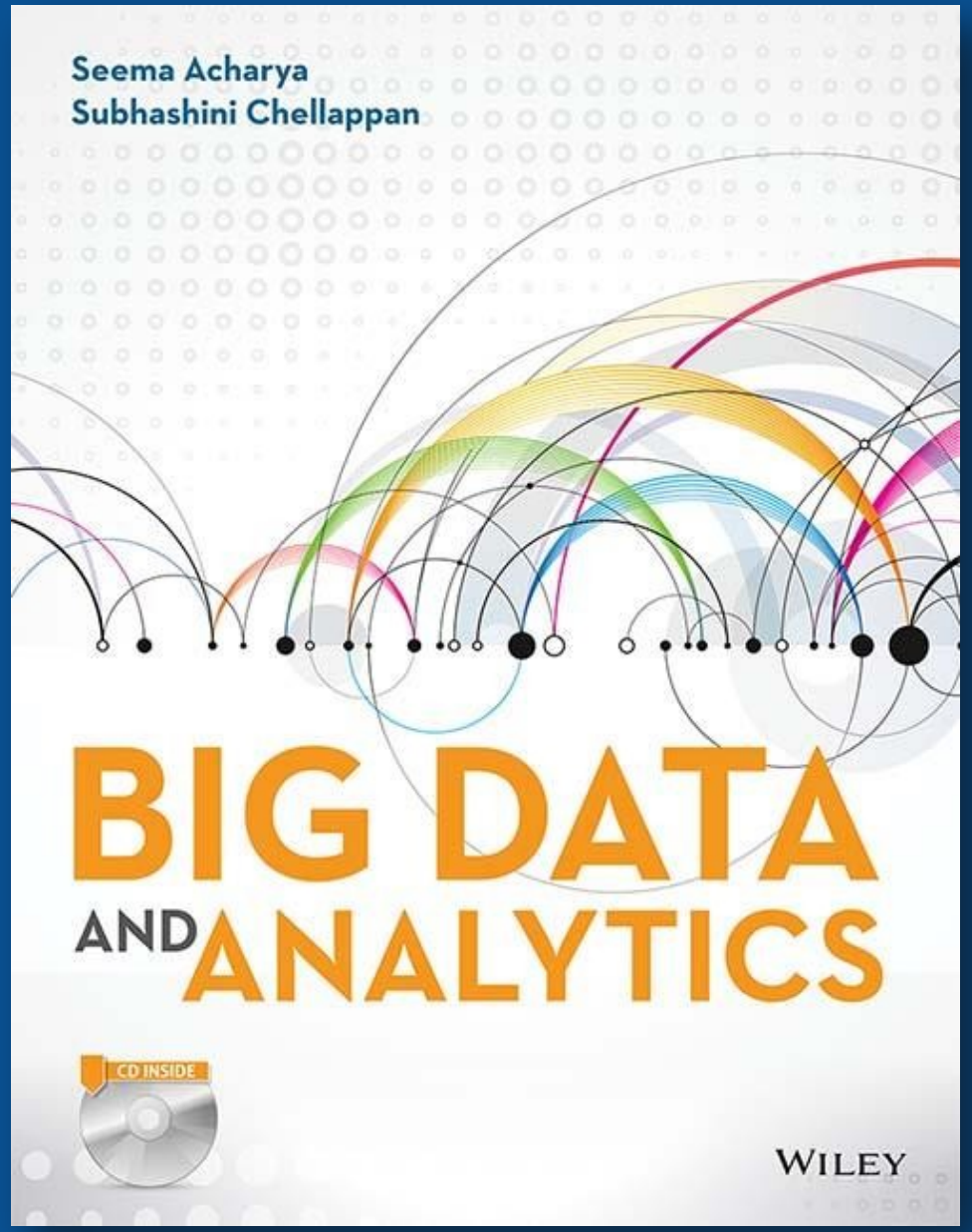


Big Data And Analytics

Seema Acharya
Subhashini Chellappan

Chapter 6

Introduction to MongoDB

Learning Objectives and Learning Outcomes

Learning Objectives	Learning Outcomes
Introduction to MongoDB 1. To study the features of MongoDB. 2. To learn how to perform CRUD operations. 3. To study aggregation. 4. To study the MapReduce Framework. 5. To import from and export to CSV format.	a) To comprehend the reasons behind the popularity of NoSQL database. b) To be able to perform CRUD operations. c) To comprehend MapReduce framework. d) To understand the aggregation. e) To be able to successfully import from and export to CSV.

Session Plan

Lecture time	90 to 120 minutes
Q/A	15 minutes

Agenda

- NoSQL: Types of NoSQL databases
- CAP Theorem
- Terms used in RDBMS and MongoDB
- MongoDB Query Language: CRUD
- Finding documents based on search criteria
- Dealing with Null values
- Count, Limit, Sort Skip, Aggregate Functions

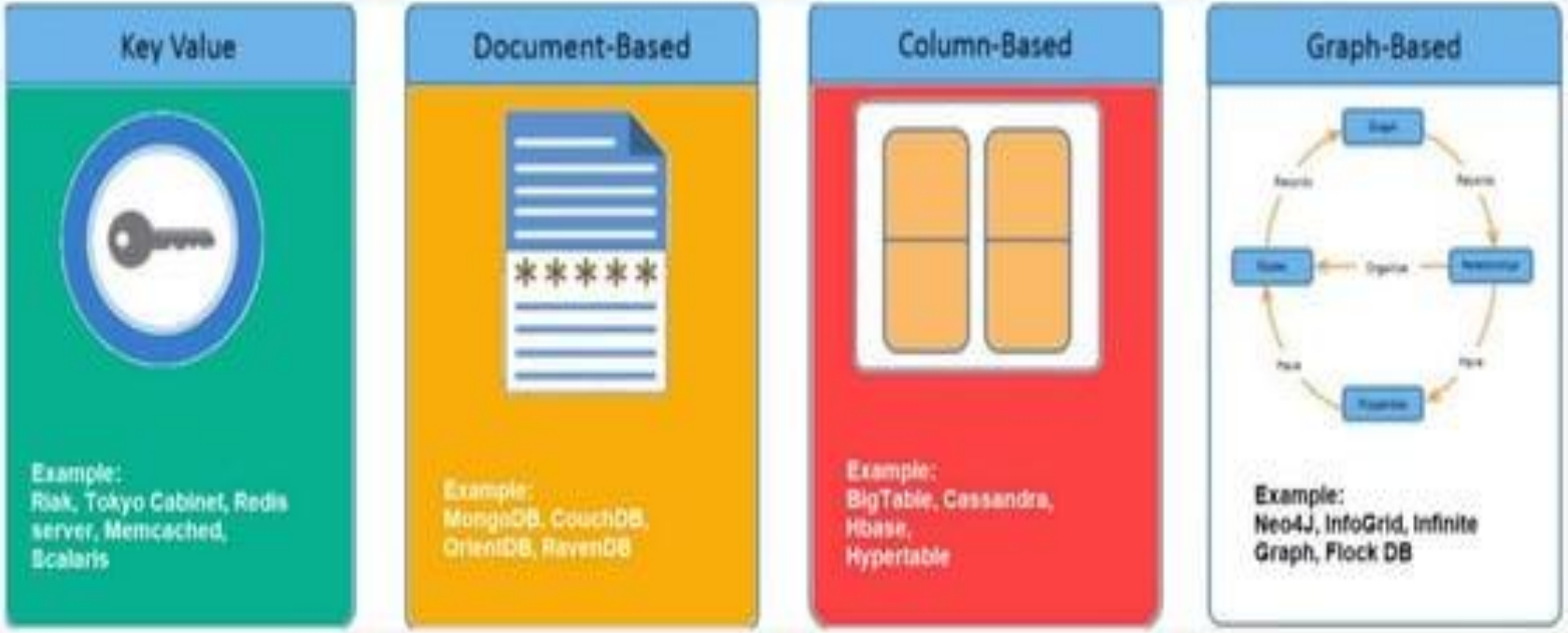
MongoDB- An Introduction

What is NoSQL Database?

- The term "NoSQL database" refer to any "Non-relational" or "Not only SQL" databases provides a mechanism for storage and retrieval of data in a format other than tabular relations model used in relational databases.
- NoSQL database doesn't use tables for storing data.
- It is generally used to store big data and real-time web applications.
- Flexible schema i.e., no predefined or rigid schema.
- It avoids joins and is easy to scale.

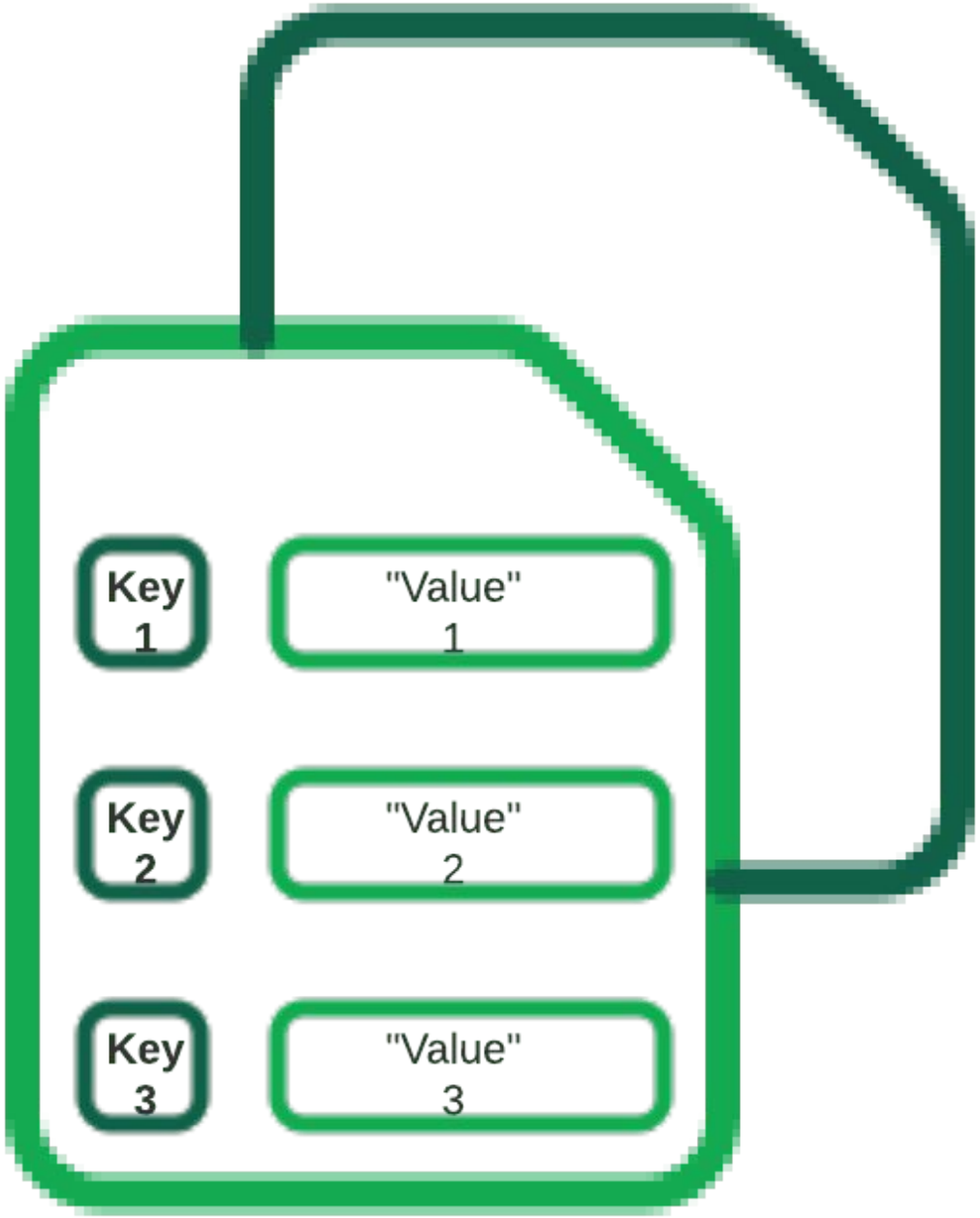
Types of NoSQL Database

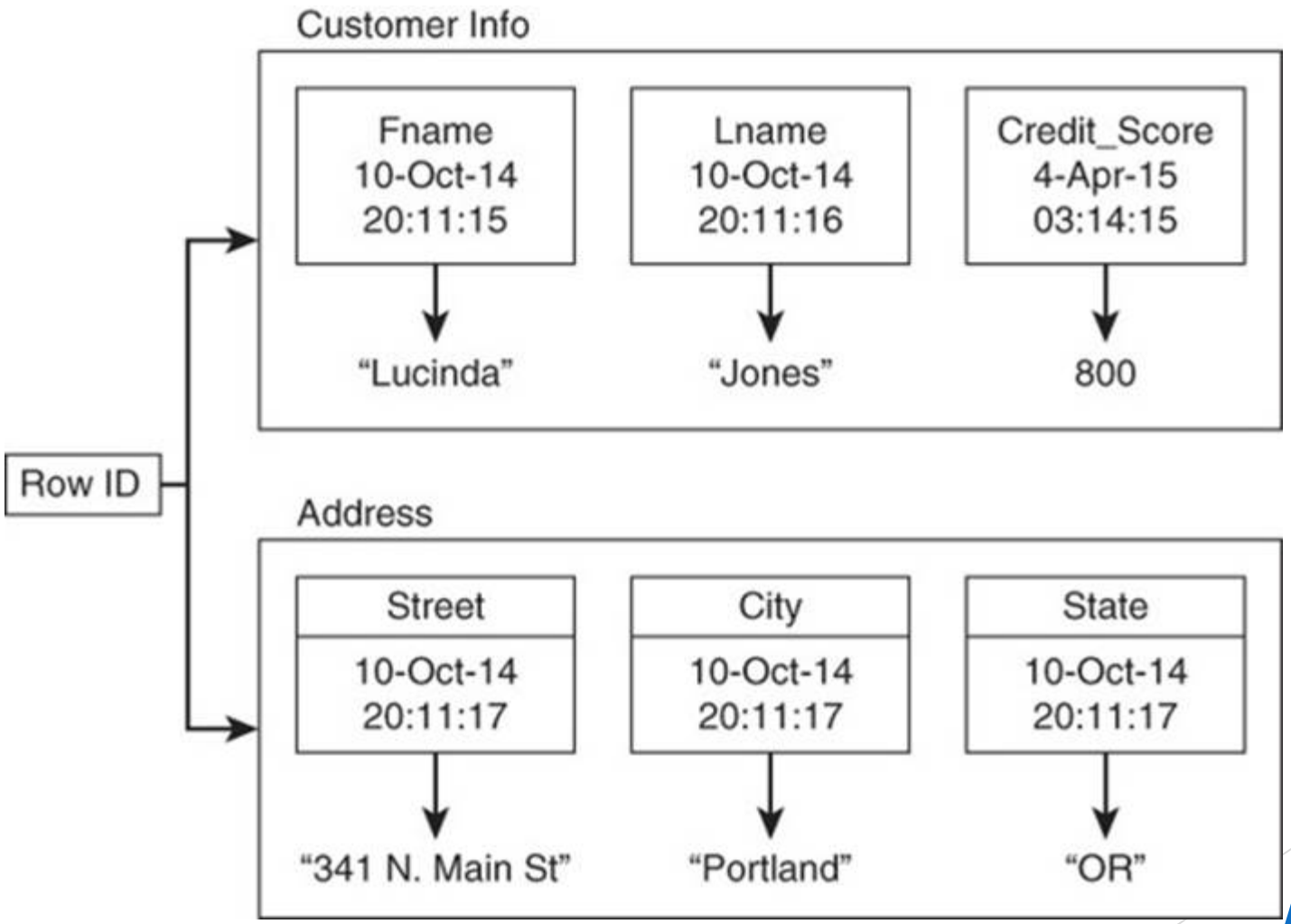
NoSQL Databases are mainly categorized into four types: Key-value pair, Column-oriented, Graph-based & Document-oriented.



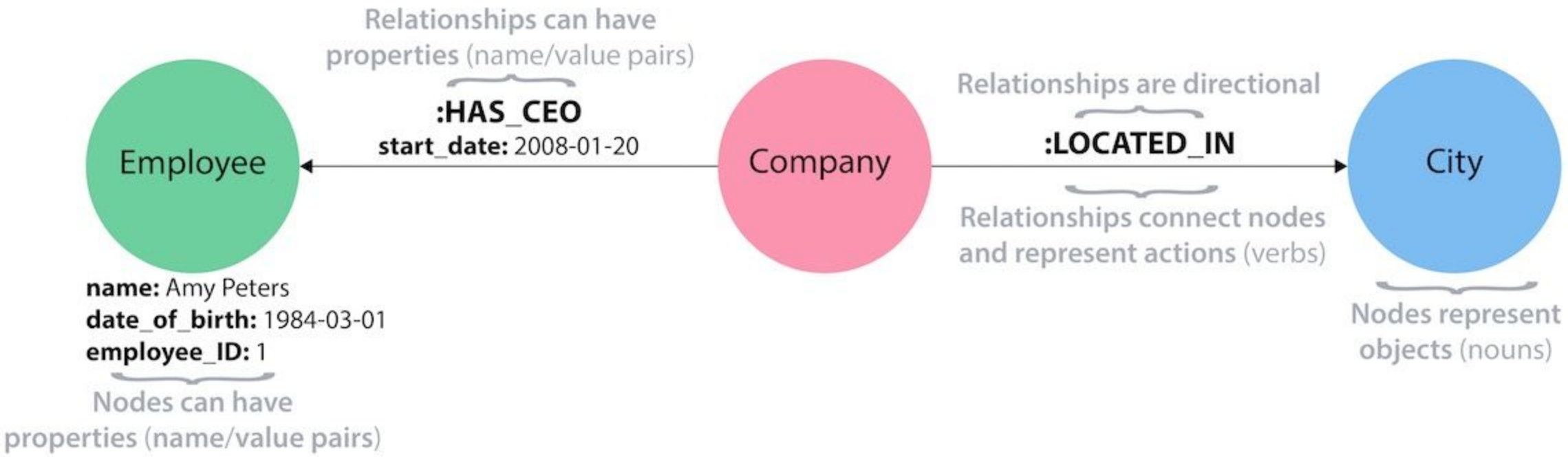
Types of NoSQL

Key value data store	Column-oriented data store	Document data store	Graph data store
• Riak • Redis • Membase	• Cassandra • HBase • HyperTable	• MongoDB • CouchDB • RavenDB	• InfiniteGraph • Neo4 • Allegro Graph


Key 1
"Value" 1
Key 2
"Value" 2
Key 3
"Value" 3


Customer Info
Fname
10-Oct-14
20:11:15
Lname
10-Oct-14
20:11:16
Credit_Score
4-Apr-15
03:14:15
"Lucinda"
"Jones"
800
Row ID
Address
Street
10-Oct-14
20:11:17
City
10-Oct-14
20:11:17
State
10-Oct-14
20:11:17
"341 N. Main St"
"Portland"
"OR"

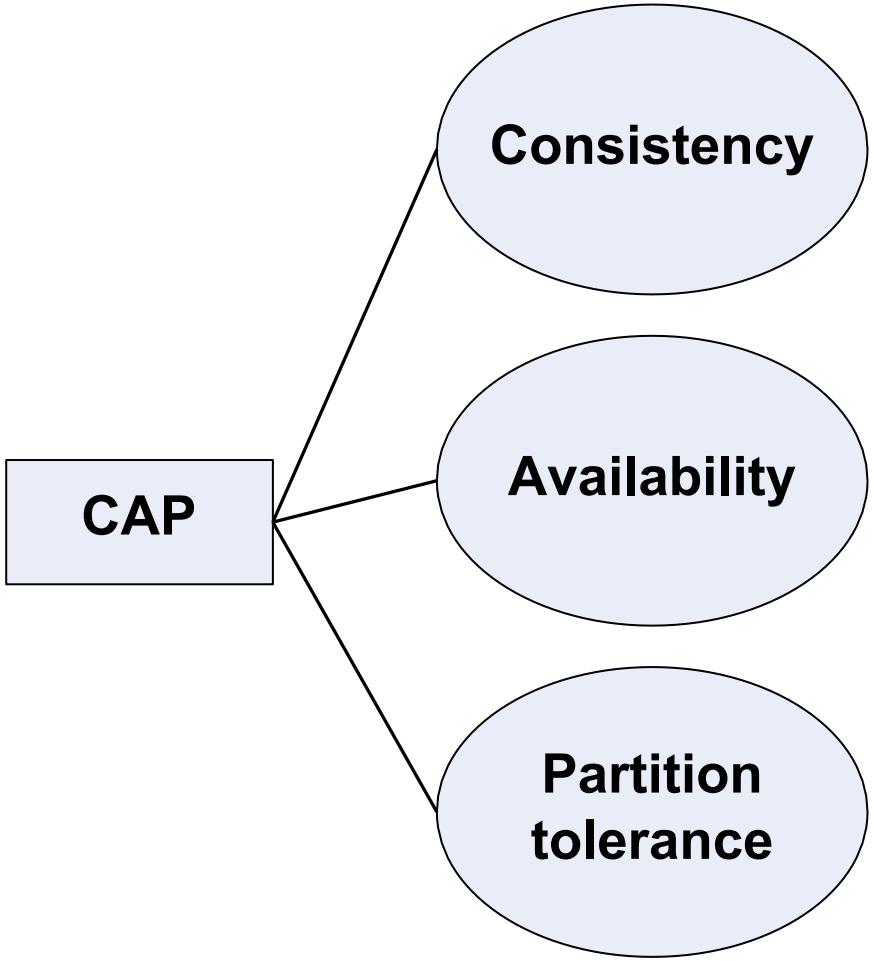
```
{_id: ObjectId("5effaa5662679b5af2c58829"),
 email: “email@example.com”,
 name: {given: “Jesse”, family: “Xiao”},
 age: 31,
 addresses: [{label: “home”,
              street: “101 Elm Street”,
              city: “Springfield”,
              state: “CA”,
              zip: “90000”,
              country: “US”},
             {label: “mom”,
              street: “555 Main Street”,
              city: “Jonestown”,
              province: “Ontario”,
              country: “CA”}]

}
```


Relationships can have properties (name/value pairs)
:HAS_CEO
start_date: 2008-01-20
Employee
Company
Relationships are directional
:LOCATED_IN
City
Relationships connect nodes and represent actions (verbs)
Nodes represent objects (nouns)
name: Amy Peters
date_of_birth: 1984-03-01
employee_ID: 1
Nodes can have properties (name/value pairs)

SQL Vs NoSQL

SQL	NoSQL
SQL databases are primarily called RDBMS or Relational Databases.	NoSQL databases are primarily called as Non-relational or distributed database.
SQL databases are table-based databases.	NoSQL databases can be document based, key-value pairs, graph databases.
Vertical Scalability	Horizontal Scalability
Fixed or Predefined schema.	Flexible schema.
Not suitable for hierarchical data storage.	Suitable for hierarchical data storage.
Oracle, MySQL, Microsoft SQL Server, and PostgreSQL.	Document: MongoDB and CouchDB, Key-value: Redis and DynamoDB, Column-based: Cassandra and HBase, Graph: Neo4j and Amazon Neptune.

Brewer's CAP Theorem

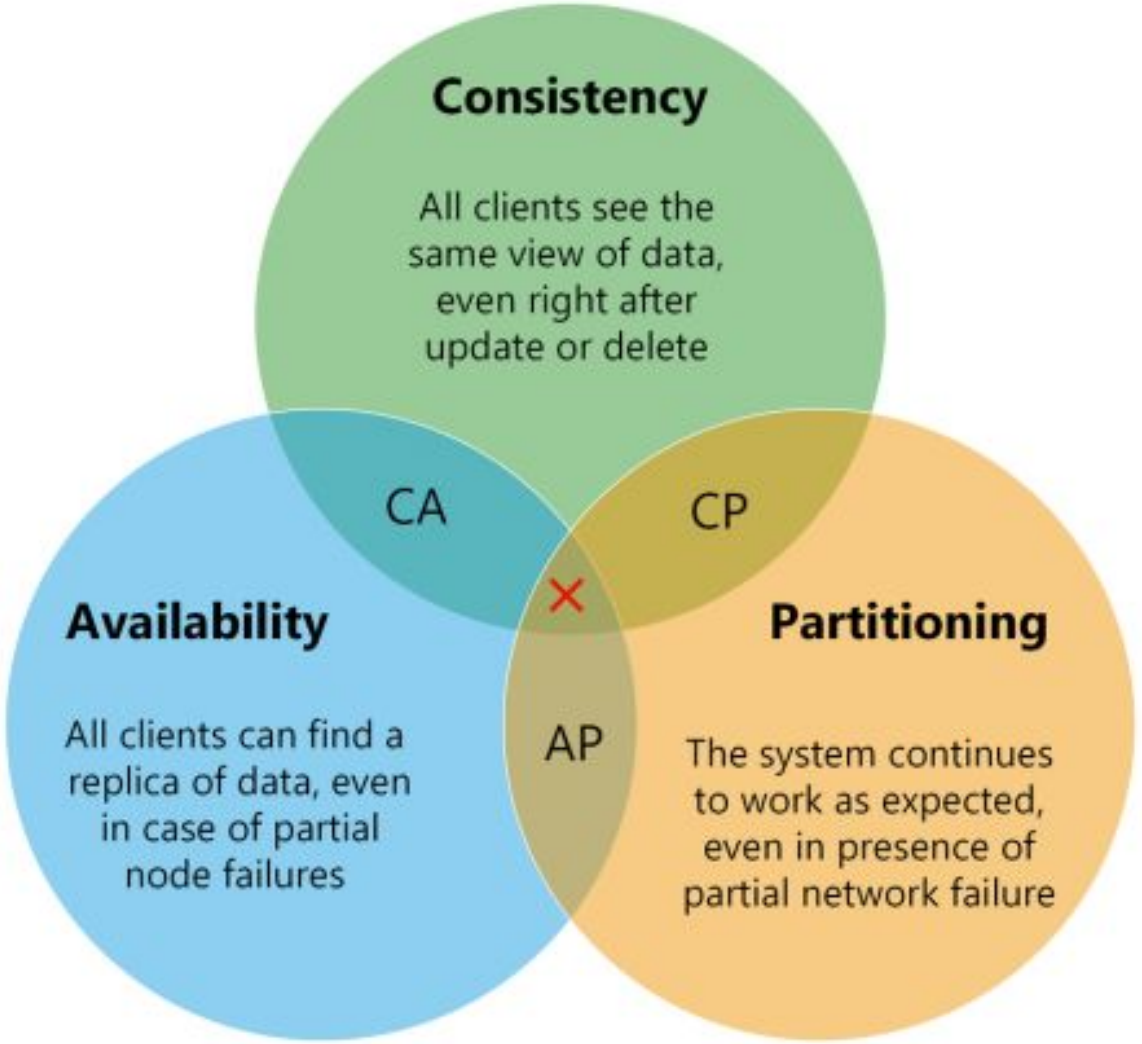


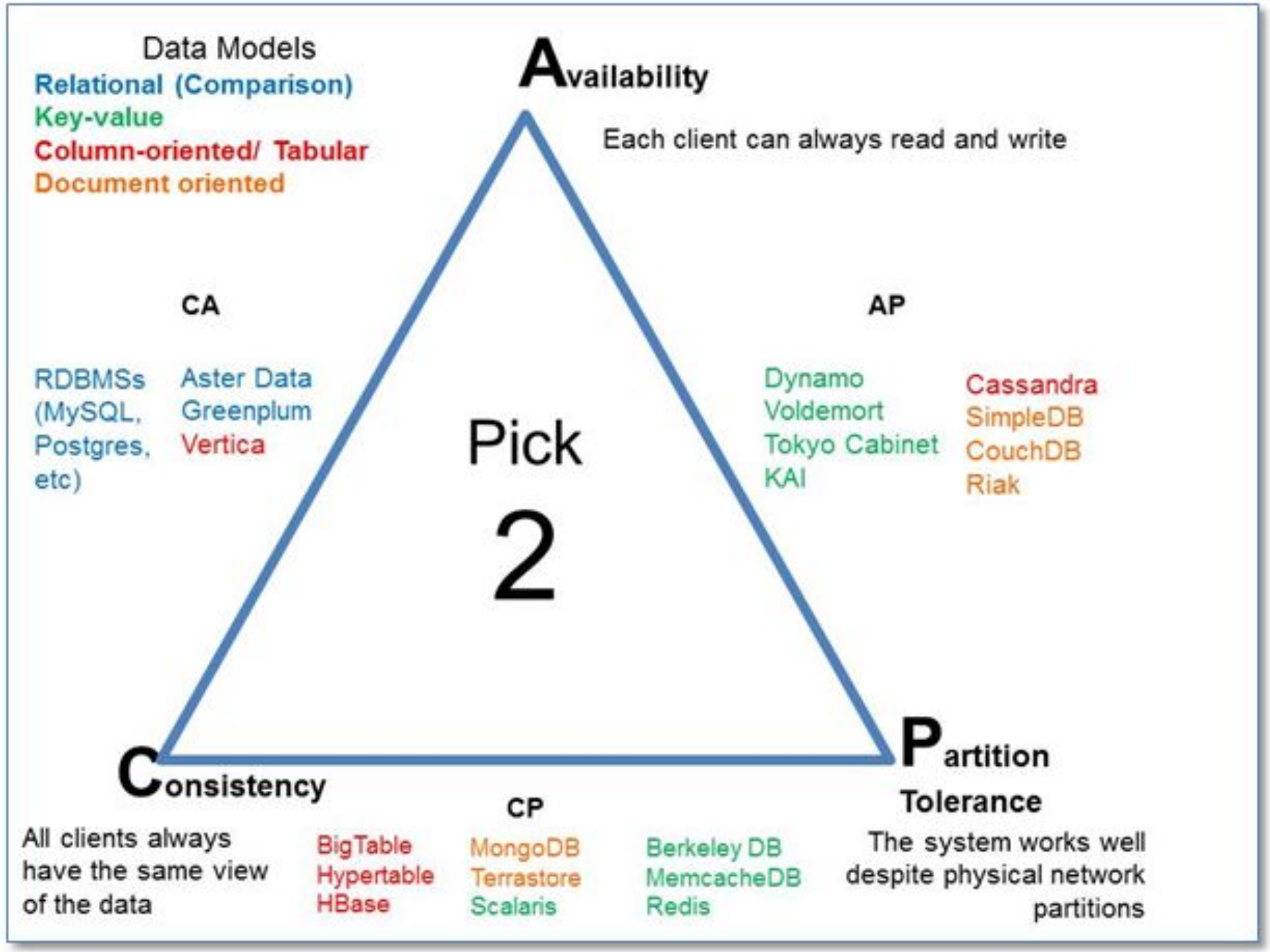
- **Consistency** implies that every read fetches the last write.
- **Availability** implies that reads and writes always succeed. In other words, each non-failing node will return a response in a reasonable amount of time.
- **Partition tolerance** implies that the system will continue to function when network partition occurs.

When to choose consistency over availability and vice-versa

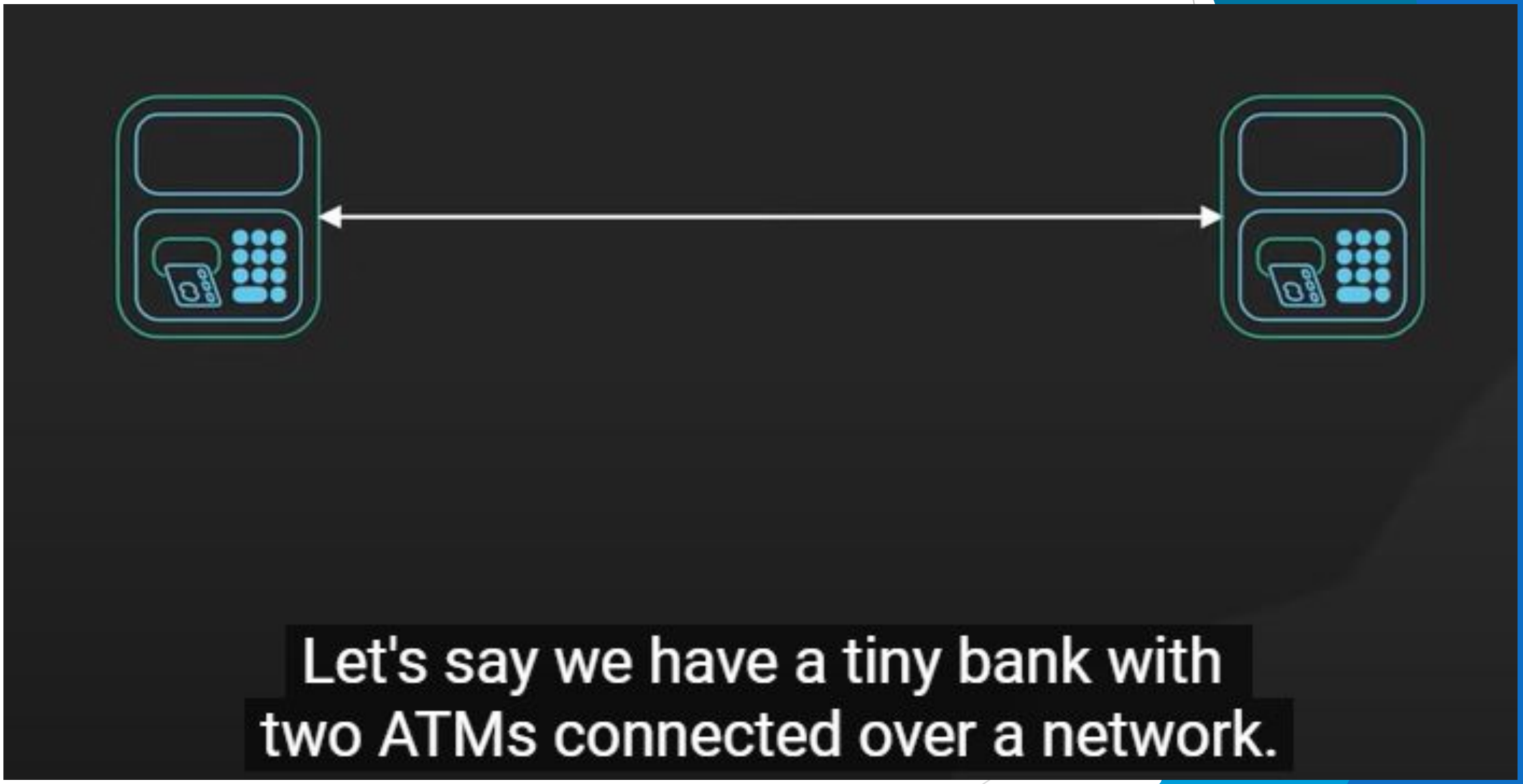
1. Choose availability over consistency when your business requirements allow some flexibility around when the data in the system synchronizes.

2. Choose consistency over availability when your business requirements demand atomic reads and writes.

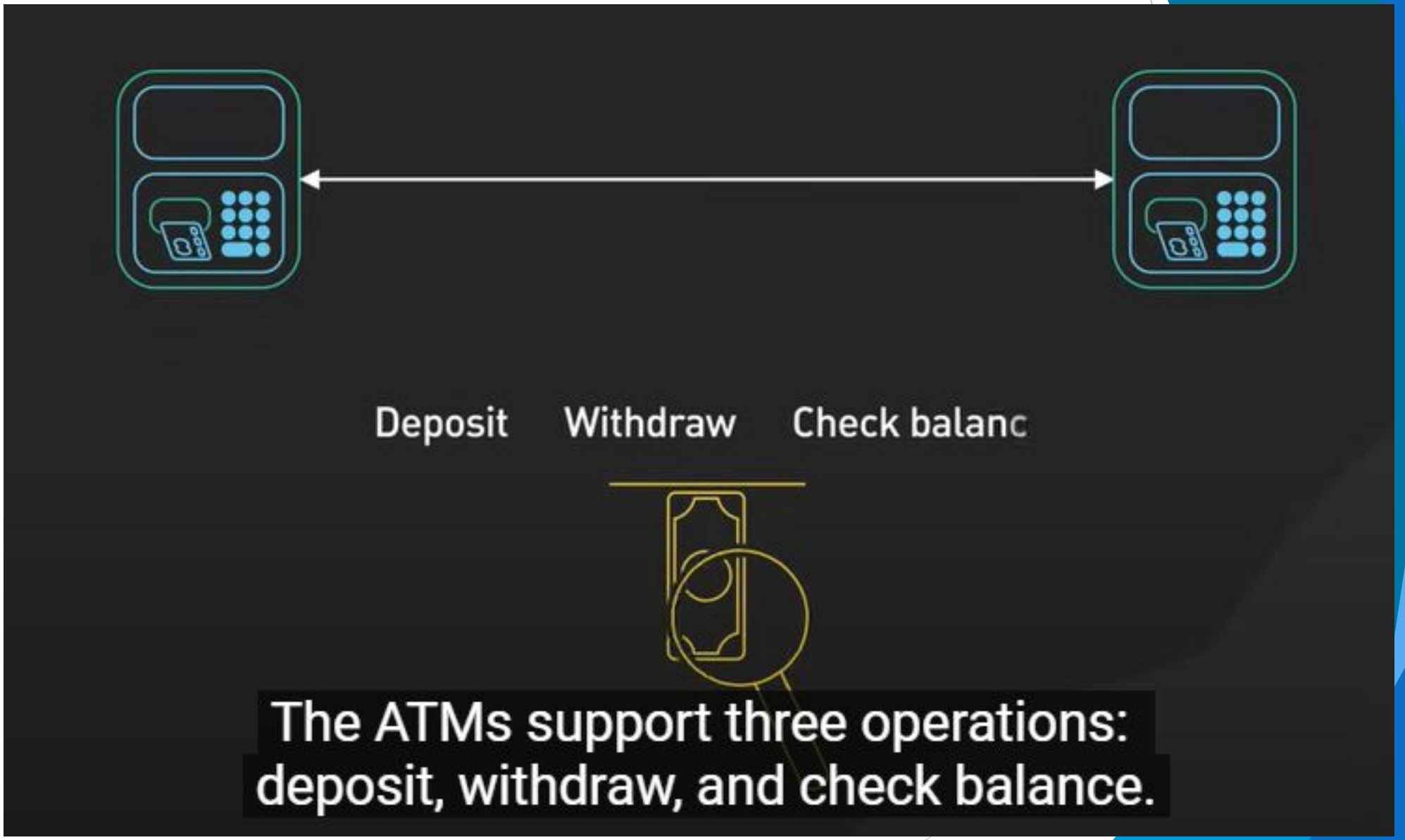

Consistency
All clients see the same view of data, even right after update or delete
CA
CP
Availability
All clients can find a replica of data, even in case of partial node failures
AP
Partitioning
The system continues to work as expected, even in presence of partial network failure


Data Models
Relational (Comparison)
Key-value
Column-oriented/ Tabular
Document oriented
Availability
Each client can always read and write
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
Pick 2
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Consistency
All clients always have the same view of the data
CP
BigTable
Hypertable
HBase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance
The system works well despite physical network partitions

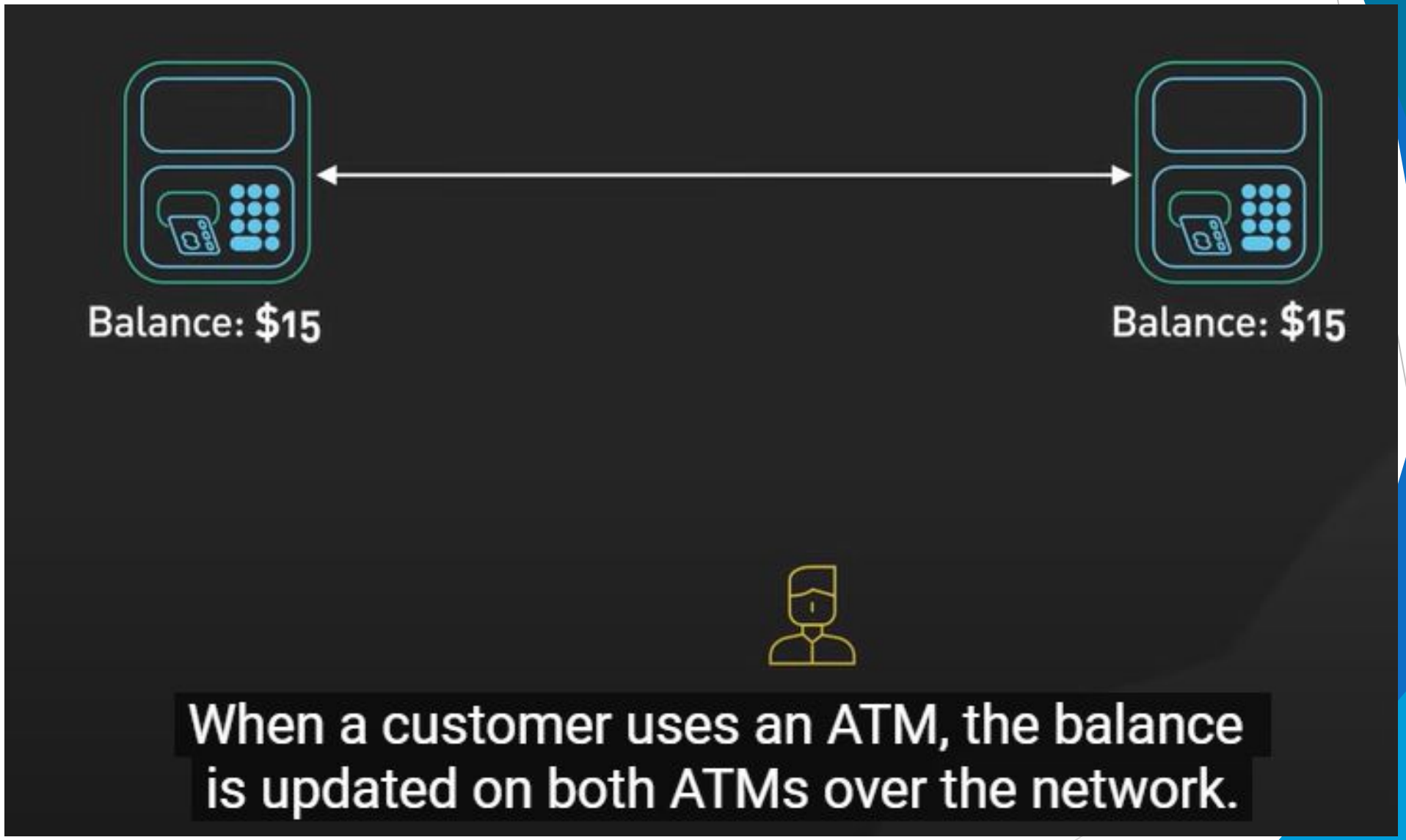
CAP Theorem


Let's say we have a tiny bank with
two ATMs connected over a network.

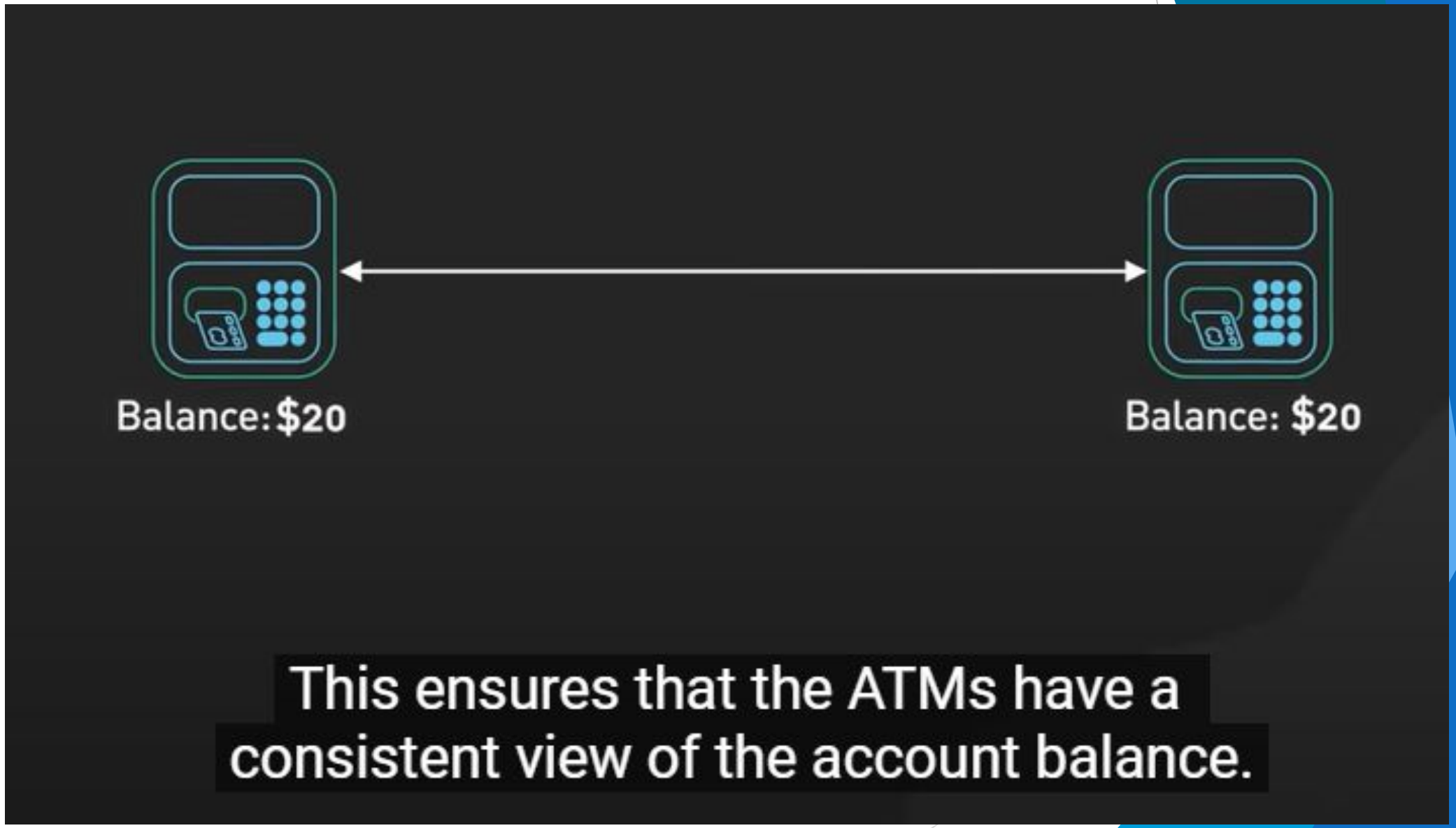
CAP Theorem


Deposit
Withdraw
Check balanc
The ATMs support three operations:
deposit, withdraw, and check balance.

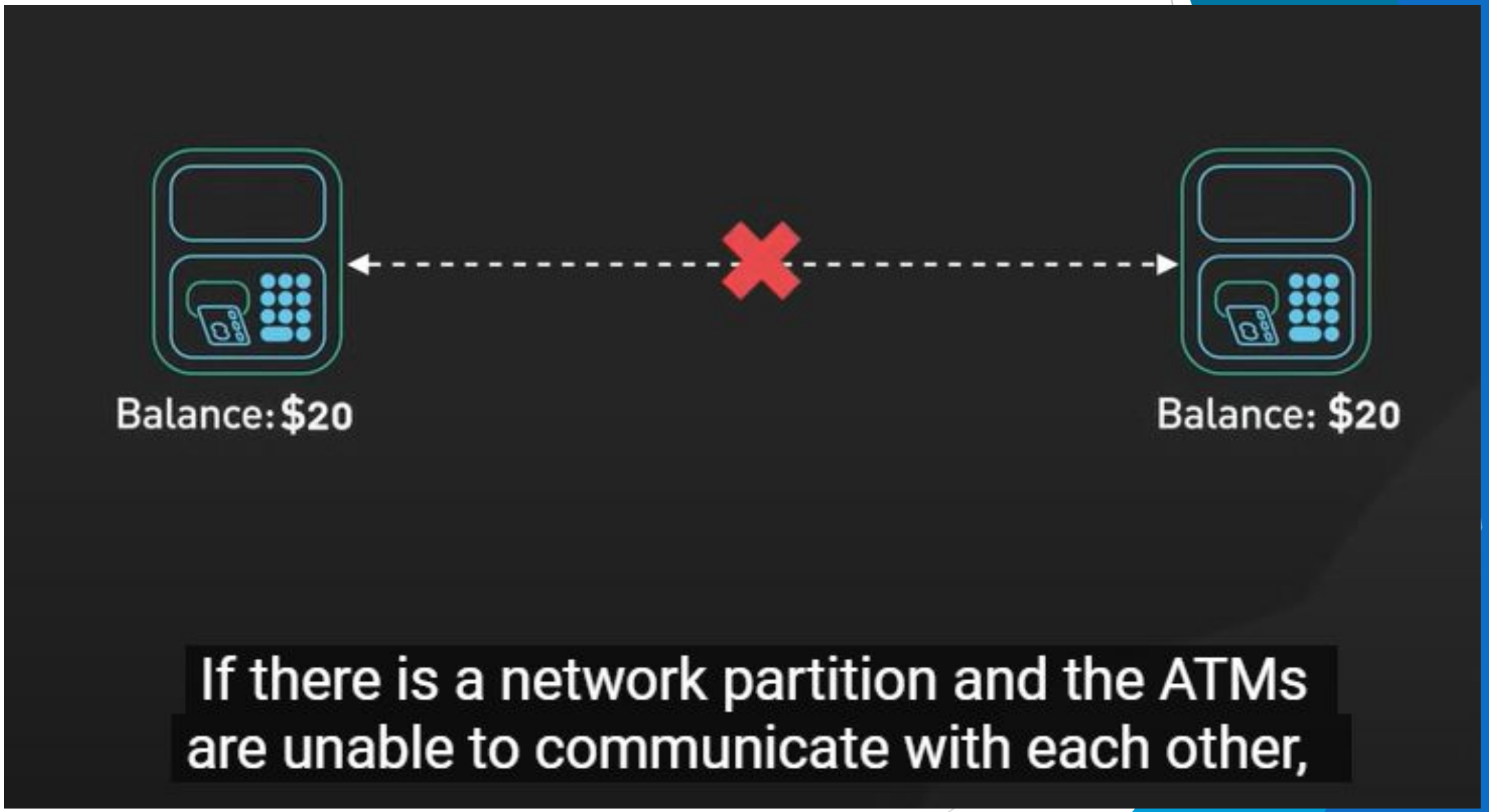
CAP Theorem


Balance: $15
Balance: $15
When a customer uses an ATM, the balance is updated on both ATMs over the network.

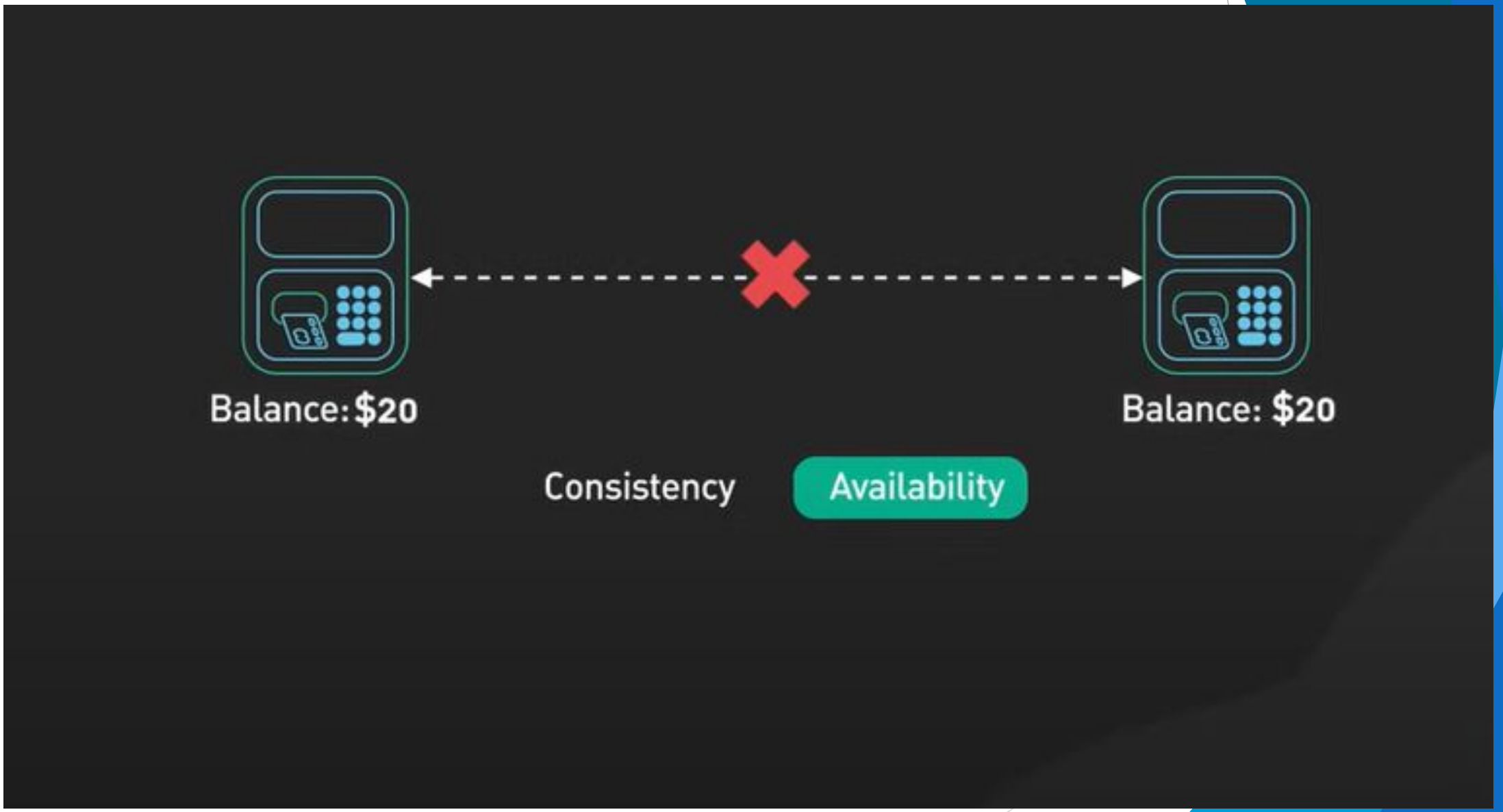
CAP Theorem


Balance: $20
Balance: $20
This ensures that the ATMs have a consistent view of the account balance.

CAP Theorem

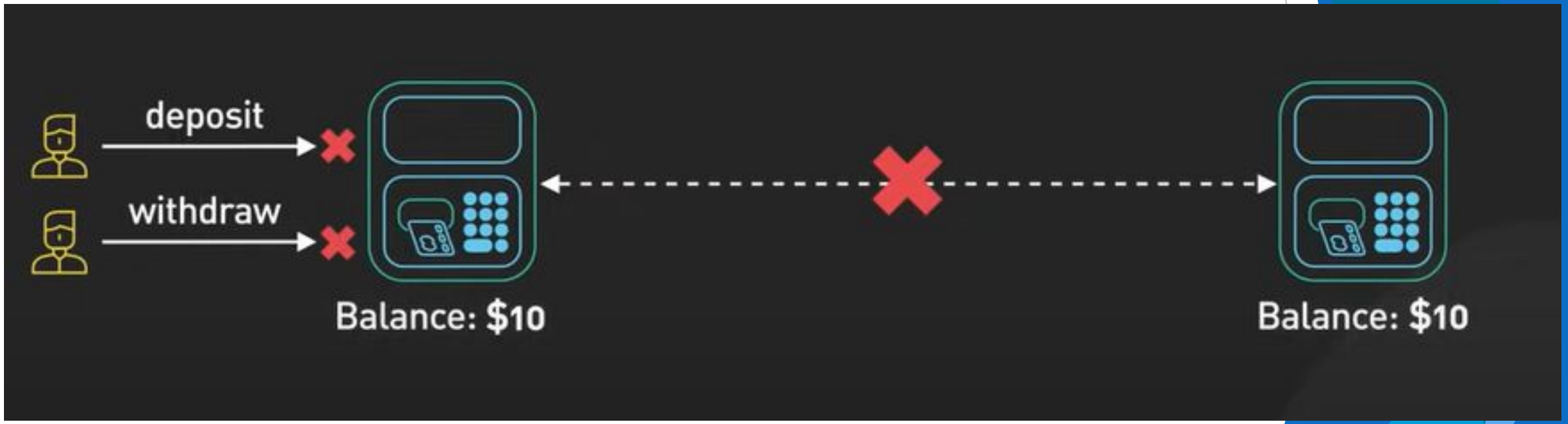

Balance: $20
Balance: $20
If there is a network partition and the ATMs are unable to communicate with each other,

CAP Theorem



CAP Theorem

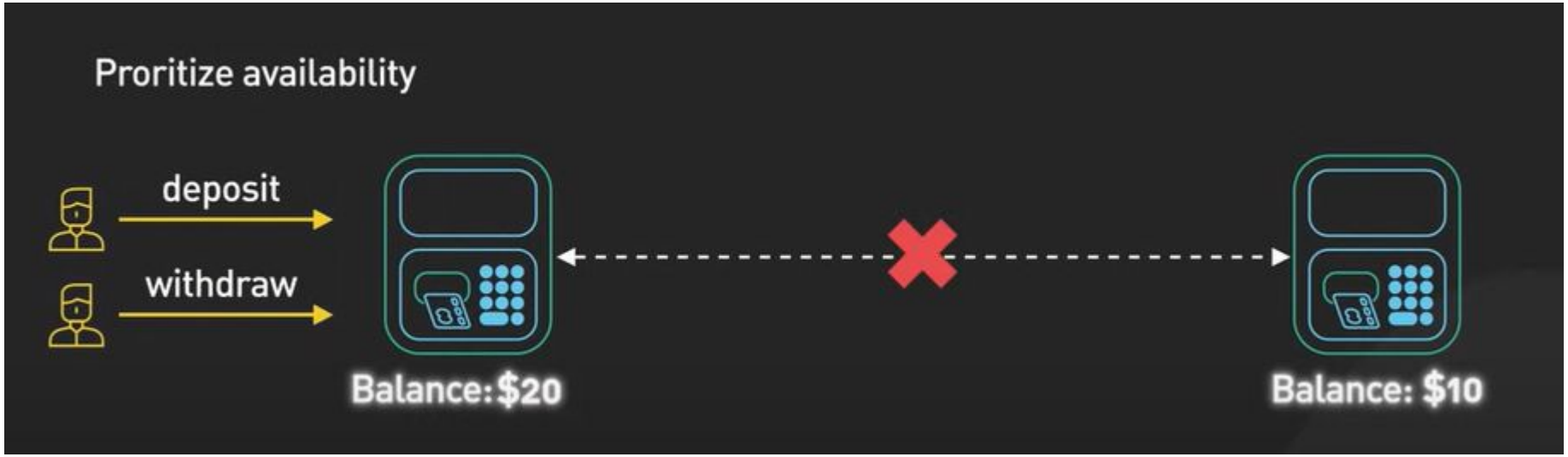
- If the bank prioritizes consistency, ATM may refuse to process deposits or withdrawls until the Partition is resolved.



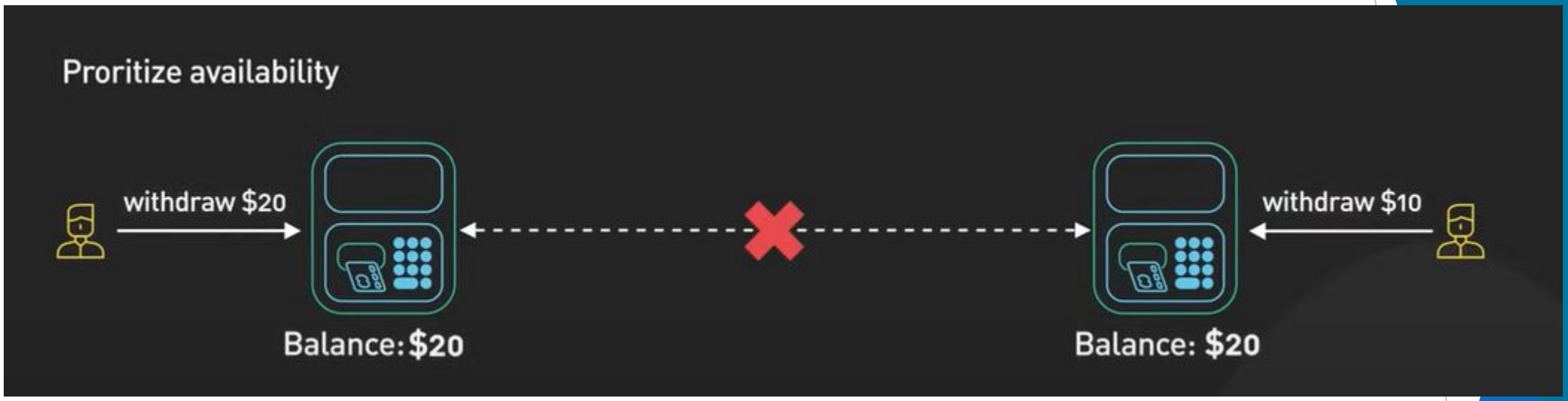
- This ensures that the balance remains **consistent**, but the system is **unavailable** to the Customer.

CAP Theorem

- If the Bank prioritize Availability, the ATM may allow withdrawals and deposits to occur, but the data Remain Inconsistent until the Partition Tolerance is resolved.



CAP theorem



When there is a network Partition, The Customer could withdraw the entire balance from both the ATMs.

CAP theorem

- When the Network comes back online, the inconsistency is resolved and now the balance is negative. That is not good.

When to use NoSQL?

1. When a huge amount of data needs to be stored and retrieved.
2. The relationship between the data you store is not that important
3. The data changes over time and is not structured.
4. Constraints and Joins support is not required at database level.
5. The data is growing continuously, and you need to scale the database regularly to handle the data.

What is MongoDB?

MongoDB is:

1. Cross-platform- .
2. Open source.
3. Non-relational.
4. Distributed.
5. NoSQL.
6. Document-oriented data store.

Why MongoDB?

Why MongoDB?

- Open Source
- Distributed- storing data across multiple servers or computers instead of just one
- Fast In-Place Updates- A method of modifying data within a database or data structure where only the specific parts that need changing are altered directly, without needing to read the entire data set or create a temporary copy
- Replication- the process of copying data from a primary server to multiple secondary servers
- Full Index Support -MongoDB supports many types of indexes, including single-field, compound, multikey, geospatial, text search, hashed, and wildcard indexes. These indexes help MongoDB execute queries efficiently.
- Rich Query Language - a query language that supports a wide range of queries and can search and retrieve data based on specific conditions.
- Easy Scalability- Easy scalability primarily through its built-in support for "horizontal scaling" using a feature called "sharding," which allows you to distribute data across multiple servers by splitting it into smaller chunks,
- Auto sharding- MongoDB automatically distributes data across multiple shards, allowing the database system to handle large datasets and high user demands

JSON

Using JSON

- JSON, or JavaScript Object Notation, is a human-readable data interchange format.

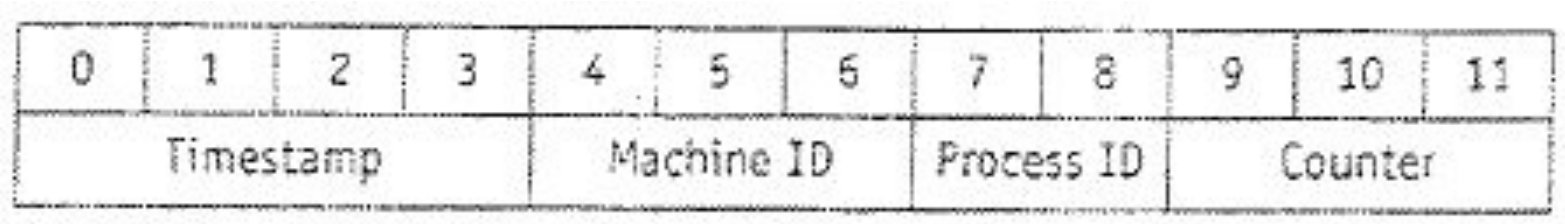
- JSON *objects* are associative containers, wherein a string key is mapped to a value (which can be a number, string, boolean, array, an empty value — null, or even another object).

- **BSON**, or Binary JSON, is the data format that MongoDB uses to organize and store c

```
{ _id:
Name :  "Sujeet",
Age :  22,
Address :  { {street: "H1 Block",
             City: "Sultanpuri",
             State: "Delhi"
             Zip: "110086"
             Country: "India"} }
}
```

Creating or Generating a Unique Key

- Each JSON document should have a unique identifier.
- It is similar to the primary key of relational databases.
- This facilitates search for documents based on the unique identifier.
- ObjectIds use 12 bytes of storage, which gives them a string representation that is 24 hexadecimal digits: 2 digits for each byte.



Support for Dynamic Queries

MongoDB has extensive support for dynamic queries.

This is in keeping with traditional RDBMS wherein we have static data and dynamic queries.
"Static data" refers to a fixed dataset that doesn't change frequently
"dynamic queries" are queries that can be constructed and executed at runtime

Example:
Static data:
A database table containing a list of all countries with their corresponding country codes.
Dynamic query:
A query that retrieves a list of customers from this table based on a user-selected region, where the region filter is provided dynamically when the user makes a selection.

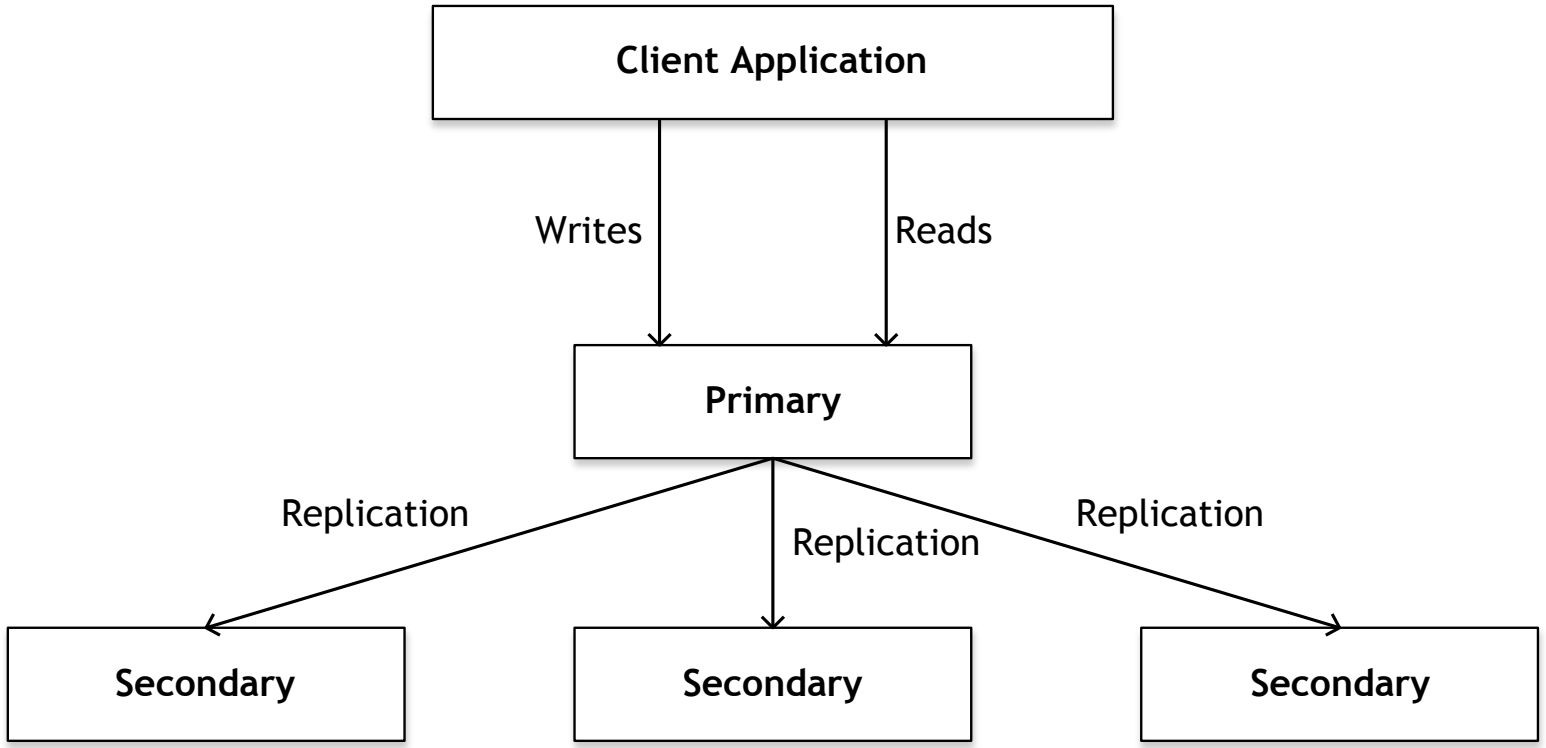
Storing Binary Data

MongoDB provides GridFS to support the storage of binary data. It can store up to 4 MB of data. This suffices for photographs or small video clips.

If one wishes to store movie clips, it stores the metadata in a collection called “file”. It then breaks the data into small pieces called chunks and stores it in the chunks collection.

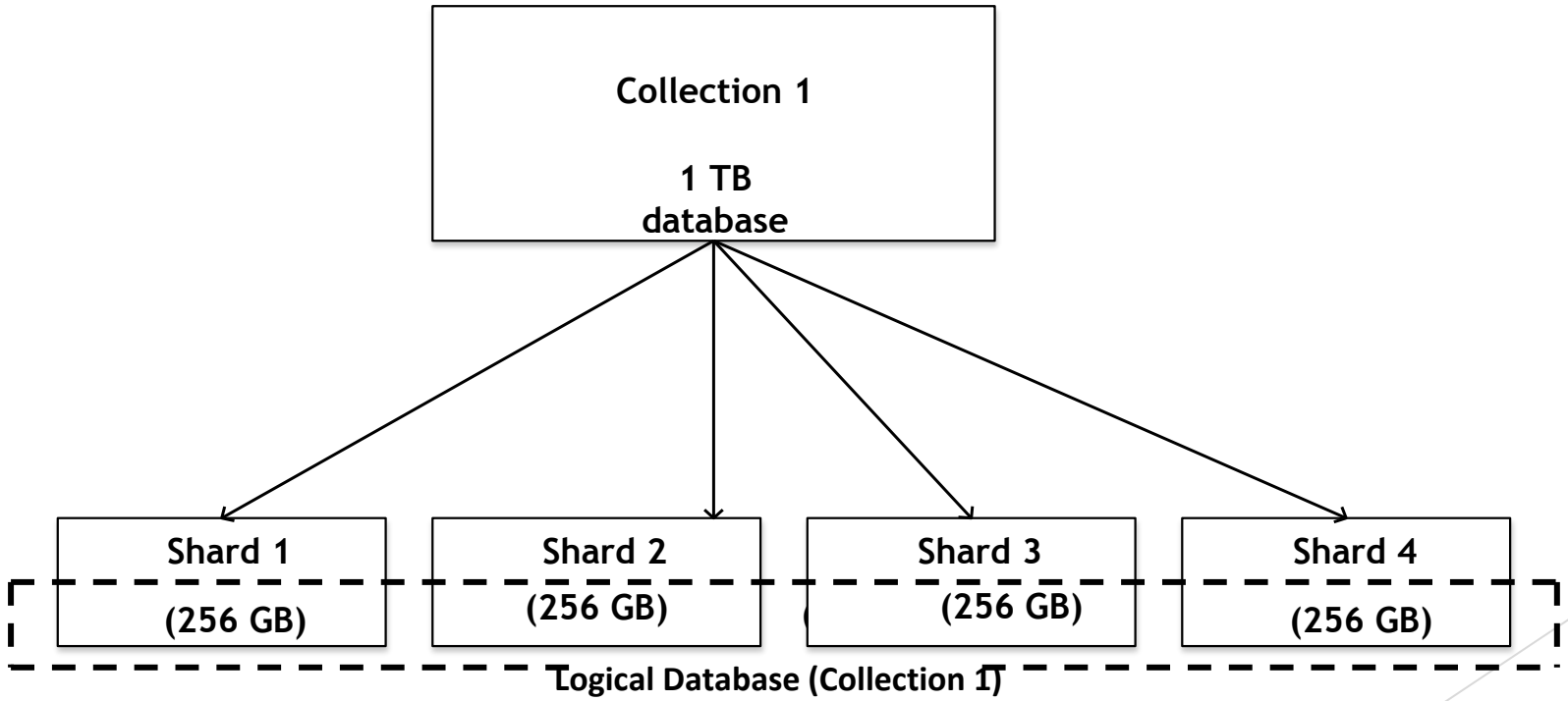
Replication in MongoDB

- MongoDB provides data redundancy and high availability.
- It helps to recover from hardware failure and service interruption.

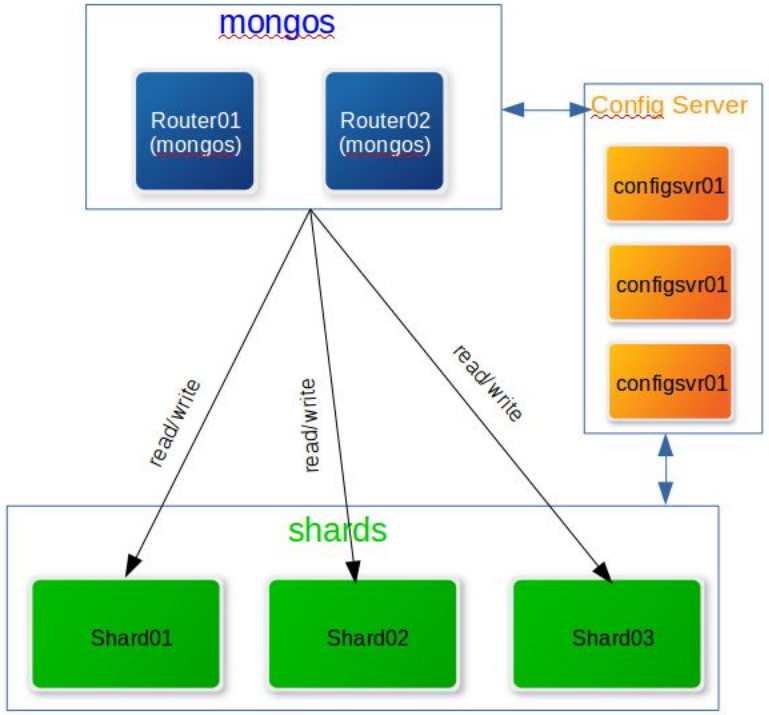


Sharding in MongoDB

- Sharding is a method for distributing or partitioning data across multiple machines.
- Horizontal scaling, also known as scale-out, refers to adding machines to share the data set and load.



Sharding in MongoDB



In MongoDB, a sharded cluster consists of:

•Shards

•Mongos

•Config servers

- A **shard** is a replica set that contains a subset of the cluster's data.

- The **mongos** acts as a query router for client applications, handling both read and write operations. It dispatches client requests to the relevant shards and aggregates the result from shards into a consistent client response. Clients connect to a mongos, not to individual shards.

- **Config servers** are the authoritative source of sharding metadata. The sharding metadata reflects the state and organization of the sharded data. The metadata contains the list of sharded collections, routing information, etc.

Advantages of Sharding

- Increased read/write throughput.
- Increased storage capacity.
- Data Locality.

Who's is using MongoDB?

MongoDB has been adopted as backend software by a few major websites and services including Toyota, Cisco, Shutterfly, Adobe, Ericsson, Craigslist, eBay, and Foursquare.

Terms Used in RDBMS and MongoDB

Terms Used in RDBMS and MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Record	Document
Columns	Fields / Key Value pairs
Index	Index
Joins	Embedded documents
Primary Key	Primary key (_id is a identifier)

SQL Vs MongoDB Terms

SQL Term	MongoDB Term
Database	Database
Table	Collection
Index	Index
Row	BSON document
Column	BSON field
Primary Key	_id field
Group by	Aggregation
Join	Embedding and Linking

Data Types in MongoDB

Data Types in MongoDB

String	Must be UTF-8 valid. Most commonly used data type.
Integer	Can be 32-bit or 64-bit (depends on the server).
Boolean	To store a true/false value.
Double	To store floating point (real values).
Min/Max keys	To compare a value against the lowest or highest BSON elements.
Arrays	To store arrays or list or multiple values into one key.
Timestamp	To record when a document has been modified or added.
Null	To store a NULL value. A NULL is a missing or unknown value.
Date	To store the current date or time in Unix time format. One can create object of date and pass day, month and year to it.
Object ID	To store the document's id.
Binary data	To store binary data (images, binaries, etc.).
Code	To store javascript code into the document.
Regular expression	To store regular expression.

Create Database

The **use Database_name** command makes a new database in the system if it does not exist, if the database exists it uses that database.

For example, use mydb.
Now your database is ready of name mydb.

```
>use mydb
switched to db mydb
```

List all databases

To list down all the databases, use the command below:
show dbs

This command lists down all the databases and their size on the disk.

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

mydb database is not created until you save a document in it.

CRUD in MongoDB

Collections

To create a collection by the name “Person”. Let us take a look at the
collection list prior to the creation of the new collection “Person”.

```
db.createCollection(“Person”);
```

List all Collections

To see all collections in a database, use the command below:
show collections

```
>show collections
hi
students
```

Collections

To drop a database

```
db.dropDatabase()
```

To drop a collection by the name “student”.

```
db.student.drop();
```

Create Documents

Two methods insert or create documents in collection:

1. db.collection_name.insertOne()
2. db.collection_name.insertMany()

```
>db.students.insertOne({name:"Ved Pandey",age:22,gender:`Male`,course:`BCA`})
{
    "acknowledged" : true,
    "insertedId" : ObjectId("6395da769e1a378e0e4a9f72")
}

>db.students.insertMany([{name:"Neha Singh",age:21,gender:`Female`,course:`B.Com`},{name:"Ankita Tripathi",age:22,gender:`Female`,course:`MCA`,email:`nehagmail.com`}])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("6395daf9f05532c3d92c7f51"),
        ObjectId("6395daf9f05532c3d92c7f52")
    ]
}
```

Retrieving Documents

- Use the below command to retrieve documents:

db.collection_name.find()

- For a proper formatted output, use the below command: pretty() with find().

db.collection_name.find().pretty()

```
>db.students.find()
{ "_id" : ObjectId("6395d963f07f724a7192d41a"), "name" : "Surya Gupta", "age" : 23, "gender" : "Male", "course" : "B.Tech" }
{ "_id" : ObjectId("6395da769e1a378e0e4a9f72"), "name" : "Ved Pandey", "age" : 22, "gender" : "Male", "course" : "BCA" }
{ "_id" : ObjectId("6395daf9f05532c3d92c7f51"), "name" : "Neha Singh", "age" : 21, "gender" : "Female", "course" : "B.Com" }
{ "_id" : ObjectId("6395daf9f05532c3d92c7f52"), "name" : "Ankita Tripathi", "age" : 22, "gender" : "Female", "course" : "MCA", "email" : "nehagmail.com" }
```

Update Method

Insert the document for “Aryan David” into the Students collection only if it does not already exist in the collection. However, if it is already present in the collection, then update the document with new values. (Update his Hobbies from “Skating” to “Chess”.) Use “Update else insert” (if there is an existing document, it will attempt to update it, if there is no existing document then it will insert it).

db.Students.update({_id:3, StudName:"Aryan David", Grade: "VII"},{$set:{Hobbies: "Skating"}},{upsert:true});

Find Method

To search for documents from the “Students” collection based on certain search criteria.

db.Students.find({StudName:"Aryan David"});

Find Method

To display only the StudName and Grade from all the documents of the Students collection. The identifier _id should be suppressed and NOT displayed.

db.Students.find({},{StudName:1,Grade:1,_id:0});

To display the StudName, Grade as well the identifier, _id from the document of the Students collection where the _id column is 1.

db.Students.find({_id:1},{StudName:1,Grade:1});

To display the StudName, Grade from the document of the Students collection where the _id column is 1. The _id field should not be displayed.

db.Students.find({_id:1},{StudName:1,Grade:1,_id:0});

Find Method

To find those documents where the Grade is set to ‘VII’

db.Students.find({Grade:{$eq:'VII'}}).pretty();

To find those documents where the Grade is not set to ‘VII’

db.Students.find({Grade:{$ne:'VII'}}).pretty();

Find Method

To find those documents from the Students collection where the Hobbies is set to either ‘Chess’ or is set to ‘Skating’.

db.Students.find ({Hobbies :{ $in: ['Chess','Skating']}}).pretty ();

To find those documents from the Students collection where the Hobbies is set neither to ‘Chess’ nor is set to ‘Skating’.

db.Students.find ({Hobbies :{ $nin: ['Chess','Skating']}}).pretty ();

Find Method

To find those documents from the Students collection where the Hobbies is set to ‘Graffiti’ and the StudName is set to ‘Hersch Gibbs’ (AND condition)

db.Students.find ({Hobbies :’Graffiti’, StudName:’Hersch Gibbs’}).pretty();

Find Method

To find documents from the Students collection where the StudName begins with “M”.

db.Students.find({StudName:/^M/}).pretty();

OR

db.Students.find({StudName:{$regex:”^M”}}).pretty();

To find documents from the Students collection where the StudName Ends in “s”.

db.Students.find({StudName:/s$/}).pretty();

Find Method

To find documents from the Students collection where the StudName has an “e”
in any position.

db.Students.find({StudName:/e/}).pretty();

OR

db.Students.find({StudName:/.*e.*/}).pretty();

OR

db.Students.find({StudName:{$regex:”e”}}).pretty();

Find method

To find those documents from the Students collection where the StudName ends in “a”.

db.Students.find({StudName:{$regex:”a$”}}).pretty();

Dealing with NULL values

To add a new field with null value in existing documents(_id:3 and _id:4) of Students collection. A NULL is a missing or unknown value. When we place NULL as a value for a specified field , it implies that currently we do not know the value or the value is missing. We can always update the value of the field once we know it.

Before we execute the commands to update documents with a null value in a column, let us first view the two documents.

```
db.Students.find({$or:[{_id:3},{_id:4}]});
```

Update the documents with NULL values in the “Location” column.

```
db.Students.update({_id:3},{$set:{Location:null}});
db.Students.update({_id:4},{$set:{Location:null}});
```

Dealing with NULL values

To search for NULL values in the Location column.

db.Students.find({Location:{$eq:null}});

To remove “Location” field having “NULL” values from the documents (_id:3 and _id:4) from the Students collection.

db.Students.update({_id:3},{$unset:{Location:null}});
db.Students.update({_id:4},{$unset:{Location:null}});

Count method

To find the number of documents in the Students collection.

db.Students.count();

To find the number of documents in the Students collection wherein the Grade is VII .

db.Students.count({Grade:”VII”});

Sort Method

To sort the documents from the Students collection in the ascending order of StudName.

db.Students.find().sort({StudName:1}).pretty();

To sort the documents from the Students collection in the descending order of StudName.

db.Students.find().sort({StudName:-1}).pretty();

To sort the documents from the Students collection first on Grade in ascending order and then on Hobbies in descending order.

db.Students.find().sort({Grade:1,Hobbies:-1}).pretty();

To sort the documents from the Students collection first on Grade in ascending order and then on Hobbies in ascending order.

db.Students.find().sort({Grade:1,Hobbies:1}).pretty();

Skip Method

To skip the first 2 documents from the Students collection.

db.Students.find().skip(2).pretty();

To sort the documents from the Students collection and skip the first document from the output.

db.Students.find().skip(1).pretty().sort({StudName:1});

To display the last 2 documents from the Students collection.

db.Students.find().pretty().skip(db.Students.count()-2);

To retrieve the third,fourth and fifth document from the Students collection.

db.Students.find().pretty().skip(2).limit(3);

Delete Documents

- There are 2 ways to delete documents:

1. db.collection_name.deleteOne()
2. db.collection_name.deleteMany()

```
>db.students.deleteOne({age:23})
{ "acknowledged" : true, "deletedCount" : 1 }
```

```
>db.students.deleteMany({age:22})
{ "acknowledged" : true, "deletedCount" : 3 }

>db.students.find().pretty()
{
    "_id" : ObjectId("6395daf9f05532c3d92c7f51"),
    "name" : "Neha Singh",
    "age" : 21,
    "gender" : "Female",
    "course" : "B.Com"
}
{
    "_id" : ObjectId("6395eacc3d031007a5f010b8"),
    "name" : "Neha Singh",
    "age" : 21,
    "gender" : "Female",
    "course" : "B.Com"
}
```

Adding new field - update method

To add a new field to an existing document in MongoDB, you can use the updateOne() or updateMany() method along with the $set operator.

```
db.collection.updateOne(
  // Filter to select the document(s) you want to update
  { <filter> },
  // Update operation
  {
    // $set operator to add a new field
    $set: {
      newField: "value" // Specify the new field and its value
    }
  }
);

db.users.updateOne(
  { name: "John" }, // Filter documents where name is "John"
  { $set: { age: 30 } } // Add the new field "age" with value 30
);
```

Removing an existing field

To remove an existing field from an existing document in MongoDB, you can use the updateOne() or updateMany() method along with the $unset operator.

```
db.collection.updateOne(
  // Filter to select the document(s) you want to update
  { <filter> },

  // Update operation
  {
    // $unset operator to remove a field
    $unset: {
      fieldName: "" // Specify the name of the field you want to remove
    }
  }
);

db.users.updateOne(
  { name: "John" }, // Filter documents where name is "John"
  { $unset: { age: "" } } // Remove the "age" field
);
```

Save Method

Save() method will insert a new document if the document with the specified _id does not exist. However, if a document with specified id exists, it replaces the existing document with the new one.

```
// Update an existing document with _id "123abc"
db.collection.save({
  _id: "123abc",
  name: "Updated Name",
  age: 30,
  status: "active"
});

// Insert a new document if _id "456def" doesn't already exist
db.collection.save({
  _id: "456def",
  name: "New Document",
  age: 25,
  status: "inactive"
});
```

Remove Method

Remove() method is used to delete documents from a collection that match a specified query. It can remove either a single document or multiple documents depending on the criteria provided.

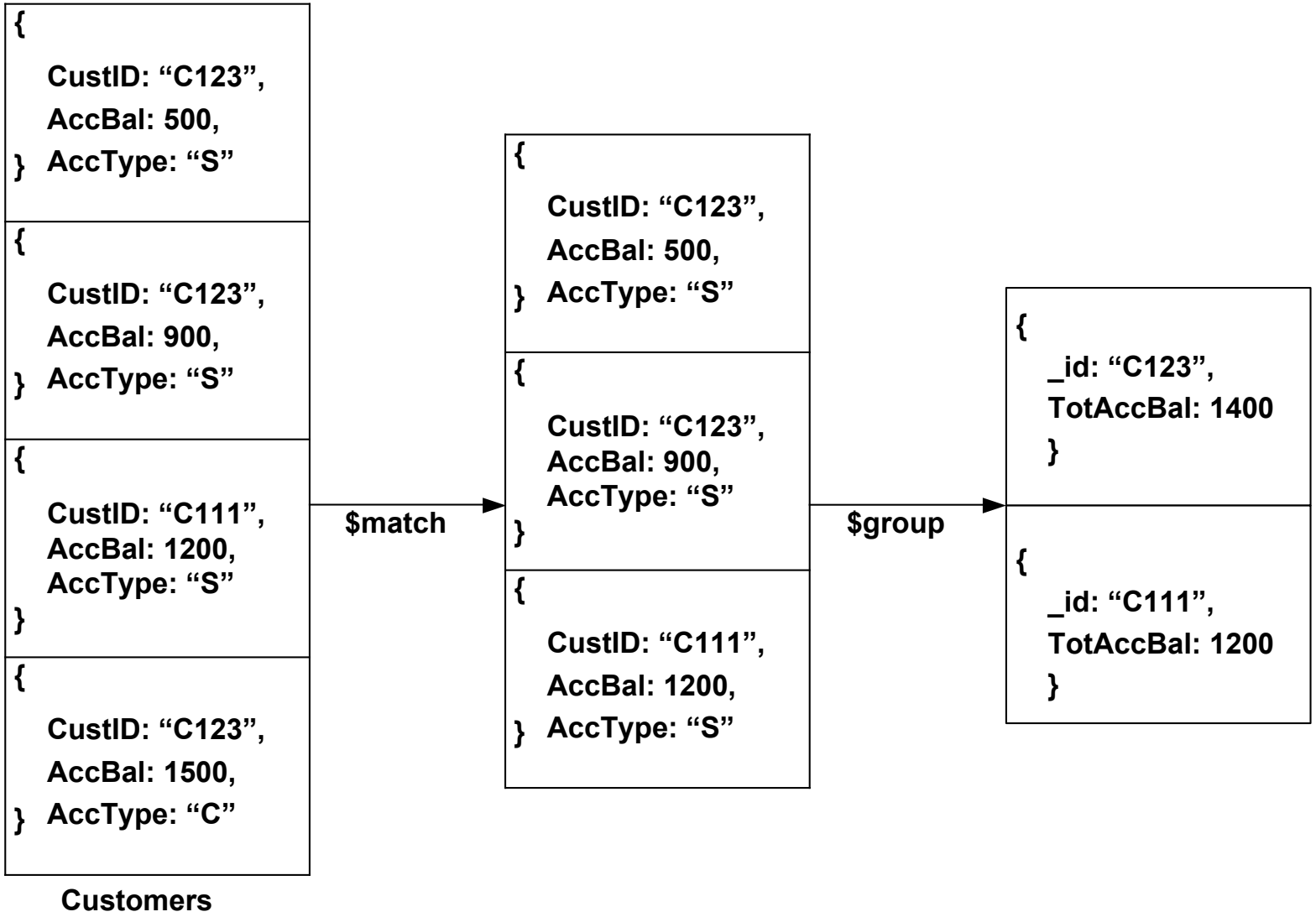
```
db.collection.remove(
   <query>,
   {
     justOne: <boolean>,
     writeConcern: <document>
   }
)
```

```
db.users.remove({ name: "John" });
```

```
db.users.remove({ name: "John" }, { justOne: true });
```

Aggregate Function

Aggregate Function



Aggregate Function

To group on “CustID” and compute the sum of “AccBal”, use the below syntax:

```
db.Customers.aggregate({$group:{_id: “$CustID”,”TotAccBal:{$sum:”$AccBal”}}});
```

First filter on “AccType:S” and then group it on “CustID” and then compute the sum of “AccBal” and then filter those documents wherein the “TotAccBal” is greater than 1200, use the below syntax:

```
db.Customers.aggregate( { $match : {AccType : "S" } },
{ $group : { _id : "$CustID",TotAccBal : { $sum : "$AccBal" } } },
{ $match : {TotAccBal : { $gt : 1200 } }});
```

Aggregate Function

- To group on “CustID” and compute the average of the “AccBal” for each group:

```
db.Customers.aggregate({ $group : { _id : "$CustID",TotAccBal : { $avg : "$AccBal" } } });
```

- To group on “CustID” and determine the maximum “AccBal” for each group:

```
db.Customers.aggregate({ $group : { _id : "$CustID",TotAccBal : { $max : "$AccBal" } } });
```

- To group on “CustID” and determine the minimum “AccBal” for each group:

```
db.Customers.aggregate({ $group : { _id : "$CustID",TotAccBal : { $min : "$AccBal" } } });
```

MongoImport

Import data from a CSV file

Given a CSV file “sample.txt” in the D: drive, import the file into the MongoDB collection, “SampleJSON”. The collection is in the database “test”.

Mongoimport --db test --collection SampleJSON --type=csv --headerline --file d:\sample.txt

--headerline instructs mongoimport to determine the name of the fields using the first line in the CSV file(If using --type csv or --type tsv)

--fieldFile =<filename> option allows you to specify a file that holds a list of field names if your CSV or TSV file does not include field names in the first line of the file.

--file=<filename> specifies the location and name of a file containing the data to import. If you do not specify a file, mongoimport reads data from standard input.

MongoExport

Export data to a CSV file

This command used at the command prompt exports MongoDB JSON documents from “Customers” collection in the “test” database into a CSV file “Output.txt” in the D: drive.

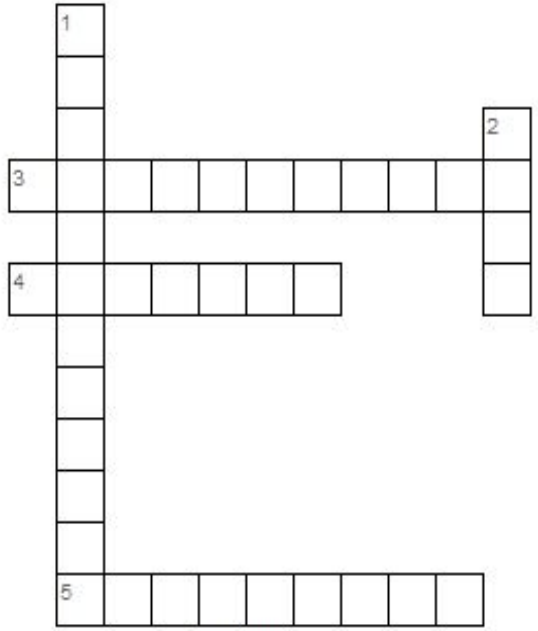
Mongoexport --db test --collection Customers --csv --fieldFile d:\fields.txt --out d:\output.txt

Answer a few quick questions …

Crossword

MongoDB

Basic puzzle on MongoDB



Across

3 MongoDB database stores its data in .....

4 MongoDB uses ---------- schemas.

5 A collection holds one or more -- ----------.

Down

1 MongoDB uses ---------- files.

2 MongoDB uses ---------, a binary object format similar to, but more expensive than JSON.

Answer Me

- What is MongoDB?
- Comment on Auto-sharding in MongoDB.
- What are collections and documents?
- What is JSON?
- Explain your understanding of Update In-Place.

Summary please…

Ask a few participants of the learning program to summarize the lecture.

References …

Further Readings

http://www.mongodb.org/

https://university.mongodb.com/

http://www.tutorialspoint.com/mongodb/

Thank you