

Easy Bean Mappings with MapStruct 1.2

19 June 2017

Thomas Much

 @thmuch

1100 1010 1111 1110 1011 1010 1011 1110



www.jughh.de

Who's talking?

- Thomas Much
- Agile Developer Coach & XP Coder (Java et al.)
- Likes to use cool little tools & libraries (and tell other people about them)



Java & beans, some history



Form
Beans

DTOs

POJOs

JPA
Entities

Type safe,
fast
bean mappings!

1995

1998

1999

2000

2006

1997

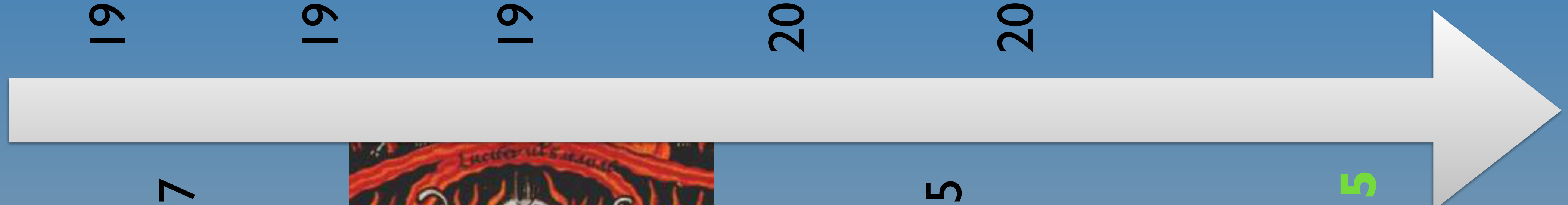
JavaBeans



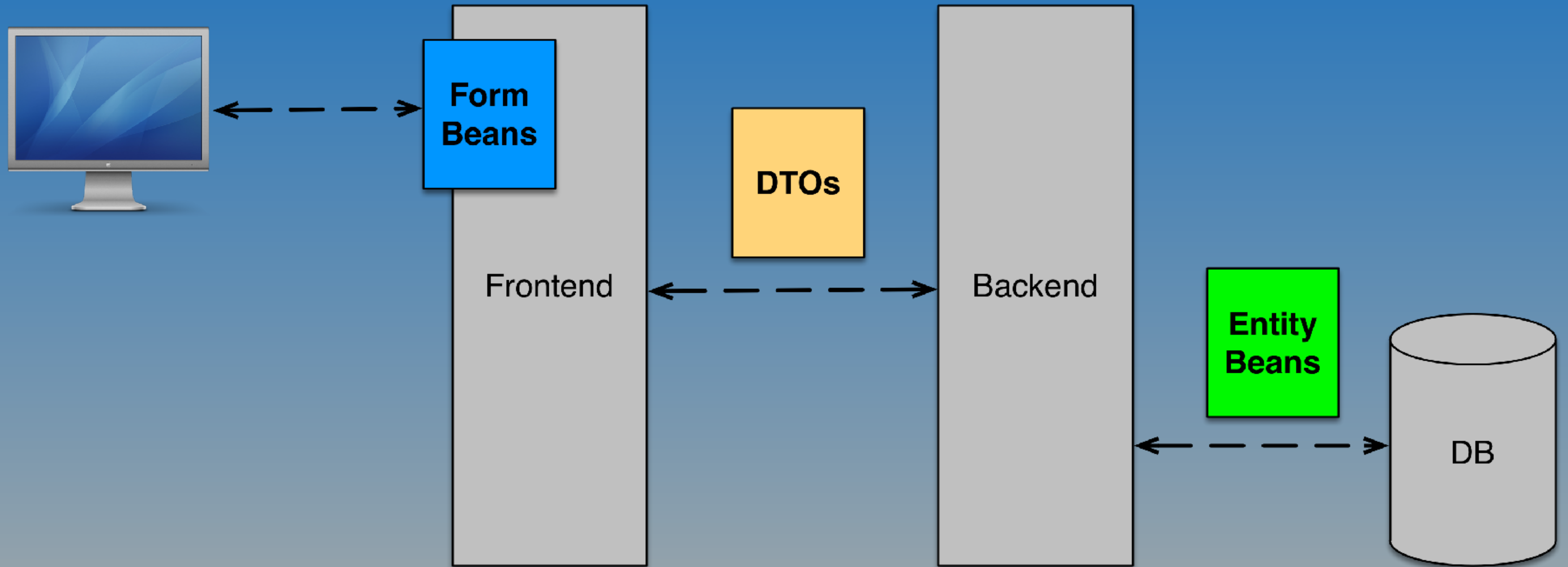
2005

JAXB
Entities

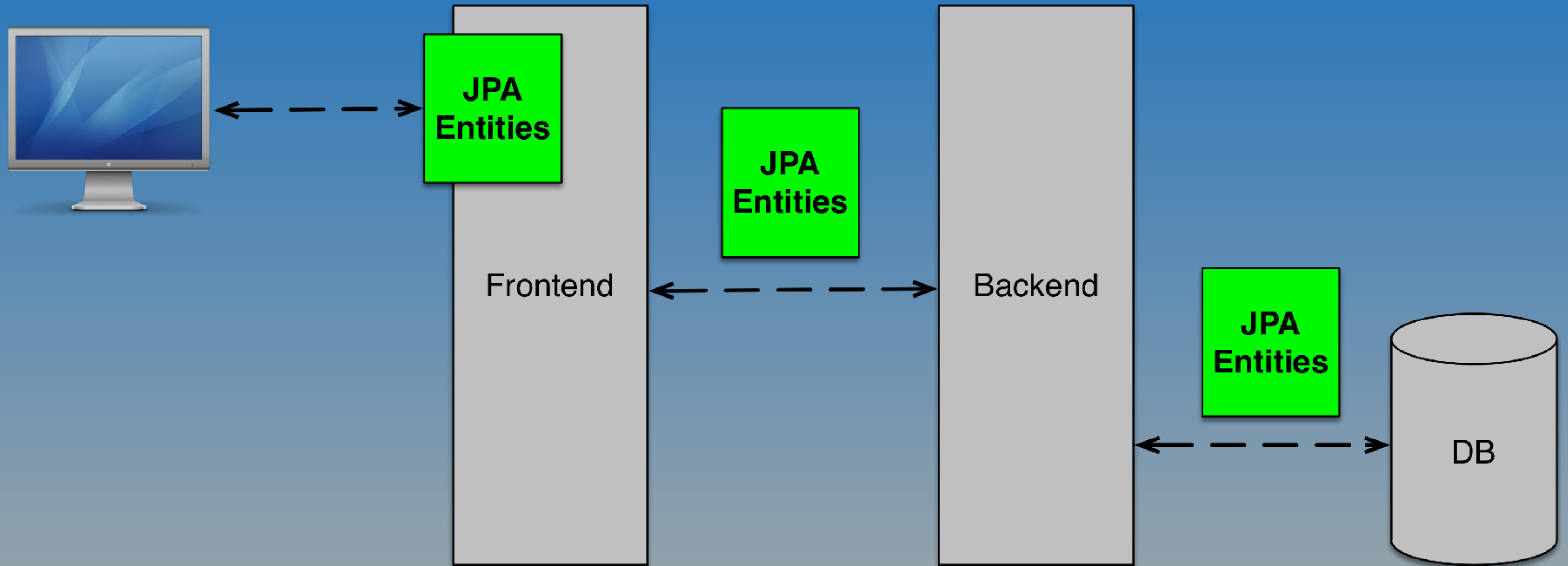
2015



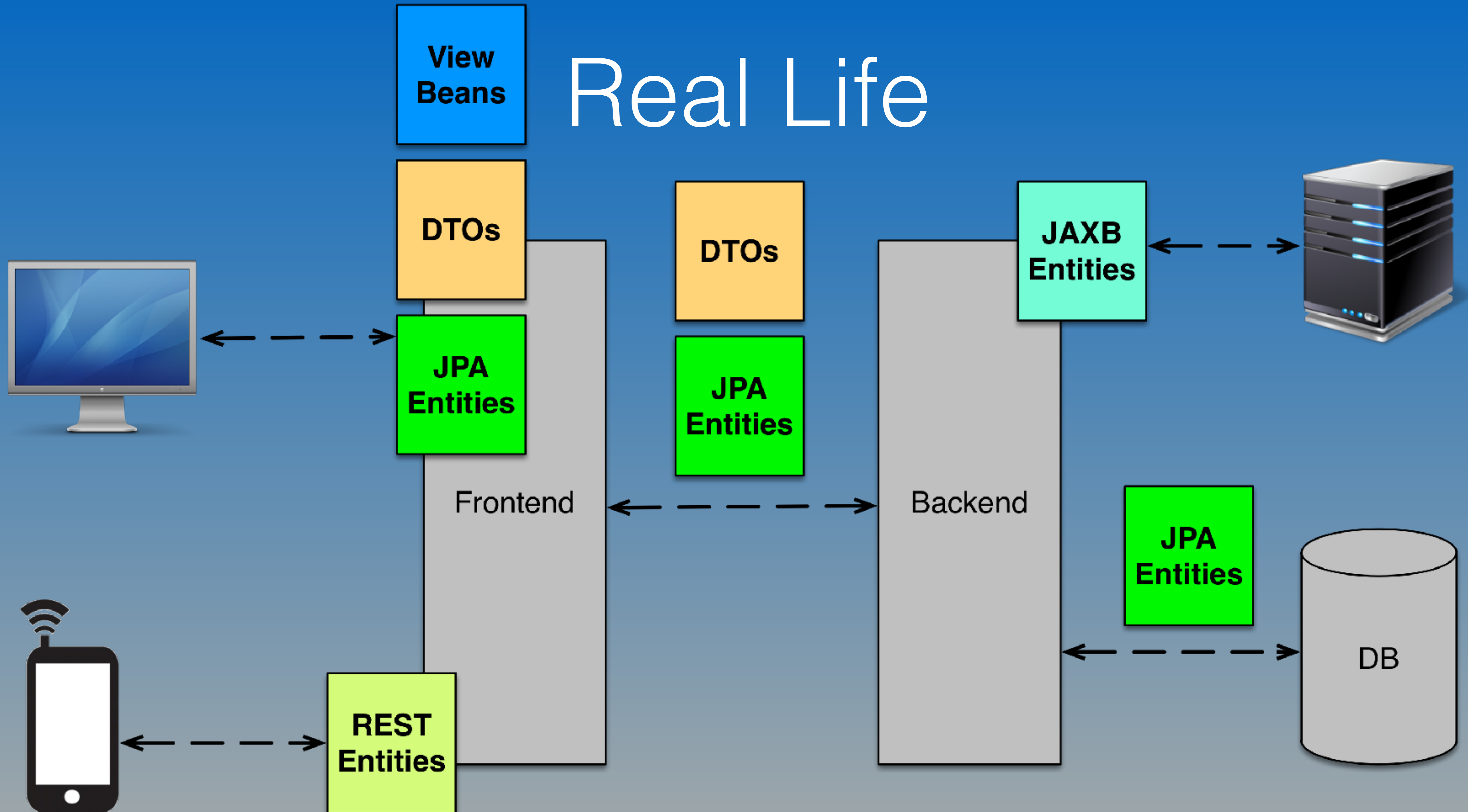
System architecture, old style



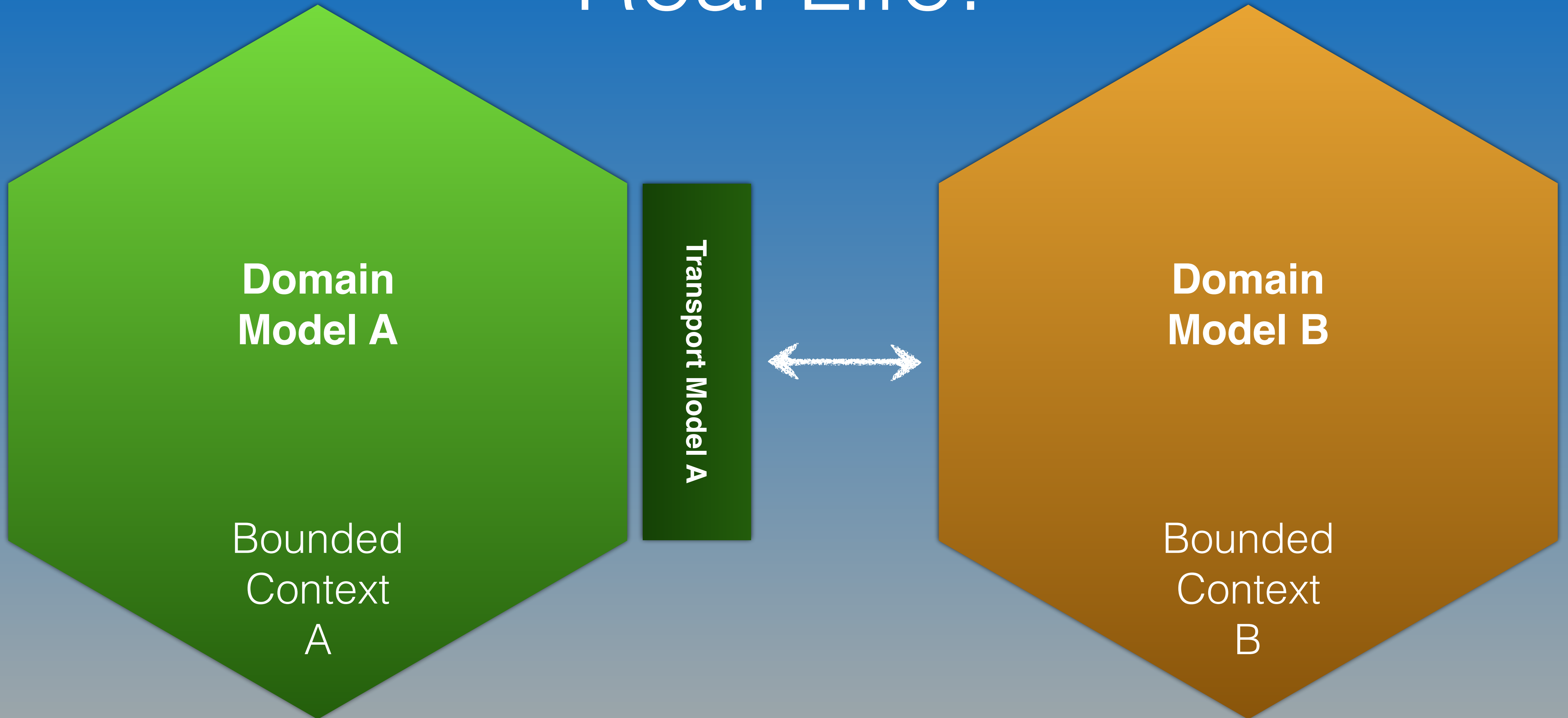
The promise of Java EE 5+ & Co.



Real Life



Real Life!



The necessary mappings...

```
@Entity
@Table(name = "CUSTOMERS")
public class Customer {

    @Id
    Long id;

    long customerId;

    String name;

    @Enumerated
    Title title;

    ... getters and setters! ...
}
```



```
public class CustomerDTO {

    long id;

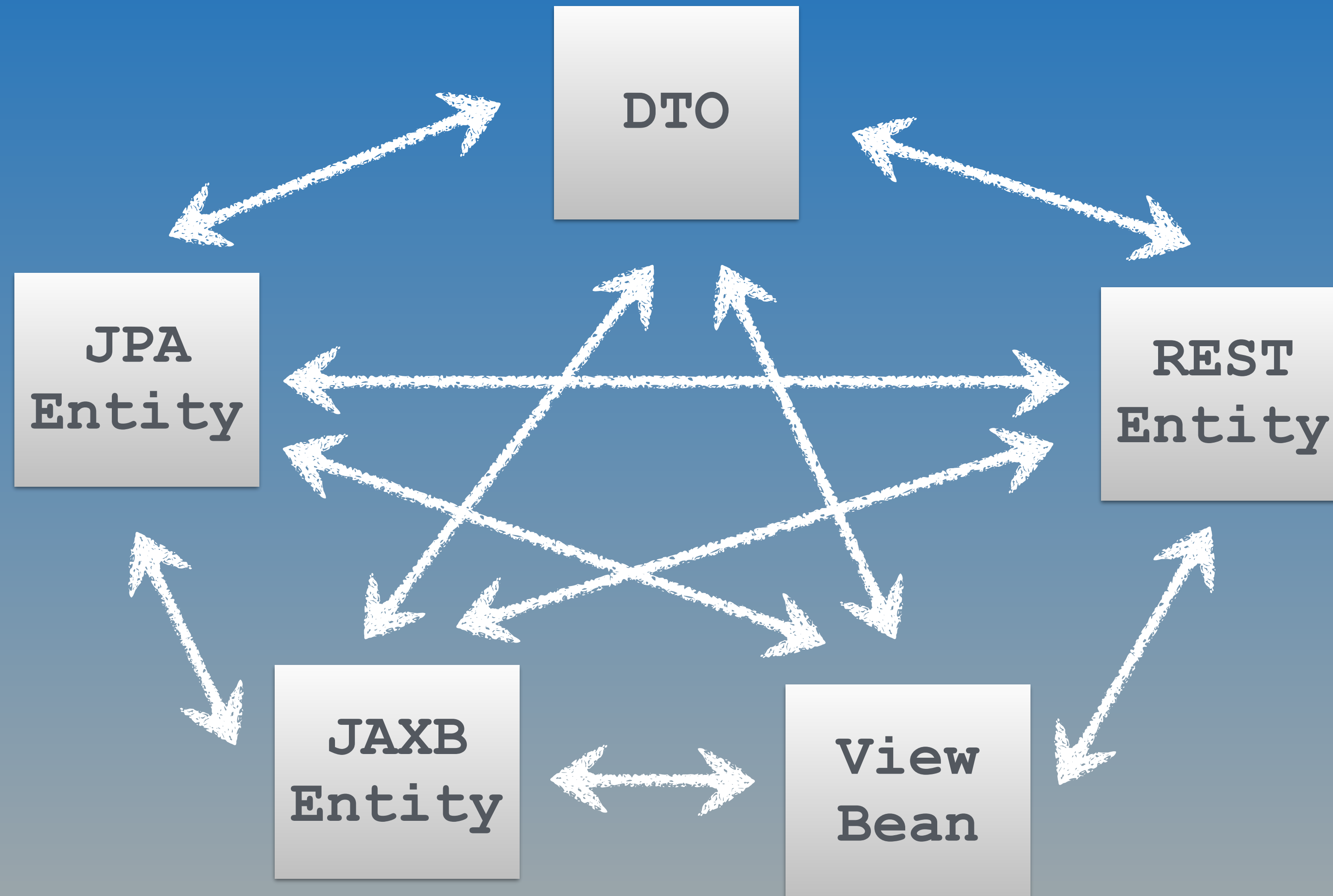
    String customerId;

    String name;

    String title;

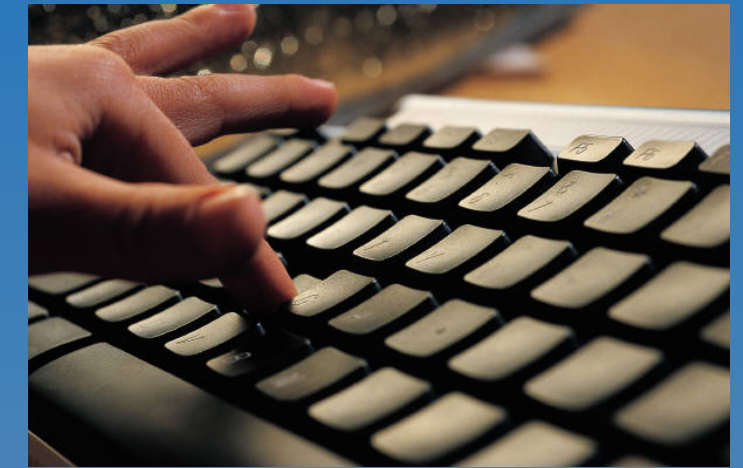
    ... getters and setters! ...
}
```


... can be quite tedious



Mapping implementation – how?

- DIY
 - Call getters / setters manually
 - Use own reflection library
- Reflection-based mapping libraries:
 - Apache BeanUtils, Dozer, ...
- Problems: Manual effort, missing type safety, performance ...



Performance... Performance!



https://twitter.com/joao_b_reis/status/559780053979250688



- Annotation processor, generates mapping (source) code.
- No reflection!
- Type safe and *fast*.
- Requires at least Java 6, special support for Java 8.
- Small runtime dependency (< 20 K)
or none at all (depending on the component model).

Versions

- MapStruct 1.0.Final released in 2015.
- MapStruct 1.1: **Rock solid. Used in production.**
- MapStruct 1.2: Beta.

JARs (Dependencies)



1. MapStruct Core JDK 8

org.mapstruct » mapstruct-jdk8

MapStruct annotations to be used with JDK 8 and later

Last Release on May 30, 2017

Java 8+



2. MapStruct Core

org.mapstruct » mapstruct

MapStruct Core

Last Release on May 30, 2017

Java 6 / 7



3. MapStruct Processor

org.mapstruct » mapstruct-processor

MapStruct Processor

Last Release on May 30, 2017

Java 6+, not needed at runtime

<http://mvnrepository.com/artifact/org.mapstruct>

Example – map JPA entity to DTO

```
@Entity
@Table(name = "CUSTOMERS")
public class Customer {

    @Id
    Long id;

    long customerId;

    String name;

    @Enumerated
    Title title;

    ... getters and setters! ...
}
```



```
public class CustomerDTO {

    long id;

    String customerId;

    String name;

    long title;

    ... getters and setters! ...
}
```

Our task:
Map all public properties
(including superclasses).

Make a wish!

```
public interface CustomerMapper {  
    CustomerDTO customer2DTO (Customer customer) ;  
}
```

Make a wish!


```
@Mapper(componentModel = "cdi")  
public interface CustomerMapper {
```

```
    CustomerDTO customer2DTO (Customer customer);
```

```
}
```

```
@Inject  
private CustomerMapper mapper;  
...  
  
CustomerDTO dto =  
    mapper.customer2DTO ( customer );
```

Generate the implementation

- How do we generate the implementation for the interface?
- **Press "Save"  in your IDE.**
- More precisely: Run the compiler (e.g. javac) ...
- ... and let the annotation processor kick in.



Live
Demo

Generated source code

```
@Generated(
    value = "org.mapstruct.ap.MappingProcessor",
    date = "2017-06-15T16:41:17+0200",
    comments = "version: 1.2.0.Beta3, compiler: Eclipse JDT (DE)"
)
@ApplicationScoped
public class CustomerMapperImpl implements CustomerMapper {

    @Override
    public CustomerDTO customer2DTO(Customer customer) {

        if (customer == null) {
            return null;
        }

        CustomerDTO customerDTO = new CustomerDTO();

        customerDTO.setCustomerId(String.valueOf(customer.getCustomerId()));

        if (customer.getId() != null) {
            customerDTO.setId(String.valueOf(customer.getId()));
        }

        customerDTO.setName(customer.getName());

        if (customer.getTitle() != null) {
            customerDTO.setTitle(customer.getTitle().name());
        }

        return customerDTO;
    }
}
```

Thread-safe.

Clean.
Easy to read.

@Generated attributes
can be turned off.

No container? No problem!

```
@Mapper
public interface CustomerMapper {

    CustomerMapper INSTANCE =
        Mappers.getMapper( CustomerMapper.class );

    CustomerDTO customer2DTO( Customer customer );

}
```

```
CustomerDTO dto =
    CustomerMapper.INSTANCE.customer2DTO( customer );
```

Type (& value) safety

```
CustomerMapper.java x CustomerDTO.java Customer.java
1 package mapper;
2
3+ import org.mapstruct.Mapper;[]
7
8 @Mapper
9 public interface CustomerMapper {
10
11 CustomerDTO customer2DTO(Customer customer);|
12
13 }
14
```

Problems x @ Javadoc Declaration

1 error, 1 warning, 0 others

Description	Resource
▼ Errors (1 item)	
✖ Can't map property "model.Title title" to "java.lang.Long title"....	CustomerMapper.java
▼ Warnings (1 item)	
⚠ Unmapped target property: "name".	CustomerMapper.java

WARN / ERROR / IGNORE

set policy globally
or per mapper
(or use default)

Basic mappings

```
@Mapper
public interface CustomerMapper {

    @Mappings({
        @Mapping(target="custNo", source="customerId"),
        @Mapping(target="password", ignore=true),
        @Mapping(target="lastLogin", dateFormat="dd.MM.yyyy")
    })
    CustomerDTO customer2DTO(Customer customer);

}
```

Mapping enum values

```
@Mapper
public interface CustomerMapper {

    @ValueMapping(target="HR", source="MR")
    @ValueMapping(target="FRU", source="MRS")
    @ValueMapping(target="FRK", source="MS")
    @ValueMapping(target="<NULL>", source="<ANY_UNMAPPED>")
    DtoTitle title2DtoTitle(Title title);

    CustomerDTO customer2DTO(Customer customer);

}
```

See [MappingConstants](#)

Lots of implicit type conversions

primitives ↔ wrappers

number types (incl. Big...) ↔ other number types & precisions

String ↔ nearly everything (incl. enums & date/number formats)

Date/Calendar ↔ Joda ↔ Java 8 date/time

JAXB ↔ elements, collections

Object references

```
@Mapper
public interface CustomerMapper {

    CustomerDTO customer2DTO (Customer customer);
}
```

That's all! 😊

```
@Entity
public class Customer {

    Address address;

    ... other properties ...
    ... getters and setters! ...
}
```

Customizing reference mapping

```
@Mapper(disableSubMappingMethodsGeneration=true)
public interface CustomerMapper {

    CustomerDTO customer2DTO(Customer customer);

    AddressDTO address2DTO(Address address);

}
```

Using other mappers

```
@Mapper(uses=AddressMapper.class)
public interface CustomerMapper {

    CustomerDTO customer2DTO(Customer customer);
}

@Mapper
public interface AddressMapper {

    AddressDTO address2DTO(Address address);
}
```

Custom mapping implementation

```
@Mapper
public interface CustomerMapper {

    CustomerDTO customer2DTO (Customer customer);

    default AddressDTO address2DTO (Address address) {
        AddressDTO dto = new AddressDTO ();
        ... custom mapping code ...
        return dto;
    }
}
```

Or use an abstract class instead.

Use your existing mapping code

```
@Mapper(uses=LegacyAddressMapper.class)
public interface CustomerMapper {
    CustomerDTO customer2DTO(Customer customer);
}

public class LegacyAddressMapper {
    public AddressDTO address2DTO(Address address) {
        ... custom mapping code ...
        return dto;
    }
}
```

Provide default constructor
or *static* mapping method.

Multiple source parameters

```
@Mapper
public interface PrintLabelMapper {

    @Mapping(target="name", source="customer.name")
    @Mapping(target="street", source="address.street")
    @Mapping(target="city", source="address.city")
    PrintLabelDTO customerAddress2DTO(
        Customer customer, Address address);
}
```

Path expressions can generally be used in "source" and "target".

Updates & reverse mappings

```
@Mapper
```

```
public interface CustomerMapper {
```

```
    CustomerDTO customer2DTO (Customer customer) ;
```

```
    void updateCustomerDTO (Customer customer,  
                             @MappingTarget CustomerDTO dto) ;
```

May return target
(for fluent API).

Max. 1 @MappingTarget!

```
    Customer dto2Customer (CustomerDTO dto) ;
```

```
}
```

Reuse mapping configurations

```
@Mapper
public interface CustomerMapper {

    @Mapping( ... )
    CustomerDTO customer2DTO (Customer customer);

    @InheritConfiguration (name="customer2DTO")
    void updateCustomerDTO (Customer customer,
        @MappingTarget CustomerDTO dto);

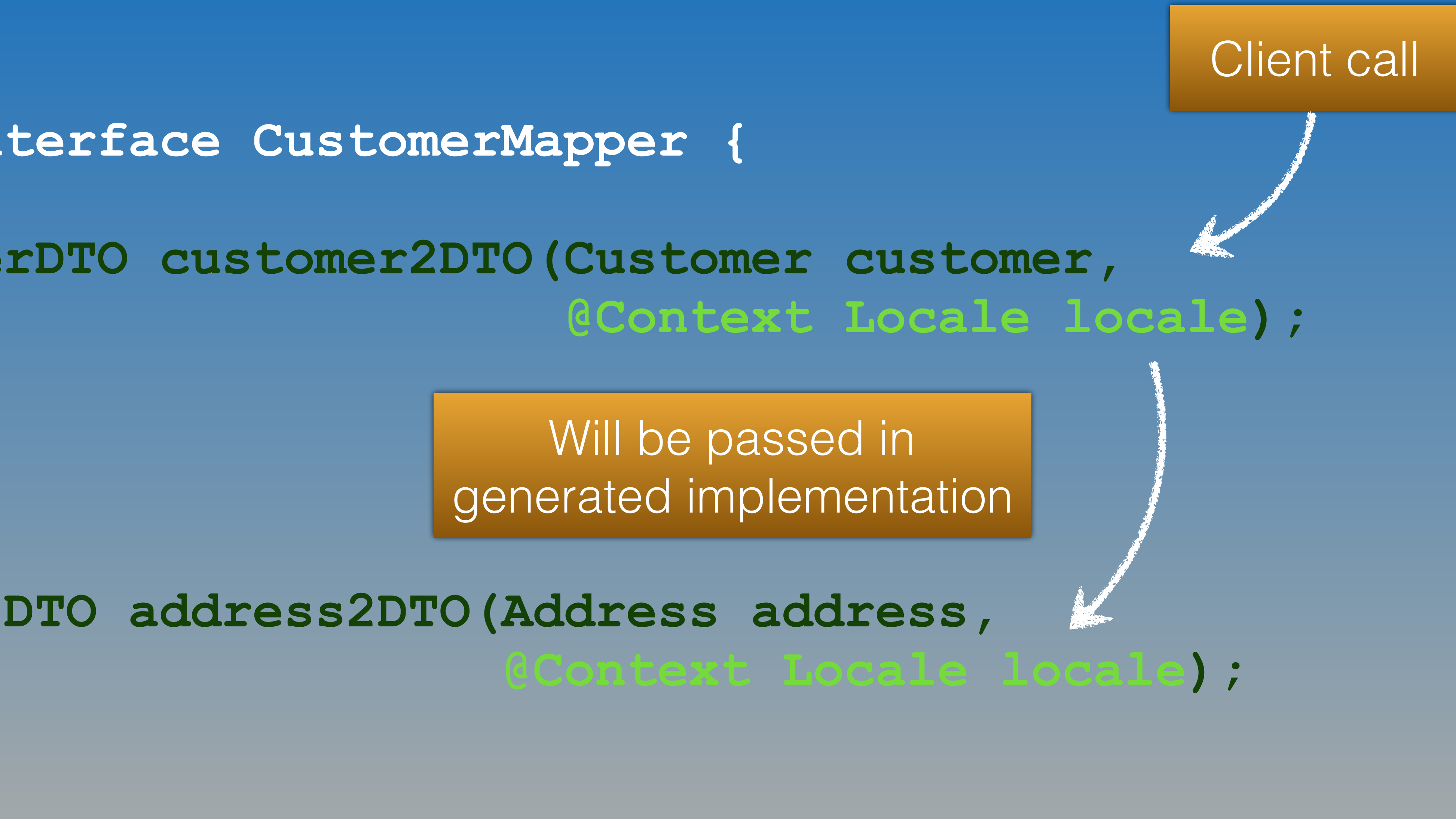
    @InheritInverseConfiguration (name="customer2DTO")
    Customer dto2Customer (CustomerDTO dto);
}
```

Context parameters

```
@Mapper
public interface CustomerMapper {

    CustomerDTO customer2DTO (Customer customer,
                             @Context Locale locale);

    AddressDTO address2DTO (Address address,
                            @Context Locale locale);
}
```



Client call

Will be passed in generated implementation

Target type in custom mappers

```
@ApplicationScoped
public class JpaEntityManager {

    @PersistenceContext
    private EntityManager manager;

    public <T extends BaseEntity> T resolve(Long id,
                                           @TargetType Class<T> entityClass) {
        return (id != null) ?
            manager.find(entityClass, id) : null;
    }
}
```


Object factories

```
public class DtoFactory {  
    public CustomerDTO createCustomerDTO() {  
        return new CustomerDTO();  
    }  
}
```

```
public class EntityFactory {  
    public <T extends BaseEntity>  
        T createEntity(@TargetType Class<T> entityClass) {  
        return entityClass.newInstance();  
    }  
}
```

Streams, collections ... of beans

@Mapper

```
public interface CustomerMapper {
```

```
    CustomerDTO customer2DTO (Customer customer);
```

```
    List<CustomerDTO> list2List (List<Customer> customer);
```

```
    List<CustomerDTO> array2List (Customer[] customer);
```

```
    CustomerDTO[] set2Array (Set<Customer> customer);
```

```
    List<CustomerDTO> stream2List (Stream<Customer> cust);
```

```
    Stream<CustomerDTO> set2Stream (Set<Customer> cust);
```

```
}
```

Streams, collections ... of objects

```
@Mapper
public interface StreamCollectionMapper {

    Set<Long> int2Long(Stream<Integer> stream);
    Stream<Long> int2Long(List<Integer> list);

    @IterableMapping(dateFormat="dd.MM.yyyy")
    List<String> date2String(List<Date> list);

    @MapMapping(valueDateFormat="dd.MM.yyyy")
    Map<String, String> map2map(Map<Long, Date> map);
}
```

Exceptions

Every checked exception

not declared in the mapping method's signature

will be rethrown as a `RuntimeException`.

Customizing around invocation

- **Callbacks**

- @BeforeMapping, @AfterMapping

- **Decorators**

- @DecoratedWith built-in – but better use CDI decorators.

MapStruct SPI

- Customize **accessor naming strategy**
 - e.g. fluent API instead of get/set
- Mapping **exclusions**
 - Exclude certain types from automatic method generation

There's a lot more ...

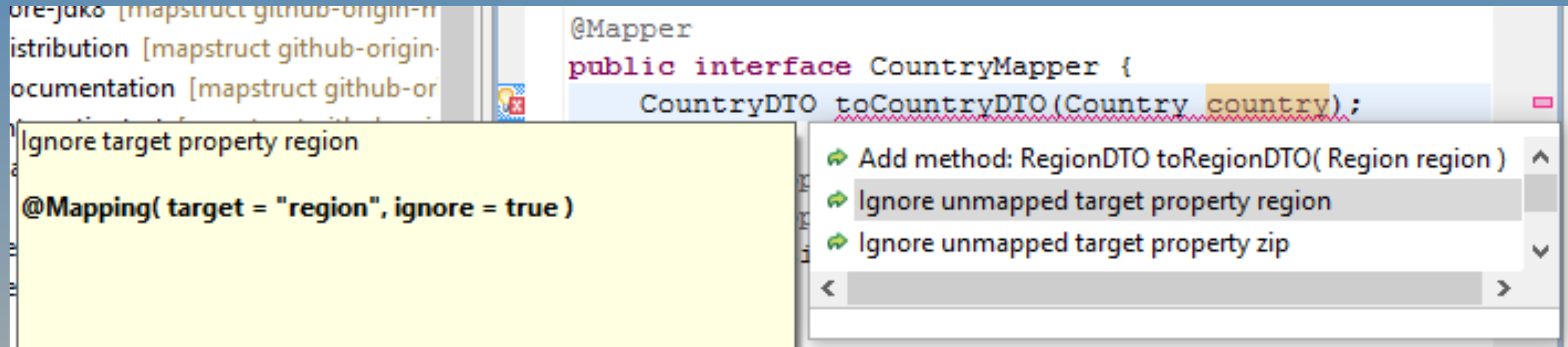
- Constant values & Java expressions as "source".
- Default values, if source is null.
- Order of setter invocation ("dependsOn").
- Customized null checks & default null values.
- Centralized mapping configurations ("config", @MapperConfig).
- Selection of ambiguous mapping methods by qualifier or name.

Build tool integration

- javac
 - SPI – just put MapStruct on the classpath!
- Maven, Ant, Gradle
 - See documentation for (easy) setup instructions:
 - Add MapStruct dependency, setup annotation processor.

Eclipse plug-in

- Code completion
- Quick fixes
- <https://github.com/mapstruct/mapstruct-eclipse>



New in MapStruct 1.2

- Support for Java 8 streams.
- Mappings based on public fields.
- Automatic generation of sub-mapping methods.
- Integration with Project Lombok.
- Support for Java 9 (experimental).

MapStruct – Recap

- Meets current, real-world bean mapping needs.
- Generated code is fast, clean and easy to read.
- Flexible mappings, built-in and custom.
- Easy integration with different component models.
- Extensive and well-written documentation.



Curious? Intrigued? Questions?



<http://mapstruct.org/>

<https://github.com/mapstruct/mapstruct-examples>



Thank you:



Gunnar Morling is the original author of MapStruct and leads the project.

He is a long-time Java developer and open-source committer. He is part of the [Hibernate](#) team at [Red Hat](#), where he works on Hibernate OGM, Validator and Search. You can follow him on [Google+](#) and [Twitter](#) or check out his [Blog](#).



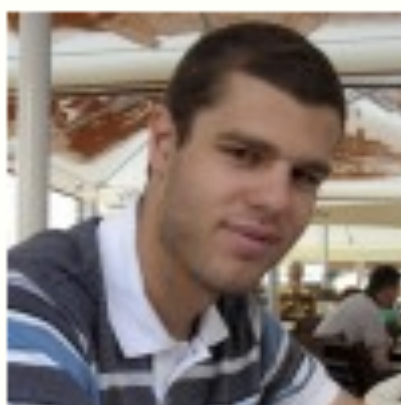
Andreas Gudian was the first committer to join Gunnar in his efforts.

He is an experienced developer and architect for enterprise Java applications, where MapStruct can make a real difference. Andreas contributes to various open source projects and is also committer at the Apache Maven project.



Sjaak Derksen, enthusiastic first-hour user of MapStruct

He has well over 15 years of experience in Java / JEE development as architect, technical lead, developer and tester in the domain of Telecommunications. Sjaak started working more recently on spatial subsurface data interchange (e.g. Inspire, GML) where he believes MapStruct can be a true asset, reducing the amount of repetitive, error-prone work.



Filip Hrisafov, the newest member of the MapStruct team

He is a young Java developer and consultant, who uses MapStruct to help him in his daily work on Java enterprise applications. He is passionate about Open Source and contributes to various open source projects.

Thank you!

thomas@muchsoft.com

www.javabarista.de

 @thmuch

1100 1010 1111 1110 1011 1010 1011 1110