Chapter 6.
Basic Cluster Administration

References

- MongoDB The Definitive Guide: Powerful and Scalable Data Storage 3rd Edition
- https://docs.mongodb.com/
- https://www.mongodb.com/docs/manual/

Contents

1. mongod

Learning Objectives

- Understand mongod
- Communicate with mongod
- Configure for mongod

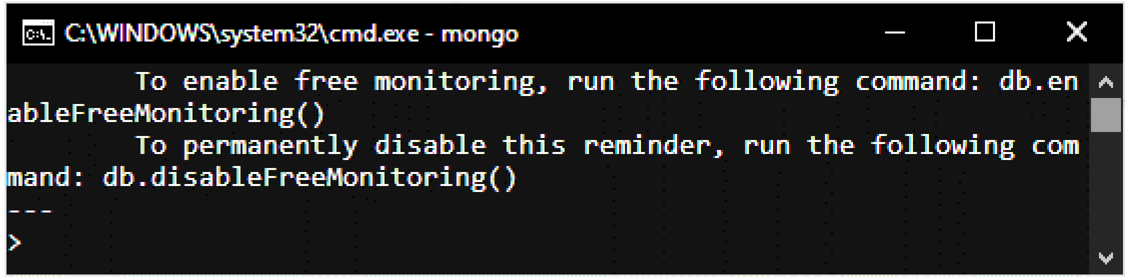
What is mongod?

- mongod is the main daemon process for MongoDB.
- The core server of the database, handling connections, requests, and most importantly, persisting your data.
- MongoDB deployment may consist of more than one server. Our data may be distributed in a replica set or across a sharded cluster.
- We run a separate mongod process for each server.

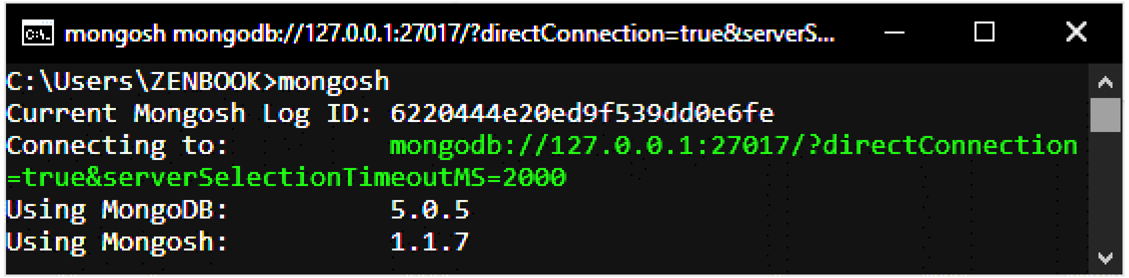
mongod
mongod
mongod

What is mongod?

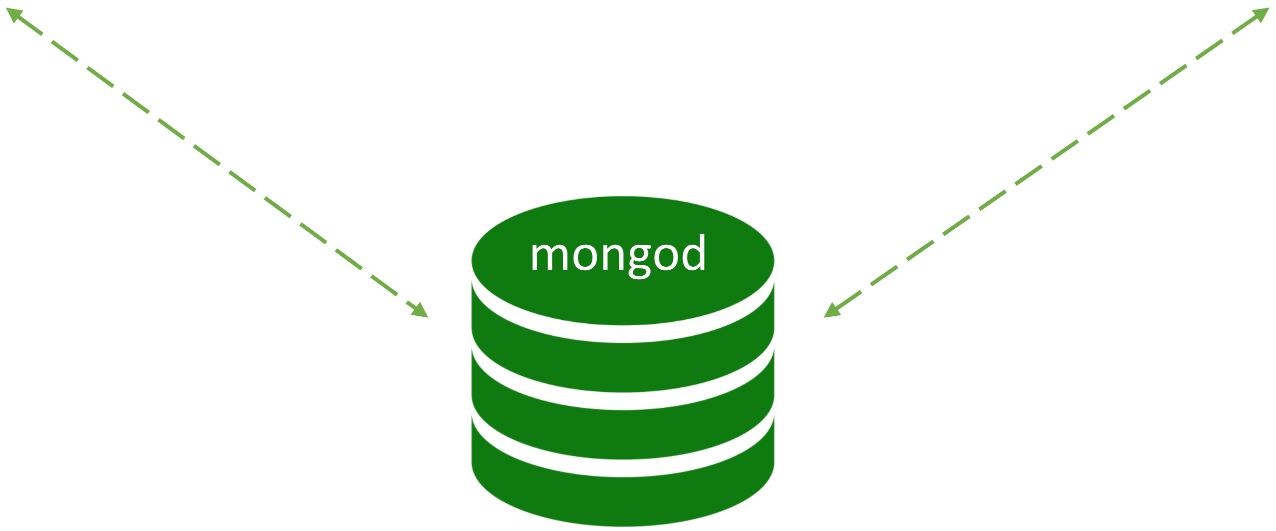
- When we launch mongod, we're essentially starting up a new database. But we don't interact with the mongod process directly. Instead, we use a database client to communicate with mongod (*mongosh*, *mongo*).



```
        To enable free monitoring, run the following command: db.en
ableFreeMonitoring()
        To permanently disable this reminder, run the following com
mand: db.disableFreeMonitoring()
---
>
```


mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverS...

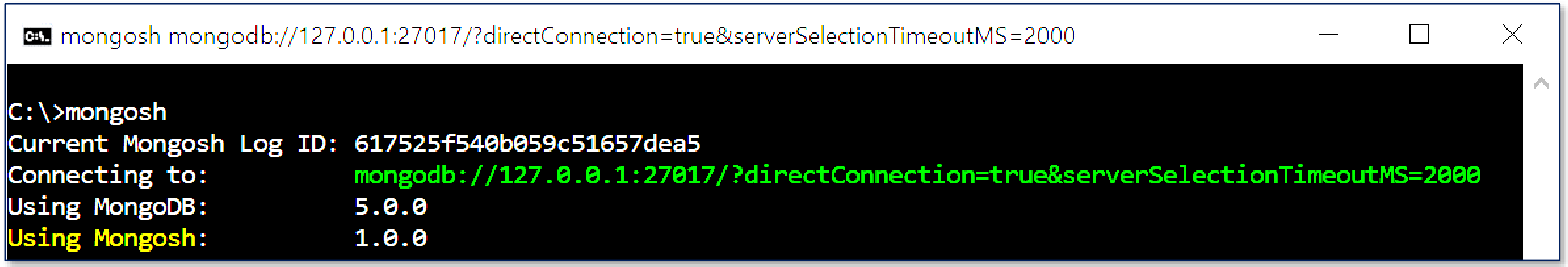
```
C:\Users\ZENBOOK>mongosh
Current Mongosh Log ID: 6220444e20ed9f539dd0e6fe
Connecting to:          mongodb://127.0.0.1:27017/?directConnection
=true&serverSelectionTimeoutMS=2000
Using MongoDB:          5.0.5
Using Mongosh:          1.1.7
```



What is mongod?

Default configuration:

- Bind to localhost by default *(127.0.0.1)*.
- The port mongod listens on will default to 27017.
- The default dbpath is /data/db *(this folder should be available when we run mongod)*.
- Authentication is turned off by default, so clients are not required to authenticate before accessing the database.



mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

```
C:\>mongosh
Current Mongosh Log ID: 617525f540b059c51657dea5
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Using MongoDB:          5.0.0
Using Mongosh:          1.0.0
```

Mongod options

Example:

```
mongod --dbpath /data/db --logpath /data/log/mongod.log --replSet 'M103' --keyFile /data/keyfile --bind_ip '127.0.0.1'
```

--help: output the various options for mongod with a description of their functionality.

- mongod --help or mongod -h

--dbpath *<directory path>*: specify where all data files of the database are stored.

--port *<port number>*: specify the port on which mongod will listen for client connections.

- Run mongo shell connect to above mogod: mongosh --port *27018*

--bind_ip: specify which IP addresses mongod should bind to. When mongod binds to an IP address, clients from that address are able to connect to mongod.

- mongod --bind_ip *localhost* --port *27018* --dbpath *'c:\mongoDB\data\db'*
- mongod --bind_ip *localhost, 123.123.123.123* --port *27018* --dbpath *'c:\mongoDB\data\db'*
- If using the bind_ip option with external IP addresses, it's recommended to enable auth to ensure that remote clients connecting to mongod have the proper credential

--auth: enables authentication to control which users can access the database. When auth is specified, all database clients who want to connect to mongod first need to authenticate.

Mongod – Configuration File

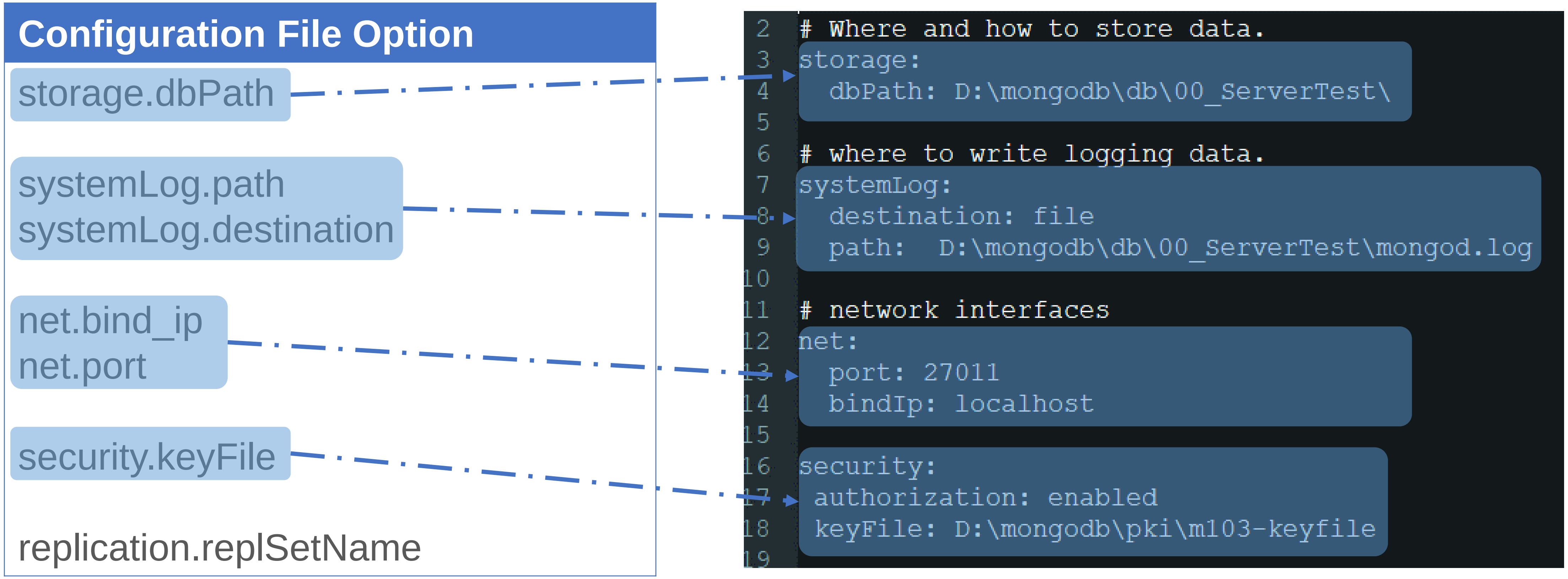
- Configuration file is a way to organize the options you need to run the mongod process into an easy to parse YAML (*Yet Another Markup Language*) file
- Why do we need to use configuration file?

```
mongod --dbpath /data/db --logpath /data/log/mongod.log --replSet 'M103' --keyFile /data/keyfile --bind_ip '127.0.0.1'
```

Command Line Options	Configuration File Option
--dbpath	storage.dbPath
--logpath	systemLog.Path and systemLog.destination
--bind_ip	net.bind_ip
--port	net.port
--replSet	replication.replSetName
--keyFile	security.keyFile
…	…

Mongod – Configuration File

YAML file



```
# Where and how to store data.
storage:
  dbPath: D:\mongodb\db\00_ServerTest\

# where to write logging data.
systemLog:
  destination: file
  path:  D:\mongodb\db\00_ServerTest\mongod.log

# network interfaces
net:
  port: 27011
  bindIp: localhost

security:
 authorization: enabled
 keyFile: D:\mongodb\pki\m103-keyfile
```

Launch mongod with the --config command line option:

mongod --config *'d:\CSDL_NoSQL\config_files\mongod.cfg'*

Mongod – Basic Commands

Cover a few of the basic commands necessary to interact with the MongoDB cluster.

These methods are available in the **mongodb shell** that wrap underlying database commands.

- **db.<method>()**: DB shell helpers, interact with the database.
- **db.<collection>.<method>()**: shell helpers for collection level operations.
- **rs.<method>()**: rs helper methods, control replica set deployment and management.
- **sh.<method>()**: sh helper mrthods, control sharded cluster deployment and management.

(Read More)

Mongod – Basic Commands

User management, example:

- db.createUser()
- db.dropUser()

Collection management, example :

- db.<collection>.renameCollection(*<target>, <dropTarget>*) *[dropTarget: optional]*
- db.<collection>.createIndex(*<keys>, <options>, <commitQuorum>*)
- db.<collection>.drop(*<options>*)

Database management, example:

- db.dropDatabase(*<writeConcern>*) *[removes current database]*
- db.createCollection(*<name>, <options>*)

Database status, example:

- db.serverStatus()

Mongod – Basic Security

How do we secure the data?

Authentication	Authorization
• Verifies the identity of a user • Answers the question : Who are you?	• Verifies the previliges of a user • Answers the question: What do you have access to?

- **SCRAM**: default and most basic form of client authentication (password security)
- **X.509**: certificate for authentication, more secure and more complex
- LDAP
- Kerberos

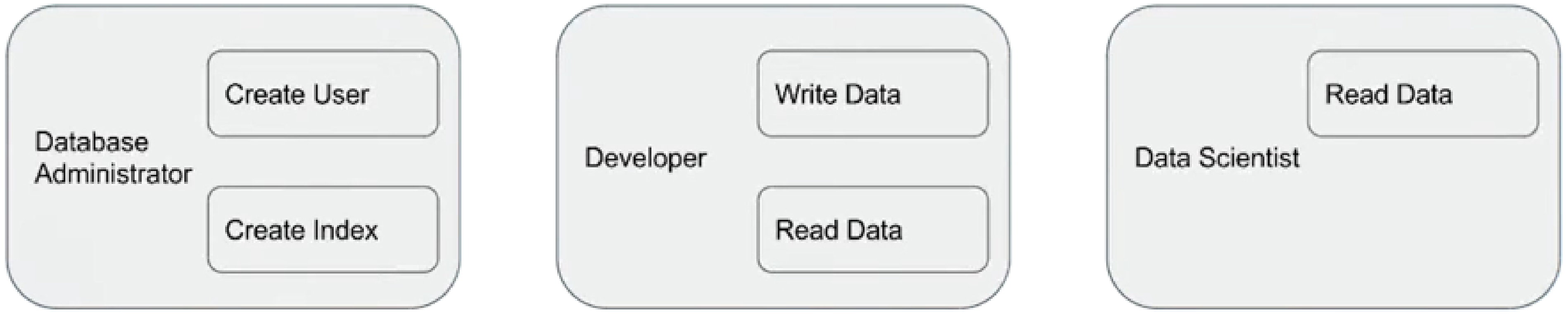
Only for MongoDB Enterprise

- Each user has one or more **Roles**.
- Each **Role** has one or more **Privileges**.
- A **Privilege** represent a group of **actions** and the **resources** that those actions apply to.

Mongod – Basic Security

Authorization: Role based access control

Roles support a high level of responsibility isolation for operational task:



- To enable role-based access control or authorization on cluster: enable authorization in configuration file (it implicitly enables authentication).
- By default, MongoDB doesn't give you any users.
- Always create a user with the administrative role first so you can create other users after.

```
#enable security
security:
    authorization: enabled
```

Mongod – Basic Security

Localhost exception:

- Allows you to access a MongoDB server that enforces authentication but does not yet configured user for you to authenticate with.
- Must run mongo/mongosh from the same host running MongoDB server.
- Localhost exception closes after you create your first user.
- Always create a user with administrative privileges first.

```
db.createUser {
        user: "<name>",
        pwd: passwordPrompt(),          // Or "<cleartext password>"
        customData: { <any information> },
        roles: [ { role: "<role>", db: "<database>" } | "<role>", ... ]
}
```

Mongod – Basic Security

Example:

Run MongoDB server that enforces authentication (no user created):

```
mongod --config 'D:\HeQTCSDL_NoSQL\config_files\mongod.conf'
```

```
net:
 port: 27017
 bindIp: 127.0.0.1
# enable security
security:
  authorization: enabled
```

Run mongosh from the same host running MongoDB server:

```
mongosh --host 127.0.0.1:27017
```

Create your first user:

```
use admin
db.createUser( { user : 'root', pwd : 'root', roles : ['root'] })
```

Exit mongosh then run again with 'root' user:

```
mongosh --username root --password root --authenticationDatabase admin
```

or

```
mongosh admin -u root -p root
```

Mongod – Basic Security

Roles in MongoDB

- **Build-In roles:** pre-packaged MongoDB roles.
- **Custom roles:** tailored roles to attend specific needs of specific users.
- **Database users:** will be granted roles to perform operations of MongoDB.

Mongod – Basic Security

Roles structure

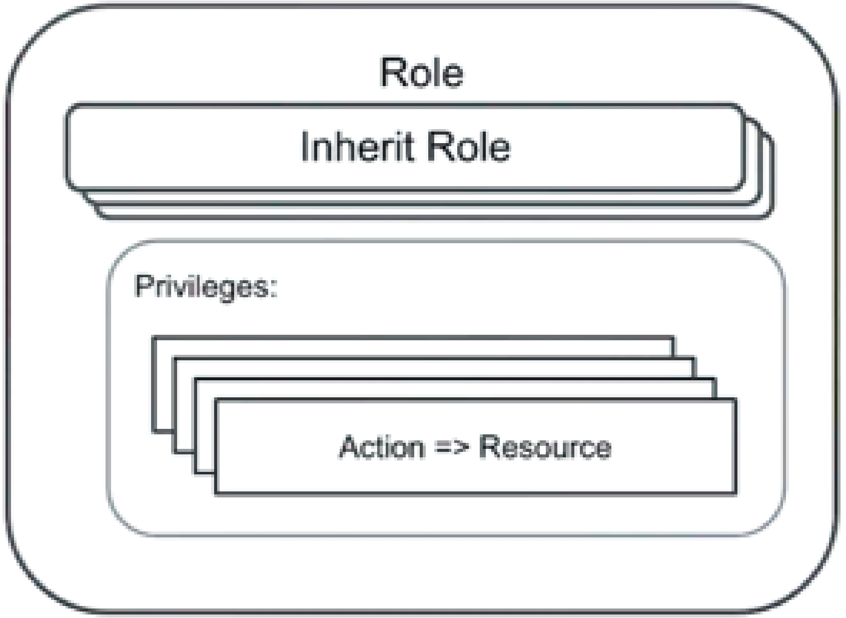
A role is composed of:

- Set of **privileges** that **role** enables
- All **privileges** that role defines will be made available to its users
- **Privilege** defines the **action**, or **actions**, that can be performed over a **resource**
- Resources:
  - Database
  - Collection or set of collections
  - Cluster: Replica set, Shard cluster

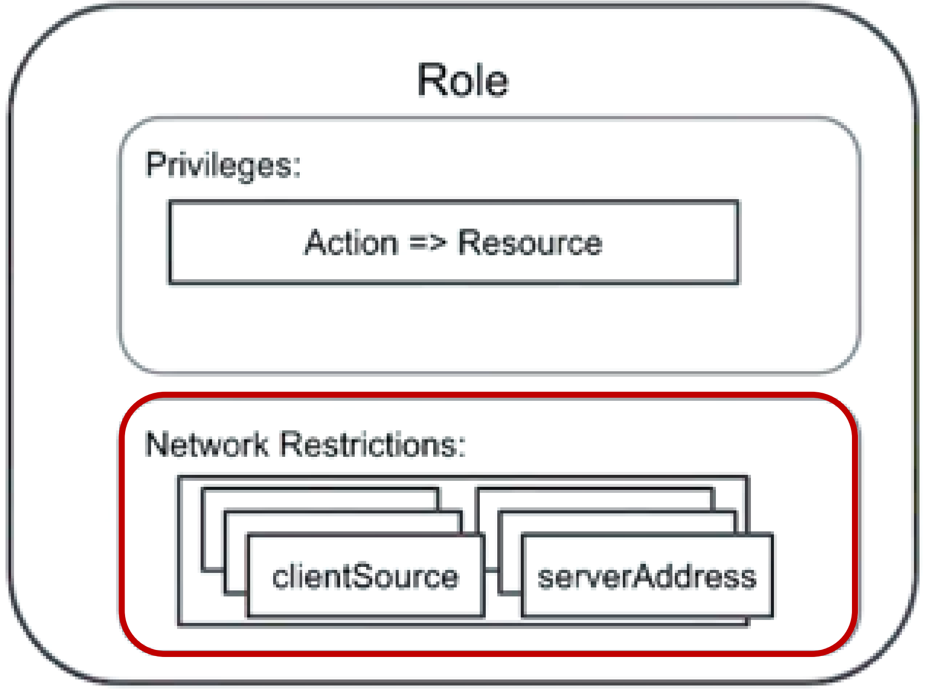
```
{resource: {cluster: true}, action: ['shutdown']}
```
A role with privilege, allowed to shut down any member of the cluster

Mongod – Basic Security

Roles structure



A role can also inherit from other roles



We can also define network authentication restrictions at the role level

Mongod – Basic Security

Build-In roles

Role levels	Roles
Database Users	read, readWrite
Database Administration	userAdmin, dbAdmin
Cluster Administration	clusterAdmin, clusterManager, clusterMonitor, hostManager
Backup and Restore	backup, restore
Super User	root (root is also a role at the all database level)
AllDatabase	readAnyDatabase, readWriteAnyDatabase dbAdminAnyDatabase, userAdminAnyDatabase

(read more Built-In Roles)

Mongod – Basic Security

Build-In Roles: userAdmin

- Allows user to do all operations around user management. Not able to do anything related with data management or data modifications.
- Provides the ability to create and modify roles and users on the current database. Since the userAdmin role allows users to grant any privilege to any user, including themselves, the role also indirectly provides superuser access to either the database or, if scoped to the admin database, the cluster.

(read more userAdmin role)

Example:

Run mongod with config file:

```
mongod --config 'D:\mongod.conf'
```

Run mongosh to connect to MongoDB server with root user:

```
mongosh admin -u root -p root
```

Create *securityUser* and grant *userAdmin* role

```
use admin        //all user should be created on the database admin for simplicity reasons
db.createUser( { user : 'securityUser', pwd : '123', roles : [ { db : 'admin', role : 'userAdmin' } ] }
)
```

Mongod – Basic Security

Build-In Roles: dbAdmin

- Provides the ability to perform administrative tasks such as schema-related tasks, indexing, and gathering statistics. This role does not grant privileges for user and role management.
- Everything that is related with DDL *(data definition language)*, this user will be able to perform.
- Everything that is related with the DML *(data modification language)* operations, he will not be able to do.

(read more dbAdmin role)

Example:

Create securityUser and grant dbAdmin role:

```
use admin
db.createUser( { user : 'DBAcourse', pwd : '123', roles : [ { db : 'mongoCourse', role : 'dbAdmin' } ] } )
```

//in this case, the roles of dbAdmin only be granted to mongoCourse db.

Roles can vary between databases. We can have a given user with different roles on a per database basis:

```
db.grantRolesToUser( 'DBAcourse', [ { db : 'reporting', role : 'dbOwner' } ] )
```

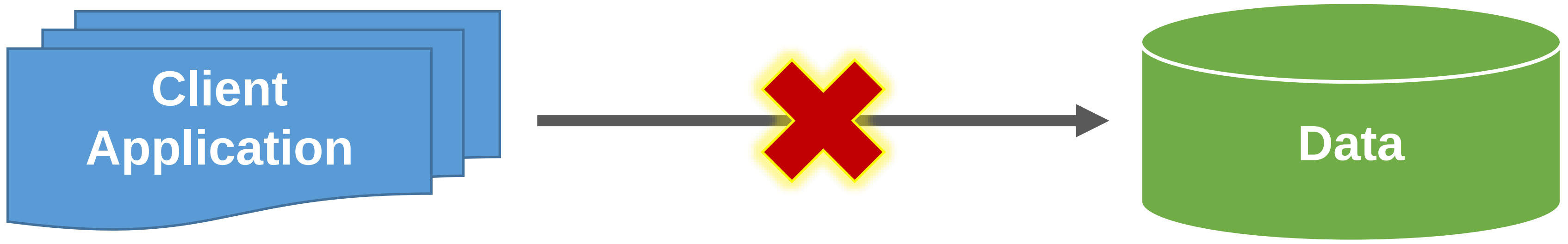
dbOwner role as a meta role. This role combines the privileges granted by the **readWrite**, **dbAdmin**, **userAdmin** roles

2. Replication

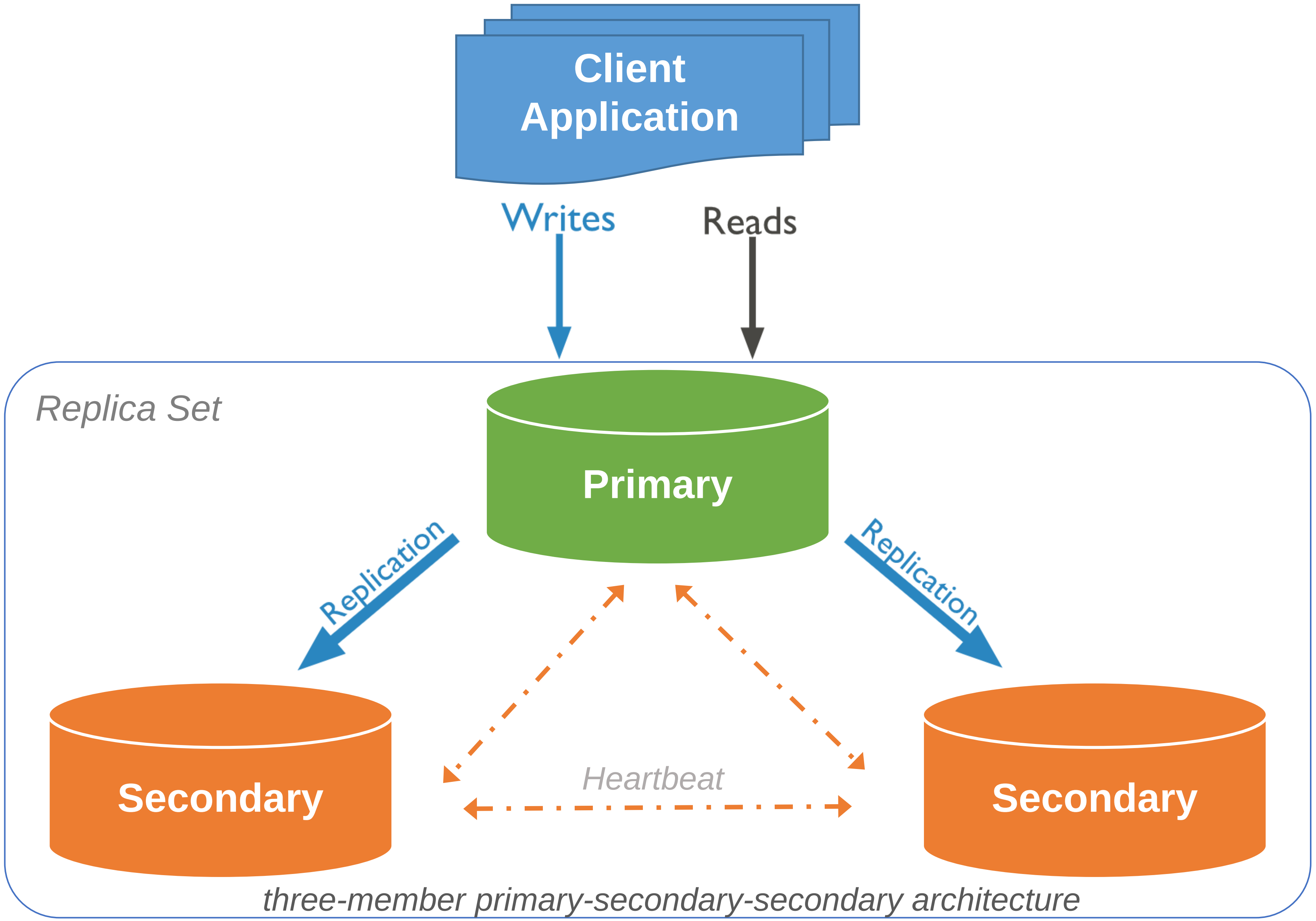
Replication

- Replication: Maintain multiple copies of your data – **Really important**
- Why**:**
  - Can never assume all servers will always be available
  - To make sure, if server goes down, you can still access your data → Redundancy and Data Availability
  - Replication can provide increased read capacity as clients can send read operations to different servers

If the database is hosted on a single server → standalone node



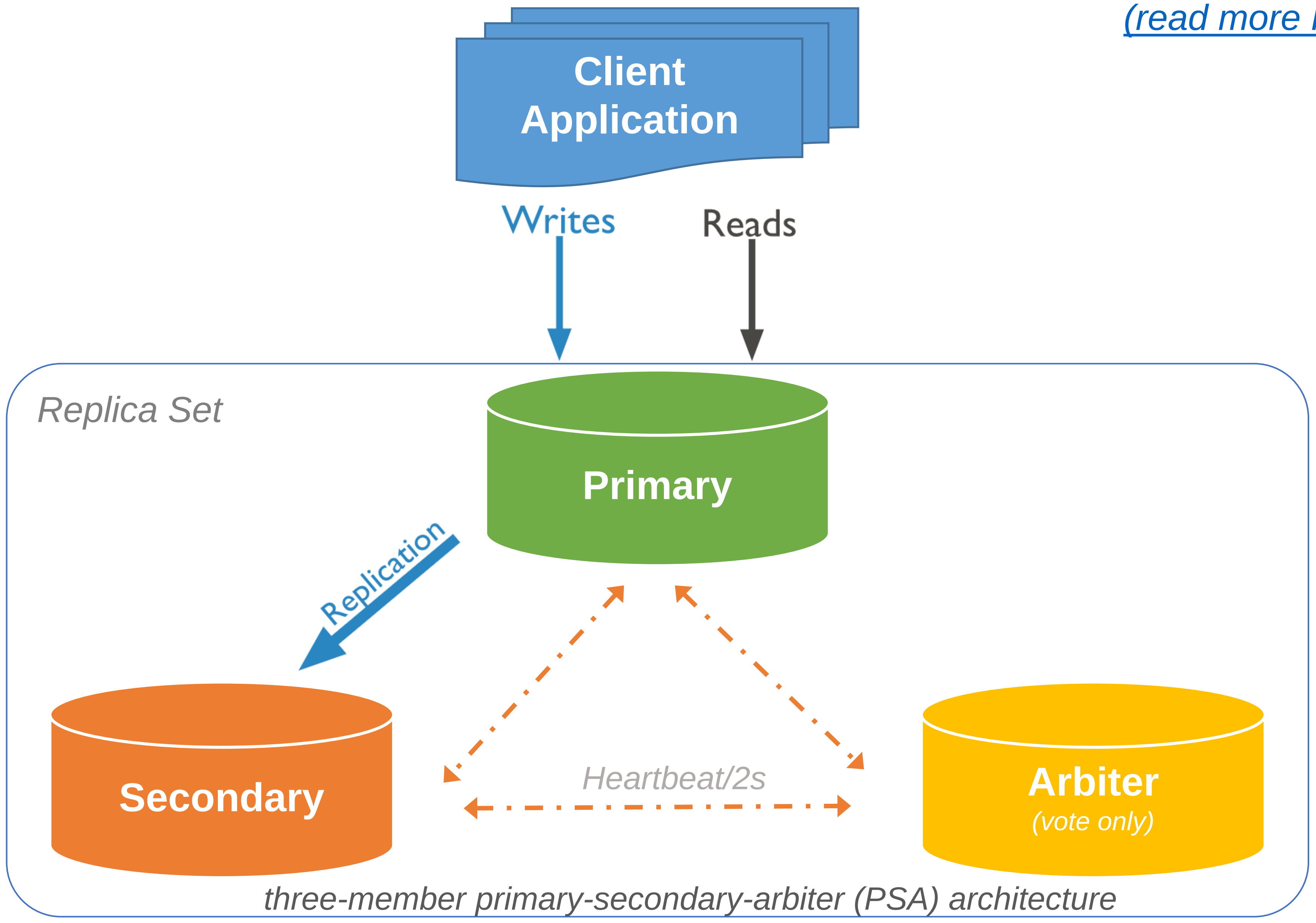
Replication



three-member primary-secondary-secondary architecture

Replication

(read more Replica Set Arbiter)

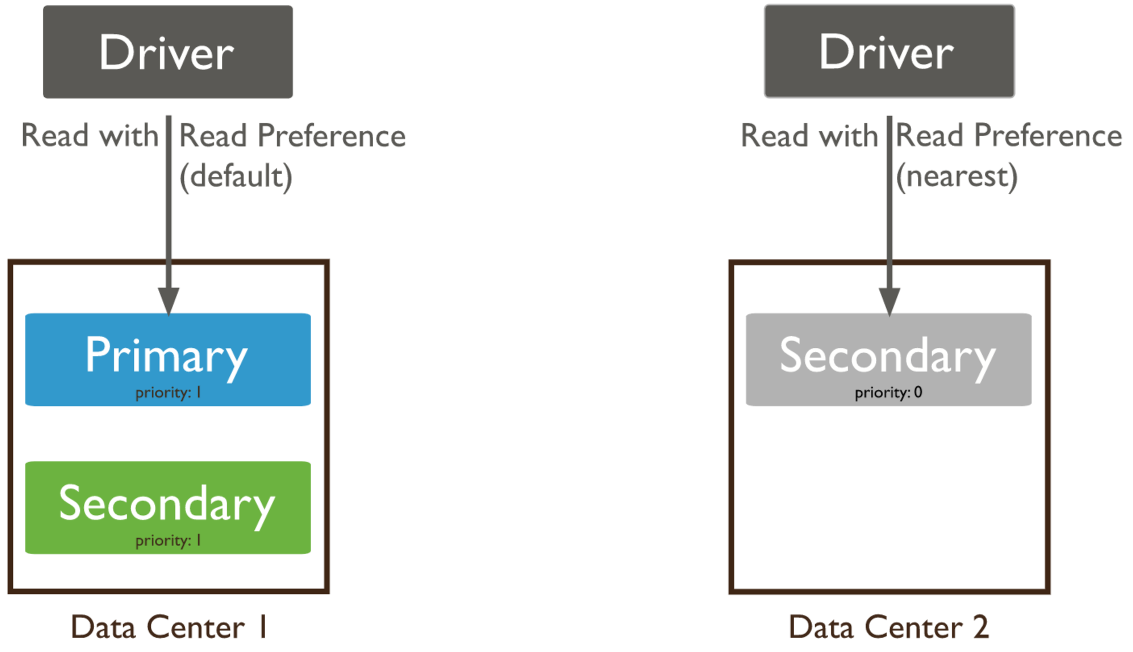


three-member primary-secondary-arbiter (PSA) architecture

Replication

- A replica set is a group of mongod instances that maintain the same data set. A replica set contains several data bearing nodes and optionally one arbiter node. Of the data bearing nodes, one and only one member is deemed the primary node, while the other nodes are deemed secondary nodes.

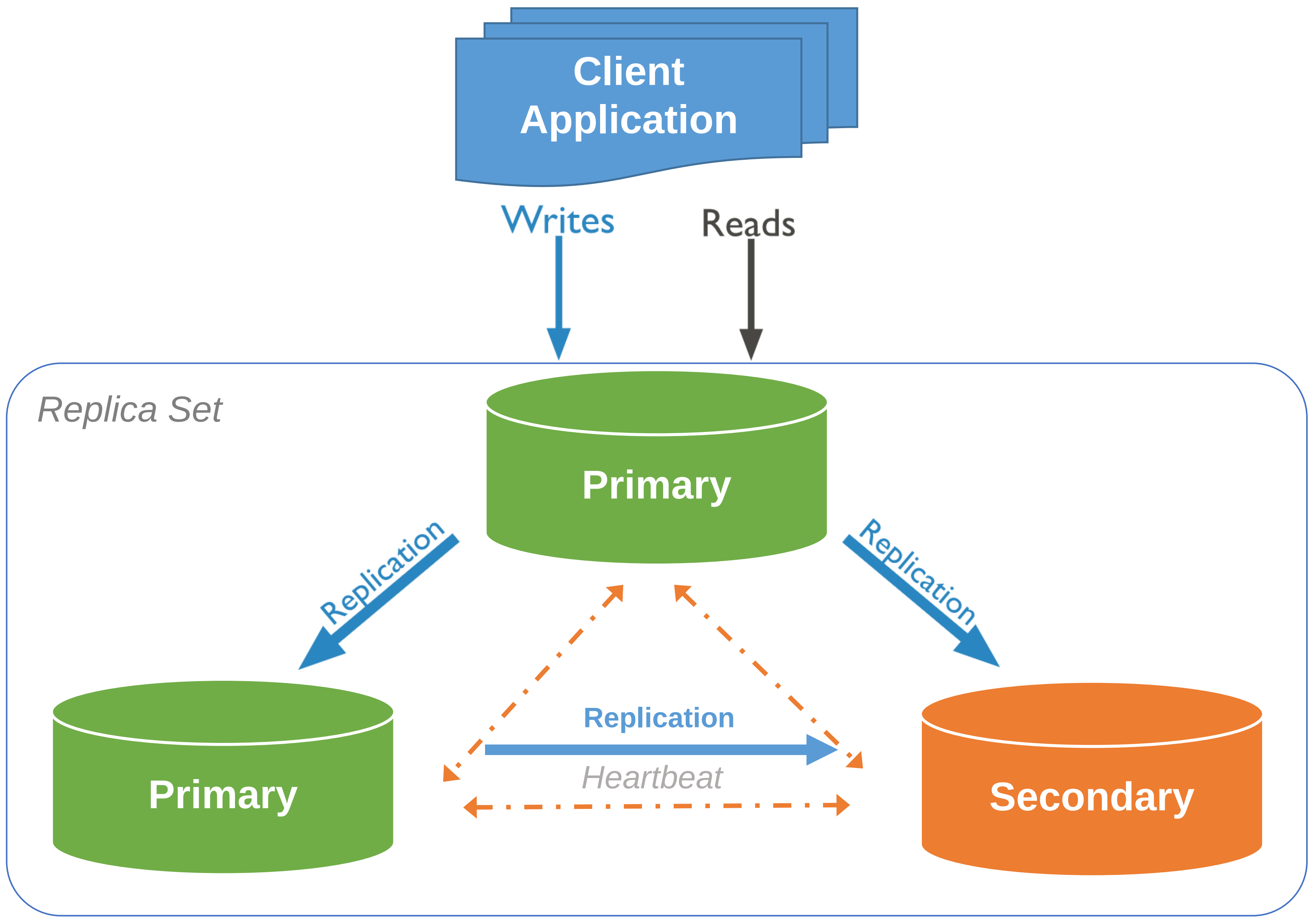
- Although clients cannot write data to secondaries, clients can read data from secondary members. See Read Preference for more information on how clients direct read operations to replica sets.



- A secondary can become a primary. If the current primary becomes unavailable, the replica set holds an election to choose which of the secondaries becomes the new primary.

(read more Replica Set)

Replication



Replication

- The replica set cannot process write operations until the election completes successfully. The replica set can continue to serve read queries if such queries are configured to run on secondaries.

  - The median time before a cluster elects a new primary should not typically exceed 12 seconds. *(read more Replica Set Elections)*

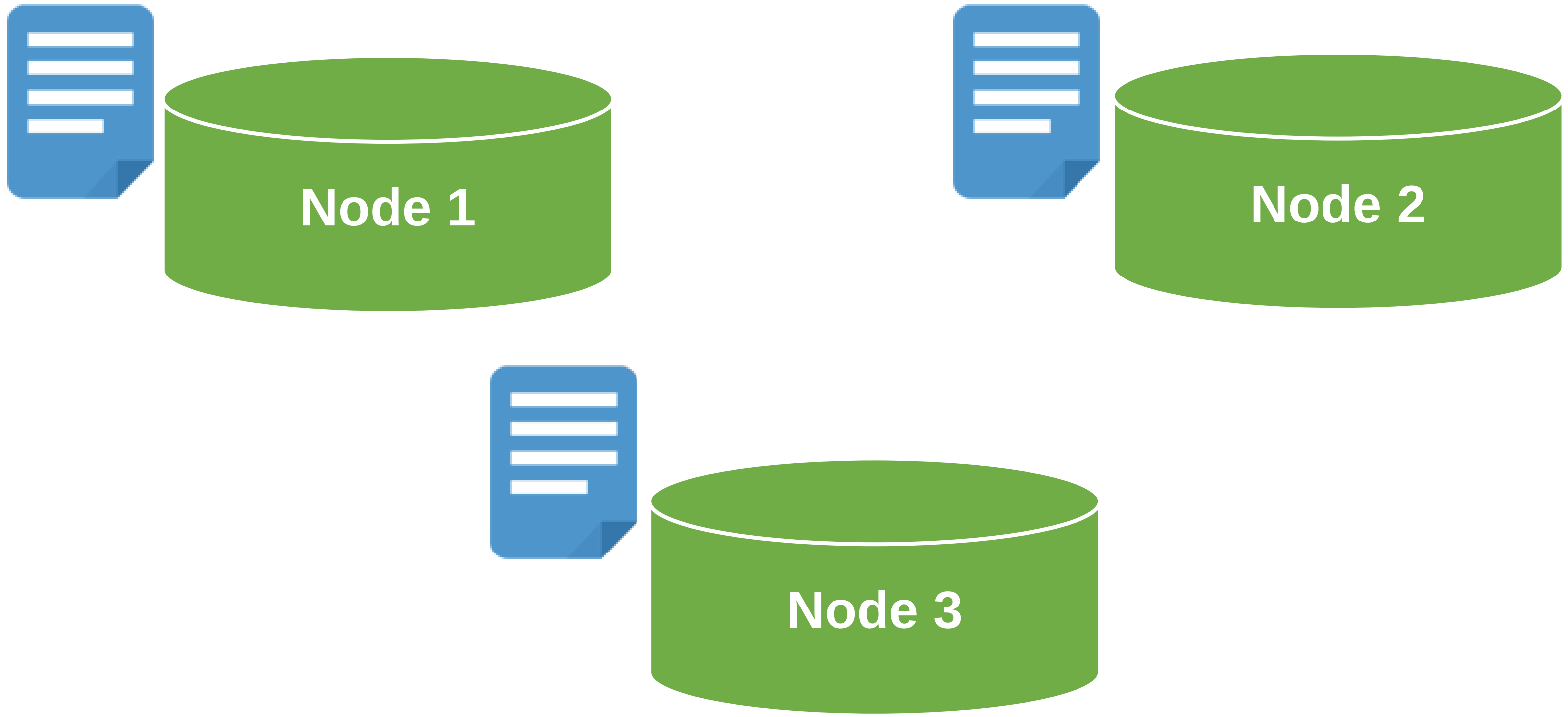
- You can configure a secondary member for a specific purpose. You can configure a secondary to:

  - Prevent it from becoming a primary in an election, which allows it to reside in a secondary data center or to serve as a cold standby. See Priority 0 Replica Set Members.
  - Prevent applications from reading from it, which allows it to run applications that require separation from normal traffic. See Hidden Replica Set Members.
  - Keep a running "historical" snapshot for use in recovery from certain errors, such as unintentionally deleted databases. See Delayed Replica Set Members.

Replication

Setting up a Replica Set

- mongod won't be able to communicate with each other until we connect them



Replication

Setting up a Replica Set:

1. Use configuration file for standalone mongod;
2. Start a mongod with configuration file;
3. Start a mongo and connect to one of mongo instance;
4. Initialize replica set;
5. Create root user;
6. Exit out of this mongo and then log back in as m-admin user;
7. Add nodes to Replica set

Replication

Setting up a Replica Set:
1- Use configuration file for standalone mongod

```
storage:
  dbPath: d:\mongodb\db\node1

net:
  bindIp: localhost
  port: 27011

security:
  authorization: enabled
  keyFile: d:\mongodb\pki\m-keyfile

systemLog:
  destination: file
  path: d:\mongodb\db\node1\mongod.log
  logAppend: true

replication:
  replSetName: rep-example
```

Optional, it is used to encrypt data exchanged between client application and mongodb

```
2

enabled
ngodb\pki\m-keyfile

e
odb\db\node2\mongod.log
ue

rep-example
```

```
ongodb\db\node3

ost

 enabled
ongodb\pki\m-keyfile

le
odb\db\node3\mongod.log
ue

: rep-example
```

Replication

Setting up a Replica Set:

2- Start a mongod with configuration file:

```
mongod --config 'c:\mongoDB\configs\node1.conf'
mongod --config 'c:\mongoDB\configs\node2.conf'
mongod --config 'c:\mongoDB\configs\node3.conf'
```

3- Start a mongo and connect to one of mongo instance:

```
mongo --host 127.0.0.1:27011        ||        mongo --host localhost:27011
```

4- Initialize replica set:

```
rs.initiate()
```

5- Create root user:

```
use admin
db.createUser({
  user: 'm-admin',
  pwd: 'm-pass',
  roles: [ { role : 'root', db : 'admin' } ]
})
```

5.1 set auth cho db : db.auth('m-admin','m-pass'})

Replication

Setting up a Replica Set

6- Exit out of this mongo and then log back in as m-admin user

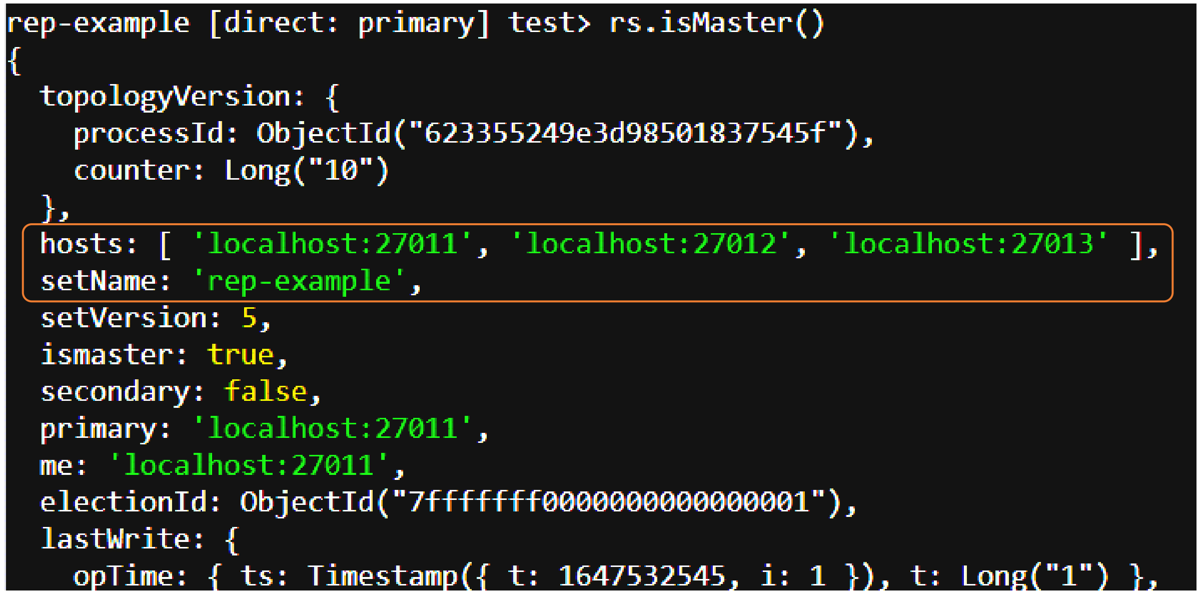
```
mongo --host m-example/localhost:27011 -u m-admin -p m-pass --authenticationDatabase admin
```

Replica set name

7- Add nodes to Replica set

```
rs.add( 'localhost:27012' )
rs.add( 'localhost:27013' )
```

To check status of Replica set: rs.status()
To check if the current node is primary: rs.isMaster()



```
rep-example [direct: primary] test> rs.isMaster()
{
  topologyVersion: {
    processId: ObjectId("623355249e3d98501837545f"),
    counter: Long("10")
  },
  hosts: [ 'localhost:27011', 'localhost:27012', 'localhost:27013' ],
  setName: 'rep-example',
  setVersion: 5,
  ismaster: true,
  secondary: false,
  primary: 'localhost:27011',
  me: 'localhost:27011',
  electionId: ObjectId("7fffffff0000000000000001"),
  lastWrite: {
    opTime: { ts: Timestamp({ t: 1647532545, i: 1 }), t: Long("1") },
```

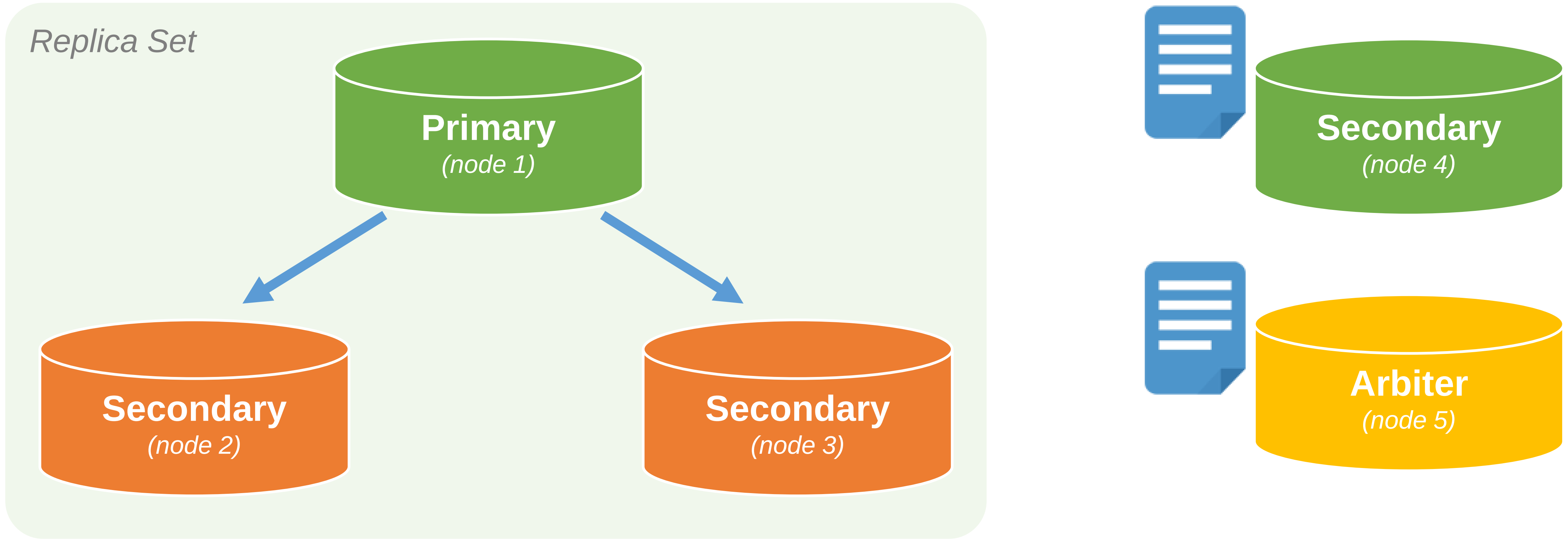
Replication

Replication Configuration Document:

- The replica set configuration document is a simple BSON document that we manage using a JSON representation
- Can be configured from the shell
- There are set of mongo shell replication helper methods that make it easier to manage
  - rs.add() : *Adds a member to a replica set.*
  - rs.addArb() : *Adds an arbiter to a replica set.*
  - rs.initiate() : *Initializes a new replica set.*
  - rs.remove() : *Remove a member from a replica set.*
  - rs.reconfig() : *Reconfigures a replica set by applying a new replica set configuration object.*
  - … *(seft study)*

Replication

Reconfiguring a Running Replica Set:



Replication

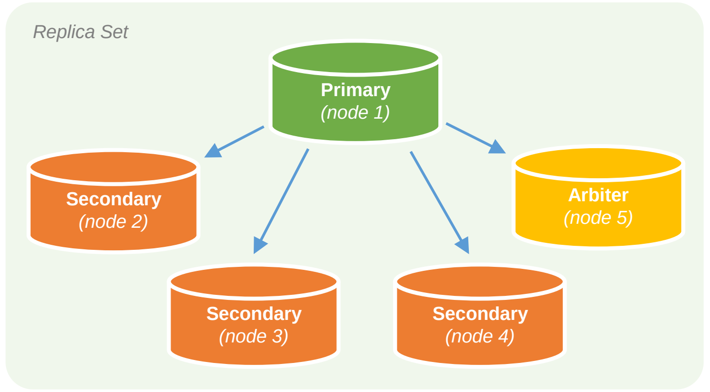
Reconfiguring a Running Replica Set:

1- Create config files for the secondaries 3 and arbiter nodes

```
storage:
  dbPath: d:\mongodb\db\node4

net:
  bindIp: localhost
  port: 27014

security:
  authorization: enabled
  keyFile: d:\mongodb\pki\m-keyfile

systemLog:
  destination: file
  path: d:\mongodb\db\node4\mongod.log
  logAppend: true

replication:
  replSetName: rep-example
```

```
storage:
  dbPath: d:\mongodb\db\arbiter

net:
  bindIp: localhost
  port: 28000

security:
  authorization: enabled
  keyFile: d:\mongodb\pki\m-keyfile

systemLog:
  destination: file
  path: d:\mongodb\db\arbiter\mongod.log
  logAppend: true

replication:
  replSetName: rep-example
```

Replication



Reconfiguring a Running Replica Set:

2- Starting up mongod processes for our fourth node and arbiter

```
mongod --config 'c:\mongoDB\configs\node4.conf’
mongod --config 'c:\mongoDB\configs\arbiter.conf’
```

3- Run Mongo shell and connect to the replica set m-example

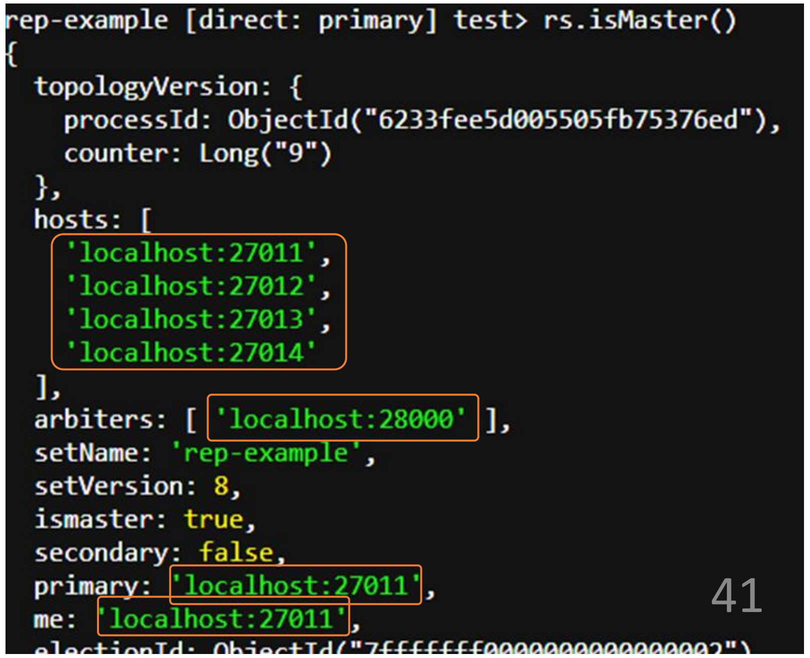
```
mongo --host m-example/localhost:27011 -u 'm-admin' -p 'm-pass' --authenticationDatabase 'admin’
```

4- From the Mongo shell of the replica set, adding the new secondary and the new arbiter:

```
rs.add( 'localhost:27014’ )
rs.addArb( 'localhost:28000’ )
```

5- Checking replica set make up after adding two new nodes:

```
rs.isMaster()
```



```
rep-example [direct: primary] test> rs.isMaster()
{
  topologyVersion: {
    processId: ObjectId("6233fee5d005505fb75376ed"),
    counter: Long("9")
  },
  hosts: [
    'localhost:27011',
    'localhost:27012',
    'localhost:27013',
    'localhost:27014'
  ],
  arbiters: [ 'localhost:28000' ],
  setName: 'rep-example',
  setVersion: 8,
  ismaster: true,
  secondary: false,
  primary: 'localhost:27011',
  me: 'localhost:27011',
```

Replication

Reconfiguring a Running Replica Set:

- Removing the arbiter from our replica set:

```
rs.remove( 'localhost:28000' )
```

- Assigning the current configuration to a shell variable we can edit, in order to reconfigure the replica set:

```
cfg = rs.conf()
```

- Editing our new variable cfg to change topology - specifically, by modifying cfg.members:

```
cfg.members[3].votes = 0
cfg.members[3].hidden = truea
cfg.members[3].priority = 0
```

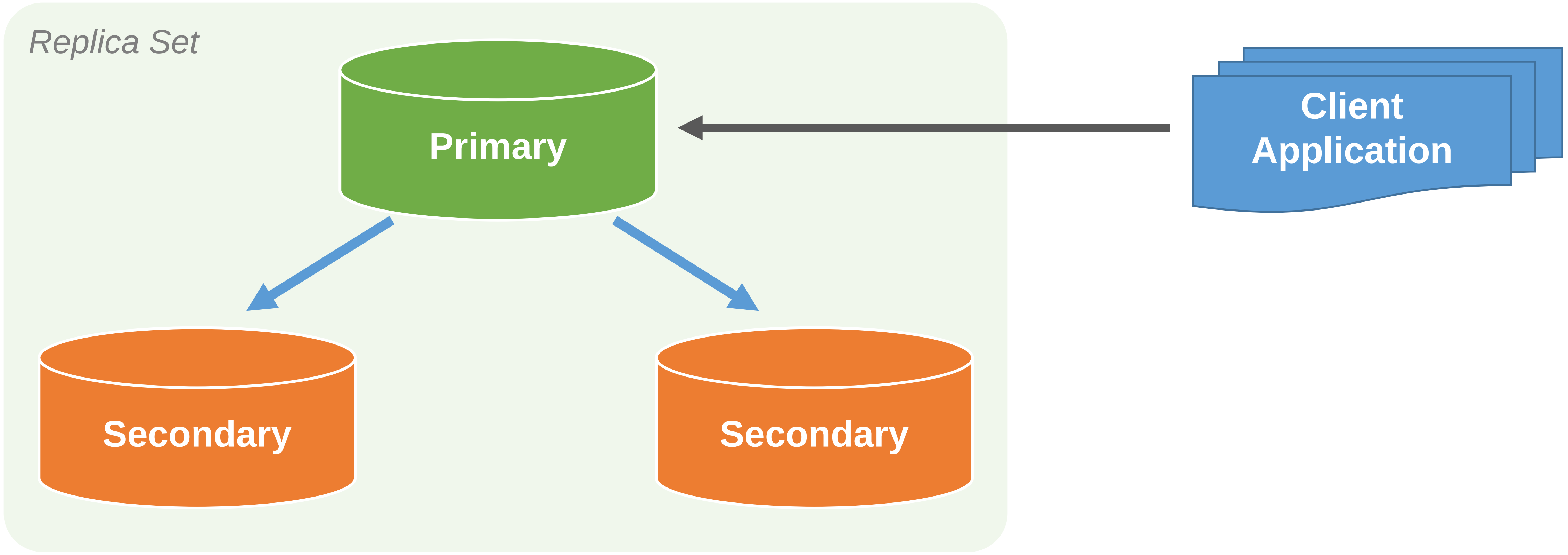
- Updating our replica set to use the new configuration cfg:

```
rs.reconfig( cfg )
```

Replication

Failover and Elections:

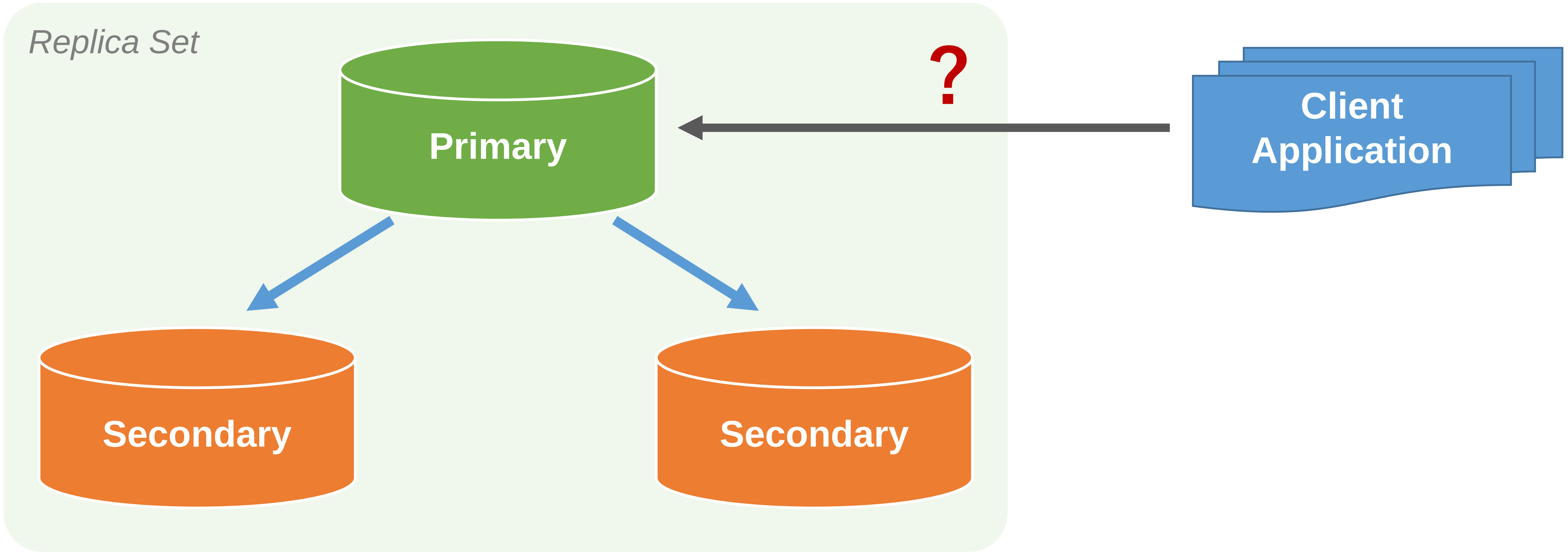
- Primary node is the first point where the client application accesses the database.
- If secondaries go down, the client will continue communicating with the node acting as primary until the primary is unavailable.



Replication

Failover and Elections:

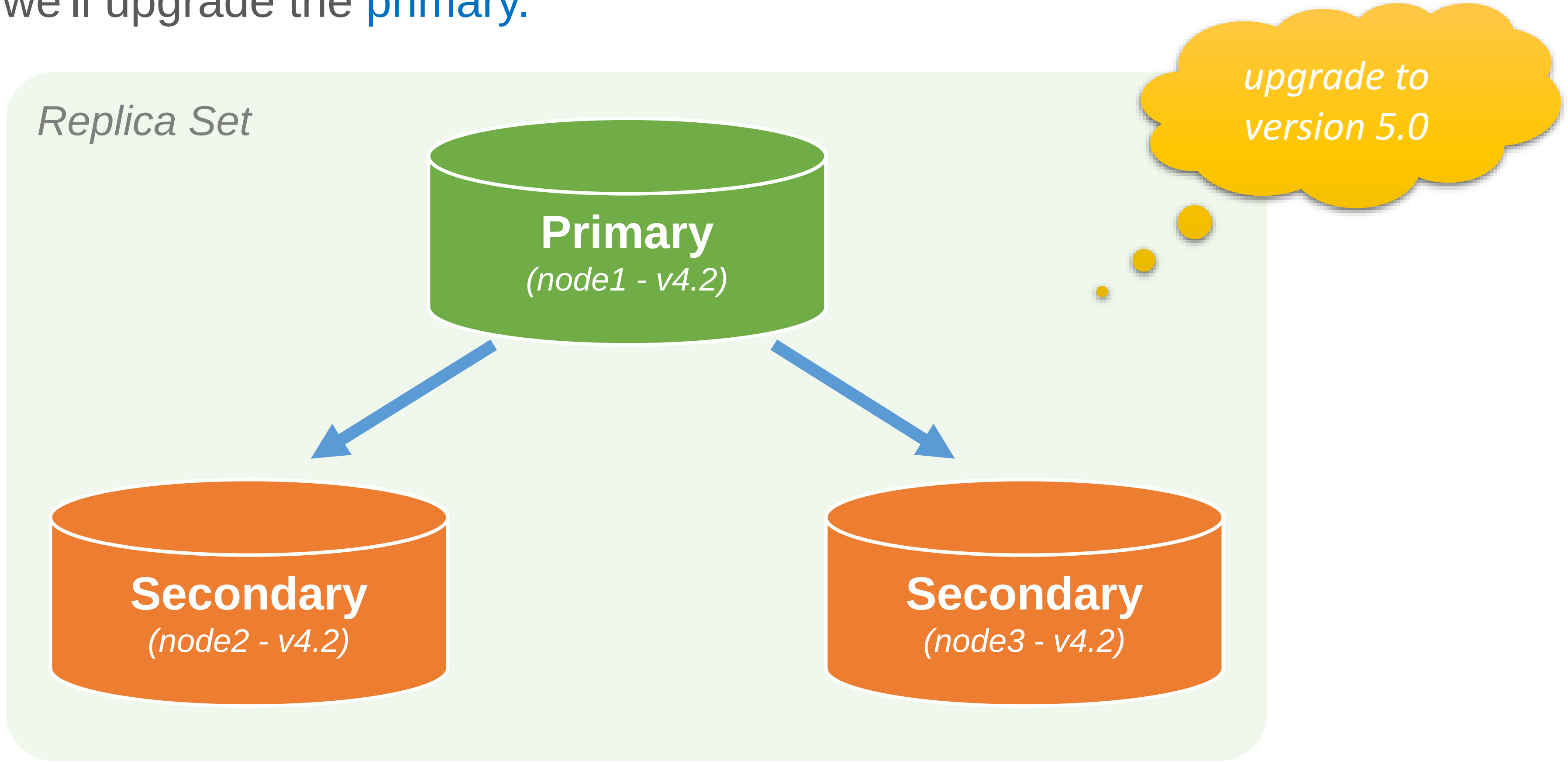
- What would cause a primary to become unavailable? → a common reason is maintenance.



Replication

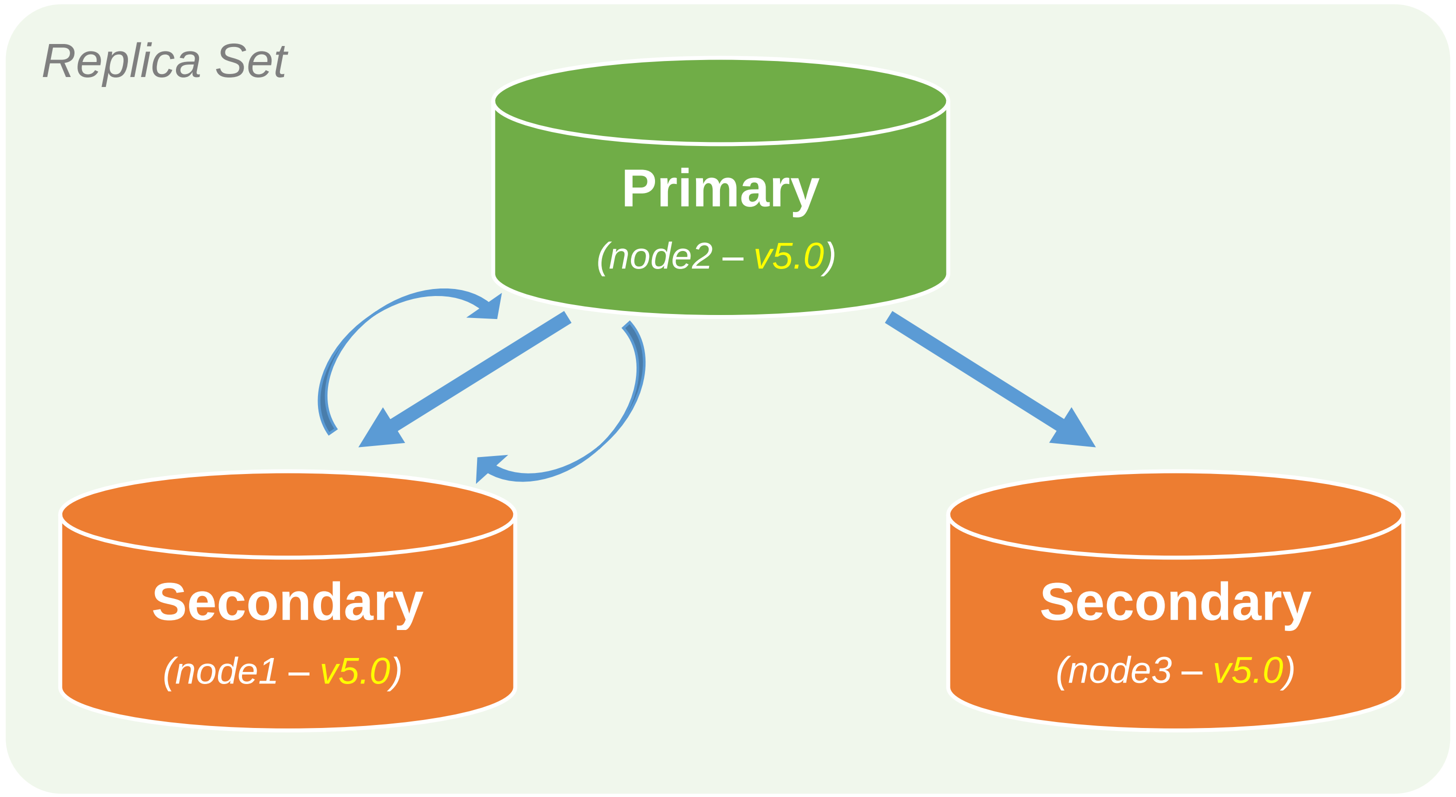
Failover and Elections:

- Let's say we want to roll upgrade on a three nodes replica set.
- A rolling upgrade just means we're upgrading one server at a time, starting with the secondaries and eventually, we'll upgrade the primary.



Replication

Failover and Elections:



3. Sharding

Mongod – Sharding

What is Sharding?

- In a replica set, we have more than one server in our database and each server has to contain the entire dataset
- What do we do when the data grows, and the servers can't work properly?

  Vertical Scaling: increase the capacity of individual server:

  - More RAM
  - More disk space
  - More powerful CPU

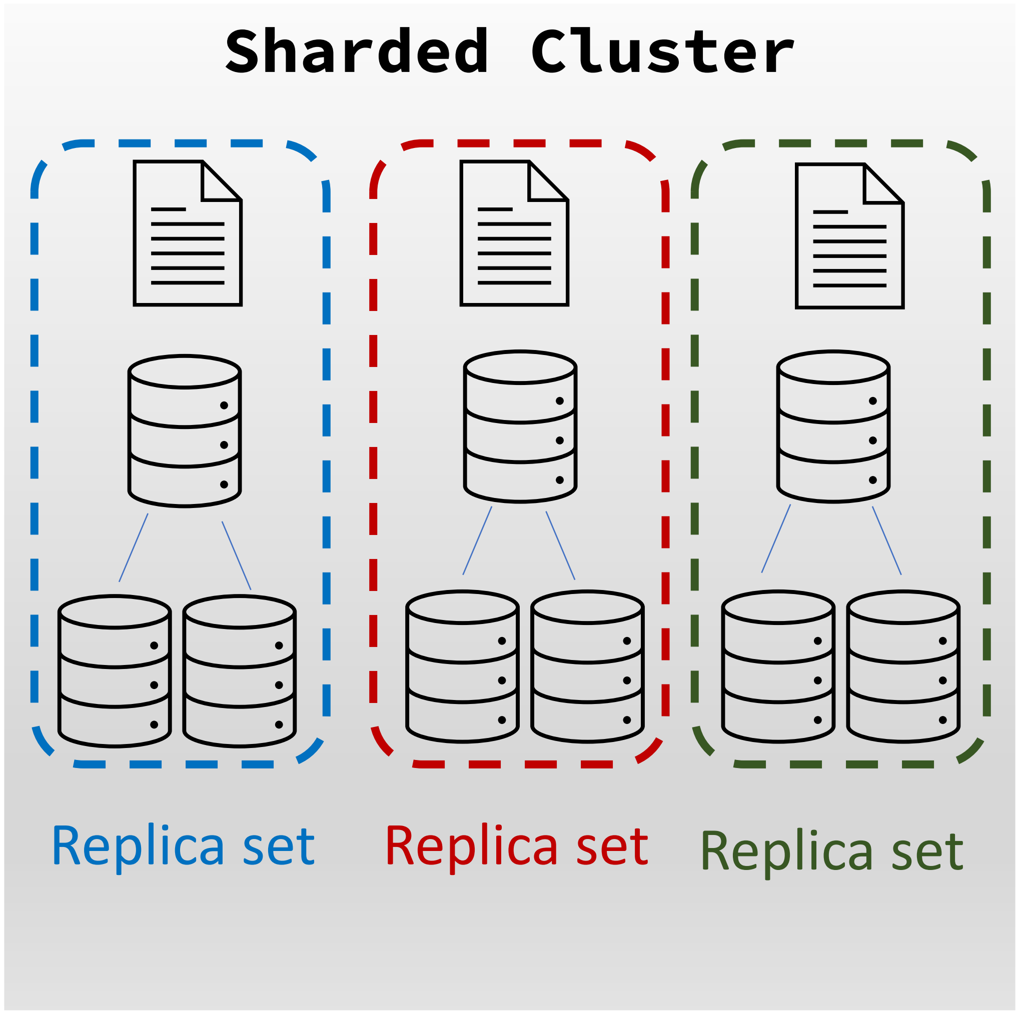
  $\Rightarrow$

  - Potentially become very expensive
  - Cloud-based providers aren't going to let us scale vertically forever

Mongod – Sharding

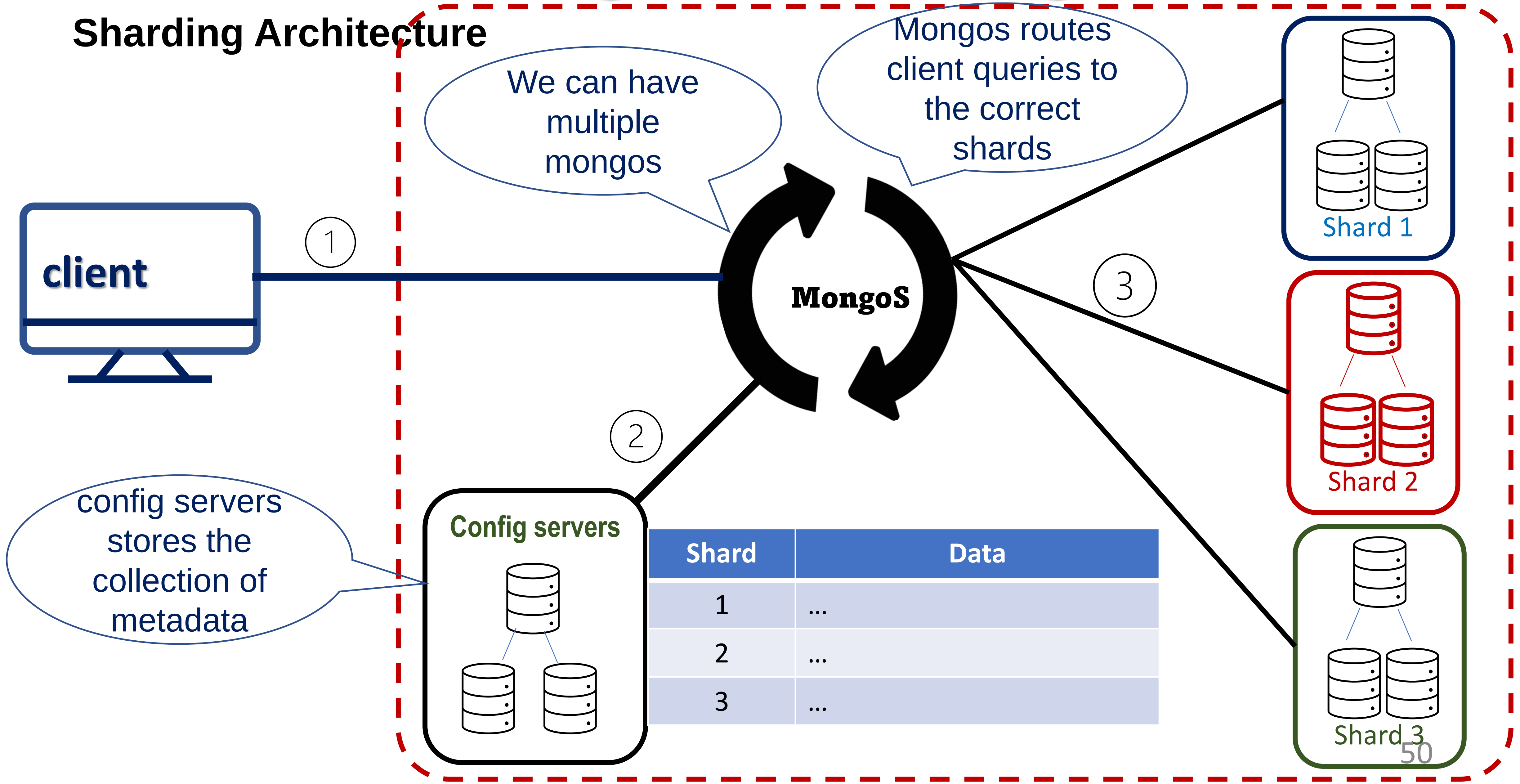
What is Sharding?

- MongoDB, scaling is done horizontally
- The way we distribute data in MongoDB is called Sharding
- Sharding allows us to grow our dataset without worrying about being able to store it all on one server
- To guarantee high availability in our Sharded Cluster, we deploy each shard as a replica set



Mongod – Sharding

Sharding Architecture

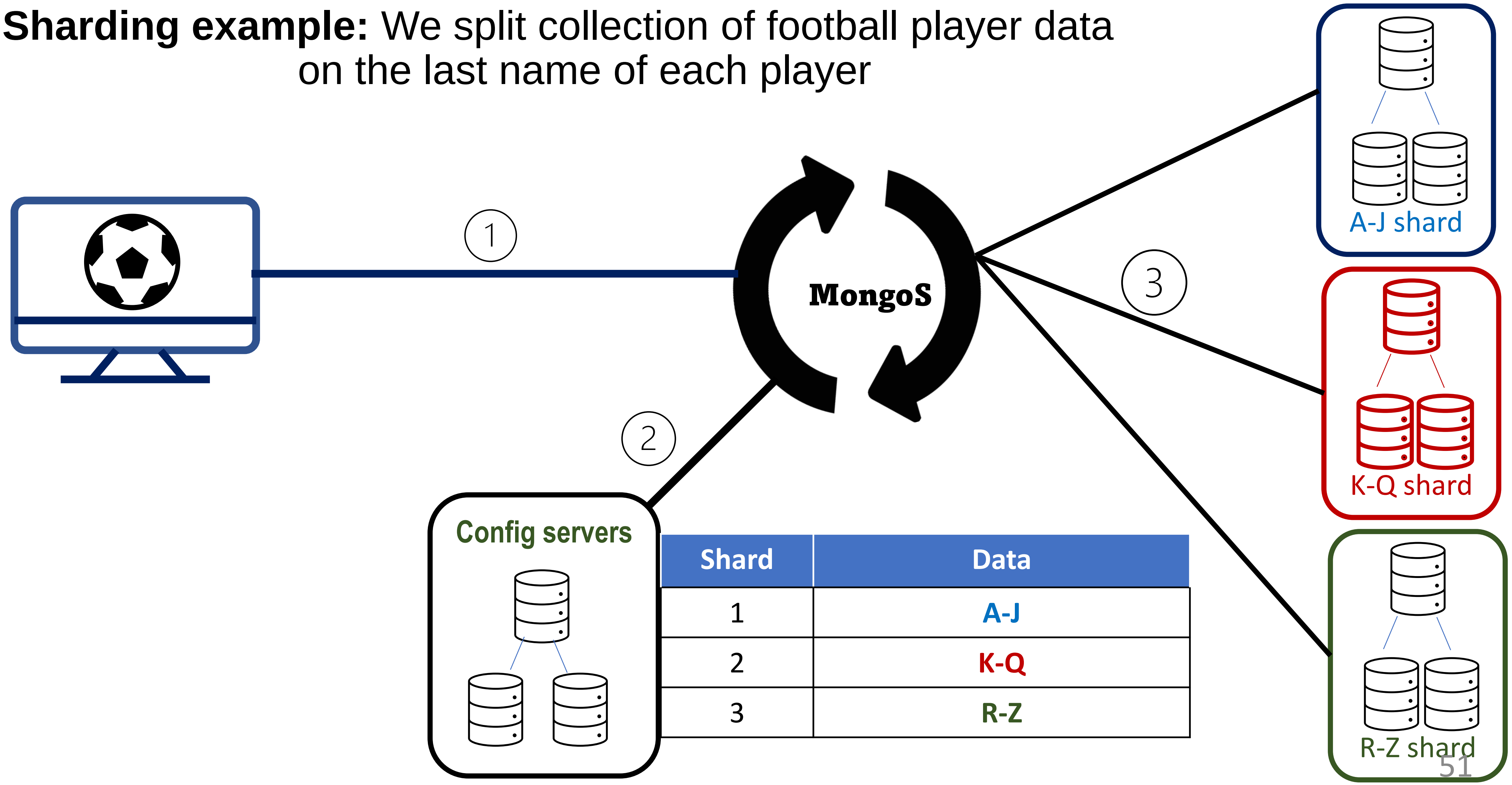


client
1
We can have multiple mongos
Mongos routes client queries to the correct shards
MongoS
3
Shard 1
Shard 2
Shard 3
2
Config servers
config servers stores the collection of metadata

Shard	Data
1	…
2	…
3	…

Mongod – Sharding

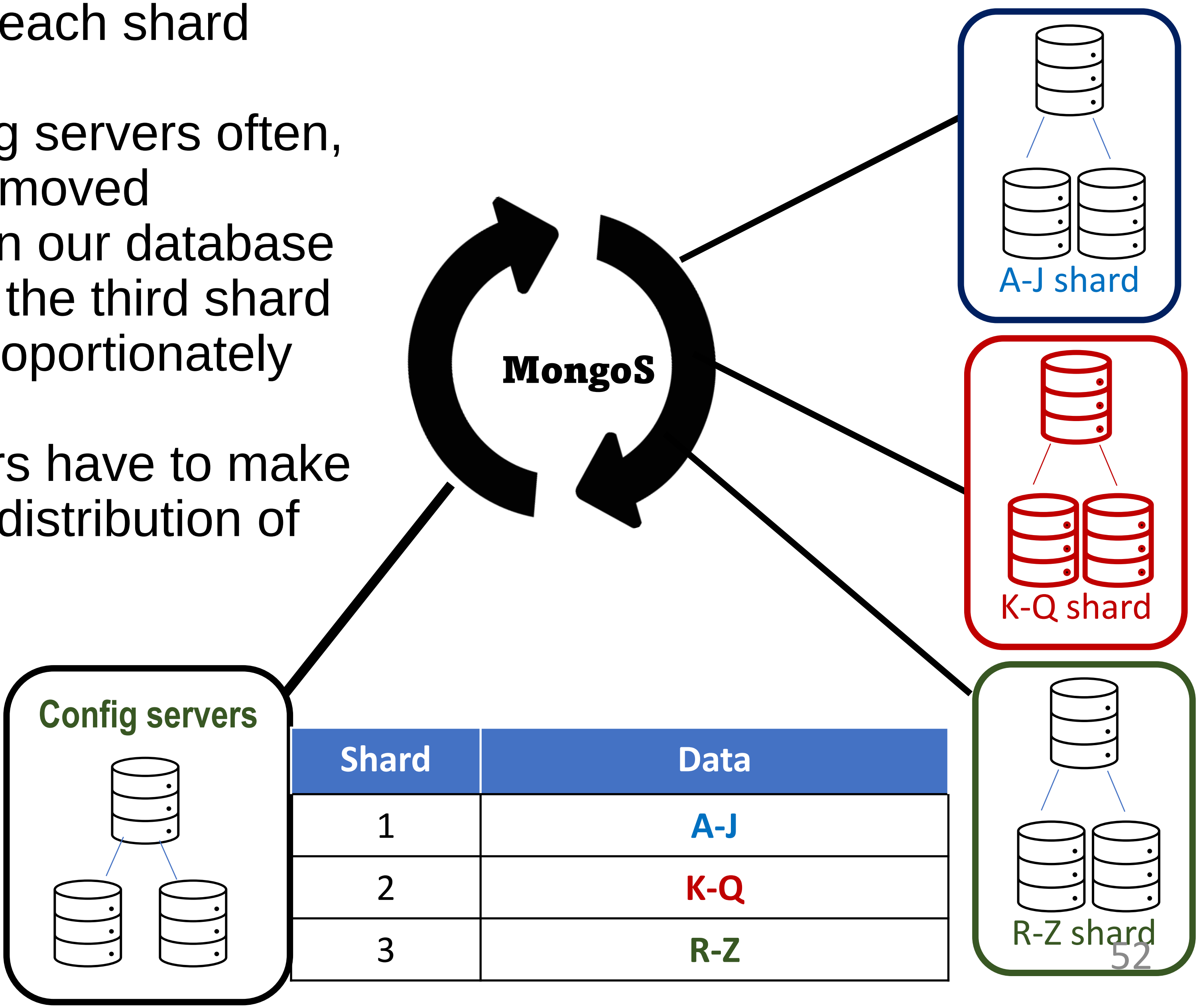
Sharding example: We split collection of football player data on the last name of each player



Shard	Data
1	A-J
2	K-Q
3	R-Z

Mongod – Sharding

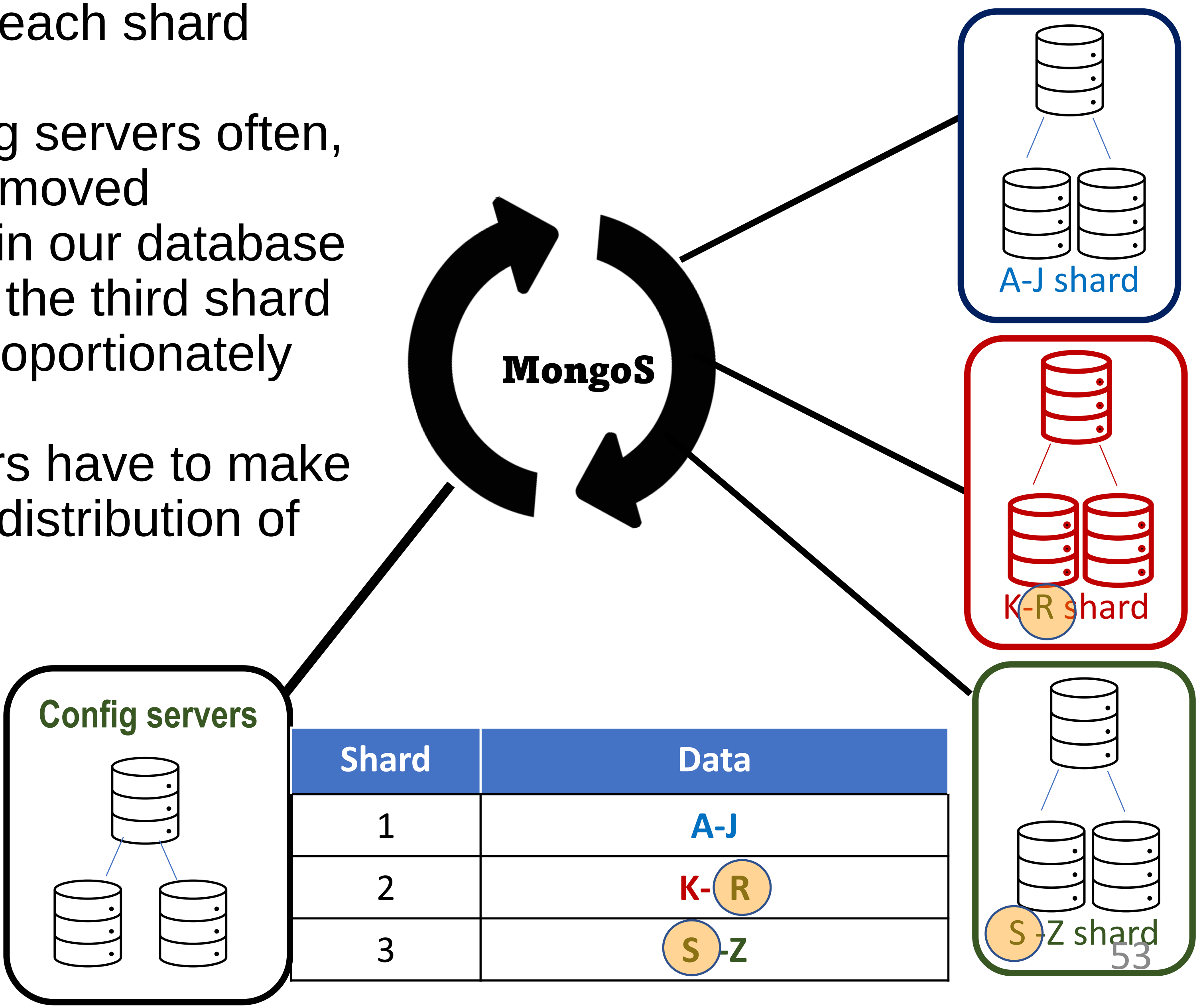
- Information contained on each shard might change with time
- Mongos queries the config servers often, in case a piece of data is moved
- Example: a lot of people in our database with the last name Smith, the third shard is going to contain a disproportionately large amount of data
- In that case, config servers have to make sure that there's an even distribution of data across each part



Shard	Data
1	A-J
2	K-Q
3	R-Z

Mongod – Sharding

- Information contained on each shard might change with time
- Mongos queries the config servers often, in case a piece of data is moved
- **Example**: a lot of people in our database with the last name Smith, the third shard is going to contain a disproportionately large amount of data
- In that case, config servers have to make sure that there's an even distribution of data across each part

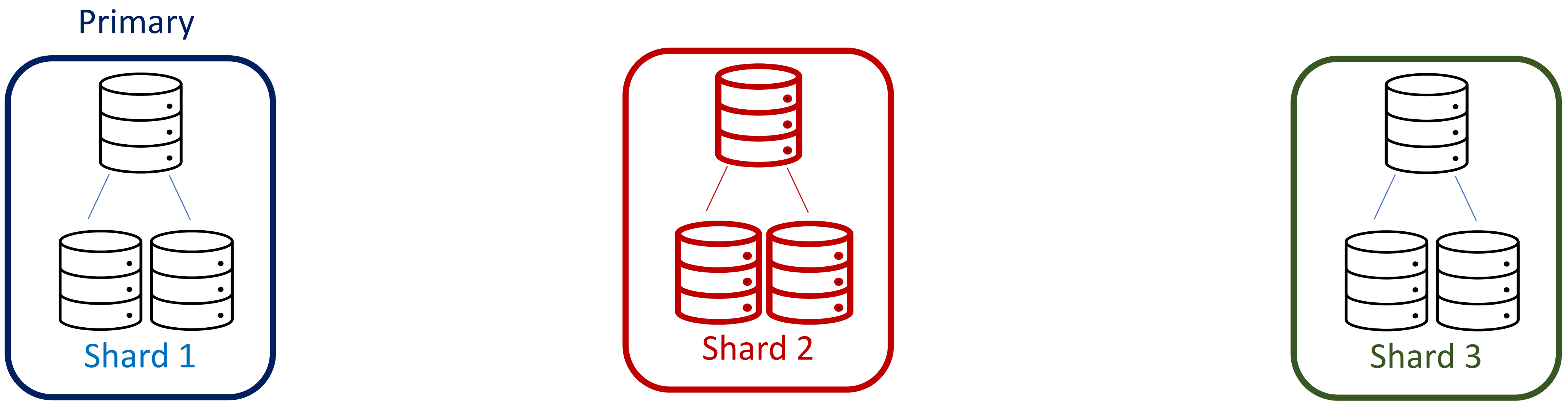


Shard	Data
1	A-J
2	K-R
3	S-Z

Mongod – Sharding

Primary Shard

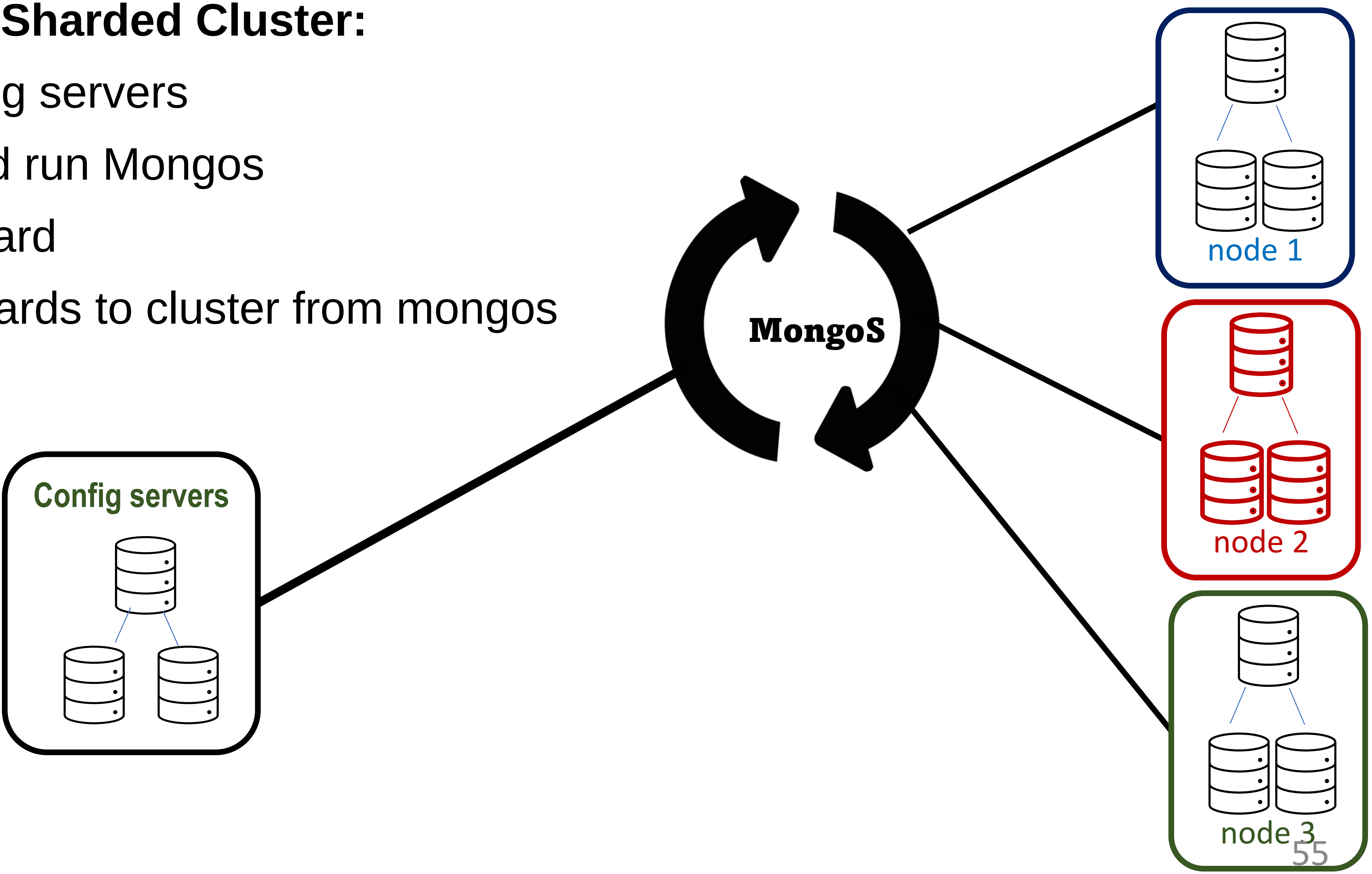
- In the sharded cluster, we have the primary shard
- Each database will be assigned a primary shard
- All the non-sharded collections on that database will remain on primary shard *(not all the collections in a sharded cluster need to be distributed)*



Mongod – Sharding

Setting Up a Sharded Cluster:

- Build config servers
- Config and run Mongos
- Config Shard
- Adding shards to cluster from mongos



Mongod – Sharding

Setting Up a Sharded Cluster: 1) Build config servers

- Create configuration file for config servers:

C:\mongoDB\configs\csrs_1.conf

```
sharding:
  clusterRole: configsvr
replication:
  replSetName: m-csrs
security:
  keyFile: C:\mongoDB\pki\m-keyfile
net:
  bindIp: localhost
  port: 26001
systemLog:
  destination: file
  path:  C:\mongoDB\db\csrs1\mongod.log
  logAppend: true
storage:
  dbPath:  C:\mongoDB\db\csrs1
```

Mongod – Sharding

Setting Up a Sharded Cluster: 1) Build config servers

- Create configuration file for config servers:

C:\mongoDB\configs\csrs_2.conf

```
sharding:
  clusterRole: configsvr
replication:
  replSetName: m-csrs
security:
  keyFile: C:\mongoDB\pki\m-keyfile
net:
  bindIp: localhost
  port: 26002
systemLog:
  destination: file
  path:  C:\mongoDB\db\csrs2\mongod.log
  logAppend: true
storage:
  dbPath:  C:\mongoDB\db\csrs2
```

Mongod – Sharding

Setting Up a Sharded Cluster: 1) Build config servers

- Create configuration file for config servers:

C:\mongoDB\configs\csrs_3.conf

```
sharding:
  clusterRole: configsvr
replication:
  replSetName: m-csrs
security:
  keyFile: C:\mongoDB\pki\m-keyfile
net:
  bindIp: localhost
  port: 26003
systemLog:
  destination: file
  path:  C:\mongoDB\db\csrs3\mongod.log
  logAppend: true
storage:
  dbPath:  C:\mongoDB\db\csrs3
```

Mongod – Sharding

Setting Up a Sharded Cluster: 1) Build config servers

- Starting the config servers:

```
mongod --config 'c:\mongoDB\configs\csrs_1.conf'

mongod --config 'c:\mongoDB\configs\csrs_2.conf'

mongod --config 'c:\mongoDB\configs\csrs_3.conf'
```

Mongod – Sharding

Setting Up a Sharded Cluster: 1) Build config servers

- Run mongo shell and connect to one of the config servers:

```
mongo --port 26001
```

- Initiating the CSRS (from mongo shell):

```
rs.initiate()
```

- Creating super user on CSRS (from mongo shell):

```
use admin
db.createUser({
        user: 'm-admin',
        pwd: 'm-pass',
        roles: [
                {role: 'root', db: 'admin'}
        ]
})
```

Mongod – Sharding

Setting Up a Sharded Cluster: 1) Build config servers

- Authenticating as the super user (from mongo shell)):

```
db.auth('m-admin', 'm-pass')
```

- Initiating the CSRS (from mongo shell):

```
rs.initiate()
```

- Add the second and third node to the CSRS replica set:

```
use admin
rs.add('localhost:26002')
rs.add('localhost:26003')
```

Mongod – Sharding

Setting Up a Sharded Cluster: 2) Config and run Mongos

- Mongos config (mongos.conf):

C:\mongoDB\configs\mongos.conf

```
sharding:
  configDB: m-csrs/localhost:26001,localhost:26002,localhost:26003
security:
  keyFile: C:\mongoDB\pki\m-keyfile
net:
  bindIp: localhost
  port: 26000
systemLog:
  destination: file
  path: C:\mongoDB\db\mongos.log
  logAppend: true
```

Mongod – Sharding

Setting Up a Sharded Cluster: 2) Config and run Mongos

- Start the mongos server:

```
mongos --config 'c:\mongoDB\configs\mongos.conf
```

- Run mongo shell and connect to mongos:

```
mongo --port 26000 --username m-admin --password m-pass --authenticationDatabase
```

- Check sharding status:

```
sh.status()
```

Mongod – Sharding

Setting Up a Sharded Cluster: 3. Config Shard [1] (using of Replica set m-example)

- Updated configuration for node1.conf, node2.conf, node3.conf :

C:\mongoDB\configs\node1.conf

```
sharding:
  clusterRole: shardsvr
storage:
  dbPath: C:\mongoDB\db\node1
  wiredTiger:
    engineConfig:
      cacheSizeGB: .25
net:
  bindIp: localhost
  port: 27011
security:
  authorization: enabled
  keyFile: C:\mongoDB\pki\m-keyfile
systemLog:
  destination: file
  path: C:\mongoDB\db\node1\mongod.log
  logAppend: true
```

Mongod – Sharding

Setting Up a Sharded Cluster: 3. Config Shard [1] (using of Replica set m-example)

- Updated configuration for node1.conf, node2.conf, node3.conf :

C:\mongoDB\configs\node2.conf

```
sharding:
  clusterRole: shardsvr
storage:
  dbPath: C:\mongoDB\db\node2
  wiredTiger:
    engineConfig:
      cacheSizeGB: .25
net:
  bindIp: localhost
  port: 27012
security:
  authorization: enabled
  keyFile: C:\mongoDB\pki\m-keyfile
systemLog:
  destination: file
  path: C:\mongoDB\db\node2\mongod.log
  logAppend: true
replication:
  replSetName: m-example
```

Mongod – Sharding

Setting Up a Sharded Cluster: 3. Config Shard [1] (using of Replica set m-example)

- Updated configuration for node1.conf, node2.conf, node3.conf :

C:\mongoDB\configs\node3.conf

```
sharding:
  clusterRole: shardsvr
storage:
  dbPath: C:\mongoDB\db\node3
  wiredTiger:
    engineConfig:
      cacheSizeGB: .25
net:
  bindIp: localhost
  port: 27013
security:
  authorization: enabled
  keyFile: C:\mongoDB\pki\m-keyfile
systemLog:
  destination: file
  path: C:\mongoDB\db\node3\mongod.log
  logAppend: true
replication:
  replSetName: m-example
```

Mongod – Sharding

Setting Up a Sharded Cluster: 3. Config Shard [1] (using of Replica set m-example)

- Run mongod with corresponding config files: node1.conf, node2.conf, node3.conf:

```
mongod --config 'c:\mongoDB\configs\node1.conf'

mongod --config 'c:\mongoDB\configs\node2.conf'

mongod --config 'c:\mongoDB\configs\node3.conf'
```

Mongod – Sharding

Setting Up a Sharded Cluster: 4) Adding new shard to cluster from mongos

```
sh.addShard('m-example/localhost:27012')
```

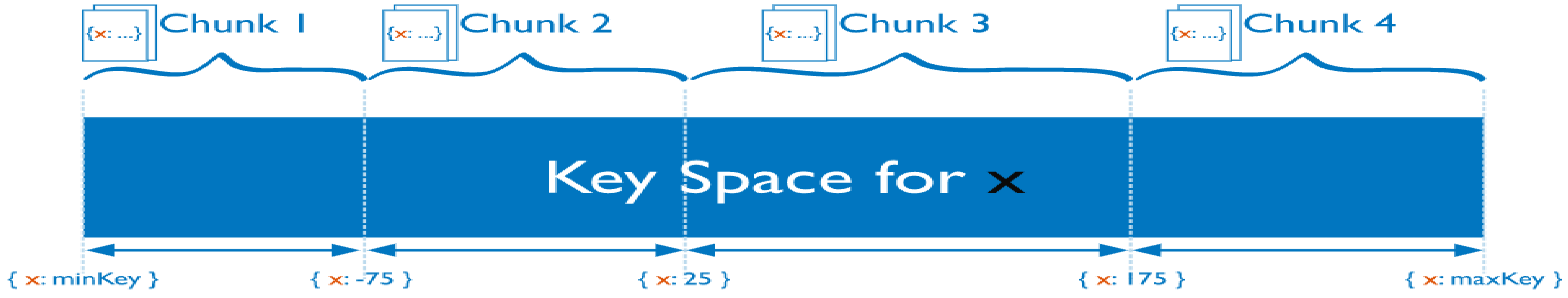
- Check sharding status:

```
sh.status()
```

Mongod – Sharding

Shard Keys:

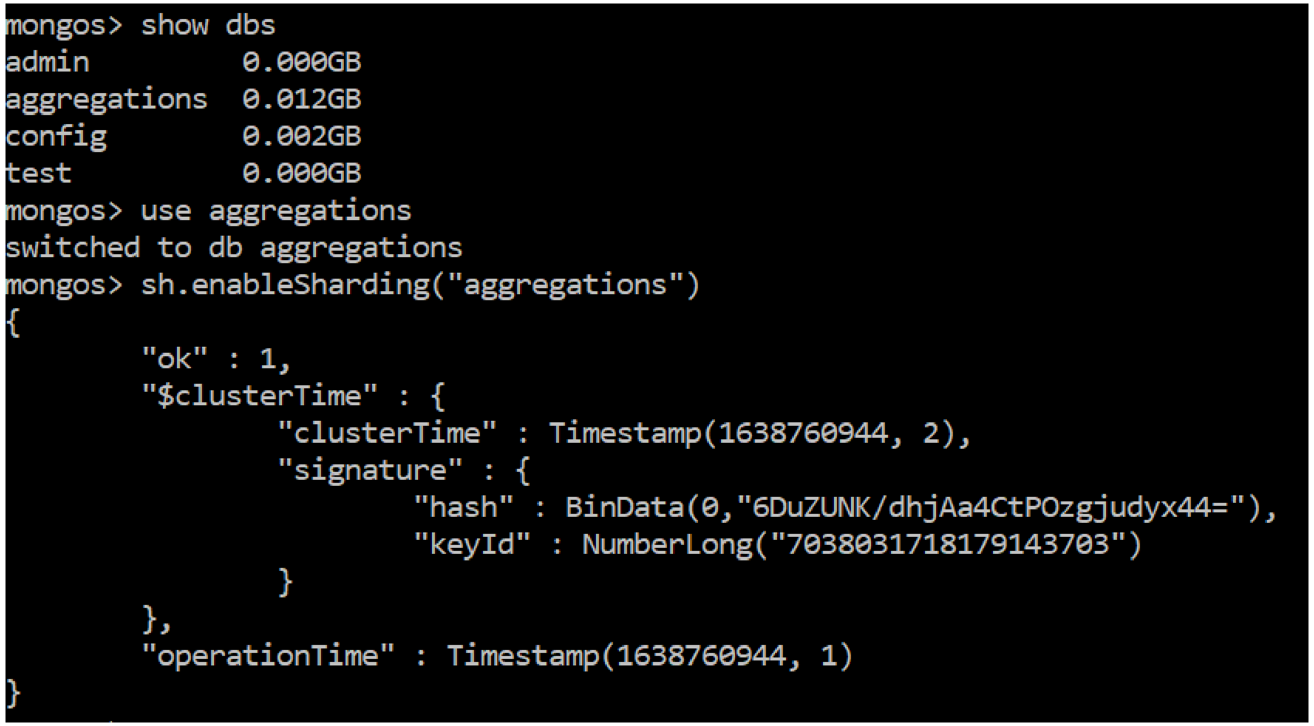
- The indexes field or fields
- MongoDB divides the span of shard key values into non-overlapping ranges of shard key values
- Each range is associated with a chunk
- MongoDB attempts to distribute chunks evenly among the shards in the cluster
- Cannot unshard a collection



Mongod – Sharding

Shard Keys: How to shard

- Use sh.enableSharding('<database>') to enable sharding for the specified database
- Use db.collection.createIndex() to create index for shard key
- Use sh.shardCollection('<database>', '<collection>',{shard key}) to shard collection

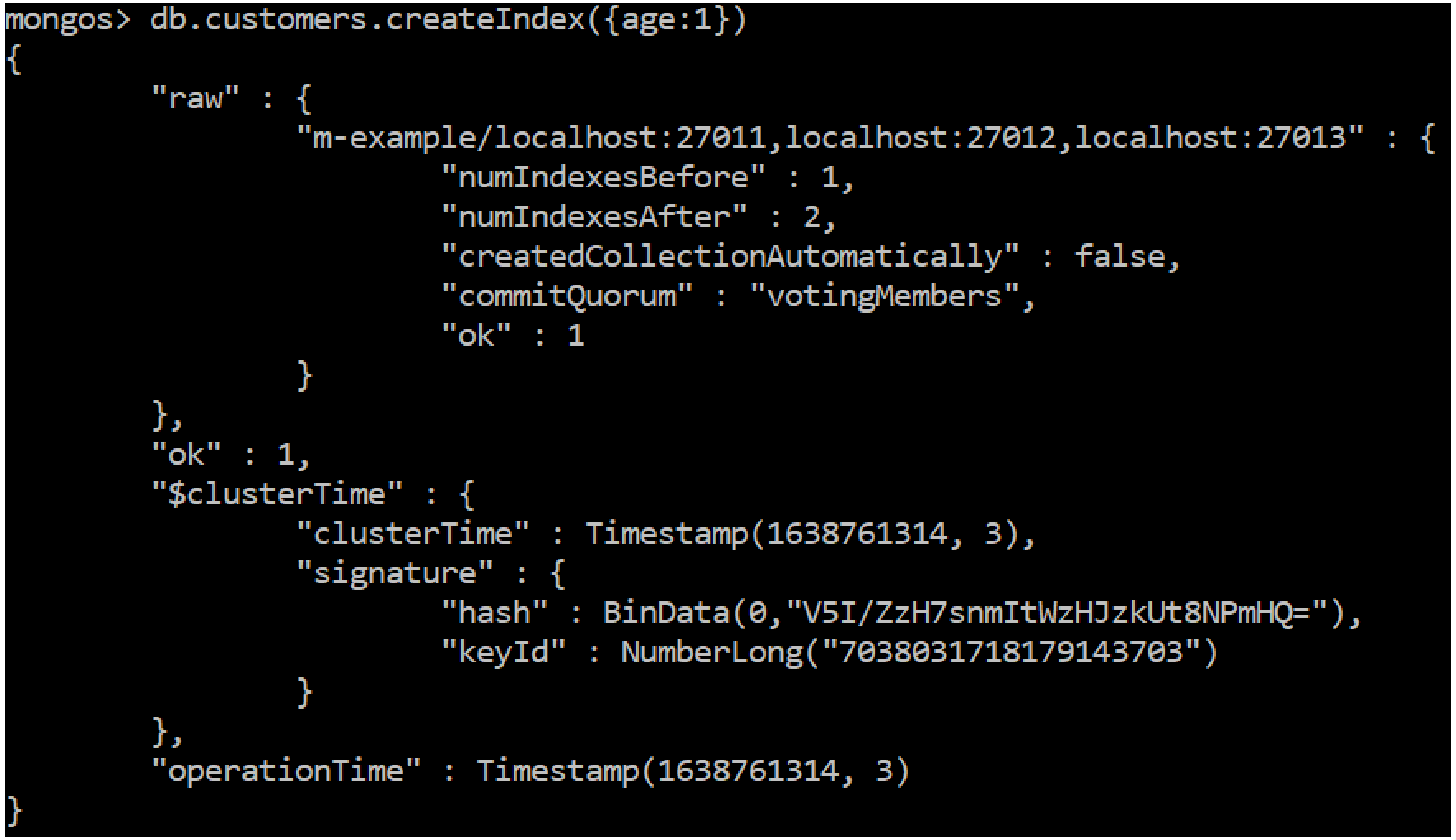


```
mongos> show dbs
admin          0.000GB
aggregations   0.012GB
config         0.002GB
test           0.000GB
mongos> use aggregations
switched to db aggregations
mongos> sh.enableSharding("aggregations")
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1638760944, 2),
                "signature" : {
                        "hash" : BinData(0,"6DuZUNK/dhjAa4CtPOzgjudyx44="),
                        "keyId" : NumberLong("7038031718179143703")
                }
        },
        "operationTime" : Timestamp(1638760944, 1)
}
```

Mongod – Sharding

Shard Keys: How to shard

- Use db.collection.createIndex() to create index for shard key

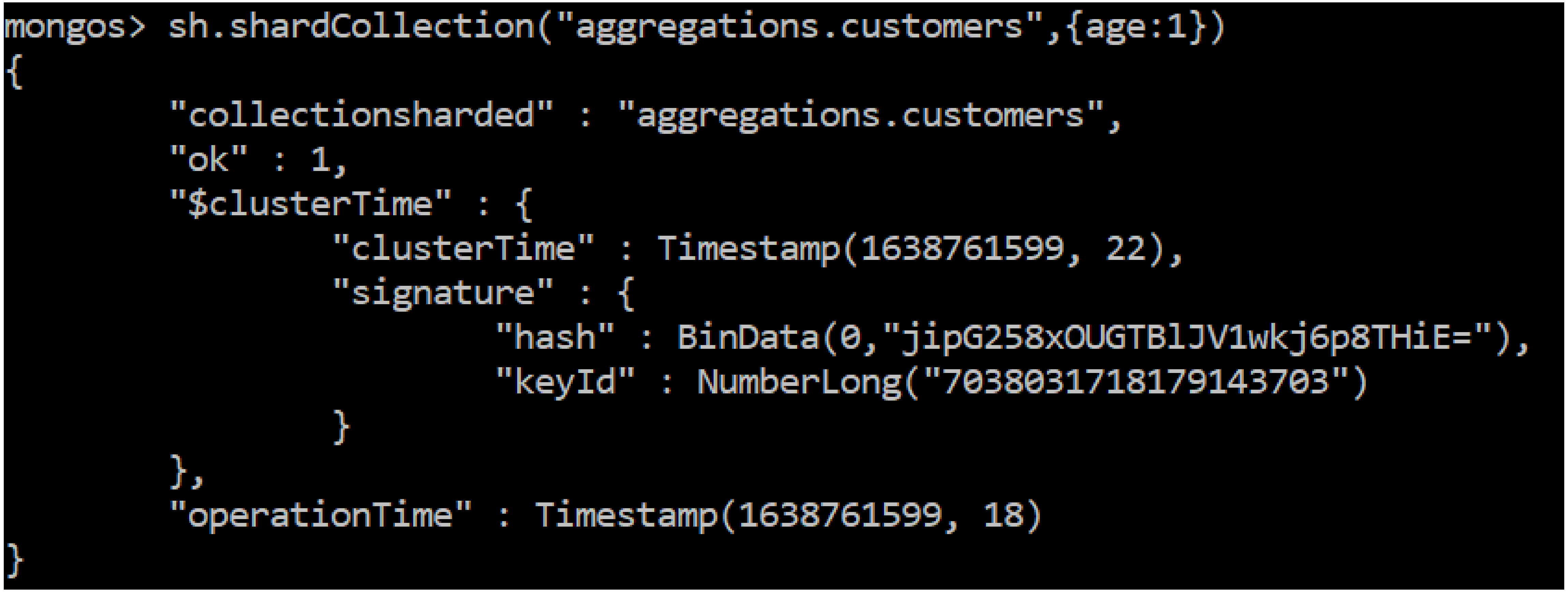


```
mongos> db.customers.createIndex({age:1})
{
        "raw" : {
                "m-example/localhost:27011,localhost:27012,localhost:27013" : {
                        "numIndexesBefore" : 1,
                        "numIndexesAfter" : 2,
                        "createdCollectionAutomatically" : false,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                }
        },
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1638761314, 3),
                "signature" : {
                        "hash" : BinData(0,"V5I/ZzH7snmItWzHJzkUt8NPmHQ="),
                        "keyId" : NumberLong("7038031718179143703")
                }
        },
        "operationTime" : Timestamp(1638761314, 3)
}
```

Mongod – Sharding

Shard Keys: How to shard

- Use sh.shardCollection('<database>', '<collection>',{shard key}) to shard collection



```
mongos> sh.shardCollection("aggregations.customers",{age:1})
{
        "collectionsharded" : "aggregations.customers",
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1638761599, 22),
                "signature" : {
                        "hash" : BinData(0,"jipG258xOUGTBlJV1wkj6p8THiE="),
                        "keyId" : NumberLong("7038031718179143703")
                }
        },
        "operationTime" : Timestamp(1638761599, 18)
}
```

Mongod – Sharding

Shard Keys: Picking a Good Key

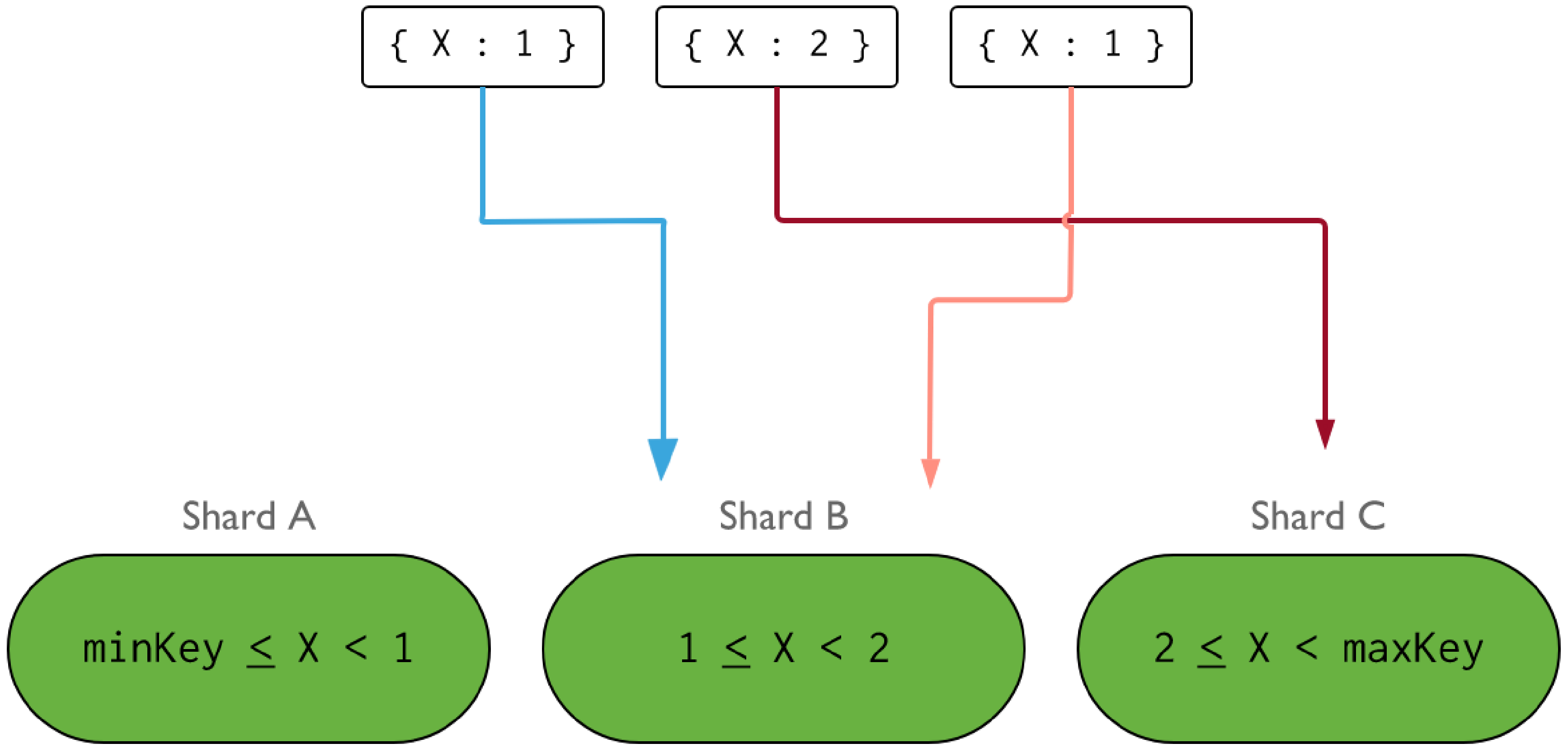
When you choose your shard key, consider:

- The **cardinality** of the shard key
- The **frequency** with which shard key values occur
- Whether a potential shard key grows **monotonically**
- Sharding **Query Patterns**
- Shard Key Limitations (Starting in version 4.4, MongoDB removes the limit on the shard key size)

Mongod – Sharding

Shard Keys: Picking a Good Key - Cardinality:

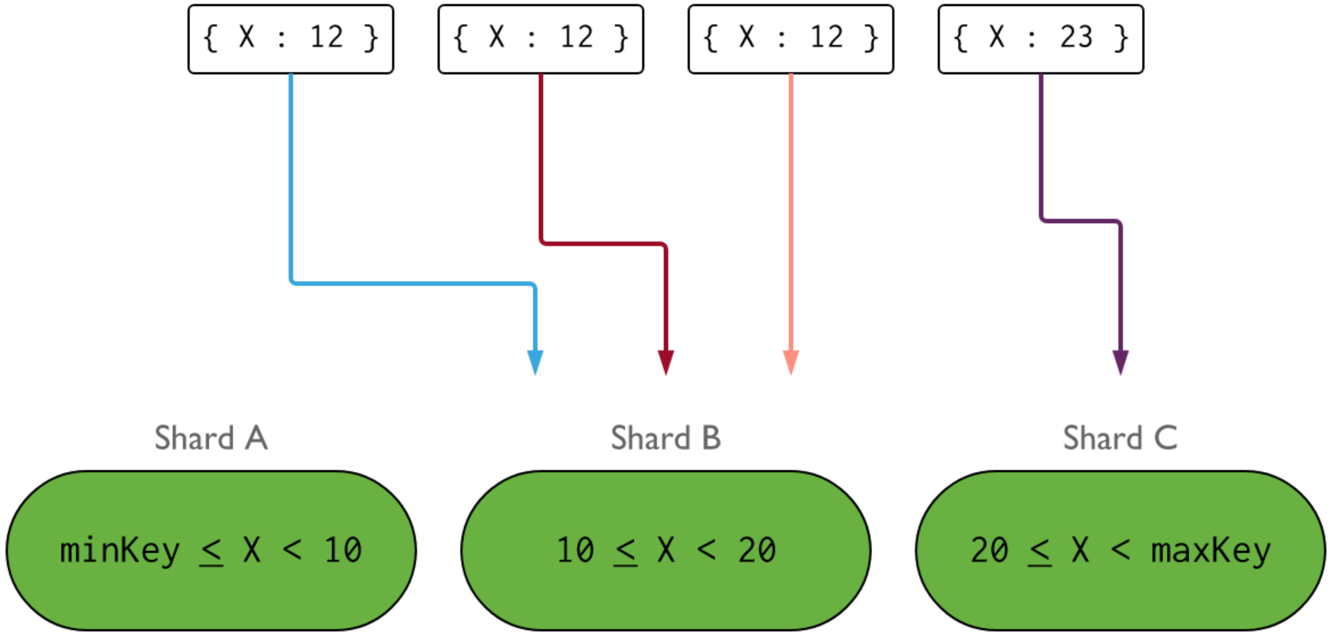
- Choose a shard key with high cardinality (many possible unique values)
- low cardinality reduces the effectiveness of horizontal scaling in the cluster



Mongod – Sharding

Shard Keys: Picking a Good Key - Frequency:

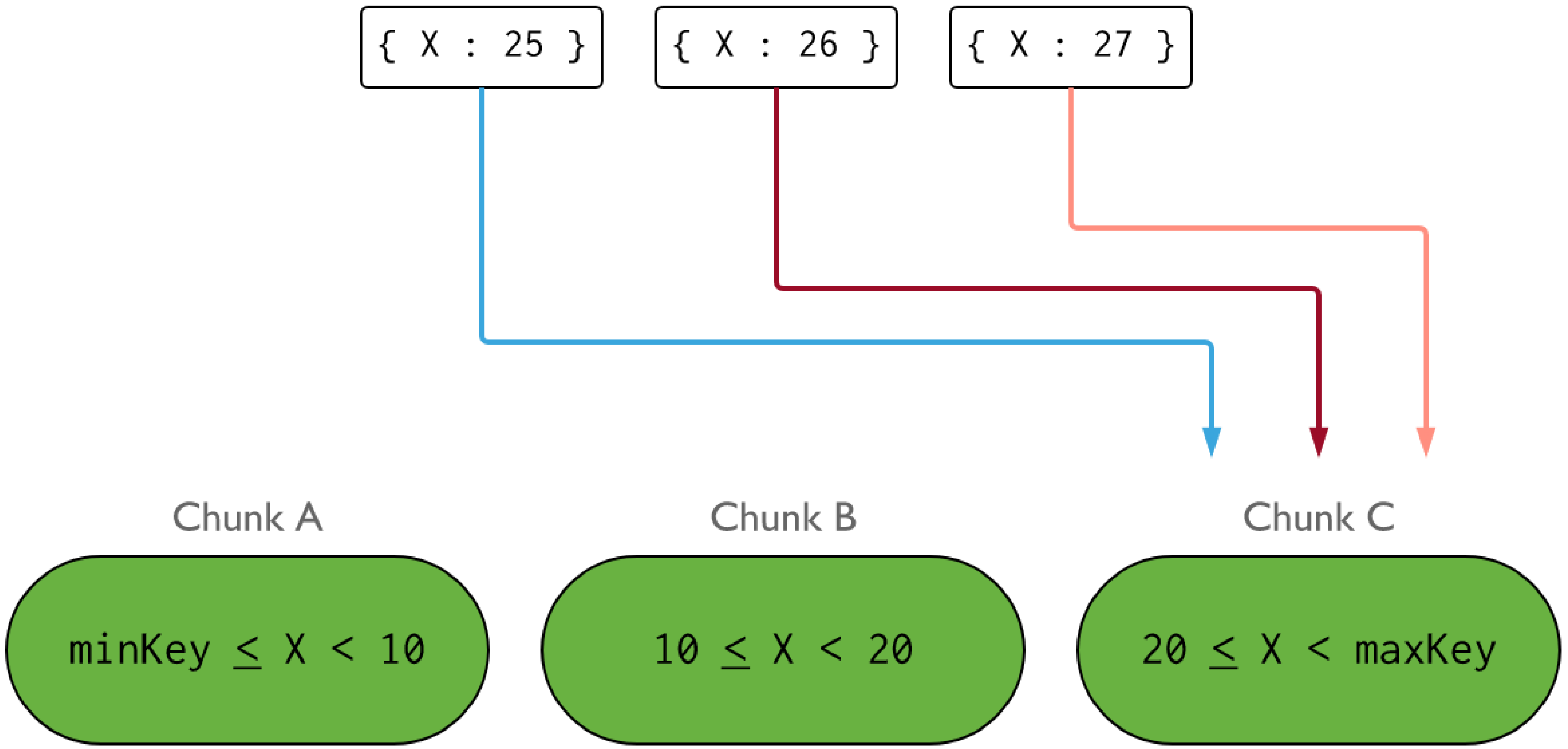
- **frequency** of the shard key represents how often a given shard key value occurs in the data
- If the majority of documents contain only a subset of the possible shard key values, then the chunks storing the documents with those values can become a bottleneck within the cluster



Mongod – Sharding

Shard Keys: Picking a Good Key - Monotonically Changing:

- A shard key on a value that increases or decreases monotonically is more likely to distribute inserts to a single chunk within the cluster



Mongod – Sharding

Shard Keys: Picking a Good Key - Sharding Query Patterns:

- When you choose a shard key, consider your most common query patterns and whether a given shard key covers them.

- When the queries do not contain the shard key, the queries are broadcast to all shards for evaluation → inefficient

Question?