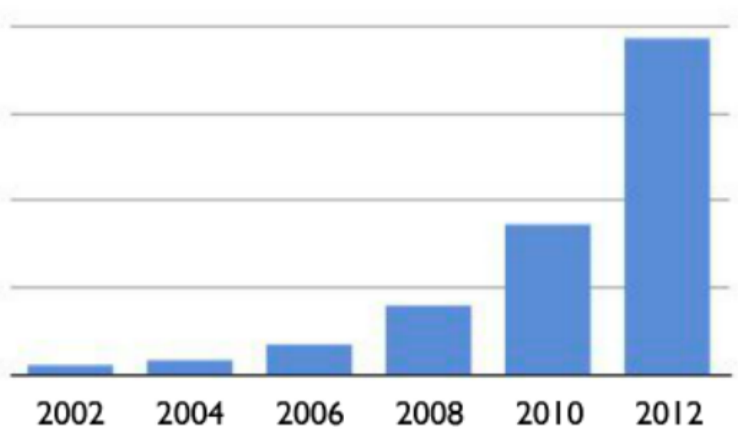
NoSQL Introduction

CS 377: Database Systems

Recap: Data Never Sleeps

https://www.domo.com/blog/2015/08/data-never-sleeps-3-0/

Web 2.0

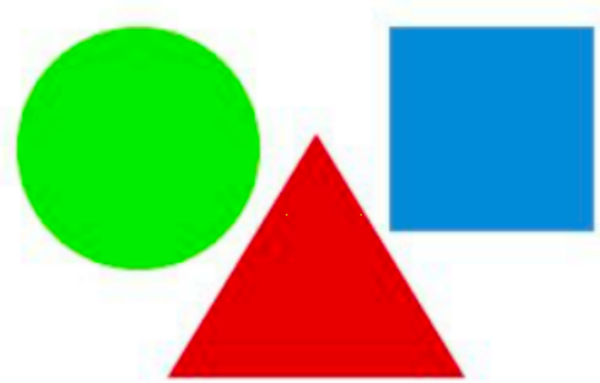


Big data

Connectivity

P2P Knowledge

Concurrency

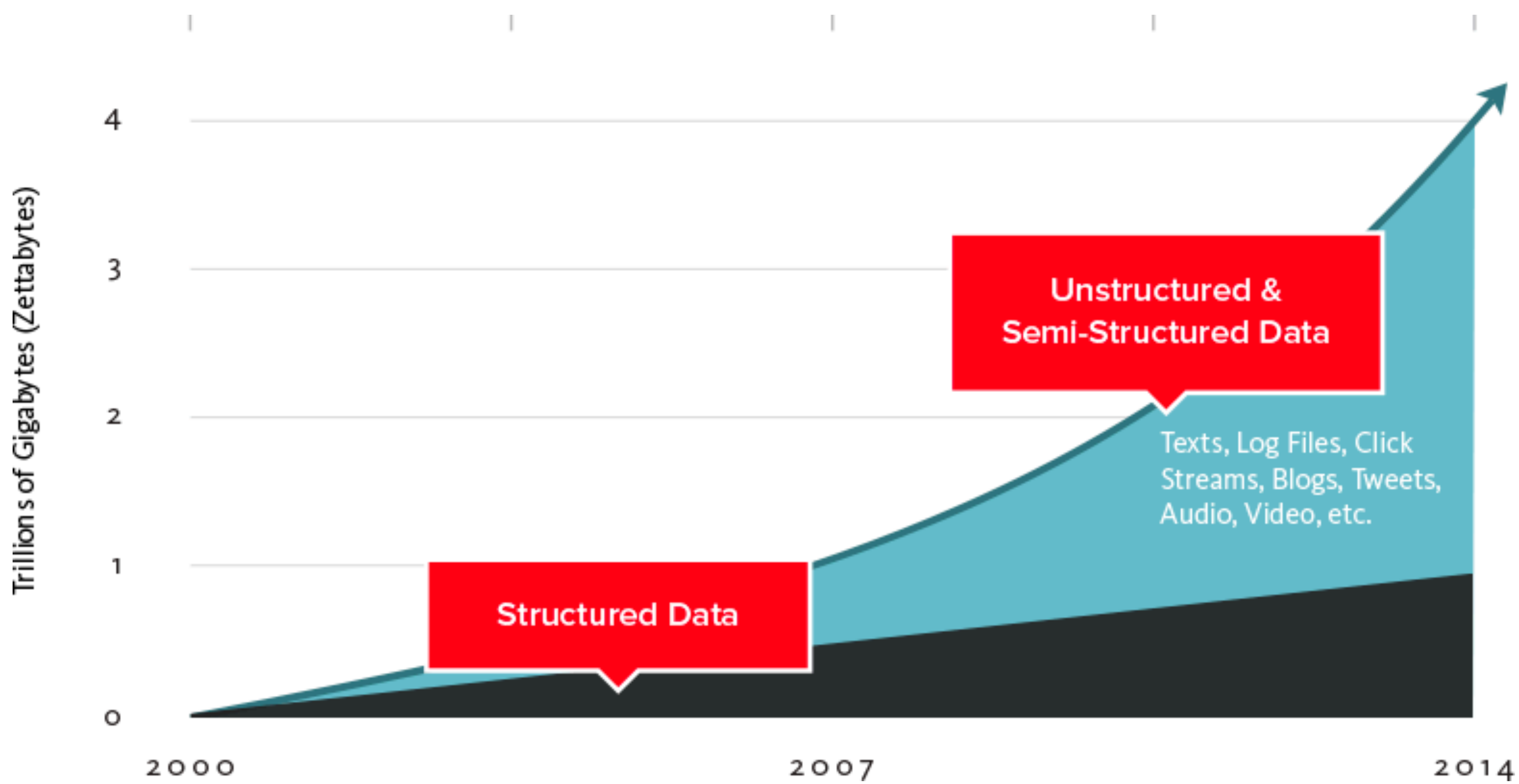


Diversity

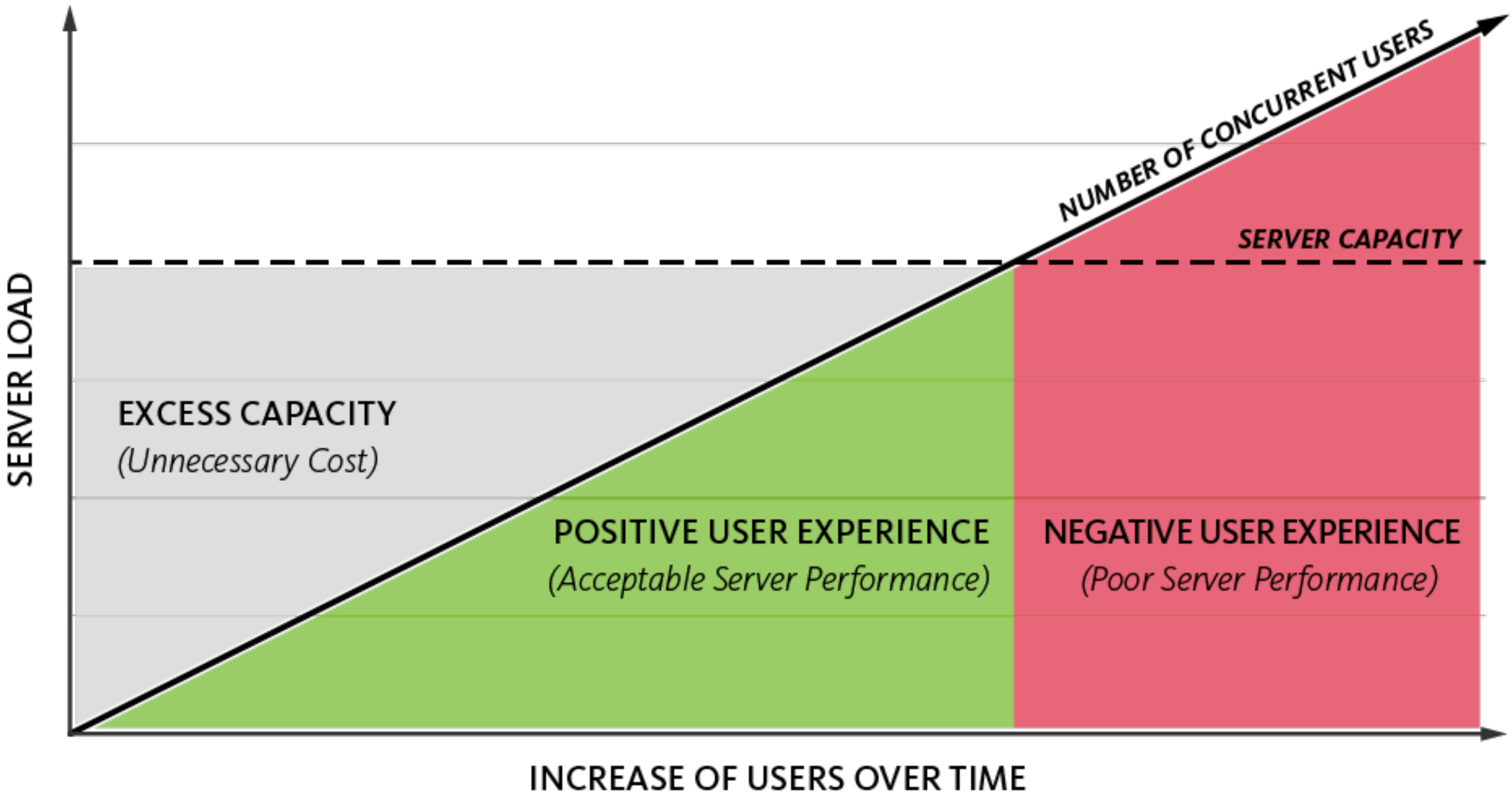
Cloud-Grid

Lorenzo Alberton Talk, "NoSQL Databases: Why, what and when"

Growth of Unstructured Data

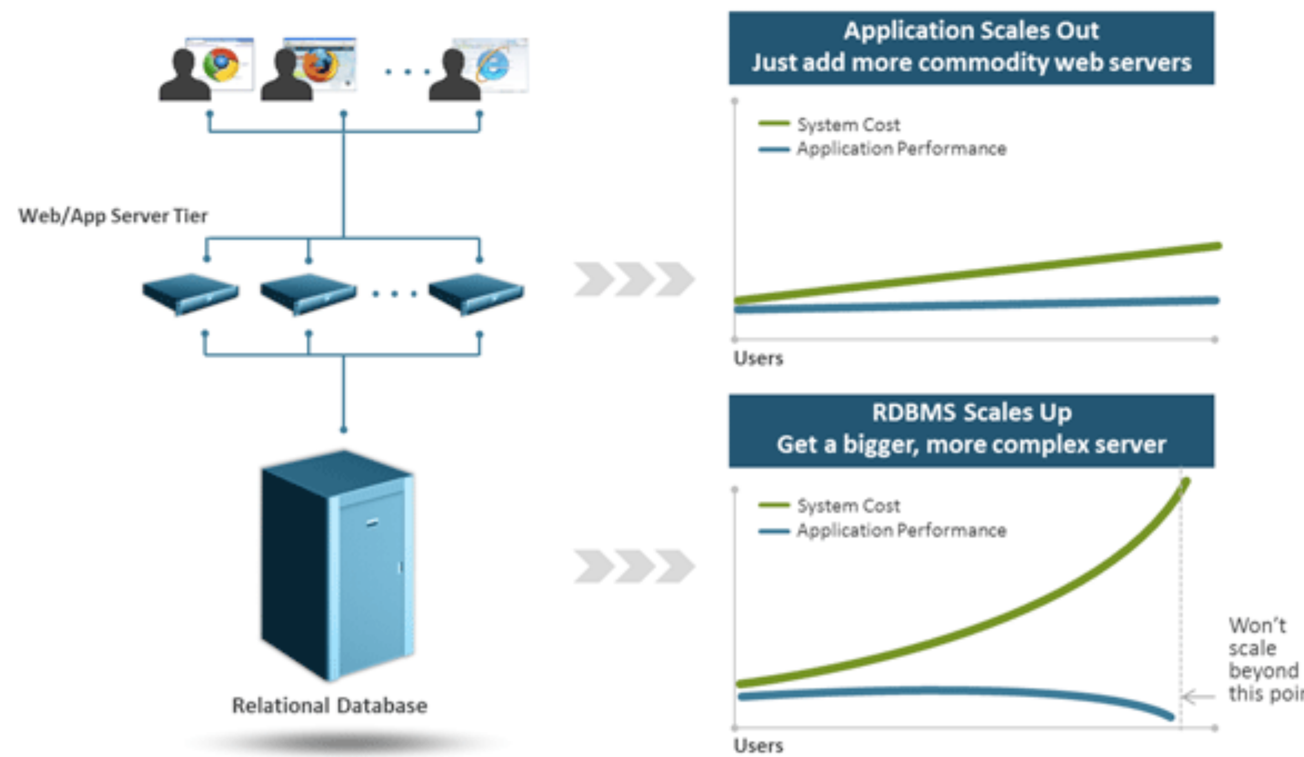


Meeting Demands



RDBMS Scaling: Add Hardware

- Large servers are highly complex, proprietary, and disproportionately expensive
- Physical limitations of systems: only so much power can be added



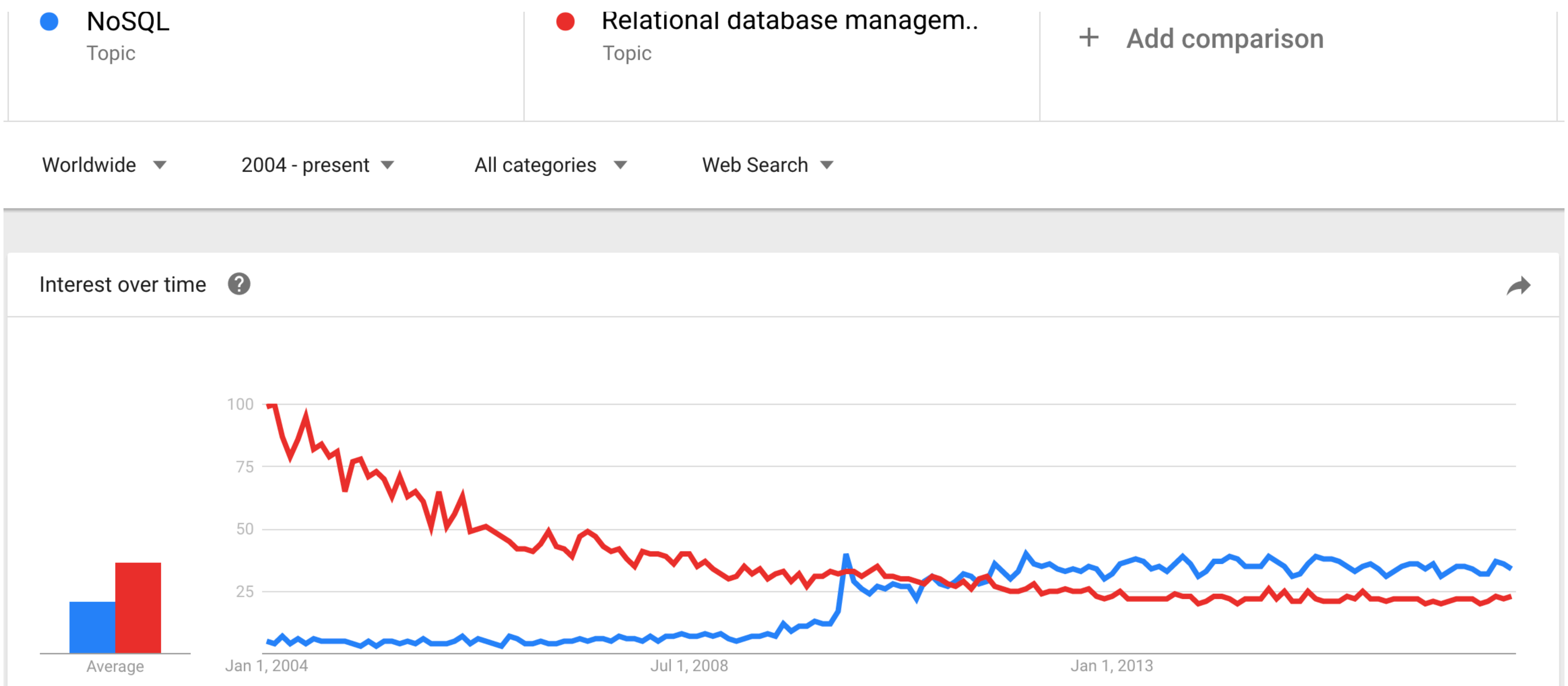
http://www.qbit.gr/news.php?n_id=933&screen=3

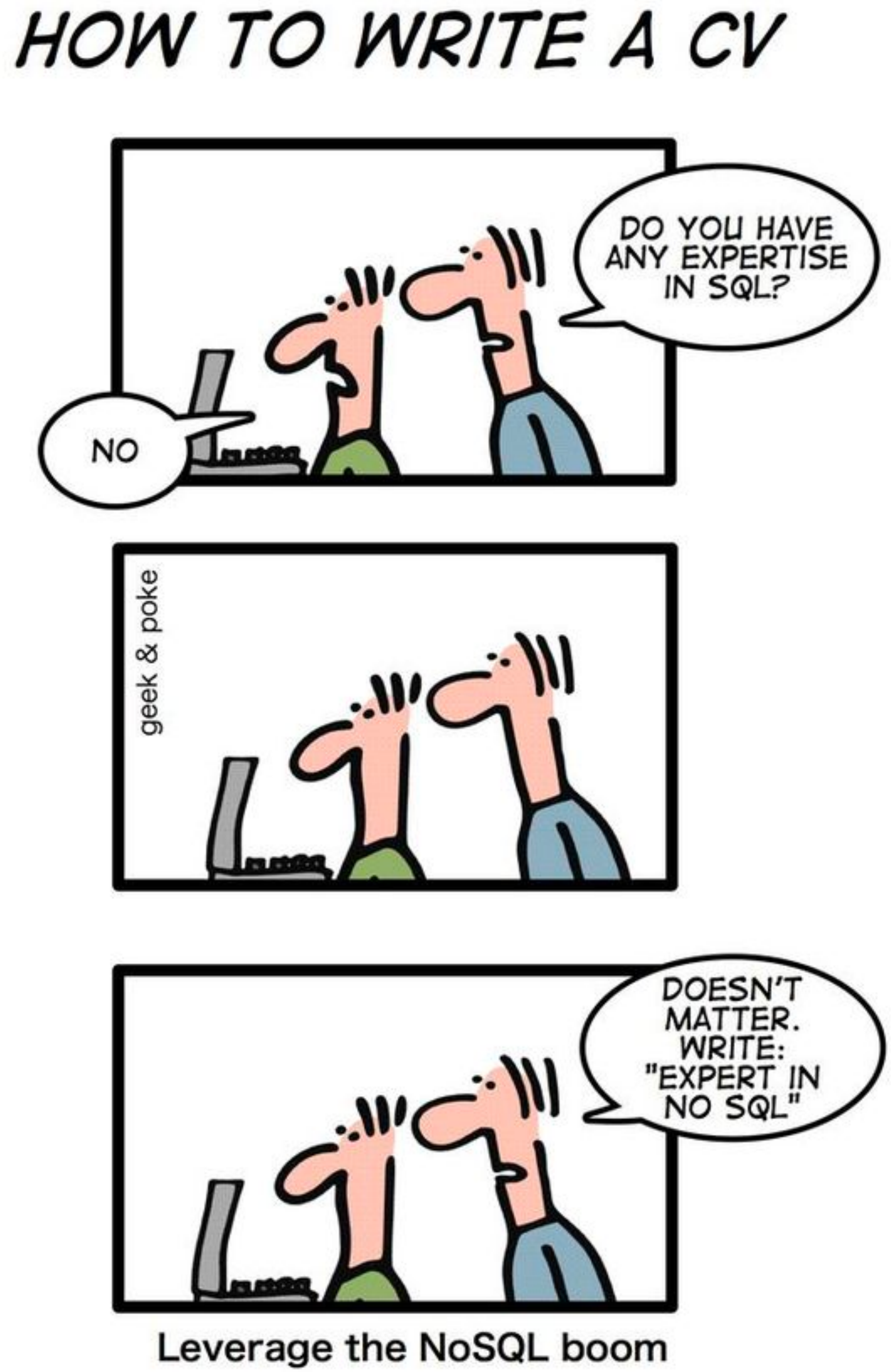
NoSQL: Motivation

- Users do both updates and reads and scaling transactions to parallel or distributed DBMS is hard
- Large servers are too expensive with maximum capacity
- Load can increase rapidly with web traffic and unpredictability
- Google and Amazon developed their own alternative approaches, BigTable and DynamoDB respectively

NoSQL: New Hipster

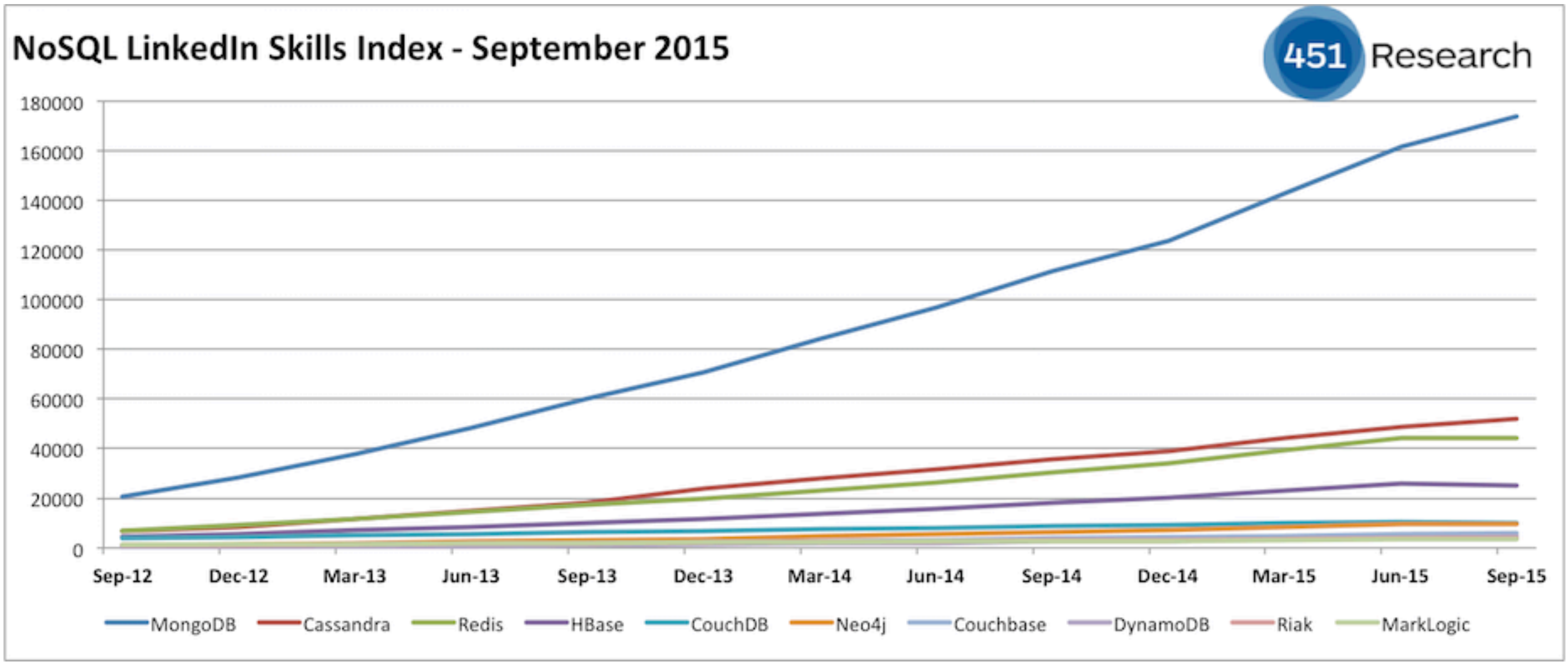
NoSQL: New Hipster





http://geekandpoke.typepad.com/geekandpoke/2011/01/nosql.html

NoSQL: Job Market



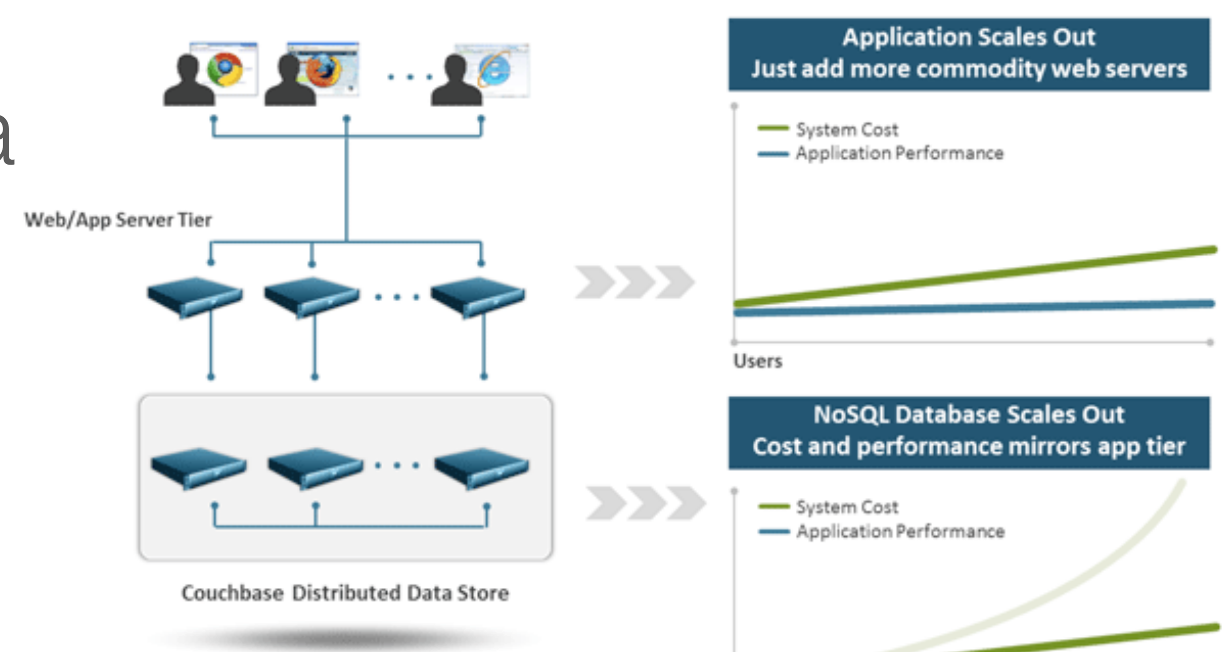
https://blogs.the451group.com/information_management/2015/10/01/nosql-linkedin-skills-index-september-2015/

What is NoSQL?

- “Not only SQL”
- Scalable by partitioning (sharding) and replication
- Distributed, fault-tolerant architecture
- Flexible schema — no fixed schema or structure
- Not a replacement for RDMBS but compliments it

NoSQL: Scaling

- Easier, linear approach to scale
- Auto-sharding spreads data across servers without application impact
- Distributed query support
- Better handling of traffic spikes



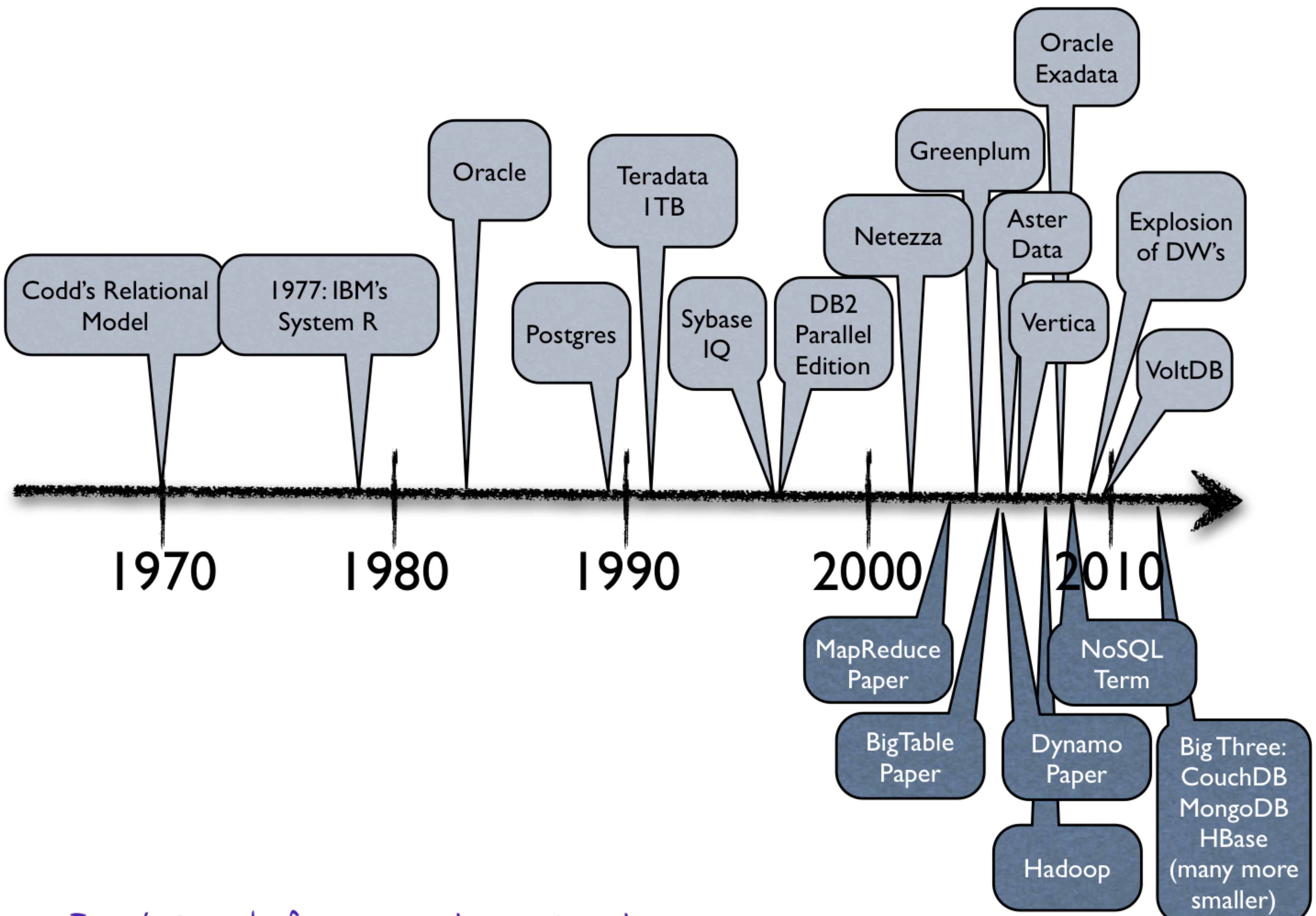
http://www.qbit.gr/news.php?n_id=933&screen=3

Review: ACID

A	C	I	D
Atomicity: Transactions are all or nothing	**Consistency**: Only valid data is saved	**Isolation**: Transactions do not affect each other	**Durability**: Written data will not be lost

But, pitfalls of DBMS with regards to latency, partition tolerance, and high availability!

DBMS Evolution



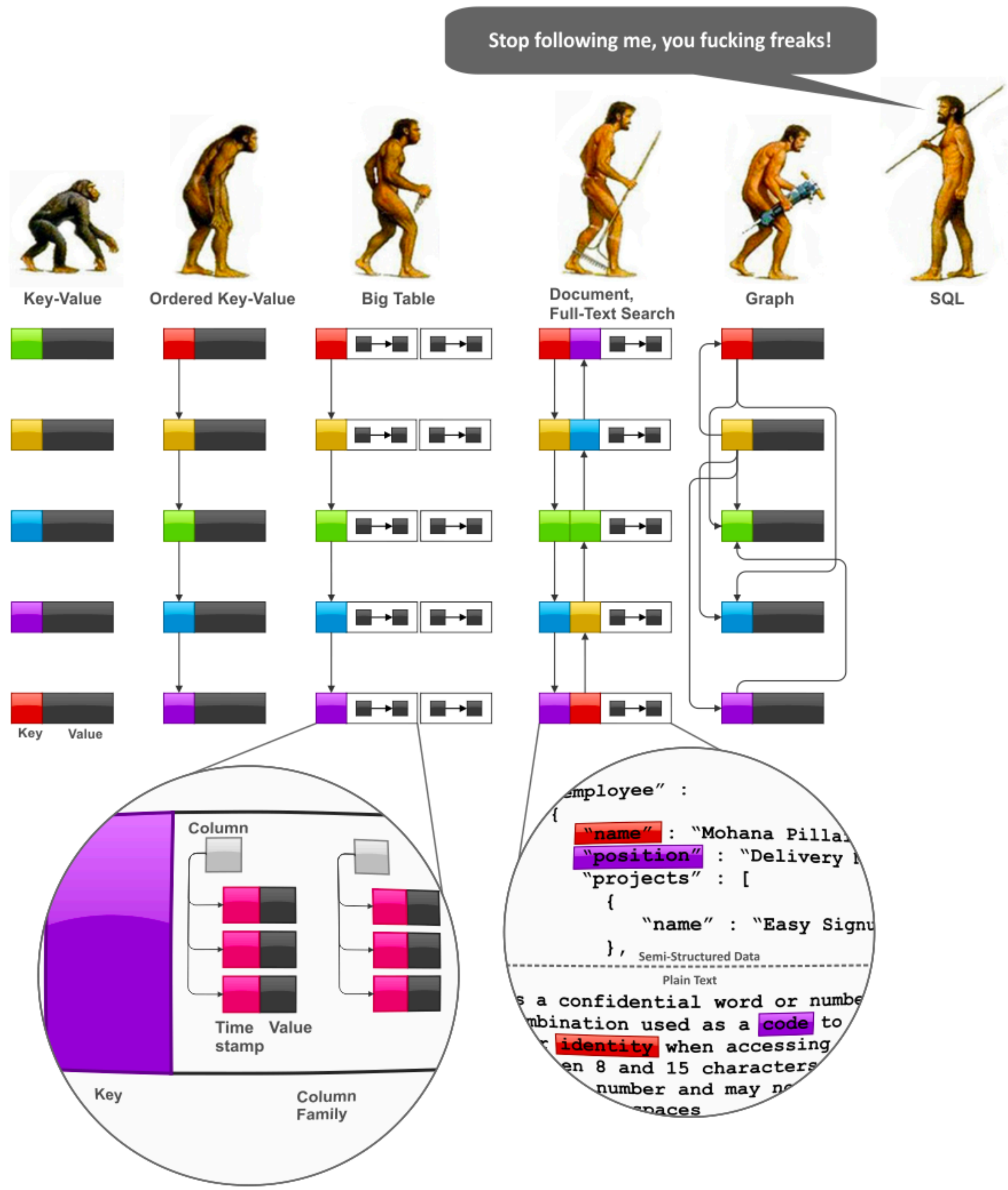
First Round of Database Wars

Object Databases challenge

Semi-structured

http://www.benstopford.com/2012/07/28/thoughts-on-big-data-technologies-part-1/

“Imaginary” Evolution of NoSQL



https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/

End of RDBMS

BLOG@CACM

The End of a DBMS Era (Might be Upon Us)

By Michael Stonebraker
June 30, 2009
Comments (7)

VIEW AS: SHARE:

ONCE MORE INTO THE CODE

By David Intersimone, Computerworld | FEB 2, 2010 12:38 PM PT

OPINION

The end of SQL and relational databases? (part 1 of 3)

overwhelming market s
a single relational engi
Moreover, the code line
elderly, in all cases dati
vendors sell software th
extended and morphed
these legacy systems ar
deserve to be sent to th

Here's why.

The relational model is dead, SQL is dead, and I don't feel so good myself

Paolo Atzeni Christian S. Jensen Giorgio Orsi Sudha Ram
Letizia Tanca Riccardo Torlone

ABSTRACT

We report the opinions expressed by well-known database researchers on the future of the relational model and SQL during a panel at the International Workshop on Non-Conventional Data Access (NoCoDa 2012), held in Florence, Italy in October 2012 in conjunction with the 31st International Conference on Conceptual Modeling. The panelists include: Paolo Atzeni (Università Roma Tre, Italy), Umeshwar Dayal (HP Labs, USA), Christian S. Jensen (Aarhus University, Denmark), and Sudha Ram (University of Arizona, USA). Quotations from movies are used as a playful though effective way to convey the dramatic changes that database technology and research are currently undergoing.

ing data using the relational model. The debate on SQL vs. NoSQL is as much a debate on SQL, the language, as on the relational model and its various implementations.

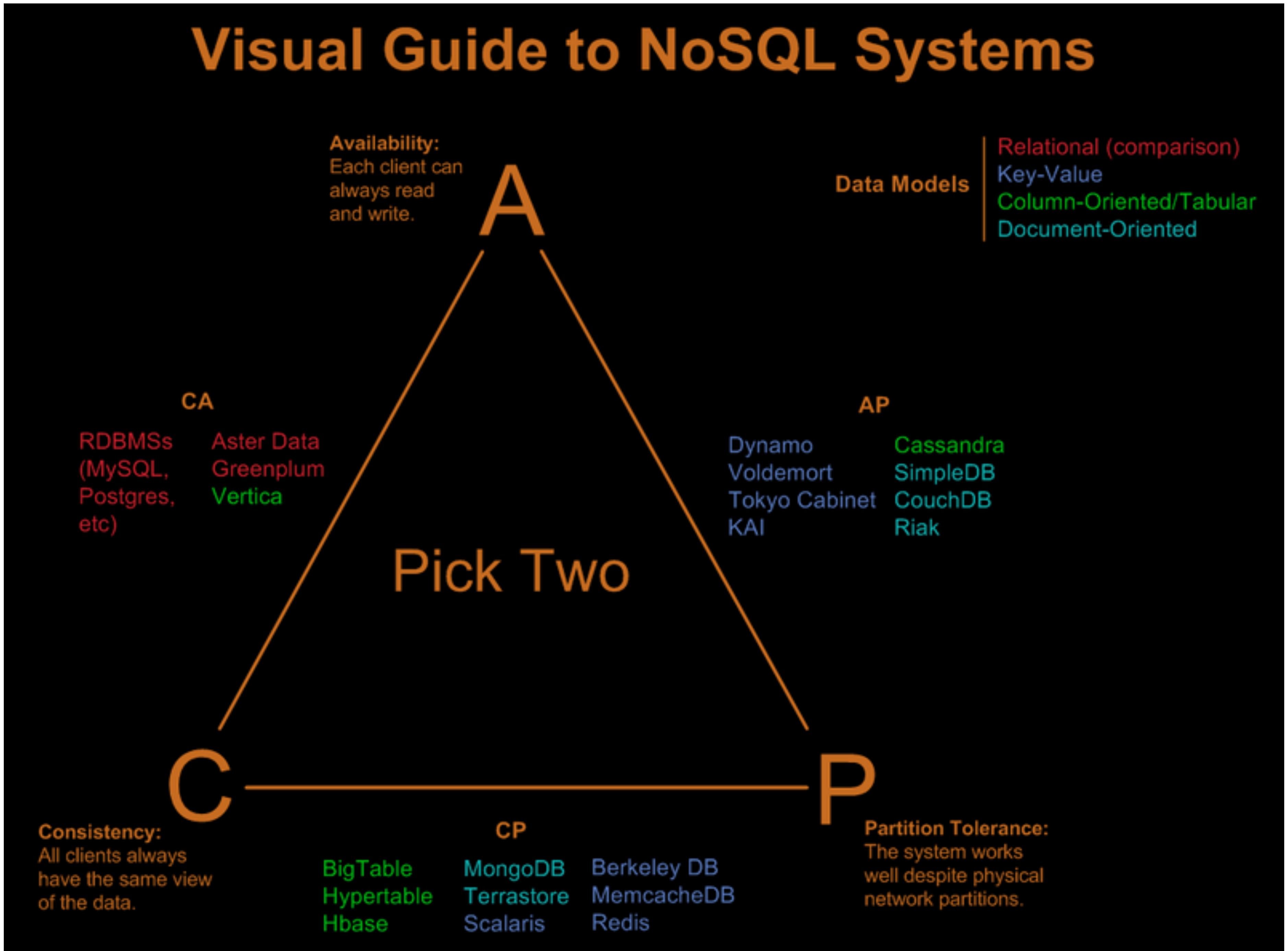
Relational database management systems have been around for more than thirty years. During this time, several revolutions (such as the Object Oriented database movement) have erupted, many of which threatened to doom SQL and relational databases. These revolutions eventually fizzled out, and none made even a small dent in the dominance of relational databases. The latest revolution appears to be from NoSQL databases that are touted to be non-relational, horizontally scalable, distributed and, for the most part, open source.

CAP Theorem

"Of three properties of shared-data systems — data Consistency, system Availability, and tolerance to network Partitions — only two can be achieved at any given moment in time" — Brewer, 1999

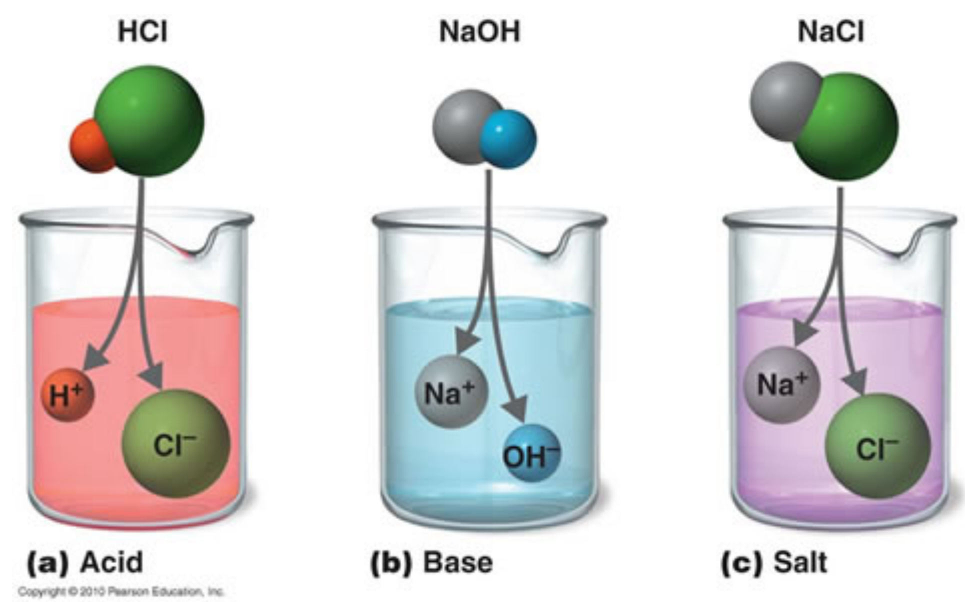
- Consistency: all nodes see the same data at the same time
- Availability: guarantee that every request receives a response about whether it was successful or failed
- Partition tolerance: system continues to operate despite arbitrary message loss or failure of part of the system

NoSQL Systems and CAP



http://blog.nahurst.com/visual-guide-to-nosql-systems

Changing pH of Transactions



ACID —> BASE

NoSQL Paradigm: BASE

- Basically Available: replication and sharing to reduce likelihood of data unavailability and use partitioning of the data to make any remaining failures partial
- Soft state: allow data to be inconsistent, which means that the state of system may change over time even without input
- Eventually consistent: at some future point in time, the data assumes a consistent state and not immediate like ACID

NoSQL: Categories

- Four groups:
  - Key-value stores
  - Column-based families or wide column systems
  - Document stores
  - Graph databases ← Debate about whether it is NoSQL
- Categories can be subject to change in the future

NoSQL: Categories

All in the NoSQL Family

NoSQL databases are geared toward managing large sets of varied and frequently updated data, often in distributed systems or the cloud. They avoid the rigid schemas associated with relational databases. But the architectures themselves vary and are separated into four primary classifications, although types are blending over time.

Document databases

Store data elements in document-like structures that encode information in formats such as JSON.

+

Common uses include content management and monitoring Web and mobile applications.

+

EXAMPLES:
Couchbase Server, CouchDB, MarkLogic, MongoDB

Graph databases

Emphasize connections between data elements, storing related "nodes" in graphs to accelerate querying.

+

Common uses include recommendation engines and geospatial applications.

+

EXAMPLES:
Allegrograph, IBM Graph, Neo4j

Key-value databases

Use a simple data model that pairs a unique key and its associated value in storing data elements.

+

Common uses include storing clickstream data and application logs.

+

EXAMPLES:
Aerospike, DynamoDB, Redis, Riak

Wide column stores

Also called table-style databases—store data across tables that can have very large numbers of columns.

+

Common uses include Internet search and other large-scale Web applications.

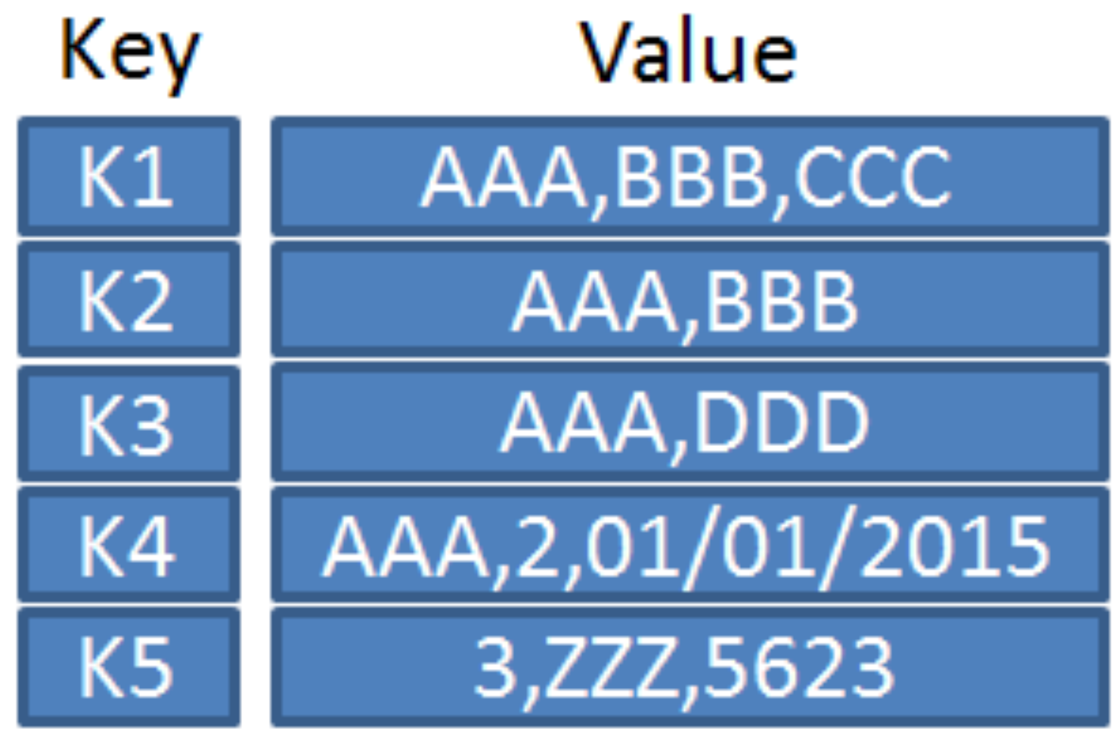
+

EXAMPLES:
Accumulo, Cassandra, HBase, Hypertable, SimpleDB

http://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL

Key-Value Store

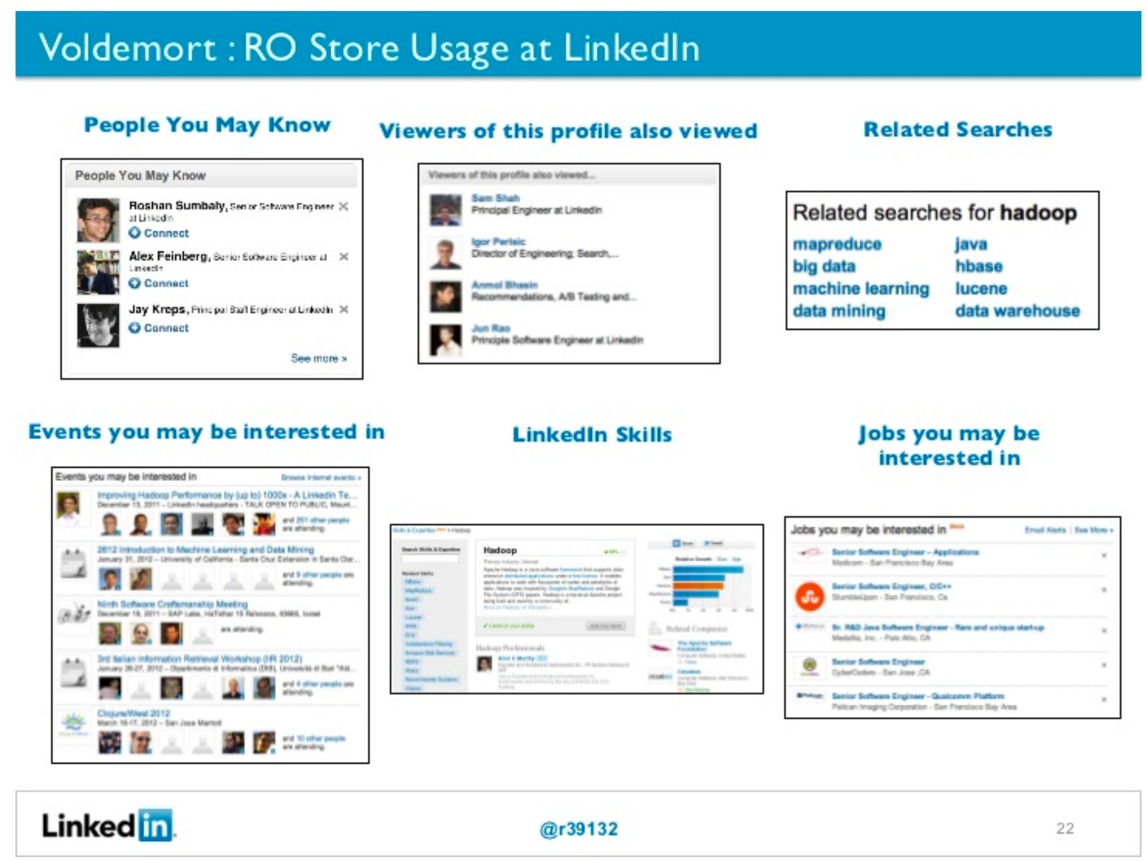
- Simplest NoSQL databases — collection of key, value pairs
- Queries are limited to query by key
- Example: Riak, Redis, Voldermort, DynamoDB, MemcacheDB



https://upload.wikimedia.org/wikipedia/commons/5/5b/KeyValue.PNG

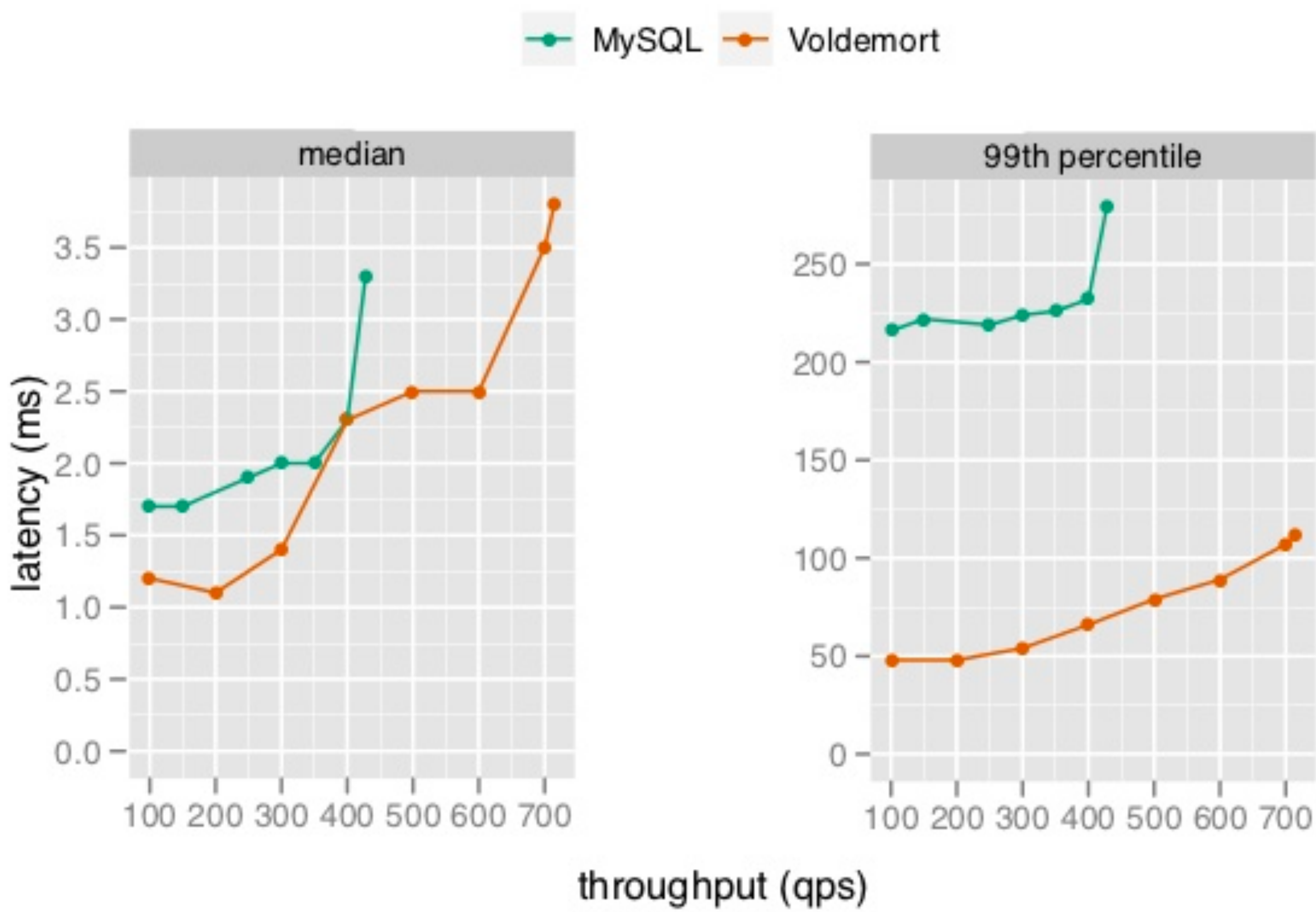
Key-Value Store: Voldemort

- Distributed data store used by LinkedIn for high-scalability storage
- Named after fictional Harry Potter villain
- Addresses two usage patterns
  - Read-write store
  - Read-only store



http://www.slideshare.net/r39132/linkedin-data-infrastructure-qcon-london-2012/22-Voldemort_RO_Store_Usage_at

Voldemort vs MySQL: Read Only



100 GB data, 24 GB RAM

http://www.slideshare.net/r39132/linkedin-data-infrastructure-qcon-london-2012/25-Voldemort_RO_Store_Performance_TP

Column-Based Families

- Data is stored in a big table except you store columns of data together instead of rows
- Access control, disk and memory accounting performed on column families
- Example: HBase, Cassandra, Hypertable

Example: Column-Based

Multi-valued

Row Oriented
(RDBMS Model)

id	Name	Age	Interests
1	Ricky		Soccer, Movies, Baseball
2	Ankur	20	
3	Sam	25	Music

null

Column Oriented
(Multi-value sorted map)

id	Name
1	Ricky
2	Ankur
3	Sam

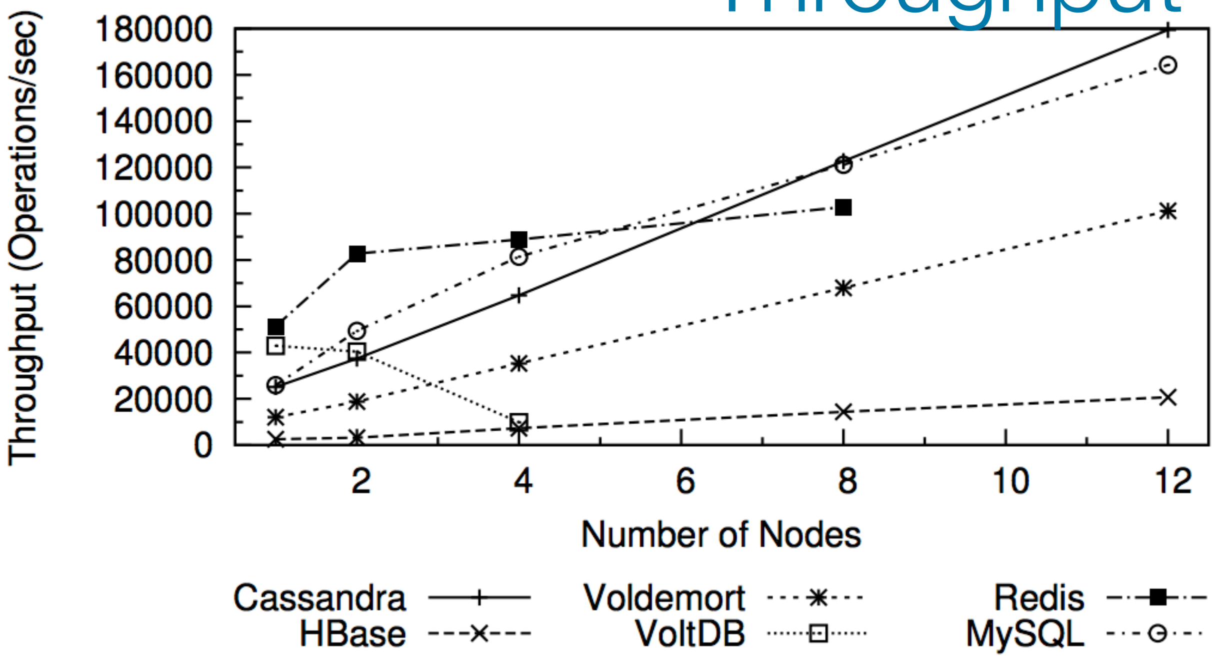
id	Age
2	20
3	25

id	Interests
1	Soccer
1	Movies
1	Baseball
3	Music

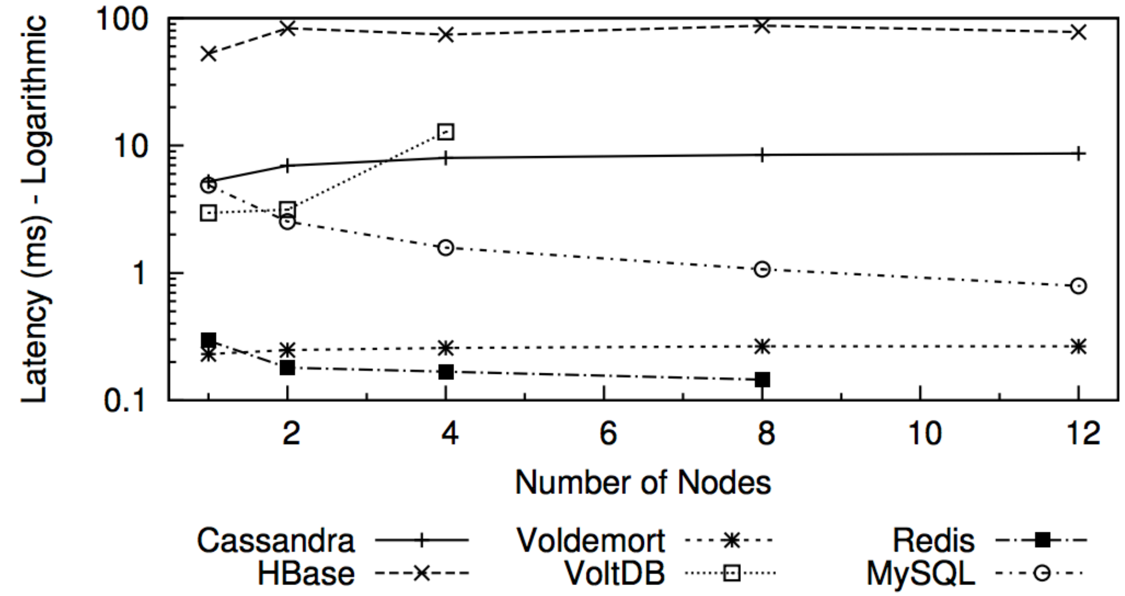
https://dzone.com/articles/bigtable-model-cassandra-and

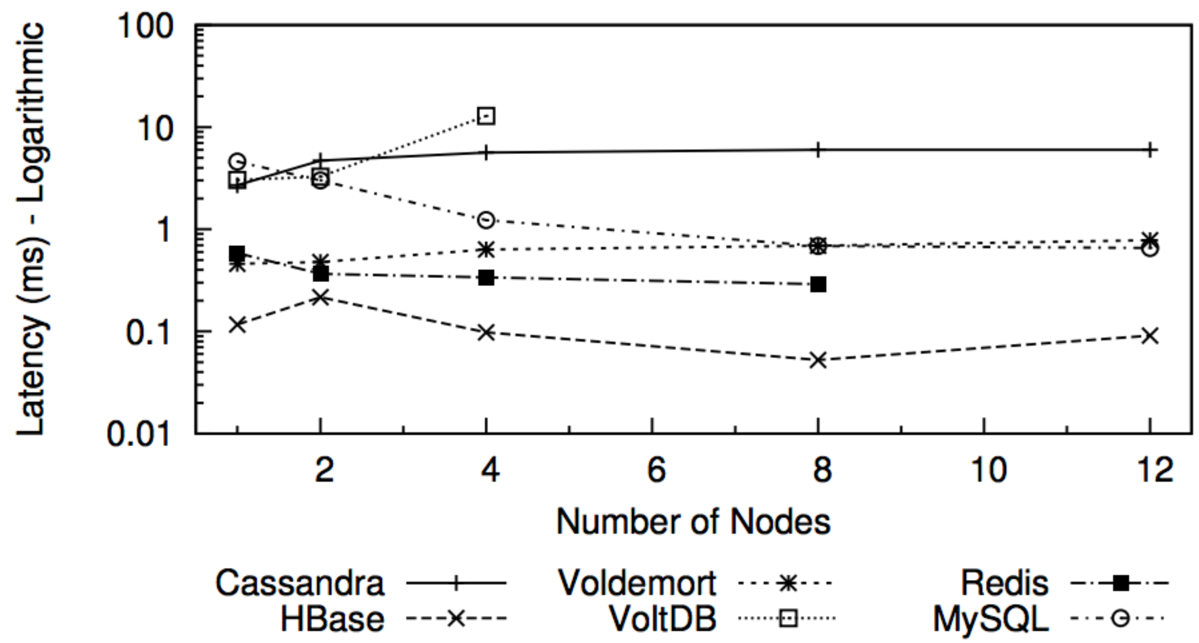
Comparison: 95% Read - 5% Write

Throughput



Read latency



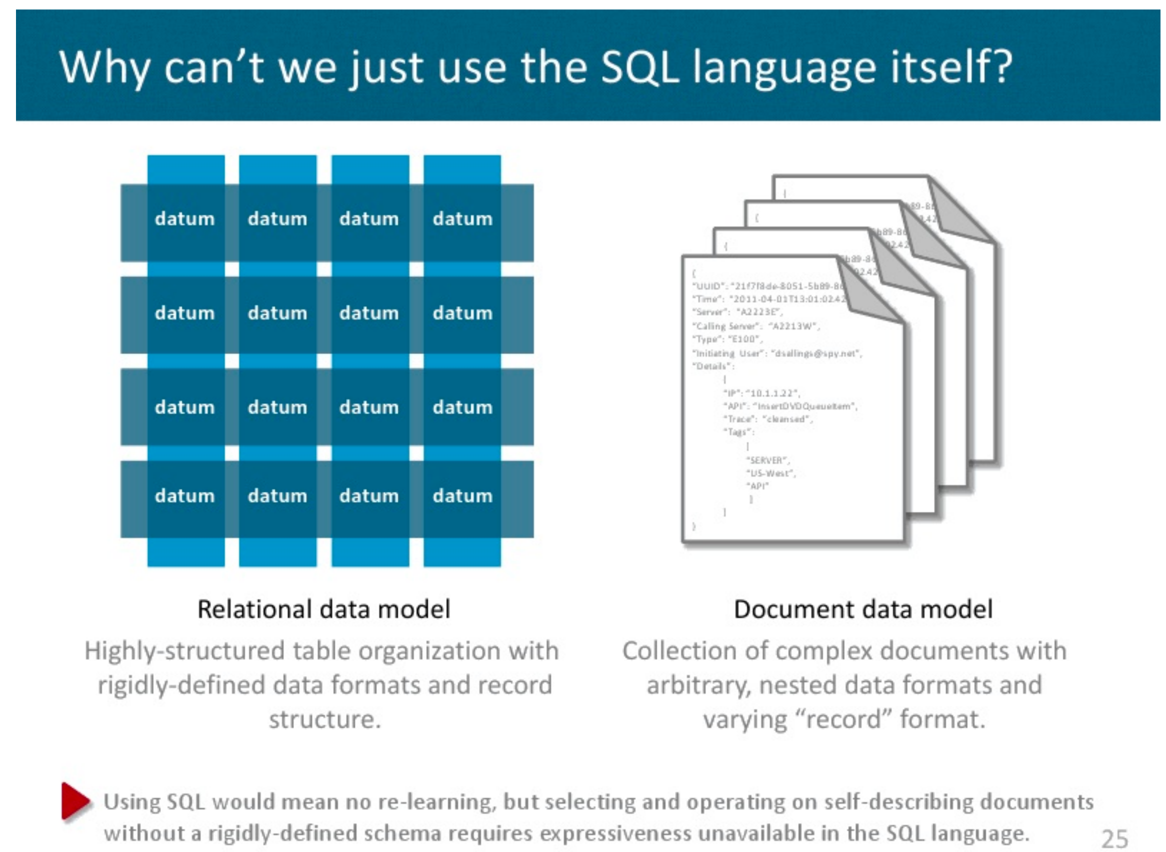


Write latency

http://vldb.org/pvldb/vol5/p1724_tilmannrabl_vldb2012.pdf

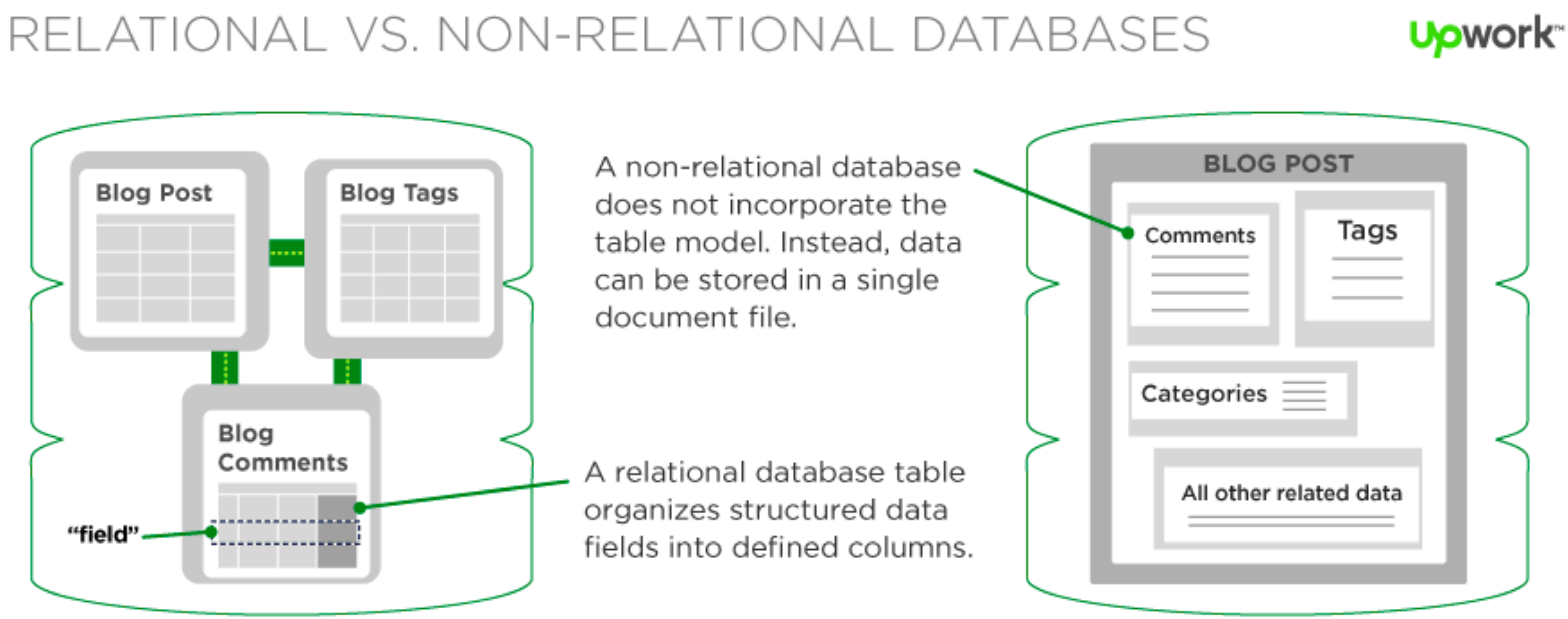
Document Databases

- Collections of similar documents
- Each document can resemble a complex model
- Examples: MongoDB, CouchDB



https://gigaom.com/wp-content/uploads/sites/1/2011/07/unql-1.jpg

Relational vs Non-relational DB



https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/

JavaScript Object Notation (JSON)

- Simple, text-based way to store and transmit data
- Alternative data model for semi-structured data
- Compact and easy to read
- Maps easily to data structures used by most programming languages

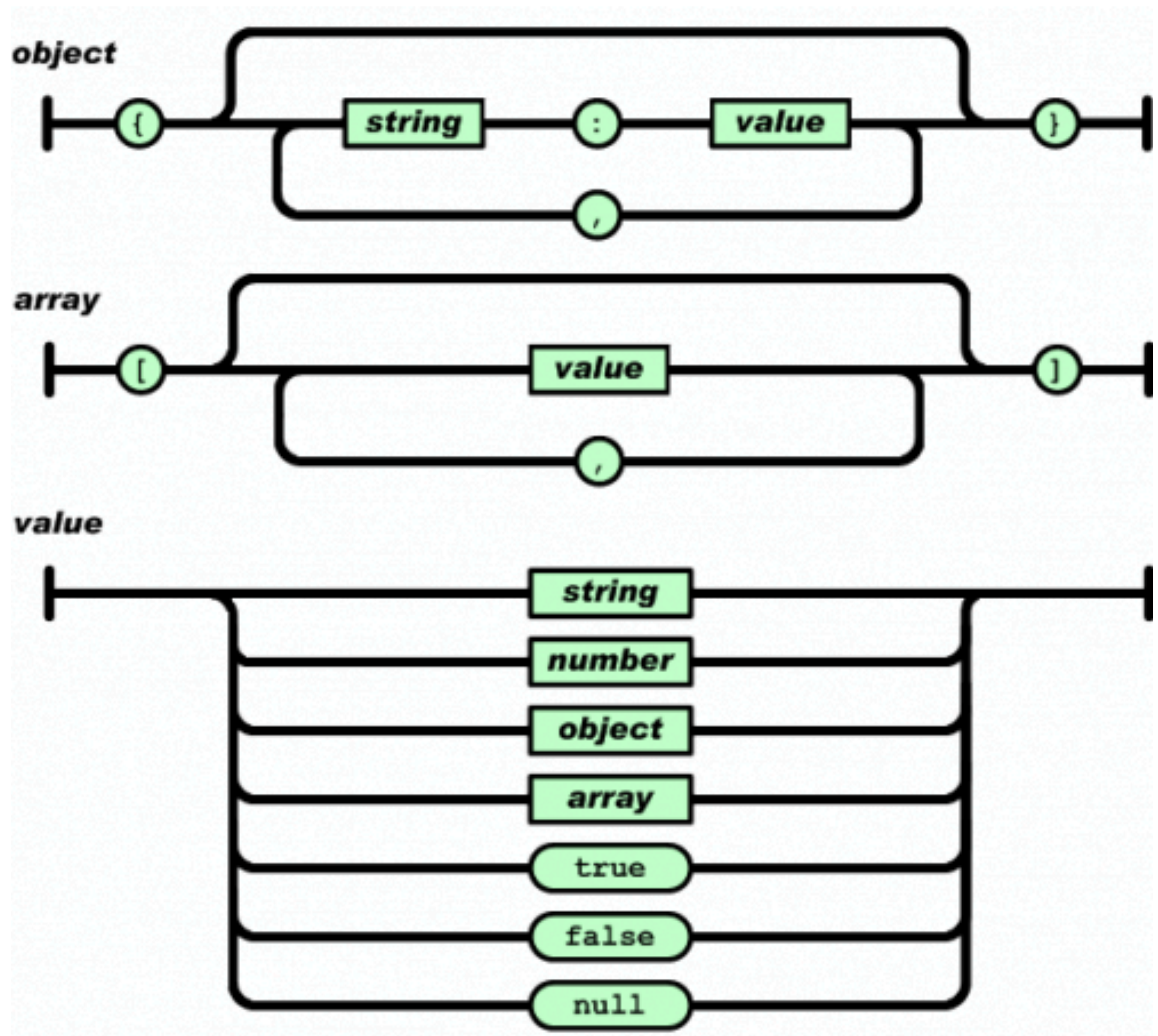

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumber": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax", "number": "646 555-4567" }
    ]
}
```


http://natishalom.typepad.com/.a/6a00d835457b7453ef0133f2872d36970b-pi

JSON

- Typically used in web applications to send data from server to browser
- Built on two key structures
  - Object is a sequence of fields (name, value pairs)
  - Array of values

JSON

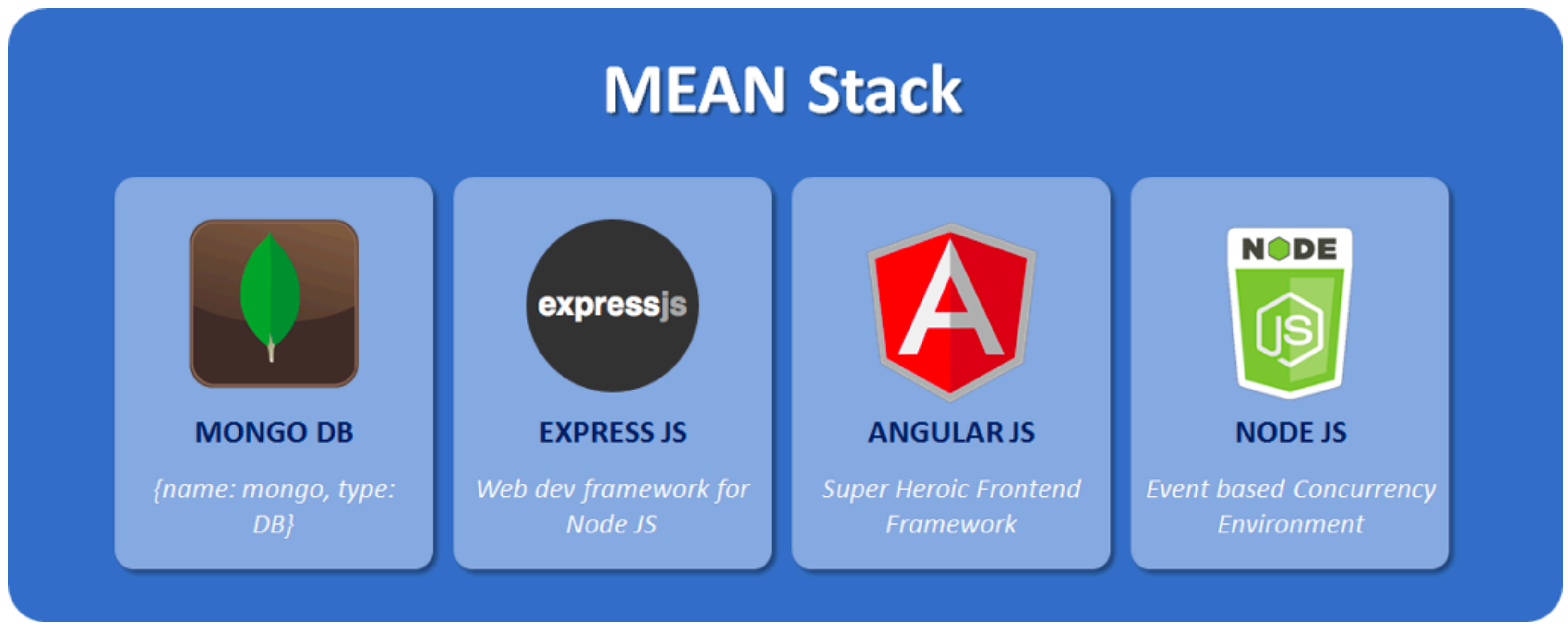


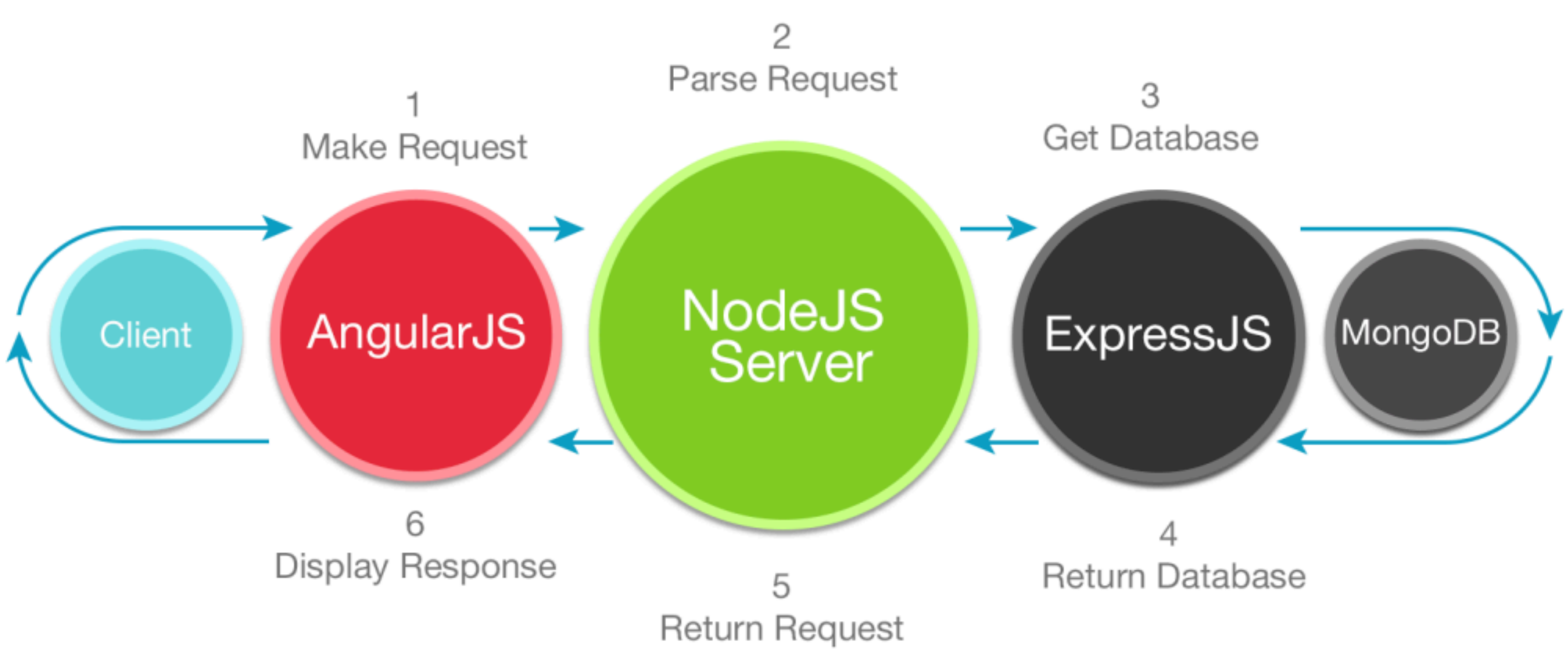
http://interactive-matter.eu/blog/2010/08/14/ajson-handle-json-with-arduino/

Document Database: MongoDB

- Open-source NoSQL database released in 2009
- Database contains zero or more collections
- Collection can have zero or more documents
    - Documents can have multiple fields
    - Documents need not have the same fields

Modern Web Stack





https://www.dealfuel.com/seller/mean-stack-tutorial/

MongoDB: Document

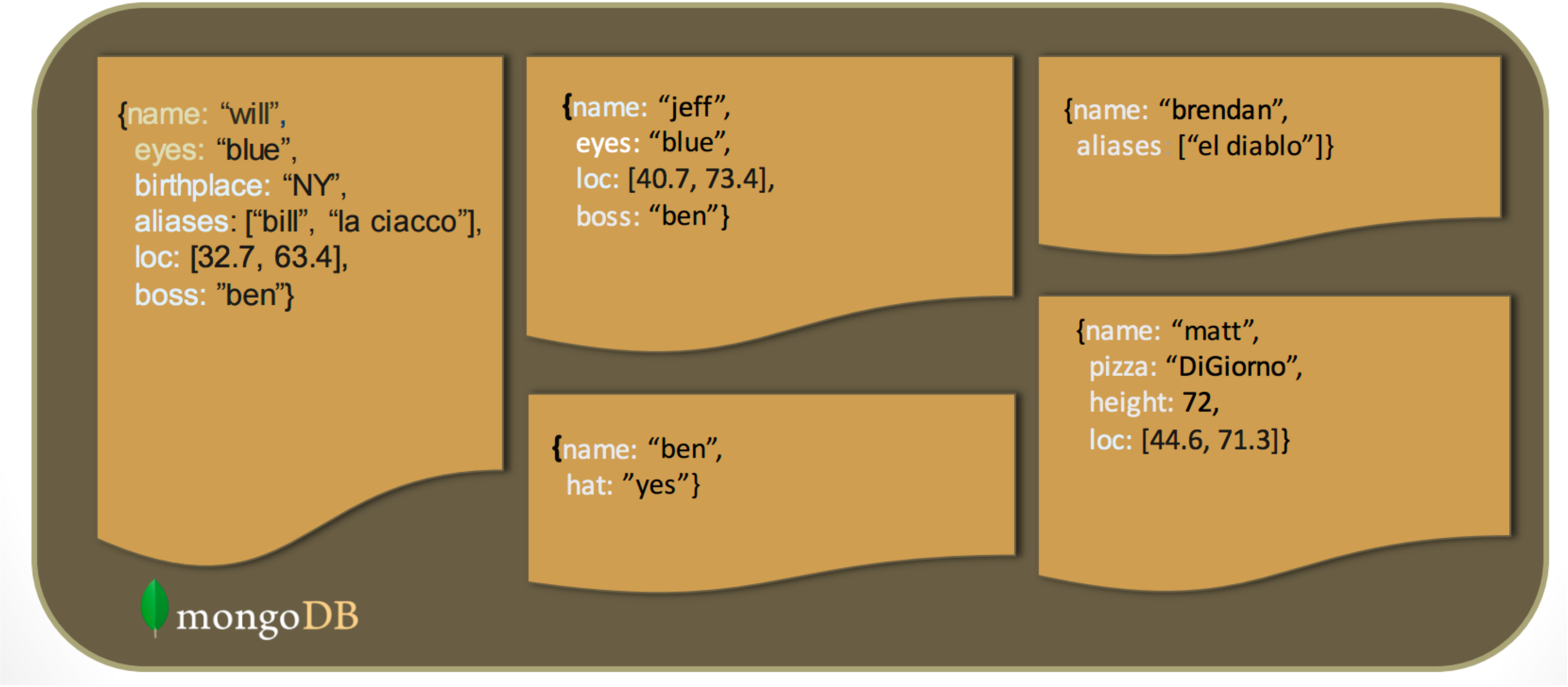
```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Embedded sub-document (contact)

Embedded sub-document (access)

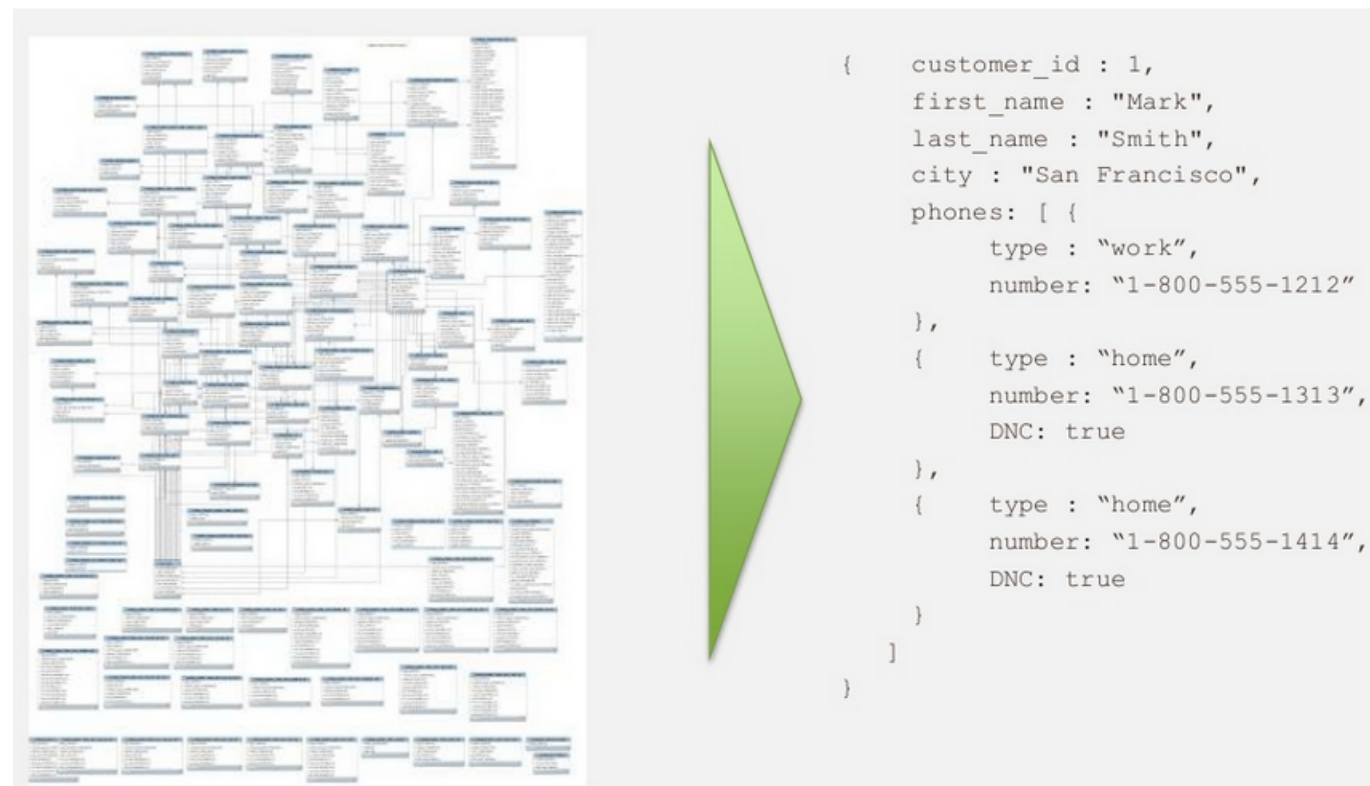
https://docs.mongodb.com/v3.2/core/data-modeling-introduction/

MongoDB: Collection



MongoDB vs RDBMS

- Collection vs table
- Document vs row
- Field vs column
- Schema-less vs Schema-oriented

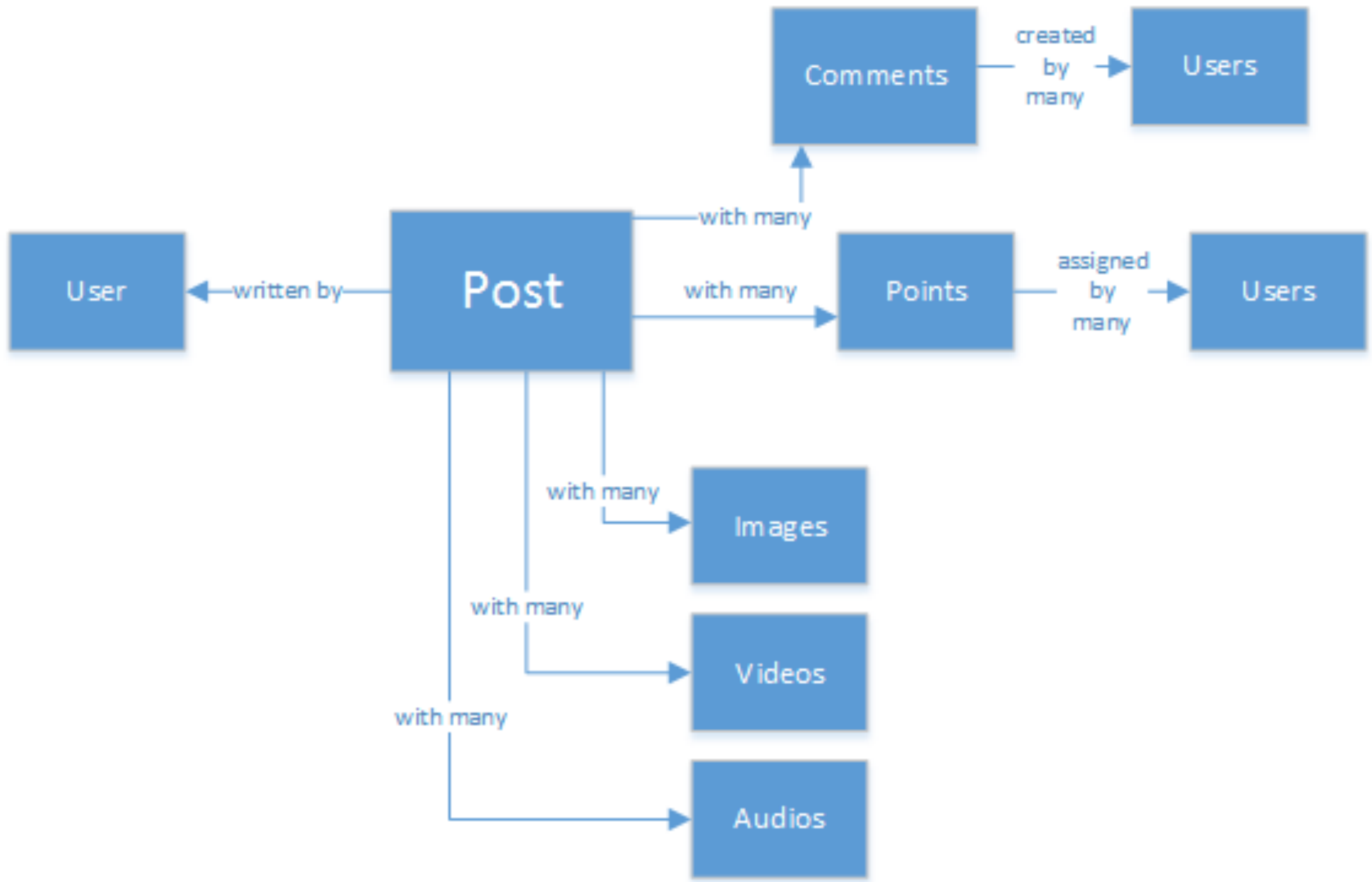


http://s3.amazonaws.com/info-mongodb-com/_com_assets/media/sql-v-mongodb-1.png

Example: Facebook++

- Users can create posts and add pictures, videos and music to them
- Other users can comment on posts and give points (likes) to rate posts
- Landing page has a feed of posts that users can share and interact with
- How would you design this in SQL?

Example: Facebook++ in SQL



What happens when I need to display a single post and all the information related to it?

https://docs.microsoft.com/en-us/azure/documentdb/documentdb-nosql-vs-sql

Facebook++: MongoDB “schema”

```
post = {
    “author”: “Joyce Ho”,
    “title”: “Everybody should take CS 377”,
    “images”: [“http://smileyface.png",
            http://exclamationpt.png"],
    “comments”:[
        {“Alice”: “Your class is too much work!”},
        {“Bob”: “ACID is not as cool as you think”}
    ]
}
```

Example: Insert Documents

```
db.inventory.insertMany([
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" },
status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" },
status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" },
status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" },
status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" },
status: "A" }
]);
```

https://docs.mongodb.com/manual/tutorial/query-documents/

MongoDB: Query

Query	SQL	Mongo
Select all documents	SELECT * FROM inventory	db.inventory.find({})
Equality condition	SELECT * FROM inventory WHERE status = “D”	db.inventory.find({status: “D”})
Or condition	SELECT * FROM inventory WHERE status = "A" OR qty < 30	db.inventory.find({ $or: [{ status: "A" }, { qty: { $lt: 30 } }] })

https://docs.mongodb.com/manual/tutorial/query-documents/

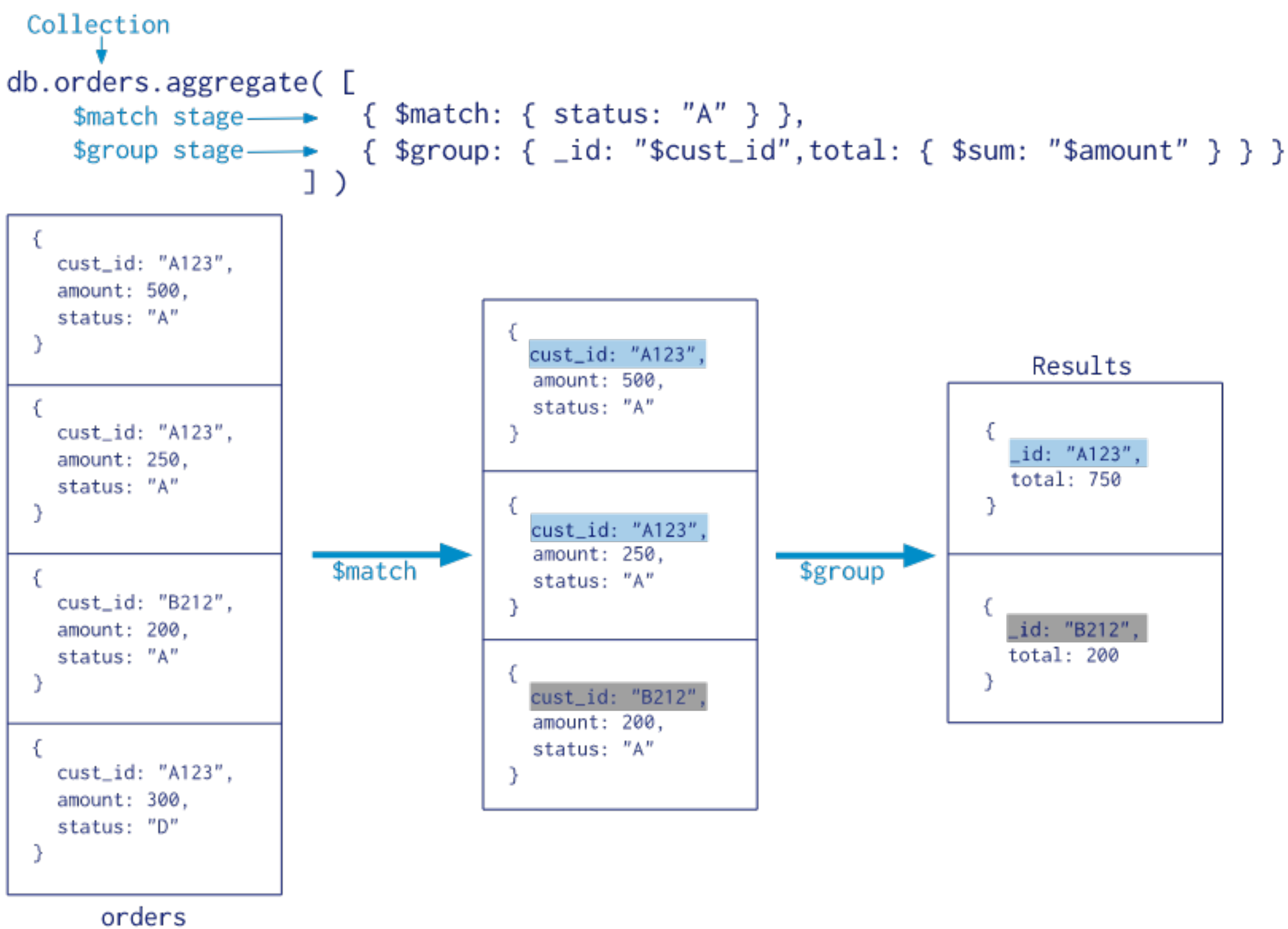
MongoDB: Benefits

- Embedded objects brought back in the same query as the parent object
    - No need to join 8 tables to retrieve content for a single post
- Document model matches your domain well, it can be much easier to comprehend than figuring out nasty joins
- Keeps functionality that works well in RDBMS such as ad-hoc queries and indexes

MongoDB: Aggregation

- Aggregation framework provides SQL-like aggregation functionality
  - Documents from a collection pass through aggregation pipeline which transforms objects as they pass through
  - Output documents based on calculations performed on input documents

MongoDB: Aggregation



https://docs.mongodb.com/manual/aggregation/

MongoDB: Functionality

- Map reduce functionality to perform complex aggregator functions given a collection of key, value pairs
- Indexes to match the query conditions and return the results using only the index (B-tree index)

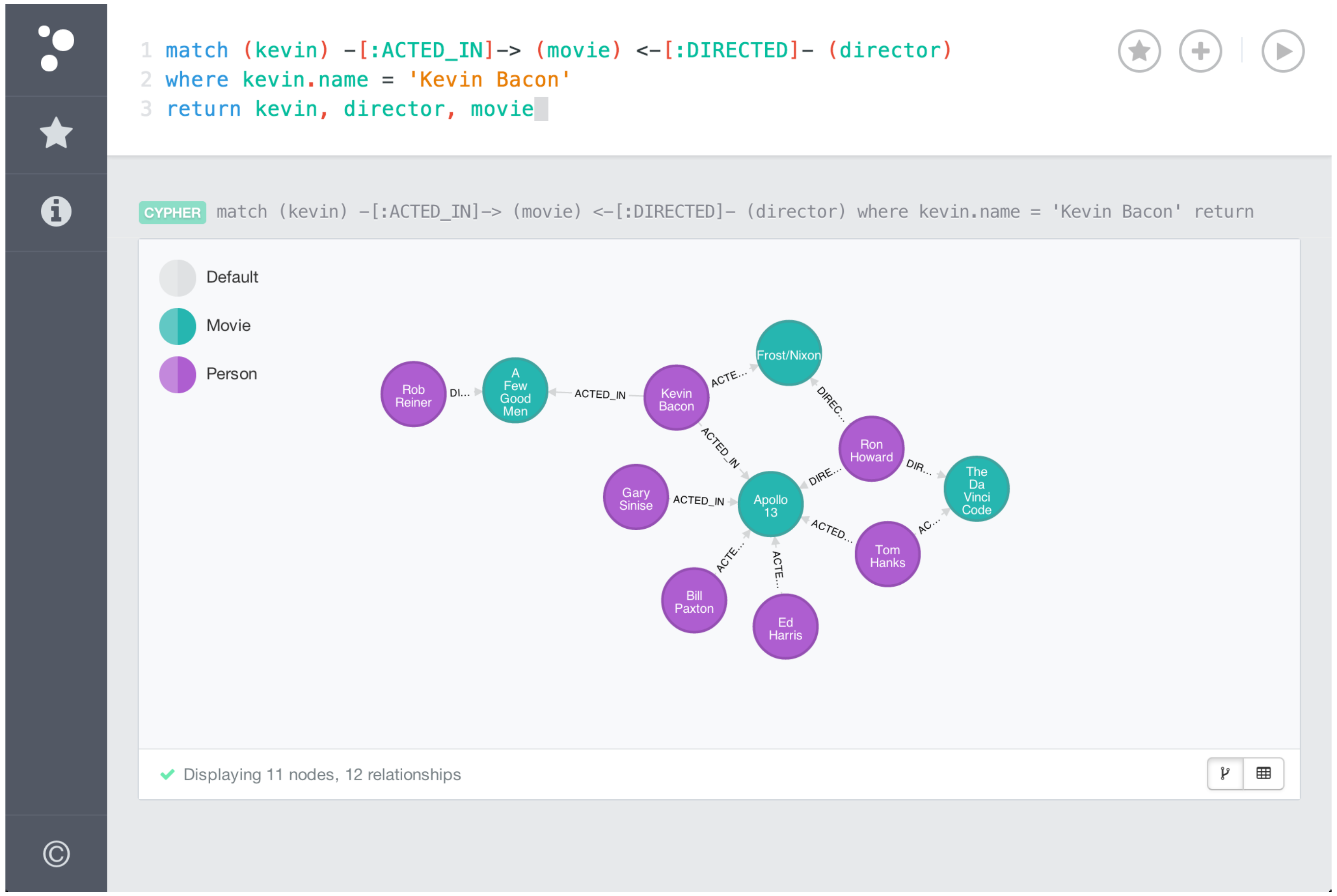
MongoDB: Pitfalls

- Query can only access a single collection
    - Joins of documents are not supported
- Long running multi-row transactions are not distributed well
- Atomicity is only provided for operations on a single document
    - Group together items that need to be updated together

Graph Database

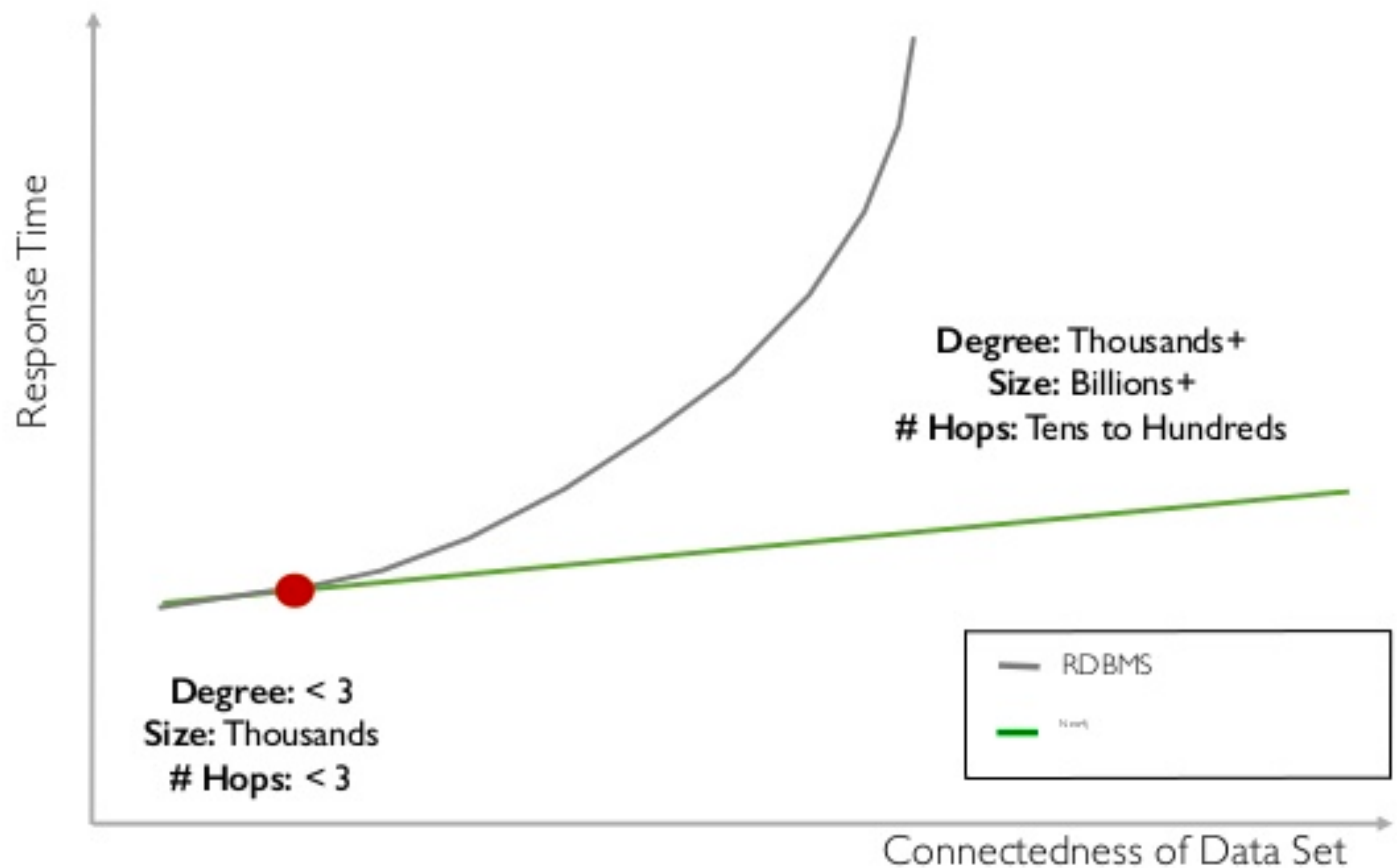
- Collection of vertices (nodes) and edges (relations) and their properties

- Example: AllegroGraph, VertexDB, Neo4j



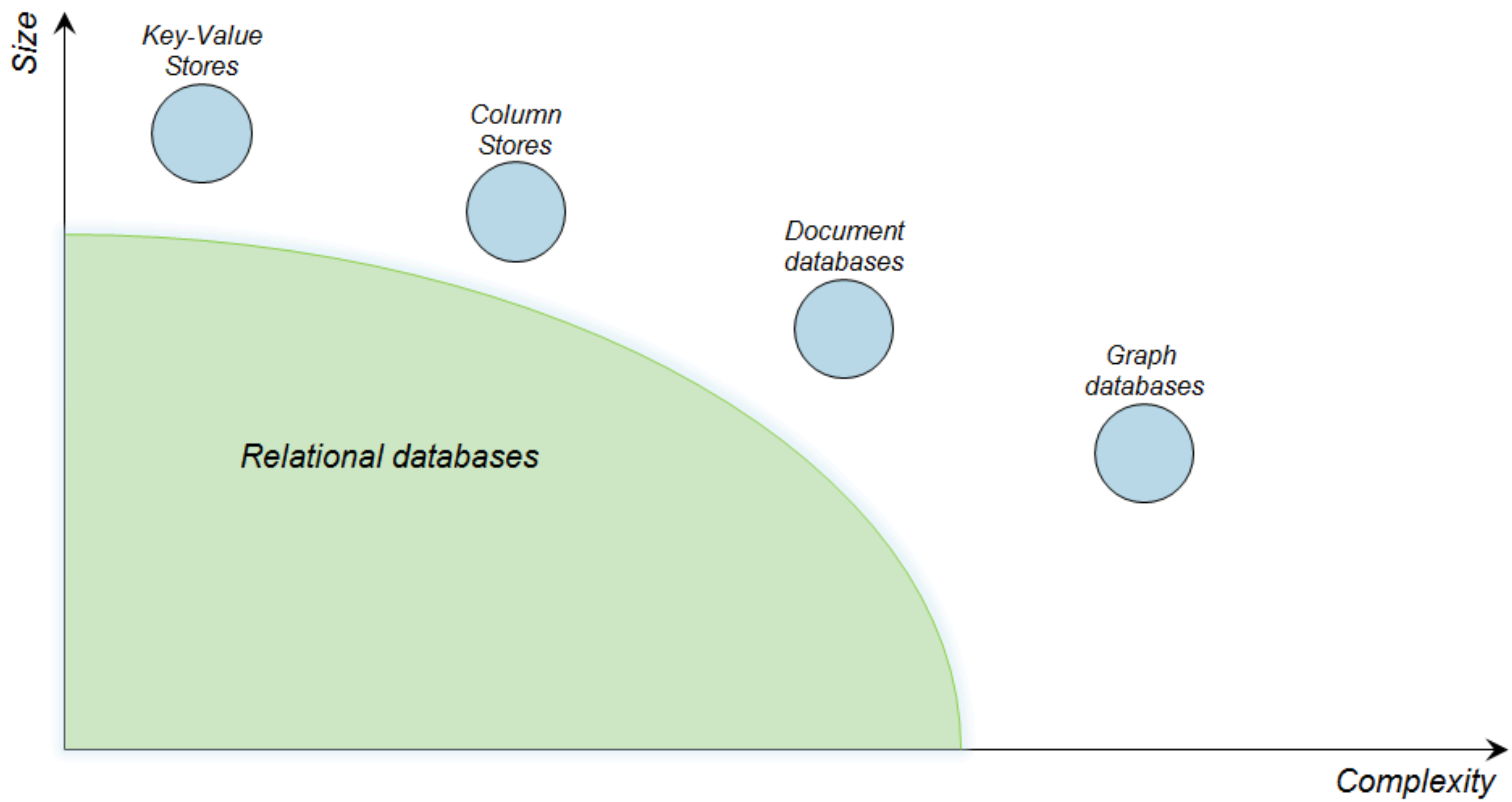
http://www.apcjones.com/talks/2014-03-26_Neo4j_London/images/neo4j_browser.png

RDBMS vs Native Graph Database



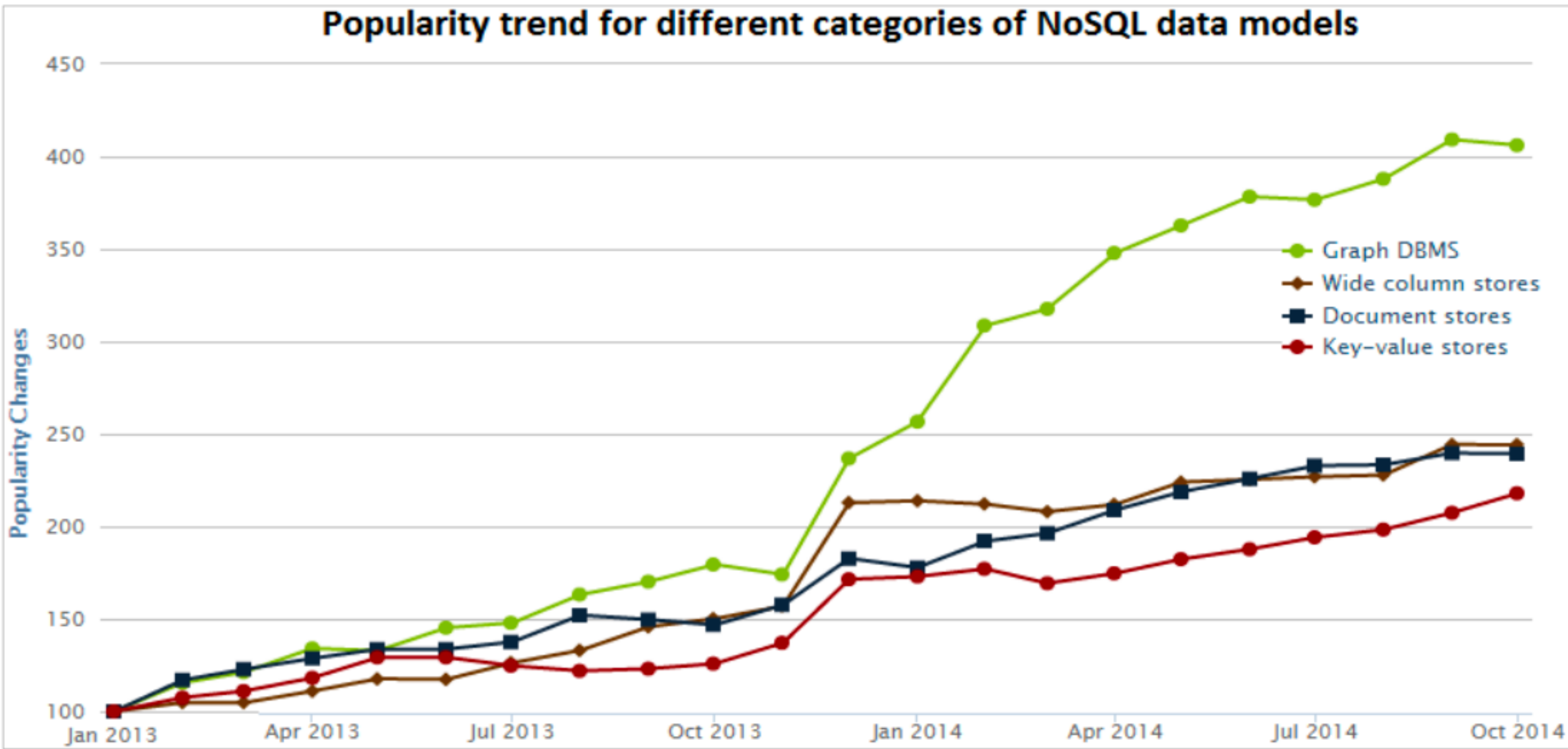
http://www.slideshare.net/maxdemarzi/graph-database-use-cases

Focus of Different Categories



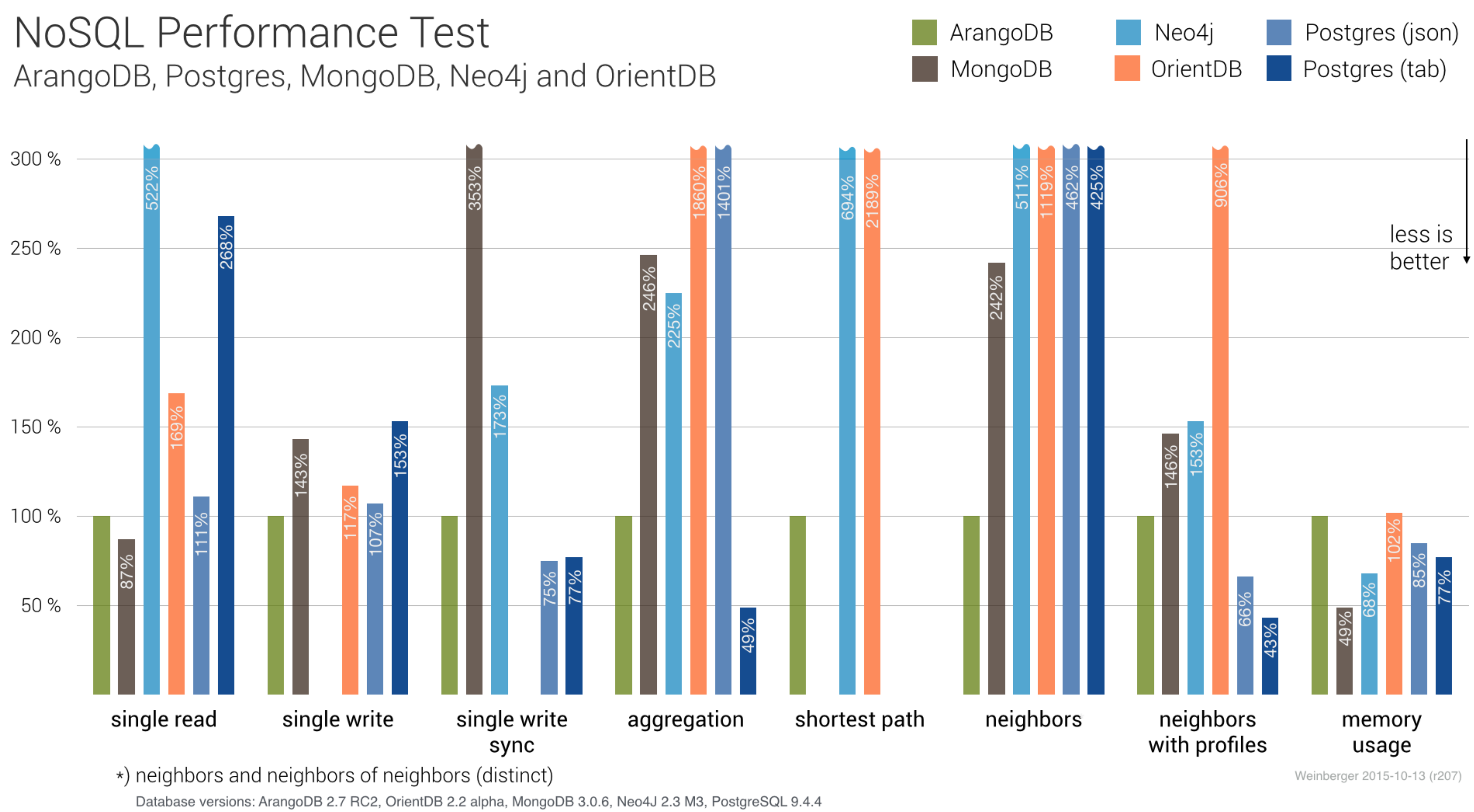
https://techietrack.wordpress.com/2015/02/06/nosql-database-types/

Popularity of Different Categories



http://web.cs.iastate.edu/~sugamsha/articles/Classification%20and%20Comparison%20of%20Leading%20NoSQL%20Big%20Data%20Models%2009%2022%202014.pdf1

NoSQL Performance Test



https://www.arangodb.com/wp-content/uploads/2015/09/chart_v2071.png

NoSQL: Comparison

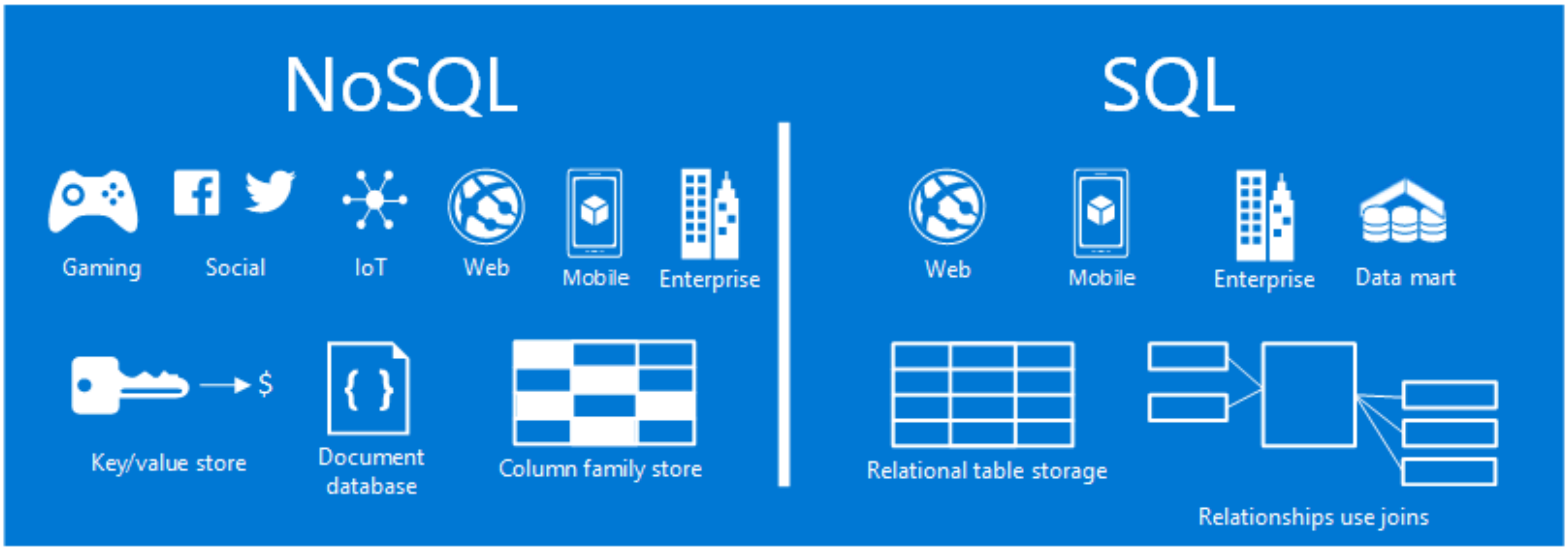
TYPES OF NON-RELATIONAL DATABASES

upwork™

TYPES	PERFORMANCE	SCALABILITY	FLEXIBILITY	COMPLEXITY
KEY-VALUE STORE	high	high	high	none
COLUMN STORE	high	high	moderate	low
DOCUMENT	high	variable (high)	high	low
GRAPH DATABASE	variable	variable	high	high

https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/

NoSQL vs SQL



https://docs.microsoft.com/en-us/azure/documentdb/documentdb-nosql-vs-sql

NoSQL: Use Cases

- Bigness: big data, big number of users, big number of computers, …
- Massive write performance: high volume to fit on a single node
- Fast key-value access: lower latency
- Flexible schema & datatypes: complex objects can be easily stored without a lot of mapping
- No single point of failure

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

NoSQL: Use Cases

- Generally available parallel computing
- Easier maintainability, administration, and operations
- Programmer ease of use: accessing data is intuitive for developers
- Right data model for the right problem: graph problem should be solved via a graph database
- Distributed systems support: designed to operate in distributed scenarios

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

NoSQL Challenges

- Lack of maturity — numerous solutions still in their beta stage
- Lack of commercial support for enterprise users — many are still open source projects
- Lack of support for data analysis and business intelligence
- Maintenance efforts and skills are required
- Experts are hard to find (although becoming more prevalent these days)

Jumping on NoSQL Bandwagon?

- Data model and query support
  - Do you want/need the power of something like SQL?
  - Do you want/need fixed or flexible schemas
- Scale
  - Do you want/need massive scalability?
  - Are you willing to sacrifice replica consistency?

Jumping on NoSQL Bandwagon?

- Agility and growth
  - Are you building a service that could grow exponentially?
  - Are you optimizing for quick, simple coding or maintainability?

NoSQL: Recap

- Motivation for NoSQL
- CAP theorem
- ACID vs BASE
- NoSQL categories
- Use cases and challenges