INTRODUCTION TO No SQL DATABASE
Course Code: 210304601

BCA

Semester-5

Prepared By: Prof.(Dr.) Shital Patel

Overview of all Unit

- **Unit-1 Introduction to No SQL**
- **Unit-2 No SQL Data Architecture Pattern**
- **Unit-3 Performing CRUD Operations on No SQL**
- **Unit-4 Using No SQL to manage big data**
- **Unit-5 Selecting the right No SQL Solution**

Unit-1

Introduction to No SQL

Content

- **Introduction to No SQL**
- Why No SQL?
- History of No SQL Databases
- Features of No SQL
- No SQL Business drivers
- Four types of No SQL Database
- What is the CAP Theorem?
- Advantages and Disadvantages of No SQL

Why NoSQL

- NoSQL is preferred over SQL in many cases because it offers more flexibility and scalability.
- The primary benefit of using a NoSQL system is that it provides developers with the ability to store and access data quickly and easily, without the overhead of a traditional relational database.
- As a result, development teams can focus on delivering features and core business logic faster, without worrying about the underlying data storage implementation.

What is NoSQL and what is a NoSQL database?

- NoSQL is a type of database management system (DBMS) that is designed to handle and
- store large volumes of unstructured and semi-structured data.
- NoSQL databases use flexible data models that can adapt to changes in data structures and
- are capable of scaling horizontally to handle growing amounts of data.
- The term NoSQL originally referred to “non-SQL” or “non-relational” databases, but the
- term has since evolved to mean “not only SQL,” as NoSQL databases
- NoSQL database technology stores information in JSON documents instead of columns
- and rows used by relational databases

Comparison of SQL vs NoSQL

- With a basic understanding of what SQL vs NoSQL is, let's take a look at this quick comparison chart to see what sets the two apart:

SQL	NoSQL
Stands for Structured Query Language	Stands for Not Only SQL
Relational database management system (RDBMS)	Non-relational database management system
Suitable for structured data with predefined schema	Suitable for unstructured and semi-structured data

Data is stored in tables with columns and rows	Data is stored in collections or documents
Follows ACID properties (Atomicity, Consistency, Isolation, Durability) for transaction management	Does not necessarily follow ACID properties
Supports JOIN and complex queries	Does not support JOIN and complex queries
Uses normalized data structure	Uses denormalized data structure
Requires vertical scaling to handle large volumes of data	Horizontal scaling is possible to handle large volumes of data
Examples: MySQL, PostgreSQL, Oracle, SQL Server, Microsoft SQL Server	Examples: MongoDB, Cassandra, Couchbase, Amazon DynamoDB, Redis

SQL Databases

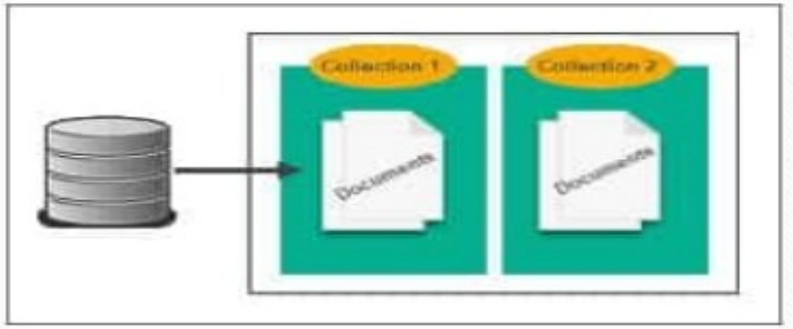
Table

ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

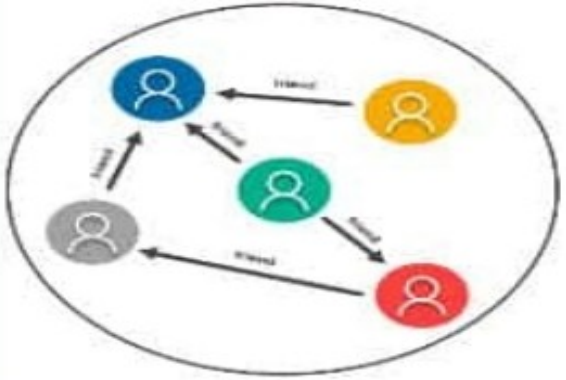
NoSQL Databases

Key-value

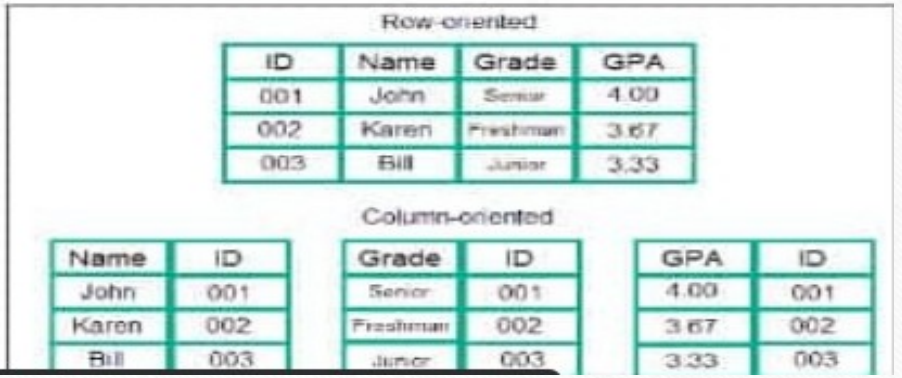
Document



Graph



Wide-column



History behind the creation of NoSQL Databases

- In the early 1970, Flat File Systems are used.
- Data were stored in flat files and the biggest problems with flat files are each company implement their own flat files and there are no standards.
- It is very difficult to store data in the files, retrieve data from files because there is no standard way to store data.
- Then the relational database was created by E.F. Codd and these databases answered the question of having no standard way to store data.

- But later relational database also get a problem that it could not handle big data, due to this problem there was a need of database which can handle every types of problems then NoSQL database was developed.
- The acronym NoSQL was first used in 1998 by Carlo Strozzi while naming his lightweight, open source “relational” database that did not use SQL.
- The name came up again in 2009 when Eric Evans and Johan Oskarsson used it to describe non-relational databases.
- Relational databases are often referred to as SQL systems. The term NoSQL can mean either “No SQL systems” or the more commonly accepted translation of “Not only SQL,” to emphasize the fact some systems might support SQL-like query languages. Flexible data structures, instead of standard tabular relationships.

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

What is Database

- Database is a collection of structured data or information which is stored in a computer system and can be accessed easily. A database is usually managed by a Database Management System (DBMS).
- NoSQL is a non-relational database that is used to store the data in the nontabular form. It is a method to database design that focuses on providing a mechanism for storage and retrieval of data that is suited in means other than the tabular relations used in relational databases.
- NoSQL stands for Not only SQL .

Features of NoSQL

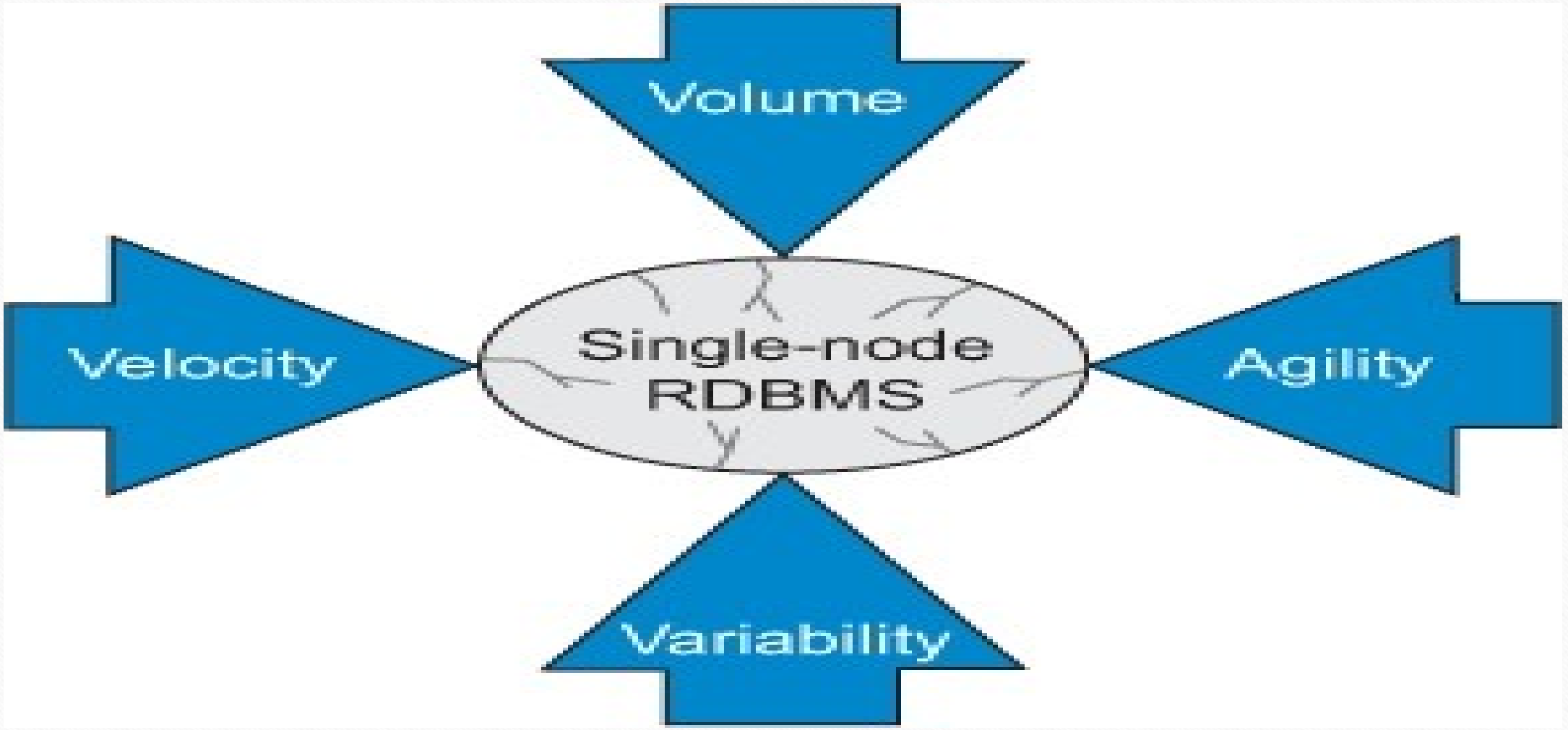
- **Low latency :** NoSQL databases are designed to be distributed across multiple servers,
- which allows them to handle large volumes of data, reduce the latency of data access, and
- provide faster response times.
- **Horizontal scalability :** This is done by increasing the number of servers parallelly.
- Large number of concurrent users supported.
- **Optimized for large data volumes** — either structured, semi-structured or unstructured.

Features of NoSQL

- **Distributed architecture** that allows handling bigger amounts of data.
- **Higher performance, speed and scalability**.
- **Non-relational :**NoSQL databases never follow the relational model, No complex features
- like query languages, query planners, referential integrity joins, ACID
- **Schema-free :** Do not require any sort of definition of the schema of the data
- **Simple API :** Offers easy to use interfaces for storage and querying data provide
- **Distributed :** Multiple NoSQL databases can be executed in a distributed fashion

NoSQL Business drivers

- NoSQL systems have unique characteristics and capabilities that can be used alone or in conjunction with your existing systems.
- Many organizations considering NoSQL systems do so to overcome common issues such as
- volume, velocity, variability, and agility, the business drivers behind the NoSQL movement.


Volume
Velocity
Single-node
RDBMS
Agility
Variability

NoSQL Business drivers

- In figure, we see how the business driver's volume, velocity, variability, and agility apply
- pressure to the single CPU system, resulting in the cracks. Volume and velocity refer to the ability
- to handle large datasets that arrive quickly. Variability refers to how diverse data types don’t fit into
- structured tables, and agility refers to how quickly an organization responds to business change.

Business drivers of NoSQL

- **Volume:**
- The need to scale out (also known as horizontal scaling), rather than scale up (faster processors), moved organizations from serial to parallel processing were data problems are split into separate paths and sent to separate processors to divide and conquer the work.

Business drivers of NoSQL

- **Velocity:**
- It refers to how quickly data is generated and how quickly that data moves. The ability of a single processor system to rapidly read and write data. When single processors RDBMSs are used as a back end to a web storefront, the random bursts in web traffic slow down response for everyone and tuning these systems can be costly when both high read and write throughput is desired.

Business drivers of NoSQL

- **Variability:**
- The number of inconsistencies in the data. Capturing and reporting on exception data struggle when attempting to use rigid database schema structures imposed by RDBMS systems. For example, if a business unit wants to capture a few custom fields for a particular customer, all customer rows within the database need to store this information even though it doesn't apply. Adding new columns to an RDBMS requires the system to be shut down and ALTER TABLE commands to be run. When a large database is large, this process can impact system availability, losing time and money.

Business drivers of NoSQL

- **Agility:**
- putting data into and getting data out of the database. If your data has nested and repeated subgroups of data structures you need to include an object-relational mapping layer. The responsibility of this layer is to generate the correct combination of INSERT, UPDATE, DELETE and SELECT SQL statements to move object data to and from the RDBMS persistence layer. This process is not simple and is associated with the largest barrier to rapid change when developing new or modifying existing applications

Types of No SQL Databases

- **No SQL Databases** are mainly categorized into four types:
  1. Key-value pair,
  2. Column-oriented,
  3. Graph-based and
  4. 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.

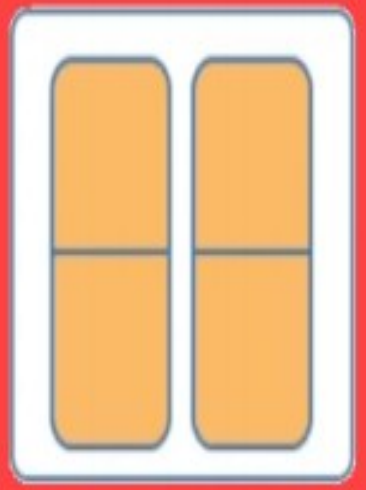
Key Value

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

Document-Based

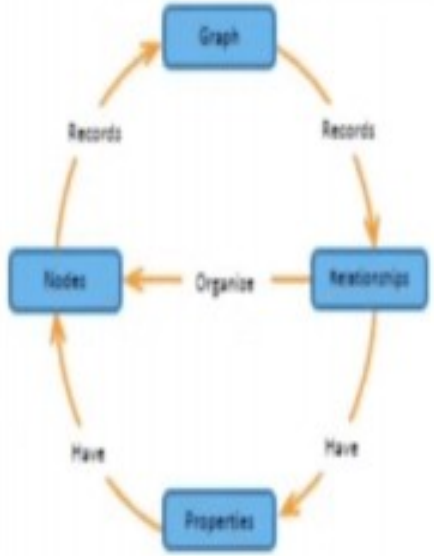
Example:
MongoDB, CouchDB, OrientDB, RavenDB

Column-Based



Example:
BigTable, Cassandra, Hbase, Hypertable

Graph-Based



Example:
Neo4J, InfoGrid, Infinite Graph, Flock DB

The CAP Theorem

• It is very important to understand the limitations of the NoSQL database. NoSQL can not provide **consistency** and **high availability** together.

• This was first expressed by **Eric Brewer** in **CAP** Theorem.

• The three letters in CAP refer to three desirable properties of distributed systems with replicated data: **Consistency (**among replicated copies), **Availability** (of the system for read and write operations) and **Partition tolerance** (in the face of the nodes in the system being partitioned by a network fault).

The CAP Theorem

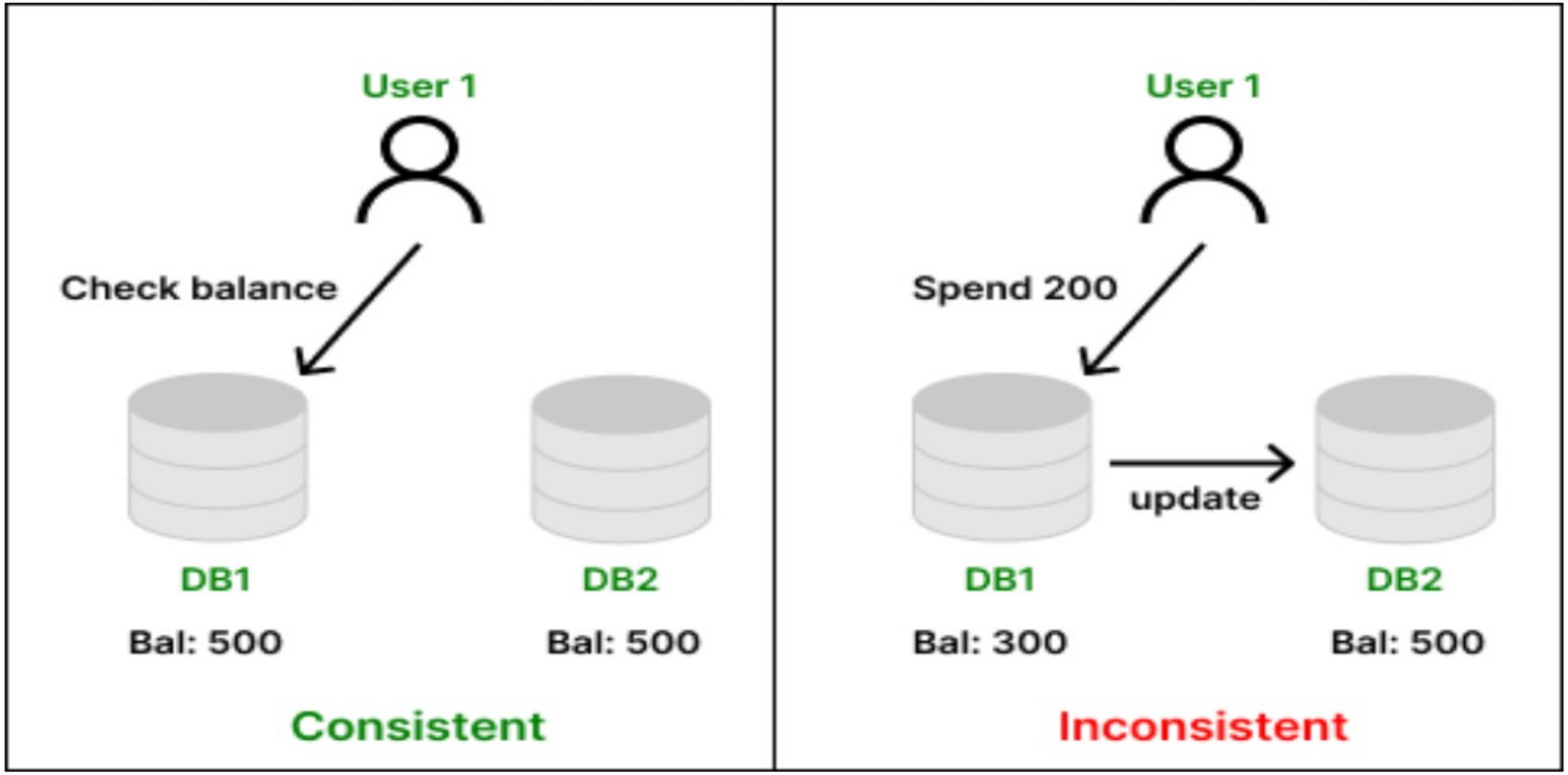
- **Consistency** means that all nodes in the network see the same data at the same time.
- **Availability** is a guarantee that every request receives a response about whether it was successful or failed. Every request receives a response, but no guarantee that it contains the most recent write.
- **Partition Tolerance** is a guarantee that the system continues to operate despite arbitrary message loss or failure of part of the system. In other words, even if there is a network outage in the data center and some of the computers are unreachable, still the system continues to perform.

Consistency

- Consistency means that all the nodes (databases) inside a network will have the same copies of a replicated data item visible for various transactions.
- It guarantees that every node in a distributed cluster returns the same, most recent, and successful write.
- It refers to every client having the same view of the data.
- There are various types of consistency models. Consistency in CAP refers to sequential consistency, a very strong form of consistency.

Example

- For example, a user checks his account balance and knows that he has 500 rupees. He spends 200 rupees on some products.
- Hence the amount of 200 must be deducted changing his account balance to 300 rupees. This change must be committed and communicated with all other databases that hold this user’s details.
- Otherwise, there will be inconsistency, and the other database might show his account balance as 500 rupees which is not true.



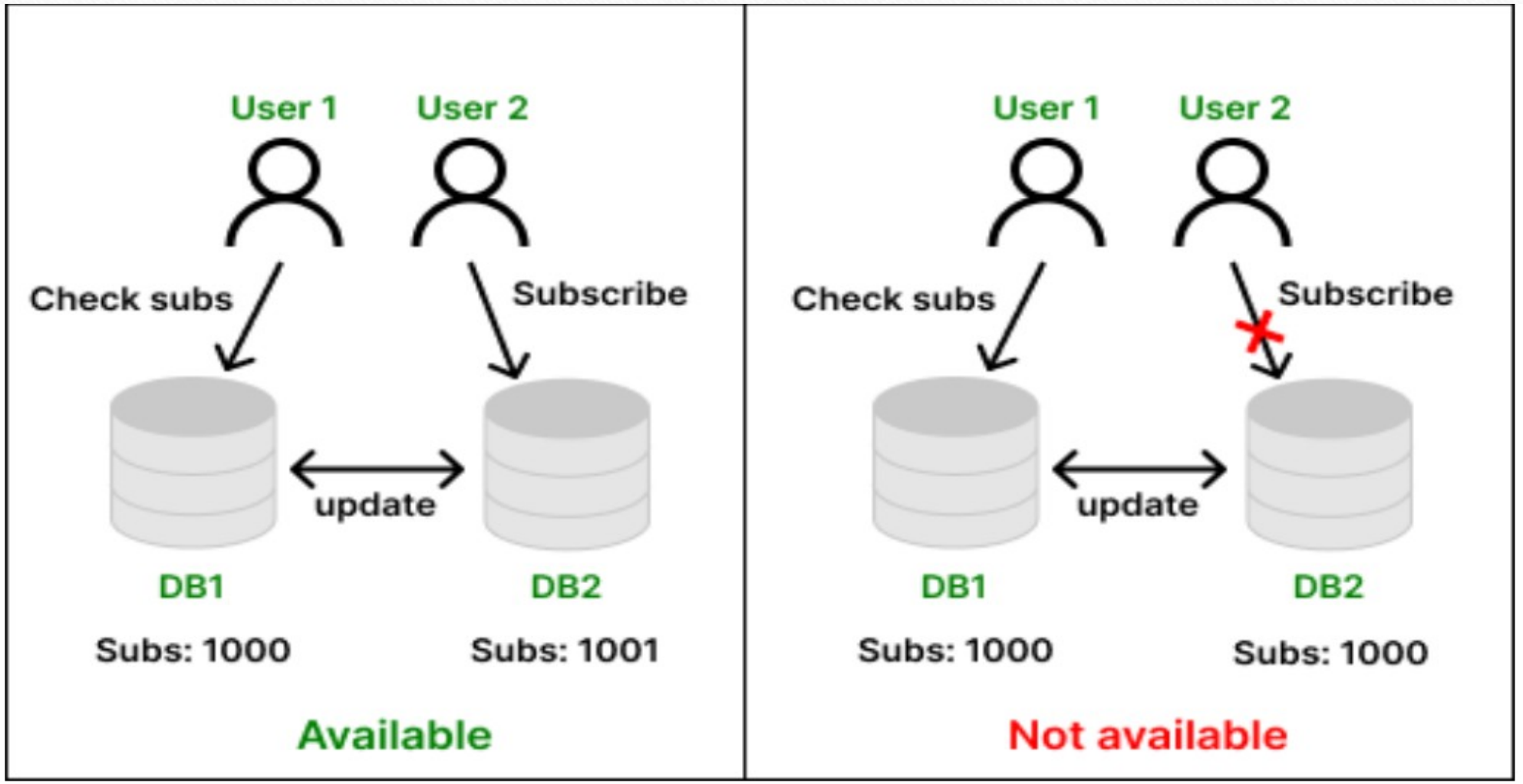
Consistency problem

Availability

- Availability means that each read or write request for a data item will either be processed successfully or will receive a message that the operation cannot be completed.
- Every non-failing node returns a response for all the read and write requests in a reasonable amount of time.
- The key word here is **“every”.**
- In simple terms, every node (on either side of a network partition) must be able to respond in a reasonable amount of time.

Example

- For example, user A is a content creator having 1000 other users subscribed to his channel.
- Another user B who is far away from user A tries to subscribe to user A’s channel.
- Since the distance between both users are huge, they are connected to different database node of the social media network.
- If the distributed system follows the principle of availability, user B must be able to subscribe to user A’s channel.



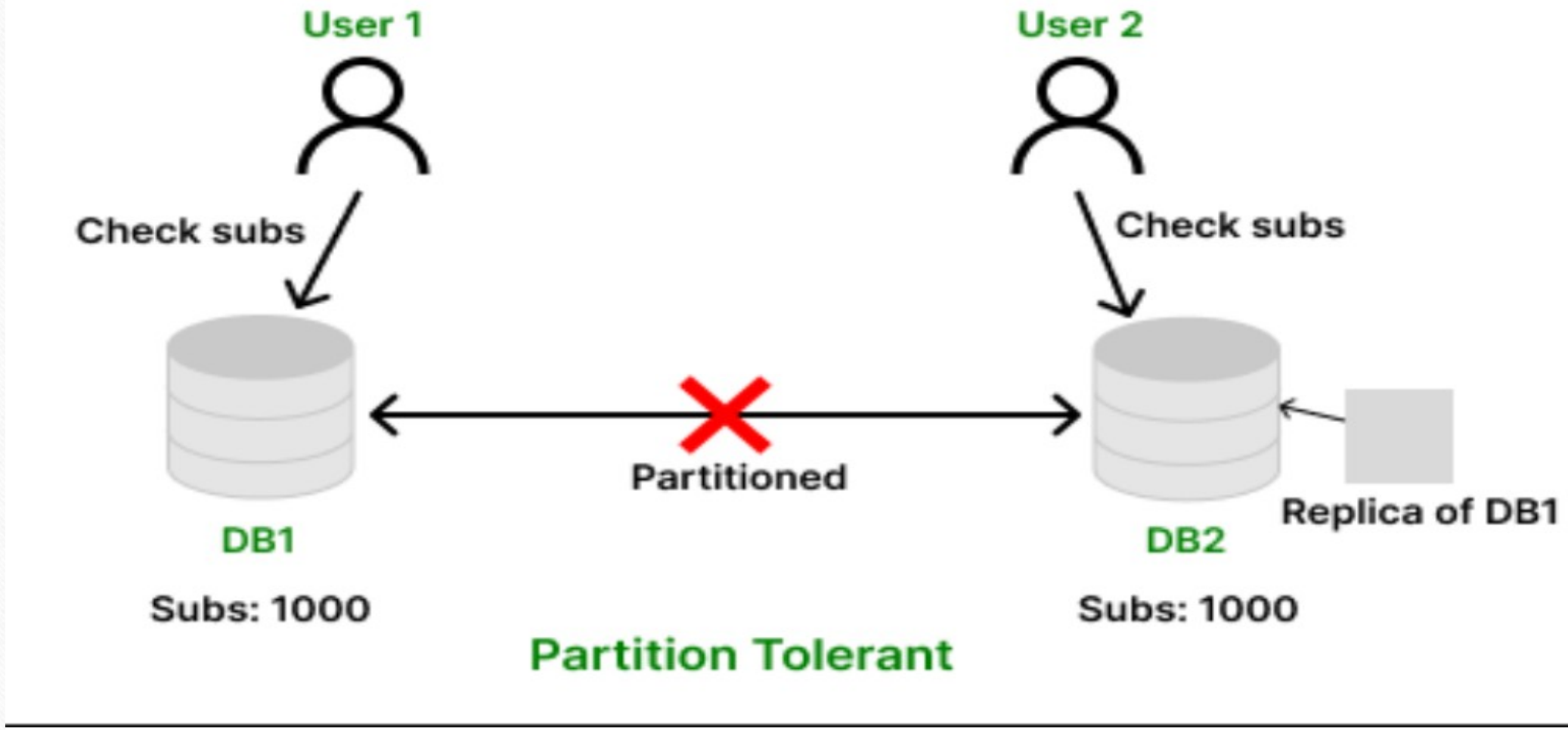
Availability problem

Partition tolerance

- Partition tolerance means that the system can continue operating even if the network connecting the nodes has a fault that results in two or more partitions, where the nodes in each partition can only communicate among each other.
- That means, the system continues to function and upholds its consistency guarantees in spite of network partitions.
- Network partitions are a fact of life.
- Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

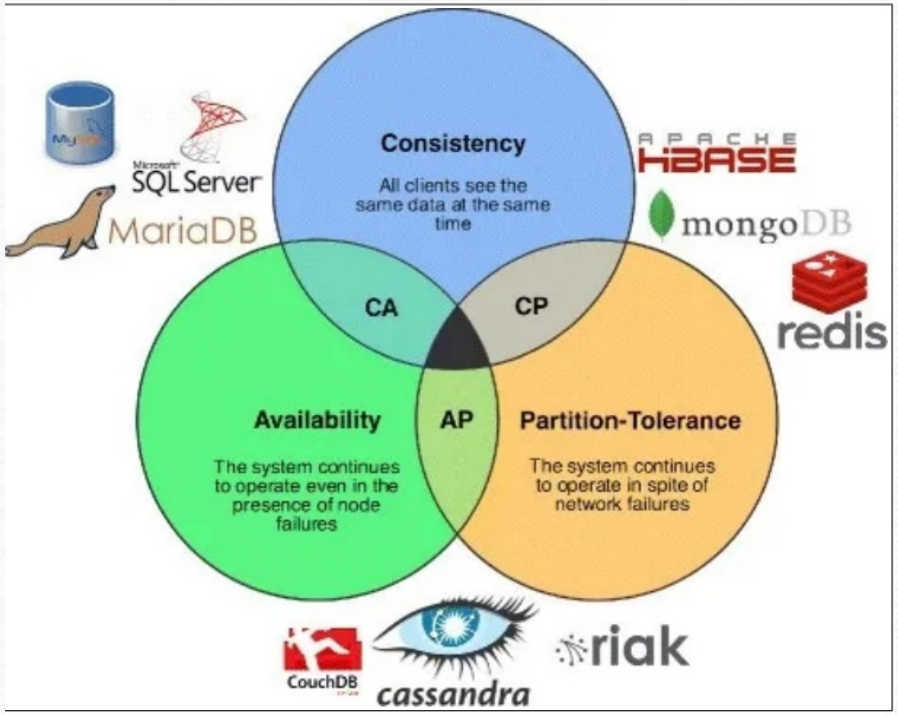
Example

- For example, take the example of the same social media network where two users are trying to find the subscriber count of a particular channel.
- Due to some technical fault, there occurs a network outage, the second database connected by user B losses its connection with first database.
- Hence the subscriber count is shown to the user B with the help of replica of data which was previously stored in database 1 backed up prior to network outage. Hence the distributed system is partition tolerant.



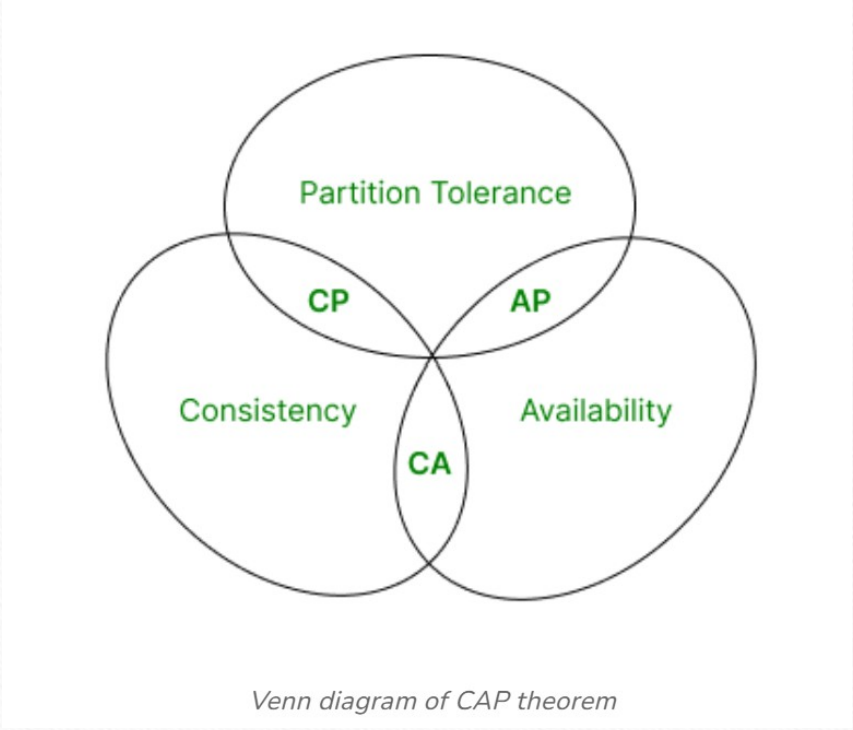
Partition Tolerance

- **CAP theorem** or **Eric Brewers theorem** states that we can only achieve at most two out of three guarantees for a database:
- Consistency, Availability, and Partition Tolerance.



CAP theorem

- The CAP theorem states that distributed databases can have at most two of the three properties:
- consistency, availability, and partition tolerance.
- As a result, database systems prioritize only two properties at a time.



Venn diagram of CAP theorem

Consistency vs availability

- One way to understand the trade-off between consistency and availability is to imagine a scenario where two users try to update the same data in a distributed system.
- If the system prioritizes **consistency**, it will ensure that both users see the same version of the data, but it may have to reject or delay some requests if there is a network partition.
- This means that the system sacrifices **availability for consistency.**
- On the other hand, if the system prioritizes **availability,** it will accept and respond to both requests, but it may allow different nodes to have different versions of the data.
- This means that the system sacrifices **consistency for availability.**

Each type of NoSQL database has its own advantages and disadvantages, and its own way of dealing with the CAP theorem.

• **Key-value databases** tend to favor **availability over consistency**, meaning that they can handle high availability and low latency, but they may suffer from data inconsistency or conflicts.

• **Document databases** tend to favor **consistency over availability,** meaning that they can handle complex queries and data integrity, but they may suffer from low availability or high latency.

• Column databases tend to be **hybrid,** meaning that they can balance between **consistency and availability**, depending on the configuration and the use case.

• Graph databases tend to favor **consistency over availability**, meaning that they can handle complex queries and data integrity, but they may suffer from low availability or high latency.

CAP theorem categorizes

- The CAP theorem categorizes systems into three categories:
- **AP (Available and Partition Tolerant) database:**
- An AP database delivers availability and partition tolerance at the expense of consistency.
- When a partition occurs, all nodes remain available but those at the wrong end of a partition might return an older version of data than others.
- When the partition is resolved, the AP databases typically resync the nodes to repair all inconsistencies in the system.

CAP theorem categorizes

- **CP (Consistent and Partition Tolerant) database:**
- A CP database delivers consistency and partition tolerance at the expense of availability.
- When a partition occurs between any two nodes,
- the system has to shut down the non-consistent node (i.e., make it unavailable) until the partition is resolved.

CAP theorem categorizes

- **CA (Consistent and Available) database:**
- A CA delivers consistency and availability in the absence of any network partition. Often a single node's DB servers are categorized as CA systems.
- Single node DB servers do not need to deal with partition tolerance and are thus considered CA systems.

Advantages of NoSQL

1. Flexible Data Structures –

- NoSQL databases allow for more flexible data structures than traditional relational databases.
- This means that data can be stored in a variety of formats,which is particularly useful when dealing with unstructured or semi-structured data, such as social media posts or log files.

2. Scalability

- NoSQL databases are highly scalable, which means they can easily handle large amounts of data and traffic.
- This is achieved through a distributed architecture than allows data to be spread across multiple servers, making it easy to add more servers as needed.

3. High Performance –

- NoSQL databases are designed for high performance, meaning that they can process large amounts of data quickly and efficiently.
- This is especially important for applications that require real-time data processing, such as financial trading platforms or social media analytics tools.

4. Availability –

- NoSQL databases are highly available, which means that they are designed to be up and running at all times. This is achieved through a distributed architecture that allows
- data to be replicated across multiple servers, ensuring that the system is always available,even if one or more servers fail.

5. Cost-Effective –

- NoSQL databases can be cost-effective, especially for large-scale applications.
- This is because they are designed to be run on commodity hardware, rather than expensive proprietary hardware, which can save companies a significant amount of money.

Disadvantages of NoSQL

1. Limited Query Capability –

- NoSQL databases offer limited query capability when compared to traditional relational databases.
- This is because NoSQL databases are designed to handle unstructured data, which can make it difficult to perform complex queries or retrieve data in a structured manner.

2. Data Consistency –

- NoSQL databases often sacrifice data consistency for scalability and performance.
- This means that there may be some lag time between when data is written to the database and when it is available for retrieval.
- Additionally, because NoSQL databases often use distributed architectures, there may be instances where different nodes of the database contain different versions of the same data.

3. Lack of Standardization –

- NoSQL databases lack standardization, meaning that different NoSQL databases can have vastly different structures and query languages.
- This can make it difficult for developers to learn and work with different NoSQL databases.

4. Limited Tooling –

- Because NoSQL databases are a relatively new technology, there is limited tooling available for them when compared to traditional relational databases.
- This can make it more difficult for developers to work with NoSQL databases and to debug issues when they arise.

5. Limited ACID Compliance –

- NoSQL databases often sacrifice ACID compliance for scalability and performance.
- ACID compliance refers to a set of properties that guarantee that database transactions are processed reliably.
- Because NoSQL databases often use distributed architectures and eventual consistency models, they may not always be fully ACID compliant.

THANK YOU