CALLISTA
— ENTERPRISE —

REACTIVE PROGRAMMING

WITH RXJAVA

CADEC 2017 - Pär Wenåker, Anders Asplund

AGENDA

- What and why
- RxJava
- Future
- Demo

WHAT IS REACTIVE PROGRAMMING?

Creator of Reactive Extensions @ Microsoft

Endless confusion

ATTEMPT OF DEFINITION

REACTIVE PROGRAMMING MODEL

- Asynchronous and non-blocking applications.
- Functional in style.
- Readable and composable APIs.
- Handles asynchronous streams of data.

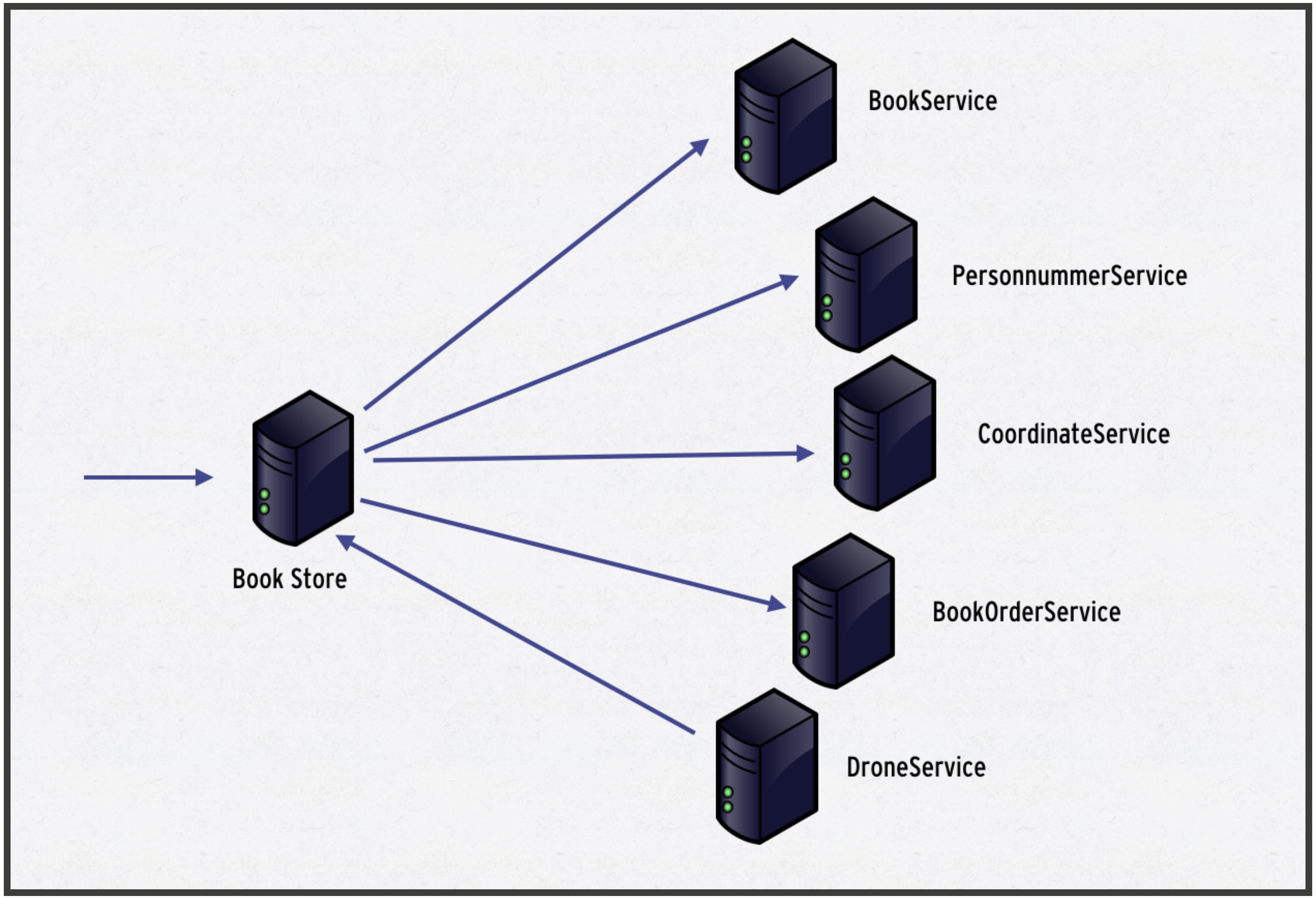
WHY?

- Distributed applications (microservices)
- Cloud environments
- Streaming Big Data
- Internet of Things

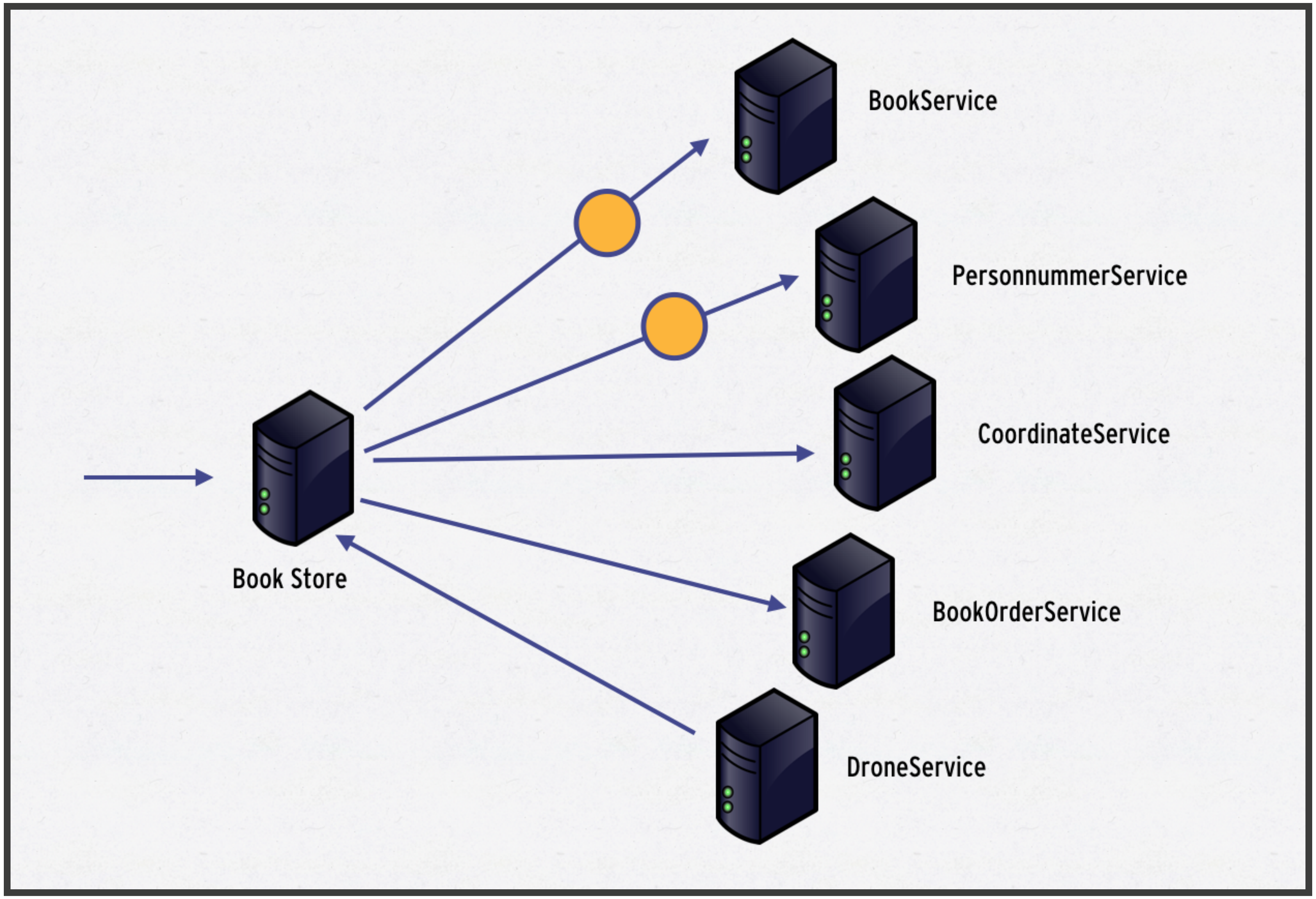
Better resource utilisation

✓

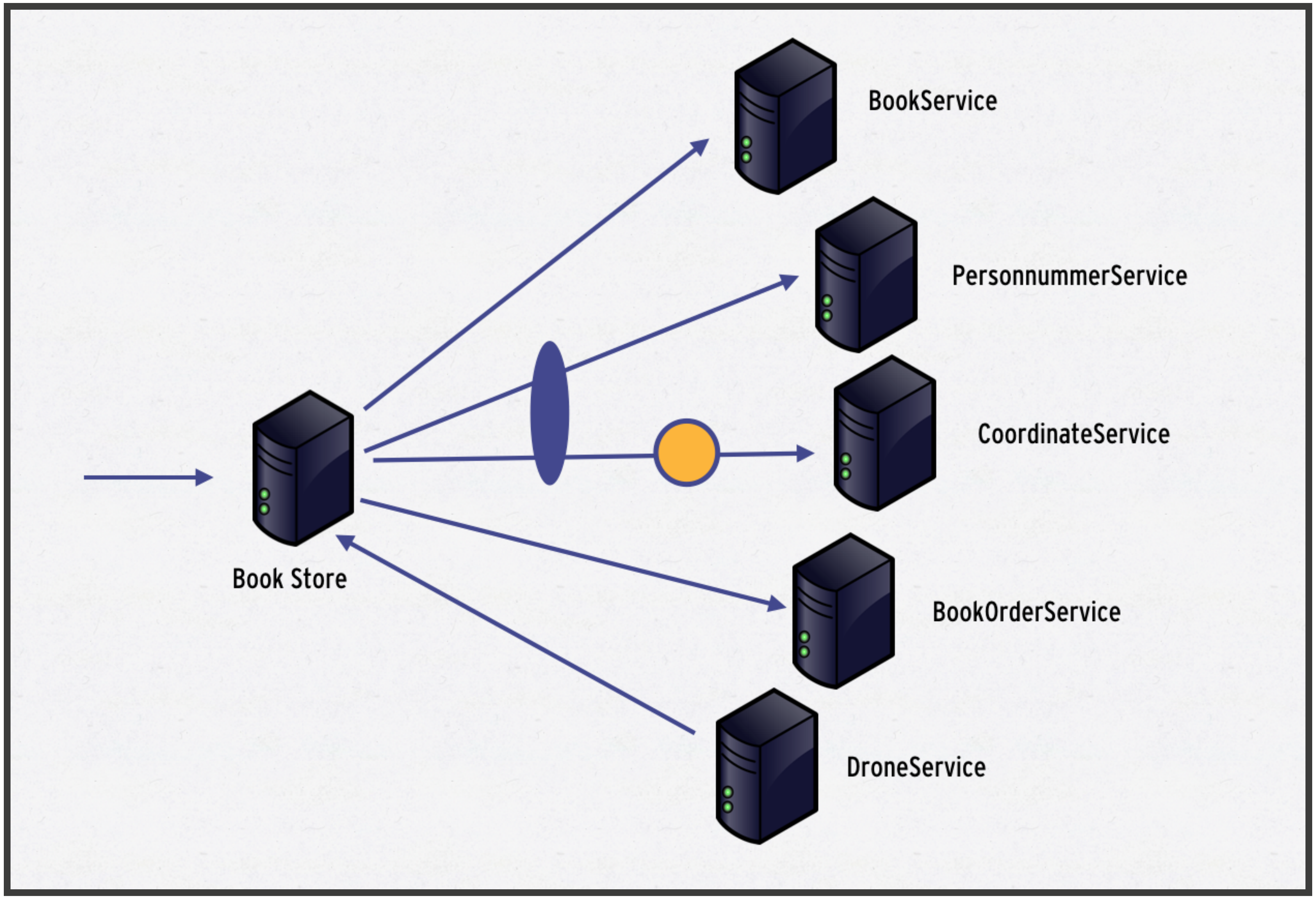
EXAMPLE SERVICE



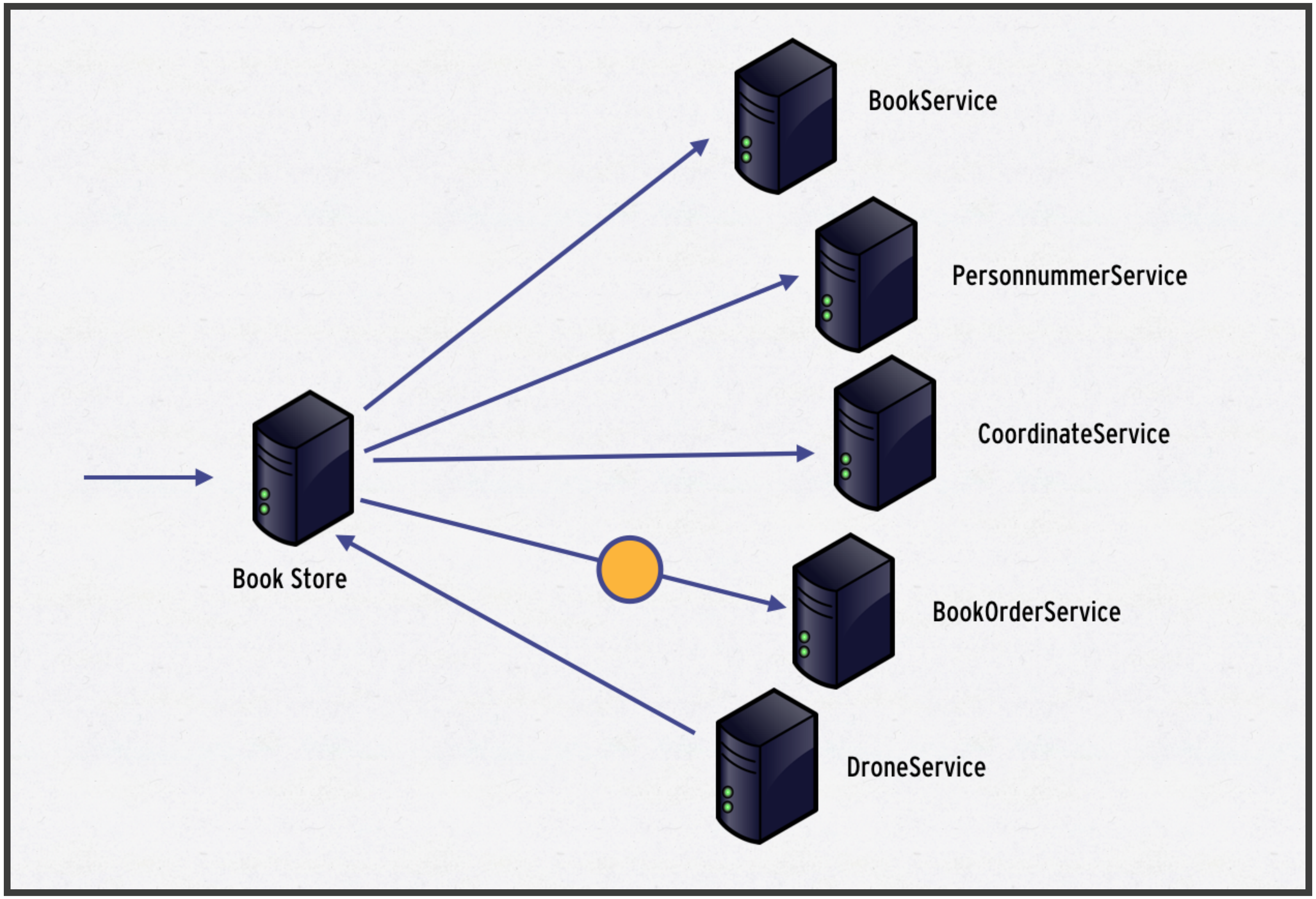
EXAMPLE SERVICE



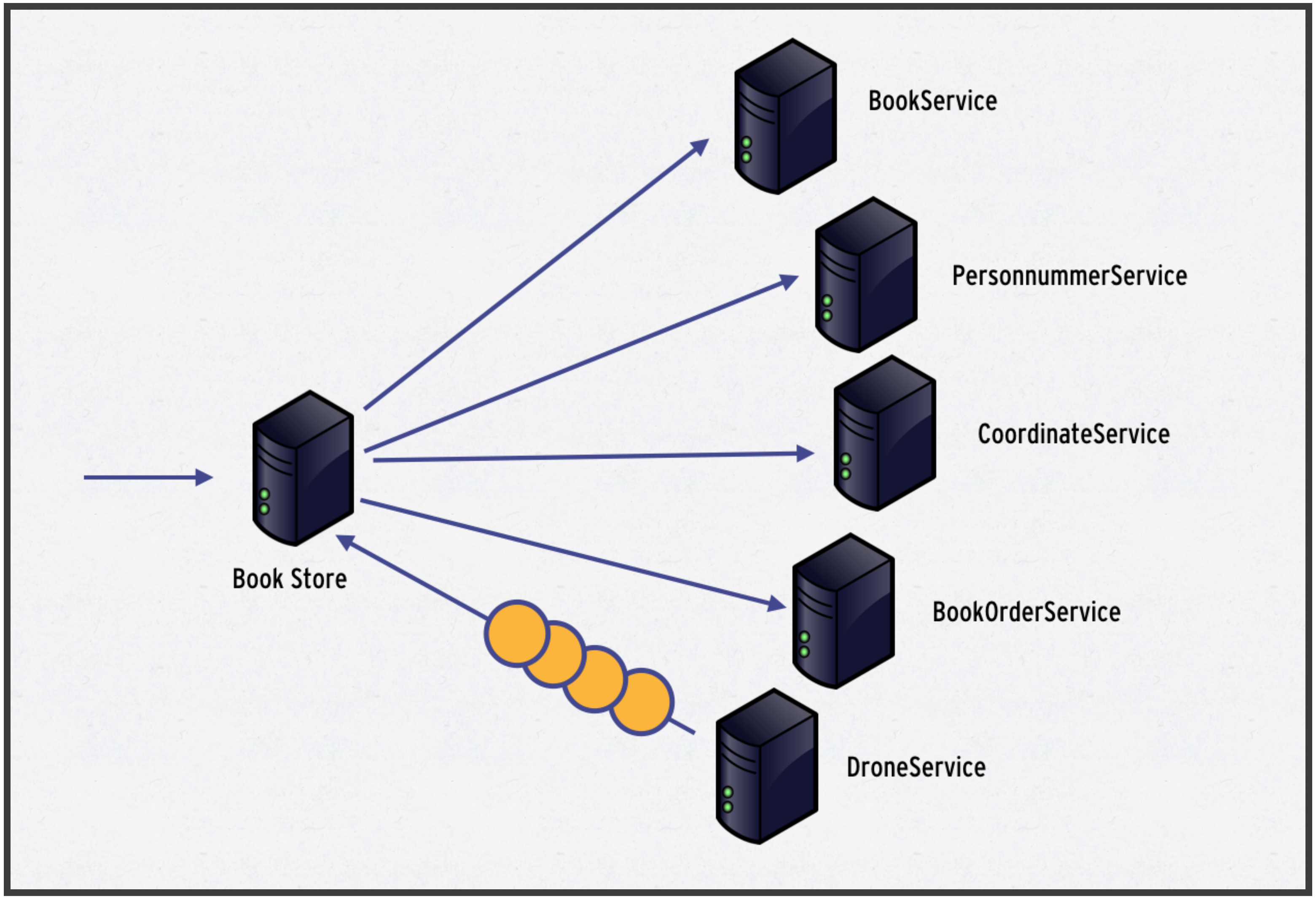
EXAMPLE SERVICE



EXAMPLE SERVICE



EXAMPLE SERVICE



ASYNCHRONOUS CODE PRE JAVA 1.8

```
Future<Response> f = httpClient.get("http://www.gp.se/");
Response r = f.get(); // Blocking!!!
```

Not Asychronous!!

ASYNCHRONOUS CALL WITH CALLBACK

```
httpClient.get("http://www.gp.se", response -> {
   /* Handle response */
});
```

MULTIPLE ASYNCHRONOUS CALLS WITH CALLBACK

```
httpClient.get("http://www.gp.se", response -> {
   /* Handle response */
});

httpClient.get("http://www.dn.se", response -> {
   /* Handle response */
});
```

Needs state machine...

ASYNCHRONOUS CALL WITH CALLBACK

```
State state = ...;

httpClient.get("http://www.gp.se", response -> {
   state.addGp(response);
});

httpClient.get("http://www.dn.se", response -> {
   state.addDn(response);
});
```

Needs synchronization!!

ASYNCHRONOUS CALL WITH CALLBACK

```
State state = ...;
httpClient.get("http://www.gp.se", response -> {
   synchronized(state) {
      state.addGp(response);
   }
});
httpClient.get("http://www.dn.se", response -> {
   synchronized(state) {
      state.addDn(response);
   }
});
```

LET'S ADD SOME ERROR HANDLING....

THE PROBLEM IS...

- "Void returning functions"
- Mutable state

"Code that deals with more than one event or asynchronous computation gets complicated quickly as it needs to build a state-machine to deal with ordering issues.

Next to this, the code needs to deal with successful and failure termination of each separate computation. This leads to code that doesn't follow normal control-flow, is hard to understand and hard to maintain."

http://go.microsoft.com/fwlink/?LinkID=205219

Reactive Programming to rescue!!

DISCLAIMER 1

Is not easy and has a significant learning curve!

DISCLAIMER 2

Requires a shift from imperative programing to asynchronous, non-blocking and functional style.

Do not rely on side-effects and mutable state!

```
Observable<Tweets> twitterFirehose = ...
twitterFirehose
  .sample(5, MILLISECONDS)
  .filter(t -> hasImage(t))
  .flatMap(t -> getImage(t.image))
  .publish()

tweets.subscribe(/* Do something */);
tweets.subscribe(/* Do something else */);
```

JAVA 1.8 COMPLETABLEFUTURES

```
CompletableFuture<Response> response1 =
   httpClient.get("http://gp.se");

CompletableFuture<Response> response2 =
   httpClient.get("http://dn.se");

CompletableFuture<State> state =
   response1.thenCombine(response2,
      (value1, value2) -> f(value1, value2)
   );
state.thenAccept(s -> /* Handle result */)
     .exceptionally(/*Handle error*/ return errorStatus; });
```

RXJAVA

```
Observable<Response> resp1 = httpClient.get("http://gp.se");
Observable<Response> resp2 = httpClient.get("http://dn.se");

Observable<Status> status =
   resp1.zipWith(resp2, (r1,r2) -> f(r1,r2))
        .onErrorReturn(t -> {/*Handle error*/ return errorStatus; });

status.subscribe(/* Handle result */)
```

CONTAINER FOR THE FUTURE VALUE.

- Observable
- CompletableFuture

METHOD ON THE CONTAINER FOR MANIPULATING THE FUTURE VALUE.

- zipWith
- thenApply
- thenAccept
- onError
- etc.

STREAMS

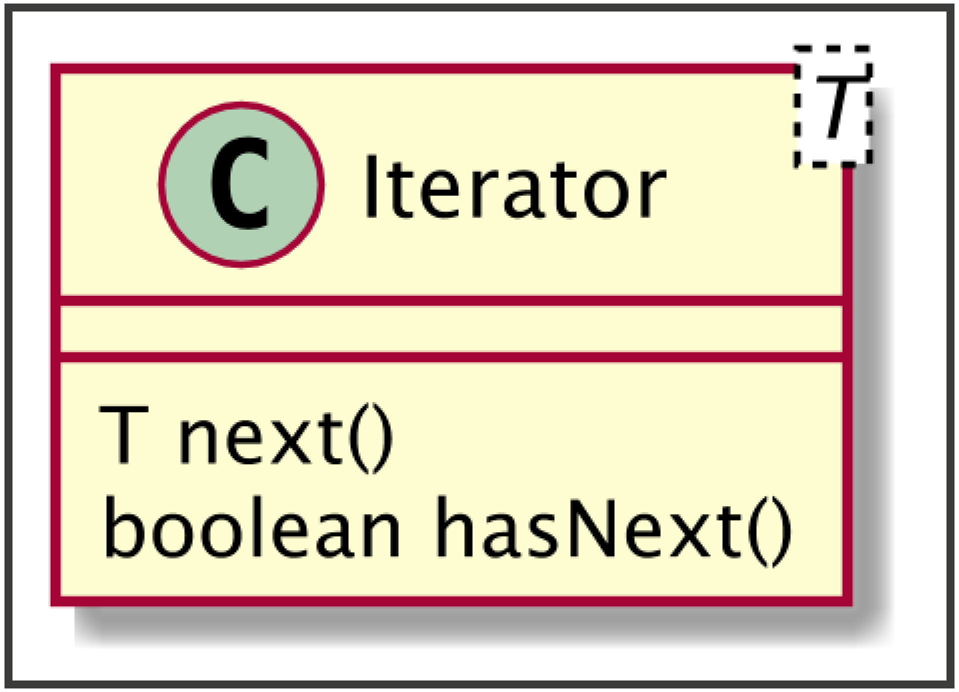
CompletableFuture handles only one element!

Observables!

AN OBSERVABLE IS...

a combination of the iterator and observer patterns!

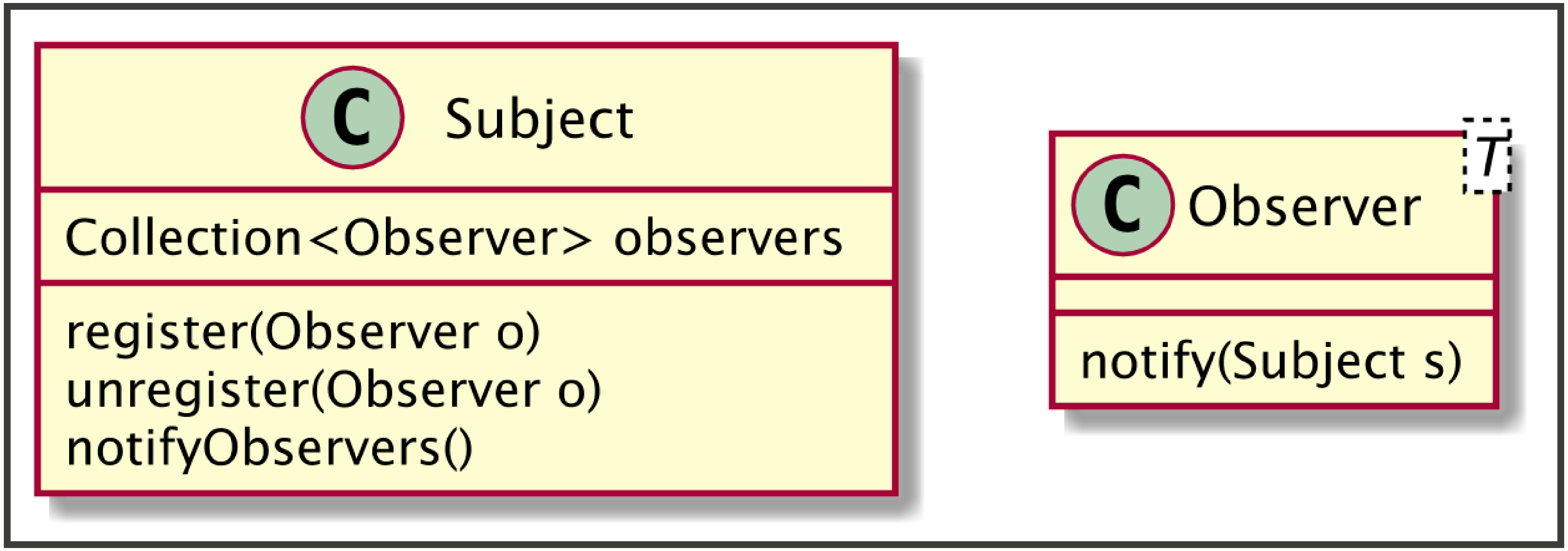
ITERATOR PATTERN



- Retreive data => next()
- Complete => hasNext()
- Error => throws Exception

PULL BASED

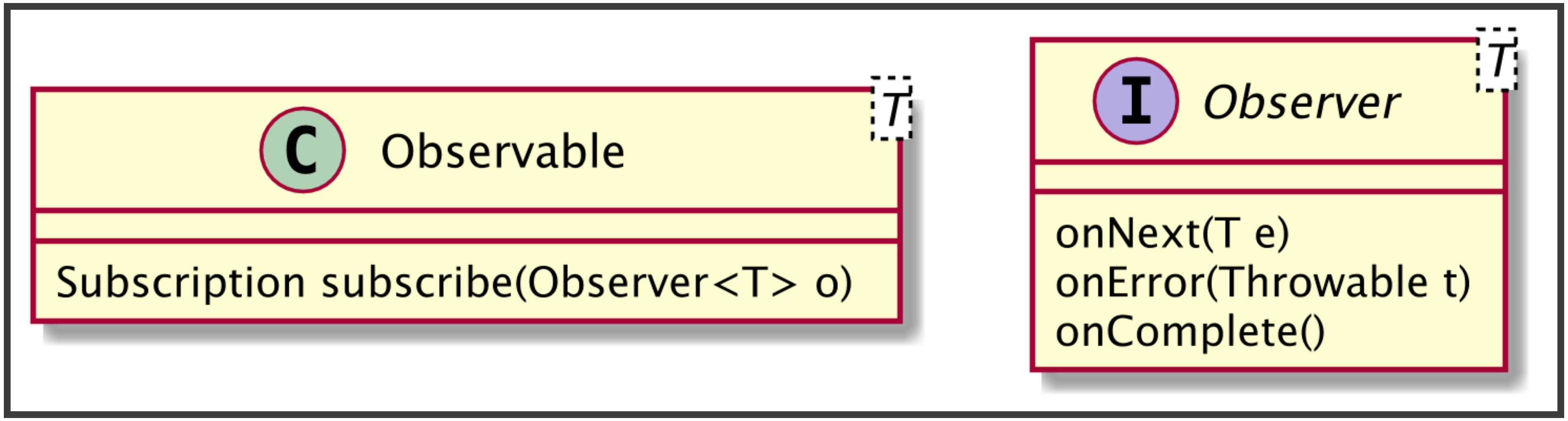
OBSERVER PATTERN



- Register Observer with Subject
- Subject pushes data to Observer
- Complete & Error???

PUSH BASED

OBSERVABLE PATTERN



- Register Observer with Observable (subject)
- Observable pushes data to the Observer with onNext()
- Observable signals end of data with onComplete()
- Observable signals errors with onError()

PUSH BASED

EXAMPLE 1

```
Observable<Integer> obs = Observable.create(observer -> {
        observer.onNext(1);
        observer.onNext(2);
        observer.onComplete();
});
println("Before subscribe")
obs.subscribe(v -> println(v));
println("After subscribe")


Thread[main,5,main] : Before subscribe
Thread[main,5,main] : 1
Thread[main,5,main] : 2
Thread[main,5,main] : After subscribe
```

EXAMPLE 2

```
Observable<Integer> obs = Observable.create(observer -> {
     new Thread(() -> {
         observer.onNext(1);
         observer.onNext(2);
         observer.onComplete();
     }).start();
 });
println("Before subscribe")
obs.subscribe(v -> println(v));
println("After subscribe")
```

```
Thread[main,5,main] : Before subscribe
Thread[main,5,main] : After subscribe
Thread[Thread-0,5,main] : 1
Thread[Thread-0,5,main] : 2
```

Don't do this...

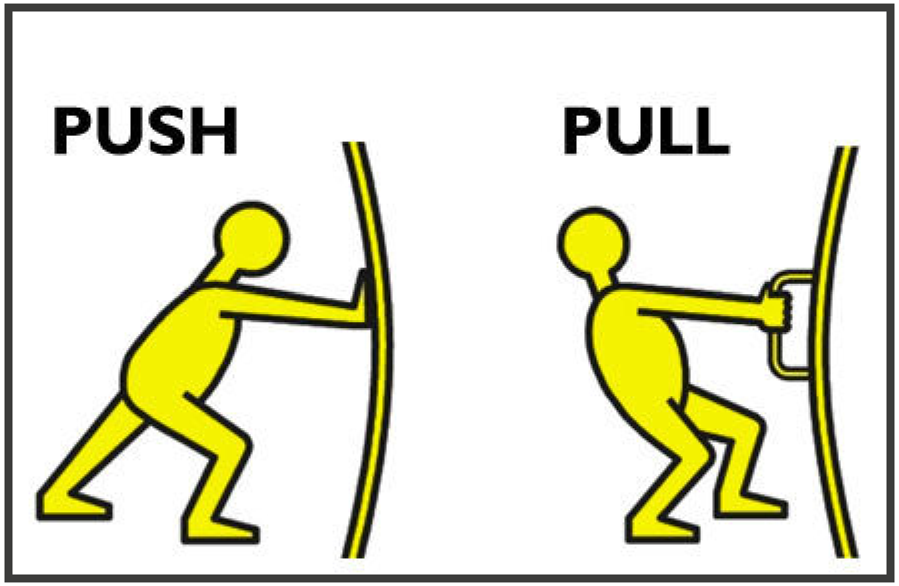
EXAMPLE 3

```
Observable<Response> obs = Observable.create(observer -> {
     httpClient.get("http://www.gp.se", response -> {
         observer.onNext(response);
         observer.onComplete();
     });
});
obs.subscribe(response -> /* Handle output */);
```

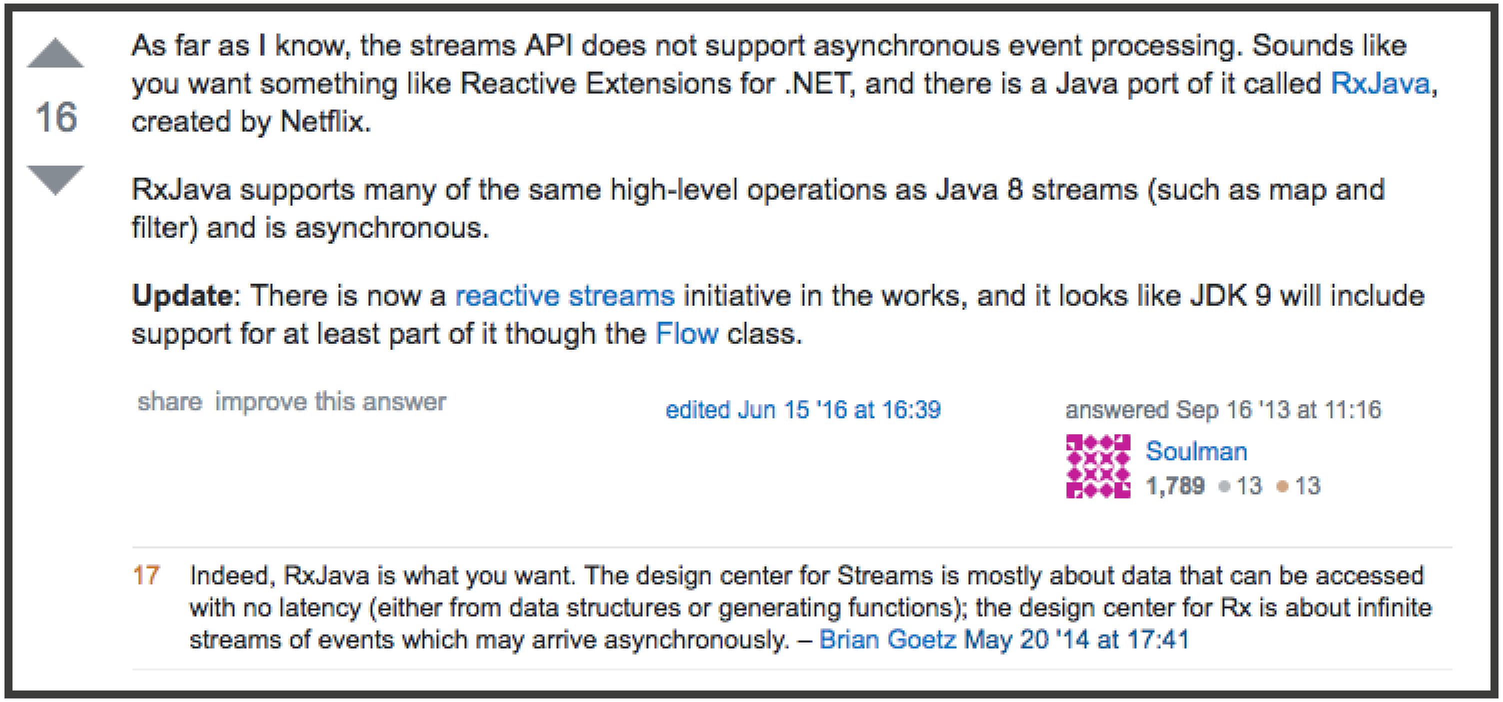
SINGLE

```
Single<Response> obs = Single.create(observer -> {
     httpClient.get("http://www.gp.se", response -> {
         observer.onSuccess(response);
     });
});
obs.subscribe(response -> /* Handle output */);
```

	Single	**Multiple**
	Single	**Multiple**
Synchronous	T get()	Iterator<T>
Asynchronous	Future<T>	Observable<T>


PUSH
PULL

JAVA8 STREAMS?



16

As far as I know, the streams API does not support asynchronous event processing. Sounds like you want something like Reactive Extensions for .NET, and there is a Java port of it called RxJava, created by Netflix.

RxJava supports many of the same high-level operations as Java 8 streams (such as map and filter) and is asynchronous.

Update: There is now a reactive streams initiative in the works, and it looks like JDK 9 will include support for at least part of it though the Flow class.

share improve this answer

edited Jun 15 '16 at 16:39

answered Sep 16 '13 at 11:16

Soulman

1,789 ● 13 ● 13

17 Indeed, RxJava is what you want. The design center for Streams is mostly about data that can be accessed with no latency (either from data structures or generating functions); the design center for Rx is about infinite streams of events which may arrive asynchronously. – Brian Goetz May 20 '14 at 17:41

http://bit.ly/2hU6wdA

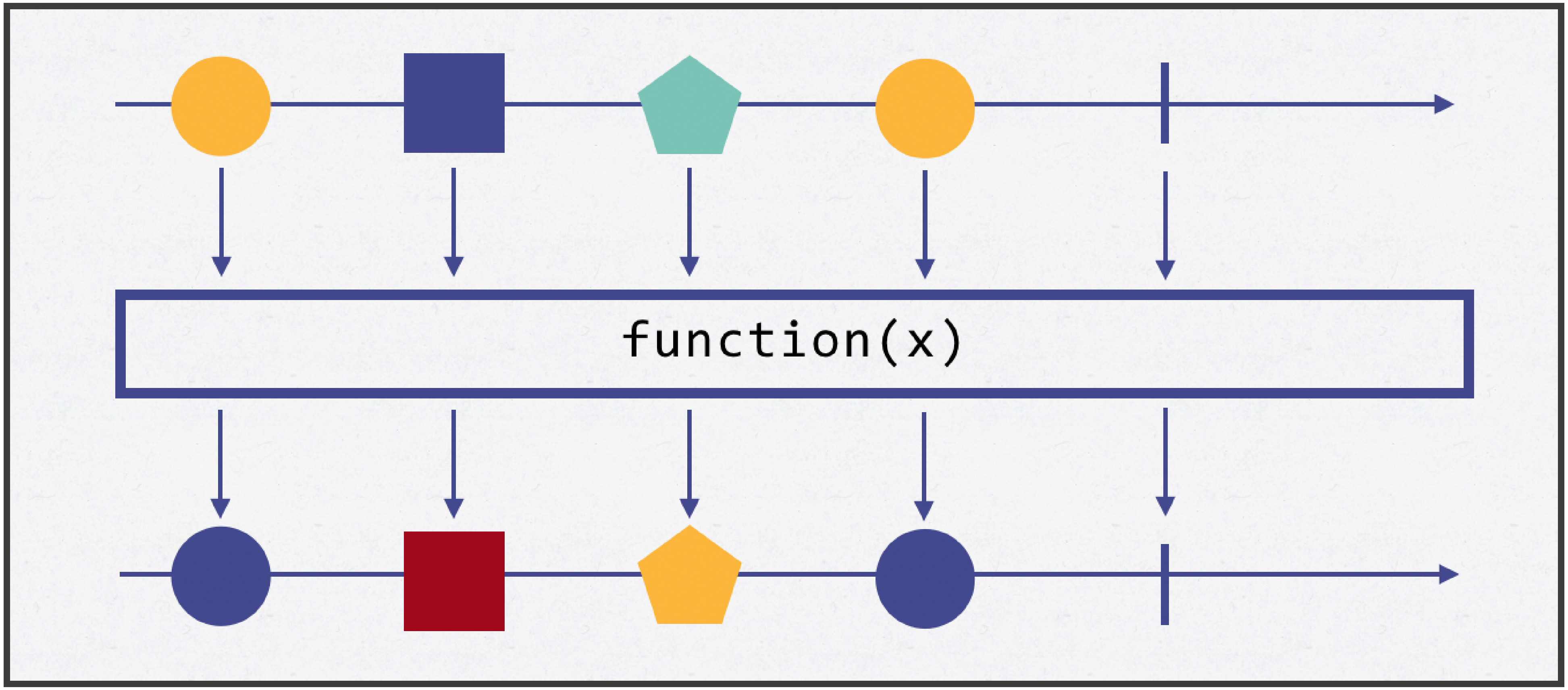
"Indeed, RxJava is what you want. The design center for Streams is mostly about data that can be accessed with no latency (either from data structures or generating functions); the design center for Rx is about infinite streams of events which may arrive asynchronously."

- Brian Goetz

https://youtu.be/fabN6HNZ2qY

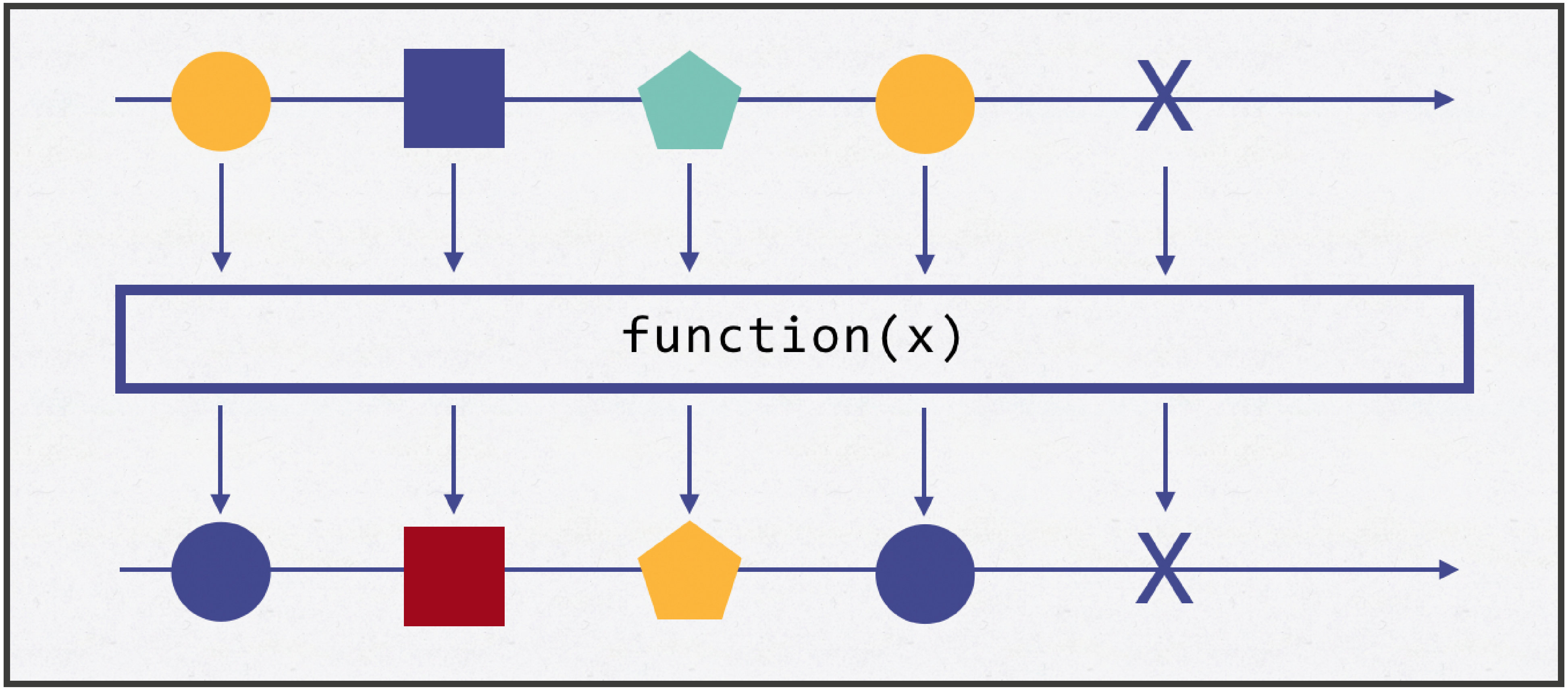
MARBLE DIAGRAM

Terminates with onComplete

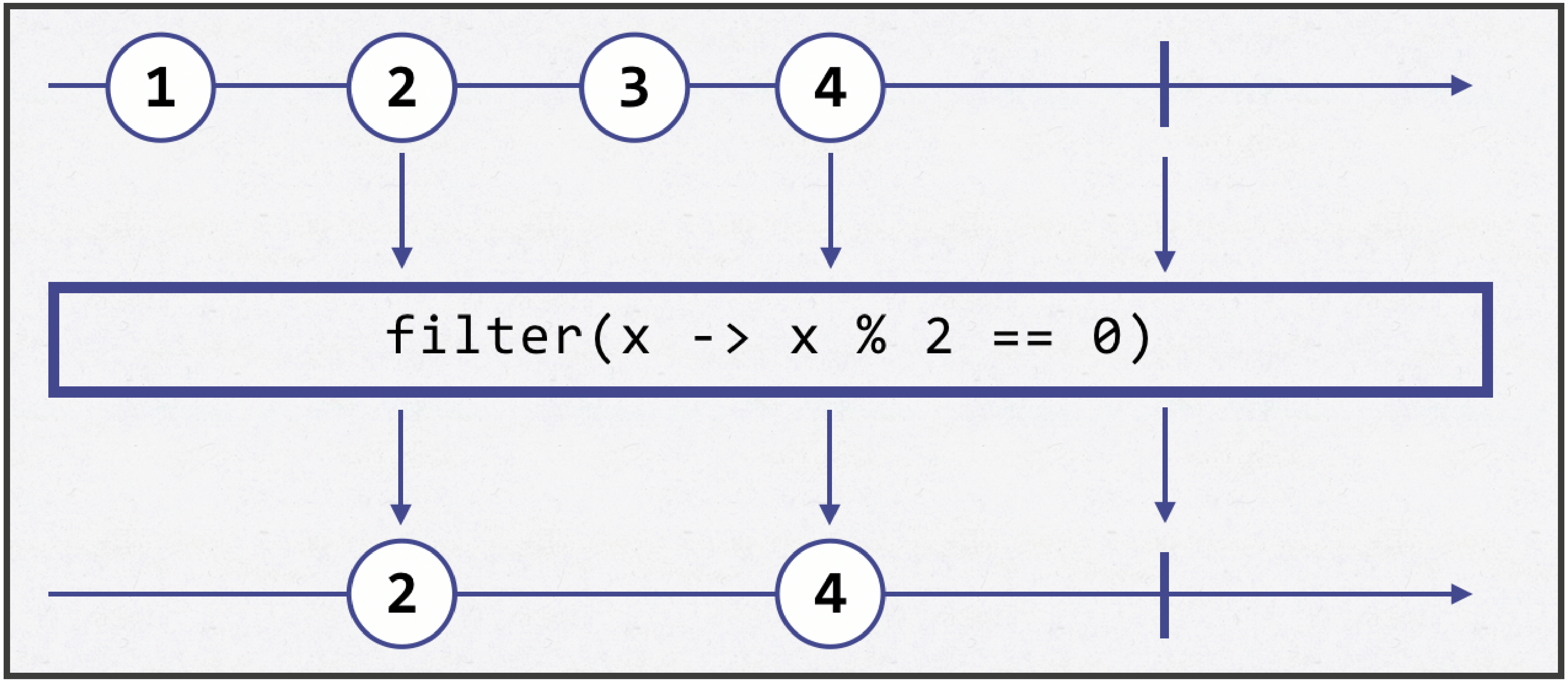


MARBLE DIAGRAM

Terminates with onError

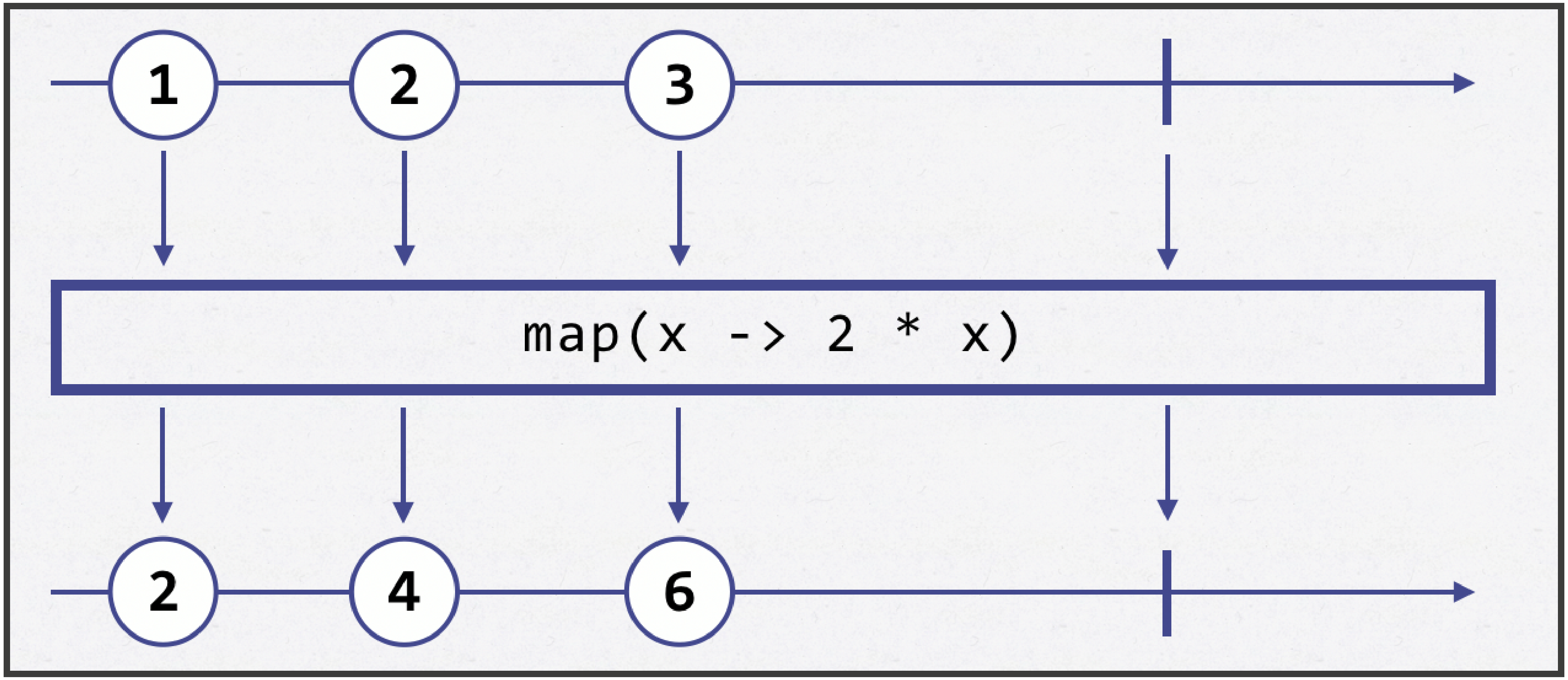


MARBLE DIAGRAM FILTER



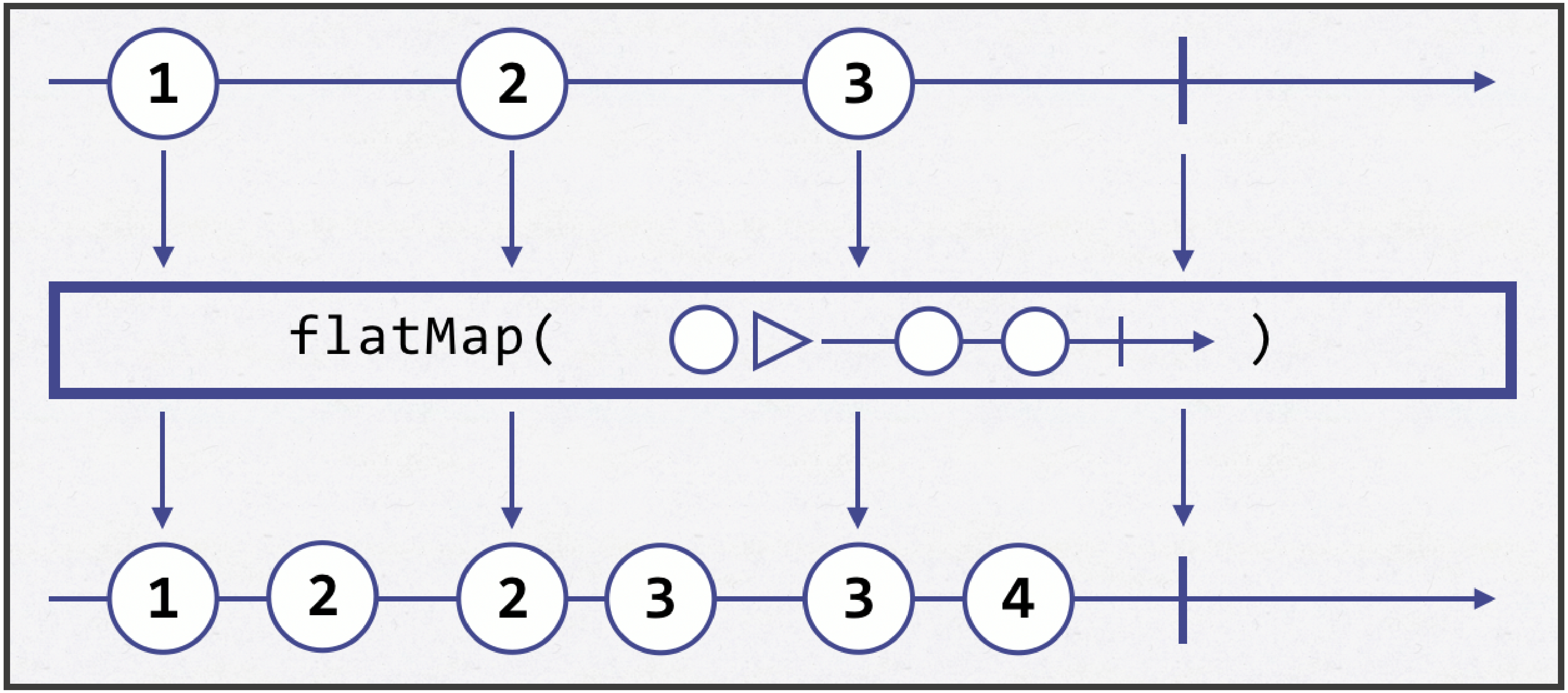
```java
Observable<Integer> obs1 = ...
Observable<Integer> obs2 = obs1.filter(x -> x%2 == 0)
```

MARBLE DIAGRAM MAP



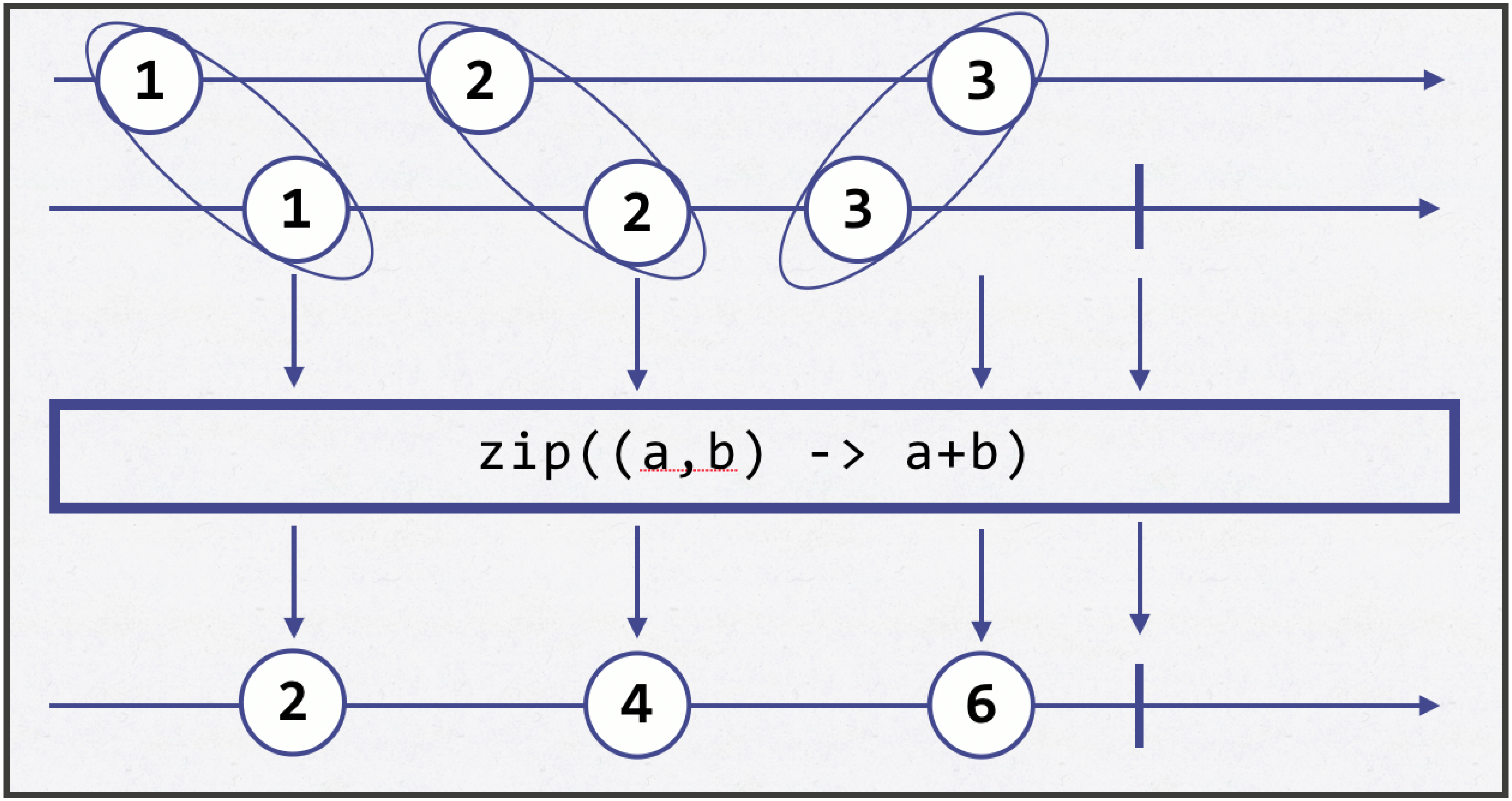
```
Observable<Integer> obs1 = ...
Observable<Integer> obs2 = obs1.map(x -> 2 * x)
```

MARBLE DIAGRAM FLATMAP



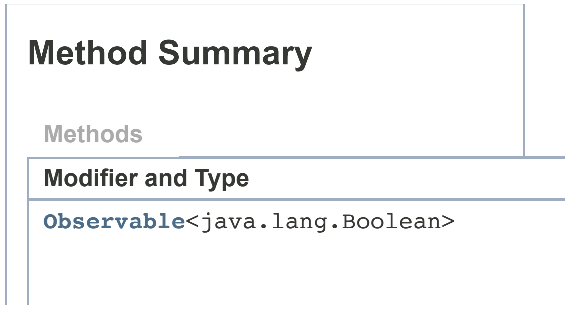
```
Observable<Integer> obs1 = ...
Observable<Integer> obs2 = obs1.flatMap(x -> Observable.just(x, x+1))
```

MARBLE DIAGRAM ZIP



```java
Observable<Integer> obs1 = ...
Observable<Integer> obs2 = ...
Observable<Integer> obs3 = obs1.zipWith(obs2, (a,b) -> a+b)
```

OBSERVABLE METHOD SUMMARY



Method Summary

Methods

Modifier and Type
`Observable<java.lang.Boolean>`

BACKPRESSURE

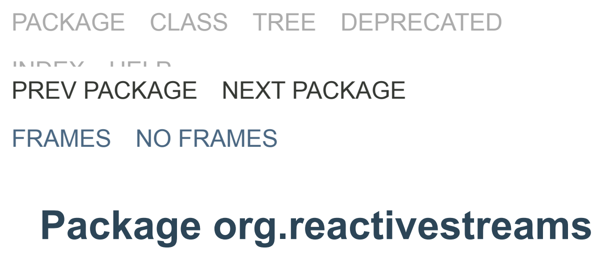
REACTIVE STREAMS

KAAZING Lightbend Pivotal

redhat twitter NETFLIX

http://www.reactive-streams.org

REACTIVE STREAMS



PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV PACKAGE NEXT PACKAGE

FRAMES NO FRAMES

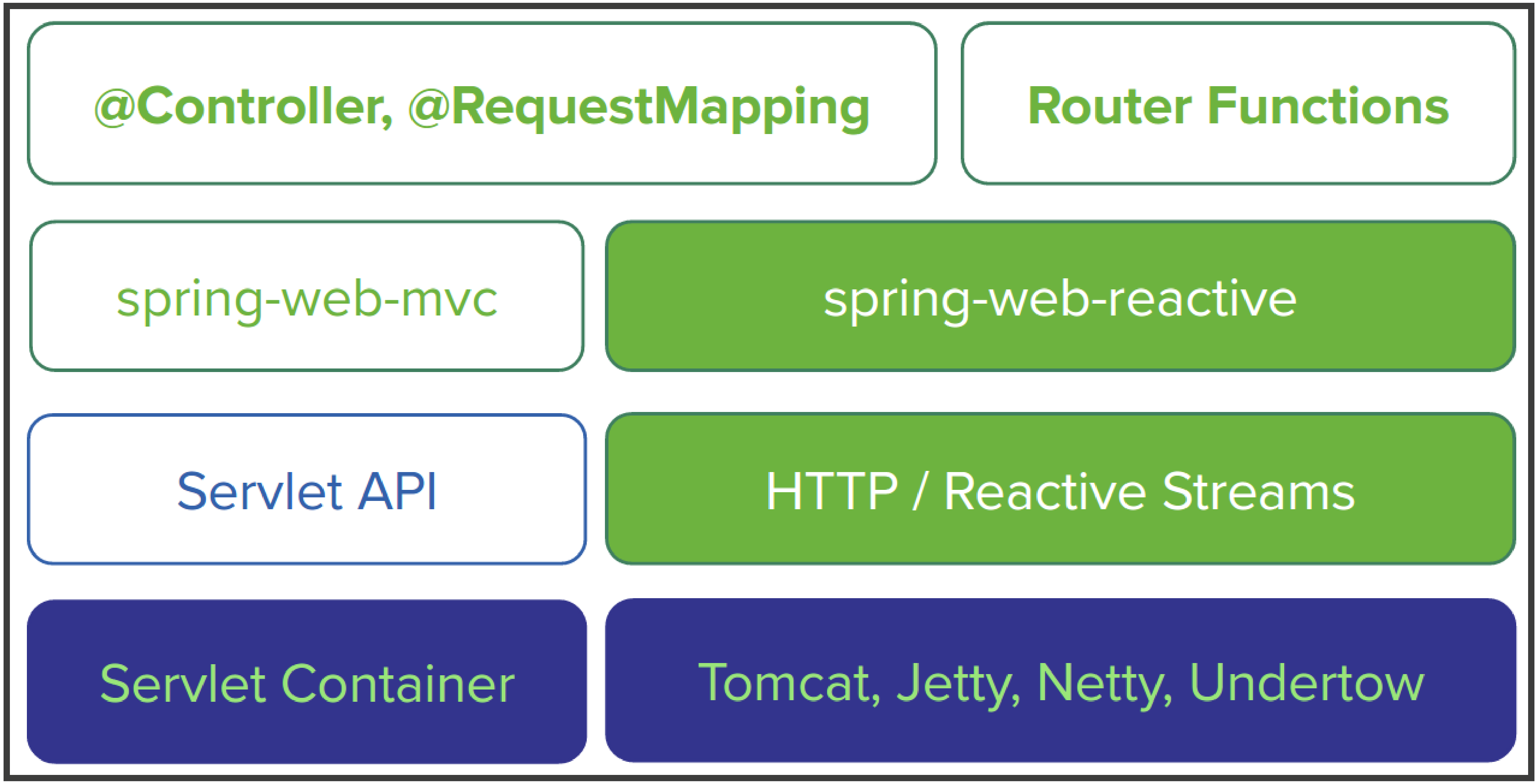
Package org.reactivestreams

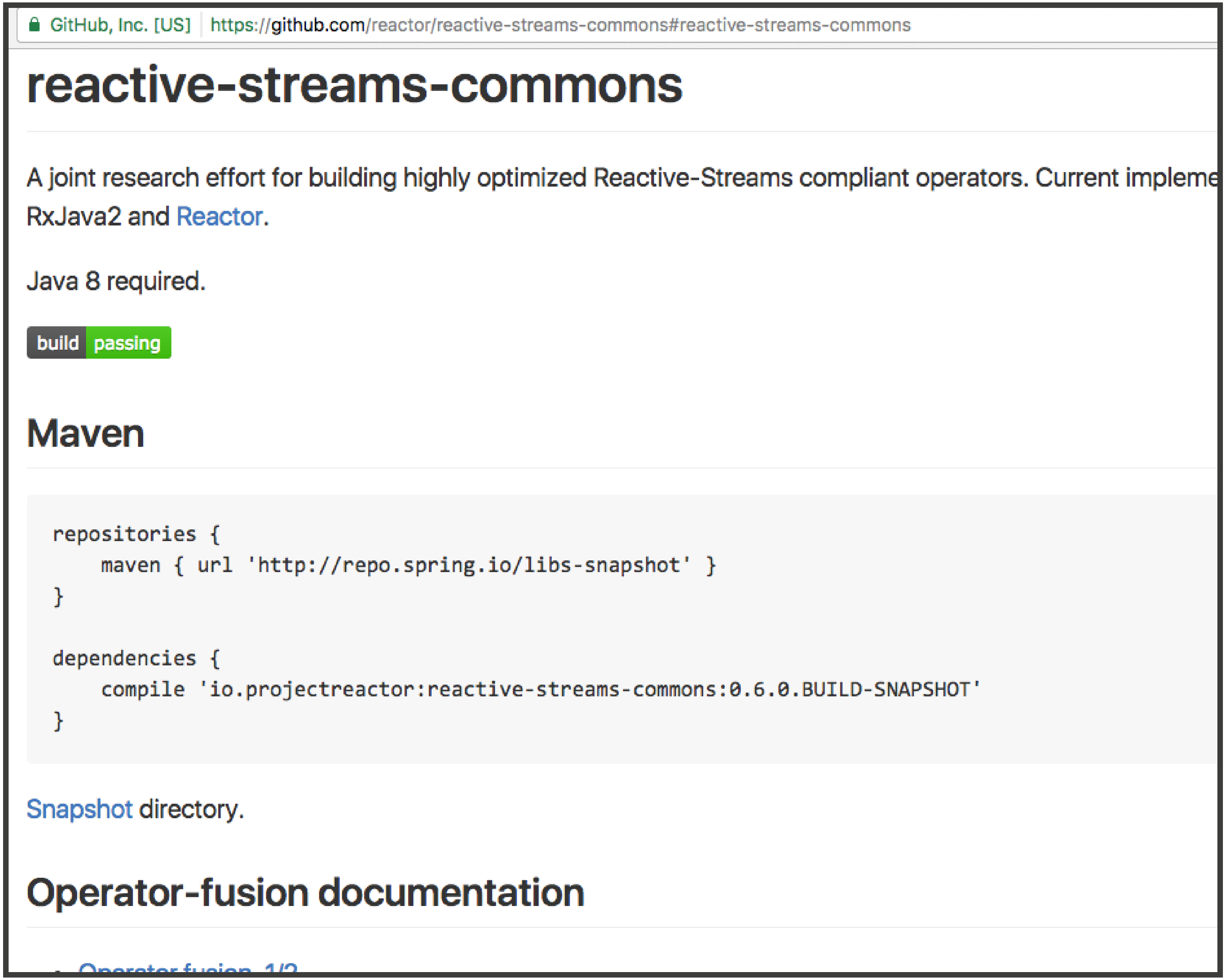
JDK9 FLOW

"Until now, one missing category was "push" style operations on items as they become available from an active source." /Doug Lea

[concurrency-interest] jdk9 Candidate classes Flow and

SPRING WEB REACTIVE





GitHub, Inc. [US] https://github.com/reactor/reactive-streams-commons#reactive-streams-commons

reactive-streams-commons

A joint research effort for building highly optimized Reactive-Streams compliant operators. Current impleme
RxJava2 and Reactor.

Java 8 required.

build passing

Maven

```
repositories {
    maven { url 'http://repo.spring.io/libs-snapshot' }
}

dependencies {
    compile 'io.projectreactor:reactive-streams-commons:0.6.0.BUILD-SNAPSHOT'
}
```

Snapshot directory.

Operator-fusion documentation

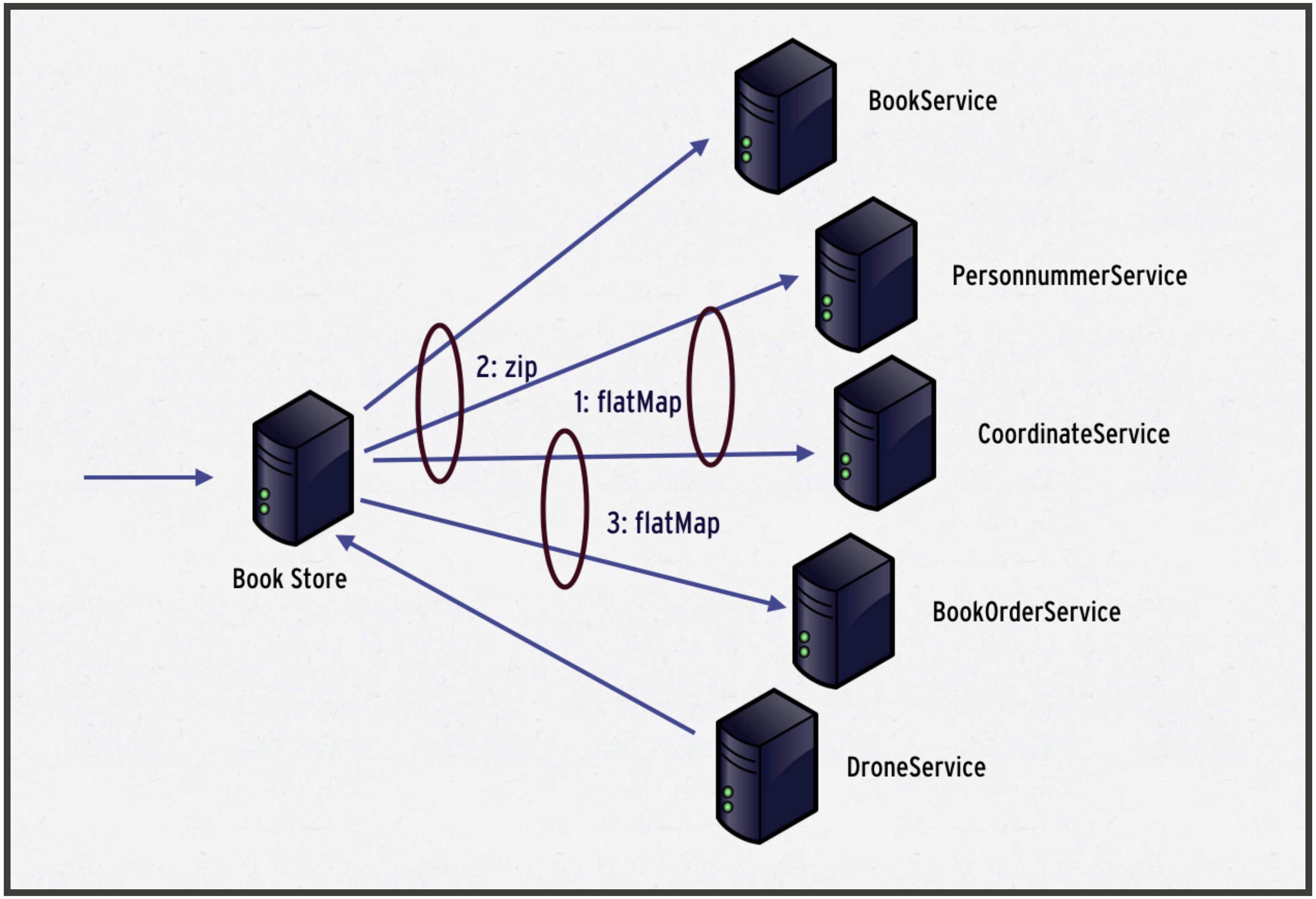
SPRING WEB REACTIVE

```
@RestController
public class PersonController {

        private final PersonRepository repository;

        public PersonController(PersonRepository repository) {
                this.repository = repository;
        }

        @PostMapping("/person")
        Mono<Void> create(@RequestBody Publisher<Person> personStream) {
                return this.repository.save(personStream).then();
        }

        @GetMapping("/person")
        Flux<Person> list() {
                return this.repository.findAll();
        }

        @GetMapping("/person/{id}")
        Mono<Person> findById(@PathVariable String id) {
                return this.repository.findOne(id);
        }
}
```

CONCLUSIONS

- Asynchronous programming is hard
- The Rx model provides abstractions to handle the complexity
- Rx is a valuble tool in the toolbox for the enterprise java developer.

EXAMPLE SERVICE


BookService
PersonnummerService
2: zip
1: flatMap
CoordinateService
3: flatMap
Book Store
BookOrderService
DroneService