Ciência da Computação

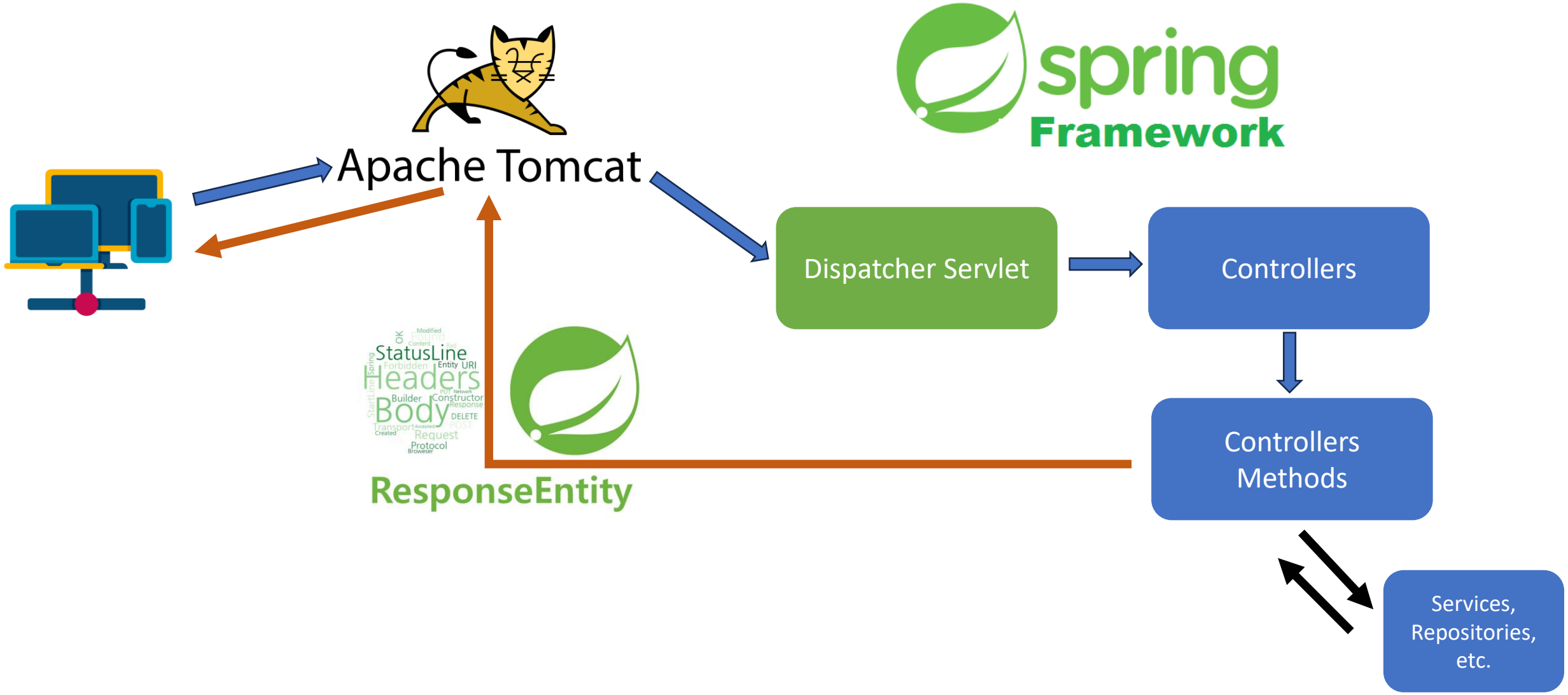
Programação Orientada a Objetos Avançado

Prof. Luciano Rodrigo Ferretto

Entendendo o Fluxo da Requisição no Spring Boot: Do Tomcat ao Controller

Como nossa requisição chega até a nossa Controller???

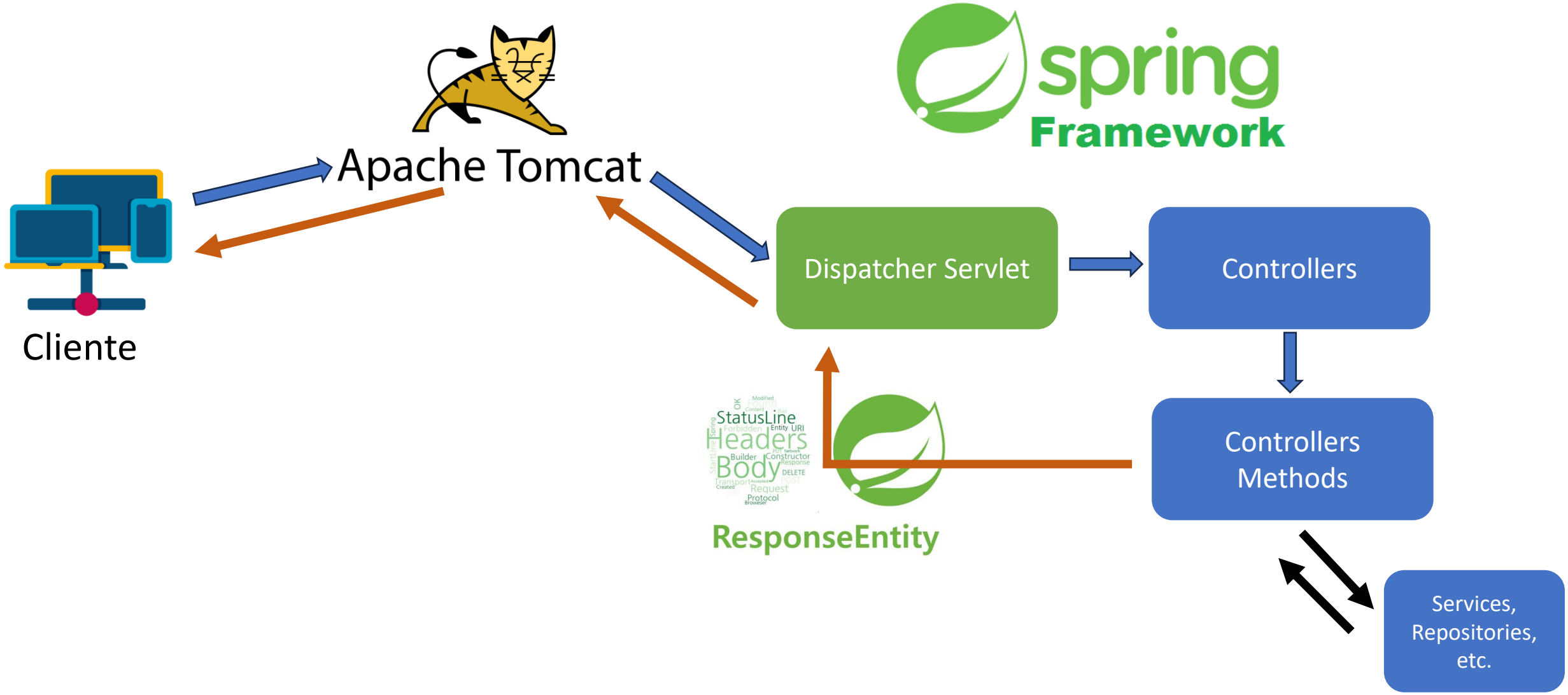
Fluxo de Requisição HTTP com Spring Boot



Entendendo o Fluxo da Requisição no Spring Boot: Do Tomcat ao Controller

- **O Cliente (Navegador/Postman)**: É a pessoa que faz o pedido. Ele não sabe como a comida é feita, só sabe o que quer pedir.
- **O Garçom (Tomcat)**: É o primeiro a receber o pedido do cliente. Ele não cozinha, apenas anota o pedido e o leva para a cozinha. No nosso caso, o Tomcat (Spring Boot já traz embutido) é o servidor web. Ele "ouve" as requisições HTTP que chegam na porta do seu aplicativo (geralmente 8080).
  - **Ponto Chave**: O Tomcat é um contêiner de servlets. Ele é responsável por gerenciar as requisições HTTP de baixo nível, como criar um HttpServletRequest e um HttpServletResponse para cada requisição.

Fluxo de Requisição HTTP com Spring Boot

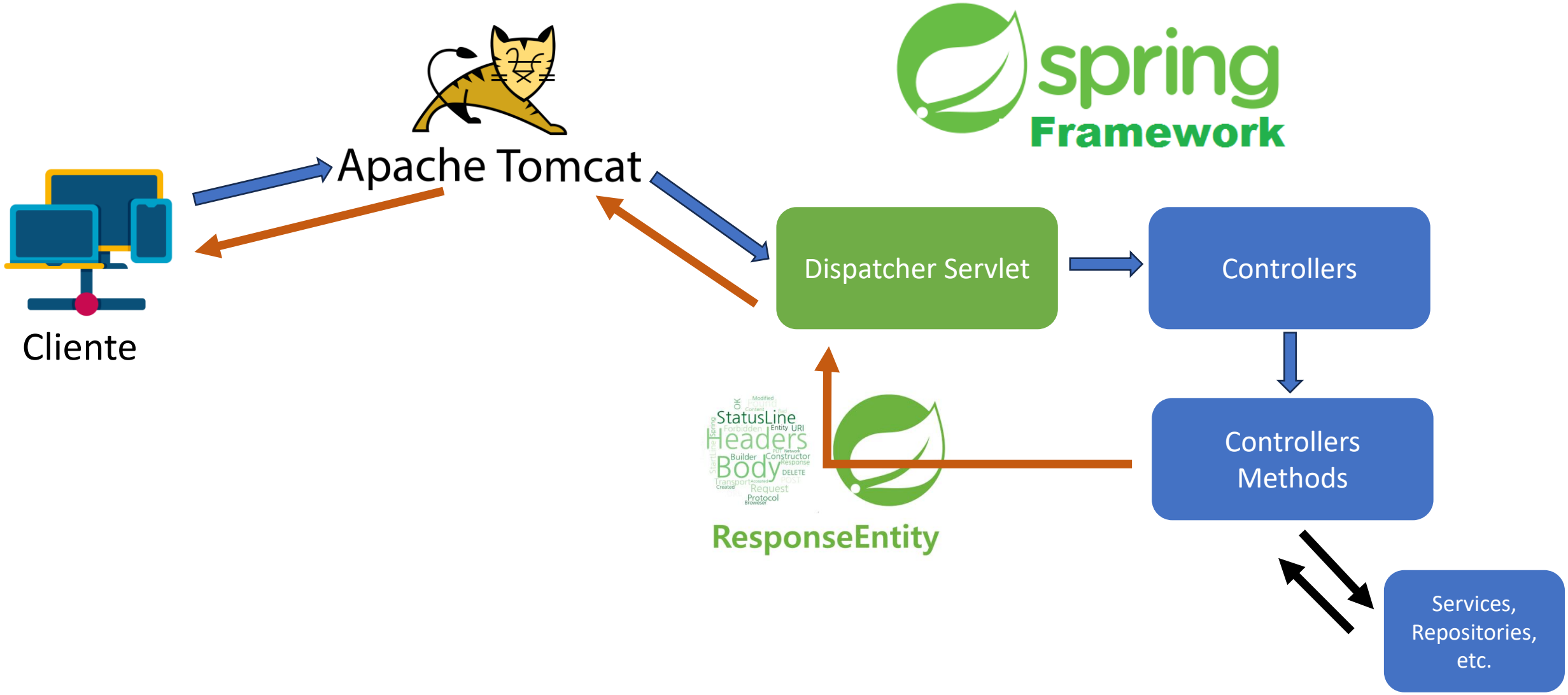


Entendendo o Fluxo da Requisição no Spring Boot: Do Tomcat ao Controller

- **A Cozinha (Spring Framework)**: Uma vez que o garçom (Tomcat) recebe o pedido, ele o entrega para a cozinha. A cozinha é o Spring Framework. Ele é o "cérebro" que orquestra todo o processo.

    - **DispatcherServlet**: Dentro do Spring, o "chefe da cozinha" é o DispatcherServlet. Ele é o ponto de entrada para todas as requisições que chegam ao Spring. Ele recebe a requisição do Tomcat.
    - **Mapeamento de Requisições**: O DispatcherServlet então olha para o pedido (a URL, o método HTTP como GET, POST, etc.) e tenta encontrar o "cozinheiro" certo para aquele pedido. É aqui que o *@RequestMapping* e *@RestController* entram em jogo. Ele sabe que a URL */auth/signup* com um método POST deve ser direcionada para a *AuthController*.
    - **Os Cozinheiros (Controllers)**: Sua *AuthController* é o "cozinheiro" especializado em lidar com pedidos relacionados à autenticação. Quando o DispatcherServlet encontra o mapeamento para */auth/signup* e POST, ele entrega a requisição para o método *signup* da sua *AuthController*.

Fluxo de Requisição HTTP com Spring Boot

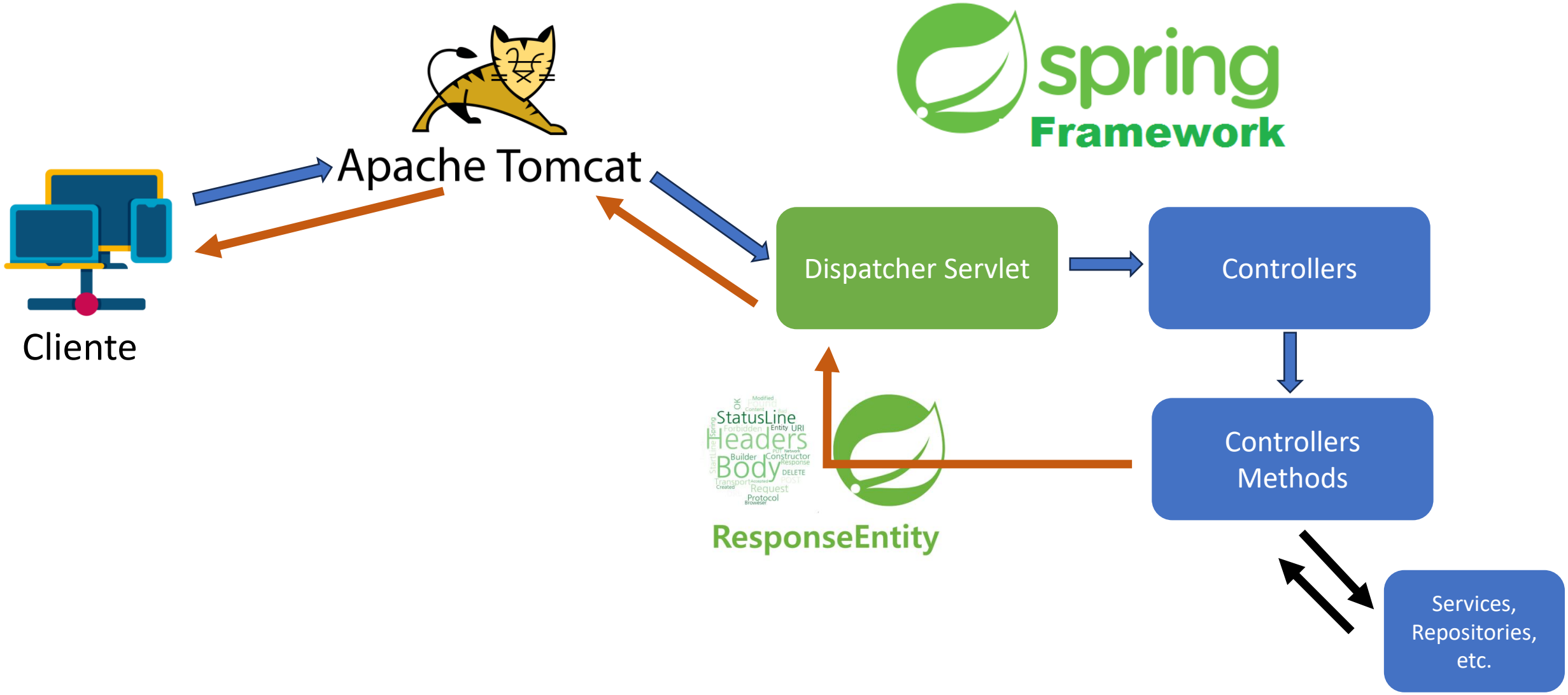


Entendendo o Fluxo da Requisição no Spring Boot: Do Tomcat ao Controller

- **O Processo de Cozinha (Métodos do Controller e Serviços)**:

  - O método *signup* na *AuthController* recebe os ingredientes (*@RequestBody SignupDTO dto*).
  - Ele prepara parte do prato (cria o *UserEntity*, copia as propriedades, seta o tipo de usuário).
  - E então, para as partes mais complexas da receita (salvar o usuário, fazer validações), ele delega para outro "cozinheiro" mais especializado: o *UserService*.

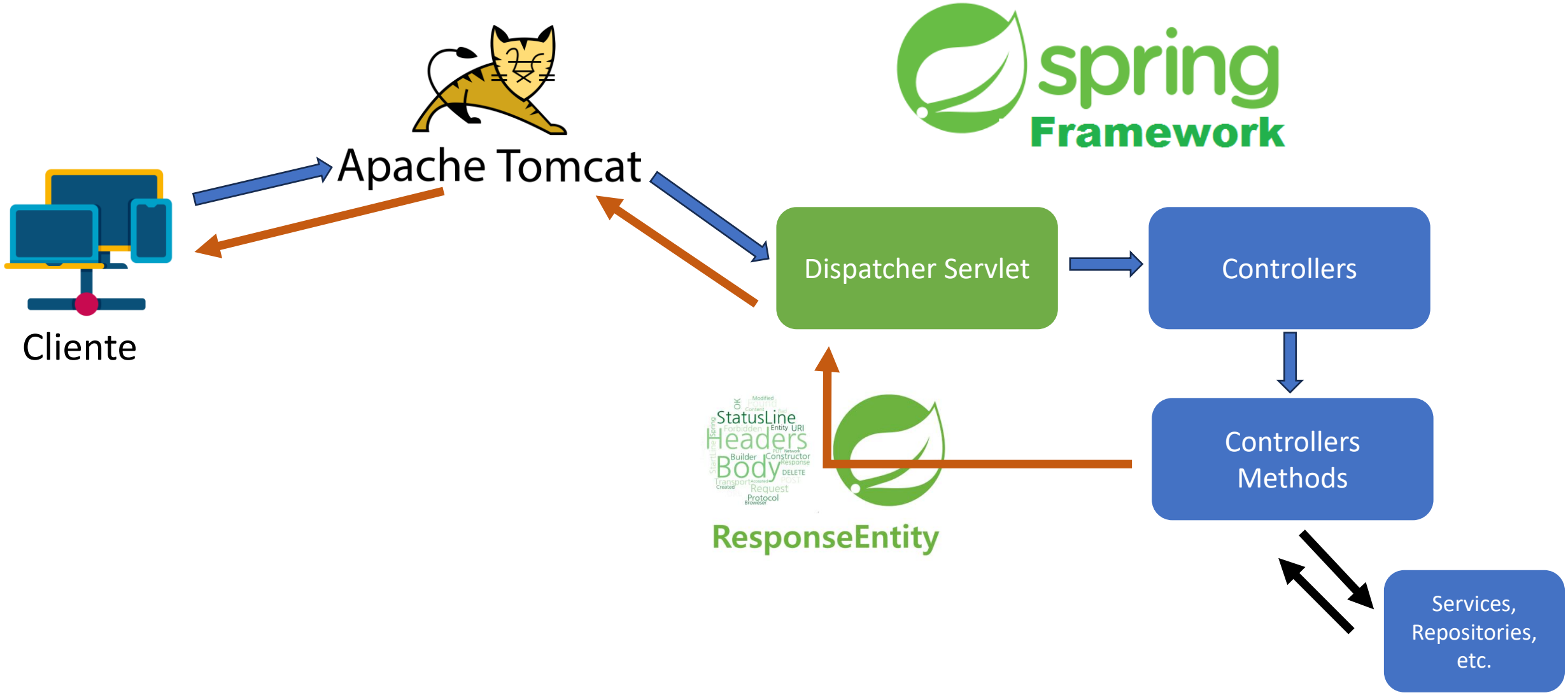
Fluxo de Requisição HTTP com Spring Boot



Entendendo o Fluxo da Requisição no Spring Boot: Do Tomcat ao Controller

- **Devolvendo o Prato (*ResponseEntity*)**: Uma vez que o *UserService* termina seu trabalho, a *AuthController* monta o prato final (*UserEntity*) e o entrega de volta ao *DispatcherServlet* em uma bandeja (*ResponseEntity*).
- **Garçom de Volta (Tomcat)**: O *DispatcherServlet* entrega a bandeja para o *Tomcat*, que então envia a resposta HTTP de volta para o cliente.

Fluxo de Requisição HTTP com Spring Boot



AuthController como um Bean Spring: O Que Significa Ser Gerenciado Pelo Spring?

Quem cria os objetos???

Vamos pensar um pouco...

```java
public class ExampleMethods {

    public void exampleMethodInstance() {
        System.out.println("""
                Este é um método de instância.
                Para chamar este método, você precisa criar uma instância da classe ExampleMethods.
                Exemplo:
                    ExampleMethods example = new ExampleMethods();
                    example.exampleMethodInstance();
                """);
    }

    public static void exampleMethodStatic() {
        System.out.println("""
                Este é um método estático, ou seja, um método da Classe.
                Você pode chamar este método diretamente na classe ExampleMethods, sem precisar criar uma instância.
                Exemplo:
                    ExampleMethods.exampleMethodStatic();
                """);
    }
}
```

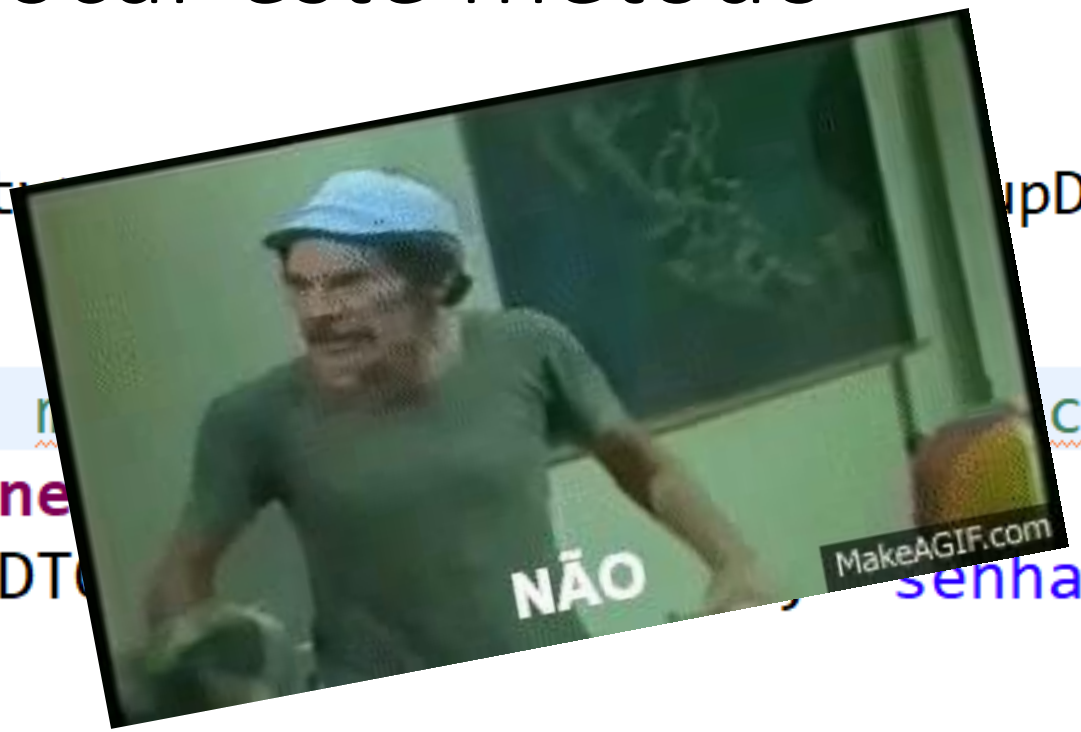
Que tipo de método é esse que está na nossa controladora???

```java
@PostMapping("/signup")
public ResponseEntity<UserEntity> signup(@RequestBody SignupDTO dto) throws Exception{
    // Criamos a entidade (novo objeto)
    UserEntity user = new UserEntity();
    // Copia-se as propriedades da DTO para a entidade
    BeanUtils.copyProperties(dto, user);
    // Seta-se os valores que não vieram no DTO
    user.setType(UserType.Common);

    service.save(user);

    return ResponseEntity.status(HttpStatus.CREATED).body(user);
}
```

Método de Instância

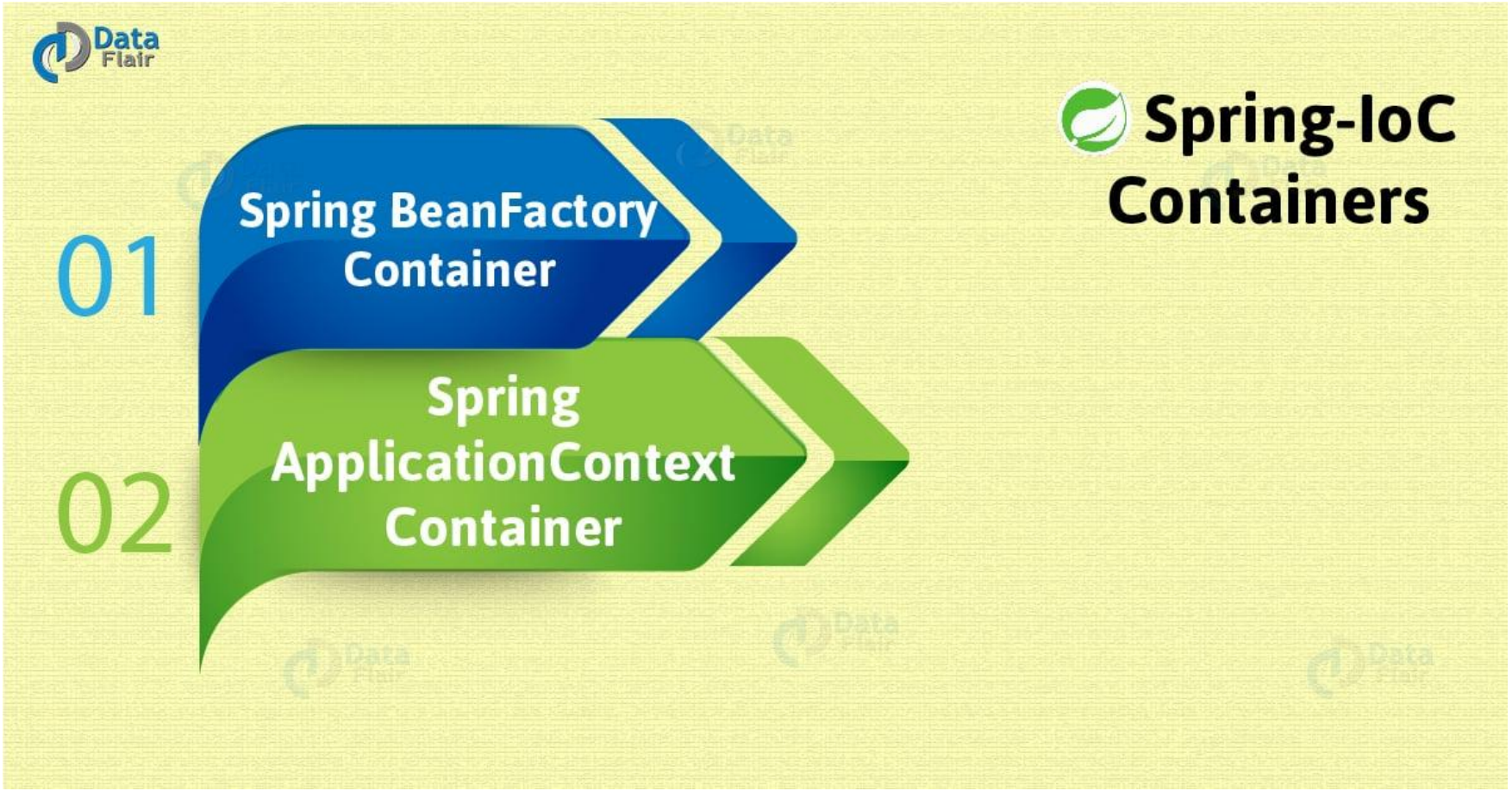
E como podemos invocar este método *signup???*

```
public ResponseEntity<UserEntit                    upDTO dto)

// Exemplo de como chamar um m                     classe
AuthController controller = ne
controller.signup(new SignupDT                     senha"));
```



No nosso código atual, existe essa instanciação??? `new AuthController();`

Então, quem instancia/cria esse objeto???



O Problema da Criação de Objetos

- Na programação tradicional, se você precisa usar um objeto, você o cria usando **new**. Por exemplo, *UserService service = new UserService()*;.
- Isso é perfeitamente válido, mas tem algumas desvantagens:

  - **Acoplamento Forte**: Sua *AuthController* está "presa" a *UserService*. Se no futuro você quiser usar uma implementação diferente de *UserService* (talvez um *UserServiceV2* ou um *MockUserService* para testes), você terá que mudar o código da *AuthController*.
  - **Gerenciamento Manual**: Você precisa se preocupar em criar e destruir os objetos. Para objetos simples, isso não é um problema, mas em sistemas complexos, com muitos objetos e dependências entre eles, isso se torna um pesadelo.

A Solução do Spring: Inversion of Control (IoC) e Contêiner IoC

- O Spring **inverte o controle**. Em vez de você criar os objetos, o Spring se encarrega de criar e gerenciar os objetos para você.
- O **Contêiner IoC** (também conhecido como **Application Context**) é o "gerente de objetos" do Spring. Ele é responsável por:

  - **Instanciar Objetos**: Ele cria os objetos que você precisa (como sua *AuthController* e *UserService*).
  - **Configurar Objetos**: Ele injeta as dependências entre eles. Se a *AuthController* precisa de um *UserService*, o Spring cria o *UserService* e o "entrega" para a *AuthController*.
  - **Gerenciar o Ciclo de Vida**: Ele sabe quando um objeto deve ser criado, quando ele pode ser reutilizado e quando deve ser destruído.

O que é um “Bean Spring”?

- Um **Spring Bean** é simplesmente um objeto que é instanciado, montado e gerenciado pelo **contêiner IoC** do Spring.
- Quando você anota sua *AuthController* com *@RestController*, você está dizendo ao Spring:

"Ei, Spring, eu quero que você gerencie essa classe para mim. Ela é um componente da minha aplicação, e você deve tratá-la como um Bean."

O que é um “Bean Spring”?

- **Explicação prática:**
- Um Bean é um objeto que o Spring cria e gerencia automaticamente.
- Toda classe anotada com *@RestController*, *@Service*, *@Repository* etc., é registrada como um Bean no container do Spring.

“É como se o Spring tivesse um armário com objetos prontos. Sempre que precisa de um controller ou service, ele pega do armário, ao invés de criar um novo do zero.”

Por que é vantajoso?

- **Acoplamento Fraco**: Quando você injeta dependências (como você fará com a injeção por construtor), a *AuthController* não sabe como o *UserService* foi criado, apenas que ele tem um *UserService* para usar. Isso torna seu código mais flexível e fácil de testar.
- **Reusabilidade**: O Spring pode reutilizar as instâncias dos Beans. Geralmente, um RestController é um singleton por padrão, o que significa que o Spring cria apenas uma instância dele e a reutiliza para todas as requisições. Isso economiza recursos e torna a aplicação mais eficiente.
- **Gerenciamento Centralizado**: O Spring lida com a complexidade de criar e conectar todos os componentes da sua aplicação. Você se concentra na lógica de negócio.

Certo, mas e o nosso código???

O container IoC irá criar e gerenciar objetos dessa classe

```java
@RestController
@RequestMapping("/auth")
public class AuthController {

    private final UserService service;

    public AuthController() {
        super();
        this.service = new UserService();
    }
```

Certo, mas e o nosso código???

```java
@RestController
@RequestMapping("/auth")
public class AuthController {

    private final UserService service;

    public AuthController(UserService service) {
        super();
        this.service = service;
    }
```

Como é o container IoC quem vai criar o objeto.

Vamos deixar que ele seja o responsável por ***injetar essa dependência via método construtor***

```java
@RestController
@RequestMapping("/auth")
public class AuthController {

    private final UserService service;

    public AuthController(UserService service) {
        super();
        this.service = service;
    }
```


```
***************************
APPLICATION FAILED TO START
***************************

Description:

Parameter 0 of constructor in br.edu.atitus.api_sample.controllers.AuthController required
        a bean of type 'br.edu.atitus.api_sample.services.UserService' that could not be found.
```

Por que deu erro?

- Esse erro quer dizer que o Spring não achou esse “feijão” na sua prateleira!!!
- Para resolver, precisamos informar ao container IoC que ele deve tratar a classe *UserService* como um Bean do Spring.

```java
@Service
public class UserService {
```

Ao iniciar a aplicação, o container IoC do Spring irá instanciar um objeto desta classe, e assim poderá utilizar quando precisar.

```java
@RestController
@RequestMapping("/auth")
public class AuthController {

    private final UserService service;

    public AuthController(UserService service) {
        super();
        this.service = service;
    }
```

Agora, quando o Spring vai instanciar a *AuthController*, ele percebe que ela precisa de um *UserService*. Então, o Spring irá buscar a instância criada na inicialização