SNS COLLEGE OF TECHNOLOGY COIMBATORE

AN AUTONOMOUS INSTITUTION

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade

Approved by AICTE New Delhi & affiliated to the Anna University, Chennai

DEPARTMENT OF MCA

Course Name : 23CAT603 - DATA BASE MANAGEMENT SYSTEM

Class : I Year / I Semester

Unit V - COLUMN ORIENTED DATABASE

Topic I – Definition of NOSQL

What is NoSQL?

NoSQL Database is a non-relational Data Management System, that does not require a fixed schema. It avoids joins, and is easy to scale. The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps.

For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.

NoSQL database stands for "Not Only SQL" or "Not SQL." Though a better term would be "NoREL", NoSQL caught on.

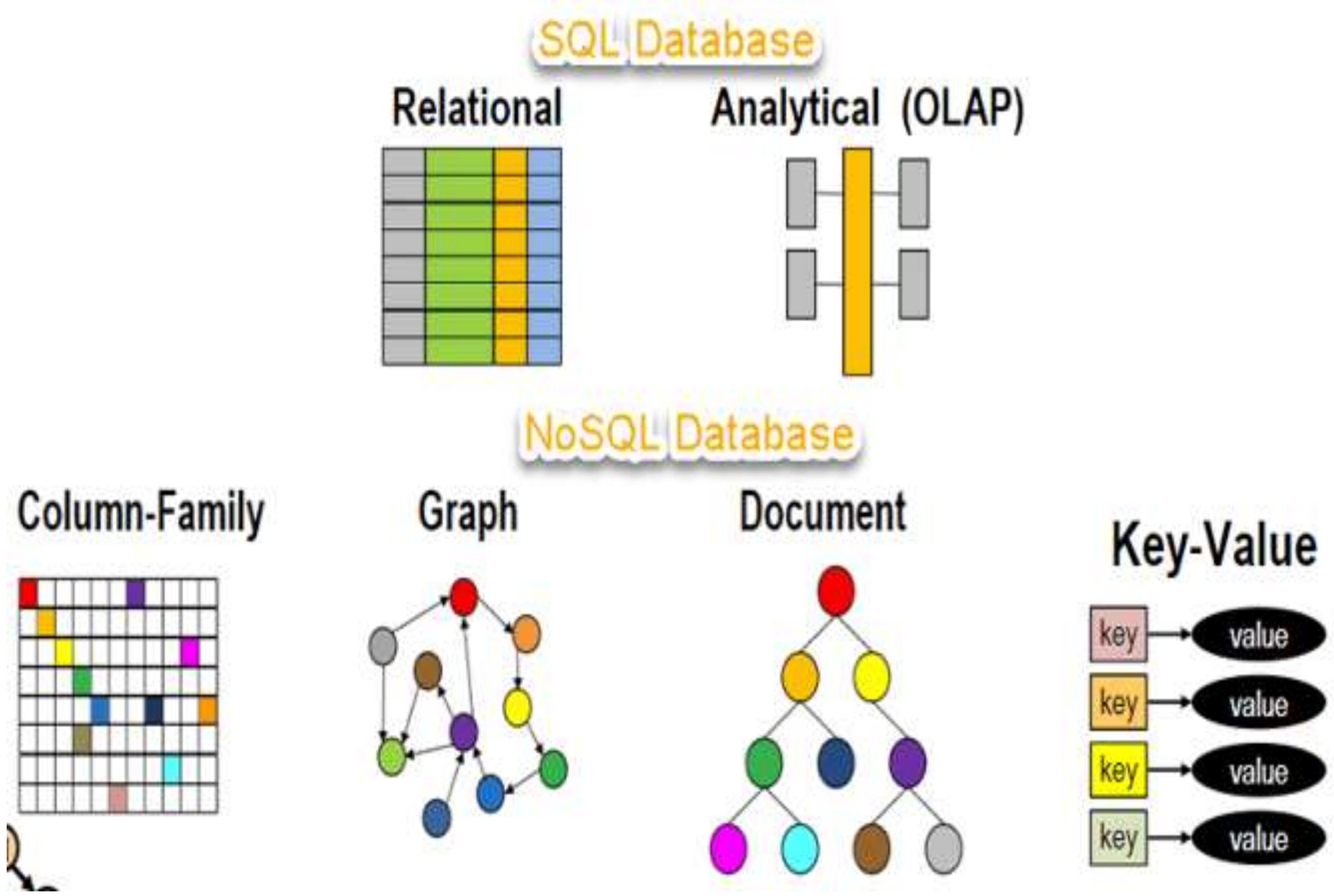
Carl Strozz introduced the NoSQL concept in 1998.

What is NoSQL?

Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data.

Let's understand about NoSQL with a diagram in this NoSQL database

SQL and NoSQL?



NoSQL

- Why NoSQL?
- Brief History of NoSQL Databases
- Features of NoSQL
- Types of NoSQL Databases
- Query Mechanism tools for NoSQL
- What is the CAP Theorem?
- Eventual Consistency
- Advantages and disadvantages of NoSQL

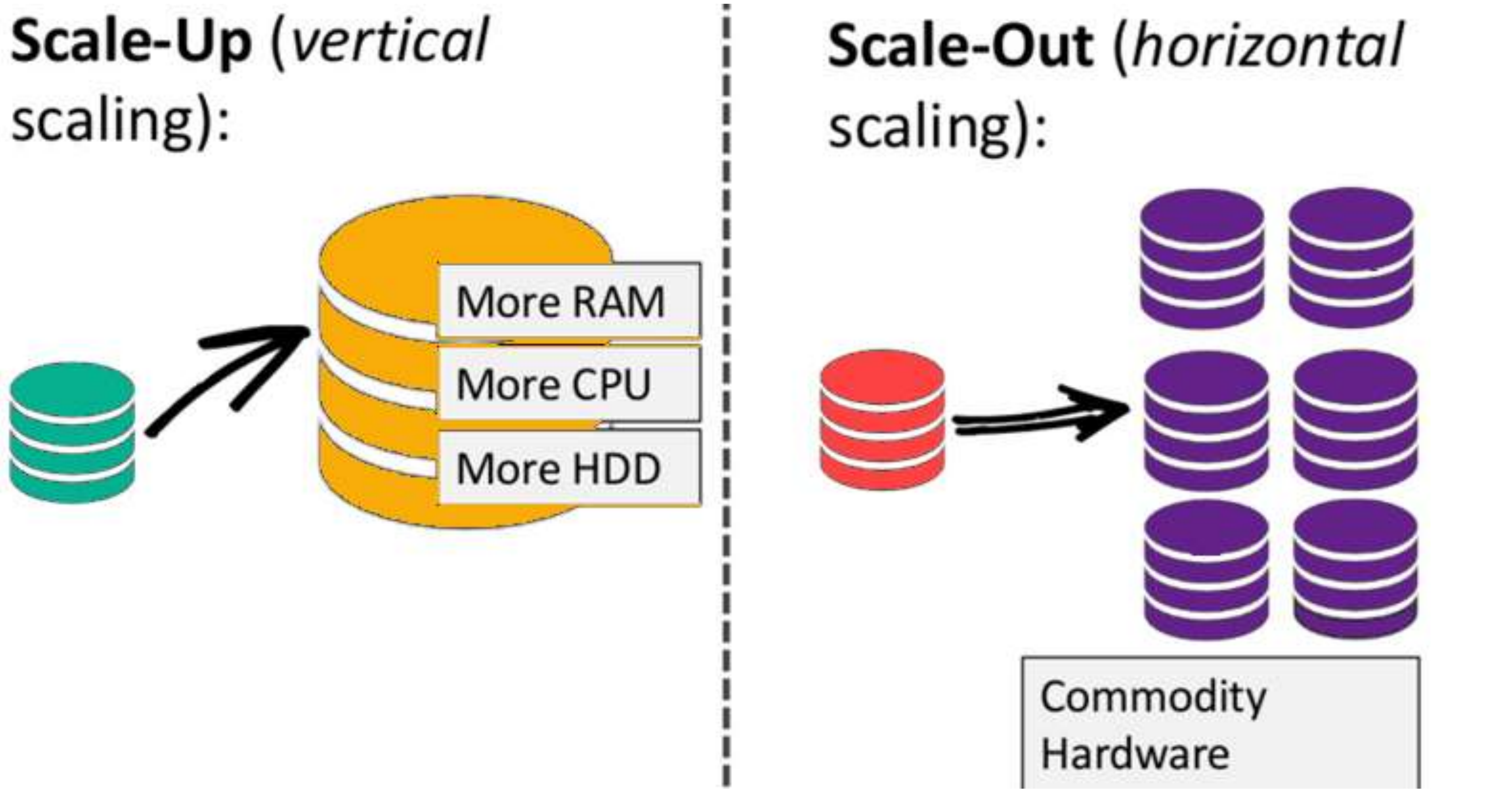
NoSQL

The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.

To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.

The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out."

NoSQL


Scale-Up (*vertical* scaling):
More RAM
More CPU
More HDD
Scale-Out (*horizontal* scaling):
Commodity Hardware

NoSQL

NoSQL database is non-relational, so it scales out better than relational databases as they are designed with web applications in mind.

Brief History of NoSQL Databases

1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database

2000- Graph database Neo4j is launched

2004- Google BigTable is launched

2005- CouchDB is launched

2007- The research paper on Amazon Dynamo is released

2008- Facebooks open sources the Cassandra project

2009- The term NoSQL was reintroduced

Features of NoSQL

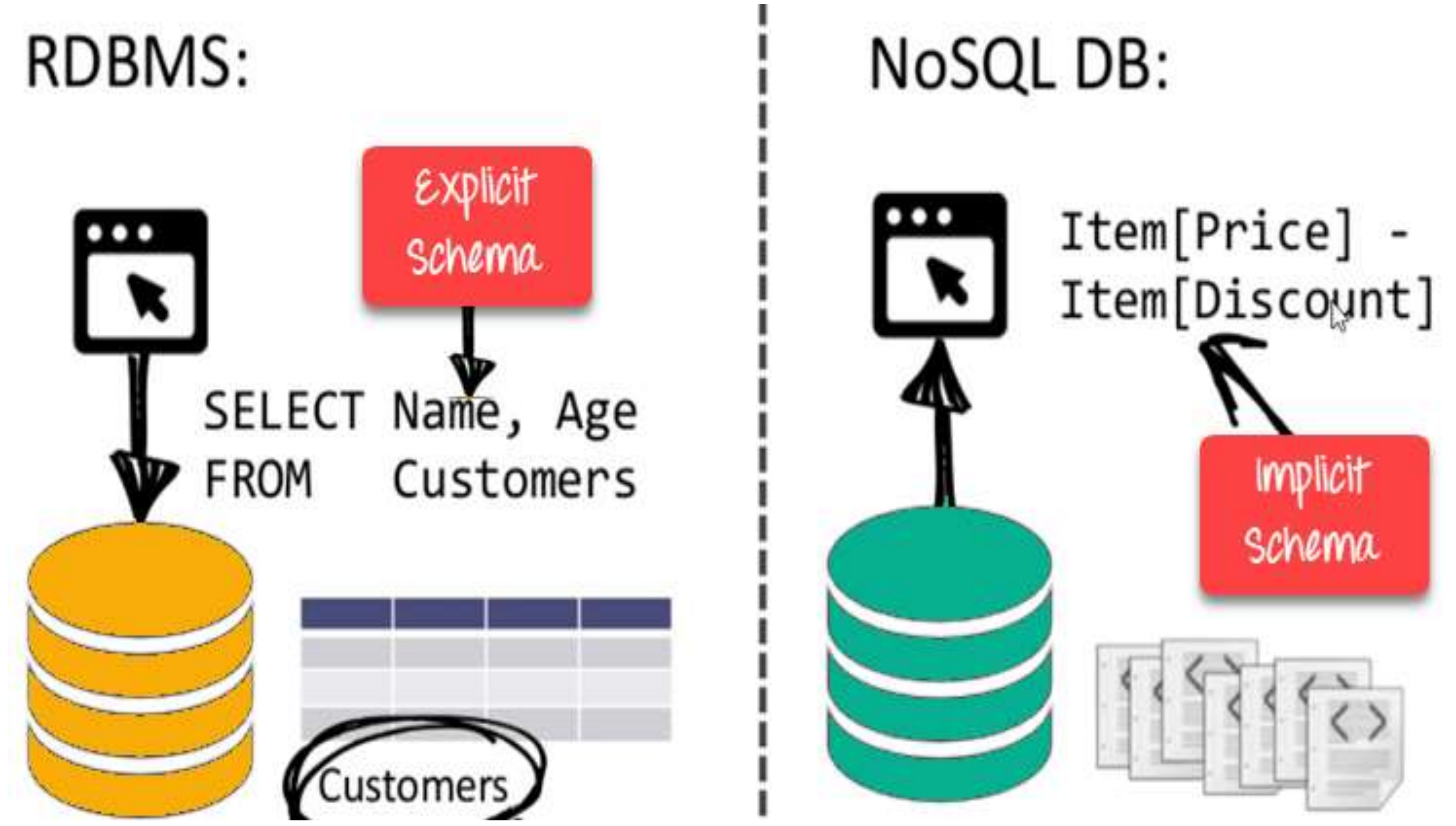
Non-relational

NoSQL databases never follow the relational model
Never provide tables with flat fixed-column records
Work with self-contained aggregates or BLOBs
Doesn't require object-relational mapping and data normalization
No complex features like query languages, query planners, referential integrity joins, ACID

Schema-free

NoSQL databases are either schema-free or have relaxed schemas
Do not require any sort of definition of the schema of the data
Offers heterogeneous structures of data in the same domain

Features of NoSQL



Features of NoSQL

Simple API

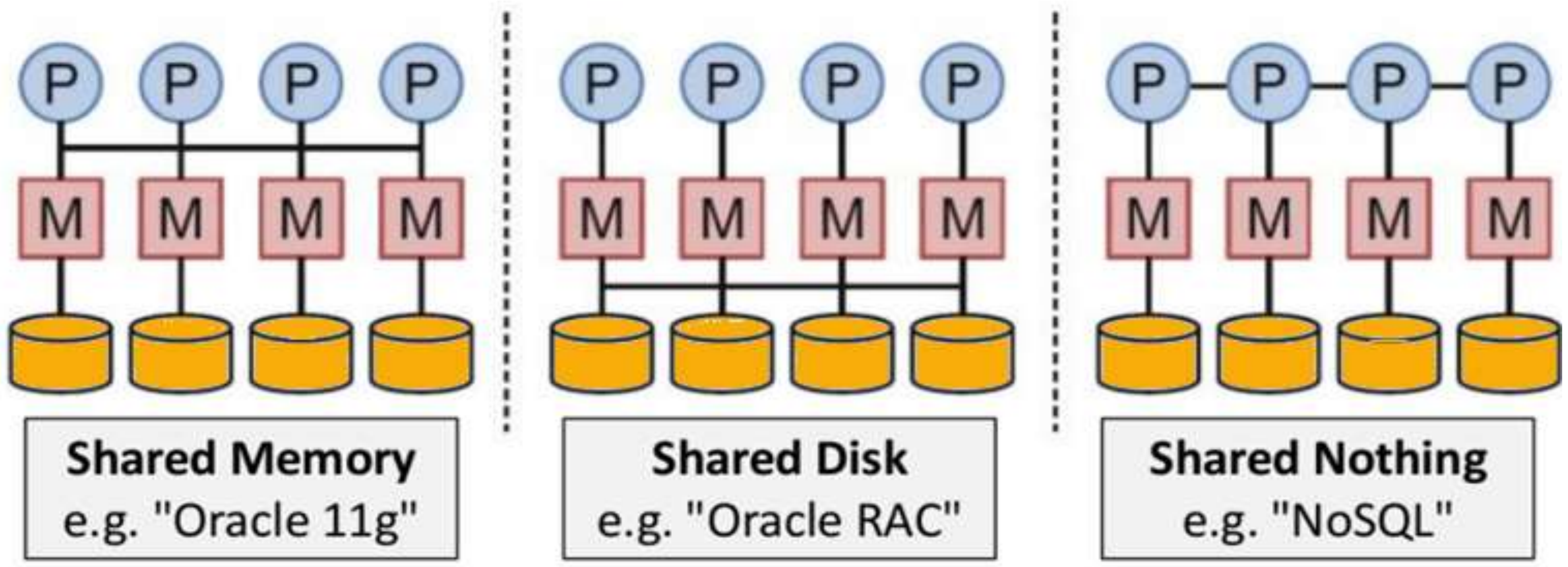
Offers easy to use interfaces for storage and querying data provided
APIs allow low-level data manipulation & selection methods
Text-based protocols mostly used with HTTP REST with JSON
Mostly used no standard based NoSQL query language
Web-enabled databases running as internet-facing services

Distributed

Multiple NoSQL databases can be executed in a distributed fashion
Offers auto-scaling and fail-over capabilities
Often ACID concept can be sacrificed for scalability and throughput
Mostly no synchronous replication between distributed nodes Asynchronous Multi-Master Replication, peer-to-peer, HDFS Replication
Only providing eventual consistency

Features of NoSQL

Shared Nothing Architecture. This enables less coordination and higher distribution.



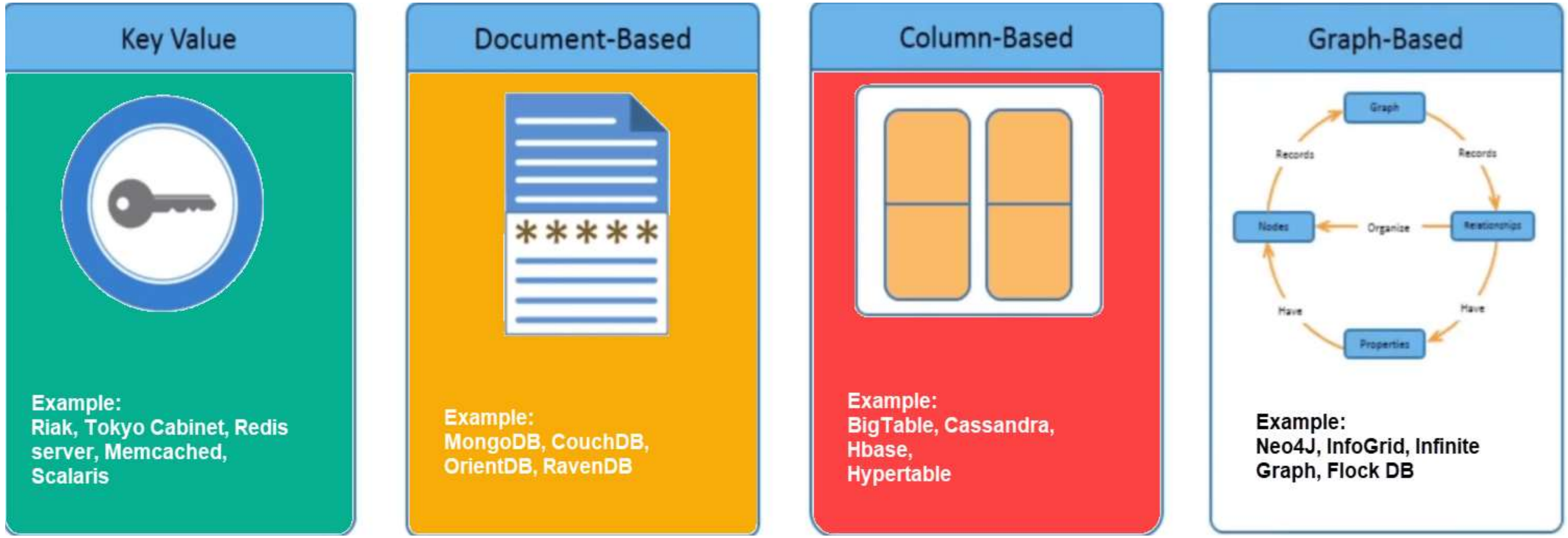
Types of NoSQL Databases

NoSQL Databases are mainly categorized into four types: Key-value pair, Column-oriented, Graph-based and Document-oriented. Every category has its unique attributes and limitations. None of the above-specified database is better to solve all the problems. Users should select the database based on their product needs.

Types of NoSQL Databases:

Key-value Pair Based
Column-oriented Graph
Graphs based
Document-oriented

Types of NoSQL Databases



Key Value

Example:
Riak, Tokyo Cabinet, Redis server, Memcached, Scalaris

Document-Based

Example:
MongoDB, CouchDB, OrientDB, RavenDB

Column-Based

Example:
BigTable, Cassandra, Hbase, Hypertable

Graph-Based

Example:
Neo4J, InfoGrid, Infinite Graph, Flock DB

Key Value Pair Based

Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load.

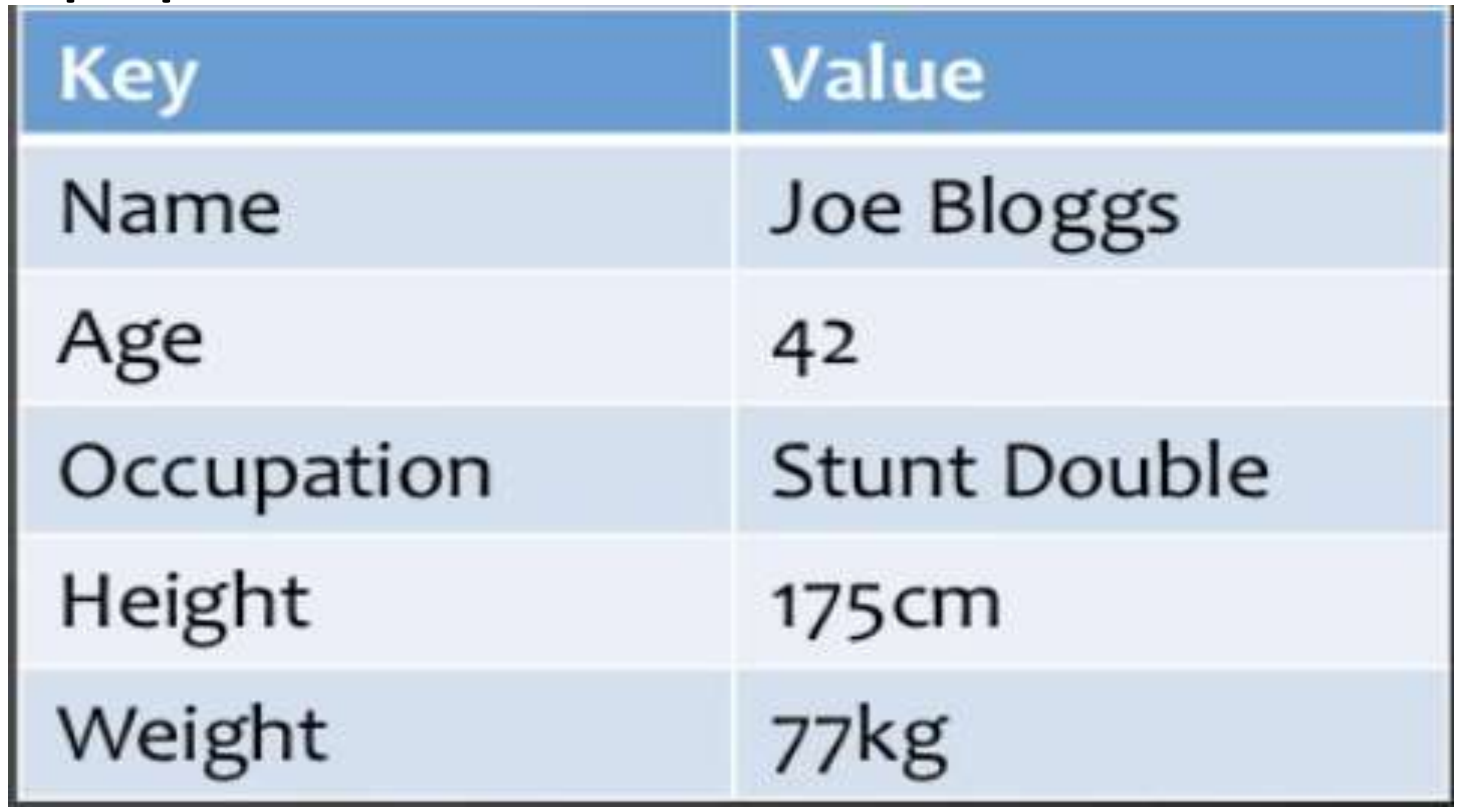
Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.

For example, a key-value pair may contain a key like "Website" associated with a value like “agri".

Key Value Pair Based

It is one of the most basic NoSQL database example. This kind of NoSQL database is used as a collection, dictionaries, associative arrays, etc. Key value stores help the developer to store schema-less data. They work best for shopping cart contents.

Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases. They are all based on Amazon's Dynamo paper.



Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

Key Value Pair Based

Example 1: Storing User Data in Redis (JSON Format)

You might store user data in Redis with the user ID as the key and the user's details as the value.

Key: user:1001

Value (JSON format):

```
{
 "name": "Alice",
 "age": 30,
 "email": "alice@example.com",
 "location": "New York"
}
```

To set this in Redis:

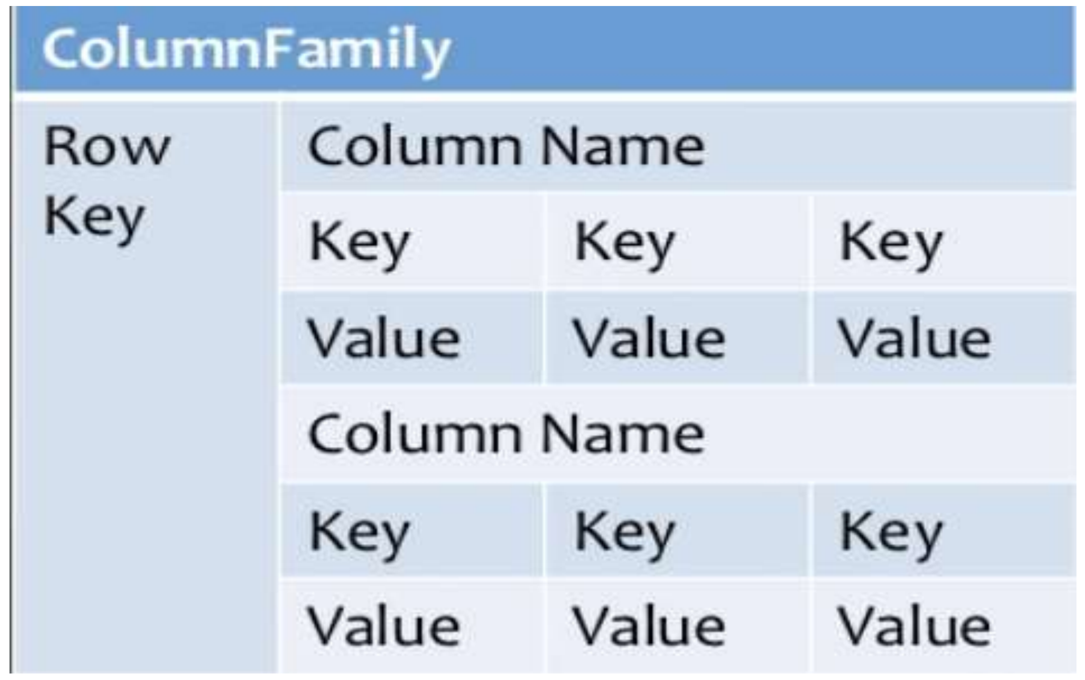
```
SET user:1001 '{"name": "Alice", "age": 30, "email": "alice@example.com", "location": "New York"}‘
```

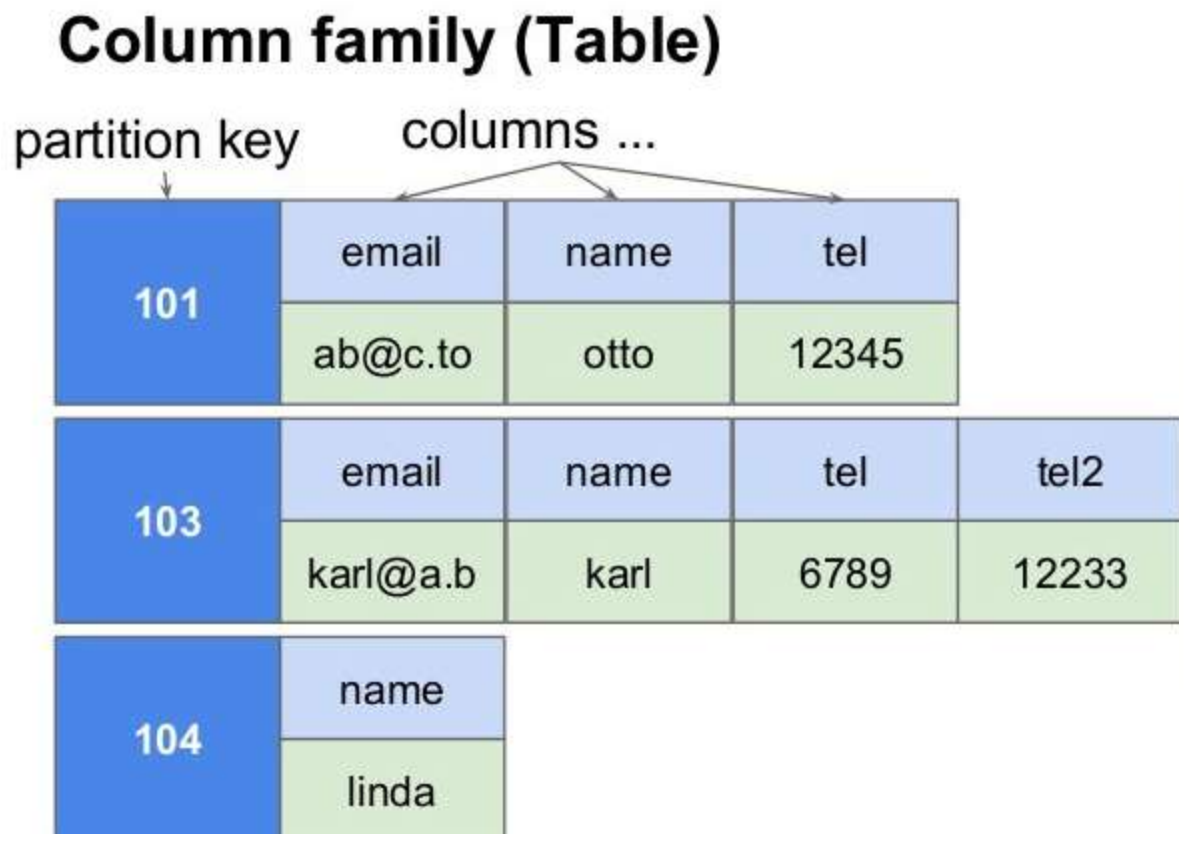
To retrieve the data:

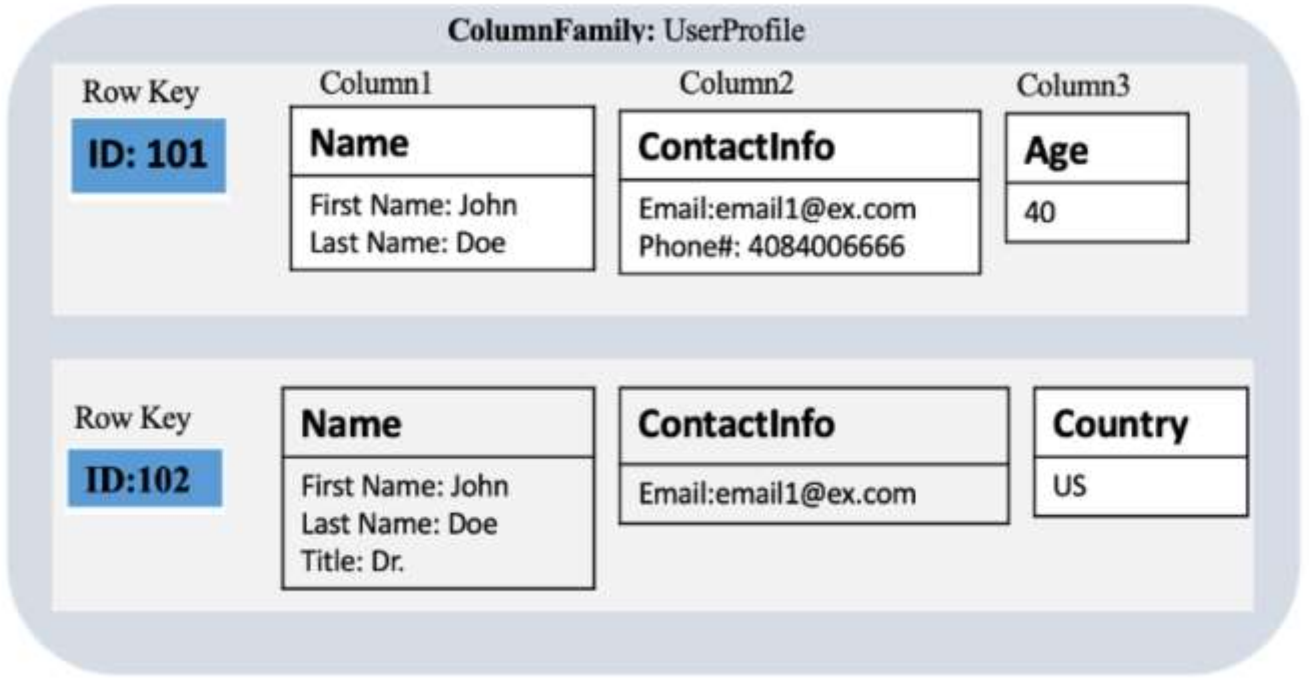
```
GET user:1001
```

Column-based

Column-oriented databases work on columns and are based on BigTable paper by Google. Every column is treated separately. Values of single column databases are stored contiguously.







Column-based

Example Table (Column Family) in HBase:

```
create 'users', 'personal_info', 'social_info‘
```

Example row in HBase:

```
rowkey: 123e4567-e89b-12d3-a456-426614174000
columns:
  personal_info:name -> John Doe
  personal_info:age -> 29
  personal_info:email -> john@example.com
  social_info:friends -> Jane, Alice
```

Column-based

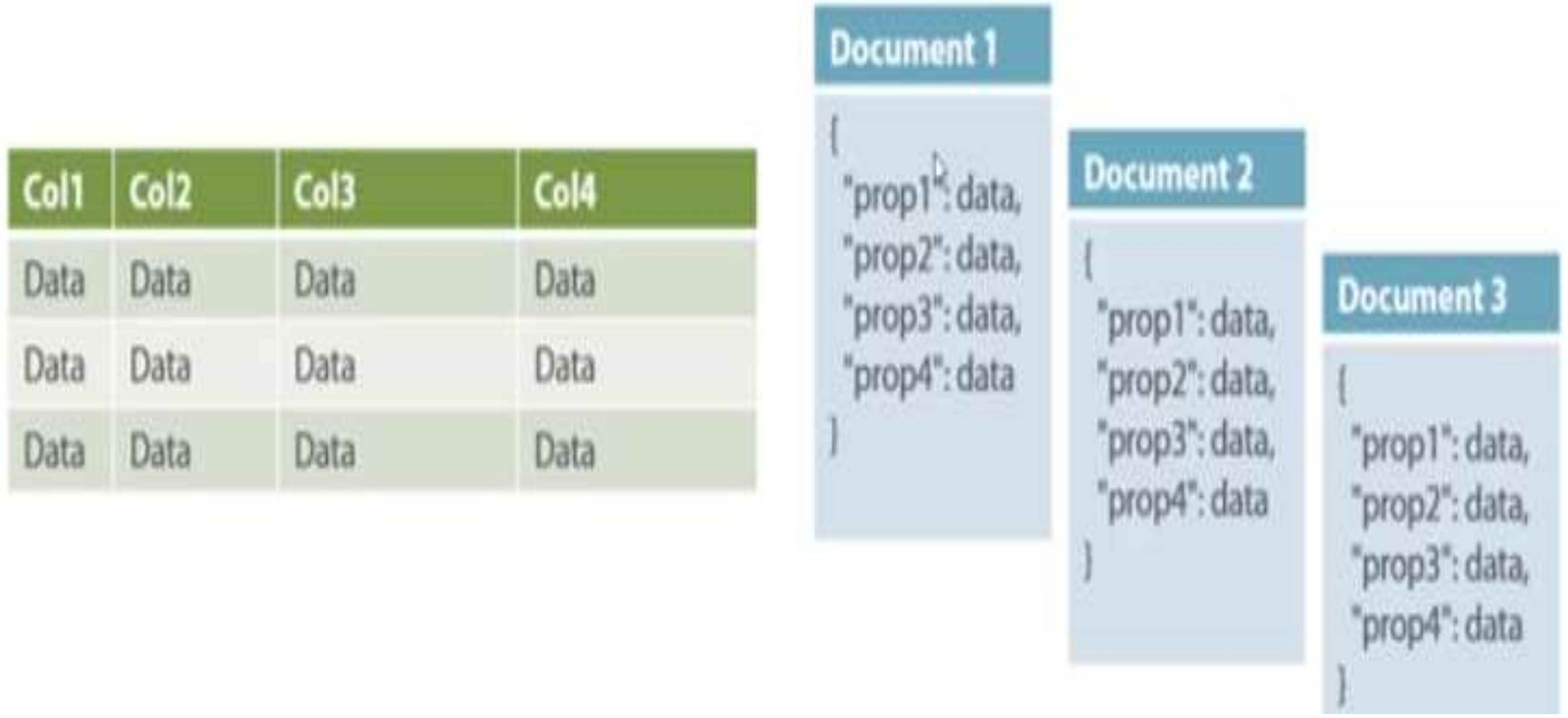
They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.

Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, Library card catalogs,

HBase, Cassandra, HBase, Hypertable are NoSQL query examples of column based database.

Document-Oriented

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.



Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Document 1
```
{
 "prop1":data,
 "prop2":data,
 "prop3":data,
 "prop4":data
}
```

Document 2
```
{
 "prop1":data,
 "prop2":data,
 "prop3":data,
 "prop4":data
}
```

Document 3
```
{
 "prop1":data,
 "prop2":data,
 "prop3":data,
 "prop4":data
}
```

Document-Oriented

Example Document in MongoDB:

```
{
 "_id": ObjectId("60b8d295f1b1c3b3f1d212c9"),
 "name": "John Doe",
 "age": 29,
 "address": {
  "street": "123 Main St",
  "city": "Metropolis",
  "state": "NY"
 },
 "email": "johndoe@example.com",
 "orders": [
  {
   "order_id": "A123",
   "date": "2024-12-20",
   "items": ["Laptop", "Mouse"]
  },
  {
   "order_id": "B456",
   "date": "2024-12-19",
   "items": ["Smartphone"]
  }
 ]
}
```

Document-Oriented

Example Document in CouchDB:

```
{
 "_id": "user-123",
 "_rev": "1-2345abcd",
 "type": "user",
 "name": "Jane Smith",
 "email": "janesmith@example.com",
 "preferences": {
  "theme": "dark",
  "language": "en"
 }
}
```

Document-Oriented

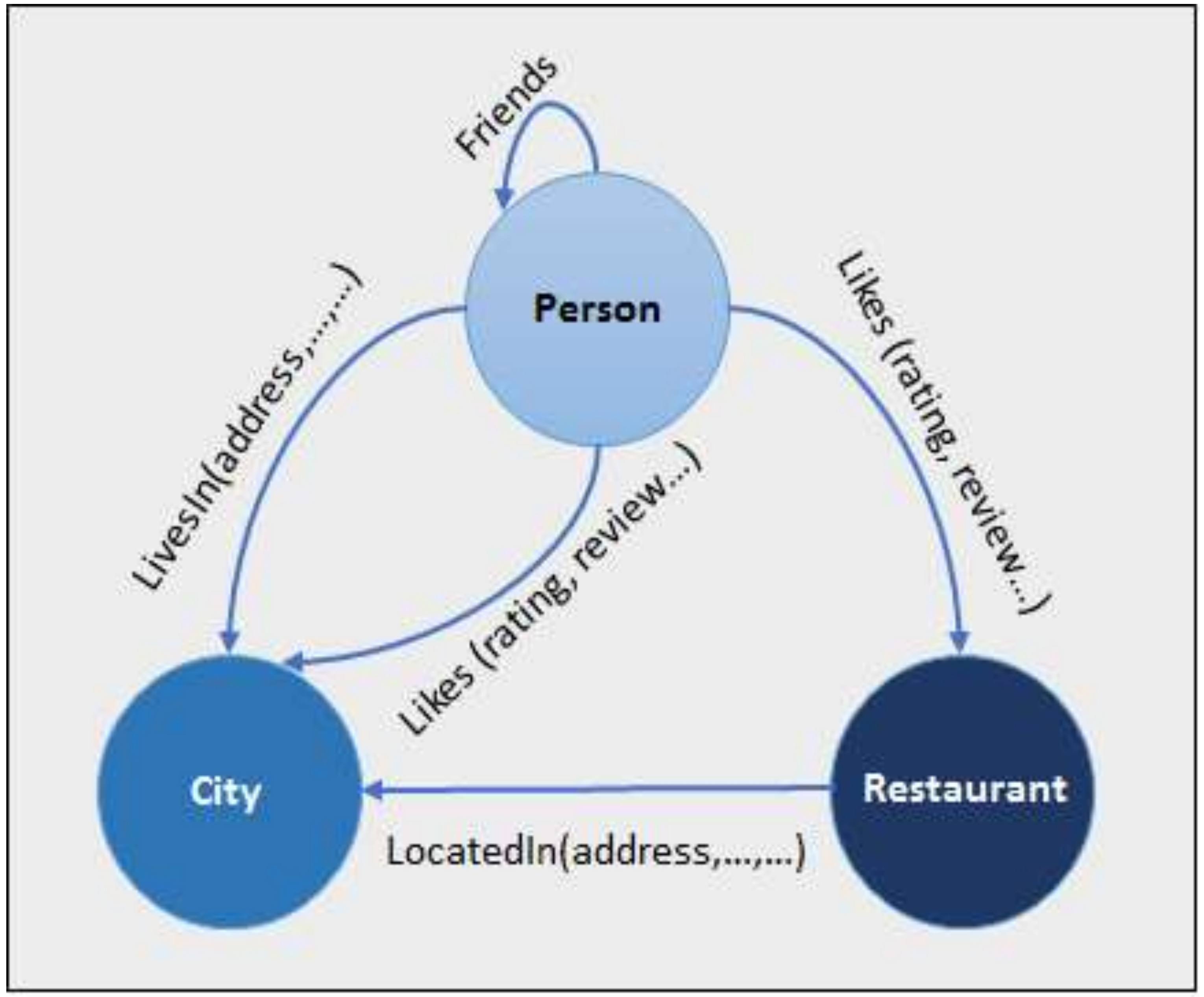
In this diagram on your left you can see we have rows and columns, and in the right, we have a document database which has a similar structure to JSON. Now for the relational database, you have to know what columns you have and so on. However, for a document database, you have data store like JSON object. You do not require to define which make it flexible.

The document type is mostly used for CMS systems, blogging platforms, real-time analytics & e-commerce applications. It should not use for complex transactions which require multiple operations or queries against varying aggregate structures.

Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes, MongoDB, are popular Document originated DBMS systems.

Graph-Based

A graph type database stores entities as well the relations amongst those entities. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge has a unique identifier.


Friends
Person
LivesIn(address,.......)
Likes (rating, review...)
Likes (rating, review...)
City
Restaurant
LocatedIn(address,...,...)

Graph-Based

Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature. Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them.

Graph base database mostly used for **social networks, logistics, spatial data.**

Neo4J, Infinite Graph, OrientDB, FlockDB are some popular graph-based databases.

Query Mechanism tools for NoSQL
The most common data retrieval mechanism is the REST-based retrieval of a value based on its key/ID with GET resource

Document store Database offers more difficult queries as they understand the value in a key-value pair. For example, CouchDB allows defining views with MapReduce

Graph-Based

Example: A simple social network where users are connected through friendships.

```
CREATE (alice:Person {name: 'Alice', age: 30}),
       (bob:Person {name: 'Bob', age: 25}),
       (carol:Person {name: 'Carol', age: 27}),
       (alice)-[:FRIEND_WITH]->(bob),
       (bob)-[:FRIEND_WITH]->(carol)
```

This Cypher query creates three nodes (Alice, Bob, and Carol) and defines relationships between them, specifically that Alice and Bob are friends, and Bob and Carol are friends.

Query: To find all of Bob's friends:

```
MATCH (bob:Person {name: 'Bob'})-[:FRIEND_WITH]->(friend)
RETURN friend.name;
```

What is the CAP Theorem?

CAP theorem is also called brewer's theorem. It states that is impossible for a distributed data store to offer more than two out of three guarantees

Consistency
Availability
Partition Tolerance

Consistency:

The data should remain consistent even after the execution of an operation. This means once data is written, any future read request should contain that data. For example, after updating the order status, all the clients should be able to see the same data.

What is the CAP Theorem?

Availability:

The database should always be available and responsive. It should not have any downtime.

Partition Tolerance:

Partition Tolerance means that the system should continue to function even if the communication among the servers is not stable. For example, the servers can be partitioned into multiple groups which may not communicate with each other. Here, if part of the database is unavailable, other parts are always unaffected.

What is the CAP Theorem?

Eventual Consistency

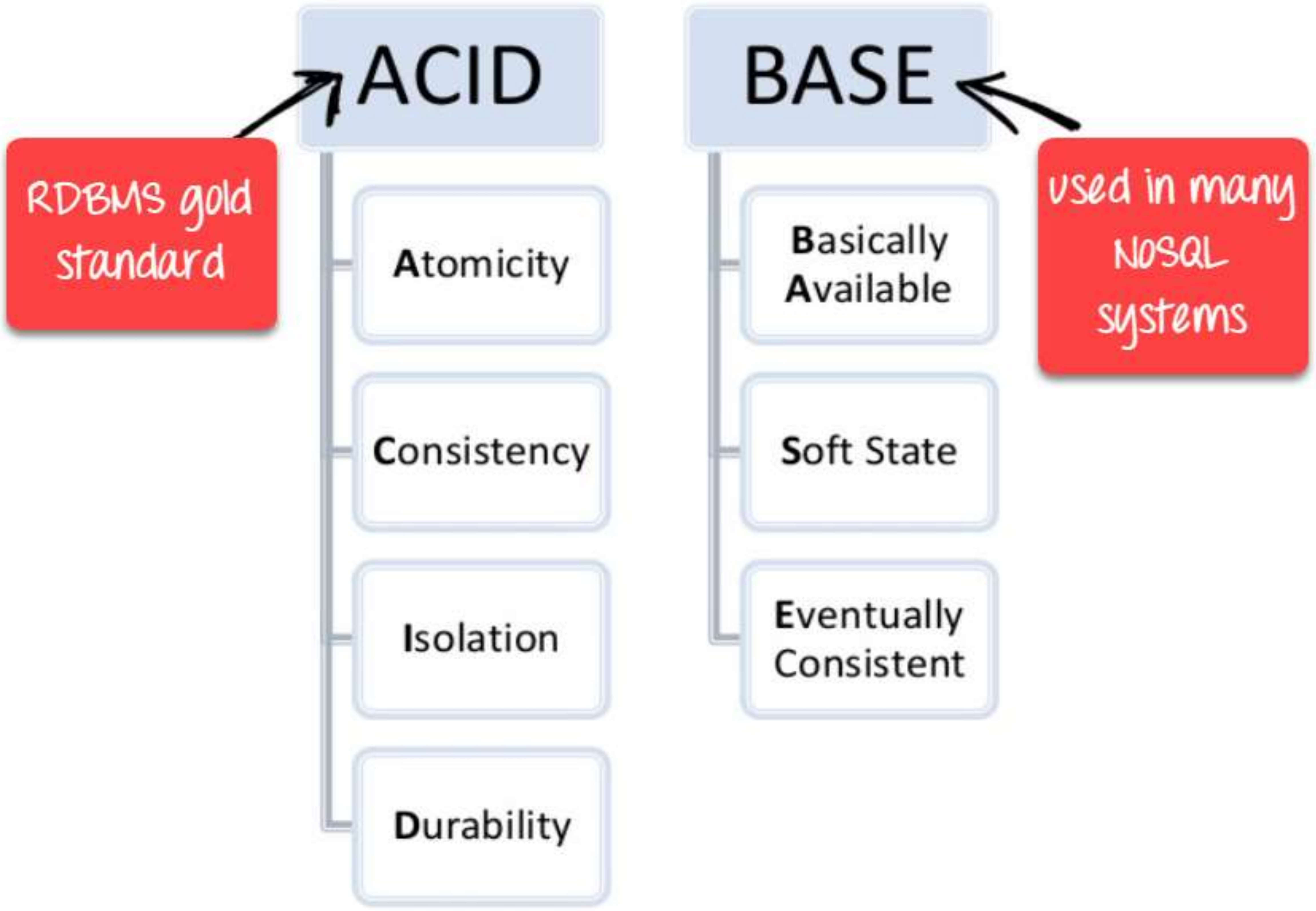
The term "eventual consistency" means to have copies of data on multiple machines to get high availability and scalability. Thus, changes made to any data item on one machine has to be propagated to other replicas.

Data replication may not be instantaneous as some copies will be updated immediately while others in due course of time. These copies may be mutually, but in due course of time, they become consistent. Hence, the name eventual consistency.

BASE: Basically Available, Soft state, Eventual consistency

Basically, available means DB is available all the time as per CAP theorem
Soft state means even without an input; the system state may change
Eventual consistency means that the system will become consistent over time

What is the CAP Theorem?



Advantages of NoSQL

- Can be used as Primary or Analytic Data Source
- Big Data Capability
- No Single Point of Failure
- Easy Replication
- No Need for Separate Caching Layer
- It provides fast performance and horizontal scalability.
- Can handle structured, semi-structured, and unstructured data with equal effect
- Object-oriented programming which is easy to use and flexible
- NoSQL databases don't need a dedicated high-performance server
- Support Key Developer Languages and Platforms
- Simple to implement than using RDBMS
- It can serve as the primary data source for online applications.
- Handles big data which manages data velocity, variety, volume, and complexity
- Excels at distributed database and multi-data center operations
- Eliminates the need for a specific caching layer to store data
- Offers a flexible schema design which can easily be altered without downtime or service disruption

Disadvantages of NoSQL

- No standardization rules
- Limited query capabilities
- RDBMS databases and tools are comparatively mature
- It does not offer any traditional database capabilities, like consistency when multiple transactions are performed simultaneously.
- When the volume of data increases it is difficult to maintain unique values as keys become difficult
- Doesn't work as well with relational data
- The learning curve is stiff for new developers
- Open source options so not so popular for enterprises.

Example NoSQL JSON

- Example API Response (Weather Data):

```
{
 "location": {
  "city": "New York",
  "country": "USA",
  "latitude": 40.7128,
  "longitude": -74.0060
 },
 "current_weather": {
  "temperature": 18.5,
  "humidity": 60,
  "description": "Clear sky",
  "wind": {
   "speed": 5.4,
   "direction": "North"
  }
 },
 "forecast": [
  {
   "day": "2024-12-21",
   "temperature_min": 16,
   "temperature_max": 21,
   "precipitation": 0,
   "description": "Partly cloudy"
  },
  {
   "day": "2024-12-22",
   "temperature_min": 14,
   "temperature_max": 19,
   "precipitation": 30,
   "description": "Showers"
  }
 ]
}
```

Example JSON

location: Contains details about the location, like city, country, latitude, and longitude.
current_weather: Provides the current temperature, humidity, weather description, and wind details.
forecast: An array of forecasted weather for upcoming days, with details like temperature and precipitation.
This JSON format allows a client application (like a weather app) to easily parse and display relevant weather information.

Reference

1. https://www.tutorialspoint.com/dbms/dbms_file_structure.htm#:~:text=Relative%20data%20and%20information%20is,blocks%20that%20can%20store%20records.
2. https://www.javatpoint.com/dbms-file-organization
3. https://www.tutorialspoint.com/dbms/dbms_storage_system.htm

THANK YOU