BDA_UNIT5

Introduction to NoSQL

Prepared by Mrs K H Vijaya Kumari, Asst. Professor, Dept of IT

Review of Relational Databases

- Relational databases are also called Relational Database Management Systems (RDBMS) or SQL databases.
- Most popular of these have been Microsoft SQL Server, Oracle Database, MySQL, and IBM DB2.
- The RDBMS's are used mostly in large enterprise scenarios, with the exception of MySQL, which is also used to store data for Web applications
- The relational data model provided a standard way of representing and querying data that could be used by any application.
- SQL provides an internally consistent mathematical language that makes it easier to improve the performance of all database queries

Review of Relational Databases

Four crucial properties define relational database transactions: atomicity, consistency, isolation, and durability—typically referred to as ACID.

- **Atomicity** defines all the elements that make up a complete database transaction.
- **Consistency** defines the rules for maintaining data points in a correct state after a transaction.
- **Isolation** keeps the effect of a transaction invisible to others until it is committed, to avoid confusion.
- **Durability** ensures that data changes become permanent once the transaction is committed.

Review of Relational Databases

Pros

- Relational databases work with structured data.
- They support ACID transactional consistency and support "joins."
- They come with built-in data integrity and a large eco-system.
- Relationships in this system have constraints.
- There is limitless indexing. Strong SQL.

Cons

- Relational Databases do not scale out horizontally very well (concurrency and data size), only vertically, (unless you use sharding).
- Data is normalized, meaning lots of joins, which affects speed.
- They have problems working with semi-structured data.

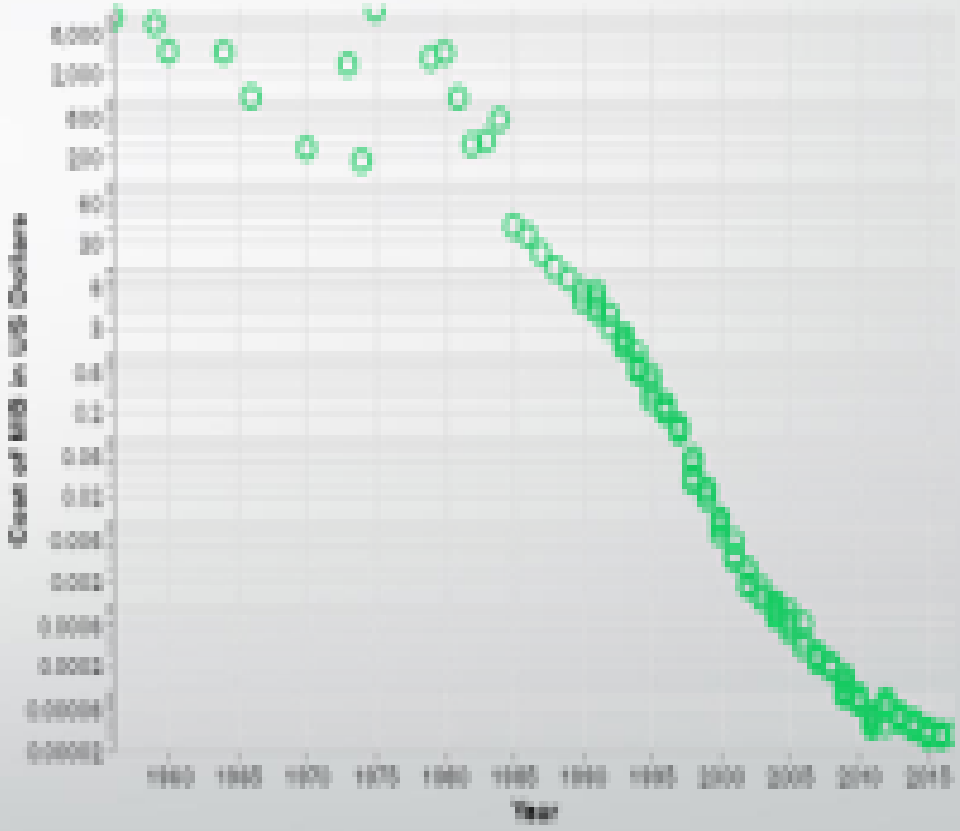
Non-Relational Databases

Non-Relational Databases are the ones that store the data in format other than relational tables.

Non-Relational Databases are generally referred to as NoSQL databases that indicate "Not Only SQL"

As shown in the figure, the cost of data rapidly decreased. Therefore the amount of data that applications needed to store and query increased. This data came in all shapes and sizes — structured, semi-structured, and polymorphic — and defining the schema in advance became nearly impossible.

NoSQL databases allow developers to store huge amounts of unstructured data, giving them a lot of flexibility.



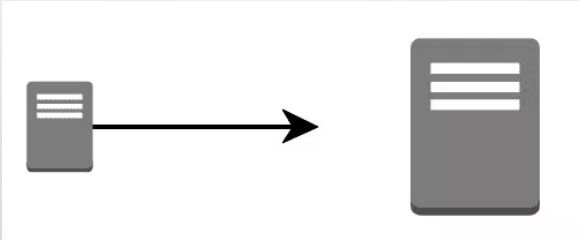
NoSQL Database Features

- Flexible schemas

Unlike SQL databases, where you must determine and declare a table's schema before inserting data. MongoDB's collections, by default, do not require their documents to have the same schema

- Horizontal scaling

Horizontal scaling, also known as scale-out, refers to bringing on additional nodes to share the load. This is difficult with relational databases due to the difficulty in spreading out related data across nodes.



Vertical scaling refers to increasing the processing power of a single server or cluster. Both relational and non-relational databases can scale up, but eventually, there will be a limit in terms of maximum processing power and throughput. Additionally, there are increased costs with scaling up to high-performing hardware, as costs do not scale linearly.

NoSQL Database Features

- Fast queries due to the data model

  These *denormalized* data models allow applications to retrieve and manipulate related data in a single database operation

- Ease of use for developers

  They allows you to immediately start building your application without spending time configuring a database.

Types of NoSQL databases

Over time, four major types of NoSQL databases emerged: document databases, key-value databases, wide-column stores, and graph databases.

- Document databases

  store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. The values can typically be a variety of types including things like strings, numbers, booleans, arrays, or objects.

- Key-value databases

  simpler type of database where each item contains keys and values.

- Wide-column stores

  store data in tables, rows, and dynamic columns.

- Graph databases

  store data in nodes and edges. Nodes typically store information about people, places, and things, while edges store information about the relationships between the nodes.

Difference between SQL and NoSQL

SQL	NoSQL
RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)	Non-relational or distributed database system.
These databases have fixed or static or predefined schema	They have dynamic schema
These databases are not suited for hierarchical data storage.	These databases are best suited for hierarchical data storage.
These databases are best suited for complex queries	These databases are not so good for complex queries
Vertically Scalable	Horizontally scalable
Follows ACID property	Follows CAP(consistency, availability, partition tolerance)

Difference between SQL and NoSQL

The Main Differences:

- **Type –**

  SQL databases are primarily called as Relational Databases (RDBMS);

  NoSQL database are primarily called as non-relational or distributed database.

- **Language –**

  SQL databases defines and manipulates data based structured query language (SQL). SQL is restrictive as it requires you to use predefined schemas to determine the structure of your data before you work with it. Also all of your data must follow the same structure. This can require significant up-front preparation which means that a change in the structure would be both difficult and disruptive to your whole system.

  A NoSQL database has dynamic schema for unstructured data. Data is stored in many ways. This flexibility means that documents can be created without having defined structure first. The syntax varies from database to database, and you can add fields as you go.

- **Scalability –**
  In almost all situations SQL databases are vertically scalable. This means that you can increase the load on a single server by increasing things like RAM, CPU or SSD. But on the other hand NoSQL databases are horizontally scalable. This means that you handle more traffic by sharding, or adding more servers in your NoSQL database. It is similar to adding more floors to the same building versus adding more buildings to the neighborhood. Thus NoSQL can ultimately become larger and more powerful, making these databases the preferred choice for large or ever-changing data sets.

Difference between SQL and NoSQL

- **Structure –**
  SQL databases are table-based on the other hand NoSQL databases are either key-value pairs, document-based, graph databases or wide-column stores. This makes relational SQL databases a better option for applications that require multi-row transactions such as an accounting system or for legacy systems that were built for a relational structure.

- **Property followed –**
  SQL databases follow ACID properties (Atomicity, Consistency, Isolation and Durability) whereas the NoSQL database follows the Brewers CAP theorem (Consistency, Availability and Partition tolerance).

- **Support –**
  Great support is available for all SQL database from their vendors. Also a lot of independent consultations are there who can help you with SQL database for a very large scale deployments but for some NoSQL database you still have to rely on community support and only limited outside experts are available for setting up and deploying your large scale NoSQL deployments.

Need for NoSQL

- The increasing business needs of the hour, require the scaling of existing databases. The availability of huge amounts of data with its diverse forms demands the storage and retrieval of the same in a flexible manner.
- As the existing SQL systems are well organized and can handle thousands of queries within no time. But this is applicable only to a small scale structured data.
- Although the NoSQL databases exists since 1960,it has been recently found that their **high operations speed and flexibility** in storing the data is suitable for present data era.

Need for NoSQL

5 important features of NoSQL databases that demonstrate when to use it…

1. Multi-Model

Relational databases store data in a **fixed and predefined structure**. It means when you start development you will have to define your data schema in terms of tables and columns. You have to change the schema every time the requirements change. This will lead to creating new columns, defining new relations, reflecting the changes in your application, discussing with your database administrators, etc.

NoSQL database provides much more flexibility when it comes to handling data. There is **no requirement to specify the schema** to start working with the application. Also, the NoSQL database doesn't put a restriction on the types of data you can store together. It allows you to add more new types as your needs change. These all are the reasons NoSQL is best suited for **agile development** which requires **fast implementation**. Developers and architects choose NoSQL to handle data easily for various kinds of agile development application requirements.

Need for NoSQL

2. Easily Scalable

The primary reason to choose a NoSQL database is easy scalability. Well, relational databases can also be scaled but not easily and at a lower cost. Relational databases are built on the concept of traditional master-slave architecture. Scaling up means upgrading your servers by adding more processors, RAM, and hard-disks to your machine to handle more load and increase capacity. You will have to divide the databases into smaller chunks across multiple hardware servers instead of a single large server. This is called sharding which is very complicated in relational databases. Replacing and upgrading your database server machines to accommodate more throughput results in downtime as well. These things become a headache for developers and architects.

NoSQL database built with a masterless, peer-to-peer architecture. Data is partitioned and balanced across multiple nodes in a cluster, and aggregate queries are distributed by default. This allows easy scaling in no time. Just executing a few commands will add the new server to the cluster. This scalability also improves performance, allowing for continuous availability and very high read/write speeds.

Need for NoSQL

3. Distributed

Relational databases use a centralized application that is location-dependent (e.g. single location), especially for write operations. On the other hand, the NoSQL database is designed to distribute data on a global scale. It uses multiple locations involving multiple data centers and/or cloud regions for write and read operations. This distributed database has a great advantage with masterclass architecture. You can maintain continuous availability because data is distributed with multiple copies where it needs to be.

4. **Redundancy and Zero Downtime**

What will happen if the hardware will fail? Well, NoSQL is also designed to handle these kinds of critical situations. Hardware failure is a serious concern while building an application. Rather than requiring developers, DBAs, and operations staff to build their redundant solutions, it can be addressed at the architectural level of the database in NoSQL. If we talk about Cassandra then it uses several heuristics to determine the likelihood of node failure. Riak follows the network partitioning approach (when one or more nodes in a cluster become isolated) and repair itself.

The masterclass architecture of the NoSQL database allows multiple copies of data to be maintained across different nodes. If one node goes down then another node will have a copy of the data for easy and fast access. This leads to zero downtime in the NoSQL database. When one considers the cost of downtime, this is a big deal.

Need for NoSQL

5. Big Data Applications

NoSQL can handle the massive amount of data very quickly and that's the reason it is best suited for the big data applications. NoSQL databases ensure data doesn't become the bottleneck when all of the other components of your server-side application are designed to be seamless and fast.

As we have mentioned about many features and advantages of using the NoSQL database but that doesn't mean that it fits in all kinds of applications. There are some downsides of NoSQL as well so choosing the right database depends on your type of application or nature of data. Some projects are better suited to using an SQL database, while others work well with NoSQL. Some could use both interchangeably.

If you are storing transactional data then SQL is your friend. Relational databases were created to process transactions and they are very good with that. If your data is analytical then think about NoSQL, SQL was never intended for data analytics and with large amounts of data that can become an issue.

Columnar Database

While a relational database is optimized for storing rows of data, typically for transactional applications, a columnar database is optimized for fast retrieval of columns of data, typically in analytical applications.

Columnar storage for database tables is an important factor in optimizing analytic query performance because it drastically reduces the overall disk I/O requirements and reduces the amount of data you need to load from disk.

In a typical relational database table, each row contains field values for a single record. In row-wise database storage, data blocks store values sequentially for each consecutive column making up the entire row. If block size is smaller than the size of a record, storage for an entire record may take more than one block. If block size is larger than the size of a record, storage for an entire record may take less than one block, resulting in an inefficient use of disk space. In online transaction processing (OLTP) applications, most transactions involve frequently reading and writing all of the values for entire records, typically one record or a small number of records at a time. As a result, row-wise storage is optimal for OLTP databases.

Columnar Database

Using columnar storage, each data block holds column field values for as many as three times as many records as row-based storage. This means that reading the same number of column field values for the same number of records requires a third of the I/O operations compared to row-wise storage. In practice, using tables with very large numbers of columns and very large row counts, storage efficiency is even greater.

An added advantage is that, since each block holds the same type of data, block data can use a compression scheme selected specifically for the column data type, further reducing disk space and I/O. For more information about compression encodings based on data types, see Compression encodings.

The savings in space for storing data on disk also carries over to retrieving and then storing that data in memory. Since many database operations only need to access or operate on one or a small number of columns at a time, you can save memory space by only retrieving blocks for columns you actually need for a query. Where OLTP transactions typically involve most or all of the columns in a row for a small number of records, data warehouse queries commonly read only a few columns for a very large number of rows. This means that reading the same number of column field values for the same number of rows requires a fraction of the I/O operations and uses a fraction of the memory that would be required for processing row-wise blocks. In practice, using tables with very large numbers of columns and very large row counts, the efficiency gains are proportionally greater. For example, suppose a table contains 100 columns. A query that uses five columns will only need to read about five percent of the data contained in the table. This savings is repeated for possibly billions or even trillions of records for large databases. In contrast, a row-wise database would read the blocks that contain the 95 unneeded columns as well.

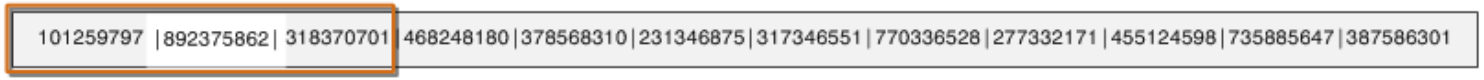
Columnar Database

Row-based block Storage

SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL

Columnar block Storage

SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL



CAP Theorem

- Consistency

Consistency means that all nodes in the network see the same data at the same time.

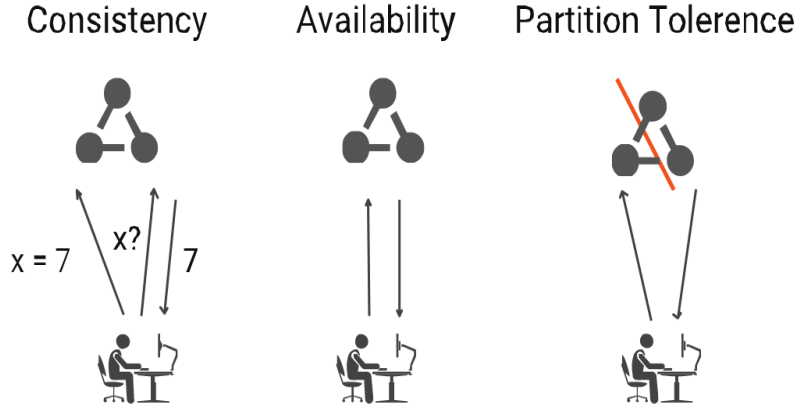
- Availability

Availability is a guarantee that every request receives a response about whether it was successful or failed.

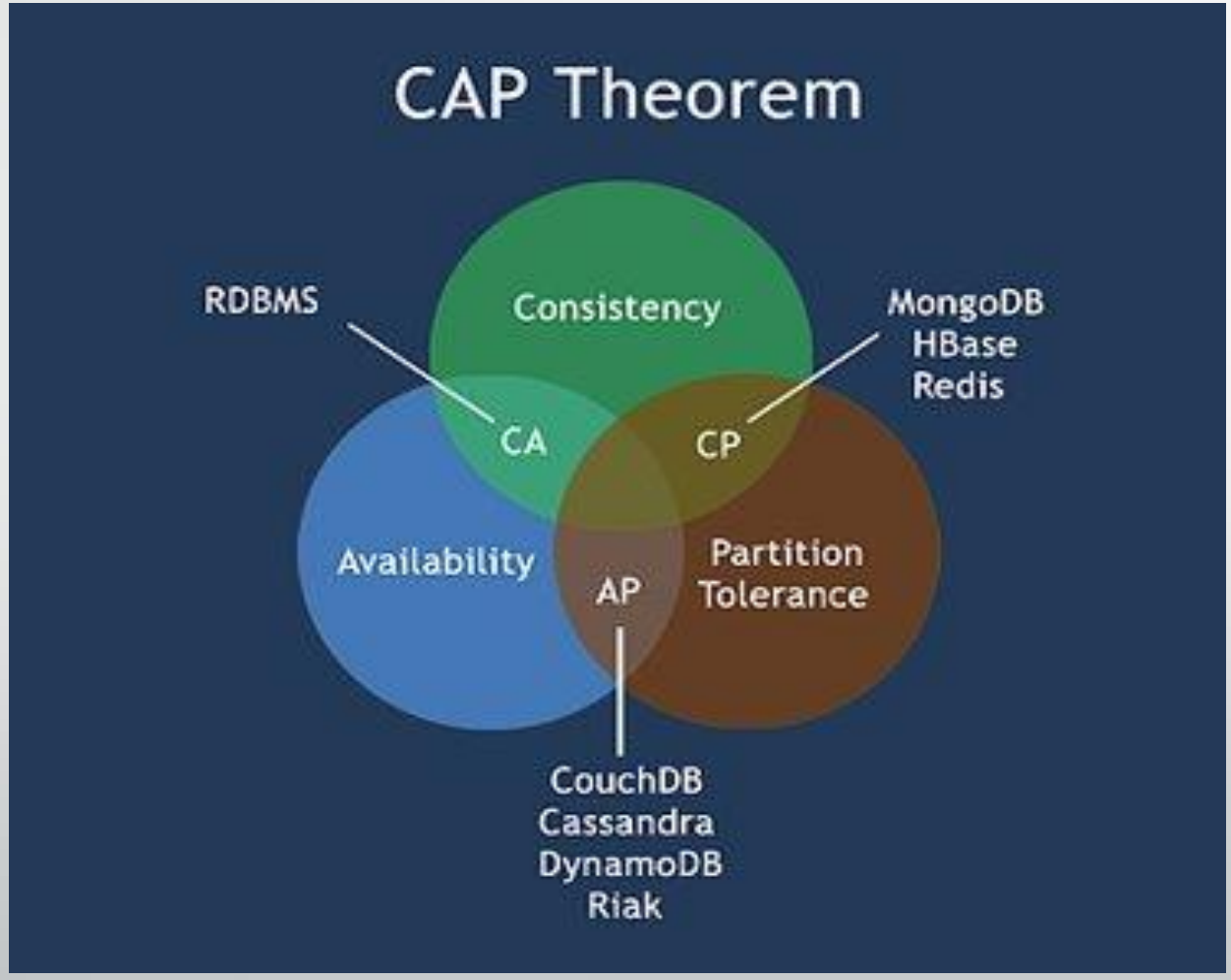
- Partition Tolerance

guarantees that the system continues to operate despite arbitrary message loss or failure of part of the system.

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


CAP Theorem
RDBMS
Consistency
MongoDB
HBase
Redis
CA
CP
Availability
AP
Partition
Tolerance
CouchDB
Cassandra
DynamoDB
Riak

Tradeoff Scenarios of CAP Theorem

Consistency vs High Availability

In NoSQL systems Partition Tolerance is inevitable. Therefore we must be prepared to handle it by default.

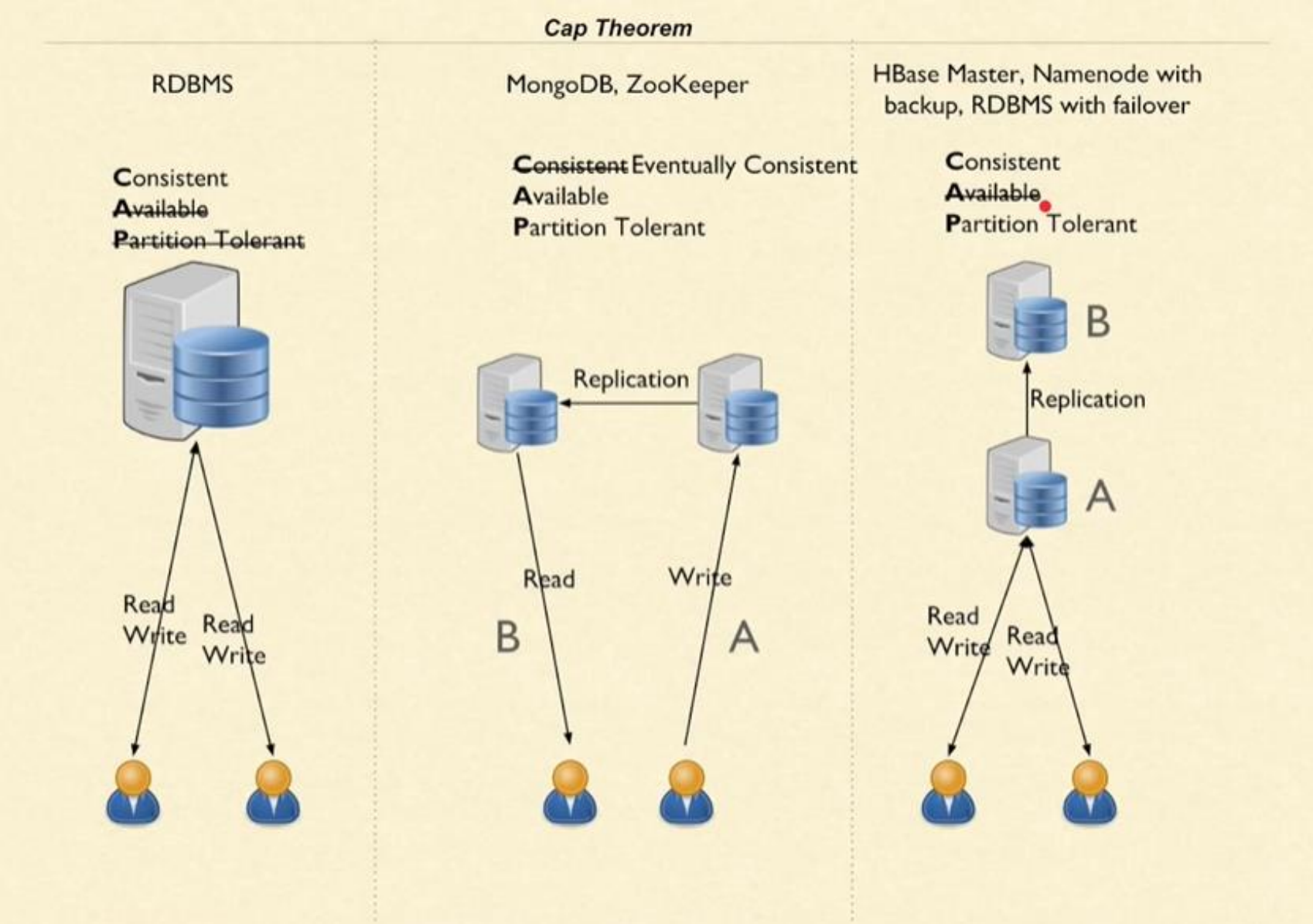
Compromising High Availability

The first one is RDBMs where reading and writing of data happens on the same machine. Such systems are consistent but not partition tolerant because if this machine goes down, there is no backup. Also, if one user is modifying the record, others would have to wait thus compromising the high availability.

We can have a Server A with backup B . Every new change or modification at A is propagated to the backup machine B. There is only one machine which is interacting with the readers and writers. So, It is consistent but not highly available.

Compromising Consistency

Only one machine can accept modifications while the reads can be done from all machines. In such systems, the modifications flow from that one machine to the rest. Such systems are highly available as there are multiple machines to serve. Also, such systems are partition tolerant because if one machine goes down, there are other machines available to take up that responsibility. Since it takes time for the data to reach other machines from the node A, the other machine would be serving older data. This causes inconsistency. Though the data is eventually going to reach all machine and after a while, things are going to okay. There we call such systems **eventually consistent** instead of strongly consistent. This kind of architecture is found in Zookeeper and MongoDB.


Cap Theorem
RDBMS
Consistent
Available
Partition Tolerant
Read
Write
Read
Write
MongoDB, ZooKeeper
Consistent Eventually Consistent
Available
Partition Tolerant
Replication
Read
Write
B
A
HBase Master, Namenode with backup, RDBMS with failover
Consistent
Available
Partition Tolerant
B
Replication
A
Read
Write
Read
Write