



라이브 세션-2022.10.13(목)-Reactor

✓ 리액티브 시스템과 리액티브 프로그래밍

1 리액티브 시스템이란?

- 리액션이 좋은 사람은 어떤 사람인가요?
- 그렇다면 리액션이 좋은 시스템은?
 - 클라이언트의 요청에 반응을 잘하는 시스템
 - 반응에 느린 시스템의 예를 볼까요?

2 리액티브 프로그래밍이란?

- 리액티브 시스템에서 사용할 수 있는 프로그래밍 모델
- 선언형 프로그래밍 방식을 사용
- 함수형 프로그래밍 방식을 이용
- 명령형 프로그래밍 방식과 선언형 프로그래밍 방식의 차이
 - 예제 코드로 리뷰
- 리액티브 프로그래밍의 예제 코드 리뷰(Hello Reactive)

3 리액티브 스트림즈(Reactive Streams)란?

- 리액티브 프로그래밍 구현체를 위한 스펙(사양, Specification)
- <https://github.com/reactive-streams/reactive-streams-jvm>
- 리액티브 스트림즈 구현체 종류

- Reactor
- RxJava
- Akka Streams
- Java Flow API
- 기타 언어별 Rx 라이브러리
 - RxJS
 - RxAndroid
 - RxPython
 - RxKotlin
 - ...

4 리액티브 프로그래밍에서 사용되는 용어 살펴보기

- 코드로 확인합시다
-

Project Reactor


1 Project Reactor란?

- 리액티브 프로그래밍을 위한 리액티브 라이브러리
- 리액티브 스트림즈(Reactive Streams)의 구현체
- Spring WebFlux 프레임워크의 핵심 중에 핵심

2 Reactor의 특징


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.

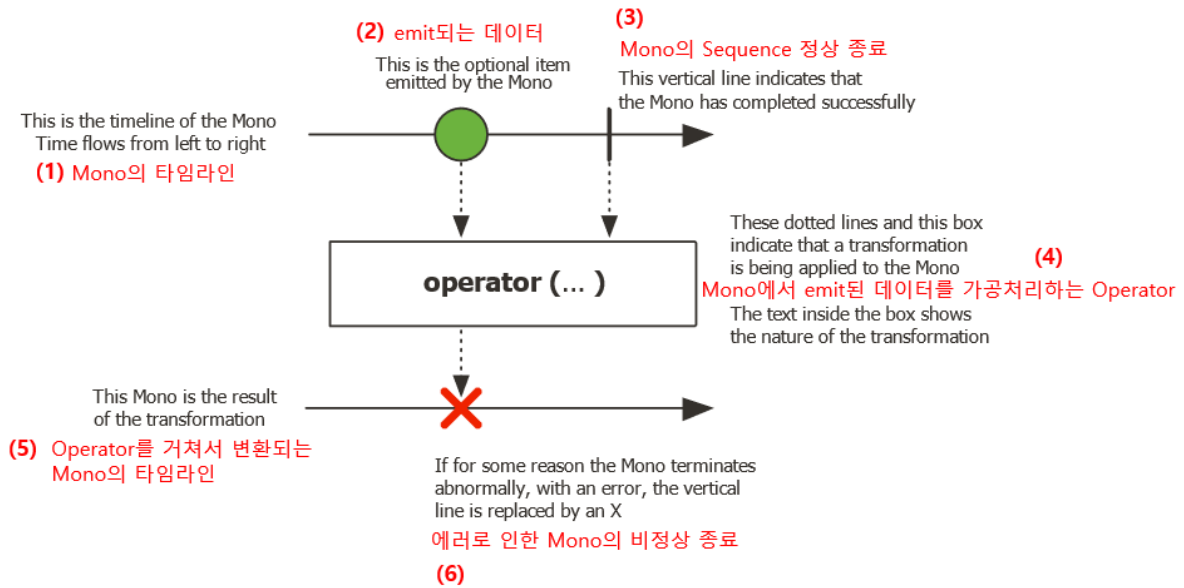
3 Hello, Reactor 코드로 보는 Reactor 구성 요소 살펴보기

- 코드로 리뷰

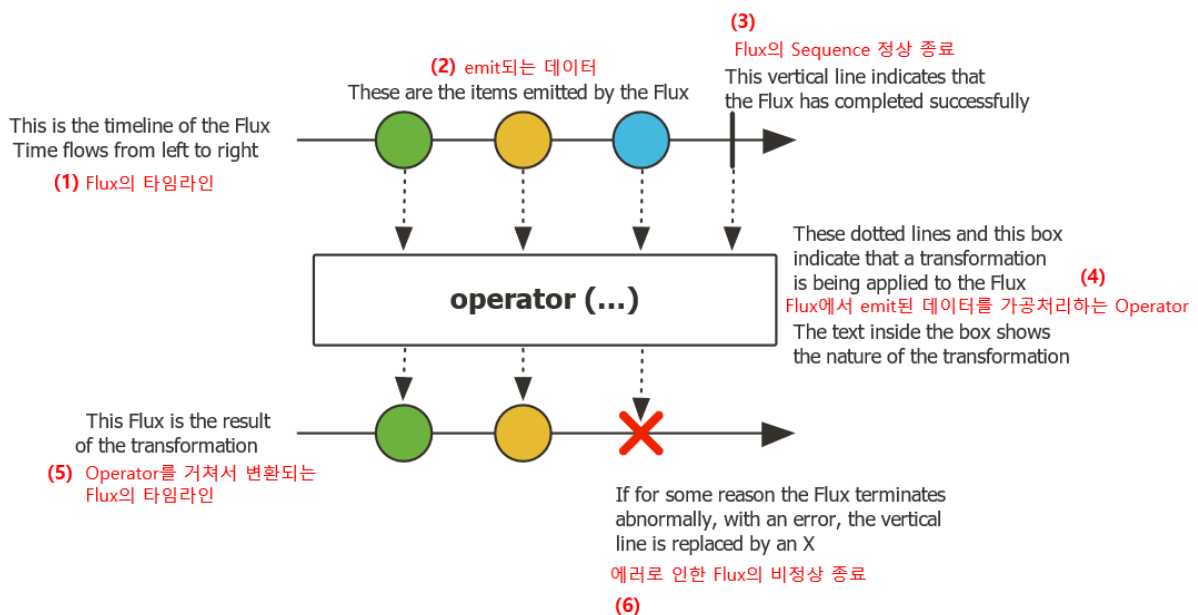
4 마블 다이어그램

- 마블 다이어그램 보는 방법
- 마블 다이어그램으로 Mono, Flux 이해하기

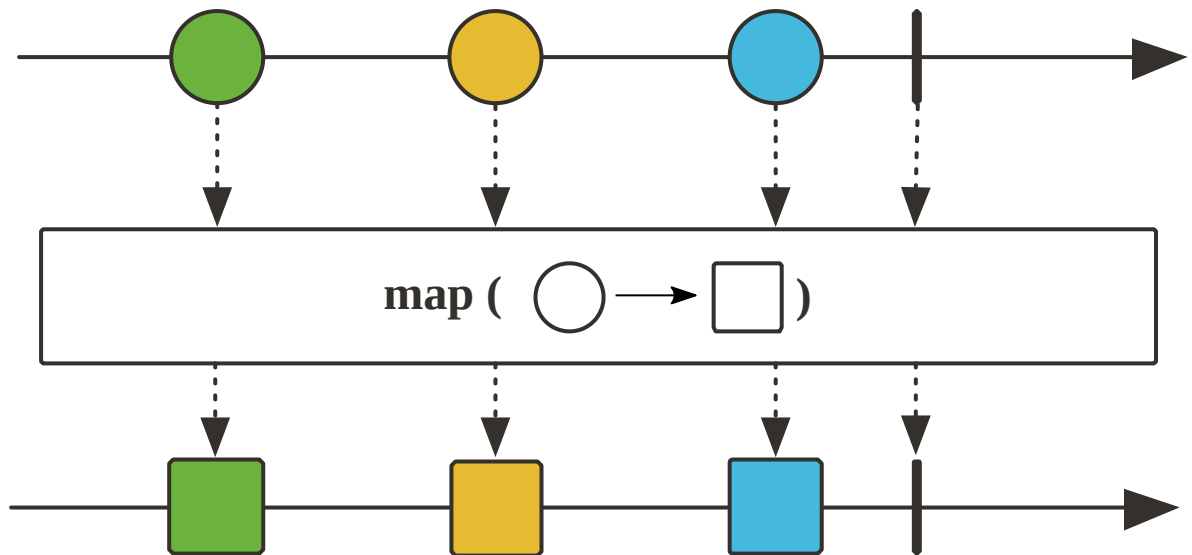
✓ Mono의 마블 다이어그램



✓ Flux의 마블 다이어그램



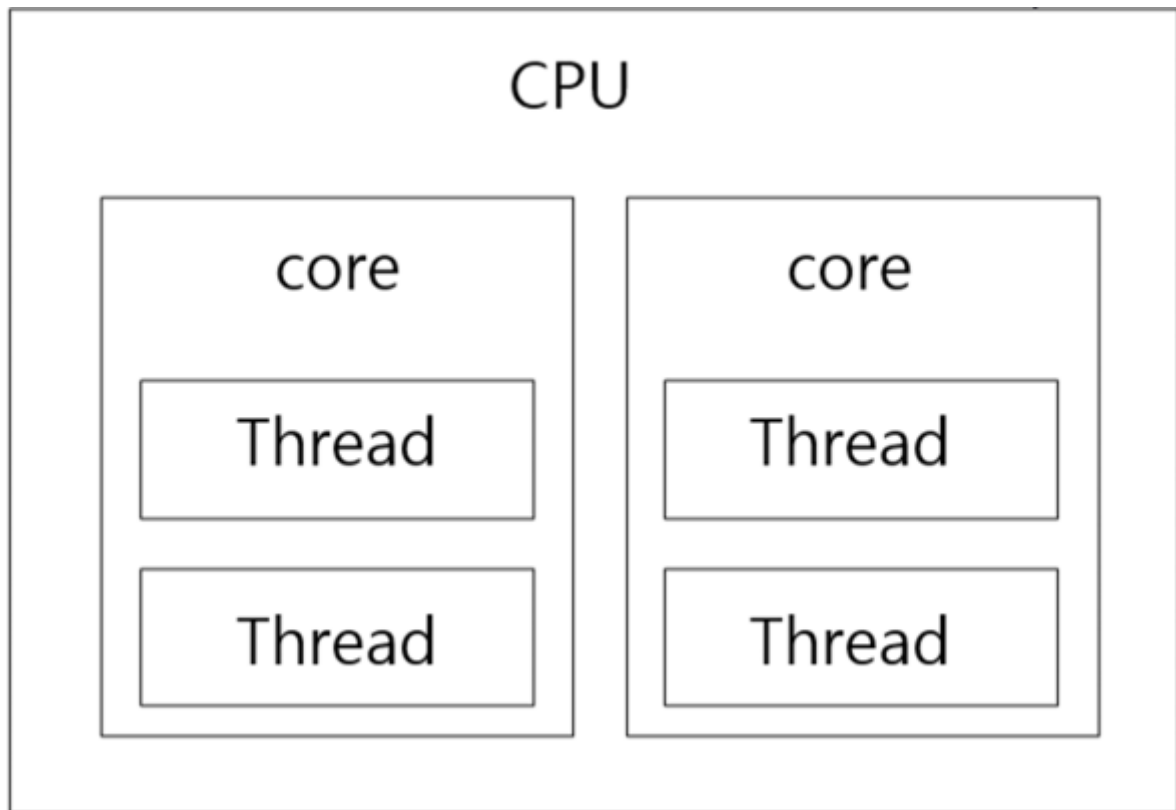
✓ Operator 마블 다이어그램 예



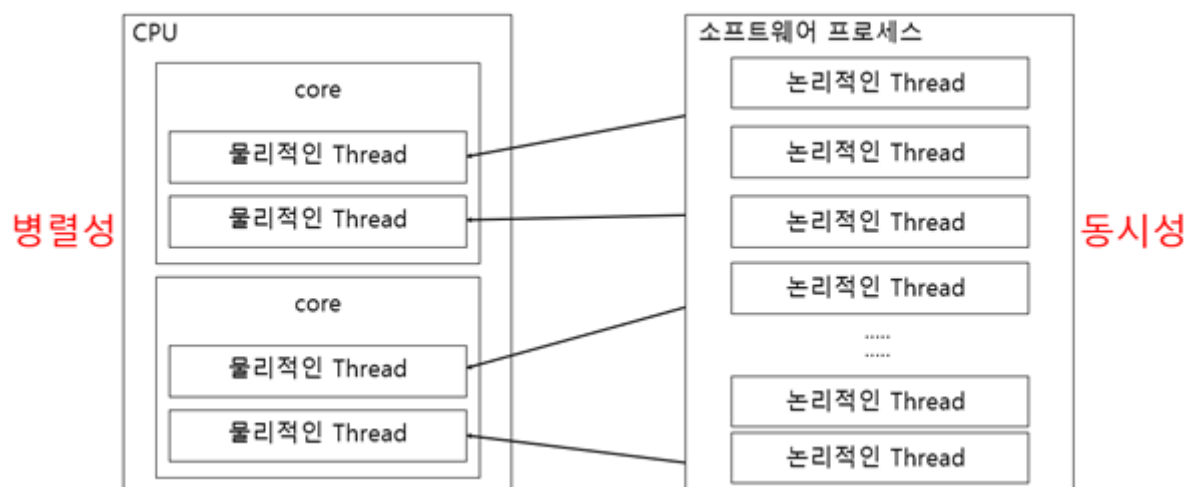
5 스케줄러(Scheduler)란?

- Scheduler는 쓰레드를 관리하는 관리자의 역할
- 즉, 복잡한 멀티쓰레딩 프로세스를 단순하게 해주면서 쓰레드 간에 발생할 수 있는 문제점 등을 개발자가 신경쓰지 않아도 된다.
- ★ Scheduler를 이해하기 위해서는 쓰레드를 먼저 이해해야 한다.
 - 쓰레드를 간단하게 이해해 봅시다.

✓ 코어와 물리적인 쓰레드의 관계



✓ 물리적인 쓰레드와 논리적인 쓰레드의 관계



- 내 컴퓨터의 코어 및 물리적인 Thread 확인 해보기
- **Java의 쓰레드 예제 코드 확인**

★ **publishOn()**

- 전달 받은 데이터를 가공 처리하는 Operator 앞에 추가해서 실행 쓰레드를 별도로 추가할 수 있다.

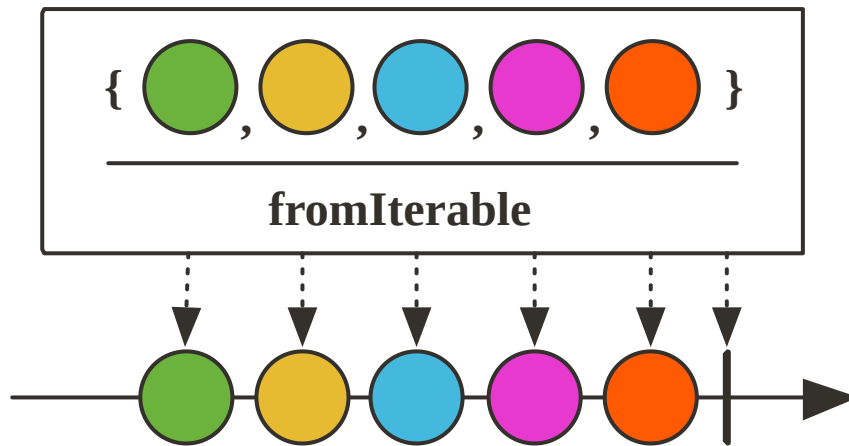
★ **subscribeOn()**

- 데이터 소스에서 데이터를 emit하는 원본 Publisher의 실행 쓰레드를 지정한다.

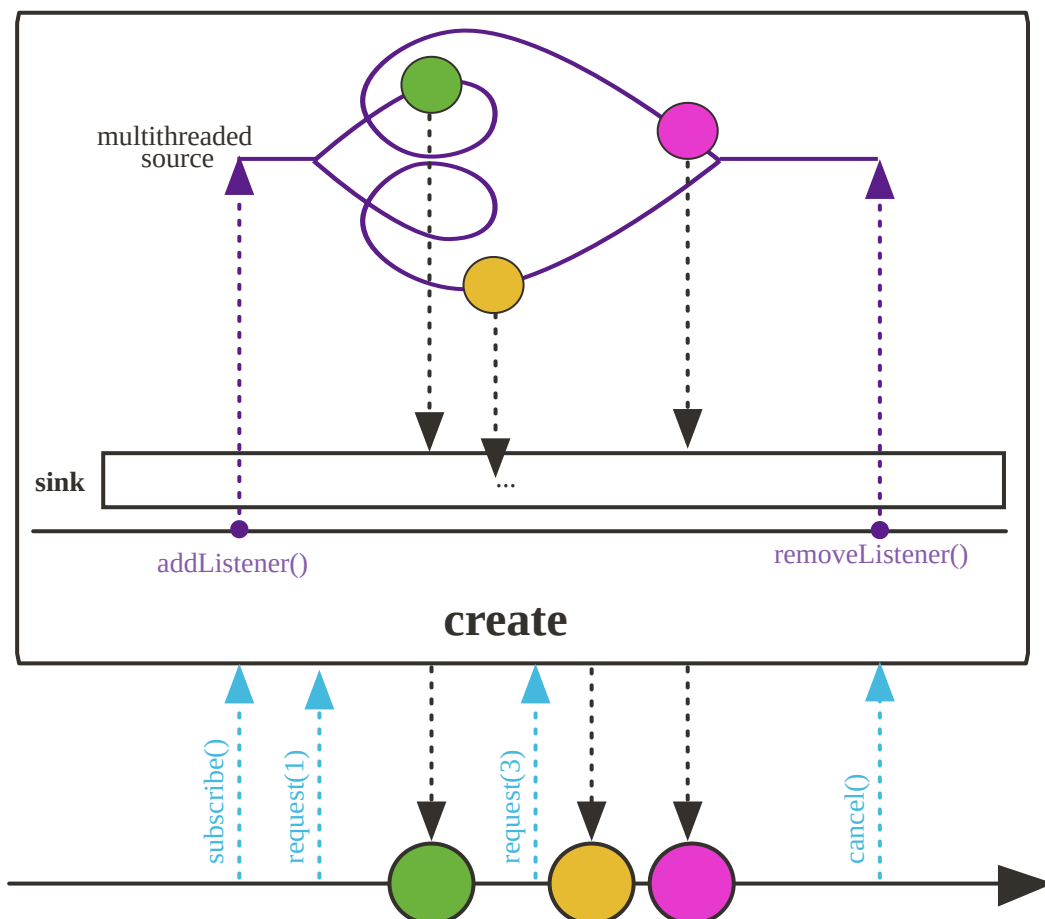
6 Operators

- Reactor API 문서에서 Operator 확인해보기
 - <https://projectreactor.io/docs/core/release/api/>
- 상황별로 분류된 Operator 목록 확인해보기
 - <https://projectreactor.io/docs/core/release/reference/#which-operator>
- 자주 사용되는 Operator
 - **Sequence 생성**
 - `fromIterable()`
 - `fromStream()`
 - `create()`

✓ **fromIterable()**



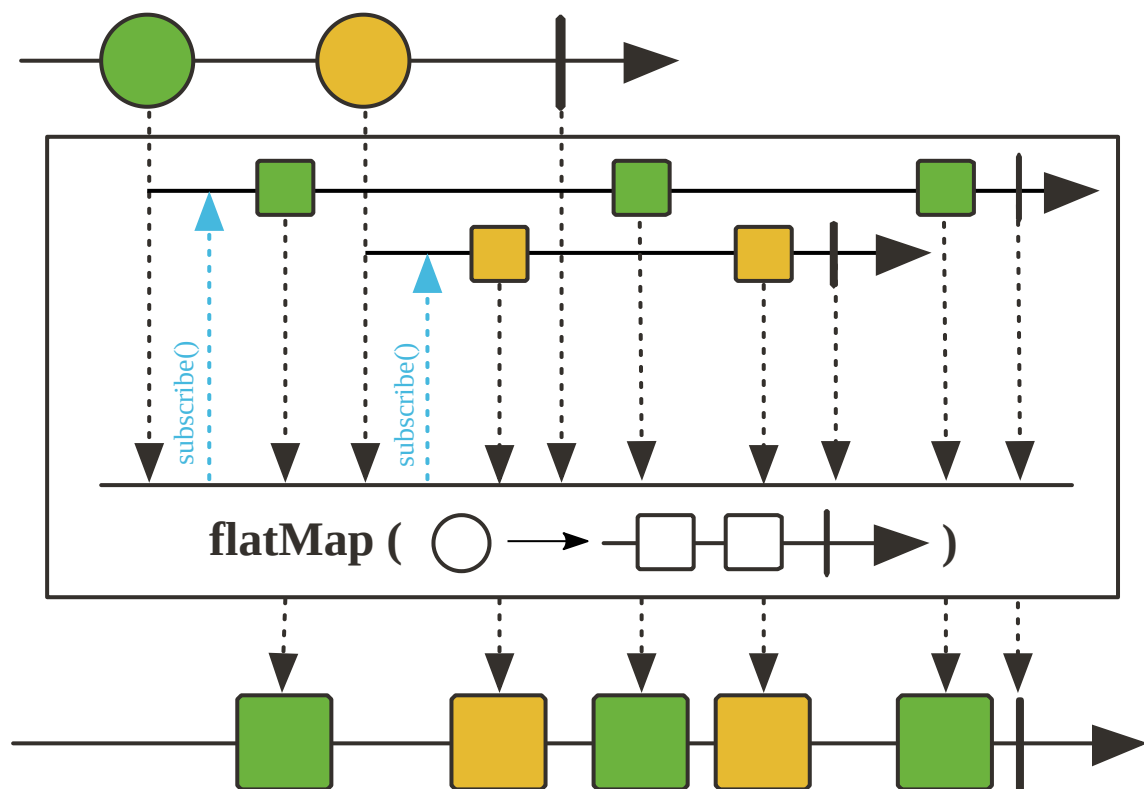
✓ `create()`



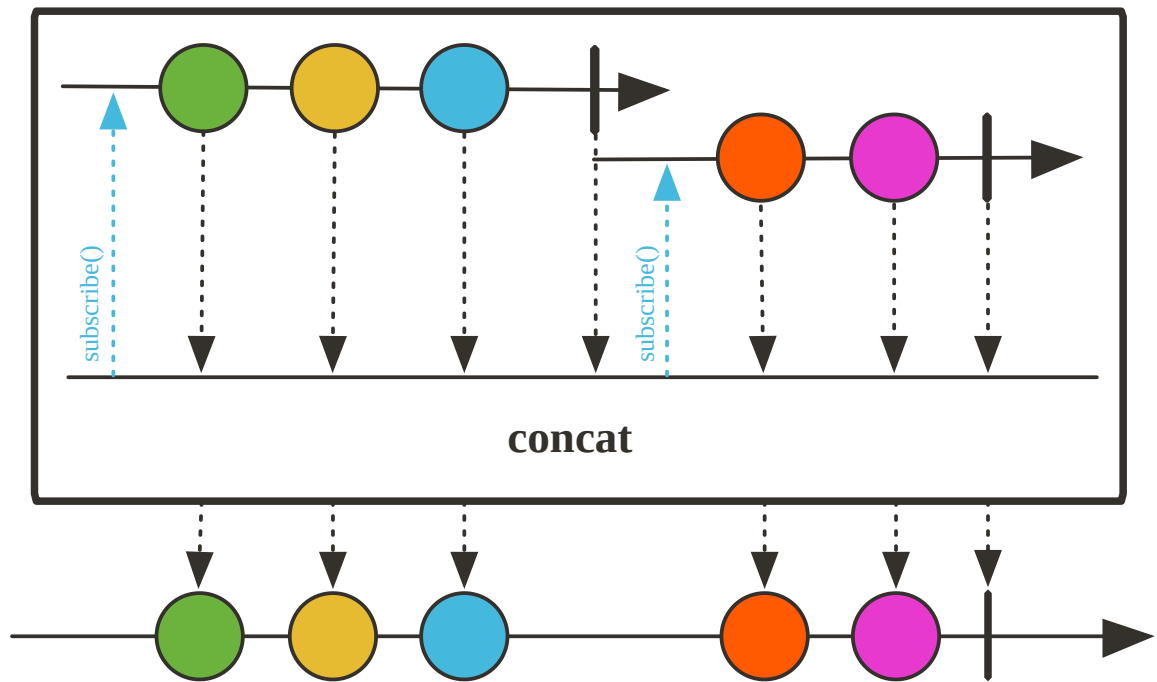
◦ Sequence에서의 데이터 변환

- `map()`
- `flatMap()`
- `concat()`
- `zip()`

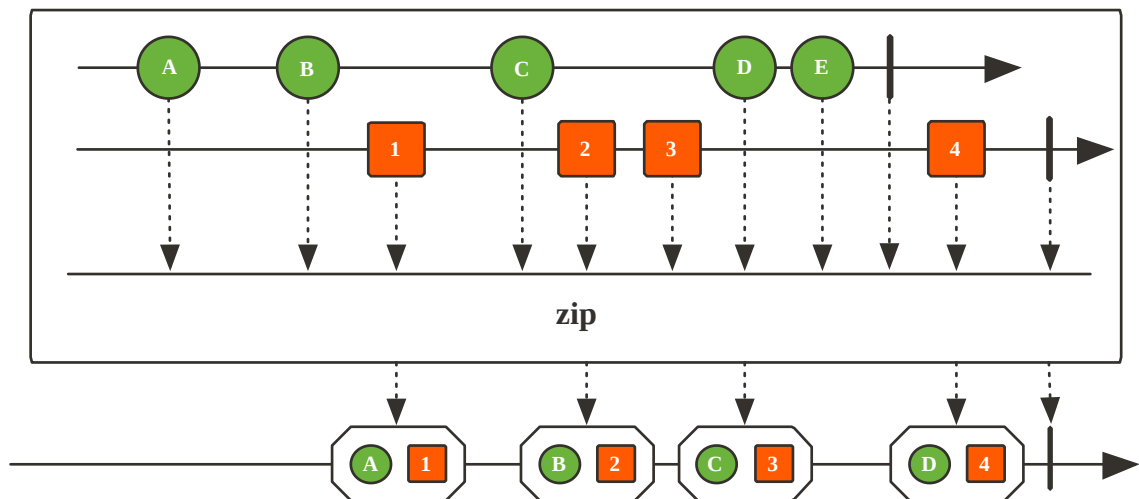
✓ `flatMap()`



✓ `concat()`



✓ `zip()`



- Sequence에서의 데이터 필터링

- `filter()`
- `take()`
- `skip()`

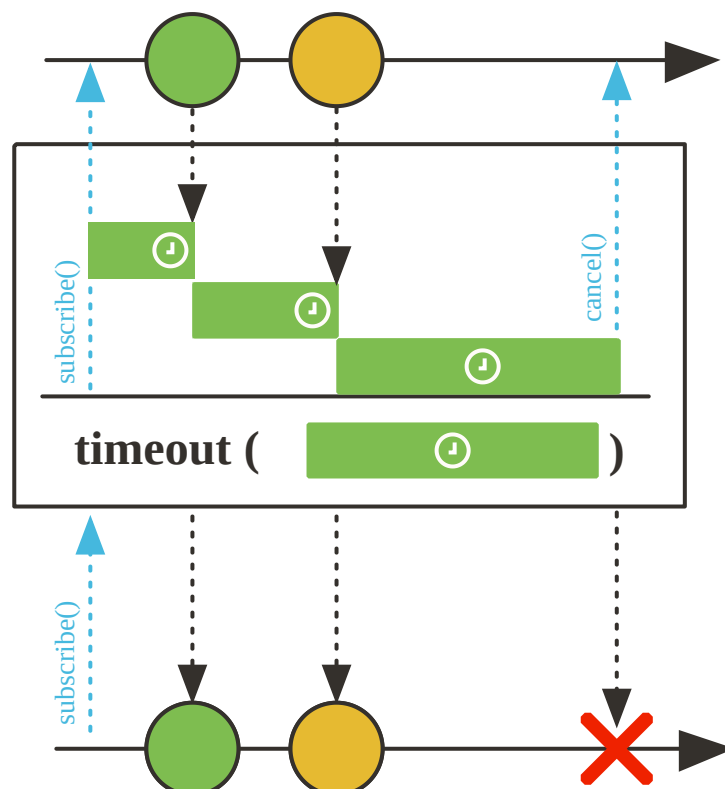
- 에러 핸들링

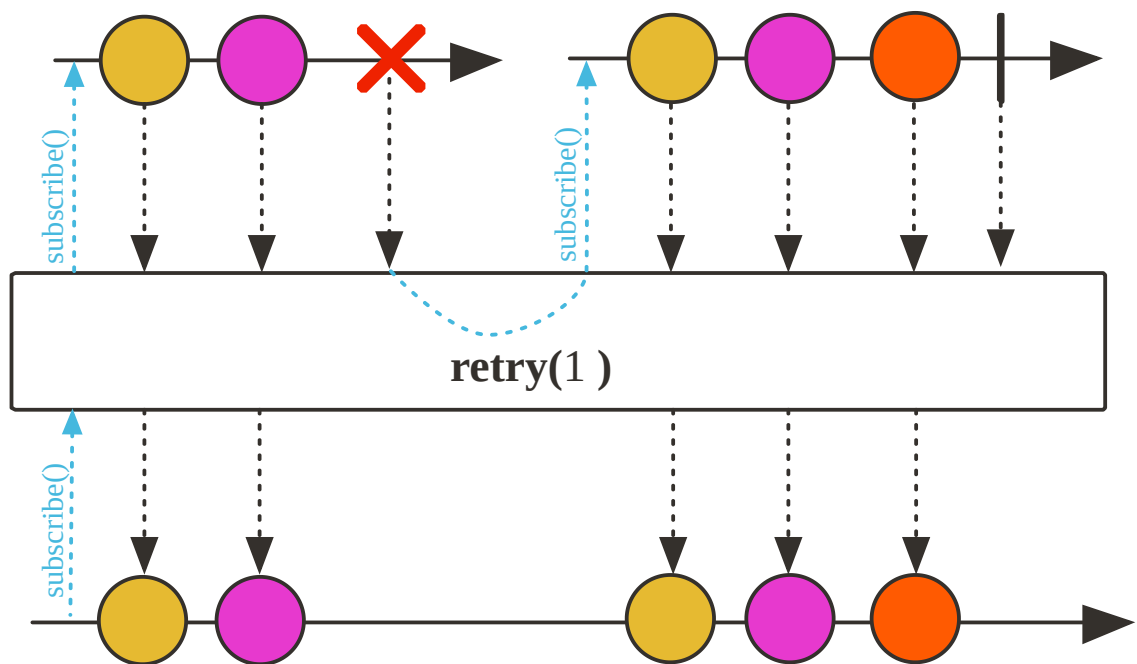
- `error()`
- `timeout()`
- `retry()`

- ✓ `error()`

- 예제 코드로 확인

- ✓ `timeout()` / `retry()`





✔ 리액티브 프로그래밍을 어디에 써 먹을 수 있을까?

- Spring WebFlux 기반의 Non-Blocking 애플리케이션에서 써먹을 수 있다.
- Blocking I/O 애플리케이션에서도 복잡한 데이터 가공 처리를 위해서 써먹을 수 있다.
- MSA(Microservice Architecture)에서 중요한 역할을 한다.