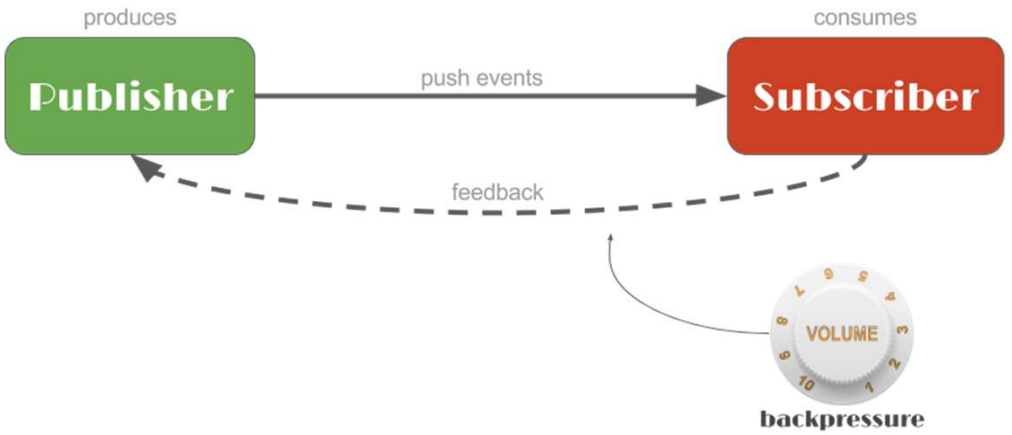
Spring Reactor – Reactive programming

Motivation

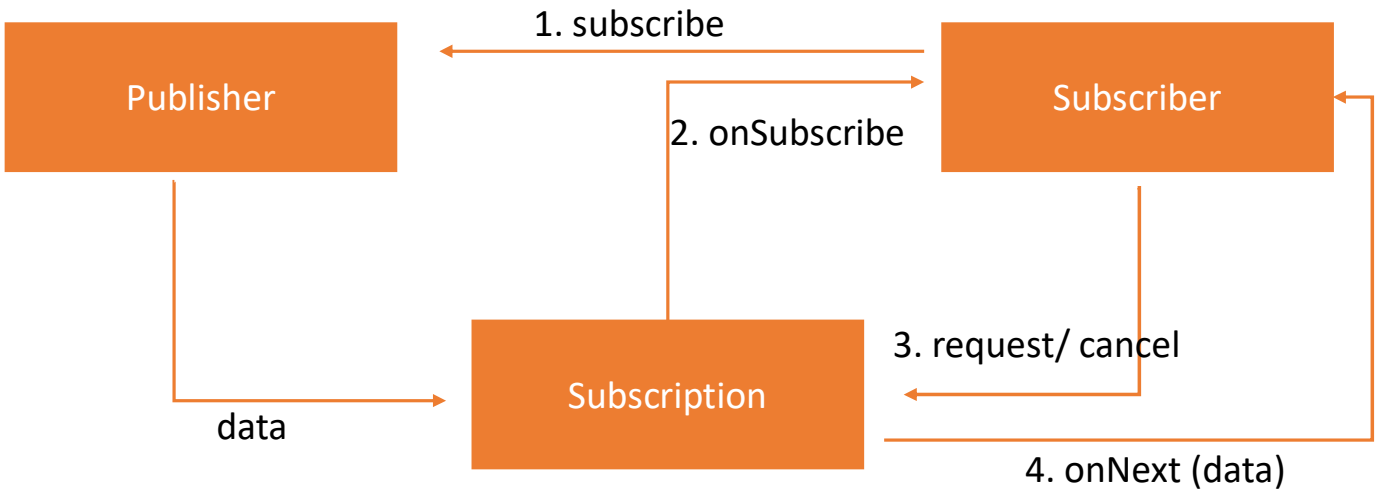
- → Non-blocking
- → Scalability
- → Better resource utilization
- → Functional style of coding
- → Backpressure on data streams
- → Asynchronous

Manifesto

- → Responsive
- → Resilient
- → Elastic
- → Message driven


produces
consumes
Publisher
push events
Subscriber
feedback
VOLUME
backpressure

Flow-control



Reactive programming

- Core Interfaces
  - Publisher<T>
  - Subscriber<T>
  - Subscription
  - Processor

Reactive programming

- → Implementations
  - → Spring Reactor
  - → RxJava
  - → Akka
- → Interoperable

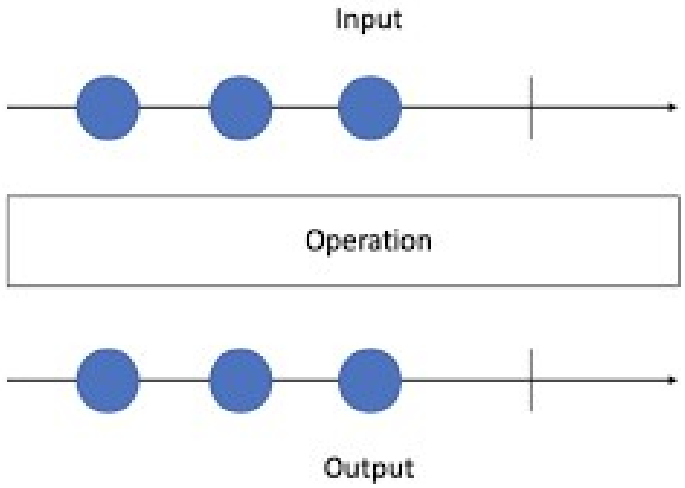
Spring Reactor

- Reactive implementation from Spring framework
- Spring framework depends on this
- First class support from Spring Boot 2.x onwards
- Followed by Spring Security 5, Spring Data JPA

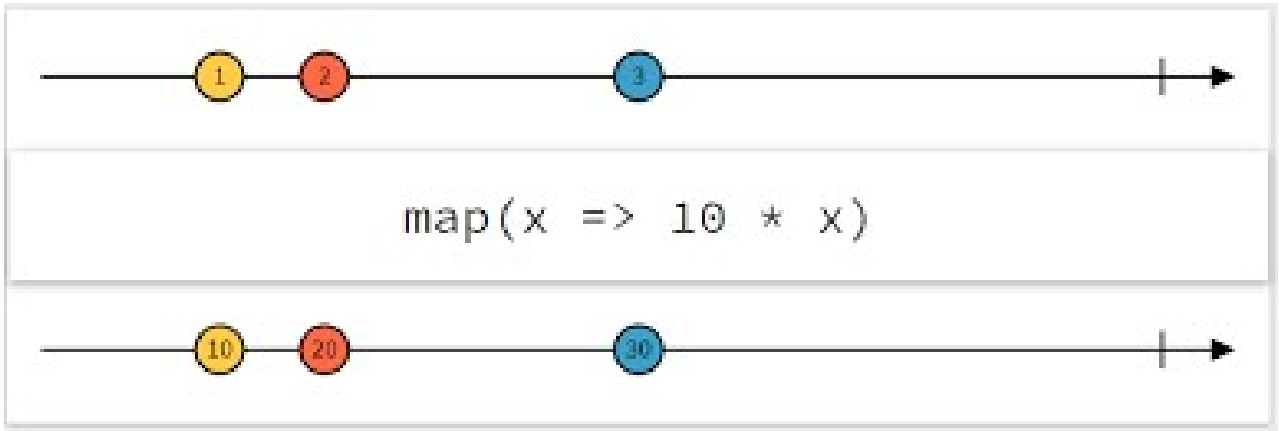
Spring Reactor Vs Streams

- → Streams are for single use. Flux can be subscribed to many times
- → Streams are pull based. Flux is push based
- → Streams work with local datastore like collections
- → Streams are synchronous sequence.
- → Flux -> Completable Future + Streams

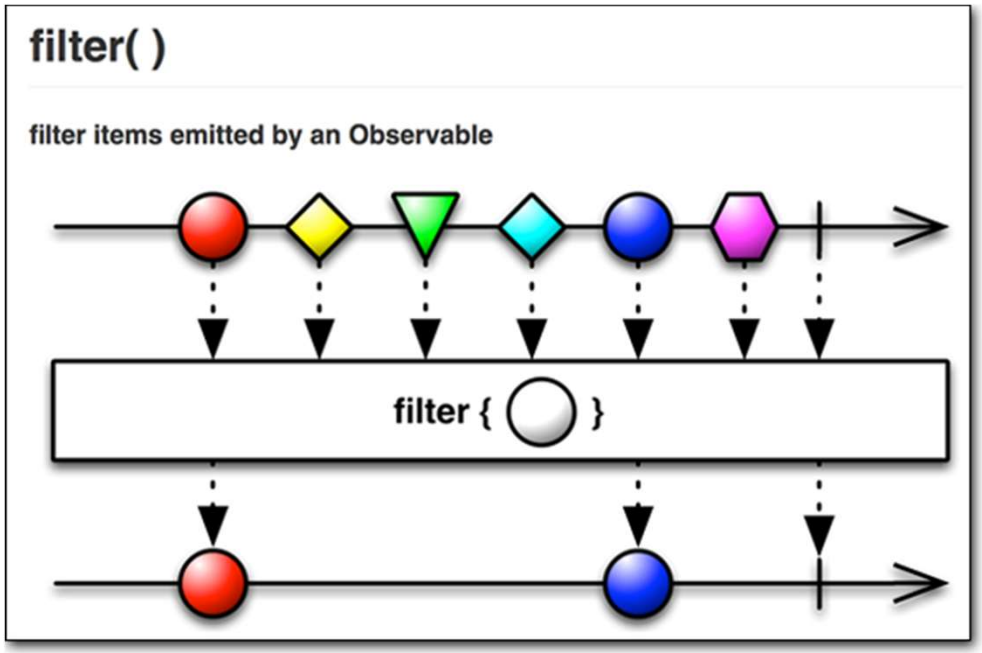
Marble diagrams



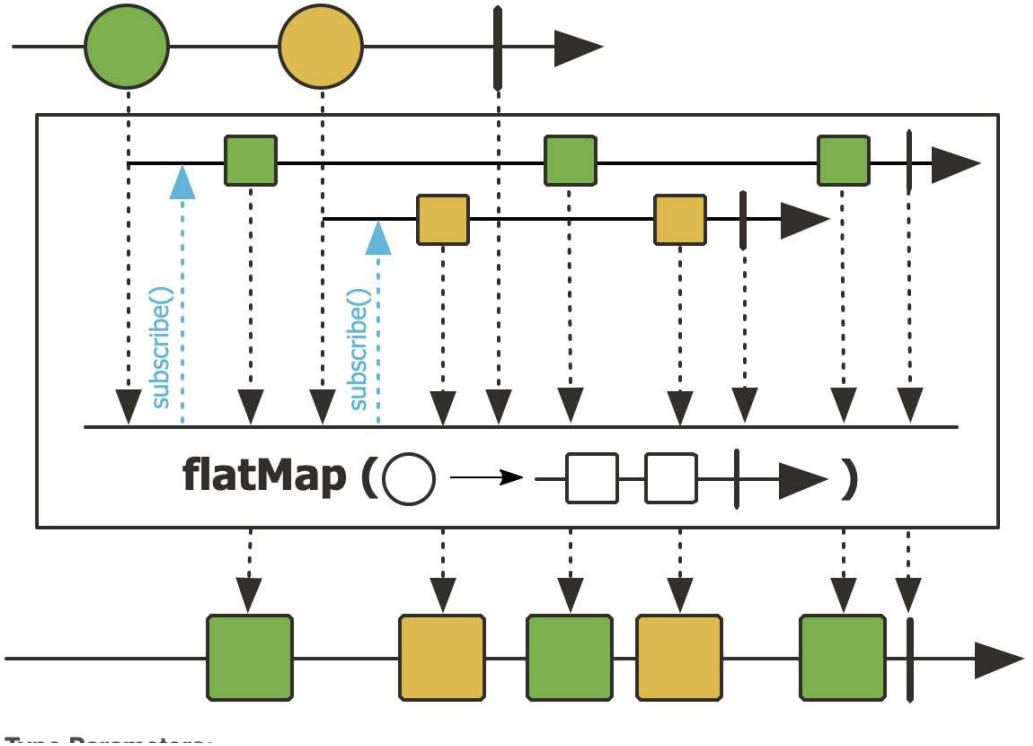
map



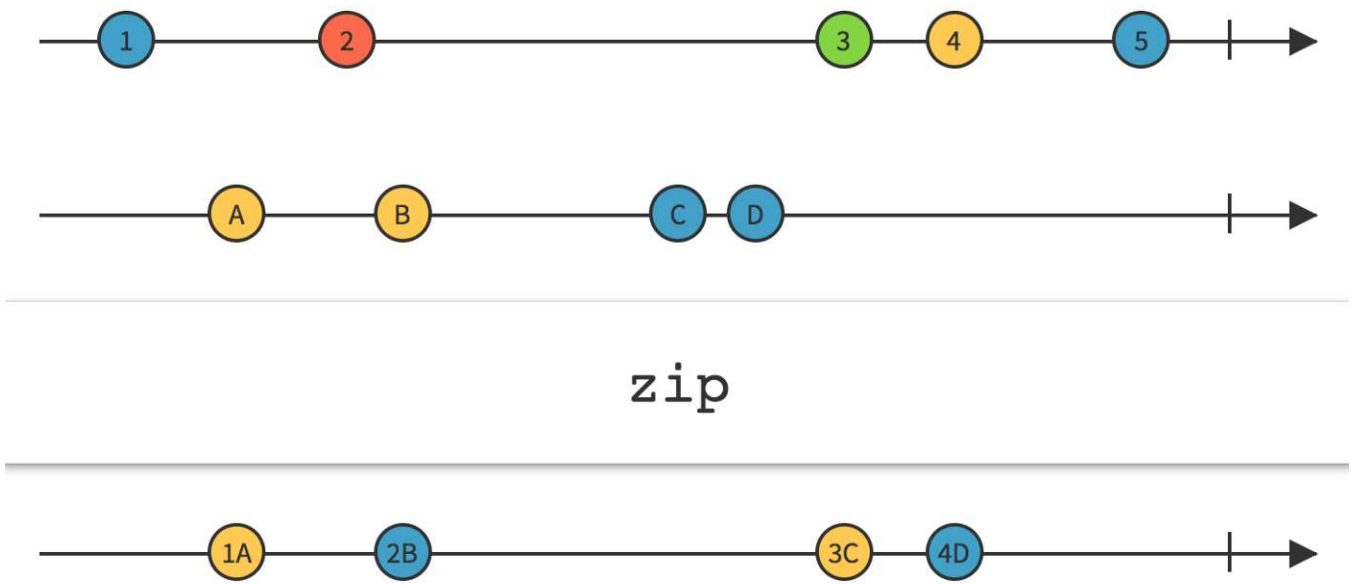
filter



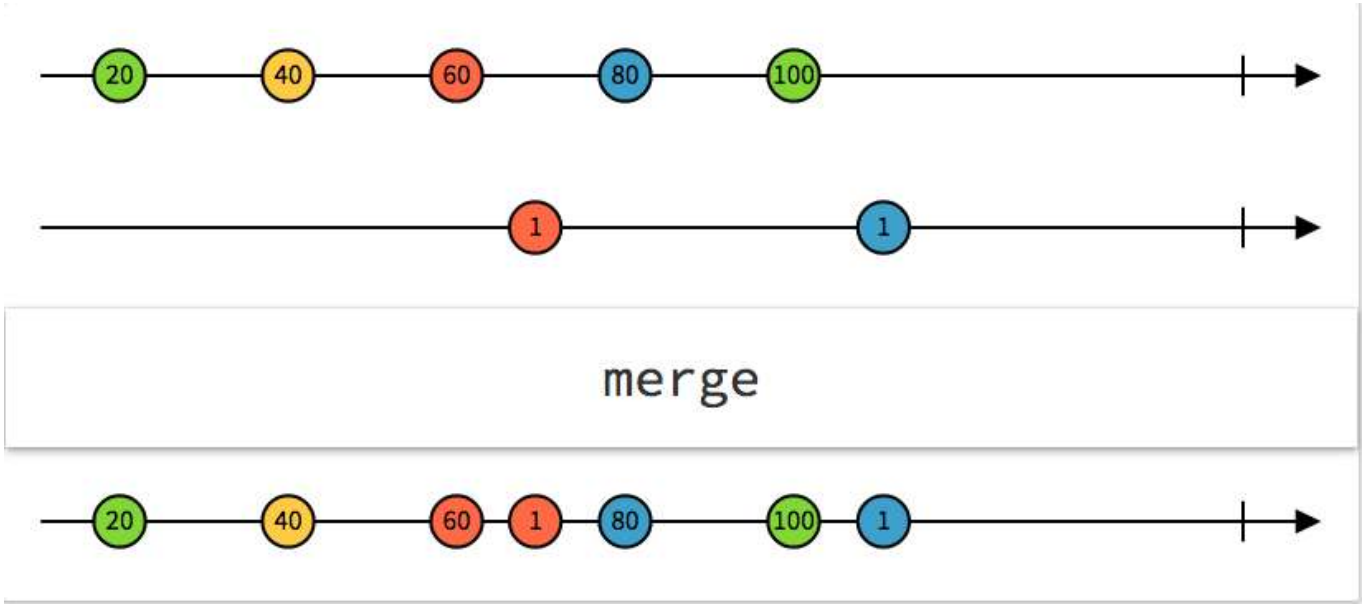
flatMap



zip



merge



switchMap