

Servlet vs Reactive Choosing the Right Stack

Rossen Stoyanchev

This talk

Servlet and Reactive stacks
Spring Framework 5

Big shift towards async
Learn about the options, make choices

About me

Spring Framework committer

Web development

Spring WebFlux from inception to GA

Pivotal[®]

Ask Me Anything

(immediately after this session)

Also featuring:

Phil Webb → Spring Boot lead

Stephane Maldini → Reactor lead

Motivation for change

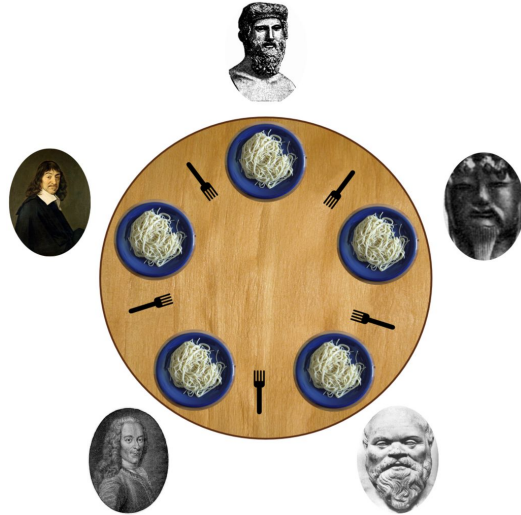
REACTIVE

REACTIVE, EVERYWHERE ??



Asynchronicity

Thread pools



Asynchronicity

Actors

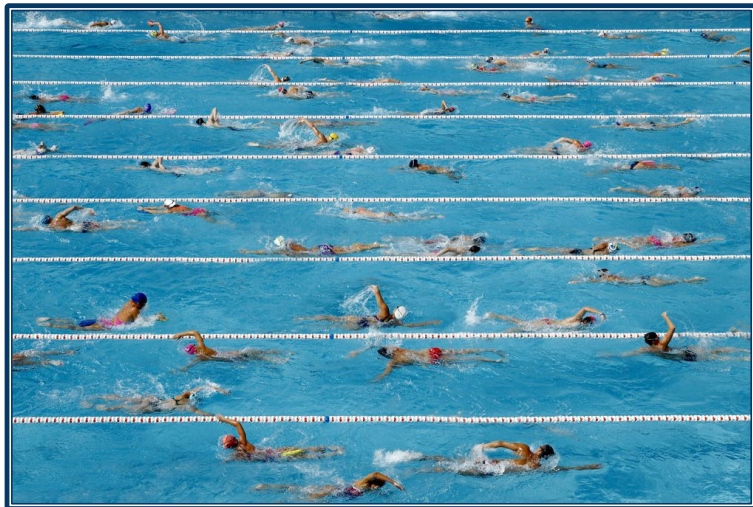
Fibers / Project Loom

Coroutines

Event loop

Thread pools

Servlet



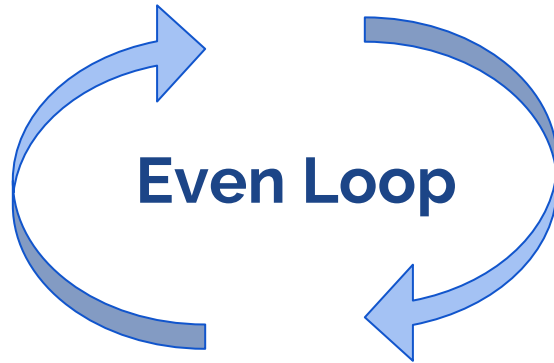
100s, 1000s threads
(blocked)

Reactive



Small, fixed # of threads
(running)

Non-blocking concurrency



Declarative composition of asynchronous logic

CompletableFuture, ReactiveX, Reactor, ...

Java 8 lambdas

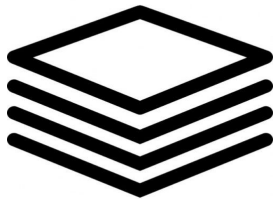
Functional programming models

An aerial, high-angle photograph of a dense urban skyline, likely New York City, featuring numerous skyscrapers and buildings. The image is in grayscale. A semi-transparent white rectangular box with a thin black border is centered over the middle of the image. Inside this box, the text "Stack choices" is written in a large, black, sans-serif font, and "and why choice matters" is written below it in a smaller, blue, italicized serif font.

Stack choices
and why choice matters

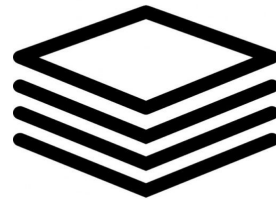
@Controller

Servlet



@Controller

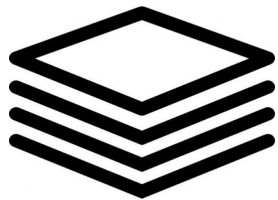
Reactive



Reactive client

@Controller

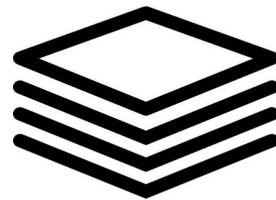
Servlet



Reactive client

@Controller

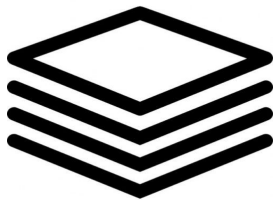
Reactive



Reactive client

@Controller

Servlet

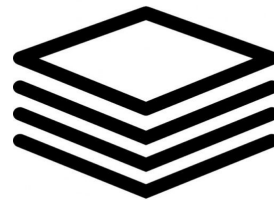


Reactive client

@Controller

Functional
endpoint

Reactive



Reactive client

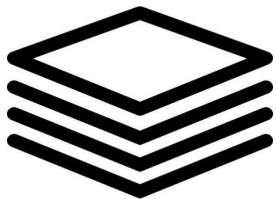
@Controller

Reactive client

@Controller

Functional
endpoint

Servlet

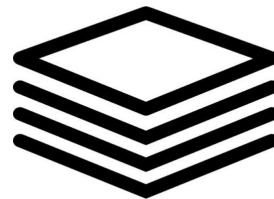


Tomcat

Jetty

Servlet container

Reactive



Tomcat

Jetty

Servlet 3.1 container

Reactive client

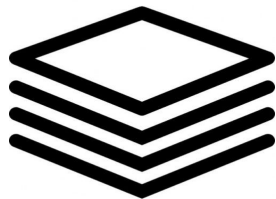
@Controller

Reactive client

@Controller

Functional
endpoint

Servlet

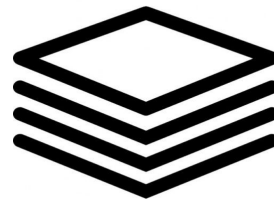


Tomcat

Jetty

Servlet container

Reactive



Netty

Tomcat

Jetty

Undertow

Servlet 3.1 container

Demo

An architectural rendering of a modern, multi-level building with a central courtyard. The building features numerous balconies, terraces, and glass railings, with people visible on various levels. The central courtyard is a large, open space with a paved walkway, a central tree, and a large, curved, white structure that appears to be a staircase or a decorative element. The overall atmosphere is bright and airy, with a focus on verticality and open space.

Stack architecture

Spring MVC

Servlet API

Blocking I/O

Tomcat, Jetty, ...



History of Servlet API

1997	1.0	
...	...	
2009	3.0	Async Servlet
2009	3.1	Servlet non-blocking I/O
...	...	

**Controller +
Reactive client**

Async Servlet

Servlet API

Blocking I/O

Tomcat, Jetty, ...

History of Servlet API

1997	1.0	
...	...	
2009	3.0	Async Servlet
2009	3.1	Servlet non-blocking I/O
...	...	

**Controller +
Reactive client**

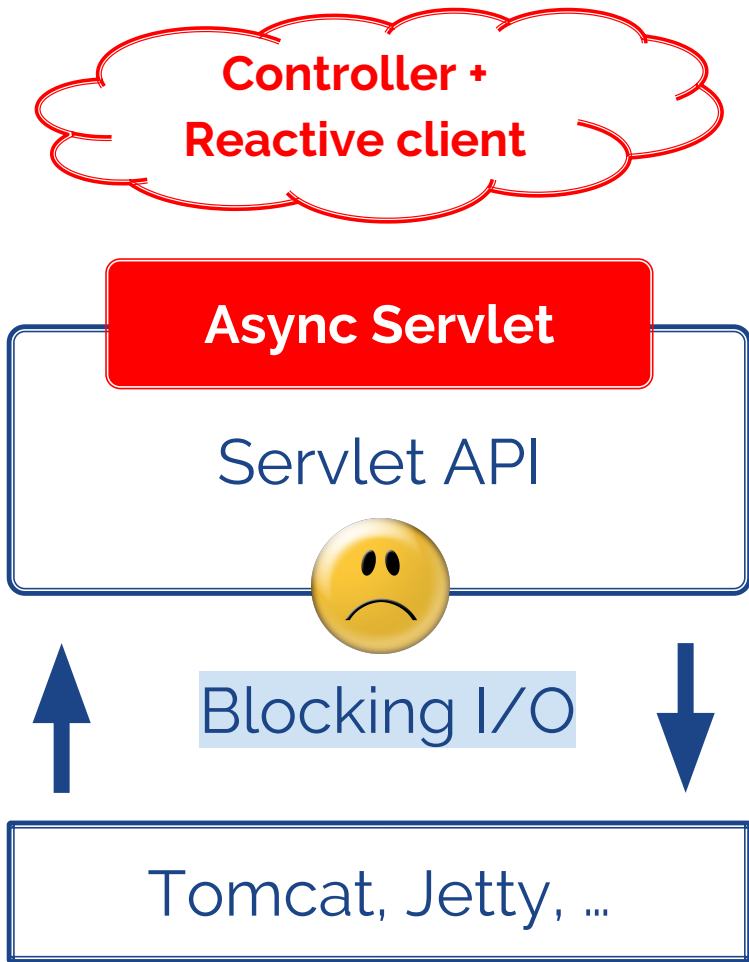
Async Servlet

Servlet API



Blocking I/O

Tomcat, Jetty, ...



**Controller +
Reactive client**

Async Servlet

Servlet API

Blocking I/O

Tomcat, Jetty, ...

Spring Web API

Reactor, Reactive Streams

Reactor
Netty

**Servlet 3.1
Non-blocking I/O**

Netty

Tomcat, Jetty, ...

Spring MVC

Servlet API

Blocking I/O

Tomcat, Jetty, ...

Spring WebFlux

Spring Web API

Reactor, Reactive Streams

Non-blocking I/O

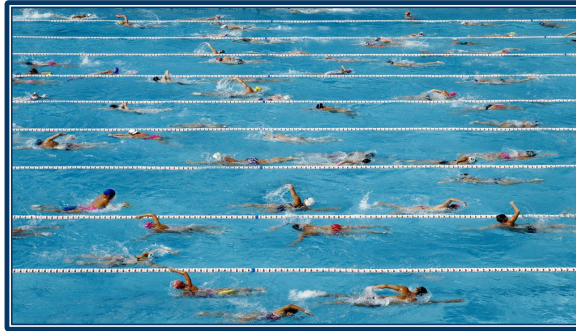
Netty

Tomcat, Jetty, ...

Reactive, non-blocking
back pressure

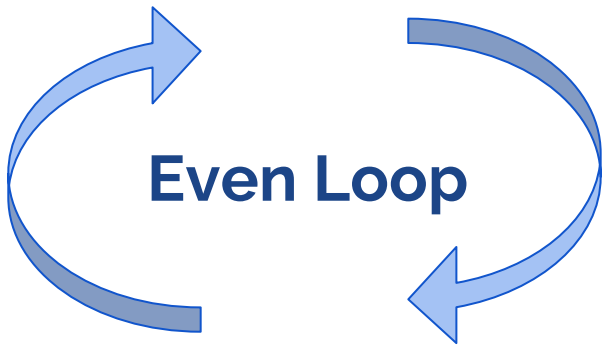


← - - - - **Blocking expected** - - - - →



Tomcat, Jetty, ...

← - - - - **Blocking not expected** - - - - →



WebFilter, WebHandler

Mono<Void>

Request

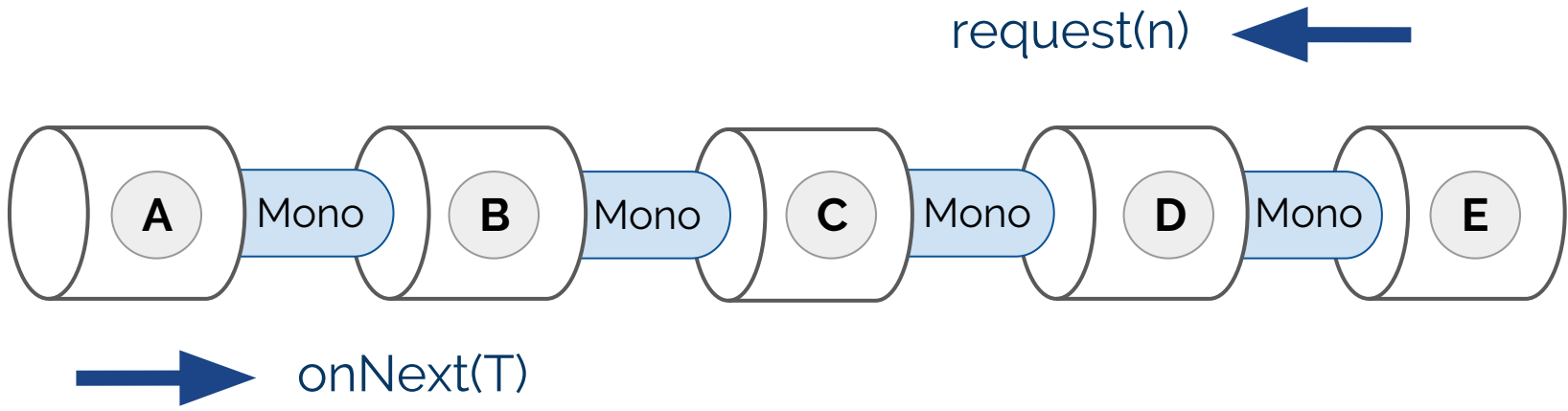
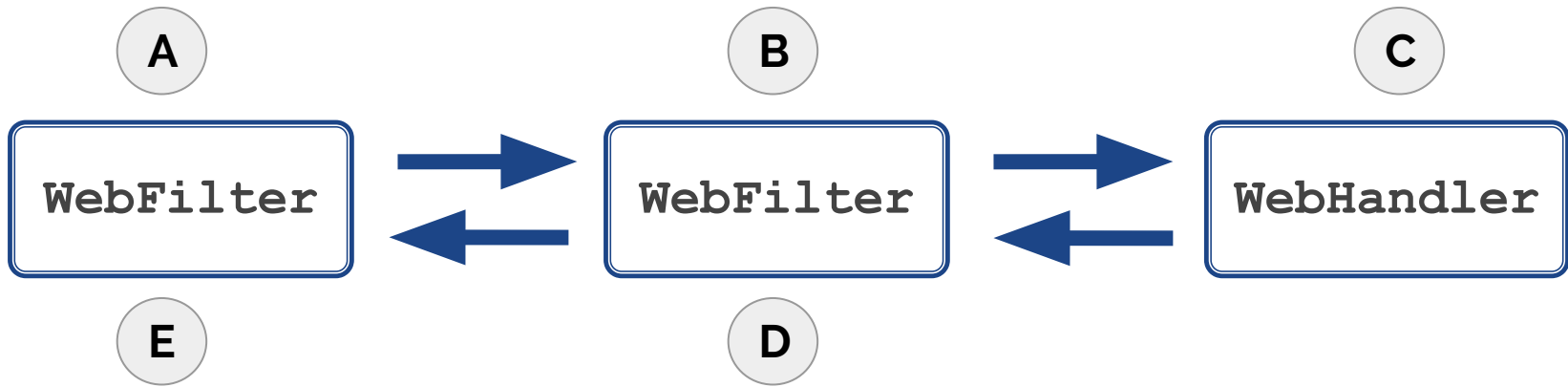
```
Flux<DataBuffer> getBody ()
```


Response

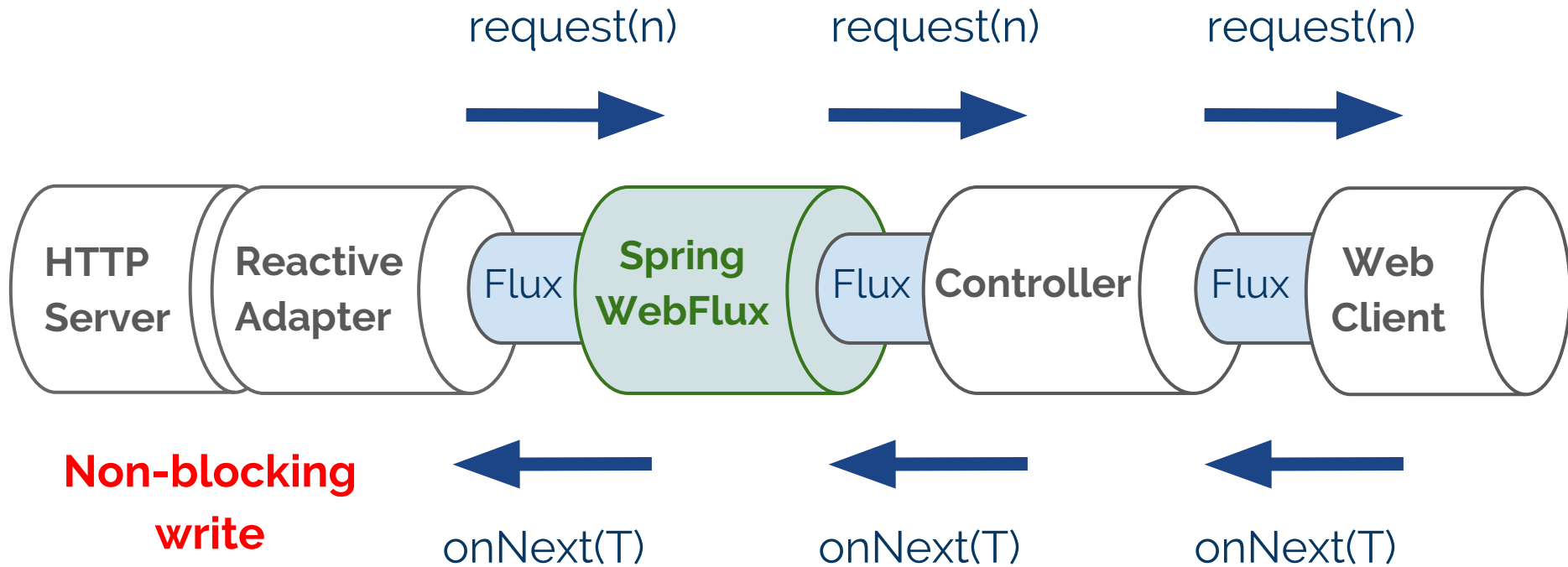
```
writeWith (Flux<DataBuffer>)
```

Codecs

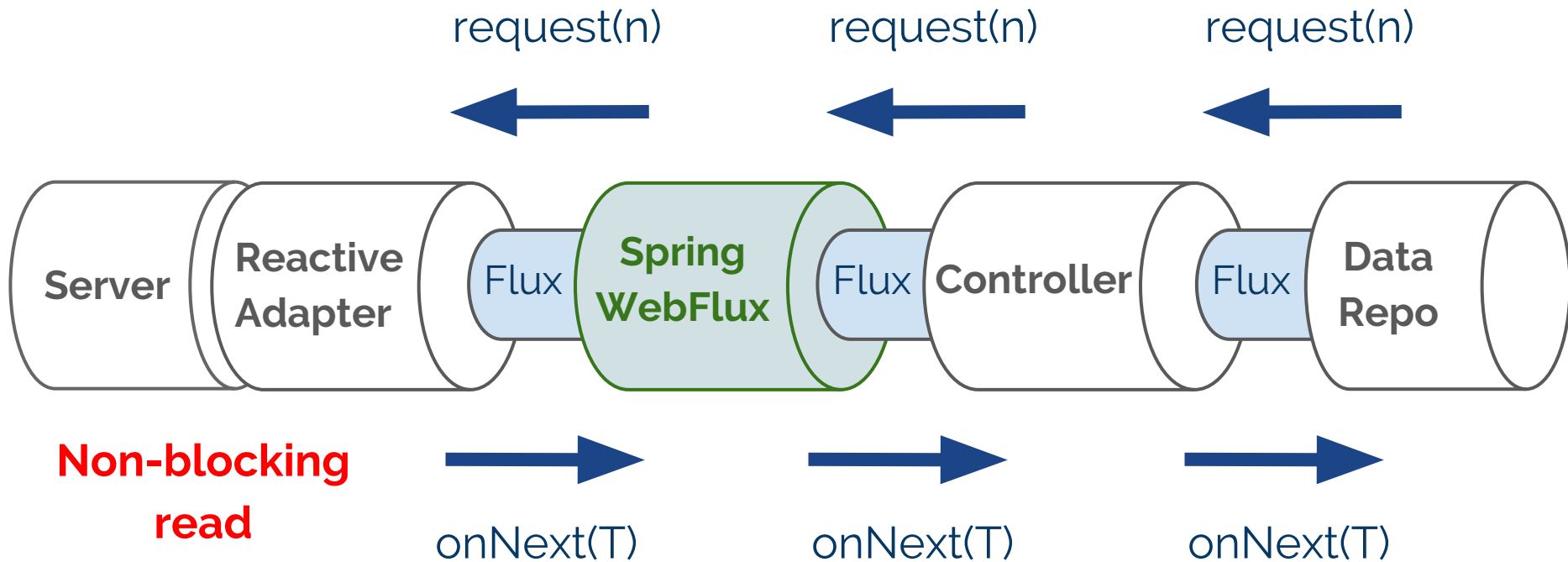
Flux<DataBuffer>  **Flux<T>**



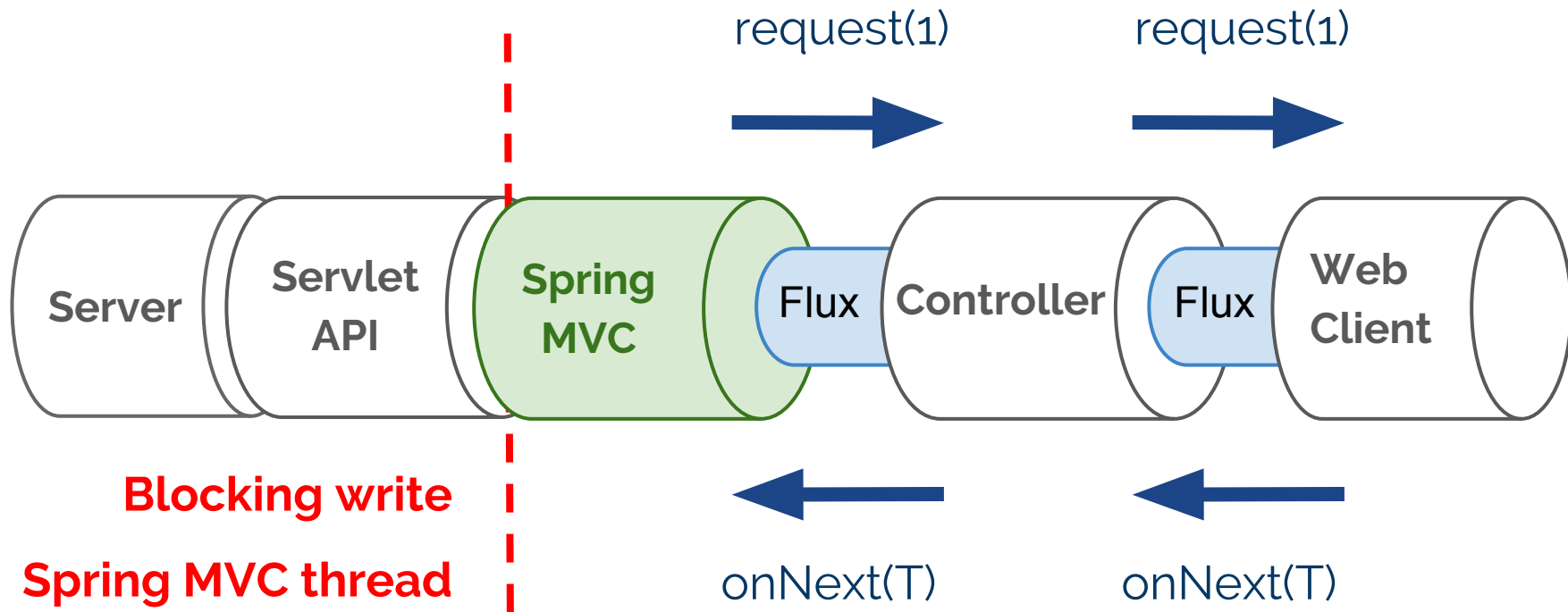
WebFlux Response



WebFlux Request



Spring MVC Response



Demo

Choosing

The right stack for the job at hand

Spring MVC

If it ain't broken, don't fix it

Imperative logic is the easiest to write and debug

Check application dependencies (JDBC, JPA)

Spring MVC + Reactive Client

Compose **remote service calls**

Reactive data (Mongo, Cassandra, Redis)

Response **stream**

Spring WebFlux

Use cases → easy to stream up or down

Scale → efficient at I/O

Programming model → functional endpoints

Spring WebFlux Performance

Efficient scale with less hardware resources

It's not about speed

Latency is key to exhibit differences

Thank You

[https://github.com/
rstoyanchev/demo-reactive-spring](https://github.com/rstoyanchev/demo-reactive-spring)



rstoya05