Spring MVC Introduction

Cognizant
Passion for building stronger businesses

About the Author

Created By:	Ramesh C.P. (161646)
Credential Information:	SCJP, 8+ years of experience in technical training
Version and Date:	Spring MVC/PPT/1011/3.0

Cognizant Certified Official Curriculum

Icons Used

Questions

Tools

Hands on Exercise

Coding Standards

Test Your Understanding

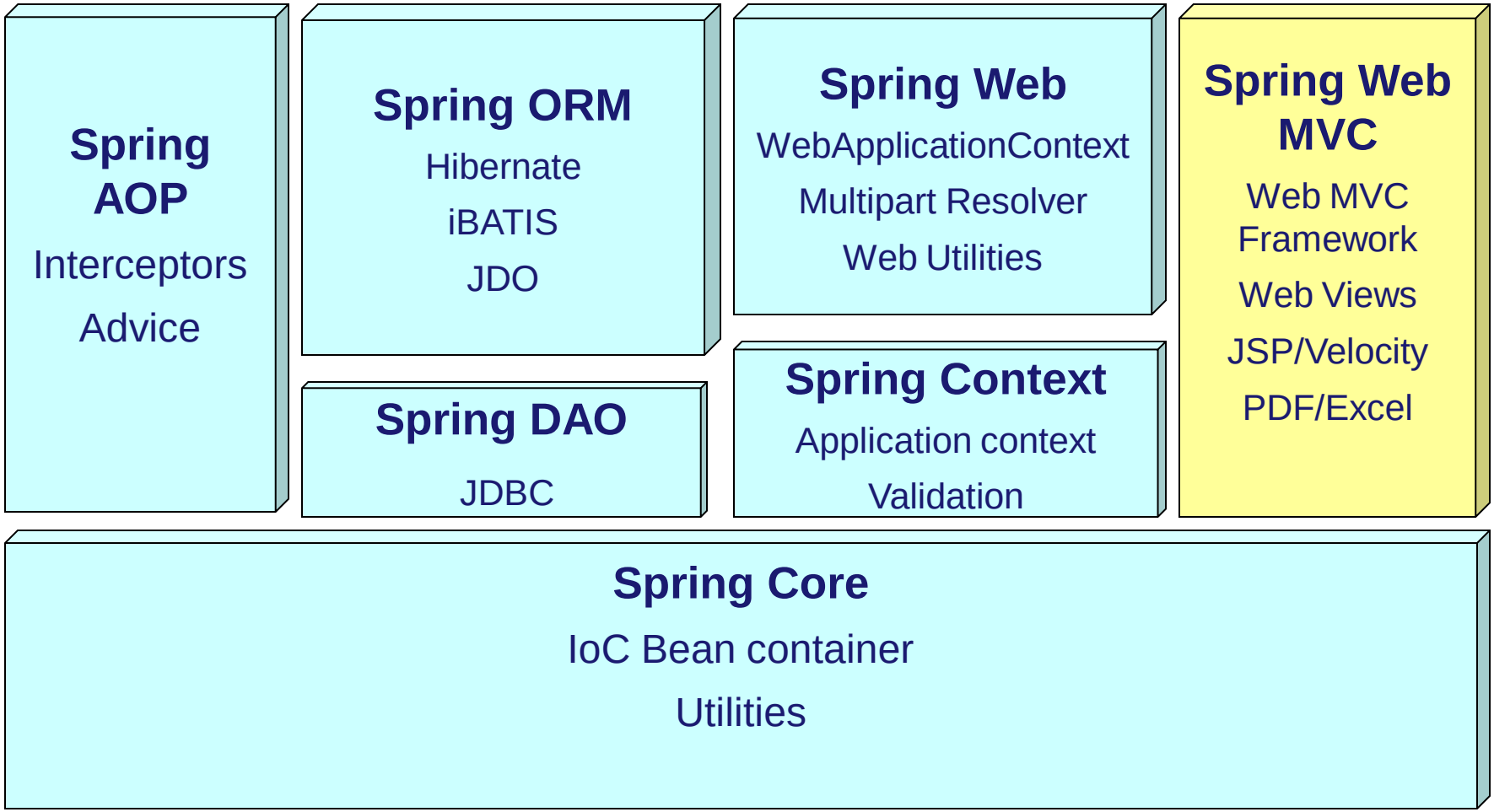
Reference

Demonstration

A Welcome Break

Contacts

Spring Architecture



Spring MVC Introduction: Overview

- What is Spring MVC?
  - Request-response based web application framework, such as Struts and WebWork
  - Flexible and extensible via component's interfaces
  - Simplifies testing through Dependency Injection
  - Simplifies form handling through its parameter binding, validation and error handling
  - Abstracts view technology (JSP, Velocity, FreeMarker Excel , PDF)
  - Annotation based programming model

Overview of Spring MVC

- Spring MVC is an MVC web framework
  - Many ready-to-use and highly customizable Controllers
  - Many different View models (JSP, PDF, XSL, Excel, etc.) and View Resolvers
  - Data Binding for forms
  - Wizard functionality for multiple forms
  - Validation
  - I18N and Theme support
  - Interceptors
  - Miscellaneous: file upload, tag library, etc.

Spring MVC Introduction: Objectives

- Objective:

After completing this chapter you will be able to:

- Explain the Spring MVC architecture
- Describe Various components in the Spring MVC
- Configure DispatcherServlet in web.xml

Servlets ruled the world

- And things were good
- Faster than CGI programs
- More scalable with built-in threading capability
- Value-added features:
  - Request/Response, Sessions
- Full Java API access
  - JDBC, JMS, EJB, JavaMail, and so on

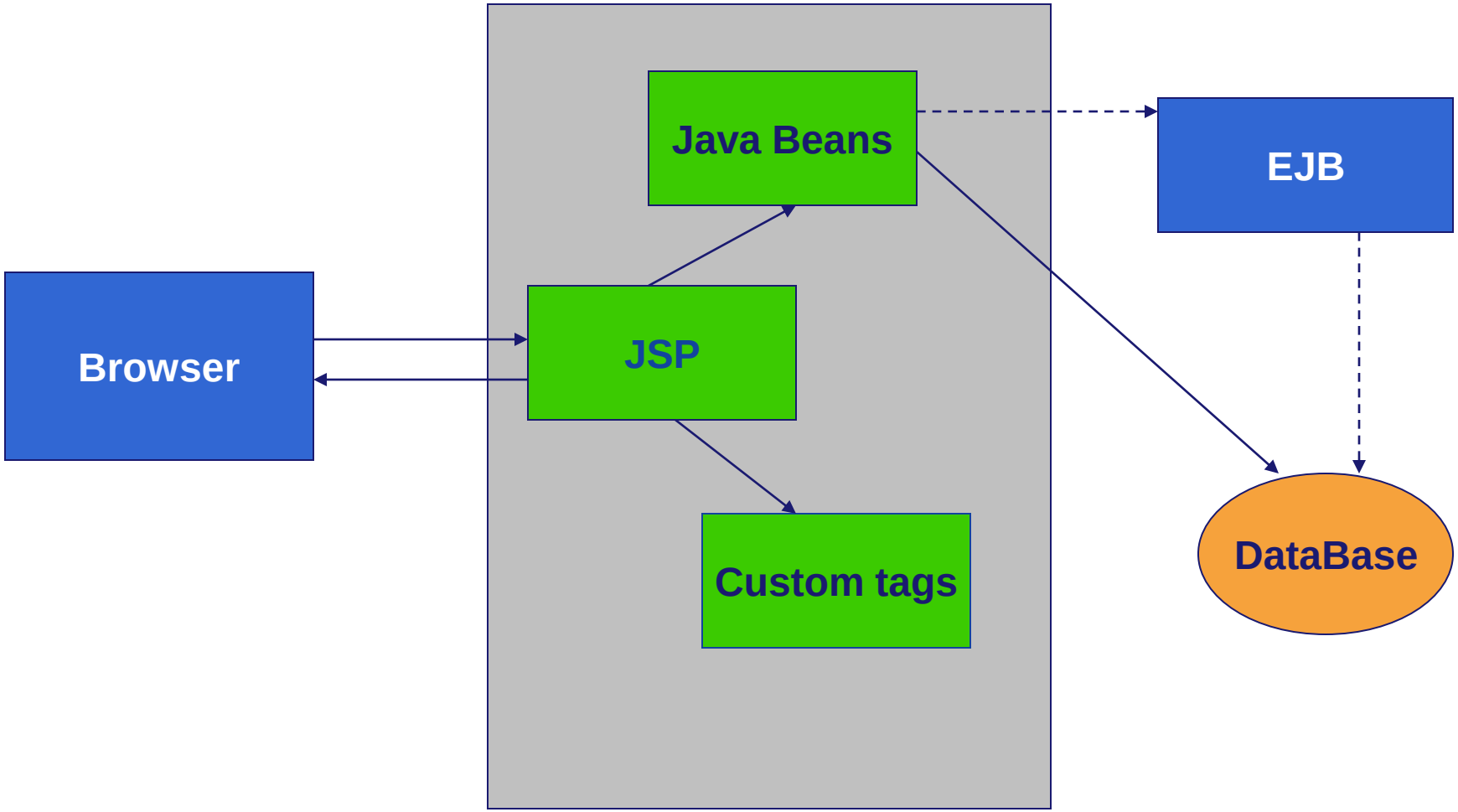
Problems with Servlets

- out.println in the code became tedious
- Takes a programmer to write HTML pages:
  - Skill mismatch
- There has to be a better way

Then came the JSPs

- Page-centric view of the world
- Limited programmer involvement
- JSPs became ubiquitous for dynamic HTML page generation

Model 1 Architecture



Model 1 Architecture (Contd.)

- JSP centric
- Suitable for small systems
- Susceptible to abuse:
  - Excessive Java code in JSPs
  - Flow control issues

Model 2 Architecture (MVC)

- Use servlet for flow control and JSPs for HTML pages and dynamic content
- Allow programmer/Web designer specialization
- MVC:
  - Controller = Servlet + Helper classes
  - Model = Application data/Operation
  - View = Output rendered to user

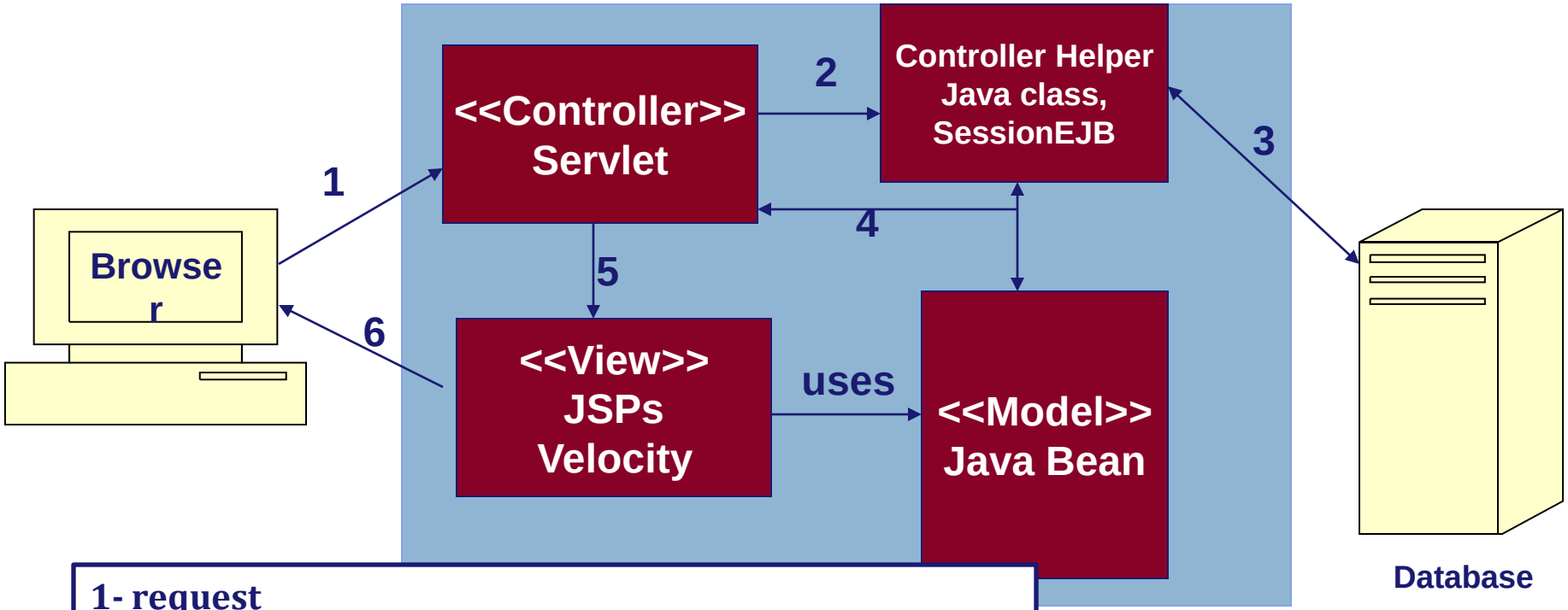
MVC Design Pattern

- Context:
  - MVC is an architectural pattern that is used when developing interactive applications.
- Problems:
  - User interfaces changes often, especially on the internet where look-and-feel is a competitive issue.
  - Also, the same information is presented in different ways.
  - The core business logic and data is stable.

MVC Design Pattern (Contd.)

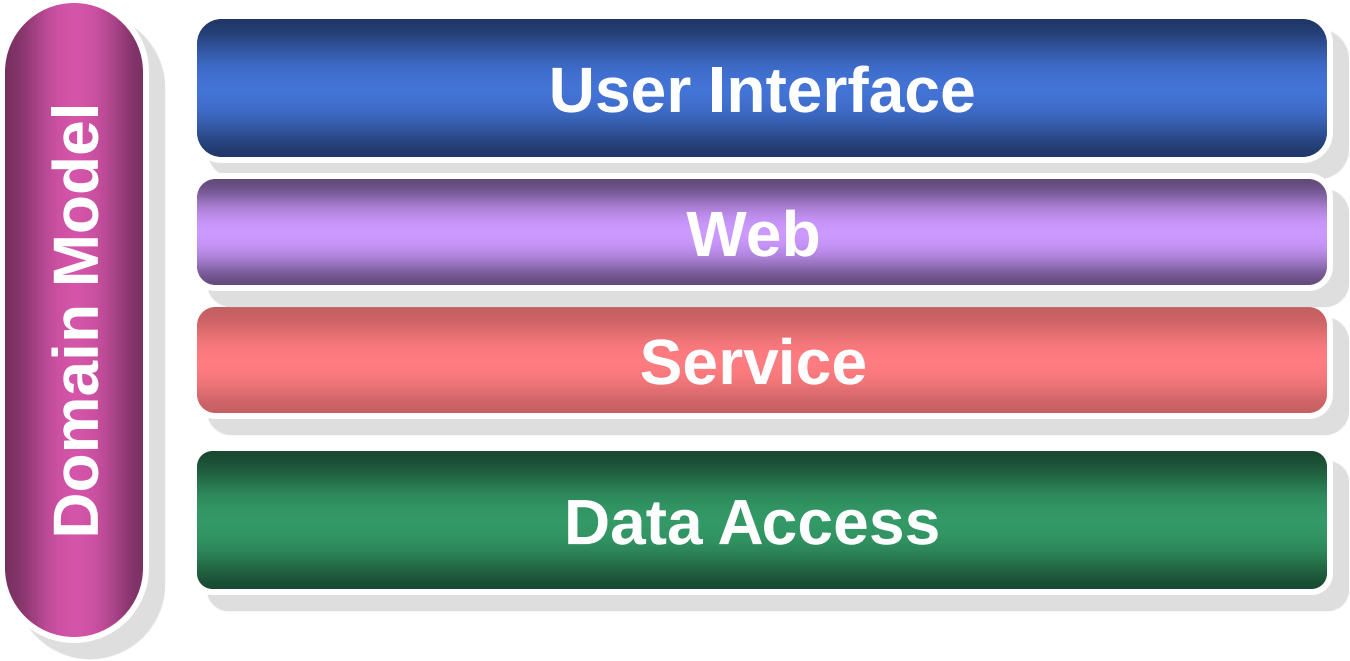
- Solution:
  - Use the software engineering principle of "Separation of concerns" to divide the application into three areas:
    - **Model** encapsulates the core data and functionality.
    - **View** encapsulates the presentation of the data. There can be many views for the common data.
    - **Controller** accepts requests and makes requests from the model for the data to produce a new view.

MVC Architecture



1- request
2-Access Model and invoke business logic
3-Connection to data base and get/save data
4- Get result and set in scope
5-Dispatch to next view
6-Response

Spring MVC application Layers



User Interface Layer

- org.springframework.web.servlet.View – interface that represents a view or page, of the Web application.
  - It is completely generic and has no specific view rendering dependencies
- Spring MVC natively supports JSP, FreeMarker, Velocity, XSLT, JasperReports, Excel and PDF.
- org.springframework.web.servlet.ViewResolver-provides a map between view instances and their logical names

Web Layer

- Spring MVC provides an org.springframework.web.servlet.mvc.Controller interface and a very rich class hierarchy below it for its web layer contract.

Service Layer

- Spring framework does not provide any interfaces or classes for implementing the business aspects of the service layer, because the service layer is specific to the application.
- Spring helps in injecting instances of the service into the web Controllers.
- Spring will also enhance your service layer with services such as transaction management, performance monitoring and even pooling if you decide you need it.

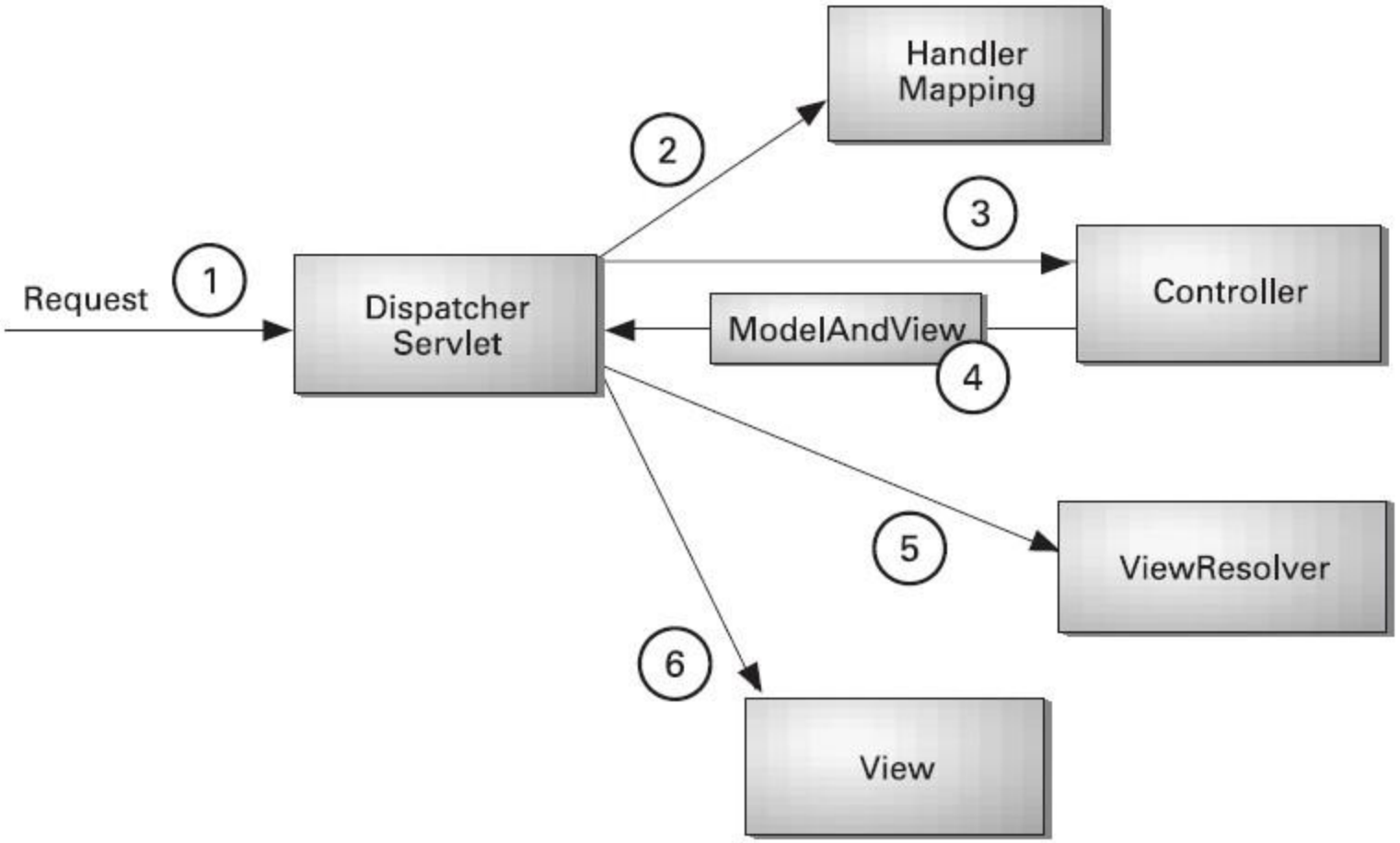
Domain Model Layer

- Just like the service layer, Spring does not provide any base interface for your object model.
- Spring helps in enhancing your domain model via AOP.
- To Spring, both the service layer and domain model are simply a set of POJOs.

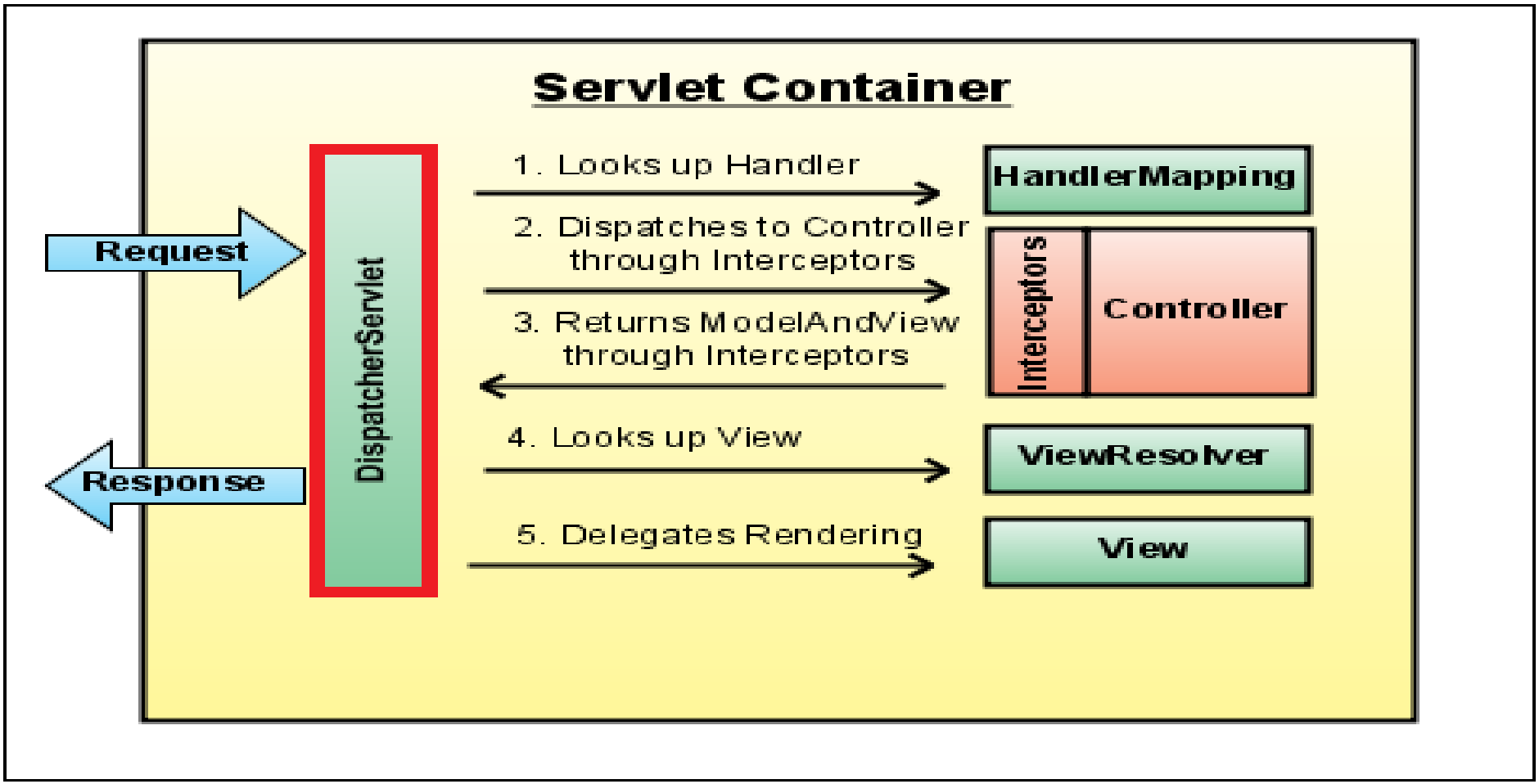
Data Access Layer

- Spring provides a common patterns for interacting with the data access layer:
  - HibernateTemplate for Hibernate
  - JdbcTemplate for JDBC
  - SqlMapTemplate for iBATIS
- One of the main benefits of Spring is its very rich and deep data access exception hierarchy.

Request Life Cycle in Spring MVC



Dispatcher Servlet



Dispatcher Servlet (Contd.)

- Heart of Spring MVC
- Example of *Front Controller* pattern - entry point for all Spring MVC requests
- Loads WebApplicationContext
- Controls workflow and mediates between MVC components
- Loads sensible default components if none are configured

Configuring the DispatcherServlet

web.xml

```
…
<!- the spring config file name should be
  dispatcher-servlet.xml -->
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>

  org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

Configuring the DispatcherServlet

web.xml

```
…
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>*.html</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>DispatcherSample</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>DispatcherSample</servlet-name>
  <url-pattern>*.form</url-pattern>
</servlet-mapping>…
```

Configuring the Dispatcher Servlet (Contd.)

web.xml

```
…
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/services.xml</param-value>
</context-param>

<listener>
    <listener-class>
     org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
…
```

configures parent context

Two Application Context

- Spring MVC application usually have two ApplicationContexts configured in a parent-child relationships.
- The parent context or Root ApplicationContext, contains all of the non-web specific beans such as the services, DAOs etc and it is created via ContextLoaderListener.
- DispatcherServlet also creates the WebApplicationContext (from the config file under the name <servletname>-servlet.xml), which contains the web-specific components such as the Controllers, ViewResolvers and so on.
- The WebApplicationContext is nested inside the root ApplicationContext so that the web components can easily find their dependencies

The WebApplicationContext

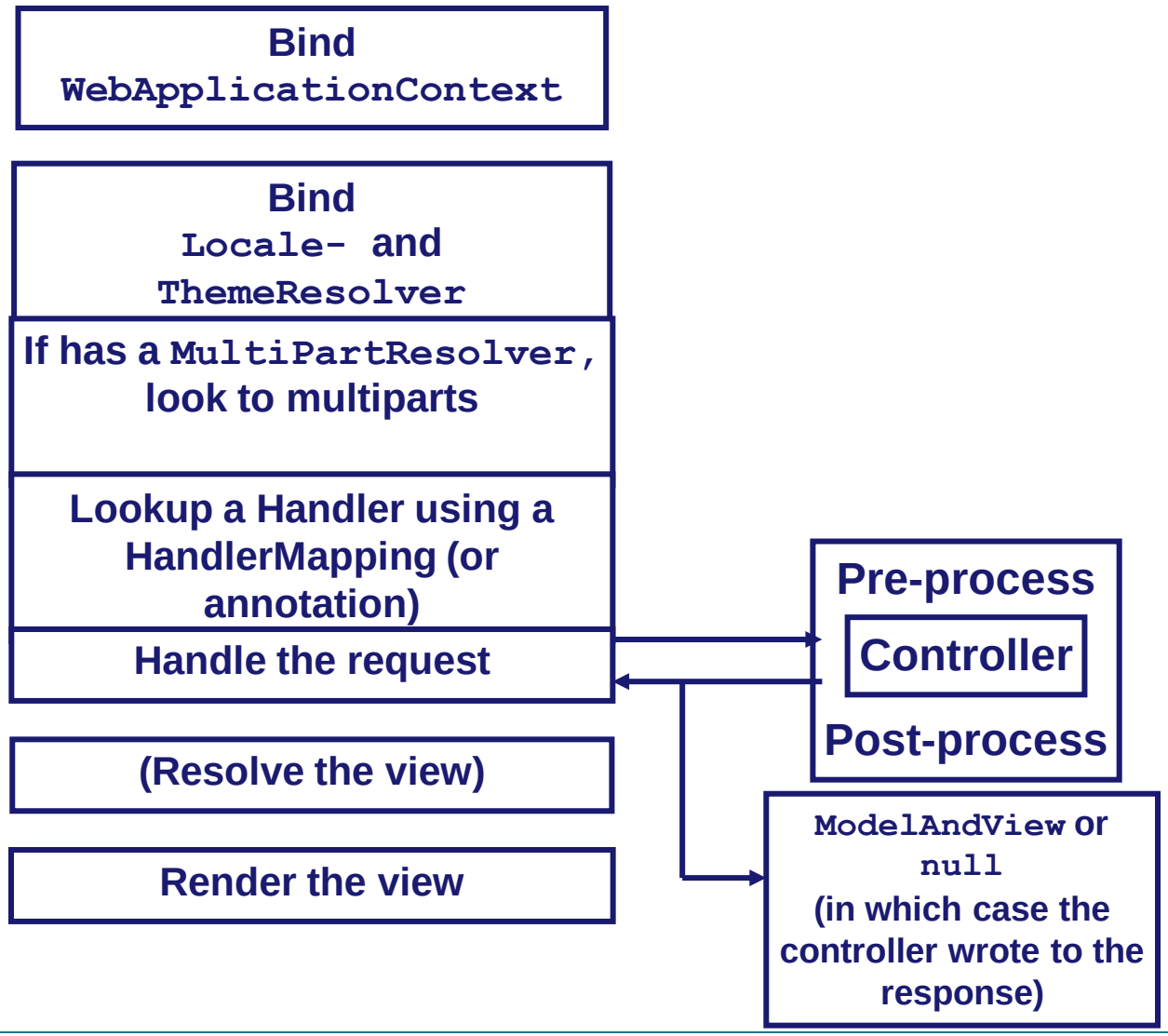
- The *WebApplicationContext* is similar to *ApplicationContext* except it:
  - Can resolve *Themes* and *Locales* (covered later)
  - Is bound to the `ServletContext` and placed in the `Request`

The WebApplicationContext (Contd.)

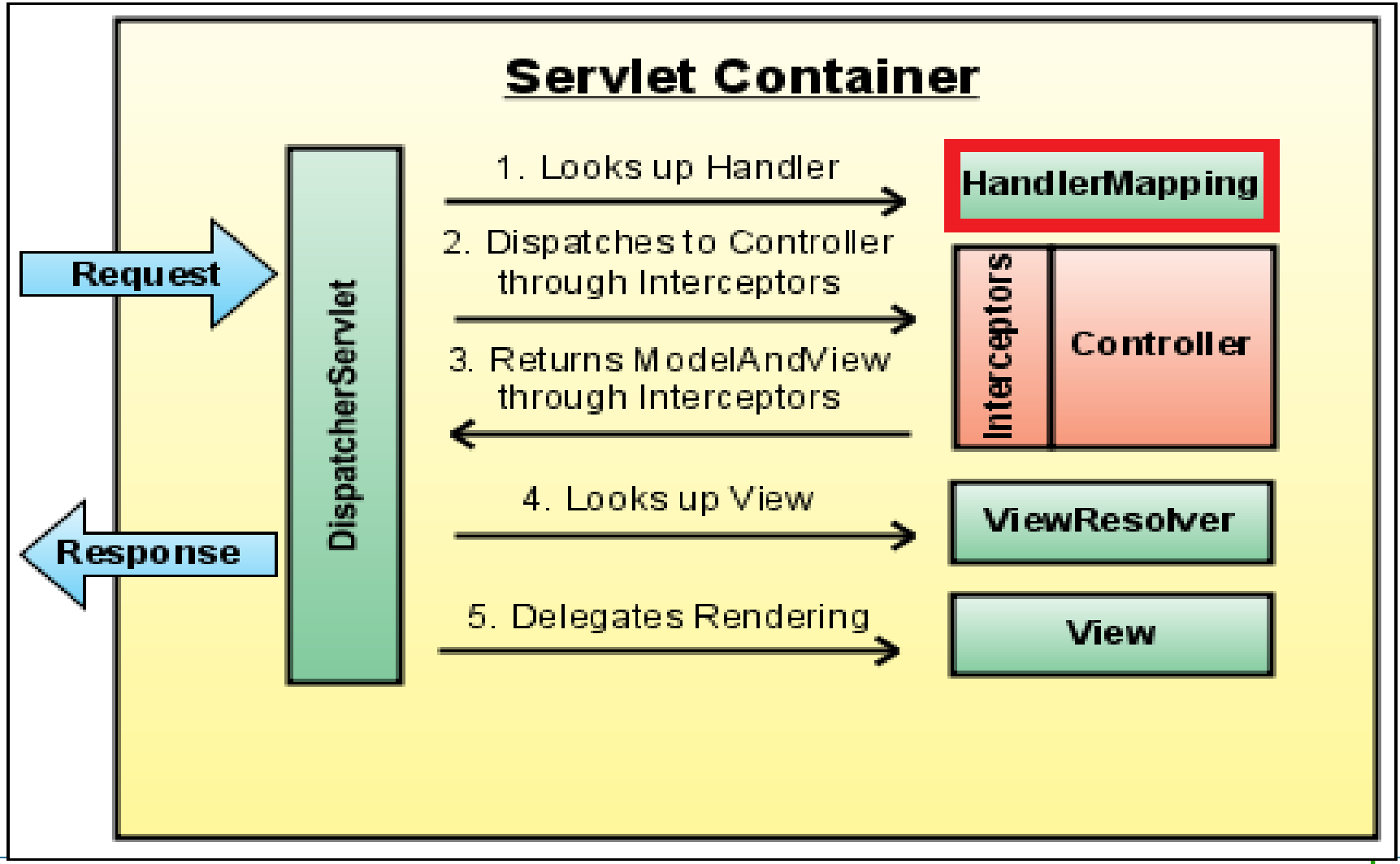
- The *WebApplicationContext* uses the following set of beans:

Beans of type	Description
`HandlerMapping`	Defines criteria for which interceptors and controller to use
`Controller`	Defines the request handlers (they contain the process logic)
`ViewResolver`	Maps names to actual View objects (as opposed to being hard coded)
`LocaleResolver`	Resolves the `Locale` to use for this client (used for `I`18N)
`ThemeResolver`	Resolves Themes based on input such as Cookies
`MultipartResolver`	Used for file upload capabilities of Spring MVC
`HandlerExceptionResolver`	Maps exceptions to `Views`

Workflow of Request Handling



HandlerMapping



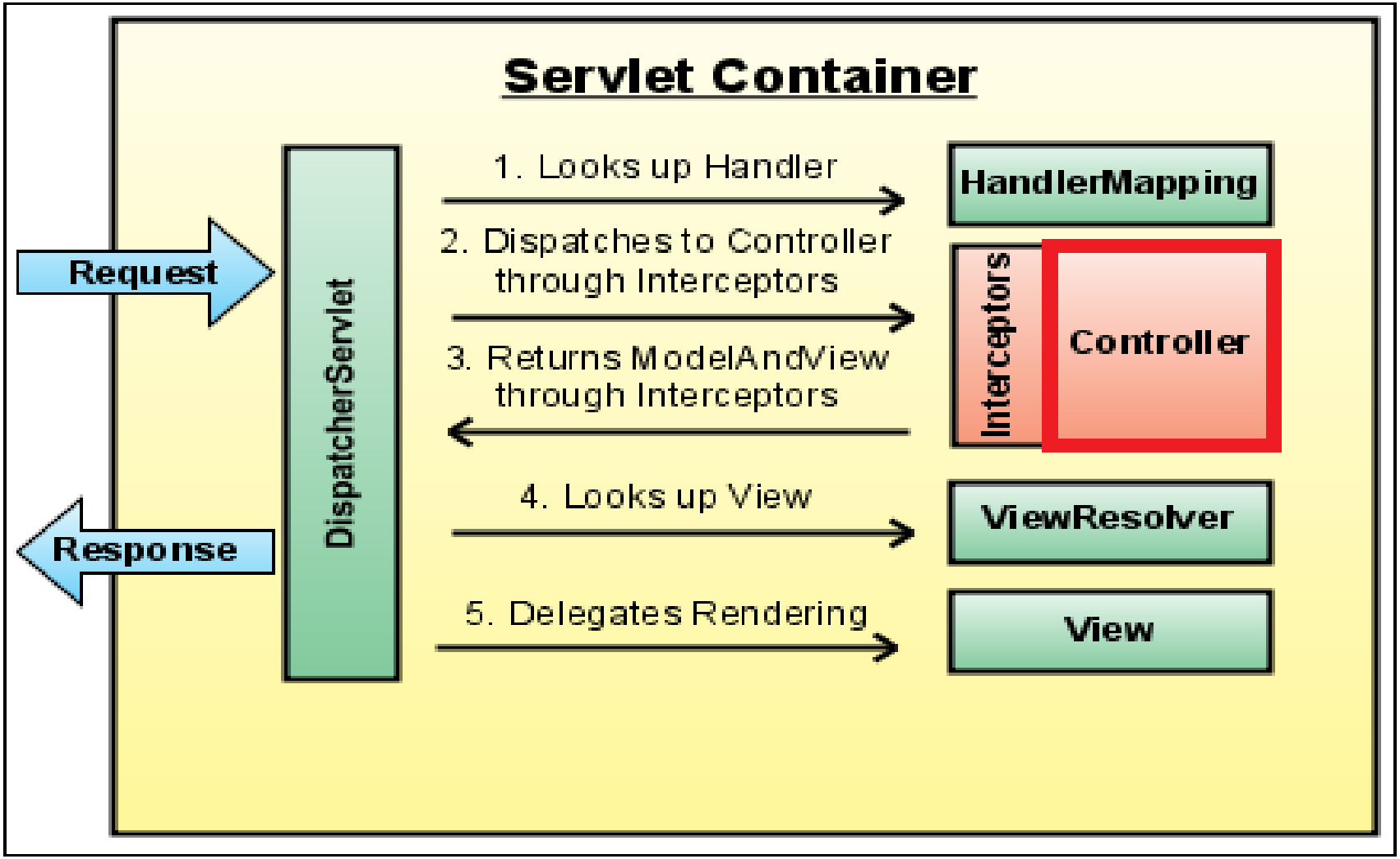
HandlerMapping interface

- Maps incoming requests to a corresponding Handler, typically a Controller.
- Rarely needs to be implemented directly – many useful implementations are provided.
- Also ties Interceptors to a mapped Controllers.
- Can provided multiple HandlerMappings; priority is set using **`Ordered`** interface.

HandlerMapping Implementations

- BeanNameUrlHandlerMapping
- SimpleUrlHandlerMapping
- ControllerClassNameHandlerMapping

Controllers



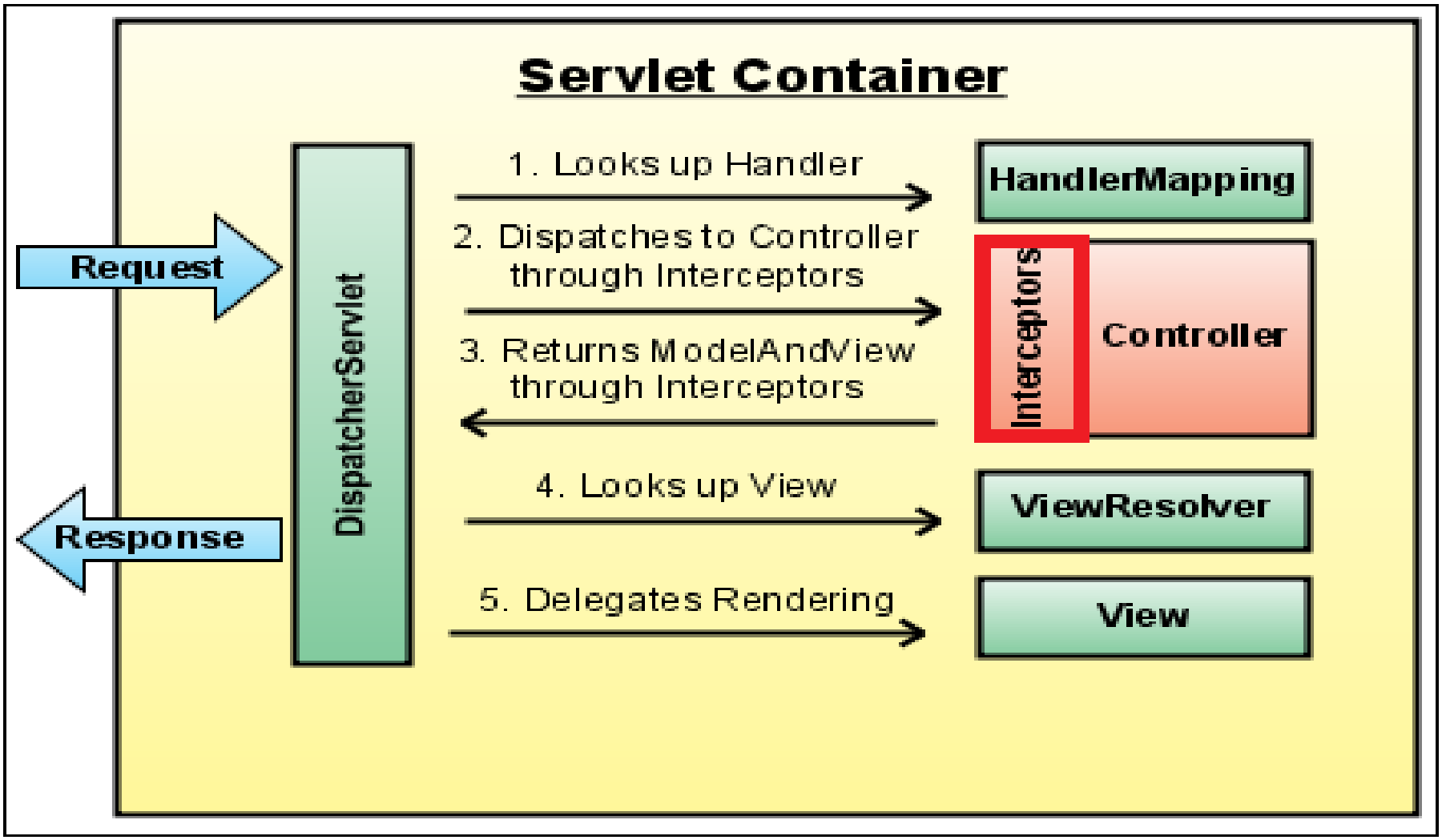
Controller Interface

- Handles the processing of the request
- Interface parameters mimics **`HttpServlet`**
  - **`handleRequest(HttpServletRequest, HttpServletResponse)`**
- Returns a **`ModelAndView`** object
- Implementations are typically thread-safe
- Since 2.5.x release, Spring has introduced an annotation-based programming model for MVC controllers, @Controller.

Model And View object

- Encapsulates both model and view that is to be used to render model.
- Model is represented as a **`java.util.Map`**
- Objects can be added to without name:
  - **`addObject(String, Object)`** – added with explicit name
  - **`addObject(Object)`** – added using name generation (*Convention over Configuration*)
- View is represented by **`String`** or **`View`** object.

Interceptors



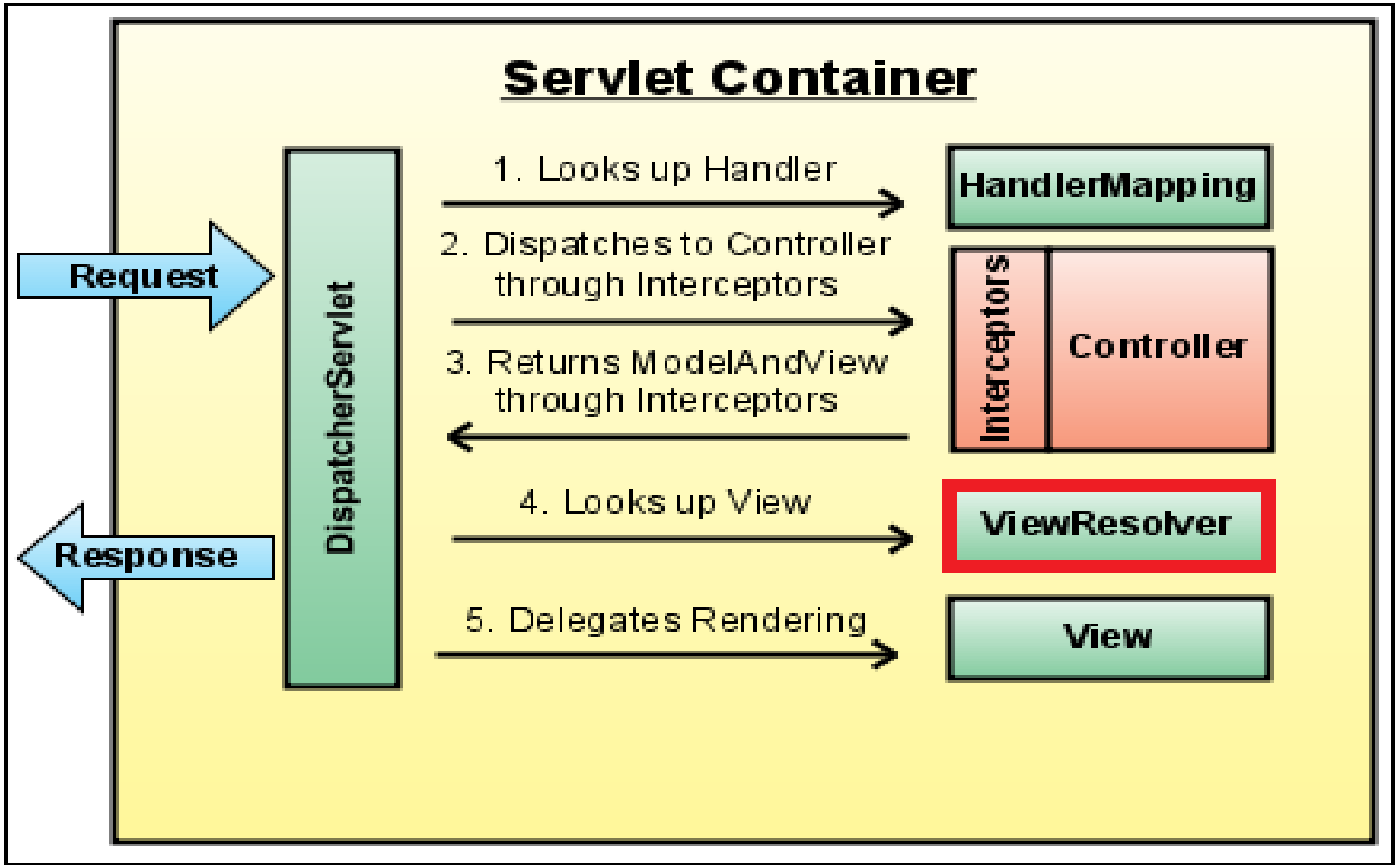
Interceptors (Contd.)

- Add additional functionality before and after request
- Contains to interception methods – **`preHandle`** and **`postHandle`**
- Contains one callback method – **`afterCompletetion`**
- Associated with a set of **`Controllers`** via a **`HandlerMapping`**

Interceptor implementations

- HandlerInterceptorAdapter
- UserRoleAuthorizationInterceptor
- LocaleChangeInterceptor

ViewResolver



ViewResolver (Contd.)

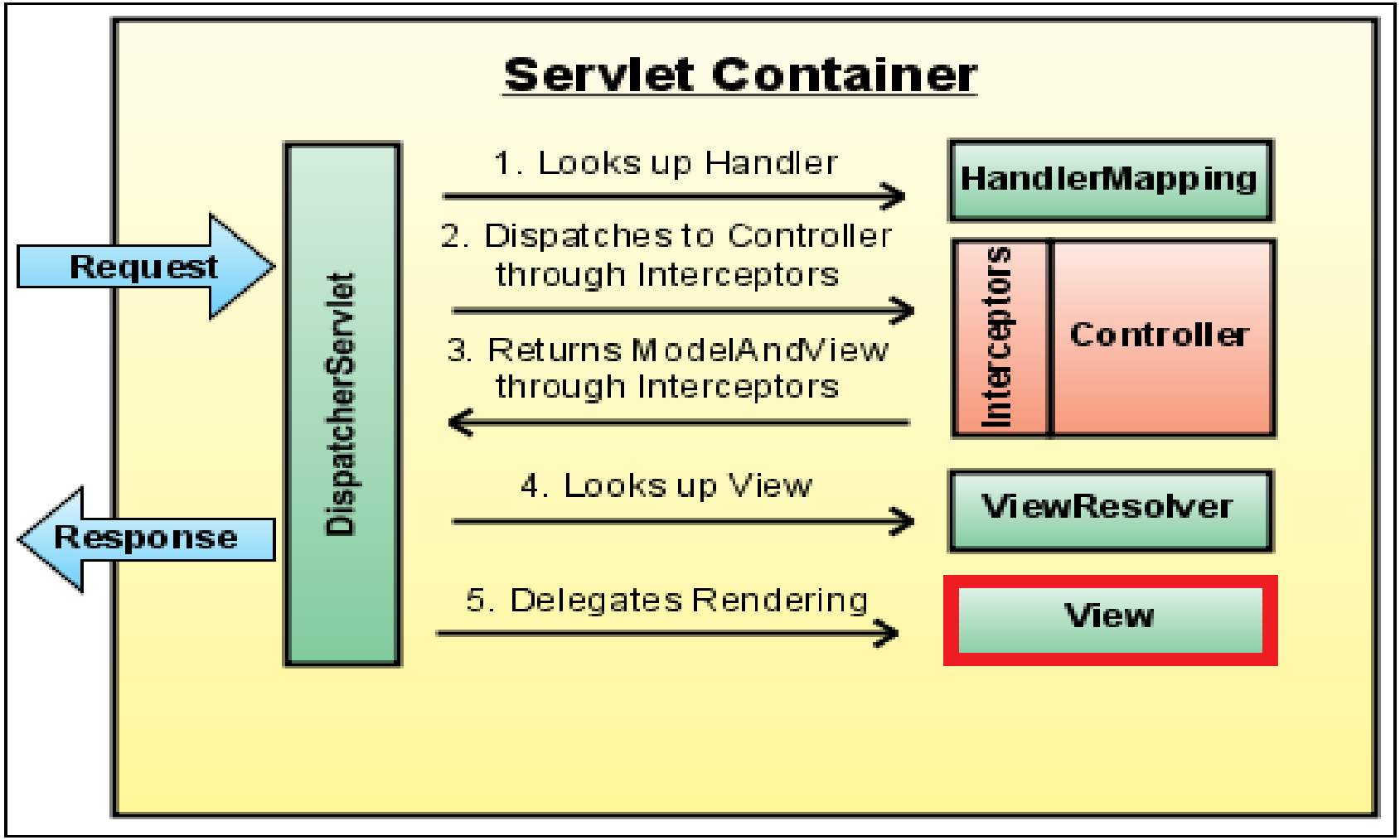
- Resolves a logical view name to a View object
- Orderable, so they be chained
- For JSP user, typical implementation is

  `InternalResourceViewResolver`

Other ViewResolver Implementations

- VelocityViewResolver
- FreeMarkerViewResolver
- ResourceBundleViewResolver
- XmlViewResolver

View



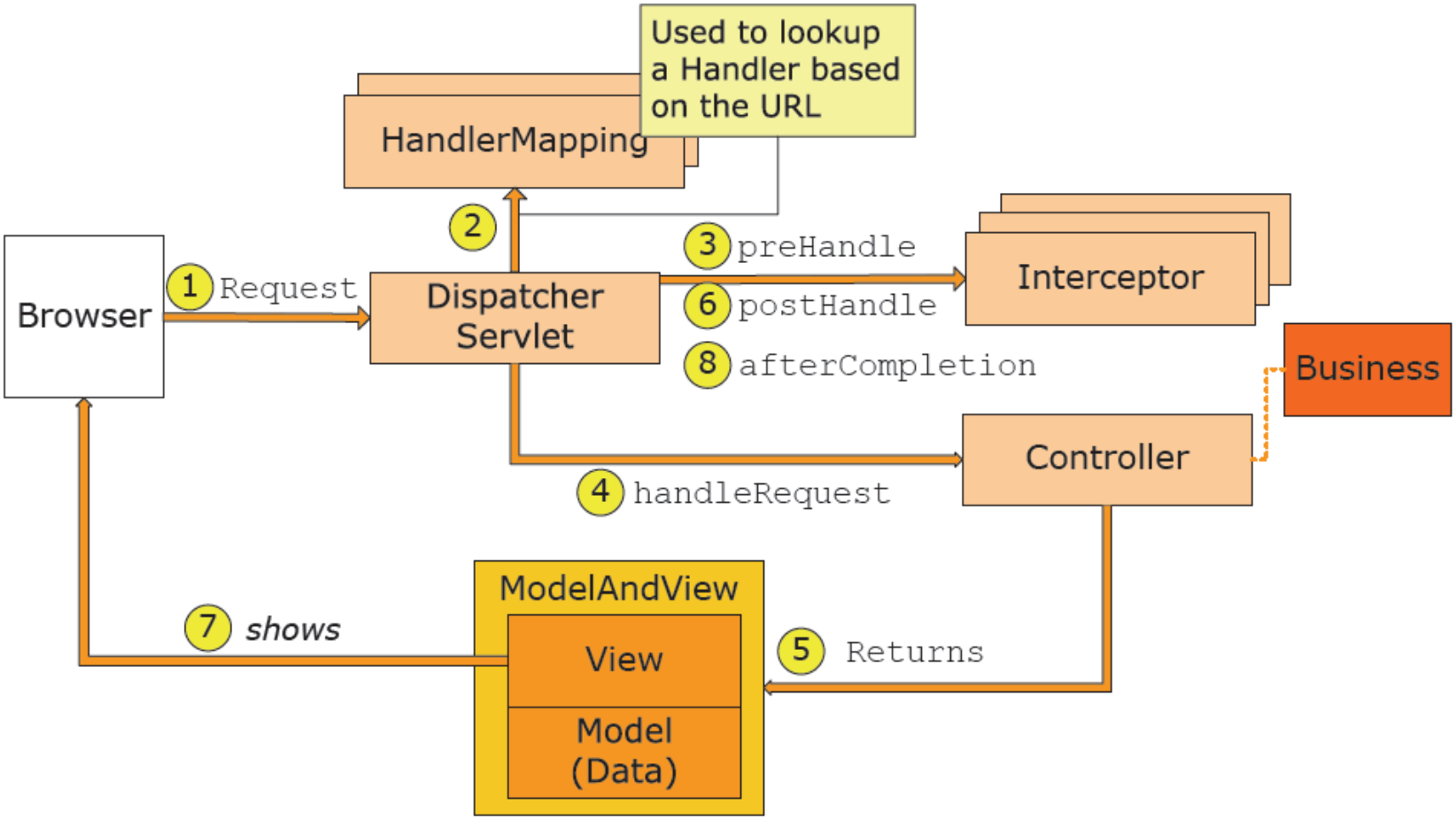
View (Contd.)

- Contains implementations for several template technologies:
  - InternalResourceView (JSP)
  - JstlView (JSP + JSTL)
  - VelocityView (Velocity)
  - FreeMarkerView (FreeMarker)
  - TilesView (Tiles)
  - TilesJstlView (Tiles + JSTL)

View (Contd.)

- Also contains views that support rendering:
  - Excel files
  - PDF files
  - XSLT results
  - Jasper Reports

Review of Architecture



Additional Functionality

- Exception Handling
  - Spring MVC offers a `HandlerExceptionResolver` to handle exceptions that occur while processing a request
  - Maps an Exception type to a particular `ModelAndView`
  - More flexible than the standard *web.xml* exception mapping
  - Mapping can be based on data in the request, response, or the exception in the model
- I18N
  - Spring MVC supports I18N
  - The Locale for a request is resolved using a `LocaleResolver` (e.g. based on the header, session or a cookie)
  - Spring will transparently use locale specific configuration (for example property files)

Additional Functionality (Contd.)

- Themes
  - A site can be themed (personalized look and feel)
  - Theme-specific data can be placed in a resource bundle
  - Theme properties typically include: stylesheets, background, etc.
  - Similar to I18N but differences are based on something besides the Locale
  - Current Theme for a request is resolved using a `ThemeResolver` (e.g. based on cookie or session)
- Wizard
  - Spring MVC facilitates a wizard-style interface
  - Wizard includes multiple views (e.g. forms)
  - Order, navigation and pagination can be configured
  - Uses the `AbstractWizardFormController`
- Uploading
  - Spring MVC offers support for file uploads
  - Enabled by specifying a `MultipartResolver`
  - Bundled implementations include *Jakarta Commons FileUpload* and Jason Hunter's *COS (com.oreilly.servlet)*

Spring MVC Introduction

Time for a Break !

Spring MVC Introduction

- Questions from participants

Spring MVC Introduction: Summary

- Spring framework binds different layers in MVC together transparently, without dictating application design or implementation.
- A Spring MVC application is initialized via the Web.xml, but the configuration is performed via the bean definition XML files such as applicationContext.xml and spring-servlet.xml.
- Spring provided its own implementations for HandlerMapping, Controller, ViewResolver.

Spring MVC Introduction: Source

- http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/spring-web.html
- http://docs.huihoo.com/spring/3.0.x/en-us/pt05.html
- http://static.springsource.org/spring/docs/3.0.x/javadoc-api/

You have successfully completed Spring MVC Introduction

Click here to proceed

Cognizant
Passion for building stronger businesses