MongoDB Replication

Kayartaya Vinod

Replication

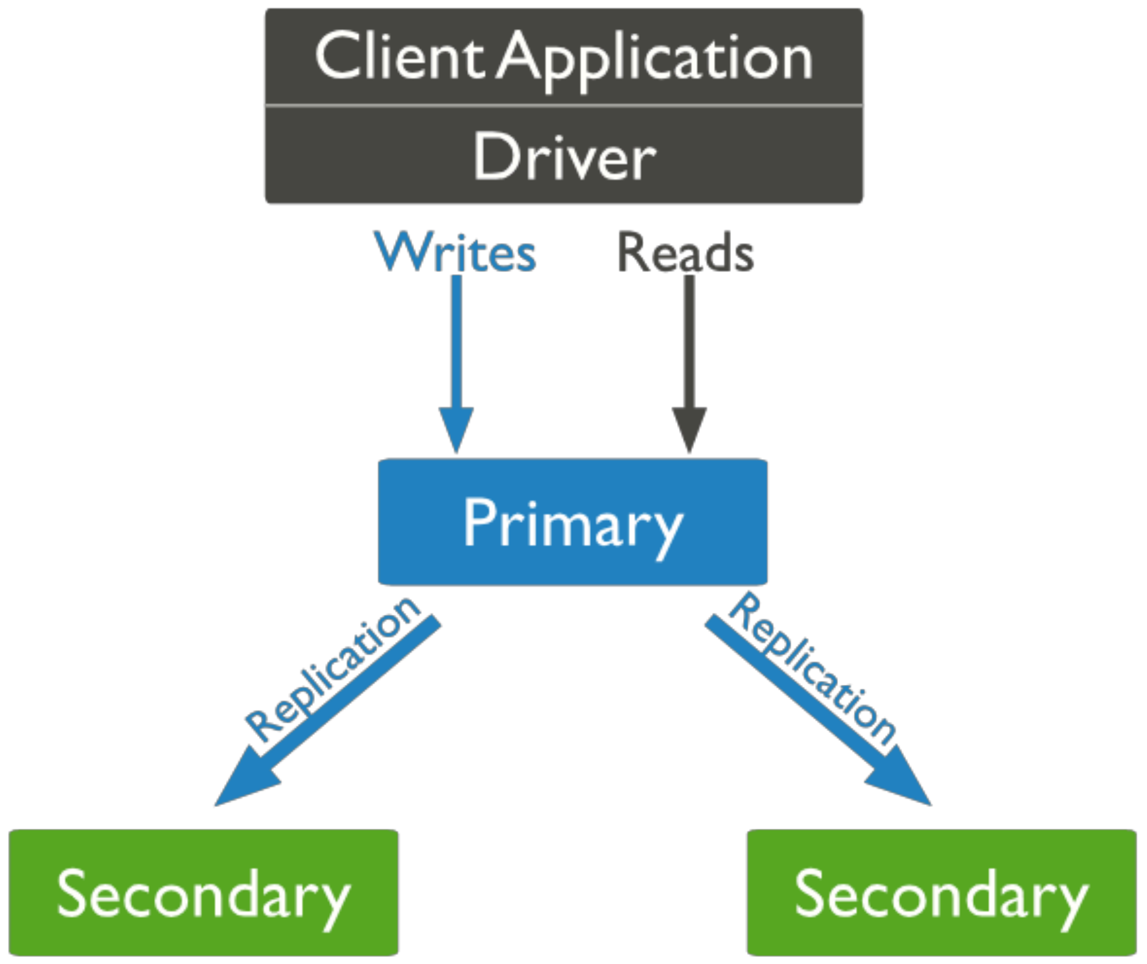
- A replica set in MongoDB is a group of mongod processes that maintain the same data set.
- Replica sets provide redundancy and high availability, and are the basis for all production deployments.
- With multiple copies of data on different database servers, replication provides a level of fault tolerance against the loss of a single database server.

Replication

- In some cases, replication can provide increased read capacity as clients can send read operations to different servers.
- Maintaining copies of data in different data centers can increase data locality and availability for distributed applications.
- You can also maintain additional copies for dedicated purposes, such as disaster recovery, reporting, or backup.

Replica 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.
- The primary node receives all write operations.



Replica Set

- The primary records all changes to its data sets in its 'operation log', i.e. oplog.

- The secondaries replicate the primary’s oplog and apply the operations to their data sets such that the secondaries’ data sets reflect the primary’s data set.

- If the primary is unavailable, an eligible secondary will hold an election to elect itself the new primary.

Replica Set

- You may add an extra mongod instance to a replica set as an arbiter, which does not maintain a data set.

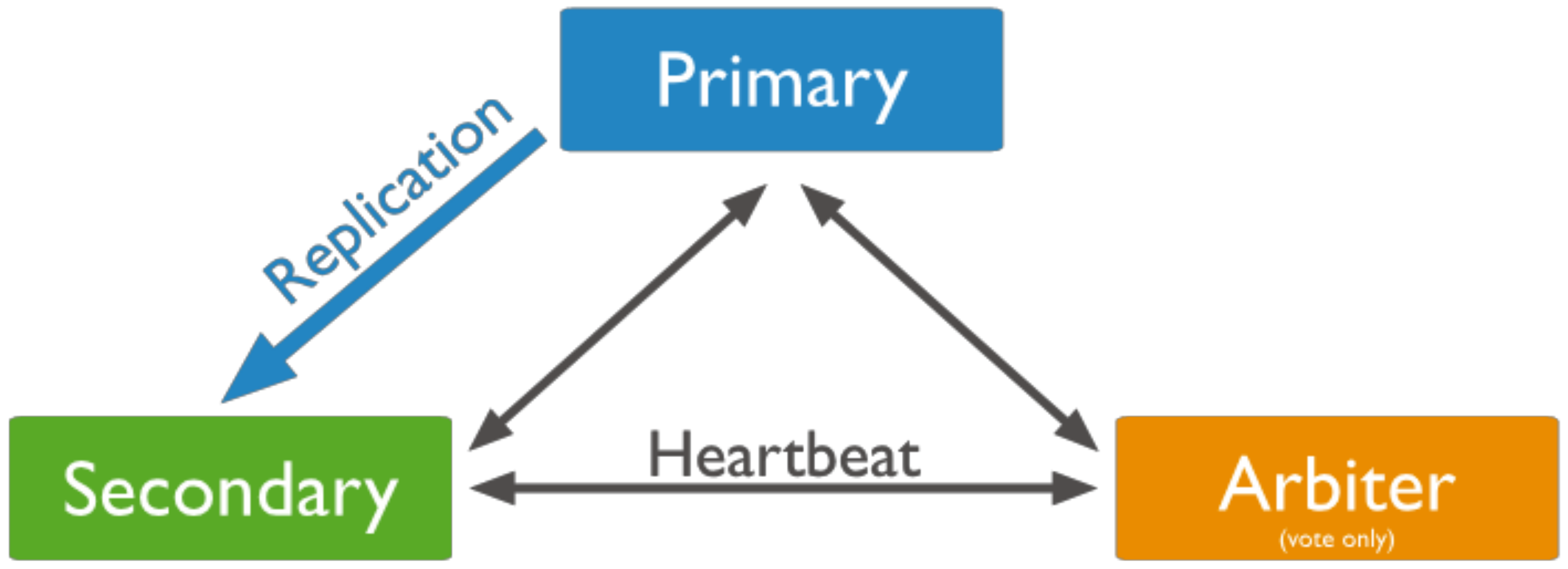
- The purpose of an arbiter is to maintain a quorum in a replica set by responding to heartbeat and election requests by other replica set members.

- If your replica set has an even number of members, add an arbiter to obtain a majority of votes in an election for primary.

- Arbiters do not require dedicated hardware.

Replica Set

- An arbiter will always be an arbiter whereas a primary may step down and become a secondary and a secondary may become the primary during an election.



Asynchronous Replication

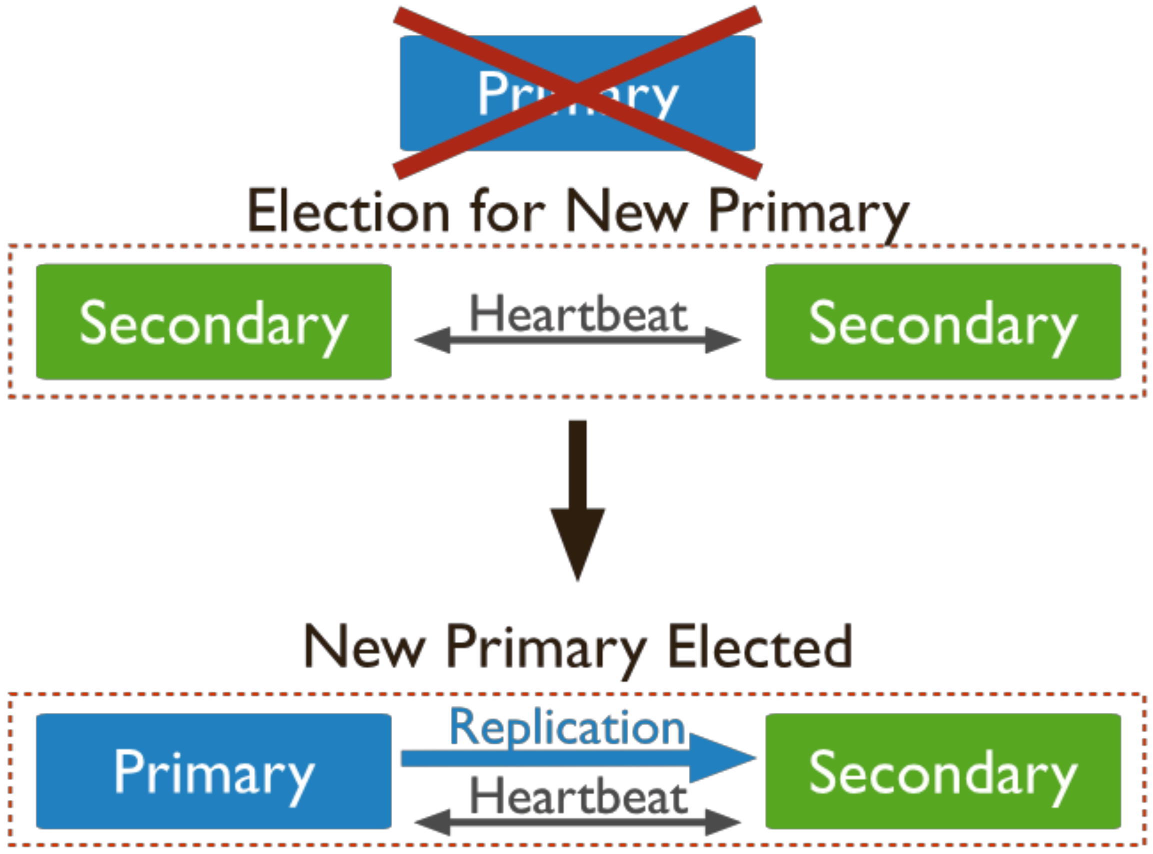
- Secondaries apply operations from the primary asynchronously.

- By applying operations after the primary, sets can continue to function despite the failure of one or more members.

Automatic Failover

- When a primary does not communicate with the other members of the set for more than the configured electionTimeoutMillis period (10 seconds by default), an eligible secondary calls for an election to nominate itself as the new primary.

- The cluster attempts to complete the election of a new primary and resume normal operations.



Automatic Failover

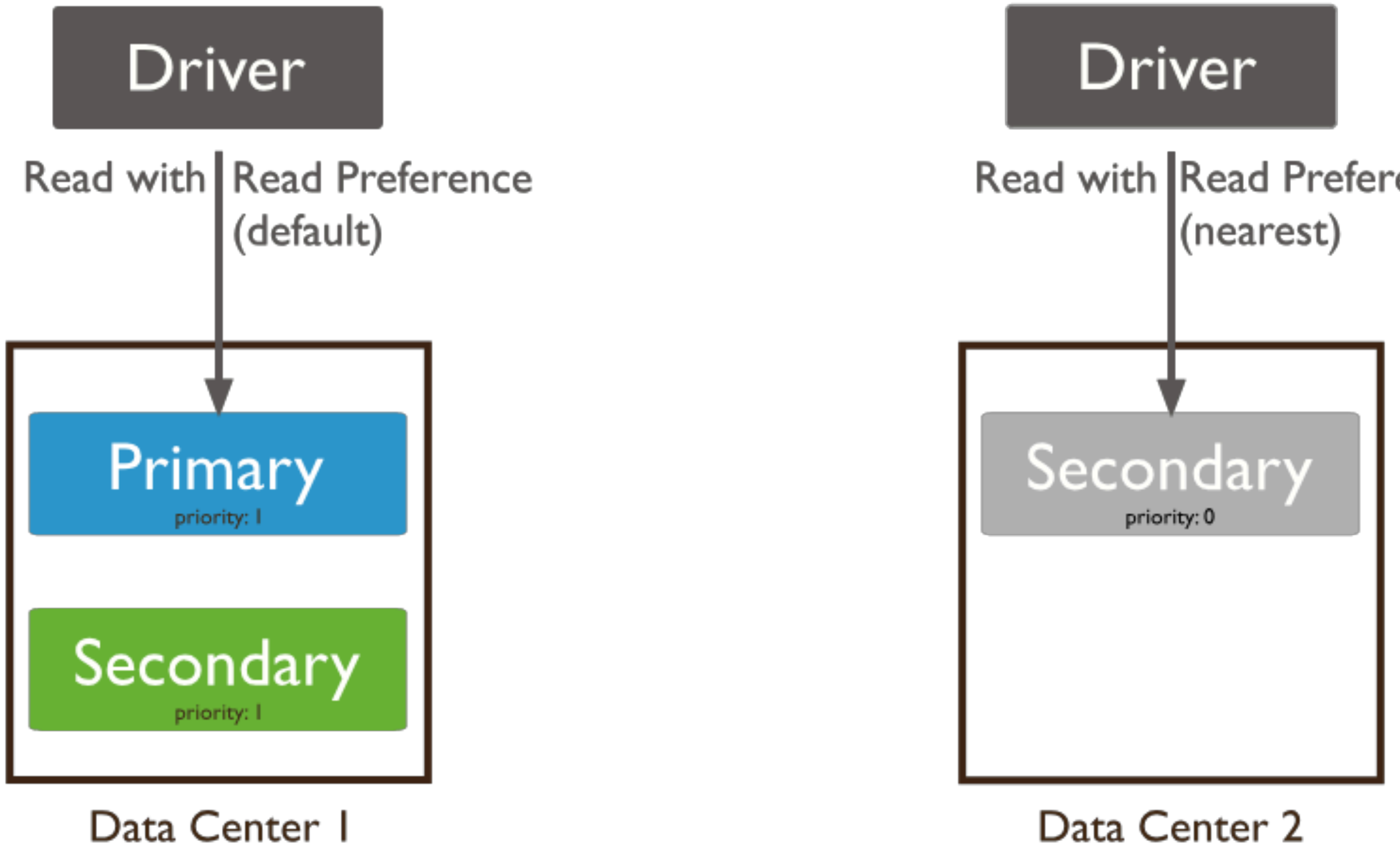
- The replica set cannot process write operations until the election completes successfully.
- The replica set can continue to serve read queries if such queries are configured to run on secondaries while the primary is offline.

Read Operations

- By default, clients read from the primary
  - However, clients can specify a read preference to send read operations to secondaries.
- Asynchronous replication to secondaries means that reads from secondaries may return data that does not reflect the state of the data on the primary.

Read Preference

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



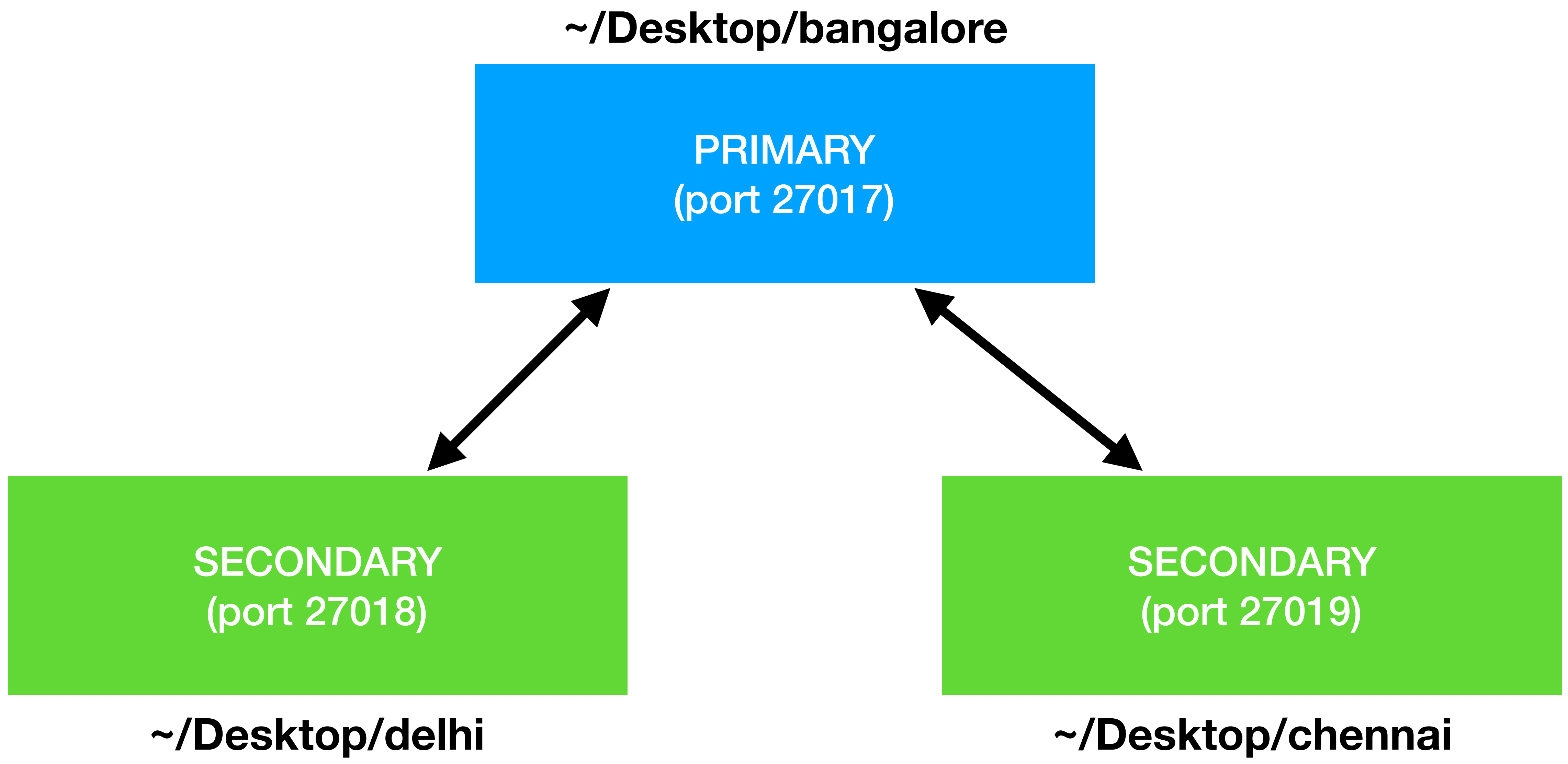
Read Preference

- primaryPreferred
- secondary
- secondaryPreferred
- nearest

- In general, do not use secondary and secondaryPreferred to provide extra capacity for reads, because:
    - All members of a replica have roughly equivalent write traffic; as a result, secondaries will service reads at roughly the same rate as the primary.
    - Replication is asynchronous and there is some amount of delay between a successful write operation and its replication to secondaries.
        - Reading from a secondary can return stale data; reading from different secondaries may result in non-monotonic reads.

Replica Set - How to?

For demo purpose, we will be running 3 mongos instances in the same computer with different port numbers and data directories



Replica Set - How to?

```
mongod --replSet demoRs \
   --bind_ip localhost,172.20.10.2 \
   --dbpath ~/Desktop/bangalore --port 27017
```

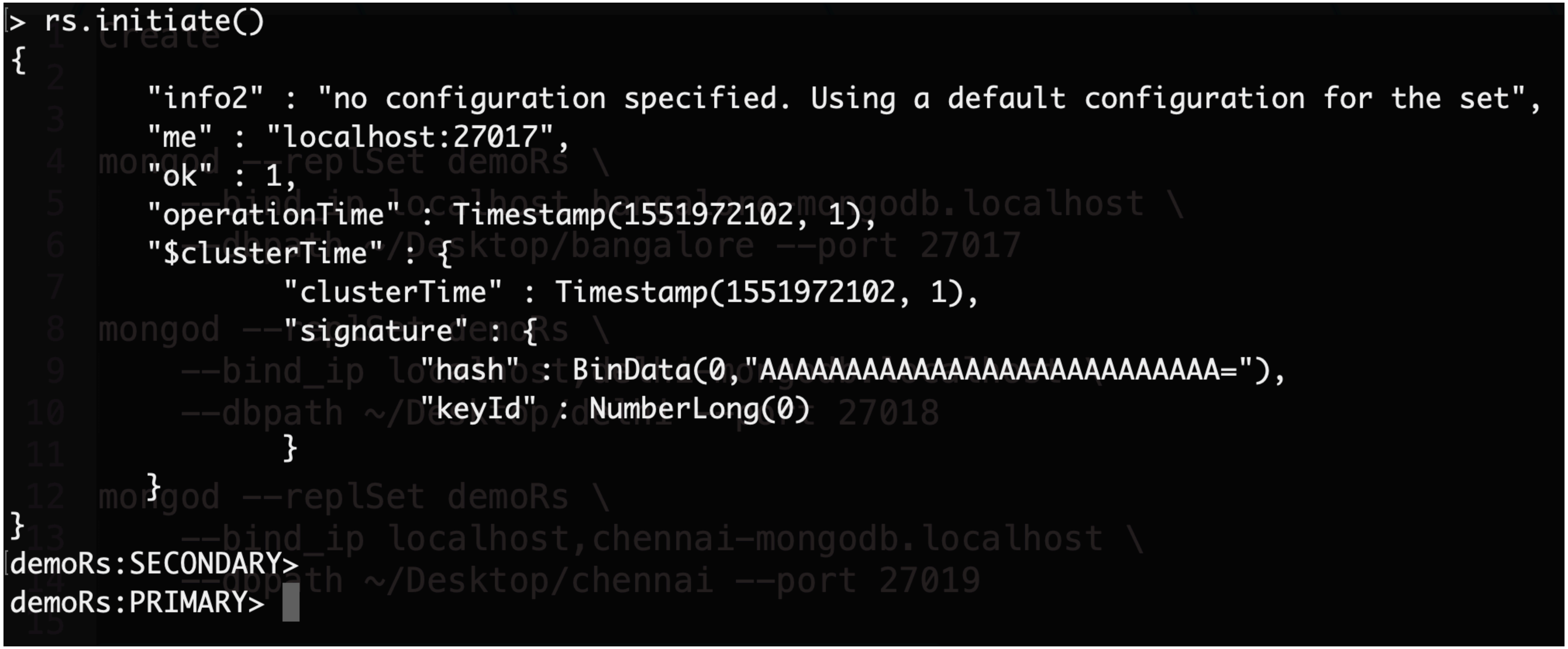
```
mongod --replSet demoRs \
   --bind_ip localhost,172.20.10.2 \
   --dbpath ~/Desktop/delhi --port 27018
```

```
mongod --replSet demoRs \
   --bind_ip localhost,172.20.10.2 \
   --dbpath ~/Desktop/chennai --port 27019
```

Replica Set - How to?

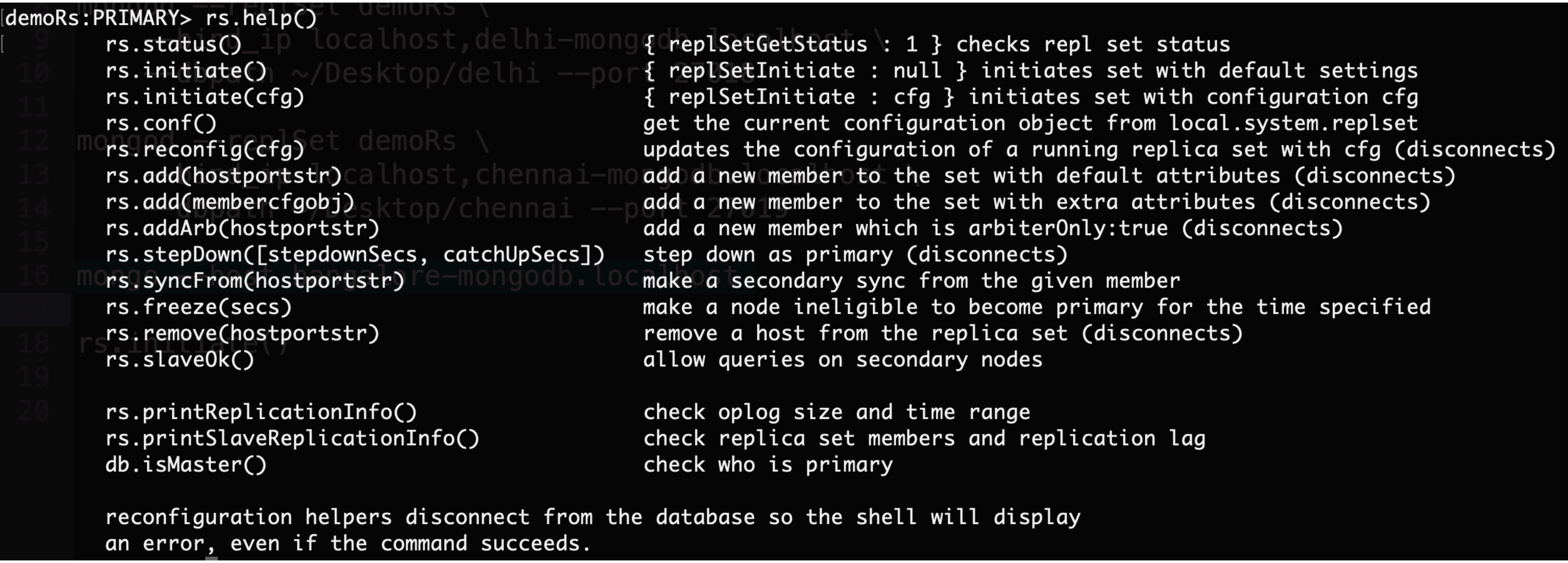
- Use ‘mongo’ to connect one of the 3 servers started
  - This will be treated as the PRIMARY
- Execute the rs.initiate() function to initiate the replica set

Replica Set - How to?



```
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration for the set",
        "me" : "localhost:27017",
        "ok" : 1,
        "operationTime" : Timestamp(1551972102, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1551972102, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
demoRs:SECONDARY>
demoRs:PRIMARY>
```

Replica Set - How to?

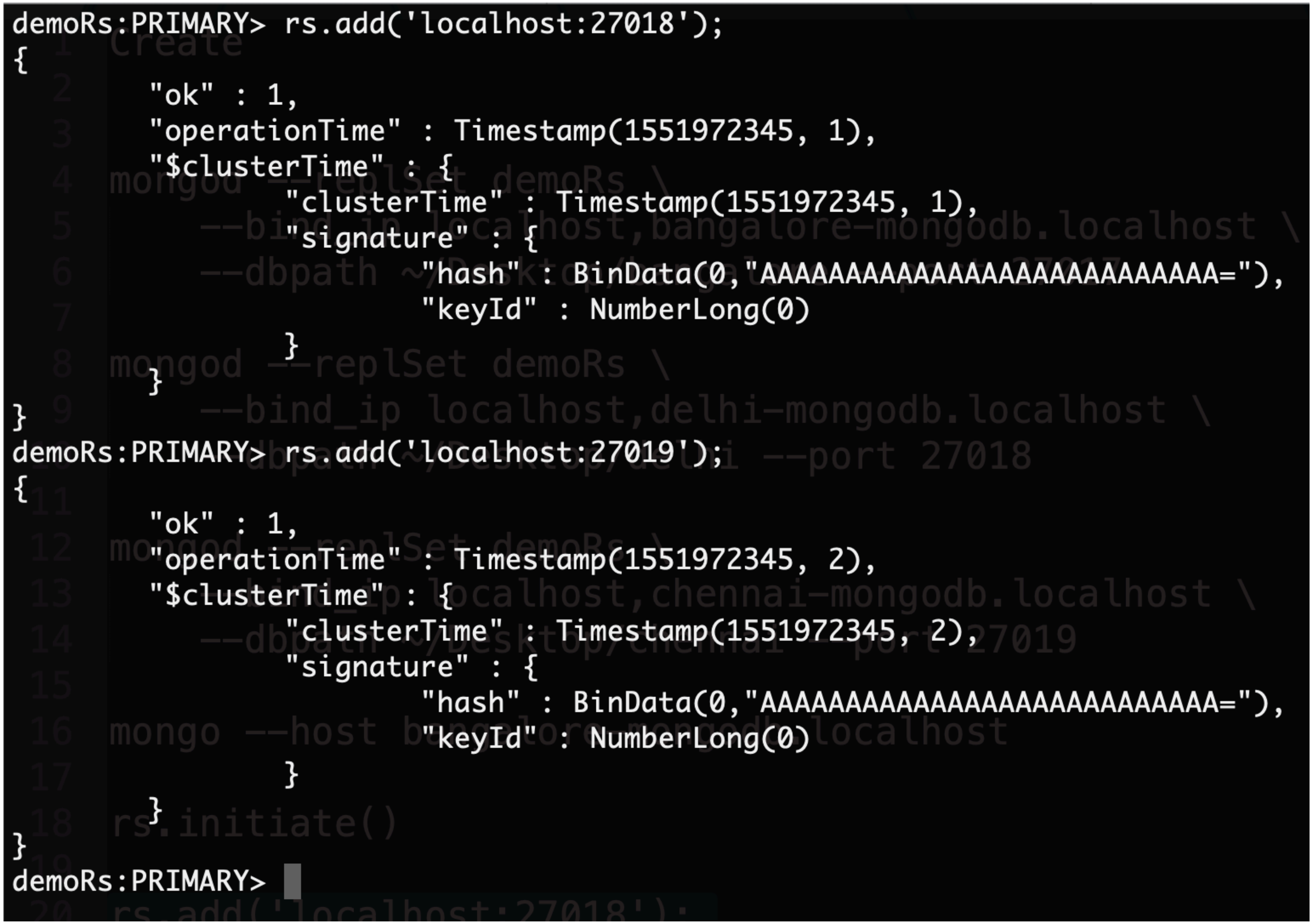


```
demoRs:PRIMARY> rs.help()
	rs.status()                                { replSetGetStatus : 1 } checks repl set status
	rs.initiate()                              { replSetInitiate : null } initiates set with default settings
	rs.initiate(cfg)                           { replSetInitiate : cfg } initiates set with configuration cfg
	rs.conf()                                  get the current configuration object from local.system.replset
	rs.reconfig(cfg)                           updates the configuration of a running replica set with cfg (disconnects)
	rs.add(hostportstr)                        add a new member to the set with default attributes (disconnects)
	rs.add(membercfgobj)                       add a new member to the set with extra attributes (disconnects)
	rs.addArb(hostportstr)                     add a new member which is arbiterOnly:true (disconnects)
	rs.stepDown([stepdownSecs, catchUpSecs])   step down as primary (disconnects)
	rs.syncFrom(hostportstr)                   make a secondary sync from the given member
	rs.freeze(secs)                            make a node ineligible to become primary for the time specified
	rs.remove(hostportstr)                     remove a host from the replica set (disconnects)
	rs.slaveOk()                               allow queries on secondary nodes

	rs.printReplicationInfo()                  check oplog size and time range
	rs.printSlaveReplicationInfo()             check replica set members and replication lag
	db.isMaster()                              check who is primary

	reconfiguration helpers disconnect from the database so the shell will display
	an error, even if the command succeeds.
```

Replica Set - How to?



```
demoRs:PRIMARY> rs.add('localhost:27018');
{
        "ok" : 1,
        "operationTime" : Timestamp(1551972345, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1551972345, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
demoRs:PRIMARY> rs.add('localhost:27019');
{
        "ok" : 1,
        "operationTime" : Timestamp(1551972345, 2),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1551972345, 2),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
demoRs:PRIMARY> 
```