Course MongoDB

Topics (1)

Database models
SQL vs NoSQL
Introduction to MongoDB
CRUD operation
How to design schema of MongoDB
Improve performance

Topics (2)

Administration tasks
Scalable
Security
Monitoring and Operation

Database Models

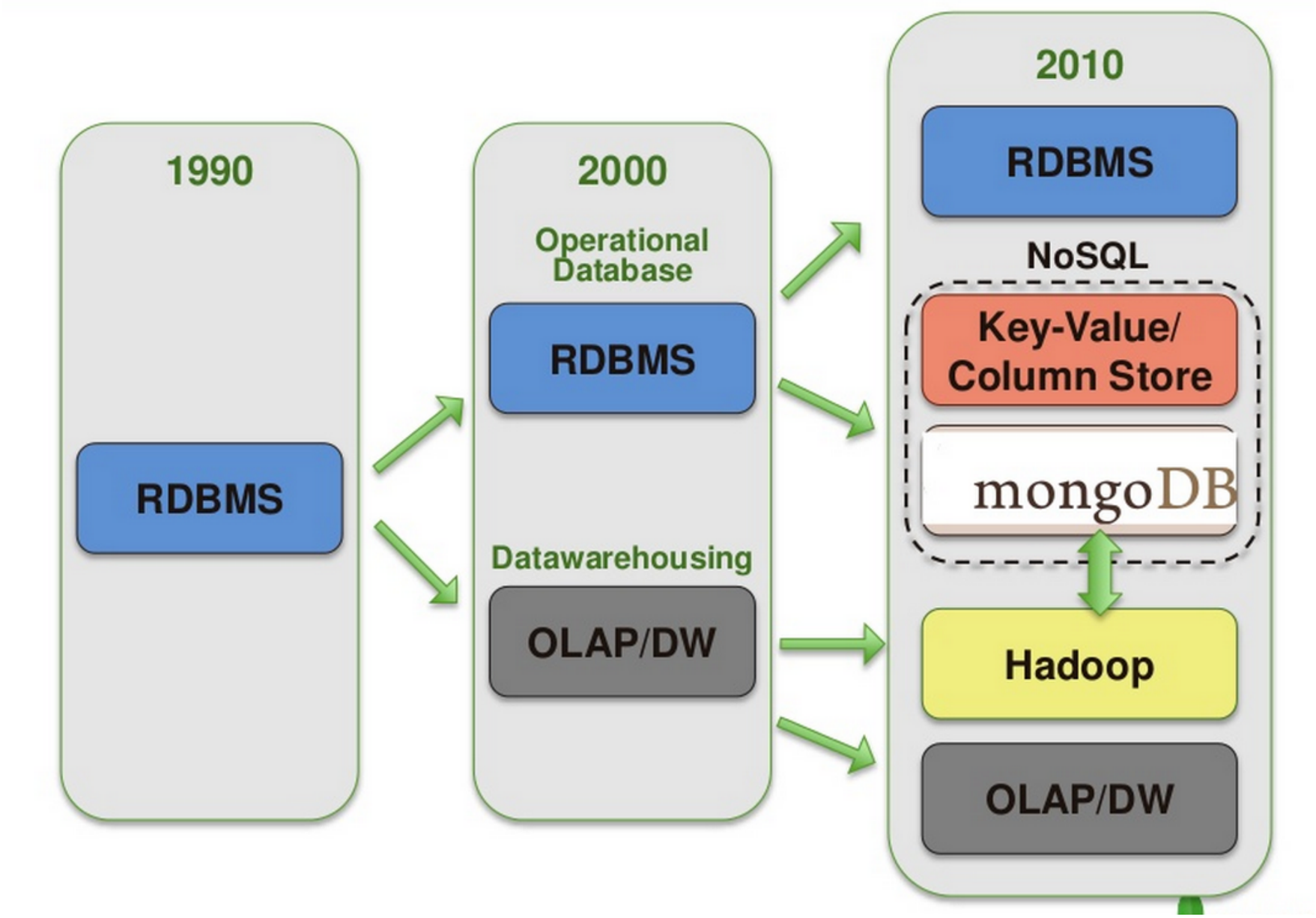
Database ranking

381 systems in ranking, November 2021

Rank			DBMS	Database Model	Score		
Nov 2021	Oct 2021	Nov 2020			Nov 2021	Oct 2021	Nov 2020
1.	1.	1.	Oracle	Relational, Multi-model	1272.73	+2.38	-72.27
2.	2.	2.	MySQL	Relational, Multi-model	1211.52	-8.25	-30.12
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	954.29	-16.32	-83.35
4.	4.	4.	PostgreSQL	Relational, Multi-model	597.27	+10.30	+42.22
5.	5.	5.	MongoDB	Document, Multi-model	487.35	-6.21	+33.52
6.	6.	↑ 7.	Redis	Key-value, Multi-model	171.50	+0.15	+16.08
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	167.52	+1.56	+5.90
8.	8.	8.	Elasticsearch	Search engine, Multi-model	159.09	+0.84	+7.54
9.	9.	9.	SQLite	Relational	129.80	+0.43	+6.48
10.	10.	10.	Cassandra	Wide column	120.88	+1.61	+2.13

https://db-engines.com/en/ranking

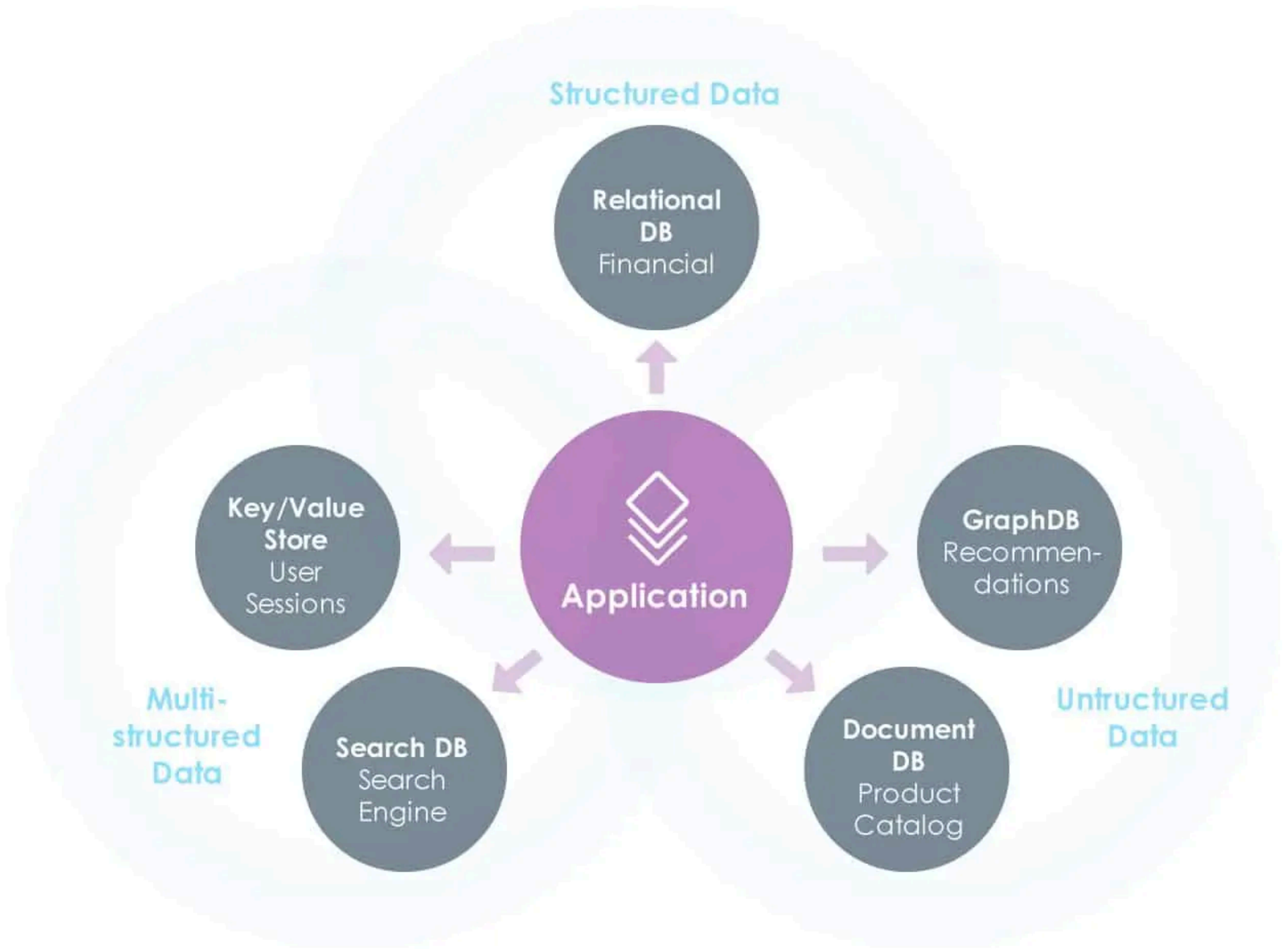
Evolution of database



Polyglot Database

https://martinfowler.com/bliki/PolyglotPersistence.html

Polyglot Database



NoSQL

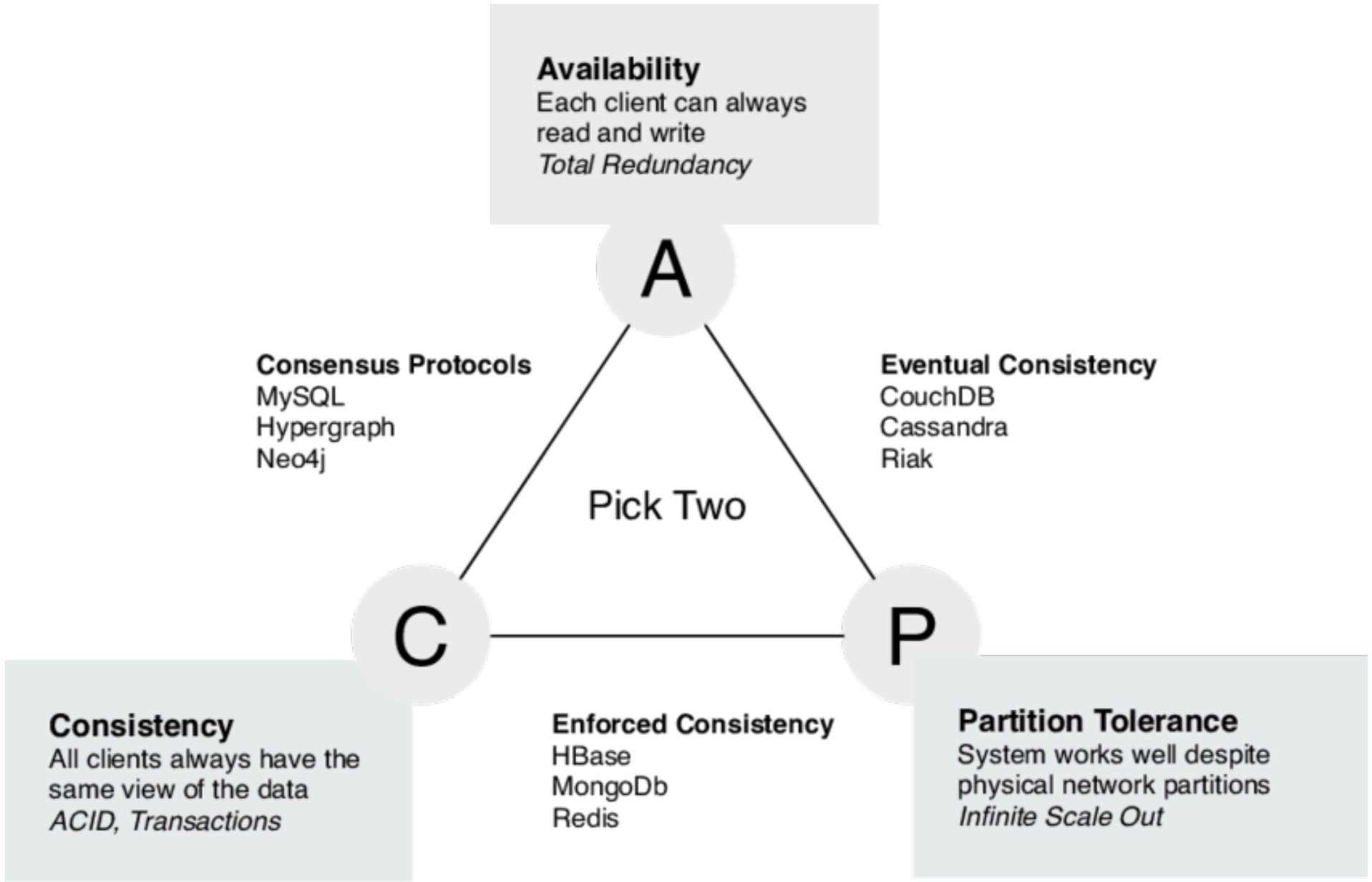
Not Only SQL
Non-relation
No join
No complex transaction
Focus on horizontal scalable
Need new database model

NoSQL World

Document Database	Graph Databases
 Couchbase MarkLogic mongoDB	 InfiniteGraph The Distributed Graph Database
Wide Column Stores	**Key-Value Databases**
   	  HYPERTABLE INC  Cassandra APACHE HBASE Amazon SimpleDB

@cloudtxt http://www.aryannava.com

CAP Theorem


Availability
Each client can always read and write
Total Redundancy
A
Consensus Protocols
MySQL
Hypergraph
Neo4j
Eventual Consistency
CouchDB
Cassandra
Riak
Pick Two
C
P
Consistency
All clients always have the same view of the data
ACID, Transactions
Enforced Consistency
HBase
MongoDb
Redis
Partition Tolerance
System works well despite physical network partitions
Infinite Scale Out

MongoDB

MongoDB

Humongous

hu·mon·gous
/(h)yo͞o'mäNGgəs,-'məNG-/

adjective NORTH AMERICAN *informal*

huge; enormous.
"a humongous steak"

MongoDB

Open sources
Scalable
High-performance
Schema-free, dynamic schema

Document-oriented database

MongoDB

NoSQL documentDB
Stored in collections

Document

Way to organize and store data
Set of field-values pairs

```
{
    "id": 1,
    "name": "mongodb",
    "age": 30
}
```

JSON format

JavaScript Object Notation
Start and end with **{ }**
Separate each key and value with **:**
Separate each key-value with **,**

Text-based format

JSON use cases

APIs
Configuration files
Log message
Database storage

JSON as database

Text-based format (passing is very slow)
Less space-efficient
Only support a limited number of data types

BSON

Binary JSON
Bridges the gap between binary and JSON format

https://bsonspec.org/

BSON Goals

Speed
Space
Flexibility

Designed for high performance

JSON vs BSON

	JSON	BSON
Encoding	UTF-8 string	Binary
Data types	String, Boolean, Number, Array	String, Boolean, Number, Array, Date, Binary
Readability	Human and Machine	Machine only

https://www.mongodb.com/json-and-bson

Summary

BSON => store

MongoDB stores data in BSOB, internally and over the network

JSON => view

Can be natively stored and retrieved in MongoDB

https://bsonspec.org/

RDBMS vs MongoDB

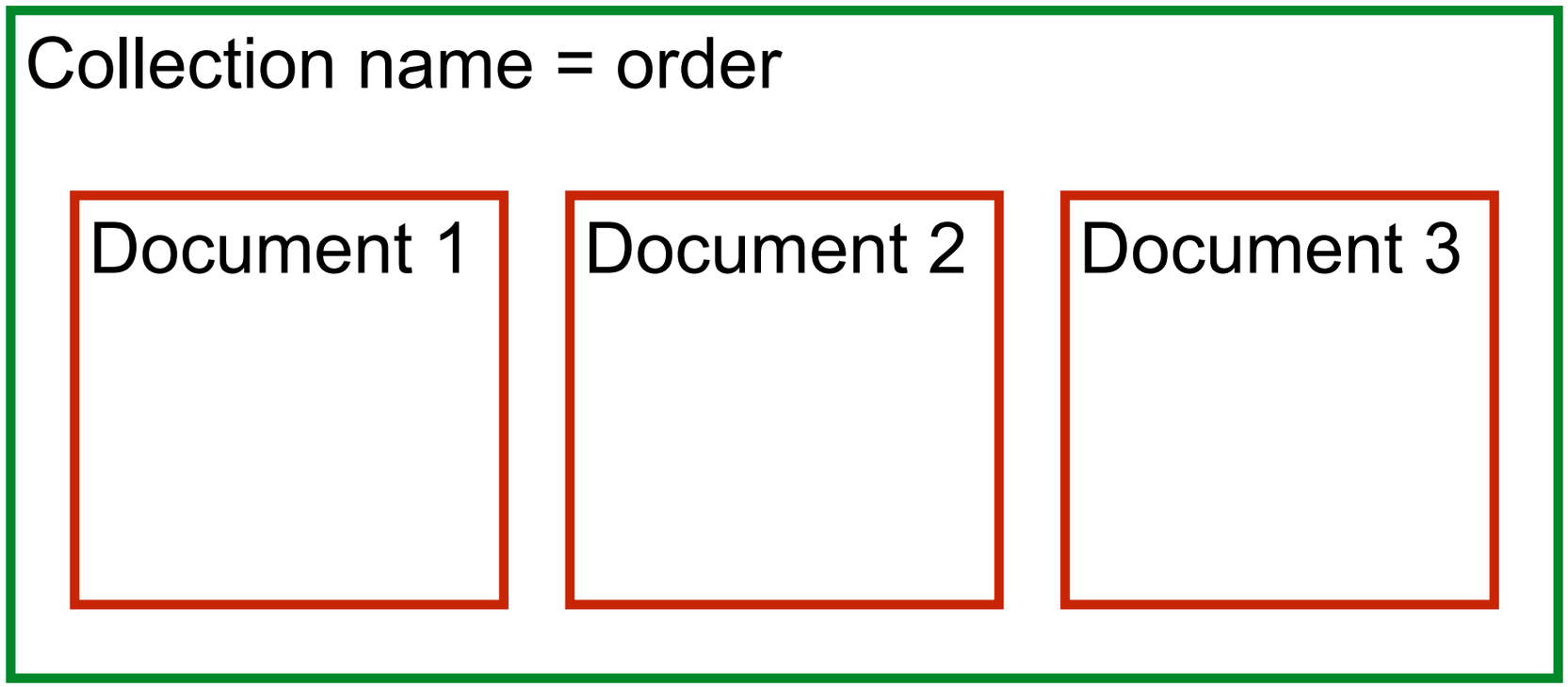
Database	Table	Row	Column

Database	Collection	Document (JSON, BSON)	Field

https://docs.mongodb.com/manual/reference/sql-comparison/

Collection in MongoDB

An organized store of documents



Collections types

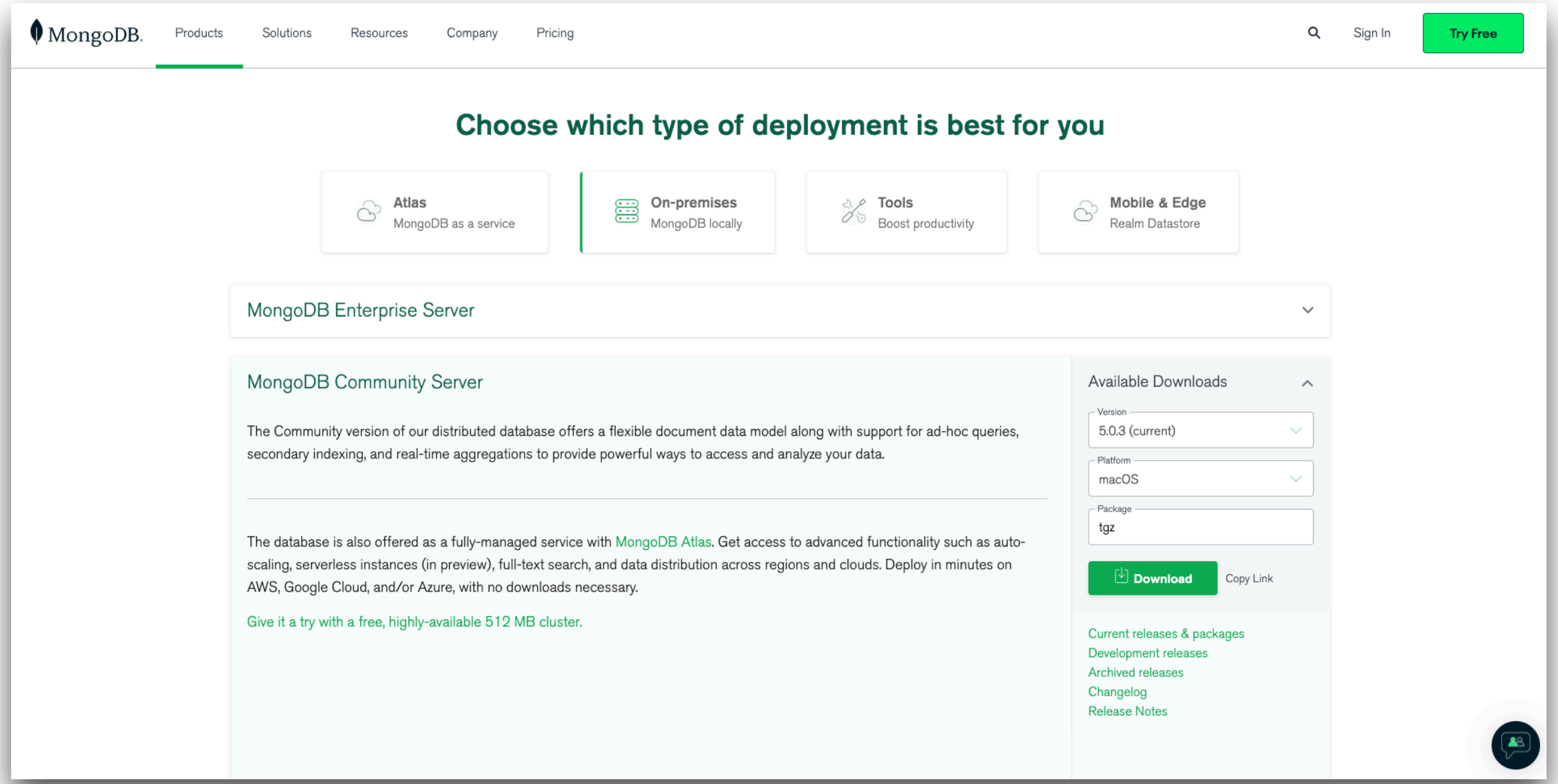
Default
Views
On-demand material views
Capped collection (fixed-size)
Time series collection

Installation

Choices

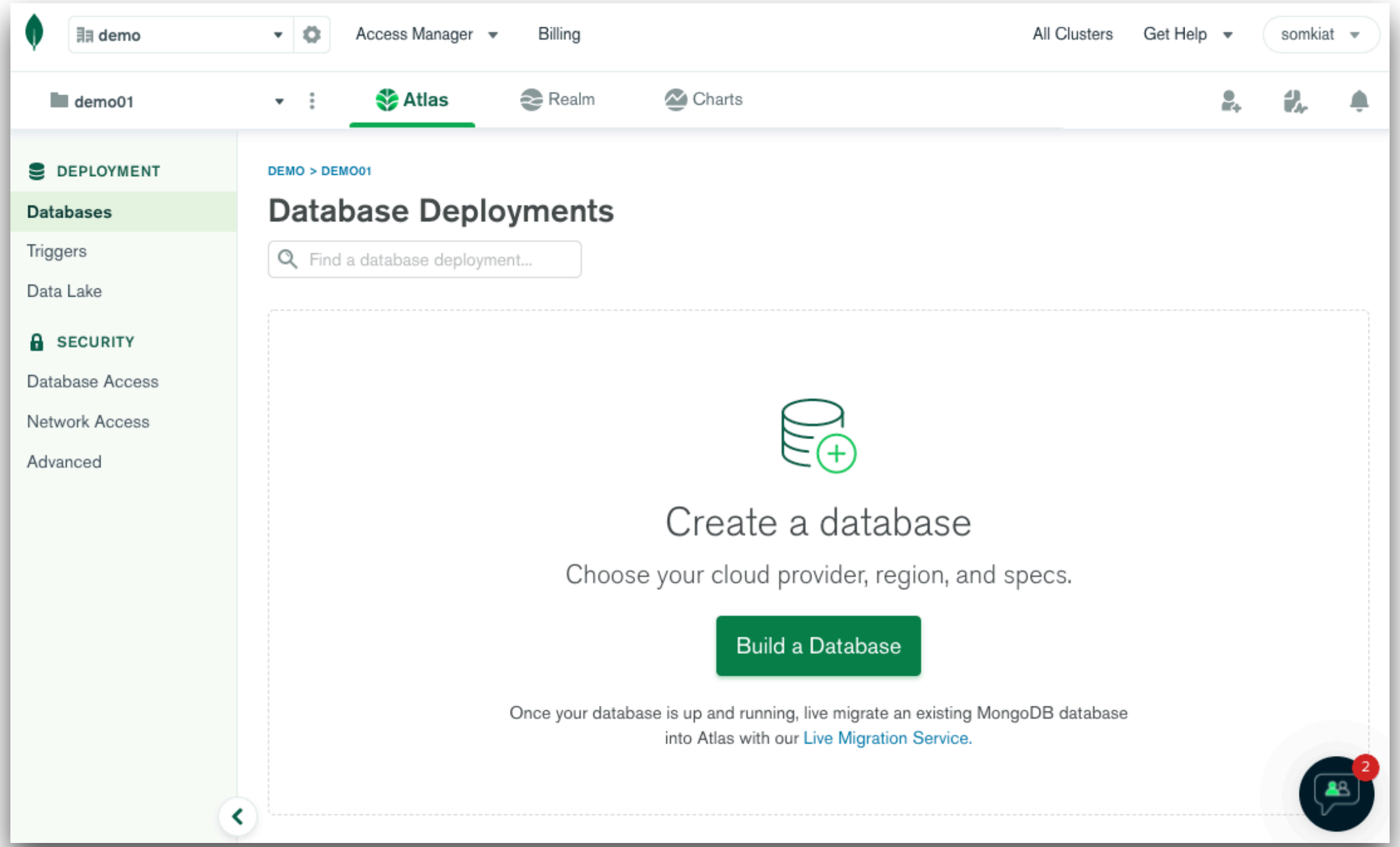
On premise
On cloud

On premise



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

On cloud (Atlas)



CRUD operations
In mongo shell

CRUD operations

Operation	Example
Create	insertOne() insertMany()
Read	find() - criteria, limit, sorting
Update	updateOne() updateMany() replaceOne()
Delete	deleteOne() deleteMany()
Upsert (Update or insert)	updateOne(… “upset”: true)

Build write operations

Write one or more documents
Transaction per document
Ordered vs Unordered operation

Use bulkWrite()

Upsert ?

If `upsert` is `true`

Is there a match?

Update the matched document

Is there a match?

Insert a new document

Workshop

MongoDB Query Language (MQL)

Query abd projection operator
Update operator
Comparison operator
Aggregation pipeline

Comparison operator

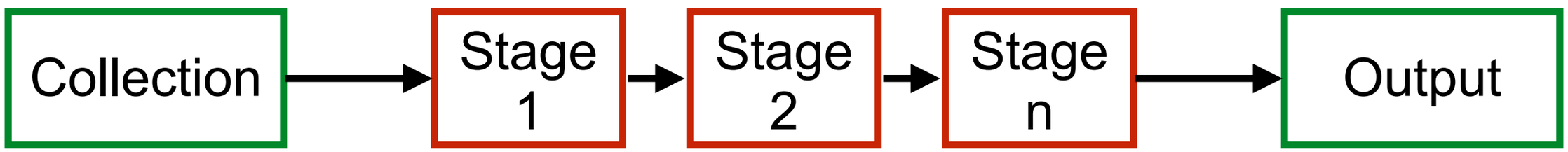
Operator	Description
$eq	Equal to
$ne	Not equal to
$gt	Greater than
$gte	Greater than or equal
$lt	Less than
$lte	Less than or equal

Update operator

Modify data in database
$inc, $set and $unset

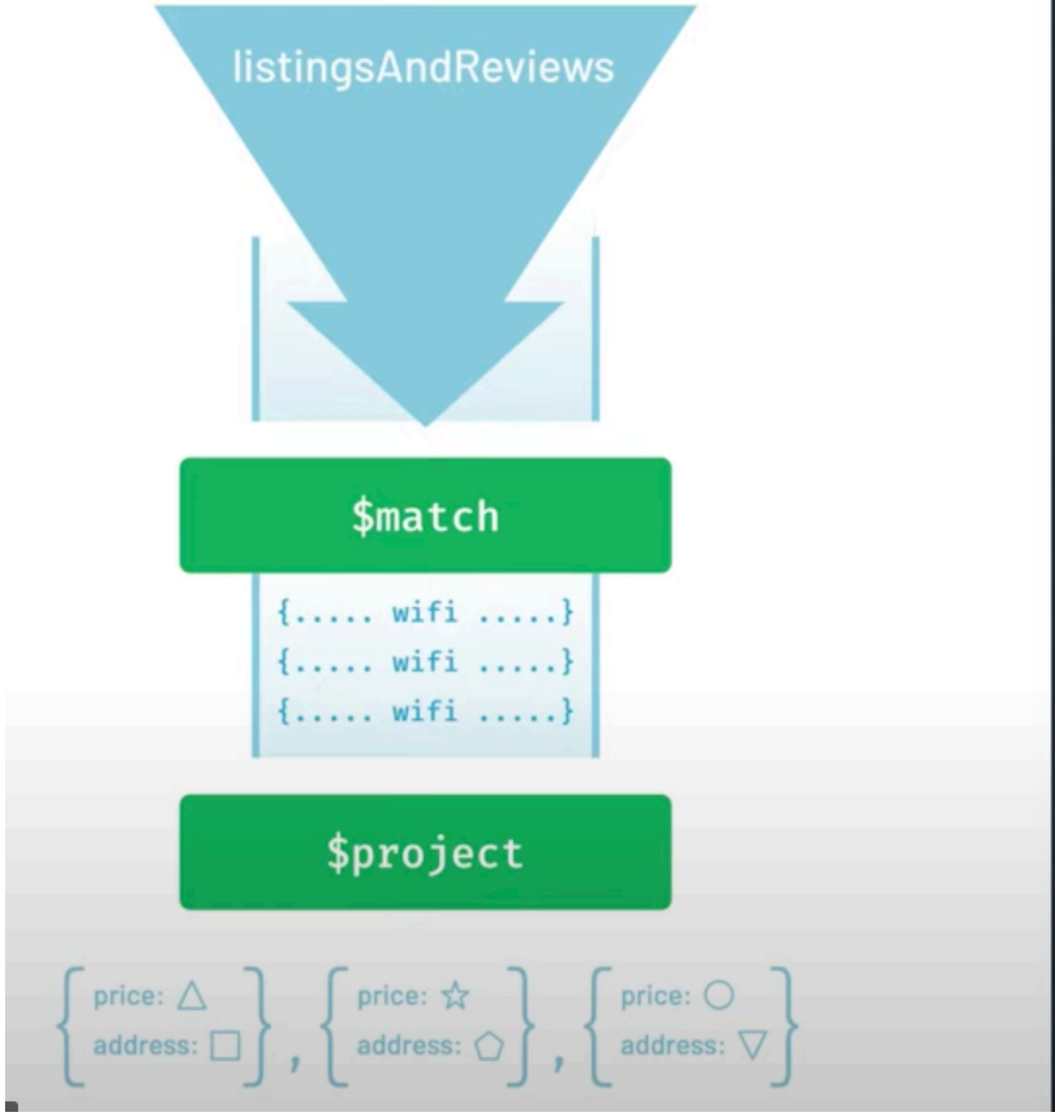
Aggregation pipeline

Another way to query data in MongoDB
Powerful to transform data
Break down a complex query in to easier stages



https://docs.mongodb.com/manual/core/aggregation-pipeline/

Aggregation pipeline




```
[

 { $match:
   {"amenities":"Wifi"}
  },

 { $project:
    {"price": 1,
    "address": 1,
    "_id": 0 }
 }

]
```

Syntax

```
Cluster0-shard-0:PRIMARY> db.solarSystem.aggregate([{
...    $match: {
...      atmosphericComposition: { $in: [/O2/] },
...      meanTemperature: { $gte: -40, $lte: 40 }
...    }
... }, {
...    $project: {
...      _id: 0,
...      name: 1,
...      hasMoons: { $gt: ["$numberOfMoons", 0] }
...    }
... }], { allowDiskUse: true}
... )
```

https://docs.mongodb.com/manual/meta/aggregation-quick-reference/

Workshop

Data modeling

https://docs.mongodb.com/manual/core/data-modeling-introduction/

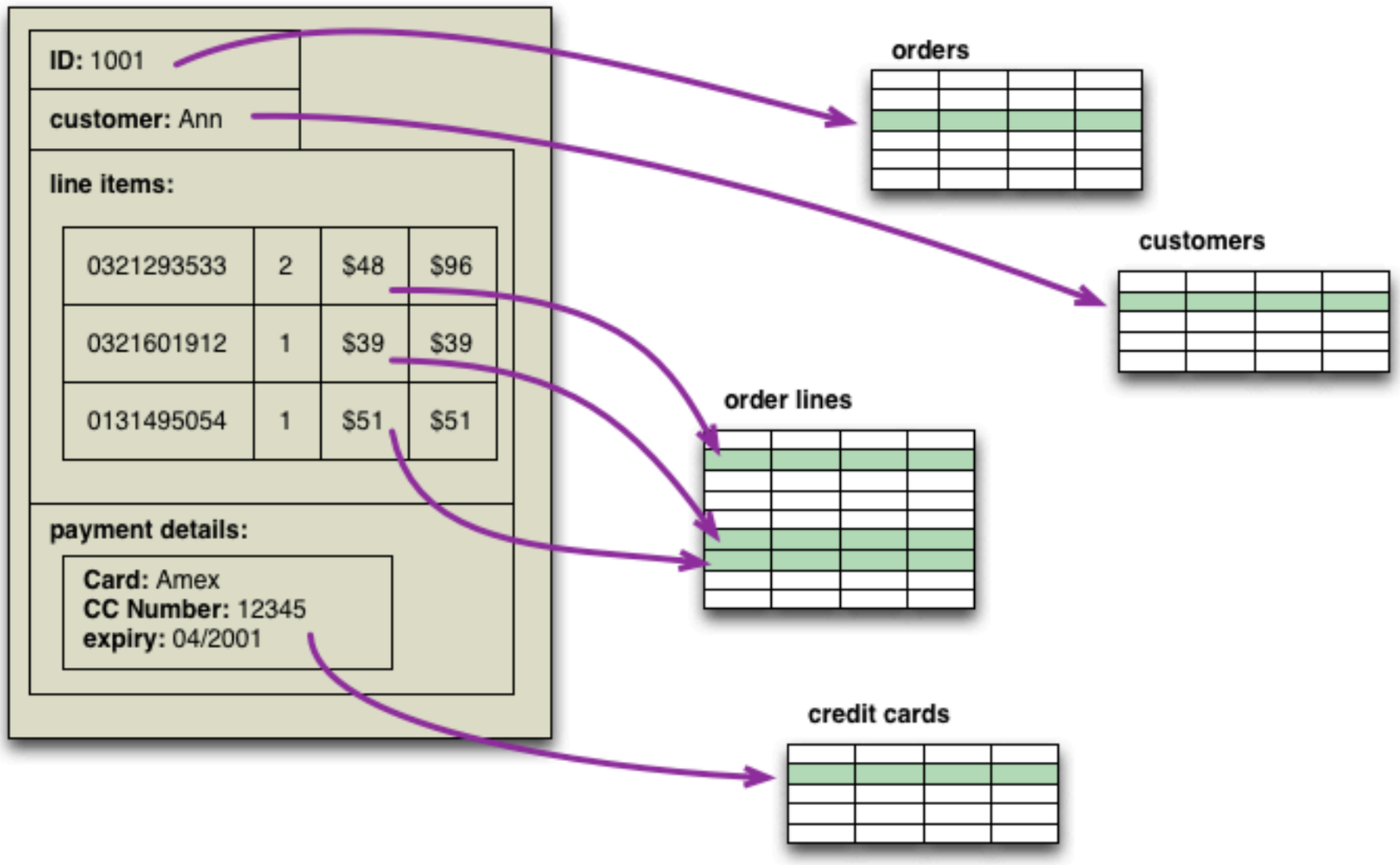
Data modeling in MongoDB

Way to organize fields in a document
To support your application **performance**
And **query** capabilities

Design for read !!

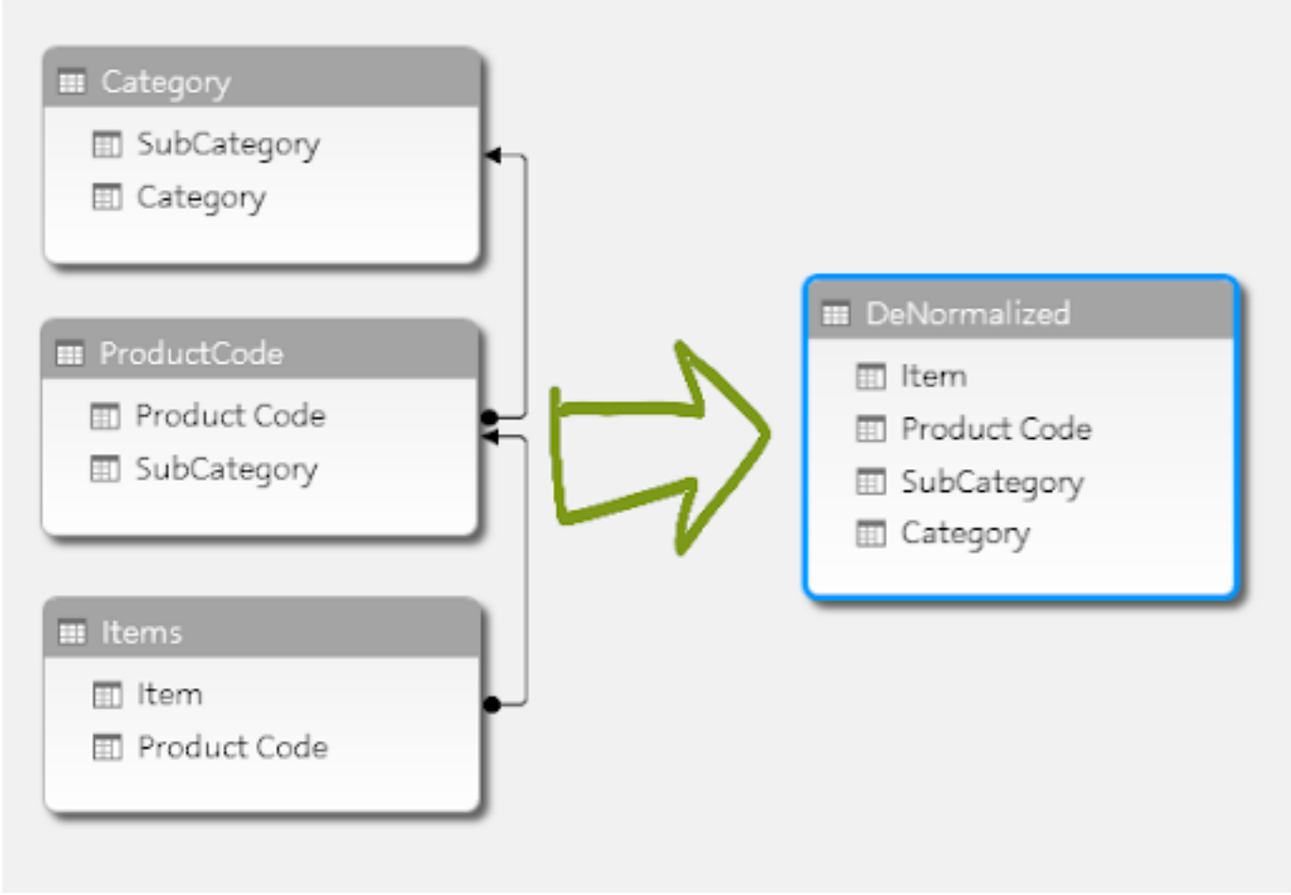
Normalization

Design for **write** or **read** ?



Denormalization

Design for **write** or **read** ?



Comparison

Normalization	Denormalization
Non-redundant and consistency data	Combine data that can be query more speed
More tables and joins	Decrease tables and joins
Optimized usage	Wastage
No redundant data	More redundant data
Design for write	Design for read

MongoDB Rule

Data is stored in the way
that it is used

More questions ?

How it will be queried ?
Who is using our application ?
Ratio between reads and writes ?

Design for users

Keep in mind

Data that is used together should be stored together

Evolving application implies an evolving data model too

Performance issues

Data modeling phases

Workload (data size, read/write)
Relationships
Patterns (optimization)

Document structures

Embedded data
References

Relationships

One-to-one

One-to-many

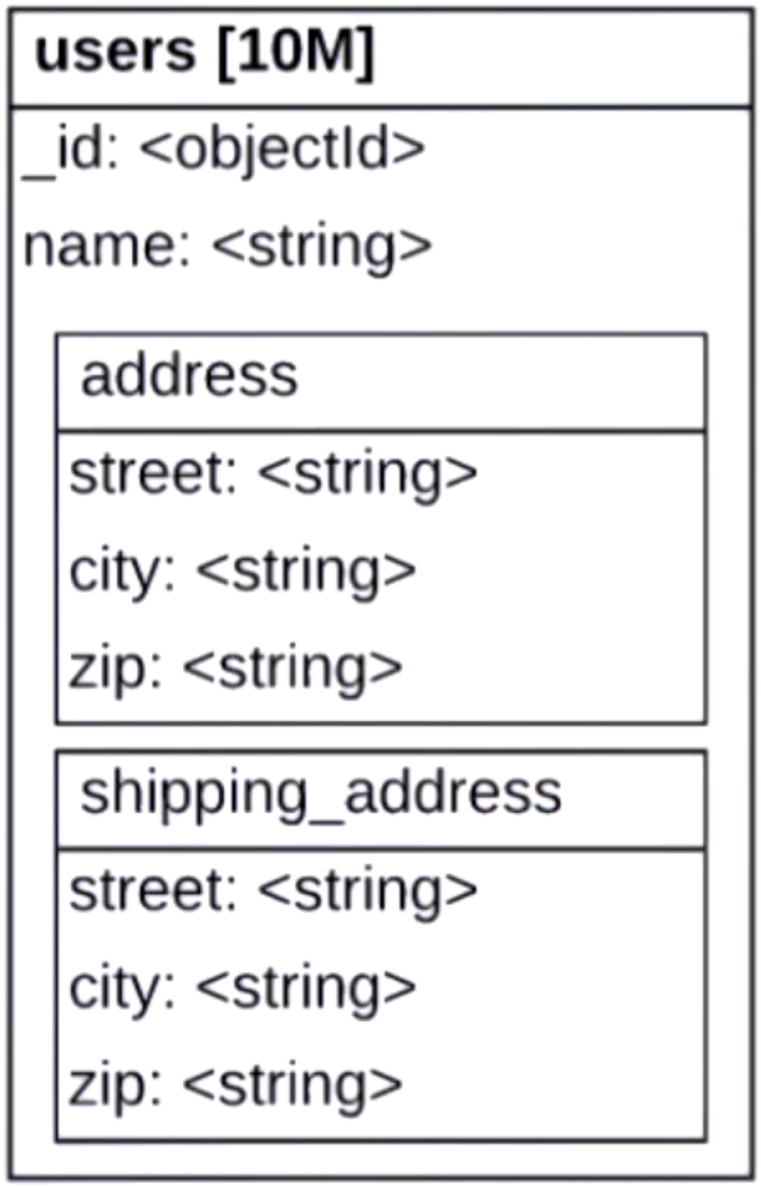
Many-to-many

One-to-One

One-to-One

Embed vs Reference ?

One-to-One

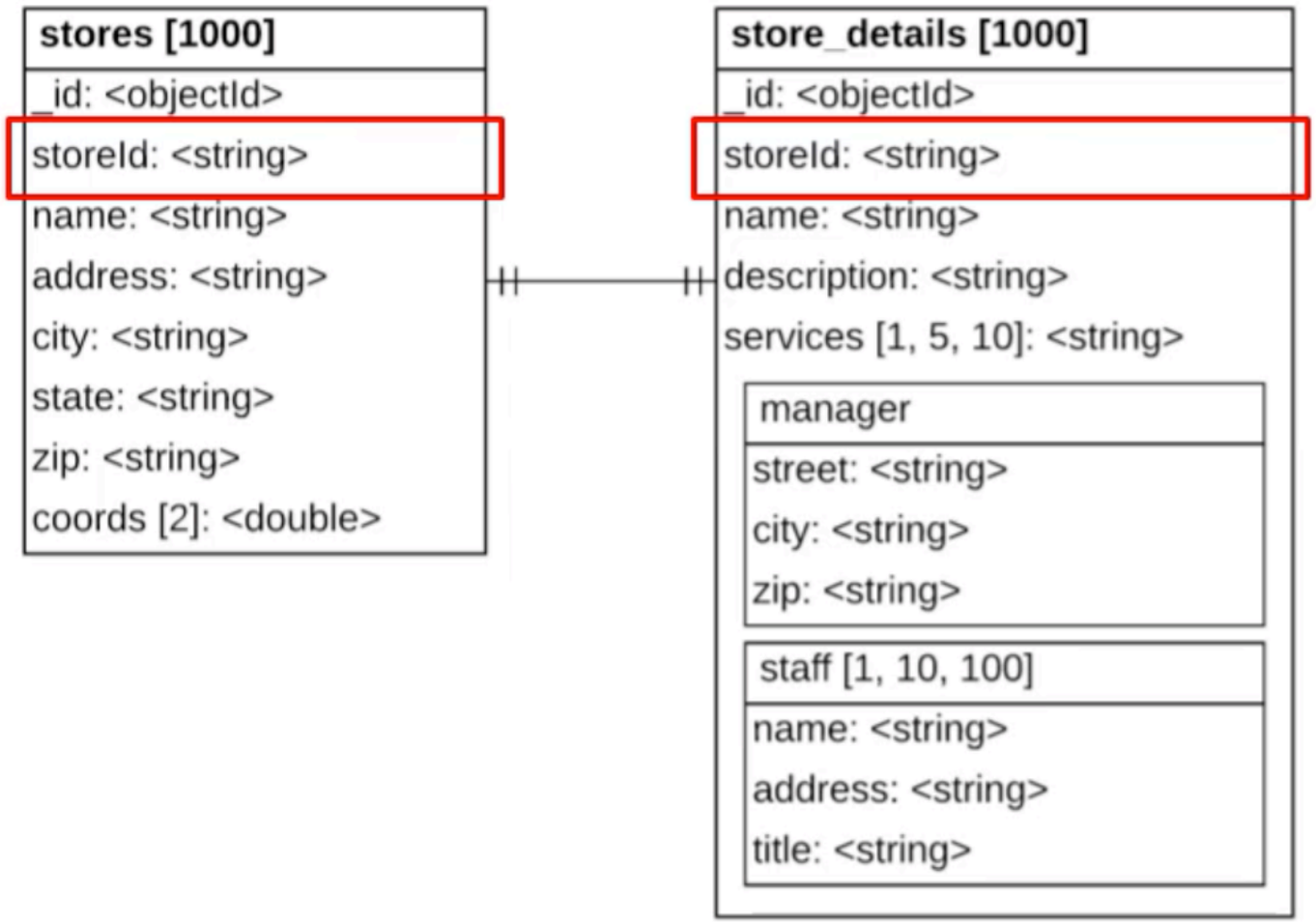


Embed vs Reference ?

One-to-One

Embed vs Reference ?

One-to-One



Embed vs Reference ?

One-to-One

Prefer embedding over reference for simplicity
Use subdocuments to organize the fields
Use reference for optimization purpose

One-to-Many

One-to-many !!

A mother and children ?
A person and address ?

Embed vs Reference ?

One-to-many

One-to-zillion
One-to-many (low)

One-to-Many

Embed vs Reference ?

One-to-Many

Embed vs Reference ?

One-to-Many

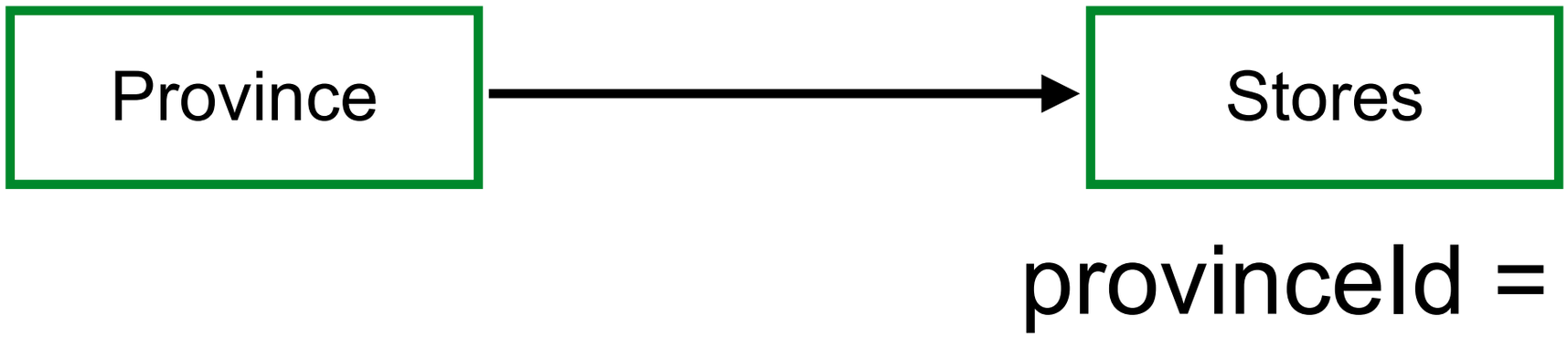
Embed vs Reference ?

One-to-Many

Stores = [1,2,3,4]

Embed vs Reference ?

One-to-Many



Embed vs Reference ?

One-to-Many

Stores = [1,2,3,4]

provinceId = 1

Embed vs Reference ?

One-to-Many

A Lot of choices
Choose the side between **one** and **many**
Duplication when use embedding
Embed on the side of the most query

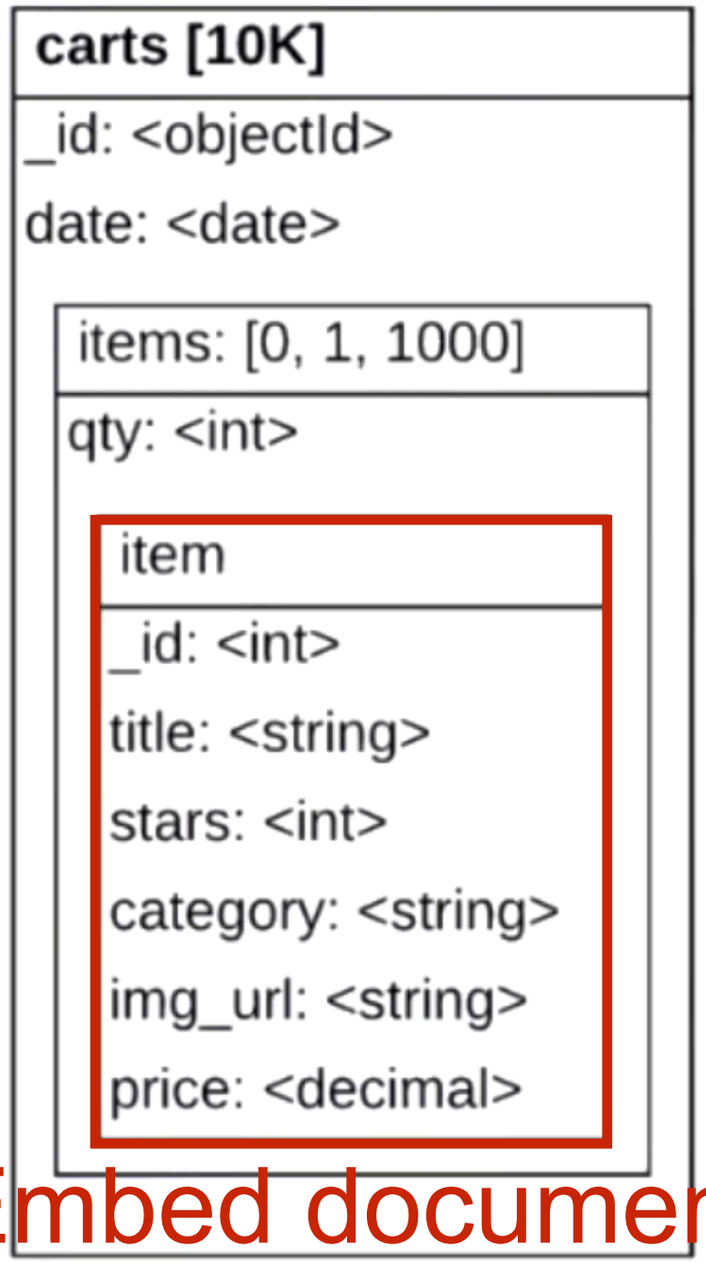
Prefer referencing when associated documents are not always needed with often query documents

Many-to-Many

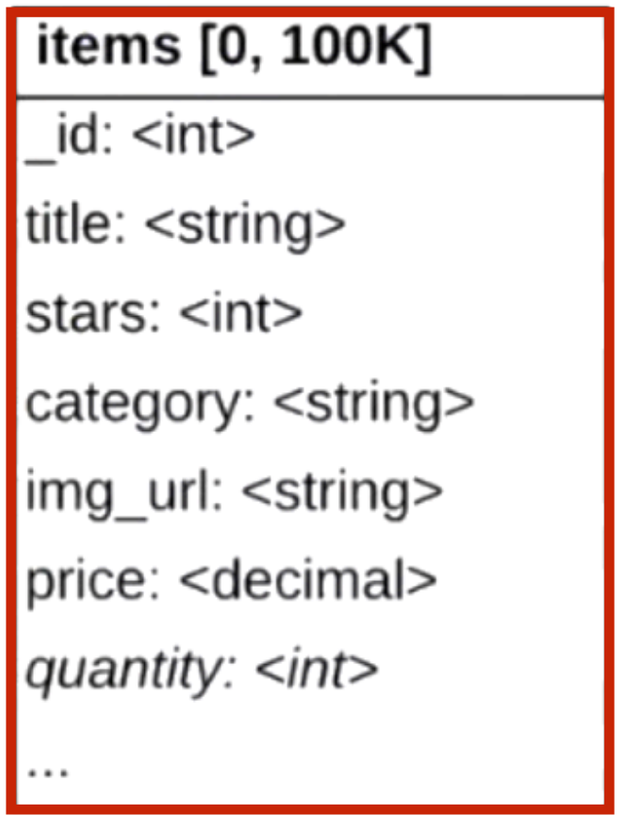
Many-to-Many

Embed vs Reference ?

Many-to-Many



Embed document



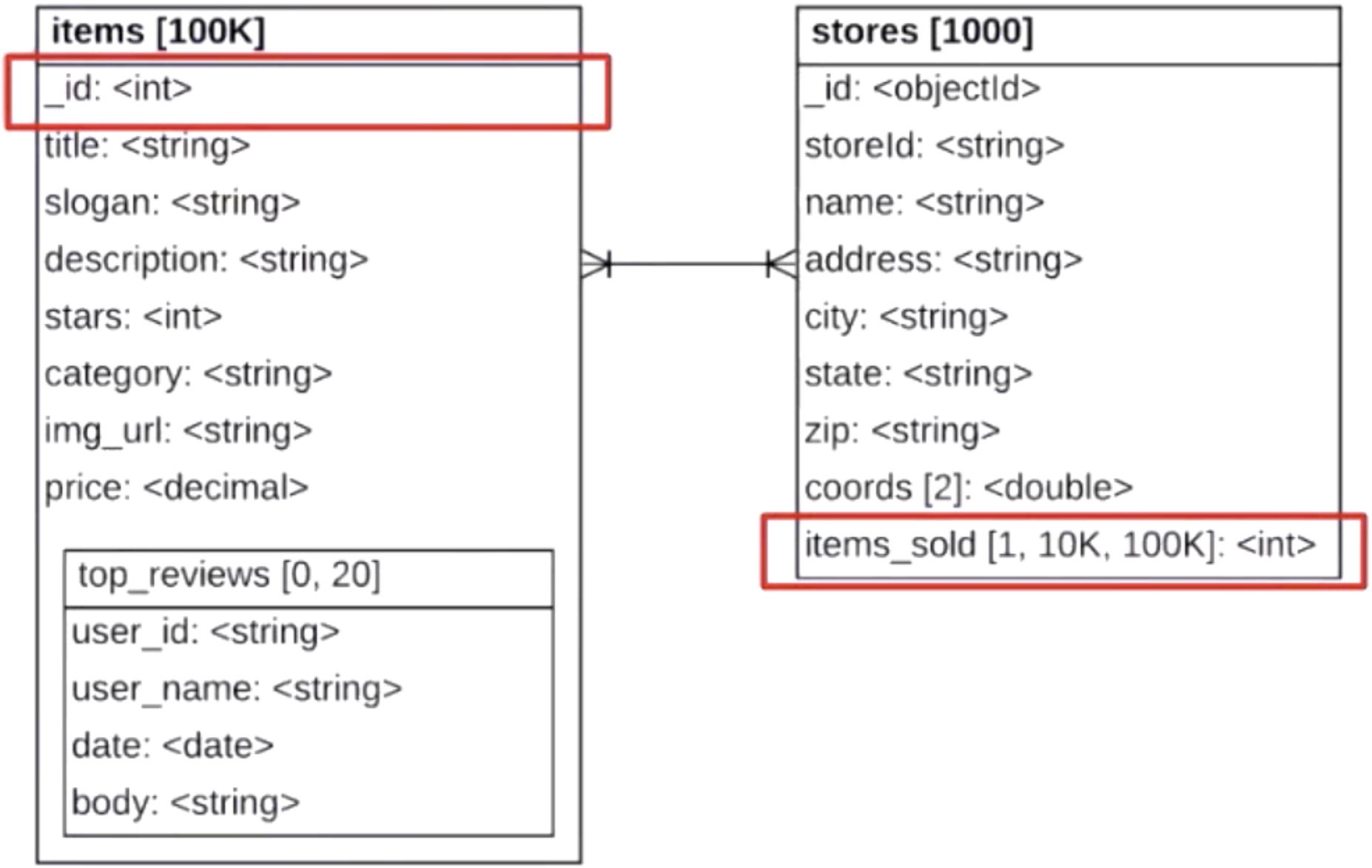
Keep in other collection

Many-to-Many

Find all items sold in each store ?

Embed vs Reference ?

Many-to-Many


items [100K]
_id: <int>
title: <string>
slogan: <string>
description: <string>
stars: <int>
category: <string>
img_url: <string>
price: <decimal>
top_reviews [0, 20]
user_id: <string>
user_name: <string>
date: <date>
body: <string>
stores [1000]
_id: <objectId>
storeId: <string>
name: <string>
address: <string>
city: <string>
state: <string>
zip: <string>
coords [2]: <double>
items_sold [1, 10K, 100K]: <int>

Many-to-Many

Prefer reference over embedding to avoid duplication

Prefer embedding on the most query side

Workshop

MongoDB Deployment

Goals

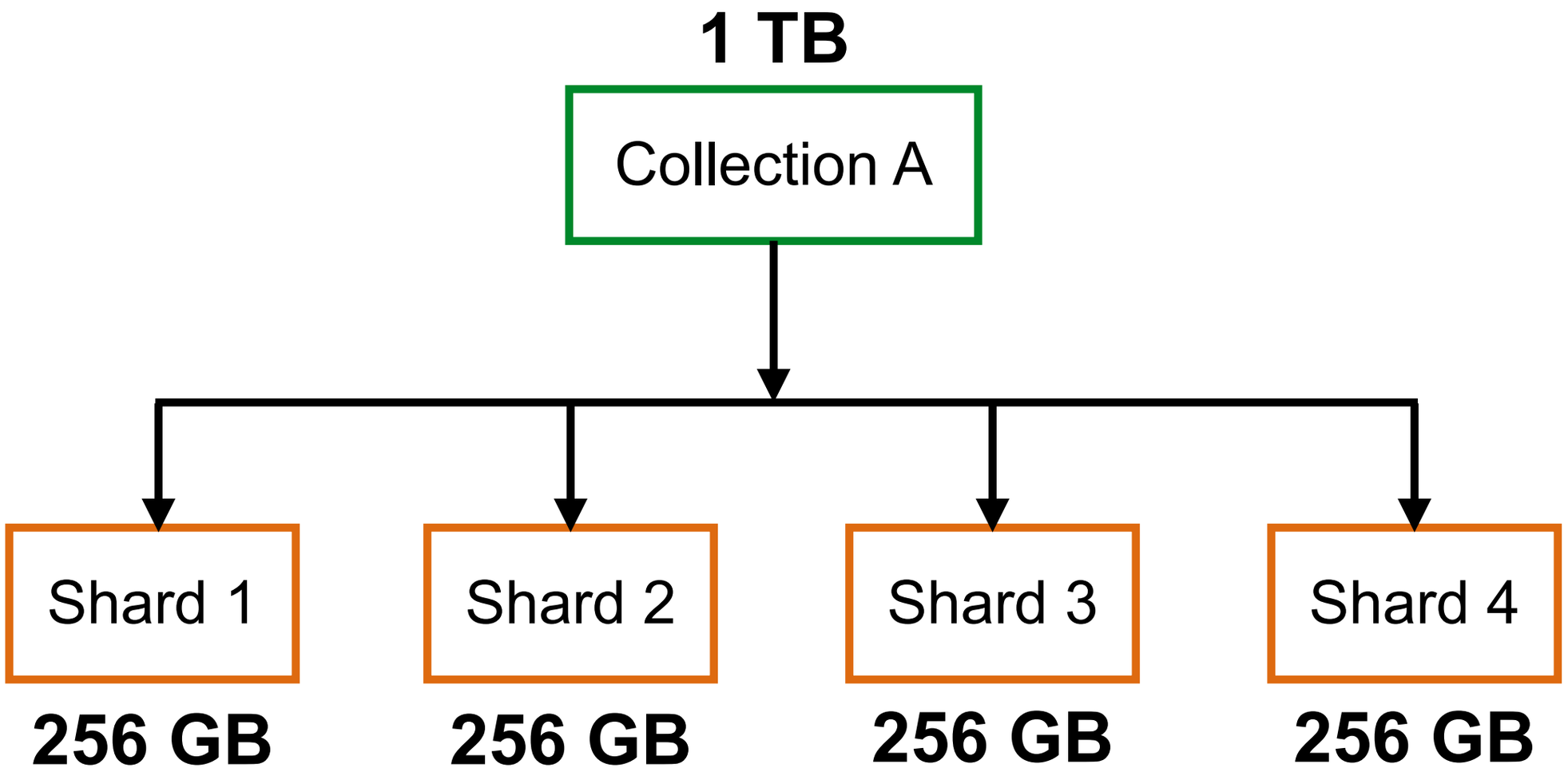
Architecture of a **sharded** cluster
Query handling in a **sharded** cluster
Data distribution method

Sharding

Distributing data across multiple machines
Horizontal scaling

In MongoDB with large data sets, needed **high throughput** operations

Sharding


1 TB
Collection A
Shard 1
Shard 2
Shard 3
Shard 4
256 GB
256 GB
256 GB
256 GB

When to use Shards ?

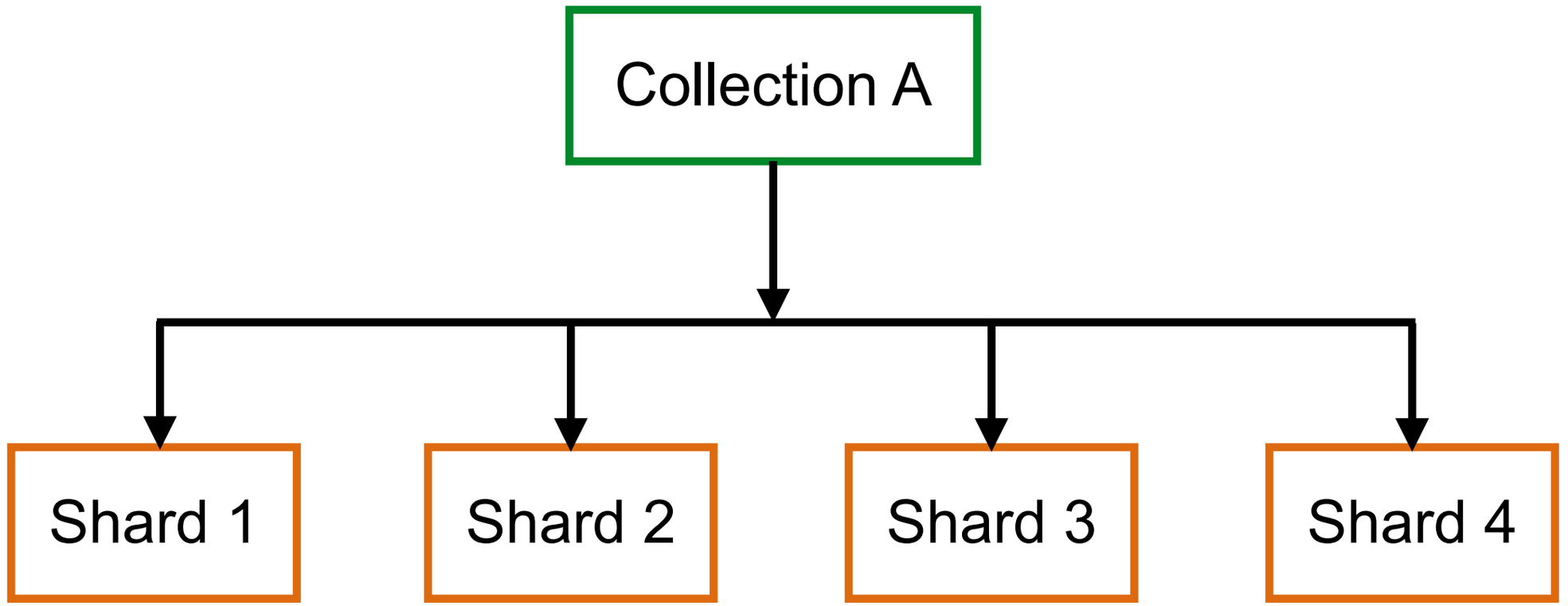
Write workload on a single server exceed that server's capacity

Working set no longer fits into RAM

Single server can no longer handle the size of data set

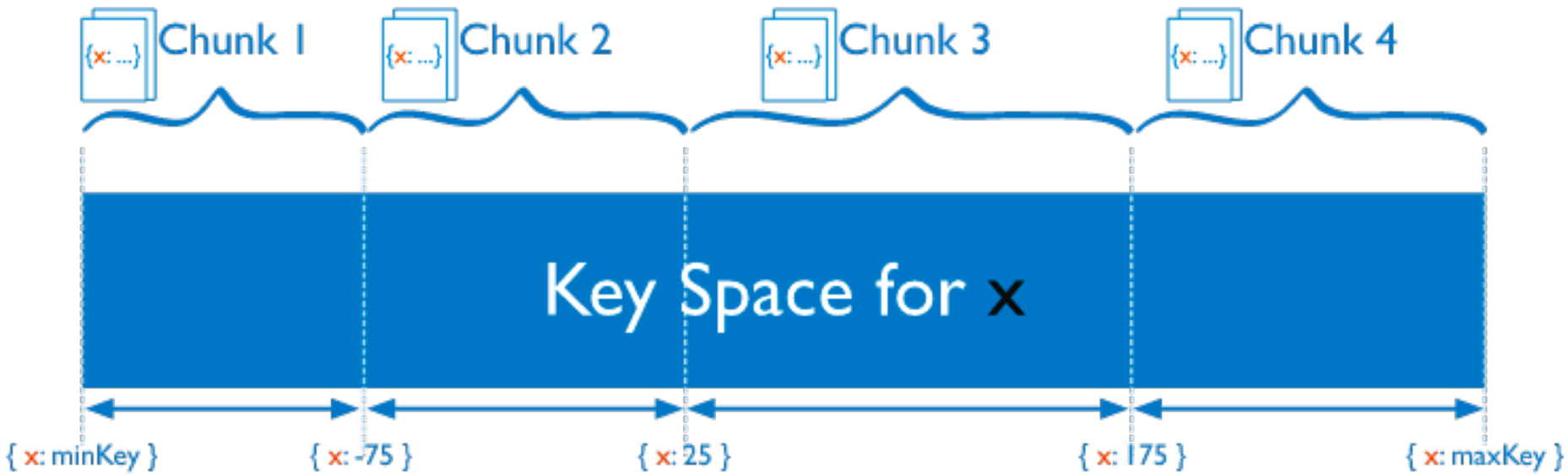
Shard key

Use to determine how documents in a collection are distributed across the shards



Shard key

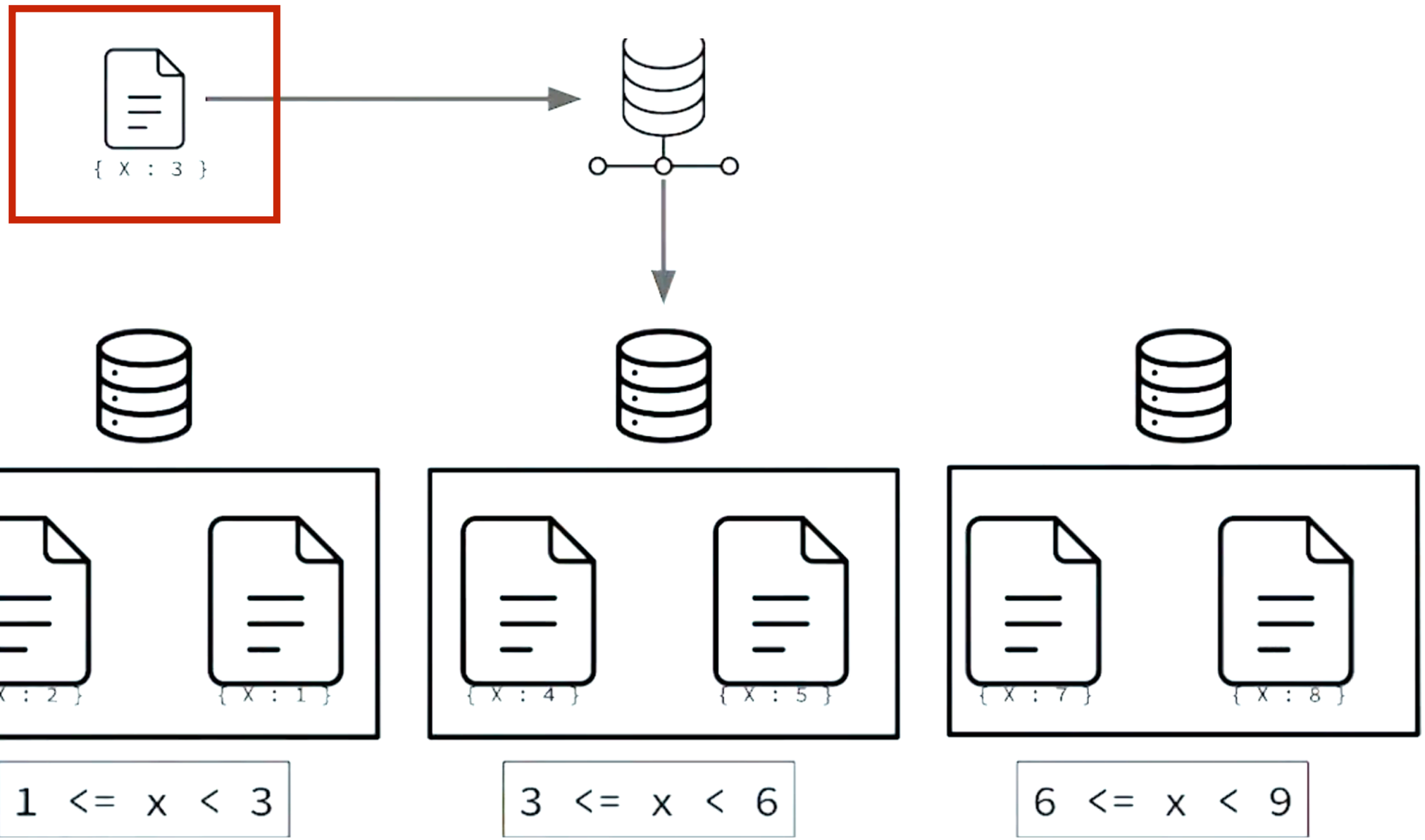
Use to determine how documents in a collection are distributed across the shards



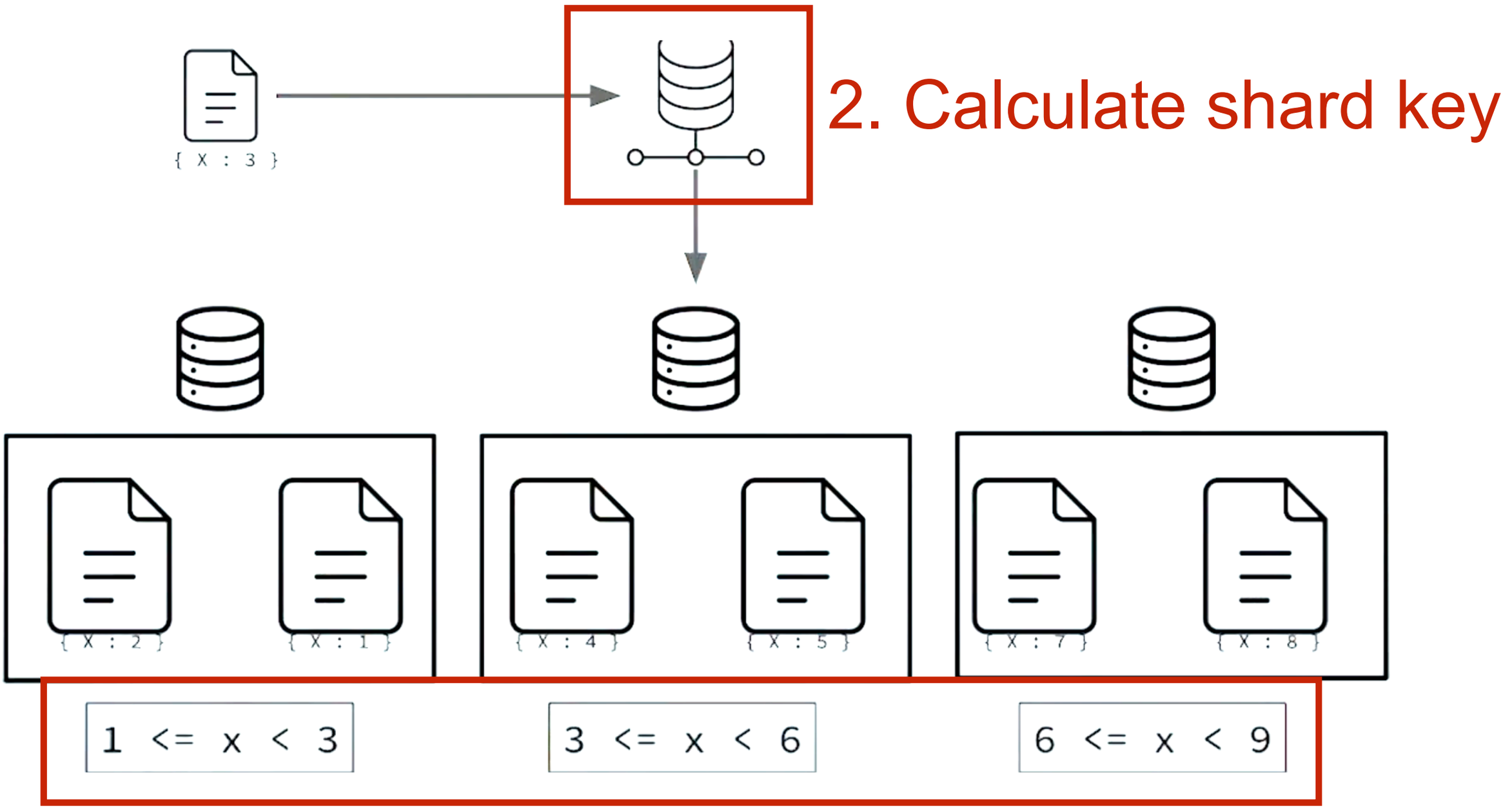
https://docs.mongodb.com/manual/core/sharding-shard-key/

Shard key

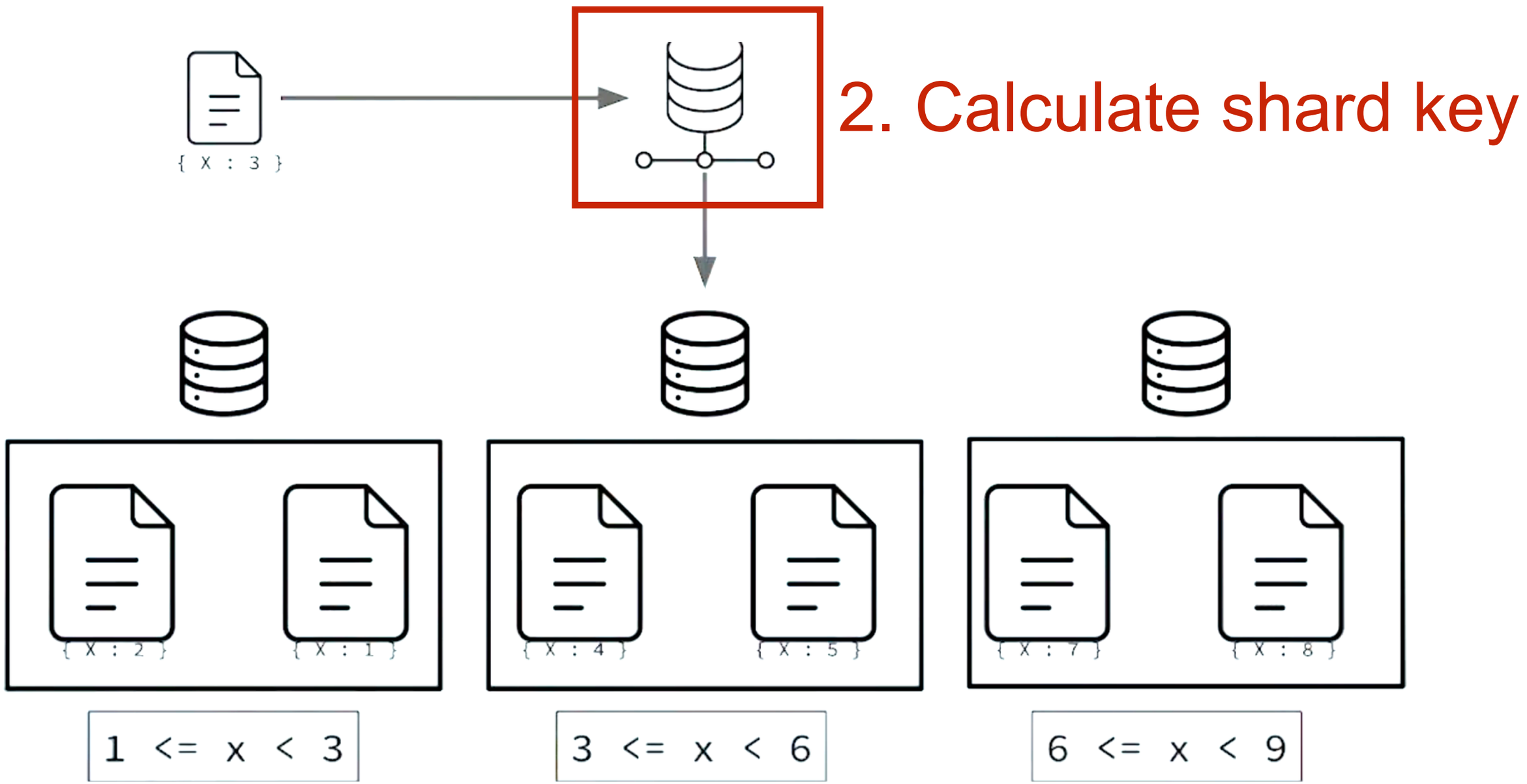
1. Create a document



Shard key



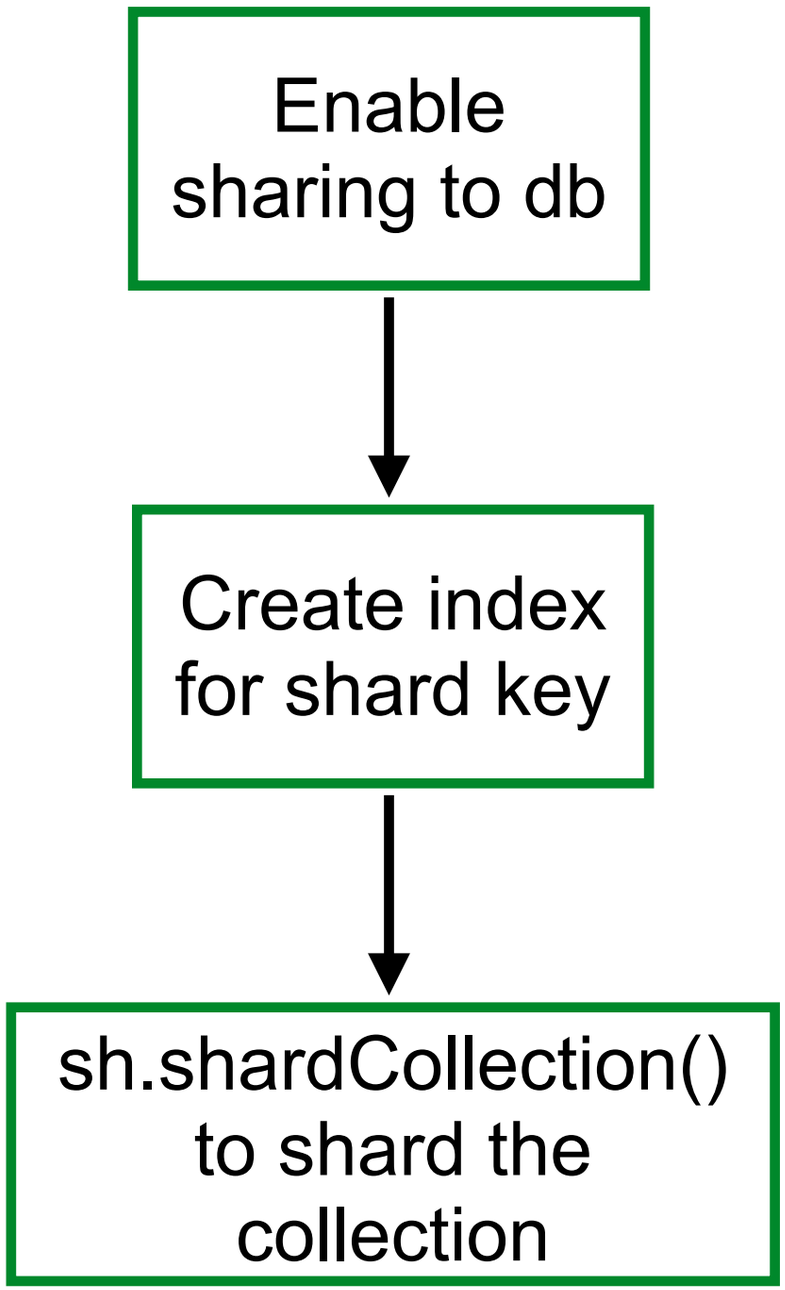
Shard key



Shard key

Must be indexed
Immutable (can’t be change)
Permanent

How to Shard ?


Enable sharing to db
Create index for shard key
sh.shardCollection() to shard the collection

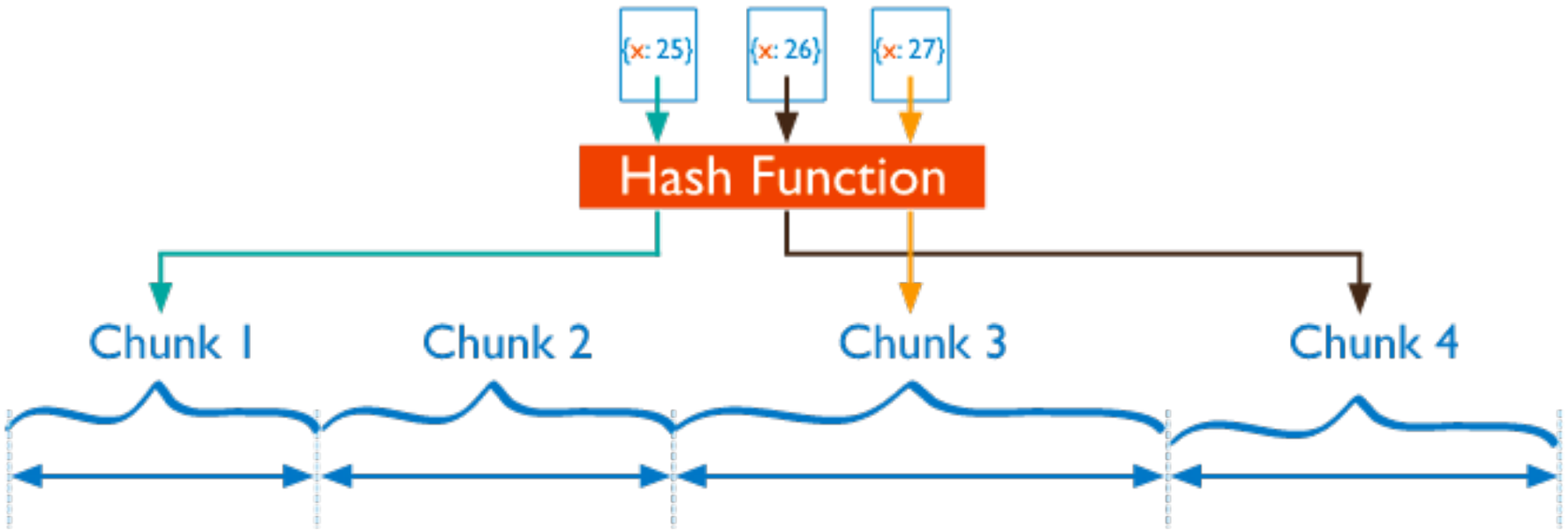
Choose shard key ?

The goal is a shard key whose values provides **good write distribution**

Cardinality
Frequency
Monotonic change

Hashed Shard key

Single field hashed index
Compound hashed index



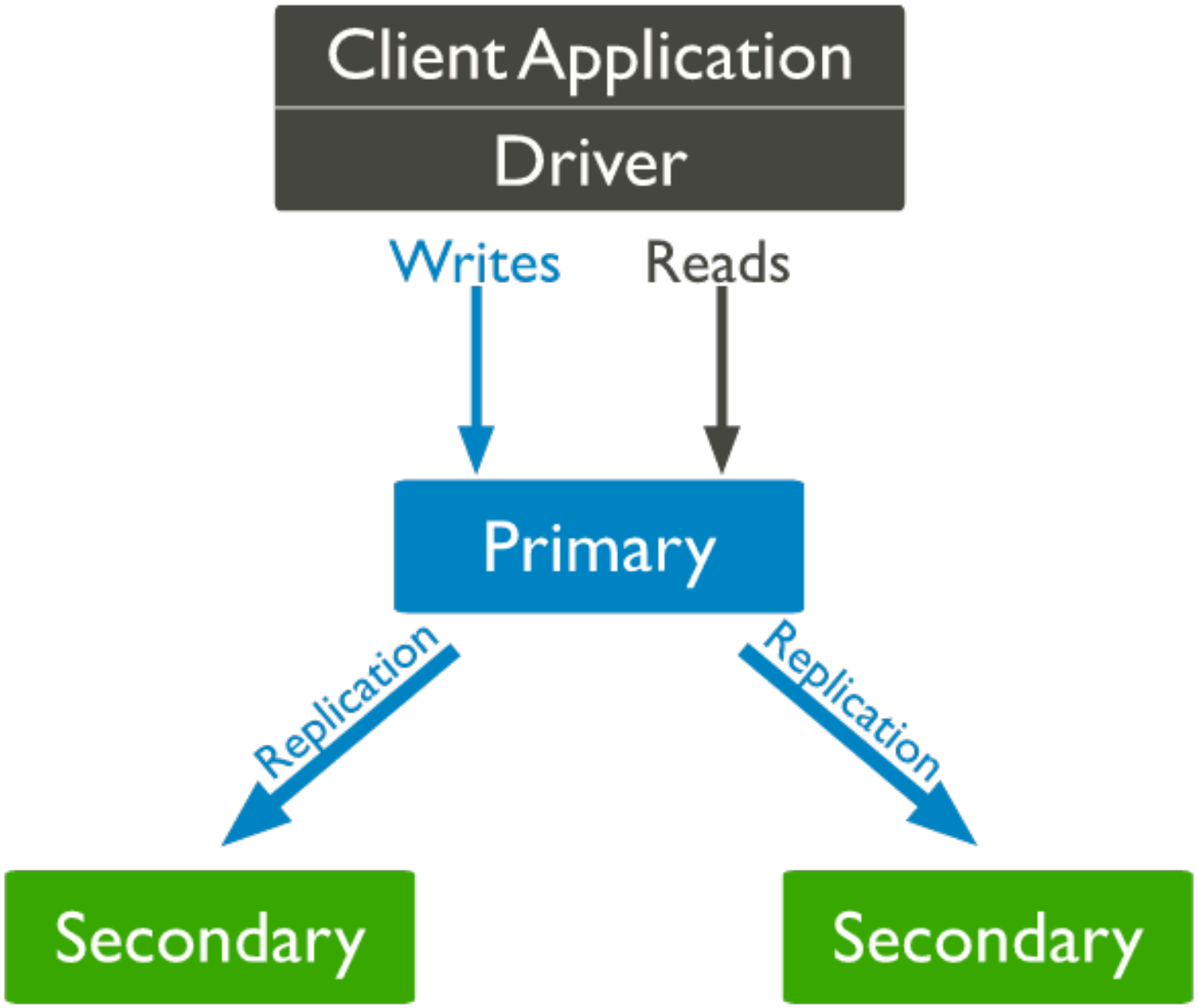
https://docs.mongodb.com/manual/core/hashed-sharding/

Workshop

MongoDB Deployment

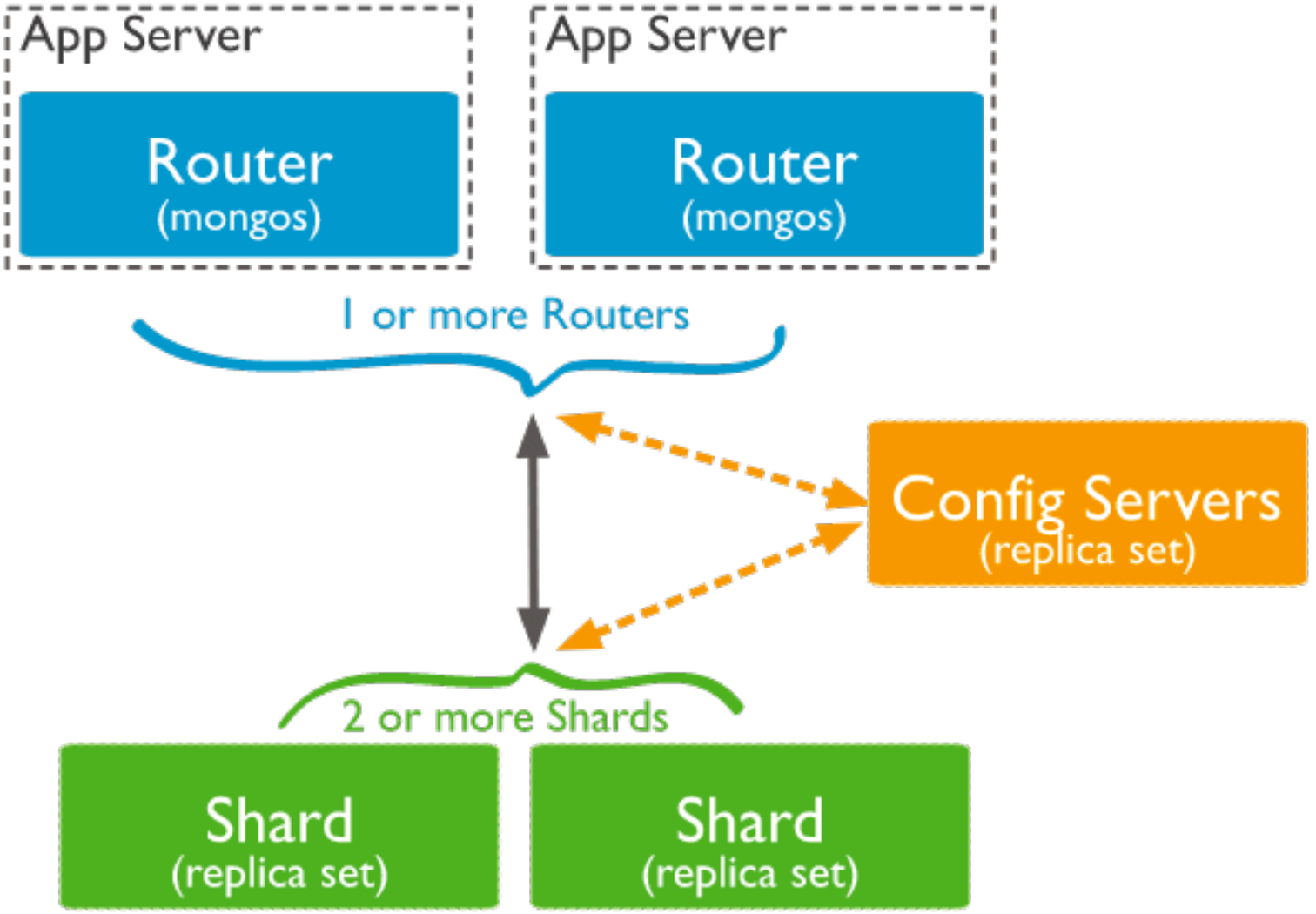
Standalone or Cluster
Replication
Shard cluster

Replication Deployment



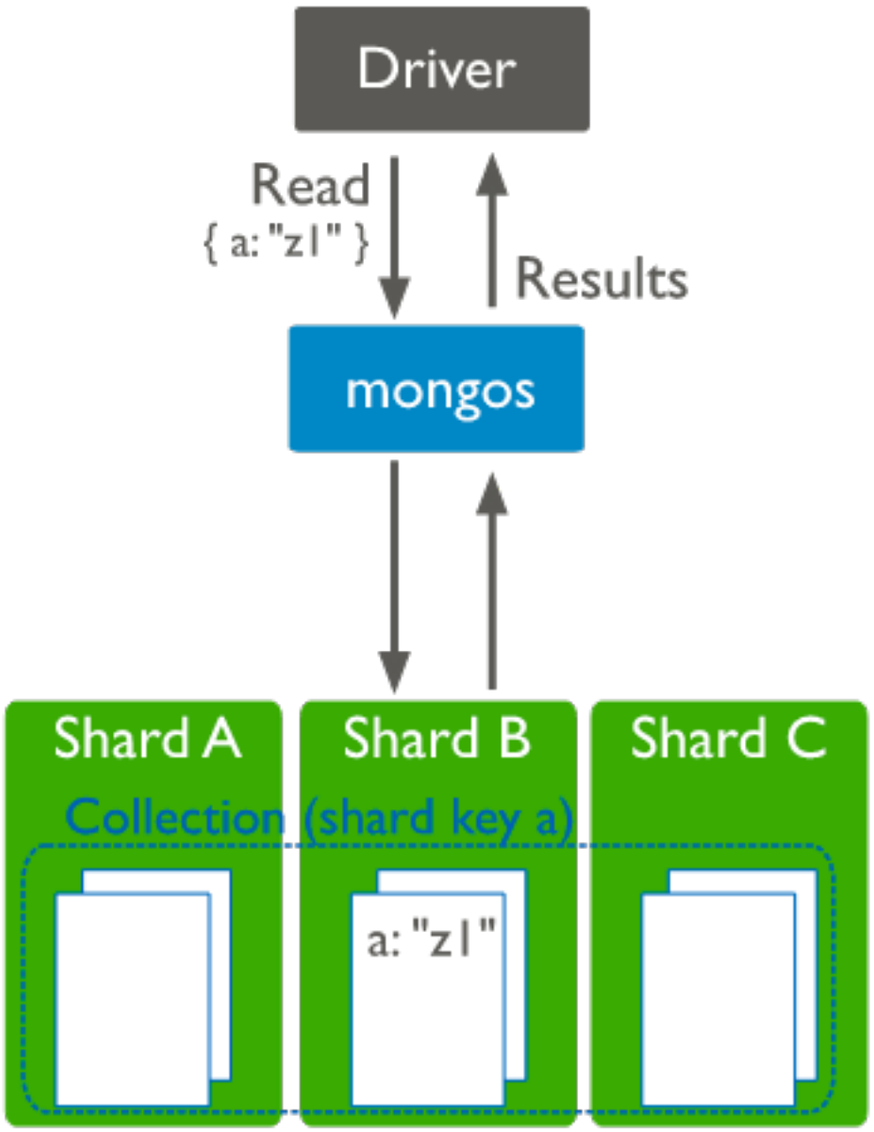
https://docs.mongodb.com/manual/replication/

Shard Cluster Deployment



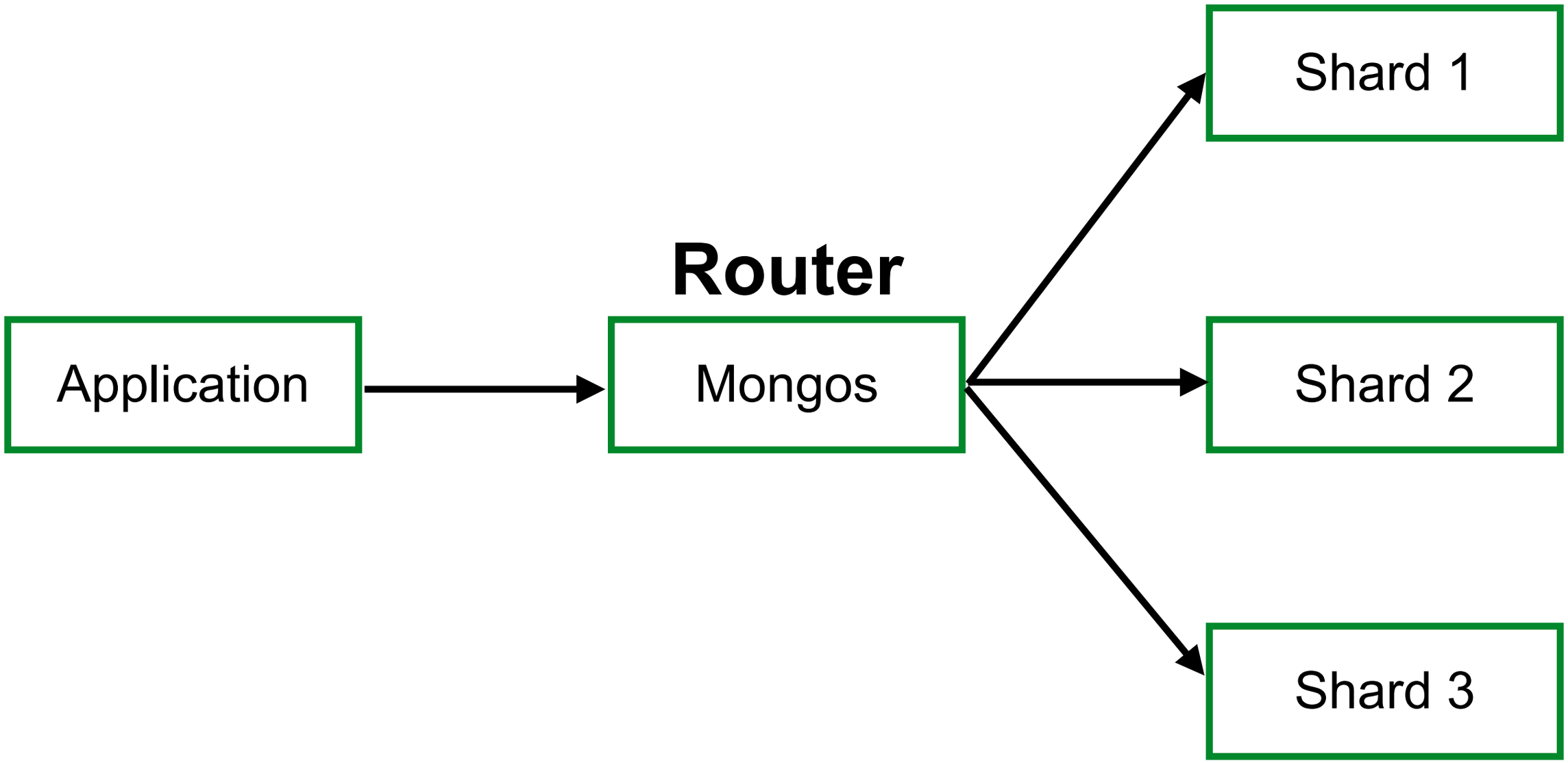
https://docs.mongodb.com/manual/core/sharded-cluster-components/

Shard Cluster Deployment

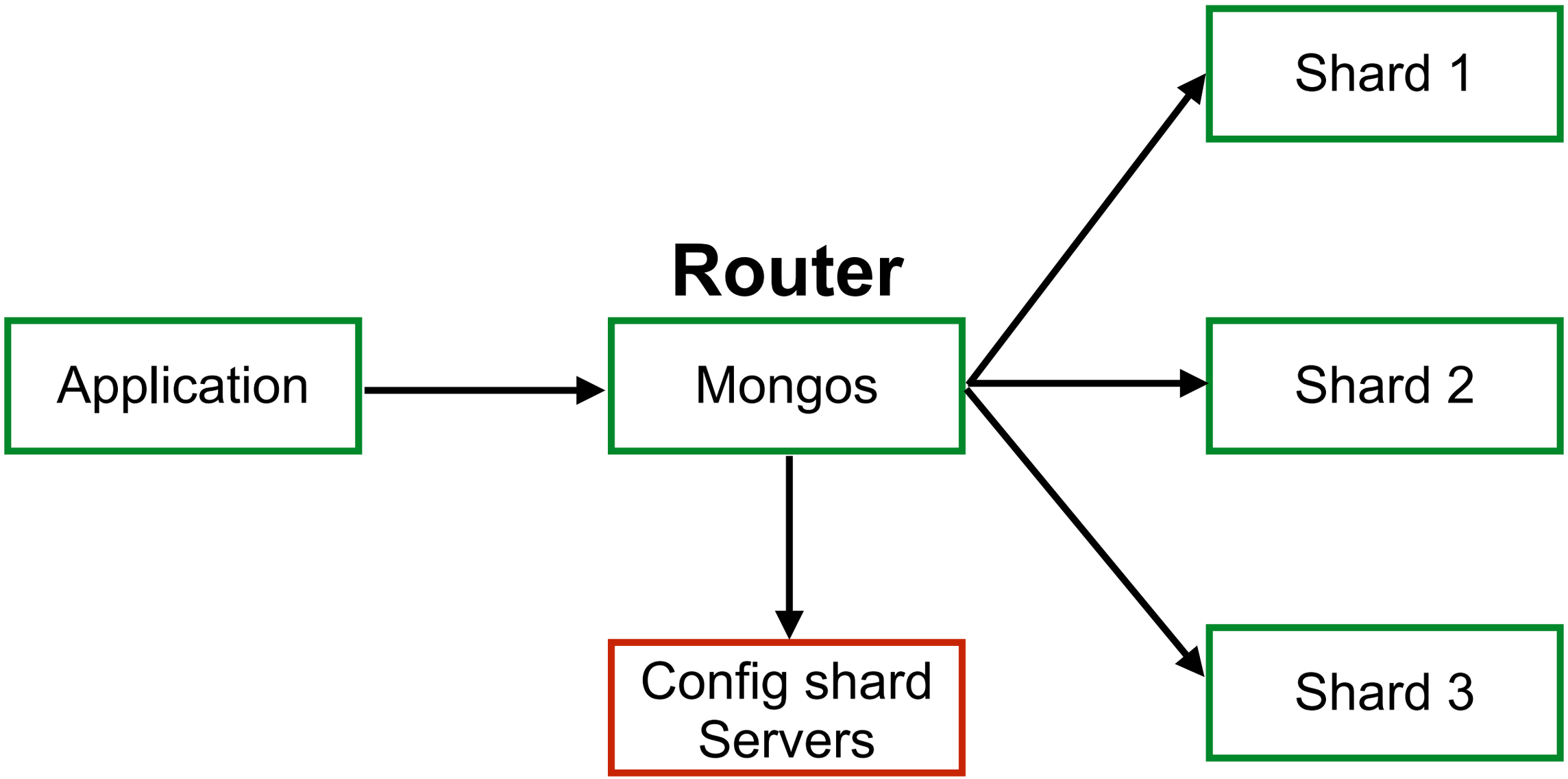


https://docs.mongodb.com/manual/core/sharded-cluster-components/

Shard Cluster Deployment



Shard Cluster Deployment



Query from shard cluster

sort()
limit()
skip()

https://docs.mongodb.com/manual/core/distributed-queries/

Query from shard cluster

sort()

The mongos pushes the sort to each shard and merge-sorts the result

limit()

The mongos passes the limit to each targeted shard, then re-applies the limit to the merge set of results

skip()

The mongos performs the skip against the merged set of results

Targeted vs Scatter gather query

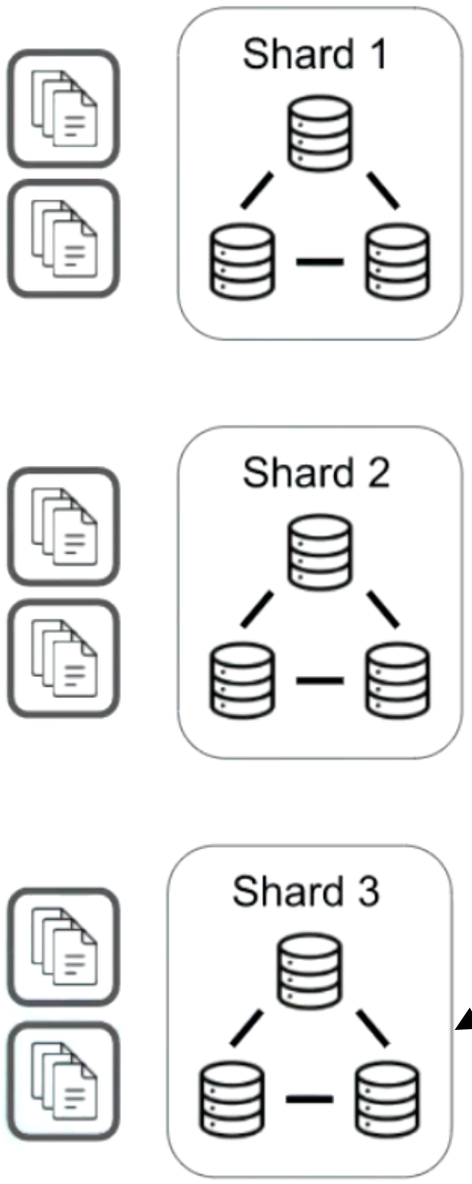
Targeted queries requires the shard key

Without shard key, the mongos must perform a **scatter-gather query**

Targeted query

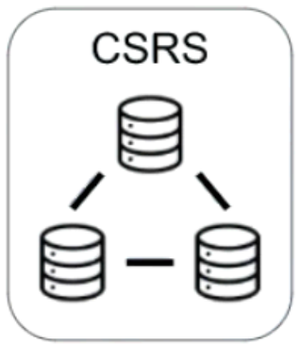
```
db.products.find( {"sku" : 20009151 } )
```

Shard key



mongos

Shard	Data
1	*minKey -> 10000000*
2	*10000000 -> 20000000*
3	*20000000 -> maxKey*

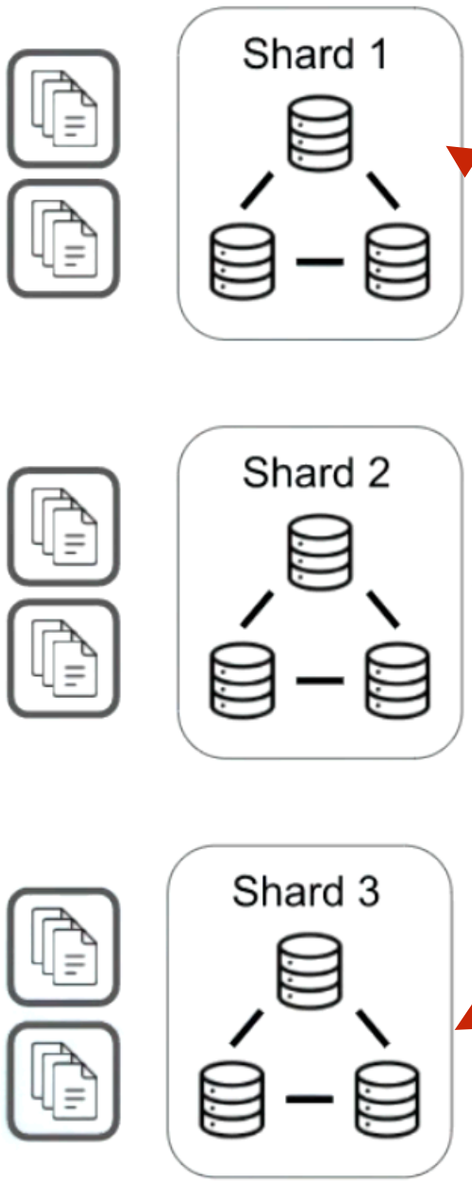


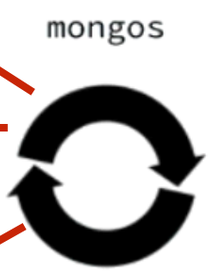
Shard	Data
1	*minKey -> 10000000*
2	*10000000 -> 20000000*
3	*20000000 -> maxKey*

Scatter gather query

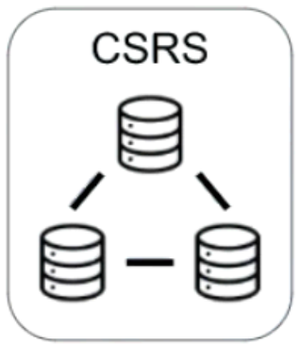
```
db.products.find( {"sku" : 20009151 } )
```

No Shard key





Shard	Data
1	*minKey -> 10000000*
2	*10000000 -> 20000000*
3	*20000000 -> maxKey*



Shard	Data
1	*minKey -> 10000000*
2	*10000000 -> 20000000*
3	*20000000 -> maxKey*

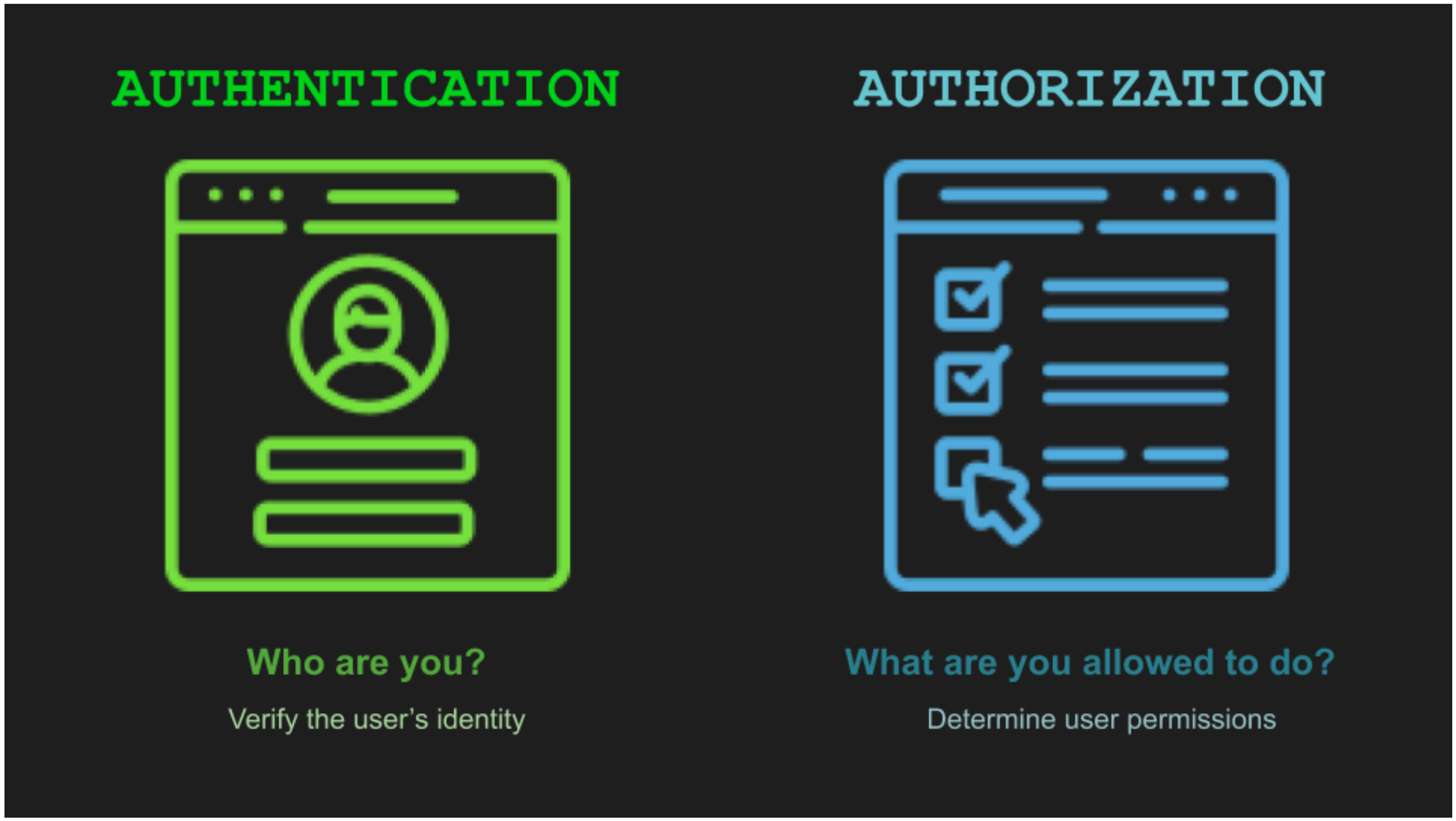
Workshop

Security

Security in MongoDB

User authentication
Role-Based Access Control (RBAC)
Internal communication authentication

Authentication vs Authorization



Working with MongoDB

Create user document

```
db.createUser(
      {
        user: "accountAnalytics",
        pwd: "LionelMessi1987",
        roles: [
                  { role: "read", db: "accounts" },
                  { role: "readWrite", db: "analytics" }
                ]
      }
)
```

Working with MongoDB

Create roles of user

```
db.createUser(
      {
        user: "accountAnalytics",
        pwd: "LionelMessi1987",
        roles: [
                { role: "read", db: "accounts" },
                { role: "readWrite", db: "analytics" }
              ]
      }
)
```

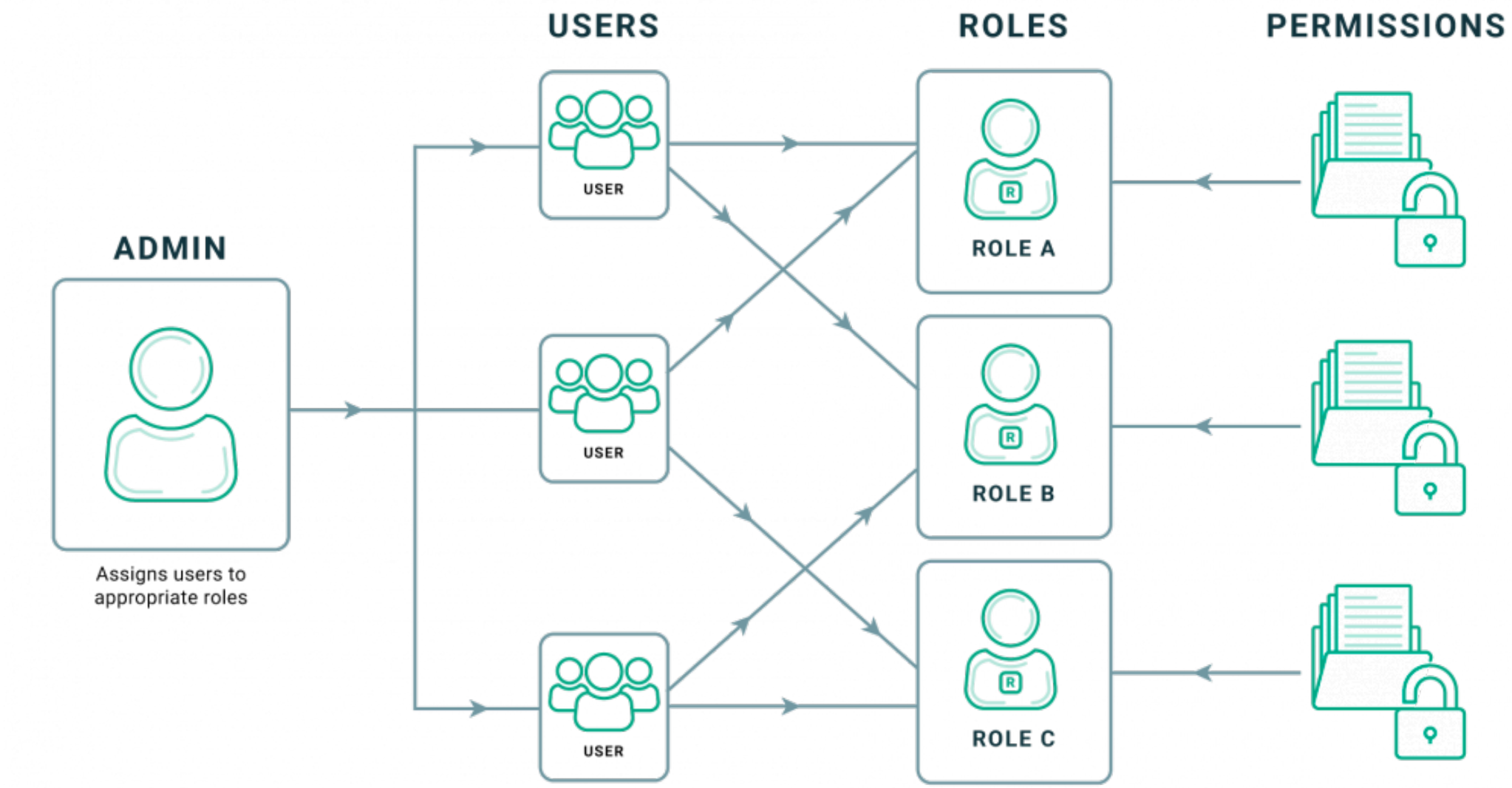
Quality Assurance feedback

Database is secure ?

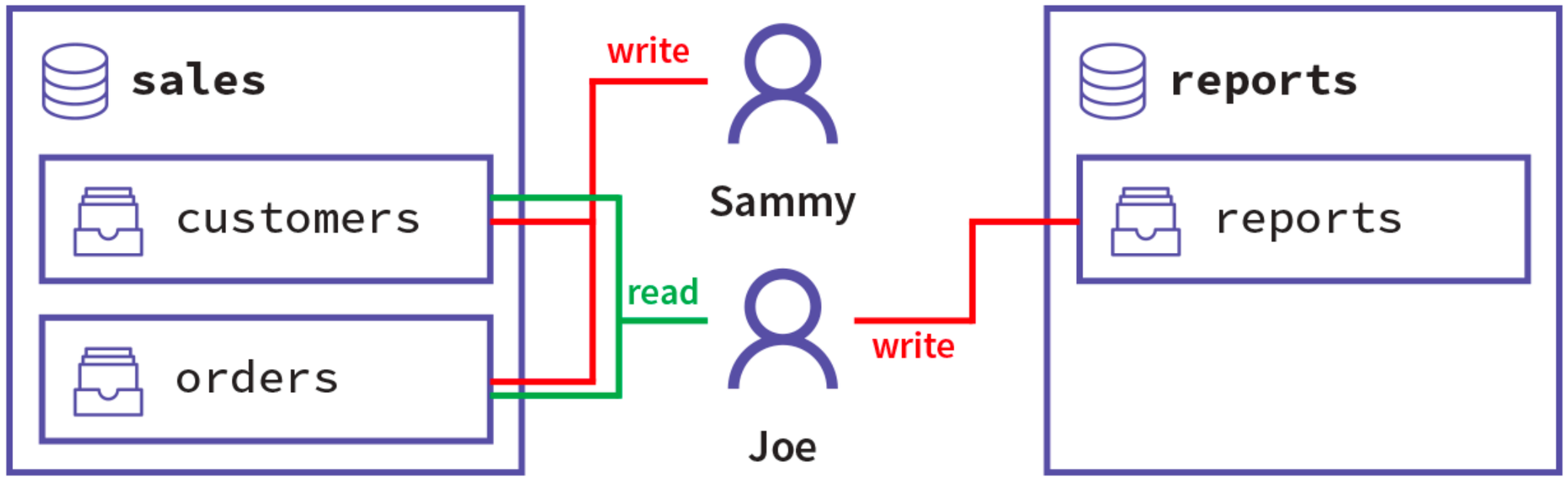
Database allows users to perform commands without authenticating first !!

Workshop

Role-Based Access Control



Role-Based Access Control



Role-Based Access Control

Build-in roles
User-defined roles

https://docs.mongodb.com/manual/core/authorization/

Build-in roles

MongoDB provides build-in roles to offer a set of privileges that are commonly needed in a database system

https://docs.mongodb.com/manual/reference/built-in-roles/

Build-in roles

Build-in roles

Group name	Roles
Database user	read readWrite
Database administration	dbAdmin userAdmin dbOwner
Cluster admin	clusterManager, clusterMonitor hostManager, clusterAdmin
Backup and restore	backup restore
All databases	readAnyDatabase, readWriteAnyDatabase userAdminAnyDatabase, dbAdminAnyDatabase
Superuser	dbOwner,userAdmin userAdminAnyDatabase

Workshop

User-defined roles

If build-in roles do not provide all the expected privileges, database administrators can be define custom roles using the **createRole()**

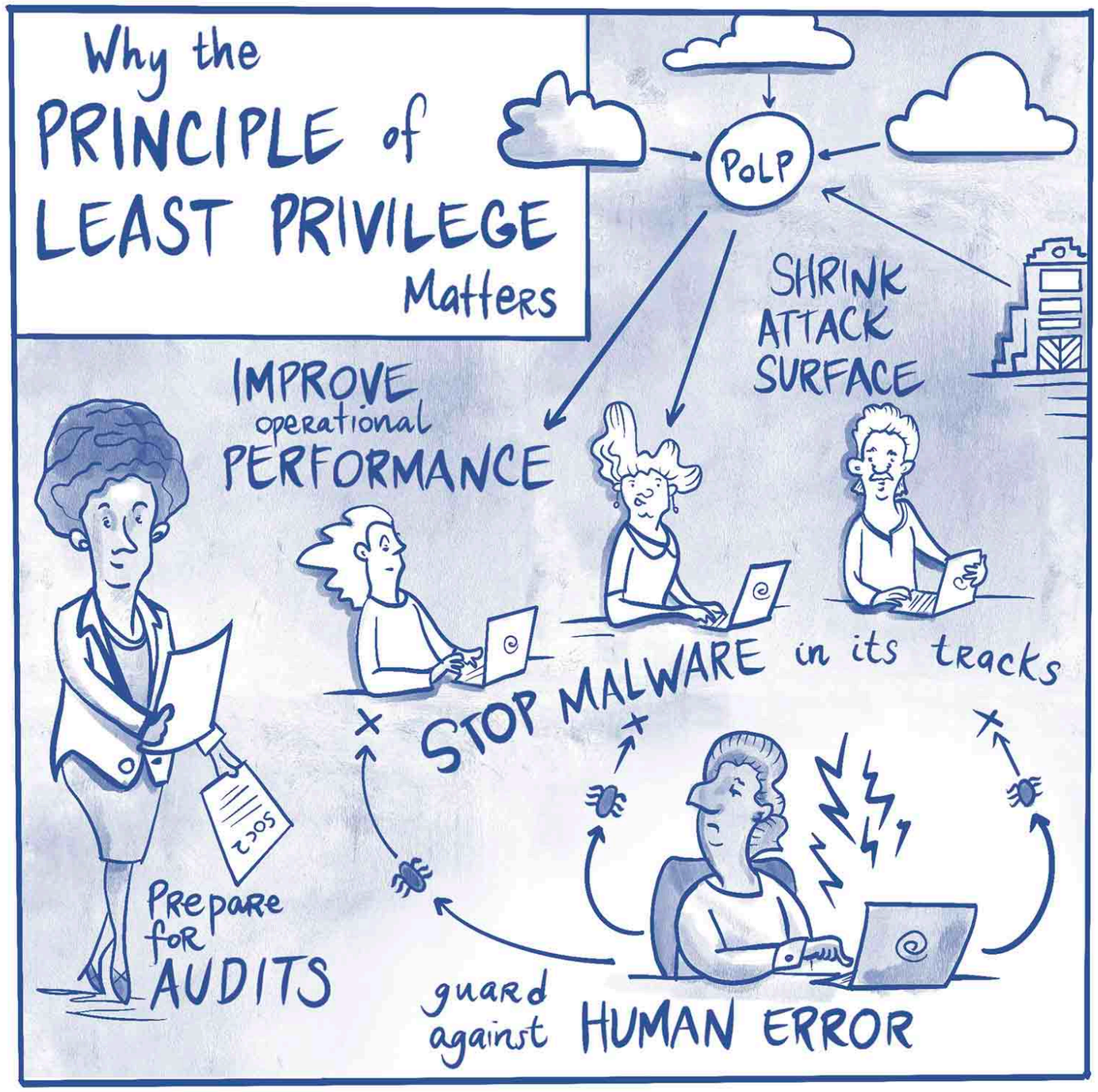
https://docs.mongodb.com/manual/core/security-user-defined-roles/

Why create user-defined roles ?

Fine-grained control to user administrators
Build-in roles grant too many privileges

User admin can adhere
to the **Principle of Least Privilege (PoLP)**

Principle of Least Privilege



Principle of Least Privilege

Users should have the least privilege required for their intended purpose

https://en.wikipedia.org/wiki/Principle_of_least_privilege

User-defined roles

Method name	Description
db.createRole()	Create a role and its privileges
db.updateRole()	Update the user-defined role
db.dropRole()	Delete a user-defined role
db.grantPriviledgesToRole()	Assigns new privileges to a role
db.revokePriviledgesFromRole()	Removes privileges from a roles

Build-in role : userAdmin

List of privilege actions

changeCustomData	grantRole
changePassword	revokeRole
createRole	setAuthenticationRestriction
createUser	viewRole
dropRole	viewUser
dropUser	

Build-in role : userAdmin

Need only 3 privilege actions for this role

changeCustomData	**grantRole**
changePassword	**revokeRole**
createRole	setAuthenticationRestriction
createUser	**viewRole**
dropRole	viewUser
dropUser	

Create new role

For all databases and collections

```
db.createRole(
   {
     role: "grantRevokeRolesAnyDatabase",
     privileges: [
       {
         resource: { db: "", collection: "" },
         actions: [ "grantRole", "revokeRole", "viewRole" ]
       }
     ],
     roles: []
   }
```

Create new role

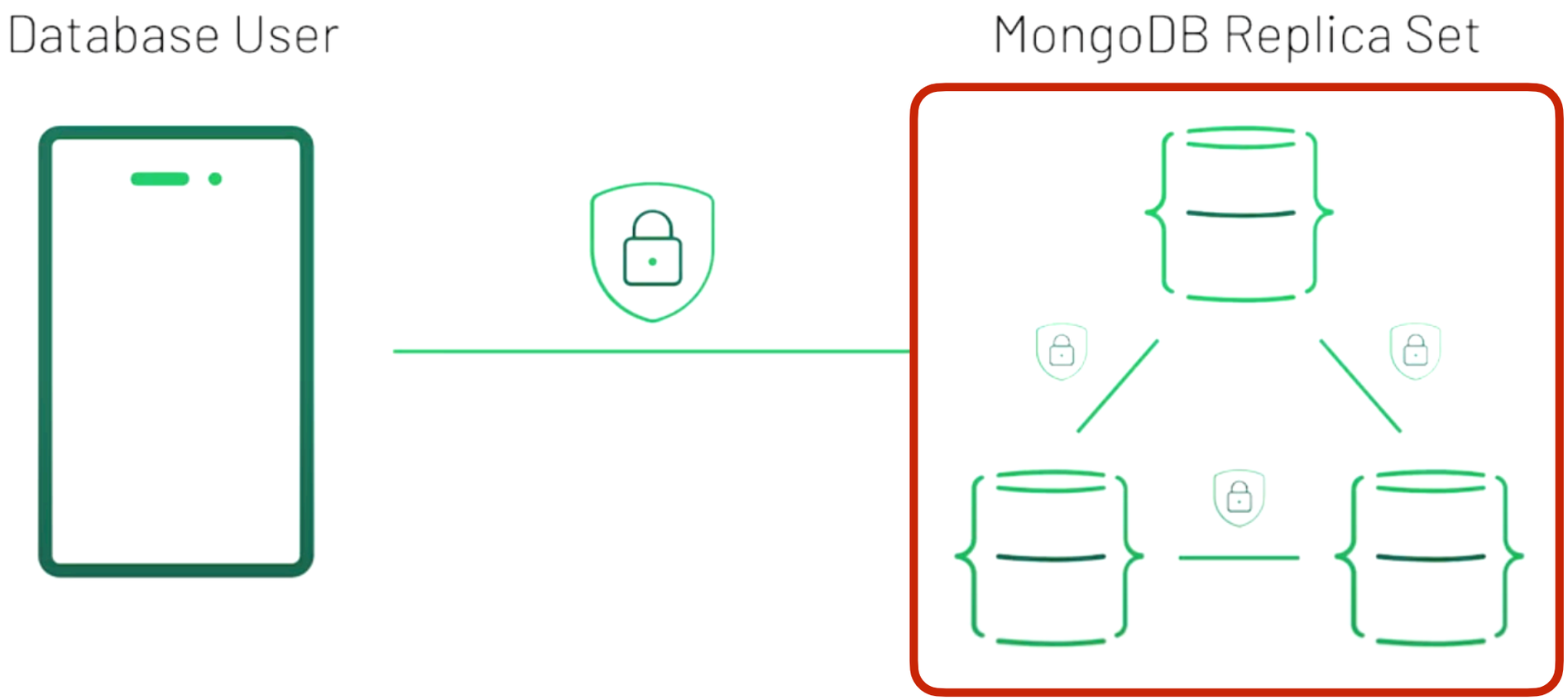
Add privilege actions to this role

```
db.createRole(
   {
     role: "grantRevokeRolesAnyDatabase",
     privileges: [
       {
         resource: { db: "", collection: "" },
         actions: [ "grantRole", "revokeRole", "viewRole" ]
       }
     ],
     roles: []
   }
)
```

Workshop

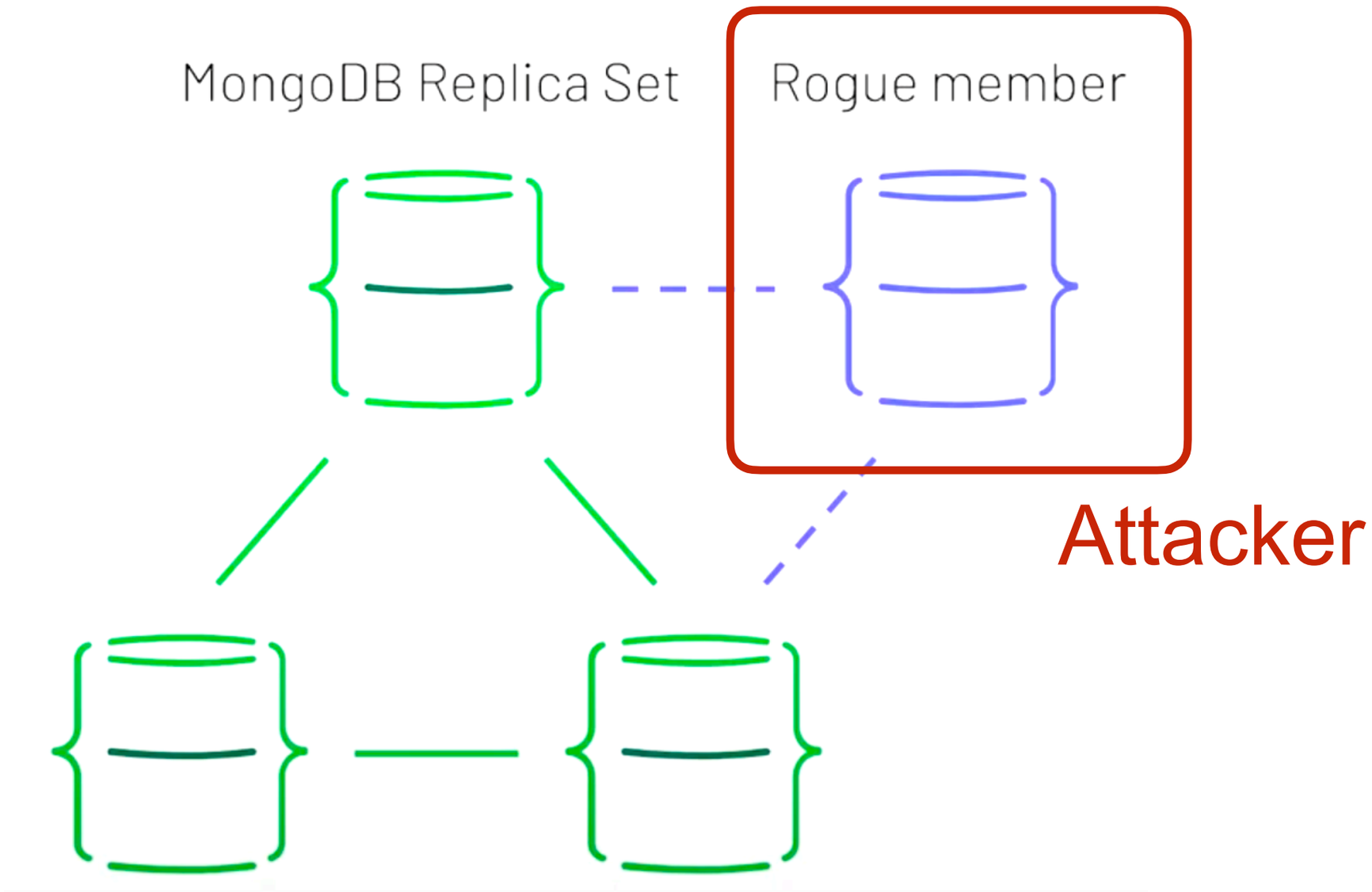
Internal Authentication ?

To verify the identity of one MongoDB instance to another



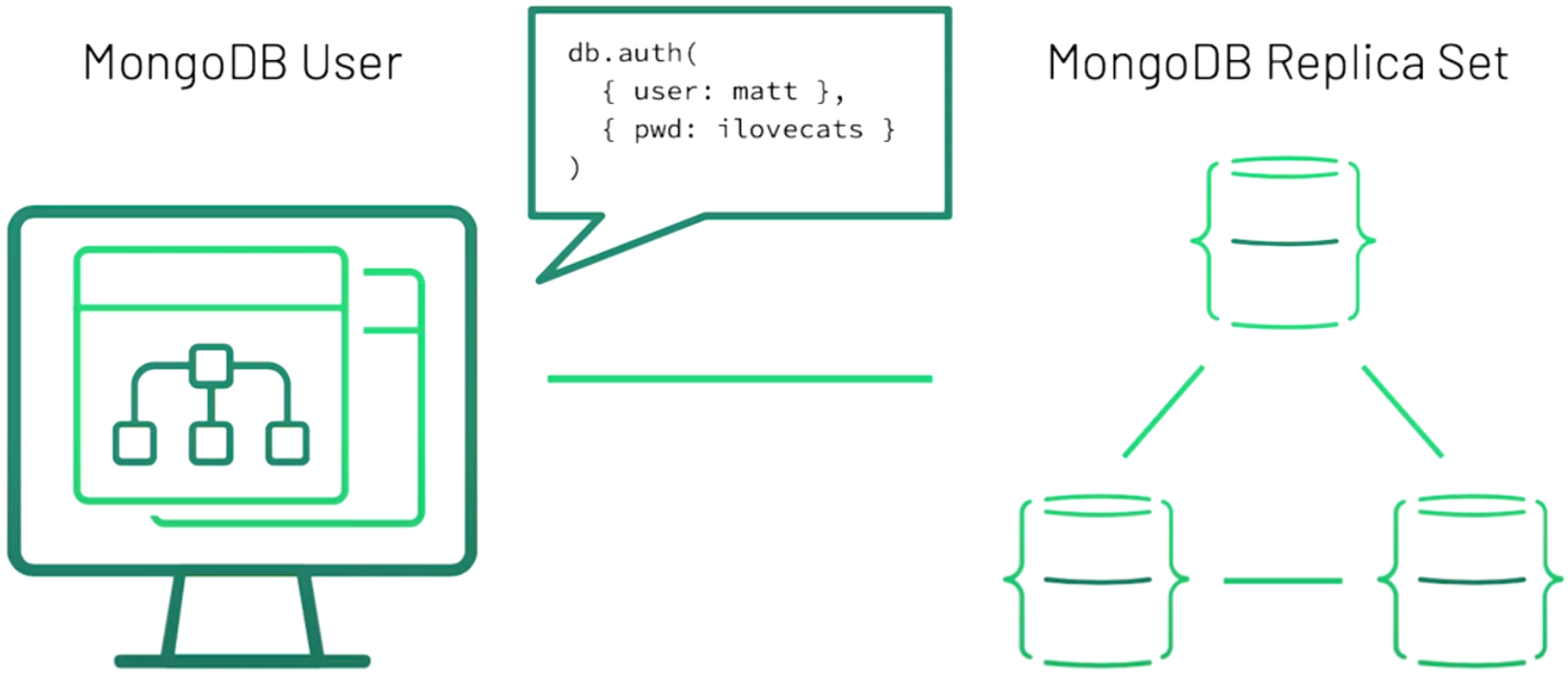
https://docs.mongodb.com/manual/core/security-internal-authentication/

Replica sets without internal communication



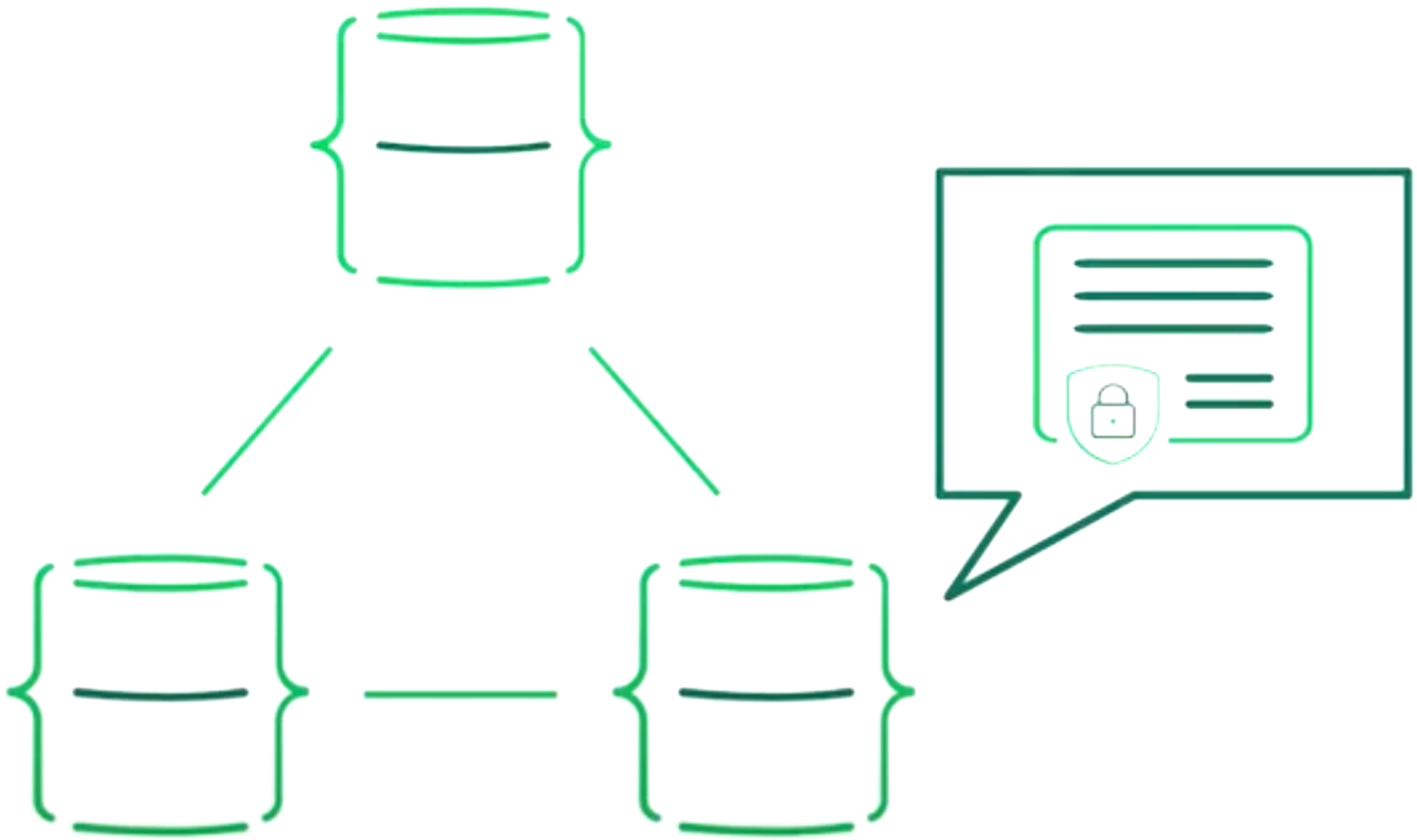
SCRAM

Use keyfile
Default authentication for MongoDB



https://docs.mongodb.com/manual/core/security-scram/

x.509 certificates



https://docs.mongodb.com/manual/core/security-x.509/

x.509 certificates

Verify members of replica set are authenticating to each other

Encrypt sent over the network

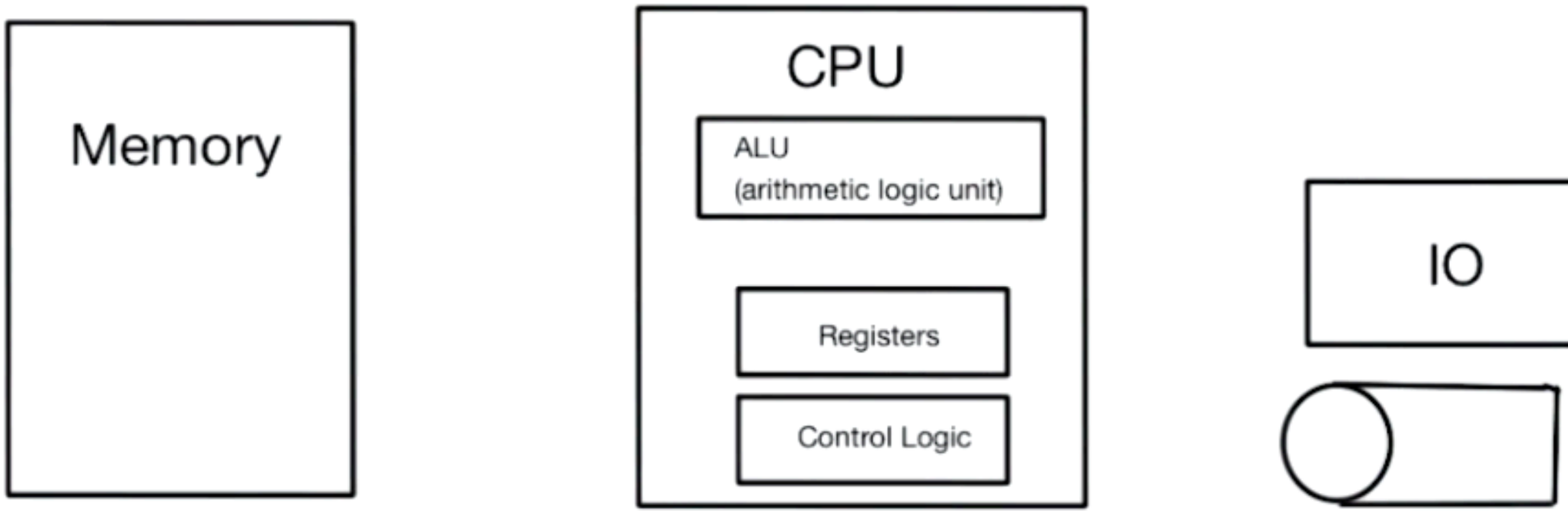
Workshop

Performance

Performance

Hardware
Configuration
Indexes
CRUD optimization
Performance on cluster

Hardware


Memory
CPU
ALU
(arithmetic logic unit)
Registers
Control Logic
IO

Memory with MongoDB

Aggregation
Index traversing
Write operations
Query engine
Connections

CPU with MongoDB

Storage engine
Concurrency model

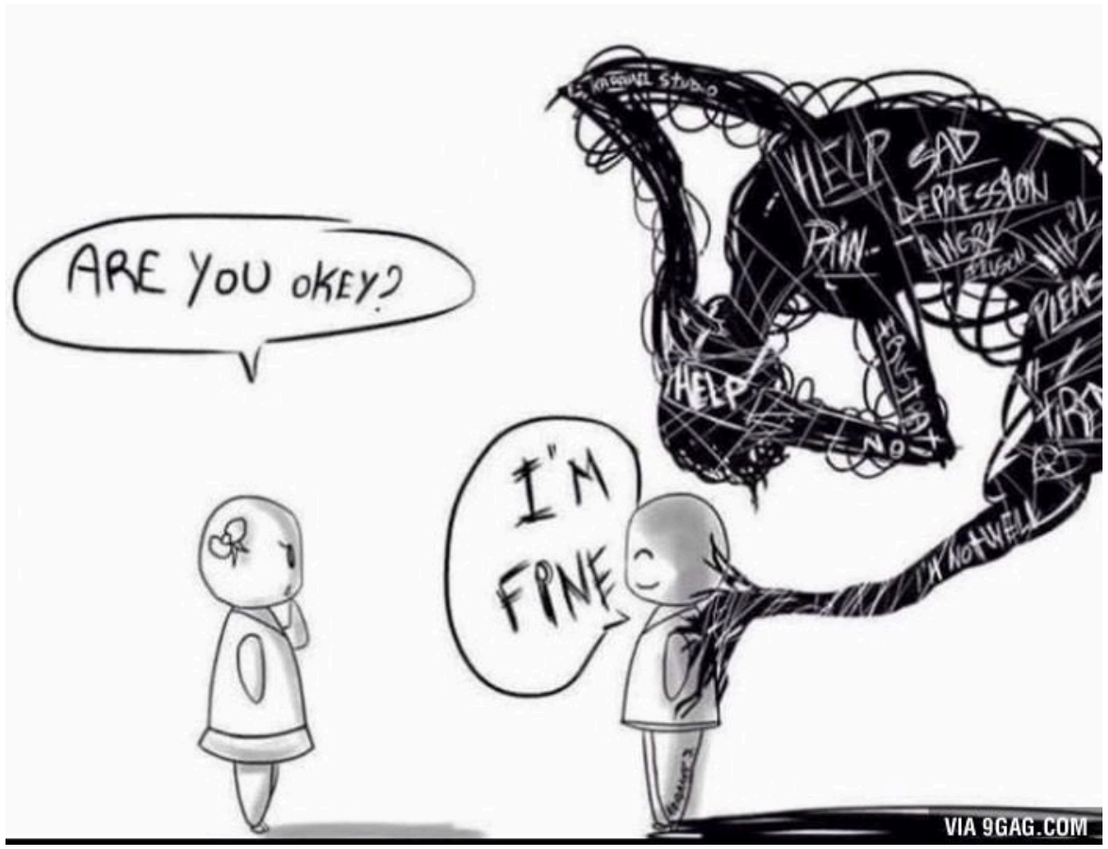
Indexed

https://docs.mongodb.com/manual/indexes/

Indexes

Make query more efficient

Most impactful ways to improve query performance



Index types

Single
Compound
Multikey
Geospatial
Text
Hashed

Better indexes ?

Build the right indexes for your criteria
Learn how to MongoDB picks a index
Improve query performance with **explain**
Learn about other **index types**

Avoid sorting !!

https://docs.mongodb.com/manual/tutorial/sort-results-with-indexes

Working with Distributed system

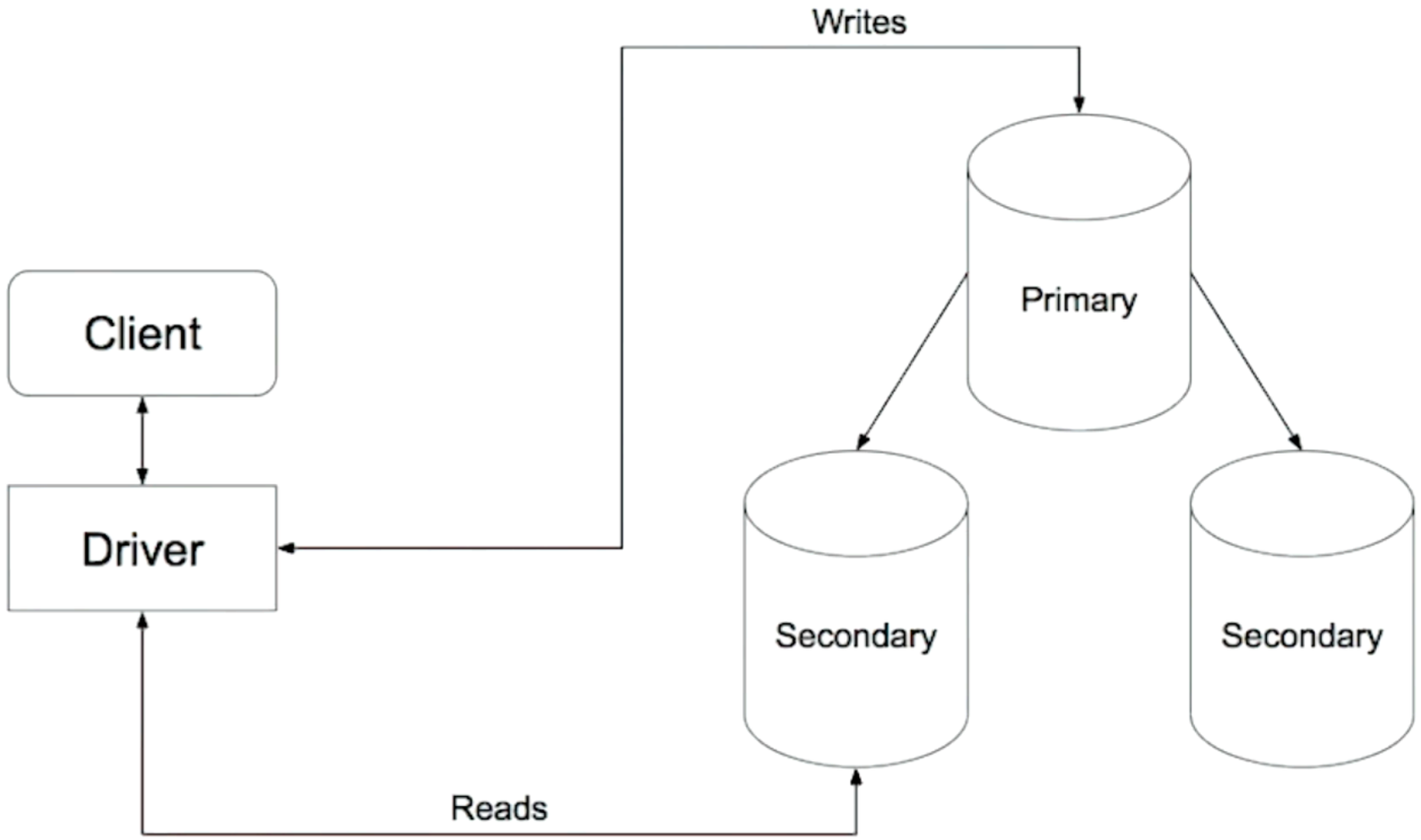
Consider latency
Data is spread across different nodes
Read and Write operations

Increase write performance

Vertical vs Horizontal scale
Shard key rules
Bulk write

Read from secondary

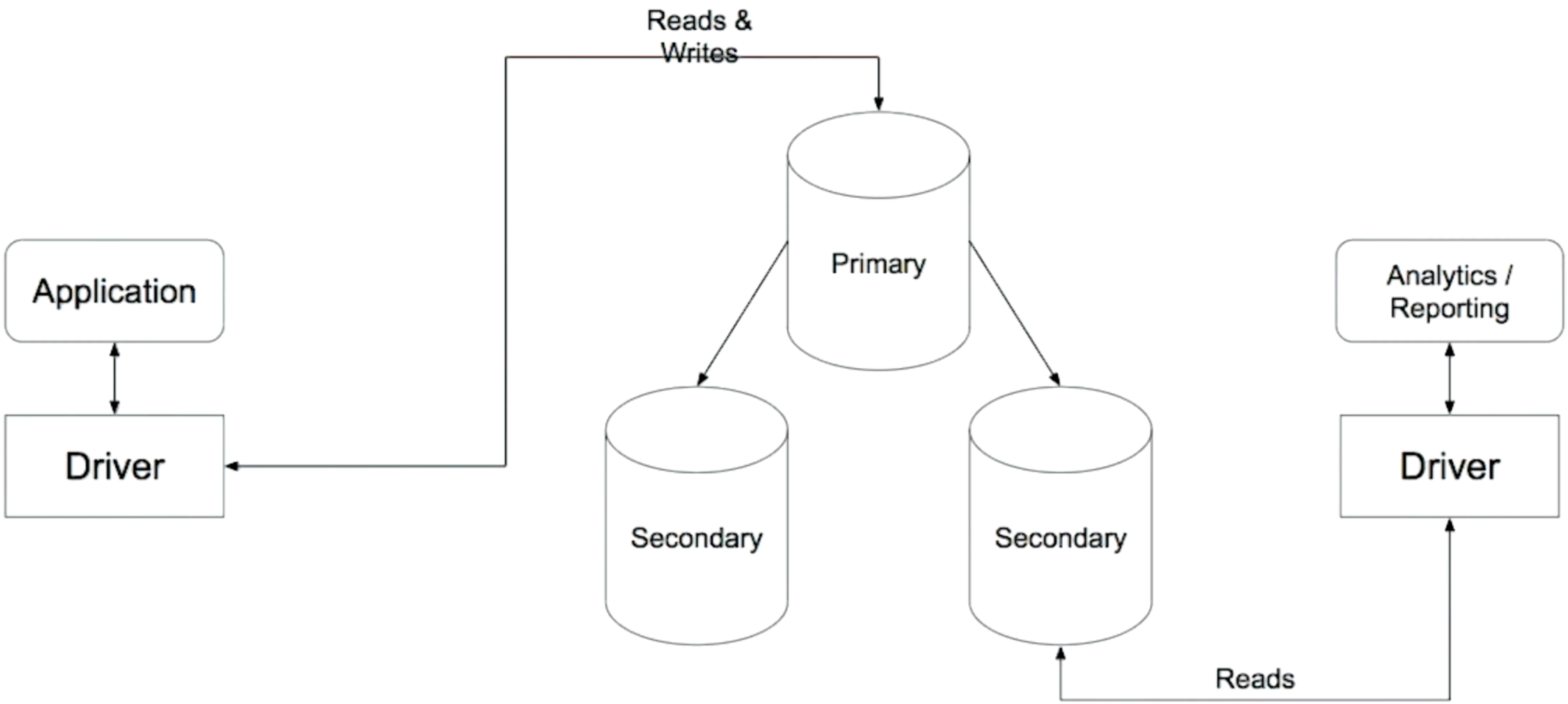
Eventually consistency




```
db.people.find().readPref("secondary")
```

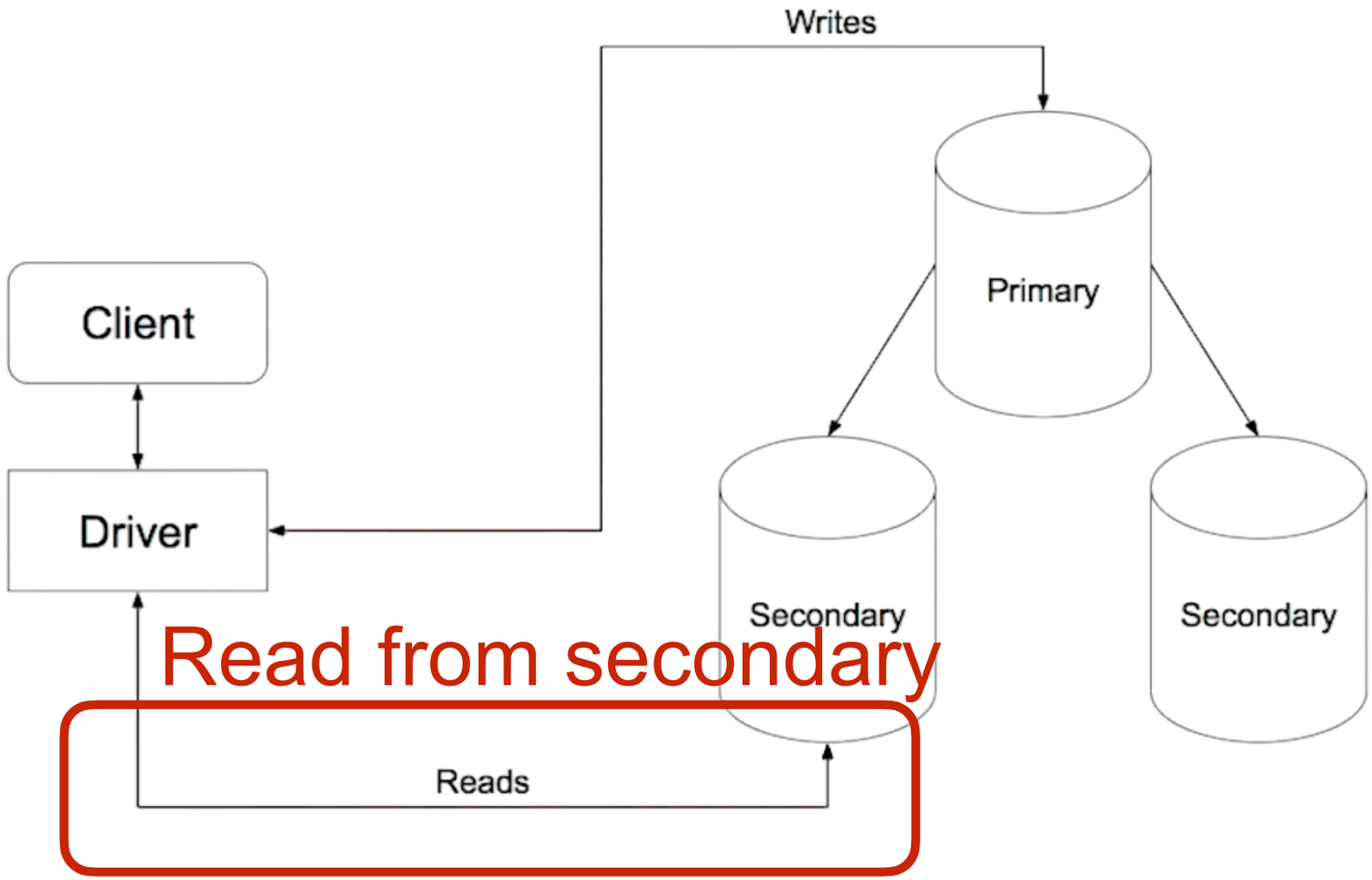
Use case

```
db.people.find()
```




```
db.people.find().readPref("secondary")
```

Be careful !!




```
db.people.find().readPref("secondary")
```

Workshop

Q/A