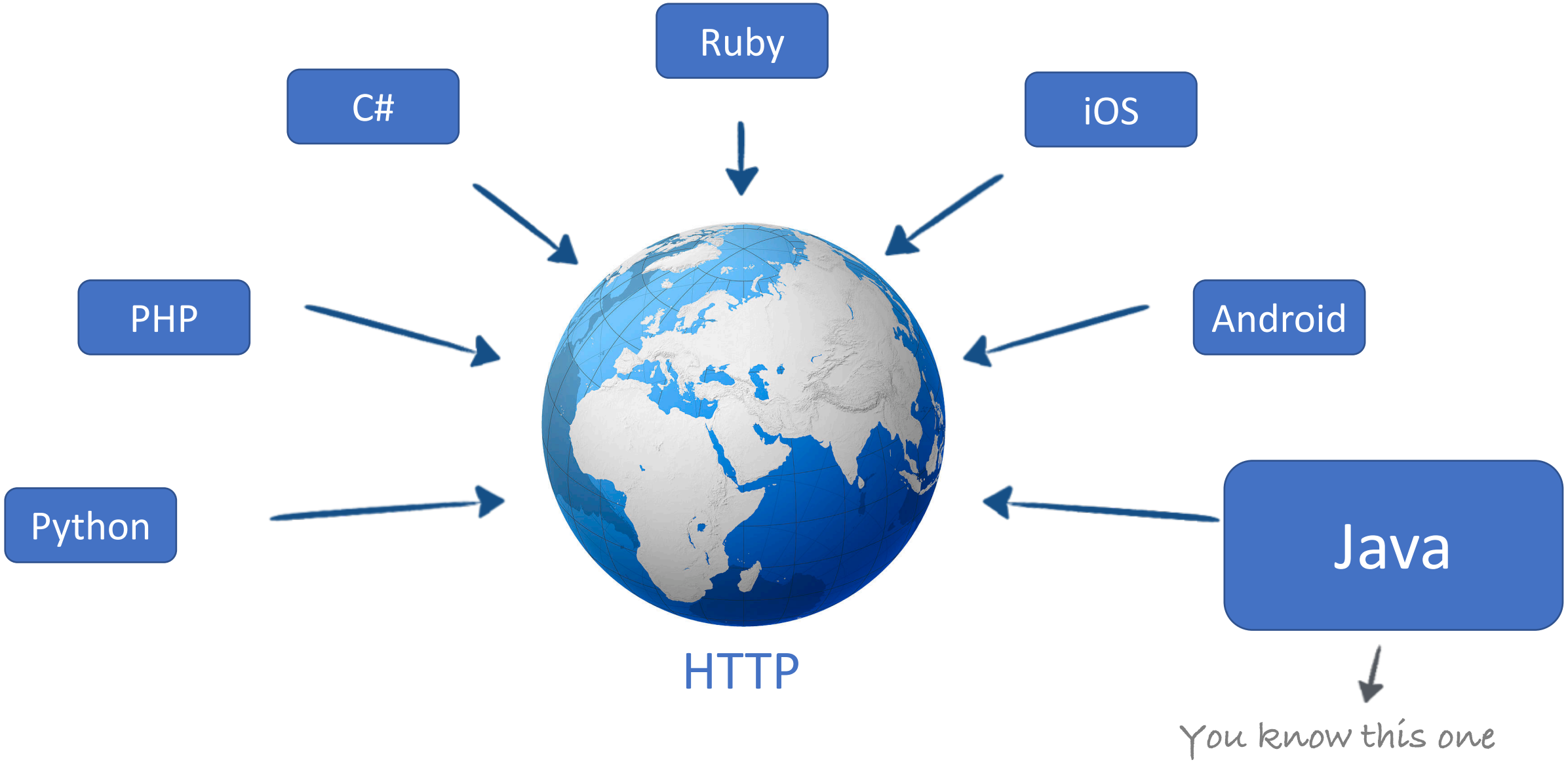
# 1. Frameworks

What to choose?


Ruby
C#
iOS
PHP
Android
Python
Java
HTTP
You know this one

# Java (Web) frameworks …

# Java (Web) frameworks

| Pure Java | Other JVM |
| --- | --- |
| Spring | Grails (Groovy) |
| Struts | Play (Java - Scala) |
| JSF | |
| Wicket | |
| Vaadin | |
| GWT | |

| Request based | Component based |
| --- | --- |
| SpringMVC | JSF |
| Struts | Wicket |
| Play | Vaadin |
| Grails | GWT |

# Request based

- Requests are handled by controllers
- Recognisable HTTP flow
- Examples:
  - Spring MVC
  - Struts
  - Grails
  - Play


# Component based

- Higher level of abstraction
- Build views with UI-components
- Request-response flow is less obvious
- Examples:
  - JSF
  - Wicket
  - Vaadin
  - GWT

# JavaScript

- Client-side:
  - Angular, Ember, Backbone, …
  - Often integrated with Java backend

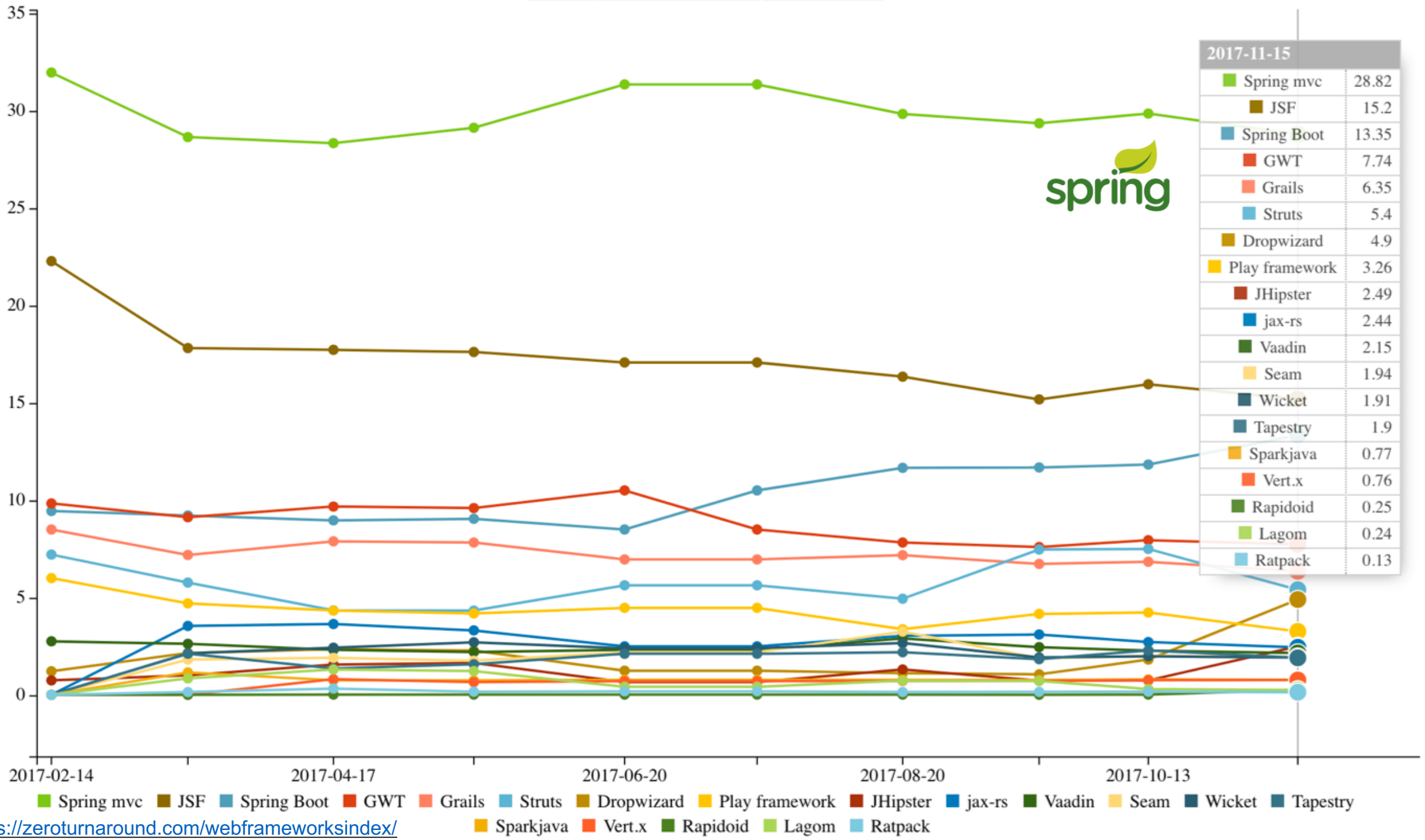

- Server side: Node.js
  - Express, Meteor, Sails, …


- Often used in compination
- → Web 4

# Best ?

- ✓Make a list of functions that are important to you
- ✓Choose 3 or 4 frameworks and make a small web app
- ✓Compare each framework
- ✓Make your decision...


http://www.slideshare.net/mraible/comparing-jvm-web-frameworks-february-2014

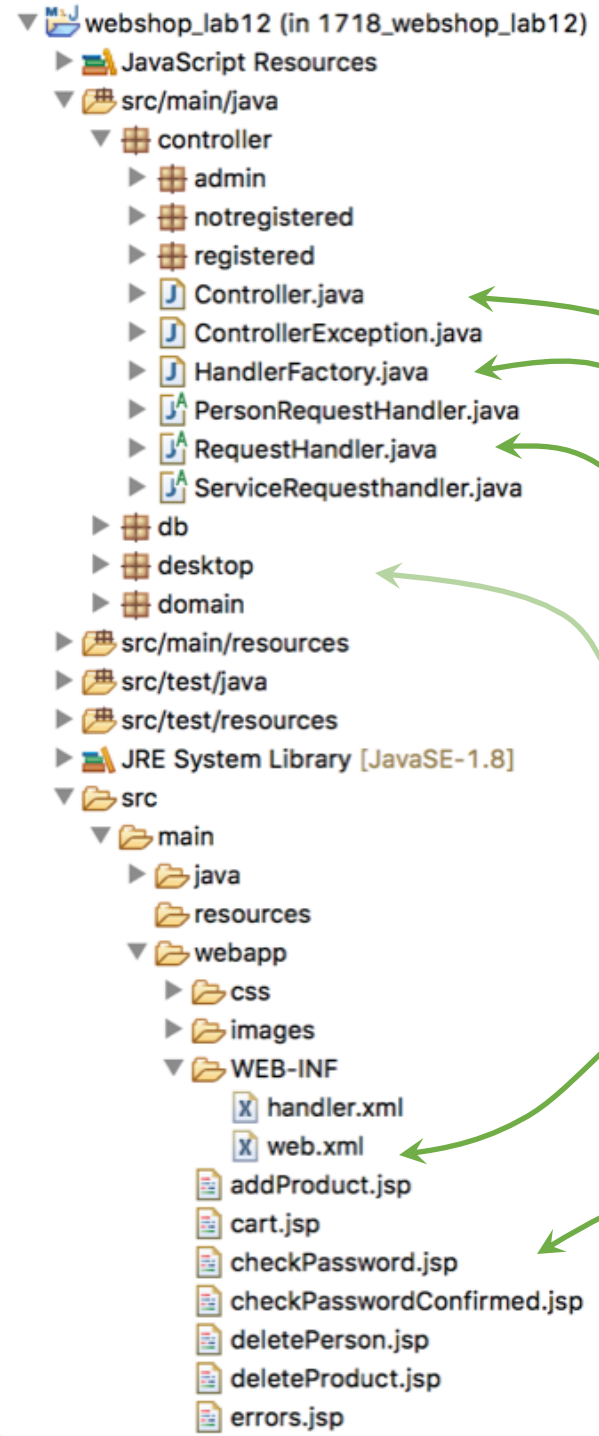



https://zeroturnaround.com/webframeworksindex/


Always Keep It
OPEN

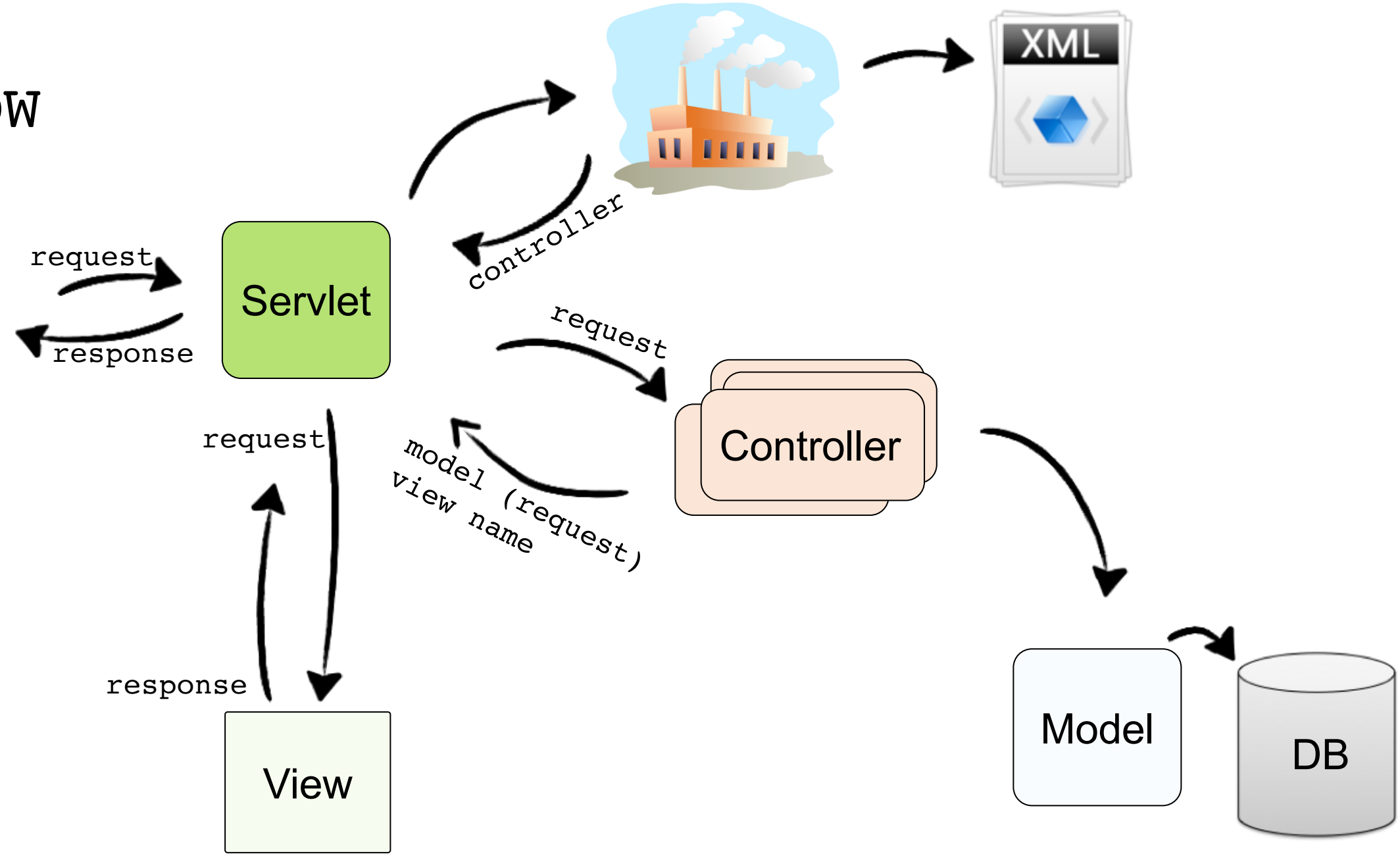
# Spring

# Recap Java Web Applications

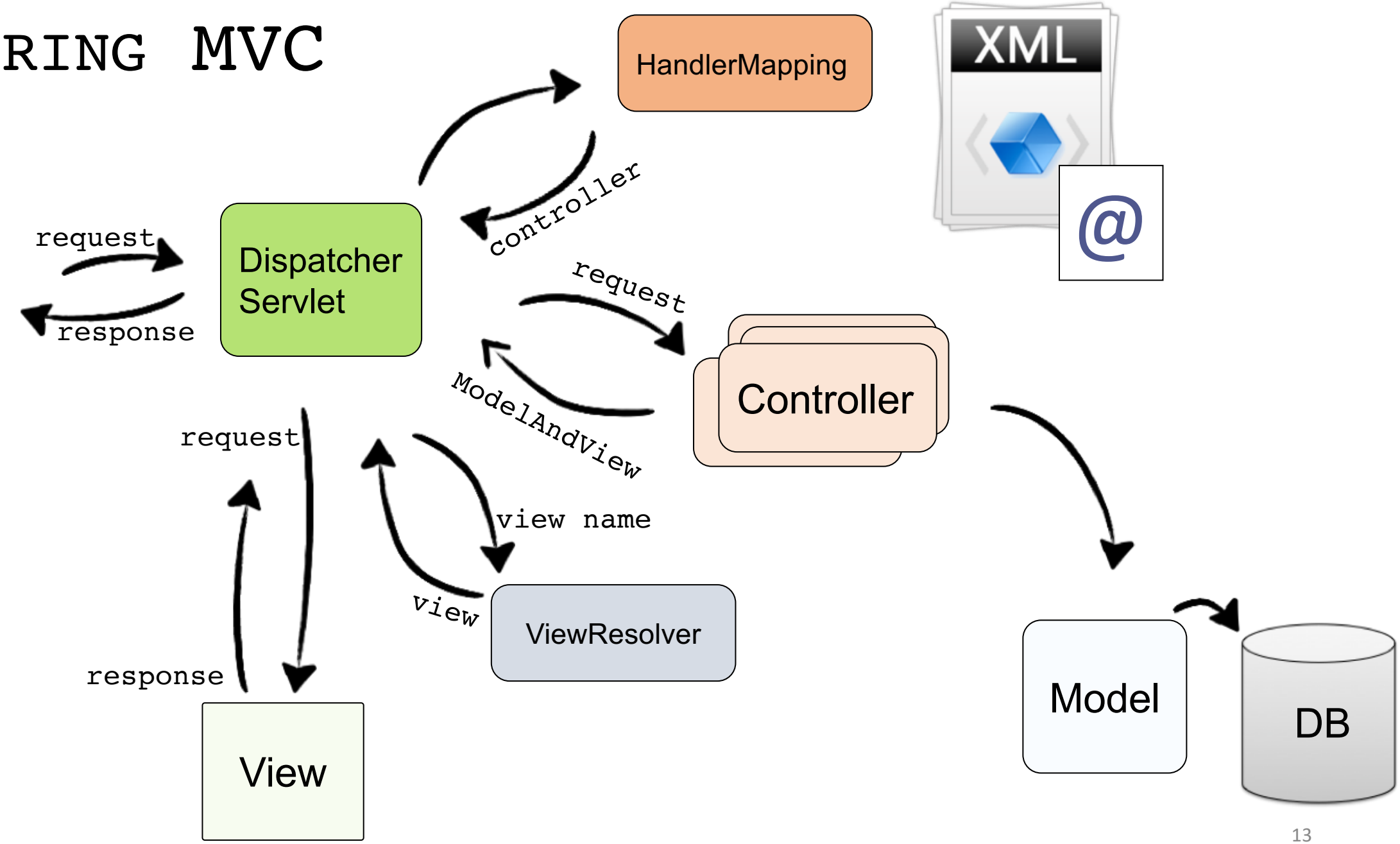



- Front controller pattern
  - Front Controller
  - Factory
  - Handlers
- Domain
- Config
  - web.xml
  - ServletContext, …
- Jsp

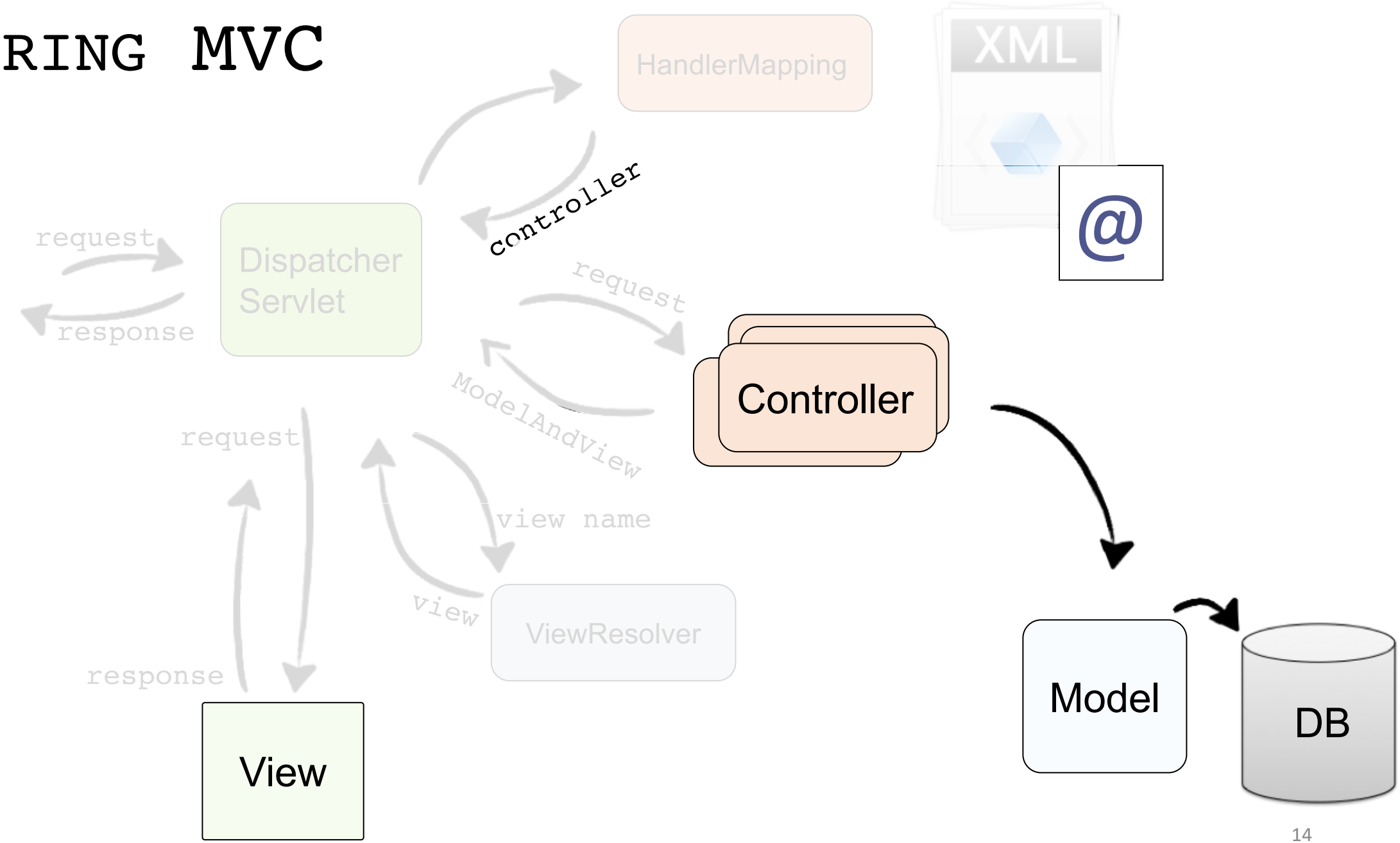
# Flow


XML
controller
request
response
Servlet
request
Controller
request
model (request)
view name
response
View
Model
DB

# Spring MVC


HandlerMapping
XML
@
controller
request
response
Dispatcher Servlet
request
Controller
ModelAndView
request
view name
view
ViewResolver
response
View
Model
DB

# Spring MVC


HandlerMapping
XML
@
controller
request
Dispatcher Servlet
response
request
Controller
ModelAndView
request
view name
view
ViewResolver
response
View
Model
DB

# DispatcherServlet

Dispatcher Servlet

- Part of Spring framework

- **Front controller**:
  1. Asks *Handler mapping* for controller
  2. Passes request to *controller*
  3. Asks *ViewResolver* for view
  4. Passes model to *view*

`org.springframework.web.servlet.`**`DispatcherServlet`**`.class`

# HandlerMapping

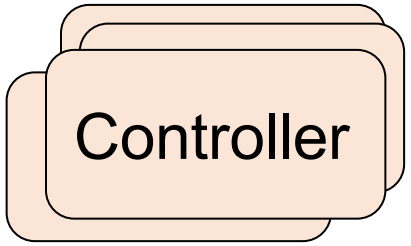
HandlerMapping

- Part of Spring framework

- **Retrieves controller** for given url
  - different ways of mapping → different Spring classes
  - you decide which one to use

```
….SimpleUrlHandlerMapping.class
….RequestMappingHandlerMapping.class
….BeanNameUrlHandlerMapping.class
```

# Controllers




- Written by you:
  - POJO
  - Performs action on model
  - Returns model and view

```
@Controller
@RequestMapping(value = "/country")
public class CountryController {
    private final TourismService service …;

    @RequestMapping(method = RequestMethod.GET)
    public ModelAndView getCountries() {
        return new ModelAndView("countries", "countries", service.getCountries());
    }
}
```

# ViewResolver

ViewResolver

- Part of Spring framework

- **Retrieves view** with given view name
  - different ways of resolving → different Spring classes
  - you decide which one to use

```
….InternalResourceViewResolver.class

….XmlViewResolver.class

….TilesViewResolver.class

…
```

# View

View

- Written by you:
  - JSP as you know it
  - extra tags available

```
<%@ page language="java" contentType="text/html; charset=UTF-8” pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE html>
<html>
<jsp:include page="head.jsp"><jsp:param value="User Overview" name="title" /></jsp:include>
<body>
   <div id="container">
      <main>
         <table>
            <tr>
               <th>E-mail</th> <th>First Name</th> <th>Last Name</th>
            </tr>
            <c:forEach var="user" items="${personList}">
```

# How does it fit together?

1. Add Spring **dependencies**

2. **Register** DispatcherServlet
3. **Configure** DispatcherServlet

4. Write **domain** classes
5. Write (JSP) **view** pages
6. Write Spring **controllers**

# Spring Dependencies

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns=…>
    <modelVersion>4.0.0</modelVersion>

    <groupId>be.ucll.demo</groupId>
    <artifactId>product</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.0.3.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
```

# Register DispatcherServlet

## how your learned it in web 2

... bit old fashioned

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" ...
    …
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>
           org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>*.htm</url-pattern>
    </servlet-mapping>
    …
</web-app>
```

# Register DispatcherServlet

## The modern way: Java configuration

Spring class

```
public class WebInitializer
                    extends AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[]{ ApplicationConfig.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[]{ DispatcherServletConfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] {"*.htm"};
    }
}
```

Written by you

# Configure Servlet

## To Be Continued…

```
public class ApplicationConfig {

    @Bean(name = "service")
    public ProductService getService() {
        return new ProductService("MEMORY");

    }
}
```

```
@Configuration
@ComponentScan("be.ucll.product.web")
@EnableWebMvc
public class DispatcherServletConfig implements WebMvcConfigurer {

    @Bean
    public InternalResourceViewResolver getViewResolver() {
        InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
        viewResolver.setViewClass(JstlView.class);
        viewResolver.setPrefix("/WEB-INF/jsp/");
        viewResolver.setSuffix(".jsp");
        return viewResolver;
    }
}
```

# Summary

- **SPRING** provides:
  - `DispatcherServlet.class`
  - `SimpleUrlHandlerMapping.class`
  - `RequestMappingHandlerMapping.class`
  - `BeanNameUrlHandlerMapping.class`
  - `InternalResourceViewResolver.class`
  - `XmlViewResolver.class`
  - `TilesViewResolver.class`
  - And much more…

- **YOU** write:
  - Domain classes
  - Controllers
  - (JSP) views
  - Configuration files

?