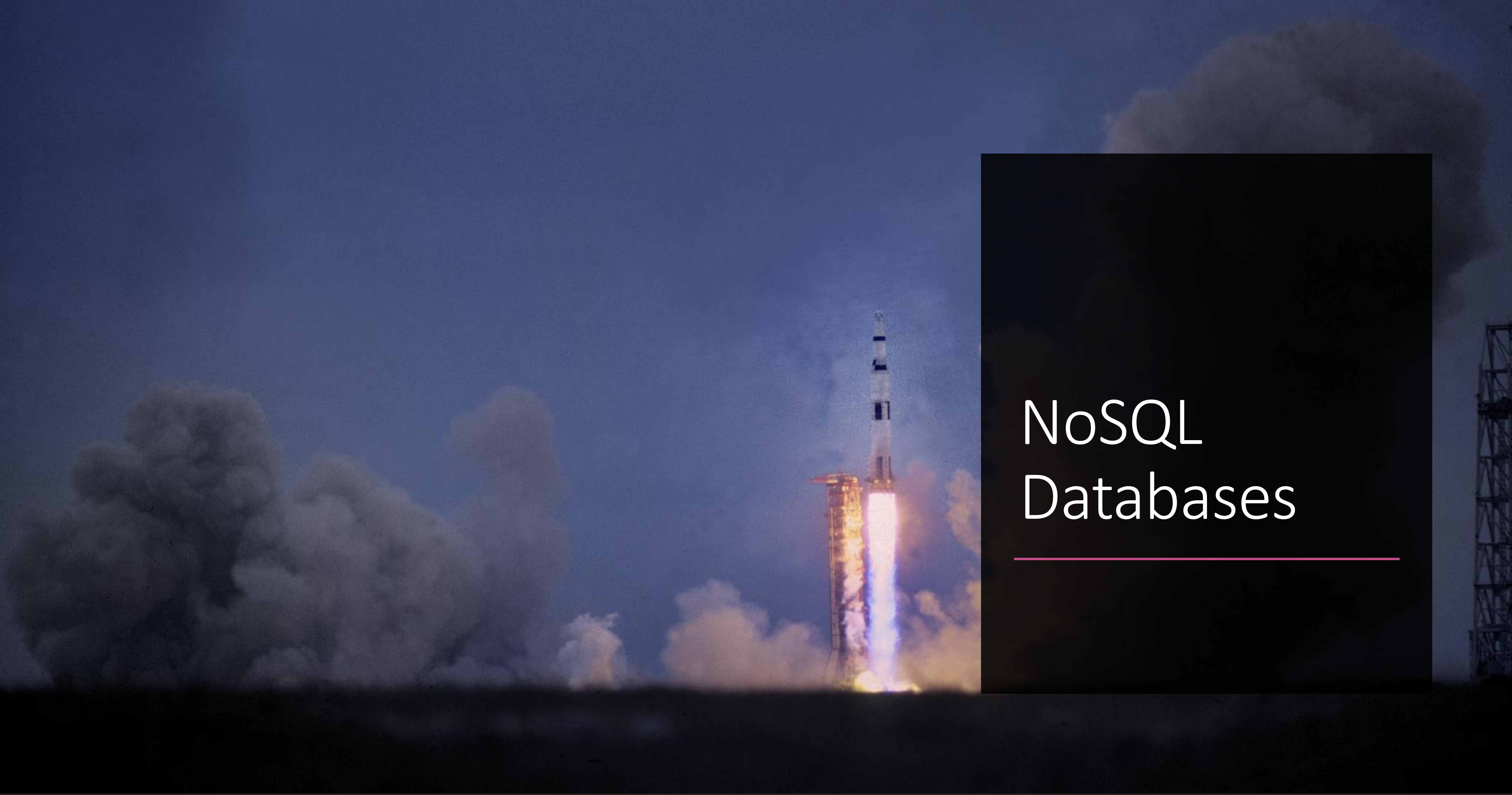


NoSQL Databases

Why NoSQL?

Volume and velocity of data generation has exploded

NoSQL resurfaced as a way to address growing data needs

Typically distributed implementations (no single node can manage the volume of data)

Typically sacrifices some features of traditional relational systems to support massive scale of data
- Over time, more features have been incorporated

Not necessarily a substitute for traditional SQL DB

NoSQL databases

Main problems with SQL

1. You don’t have your data in one computer not even in one network

2. You’re not the owner of all the data

3. You can’t put the data in one place

4. It’s uncordinated in time and space

5. It’s not always well structured

Main characteristics of NoSQL

Data Models. In RDBMS, the data schema must be defined first, while in NoSQL, the data schema is dynamic.

Information Structure. Management of unstructured data, which is closer to the type of information used today. For example: email, text, social media posts, video.

Licensing. The vast majority of NoSQL systems are open source.

Consistency

Consistency refers to reliability of functions' performance

- Read operation returns value of last write, all read operations performed at the same time epoch return the same value (regardless of where they were initiated)

Different range of consistency models:

- Strong: Updates are ordered and reads reflect latest update
- Timeline: Updates are ordered in all replicants, but reads at a given replicant might be stale
- Eventual: No guarantees about updates being applied in order in all replicants, reads might be stale and no guarantee when it will reflect latest updates

Availability and Partition Tolerance

Availability: A system’s ability to complete a certain operation

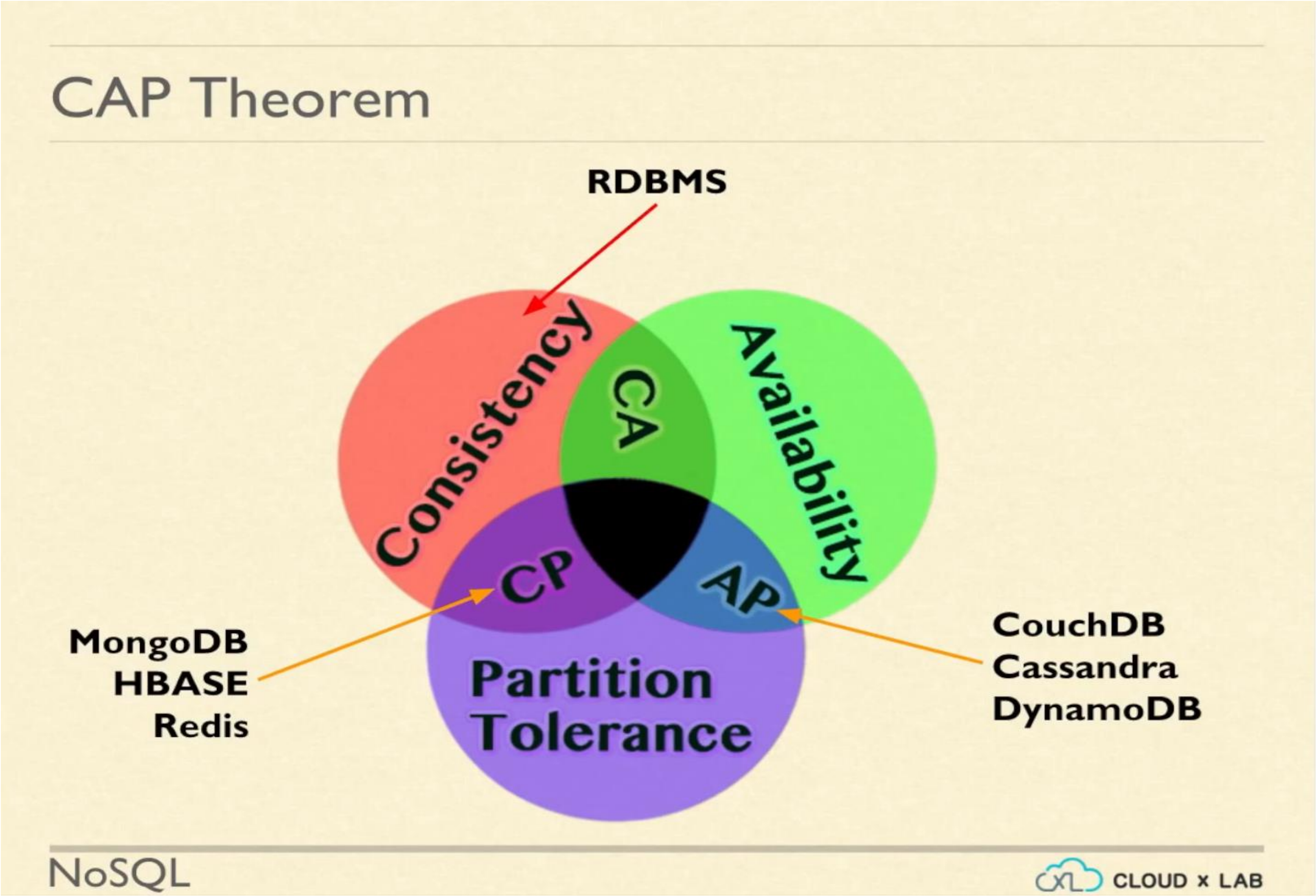
- Usually not a simple binary, different availabilty for different operations

Partition tolerance: A system’s resiliance to function even when an arbitrary number of messages between nodes have being dropped or delayed by the network

CAP Theorem

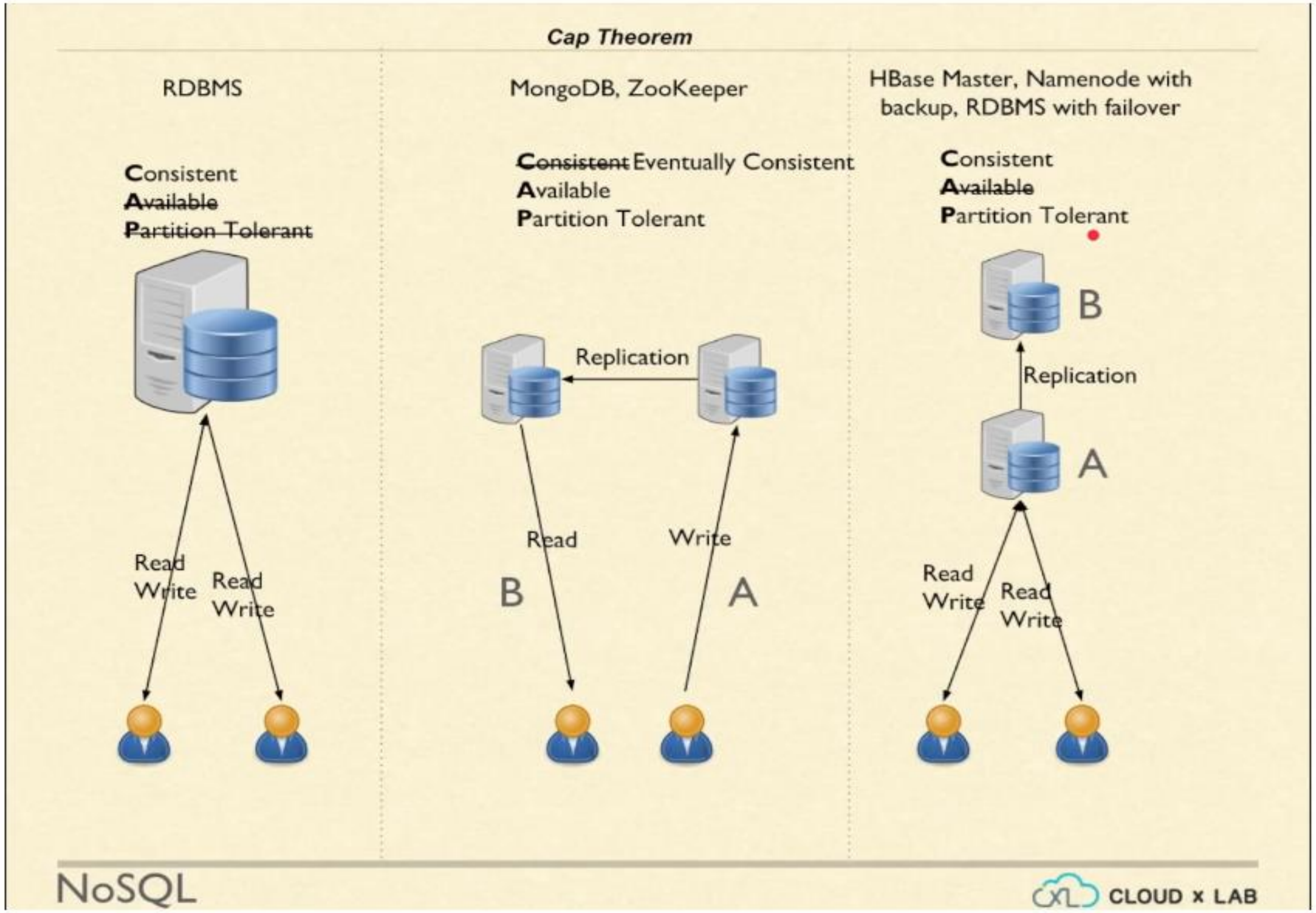
Consistency and availability are a trade-off

- A system that is highly available needs to allow operations to succeed even if some nodes are unreachable
  - System is potentially incosistent until said nodes acknowledge the operation
- A system that is strongly consistent must make sure all relevant nodes for an operation are reachable and successfully process the operation for it to succeed
  - Most extreme case is having just one node, impacts availability



From: https://cloudxlab.com/assessment/displayslide/345/nosql-cap-theorem

NoSQL - CAP Theorem



From: https://cloudxlab.com/assessment/displayslide/345/nosql-cap-theorem

ACID

Relational DBs typically provide ACID properites for transactions:

- Atomicity: all-or-nothing
- Consistency: system must be left in a consistent state after transaction
- Isolation: changes made by transaction only visible after transaction finishes
  - Hardest to implement, ranges from weak to linearizable
- Durability: once a transaction has committed, the effects remain even in case of crashes

NoSQL DBs vary widely in their support of these guarantees.

- Typically provide BASE properties

BASE

BAsically available:

- DB is always available for reads and writes, even in the event of a partial failure or network partitioning.
- Sacrifices consistency for availability (there may be inconsistencies in the data under certain conditions).

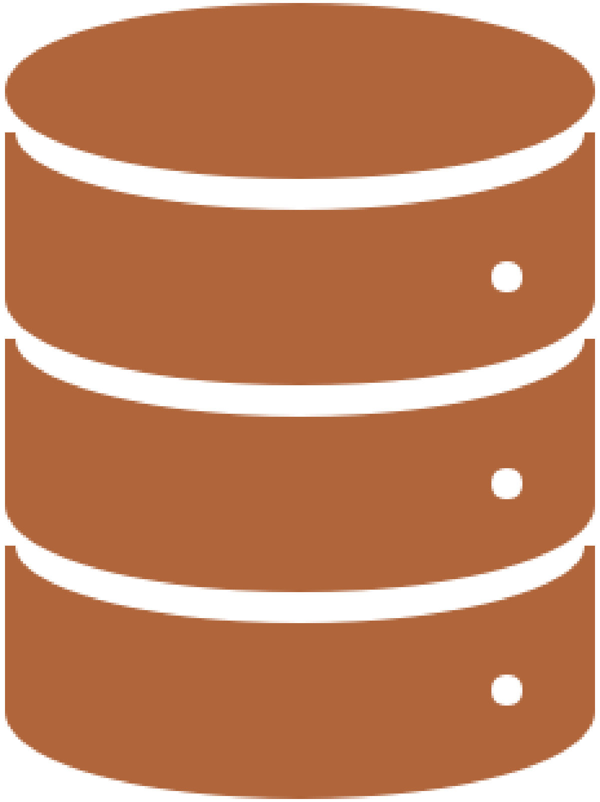
Soft State

- State of the db may change over time, even without input.
  - Byproduct of replication, caching, and eventual consistency.
- DB does not enforce strict consistency at all times.

Eventual Consistency

- DB will eventually become consistent, but not necessarily immediately.
  - DB does not guarantee that any two nodes will have the same data at the same time.
  - It does guarantee all nodes will converge to the same state eventually.

The system is basically always available but is only eventually consistent.



When to use NoSQL

We don’t require all ACID properties
- Some data is not necessarily durable (can be “forgotten” after a period of time)

Data is not structured

Data is too complex to structure in a relational schema

High volume of read/write operations per time unit

Massive amounts of data
- Horizontally scalable

High availability

BASE characteristics of NoSQL databases are designed to provide a more flexible and scalable approach to data management, with a focus on availability and performance rather than strict consistency.

When not to use NoSQL

ACID transactions required

- Applications that require immediate and strict consistency guarantees, such as financial or transactional systems.

Complex queries

- Becoming less relevant thanks to efforts like PartiQL
- Depends on DB

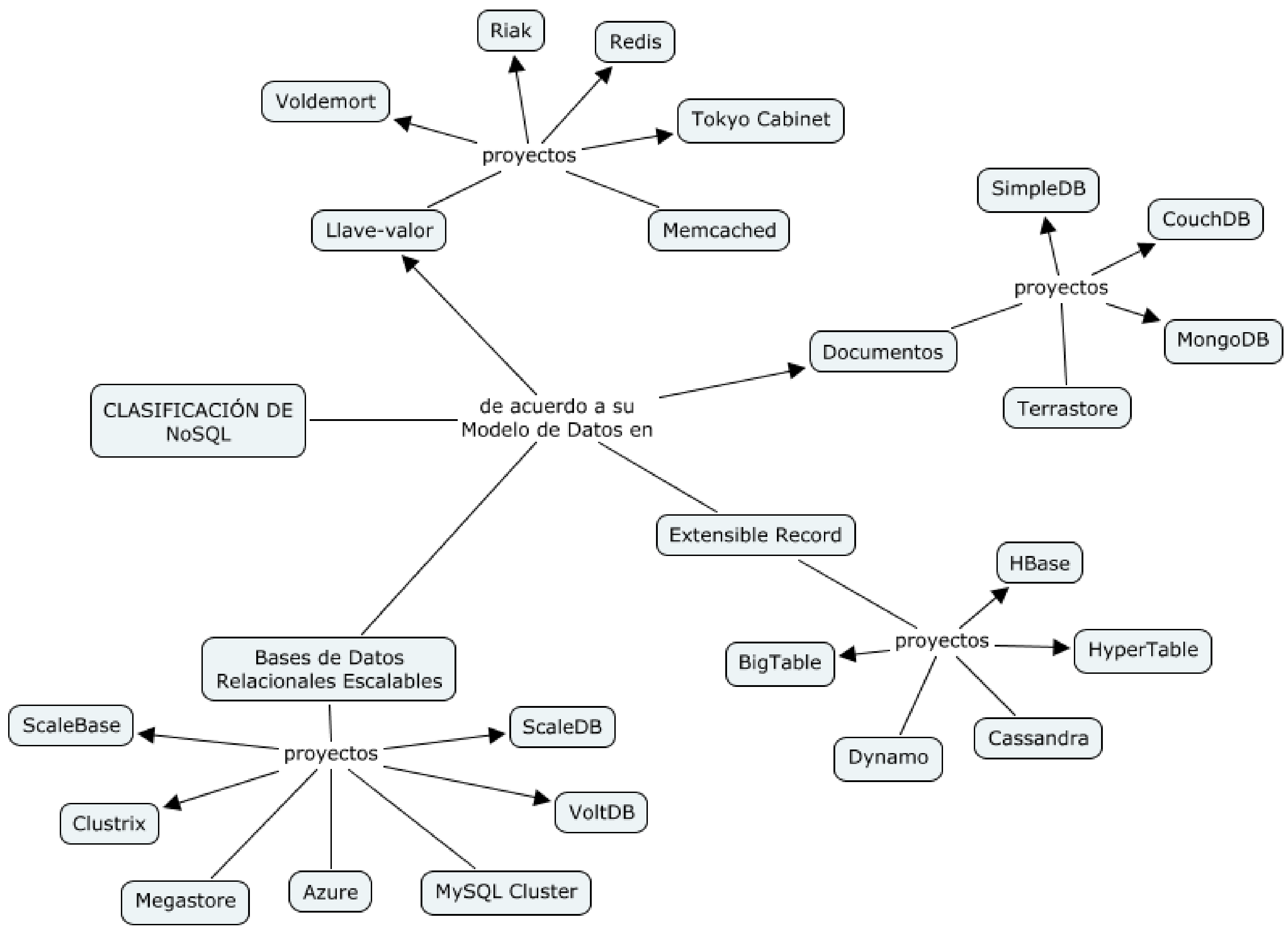
Data warehousing

- Relational DBs still have an Edge

* Not necesseraily one or the other, for complex systems might be adviceable to combine NoSQL and SQL solutions

Types of NoSQL databases

- Key-Value storage
- Document-Oriented databases
- Graphs-oriented databases
- Column-oriented databases / extensible record


CLASIFICACIÓN DE NoSQL
de acuerdo a su Modelo de Datos en
Llave-valor
proyectos
Voldemort
Riak
Redis
Tokyo Cabinet
Memcached
Documentos
proyectos
SimpleDB
CouchDB
MongoDB
Terrastore
Extensible Record
proyectos
HBase
HyperTable
BigTable
Dynamo
Cassandra
Bases de Datos Relacionales Escalables
proyectos
ScaleBase
ScaleDB
Clustrix
VoltDB
Megastore
Azure
MySQL Cluster

Key-value databases

Definition

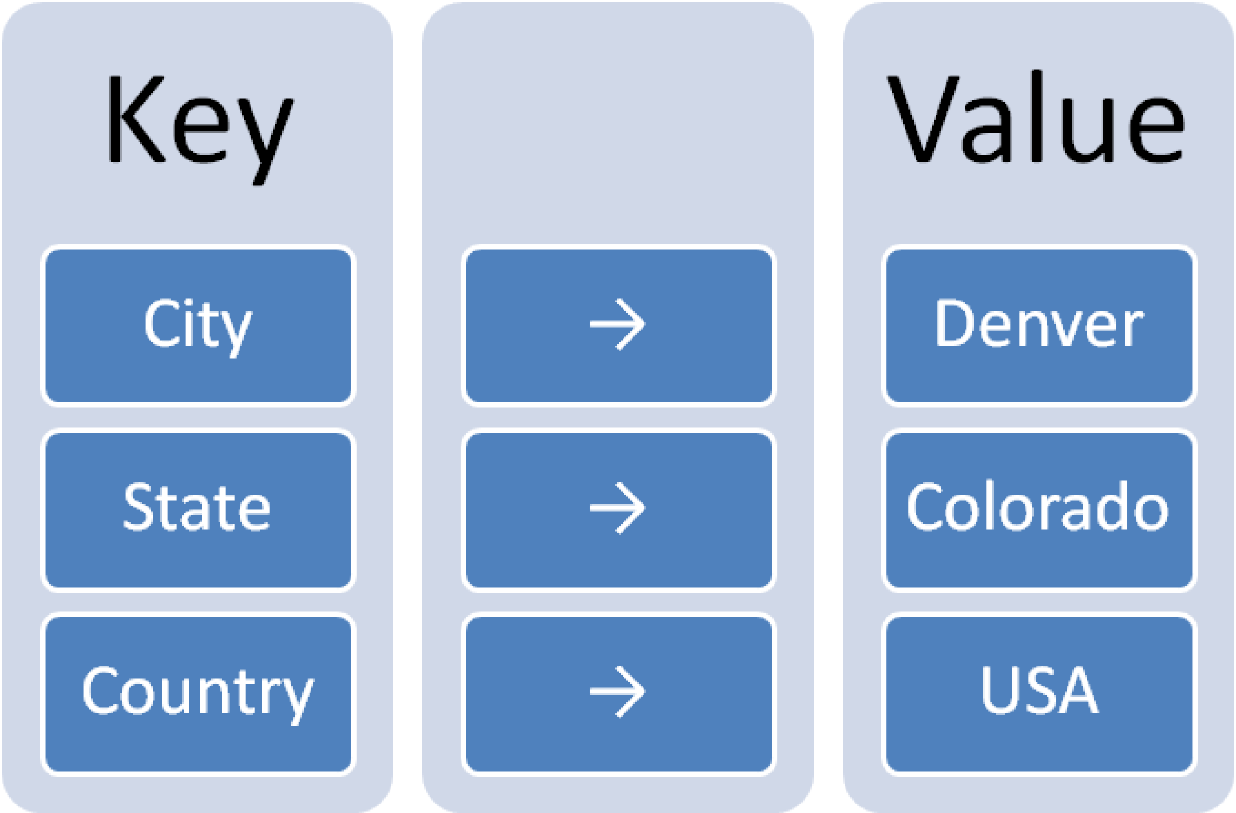
- Associative array stored on a disk; it is a single key lookup, a dictionary

Pros/Cons

- They can be read very quickly
- Not so good for reverse lookups or additional analytics.

Examples

- *Redis, Amazon DynamoDB, Flare, Voldemort*



How do they work?

Basic data structure

- <key, value> pairs

Characteristics

- Keys are unique

Basic operations

- Insert pairs
- Delete pairs
- Update values
- Find a value associated to a key

When to use them

- When working with huge amounts of data that does not need relational constraints and integrity

Document database

Definition

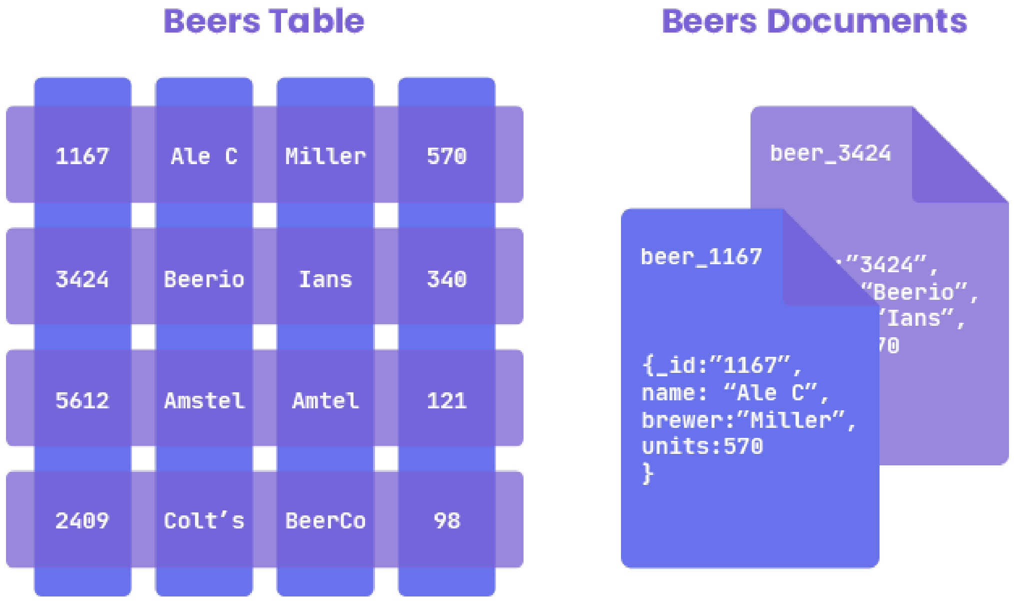
- A place to store documents.
- Documents are self-describing structures and usually similar to each other, but they don’t have to be the same.
- Popular document types: JSON-like, XML

Pros/Cons

- Documents can vary from each other and still belong to the same collection
- Support for rich querying*
- No relational capabilites

Example

- MongoDB, Amazon DocumentDB, CouchDB, BaseX



How do they work?

Basic data structure

- Documents

Characteristics

- Documents have unique IDs

Basic operations

- Insert document
- Delete document
- Update documents matching some criteria
- Find documents matching some criteria

When to use them

- When working with huge amounts of data that does not need relational constraints
- When working with flexible schemas

Column databases

Definition

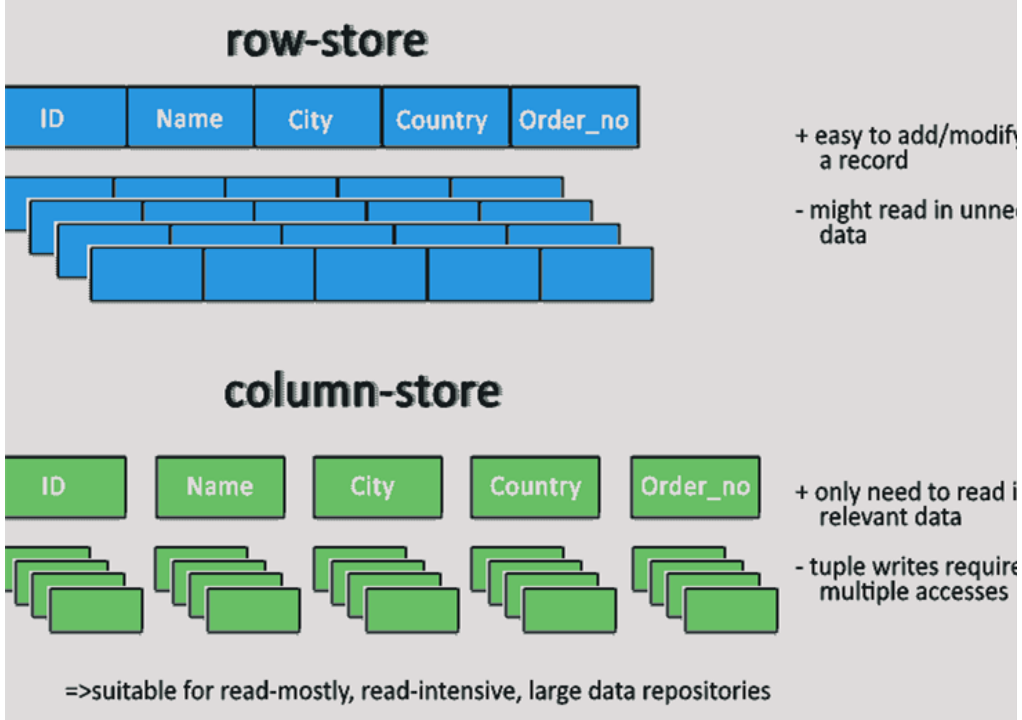
- Store data in columned families

Pros /Cons

- It is possible to get whole information in a single column in a more efficient manner
- Highly-efficient compression algorithms can be used for the homogeneous data in a column
- Columnar independent queries don't require to be locked out
- Slower when writing new "rows"

Examples

- Druid, MonetDB, MariaDB ColumnStore, Cassandra*



How do they work?

Basic data structure

- Columns of data

Characteristics

- The columns of every table are stored in their own structures
- Rows and tables are reassembled from the columns
- TAXIR was the first DB of this kind (1969)

Basic operations

- CRUD for columns

Extensible record database

Definition

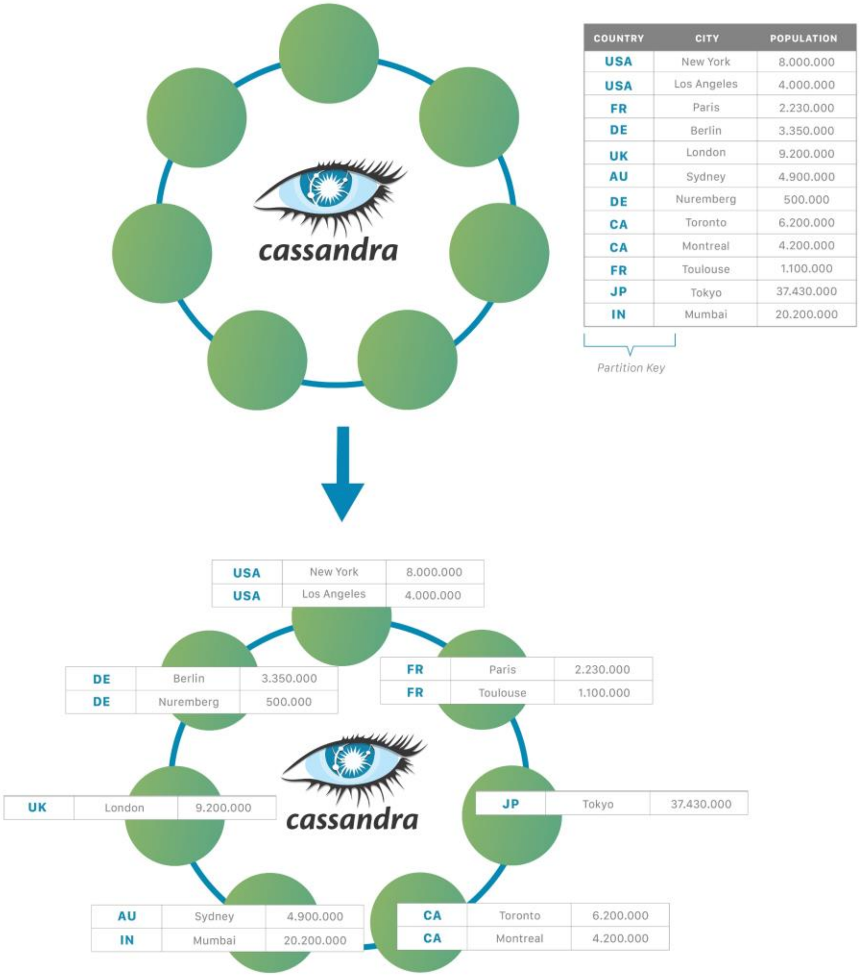
- Store data as rows and columns partitioned into multiple nodes (horizontal and vertical partition)

Pros /Cons

- Horizontal scalability
- Fault tolerant

Examples

- Cassandra, HBase



How do they work?

Basic data structure

- Columns of data

Characteristics

- Data is automatically partitioned based on partition key
- Hash function on partition key determines where data will be located

Basic operations

- CRUD for columns

Graph database

Definition

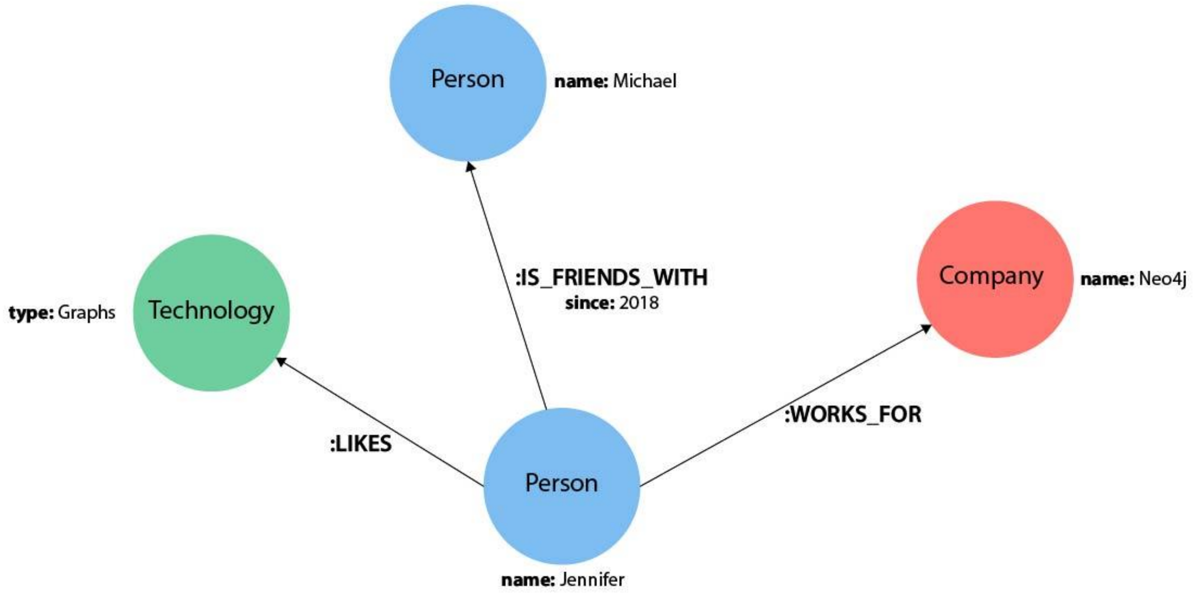
- *Uses graph* structures for queries, with *nodes*, *edges*, and *properties*, to represent and store data

Pros/Cons

- They have a number of small records with a lot of relationships between them

Examples

- AllegroGraph, Neo4j, GraphQL, Amazon Neptune



How do they work?

Basic data structure

- Nodes and relationships

Characteristics

- Relationships stored natively alongside data elements
- Optimized for traversing data quickly
- Rich querying capabilities centered on how data relates

Basic operations

- Insert node/relationship
- Delete node/relationship
- Update node/relationship matching some criteria
- Find node/relationship matching some criteria

Multi-model database

Definition

- Designed to handle multiple data models against a single integrated backend

Pros/Cons

- Databases have supported only one model, such as:
  - relational database, document-oriented database, graph database
- A database that combines many of these is multi-model.

Examples

- Allegrograph
- Couchbase
- Redis