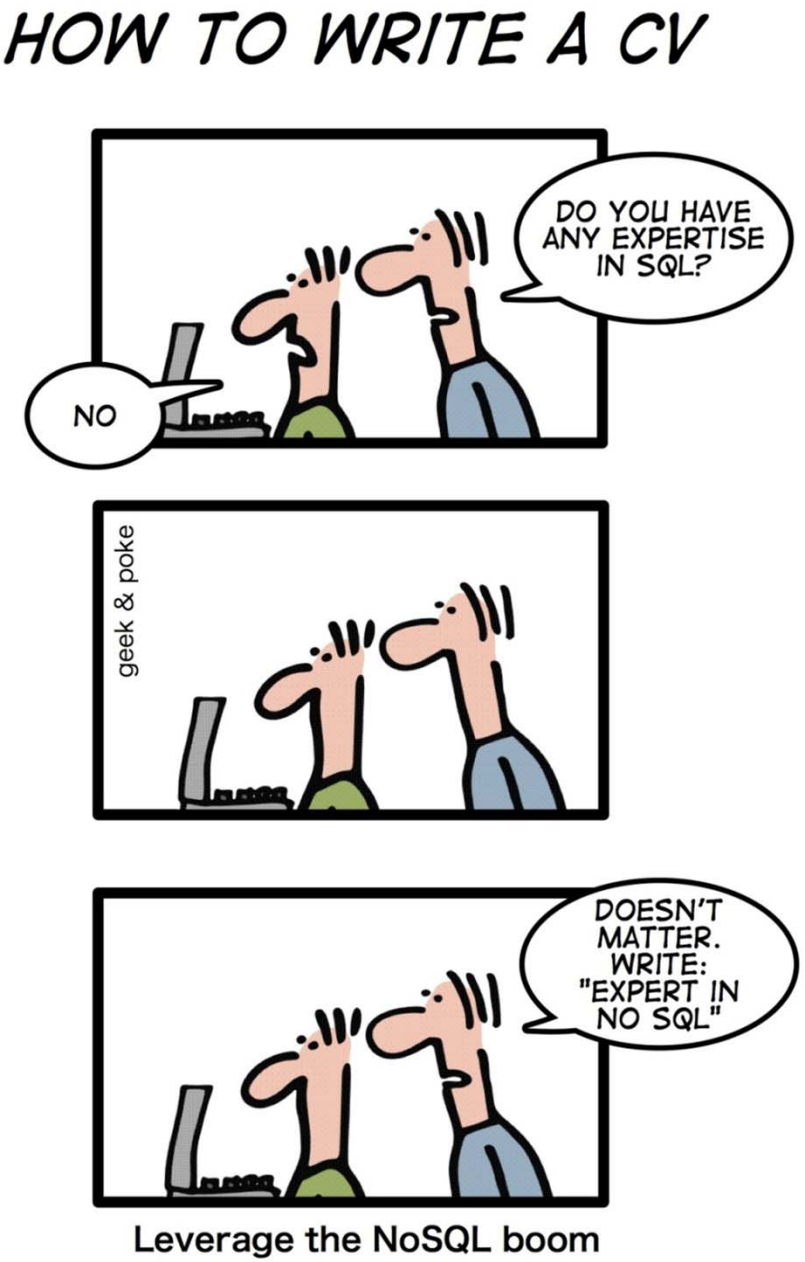
Introduction to NoSQL Databases

Slides based on multiple sources



Outline

- SQL Databases
  - SQL Standard
  - SQL Characteristics
- NoSQL Databases
  - NoSQL Defintion
  - General Characteristics
  - NoSQL Database Types
  - NoSQL Database Examples

What is in the Name: NoSQL

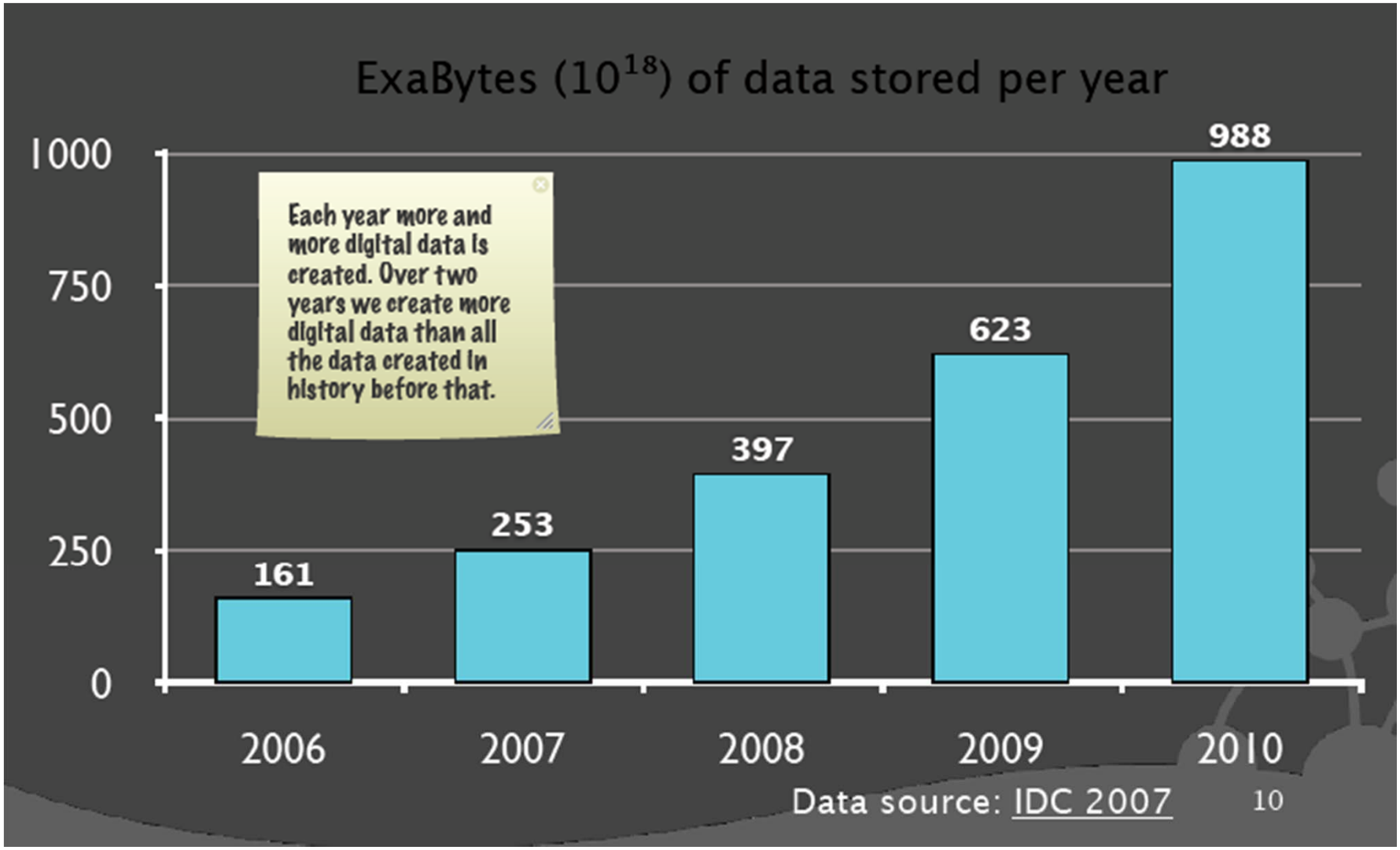
- Is it No to SQL ?

- Not Only SQL

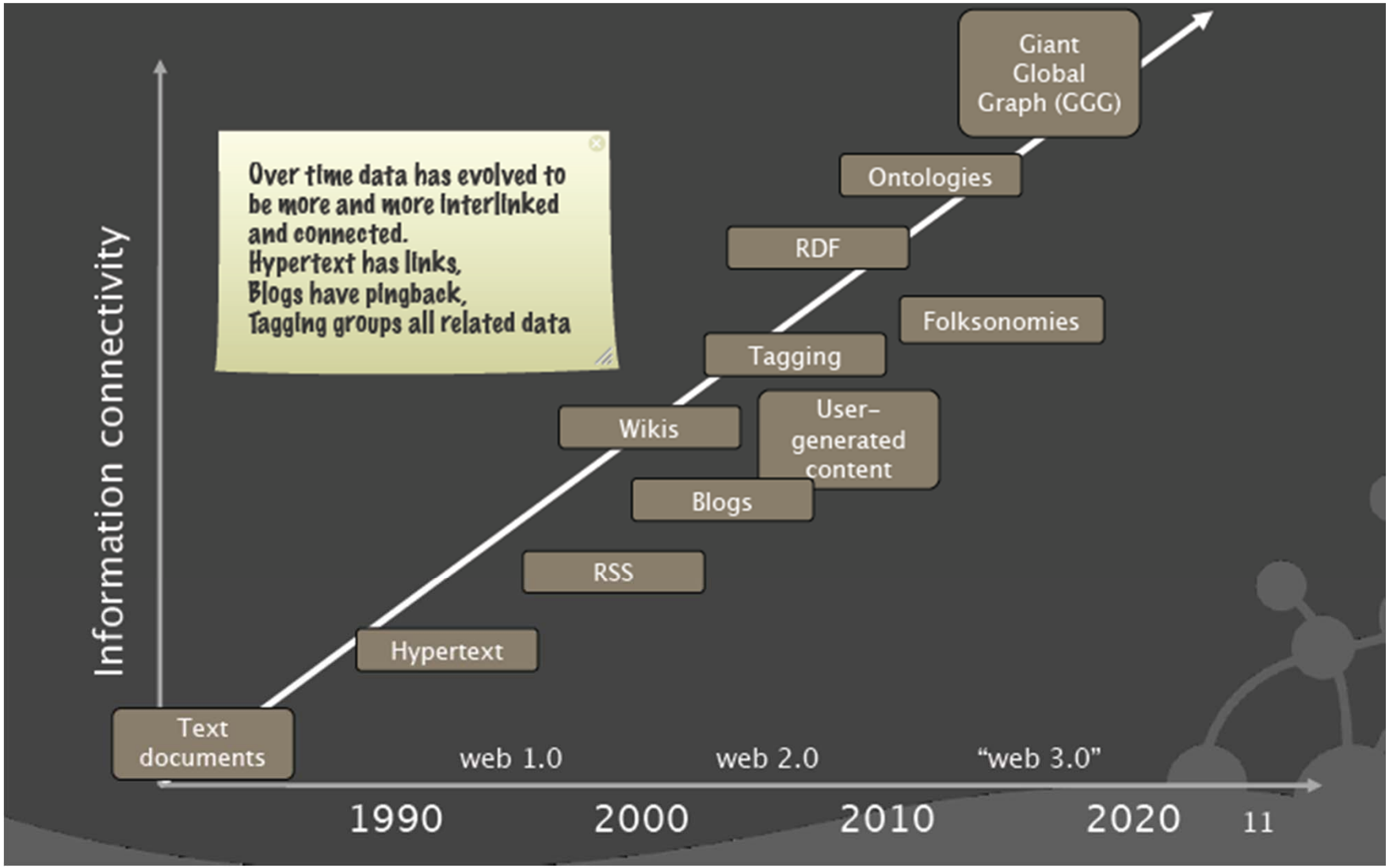
NOSQL - Defined by what it is Not

- “Any database that is not a Relational Database”
- The term was coined at a meetup with the creators behind some
- prominent emerging databases
- “Non-Relational Databases” might be more correct
- - But it’s a mouthful!
  - ... then there was a conference ...
  - ... and a mailing list ...
  - ... the name caught on ...
  - ... then there were more conferences ...
  - ... and here we are!

Why NoSQL: Trend 1 Data Size



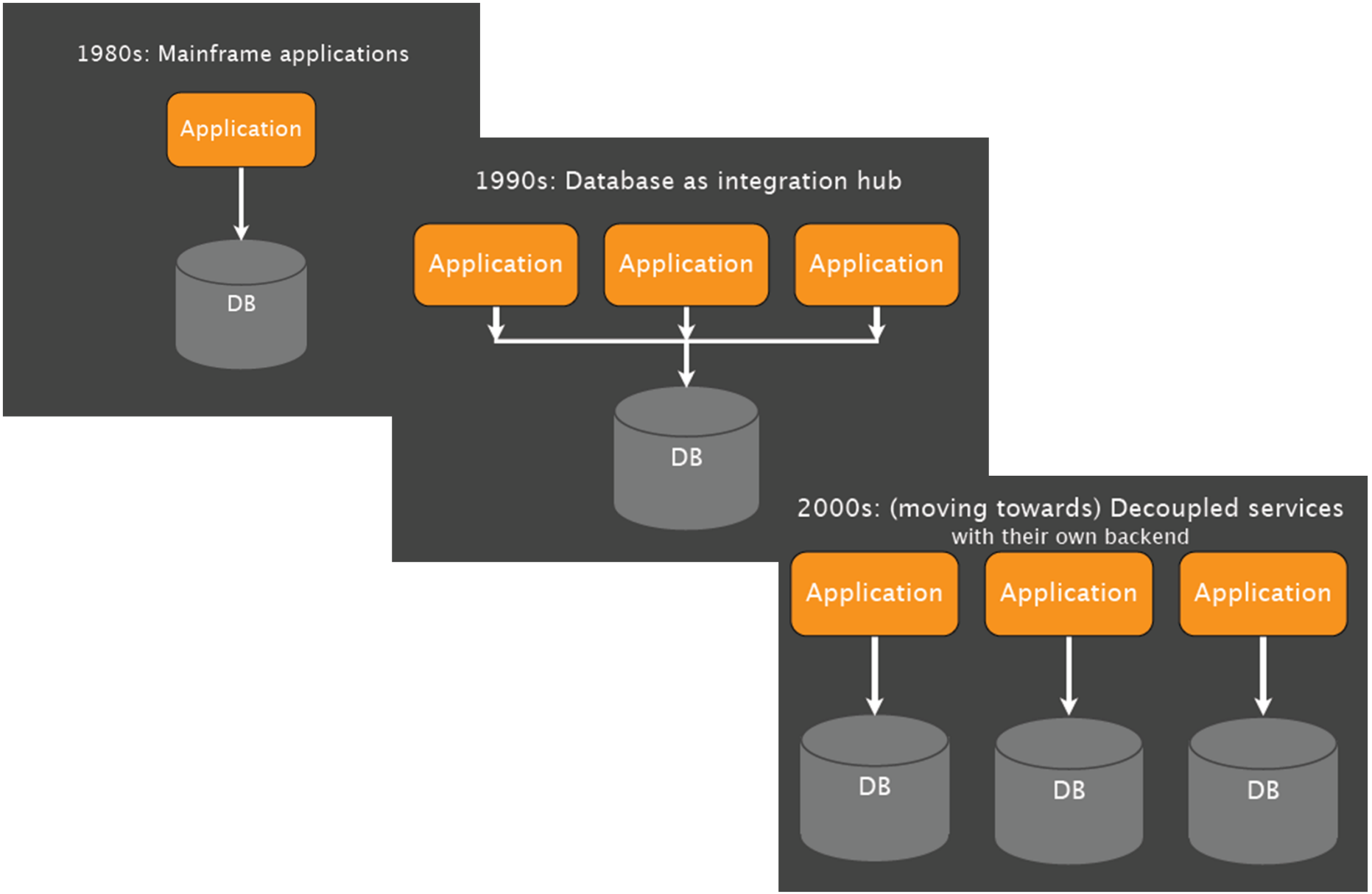
Why NoSQL: Trend 2 Connectedness



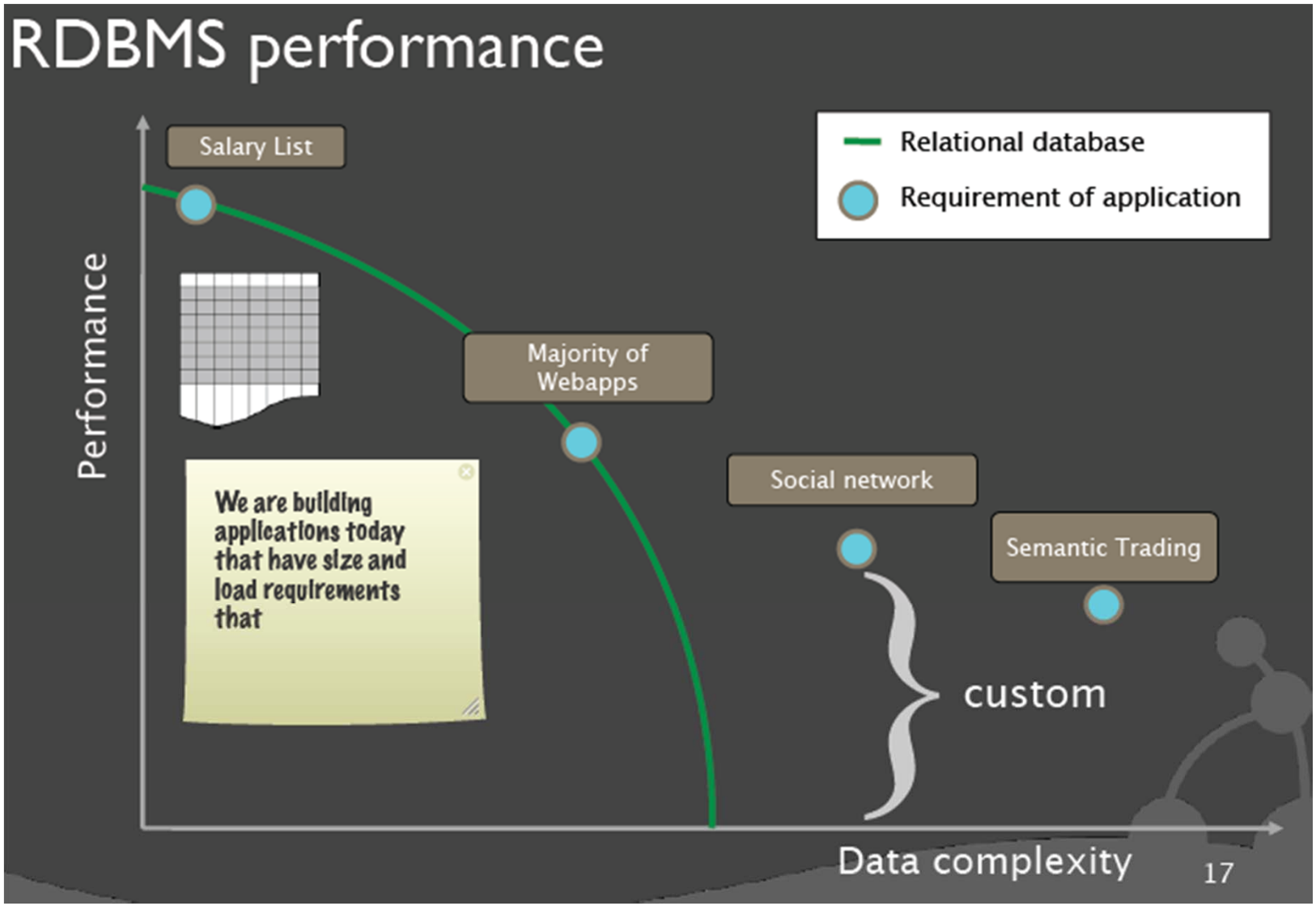
Why NoSQL: Trend 3 Semi-structure

- Individualization of content
- In the salary lists of the 1970s, all elements had exactly one job
- In the salary lists of the 2000s, we need 5 job columns! Or 8? Or 15?
- All encompassing “entire world views”
- Store more data about each entity
- Trend accelerated by the decentralization of content generation that is the hallmark of the age of participation (“web 2.0”)

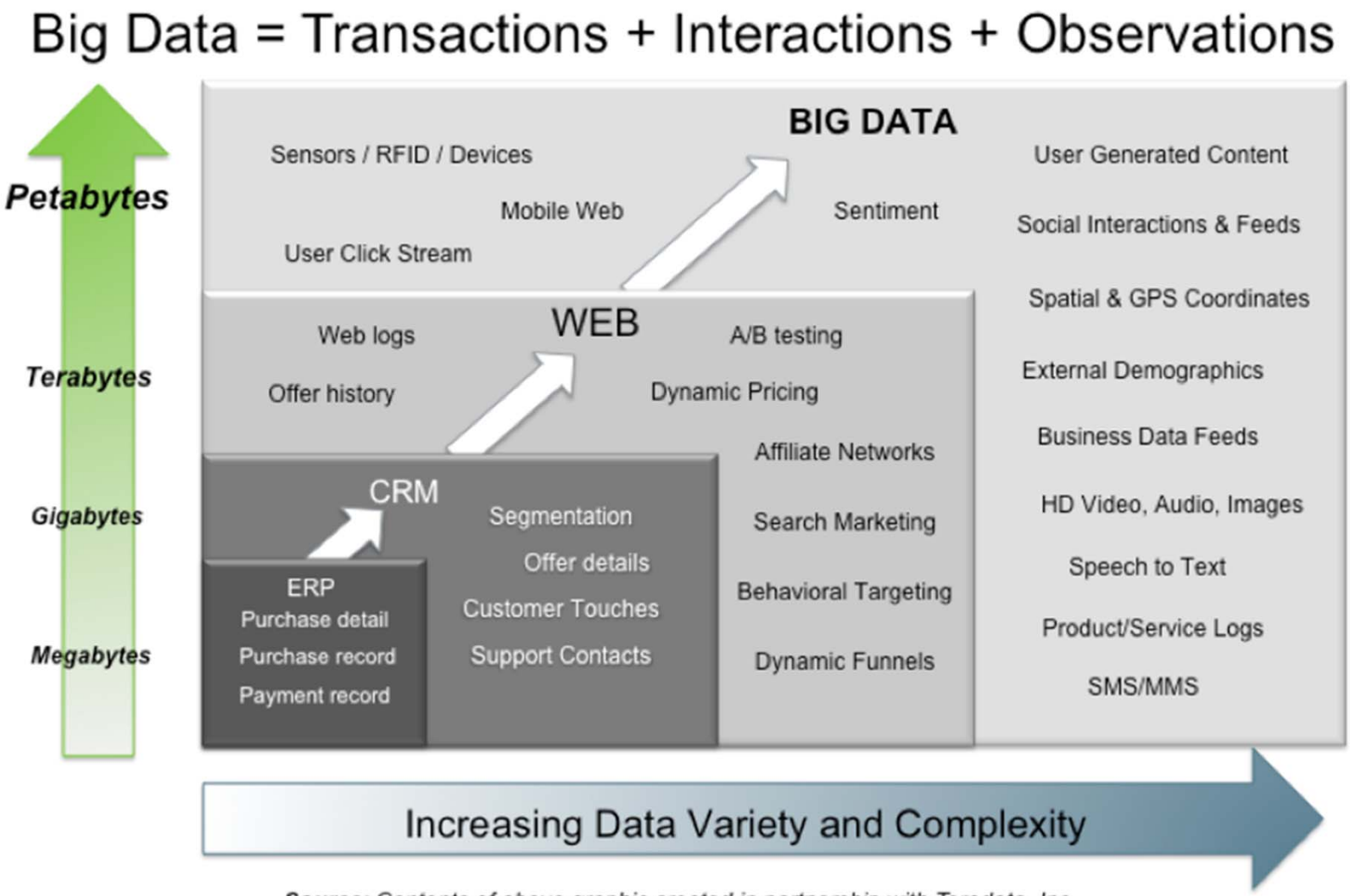
Why NoSQL: Trend 4 Architecture



Limitations of RDBMS



Evolution of NoSQL DBs



***Source**: Contents of above graphic created in partnership with Teradata, Inc.*

Why NoSQL

- ACID doesn’t scale well
- Web apps have different needs (than the apps that RDBMS were designed for)
  - Low and predictable response time (latency)
  - Scalability & elasticity (at low cost!)
  - High availability
  - Flexible schemas / semi-structured data
  - Geographic distribution (multiple datacenters)
- Web apps can (usually) do without
  - Transactions / strong consistency / integrity
  - Complex queries

NoSQL use cases

1. Massive data volumes
   - Massively distributed architecture required to store the data
   - Google, Amazon, Yahoo, Facebook – 10-100K servers
2. Extreme query workload
3. Impossible to efficiently do joins at that scale with an RDBMS
   - Schema evolution
   - Schema flexibility (migration) is not trivial at large scale
   - Schema changes can be gradually introduced with NoSQL

Dis / Advantages of NoSQL

- Advantages
  - Massive scalability
  - High availability
  - Lower cost (than competitive solutions at that scale)
  - (usually) predictable elasticity
  - Schema flexibility, sparse & semi-structured data
- Disadvantages
  - Limited query capabilities (so far)
  - Eventual consistency is not intuitive to program for
  - Makes client applications more complicated
  - No standardizatrion
  - Portability might be an issue
  - Insufficient access control

SQL Characteristics

- Data stored in columns and tables
- Relationships represented by data
- Data Manipulation Language
- Data Definition Language
- Transactions
- Abstraction from physical layer

SQL Physical Layer Abstraction

- Applications specify what, not how
- Query optimization engine
- Physical layer can change without modifying applications
  - Create indexes to support queries
  - In Memory databases

Data Manipulation Language (DML)

- Data manipulated with Select, Insert, Update, & Delete statements
  - Select T1.Column1, T2.Column2 …
    From Table1, Table2 …
    Where T1.Column1 = T2.Column1 …
- Data Aggregation
- Compound statements
- Functions and Procedures
- Explicit transaction control

Data Definition Language

- Schema defined at the start
- Create Table (Column1 Datatype1, Column2 Datatype 2, …)
- Constraints to define and enforce relationships
  - Primary Key
  - Foreign Key
  - Etc.
- Triggers to respond to Insert, Update , & Delete
- Stored Modules
- Alter …
- Drop …
- Security and Access Control

Transactions – ACID Properties

- **A**tomic – All of the work in a transaction completes (commit) or none of it completes
- **C**onsistent – A transaction transforms the database from one consistent state to another consistent state. Consistency is defined in terms of constraints.
- **I**solated – The results of any changes made during a transaction are not visible until the transaction has committed.
- **D**urable – The results of a committed transaction survive failures

Brewer's CAP Theorem
(E. Brewer, N. Lynch)

A distributed system can support only two of the following characteristics:

- Consistency
- Availability
- Partition tolerance

.

Consistency

- all nodes see the same data at the same time
- client perceives that a set of operations has occurred all at once
- More like Atomic in ACID transaction properties

Availability

- Node failures do not prevent survivors from continuing to operate
- Every operation must terminate in an intended response

Partition Tolerance

- the system continues to operate despite arbitrary message loss
- Operations will complete, even if individual components are unavailable

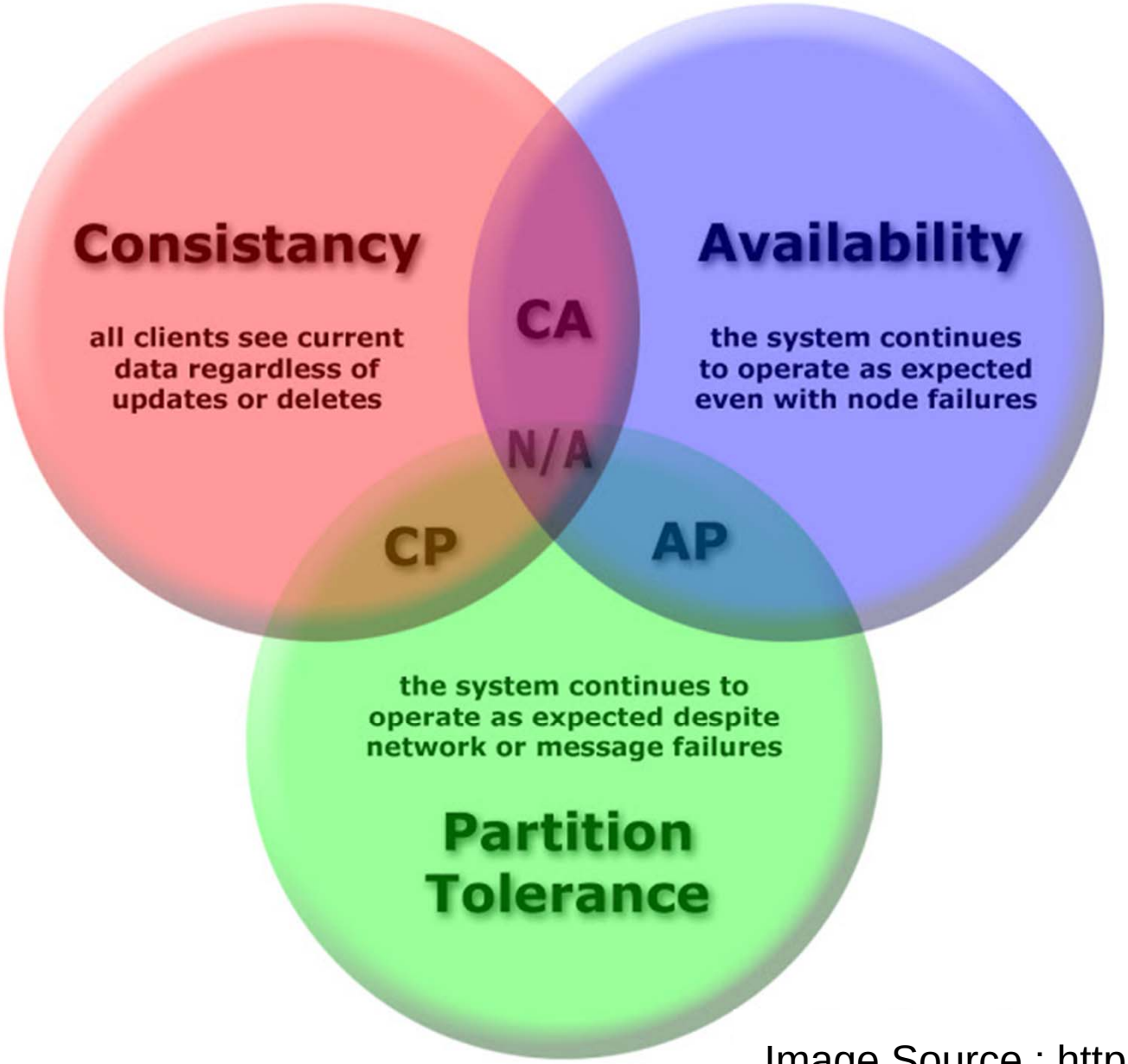


Image Source : http://blog.nosqltips.com

BASE Transactions

- Acronym contrived to be the opposite of ACID
  - **B**asically **A**vailable,
  - **S**oft state,
  - **E**ventually Consistent
- Characteristics
  - Weak consistency – stale data OK
  - Availability first
  - Best effort
  - Approximate answers OK
  - Aggressive (optimistic)
  - Simpler and faster

BASE Model

Focuses on Partition tolerance and availability and throws consistency out the window

- Basic Availability :
  - This constraint states that the system does guarantee the availability of the data as regards CAP Theorem
- Soft State :
  - The state of the system could change over time and called as ‘soft’ state
- Eventual Consistency :
  - The system will eventually become consistent once it stops receiving input. The data will propagate to everywhere it should sooner or later, but the system will continue to receive input and is not checking the consistency of every transaction before it moves onto the next one.

NoSQL Database Types

Discussing NoSQL databases is complicated because there are a variety of types:

- Column Store – Each storage block contains data from only one column
- Document Store – stores documents made up of tagged elements
- Key-Value Store – Hash table of keys

Other Non-SQL Databases

- XML Databases
- Graph Databases
- Codasyl Databases
- Object Oriented Databases
- Etc…

NoSQL Example: Column Store

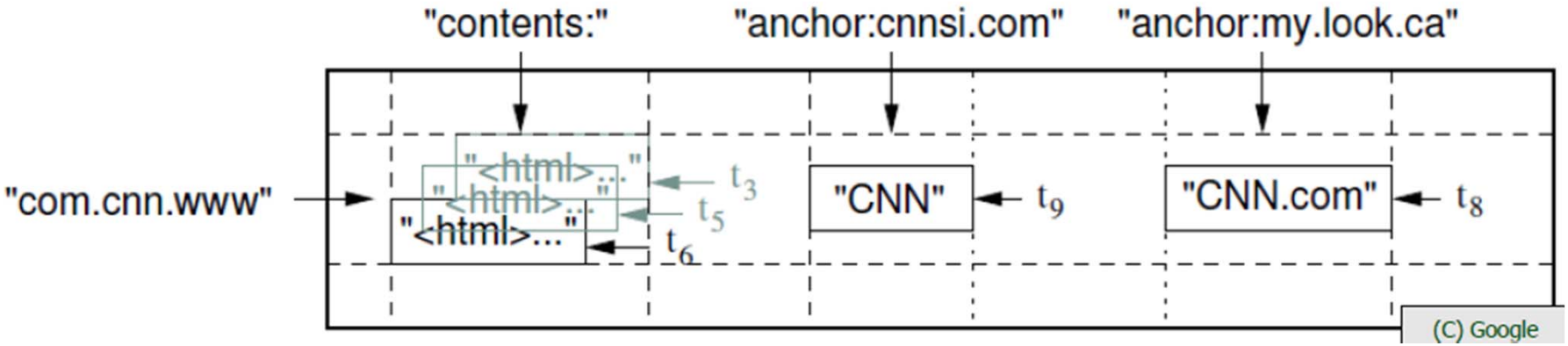
- Each storage block contains data from only one column
- Example: Hadoop/Hbase
  - http://hadoop.apache.org/
  - Yahoo, Facebook
- Example: Ingres VectorWise
  - Column Store integrated with an SQL database
  - http://www.ingres.com/products/vectorwise

Column Store Comments

- More efficient than row (or document) store if:
  - Multiple row/record/documents are inserted at the same time so updates of column blocks can be aggregated
  - Retrievals access only some of the columns in a row/record/document

Ex. Column Store Google BigTable

- Google, ~2006
- Sparse, distributed, persistent multidimensional sorted map
- *(row, column, timestamp)* dimensions, value is *string*
- Key features
  - Hybrid row/column store
  - Single master (stand-by replica)
  - Versioning



Google BigTable (Data Model)

- *Row*
  - Keys are arbitrary strings
  - Data is sorted by row key
- *Tablet*
  - Row range is dynamically partitioned into tablets (sequence of rows)
  - Range scans are very efficient
  - Row keys should be chosen to improve *locality* of data access
- *Column, Column Family*
  - Column keys are arbitrary strings, unlimited number of columns
  - Column keys can be grouped into families
  - Data in a CF is stored and compressed together (Locality Groups)
  - Access control on the CF level

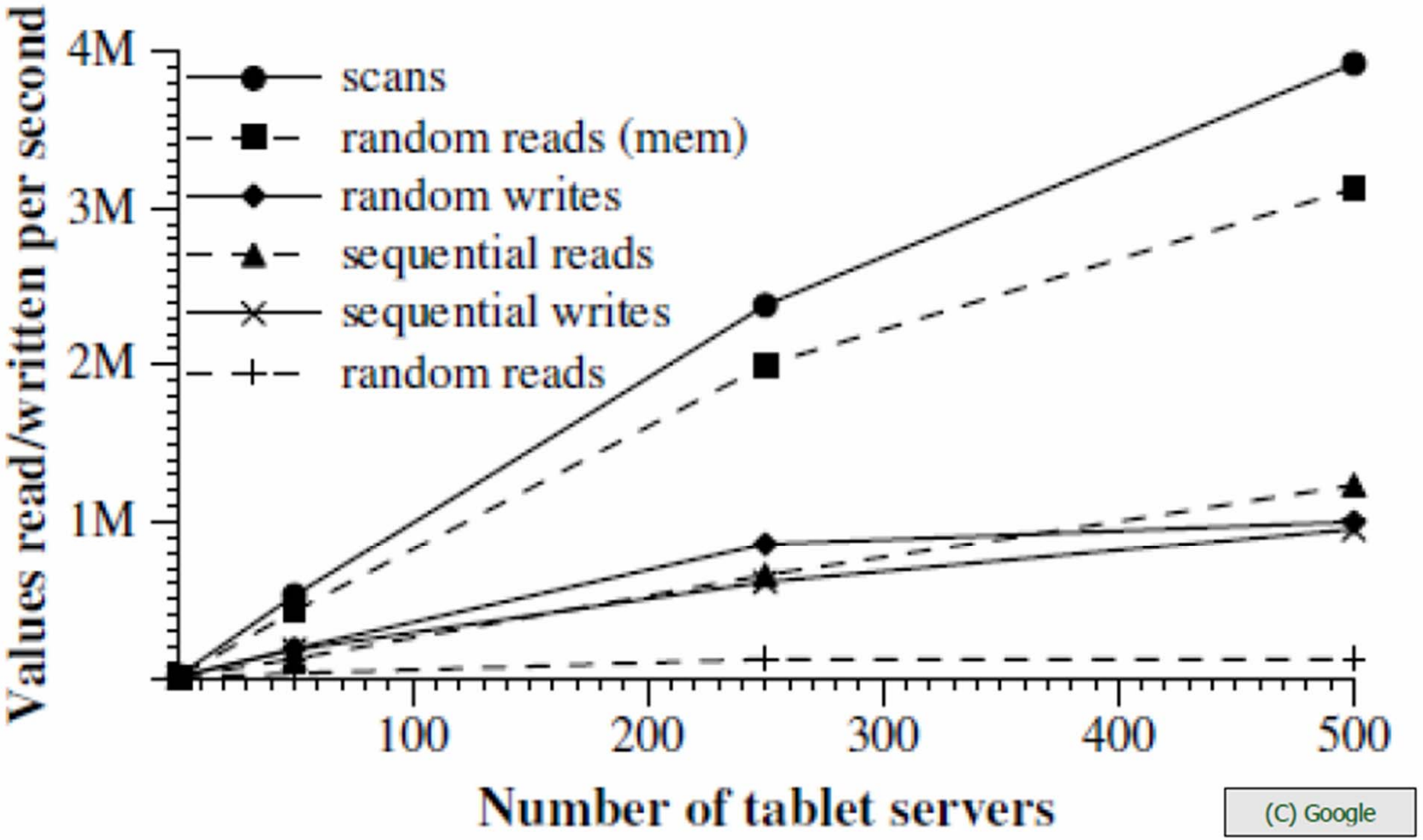
Google BigTable (Data Model Cont.)

- *Timestamps*
  - Each cell has multiple versions
  - Can be manually assigned
- *Versioning*
  - Automated garbage collection
  - Retain *last N* versions
  - Retain versions *newer than TS*
- Architecture
  - Data stored on *GFS*
  - Relies on *Chubby* (distributed lock service, Paxos)
  - 1 *Master server*
  - thousands of *Tablet servers*

Google BigTable (Architecture)

- *Master server*
  - Assign tablets to Tablet Servers
  - Balance TS load
  - Garbage collection
  - Schema management
  - Client data does **not** move through the MS (directly through TS)
  - Tablet location **not** handled by MS
- *Tablet server (many)*
  - thousands of tablets per TS
  - Manages Read / Write / Split of its tablets

Google BigTable (Performance)

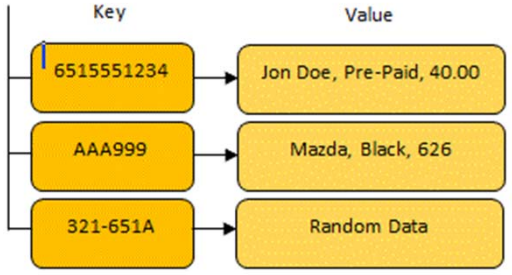


NoSQL Examples: Key-Value Store

- Hash tables of Keys
- Values stored with Keys
- Fast access to small data values
- Example – Project-Voldemort
  - http://www.project-voldemort.com/
  - Linkedin
- Example – MemCacheDB
  - http://memcachedb.org/
  - Backend storage is Berkeley-DB

Key-Value Store

- Store items as alpha-numeric identifiers(keys) and associated values in simple, standalone tables (Hash table)
- The values can be simple text strings or more complex list and sets.
- Query can usually only be performed against keys and it limited to exact matches
- Primary Use :
  - Manage user profile or session or retrieving product names
  - Amazon core services use Dynamo as distributed data store



Example Key Value Store: Dynamo

- Amazon, ~2007
- P2P key-value store
  - Object versioning
  - Consistent hashing
  - Gossip – membership & failure detection
  - Quorum reads
- Requirements & assumptions
  - *Simple query model* (unique keys, blobs, no schema, no multi-access)
  - *Scale out* (elasticity)
  - *Eventual consistency* (improved availability)
  - *Decentralized* & *symmetric* (P2P), *heterogeneous* (load distribution)
  - *Low latency / high throughput*
  - *Internal service* (no security model)

NoSQL Example: Document Store

- Example: CouchDB
  - http://couchdb.apache.org/
  - BBC
- Example: MongoDB
  - http://www.mongodb.org/
  - Foursquare, Shutterfly
- JSON – JavaScript Object Notation

Document Store Example: CouchDB

- Schema-free, document oriented database
  - Documents stored in JSON format (XML in old versions)
  - B-tree storage engine
  - MVCC model, no locking
  - no joins, no PK/FK (UUIDs are auto assigned)
  - Implemented in Erlang
  - 1st version in C++, 2nd in Erlang and 500 times more scalable (source: “Erlang Programming” by Cesarini & Thompson)
  - Replication (incremental)
- Documents
  - UUID, version
  - Old versions retained

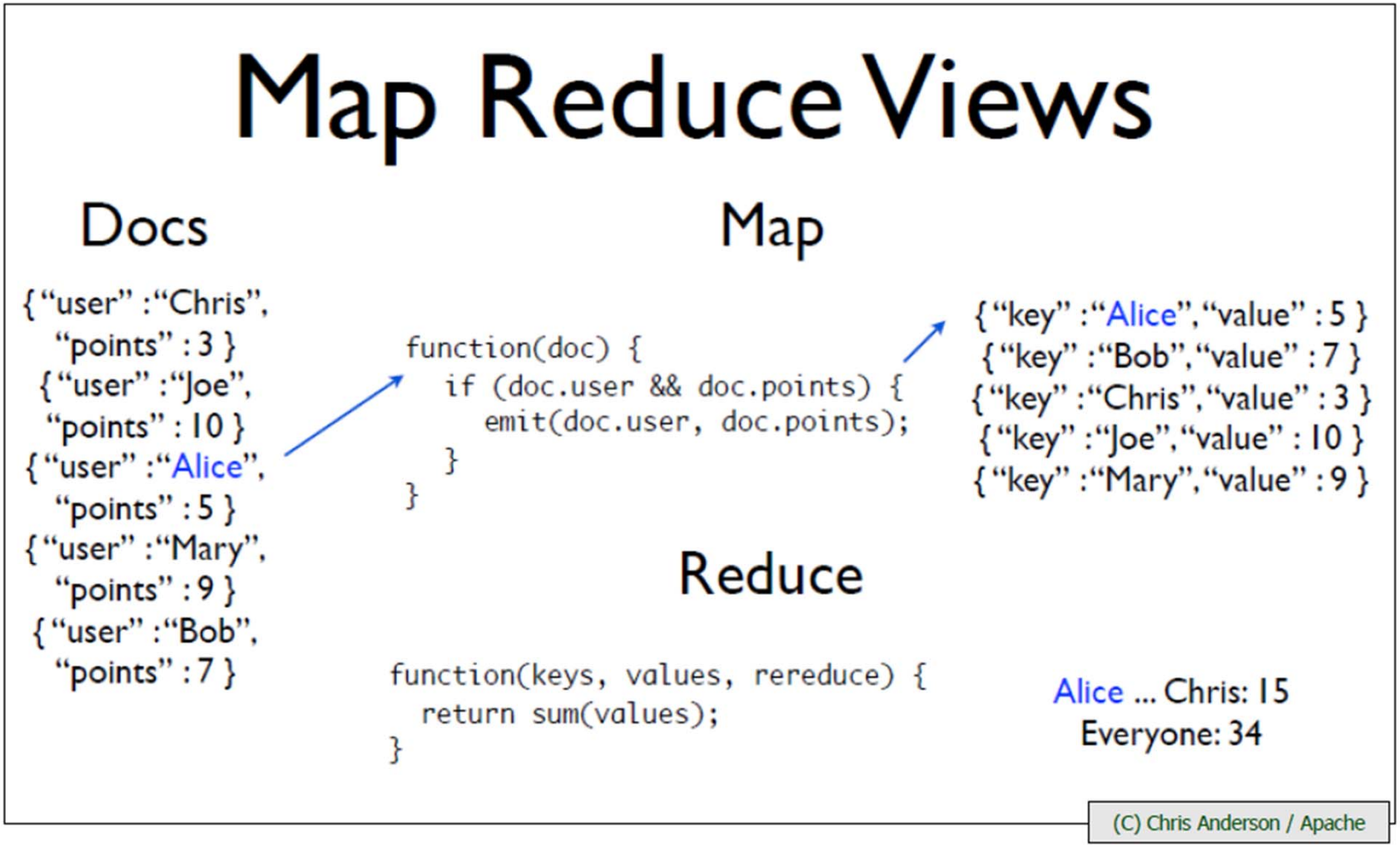
CouchDB JSON Example

```
{
  "_id": "guid goes here",
  "_rev": "314159",

  "type": "abstract",

  "author": "Keith W. Hare"

  "title": "SQL Standard and NoSQL Databases",

  "body": "NoSQL databases (either no-SQL or Not Only SQL)
           are currently a hot topic in some parts of
           computing.",
  "creation_timestamp": "2011/05/10 13:30:00 +0004"
}
```

CouchDB JSON Tags

- "_id"
  - GUID – Global Unique Identifier
  - Passed in or generated by CouchDB
- "_rev"
  - Revision number
  - Versioning mechanism
- "type", "author", "title", etc.
  - Arbitrary tags
  - Schema-less
  - Could be validated after the fact by user-written routine

MapReduce on CouchDB



MapReduce

- Technique for indexing and searching large data volumes
- Two Phases, Map and Reduce
  - Map
    - Extract sets of Key-Value pairs from underlying data
    - Potentially in Parallel on multiple machines
  - Reduce
    - Merge and sort sets of Key-Value pairs
    - Results may be useful for other searches

MapReduce

- Map Reduce techniques differ across products
- Implemented by application developers, not by underlying software

Map Reduce Patent

Google granted US Patent 7,650,331, January 2010

System and method for efficient large-scale data processing

A large-scale data processing system and method includes one or more application-independent map modules configured to read input data and to apply at least one **application-specific map operation** to the input data to produce intermediate data values, wherein the map operation is automatically parallelized across multiple processors in the parallel processing environment. A plurality of intermediate data structures are used to store the intermediate data values. One or more application-independent reduce modules are configured to retrieve the intermediate data values and to apply at least one **application-specific reduce operation** to the intermediate data values to provide output data.

NoSQL “Definition”

From www.nosql-database.org:

> Next Generation Databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontal scalable**. The original intention has been **modern web-scale databases**.
>
> The movement began early 2009 and is growing rapidly. Often more characteristics apply as: **schema-free, easy replication support, simple API, eventually consistent** / **BASE** (not ACID), a **huge data amount**, and more.

NoSQL Distinguishing Characteristics

- Large data volumes
  - Google's "big data"
- Scalable replication and distribution
  - Potentially thousands of machines
  - Potentially distributed around the world
- Queries need to return answers quickly
- Mostly query, few updates
- Asynchronous Inserts & Updates
- Schema-less
- ACID transaction properties are not needed – BASE
- CAP Theorem
- Mostly Open Source development

Storing and Modifying Data

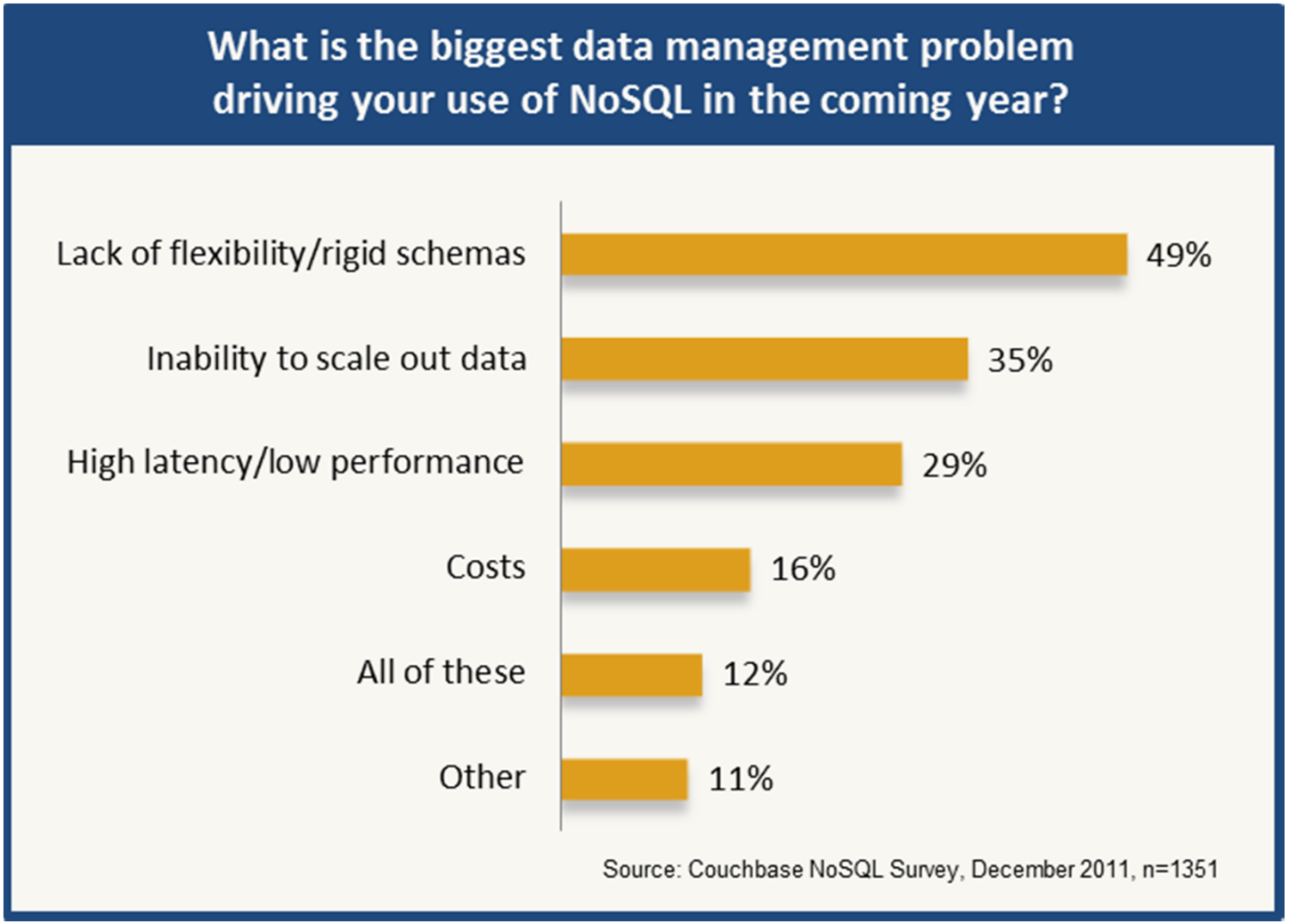
- Syntax varies
  - HTML
  - Java Script
  - Etc.
- Asynchronous – Inserts and updates do not wait for confirmation
- Versioned
- Optimistic Concurrency

Retrieving Data

- Syntax Varies
  - No set-based query language
  - Procedural program languages such as Java, C, etc.
- Application specifies retrieval path
- No query optimizer
- Quick answer is important
- May not be a single “right” answer

Adoption of NoSQL Database

- Couchbase survey was conducted in the 2012.



NoSQL Summary

- NoSQL databases reject:
  - Overhead of ACID transactions
  - “Complexity” of SQL
  - Burden of up-front schema design
  - Declarative query expression
- Programmer responsible for
  - Step-by-step procedural language
  - Navigating access path

Databases Landscape with respect to Availability, Consistency, Partition Tolerance

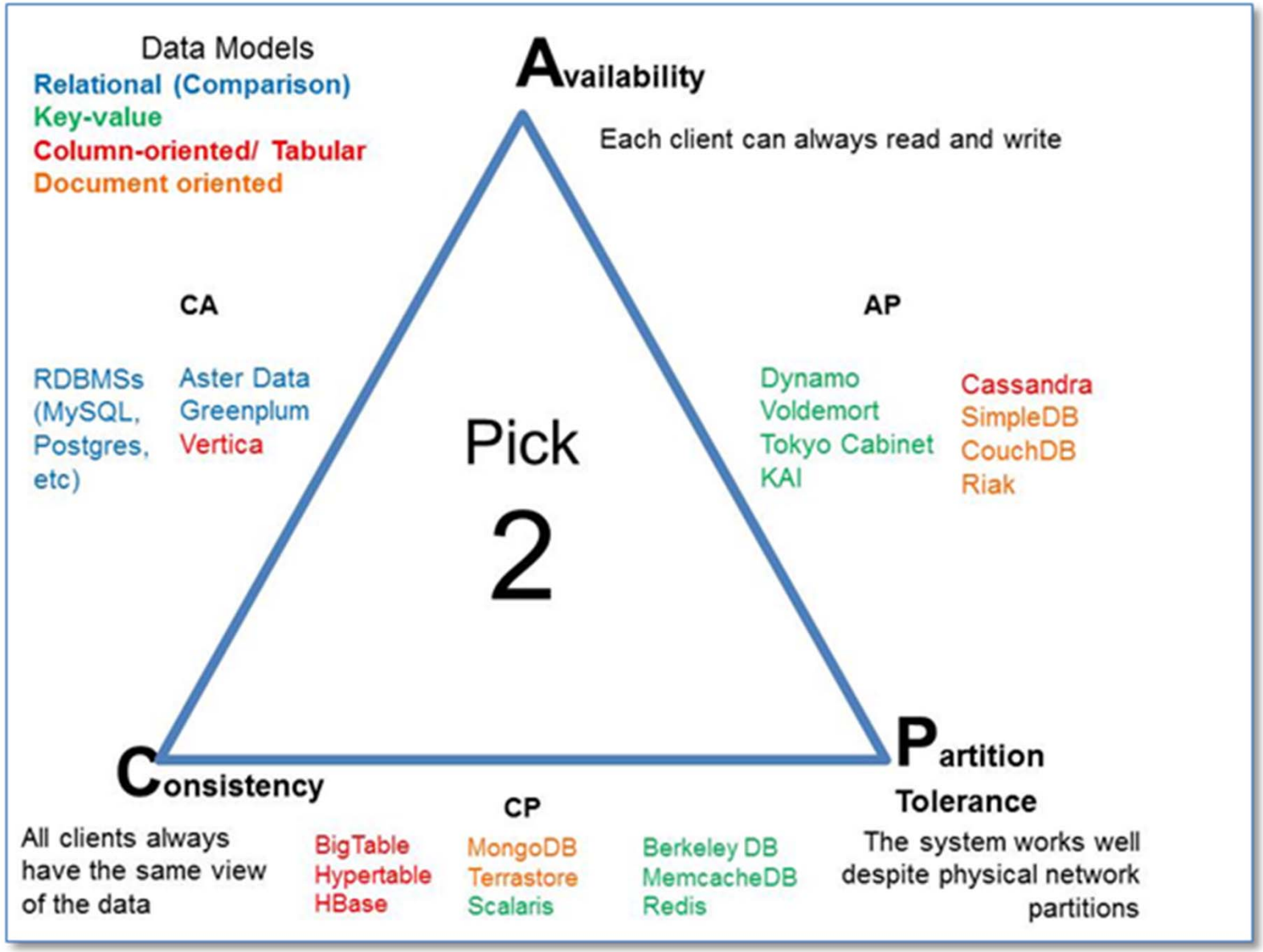


Image Source : http://blog.flux7.com/blogs/nosql/cap-theorem-why-does-it-matter

Database Ranking

283 systems in ranking, November 2015

Rank Nov 2015	Rank Oct 2015	Rank Nov 2014	DBMS	Database Model	Score Nov 2015	Score Oct 2015	Score Nov 2014
1.	1.	1.	Oracle	Relational DBMS	1480.95	+13.99	+28.82
2.	2.	2.	MySQL	Relational DBMS	1286.84	+7.88	+7.77
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1122.33	-0.90	-97.87
4.	4.	↑ 5.	MongoDB +	Document store	304.61	+11.34	+59.87
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	285.69	+3.56	+28.33
6.	6.	6.	DB2	Relational DBMS	202.52	-4.28	-3.71
7.	7.	7.	Microsoft Access	Relational DBMS	140.96	-0.87	+2.12
8.	8.	↑ 9.	Cassandra +	Wide column store	132.92	+3.91	+40.93
9.	9.	↓ 8.	SQLite	Relational DBMS	103.45	+0.78	+8.17
10.	10.	↑ 11.	Redis +	Key-value store	102.41	+3.61	+20.06

Image captured at : http://db-engines.com/en/ranking

NOTE: Most of the major Relational DB Vendors have included NoSQL components to their solutions to stay ahead of the competition.

Web References

- “NoSQL -- Your Ultimate Guide to the Non - Relational Universe!” http://nosql-database.org/links.html
- “NoSQL (RDBMS)” http://en.wikipedia.org/wiki/NoSQL
- PODC Keynote, July 19, 2000. *Towards Robust Distributed Systems*. Dr. Eric A. *Brewer*. Professor, UC Berkeley. Co-Founder & Chief Scientist, Inktomi **.** *www.eecs.berkeley.edu/~**brewer**/cs262b-2004/PODC-keynote.pdf*
- “Brewer's CAP Theorem” posted by Julian Browne, January 11, 2009. http://www.julianbrowne.com/article/viewer/brewers-cap-theorem

Web References

- “Exploring CouchDB: A document-oriented database for Web applications”, Joe Lennon, Software developer, Core International. http://www.ibm.com/developerworks/opensource/library/os-couchdb/index.html
- “Graph Databases, NOSQL and Neo4j” Posted by Peter Neubauer on May 12, 2010 at: http://www.infoq.com/articles/graph-nosql-neo4j
- “Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase comparison”, Kristóf Kovács. http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis
- “Distinguishing Two Major Types of Column-Stores” Posted by Daniel Abadi onMarch 29, 2010 http://dbmsmusings.blogspot.com/2010/03/distinguishing-two-major-types-of_29.html

Web References

- “MapReduce: Simplified Data Processing on Large Clusters”, Jeffrey Dean and Sanjay Ghemawat, December 2004. http://labs.google.com/papers/mapreduce.html
- “Scalable SQL”, ACM Queue, Michael Rys, April 19, 2011 http://queue.acm.org/detail.cfm?id=1971597
- “a practical guide to noSQL”, Posted by Denise Miura on March 17, 2011 at http://blogs.marklogic.com/2011/03/17/a-practical-guide-to-nosql/

Books

- “CouchDB *The Definitive Guide*”, J. Chris Anderson, Jan Lehnardt and Noah Slater. O’Reilly Media Inc., Sebastopool, CA, USA. 2010
- “Hadoop *The Definitive Guide*”, Tom White. O’Reilly Media Inc., Sebastopool, CA, USA. 2011
- “MongoDB *The Definitive Guide*”, Kristina Chodorow and Michael Dirolf. O’Reilly Media Inc., Sebastopool, CA, USA. 2010