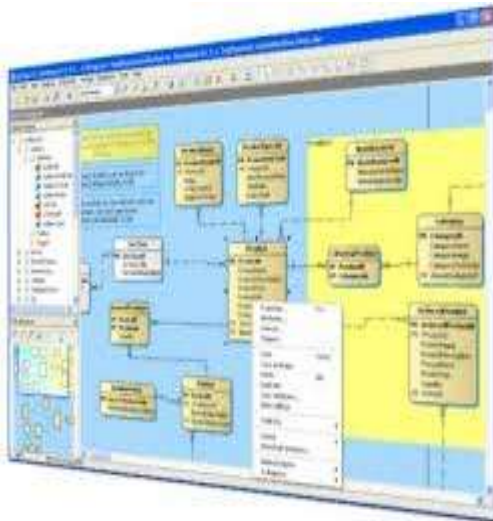


A NO-SQL Databases Overview

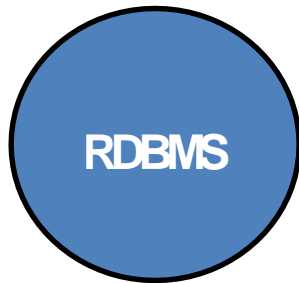
BY: Eli Leiba



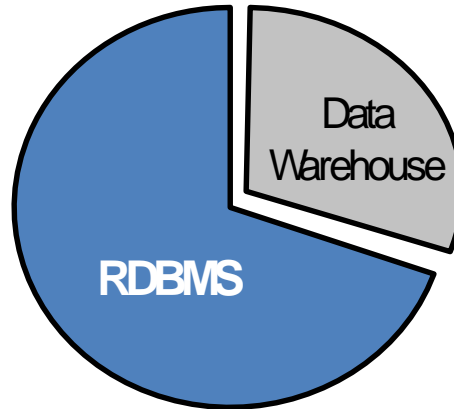
NOSQL DATABASES

1. What are the new database "architecture patterns" introduced by the NoSQL movement?
2. What types of problems do they address?
3. How do you match the right problem with the right database pattern?

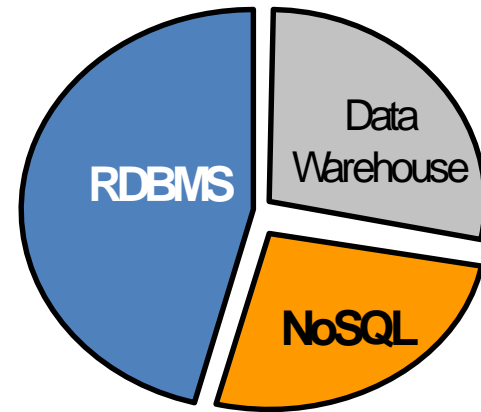
Three Eras of Databases



1985-1995



1995-2010



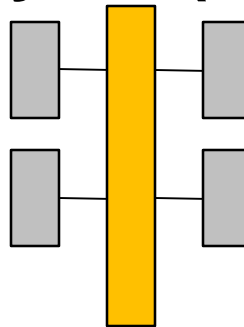
2010-Now

- RDBMS for transactions, Data Warehouse for analytics and NoSQL for ...?

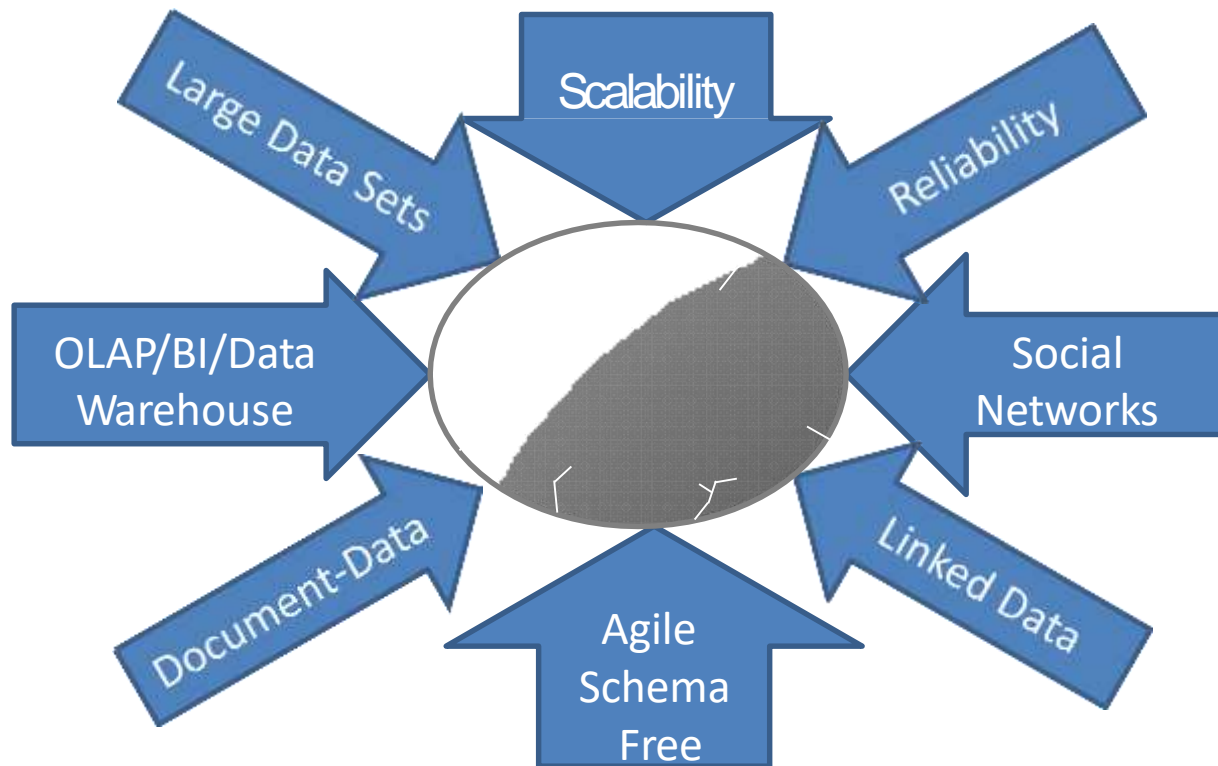
Before NoSQL

Relational

Analytical (OLAP)

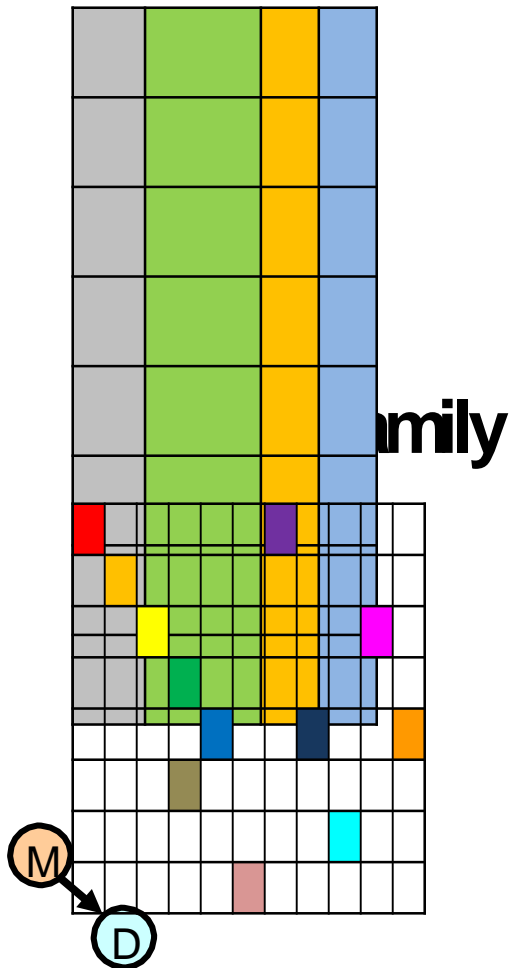


Pressures on Single Node RDBMS Architectures

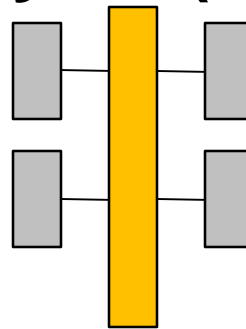


After NoSQL

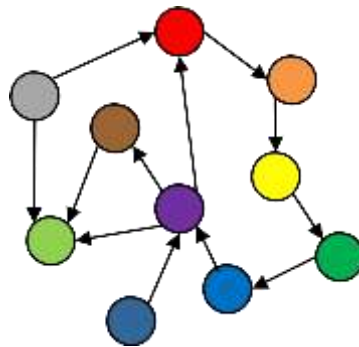
Relational



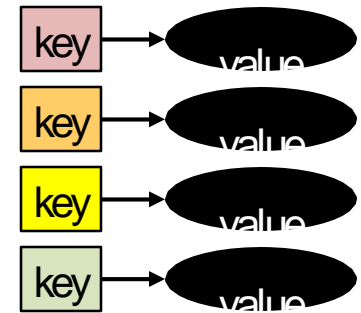
Analytical (OLAP)



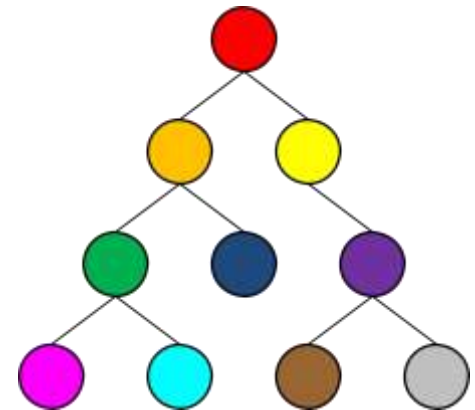
Graph



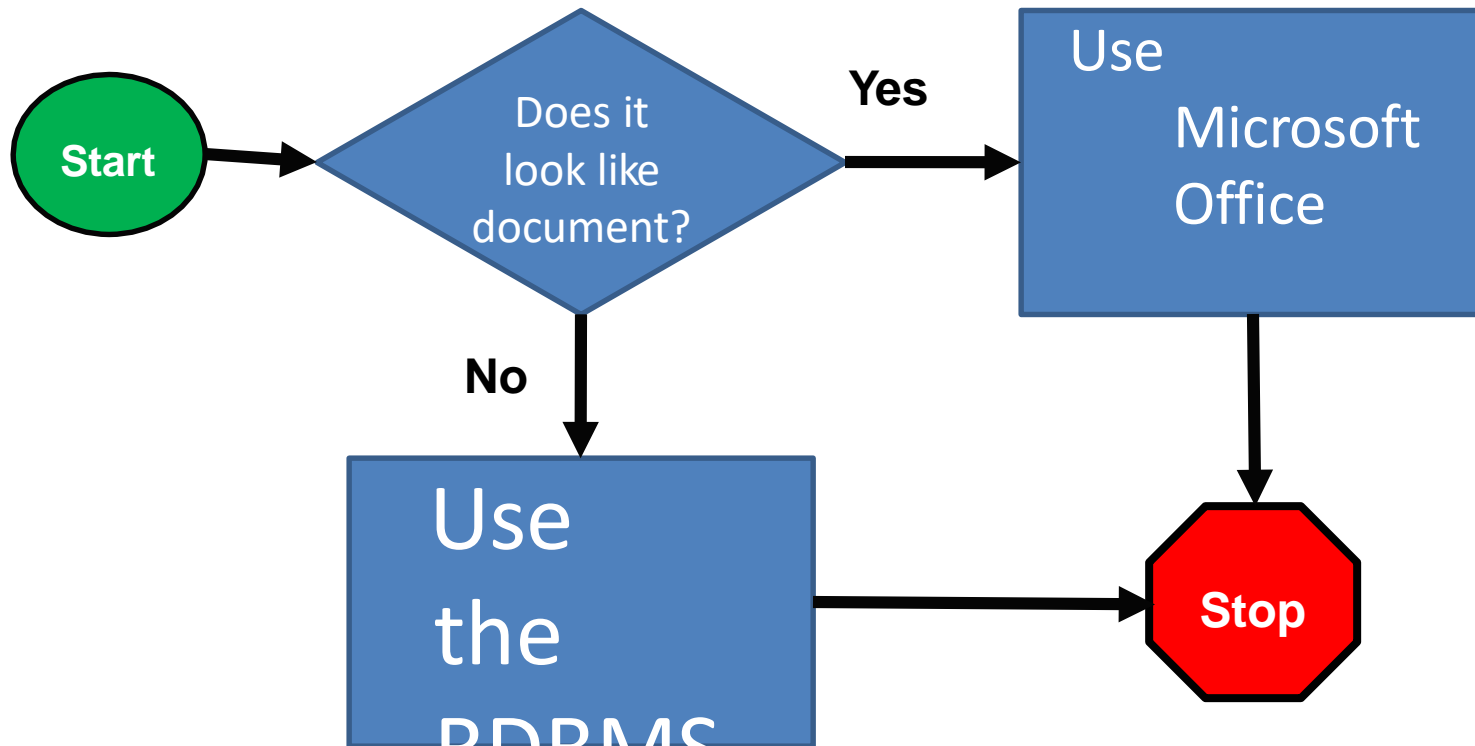
Key-Value



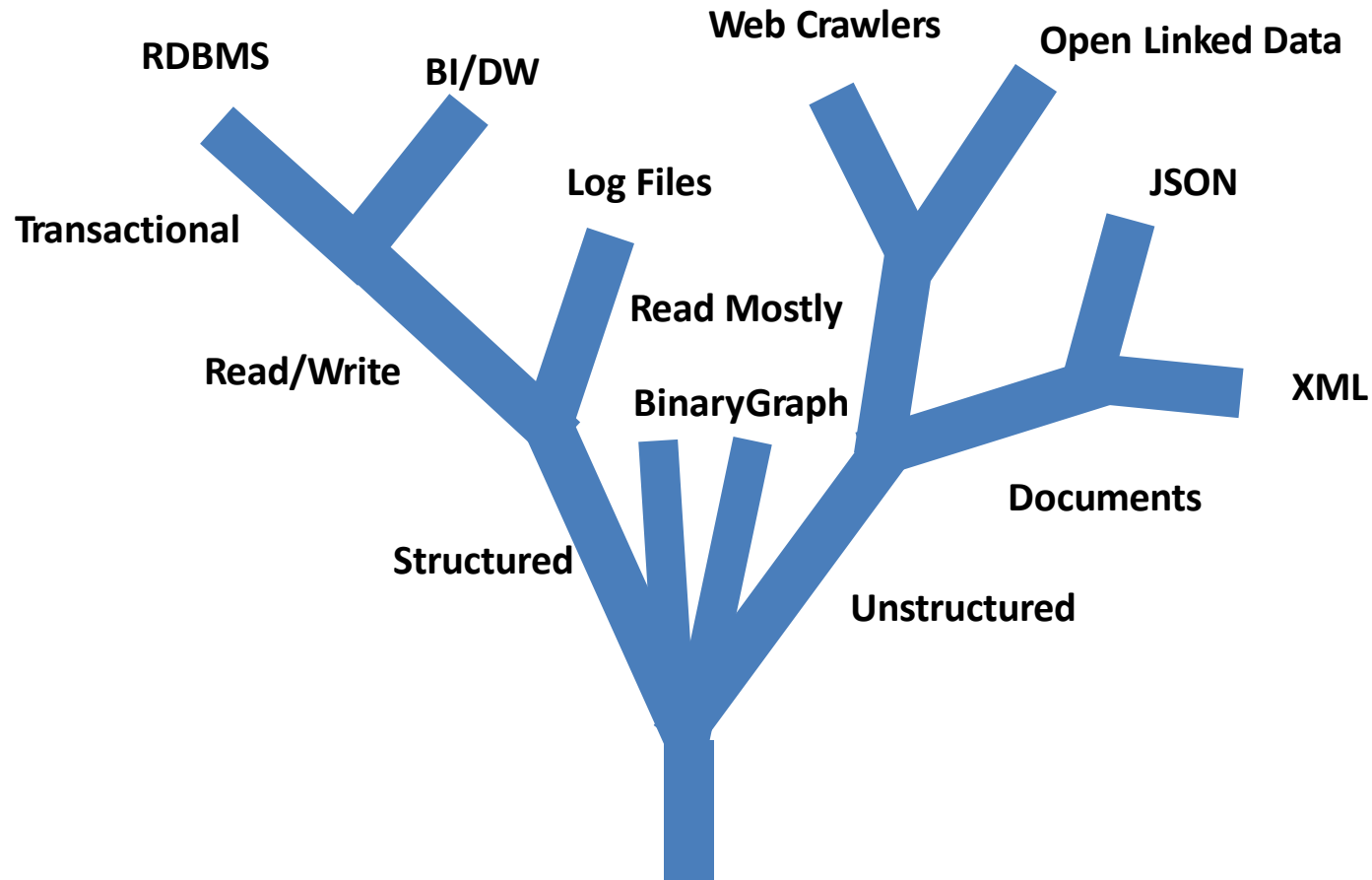
Document



Before NoSQL DB Selection Was Easy!



An evolving tree of data types



Many Uses of Data

- Transactions (OLTP)
- Analysis (OLAP)
- Search and Findability
- Enterprise Agility
- Discovery and Insight
- Speed and Reliability
- Consistency and Availability

Simplicity is a Virtue



Photo from flickr by PSNZ Images

- Many modern systems derive their strength by dramatically limiting the features in their system and focus on a specific task
- Simplicity allows database designer to focus on the primary business drivers

Simplicity is a Design Style

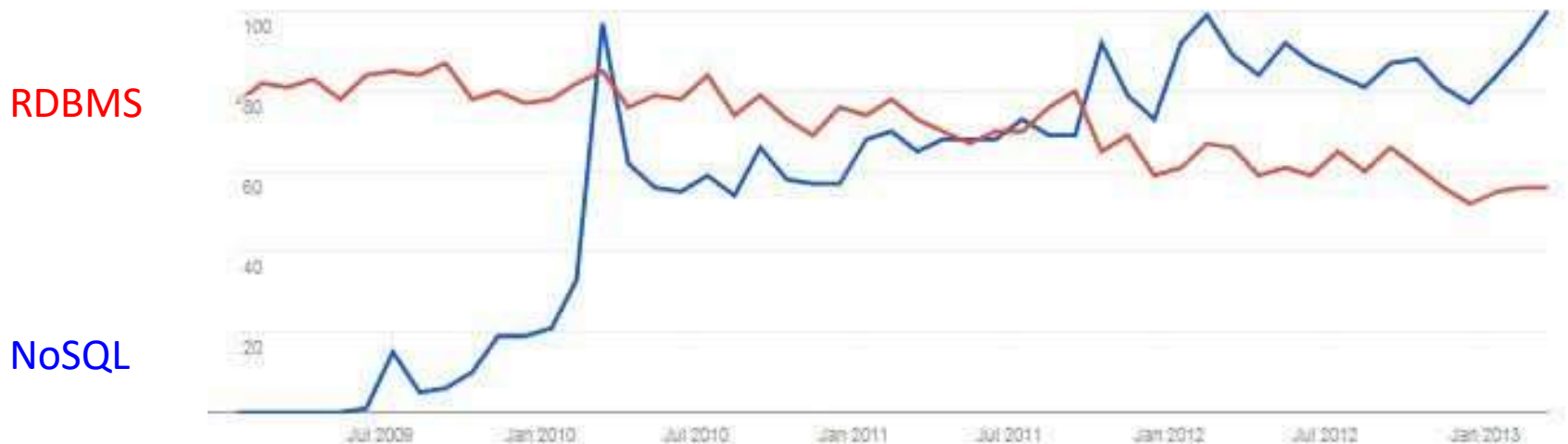


- Focus only on simple systems that solve many problems in a flexible way
- Examples:
 - Touch screen interfaces
 - Key/Value data stores

RDBMS vs. NoSQL

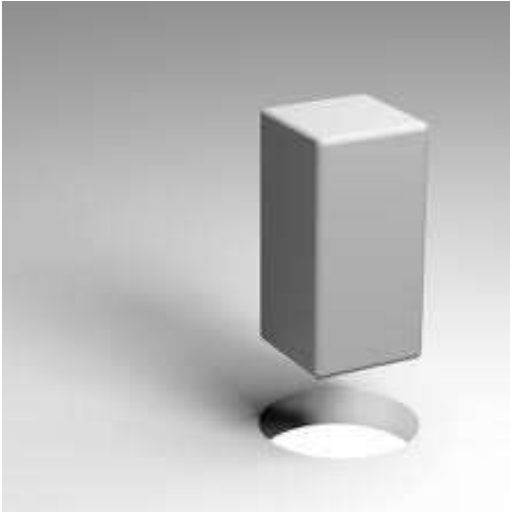
- NoSQL is real and it's here to stay

Google Trends



<http://www.google.com/trends/explore#q=nosql%2C%20rdbms&date=1%2F2009%2051m&cmpt=q>

Eric Evans



“The whole point of seeking alternatives [to RDBMS systems] is that you need to solve a problem that relational databases are a bad fit for.”

Eric Evans
Rackspace

The NO-SQL Universe

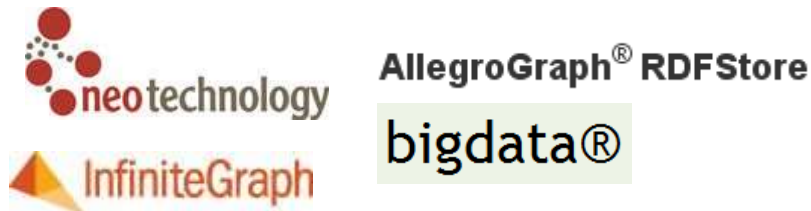
Key-Value Stores



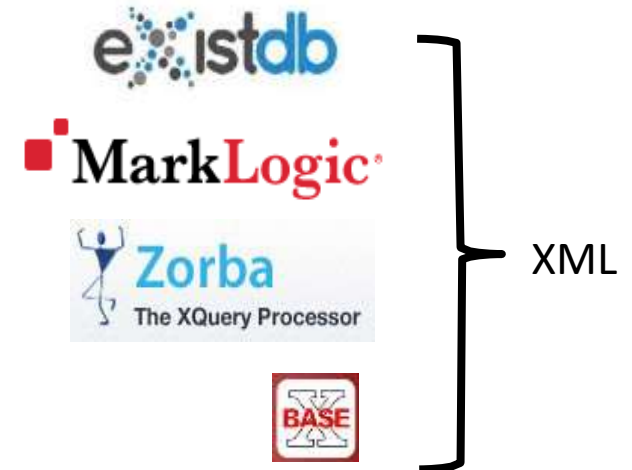
Document Stores



Graph/Triple Stores



Column-Family Stores



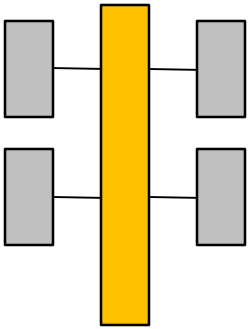
Relational

- Data is usually stored in row by row manner (row store)
- Standardized query language (SQL)
- Data model defined **before** you add data
- Joins merge data from multiple tables
- Results are tables
- **Pros:** mature ACID transactions with fine-grain security controls
- **Cons:** Requires up front data modeling, does not scale well

Examples:

Oracle, MySQL,
PostgreSQL,
Microsoft SQL
Server, IBM DB/2

Analytical (OLAP)

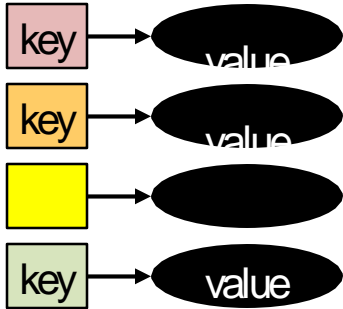


Examples:

Cognos, Hyperion,
Microstrategy,
Pentaho, Microsoft,
Oracle, Business
Objects

- Based on "Star" schema with central fact table for each event
- Optimized for analysis of read-analysis of historical data
- Use of MDX language to count query "measures" for "categories" of data
- **Pros:** fast queries for large data
- **Cons:** not optimized for transactions and updates

Key-Value Stores



- Keys used to access opaque blobs of data
- Values can contain any type of data (images, video)

Pros: scalable, simple API
(put, get, delete)

Cons: no way to query based on the content of the value

Examples:

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

Key Value Stores

Key	Value



Blob datatype

string datatype

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

The Locker Metaphor




Key-Values Stores are Like Dictionaries

The "key" is just the word "gouge"

The "value" is all the definitions and images

gouge | gouj |
noun
1 a chisel with a concave blade, used in carpentry, sculpture, and surgery.
2 an indentation or groove made by gouging.

verb [trans.]
1 make (a groove, hole, or indentation) with or as if with a gouge : *the channel had been **gouged out** by the ebbing water.*
• make a rough hole or indentation in (a surface), esp. so as to mar or disfigure it : *he had wielded the blade inexpertly, gouging the grass in several places.*
• (**gouge something out**) cut or force something out roughly or brutally : *one of his eyes had been gouged out.*
2 informal overcharge; swindle : *the airline ends up gouging the very passengers it is supposed to assist.*

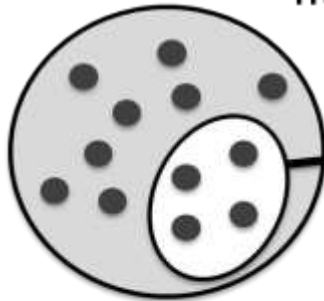


gouge 1

The diagram shows a dictionary entry for the word "gouge". A curved arrow points from the text "The 'key' is just the word 'gouge'" to the word "gouge" in the entry. Another curved arrow points from the text "The 'value' is all the definitions and images" to the entire entry, including the definitions and the illustration of the gouge.

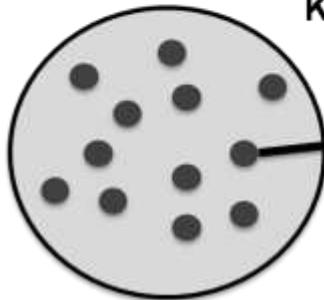
No Subset Queries in Key-Value Stores

Traditional Relational Model



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

Key-Value Store Model



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

Types of Key-Value Stores

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

Memcached



- Open source in-memory key-value caching system
- Make effective use of RAM on many distributed web servers
- Designed to speed up dynamic web applications by alleviating database load
- RAM resident key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering
- Simple interface for highly distributed RAM caches
- 30ms read times typical
- Designed for quick deployment, ease of development APIs in many languages



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



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



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

Column-Family

Red					Purple				
	Yellow								
		Yellow							Pink
			Green						
				Blue		Dark Blue			Orange
			Brown						
							Cyan		
				Pink					

Examples:

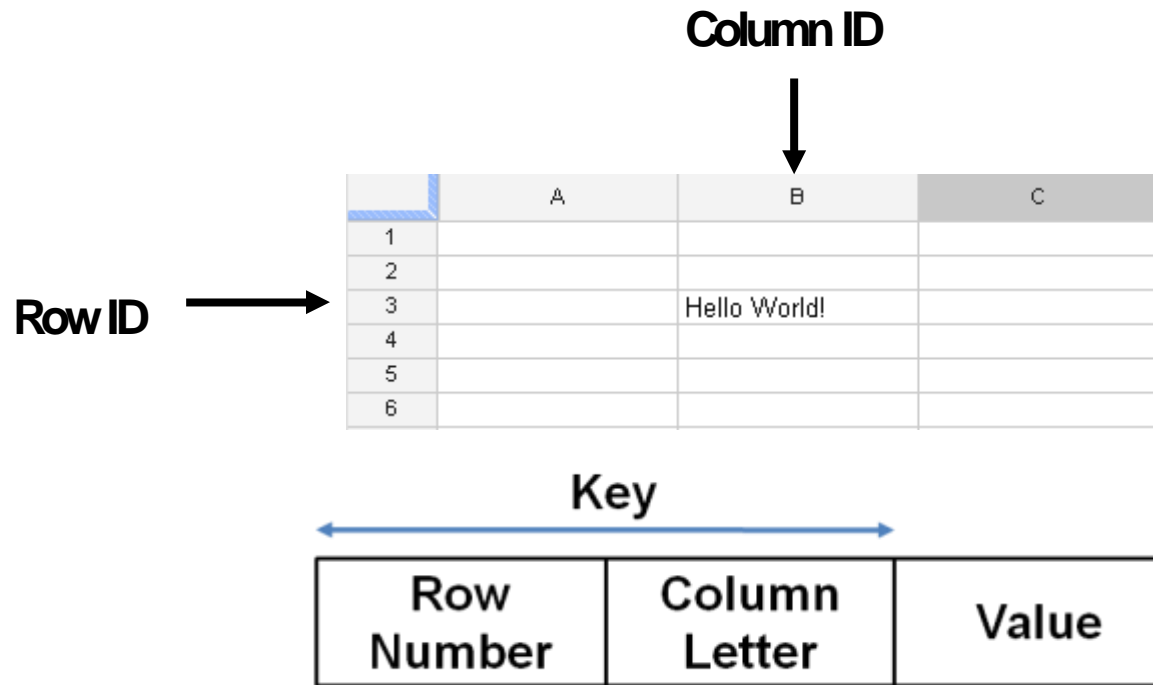
Cassandra, HBase,
Hypertable, Apache
Accumulo, Bigtable

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

Column Family (Bigtable)

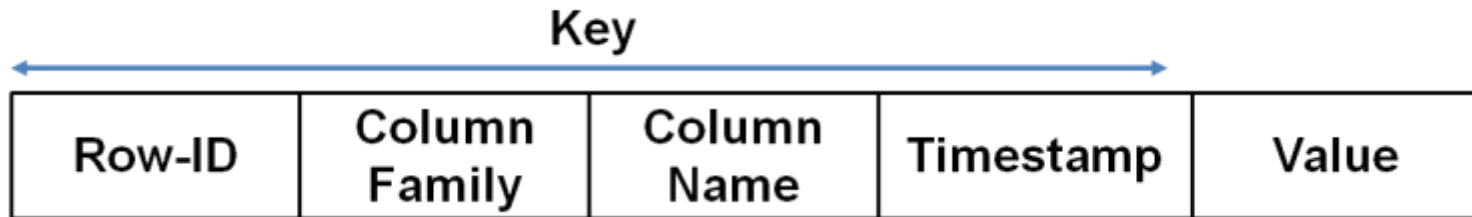
- The champion of "Big Data"
- Excel at highly saleable systems
- Tightly coupled with MapReduce
- Technically a "sparse matrix" where most cells have no data
- Generating a list of all columns is non-trivial
- Examples:
 - Google Bigtable
 - Hadoop HBase
 - Hypertable

Spreadsheets Use a Row/Column as a Key



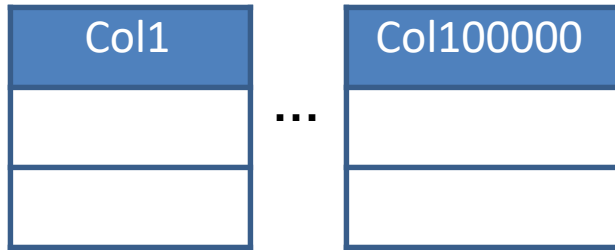
- Bigtable systems use a combination of row and column information as part of their key

Keys Include Family and Timestamps



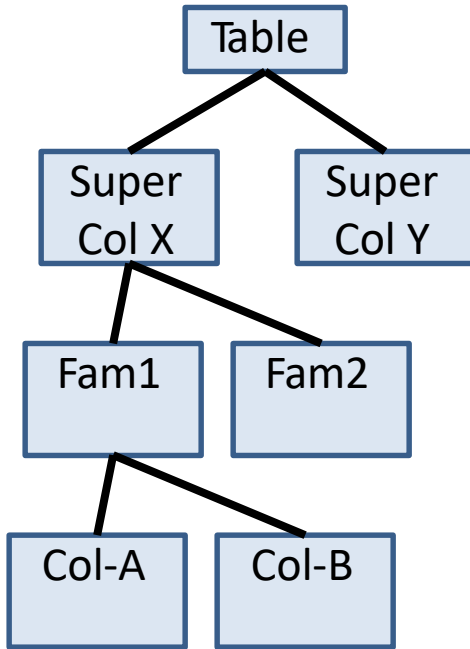
- Bigtable systems have keys that include not just row and column ID but other attributes
- Column Families are created when a table is created
- Timestamps allows multiple versions of values
- Values are just ordered bytes and have no strongly typed data system

Column Store Concepts



- Preserve the table-structure familiar to RDBMS systems
- Not optimized for "joins"
- One row could have millions of columns but the data can be very "sparse"
- Ideal for high-variability data sets
- Column families allow to query all columns that have a specific property or properties
- Allow new columns to be inserted without doing an "alter table"
- Trigger new columns on inserts

Column Families



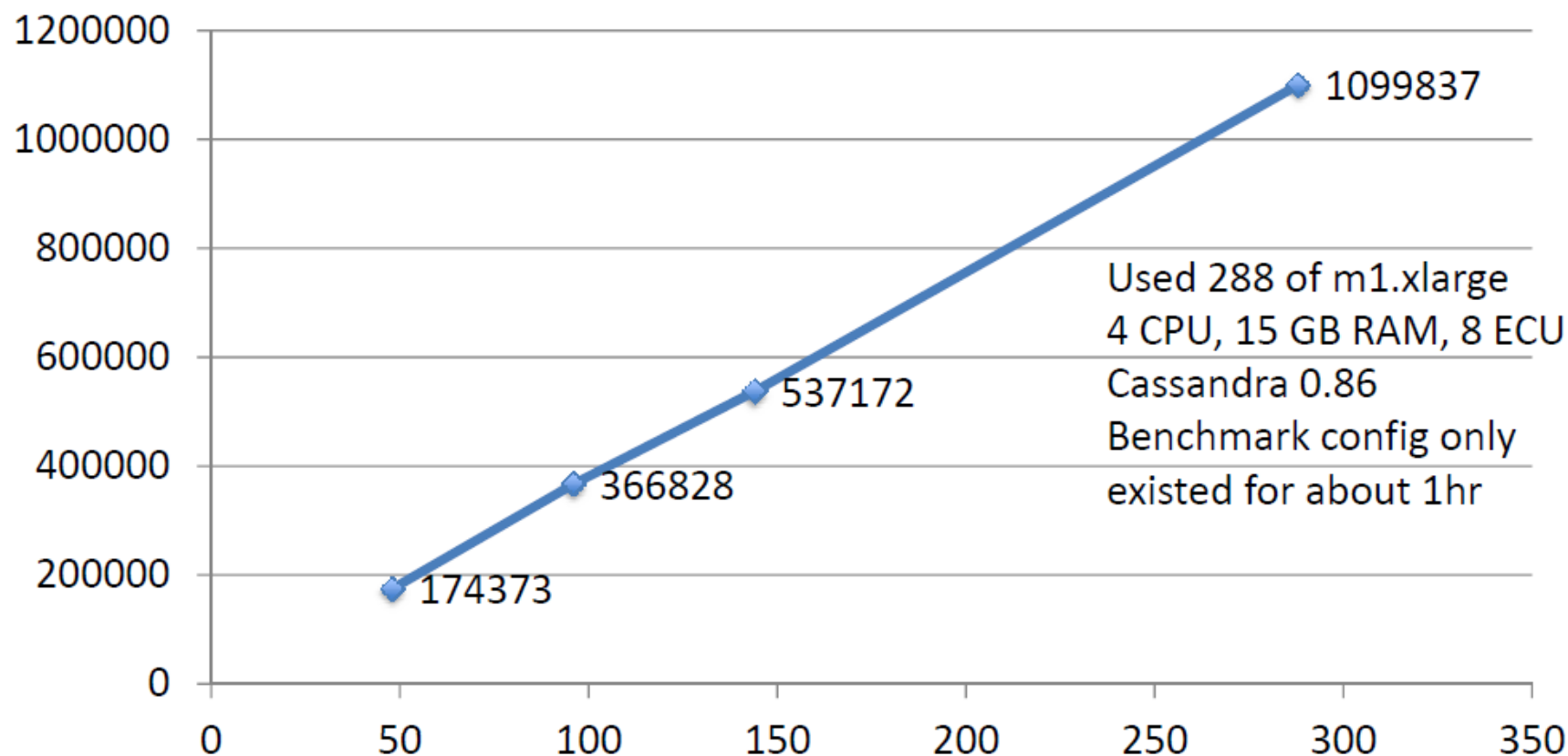
- Group columns into "Column families"
- Group column families into "Super-Columns"
- Be able to query all columns with a family or super family
- Similar data grouped together to improve speed



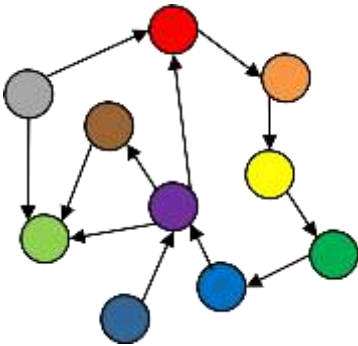
- Apache open source column family database supported by DataStax
- Peer-to-peer distribution model
- Strong reputation for linear scale out (millions of writes/second)
- Database side security
- Written in Java and works well with HDFS and MapReduce



Client Writes/s by node count – Replication Factor = 3



Graph Store



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

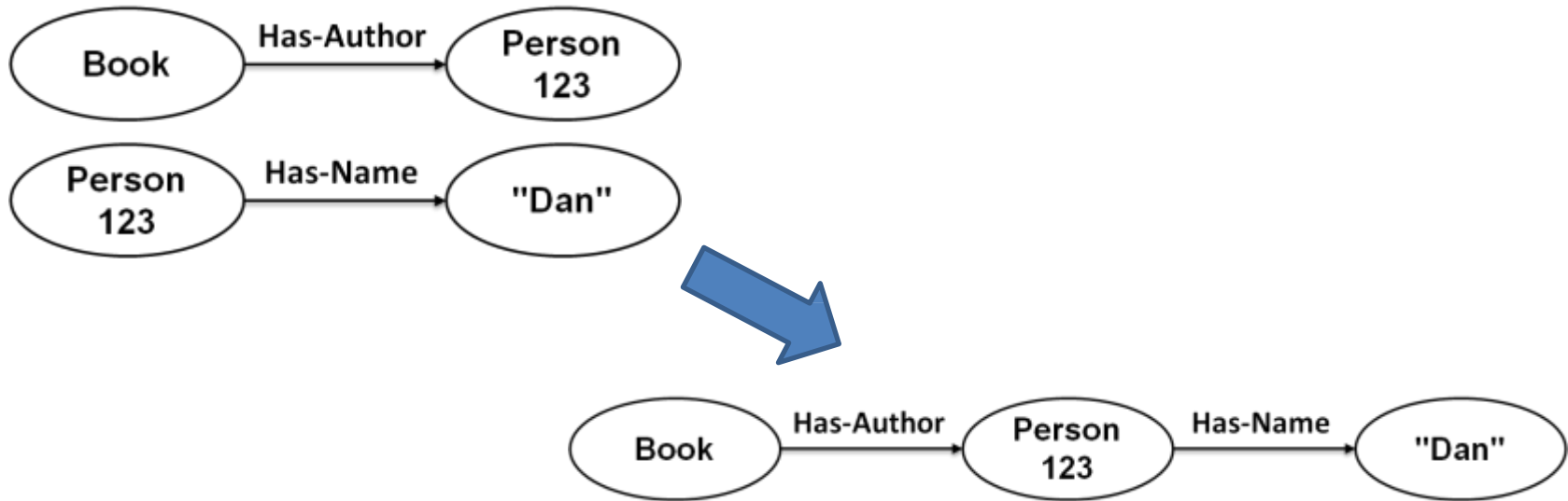
Examples:

Neo4j, AllegroGraph,
Bigdata triple store,
InfiniteGraph,
StarDog

Graph Stores

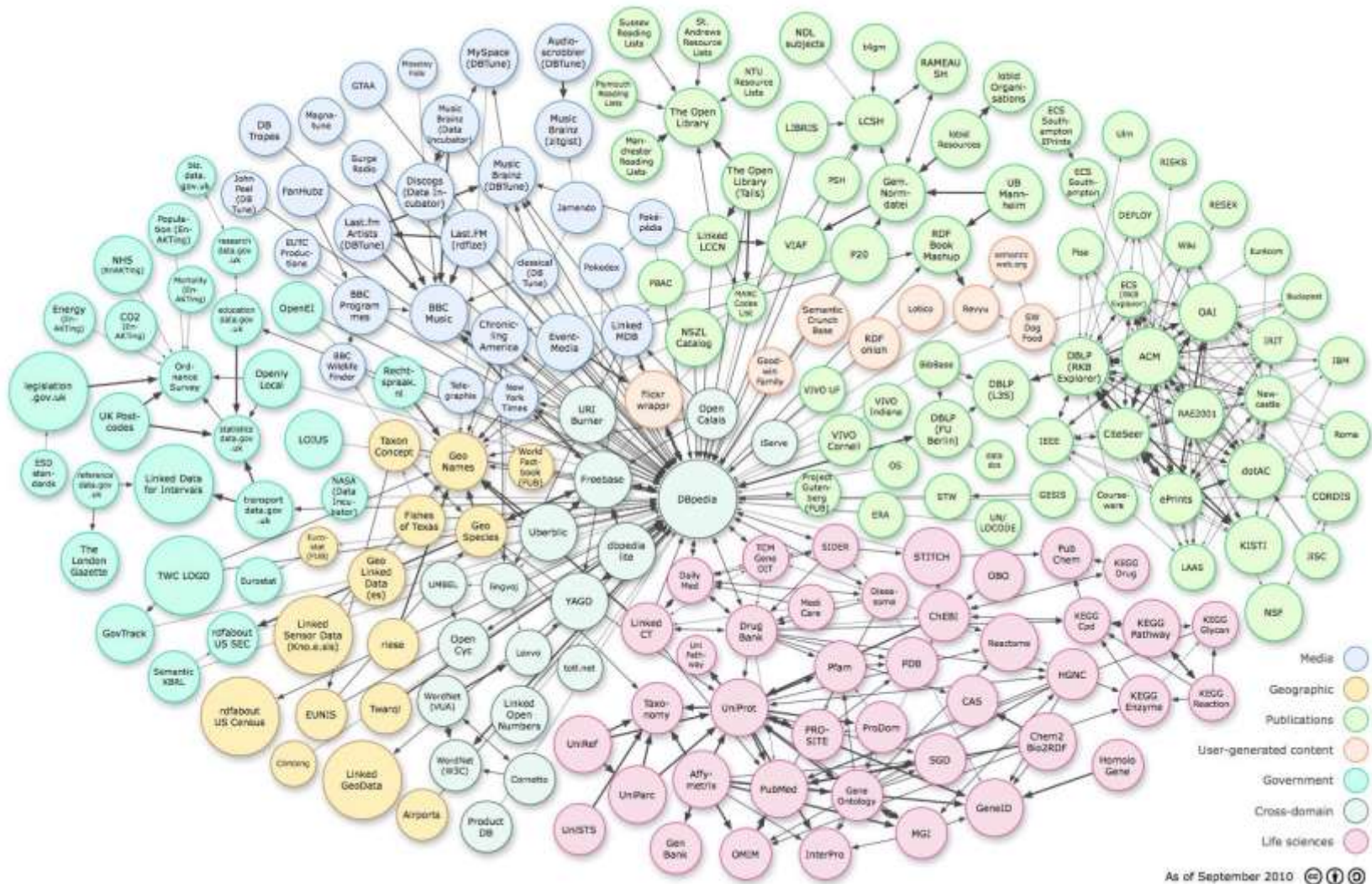
- Used when the relationship and relationships types between items are critical
- Used for
 - Social networking queries: "friends of my friends"
 - Inference and rules engines
 - Pattern recognition
 - Used for working with open-linked data
- Automate "joins" of public data

Nodes are "joined" to create graphs



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

Open Linked Data

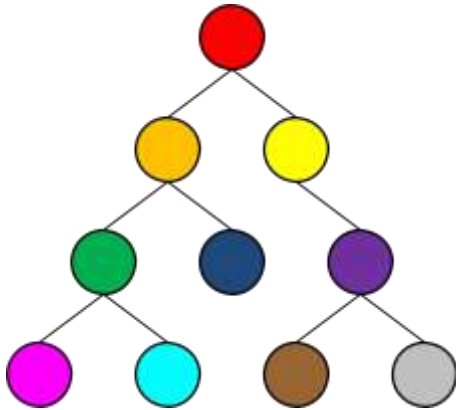


Neo4J



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

Document Store



Examples:

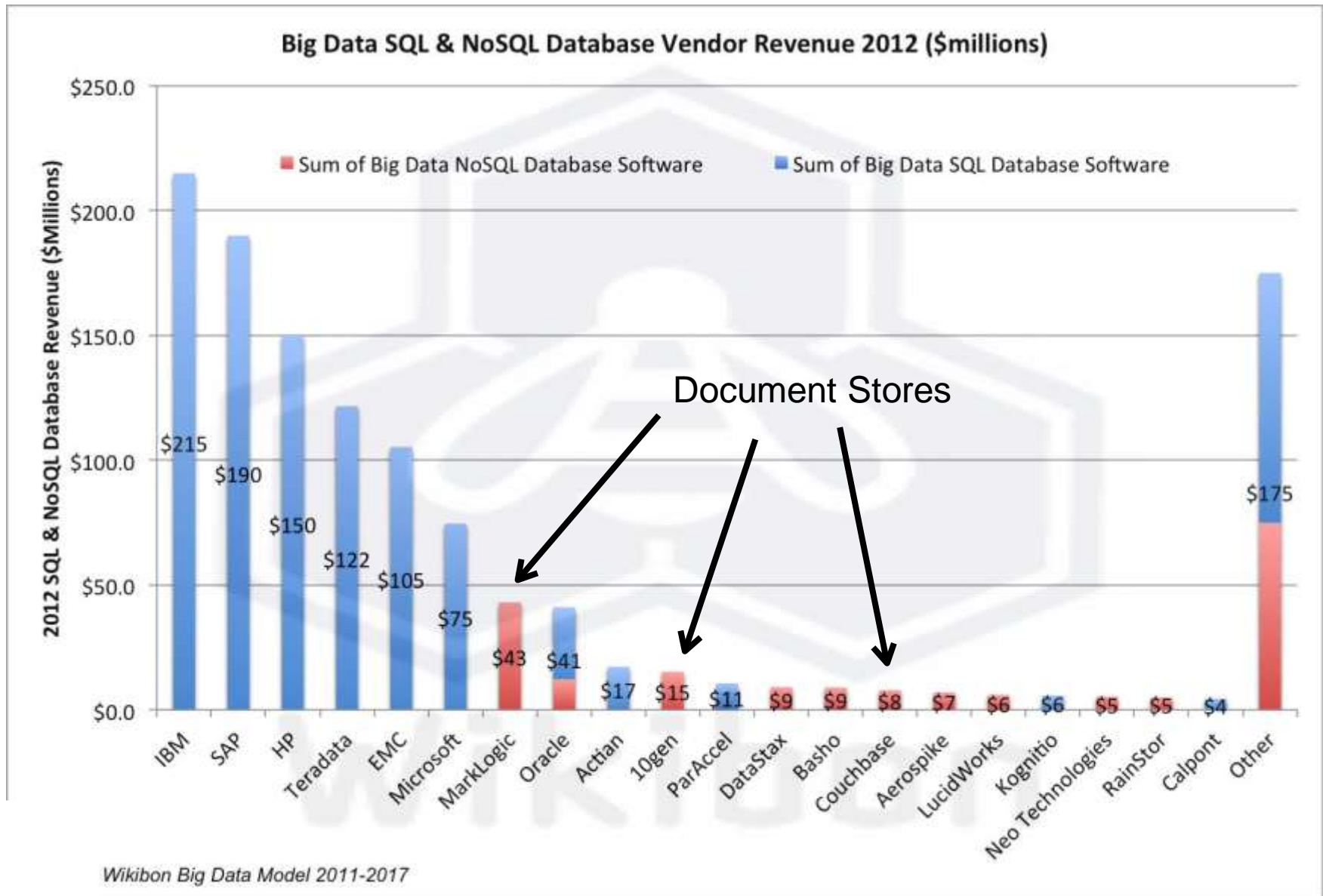
MarkLogic,
MongoDB,
Couchbase,
CouchDB, eXist-db

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

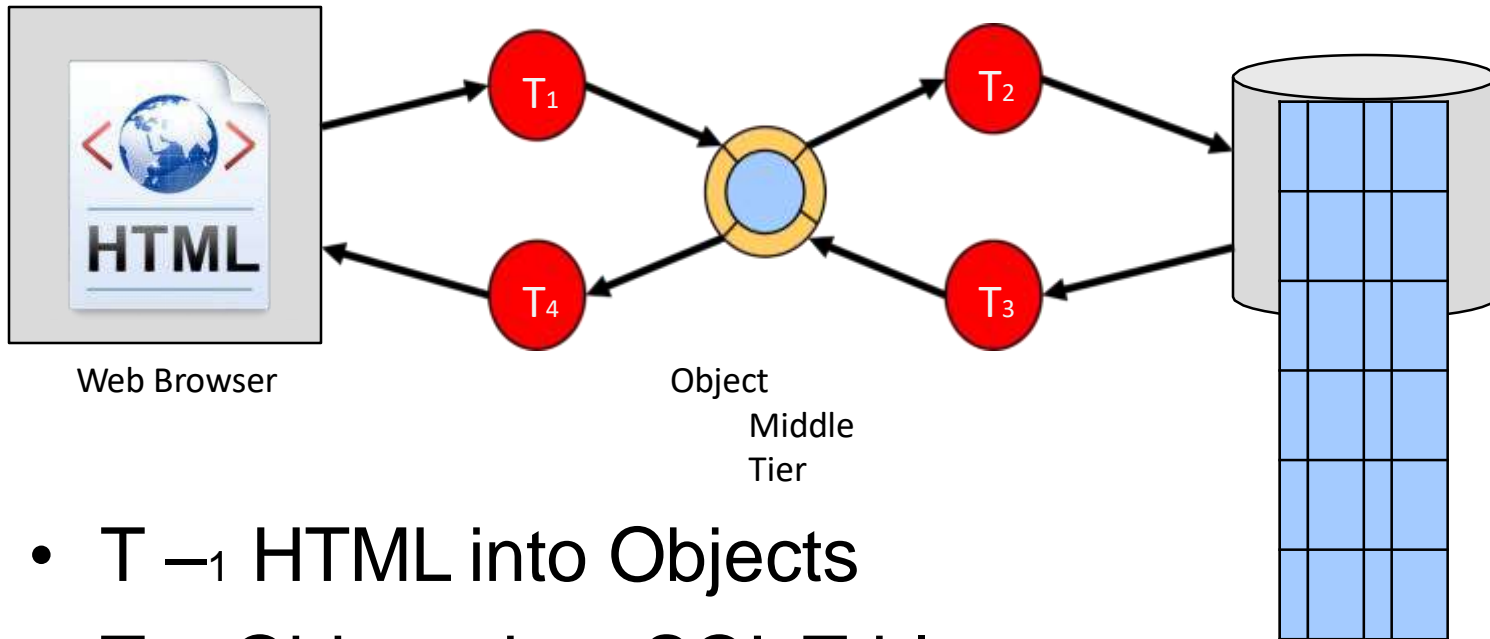
Document Stores

- Store machine readable documents together as a single blob of data
- Use JSON or XML formats to store documents
- Similar to "object stores" in many ways
- No shredding of data into tables
- Sub-trees and attributes of documents can still be queried XQuery or other document query languages
- Quickly maturing to include ACID transaction support
- Lack of object-relational mapping permits agile development
- Fastest growing revenues (MarkLogic, MongoDB, Couchbase)

Estimated Big Data and NoSQL Sales

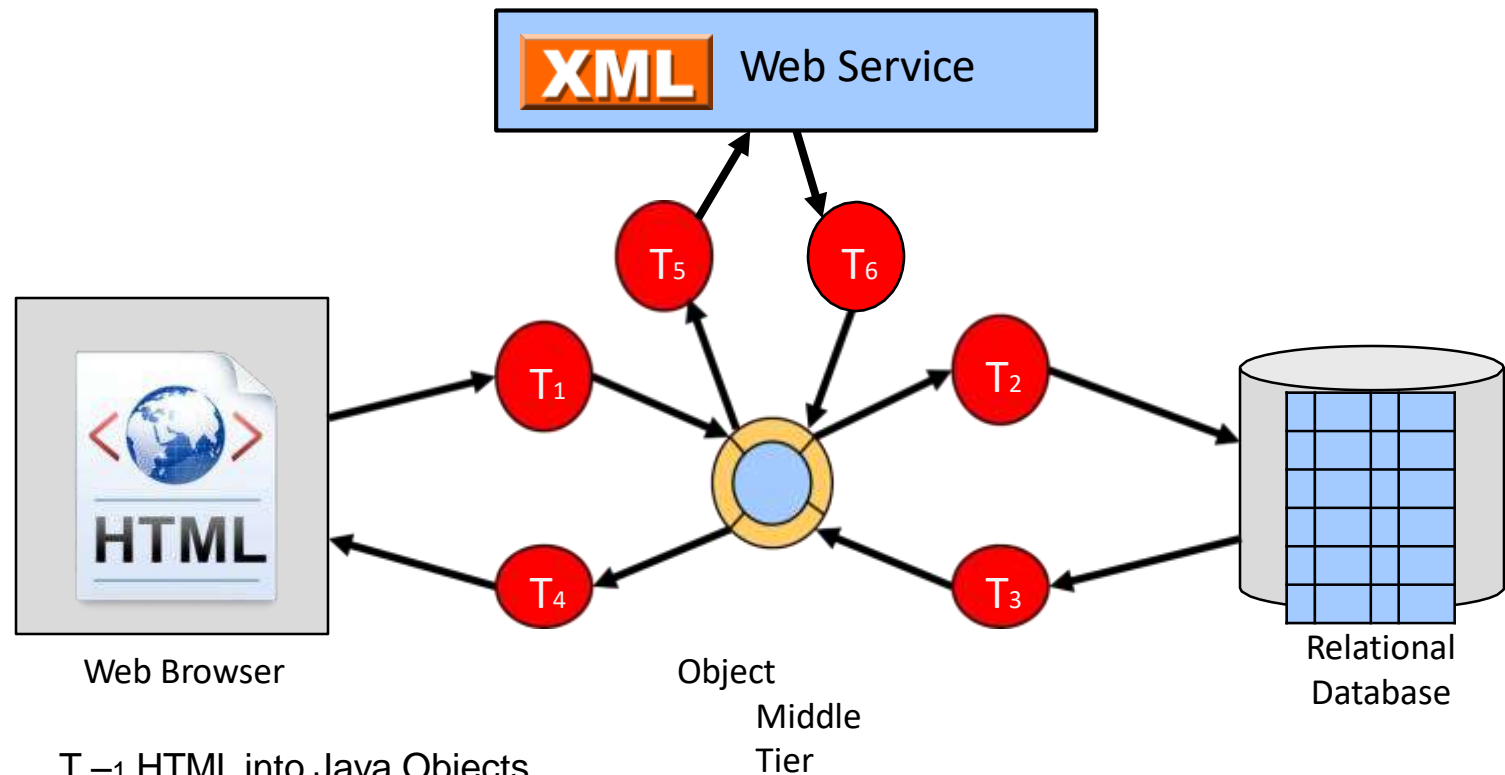


Object Relational Mapping



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

The Addition of XML Web Services

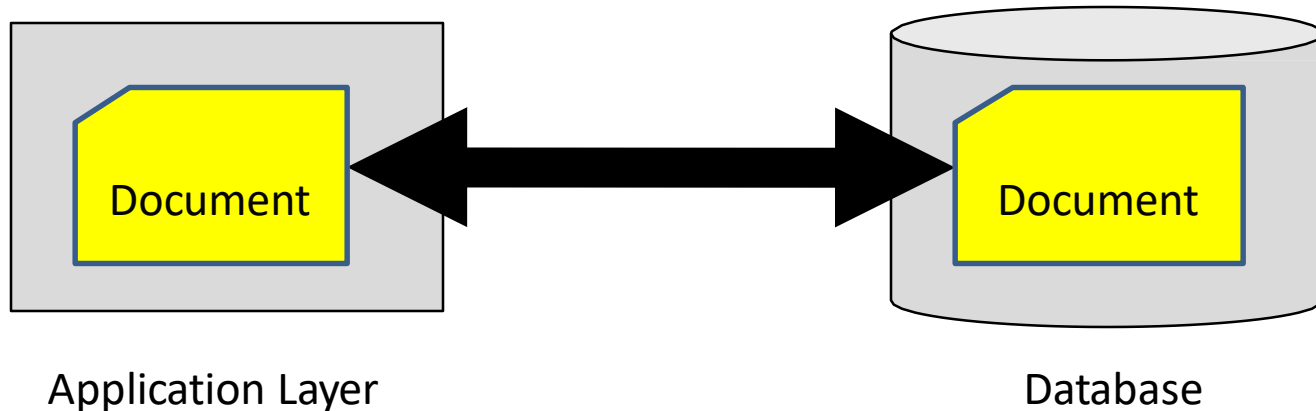


- T₁ HTML into Java Objects
- T₂ Java Objects into SQL Tables
- T₃ Tables into Objects
- T₄ Objects into HTML
- T₅ Objects to XML
- T₆ XML to Objects

'The Vietnam of Applications'

- Object-relational mapping has become one of the most complex components of building applications today
- A "Quagmire" where many projects get lost
- Many "heroic efforts" have been made to solve the problem
 - Java Hibernate Framework
 - Ruby on Rails
- But sometimes the **best** way to avoid complexity is to keep your architecture very simple

Document Stores Need No Translation



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

Zero Translation (XML)

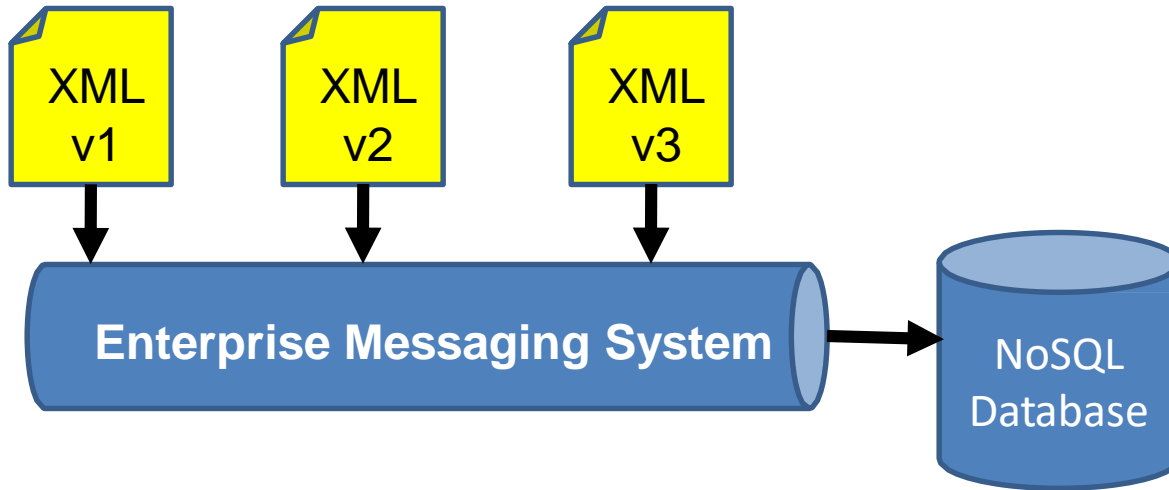


- XML lives in the web browser (**X**Forms)
- **REST** interfaces
- XML in the database (Native XML, **X**Query)
- **XRX** Web Application Architecture
- No translation!

"Schema Free"

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

Schema-Free Integration



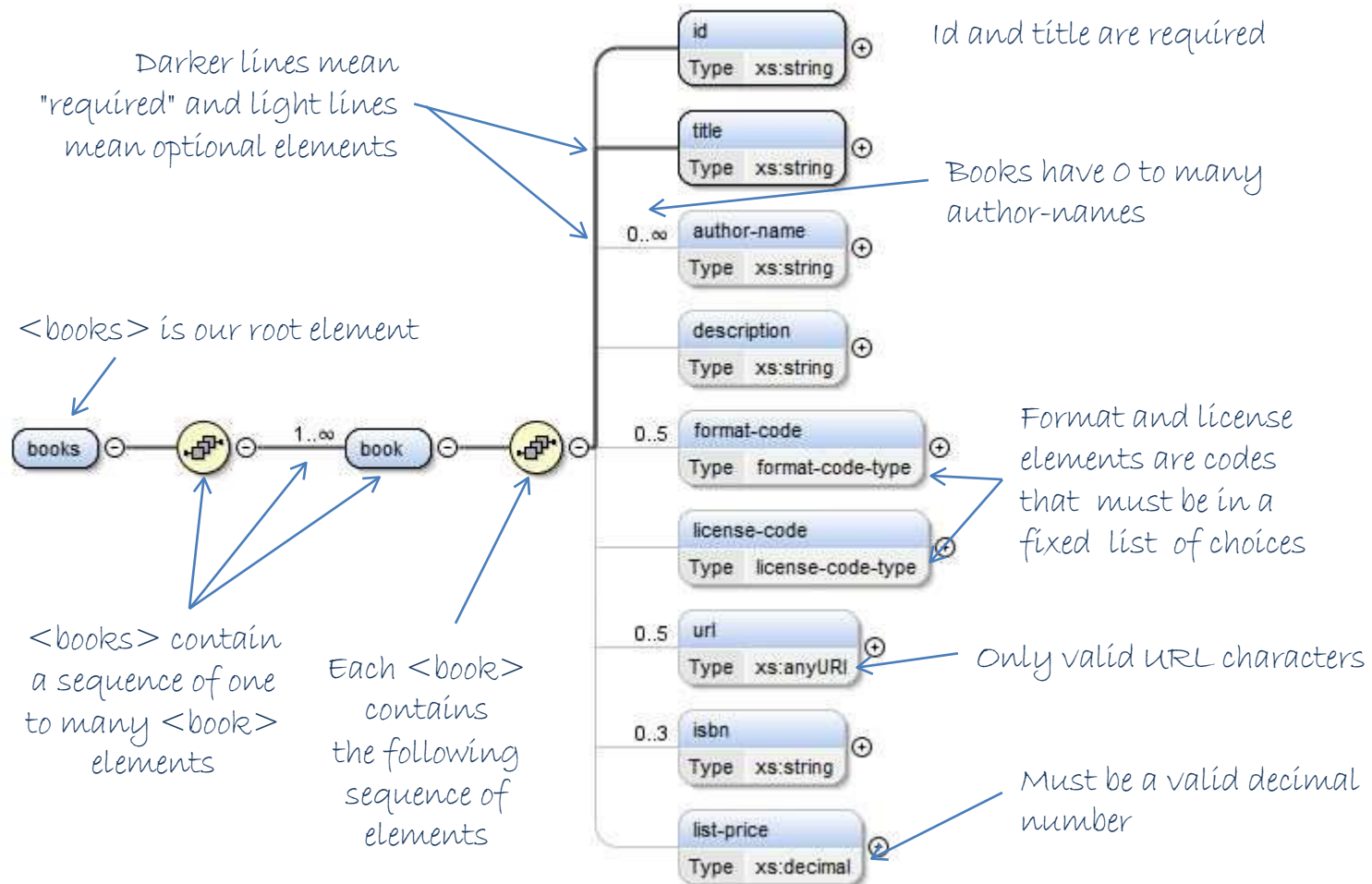
"We can easily store the data that we **actually** get, not the data we **thought** we would get."

Upfront ER Modeling is Not Required



- You do not have to **finish** modeling your data before you insert your first records
- No Data Definition Language "DDL" is needed
- Metadata is used to create indexes as data arrives
- Modeling becomes a statistical process – write queries to find exceptions and normalize data
- Exceptions make the rules but can still be used
- Data validation can still be done on documents using tools such as XML Schema and business rules systems like Schematron

Document Structure





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



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



Couchbase

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



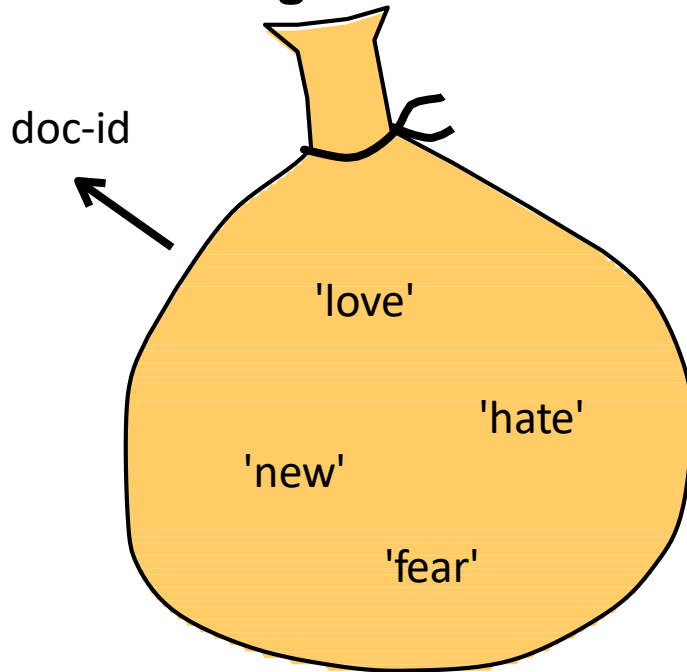
- Apache CouchDB
- Open source JSON data store
- Document Model
- Written in ERLANG
- RESTful JSON API
- Distributed, featuring robust, incremental replication with bi-directional conflict detection and management
- B-Tree based indexing
- Mobile version



- Open source native XML database
- Strong support for XQuery and XQuery extensions
- Heavily used by the Text Encoding Initiative (TEI) community and XRX/XForms communities
- Integrated Lucene search
- Collection triggers and versioning
- Extensive XQuery libs (EXPath)
- Version 2.0 has replication

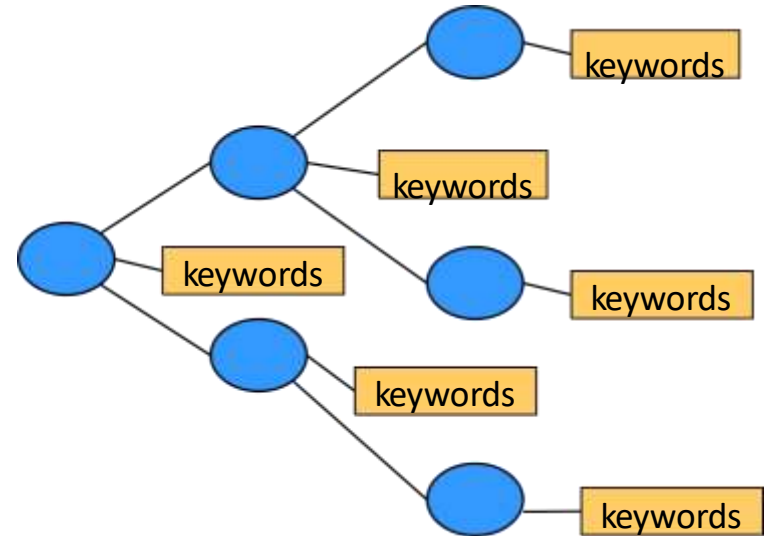
Two Models

"Bag of Words"



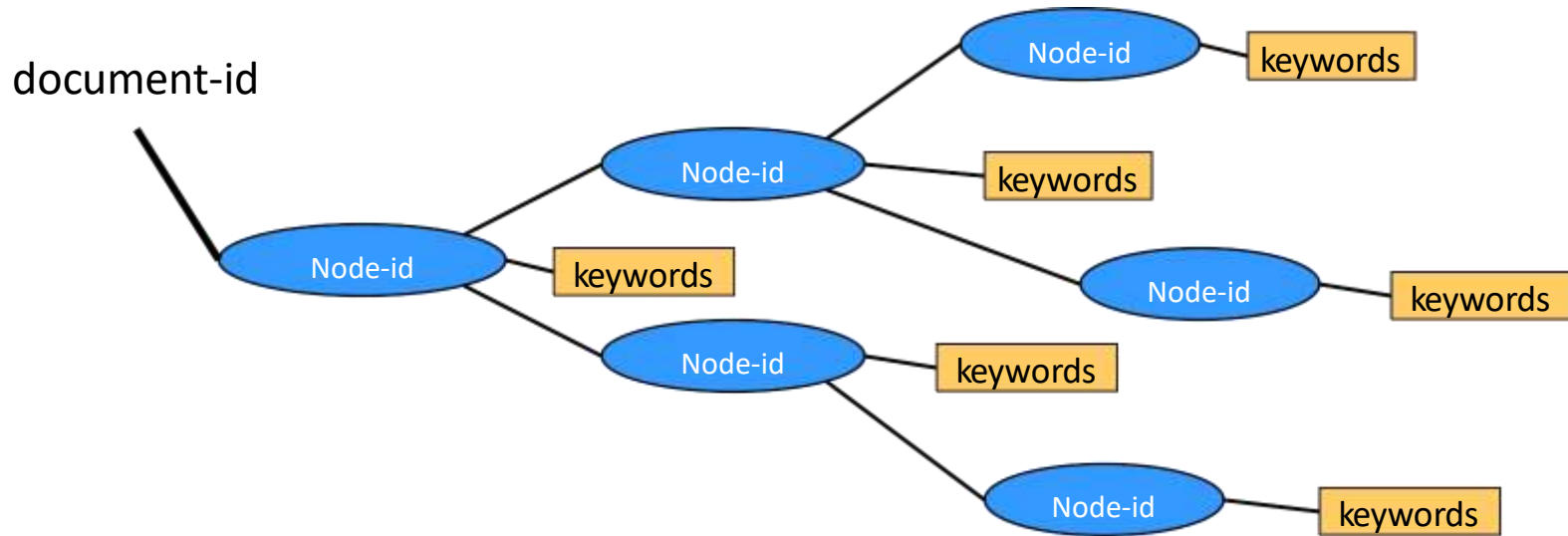
- All keywords in a single container
- Only count frequencies are stored with each word

"Retained Structure"



- Keywords associated with each sub-document component

Keywords and Node IDs



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

Hybrid architectures

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