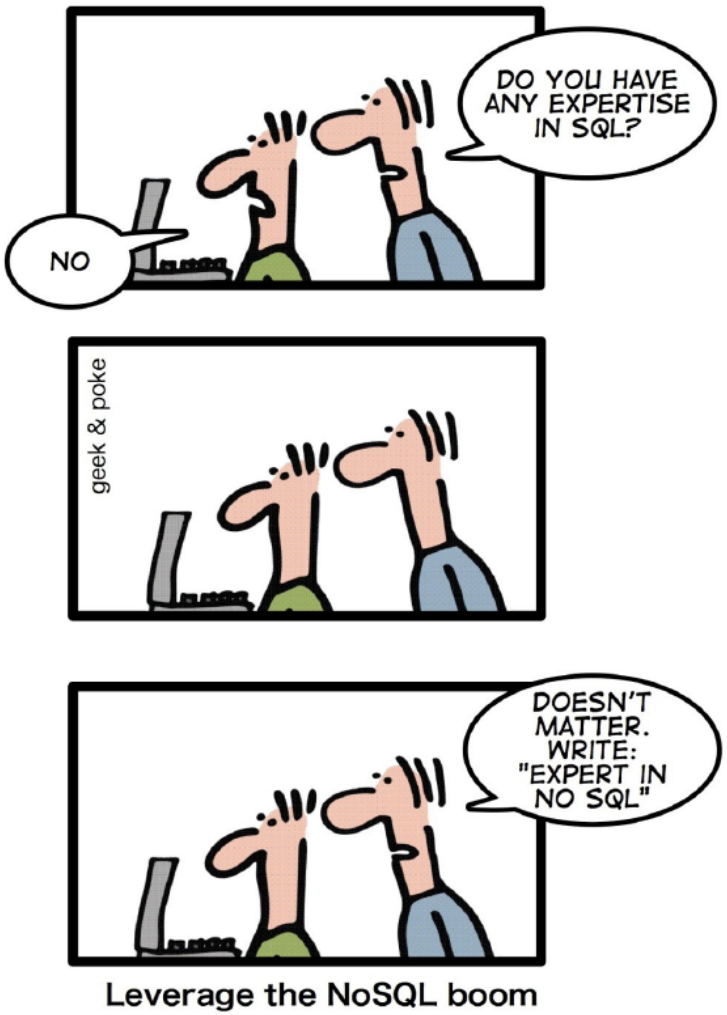
Scaling RDBMSs is Expensive and Inefficient



[http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQLWhitepaper.pdf]


Not
Only
SQL

HOW TO WRITE A CV



Leverage the NoSQL boom

NoSQL

- Avoidance of unneeded complexity
- High throughput
- Horizontal scalability and running on commodity hardware
- Compromising reliability for better performance

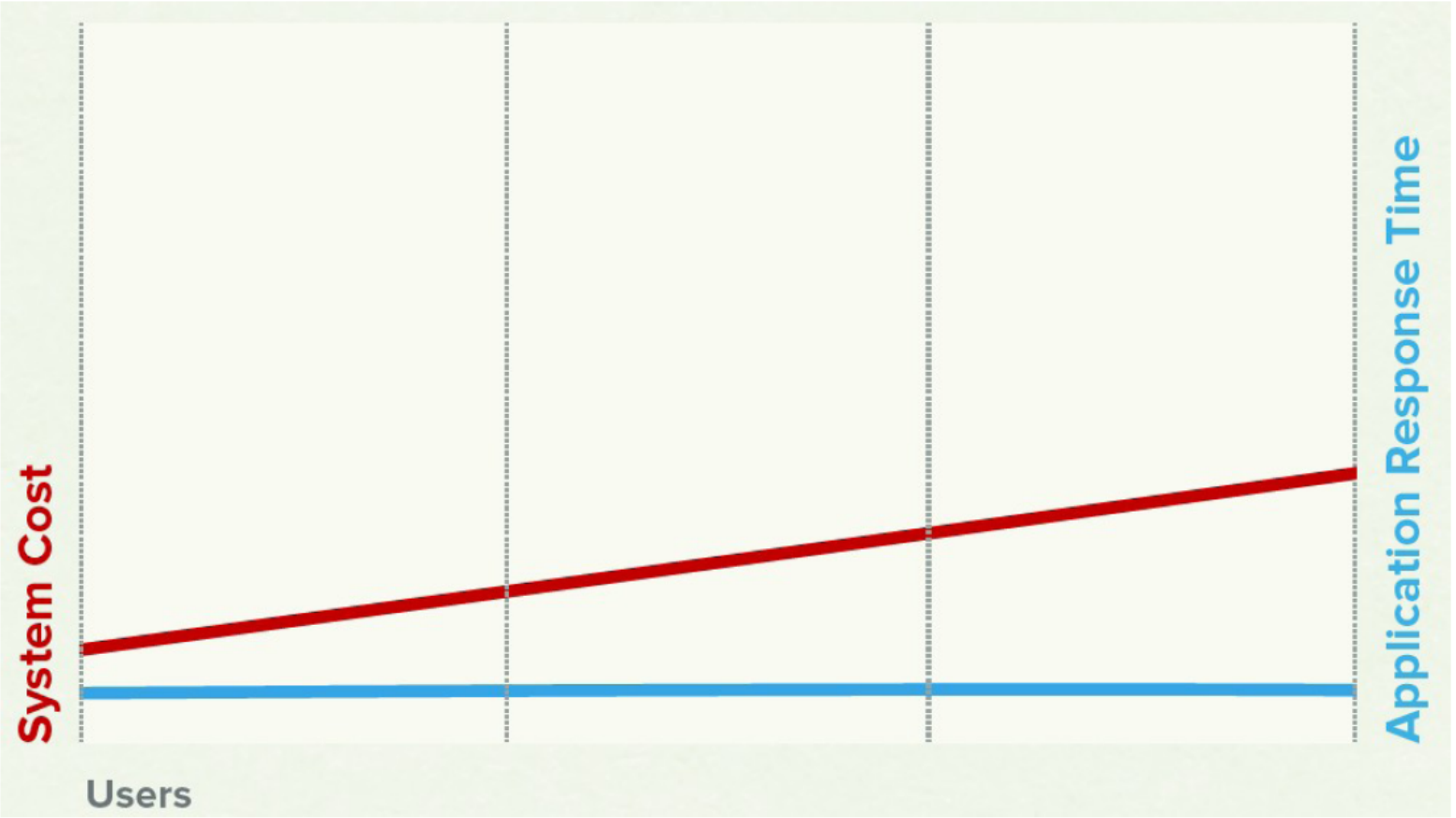
The NoSQL Movement

- It was first used in 1998 by Carlo Strozzi to name his relational database that did not expose the standard SQL interface.

- The term was picked up again in 2009 when a Last.fm developer, Johan Oskarsson, wanted to organize an event to discuss open source distributed databases.

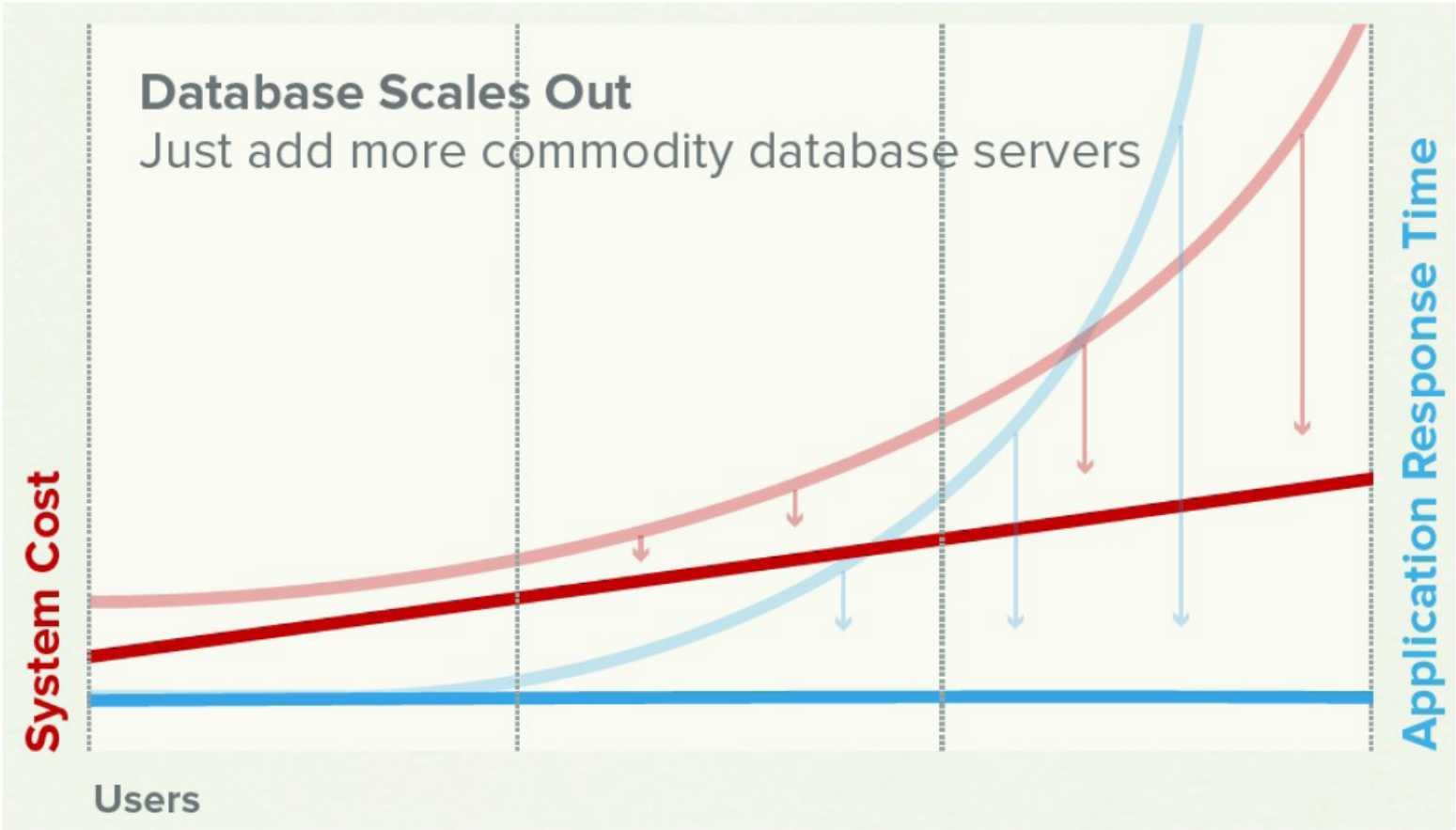
- The name attempted to label the emergence of a growing number of non-relational, distributed data stores that often did not attempt to provide ACID.

NoSQL Cost and Performance



[http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQLWhitepaper.pdf]

RDBMS vs. NoSQL



[http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQLWhitepaper.pdf]

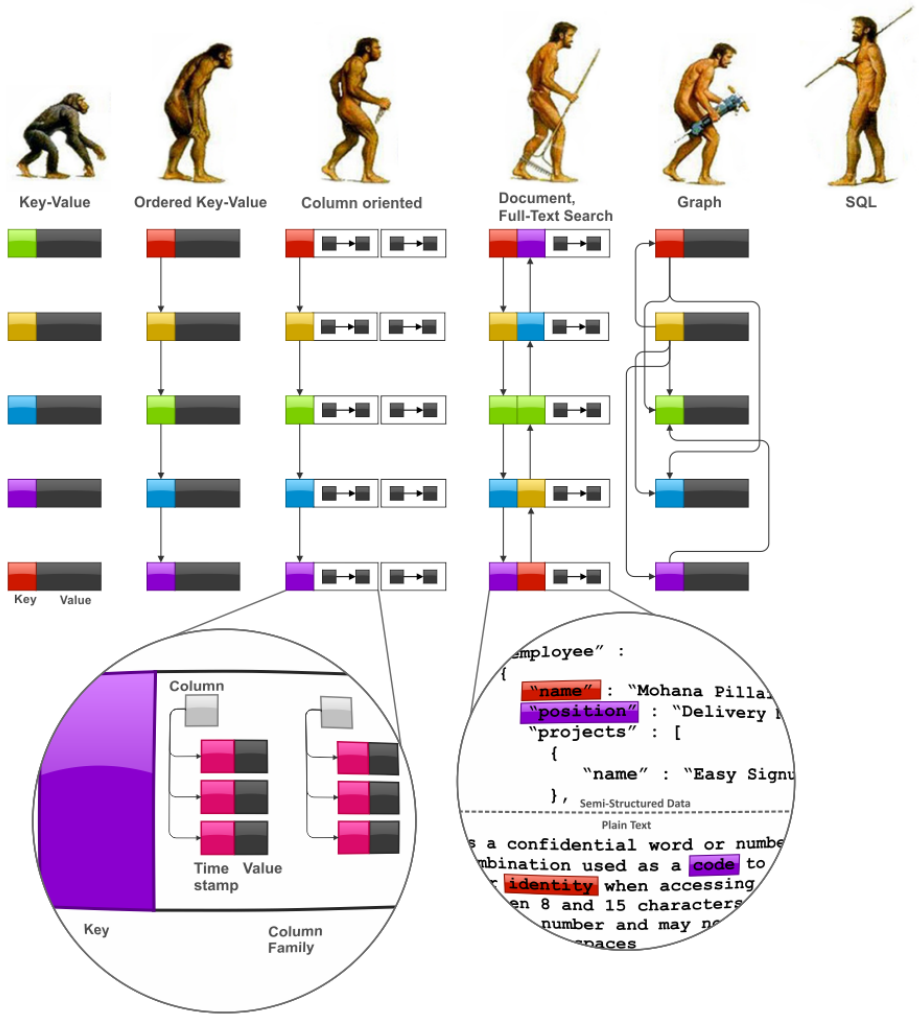
Basic Concepts

- Data model
- Partitioning
- Consistency

Basic Concepts

- Data model
- Partitioning
- Consistency

NoSQL Data Models



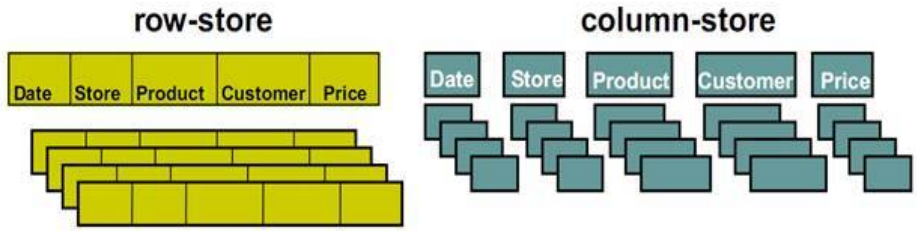
[http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques]

Key-Value Data Model

- Collection of key/value pairs.
- Ordered Key-Value: processing over key ranges.
- Dynamo, Scalaris, Voldemort, Riak, ...

Column-Oriented Data Model

- Similar to a key/value store, but the value can have multiple attributes (Columns).
- Column: a set of data values of a particular type.
- Store and process data by column instead of row.
- BigTable, Hbase, Cassandra, ...



Document Data Model

- Similar to a column-oriented store, but values can have complex documents, instead of fixed format.
- Flexible schema.
- XML, YAML, JSON, and BSON.
- CouchDB, MongoDB, ...

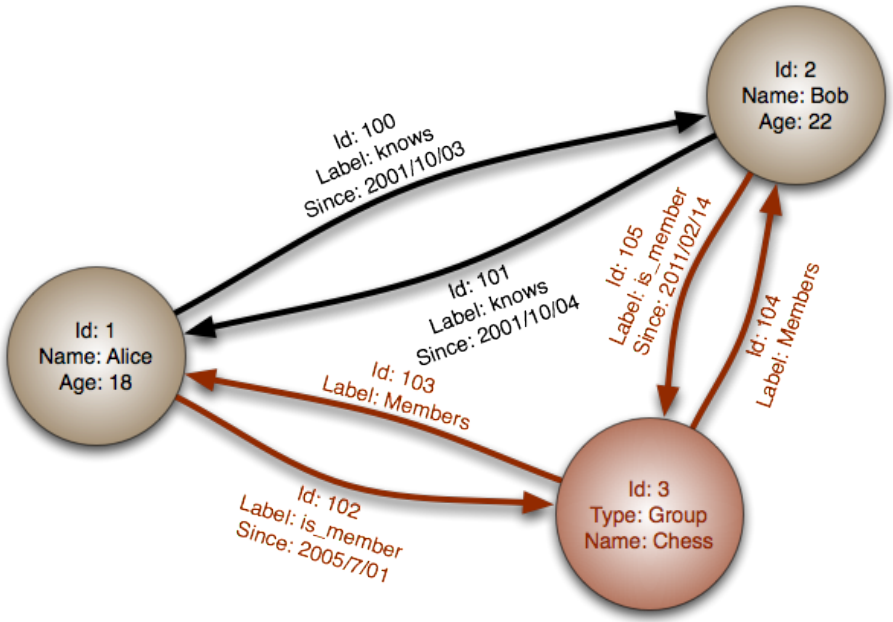
```
{
    FirstName: "Bob",
    Address: "5 Oak St.",
    Hobby: "sailing"
 }

 {
   FirstName: "Jonathan",
   Address: "15 Wanamassa Point Road",
   Children: [
        {Name: "Michael", Age: 10},
        {Name: "Jennifer", Age: 8},
   ]
  }
```

Graph Data Model

- Uses graph structures with nodes, edges, and properties to represent and store data.

- Neo4J, InfoGrid, ...



[http://en.wikipedia.org/wiki/Graph_database]

Basic Concepts

- Data model
- Partitioning
- Consistency

Partitioning

- If size of data exceeds the capacity of a single machine: partitioning
- Sharding data (horizontal partitioning).
- Consistent hashing is one form of automatic sharding.

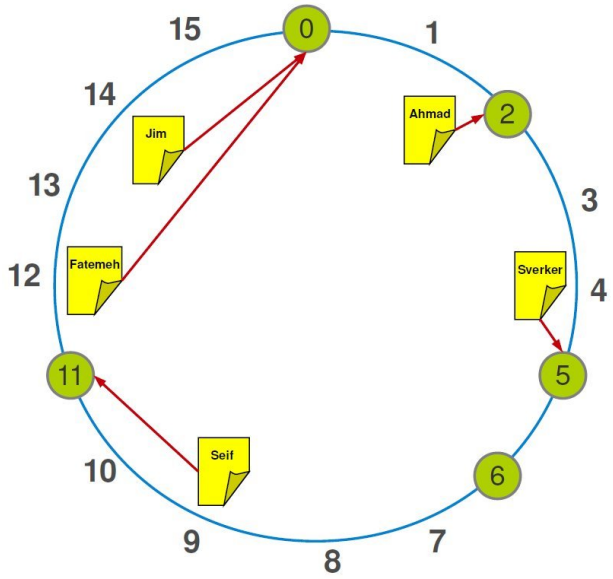
Consistent Hashing

- Hash both data and nodes using the same hash function in a same id space.
- `partition = hash(d) mod n`, `d`: data, `n`: number of nodes

Consistent Hashing

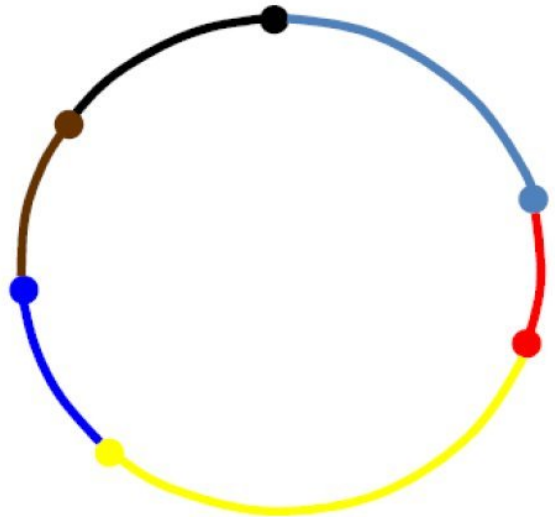
- Hash both data and nodes using the same hash function in a same id space.
- `partition = hash(d) mod n`, `d`: data, `n`: number of nodes

```
hash("Fatemeh") = 12
hash("Ahmad") = 2
hash("Seif") = 9
hash("Jim") = 14
hash("Sverker") = 4
```



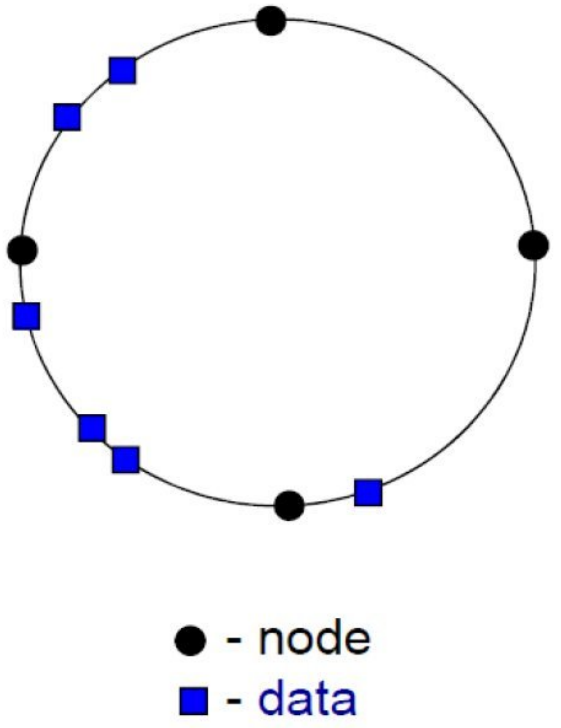
Load Imbalance (1/4)

- Consistent hashing may lead to imbalance.
- Node identifiers may not be balanced.



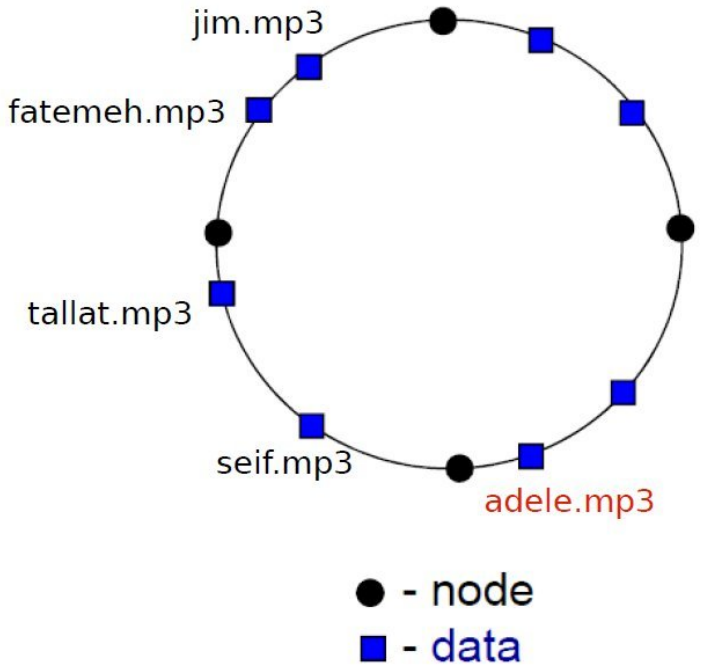
Load Imbalance (2/4)

- Consistent hashing may lead to imbalance.
- Data identifiers may not be balanced.



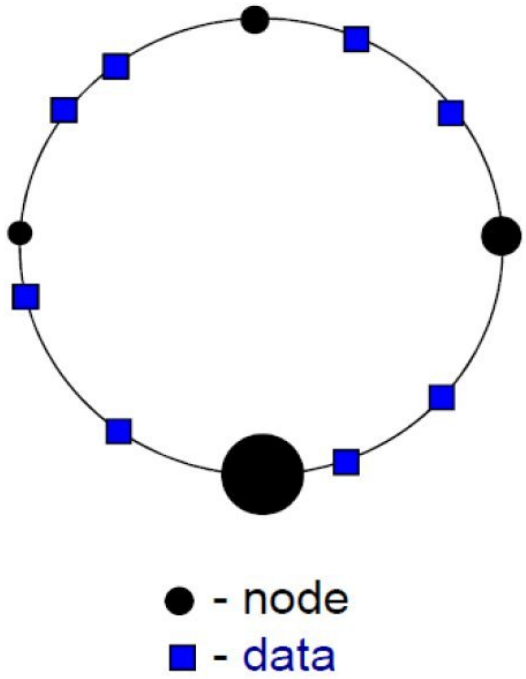
Load Imbalance (3/4)

- Consistent hashing may lead to imbalance.
- Hot spots.



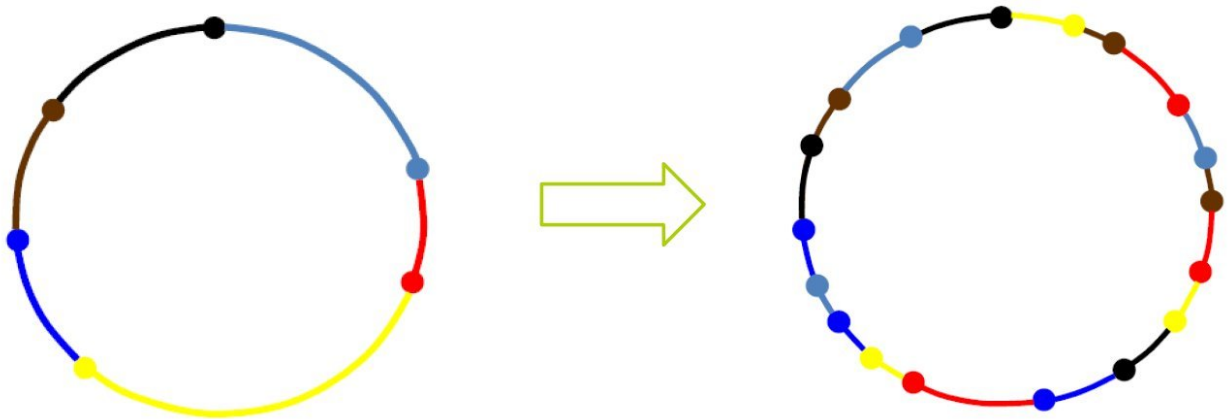
Load Imbalance (4/4)

- Consistent hashing may lead to imbalance.
- Heterogeneous nodes.



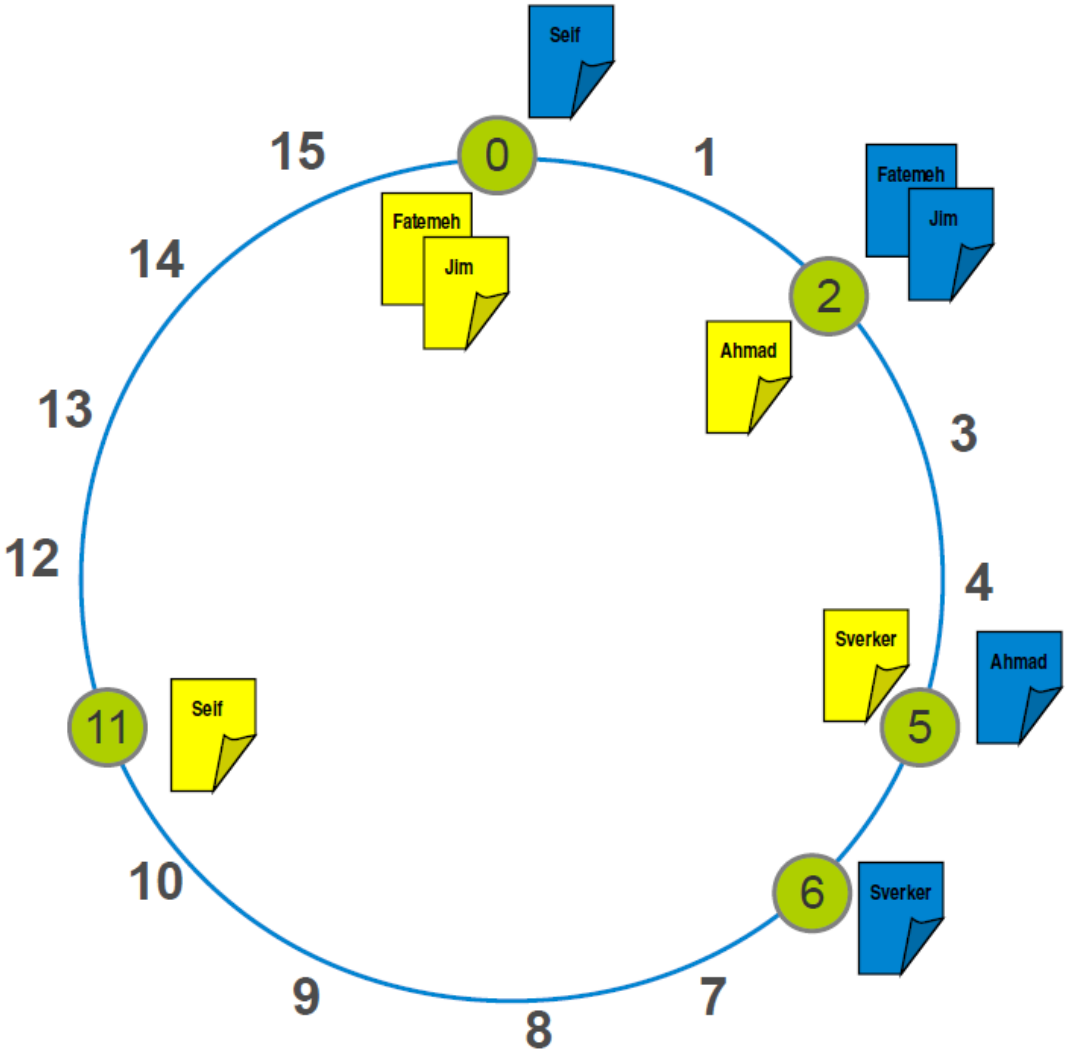
Load Balancing via Virtual Nodes

- Each physical node picks multiple random identifiers.
- Each identifier represents a virtual node.
- Each node runs multiple virtual nodes.



Replication

- To achieve high availability and durability, data should be replicates on multiple nodes.

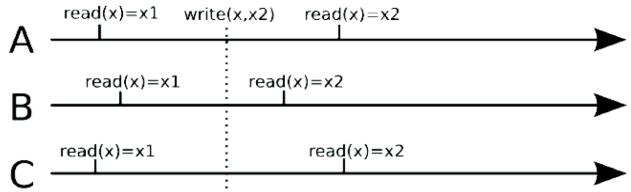


Basic Concepts

- Data model
- Partitioning
- Consistency

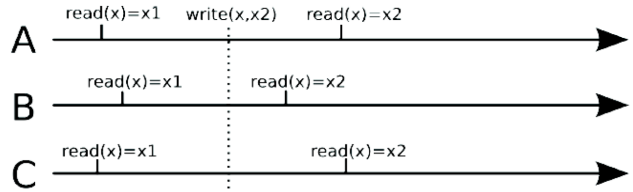
Consistency

- Strong consistency
  - After an update completes, any subsequent access will return the updated value.

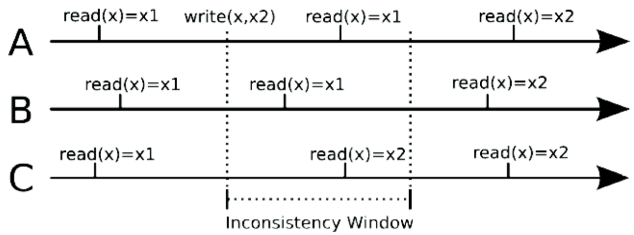


Consistency

- Strong consistency
  - After an update completes, any subsequent access will return the updated value.



- Eventual consistency
  - Does not guarantee that subsequent accesses will return the updated value.
  - Inconsistency window.
  - If no new updates are made to the object, eventually all accesses will return the last updated value.



Quorum Model

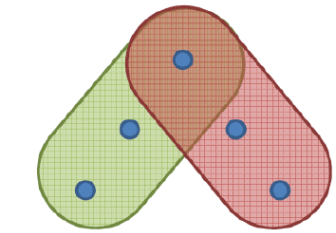
- N: the number of nodes to which a data item is replicated.
- R: the number of nodes a value has to be read from to be accepted.
- W: the number of nodes a new value has to be written to before the write operation is finished.

- To enforce strong consistency: $R + W > N$

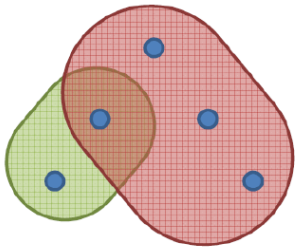
Quorum Model

- N: the number of nodes to which a data item is replicated.
- R: the number of nodes a value has to be read from to be accepted.
- W: the number of nodes a new value has to be written to before the write operation is finished.

- To enforce strong consistency: $R + W > N$



R = 3, W = 3, N = 5



R = 4, W = 2, N = 5

Consistency vs. Availability

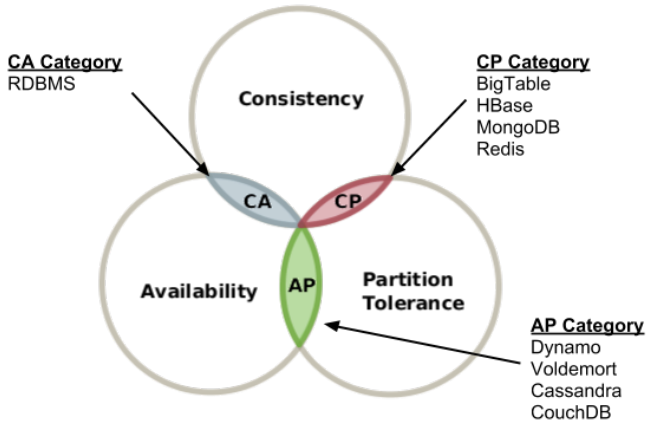
- The large-scale applications have to be reliable: availability + redundancy
- These properties are difficult to achieve with ACID properties.
- The BASE approach forfeits the ACID properties of consistency and isolation in favor of availability, graceful degradation, and performance.

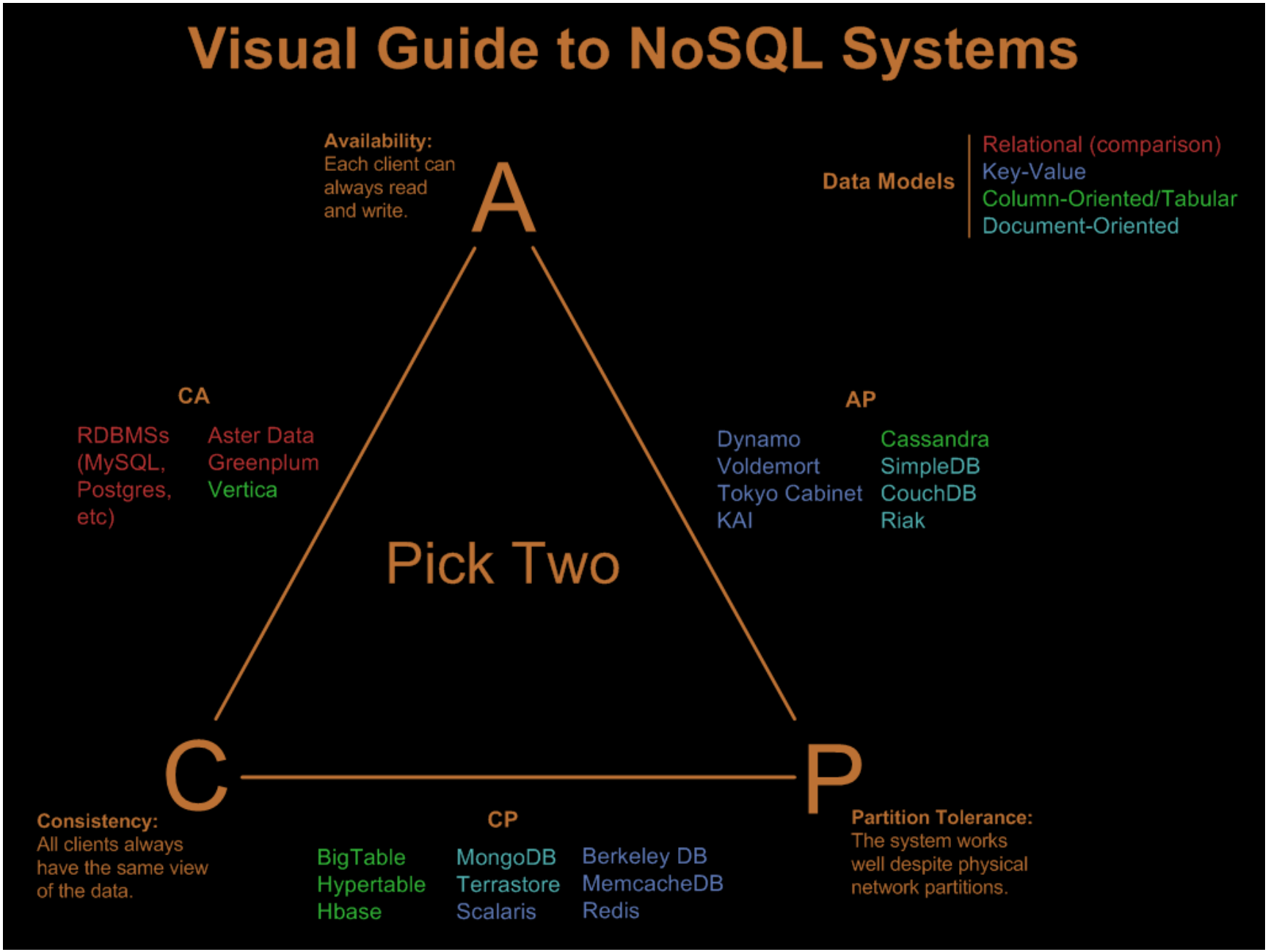
BASE Properties

- Basic Availability
  - Possibilities of faults but not a fault of the whole system.
- Soft-state
  - Copies of a data item may be inconsistent
- Eventually consistent
  - Copies becomes consistent at some later time if there are no more updates to that data item

CAP Theorem

- Consistency
  - Consistent state of data after the execution of an operation.
- Availability
  - Clients can always read and write data.
- Partition Tolerance
  - Continue the operation in the presence of network partitions.
- You can choose only two!




Visual Guide to NoSQL Systems
Availability:
Each client can always read and write.
A
Data Models
Relational (comparison)
Key-Value
Column-Oriented/Tabular
Document-Oriented
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick Two
C
P
Consistency:
All clients always have the same view of the data.
CP
BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance:
The system works well despite physical network partitions.

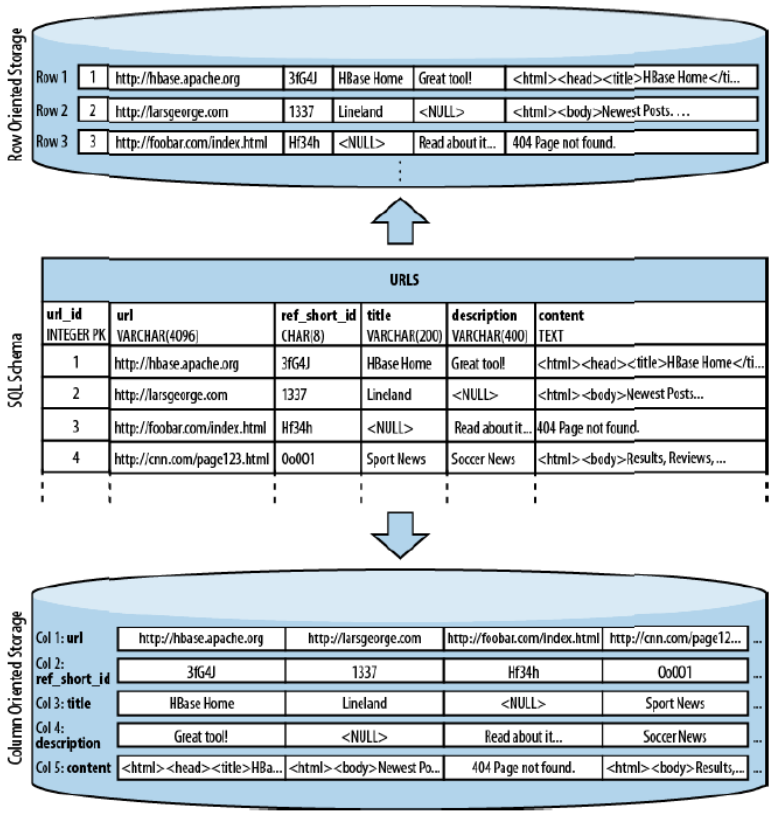

APACHE
HBASE

HBase

- Type of NoSQL database, based on Google Bigtable
- Column-oriented data store, built on top of HDFS
- CAP: strong consistency and partition tolerance

Columns-Oriented Databases

- In many analytical databases queries, few attributes are needed.
- Column values are stored contiguously on disk: reduces I/O.



[Lars George, "Hbase: The Definitive Guide", O'Reilly, 2011]

HBase Data Model (1/5)

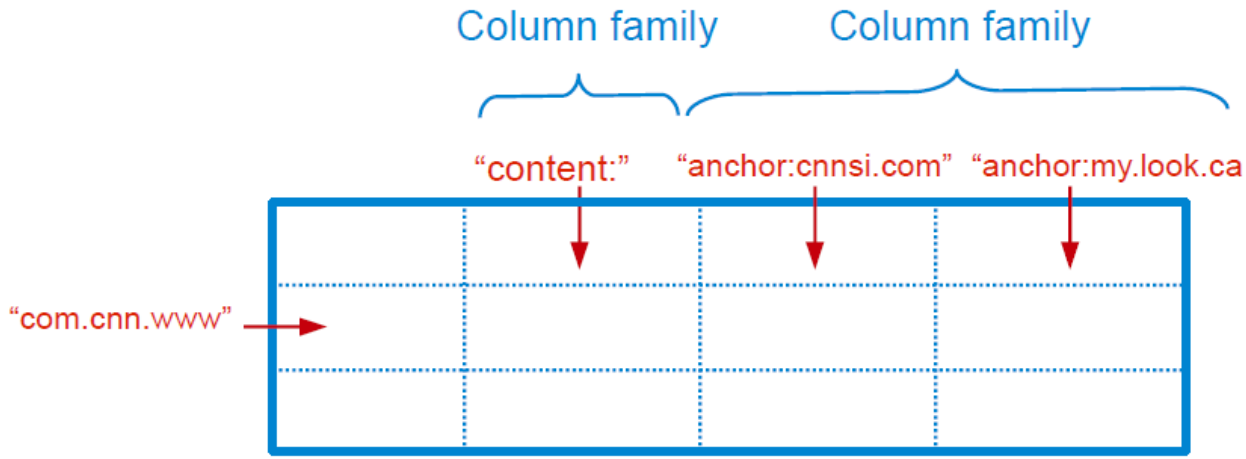
- Table
- Distributed multi-dimensional sparse map

HBase Data Model (2/5)

- Rows
- Every read or write in a row is atomic.
- Rows sorted in lexicographical order.

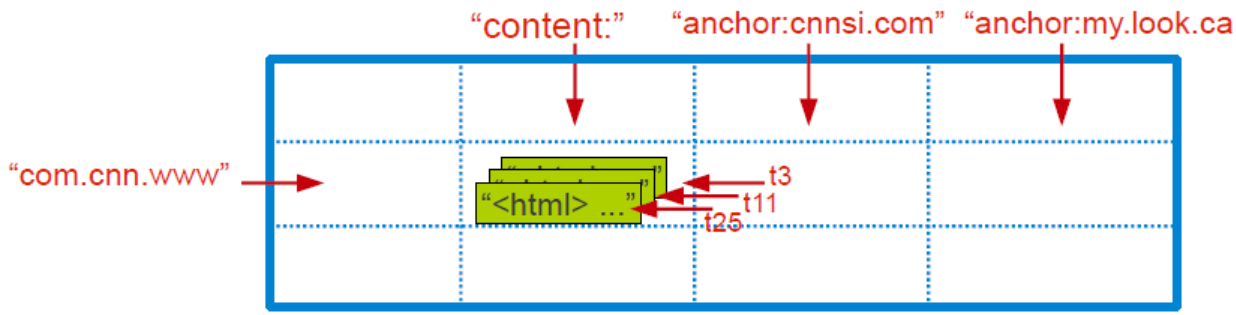
HBase Data Model (3/5)

- Column
- The basic unit of data access.
- Column families: group of (the same type) column keys.
- Column key naming: family:qualifier
- All columns in a column family are stored together in the same low level storage file (HFile).



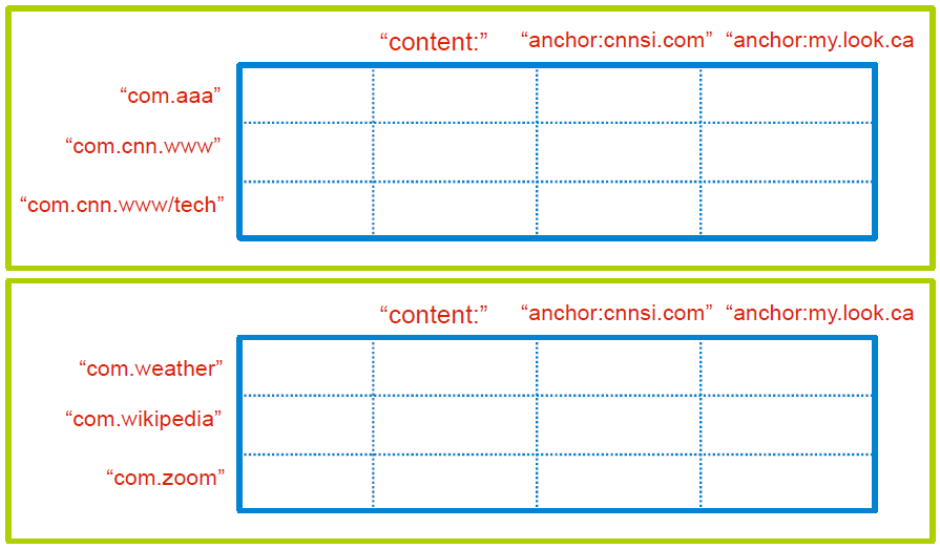
HBase Data Model (4/5)

- Timestamp
- Each column value may contain multiple versions.

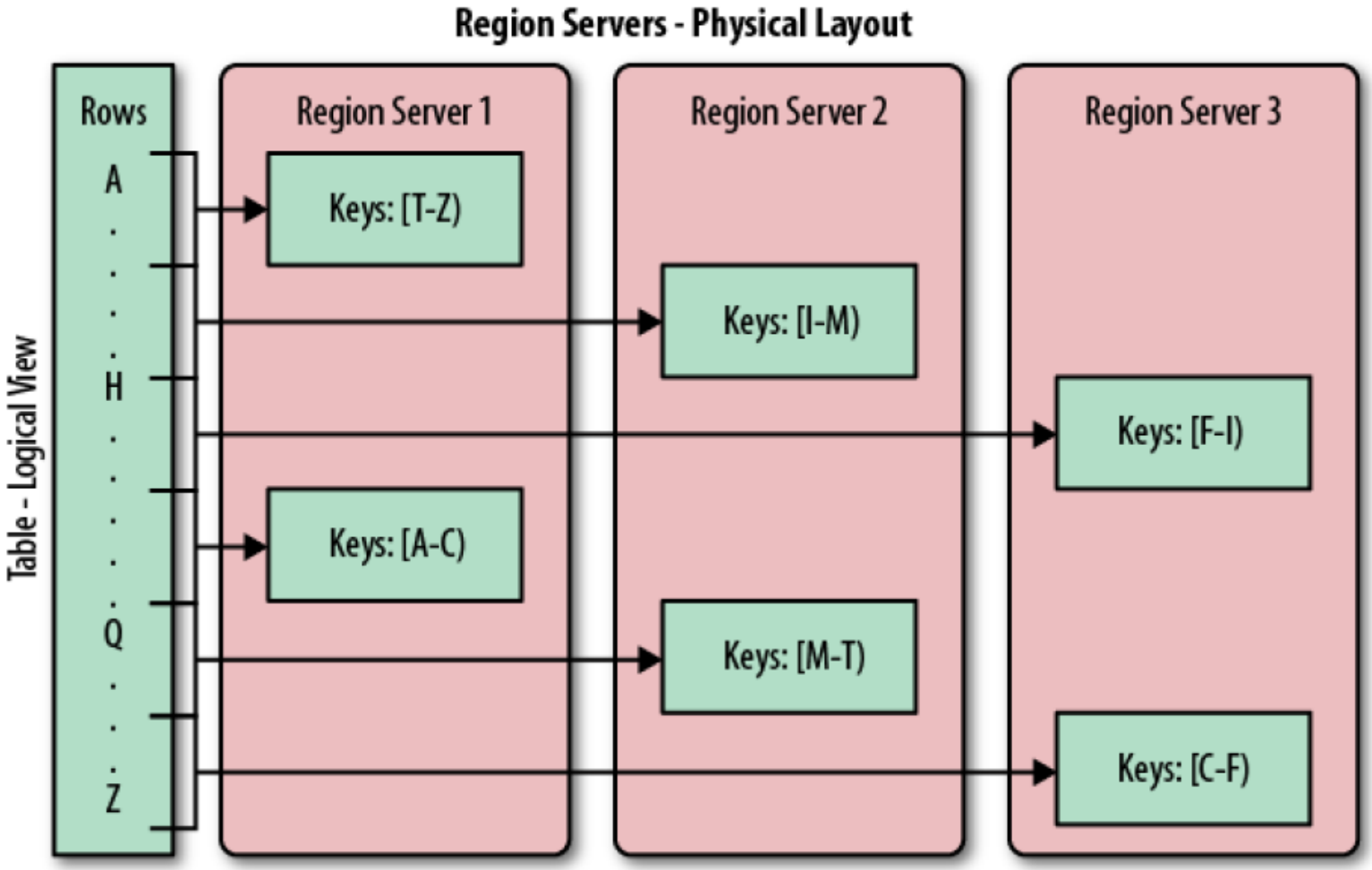


HBase Data Model (5/5)

- Region: contiguous ranges of rows stored together.
- Tables are split by the system when they become too large.
- Auto-Sharding
- Each region is served by exactly one region server.

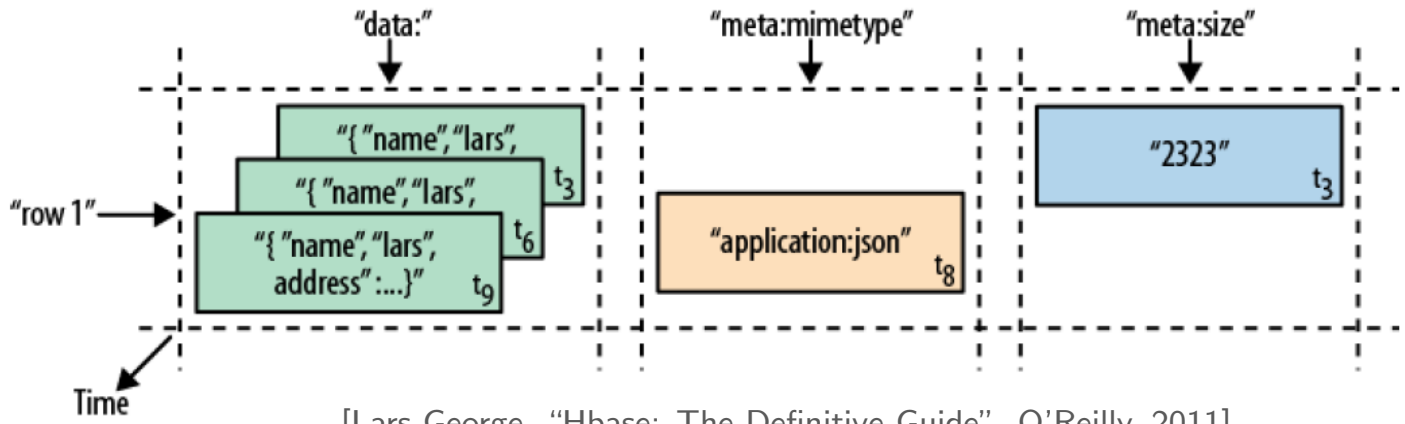


Region and Region Server



[Lars George, "Hbase: The Definitive Guide", O'Reilly, 2011]

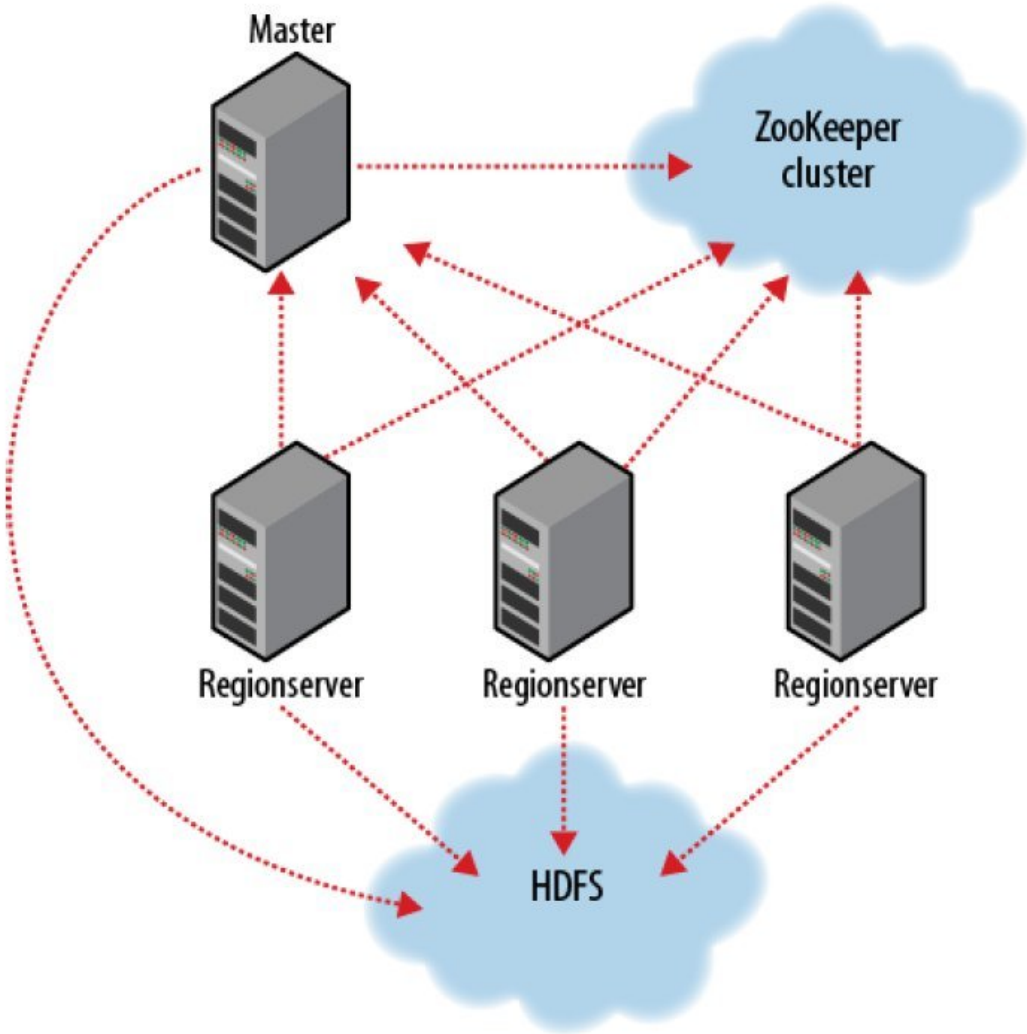
HBase Cell



[Lars George, "Hbase: The Definitive Guide", O'Reilly, 2011]

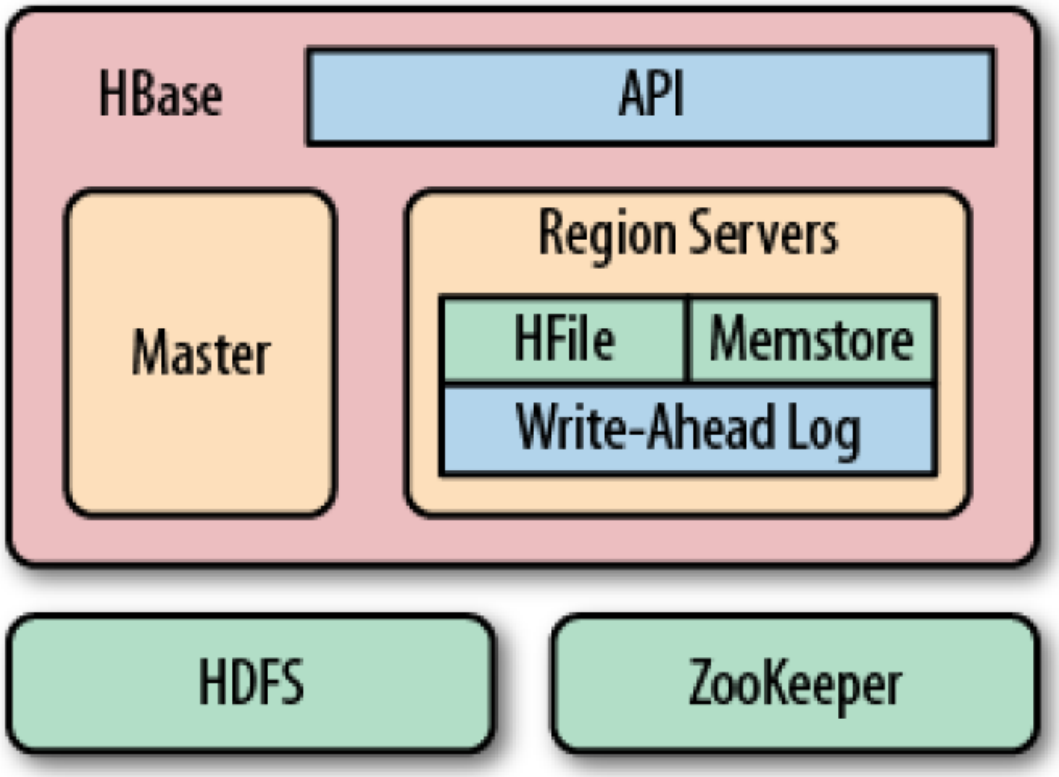
- (Table, RowKey, Family, Column, Timestamp) $\rightarrow$ Value

HBase Cluster



[Tom White, "Hadoop: The Definitive Guide", O'Reilly, 2012]

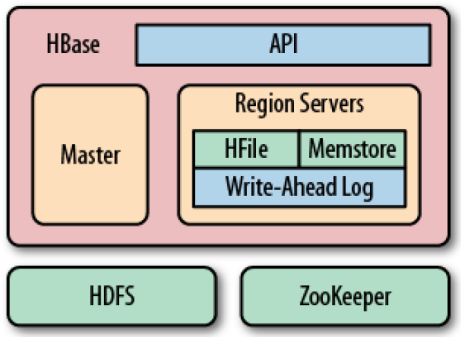
HBase Components



[Lars George, "Hbase: The Definitive Guide", O'Reilly, 2011]

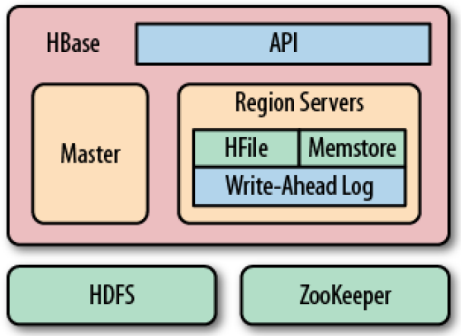
HBase Components - Region Server

- Responsible for all read and write requests for all regions they serve.
- Split regions that have exceeded the thresholds.
- Region servers are added or removed dynamically.



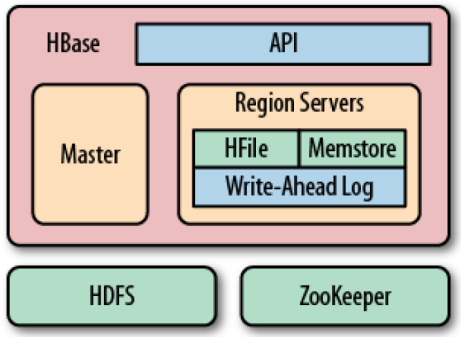
HBase Components - Master

- Responsible for managing regions and their locations.
  - Assigning regions to region servers (uses Zookeeper).
  - Handling load balancing of regions across region servers.
- Doesn't actually store or read data.
  - Clients communicate directly with region servers.
- Responsible for schema management and changes.
  - Adding/removing tables and column families.



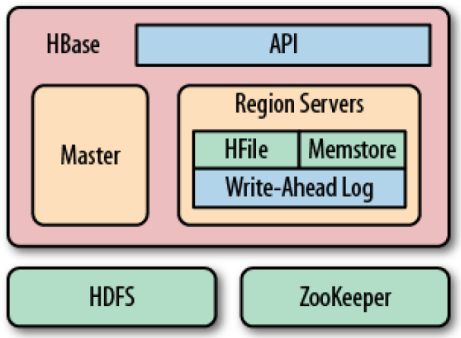
HBase Components - Zookeeper

- A coordinator service: not part of HBase
- Master uses Zookeeper for region assignment.
- Ensures that there is only one master running.
- Stores the bootstrap location for region discovery: a registry for region servers



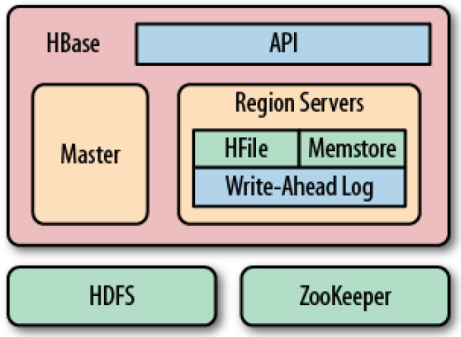
HBase Components - HFile

- The data is stored in HFiles.
- HFiles are immutable sequences of blocks and saved in HDFS.
- Block index is stored at the end of HFiles.
- Cannot remove key-values out of HFiles.
- Delete marker (tombstone marker) indicates the removed records.
  - Hides the marked data from reading clients.
- Updating key/value pairs: picking the latest timestamp.



HBase Components - WAL and memstore

- When data is added it is written to a log called Write Ahead Log (WAL) and is also stored in memory (memstore).
- When in-memory data exceeds maximum value it is flushed to an HFile.



Compaction

- Flushing memstores to disk causes more and more HFiles to be created.
- HBase uses a compaction mechanism to merges the files into larger ones.
- Minor compaction: rewriting smaller files into fewer but larger ones.
- Major compaction: rewriting all files within a column family for a region into a single new one.

Summary

- NoSQL data models:
  - key-value, column-oriented, document-oriented, graph-based
- Sharding and consistent hashing
- ACID vs. BASE
- CAP (Consistency vs. Availability)
- HBase: column-oriented NoSQL database
  - Master, Region server, HFile, memstore, AWL

HBase Installation and Shell

HBase Installation

- Three options
  - Local (Standalone) Mode
  - Pseudo-Distributed Mode
  - Fully-Distributed Mode

Installation - Local

- Uses local filesystem (not HDFS).
- Runs HBase and Zookeeper in the same JVM.

Installation - Pseudo-Distributed (1/3)

- Requires HDFS.
- Mimics Fully-Distributed but runs on just one host.
- Good for testing, debugging and prototyping, not for production.
- Configuration files:
  - `hbase-env.sh`
  - `hbase-site.xml`

Installation - Pseudo-Distributed (2/3)

- Specify environment variables in `hbase-env.sh`

```
export JAVA_HOME=/opt/jdk1.7.0_51
```

Installation - Pseudo-Distributed (3/3)

- Starts an HBase Master process, a ZooKeeper server, and a Region-Server process.
- Configure in `hbase-site.xml`

```
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>

<property>
  <name>hbase.rootdir</name>
  <value>hdfs://localhost:8020/hbase</value>
</property>
```

Start HBase and Test

- Start the HBase daemon.

```
start-hbase.sh
hbase shell
```

- Web-based management
  - Master host: `http://localhost:60010`
  - Region host: `http://localhost:60030`

HBase Shell

```
status

list

create 'Blog', {NAME=>'info'}, {NAME=>'content'}

# put 'table', 'row_id', 'family:column', 'value'
put 'Blog', 'Matt-001', 'info:title', 'Elephant'
put 'Blog', 'Matt-001', 'info:author', 'Matt'
put 'Blog', 'Matt-001', 'info:date', '2009.05.06'
put 'Blog', 'Matt-001', 'content:post', 'Do elephants like monkeys?'

# get 'table', 'row_id'
get 'Blog', 'Matt-001'
get 'Blog', 'Matt-001', {COLUMN=>['info:author','content:post']}

scan 'Blog'
```

Questions?