Hibernate Introduction

CS544: Enterprise Architecture

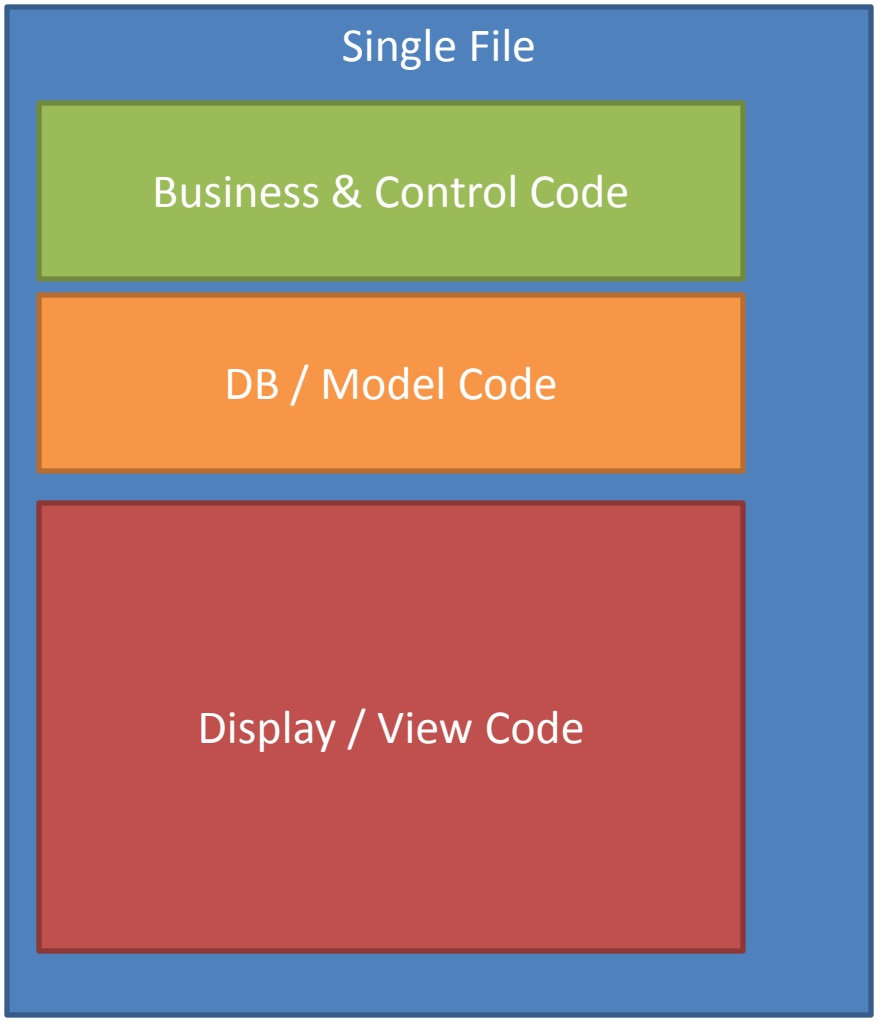
Introduction

- We will look at the architectural requirements of enterprise applications and how this relates to the technologies that we will be using for this course.
- We will look at what the different logical layers are as an application grows
- We will look at what Spring is and how it relates to the different layers
- We will look at what Hibernate is and how it relates to the different layers

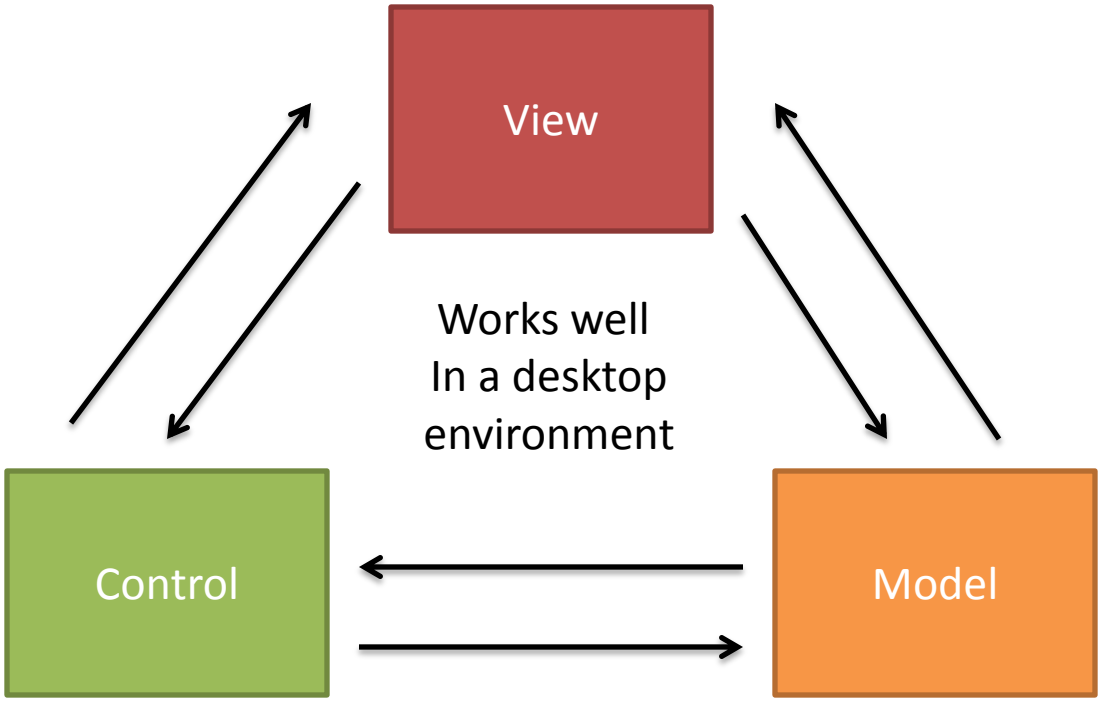
Course Introduction:

ARCHITECTURE

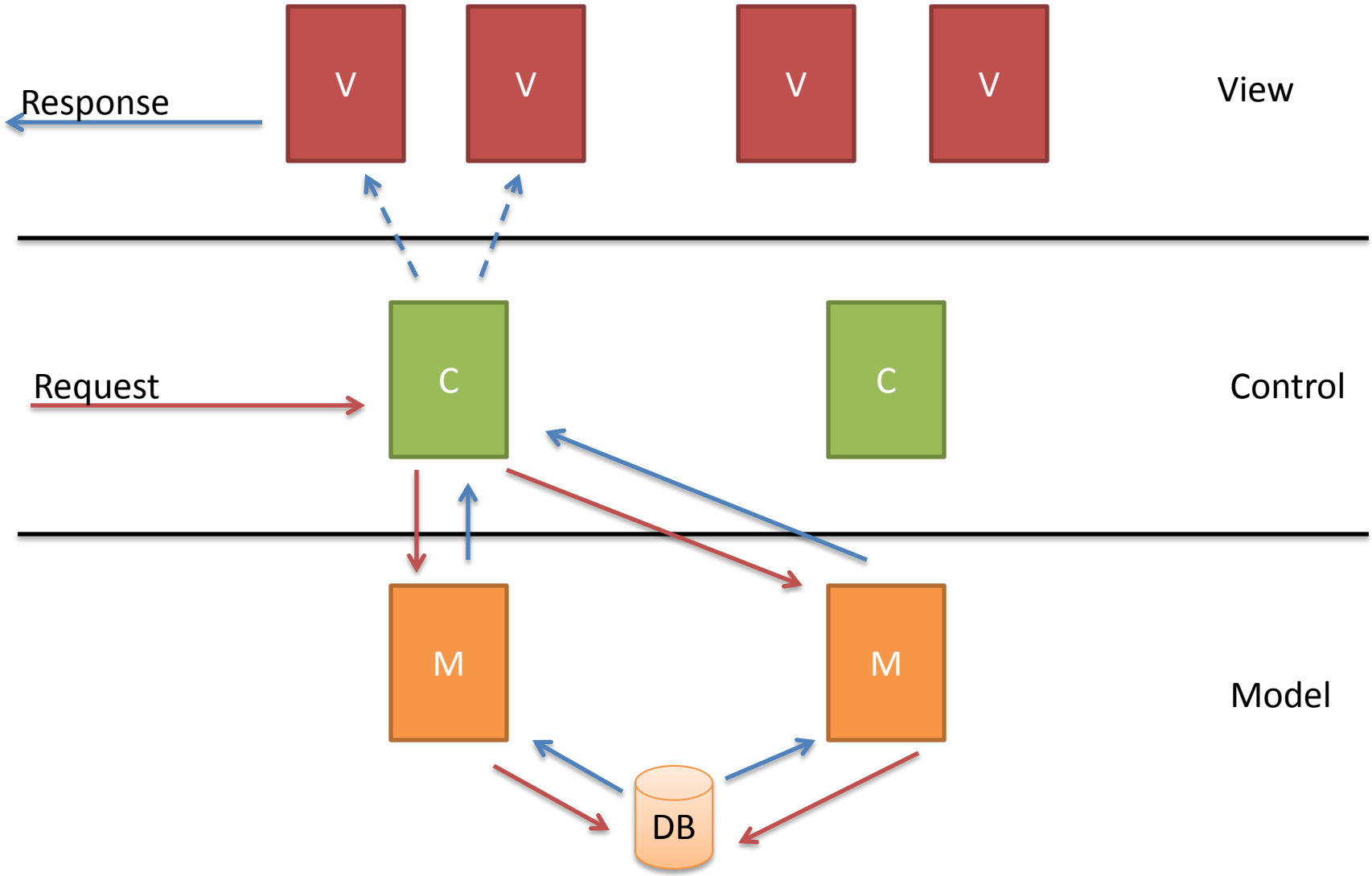
Model 1



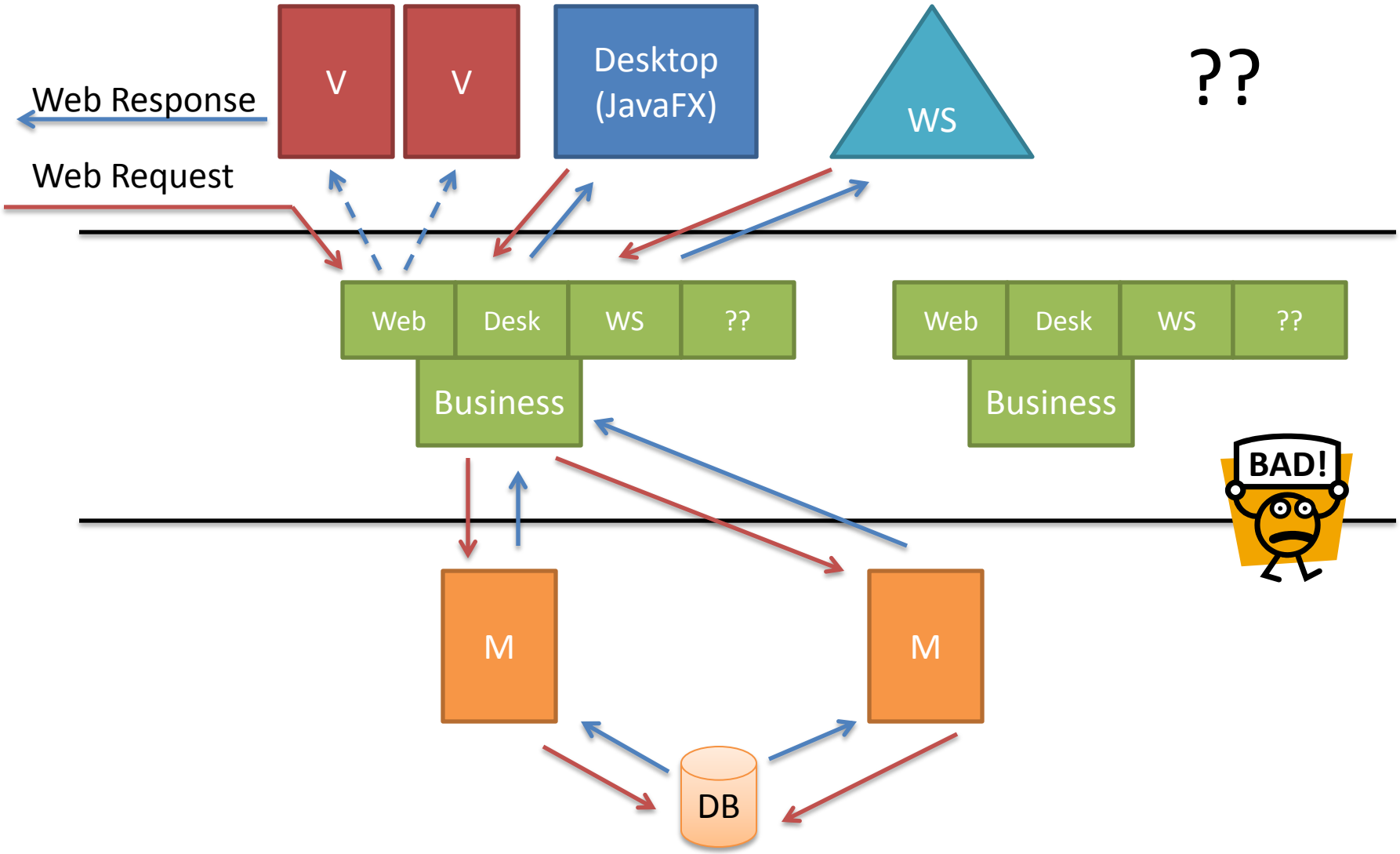
Classic MVC (Model 2)



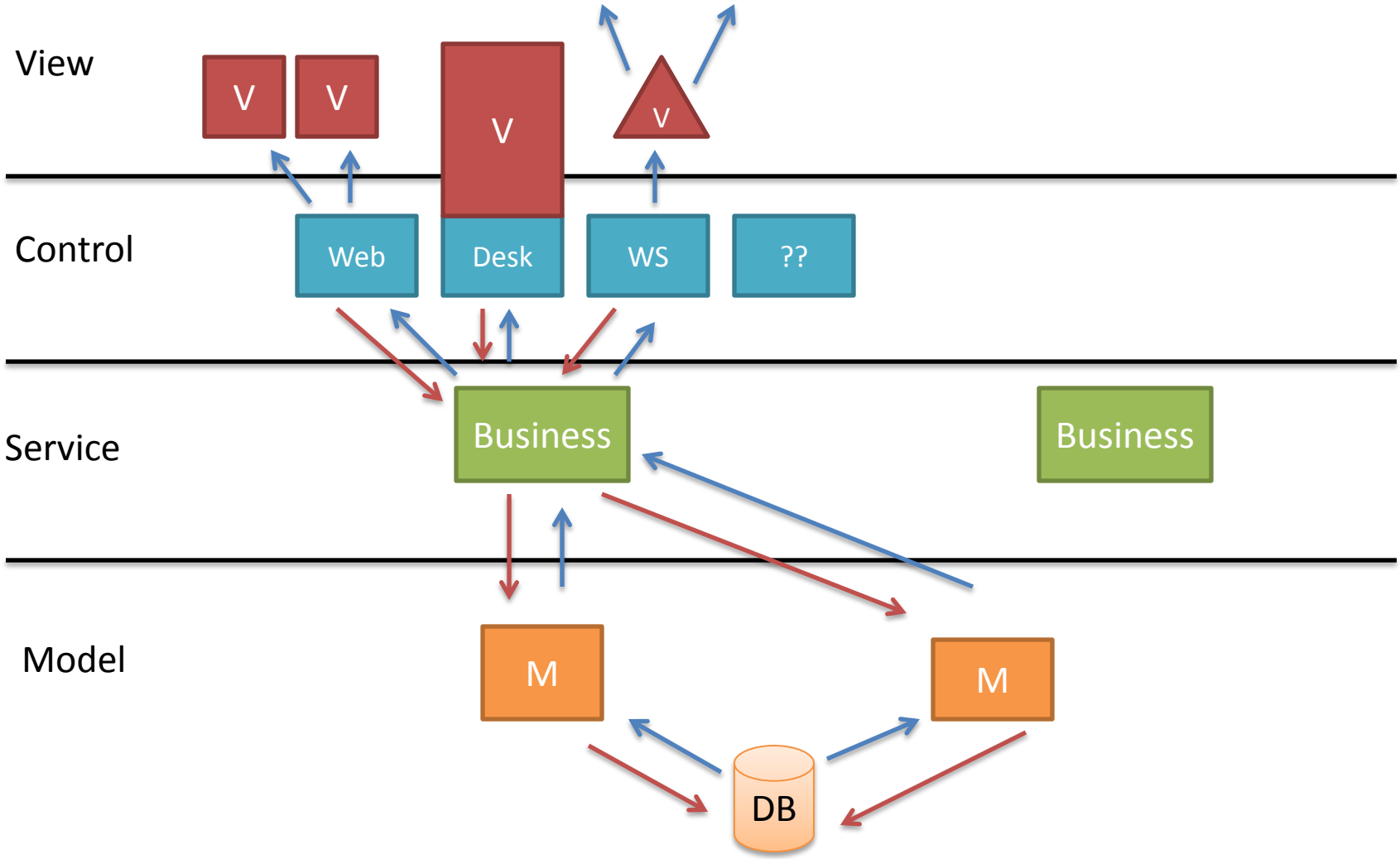
Three Tier / Web MVC



Multiple Types of Clients



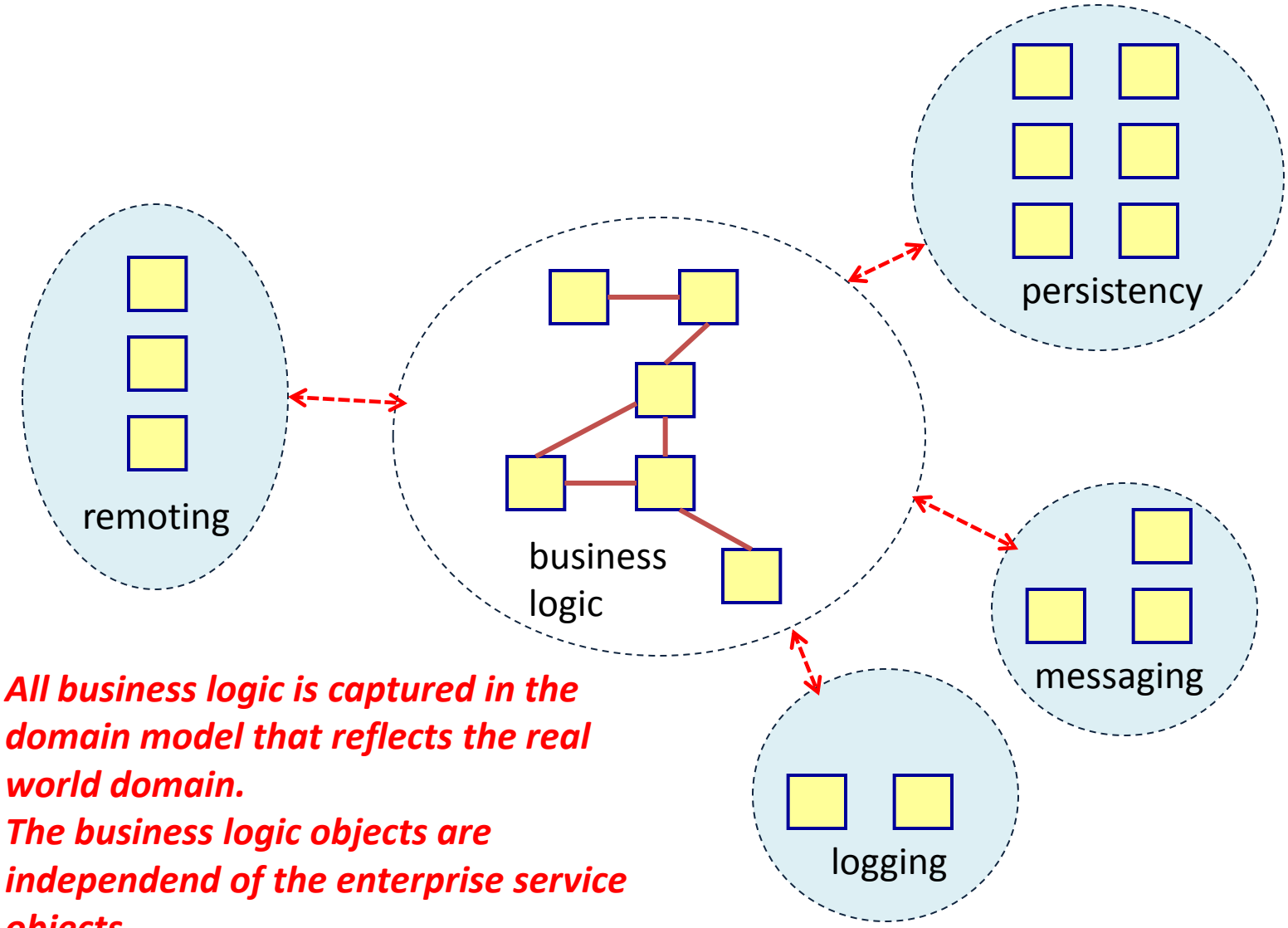
Service Oriented Architecture



Course Introduction:

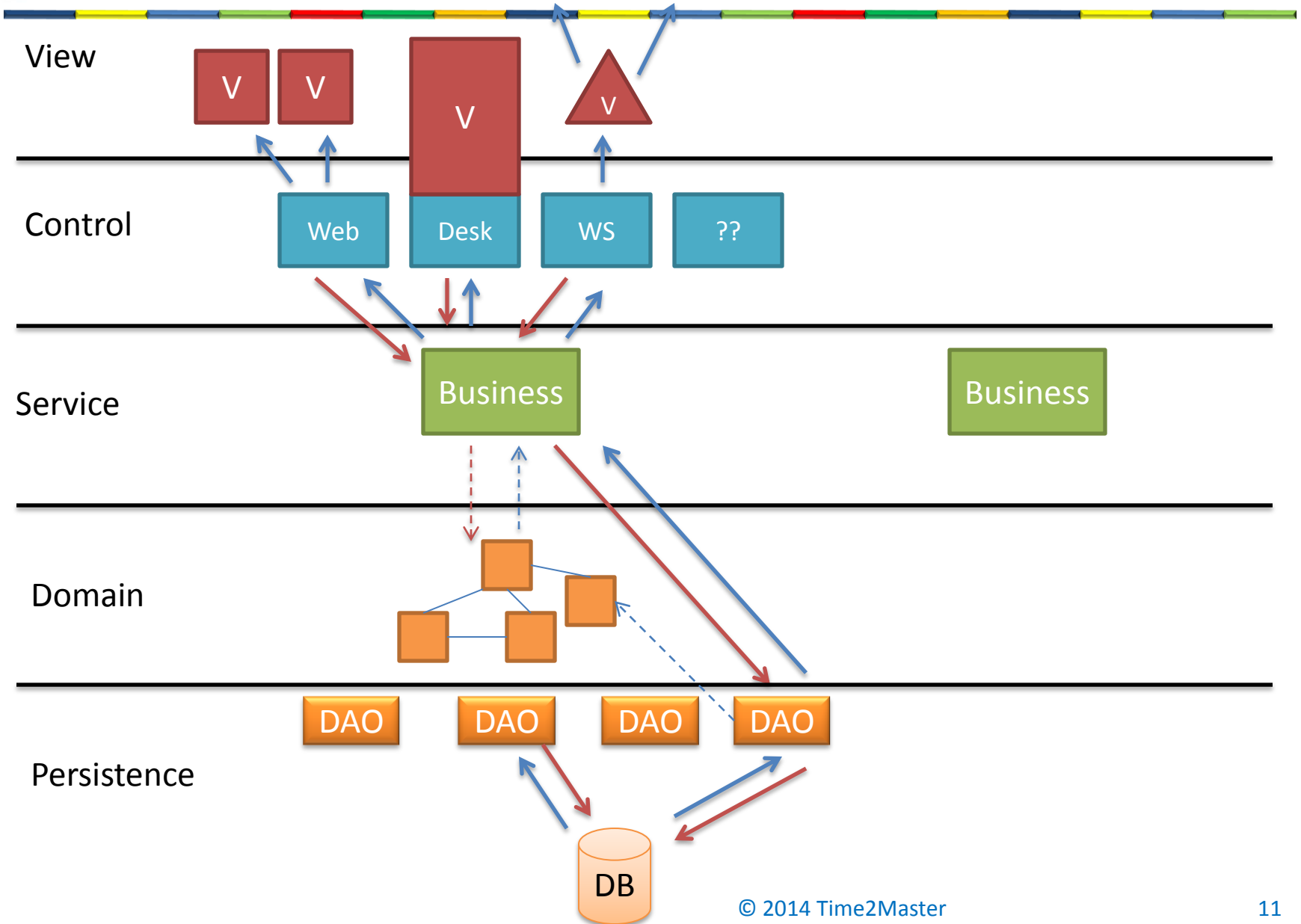
PRINCIPLES

Domain-Driven Design (DDD)

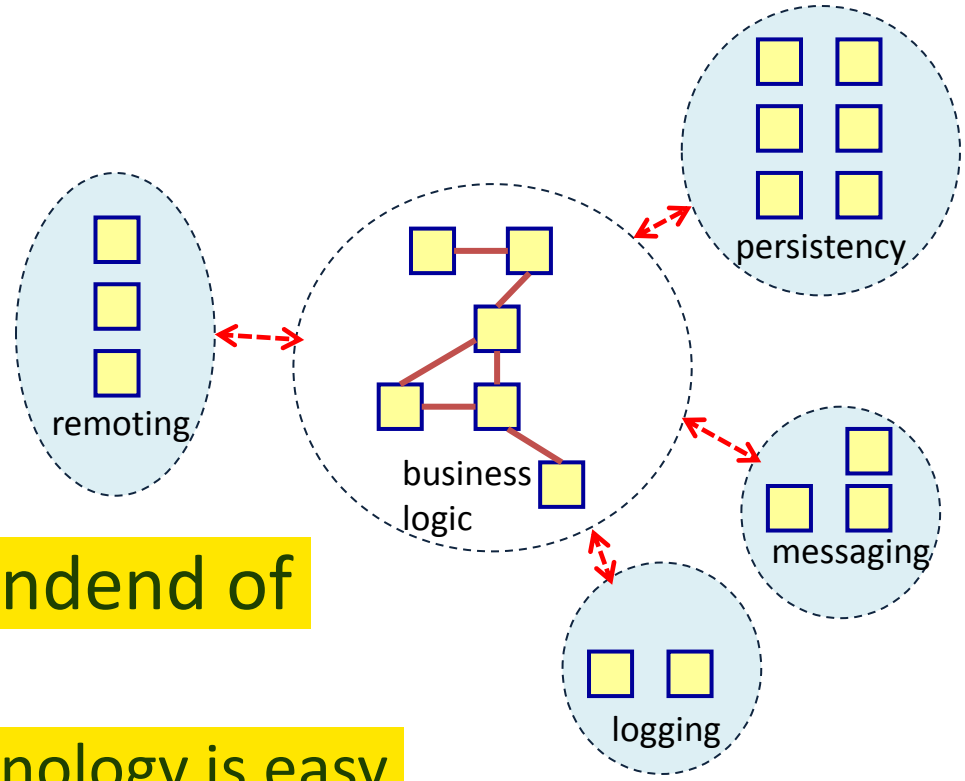


1. ***All business logic is captured in the domain model that reflects the real world domain.***
2. ***The business logic objects are independend of the enterprise service objects***

Service Oriented Architecture



Advantages of DDD

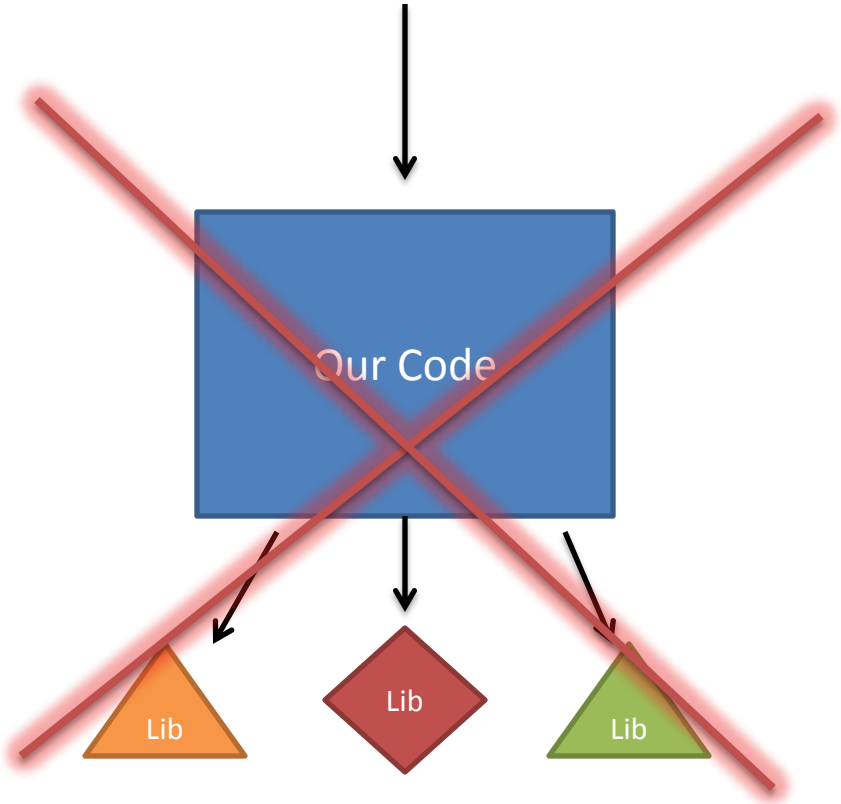


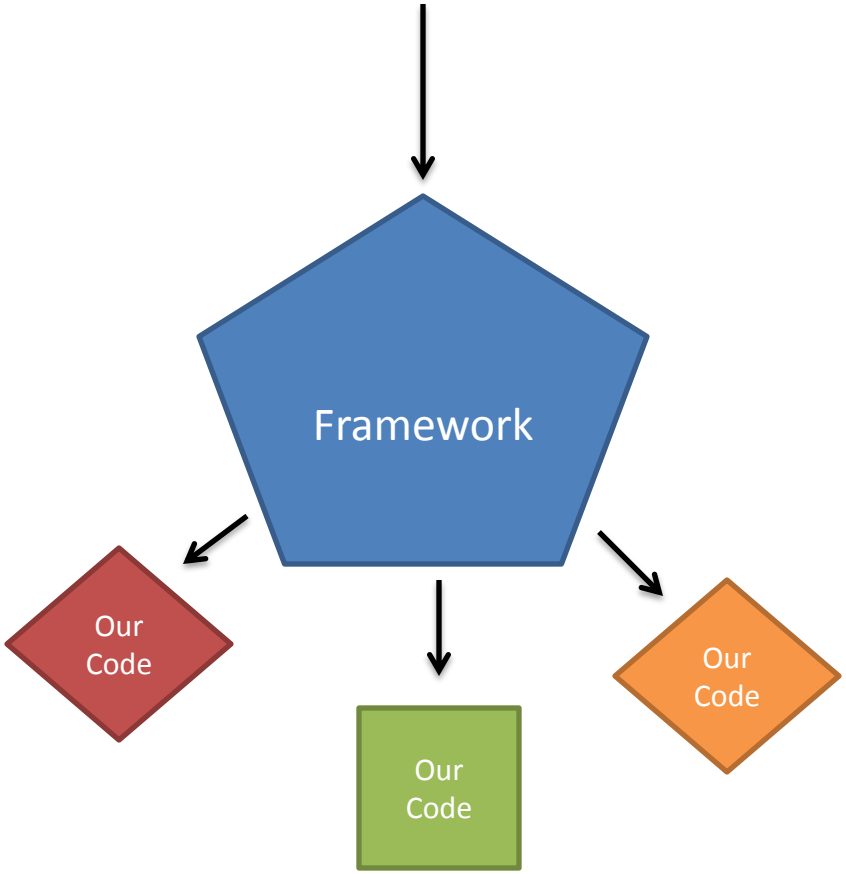
- Business logic is independend of technology changes
  - Switching between technology is easy
- Business logic is easy to understand
  - Easy to write, test, modify

12

Frameworks / Inversion of Control

- The Hollywood Principle:
  - Don't call us, we'll call you





Declarative Programming - Annotations or XML -

- Service Helpers
  - Transactions
  - Security
  - Logging
  - AOP
- Object Relational Mapping
  - Identity
  - Attributes
  - Associations
  - Meta Data

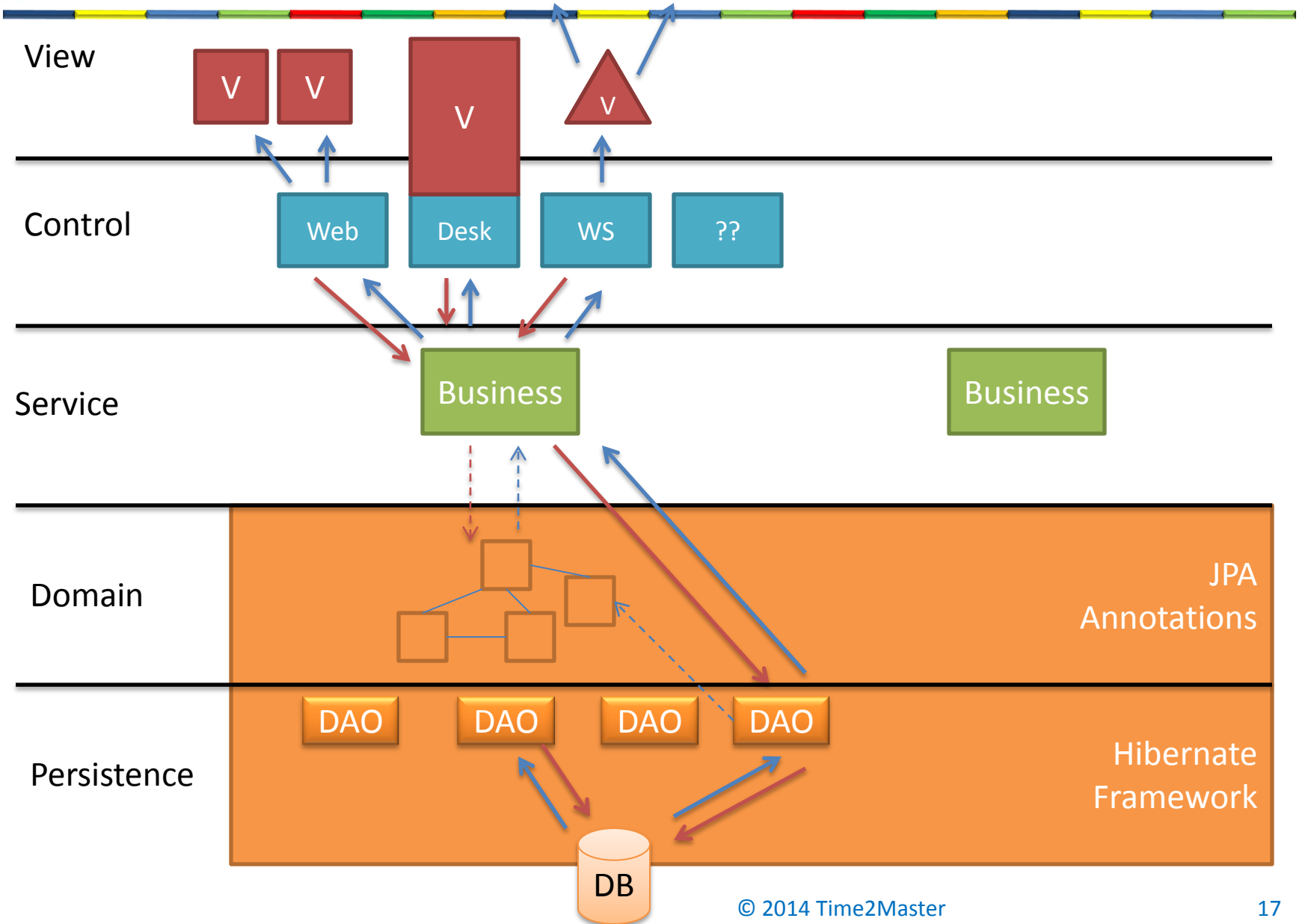
Separation of Concerns

- Different Architectural Layers
- Plain Old Java Objects
  - Java Bean Standard
- In Summary everything is about SoC:
  - Separate Business from Technology

Course Introduction:

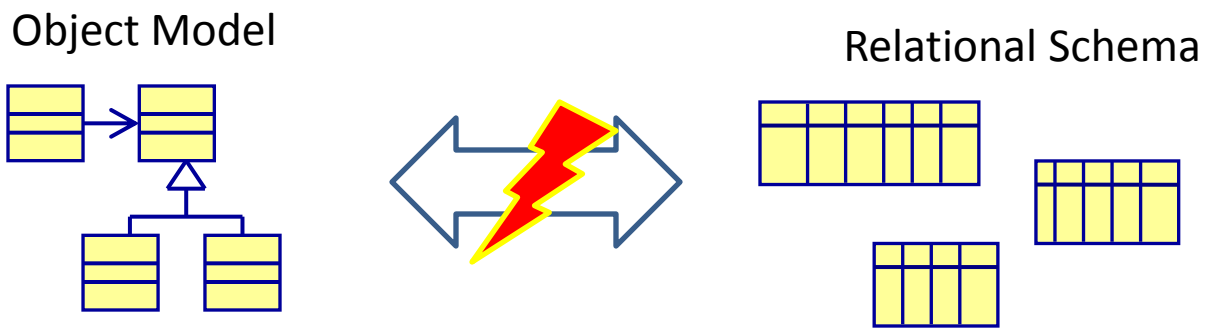
HIBERNATE

Framework for the persistence layer



Object-Relational Mismatch

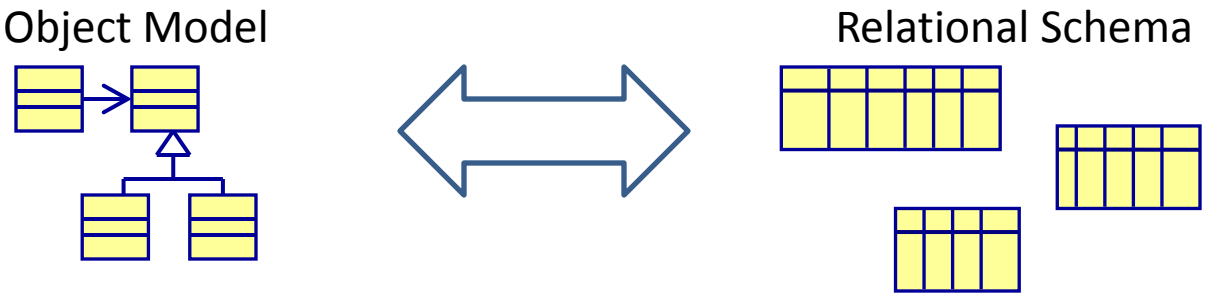
Object Oriented	Relational Database
Objects are instantiations of classes and have identity (object1 == object2)	In the relational model the table name and primary key are used to identity a row in a table
Objects have associations (one-to-one, many-to-one, …)	Relational model has foreign keys and link tables
OO has inheritance	Relational model has no such thing
Data can be accessed by following object associations	Data can be accessed using queries and joins



Java Persistence Possibilities

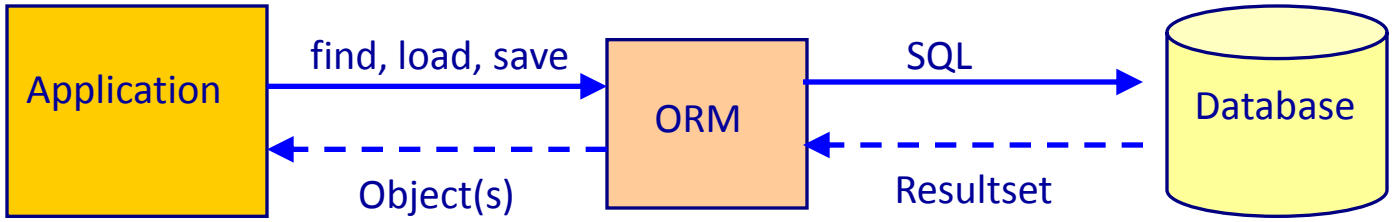
Possibility	Example
Stored Procedures	Stored PL/SQL or Transact-SQL procedures
SQL in the Application	Putting SQL in strings inside the application, using the JDBC API straight or wrapped by the Spring JDBC template
iBatis SQL maps	Moving SQL into XML configuration removing JDBC plumbing code overhead
Entity Beans 2.1	Using a Java Enterprise Edition 2.1 application server with Entity Beans
Object Relational Mapping	Using tools such as Hibernate, Toplink, JDO, and JPA to map an Object Model onto a Relational Schema

More OO Friendly



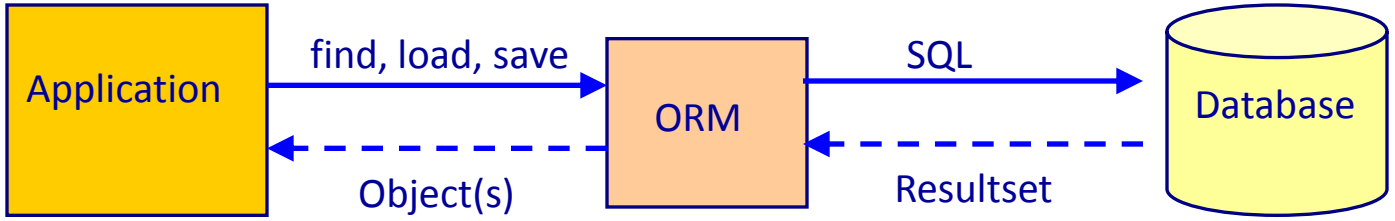
Object Relational Mapping (ORM)

- Object Relational Mapping lets the programmer focus on the Object Model
  - Supports Domain Driven Development (DDD)
  - Programmer can just work with objects
  - Once an object has been retrieved any related objects are automatically loaded as needed
  - Changes to objects can automatically be stored in the database



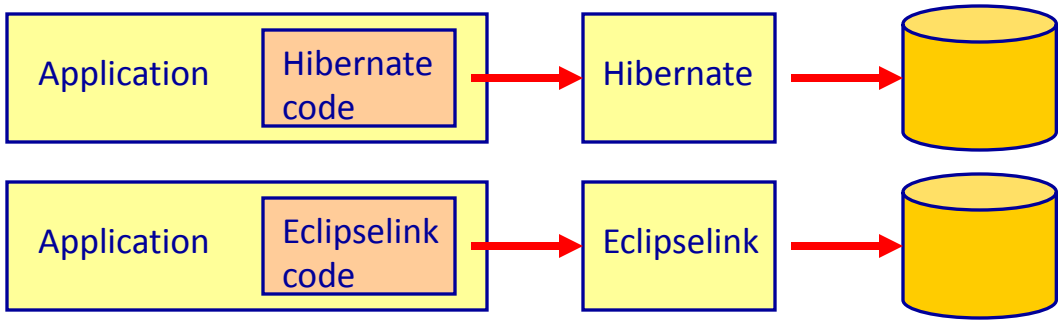
Advantages of ORM

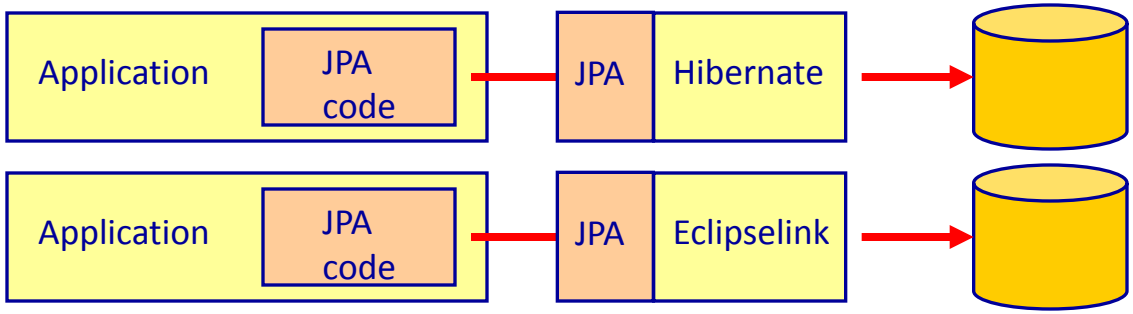
Advantage	Details
Productivity	•Fewer lines of persistency code
Maintainability	•Fewer lines of persistency code •Mapping is defined in one place
Performance	•Caching •Higher productivity gives more time for optimization ✓Projects under time pressure often don't have time for optimization •The developers of the ORM put a lot of effort in optimizing the ORM



The Java Persistence API (JPA)

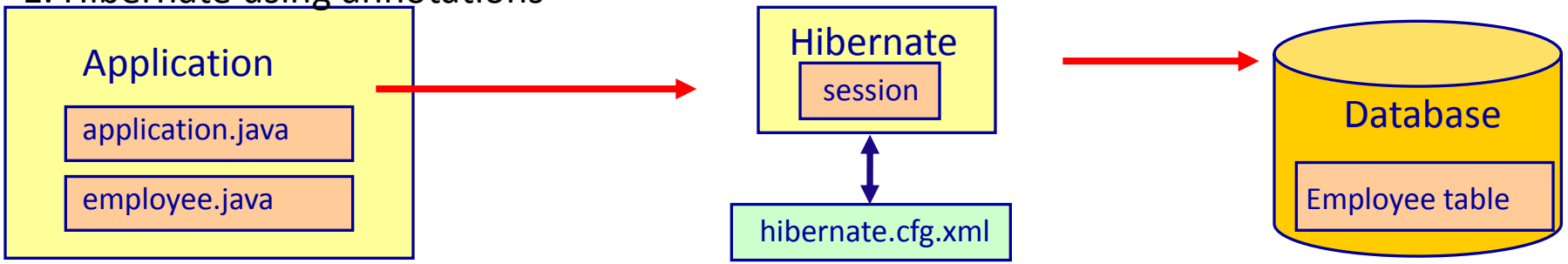
- JPA is a Java standard for ORM persistency



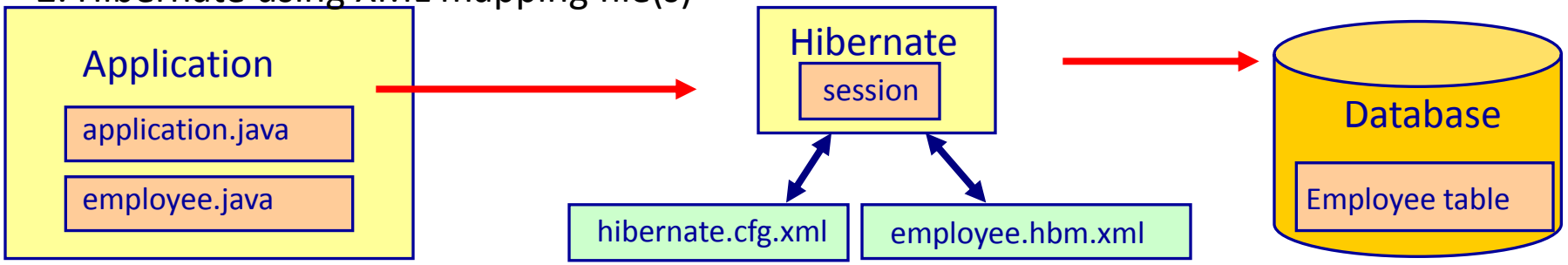


3 ways to use Hibernate

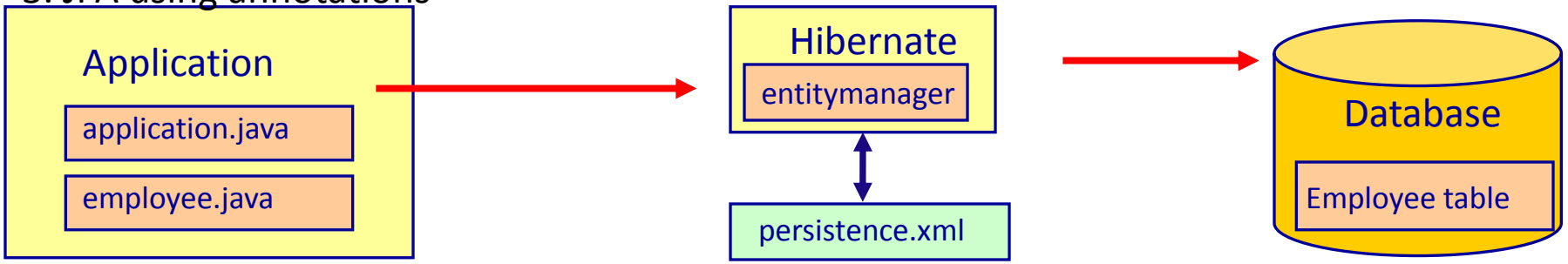
1. Hibernate using annotations



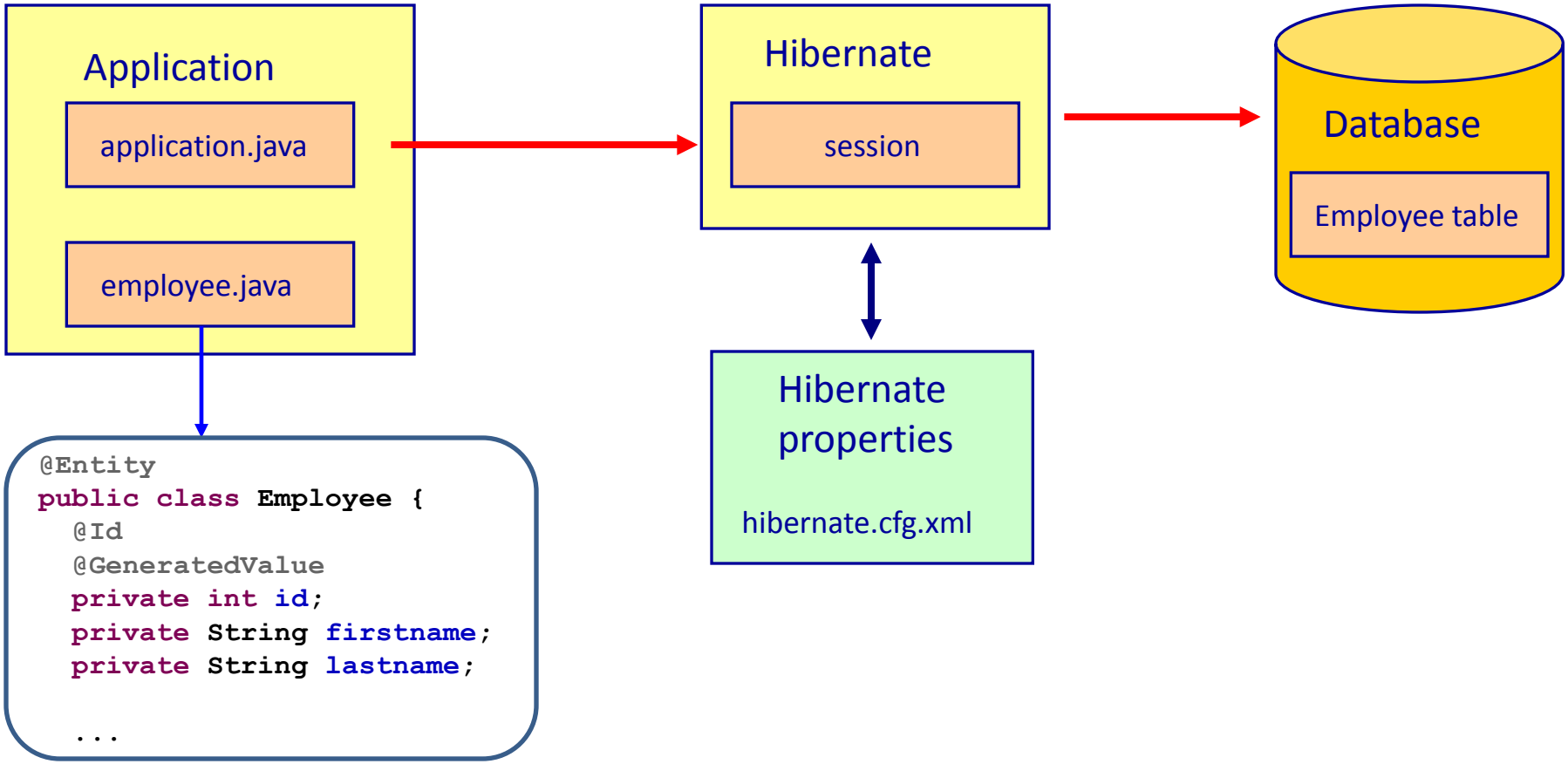
2. Hibernate using XML mapping file(s)



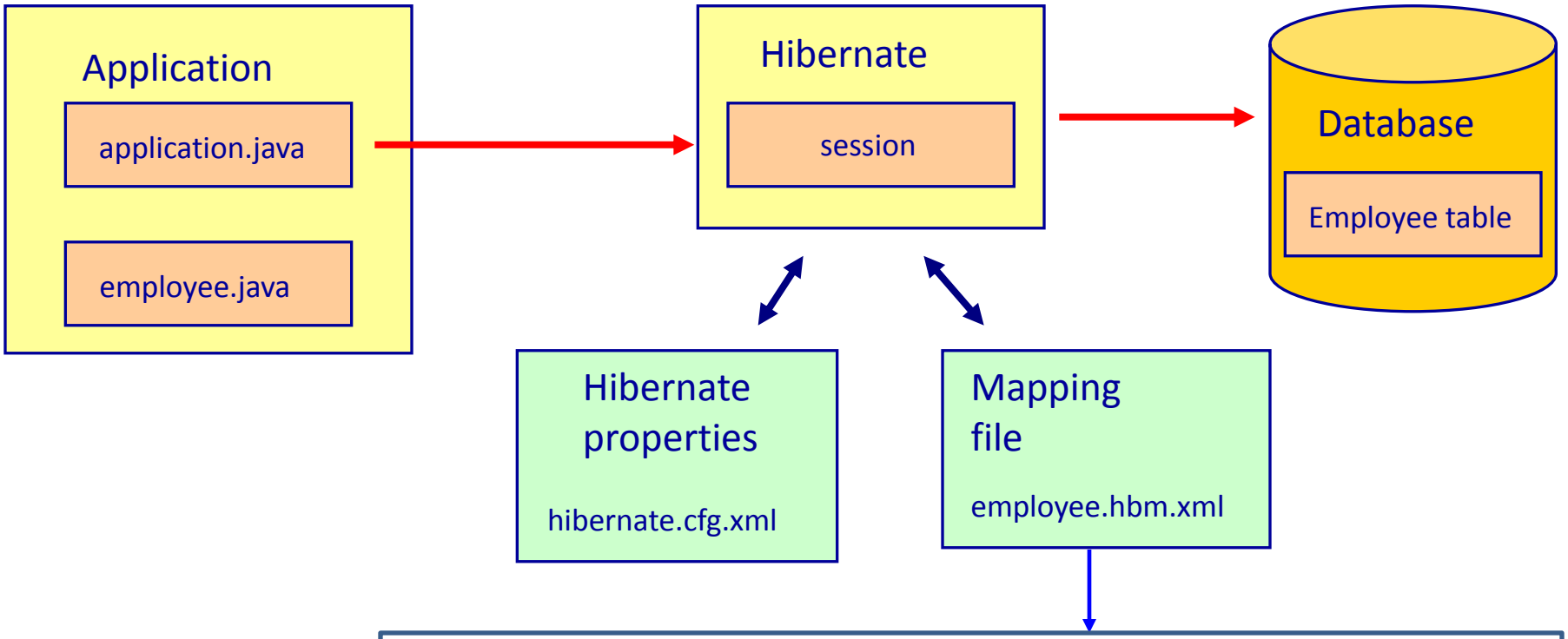
3. JPA using annotations



1. Hibernate using Annotations

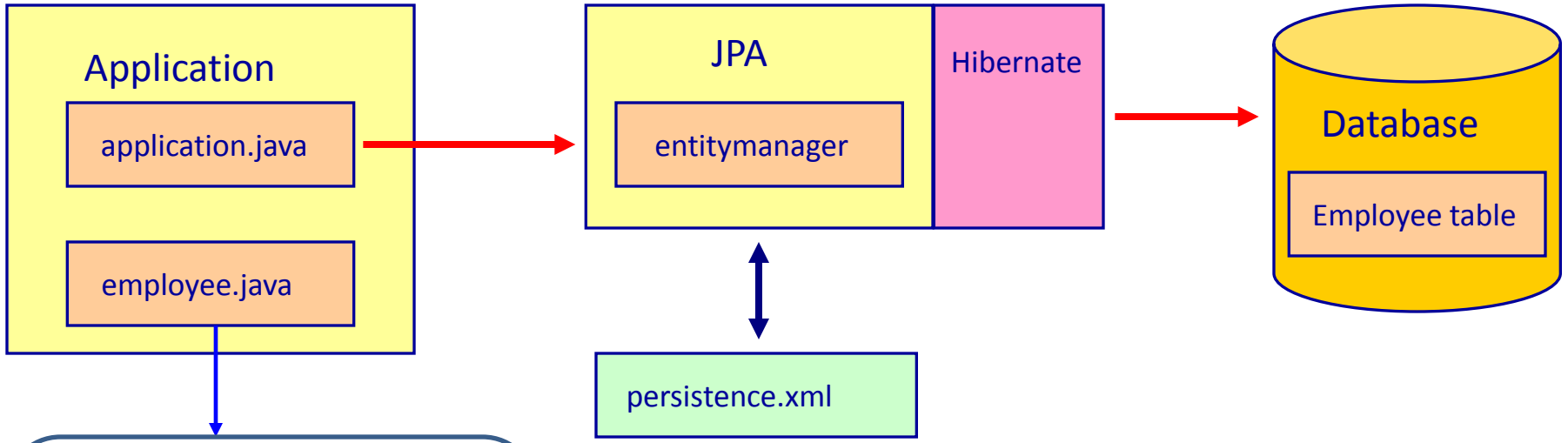


2. Hibernate using XML mapping file(s)




```
<hibernate-mapping>
  <class name="intro.xml.Employee" table="employee">
    <id name="id" type="int" column="ID" >
      <generator class="increment" />
    </id>
    <property name="firstname" column="firstname" type="string" />
    <property name="lastname" column="lastname" type="string" />
  </class>
</hibernate-mapping>
```

3. JPA using Annotations




```
@Entity
public class Employee {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;

  ...
```

Simple Hibernate Example

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Employee {

  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;

  public Employee() { }

  …
}
```

Employee table

id	firstname	lastname

Hibernate Annotations Configuration

hibernate.cfg.xml

```
<?xml version="1.0" encoding="windows-1252" ?>
 <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- MySQL DB running on localhost -->
        <property name="connection.url">jdbc:mysql://localhost/test</property>
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.username">root</property>
        <property name="connection.password">root</property>
        <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
         <!-- Mapping files -->
         <mapping class="intro.annotations.Employee"/>
    </session-factory>
</hibernate-configuration>
```

An XML Example

```
public class Employee {
    private String firstname;
    private String lastname;
    private int id;

    public Employee() {
    }
    …
}
```

Every entity must have a null argument constructor

Employee table

id	firstname	lastname

Employee.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >

<hibernate-mapping>
  <class name="intro.xml.Employee" table="employee">
    <id name="id" type="int" column="ID" >
      <generator class="increment" />
    </id>
    <property name="firstname" column="firstname" type="string" />
    <property name="lastname" column="lastname" type="string" />
  </class>
</hibernate-mapping>
```

Hibernate Configuration File

hibernate.cfg.xml

```
<?xml version="1.0" encoding="windows-1252" ?>
 <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- MySQL DB running on localhost -->
        <property name="connection.url">jdbc:mysql://localhost/test</property>
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.username">root</property>
        <property name="connection.password">root</property>
        <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
         <!-- Mapping files -->
         <mapping resource="intro/xml/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Hibernate Application Example

```
public class Application {
    private static SessionFactory sessionFactory;

    /* Reads hibernate.cfg.xml and prepares Hibernate for use     */
    protected static void setUp() throws Exception {
        // A SessionFactory is set up once for an application!
        final StandardServiceRegistry registry = new StandardServiceRegistryBuilder()
                .configure("cs544/hibernate_intro/annotations/hibernate.cfg.xml")
                .build();
        try {
            sessionFactory = new MetadataSources(registry).buildMetadata().buildSessionFactory();
        } catch (Exception e) {
            e.printStackTrace();
            StandardServiceRegistryBuilder.destroy(registry);
        }
    }

    protected static void tearDown() throws Exception {
        if (sessionFactory != null) {
            sessionFactory.close();
        }
    }
```

Hibernate Application Continued

```
public static void main(String[] args) throws Exception {
        setUp();

        Session session = sessionFactory.openSession();
        session.beginTransaction();

        // Create new instance of Employee and set values in it
        Employee employee = new Employee();
        employee.setFirstname("Frank");
        employee.setLastname("Miller");
        // save the employee
        session.persist(employee);

        session.getTransaction().commit();
        session.close();

        session = sessionFactory.openSession();
        session.beginTransaction();

        // retieve all employees
        List<Employee> employeeList = session.createQuery("from Employee").list();
        for (Employee emp : employeeList) {
            System.out.println("firstname= " + emp.getFirstname()
                    + ", lastname= " + emp.getLastname());
        }
        session.getTransaction().commit();

        // Close the SessionFactory (best practice)
        tearDown();
    }
}
```

Output:

```
firstname= Frank, lastname= Miller
```

Hibernate configuration :show_sql

hibernate.cfg.xml

```
<?xml version="1.0" encoding="windows-1252" ?>
 <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- MySQL DB running on localhost -->
        <property name="connection.url">jdbc:mysql://localhost/test</property>
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.username">root</property>
        <property name="connection.password">root</property>
        <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>

         <!-- Show all SQL DML executed by Hibernate -->
         <property name="show_sql">true</property>
         <!-- Mapping files -->
         <mapping resource="intro/xml/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Show the SQL that Hibernate sends to the database

Example Output:

```
Hibernate: insert into Employee (id, firstname, lastname) values (null, ?, ?)
Hibernate: call identity()
Hibernate: select employee0_.id as id0_, employee0_.firstname as firstname0_,
employee0_.lastname as lastname0_ from Employee employee0_
firstname= Frank, lastname= Miller
```

Hibernate configuration :hbm2ddl

hibernate.cfg.xml

```
<?xml version="1.0" encoding="windows-1252" ?>
 <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
 <!-- MySQL DB running on localhost -->
        <property name="connection.url">jdbc:mysql://localhost/test</property>
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.username">root</property>
        <property name="connection.password">root</property>
        <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>

         <property name="hbm2ddl.auto">create</property>
         <!-- Needed since Hibernate 5 to generate identity columns properly -->
         <property name="hibernate.id.new_generator_mappings">false</property>

         <!-- Show all SQL DML executed by Hibernate -->
         <property name="show_sql">true</property>
         <!-- Mapping files -->
         <mapping resource="Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Create the database tables during the starttup of the application

JPA Example

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Employee {

  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;

  public Employee() { }

  …
}
```

No difference in annotations

Employee table

id	firstname	lastname

JPA Configuration

META-INF/persistence.xml

```
<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="cs544.hibernate.intro.jpa">
        <description>
            Persistence unit for the Hibernate Introduction
        </description>

        <class>cs544.hibernate_intro.jpa.Employee</class>

        <properties>
            <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
            <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost/test" />
            <property name="javax.persistence.jdbc.user" value="root" />
            <property name="javax.persistence.jdbc.password" value="root" />

            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.hbm2ddl.auto" value="create" />
        </properties>
    </persistence-unit>
</persistence>
```

JPA Application 1/2

```
public class Application {
    private static EntityManagerFactory entityManagerFactory;

    /* Reads persistence.xml and looks for specified unit name   */
    protected static void setUp() throws Exception {
        entityManagerFactory =
                Persistence.createEntityManagerFactory("cs544.hibernate.intro.jpa");
    }

    protected static void tearDown() throws Exception {
        entityManagerFactory.close();
    }
```

JPA Application Continued

```
public static void main(String[] args) throws Exception {
        setUp();

        EntityManager em = entityManagerFactory.createEntityManager();
        em.getTransaction().begin();

        // Create new instance of Employee and set values in it
        Employee employee = new Employee();
        employee.setFirstname("Frank");
        employee.setLastname("Miller");
        // save the employee
        em.persist(employee);

        em.getTransaction().commit();
        em.close();

        em = entityManagerFactory.createEntityManager();
        em.getTransaction().begin();

        // retieve all employees
        List<Employee> employeeList = em.createQuery("from Employee").getResultList();
        for (Employee emp : employeeList) {
            System.out.println("firstname= " + emp.getFirstname()
                    + ", lastname= " + emp.getLastname());
        }
        em.getTransaction().commit();

        // Close the SessionFactory (best practice)
        tearDown();
    }
}
```

Output:

```
firstname= Frank, lastname= Miller
```

Active Learning

- In which ways do the OO model and the Relational model conflict?

- Why would it be good to use the show_sql hibernate configuration?

Hibernate Summary

- We talked about the object / relational mismatch and the various Java persistence possibilities
- Of the various Java Persistence possibilities ORM mapping is the most OO friendly
- We showed a small, although complete Hibernate application example with both XML and JPA mapping.
- We also gave some Hibernate configuration options that are useful for development

Main Point

- Although different Java Persistence possibilities exist, Object Relational Mapping is the most OO friendly, providing the greatest unity within diversity.
- *Science of Consciousness*: Practicing TM allows us to more clearly see the unity within diversity and thereby live an easier life, in greater harmony with our surroundings.