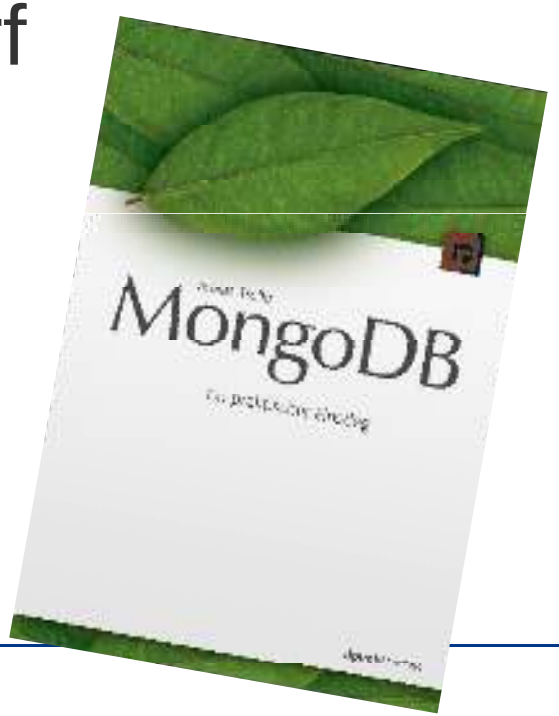

# Spring Data, Jongo & Co.

## Java Persistenz-Frameworks für MongoDB

Tobias.Trelle@codecentric.de @tobiastrelle

# Tobias Trelle

- Senior IT Consultant @ codecentric AG (Düsseldorf)
- Organisator MongoDB Usergruppe Düsseldorf
- Autor MongoDB-Buch (dpunkt-Verlag)



# Where have all my tables gone …

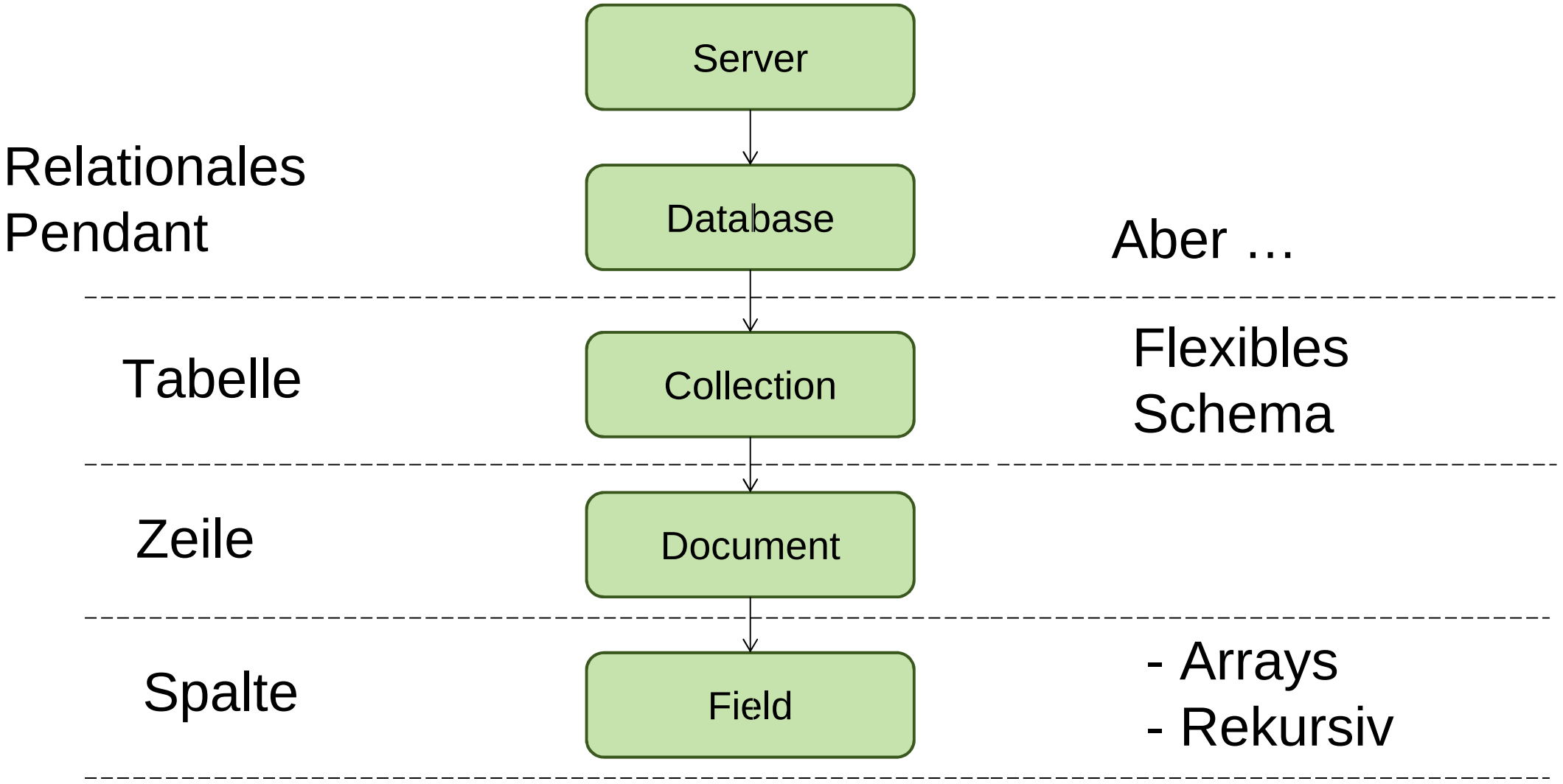
ORM is dead

long live ODM

# MongoDB?

- NoSQL-Datenbank / Open Source
- Dokumentenorientiert
- Hochperformant, horizontal skalierbar (scale-out)
- Replication & Sharding out-of-the-box
- Map/Reduce
- Geospatial Indexes / Queries

# Grundkonzept MongoDB-Server



# Document

```
{
 title: "MongoDB"
 versions: [
    { major: 2, minor: 6},
    { major: 2, minor: 4}
 ]
}
```

# JSON vs. BSON

```
{ hello: "MongoDB" }

\x18\x00\x00\x00
    \x02
       hello\x00
       \x08\x00\x00\x00MongoDB\x00
\x00
```

# CRUD = IFUR

**i**nsert(…)

**f**ind(…), findOne(…)

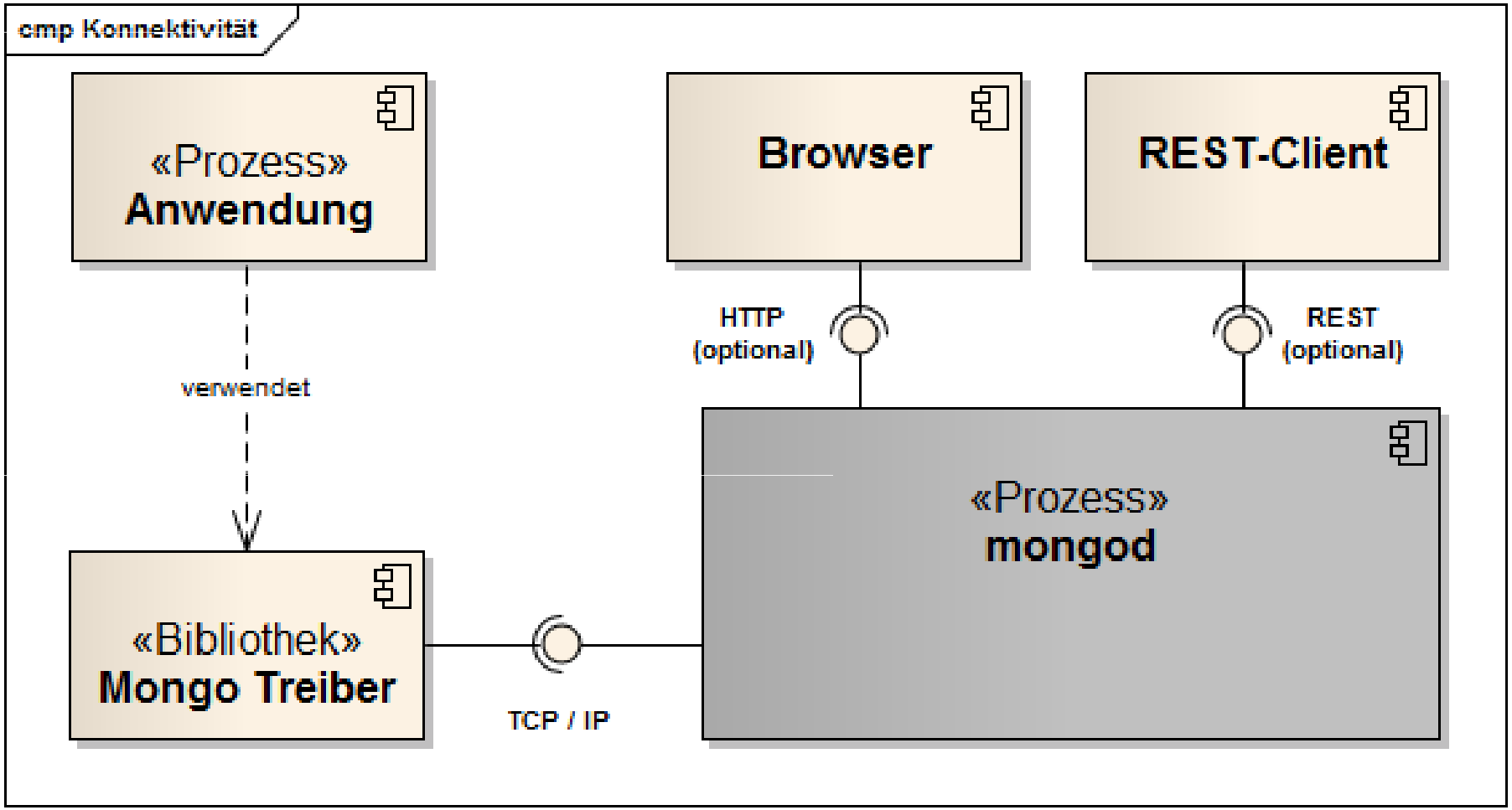
**u**pdate(…)

**r**emove()

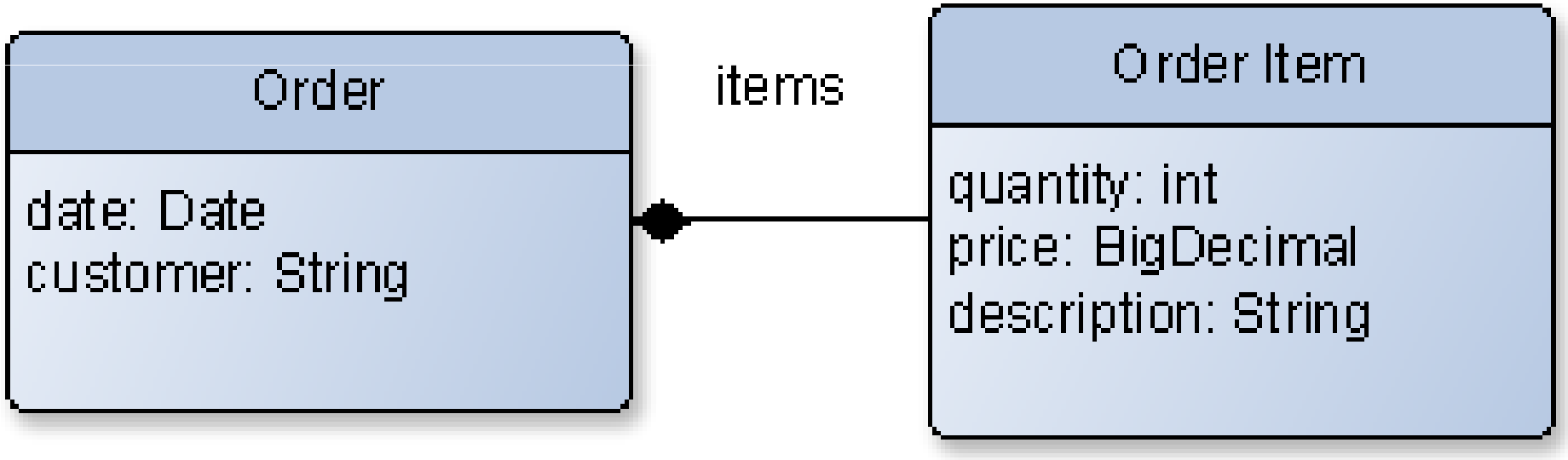
- MongoDB Java Driver

- Spring Data MongoDB
- Morphia
- Jongo
- EclipseLink for MongoDB

# Konnektivität



# Use Case




```
db.order.find( {"items.quantity": ? } )
```

# Mongo Java Driver

# MongoDB Drivers

- **One** wire protocol for **all** client languages
- A driver implementation per language
- Responsibilities:
  - Converting language dependent data structures ← → BSON
  - Generating ObjectId for _id field
- Overview: http://www.mongodb.org/display/DOCS/Drivers

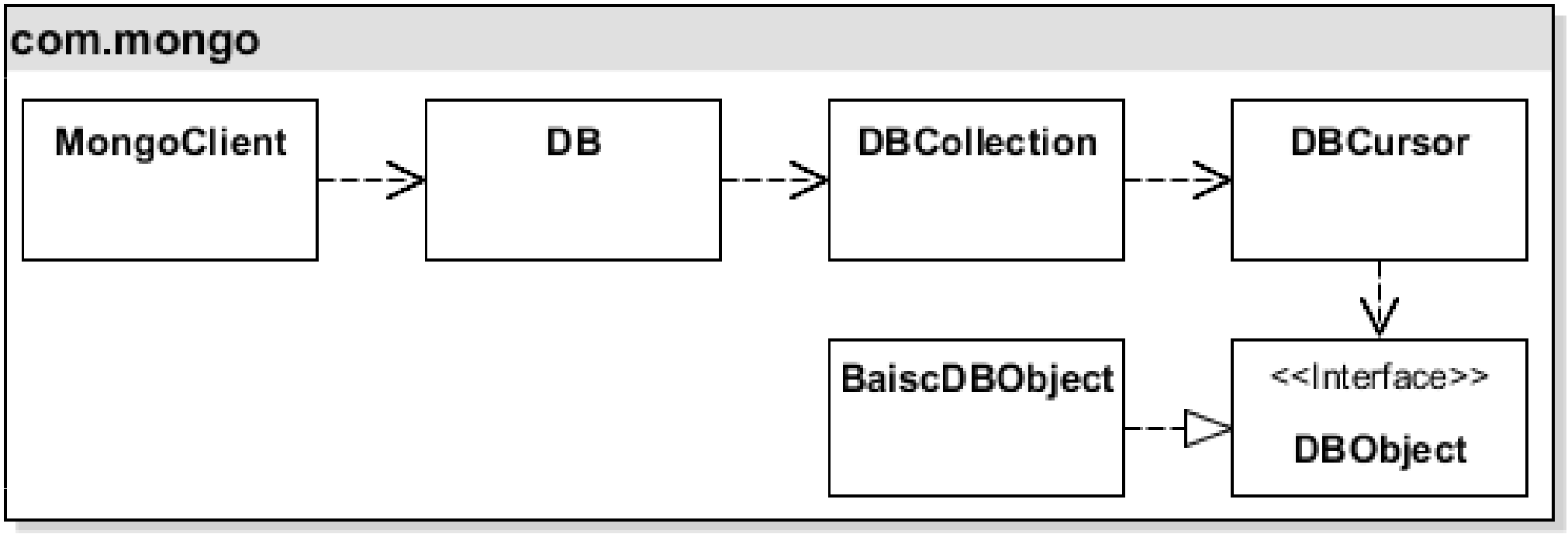
# Java Driver

- One JAR w/o further dependencies:

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongo-java-driver</artifactId>
    <version>2.11.3</version>
</dependency>
```

- github:
  https://github.com/mongodb/mongo-java-driver

# Java Driver: API Overview



# Java Driver: Connect to MongoDB

```
import com.mongodb.MongoClient;

// Default: localhost:27017
mongo = new MongoClient();

// Sharding: mongos server
mongo = new MongoClient("mongos01", 4711);

// Replica set
mongo = new MongoClient(Arrays.asList(
      new ServerAddress("replicant01", 10001),
      new ServerAddress("replicant02", 10002),
      new ServerAddress("replicant03", 10003)
      ));
```

# Java Driver: Database / Collection

```
import com.mongodb.DB;
import com.mongodb.DBCollection;

DB db = mongo.getDB("test");

DBCollection collection =
    db.getCollection("foo");
```

# Java Driver: Documents

```
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;

// insert document
DBObject doc = new BasicDBObject();
doc.put("date", new Date());
doc.put("i", 42);

collection.insert(doc);
```

# Java Driver: Queries

```
import com.mongodb.DBCursor;

DBCursor cursor;

cursor = collection.find(); // all documents

// documents w/ {i: 42}
cursor = collection.find(
     new BasicDBObject("i", 42) );

document = cursor.next();
...
```

# Java Driver: Order Use Case

```
DB db = mongo.getDB("test");
DBCollection collection = db.getCollection("order");
DBObject order;
List<DBObject> items = new ArrayList<DBObject>();
DBObject item;

// order
order = new BasicDBObject();
order.put("date", new Date());
order.put("custInfo" , "Tobias Trelle");
order.put("items", items);
// items
item = new BasicDBObject();
item.put("quantity", 1);
item.put("price", 47.11);
item.put("desc", "Item #1");
items.add(item);
item = new BasicDBObject();
item.put("quantity", 2);
item.put("price", 42.0);
item.put("desc", "Item #2");
items.add(item);

collection.insert(order);
```

# Java Driver: Order Use Case

```
DB db = mongo.getDB("test");
DBCollection collection = db.getCollection("order");
DBObject query;
DBObject document;
DBCursor cursor;

query = new BasicDBObject("items.quantity", 2);
cursor = collection.find(query);

while ( cursor.hasNext() ) {
      document = cursor.next();
      println(document);
}
```

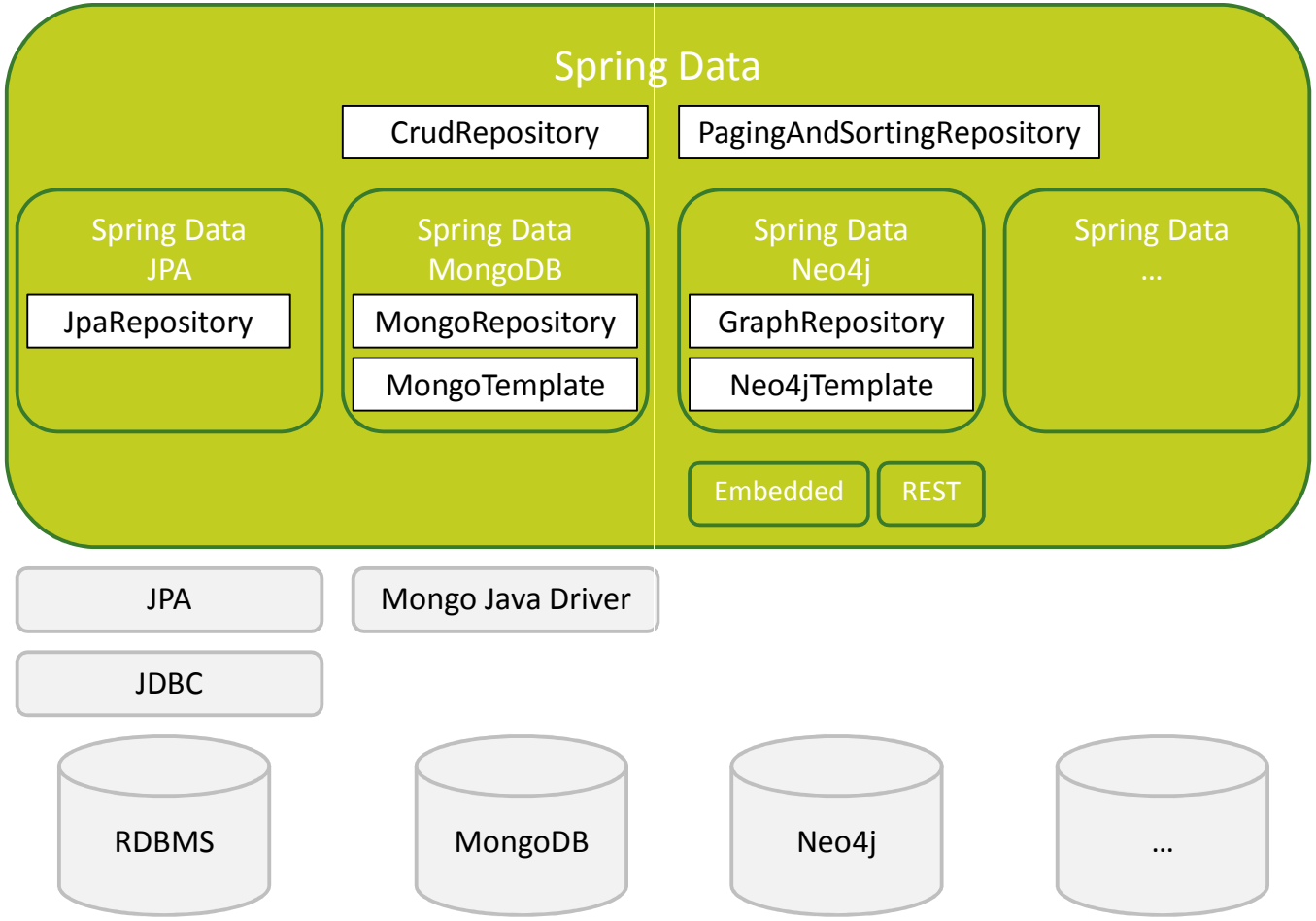
# Spring Data MongoDB

# Spring Data MongoDB – Fact Sheet

| | |
|---|---|
| Vendor | Pivotal / SpringSource |
| License | Apache License, Version 2.0 |
| Documentation | http://www.springsource.org/spring-data/mongodb |
| Main Features | • Repository Support<br>• Object/Document Mapping<br>• Templating |

# Spring Data

## Common patterns for RDBMS and NoSQL data stores




Quelle: http://www.infoq.com/articles/spring-data-intro

# Spring Data MongoDB

## Templating

- Resource abstraction
- Configure connections to mongod / mongos node(s)
- Collection lifecycle ( create, drop)
- Map/Reduce / Aggregation

## Object Mapping

- Annotation based: @Document, @Field, @Index etc.
- Classes are mapped to collections, Java Objects to documents

## Repository Support

- Queries are derived from methods signatures
- Annotated Queries

# Spring Data MongoDB: Configuration

```
<!-- Connection to MongoDB server -->
<mongo:db-factory host="localhost" port="27017" dbname="test" />

<!-- MongoDB Template -->
<bean id="mongoTemplate„
    class="org.springframework.data.mongodb.core.MongoTemplate">
  <constructor-arg name="mongoDbFactory" ref="mongoDbFactory"/>
</bean>

<!-- Package w/ automagic repositories -->
<mongo:repositories base-package="mongodb" />
```

# Spring Data MongoDB: Template

**Detailed configuration of a MongoDB connection:**

```
<mongo:mongo host="${mongo.host}" port="${mongo.port}">
  <mongo:options
        connections-per-host="${mongo.connectionsPerHost}„
        threads-allowed-to-block-for-connection
        multiplier="${mongo.threadsAllowedToBlockForConnectionMultiplier}„
        connect-timeout="${mongo.connectTimeout}„
        max-wait-time="${mongo.maxWaitTime}„
        auto-connect-retry="${mongo.autoConnectRetry}„
        socket-keep-alive="${mongo.socketKeepAlive}„
        socket-timeout="${mongo.socketTimeout}„
        slave-ok="${mongo.slaveOk}„
        write-number="1„
        write-timeout="0„
        write-fsync="true"/>
</mongo:mongo>
<mongo:db-factory dbname= "test" mongo-ref="mongo"/>
```

# Spring Data MongoDB: Object Mapping

```
public class Order {
  @Id private String id;
  private Date date;
  @Field("custInfo") private String customerInfo;
  List<Item> items;  ...
}

public class Item {
  private int quantity;
  private double price;
  @Field("desc") private String description;
  ...
}
```

# Spring Data MongoDB: Repository Support

```
public interface OrderRepository extends
  MongoRepository<Order, String> {

  List<Order> findByItemsQuantity(int quantity);

—@Query(
  —value = "{ custInfo: ?0 }",
  —fields = "{_id:0, items:1}")
—List<Order> findOnlyItems(String name);
}
```

# Spring Data MongoDB: Additional Goodies

- Map/Reduce / Aggregation framework
- Index Management
- Support for GridFS
- Geopspatial indexes / queries
- Optimistic Locking

# Morphia

# Morphia – Fact Sheet

| | |
|---|---|
| Vendor | MongoDB Inc. |
| License | Apache License, Version 2.0 |
| Documentation | https://github.com/mongodb/morphia |
| Main Features | • Object/Document Mapping<br>• Custom Query API<br>• DAO support |

# Morphia: Object Mapping

```
public class Order {
   @Id private ObjectId id;
   private Date date;
   @Property("custInfo") private String customerInfo;
   @Embedded List<Item> items;
   ...
}
public class Item {
   private int quantity;
   private double price;
   @Property("desc") private String description;
   ...
}
```

# Morphia: Queries

```
public class OrderDao extends BasicDAO<Order, ObjectId> {

   List<Order> findByItemsQuantity(int quantity) {
        return
        find( createQuery().filter("items.quantity", quantity))
        .asList();
   }
   List<Order> findByItemsPriceGreaterThan(double price) {
        return
        find( createQuery().field("items.price").greaterThan(price) )
        .asList();
   }
   …
}
```

# Morphia: Queries

```
public class OrderDao extends BasicDAO<Order, ObjectId> {

— // Projection
— List<Item> findItems(String customerName) {
—    List<Order> orders = find(createQuery().
—       field("custInfo").equal(customerName).
—       retrievedFields(true, "items")
—        ).asList();

—    // check for null in production code!
—    return orders.get(0).getItems();
— }
}
```

# Morphia: Custom query syntax – why?

| Morphia | Mongo Query |
|---|---|
| = | $eq |
| !=, <> | $neq |
| >, <, >=,<= | $gt, $lt, $gte, $lte |
| in, nin | $in, $nin |
| elem | $elemMatch |
| … | …. |

# Jongo

# Jongo – Fact Sheet

| | |
|---|---|
| Developer | Benoît Guérout, Yves Amsellem |
| License | Apache License, Version 2.0 |
| Documentation | http://jongo.org/ |
| Main Features | • Object/Document Mapping<br>• Custom Query API |

# Jongo: Object Mapping

```
public class Order {
   private ObjectId id;
   private Date date;
   @JsonProperty("custInfo") private String customerInfo;
   List<Item> items;
… }

public class Item {
   private int quantity;
   private double price;
   @JsonProperty("desc") private String description;
   …
}
```

# Jongo: Queries

```
// Java driver API
MongoClient mc = new MongoClient();
DB db = mc.getDB("odm_jongo");
// Jongo API entry point
Jongo jongo = new Jongo(db);
MongoCollection orders = jongo.getCollection("order");

// no DAO needed
Iterable<Order> result =
  orders.find("{\"items.quantity\": #}", 2).as(Order.class);
// Projection
Iterable<X> result =
  orders.find().fields("{_id:0, date:1, custInfo:1}").as(X.class);
```

# EclipseLink NoSQL

# EclipseLink NoSQL MongoDB – Fact Sheet

| | |
|---|---|
| Vendor | Eclipse Foundation |
| License | Eclipse Public License (EPL) |
| Documentation | http://www.eclipse.org/eclipselink/documentation/2.4/concepts/nosql.htm |
| Main Features | • JPA API (Subset)<br>• JPQL Query Language / Native Queries |

# EclipseLink NoSQL

- Implements JPA API (subset)
- JP-QL query are translated to native datastore queries
- Oracle NoSQL, **MongoDB**

# EclipseLink NoSQL MongoDB: Configuration

```
<persistence version="2.0" …>
<persistence-unit name="mongodb">
  <class>hibernate.Order</class>
  <class>hibernate.Item</class>
   <properties>
    <property name="eclipselink.target-database"
              value="org.eclipse.persistence.nosql.adapters.mongo.MongoPlatform"/>
    <property name="eclipselink.nosql.connection-spec"
              value="org.eclipse.persistence.nosql.adapters.mongo.MongoConnectionSpec"/>
    <property name="eclipselink.nosql.property.mongo.port" value="27017"/>
    <property name="eclipselink.nosql.property.mongo.host" value="127.0.0.1"/>
    <property name="eclipselink.nosql.property.mongo.db" value="odm_eclipselink"/>
    <property name="eclipselink.logging.level" value="FINE"/>
   </properties>
</persistence-unit>
</persistence>
```

# EclipseLink NoSQL MongoDB : Object Mapping

```
@Entity
@NoSql(dataFormat=DataFormatType.MAPPED)
public class Order {
   @GeneratedValue
   @Id
   @Field(name = "_id")
   private String id;

   private Date date;

   @Column(name = "custInfo") private String customerInfo;

   @ElementCollection
   private List<Item> items;
```

# EclipseLink NoSQL MongoDB : Object Mapping

```
@Embeddable
@NoSql(dataFormat=DataFormatType.MAPPED)
public class Item {

   private int quantity;

   private double price;

   @Column(name="desc") private String description;
   ...
```

# EclipseLink NoSQL MongoDB : Queries

```
// JPQL
Order order = em.createQuery(
    – "SELECT o FROM Order o JOIN o.items i WHERE
      i.quantity = 2",
    – Order.class).getSingleResult();

// native
Order order = (Order)em.createNativeQuery(
    – "db.ORDER.findOne({_id: \"" + id + "\"})",
    – Order.class).getSingleResult();
```

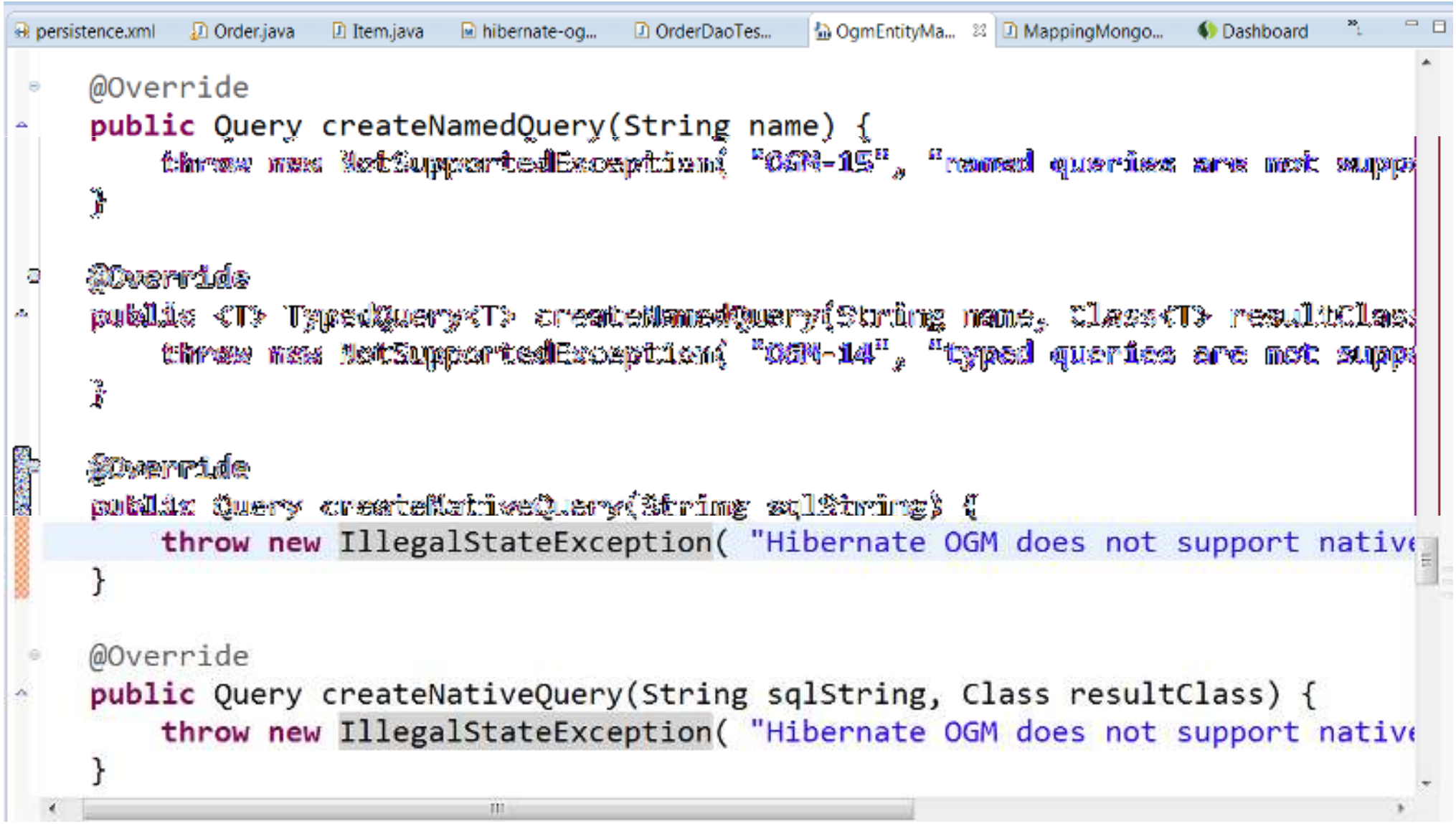
# EclipseLink NoSQL MongoDB

- Problems w/ nested native queries
- Uses **relational** API, TX required(?)

# More JPA Providers for MongoDB

- Hibernate OGM
  - Very early beta
  - No support for queries
- Data Nucleus

# Hibernate OGM: OgmEntityManager



# Judge yourself …

**Spring Data MongoDB**

https://github.com/ttrelle/spring-data-examples

**Morphia**

https://github.com/ttrelle/morphia-mongodb-examples

**Jongo**

https://github.com/ttrelle/jongo-examples

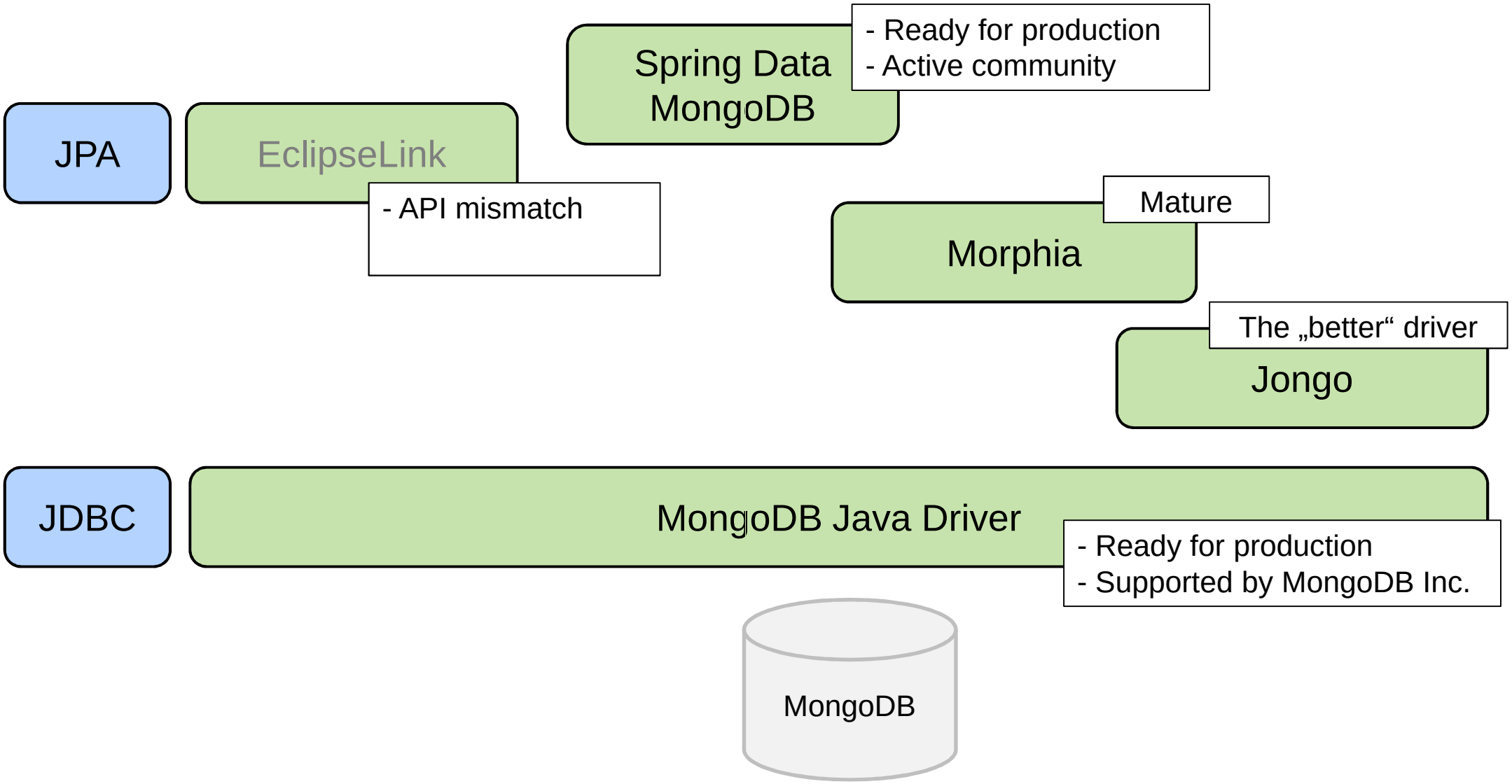
**EclipseLink**

https://github.com/ttrelle/eclipselink-mongodb-examples

# Summary

| | ODM Annotations | Queries |
|---|---|---|
| **Java Driver** | -- | Nested BasicDBObject‘s |
| **Spring Data MongoDB** | Custom | Interface w/<br>- derived queries<br>- Native queries |
| **EclipseLink** | JPA + Custom | JPQL: @Named(Native)Query + EntityManager |
| **Jongo** | Jackson | JSON queries via collection wrapper |
| **Morphia** | Custom | BasicDAO super class w/ 2 flavours of fluent API |

# Which one should I use?



# MUG Düsseldorf


MongoDB User Group Düsseldorf

https://www.xing.com/net/mongodb-dus

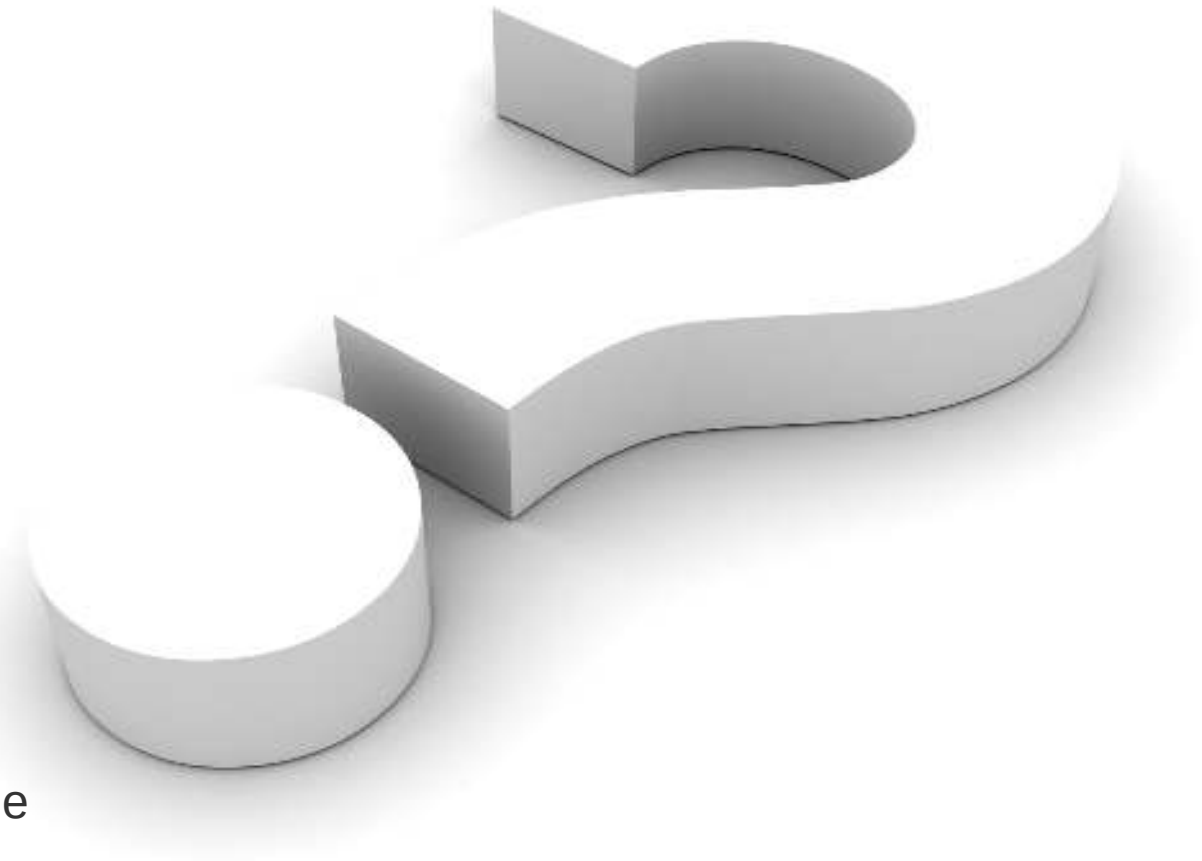
@MongoDUS

# QUESTIONS?

Tobias Trelle

codecentric AG
Merscheider Str. 1
42699 Solingen

tel +49 (0) 212.233628.47
fax +49 (0) 212.233628.79
mail Tobias.Trelle@codecentric.de
twitter @tobiastrelle



www.codecentric.de

blog.codecentric.de/en/author/tobias-trelle

www.xing.com/net/mongodb-dus