



Hibernate OGM

Michal Linhard

Quality Assurance Engineer, Red Hat

Red Hat Developer Conference

February 17th, 2012

About me

- Quality Assurance Engineer at JBoss / Red Hat
- Formerly played with JBoss AS / EAP
- Now having fun with Infinispan / Enterprise Datagrid
- Performance / system resilience tests in clustered environment



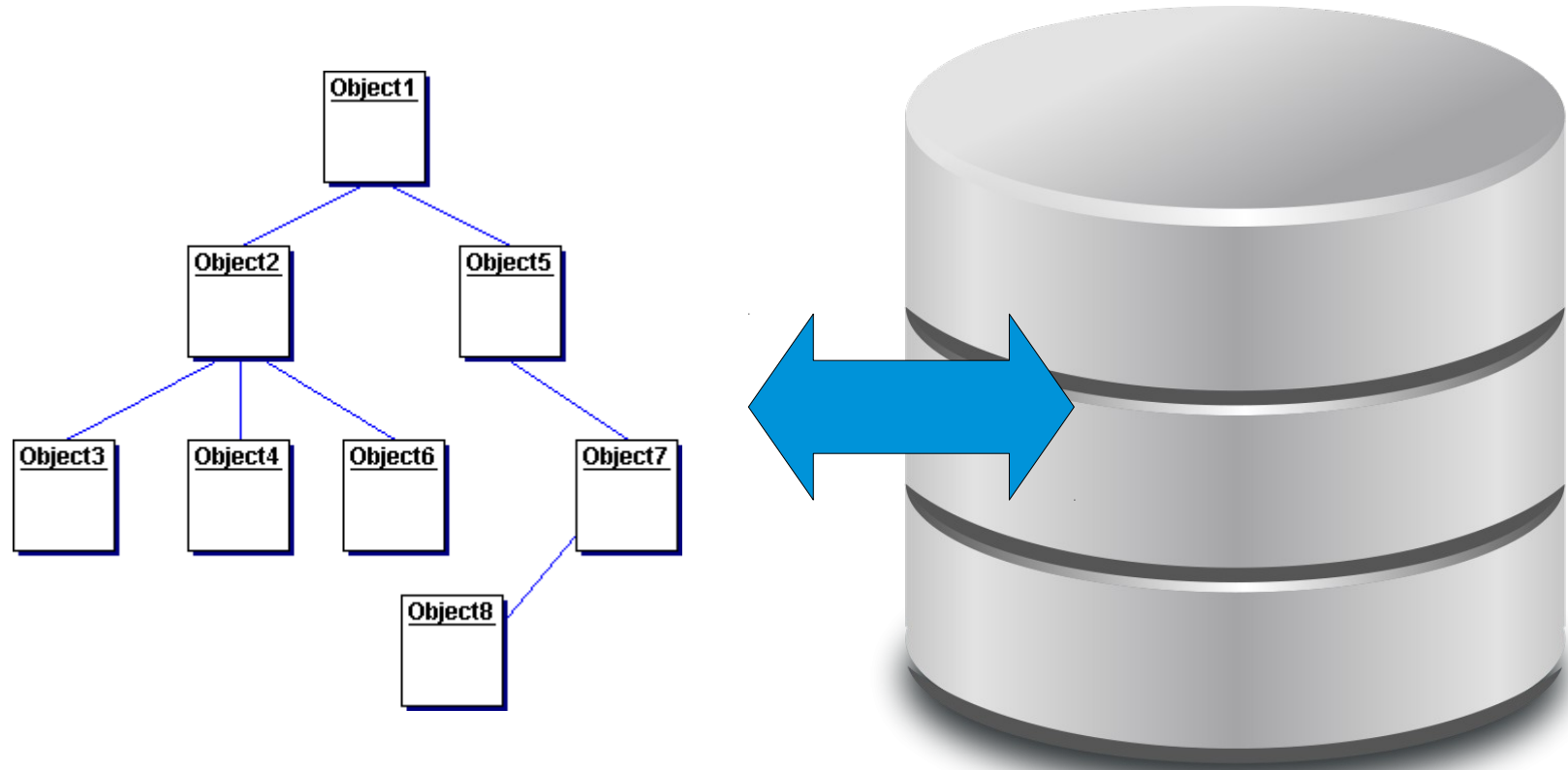




OGM = Object/Grid Mapper



We need to store objects



JEE has a well known solution: JPA

← javax.persistence.* annotations

```
@Entity
@Table(uniqueConstraints = @UniqueConstraint(columnNames = "email"))
public class Member implements Serializable {

    @Id
    @GeneratedValue
    private Long id;

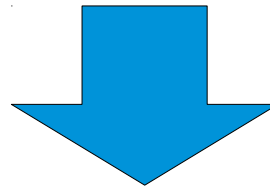
    @NotNull
    @Size(min = 1, max = 25)
    @Pattern(regexp = "[A-Za-z ]*", message = "must contain only letters and spaces")
    private String name;

    @NotNull
    @NotEmpty
    @Email
    private String email;

    ...
}
```



ORM = Object-Relational Mapping



id [PK] bigint	email character varying(255)	name character varying(255)	phone_number character varying(12)
0	john.smith@mailinator.com	John Smith	2125551212
1	michal@linhard.sk	Michal Linhard	421908380703



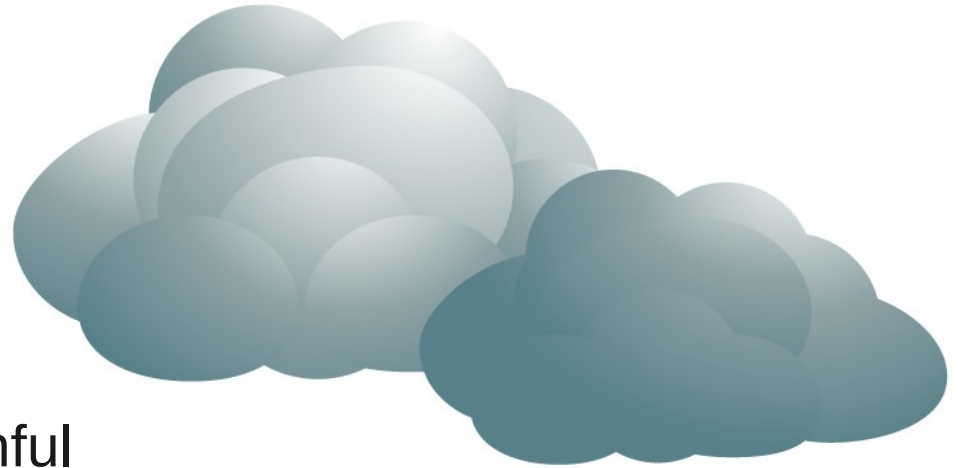
Relational Databases

- Brought peace and order for 30 years
- Data structure abstraction
- Safety net
- Transactions
- Referential integrity
- (simple) types
- Proven usefulness
- Tuning, backup, resilience



Relational databases

- (Some) limitations:
 - plan for scale is hard
 - data model changes are painful
- New needs
 - limitless data for later analysis
 - risk of being successful
- Cloud



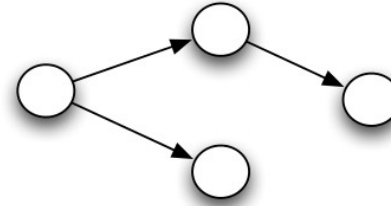
NoSQL Alternative (**Not Only SQL**)

- Goals
 - very different
 - large dataset
 - high availability
 - low latency / higher throughput
 - specific data access pattern



Not Only SQL

- Document based stores
- Column based
- Graph oriented databases
- Key / value stores



key	value
123	Address@23
126	"Booya"

```
{ "user" : {  
  "id": "124",  
  "name": "Emmanuel",  
  "addresses" : [  
    { "city": "Paris", "country": "France" },  
    { "city": "Atlanta", "country": "USA" }  
  ]  
}
```



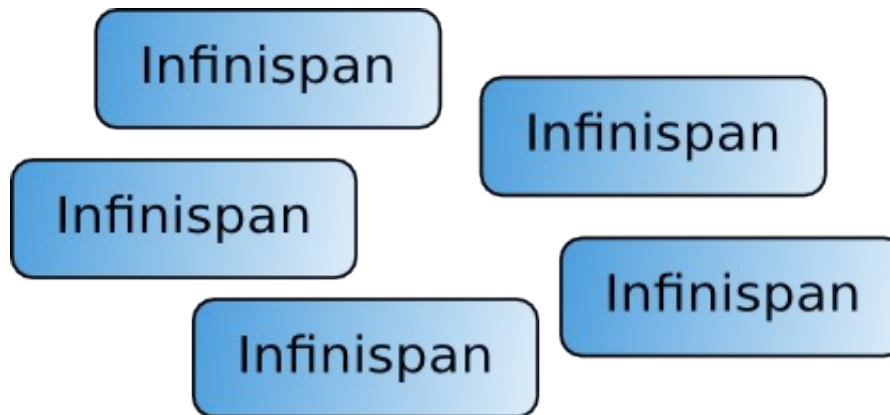
NoSQL continued ... Flexibility at a cost

- Programming model
 - one per product :(
- no schema => app driven schema
- query (Map Reduce, specific DSL, ...)
- transaction
- durability / consistency



Introducing Infinispan

- Clustered in-memory Key/Value store
- Each node is equal, scale by adding or killing nodes
- No bottlenecks, by design
- Cloud network friendly
 - Uses JGroups as communication layer



Introducing Infinispan

- Support for transactions
- Cache loaders (Cassandra, JDBC, Amazon S3, ...)
- Lucene integration
- Some Hibernate integrations
 - Second level cache
 - Hibernate Search indexing backend

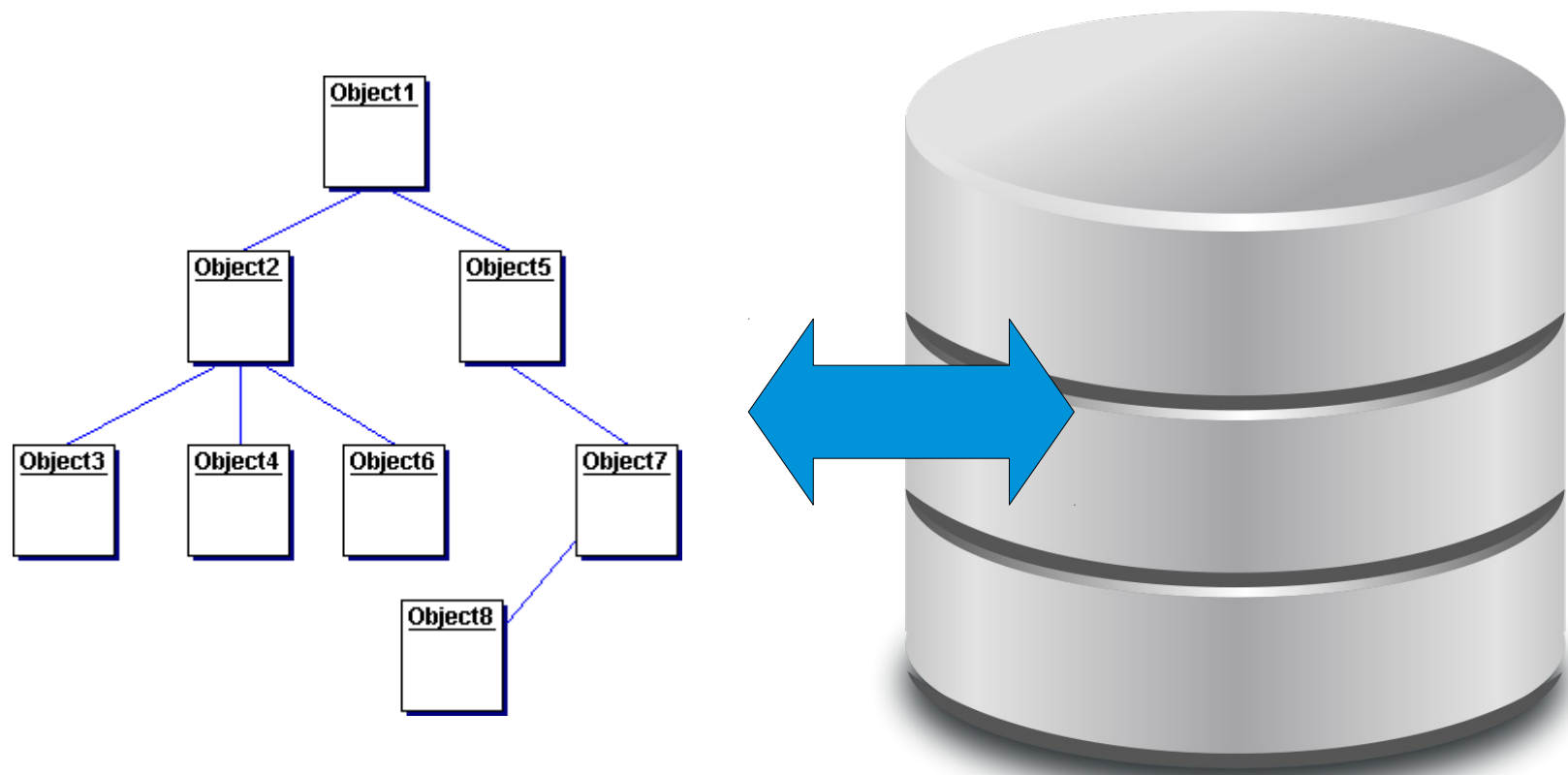


Introducing Infinispan

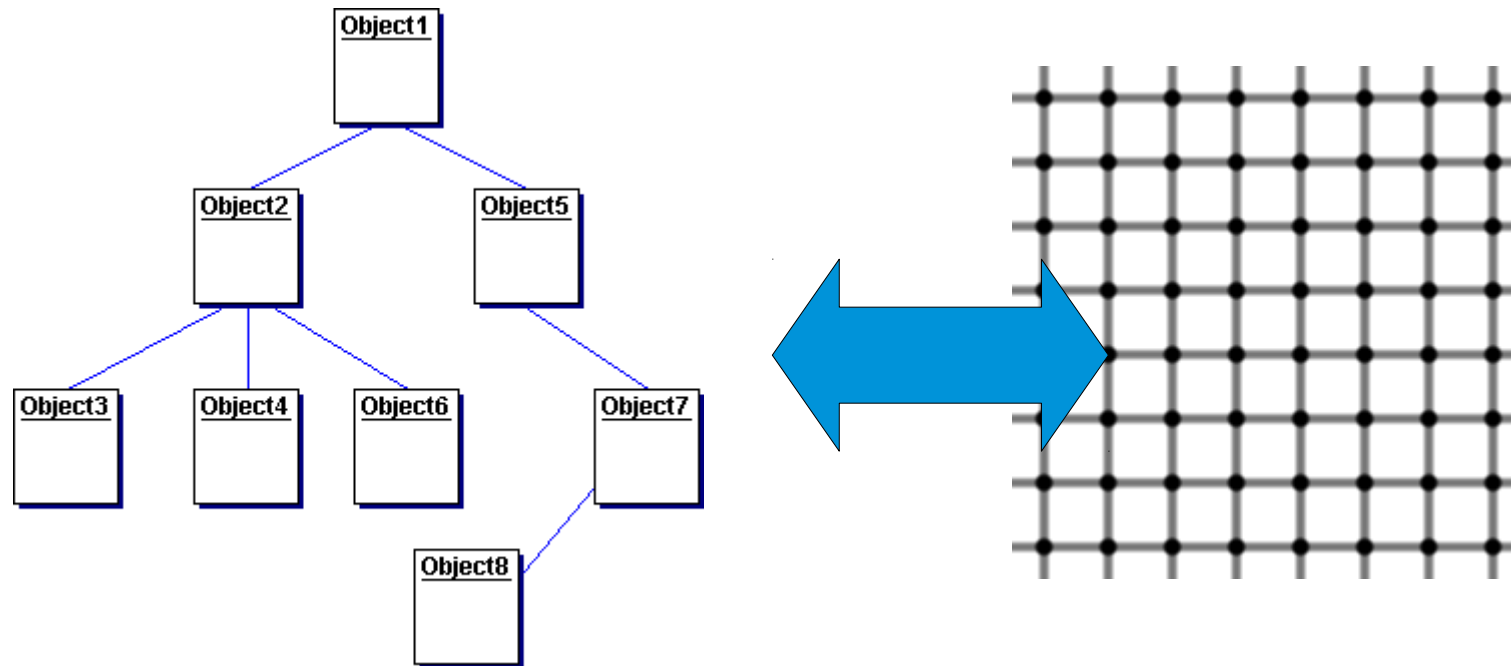
- It's a `java.util.concurrent.ConcurrentMap` !
- `map.put("user-34", userInstance);`
- `map.get("user-34");`
- `map.remove("user-34");`
- `map.putIfAbsent("user-38", another);`



Back to the beginning ...



Wanna move to NoSQL Storage ?



But don't want to learn new API / Programming Model ?



Introducing HIBERNATE OGM

- JPA for NoSQL
- Encourage new data usage patterns
- Familiar environments
- Ease of use
- Easy to jump in / out
- Push NoSQL exploration in enterprises
- “PaaS for existing API” initiative

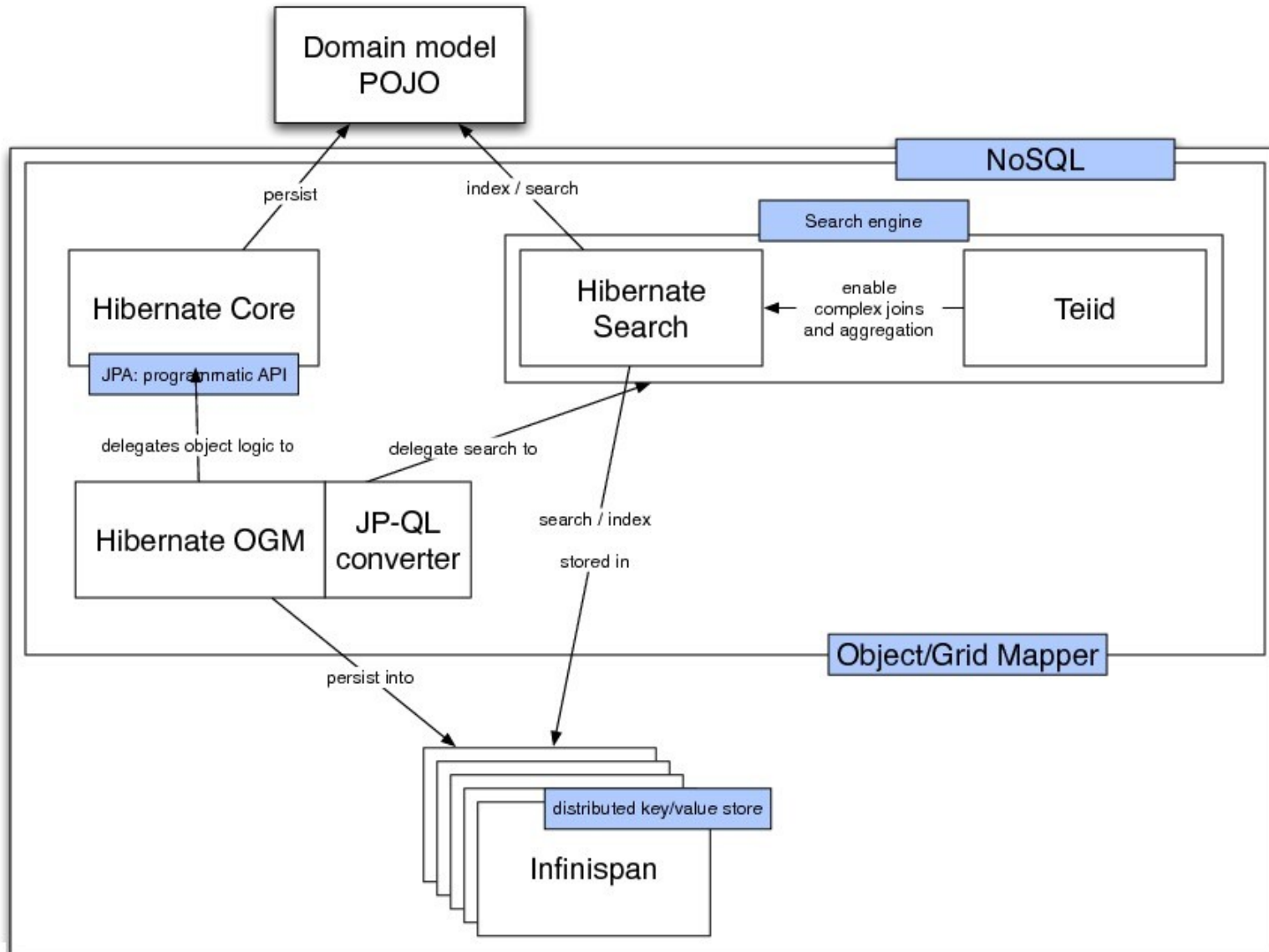


Introducing HIBERNATE OGM

- Currently works with Infinispan
- Object CRUD (+associations)
- Reuses Hibernate Search, Hibernate Core
- JP-QL queries
- Not a silver bullet



General Architecture



Introducing HIBERNATE OGM

Concepts



Schema or no schema?

- Schema-less
 - move to new schema very easy
 - app deal with old and new structure or migrate all data
 - need strict development guidelines
- Schema
 - reduce likelihood of rogue developer corruption
 - share with other apps
 - “didn’t think about that” bugs reduced



Entities as serialized blobs?

- Serialize objects into the (key) value
 - store the whole graph?
 - maintain consistency with duplicated objects
 - guaranteed identity $a == b$
 - concurrency / latency
 - structure change and (de)serialization, class definition changes



OGM's approach to schema

- Keep what's best from relational model
 - as much as possible
 - tables / columns / PKs
- Decorrelate object structure from data structure
- Data stored as (self-described) tuples
- Core types limited
 - portability



OGM's approach to schema

- Store metadata for queries
 - Lucene index
- CRUD operations are key lookups



Storage - Entities

- Entities are stored as tuples (Map<String, Object>)
- **Key** is composed of
 - table name, PK column names, PK values
- **Value** is Map<String, Object>
 - String: column name
 - Object: simple type (serializable)
 - e.g. {id=1, name="Charlie", date-of-birth=23-03-1983}

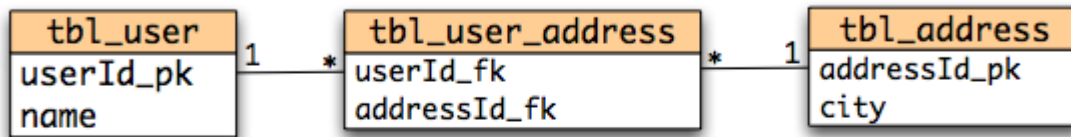


Storage - Associations

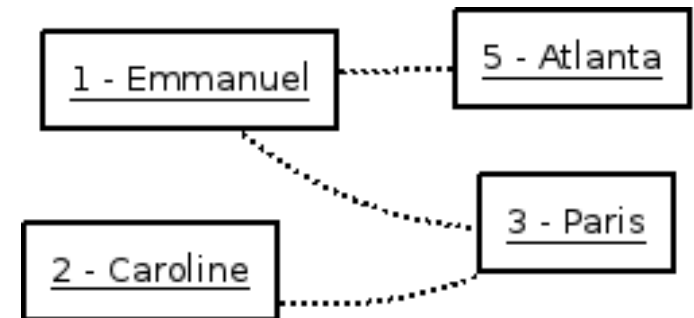
- Cannot store exactly like relational DBs (key lookup)
 - still tuple based
- Each association in two keys (both sides)
 - table name, FK column names, FK values (for a given side)
- Value is the list of tuples
- Focus on speedy reads
 - association writes involve two key lookups



How does it work?



key	value
tbl_user,userId_pk,1	{userId_pk=1,name="Emmanuel"}
tbl_user,userId_pk,2	{userId_pk=2,name="Caroline"}
tbl_address,addressId_pk,3	{addressId_pk=3,city="Paris"}
tbl_address,addressId_pk,5	{addressId_pk=5,city="Atlanta"}
tbl_user_address,userId_fk,1	{ {userId_fk=1, addressId_fk=3}, {userId_fk=1, addressId_fk=5} }
tbl_user_address,userId_fk,2	{ {userId_fk=2, addressId_fk=3} }
tbl_user_address,addressId_fk,5	{ {userId_fk=1, addressId_fk=5} }
tbl_user_address,addressId_fk,3	{ {userId_fk=1, addressId_fk=3}, {userId_fk=2, addressId_fk=3} }



Queries

- Hibernate Search indexes entities
- Store Lucene indexes in Infinispan
- JP-QL to Lucene query transformation
- Works for simple queries
 - Lucene is not a relational SQL engine



Queries

select a from Animal a where a.size > 20

```
> animalQueryBuilder
  .range().onField("size").above(20).excludeLimit()
  .createQuery();
```

select u from Order o join o.user u where o.price > 100 and u.city = "Paris"

```
> orderQB.bool()
  .must(
    orderQB.range()
    .onField("price").above(100).excludeLimit().createQuery() )
  .must(
    orderQB.keyword("user.city").matching("Paris").createQuery())
  .createQuery();
```



Conclusion

- JPA for NoSQL
- Reusing mature projects
- Keep the good of the relational model
- Do queries too
- Alpha quality
- Quite promising and exciting
- No Silver bullet



More info

- Project page
 - <http://www.hibernate.org/subprojects/ogm.html>
- Code
 - <https://github.com/hibernate/hibernate-ogm/>

Questions ?

