Spring overview

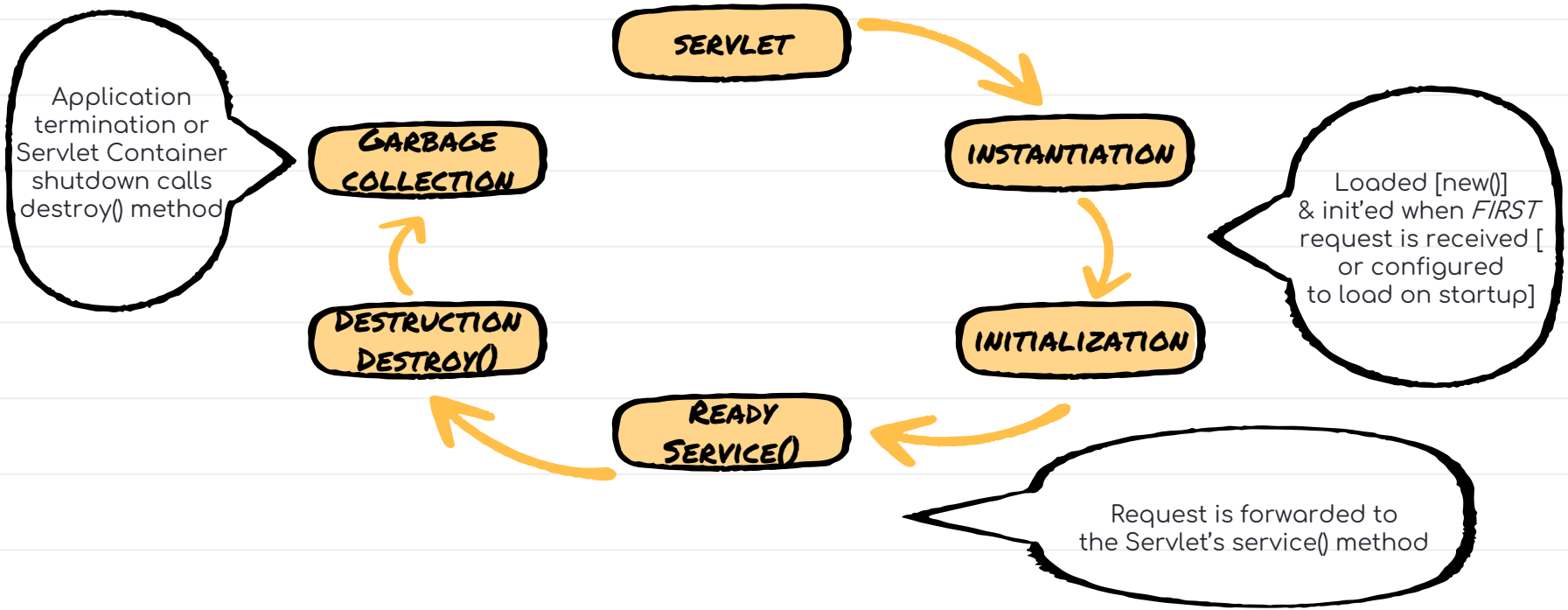
Teaching Faculty: Dr. Muhyieddin Al-Tarawneh

Prepared by Muhyieddin **AL-TARAWNEH**, Umur **INAN**

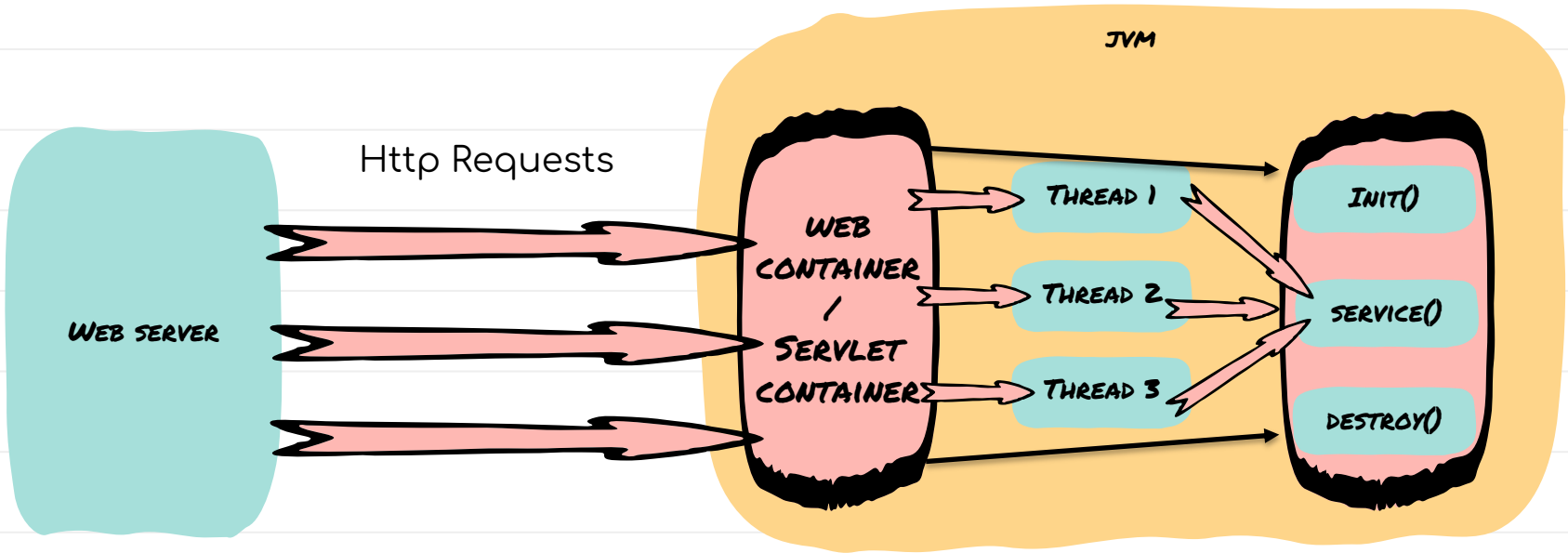
SERVLETS

- A servlet is server-side java code that can handle http requests and return dynamic content.
- Servlets are managed by a *servlet engine* or *container*.
- Each request that comes in results in the spawning of a new thread that runs a servlet.

Servlet Lifecycle



SERVLET


JVM
Http Requests
WEB SERVER
WEB CONTAINER / SERVLET CONTAINER
THREAD 1
THREAD 2
THREAD 3
INIT()
SERVICE()
DESTROY()

Three Names of a Servlet

"servlet-class" is the Java name of the class.

```
<servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>com.umur.hello</servlet-class>
</servlet>
```

"servlet-name" is the internal name of the servlet.

```
servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

"url-pattern" is the way the servlet is specified in the Browser.

Architecture

- **Architecture** is an abstract plan that can include design patterns, modules, and their interactions.

In this course we will focus on

- Architecture **Implementation or Realization**

which incorporates

- **Frameworks** - *architected* "physical" structures on which you build your application.

specifically we will use

- The **Spring Framework**, an **Enterprise Application** Development environment for building large scale enterprise applications and **React** for Client-side.

Web Application Architecture

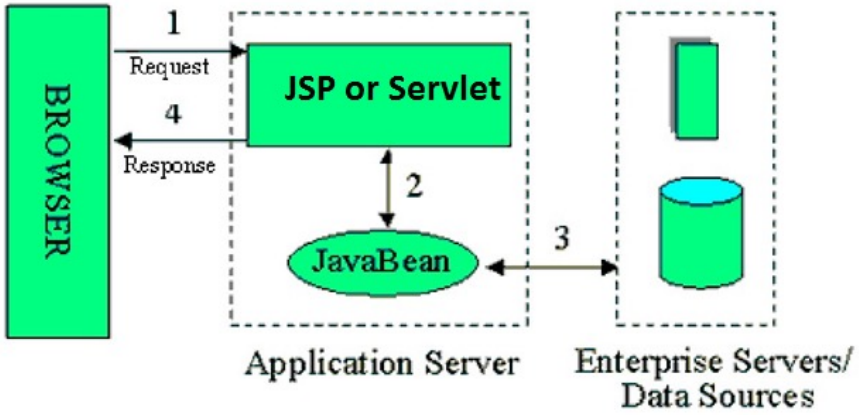
This course is concerned with the realization of scalable web applications as defined by a Web Application Architecture.

A **Web Application Architecture**
is a **part** of an **Enterprise Architecture**

A Scalable Web Application is an Enterprise Application...

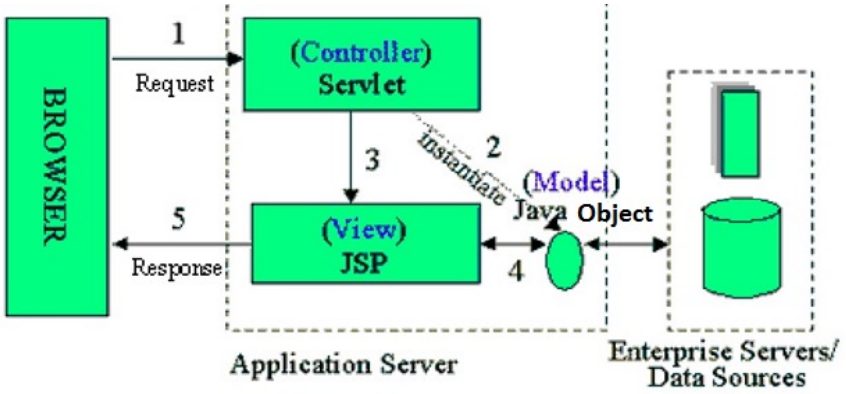
An application - that is large & complex – well beyond the individual or small business use case.

Model 1 Architecture



Model 1 mixes view and business logic inside each handling servlet (or JSP). This makes it more difficult to change the view independently of that logic, and difficult to change business logic without changing the view.

Model 2 Architecture

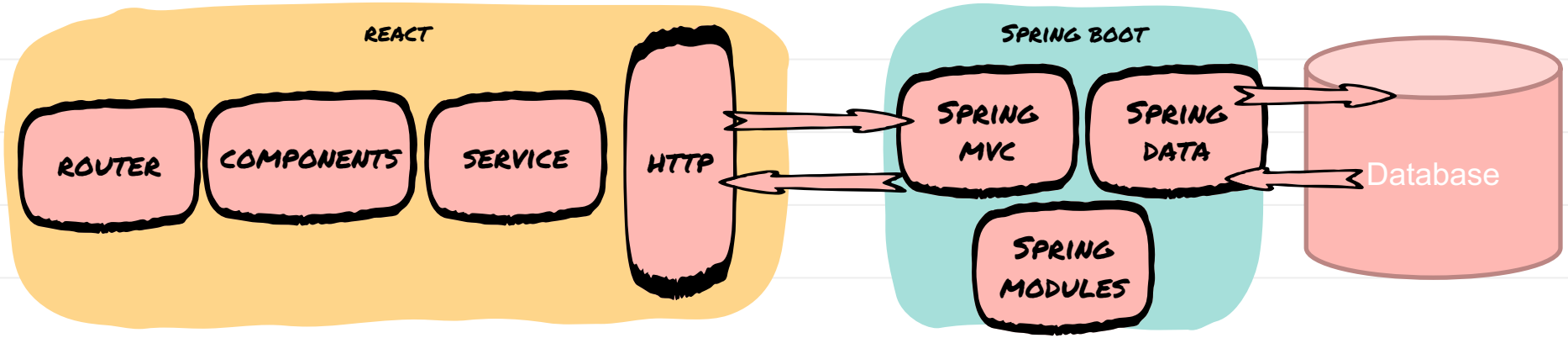


Model II cleanly separates business data and logic from the view, and the two are connected through a controller servlet. The model allows for multiple controllers/servlets, [e.g., one per GET/POST pair].

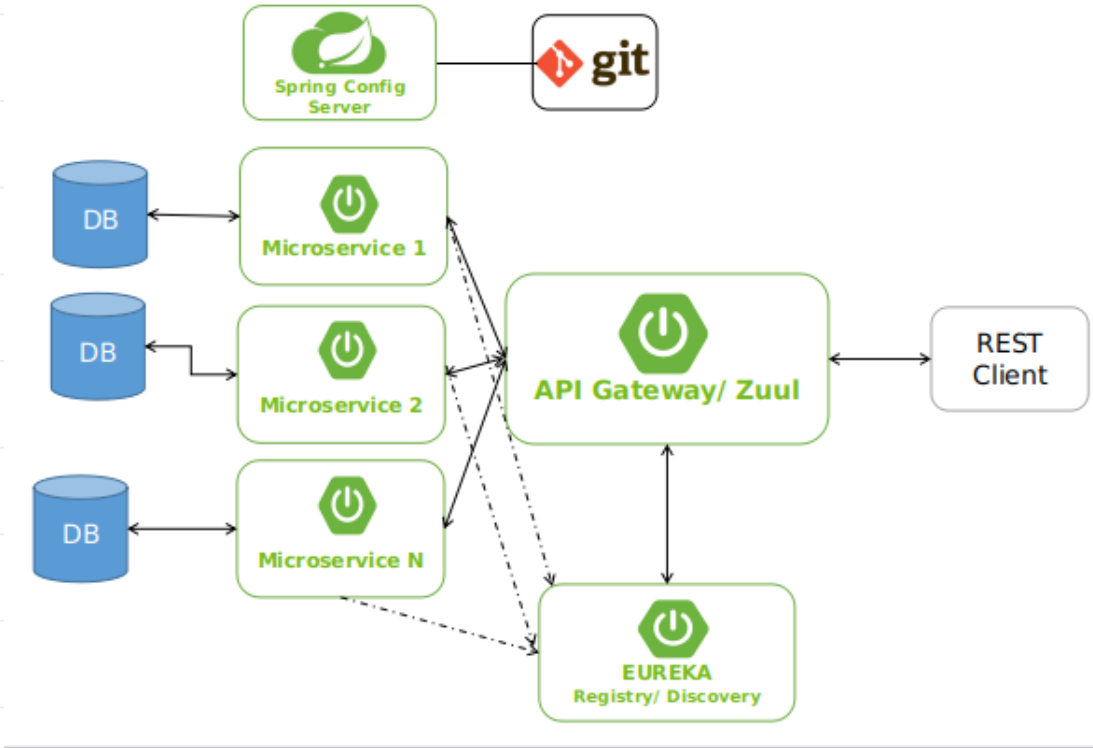
Typical MVC Framework implementations have just **one controller servlet** which centralizes common tasks.

Model 3 Architecture built in this course

- This course emphasizes on building a modern architecture separating the view from the server-side.
- This is more efficient, scalable and widely applied in today's industrial market for several reasons explained within course.



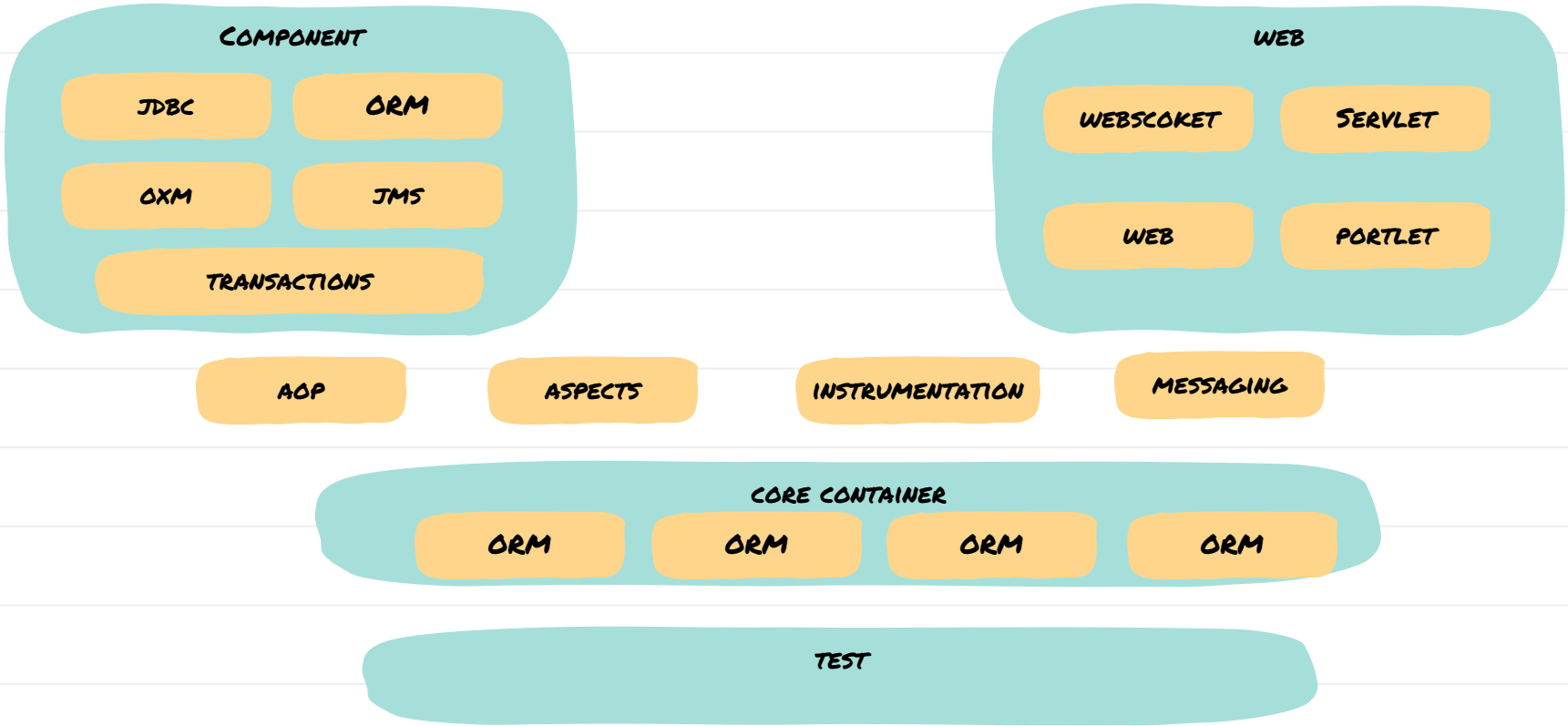
Microservices Architecture



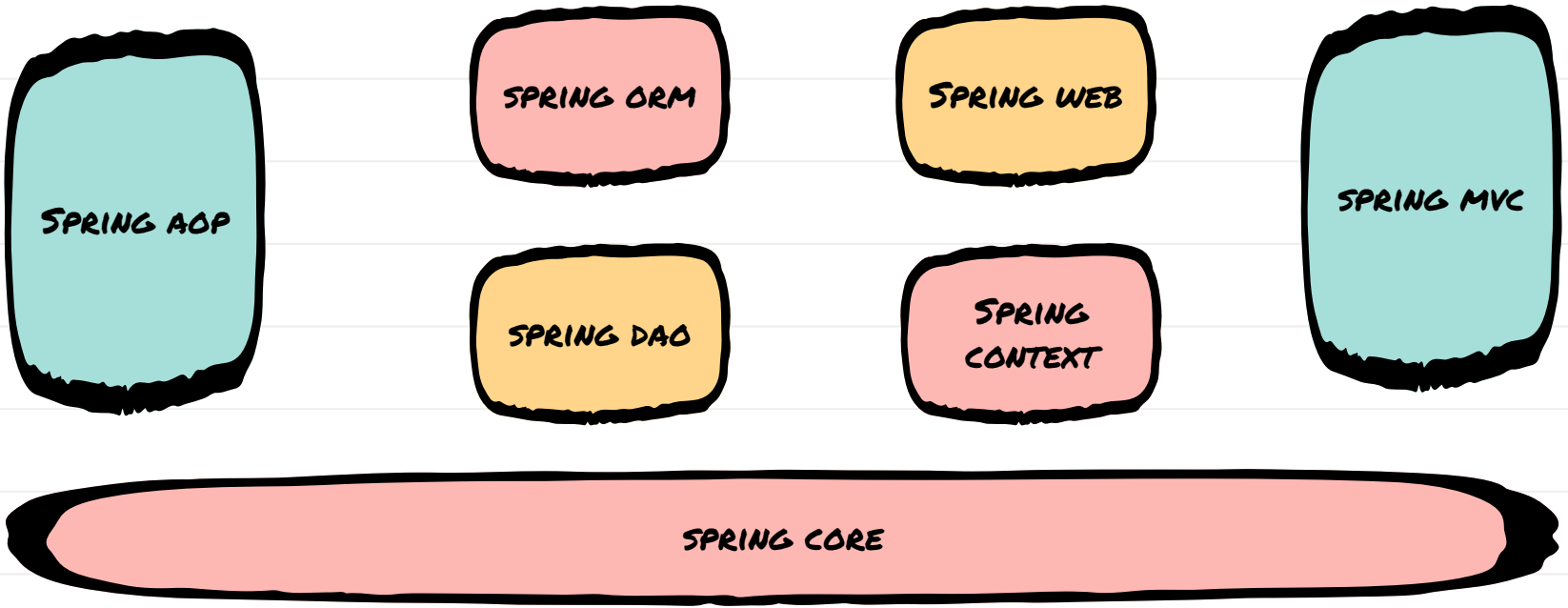
Main Point

When you use a seperated server-side according to a Model 3 architecture, there is a servlet that acts as a controller (**process of knowing**) that sets attribute values based on computations and results from a business model (**knower**), then dispatches the request to the client-side (**known**). The API then delivers the attribute values to the designated places being sent to the browser.

Spring framework



Spring architecture



Inversion of Control

- Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a **container** or framework.

- **Inversion of Control** (IoC or IOC) describes a system that follows the Hollywood Principle.

- **Hollywood Principle** states, "Don't Call Us, We'll Call You."

Inversion of Control

- 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.

Inversion of Control

Objects do not create other objects that they depend on.

IoC is implemented using **Dependency Injection**(DI).

Main Point

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

Dependency injection

- Dependency injection is a pattern that is used to implement IoC, where the control being inverted is setting an object's dependencies.

Dependency injection

- 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[by Type]

```
@Autowired
ProductService productService;
```

Dependency injection examples

- Setter based [by Name]

```
ProductService productService;

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

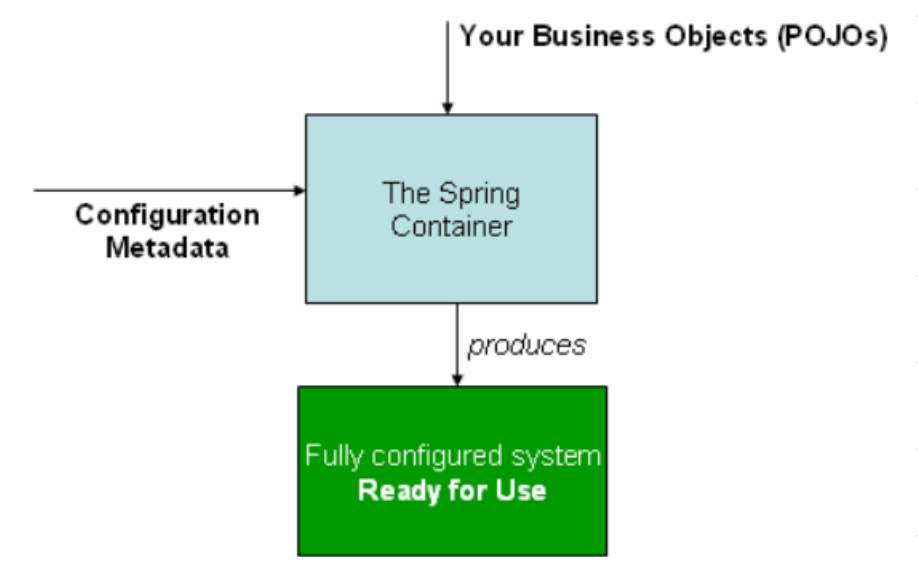
Dependency injection examples

- Constructor based:

```
ProductService productService;

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

The Spring container



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.

JavaBean vs POJO vs Spring Bean

- JavaBean
  - Adhere to Sun's JavaBeans specification.
  - Implements Serializable interface.
  - Must have default constructor, setters & getters.
  - Reusable Java classes for visual application composition.

JavaBean vs POJO vs Spring Bean

- POJO
  - ‘Fancy’ way to describe ordinary Java Objects
  - Doesn’t require a framework
  - Doesn’t require an application server environment
  - Simpler, lightweight compared to ‘heavyweight’ EJBs

JavaBean vs POJO vs Spring Bean

- Spring Bean
  - Spring managed - configured, instantiated and injected.

Java object can be a JavaBean, a POJO and a Spring bean all at the same time.

Create an Object of AClass

```
public class AClass {
 private BClass b;
 public AClass(BClass b)  {
    this.b= b;
  }

}
```

```
public class BClass {
   private CClass c;
   public BClass(CClass
    c){
    this.c=c;
   }
}
```

```
public class CClass {
   private DClass d;

   public CClass(DClass d){
     this.d= d;
   }
}
```

```
public class DClass {
   private EClass e;
   public DClass (EClass e){
     this.e=e;
   }
}
```

```
public class EClass {

}
```

Better Method?

```
public class Main {
    EClass e = new EClass();
    DClass d = new DClass(e);
    CClass c = new CClass(d);
    BClass b = new BClass(c);

    AClass a = new AClass(b);
}
```

Dependency injection

- Ask to the container to give an object of the class by injecting all dependencies.

```
@Autowired
    private AClass a;
```

How does container find dependencies ???

We need to put our dependencies (classes) to the container.

The container should be aware of our classes !!!

Bean

- The objects that form the backbone of your application and that are managed by the Spring IoC container are called beans.
- A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.
- A bean is simply one of many objects in your application

@Configuration

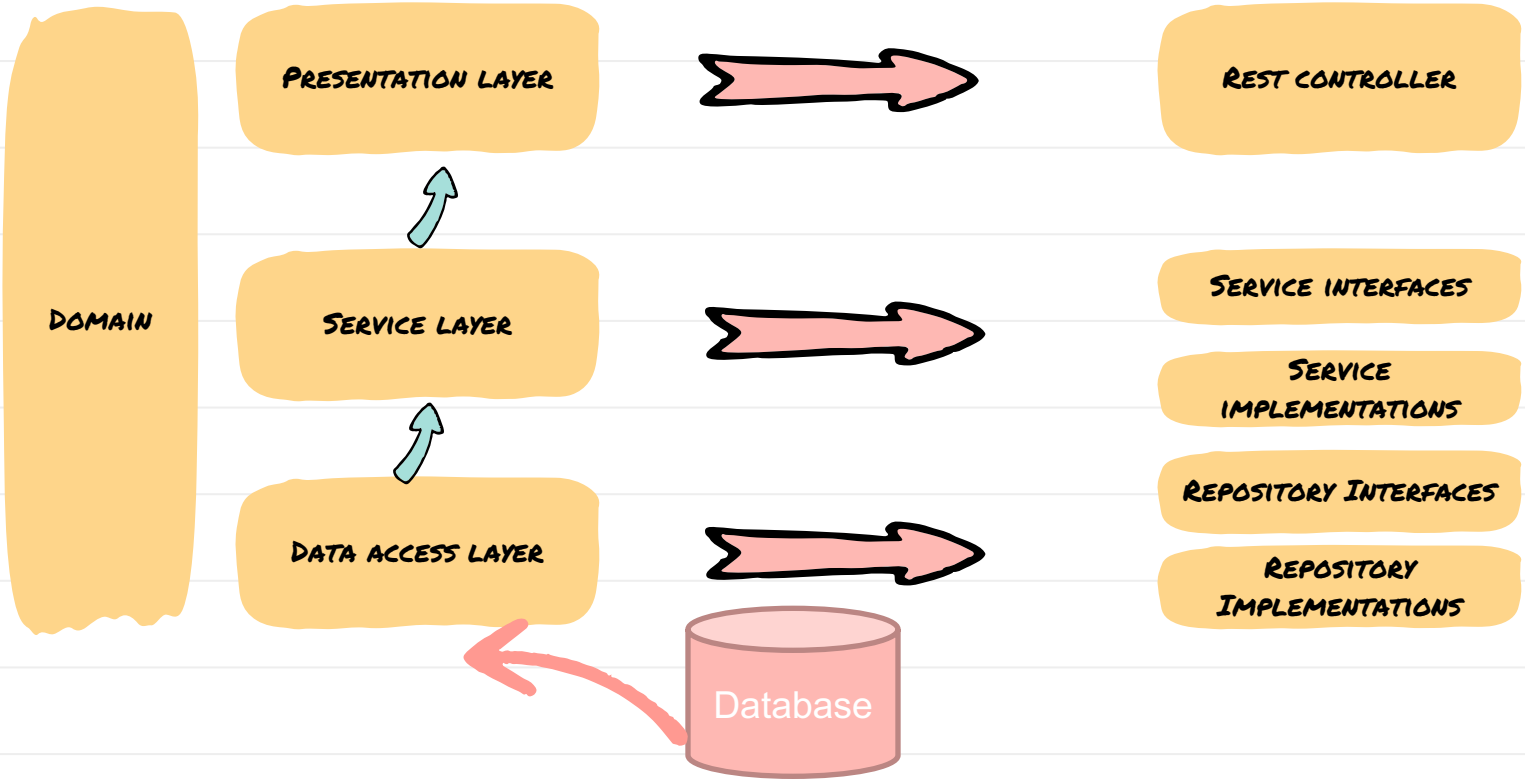
- It is a class-level annotation indicating that an object is a source of bean definitions.

- @Configuration classes declare beans via public @Bean annotated methods.

N-TIER ARCHITECTURE

- It divides an application into logical layers and physical tiers.
- Layers are a way to separate responsibilities and manage dependencies.
- Each layer has a **specific** responsibility.
- A higher layer can use services in a lower layer, but not the other way around

N-TIER ARCHITECTURE



Stereotype annotations

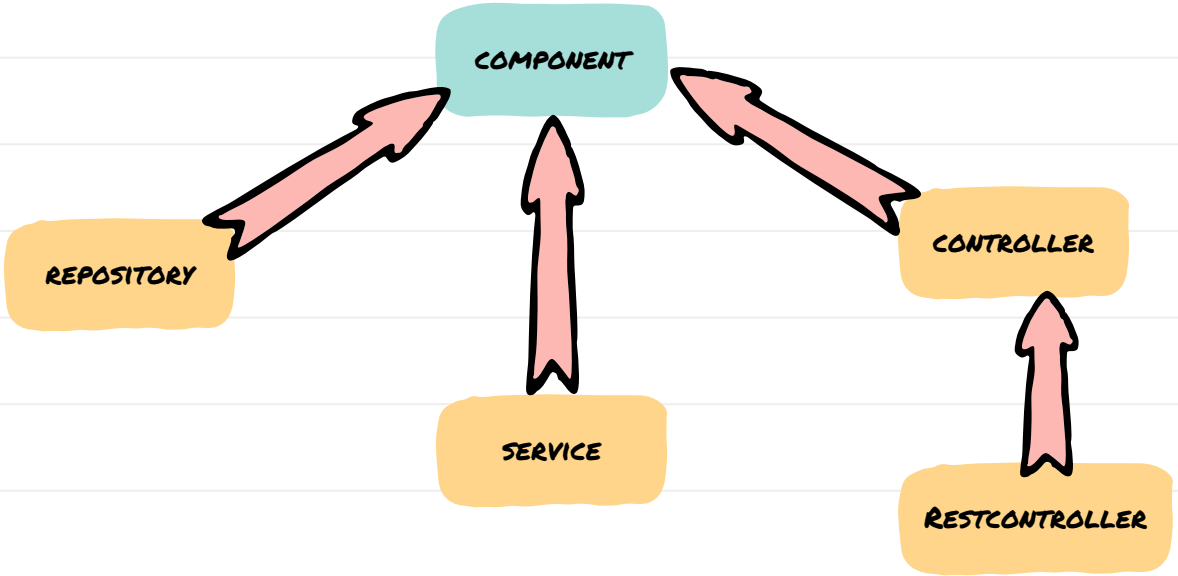
- There are some Stereotype meta-annotations which is derived from **@Component** those are:

- **@Controller**: Which is used to create Spring beans at the controller layer.

- **@Service**: Used to create Spring beans at the Service layer.

- **@Repository**: Which is used to create Spring beans for the repositories at the DAO layer.

Stereotype annotations



@CONTROLLER

- is used to indicate the class is a Spring controller.

@RestController

- is a specialized version of the controller.

- It includes the @Controller and @ResponseBody annotations, and as a result, simplifies the controller implementation

@Service

- It is a stereotype for the service layer.

- The @Service marks a Java class that performs some service, such as execute business logic, perform calculations and call external APIs

@Repository

- This annotation is used on Java classes that directly access the database.

- Its job is to catch persistence specific exceptions and rethrow them as one of Spring's unified unchecked exception.

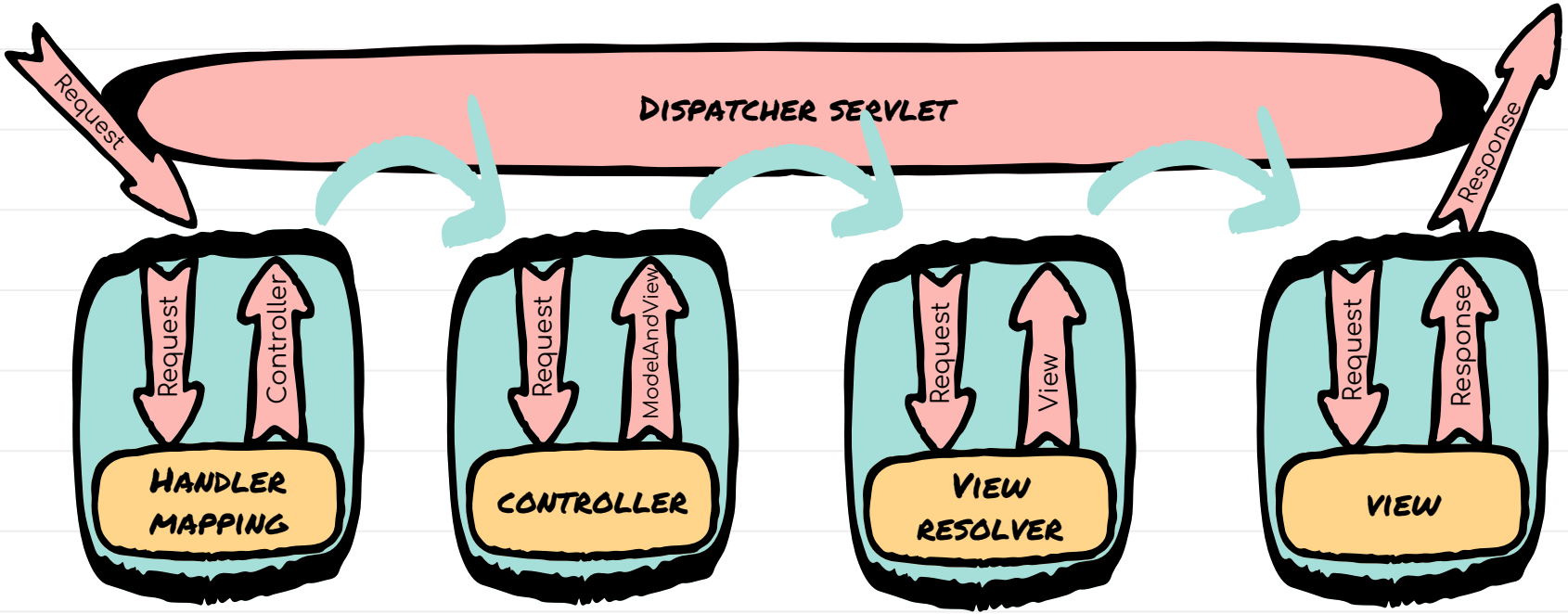
Spring MVC Flow

- The *DispatcherServlet* first receives the request.
- The *DispatcherServlet* consults the *HandlerMapping* and invokes the *Controller* associated with the request.
- The *Controller* process the request by calling the appropriate service methods and returns a *ModeAndView* object to the *DispatcherServlet*. The *ModeAndView* object contains the model data and the view name.

Spring MVC Flow

- The *DispatcherServlet* sends the view name to a *ViewResolver* to find the actual *View* to invoke.
- Now the *DispatcherServlet* will pass the model object to the *View* to render the result.
- The *View* with the help of the model data will render the result back to the user.

Spring MVC Flow



Prepared by Muhyieddin **AL-TARAWNEH**, Umur **INAN**

What is Spring Boot?

- Spring Boot is a project built on the top of the Spring framework.
- It provides a simpler and faster way to set up, configure, and run both simple and web-based applications.

Spring Boot Features

- **Auto-configuration:** It sets up your application based on the surrounding environment, as well as hints what the developers provide.
- **Standalone:** Literally, it's completely standalone. Hence, you don't need to deploy your application to a web server or any special environment. Your only task is to click on the button or give out the run command, and it will start.
- **Opinionated:** This means that the framework chooses how to things for itself. In the most cases, the developers use the same most popular libraries. All that the Spring Boot does is that it loads and configures them in the most standard way. Hence, the developers don't need to spend a lot of time to configure up the same thing over and over again.

Spring Boot Features

- Spring Boot auto-configuration attempts to automatically configure your Spring application based on the jar dependencies that you have added.

- Guess and configure beans that you are likely to need.

Embedded Server

- With Embedded server, you can run a web application as a normal Java application.
- Embedded server implies that our deployable unit contains the binaries for the server (example, tomcat.jar).
- spring-boot-starter-web: includes a dependency on starter tomcat.

Pom.xml

- Spring Boot provides several “Starters” that let you add jars to your class path.
- spring-boot-starter-parent
  - inherits dependency management from spring-boot-dependencies
  - Override property e.g., java.version
  - default configuration for plugins e.g. maven-jar-plugin
- Other starters
  - spring-boot-starter-web
  - spring-boot-starter-thymeleaf
  - spring-boot-starter-test

Main points

- Frameworks make development easier and more effective by providing a secure and reliable foundation on which to build upon.
- The simplest form of awareness, Transcendental Consciousness, provides a strong foundation for a rewarding and successful life.
- 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.