AIST GROUP

Spring Web Flux

By: Səlminaz Kərimli

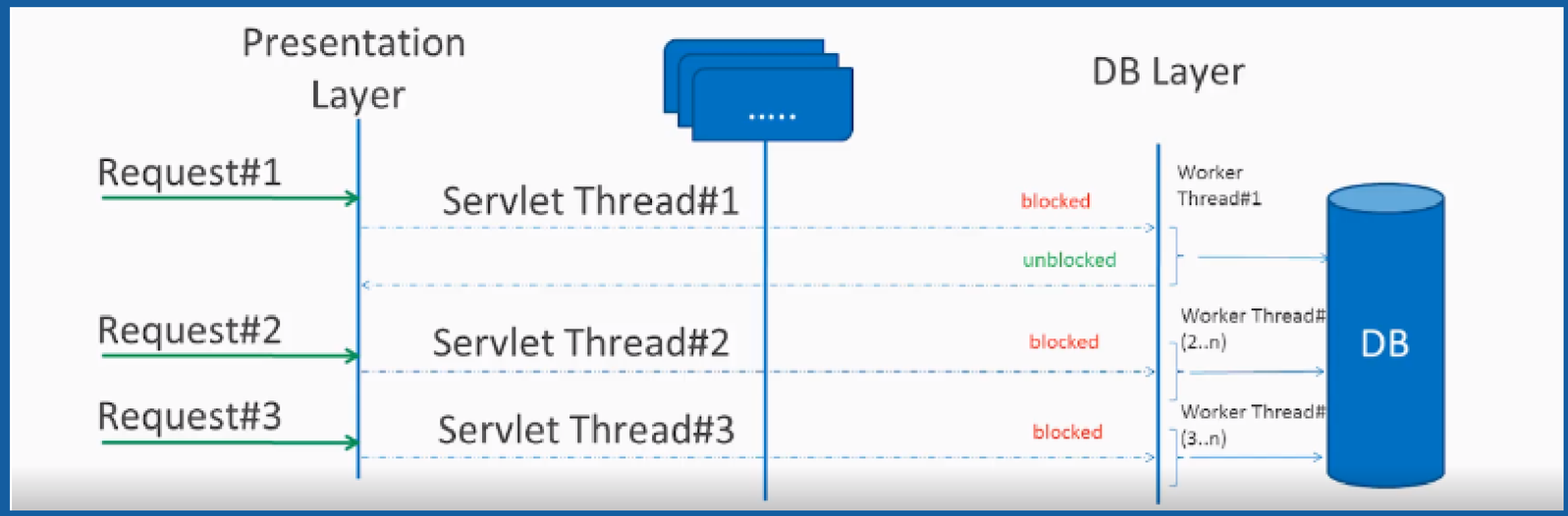
Introduction

The reactive-stack web framework, Spring WebFlux, has been added to Spring 5. It is fully non-blocking, supports reactive streams back pressure, and runs on such servers as Netty, Undertow, and Servlet 3.1+ containers

Reactive Programming

Reactive programming is a programming paradigm that promotes an asynchronous, non-blocking, event-driven approach to data processing. Reactive programming involves modeling data and events as observable data streams and implementing data processing routines to react to the changes in those streams.

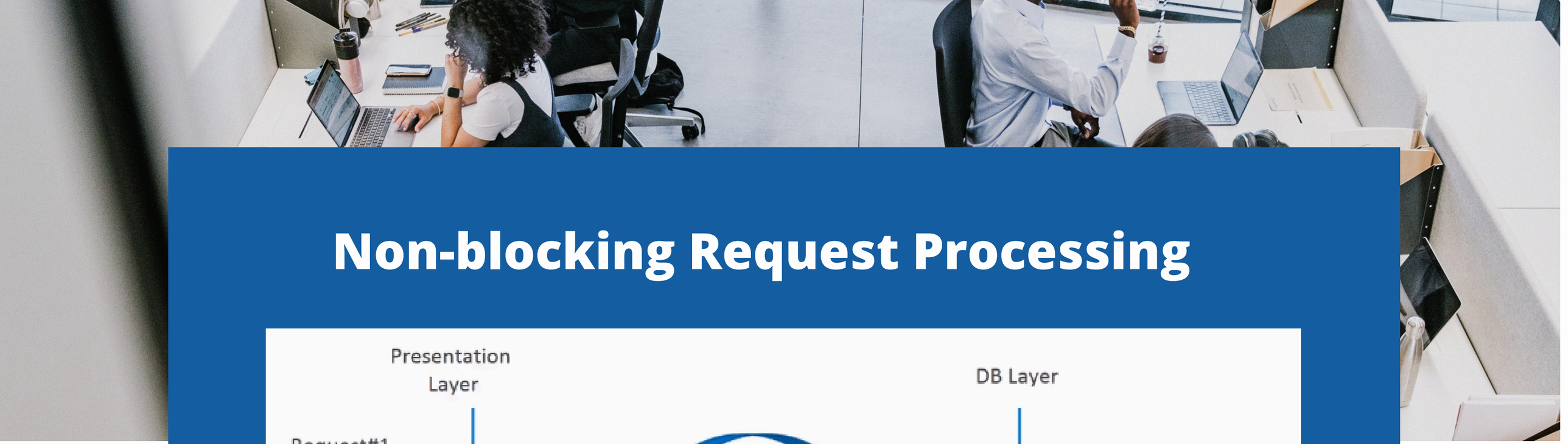
Blocking Request Processing



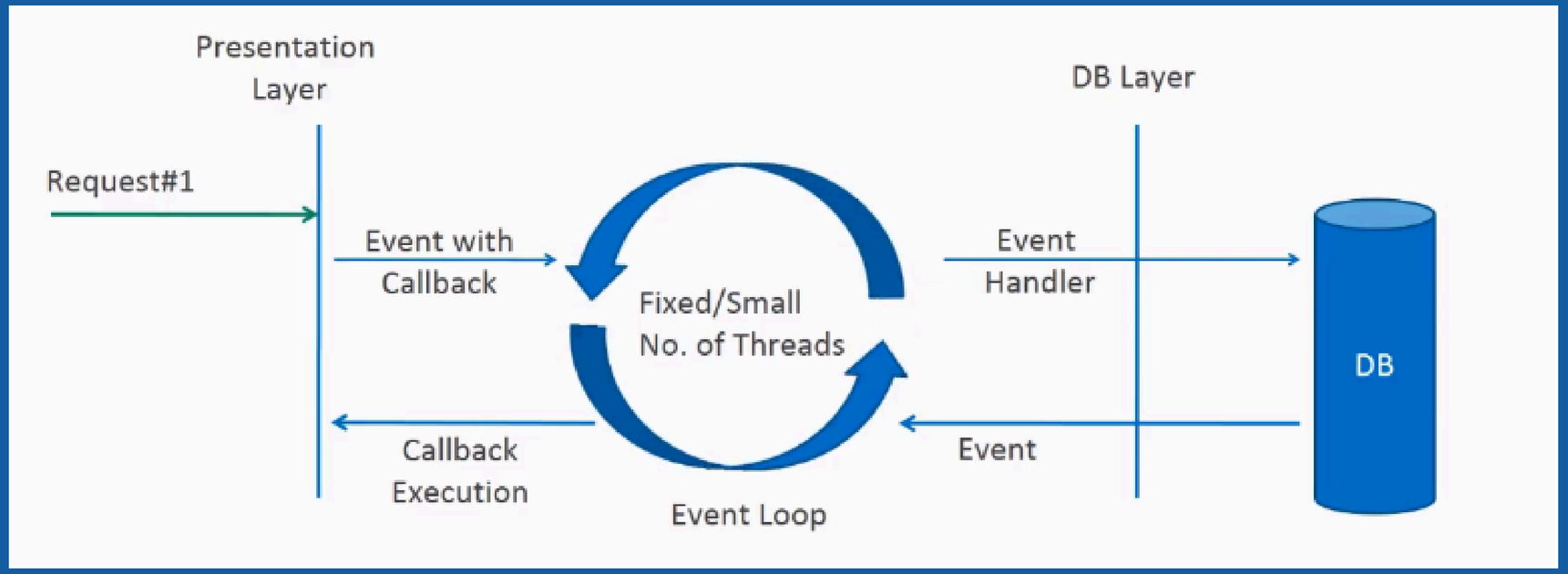
Blocking Request Processing

In traditional MVC applications, a new servlet thread is created (or obtained from the thread pool) when a request comes to the server. It delegates the request to worker threads for I/O operations such as database access etc. During the time worker threads are busy, the servlet thread (request thread) remains in waiting status, and thus it is blocked. It is also called synchronous request processing.

As a server can have some finite number of request threads, it limits the server's capability to process that number of requests at maximum server load. It may hamper the performance and limit the full utilization of server capability.



Non-blocking Request Processing



Non-blocking Request Processing

In non-blocking or asynchronous request processing, no thread is in waiting state. There is generally only one request thread receiving the request.

All incoming requests come with an event handler and callback information. Request thread delegates the incoming requests to a thread pool (generally a small number of threads) which delegates the request to its handler function and immediately starts processing other incoming requests from the request thread.

When the handler function is complete, one thread from the pool collects the response and passes it to the call back function.

Non-blocking nature of threads helps in scaling the performance of the application. A small number of threads means less memory utilization and less context switching.

What is Reactive Programming?

The term, “reactive,” refers to programming models that are built around reacting to changes. It is built around the publisher-subscriber pattern (observer pattern). In the reactive style of programming, we make a request for resources and start performing other things. When the data is available, we get the notification along with data in the callback function. The callback function handles the response as per application/user needs.

One important thing to remember is back pressure. In non-blocking code, back-pressure controls the rate of events so that a fast producer does not overwhelm its destination.

Reactive web programming is well-suited for applications that involve streaming data and real-time interactions. By using non-blocking and event-driven mechanisms, we can design excellent solutions for these applications.

However, reactive programming can also benefit traditional CRUD applications, it is worth mentioning that while reactive APIs offload tasks to non-blocking threads, they also require proper thread management. Incorrect handling of threads or blocking operations within a reactive context can lead to thread contention and performance issues.

Also, reactive programming adds more unnecessary complexity without substantial gains in low concurrent traditional applications. For such applications, a traditional synchronous approach may be more straightforward and suitable.

If you are developing the next Facebook or Twitter with lots of data, real-time analytics applications, chat applications, or live update websites, a reactive API might be just what you are looking for.

Reactive Streams API

The new Reactive Streams API was created by engineers from Netflix, Pivotal, Lightbend, RedHat, Twitter, and Oracle, among others .
It defines four interfaces:

01

Publisher

The publisher emits a sequence of events to subscribers according to the demand received from its subscribers. A publisher can serve multiple subscribers.

02

Subscriber

Receives and processes events emitted by a Publisher. Please note that no notifications will be received until Subscription#request(long) is called to signal the demand. It has four methods to handle various kinds of responses received.

03

Subscription

Defines a one-to-one relationship between a Publisher and a Subscriber. It is used to both signal desire for data and cancels demand (and allow resource cleanup).

04

Processor

Represents a processing stage consisting of both a Subscriber and a Publisher and obeys both contracts.

Interface methods

Publisher.java

```
public interface Publisher<T> {
  public void subscribe(Subscriber<? super T> s);
}
```

Subscription.java

```
public interface Subscription<T> {
public void request(long n);
public void cancel();
}
```

Subscriber.java

```
public interface Subscriber<T> {
  public void onSubscribe(Subscription s);
  public void onNext(T t);
  public void onError(Throwable t);
  public void onComplete();
}
```

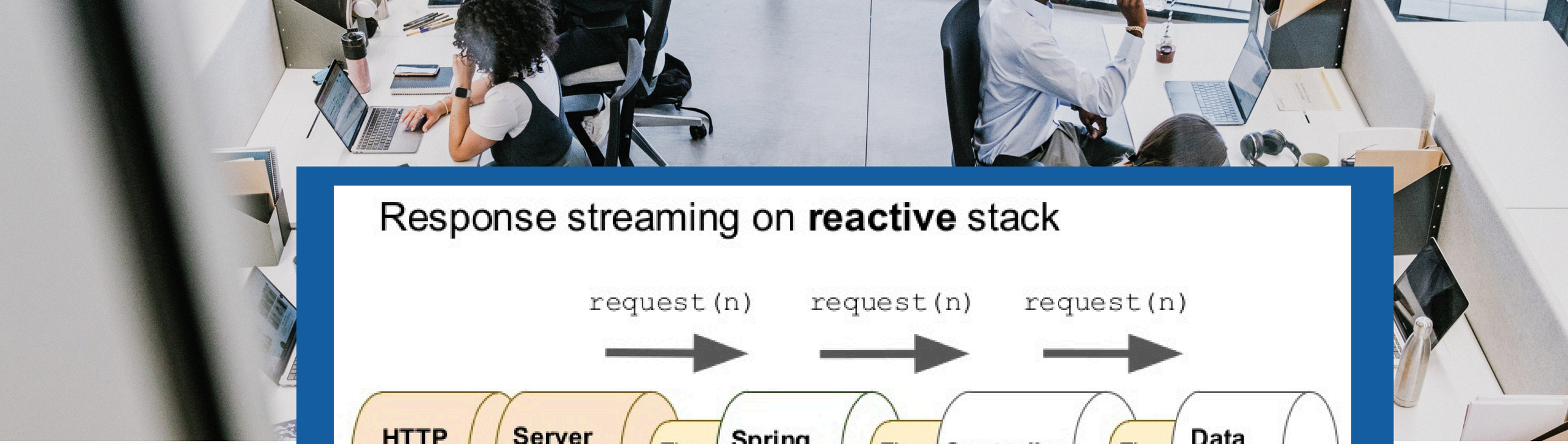
Processor.java

```
public interface Processor<T, R> extends Subscriber<T>,
Publisher<R> {}
```

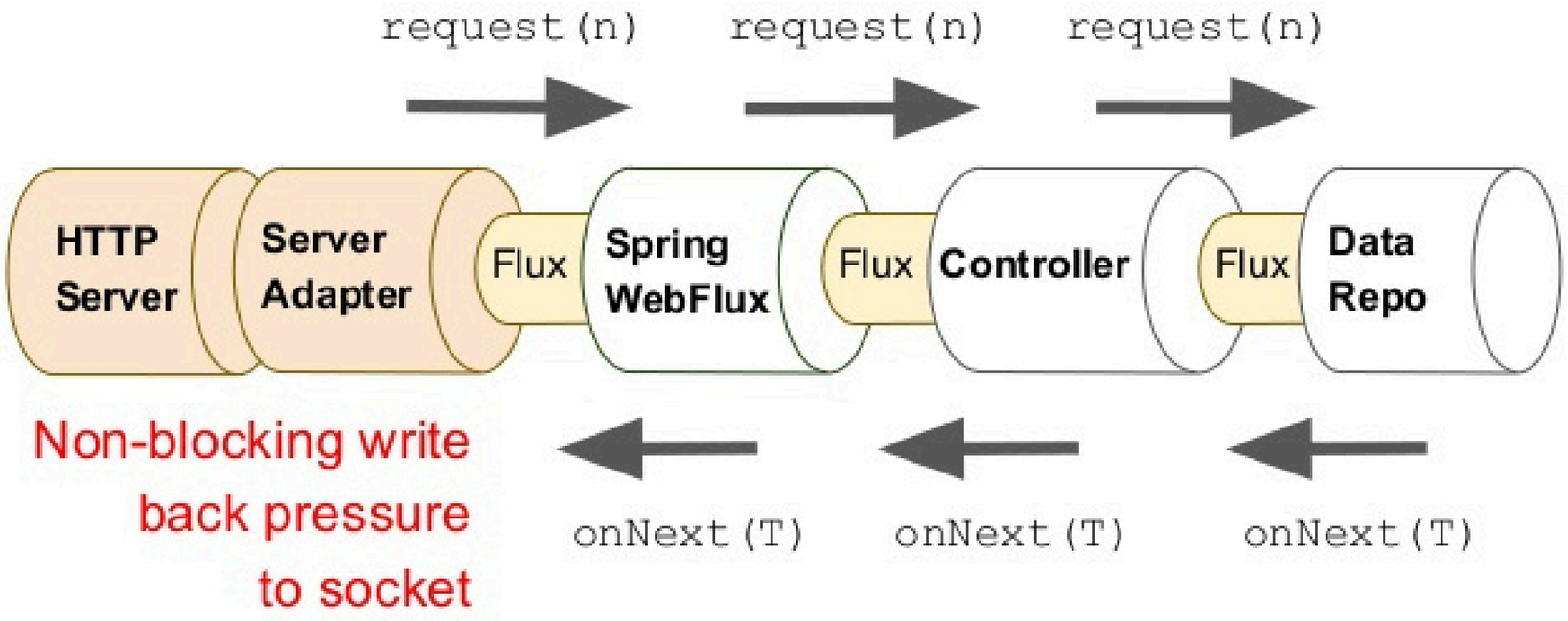
Two popular implementations of reactive streams are RxJava and Project Reactor

What is Spring WebFlux?

Spring WebFlux is a parallel version of Spring MVC and supports fully non-blocking reactive streams. It supports the back pressure concept and uses Netty as the inbuilt server to run reactive applications. If you are familiar with the Spring MVC programming style, you can easily work on webflux also.
Spring webflux uses project reactor as the reactive library. Reactor is a Reactive Streams library; therefore, all of its operators support non-blocking back pressure. It is developed in close collaboration with Spring.



Response streaming on **reactive** stack



Spring WebFlux heavily uses two publishers :

Mono: Returns 0 or 1 element.

A Reactive Streams Publisher with basic rx operators that emits at most one item via the onNext signal then terminates with an onComplete signal (successful Mono, with or without value), or only emits a single onError signal (failed Mono).

```
Mono<String> mono = Mono.just("Alex");
Mono<String> mono = Mono.empty();
```

Flux: Returns 0...N elements.

A Flux can be endless, meaning that it can keep emitting elements forever. Also it can return a sequence of elements and then send a completion notification when it has returned all of its elements.

```
Flux<String> flux = Flux.just("A", "B", "C");
Flux<String> flux = Flux.fromArray(new String[]{"A", "B", "C"});
Flux<String> flux = Flux.fromIterable(Arrays.asList("A", "B", C"));

//To subscribe call method
flux.subscribe();
```

To build a truly non-blocking application, we must aim to create/use all of its components as non-blocking i.e. client, controller, middle services and even the database. If one of them is blocking the requests, our aim will be defeated.

Advantages of WebFlux

- **Scalability**: It can handle more requests concurrently using less resources due to non-blocking I/O. This improves scalability.
- **Responsiveness**: Applications remain highly responsive even under heavy load since requests are not blocked waiting for I/O.
- **Resource efficiency**: Fewer threads are required to handle the same number of requests than blocking applications.
- **Resiliency**: Failure of one request impacts only some. Backpressure prevents overloading.

Disadvantages of WebFlux

- **Debugging challenges**: Asynchronous code can be more difficult to debug compared to sequential blocking code.
- **Complexity**: Reactive code involving streams, and callbacks can become more complex than simple blocking code.
- **Latency sensitivity**: Reactive applications require consistently low latency to perform well compared to blocking.
- **Migration effort**: Existing blocking code needs to be refactored for non-blocking APIs, which requires time and effort.

Use Cases of WebFlux

- Financial trading systems
- Real-time collaboration tools
- Video/audio streaming Platforms
- Live tracking systems