

Comparing & Contrasting Spring WebMVC & WebFlux

Douglas C. Schmidt

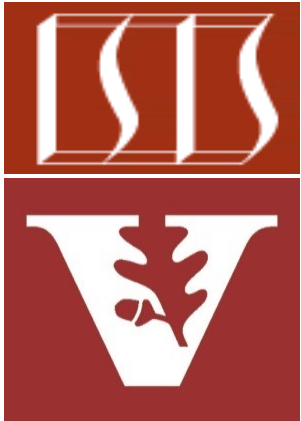
d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

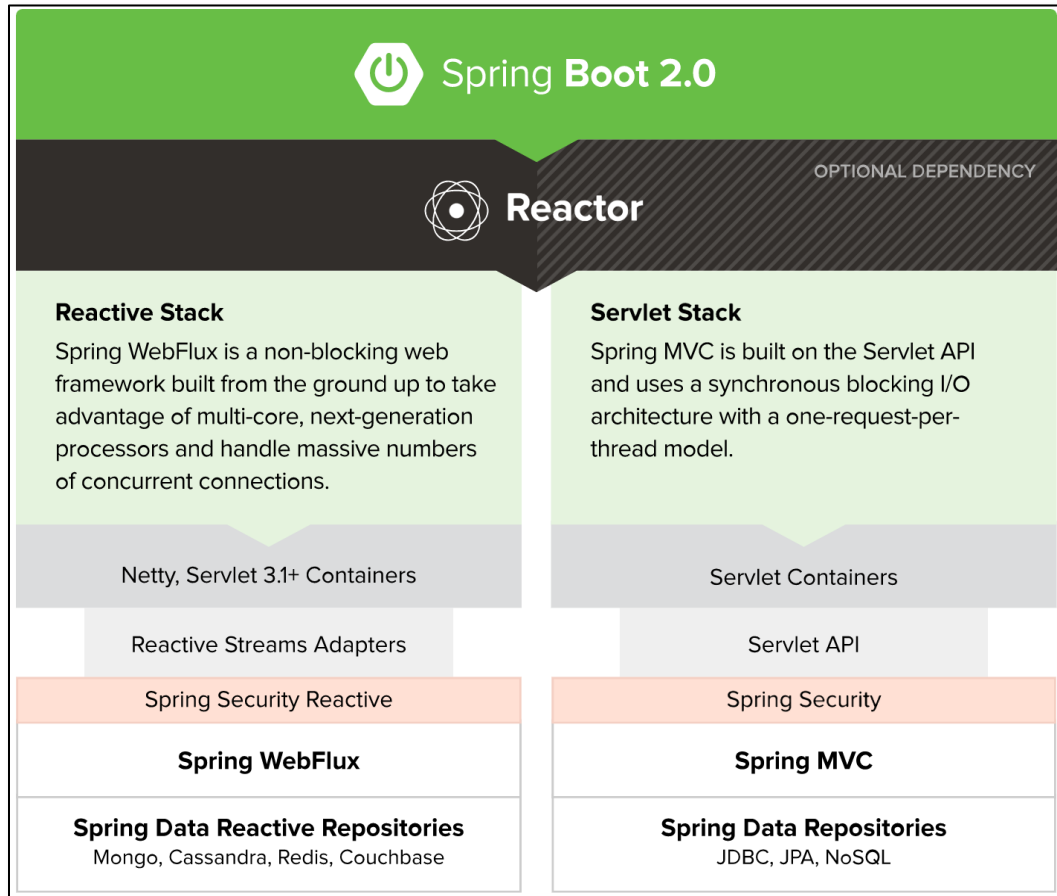
**Institute for Software
Integrated Systems**

**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Lesson

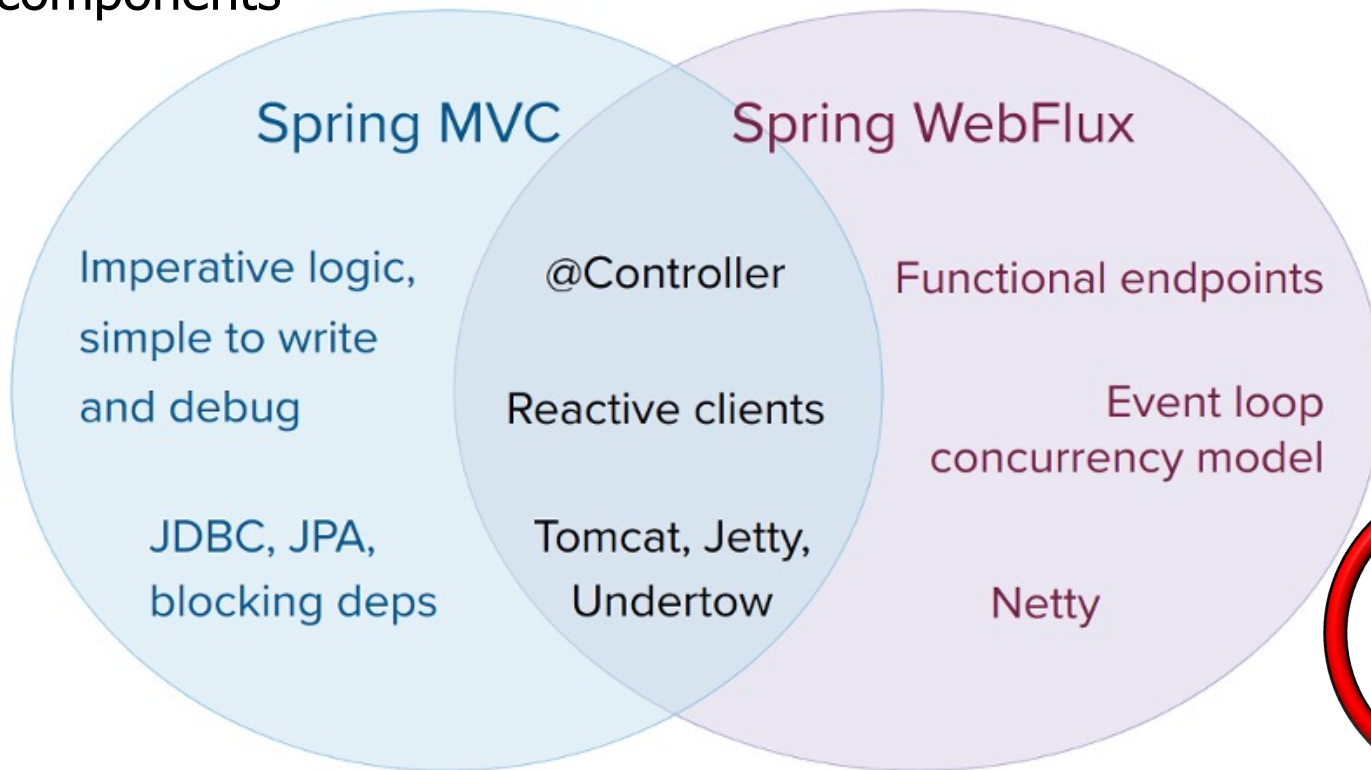
- Recognize the similarities & differences between Spring WebMVC & WebFlux frameworks supported by Spring Boot 2.0



Comparing & Contrasting Spring WebMVC & WebFlux

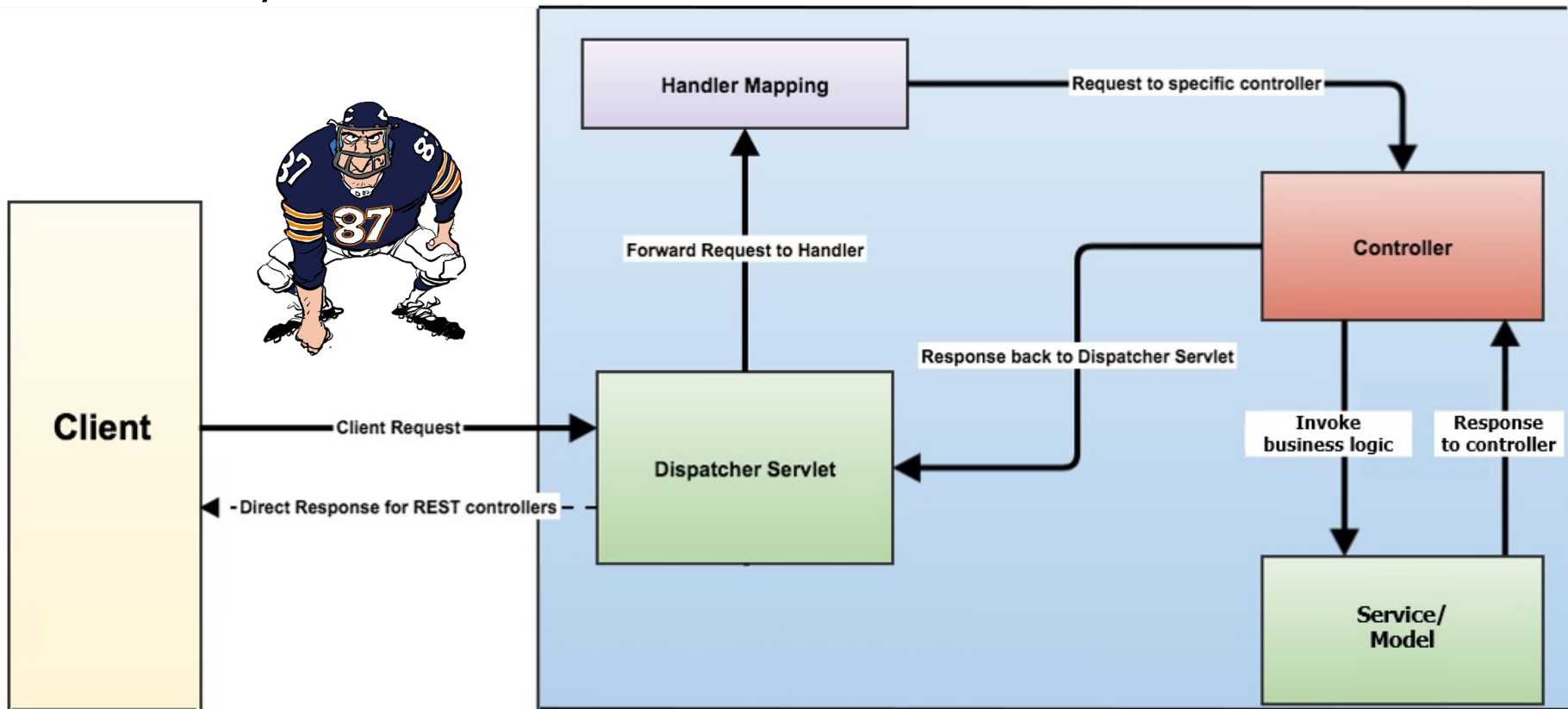
Comparing & Contrasting Spring WebMVC & WebFlux

- Spring WebMVC & WebFlux have similarities & differences wrt functionality & internal components



Comparing & Contrasting Spring WebMVC & WebFlux

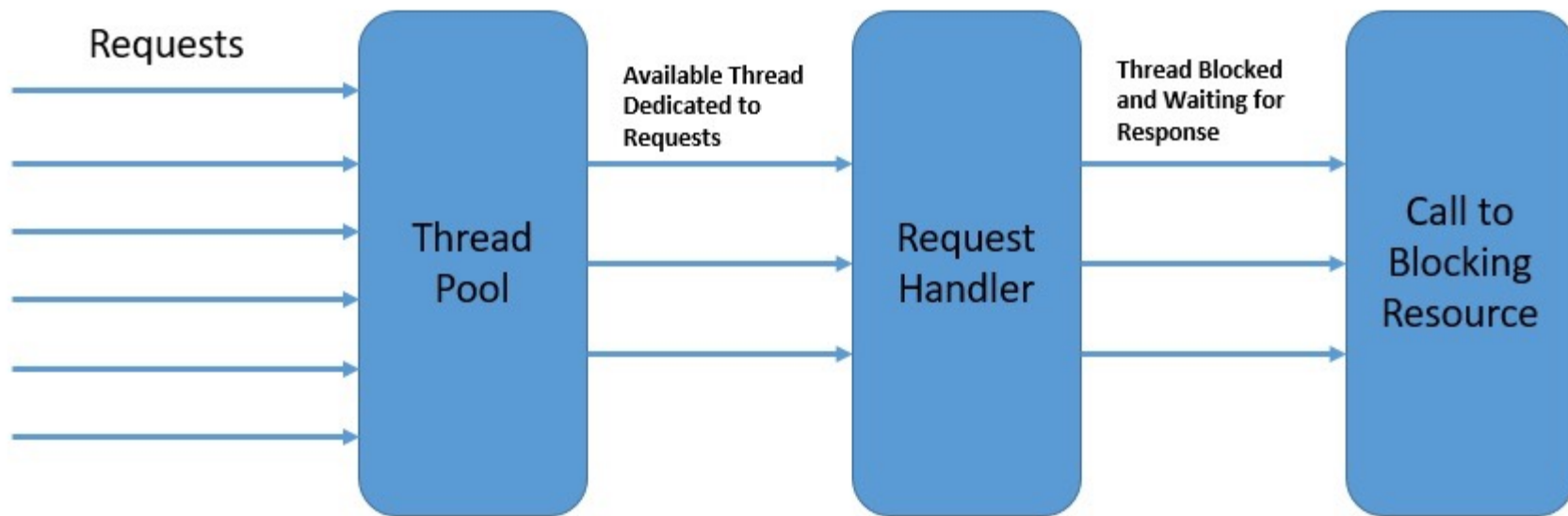
- WebMVC is sync



Built on Servlet API & uses a sync I/O w/one-thread-per-request model (by default)

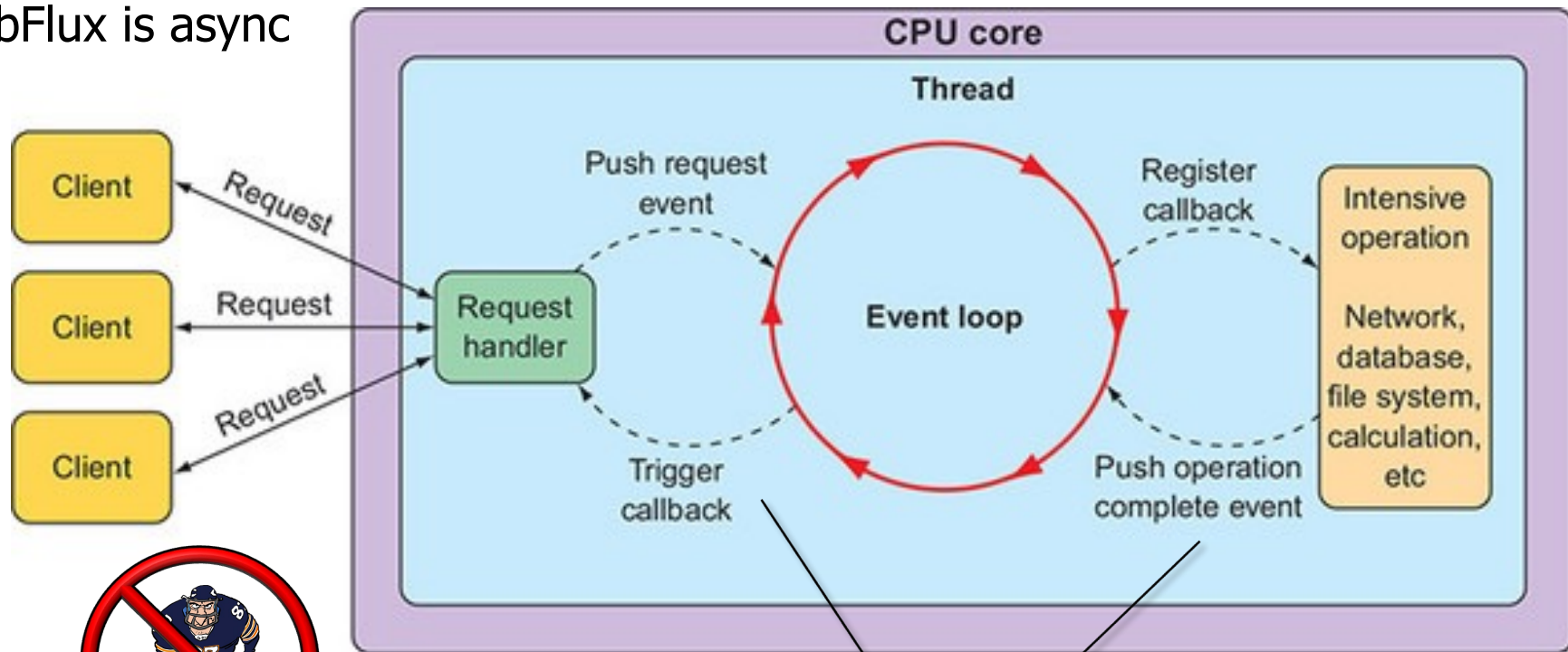
Comparing & Contrasting Spring WebMVC & WebFlux

- WebMVC is sync
- The server uses a thread-per-request, where each thread handles a single request at a time



Comparing & Contrasting Spring WebMVC & WebFlux

- WebFlux is async

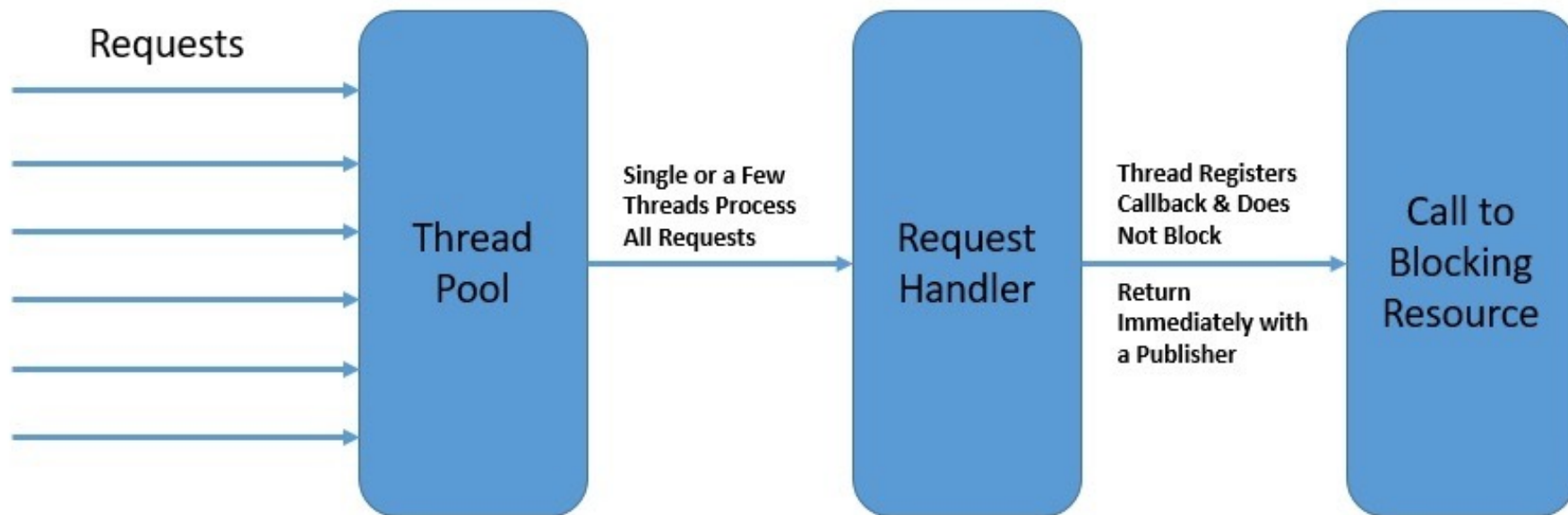


Callbacks are transparent to server code that uses Mono & Flux reactive types

Non-blocking I/O that leverages multiple cores & handles large # of connections

Comparing & Contrasting Spring WebMVC & WebFlux

- WebFlux is async
 - It uses a completely non-blocking environment that can achieve higher concurrency with better resource utilization

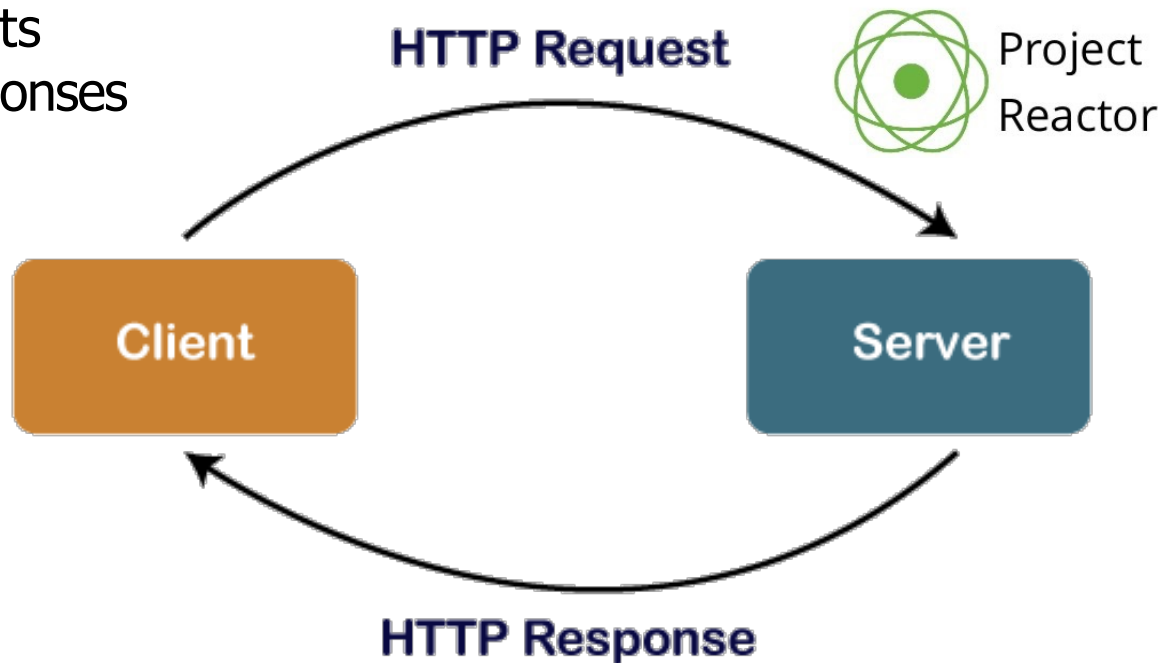


See www.baeldung.com/spring-webflux-concurrency

Accessing Mono & Flux Endpoints Seamlessly

Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses



See docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html

Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses
- WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints

```
Flux<Airport> airports = webClient
    .get()
    .uri(baseUrl + AIRPORT
        + "/" + AIRPORTS)
    .retrieve()
    .bodyToFlux(Airport.class);
```

```
Airport[] airports = restTemplate
    .getForEntity(baseUrl + AIRPORT + "/" + AIRPORTS,
        Airport[].class)
    .getBody();
```

Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses
- WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints

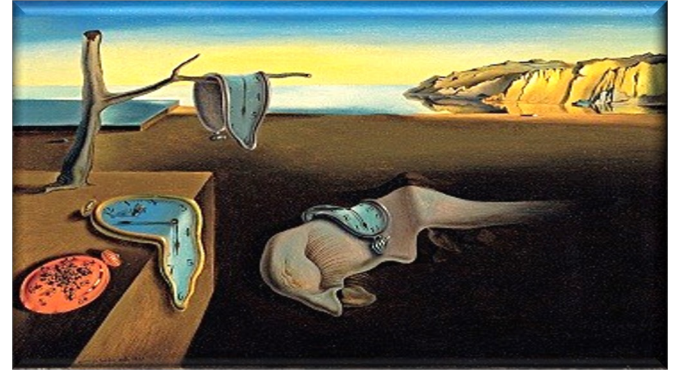
```
Flux<Airport> airports = webClient
    .get()
    .uri(baseUrl + AIRPORT
        + "/" + AIRPORTS)
    .retrieve()
    .bodyToFlux(Airport.class);
```

Use auto-wired fields here!

```
Airport[] airports = restTemplate
    .getForEntity(baseUrl + AIRPORT + "/" + AIRPORTS,
        Airport[].class)
    .getBody();
```

Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses
- WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints



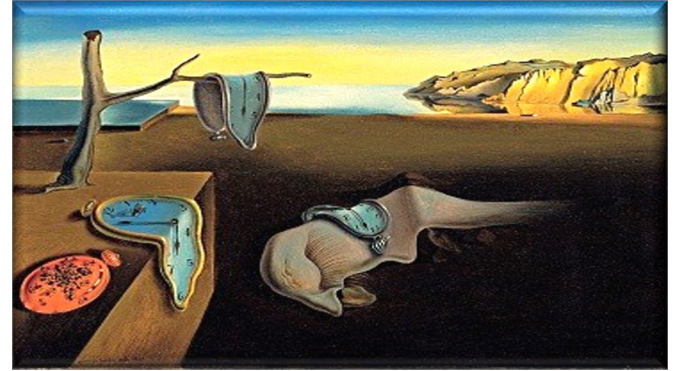
```
Airport[] airports = restTemplate
    .getForEntity(baseUrl + AIRPORT + "/" + AIRPORTS,
        Airport[].class)
    .getBody();
```

RestTemplate treats reactive types synchronously from the perspective of a client

However, no changes are required on the (reactive) server side

Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses
- WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints



```
Airport[] airports = restTemplate
    .getForEntity(baseUrl + AIRPORT + "/" + AIRPORTS,
                  Airport[].class)
    .getBody();
```

```
Flux<Airports> Flux.fromIterable
    (airports != null ? List.of(airports) : Collections.emptyList());
```

*Easy to convert back
to reactive types*

Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses
- WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints

```
Flux<Airport> airports = webClient
    .get()
    .uri(baseUrl + AIRPORT
        + "/" + AIRPORTS)
    .retrieve()
    .bodyToFlux(Airport.class);
```

WebClient leverages reactive types more effectively since responses are emitted as soon as they are available



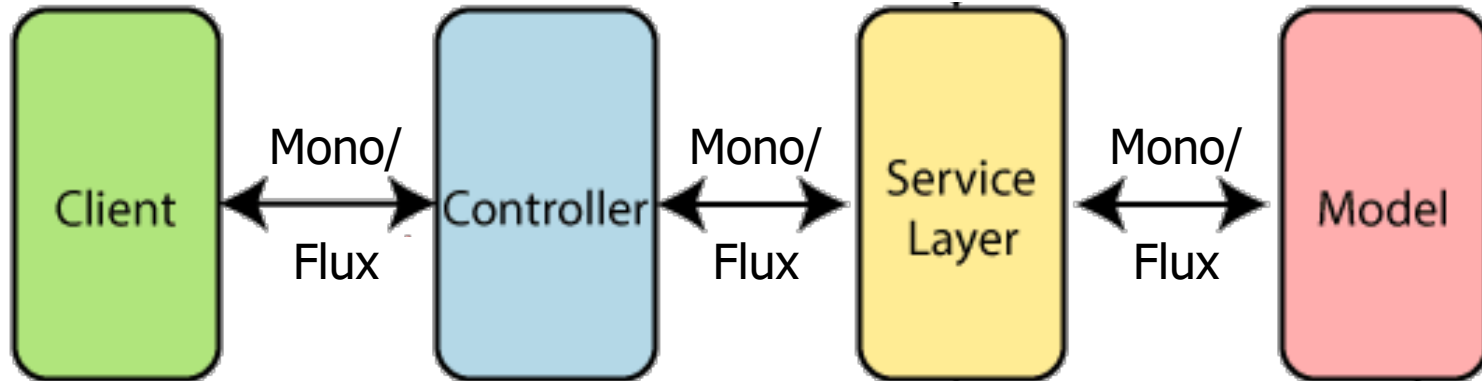
See www.baeldung.com/spring-webflux-concurrency

Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses
- WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints

```
Flux<Airport> airports = webClient  
    .get()  
    .uri(baseUrl + AIRPORT  
        + "/" + AIRPORTS)  
    .retrieve()  
    .bodyToFlux(Airport.class);
```

WebClient also enables end-to-end asynchrony



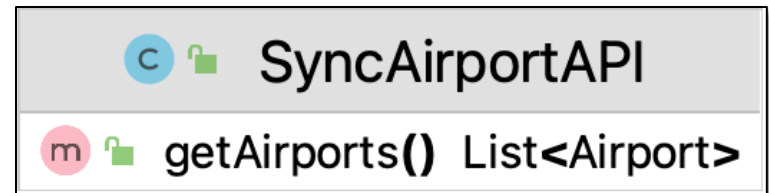
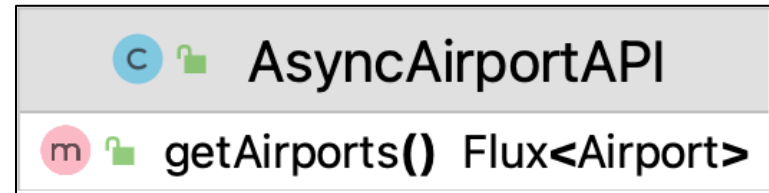
An HTTP request is not sent until subscribe() is called (& runs in thread pool)

Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses
 - WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints
- HTTP interface can also be used in Spring 6 & beyond in lieu of Web Client or RestTemplate

```
Flux<Airport> mAsyncAirports =  
    mAsyncAirportAPI.getAirports();
```

```
List<Airport> mSyncAirports =  
    mSyncAirportAPI.getAirports();
```



See www.baeldung.com/spring-6-http-interface

Accessing Mono & Flux Endpoints Seamlessly

- JSON encoding/decoding is similar for reactive WebFlux Mono/Flux types or traditional WebMVC Ref Types/List types

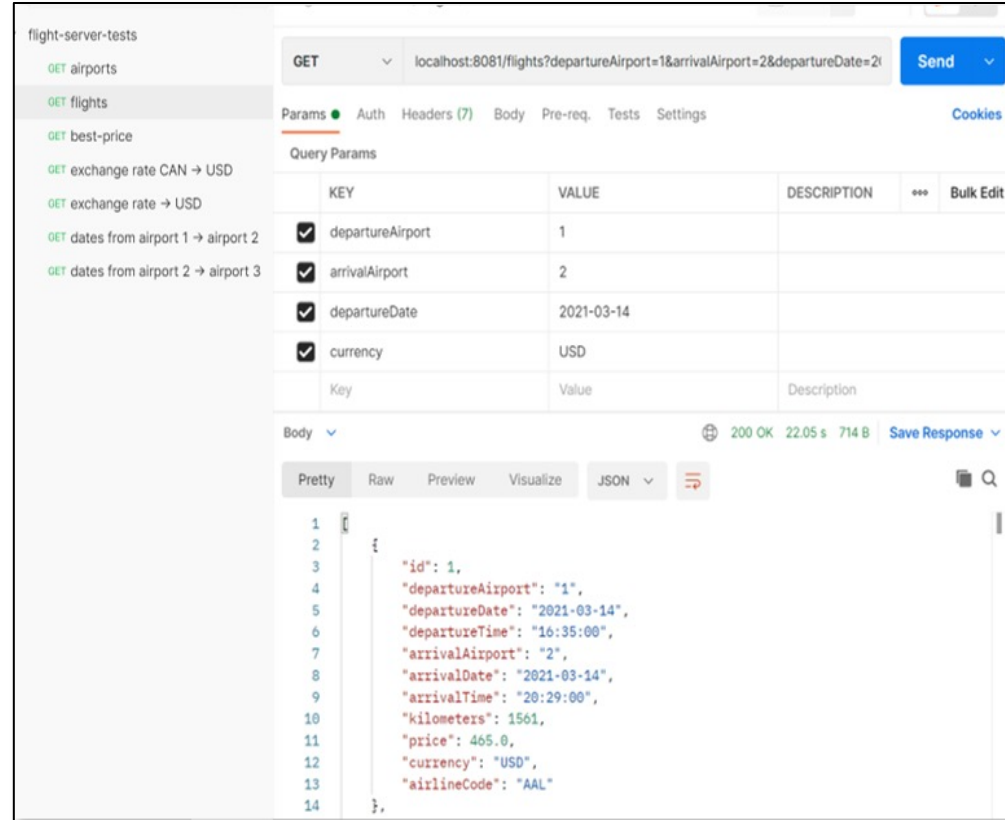


```
GET flighthost:8081/airports
```

```
[  
  {  
    "airportCode": "ALB",  
    "airportName": "Albany, NY"  
  },  
  {  
    "airportCode": "AMA",  
    "airportName": "Amarillo, TX"  
  },  
  {  
    "airportCode": "ATL",  
    "airportName": "Atlanta, GA"  
  }, ...  
]
```

Accessing Mono & Flux Endpoints Seamlessly

- JSon encoding/decoding is similar for reactive WebFlux Mono/Flux types or traditional WebMVC Ref Types/List types
- Tools like Postman can work seamlessly with either



The screenshot shows the Postman interface for a REST client. The URL is `localhost:8081/flights?departureAirport=1&arrivalAirport=2&departureDate=2021-03-14`. The request method is GET. The query parameters are:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> departureAirport	1			
<input checked="" type="checkbox"/> arrivalAirport	2			
<input checked="" type="checkbox"/> departureDate	2021-03-14			
<input checked="" type="checkbox"/> currency	USD			

The response status is 200 OK, 22.05 s, 714 B. The response body is shown in JSON format:

```
1 {
2   "id": 1,
3   "departureAirport": "1",
4   "departureDate": "2021-03-14",
5   "departureTime": "16:35:00",
6   "arrivalAirport": "2",
7   "arrivalDate": "2021-03-14",
8   "arrivalTime": "20:29:00",
9   "kilometers": 1561,
10  "price": 465.0,
11  "currency": "USD",
12  "airlineCode": "AAL"
13 }
14
```

See www.postman.com

End of Comparing & Contrasting Spring WebMVC & WebFlux