tarantool

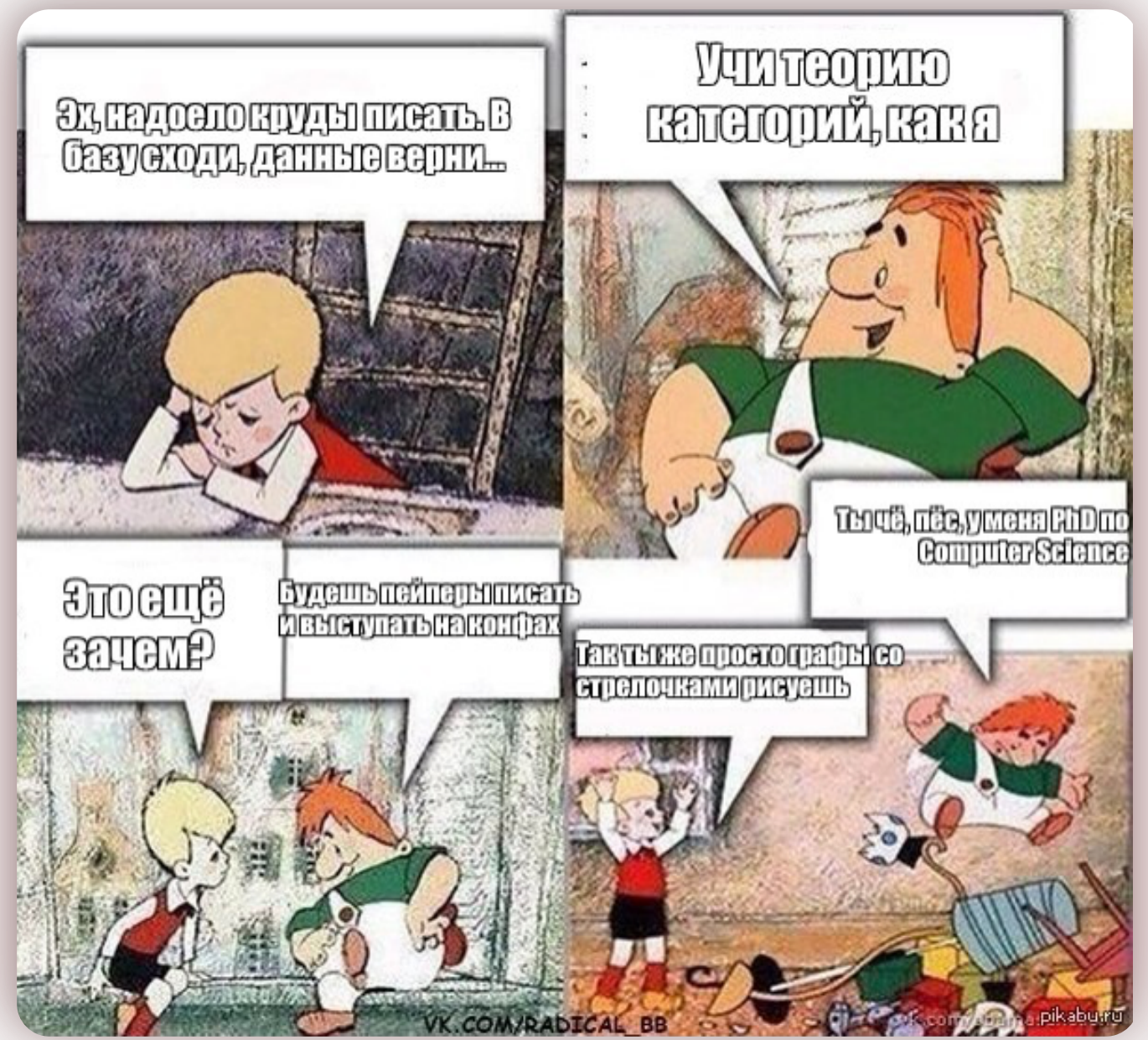
developed by @ mail

Working with sharded data with the taste of Spring Data

Alexey Kuzin

Introduction

The problem


Эх, надоело круды писать. В базу сходи, данные верни...
Учи теорию категорий, как я
Ты чё, пёс, у меня PhD по Computer Science
Это ещё зачем?
Будешь пейперы писать и выступать на конфах
Так тыже просто графы со стрелочками рисуешь
VK.COM/RADICAL_BB
pikabu.ru

Create

Read

Update

Delete

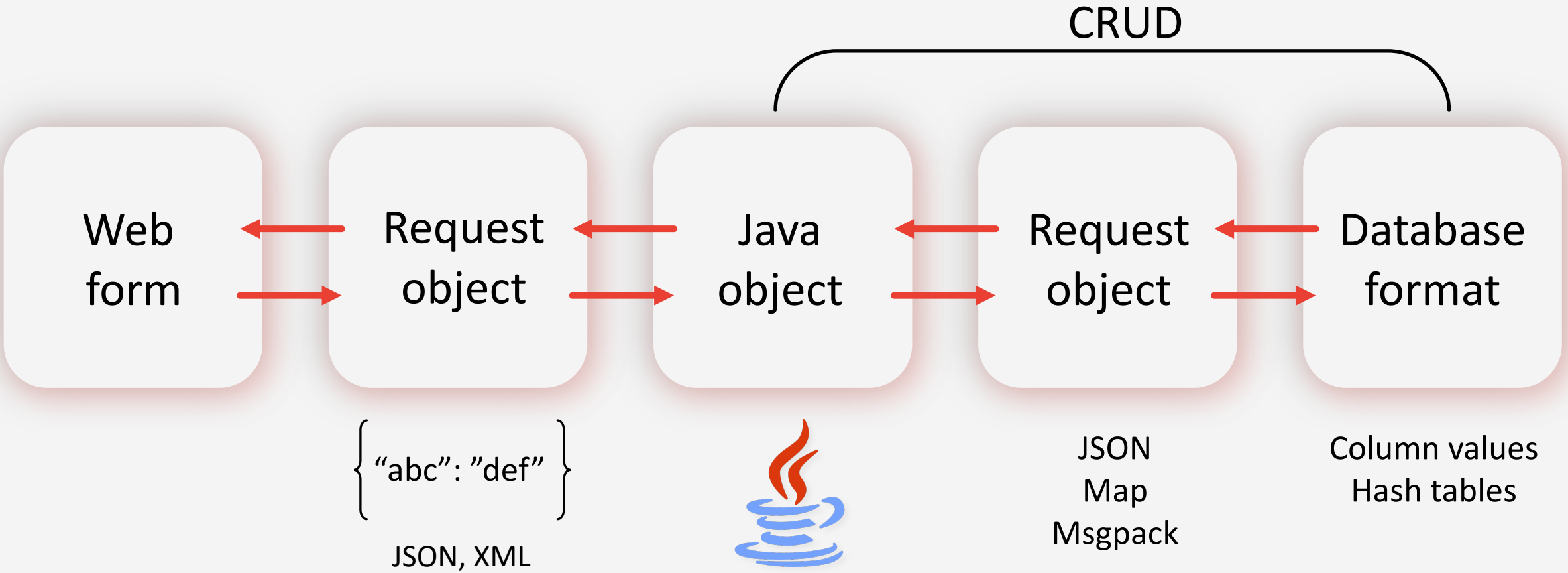
REST API

Persistent Storage

Typical 3-layer architecture

In practice

Typical information exchange flow within the system



Typical code with JPA EntityManager


```
@Repository
@Transactional(readOnly = true)
class JpaCustomerRepository implements CustomerRepository {

    @PersistenceContext
    private EntityManager em;

    @Override
    @Transactional
    public Customer save(Customer customer) {
        if (customer.getId() == null) { em.persist(customer);
        return customer;
    } else {
        return em.merge(customer); }
    }

    @Override
    public Customer findByEmailAddress(EmailAddress emailAddress) {
        TypedQuery<Customer> query = em.createQuery(
            "select c from Customer c where c.emailAddress = :emailAddress", Customer.class);
        query.setParameter("emailAddress", emailAddress);
        return query.getSingleResult();
    }
}
```

The Growth problem

Maintaining the system throughput

Problems:

Growing amount of data,
more transactions

Evolving data model

Need to maintain the same SLA or
make the respond times faster

Examples: online support chats,
multiplayer video games,
online retail

Solutions:

1. Optimizing the communication with DB and/or vertical scaling

2. Changing the data model and/or horizontal scaling

3. Changing the datastore technology

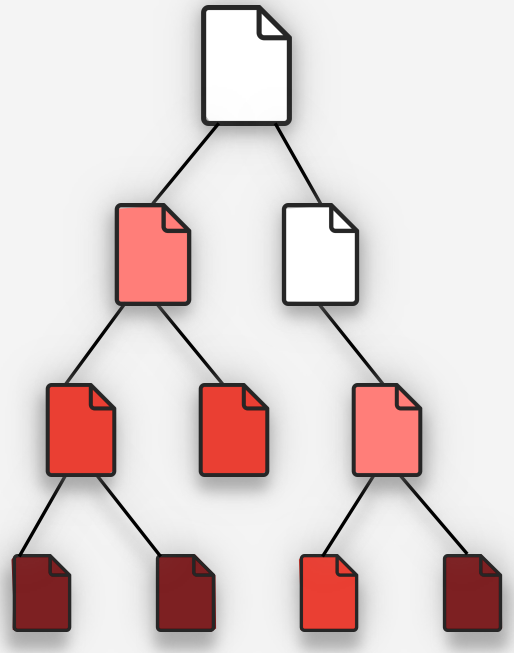
4. A combination of any of the variants above

Data migration
Code refactoring

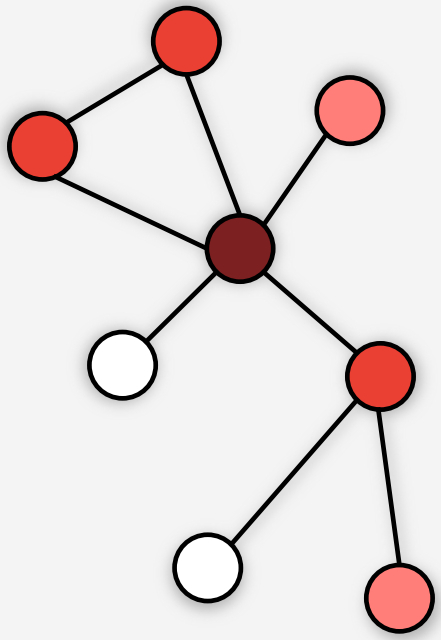
NoSQL Databases

Different data models and features

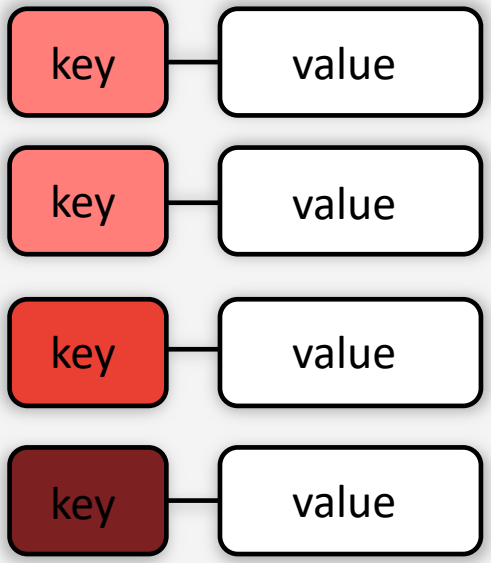
Document



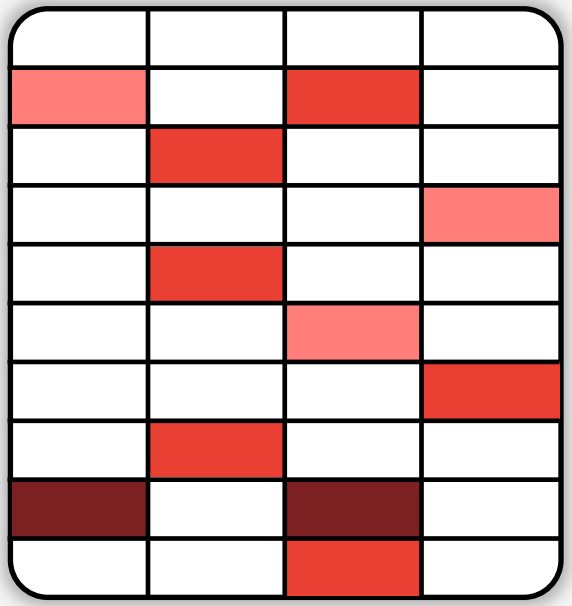
Graph



Key-Value



Wide-column



Some databases support ANSI SQL, but no single, unified query language

Special query languages supporting the key features

Some databases expand the SQL set with custom features

ClickHouse:

SAMPLE, PREWHERE

Hive:

CLUSTER BY, DISTRIBUTE BY

Special query languages supporting the key features

Some databases use custom query languages

SQL

```
SELECT p2.person_name FROM people p1
JOIN friend ON (p1.person_id = friend.person_id)
JOIN people p2 ON (p2.person_id = friend.friend_id)
WHERE p1.person_name = 'Jack';
```

Graph queries (Cypher)

```
MATCH (p1:person)-[:FRIEND-WITH]-(p2:person)
WHERE p1.name = "Jack"
RETURN p2.name
```

Databases use their own protocols

Elasticsearch, MongoDB

Protocol: HTTP

Result: JSON

```
GET _xpack/sql?format=txt {
    "query": "SELECT avg(system.process.memory.size),
system.process.name FROM metricbeat* system.process.name GROUP BY
system.process.name",
    "fetch_size":5
}
```

Tarantool

Protocol: IPROTO (binary)

Query:

`conn:execute([[SELECT dd, дд AS д FROM t1;]])`

Result: MsgPack array

```
4                       MP_MAP, size 4
43                        IPROTO_STMT_ID
ce c2 3c 2c 1e            MP_UINT = statement id
34                        IPROTO_BIND_COUNT
00                        MP_INT = 0 = number of parameters to bind
33                        IPROTO_BIND_METADATA
90                        MP_ARRAY, size 0 = there are no parameters to bind
32                        IPROTO_METADATA
92                        MP_ARRAY, size 2 (i.e. 2 columns)
85                          MP_MAP, size 5 (i.e. 5 items for column#1)
00 a2 44 44                   IPROTO_FIELD_NAME + 'DD'
01 a7 69 6e 74 65 67 65 72    IPROTO_FIELD_TYPE + 'integer'
03 c2                         IPROTO_FIELD_IS_NULLABLE + false
04 c3                         IPROTO_FIELD_IS_AUTOINCREMENT + true
...                         ...
```

Scaling DB out: sharding

Original table

ID	NAME	AGE	DEPT
1	Brian	Goetz	Head
2	Doug	Lea	South
3	Joshua	Bloch	North

Vertical partitioning

ID	NAME	AGE
1	Brian	Goetz
2	Doug	Lea
3	Joshua	Bloch

ID	DEPT
1	Head
2	South
3	North

Horizontal partitioning

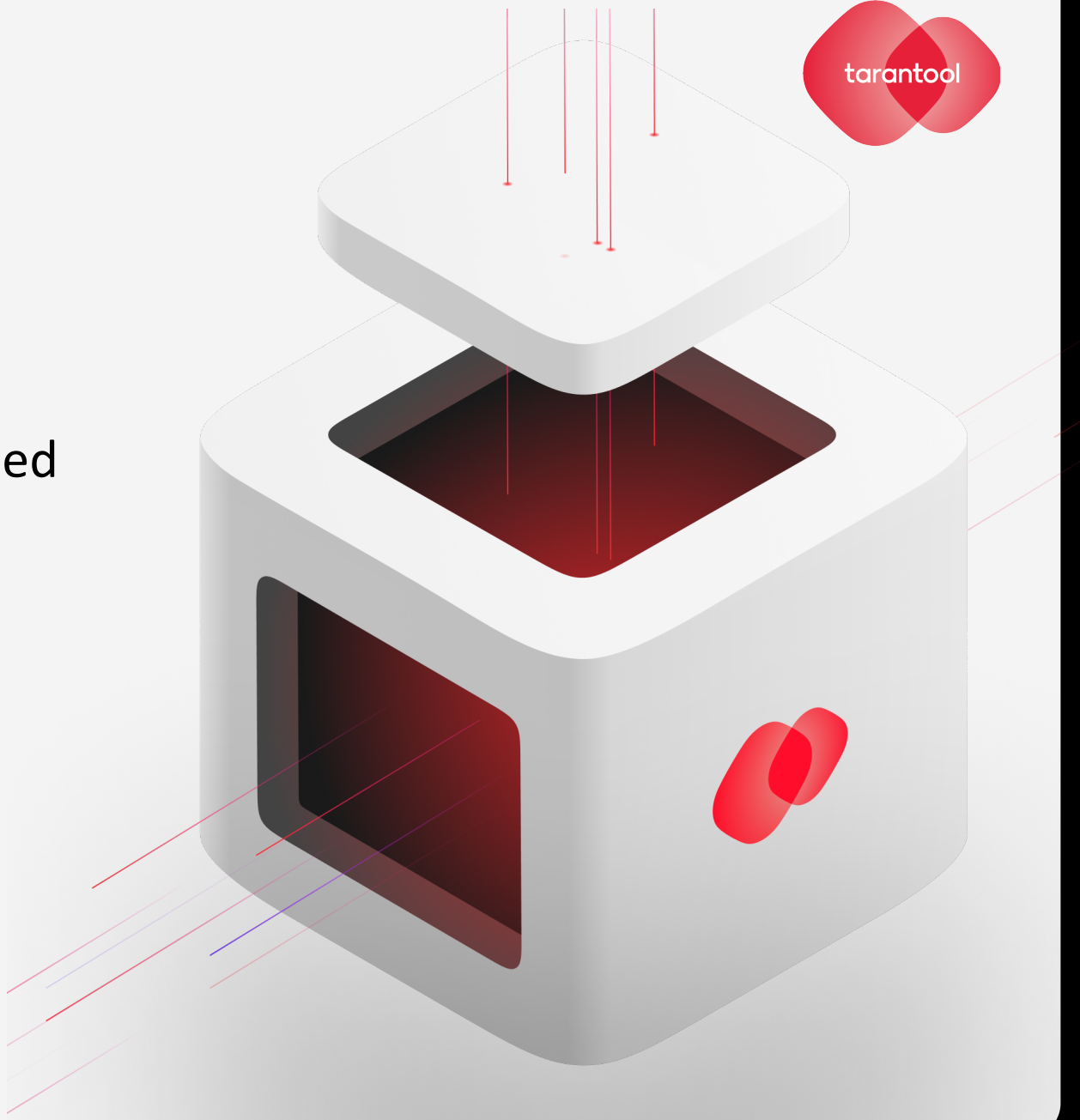
ID	SID	NAME	AGE	DEPT
1	1	Brian	Goetz	Head
3	1	Joshua	Bloch	North

ID	SID	NAME	AGE	DEPT
2	2	Doug	Lea	South

Sharding complexities

- Even writing declarative SQL queries
  we need to know how it will be performed

- User must access distributed database
  seamlessly as a non-distributed one



Spring Data: a programming model

Templates

Provides facades for operations, supported by the persistent storage

Repositories

Provides facades for CRUD operations

Object Mapping

Provides data conversion from raw data (JSON, driver-specific structures) into POJO

Spring Data Internals

Deal with it!

Writing your own module

1

How Spring Data repositories work

2

Dealing with sharding

3

Advantages and disadvantages of DB access without JPA

Repositories

@Repository – the old way

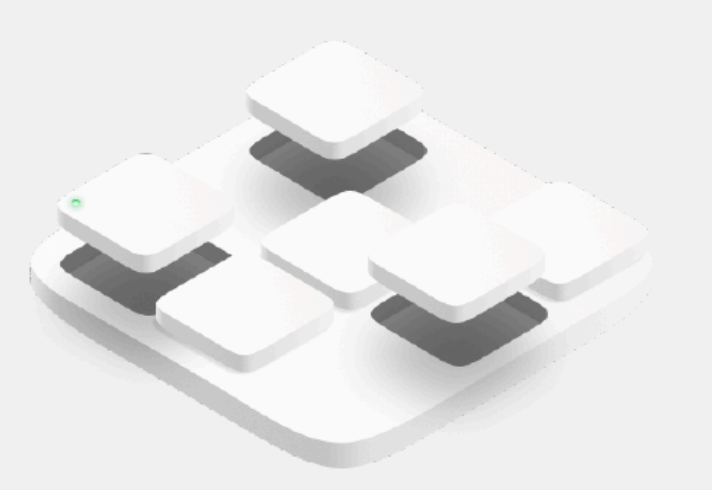
- Special case of @Component
- Provides special Exception translation

@EnableXXXRepositories

- Enables constructing of proxy beans for corresponding repositories
- Configures where to scan entities for the corresponding repositories

Interfaces

- `Repository<T, ID>`, `CrudRepository<T, ID>`, `PagingAndSortingRepository<T, ID>`
- Mark interfaces as candidates for proxy bean creation
- `@RepositoryDefinition` for completely custom interface



Repositories: configuration

1

EnableXXXRepositories provides

- List of packages for entities
- Basic entity interface (e.g. @Document, @Entity or @Tuple)
- Basic repository interface which is supported for proxying (e.g. TarantoolRepository)
- Query lookup strategy (default Key.CREATE_IF_NOT_FOUND)
- Repository factory and factory bean classes
- Where named queries are stored
- Custom properties

2

Proxy beans are created lazily

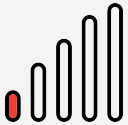
3

Proxy beans for repositories use default repository implementations

(e.g. SimpleTarantoolRepository) or explicit implementations (usually named like MyRepositoryImpl)

Repositories: defining query methods

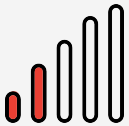
1



CrudRepository

CRUD methods (findById, findAll, create, update, delete)

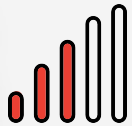
2



PagingAndSortingRepository

Pagination and sorting – PageRequest, Page, Sort parameters

3

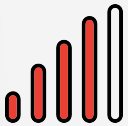


Derived queries

(from the method name) – depends on query lookup strategy

```
List<Person> findByLastnameLike(
    String lastname);
```

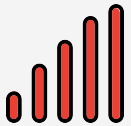
4



Property expressions

(SPEL, Mongo query documents, etc)

5



Special annotations and annotation properties

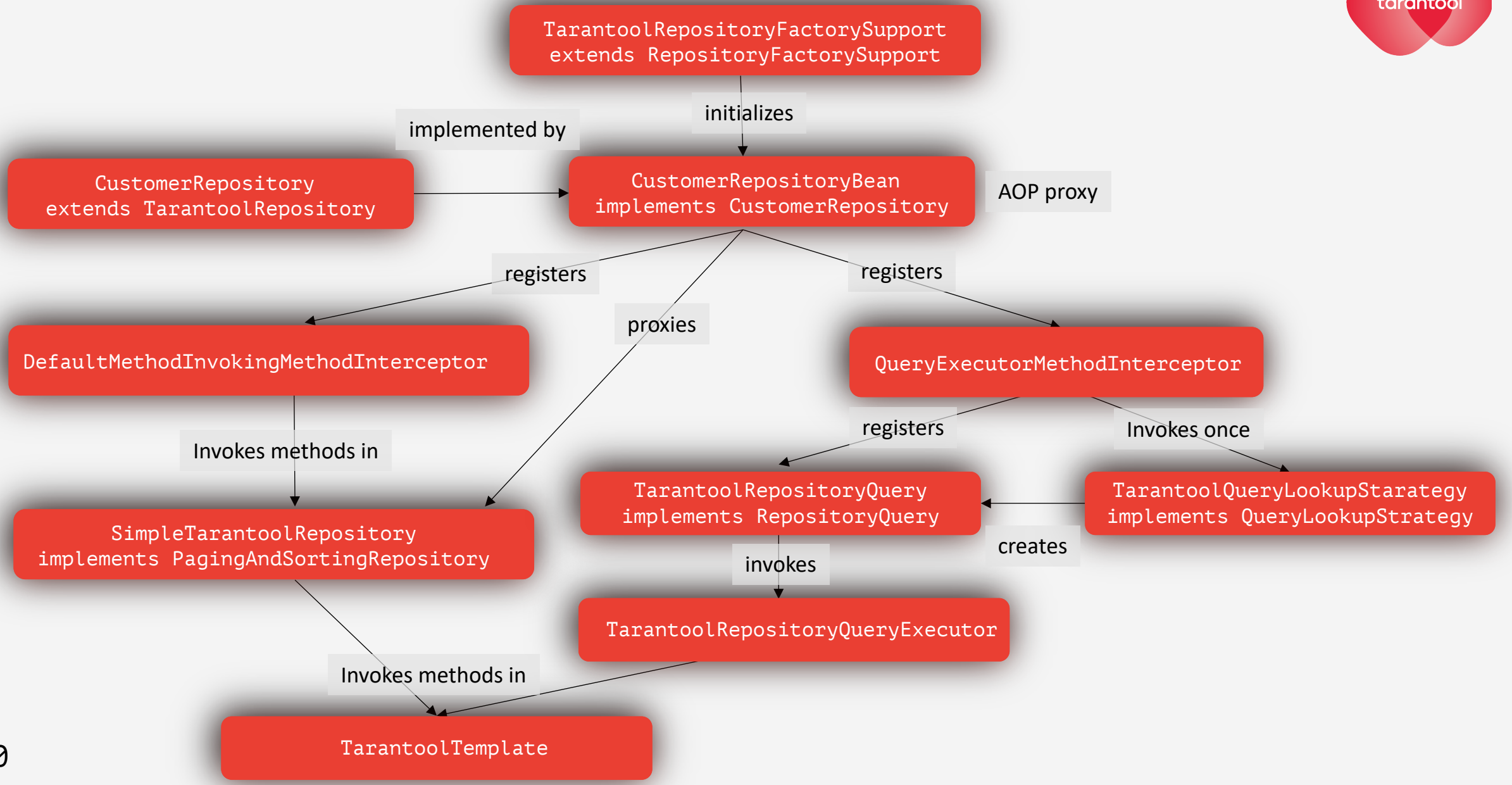
Mongo:

```
@ExistsQuery(value = "{ 'lastname' : ?0 }")
boolean someExistQuery(String lastname);
```

Tarantool:

```
@Query(function = "update_by_complex_query")
void updateYear(Integer id, Integer year);
```

Repositories: initialization (simplified)



Repositories: registering queries

Mongo:

mongo-named-queries.properties:

```
1: Person.findByNamedQuery={'firstname' : ?0}
```

PersonRepository.java:

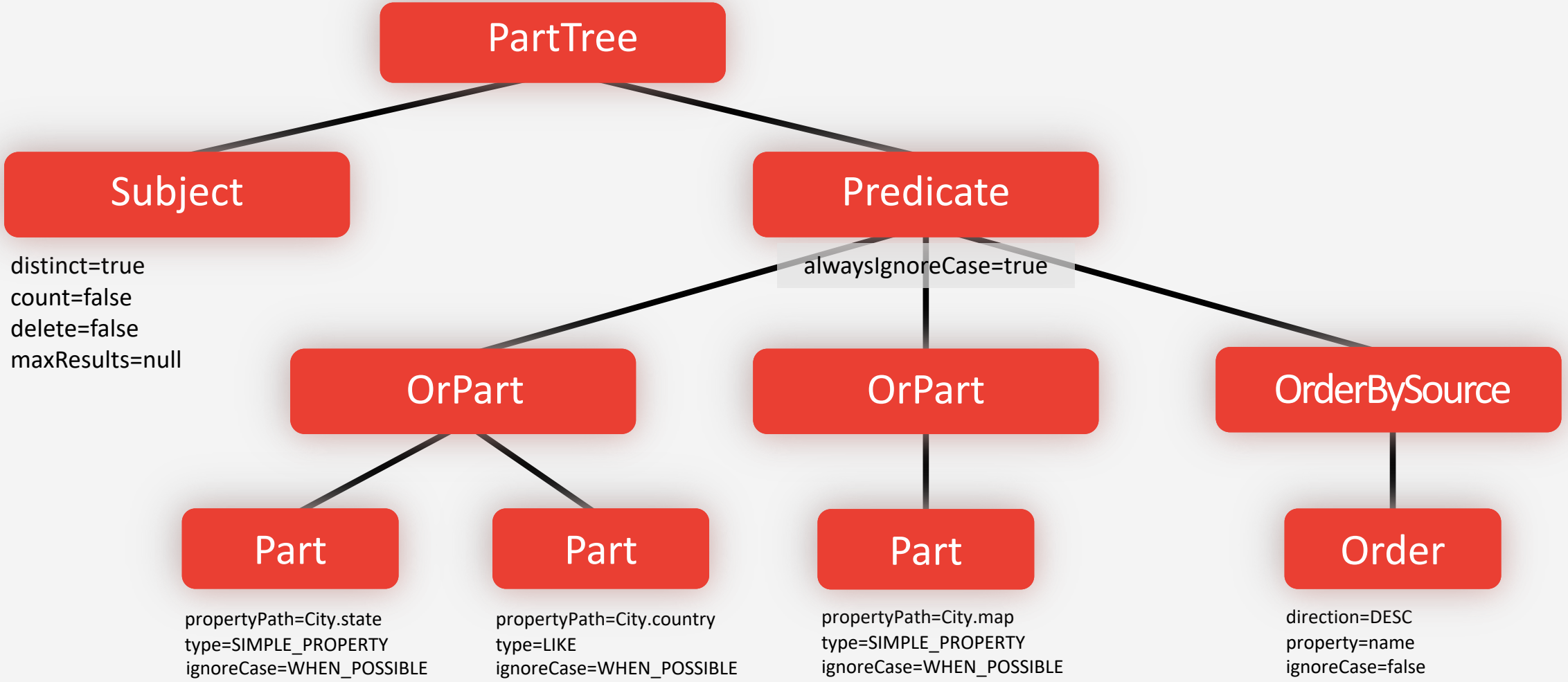
```
@Query(value = "{ 'fans' : { '$elemMatch' : { '$ref' : 'user' } } }", fields = "{ 'fans.$': ?0 }")
Person findWithArrayPositionInProjectionWithDbRef(int position);

@Aggregation("{ '$project': { '_id' : '$lastname' } }")
List<String> findAllLastnames();
```

MongoQueryLookupStrategy:

```
@Override
public RepositoryQuery resolveQuery(Method method, RepositoryMetadata metadata, ProjectionFactory factory,NamedQueries namedQueries) {

        MongoQueryMethod queryMethod = new MongoQueryMethod(method, metadata, factory, mappingContext);
        String namedQueryName = queryMethod.getNamedQueryName();

        if (namedQueries.hasQuery(namedQueryName)) {
                String namedQuery = namedQueries.getQuery(namedQueryName);
                return new StringBasedMongoQuery(namedQuery, queryMethod, operations, EXPRESSION_PARSER, evaluationContextProvider);
        } else if (queryMethod.hasAnnotatedAggregation()) {
                return new StringBasedAggregation(queryMethod, operations, EXPRESSION_PARSER, evaluationContextProvider);
        } else if (queryMethod.hasAnnotatedQuery()) {
                return new StringBasedMongoQuery(queryMethod, operations, EXPRESSION_PARSER, evaluationContextProvider);
        } else {
                return new PartTreeMongoQuery(queryMethod, operations, EXPRESSION_PARSER, evaluationContextProvider);
        }
}
```

Repositories: method into PartTree

findDistinctByStateAndCountryLikeOrMapAllIgnoringCaseOrderByNameDesc

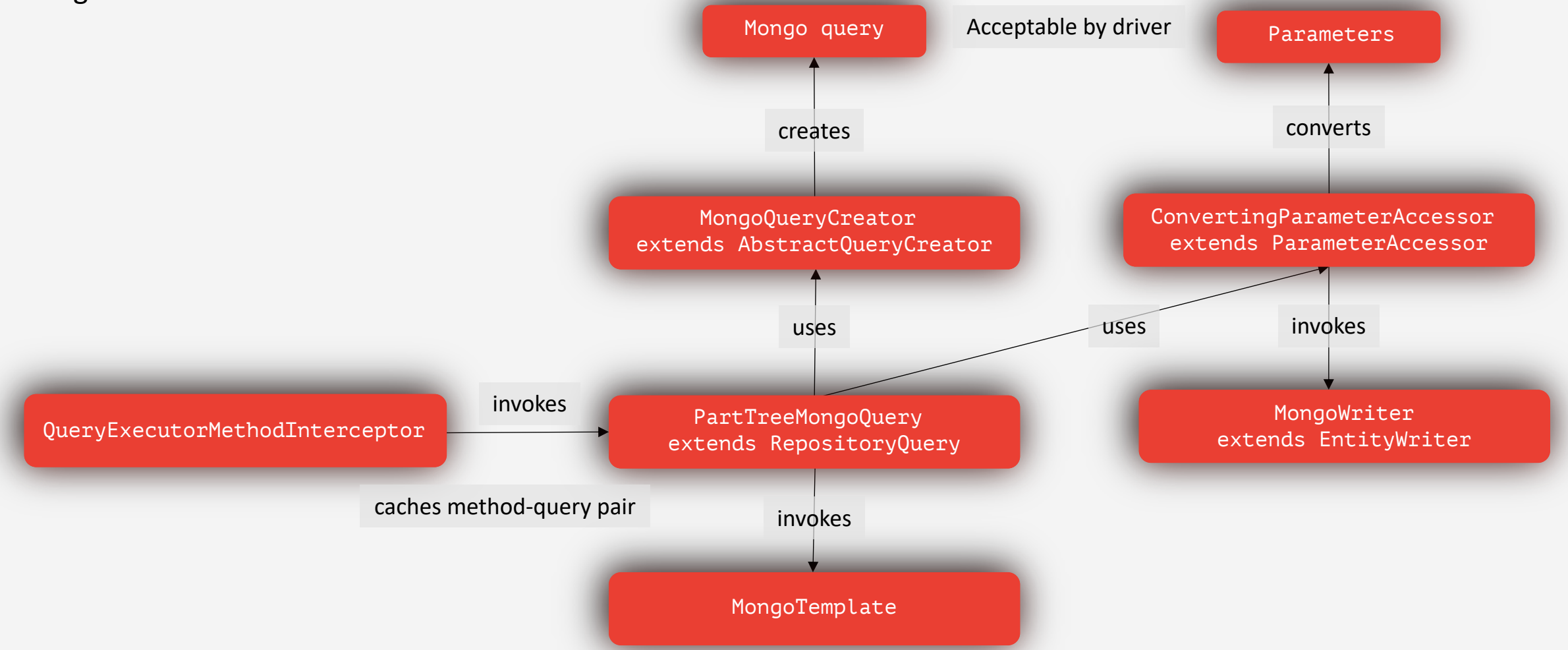


Repositories: example


```
@Configuration
@EnableTarantoolRepositories
class ApplicationConfig {
}


public interface CustomerRepository extends TarantoolRepository<Customer, Long> {
    Customer findByName(String name);

    @Query(function="find_by_email_address")
    Customer findByEmailAddress(EmailAddress email);
}


@Component
public class MyRepositoryClient {
    @Autowired
    private final CustomerRepository repository;
    ...
    public void someBusinessMethod(EmailAddress email) {
        Customer customer = repository.findByEmailAddress(email);
    }
}
```

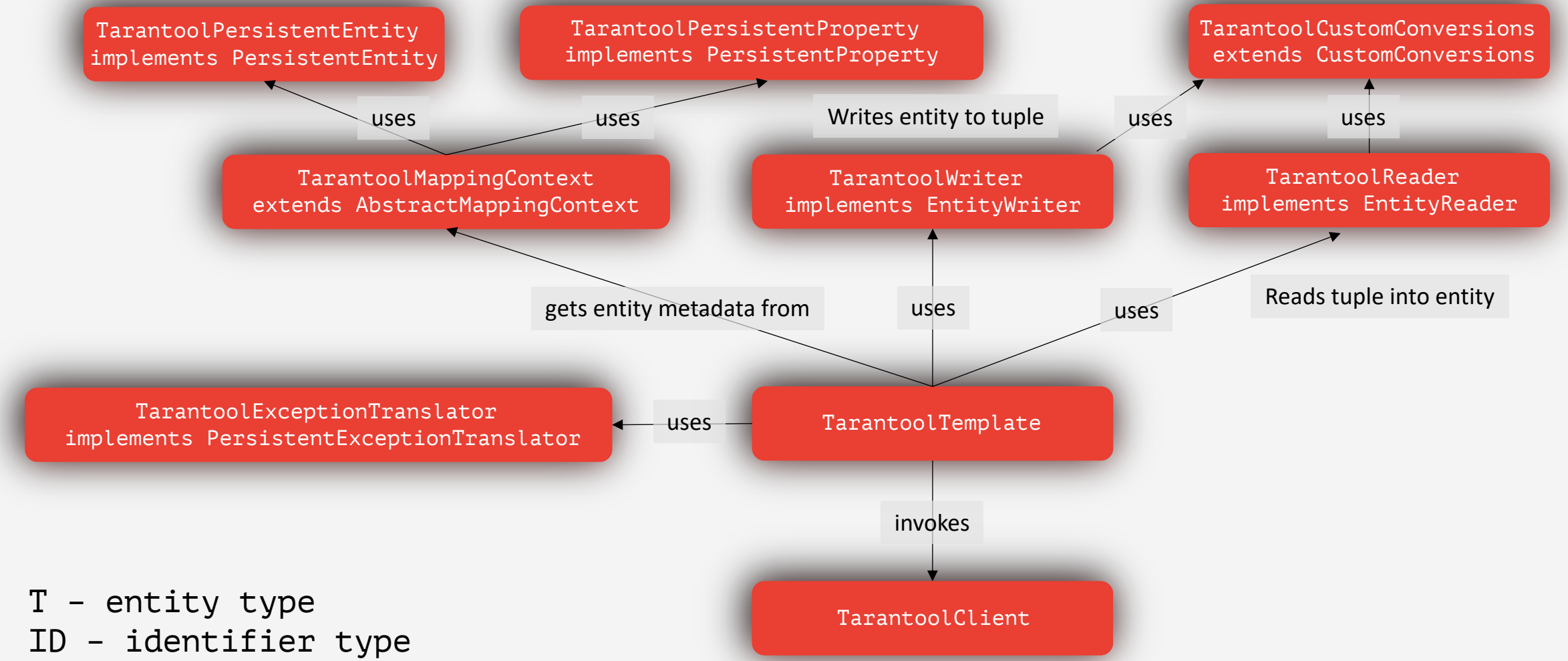
Repositories: constructing and invoking a query from PartTree (simplified)

Mongo:



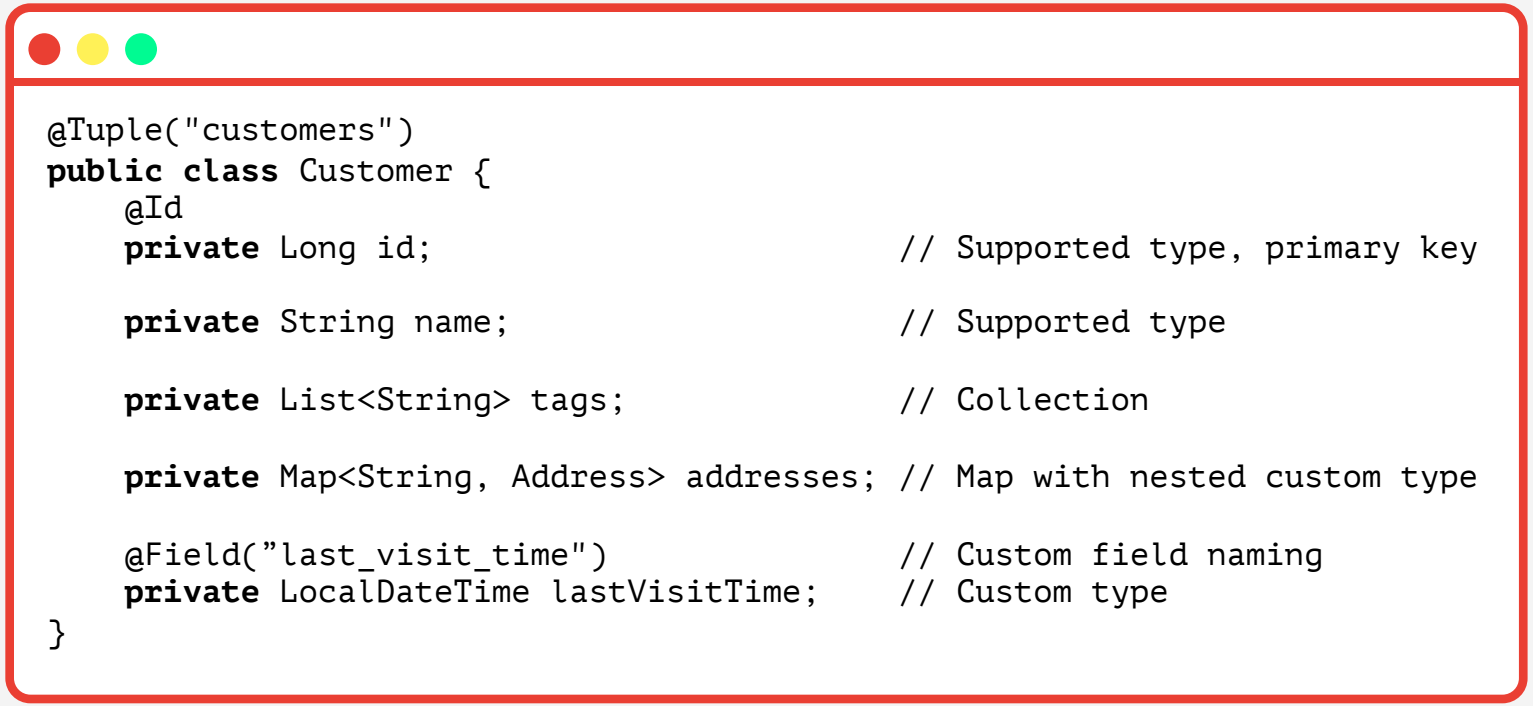
Repositories: how driver API is invoked (simplified)

Tarantool:



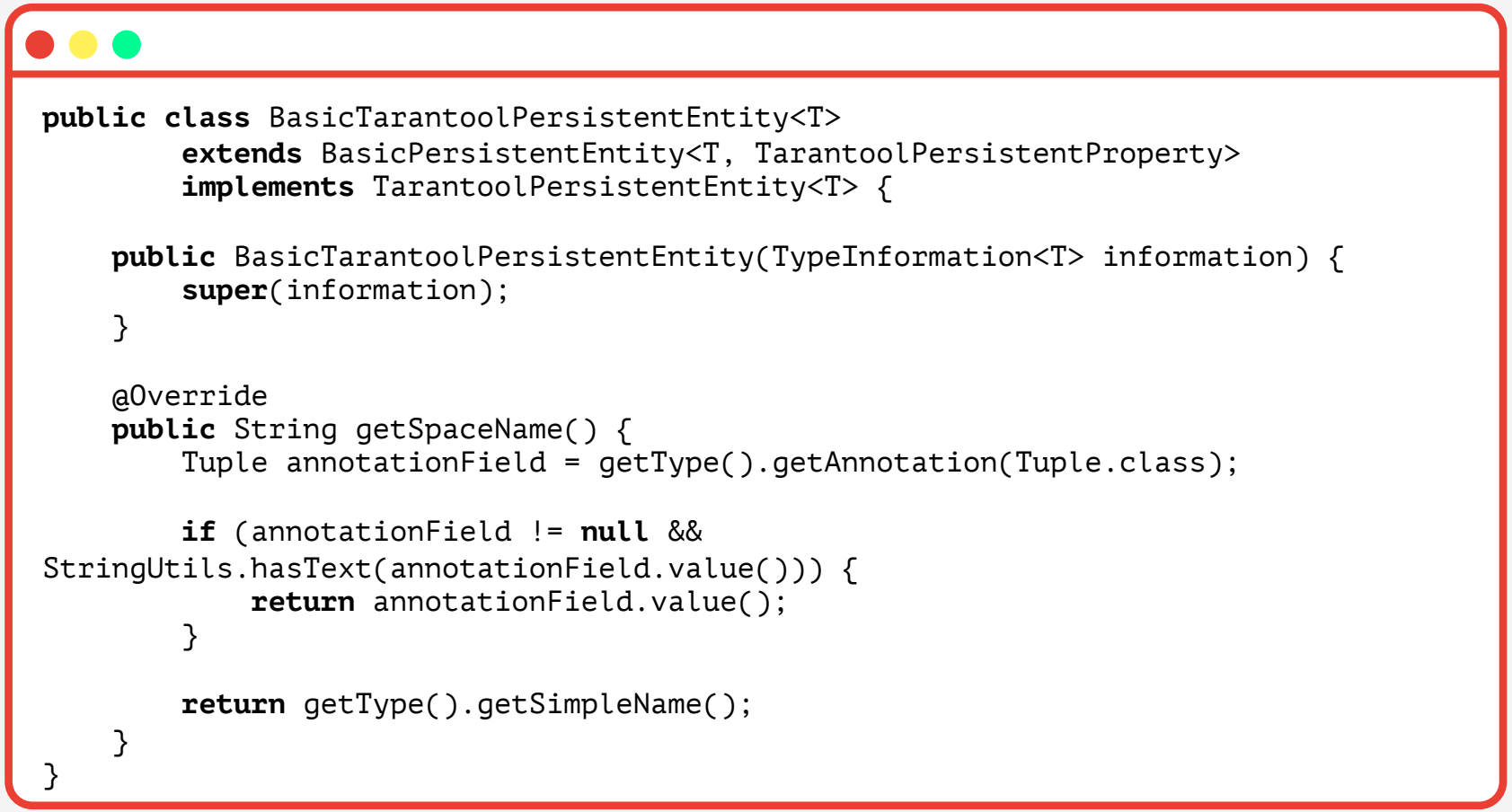
T - entity type
ID - identifier type

Entities: example



```
@Tuple("customers")
public class Customer {
    @Id
    private Long id;                              // Supported type, primary key

    private String name;                          // Supported type

    private List<String> tags;                    // Collection

    private Map<String, Address> addresses; // Map with nested custom type

    @Field(”last_visit_time")                     // Custom field naming
    private LocalDateTime lastVisitTime;          // Custom type
}
```

Entities: entity metadata



```
public class BasicTarantoolPersistentEntity<T>
        extends BasicPersistentEntity<T, TarantoolPersistentProperty>
        implements TarantoolPersistentEntity<T> {

    public BasicTarantoolPersistentEntity(TypeInformation<T> information) {
        super(information);
    }

    @Override
    public String getSpaceName() {
        Tuple annotationField = getType().getAnnotation(Tuple.class);

        if (annotationField != null &&
StringUtils.hasText(annotationField.value())) {
            return annotationField.value();
        }

        return getType().getSimpleName();
    }
}
```

Entities: property metadata


```
public class BasicTarantoolPersistentProperty extends AnnotationBasedPersistentProperty<TarantoolPersistentProperty>
        implements TarantoolPersistentProperty {

    private final FieldNamingStrategy fieldNamingStrategy;
    public BasicTarantoolPersistentProperty(Property property, TarantoolPersistentEntity<?> owner,
                                            SimpleTypeHolder simpleTypeHolder, FieldNamingStrategy fieldNamingStrategy) {
        super(property, owner, simpleTypeHolder);
        this.fieldNamingStrategy = fieldNamingStrategy == null ?
                PropertyNameFieldNamingStrategy.INSTANCE : fieldNamingStrategy;
    }

    @Override
    public String getFieldName() {
        Field annotationField = AnnotatedElementUtils.findMergedAnnotation(getField(), Field.class);

        if (annotationField != null && StringUtils.hasText(annotationField.value())) {
            return annotationField.value();
        }

        return fieldNamingStrategy.getFieldName(this);
    }
}
```

Entities: custom conversions


```
public class TarantoolCustomConversions extends CustomConversions {

    private static final List<Object> STORE_CONVERTERS;
    private static final StoreConversions STORE_CONVERSIONS;

    static {
        List<Object> converters = new ArrayList<>();

        converters.addAll(DateConverters.getConvertersToRegister());
        converters.addAll(TarantoolJsr310Converters.getConvertersToRegister());

        STORE_CONVERTERS = Collections.unmodifiableList(converters);
        STORE_CONVERSIONS = StoreConversions.of(TarantoolSimpleTypes.HOLDER, STORE_CONVERTERS);
    }

    public TarantoolCustomConversions(Collection<?> converters) {
        super(STORE_CONVERSIONS, converters);
    }
}
```

Entities: custom converters


```
@ReadingConverter
public enum NumberToLocalDateTimeConverter implements Converter<Number, LocalDateTime> {

    INSTANCE;

    @Override
    public LocalDateTime convert(Number source) {
        return source == null ? null
                : ofInstant(DateConverters.SerializedObjectToDateConverter.INSTANCE.convert(source).toInstant(),
                systemDefault());
    }
}

@WritingConverter
public enum LocalDateTimeToLongConverter implements Converter<LocalDateTime, Long> {

    INSTANCE;

    @Override
    public Long convert(LocalDateTime source) {
        return source == null ?
            null :
            DateConverters.DateToLongConverter.INSTANCE.convert(
                Date.from(source.atZone(systemDefault()).toInstant()));
    }
}
```

Sources and useful links

- More about Tarantool:
  https://www.tarantool.io/
- Introduction to the Spring Data:
  https://www.infoq.com/articles/spring-data-intro/
- Spring Data core:
  https://github.com/spring-projects/spring-data-commons/
- Short explanation how Spring Data works and how PartTree is built:
  https://lxd.me/spring-data-jpa-internals-en
- Spring Data core reference documentation:
  https://docs.spring.io/spring-data/commons/docs/current/reference/html/#reference

Alexey Kuzin, Mail.ru

https://tarantool.io/

https://github.com/akudiyar

https://t.me/akudiyar