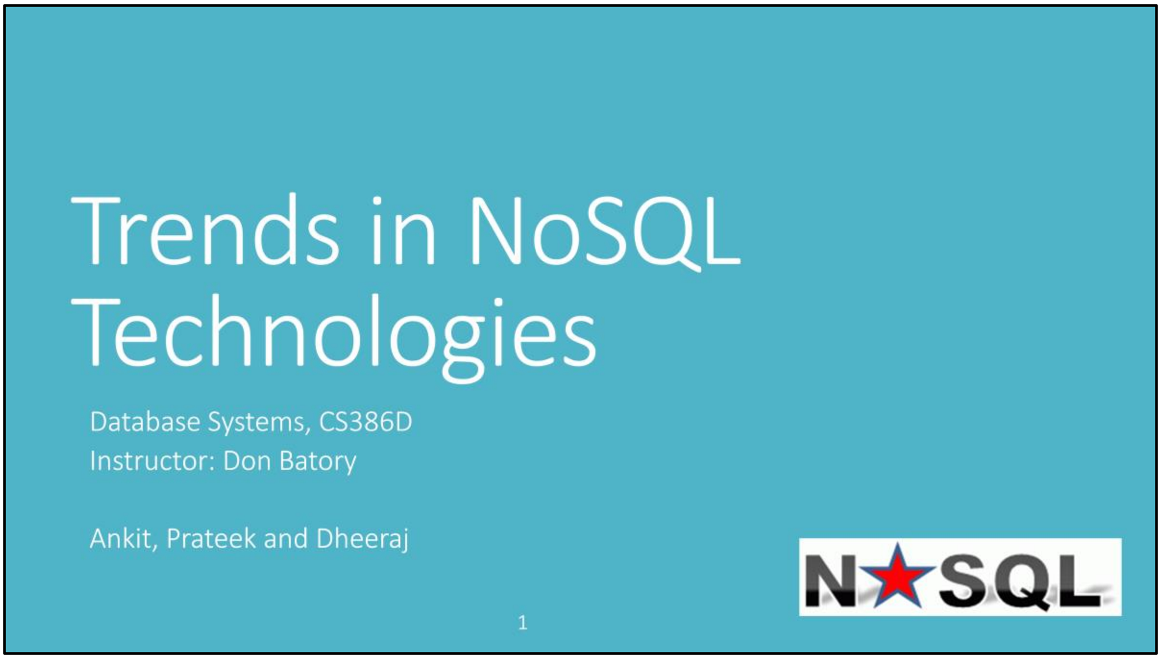


Trends in NoSQL Technologies

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

Goal of the presentation is to give an introduction of NoSQL databases, why they are there.

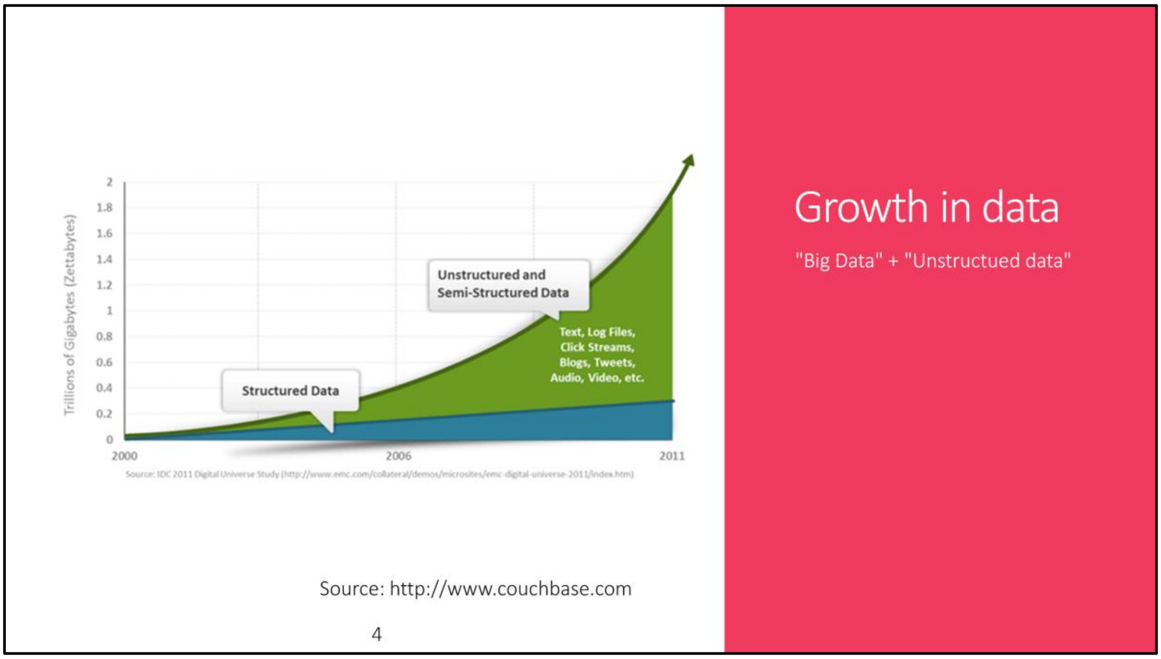
We want to present "Why?" first to explain the need of something like "NoSQL" and then in "What?" we go in detail.

In addition there are lots and lots of NoSQL databases available, we have chosen some widely used databases in the industry.
We think it's important that one should be aware of these databases and have the basic understanding of why they exist, and how they are different.

Why NoSQL?

New Trends

Justify their usage. Let's look at new trends in recent years.



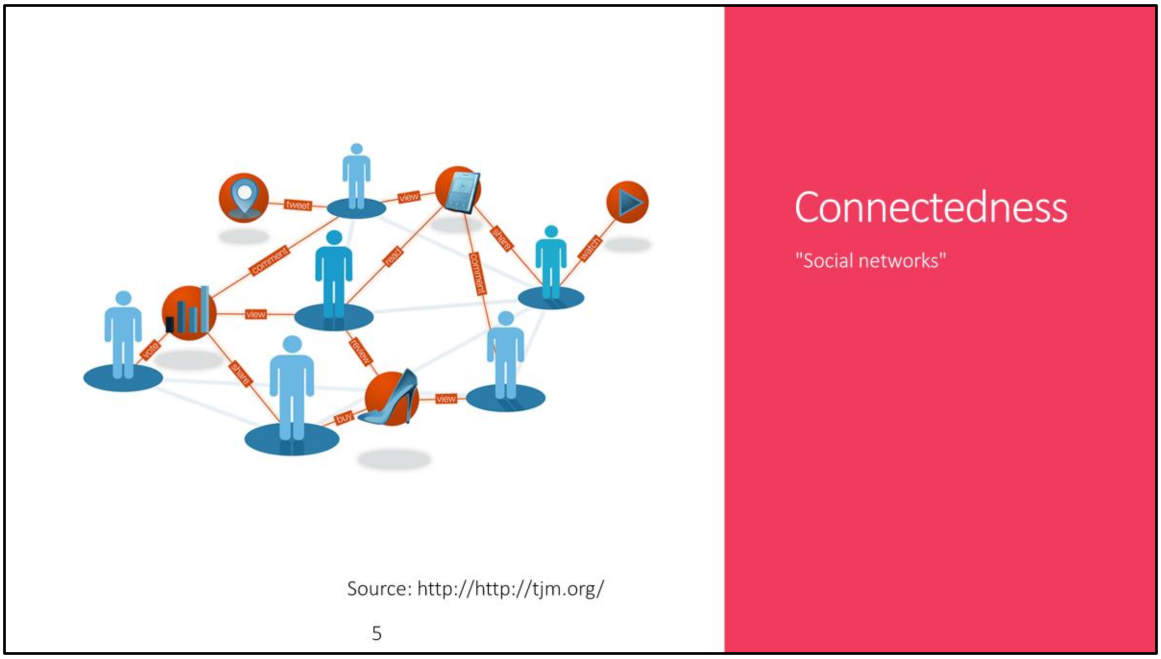
Source: IDC 2011 Digital Universe Study (http://www.emc.com/collateral/demos/microsites/emc-digital-universe-2011/index.htm)

Source: http://www.couchbase.com

Growth in data

"Big Data" + "Unstructued data"

1. Each year more and more data is created. Over two years we create more digital data than all the data created in history before that!
2. The rigidly defined, schema-based approach used by relational databases makes it impossible to quickly incorporate new types of data.
3. RDBMs are really good at transactions. perfected over the years. but huge amount of data today doesn't require transactional properties.
3. NoSQL provides a data model that maps better to these needs.



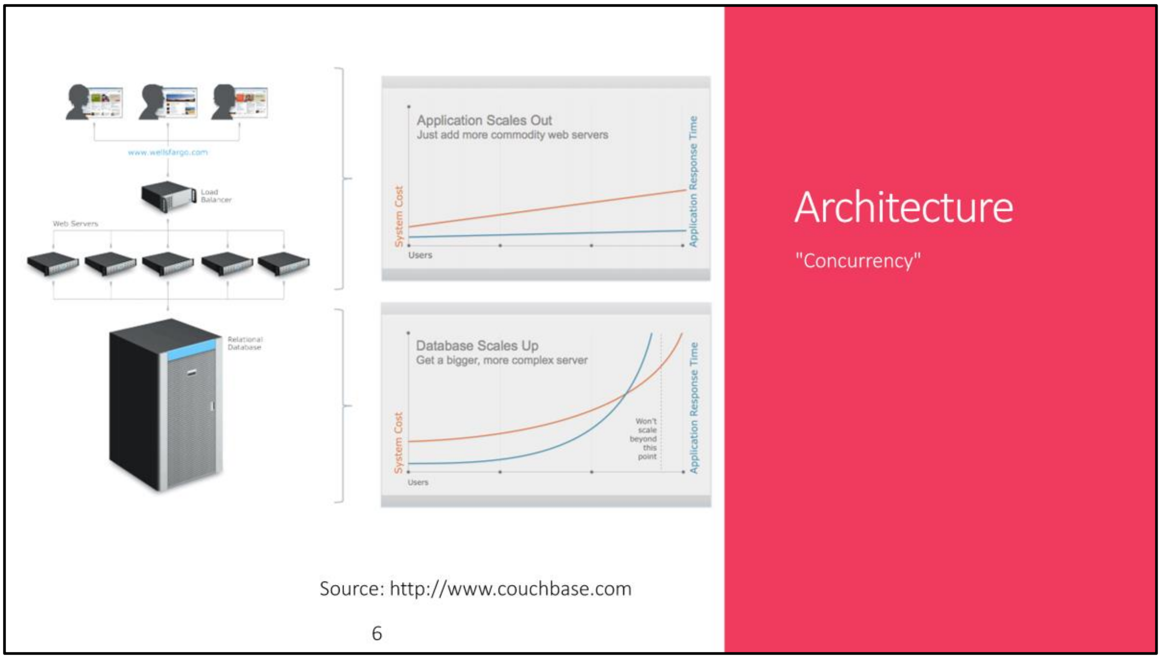
Connectedness

"Social networks"

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

5

1. Data now has much more complex relations. It has evolved from hyptertext, RSS, blogs(have backlinks) to highly complex social graphs.
2. No more efficient to represent in strict tables. - We need different **data models**. Graph databases.

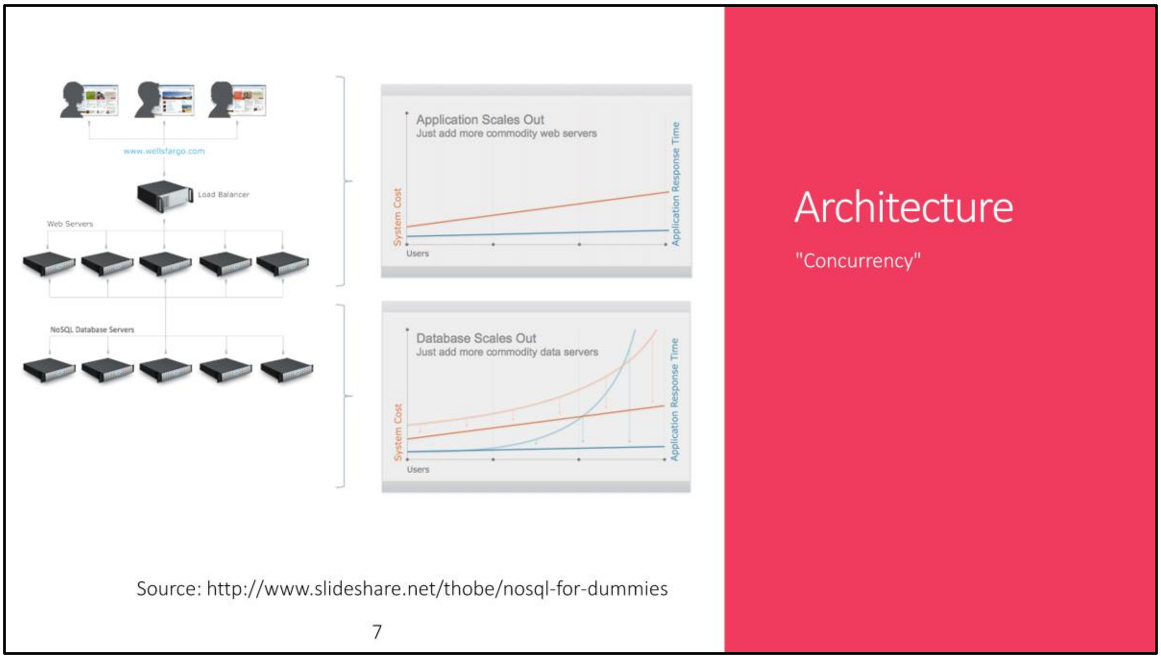

www.wellsfargo.com
Load Balancer
Web Servers
Relational Database
Application Scales Out
Just add more commodity web servers
System Cost
Application Response Time
Users
Database Scales Up
Get a bigger, more complex server
System Cost
Application Response Time
Won't scale beyond this point
Users

Architecture

"Concurrency"

Source: http://www.couchbase.com

1. Relational databases are fundamentally centralized. 3-tier systems. Scale up system.
2. To scale the application you add more web servers.
3. To support more concurrent users and/or store more data,
you need a bigger and bigger server with more CPUs, more memory, and more disk storage to keep all the tables .
4. Maintaining this single server becomes a headache both in terms of man power and cost.



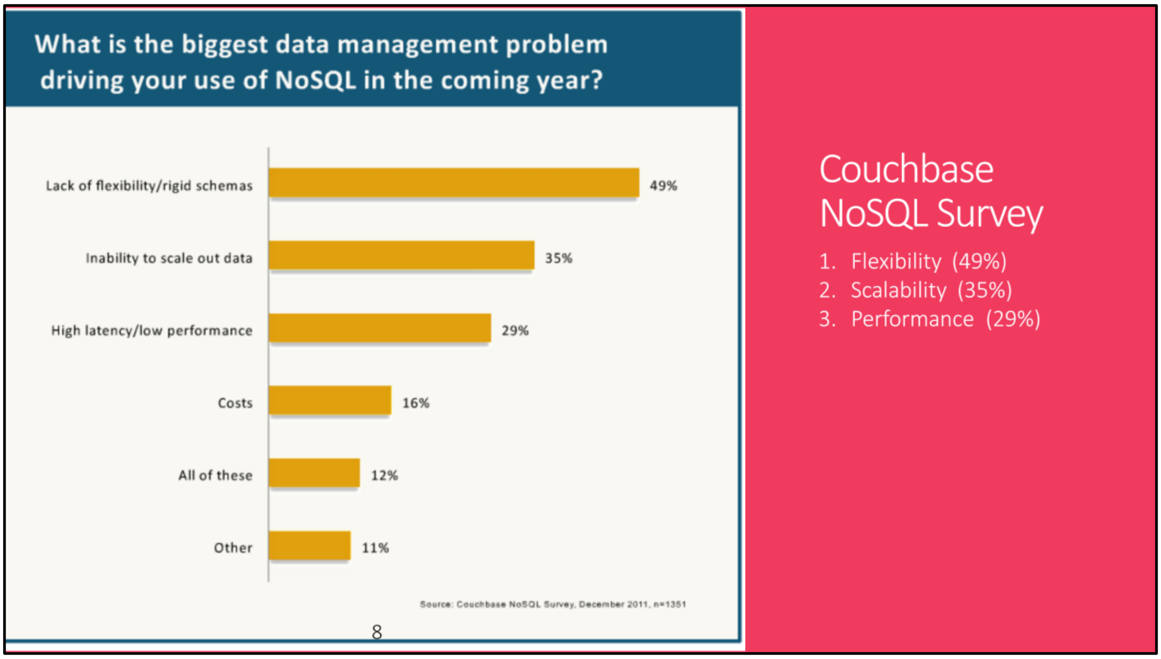
Source: http://www.slideshare.net/thobe/nosql-for-dummies

7

Architecture

"Concurrency"

Now we are moving towards distributed databases.
We'll talk more about this later - ACID properties. Relational databases aims for consistency .
In a distributed environment we need to make a choice because of CAP.



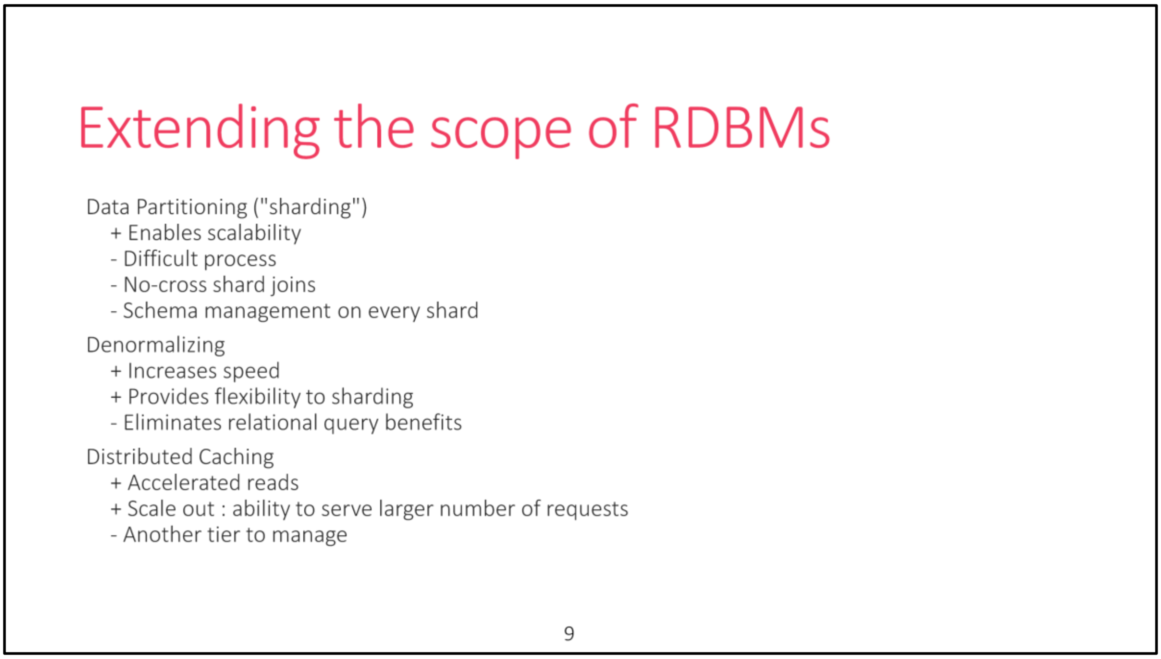
What is the biggest data management problem driving your use of NoSQL in the coming year?

Source: Couchbase NoSQL Survey, December 2011, n=1351

Couchbase NoSQL Survey

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

A survey done by couchbase.com shows that the major reason for choosing NoSQL databases are Flexibility and Scalability.



Extending the scope of RDBMs

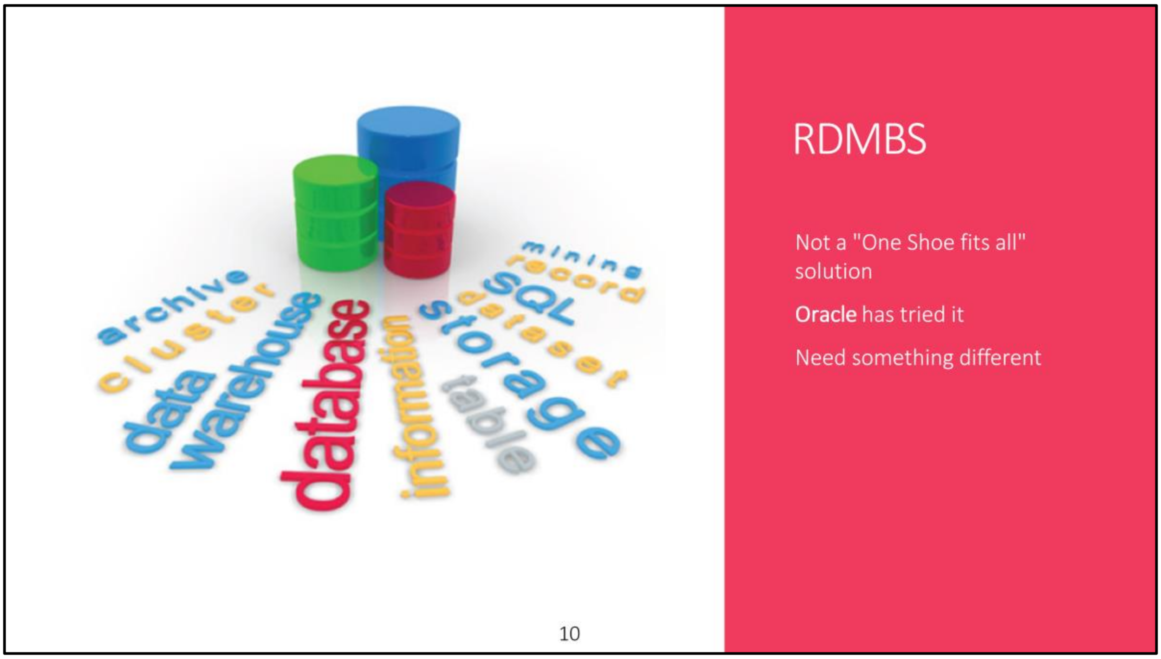
Data Partitioning ("sharding")
- + Enables scalability
- - Difficult process
- - No-cross shard joins
- - Schema management on every shard

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

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

1. SQL databases are very reliable and mature technologies.
People have tried to extend the scope by changing SQL databases to adapt to the new trends that we saw.
Distributed caching - offload reads, in memory cached, using memcached over SQL server. (highly common, lot of big companies use it)
Example: Zynga - roughly 600 memcached databases over 400 SQL databases.
Massive software - difficult management.


archive
cluster
data
warehouse
database
information
storage
table
dataset
SQL
record
mining

RDMBS

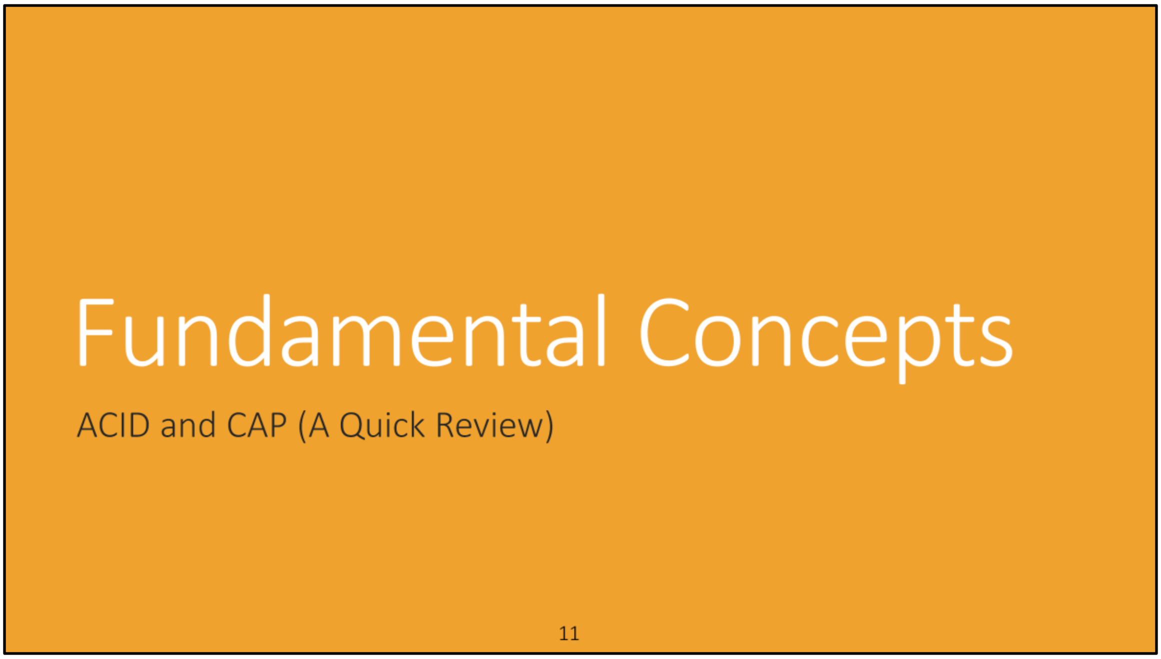
Not a "One Shoe fits all" solution

Oracle has tried it

Need something different

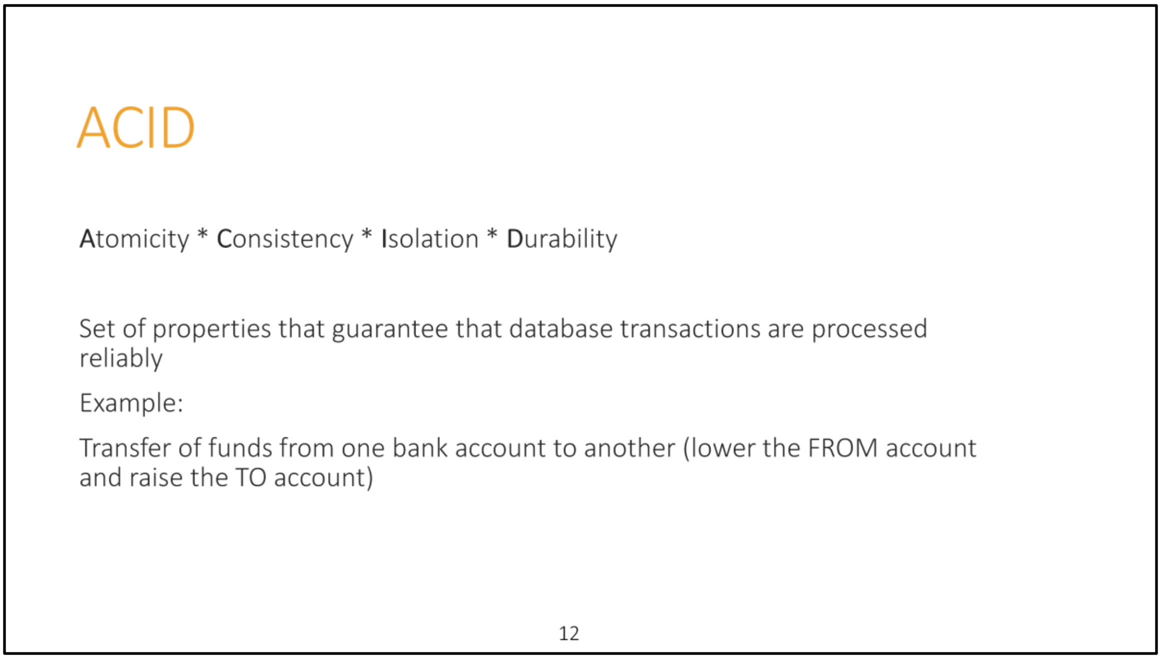
10

Lot of vendors have tried to extend the scope but what's evident is that one solution is not enough.



Fundamental Concepts

ACID and CAP (A Quick Review)



ACID

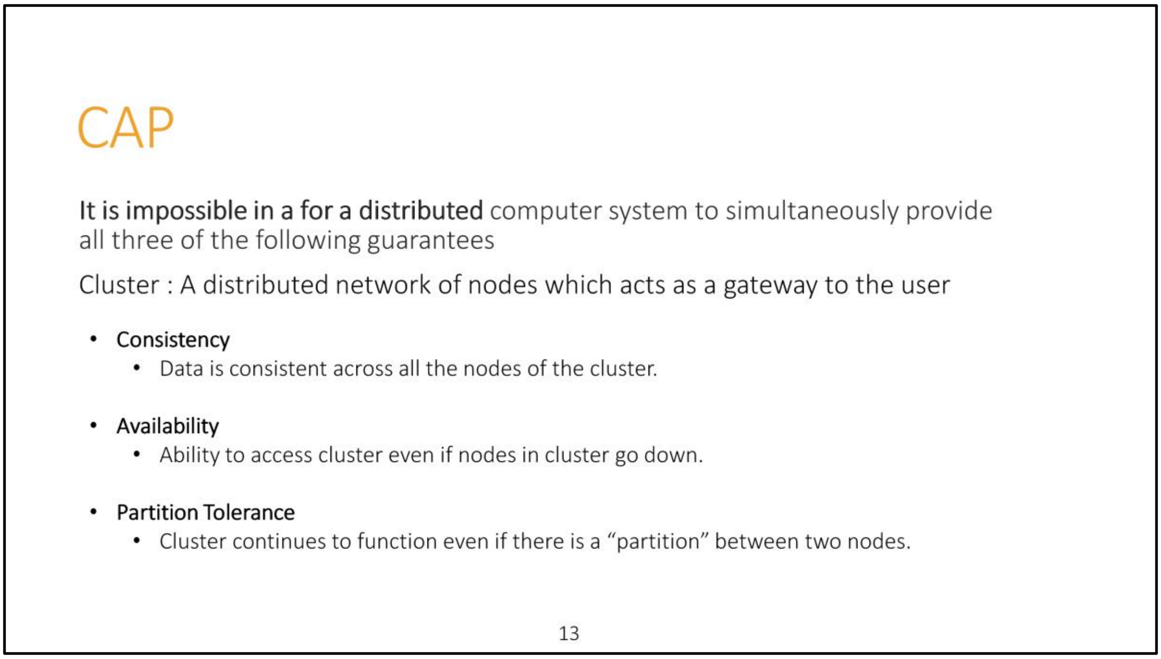
Atomicity * **C**onsistency * **I**solation * **D**urability

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)

Will spend a minute or two on ACID slides, basically a very quick review.



CAP

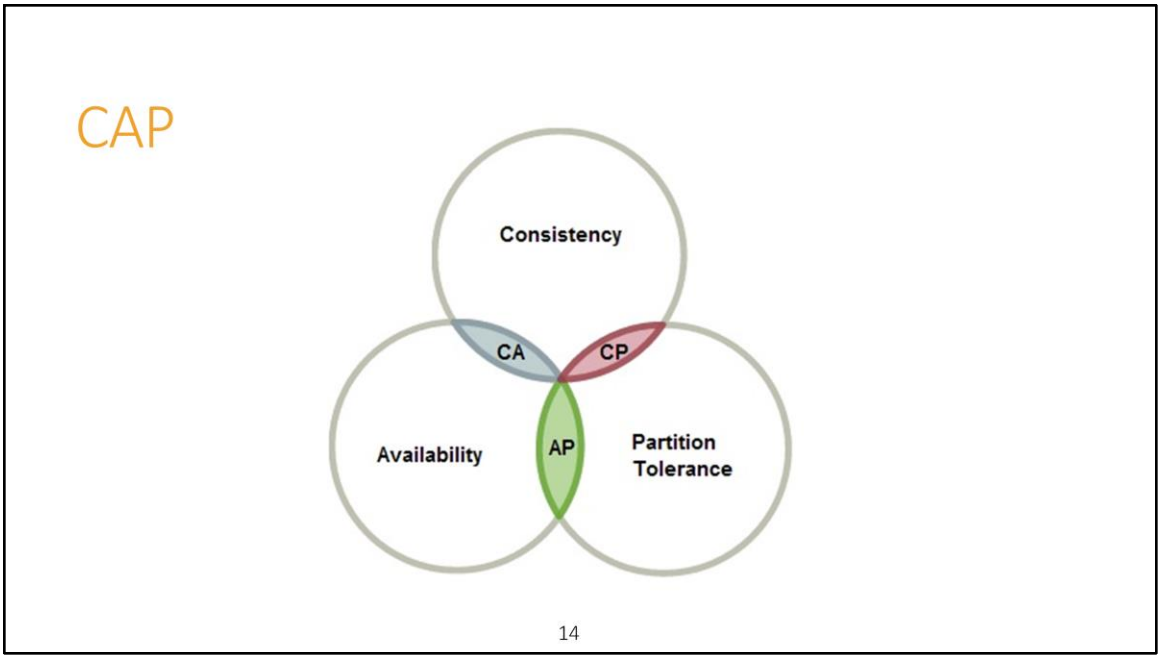
It is impossible in a for a distributed computer system to simultaneously provide all three of the following guarantees

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.

Single machines: partition tolerance is irrelevant. consistency and availability can be achieve on a single machine.

Consistency: so you can read or write to/from any node and get the same data.



CAP

Consistency

CA

CP

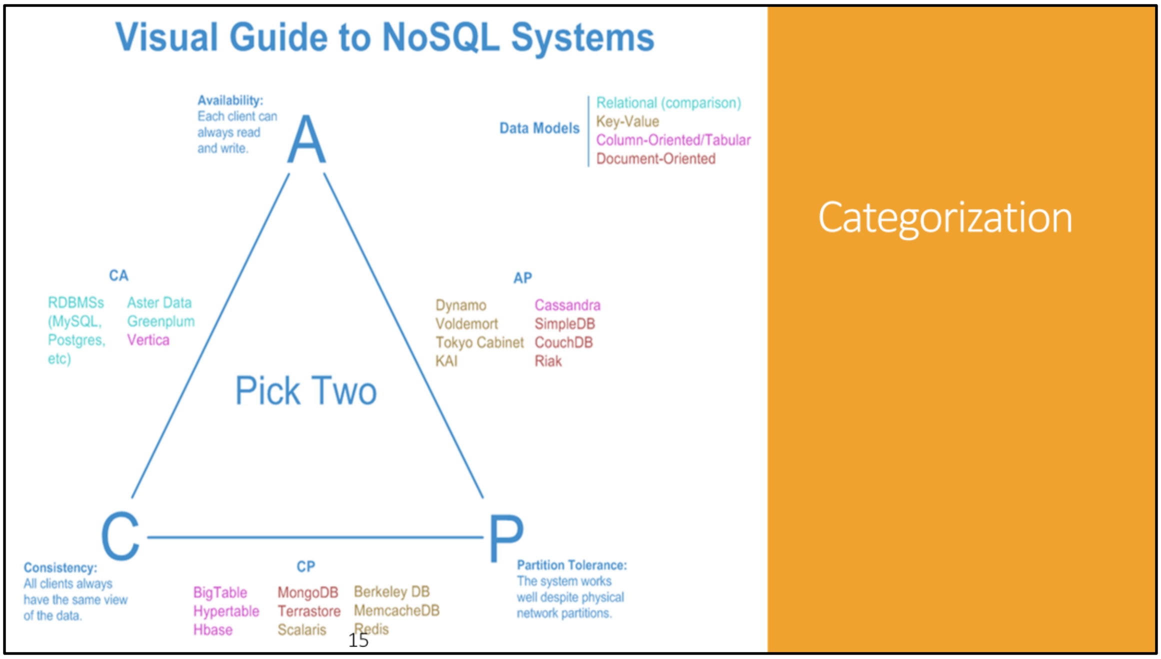
Availability

AP

Partition Tolerance

14

We will not spend much time on this, since there is a group that's presenting CAP in quite a detail. Only thing to take from this slide is that all three properties cannot be achieved at the same time.



Categorization

Visual Guide to NoSQL Systems

Availability: Each client can always read and write.

Data Models: Relational (comparison), Key-Value, Column-Oriented/Tabular, Document-Oriented

A

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.

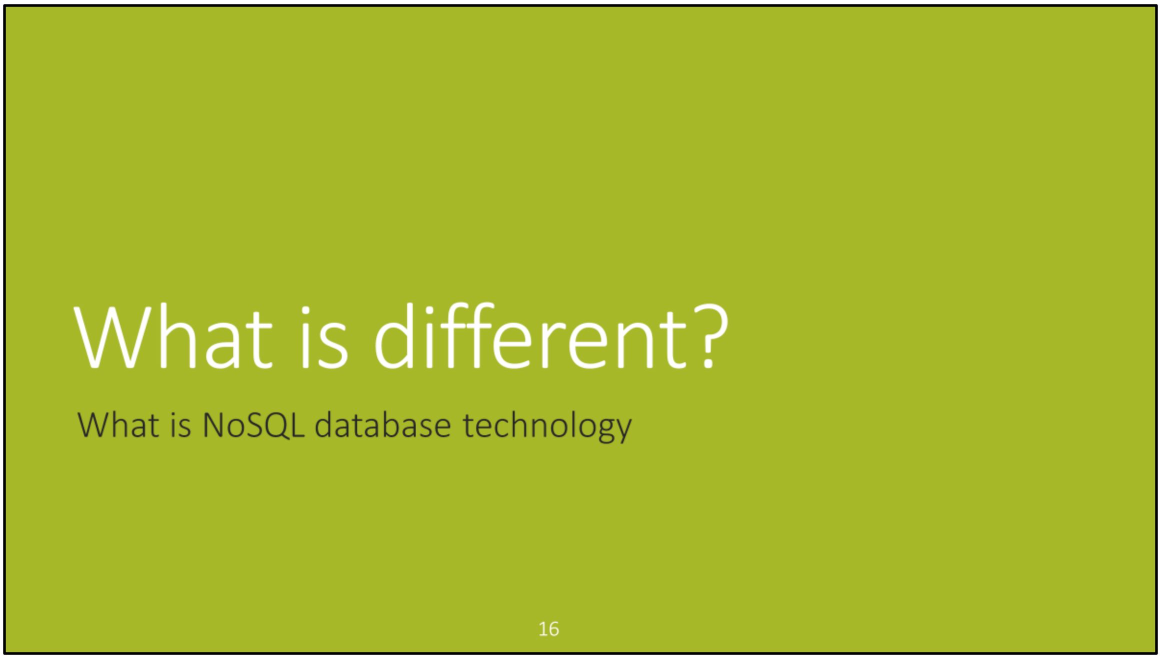
CP: BigTable, Hypertable, Hbase, MongoDB, Terrastore, Scalaris, Berkeley DB, MemcacheDB, Redis

Partition Tolerance: The system works well despite physical network partitions.

An illustration to show where most of the NoSQL and Relational databases lie on the CAP spectrum.

It is interesting to see that the databases following CA model are primarily relational databases, this is because, they are not built for partitioning and distributed structure.

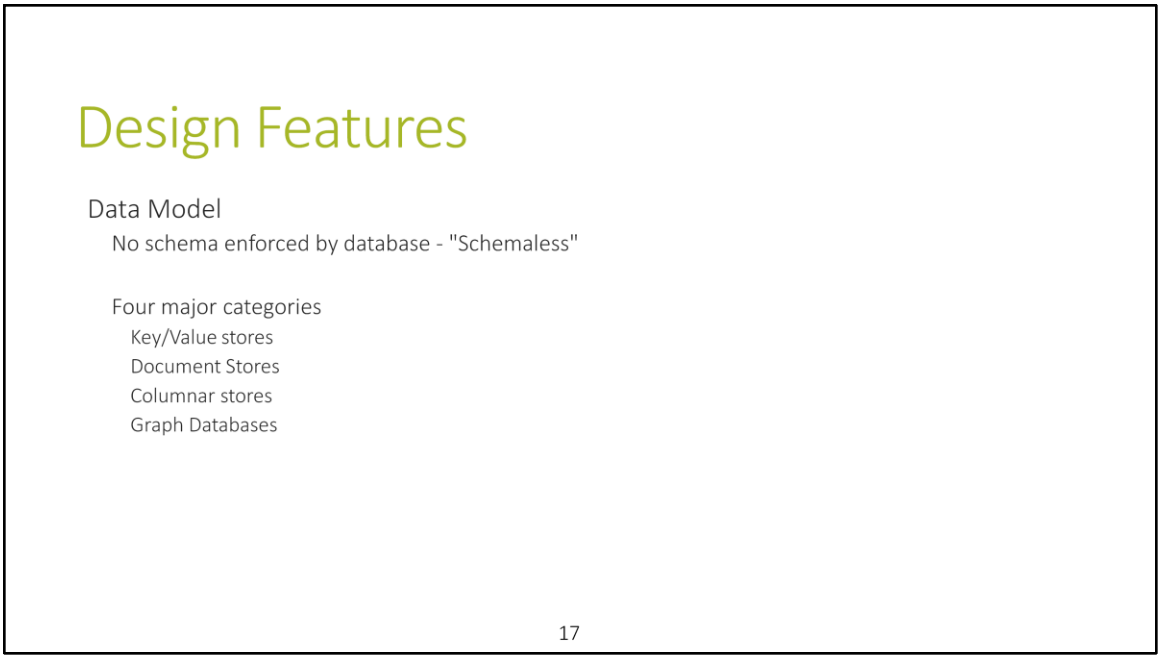
NoSQL databases either show CP model or AP model. We will discuss a single database from each as our case study.



What is different?

What is NoSQL database technology

Not just SQL

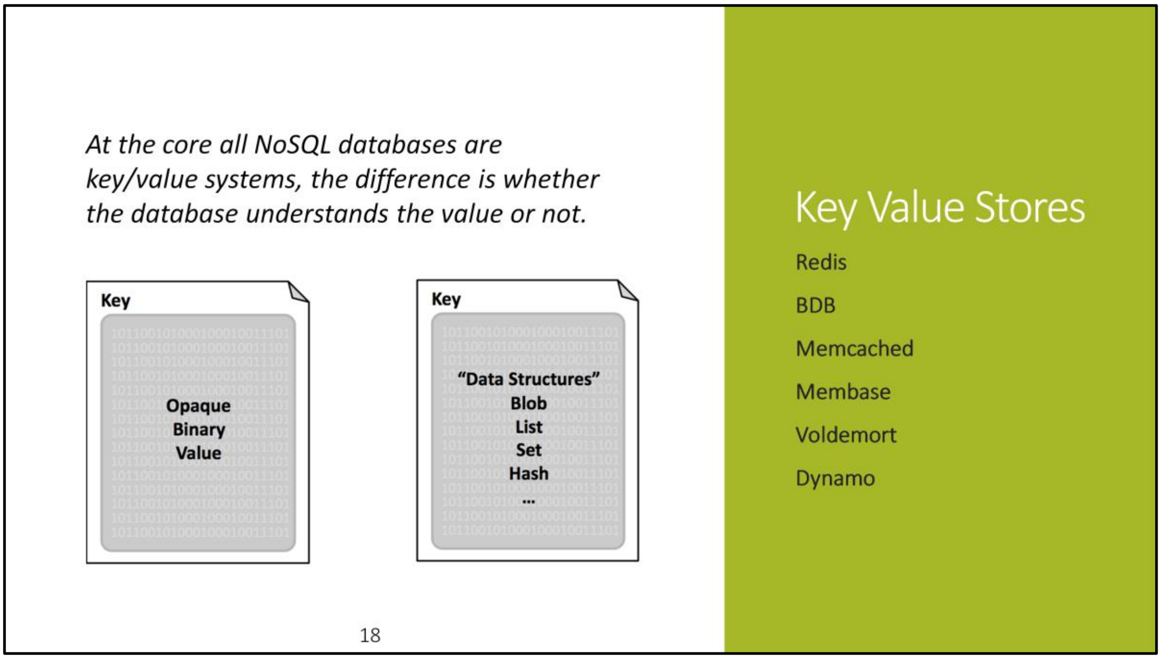


Design Features

Data Model

- No schema enforced by database - "Schemaless"
- Four major categories
  - Key/Value stores
  - Document Stores
  - Columnar stores
  - Graph Databases

1. A paradigm shift from the traditional data model. SQL databases enforce a strict schema, whereas NoSQL databases has a week notion of schema.
At the core all NoSQL databases are key/value systems, the difference is whether the database understands the value or not.
Different type of NoSQL databases have different properties. We'll see four major data models in a minute.
2. As we are moving towards distributed databases and not all the data is transactional we need a separate set of guarantees.



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

Key

Opaque
Binary
Value

Key

"Data Structures"
Blob
List
Set
Hash
…

18

Key Value Stores

Redis

BDB

Memcached

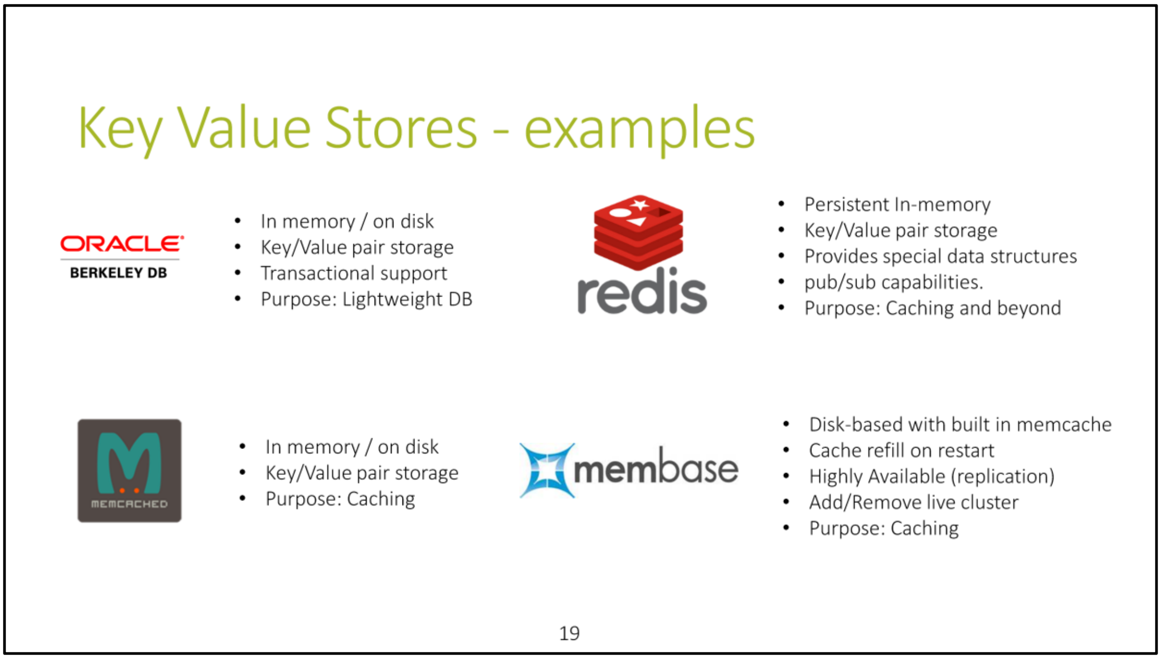
Membase

Voldemort

Dynamo

1. Key/Value stores don't understand the data in value. To query a key/value database you must have the key.
2. Redis is a very popular database with support of special data structures where values are of special kind. It can perform common operations on the provided dataset.
3. Another database that deserves a mention here is membase. It's an in-memory only database. Disk-based, fill cache, ADD/Remove nodes on the fly.

So you have datastores with different features like only in-memory, persistent, support for data structures -> this shows amount of diversity in NoSQL databases.



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

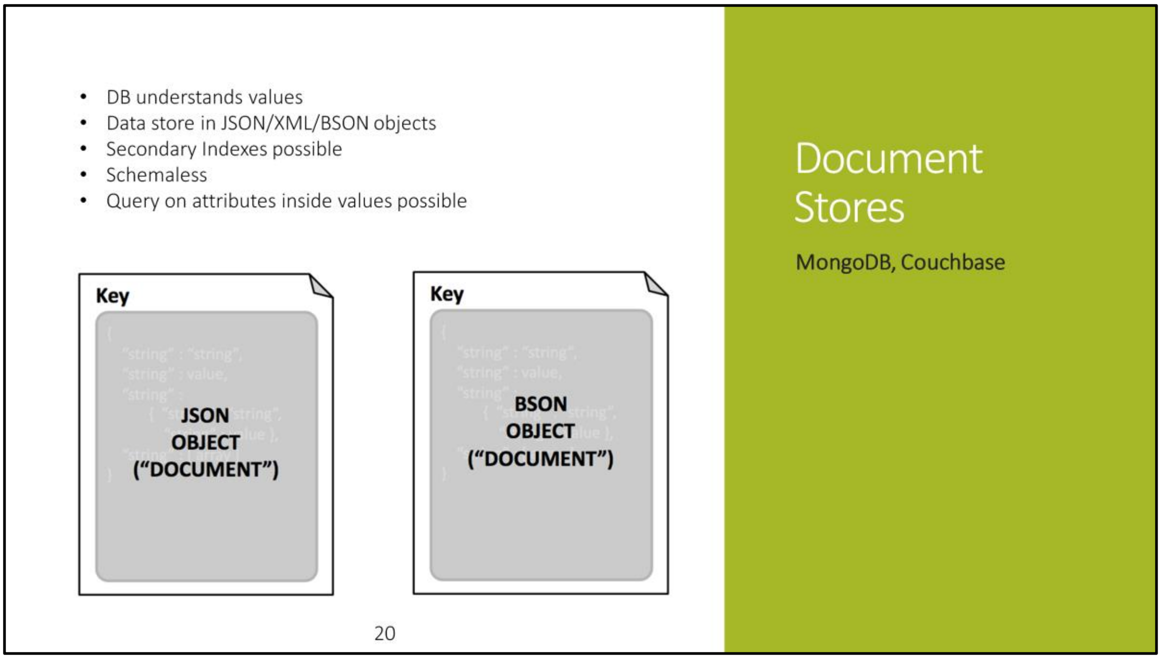
MEMCACHED

- 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

1. Key/Value stores don't understand the data in value. To query a key/value database you must have the key.
2. Redis is a very popular database with support of special data structures where values are of special kind. It can perform common operations on the provided dataset.
So you have datastores with different features like only in-memory, persistent, support for data structures -> this shows amount of diversity in NoSQL databases.
3. Apache Dynamo is also one of them, which we will discuss in detail as a case study.



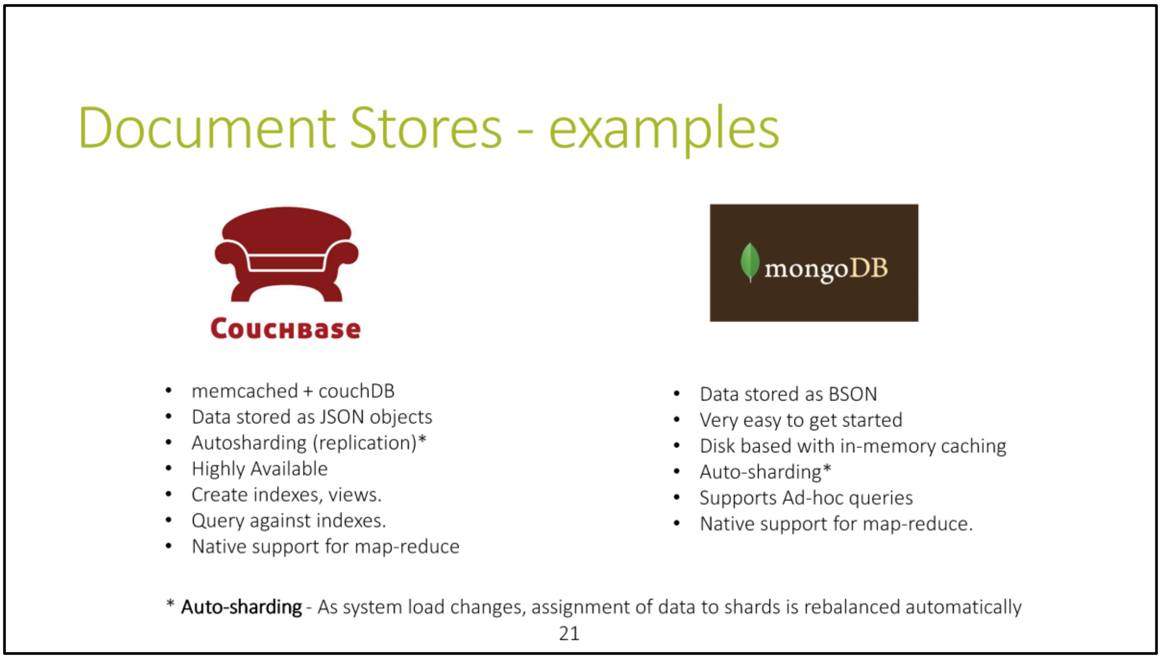
Document Stores

MongoDB, Couchbase

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

Key
JSON OBJECT ("DOCUMENT")
Key
BSON OBJECT ("DOCUMENT")

Instead of Value the database takes in a document which is semi structured data. Some use JSON, some XML and other BSON.



Document Stores - examples

Couchbase

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

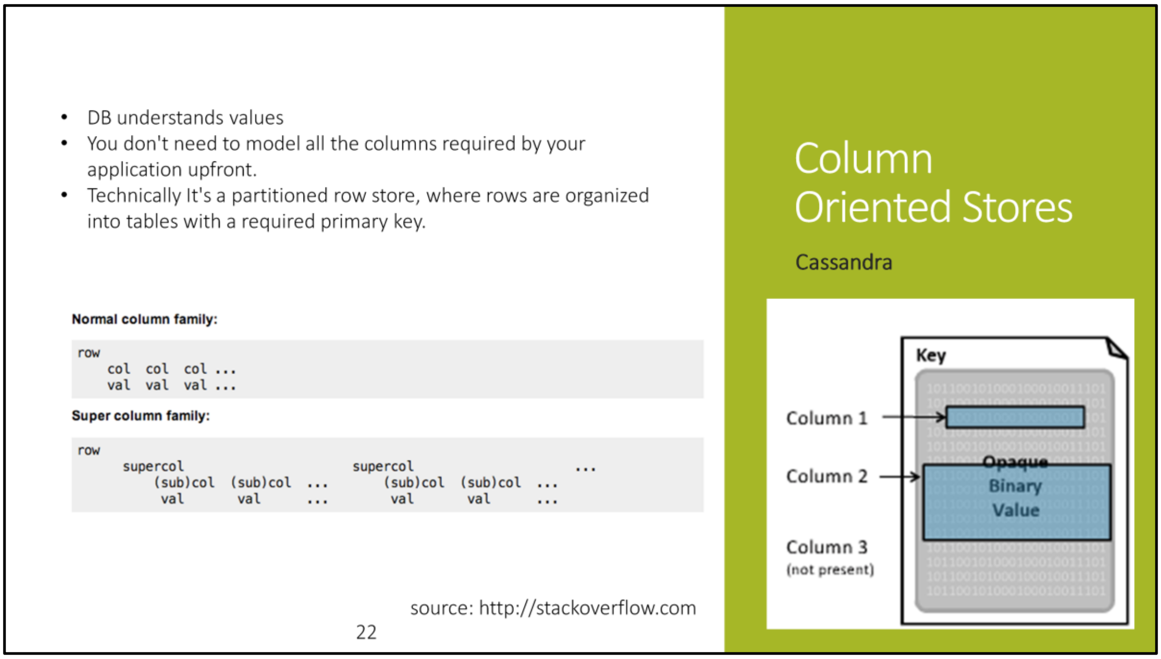
mongoDB

- 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

1. BSON - binary version of JSON objects. Higher performance on the wire and compact storage .

2. In couchbase you need to materialize views to make ad-hoc queries. Declare what your indexes will be, you can query.
MongoDB doesn't require xanti declaration of indexes to query.
Ad-hoc queries are queries that are created on the fly with a variable parameters.



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

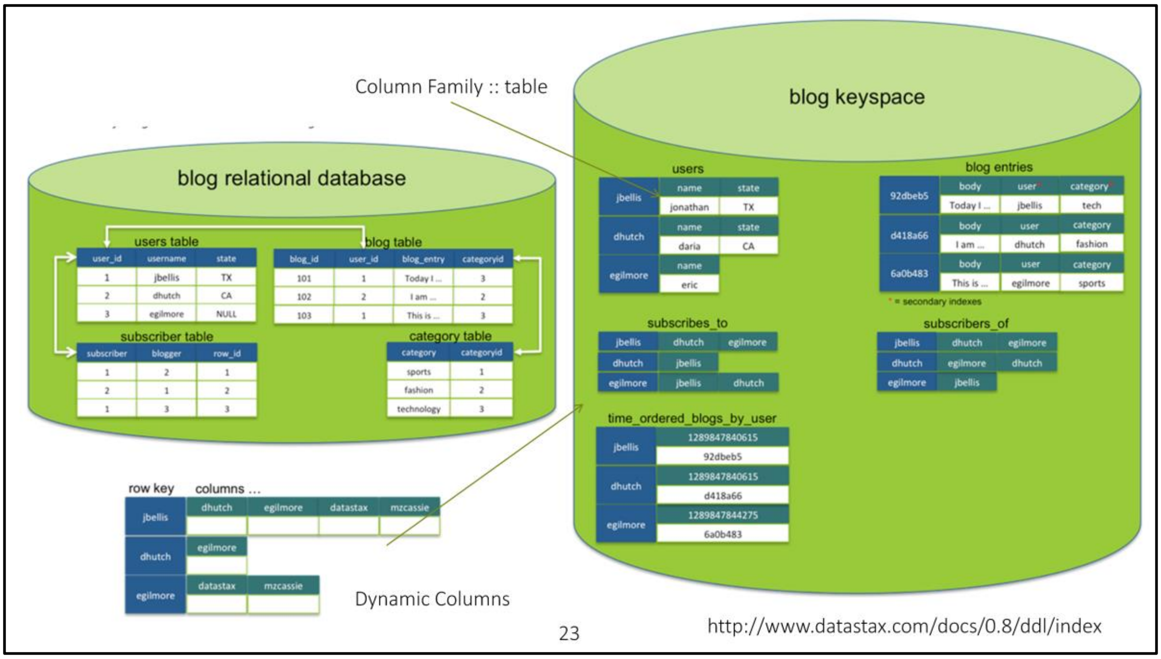
22

Concept is still the same. Key -> Value

Notion of column forms - i.e, instead of writing the whole document at a single physical location the document is now written split across these column forms/families.

Say a document has 10 columns or 10 attributes: you could write subsets of columns at particular locations so that queries on those columns are answered faster. This works well for predefined schema - HP Vertica.

Cassandra is a little different from this type of storage. Cassandra writes these to different family objects which by themselves are column dependent stores. This is driven not by the schema but by the queries that are expected to be answered.



Column Family :: table

blog relational database

users table

user_id	username	state
1	jbellis	TX
2	dhutch	CA
3	egilmore	NULL

blog table

blog_id	user_id	blog_entry	categoryid
101	1	Today I ...	3
102	2	I am ...	2
103	1	This is ...	3

subscriber table

subscriber	blogger	row_id
1	2	1
2	1	2
1	3	3

category table

category	categoryid
sports	1
fashion	2
technology	3

blog keyspace

users

jbellis	name	state
	jonathan	TX
dhutch	name	state
	daria	CA
egilmore	name	
	eric	

blog entries

92dbeb5	body	user*	category*
	Today I ...	jbellis	tech
d418a66	body	user	category
	I am ...	dhutch	fashion
6a0b483	body	user	category
	This is ...	egilmore	sports

* = secondary indexes

subscribes_to

jbellis	dhutch	egilmore
dhutch	jbellis	
egilmore	jbellis	dhutch

subscribers_of

jbellis	dhutch	egilmore
dhutch	egilmore	dhutch
egilmore	jbellis	

time_ordered_blogs_by_user

jbellis	1289847840615
	92dbeb5
dhutch	1289847840615
	d418a66
egilmore	1289847844275
	6a0b483

row key	columns ...			
jbellis	dhutch	egilmore	datastax	mzcassie
dhutch	egilmore			
egilmore	datastax	mzcassie		

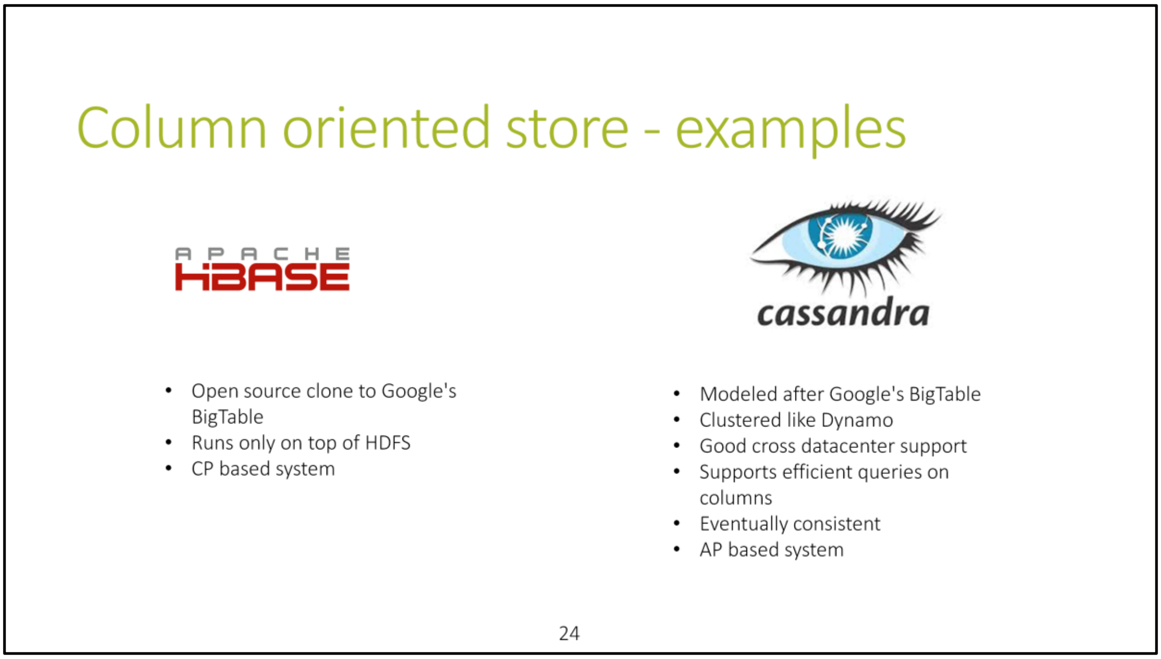
Dynamic Columns

23

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

BigTable coined the column oriented structure.

Joins as in relational databases is not supported. Usually different column family objects are there in a keyspace, each supporting one or more queries. To achieve the effect of joins, some extent of denormalization is necessary.


APACHE HBASE
cassandra

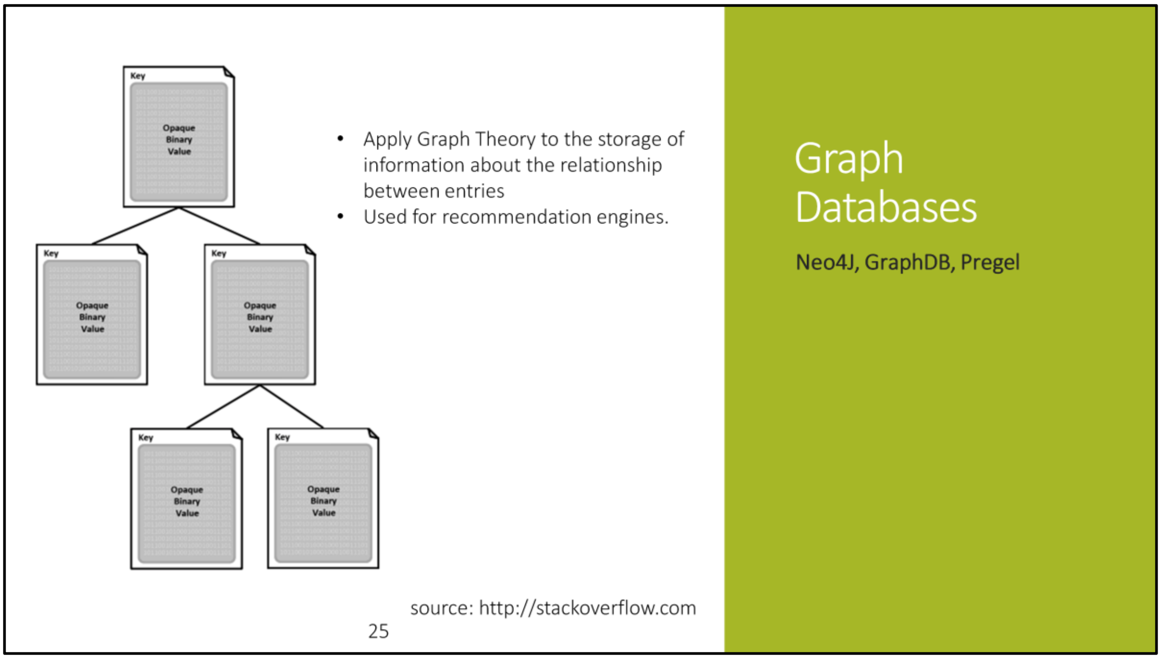
Column oriented store - examples

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

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

1. HBase runs only on top of HDFS while Cassandra can run on various file systems
2. Both are modeled as per BigTable's model
3. CP : Handles Consistency, Partioning out of the three in CAP.
4. AP : Handles Availability, Partioning out of the three in CAP.

Cassandra supports reads and writes in case of network partition and patches it up later thus resulting in eventual consistency whereas Couchbase prevents these network partitioned writes thus maintaining consistency at any time.



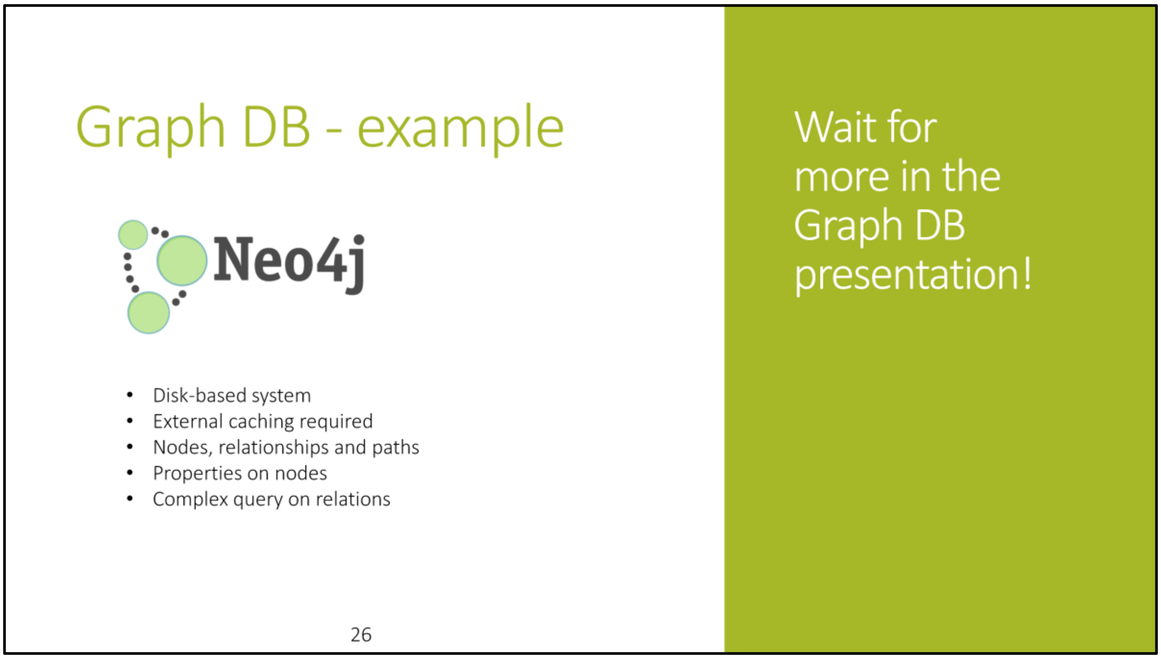
Graph Databases

Neo4J, GraphDB, Pregel

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

source: http://stackoverflow.com

Concept is still the same. Key -> Value



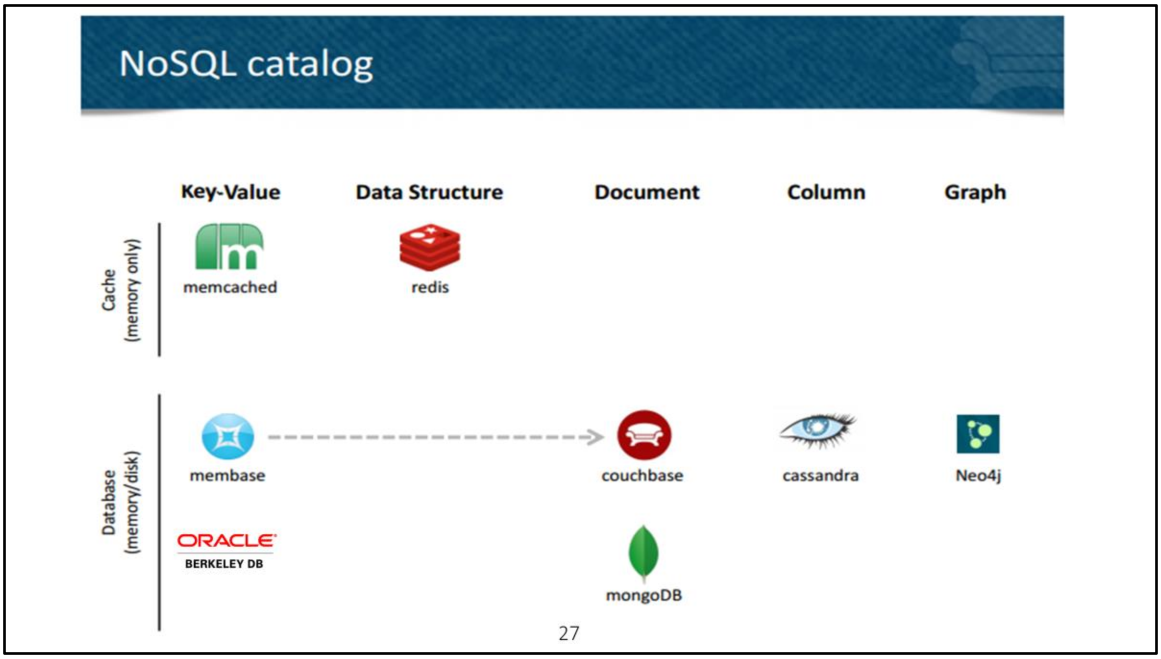
Graph DB - example

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!

When performing a write transaction on a slave each write operation will be synchronized with the master (locks will be acquired on both master and slave). When the transaction commits it will first be committed on the master and then, if successful, on the slave. To ensure consistency, a slave has to be up to date with the master before performing a write operation

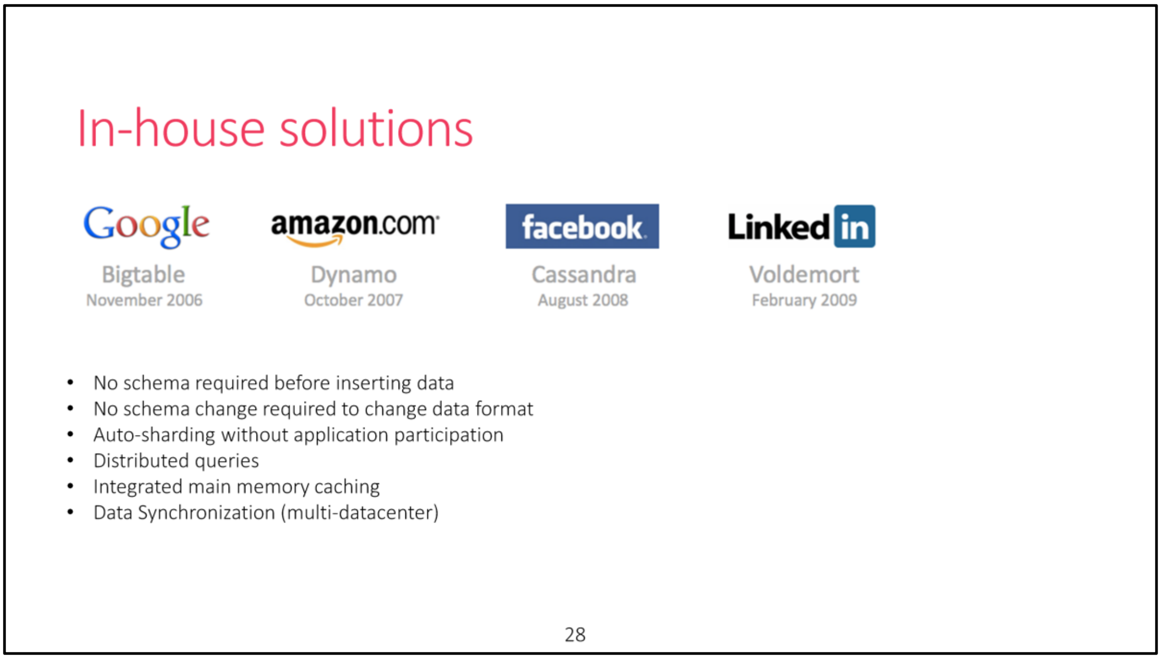


NoSQL catalog

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

27

Couchbase = Membase(front backend for HA)+ CouchDB (deeper backend to provide query functionality)
BDB can be setup as a persistent database. Depends on the config. Mostly used as embedded database.
BDB when compared to membase has much much lower concurrency rates supporting only in the lower tens.
Also membase is memcached cluster compatible whereas there is no implemented notion of bdb cluster.

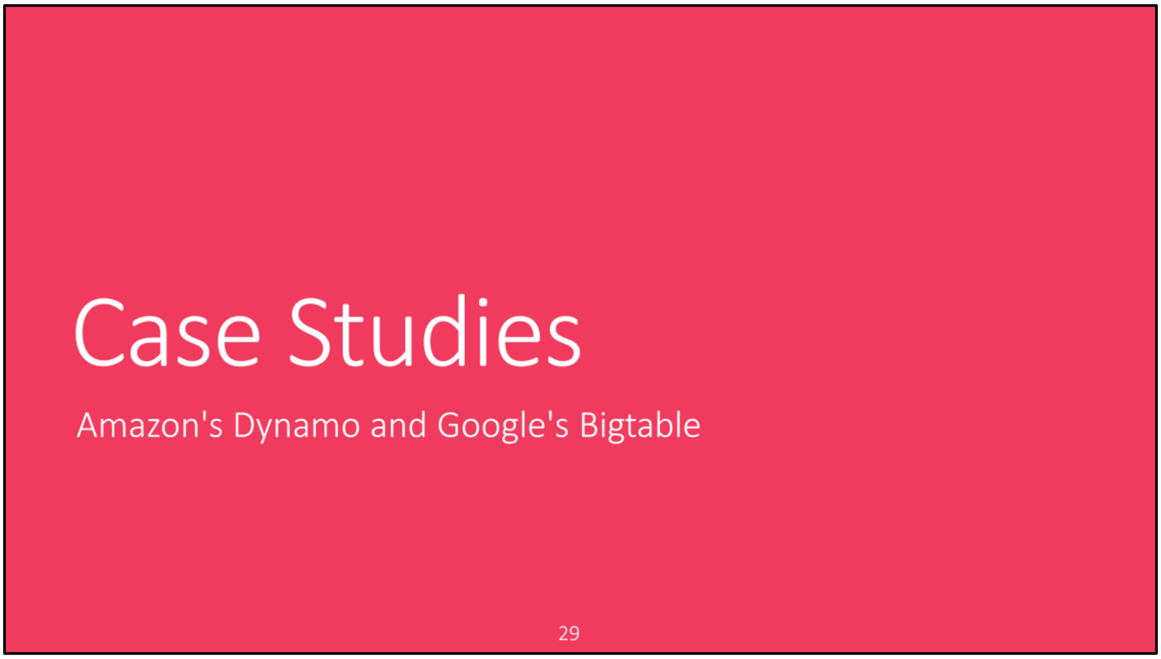


In-house solutions

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)

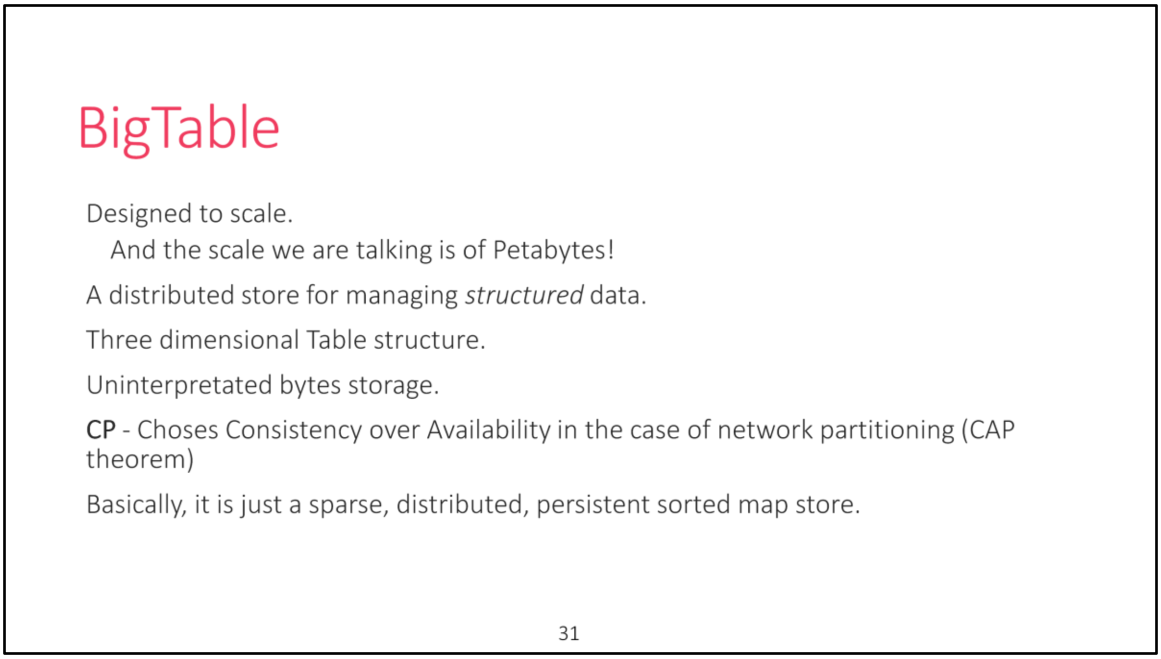
To address above problems lot of big companies developed their in-house solutions. Non-relational, cluster friendly, open-source,



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)

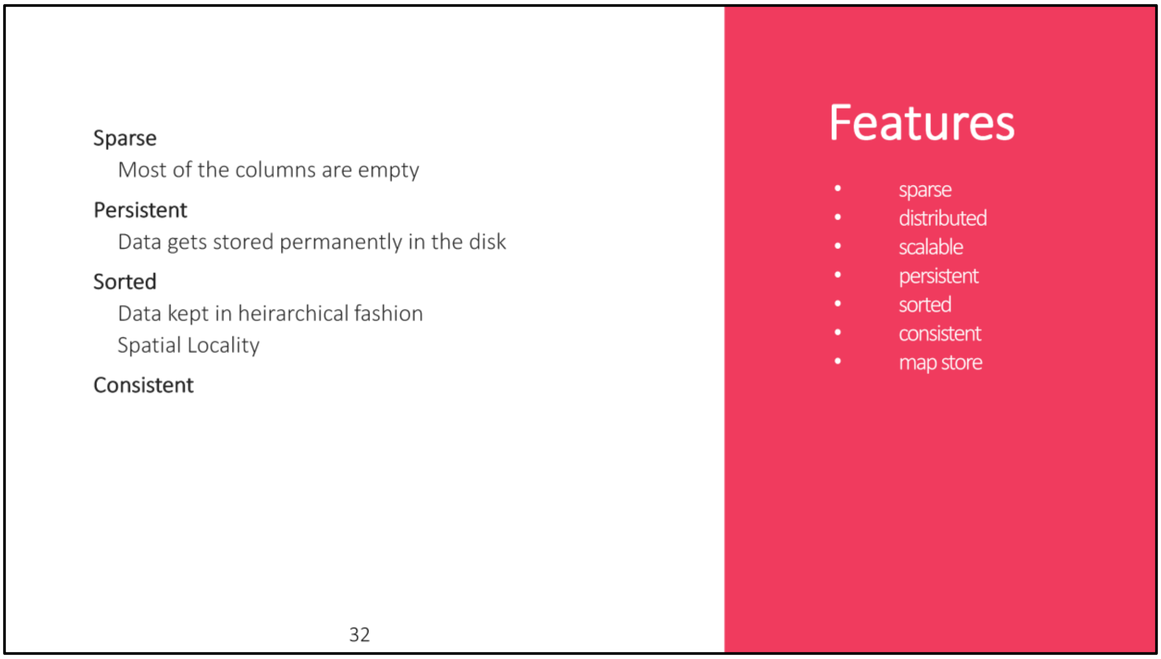
Basically, it is just a sparse, distributed, persistent sorted map store.

Structured because data is stored in an indexed map.

3-dimensional structure because it is just a large map that is indexed by a row key, column key, and a timestamp, which act as the dimensions. Will be more clear in the next slide.

Uninterpretated becuase Each value within the map is just an array of bytes that is eventually interpreted by the application.

Consistency over Availability: BigTable will preserve the guarantees of its atomic reads and writes by refusing to respond to some requests. It may decide to shut down entirely (like the clients of a single-node data store), refuse writes (like Two-Phase Commit), or only respond to reads and writes for pieces of data whose “master” node is inside the partition component (like Membase).It responds only after having quorom of locks [Paxos] which is managed by Chubby. [not in current scope]



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

32

Sparse : The table is sparse, meaning that different rows in a table may use different columns, with many of the columns empty for a particular row.

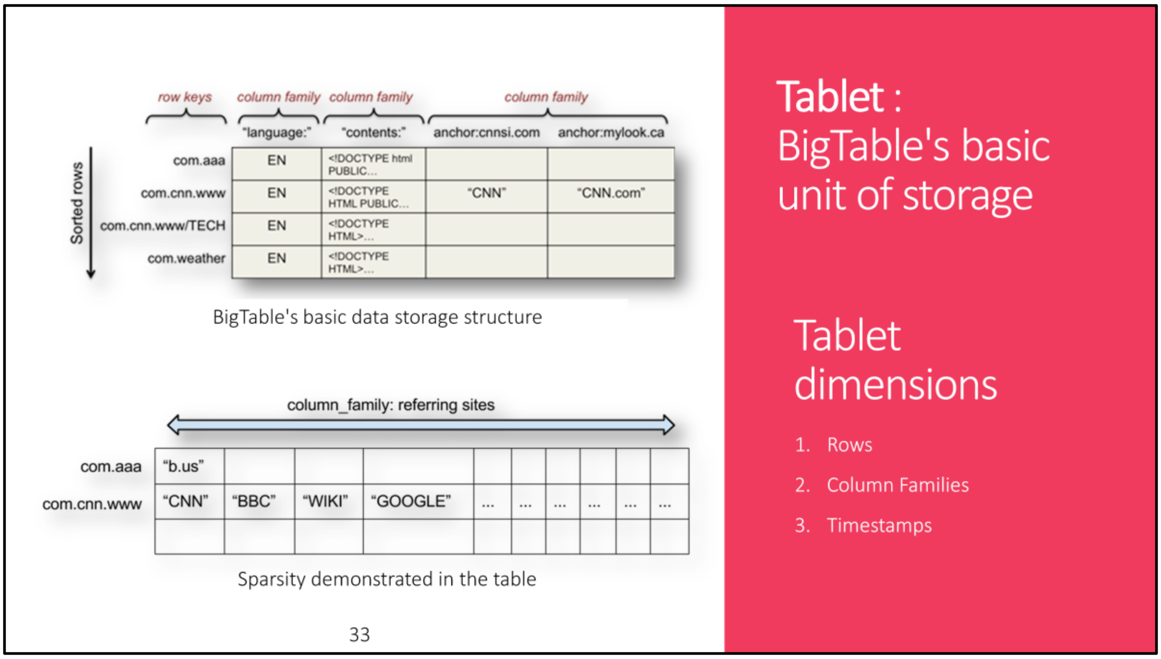
Distributed : BigTable's data is distributed among many independent machines. At Google, BigTable is built on top of GFS (Google File System). The Apache open source version of BigTable, HBase, is built on top of HDFS (Hadoop Distributed File System) or Amazon S3. The table is broken up among rows, with groups of adjacent rows managed by a server. A row itself is never distributed.

Scalable : Without changing applications, more and more nodes can be added to the network to make the cluster more scalable.

Sorted

A key is hashed to a position in a table. BigTable sorts its data by keys. This helps keep related data close together, usually on the same machine — assuming that one structures keys in such a way that sorting brings the data together. For example, if domain names are used as keys in a BigTable, it makes sense to store them in reverse order to ensure that related domains are close together.

map A **map** is an associative array; a data structure that allows one to look up a value to a corresponding key quickly. BigTable is a collection of (key, value) pairs where the key identifies a row and the value is the set of columns.



row keys	column family	column family	column family	
	"language:"	"contents:"	anchor:cnnsi.com	anchor:mylook.ca
com.aaa	EN	<!DOCTYPE html PUBLIC...		
com.cnn.www	EN	<!DOCTYPE HTML PUBLIC...	"CNN"	"CNN.com"
com.cnn.www/TECH	EN	<!DOCTYPE HTML>...		
com.weather	EN	<!DOCTYPE HTML>...		

BigTable's basic data storage structure

com.aaa	"b.us"									
com.cnn.www	"CNN"	"BBC"	"WIKI"	"GOOGLE"	…	…	…	…	…	…

Sparsity demonstrated in the table

33

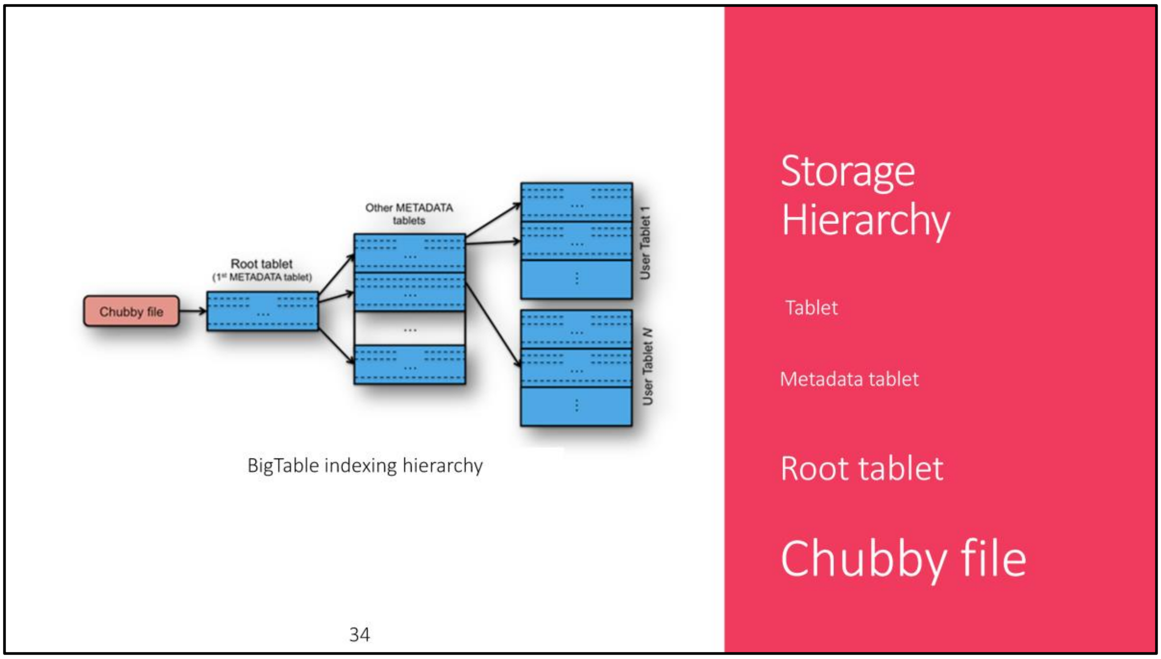
Tablet : BigTable's basic unit of storage

Tablet dimensions

1. Rows
2. Column Families
3. Timestamps

A table is indexed by rows. Each row contains one or more named **column families**. Column families are defined when the table is first created. Within a column family, one may have one or more named columns. All data within a column family is usually of the same type.

The implementation of BigTable usually compresses all the columns within a column family together. Columns within a column family can be created on the fly. Rows, column families and columns provide a three-level naming hierarchy in identifying data.
To get data from BigTable, you need to provide a fully-qualified name in the form *column-family:column*.



BigTable indexing hierarchy

34

Storage Hierarchy

Tablet

Metadata tablet

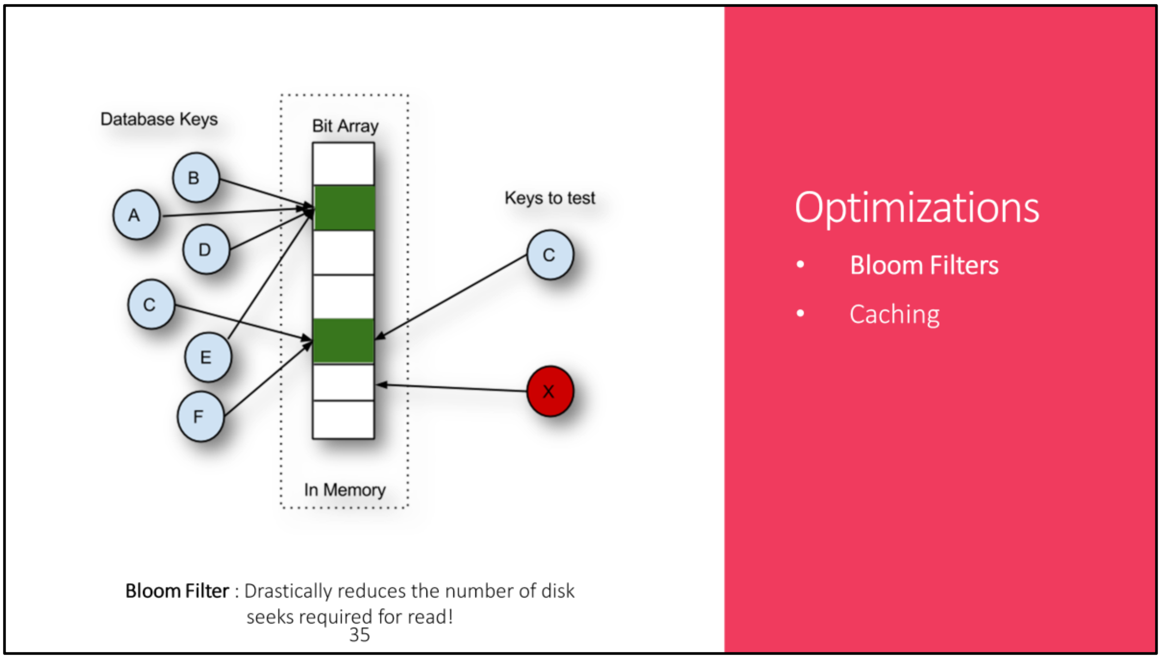
Root tablet

Chubby file

Chubby is a highly available and persistent distributed lock service that manages leases for resources and stores configuration information.
In BigTable, Chubby is used to:

- ensure there is only one active master
- store the bootstrap location of BigTable data
- discover tablet servers

Locating rows within a BigTable is managed in a three-level hierarchy. The root (top-level) tablet stores the location of all **Metadata tablets** in a special **Metadata tablet**. Each Metadata table contains the location of user data tablets. This table is keyed by node IDs and each row identifies a tablet's table ID and end row. For efficiency, the client library caches tablet locations.



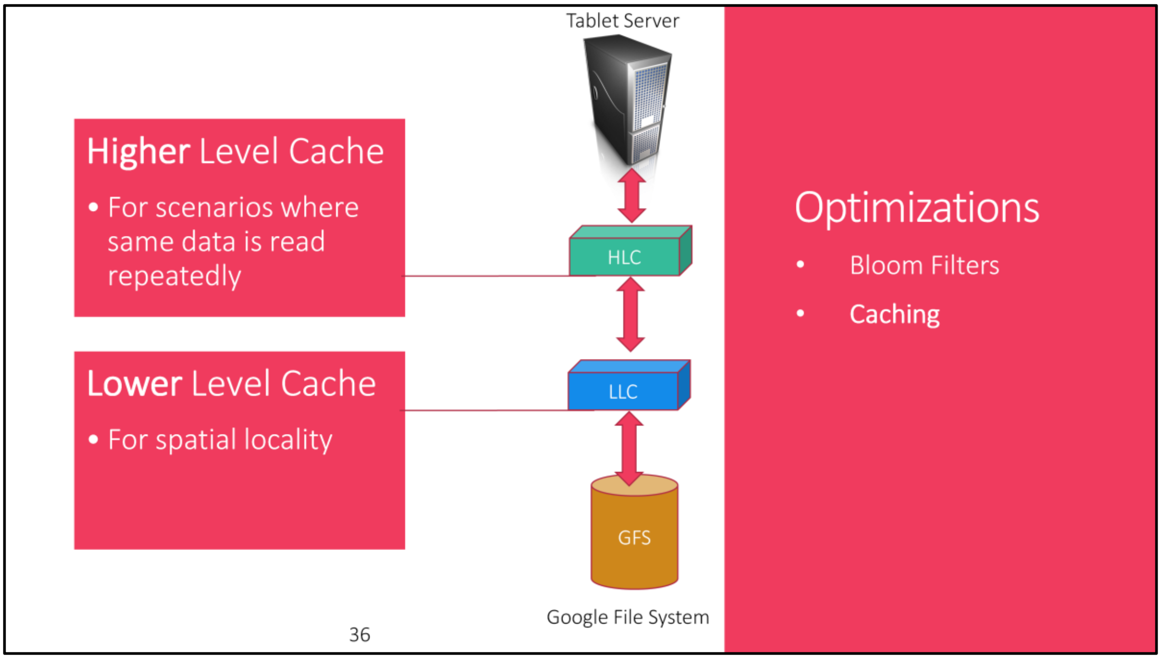
Optimizations

- **Bloom Filters**
- Caching

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

Need of Bloom Filters:

Typically, a read operation has to read from the user tables that make up the state of a tablet. If these are not in memory , we may end up doing many disk accesses. We reduce the number of accesses by allowing clients to specify that Bloom filters should be created for these user tables. A Bloom filter allows us to ask whether an user table might contain any data for a specified row/column pair. Thus, a small amount of tablet server memory used for storing Bloom filters drastically reduces the number of disk seeks required for read operations. Interesting, isn't it!



Optimizations

- Bloom Filters
- **Caching**

Higher Level Cache

- For scenarios where same data is read repeatedly

Lower Level Cache

- For spatial locality

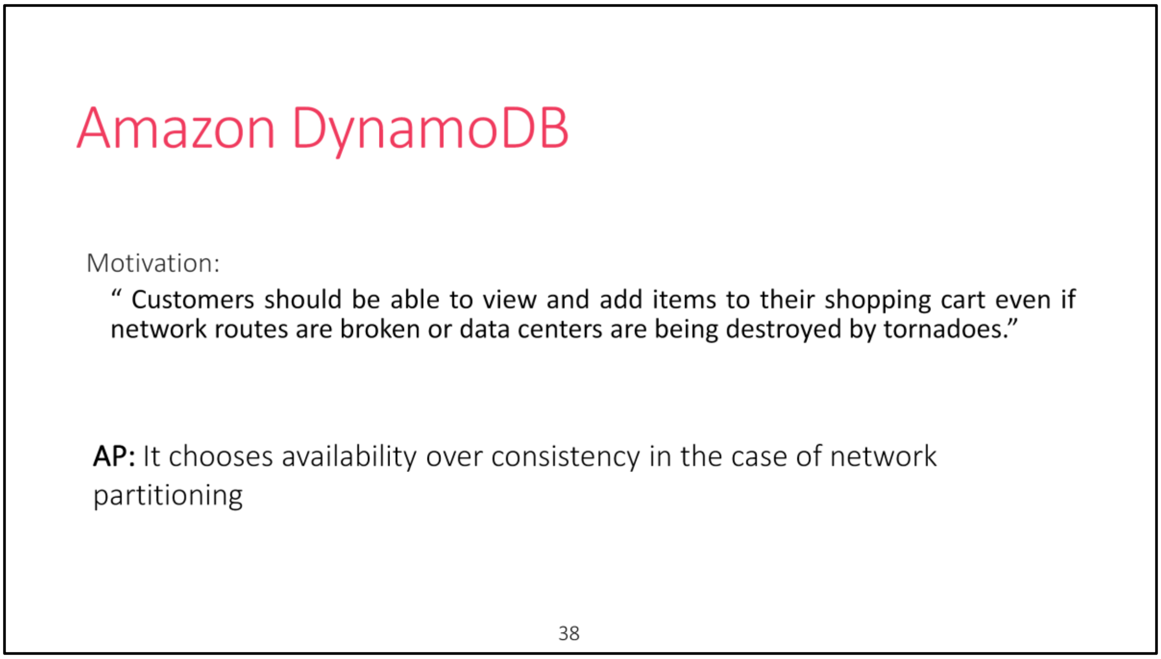
36

To improve read performance, tablet servers use two levels of caching.

The Scan Cache is a higher level cache that caches the key-value pairs returned by the user table interface to the tablet server code. It is most useful for applications that tend to read the same data repeatedly.

The Block Cache is a lower-level cache that caches row blocks that were read from GFS. It is useful for applications that tend to read data that is close to the data they recently read (e.g., sequential reads, or random reads of different columns in the same locality group within a hot row)

Amazon DynamoDB



Amazon DynamoDB

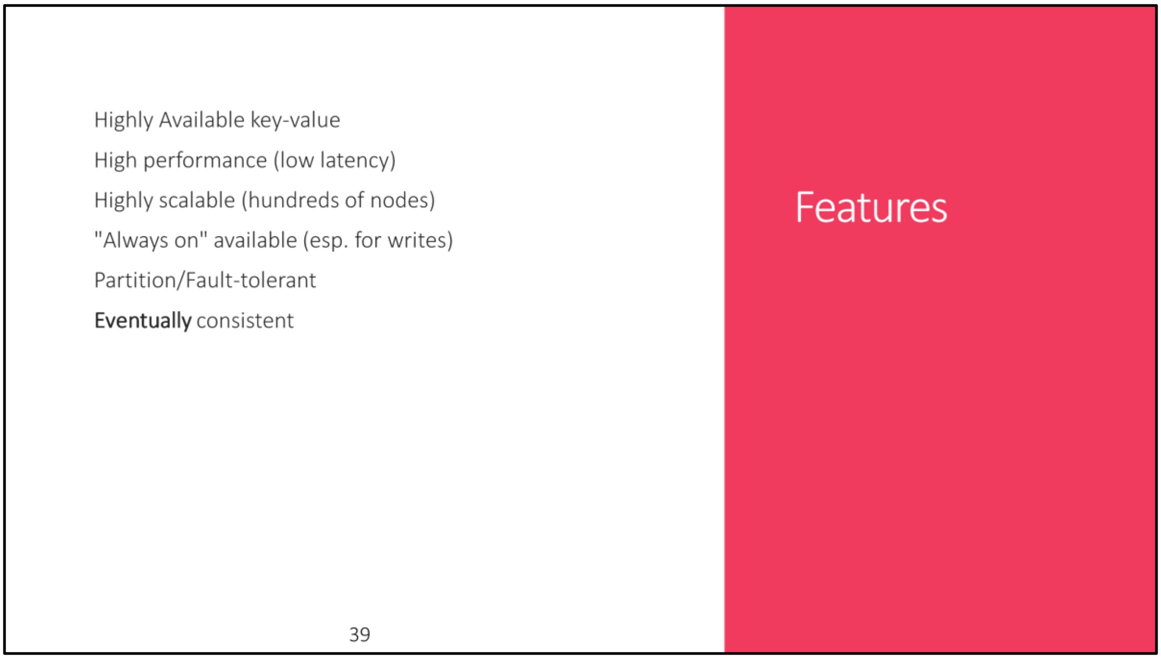
Motivation:

> “ 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

38

DynamoDB is database from amazon that they designed to solve their availability issues. Lot of their services didn't need transactional capabilities, and they required simple key value access. They were ready to tolerate some inconsistency (for example, an item may appear in the shopping cart after you have deleted it), however you should always be able to add items to the shopping cart even in presence of failures.



Features

Highly Available key-value

High performance (low latency)

Highly scalable (hundreds of nodes)

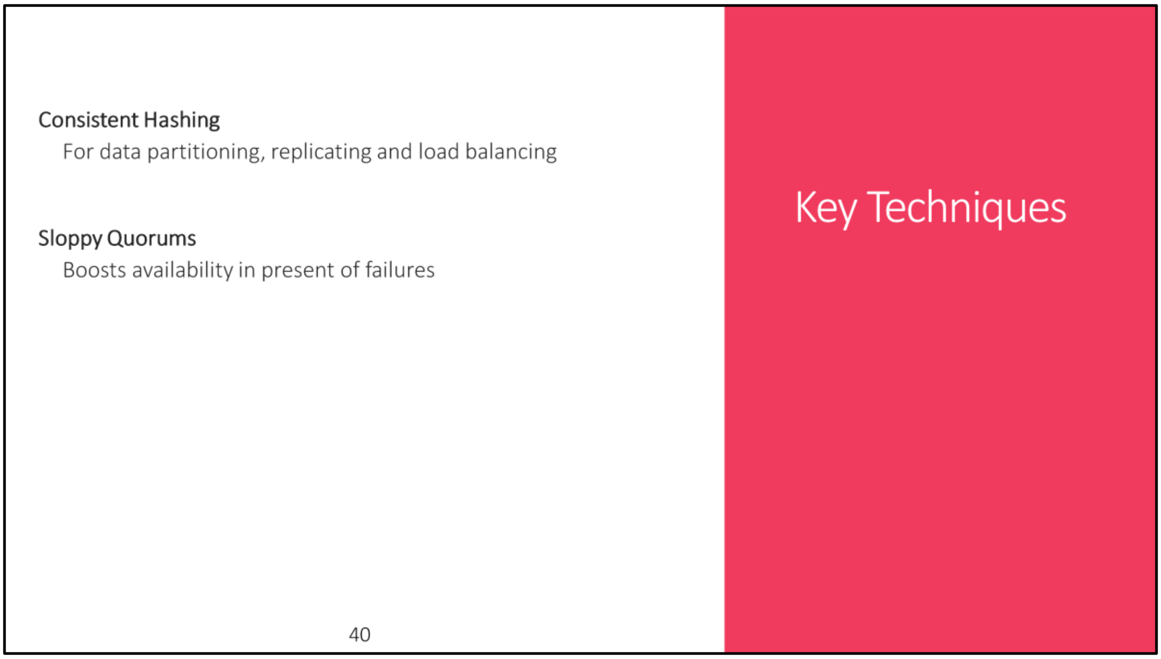
"Always on" available (esp. for writes)

Partition/Fault-tolerant

Eventually consistent

39

low latency, SLA (service level agreement) of serving 99.9% of requests with response within 300ms at a max rate of 500req/sec



Key Techniques

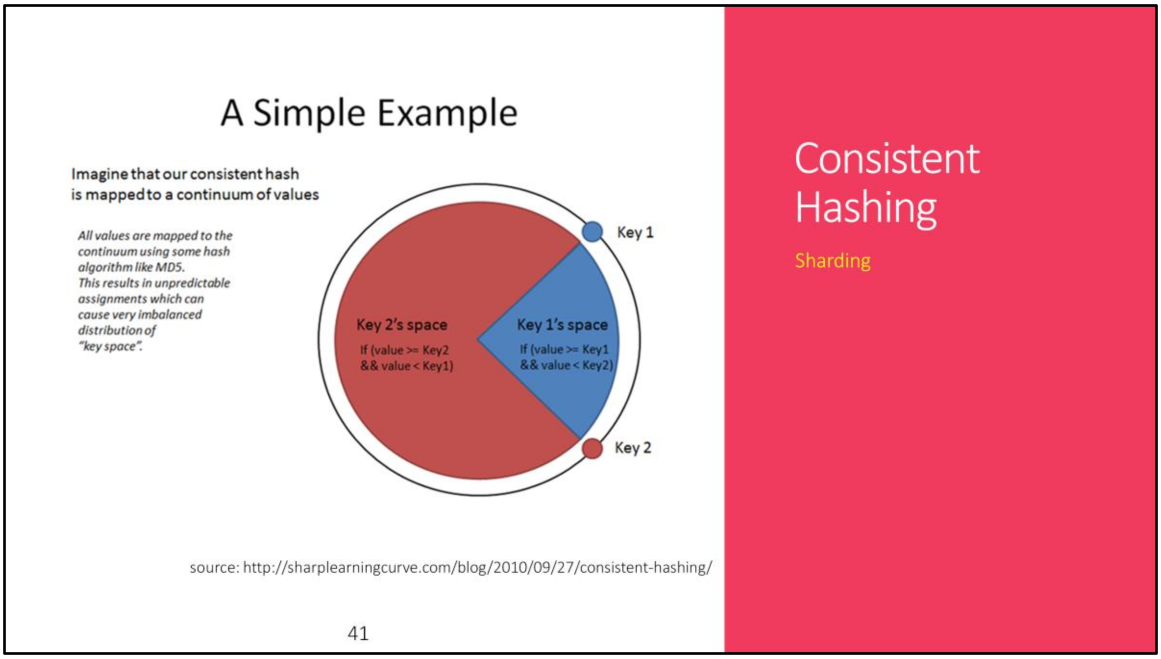
Consistent Hashing

For data partitioning, replicating and load balancing

Sloppy Quorums

Boosts availability in present of failures

Key techniques that the dynamo chooses.



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".

Key 1
Key 2
Key 2's space
If (value >= Key2 && value < Key1)
Key 1's space
If (value >= Key1 && value < Key2)

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

41

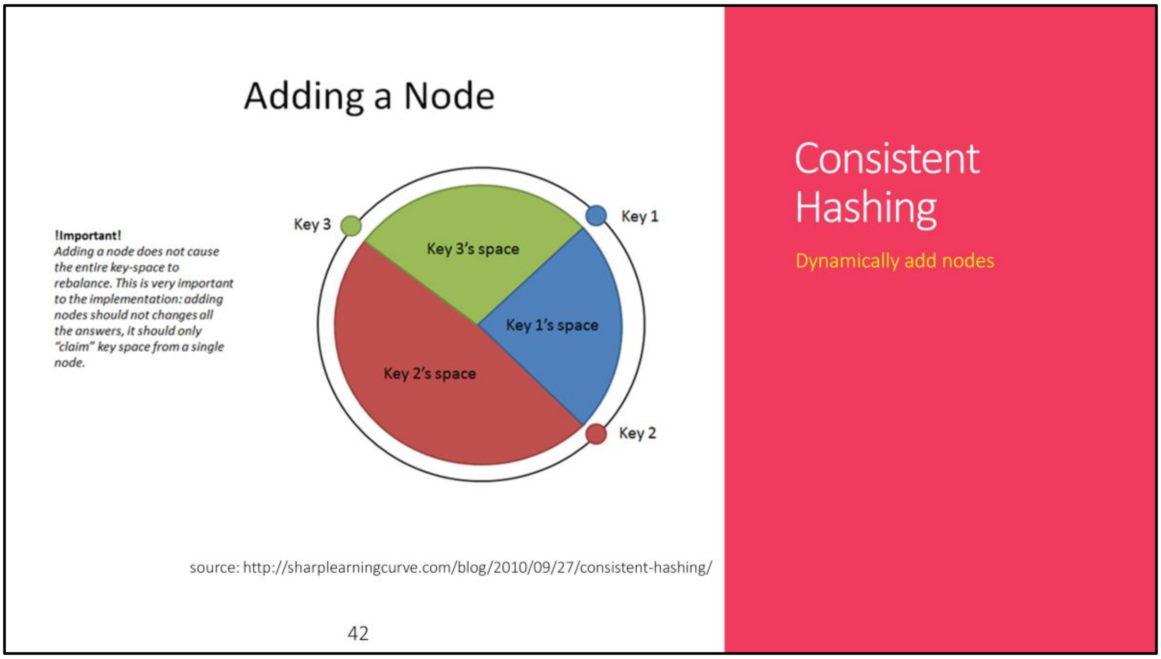
Consistent Hashing

Sharding

Dynamo uses consistent hashing to distribute content to nodes. Ring is the core of consistent hashing. In consistent hashing you map your data to points on ring. Ring is divided into regions and each region is then mapped to physical servers.

However this approach may lead to load imbalance.

allows you to have diverse set of machines by assigning diff. virtual nodes. Moreover it allows you add/remove nodes on the fly.



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.

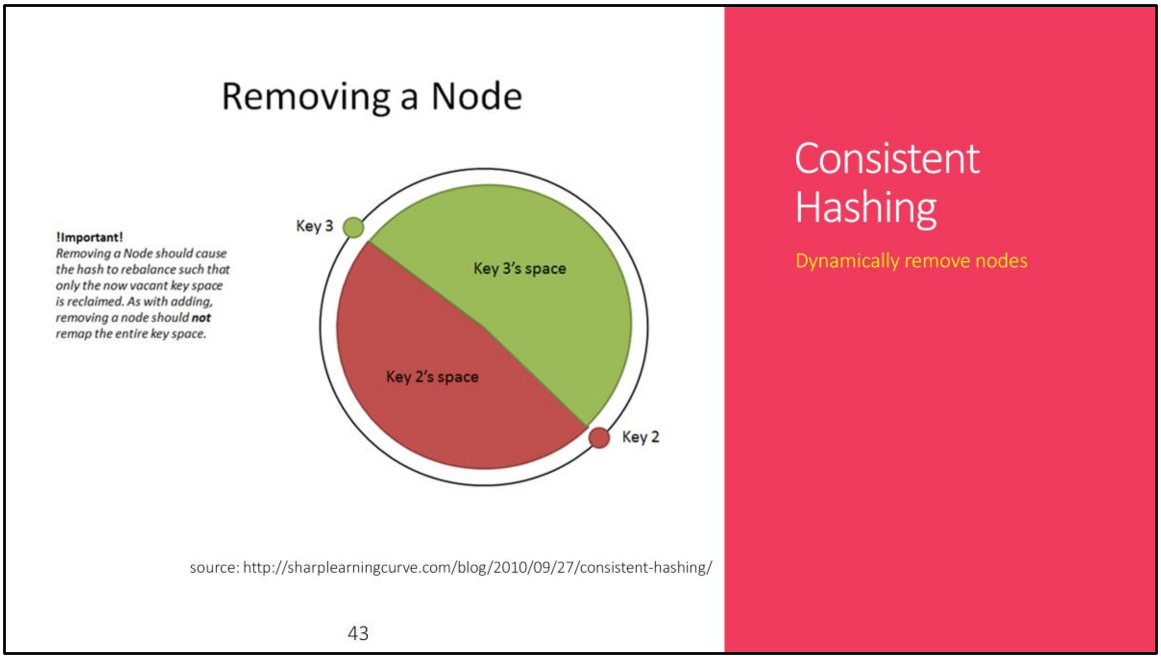
source: http://sharplearningcurve.com/blog/2010/09/27/consistent-hashing/

42

Consistent Hashing

Dynamically add nodes

adding a node requires on an average 1/n+1 nodes to move.



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.*

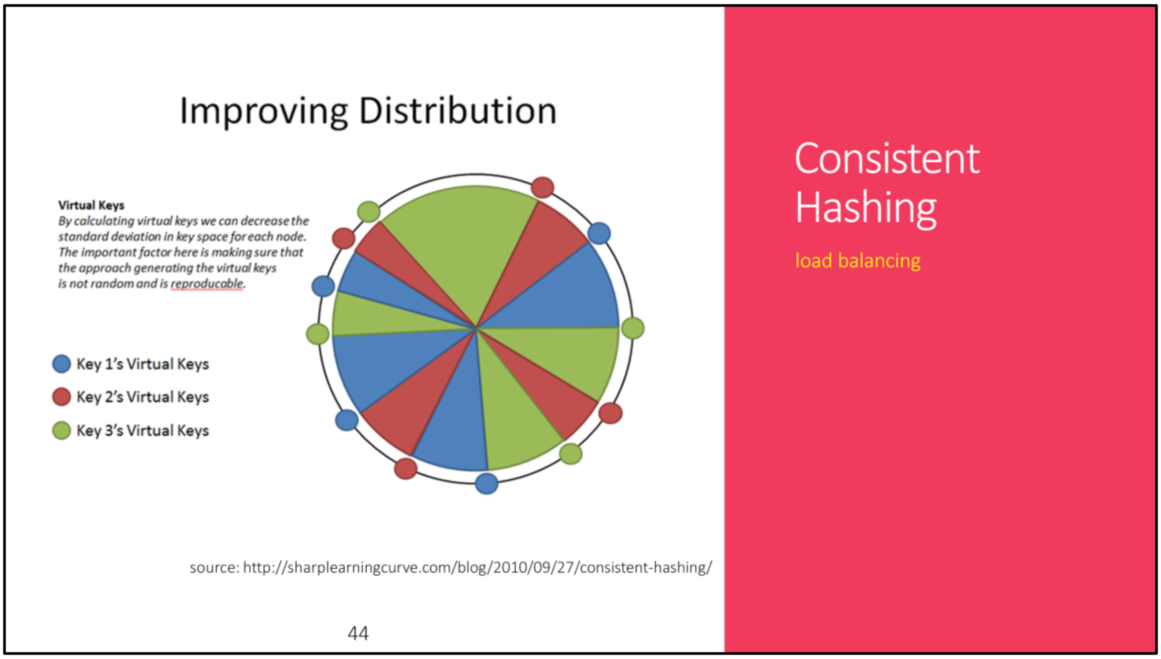
source: http://sharplearningcurve.com/blog/2010/09/27/consistent-hashing/

43

Consistent Hashing

Dynamically remove nodes

Removing a node requires only content of removed node to be shifted.



Improving Distribution

Virtual Keys
By calculating virtual keys we can decrease the standard deviation in key space for each node. The important factor here is making sure that the approach generating the virtual keys is not random and is reproducable.

- ● Key 1's Virtual Keys
- ● Key 2's Virtual Keys
- ● Key 3's Virtual Keys

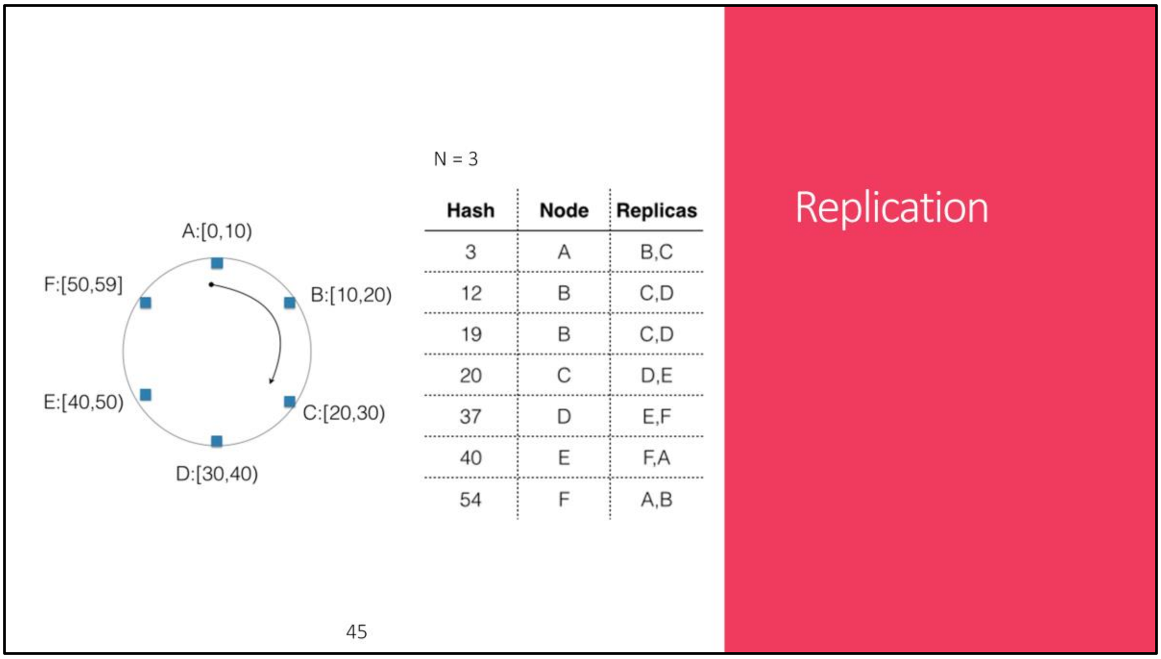
source: http://sharplearningcurve.com/blog/2010/09/27/consistent-hashing/

44

Consistent Hashing

load balancing

Dynamo uses virtual nodes where multiple virtual nodes are assigned to physical nodes. This helps in balancing of load



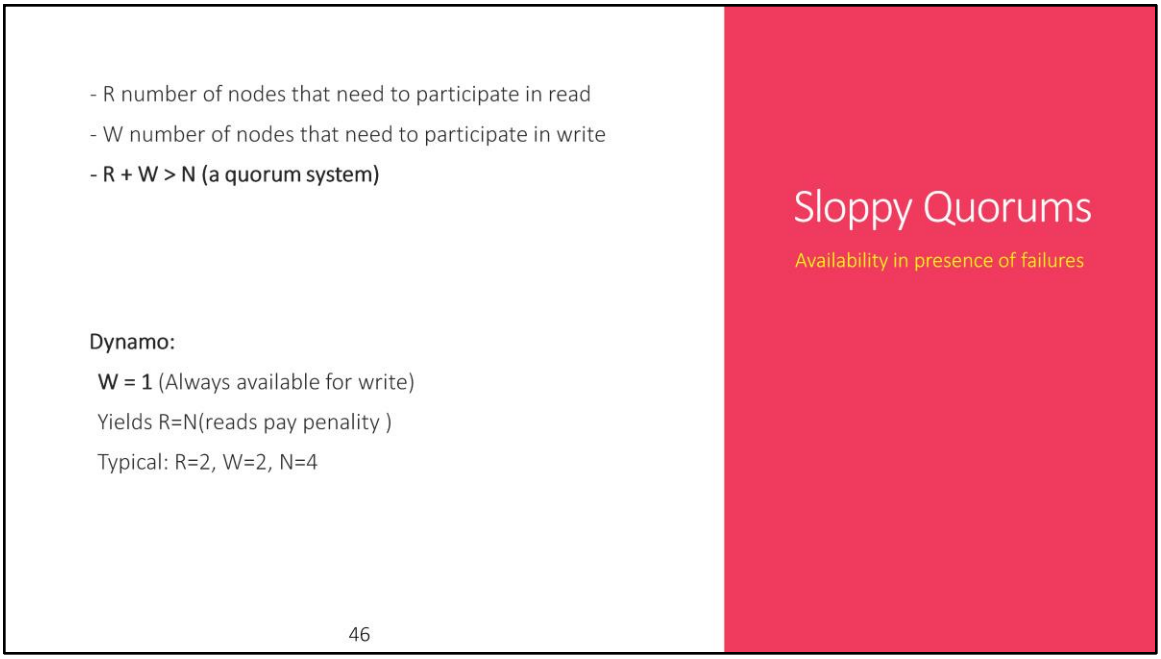
Replication

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

Now we know how to distribute data. Consistent hashing also makes it easier to replicate data. Simply choose next two nodes in the cycle and replicate the data to those nodes.

In the above figure N = 3. So the data is replicated to total 3 nodes. In the given example, if the hash maps to 3, then it lies in the region of A. We put the data in A, now we follow the cycle and replicate the data to two more available nodes.



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:

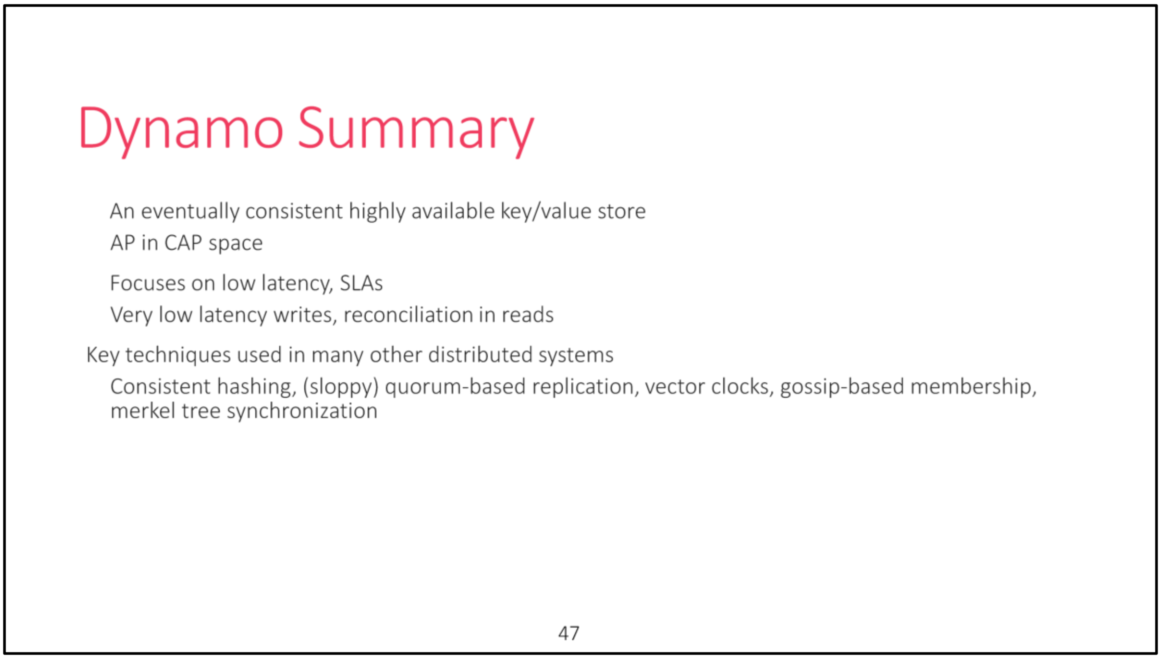
W = 1 (Always available for write)

Yields R=N(reads pay penality)

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

46

"Sloppy quorums" choose the first N healthy nodes. This may lead to inconsistencies. Strict quorum systems become unavailable in case of simplest of failures, so sloppy quorums are used.



Dynamo Summary

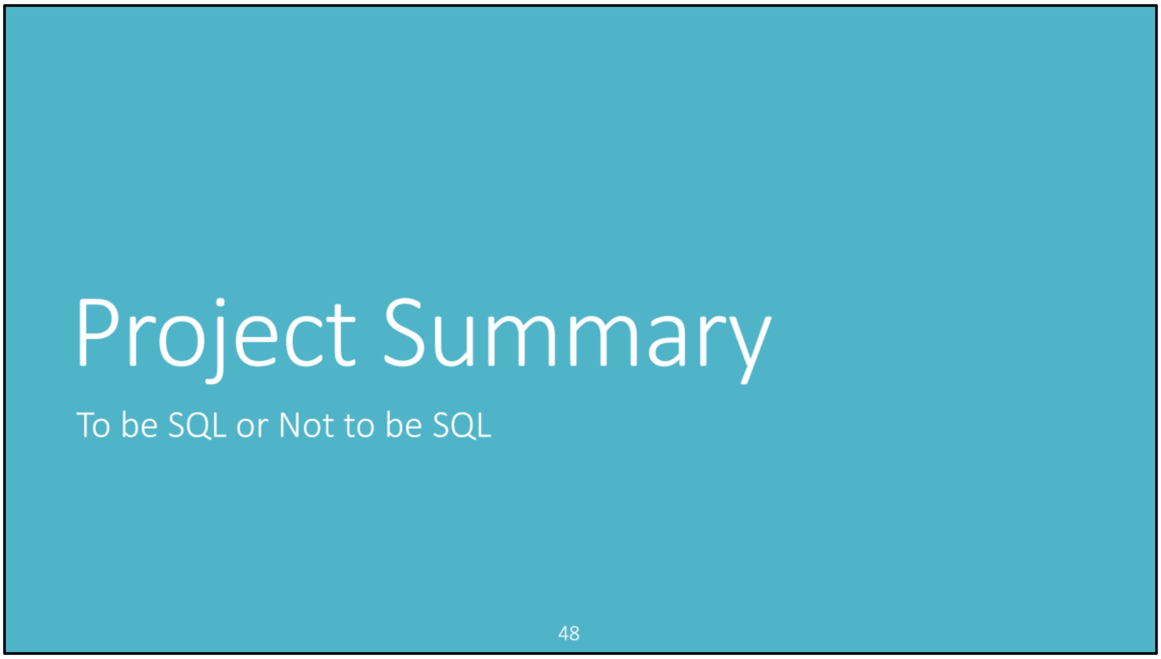
- An eventually consistent highly available key/value store
- AP in CAP space
- Focuses on low latency, SLAs
- Very low latency writes, reconciliation in reads

Key techniques used in many other distributed systems

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

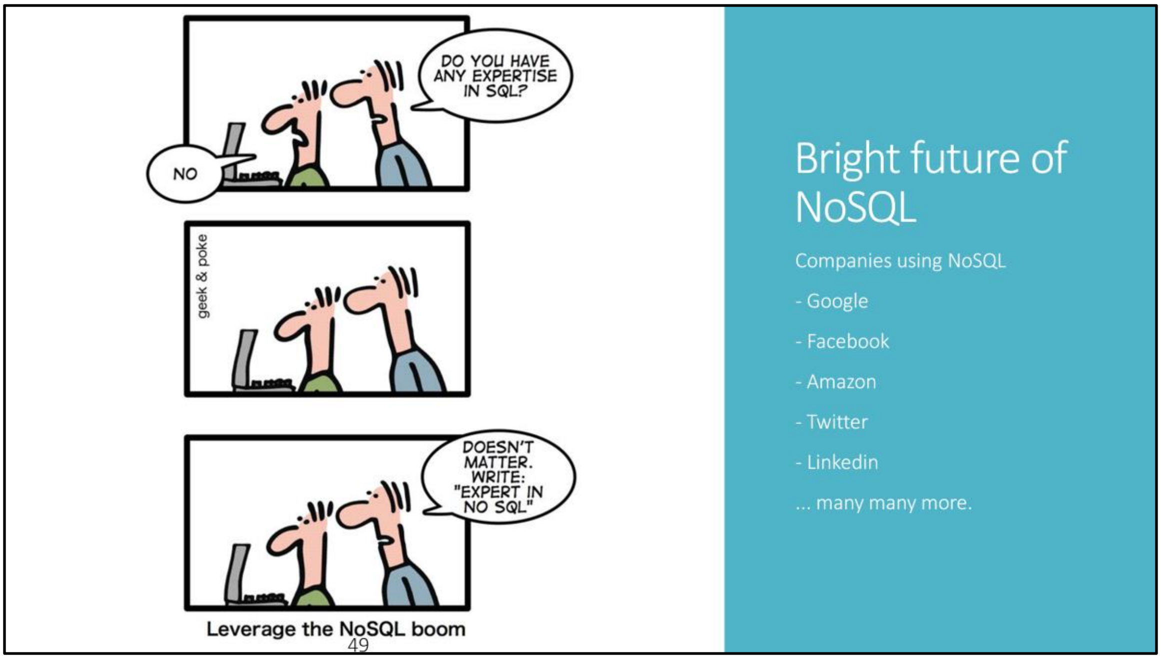
Key ranges because one tree per key range. Merkel tree used for synchronizing replicas.

Each node keep route information to all other nodes. Routing can be done by load balancer or client library.
Using client lib. it directly goes the node in the "preference list", however in case of load balancer - node routes the request to first node in list
Also uses unreliable failure detection to identify failed nodes. Keeps checking in case of partitions also...built into the nodes and not a separate entities.



Project Summary

To be SQL or Not to be SQL


DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom

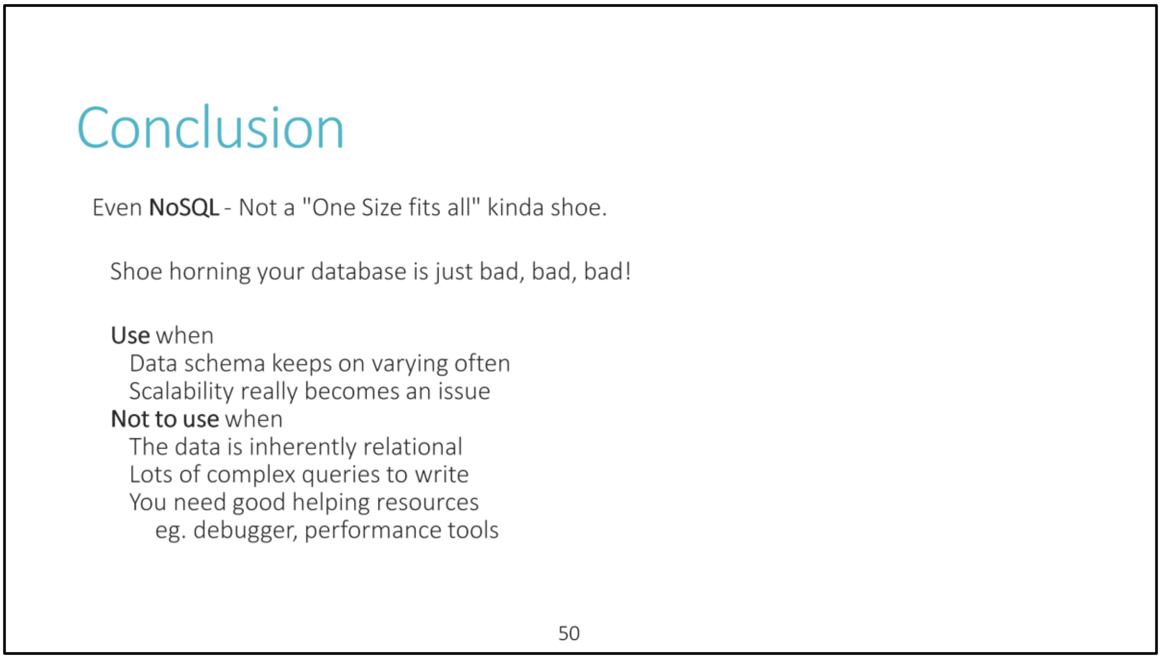
Bright future of NoSQL

Companies using NoSQL

- Google
- Facebook
- Amazon
- Twitter
- Linkedin

... many many more.

Hot topic in tech industry
More and more companies handling a lot of data are adding NoSQL to their workflow



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

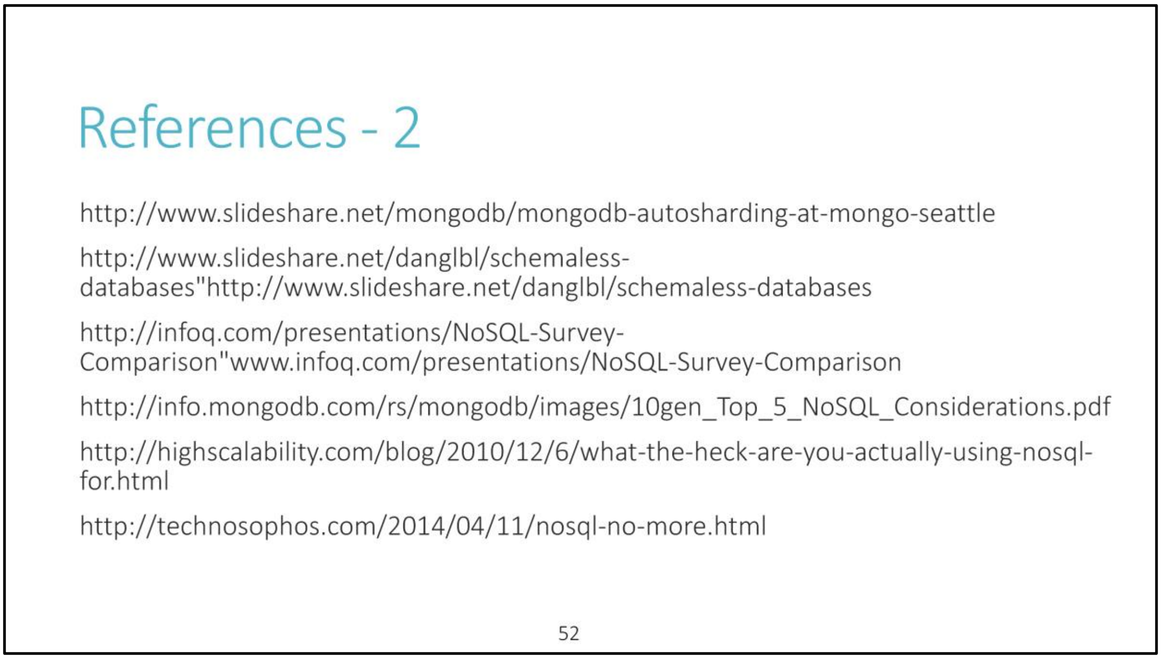
Bigtable: A Distributed Storage System for Structured Data. ACM Trans. Comput. Syst. 26, 2, Article 4 (June 2008). Fay Chang et. al

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

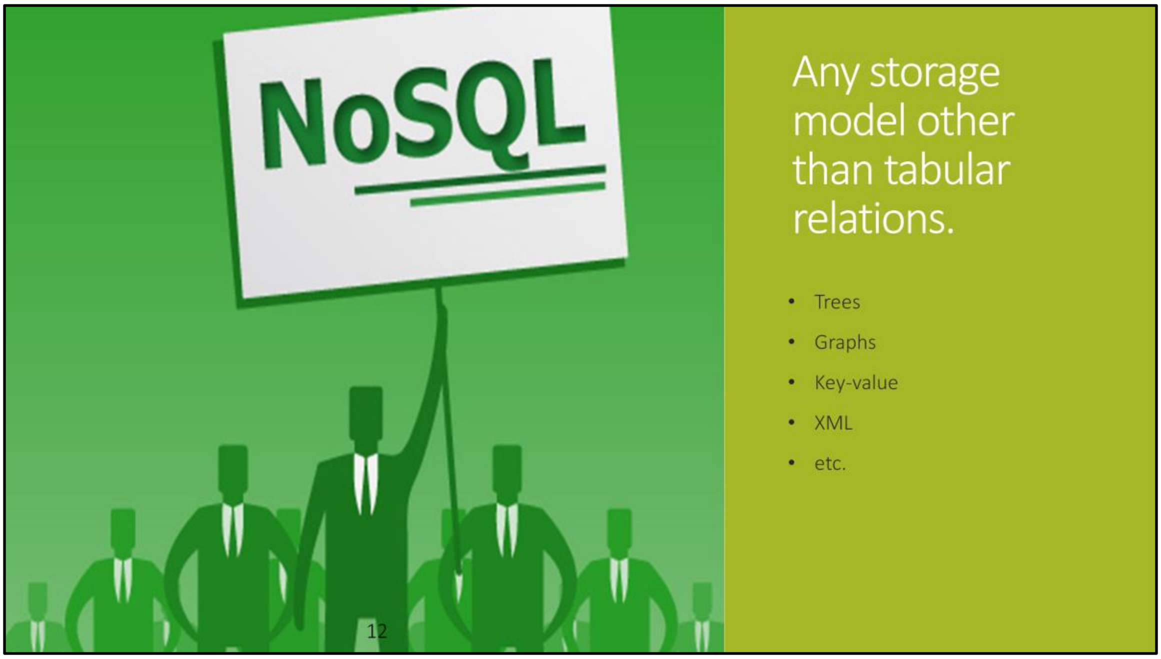
http://highscalability.com/blog/2010/12/6/what-the-heck-are-you-actually-using-nosql-for.html

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.

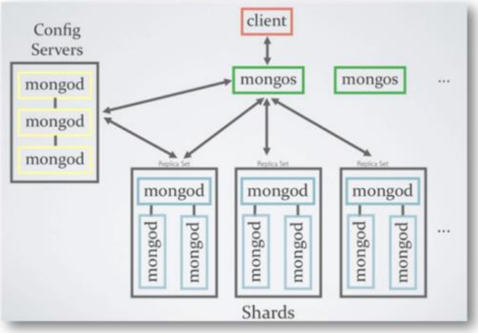
12

1. Social networks are often persisted in the form of trees and graphs.

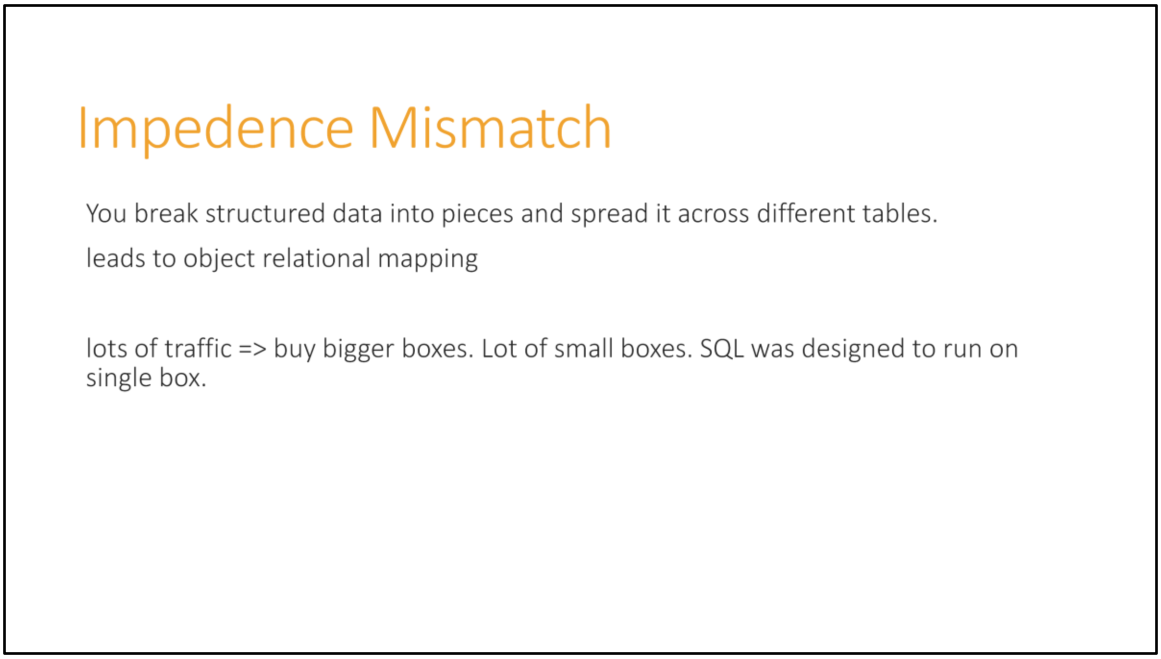
2. Other NoSQL models resemble storing blobs against a key or even a complete XML documents against a key.

3. The main characterstic of these models are that they do not interact with each other unlike relations. Here model can be referred to the data structure used for the data storage in the database. By interacting, we mean that one data structure is independent in itself. It would never need to "join" with other data structure to get any other data.

Auto-sharding



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

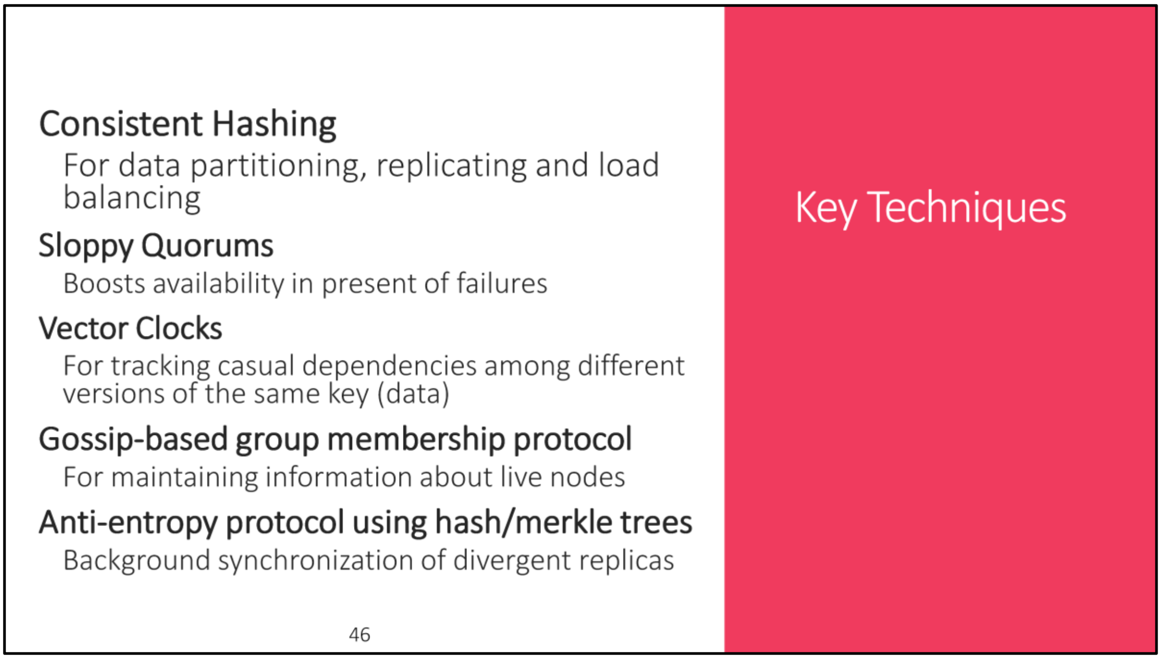


Impedence Mismatch

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

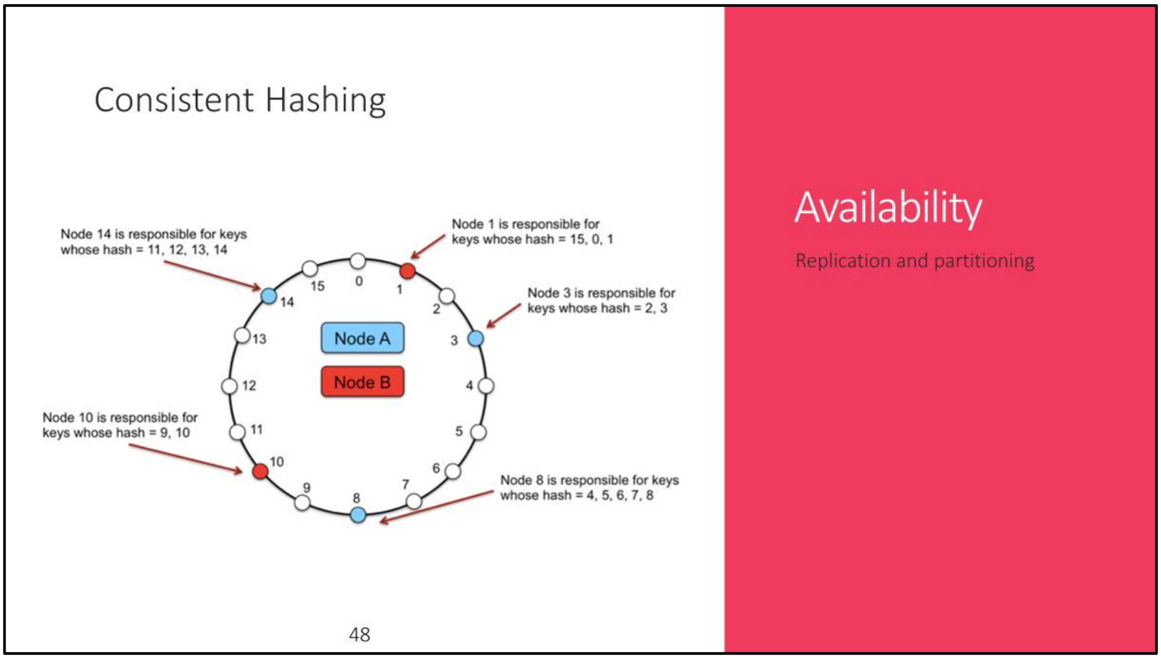
Sloppy Quorums
- Boosts availability in present of failures

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

Key techniques that the dynamo chooses.



Consistent Hashing

Node 14 is responsible for keys whose hash = 11, 12, 13, 14

Node 1 is responsible for keys whose hash = 15, 0, 1

Node 3 is responsible for keys whose hash = 2, 3

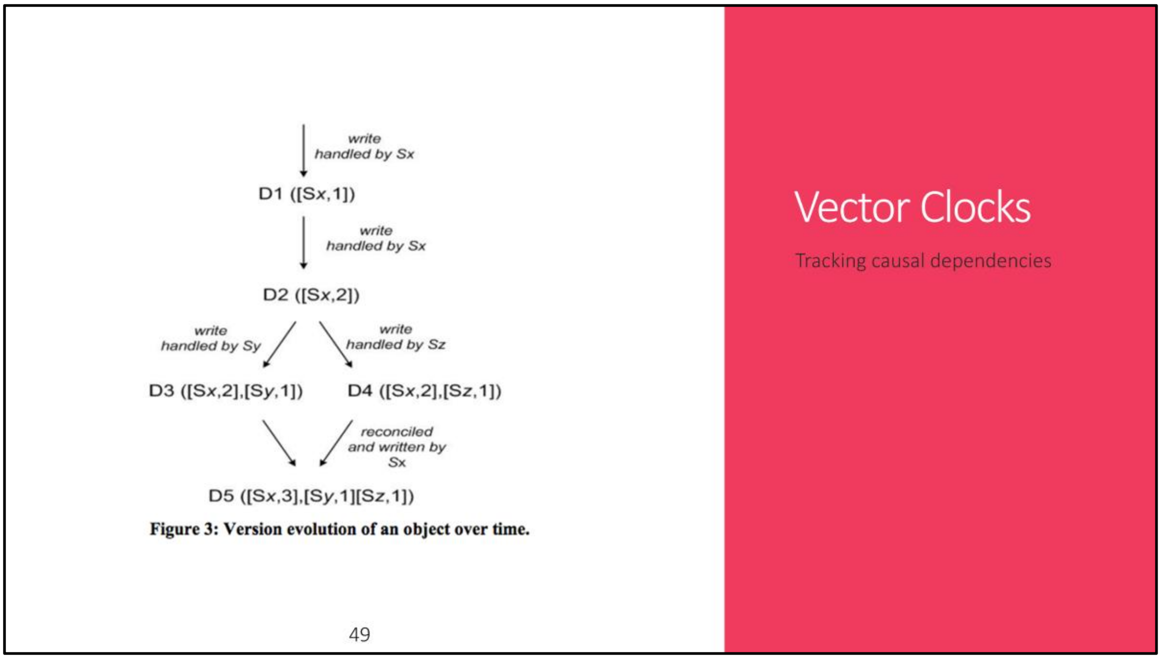
Node 10 is responsible for keys whose hash = 9, 10

Node 8 is responsible for keys whose hash = 4, 5, 6, 7, 8

48

Availability

Replication and partitioning

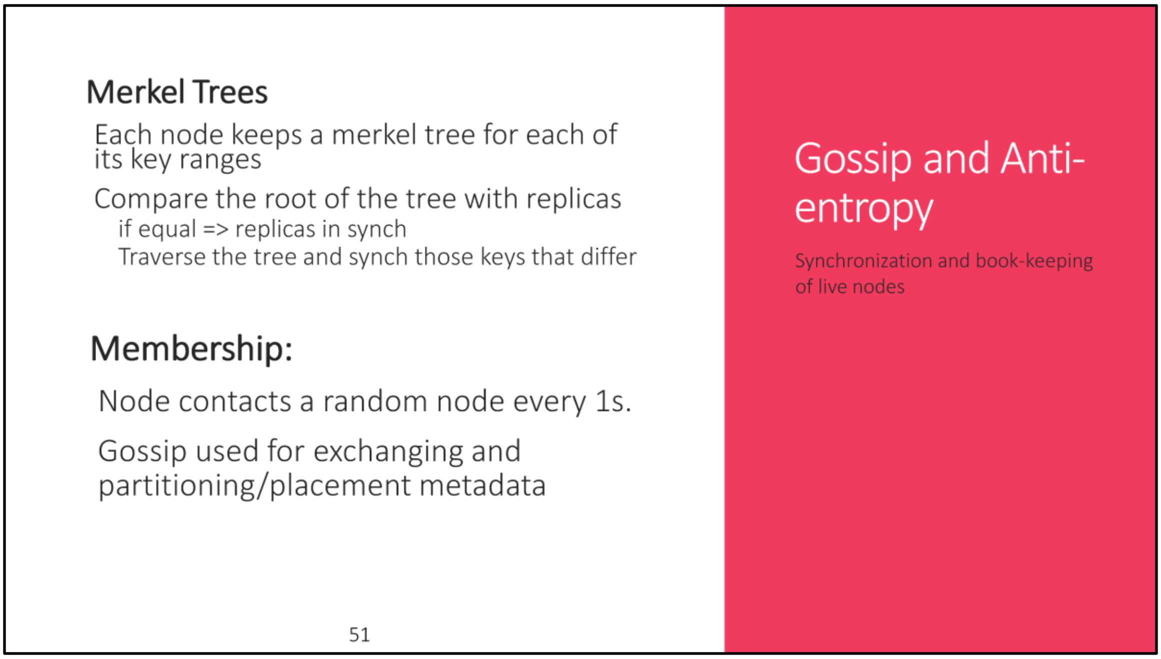


Vector Clocks

Tracking causal dependencies

Figure 3: Version evolution of an object over time.

Each write to a key K is associated with a vector clock VC(K)
Track the version of data.



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

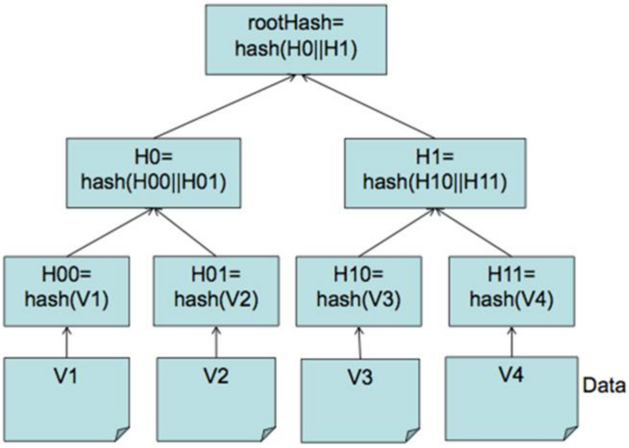
Gossip and Anti-entropy

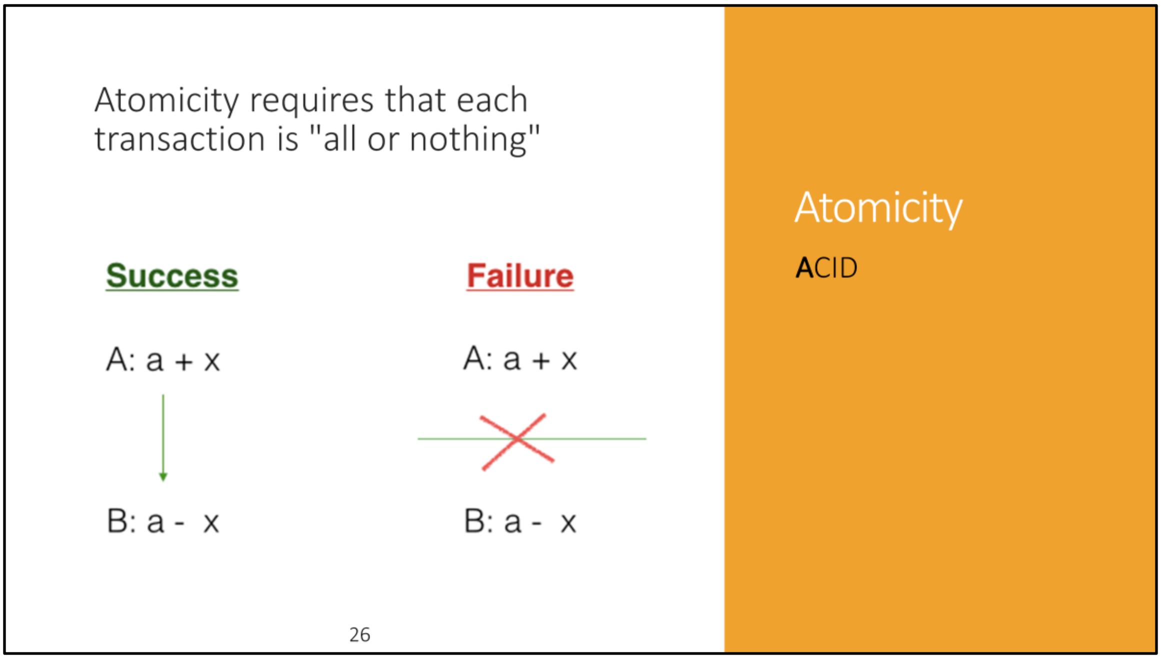
Synchronization and book-keeping of live nodes

Key ranges because one tree per key range. Merkel tree used for synchronizing replicas.

Each node keep route information to all other nodes. Routing can be done by load balancer or client library.
Using client lib. it directly goes the node in the "preference list", however in case of load balancer - node routes the request to first node in list
Also uses unreliable failure detection to identify failed nodes. Keeps checking in case of partitions also...built into the nodes and not a separate entities.

Merkel Trees





Atomicity requires that each transaction is "all or nothing"

Success

A: a + x

B: a - x

Failure

A: a + x

B: a - x

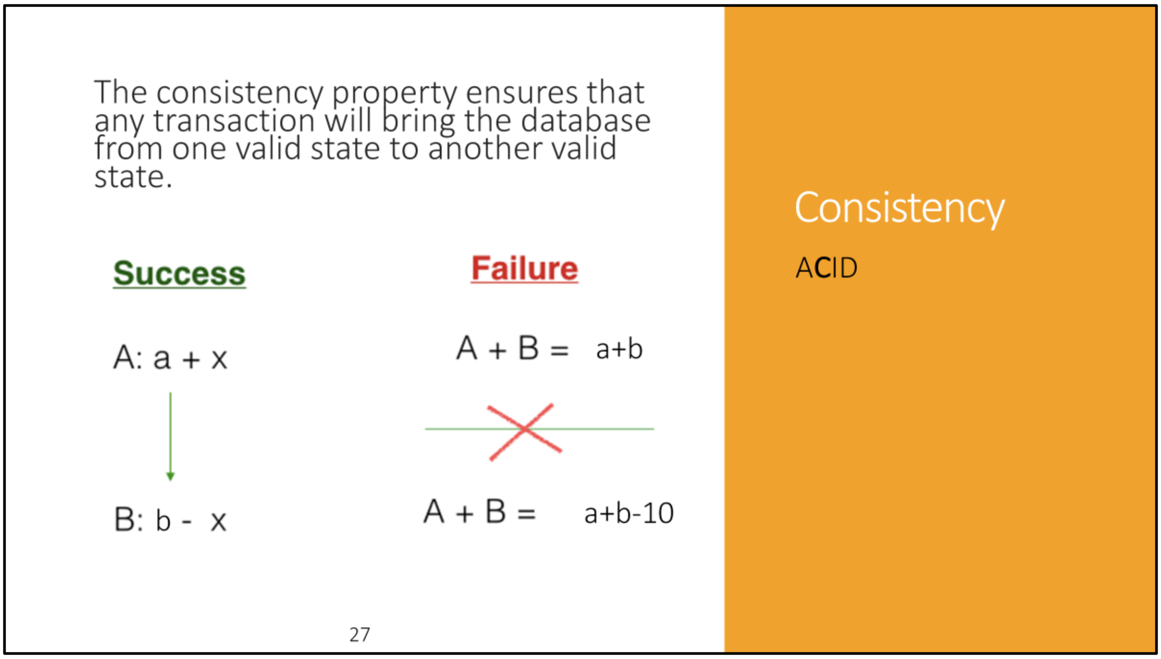
26

Atomicity

ACID

In an **atomic transaction**, a series of database operations either *all* occur, or *nothing* occurs. A guarantee of atomicity prevents updates to the database occurring only partially, which can cause greater problems than rejecting the whole series outright.

Atomicity is said to be fulfilled in the example if either A and B both occur or neither of A or B occurs, i.e. all or none.



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

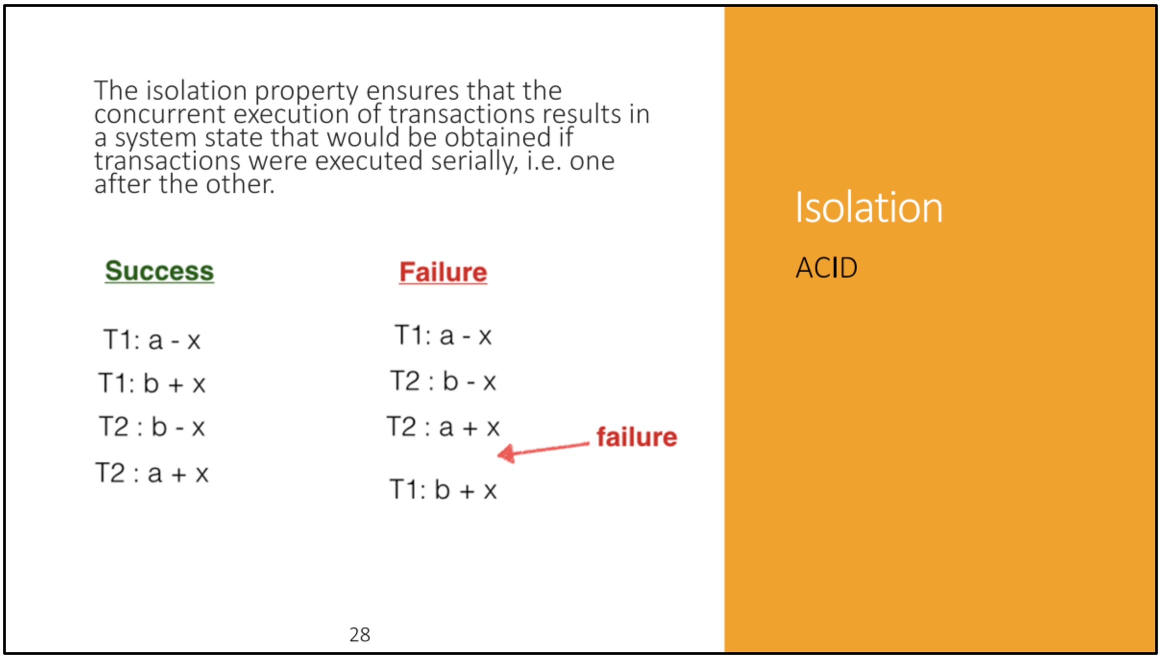
A + B = a+b

A + B = a+b-10

Consistency

ACID

Consistency of the transaction in the above example requires that the total sum of A and B remain constant before and after the transaction. If after transactions, the total sum of A and B becomes a+b-10, then the database is not consistent.



Isolation

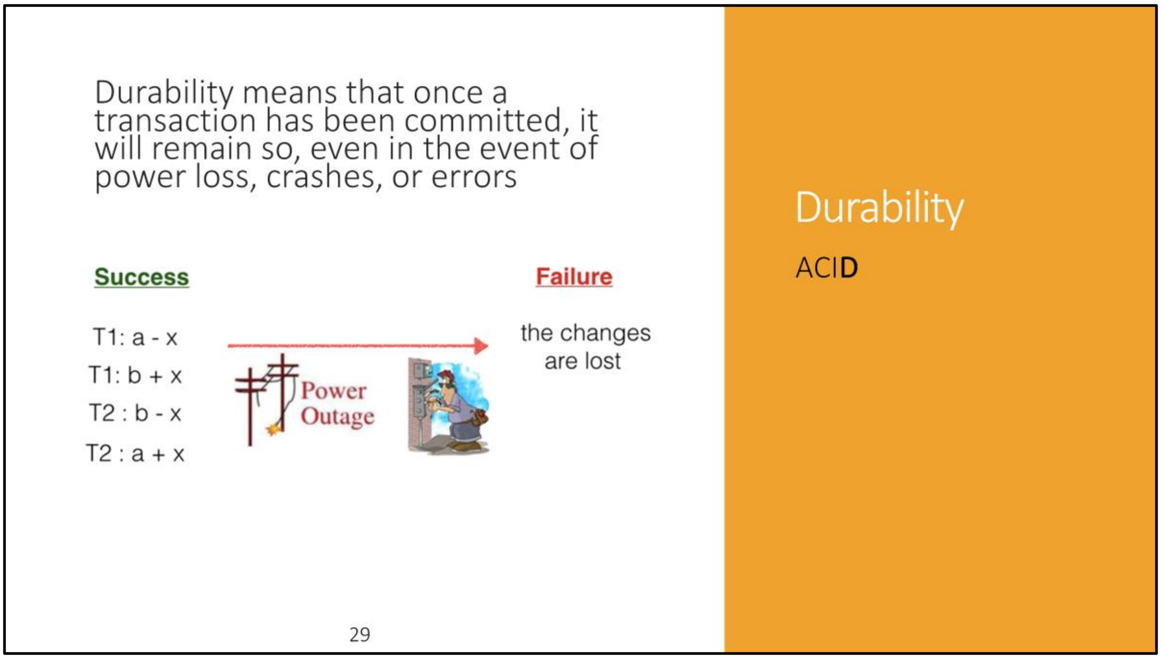
ACID

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	Failure
T1: a - x	T1: a - x
T1: b + x	T2 : b - x
T2 : b - x	T2 : a + x
T2 : a + x	T1: b + x

28

Concurrency control comprises the underlying mechanisms in a DBMS which handles isolation and guarantees related correctness. It is heavily utilized by the database and storage engines both to guarantee the correct execution of concurrent transactions. (All discussed in detail in the class)



Power
Outage

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

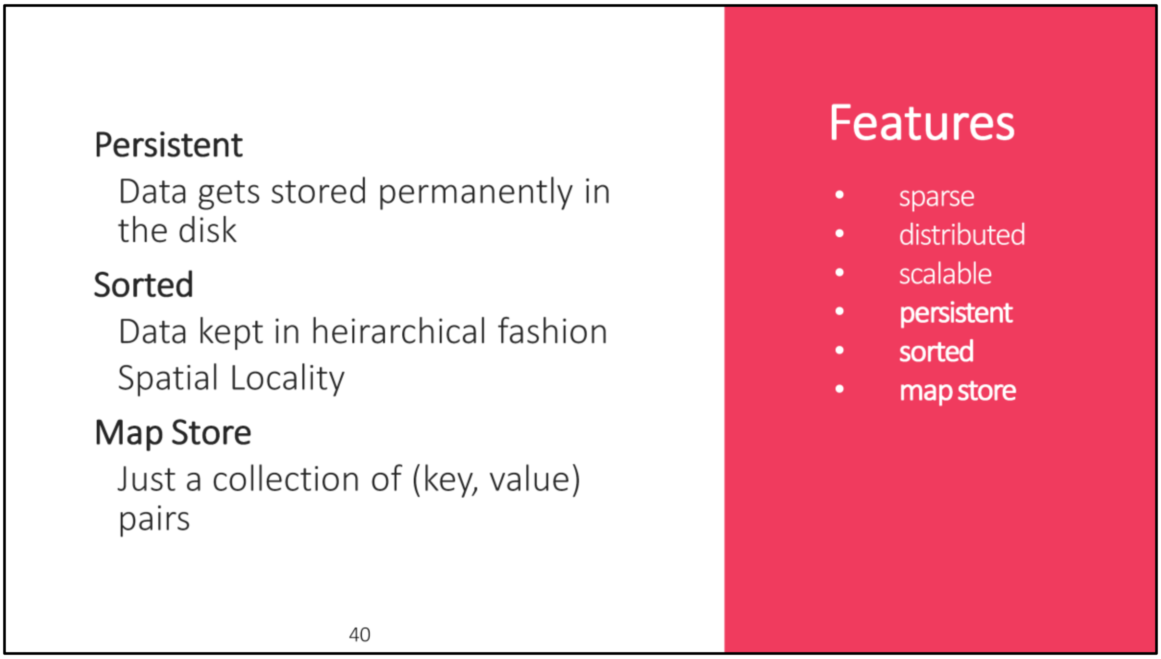
T2 : a + x

Failure

the changes are lost

29

Durability is the ACID property which guarantees that transactions that have committed will survive permanently. For example, if a flight booking reports that a seat has successfully been booked, then the seat will remain booked even if the system crashes.



Features

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

Persistent

Data gets stored permanently in the disk

Sorted

Data kept in heirarchical fashion

Spatial Locality

Map Store

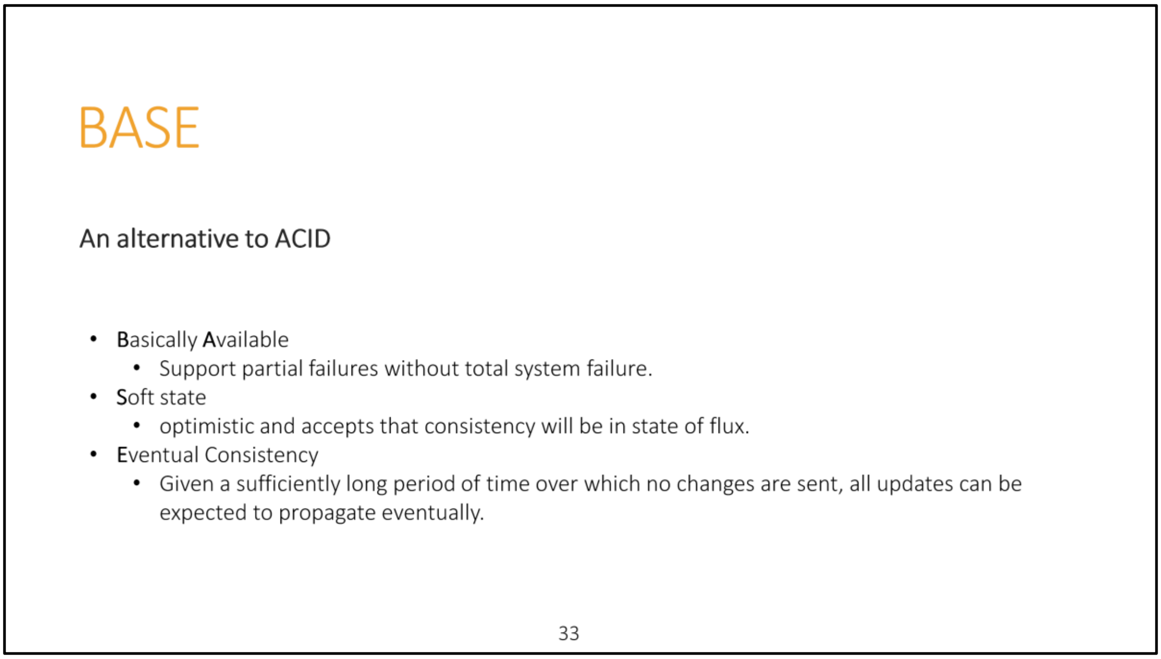
Just a collection of (key, value) pairs

40

Sorted

A key is hashed to a position in a table. BigTable sorts its data by keys. This helps keep related data close together, usually on the same machine — assuming that one structures keys in such a way that sorting brings the data together. For example, if domain names are used as keys in a BigTable, it makes sense to store them in reverse order to ensure that related domains are close together.

map A **map** is an associative array; a data structure that allows one to look up a value to a corresponding key quickly. BigTable is a collection of (key, value) pairs where the key identifies a row and the value is the set of columns.



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.

According to CAP you can pick only two of the alternatives.

BASE focuses on Availability and Partition tolerance whereas ACID focuses on Consistency and Availability.