Trends in NoSQL Technologies

Database Systems, CS386D

Instructor: Don Batory

Ankit, Prateek and Dheeraj

Agenda

Fundamental Concepts

Why NoSQL?

What is NoSQL?

NoSQL Taxonomy

Case Studies

Project Summary

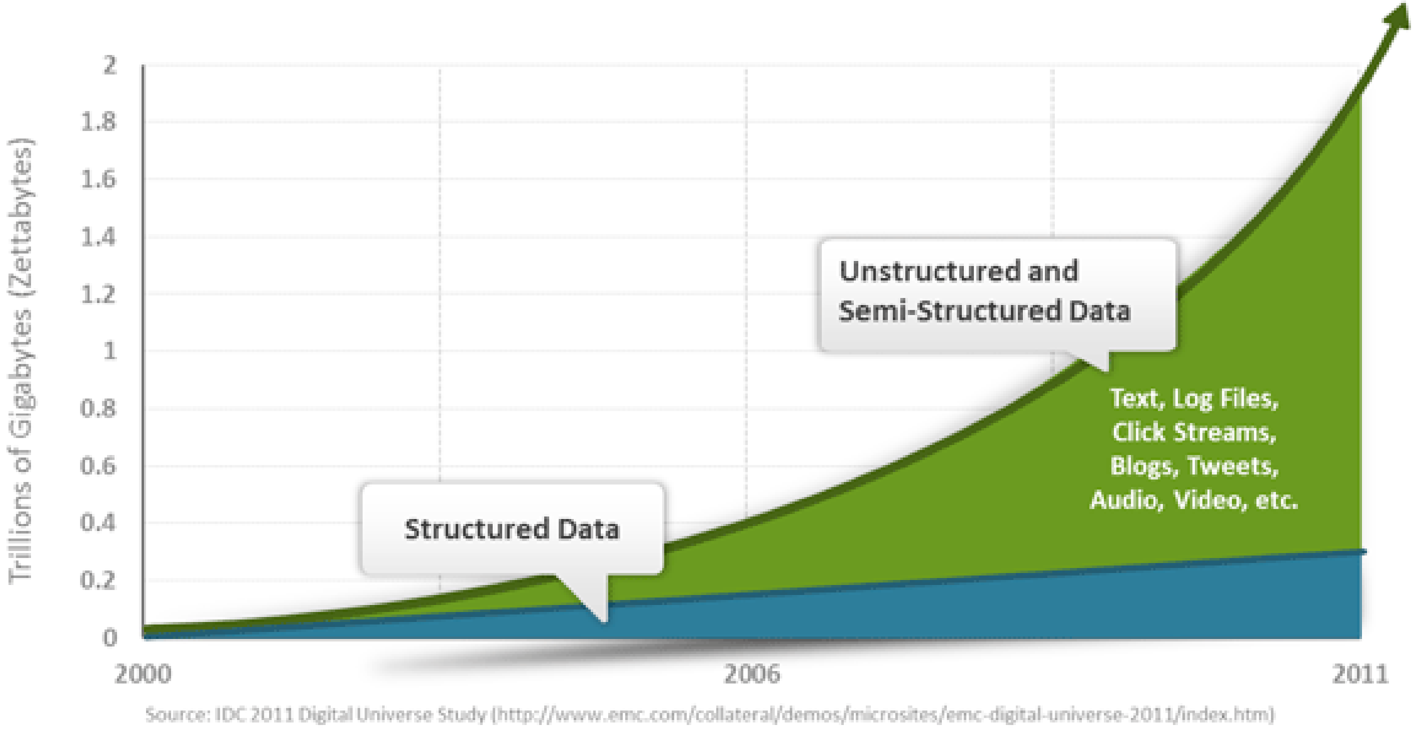
References

Why NoSQL?

New Trends

Growth in data

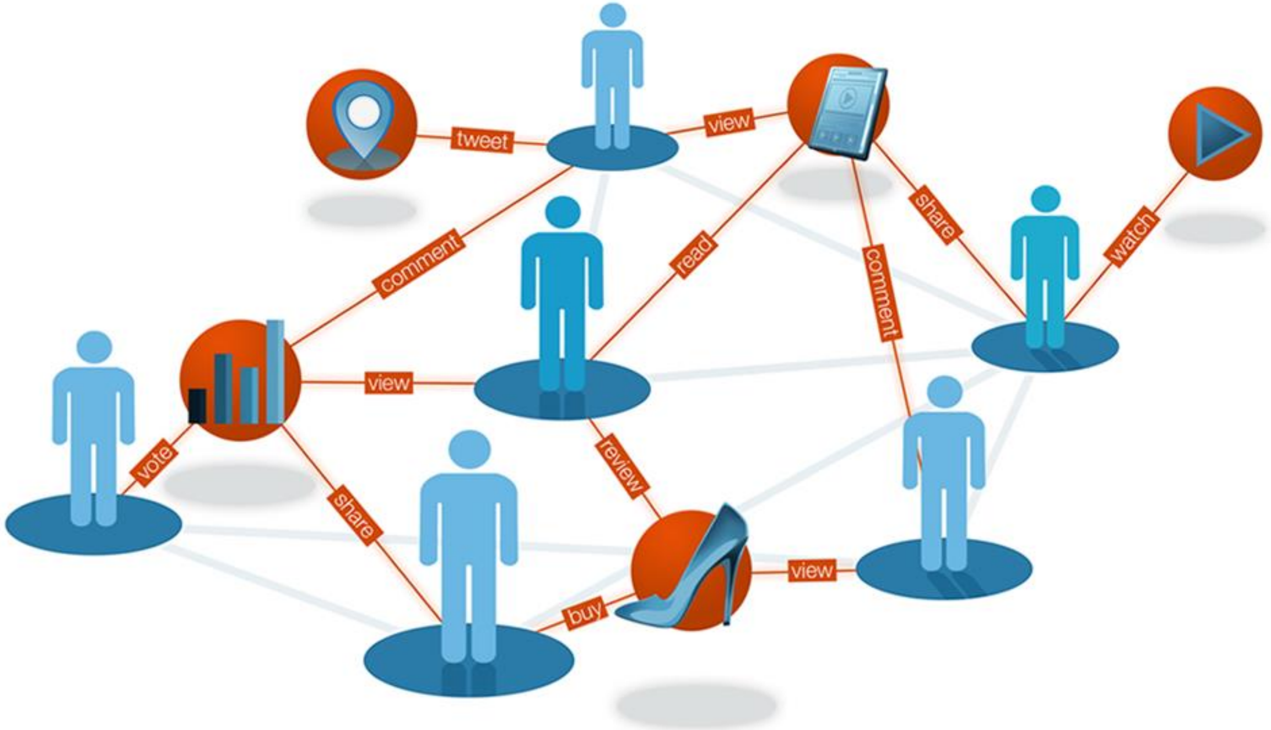
"Big Data" + "Unstructued data"



Source: http://www.couchbase.com

Connectedness

"Social networks"



Source: http://http://tjm.org/

Architecture

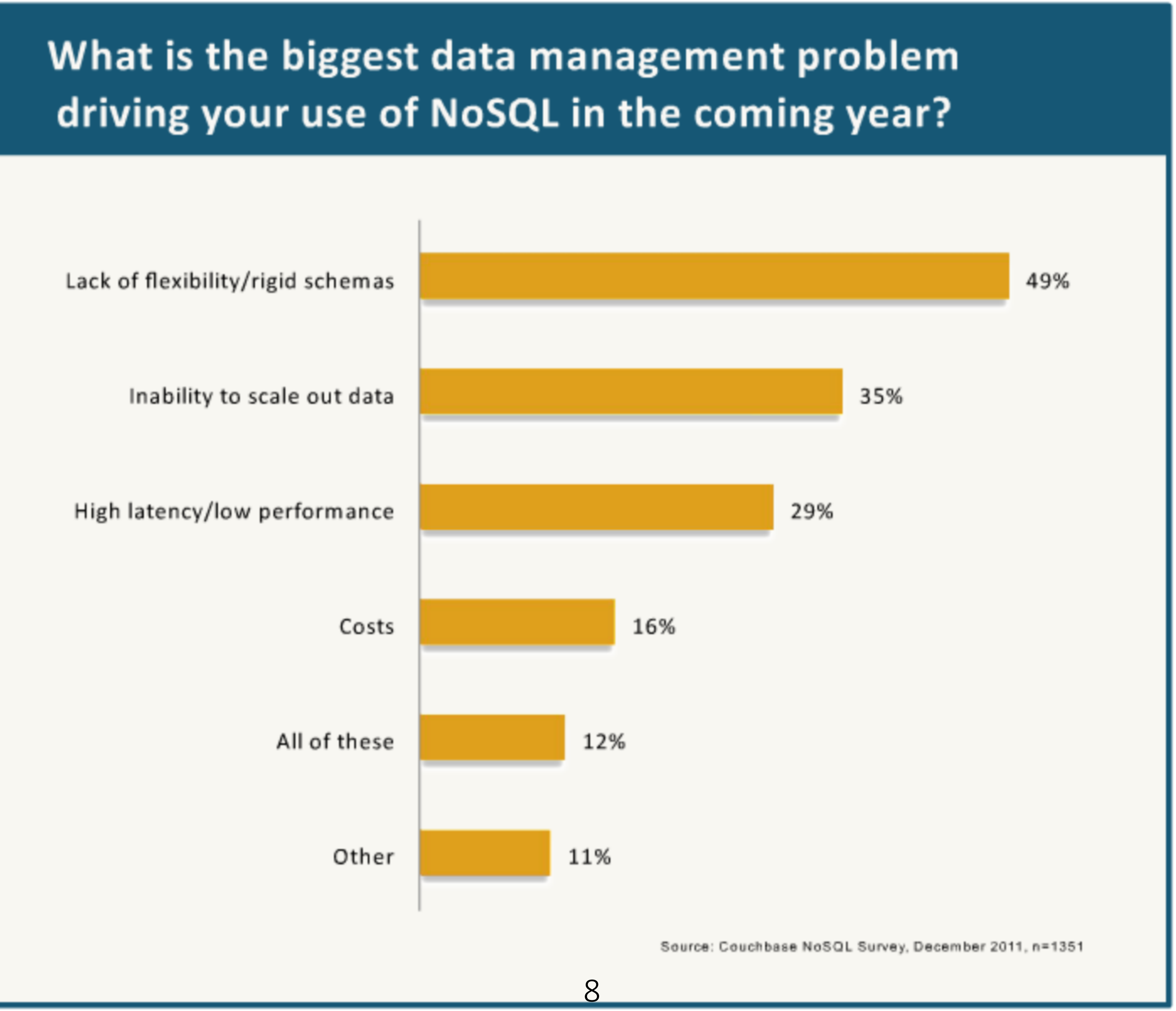
"Concurrency"

Source: http://www.couchbase.com

Architecture

"Concurrency"

Source: http://www.slideshare.net/thobe/nosql-for-dummies



Couchbase NoSQL Survey

1. Flexibility (49%)
2. Scalability (35%)
3. Performance (29%)

Extending the scope of RDBMs

Data Partitioning ("sharding")
+ Enables scalability
- Difficult process
- No-cross shard joins
- Schema management on every shard

Denormalizing
+ Increases speed
+ Provides flexibility to sharding
- Eliminates relational query benefits

Distributed Caching
+ Accelerated reads
+ Scale out : ability to serve larger number of requests
- Another tier to manage

RDMBS

Not a "One Shoe fits all" solution

Oracle has tried it

Need something different

Fundamental Concepts

ACID and CAP (A Quick Review)

ACID

Atomicity * Consistency * Isolation * Durability

Set of properties that guarantee that database transactions are processed reliably

Example:

Transfer of funds from one bank account to another (lower the FROM account and raise the TO account)

CAP

It is impossible in a for a distributed computer system to simultaneously provide all three of the following guarantees

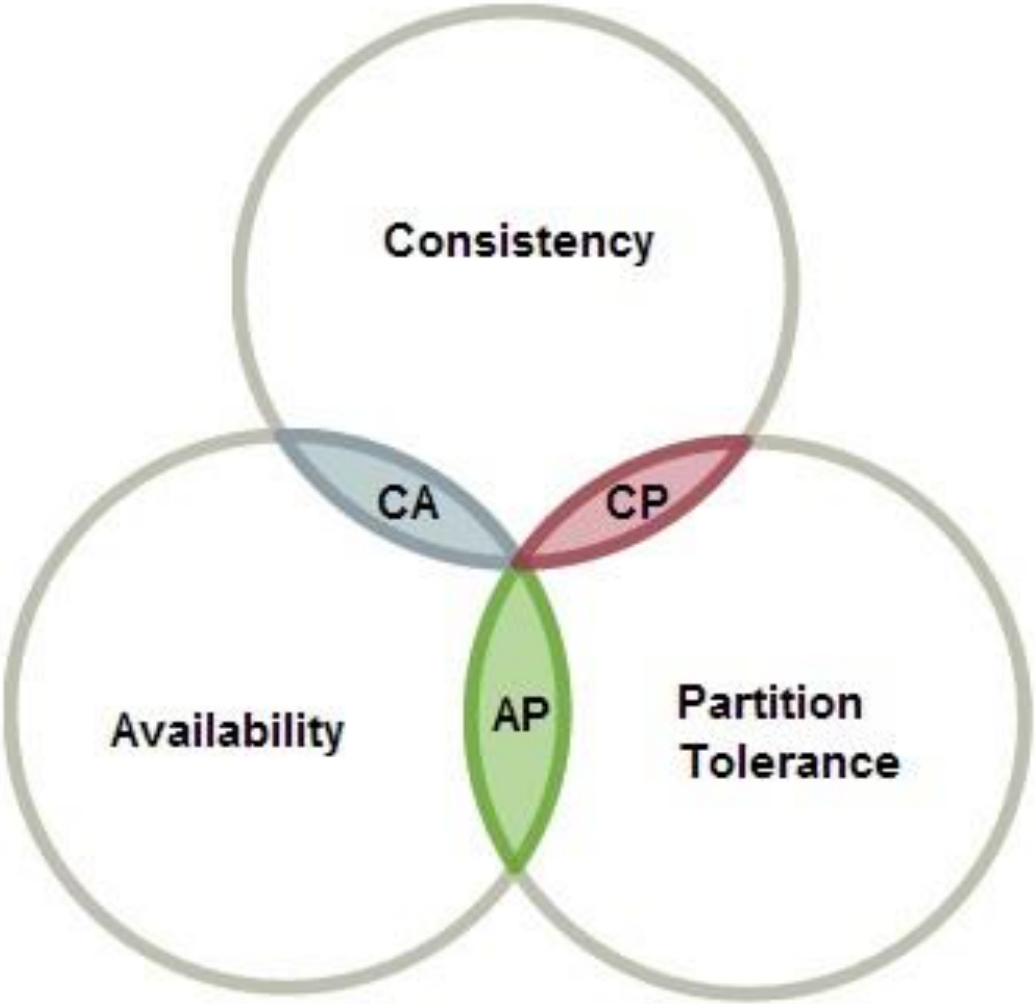
Cluster : A distributed network of nodes which acts as a gateway to the user

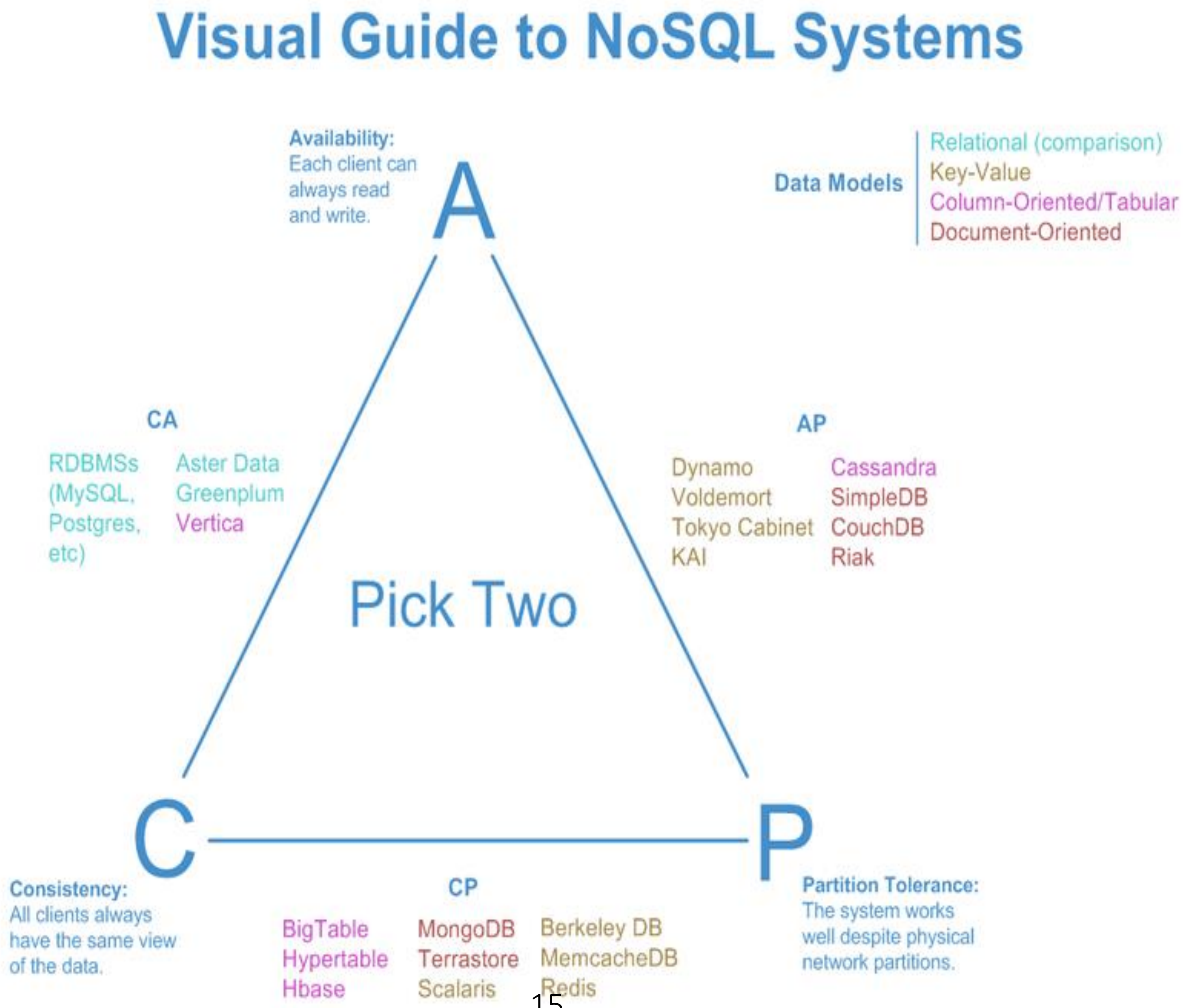
- **Consistency**
  - Data is consistent across all the nodes of the cluster.

- **Availability**
  - Ability to access cluster even if nodes in cluster go down.

- **Partition Tolerance**
  - Cluster continues to function even if there is a “partition” between two nodes.

CAP





Categorization

What is different?

What is NoSQL database technology

Design Features

Data Model

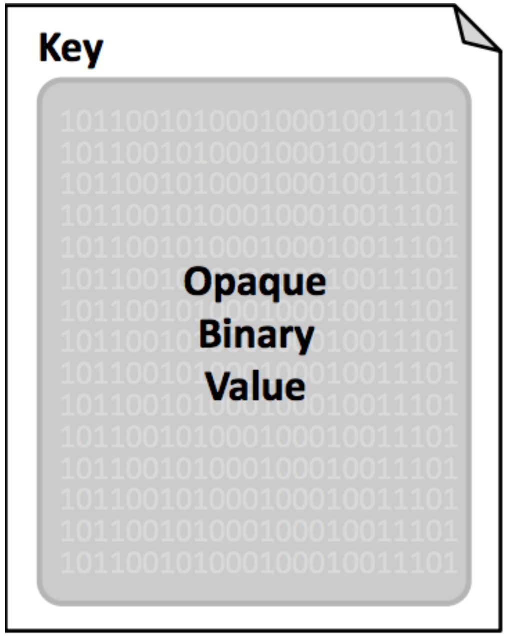
- No schema enforced by database - "Schemaless"

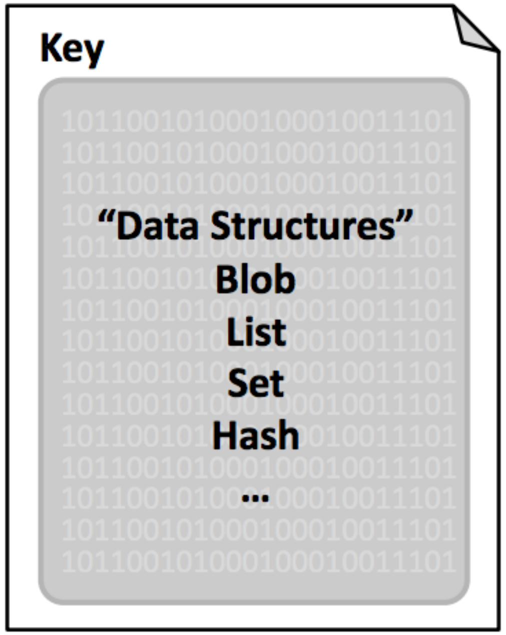
- Four major categories
  - Key/Value stores
  - Document Stores
  - Columnar stores
  - Graph Databases

Key Value Stores

- Redis
- BDB
- Memcached
- Membase
- Voldemort
- Dynamo

At the core all NoSQL databases are key/value systems, the difference is whether the database understands the value or not.





Key Value Stores - examples

ORACLE BERKELEY DB

- In memory / on disk
- Key/Value pair storage
- Transactional support
- Purpose: Lightweight DB

redis

- Persistent In-memory
- Key/Value pair storage
- Provides special data structures
- pub/sub capabilities.
- Purpose: Caching and beyond

MEMCACHED

- In memory / on disk
- Key/Value pair storage
- Purpose: Caching

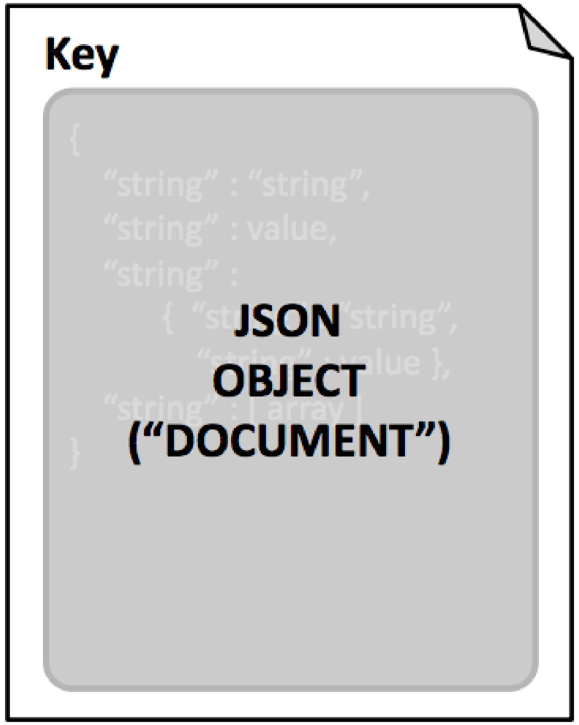
membase

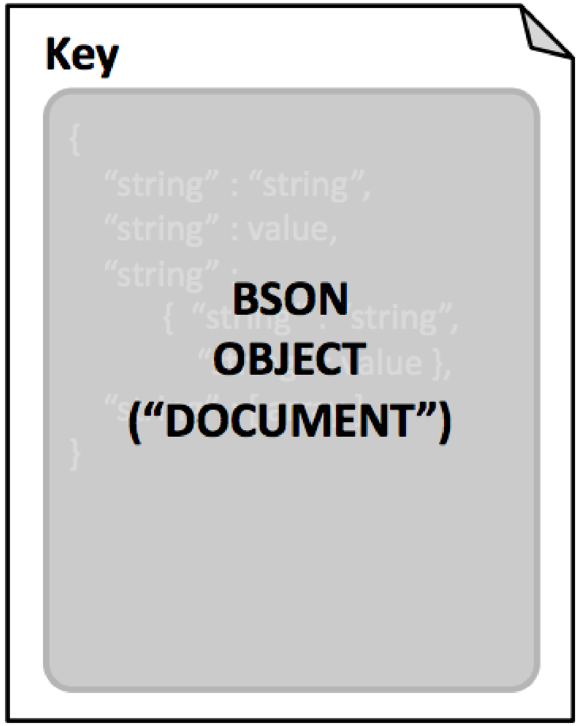
- Disk-based with built in memcache
- Cache refill on restart
- Highly Available (replication)
- Add/Remove live cluster
- Purpose: Caching

Document Stores

MongoDB, Couchbase

- DB understands values
- Data store in JSON/XML/BSON objects
- Secondary Indexes possible
- Schemaless
- Query on attributes inside values possible





Document Stores - examples

- memcached + couchDB
- Data stored as JSON objects
- Autosharding (replication)*
- Highly Available
- Create indexes, views.
- Query against indexes.
- Native support for map-reduce

- Data stored as BSON
- Very easy to get started
- Disk based with in-memory caching
- Auto-sharding*
- Supports Ad-hoc queries
- Native support for map-reduce.

* **Auto-sharding** - As system load changes, assignment of data to shards is rebalanced automatically

Column Oriented Stores

Cassandra

- DB understands values
- You don't need to model all the columns required by your application upfront.
- Technically It's a partitioned row store, where rows are organized into tables with a required primary key.

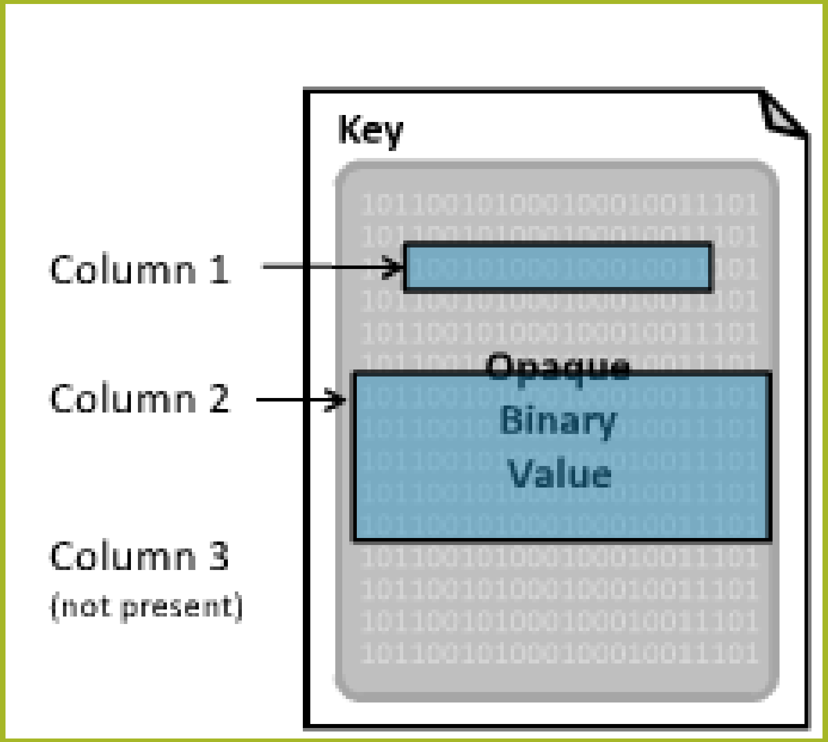
Normal column family:

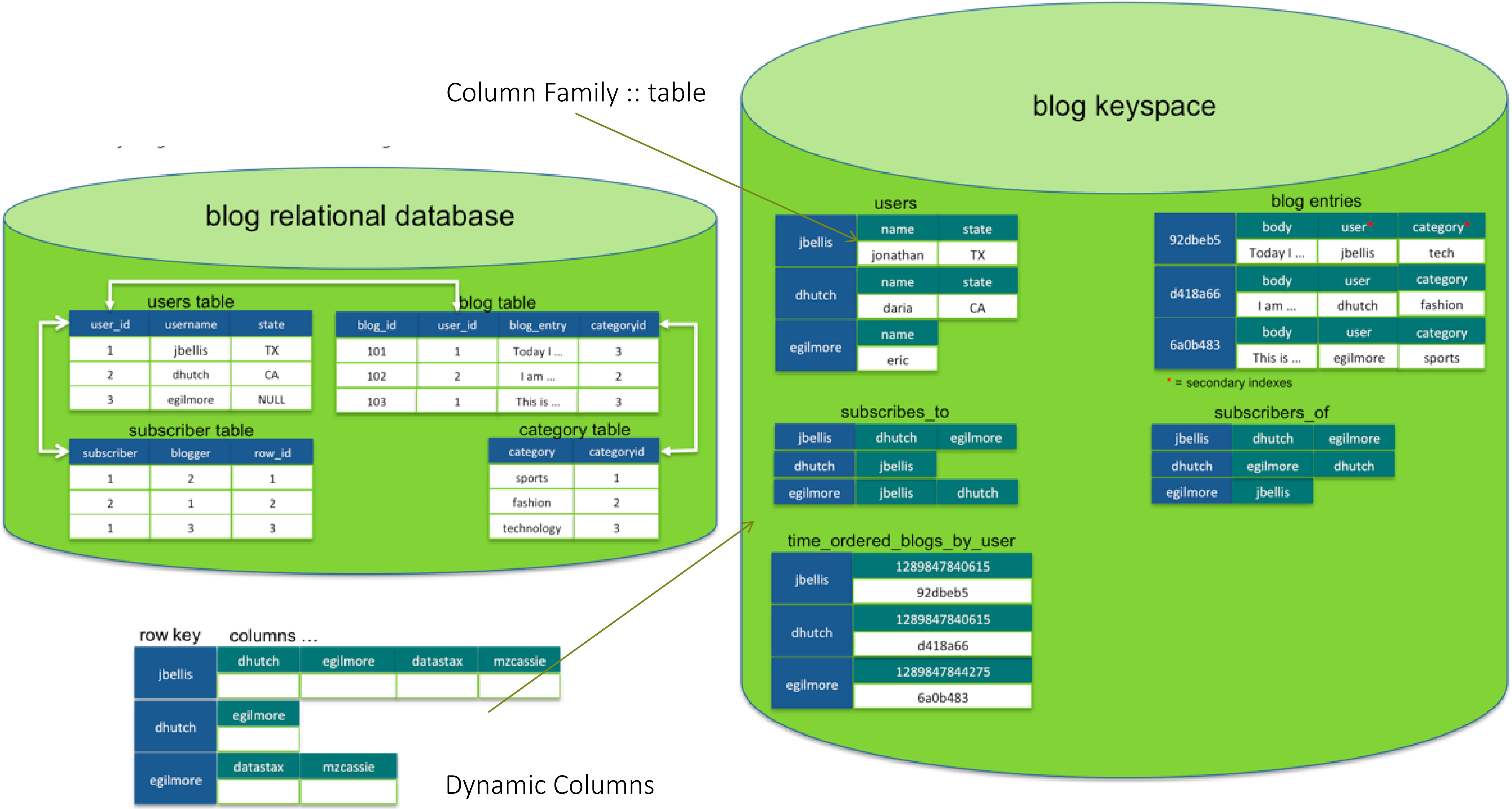
```
row
    col  col  col ...
    val  val  val ...
```

Super column family:

```
row
      supercol                      supercol                      ...
          (sub)col  (sub)col  ...       (sub)col  (sub)col  ...
           val        val     ...        val        val     ...
```

source: http://stackoverflow.com





http://www.datastax.com/docs/0.8/ddl/index

Column oriented store - examples

APACHE HBASE

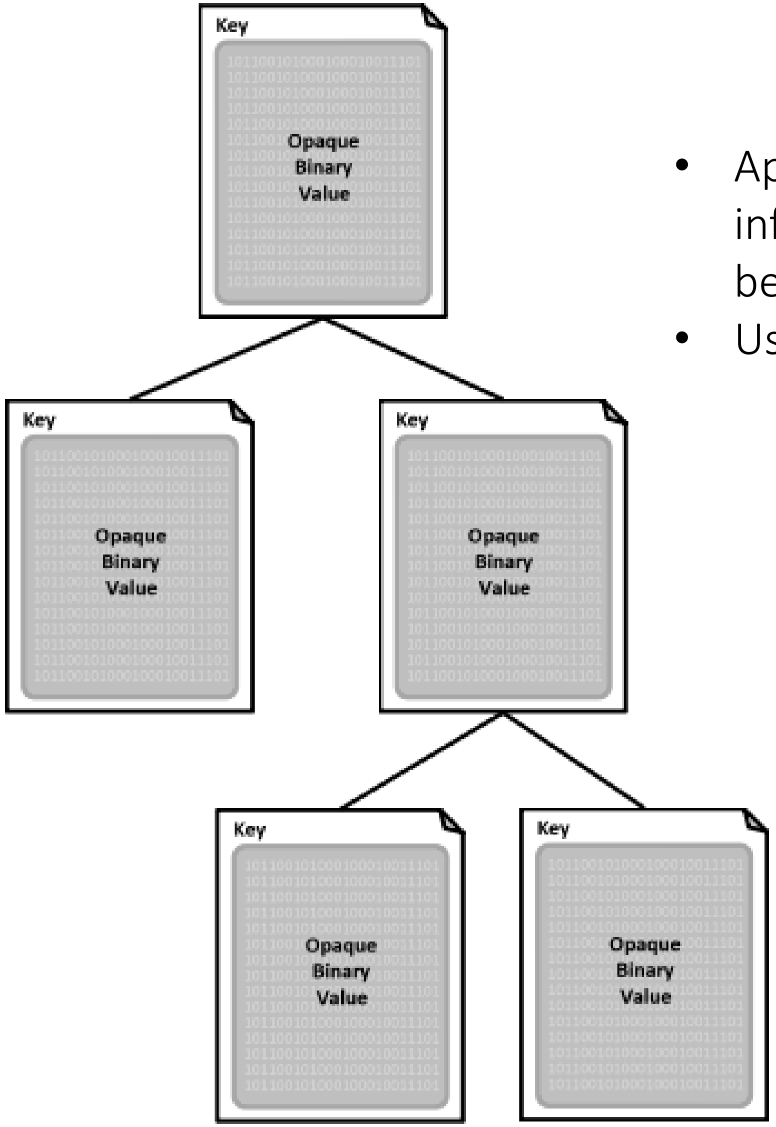
- Open source clone to Google's BigTable
- Runs only on top of HDFS
- CP based system

cassandra

- Modeled after Google's BigTable
- Clustered like Dynamo
- Good cross datacenter support
- Supports efficient queries on columns
- Eventually consistent
- AP based system

Graph Databases

Neo4J, GraphDB, Pregel



- Apply Graph Theory to the storage of information about the relationship between entries
- Used for recommendation engines.

source: http://stackoverflow.com

Graph DB - example

Neo4j

- Disk-based system
- External caching required
- Nodes, relationships and paths
- Properties on nodes
- Complex query on relations

Wait for more in the Graph DB presentation!

NoSQL catalog

	Key-Value	Data Structure	Document	Column	Graph
Cache (memory only)	memcached	redis			
Database (memory/disk)	membase		couchbase	cassandra	Neo4j
	ORACLE BERKELEY DB		mongoDB		

In-house solutions

Google	amazon.com	facebook	LinkedIn
Bigtable	Dynamo	Cassandra	Voldemort
November 2006	October 2007	August 2008	February 2009

- No schema required before inserting data
- No schema change required to change data format
- Auto-sharding without application participation
- Distributed queries
- Integrated main memory caching
- Data Synchronization (multi-datacenter)

Case Studies

Amazon's Dynamo and Google's Bigtable

Google's BigTable

BigTable

Designed to scale.

And the scale we are talking is of Petabytes!

A distributed store for managing *structured* data.

Three dimensional Table structure.

Uninterpretated bytes storage.

CP - Choses Consistency over Availability in the case of network partitioning (CAP theorem)

Basically, it is just a sparse, distributed, persistent sorted map store.

Features

- sparse
- distributed
- scalable
- persistent
- sorted
- consistent
- map store

Sparse

Most of the columns are empty

Persistent

Data gets stored permanently in the disk

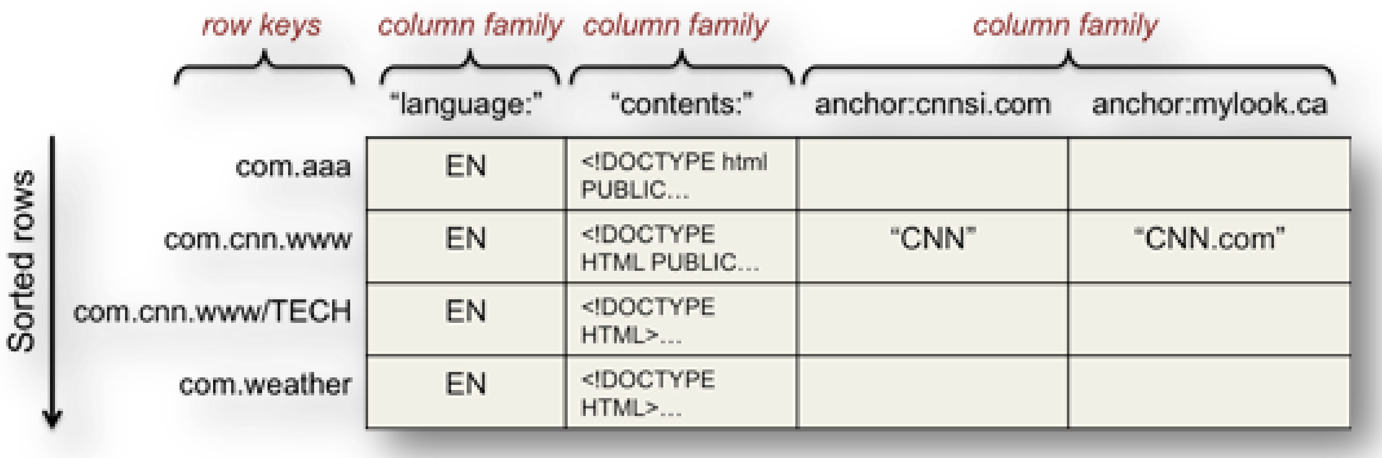
Sorted

Data kept in heirarchical fashion

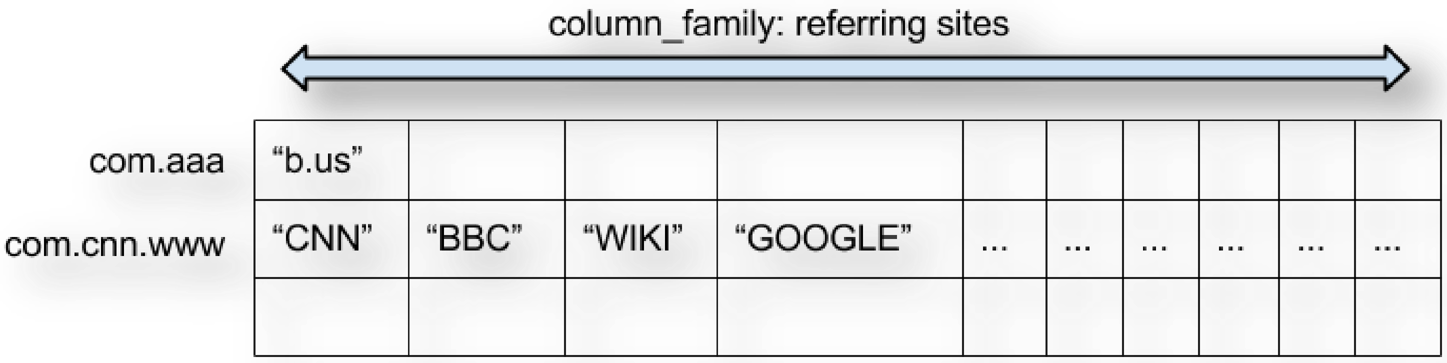
Spatial Locality

Consistent

Tablet :
BigTable's basic unit of storage



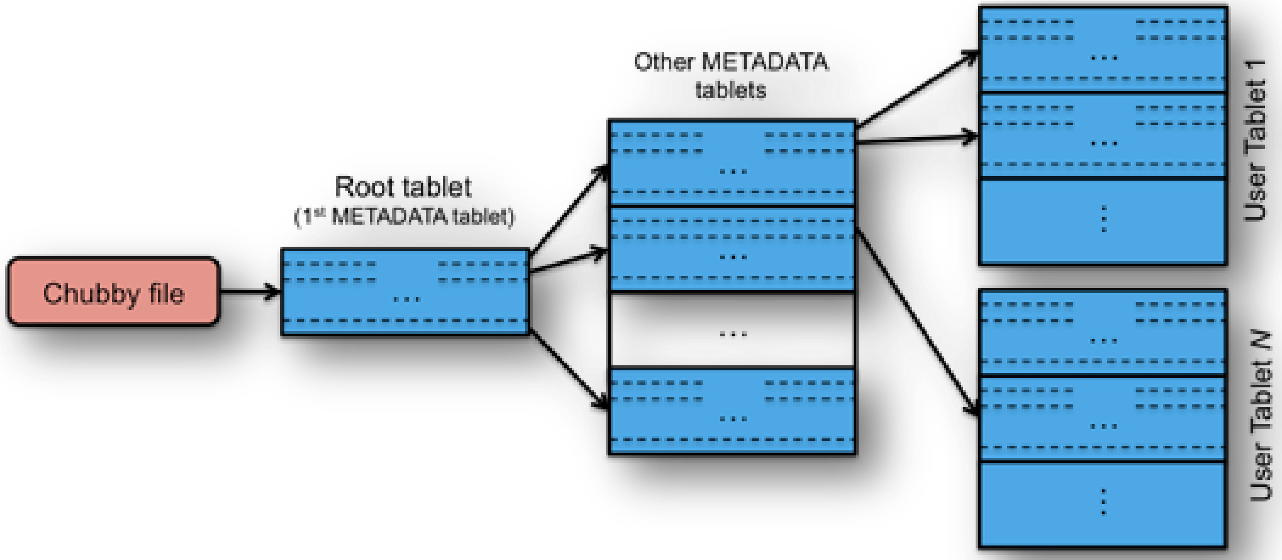
BigTable's basic data storage structure



Sparsity demonstrated in the table

Tablet dimensions

1. Rows
2. Column Families
3. Timestamps



BigTable indexing hierarchy

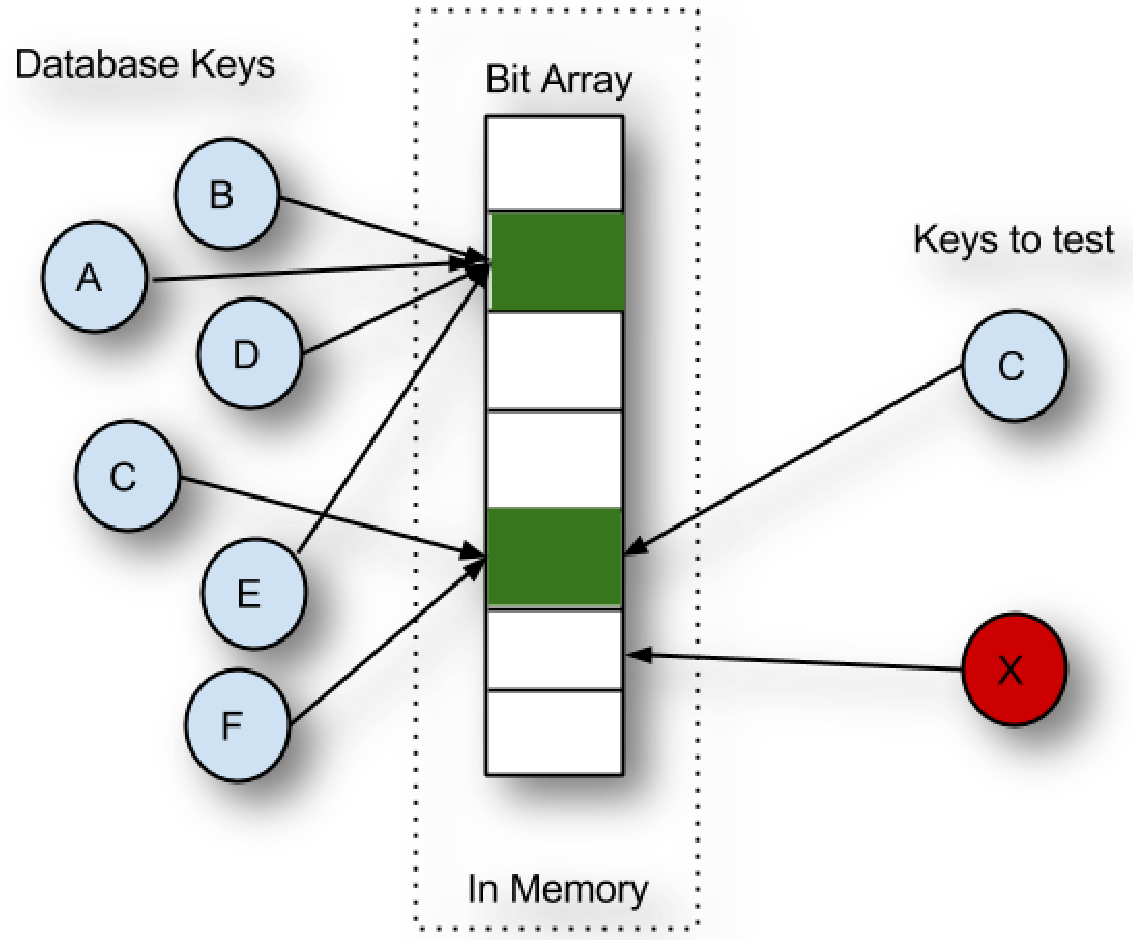
Storage Hierarchy

Tablet

Metadata tablet

Root tablet

Chubby file



Bloom Filter : Drastically reduces the number of disk seeks required for read!

Optimizations

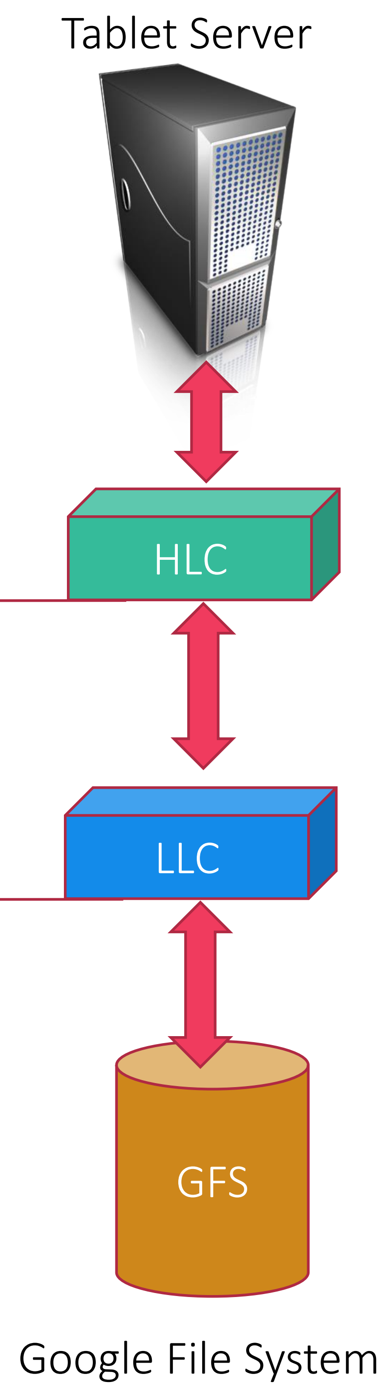
- **Bloom Filters**
- Caching

Higher Level Cache

- For scenarios where same data is read repeatedly

Lower Level Cache

- For spatial locality



Optimizations

- Bloom Filters
- **Caching**

Amazon DynamoDB

Amazon DynamoDB

Motivation:

“ Customers should be able to view and add items to their shopping cart even if network routes are broken or data centers are being destroyed by tornadoes.”

AP: It chooses availability over consistency in the case of network partitioning

Features

Highly Available key-value

High performance (low latency)

Highly scalable (hundreds of nodes)

"Always on" available (esp. for writes)

Partition/Fault-tolerant

Eventually consistent

Key Techniques

Consistent Hashing

For data partitioning, replicating and load balancing

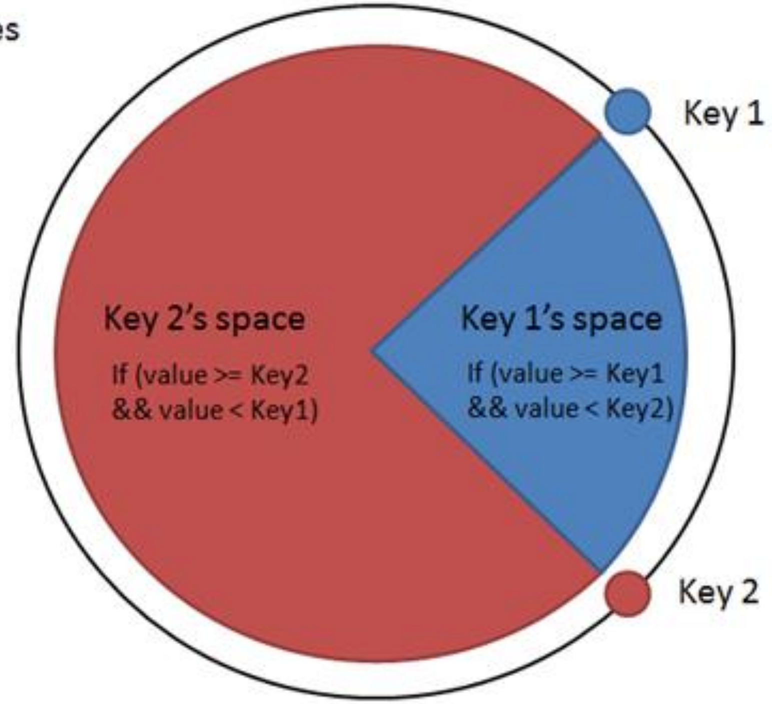
Sloppy Quorums

Boosts availability in present of failures

A Simple Example

Imagine that our consistent hash
is mapped to a continuum of values

All values are mapped to the continuum using some hash algorithm like MD5. This results in unpredictable assignments which can cause very imbalanced distribution of "key space".



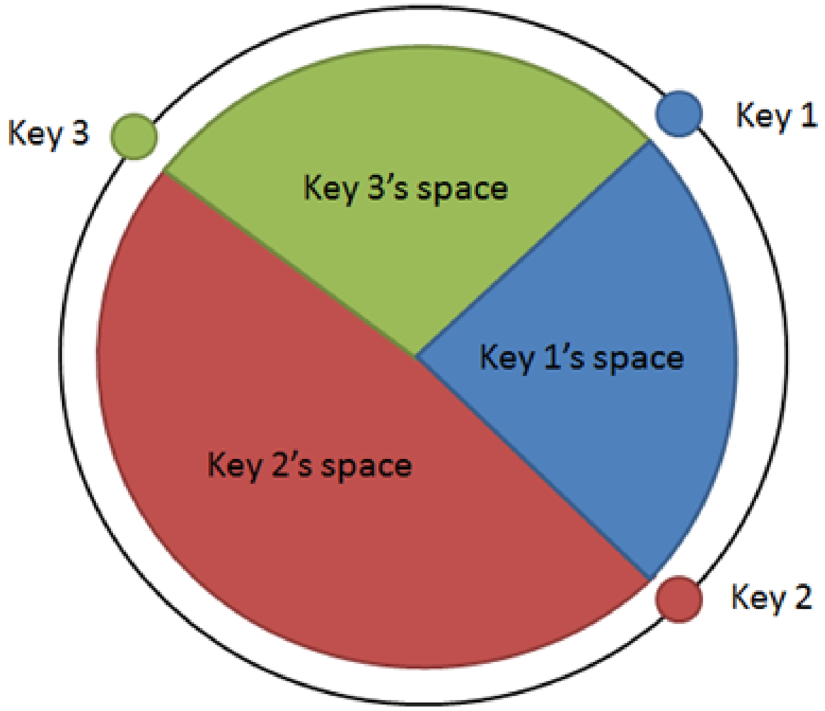
source: http://sharplearningcurve.com/blog/2010/09/27/consistent-hashing/

Consistent Hashing

Sharding

Adding a Node

!Important!
Adding a node does not cause the entire key-space to rebalance. This is very important to the implementation: adding nodes should not changes all the answers, it should only "claim" key space from a single node.



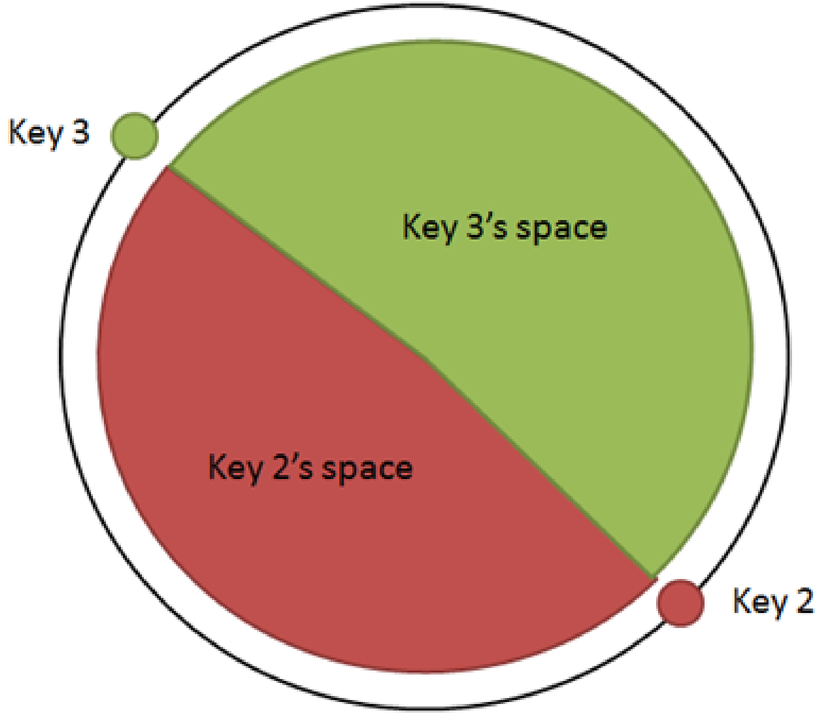
source: http://sharplearningcurve.com/blog/2010/09/27/consistent-hashing/

Consistent Hashing

Dynamically add nodes

Removing a Node

!Important!
Removing a Node should cause the hash to rebalance such that only the now vacant key space is reclaimed. As with adding, removing a node should ***not*** *remap the entire key space.*



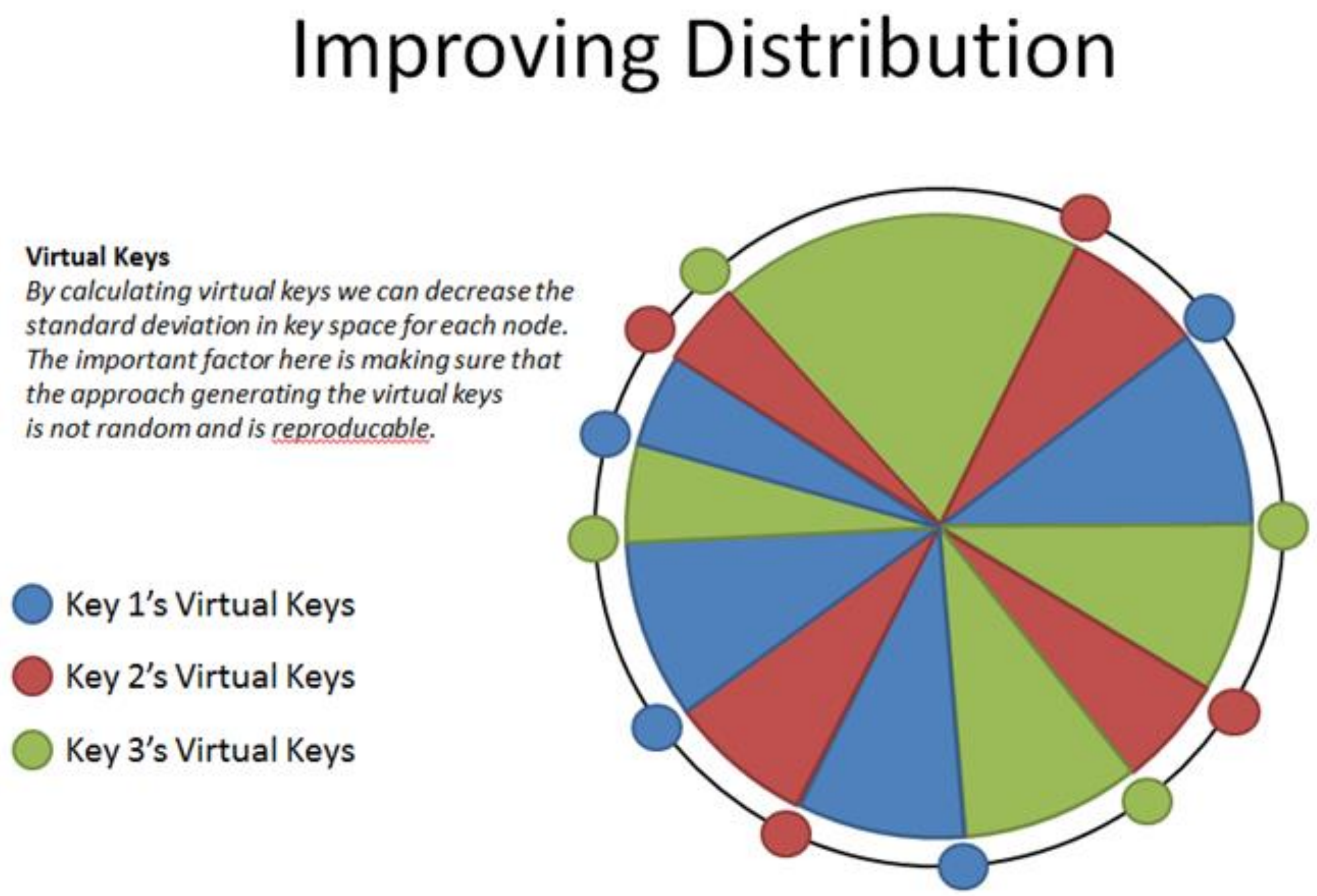
source: http://sharplearningcurve.com/blog/2010/09/27/consistent-hashing/

Consistent Hashing

Dynamically remove nodes

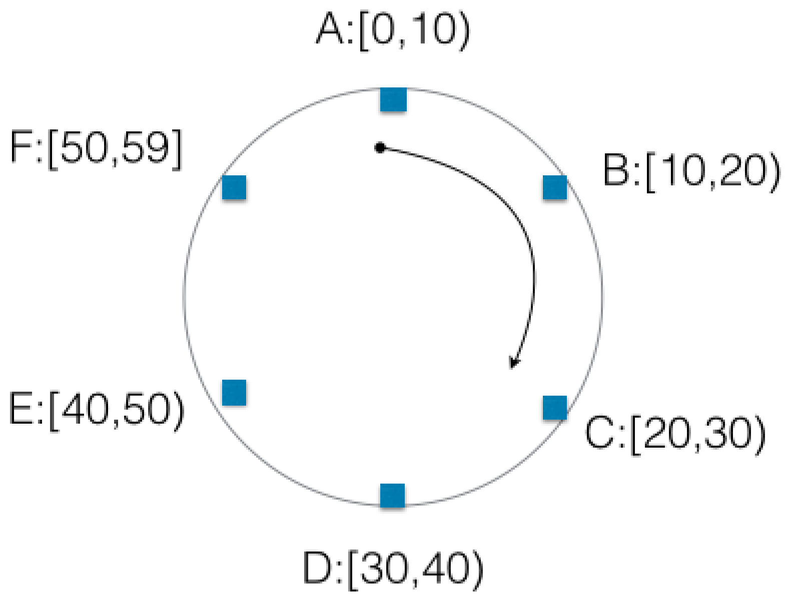
Consistent Hashing

load balancing



source: http://sharplearningcurve.com/blog/2010/09/27/consistent-hashing/

Replication



N = 3

Hash	Node	Replicas
3	A	B,C
12	B	C,D
19	B	C,D
20	C	D,E
37	D	E,F
40	E	F,A
54	F	A,B

Sloppy Quorums

Availability in presence of failures

- R number of nodes that need to participate in read
- W number of nodes that need to participate in write
- **R + W > N (a quorum system)**

Dynamo:

W = 1 (Always available for write)

Yields R=N(reads pay penality)

Typical: R=2, W=2, N=4

Dynamo Summary

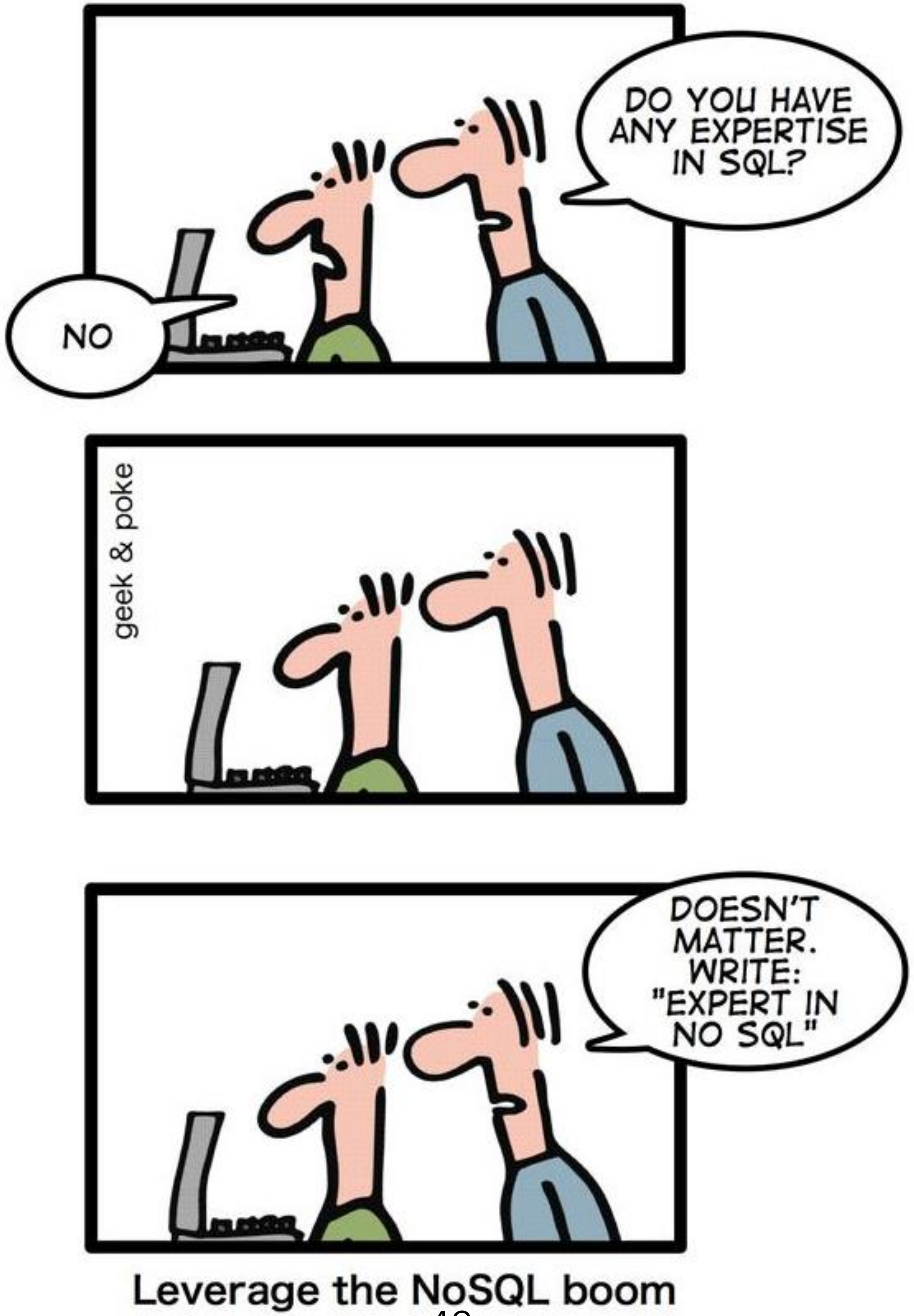
- An eventually consistent highly available key/value store
- AP in CAP space
- Focuses on low latency, SLAs
- Very low latency writes, reconciliation in reads

Key techniques used in many other distributed systems

- Consistent hashing, (sloppy) quorum-based replication, vector clocks, gossip-based membership, merkel tree synchronization

Project Summary

To be SQL or Not to be SQL



Bright future of NoSQL

Companies using NoSQL

- Google

- Facebook

- Amazon

- Twitter

- Linkedin

... many many more.

Conclusion

Even **NoSQL** - Not a "One Size fits all" kinda shoe.

Shoe horning your database is just bad, bad, bad!

Use when

- Data schema keeps on varying often
- Scalability really becomes an issue

Not to use when

- The data is inherently relational
- Lots of complex queries to write
- You need good helping resources
  - eg. debugger, performance tools

References - 1

Dynamo: amazon's highly available key-value store. In Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles (SOSP '07). ACM, New York, NY, USA, 205-220

Bigtable: A Distributed Storage System for Structured Data. ACM Trans. Comput. Syst. 26, 2, Article 4 (June 2008). Fay Chang et. al

http://docs.mongodb.org/manual/core/sharding-introduction

http://mongodb.com/learn/nosql"http://www.mongodb.com/learn/nosql

http://www.cs.rutgers.edu/~pxk/417/notes/content/bigtable.html

http://en.wikipedia.org/wiki/ACID

References - 2

http://www.slideshare.net/mongodb/mongodb-autosharding-at-mongo-seattle

http://www.slideshare.net/danglbl/schemaless-databases"http://www.slideshare.net/danglbl/schemaless-databases

http://infoq.com/presentations/NoSQL-Survey-Comparison"www.infoq.com/presentations/NoSQL-Survey-Comparison

http://info.mongodb.com/rs/mongodb/images/10gen_Top_5_NoSQL_Considerations.pdf

http://highscalability.com/blog/2010/12/6/what-the-heck-are-you-actually-using-nosql-for.html

http://technosophos.com/2014/04/11/nosql-no-more.html

Questions

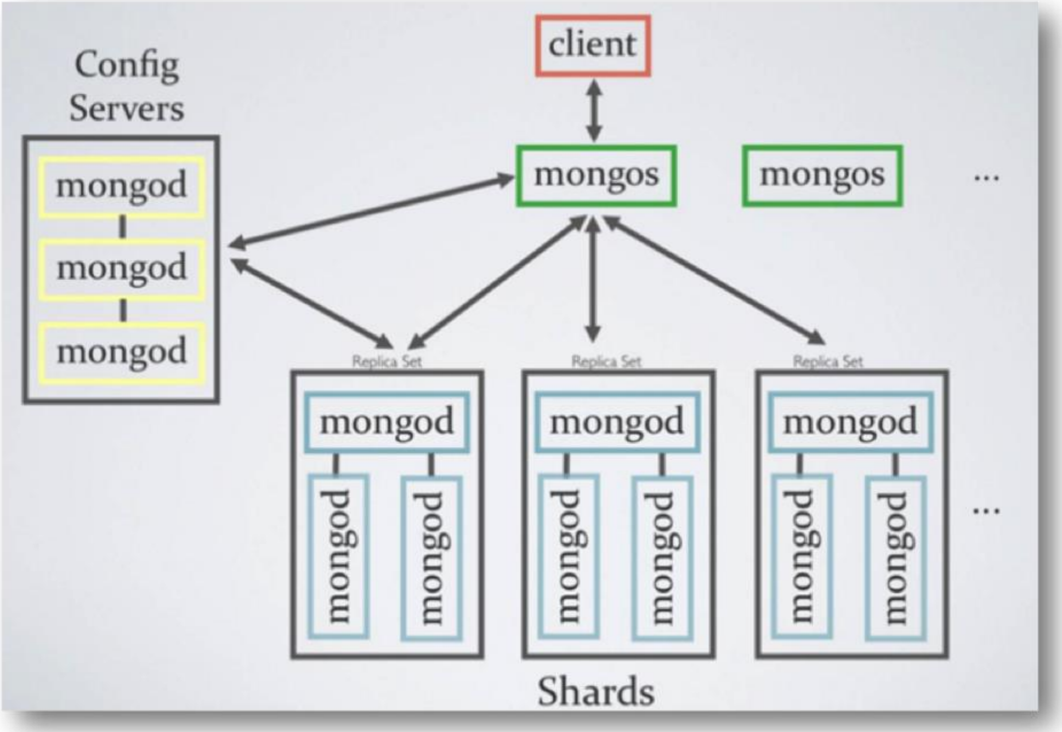
Backup Slides

Basic Concepts

Any storage model other than tabular relations.

- Trees
- Graphs
- Key-value
- XML
- etc.

Auto-sharding



TODO:
http://www.scalebase.com/extreme-scalability-with-mongodb-and-mysql-part-1-auto-sharding/

Impedence Mismatch

You break structured data into pieces and spread it across different tables.

leads to object relational mapping

lots of traffic => buy bigger boxes. Lot of small boxes. SQL was designed to run on single box.

Key Techniques

Consistent Hashing
- For data partitioning, replicating and load balancing

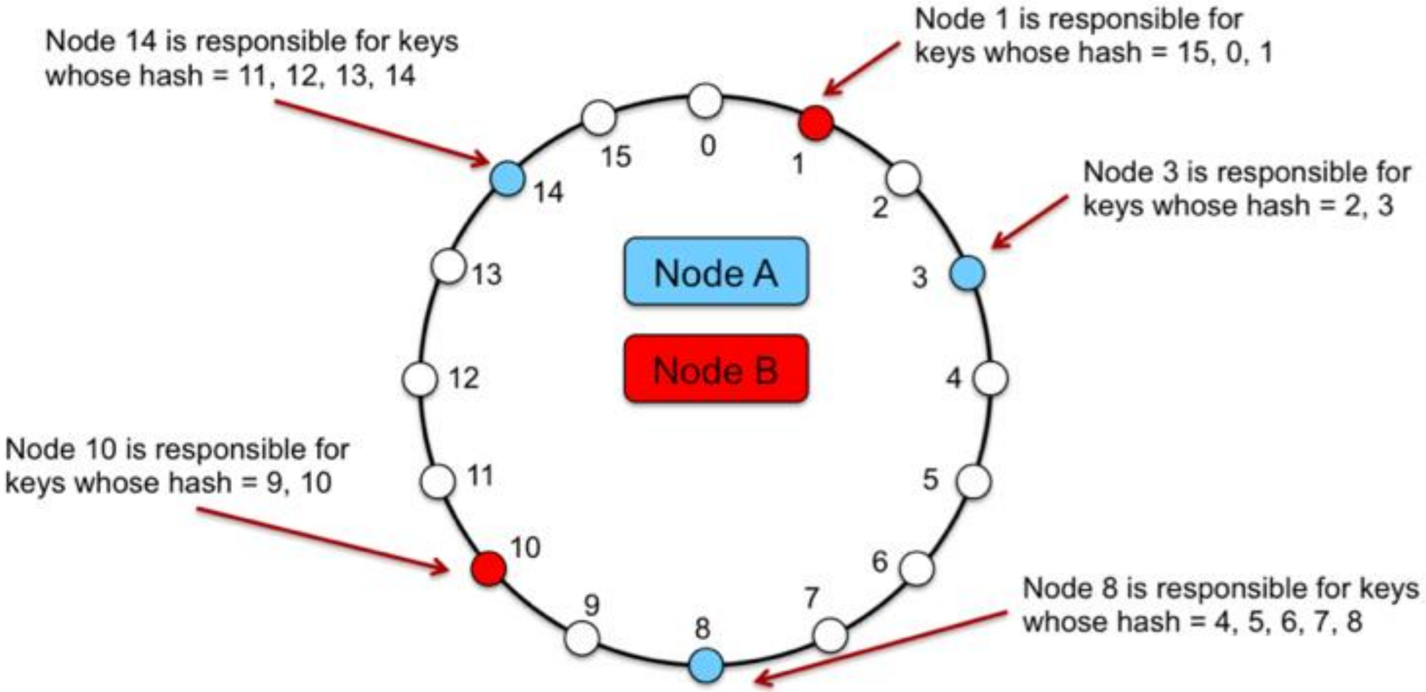
Sloppy Quorums
- Boosts availability in present of failures

Vector Clocks
- For tracking casual dependencies among different versions of the same key (data)

Gossip-based group membership protocol
- For maintaining information about live nodes

Anti-entropy protocol using hash/merkle trees
- Background synchronization of divergent replicas

Consistent Hashing



Availability

Replication and partitioning

Vector Clocks

Tracking causal dependencies

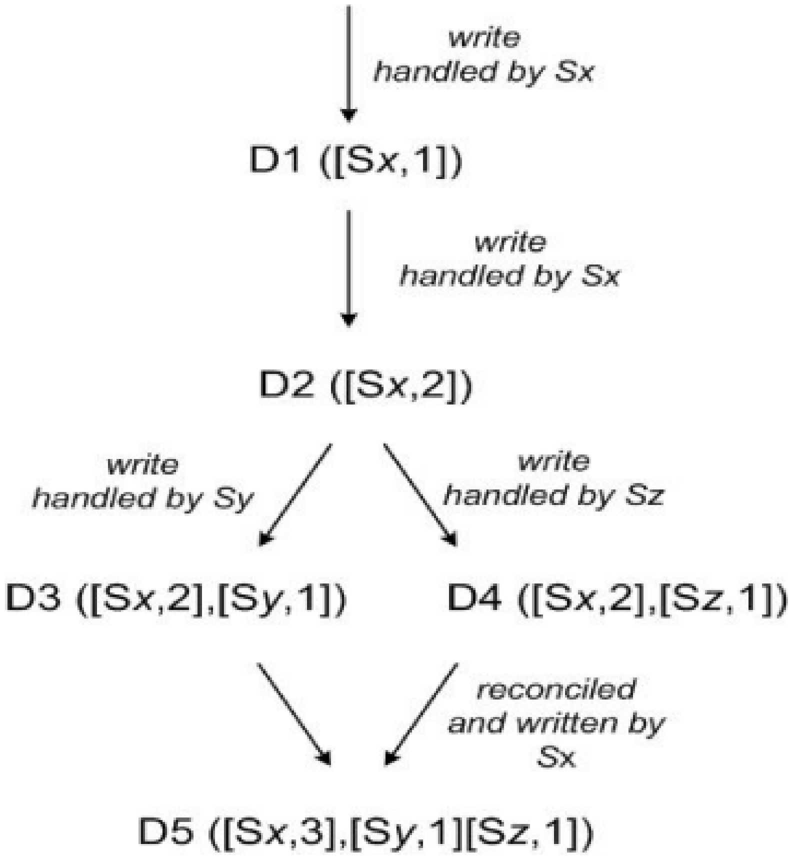


Figure 3: Version evolution of an object over time.

Merkel Trees

Each node keeps a merkel tree for each of its key ranges

Compare the root of the tree with replicas

- if equal => replicas in synch
- Traverse the tree and synch those keys that differ

Membership:

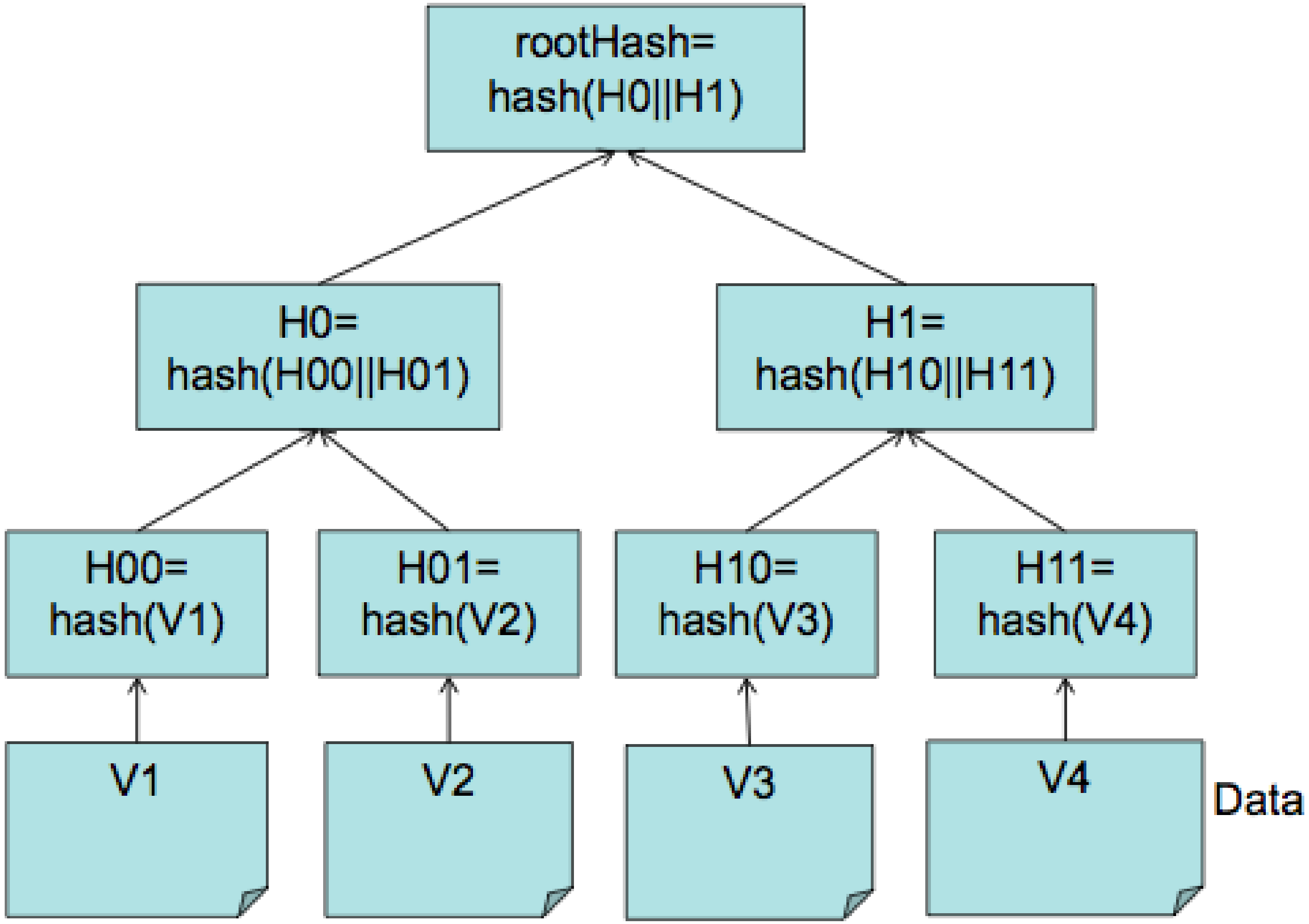
Node contacts a random node every 1s.

Gossip used for exchanging and partitioning/placement metadata

Gossip and Anti-entropy

Synchronization and book-keeping of live nodes

Merkel Trees



Atomicity requires that each transaction is "all or nothing"

Success	Failure
A: a + x	A: a + x
B: a - x	B: a - x

Atomicity

ACID

Consistency

ACID

The consistency property ensures that any transaction will bring the database from one valid state to another valid state.

Success

A: a + x

↓

B: b - x

Failure

A + B = a+b

A + B = a+b-10

Isolation

ACID

The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e. one after the other.

Success

T1: a - x

T1: b + x

T2 : b - x

T2 : a + x

Failure

T1: a - x

T2 : b - x

T2 : a + x

failure

T1: b + x

Durability

ACI**D**

Durability means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors

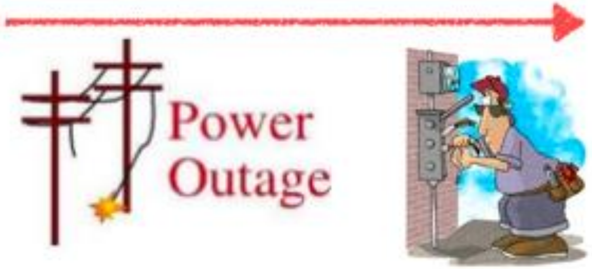
Success

T1: a - x

T1: b + x

T2 : b - x

T2 : a + x



Failure

the changes are lost

Persistent

Data gets stored permanently in the disk

Sorted

Data kept in heirarchical fashion

Spatial Locality

Map Store

Just a collection of (key, value) pairs

Features

- sparse
- distributed
- scalable
- **persistent**
- **sorted**
- **map store**

BASE

An alternative to ACID

- **B**asically **A**vailable
  - Support partial failures without total system failure.
- **S**oft state
  - optimistic and accepts that consistency will be in state of flux.
- **E**ventual Consistency
  - Given a sufficiently long period of time over which no changes are sent, all updates can be expected to propagate eventually.