

The logo for Devoxx is located on the left side of the slide. It consists of the word "Devoxx" in a stylized, white, blocky font. The "x"s are crossed with orange lines. A small "TM" trademark symbol is positioned above the top right of the "x"s.

Developing Reactive applications with Reactive Streams and Java 8

Brian Clozel / Sébastien Deleuze
Pivotal








Brian Clozel

- Lyon, France +  la cordée
- Spring Framework
- Spring Boot
- @bclozel



Sébastien Deleuze

- Remote worker at **Pivotal** from  la cordée
 -  Reactor
 -  Spring Framework 5
 -  Kotlin support
-  mix-IT conference staff member



Agenda

- Introduction to Reactive
- Live coding with Reactor Core 3
- Coffee break (20 minutes)
- Building a Reactive application with Spring Boot
- Live coding



Why going Reactive?

More for **scalability** and **stability** than for speed



Reactive, what is it?

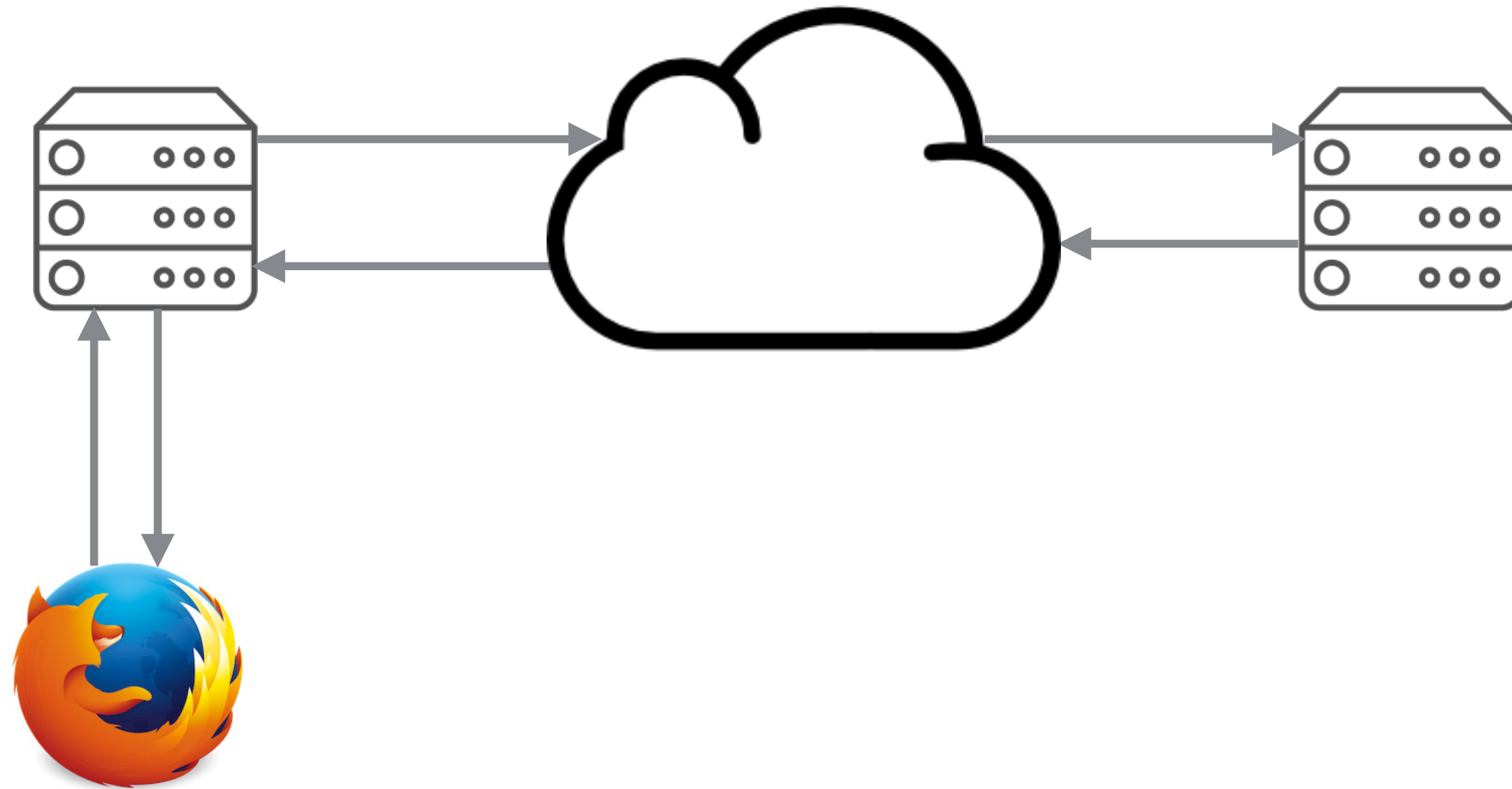
Reactive is used to broadly define **event-driven systems**



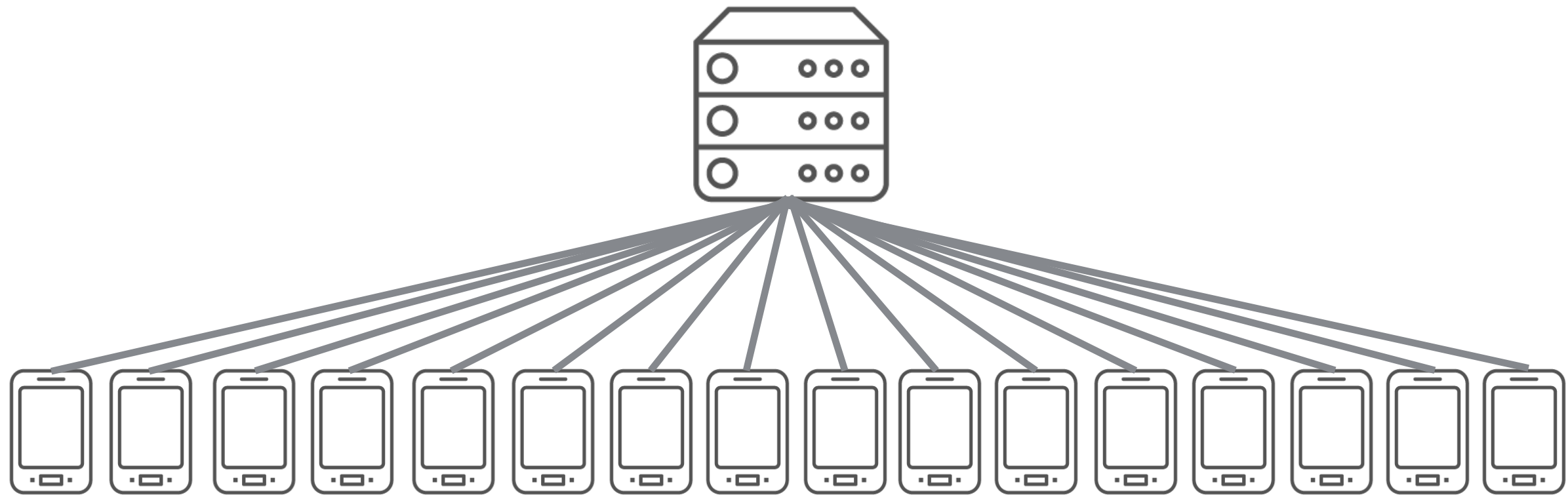
Reactive programming

Moving imperative logic to **async, non-blocking, functional**-style code, in particular when interacting with external resources

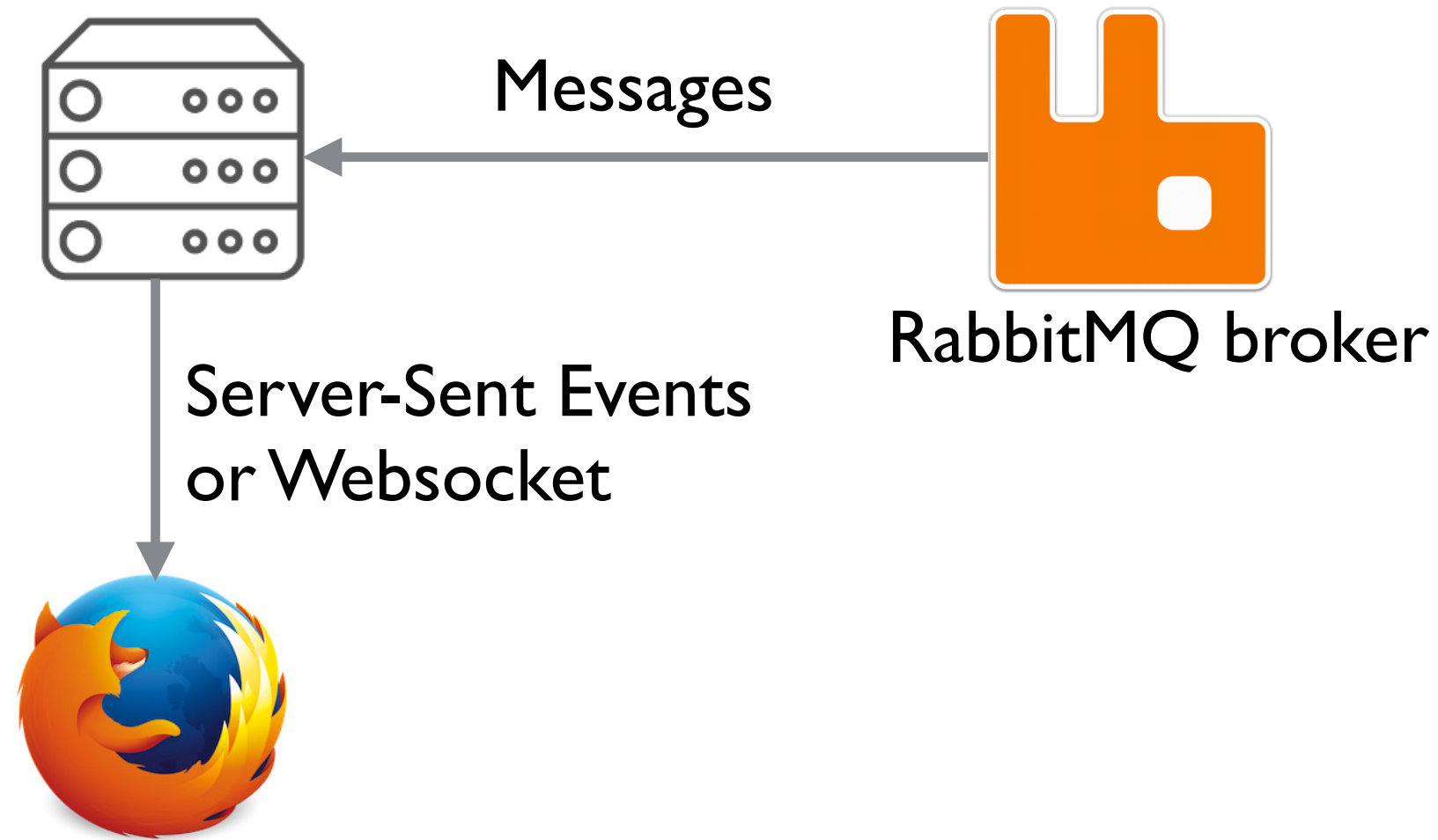
Use case: remote call with latency



Use case: serve a lot of slow clients



Use case: push events to the client





Other use cases

- Live database queries
- Mobile (Android)
- Big Data
- Real time analytics
- HTTP/2
- ...



Can't we just use Java 8 lambdas?

```
public class CallbackHell {  
  
    public void callbackHell() {  
  
        asyncMethod(a ->  
            asyncMethod(b ->  
                asyncMethod(c ->  
                    asyncMethod(  
                        d -> System.out.println("Values received: " + a + "," + b + "," + c + "," + d),  
                        dEx -> System.err.println("An error occurred: " + dEx)  
                    )  
                , cEx -> System.err.println("An error occurred: " + cEx))  
            , bEx -> System.err.println("An error occurred: " + bEx)),  
        aEx -> System.err.println("An error occurred: " + aEx));  
    }  
  
    private void asyncMethod(Consumer<Object> success, Consumer<? super Throwable> failure) {  
        // ....  
    }  
}
```

Can't we just use Java 8 types?

Type	Non-blocking	Streaming
Future<T>		
CompletableFuture<T>	X	
Stream<T>		X
InputStream / OutputStream		X

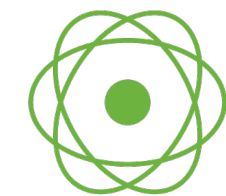


We need tools

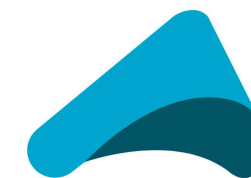
Reactive Streams & Reactive APIs



RxJava



Reactor Core



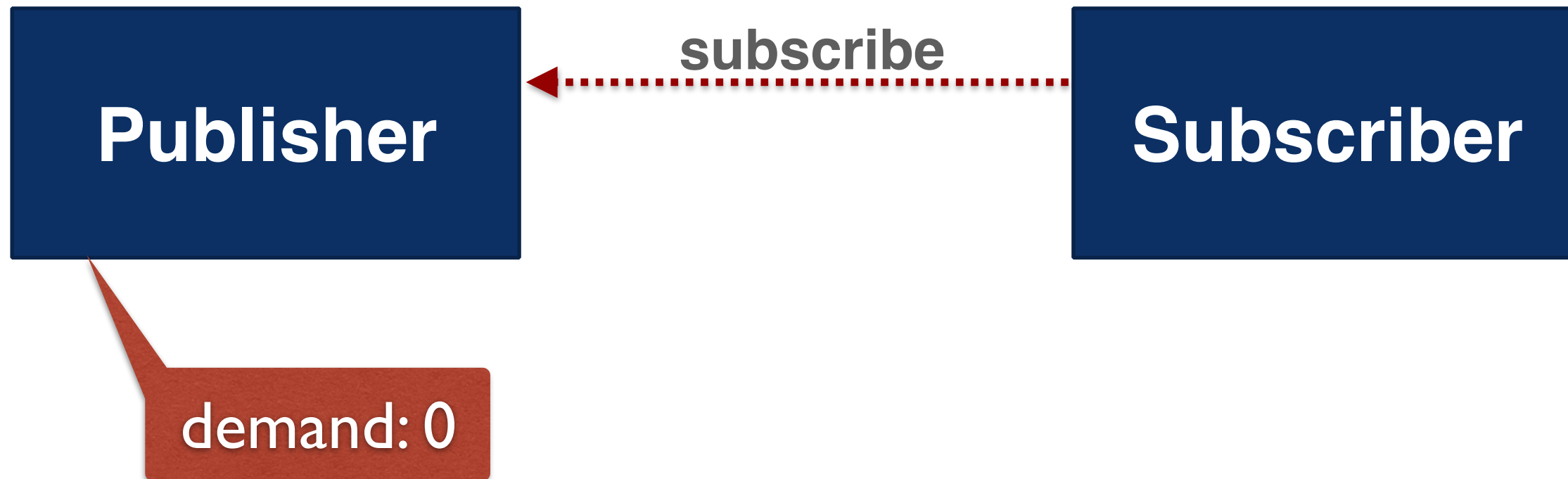
Akka Streams



Reactive Streams

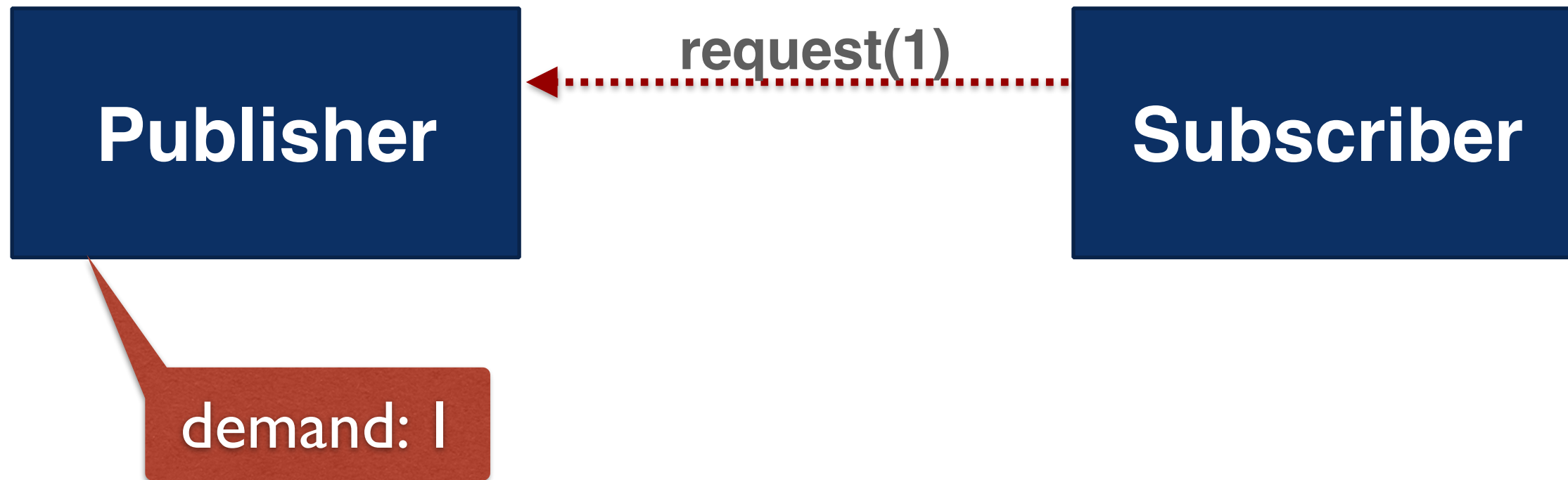
- **Reactive Streams** is a contract for asynchronous stream processing with **non-blocking back pressure** handling
- De-facto standard for the **behaviour** of reactive libraries and for **interoperability**
- Co-designed by Netflix, Lightbend, Pivotal, RedHat, Kaazing, Twitter, and many others
- Implemented by RxJava 2, Reactor, Akka Stream ...

Back-pressure

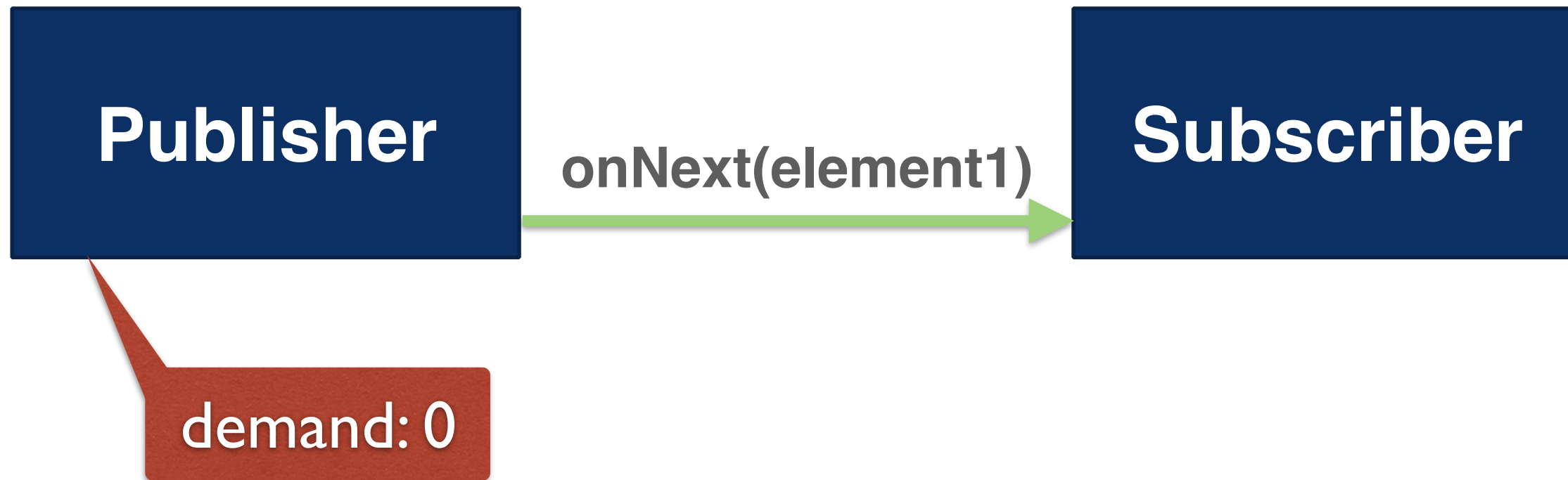




Back-pressure



Back-pressure





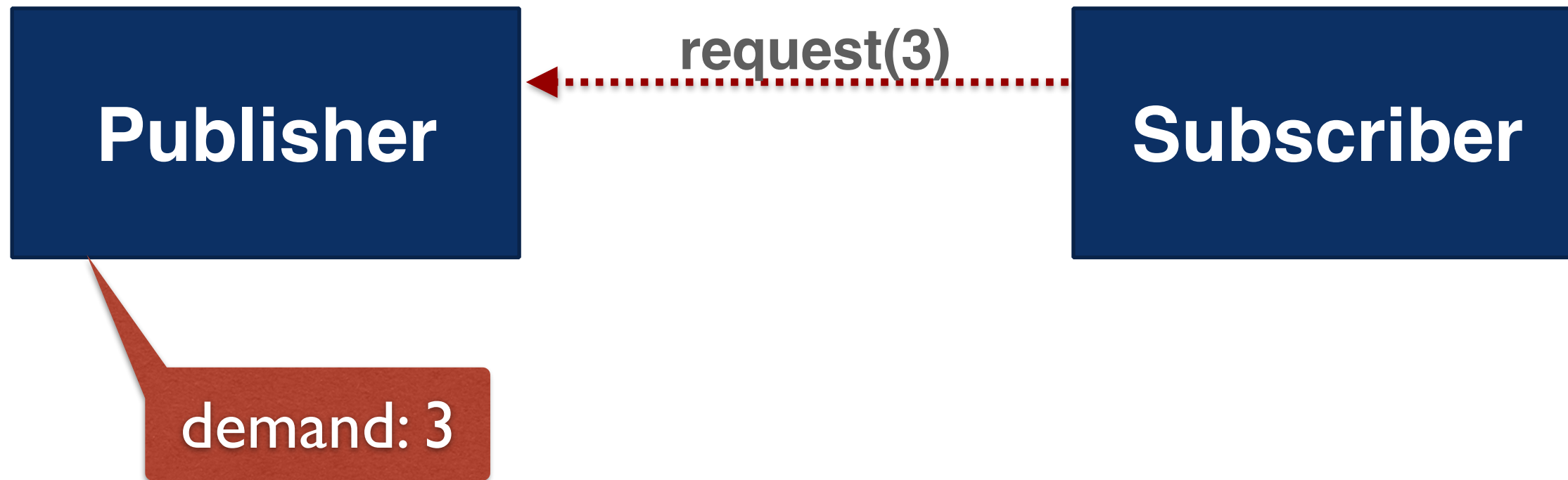
Back-pressure



Publisher **is not allowed** to send new elements,
even if new ones are ready.



Back-pressure



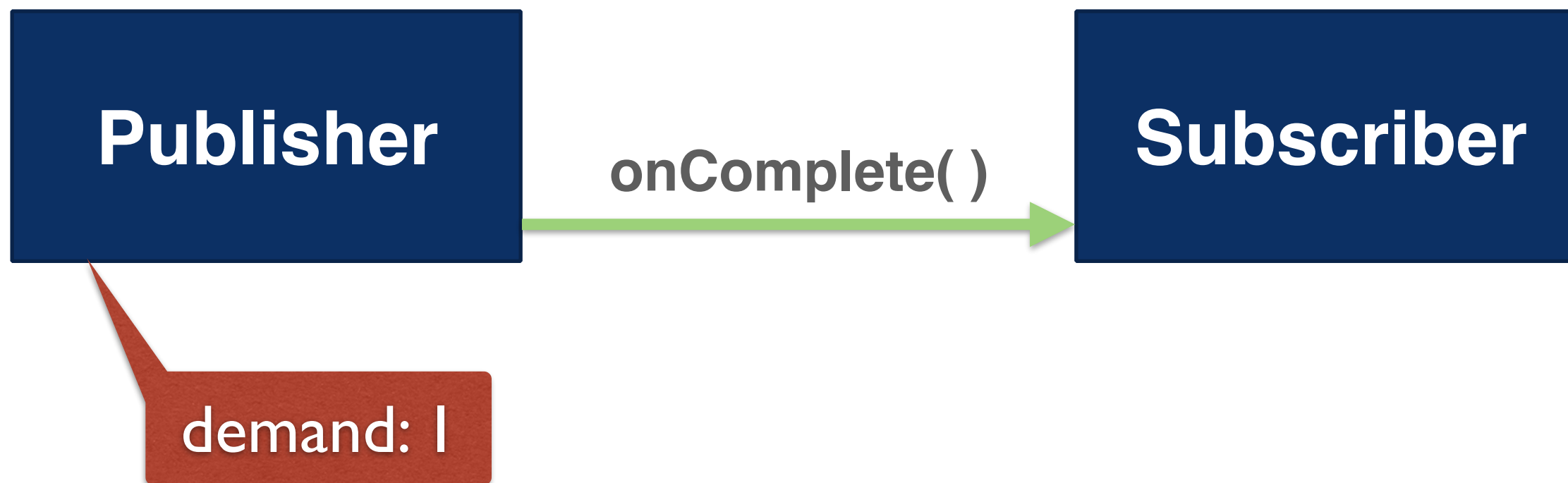
Back-pressure



Back-pressure



Back-pressure





Back-pressure

- Allows to control the amount of inflight data
- Regulate the transfer between
 - Slow publisher and fast consumer
 - Fast publisher and slow consumer



Reactive Streams is 4 interfaces (and a TCK)

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

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

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

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



Available in 2 distinct packages



As a standalone JAR with `org.reactivestreams` package:

- Publisher
- Subscriber
- Subscription
- Processor

Same interfaces are included in the upcoming Java 9 in `java.util.concurrent`:

- `Flow.Publisher`
- `Flow.Subscriber`
- `Flow.Subscription`
- `Flow.Processor`



Reactive APIs on the JVM

Apply a wide range of transformations to your data with operators: merge, buffer, split, transform, delay ...



RxJava



Akka Streams



Reactor Core



Reactive APIs on the JVM (1/3)

Reactive API	Types for 0..n elements	Types for 0..1 elements
RxJava 1	Observable	Single (1) Completable (0)
Akka Stream 2	Source Sink Flow	
Reactor Core 3	Flux	Mono (0..1)
RxJava 2	Flowable Observable	Single (1) Maybe (0..1) Completable (0)



Reactive APIs on the JVM (2/3)

Reactive API	Reactive Streams types	Non Reactive Streams types
RxJava 1		Observable Single Completable
Akka Stream 2	Source Sink Flow	
Reactor Core 3	Flux Mono	
RxJava 2	Flowable	Observable Single Maybe Completable

Limited back-pressure support



Reactive APIs on the JVM (3/3)

Reactive API	Generation	Support
RxJava 1	2nd	Limited back-pressure
Akka Stream 2	3rd	Reactive Streams + actor fusion
Reactor Core 3	4th	Reactive Streams + operator fusion
RxJava 2	4th	Reactive Streams + operator fusion

Reactive APIs on the JVM (3/3)

Reactive API	Generation	Support
RxJava 1	2nd	Limited
Reactor Core 3	4th	Reactive Streams + operator fusion
RxJava 2	4th	Reactive Streams + operator fusion

Check out
<http://akarnokd.blogspot.fr/2016/03/operator-fusion-part-1.html>

RxJava 2 or Reactor Core 3?



David Karnok

@akarnokd



Abonné

Use Reactor 3 if you are allowed to use Java 8+,
use RxJava 2 if you are stuck on Java 6+ or need
your functions to throw checked exceptions

Voir la traduction

RETWEETS
33

J'AIME
48



14:49 - 10 sept. 2016





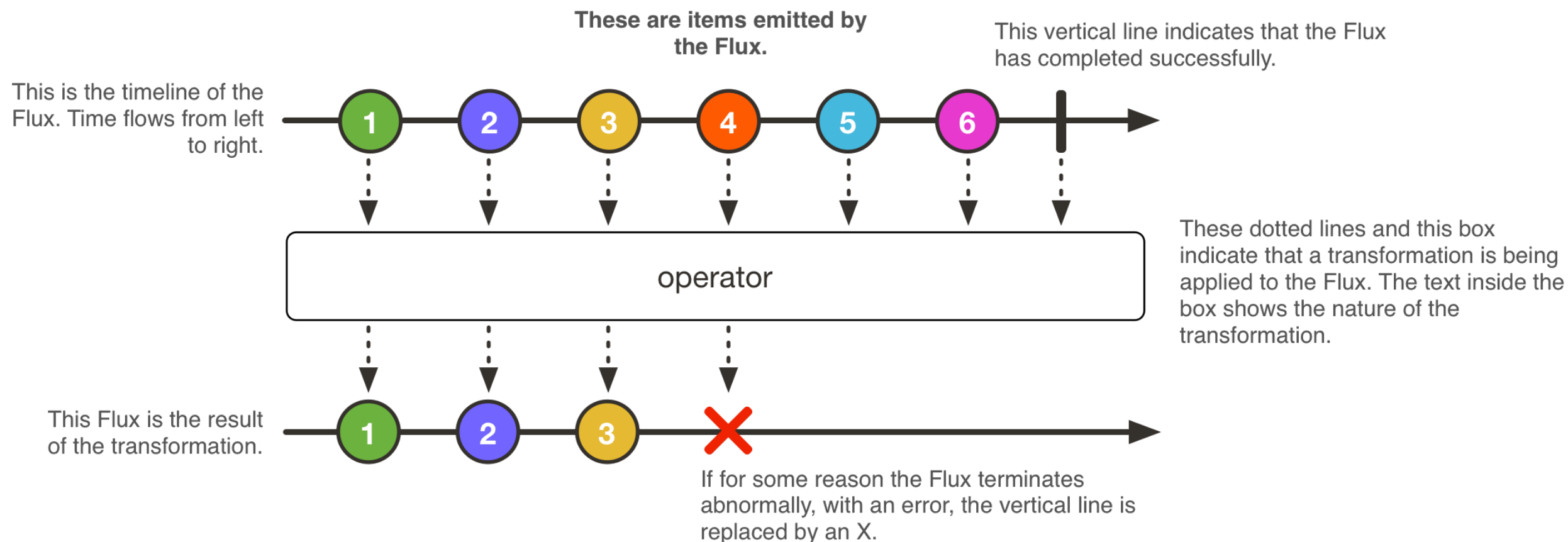
Focus on Reactor Core 3

- Natively designed on top of Reactive Streams
- Lightweight API with 2 types: **Mono** and **Flux**
- Native Java 8 support and optimisations
- Single 1 Mbytes JAR
- Focus on performance
- Reactive foundation of Spring Framework 5



Flux<T>

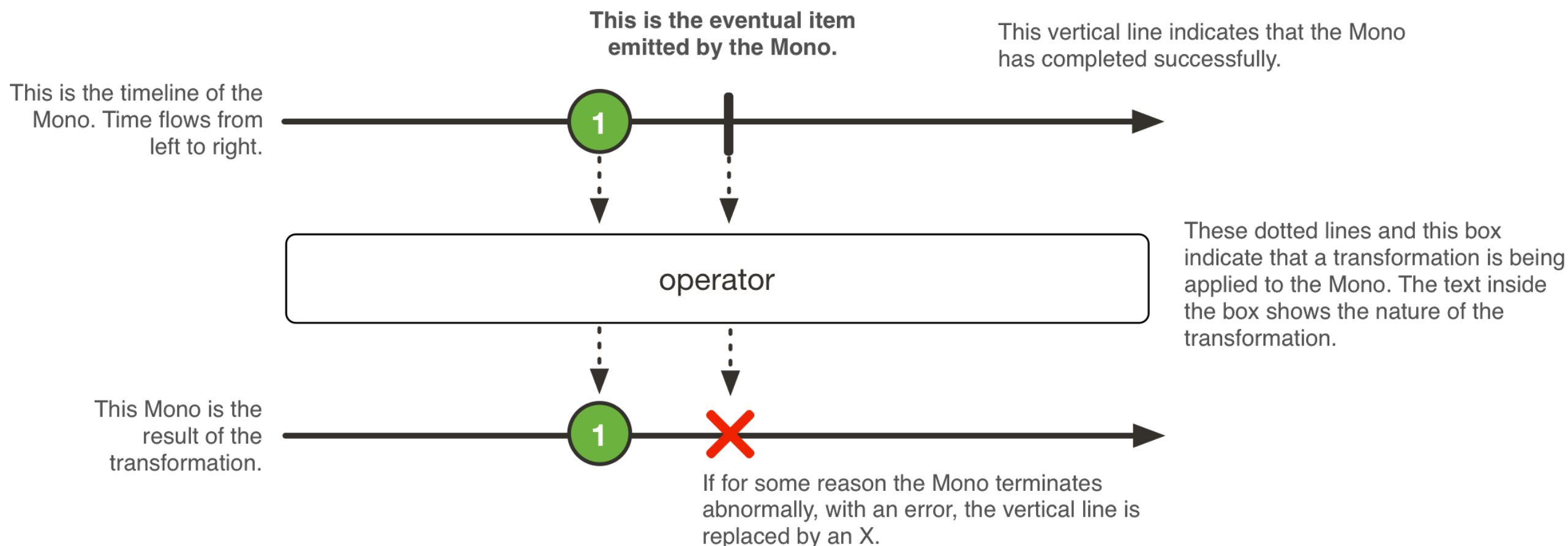
- Implements Reactive Streams **Publisher**
- 0 to n elements
- Operators: `flux.map(...).zip(...).flatMap(...)`





Mono<T>

- Implements Reactive Streams **Publisher**
- 0 to 1 element
- Operators: `mono.then(...).otherwise(...)`



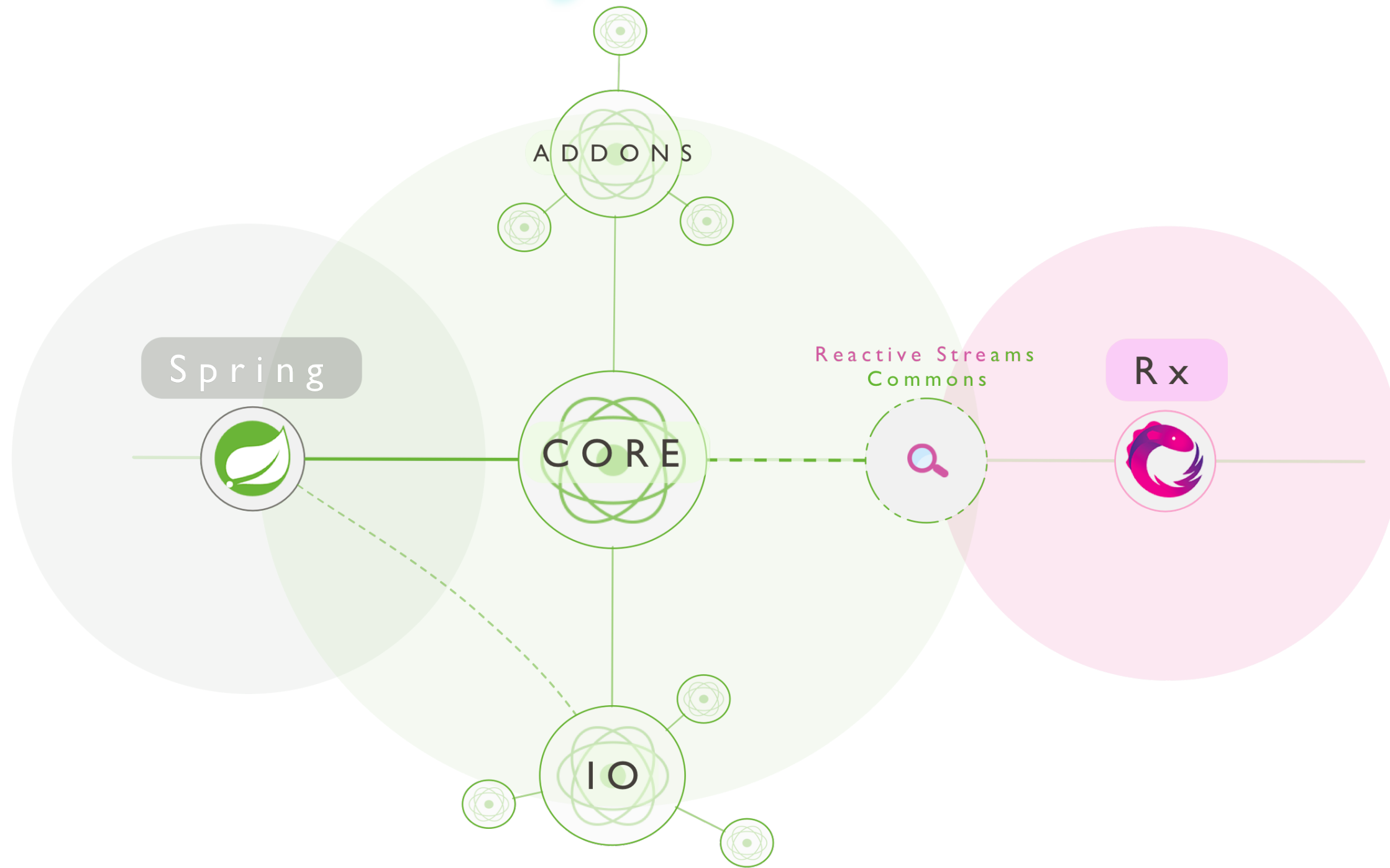


StepVerifier

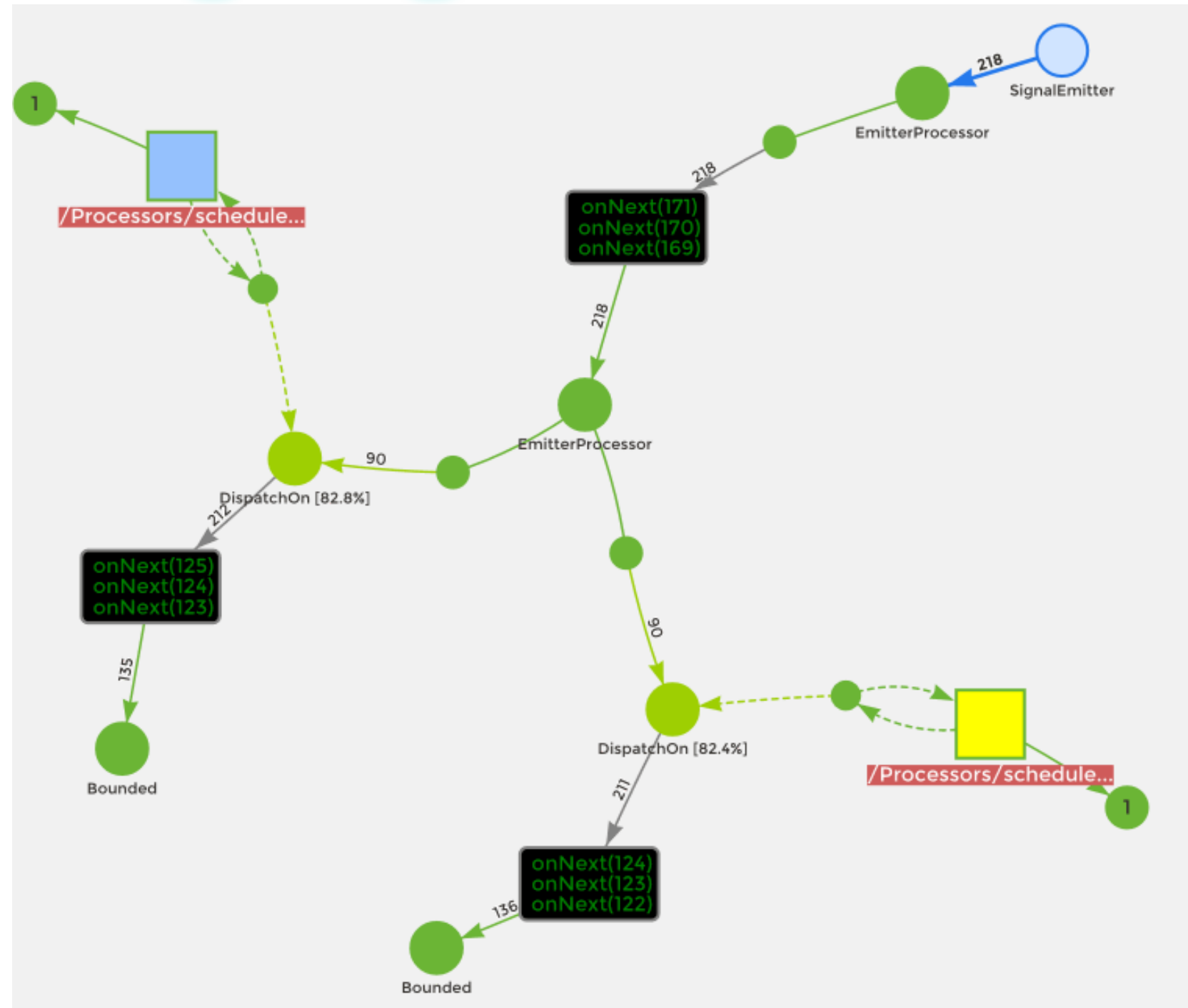
- Designed to **test** easily Reactive Streams **Publishers**
- Carefully designed after writing thousands of Reactor and Spring reactive tests

```
StepVerifier.create(flux)
    .expectNext("foo", "bar")
    .expectComplete()
    .verify();
```

Reactor 3 ecosystem



Upcoming high-level features like ...





Live coding session

- Creating Mono and Flux
- StepVerifier
- Transform: `map()` + `flatMap()`
- Merge
- Request and Back pressure handling
- Error handling
- Convert & adapt: RxJava 2, List
- Reactive to blocking and other way around

**Flux.just(🍊, ☕);
see you in 20 minutes!**

DEV
VOXX™

Spring Reactive Web



+



spring



Building a Reactive application

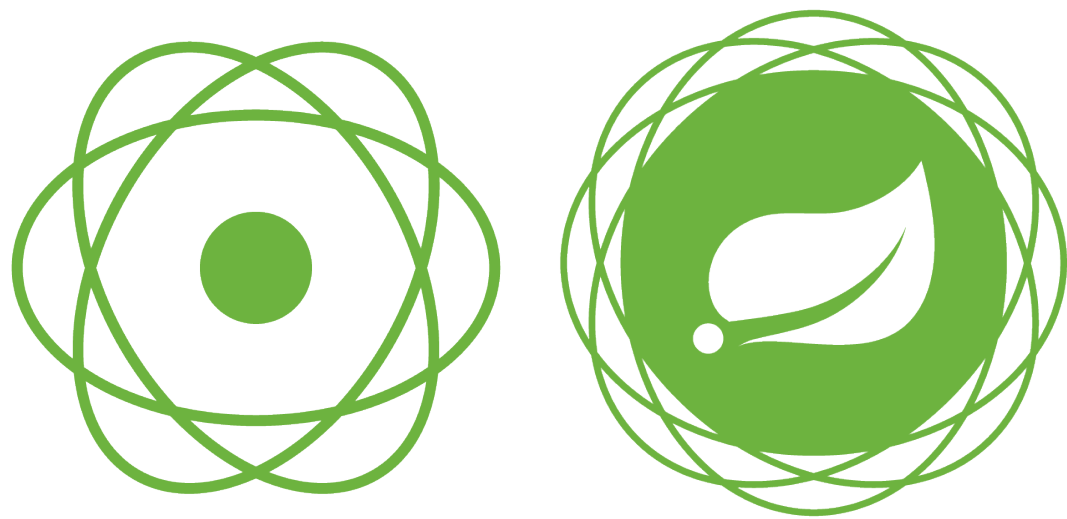
Building a web application with:

- Spring Boot Reactive starter (Experimental ⚠️)
- Spring Boot 2.0 (SNAPSHOT 🛠️)
- Spring Framework 5.0 (MILESTONE 🆕)
- Reactor 3.0 (RELEASE 🤘)

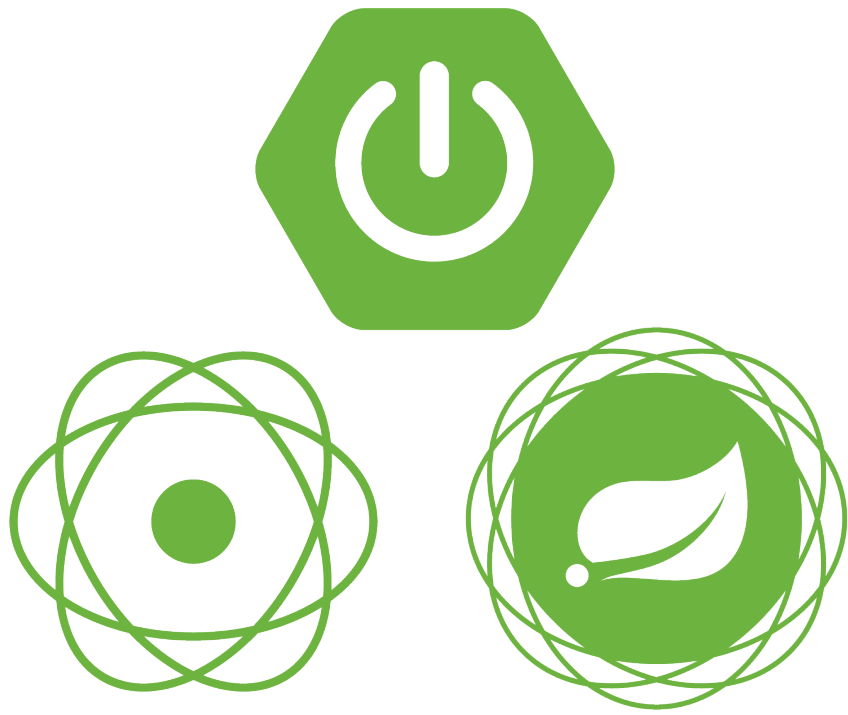
DEVOXX™



DEVOXX™



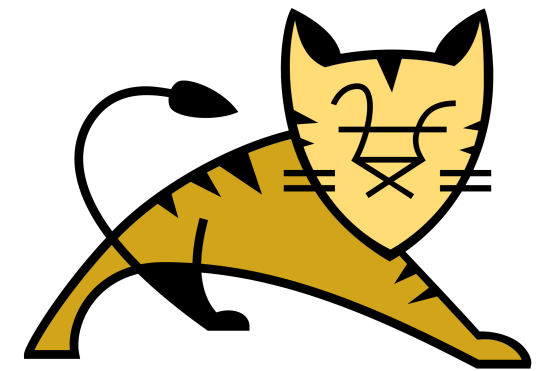
DEVVOXX™



Framework



jetty://



Multiple runtimes



Live coding session

- Calling remote REST APIs
- Dealing with datastores
- Error handling
- Rendering HTML views
- Serving static resources
- Server Sent Events
- And more!



Our experience with Reactor 3



Changing your mindset

Non-blocking when Thread/IO boundary
(Servlet async IO, Java NIO Channels, ...)



Changing your mindset

Revisit/reconsider some concepts
(ThreadLocal, Transactions, shared state, ...)



Changing your mindset

Common language with your peers
(async, non-blocking, back-pressure, reactivex, reactive
streams, operators...)



Reactive Streams

1. Don't implement Reactive Streams interfaces yourself (or do it "for fun")
2. Reactive Streams by itself is not enough
3. Nothing happens until something subscribes to the Publisher



Concurrency

1. Debugging concurrency issues is hard
2. concurrency Debugging is issues hard



Reactive ecosystem

1. Carefully select your libraries / API / drivers
2. Tooling is important
3. Define additional code style rules in your team



Links

- @ProjectReactor & <https://projectreactor.io>
- <https://github.com/reactor/lite-rx-api-hands-on>
- <https://github.com/bclozel/spring-reactive-university>
- <https://spring.io>
- <https://spring.io/blog/2016/04/19/understanding-reactive-types>

Reactive Web Applications with Spring 5

Rossen Stoyanchev

Friday 9:30, Room 8





The Spring BOF

Spring team

Wednesday 7:00 PM, BOF2





Questions?