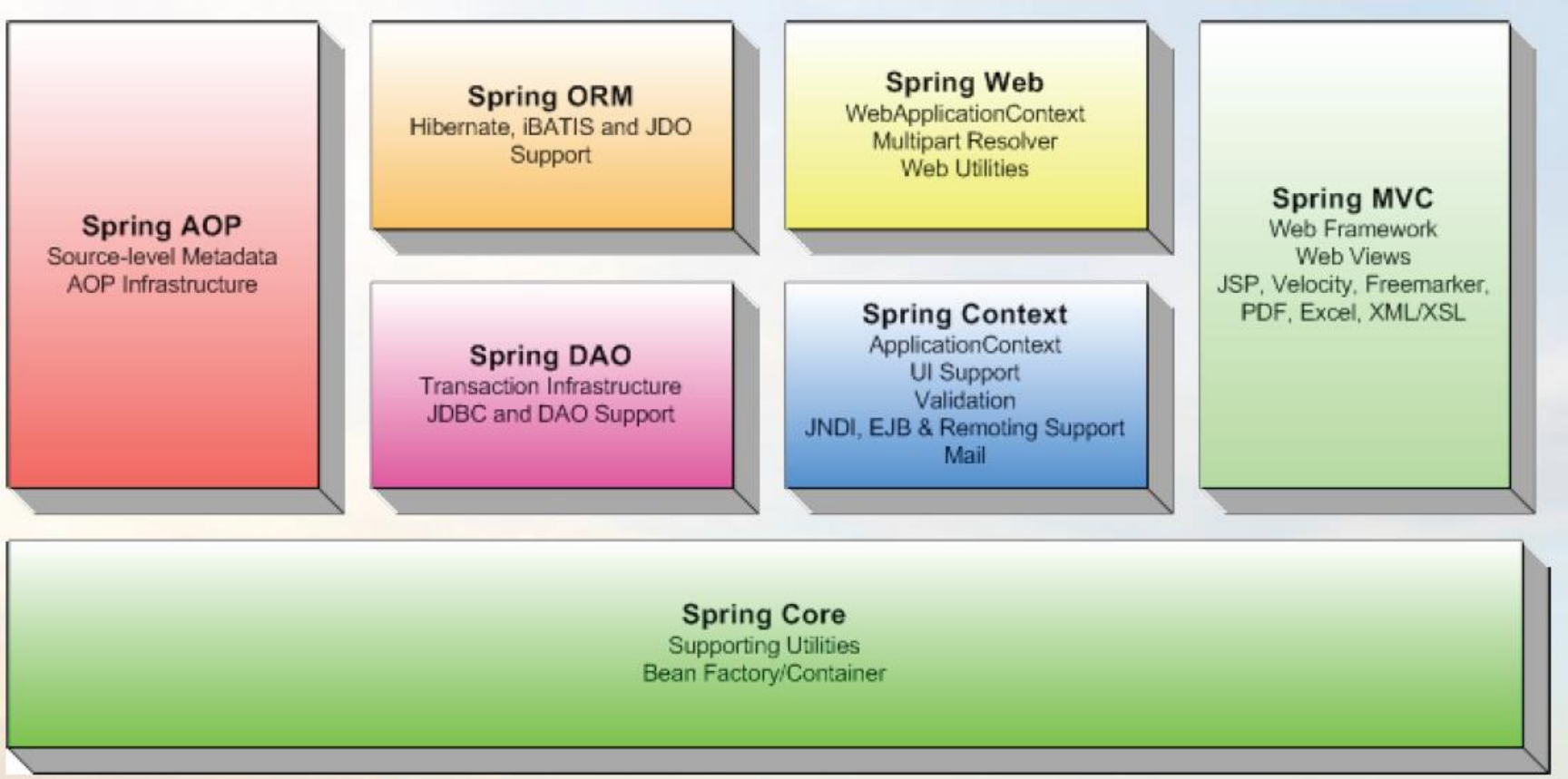
Lesson 4 Spring & Spring MVC Framework

Infinite Diversity Arising from Unity

Spring Architecture

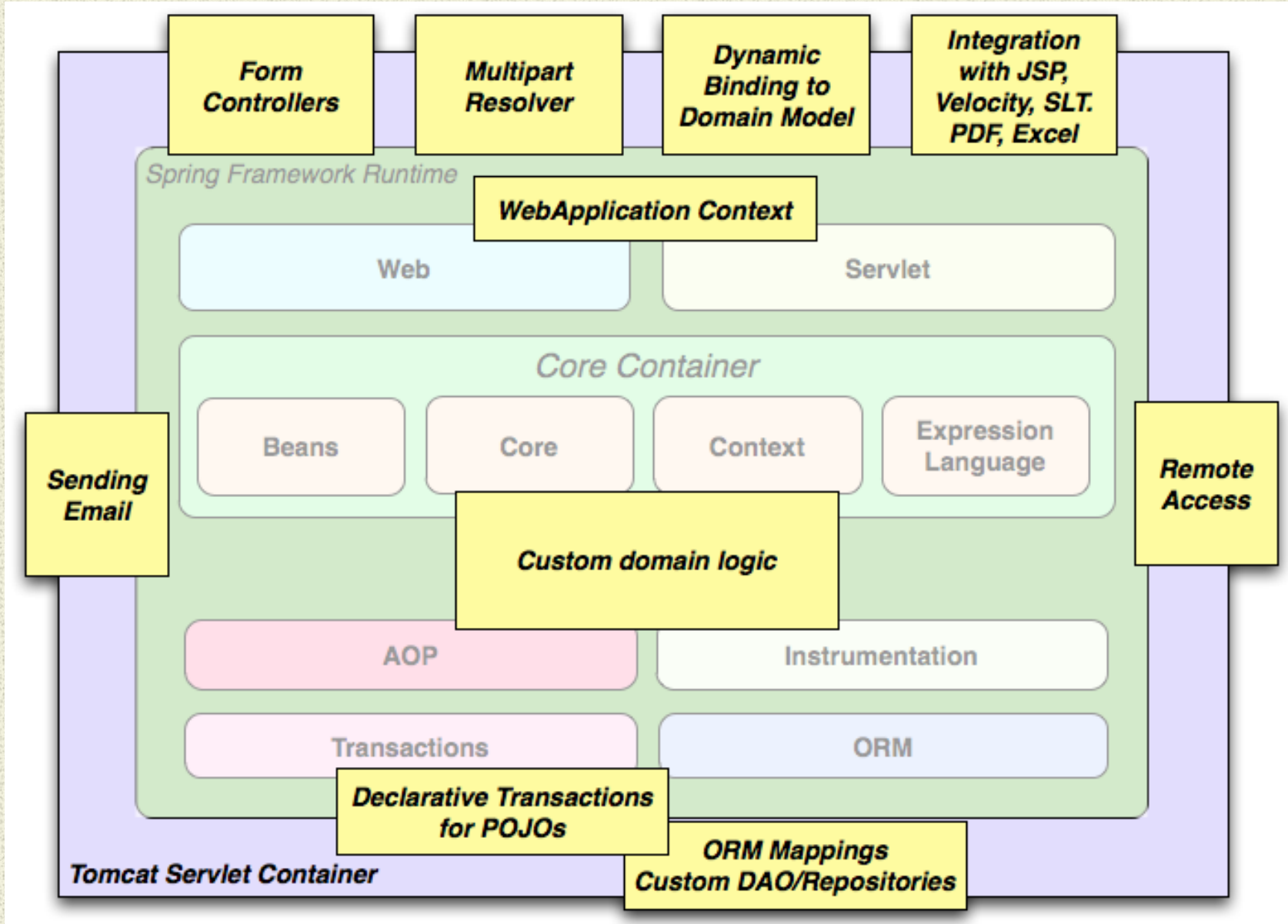


Spring Framework

- Infrastructure support for developing Java applications.
- Configure disparate components into a fully working application ready for use.
- Build applications from “plain old Java objects” (POJOs)
- Non-intrusive - domain logic has little or no dependencies on framework
- Lightweight application model is that of a layered [N-tier] architecture. Spring 3 Tiers:

1. Presentation objects such as Spring MVC controllers are typically configured in a distinct ***presentation context[tier]***
2. Service objects, business-specific objects, etc. exist in a distinct ***business context[tier]***
3. Data access objects exist in a distinct ***persistence context[tier]***

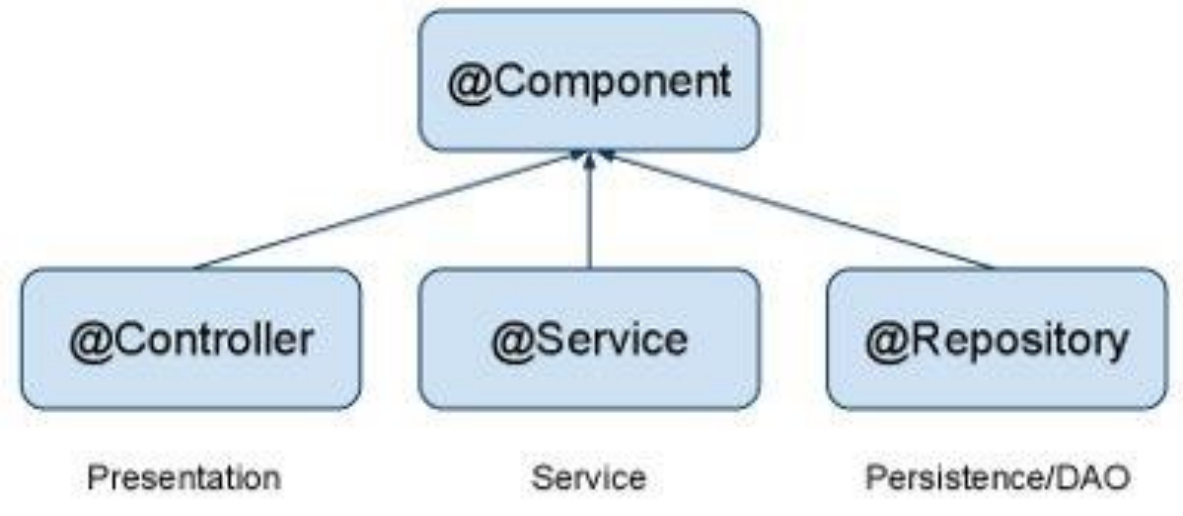
Spring N-tier Architecture



Spring Configuration Metadata

- **XML based**
  - Wire components without touching their source code or recompiling them.
  - CLAIM: Annotated classes are no longer POJOs ****
  - Configuration centralized and easier to control.
- **Annotation [Version 2.5]**
  - Component wiring close to the source
  - Shorter and more concise configuration.
- **JavaConfig [Version 3.0]**
  - Define beans external to your application classes by using Java rather than XML files
- Annotation injection is performed *before* XML injection. XML is the “last word”

Backend Components



@Component is a generic stereotype for any Spring-managed component. @Repository, @Service, and @Controller are specializations of @Component for more specific use cases, for example, in the persistence, service, and presentation layers, respectively.

Annotate based on Function

- OPTION - annotate all your component classes with @Component

- Using @Repository, @Service, and @Controller is:
  - Better suited for processing by tools
    - @Repository - automatic translation of exceptions
    - @Controller – rich set of framework functionality
    - @Service – “home” of @Transactions
  - More properly suited for associating with aspects
  - May carry additional semantics in future releases of the Spring Framework.

Spring MVC XML Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <context:component-scan base-package="app04a.controller"/>
    <mvc:annotation-driven/>

    <mvc:resources mapping="/css/**" location="/css/"/>

    <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>
</beans>
```

XML Configuration file – enable annotations

<context:component-scan base-package= “pkg,pkg…" **>**

Scans defined packages to find and register @Component-annotated classes and activate basic annotations[e.g. @Autowired] - within the current application context

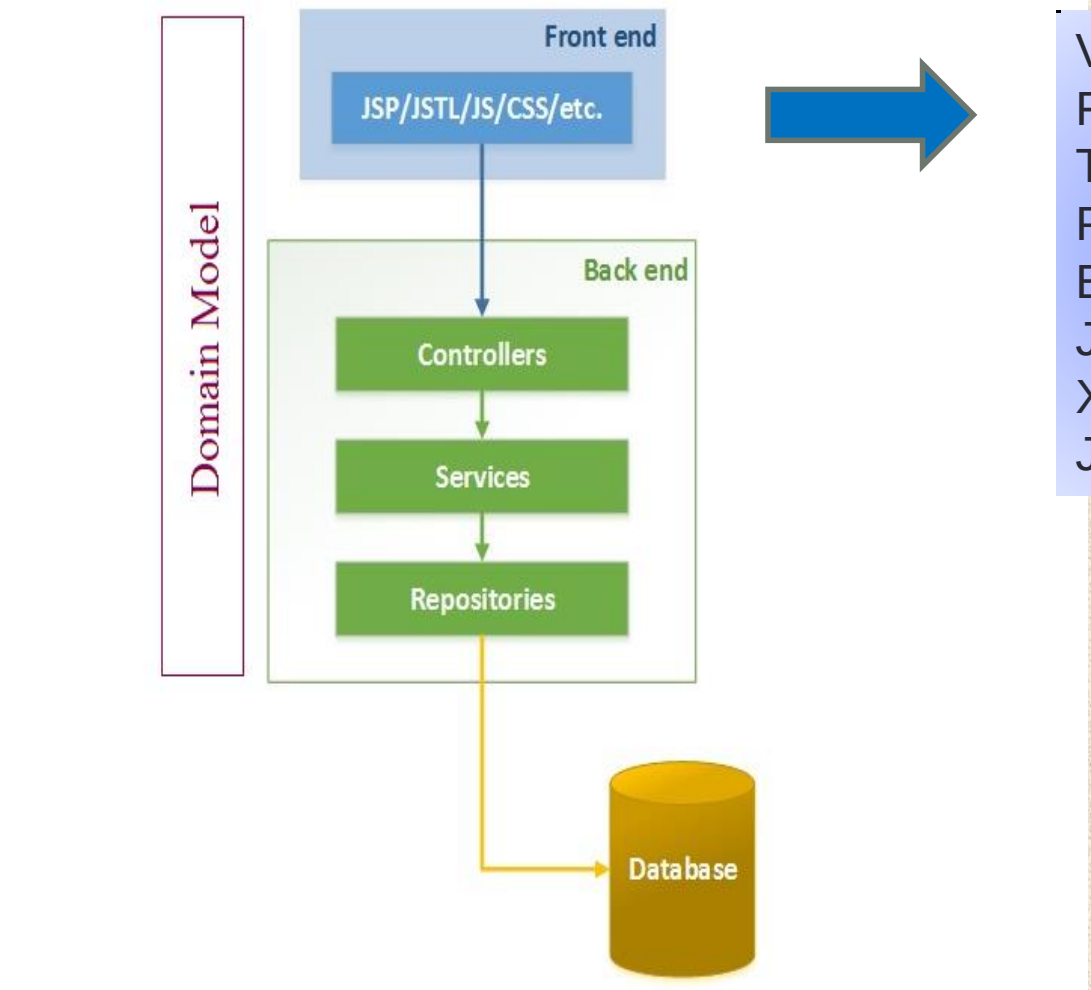
<mvc:annotation-driven/>

Enables support for specific annotations [e.g. @RequestMapping, etc.] that are required for Spring MVC to dispatch requests to @Controllers. It is based on MVC XML namespace

<tx:annotation-driven />

Enables support for specific annotations that are required for Spring Transactions @Transaction It is based on transaction XML namespace.

Spring Layers – With Spring MVC Layer



- Velocity
- FileMarker
- ThymeLeaf
- PDF
- Excel
- JSF
- XML
- JSON

Service Layer

- **Issue: not whether or not it is needed**

BUT

What it contains

Domain Driven Design

- Primary focus - the core domain and domain logic.
- Complex designs based on a model of the domain.
- Collaboration between technical and domain experts to iteratively refine a conceptual model that addresses particular domain problems.

- RESULT : a Rich Domain Model

Anemic Domain Model

- Contains objects properly named after the nouns in the domain space
- Objects are connected with the rich relationships and structure that true domain models have.

- Little or no behavior – bags of getters and setters.

Service Layer

- In a perfect world:

"Thin Layer"

- No business rules or knowledge
- Coordinates tasks
- Delegates work to collaborations of domain objects

"The Reality"

Main Point

- An N Tier Architecture separates an application into layers thereby supporting a separation of concerns making any application more efficient, modular and scalable. *Life is structured in layers. It is a structure that is both stable and flexible, consistent yet variable and it encompasses an infinite range of possibilities*

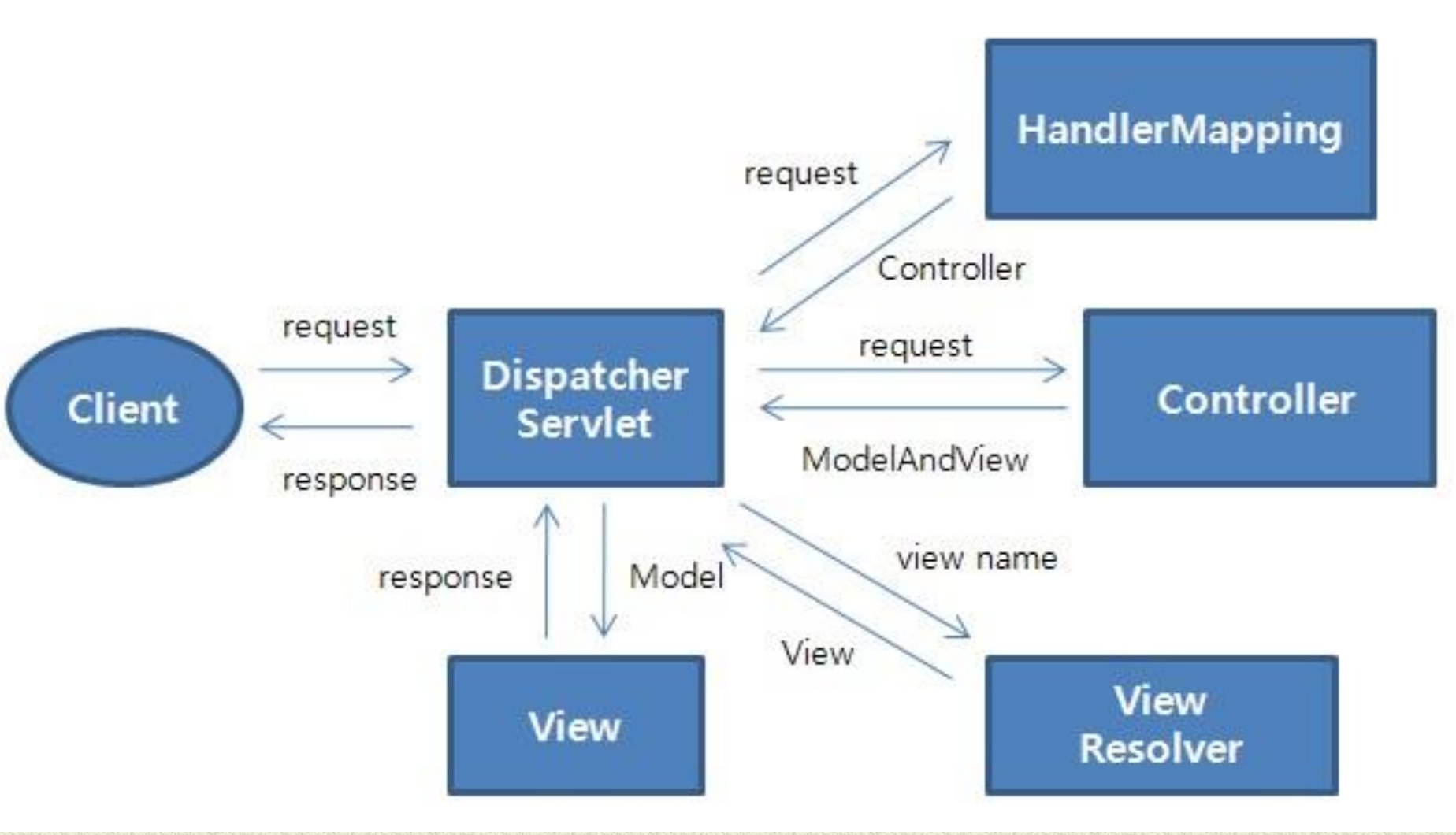
Spring MVC

Distinct Separation of Concerns

Cleary defined interfaces for role/responsibilities “beyond” Model-View-Controller

- Single Central Servlet
  - Manages HTTP level request/response
  - delegates to defined interfaces
- Models integrate/communicate with views
  - No need for separate form objects
- Views are plug and play
- Controllers allowed to be HTTP agnostic
-

Spring MVC Major Interfaces



Spring MVC Flow

- After receiving an HTTP request, *DispatcherServlet* consults the *HandlerMapping* to call the appropriate *Controller*.
- The *Controller* takes the request and calls the appropriate service methods based on GET or POST method. The service method will set model data based on defined business logic and return view name to the *DispatcherServlet*.
- The *DispatcherServlet* will take help from *ViewResolver* to pickup the defined view for the request.
- Once view is finalized, The *DispatcherServlet* passes the model data to the view which is finally rendered on the browser.

Spring MVC DispatcherServlet

- Single Central Servlet that dispatches requests to controllers and offers other functionality that facilitates the development of web applications.

- Completely integrated with the Spring container

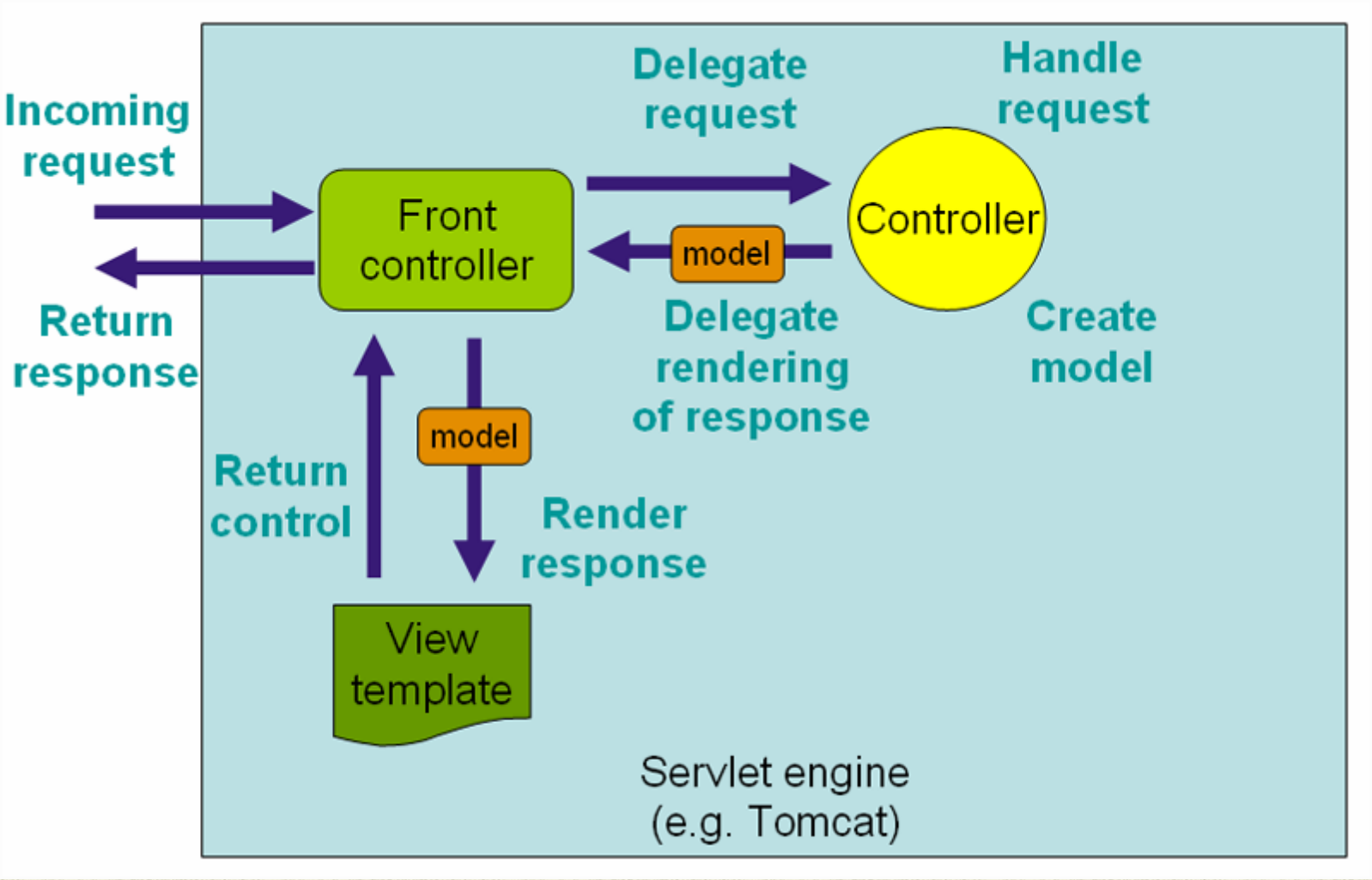
  Able to “exploit” Spring framework features

- Has a WebApplicationContext, which inherits all the beans already defined in the root WebApplicationContext.

- DispatcherServlet - "Front Controller" design pattern

  Common pattern used by MVC frameworks

Spring MVC Front Controller



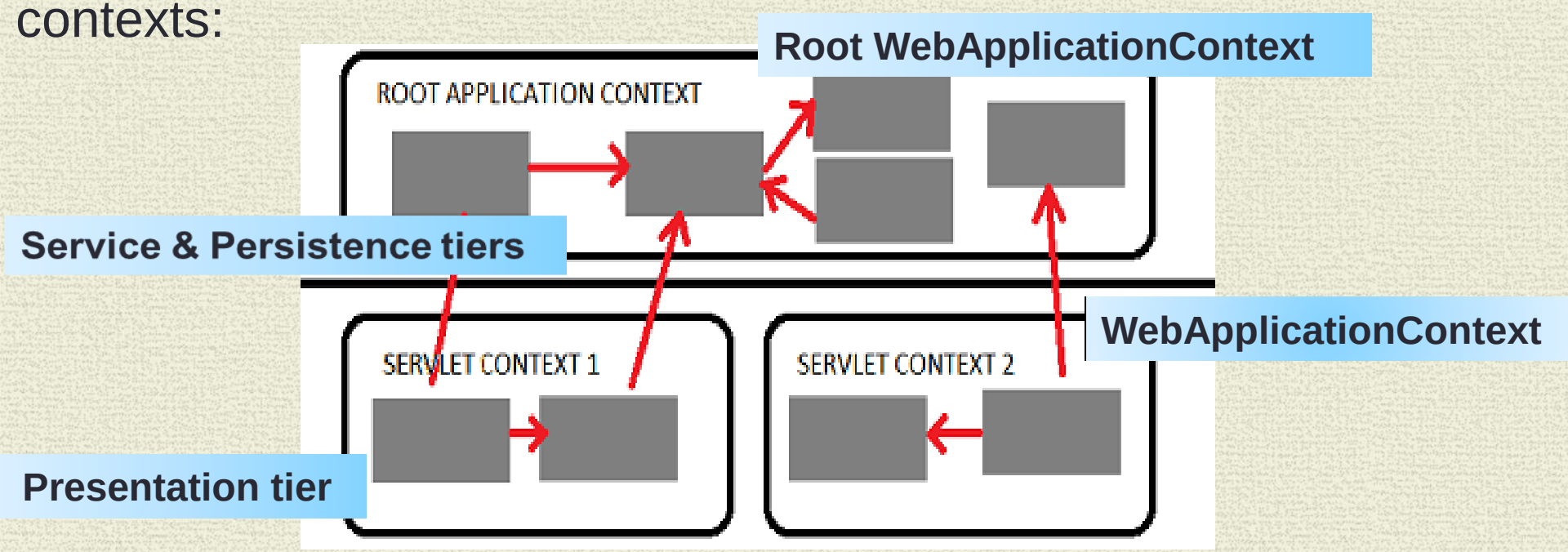
Spring MVC Front Controller Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd
    <servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/config/springmvc-config.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

Web Application Context

- Spring has multilevel application context hierarchies.
- Web apps by default have two hierarchy levels, root and servlet contexts:



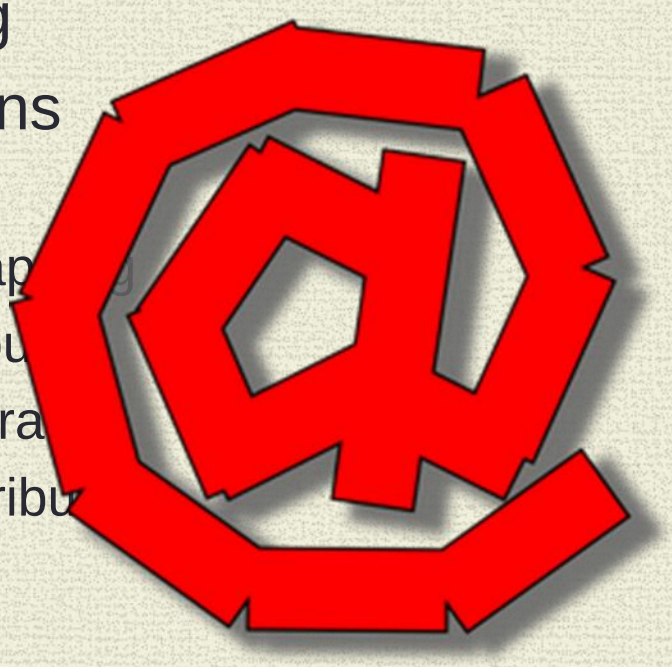
- **Presentation tier has a WebApplicationContext [Servlet Context] which inherits all the resources already defined in the root WebApplicationContext [Services, Persistence]**

Main Point

- The basic ingredients of a Spring MVC application include web pages for the view (the known), the back end domain logic and data (knower or underlying intelligence), and the Spring framework and Dispatcher Servlet and managed beans as the controller to connect the view and model. *Knowledge is the wholeness of knower, known, and process of knowing.*

Spring MVC Architecture & Annotations

- Handler Mapping
- Spring Annotations
  - @Controller
    - @RequestMap
    - @ModelAttribu
    - @RequestPara
    - @SessionAttribu
- ViewResolvers
- Views



Spring MVC @Controller

- Spring implements a controller in a very abstract way, which enables you to create a wide variety of controllers.
- Spring Controllers do not extend specific base classes or implement specific interfaces
- They do not have direct dependencies on Servlet APIs, although you can easily configure access to Servlet facilities. [actual request, response objects, etc.]
-

@RequestMapping variables

@RequestMapping()

- **1. Value = “/product”**
- **2. Method=RequestMethod.GET**
- _____

- **3. consumes="application/json"**

  content-type request header
- **4. produces="application/json**

  Accept request header
- **5. params="myParam=myValue“**

  request parameter conditions such as "myParam", "!myParam", or "myParam=myValue".
- **6. headers="myHeader=myValue"**

  request header conditions such as "myHeader", "! myHeader", or myHeader=myValue

@RequestMapping Using Method Parameter

- @RequestMapping(value = "/add", method = RequestMethod.GET)
- public String getAddNewProductForm(Product newProduct) {
- return "addProduct";
- }
-
- @RequestMapping(value = "/add", method = RequestMethod.POST)
- public String processAddNewProductForm (Product productToBeAdded) {
  }

- NOTE: multiple URLs is also valid
- @RequestMapping({"/","/product_input"})

@RequestParam

- Placed on Method argument
- http://localhost:8091/webstore3/products/product?id=P1234

```
public String getProductById(@RequestParam("id")String productId,Model
model) {
model.addAttribute("product", productService.getProductById(productId));
 }
```

- **Handling multiple values [e.g., multiple selection list]**

http://localhost:8091/store/sizechoices?sizes=small&sizes=medium&sizes=large

```
public String getSizes(@RequestParam(“sizes")String sizeArray[]
```

Simple Spring Controller example

```
package app04a.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import app04a.domain.Product;

@Controller
public class ProductController {

    @RequestMapping(value={"/","/product"}, method = RequestMethod.GET)
    public String inputProduct() {
        return "ProductForm";
    }

    @RequestMapping(value="/product", method = RequestMethod.POST)
    public String saveProduct(Product product ) {
        return "ProductDetails";
    }
}
```

Inversion of Control [IOC]

Objects do not create other objects that they depend on.

- Promotes loose coupling between classes and subsystems
- Adds potential flexibility to a codebase for future changes.
- Classes are easier to unit test in isolation.
- Enable better code reuse.

- IOC is implemented using **Dependency Injection**(DI).

Dependency Injection [DI]

- DI exists in three major variants
- Dependencies defined through:
  - Property-based dependency injection.
  - Setter-based dependency injection.
  - Constructor-based dependency injection
- Container *injects* dependencies when it creates the bean.

Dependency Injection examples

- **Property based:**

```
@Autowired
ProductService productService;
```

- **Setter based:**

```
ProductService productService;
@Autowired
public void setProductService(ProductService productService){
        this.productService = productService;
```

- **Constructor based:**

```
ProductService productService;
@Autowired
public ProductController(ProductService productService) {
        this.productService = productService;
```

Main Point

- Annotations are metadata that allow the knowledge about the creation of an object to reside with the object itself
- *The self-referral nature of Transcendental Consciousness makes all our actions clearer and more powerful*

DEMO

Product4a as a JSP

Product4a as a Controller