

1.– 4. September 2014
in Nürnberg



Herbstcampus

Wissenstransfer
par excellence

Kein blosses Schema?

NoSQL mit JPA und Hibernate OGM

Thomas Much

muchsoft.com

NoSQL ...

- Herzlich Willkommen in der NoSQL-Welt!

**Not
Only**  **SQL**

NoSQL und Big Data

- NoSQL
 - nicht-relationale Datenbanken, verteilt, skalierbar
 - oft Schema-frei
 - ⇒ „Structured Storage“
 - oft keine ACID-Transaktionen
 - ⇒ „weak consistency“, „eventual consistency“ (BASE) o.ä.
- Big Data
 - Datenmengen: Terabyte 10^{12} , Petabyte 10^{15} , Exabyte 10^{18}
 - Algorithmen wie MapReduce



Besser „NoREL“?

NoSQL ...

- Vier Arten von NoSQL-Datenbanken (YMMV):
 - **Document**
 - CouchDB, MongoDB, ..., Lucene, ...
 - **Graph**
 - Neo4j, ...
 - **Key-Value**
 - Redis, Infinispan, EhCache, ...
 - **Column**
 - BigTable, HBase, Cassandra, ...

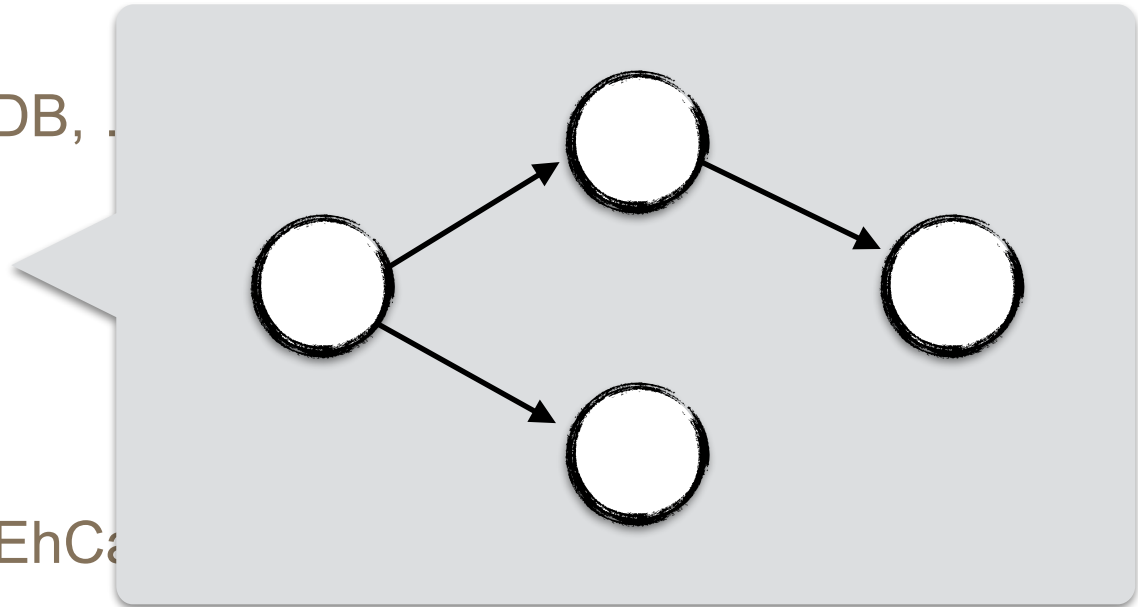
NoSQL ...

- Vier Arten von NoSQL-Datenbanken
 - **Document**
 - CouchDB, MongoDB, ...
 - Graph
 - Neo4j, ...
 - Key-Value
 - Redis, Infinispan, EhCache, ...
 - Column
 - BigTable, HBase, Cassandra, ...

```
{ "id": "1234",  
  "name": "Thomas Much",  
  "adressen": [  
    { "prio": "1",  
      "email": "info@muchsoft.com" },  
    { "prio": "2",  
      "email": "much@me.com" }  
  ]  
}
```

NoSQL ...

- Vier Arten von NoSQL-Datenbanken (YMMV):
 - Document
 - CouchDB, MongoDB, ...
 - **Graph**
 - Neo4j, ...
 - Key-Value
 - Redis, Infinispan, EhCache
 - Column
 - BigTable, HBase, Cassandra, ...



NoSQL ...

- Vier Arten von NoSQL-Datenbanken (YMMV):
 - Document
 - CouchDB, MongoDB, Lucene....
 - Graph
 - Neo4j, ...
 - **Key-Value**
 - Redis, Infinispan, EhCache
 - Column
 - BigTable, HBase, Cassandra

Key	Value
123	„Thomas“
124	„info@muchsoft.com“
250	“ ”
1000	1,95583

NoSQL ...

- Vier Arten von NoSQL-Datenbanken (YMMV):
 - Document
 - CouchDB, MongoDB, Lucene.
 - Graph
 - Neo4j, ...
 - Key-Value
 - Redis, Infinispan, EhCache
 - **Column**
 - BigTable, HBase, Cassandra

1 iOS: (A: 6.0 ⌚, B: 7.0 ⌚, C: 7.1 ⌚)

2 iOS: (B: 7.0.6 ⌚) User: (B: Thomas ⌚)

3 Sprache: (B: Java ⌚, E: JavaScript ⌚)

NoSQL – kurz zusammengefasst

- Kein JDBC, kein SQL
- Kein festes Schema, unterschiedlichste Datenmodelle
- Keine garantierten Transaktionen
- Hohe Performance-Anforderungen

- Das macht NoSQL-Datenbanken zum perfekten Backend für...

Ähem.

NoSQL mit ...

Hibernate ? ? ?

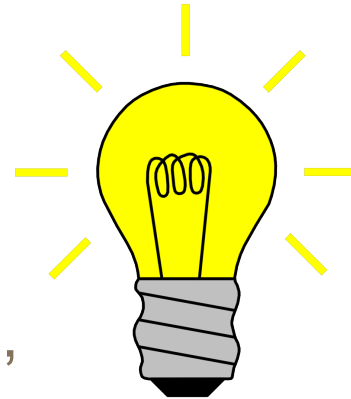
Hibernate?

- **Hibernate ORM** (Object Relational Mapping) \approx „Hibernate“
- Bildet *Domänenmodelle* auf relationale Datenbanken ab; benötigt JDBC(-Treiber) und SQL
- Version 1.0 schon 2002
Referenzimplementierung für JPA 1.0 (2006)
Unterstützt JPA 2.0 seit 3.5 (2010), JPA 2.1 seit 4.3 (2013)
- De-facto-Standard für ORM in Java
mit *Hibernate-API* bzw. mit *Java-Persistence-API*

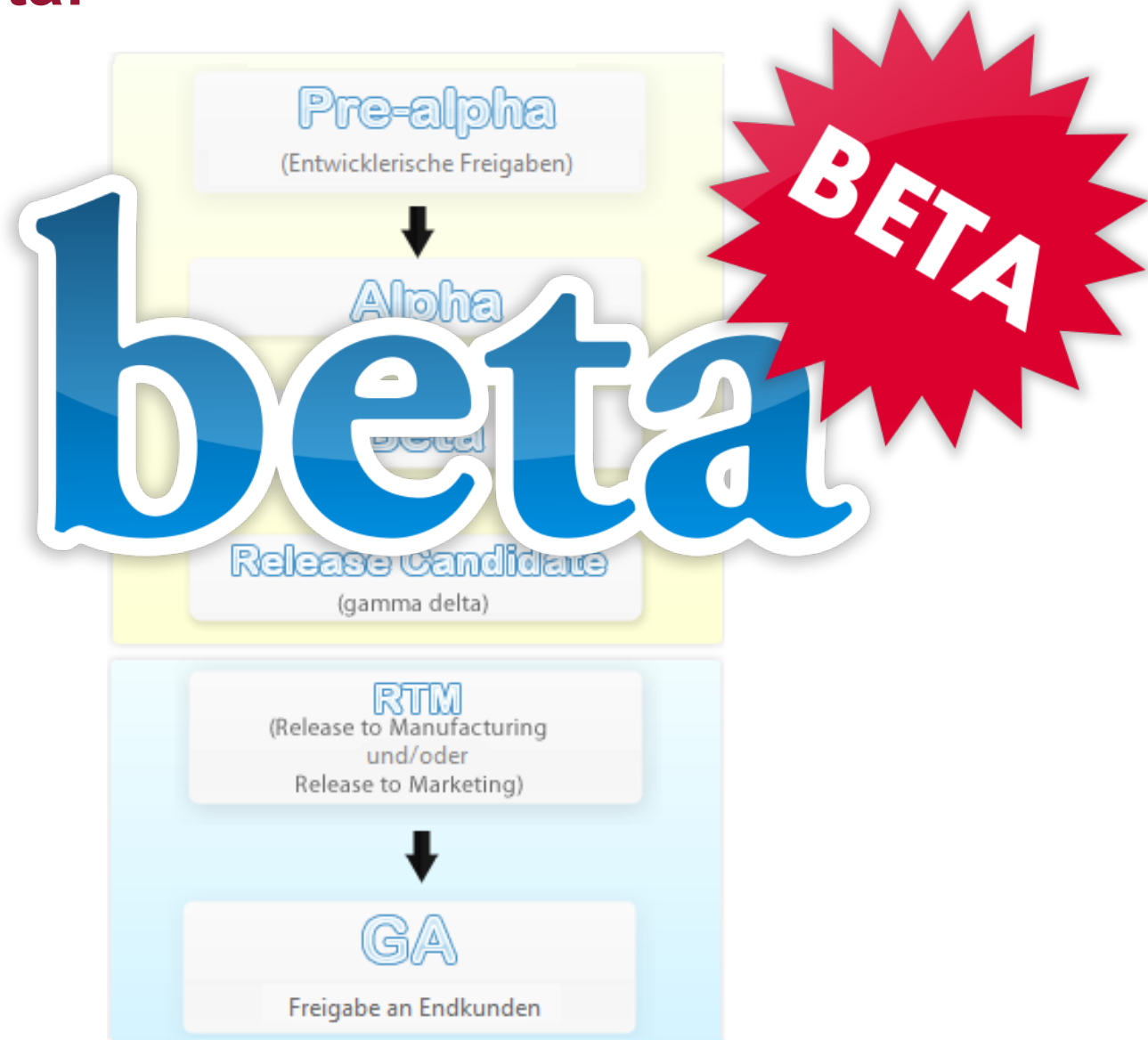
- Neben ORM gibt es weitere Hibernate-Projekte:
Search, Validator, OGM ...

Hibernate OGM!

- **Hibernate OGM** („Object Grid Mapping“)
 - wegen „Data Grids“ als eine der NoSQL-Kategorien
- Idee:
ORM-Kern verwenden (Entity-Mapping, Abfragen etc.),
aber auf NoSQL-Datenbanken abbilden
- Vorteil:
Standard- (JPA) oder zumindest bekannte (Hibernate) API
- Seit 2011 in der Entwicklung
- Aktuelle Version 4.1-Beta (bisher noch kein Final-Release)

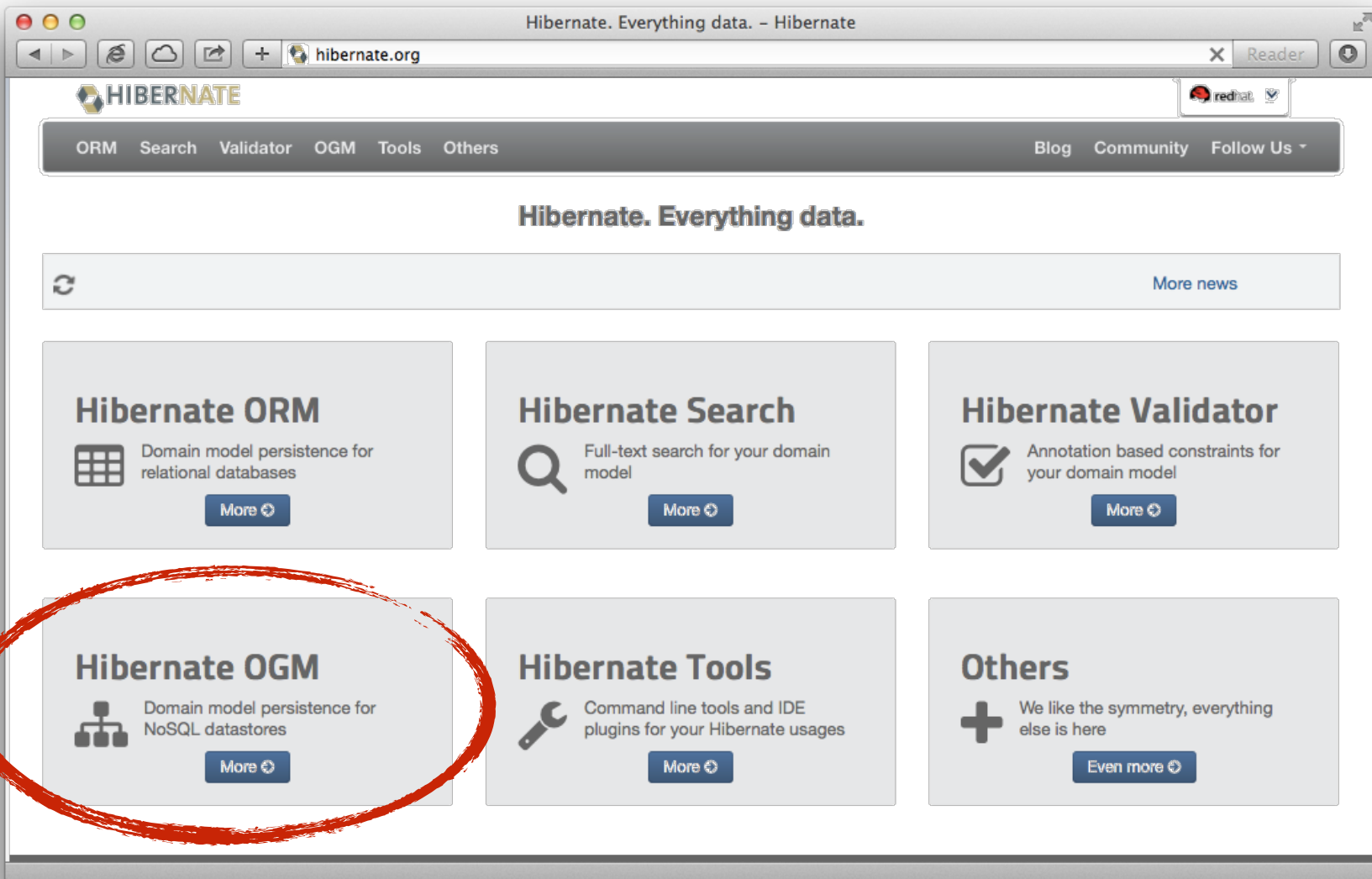


Achtung, Beta!



[http://de.wikipedia.org/wiki/Entwicklungsstadium_\(Software\)#Beta-Version](http://de.wikipedia.org/wiki/Entwicklungsstadium_(Software)#Beta-Version)

Beta – aber bereits Frontpage-Projekt



Unterstützte NoSQL-Datenbanken

- Derzeit



- Geplant



Hibernate OGM – Features

- Anbindung diverser NoSQL-Produkte über *Dialekte*
 - auch gleichzeitig in einer Anwendung
 - auch gemischt mit SQL-Datenbanken
- Abfragen (Queries)
 - JPQL (wird in native Abfragen konvertiert)
 - Native Queries
 - Volltext-Abfragen (mittels Hibernate Search)
- Produkt(familien)-spezifische API (ergänzend zu JPA)

JPA und NoSQL – passt das?

- JPA = *Java Persistence* (nicht „Java Relational Persistence“)
 - auch wenn der Standard derzeit nur relationale DBs behandelt
- Viele Konzepte passen:
`@Embeddable` (Komponenten), `@ElementCollection`
- Vieles lässt sich ohne große Verrenkungen abbilden:
Assoziationen, Unterklassen
- Voraussetzung: Domänenmodell
 - ähnliche Abwägung wie bei JPA vs. JDBC

NoSQL mit JPA-Entites

```
@Entity
@Table(name = "artikel")
public class Artikel {
    @Id @GeneratedValue(generator = "uuid")
    @GenericGenerator(name = "uuid", strategy = "uuid2")
    private String id;

    private String name;
    private String beschreibung;
    private Double preis;

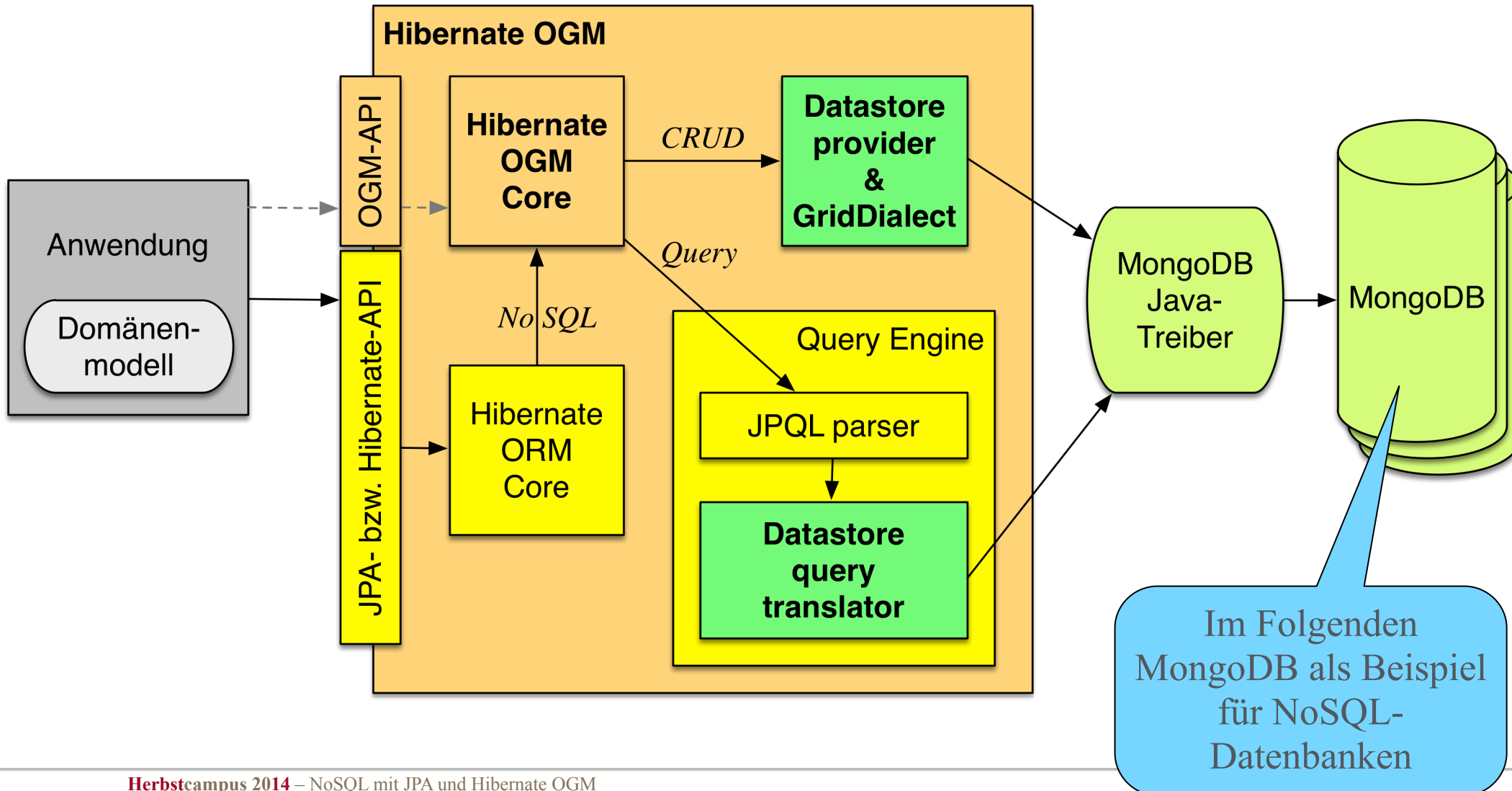
    @ManyToOne
    private Farbe farbe;
```

Getter/Setter etc. ausgelassen

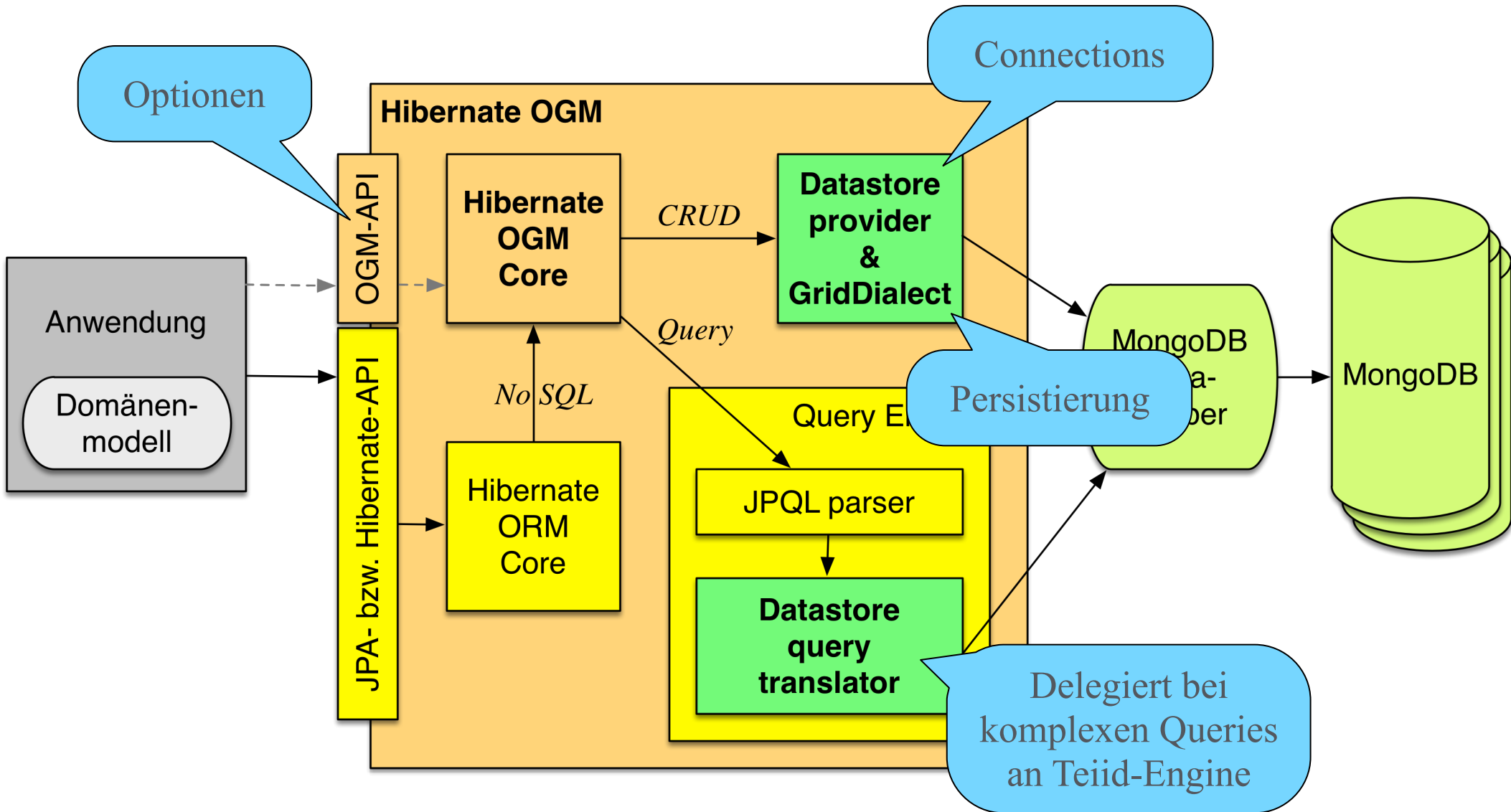
JPA und NoSQL – passt das?

- **Vorteile:**
 - Abstraktion auf Entity-Klassen-Ebene
(zur Laufzeit viele Optimierungen durch Hibernate möglich)
 - Bekannte API + Semantik
 - Späte Wahl des konkreten NoSQL-Produkts möglich!
- **Nachteile:**
 - Abstraktion (Performance?)
- Ähnliche Diskussion wie bei pro/contra Hibernate (JPA)

Hibernate OGM – Architektur



Hibernate OGM – Architektur



Entities persistieren...

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("meine-mongodb-pu");

TransactionManager transactionManager =
    com.arjuna.ats.jta.TransactionManager.transactionManager();

transactionManager.begin();

EntityManager em = emf.createEntityManager();

Artikel artikel = new Artikel("iMac",
    "iMac 27\"", 16 GB RAM, 1 TB Fusion Drive", 2999.0);

em.persist( artikel );

em.flush();

em.close();

transactionManager.commit();
```

... mit Transaktionen???

- MongoDB unterstützt keine TAs (so wie viele NoSQL-DBs)
 - Atomarität nur auf Dokument-Ebene
- JPA-TA-Klammer trotzdem empfohlen!
(Aber nicht erzwungen)
- Relevant für die NoSQL-Datenbanken, die Transaktionen unterstützen (z.B. Neo4j mit ACID-TAs)
- Bei MongoDB für automatisches Flush beim TA-Commit
 - Rollback hier aber nicht möglich

Konfiguration in persistence.xml (1)

```
<?xml version="1.0"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">

  <persistence-unit name="meine-mongodb-pu" transaction-type="JTA">

    <provider>
      org.hibernate.ogm.jpa.HibernateOgmPersistence
    </provider>

    <jta-data-source>java:/DefaultDS</jta-data-source>

    <properties> ... </properties>

  </persistence-unit>
</persistence>
```

Nur im Java-EE-Container
nötig (wg. Java EE Spec)

Konfiguration in persistence.xml (2)

```
<properties>

  <property name="hibernate.transaction.jta.platform"
    value="org.hibernate...JBossStandAloneJtaPlatform" />

  <property name="hibernate.ogm.datastore.provider"
    value="mongodb" />

  <property name="hibernate.ogm.datastore.database"
    value="test" />

  <property name="hibernate.ogm.datastore.host" value="127.0.0.1"/>
  <property name="hibernate.ogm.datastore.port" value="27017" />

  <!-- username, password, connection_timeout ... -->

</properties>
```

MongoDB

```
StraylightMBA:~ much$ mongod
mongod --help for help and startup options
Fri Mar 14 12:31:26.052 [initandlisten] MongoDB starting : pid=1564 port=27017 dbpath=/data/db/
64-bit host=StraylightMBA.local
Fri Mar 14 12:31:26.052 [initandlisten]
Fri Mar 14 12:31:26.052 [initandlisten] ** WARNING: soft rlimits too low. Number of files is 25
6, should be at least 1000
Fri Mar 14 12:31:26.052 [initandlisten] db version v2.4.9
Fri Mar 14 12:31:26.052 [initandlisten] git version: 52fe0d21959e32a5bdbecdc62057db386e4e029c
Fri Mar 14 12:31:26.052 [initandlisten] build info: Darwin bs-osx-106-x86-64-2.10gen.cc 10.8.0
Darwin Kernel Version 10.8.0: Tue Jun 7 16:32:41 PDT 2011; root:xnu-1504.15.3~1/RELEASE_X86_64
x86_64 BOOST_LIB_VERSION=1_49
Fri Mar 14 12:31:26.052 [initandlisten] allocator: system
Fri Mar 14 12:31:26.052 [initandlisten] options: {}
Fri Mar 14 12:31:26.053 [initandlisten] journal dir=/data/db/journal
Fri Mar 14 12:31:26.053 [initandlisten] recover : no journal files present, no recovery needed
Fri Mar 14 12:31:26.082 [websvr] admin web console waiting for connections on port 28017
Fri Mar 14 12:31:26.083 [initandlisten] waiting for connections on port 27017
```

```
StraylightMBA:~ much$ mongo
MongoDB shell version: 2.4.9
connecting to: test
Server has startup warnings:
Fri Mar 14 12:31:26.052 [initandlisten]
Fri Mar 14 12:31:26.052 [initandlisten] ** WARNING: soft rlimits too low.
Number of files is 256, should be at least 1000
> show dbs
local  0.078125GB
test   (empty)
> use test
switched to db test
> show collections
>
```



Live
Demo

Artikel speichern ...

```
@Entity
@Table(name = "artikel")
public class Artikel {
    @Id @GeneratedValue(generator = "uuid")
    @GenericGenerator(name = "uuid", strategy = "uuid2")
    private String id;

    private String name;
    private String beschreibung;
    private Double preis;

    public Artikel() {}

    public Artikel(String name, String beschreibung, Double preis)...
```



Mongo-
Shell
(JSON)

... Mongo-Shell ... (JavaScript!)

```
> db
test

> show collections
artikel
system.indexes

> db.artikel.find()
{ "_id" : "bf391a40-b1a3-4b7f-a92b-f5aa8fbbd155",
  "preis" : 876.54, "name" : "iPhone 6",
  "beschreibung" : "256 GB, Farbe: Blue Dalmatian" }
{ "_id" : "ae5b8cf9-3a3b-45ff-b763-9f7f1c254af4",
  "preis" : 2999, "name" : "iMac", "beschreibung" :
  "iMac 27\", 16 GB RAM, 1 TB Fusion Drive" }

> db.artikel.findOne()
{
  "_id" : "bf391a40-b1a3-4b7f-a92b-f5aa8fbbd155",
  "preis" : 876.54,
  "name" : "iPhone 6",
  "beschreibung" : "256 GB, Farbe: Blue Dalmatian"
}
```



JPA-
Entity

... Artikel laden (mit JPQL)

```
transactionManager.begin();  
EntityManager em = emf.createEntityManager();  
  
TypedQuery<Artikel> q =  
    em.createQuery("select a from Artikel a", Artikel.class);  
  
List<Artikel> artikelListe = q.getResultList();  
  
for (Artikel a : artikelListe) { ... }  
  
em.flush();  
em.close();  
transactionManager.commit();
```

Datensätze ohne OGM / JPA

```
> art = { name: "iWatch", preis: 1499.9 }  
{ "name" : "iWatch", "preis" : 1499.9 }
```

```
> db.artikel.insert( art )
```

```
> db.artikel.find()
```

```
{ "_id" : "bf391a40-b1a3-4b7f-a92b-f5aa8fbbd155", "preis" :  
876.54, "name" : "iPhone 6", "beschreibung" :  
"Blue Dalmatian" }  
{ "_id" : "ae5b8cf9-3a3b-45ff-b763-9f7f1c25  
2999", "name" : "iMac", "beschreibung" : "iM  
1 TB Fusion Drive" }  
{ "_id" : ObjectId("5329fa368b95bcac0b9e729e"), "name" :  
"iWatch", "preis" : 1499.9 }
```

MongoDB-ObjectId
kann beim Laden von OGM
nicht gemappt werden...

Kommt in 4.1 Beta 7
(OGM-526)

```
Exception in thread "main" javax.persistence.PersistenceException: org.hibernate.PropertyAccessException: could not set a field value for model.Artikel.id
```

```
Caused by: java.lang.IllegalArgumentException: Can not set java.lang.String field model.Artikel.id to org.bson.types.ObjectId
```

Produktspezifische Abfragen

```
Query q = em.createNativeQuery("{name:'iMac'}", Artikel.class);

Query q = em.createNativeQuery(
    "{$and: [ {name:'iMac'}, {preis:2999} ]}", Artikel.class);

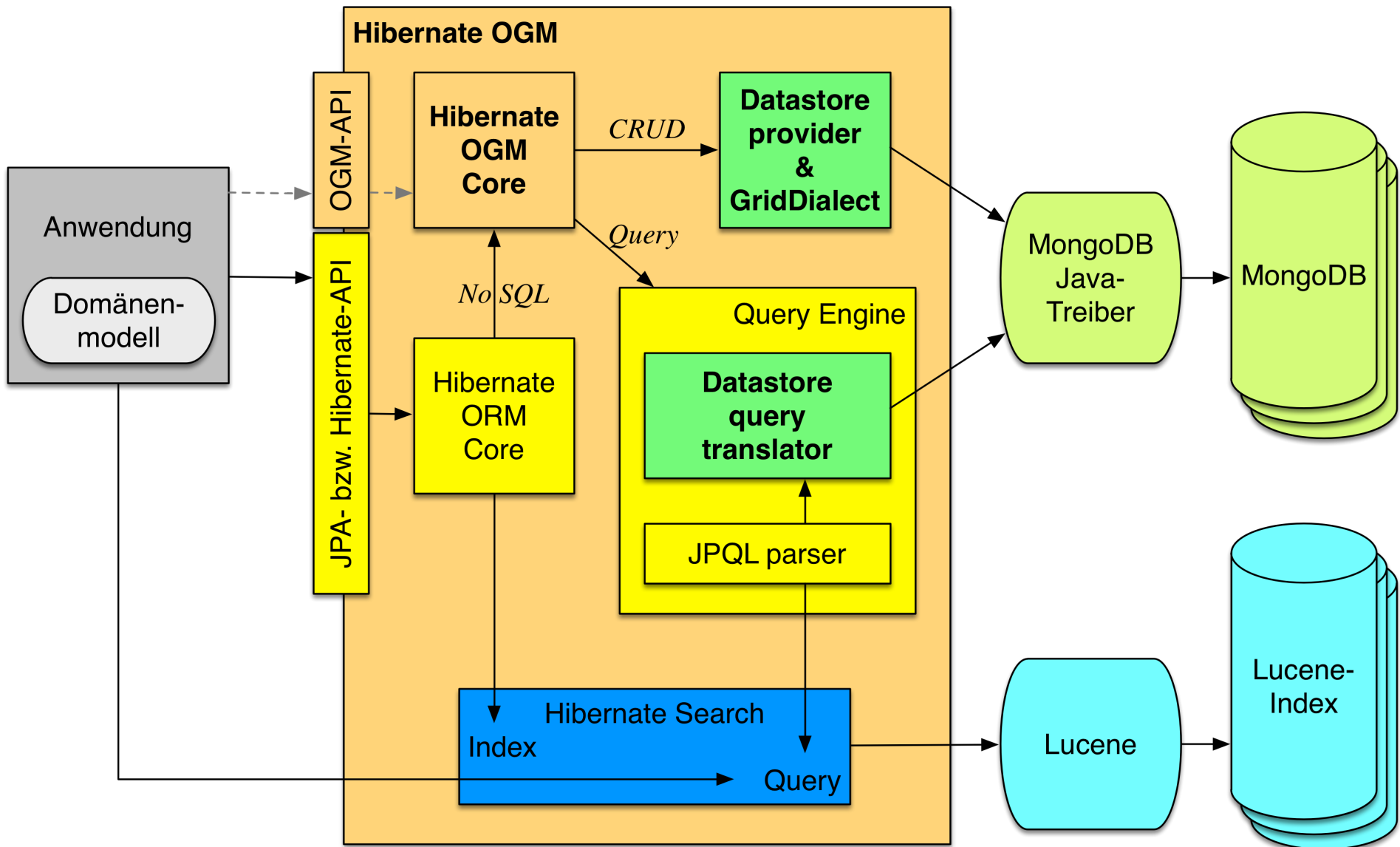
Query q = em.createNativeQuery(
    "{$query: {name:'iMac'}, $orderby: {name:1}}", Artikel.class);

Query q = em.createNamedQuery("alleImacs");

Artikel imac = (Artikel) q.getSingleResult();
```

```
@Entity
@Table(name = "artikel")
@NamedNativeQuery(
    name = "alleImacs",
    query = "{$query: {name:'iMac'}, $orderby: {name:1}}",
    resultClass = Artikel.class)
public class Artikel { ... }
```


Hibernate OGM / Search – Architektur



Volltext-Index mit Hibernate Search

@Indexed

@Entity

```
public class Artikel {
```

```
    @Id ... private String id;
```

```
    @Field(analyze = Analyze.NO)
```

```
    private String name;
```

```
    @Field(analyze = Analyze.YES)
```

```
    private String beschreibung;
```

```
    ...
```

```
}
```

```
<property
  name="hibernate.search.default.directory_provider"
  value="filesystem" />
<property
  name="hibernate.search.default.indexBase"
  value="/data/search" />
```

- Damit ist bereits eine JPQL-Abfrage in NoSQL-Datenbanken möglich, die keine eigene Abfragesprache besitzen!

Abfragen mit Hibernate Search

```
EntityManager em = ...;
```

```
FullTextEntityManager ftem =  
    Search.getFullTextEntityManager( em );
```

```
QueryBuilder b =  
    ftem.getSearchFactory().buildQueryBuilder()  
        .forEntity(Artikel.class).get();
```

```
Query luceneQuery =  
    b.keyword().onFields("name", "beschreibung")  
        .matching("imac").createQuery();
```

```
FullTextQuery ftQuery =  
    ftem.createFullTextQuery(luceneQuery, Artikel.class);
```

```
ftQuery.initializeObjectsWith(ObjectLookupMethod.SKIP,  
    DatabaseRetrievalMethod.FIND_BY_ID);
```

```
List<Artikel> artikelListe = ftQuery.getResultList();
```

```
...
```

Etwas mehr Mapping, bitte

- Eingebettete Komponenten
 - Listen von Werten / Komponenten
 - Assoziationen
 - Vererbung
-
- Im Folgenden: Entity mit Mapping & JSON in der Datenbank

Eingebettete Komponenten...

```
@Entity
@Table(name = "artikel")
public class ArtikelMitVariante {
    @Id ... uuid ...
    private String id;

    private String name;
    private String beschreibung;
    private Double preis;

    @Embedded
    private Variante variante;
    ...
}
```



Mongo-
Shell
(JSON)

```
@Embeddable
public class Variante {
    private String groesse;
    private String farbe;
    ...
}
```

... Eingebettete Komponenten

```
> db.artikel.find()
{ "_id" : "bf391a40-b1a3-4b7f-a92b-f5aa8fbbd155",
  "preis" : 876.54, "name" : "iPhone 6",
  "beschreibung" : "256 GB, Farbe: Blue Dalmatian" }
...
{ "_id" : "df40aa17-005e-4915-a83b-63617ada831f",
  "beschreibung" : "Schönes T-Shirt", "name" : "T-
  Shirt", "preis" : 19.95, "variante" : { "farbe" :
  "orange", "groesse" : "S" } }

> db.artikel.findOne( {name:"T-Shirt"} )
{
  "_id" : "df40aa17-005e-4915-a83b-63617ada831f",
  "beschreibung" : "Schönes T-Shirt",
  "name" : "T-Shirt",
  "preis" : 19.95,
  "variante" : {
    "farbe" : "orange",
    "groesse" : "S"
  }
}
```



JPA-
Entity



Collection
als Artikel
laden?

Listen von Werten / Komponenten...

```
@Entity
@Table(name = "artikel")
public class ArtikelMitGroessen {
    @Id ... uuid ...
    private String id;

    private String name;
    private String beschreibung;
    private Double preis;

    @ElementCollection
    private List<String> groessen = new ArrayList<>();
    ...
}
```



Mongo-
Shell
(JSON)

... Listen von Werten / Komponenten

```
> db.artikel.findOne( {name:"Poloshirt"} )
{
  "_id" : "710b2cf5-9463-4243-bd93-86163b387903",
  "beschreibung" : "Buntes Poloshirt",
  "groessen" : [
    "L",
    "M",
    "S",
    "XL"
  ],
  "name" : "Poloshirt",
  "preis" : 14.95
}
```



JPA-
Entity

Assoziationen...

```
@Entity
@Table(name = "artikel")
public class ArtikelMitFarbe {
    @Id ... uuid ...
    private String id;

    private String name;
    private String beschreibung;
    private Double preis;

    @ManyToOne(cascade = CascadeType.PERSIST)
    private Farbe farbe;
    ...
}
```

```
@Entity
@Table(name = "farben")
public class Farbe {
    @Id
    private String name;
    ...
}
```



Mongo-
Shell
(JSON)

... Assoziationen

```
> db.artikel.find()
```

```
{ "_id" : "1ea30dc7-005b-4eaf-a325-d6df08059501",  
  "farbe_name" : "rot", "preis" : 9.95, "name" : "T-  
Shirt 1a", "beschreibung" : "Rotes T-Shirt" }
```

```
{ "_id" : "fbe4b0cb-d6c7-4d50-a890-9c25cc9d70fc",  
  "farbe_name" : "rot", "preis" : 17.95, "name" : "T-  
Shirt 1b", "beschreibung" : "Rotes T-Shirt" }
```

```
{ "_id" : "44c543a8-d8e4-4de4-8066-753fa5e4949b",  
  "farbe_name" : "grün", "preis" : 19.95, "name" : "T-  
Shirt 2", "beschreibung" : "Grünes T-Shirt" }
```

```
> db.farben.find()
```


```
{ "_id" : "rot" }
```

```
{ "_id" : "grün" }
```



JPA-
Entities

(MongoDB-)spezifische Konfiguration

- 
- Verknüpfungsinformationen für Assoziationen werden standardmäßig in der Entity abgelegt.
 - Optional in separater Assoziations-Kollektion:

```
import org.hibernate.ogm.datastore.document.options.*;
import org.hibernate.ogm.datastore.mongodb.options.*;

@Entity
@Table(name = "artikel")
@AssociationStorage(
    AssociationStorageType.ASSOCIATION_DOCUMENT)
@AssociationDocumentStorage(
    AssociationDocumentType.COLLECTION_PER_ASSOCIATION)
public class ArtikelMitFarbe { ... }
```

Funktioniert derzeit
noch nicht wirklich...

- Konfiguration alternativ über Properties oder mit der Options-API.

Vererbung...

```
@Entity
@Table(name = "artikel")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Artikel { ... }
```

DB-Collection ist „artikel“ –
funktioniert also eher wie
SINGLE_TABLE

Andere Vererbungs-
Strategien werden laut
Doku *nicht* unterstützt

```
@Entity
public class ArtikelMitDatum extends Artikel {
    private Date datum;
    ...
}
```

Mongo-
Shell
(JSON)

... Vererbung

```
> show collections
```

```
artikel
```

```
> db.artikel.find()
```

```
{ "_id" : "7a6cc2d4-85db-44cf-afa2-847a155aec10",  
  "preis" : 876.54, "DTYPE" : "Artikel", "name" :  
  "iPhone 6", "beschreibung" : "256 GB, Farbe: Blue  
  Dalmatian" }
```

```
{ "_id" : "8036b24a-37aa-43e8-9306-b558a5f4d8e3",  
  "preis" : 2999, "DTYPE" : "Artikel", "name" : "iMac",  
  "beschreibung" : "iMac 27\", 16 GB RAM, 1 TB Fusion  
  Drive" }
```

```
{ "_id" : "5500c09d-d611-4f4c-99bc-7cfb5549880d",  
  "preis" : 19.95, "DTYPE" : "ArtikelMitDatum", "name" :  
  "T-Shirt", "beschreibung" : "Schönes T-Shirt", "datum"  
  : ISODate("2014-03-20T20:48:15.543Z") }
```

DTYPE ist jetzt auch
bei normalen Artikeln
erforderlich...
Work in Progress?!

JPA-
Entities

Hibernate-API

- Anstelle der Java-Persistence-API kann auch die proprietäre Hibernate-API verwendet werden
 - Verzicht auf JPA-Standard aber bitte gut abwägen...

```
Configuration cfg = new OgmConfiguration();  
  
cfg.setProperty("hibernate.ogm.datastore.provider", "mongodb");  
...  
  
cfg.addAnnotatedClass( ... );  
  
SessionFactory sf = cfg.buildSessionFactory();  
...
```

Andere NoSQL-Datenbanken

- Map – für Unit-Tests
 - Infinispan – (sehr) gute Unterstützung, JTA
 - EhCache – gute Unterstützung, (derzeit) keine TA-Nutzung
 - Neo4J – experimentell, JTA / ACID;
in den letzten Beta-Versionen deutlich verbessert
 - CouchDB – experimentell, keine TA
-
- Zur Konfiguration der unterstützten NoSQL-Datenbanken stehen spezielle Properties (und teilweise auch Annotationen) zur Verfügung.

Ausblick / Roadmap nach OGM 4.1

- Hibernate OGM 4.2
 - benutzerdefinierte Typen
 - gleichzeitiges Schreiben in mehrere Backends (No+SQL)
 - Cassandra? HBase?
 - Anpassung an Hibernate ORM 5
- Hibernate OGM 4.3
 - „Polyglotte Persistenz“
 - Queries verbessern (Joins, Map/Reduce)
- danach
 - (Nicht-)Schema-Migrationen (mit Annotationen und/oder API)

Hibernate OGM – Fazit (1)

- Hibernate OGM übernimmt relationale Konzepte, sofern sie sinnvoll für NoSQL-Datenbanken sind:
 - Domänenmodell als Struktureinheit
 - Primärschlüssel
 - „Fremdschlüssel“ für Verbindungen zwischen Entities
- Mit diesen Konzepten hat das relationale Modell vor über 30 Jahren die damalige Datenbank-Welt vereinheitlicht
 - bei OGM aber ohne strenge Konsistenzprüfungen
 - „Convention over Constraints“ 😊

Hibernate OGM – Fazit (2)

- Derzeit können JPA-Entities schon gut in NoSQL-Datenbanken gespeichert werden
 - in einem Format, das auch für andere (Nicht-OGM-) Anwendungen gut lesbar ist
- Das Einlesen beliebiger fremd-geschriebener NoSQL-Datenbanken klappt dagegen nicht immer
- teilweise noch buggy, Features teilweise noch unvollständig
- Performance gut testen
 - ist bei großen Assoziationen derzeit nicht wirklich gut

Hibernate OGM – Fazit (3)

- Spannende – weil erstaunlich gut passende – Verwendung von vorhandener Standard-API (JPA) für aktuellen NoSQL-Trend
- Trotz Beta schon einen Blick wert!
 - auch wenn bis zur Final noch einiges zu tun ist (mitmachen! 😊)
- Produktiveinsatz muss derzeit noch sehr gut geprüft werden
 - evtl. sind die proprietären APIs derzeit noch geeigneter / stabiler
- Guter (einfacher!) Einstieg in die NoSQL-Welt
 - gerade auch für Unternehmen mit etwas „traditionellerer“ IT



1.– 4. September 2014
in Nürnberg



Herbstcampus

Wissenstransfer
par excellence

Vielen Dank! 😊

Thomas Much

www.muchsoft.com

