NoSQL

Relational Databases are awesome…

- Relational Databases solve most data problems
  - **Persistence**
    - We can store data, and it will remain stored!
  - **Integration**
    - We can integrate lots of different apps through a central DB
  - **Concurrency**
    - ACID transactions, strong consistency
  - **SQL**
    - Standard, well understand, very expressive

Trends and Issues

- Key trends include:
  - Increasing volume of **data and traffic**
  - More complex **data**

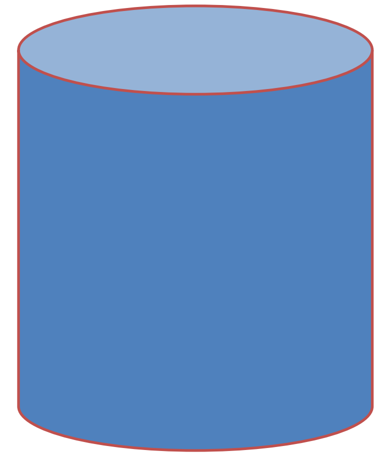
- Key issues include:
  - The **impedance mismatch** problem

Trends – Data Size

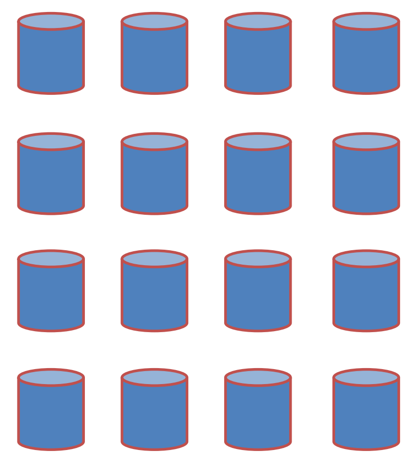
- We are creating, storing, processing… more data than ever before!
  - From **2005 to 2020**, the digital universe will grow by a factor of **300** !!
  - from **130 exabytes to 40,000 exabytes** (**40 trillion gigabytes)**
  - More than 5,200 gigabytes for every person in 2020
- "From now until 2020, the digital universe will about **double every two years**." - IDC – *The Digital Universe in 2020*

Dealing with data size trends

- Build **Bigger** Database machines
  - This can be **expensive**
  - **Fundamental limits** to machine size



- Build **Clusters** of smaller machines
  - Lots of **small machines** *(commodity machines)*
  - Each machine is cheap, potentially unreliable
  - Needs a DBMS which **understands clusters**



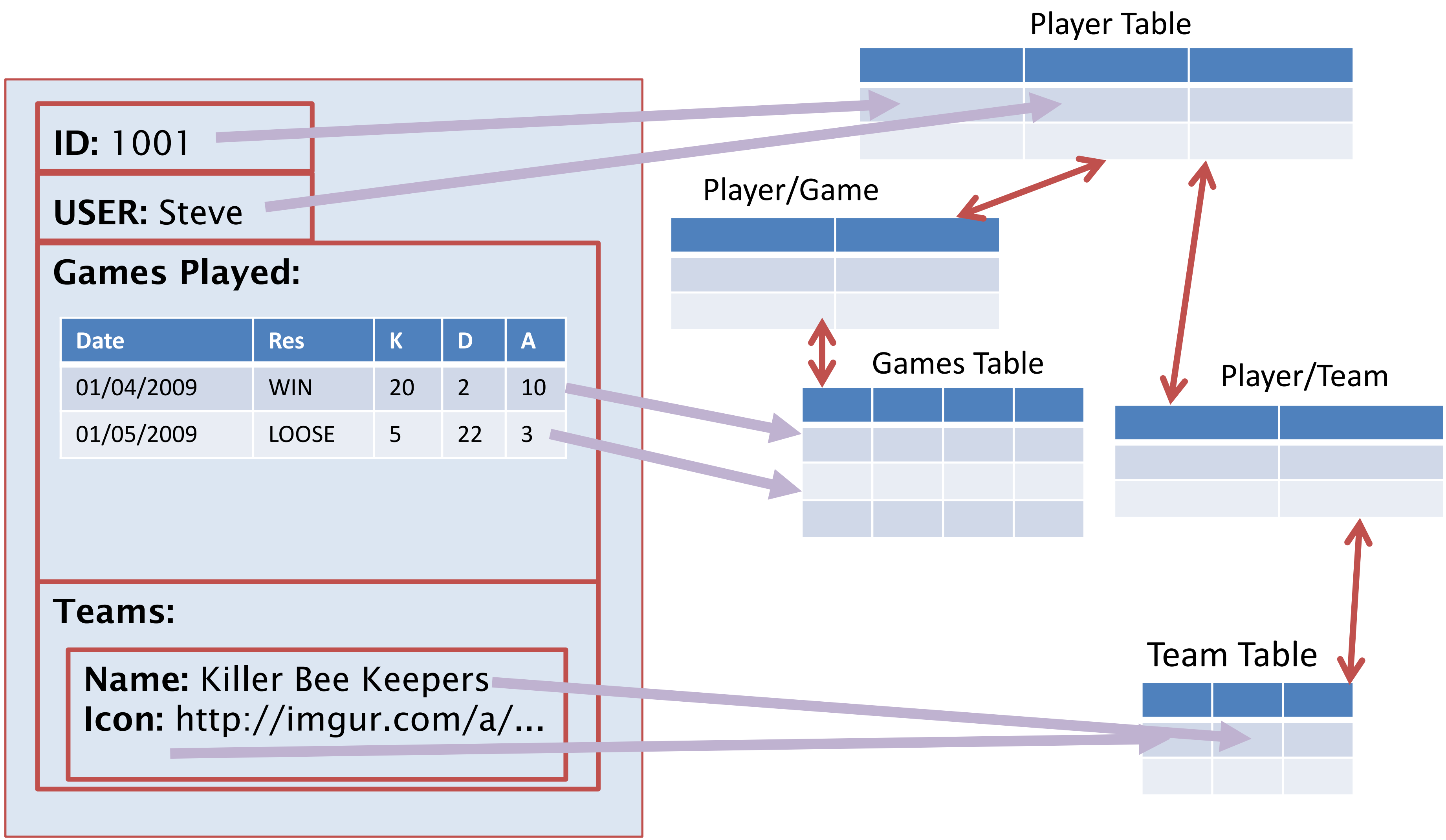
Relational Databases have issues…

- In dealing with (horizontal) scale
  - Designed to work on **single, large machines**
  - Difficult to **distribute** effectively
- More subtle: **An Impedance Mismatch**
  - We create logical structures in memory ***and then rip them apart to stick it in an RDBMS***
  - The RDBMS **data model** often **disjoint** from its intended use *(Normalization sucks sometimes)*
  - Uncomfortable to program with (joins and ORM (object relational mapping), etc.)

Impedance Mismatch Issue

- Object Orientation
  - Based on **software engineering** principals
- Relational Paradigms
  - Based on **mathematics** and **set theory**
- Mapping from one world to the other has problems

Example – Impedance Mismatch



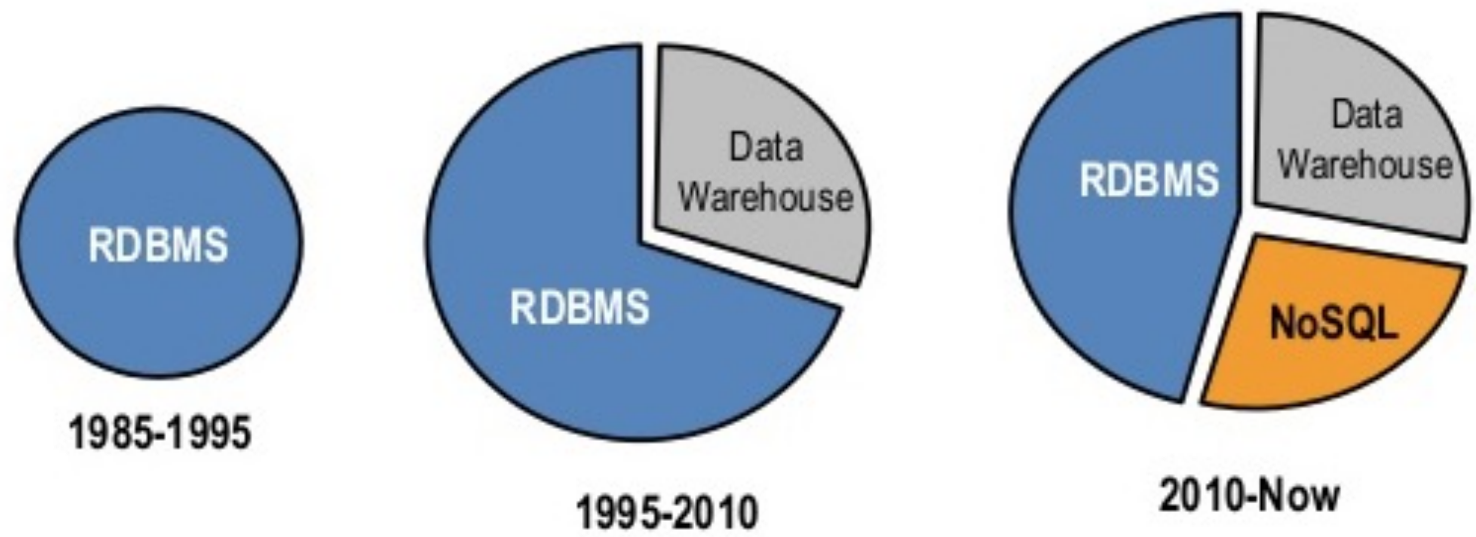
NoSQL – A Movement

- NoSQL was created to address
  - **Scalability** problems
  - **Impedance mismatch**
- Key features of NoSQL include
  - **Non-Relational** (Though they can be, but aren't good at it)
  - **Schema-less** (Except the implicit schema on application side)
  - **Inherently Distributed** (In different ways, some more so than others)
  - **Open Source** (mostly… not including e.g. Oracle's NoSQL)

Defining NoSQL isn't Easy

- Why? Its quite hard to define a movement based around a **negative**
  - Is a CSV file NoSQL? *(How about a turnip?)*
  - How about a pre-relational database *(MARK IV, IDMS)*
- NoSQL is not strictly definable
  - But many folks have **certainly tried**!

Three Eras of Databases



- RDBMS for transactions
- Data Warehouse for analytics
- NoSQL for scalability

NoSQL Distinguishing Characteristics

- Large data volumes
- 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 (some say schema-free)
- ACID transaction properties are not used
  - CAP Theorem
  - BASE
- Open source development

Recall Relational Transactions The 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
- **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

Relaxing ACID

- ACID is **a big deal** in traditional RDBMS
  - Broken transactions in a banking is **a big deal**
  - Money leaving your account but not entering the seller’s account is **a big deal**
- BUT in some situations the use case doesn’t need all or some of ACID
  - Seeing an old version of a **facebook post**
  - A **shopping cart** forgetting your items
  - A **tweet or two out** of the 138 million per day being lost (or temporarily lost)

CAP Theorem – What's a CAP?

- **Consistent:** Writes are atomic, all subsequent requests retrieve the new value
- **Available:** The database will always return a value so long as the server is running
- **Partition Tolerant:** The system will still function even if the cluster network is partitioned (i.e. the cluster loses contact with parts of itself (split brain))
- The well cited issue is:
  - **Of these 3**, you can only build an algorithm which **satisfies 2**
- Formal proof by Gilbert and Lynch: http://portal.acm.org/citation.cfm?doid=564585.564601

Is relational database a CA system?

Yes

No

Is relational database a CA system?

Yes

0%

No

0%

Is relational database a CA system?

Yes

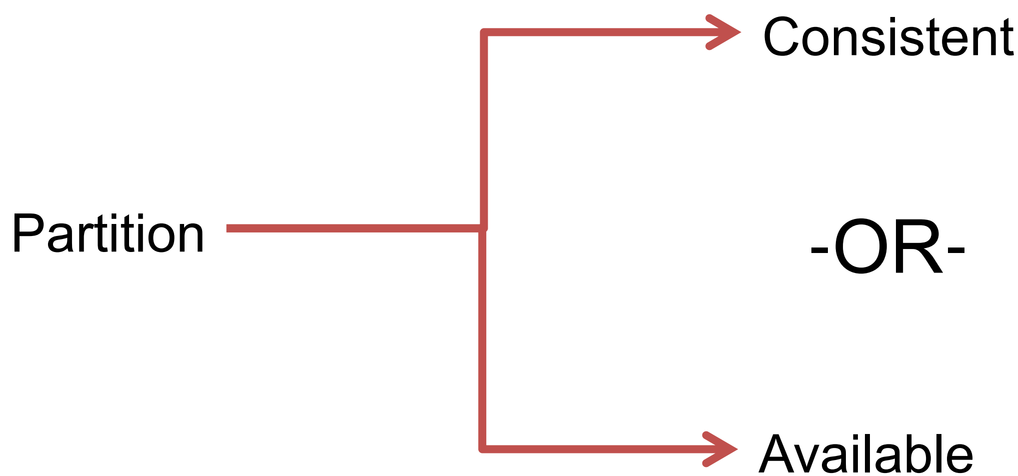
0%

No

0%

CAP – Another Perspective

- If you have a system that can get a network partition
  (if your system is distributed this will definitely happen)



- You must make a choice between:
  - Consistency (**disallow writes** during the partition)
  - Availability (have **potential inconsistency**)

BASE – An alternative to ACID

- If we want CAP's P (Partition protection), ACID can be restrictive, we can be BASE
- Acronym contrived to be the opposite of ACID
  - **B**asic **A**vailability
    - The application works basically all the time
  - **S**oft-state
    - Does not have to be consistent all the time
  - **E**ventual consistency
    - But will be in some known state eventually

Eventual Consistency

- A work around of CAP
- From Amazon's Dynamo paper:

  *"the storage system guarantees that if no new updates are made to the object, **eventually all accesses** will return the last updated value."*

  – For **certain systems**, this is good enough

BASE Transactions

- **Characteristics**
  - Weak consistency – stale data OK
  - Availability first
  - Approximate answers OK
  - Simpler and faster

NoSQL Varieties

redis

basho

mongoDB

{name: "mongo", type:"DB"}

MarkLogic

Jena

Neo4j the graph database

APACHE HBASE

hadoop

Cassandra

4store

NoSQL Varieties

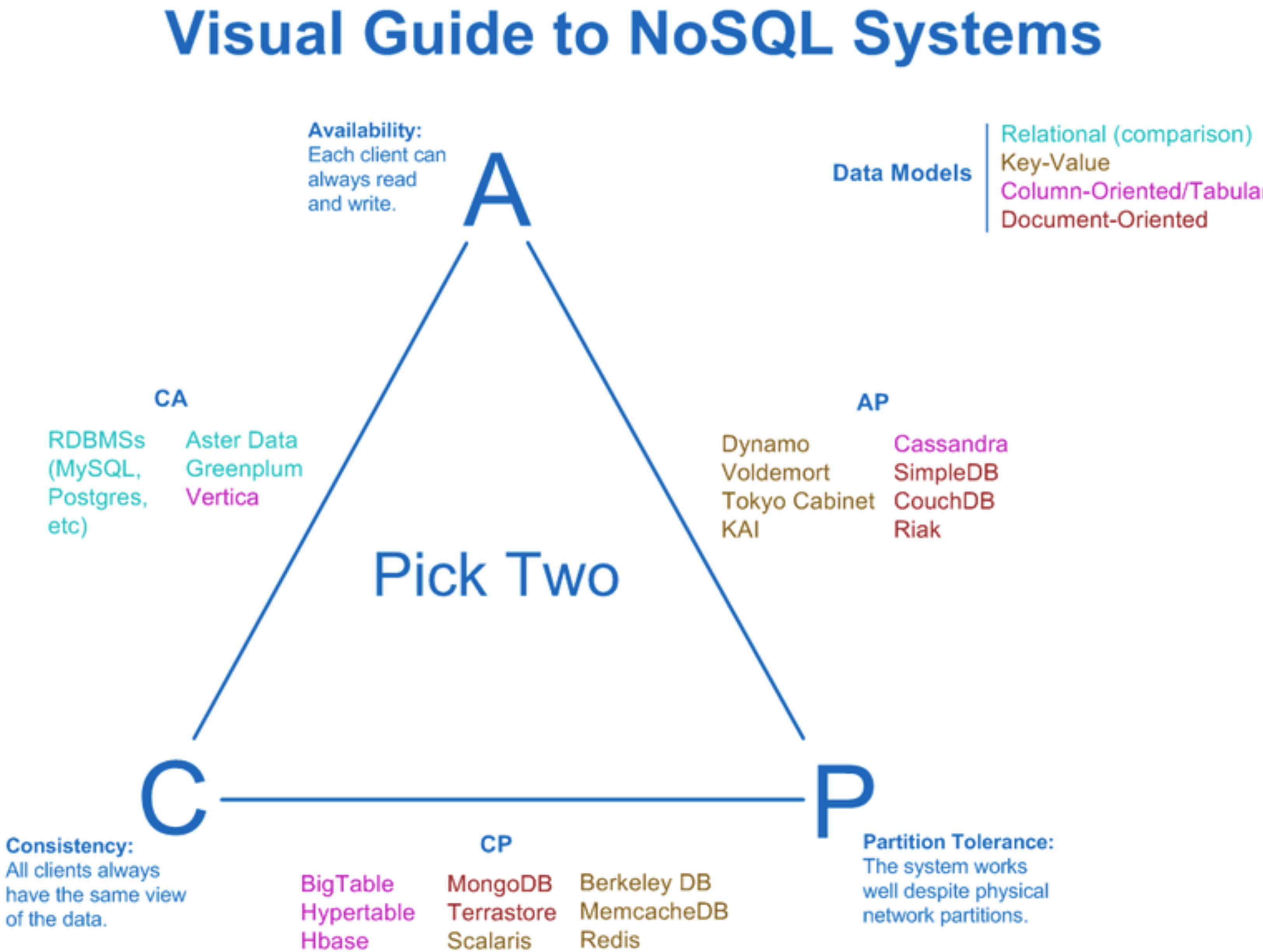
Key-Value stores

Document Oriented

Column Oriented

Graph DBs

CAP Theorem – The DB perspective

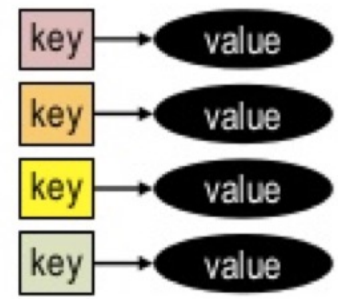


NoSQL Databases

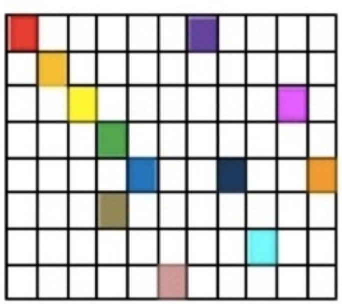
Examples:

- Key-value: Riak
- Document: MongoDB
- Column-Family: Cassandra
- Graph: Neo4j

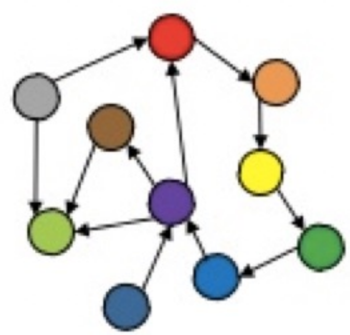
Key-Value



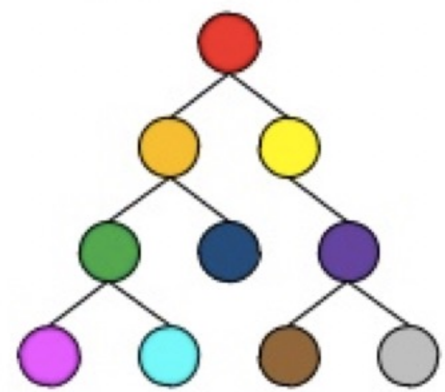
Column-Family



Graph



Document



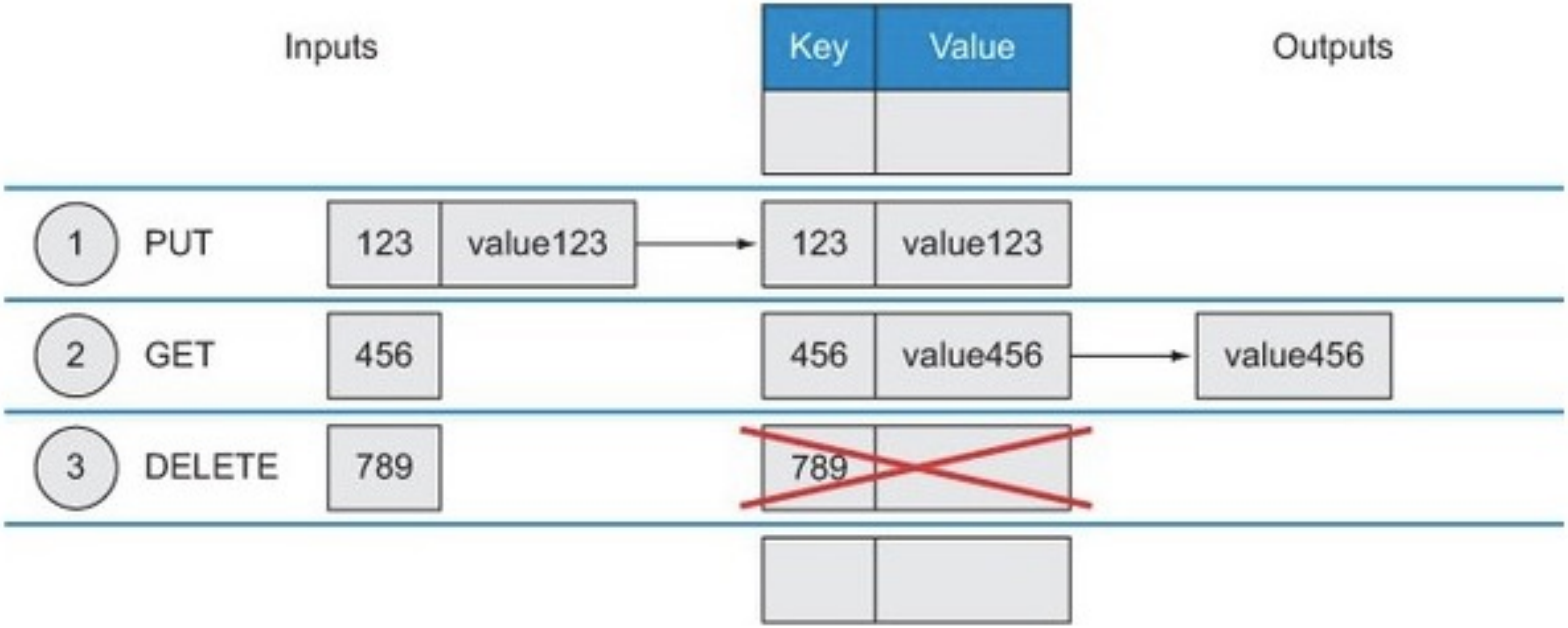
Key-Value Store Basics

- A collection of key-value pairs
- Give database a **key**, database returns a **value**
- The key is usually a string
- The value can be anything (text, structure, an image, etc.)
  - Database often unaware of value content
- Use of hash table

Key-Value Store Characteristics

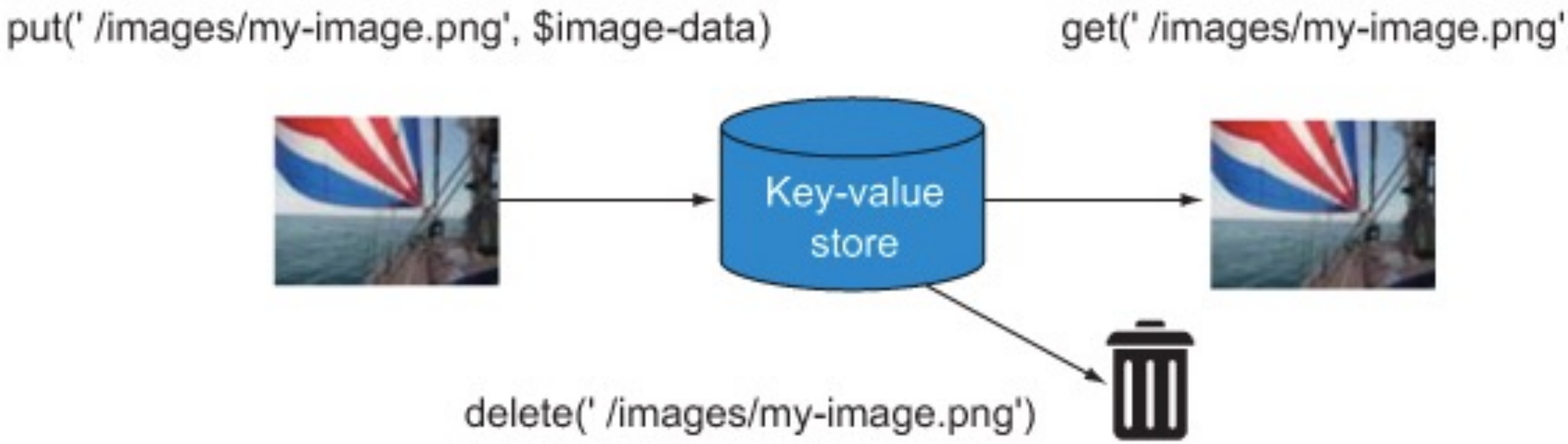
- Consistency is applicable only for operations on a single key
- Query by the key – and that's it
- Don't care what is stored in the value part of the key-value pair
- Scale by using sharding
  - The value of the key determines on which node the key is stored

Using Key-Value Store



Key-Value Store Example

"Values" can contain any type of data, e.g. images, video



Key-Value Store Example

Websites: using URLs as keys

Key	Value
http://www.example.com/index.html	<html>…
http://www.example.com/about.html	<html>…
http://www.example.com/products.html	<html>…
http://www.example.com/logo.png	Binary…

Suitable Use Cases

- Store session information
- User profiles, preferences
- Shopping cart data
  - Available all the time

When not to use

- Relationships among data
- Multioperation transactions
- Query by data
- Operations by sets

Key-Value Store Example Systems

- Project Voldemort
  - http://www.project-voldemort.com/
  - LinkedIn’s NoSQL key-value storage engine
- DynamoDB
  - http://aws.amazon.com/dynamodb/
  - NoSQL database service by Amazon
- MemCacheDB
  - https://memcached.org
  - Backend storage is Berkeley DB

Key-Value Stores Products

- **Riak**
  - **Buckets**/Keys/Values
  - Query with key, process with map-reduce
  - Secondary Indexes (metadata)
- **Redis**
  - More understanding of value types (strings, integers, lists, hashes)
  - In memory (very fast)

Amazon S3

- All objects you store in S3 will be in buckets
- Buckets store key-object pairs, where the key is a string and the object is whatever type of data you have (like images, XML files, digital music)
- Keys are unique within a bucket, meaning no two objects will have the same key-value pair

Document Database

- Like key-value stores except: value is a document and searchable
- Data stored in nested hierarchies
- Document: JSON, XML, other semi-structured formats
- Any item in a document can be queried
- **Pros**: No object-relational mapping layer, ideal for search
- **Cons**: Complex to implement, incompatible with SQL

Document Database Basics

- Database as storage of a **mass of different documents**
- A document…
  - is a **complex data structure**
  - can **contain completely different data** from other documents
- Document data stores **understand** their documents
  - Queries can **run against values** of document fields
  - **Indexes** can be constructed for document fields
  - Batch style (**mapreduce** etc.) often supported

Suitable Use Cases

- Event logging
- Content management systems, blogging platforms
- Web analytics or real-time analytics
- E-commerce applications

Document Database Example

```
{
"_id": "1",
  "name": "steve",
  "games_owned": [
    {"name":"Super Meat Boy"},
    {"name":"FTL"},
  ],
}
```

```
{
  "_id": "2",
  "name": "darren",
  "handle":"zerocool",
  "games_owned": [
    {"name":“FTL"},
    {"name":“Assassin’s Creed 3“, “dev”: “ubisoft”},
  ],
}
```

Can new attributes be created without the need to define them or to change the existing documents for document DB?

- Yes
- No

Total Results: 0

Can new attributes be created without the need to define them or to change the existing documents for document DB?

Yes

No

Can new attributes be created without the need to define them or to change the existing documents for document DB?

Yes

0%

No

0%

Can new attributes be created without the need to define them or to change the existing documents for document DB?

Yes

0%

No

0%

Document Database Example Systems

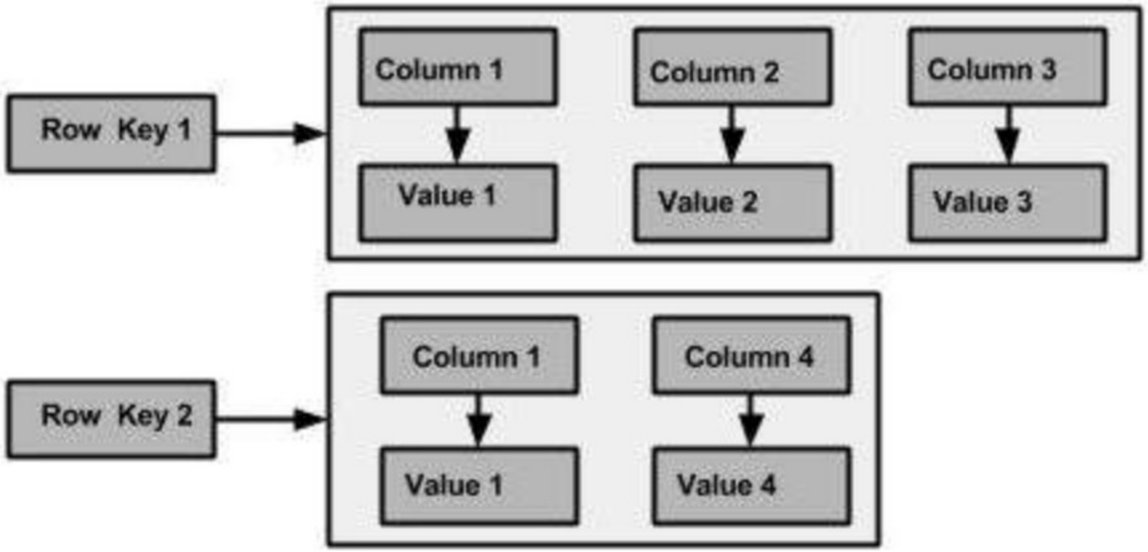
- MongoDB
  - Master/Slave design
  - .find() queries
  - Favours consistency to availability
- CouchDB
  - Only map reduce queries http://sitr.us/2009/06/30/database-queries-the-couchdb-way.html
  - Favours availability to consistency

Column-Family Stores

- Store data in column families as rows
  - A row is a collection of columns associated to a key
  - Each column is a name-value pair
  - Various rows can have different columns
  - A collection of similar rows makes a column family
- Queries can be done on rows, column families and column names
- **Pros**: Good scale out
- **Cons**: Cannot query blob content, row and column designs are critical

Column-Families Data Model

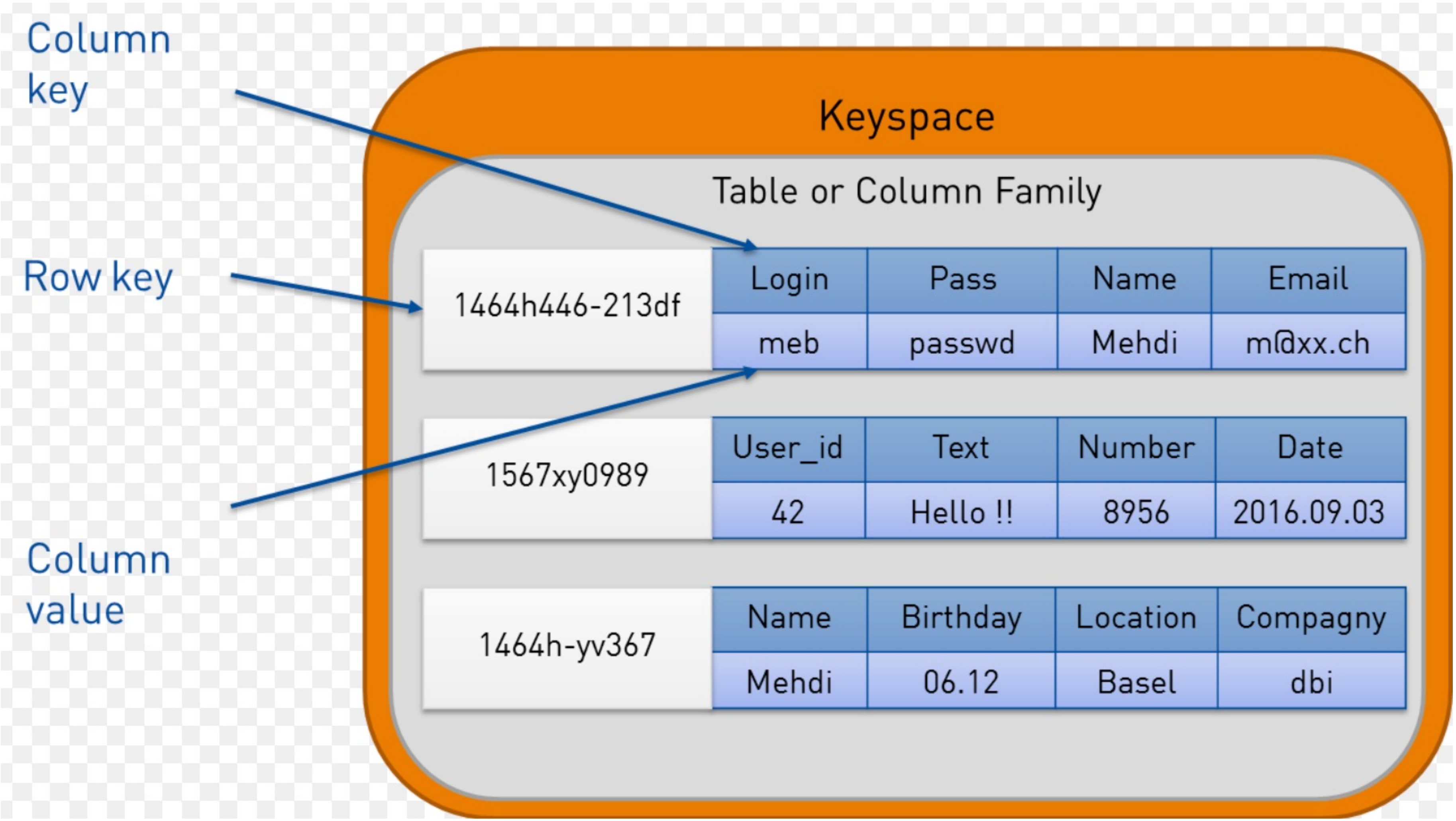
- Column-family



Column-Family Basics

- Entries held in rows
  - Rows have unique **keys**
- Tables define a set of **“column families”**
- Rows contain 0 or more columns for each column family
- No Schema: **Columns** in a **family** change per row
- On Querying:
  - Key lookup is fast
  - Batch processing via map reduce

Column-Family Example

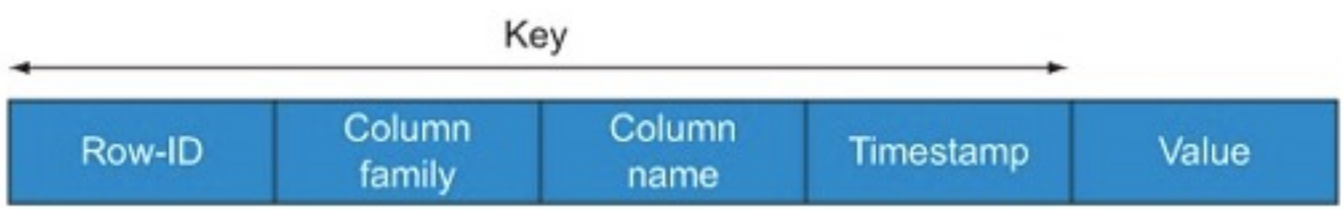


Column-Family Keys

Simplified view: as a spreadsheet. The key for the cell containing “Hello World!” is 3B

	A	B	C
1			
2			
3		Hello World!	
4			
5			
6			

More complicated keys:



Does rowid need to be unique across different column families?

Yes

No

Does rowid need to be unique across different column families?

Yes

0%

No

0%

Does rowid need to be unique across different column families?

Yes

0%

No

0%

Column-Family Example Systems

- Google BigTable
  - Almost all column-family store systems are influenced by BigTable
- HBase
- Cassandra
  - http://cassandra.apache.org/
  - Developed at Facebook
  - The most popular column-family store system (according to http://db-engines.com/en/ranking/wide+column+store, March 2014)

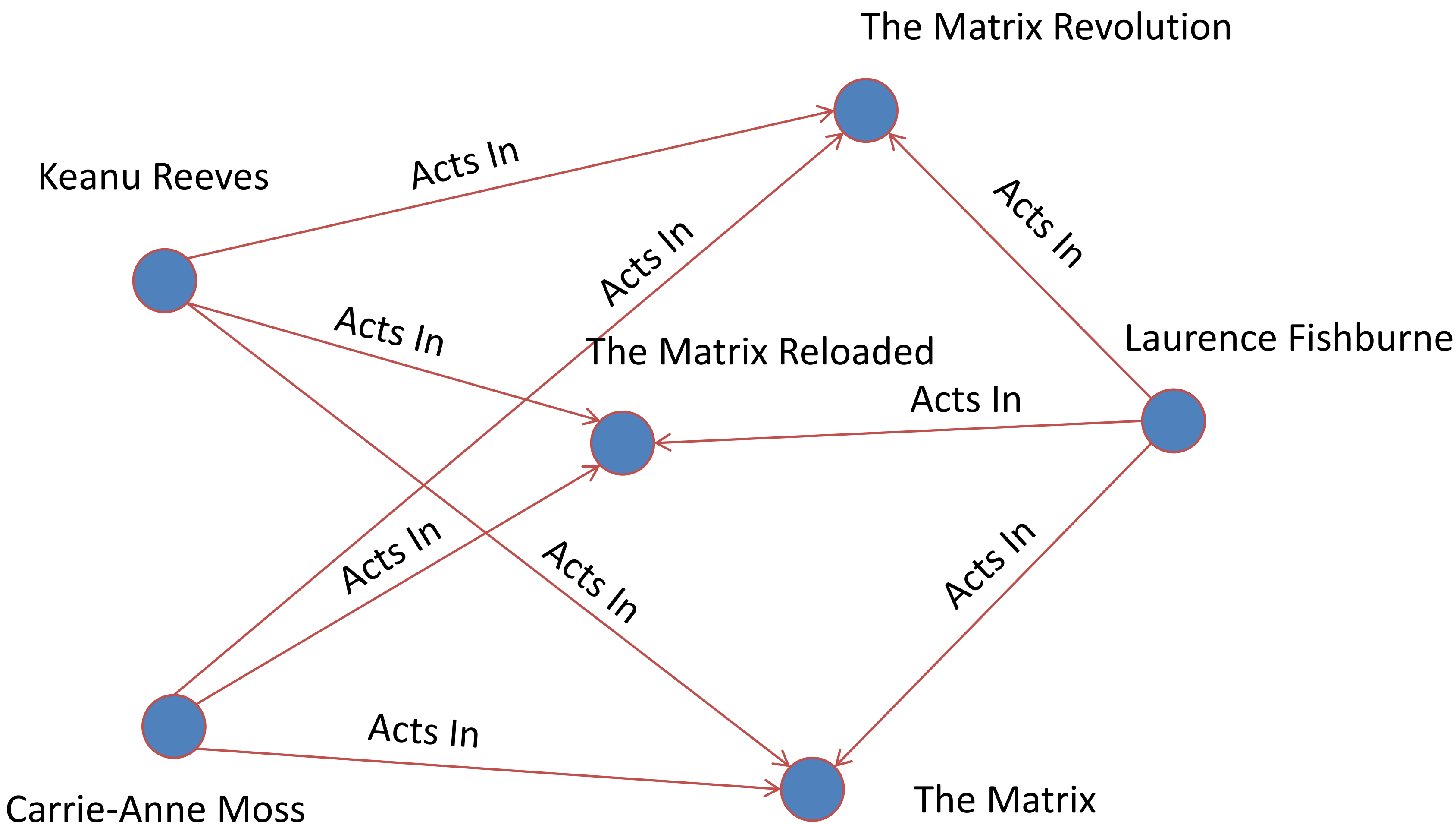
Google apps that use BigTable

- Google Earth
  - RowID for the longitude portion of the map
  - Column name for latitude
    - One map for each square mile on Earth, you could have 15,000 distinct row IDs and 15,000 distinct column IDs

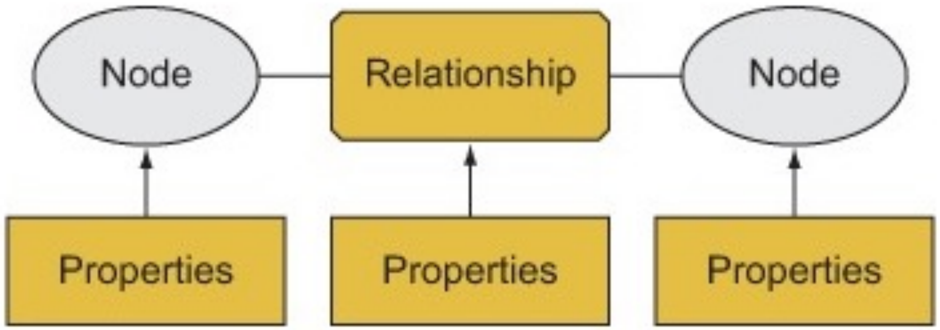
Graph Store

- Two main components
  - **Nodes:** entities with properties
  - **Edges:** relationships with properties **(**An edge can have a direction)
- Queries are really graph traversals
- Ideal when relationships between data is key, e.g. **social networks (clique identification)**
- **Pros**: Fast network search
- **Cons**: Can have poor scalability when graphs don't fit into one machine, specialized query language (these can sometimes be addressed by Hadoop and other solutions)

Graph Store Basics



Graph Data

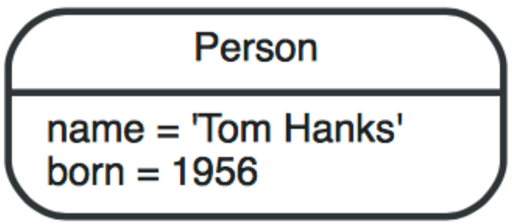


Social network
Internet
Human cells
Brain
Road network
⋮

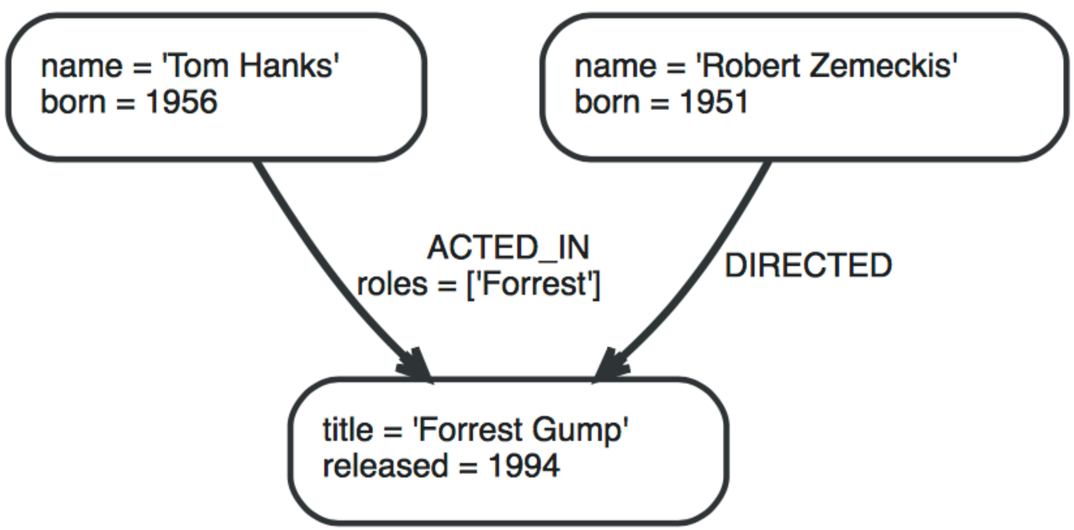
- Query examples:
  - Mutual friends between two people?
  - Find "influencers" in a group (to target as potential customers)

Graph Store System Example

- Neo4j
  - Nodes can have labels



  - Relationships are directional



Neo4j Characteristics

- Not distributed
- ACID transactions
- Cypher for query:

```
START m=node:node_auto_index(id="603")
MATCH a-[:ACTS_IN]->m
RETURN a;
```

Neo4j
the graph database

NoSQL DB Types, Usages, Examples

Type	Typical usage	Examples
Key-value store—A simple data storage system that uses a key to access a value	• Image stores • Key-based filesystems • Object cache • Systems designed to scale	• Berkeley DB • Memcache • Redis • Riak • DynamoDB
Column family store—A sparse matrix system that uses a row and a column as keys	• Web crawler results • Big data problems that can relax consistency rules	• Apache HBase • Apache Cassandra • Hypertable • Apache Accumulo
Graph store—For relationship-intensive problems	• Social networks • Fraud detection • Relationship-heavy data	• Neo4j • AllegroGraph • Bigdata (RDF data store) • InfiniteGraph (Objectivity)
Document store—Storing hierarchical data structures directly in the database	• High-variability data • Document search • Integration hubs • Web content management • Publishing	• MongoDB (10Gen) • CouchDB • Couchbase • MarkLogic • eXist-db • Berkeley DB XML

NoSQL Summary

- NoSQL databases reject:
  - Overhead of ACID transactions
  - “Complexity” of SQL
  - Burden of up-front schema design
  - Declarative query expression

SQL vs. NoSQL Summary

- SQL Databases
  - Predefined Schema
  - Standard definition and interface language
  - Tight consistency
  - Fine-grained security on columns/rows using views
  - Well defined semantics
- NoSQL Database
  - No predefined Schema
  - Per-product definition and interface language
  - Getting an answer quickly is more important than getting a correct answer
  - Query standards have yet to be established

Bottom Line

- Relational databases will continue to be an appropriate solution to many business problems for the foreseeable future
- RDBMSs are continuing to evolve and are making it possible to relax ACID requirements and manage document-oriented structures
- But there are situations where relational databases aren’t the best match for a business problem
  - Need to consider nature of data, business requirements, and trade-offs between consistency, availability and scalability