An introduction to NoSQL databases

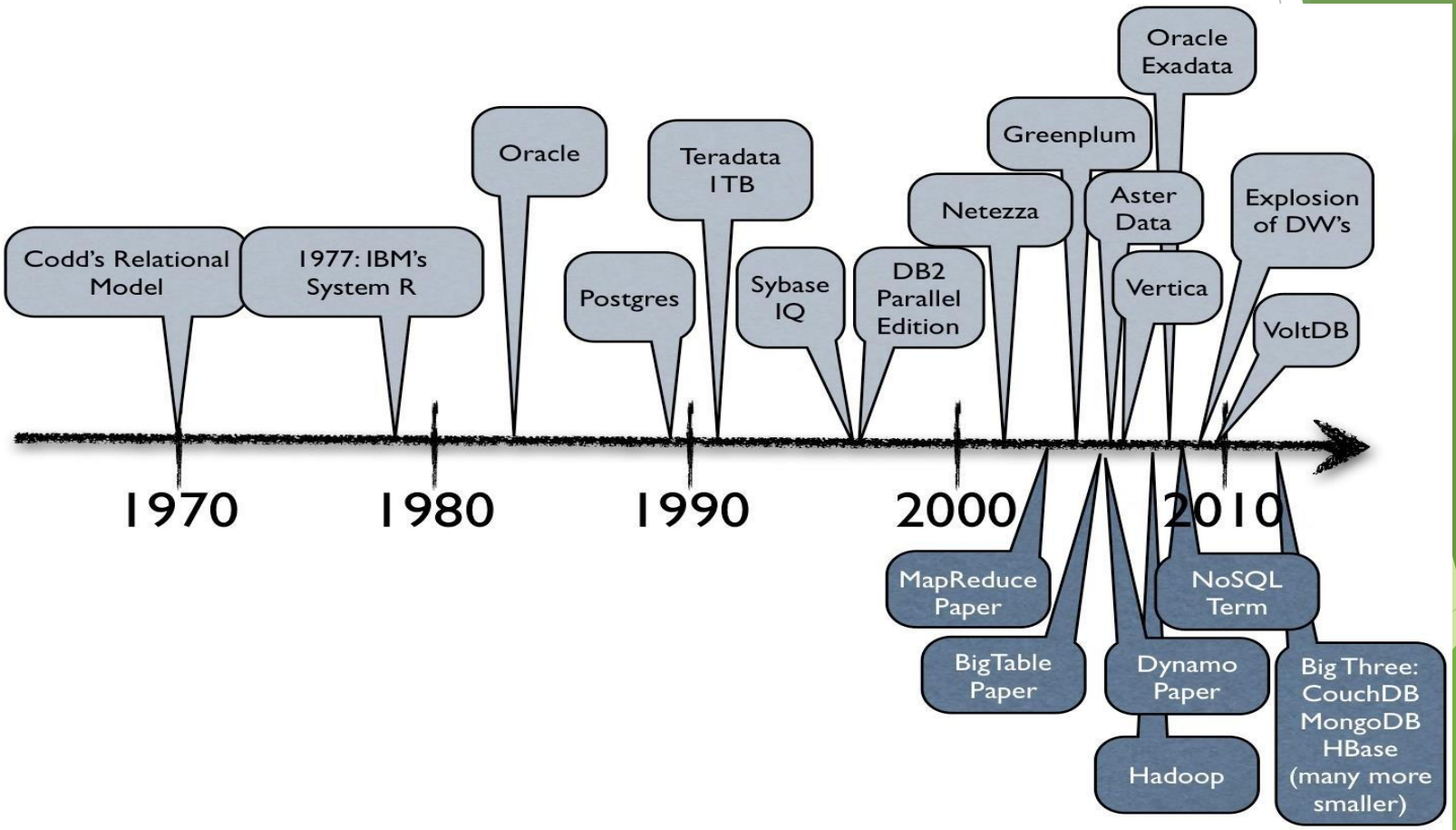
Contents

- History of databases
- The Value of Relational Databases
- The Emergence of NoSQL
- Aggregate data models
- Distribution models
- Beyond NoSQL
- Choosing your Database

Introduction

- Database - Organized collection of data
- DBMS - Database Management System: a software package with computer programs that controls the creation, maintenance and use of a database
- Databases are created to operate large quantities of information by inputting, storing, retrieving, and managing that information.

Abrief history



Market Needs

- Database capable to handle explosion of social media sites(FB,Twitter) with large data needs.
- Explosion of storage needs in large websites Google, Yahoo.
- To work on clusters.
- Capable to tune with 21st century.
- Schema less, without large relations, open source.

- Reasons to use No-SQL:
- Analytical
- Scale
- Redundancy
- Flexibility
- Rapid development
- **Why No-SQL?**
- Easy and ready to manage with clusters.
- Suitable for upcoming data explosions.
- Not required to keep track with data structure.
- Provide easy and flexible system.

- What is No-SQL?
- Not Only SQL, implying that when designing a software solution or product, there are more than one storage mechanism that could be based on needs.
- Rise of No-SQL is Polyglot Persistence.
- Polyglot Programming – Applications written in a mix of language to take advantage of different language and suitable for handling different problems.

- Where No-SQL is used?
- Google (Bigtable(2004), LevelDB)
- LinkedIn (Voldemort(2008))
- Facebook (Cassandra)
- Twitter (Hadoop /Hbase, FlockDB, Cassandra(2008))
- Netflix (SimpleDB, Hadoop/Hbase, Cassandra)
- CERN (CouchDB(2005))

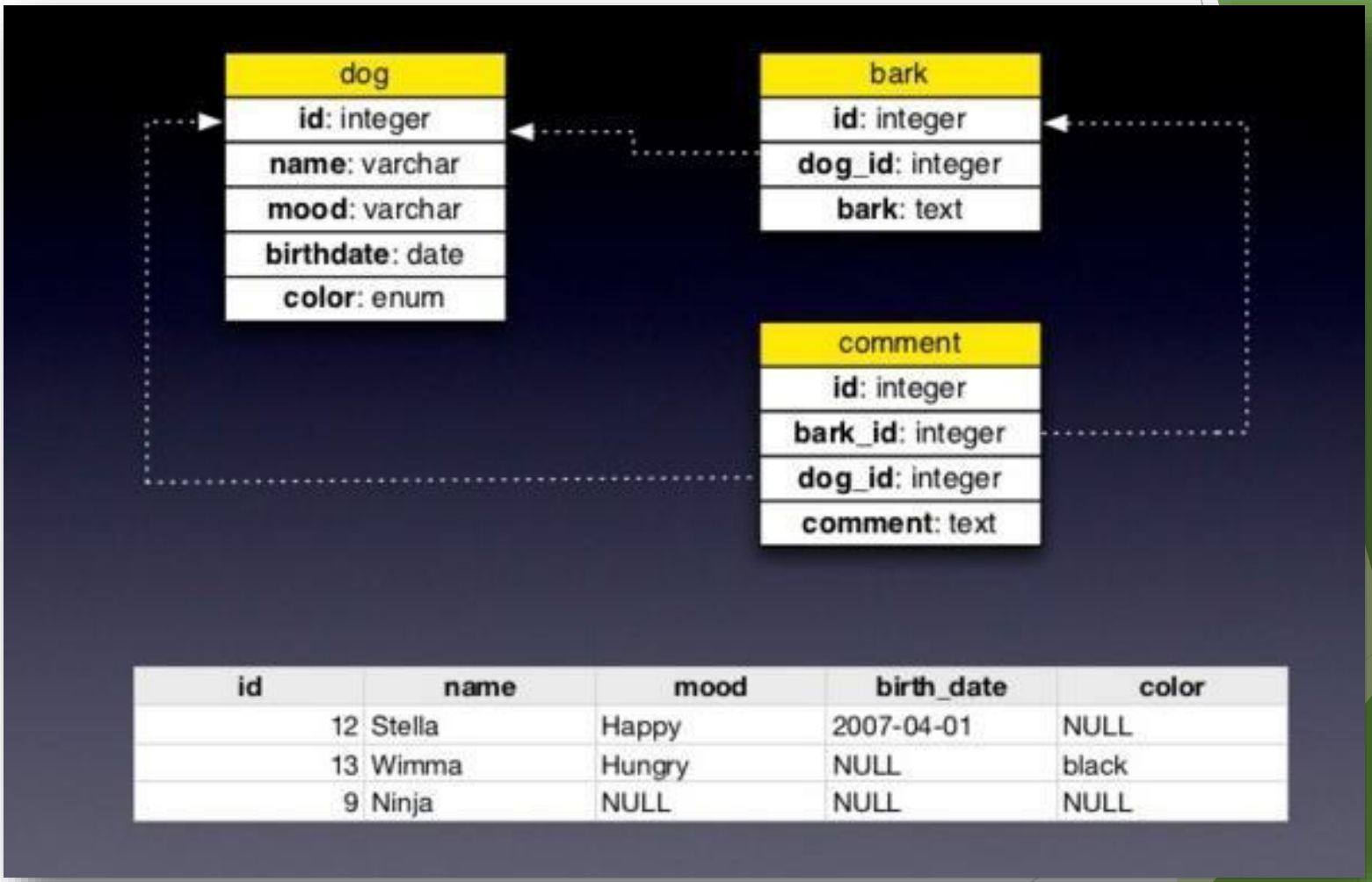
The Value of Relational databases

- Benefits of Relational databases:

- Getting at Persistent data
- Concurrency
- Integration
- A standard model
- Impedance mismatch
- Application and Integration Databases
- Attack of the clusters

SQL databases

RDBMS



id	name	mood	birth_date	color
12	Stella	Happy	2007-04-01	NULL
13	Wimma	Hungry	NULL	black
9	Ninja	NULL	NULL	NULL

Emergence of NoSQL Databases

But...

- Relational databases were not built for **distributed applications.**

Because...

- Joins are expensive
- Hard to scale horizontally
- Impedance mismatch occurs
- Expensive (product cost, hardware, Maintenance)

Era of Distributed Computing

NoSQL why, what and when?

But…

- Relational databases were not built for **distributed applications.**

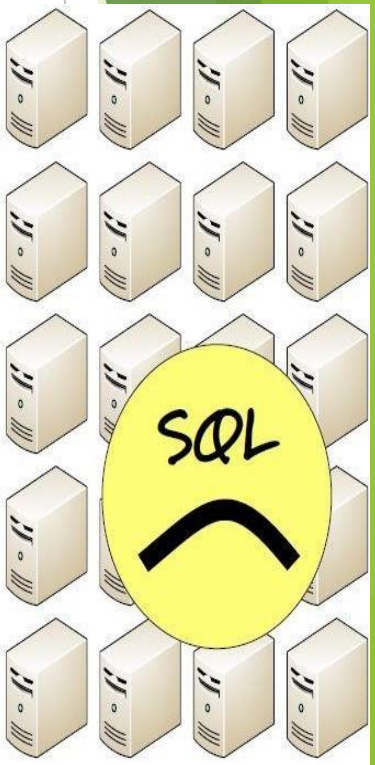
Because…

- Joins are expensive
- Hard to scale horizontally
- Impedance mismatch occurs
- Expensive (product cost, hardware, Maintenance)

And….

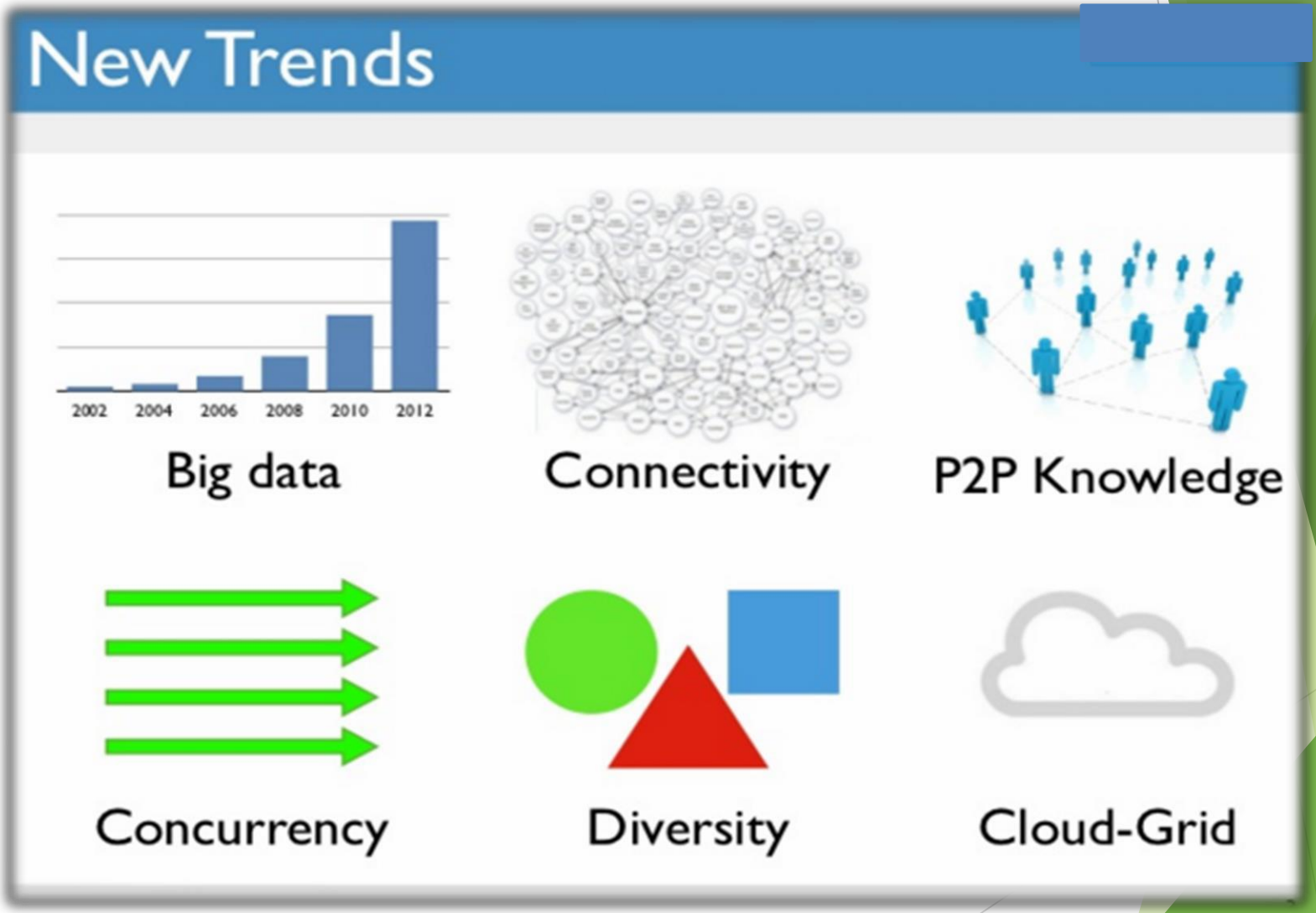
It's weak in:

- Speed (performance)
- High availability
- Partition tolerance



Why NOSQL?

Ans. Driving Trends



What is NoSQL?

- A No SQL database provides a mechanism for storage and retrieval of data that employs less constrained consistency models than traditional relational database

- No SQL systems are also referred to as "Not only SQL" to emphasize that they do in fact allow SQL-like query languages to be used.

Characteristics of NoSQL databases

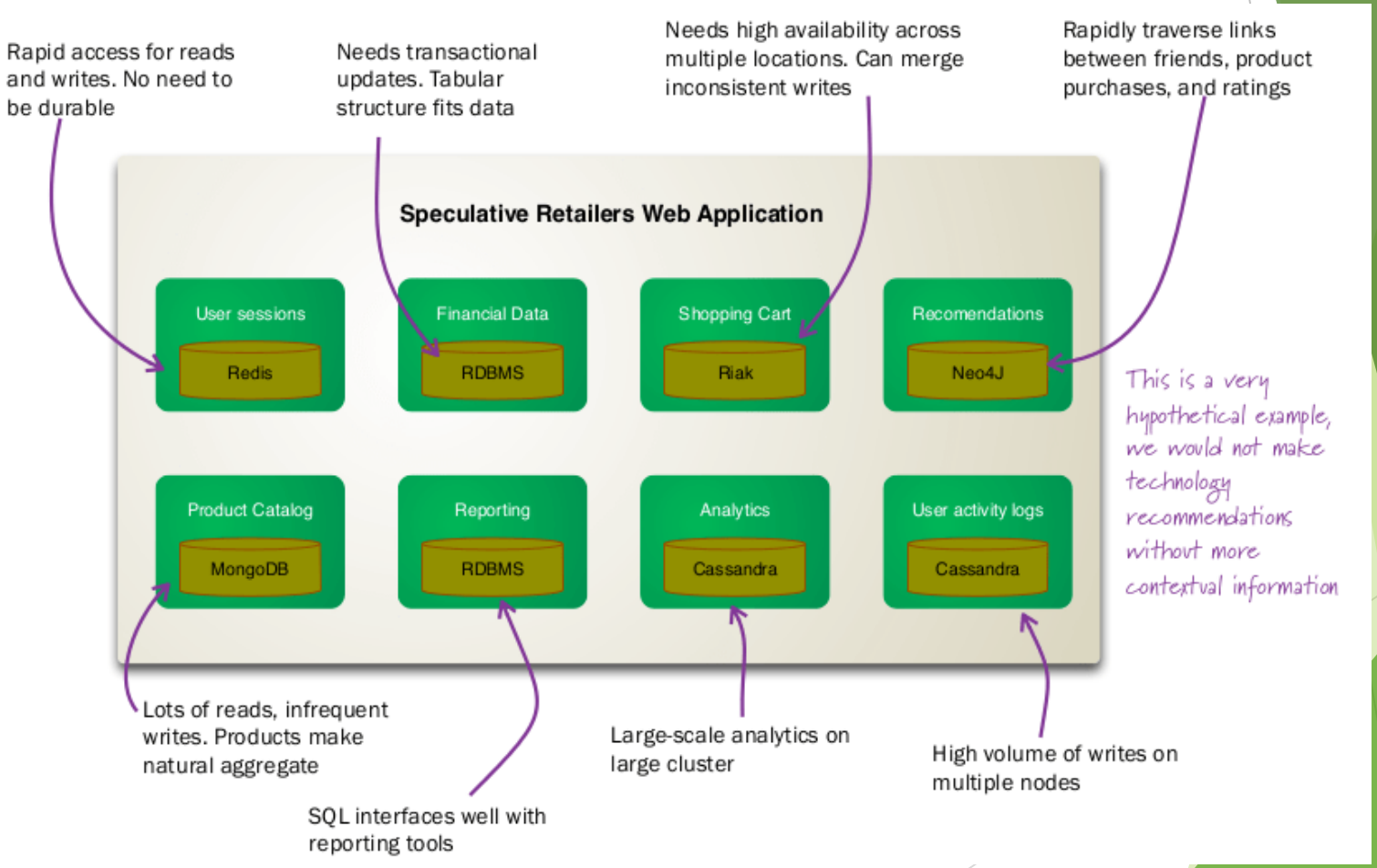
The common characteristics of NoSQL databases are

- Not using the relational model
- Running well on clusters
- Open-source
- Built for the 21st century web estates
- Schema less
- The most important result of the rise of NoSQL is Polyglot Persistence.

polyglot persistence—using different data stores in different circumstances. picking a database that understand the nature of the data we're storing and how to manipulate it. The result is that most organizations will have a mix of data storage technologies for different circumstances.

In order to make this polyglot world work, organizations also need to shift from integration databases to application databases.

Polyglot Persistence for Retailers web application



Characteristics of NoSQL databases

NoSQL avoids:

- Overhead of ACID transactions
- Complexity of SQL query
- Burden of up-front schema design
- DBA presence
- Transactions (It should be handled at application layer)

Provides:

- Easy and frequent changes to DB
- Fast development
- Large data volumes (eg. Google)
- Schema less

NoSQL why, what and when?

When and when not to use it?

WHEN / WHY ?

- When traditional RDBMS model is too restrictive (flexible schema)
- When ACID support is not "really" needed
- Object-to-Relational (O/R) impedance
- Because RDBMS is neither distributed nor scalable by nature
- Logging data from distributed sources
- Storing Events / temporal data
- Temporary Data (Shopping Carts / Wish lists / Session Data)
- Data which requires flexible schema
- **Polyglot Persistence** i.e. best data store depending on nature of data.

WHEN NOT ?

- Financial Data
- Data requiring strict ACID compliance
- Business Critical Data

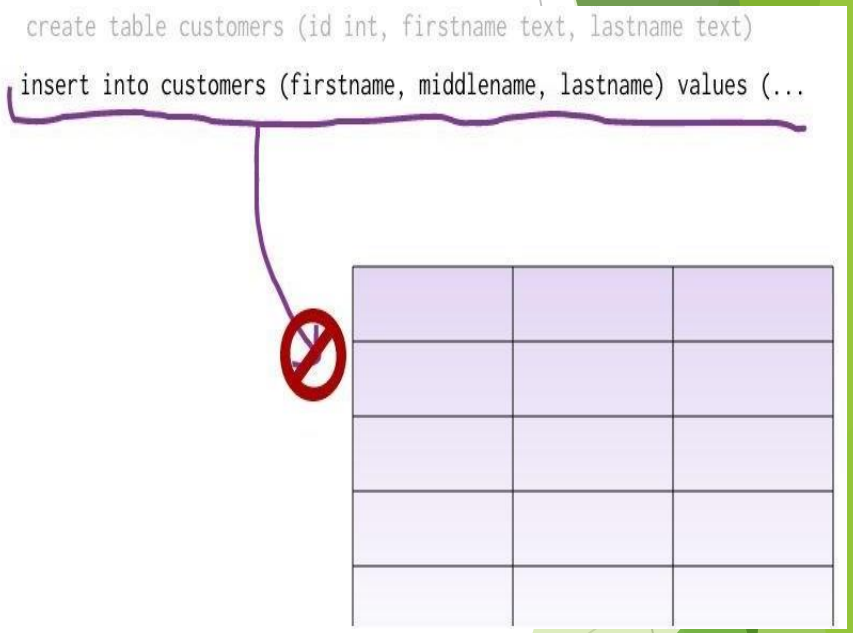
NoSQL is getting more & more popular

- Relational databases have been a successful technology for twenty years, providing persistence,concurrency control, and an integration mechanism.
- • Application developers have been frustrated with the impedance mismatch between the relational model and the in-memory data structures.
- • There is a movement away from using databases as integration points towards encapsulating databases within applications and integrating through services.
- • The vital factor for a change in data storage was the need to support large volumes of data by running on clusters. Relational databases are not designed to run efficiently on clusters.
- • NoSQL is an accidental neologism. There is no prescriptive definition—all you can make is an observation of common characteristics.

What is a schema-less datamodel?

In relational Databases:

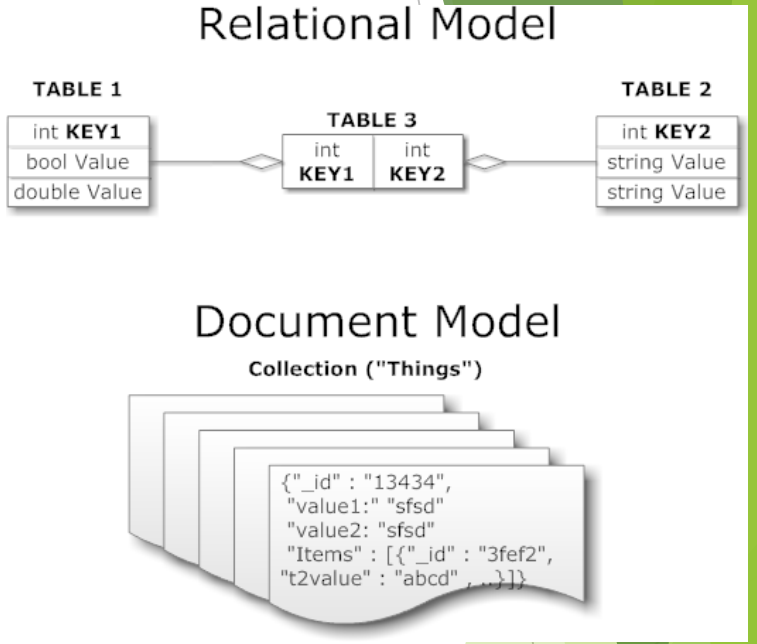
- You can't add a record which does not fit the schema
- You need to add NULLs to unused items in a row
- We should consider the datatypes. i.e : you can't add a stirng to an interger field
- You can't add multiple items in a field (You should create another table: primary-key, foreign key, joins, normalization, ... !!!)



What is a schema-less datamodel?

In NoSQL Databases:

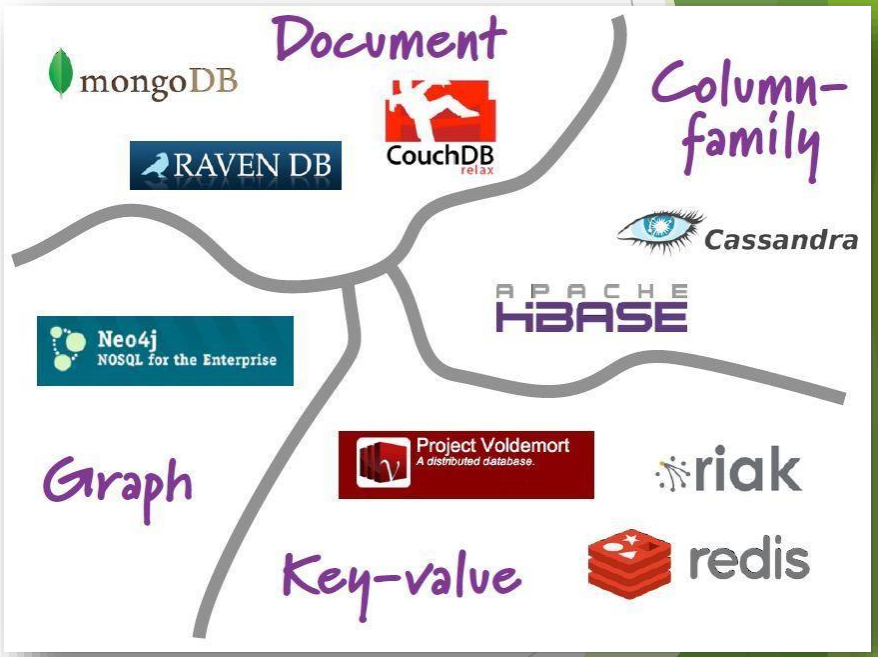
- There is no schema to consider
- There is no unused cell
- There is no datatype (implicit)
- Most of considerations are done in application layer
- We gather all items in an aggregate (document)



Aggregate Data Models

NoSQL databases are classified in four major datamodels:

- Key-value
- Document
- Column family
- Graph

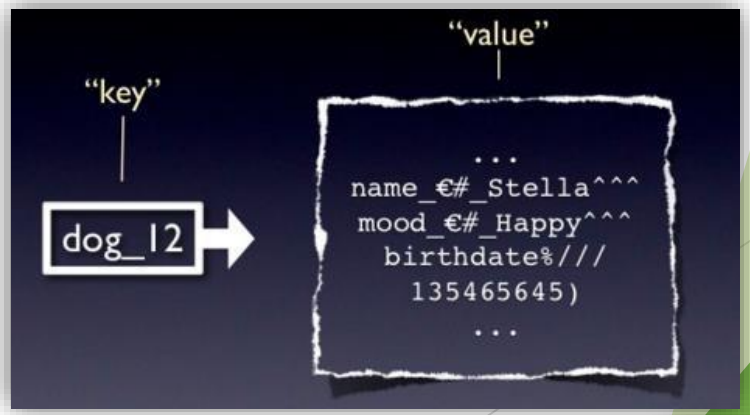


Each DB has its own query language

Key-value data model

- Simplest NOSQL databases
- The main idea is the use of a hash table
- Access data (values) by strings called keys
- Data has no required format data may have any format
- Data model: (key, value) pairs
- Basic Operations:
  Insert(key,value),
  Fetch(key),
  Update(key),
  Delete(key)

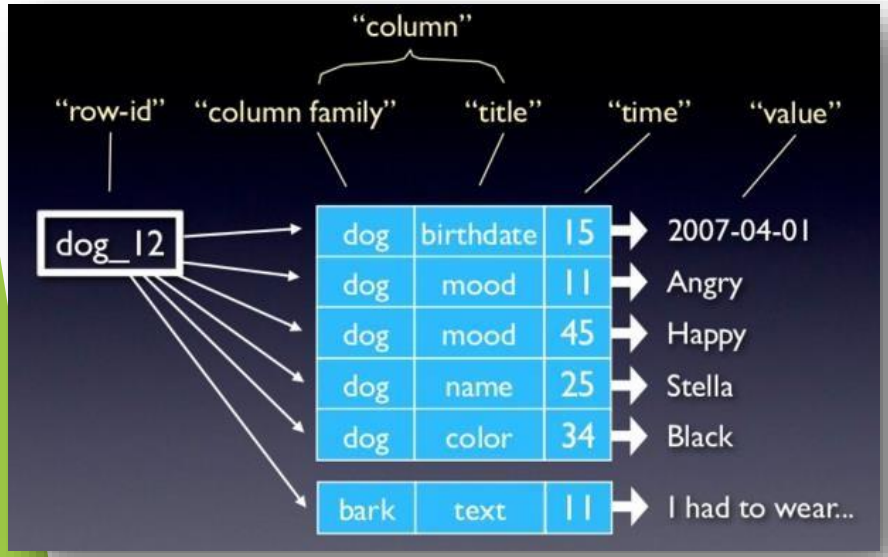
Car	
Key	Attributes
1	Make: Nissan Model: Pathfinder Color: Green Year: 2003
2	Make: Nissan Model: Pathfinder Color: Blue Color: Green Year:2005 Transmission: Auto



amazon DynamoDB

Column family data model

- The column is lowest/smallest instance of data.
- It is a tuple that contains a name, a value and a timestamp



ColumnFamily: Authors	
Key	Value
"Eric Long"	Columns Name: Value "email": "eric (at) long.com" "country": "United Kingdom" "registeredSince": "01/01/2002"
"John Steward"	Columns Name: Value "email": "john.steward (at) somedomain.com" "country": "Australia" "registeredSince": "01/01/2009"
"Ronald Mathies"	Columns Name: Value "email": "ronald (at) sodeso.nl" "country": "Netherlands, The" "registeredSince": "01/01/2010"

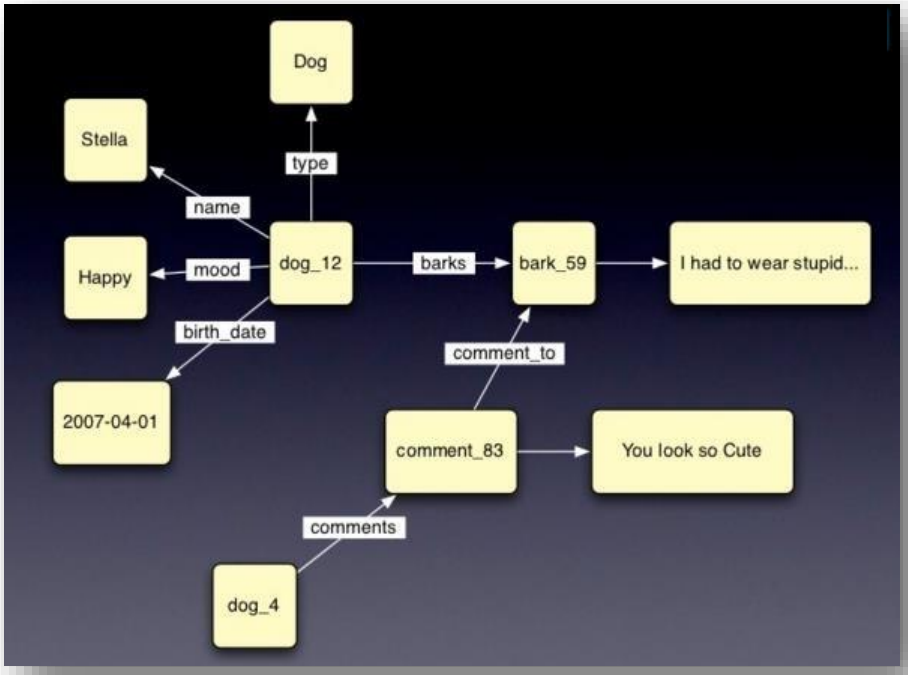
Column family data model

Some statistics about Facebook Search (using Cassandra)

- MySQL > 50 GB Data
  - Writes Average : ~300 ms
  - Reads Average : ~350 ms
- Rewritten with Cassandra > 50 GB Data
  - Writes Average : 0.12 ms
  - Reads Average : 15 ms

Graph data model

- Based on Graph Theory.
- Scale vertically, no clustering.
- You can use graph algorithms easily
- Transactions
- ACID

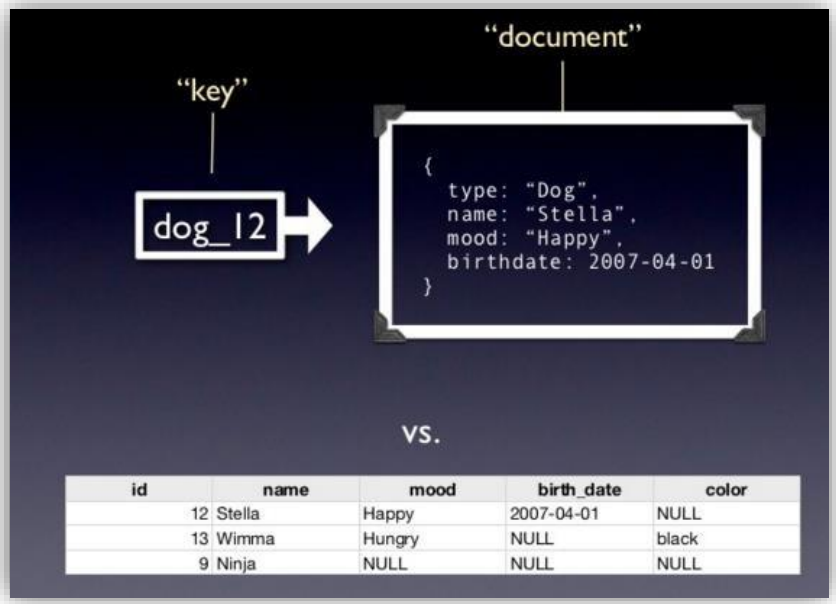


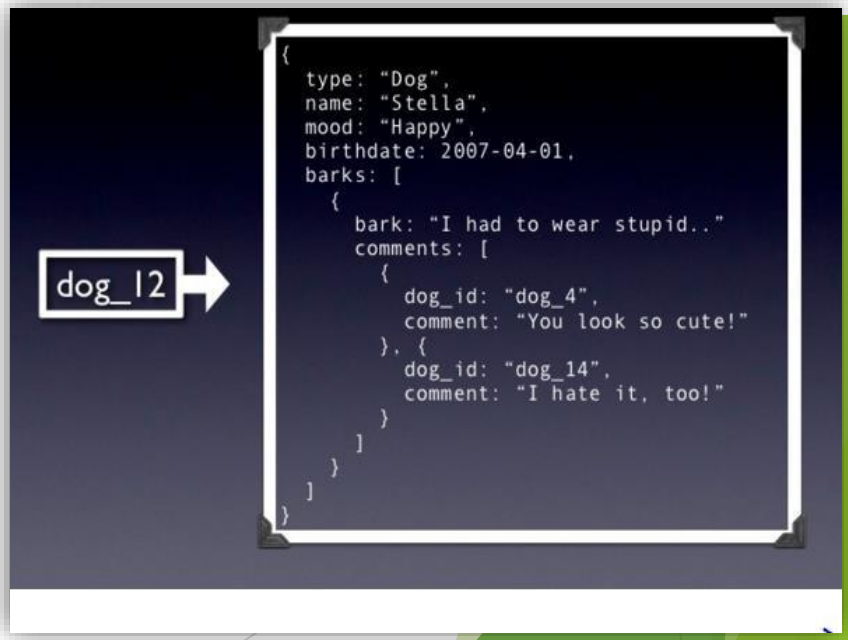
neo4j

Document based data model

- Pair each key with complex data structure known as data structure.
- Indexes are done via B-Trees.
- Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.

```
{
  person: {
    first_name: "Peter",
    last_name: "Peterson",
    addresses: [
      {street: "123 Peter St"},
      {street: "504 Not Peter St"}
    ],
  }
}
```





Document based data model

```
SELECT  name, pic, profile_url
FROM    user
WHERE   uid = me()
```


```
SELECT  message, attachment
FROM    stream
WHERE   source_id = me() AND type = 80
```

```
SELECT  name
FROM    friendlist
WHERE   owner = me()
```

```
SELECT  name
FROM    group
WHERE   gid IN ( SELECT  gid
                 FROM    group_member
                 WHERE   uid = me() )
```

```
SELECT  name, pic
FROM    user
WHERE   online_presence = "active"
        AND
        uid IN ( SELECT  uid2
                 FROM    friend
                 WHERE   uid1 = me() )
```

https://developers.facebook.com/docs/reference/fql/

Benefits of Aggregate data models

- An aggregate is a collection of data that we interact with as a unit. Aggregates form the boundaries for ACID operations with the database.
- Key-value, document, and column-family databases can all be seen as forms of aggregate oriented database.
- Aggregates make it easier for the database to manage data storage over clusters.
- Aggregate-oriented databases work best when most data interaction is done with the same aggregate; aggregate-ignorant databases are better when interactions use data organized in many different formations.

SQL vs NOSQL

Differences

	SQL Databases	No SQL Database
Example	Oracle , mysql	Mondo DB, CouchDB, Neo4J
Storage Model	Rows and tables	Key-value. Data stored as single document in JSON, XML
Schemas	Static	Dynamic
Scaling	Vertical & Horizontal	Horizontal
Transactions	Yes	Certain levels
Data Manipulation	Select, Insert , Update	Through Object Oriented API's

Distribution Models

- The primary driver of interest in NoSQL has been its ability to run databases on a large cluster. As data volumes increase, it becomes more difficult and expensive to scale up.
- Depending on your distribution model, a data store will be able to handle larger quantities of data, able to process a greater read or write traffic, or more availability in the face of network slowdowns or breakages.
- Broadly, there are two paths to data distribution: Replication and Sharding. Replication takes the same data and copies it over multiple nodes. Sharding puts different data on different nodes.

- Replication comes into two forms: master-slave and peer-to-peer.
  - **Single-server**
  - **Master-slave replication**
  - **Sharding**
  - **Peer-to-peer replication**
- **Single Server**: Run the database on a single machine that handles all the reads and writes to the data store.
- Graph databases are the obvious category here—these work best in a single-server configuration.
- If your data usage is mostly about processing aggregates, then a single-server document or key-value store may well be worthwhile because it's easier on application developers.

Sharding:

- A busy data store is busy often, because different people are accessing different parts of the dataset.
- In these circumstances horizontal scalability can be supported by putting different parts of the data onto different servers—a technique that's called sharding.
- Many NoSQL databases offer **auto-sharding**, where the database takes on the responsibility of allocating data to shards and ensuring that data access goes to the right shard. This can make it much easier to use sharding in an application.

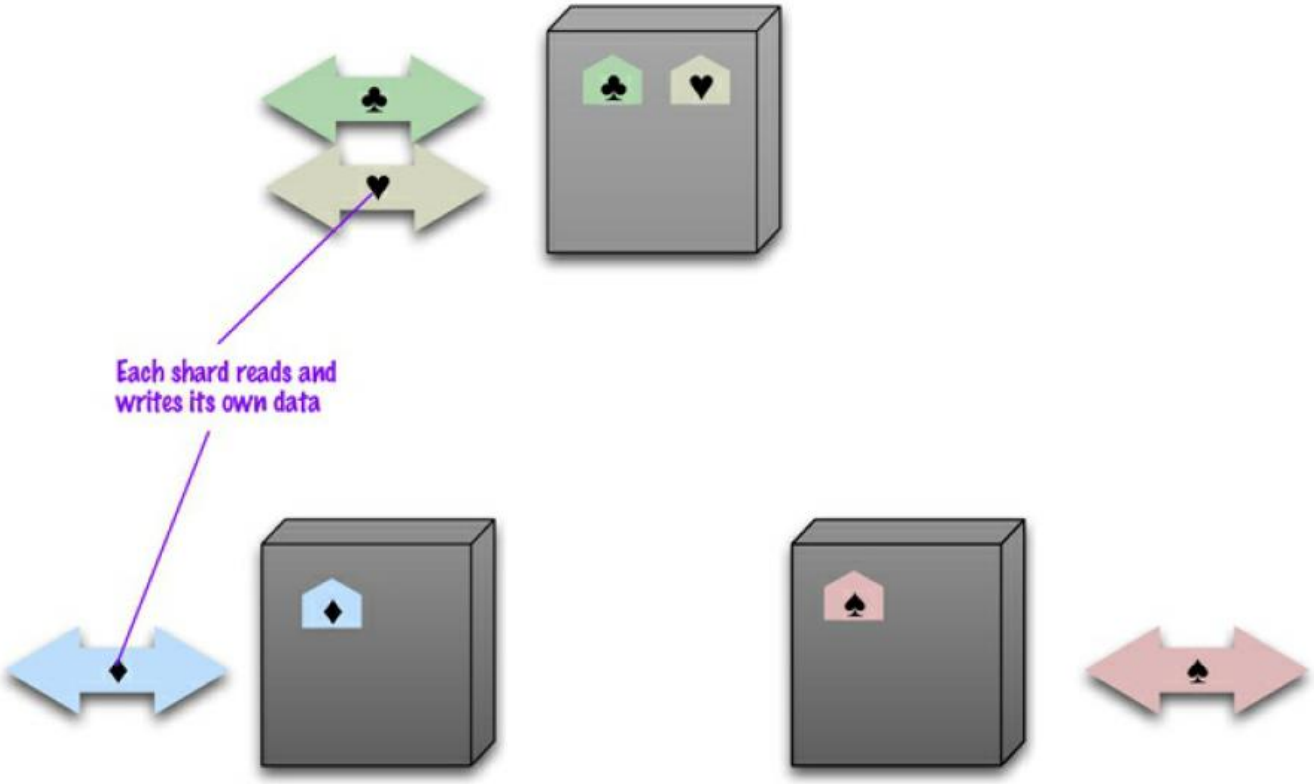


Fig: Sharding puts different data on separate nodes, each of which does its own reads and writes.

Master - Slave Replication

- Replicate data across multiple nodes. One node is designated as the master, or primary. This master is the authoritative source for the data.
- Master is responsible for processing any updates to that data.
- The other nodes are slaves, or secondaries. A replication process synchronizes the slaves with the master.
- Master-slave replication is most helpful for scaling when you have a read-intensive dataset.
- A second advantage of master-slave replication is **read resilience**:
- Even if the master fail, the slaves can still handle read requests.
- It is useful if most of your data access is reads.
- Consequently it isn't such a good scheme for datasets with heavy write traffic, although offloading the read traffic will help a bit with handling the write load.

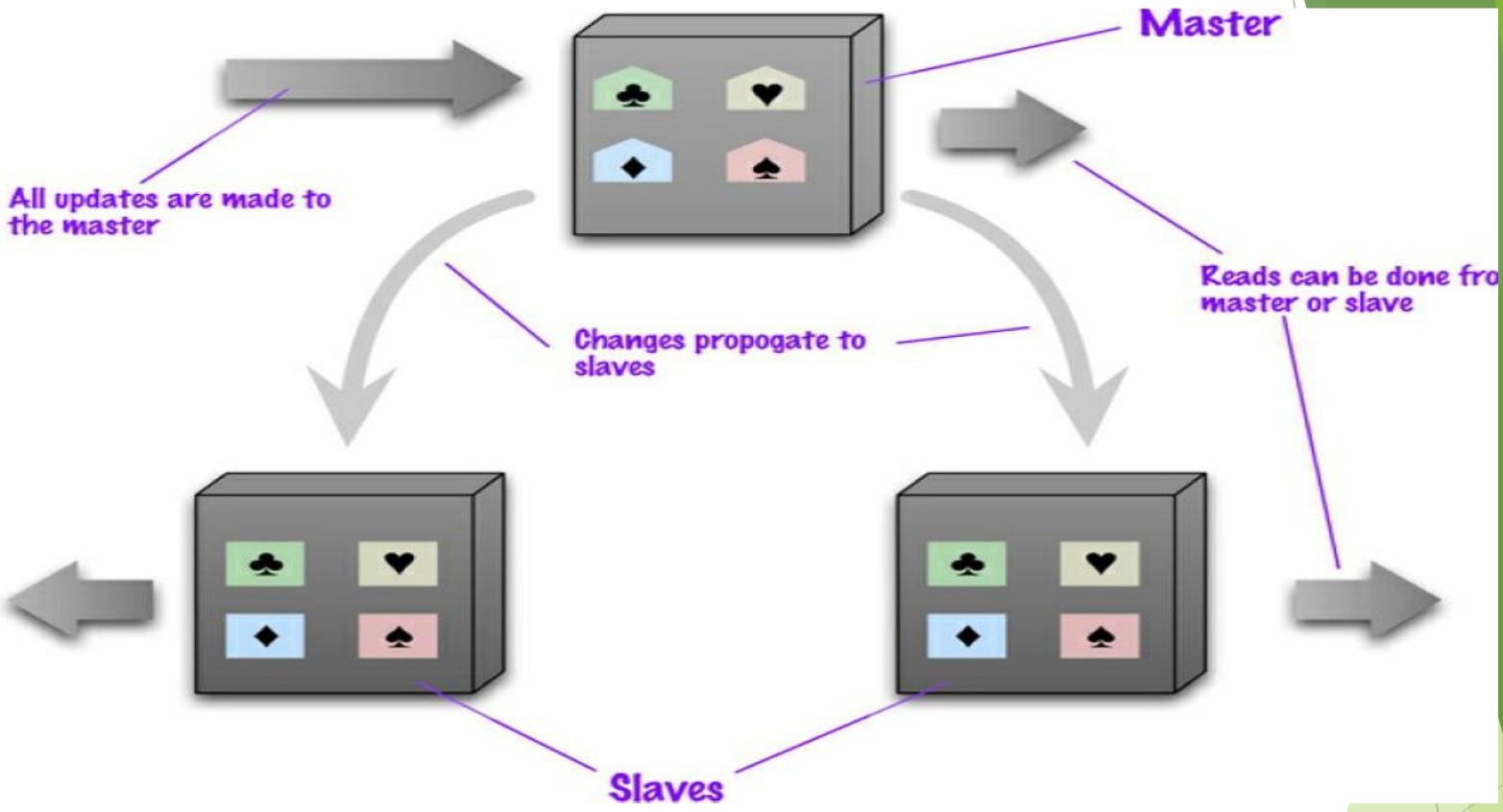


Fig: Data is replicated from master to slaves. The master services all writes; reads may come from either master or slaves.

Peer to Peer Replication

- In Peer-to-peer replication does not have a master.
- All the replicas have equal weight, they can all accept writes, and the loss of any of them doesn't prevent access to the data store.
- With a peer-to-peer replication cluster, user can ride over node failures without losing access to data.
- Furthermore, nodes can be easily added to improve your performance.
- **Inconsistency**: When you can write to two different places, two people will attempt to update the same record at the same time—a write-write conflict. Inconsistencies on read lead to problems but at least they are relatively transient.
- There are contexts with policy to merge inconsistent writes. This benefit in writing to any replica. These point trade off consistency for availability.

Beyond NoSQL

- File Systems
- Event Sourcing
- Memory Image
- Version Control
- XML Databases
- Object Databases

Choosing your Database

Features to choose database are:

- Programmer Productivity
- Data Access Performance
- Sticking with the default
- Hedging your bets

The two main reasons to use NoSQL technology are:

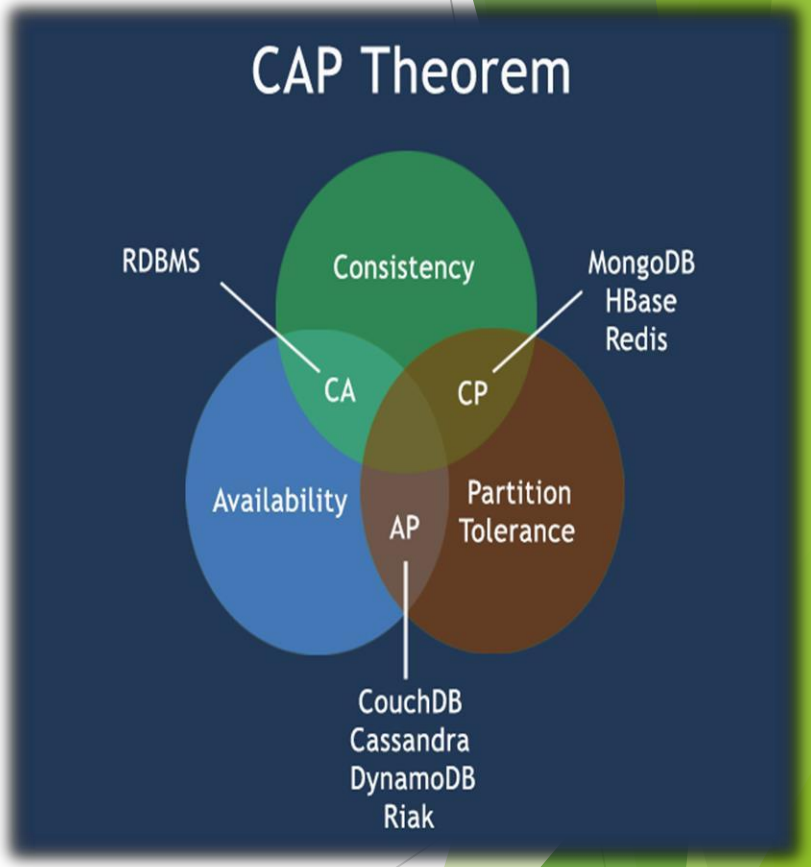
- To improve programmer productivity by using a database that better matches an application's needs.
- To improve data access performance via some combination of handling larger data volumes, reducing latency, and improving throughput.

- It’s essential to test your expectations about programmer productivity and/or performance before committing to using a NoSQL technology.
- Service encapsulation supports changing data storage technologies as needs and technology evolve.
- Separating parts of applications into services also allows you to introduce NoSQL into an existing application.
- Most applications, particularly nonstrategic ones, should stick with relational technology—at least until the NoSQL ecosystem becomes more mature.

CAP theorem

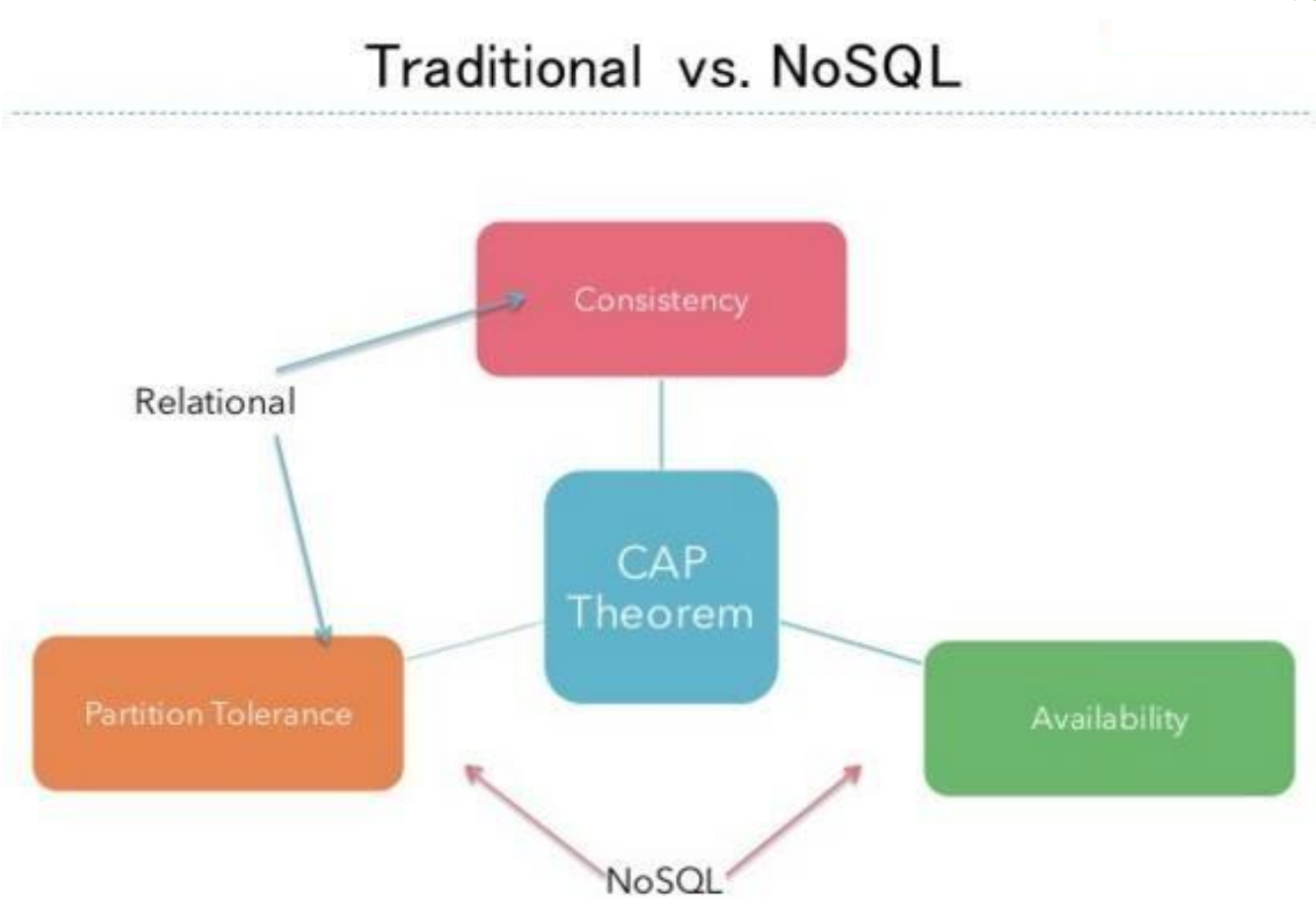
The CAP Theorem

- Impossible for any shared data-system to guarantee simultaneously all of the following three properties:
  - Consistency – once data is written, all future read requests will contain that data
  - Availability – the database is always available and responsive
  - Partition Tolerance – if part of the database is unavailable, other parts are unaffected



We can not achieve all the three items
In distributed database systems (center)

CAP theorem


Traditional vs. NoSQL
Consistency
Relational
CAP Theorem
Partition Tolerance
Availability
NoSQL