Module 00: Course Introduction

CS544: Enterprise Architecture

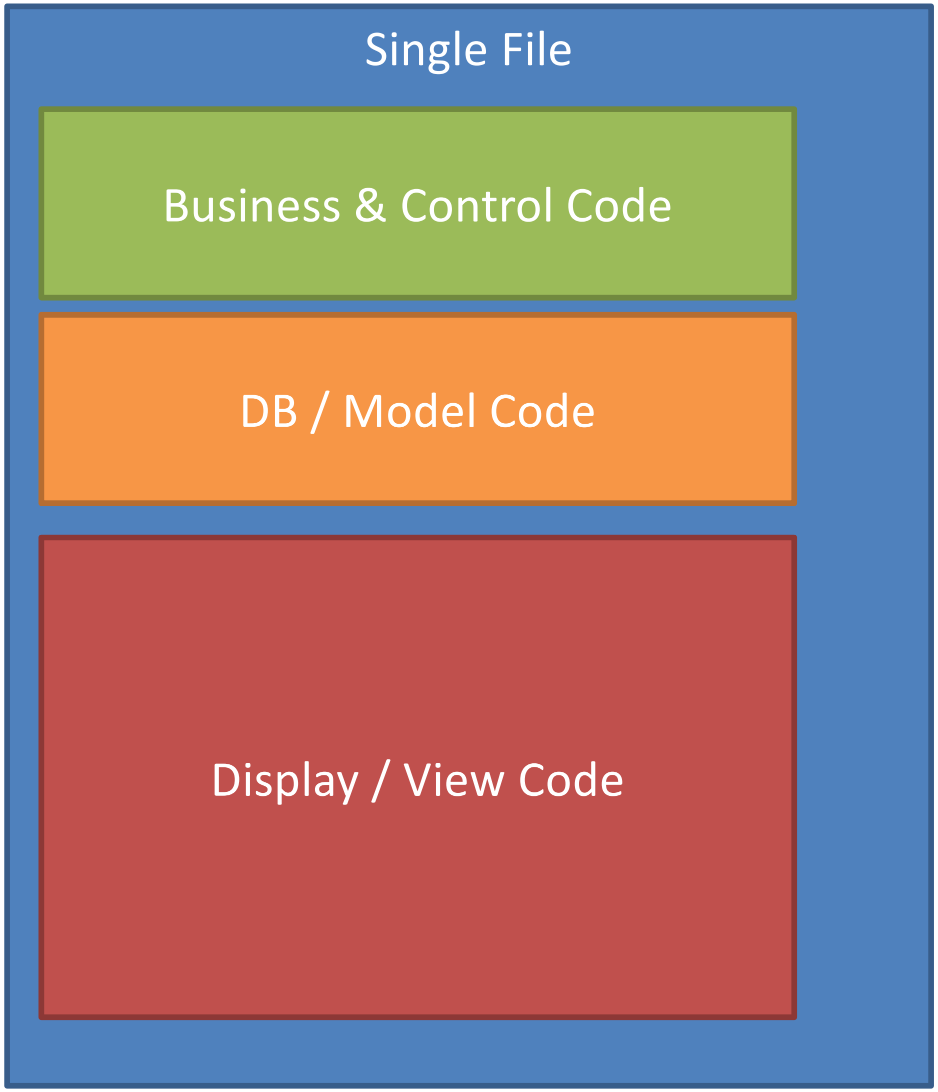
Wholeness

- 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



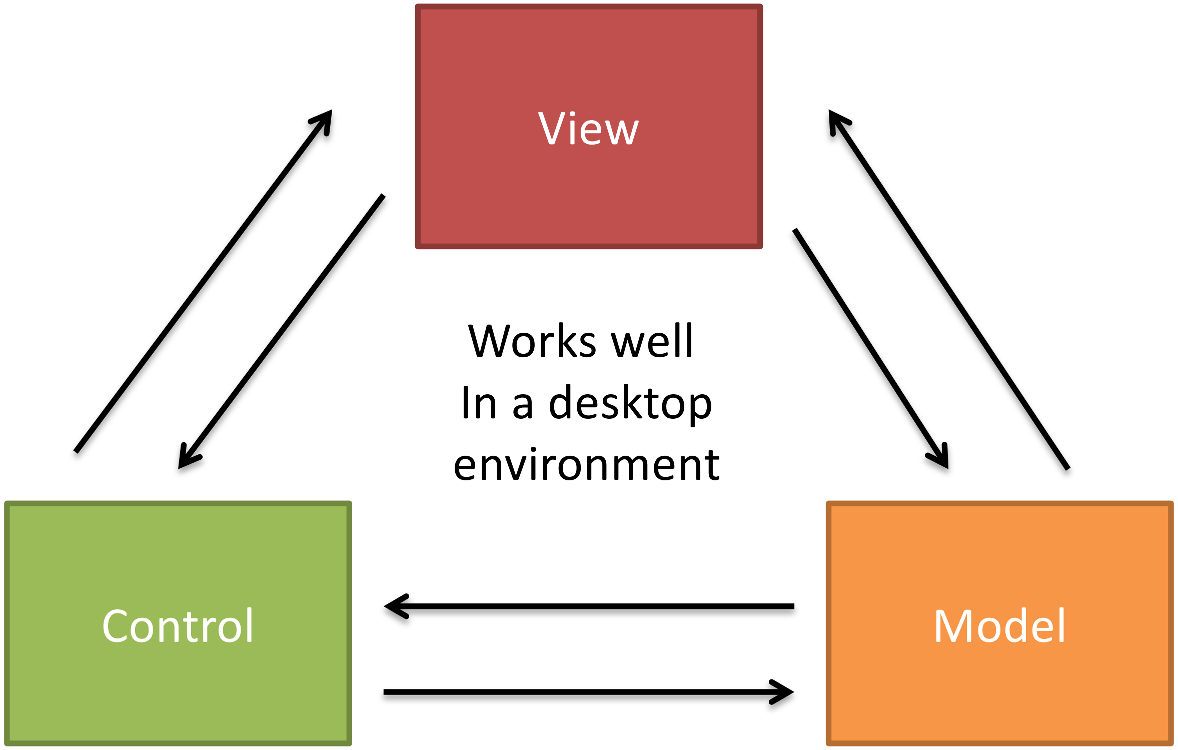
Single File

Business & Control Code

DB / Model Code

Display / View Code

Classic MVC (Model 2)



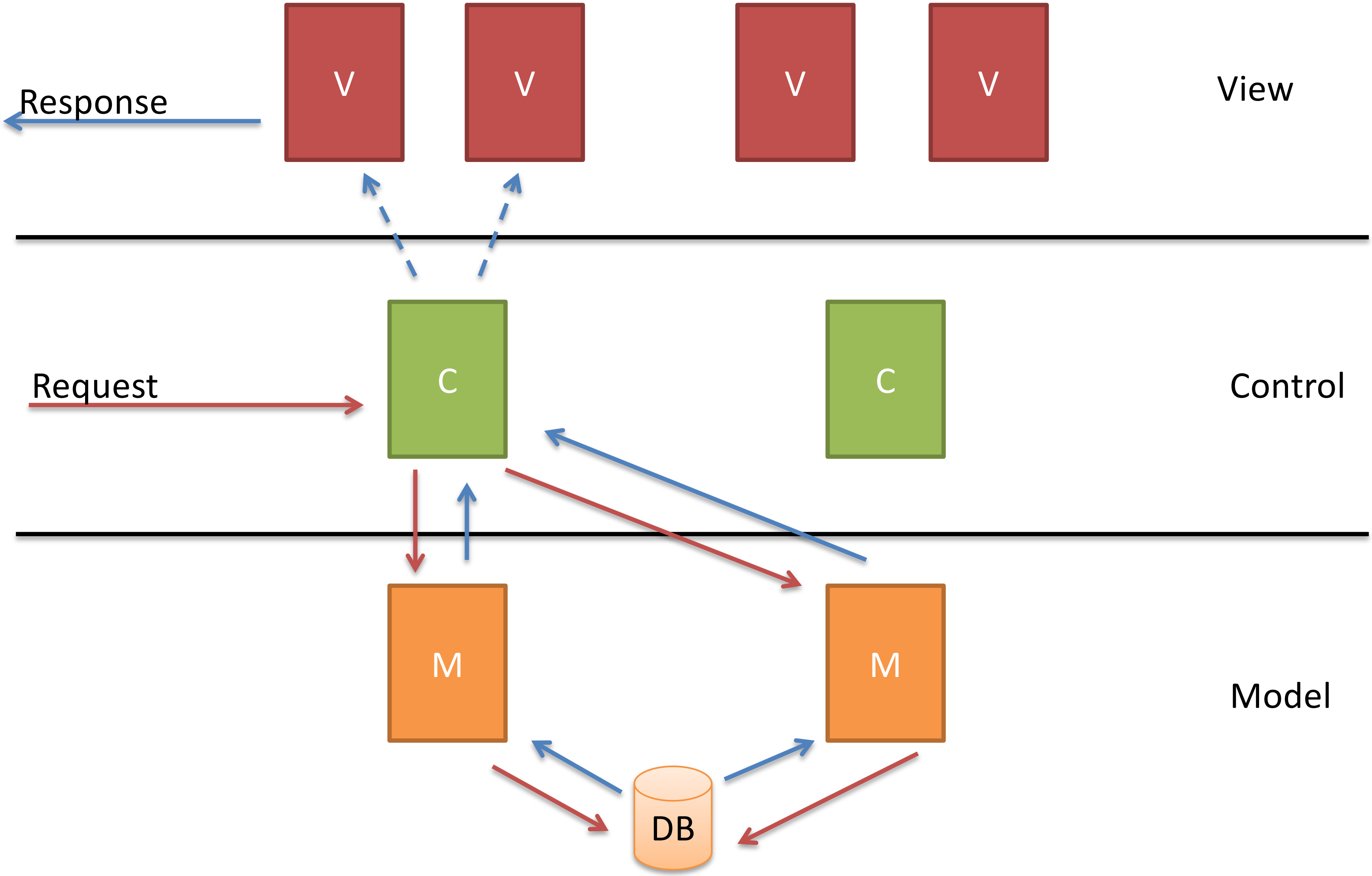
Works well
In a desktop
environment

Model 1 vs Model 2 Architecture

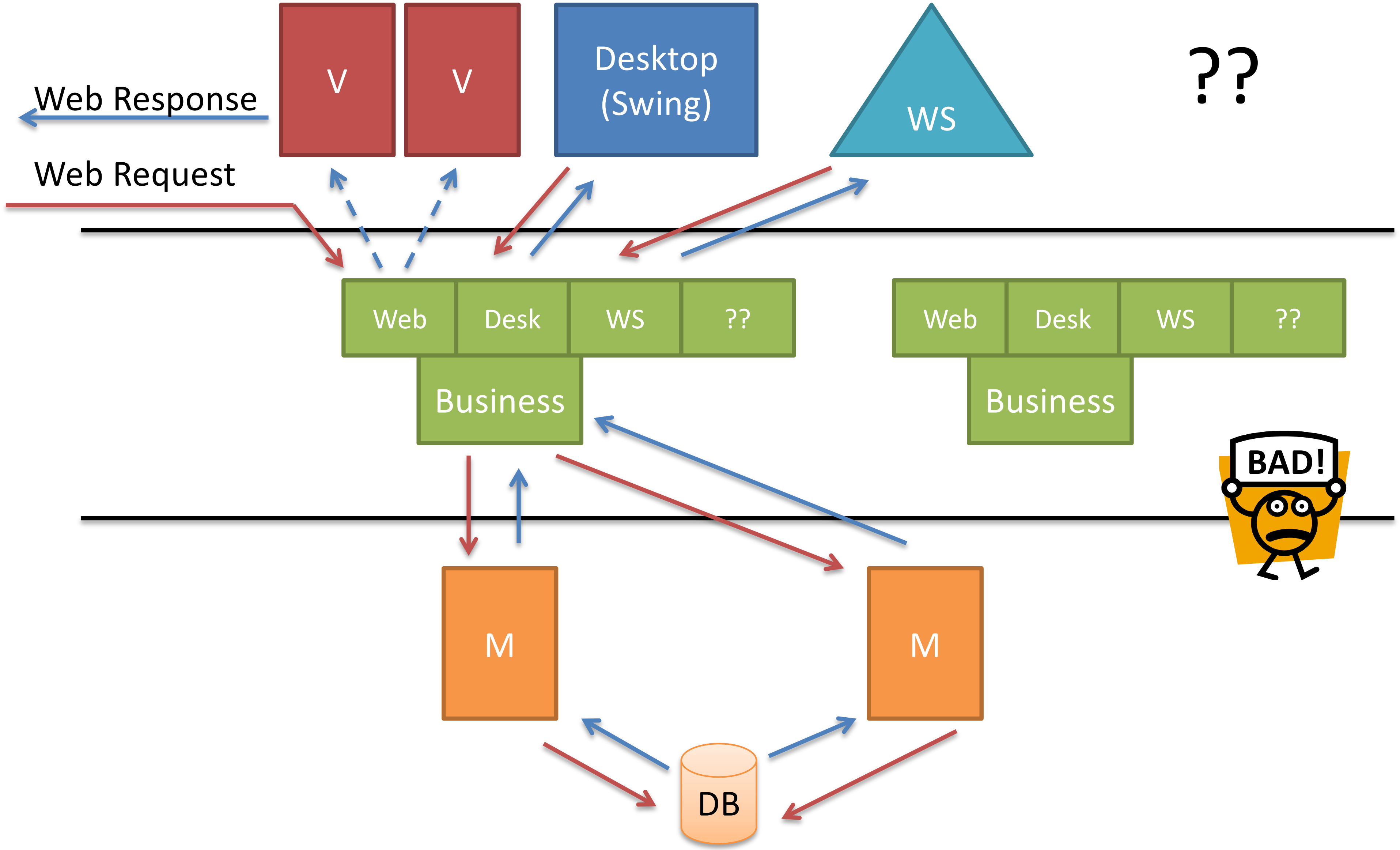
Read more about model 1 vs. model 2 here:

http://download.oracle.com/otn_hosted_doc/jdeveloper/1012/developing_mvc_applications/adf_aboutmvc2.html

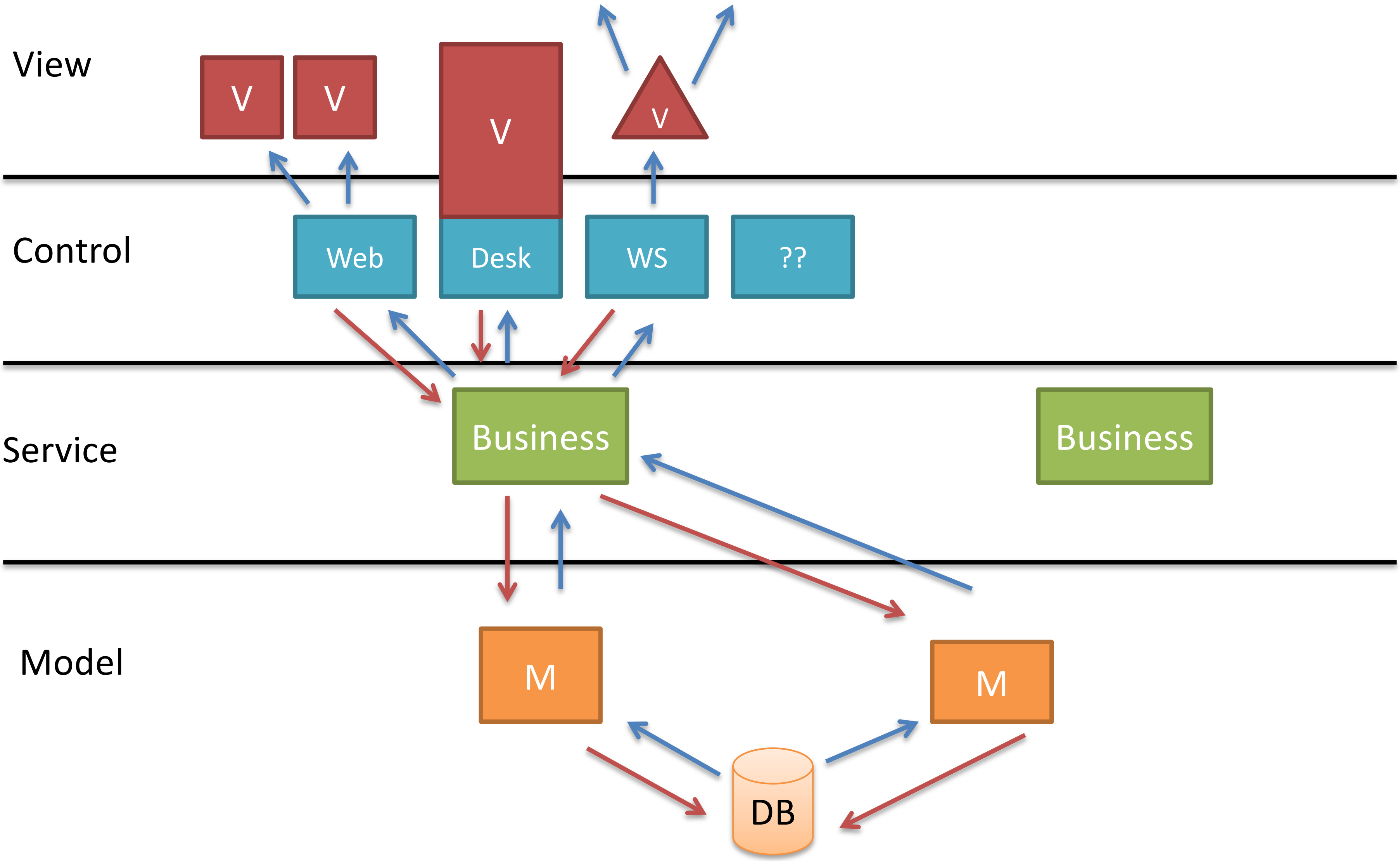
Three Tier / Web MVC



Multiple Types of Clients



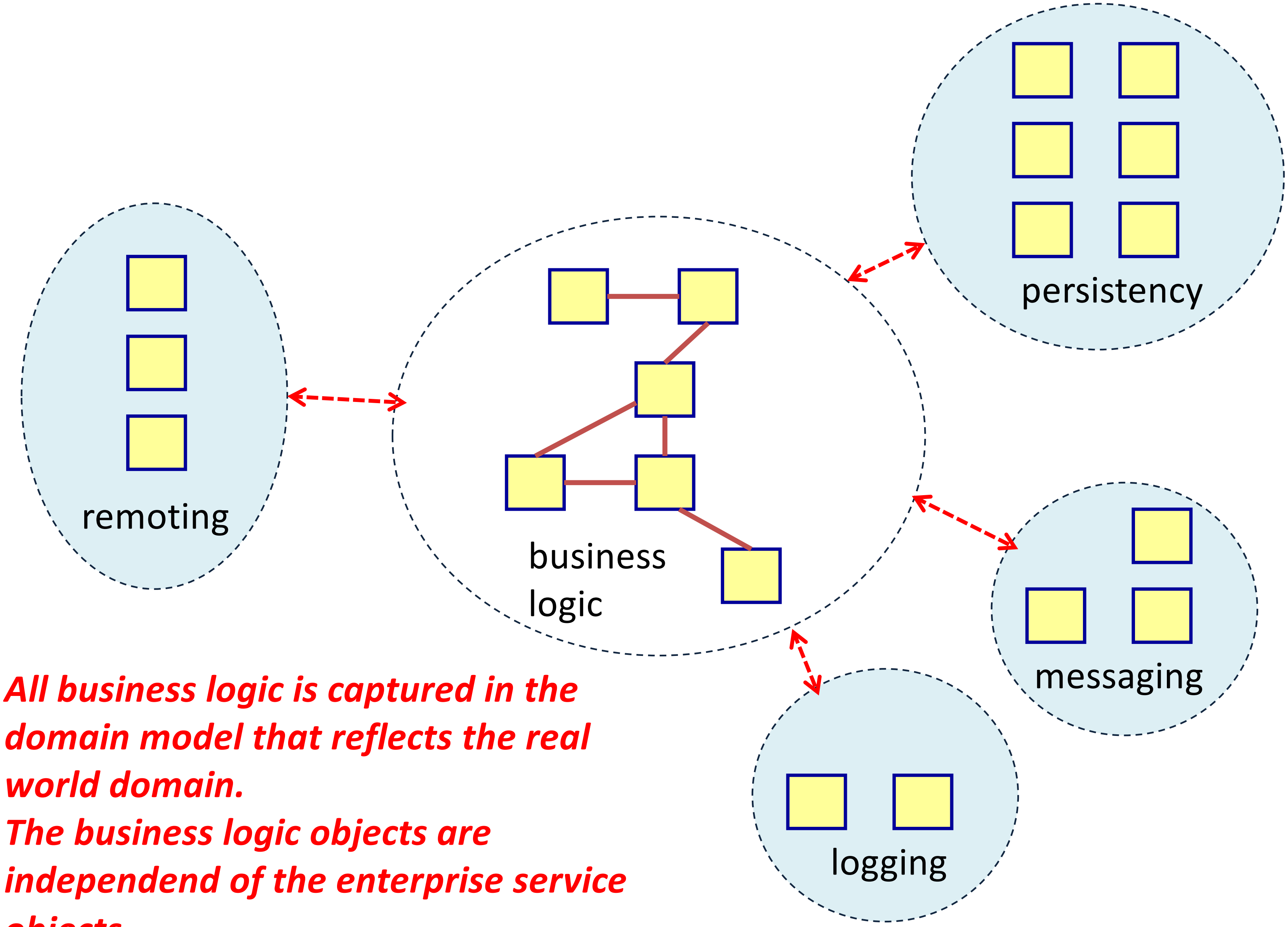
Service Oriented Architecture



Course Introduction:

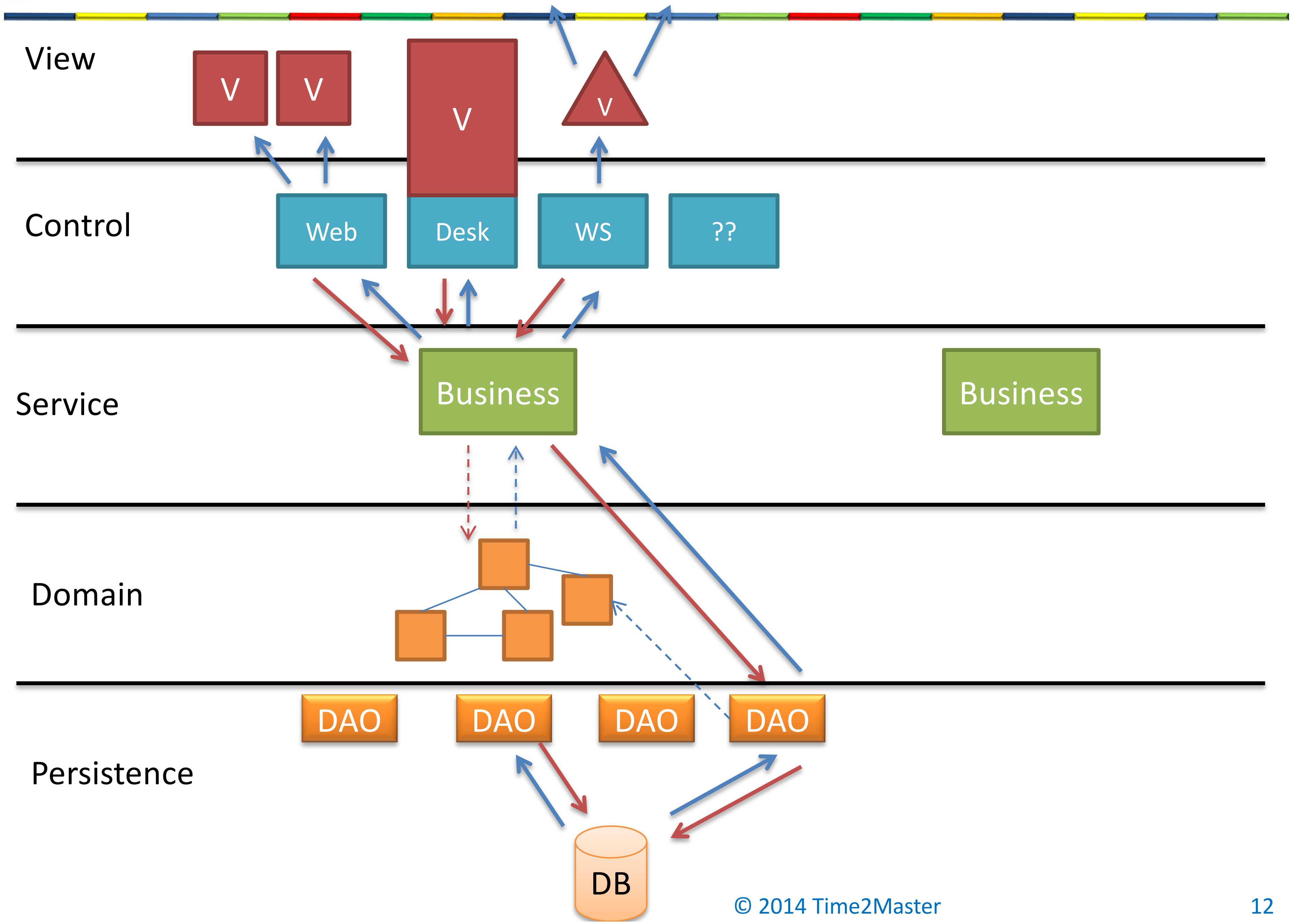
PRINCIPLES

Domain-Driven Design (DDD)

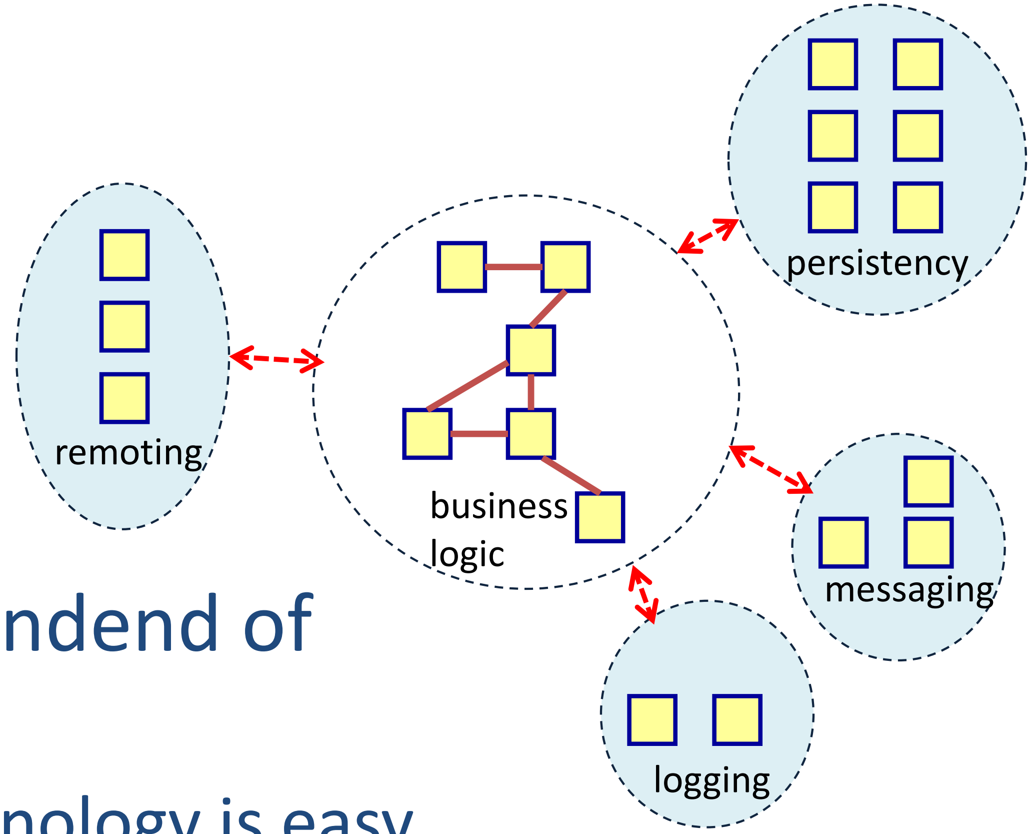


1. ***All business logic is captured in the domain model that reflects the real world domain.***
2. ***The business logic objects are independend of the enterprise service objects***

Service Oriented Architecture



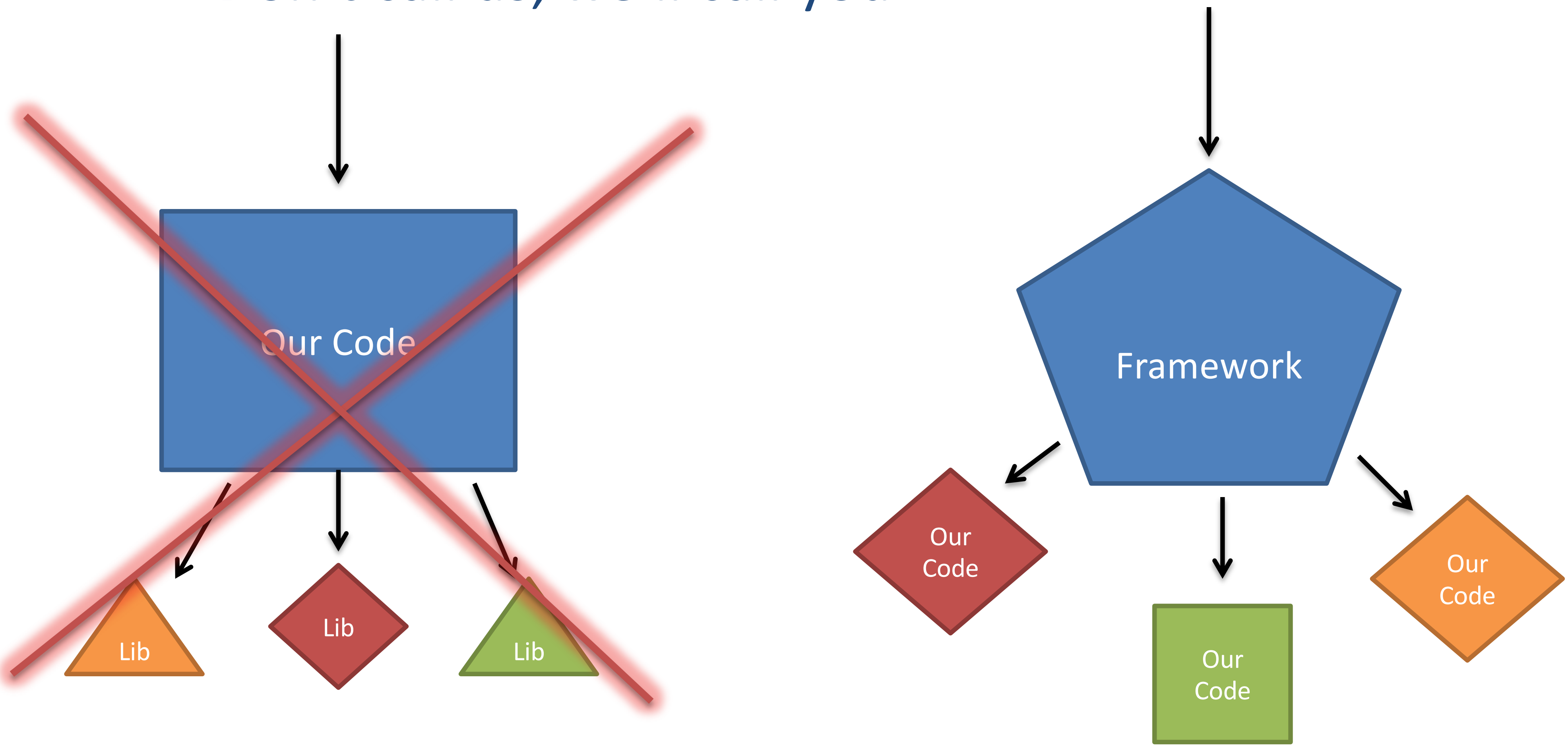
Advantages of DDD



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

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

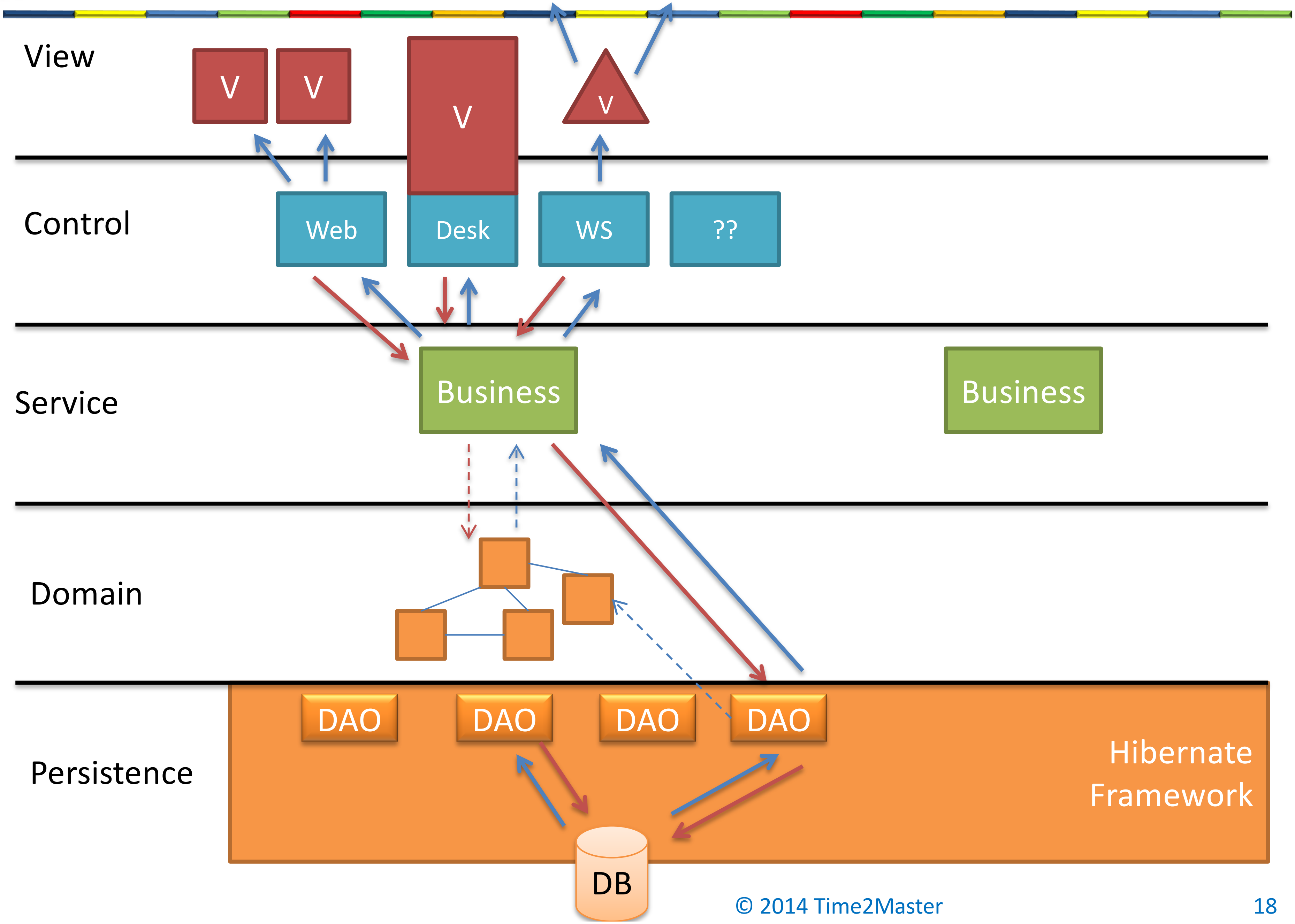
Separation of Concerns

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

Course Introduction:

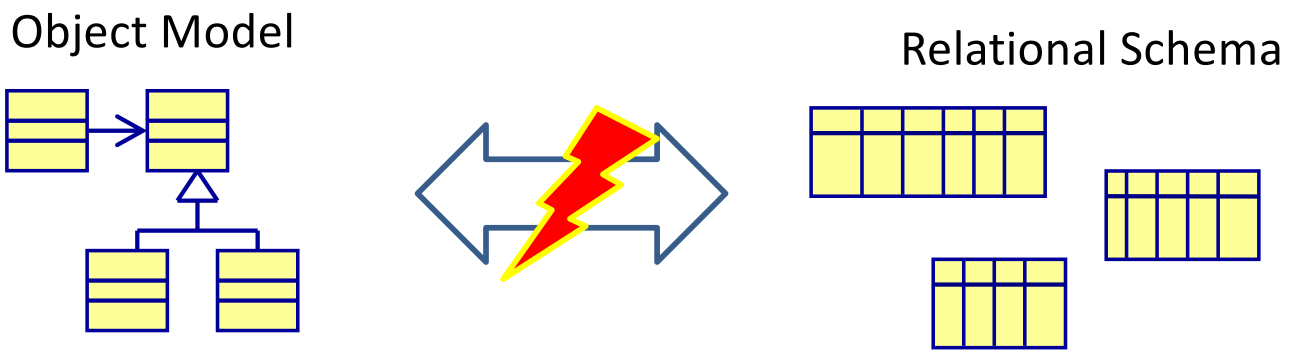
HIBERNATE

Framework for the persistence layer



Object-Relational Mismatch

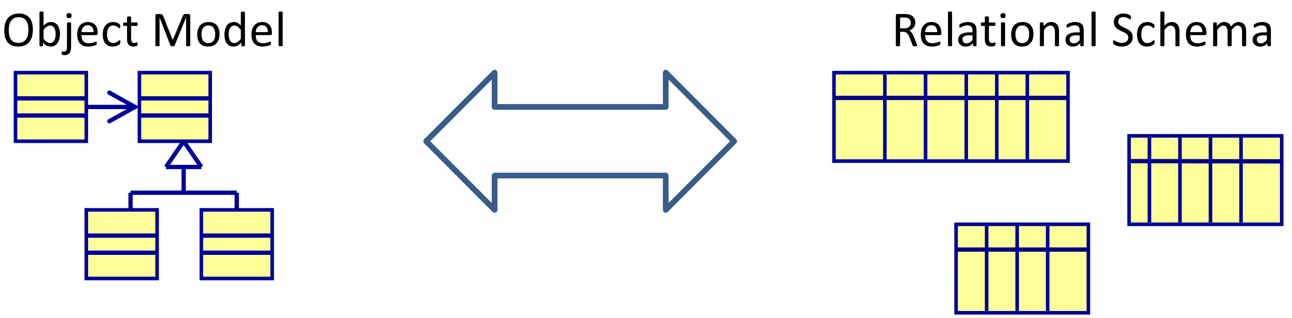
Object Oriented	Relational Database
Objects are instantiations of classes and have identity (object1 == object2)	In the relational model the table name and primary key are used to identity a row in a table
Objects have associations (one-to-one, many-to-one, …)	Relational model has foreign keys and link 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

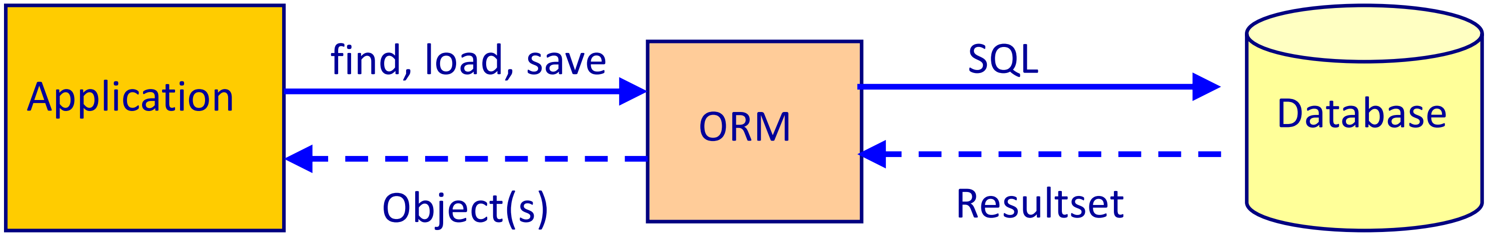


Object Model

Relational Schema

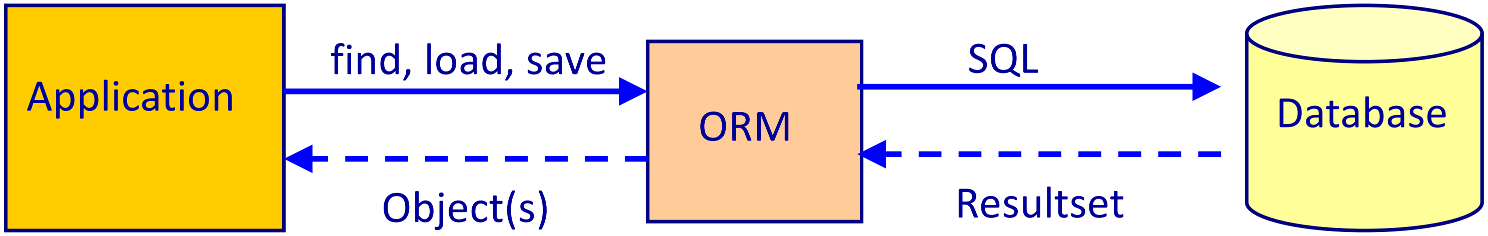
Object Relational Mapping (ORM)

- 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



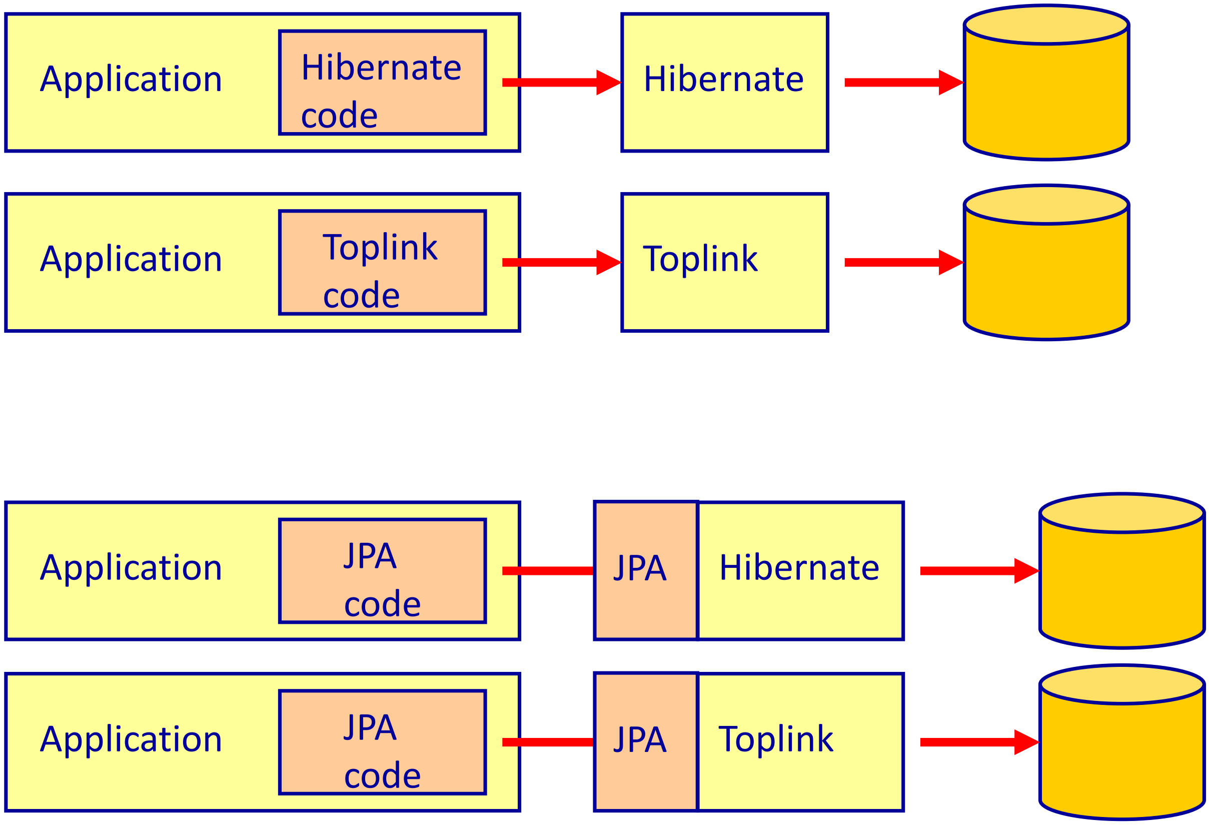
Advantages of ORM

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



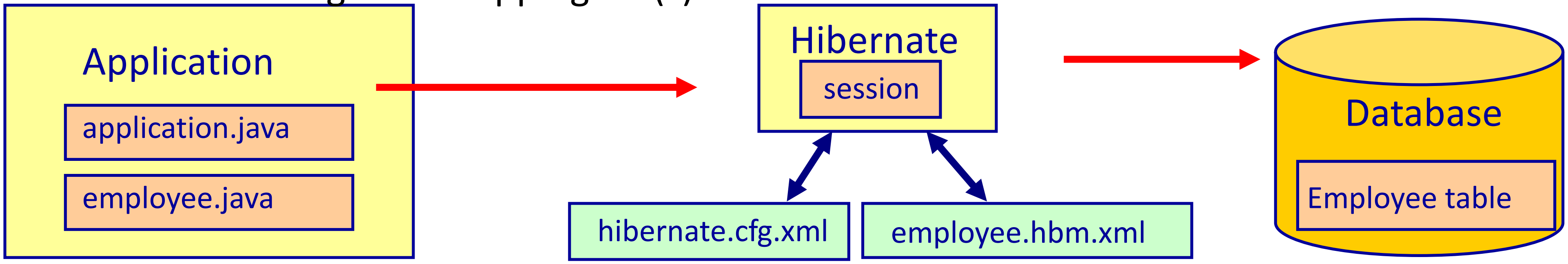
The Java Persistence API (JPA)

- JPA is a Java standard for ORM persistency



3 ways to use Hibernate

1. Hibernate using XML mapping file(s)



Application

application.java

employee.java

Hibernate

session

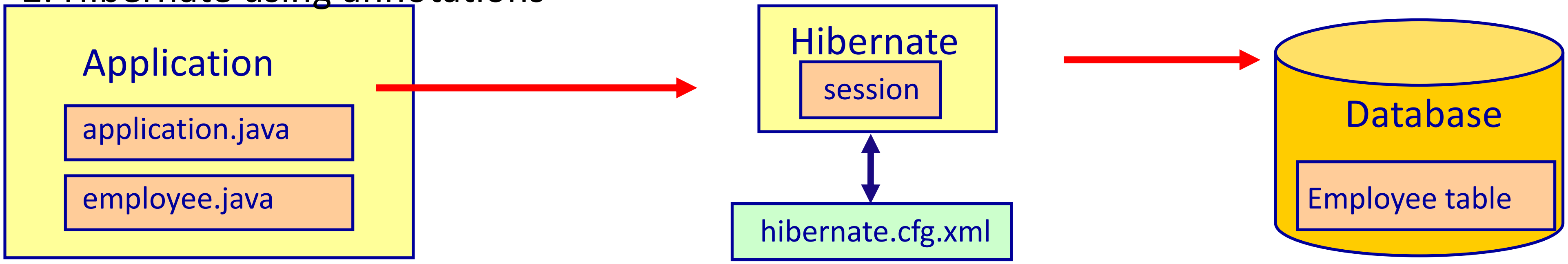
hibernate.cfg.xml

employee.hbm.xml

Database

Employee table

2. Hibernate using annotations



Application

application.java

employee.java

Hibernate

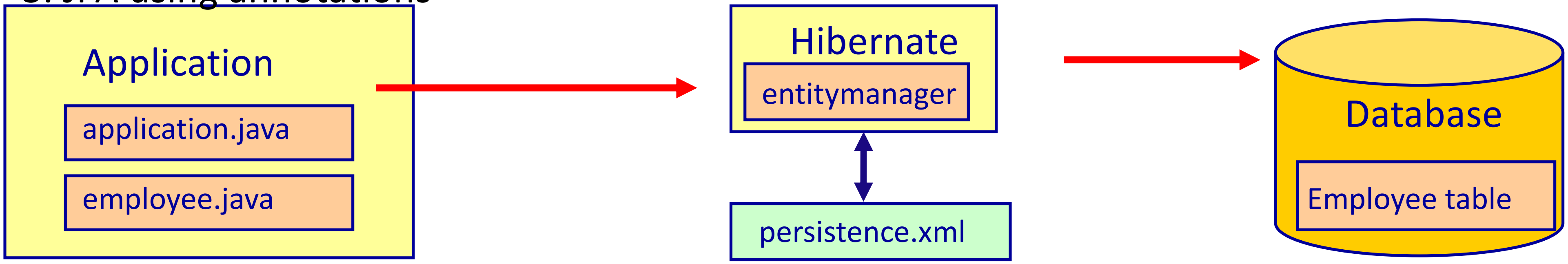
session

hibernate.cfg.xml

Database

Employee table

3. JPA using annotations



Application

application.java

employee.java

Hibernate

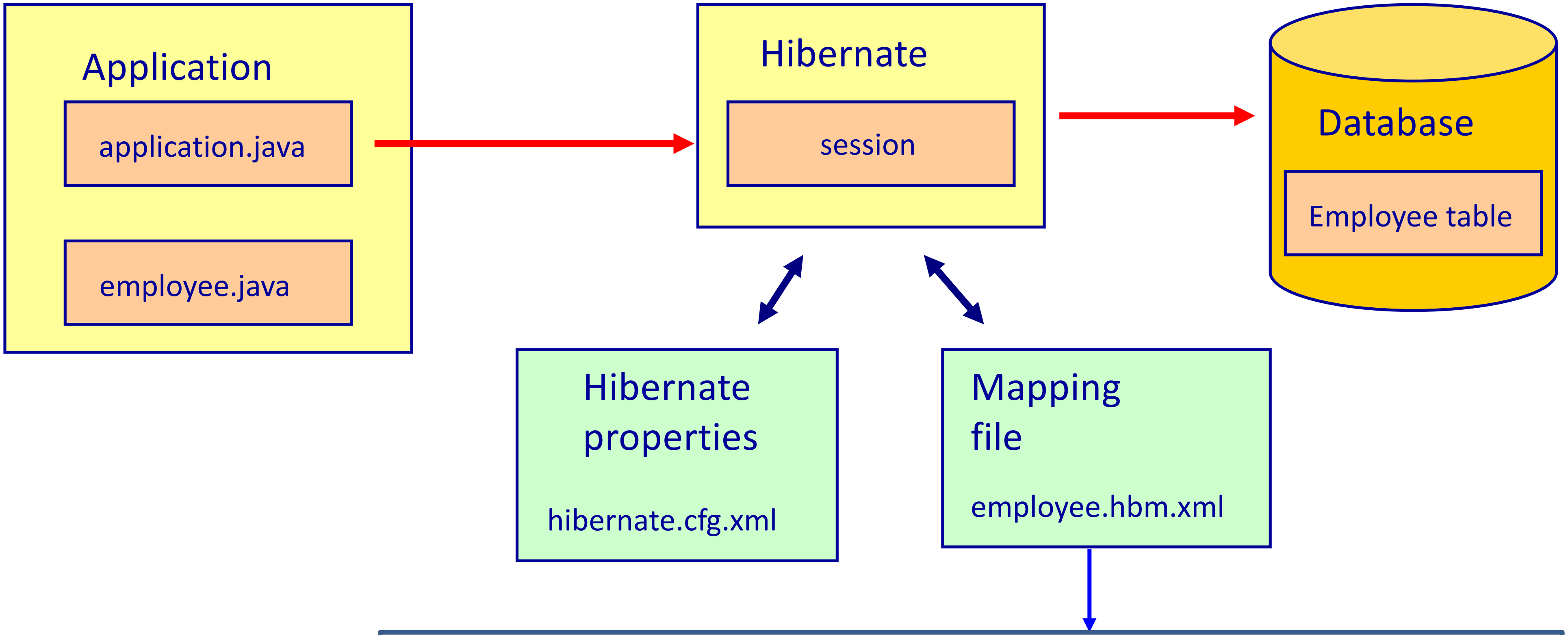
entitymanager

persistence.xml

Database

Employee table

1. Hibernate using XML mapping file(s)



Application

application.java

employee.java

Hibernate

session

Database

Employee table

Hibernate properties

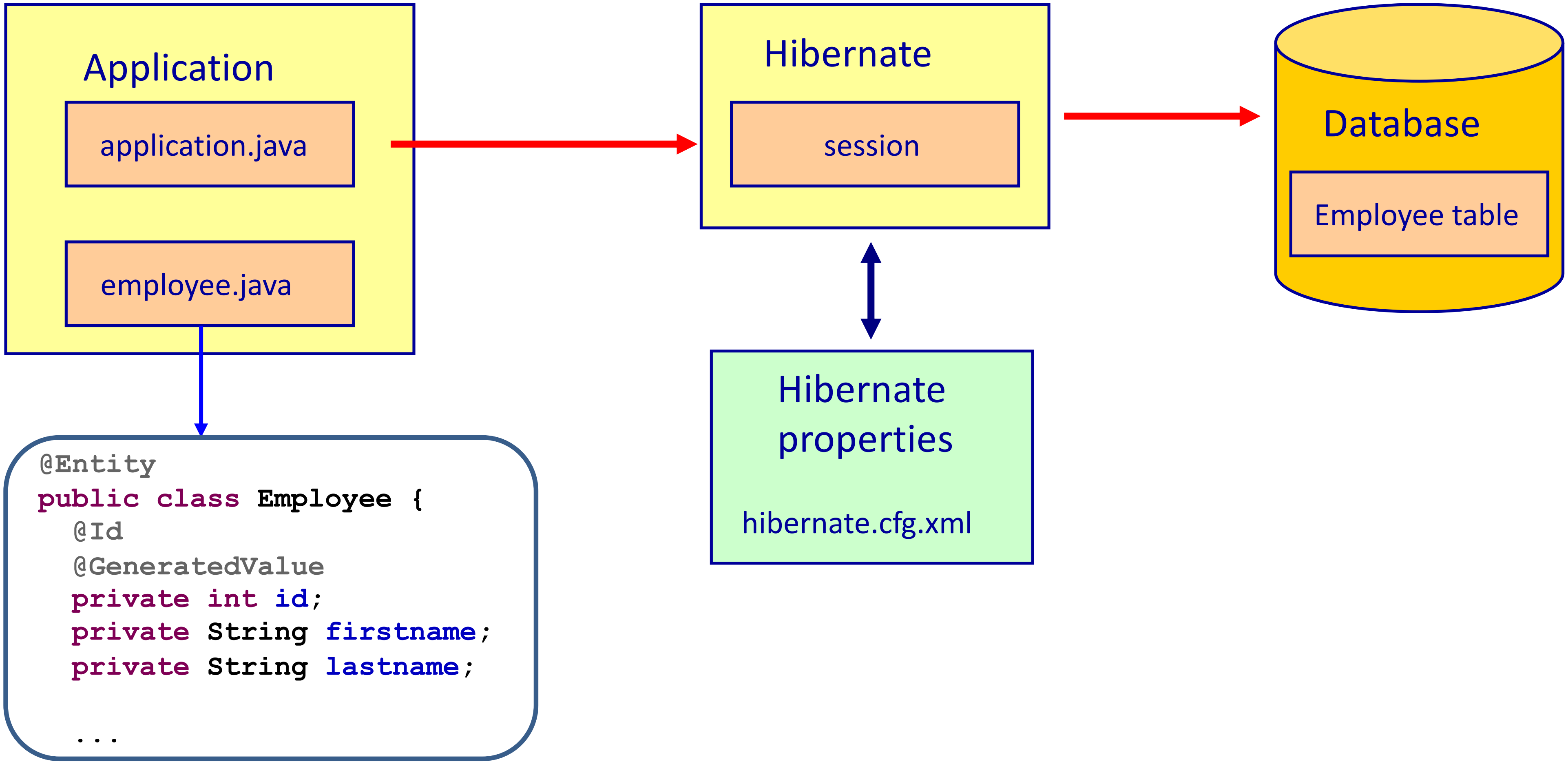
hibernate.cfg.xml

Mapping file

employee.hbm.xml

```xml
<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>
```

2. Hibernate using Annotations



Application

application.java

employee.java

Hibernate

session

Database

Employee table

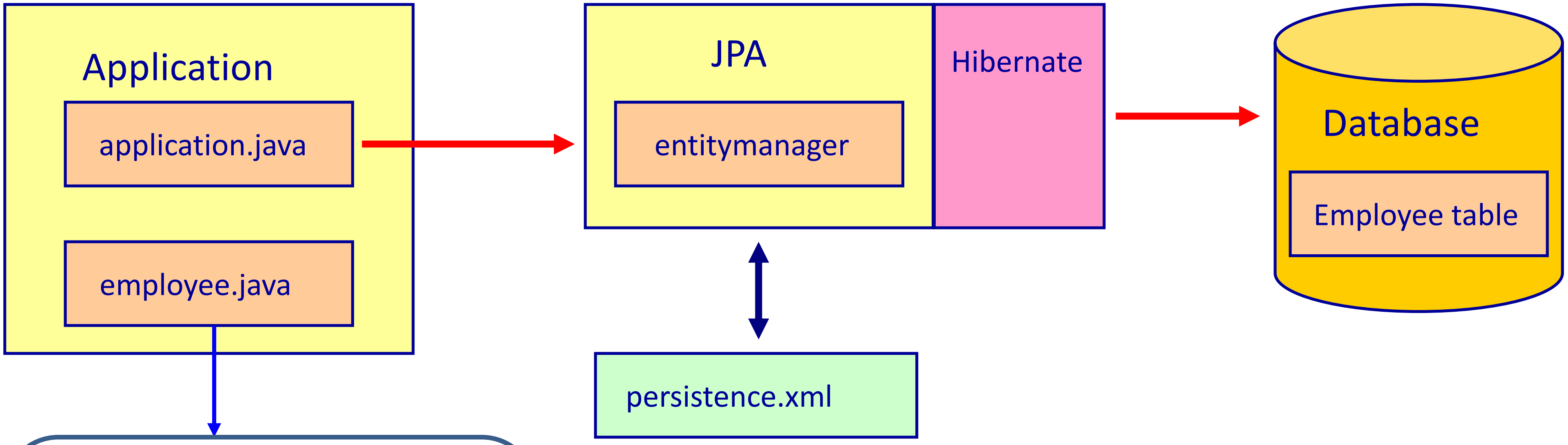
Hibernate properties

hibernate.cfg.xml

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

  ...
```

3. JPA using Annotations



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

  ...
```

A simple Hibernate Example

```java
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
<?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
<?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>
        <!-- HSQL DB running on localhost -->
        <property name="connection.url">jdbc:hsqldb:hsql://localhost/employeedb</property>
        <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property name="connection.username">sa</property>
        <property name="connection.password"></property>
        <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
        <!-- Mapping files -->
        <mapping resource="intro/xml/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Using Annotations

```
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
<?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>
        <!-- HSQL DB running on localhost -->
        <property name="connection.url">jdbc:hsqldb:hsql://localhost/employeedb</property>
        <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property name="connection.username">sa</property>
        <property name="connection.password"></property>
        <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
        <!-- Mapping files -->
        <mapping class="intro.annotations.Employee"/>
    </session-factory>
</hibernate-configuration>
```

Hibernate Application Example

```
public class Application {
private static SessionFactory sessionFactory;
  static {
    // This step will read hibernate.cfg.xml and prepare hibernate for use
    Configuration configuration = new Configuration();
    Configuration.configure("intro/annotations/hibernate.cfg.xml");
    ServiceRegistry sr =  new StandardServiceRegistryBuilder().applySettings(
        configuration.getProperties()).build();
    sessionFactory = configuration.buildSessionFactory(sr);
  }
  public static void main(String[] args) {
    // Hibernate placeholders
    Session session = null;
    Transaction tx = null;
    try {
      session = sessionFactory.openSession();
      tx = 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);

      tx.commit();
    } catch (HibernateException e) {
      tx.rollback();
      e.printStackTrace();
    } finally {
      if (session != null)
        session.close();
    }
```

Hibernate Application Continued

```
    try {
      session = sessionFactory.openSession();
      tx = 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());
      }
      tx.commit();

    } catch (HibernateException e) {
      tx.rollback();
      e.printStackTrace();
    } finally {
      if (session != null)
        session.close();
    }

    // Close the SessionFactory (not mandatory)
    sessionFactory.close();
  }
}
```

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>
        <!-- HSQL DB running on localhost -->
        <property name="connection.url">jdbc:hsqldb:hsql://localhost/employeedb</property>
        <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property name="connection.username">sa</property>
        <property name="connection.password"></property>
        <property name="dialect">org.hibernate.dialect.HSQLDialect</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>
        <!-- HSQL DB running on localhost -->
        <property name="connection.url">jdbc:hsqldb:hsql://localhost/employeedb</property>
        <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property name="connection.username">sa</property>
        <property name="connection.password"></property>
        <property name="dialect">org.hibernate.dialect.HSQLDialect</property>

        <property name="hbm2ddl.auto">create</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

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?

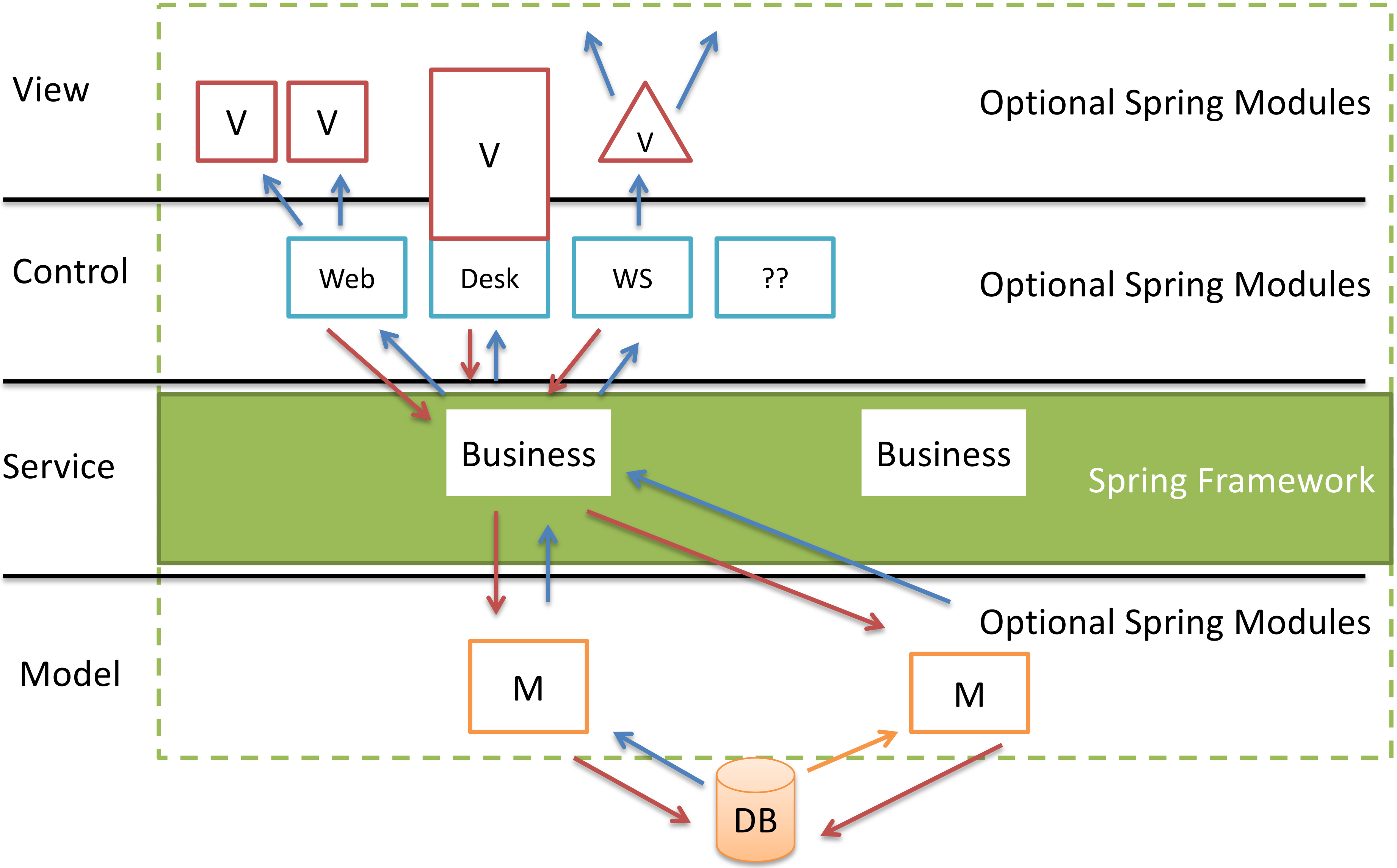
Hibernate Summary

- 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

Course Introduction:

SPRING

Framework for the Service Layer



© 2014 Time2Master

History of Spring

- Started as alternative to EJB 2.1
  - Rod Johnson book: Expert One-on-one J2EE Design And Development

- EJB 3 Is like Spring / Hibernate
  - Spring moved ahead / not tied down by legacy
  - Spring community expanded beyond EJB

- Spring becomes another JEE implementation?

Aim of the Spring framework

- Make enterprise Java application development as easy as possible, following good programming practices
  - POJO-based programming
  - Separation of concerns
  - Flexibility

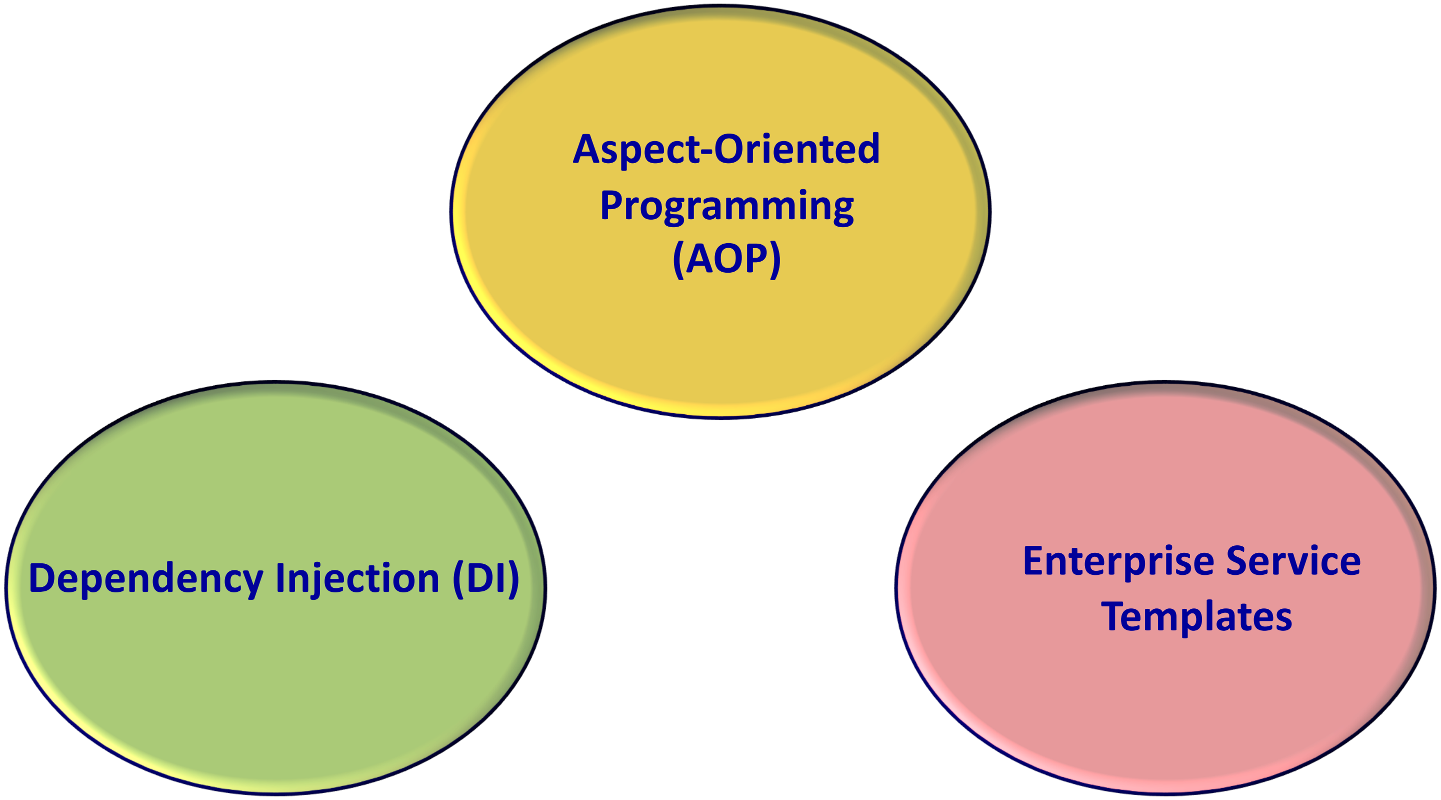
41

POJO based programming

- All code is written in java objects
  - No EJB’s
- Promotes Object-Oriented principles
- Simple to understand
- Simple to refactor
- Simple to unit test

42

Core of Spring



Aspect-Oriented Programming (AOP)

Dependency Injection (DI)

Enterprise Service Templates

43

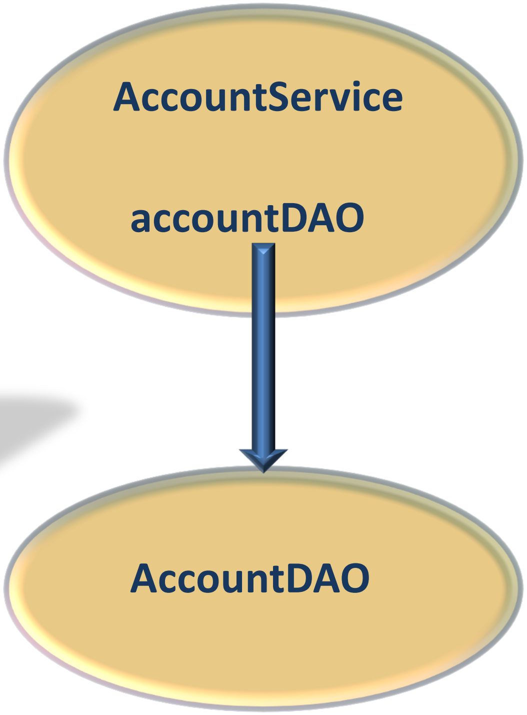
Dependency Injection

- Spring instantiates objects and wires them together

```
public class AccountService {
   private AccountDAO accountDAO;

    public void setAccountDAO(AccountDAO accountDAO) {
        this.accountDAO = accountDAO;
    }

    public Account getAccount(int accountNumber) {
        return accountDAO.loadAccount(accountNumber);
    }
}
```

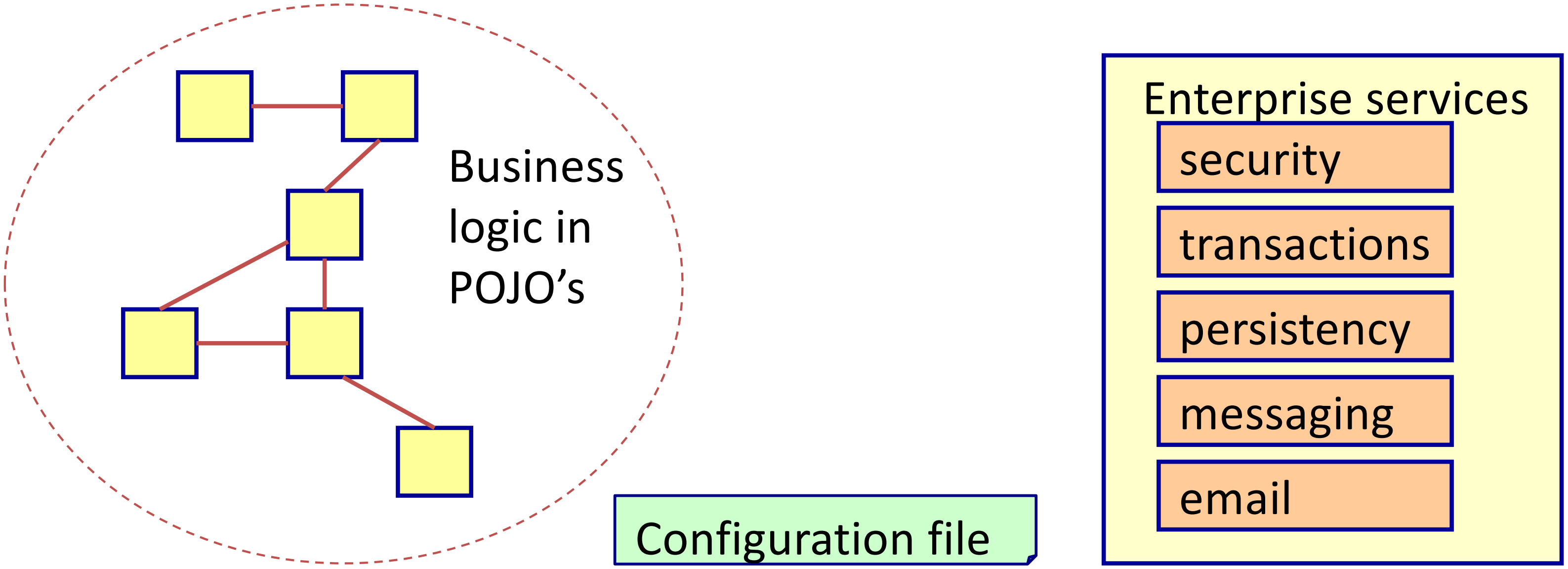


```
 <bean id="accountService" class="bank.AccountService">
   <property name="accountDAO" ref="accountDAO" />
 </bean>
 <bean id="accountDAO" class="bank.dao.AccountDAO" />
```

44

Aspect-Oriented Programming (AOP)

- Separate the crosscutting concerns (plumbing code) from the business logic code
- AOP development
  1. **Write the business logic without worrying about the enterprise services (security, transactions, logging, etc)**
  2. **Write the enterprise services**
  3. **Weave them together**



Business logic in POJO's

Configuration file

Enterprise services

- security
- transactions
- persistency
- messaging
- email

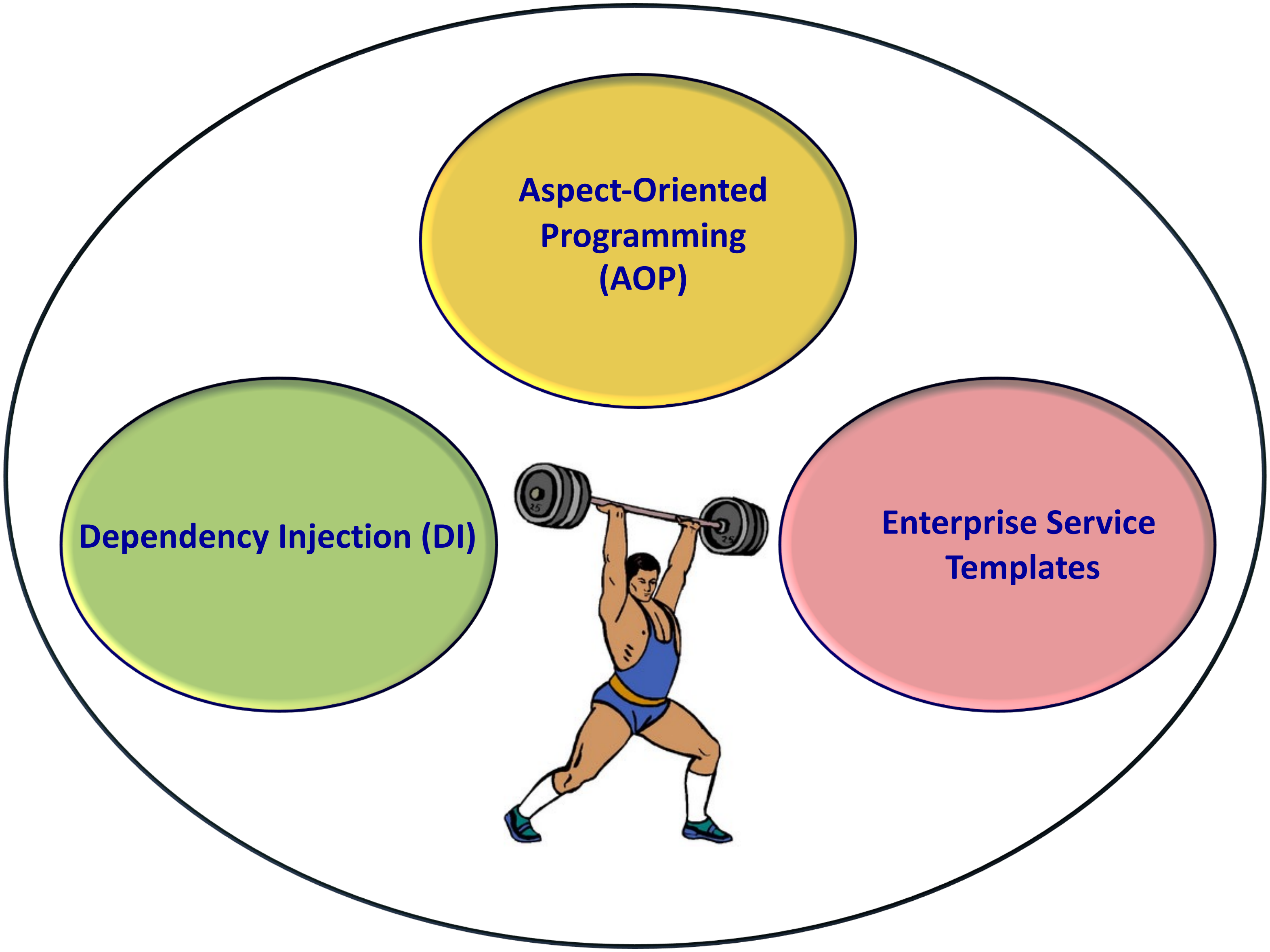
45

Enterprise Service Templates

- Makes programming the different enterprise service API’s simpler.
  - JDBC template
  - JMS template
  - JavaMail template
  - Hibernate template
- Let the programmer focus on what needs to happen instead of complexity of the specific API
  - Resource management
  - Exception handling
  - Try-catch-finally-try-catch blocks

46

The power of Spring



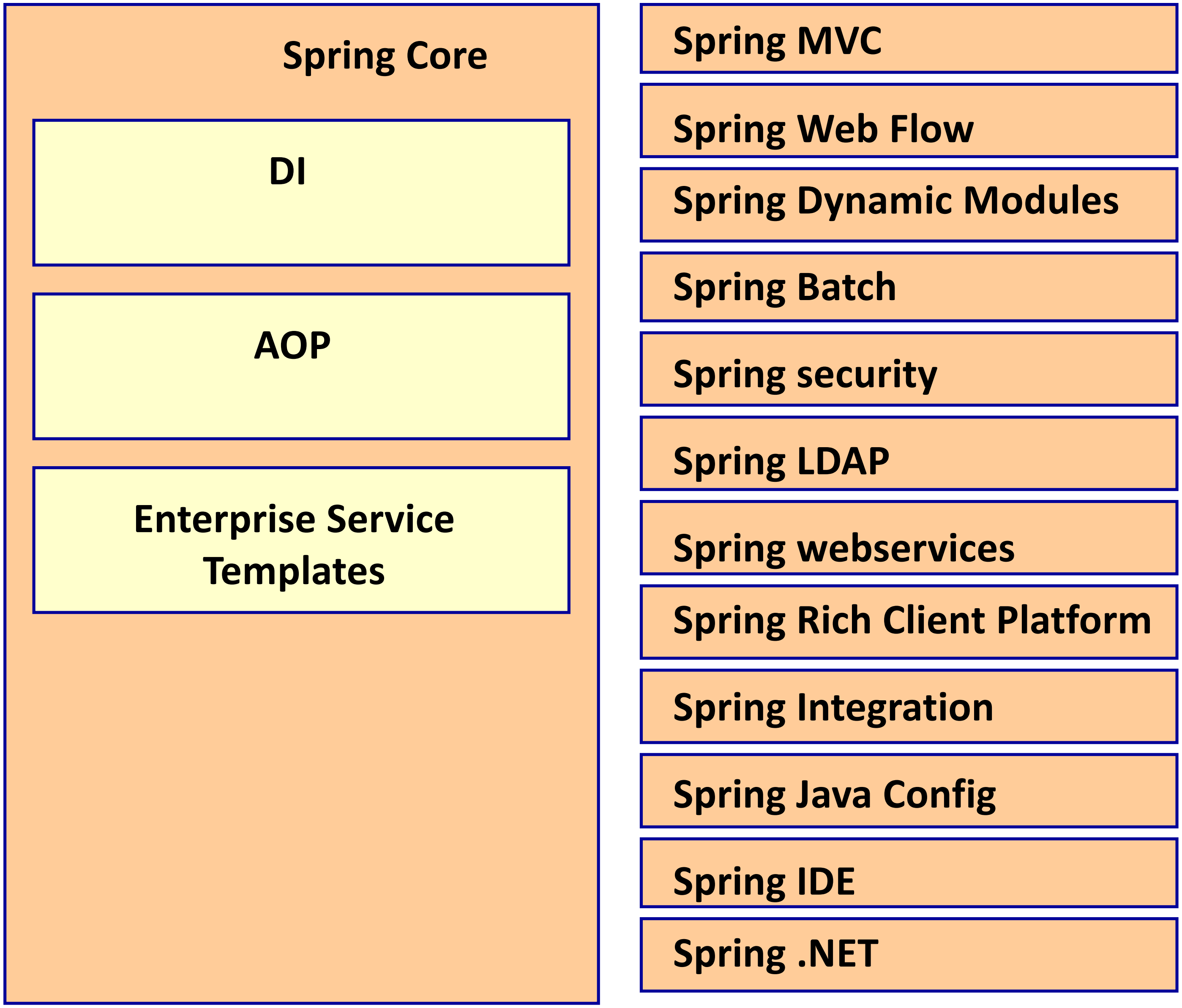
Aspect-Oriented Programming (AOP)

Dependency Injection (DI)

Enterprise Service Templates

47

Spring Portfolio



Spring Core

DI

AOP

Enterprise Service Templates

Spring MVC

Spring Web Flow

Spring Dynamic Modules

Spring Batch

Spring security

Spring LDAP

Spring webservices

Spring Rich Client Platform

Spring Integration

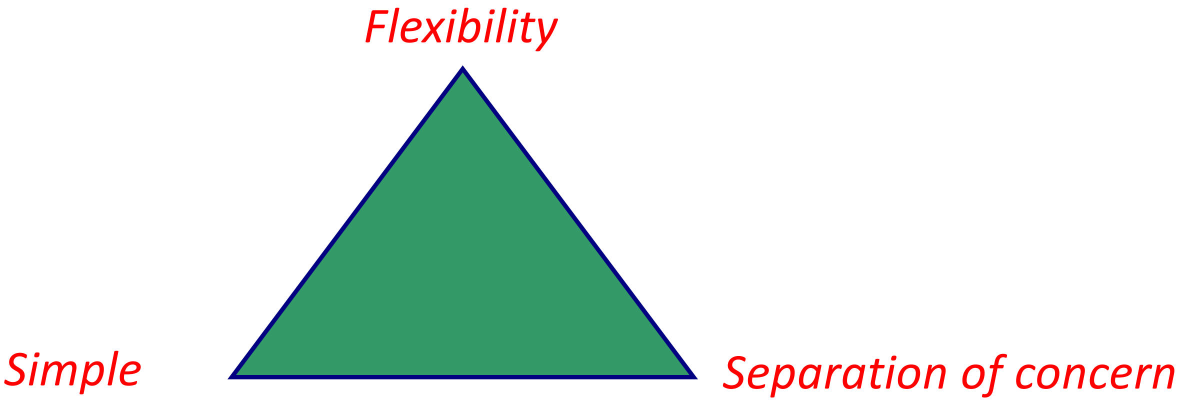
Spring Java Config

Spring IDE

Spring .NET

Advantages of Spring

- Spring makes application development simple
  - POJO based programming
  - Simple coding of enterprise java API’s
- Dependency injection gives flexibility in bean wiring
- AOP separates the business logic from the enterprise service (plumbing) code



Flexibility

Simple

Separation of concern

Disadvantages of Spring

- Spring is another framework to learn
  - But, if you use another technique or another framework you have the same problem
- Spring is not a Java EE standard
  - But, the value of a standard is not that important anymore
    - Spring is much more powerful than EJB 3.0
    - Spring has become an enterprise Java standard
- The XML file of Spring can become very complex
  - But: …
    - Spring also supports annotations
    - The Spring XML file is not that complex once you get used to it
    - The Spring XML file can be separated into multiple XML files
    - Spring also supports Java configuration

Active Learning

- What are the 3 main components of Spring?

- Why would you want to use Spring over standardized JavaEE EJBs?

Spring Summary

- Spring makes developing enterprise Java applications simpler.
- Spring started as a replacement for EJB's, but has evolved to a framework that supports all different application layers
- The core of Spring consists of DI, AOP and enterprise service templates
- There are many additional projects in the Spring eco-system that easily integrate with Spring in a modular fashion.