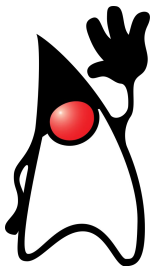
Desenvolvimento com Frameworks e Componentes

Michel Vasconcelos
michel.vasconcelos@gmail.com



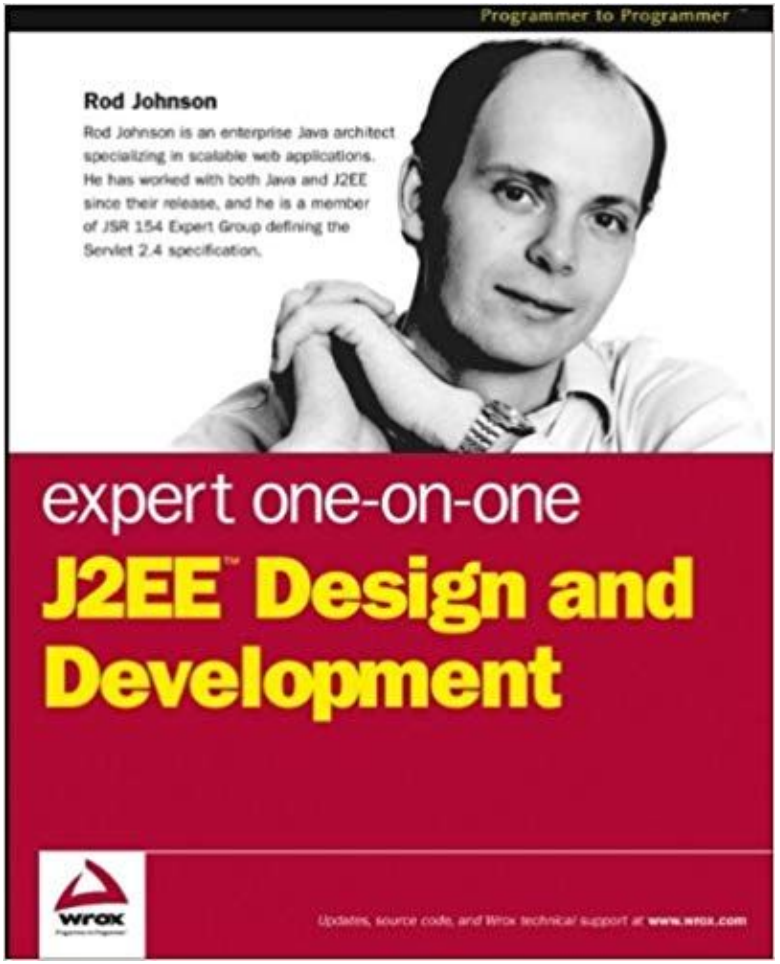
Previously on Developing with Frameworks and Components @UNI7...

Spring

Cenário

Proposta

- Framework complementar a especificação Java EE
- POJOs vs EJBs
- IoC, AOP, etc



Atualmente

SPRING BOOT

Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible.

SPRING FRAMEWORK

Provides core support for dependency injection, transaction management, web apps, data access, messaging and more.

SPRING CLOUD

Provides a set of tools for common patterns in distributed systems. Useful for building and deploying microservices.

SPRING KAFKA

Provides Familiar Spring Abstractions for Apache Kafka.

SPRING SECURITY

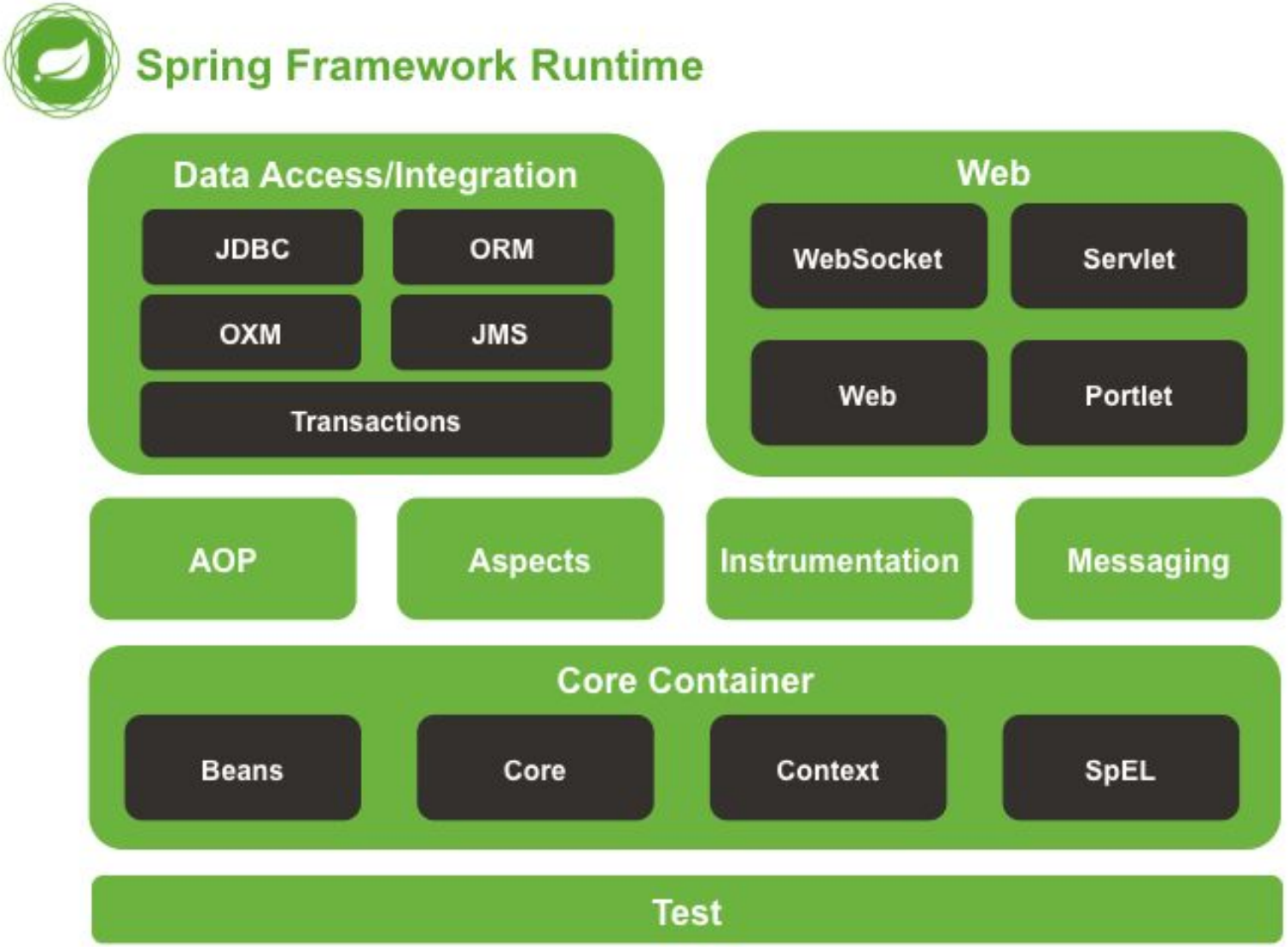
Protects your application with comprehensive and extensible authentication and authorization support.

SPRING BATCH

Simplifies and optimizes the work of processing high-volume batch operations.

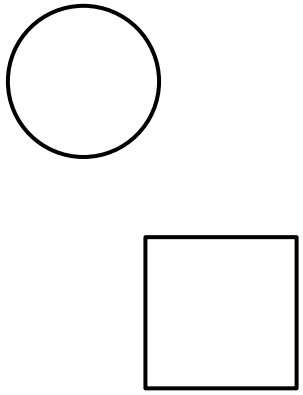
SPRING FRAMEWORK

Provides core support for dependency injection, transaction management, web apps, data access, messaging and more.



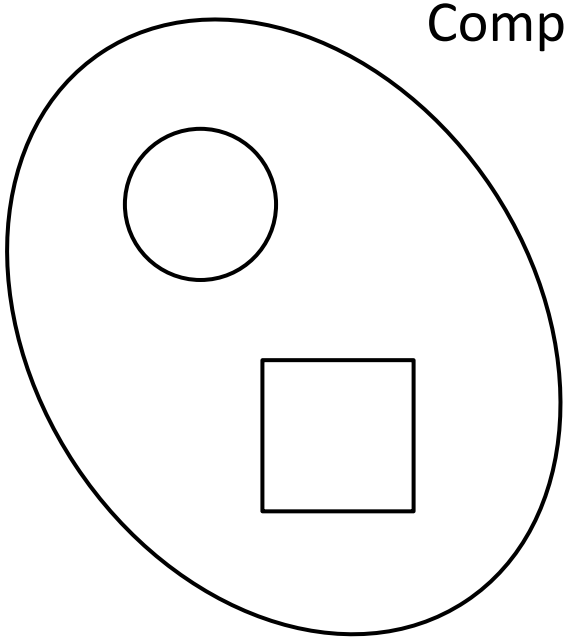
Fonte: https://docs.spring.io/spring/docs/4.3.23.RELEASE/spring-framework-reference/htmlsingle/#spring-introduction

Beans

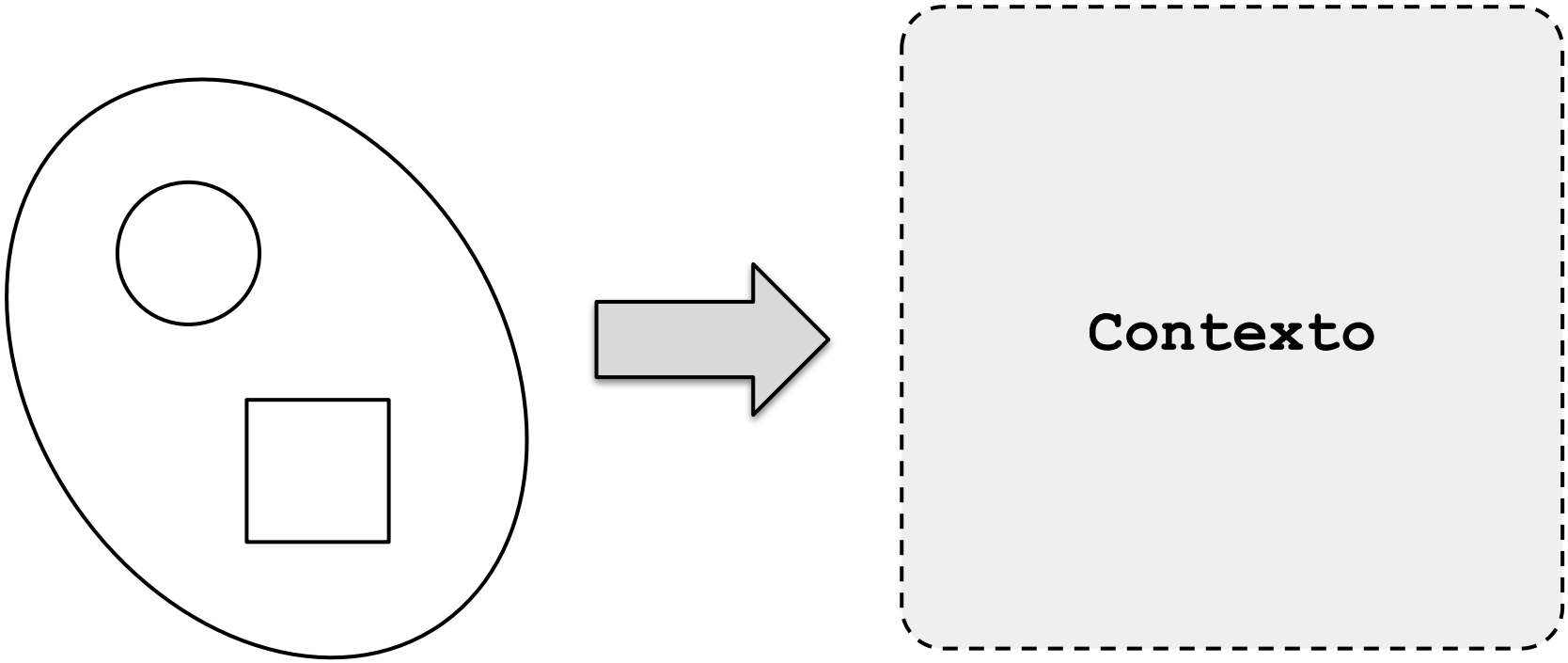


Registro

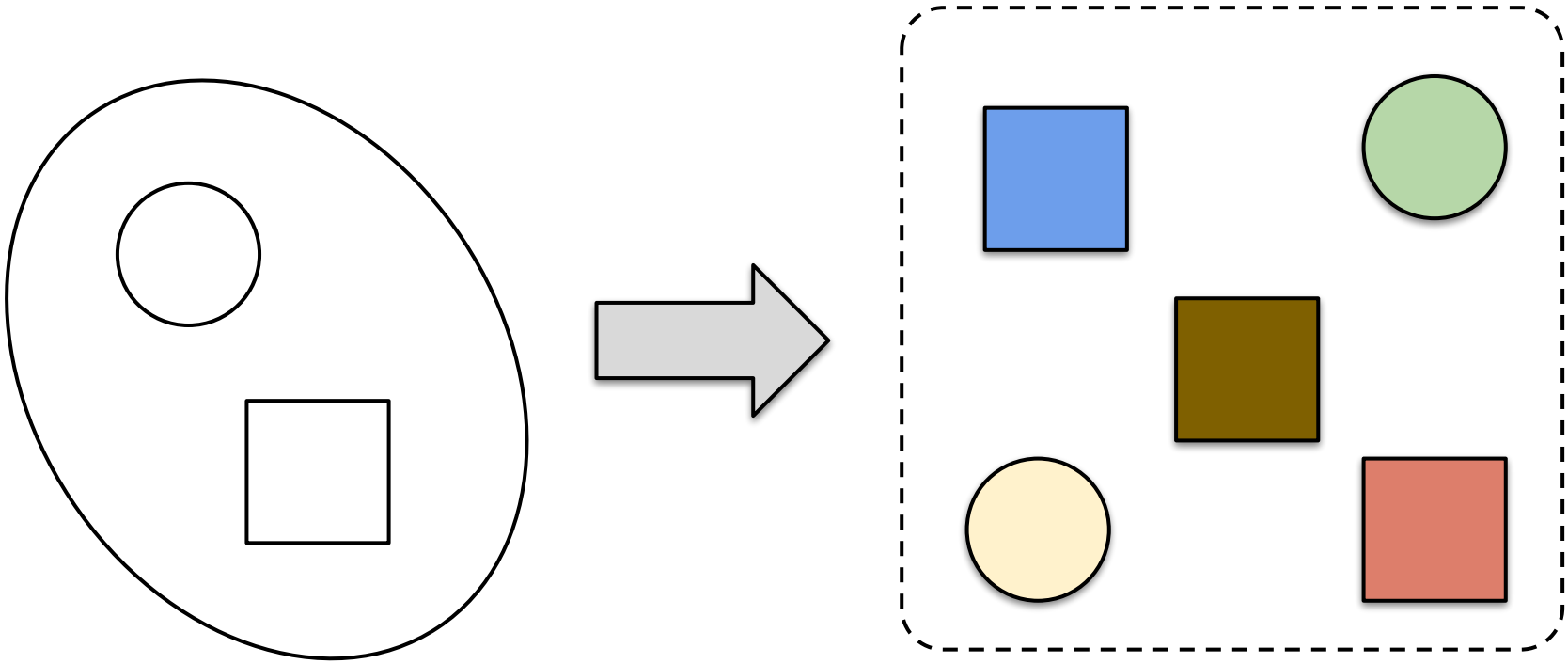
XML Definition
Component Scanning



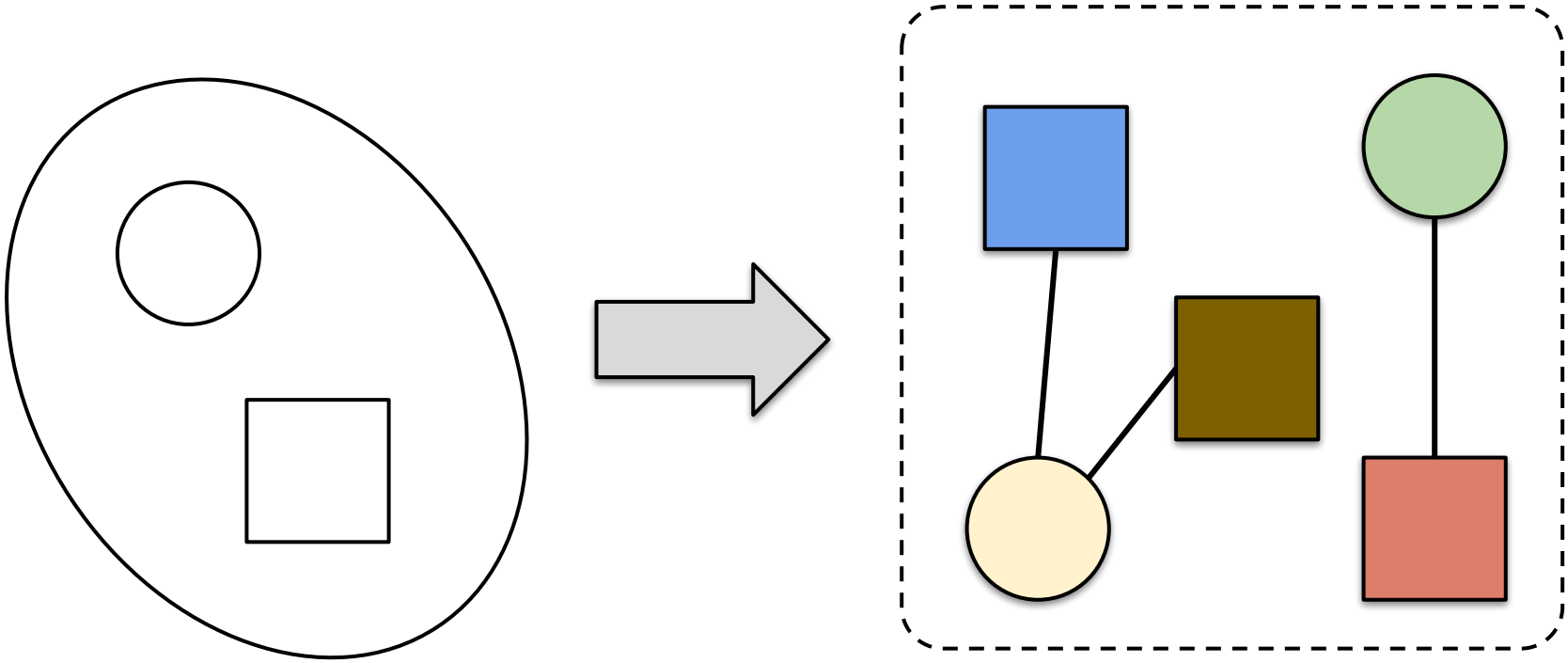
Container



Montagem



Wiring (Ligação)



Contexto

`org.springframework.context.ApplicationContext`

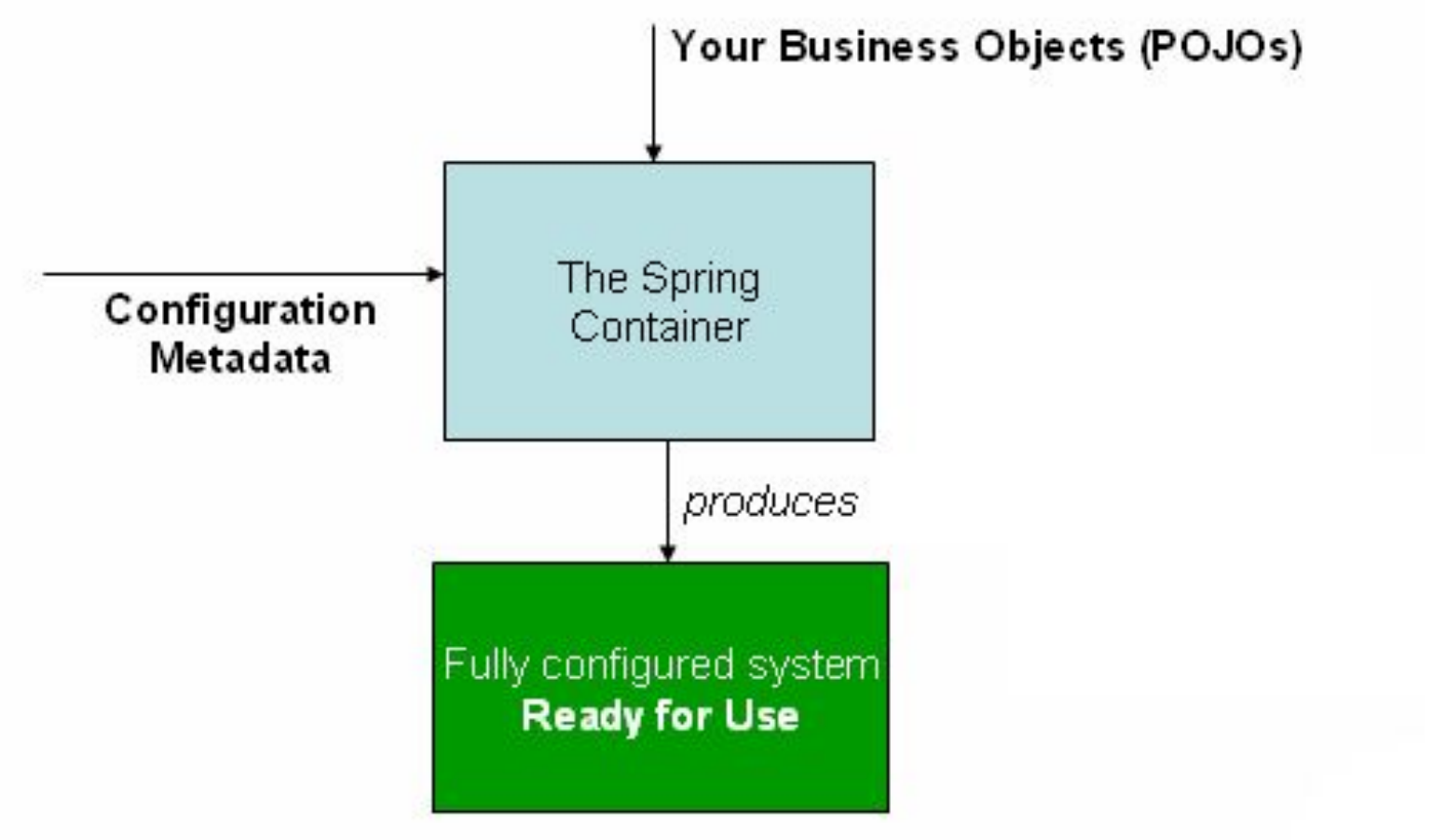
- AnnotationConfigApplicationContext
- AnnotationConfigWebApplicationContext
- ClassPathXmlApplicationContext
- FileSystemXmlApplicationContext
- GroovyWebApplicationContext
- StaticWebApplicationContext
- XmlWebApplicationContext
- etc

Bean Definition

- Receita para criação de objetos

- Métodos
  - Construtor
  - Fábricas
    - Método de instância
    - Método de classe

Inversão de Controle



Inversão de Controle

Fábricas
Service Locators
Injeção de Dependência
Polimorfismo
etc

Autowiring

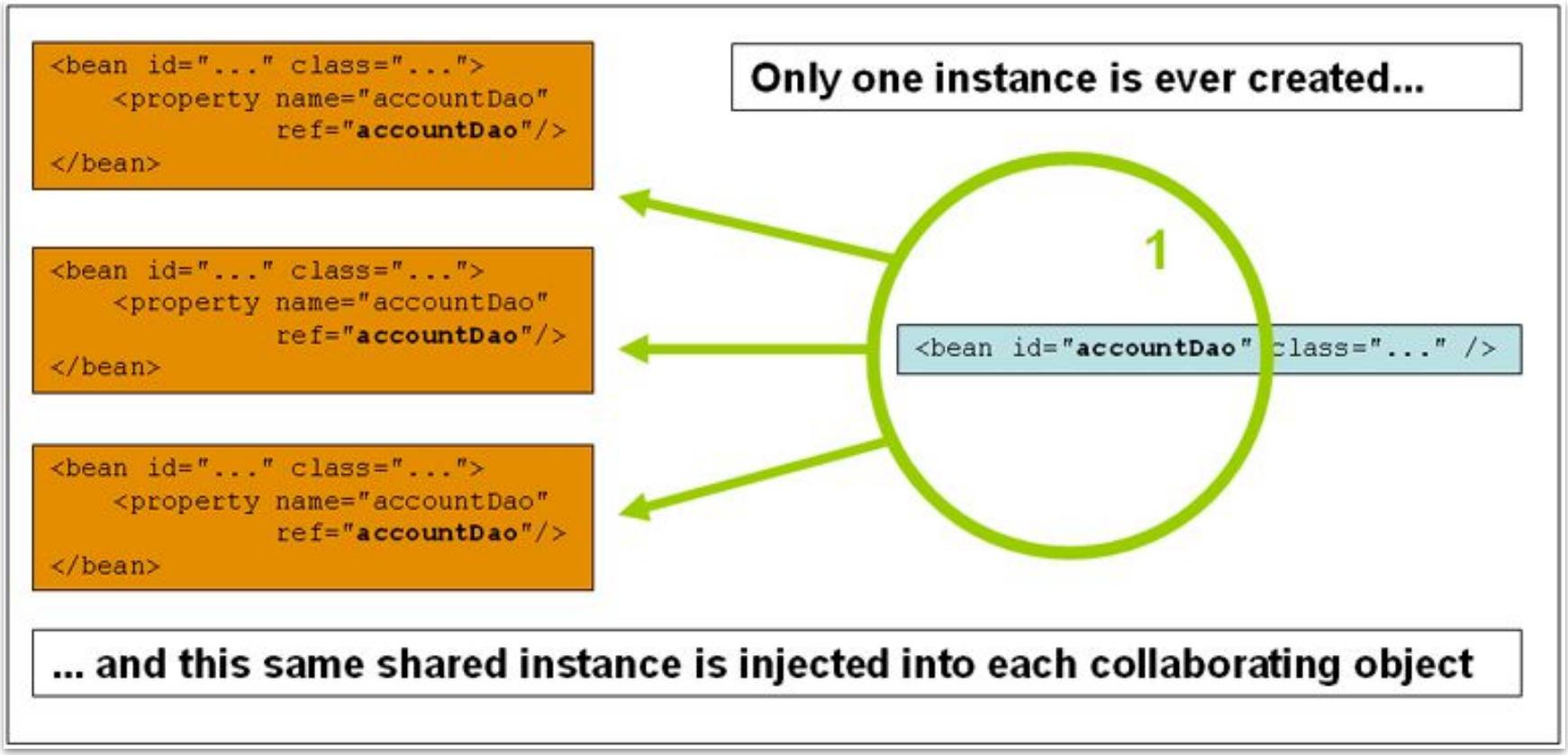
- Ligação automática entre beans
  - Nome
  - Tipo
  - Construtor

- Limitações
  - Não atende dependência para tipos simples
  - Precisão vs Ambiguidade
  - Coleções
  - Ligações explícitas sobrepõem *autowiring*

Escopo

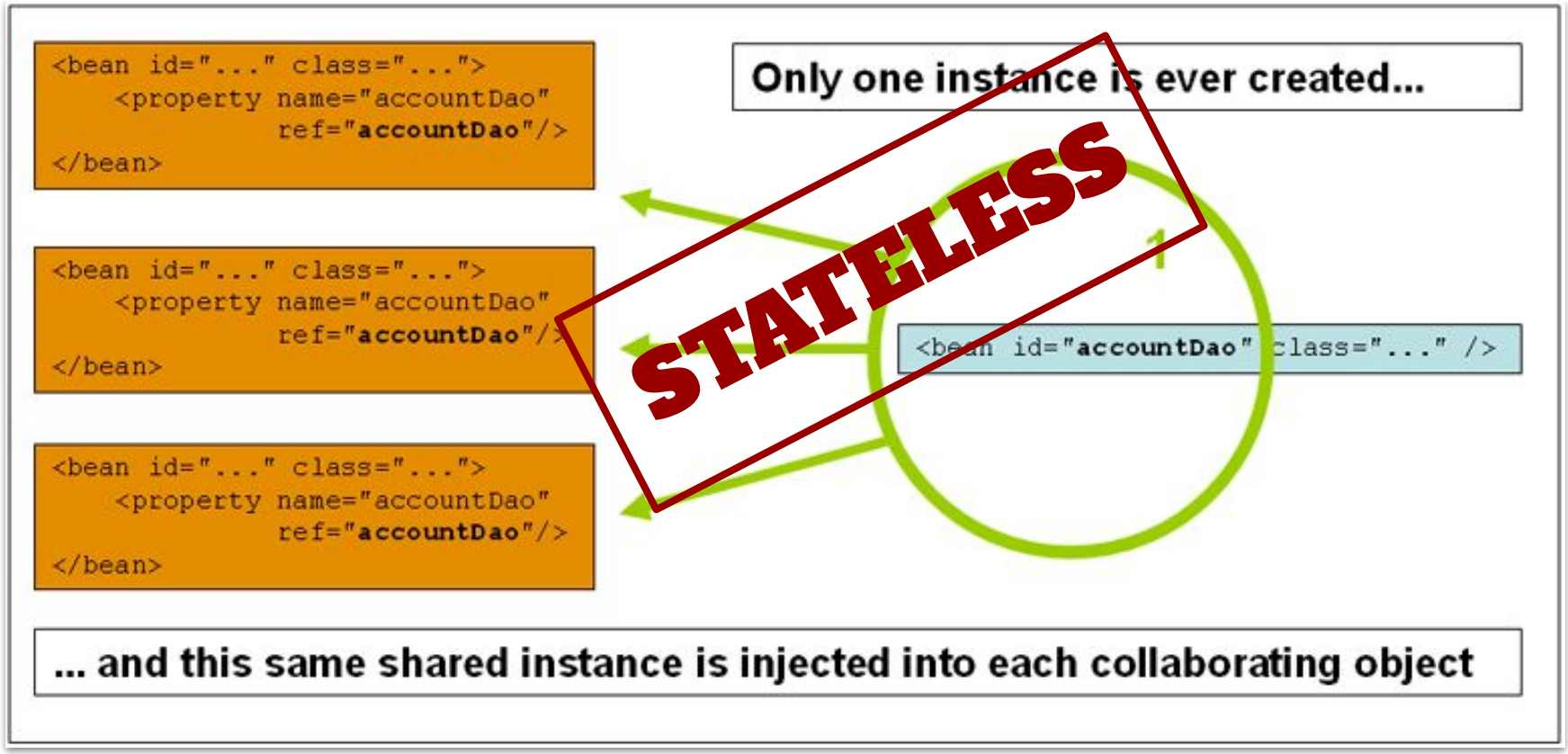
Escopo	Descrição
Singleton	Uma instância para várias referências
Prototype	Uma instância para cada referência
Request	Uma instância por request HTTP
Session	Uma instância por sessão HTTP
Application	Uma instância por aplicação Web
Websocket	Uma instância por Web Socket

Singleton



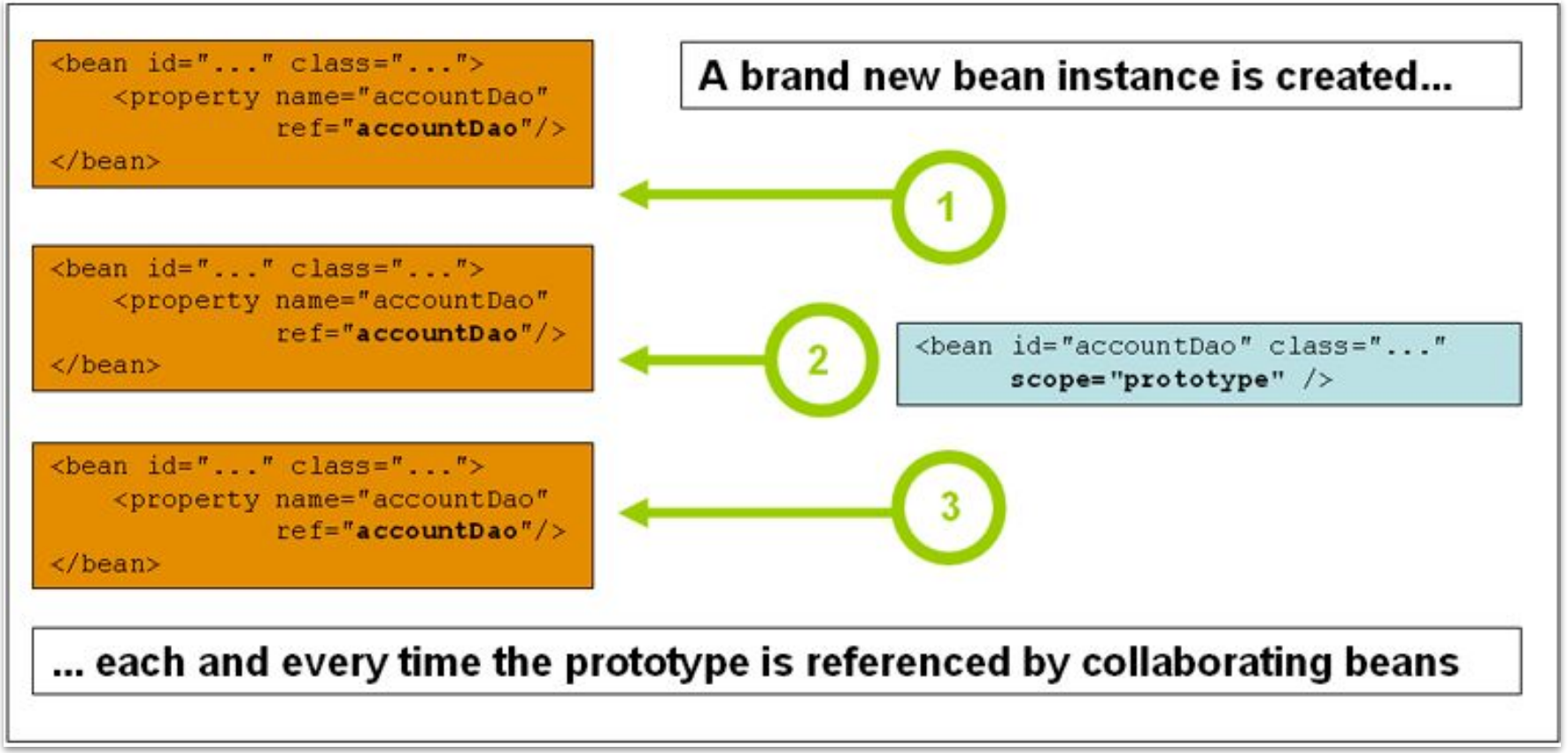
Fonte: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

Singleton



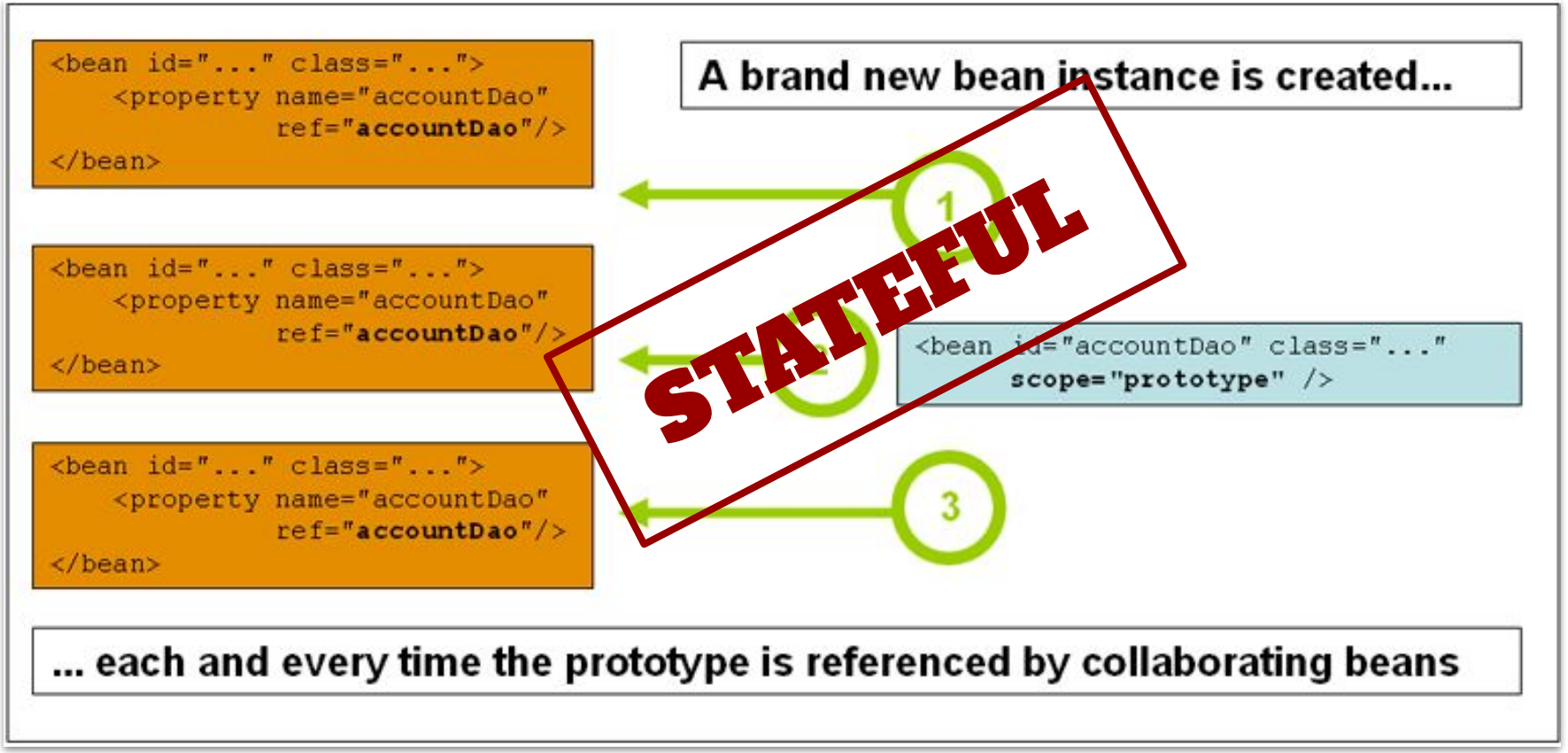
Fonte: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

Prototype



Fonte: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

Prototype



Fonte: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

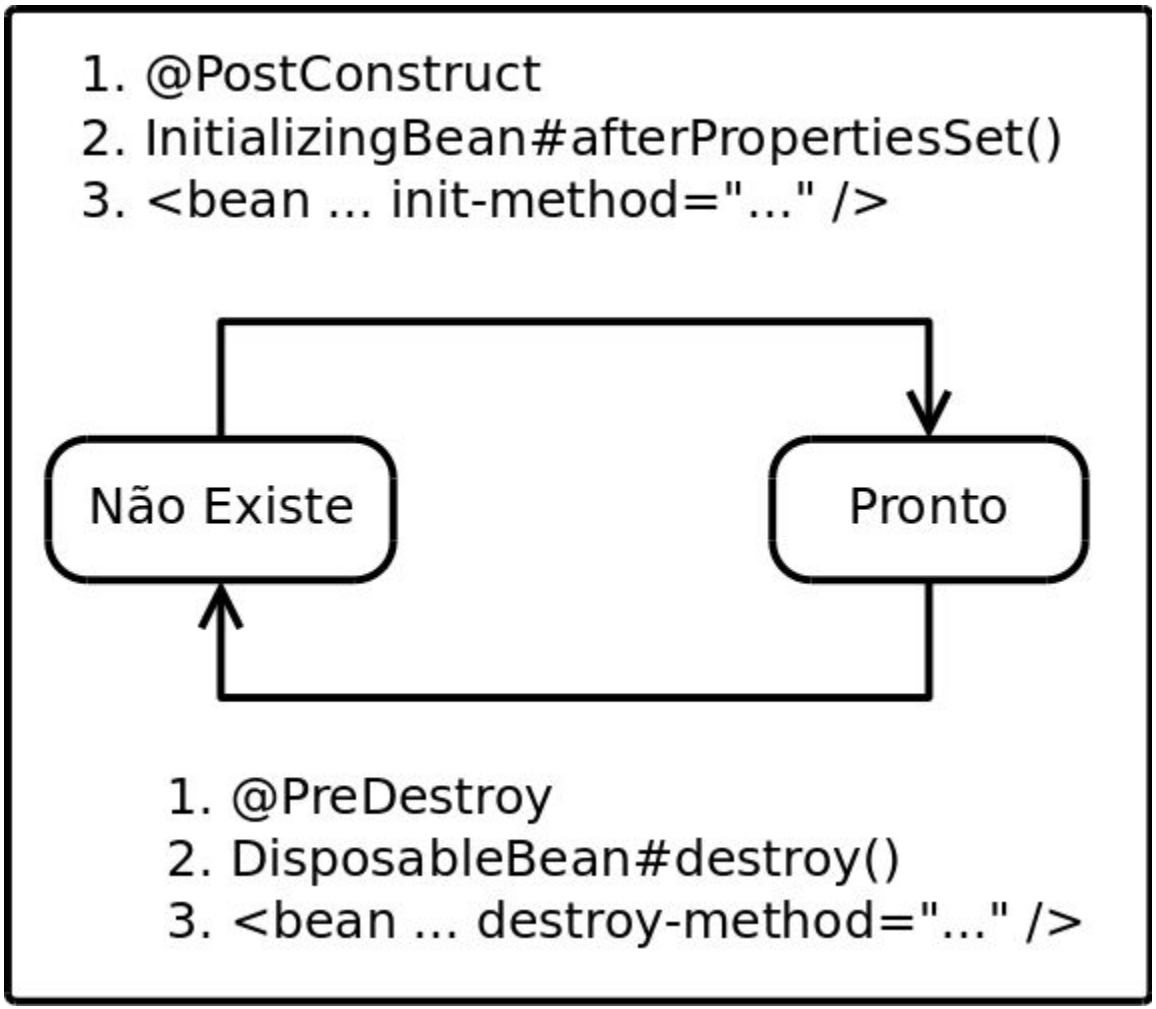
Escopo Customizado

Interface
org.springframework.beans.factory.config.Scope

```
Scope threadScope = new SimpleThreadScope();
beanFactory.registerScope("thread", threadScope);
```

```
<bean id="..." class="..." scope="thread">
```

Ciclo de vida



Configurações

- XML
- Anotações
- Código Java

XML

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

    <bean id="..." class="..."> (1) (2)
        <!-- collaborators and configuration for this bean go here -->
    </bean>

    <bean id="..." class="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>

    <!-- more bean definitions go here -->

</beans>
```

Fonte: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

Anotações

- Registro
  - XML vs Scanning
  - Anotações
    - `@Component`
    - `@Service`
    - `@Controller`

- Wiring
  - `@Autowire`
  - `@Inject`
  - `@Required`

Anotações

```
public class MovieRecommender {

    private final CustomerPreferenceDao customerPreferenceDao;

    @Autowired
    private MovieCatalog movieCatalog;

    @Autowired
    public MovieRecommender(CustomerPreferenceDao customerPreferenceDao) {
        this.customerPreferenceDao = customerPreferenceDao;
    }

    // ...
}
```

Fonte: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

Código Java

- Registro
  - Zero XML
  - Código, Scanning ou ambos
  - Anotações
    - `@Configuration`
    - `@ComponentScan`
    - `@Bean`

- Ligação
  - Modelo já utilizado
  - Métodos anotados com @Bean

Código Java

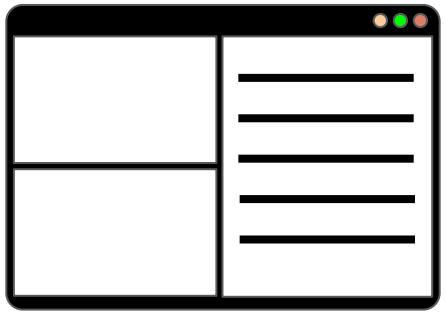
```java
@Configuration
public class AppConfig {

    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }
}
```

```xml
<beans>
    <bean id="myService" class="com.acme.services.MyServiceImpl"/>
</beans>
```

Fonte: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

Spring MVC



Handler Mapping

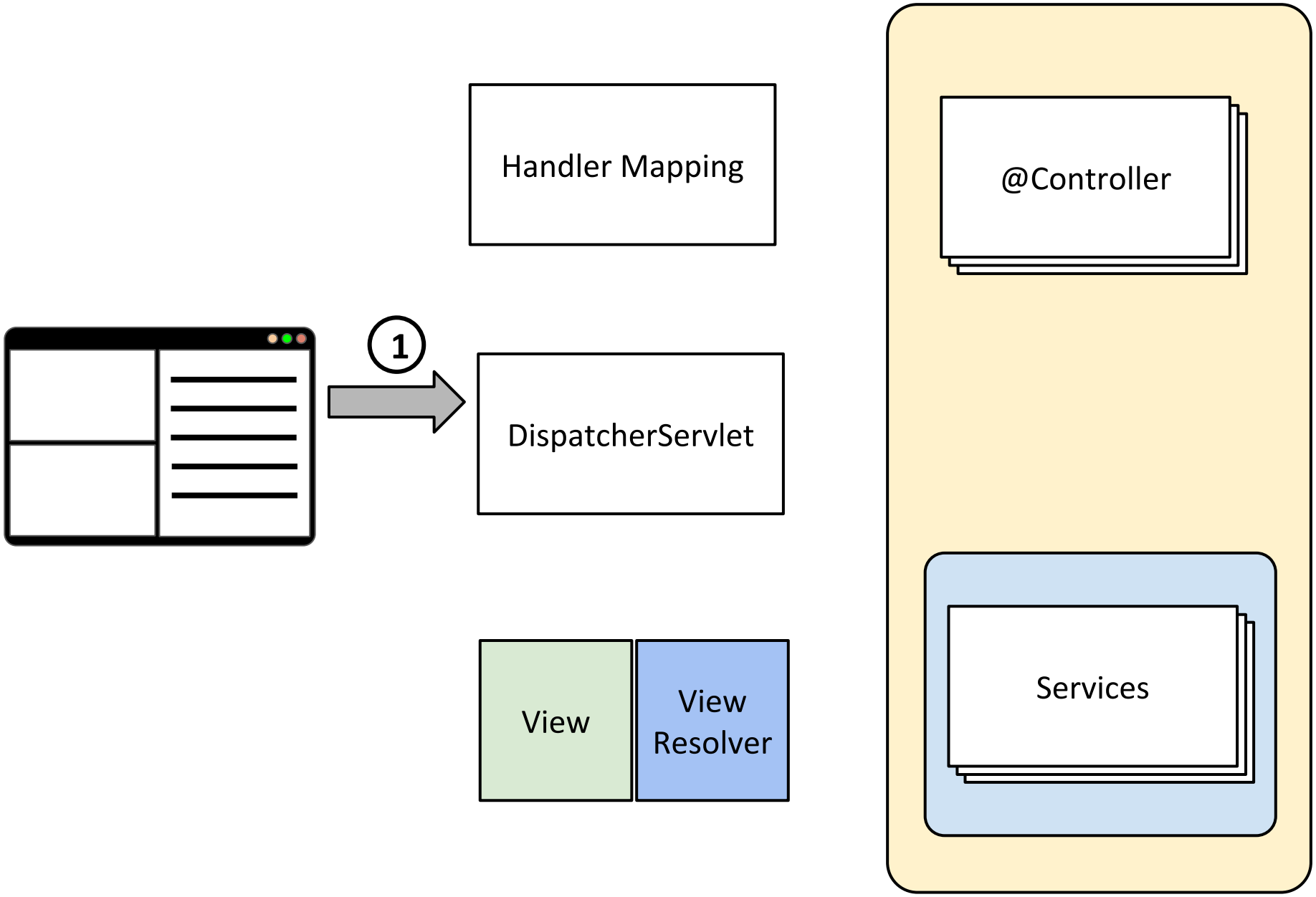
DispatcherServlet

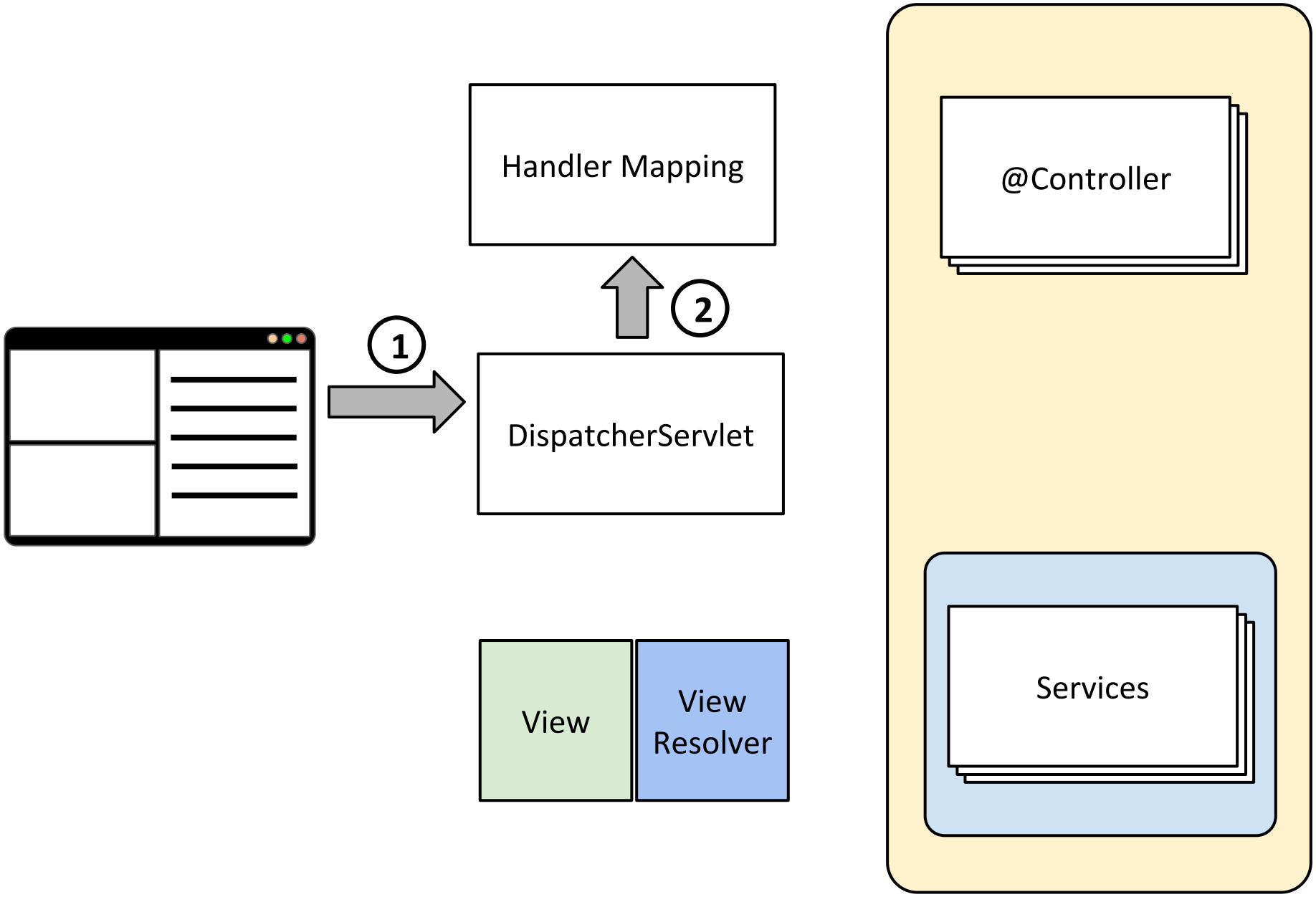
View

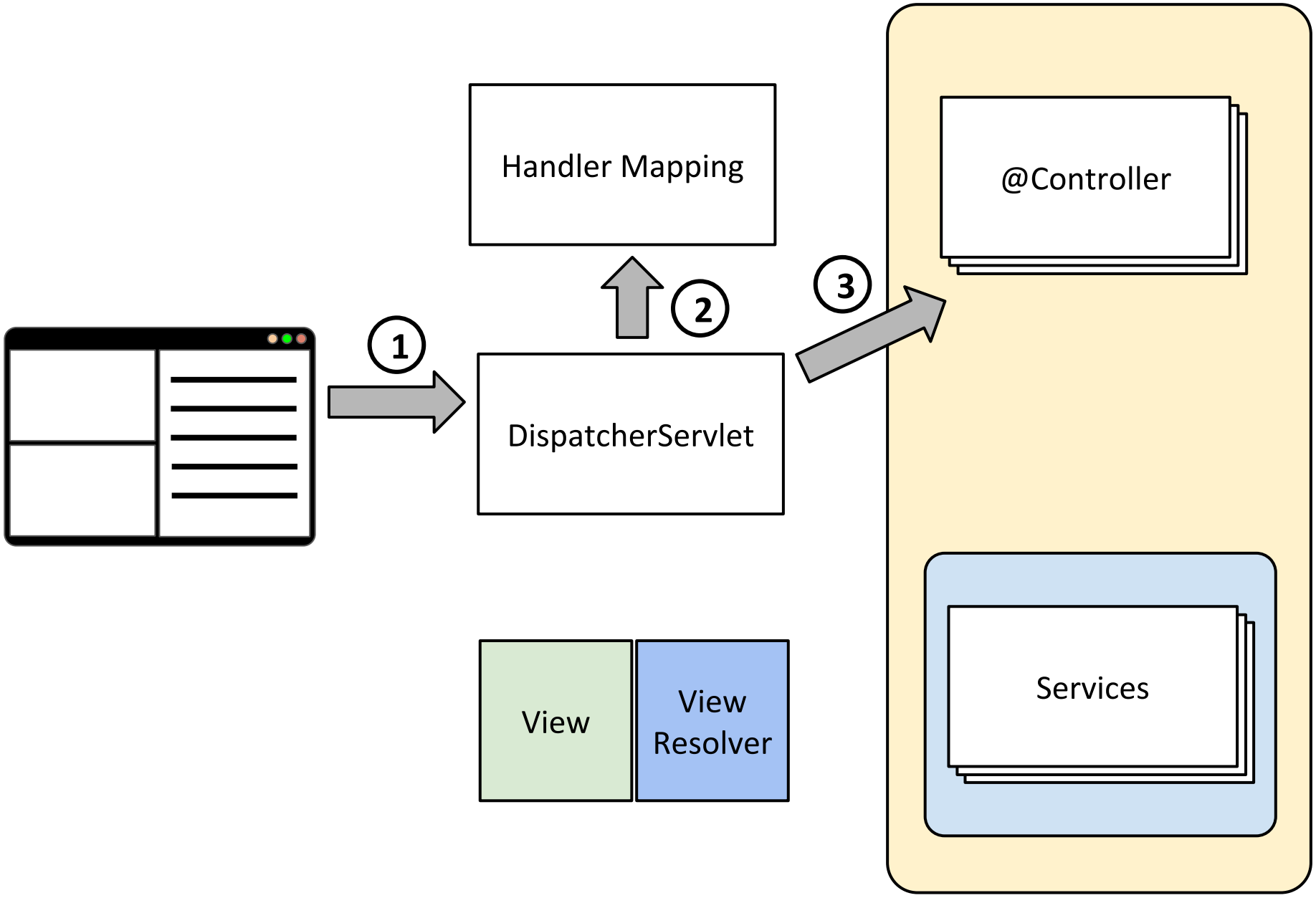
View Resolver

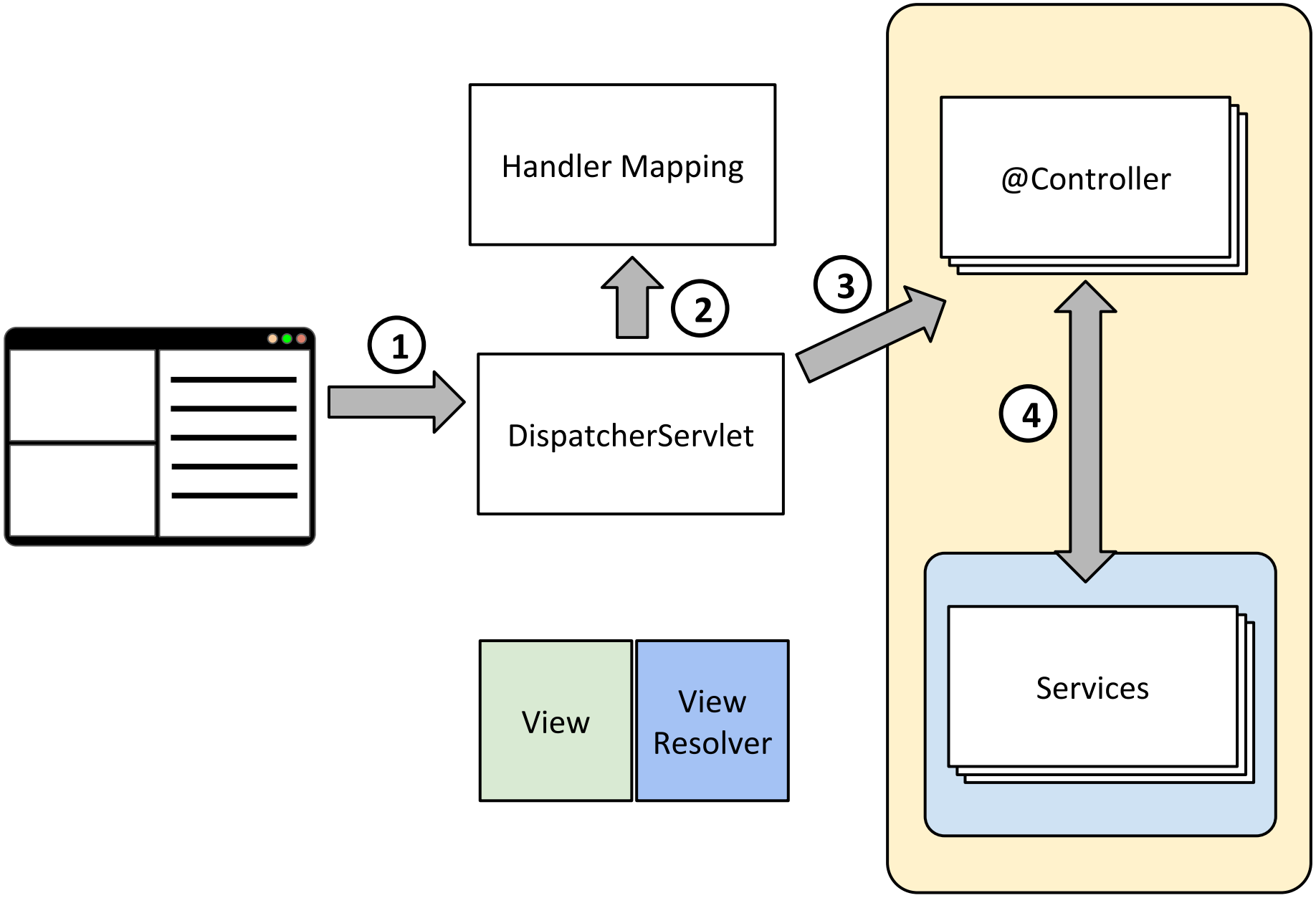
@Controller

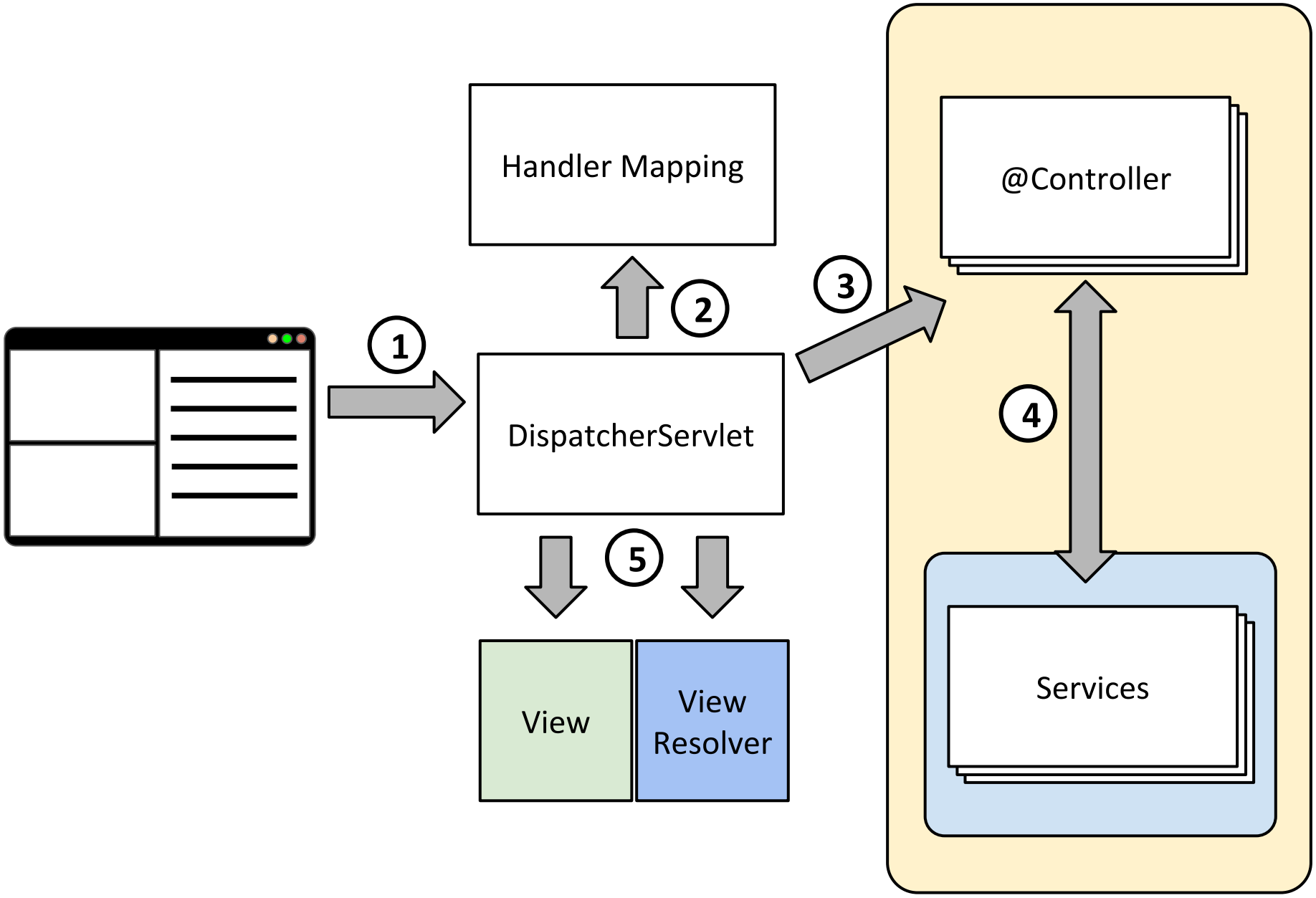
Services

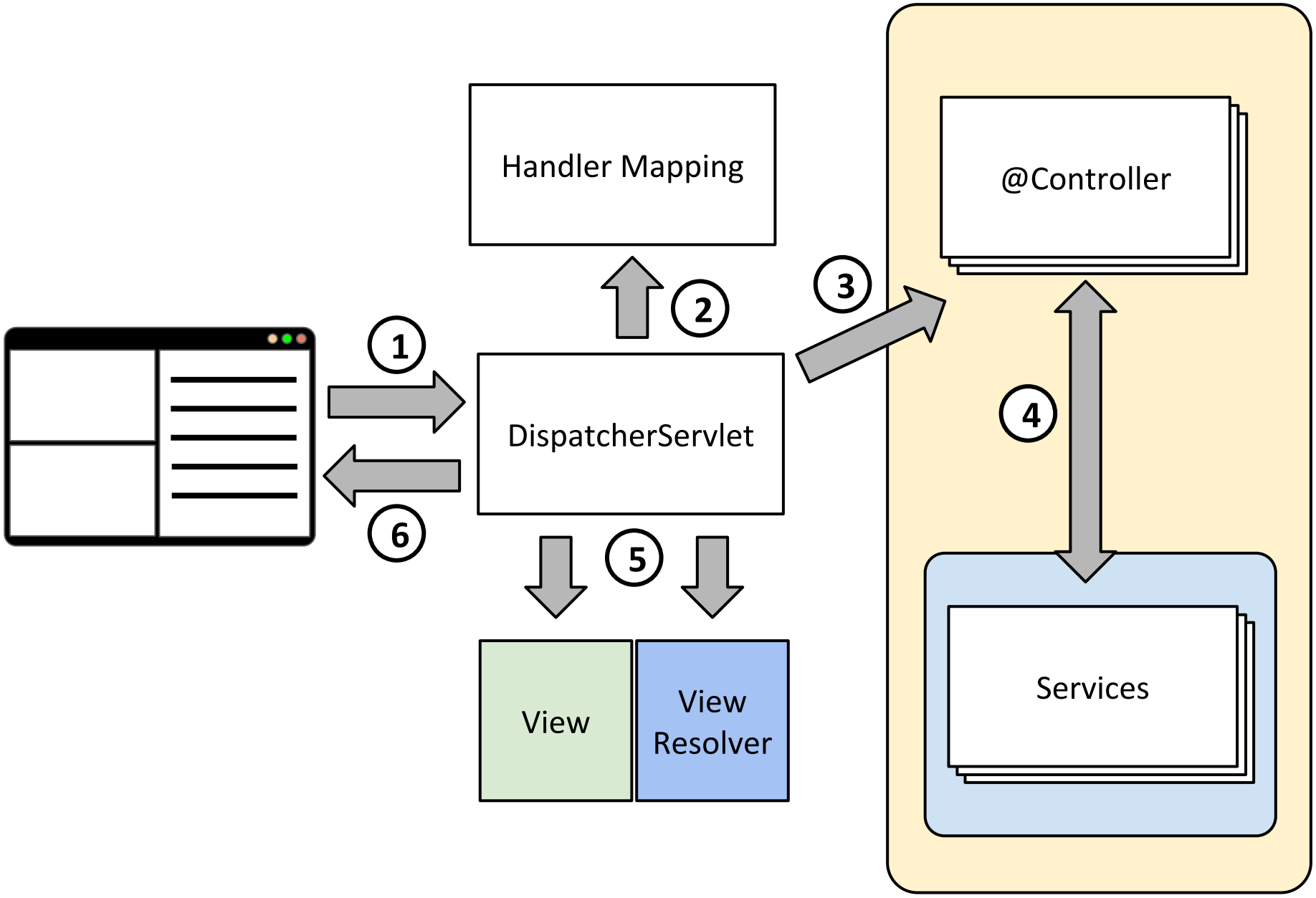

Handler Mapping
@Controller
1
DispatcherServlet
View
View Resolver
Services

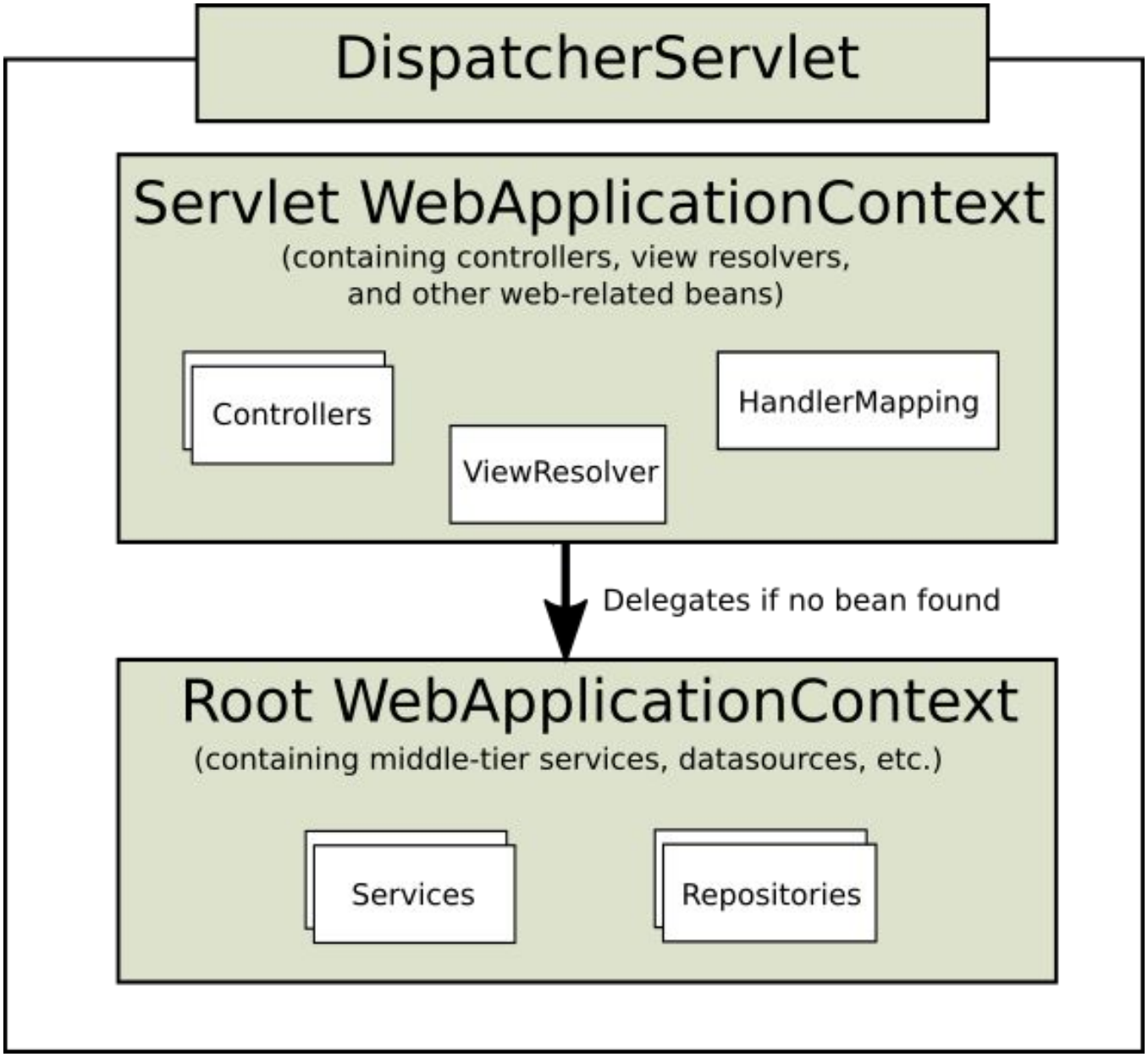

Handler Mapping
@Controller
1
2
DispatcherServlet
View
View Resolver
Services


Handler Mapping
@Controller
1
2
3
DispatcherServlet
View
View Resolver
Services


Handler Mapping
@Controller
1
2
3
4
DispatcherServlet
View
View Resolver
Services


Handler Mapping
@Controller
1
2
3
4
5
DispatcherServlet
View
View Resolver
Services


Handler Mapping
@Controller
1
2
3
4
5
6
DispatcherServlet
View
View Resolver
Services


DispatcherServlet
Servlet WebApplicationContext
(containing controllers, view resolvers,
and other web-related beans)
Controllers
ViewResolver
HandlerMapping
Delegates if no bean found
Root WebApplicationContext
(containing middle-tier services, datasources, etc.)
Services
Repositories

MVC Config

- Java Config vs XML Config
  - `@EnableWebMvc`

```
@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

    // Implement configuration methods...
}
```

Servlet Config

- Web.xml vs Código Java
  - Interface `WebApplicationInitializer`

```java
public class MyWebAppInitializer extends AbstractDispatcherServletInitializer {

    @Override
    protected WebApplicationContext createRootApplicationContext() {
        return null;
    }

    @Override
    protected WebApplicationContext createServletApplicationContext() {
        XmlWebApplicationContext cxt = new XmlWebApplicationContext();
        cxt.setConfigLocation("/WEB-INF/spring/dispatcher-config.xml");
        return cxt;
    }

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

Controles

- Classes anotadas com:
  - `@Controller`
  - `@RestController`

- `@RequestMapping` -> Mapeamento de requests para métodos
  - `@GetMapping`
  - `@PostMapping`
  - `@PutMapping`
  - `@DeleteMapping`
  - `@PatchMapping`

Controles

- Métodos de tratamento de requests suportam vários parâmetros e retornos
  - ServletRequest e ServletResponse
  - HttpSession
  - Parâmetros do request (@RequestParam)
  - Varíaveis do Path (@PathVariable)
  - View
  - ResponseBody
  - etc

- Lista extensiva em https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/web.html#mvc-ann-methods

Views e Resolvers

- Permitem o desacoplamento entre lógica de apresentação e tecnologias de exibição de dados

- Registrados como quaisquer beans

- Tipos
  - XMLViewResolver
  - URLbasedViewResolver
  - ResourceBundleViewResolver
  - etc

Views e Resolvers

```
// página myPage.jsp em /WEB-INF/view
@RequestMapping("/")
public String myPage() {
    return "myPage";
}
```

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

Transações

Pontos Importantes

- Suporta o modelo declarativo ou programático
  - XML
  - Anotações
  - Código

- Integra-se com diversas APIs
  - JDBC
  - JTA
  - JPA
  - etc

Transaction Manager

Principal interface de acesso à Transação

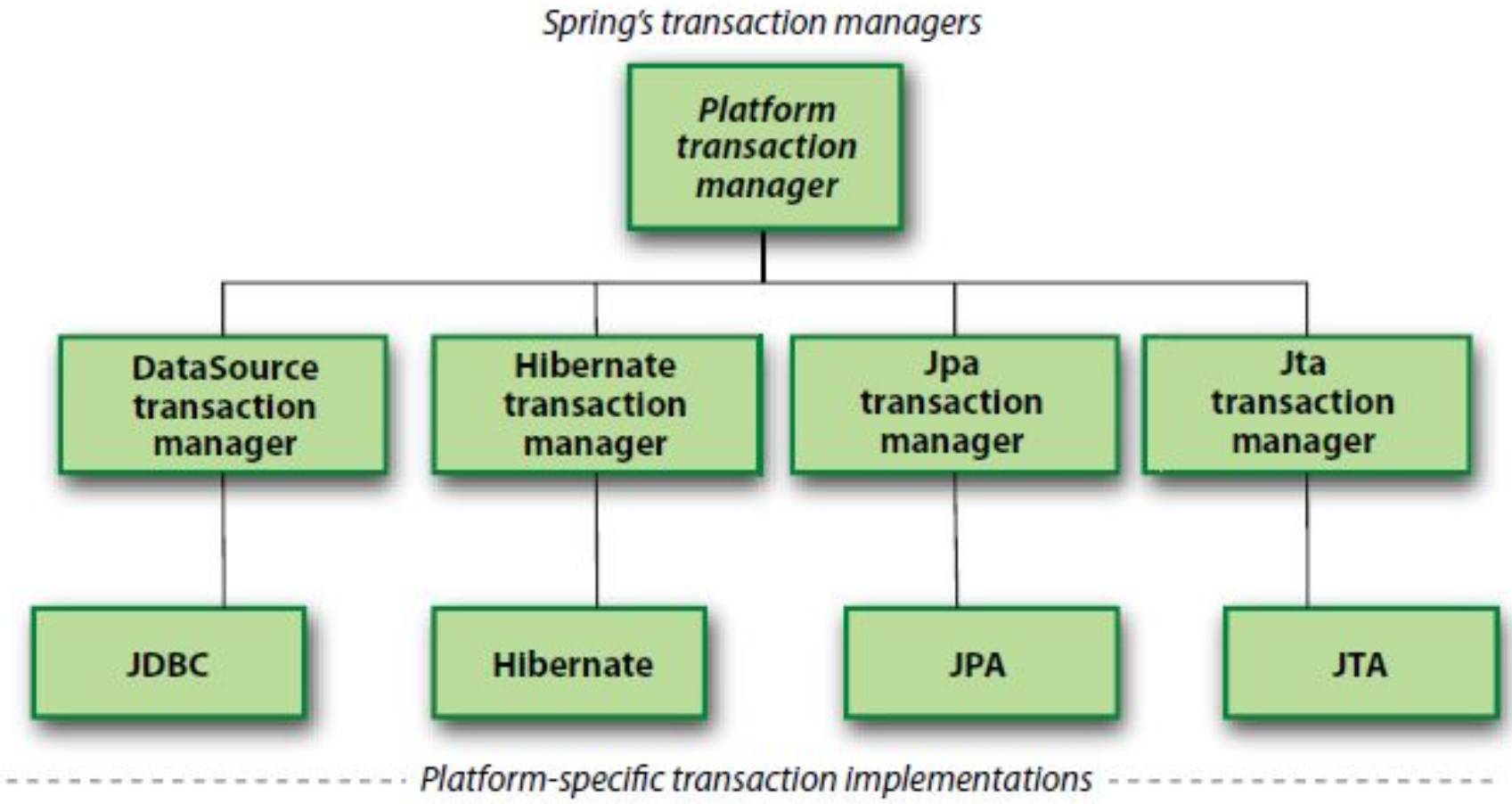
```
public interface PlatformTransactionManager {

    TransactionStatus getTransaction(TransactionDefinition definition) throws
TransactionException;

    void commit(TransactionStatus status) throws TransactionException;

    void rollback(TransactionStatus status) throws TransactionException;
}
```

Transaction Manager



Transaction Status

```
public interface TransactionStatus extends SavepointManager {

    boolean isNewTransaction();

    boolean hasSavepoint();

    void setRollbackOnly();

    boolean isRollbackOnly();

    void flush();

    boolean isCompleted();

}
```

Transaction Definition

- Propagação
  - Requires, Requires New e Nested
- Isolamento
  - Commited, Uncommited, Repeatable Read e Serializable
- Timeout
- Status

Definindo o TM

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="${jdbc.driverClassName}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
</bean>
<bean id="txManager">
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

```
@Configuration
@EnableTransactionManagement
public class TransactionManagersConfig {
    @Autowired
    EntityManagerFactory emf;
    @Autowired
    private DataSource dataSource;
    ...
    @Bean(name = "transactionManager")
    public PlatformTransactionManager transactionManager() {
        JpaTransactionManager tm =
            new JpaTransactionManager();
            tm.setEntityManagerFactory(emf);
            tm.setDataSource(dataSource);
        return tm;
    }
}
```

Utilizando

```
@Transactional(readOnly = true)
public class DefaultFooService implements FooService {

    public Foo getFoo(String fooName) {
        // do something
    }

    // these settings have precedence for this method
    @Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
    public void updateFoo(Foo foo) {
        // do something
    }
}
```

Utilizando

```
DefaultTransactionDefinition def = new DefaultTransactionDefinition();
// explicitly setting the transaction name is something that can be done only
programmatically
def.setName("SomeTxName");
def.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);

TransactionStatus status = txManager.getTransaction(def);
try {
    // execute your business logic here
}
catch (MyException ex) {
    txManager.rollback(status);
    throw ex;
}
txManager.commit(status);
```

SPRING DATA

Provides a consistent approach to data access – relational, non-relational, map-reduce, and beyond.

Missão

Fornecer um método simples e consistente para o acesso a diferentes tipos de dado, seguindo o jeito “Spring de ser”!

Interface Repositório

```
public interface CrudRepository<T, ID extends Serializable>
  extends Repository<T, ID> {

  <S extends T> S save(S entity);      ❶

  Optional<T> findById(ID primaryKey); ❷

  Iterable<T> findAll();               ❸

  long count();                        ❹

  void delete(T entity);               ❺

  boolean existsById(ID primaryKey);   ❻

  // … more functionality omitted.
}
```

Criando Repositórios

- Estenda uma das interfaces de repositório existente
- Utilize a entidade desejada e o seu Id
- Mapeie as consultas necessárias
  - *find…By, read…By, query…By, count…By, and get…By*
- Defina e injete os beans necessários

Queries

```
public interface UserRepository extends Repository<User, Long> {

  List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);
}
```

Named Queries

```
@Entity
@NamedQuery(name = "User.findByEmailAddress",
  query = "select u from User u where u.emailAddress = ?1")
public class User {

}
```

```
public interface UserRepository extends JpaRepository<User, Long> {

  List<User> findByLastname(String lastname);

  User findByEmailAddress(String emailAddress);
}
```

@Query

```
public interface UserRepository extends JpaRepository<User, Long> {

  @Query("select u from User u where u.emailAddress = ?1")
  User findByEmailAddress(String emailAddress);
}
```

```
public interface UserRepository extends JpaRepository<User, Long> {

  @Query(value = "SELECT * FROM USERS WHERE EMAIL_ADDRESS = ?1", nativeQuery = true)
  User findByEmailAddress(String emailAddress);
}
```

SPRING BOOT

Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible.

Pontos Importantes

- Visão direcionada do framework Spring para aplicações Web

- Aplicação no ar de forma “acelerada”

- Coisinhas legais como:
  - Configuração automática
  - Servidores embarcados
  - Aplicação console
  - Actuators
  - Starter Dependencies
  - Dev Tools

Auto Configuration

- Configuração automática de recursos baseada em bibliotecas existentes no ClassPath da aplicação
  - Ex.: Configuração de BD em memória

- `@EnableAutoConfiguration` / `@SpringBootApplication`

- Não se sobrepõe às configurações manuais

Starter Dependencies

- Conjunto pré-pronto de dependências para ser utilizado nos projetos
  - spring-boot-starter-web
  - spring-boot-starter-jpa
  - spring-boot-starter-security
  - etc

- Elimina a preocupação com interdependências entre bibliotecas e suas versões

Aplicação Console

- Aplicação que permite execução automática de classes \ controles no ambiente do Spring Boot

- Utiliza as nuances existentes na configuração automática e dependências iniciais

Aplicação Console

```groovy
@RestController
class ThisWillActuallyRun {

    @RequestMapping("/")
    String home() {
        "Hello World!"
    }

}
```

```
$ spring run app.groovy
```

```
michel@vega:~/ap
[michel@vega apps]$ curl -w "\n" http://localhost:8080
Hello World!
[michel@vega apps]$
```

Actuator

- Ferramenta de monitoração e inspeção da aplicação em tempo de execução
  - Beans configurados
  - Requisições recebidas e respondidas
  - Configurações automáticas realizadas
  - Comandos executados
  - Variáveis de ambiente existentes
  - Threads em execução
  - etc

Criando uma Aplicação

- Use o Spring Initializr
- Selecione as dependências necessárias
- Importe o projeto descompactado na IDE
- Siga com a implementação


VAMOS FALAR MENOS E PRATICAR MAIS?
ISSO, ISSO, ISSO

Exercício 05

Exercício 05

Crie uma aplicação que controla a lista de servidores existentes em uma empresa. Basicamente um CRUD da entidade servidor.

Exercício 05

Dicas:

- Utilize o Spring Initializr para criar seu projeto
- Utilize o H2 como banco de dados em memória caso deseje
- Crie um repositório específico para sua entidade
- Crie as buscas necessárias utilizando os padrões de nomenclatura

Referências

Spring Framework Core

https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

Spring Boot

https://docs.spring.io/spring-boot/docs/2.2.0.BUILD-SNAPSHOT/reference/html/index.html

https://docs.spring.io/spring-boot/docs/2.2.0.BUILD-SNAPSHOT/reference/html/howto.html#howto

Referências

Spring Data

https://spring.io/projects/spring-data