



# Reactive REST webservices

Veerle Ongenae

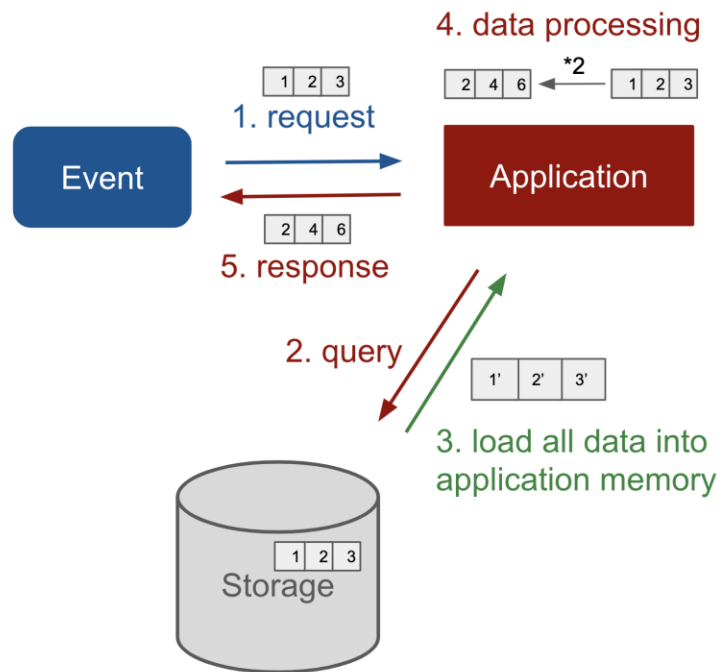


# Overzicht

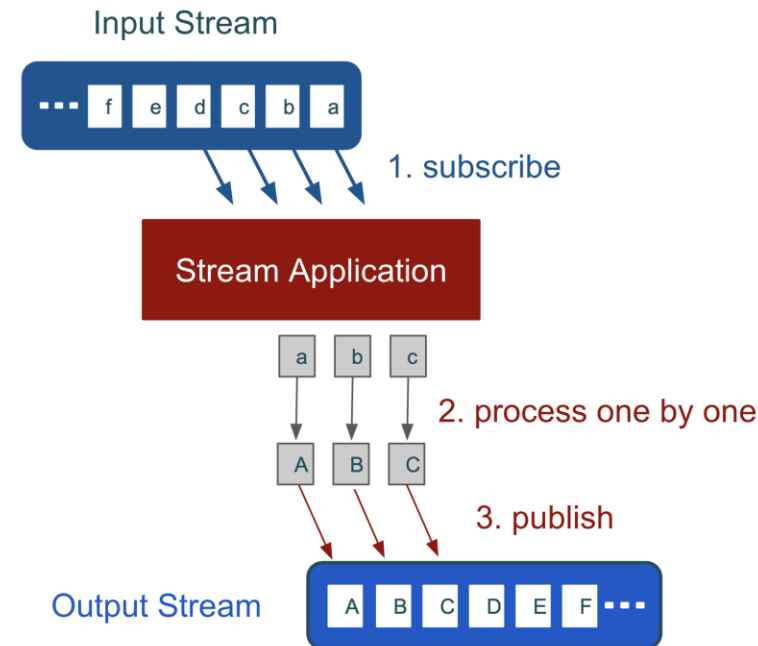
- Herhaling reactive programming



# Waarom Reactive Programming ?



Traditional Data Processing

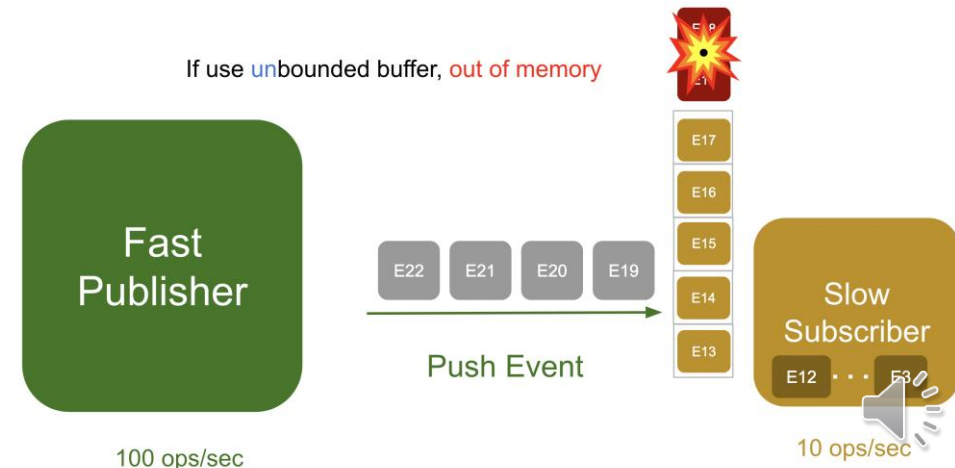
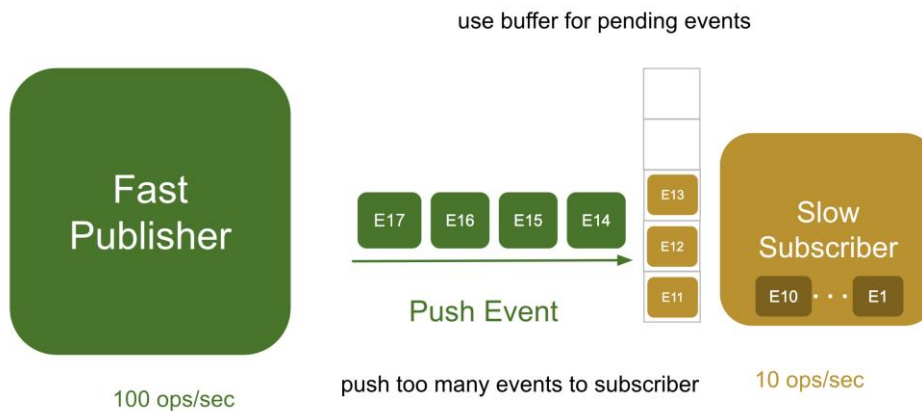
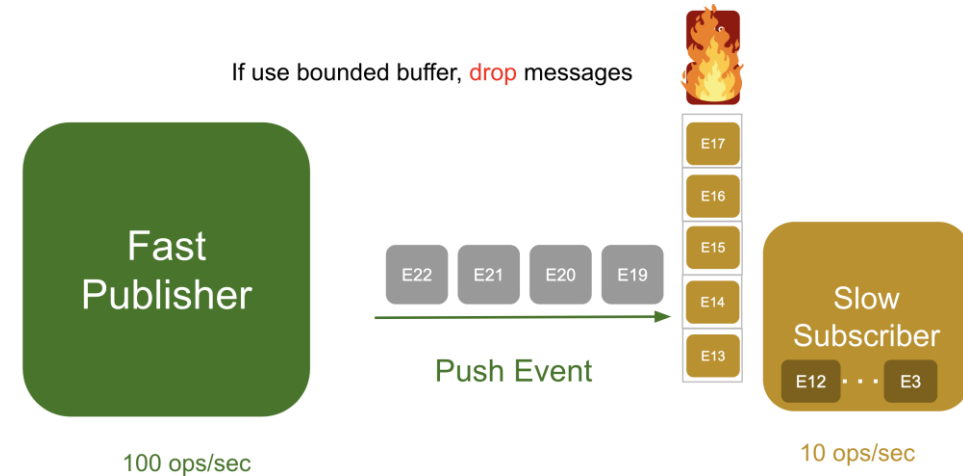
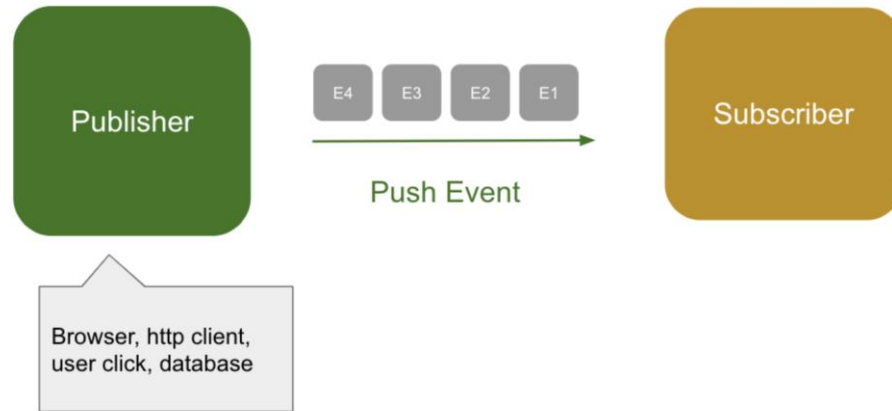


Stream Processing

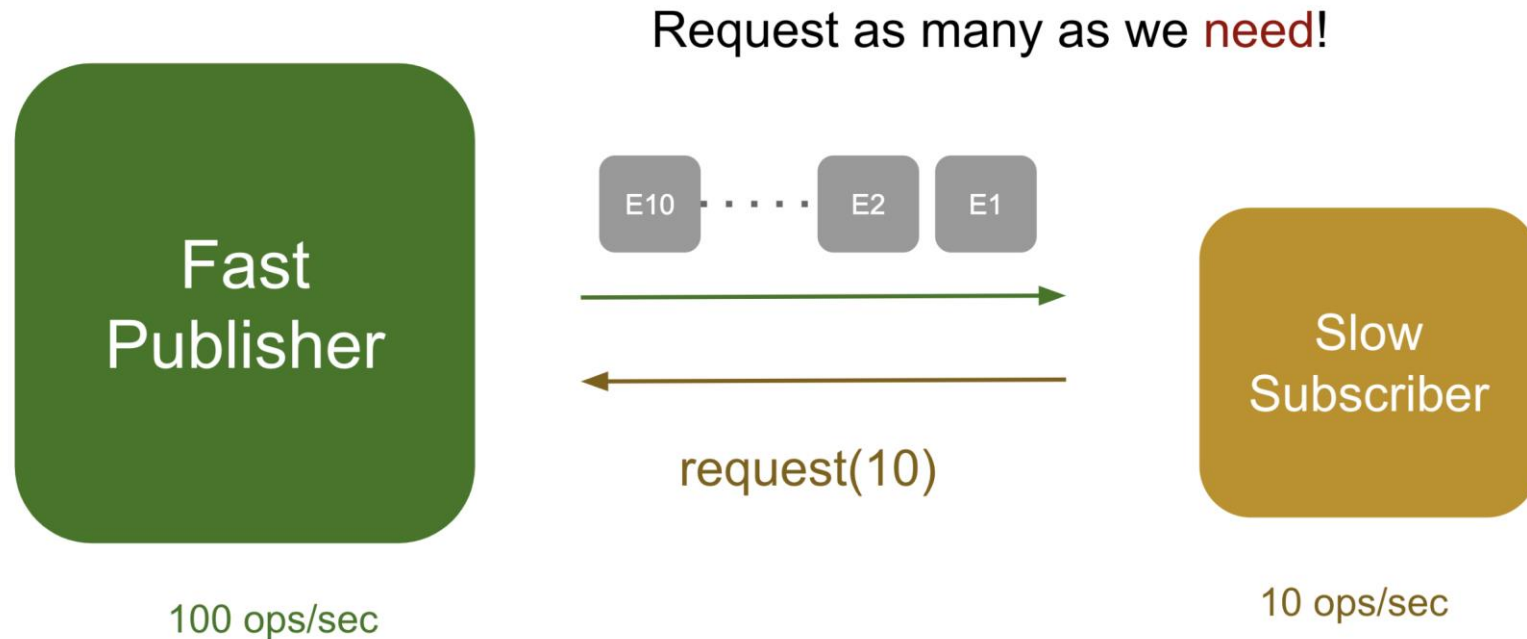
bron: <https://engineering.linecorp.com/en/blog/reactive-streams-armeria-1/>



# Observer - push



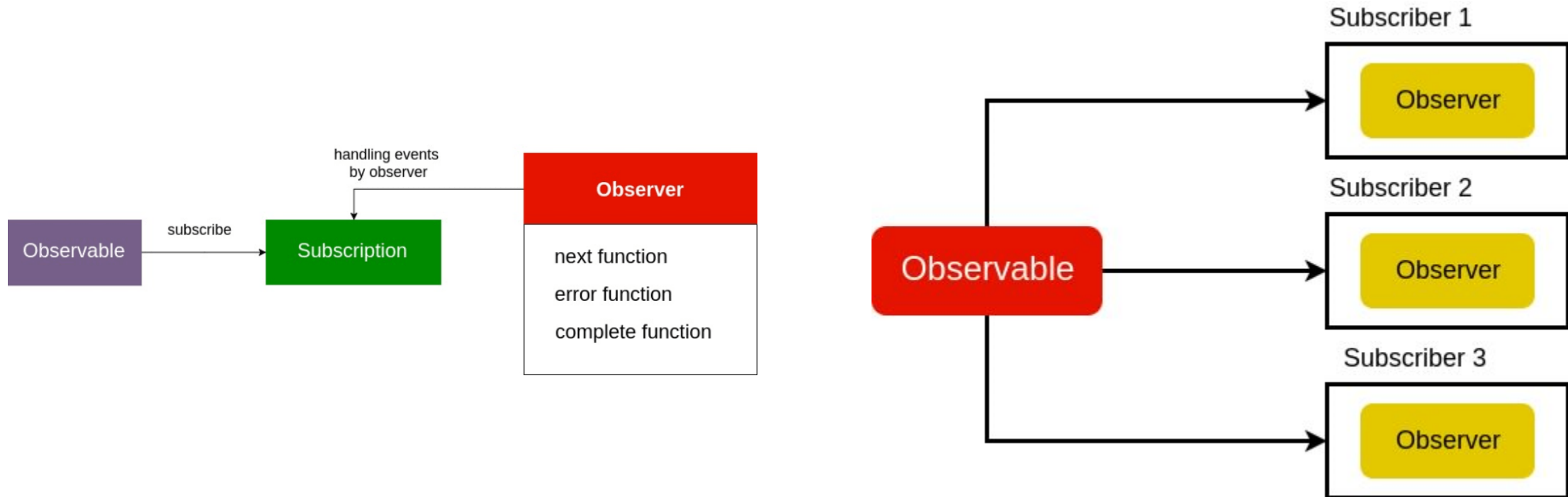
# Observer - pull



**Backpressure (tegendruk)** in softwaresystemen is het vermogen om de **communicatie te overbelasten**. Met andere woorden, zenders van informatie overstelpen consumenten met gegevens die zij niet kunnen verwerken. Men past deze term ook toe als **het mechanisme om dit te controleren en af te handelen**.



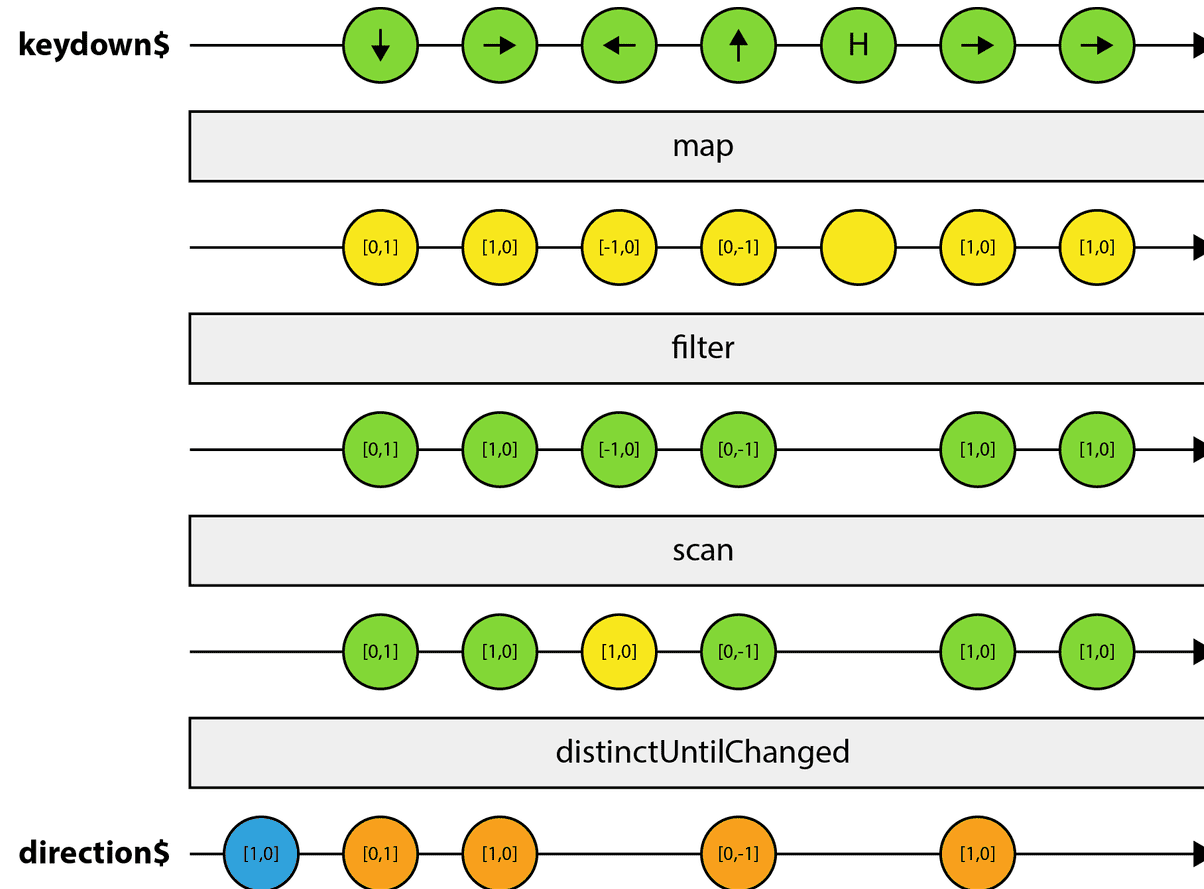
# Reactive Programming - principe



bron: <https://dev.to/sagar/reactive-programming-in-javascript-with-rxjs-4jom>



# Reactive Programming - pipeline



<https://blog.thoughttram.io/rxjs/2017/08/24/taming-snakes-with-reactive-streams.html>



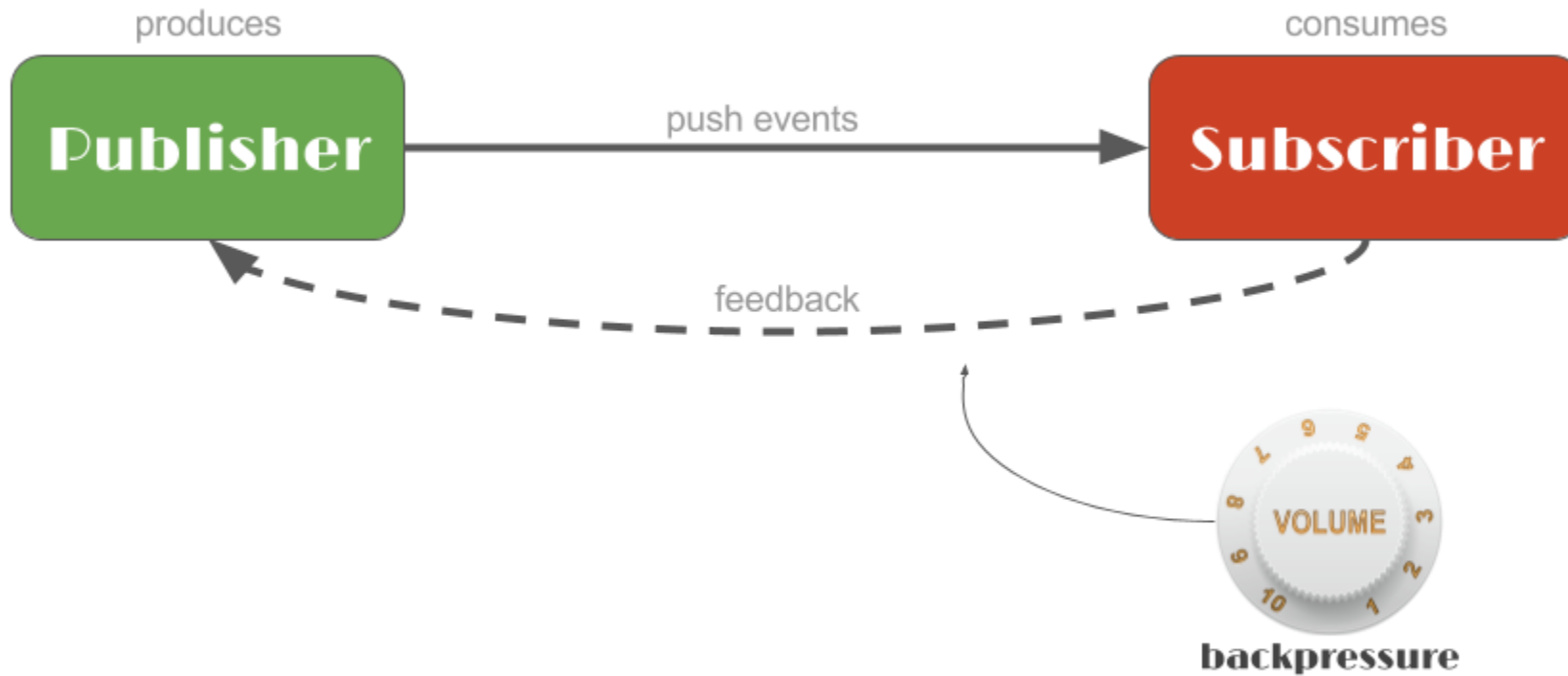
# Overzicht

- Herhaling reactive programming
- Reactive Streams





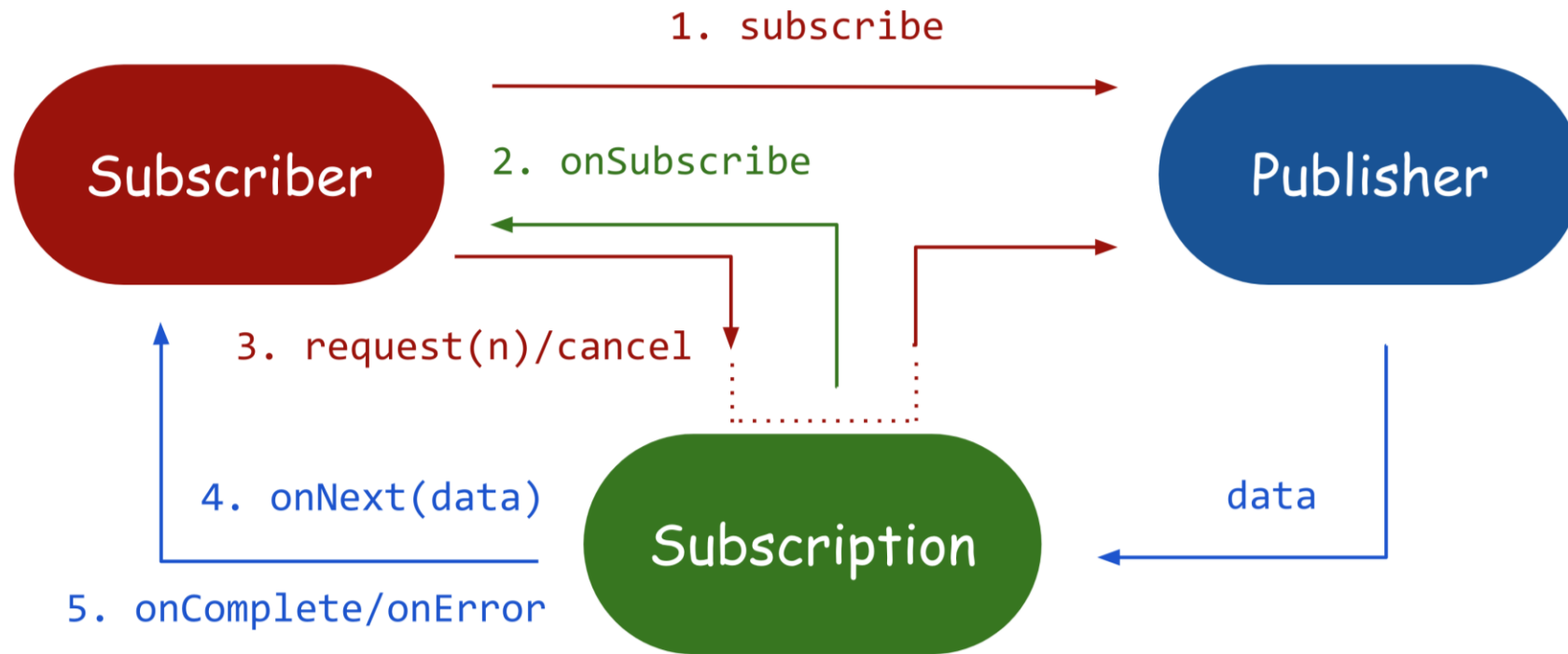
# Reactive Streams - concept



bron: <https://tech.io/playgrounds/929/reactive-programming-with-reactor-3/Intro>



# Reactive Streams API (Java)



bron: <https://engineering.linecorp.com/en/blog/reactive-streams-armeria-1/>

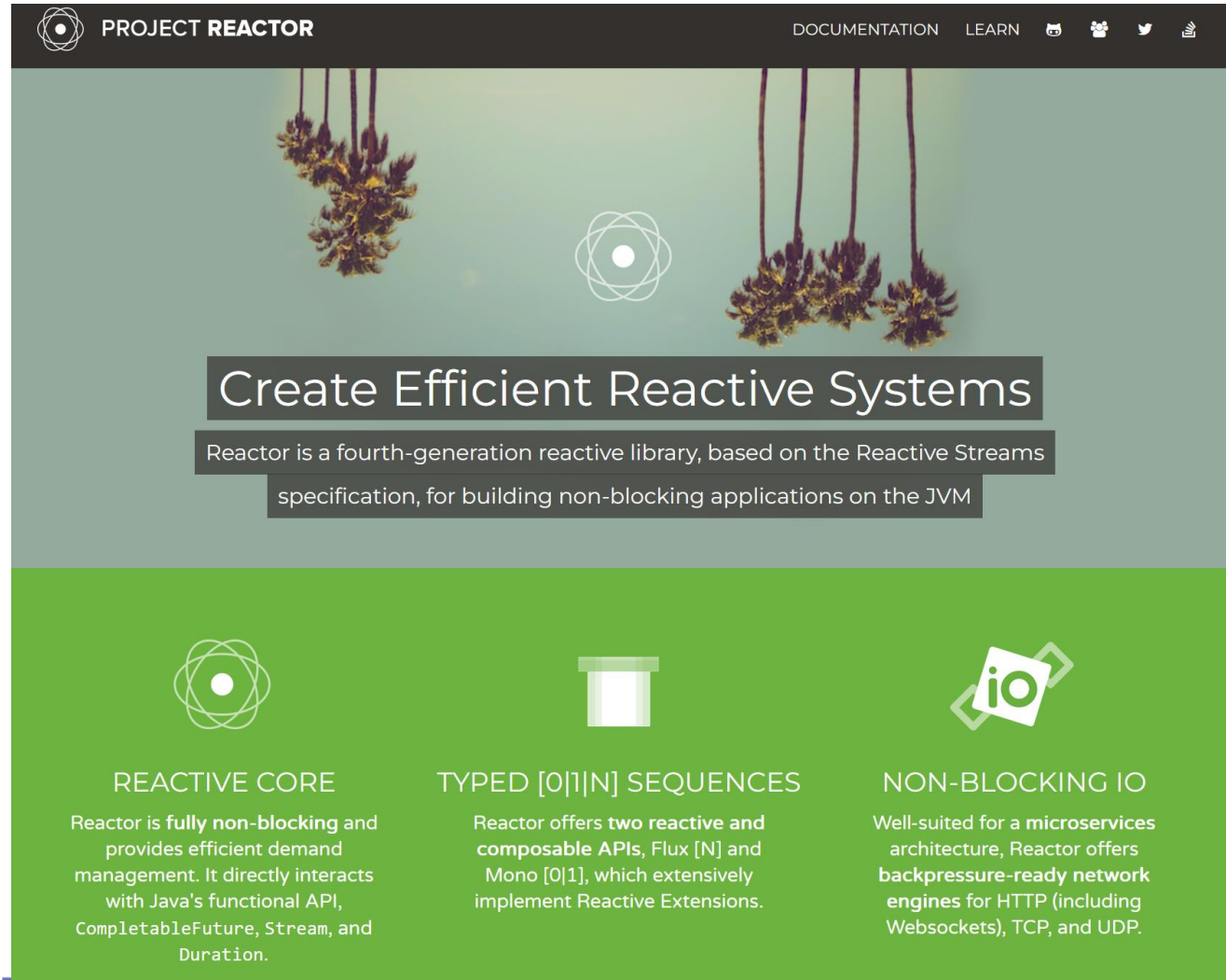


# Overzicht

- Herhaling reactive programming
- Reactive Streams
- Reactor



# Reactor



The image shows a screenshot of the Project Reactor website. The header features the Project Reactor logo and navigation links for Documentation, Learn, and social media. The main section has a background image of palm trees and a central text box that reads 'Create Efficient Reactive Systems'. Below this, a subtext box states: 'Reactor is a fourth-generation reactive library, based on the Reactive Streams specification, for building non-blocking applications on the JVM'. The bottom section is a green banner with three columns: 'REACTIVE CORE' (describing non-blocking and demand management), 'TYPED [0|1|N] SEQUENCES' (describing reactive and composable APIs like Flux and Mono), and 'NON-BLOCKING IO' (describing suitability for microservices and backpressure-ready network engines).

PROJECT REACTOR

DOCUMENTATION LEARN

Create Efficient Reactive Systems

Reactor is a fourth-generation reactive library, based on the Reactive Streams specification, for building non-blocking applications on the JVM

REACTIVE CORE

Reactor is **fully non-blocking** and provides efficient demand management. It directly interacts with Java's functional API, `CompletableFuture`, `Stream`, and `Duration`.

TYPED [0|1|N] SEQUENCES

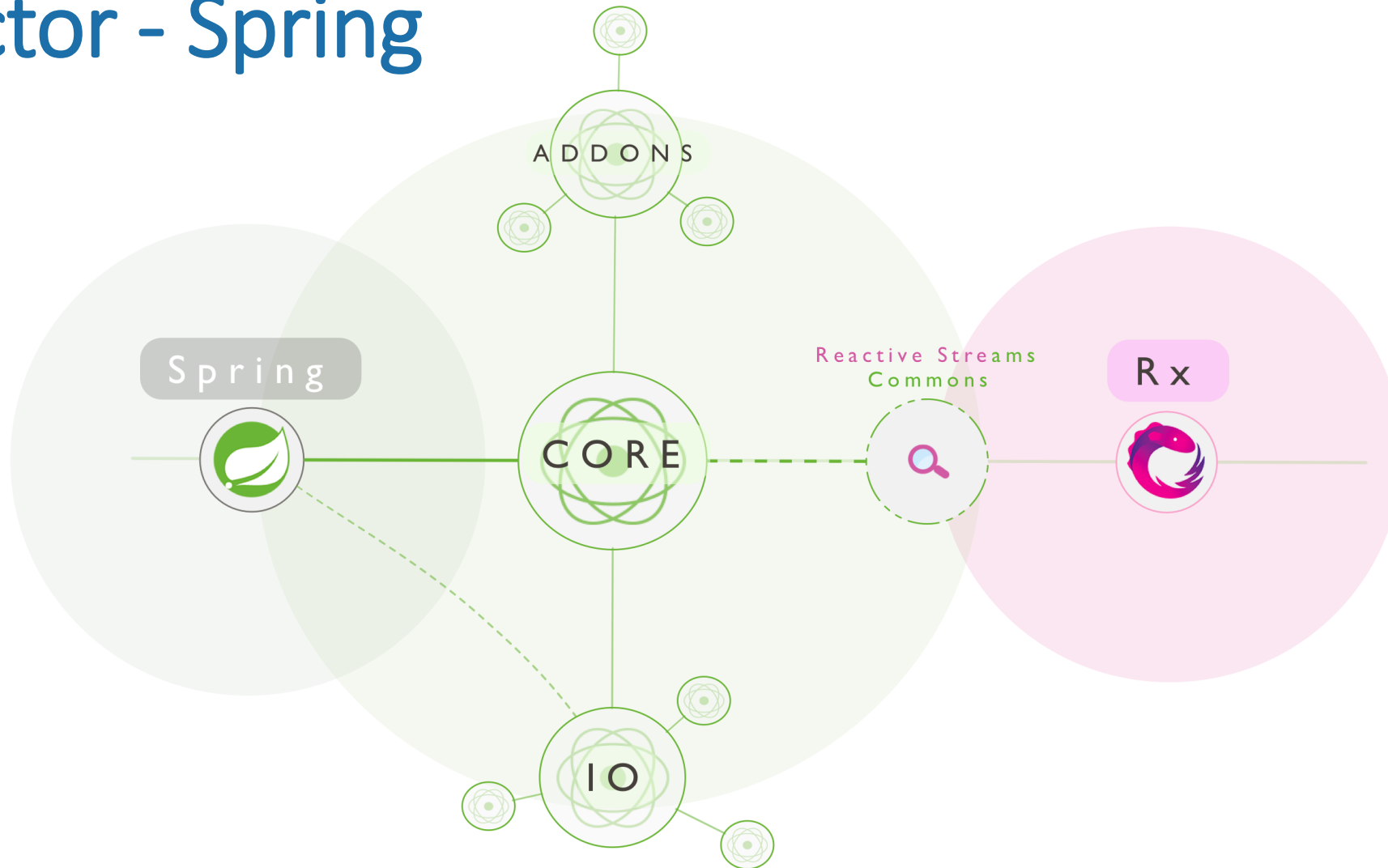
Reactor offers **two reactive and composable APIs**, `Flux [N]` and `Mono [0|1]`, which extensively implement Reactive Extensions.

NON-BLOCKING IO

Well-suited for a **microservices** architecture, Reactor offers **backpressure-ready network engines** for HTTP (including Websockets), TCP, and UDP.



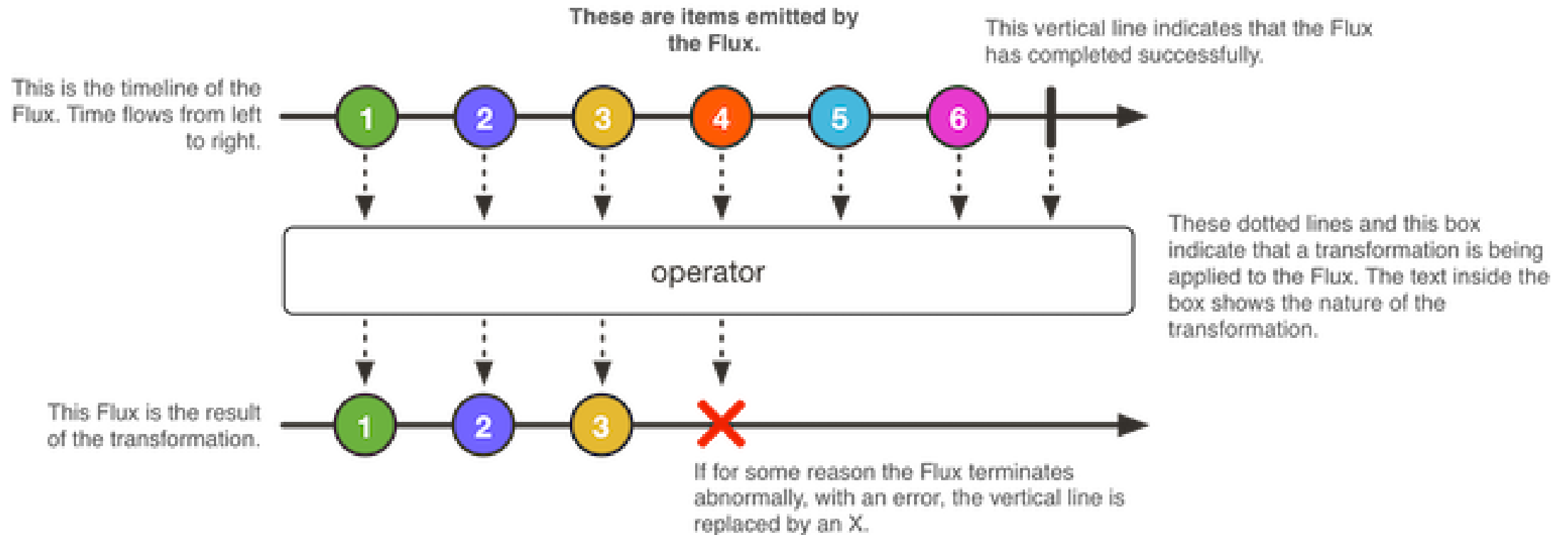
# Reactor - Spring



bron: <https://spring.io/blog/2016/03/11/reactor-core-3-0-becomes-a-unified-reactive-foundation-on-java-8>



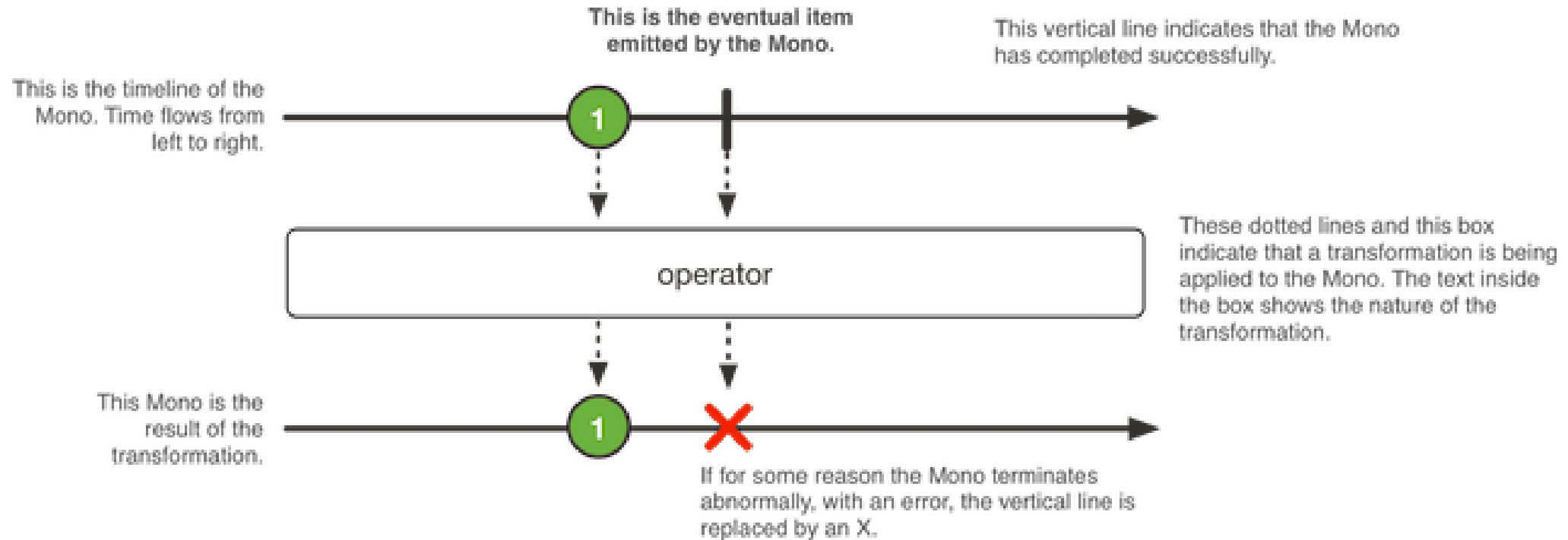
# Reactor - Flux



bron: <https://tech.io/playgrounds/929/reactive-programming-with-reactor-3/Flux>



# Reactor - Mono



bron: <https://tech.io/playgrounds/929/reactive-programming-with-reactor-3/Mono>



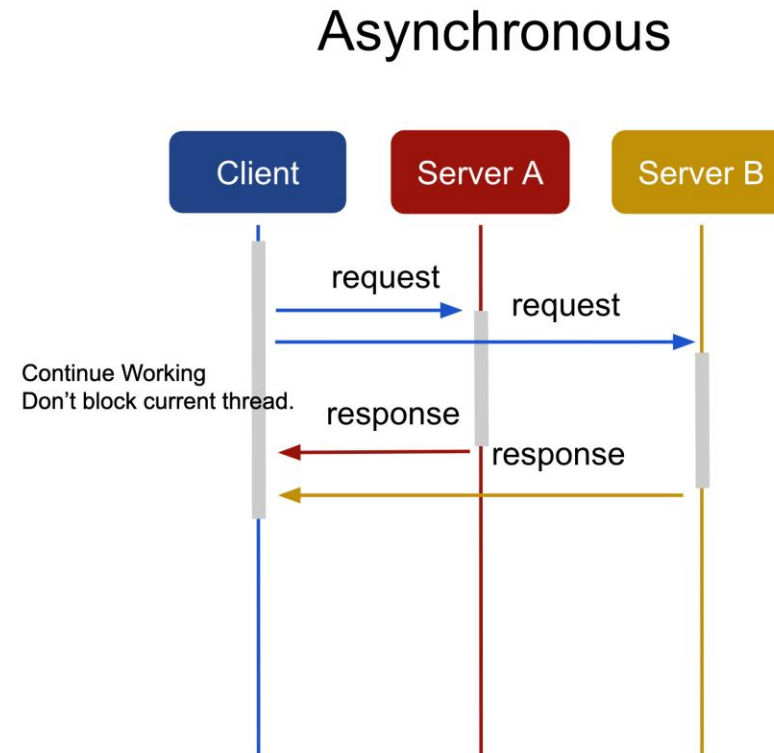
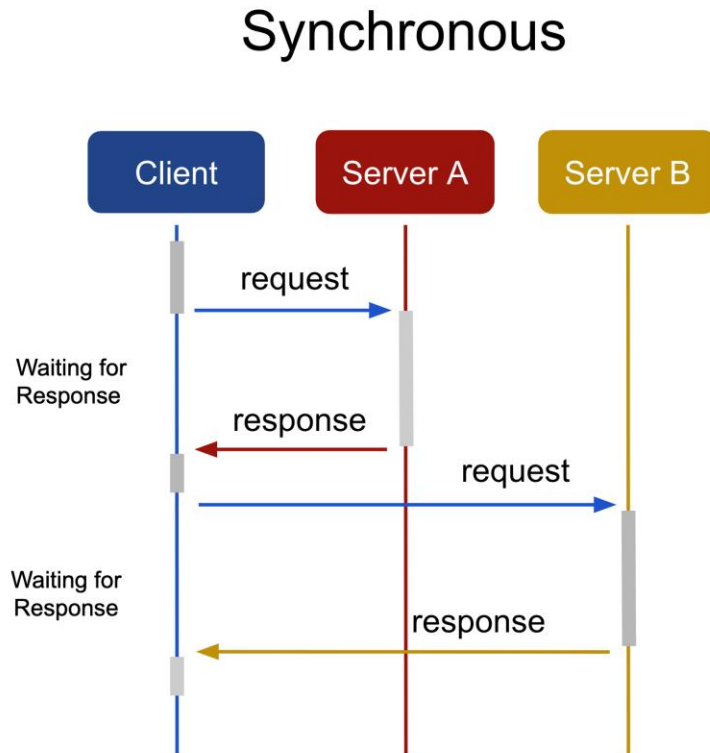
# Overzicht

- Herhaling reactive programming
- Reactive Streams
- Reactor
- Reactive REST-webservice





# Asynchrone versus synchrone client

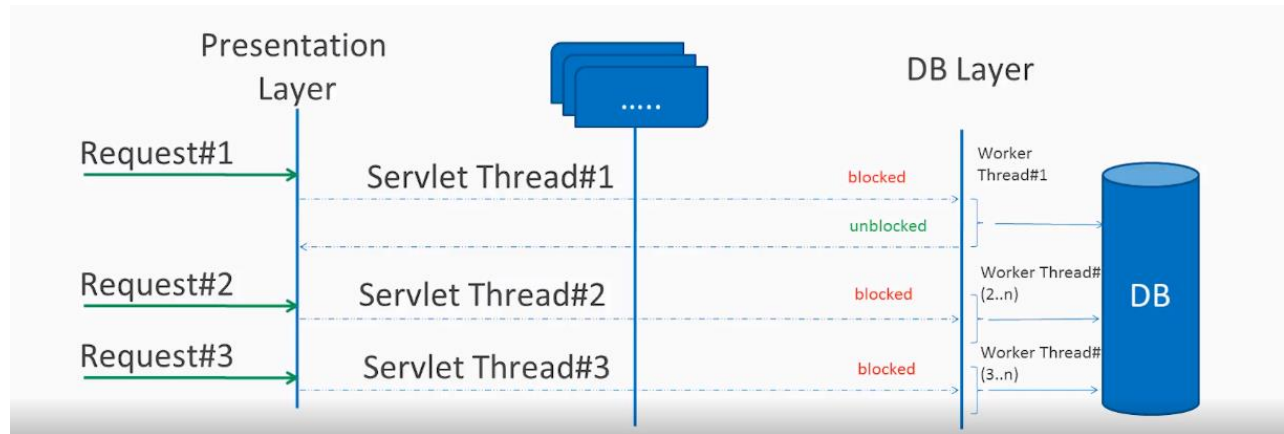


sneller  
minder resources

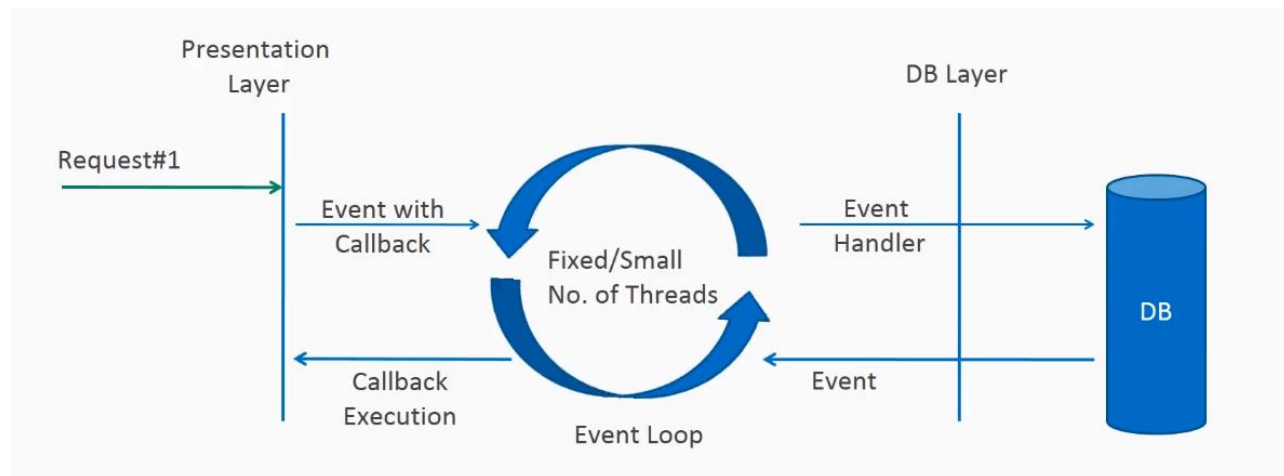
bron: <https://engineering.linecorp.com/en/blog/reactive-streams-armeria-1/>



# Asynchrone versus synchrone server



Blocking request processing

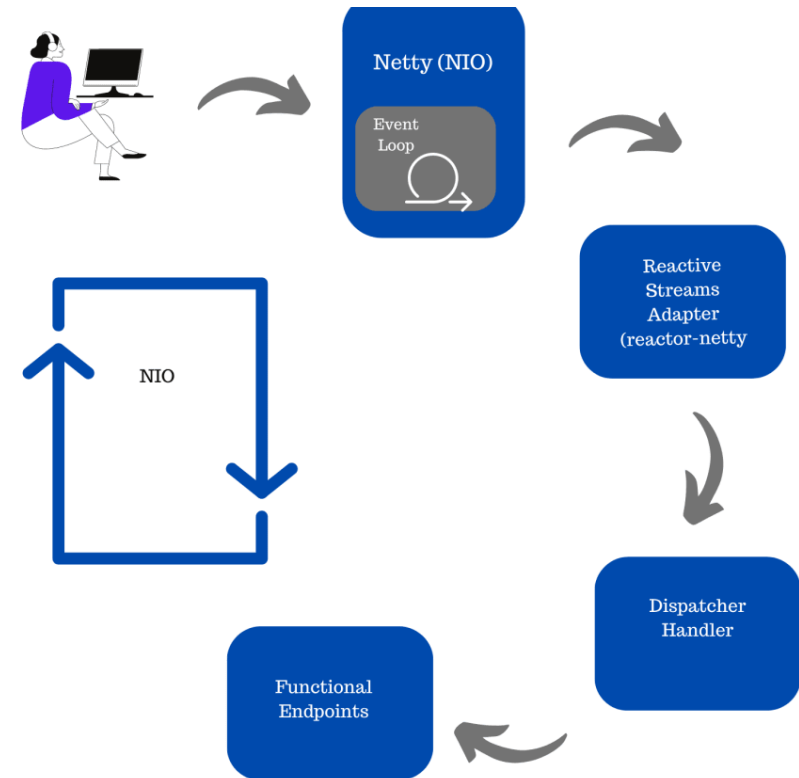
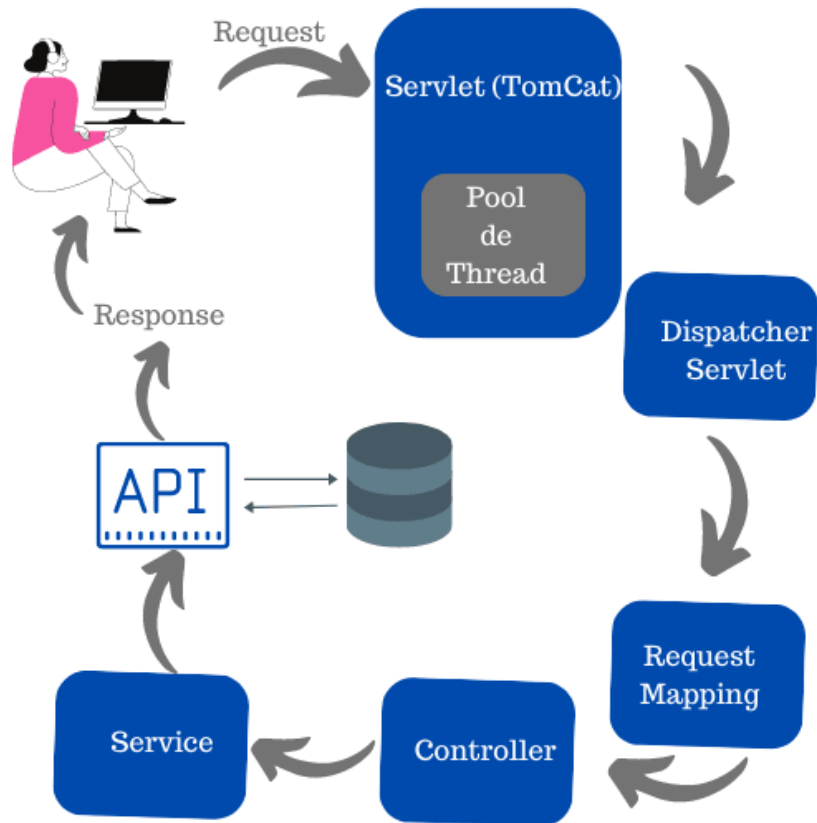


Non-blocking request processing

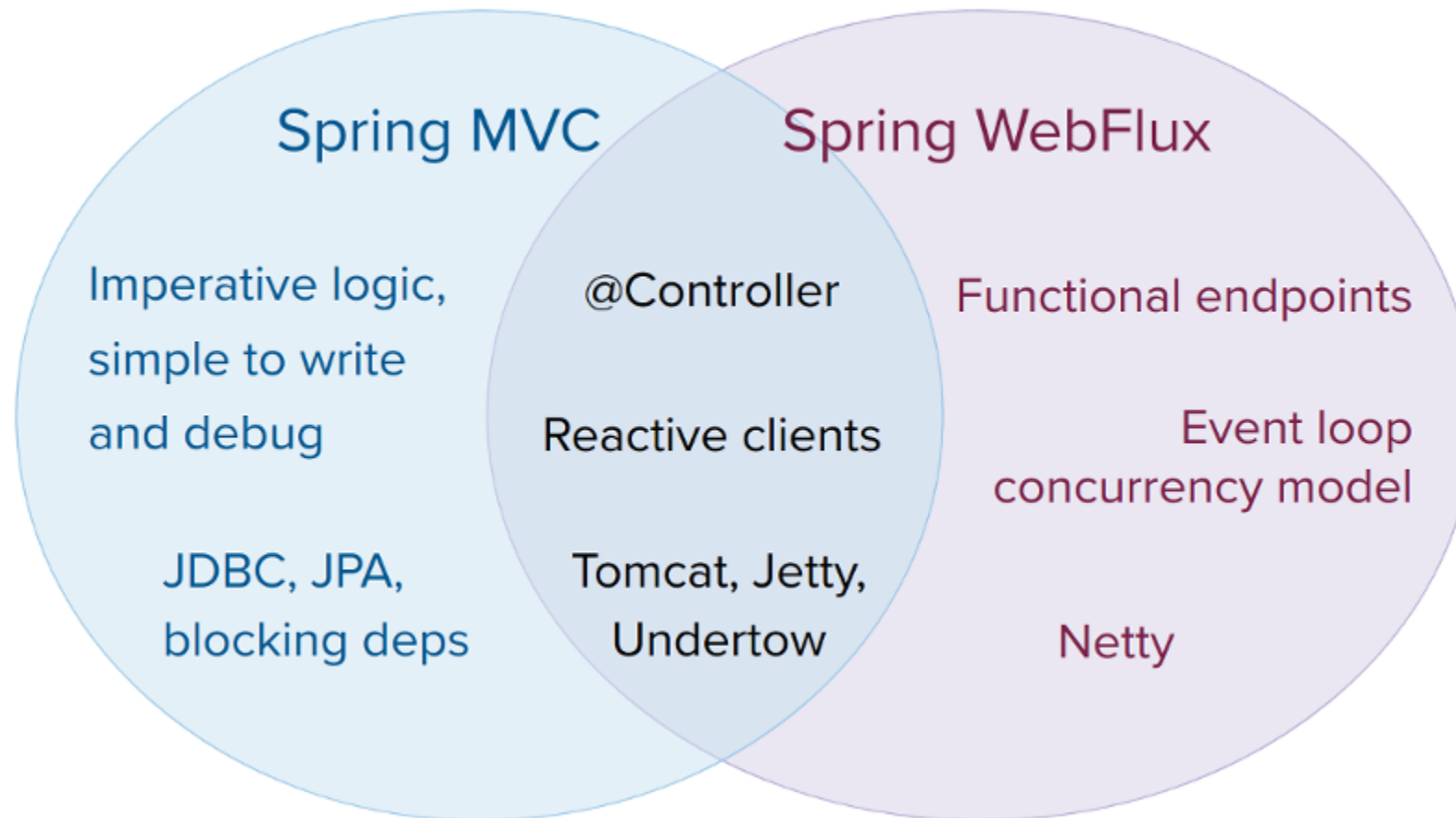
bron: <https://howtodoinjava.com/spring-webflux/spring-webflux-tutorial/>



# Spring MVC versus Spring WebFlux - Architectuur



# Spring MVC versus Spring WebFlux



bron: <https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html>





# Spring Boot 2

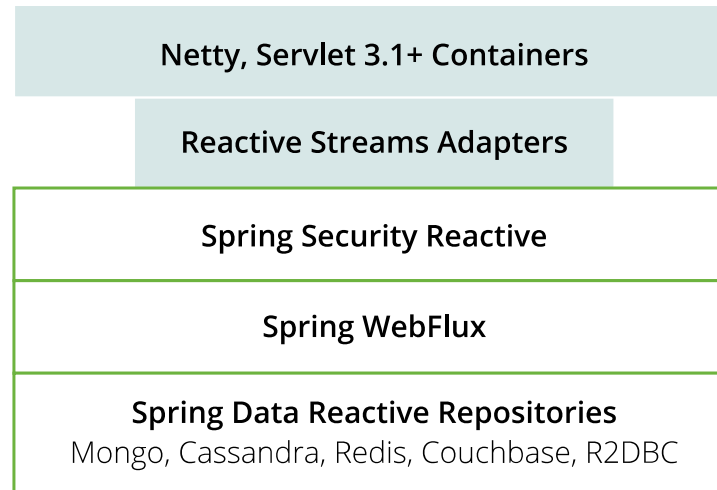


## Reactor

Optional Dependency

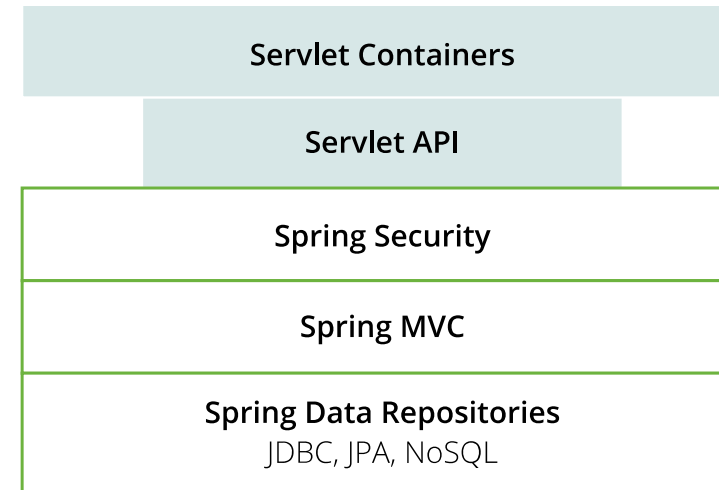
### Reactive Stack

Spring WebFlux is a non-blocking web framework built from the ground up to take advantage of multi-core, next-generation processors and handle massive numbers of concurrent connections.



### Servlet Stack

Spring MVC is built on the Servlet API and uses a synchronous blocking I/O architecture with a one-request-per-thread model.



bron: <https://spring.io/reactive>

Industrieel Ingenieur Informatica, UGent



# Spring WebFlux

## - Twee programmeermodellen

- Reactive components met **annotaties**
  - Zelfde annotaties als Spring MVC @RestController, ...
  - Resultaat methodes: Mono of Flux
  - GEEN blocking methodes gebruiken
- **Functional** Routing and Handling
  - Klasse met methodes die bepaalde aanvragen afhandelen (Handler)
    - Parameter: request-object
    - Resultaat: Mono met response-object
  - Configuratie: routing
    - Pad ~ methode handler



# Reactive DAO

```
@Service
public class BoorputDAO {
    final Random random = new Random();
    String[] ids = {"BP1", "BP2", "BP3"};

    public Flux<Boorput> geefMetingen() {
        return Flux.interval(Duration.ofSeconds(1)).take(10)
            .map(pulse -> geefMeting());
    }

    private Boorput geefMeting() {
        Boorput boorput = new Boorput();
        boorput.setId(ids[random.nextInt(ids.length)]);
        boorput.setTijdstipDebiet(LocalDateTime.now());
        boorput.setTijdstipPeil(LocalDateTime.now());
        boorput.setPeil(28 + 5 * random.nextDouble());
        boorput.setDebiet(5 + 2 * random.nextDouble());
        return boorput;
    }
}
```



# Reactive REST-webservice

```
@RestController
@RequestMapping("boorputten")
public class BoorputController {

    private BoorputDAO dao;

    public BoorputController(BoorputDAO dao) {
        this.dao = dao;
    }

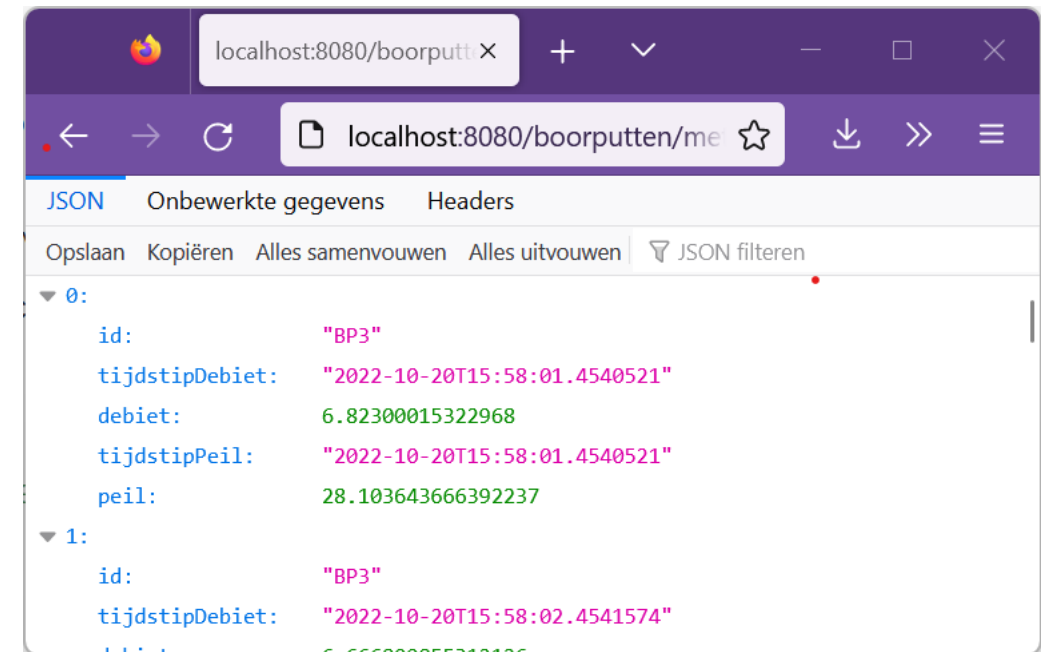
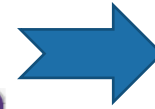
    @GetMapping("metingen")
    public Flux<Boorput> haalMetingen() {
        return dao.geefMetingen();
    }
}
```





# Browser: blocking/blokkerende client

even wachten



# Webclient: niet blokkerend (non blocking) client

```
System.out.println("start test");
ClientTestWebflux client = new ClientTestWebflux();
client.testServer();
System.out.println("na test");
```

```
public class ClientTestWebflux {

    public void testServer() {
        WebClient client = WebClient.create("http://localhost:8080");

        Flux<Boorput> boorputFlux = client.get()
            .uri("/boorputten/metingen")
            .retrieve()
            .bodyToFlux(Boorput.class);

        boorputFlux.subscribe(System.out::println);
        System.out.println("in test webflux");
    }
}
```

```
start test
in test webflux
na test
iii.vbWebflux.web.Boorput@7d721e6c
iii.vbWebflux.web.Boorput@6514c4aa
iii.vbWebflux.web.Boorput@70fe8219
iii.vbWebflux.web.Boorput@2063102e
iii.vbWebflux.web.Boorput@4d8feb30
iii.vbWebflux.web.Boorput@e7d47bf
iii.vbWebflux.web.Boorput@a97f223
iii.vbWebflux.web.Boorput@324d613f
iii.vbWebflux.web.Boorput@a9ca2d
iii.vbWebflux.web.Boorput@648d2538
```



# Spring WebFlux

## - Twee programmeermodellen

- Reactive components met **annotaties**
  - Zelfde annotaties als Spring MVC @RestController, ...
  - Resultaat methodes: Mono of Flux
  - GEEN blocking methodes gebruiken
- **Functional** Routing and Handling
  - Klasse met methodes die bepaalde aanvragen afhandelen (Handler)
    - Parameter: request-object
    - Resultaat: Mono met response-object
  - Configuratie: routing
    - Pad ~ methode handler



# Reactive REST-webservice - handler

```
@Component
public class BoorputHandler {
    private BoorputDAO dao;

    public BoorputHandler(BoorputDAO dao) {
        this.dao = dao;
    }
    public Mono<ServerResponse> metingenVoorBoorputten(ServerRequest request) {
        return ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
            .body(dao.geefMetingen(), Boorput.class);
    }
}
```

```
@FunctionalInterface
public interface HandlerFunction<T extends ServerResponse> {
    Mono<T> handle(ServerRequest request);
}
```



# Reactive REST-webservice - routing

## @Configuration

```
public class BoorputRouter {  
    @Bean  
    public RouterFunction<ServerResponse> routeBoorput(BoorputHandler boorputHandler) {  
  
        return RouterFunctions.route(RequestPredicates.GET("/boorputten/metingenAnders")  
            .and(RequestPredicates.accept(MediaType.APPLICATION_JSON)),  
            boorputHandler::metingenVoorBoorputten);  
    }  
}
```

## @FunctionalInterface

```
public interface RouterFunction<T extends ServerResponse> {  
    Mono<HandlerFunction<T>> route(ServerRequest request);  
    // ...  
}
```

```
public static <T extends ServerResponse> RouterFunction<T> route(  
    RequestPredicate predicate,  
    HandlerFunction<T> handlerFunction)
```

hulpfunctie



# Spring Data Reactive

- MongoDB
  - Document-georiënteerde databank (JSON)
- Cassandra
  - Mix tussen key-value store en databank gebruikmakend van tabellen (wide column store)
- Redis
  - In-memory key-value database
- NoSql databases



# Overzicht

- Herhaling reactive programming
- Reactive Streams
- Reactor
- Reactive REST-webservice

