mongoDB

{name: “mongo”, type:“DB”}

```
db.NLJUG.insert(
{ “presenters” :“Maikel Alderhout  & Bas van Oudenaarde”,
  “type” :    “JFALL”,
  “datum” :  ISODate(31/10/2012)  } )
```

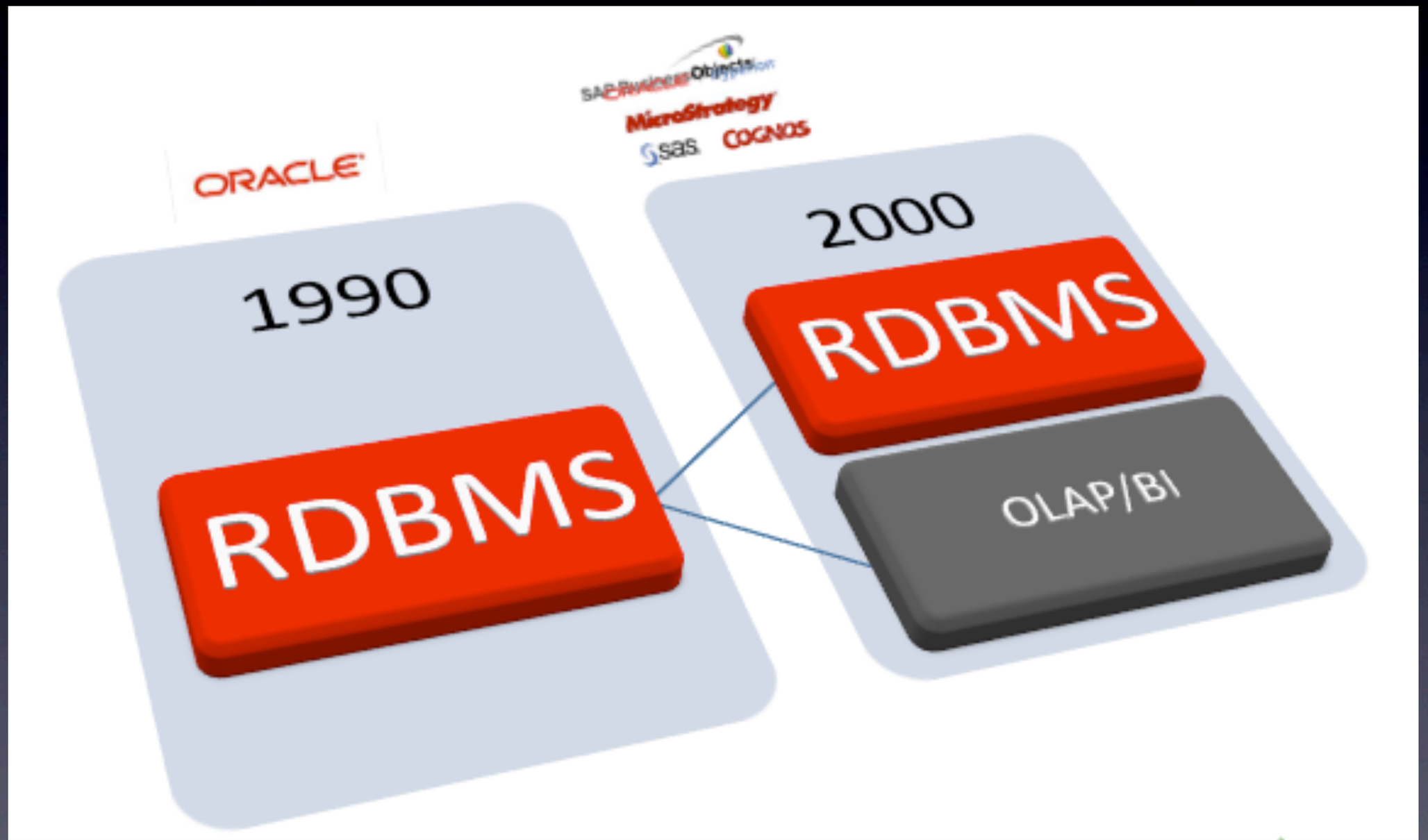
NoSQL Hype

- **Not only** ... of **No** SQL

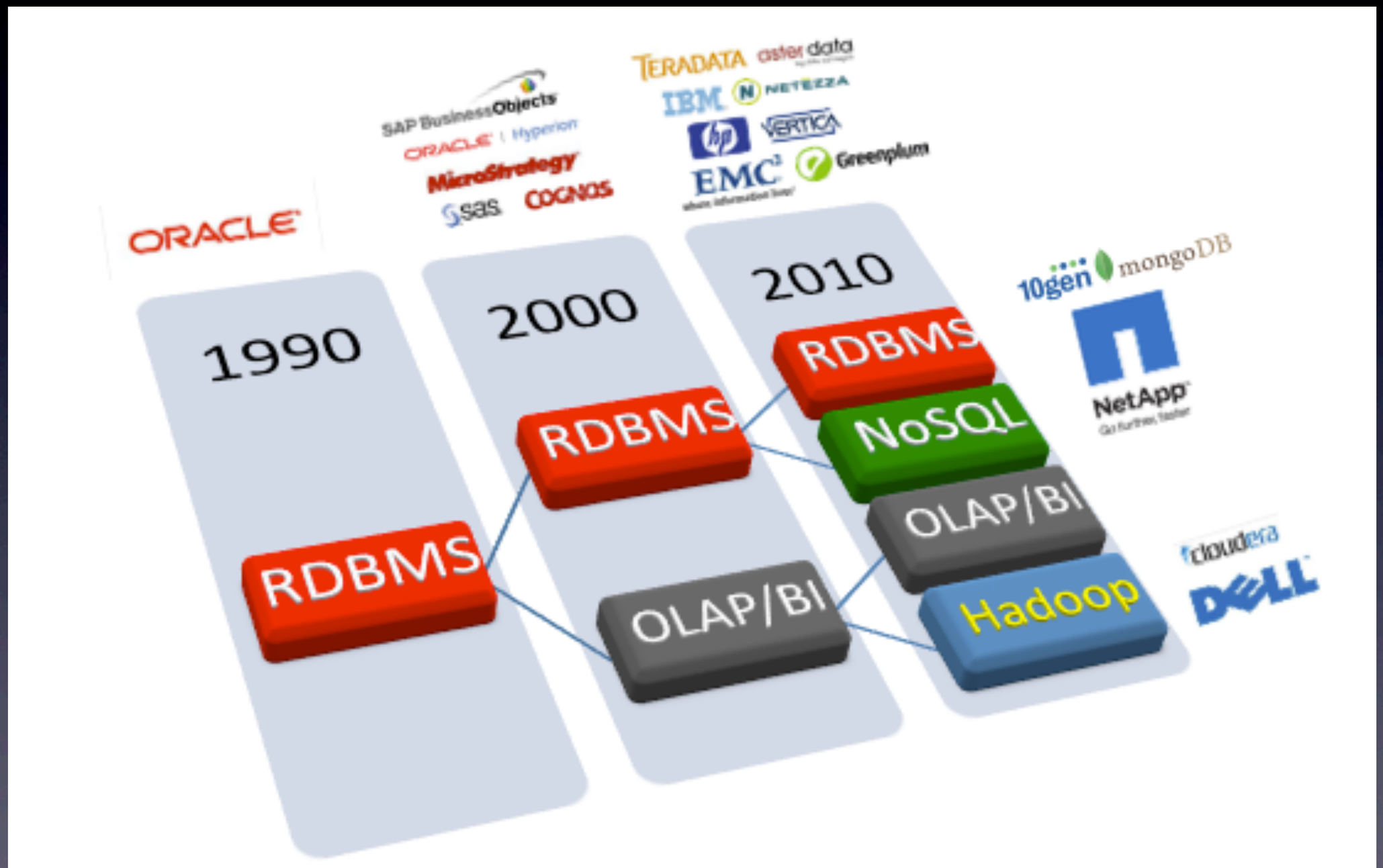
  '**not only**' relationele data opslag

Database evolution #1

Database evolution #2



Database evolution #3



De huidige trends hebben een behoorlijke impact in het traditionele database landschap

Agile Development

- Iterative
- Continuous

Data Volume,
Type & Use

- Trillions of records
- 100's of millions of queries per second
- Real-Time Analytics
- Unstructured / semi-structured

New Hardware Architectures

- Commodity servers
- Cloud Computing
- Horizontal Scaling

Hoezo Java problemen?

- Schaalbaarheid 3-tier architecturen
- ORM, mapping vanuit domain objecten.
  Lekker alles aan elkaar knopen
- Caching, niet te veel naar de database
- Alles in memory, dan maar?

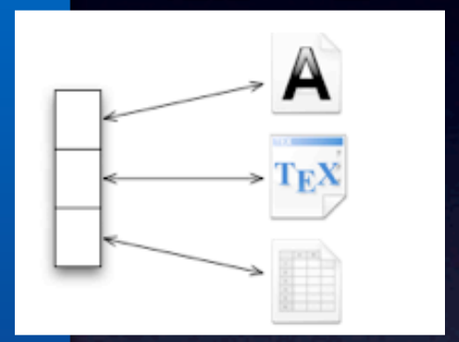
NoSQL oplossingen

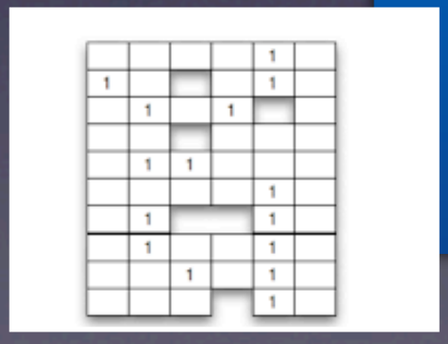
Key-value store

bv. Voldemort, Dynomite, Tokyo

Document

bv CouchDB, **MongoDB**, Riak



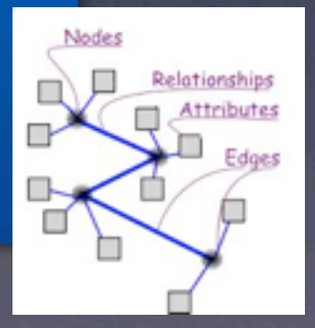


ColumnFamily / BigTable clones

bv Google BT, HBase, Hypertable, Cassandra

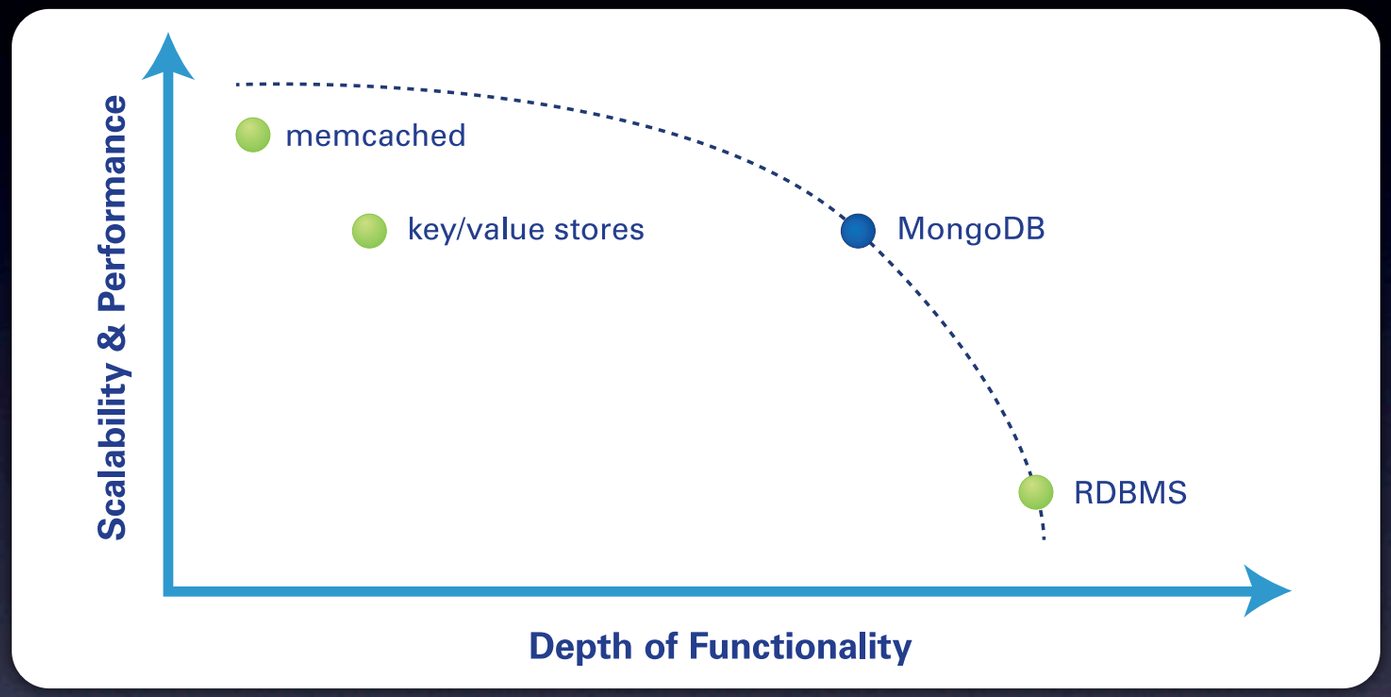
Graph databases

Neo4J, Sones, AllegroGraph




mongoDB

MongoDB is a scalable, high-performance NoSQL database.



- Open source, written in C++
- Document-oriented Storage
  - Based on JSON Documents
  - Schema-less
- Full featured indexes, query language
- Replication & High Availability
- Auto-sharding

JSON Document

Person

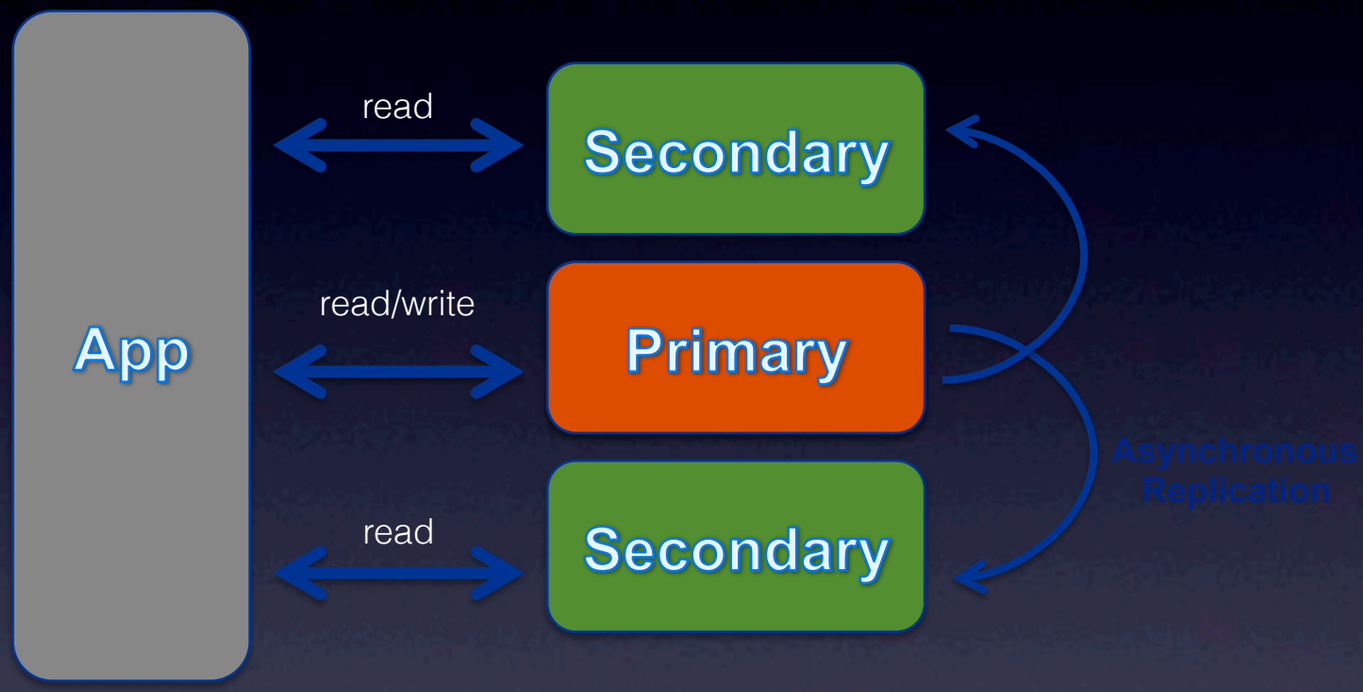
```
{

name" : "Bas",
"group" : [ "Open Source", "mongoDB", "Big Data" ]

}
```

Replication

Sharding

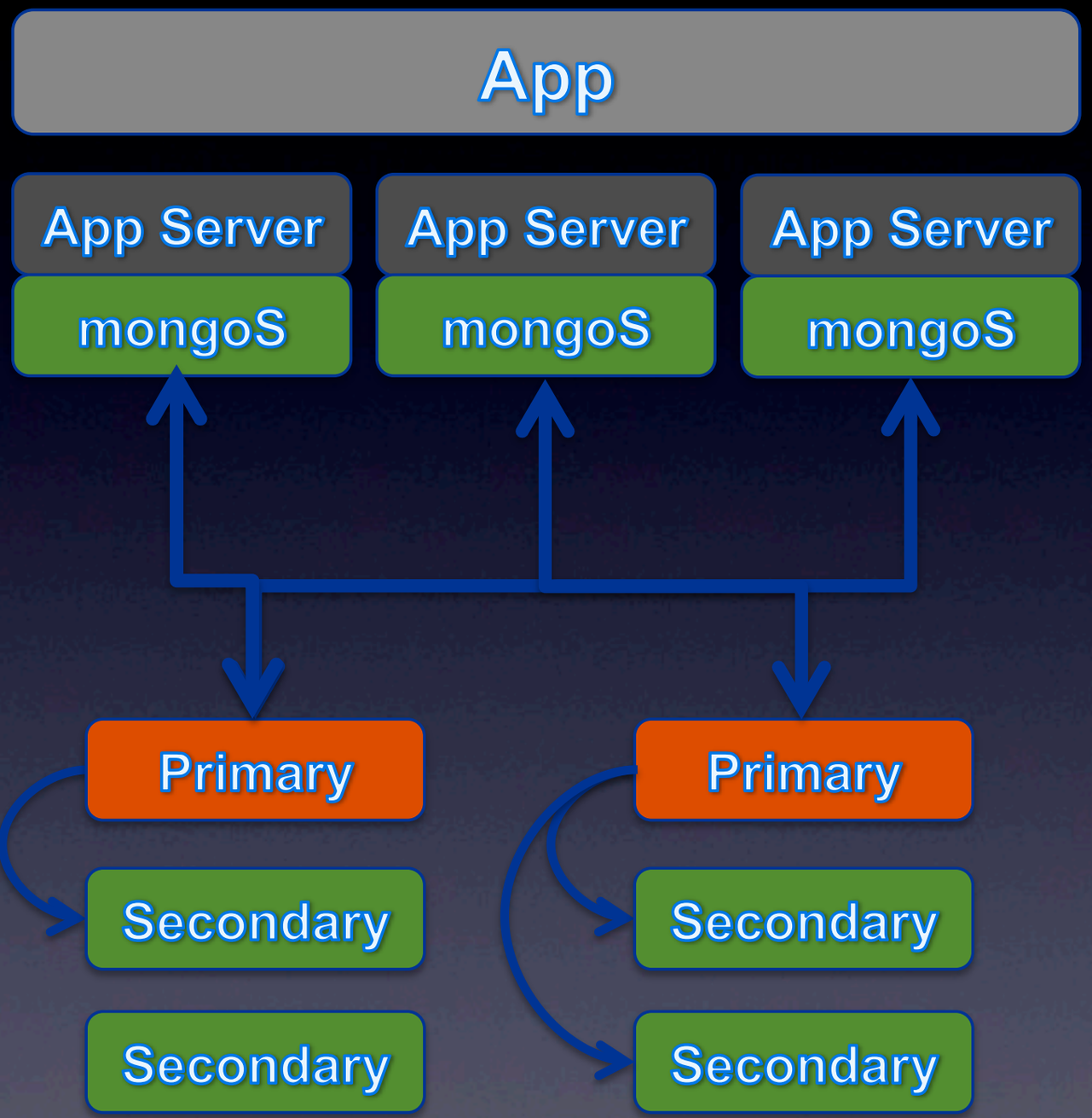
Durability

Replicatie



Automatic election of new Primary

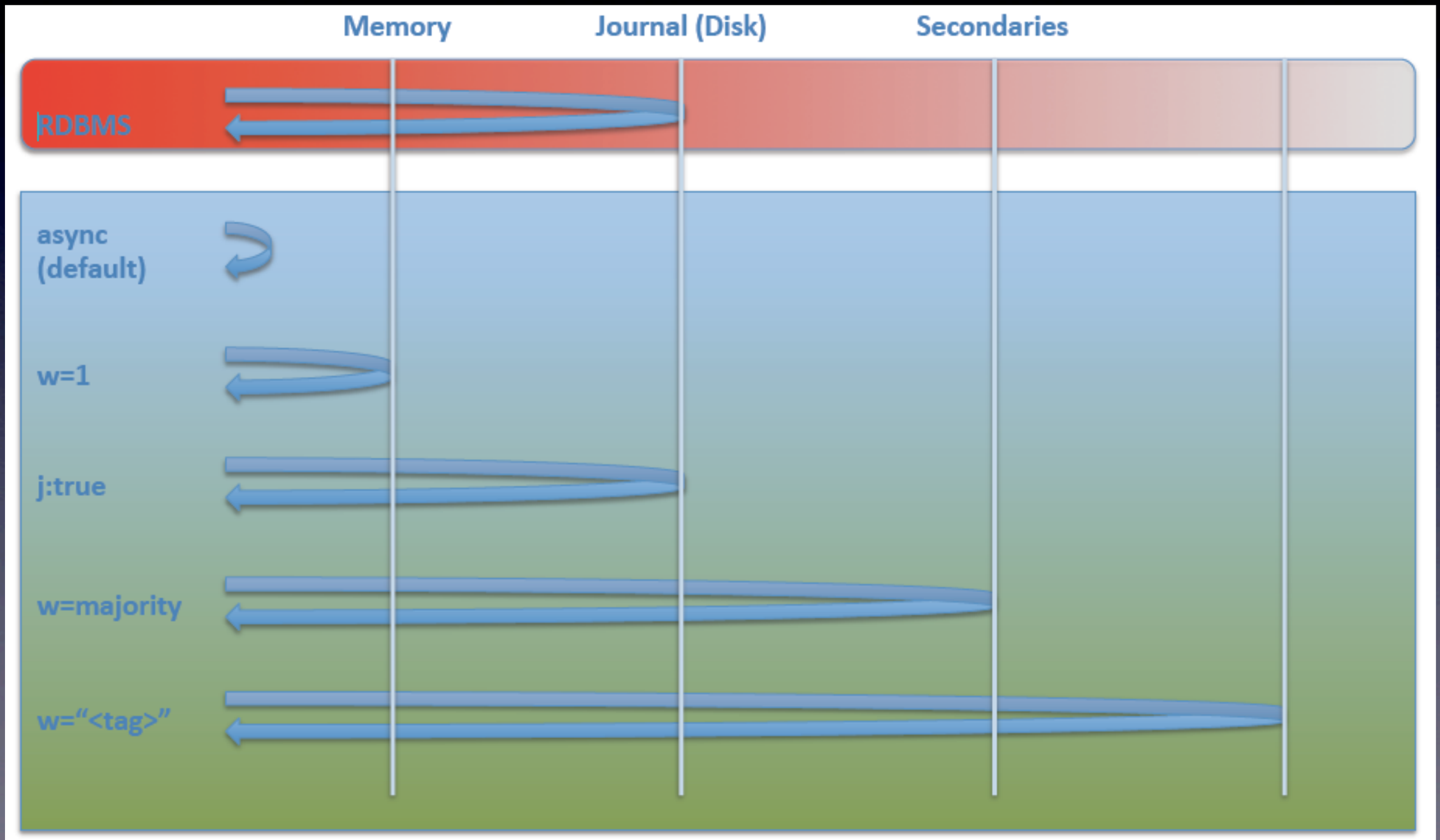
Sharding


App
App Server
App Server
App Server
mongoS
mongoS
mongoS
Primary
Primary
Secondary
Secondary
Secondary
Secondary

Sharding features

- Automatic partitioning
- Automatic Load-Balancing across shards
- Range-based
- Convert to sharded system with no downtime
- Fully consistent
- Application code unaware of data location
- Zero code changes

Durability


Memory
Journal (Disk)
Secondaries
RDBMS
async
(default)
w=1
j:true
w=majority
w="<tag>"

Lab00 Install MongoDB

OSX

In a terminal session, begin by downloading the latest release. In most cases you will want to download the 64-bit version of MongoDB en Replace x.y.z with the current stable version (i.e. 2.2.0).

```
1 curl http://downloads.mongodb.org/osx/mongodb-osx-x86_64-x.y.z.tgz > mongo.tgz
```

Once you've downloaded the release, issue the following command to extract the files from the archive:

```
1 tar -zxvf mongo.tgz
```

Before you start mongod for the first time, you will need to create the data directory. By default, mongod writes data to the /data/db/ directory. To create this directory, and set the appropriate permissions use the following commands:

```
1 sudo mkdir -p /data/db
2 sudo chown `id -u` /data/db
```

You can start a mongod directly in the terminal without creating a control script. This command assumes that the mongod binary is accessible via your system's search path, and that you have created a default configuration file located at /etc/mongod.conf.

```
mongod --config /etc/mongod.conf
```

Now you can start MongoDB from another shell

```
./bin/mongo
```

More info

Install MongoDB on onther Linux environments

Install MongoDB on Windows

Lab01 Basic CRUD

Type some commands like

```
> help
 db.help() help on db methods
 db.mycoll.help() help on collection methods
 sh.help() sharding helpers
 rs.help() replica set helpers
 help admin administrative help
 help connect connecting to a db help
 help keys key shortcuts
 help misc misc things to know
 help mr mapreduce
 show dbs show database names
 show collections show collections in current database
 show users show users in current database
 show profile show most recent system.profile entries with time >= 1ms
 show logs show the accessible logger names
 show log [name] prints out the last segment of log in memory, 'global' is default
 use <db_name> set current database
 db.foo.find() list objects in collection foo
 db.foo.find( { a : 1 } ) list objects in foo where a == 1
 it result of the last line evaluated; use to further iterate
```

```
 DBQuery.shellBatchSize = x set default number of items to display on shell
 exit quit the mongo shell
```

```
> version()
version: 2.2.0
```

```
> show databases
```

```
> show collections
```

Switch database to test

```
> use test
```

Show database statistics

```
> db.stats()
```

Show Javacript function of db.stats()

```
> db.stats
```

Make a new database

```
> use portraitGallery
switched to db portraitGallery
```

Create first documents in collection person

```
db.person.insert(
{
 "name" : "Bas",
 "group" : [ "Open Source", "mongoDB", "Big Data" ],
 } )
```

and another one

```
db.person.save(
 {
 "name" : "Maikel",
 "group" : [ "Oracle", "ExaData", "Big Data"],
 } )
```

```
Functionally, save and insert are very similar, especially if no _id
value is passed.  However, if an _id key is passed, save() will update
the document, while insert() will throw a duplicate key error.
```

```
> show collections
person
system.indexes
```

```
> db.person.count()
2
```

Querying

```
> db.person.find()
{ "_id" : ObjectId("508ae77091896857c075af71"), "name" : "Bas", "group" : [ "Open
Source", "mongoDB", "Big Data" ] }
{ "_id" : ObjectId("508ae77d91896857c075af72"), "name" : "Maikel", "group" : [
"Oracle", "ExaData", "Big Data" ] }
```

> db.person.find({“group” : “Big Data” })

```
{ "_id" : ObjectId("508ae77091896857c075af71"), "name" : "Bas", "group" : [ "Open
Source", "mongoDB", "Big Data" ] }
{ "_id" : ObjectId("508ae77d91896857c075af72"), "name" : "Maikel", "group" : [
"Oracle", "ExaData", "Big Data" ] }
```

```
> db.person.find({"group" : "Oracle" })
```

```
{ "_id" : ObjectId("508ae77d91896857c075af72"), "name" : "Maikel", "group" : [
"Oracle", "ExaData", "Big Data" ] }
```

Update

If the person Maikel later decides that he no longer wants his groups stored in his document, he can remove the value just as easily using the $unset operator:

```
> db.person.update({name: "Maikel"}, {$unset: {group: 1}})
> db.person.find()
{ "_id" : ObjectId("508ae77091896857c075af71"), "name" : "Bas", "group" : [ "Open
Source", "mongoDB", "Big Data" ] }
{ "_id" : ObjectId("508ae77d91896857c075af72"), "name" : "Maikel" }
```

```
Replace all group members:
```

```
> db.person.update({name: "Bas" },{ $set: {"group" : ["EOSS", "SQL", "OpenStack"]}})
> db.person.find()
{ "_id" : ObjectId("508ae77091896857c075af71"), "group" : [ "EOSS", "SQL",
"OpenStack" ] }
{ "_id" : ObjectId("508ae77d91896857c075af72"), "name" : "Maikel" }
```

```
Add one group member
```

```
> db.person.update({name: "Bas" },{ $push: { group: "Oracle" } })
> db.person.find()
{ "_id" : ObjectId("508aecf791896857c075af73"), "group" : [ "EOSS", "SQL",
"OpenStack", "Oracle" ], "name" : "Bas" }
{ "_id" : ObjectId("508ae77d91896857c075af72"), "name" : "Maikel" }
```

```

```

Upsert

An upsert is a special type of update. If no document is found that matches the update criteria, a new document will be created by combining the criteria and update documents. If a matching document is found, it will be updated normally. Add true parameter.

```
> db.person.update({name: "Jan" },{ $set: {"group" : ["OpenStack"]}},true)
> db.person.find()
{ "_id" : ObjectId("508aecf791896857c075af73"), "group" : [ "EOSS", "SQL",
"OpenStack" ], "name" : "Bas" }
{ "_id" : ObjectId("508ae77d91896857c075af72"), "name" : "Maikel" }
{ "_id" : ObjectId("508af10eb97c8d1dc77ab660"), "group" : [ "OpenStack" ], "name" :
"Jan" }
```

Multiple Updates

Add true parameter

```
> db.person.update( { }, { $set: {"group" : ["EOSS", "mongoDB", "OpenStack"]}}, true,
true )
> db.person.find()
{ "_id" : ObjectId("508aecf791896857c075af73"), "group" : [ "EOSS", "mongoDB",
"OpenStack" ], "name" : "Bas" }
{ "_id" : ObjectId("508ae77d91896857c075af72"), "group" : [ "EOSS", "mongoDB",
"OpenStack" ], "name" : "Maikel" }
{ "_id" : ObjectId("508af10eb97c8d1dc77ab660"), "group" : [ "EOSS", "mongoDB",
"OpenStack" ], "name" : "Jan" }
```

Delete

If given no parameters, a remove operation will clear a collection of all its documents. To get rid of Maikel type:

> db.person.remove({“name”: “Maikel”})

```
> db.person.find()
{ "_id" : ObjectId("508aecf791896857c075af73"), "group" : [ "EOSS", "mongoDB",
```

```
"OpenStack" ], "name" : "Bas" }
{ "_id" : ObjectId("508af10eb97c8d1dc77ab660"), "group" : [ "EOSS", "mongoDB",
"OpenStack" ], "name" : "Jan" }
```

If your intent is to delete the collection along with all of its indexes, use the drop() method:

```
> db.person.drop()
```

Lab02 Replication

Start by creating a data directory for each replica set member, one for the primary and one for the secundary:

```
mkdir /data/node1
mkdir /data/node2
```

Next, start each member as a separate mongod. Since you'll be running each process on the same machine, it's probably easiest to start each mongod in a separate terminal window:

```
mongod --replSet person --dbpath /data/node1 --port 40001
mongod --replSet person --dbpath /data/node2 --port 40002
```

Logon on the primary node to proceed, you need to configure the replica set, because if you examine the mongod log output, the first thing you'll notice are error messages saying that the configuration can't be found.

```
mongo localhost:40001
MongoDB shell version: 2.2.0
connecting to: localhost:40001/test
```

```
> rs.initiate()
{
"info2" : "no configuration explicitly specified -- making one",
```

```
"me" : "Computername.local:40001",
"info" : "Config now saved locally. Should come online in about a minute.",
"ok" : 1
}
```

Now connect again to the primary node, and add the secondary node:

```
person:PRIMARY> rs.add("Computername:40002")
{ "ok" : 1 }
```

Check if the configuration is ok, with rs.status():

```
person:PRIMARY> rs.status()
{
 "set" : "person",
 "date" : ISODate("2012-10-28T19:50:52Z"),
 "myState" : 1,
 "members" : [
 {
 "_id" : 0,
 "name" : "Computername.local:40001",
 "health" : 1,
 "state" : 1,
 "stateStr" : "PRIMARY",
 "uptime" : 1266,
 "optime" : Timestamp(1351453811000, 1),
 "optimeDate" : ISODate("2012-10-28T19:50:11Z"),
 "self" : true
 },
 {
 "_id" : 1,
 "name" : "Computername.local:40002",
 "health" : 1,
 "state" : 2,
 "stateStr" : "SECONDARY",
 "uptime" : 41,
 "optime" : Timestamp(1351453811000, 1),
```

```
 "optimeDate" : ISODate("2012-10-28T19:50:11Z"),
 "lastHeartbeat" : ISODate("2012-10-28T19:50:51Z"),
 "pingMs" : 0
 }
 ],
 "ok" : 1
}
```

And now its time to check if it works. We put a person in our primary database:

```
person:PRIMARY> use portraitGallery
switched to db portraitGallery
person:PRIMARY> db.person.save(
{
"name" : "Maikel",
"group" : [ "Oracle", "ExaData", "Big Data"],
} )
```

Logon on the secondary and check if the data is there, and don't forget to enable reading with rs.slaveOk() or db.getMongo().setSlaveOk()

```
mongo localhost:40002
MongoDB shell version: 2.2.0
connecting to: localhost:40002/test
person:SECONDARY> rs.slaveOk()
person:SECONDARY> use portraitGallery
switched to db portraitGallery
person:SECONDARY> db.person.find()
{ "_id" : ObjectId("508d971dda0730903bcbb612"), "name" : "Maikel", "group" : [
"Oracle", "ExaData", "Big Data" ] }
person:SECONDARY>
```

Now we can test it with a filler script. Type in the primary something like:

```
person:PRIMARY> for(i=0; i<1000000; i++) { db.person.save({person: i}); }
```

And in the secondary check if the collection is filled:

```
person:SECONDARY> db.person.find()
 { "_id" : ObjectId("508f95e9e38917f43ae20db3"), "person" : 0 }
 { "_id" : ObjectId("508f95e9e38917f43ae20db4"), "person" : 1 }
 { "_id" : ObjectId("508f95e9e38917f43ae20db5"), "person" : 2 }
 { "_id" : ObjectId("508f95e9e38917f43ae20db6"), "person" : 3 }
 { "_id" : ObjectId("508f95e9e38917f43ae20db7"), "person" : 4 }
 { "_id" : ObjectId("508f95e9e38917f43ae20db8"), "person" : 5 }
 { "_id" : ObjectId("508f95e9e38917f43ae20db9"), "person" : 6 }
 { "_id" : ObjectId("508f95e9e38917f43ae20dba"), "person" : 7 }
 { "_id" : ObjectId("508f95e9e38917f43ae20dbb"), "person" : 8 }
 { "_id" : ObjectId("508f95e9e38917f43ae20dbc"), "person" : 9 }
 { "_id" : ObjectId("508f95e9e38917f43ae20dbd"), "person" : 10 }
 { "_id" : ObjectId("508f95e9e38917f43ae20dbe"), "person" : 11 }
 { "_id" : ObjectId("508f95e9e38917f43ae20dbf"), "person" : 12 }
 { "_id" : ObjectId("508f95e9e38917f43ae20dc0"), "person" : 13 }
 { "_id" : ObjectId("508f95e9e38917f43ae20dc1"), "person" : 14 }
 { "_id" : ObjectId("508f95e9e38917f43ae20dc2"), "person" : 15 }
 { "_id" : ObjectId("508f95e9e38917f43ae20dc3"), "person" : 16 }
 { "_id" : ObjectId("508f95e9e38917f43ae20dc4"), "person" : 17 }
 { "_id" : ObjectId("508f95e9e38917f43ae20dc5"), "person" : 18 }
 { "_id" : ObjectId("508f95e9e38917f43ae20dc6"), "person" : 19 }
 Type "it" for more
 person:SECONDARY> db.person.count()
 194079
 person:SECONDARY> db.person.count()
 215657
 person:SECONDARY> db.person.count()
 228488
 person:SECONDARY> db.person.count()
 239528
 person:SECONDARY>
```

Works!!!