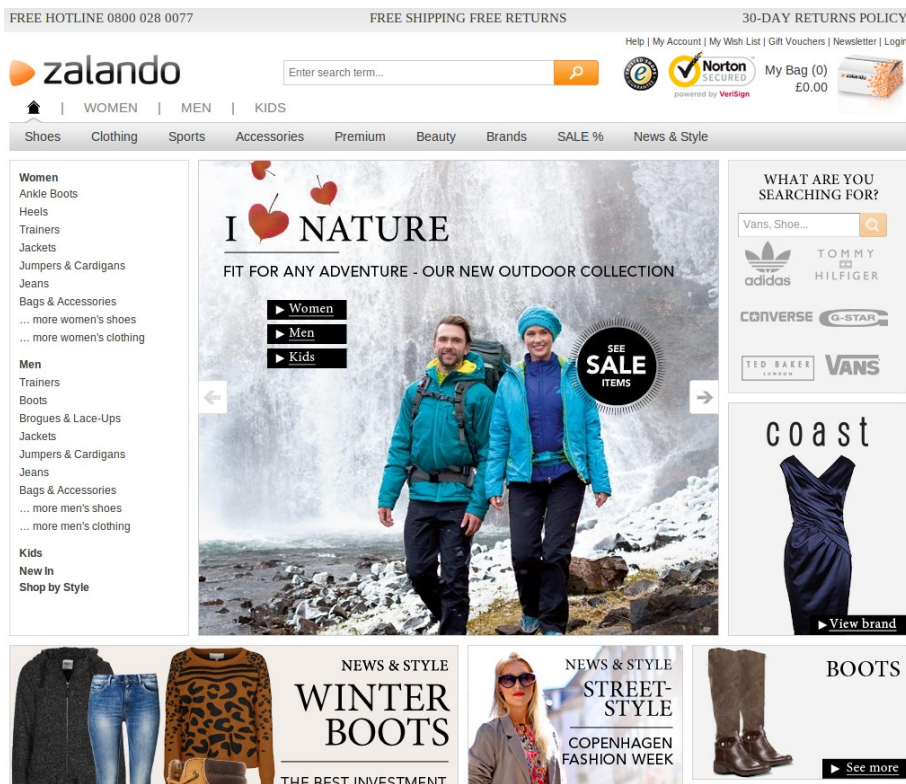


Weaving Stored Procedures into Java at Zalando

Outline

- **Introduction**
- **Stored procedure wrapper**
 - Problems before the wrapper
 - How it works
 - How to use it
 - More features including sharding
- **PGObserver**

Zalando



- 14 countries
- 1 Billion € net revenues for 2012
- 3(+1) warehouses
- Europe's largest online fashion retailer
- 150k+ articles

tech.zalando.com

Zalando platform

- Modern open source software stack
- Mostly Java
- PostgreSQL database backend
- 150 developers and counting, technology surpasses 300 in total



`tech.zalando.com`

PostgreSQL setup

- ~ 20+ Servers PostgreSQL master servers
- ~ 5.000 GB of data
- Started with PostgreSQL 9.0 rc1
- Now running version 9.0 to 9.2
 - cascading replication very welcome
 - maintenance improvements great (drop concurrently)
 - Index only scan, pg_stat_statement improvements
- Machine setup
 - 8- to 48- cores, 16GB to 128GB
 - SAN, no SAN with (2x2x RAID 1, 4x RAID 10) preferred

PostgreSQL availability

- BoneCP as Java connection pool
- All databases use streaming replication
 - Service IP for switching
- Failover is manual task
 - Monitored by Java app, Web frontend
- Significant replication delays sometimes
 - Fullpage writes, Nested Transactions, Slave load

Stored procedure experience

- Performance benefits
- Easy to change live behavior
- Makes moving to new software version easy
- Validation close to data
- Run a very simplistic transaction scope
- Cross language API layer
- More than 1000 stored procedures
 - More plpgsql than SQL than plpython

Outline

- Introduction
- Stored procedure wrapper
 - **Problems before the wrapper**
 - How it works
 - How to use it
 - More features including sharding
- PGObserver

Execution of stored procedures

- Using spring's BaseStoredProcudere
 - Initially a lot of work per stored procedure
 - One class per stored procedure
 - Write row mappers for domain object mapping
- Missing type mapper on Java side
 - Spring type mapper insufficient
 - Enums, array of types, nesting, and hstore missing
- JdbcTemplate and alternatives lack ease of use

“Ancient” history

```
@SPROC(name = "update_customer")
@SPROCParameters(
    value = {
        @SPROCParameter(name = "p_number", type = SqlType.VARCHAR),
        @SPROCParameter(name = "p_phone", type = SqlType.VARCHAR),
        @SPROCParameter(name = "p_birthday", type = SqlType.DATE),
        @SPROCParameter(name = "p_email", type = SqlType.VARCHAR),
        @SPROCParameter(name = "p_first_name", type = SqlType.VARCHAR),
        @SPROCParameter(name = "p_last_name", type = SqlType.VARCHAR),
        @SPROCParameter(name = "p_gender", type = SqlType.CHAR),
        @SPROCParameter(name = "p_title", type = SqlType.VARCHAR)
    }
)

public class UpdateCustomerProc extends BaseStoredProcedure {

    public UpdateCustomerProc(final DataSource dataSource) {
        setDataSource(dataSource);
        setResultMapper(new ModificationResultMapper());
        declareAndCompile();
    }

    public StatusMessage updateCustomer(final Customer customer) {

        return (StatusMessage) executeWithSingleResult(customer.getCustomerNumber(),
            [...],
            customer.getSex().getDbValue(), title);
    }
}
```

Goals of our wrapper

- Write as little code as possible on Java side
- One location for procedures of same topic
- One call path to any stored procedure
- “Natural” feeling for using stored procedures
 - Procedure call should look like Java method call
 - RPC like

Brief example

```
CREATE OR REPLACE FUNCTION create_customer ( p_customer t_customer )  
  RETURNS SETOF t_customer  
AS  
$$  
  -- Procedure definition  
$$  
LANGUAGE 'plpgsql' SECURITY DEFINER;
```

Brief example

```
CREATE OR REPLACE FUNCTION create_customer ( p_customer t_customer )  
  RETURNS SETOF t_customer  
AS  
$$  
  -- Procedure definition  
$$  
LANGUAGE 'plpgsql' SECURITY DEFINER;
```

```
@SProcService  
public interface CustomerExampleSProcService {  
  
    @SProcCall  
    Customer createCustomer(@SProcParam Customer customer);  
}
```

Brief example

```
@SProcService
public interface CustomerExampleSProcService {

    @SProcCall
    Customer createCustomer(@SProcParam Customer customer);
}
```

```
Customer c = new Customer();
c.setFirstName("Jan");
c.setName("Name");
```

```
Customer result = service.createCustomer(c);
```

Brief example

```
@SProcService
public interface CustomerExampleSProcService {

    @SProcCall
    Customer createCustomer(@SProcParam Customer customer);
}
```

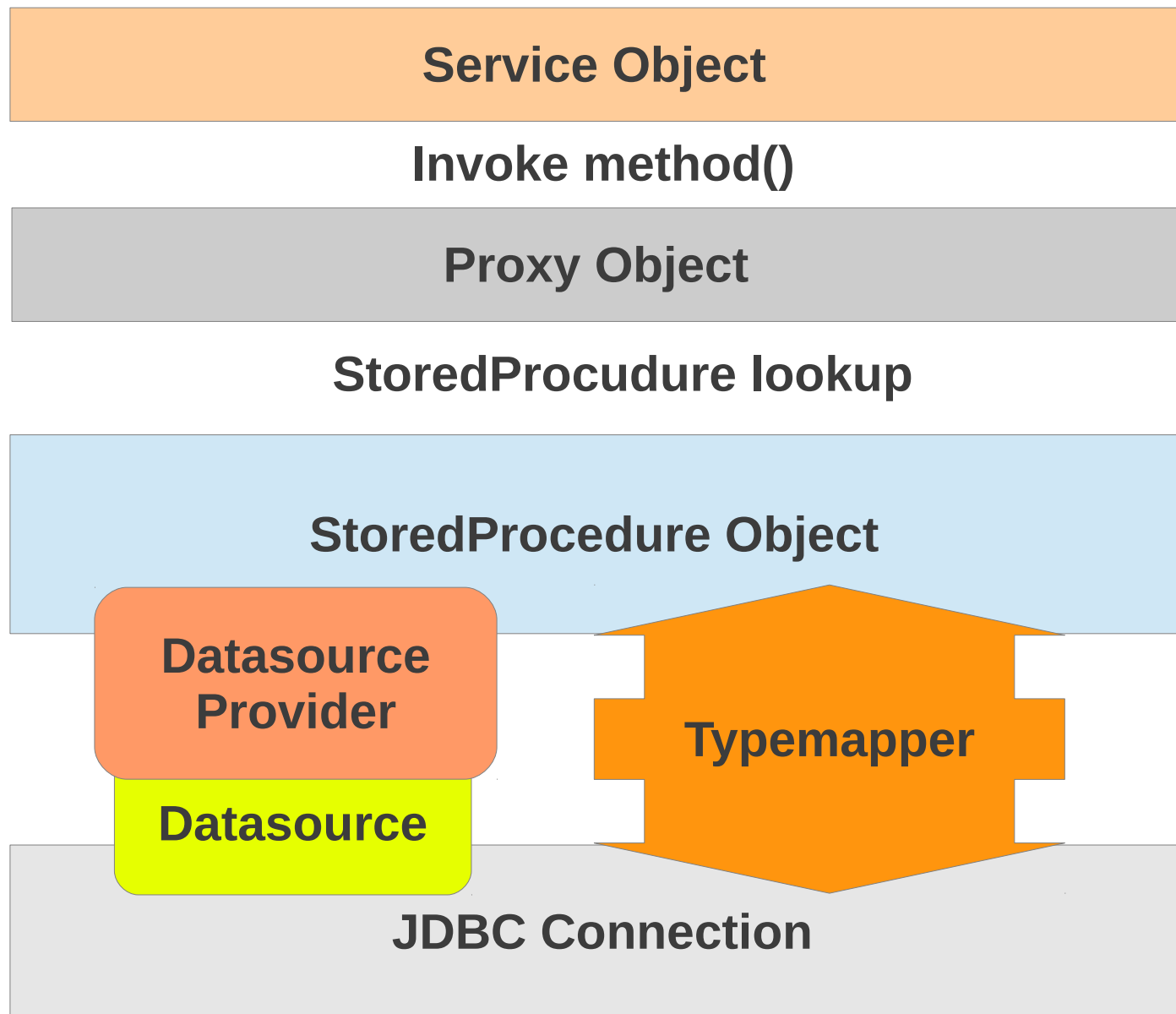
```
Customer c = new Customer();
c.setFirstName("Jan");
c.setName("Name");
```

```
Customer result = service.createCustomer(c);
```

Outline

- Introduction
- Stored procedure wrapper
 - Problems before the wrapper
 - **How it works**
 - How to use it
 - More features including sharding
- PGObserver

Under the hood



Features

- New spring compatible type mapper
 - From simple types to nested domain objects
 - Supports PG enum to Java enum
- Accessing sharded data supported
 - Result “aggregation” across shards
 - Parallel query issuing
- Advisory locking via annotation
- Set custom timeout per stored procedure
- Cross shard commit levels, including 2P commit

Type mapper

- Annotations for class and member variables
 - **@DatabaseType** and **@DatabaseField**
- CamelCase to camel_case conversion
- JPA 2.0 **@Column** annotation supported
- Addition type conversions include:
 - Nested PostgreSQL types to Java objects
 - hstore to Map<String,String>
 - PostgreSQL enum to Java enum (by name)
 - PostgreSQL array[] to List<?>()

Outline

- Introduction
- Stored procedure wrapper
 - Problems before the wrapper
 - How it works
 - **How to use it**
 - More features including sharding
- PGObserver

Using the wrapper

- Consider Java to PostgreSQL plpgsql
- First define the Java interface

```
@SProcService
public interface CustomerExampleSProcService {

    @SProcCall
    public Customer loadCustomer(@SProcParam int id);

    @SProcCall
    public Customer createCustomer(@SProcParam Customer customer);

    @SProcCall
    public int addNewAddress(@SProcParam int customerId,
                           @SProcParam Address address);
}
```

Using the wrapper

- Create class implementing previous interface

```
@Repository
public class CustomerExampleSProcServiceImpl
    extends AbstractSProcService<CustomerExampleSProcService,
                                SingleDataSourceProvider>
    implements CustomerExampleSProcService {

    @Autowired
    public CustomerExampleSProcServiceImpl(
        @Qualifier("testDataCustomerExampleProvider")
        final SingleDataSourceProvider p) {

        super(p, CustomerExampleSProcService.class);
    }

    @Override
    public Customer loadCustomer(int id) {
        return sproc.loadCustomer(id);
    }
}
```

Using the wrapper

- Define DTO classes if necessary
 - Input parameters
 - ResultSet mapping

```
@DatabaseType(name="t_customer")
public class Customer {

    @DatabaseField
    protected Integer id;

    @DatabaseField
    protected String name;
    @DatabaseField
    protected String firstName;
    @DatabaseField
    protected List<Address> addresses;

    @DatabaseField
    protected Address defaultAddress;
```

Using the wrapper

- Next create analogous PostgreSQL types

```
CREATE TYPE t_customer AS ( id int,  
                                name text,  
                                address t_address[] );
```

- Or use “OUT” columns

```
CREATE FUNCTION load_customer( INOUT id int,  
                                OUT name text,  
                                OUT address t_address[] )  
RETURNS SETOF record AS
```

- Implement stored procedures

Outline

- Introduction
- Stored procedure wrapper
 - Problems before the wrapper
 - How it works
 - How to use it
 - **More features including sharding**
- PGObserver

Running SQL queries

- `@SProcCall(sql="[...]")` may run any query
 - Benefit from type mapper
 - Relatively easy to use
 - Although mixing SQL into Java source

```
@SProcCall (sql="UPDATE t SET name = ?"  
            + " WHERE id = ? "  
            + " RETURNING id")  
int updateName (@SProcParam String newName,  
                @SProcParam int userId);  
  
// allows you then to do:  
int r = service.updateName ('Jan', 1001);
```

Sharding support

- Parameter annotation **@ShardKey**
- **@ShardKey** and **@SProcParam** may overlap

@SProcCall

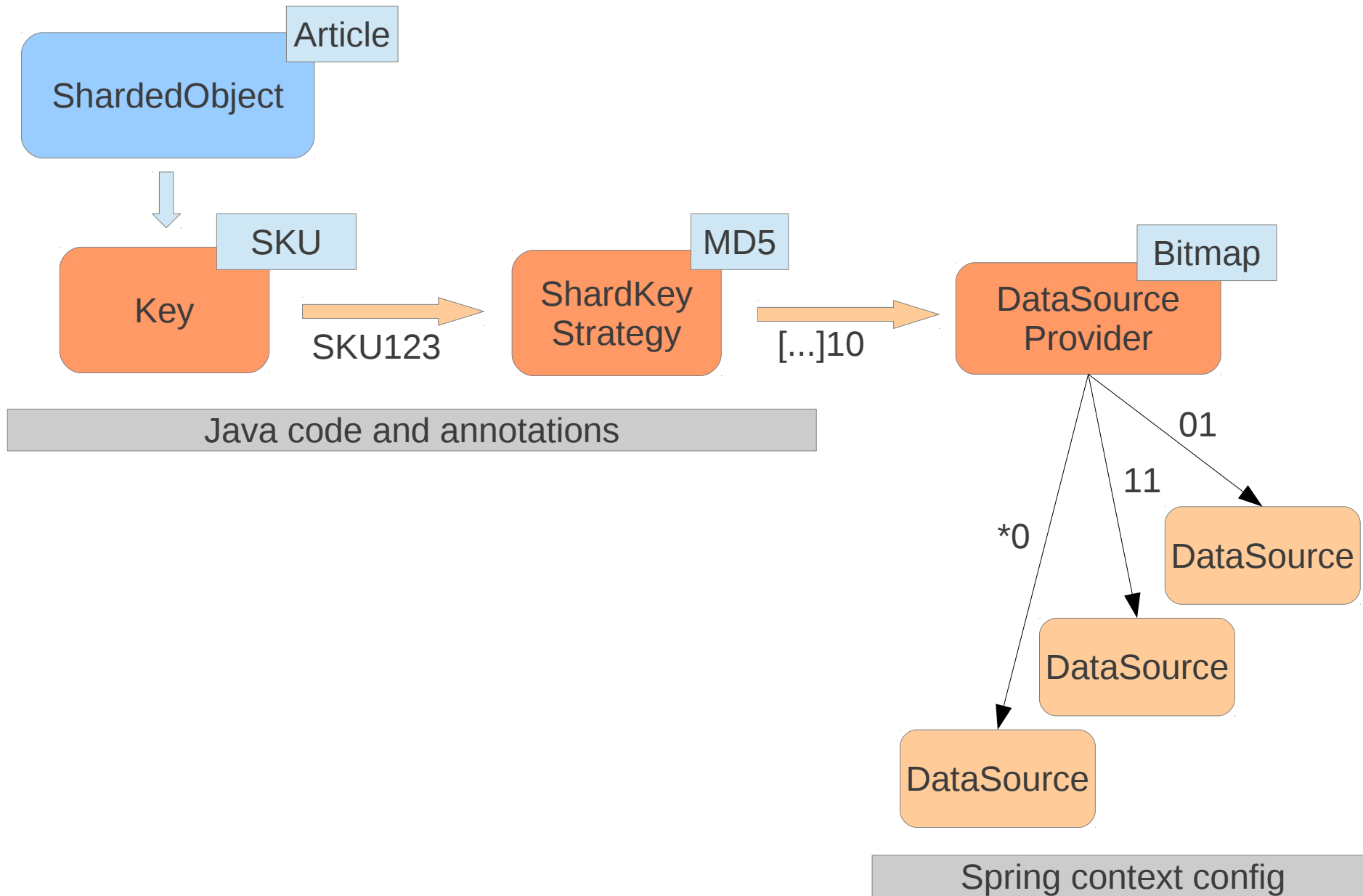
```
Customer getCustomer(@ShardKey int shardId,  
                     @SProcParam String cnumber)
```

@SProcCall

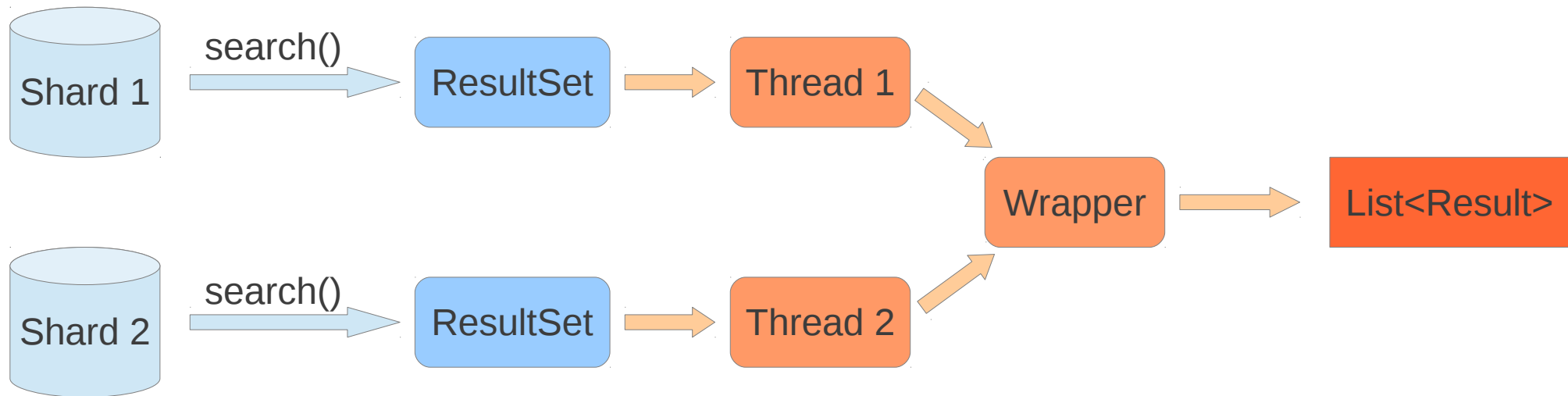
```
Article getArticle(@ShardKey @SProcParam String ean)
```

- *ShardedObject* interface for custom classes
- Added datasource providers for translation

Different datasource providers

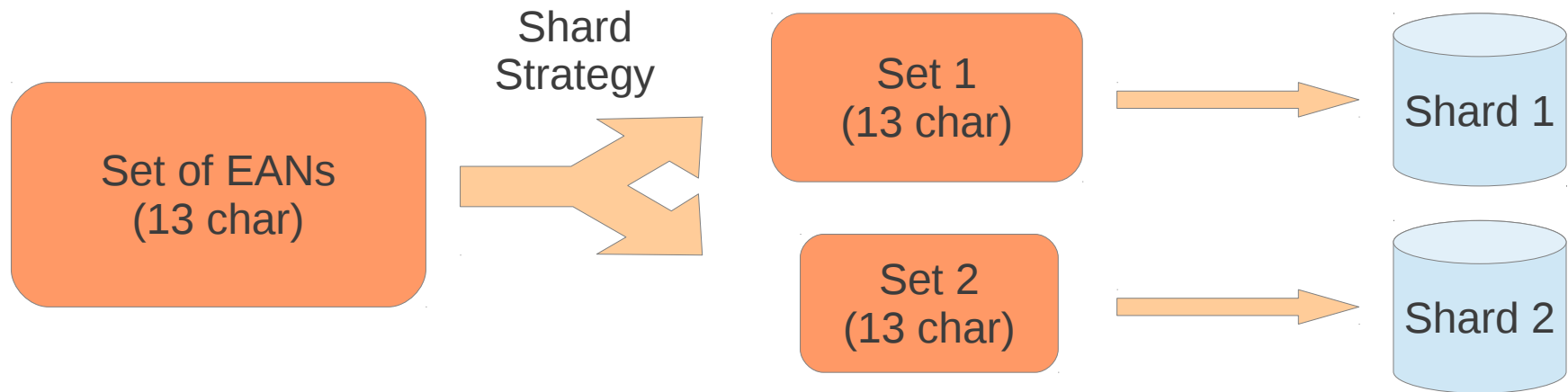


Search and “Merge” result set



- Use **searchShards** where you do not know the shard
 - will run on all shards return on first find
- Use **runOnAllShards** execute on all shards
 - Search *name like 'Na%'* and return one collection

Auto partitioning



- Java method called with one large collection
 - Wrapper will split collection according to key
 - Execute SQL for split collection on each shard
- Default behavior if **@ShardKey** is a collection

Java bean validation

- Annotation based validation (JSR 303)

```
@DatabaseField  
@NotNull  
public String a;
```

```
@DatabaseField  
@Min(4)  
@Max(6)  
@NotNull  
public Integer b;
```

- Relying on hibernate validator
- Automatically checked inside wrapper
 - Less boiler plate code
 - **@SProcService(validate = true)**

Value transformers

- Global registry for type conversions
 - e.g. for use with JodaTime class
 - Enables transparent handling of legacy types
- Usefull for *::text* to Java class conversion
 - Type safe domain classes
 - *::text* => class EAN

Per stored procedure timeout

- Trouble with global statement timeout
 - Long running queries and supposedly fast ones
- Added **@SProcCall(timeout=x)**
 - X is timeout in ms
 - Allows overwrite for long running jobs
 - Ensures limited run time for “fast” functions
 - Search functions with too few constraints

Concurrency with advisory locks

- Single database serves many Java instances
 - Synchronization may be required
- Wrapper features one enum for different locks
 - `@SProcedureCall(advisoryLockType=LOCK1)`
 - Easy locking
 - One enum warns developers of existing locks

Transaction support

- Spring's @Transactional should work
 - More or less datasource dependent
 - Sharded environment more complicated
- For multi shard operations wrapper provides
 - Context is one procedure equals one transaction
 - Immediate commit on each shard
 - Commit only if all executions were successful
 - Use two phase commit
- Enabled on SProcService or SProcCall level

Outline

- Introduction
- Stored procedure wrapper
 - Problems before the wrapper
 - How it works
 - How to use it
 - More features including sharding
- **PGObserver**

PGObserver

- Build to monitor PostgreSQL performance
 - Stored procedures as execution unit
 - Track table statistics to assist identifying causes
- Infrastructure
 - One Java data gatherer
 - Web frontend in using Python
 - Metric data is stored in PostgreSQL
 - Per service configuration of all gather intervals

PGObserver database view



Database size all tables



Top 10 Sprocs last 1 hour by total run time

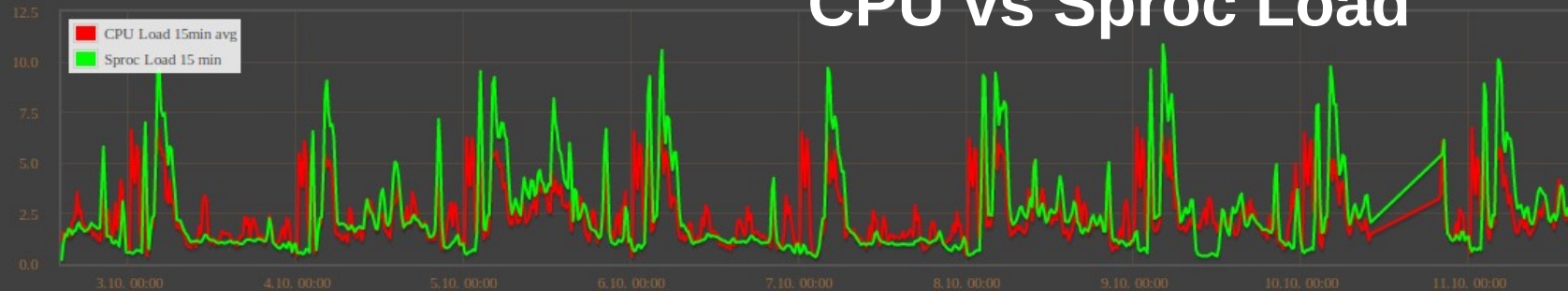
Name	Calls	Total Time	Avg. Time
get_articles_for_response_updates()	311	11m 58.561s	2.310s
get_updates_pending_and_in_processing()	9141	9m 36.958s	0.063s
calculate_simple_prices()	3552	7m 8.279s	0.121s
calculate_simple_prices()	3552	7m 7.633s	0.120s
get_creating_articles_for_blog_updates()	1258	4m 31.393s	0.216s
get_articles_for_image_updates()	285	4m 27.331s	0.938s
update_prices_from_blog_posts()	7104	4m 21.282s	0.037s
get_articles_check_ready()	20677	4m 14.160s	0.012s
get_creating_articles()	18940	3m 59.183s	0.013s
get_posts_articles_ready()	18261	2m 44.869s	0.009s

Top 10 Sprocs last 1 hour by total calls

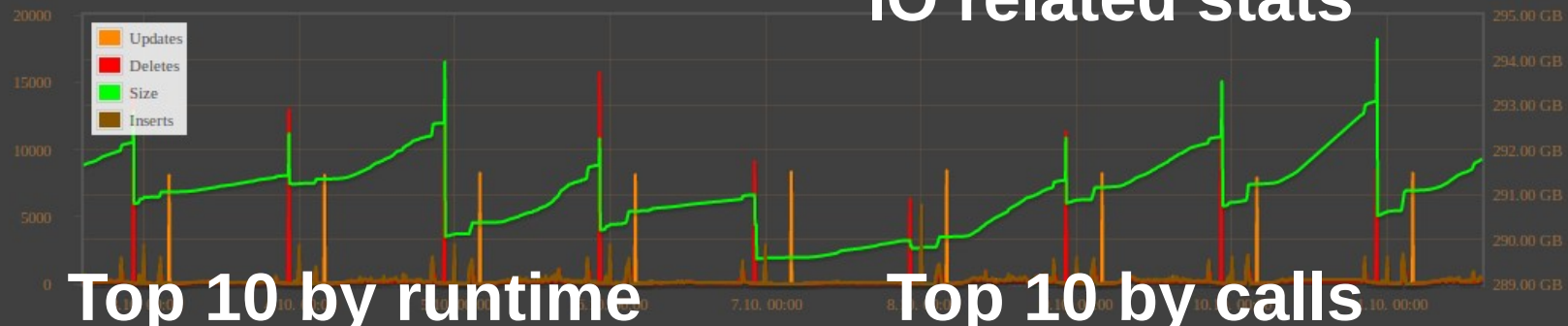
Name	Calls	Total Time	Avg. Time
insert_updates_ready()	248101	1m 19.093s	0.000s
is_price_pending()	76563	3.604s	0.000s
update_with_in_memory_articles()	56845	13.663s	0.000s
flushing_a_article_flushing()	40588	41.182s	0.001s
update_updates_ready()	28519	31.521s	0.001s
get_articles_check_ready()	20677	4m 14.160s	0.012s
get_creating_articles()	18940	3m 59.183s	0.013s
get_posts_articles_ready()	18261	2m 44.869s	0.009s
get_blog_posts_for_blog()	10882	28.645s	0.003s
get_articles_check_ready()	10558	1m 15.699s	0.007s

PGObserver database view

CPU vs Sproc Load



Database size all tables



IO related stats

Top 10 by runtime

Top 10 by calls

Top 10 Sprocs last 1 hour by total run time

Top 10 Sprocs last 1 hour by total calls

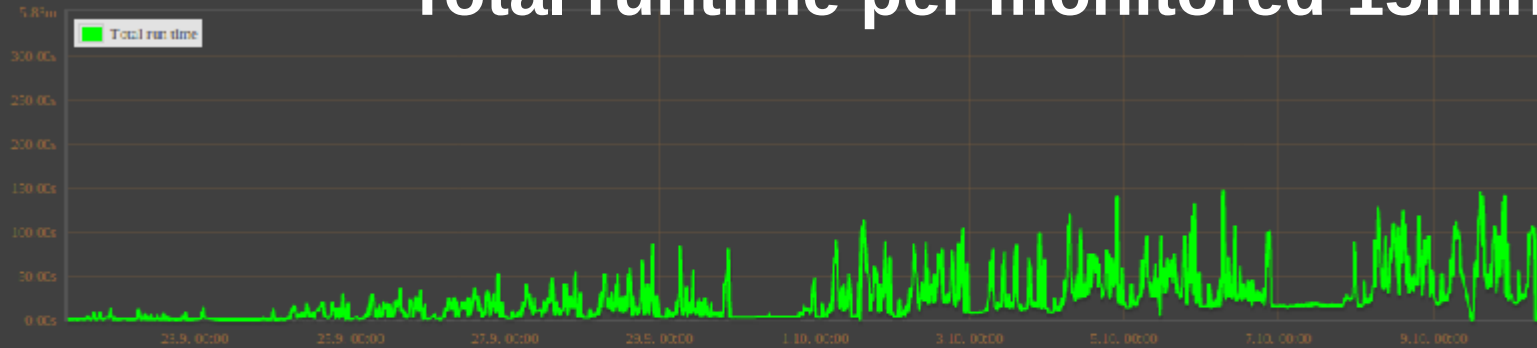
Name	Calls	Total Time	Avg. Time
get_addresses_for_response_updates	311	11m 58.561s	2.310s
get_updates_pending_and_in_processing	9141	9m 36.958s	0.063s
calculate_simple_pending	3552	7m 8.279s	0.121s
calculate_simple_pending	3552	7m 7.633s	0.120s
get_creating_addresses_for_log_updates	1258	4m 31.393s	0.216s
get_addresses_for_log_updates	285	4m 27.331s	0.938s
update_prices_from_log_updates	7104	4m 21.282s	0.037s
get_addresses_check_ready	20677	4m 14.160s	0.012s
get_creating_addresses	18940	3m 59.183s	0.013s
get_addresses_ready	18261	2m 44.869s	0.009s

Name	Calls	Total Time	Avg. Time
insert_updates_ready	248101	1m 19.093s	0.000s
is_price_pending	76563	3.604s	0.000s
update_addresses_pending	56845	13.663s	0.000s
finalize_addresses_finalizing	40588	41.182s	0.001s
update_updates_ready	28519	31.521s	0.001s
get_addresses_check_ready	20677	4m 14.160s	0.012s
get_creating_addresses	18940	3m 59.183s	0.013s
get_addresses_ready	18261	2m 44.869s	0.009s
get_addresses_for_log_updates	10882	28.645s	0.003s
get_addresses_check_ready	10558	1m 15.699s	0.007s

Sequential scan in live env.

Run time total per 15 minutes

Total runtime per monitored 15min



Total time avg.

Avg. run time per call



Self time avg.

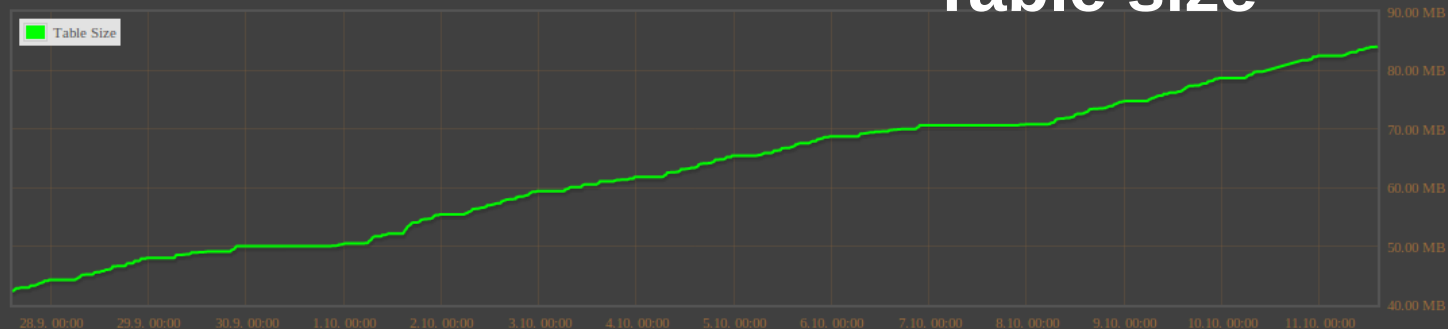
Avg. self time per call



Table I/O data excerpt

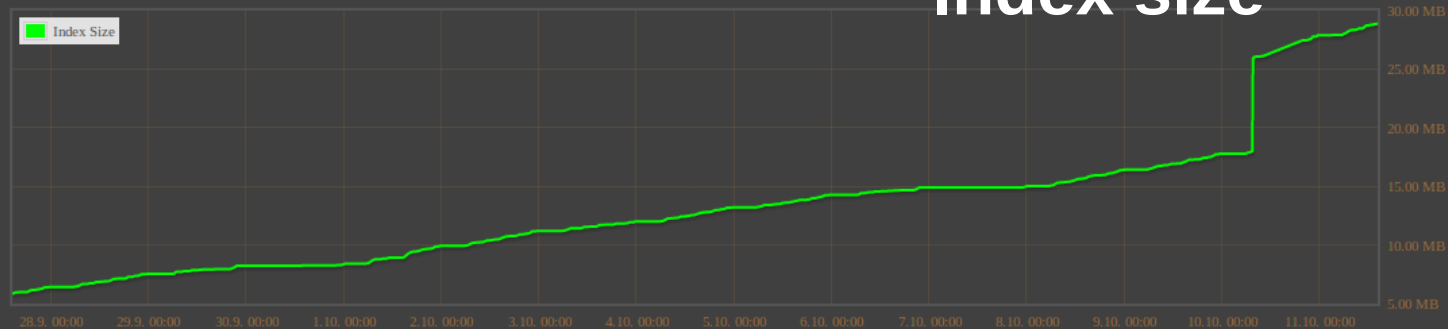
Table size

Table size



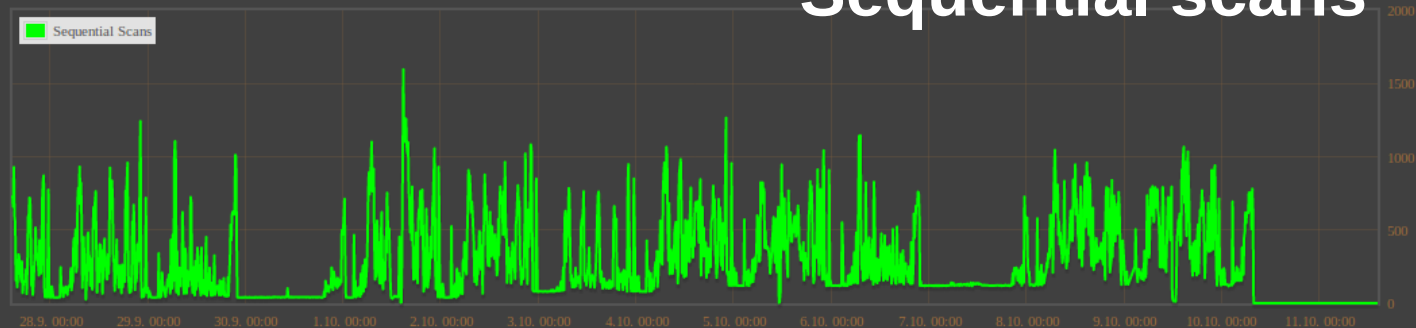
Index size

Index size



Sequential scans

Sequential scans



Summary

- Stored procedures can improve performance
- Type mapper great library to reduce code for mapping layer
- Wrapper makes procedure usage a lot easier
- Stored procedure and performance monitoring in general is very important

Visit us on:

- <http://www.github.com/zalando>
- <http://tech.zalando.com>

Thank you for listening