mongoDB

What?

mongoDB.®

Humongous

Because it can store lots and lots of data

Foundations of Document Databases with MongoDB

INTRODUCTION TO MONGODB

Agenda

- What is NoSQL?
- CAP Theorem
- Types of NoSQL Databases
- Advantages of Document Database
- Why Use MongoDB?
- JSON vs BSON
- SQL Terms vs MongoDB Terms

What is NoSQL?

NoSQL = Not Only SQL

JSON (BSON) Data Format

```
{
      "name": "Max",
      "age": 29,
      "address":
            {
                  "city": "Munich"
            },
      "hobbies": [
            { "name": "Cooking" },
            { "name": "Sports" }
      ]
}
```

Relational SQL vs Document DB

MySQL	MongoDB
Structured Query Language (SQL)	MongoDB Query Language (MQL)
Predefined schema	Dynamic schema (JSON based)
Relational keys (foreign key)	No foreign key
Triggers	No triggers
ACID properties	CAP theorem
Vertically scalable	Horizontal scalable

Three Important Questions

Is your data structured or unstructured?

What is your scalability strategy for infrastructure?

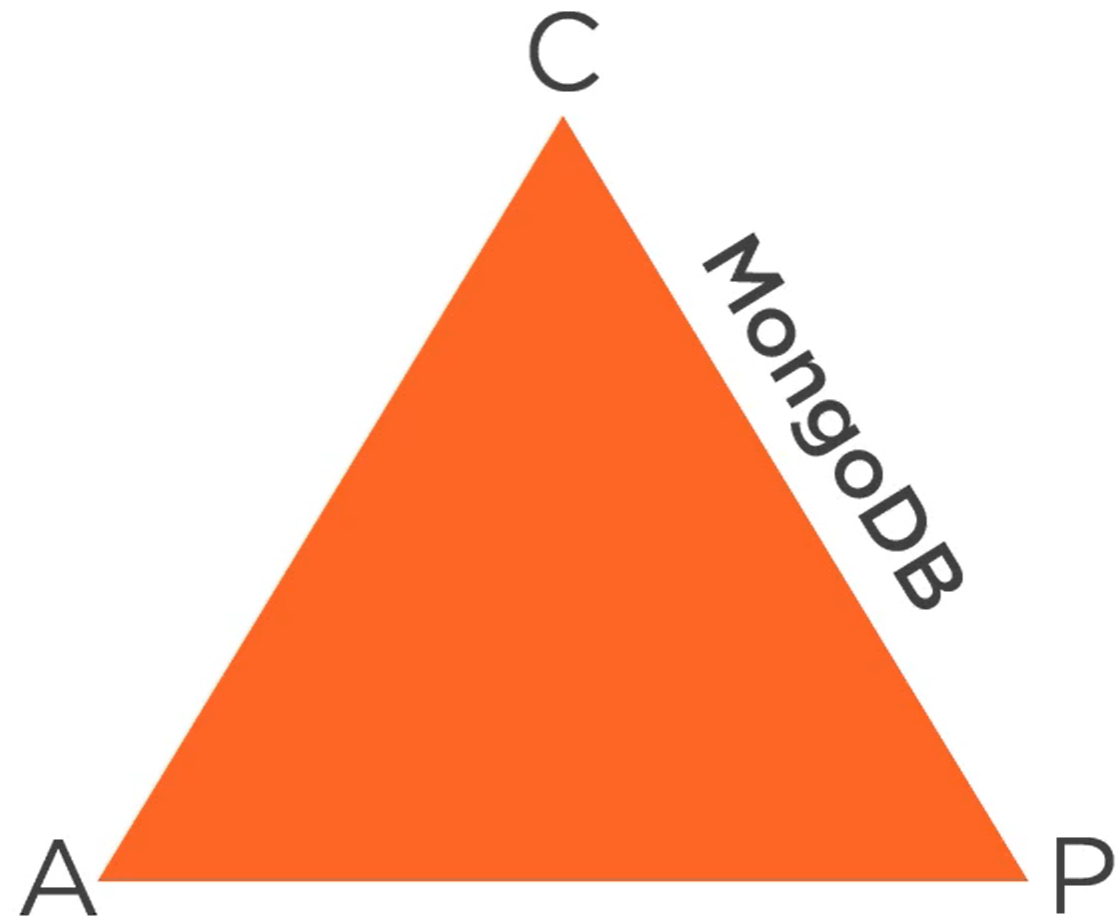
How comfortable are your devs with Object Relational Mapping?

CAP Theorem

C **Consistency**

A **Availability**

P **Partition Tolerance**



Advantages of Document Database

- **Intuitive Data Model**
- **Flexible Schema**
- **Universal JSON documents**
- **Query Data Anyway**
- **Distributed Scalable Database**

Why Use MongoDB?

- Open Source and Free (Community Ed)
- Document Database
- High Performance
- Rich Query Language
- High Availability
- Horizontal Scalability
- Multiple Storage Engine

SQL Terms vs MongoDB Terms

SQL Terms	MongoDB Terms
Database	Database
Table	Collection
Row	JSON / BSON Document
Column	Field
Index	Index

SQL Terms vs MongoDB Terms

SQL Terms	MongoDB Terms
Database	Database
Table	Collection
Row	Document
Column	Field
Index	Index
Table joins	$lookup
Primary key	Primary Key
Transactions	Transactions

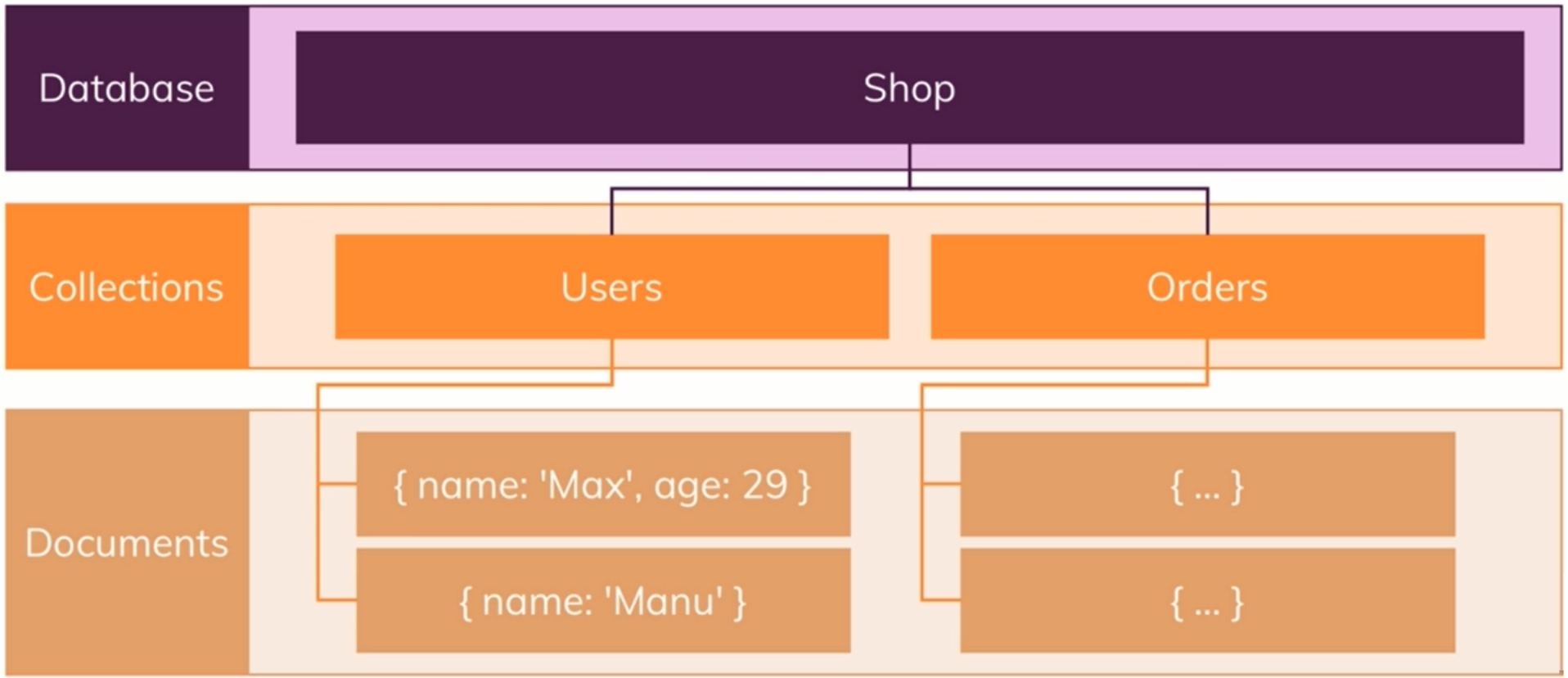
JSON (BSON) Data Format

```
{
      "name": "Max",
      "age": 29,
      "address":
            {
                  "city": "Munich"
            },
      "hobbies": [
            { "name": "Cooking" },
            { "name": "Sports" }
      ]
}
```

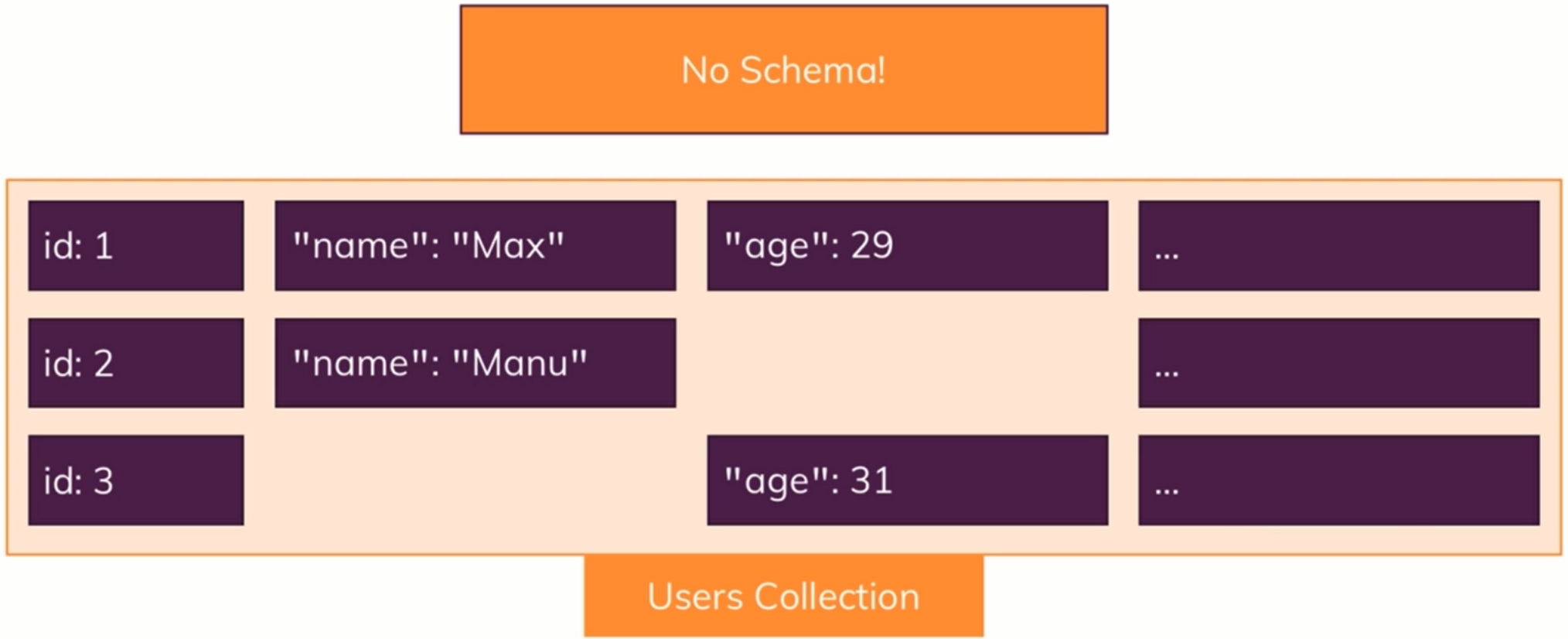
JSON vs BSON

	JSON (JavaScript Object Notation)	BSON (Binary JSON)
Encoding	UTF-8 String	Binary
Data Support	String, Boolean, Number, Array	String, Boolean, Number (Integer, Float, Long, Decimal128...), Array, Date, Raw Binary
Readability	Human and Machine	Machine Only

How it works


Database
Shop
Collections
Users
Orders
Documents
{ name: 'Max', age: 29 }
{ name: 'Manu' }
{ ... }
{ ... }

BSON Data Structure



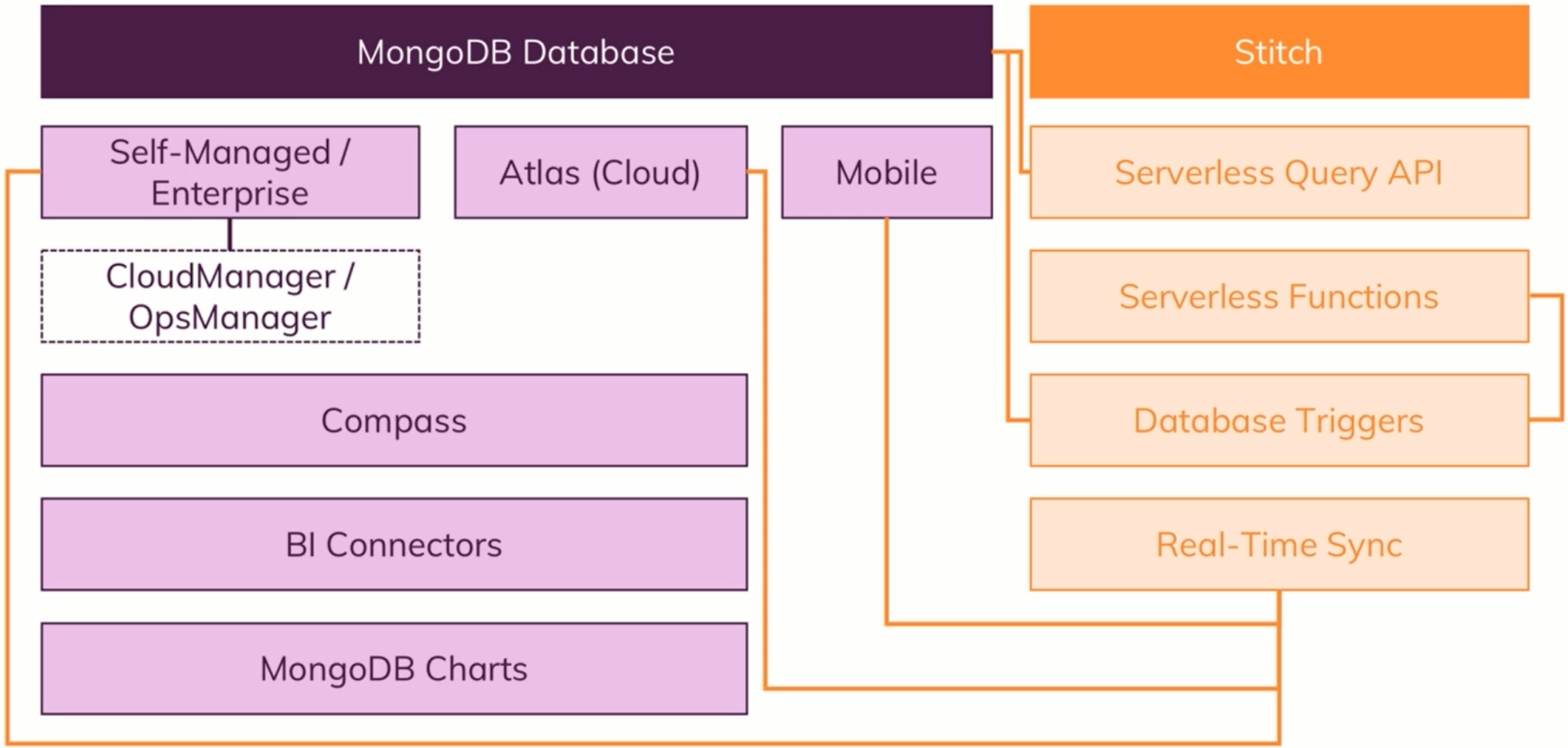
Relations

No / Few Relations!

Relational Data needs to be merged manually

Kind of...

MongoDB Ecosystem



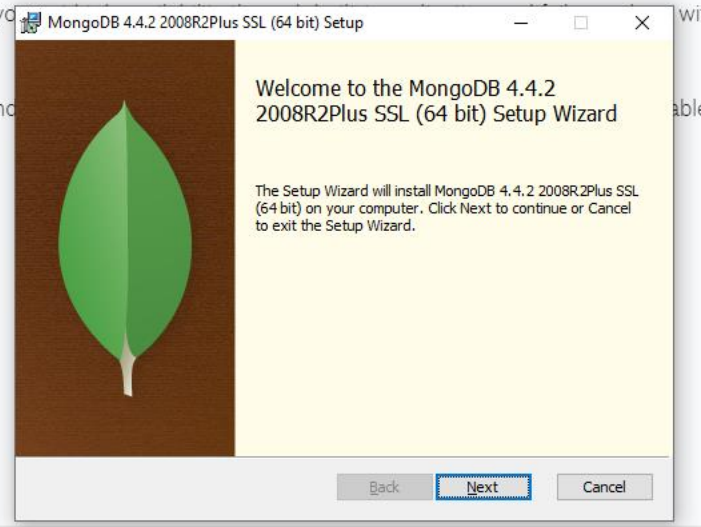
INSTALL MONGO DB

HTTPS://WWW.MONGODB.COM/TRY/DOWNLOAD/COMMUNITY

MongoDB Community Server

MongoDB offers both an Enterprise and Community version of its powerful distributed document database. The community version offers the flexible document model along with ad hoc queries, indexing, and real time aggregation to provide powerful ways to access and analyze your data. As a distributed system yo with horizontal scalability with native sharding.

The MongoDB Enterprise Server gives you all of this and able.



Available Downloads

Version: 4.4.2 (current)
Platform: Windows
Package: msi

Download | Copy Link

Current releases & packages
Development releases
Archived releases
Changelog
Release Notes

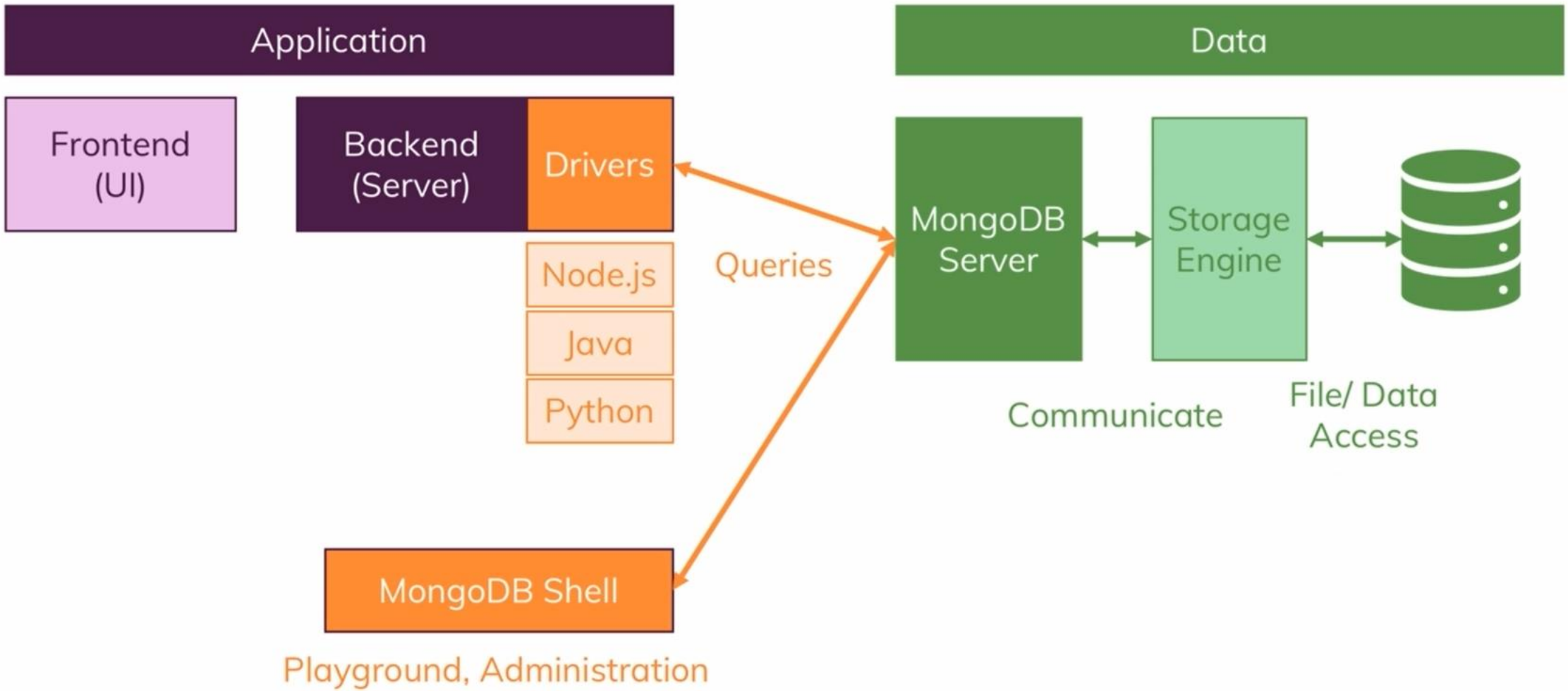
MongoDB Ops Manager

MongoDB Enterprise Kubernetes Operator

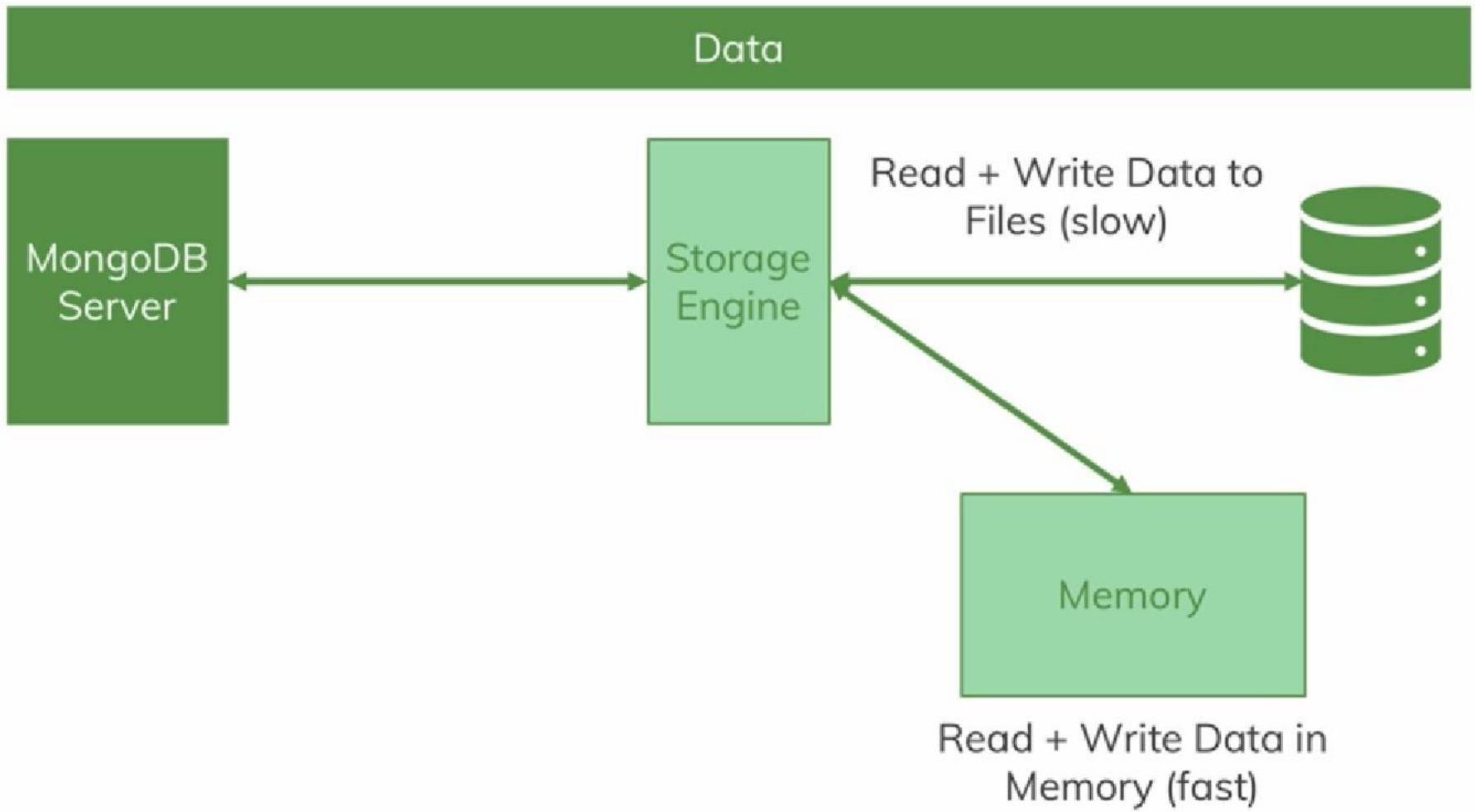
MongoDB Community Kubernetes Operator

MongoDB Charts

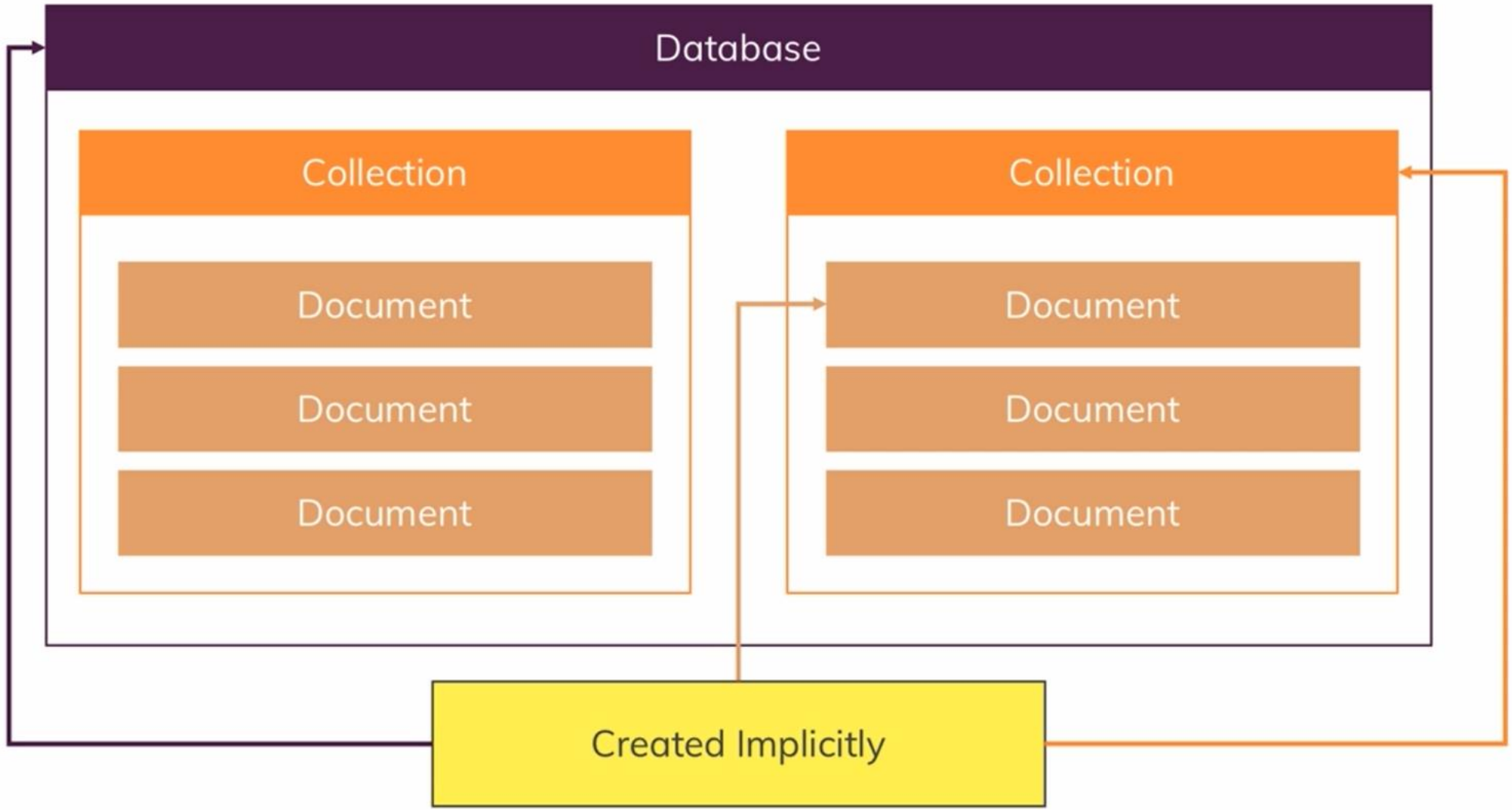
Working with MongoDB



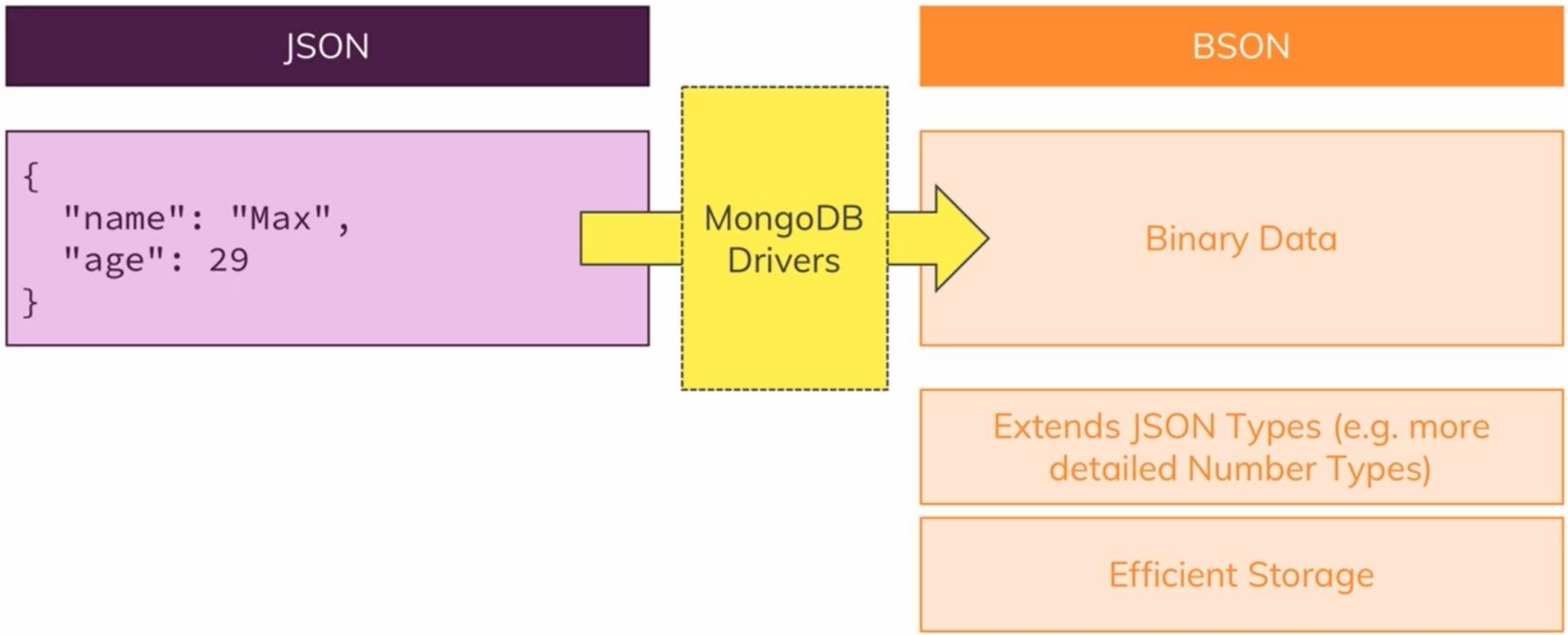
A Closer Look



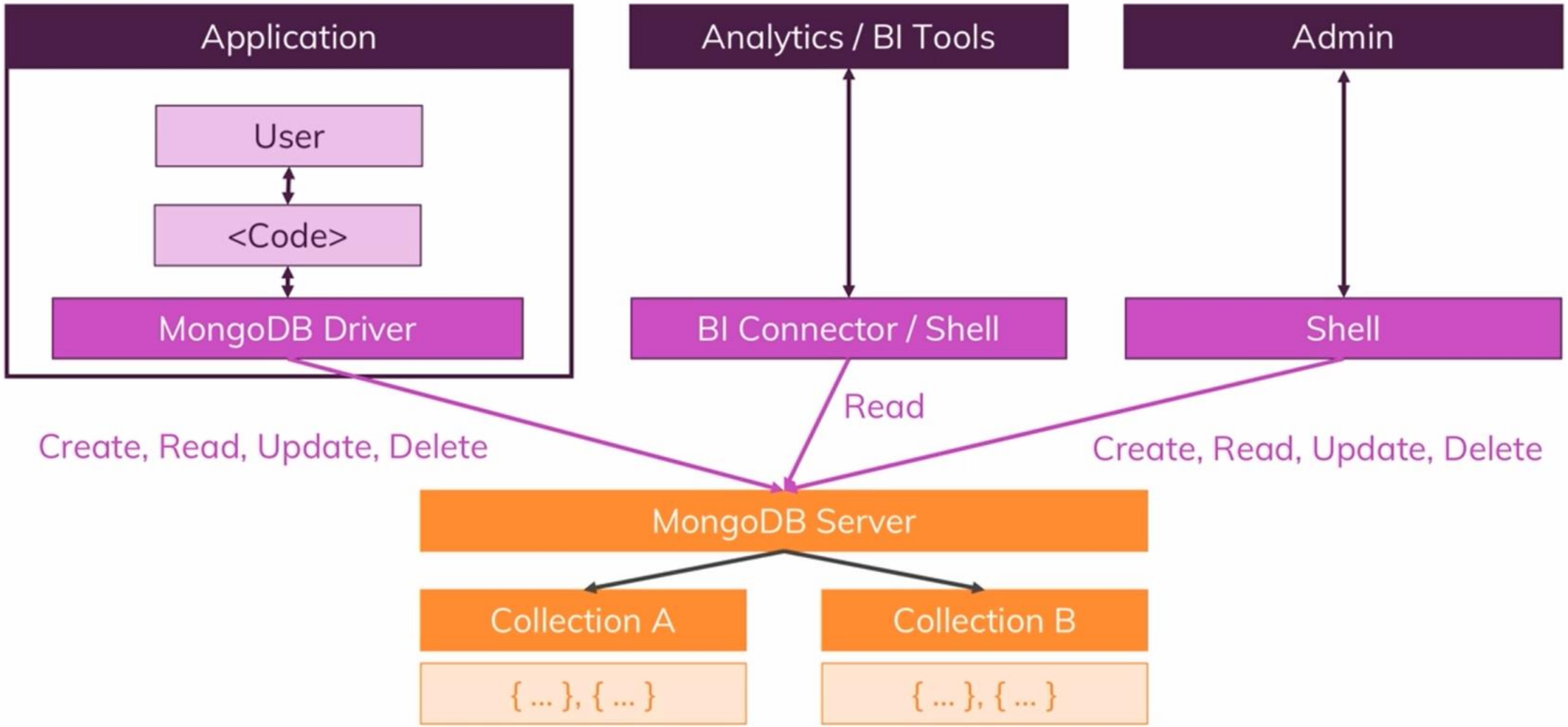
Databases, Collections, Documents



JSON vs BSON


JSON
BSON
{
"name": "Max",
"age": 29
}
MongoDB Drivers
Binary Data
Extends JSON Types (e.g. more detailed Number Types)
Efficient Storage

CRUD Operations & MongoDB



What are CRUD Operations?

CRUD operations refer to the basic Create, Read, Update and Delete operations.

CRUD – Create, Read, Update, Delete

C	Create or Insert	db.collection.insertOne()
R	Read or Find	db.collection.find()
U	Update	db.collection.updateOne()
D	Delete	db.collection.deleteOne()

CRUD Operations

Create

```
insertOne(data, options)
insertMany(data, options)
```

Update

```
updateOne(filter, data, options)
updateMany(filter, data, options)
replaceOne(filter, data, options)
```

Read

```
find(filter, options)
findOne(filter, options)
```

Delete

```
deleteOne(filter, options)
deleteMany(filter, options)
```

Create Operation

db.collection.insertOne()

db.collection.insertMany()

Things to Remember for Insert Behavior

All write operations in MongoDB are atomic on the level of a single document

If the collection does not currently exist, insert operations will create the collection

If an inserted document omits the _id field, the MongoDB driver automatically generates an ObjectId for the _id field

Things to Remember for Find Operation

db.collection.find()

- Query
- Projection
- Read Concern

Query and Query Operators

Find method supports many different query operators to filter data

- Comparison
- Logical
- Element
- Evaluation
- Geospatial
- Array
- Bitwise

Query Projection

Specifies the fields to return in the documents that match the query filter

- 1 or true : include the field
- 0 or false : exclude the field

Read Concern

Allows to control the consistency and isolation properties of the data read from replica sets and replica set shards

- Local
- Available
- Majority
- Linearizable
- Snapshot

Update Operation

db.collection.updateOne()

db.collection.updateMany()

db.collection.replaceOne()

Write Concern

Level of acknowledgement requested from MongoDB for write operations

w:1 – Ack from primary

w:0 – No ack

w:(n) – Primary + (n-1) secondary

w: majority

wtimeout: Time limit to prevent write operations from blocking indefinitely

Things to Remember for Update Behavior

Atomic on the level of a single document

_id field cannot be replaced with different value

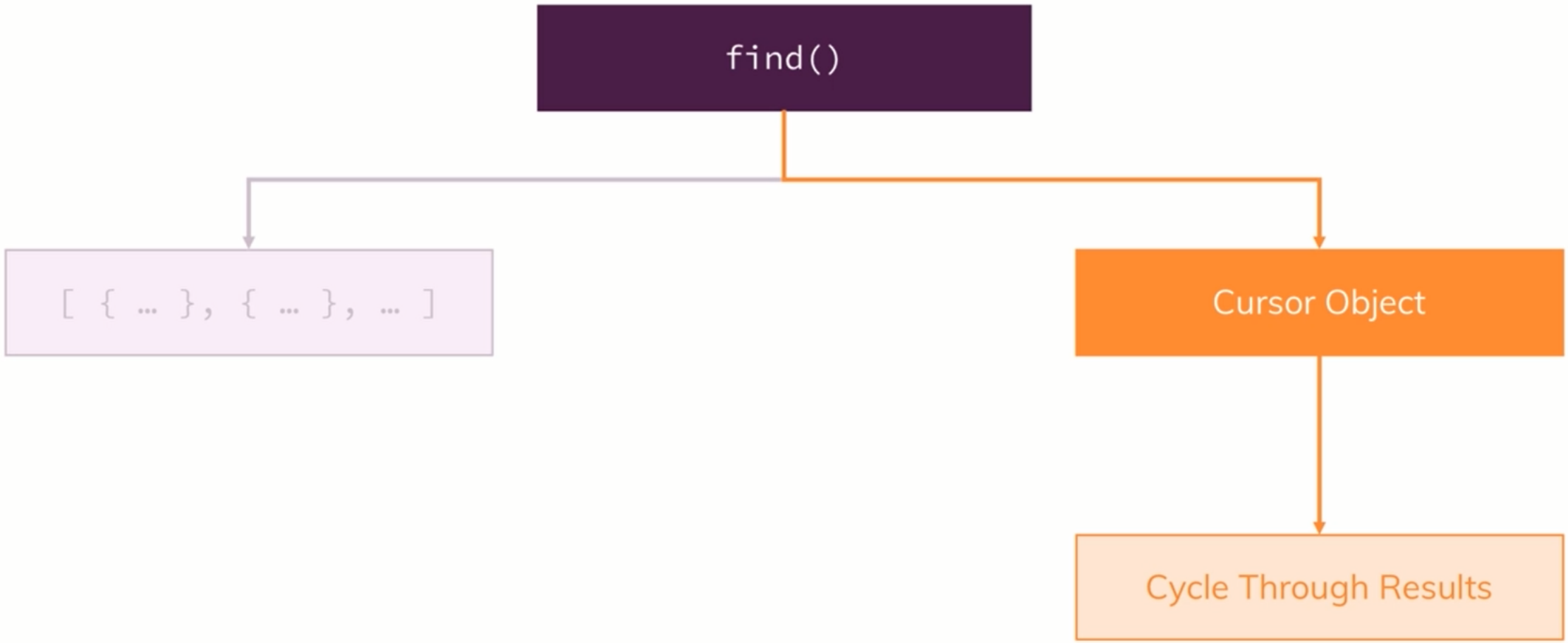
$set creates field if not already existing

upsert : true

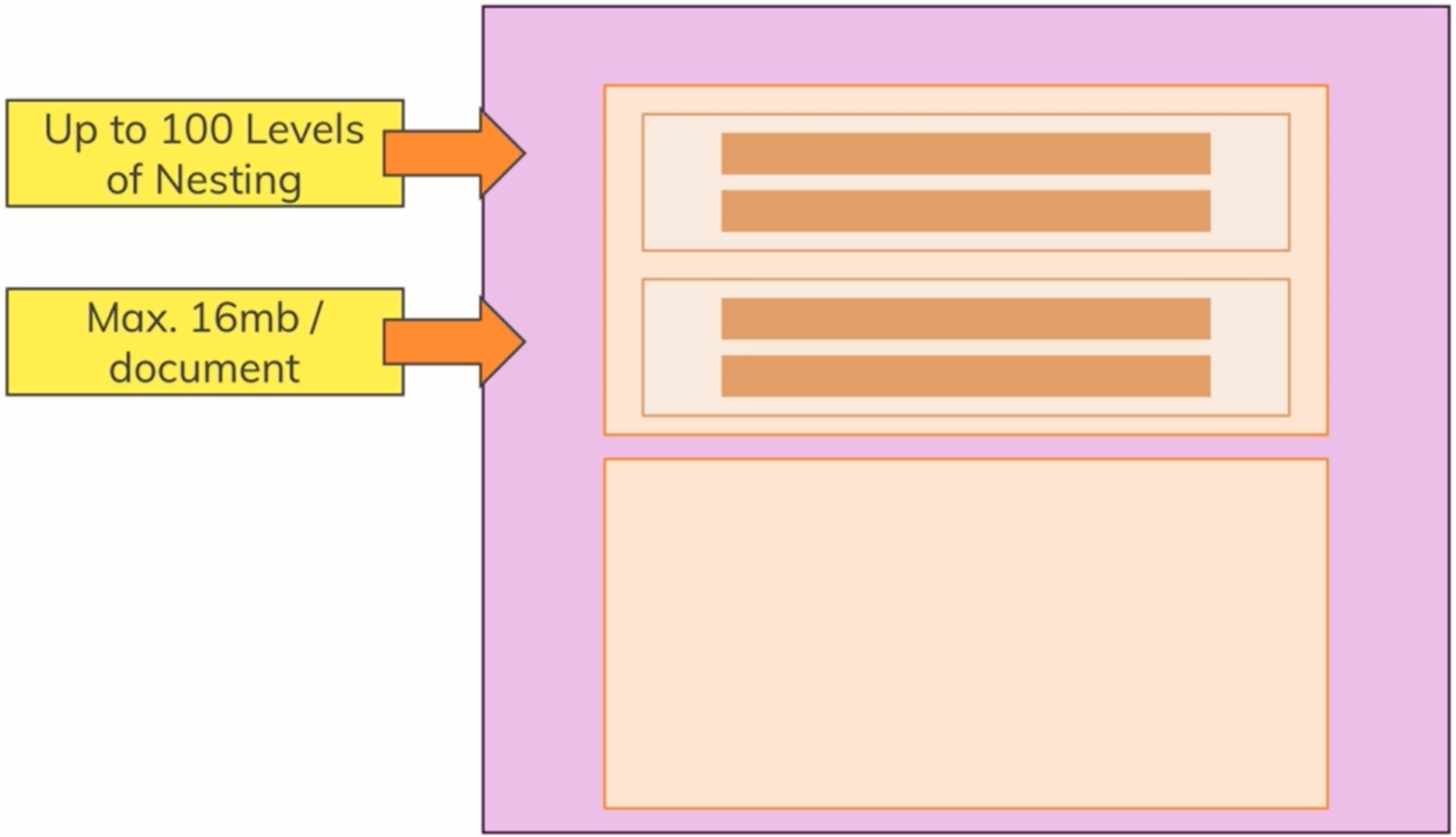
- Update on match of filter
- Insert no match of filter

Example #1: Flight Data

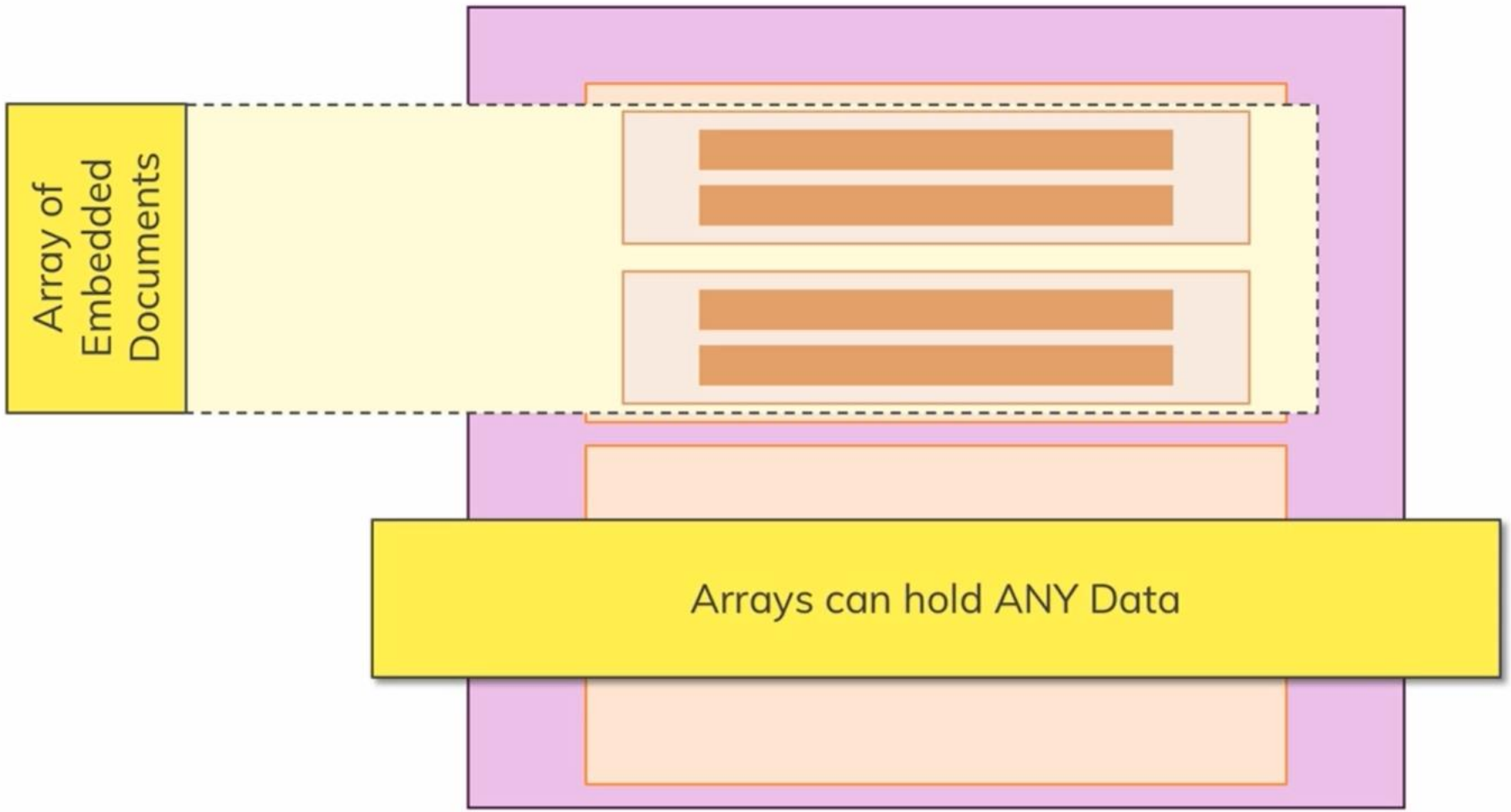
Cursors



Embedded Documents



Arrays



Databases, Collections, Documents

- A Database holds multiple Collections where each Collection can then hold multiple Documents
- Databases and Collections are created "lazily" (i.e. when a Document is inserted)
- A Document can't directly be inserted into a Database, you need to use a Collection!

CRUD Operations

- CRUD = Create, Read, Update, Delete
- MongoDB offers multiple CRUD operations for single-document and bulk actions (e.g. insertOne(), insertMany(), ...)
- Some methods require an argument (e.g. insertOne()), others don't (e.g. find())
- find() returns a cursor, NOT a list of documents!
- Use filters to find specific documents

Document Structure

- Each document needs a unique ID (and gets one by default)
- You may have embedded documents and array fields

Retrieving Data

- Use filters and operators (e.g. $gt) to limit the number of documents you retrieve

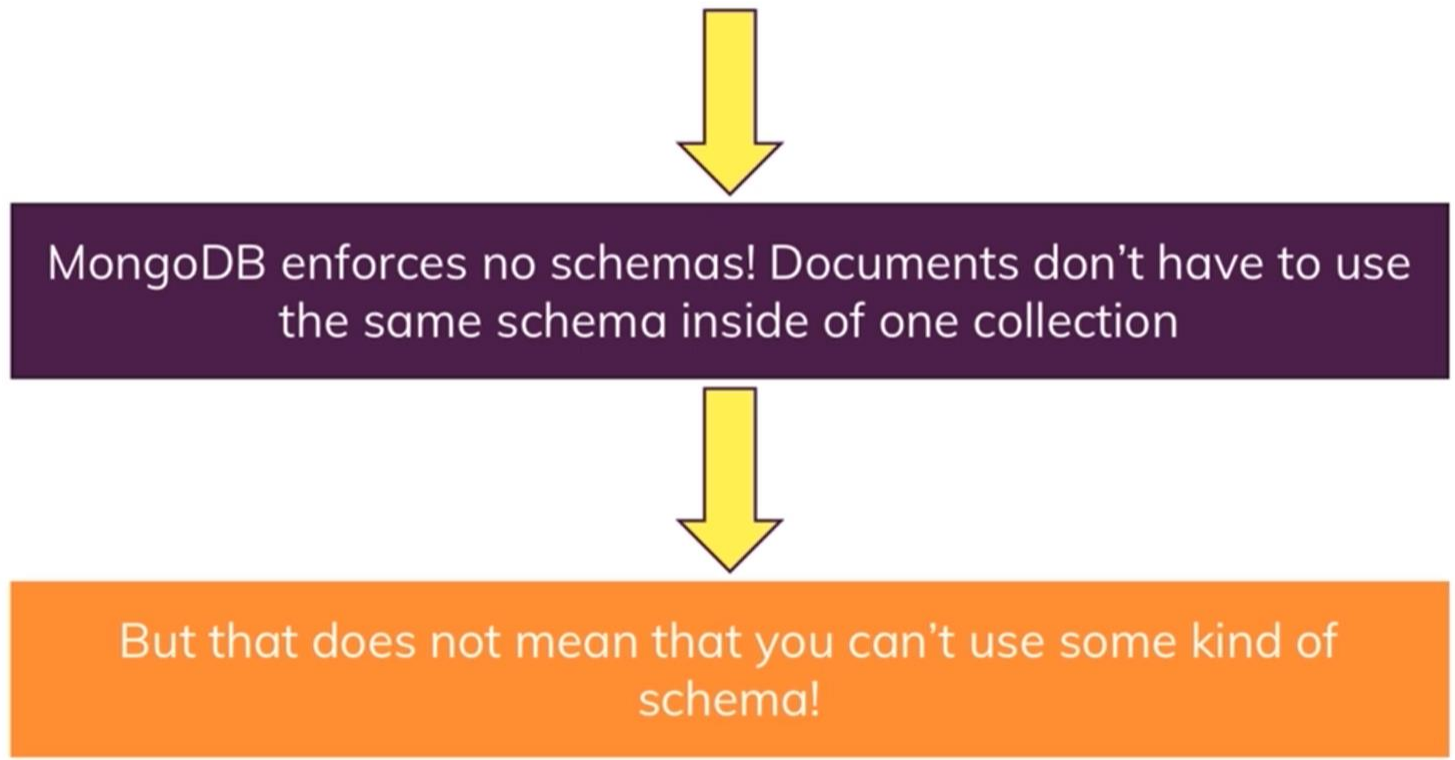
Understanding Document Schemas & Data Types

Modelling Relations

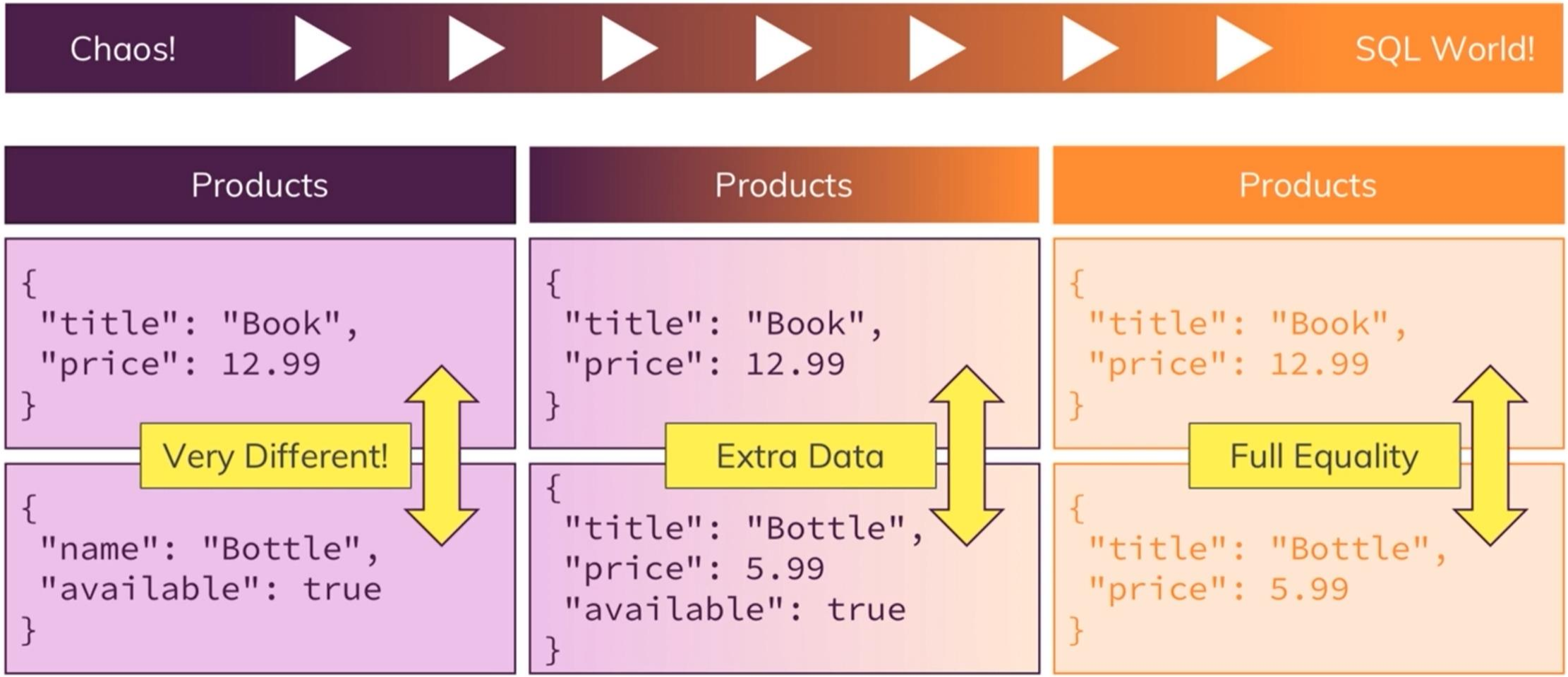
Schema Validation

Schema-less Or Not?

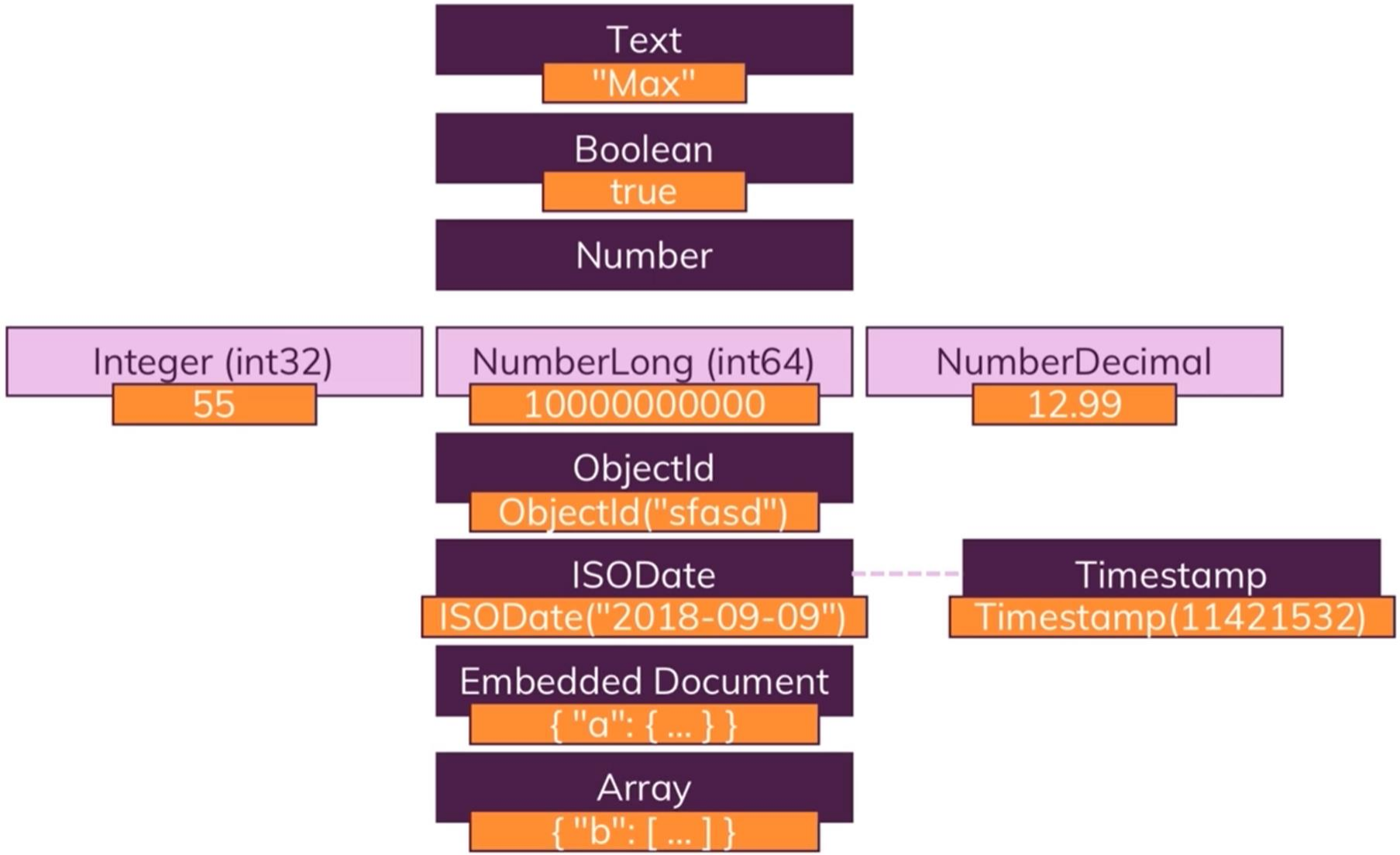
Isn't MongoDB all about having **NO** data Schemas?



To Schema Or Not To Schema



Data Types


Text
"Max"
Boolean
true
Number
Integer (int32)
55
NumberLong (int64)
10000000000
NumberDecimal
12.99
ObjectId
ObjectId("sfasd")
ISODate
ISODate("2018-09-09")
Timestamp
Timestamp(11421532)
Embedded Document
{ "a": { ... } }
Array
{ "b": [...] }

Data Schemas & Data Modelling

Which Data does my App need or generate?	User Information, Product Information, Orders, ...	Defines the Fields you'll need (and how they relate)
Where do I need my Data?	Welcome Page, Products List Page, Orders Page	Defines your required collections + field groupings
Which kind of Data or Information do I want to display?	Welcome Page: Product Names; Products Page: ...	Defines which queries you'll need
How often do I fetch my data?	For every page reload	Defines whether you should optimize for easy fetching
How often do I write or change my data?	Orders => Often Product Data => Rarely	Defines whether you should optimize for easy writing

Relations - Options

Nested / Embedded Documents

Customers

```
{
  userName: 'max',
  age: 29,
  address: {
    street: 'Second Street',
    city: 'New York'
  }
}
```

References

Lots of data duplication!

```
{
  user
  favBooks: [{…}, {…}]
}
```

omers

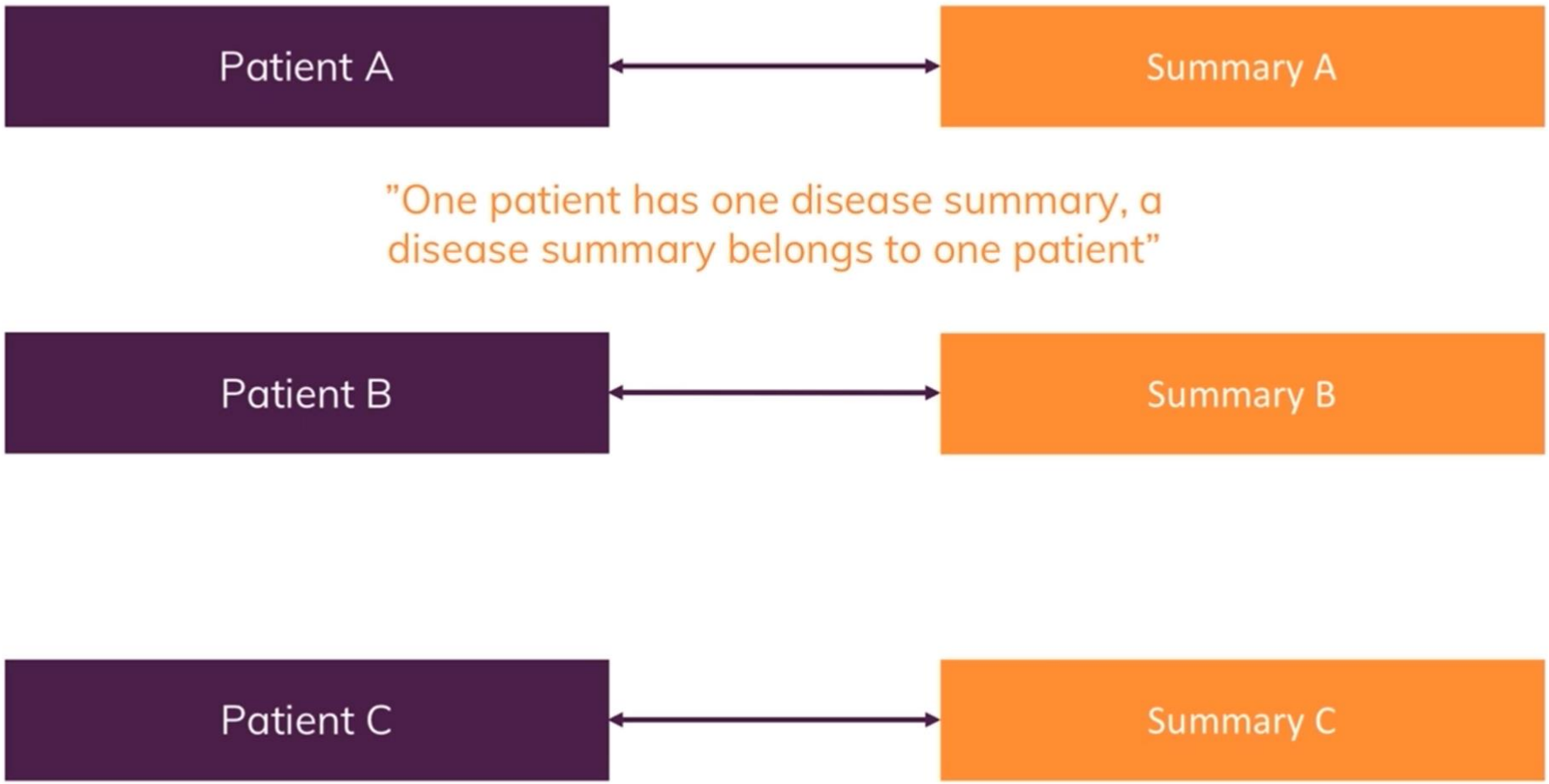
Customers

```
{
  userName: 'max',
  favBooks: ['id1', 'id2']
}
```

Books

```
{
  _id: 'id1',
  name: 'Lord of the Rings 1'
}
```

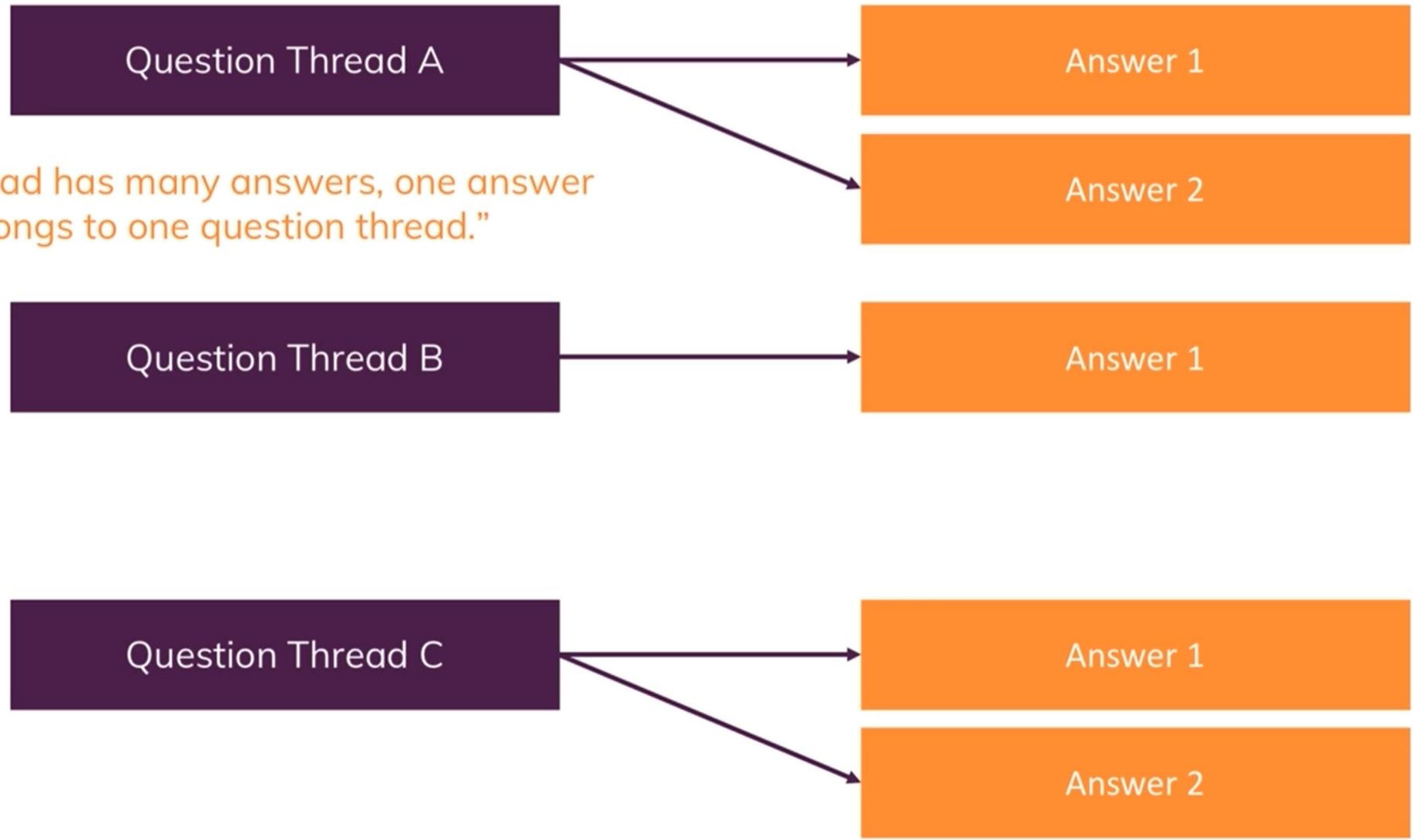
Example #1 – Patient <-> Disease Summary



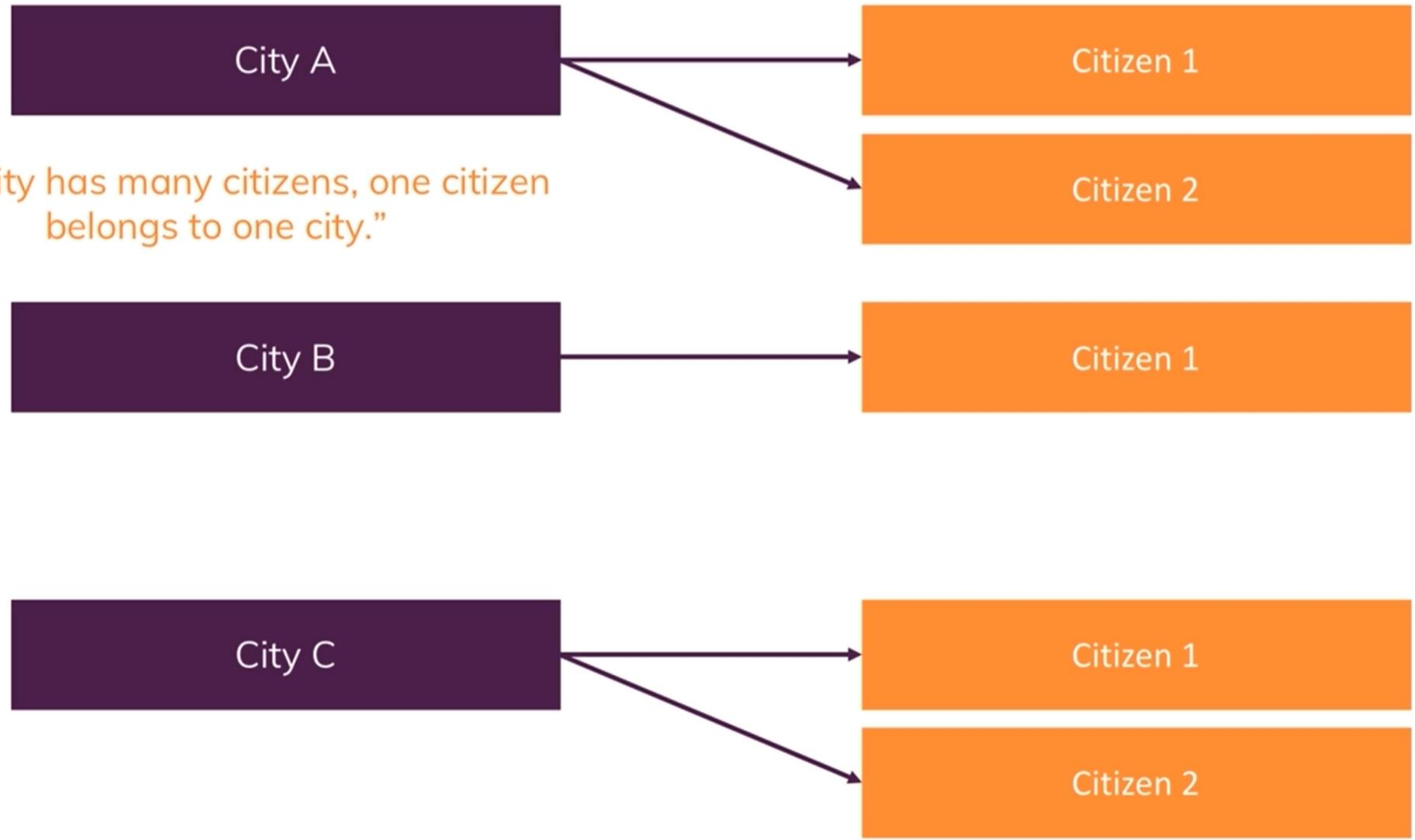
Example #2 – Person <-> Car

"One person has one car, a car belongs to one person"

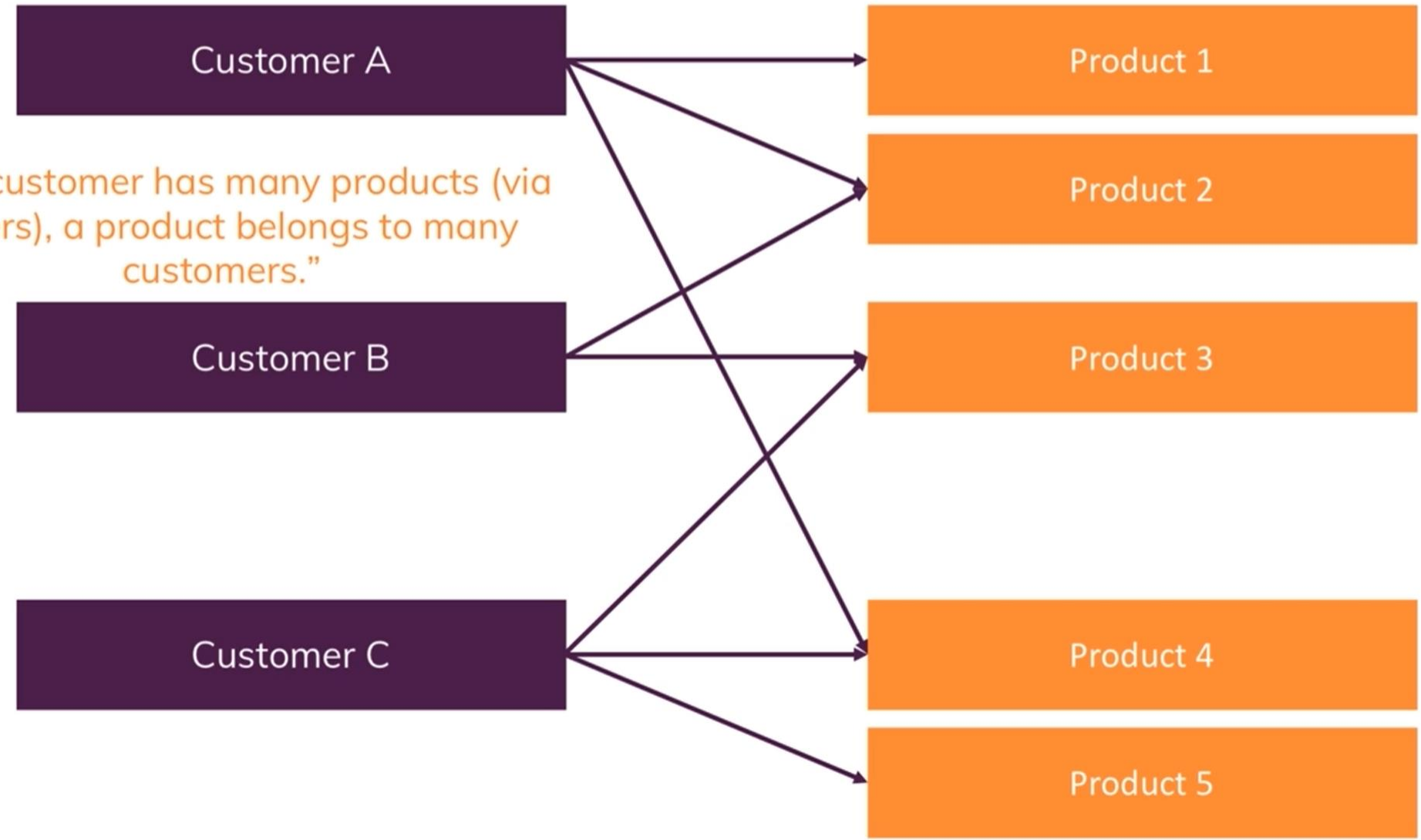
Example #3 – Thread <-> Answers



Example #4 – City <-> Citizens

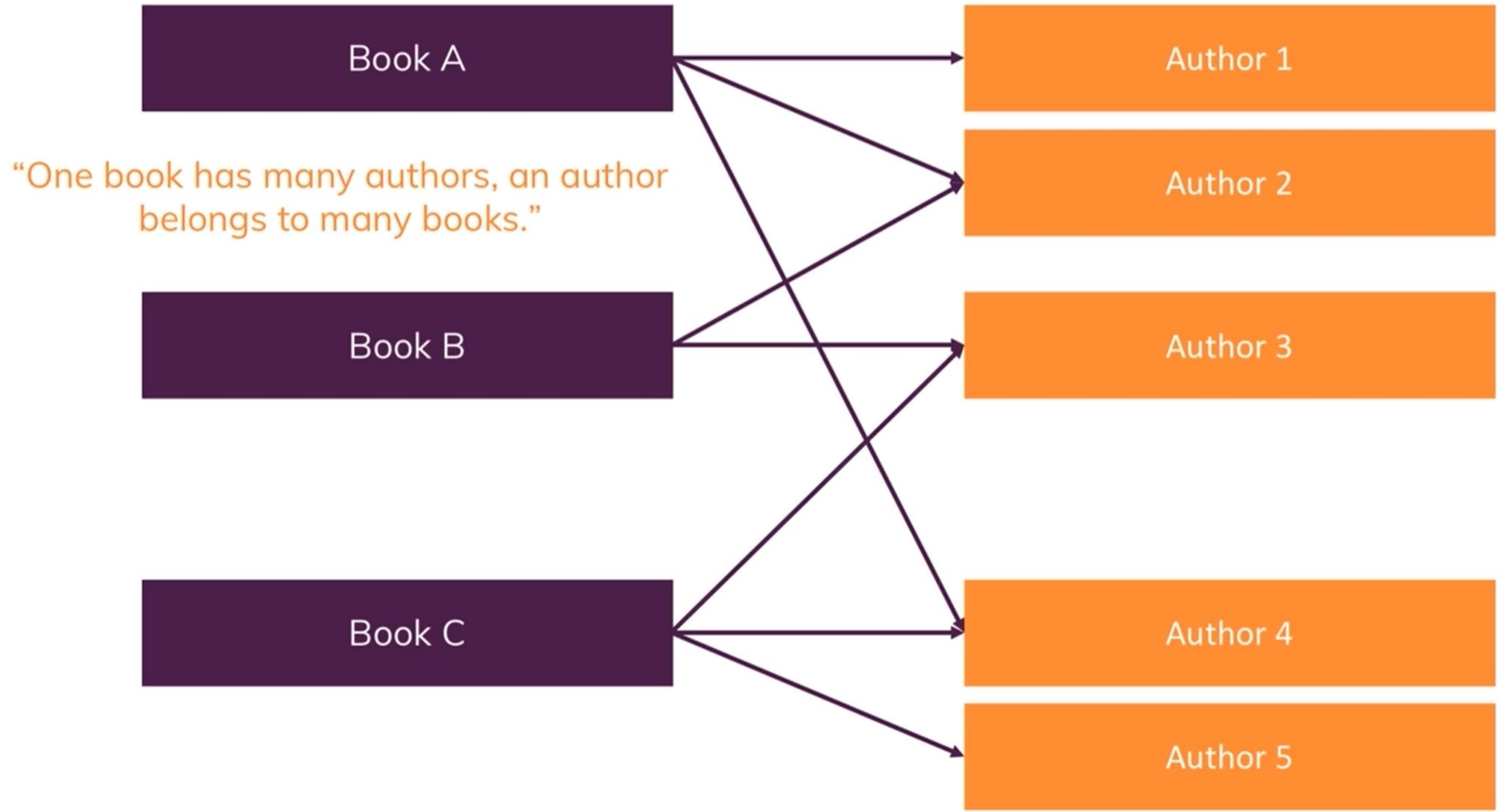


Example #5 – Customers <-> Products (Orders)



"One customer has many products (via orders), a product belongs to many customers."

Example #6 – Books <-> Authors

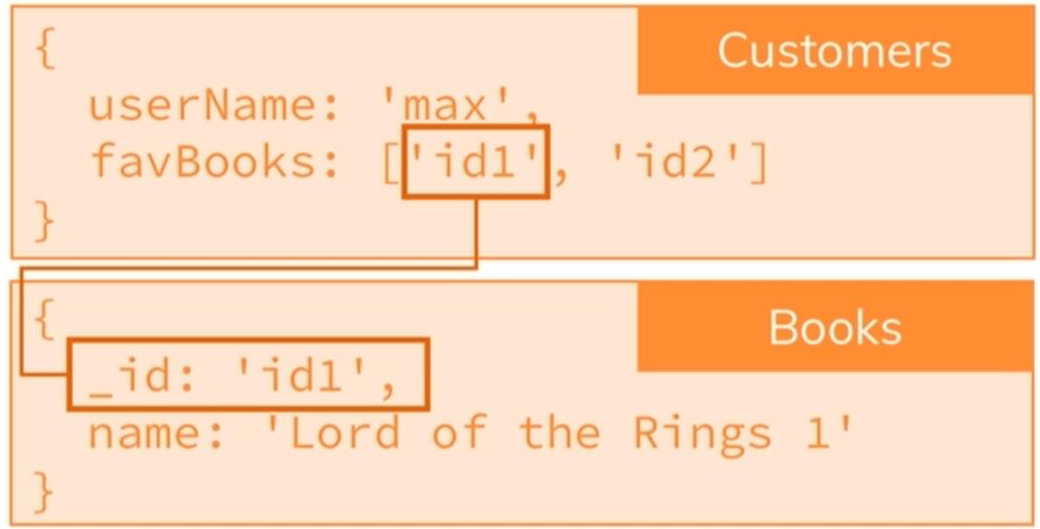


"One book has many authors, an author belongs to many books."

Relations - Options

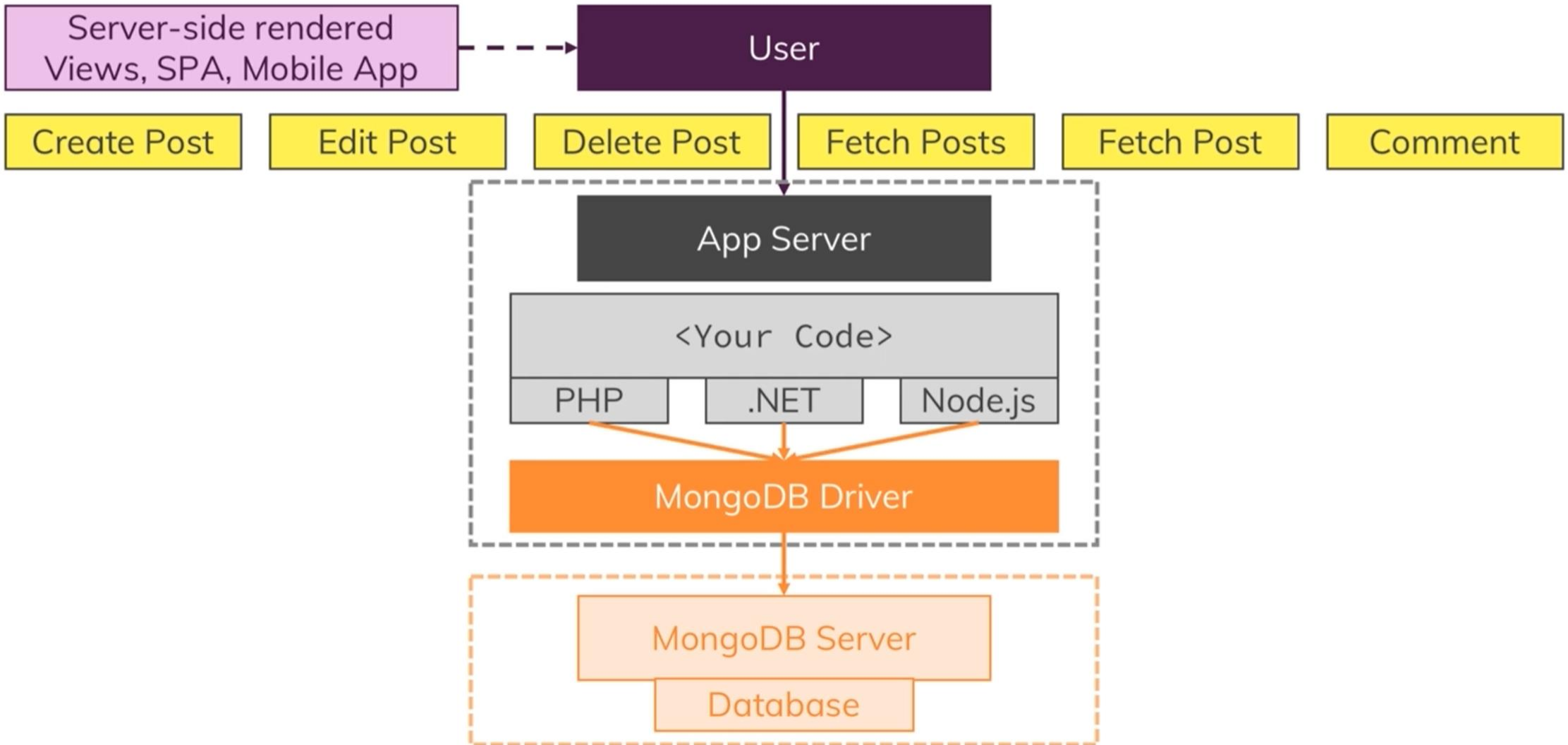
Nested / Embedded Documents	References
Group data together logically	Split data across collections
Great for data that belongs together and is not really overlapping with other data	Great for related but shared data as well as for data which is used in relations and standalone
Avoid super-deep nesting (100+ levels) or extremely long arrays (16mb size limit per document)	Allows you to overcome nesting and size limits (by creating new documents)

Joining with $lookup

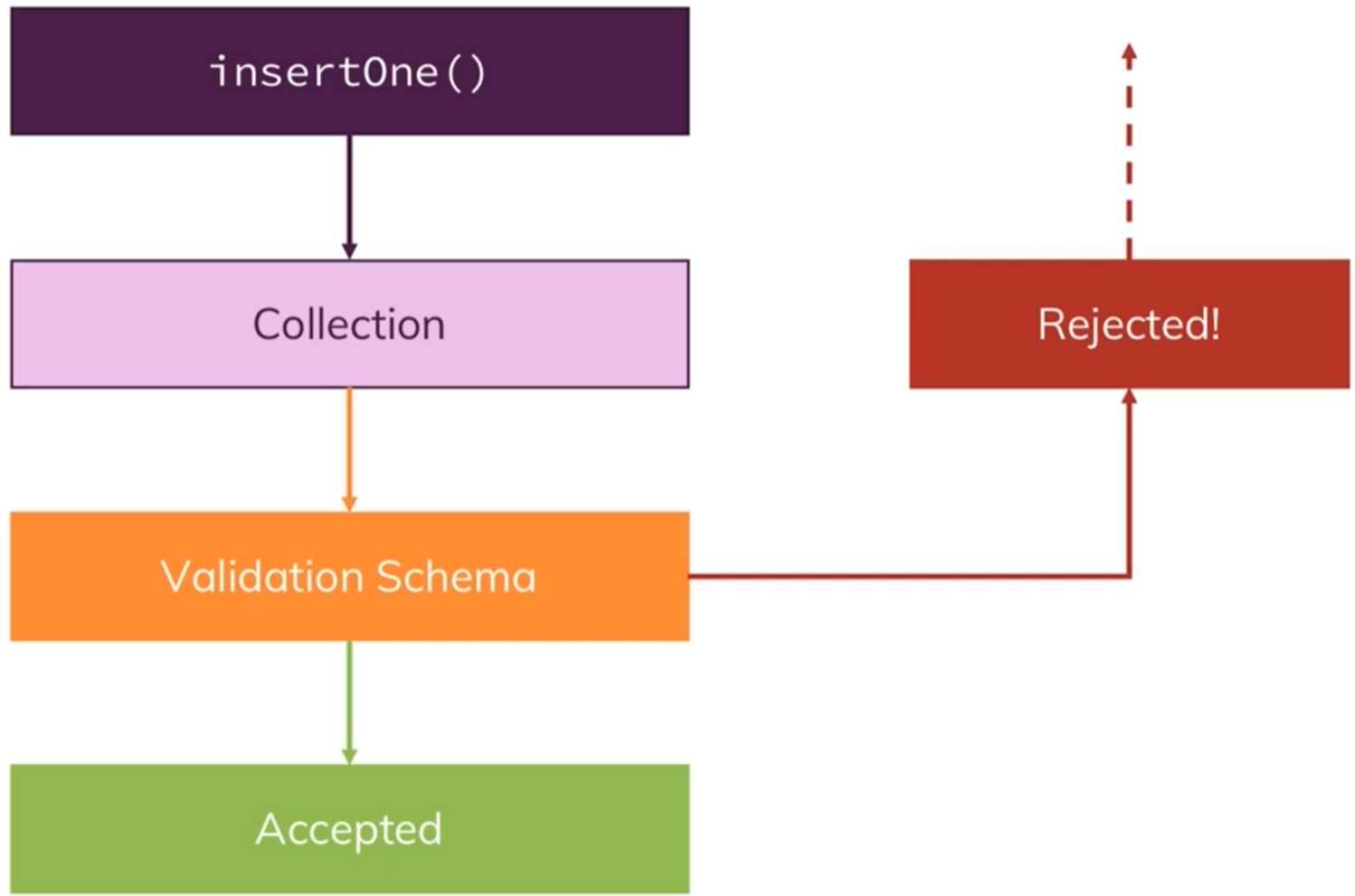



```
customers.aggregate([
  { $lookup: {
      from: "books",
      localField: "favBooks",
      foreignField: "_id"
      as: "favBookData"
    }
  }
])
```

Example Project: A Blog



Schema Validation



Schema Validation

`validationLevel`	`validationAction`
Which documents get validated?	What happens if validation fails?
strict → All inserts & updates	error → Throw error and deny insert/ update
moderate → All inserts & updates to correct documents	warn → Log warning but proceed

Data Modelling & Structuring – Things to Consider

In which Format will you fetch your Data?

How often will you fetch and change your Data?

How much data will you save (and how big is it)?

How is your Data related?

Will Duplicates hurt you (=> many Updates)?

Will you hit Data/ Storage Limits?

Modelling Schemas

- Schemas should be modelled based on your application needs
- Important factors are: Read and write frequency, relations, amount (and size) of data

Schema Validation

- You can define rules to validate inserts and update before writing to the database
- Choose your validation level and action based on your application requirements

Modelling Relations

- Two options: Embedded documents or references
- Use embedded documents if you got one-to-one or one-to-many relationships and no app or data size reason to split
- Use references if data amount/ size or application needs require it or for many-to-many relations
- Exceptions are always possible => Keep your app requirements in mind!

Working with Shell & Server

Beyond Start & Stop

What's Inside This Module?

- Start MongoDB Server as Process & Service
- Configuring Database & Log Path (and Mode)
- Fixing Issues

MongoDB Compass

Exploring the Data Visually

Diving Deeper Into CREATE

A Closer Look at Creating & Importing Documents

What's Inside This Module?

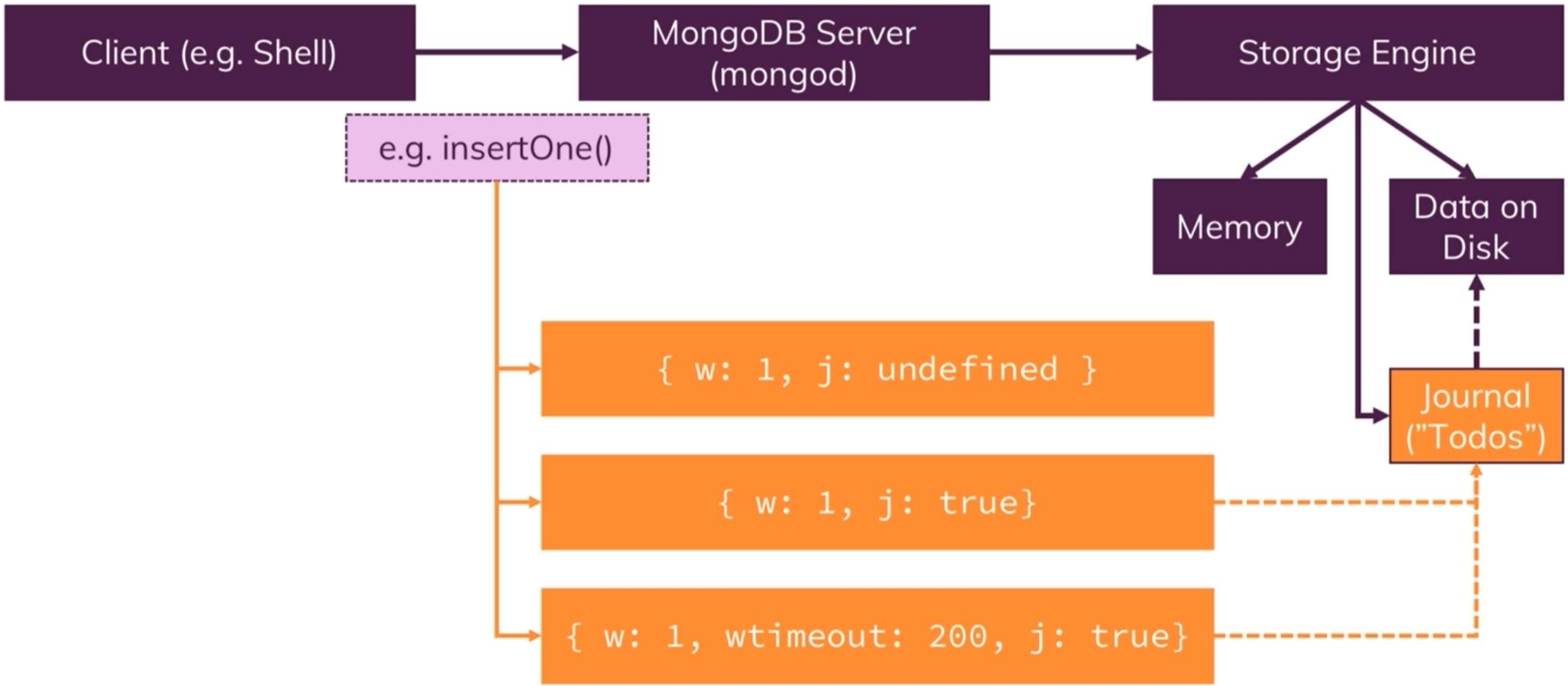
Document Creation Methods (CREATE)

Importing Documents

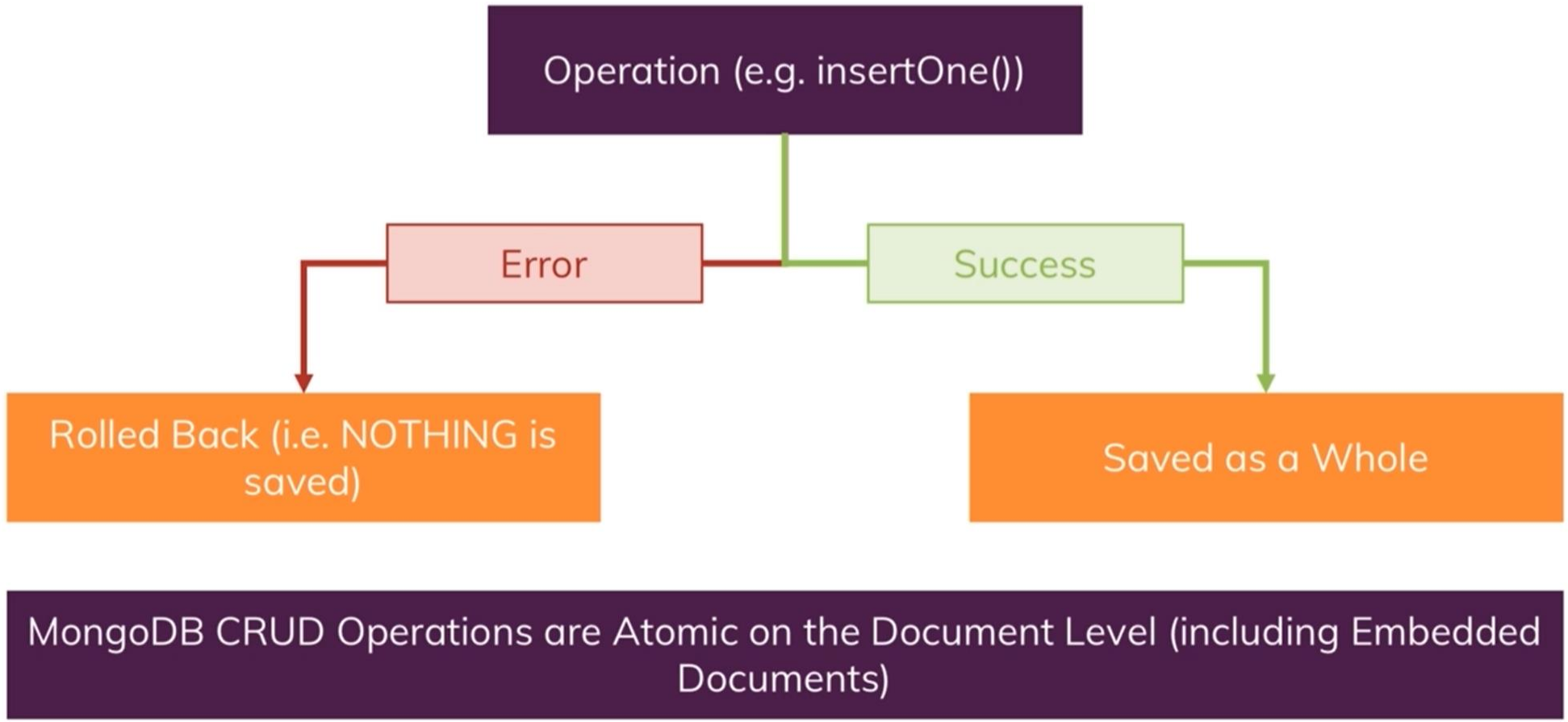
CREATE Documents

insertOne()	`db.collectionName.insertOne({field: "value"})`
insertMany()	`db.collectionName.insertMany([ {field: "value"}, {field: "value"}])`
insert()	`db.collectionName.insert()`
mongoimport	`mongoimport -d cars -c carsList --drop --jsonArray`

WriteConcern



What is “Atomicity”?



Module Summary

insertOne(), insertMany()

- You can insert documents with insertOne() (one document at a time) or insertMany() (multiple documents)
- insert() also exists but it's not recommended to use it anymore – it also doesn't return the inserted ids

WriteConcern

- Data should be stored and you can control the "level of guarantee" of that to happen with the writeConcern option
- Choose the option value based on your app requirements

Ordered Insertes

- By default, when using insertMany(), inserts are ordered – that means, that the inserting process stops if an error occurs
- You can change this by switching to "unordered inserts" – your inserting process will then continue, even if errors occurred
- In both cases, no successful inserts (before the error) will be rolled back

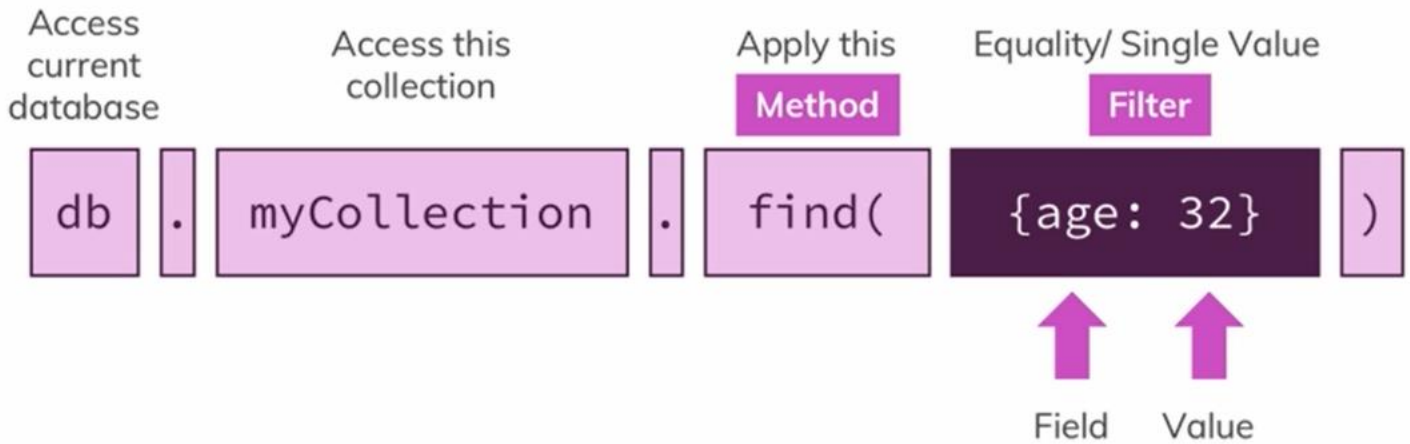
READing Documents with Operators

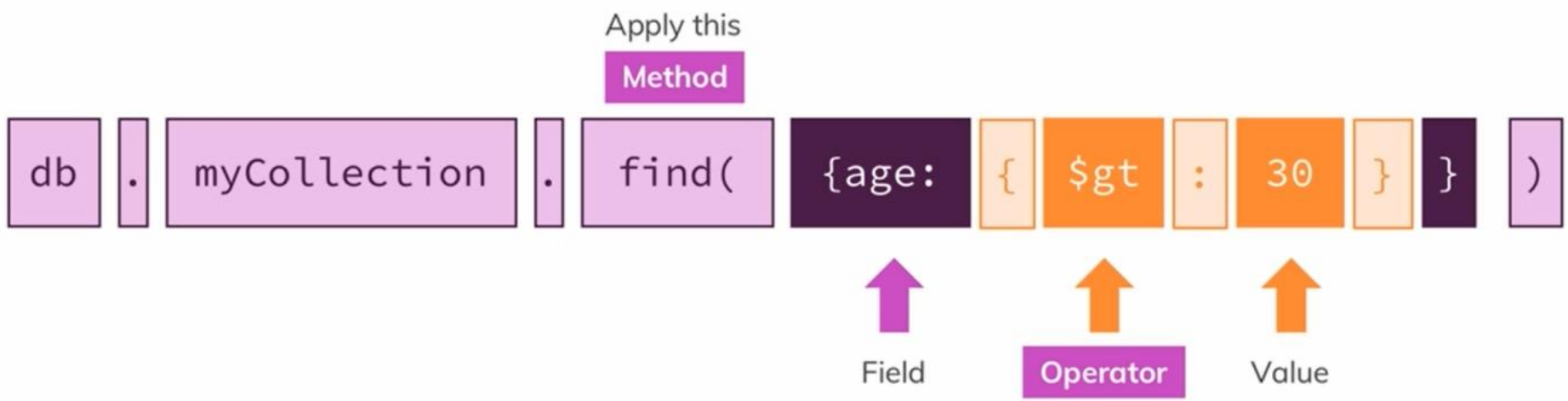
Accessing the Required Data Efficiently

What's Inside This Module?

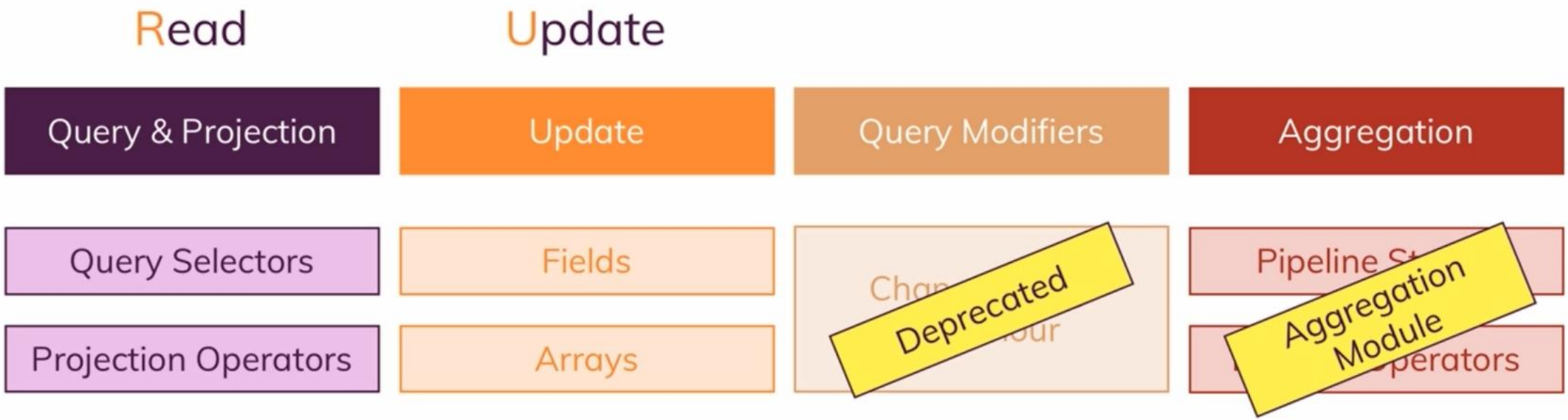
- Methods, Filters & Operators
- Query Selectors (READ)
- Projection Operators (READ)

Methods, Filters & Operators





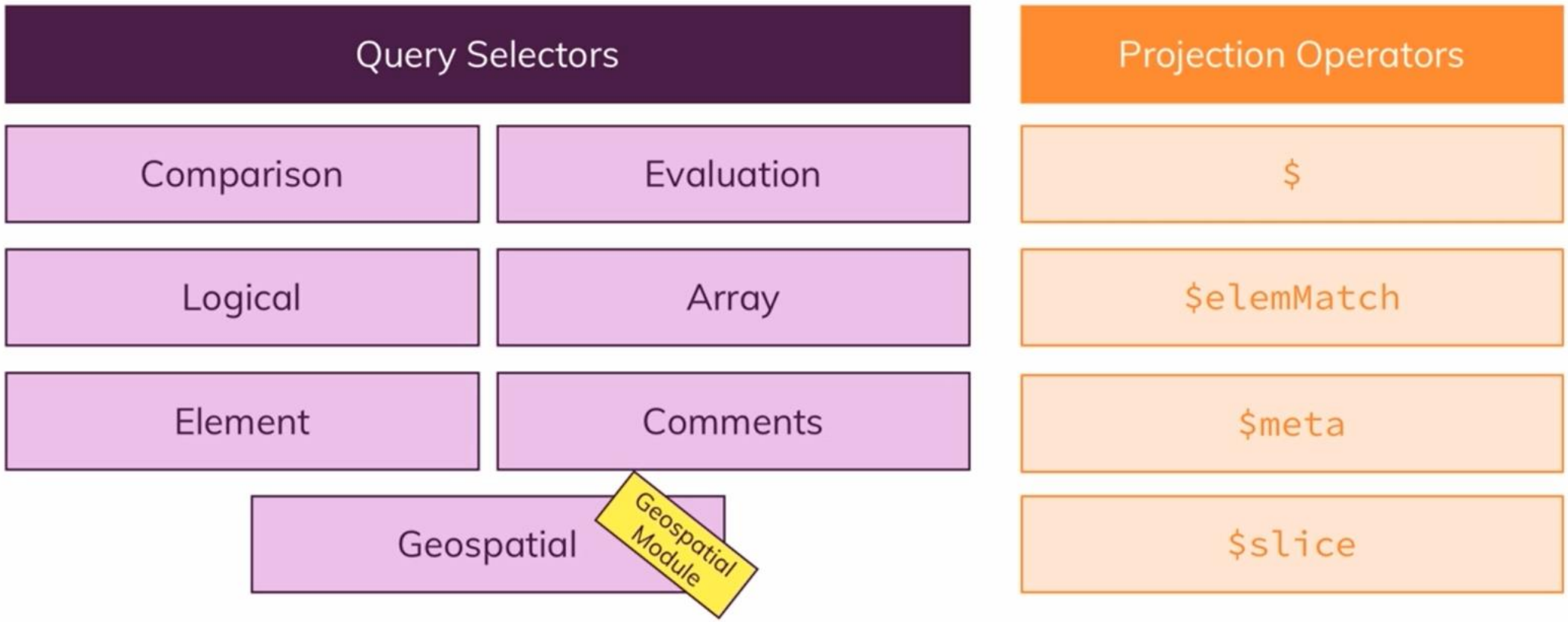
Operators


Read
Update
Query & Projection
Update
Query Modifiers
Aggregation
Query Selectors
Projection Operators
Fields
Arrays
Deprecated
Pipeline St
Aggregation Module

How Operators Impact our Data

Type	Purpose	Changes Data?	Example
Query Operator	Locate Data		$eq
Projection Operator	Modify data presentation		$
Update Operator	Modify + add additional data		$inc

Query Selectors & Projection Operators



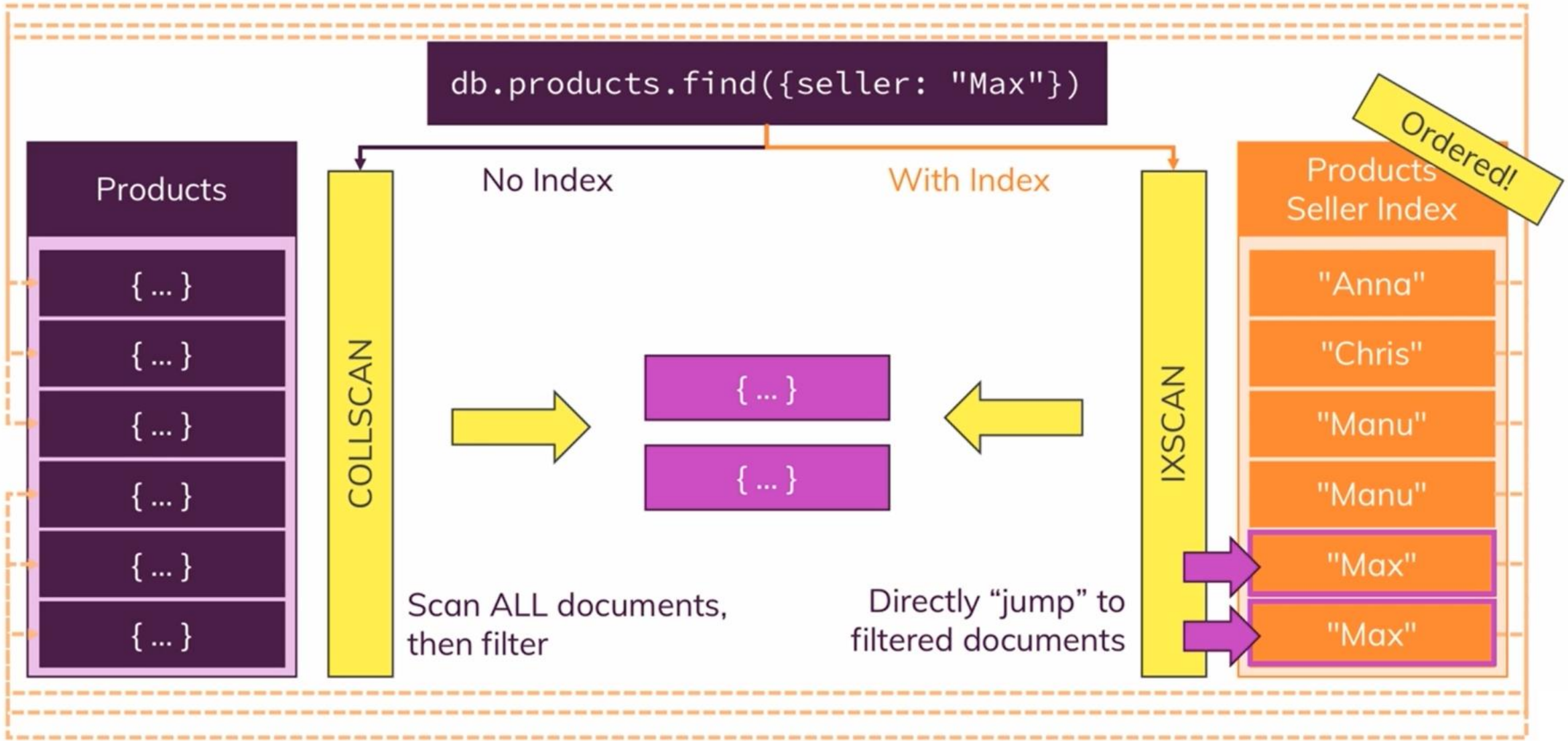
Indexes

Retrieving Data Efficiently

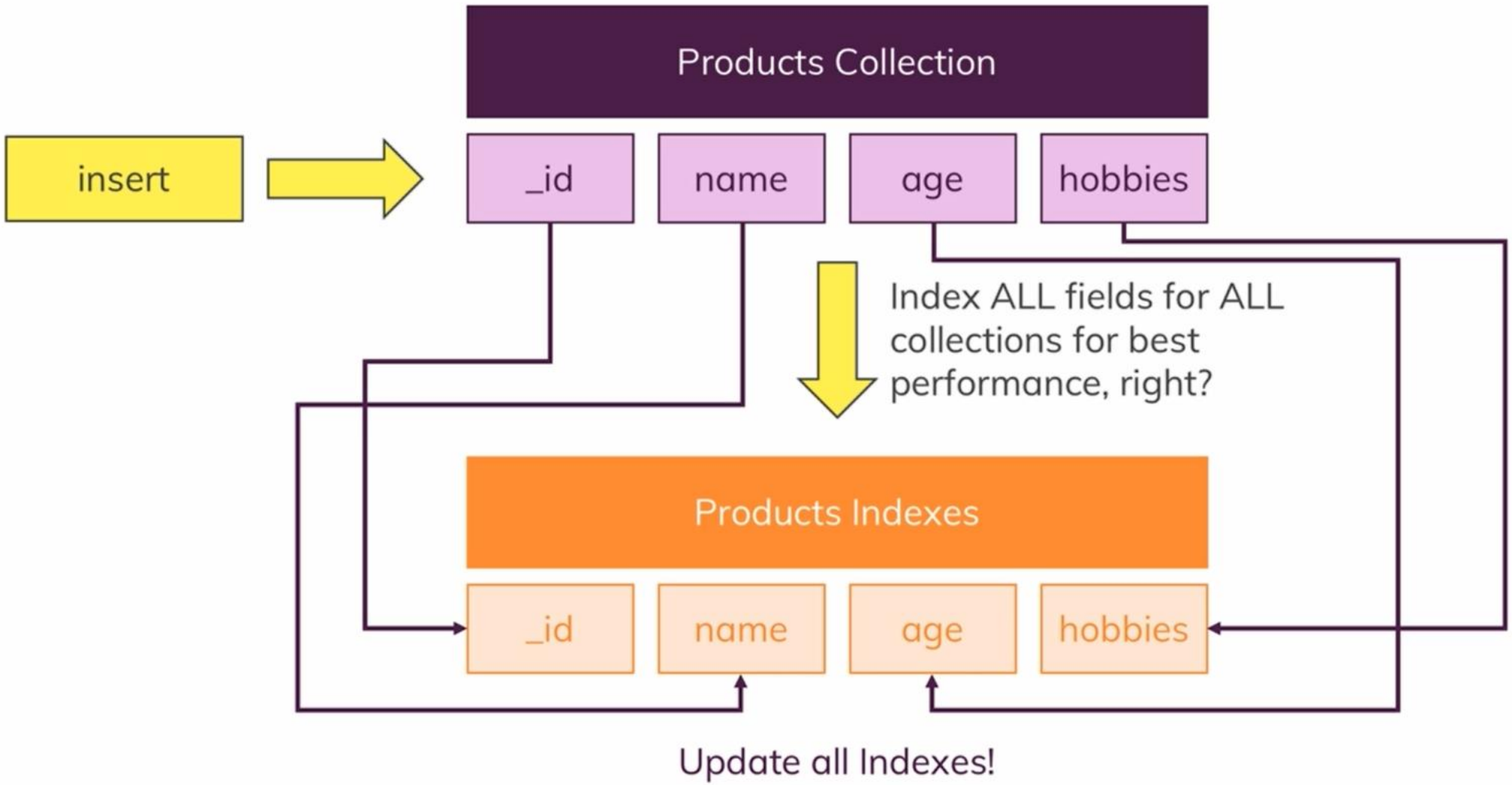
What's Inside This Module?

- What are Indexes?
- Different Types of Indexes
- Using & Optimizing Indexes

Why Indexes?


db.products.find({seller: "Max"})
No Index
With Index
Products
{ ... }
{ ... }
{ ... }
{ ... }
{ ... }
{ ... }
COLLSCAN
Scan ALL documents, then filter
{ ... }
{ ... }
IXSCAN
Directly "jump" to filtered documents
Products Seller Index
Ordered!
"Anna"
"Chris"
"Manu"
"Manu"
"Max"
"Max"

Don’t Use Too Many Indexes!



Geospatial Queries

Finding Places

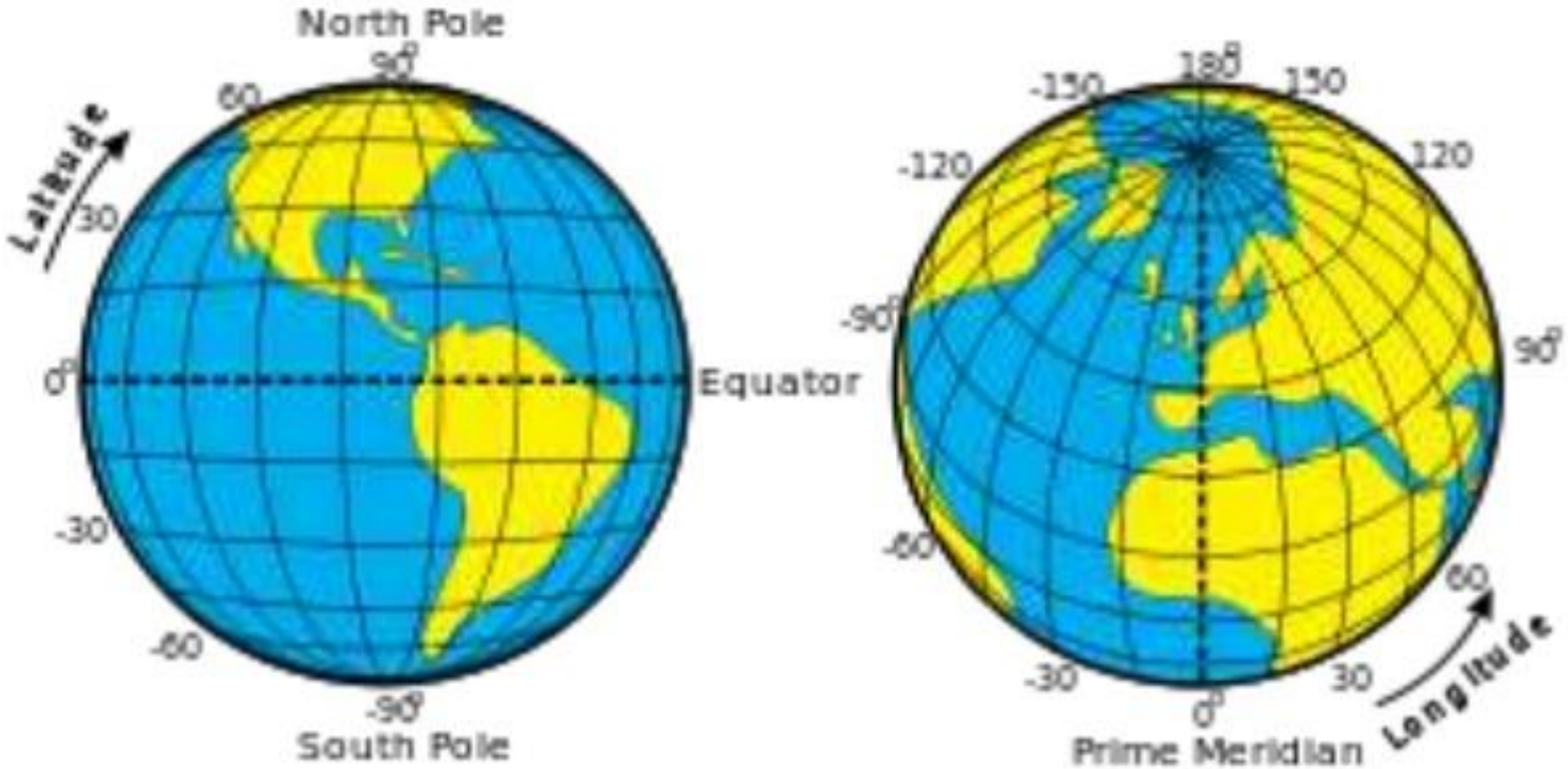
What's Inside This Module?

Storing Geospatial Data in GeoJSON Format

Querying Geospatial Data

Horizontal Lines
Latitude | **Parallels**

Vertical Lines
Longitude | **Meridians**

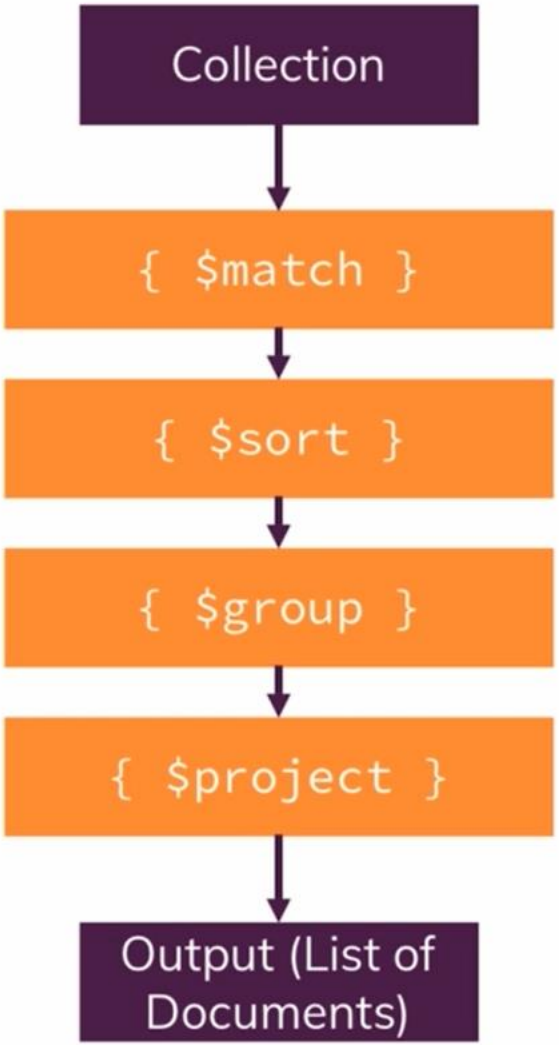


Using the Aggregation Framework

Retrieving Data Efficiently & In a Structured Way

What is the “Aggregation Framework”?

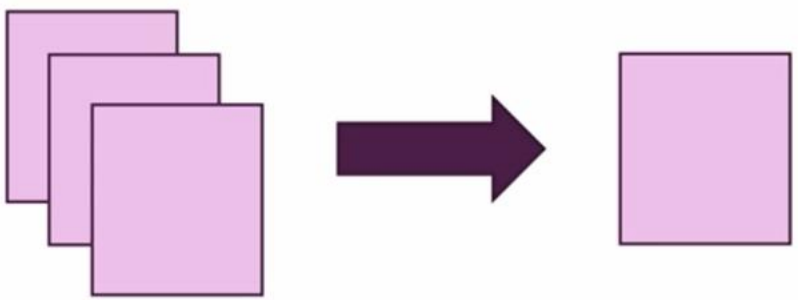
Every stage receives the output of the previous stage



$group vs $project

$group

n:1

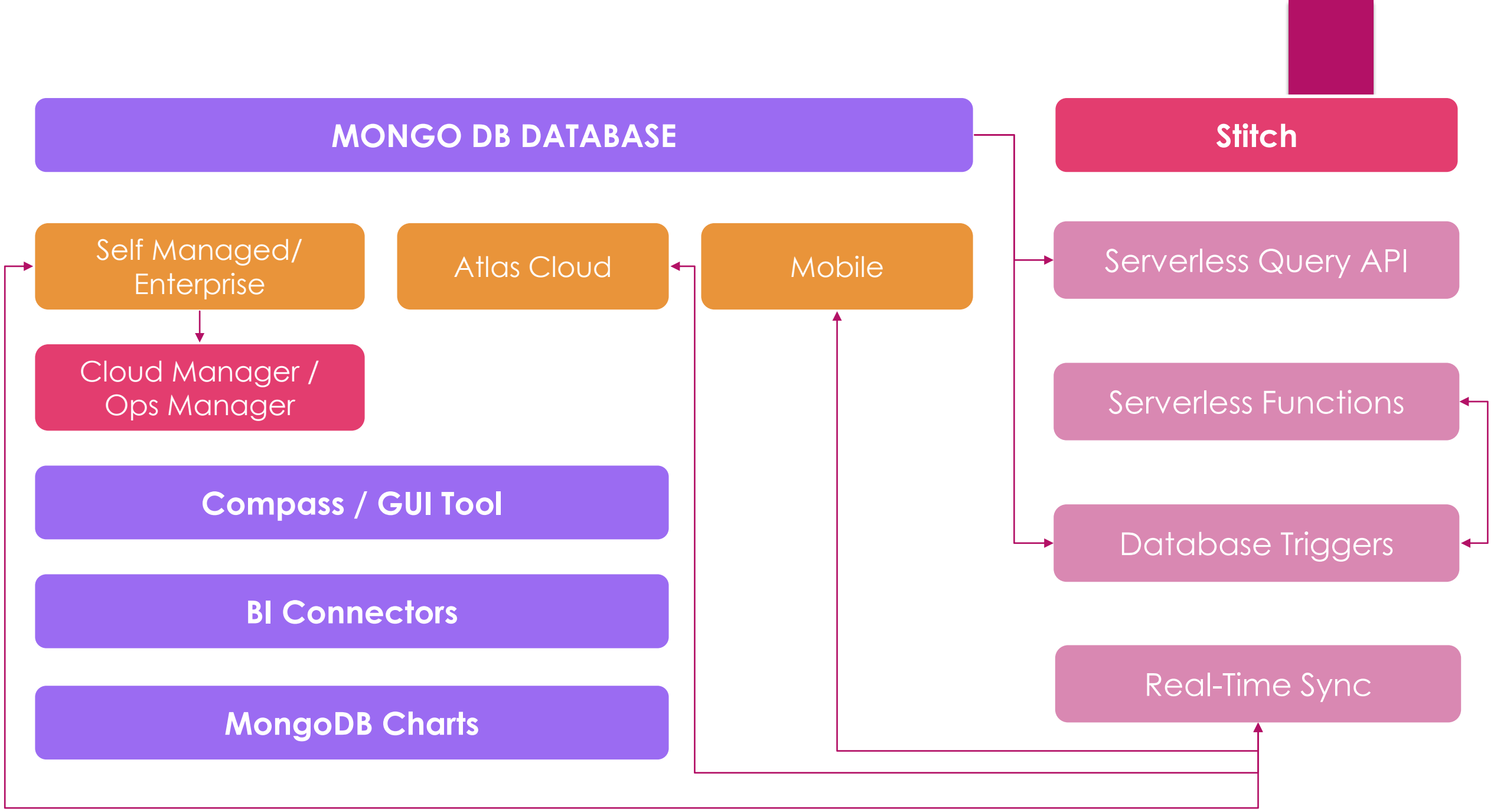


Sum, Count, Average, Build Array

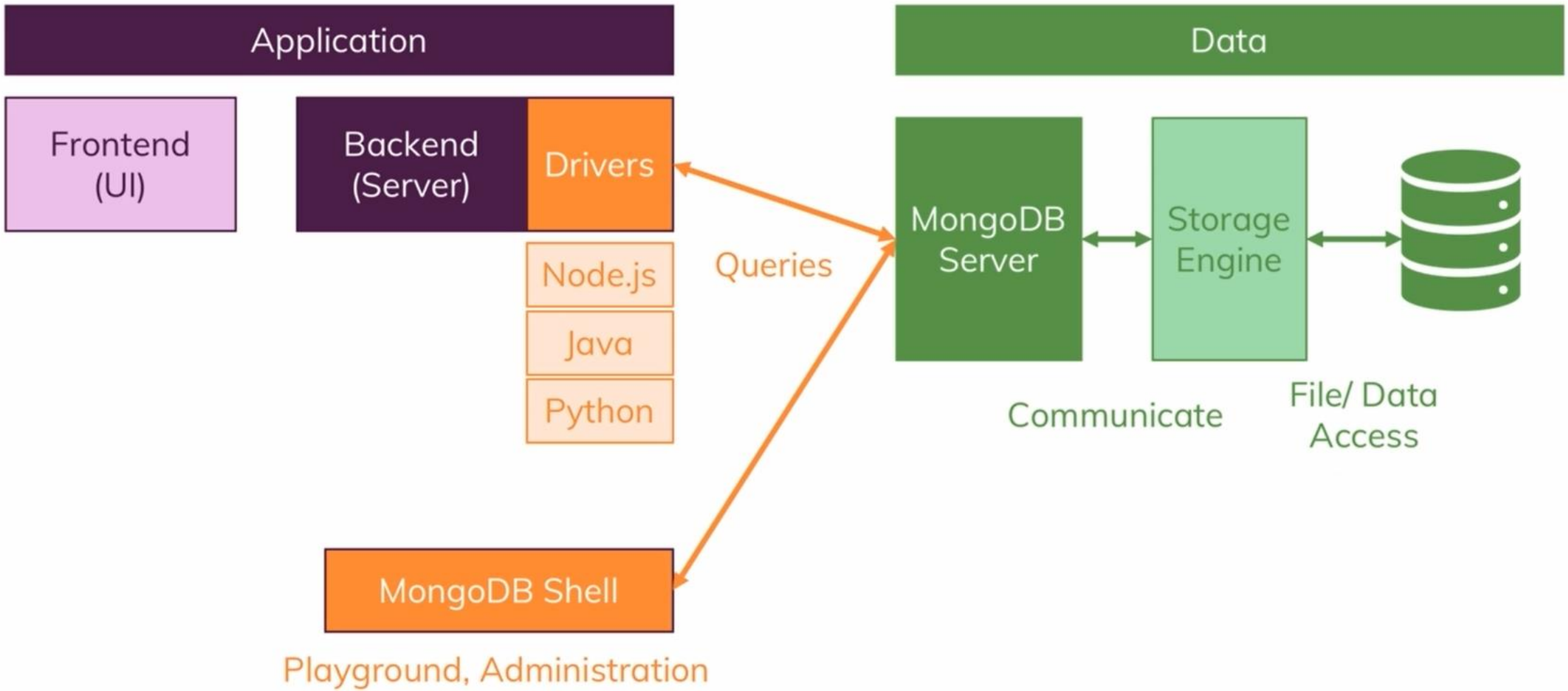
$project

1:1

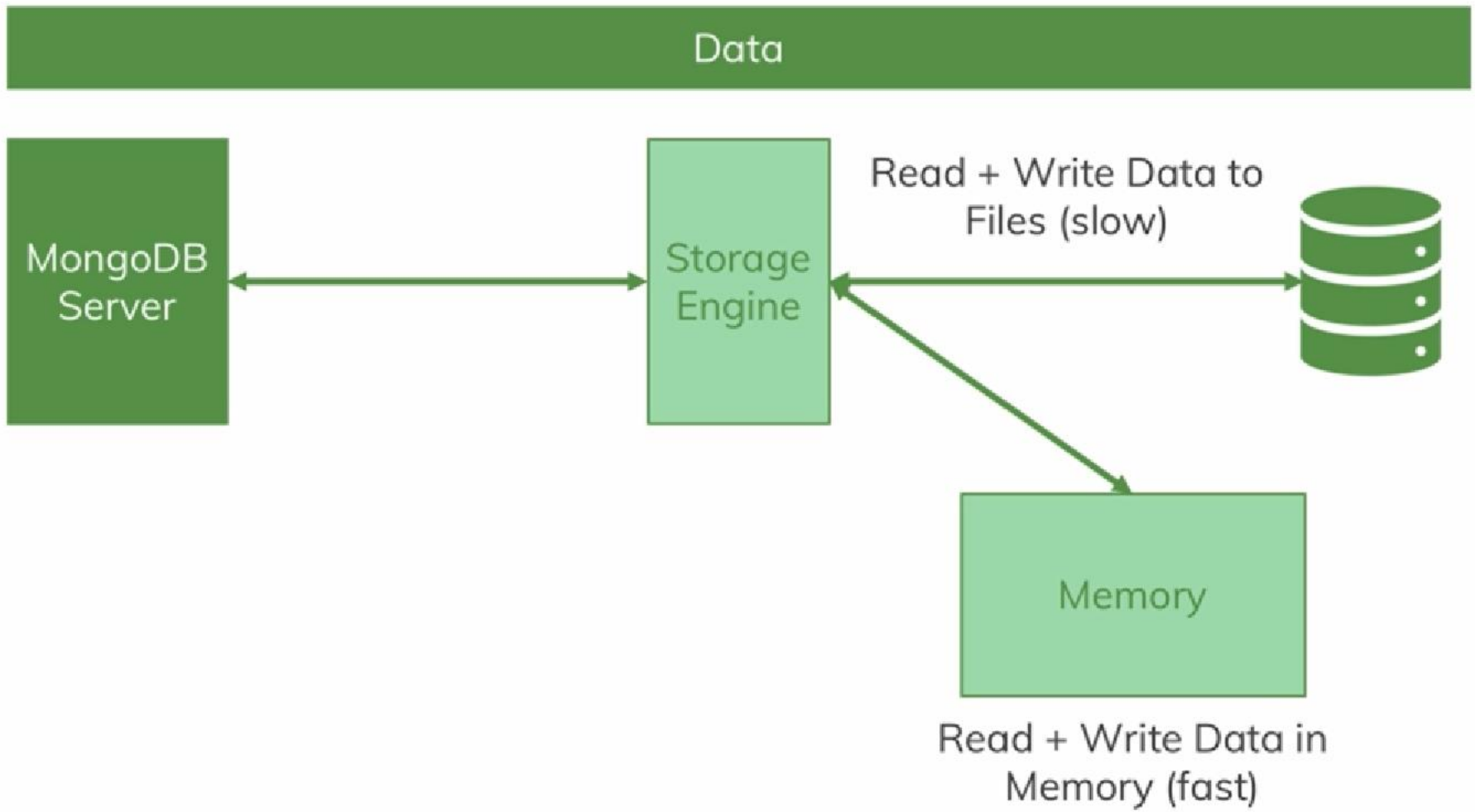
Include/ Exclude Fields, Transform Fields (within a Single Document)


MONGO DB DATABASE
Stitch
Self Managed/
Enterprise
Atlas Cloud
Mobile
Serverless Query API
Cloud Manager /
Ops Manager
Serverless Functions
Compass / GUI Tool
Database Triggers
BI Connectors
Real-Time Sync
MongoDB Charts

Working with MongoDB



A Closer Look



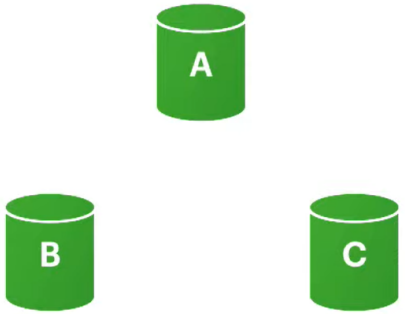
Cluster

Replica Sets

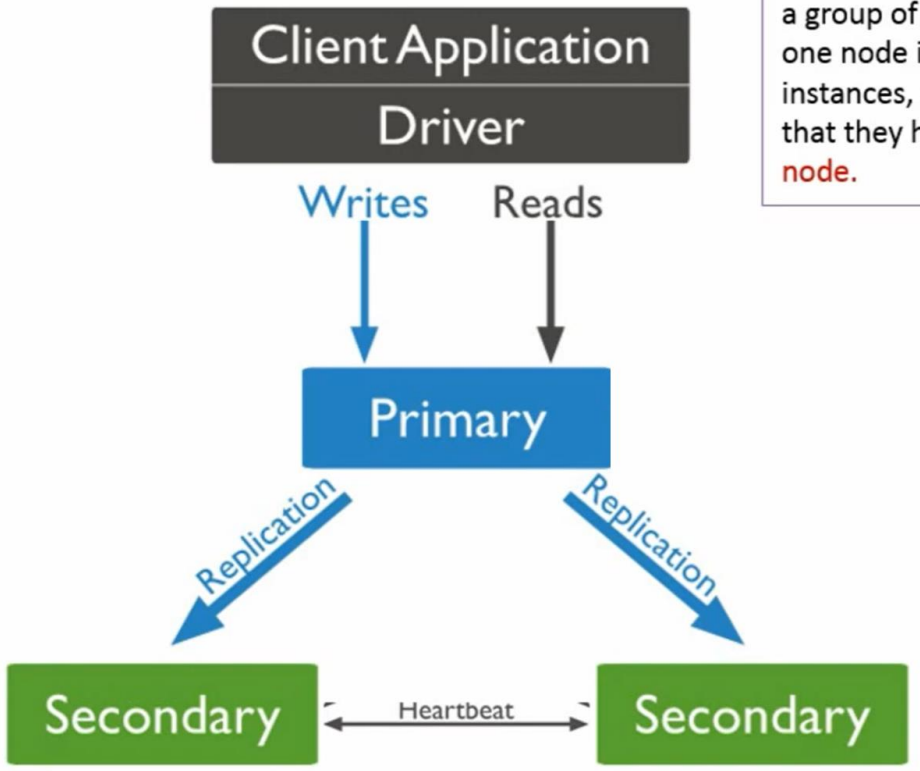
- ✓ Maintain same dataset
- ✓ Provides redundancy and high availability
- ✓ Load balancing
- ✓ Basis for all Production deployments

Sharded Cluster

- ✓ Distributing Data between different machines
- ✓ Useful when u have very large datasets



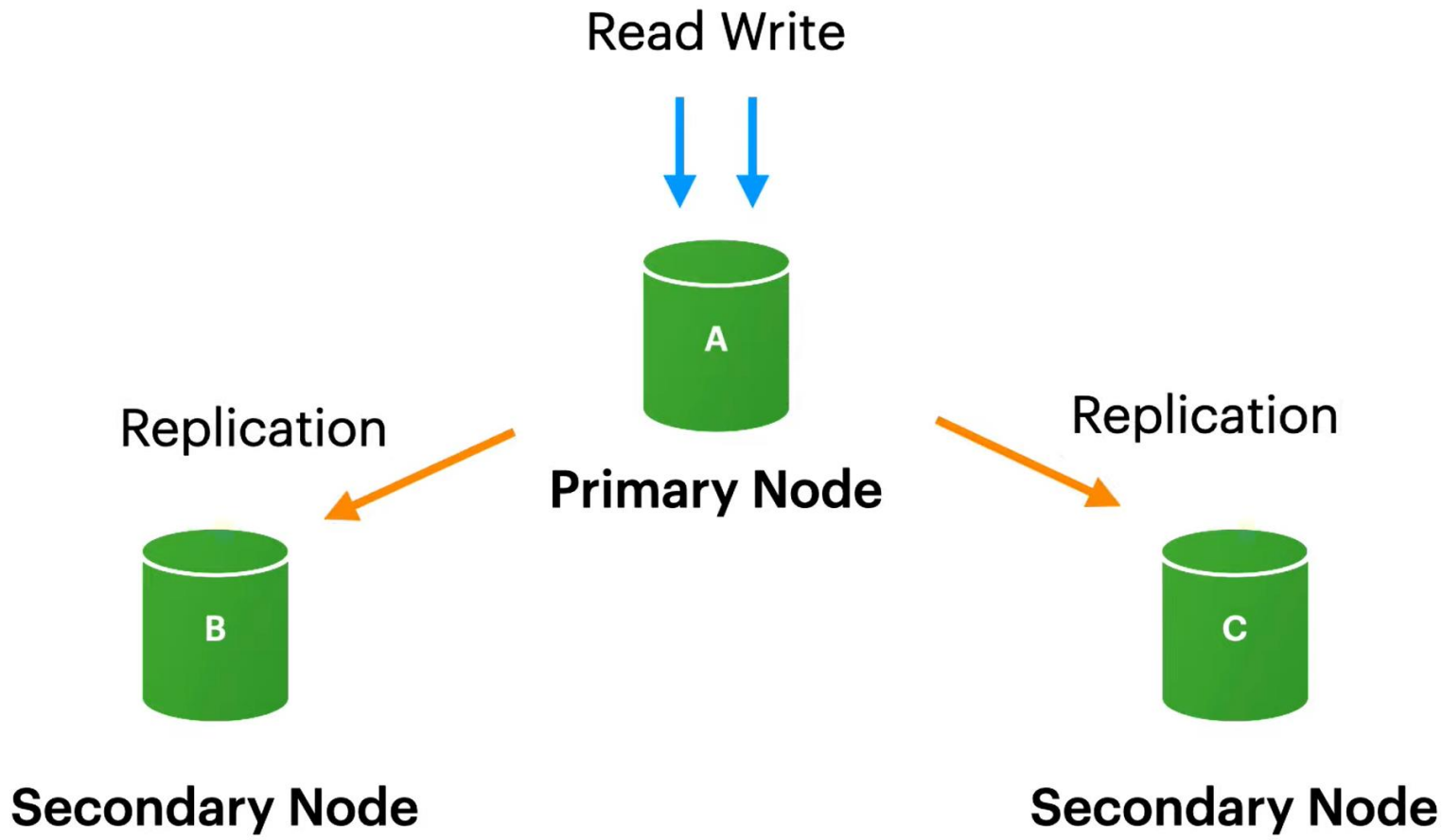
Replica Set



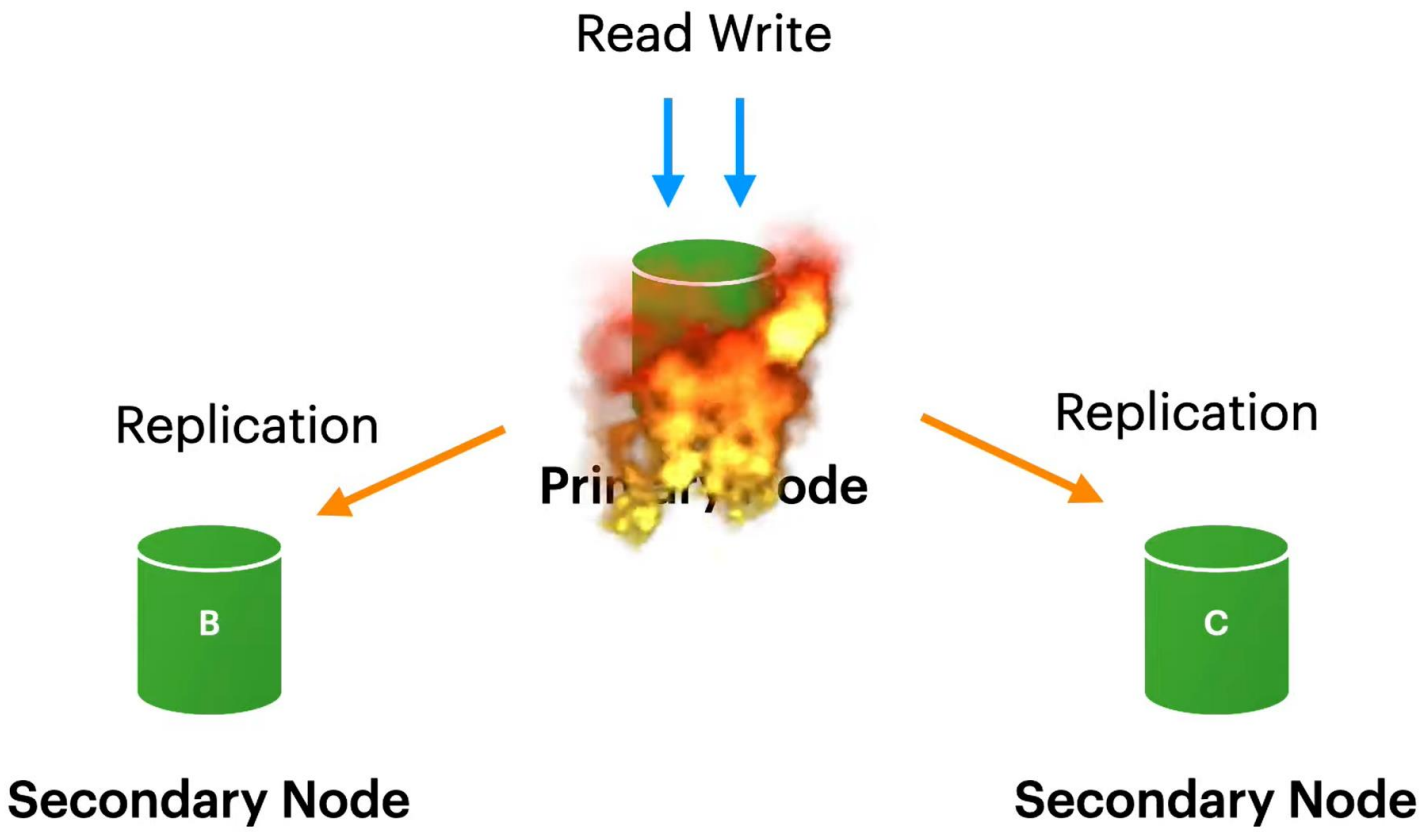
MongoDB achieves replication by the use of replica set. A replica set is a group of **mongod** instances that host the same data set. In a replica, one node is primary node that receives all write operations. All other instances, such as secondaries, apply operations from the primary so that they have the same data set. Replica set can have only one primary node.

Single Node ReplicaSet	3 Nodes ReplicaSet
Primary	1. Primary 2. Secondary1 3. Secondary2

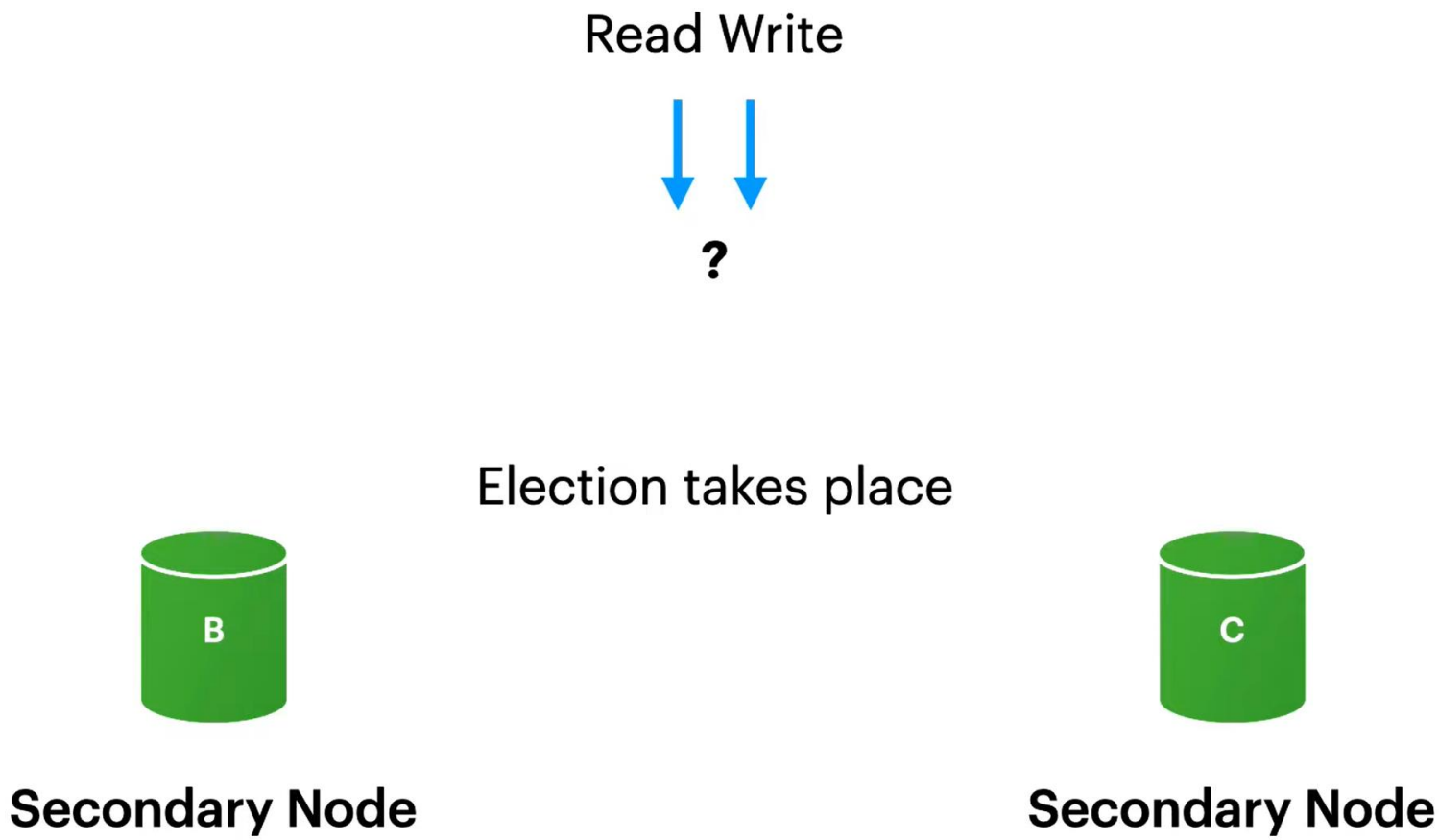
Replica Set


Read Write
A
Replication
Primary Node
Replication
B
C
Secondary Node
Secondary Node

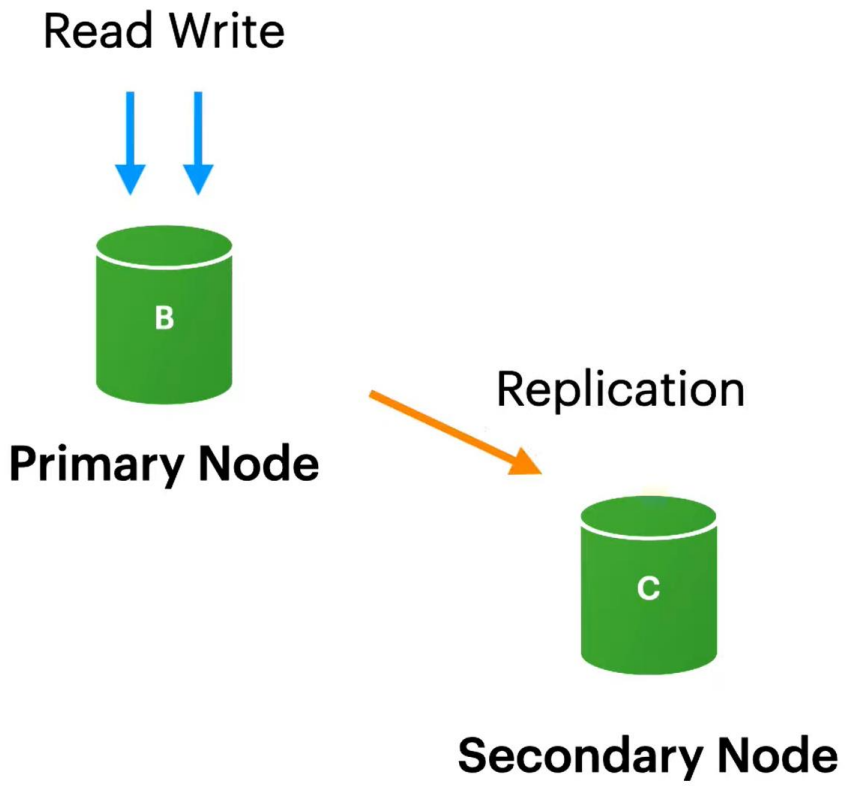
Replica Set


Read Write
Replication
Replication
B
C
Secondary Node
Secondary Node

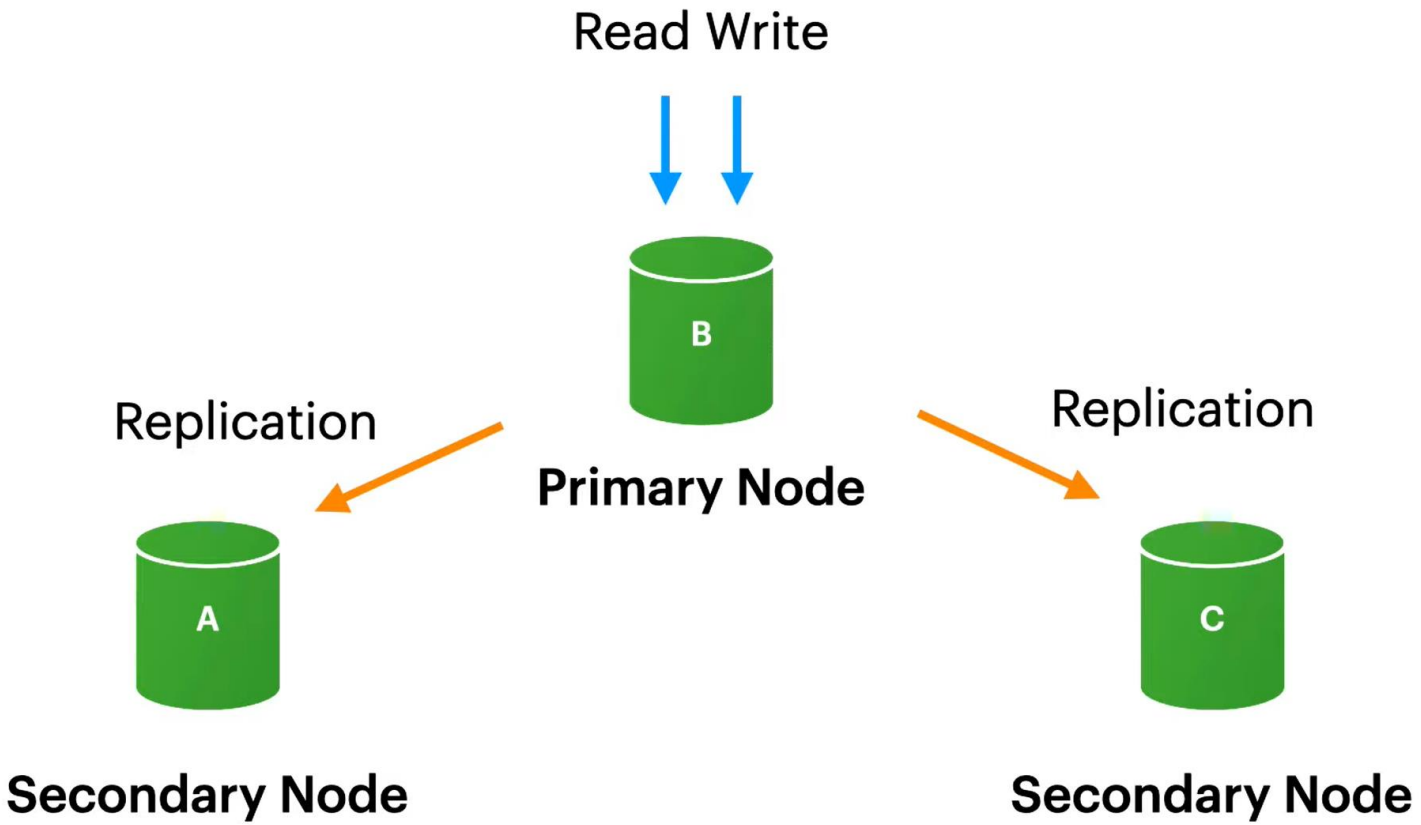
Replica Set


Read Write
?
Election takes place
B
C
Secondary Node
Secondary Node

Replica Set

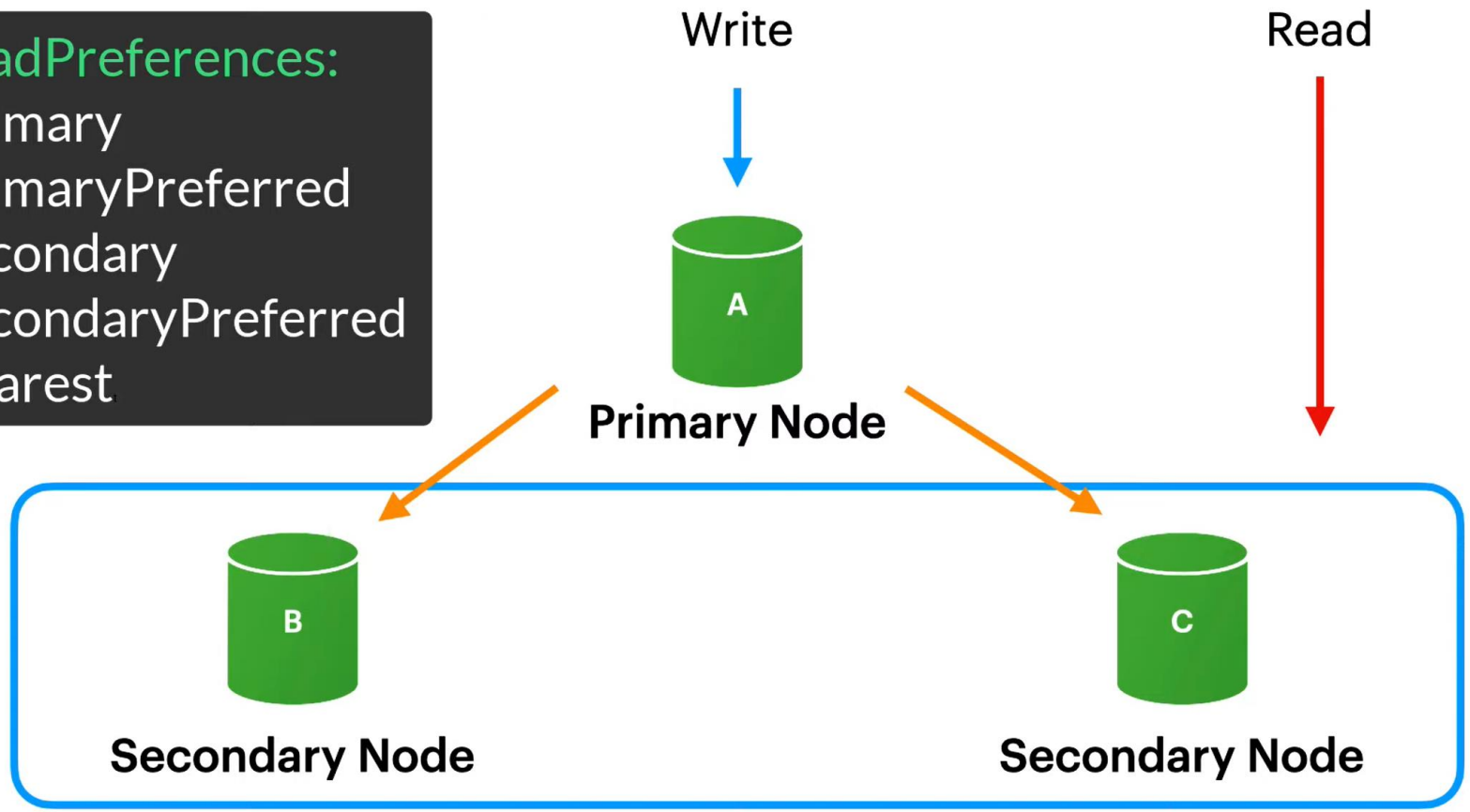


Replica Set


Read Write
B
Replication
Primary Node
Replication
A
C
Secondary Node
Secondary Node

Load Balancing

readPreferences:
primary
primaryPreferred
secondary
secondaryPreferred
nearest



Cluster Setup

Host: localhost (127.0.0.1)

Primary Node

- *Port:* 2717
- *Db path:* /mongos/db1
- `mongod --port 2717 --dbpath /mongos/db1 --replSet myReplicaSet`

Secondary Node

- *Port:* 2727
- *Db path:* /mongos/db2
- `mongod --port 2727 --dbpath /mongos/db2 --replSet myReplicaSet`

Secondary Node

- *Port:* 2737
- *Db path:* /mongos/db3
- `mongod --port 2737 --dbpath /mongos/db3 --replSet myReplicaSet`

Commands

On Primary Node

`rs.status()`

- Tells the status of replica set

`rs.initiate()`

- Initiates a replica set on primary node

`rs.add("localhost:2727")`

- Adds a member to replica set

`rs.remove("localhost:2727")`

- Removes a member from replica set

Connecting to ReplicaSet

MongoDB URI

typically

- `mongodb://localhost:27017`

For, Replica sets

- `mongodb://localhost:2717,localhost:2727,localhost:2737/?replicaSet=myRepl`

THANK YOU

Corporate Trainer & Motivational Speaker
9035351965