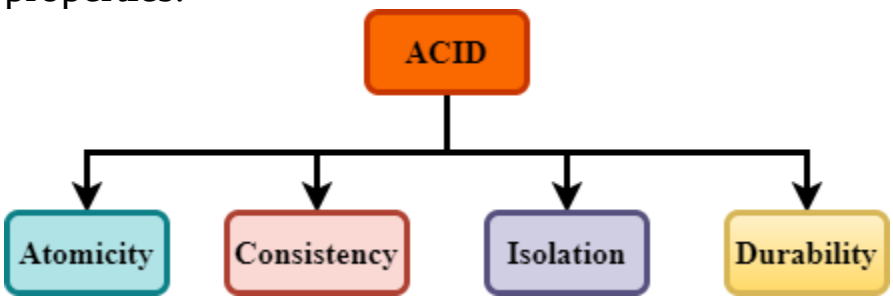
NoSQL DB

- **Overview**
  - It is a non-relational database solution for Big Data in a more efficient way.
  - RDBMS were not designed to be distributed
  - It is originally developed by Google and Amazon

- **Traditional RDBMSs**
  - Data stored in columns and tables
  - Data Manipulation Language: Select, Insert, Update, & Delete statements
  - Functions and Procedures
  - Explicit transaction control
  - Schema defined at the start
  - Triggers to respond to Insert, Update , & Delete
  - Security and Access Control
  - Transactions
  - Create indexes to support queries
  - In Memory databases
  - Vertical Scalability (or up)
    - Can be achieved by hardware upgrades (e.g., faster CPU, more memory, or larger disk)
  - **ACID properties:**
    - For every change you make, you should ensure strict ACID properties.



    - The properties are:
      - **Atomicity**:
        - An "all or nothing" rule.

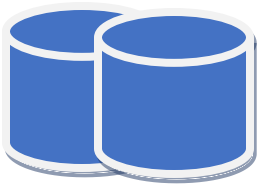
        - If a transaction fails, the entire transaction is rolled back.
    - **Consistency**:
        - A transaction is successful only once it is completely executed and saved any failure.
        - No half-completed transactions.
    - **Isolation**:
        - Multiple transactions can occur simultaneously without interfering with each other.
        - Each transaction is independent.
    - **Durability**:
        - The results of a committed transaction survive failures.
        - Ensures that once a transaction is successful (committed) to the database, it is saved in the transaction logs.
        - Changes made to the system by successful transactions are durable - saved on disks.
- Example:
    - Given a checking account A with a balance of $1000 and a saving account B with a balance of $500.
    - We want to transfer $300 from A to B.

Atomicity	
Successful Transaction	Failed Transaction
Before: A=$1000 and B=500 ➔ Total=$1500 **Step1:** Read balance(A) **Step2:** Write Balance(A) - $300 **Step3:** Read Balance(B) **Step4:** Write Balance(B)+$300 **After: A=$700 and B=$800 ➔ Total=$1500**	**Before: A=$1000 and B=500 ➔ Total=$1500** **Step1:** Read balance(A) **Step2:** Write Balance(A) - $300 **Step3:** Read Balance(B) **Failure ➔ Rollback at the beginning** ~~**Step4:** Write Balance(B)+$300~~ **After: A=$1000 and B=$500 ➔ Total=$1500**

Consistency	
Successful Transaction	Failed Transaction
Before: A=$1000 and B=500 **➔ Total=$1500** **Step1:** Read balance(A) **Step2:** Write Balance(A) - $300 **Step3:** Read Balance(B) **Step4:** Write Balance(B)+$300 **After: A=$700 and B=$800** **➔ Total=$1500**	**Before: A=$1000 and B=500 ➔ Total=$1500** **Step1:** Read balance(A) **Step2:** Write Balance(A) - $300 **Failure:** **A=$1200 and B=$500** **Data is inconsistent** **Tables should be locked.** ~~**Step3:** Read Balance(B)~~ ~~**Step4:** Write Balance(B)+$300~~

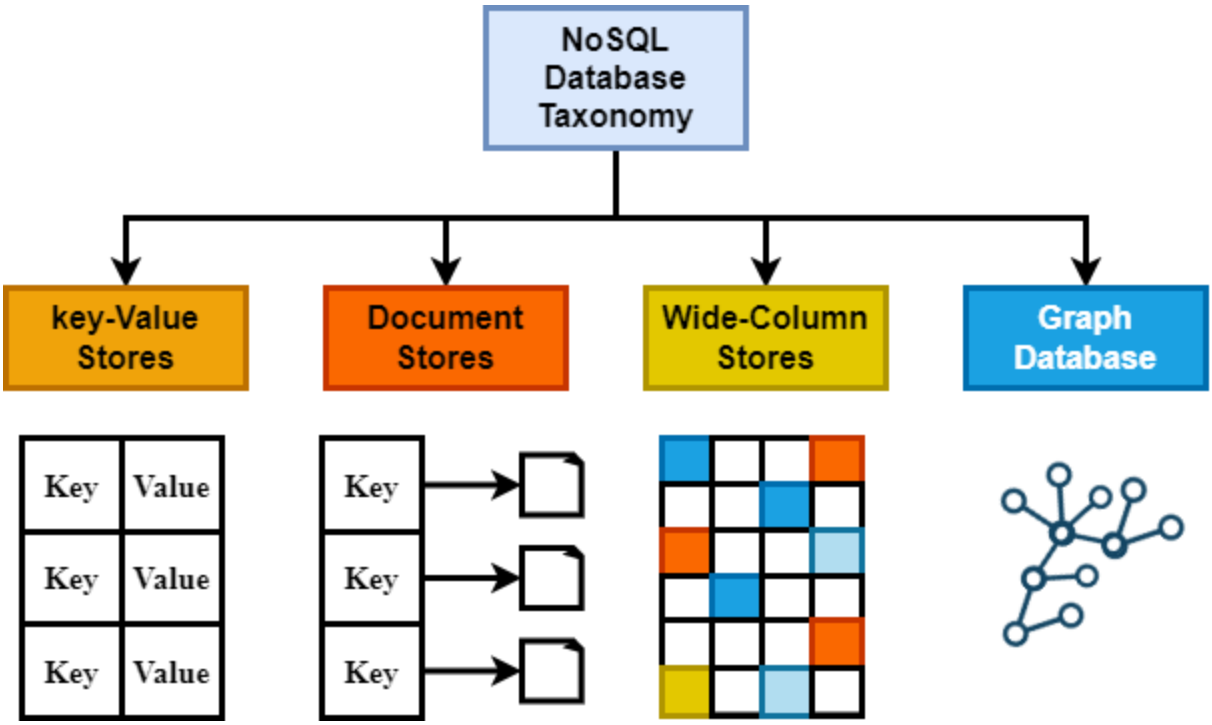
Isolation **Two concurrent transactions: T1 and T2** **Assume: A=$1000 and B=300 ➔ Total=$1300**	
T1: First	T2: Second
Step1: Read balance(A) **Step2:** Write Balance(A) - $100 **Step3:** Read Balance(B) **Step4:** Write Balance(B)+$100	**Step1:** Read balance(B) **Step2:** Write Balance(B) - $200 **Step3:** Read Balance(A) **Step4:** Write Balance(A)+$200
Assume T1 then T2	**Assume T2 then T1**
T1: First Before: A=$1000 and B=300 ➔ Total=$1300 After: A=$900 and B=$400 ➔ Total=$1300	**T2: First** Before: A=$1000 and B=300 ➔ Total=$1300 After: A=$1200 and B=$100 ➔ Total=$1300
T2: Second Before: A=$900 and B=$400 ➔ Total=$1300 **Final: A=$1100 and B=$200**	**T1: Second** Before: A=$1200 and B=$100 ➔ Total=$1300 **Final: A=$1100 and B=$200**

➔ **Total=$1300**	➔ **Total=$1300**

Durability
Updates and modifications to the database are stored in and written to disk and they persist even if system failure occurs 

- **NoSQL Database**
  - NoSQL stands for:
    - No Relational
    - No RDBMS
    - Not Only SQL
  - NoSQL is a database solution for all databases that don't follow the RDBMS principles.
  - Does not give importance to ACID properties
  - Unlike RDBMS, NoSQL is Schema-less (not a well-defined schema)
  - It is for big data applications that require mostly queries and few updates.
  - Definition:
    - Wikipedia: "A NoSQL database provides a mechanism for storage and retrieval of data that use looser consistency models than traditional relational databases in order to achieve horizontal scaling and higher availability. Some authors refer to them as "Not only SQL" to emphasize that some NoSQL systems do allow SQL-like query language to be used."
  - NoSQL solutions are designed to run on clusters or multi-node database solutions:
    - Horizontal scalability (or out):
      - Can be achieved by adding more machines
      - Requires database sharding and probably replication

- **NoSQL Database Taxonomy**
  - There are a variety of NoSQL databases:
    - key-value stores
    - Document databases
    - Column-family (aka big-table) stores
    - graph databases



- **key-value stores**
  - They are the simplest NoSQL databases.
  - Every single item in the database is stored as an attribute name (or 'key'), together with its value.
  - A design concept that exists in several programming languages: an array or dictionaries in Python, etc.
  - A value, which can be basically any piece of data or information, is stored with a key that identifies its location.
  - Key-value database schema:
    - Table name:primary_key_value:attribute_name:value
  - Example: Customer table

custID	Lname	Fname	Email
C01	May	Kate	mk@myemail.com

C02	John	Paul	jo@myemail.com
C03	Omar	Anbar	oa@myemail.com

The key-value database schema is:

Customer:C01:Lname:”Mary”
Customer:C01:Fname:”Kate”
Customer:C01:Email:”mk@myemail.com”

Customer:C02:Lname:”John”
Customer:C02:Fname:”Paul”
Customer:C02:Email:” mk@myemail.com”

Customer:C03:Lname:”Omar”
Customer:C03:Fname:”Anbar”
Customer:C03:Email:” jo@myemail.com”

- Database examples:
  - Amazon DynamoDB
  - Couchbase

- **Document databases:**
  - Pair each key with a complex data structure known as a document.
  - The main element is a "document“ that corresponds to a row in RDBMS.
  - Documents are stored in some standard formats like JSON (BSON).
  - Documents are uniquely identified via a unique key.
  - The database offers an API or query language that retrieves documents based on their contents.
  - It is a simple, powerful, and scalable data model
  - Documents are schema free:

- Different documents can have structures and schema that differ from one another. (An RDBMS requires that each row contain the same columns.)

- Example: Customer Table

custID	Lname	Fname	Email
C01	May	Kate	mk@myemail.com
C02	John	Paul	jo@myemail.com
C03	Omar	Anbar	oa@myemail.com

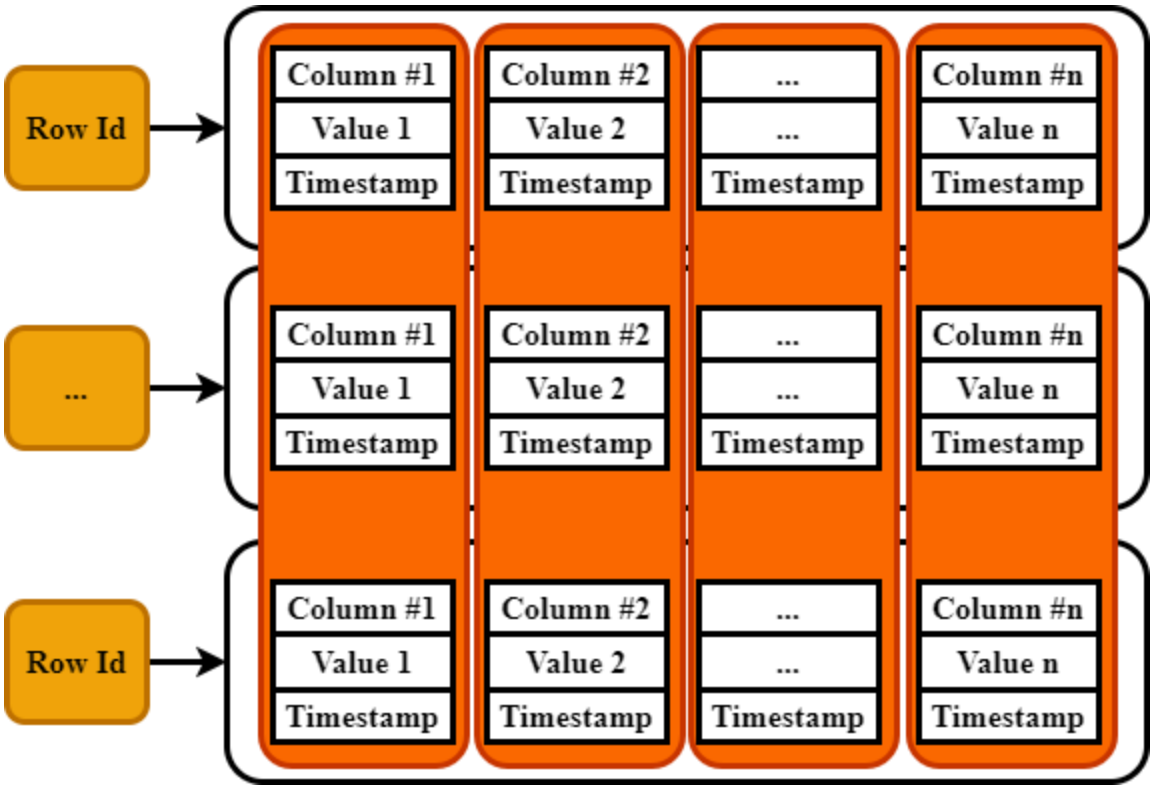
The document-oriented database schema is:

```
{ Id: C01,
  Lname: “May”,
  Fname: “Kate”,
  Email: “mk@myemail.com”
}

{ Id: C02,
  Lname: “John”,
  Fname: “Paul”,
  Email: “jp@myemail.com”
}
{ Id: C03,
  Lname: “Omar”,
  Fname: “Anbar”,
  Email: “oa@myemail.com”
}
```

- Database Example:
  - MongoDB (used in FourSquare, Github, and more)
  - CouchDB (used in Apple, BBC, Canonical, Cern, and more)

- **Column Store Database**
  - Store Optimized for queries over large datasets, and store columns of data together, instead of rows
  - Columnar databases focus on the efficiency of **read operations**: Data is organized in groups of columns (i.e., column families).
  - Compression algorithms work best on columns.
  - Column store database's structure:
    - It stores data in **column families:**
      - They are like table in an RDBMS
      - They contain rows and each row contains columns associated with a row key.
        - column family ➔ table
        - column family row ➔ table row
        - column family column ➔ table column:
          - Each column has three fields:
            - Column Key which the name of the column
            - Column value (cell)
            - Timestamp: time and date the data was inserted.
    - Each column can be thought of as a set of fields in a relational database.
    - No need to store NULL values: rows may not have the same columns.
    - Columns can be added to any row at any time without having to add it to other rows.



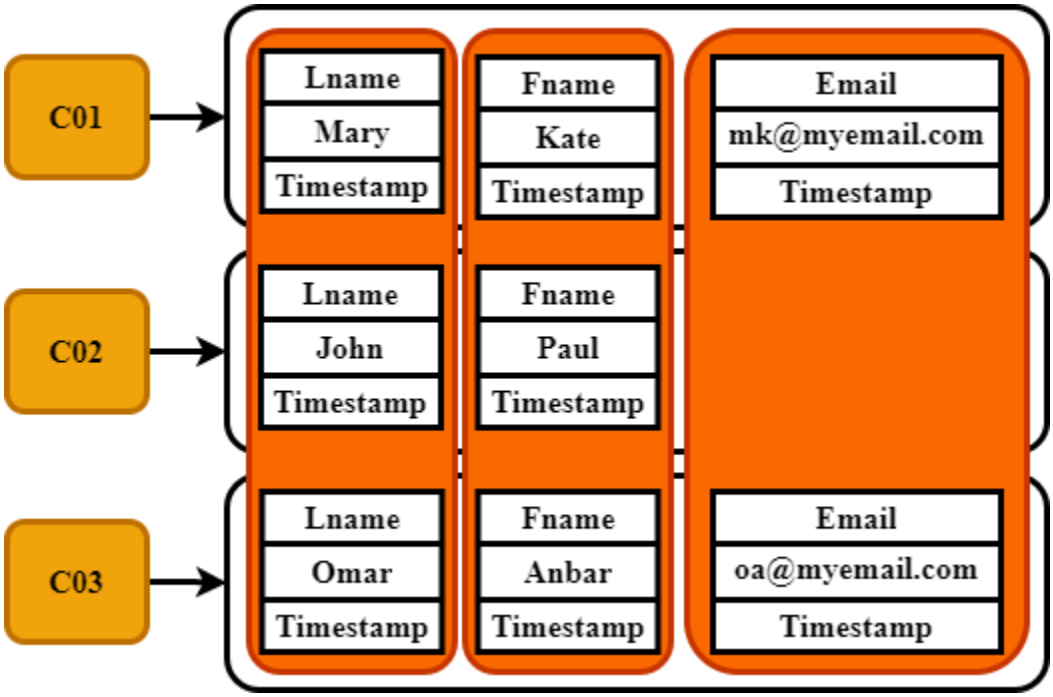
-
- **Drawback**:
  - Slow to add new data or update old data.
- Example:

custID	Lname	Fname	Email
C01	May	Kate	mk@myemail.com
C02	John	Paul	
C03	Omar	Anbar	oa@myemail.com

In row-oriented database, the data is stored as follows:

C01	May	Kate	mk@myemail.com	C02	John	Paul
NULL	C03	Omar	Anbar	oa@myemail.com		

In column-oriented database, the data is stored as follows:



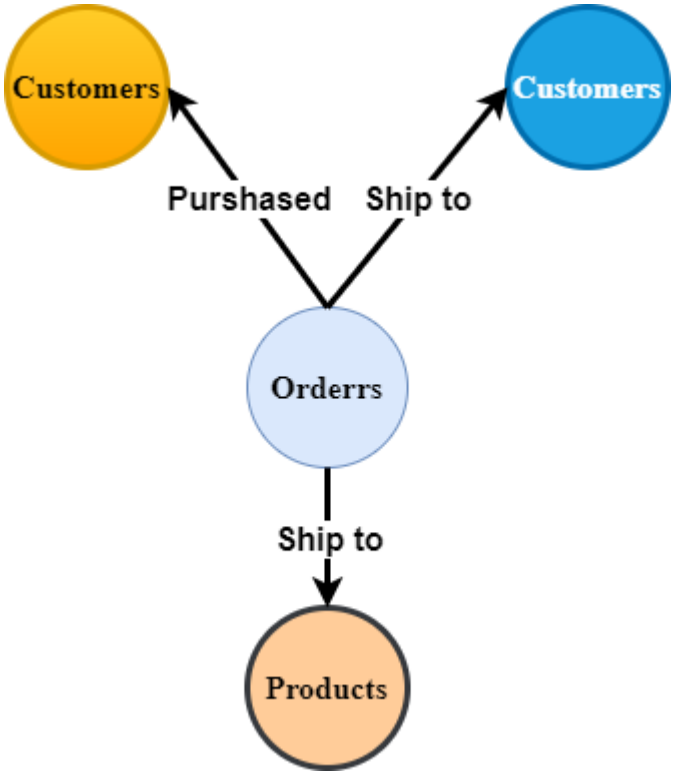
- Database Examples:
  - BigTable
  - Cassandra
  - HBase

- **Graph Databases :**
  - Data is organized in the form of a graph to store entities and their relationships.
  - A node represents an entity, and an edge represents the relationship between entities.
  - Scalability may be problematic.

custID	Lname	Fname	Email	address	City	State	zip
C01	May	Kate	mk@myemail.com	123 M street	Fairfax	VA	22033
C02	John	Paul	jo@myemail.com	123 K street	Vienna	VA	22181
C03	Omar	Anbar	oa@myemail.com	123 N street	Vienna	VA	22180

proId	Name	price
P001	Phone	$300
P002	Tv	$2000
P003	Laptop	$500

Trandid	Ordered	custid	Prodid	Qty
100	200	C01	P001	1
101	201	C02	P002	2
102	202	C01	P003	2
103	203	C01	P002	1

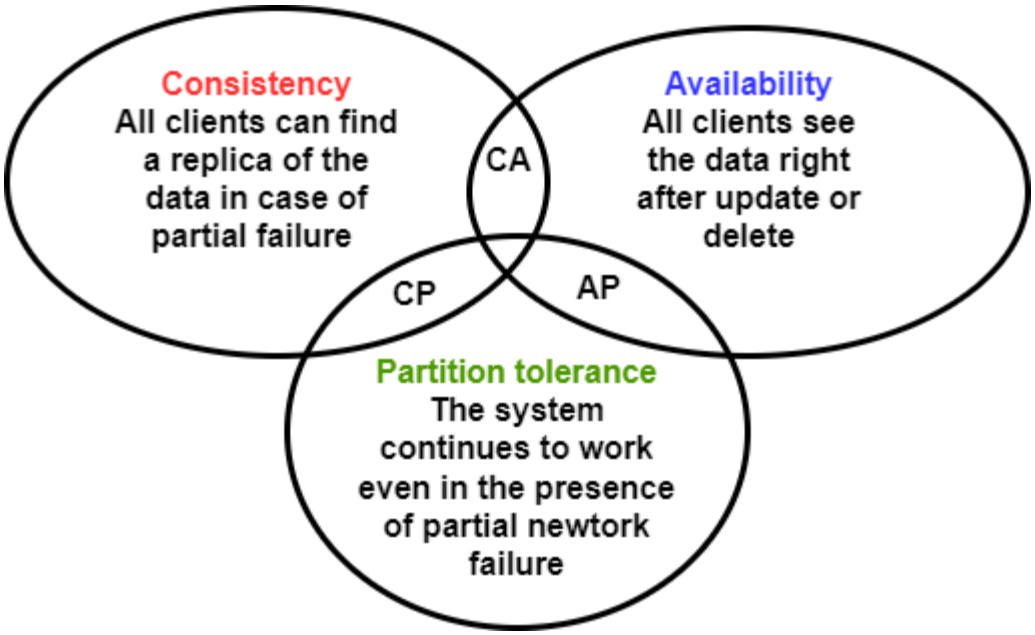


- Database examples:
  - Neo4JS
  - OrientDB
  - ArangoDB

- **Large databases Scalability: CAP Theorem & Base Properties**

  - Large NoSQL databases come with certain limitations.
  - Horizontal scalability may lead to the possibility of node failure.
  - Downtime for large companies such as Google means lost revenue.
  - For these companies to guarantee availability and scalability, they must scarify "strict" Consistency.

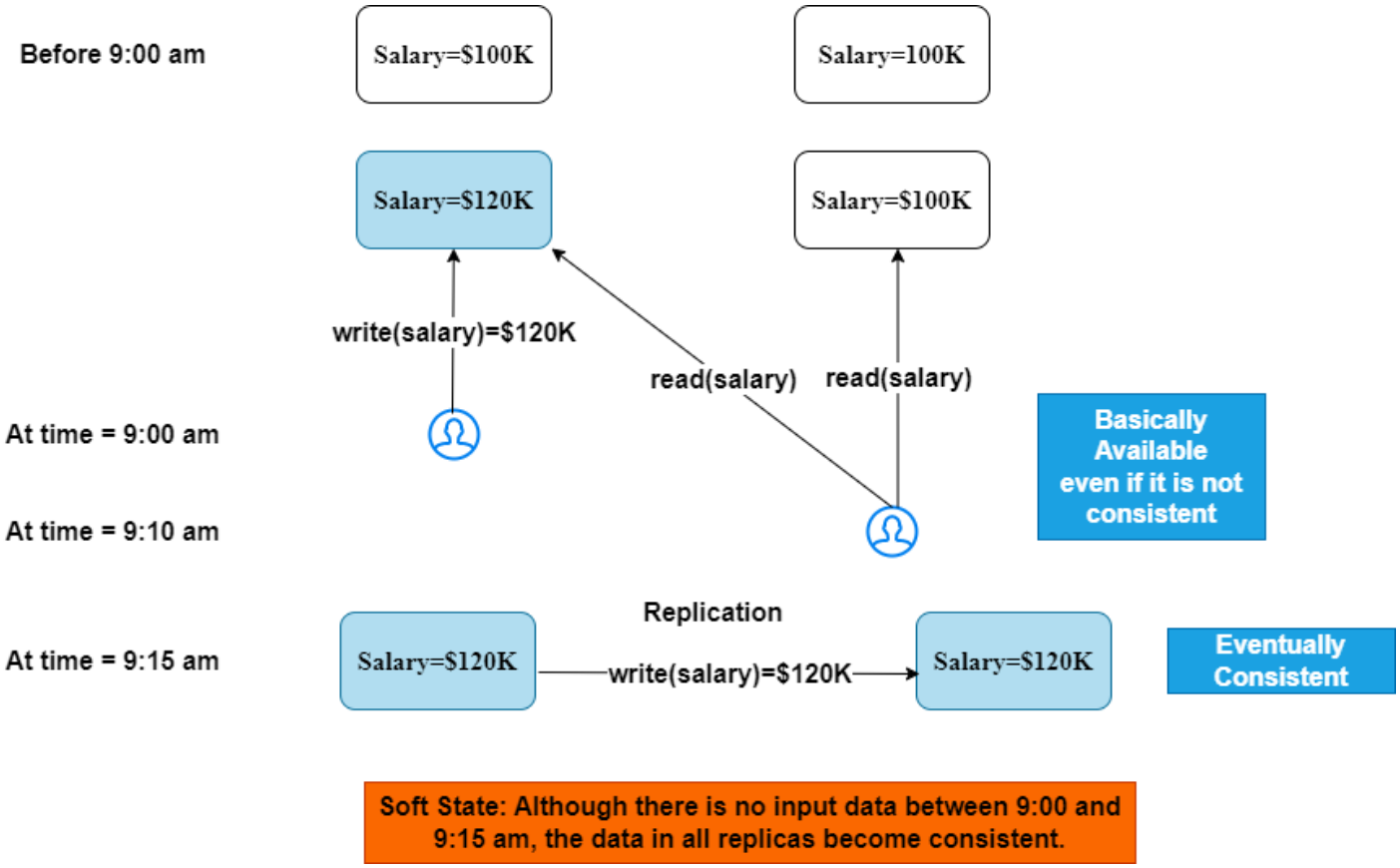
  - **CAP Theorem**
    - NoSQL cannot provide consistency and high availability at the same time.
    - CAP theorem describes the limitations of distributed databases.
    - CAP theorem or Eric Brewers theorem states that we can only achieve at most two of the following:
      - Consistency,
      - Availability, and
      - Partition Tolerance.



    - Consistency:
      - It guarantees that all nodes are up to date and see the same data.

    - A read will not be successful if not all servers are not updated and
- Availability:
    - The system continues to operate, even in case of failure.
    - Every request receives a response about whether it was successful or failed.
    - It does not guarantee that a read request returns the most recent write.
- Partition Tolerance:
    - The database continues to work despite system failure.
- Is it possible to have Availability, Consistency, and Partition Tolerance at the same time:
    - MongoDB and HBase: C & P
    - Cassandra: A & P

- **Base Properties**
    - NoSQL relies upon a softer model known as the BASE model.
    - If ACID properties belong to RDBMSs, BASE properties belong to NoSQL databases.
    - **Basically, Available**:
        - The system works all the time (available in case of failure).
        - Any request will receive an answer even in changing state
    - **Soft State**:
        - The state of the system will change over time without any update.
    - **Eventually Consistent**:
        - Not immediately consistent, but with time data will be consistent.
        - There is no guarantee about when this will occur.



- **Strengths and weaknesses of NoSQL**
  - Pros:
    - NoSQL databases are highly scalable.
    - They can be scaled horizontally as opposed to vertically-an advantage over SQL databases.
    - The right solution for big data applications
    - Less Code: Many NoSQL databases require only a few lines of code.
    - High performance: millions of transactions per second.
    - Flexibility to change data types at any time.
    - Availability and redundancy
    - Ease of use via APIs
  - Drawbacks:

- NoSQL databases is that they don't support ACID (atomicity, consistency, isolation, durability) transactions across multiple documents
- No universal query language and no joins.
- Data inconsistency
- Lack of data integrity