Large-Scale and Multi-Structured Databases

MongoDB Clustering

Prof Pietro Ducange

Ing. Alessio Schiavo

alessio.schiavo@phd.unipi.it

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

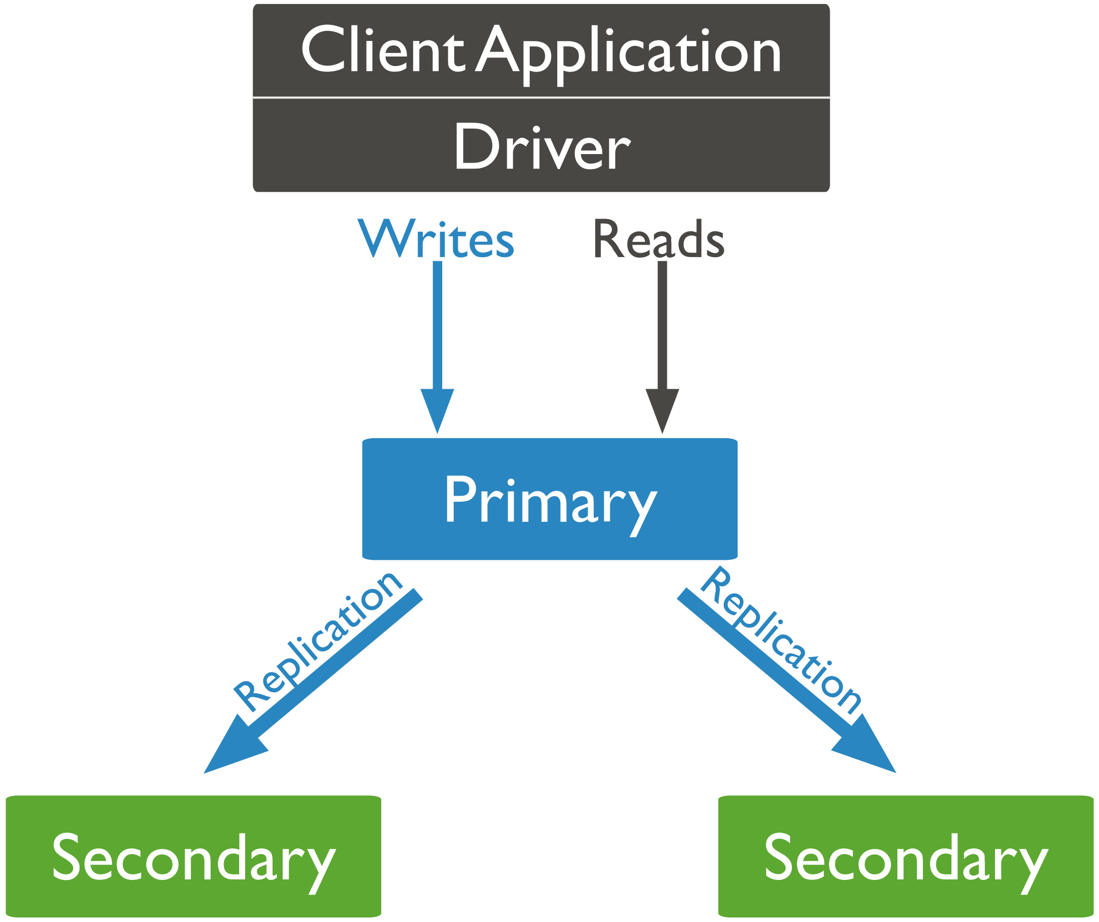
CROSSLAB Innovation for industry 4.0

Copyright Issues

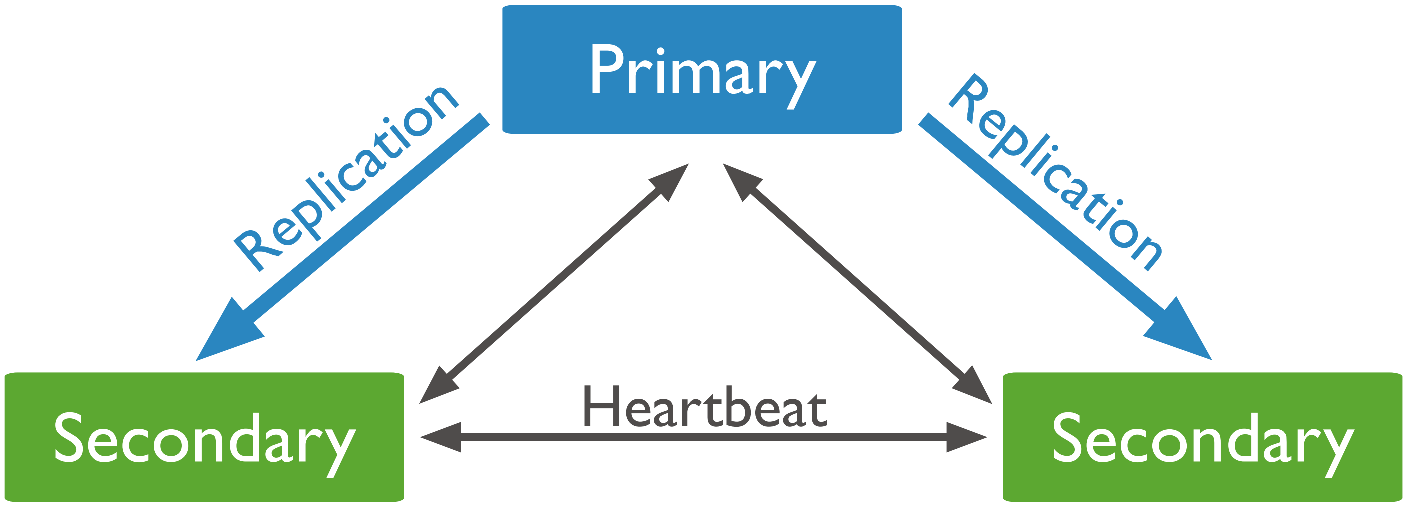
Most of the information included this presentation have been extracted from the official documentation of MongoDB Java Driver (http://mongodb.github.io/mongo-java-driver/).

Local **Replica Set Configuration & Deployment Tutorial**: https://www.mongodb.com/docs/manual/tutorial/deploy-replica-set-for-testing/

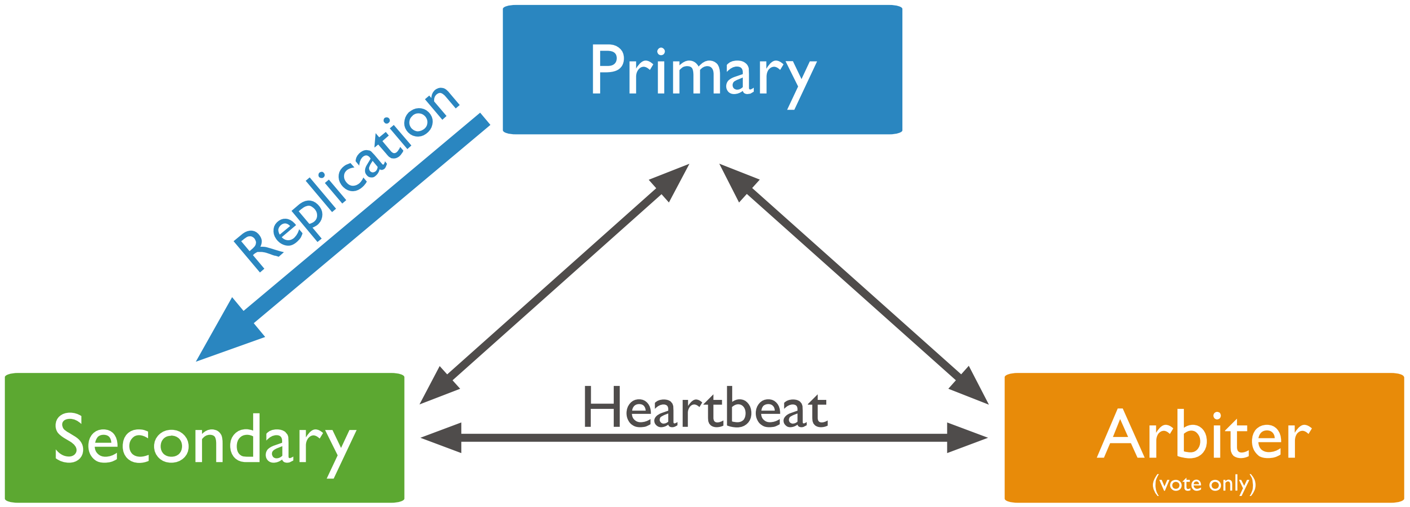
MongoDB Replication: quick intro (1)

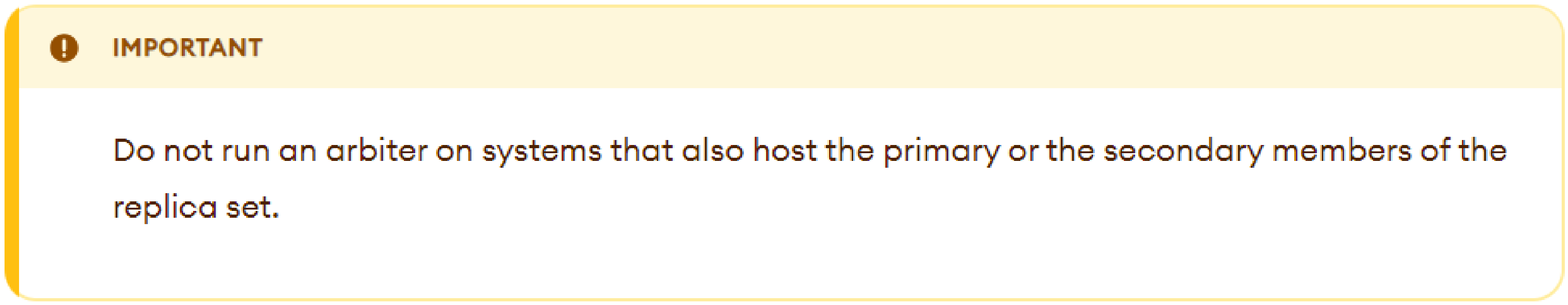


MongoDB Replication: quick intro (2)



MongoDB Replication: quick intro (3)

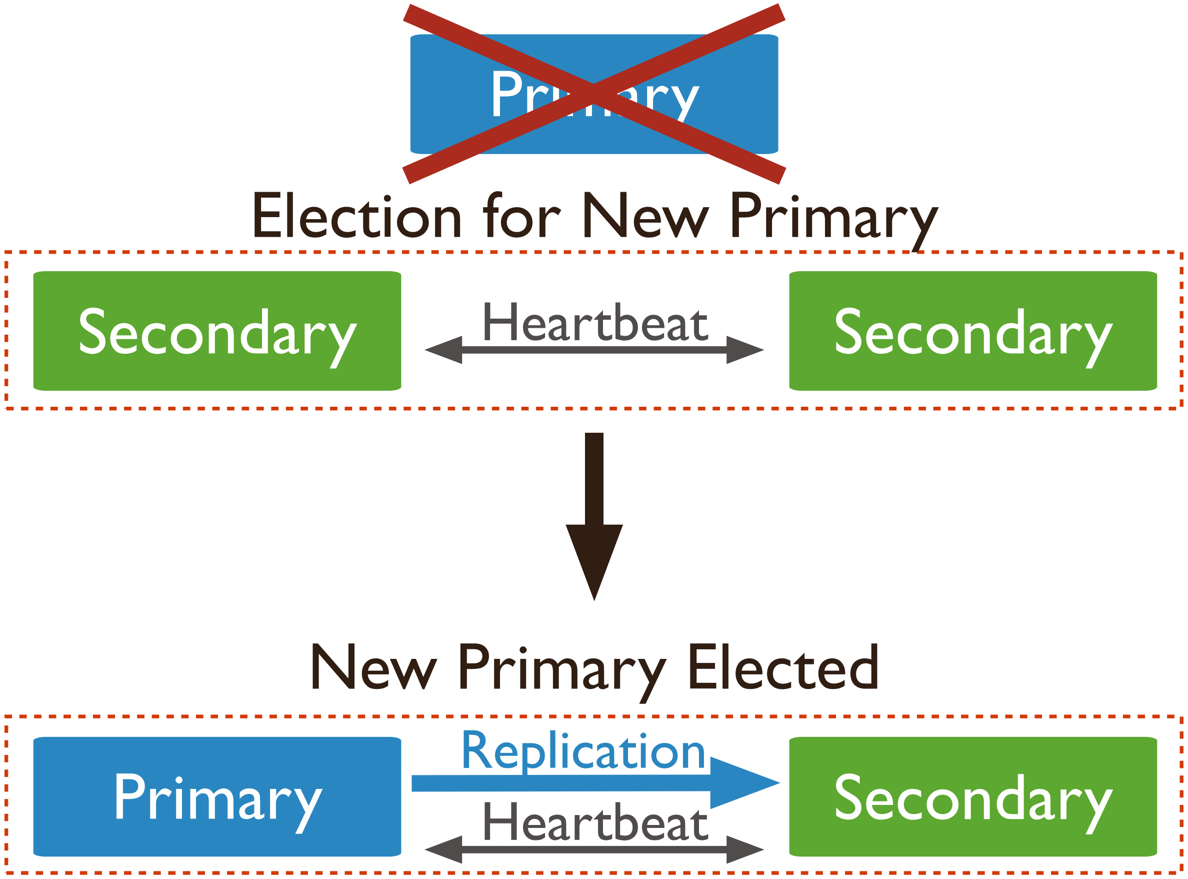



IMPORTANT

Do not run an arbiter on systems that also host the primary or the secondary members of the replica set.

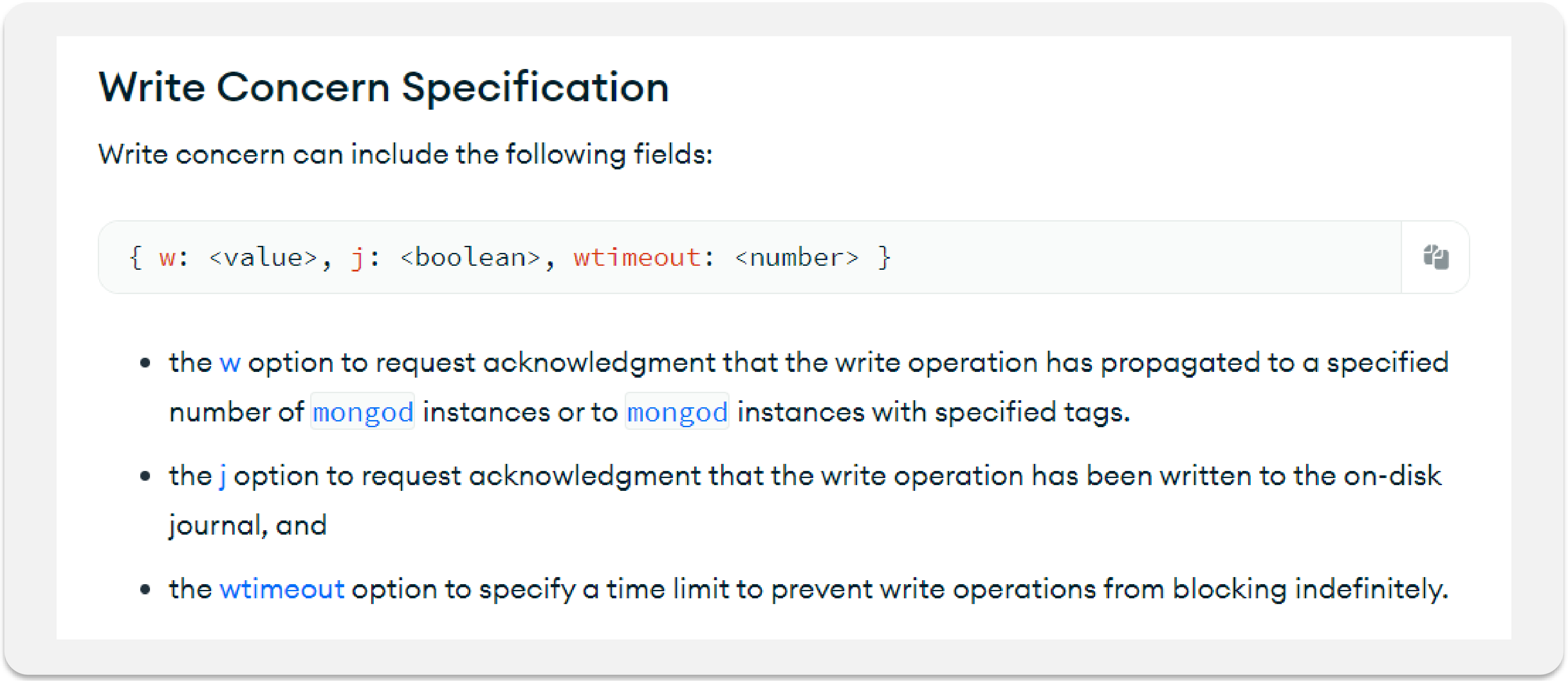
https://www.mongodb.com/docs/manual/tutorial/add-replica-set-arbiter/

MongoDB Replication: automatic failover



https://www.mongodb.com/docs/manual/core/replica-set-elections/

Write Concern (1)



Write Concern Specification

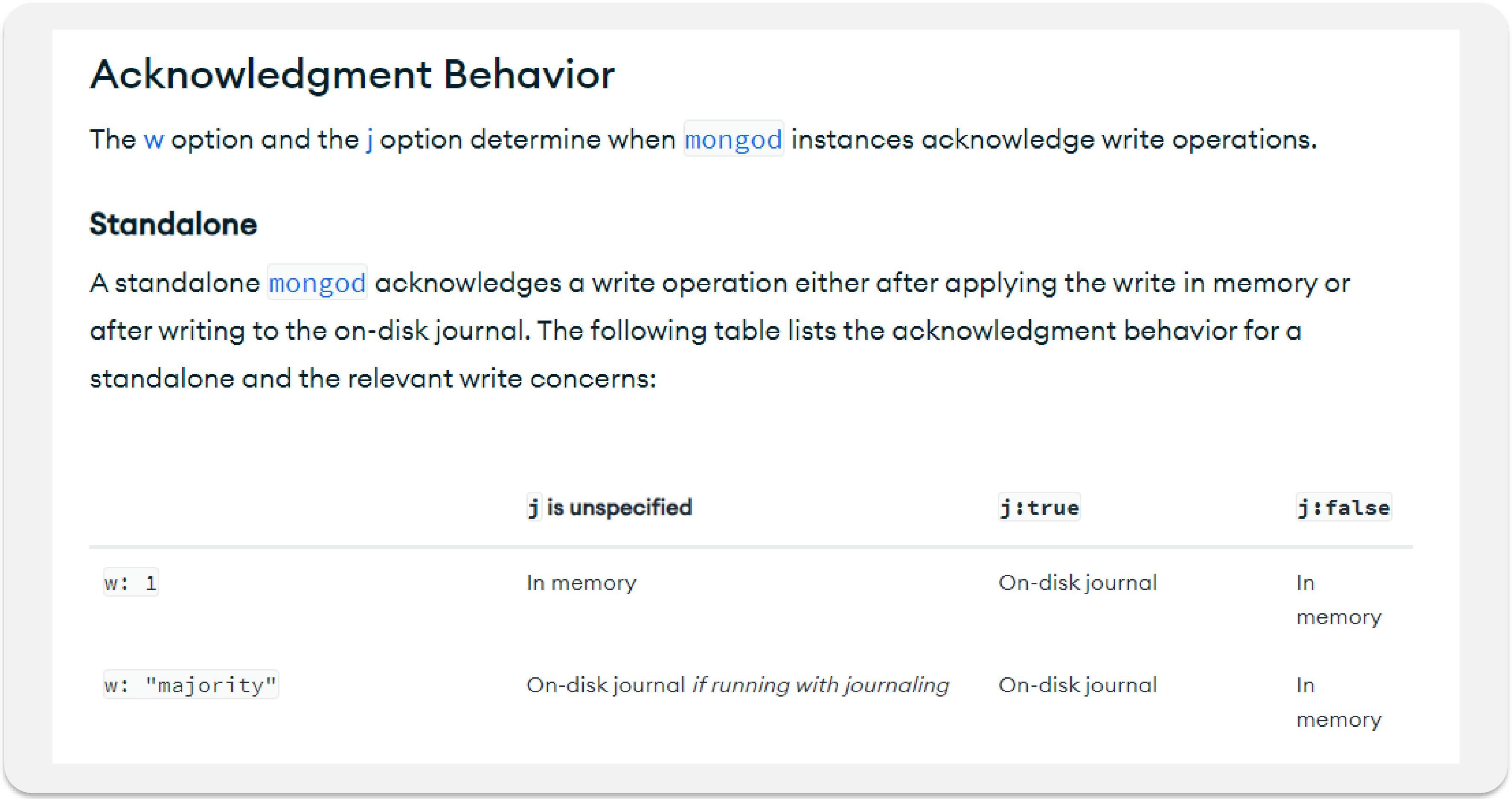
Write concern can include the following fields:

```
{ w: <value>, j: <boolean>, wtimeout: <number> }
```

- the w option to request acknowledgment that the write operation has propagated to a specified number of `mongod` instances or to `mongod` instances with specified tags.
- the j option to request acknowledgment that the write operation has been written to the on-disk journal, and
- the wtimeout option to specify a time limit to prevent write operations from blocking indefinitely.

https://www.mongodb.com/docs/manual/reference/write-concern/

Write Concern (2)



Acknowledgment Behavior

The w option and the j option determine when `mongod` instances acknowledge write operations.

Standalone

A standalone `mongod` acknowledges a write operation either after applying the write in memory or after writing to the on-disk journal. The following table lists the acknowledgment behavior for a standalone and the relevant write concerns:

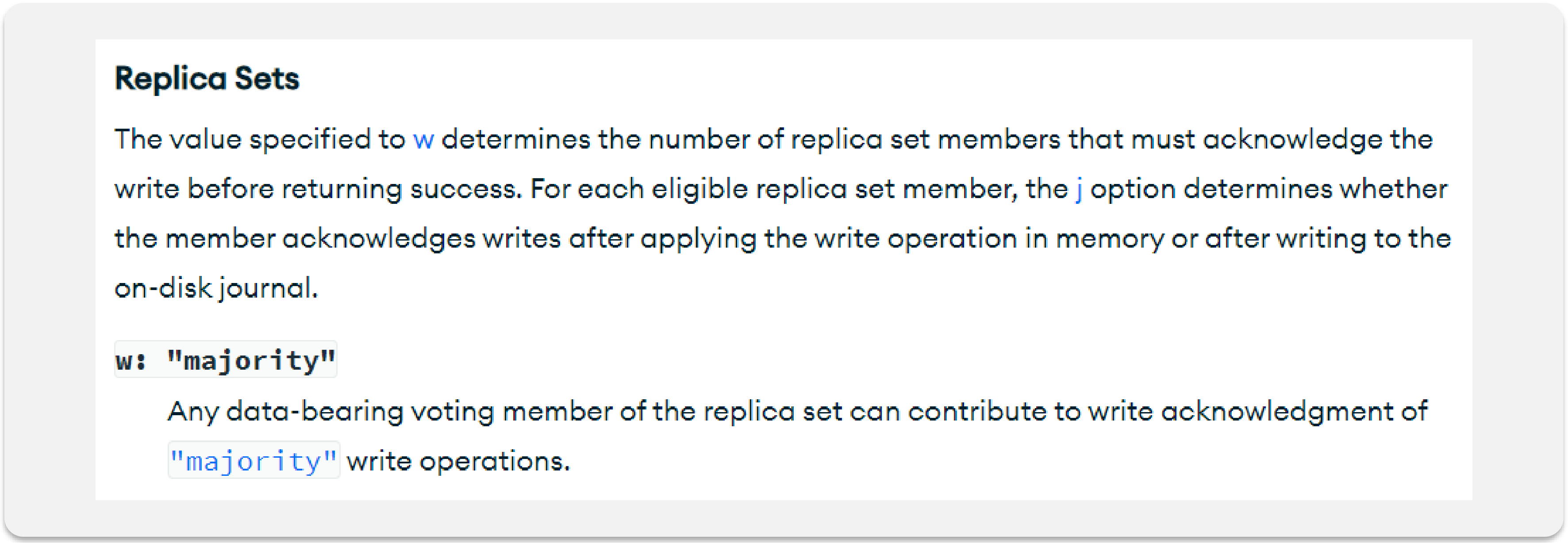
	j is unspecified	`j:true`	`j:false`
`w: 1`	In memory	On-disk journal	In memory
`w: "majority"`	On-disk journal *if running with journaling*	On-disk journal	In memory

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Write Concern (3)



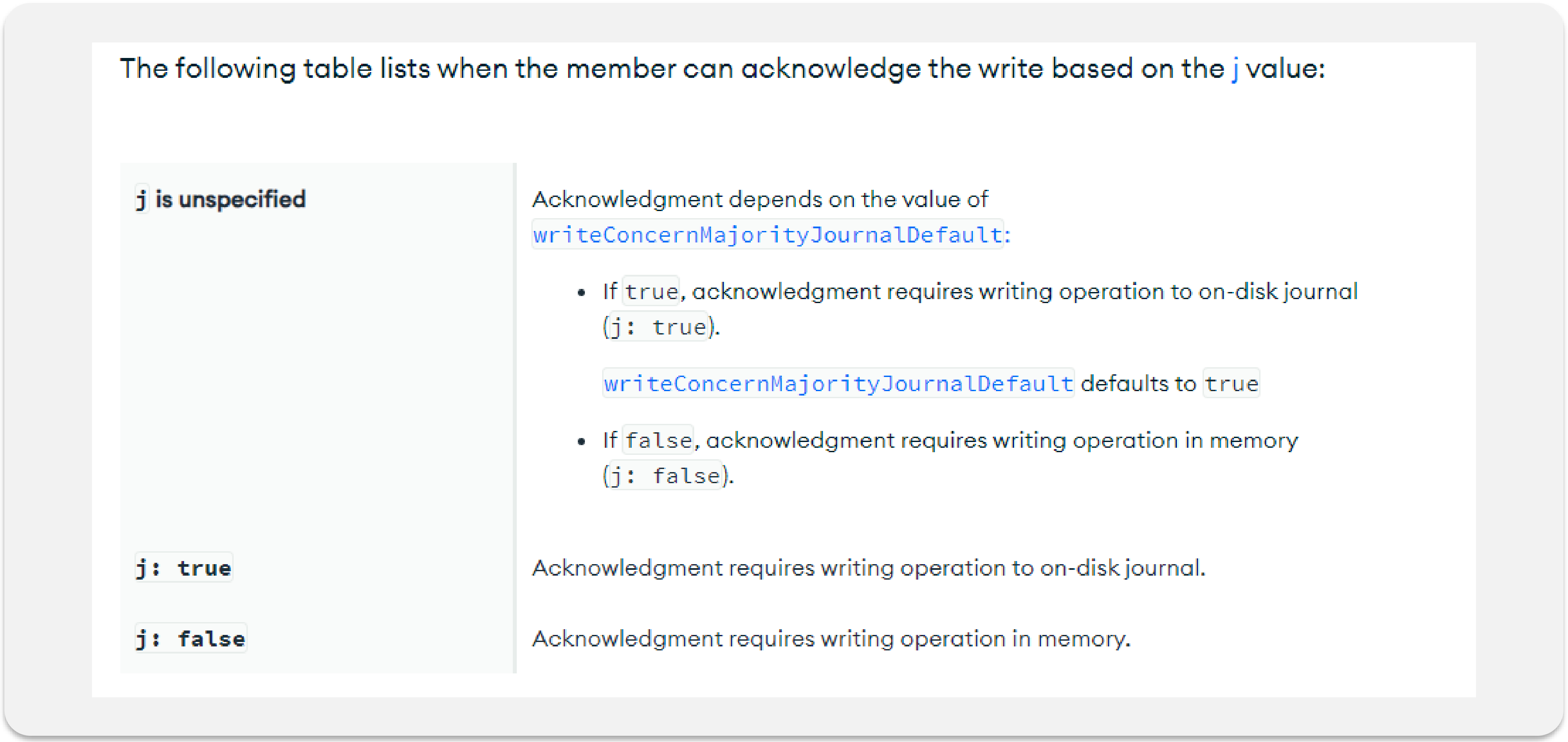
Replica Sets

The value specified to w determines the number of replica set members that must acknowledge the write before returning success. For each eligible replica set member, the j option determines whether the member acknowledges writes after applying the write operation in memory or after writing to the on-disk journal.

`w: "majority"`

Any data-bearing voting member of the replica set can contribute to write acknowledgment of `"majority"` write operations.

Write Concern (4)



The following table lists when the member can acknowledge the write based on the `j` value:

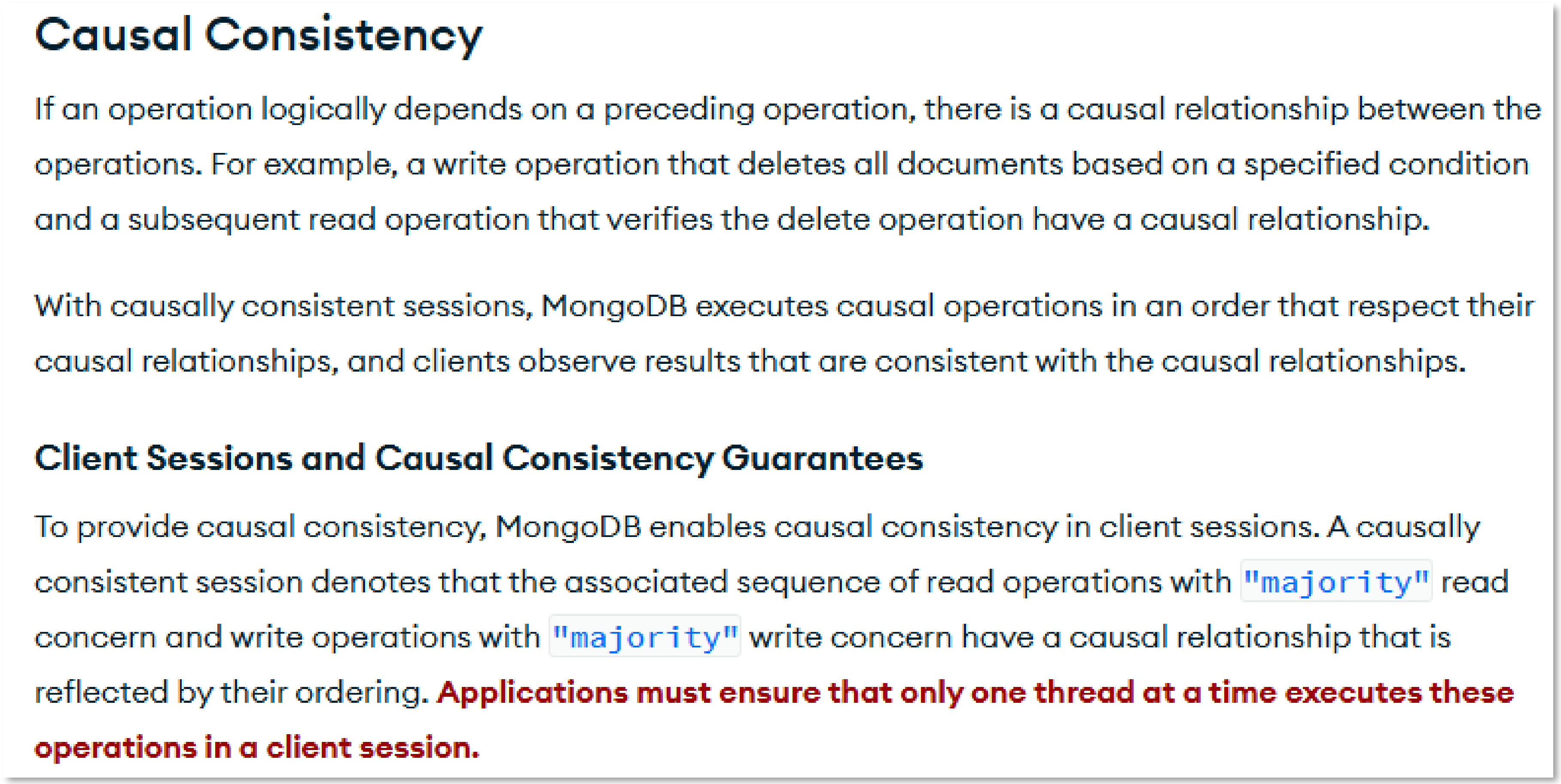
j is unspecified	Acknowledgment depends on the value of `writeConcernMajorityJournalDefault`: • If `true`, acknowledgment requires writing operation to on-disk journal (`j: true`). `writeConcernMajorityJournalDefault` defaults to `true` • If `false`, acknowledgment requires writing operation in memory (`j: false`).
`j: true`	Acknowledgment requires writing operation to on-disk journal.
`j: false`	Acknowledgment requires writing operation in memory.

Write Concern (5)

Write Concern Type	Behavior
ACKNOWLEDGED	Use the default Write Concern from the server
JOURNALED	Wait for the server to group commit to the journal file on disk.
MAJORITY	Wait on a majority of servers for the write operation.
UNACKNOWLEDGED	Return as soon as the message is written to the socket.
W1	Wait for acknowledgement from a single member.
W2	Wait for acknowledgement from two members
W3	Write operations that use this write concern will wait for acknowledgement from three members

The implicit **default Write Concern is w: majority**.

Causal Consistency in Mongo



Causal Consistency

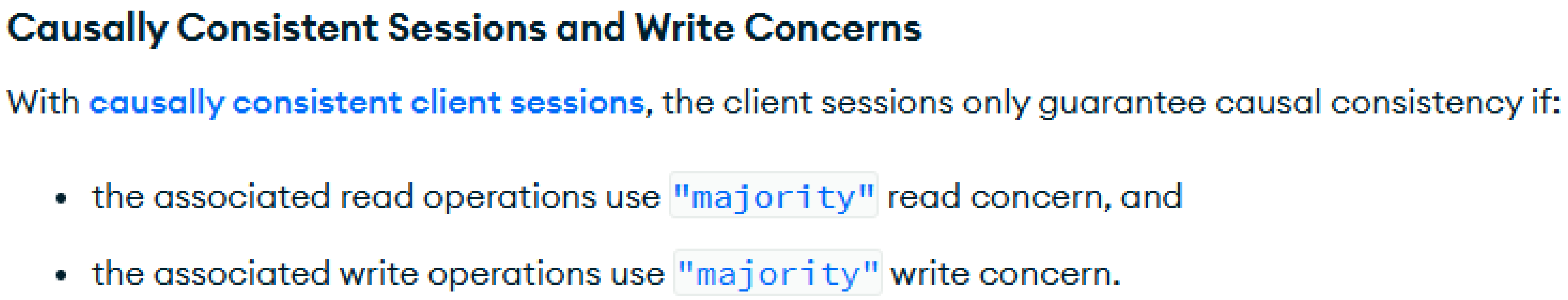
If an operation logically depends on a preceding operation, there is a causal relationship between the operations. For example, a write operation that deletes all documents based on a specified condition and a subsequent read operation that verifies the delete operation have a causal relationship.

With causally consistent sessions, MongoDB executes causal operations in an order that respect their causal relationships, and clients observe results that are consistent with the causal relationships.

Client Sessions and Causal Consistency Guarantees

To provide causal consistency, MongoDB enables causal consistency in client sessions. A causally consistent session denotes that the associated sequence of read operations with `"majority"` read concern and write operations with `"majority"` write concern have a causal relationship that is reflected by their ordering. **Applications must ensure that only one thread at a time executes these operations in a client session.**

Causal Consistency in Mongo

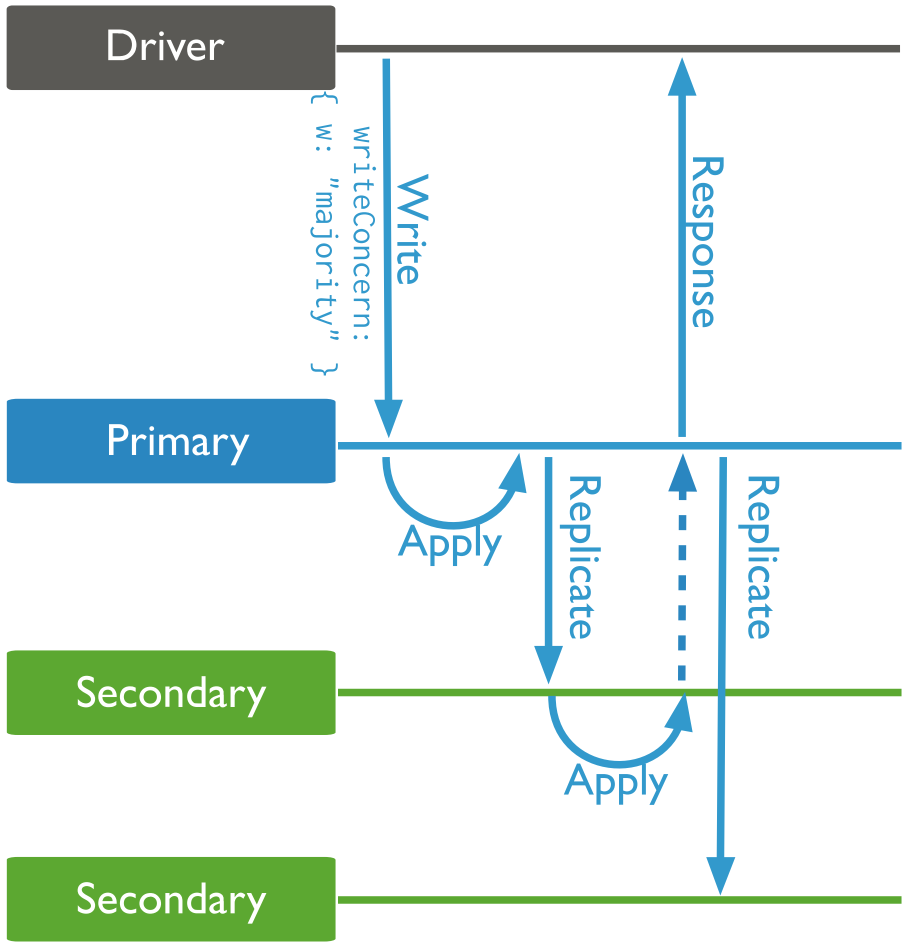


Causally Consistent Sessions and Write Concerns

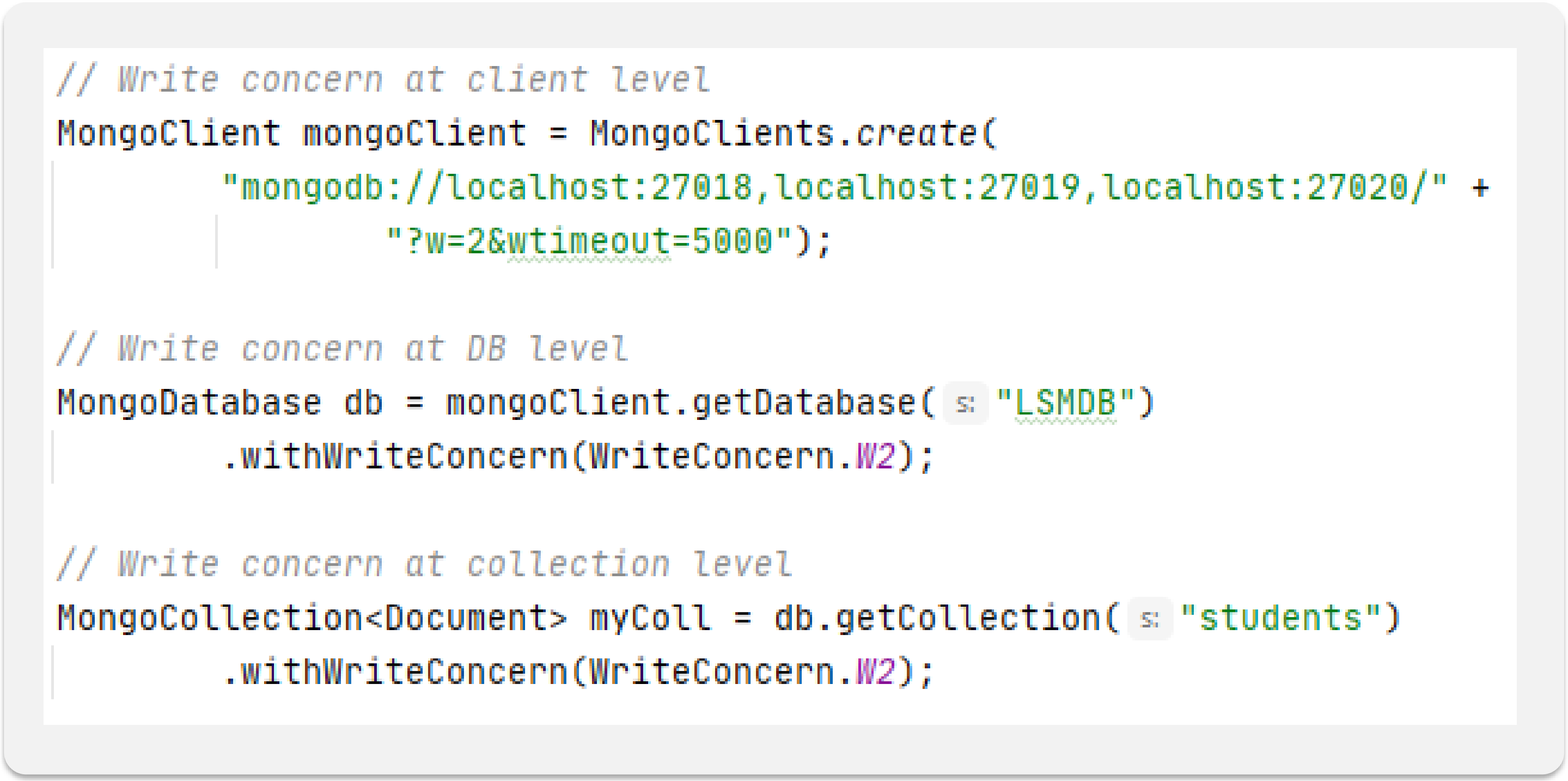
With **causally consistent client sessions**, the client sessions only guarantee causal consistency if:

- the associated read operations use `"majority"` read concern, and
- the associated write operations use `"majority"` write concern.

Write Concern in Replica Sets: acknowledgment behavior



Write Concern is organized into hierarchical levels

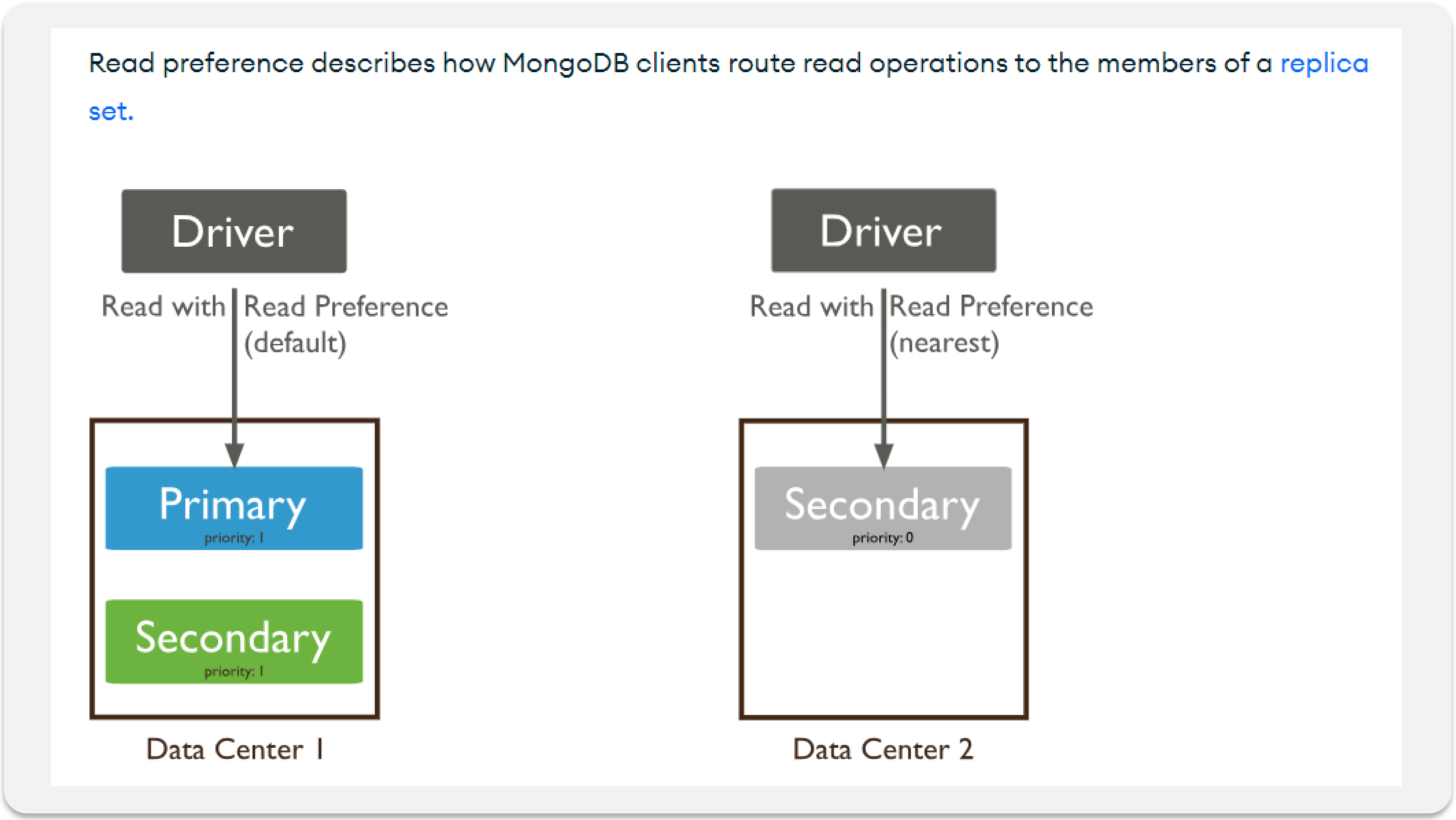


```java
// Write concern at client level
MongoClient mongoClient = MongoClients.create(
        "mongodb://localhost:27018,localhost:27019,localhost:27020/" +
                "?w=2&wtimeout=5000");

// Write concern at DB level
MongoDatabase db = mongoClient.getDatabase( s: "LSMDB")
        .withWriteConcern(WriteConcern.W2);

// Write concern at collection level
MongoCollection<Document> myColl = db.getCollection( s: "students")
        .withWriteConcern(WriteConcern.W2);
```

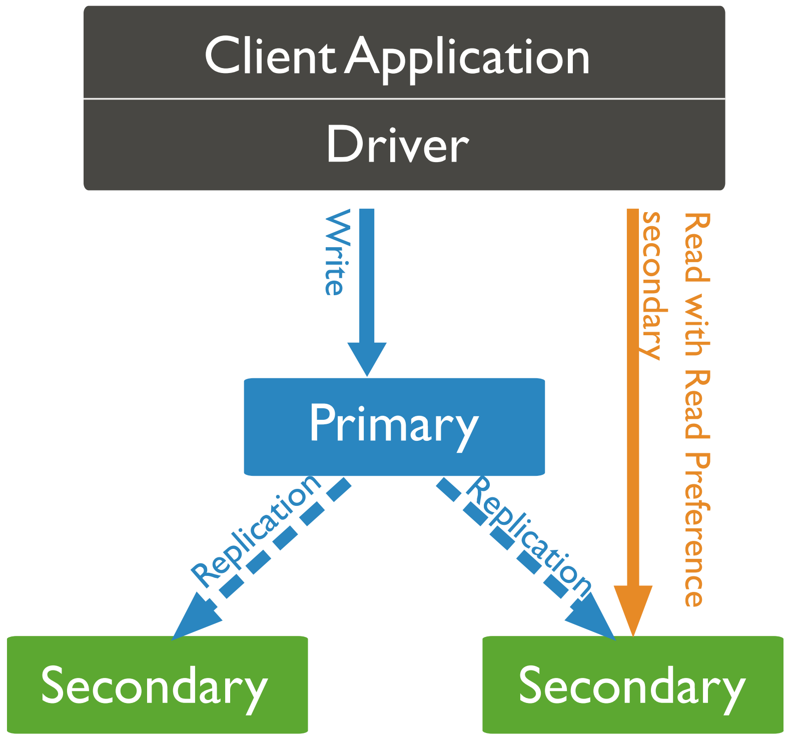
Read Preference (1)


Read preference describes how MongoDB clients route read operations to the members of a replica set.

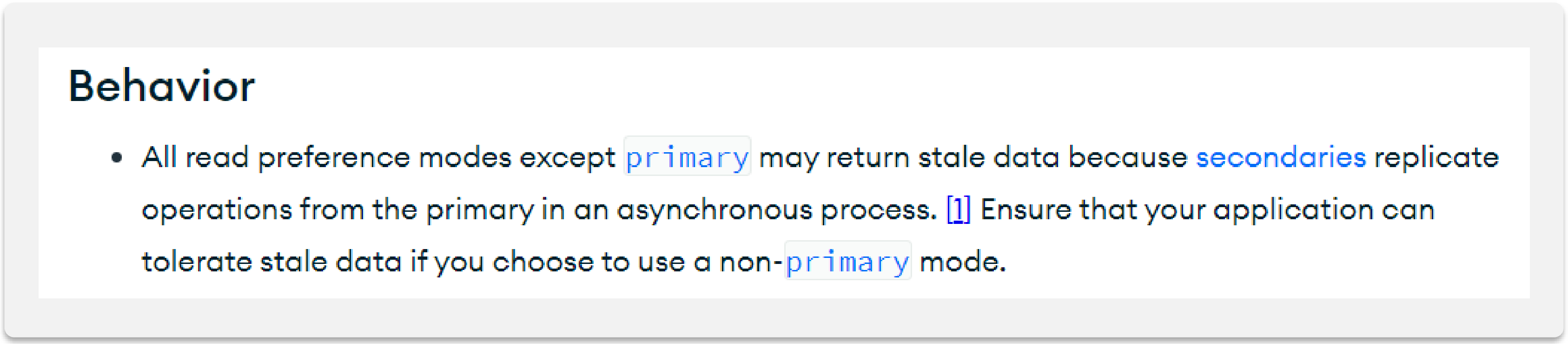
https://www.mongodb.com/docs/manual/core/read-preference/

Read Preferences (2)

- A **read preference** provides client applications with control over which nodes of a replica set are used for reads



Read Preference (3)



Behavior

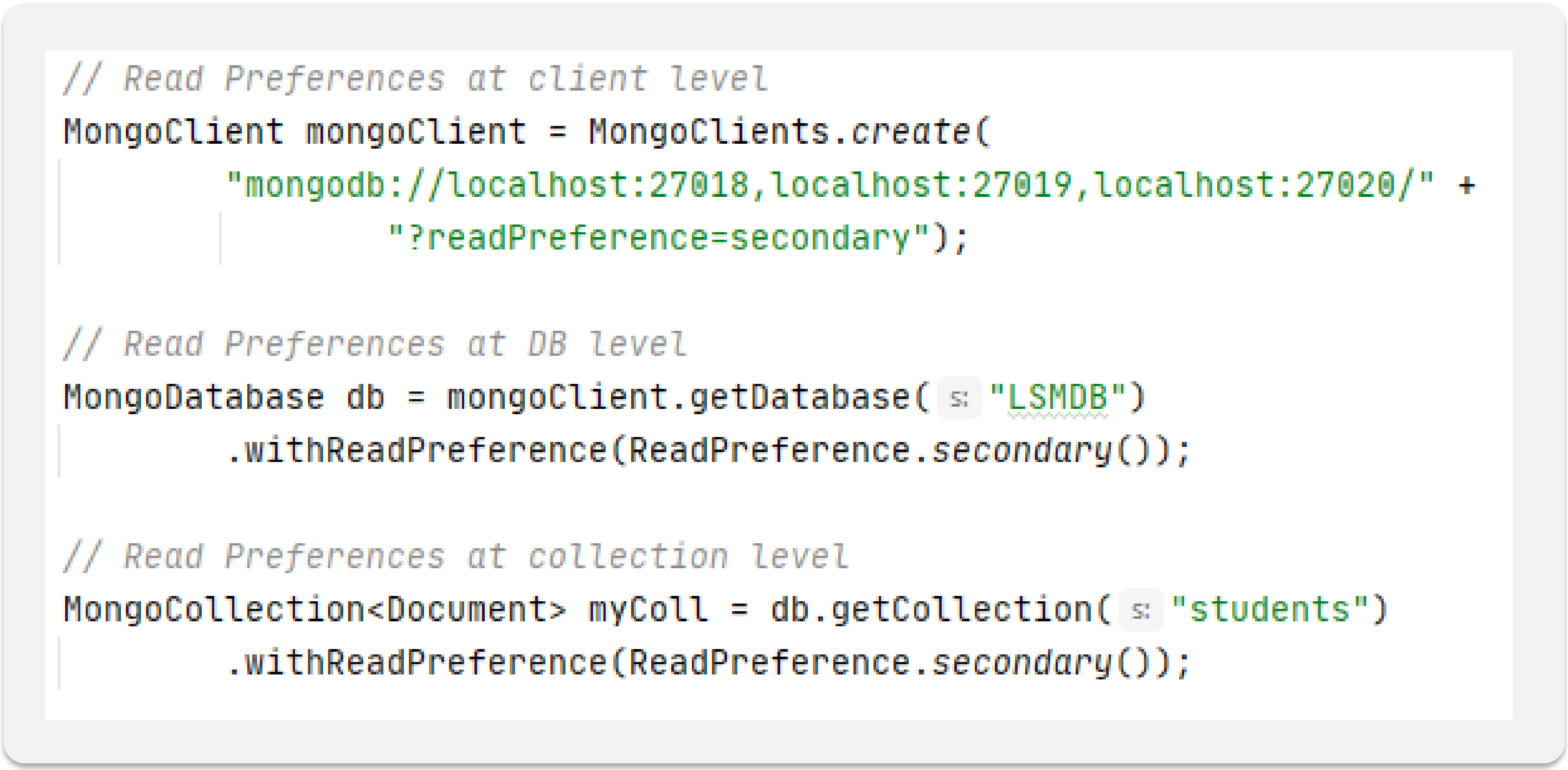
- All read preference modes except `primary` may return stale data because secondaries replicate operations from the primary in an asynchronous process. [1] Ensure that your application can tolerate stale data if you choose to use a non-`primary` mode.

Read Preference Modes

- A client application defines its read preference by selecting one of the five behavioral modes:

Read Preferences	Behavior
PRIMARY	The **default read mode. Read from primary only**
PRIMARY PREFERRED	Read from primary if available, otherwise a secondary
SECONDARY	Read from a secondary node if available, otherwise error.
SECONDARY PREFERRED	Read from a secondary if available, otherwise read from the primary.
NEAREST	Operations read from a random eligible replica set member, irrespective of whether that member is a primary or secondary, based on a specified latency threshold.

Read Preferences (4)



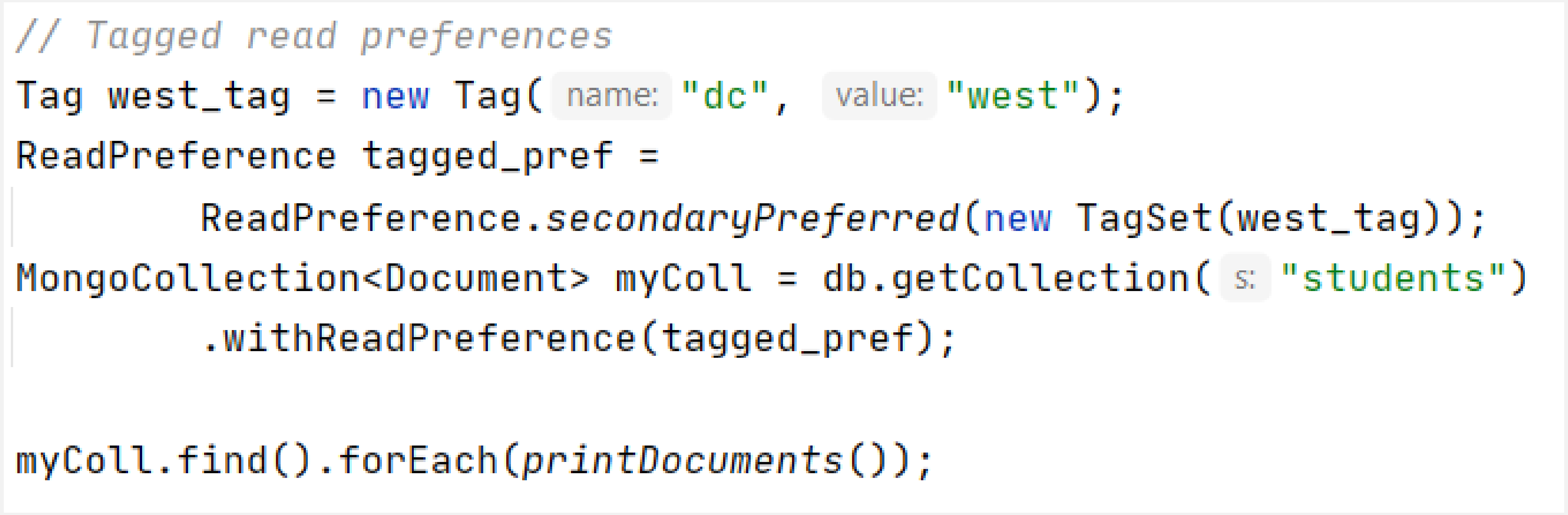
```
// Read Preferences at client level
MongoClient mongoClient = MongoClients.create(
        "mongodb://localhost:27018,localhost:27019,localhost:27020/" +
                "?readPreference=secondary");

// Read Preferences at DB level
MongoDatabase db = mongoClient.getDatabase( s: "LSMDB")
        .withReadPreference(ReadPreference.secondary());

// Read Preferences at collection level
MongoCollection<Document> myColl = db.getCollection( s: "students")
        .withReadPreference(ReadPreference.secondary());
```

Read Preferences (3)

We can use the tags configured into our replicas to state preferences on **secondary replicas** (cannot specify a tag preference on a primary)



```
// Tagged read preferences
Tag west_tag = new Tag( name: "dc",  value: "west");
ReadPreference tagged_pref =
        ReadPreference.secondaryPreferred(new TagSet(west_tag));
MongoCollection<Document> myColl = db.getCollection( s: "students")
        .withReadPreference(tagged_pref);

myColl.find().forEach(printDocuments());
```

Configure a local Replica Set

1. Create Data directories:

```
# Ubuntu
mkdir ~/data/r{1,2,3}


# Windows (PowerShell)
mkdir C:\MongoDB\data\r1
mkdir C:\MongoDB\data\r2
mkdir C:\MongoDB\data\r3
```

Start MongoDB servers (1)

2. Open 3 separate terminals and start 3 MongoDB servers on replica set mode (--replSet option)

```
# Windows (For Ubuntu it's the same)
mongod --replSet rs0 --dbpath c:\MongoDB\data\r1
--port 27018 --bind_ip localhost --oplogSize 200

mongod --replSet rs0 --dbpath c:\MongoDB\data\r1
--port 27019 --bind_ip localhost --oplogSize 200

mongod --replSet rs0 --dbpath c:\MongoDB\data\r1
--port 27020 --bind_ip localhost --oplogSize 200
```

Connect to one node of the Replica Set

3. **Open a 4th terminal** and connect to one of your mongod instances through mongosh:

```
# Windows (For Ubuntu it’s the same)
mongosh --port 27017
```

Configure the Replica set

To fully configure the replica set, follow these steps:

4. Define the following configuration (copy-paste the following within the mongo shell):

```
rsconf = {
    _id: “rs0”,
    members: [
             {_id: 0, host: “localhost:27018”},
             {_id: 1, host: “localhost:27019”},
             {_id: 2, host: “localhost:27020”}]};
```

Configure the Replica set (2)

5. Initiate the replica set (only ONCE)

```
rs.initiate(rsconf);
```

6. Check the status:

```
rs.status();
```

7. Change configuration :

```
rs.reconfig(rsconfig);
```

https://www.mongodb.com/docs/manual/reference/replica-configuration/#mongodb-rsconf-rsconf.settings

Additional configurations (1)

- Set different priorities to replica set members in order to change the default behavior during the primary election

```
{_id: 0, host: "localhost:27018", priority: 0.5}
{_id: 1, host: "localhost:27019", priority: 1},
{_id: 2, host: "localhost:27020", priority: 3}]
```

- Set tags for different members:

```
{_id:0, host:"localhost:27018", tags:{dc:"east"}},
{_id:1, host:"localhost:27019", tags:{dc:"mid"}},
{_id:2, host:"localhost:27020", tags:{dc:"west"}}]
```

https://www.mongodb.com/docs/manual/reference/replica-configuration/#mongodb-rsconf-rsconf.settings

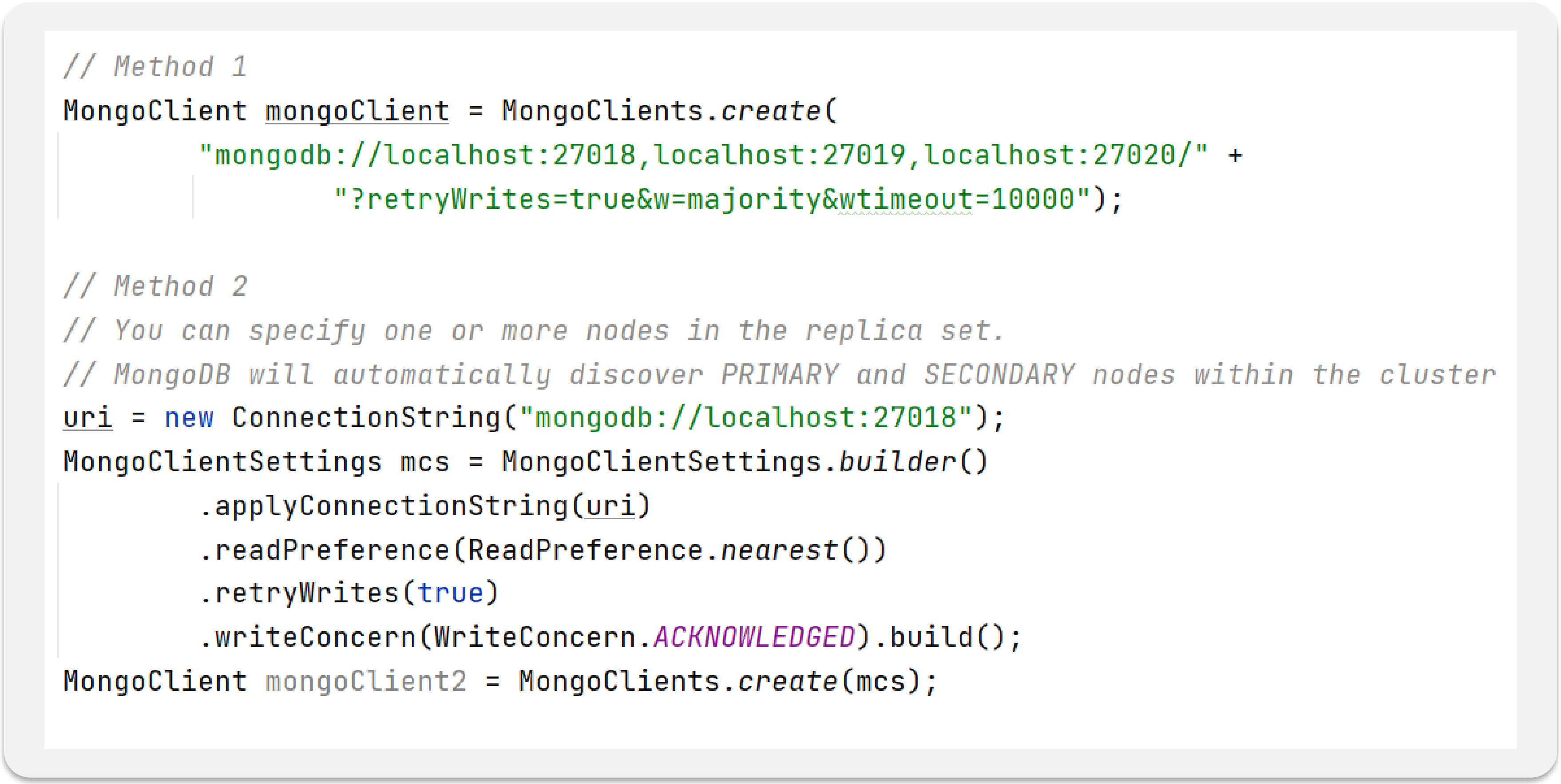
Additional configurations (2)

```
rsconf = {
    _id: "lsmdb",
    members: […],
     settings: {
        setDefaultRWConcern :{w: 2, wtimeout : 5000}}};
```

- **W** is the minimum *number of writes* that the application waits until it regains the flow of execution. The default value is *majority*, but it can be any integer >= 1.

- **Wtimeout** is the maximum *waiting time* (in *ms*) before regaining the flow of execution. If set to 0 the application will wait indefinitely or until “w” is satisfied.

Access Replica set from Java Driver



```java
// Method 1
MongoClient mongoClient = MongoClients.create(
        "mongodb://localhost:27018,localhost:27019,localhost:27020/" +
                "?retryWrites=true&w=majority&wtimeout=10000");

// Method 2
// You can specify one or more nodes in the replica set.
// MongoDB will automatically discover PRIMARY and SECONDARY nodes within the cluster
uri = new ConnectionString("mongodb://localhost:27018");
MongoClientSettings mcs = MongoClientSettings.builder()
        .applyConnectionString(uri)
        .readPreference(ReadPreference.nearest())
        .retryWrites(true)
        .writeConcern(WriteConcern.ACKNOWLEDGED).build();
MongoClient mongoClient2 = MongoClients.create(mcs);
```

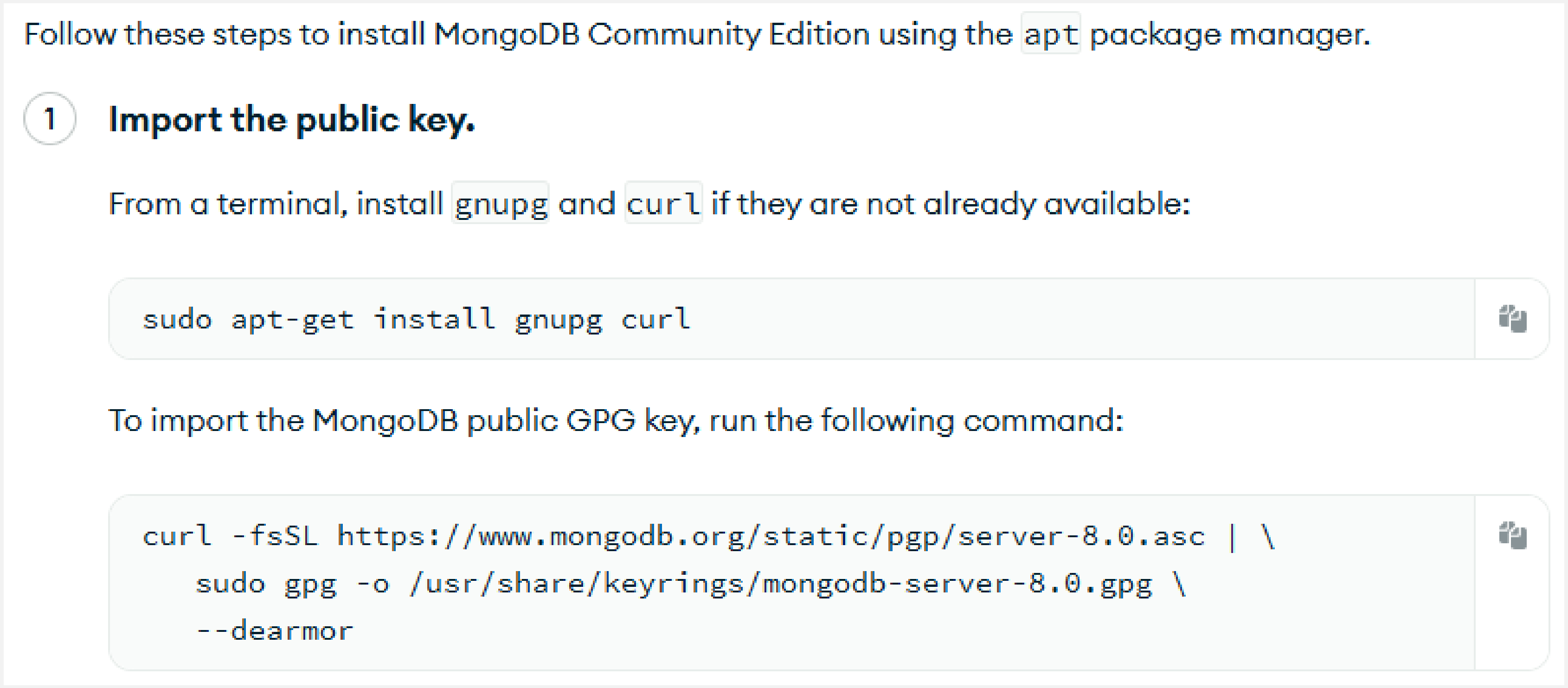
Configure a distributed Replica Set

- In this tutorial we have, at our disposal, the following **VMs**

VM	IP address	OS
Replica-0	10.1.1.11	Ubuntu 20.04.04
Replica-1	10.1.1.23	Ubuntu 20.04.04
Replica-2	10.1.1.24	Ubuntu 20.04.04

- First, we need to **connect through a VPN**
- **Install mongoDB on each VM**
- **Start and configure the servers** in a similar fashion as before

Install MongoDB on Ubuntu 20.04 (I)


Follow these steps to install MongoDB Community Edition using the `apt` package manager.

1 **Import the public key.**

From a terminal, install `gnupg` and `curl` if they are not already available:

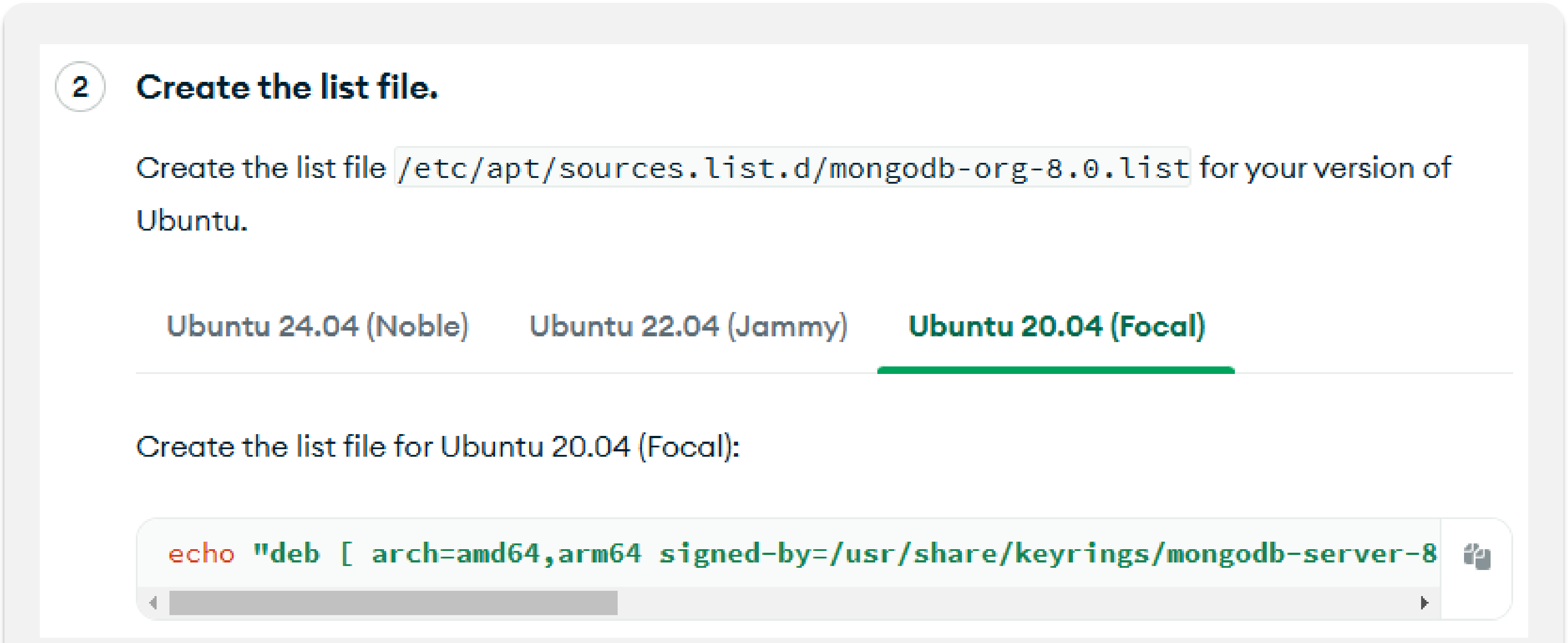
```
sudo apt-get install gnupg curl
```

To import the MongoDB public GPG key, run the following command:

```
curl -fsSL https://www.mongodb.org/static/pgp/server-8.0.asc | \
   sudo gpg -o /usr/share/keyrings/mongodb-server-8.0.gpg \
   --dearmor
```

https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/

Install MongoDB on Ubuntu 20.04 (II)



2 **Create the list file.**

Create the list file `/etc/apt/sources.list.d/mongodb-org-8.0.list` for your version of Ubuntu.

Ubuntu 24.04 (Noble) | Ubuntu 22.04 (Jammy) | **Ubuntu 20.04 (Focal)**

Create the list file for Ubuntu 20.04 (Focal):

```
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-8
```

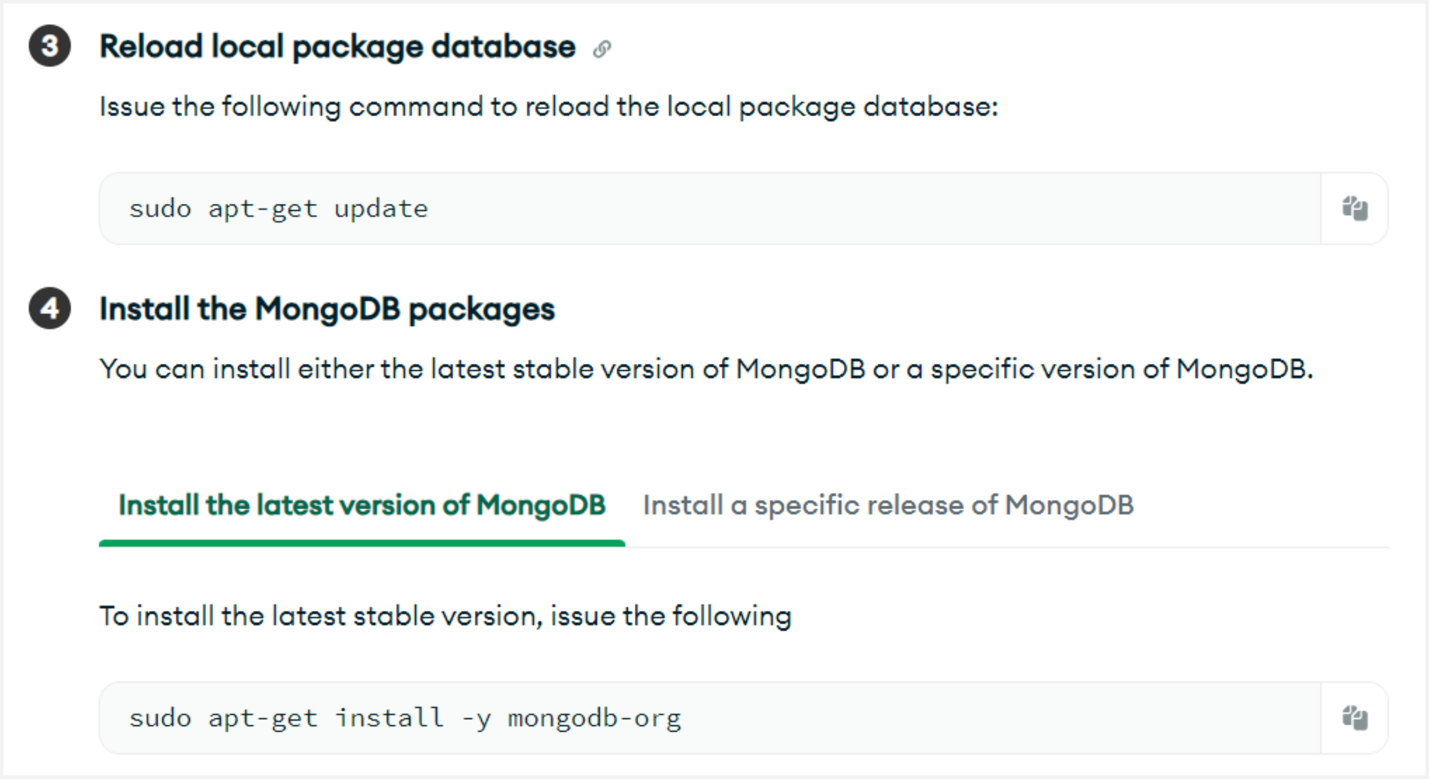
https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Install MongoDB on Ubuntu 20.04 (III)



3 Reload local package database

Issue the following command to reload the local package database:

```
sudo apt-get update
```

4 Install the MongoDB packages

You can install either the latest stable version of MongoDB or a specific version of MongoDB.

Install the latest version of MongoDB
Install a specific release of MongoDB

To install the latest stable version, issue the following

```
sudo apt-get install -y mongodb-org
```

https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/

Setup the VPN

1. Install OpenVPN from https://openvpn.net/community-downloads/
2. Download the *largescale.ovpn* file containing all the parameters for the VPN.
3. Start OpenVPN and select “Import from file”
4. Select the *largescale.ovpn* file and start the connection

If everything goes well, you should be able to connect through ssh via user=**root** and pass=**root.**

```
ssh root@10.1.1.11
```

Configure VMs (1)

- Open 3 different terminals and issues the following commands

```
#Terminal 1
ssh root@10.1.1.11
mkdir data
mongod --replSet lsmdb --dbpath ~/data --port
27020 --bind_ip localhost,10.1.1.11 --oplogSize
200
```

```
#Terminal 2
ssh root@10.1.1.23
mkdir data
mongod --replSet lsmdb --dbpath ~/data --port
27020 --bind_ip localhost,10.1.1.23 --oplogSize
200
```

Configure VMs (2)

```
#Terminal 3
ssh root@10.1.1.24
mkdir data
mongod --replSet lsmdb --dbpath ~/data --port
27020 --bind_ip localhost,10.1.1.24 --oplogSize
200
```

Configure the Replica set

- Connect to one of the VM through ssh (open a 4th terminal)

```
#Terminal 4
ssh root@10.1.1.11
mongo --port 27020

rsconf = {
   _id: "lsmdb",
   members: [
      {_id: 0, host: "10.1.1.11:27020", priority:1},
      {_id: 1, host: "10.1.1.23:27020", priority:2},
      {_id: 2, host: "10.1.1.14:27020", priority:5}]
};

rs.initiate(rsconf);
```