

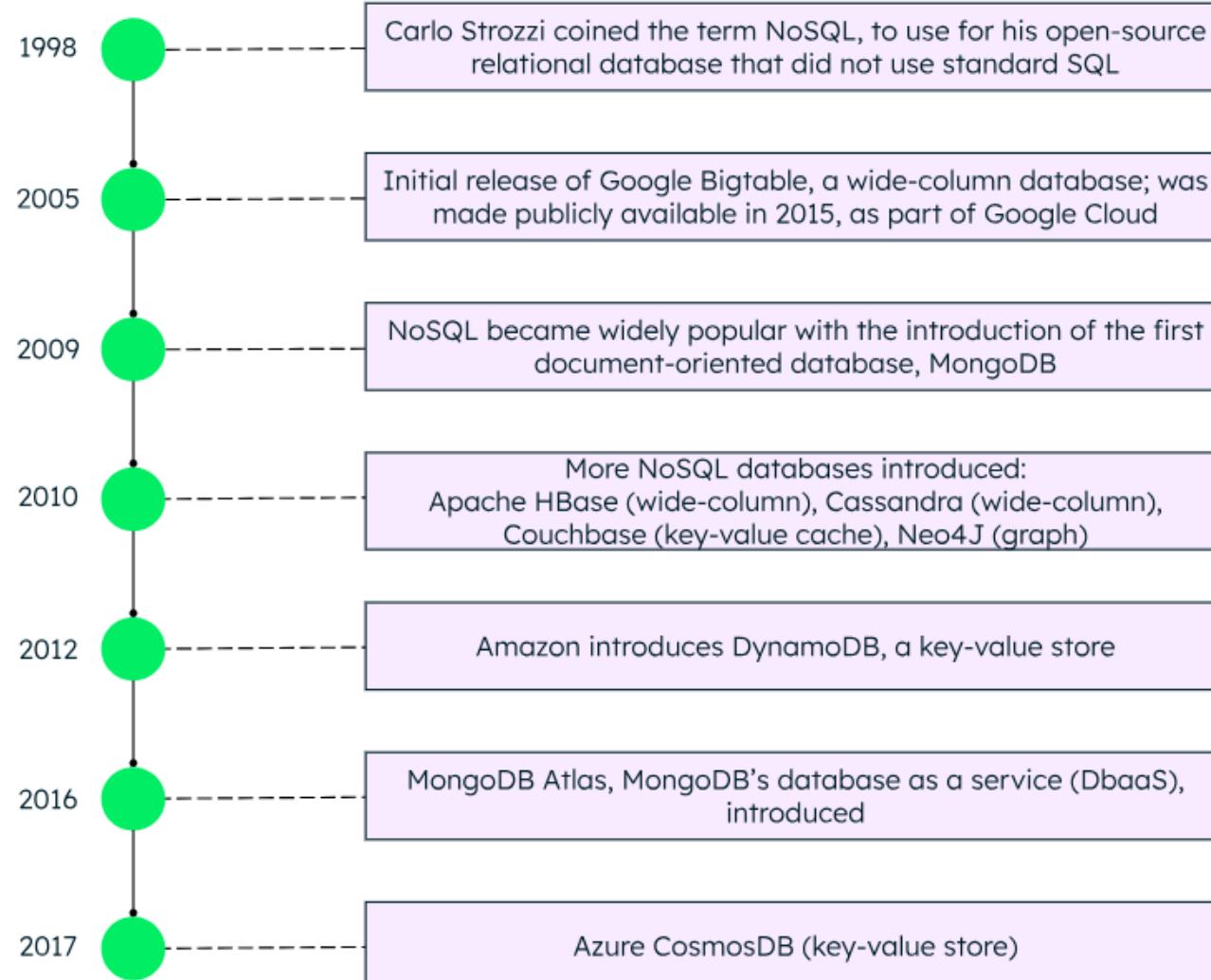
Intro to NoSQL

CS 341 Database Systems

The switch to NoSQL

- Relational databases not suited for handling **the exponential growth of real-time data or large volume of data (big data)**.
- The unstructured growth of information on the internet is a challenge for relational databases.





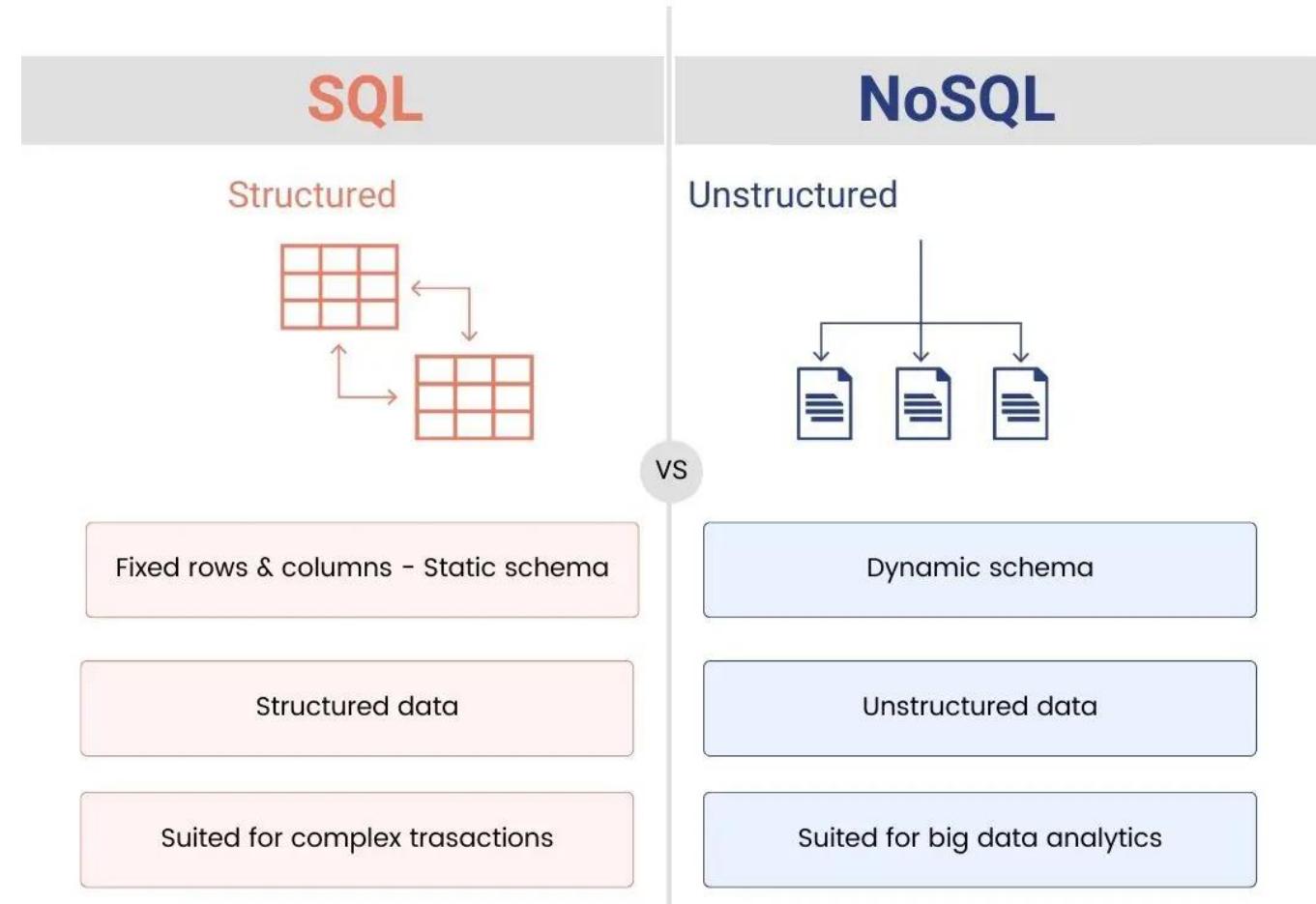
Brief History of NoSQL Databases

- Cost of storage decreased → amount of data that applications store and query increased.
- Data came in different types: structured, semi-structured and unstructured making schema definition difficult

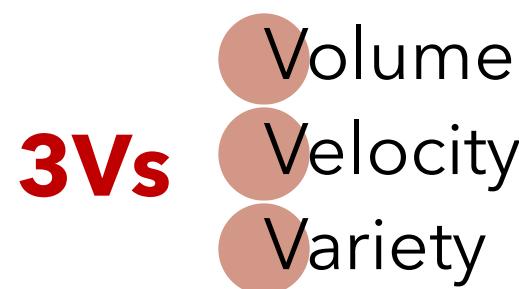
What is Wrong with RDBMS

- Rigid schema design
- Harder to scale
- Long complex join operations
- Difficult to handle data growth (unstructured data)
- A need for a designated personnel to administer DBA

Relational VS NoSQL



Uses

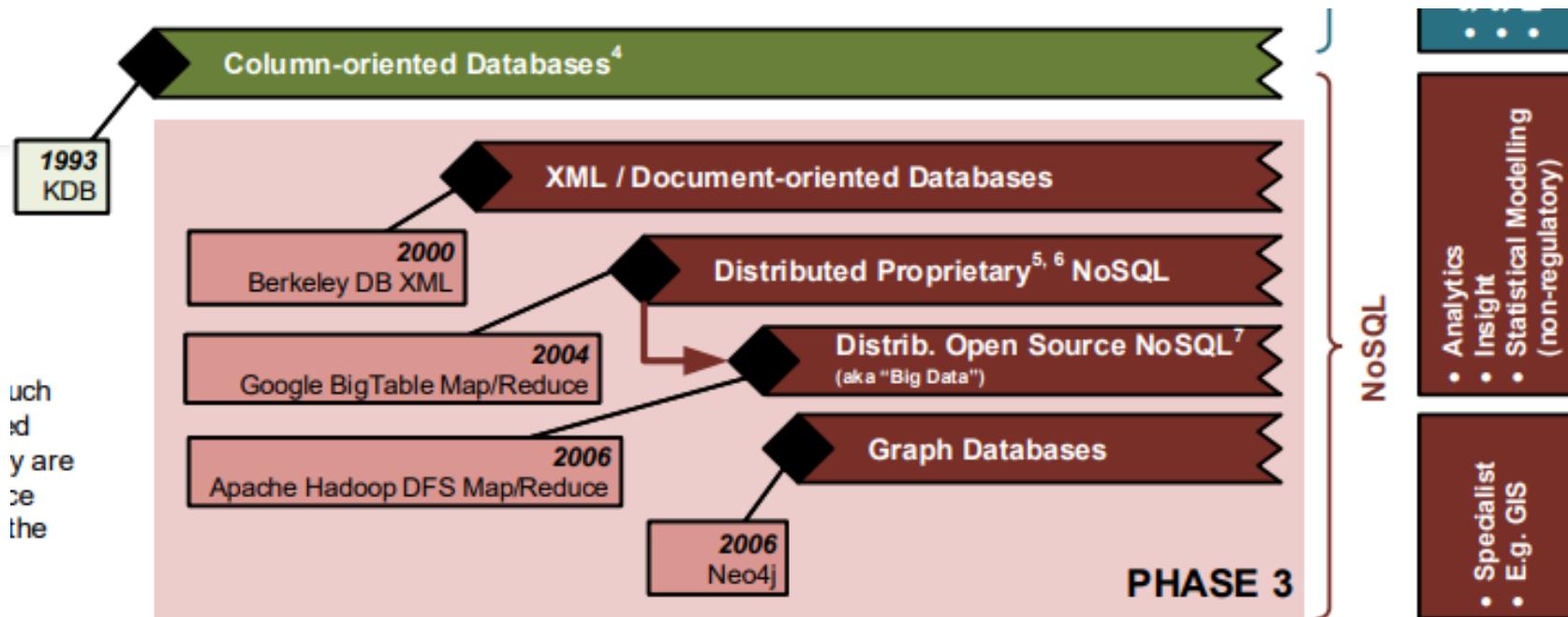


- NoSQL databases are often used in applications *where there is a **high volume of data (Big Data)** that needs to be processed and analyzed in **real-time***, such as social media analytics, data from sensors (IoT), online gaming, fraud detection, product catalog management etc.
- NoSQL may not be suitable for all applications
 - May not provide the same level of data consistency/transactional guarantees as RDBMS.

What is NoSQL?

- The principle of **NoSQL ("Not only SQL")** first appeared at the end of the 2000s and generally refers to all databases that do not store data in relational tables and whose query language is not SQL.

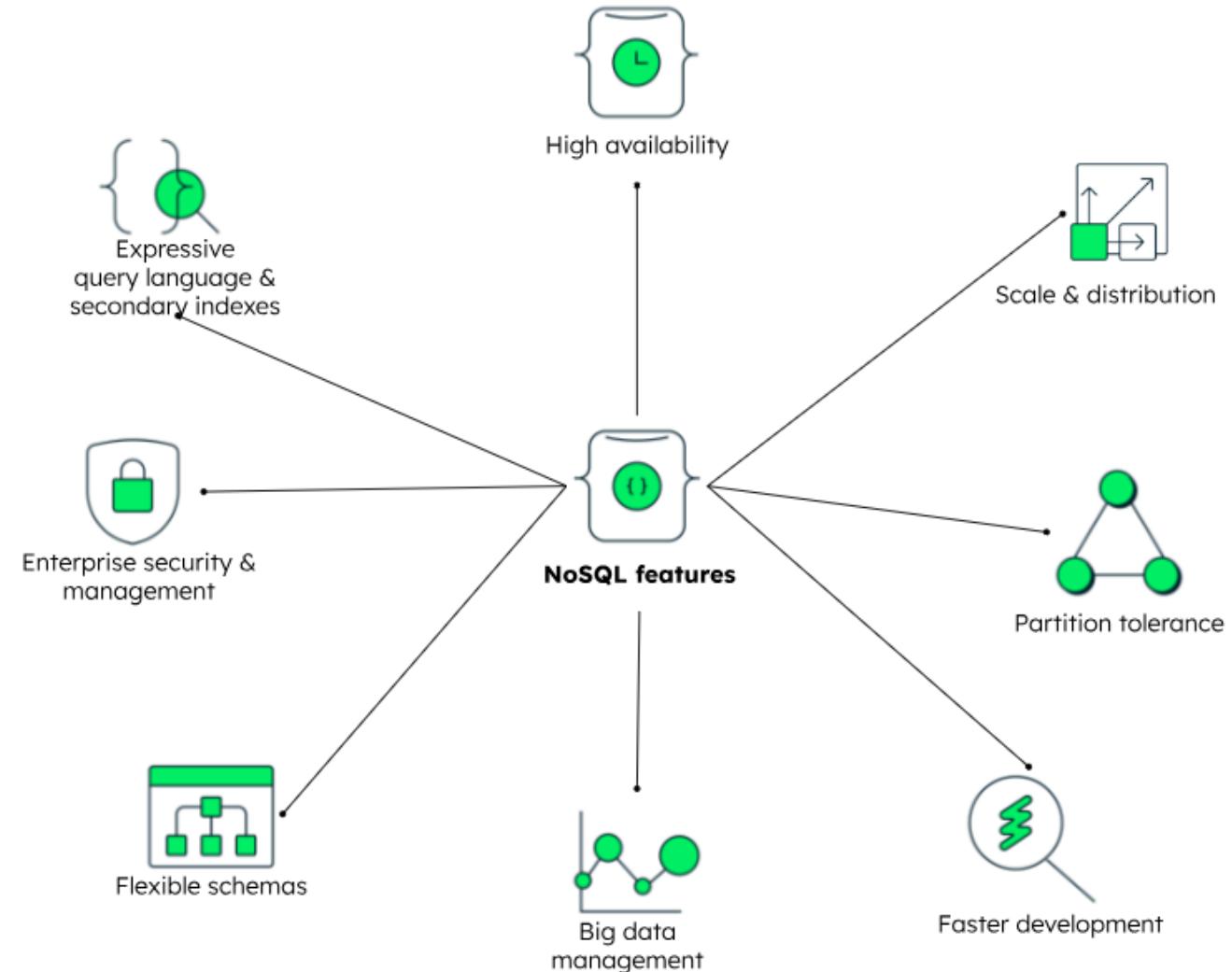
Evolution of Databases - Phase 3



Phase 3 – NoSQL technologies (such as **Big Data**) evolved from web-based businesses needing to store such vast quantities of information (multiple petabytes where $1 \text{ Pb} = 10^{15}$ bytes); so big that it had to be distributed across many machines. These were developed to sift through large of data sets searching for patterns. They are now often also applied to sensor-generated information (e.g. from jet engines). A large library of open source statistical tools is available. Data is not structured when initially stored, structure is applied when tools read the database. Here scaling is by adding more (commodity) computers to the grid. Big Data is typically used by specialist staff with a background in both technology and statistics; these are known as Data Scientists.

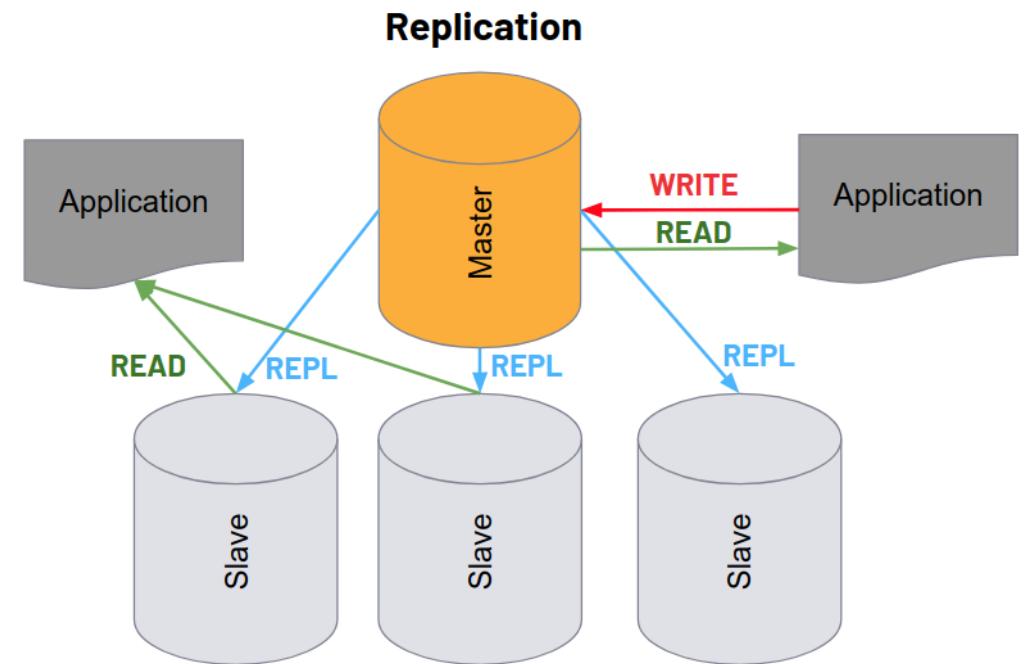
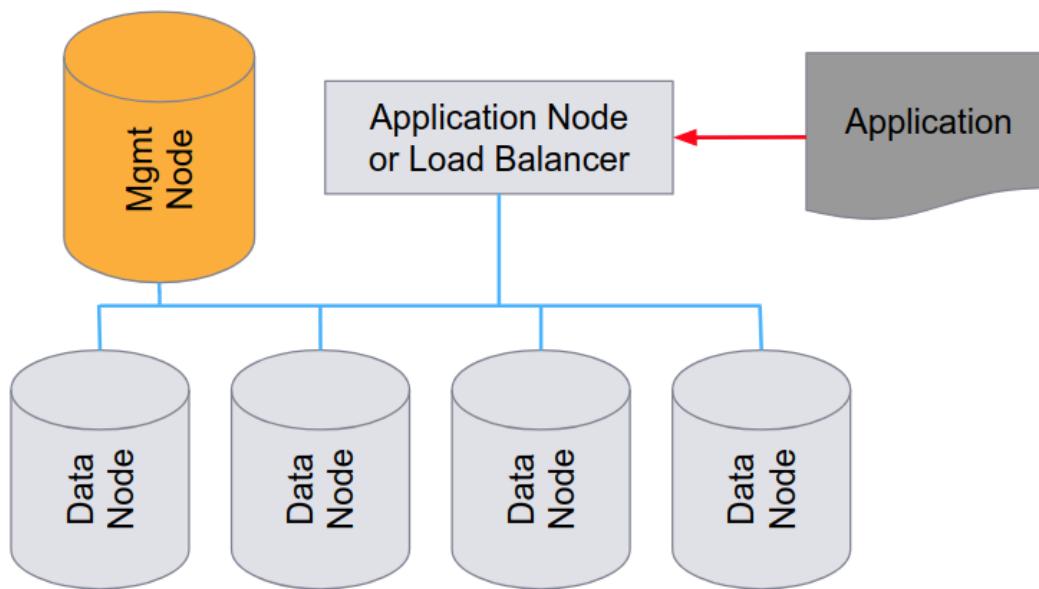
NoSQL

- An umbrella term for all databases that do not follow RDBMS principles
- NoSQL databases are:
 - **non-relational**,
 - **distributed**,
 - **horizontally scalable**
 - **open-source**



Distributed and High Availability

- NoSQL databases are often designed to be highly available and to automatically **handle node failures and data replication across multiple nodes in a database cluster.**



Horizontal Scalability

- NoSQL databases are designed to scale out by *adding more nodes to a database cluster*, making them well-suited for handling large amounts of data and high levels of traffic.
- NoSQL uses sharding for horizontal scaling:
 - Partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved → sharding.

Vertical vs Horizontal Scaling

RDBMS can scale up
NoSQL can scale up and scale out



Vertical Scaling
(Scaling up)

Vertical scaling means adding more resources to the existing machine

Horizontal Scaling
(Scaling out)

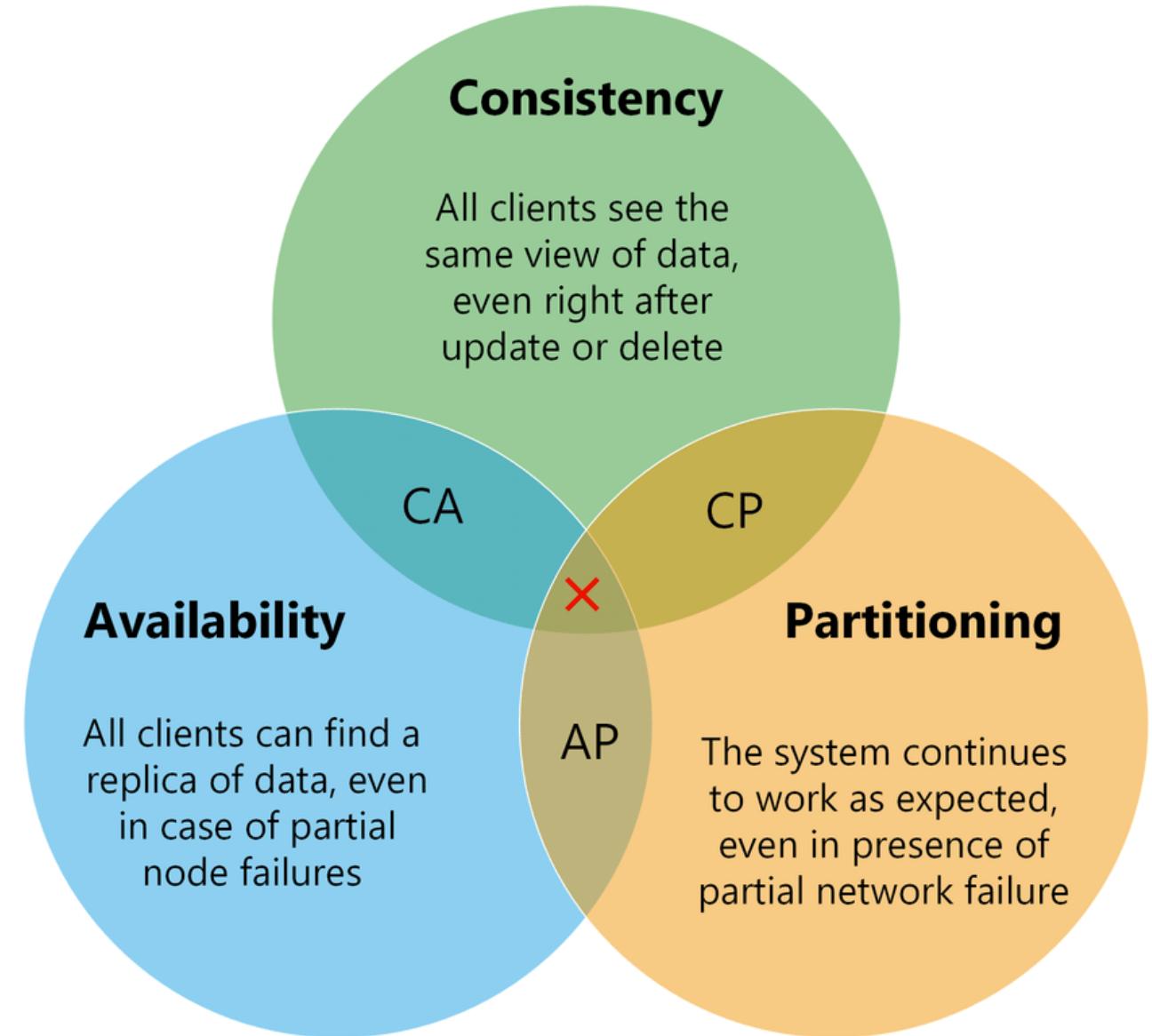
Horizontal scaling means adding more machines to handle the data.

3 Major Papers for NoSQL

- Three major papers were the “seeds” of the NoSQL movement:
 - BigTable (Google)
 - DynamoDB (Amazon)
 - CAP Theorem

CAP Theorem

- CAP theorem states that in networked shared-data systems or distributed systems, we can only achieve at most **two out of three** guarantees for a database: Consistency, Availability and Partition Tolerance.
- *Recall: A distributed system is a network of multiple nodes (physical or virtual machines) that work together to store, manage, and process data.*



Also known as Brewer's Theorem by Prof. Eric Brewer, published in 2000 at University of Berkeley.

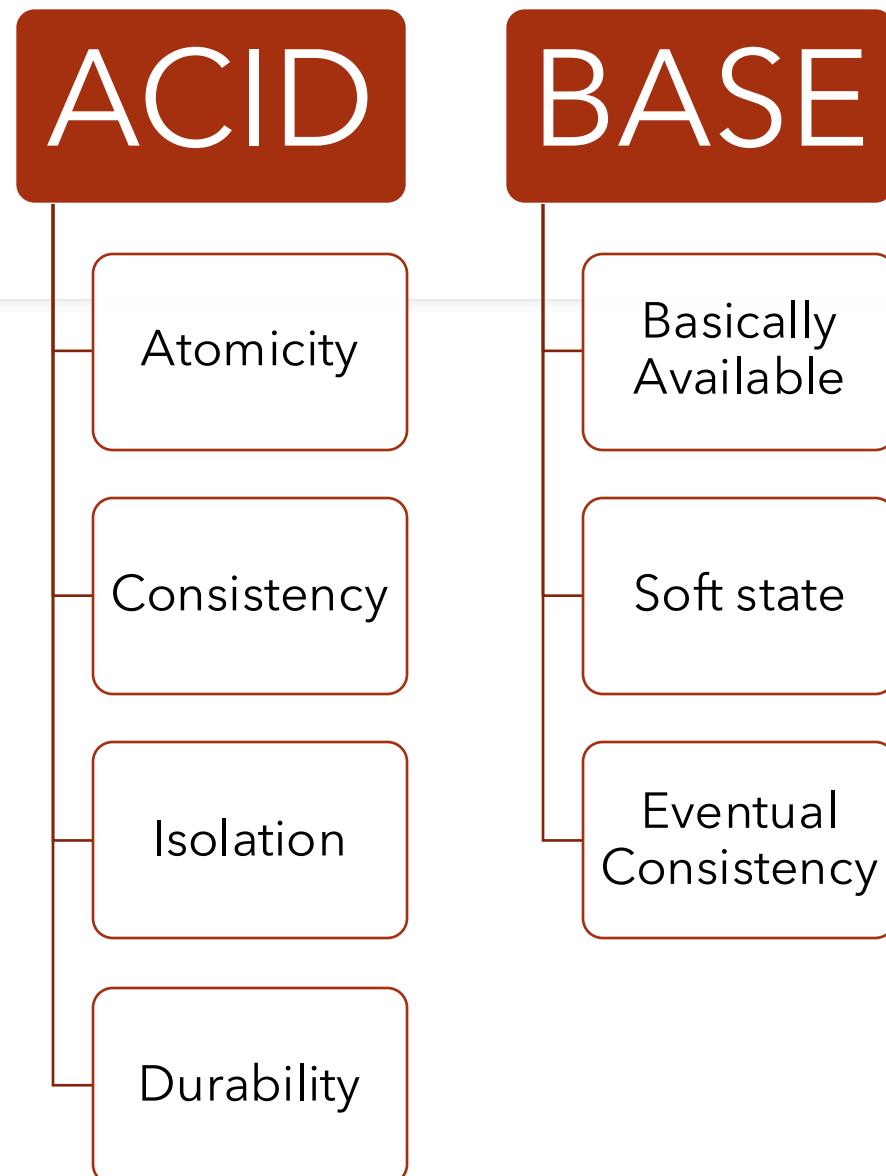
No SQL DBs are Schema Free

- NoSQL databases do not have a fixed schema and can accommodate changing data structures without the need for migrations or schema alterations.
- NoSQL databases allow developers to store and retrieve data in a flexible and dynamic manner, with support for multiple data types and changing data structures.

ACID of **RDBMS**

VS

BASE of **NoSQL**



BASE - Basically Available

- *This property refers to the fact that the database system should always be available to respond to user requests, even if it cannot guarantee immediate access to all data.*
- Nodes in a distributed environment can go down, but the whole system shouldn't be affected.

BASE - Soft state (Scalable)

- *This property refers to the fact that the state of the database can change over time, even without any explicit user intervention.*
- This can happen due background processes (periodic data replication, compaction, etc.) , updates to data (sync or exchange of info in nodes), and other factors.
- The database should be designed to handle this change gracefully, to ensure that it does not lead to data corruption or loss.

BASE - Eventual Consistency

- *This property refers to the eventual consistency of data in the database, despite changes over time.*
- In other words, the database should eventually converge to a consistent state, even if it takes some time for all updates to propagate and be reflected in the data.
- This is in contrast to the immediate consistency required by traditional ACID-compliant databases.

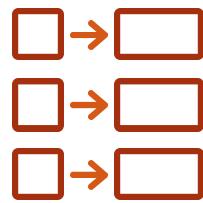
ACID VS BASE

ACID	BASE
ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee the integrity and consistency of data in a traditional database.	The BASE properties are a more relaxed version of ACID that trade off some consistency guarantees for greater scalability and availability.
ACID requires immediate consistency .	BASE only requires eventual consistency .
Better suited to traditional transactional databases.	More suitable for use in large-scale, highly-available systems.

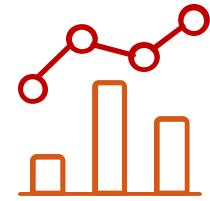
Misconception: NoSQL DB don't support ACID

- MongoDB can also be configured to provide multi-document ACID compliance
- However, NoSQL data model can eliminate need for multi-record transaction
 - In relational, multiple tables get updated together hence requiring ACID while in document store, a single document update often is sufficient, and no multi-record transaction is required.

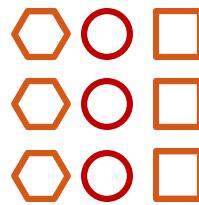
NoSQL Databases



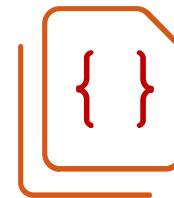
Key/Value



Graph



Wide-
Column



Document

Examples of Popular NoSQL databases

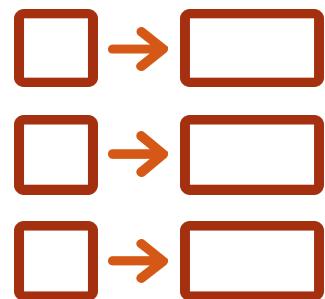
Table 14.3 NoSQL Databases

NoSQL Category	Example Databases	Developer
Key-value database	Dynamo Riak Redis Voldemort	Amazon Basho Redis Labs LinkedIn
Document databases	MongoDB CouchDB OrientDB RavenDB	MongoDB, Inc. Apache OrientDB Ltd. Hibernating Rhinos
Column-oriented databases	HBase Cassandra Hypertable	Apache Apache (originally Facebook) Hypertable, Inc.
Graph databases	Neo4j ArangoDB GraphBase	Neo4j ArangoDB, LLC FactNexus



NoSQL Databases

Key Value Database



Key/Value Database

- **Key** points to information (**value**)
- Structure allows easy partition based on key values
- E.g. Amazon DynamoDB and Redis

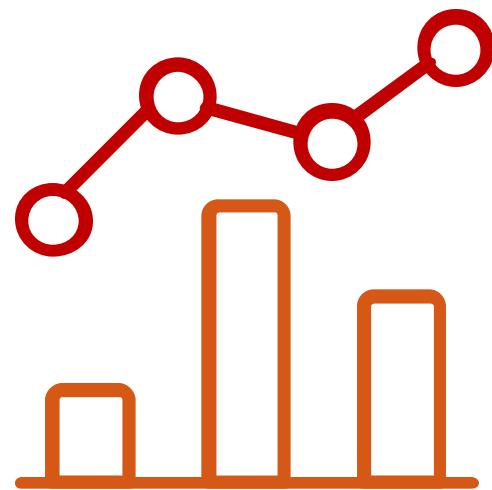
Key Value Database

Figure 14.7 Key-Value Database Storage

Bucket = Customer

Key	Value
10010	"LName Ramas FName Alfred Initial A Areacode 615 Phone 844-2573 Balance 0"
10011	"LName Dunne FName Leona Initial K Areacode 713 Phone 894-1238 Balance 0"
10014	"LName Orlando FName Myron Areacode 615 Phone 222-1672 Balance 0"

Graph Database



Graph Database

- Niche problems or highly connected data where relationships or patterns may not be obvious initially.
- Collection of nodes and edges (represents relationships)
- Related data retrieved in a single operation as opposed to joins
- E.g. Neo4J, Amazon Neptune, Mongo DB (also capable)

Graph Database

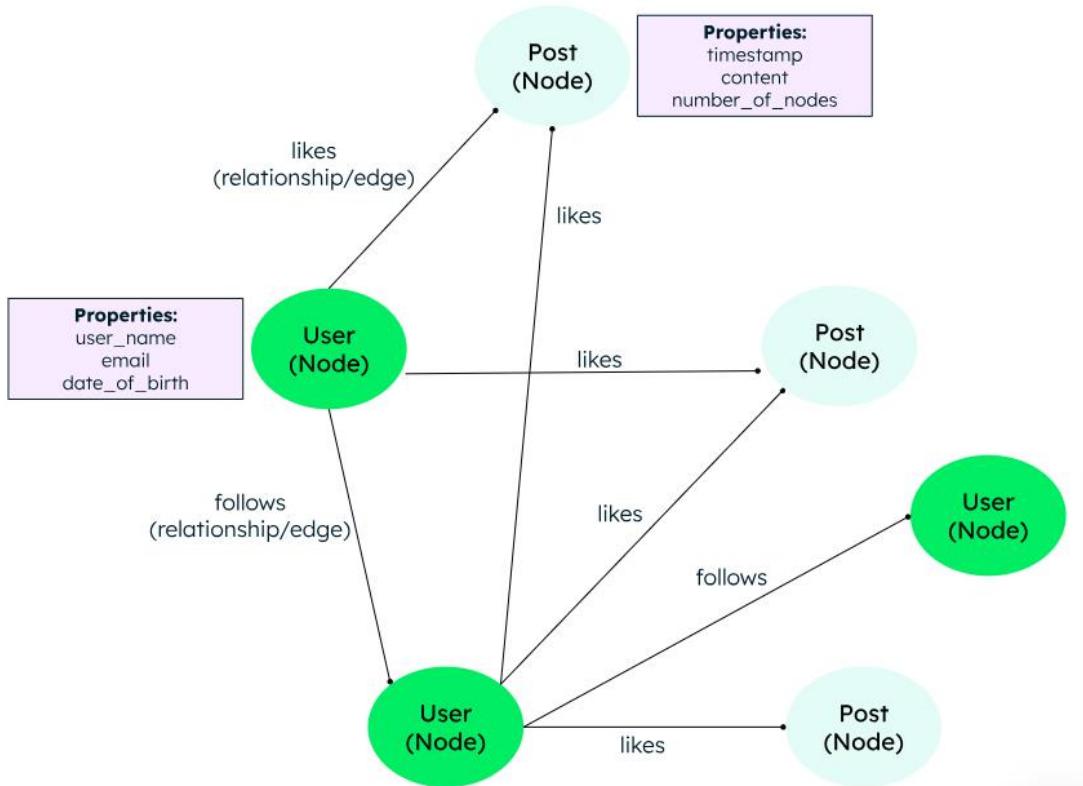
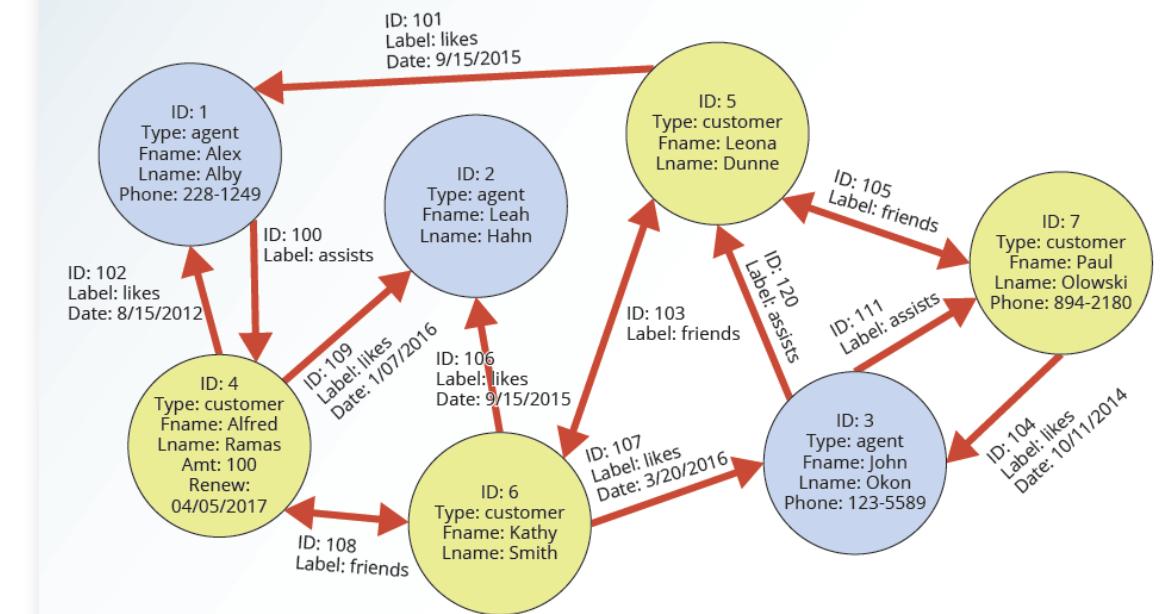
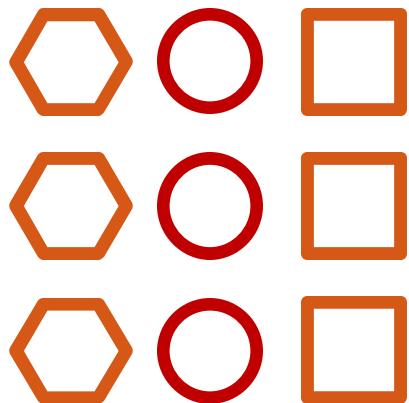


Figure 14.11 Graph Database Representation



Wide - Column Stores



Wide Column stores

- Data is stored in tables: rows and dynamic columns
- Flexible schema - different rows can have different sets of columns

Column Centric Storage

Row-centric storage				
Block 1		Block 4		
10010,Ramas,Alfred,Nashville,TN 10011,Dunne,Leona,Miami,FL		10016,Brown,James,NULL,NULL 10017,Williams,George,Mobile,AL		
Block 2		Block 5		
10012,Smith,Kathy,Boston,MA 10013,Olowski,Paul,Nashville,TN		10018,Farriss,Anne,Opp,AL 10019,Smith,Olette,Nashville,TN		
Block 3				
10014,Orlando,Myron,NULL,NULL 10015,O'Brian,Amy,Miami,FL				
Column-centric storage				
Block 1		Block 4		
10010,10011,10012,10013,10014 10015,10016,10017,10018,10019		Nashville,Miami,Boston,Nashville,NULL Miami,NULL,Mobile,Opp,Nashville		
Block 2		Block 5		
Ramas,Dunne,Smith,Olowski,Orlando O'Brian,Brown,Williams,Farriss,Smith		TN,FL,MA,TN,NULL, FL,NULL,AL,AL,TN		
Block 3				
Alfred,Leona,Kathy,Paul,Myron Amy,James,George,Anne,Olette				
CUSTOMER relational table				
Cus_Code	Cus_LName	Cus_FName	Cus_City	Cus_State
10010	Ramas	Alfred	Nashville	TN
10011	Dunne	Leona	Miami	FL
10012	Smith	Kathy	Boston	MA
10013	Olowski	Paul	Nashville	TN
10014	Orlando	Myron		
10015	O'Brian	Amy	Miami	FL
10016	Brown	James		
10017	Williams	George	Mobile	AL
10018	Farriss	Anne	Opp	AL
10019	Smith	Olette	Nashville	TN

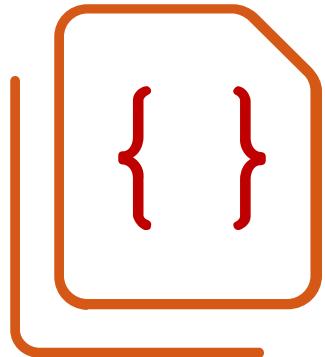
Column Family Database

- Column is a key-value pair that is similar to a cell of data in a relational database. The key is the name of the column, and the value component is the data that is stored in that column.
- Wide rows/columns → efficient retrieval of sparse and wide data
- E.g. Apache Cassandra and HBase

Figure 14.10 Column Family Database

Column Family Name	CUSTOMERS	
Key	Rowkey 1	
Columns	City	Nashville
	Fname	Alfred
	Lname	Ramas
	State	TN
Key	Rowkey 2	
Columns	Balance	345.86
	Fname	Kathy
	Lname	Smith
Key	Rowkey 3	
Columns	Company	Local Markets, Inc.
	Lname	Dunne

Document Database



Document Database

- Polymorphic data structures
- Obvious relationships using embedded arrays and documents
- Easy and natural representation
- No complex mapping between application data and database
- Flexible for semi and unstructured data
- E.g. MongoDB and Couchbase

Document Databases

Figure 14.8 Document Database Tagged Format

Collection = Customer

Key	Document
10010	{LName: "Ramas", FName: "Alfred", Initial: "A", Areacode: "615", Phone: "844-2573", Balance: "0"}
10011	{LName: "Dunne", FName: "Leona", Initial: "K", Areacode: "713", Phone: "894-1238", Balance: "0"}
10014	{LName: "Orlando", FName: "Myron", Areacode: "615", Phone: "222-1672", Balance: "0"}

Multi-model databases

- Support more than one type of NoSQL data model
- Developers can choose based on application requirements
- Unified database engine to handle multiple data models within a database instance
- E.g. CosmosDB and ArangoDB

Comparison



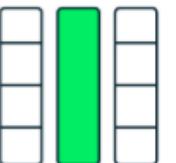
Key-value database



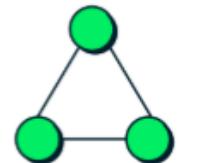
Document database



Wide-column database



Graph database



Basic support for data visualization	Basic support for data visualization	Basic support for data visualization	Basic support for data visualization
Basic authentication, limited access control	Role-based access control, encryption at-rest, encryption in transit, and in use	Role-based access control (RBAC), encryption at rest	Role-based access control (RBAC), encryption at rest, in transit
Limited support for online data archival	Automatic online data archival	Limited support for online data archival	Limited support for online data archival
Basic key-based data retrieval	Full-text search, vector search	Limited search capabilities	Limited search capabilities
Basic CRUD operations, limited data manipulation capabilities	Advanced query and data manipulation capabilities	Data manipulation and CRUD capabilities	Supports advanced graph traversal and manipulation operations
Stores any type of data	Stores JSON/BSON data	Columns and rows matrix	Nodes, edges, and relationships
Limited support for complex analytics	Suited for time-series, IoT analytics, real-time analytics	Suited for time-series data, IoT analytics	Well-suited for graph analytics
Horizontal scalability and eventual consistency	Horizontal scalability and eventual consistency	Horizontal scalability and eventual consistency	Horizontal scalability and eventual consistency
Simple indexes	Indexes on fields	Secondary indexes	Indexes on nodes/edges
Limited by key	Rich querying capability	Limited by columns	Specialized graph query
Schemaless	Flexible schema	Flexible schema	Flexible schema
Simple key-value pairs	JSON/BSON documents	Column and row families	Nodes, edges, and relationships

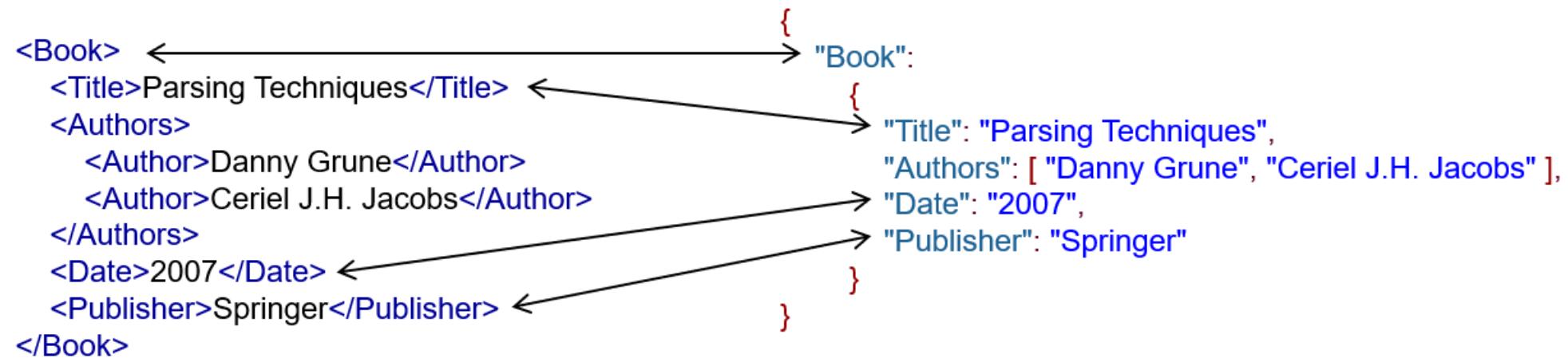
Document Databases

Types of Documents

- **XML**
 - Inspired by HTML
- **JSON - JavaScript Object Notation or BSON - Binary JSON**
 - Forms the basis of MongoDB

For a document database, being *schema-less* means that although all documents have fields, not all documents are required to have the same fields, so each document can have its own structure.

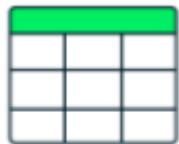
XML and JSON comparison



Tables VS Documents

- Document databases treat data as *self-contained documents*, not fragments of data.
- Example: An order document contains all related data (of entities involved e.g customer, order, products) in one document as opposed to tables in relational database.

RDBMS vs NoSQL (Document)



Relational Database

User table

ID	first_name	last_name	cell	city
1	Leslie	Yepp	8125552344	Pawnee

Hobbies table

ID	user_id	hobby
10	1	scrapbooking
11	1	eating waffles
12	1	working



MongoDB

```
{  
  "_id": 1,  
  "first_name": "Leslie",  
  "last_name": "Yepp",  
  "cell": "8125552344",  
  "city": "Pawnee",  
  "hobbies": ["scrapbooking", "eating  
waffles", "working"]  
}
```

- No need for joins
- No need for data normalization

What is **JSON**?

What is JSON?

JSON

- Stands for JavaScript Object Notation.

Purpose:

- A lightweight format for storing and transporting data.

Usage:

- Commonly used to send data from a server to a web page.

Language-Independent:

- Not tied to any specific programming language.

Self-Describing:

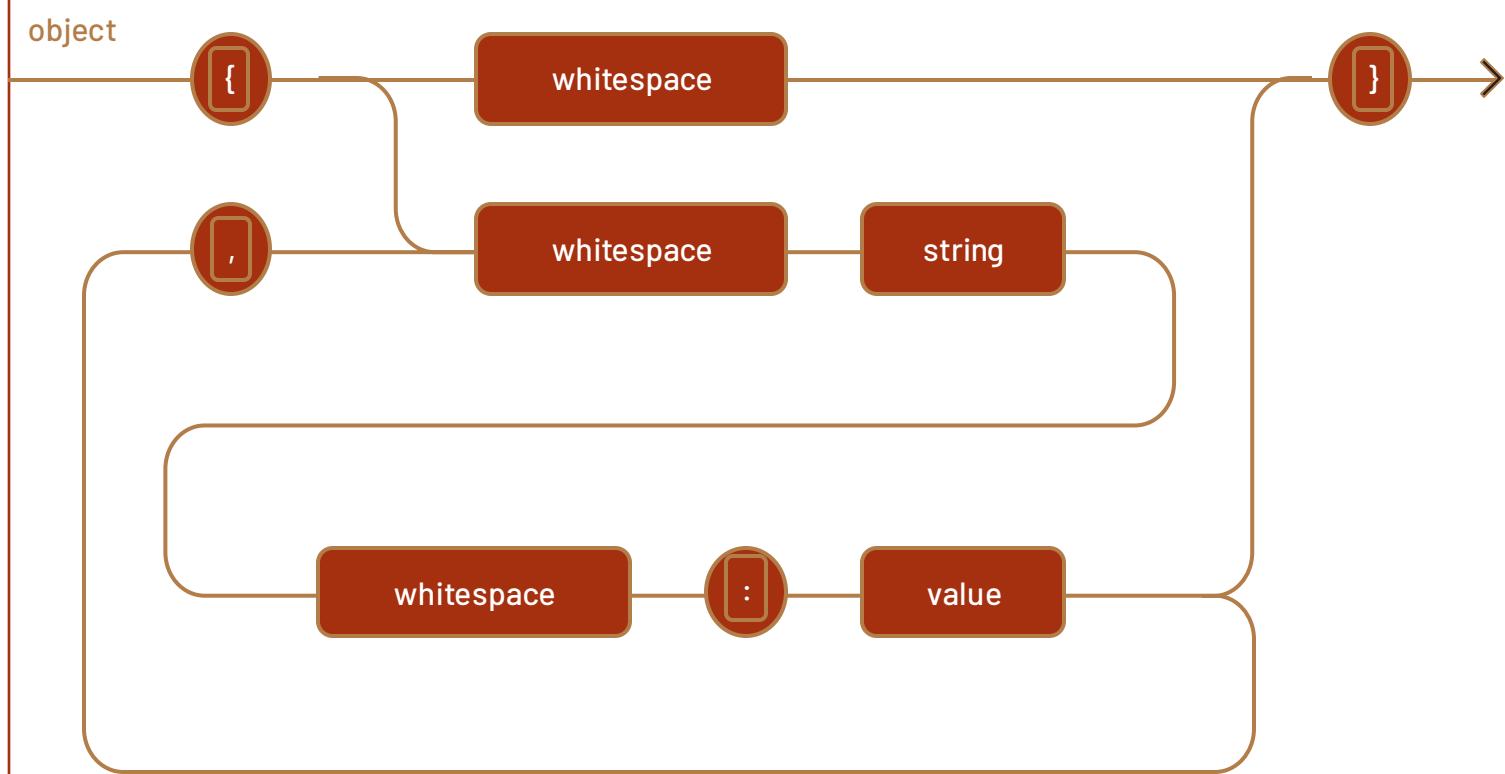
- Easy to read and understand.

JSON Object

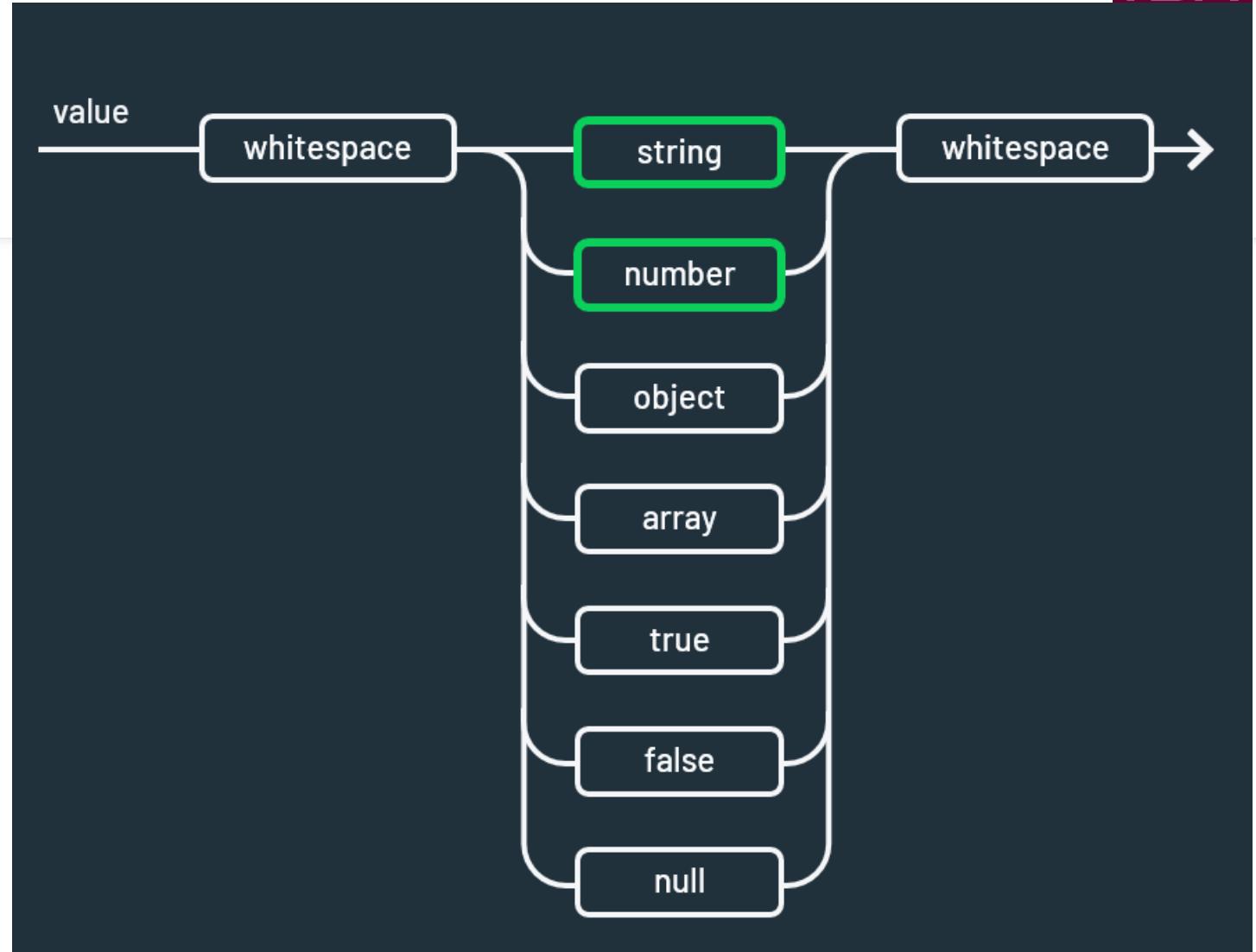
SYNTAX RULES:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

```
{  
  "_id": ObjectId(  
    "5f4f7fef2d4b45b7f11b6d7a" ),  
  "user_id": "Alpha",  
  "age": 21,  
  "Status": "Active"  
}
```



JSON Values



JSON Object

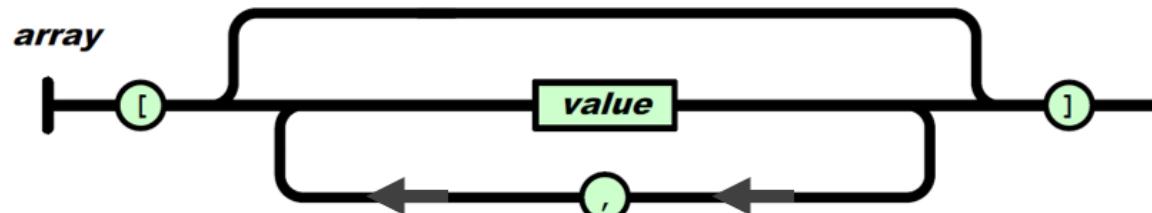
```
{  
  "Name": "Alpha",  
  "Age": 20,  
  "Enrolled": true,  
  "Contact": {  
    "Phone": 123456789,  
    "Address": null  
  },  
  "Courses": [ "DB", "OS", "SE"]  
}
```

Field-Value pairs

Contact is a JSON Object

JSON Array

Array of Objects



```
{  
  "name": "John Doe",  
  "age": 30,  
  "married": true,  
  "siblings": [  
    {"name": "John", "age": 25},  
    true,  
    "Hello World"  
  ]  
}
```

The array contains 3 items.
The first item is an object,
the second item is a boolean,
and the third item is a string.

JSON Example

- {
 "employees": [
 { "firstName": "John", "lastName": "Doe"},
 { "firstName": "Anna", "lastName": "Smith"},
 { "firstName": "Peter", "lastName": "Jones"}
]
}

Example of JSON - TESLA

SCALAR DOCUMENT

```
{  
    "manufacturer" : "Tesla Motors",  
    "class" : "full-size",  
    "body style" : "5-door liftback"  
}
```

ARRAY DOCUMENT

```
{  
    "production" : [2012, 2013, 2014],  
    "model years" : [2013],  
    "layout" : ["Rear-motor", "rear-wheel drive"]  
}
```

Example - contd.

EMBEDDED DOCUMENT

```
{  
    "designer" : {  
        "firstname" : "Franz",  
        "surname" : "von Holzhausen"  
    }  
}
```

Example - contd.

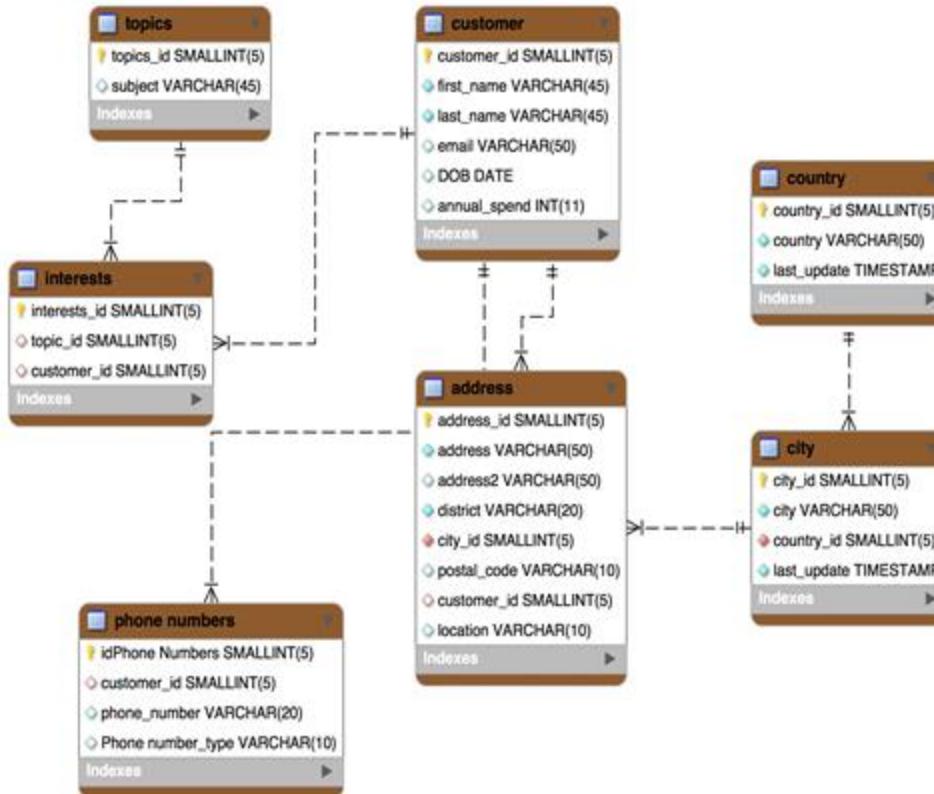
ARRAY WITH EMBEDDED DOCUMENT

```
{  
    "assembly": [  
        {  
            "country": "United States",  
            "city": "Fremont",  
            "state": "California"  
        },  
        {  
            "country": "The Netherlands",  
            "city": "Tilburg"  
        }  
    ]  
}
```

MongoDB

- MongoDB is a document database
- Installed locally or hosted on cloud
- Local installation:
 - host your own MongoDB server on your hardware.
 - requires you to manage your server, upgrades, and any other maintenance.
- For the course, we will use MongoDB Atlas, a cloud-based platform, which is simpler to setup and use.

What is MongoDB (the database) ?



Tabular (Relational) Data Model

Related data split across multiple records and tables

```
{
  "_id" : ObjectId("5ad88534e3632e1a35a58d00"),
  "name" : {
    "first" : "John", "last" : "Doe" },
  "address" : [
    {
      "location" : "work",
      "address" : { "street" : "16 Hatfields",
      "city" : "London", "postal_code" : "SE1 8DJ" },
      "geo" : { "type" : "Point", "coord" : [
        51.5065752, -0.109081 ] },
      ...
    },
    ...
  ],
  "dob" : ISODate("1977-04-01T05:00:00Z"),
  "retirement_fund" : NumberDecimal("1292815.75")
}
```

Document Data Model

Related data contained in a single, rich document

MongoDB Document

- A way to organize and store data as a set of **field-value pairs**.
- Similar to Dictionaries in Python, Maps in Java, JSON Object in JavaScript
- BSON, or **Binary JSON**, is the data format that MongoDB uses to organize and store data.
- This data format includes all JSON data structure types and adds support for types including dates, different size integers, Object_Ids, and binary data.

Document model constructs

- Fields (Attributes)
- Sub-documents (Objects)
- Arrays

Syntax Rules:

- Curly brackets **{}** : marks the start and end of the document
- Field value pairs are separated by **:**
- Field names are in quotation marks. **" "**
- Field value pairs are separated by commas **(,)**

Fields / attributes

Cars				
_id	owner	make	model	year
007	Daniel	Ferrari	GTS	1982
008	Daniel	Fiat	500	2013

Tabular (Relational) Data Model

Header with column names

Row with values

```
{  
  "_id": 007,  
  "owner": "Daniel",  
  {  
    "_id": 008,  
    "owner": "Daniel",  
    "make": "Fiat",  
    "model": "500",  
    "submodel": "S",  
    "year": 2013  
  }  
}
```

Document Data Model

Each document explicitly list the names of the fields and the values.

Object/sub-document: a one-to-one relationship

Cars		
_id	owner	make
007	Daniel	Ferrari
Q08	Daniel	Fiat

Engines			
_id	car_id	power	consumption
234808	007	660	10
Q08	008	120	45

OR

Cars				
_id	owner	make	power	consumption
007	Daniel	Ferrari	660	10
Q08	Daniel	Fiat	120	45

```
{
  "_id": 007,
  "owner": "Daniel",
  "make": "Ferrari",
  "engine": {
    "power": 660hp,
    "consumption": 10mpg
  }
}
```

Tabular (Relational) Data Model

A car has one Engine. A one-to-one relationship in a single table or across 2 tables

Document Data Model

The engine information is in its own structure in the parent entity

Array: a one-to-many

Cars		
_id	owner	make
007	Daniel	Ferrari
008	Daniel	Fiat

Wheels	
_id	car_id
234819	007
281928	007
392838	007
928038	007
950555	008

```
{  
    "_id": 007,  
    "owner": "Daniel",  
    "make": "Ferrari",  
    wheels: [  
        { "partNo": 234819 },  
        { "partNo": 281928 },  
        { "partNo": 392838 },  
        { "partNo": 928038 }  
    ],  
    ...  
}
```

Tabular (Relational) Data Model

One-to-Many relationship
from a car to the its wheels

Document Data Model

One-to-Many wheels
expressed as an array

MongoDB Collections

- An organized store of documents in MongoDB, usually with common fields between documents
- MongoDB collections do not by default enforce a single schema on a collection i.e *whilst documents can have common fields, they are not required to have the same fields.*

```
{  
  "_id": 11,  
  "user_id": "Erin",  
  "age": 29,  
  "Status": "A"  
}
```

```
{  
  "_id": 12,  
  "user_id": "Daniel",  
  "age": 25,  
  "Status": "A",  
  "Country": "USA"  
}
```