MongoDB

Build Faster, Build Smarter

Big Data

Big Data is a collection of data that is huge in volume, yet growing exponentially with time.

Data is too large in size and complexity that none of traditional data management tools can store it or process it efficiently.

Examples - Big Data

Stock Exchange	The New York Stock Exchange is an example of Big Data that generates about one terabyte of new trade data per day
Social Media	Statistic shows that 500+terabytes of new data get ingested into the databases of social media site Facebook, every day
Airlines	A single Jet engine can generate 10+ terabytes of data in 30 minutes of flight time. With many thousand flights per day, generation of data reaches up to many Petabytes

Types – Big Data

Structured	Unstructured	Semi-structured
An ‘Employee’ table in a database is an example of Structured Data	The output returned by ‘Google Search’	Personal data stored in an XML file

What is NoSQL ?

NoSQL databases (aka "not only SQL") are non-tabular databases and store data differently than relational tables.

NoSQL provides flexible schemas and scale easily with large amounts of data and high user loads

NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph.

When should NoSQL be used ?

Fast-paced Agile development

Storage of structured and semi-structured data

Huge volumes of data

Requirements for scale-out architecture

Modern application paradigms like microservices and real-time streaming

MongoDB : Introduction

MongoDB is a document-oriented NoSQL database used for high volume data storage.

MongoDB makes use of collections and documents

Documents consist of key-value pairs which are the basic unit of data in MongoDB

Collections contain sets of documents and function which is the equivalent of relational database tables

MongoDB : Components

_id	MongoDB is a document-oriented NoSQL database used for high volume data storage.
Collections	A grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL.
Cursor	A pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.
Database	Container for collections like in RDMS wherein it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.
Document	A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values

MongoDB : Components

Field	A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases
JSON	This is known as JavaScript Object Notation. This is a human-readable, plain text format for expressing structured data

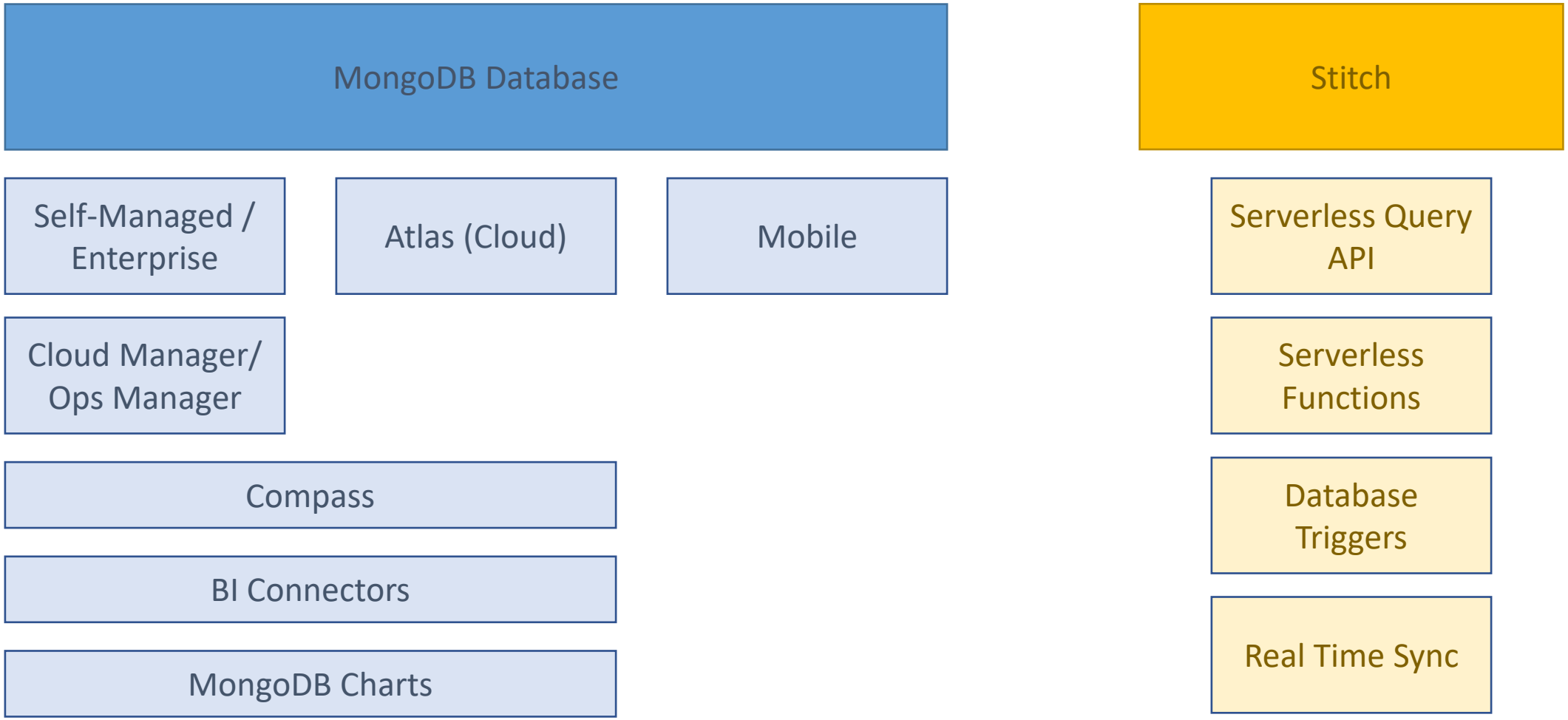
Why MongoDB ?

Document-oriented	MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents.
Ad hoc queries	MongoDB supports searching by field, range queries, and regular expression searches. Queries can be made to return specific fields within documents.
Indexing	Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.
Replication	MongoDB can provide high availability with replica sets.
Load balancing	MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances.

Difference between MongoDB & RDBMS

RDBMS	MongoDB	Difference
Table	Collection	In RDBMS, the table contains the columns and rows which are used to store the data whereas, in MongoDB, this same structure is known as a collection.
Row	Document	In RDBMS, the row represents a single, implicitly structured data item in a table. In MongoDB, the data is stored in documents.
Column	Field	In RDBMS, the column denotes a set of data values. These in MongoDB are known as Fields
Joins	Embedded Documents	In RDBMS, data is sometimes spread across various tables and in order to show a complete view of all data, a join is formed across tables to get the data. In MongoDB, the data is normally stored in a single collection, but separated by using Embedded documents. Hence there is no concept of joins in MongoDB.

MongoDB Ecosystem



DB, Shell and Tools

MongoDB Database Installation

- https://www.mongodb.com/try/download/community

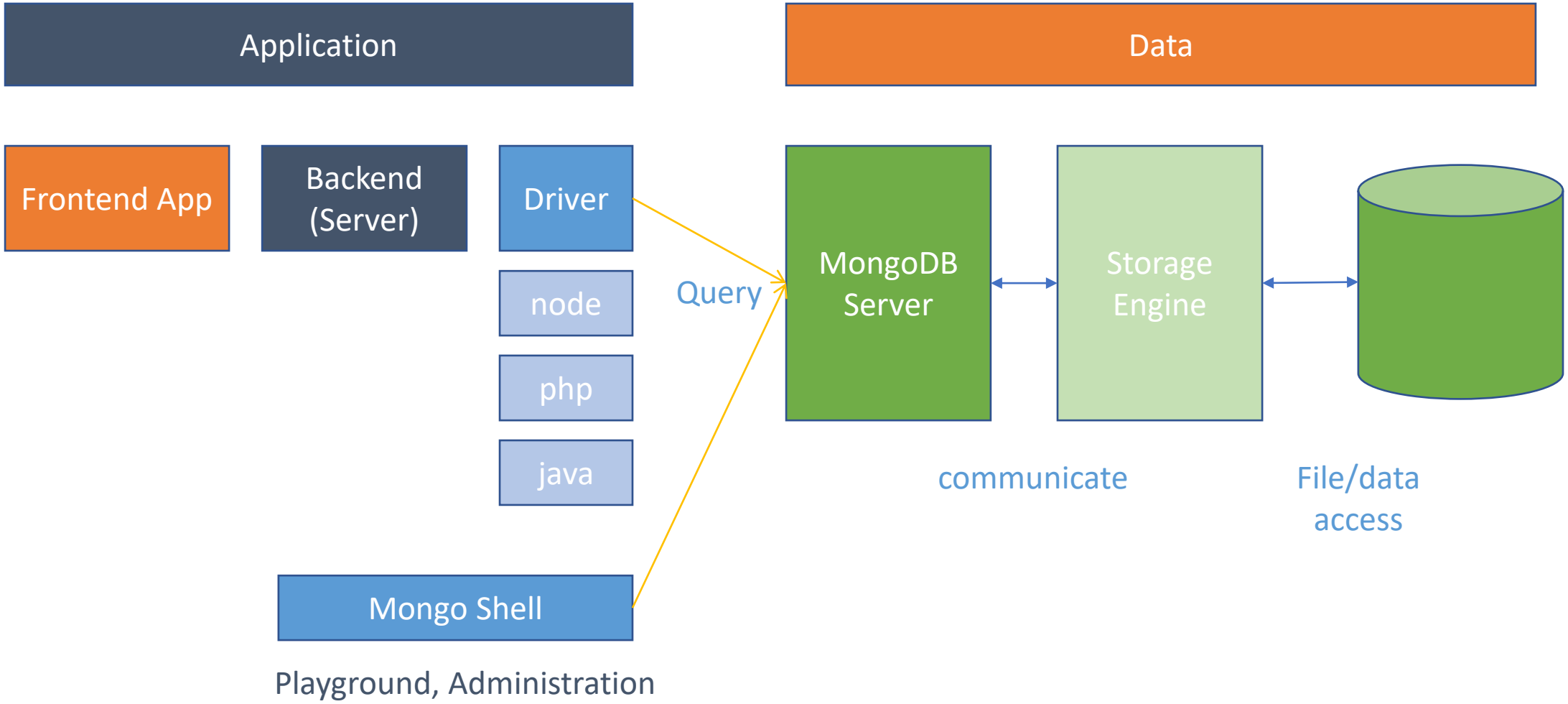
Mongo Shell Installation

- https://www.mongodb.com/try/download/shell

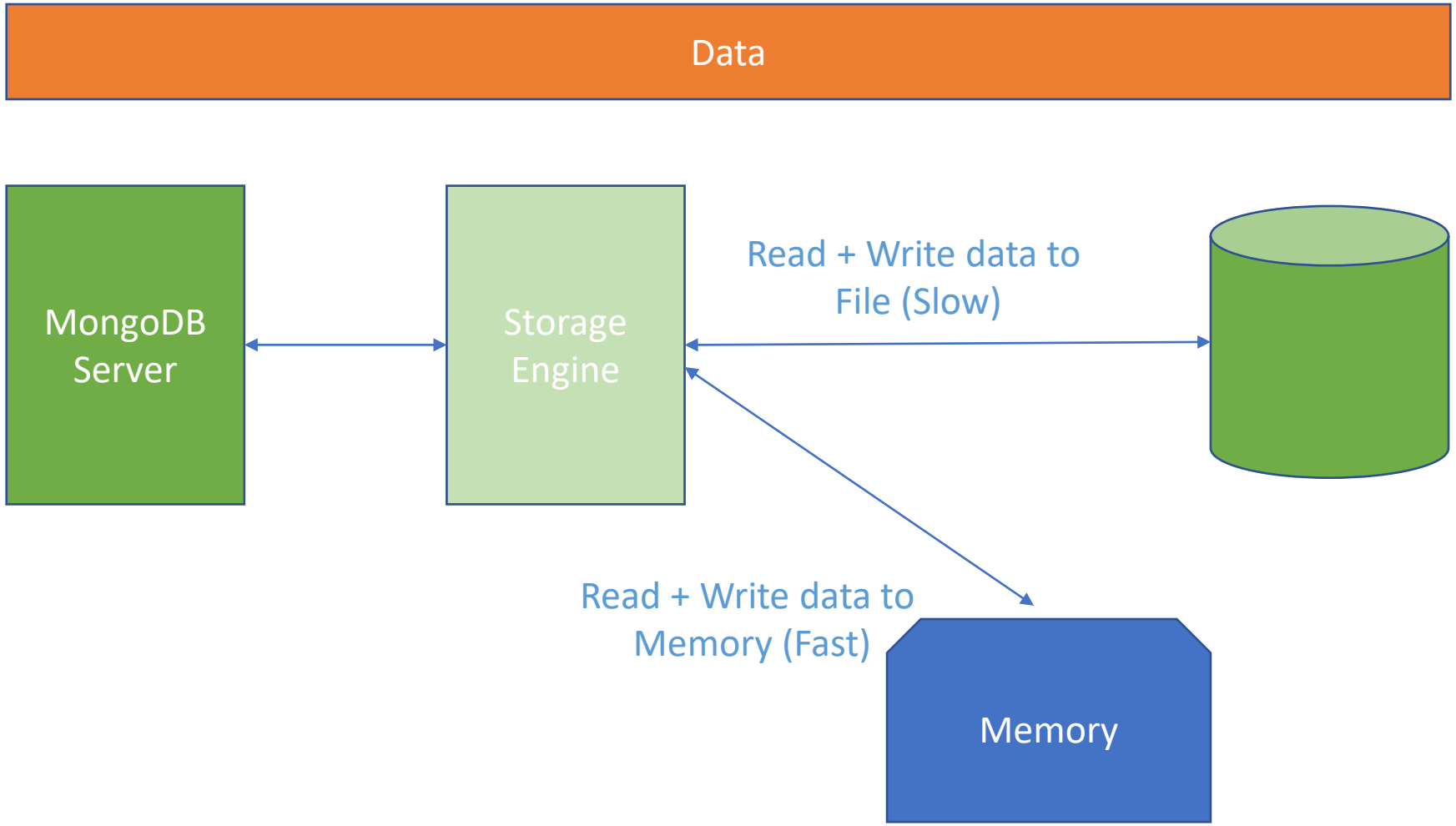
Mongo Database Tools

- https://www.mongodb.com/try/download/database-tools

MongoDB Working



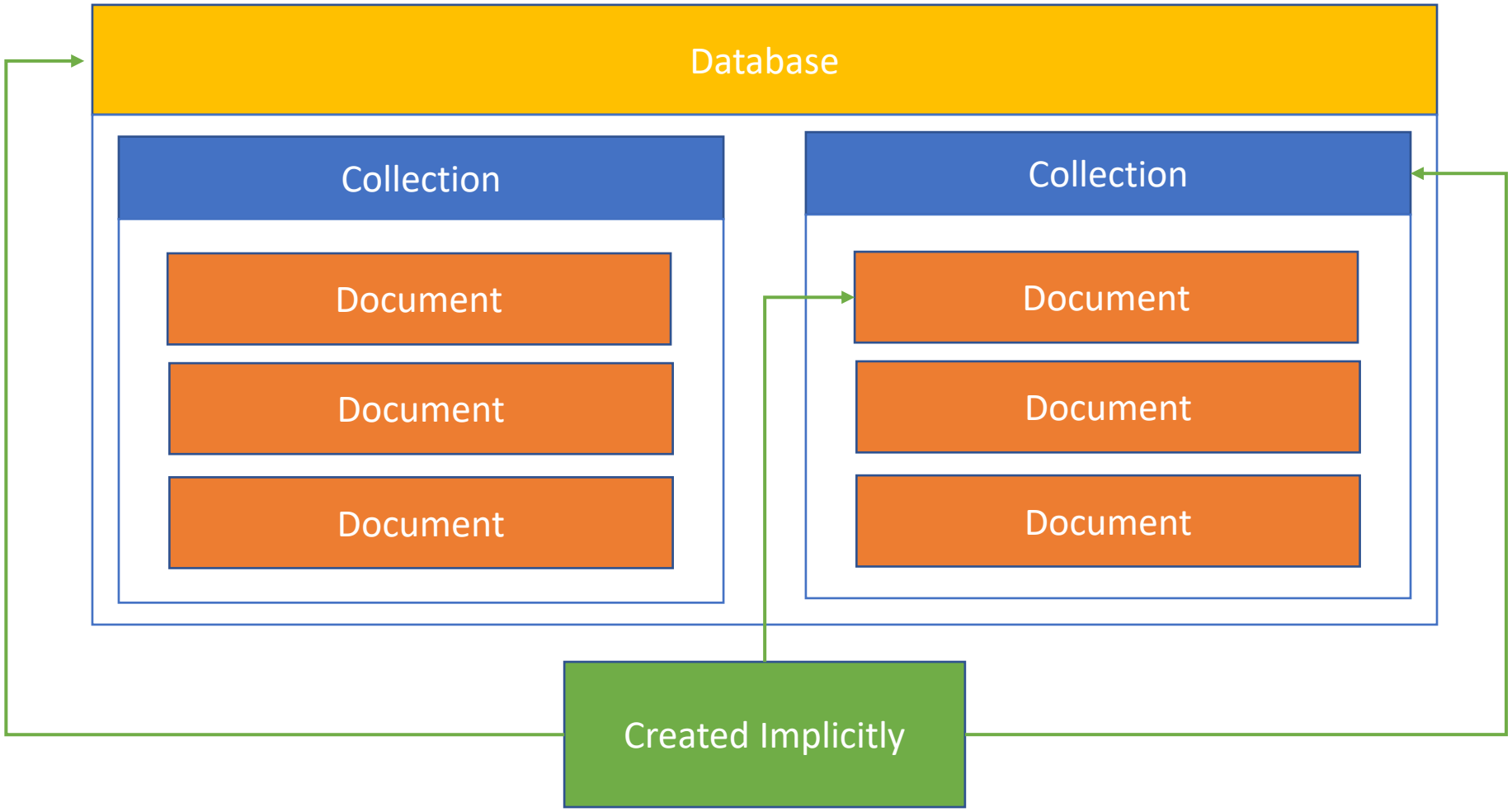
Closer look to Data Layer



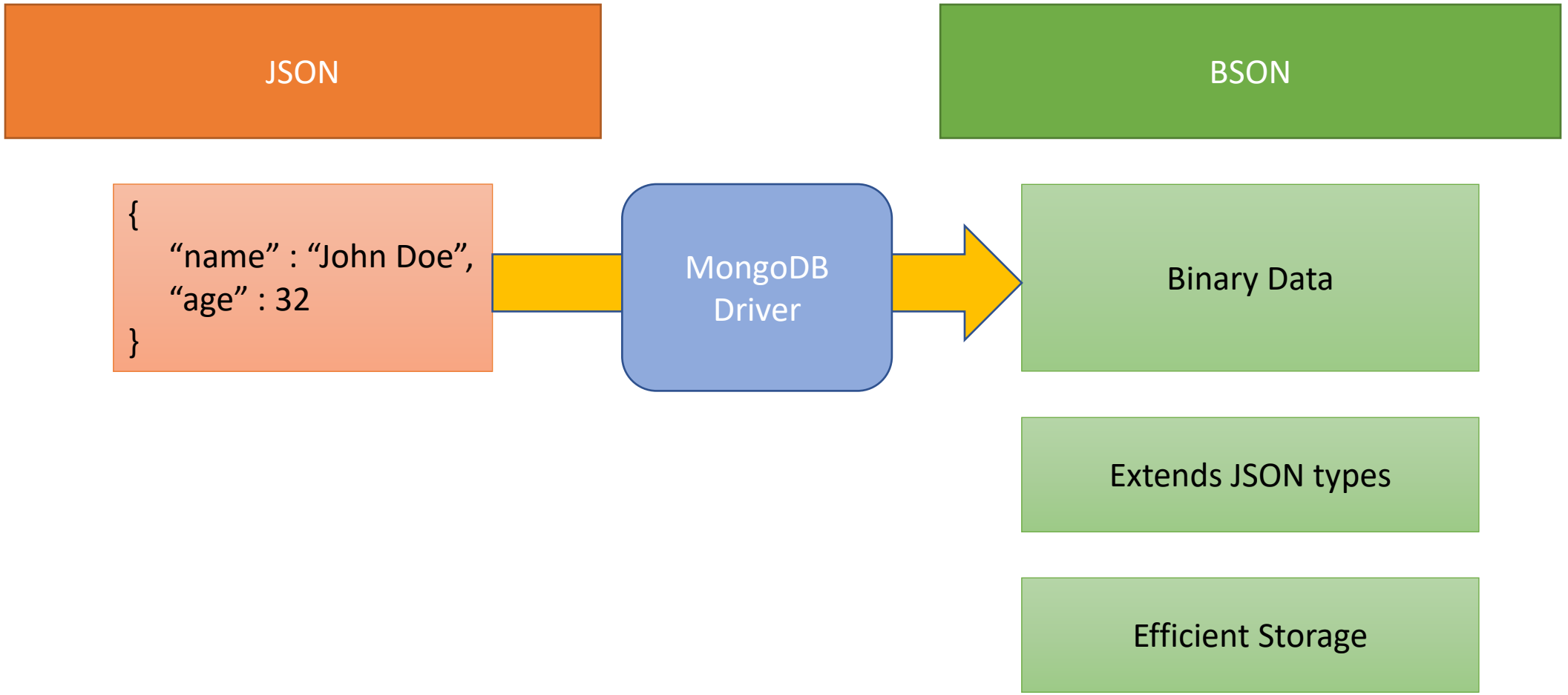
Working with Database

Document & CRUD Operations

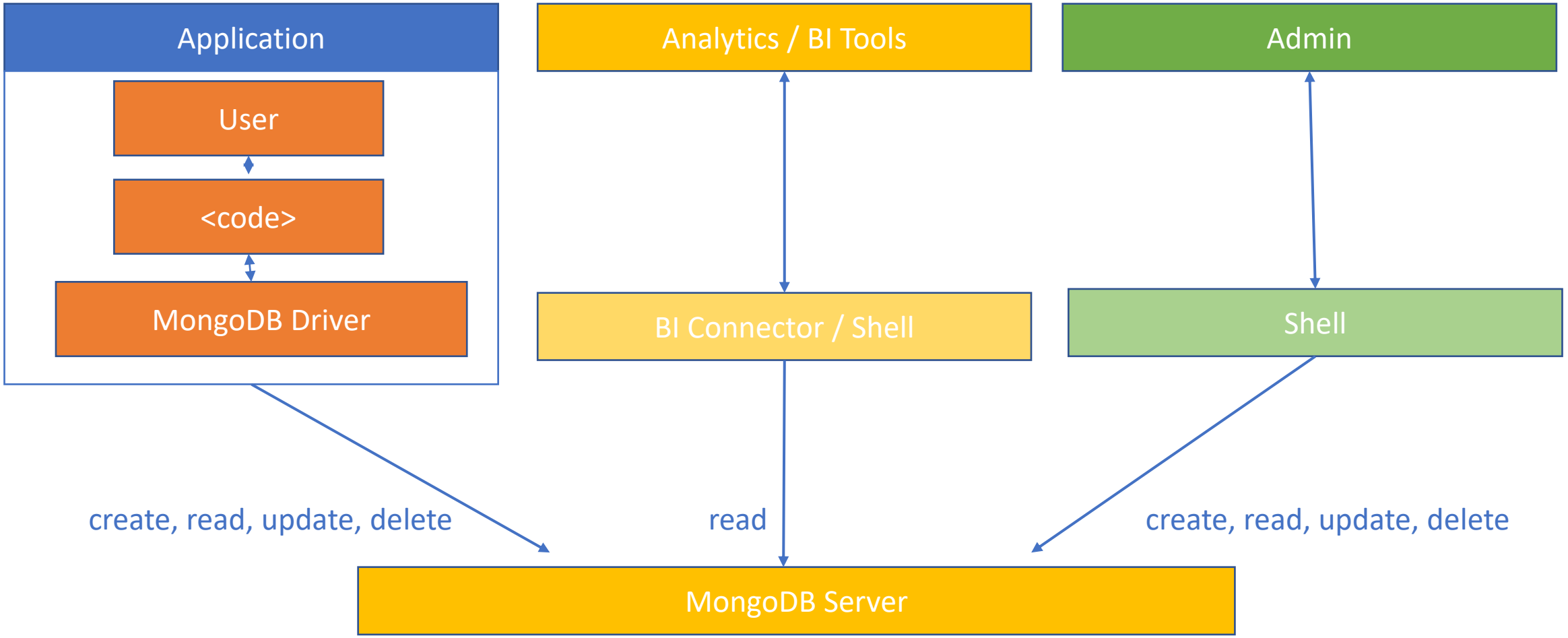
Databases, Collections, Documents



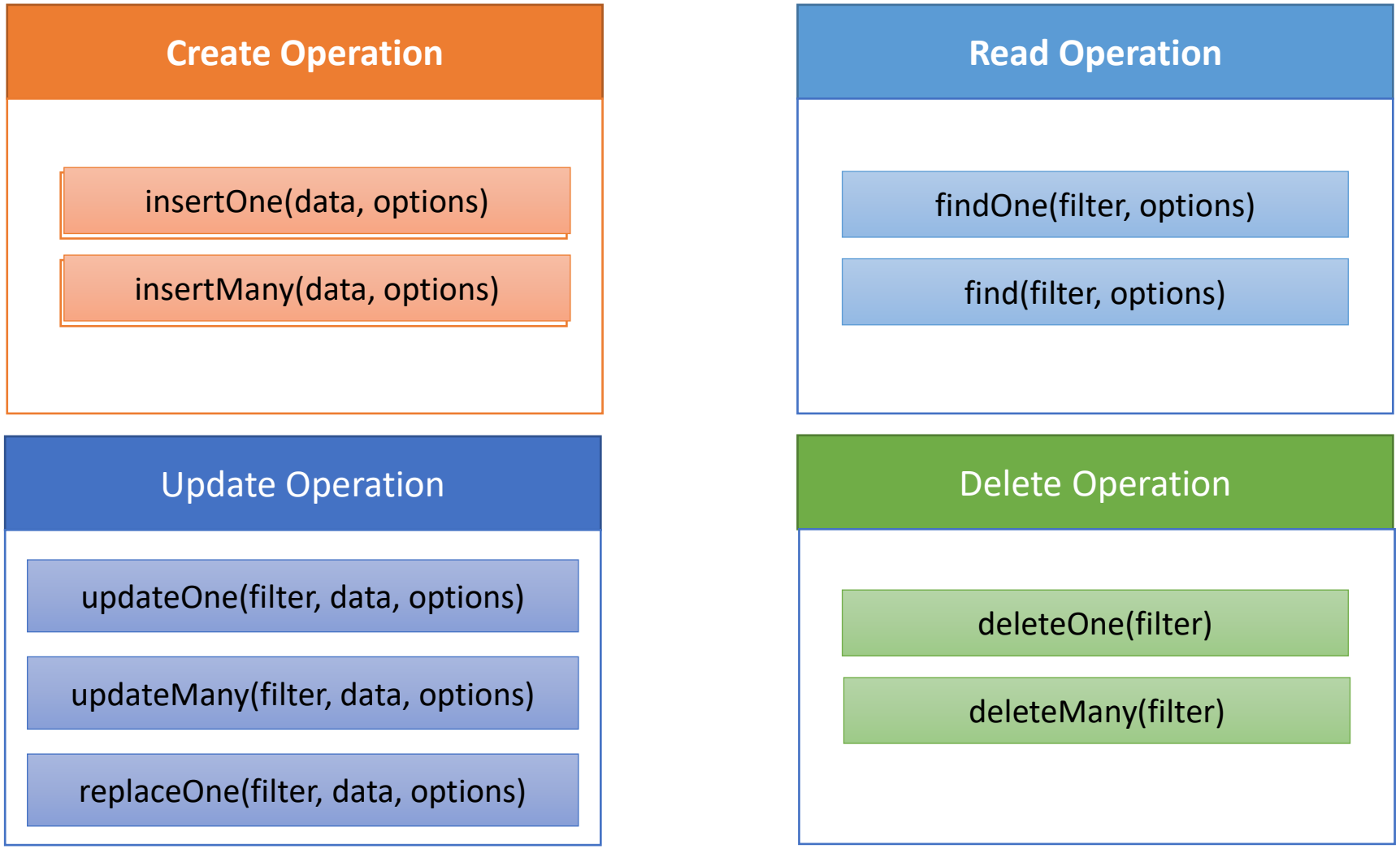
Comparing JSON vs BSON



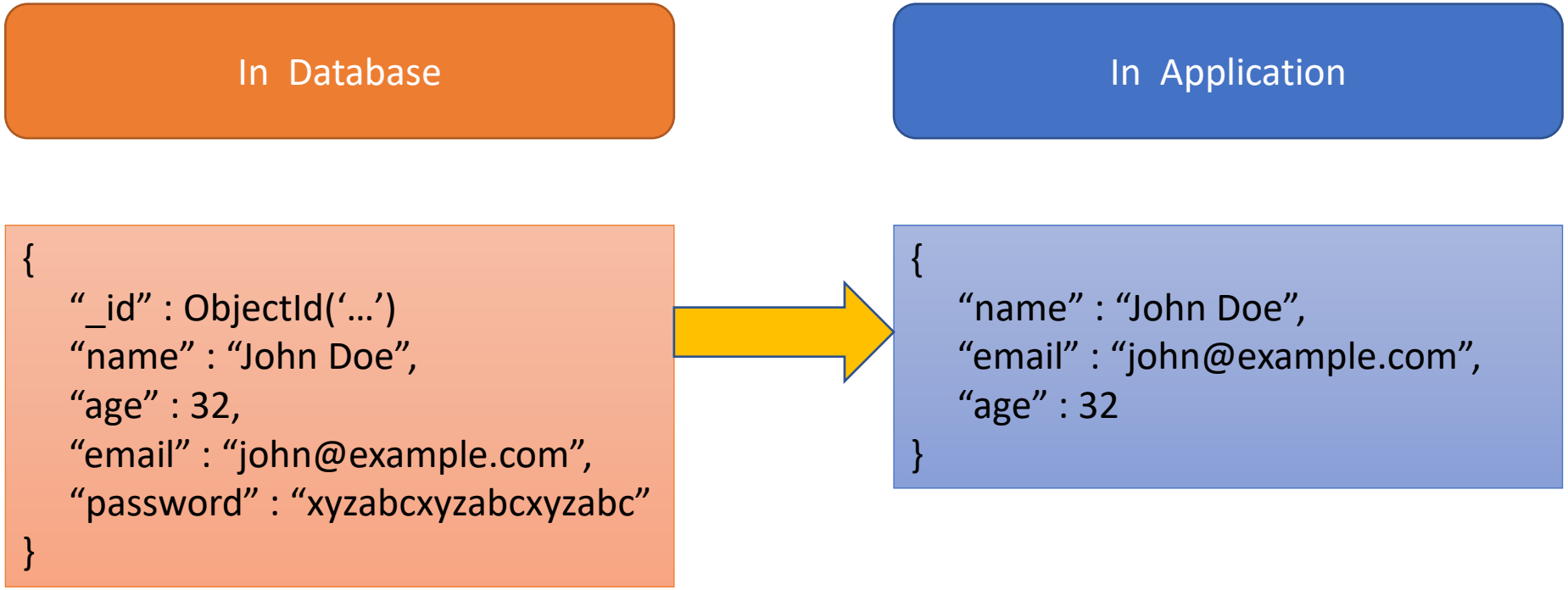
CRUD Operations & MongoDB



CRUD Operations



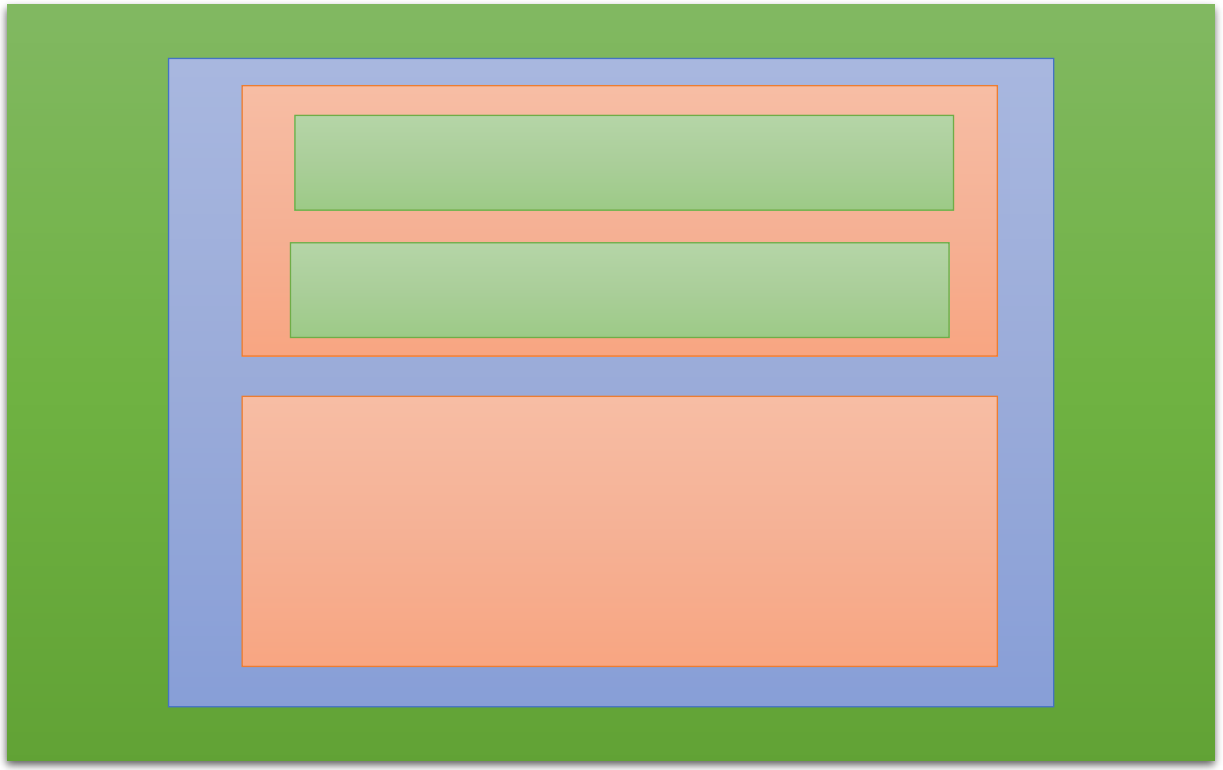
Projection – Data Exclusion



Embedded Document

Upto 100 levels of nesting allowed

Max 16Mb / document



Patients

```
{
        firstName : “john”,
        lastName : “doe”,
        age : 32,
          history : [
            {
              “disease” : “cold”,
              “treatment” : “medicine”
            },
            {
              “disease” : “cold”,
              “treatment” : “medicine”
            },
          ]
}
```

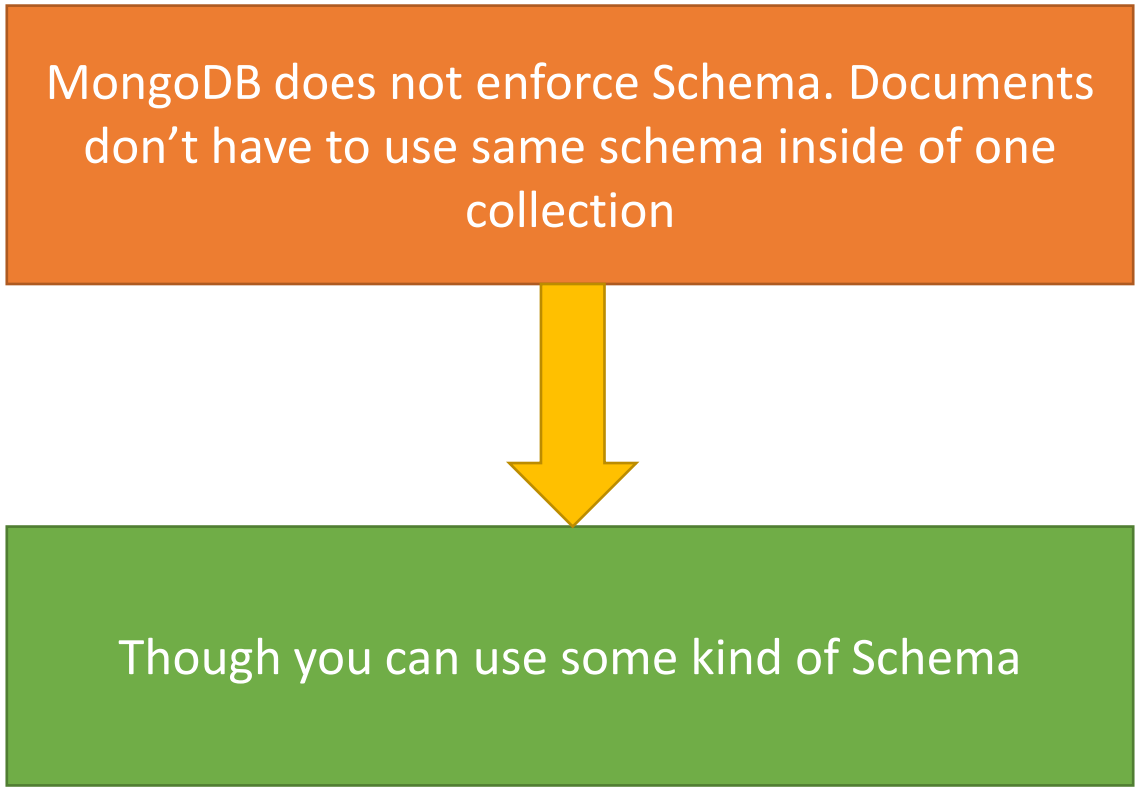
Task

- Insert three person records with at least 1 history entry per patient
- Update patient data of 1 patient with new age, name, and history entry
- Find all patient who are older than 30 (or values you inserted)
- Delete all patient who got cold as a disease

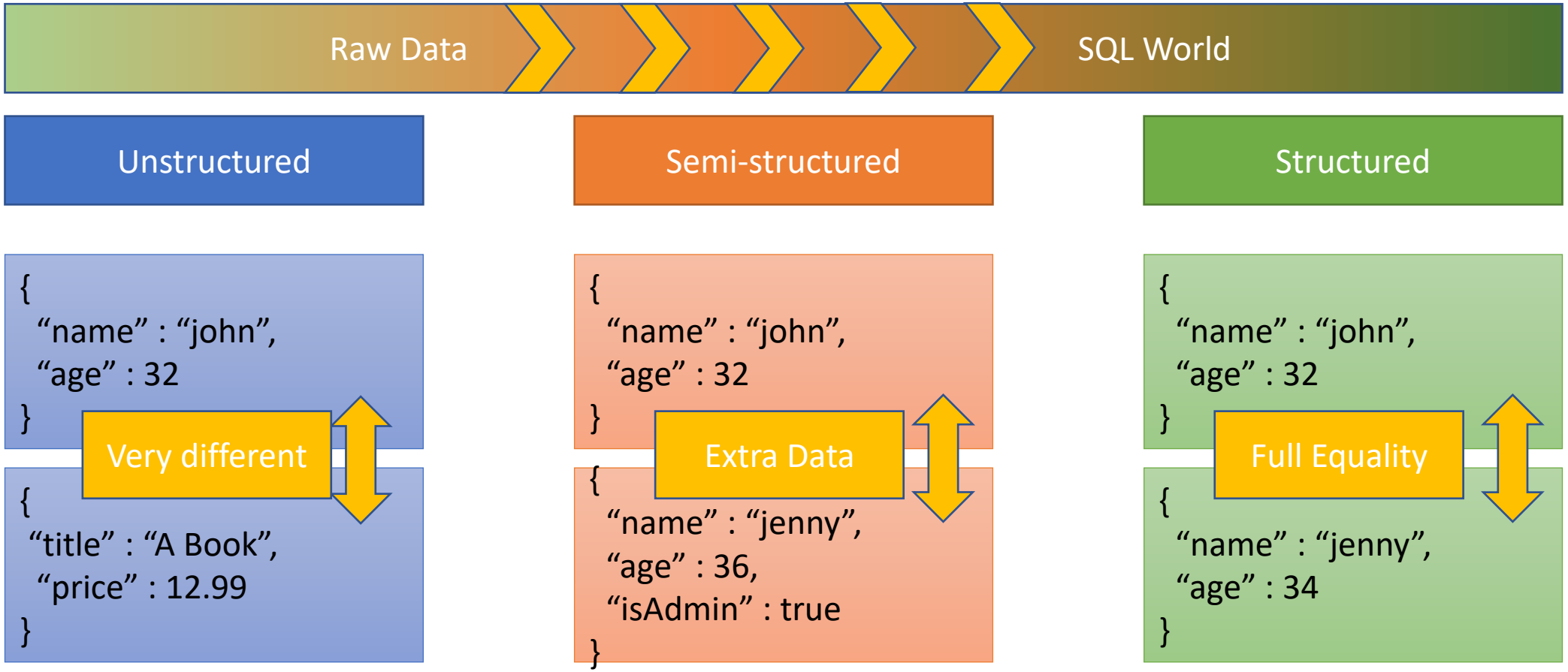
Data Schemas & Data Modelling

Storing Data Correctly

Schema-less or Not?



To Schema or NOT to Schema



Data Schemas & Data Modelling

Which data my App need or generate ?	Defines the field you will need (and how they relate)
Where do I need my data?	Defines your required collection + field grouping
Which kind of data do I want to display?	Defines which queries you will need
How often I fetch my data?	Defines whether you should optimize for easy fetching
How often do I write or change my data?	Defines whether you should optimize for easy writing

Relations - Options

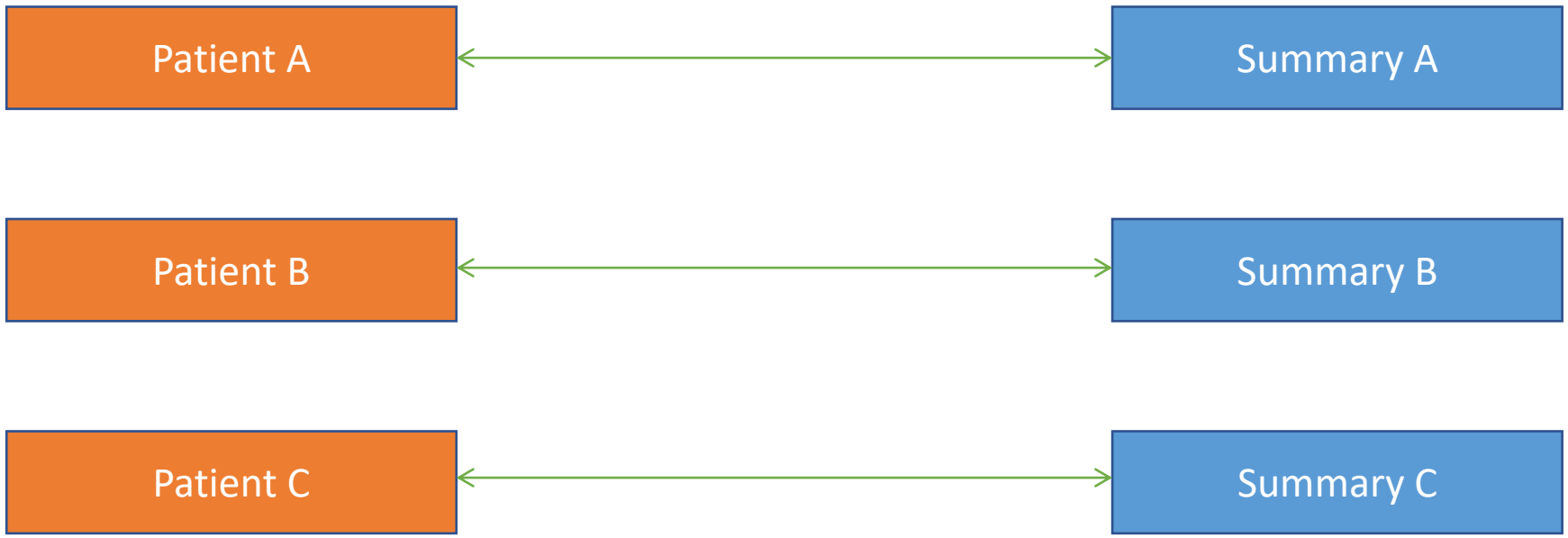
Embedded Document

```
{
    "userName" : "john doe",
    "age" : 32,
    "address"  : {
        "city" : "Bengaluru",
        "street" : "201, Main Road, Marathahalli"
    }
}
```

```
{
    "userName" : "john doe",
    "favBooks" : [
        { "bookNam                    , "author" : "Author 01"},
        { "b                    ook 2", "author" : "Author 02"},
                                : "Book 3", "author" : "Author 03"}
```

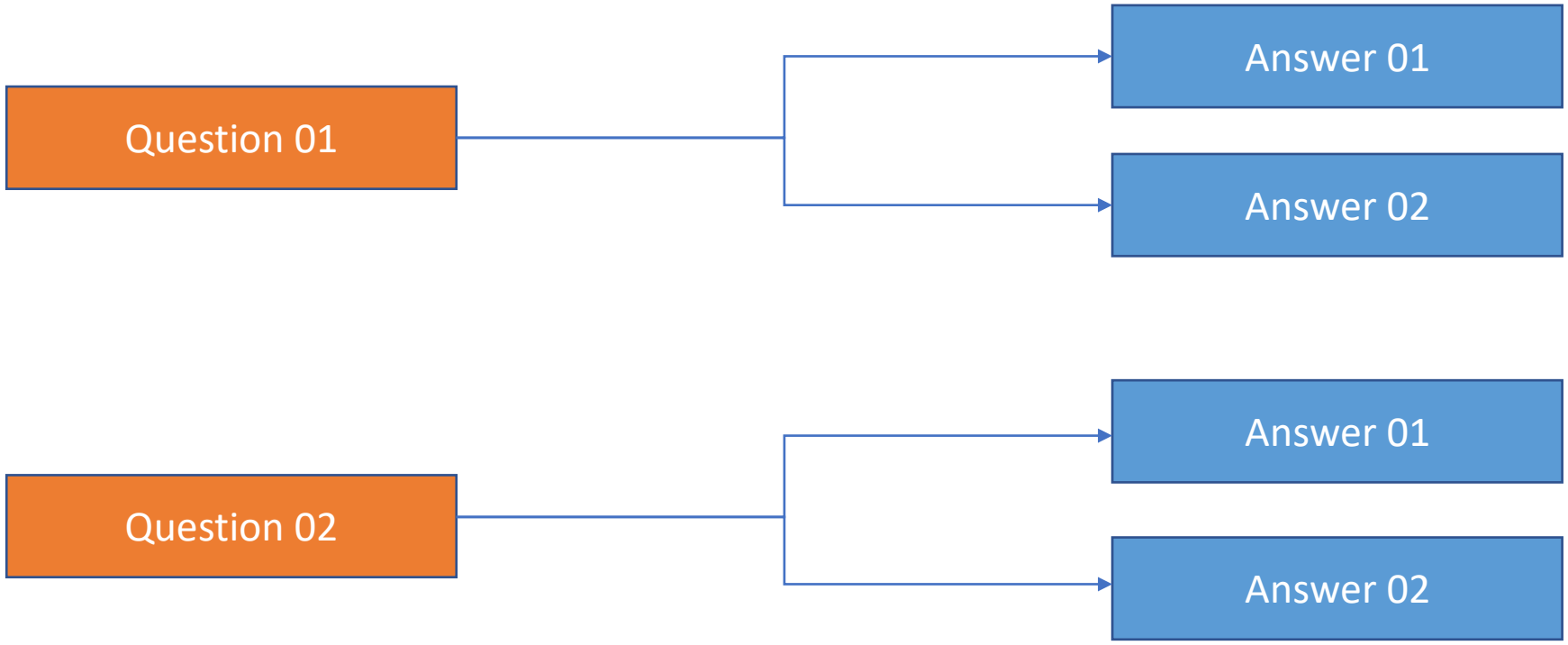
Consider using References

Case#1 Patient < - > Disease Summary



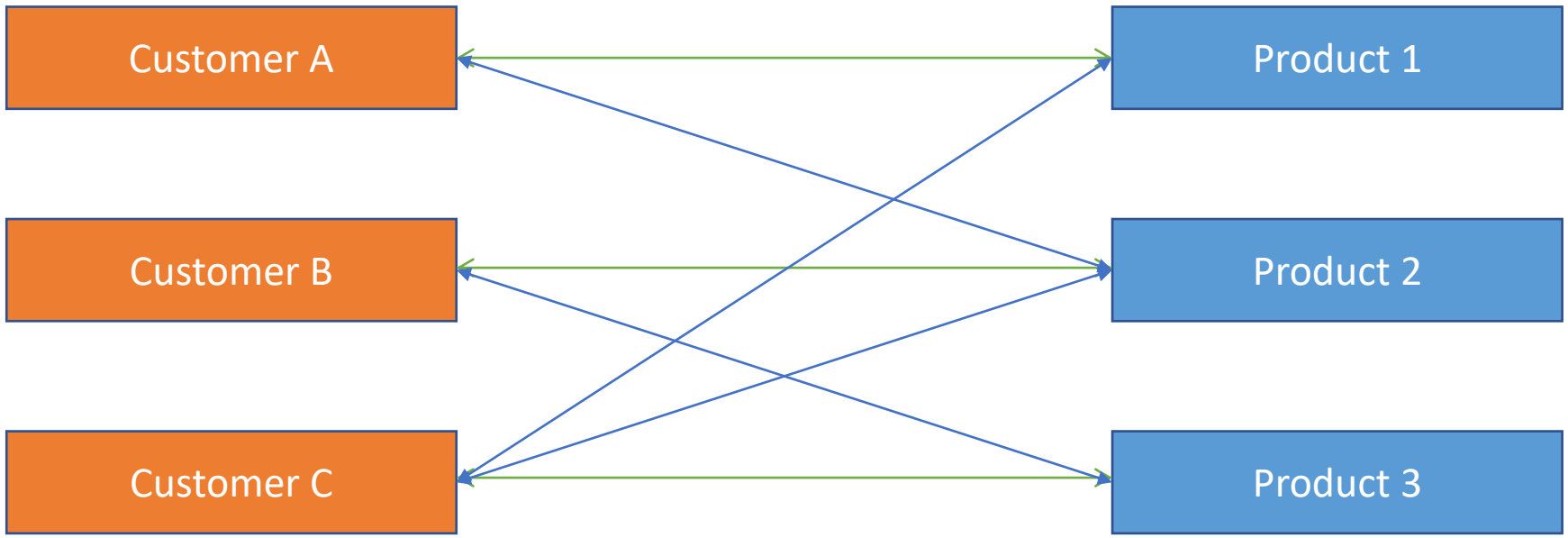
One Patient has one disease summary, and one disease summary belongs to one patient

Case#2 Question < - > Answers



One Question has Many Answers, and one Answer belongs to one Question

Case#3 Customer < - > Product (order)

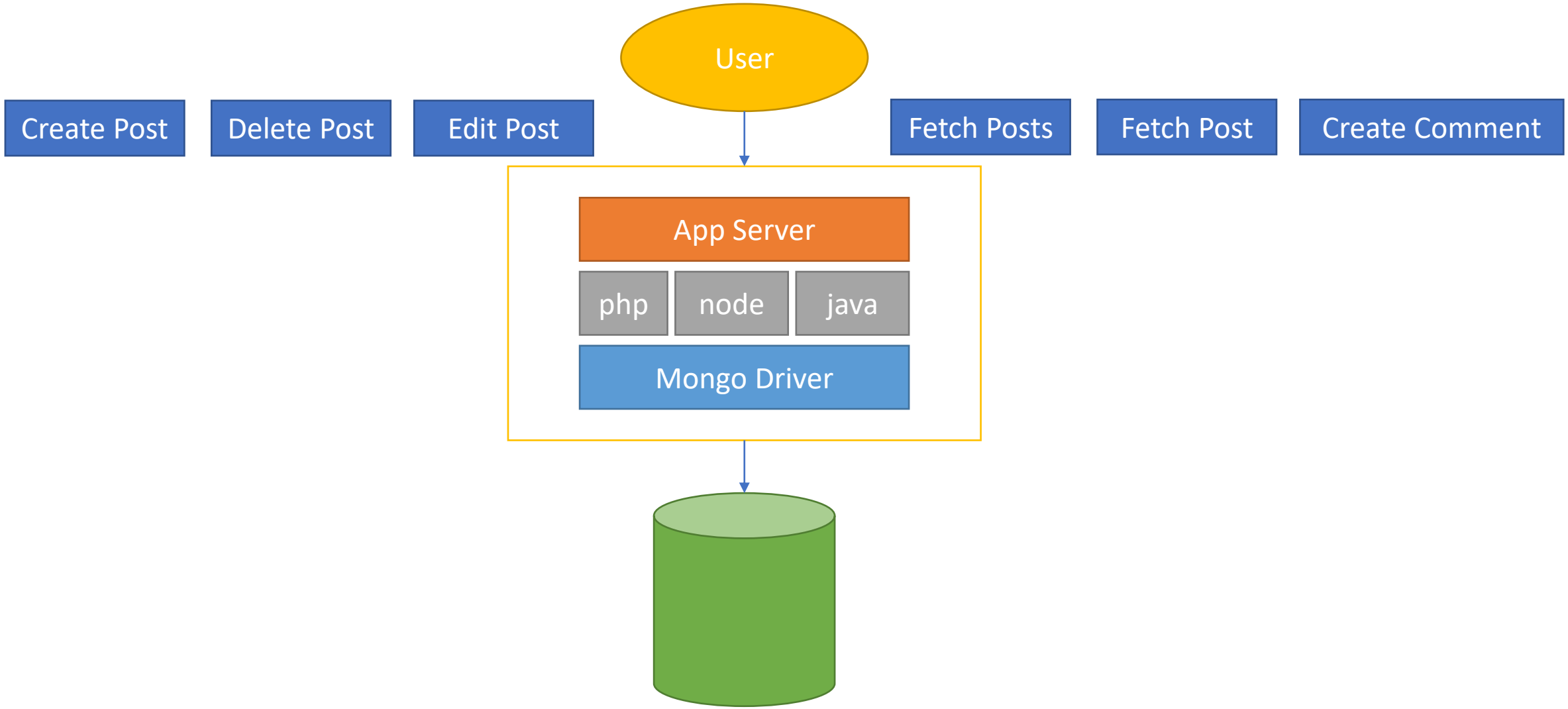


One Customer buy many Products, and one Product belongs to many Customers

Relations - Options

Embedded Document	References
Group data together logically	Splits data across collection
Great for data that group together and not 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 over nesting and size limits (by creating new documents)

Challenge Project – A Blog App

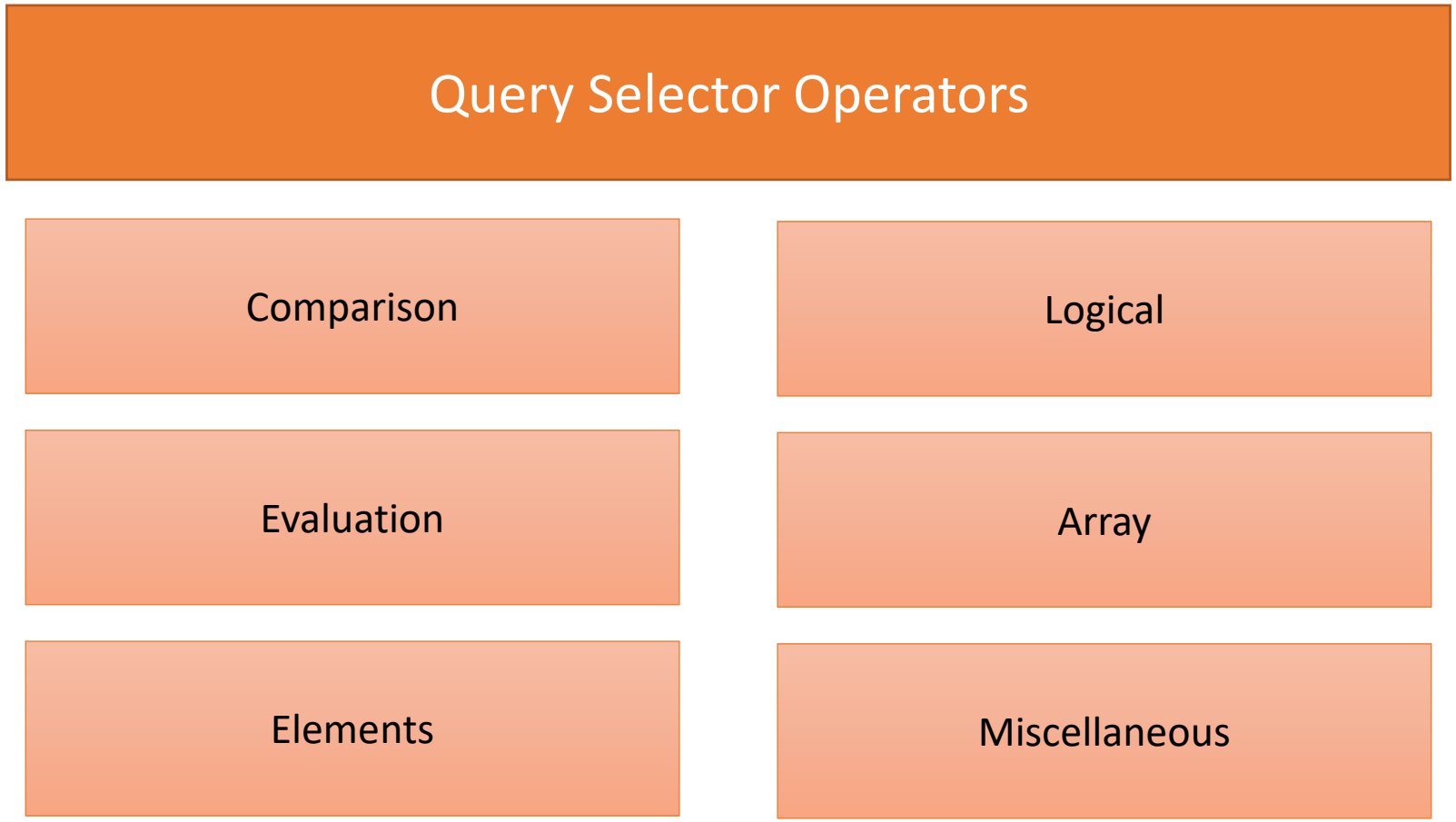


More on Insert Operations

Challenge – Insert Operation

1	Insert multiple companies (company data of your choice) into a collection – both insertOne and insertMany()
2	Deliberately insert duplicate ID data and “fix” failing addition with unordered inserts
3	Write data for a new company with both journaling being guaranteed and not being guaranteed

More on Read Operation



Challenge – Read Operation

1	Import the attached data into a new database (e.g boxOffice) and collection (e.g movieStarts)
2	Search all movies that have rating higher than 9.2 and runtime lower than 100 minutes
3	Search all movies that have genres of “Drama” or “Action”
4	Search all movies where visitors exceeded expectedVisitors

More on Update Operation

Query Selector Operators

$set, $unset

$rename

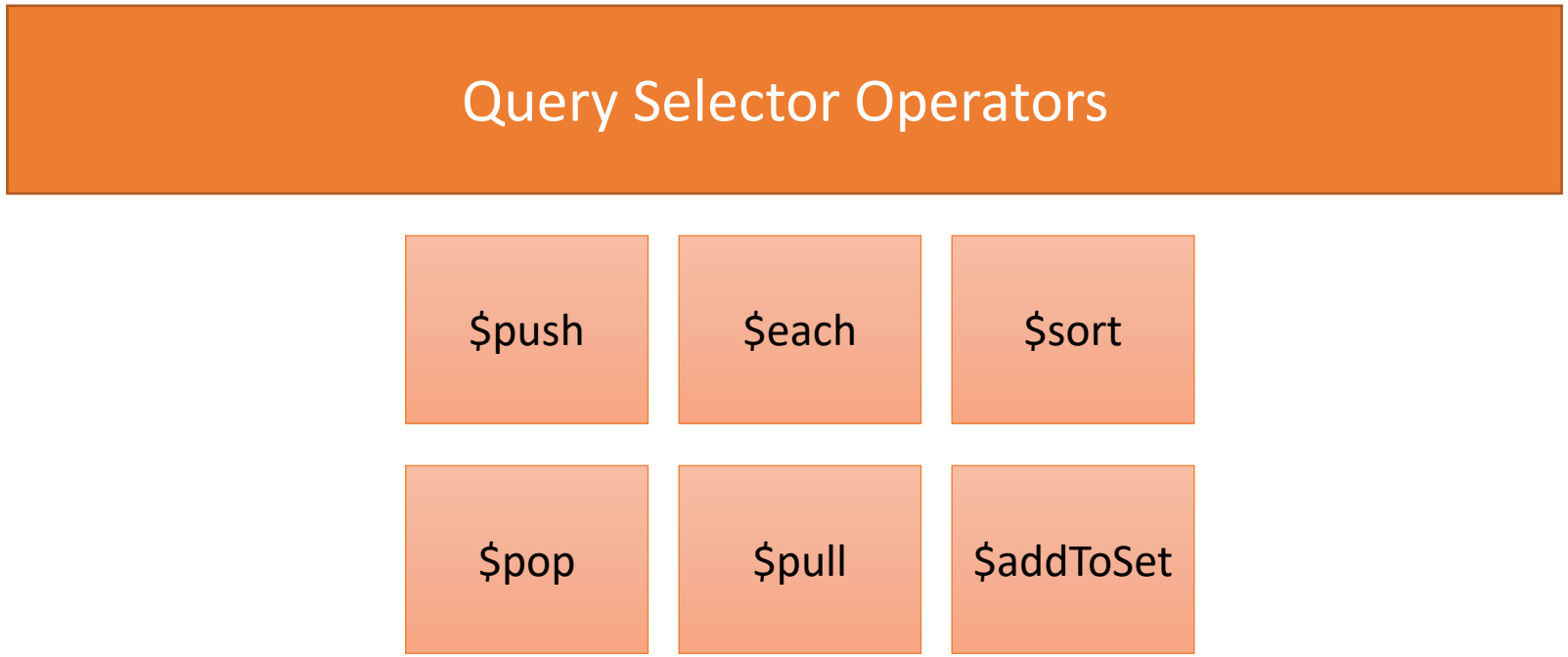
$min, $max, $mul

upsert

Challenge – Update Operation

1	Create a new collection (“sports”) and upsert two new documents into it with these field – ‘title’ and ‘requiredTeam’
2	Update all documents which require team by adding a field with the minimum amount of players required.
3	Update all documents that require a team by increasing the number of required players by 10

More on Update Operation



Indexes

Retrieve Data Efficiently

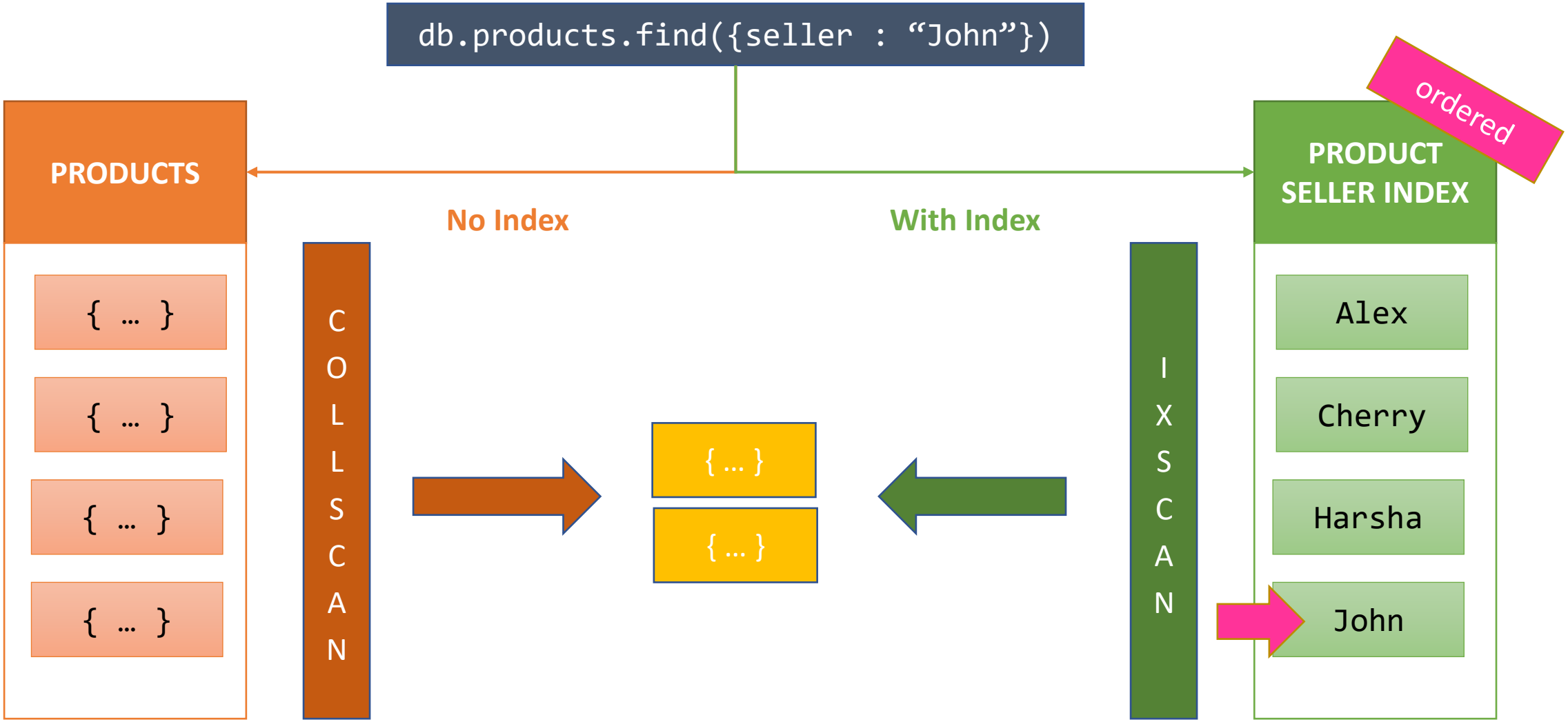
What & Why ?

Indexes allow you to retrieve data more efficiently (if used correctly) because your queries only have to look at a subset of all documents

Indexes don’t come for free, they will slow down your writes

You can use single-field, compound, multi-key (array) and text indexes

Indexes : Working



Security & User Authentication

Lock Down The Data

Security Checklist

- Authentication & Authorization
- Transport Encryption
- Encryption at REST
- Auditing
- Server and Network Config
- Backups and Software Updates

Why Roles ?

Different Types of Database Users

Administrator	Developer / Your App	Data Scientist
Needs to be able to manage the database config, create users etc	Needs to be able to insert, update, delete or fetch data (CRUD)	Needs to be able to fetch data
Does NOT need to be able to insert or fetch data	Does NOT need to be able to create users or manage the database config	Does NOT need to be able to create users, manage the database config or insert, edit or delete data

Built-in Roles

Database User	Database Admin	All Database Roles
read readWrite	dbAdmin userAdmin dbOwner	readAnyDatabase readWriteAnyDatabase userAdminAnyDatabase dbAdminAnyDatabase

Cluster Admin	Backup / Restore	Super User
clusterManager clusterMonitor hostManager clusterAdmin	backup restore	dbOwner(Admin) userAdmin(Admin) userAdminAnyDatabase root

Security Summary

MongoDB uses a Role Based Access Control approach

You create users on databases and you then log in with your credentials

Users have no rights by default, you/admin need to add roles to allow certain operations

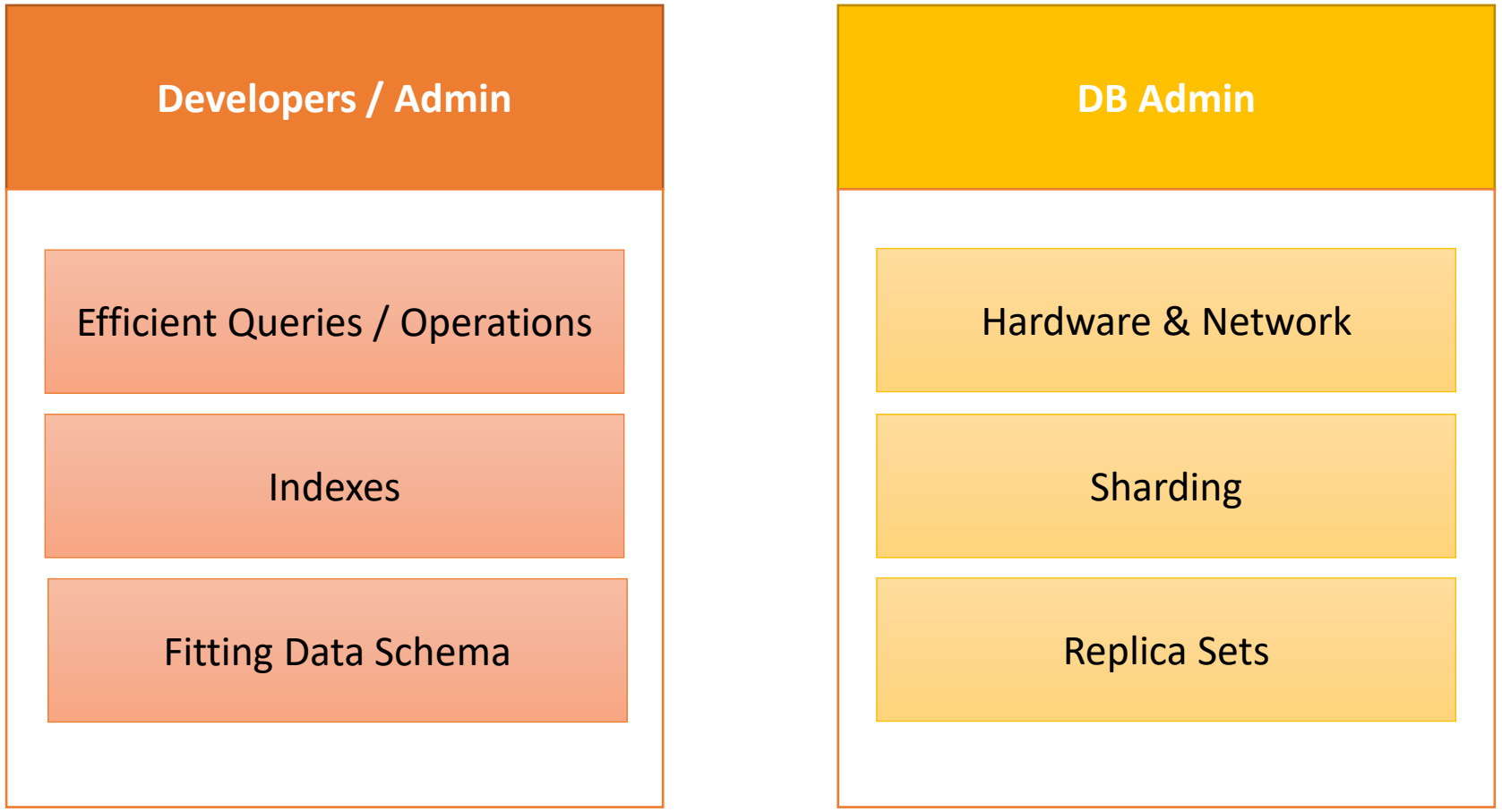
Permissions that are granted by roles (“Privileges”) are only granted for the database the user was added to unless you explicitly grant access toother databases

You can use “AnyDatabase” roles for cross-database access

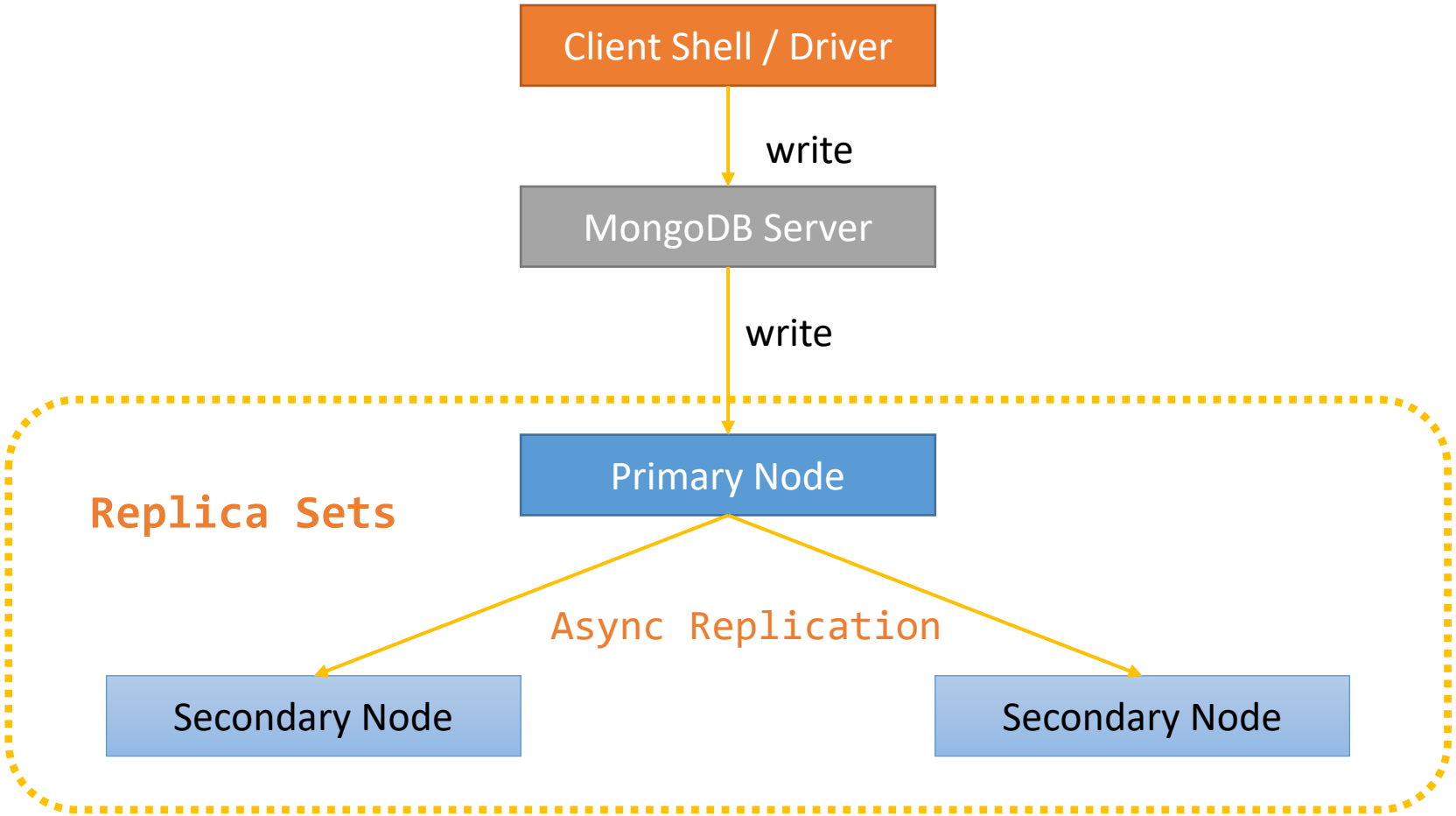
Performance & Fault Tolerance

A Look Into the Admin World

What Influences Performance?



Replica Sets

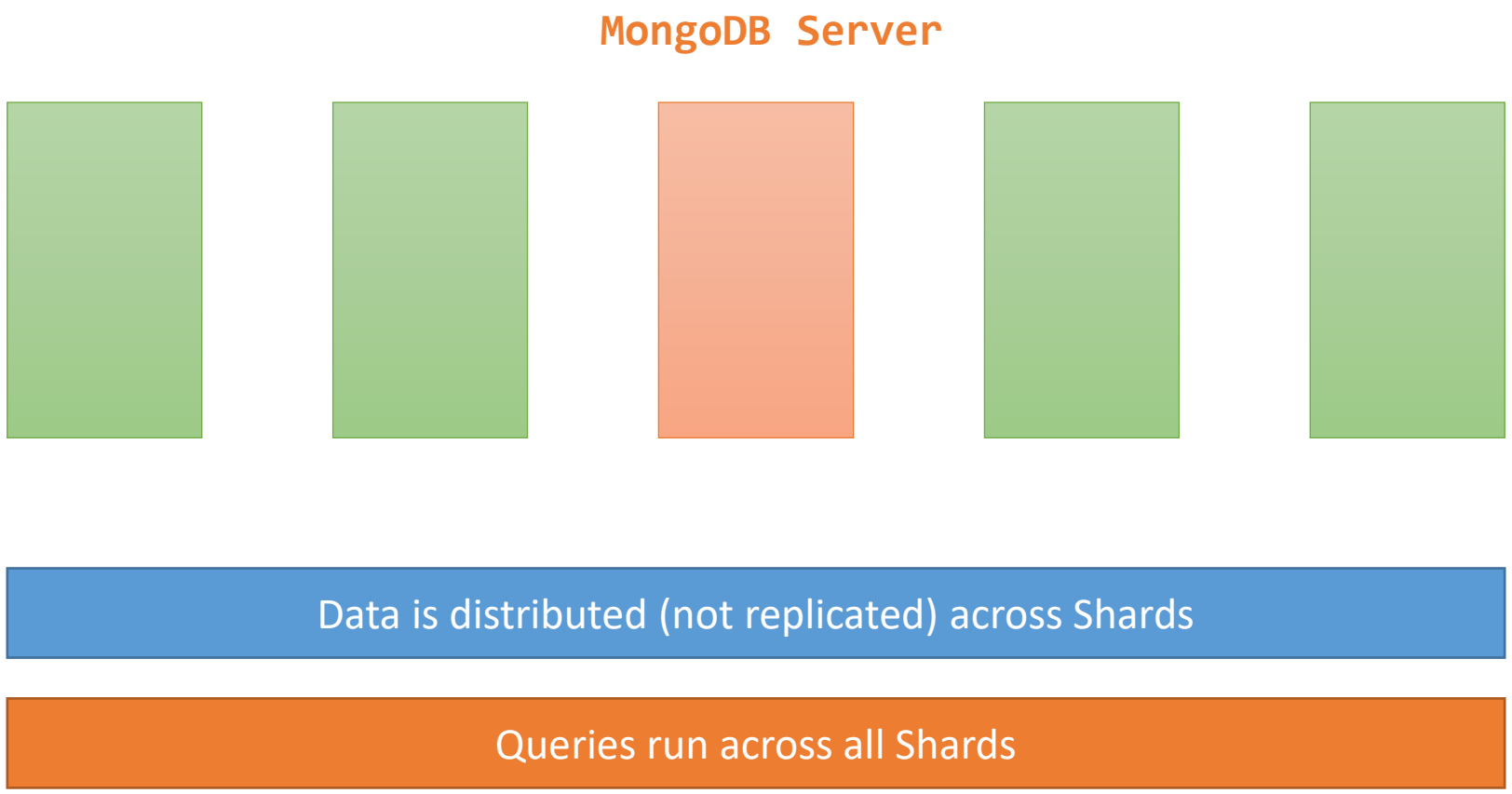


Why Replica Sets ?

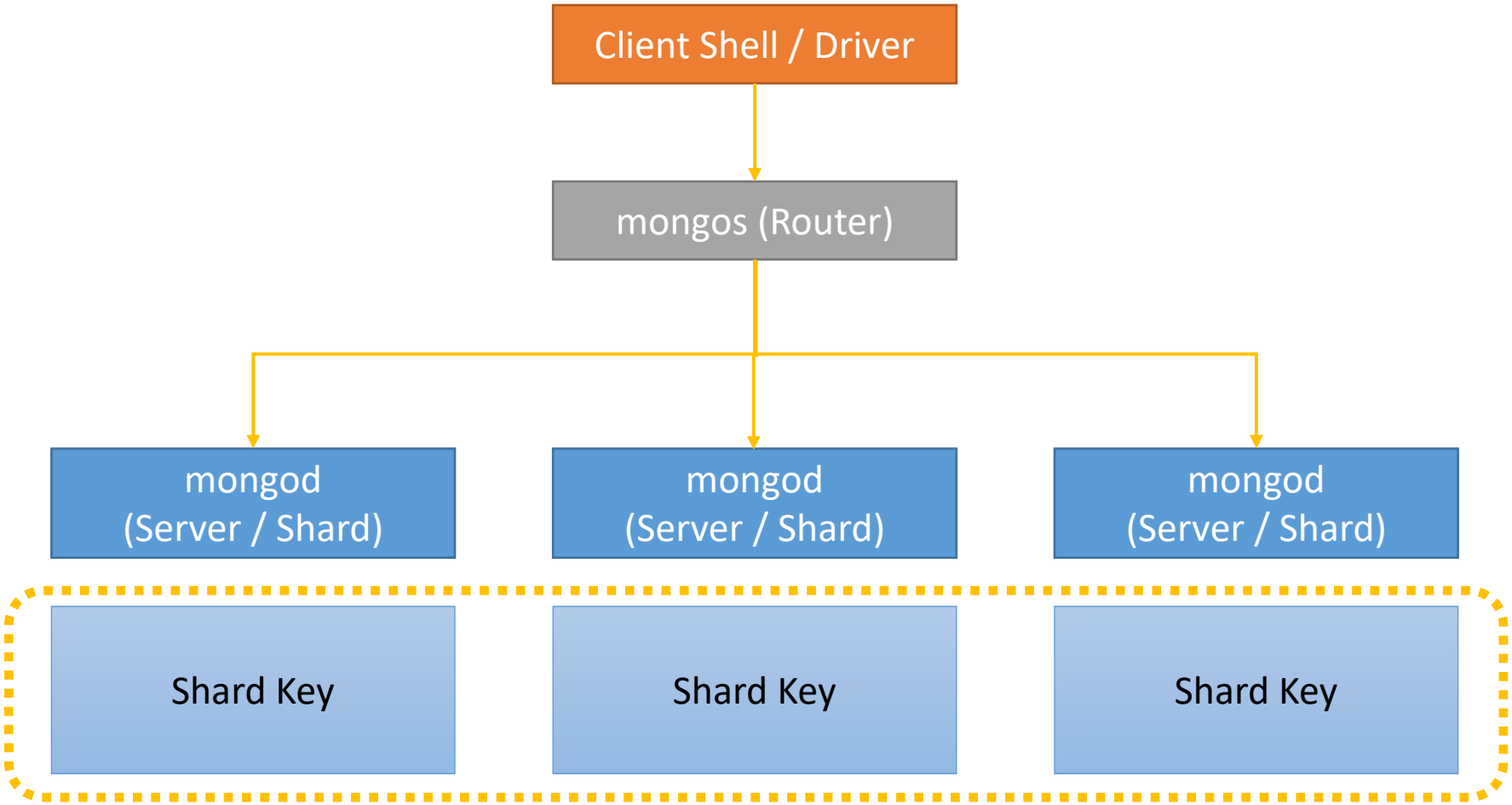
Backup / Fault Tolerance

Improved Read Performance

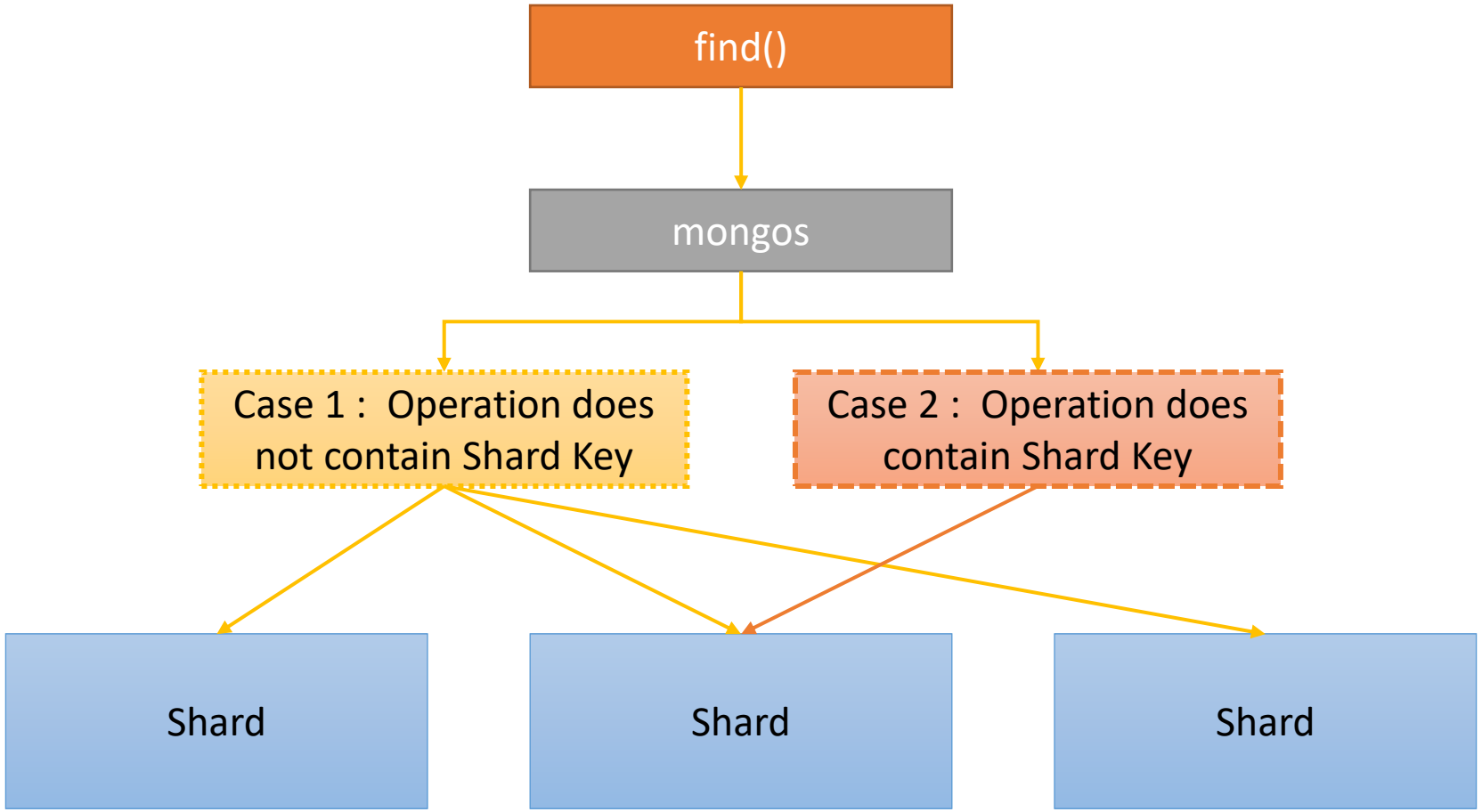
Sharding (Horizontal Scaling)



How Sharding Works ?



How Sharding Works ?



Fault & Tolerance Summary

Consider Capped Collections for cases where you want to clear old data automatically.

Performance is all about having efficient queries/ operations, fitting data formats and a best-practice MongoDB server config.

Replica sets provide fault tolerancy (with automatic recovery) and improved read performance.

Sharding allows you to scale your MongoDB server horizontally.

References

eBooks

- Learning MongoDB – Stack Overflow Contributors
- MongoDB – Notes for Professional

Web

- https://docs.mongodb.com/manual
- MongoDB Course by Maxmillian (Acadmind)