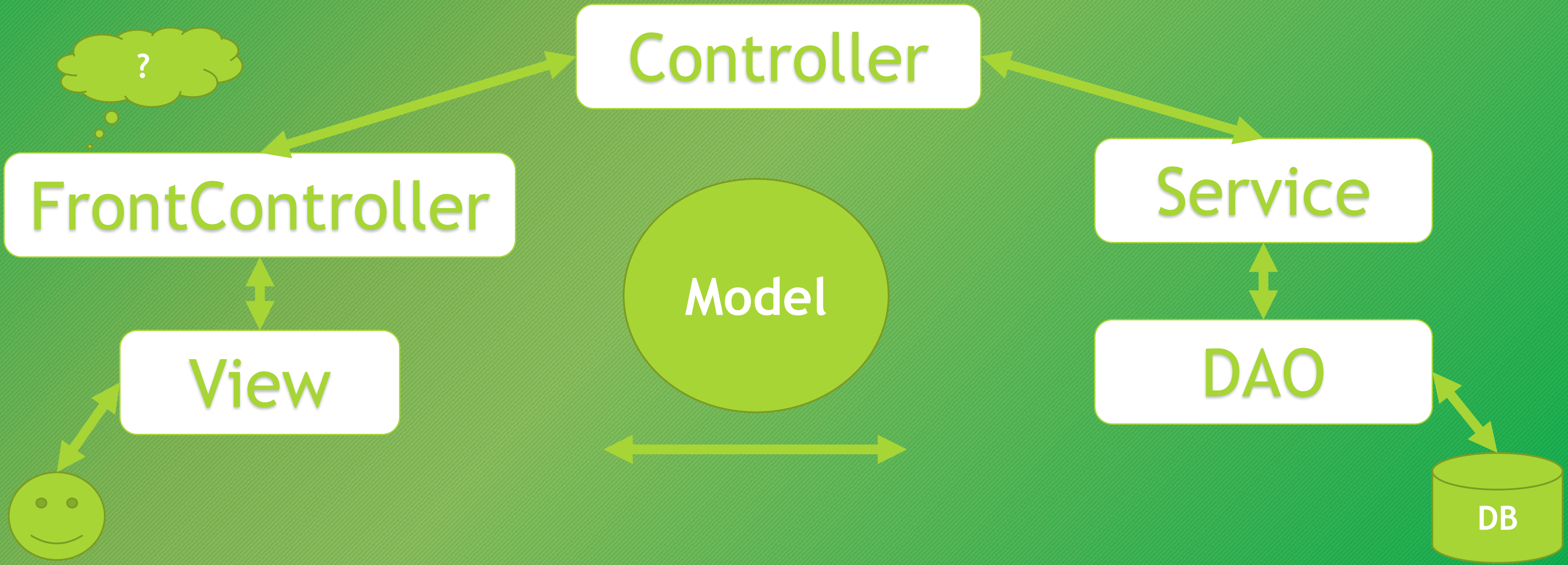
Spring: MVC

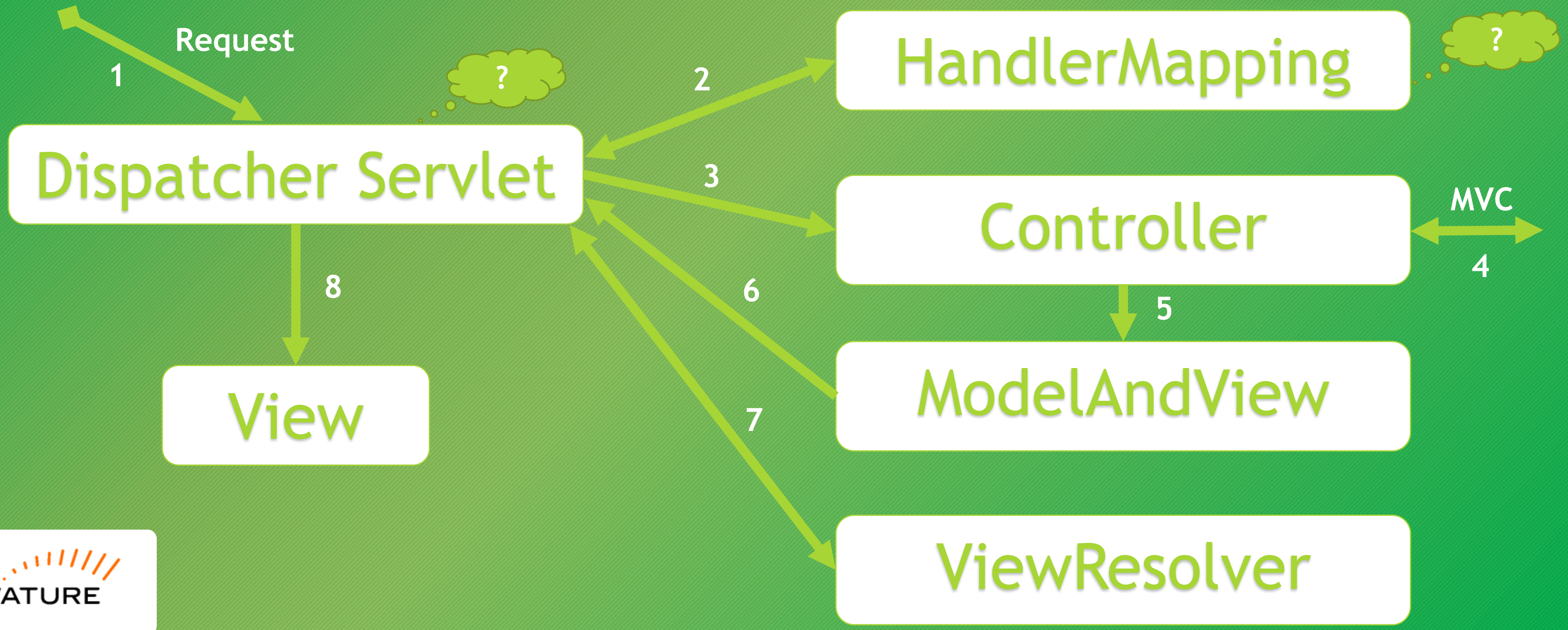
Peter Alagna Jr.

MVC

Spring solution for web applications.



Control Flow



Annotations

- **@Controller.**
  - Tells Spring that the class is a Controller.
    - It is another **Stereotype** annotation.
  - This means is going to follow the control flow.
  - It supports **RESTful** operations.
- **@RequestMapping.**
  - Tells Spring the **URI** that needs to be mapped.
  - @RequestMapping(value = "/home", method = RequestMethod.*GET)*

HandlerMapping

- Acts like a **RequestHelper** in a **FrontController**.
- Tells the **DispatcherServlet** which **Controller** to use.
- Provides the behavior for **@RequestMapping**.
- Spring < 3.2: ***DefaultAnnotationHandlerMapping*** is the implementation used.
- Spring > 3.2: ***RequestMappingHandlerMapping*** is the implementation used.

InternalResourceViewResolver

- Simplifies the **String** representing the **view** that **Controllers** return.
  - **Prefix:** /something/
  - **Suffix:** .html
  - **Return:** “home” -> “/something/home.html”.
- Configured within **dispatcher-servlet.xml**.

ContextLoaderListener

- Ties the lifecycle of the **ApplicationContext** with the **ServletContext**.
- Automates the creation of the **ApplicationContext** so that the **container** (tomcat) starts it for us.
- Configured inside **web.xml**.

Configuration

- Files needed inside **WEB-INF**:
  - **applicationContext.xml**
  - **web.xml**
  - **dispatcher-servlet.xml**
- In **applicationContext.xml**:
  - <mvc: annotation-driven />
- In **web.xml**:
  - Location of applicationContext.xml (using context-param).
  - Add ContextLoaderListener with a <listener> tag.
  - Add new servlet (dispatcher).
  - Add servlet-mapping for dispatcher.
- In **dispatcher-servlet.xml**:
  - Add **viewResolver** bean that uses **InternalResourceViewResolver** class, with its suffix and prefix.

Spring REST Introduction

In **MVC**, we can still use the **HttpServlet** request and response objects, however, we don't really need them.

- Instead of accessing parameters from the **request** object.
  - Use **@RequestBody** in your parameters: Spring will **unmarshal** the parameters coming in **JSON** format to your **POJOs**.
- Instead of writing a **POJO** with **Jackson** using the **PrintWriter** within the **response** object.
  - Use **@ResponseBody** in your return type: Spring will **marshal** your **POJOs** into **JSON** format.

Materials

- Spring MVC: https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html