
mongoDB.
FOR GIANT IDEAS

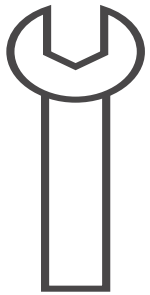
Welcome to MongoDB

WOMEN™
IN LINUX

MongoDB Introduction

MongoDB

GENERAL PURPOSE



DOCUMENT DATABASE

OPEN-SOURCE

Document Data Model

Relational

PERSON

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rome

CAR

Car_ID	Model	Year	Value	Pers_ID
101	Bently	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

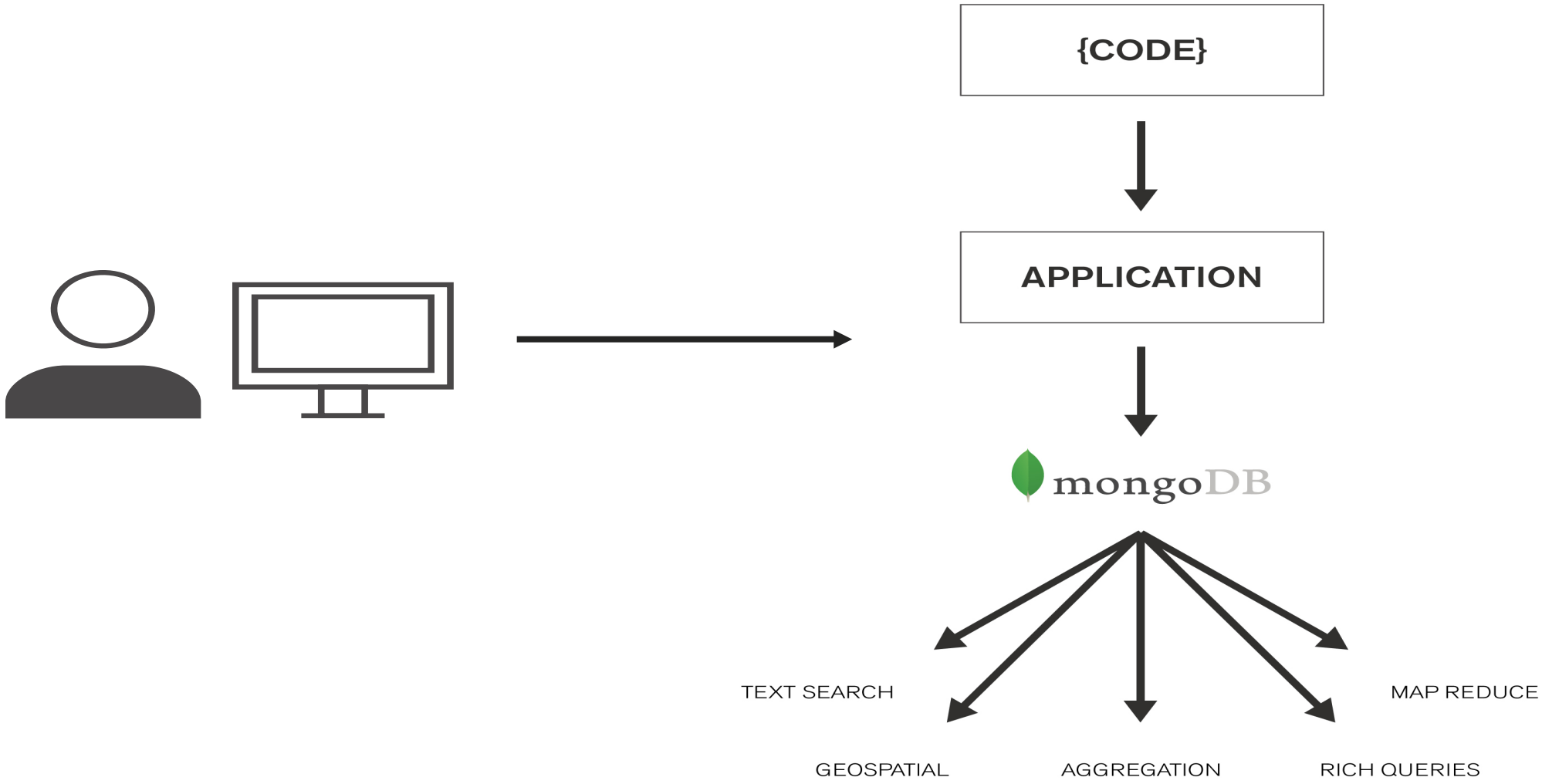
MongoDB

```
{
  first_name: ‘Paul’,
  surname: ‘Miller’,
  city: ‘London’,
  location: [45.123,47.232],
  cars: [
    { model: ‘Bentley’,
      year: 1973,
      value: 100000, … },
    { model: ‘Rolls Royce’,
      year: 1965,
      value: 330000, … }
  ]
}
```

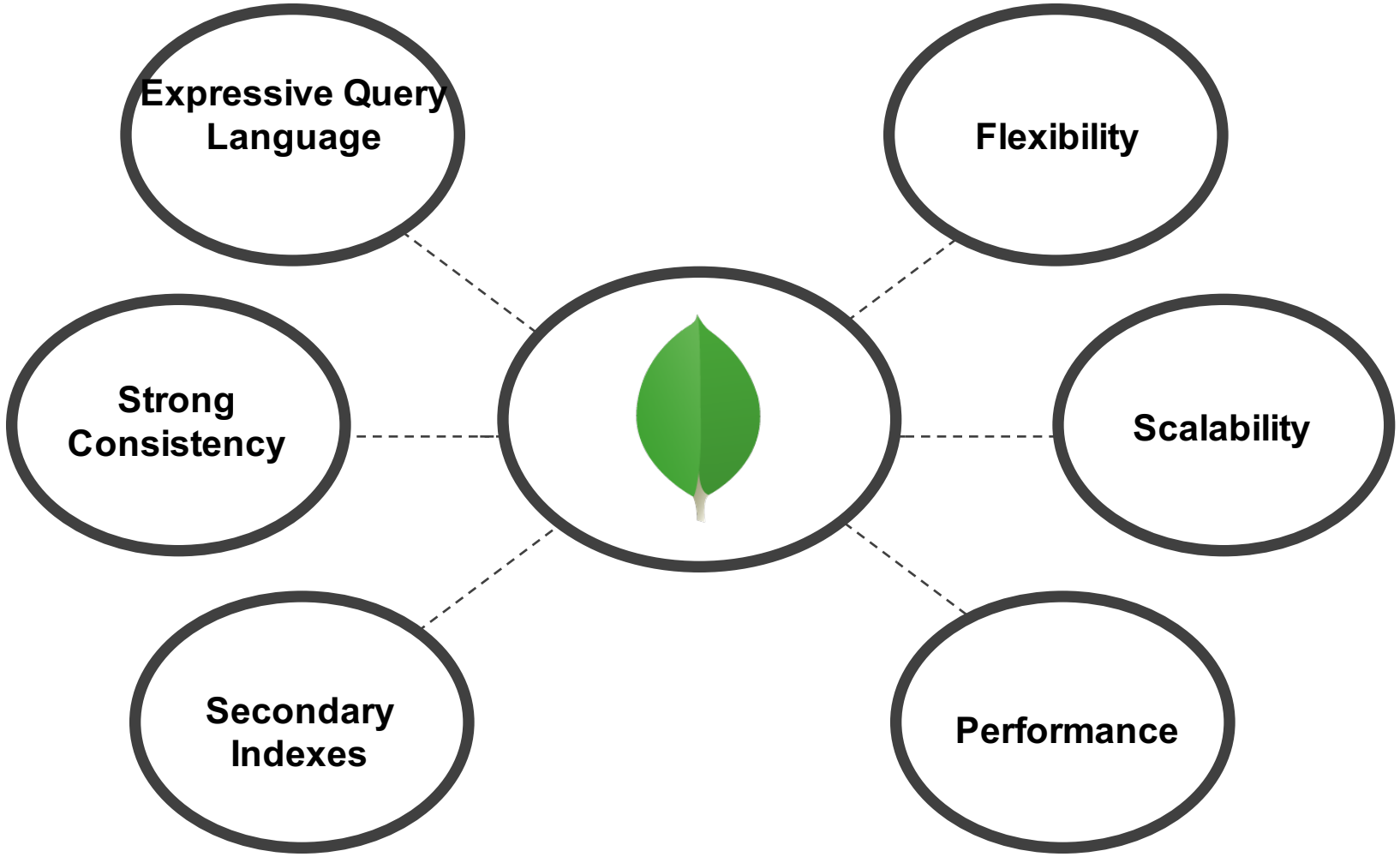
Document Model Benefits

- Agility and flexibility
  - Data models can evolve easily
  - Companies can adapt to changes quickly
- Intuitive, natural data representation
  - Developers are more productive
  - Many types of applications are a good fit
- Reduces the need for joins, disk seeks
  - Programming is more simple
  - Performance can be delivered at scale

MongoDB is Fully Featured



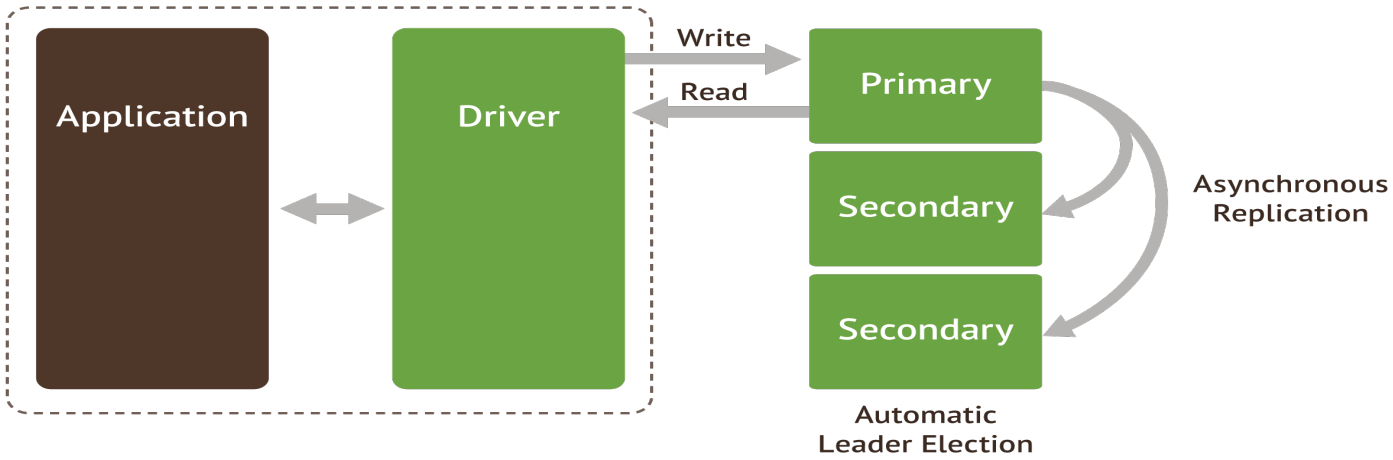
MongoDB Architecture



Relational + NoSQL

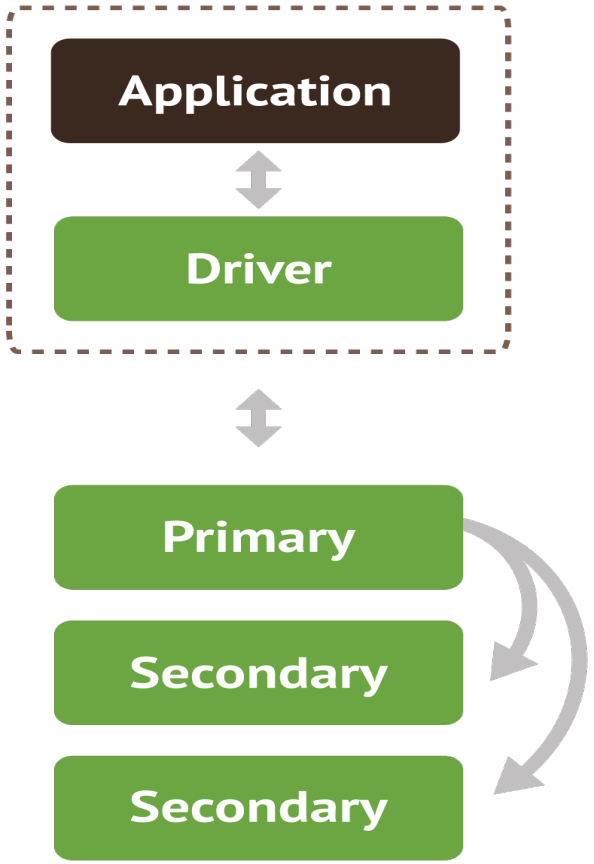
mongoDB.

High Availability



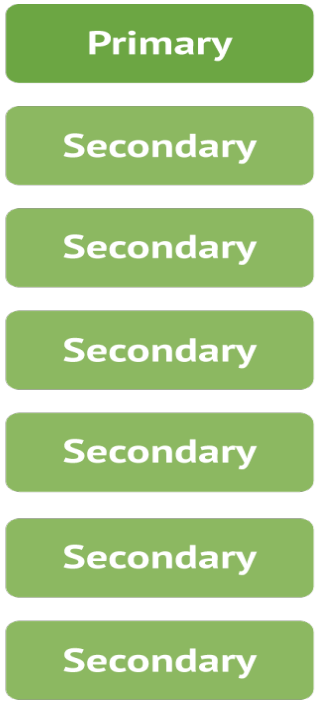
- Automated replication and failover
- Multi-data center support
- Improved operational simplicity (e.g., HW swaps)
- Data durability and consistency

Replica Sets



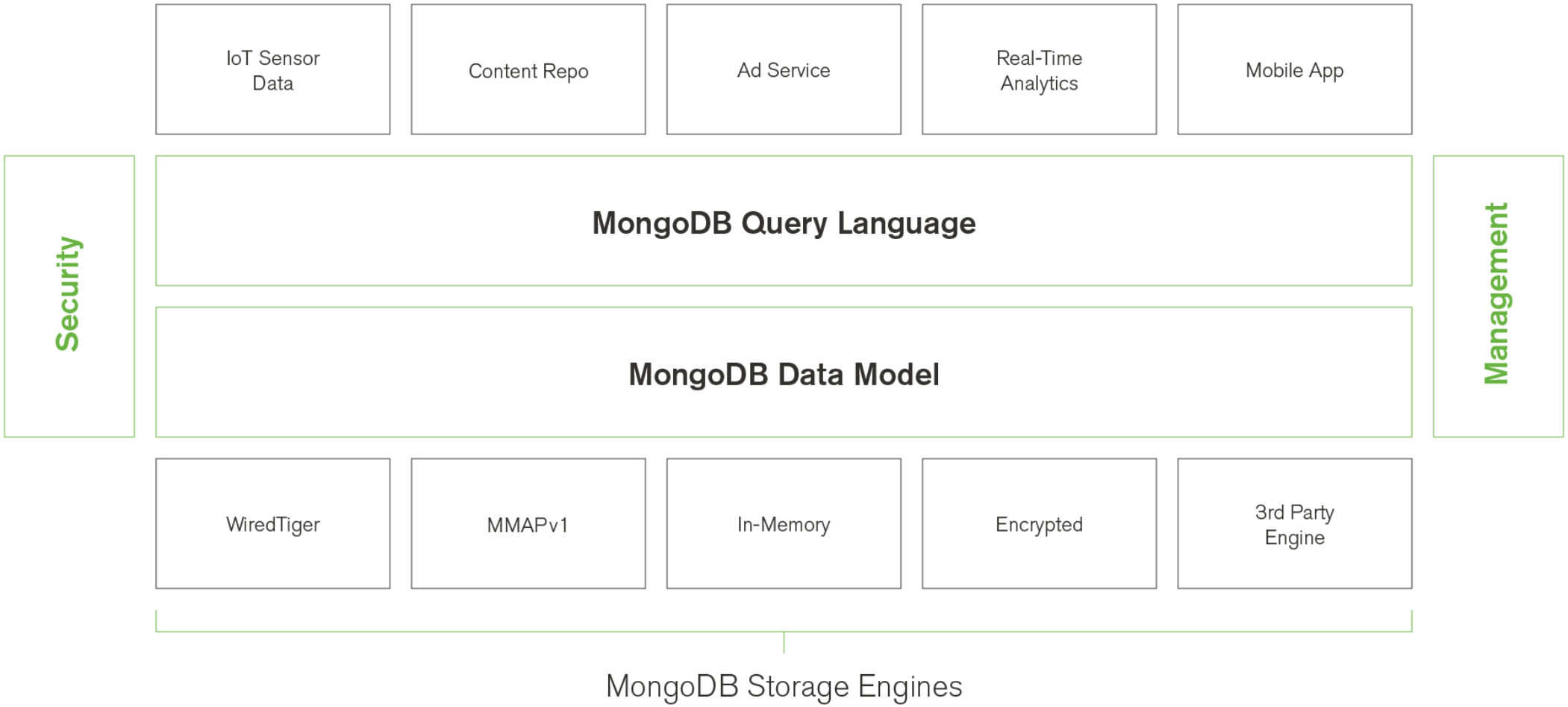
- Replica Set – two or more copies
- “Self-healing” shard
- Addresses many concerns:
  - High Availability
  - Disaster Recovery
  - Maintenance

Replica Set Benefits



Business Needs	Replica Set Benefits
High Availability	Automated failover
Disaster Recovery	Hot backups offsite
Maintenance	Rolling upgrades
Low Latency	Locate data near users
Workload Isolation	Read from non-primary replicas
Data Privacy	Restrict data to physical location
Data Consistency	Tunable Consistency

Flexible Storage Architecture



Getting Started

http://university.mongodb.com

What we are about to see

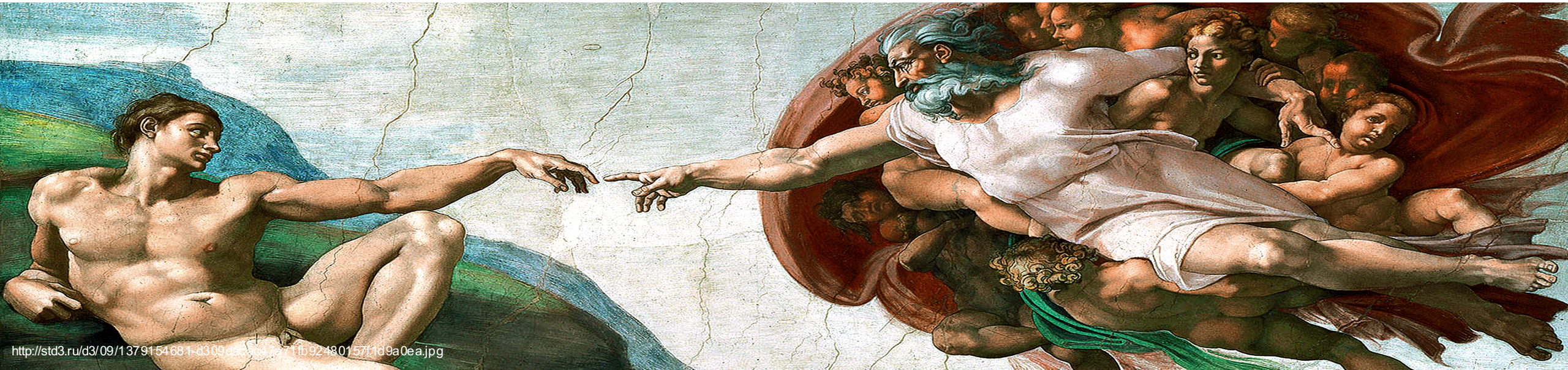
- Install MongoDB on Linux VM
- Bring up a service
- Do our first shell connection
- Perform some CRUD operations
- Administration Commands

First App

Getting Started

- Review our documentation
  - http://docs.mongodb.org/ecosystem/drivers/java/

Connecting



Connecting

```
public void connect() throws UnknownHostException{

    String host = "mongodb://localhost:27017";
    //connect to host
    MongoClient mongoClient = new MongoClient(host);
...
}
```

Connecting

```
public void connect() throws UnknownHostException{

    String host = "mongodb://localhost:27017";
    //connect to host
    MongoClient mongoClient = new MongoClient(host);
...
}
```

Host Machine

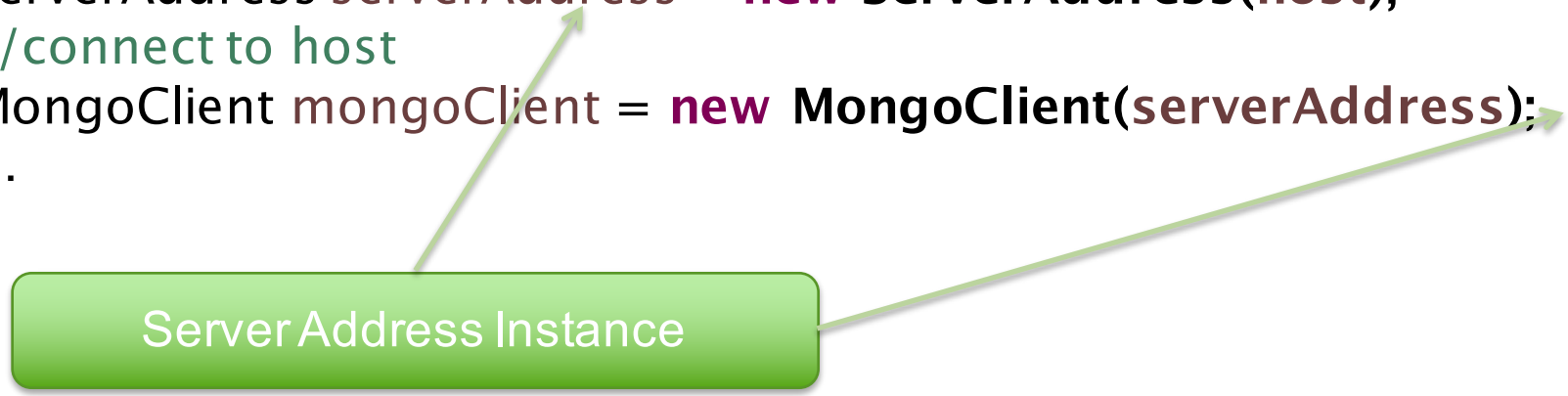
Client Instance

Connecting

```
public void connectServerAddress() throws UnknownHostException{

    String host = "mongodb://localhost:27017";
    //using ServerAddress
    ServerAddress serverAddress = new ServerAddress(host);
    //connect to host
    MongoClient mongoClient = new MongoClient(serverAddress);
    …
}
```

Connecting

```
public void connectServerAddress() throws UnknownHostException{

    String host = "mongodb://localhost:27017";
    //using ServerAddress
    ServerAddress serverAddress = new ServerAddress(host);
    //connect to host
    MongoClient mongoClient = new MongoClient(serverAddress);
    …
}
```

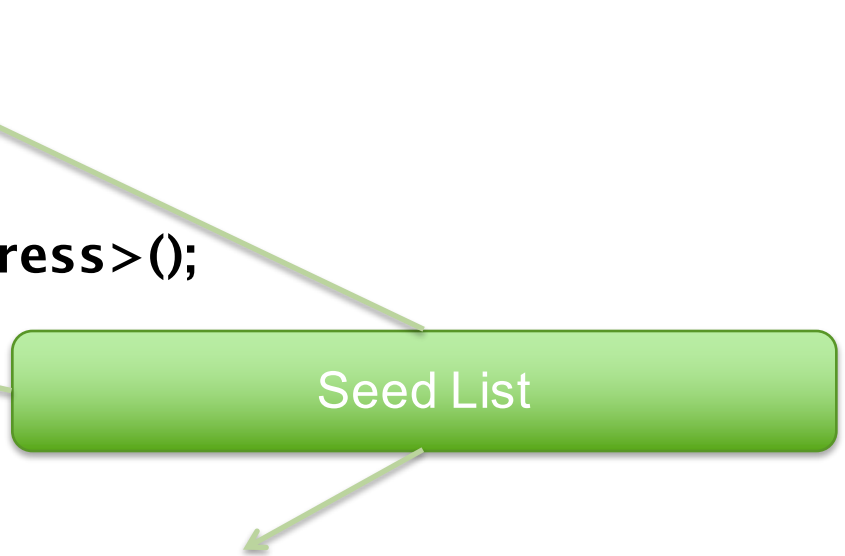


Connecting

```
public boolean connectReplicas() throws UnknownHostException{
    //replica set members
     String []hosts = {
            "mongodb://localhost:30000",
            "mongodb://localhost:30001",
            "mongodb://localhost:30000",
     };
     //list of ServerAdress
     List<ServerAddress> seeds = new ArrayList<ServerAddress>();

     for (String h: hosts){
        seeds.add(new ServerAddress(h));
     }
     //connect to host
     MongoClient mongoClient = new MongoClient(seeds);
...
}
```

Connecting

```
public boolean connectReplicas() throws UnknownHostException{
    //replica set members
     String []hosts = {
            "mongodb://localhost:30000",
            "mongodb://localhost:30001",
            "mongodb://localhost:30000",
     };
     //list of ServerAdress
     List<ServerAddress> seeds = new ArrayList<ServerAddress>();

     for (String h: hosts){
        seeds.add(new ServerAddress(h));
     }
     //connect to host
     MongoClient mongoClient = new MongoClient(seeds);
...
}
```



Database & Collections

```
public void connectDatabase(final String dbname, final String collname){
    //database instance
    DB database =  mongoClient.getDB(dbname);
    //collection instance
    DBCollection collection = database.getCollection(collname);
```

https://api.mongodb.org/java/current/com/mongodb/DB.html

https://api.mongodb.org/java/current/com/mongodb/DBCollection.html

Security

```
public void connectServerAddress() throws UnknownHostException{
        String host = "localhost:27017";
        ServerAddress serverAddress = new ServerAddress(host);

        String dbname = "test";
        String userName = "norberto";
        char[] password = {'a', 'b', 'c'};

        MongoCredential credential = MongoCredential
.createMongoCRCredential(userName, dbname, password);
        MongoClient mongoClient = new MongoClient(serverAddress, Arrays.asList(credential));
```

http://docs.mongodb.org/ecosystem/tutorial/getting-started-with-java-driver/#authentication-optional

Writing


http://std3.ru/d3/09/1379154681-d309dacac47a71fb92480157f1d9a0ea.jpg

Inserting

```
public boolean insertObject(SimplePojo p){
        //lets insert on database `test`
        String dbname = "test";
        //on collection `simple`
        String collname = "simple";
        DBCollection collection = mc.getDB(dbname).getCollection(collname);

        DBObject document = new BasicDBObject();
        document.put("name", p.getName());
        document.put("age", p.getAge());
        document.put("address", p.getAddress());
        //db.simple.insert( { "name": XYZ, "age": 45, "address": "49 Rue de Rivoli, 75001 Paris"})
        collection.insert(document);…
}
```

Inserting

```
public boolean insertObject(SimplePojo p){
        //lets insert on database `test`
        String dbname = "test";
        //on collection `simple`
        String collname = "simple";
        DBCollection collection = mc.getDB(dbname).getCollection(collname);

        DBObject document = new BasicDBObject();
        document.put("name", p.getName());
        document.put("age", p.getAge());
        document.put("address", p.getAddress());
//db.simple.insert( { "name": XYZ, "age": 45, "address": "49 Rue de Rivoli, 75001 Paris"})
        collection.insert(document);
…
}
```

Collection instance

BSON wrapper

Inserting

```
public boolean insertObject(SimplePojo p){
    …
     WriteResult result = collection.insert(document);
     //log number of inserted documents
     log.debug("Number of inserted results " + result.getN() );
    …
}
```

https://api.mongodb.org/java/current/com/mongodb/DBObject.html

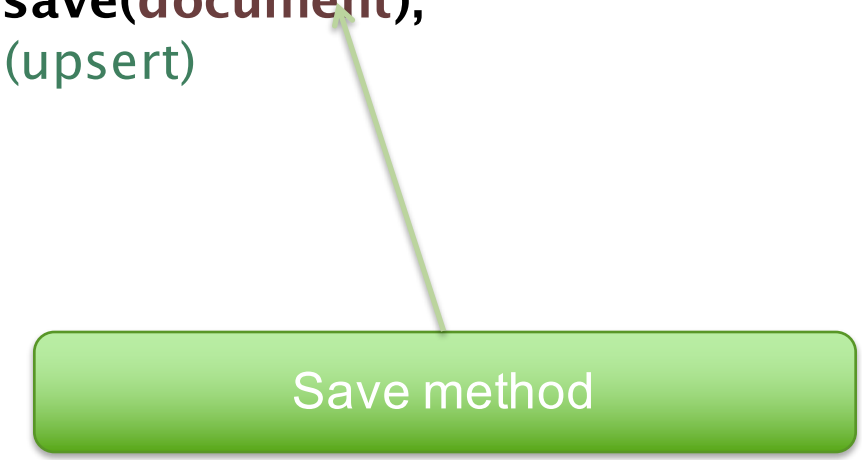
https://api.mongodb.org/java/current/com/mongodb/WriteResult.html

Update

```
public long savePojo(SimplePojo p) {

    DBObject document = BasicDBObjectBuilder.start()
            .add("name", p.getName())
            .add("age", p.getAge())
            .add("address", p.getAddress()).get();

    //method replaces the existing database document by this new one
    WriteResult res = this.collection.save(document);
    //if it does not exist will create one (upsert)

    return res.getN();
}
```

Update

```
public long savePojo(SimplePojo p) {

    DBObject document = BasicDBObjectBuilder.start()
            .add("name", p.getName())
            .add("age", p.getAge())
            .add("address", p.getAddress()).get();

    //method replaces the existing database document by this new one
    WriteResult res = this.collection.save(document);
    //if it does not exist creates one (upsert)

    return res.getN();
}
```



Update

```
public long updateAddress( SimplePojo p, Address a ){
        //{ 'name': XYZ}
        DBObject query = QueryBuilder.start("name").is(p.getName()).get();

        DBObject address = BasicDBObjectBuilder.start()
                .add("street", a.getStreet())
                .add("city", a.getCity())
                .add("number", a.getNumber())
                .add("district", a.getDistrict()).get();

        DBObject update = BasicDBObjectBuilder.start().add("name", p.getName()).add("age",
p.getAge()).add("address", address).get();


        return this.collection.update(query, update).getN();
    }
```

Update

Query Object

```
public long updateAddress( SimplePojo p, Address a ){
        //{ 'name': XYZ}
        DBObject query = QueryBuilder.start("name").is(p.getName()).get();

        DBObject address = BasicDBObjectBuilder.start()
                .add("street", a.getStreet())
                .add("city", a.getCity())
                .add("number", a.getNumber())
                .add("district", a.getDistrict()).get();

        DBObject update = BasicDBObjectBuilder.start().add("name", p.getName()).add("age",
p.getAge()).add("address", address).get();


        return this.collection.update(query, update).getN();
    }
```

Update Object

Update

```
public long updateSetAddress( SimplePojo p, Address a ){
    //{ 'name': XYZ}
    DBObject query = QueryBuilder.start("name").is(p.getName()).get();

    DBObject address = BasicDBObjectBuilder.start()
            .add("street", a.getStreet())
            .add("city", a.getCity())
            .add("number", a.getNumber())
            .add("district", a.getDistrict()).get();
    //db.simple.update( {"name": XYZ}, {"$set":{ "address": ...} } )
    DBObject update =   BasicDBObjectBuilder.start()
            .append("$set", new BasicDBObject("address", address) ).get();

    return this.collection.update(query, update).getN();
}
```

Update

```
public long updateSetAddress( SimplePojo p, Address a ){
    //{ 'name': XYZ}
    DBObject query = QueryBuilder.start("name").is(p.getName()).get();

    DBObject address = BasicDBObjectBuilder.start()
            .add("street", a.getStreet())
            .add("city", a.getCity())
            .add("number", a.getNumber())
            .add("district", a.getDistrict()).get();
    //db.simple.update( {"name": XYZ}, {"$set":{ "address": ...} } )
    DBObject update =   BasicDBObjectBuilder.start()
            .append("$set", new BasicDBObject("address", address) ).get();

    return this.collection.update(query, update).getN();
}
```

Using $set operator

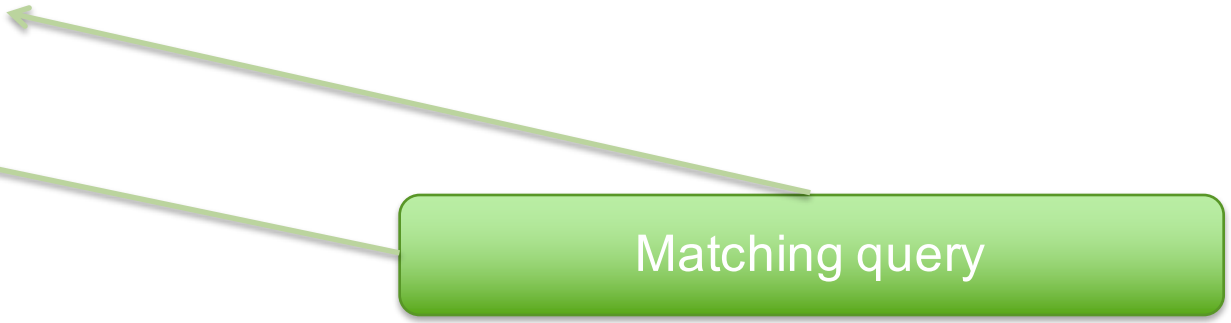
http://docs.mongodb.org/manual/reference/operator/update/

Remove

```
public long removePojo( SimplePojo p){

    //query object to match documents to be removed
    DBObject query = BasicDBObjectBuilder.start().add("name", p.getName()).get();

    return this.collection.remove(query).getN();
}
```

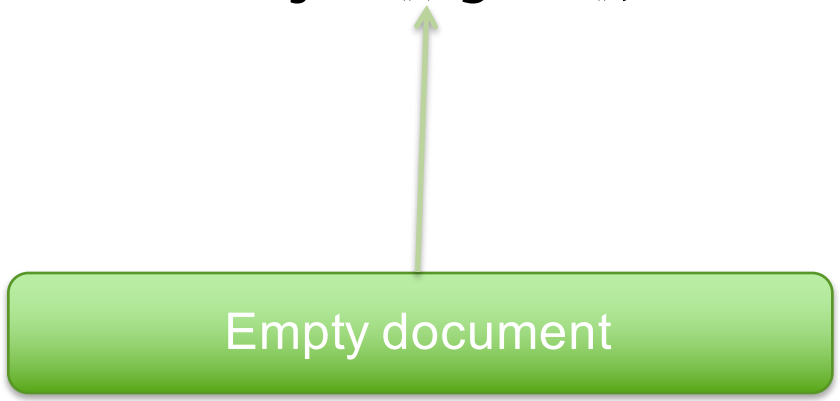
Remove

```
public long removePojo( SimplePojo p){

    //query object to match documents to be removed
    DBObject query = BasicDBObjectBuilder.start()
        .add("name", p.getName()).get();

    return this.collection.remove(query).getN();
}
```



Remove All

```
public long removeAll( ){
      //empty object removes all documents > db.simple.remove({})
      return this.collection.remove(new  BasicDBObject()).getN();

}
```



http://docs.mongodb.org/manual/reference/method/db.collection.remove/

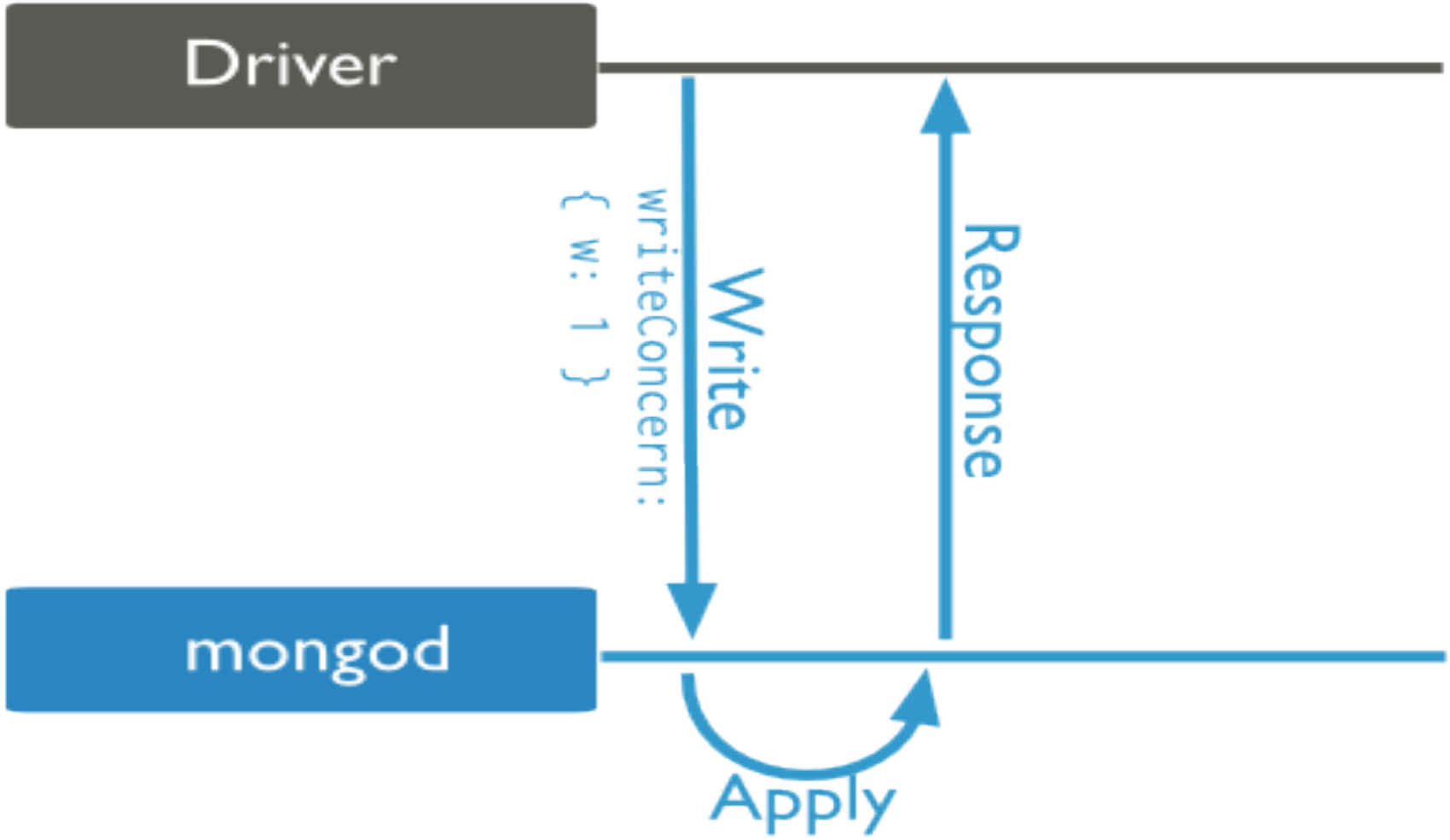
WriteConcern

WriteConcern

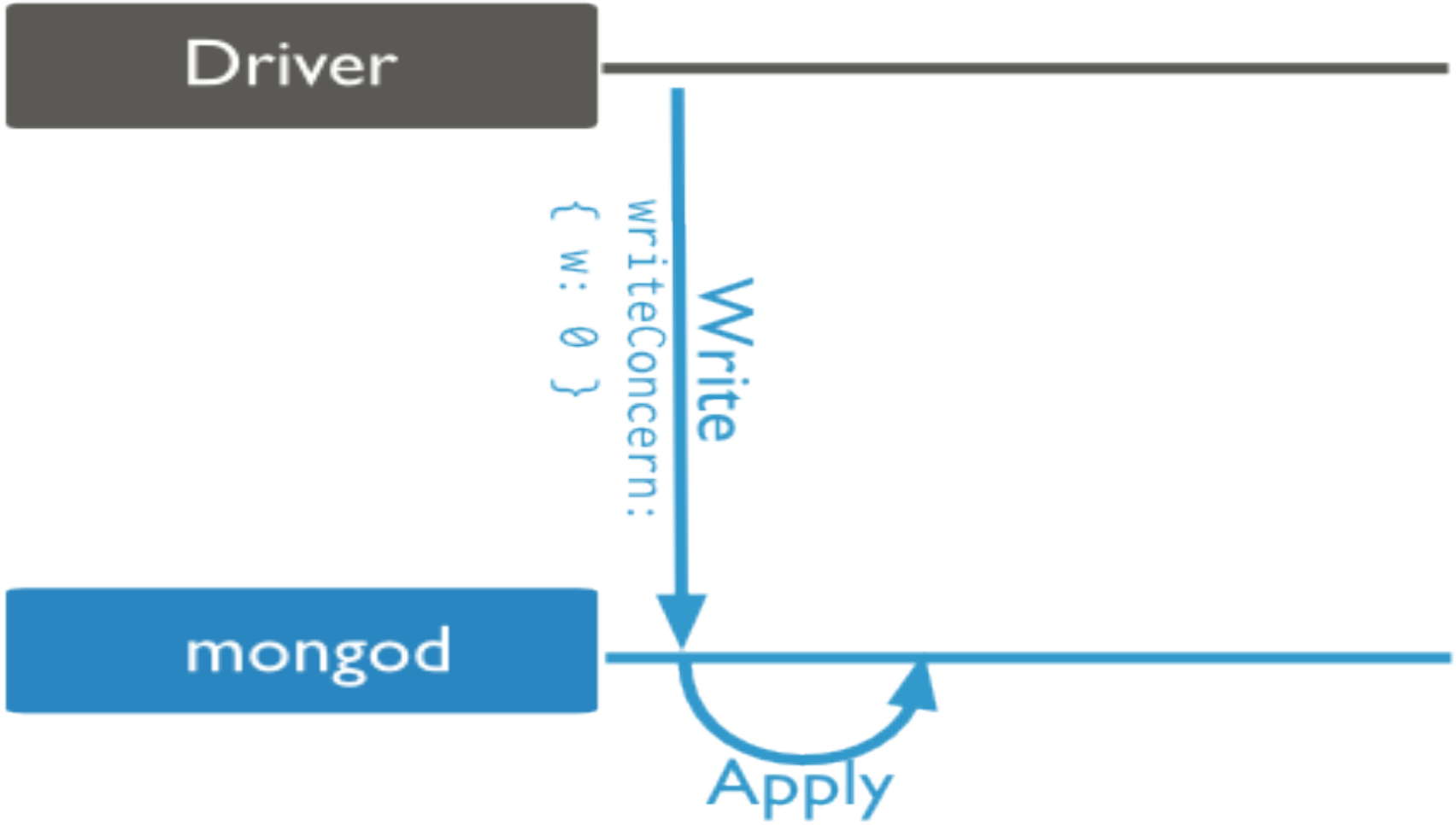
```
//Default
WriteConcern w1 = WriteConcern.ACKNOWLEDGED;

WriteConcern w0 = WriteConcern.UNACKNOWLEDGED;

WriteConcern j1 = WriteConcern.JOURNALED;

WriteConcern wx = WriteConcern.REPLICA_ACKNOWLEDGED;

WriteConcern majority = WriteConcern.MAJORITY;
```

https://api.mongodb.org/java/current/com/mongodb/WriteConcern.html

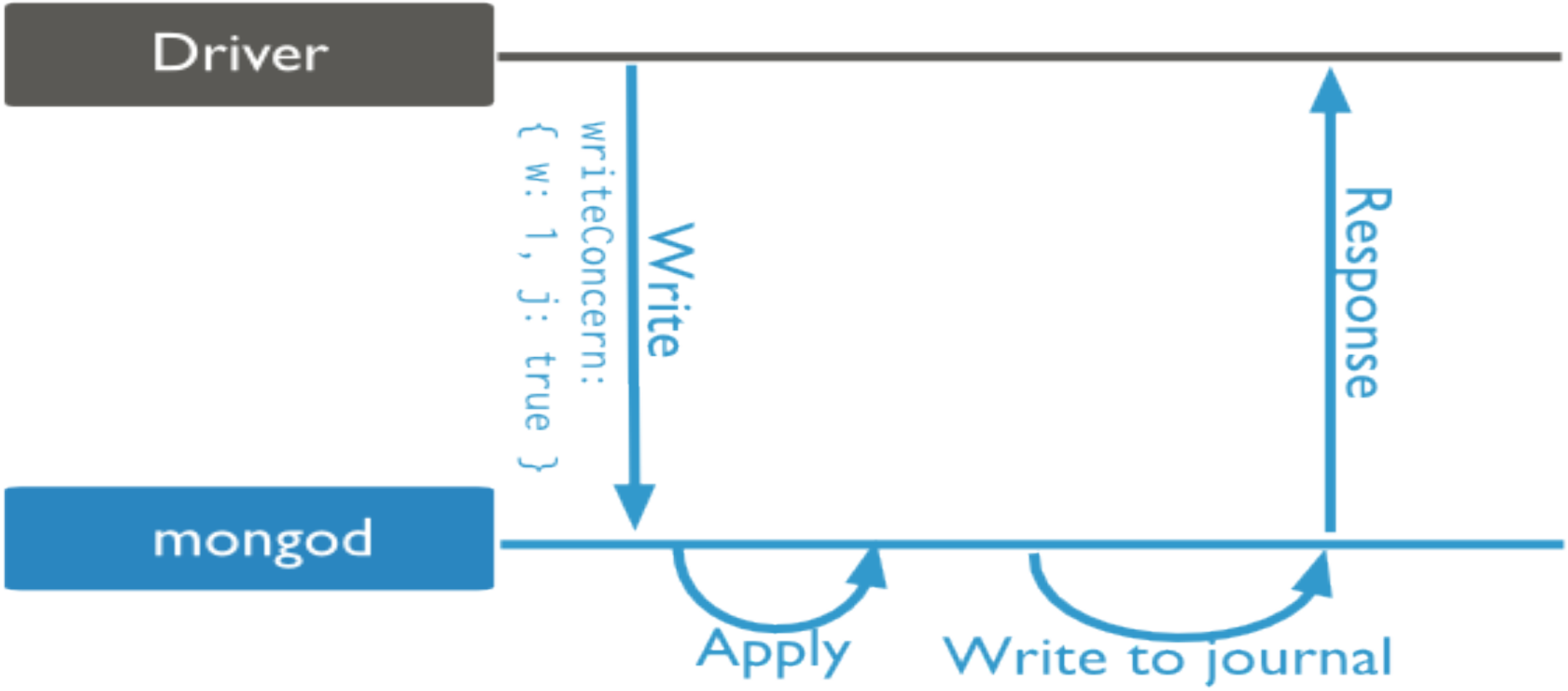
WriteConcern.*ACKNOWLEDGED*



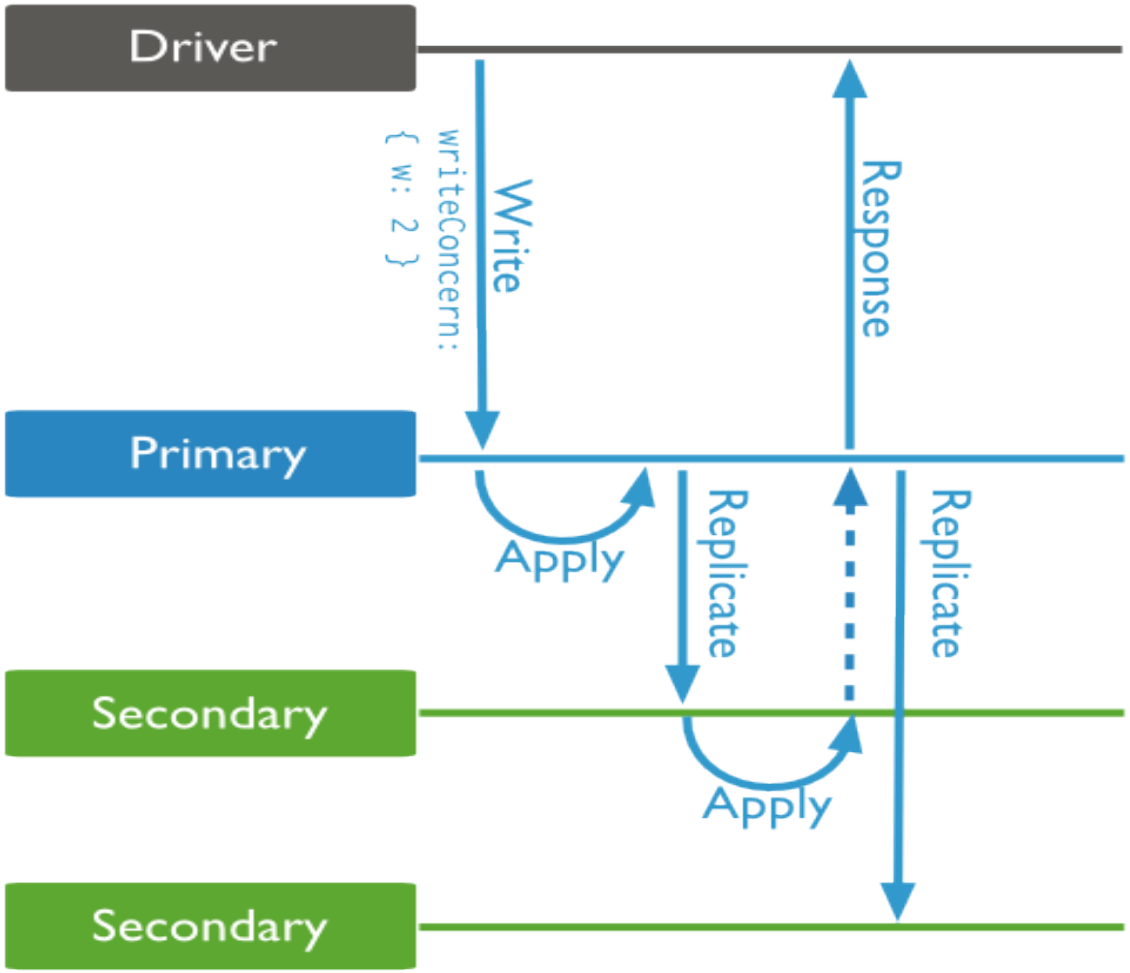
WriteConcern.*UNACKNOWLEDGED*



WriteConcern.*JOURNALED*



WriteConcern.*REPLICA_ACKNOWLEDGED*

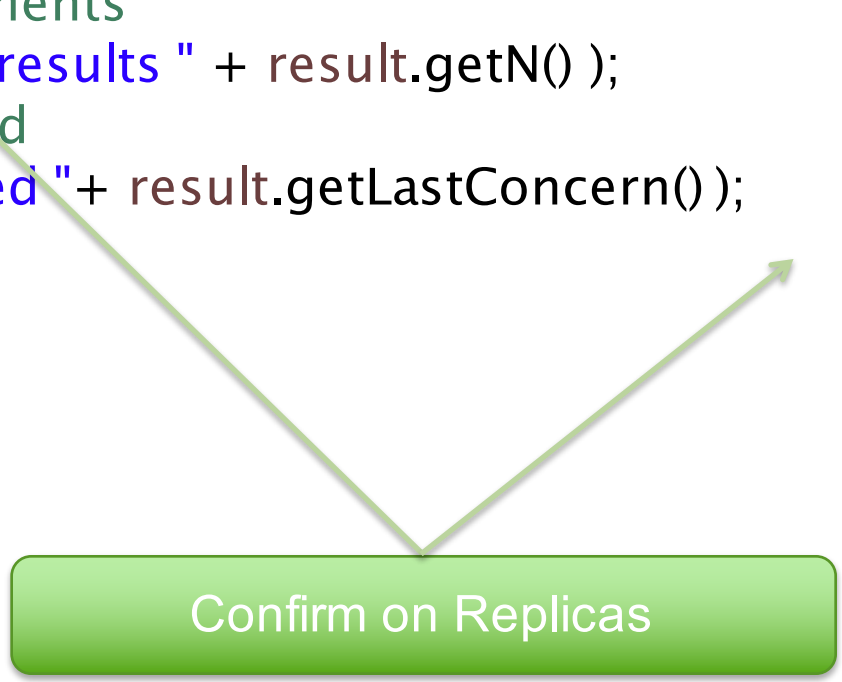


WriteConcern

```
public boolean insertObjectWriteConcern(SimplePojo p){
…
        WriteResult result = collection.insert(document, WriteConcern.REPLICA_ACKNOWLEDGED);
       //log number of inserted documents
       log.debug("Number of inserted results " + result.getN() );
       //log the last write concern used
       log.info( "Last write concern used "+ result.getLastConcern() );…
}
```

WriteConcern

```
public boolean insertObjectWriteConcern(SimplePojo p){
…
        WriteResult result = collection.insert(document, WriteConcern.REPLICA_ACKNOWLEDGED);
       //log number of inserted documents
       log.debug("Number of inserted results " + result.getN() );
       //log the last write concern used
       log.info( "Last write concern used "+ result.getLastConcern() );
…
}
```



WriteConcern

```
public void setWriteConcernLevels(WriteConcern wc){
    //connection level
    this.mc.setWriteConcern(wc);

    //database level
    this.mc.getDB("test").setWriteConcern(wc);

    //collection level
    this.mc.getDB("test").getCollection("simple").setWriteConcern(wc);

    //instruction level
    this.mc.getDB("test").getCollection("simple").insert( new BasicDBObject() , wc);
}
```

Bulk Write

http://images.fineartamerica.com/images-medium-large/2-my-old-bucket-curt-simpson.jpg

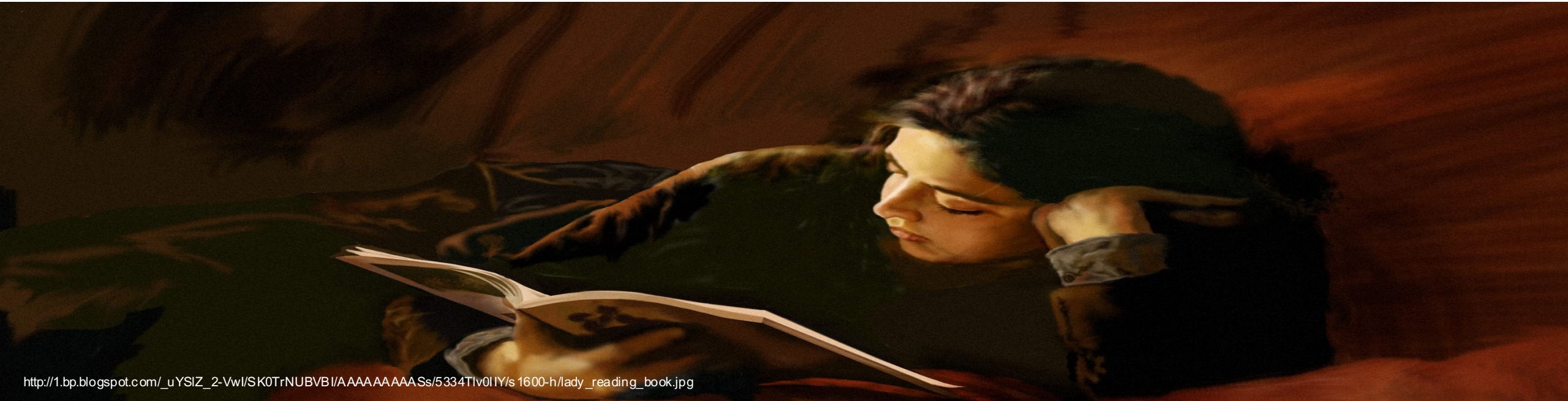
Write Bulk

```
public boolean bulkInsert(List<SimplePojo> ps){
…
 //initialize a bulk write operation
        BulkWriteOperation bulkOp = coll.initializeUnorderedBulkOperation();

        for (SimplePojo p : ps){
            DBObject document = BasicDBObjectBuilder.start()
                    .add("name", p.getName())
                    .add("age", p.getAge())
                    .add("address", p.getAddress()).get();
            bulkOp.insert(document);
        }
        //call the set of inserts
        bulkOp.execute();
…
}
```

Inserting Bulk

```
public void writeSeveral( SimplePojo p, ComplexPojo c){
    //initialize ordered bulk
    BulkWriteOperation bulk = this.collection.initializeOrderedBulkOperation();

    DBObject q1 = BasicDBObjectBuilder.start()
            .add("name", p.getName()).get();
    //remove the previous document
    bulk.find(q1).removeOne();

    //insert the new document
    bulk.insert( c.encodeDBObject() );

     BulkWriteResult result = bulk.execute();

     //do something with the result
     result.getClass();
}
```

Reading



http://1.bp.blogspot.com/_uYSlZ_2-VwI/SK0TrNUBVBI/AAAAAAAAASs/5334TIv0lIY/s1600-h/lady_reading_book.jpg

Read

```
public SimplePojo get(){

    //basic findOne operation

    DBObject doc = this.collection.findOne();
    SimplePojo pojo = new SimplePojo();

    //casting to destination type
    pojo.setAddress((String) doc.get("address"));

    pojo.setAge( (int) doc.get("age") );

    pojo.setAddress( (String) doc.get("address") );

    return pojo;
}
```

Read

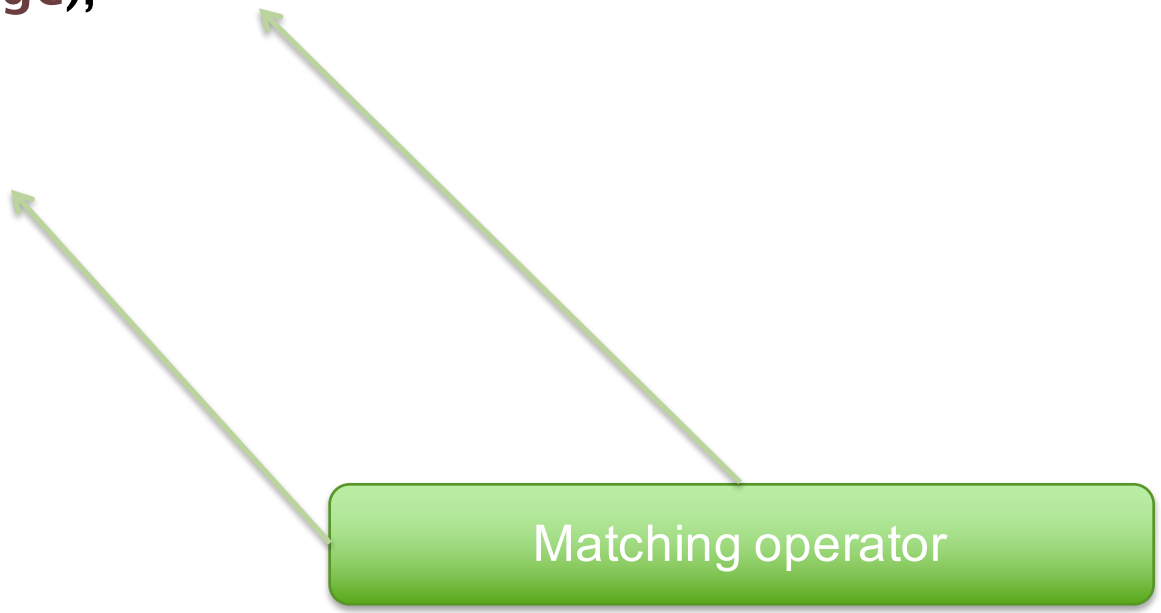
```
public List<SimplePojo> getSeveral(){

    List<SimplePojo> pojos = new ArrayList<SimplePojo>();
    //returns a cursor
    DBCursor cursor = this.collection.find();
    //iterates over the cursor
    for (DBObject document : cursor){
        //calls factory or transformation method
        pojos.add(  this.transform(document) );

    }

    return pojos;
}
```

Read


```
public List<SimplePojo> getSeveral(){

    List<SimplePojo> pojos = new ArrayList<SimplePojo>();
    //returns a cursor
    DBCursor cursor = this.collection.find();
    //iterates over the cursor
    for (DBObject document : cursor){
        //calls factory or transformation method
        pojos.add(  this.transform(document) );

    }

    return pojos;
}
```

Read

```
public List<SimplePojo> getAtAge(int age) {

    List<SimplePojo> pojos = new ArrayList<SimplePojo>();
    DBObject criteria = new BasicDBObject("age", age);

    // matching operator { "age": age}
    DBCursor cursor = this.collection.find(criteria);

    // iterates over the cursor
    for (DBObject document : cursor) {

        // calls factory or transformation method
        pojos.add(this.transform(document));

    }
    return pojos;
}
```

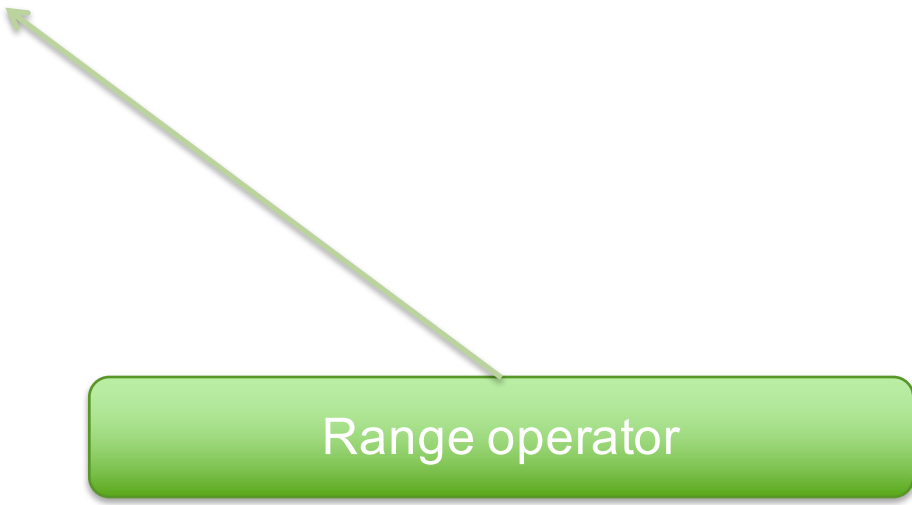


Read

```
public List<SimplePojo> getOlderThan( int minAge){
...
        DBObject criteria =   new BasicDBObject( "$gt",
                new BasicDBObject("age", minAge));

        // matching operator { "$gt": {  "age": minAge} }
        DBCursor cursor = this.collection.find(criteria);
...
}
```

Read

```
public List<SimplePojo> getOlderThan( int minAge){
        …
        DBObject criteria =   new BasicDBObject( "$gt",
                new BasicDBObject("age", minAge));

        // matching operator { "$gt": {  "age": minAge} }
        DBCursor cursor = this.collection.find(criteria);
…
}
```



http://docs.mongodb.org/manual/reference/operator/query/

Read & Filtering

```
public String getAddress(final String name) {

    DBObject criteria = new BasicDBObject("name", name);

    //we want the address field
    DBObject fields = new BasicDBObject( "address", 1 );
    //and exclude _id too
    fields.put("_id", 0);

    // db.simple.find( { "name": name}, {"_id:0", "address":1} )
    DBObject document = this.collection.findOne(criteria, fields);

    //we can check if this is a partial document
    document.isPartialObject();

    return (String) document.get("address");

}
```

Read & Filtering

```
public String getAddress(final String name) {

    DBObject criteria = new BasicDBObject("name", name);

    //we want the address field
    DBObject fields = new BasicDBObject( "address", 1 );
    //and exclude _id too
    fields.put("_id", 0);

    // db.simple.find( { "name": name}, {"_id:0", "address":1} )
    DBObject document = this.collection.findOne(criteria, fields);

    //we can check if this is a partial document
    document.isPartialObject();

    return (String) document.get("address");

}
```

Read Preference

http://2.bp.blogspot.com/-CW3UtTpWoqg/UHhJ0gzjxcI/AAAAAAAAQz0/I_ozSnrwkvo/s1600/Grigori+Semenovich+Sedov+-+Choosing+the+bride+of+Czar+Alexis.jpg

Read Preference

```
//default
ReadPreference.primary();

ReadPreference.primaryPreferred();

ReadPreference.secondary();

ReadPreference.secondaryPreferred();

ReadPreference.nearest();
```

https://api.mongodb.org/java/current/com/mongodb/ReadPreference.html

Read Preference – Tag Aware

```
//read from specific tag
DBObject tag = new BasicDBObject( "datacenter", "Porto" );

ReadPreference readPref = ReadPreference.secondary(tag);
```

http://docs.mongodb.org/manual/tutorial/configure-replica-set-tag-sets/

Read Preference

```
public String getName(  final int age ){

    DBObject criteria = new BasicDBObject("age", age);
    DBObject fields = new BasicDBObject( "name", 1 );
    //set to read from nearest node
    DBObject document = this.collection.findOne(criteria, fields,
  ReadPreference.nearest()  );

    //return the element
    return (String) document.get("name");

}
```

Read Preference

Aggregation

```
// create our pipeline operations, first with the $match
DBObject match = new BasicDBObject("$match", new BasicDBObject("type", "airfare"));

// build the $projection operation
DBObject fields = new BasicDBObject("department", 1);
fields.put("amount", 1);
fields.put("_id", 0);
DBObject project = new BasicDBObject("$project", fields );

// Now the $group operation
DBObject groupFields = new BasicDBObject( "_id", "$department");
groupFields.put("average", new BasicDBObject( "$avg", "$amount"));
DBObject group = new BasicDBObject("$group", groupFields);
```

…

Aggregation

```
// Finally the $sort operation
DBObject sort = new BasicDBObject("$sort", new BasicDBObject("amount", -1));

// run aggregation
List<DBObject> pipeline = Arrays.asList(match, project, group, sort);
AggregationOutput output = coll.aggregate(pipeline);
```

Obrigado!

Norberto Leite
Technical Evangelist
norberto@mongodb.com
@nleite

http://cl.jroo.me/z3/v/D/C/e/a.baa-Too-many-bicycles-on-the-van.jpg


mongoDB®
FOR GIANT IDEAS