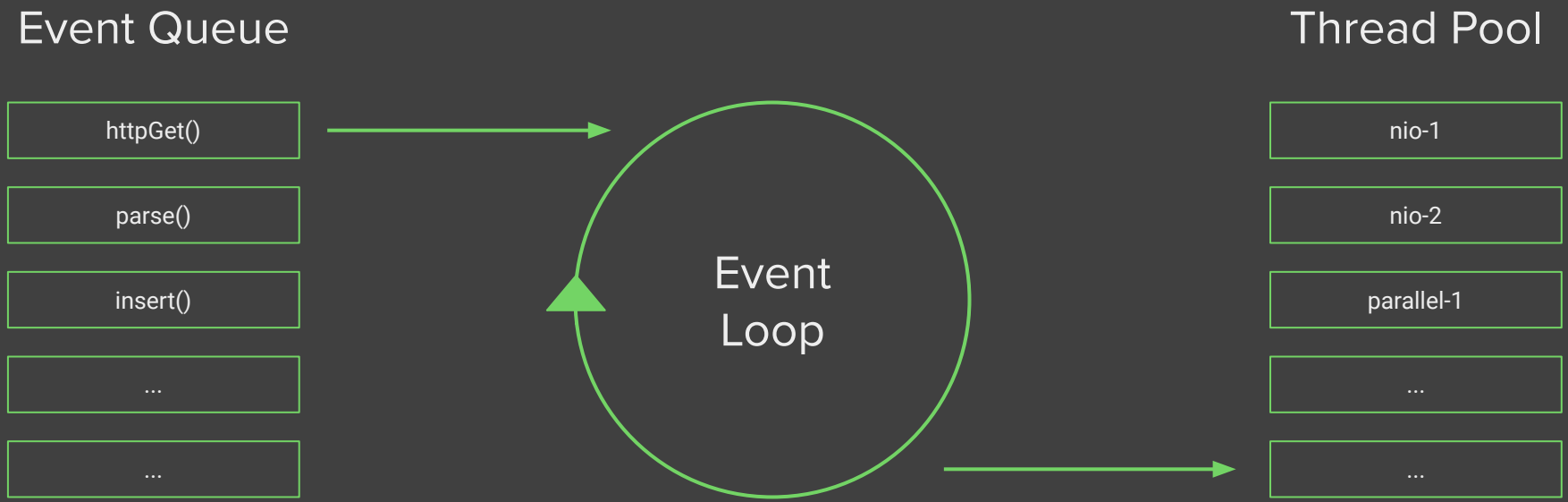
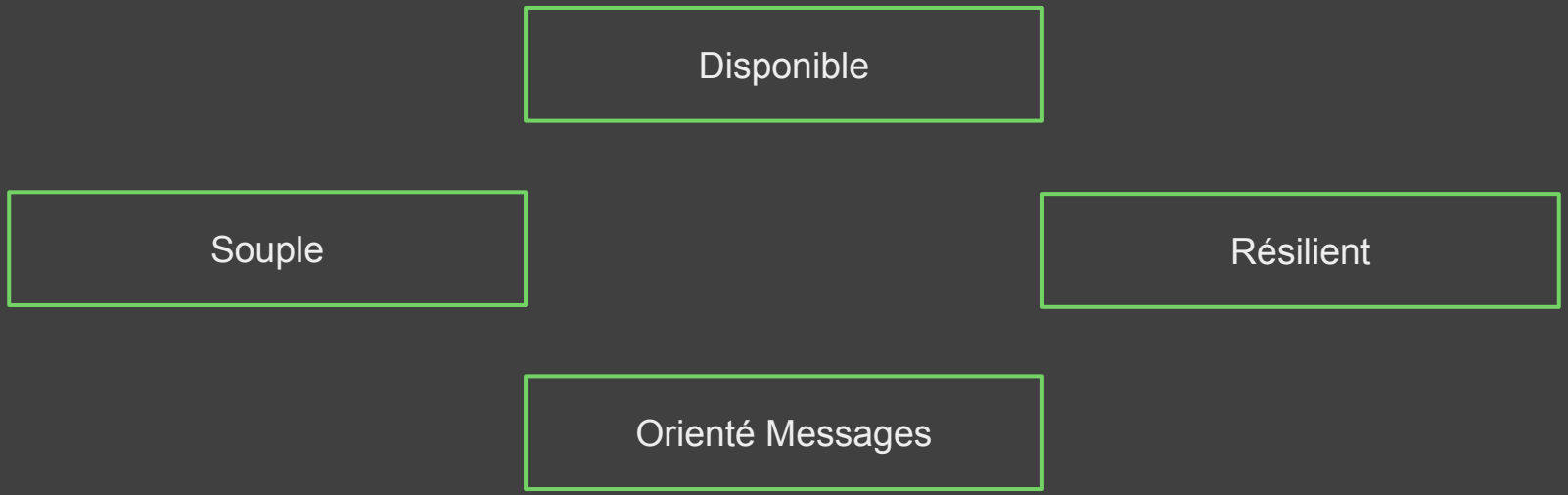
REACTOR

NightClazz

Principe


Event Queue
httpGet()
parse()
insert()
...
...
Event Loop
Thread Pool
nio-1
nio-2
parallel-1
...
...

Reactive Manifesto



https://www.reactivemanifesto.org/fr

Reactive Streams

- Défini par les acteurs travaillants sur la JVM
- Offre un contrat d'interopérabilité
- Définit quatre composants
  - Publisher
  - Subscriber
  - Subscription
  - Processor

https://www.reactive-streams.org/

Programmation Réactive VS Impérative

```
Flux.just("Element one","Element two")
    .subscribe(System.out::println);
```

≠

```
Future o = httpRequest(); // HTTP request sent as soon as the future is created
o.get();
```

Reactor 3

- Version actuelle 3.2.9
- Porté par Pivotal
- Basé sur reactive-streams
- Supporté par Spring 5 & Spring Boot 2
- Framework Spring compliant
  - SpringWeb
  - SpringData
  - SpringSecurity
  - ...

https://projectreactor.io

Reactor / dépendances

spring-boot-starter-webflux

reactor-netty

spring-boot-starter-data-mongodb-reactive

...

reactor-core

reactor-test

reactive-streams

Qui sommes nous ?

Sébastien VELAY
Consultant & Formateur @ Zenika Lille
Twitter : @seb_ve

Mickael BOIXIERE
Consultant & Formateur @ Zenika Lille
Twitter : @MickaelBoixiere

Le code lab

Repository GIT

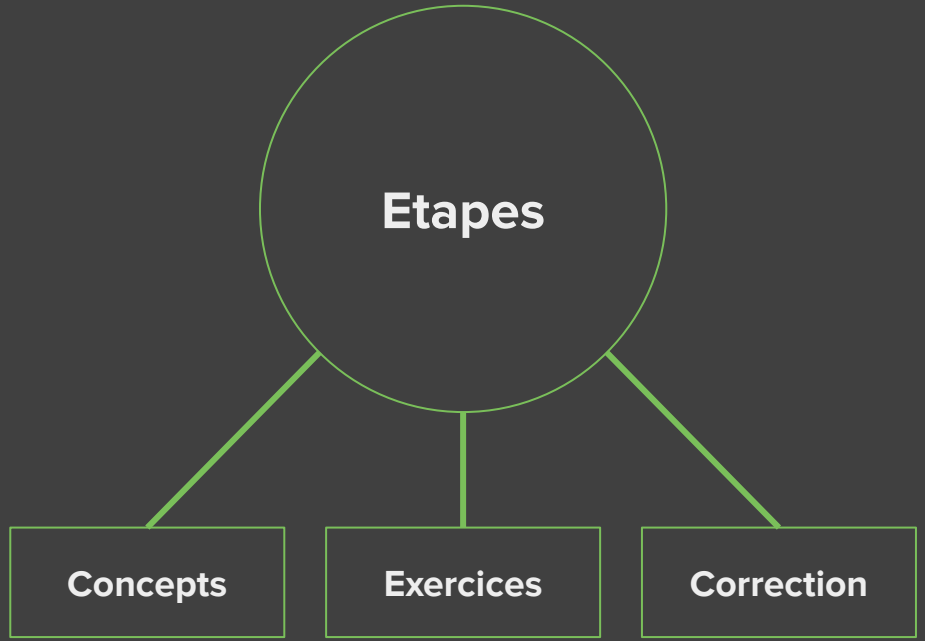
- Cloner le projet

  https://github.com/Zenika/nc-spring-reactor

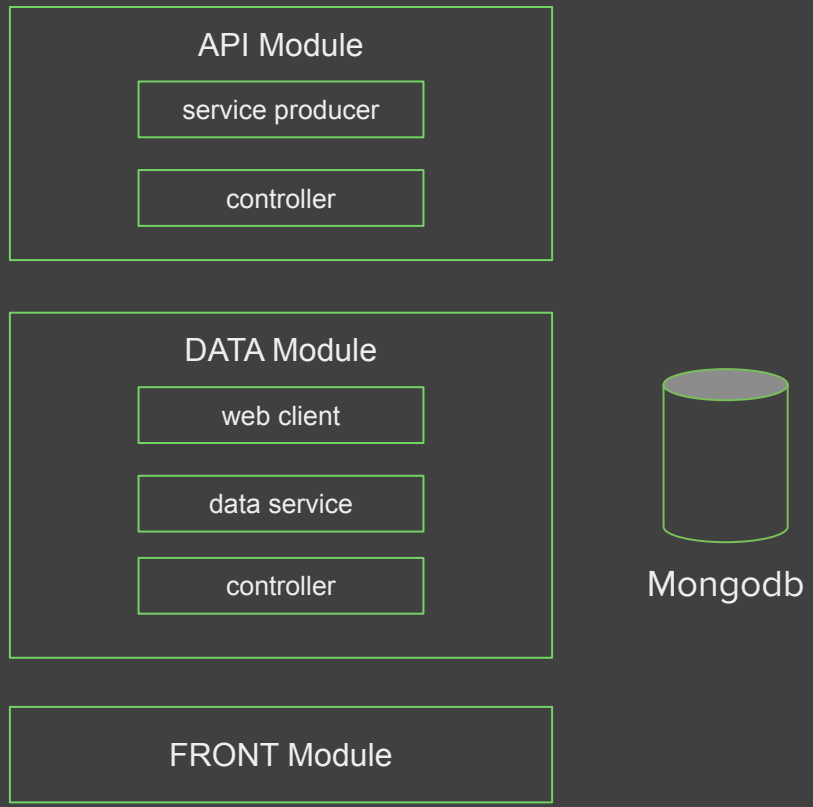
- Checkout de la branche **step1**

- “mvn clean install” -> TU failed ;)

Déroulement



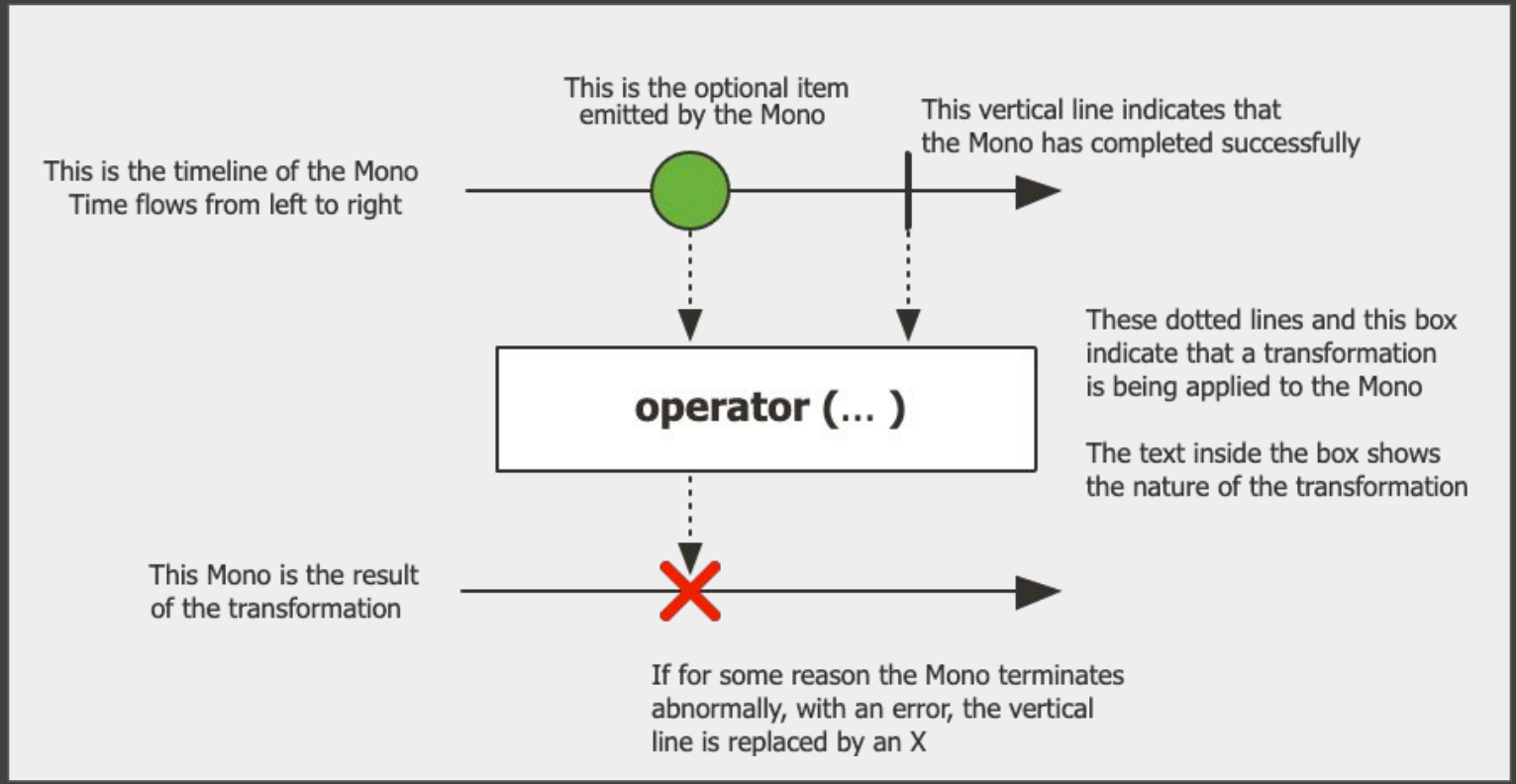
Architecture



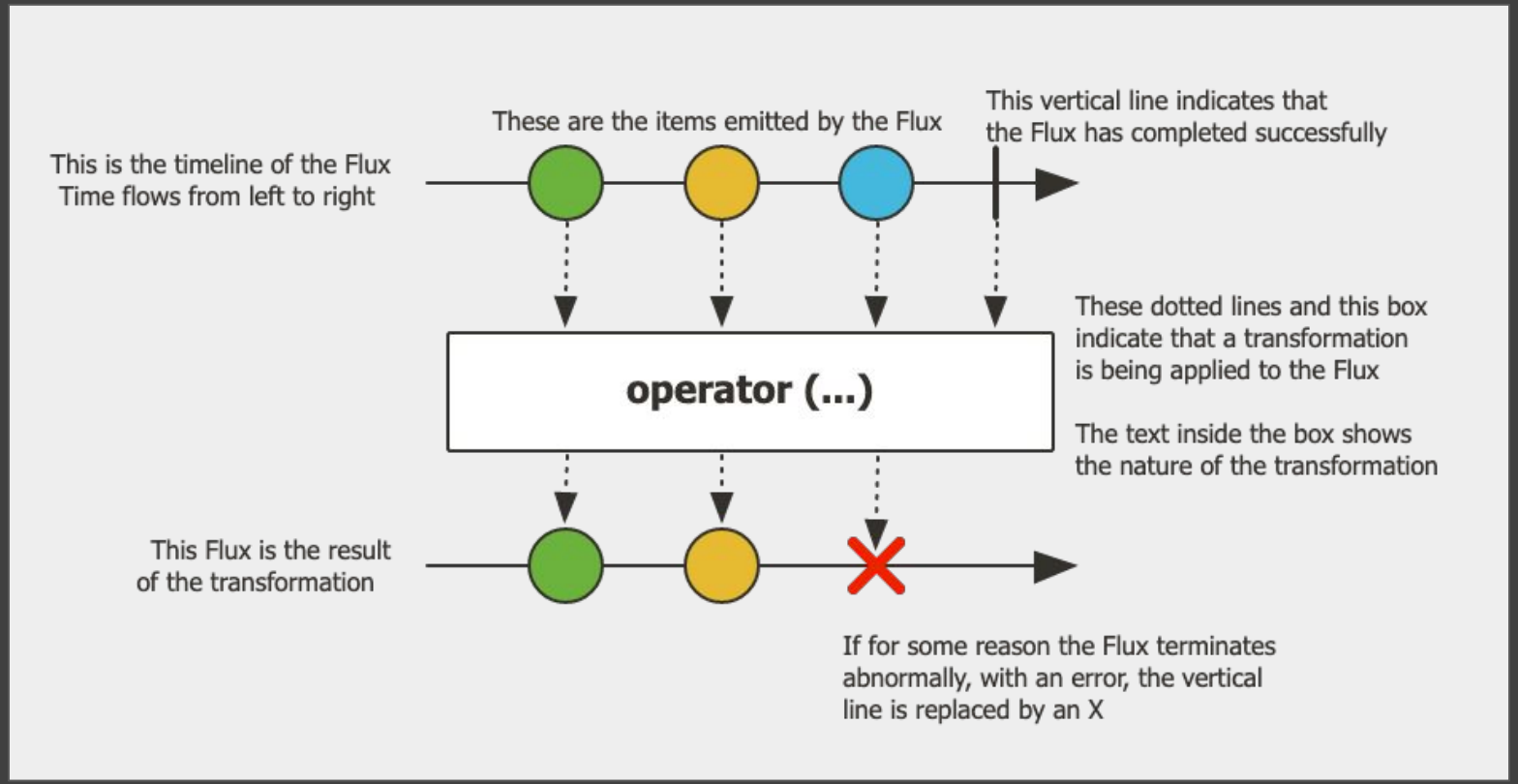
Step 1

Mono & Flux

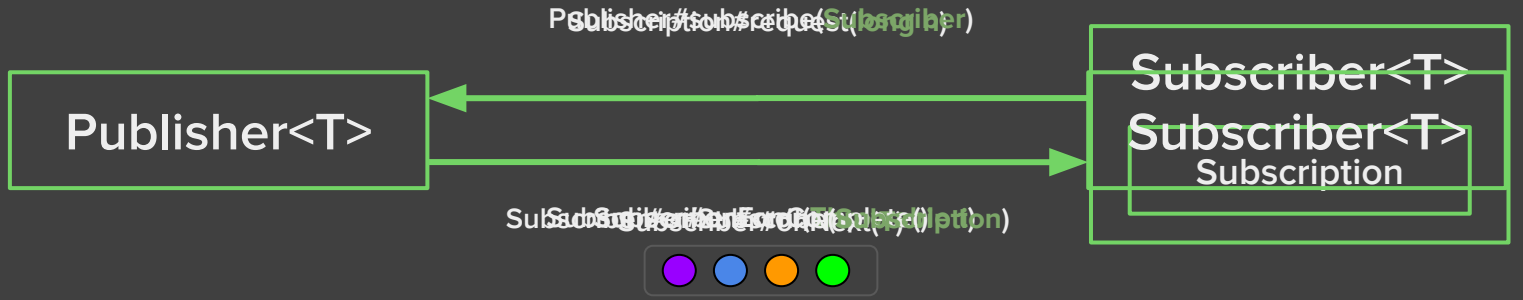
Mono<T>: 0|1 Élément



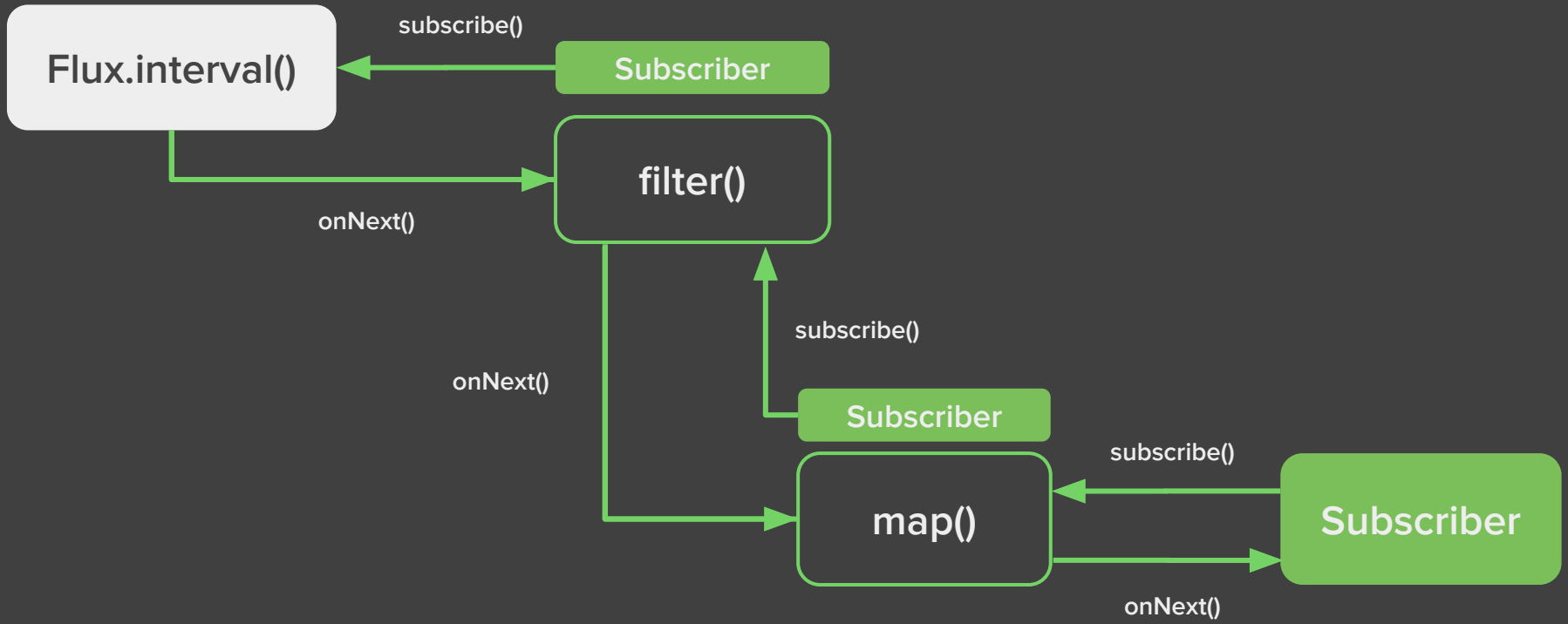
Flux<T> : 0|N Élément(s)



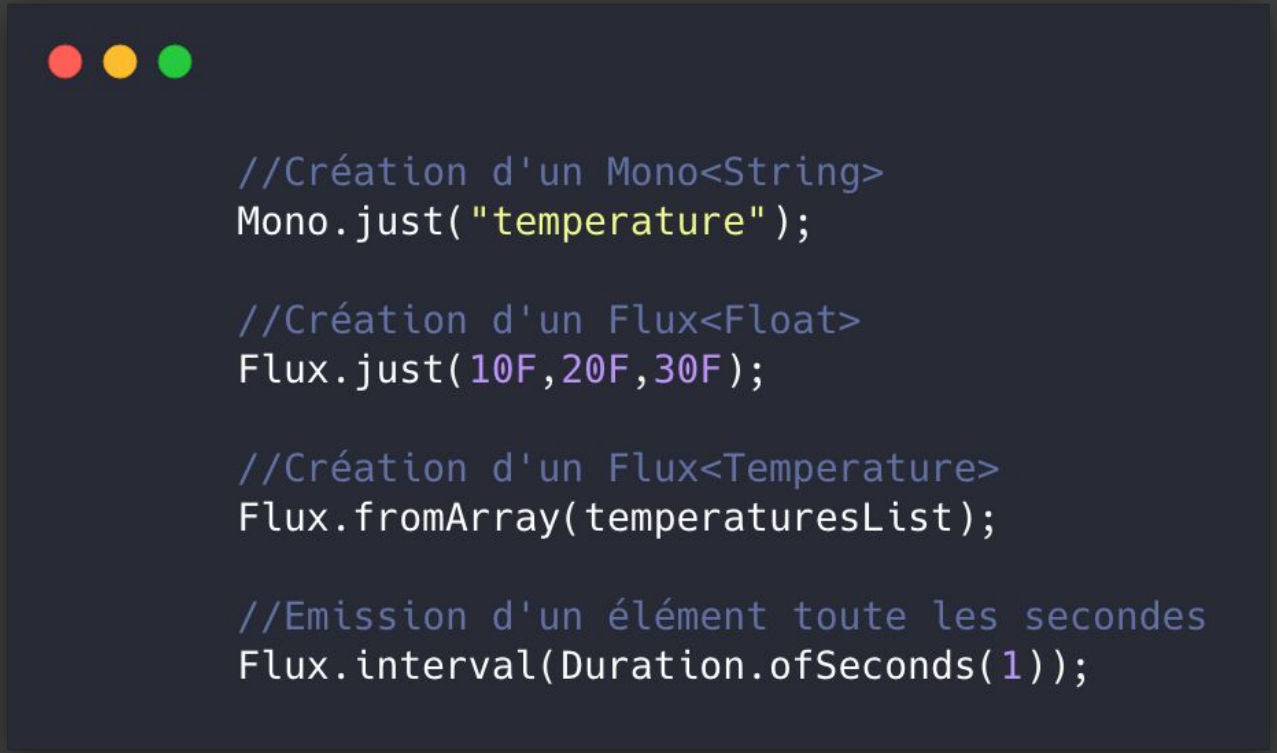
Mécanisme de souscription



Chaîne de souscription



Méthodes factory



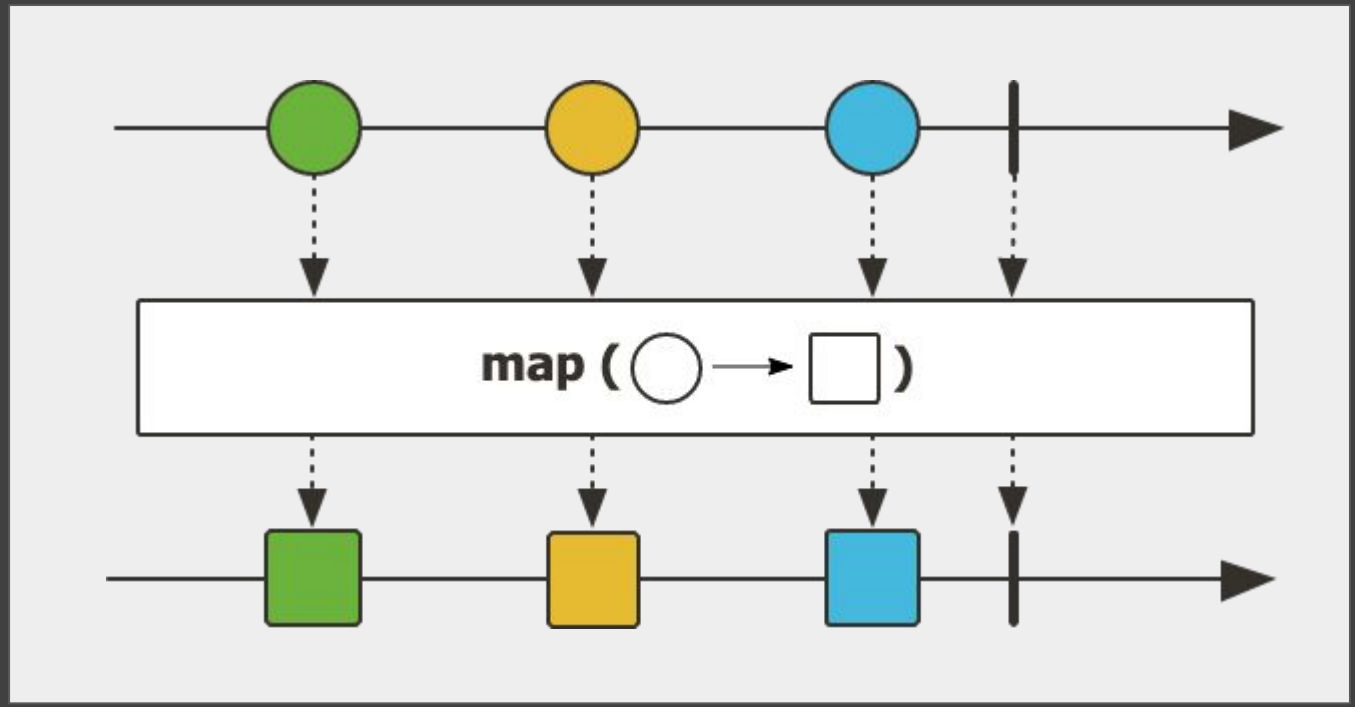
```
//Création d'un Mono<String>
Mono.just("temperature");

//Création d'un Flux<Float>
Flux.just(10F,20F,30F);

//Création d'un Flux<Temperature>
Flux.fromArray(temperaturesList);

//Emission d'un élément toute les secondes
Flux.interval(Duration.ofSeconds(1));
```

.map(a -> a)



StepVerifier

- Permet l’écriture de tests unitaires

- Vérifie le comportement de notre “chaîne réactive” lors d’une souscription.
  - J’envoie bien X éléments ?
  - Mes éléments vérifient bien cette condition (Assertion JUnit / AssertJ / ...)
  - J’ai bien l’événement “cancel” ? “next” ? “complete” ?

- S’intégre avec vos autres frameworks de test (Mockito / AssertJ / ...)

Repository GIT

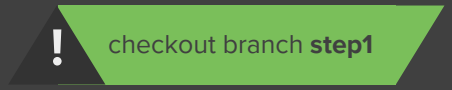
- Cloner le projet

  https://github.com/Zenika/nc-spring-reactor

- Checkout de la branche **step1**

- “mvn clean install”

Step 1 / Exercice



Consigne dans le README

Step 2

Spring WebFlux

Spring WebFlux

- Inclus dans **Spring 5**
- **Reactor 3** est une dépendance de **Spring Webflux**
- A**nnotations** @RestController, @GetMapping et aussi **RouterFunction**
- **WebClient** pour les appels réactifs (ou bloquants),
  Spring prévoit de déprécier le **RestTemplate**
- Intégré dans **Spring Security**

Contrôleur

@Controller

- Idem Spring MVC
- Concis
- Simple

RouterFunction

- Déclaration fonctionnelle des routes
- Permet de paramétrer le comportement de nos endpoints

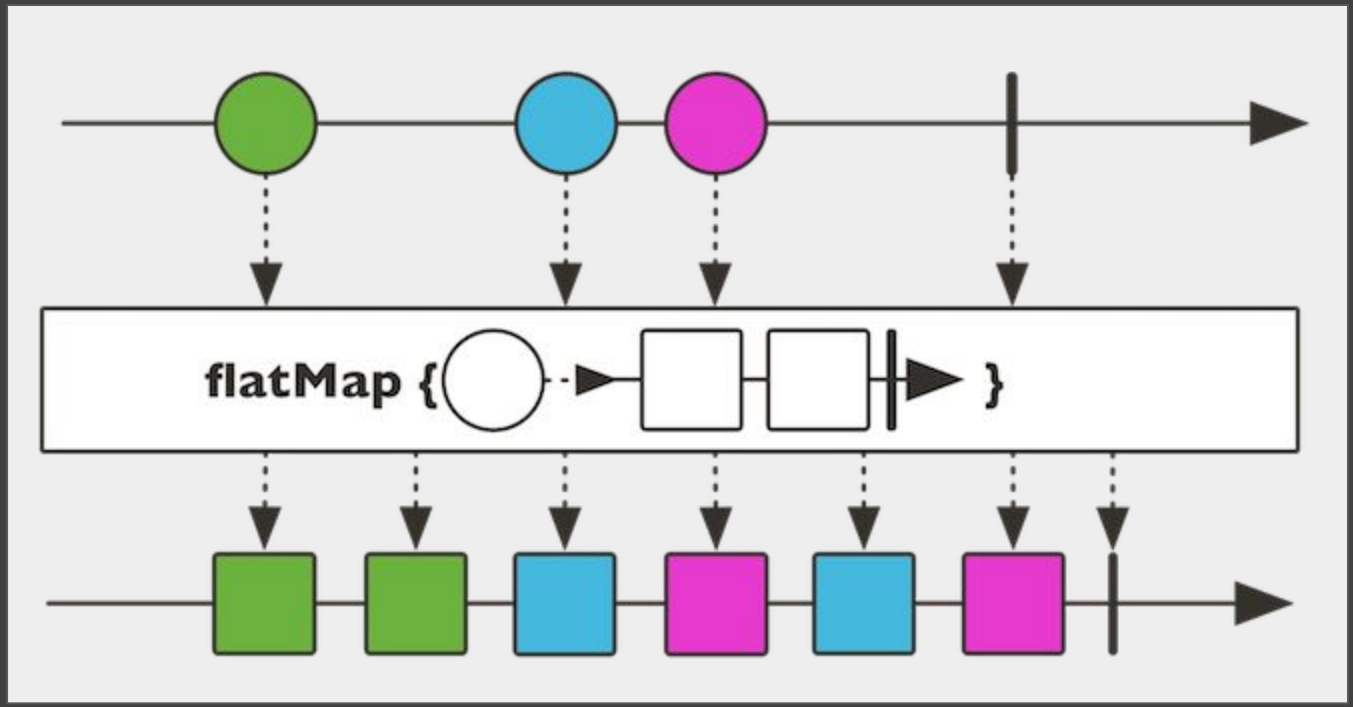
```java
@Configuration
@EnableWebFlux
public class ... implements WebFluxConfigurer {

    @Autowired
    CustomHandler customHandler;

    @Bean
    public RouterFunction<?> routerFunctionTemperature() {
        return route().path("/zenika", builder ->
                builder.nest(//
                        accept(MediaType.ALL), builder2 -> //
                                builder2//
                                        .GET("/hello", request -> customHandler.maMethode())
                                        .GET("/world", request -> customHandler...())
                                        .build()
                ))
                .before(request -> logRequest(request))
                .build();
    }

}
```

flatMap()



Exercice

! checkout branch **step2**

Consigne dans le README

Step 3

Spring WebClient

Step 3 / Concepts

- **WebClient** remplace le **RestTemplate**
- Simple d'utilisation via une API fonctionnelle
- Permet de faire des appels **Réactifs** : retourne un **Mono** ou un **Flux**
- Permet de faire des appels bloquants : retourne un objet ou une liste
- Se connecte à **Reactor Netty** par défaut, mais peut se connecter à d'autres librairies (Jetty Reactive, httpClient, ...)

Step 3 / Exercice

! checkout branch **step3**

Consigne dans le README

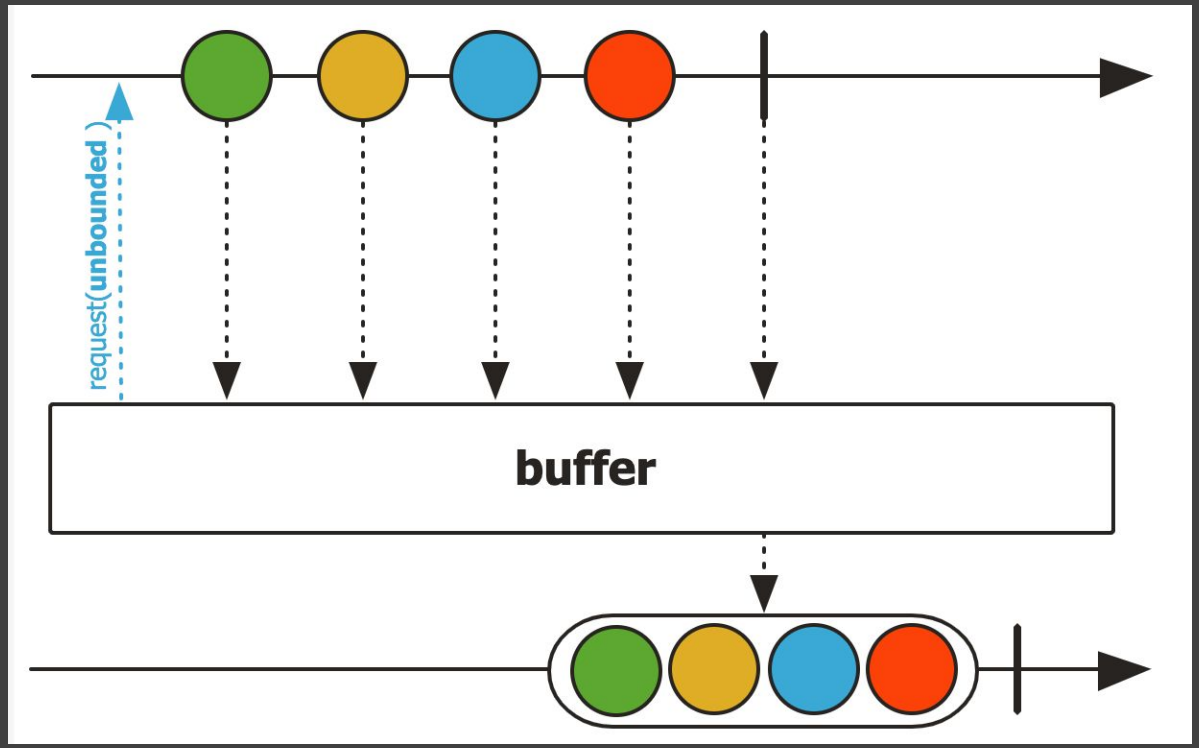
Step 4

Spring Data

Step 4 / Concepts

- Version **réactive** du framework **Spring Data**
  - Mongodb
  - Redis
  - Couchbase
  - Cassandra
- ~~CrudRepository~~ ReactiveCrudRepository
- ~~MongoRepository~~ ReactiveMongoRepository
- ~~**Optional<T>**~~, ~~List<T>~~ **Mono<T>**, **Flux<T>**
- **R2DBC** pour MySQL, Postgres & H2

buffer()



Step 4 / Exercice

! checkout branch **step4**

Consigne dans le README

Step 5

Reactor Processors

Step 5 / Concepts

- **Processor**

- A la fois un **Publisher** & un **Subscriber**
  - Souscription à un **Processor**
  - Injection de donnée à partir de différents threads

- Différents types
  - Direct
  - Synchrone
  - Asynchrone

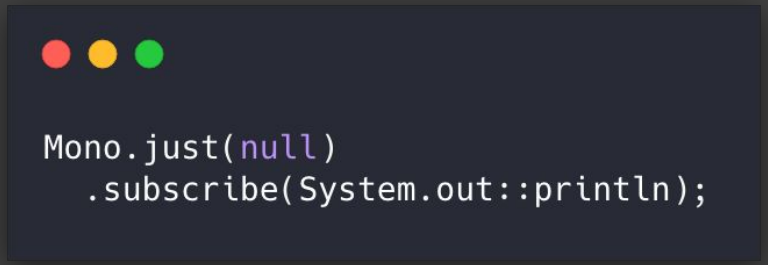
Step 5 / Exercice

! checkout branch **step5**

Consigne dans le README

Step 6 / ~~Null~~

- Reactor n’autorise pas d’éléments **null**
- Si l’on exécute ce code



```
Mono.just(null)
  .subscribe(System.out::println);
```

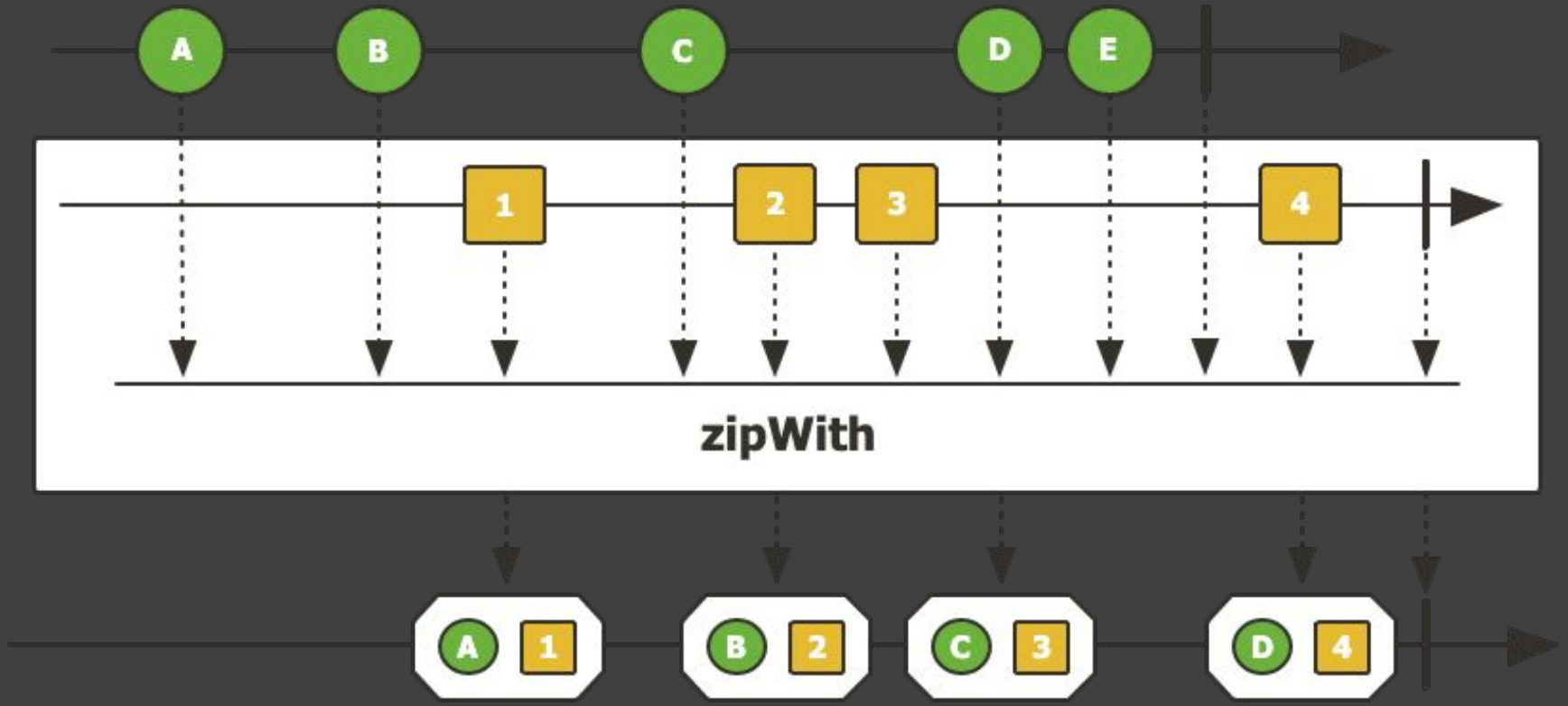
  - La méthode *Subscriber#onError(Throwable t)* sera invoquée par le **publisher**
  - Un **NullPointerException** sera levée
  - Plus aucun élément ne sera reçu par la souscription en cours

Step 6 / Gestion des erreurs

Reactor expose des opérateurs dédiés au traitement de la propagation d'Exceptions dans la pile d'exécution en modifiant la séquence.

- onErrorReturn() : retourne un nouveau élément
- onErrorResume() : substitue la séquence par une autre
- onErrorMap() : permet de renvoyer une exception
  - Plus simple qu’un onErrorResume(Flux.error(new MyException()))
- onErrorContinue() : fait en sorte de ne pas interrompre un flux

Step 6 / Zip



Step 6 / Exercice

! checkout branch **step6**

Consigne dans le README

Ce que vous devez encore apprendre

Fuseable Operators

switchMap()

Back Pressure

retry()

Context

zip()

groupBy()

switchIfEmpty()

reduce()

compose()

window()

Merci !