#JFokus

What do you mean, Backwards Compatibility?

Trisha Gee, Java Driver Developer

@trisha_gee

#JFokus

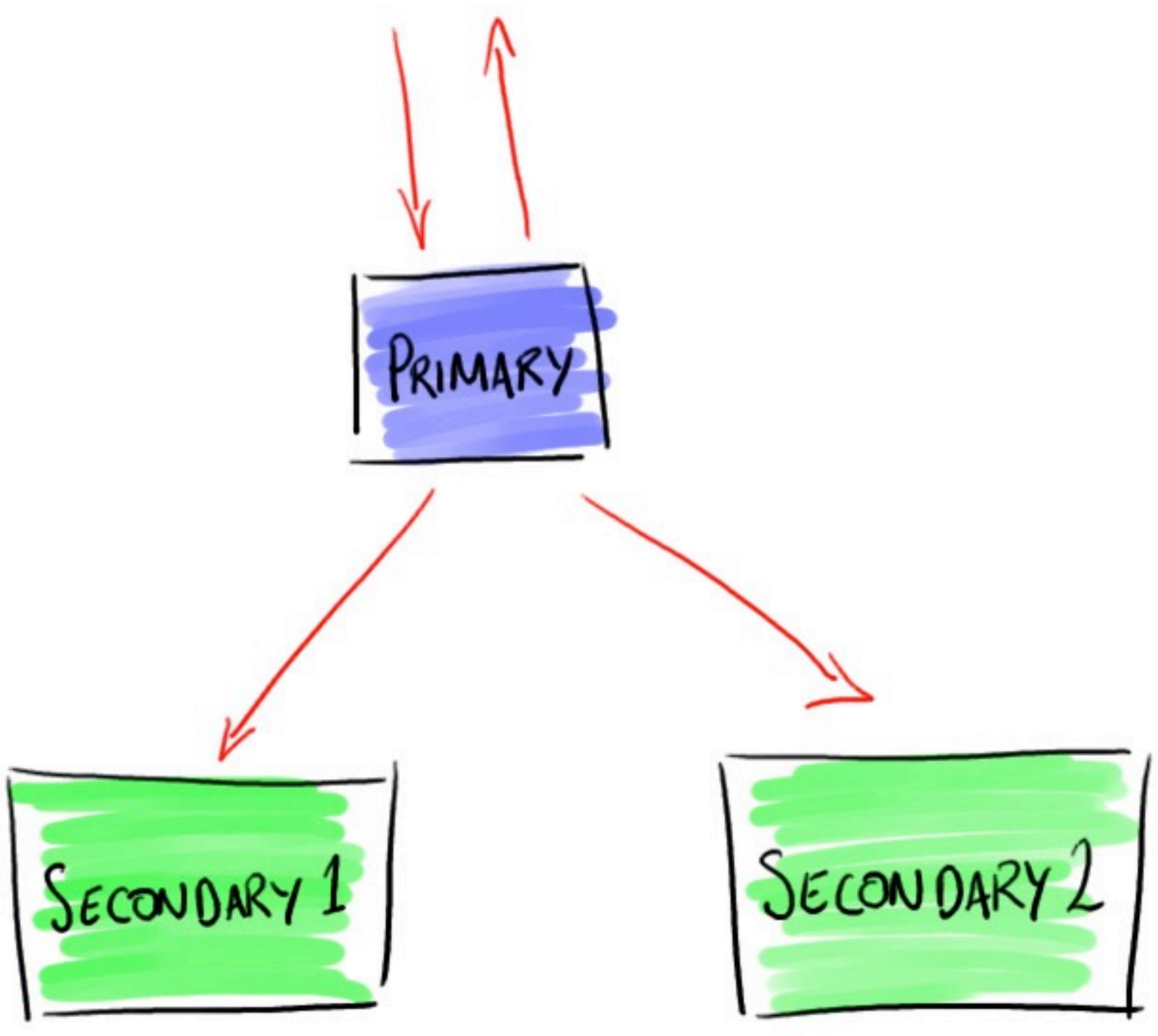
Stuff I Should Have Already Known

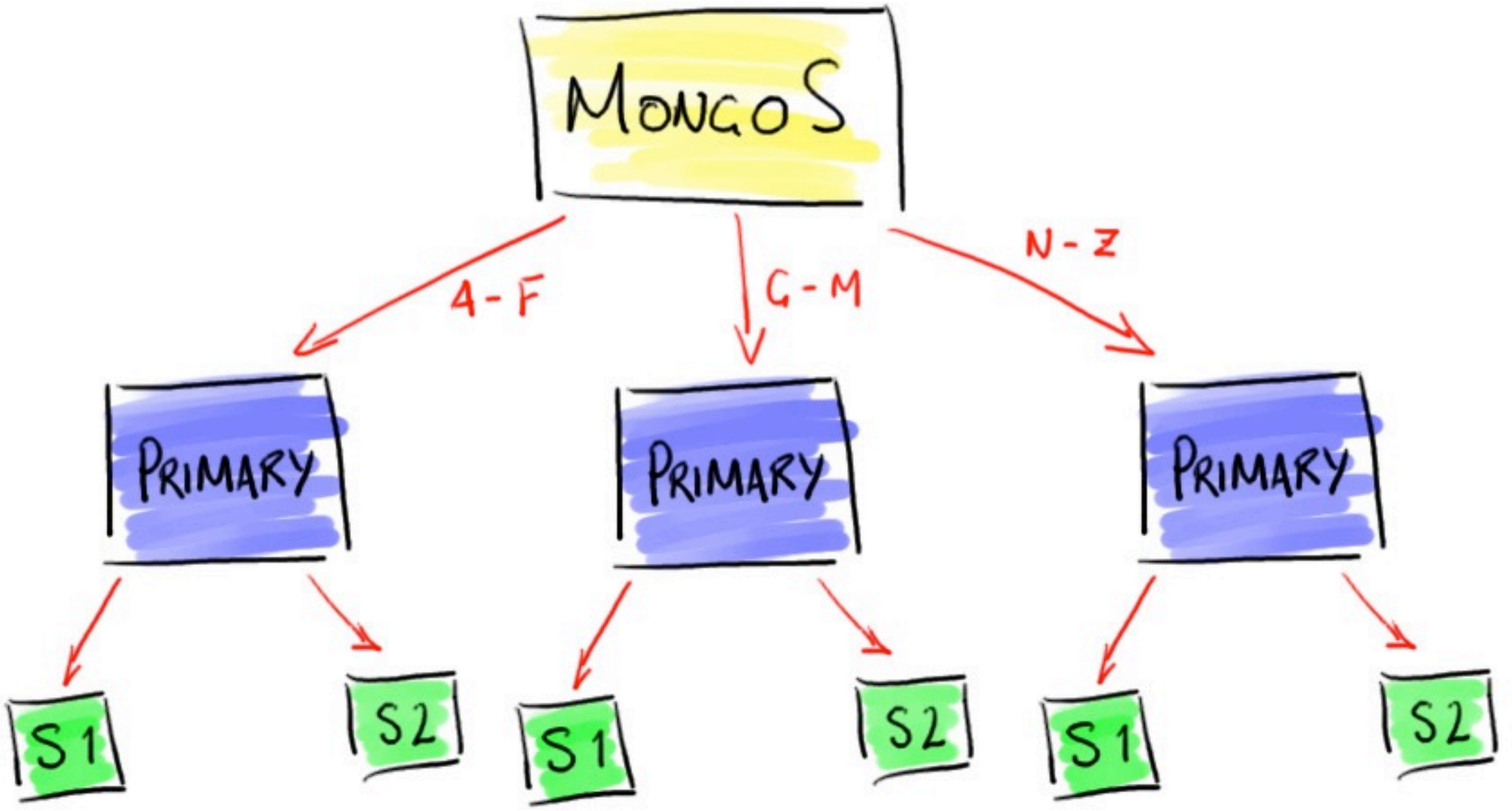
Trisha Gee, Java Driver Developer

@trisha_gee


mongoDB

```
customer = {
    _id: "joe",
    name: "Joe Bookreader",
    address: {
      street: "123 Fake St",
      city: "Faketon",
      state: "MA",
      zip: 12345
    }
    books: [ 27464, 747854, ...]
}
```


PRIMARY
SECONDARY 1
SECONDARY 2


MONGOS
A-F
G-M
N-Z
PRIMARY
PRIMARY
PRIMARY
S1
S2
S1
S2
S1
S2

MongoDB

MongoDB

Dynamic Schema

Scalable

Document Database

NoSQL

Replica Sets

Strong Consistency

Sharding

Schema Design

The shell

MMS

Backup

Map reduce

Aggregation Framework

MongoDB

Scalable

Dynamic Schema

Document Database

NoSQL

Replica Sets

Strong Consistency

Sharding

Schema Design

Python

Ruby

C

MMS

The shell

Perl

Java

Erlang

Backup

C#

PHP

Map reduce

Go

C++

Node.js

Scala

Aggregation Framework

Python

JavaScript

MongoDB

Scalable

Dynamic Schema

Document Database

NoSQL

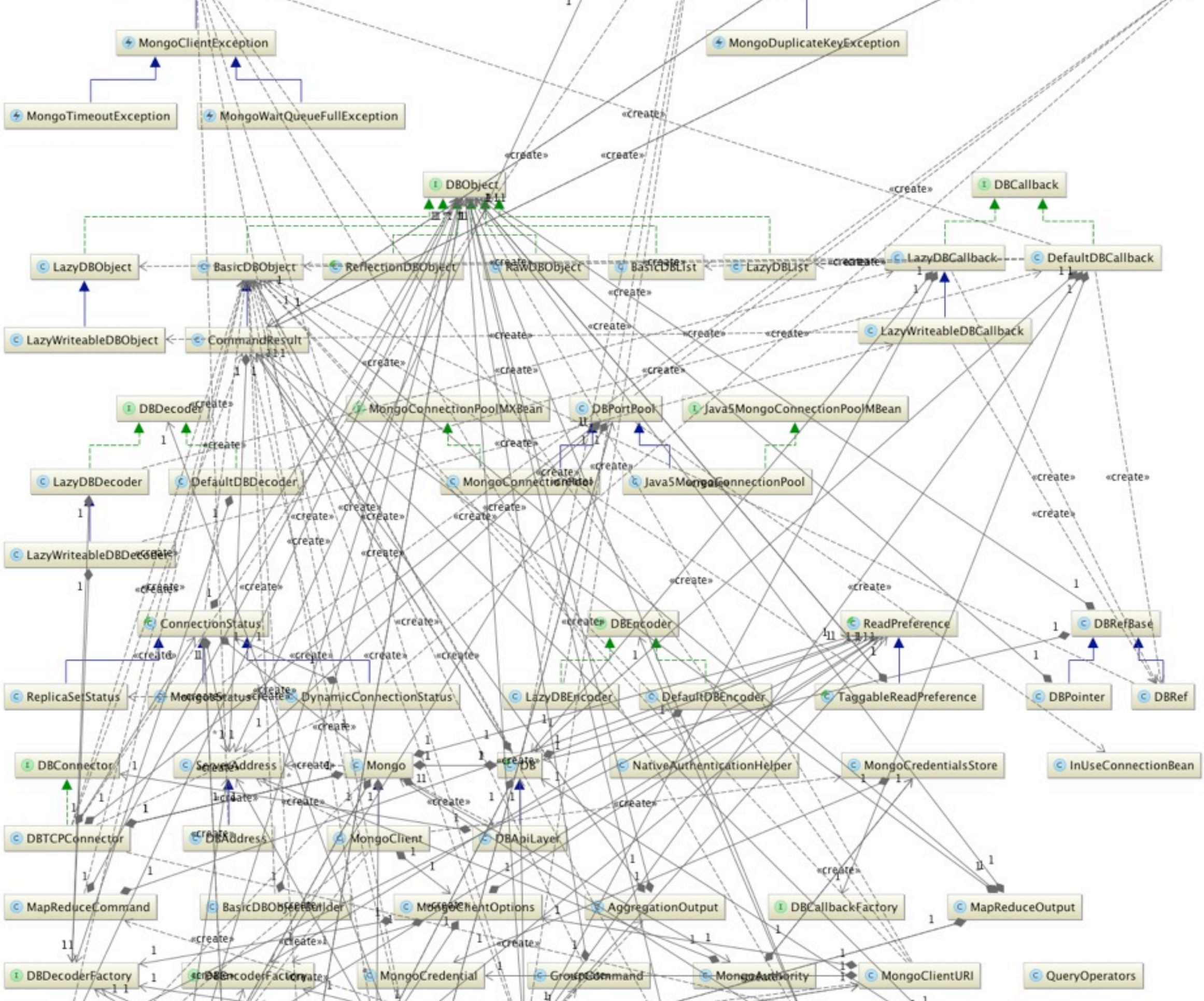
Replica Sets

Strong Consistency

Sharding

Schema Design

The Java Driver


MongoClientException
MongoDuplicateKeyException
MongoTimeoutException
MongoWaitQueueFullException
DBObject
DBCallback
LazyDBObject
BasicDBObject
ReflectionDBObject
RawDBObject
BasicDBList
LazyDBList
LazyDBCallback
DefaultDBCallback
LazyWriteableDBObject
CommandResult
LazyWriteableDBCallback
DBDecoder
MongoConnectionPoolMXBean
DBPortPool
Java5MongoConnectionPoolMBean
LazyDBDecoder
DefaultDBDecoder
MongoConnectionPool
Java5MongoConnectionPool
LazyWriteableDBDecoder
ConnectionStatus
DBEncoder
ReadPreference
DBRefBase
ReplicaSetStatus
MongoStatus
DynamicConnectionStatus
LazyDBEncoder
DefaultDBEncoder
TaggableReadPreference
DBPointer
DBRef
DBConnector
ServerAddress
Mongo
DB
NativeAuthenticationHelper
MongoCredentialsStore
InUseConnectionBean
DBTCPConnector
DBAddress
MongoClient
DBApiLayer
MapReduceCommand
BasicDBObjectBuilder
MongoClientOptions
AggregationOutput
DBCallbackFactory
MapReduceOutput
DBDecoderFactory
DBEncoderFactory
MongoCredential
GroupCommand
MongoAuthority
MongoClientURI
QueryOperators
«create»

```
     * @param coll
     * @param m
     * @param hostNeeded
     * @param readPref
     * @param decoder
     * @return
     * @throws MongoException
     */
    @Override
    public Response call( DB db, DBCollection coll, OutMessage m, ServerAddress hostNeeded, int retries,
                          ReadPreference readPref, DBDecoder decoder ){
        try {
            return innerCall(db, coll, m, hostNeeded, retries, readPref, decoder);
        } finally {
            m.doneWithMessage();
        }
    }

    // This method is recursive.  It calls itself to implement query retry logic.
    private Response innerCall(final DB db, final DBCollection coll, final OutMessage m, final ServerAddress hostNeeded,
                               final int retries, ReadPreference readPref, final DBDecoder decoder) {
        if (readPref == null)
            readPref = ReadPreference.primary();

        if (readPref == ReadPreference.primary() && m.hasOption( Bytes.QUERYOPTION_SLAVEOK ))
            readPref = ReadPreference.secondaryPreferred();

        boolean secondaryOk = !(readPref == ReadPreference.primary());

        _checkClosed();
        // Don't check master on secondary reads unless connected to a replica set
        if (!secondaryOk || getReplicaSetStatus() == null)
            checkMaster( false, !secondaryOk );

        final DBPort port = _myPort.get(false, readPref, hostNeeded);

        Response res = null;
        boolean retry = false;
        try {
            port.checkAuth( db.getMongo() );
            res = port.call( m , coll, decoder );
            if ( res._responseTo != m.getId() )
                throw new MongoException( "ids don't match" );
        }
        catch ( IOException ioe ){
            _myPort.error(port, ioe);
            retry = retries > 0 && !coll._name.equals( "$cmd" )
                    && !(ioe instanceof SocketTimeoutException) && _error( ioe, secondaryOk );
            if ( !retry ){
                throw  new MongoException.Network("Read operation to server " + port.host() + " failed on database " + db , ioe );
            }
        }
        catch ( RuntimeException re ){
            _myPort.error(port, re);
            throw re;
        } finally {
            _myPort.done(port);
        }

        if (retry)
            return innerCall( db , coll , m , hostNeeded , retries - 1 , readPref, decoder );

        ServerError err = res.getError();

        if ( err != null && err.isNotMasterError() ){
            checkMaster( true , true );
            if ( retries <= 0 ){
                throw new MongoException( "not talking to master and retries used up" );
            }
            return innerCall( db , coll , m , hostNeeded , retries -1, readPref, decoder );
        }

        return res;
    }
```

```
collection.find
```

```
find(DBObject ref)                          DBCursor
find()                                      DBCursor
find(DBObject ref, DBObject ke…             DBCursor
find(DBObject query, DBObject …             DBCursor
find(DBObject query, DBObject …             DBCursor
findAndModify(DBObject query, …             DBObject
findOne()                                   DBObject
findAndModify(DBObject query, …             DBObject
findAndRemove(DBObject query)               DBObject
findOne(DBObject o)                         DBObject
Use ⇧⌘⏎ to syntactically correct your code after completing (balance parentheses etc.)
```

> What do you want to do?

> _

> What do you want to do?

> _

Common Problems

Our Solutions

...not unique to us

which is the point

really.

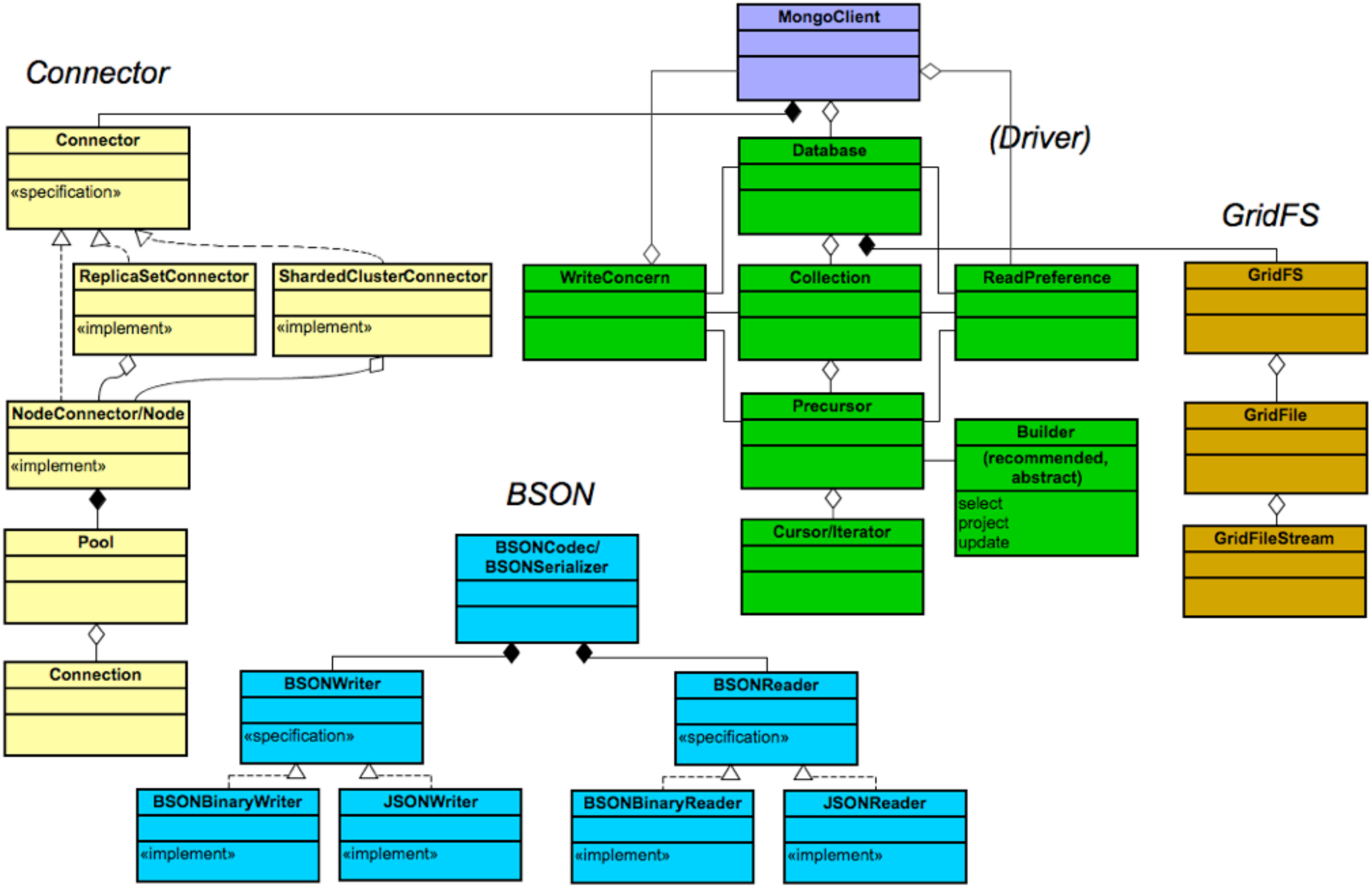
"Let me help you with that"

We're using different words

Or the same words to mean different things

Ubiquitous Language


UML



Documentation

    a. Currently there are no integration tests that cover mongos failover, authentication, replica set failover, etc.
    b. Distinguish between unit tests and integration tests
        i. There's quite a bit now that that can be tested without a running Mongo:
            1. ReadPreference, WriteConcern, QueryBuilder, etc.
9. Judicious use of interfaces
    a. Examples: MongoClient, MongoDatabase, MongoCollection. This will make it easier to create multiple implementations that have no relationship in a class hierarchy, and allow mocking for those who thinks that it's a good idea
10. Exception hierarchy
    a. Example: Distinguish between exceptions generated due to an error from the server and an error on the client. They should be types for each.
11. Prefer immutability
    a. Example: MongoOptions
12. Replace non-constant static fields with either
    a. static methods
        i. ReadPreference.PRIMARY => ReadPreference.primary()
    b. fields of other classes
        i. BSON._encoding/_decodingHooks become fields of MongoClient
13. Prefer SocketChannel to Socket and use direct ByteBuffer.

On the fence/off the table:

1. Async capabilities. The reason to leave it out of 3.0 is just timing. It's going to take longer and will delay the release of 3.0. If we design 3.0 with async in mind, it should be something we can add in later in a minor release.

Fluent API

The fluent api will be built as a set of chainable operations that emanate from a `Collection` and continue in a `CollectionView`. Each operation will either result in the execution of a CRUD operation or will return a `Collection` or a `CollectionView`.

Collection Definition

```
type Collection:

  def find(): CollectionView
    """
    Creates a CollectionView. This is useful for updates or removes that should
    happen across an entire collection or to get the total number
    of documents in a collection.

    Return:
        Returns a CollectionView with no filter.
    """

def find(filter: Document): CollectionView
    """
    Creates a CollectionView with an initial filter.

    Parameters:
        - filter: The initial filter for the CollectionView.

    Return:
        A CollectionView with the initial filter.
    """

def insert(document: Document): WriteResult
    """
    Inserts the document into the collection.  The writeOptions used
    should be taken from the default settings for the collection.
```

Your API is your UI

Naming is Hard

This takes ages...

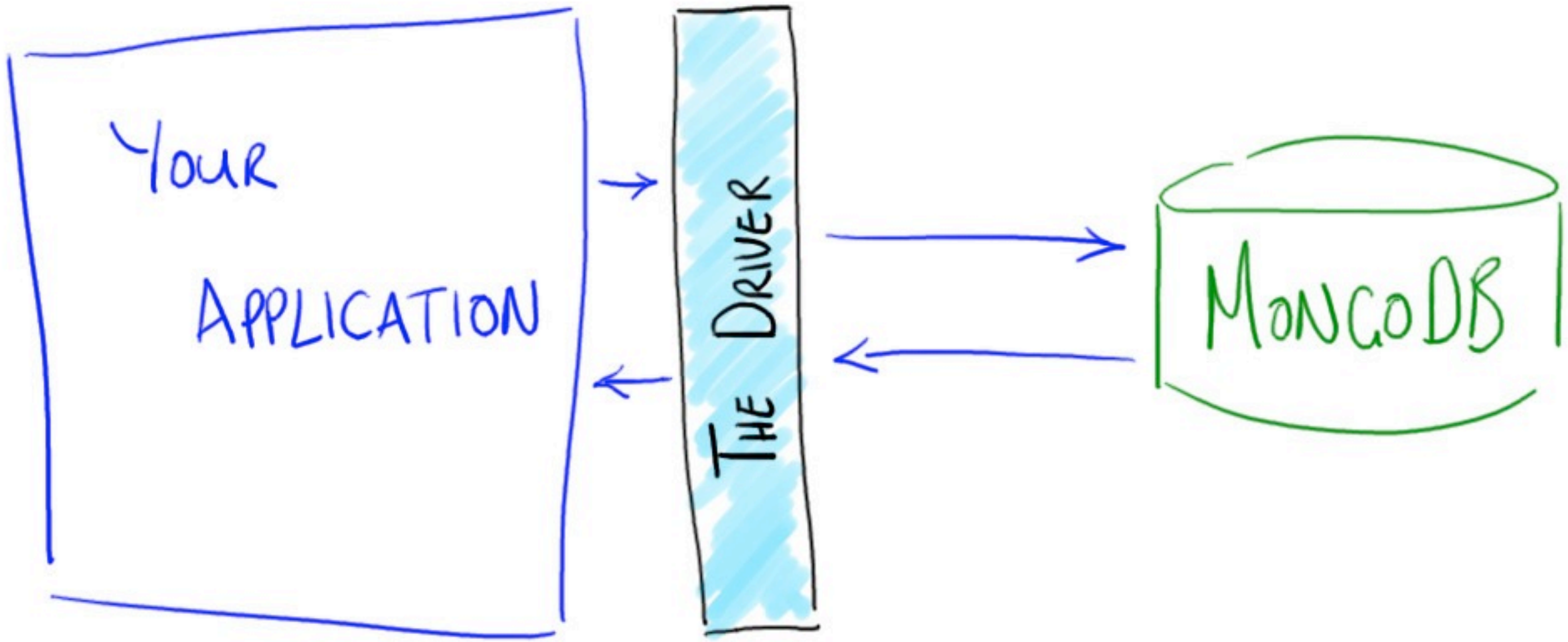
DO THIS FIRST

The code is hard to debug and modify

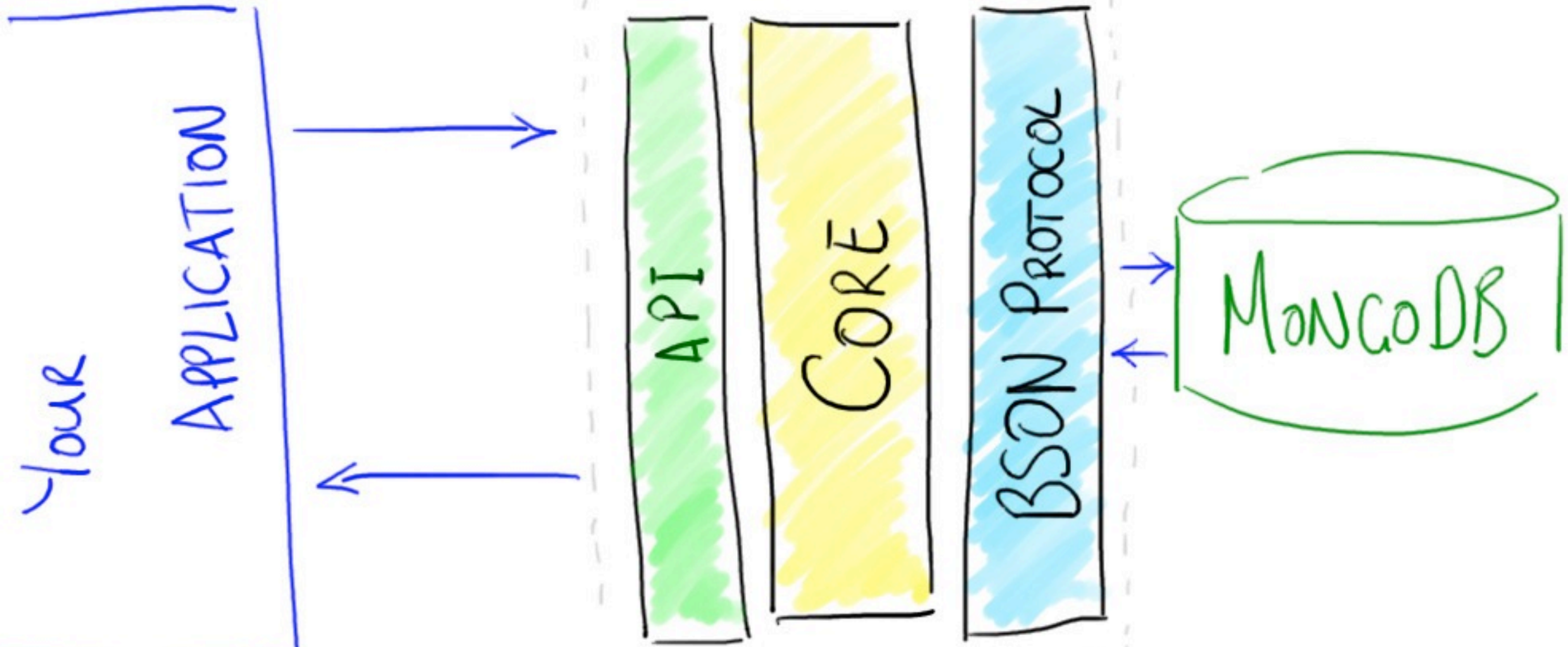
Re-write the whole driver

~~Re-write~~ the whole driver

Anti Corruption Layer

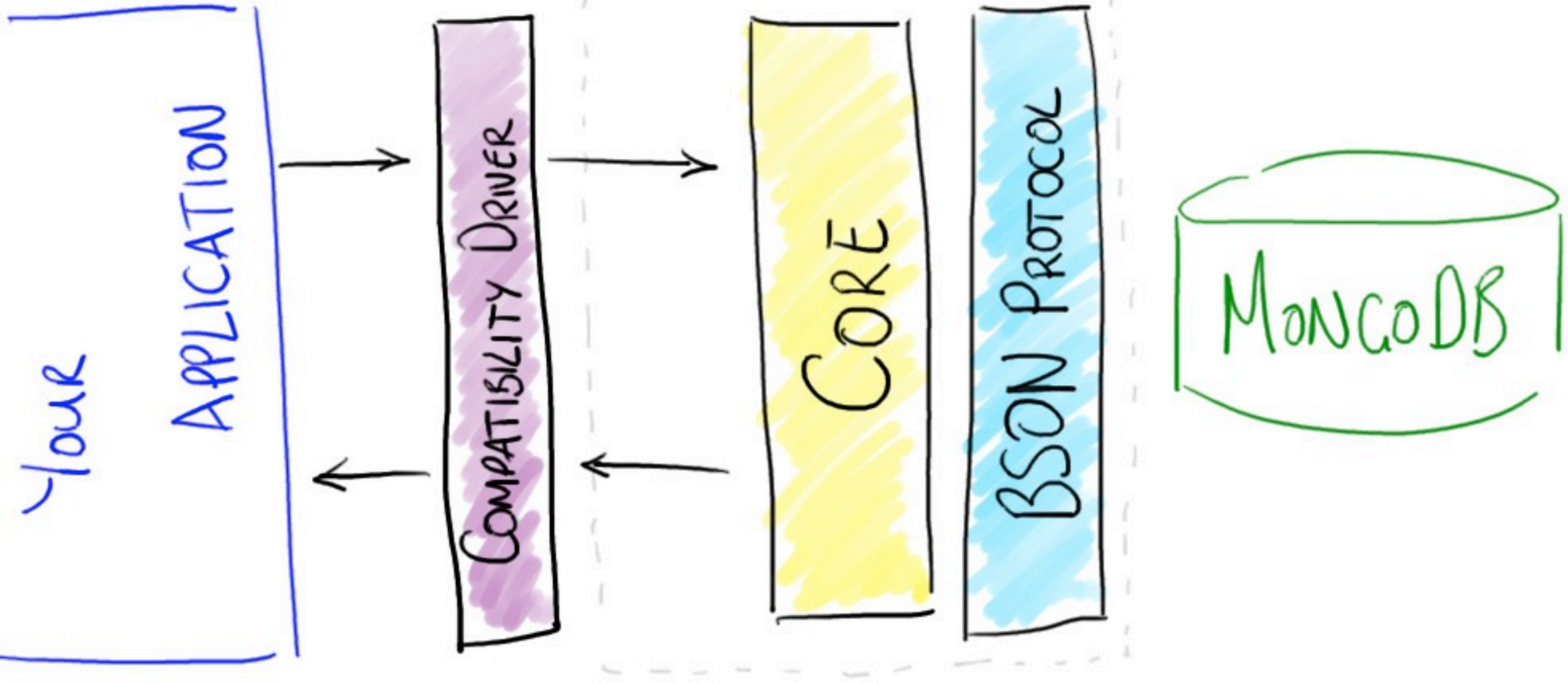


INSTEAD OF THIS...



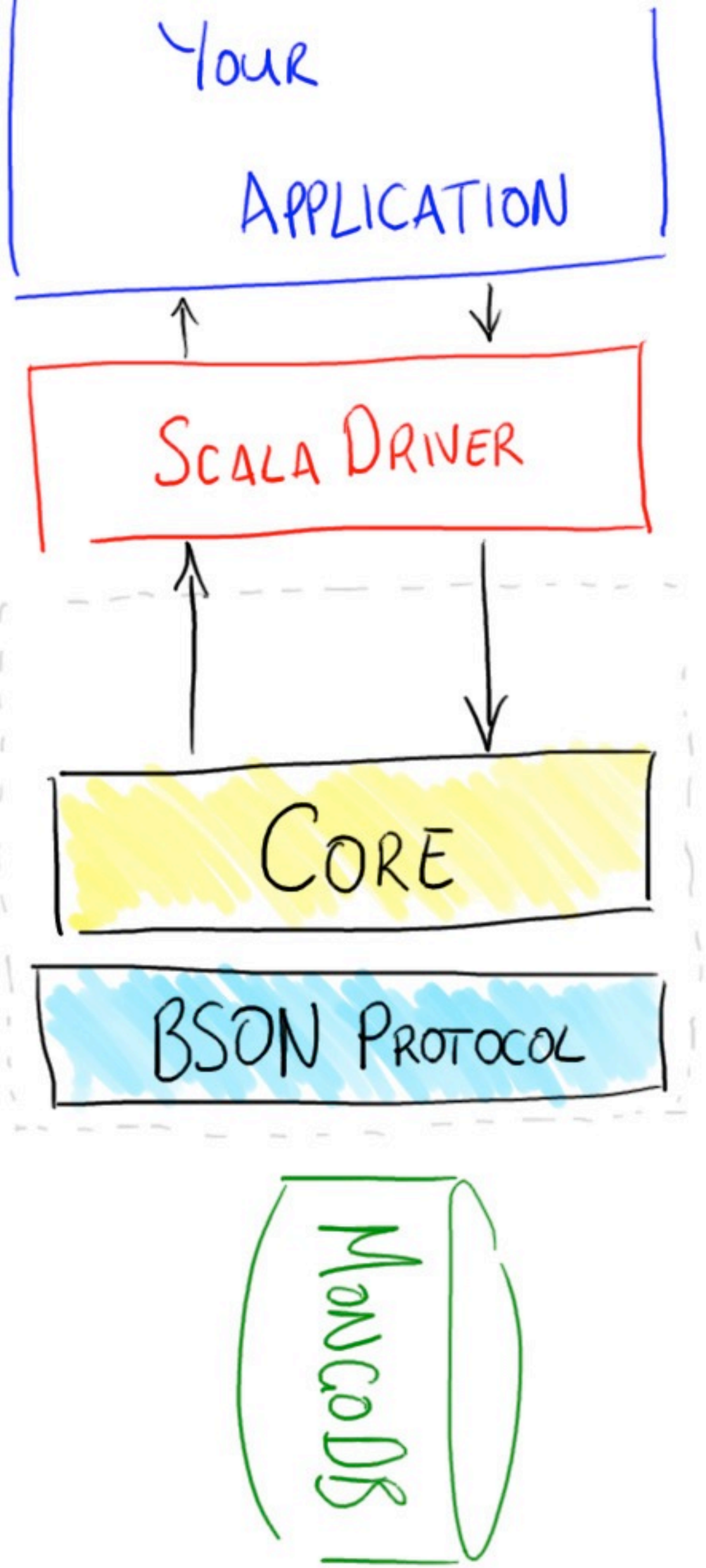
Separation of Concerns

Pluggable APIs

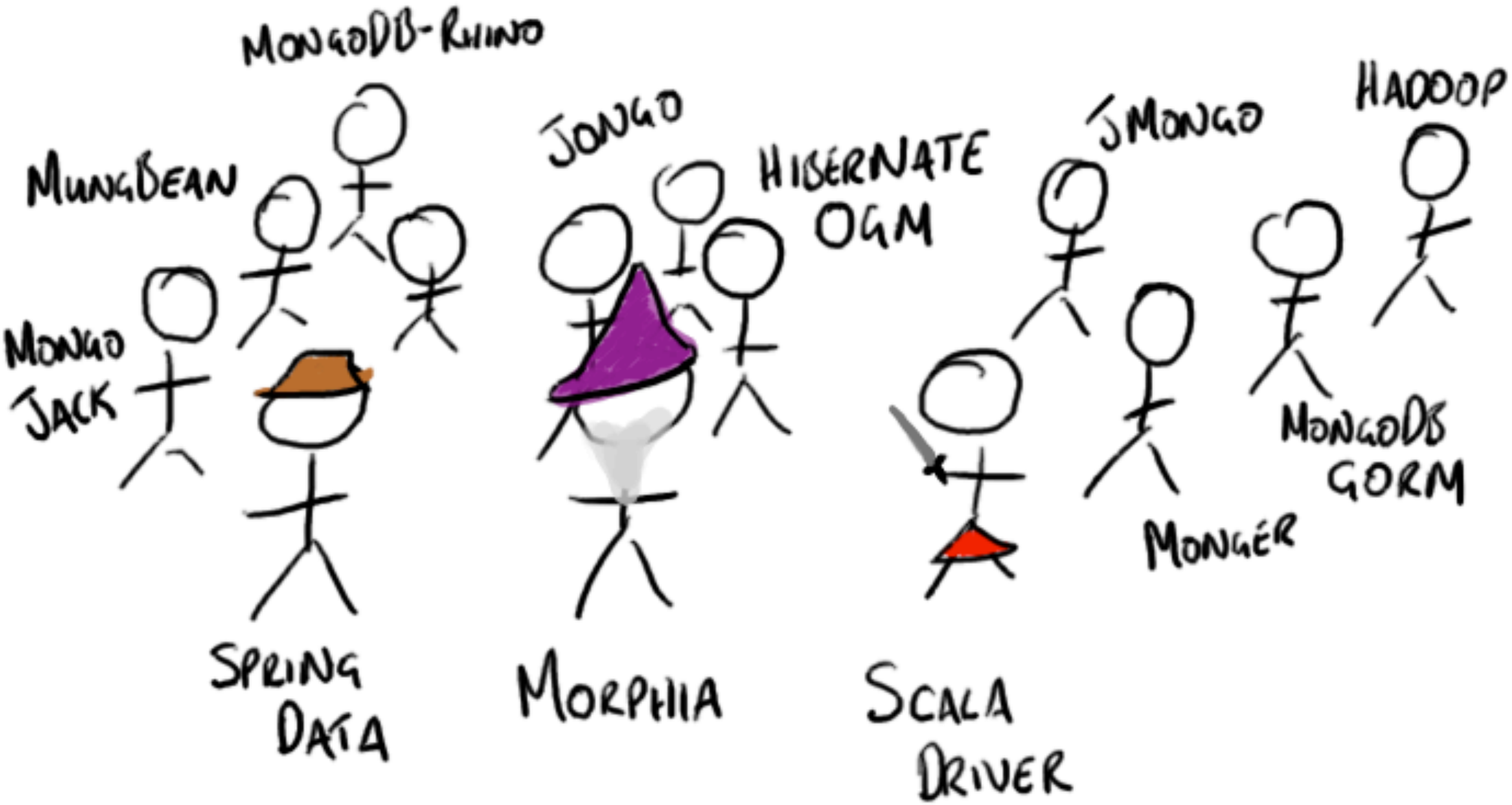


Backwards Compatible

SCALA SUPPORT



Support other libraries


MONGODB-RUBY
MUNGBEAN
MONGO JACK
SPRING DATA
JONGO
HIBERNATE OGM
MORPHIA
JMONGO
HADOOP
MONGODB GORM
MONGER
SCALA DRIVER

Easier to maintain

Confusing API

Method Overloading

```
collection.find
```

```
find(DBObject ref)                          DBCursor
find()                                      DBCursor
find(DBObject ref, DBObject ke…             DBCursor
find(DBObject query, DBObject …             DBCursor
find(DBObject query, DBObject …             DBCursor
findAndModify(DBObject query, …             DBObject
findOne()                                   DBObject
findAndModify(DBObject query, …             DBObject
findAndRemove(DBObject query)               DBObject
findOne(DBObject o)                         DBObject
Use ⇧⌘⏎ to syntactically correct your code after completing (balance parentheses etc.)
```

```
collection.upd
```

```
m update(DBObject q, DBObjec…  WriteResult
m update(DBObject q, DBObjec…  WriteResult
m update(DBObject q, DBObjec…  WriteResult
m update(DBObject q, DBObject o, boolean upsert, boolean multi,
m updateMulti(DBObject q, DB…  WriteResult
```

UPDATE

Update

```
collection.update(query, newValues);
```

Update

Update

```
collection.update(query, newValues);
```

```
collection.update(query, newValues, false, false, JOURNALED);
```

Update

DSL-ish

Update

```
collection.update(query, newValues);
```

Update

```
collection.update(query, newValues);
collection.find(query).updateOne(newValues);
```

Update

```
collection.update(query, newValues);
collection.find(query).updateOne(newValues);
```

Update

Update

```
collection.update(query, newValues);

collection.find(query).updateOne(newValues);


collection.find(query)
          .withWriteConcern(JOURNALED)
          .updateOne(newValues);
```

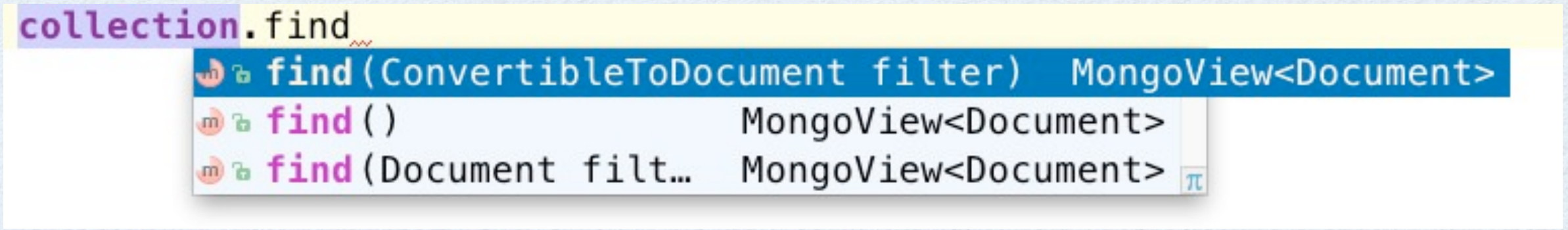
Update

```
collection.update(query, newValues);

collection.find(query).updateOne(newValues);


collection.find(query)
          .withWriteConcern(JOURNALED)
          .updateOne(newValues);

collection.update(query, newValues, false, false, JOURNALED);
```

Update

Self Documenting

IDE-friendly



FEWER CHOICES

```
collection.find(query).
```

```
project(Document selector)  MongoView<Document>
into(A target)                                A
find(ConvertibleToD…  MongoView<Document>
find(Document filte…  MongoView<Document>
limit(int limit)      MongoView<Document>
project(Convertible…  MongoView<Document>
skip(int skip)        MongoView<Document>
sort(ConvertibleToD…  MongoView<Document>
sort(Document sortC…  MongoView<Document>
upsert()              MongoView<Document>
withQueryOptions(Qu…  MongoView<Document>
Use ⇧⌘⏎ to syntactically correct your code after completing (balance parentheses etc.)
```

CTRL + SPACE

Testing is Hard and Boring

...and we can't agree on style

```java
@Test
public void shouldIgnoreCaseWhenCheckingIfACollectionExists() {
    // Given
    database.getCollection("foo1").drop();
    assertFalse(database.collectionExists("foo1"));

    // When
    database.createCollection("foo1", new BasicDBObject());

    // Then
    assertTrue(database.collectionExists("foo1"));
    assertTrue(database.collectionExists("FOO1"));
    assertTrue(database.collectionExists("fOo1"));

    // Finally
    database.getCollection("foo1").drop();
}
```

BDD-ish

VS

```
@Test
public void testInvalid() throws UnknownHostException {
    try {
        new DBAddress(null);
        fail();
    } catch (NullPointerException e) { // NOPMD
        // all good
    }

    try {
        new DBAddress("  \t\n");
        fail();
    } catch (IllegalArgumentException e) { // NOPMD
        // all good
    }
}
```

...and the tests are confusing

```
@Test
public void testMaxScan() {
    countResults(new DBCursor(collection,
                              new BasicDBObject(),
                              new BasicDBObject(),
                              primary())
                         .addSpecial("$maxScan", 4), 4);
    countResults(new DBCursor(collection,
                              new BasicDBObject(),
                              new BasicDBObject(),
                              primary())
                         .maxScan(4), 4);
}
```

Spock!

```
def 'should throw duplicate key when response has a duplicate key
error code'() {
    given:
    def commandResult = new CommandResult(new ServerAddress(),
                                          ['ok'  : 1,
                                           'err' : 'E11000',
                                           'code': 11000] as Document,
                                          1);


    when:
    ProtocolHelper.getWriteResult(commandResult)

    then:
    def e = thrown(MongoDuplicateKeyException)
    e.getErrorCode() == 11000
}
```

Really? Groovy?

Our tests are our documentation

BDD-ish thinking

Test styles converging

```groovy
def '#wc should return write concern document #commandDocument'()
{
    expect:
    wc.asDocument() == commandDocument;

    where:
    wc                                  | commandDocument
    WriteConcern.ACKNOWLEDGED           | ['w' : 1]
    WriteConcern.JOURNALED              | ['w' : 1, 'j': true]
    WriteConcern.FSYNCED                | ['w' : 1, 'fsync': true]
    new WriteConcern('majority')        | ['w': 'majority']
    new WriteConcern(2, 100)            | ['w' : 2, 'wtimeout': 100]
}
```

Data Driven Testing

```groovy
def '#wc should return write concern document #commandDocument'()
{
    expect:
    wc.asDocument() == commandDocument;

    where:
    wc                                  | commandDocument
    WriteConcern.ACKNOWLEDGED           | ['w' : 1]
    WriteConcern.JOURNALED              | ['w' : 1, 'j': true]
    WriteConcern.FSYNCED                | ['w' : 1, 'fsync': true]
    new WriteConcern('majority')        | ['w': 'majority']
    new WriteConcern(2, 100)            | ['w' : 2, 'wtimeout': 100]
}
```

is.gd/ddTest

Data Driven Testing

```groovy
def 'should throw IllegalArgumentException when required
parameter is not supplied for challenge-response'() {
    when:
    MongoCredential.createMongoCRCredential(null, 'test', ...);
    then:
    thrown(IllegalArgumentException)

    when:
    MongoCredential.createMongoCRCredential('user', null, ...);
    then:
    thrown(IllegalArgumentException)

    when:
    MongoCredential.createMongoCRCredential('user', 'test', ...);
    then:
    thrown(IllegalArgumentException)
}
```

Lesser Evil

Java 8 is coming

Lambdas

New Collections API

Date & Time

...and other stuff

But we support 1.5

But we support 1.5

But we support 1.6

Do Your Homework

Single Method Interfaces

```
DBObject query = new BasicDBObject("name", theNameToFind);
DBCursor results = collection.find(query);
for (DBObject dbObject : results) {
   // do stuff with each result
}
```

External Iteration

```
Document query = new Document("name", theNameToFind);
collection.find(query).forEach(new Block<Document>() {
    public boolean run(Document document) {
        // do stuff with each result
    }
});
```

Anonymous Inner Classes

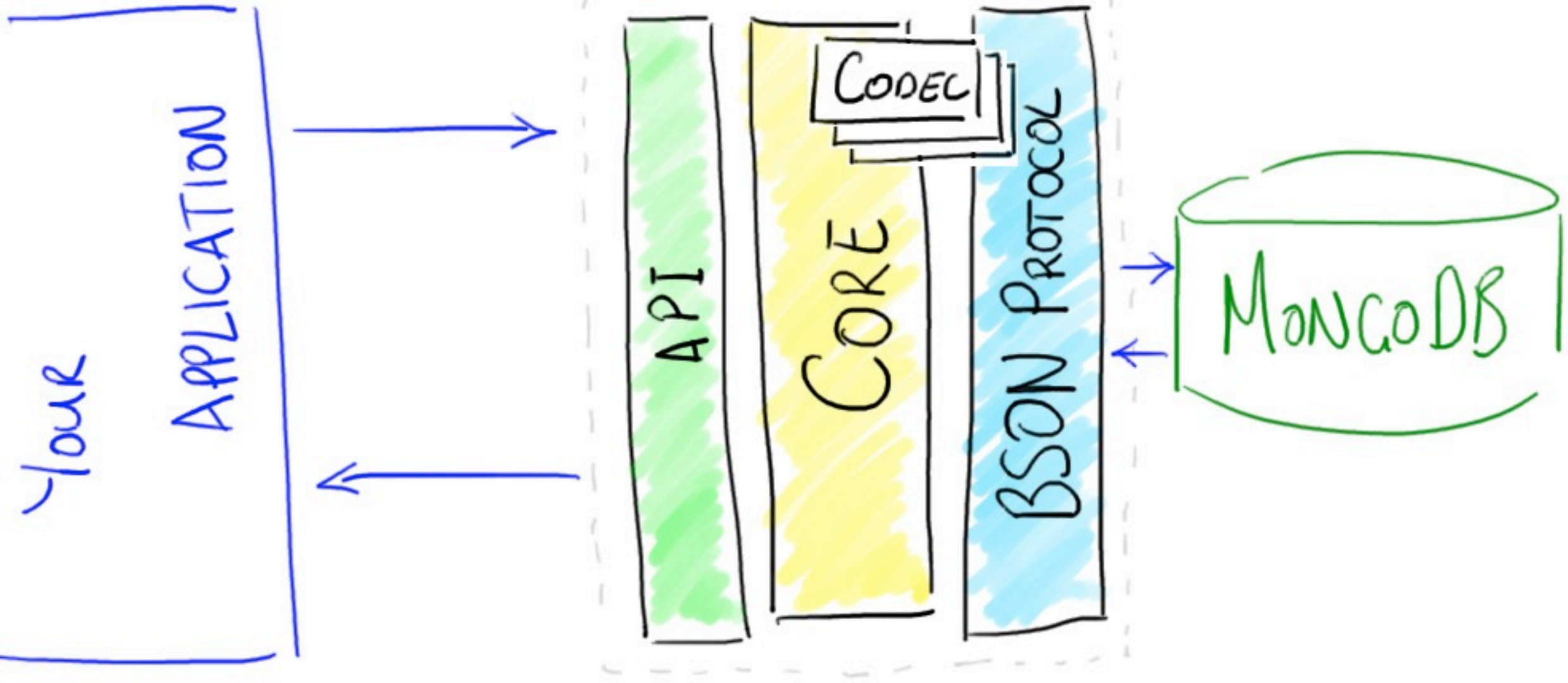
```
Document query = new Document("name", theNameToFind);
collection.find(query).forEach(document -> {

        // do stuff with each result

});
```

Lambdas!

API supports lambdas

(Mongo) Collection API Java 8-ish

Separation of Concerns



PLUGGABLE CODECS

```
MongoCollection<Document> collection =
    database.getCollection("coll");

Document query = new Document("name", theNameToFind);
Document result = collection.find(query).getOne();

Document addressDocument = (Document)result.get("address");
Customer customer = new Customer(result.getString("name"),
                        new Address(addressDocument.getString("street"),
                                    addressDocument.getString("city"),
                                    addressDocument.getString("state"),
                                    addressDocument.getInteger("zip")),
                        (List<Document>) result.get("books"));
```

Decoding

```
MongoCollection<Document> collection =
    database.getCollection("coll");

Document query = new Document("name", theNameToFind);
Document result = collection.find(query).getOne();

Document addressDocument = (Document)result.get("address");
Customer customer = new Customer(result.getString("name"),
                        new Address(addressDocument.getString("street"),
                                    addressDocument.getString("city"),
                                    addressDocument.getString("state"),
                                    addressDocument.getInteger("zip")),
                        (List<Document>) result.get("books"));
```

Decoding

```
MongoCollection<Customer> collection =
    database.getCollection("coll", new CustomerCodec());

Document query = new Document("name", theNameToFind);
Customer customer = collection.find(query).getOne();
```

Custom Codecs

Now With Generics!

```
MongoCollection<Customer> collection =
    database.getCollection("coll", new CustomerCodec());

Document query = new Document("name", theNameToFind);
Customer customer = collection.find(query).getOne();
```

Separation of concerns

Decoding code in one place

Can support new Date & Time

Caveat: Under Construction

Unfinished Business

Codecs

New Public API

Clear testing definitions

Continuous Delivery

The Driver!!!

Lessons Learnt

It's been done before:

Domain Driven Design

Separation of Concerns

The API is your UI

The tools exist

Don't wait for Java 8

What's in it for you?

Try This At Home

Try This At Work

Download & use the new MongoDB driver

is.gd/java3mongodb


Come play

#JFokus

@trisha_gee

Questions

http://is.gd/java3mongodb

I want to run the tests

Every time before I check in

I want good quality pull requests

I want it to be easy for people to start

If it can be automated, do it

Gradle

Painless Dependency Management

Easy to introduce static analysis

Wrapper = no extra overhead

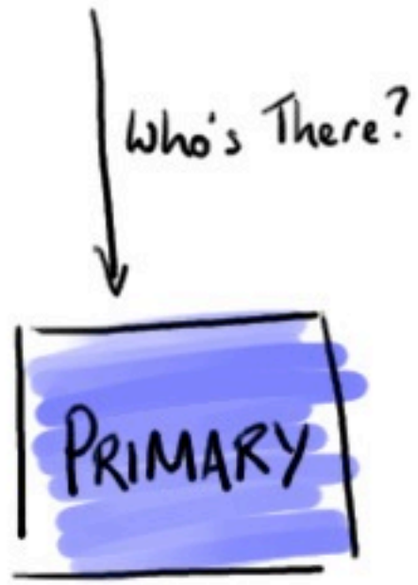
Easy to get started as a contributor

Includes IDE Setup

Performance Bottlenecks

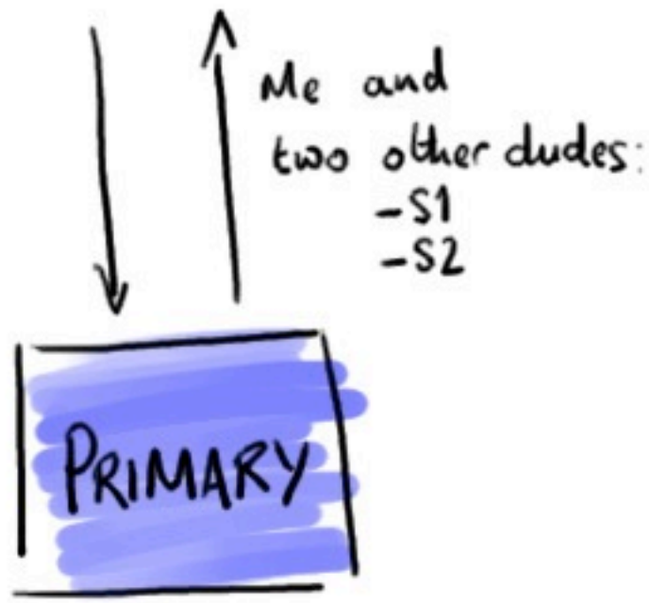
More Servers =

More Delay


Who's There?
PRIMARY


SECONDARY 1


SECONDARY 2


Me and
two other dudes:
-S1
-S2
PRIMARY

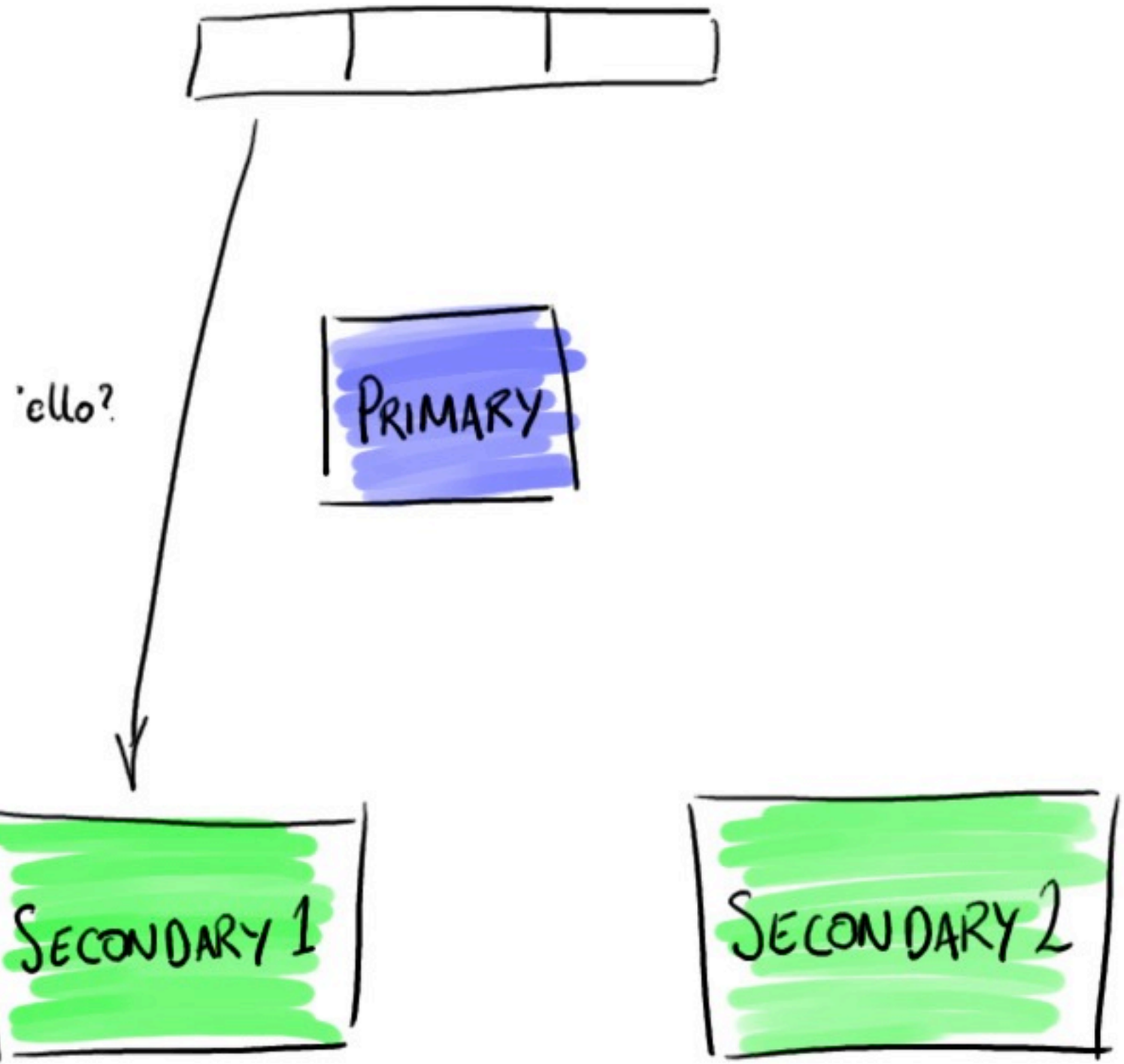

SECONDARY 1

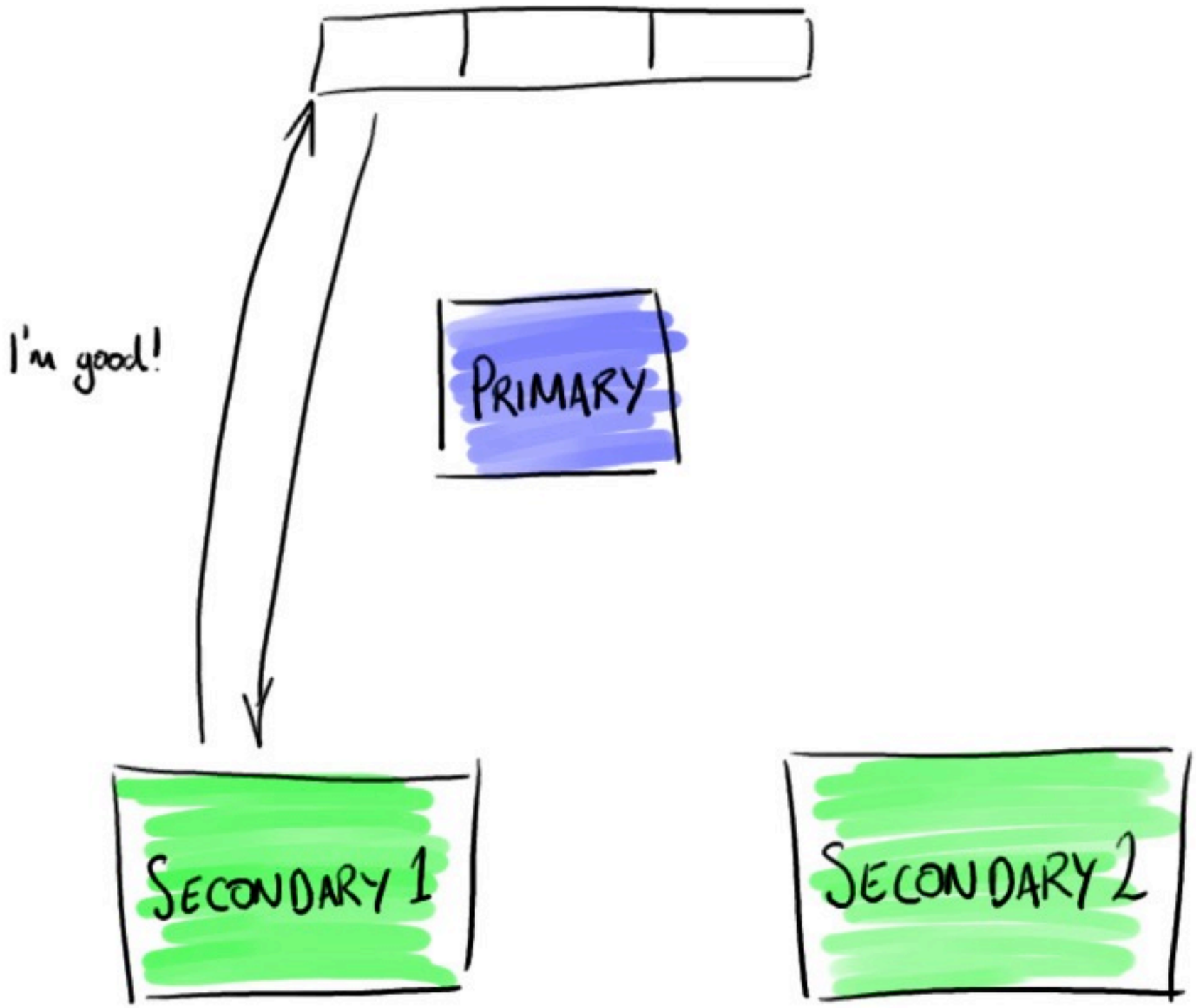

SECONDARY 2

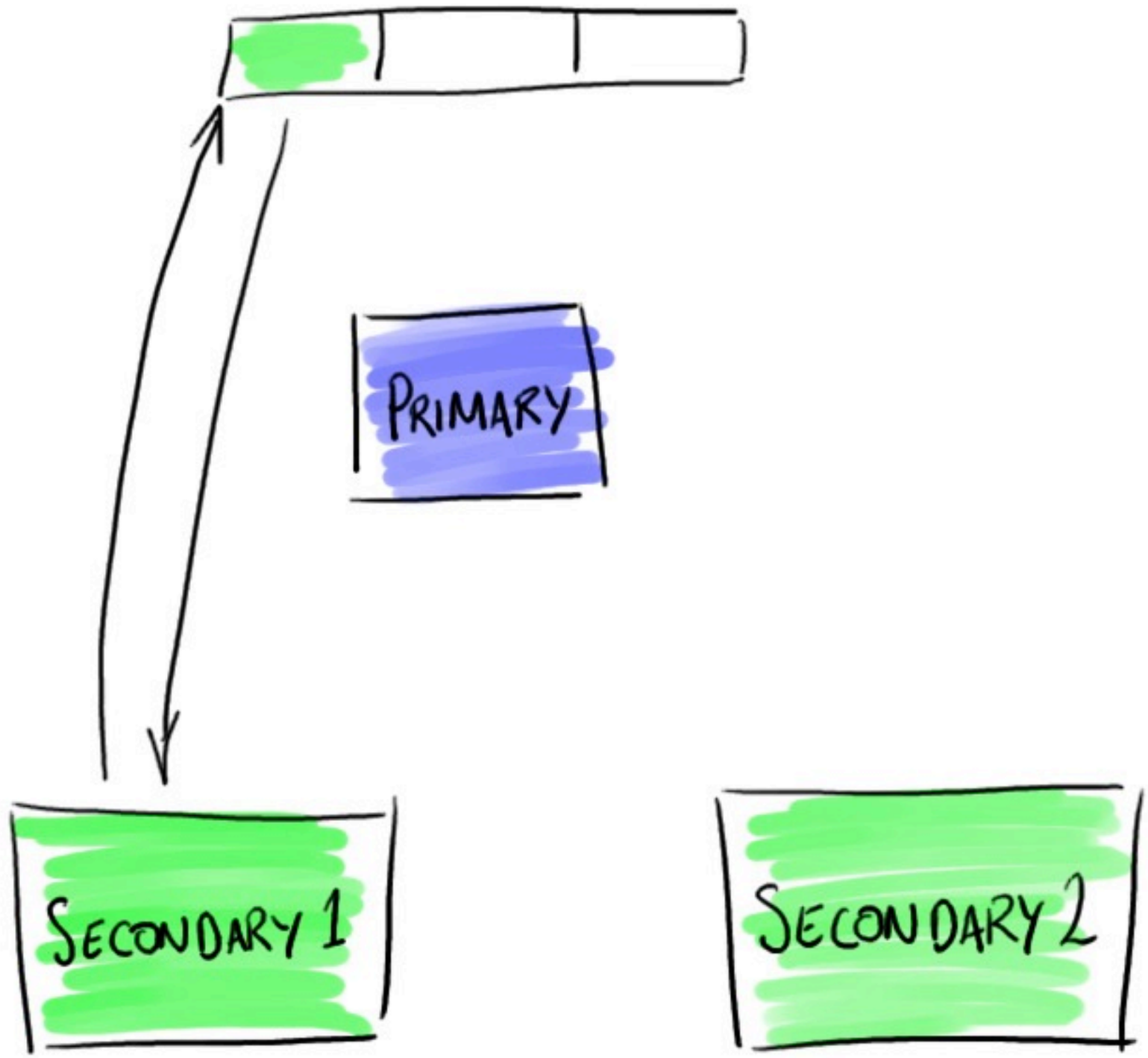

PRIMARY

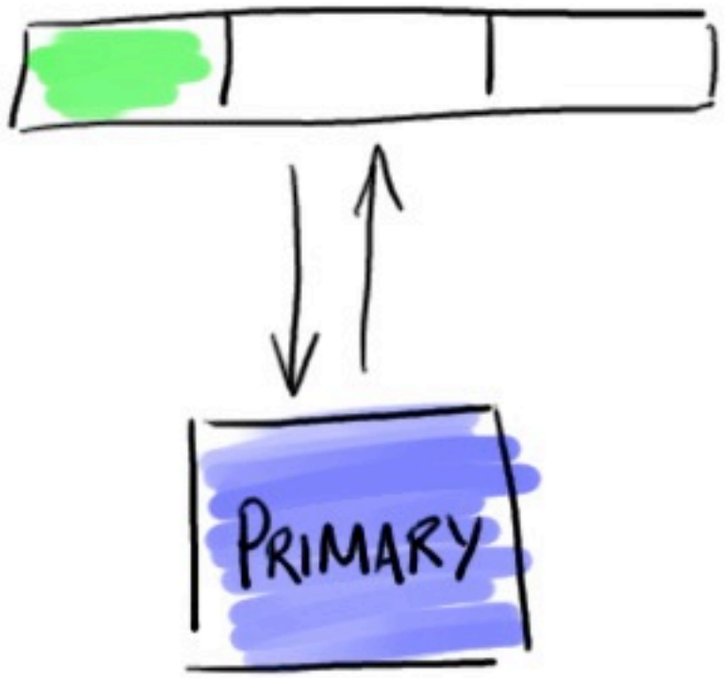

SECONDARY 1


SECONDARY 2


'ello?
PRIMARY
SECONDARY 1
SECONDARY 2

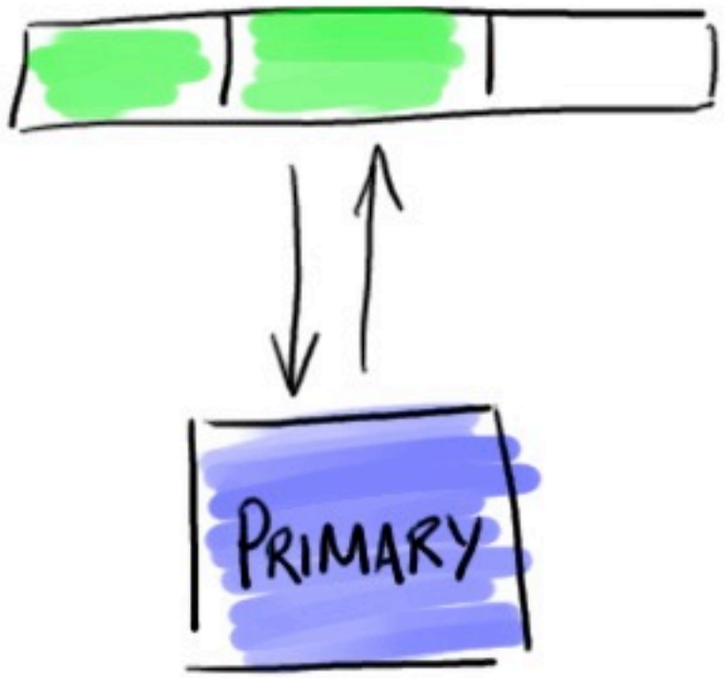

I'm good!
PRIMARY
SECONDARY 1
SECONDARY 2


PRIMARY
SECONDARY 1
SECONDARY 2


PRIMARY

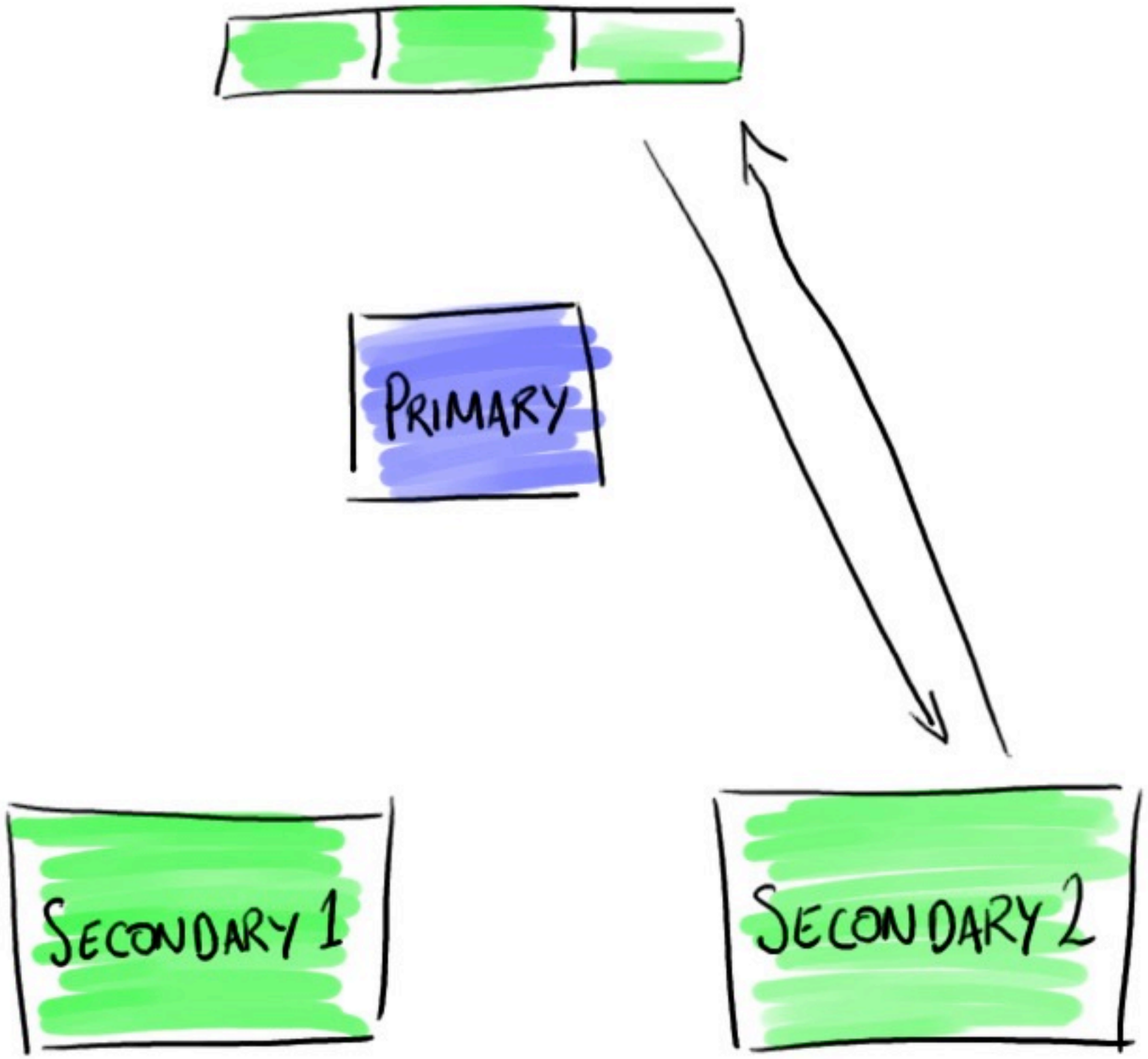

SECONDARY 1


SECONDARY 2


PRIMARY


SECONDARY 1


SECONDARY 2


PRIMARY
SECONDARY 1
SECONDARY 2

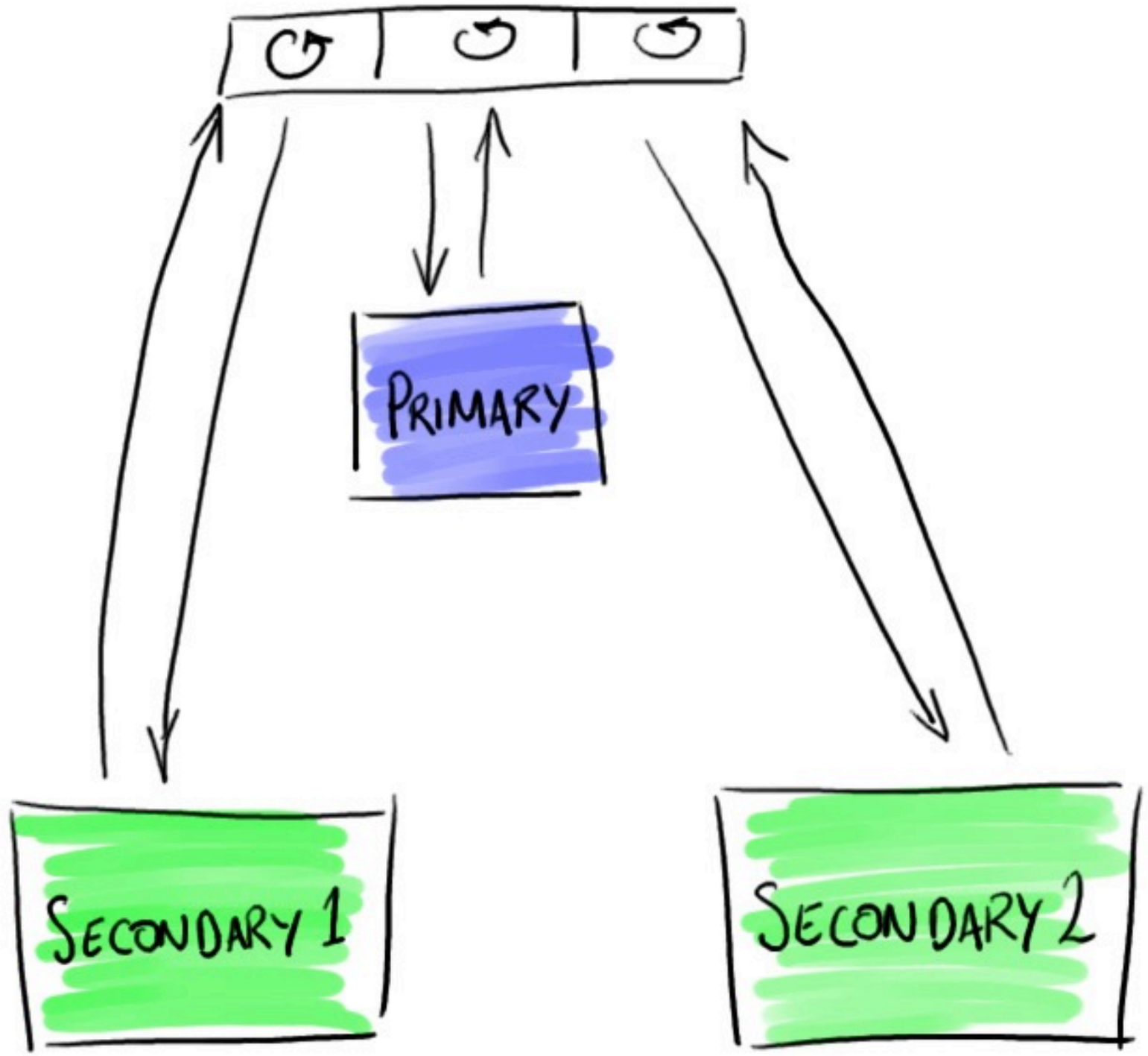
All is Well!

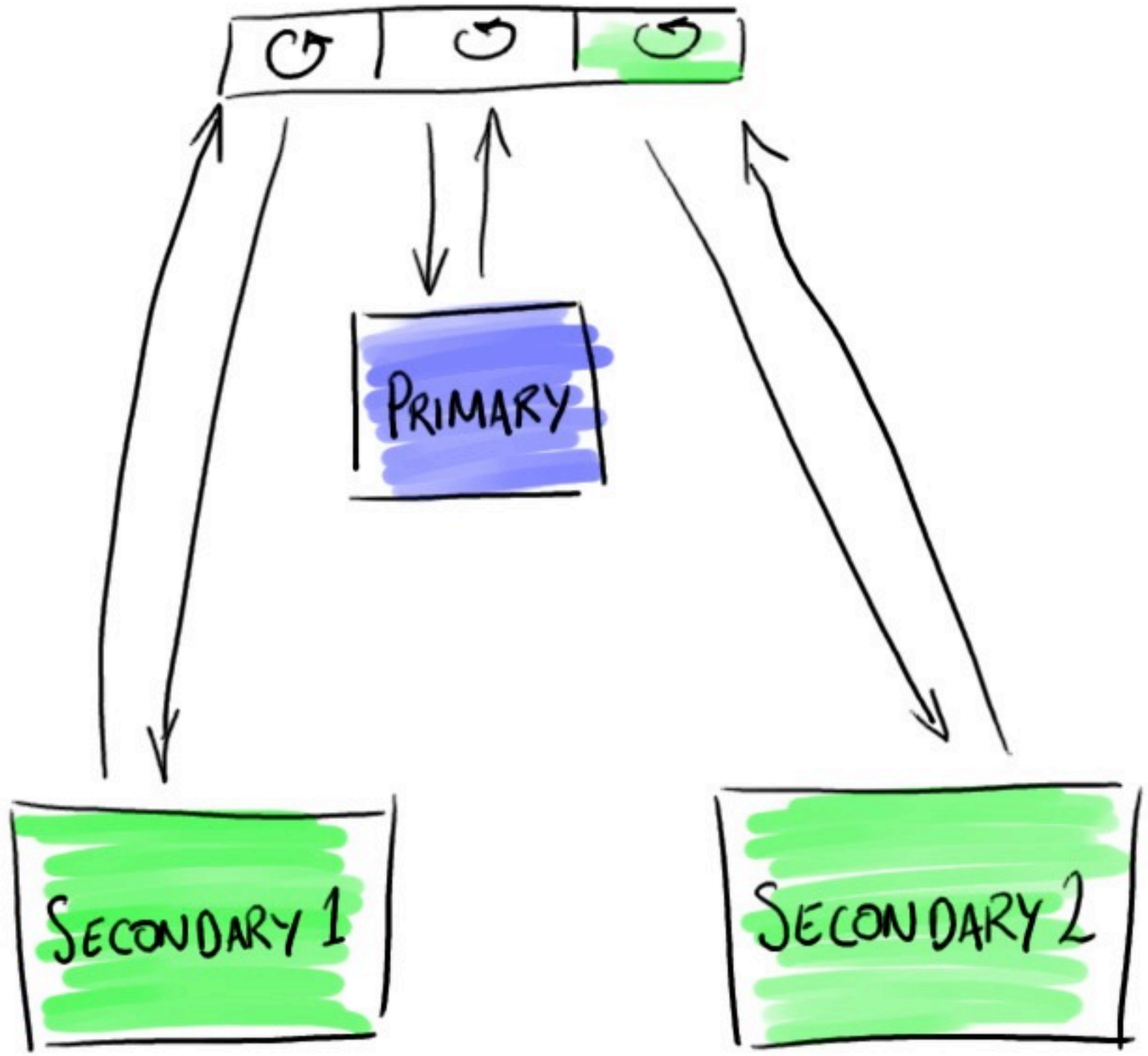
Event Driven

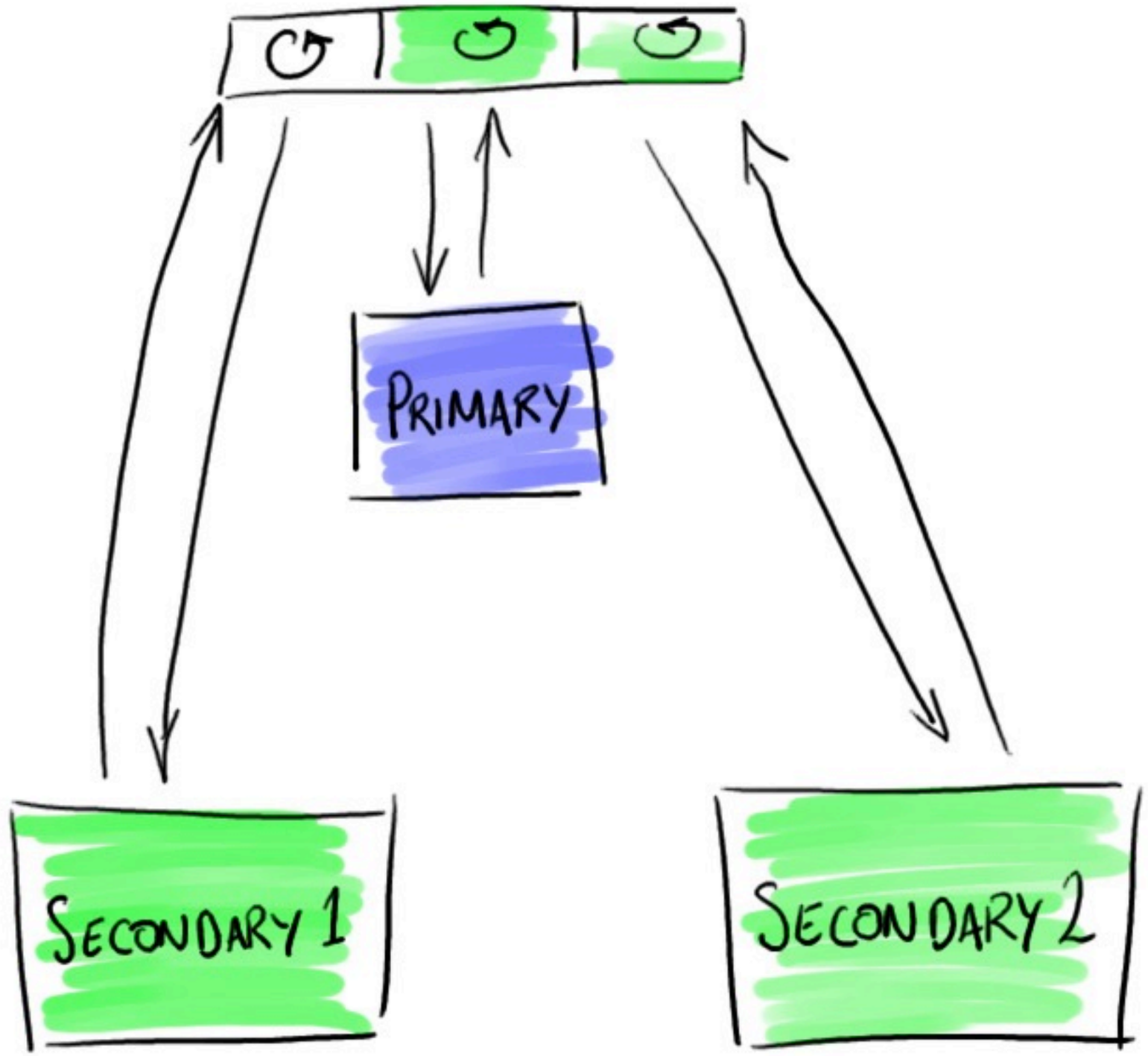

PRIMARY


SECONDARY 1


SECONDARY 2


PRIMARY
SECONDARY 1
SECONDARY 2


PRIMARY
SECONDARY 1
SECONDARY 2


PRIMARY
SECONDARY 1
SECONDARY 2

Async = Good

Events = Good

Scalable