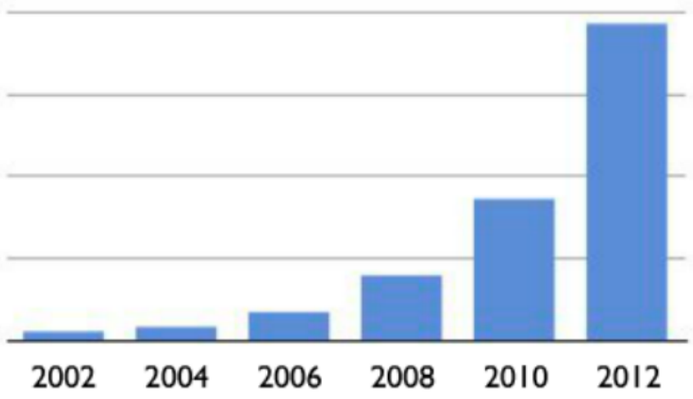
NoSQL Introduction

CS 377: Database Systems

Recap: Data Never Sleeps

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

Web 2.0

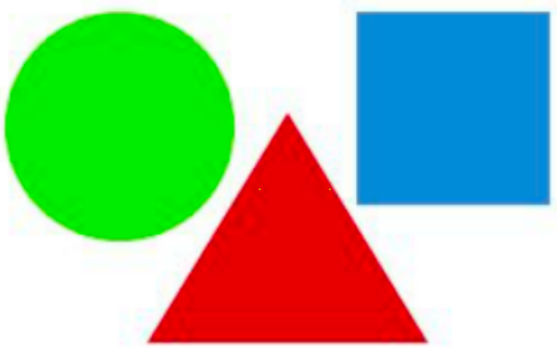


Big data

Connectivity

P2P Knowledge

Concurrency



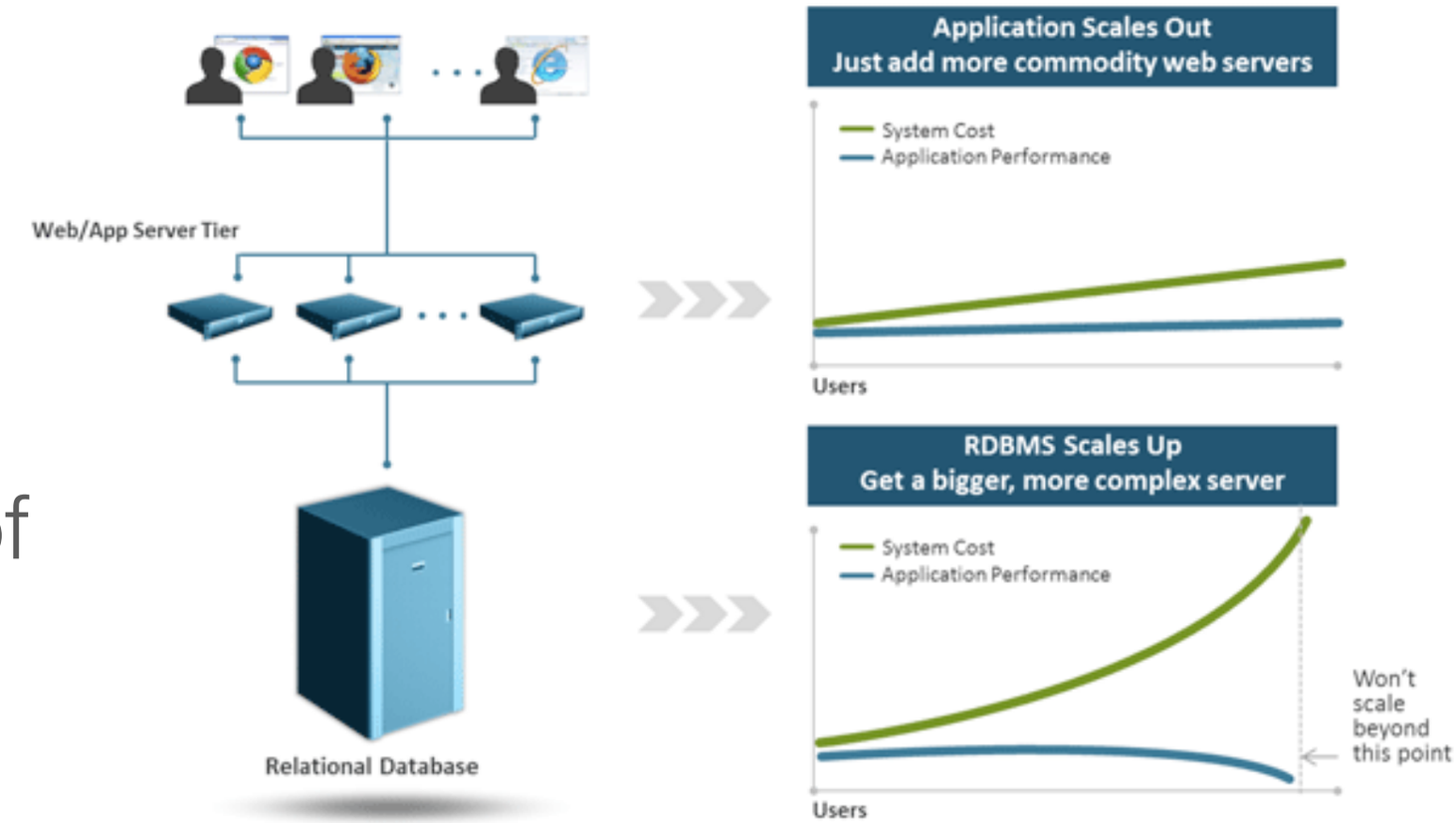
Diversity

Cloud-Grid

Lorenzo Alberton Talk, “NoSQL Databases: Why, what and when”

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

Motivation for NoSQL

- Users do both updates and reads and scaling transactions to parallel or distributed DBMS is hard

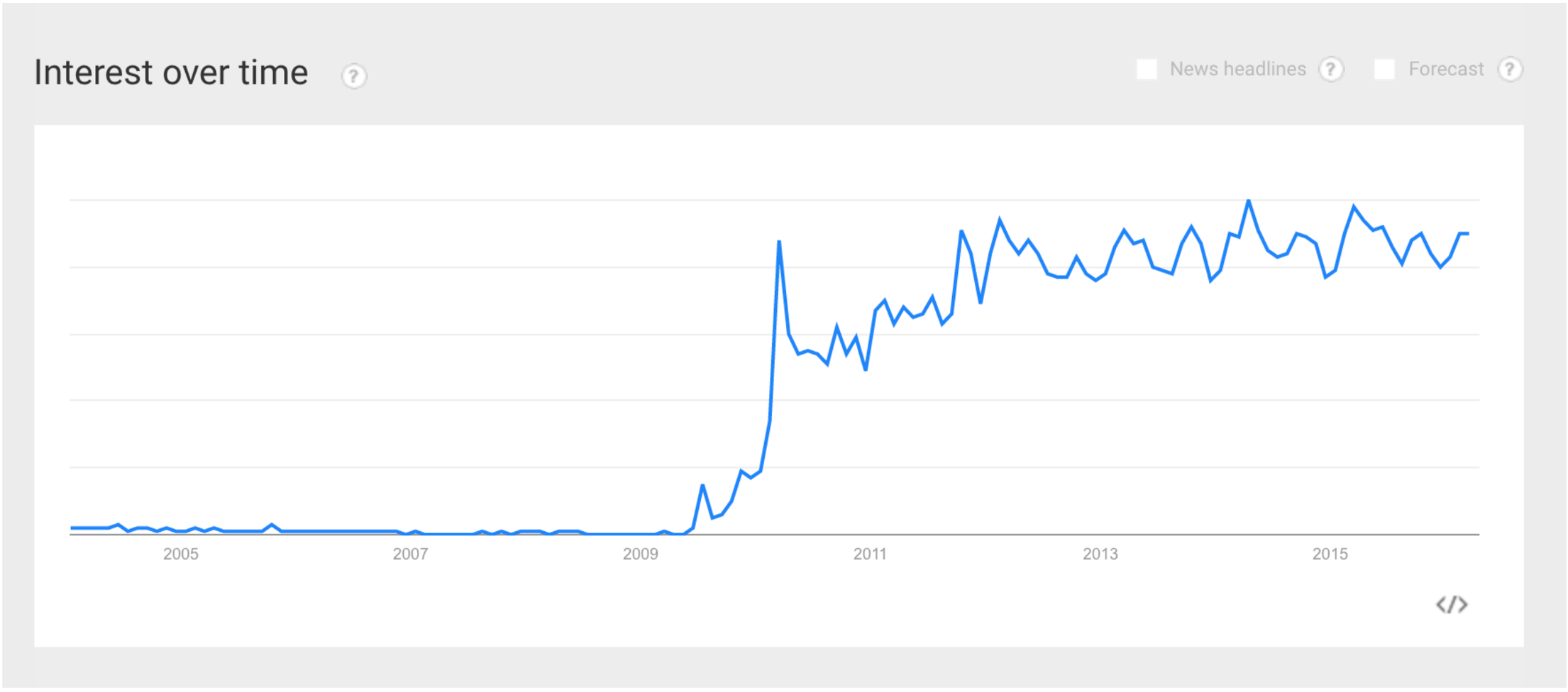
- Large servers are too expensive with maximum capacity

- Load can increase rapidly with web traffic and unpredictability

- Google and Amazon developed their own alternative approaches, BigTable and DynamoDB respectively

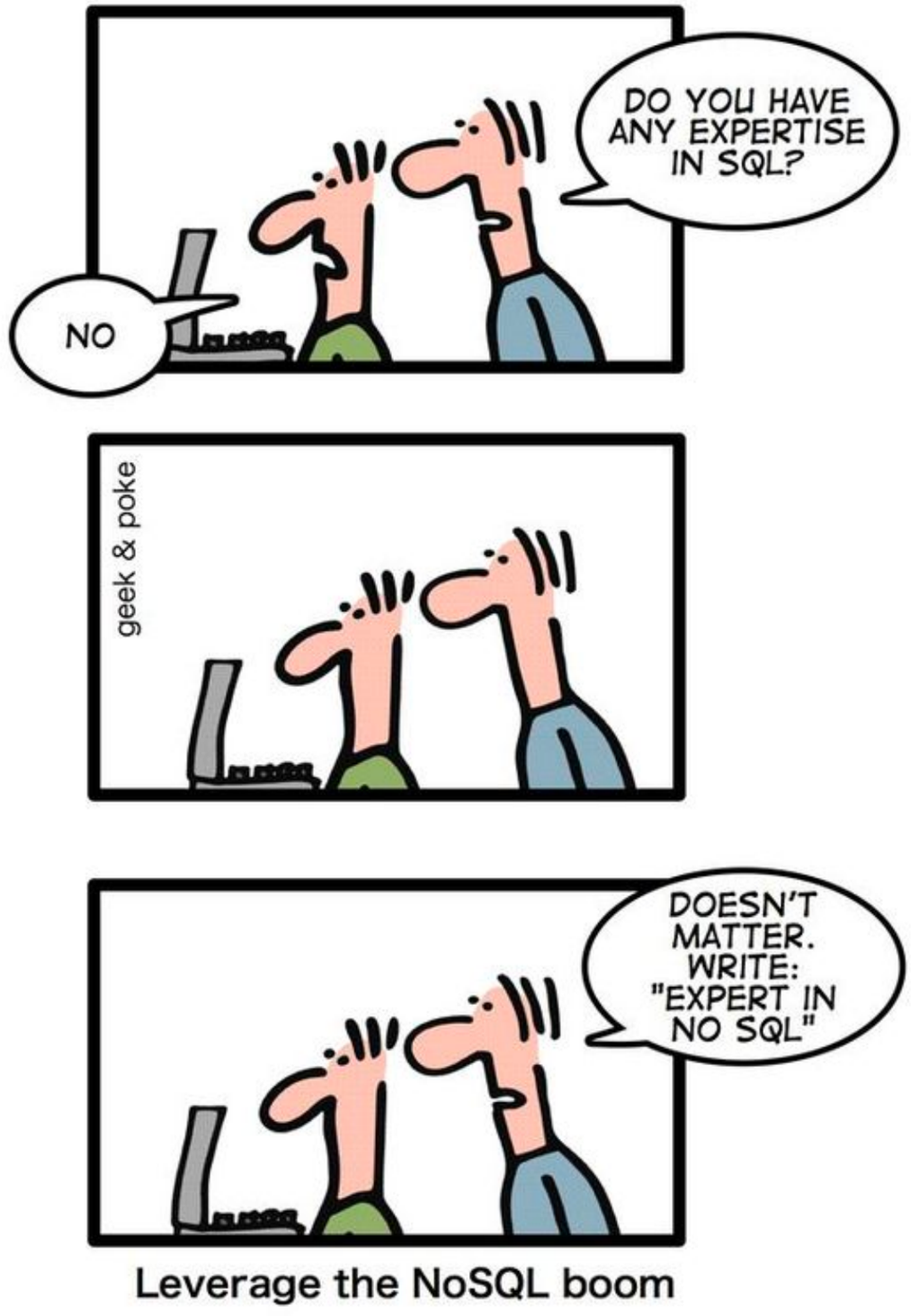
NoSQL: New Hipster

NoSQL: New Hipster (2)



http://www.google.com/trends/explore#q=NoSQL

HOW TO WRITE A CV



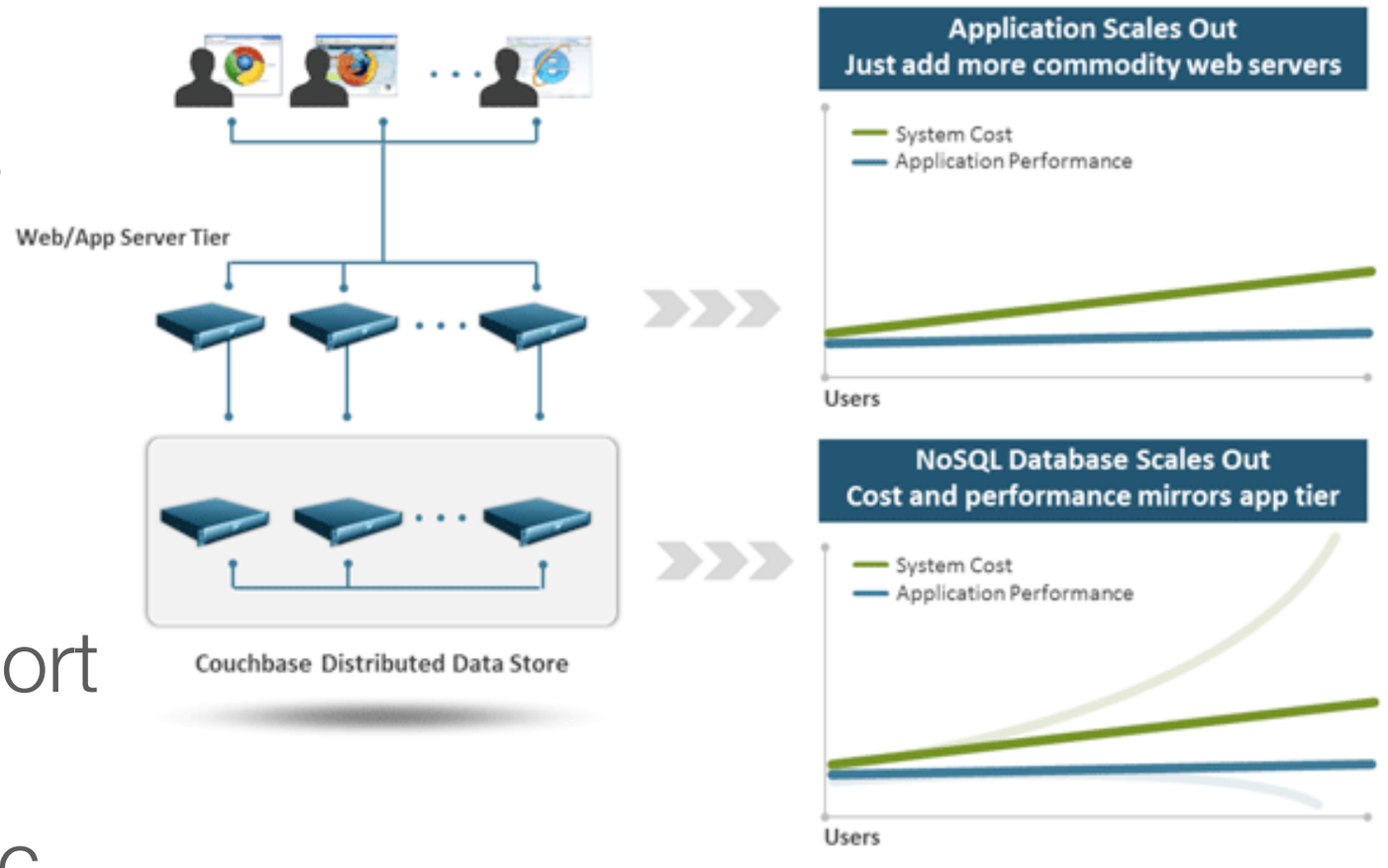
http://geekandpoke.typepad.com/geekandpoke/2011/01/nosql.html

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

Recap: ACID

- Atomicity: all or nothing
- Consistency: any transaction takes database from one consistent state to another
- Isolation: execution of one transaction is not impacted by other transactions executing at the same time
- Durability: persistence of the transactions (recover against system failures)

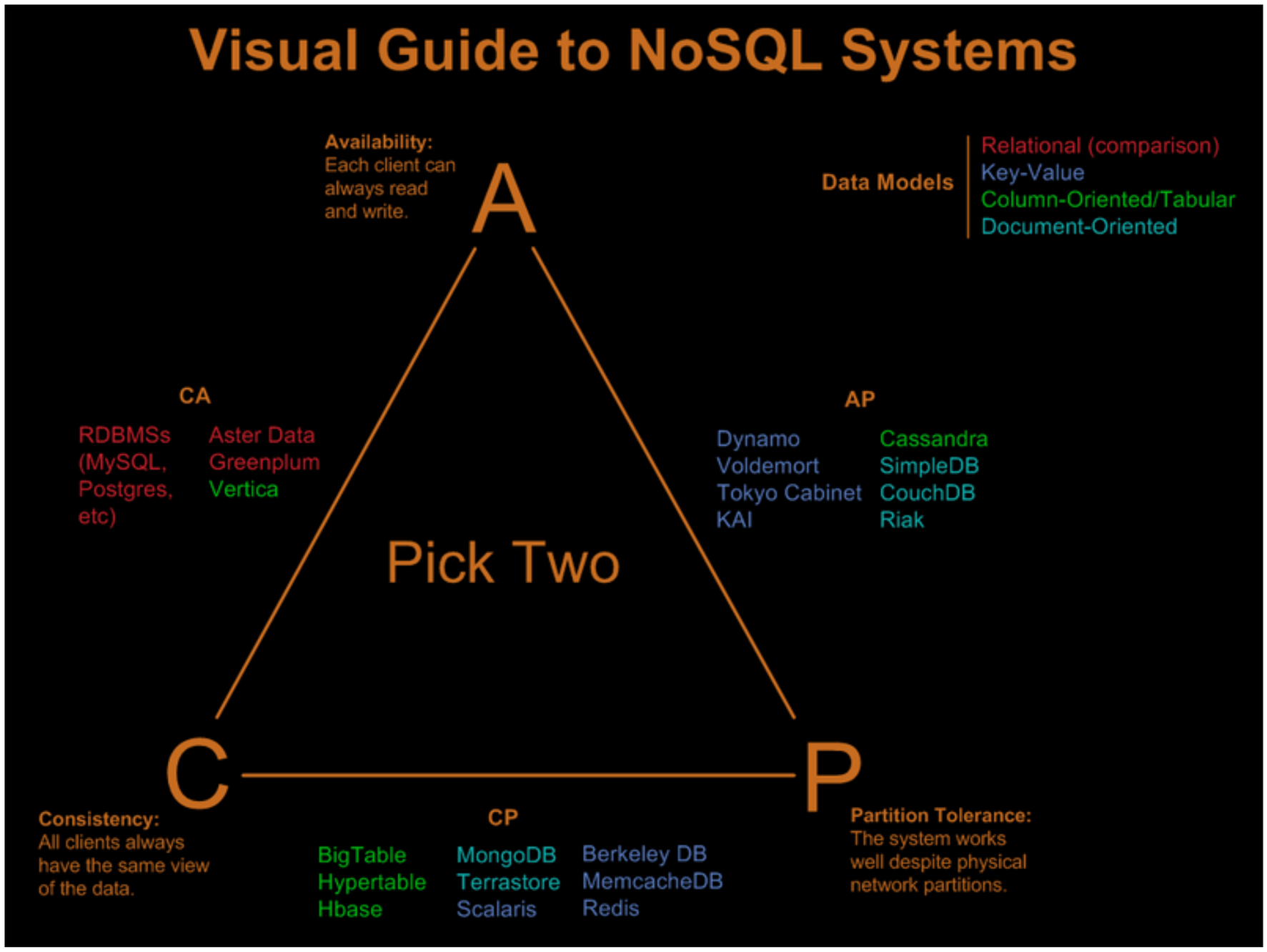
But, pitfalls of DBMS with regards to latency, partition tolerance, and high availability!

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

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
- Some debate whether graph databases is truly NoSQL
- Categories can be subject to change in the future

Key-Value Store

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

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

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

Key-Value Store: Voldemort

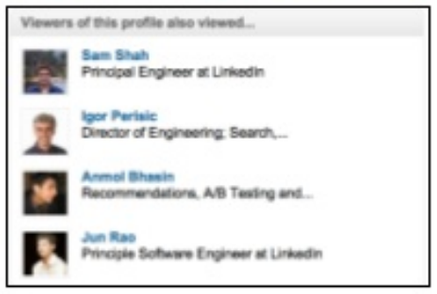
- Distributed data store used by LinkedIn for high-scalability storage

- Named after fictional Harry Potter villain

- Addresses two usage patterns
  - Read-write store
  - Read-only store

People You May Know

Viewers of this profile also viewed

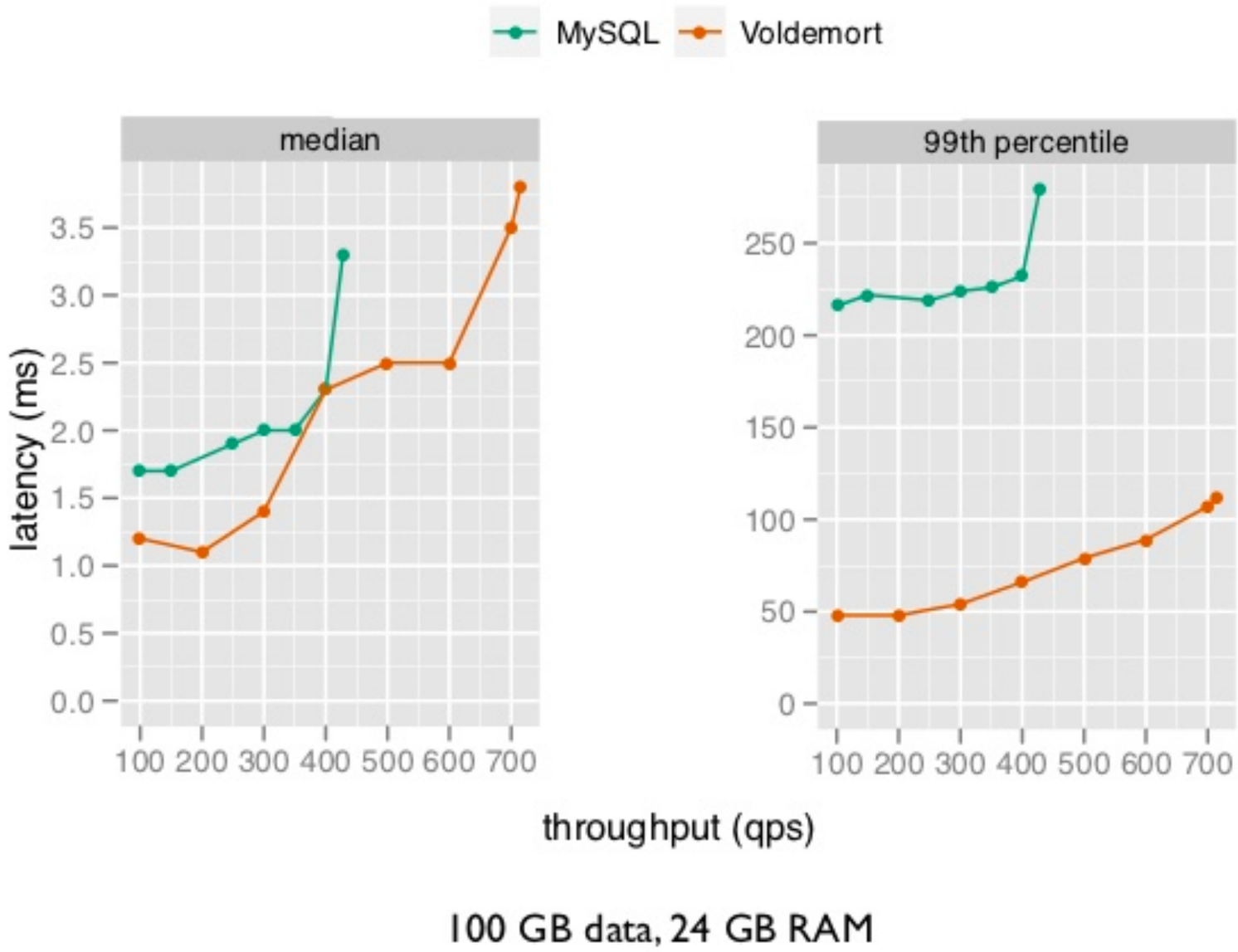


Events you may be interested in

LinkedIn Skills

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

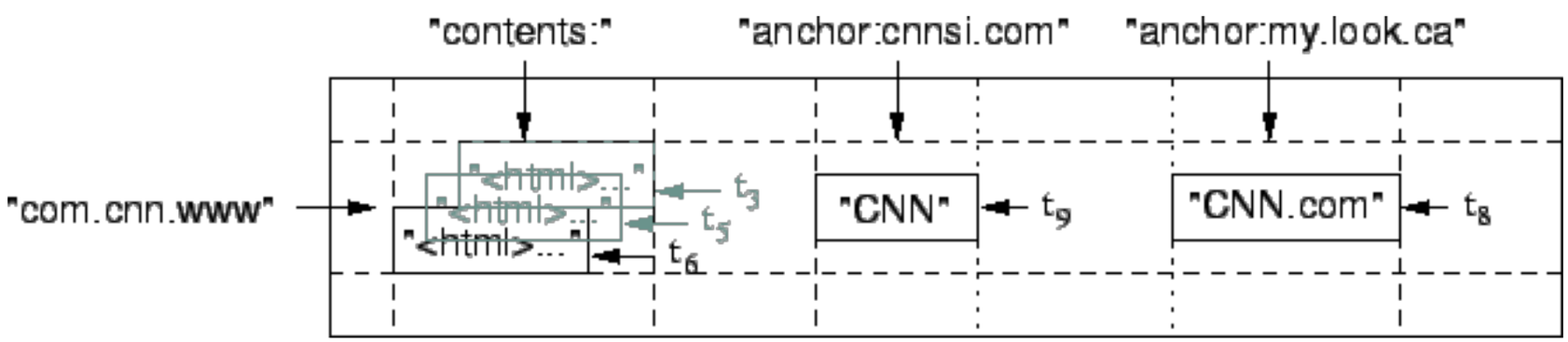
http://www.slideshare.net/r39132/linkedin-data-infrastructure-qcon-london-2012/25-Voldemort_RO_Store_Performance_TP

Column-Based Families

- Data is stored in a big table except you store columns of data together instead of rows

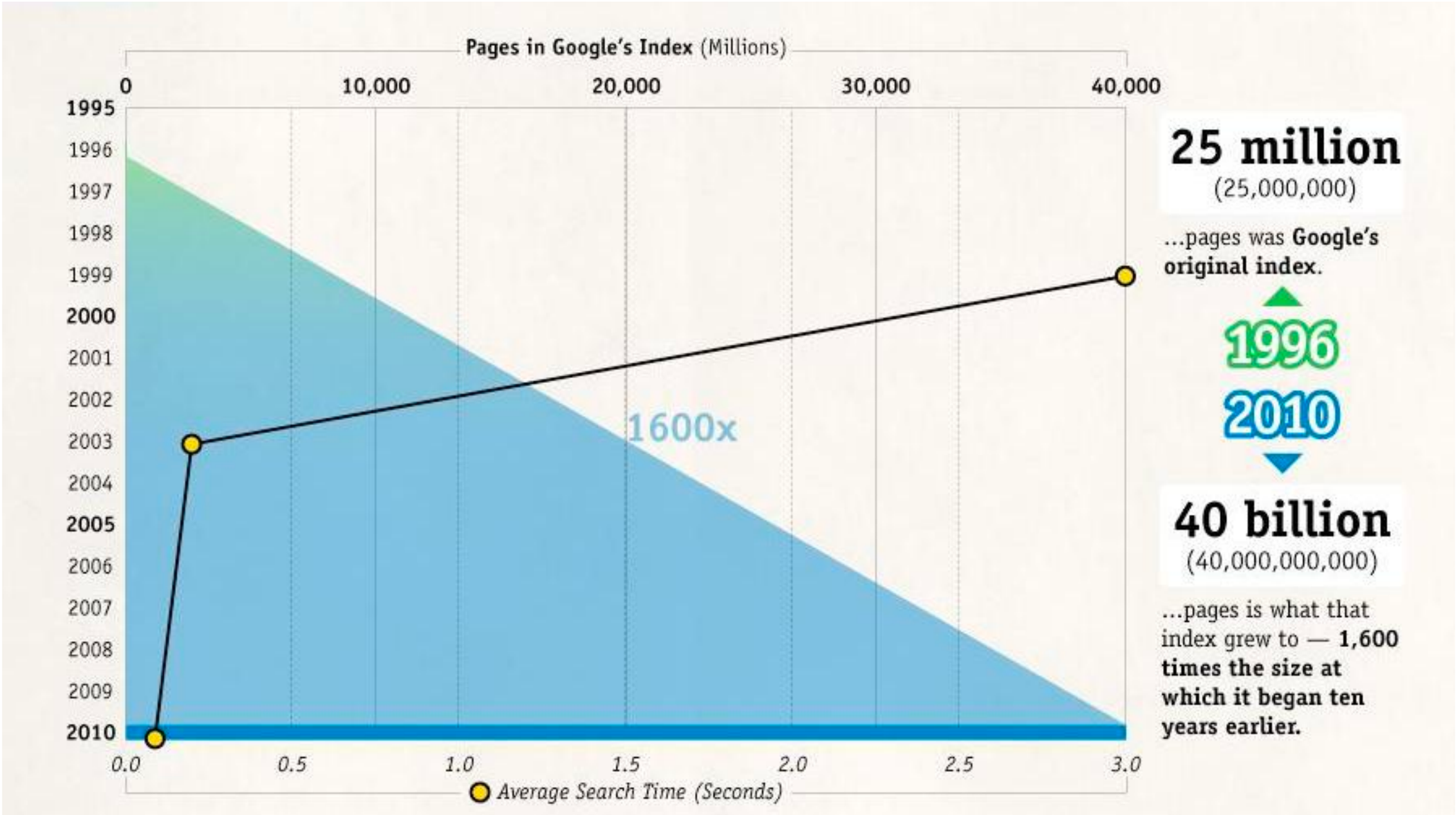
- Access control, disk and memory accounting performed on column families

- Example: HBase, Cassandra, Hypertable



https://www.usenix.org/legacy/events/osdi06/tech/chang/chang_html/img5.png

Column-Based Family: BigTable Performance



http://sandeepsamajdar.blogspot.com/2011/08/bigtable-google-database.html

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

JavaScript Object Notation (JSON)

- Alternative data model for semistructured data
- Built on two key structures
  - Object is a sequence of fields (name, value pairs)
  - Array of values
- A value can be
  - Atomic value (e.g., string)
  - Object
  - Array

```
{
    "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

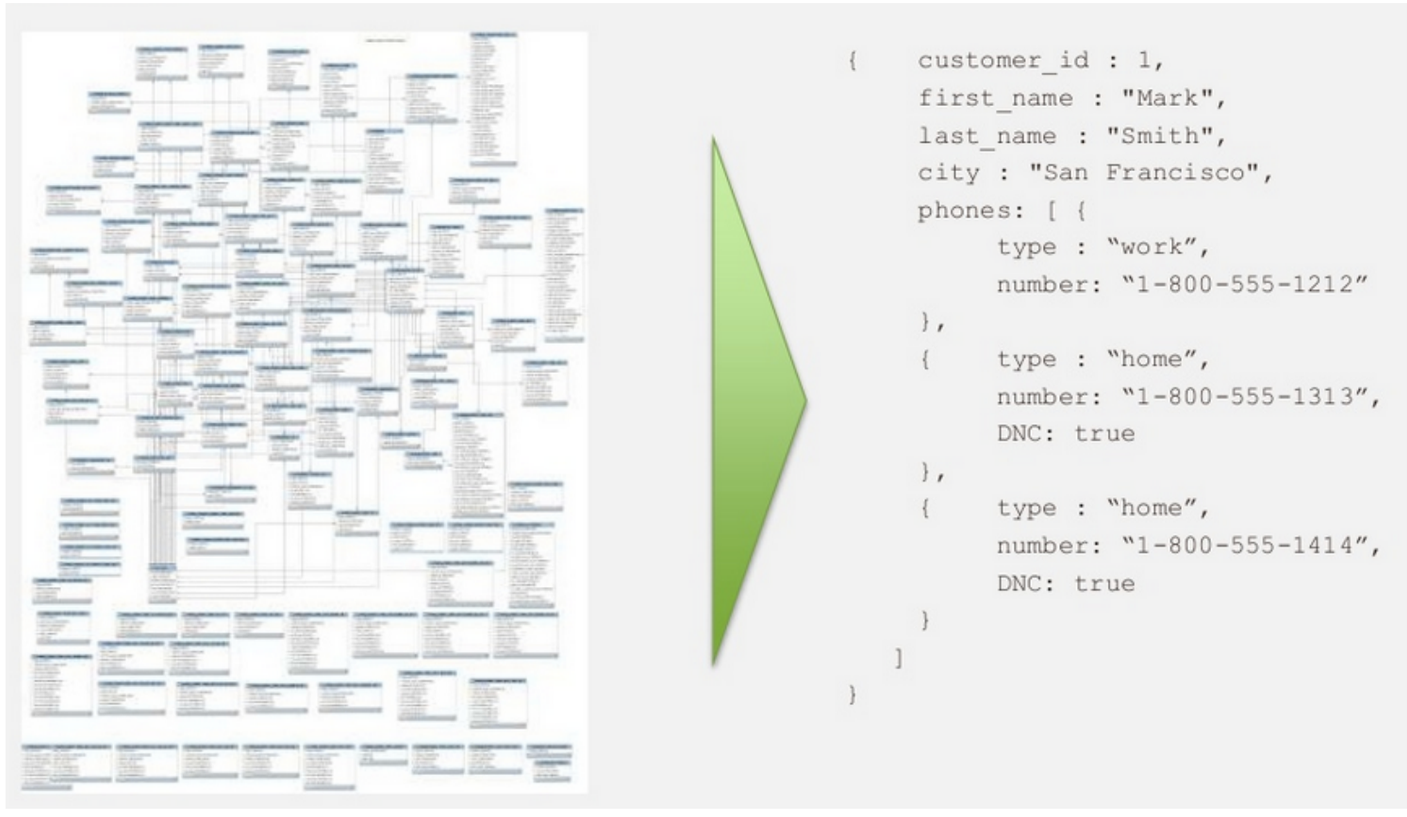
Document Database: MongoDB

- 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

https://docs.mongodb.org/manual/_images/crud-annotated-document.png

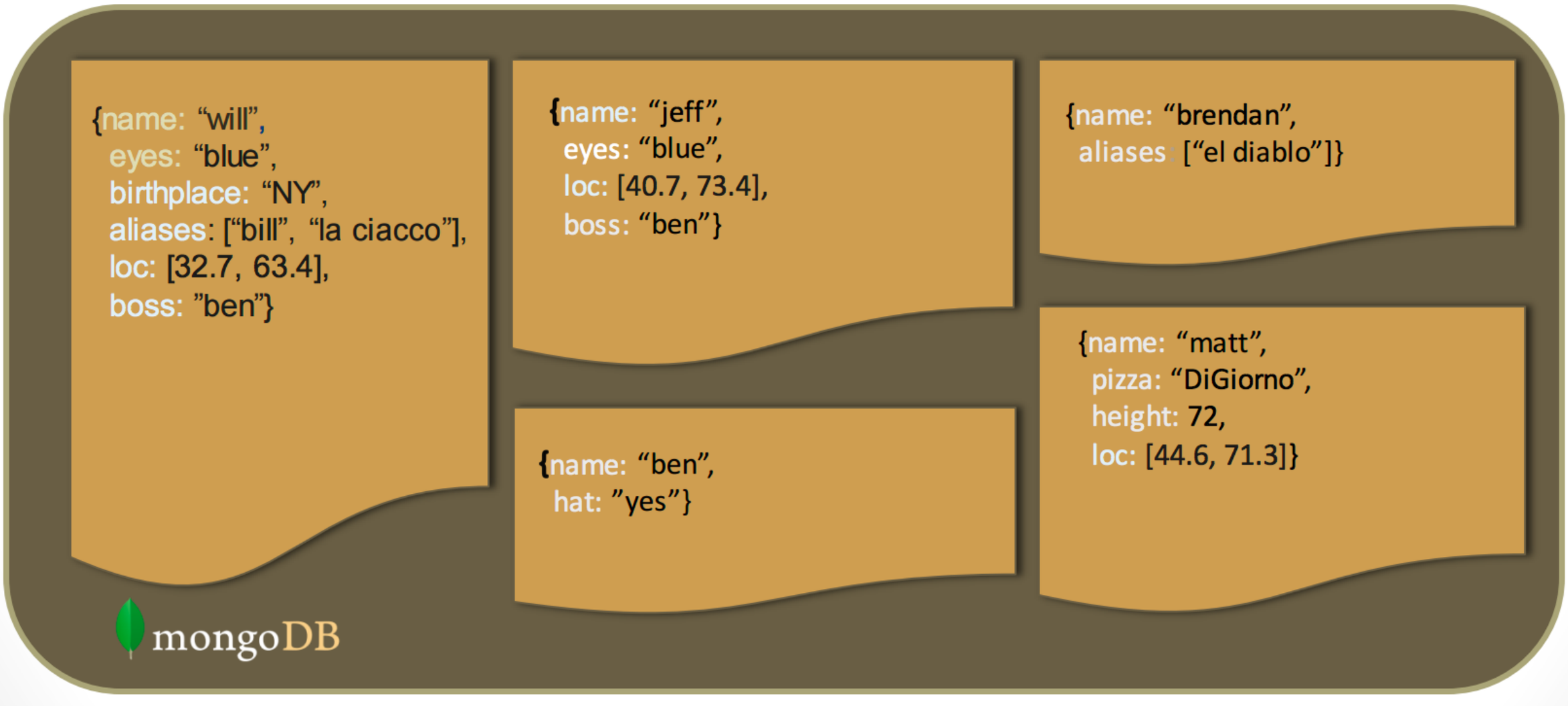
MongoDB vs Relational DBMS

- 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: MongoDB Collection



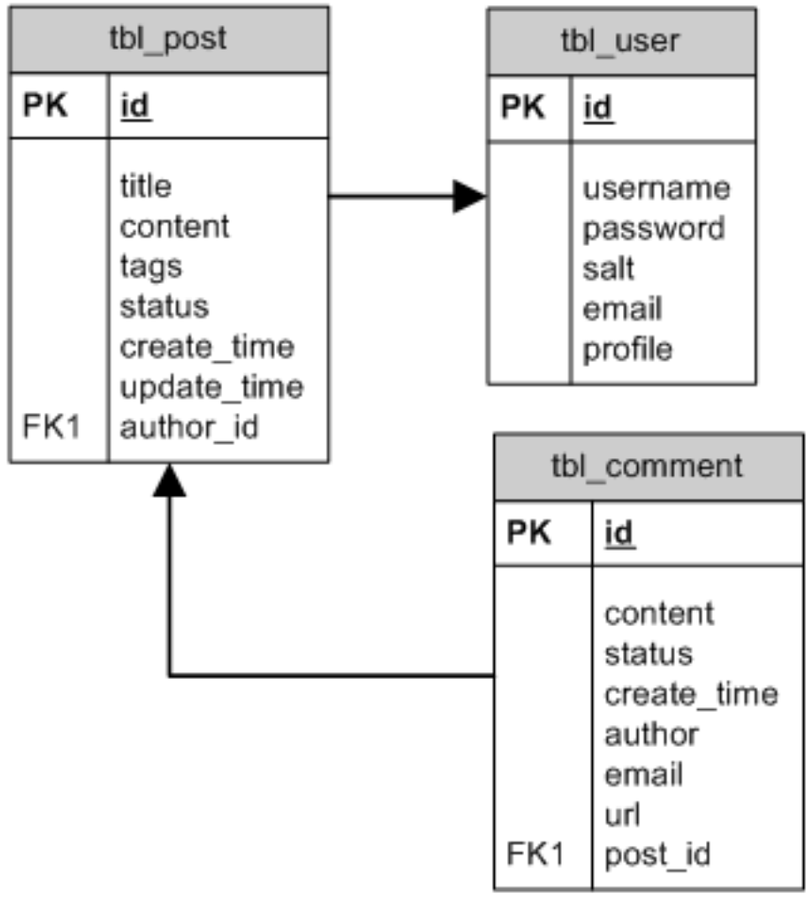
Example: Blog

- A blog post has an author, some text, and many comments

- Comments are unique per post, and one author can have many posts

- How would you design this in SQL?

Blog: Relational Database Diagram



http://www.yiiframework.com/doc/blog/1.1/en/start.design

Blog: MongoDB “schema”

- Collection for posts
- Embed comments & author name

```
post = {
    author: ‘Joyce Ho’,
    text: ‘Database systems are awesome.’,
    comments:[
                ‘Your class is too much work!’,
                ‘ACID is not as cool as you think’
                ]
}
```

MongoDB Benefits

- Embedded objects brought back in the same query as the parent object
  - No need to join 3 tables to retrieve content for a single post
- Keeps functionality that works well in RDBMS
  - Ad hoc queries
  - Indexes (fully featured & secondary)
- Document model matches your domain well, it can be much easier to comprehend than figuring out nasty joins

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

MongoDB CRUD Operations

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)

MongoDB CRUD Operations (2)

- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

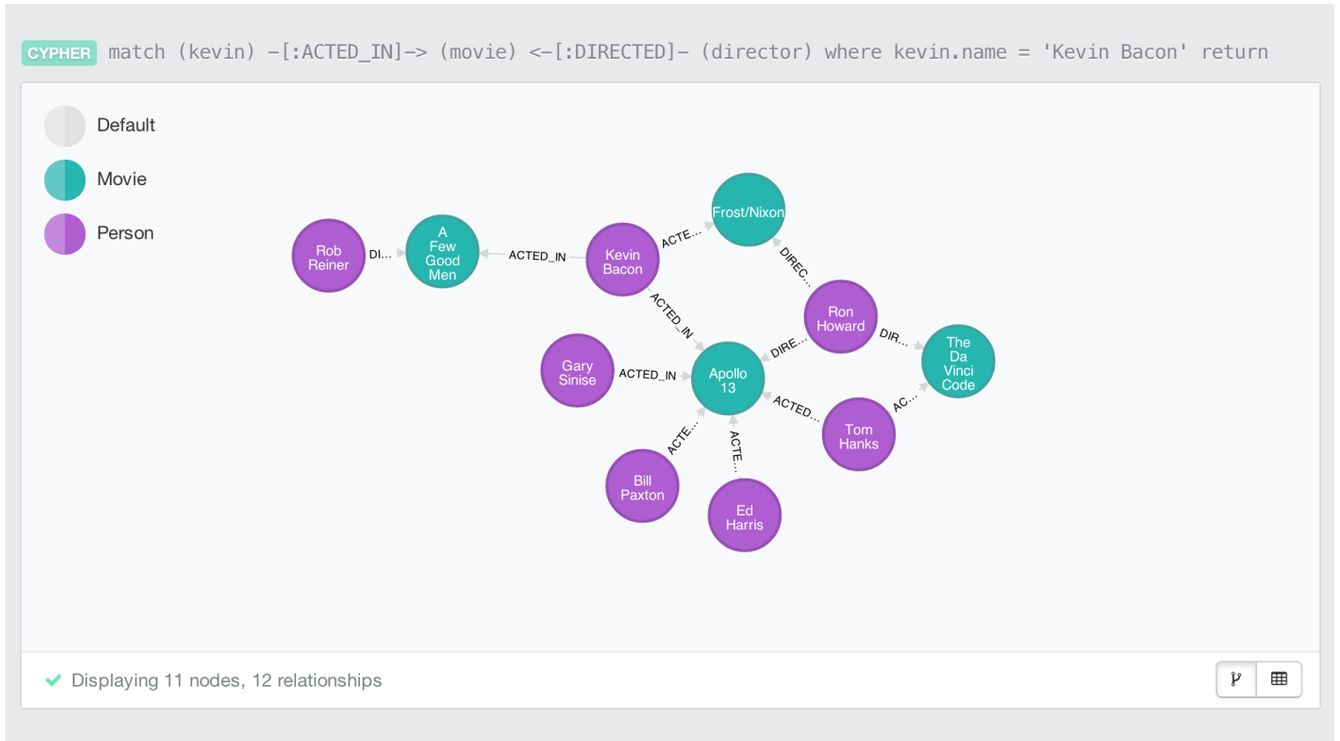
MongoDB Functionality

- 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
- 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)

Graph Database

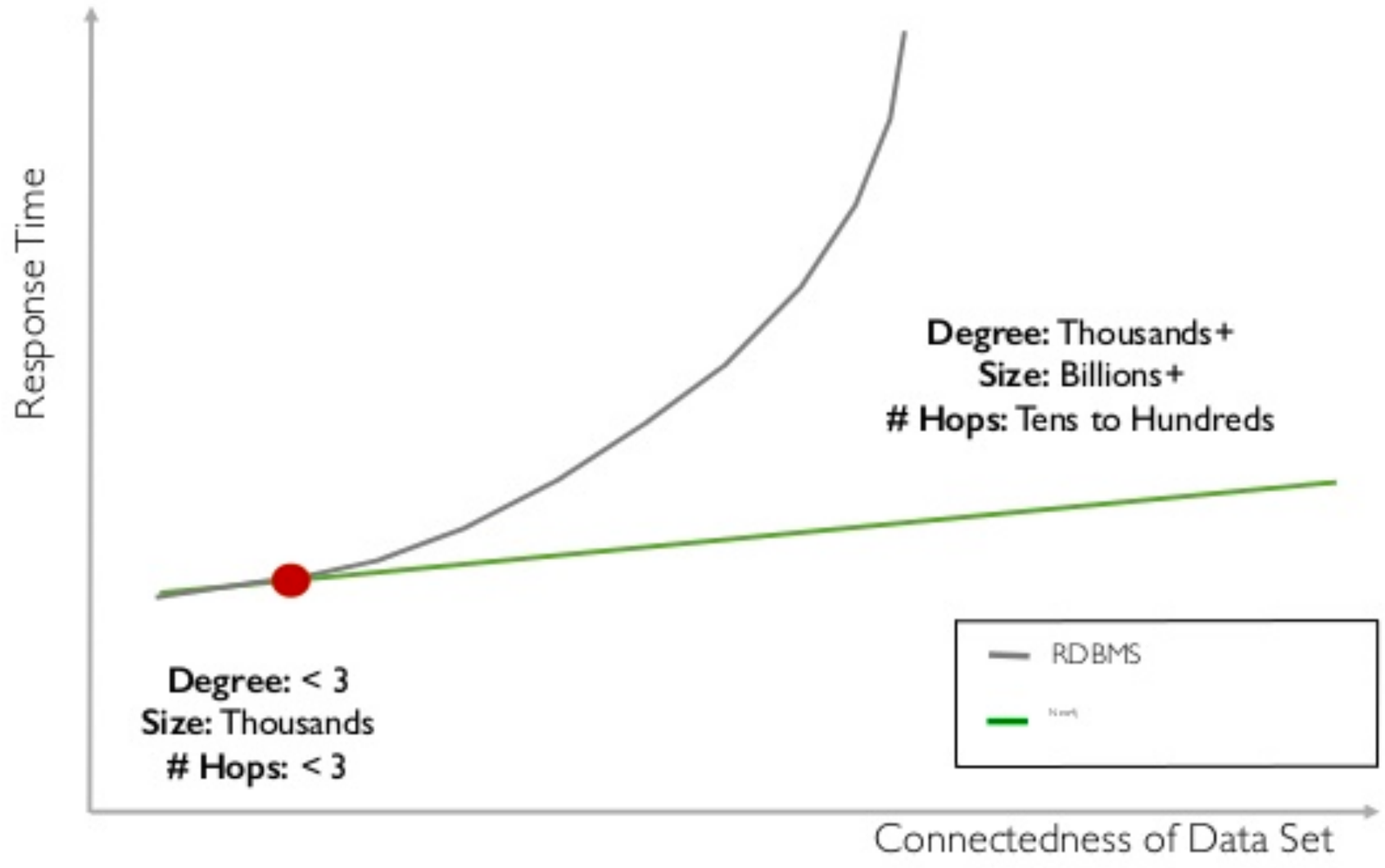
- Collection of vertices (nodes) and edges (relations) and their properties

- Example: AllegroGraph, VertexDB, Neo4j



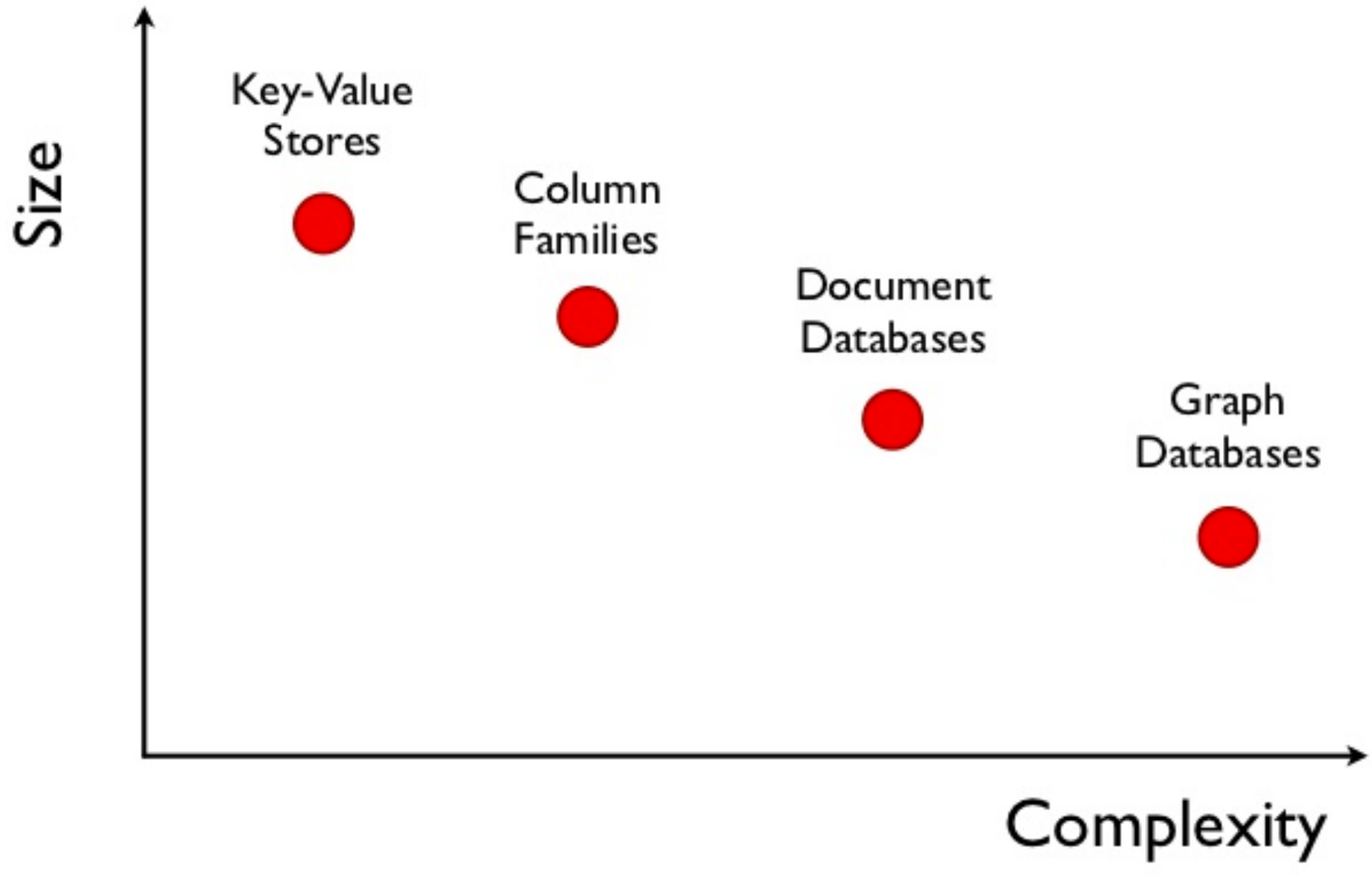
http://www.apcjones.com/talks/2014-03-26_Neo4j_London/images/neo4j_browser.png

RDBMS vs Native Graph Database



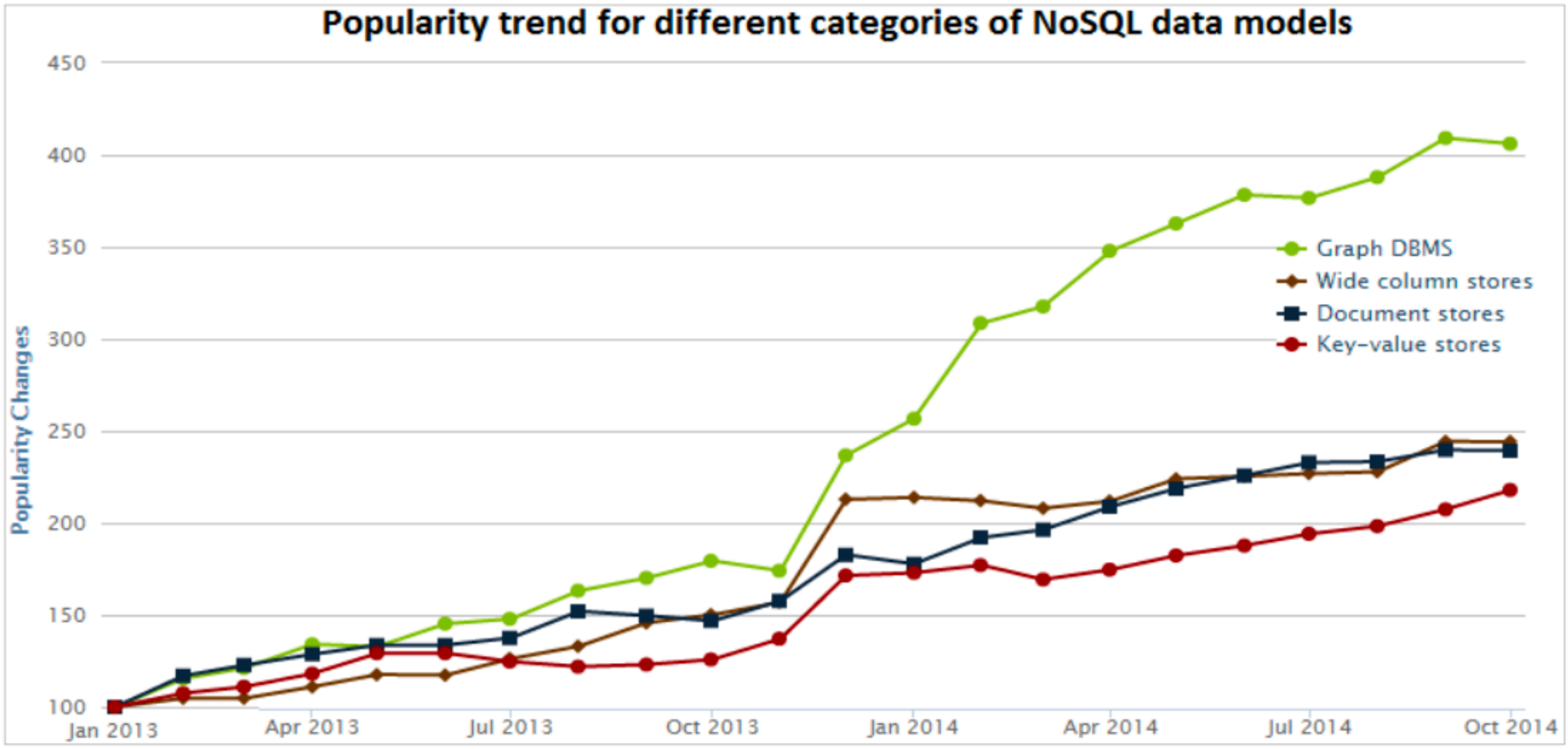
http://www.slideshare.net/maxdemarzi/graph-database-use-cases

Focus of Different Categories



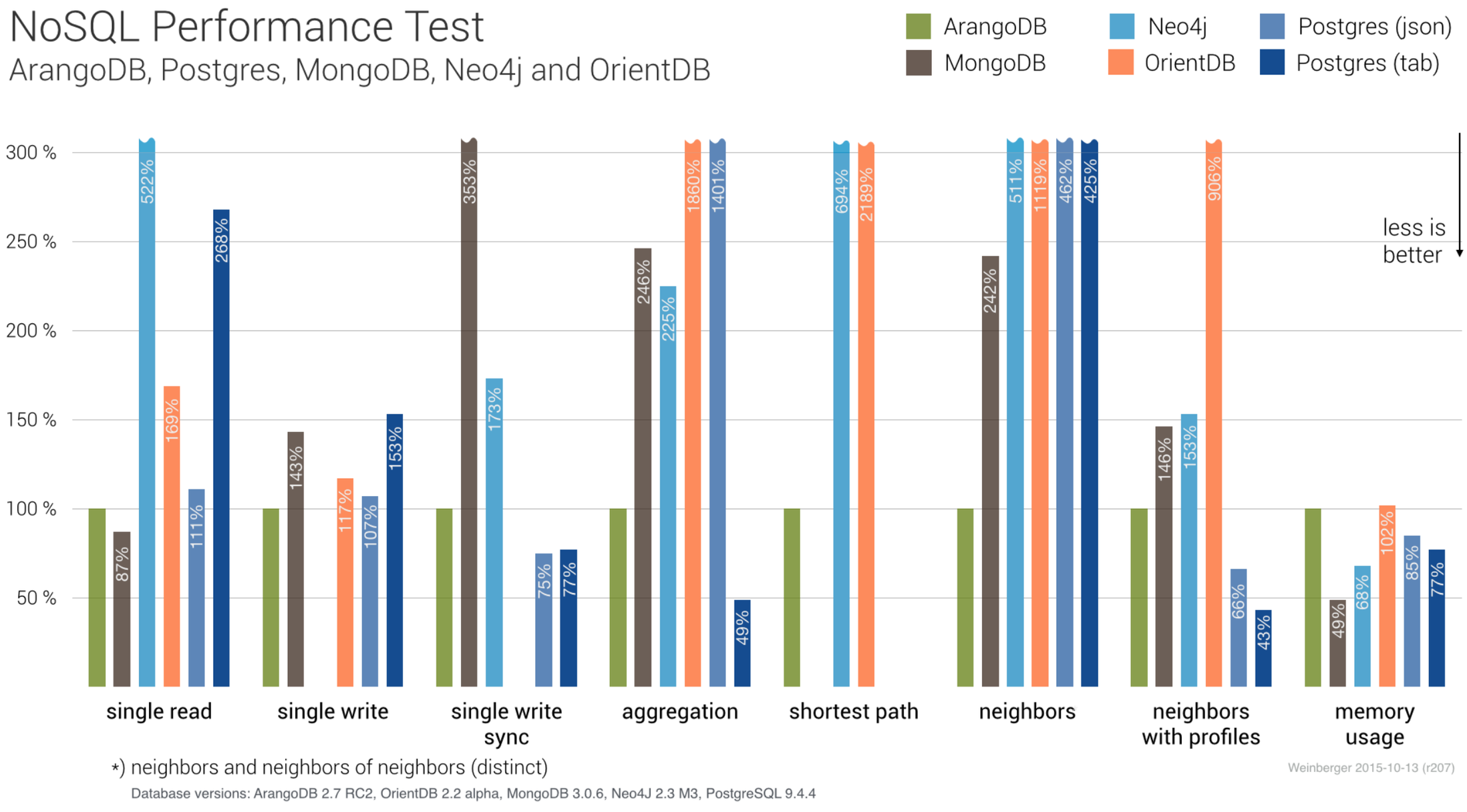
http://www.slideshare.net/emileifrem/nosql-east-a-nosql-overview-and-the-benefits-of-graph-databases

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 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 (2)

- 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? (2)

- 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