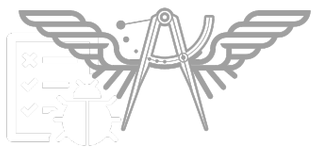
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Reactive Programming
Reactive System



Somkiat Puisungnoen

Somkiat | Home

Somkiat Puisungnoen

Update Info 1 | View Activity Log 10+ | ...

Timeline | About | Friends 3,138 | Photos | More

When did you work at Opendream?

22 Pending Items

Intro

Software Craftsmanship

Software Practitioner at สยามชำนาญกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Post | Photo/Video | Live Video | Life Event

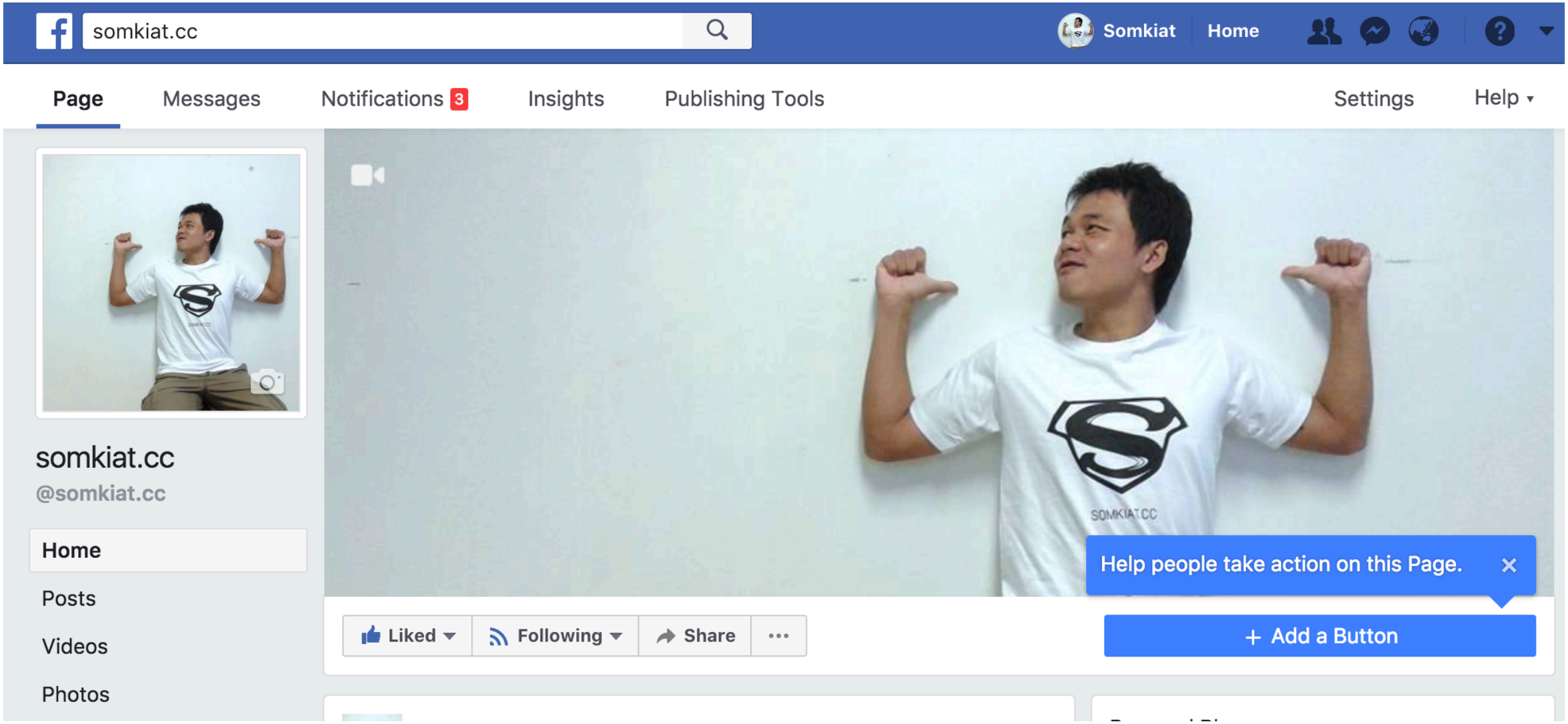
What's on your mind?

Public | Post

Somkiat Puisungnoen

15 mins · Bangkok ·

Java and Bigdata



https://github.com/up1/workshop-reactive-programming

Topics

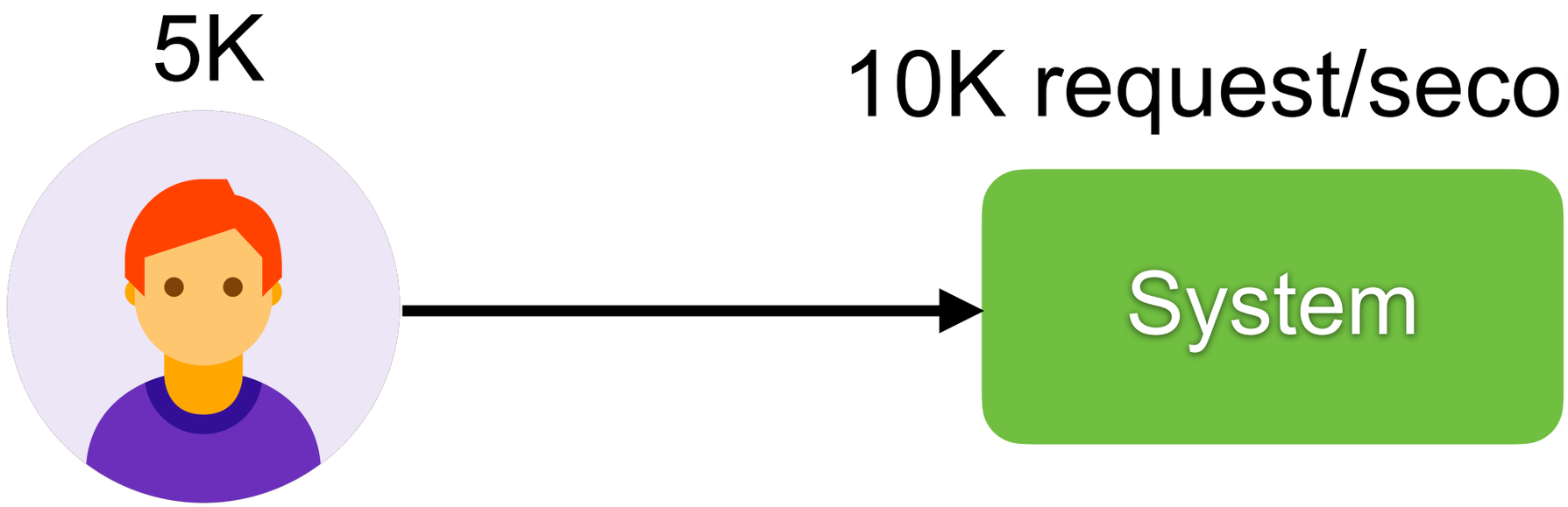
Why and What Reactive ?
Reactive architecture vs Programming
Spring Reactive (Webflux)
Demo
Testing
Q/A

Why and What

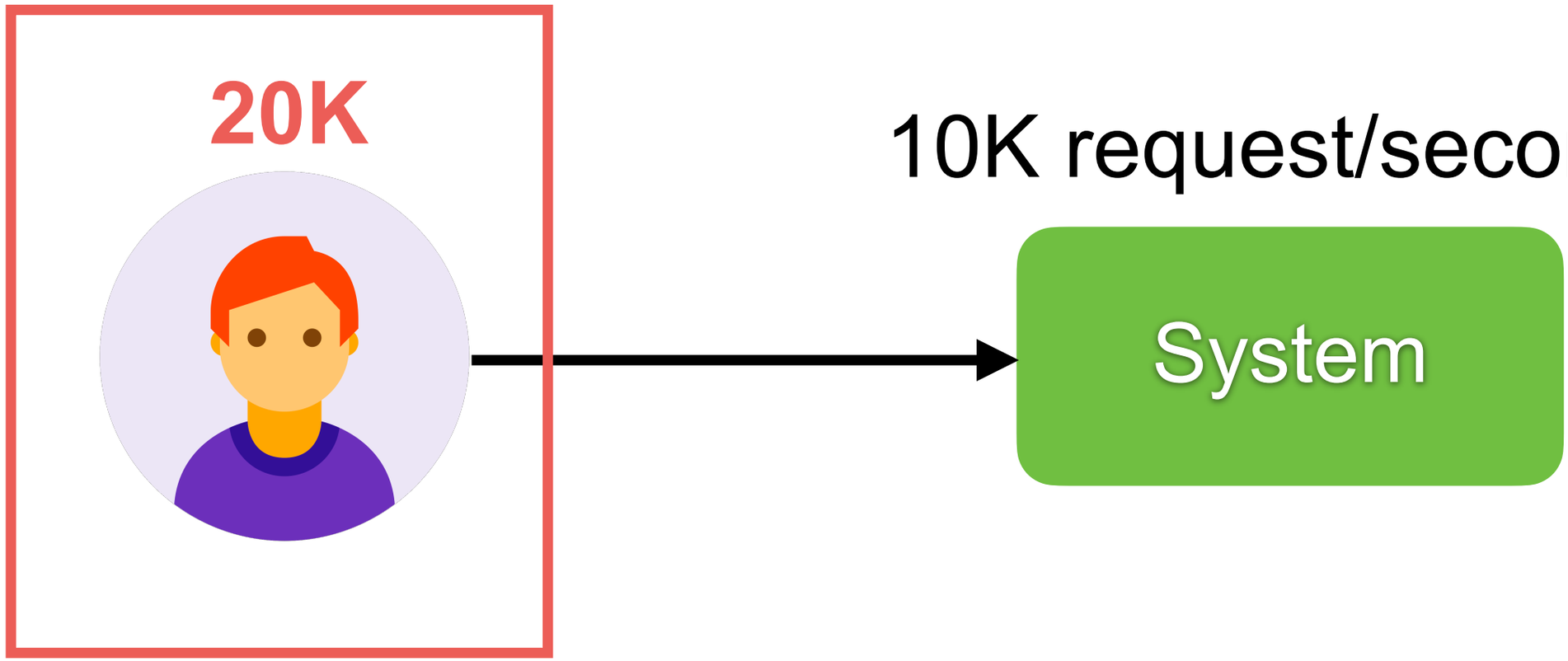
PROACTIVE

REACTIVE

Why Reactive ?

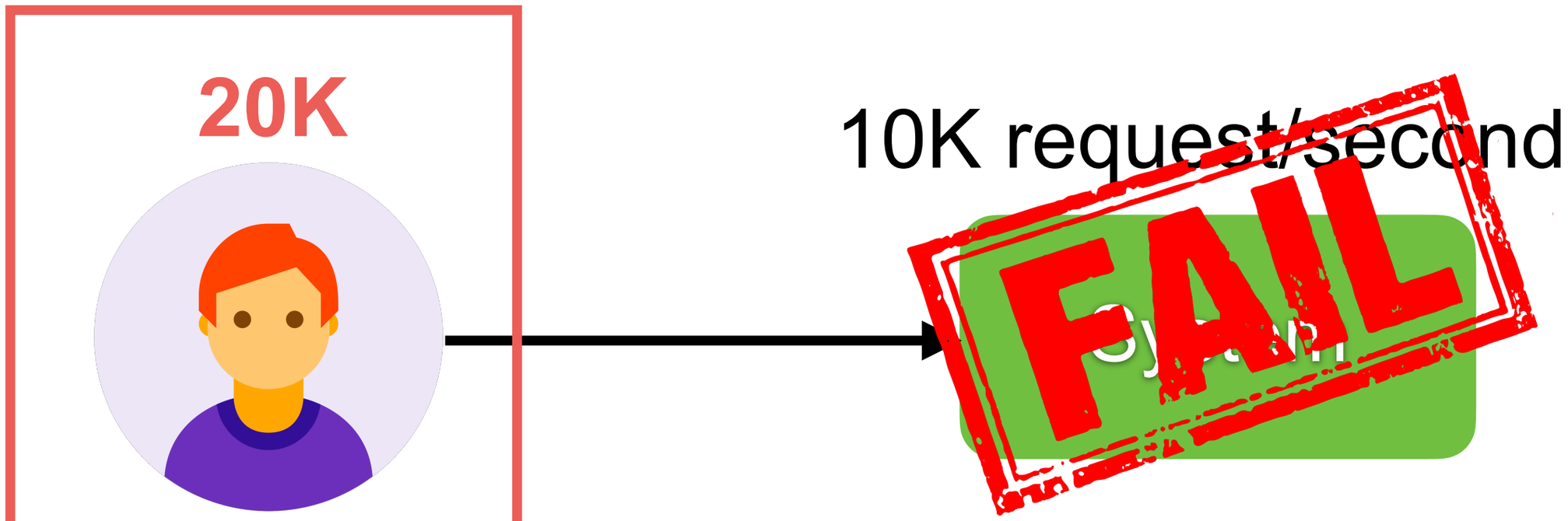


Why Reactive ?



Responsive ?

Why Reactive ?



Problems

Slow response
Outage system
User/business dissatisfied
Potential customer and money were lost

Problems

I/O tasks
Working with Database
Read and Write file
...

Thread per request model

Thread blocked until the task is completed

More threads == More memory

Scalable problem !!

Bad for utilize resources

Scalability

Scale Up

Scale Out

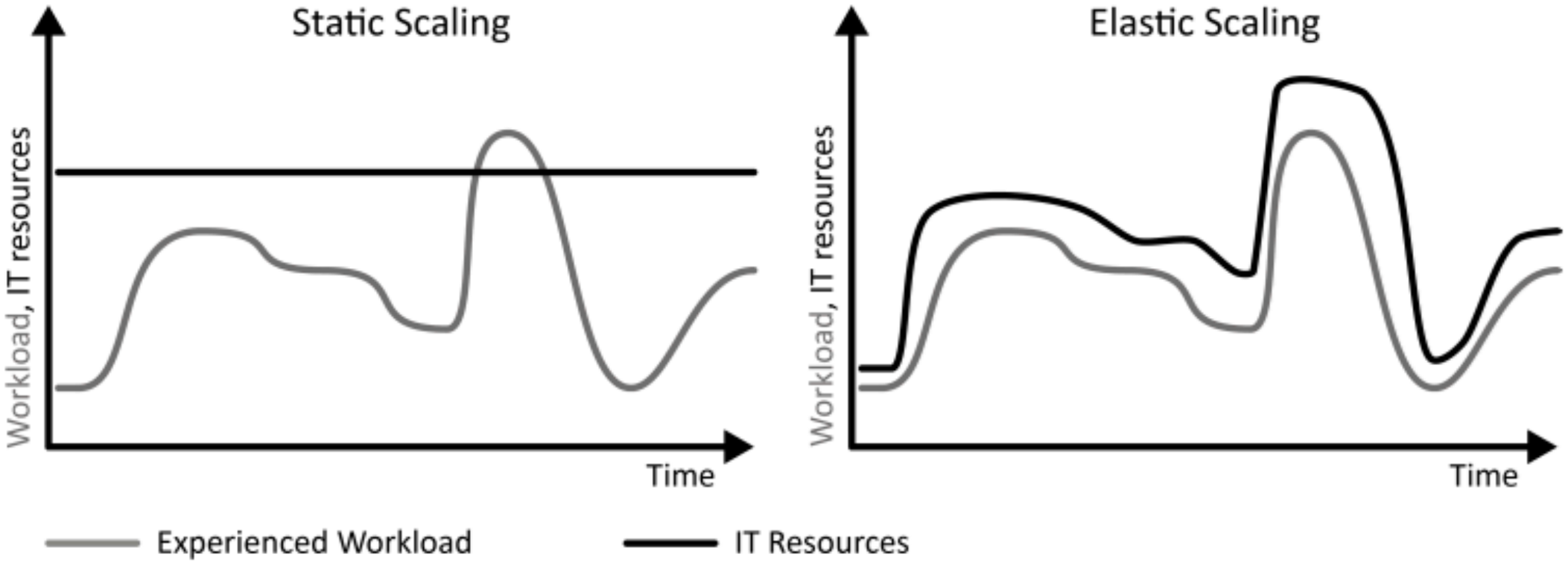
Increase cost ?

More nodes != More throughput

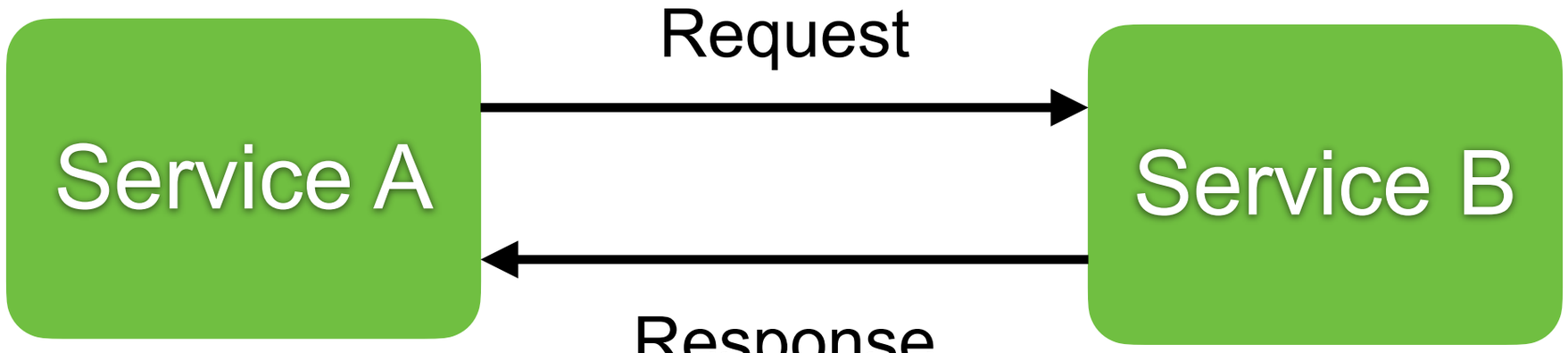
More load, More resources

Less load, Less resources

Elasticity

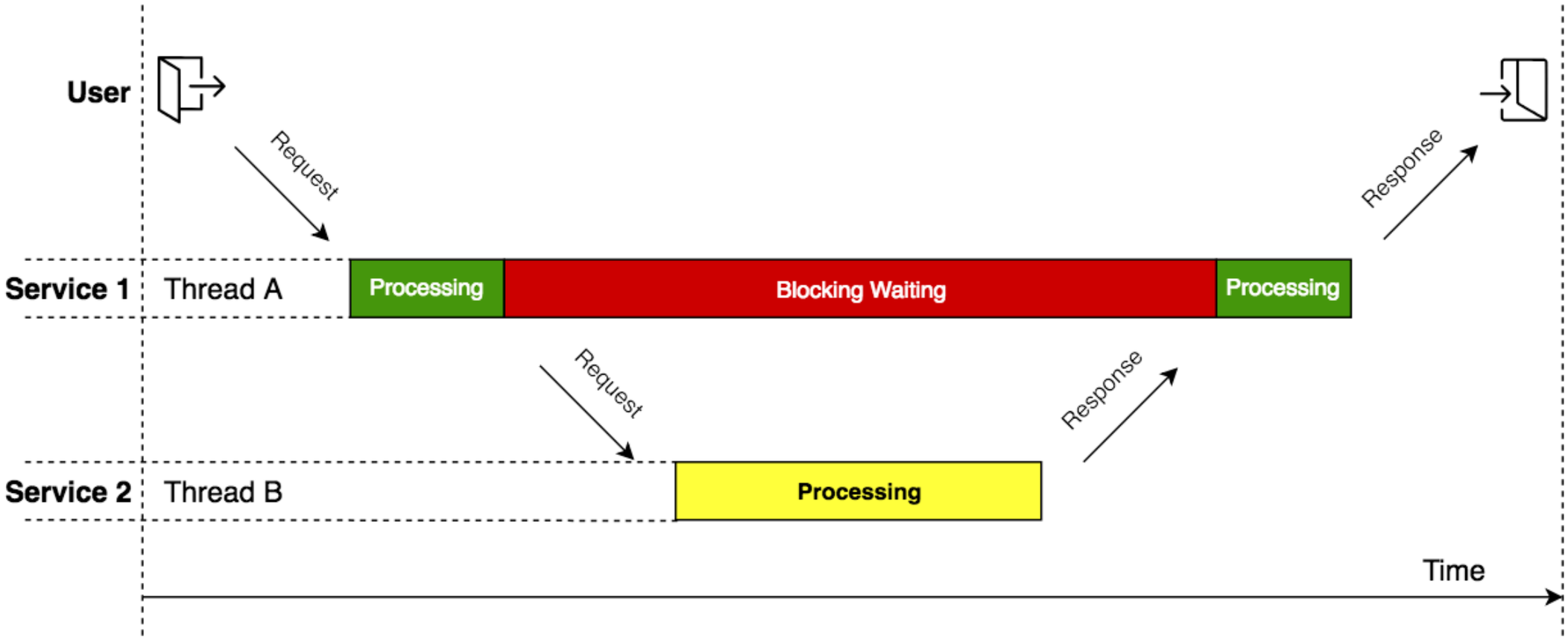


Synchronous communication

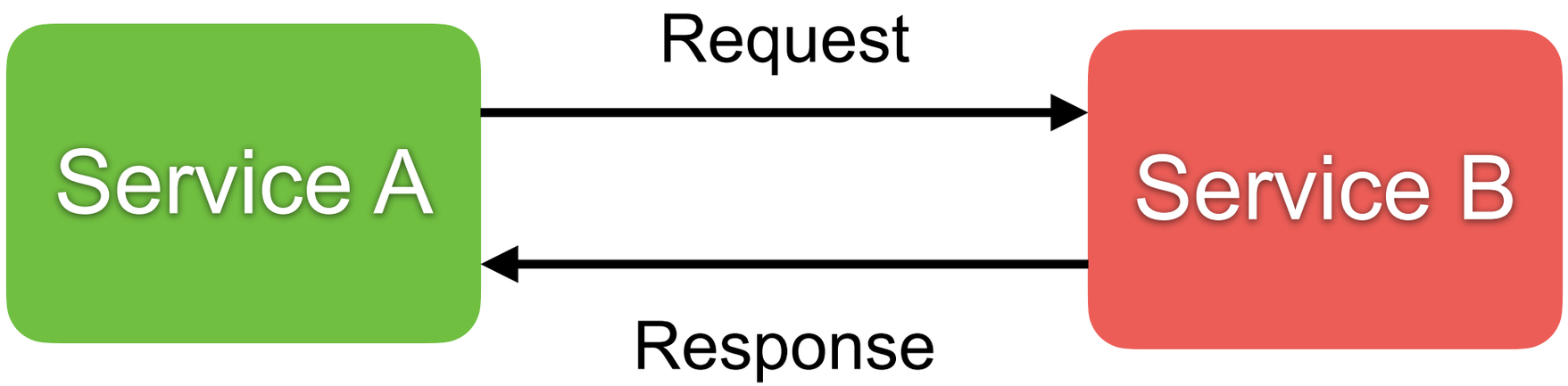


Tight coupling ?

Synchronous communication

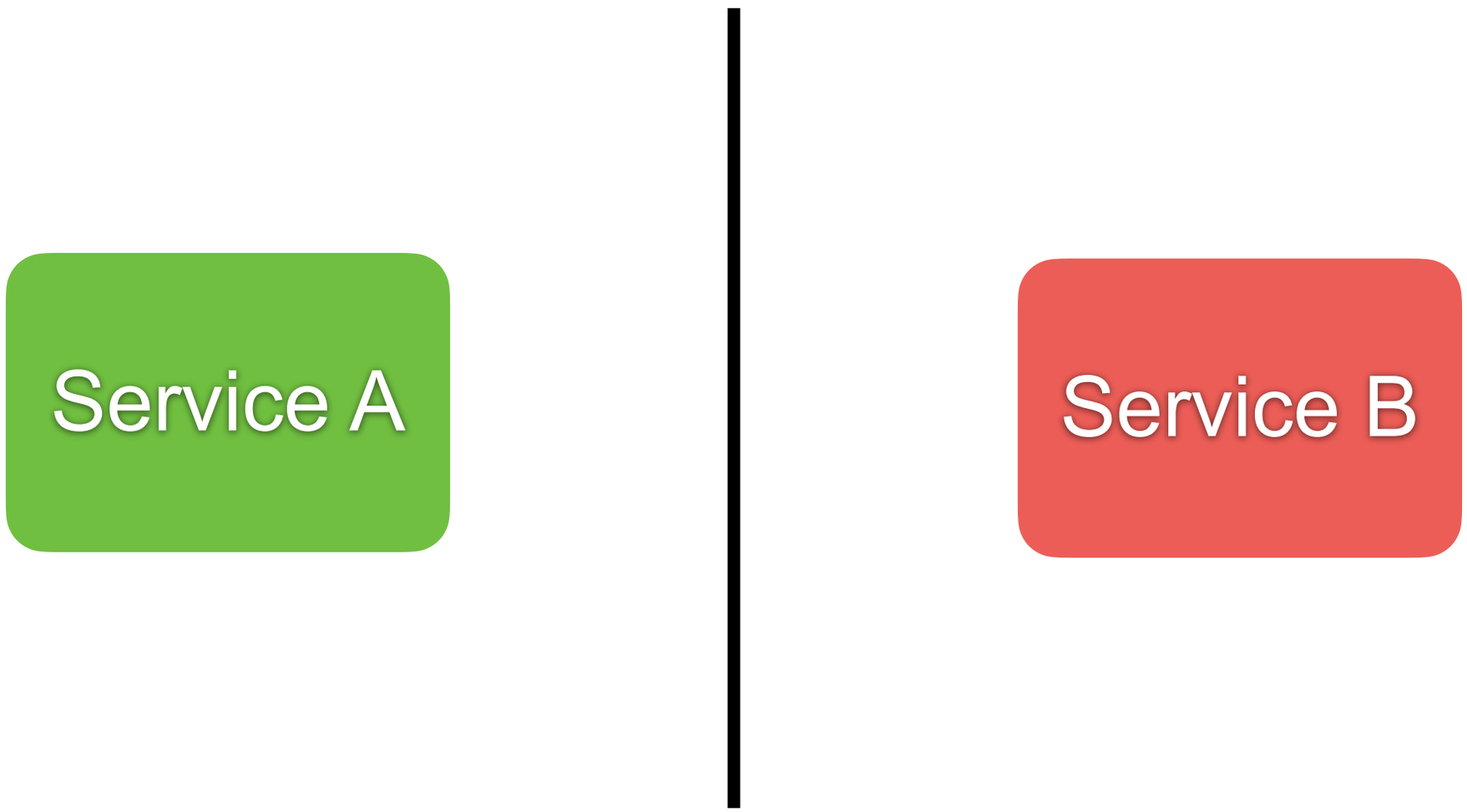


Failure ?

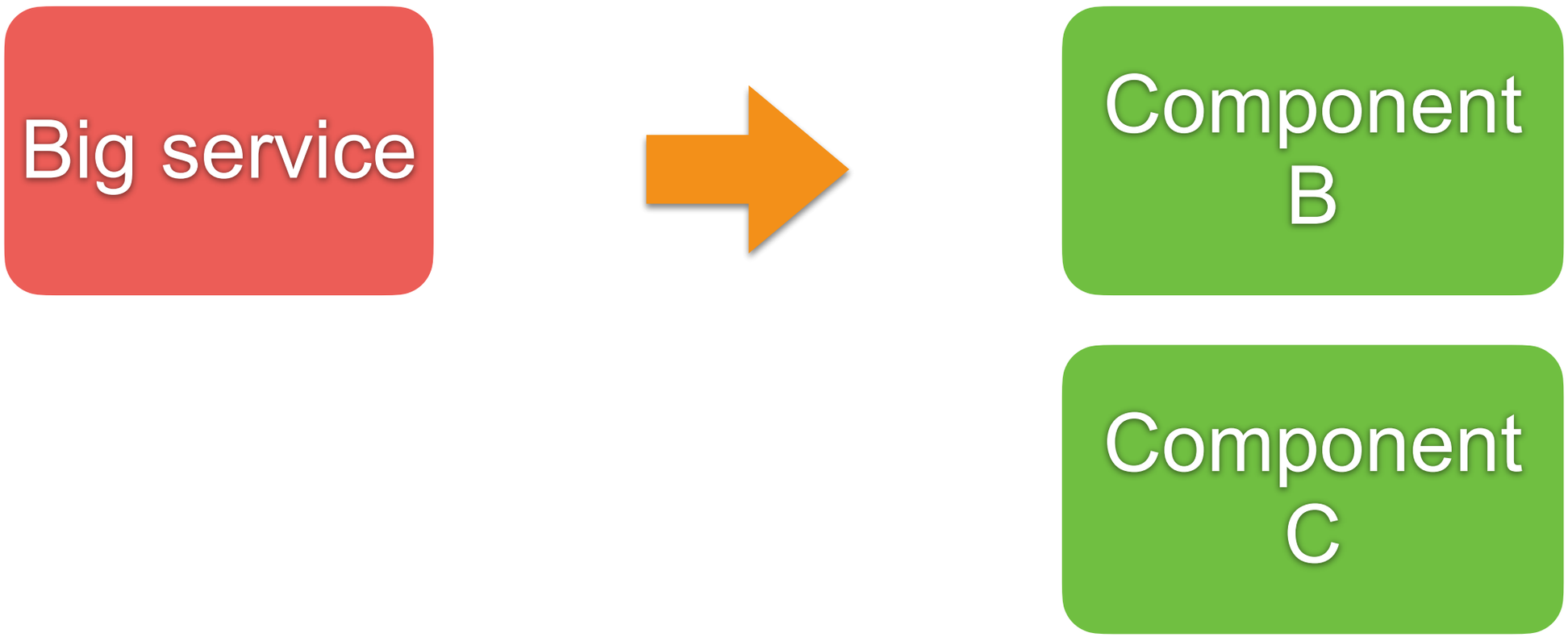


Plan for Failure !!

Isolation

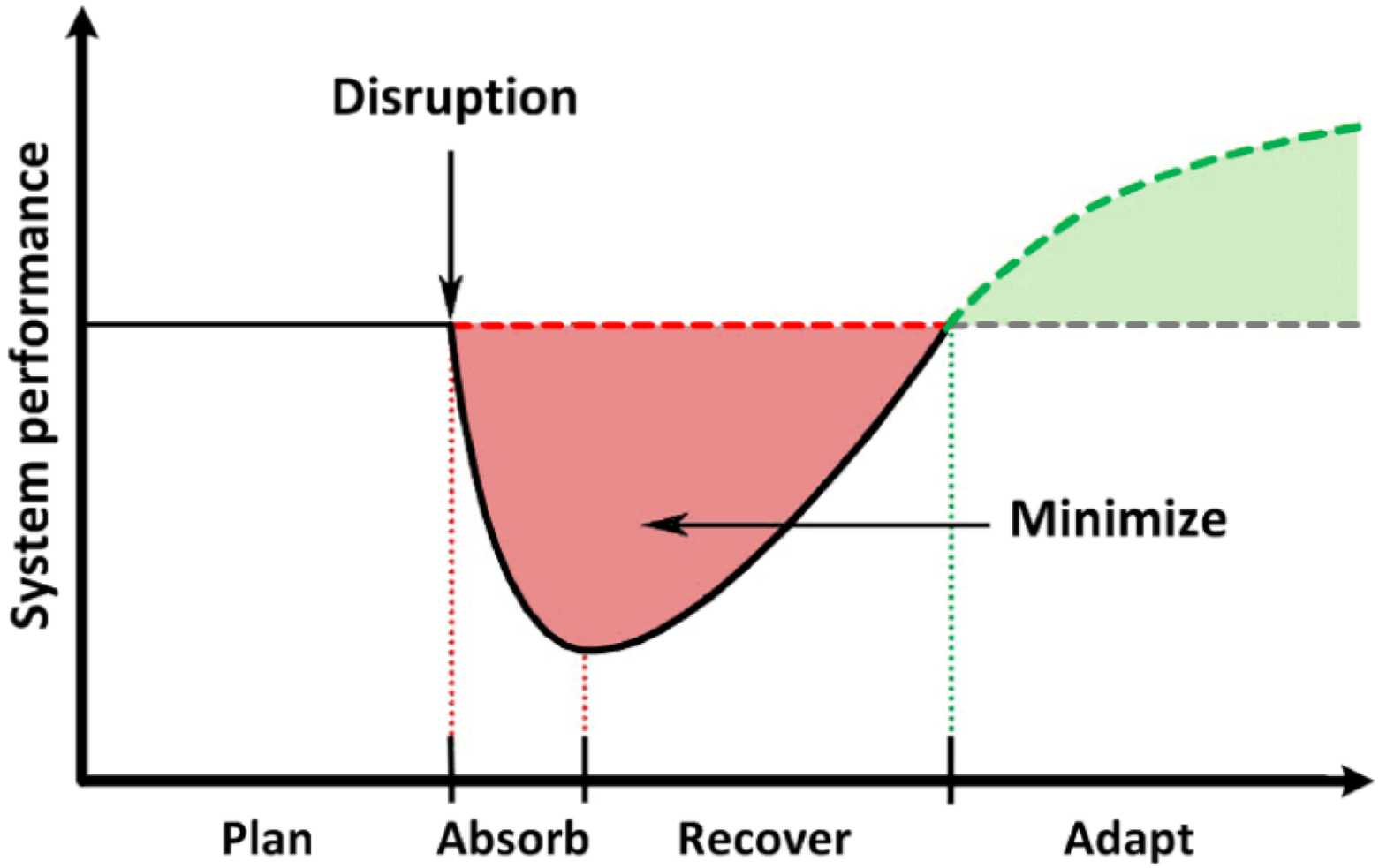


Isolate into small components

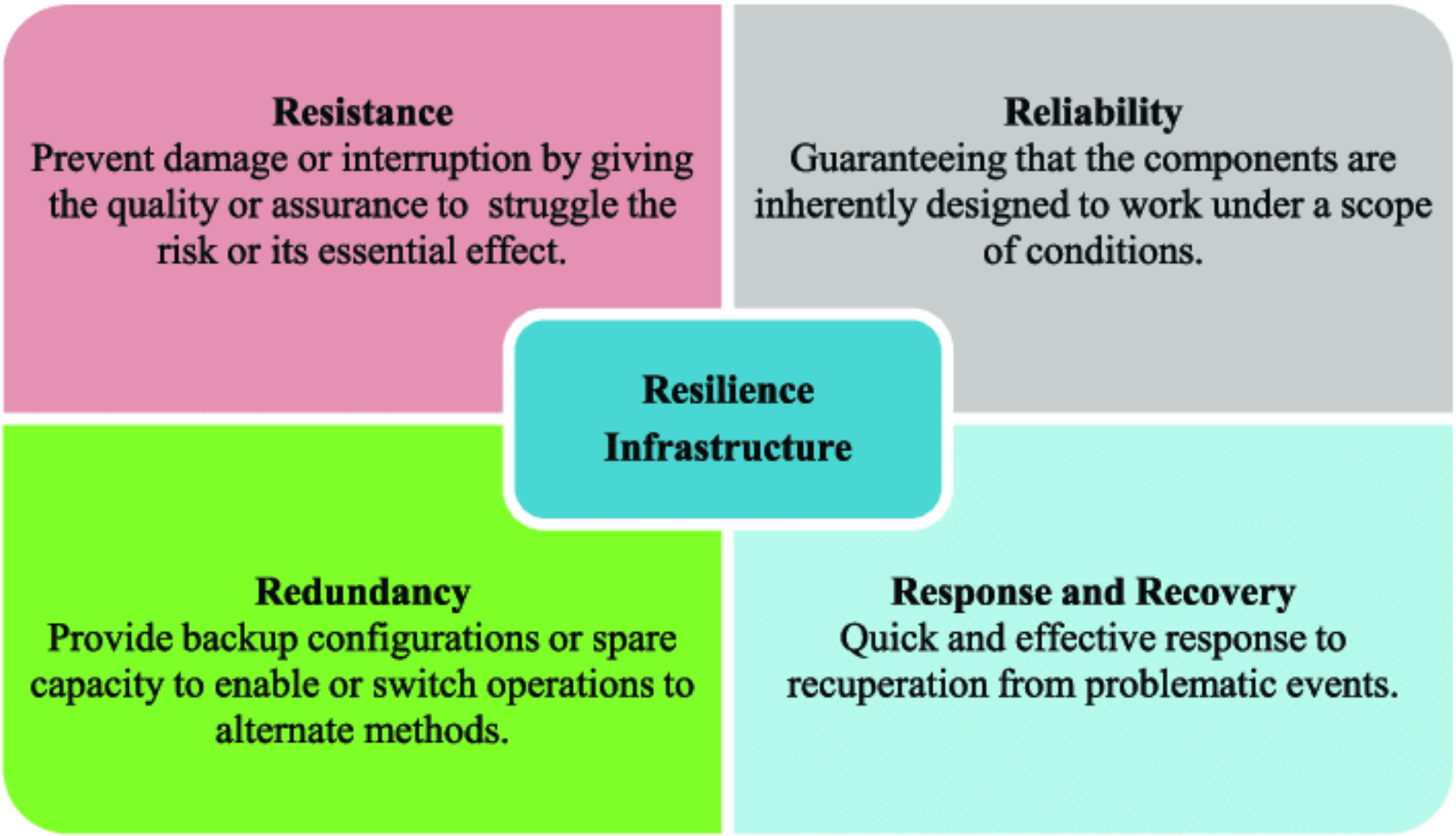


Resilience system

How to recover from failure ?



Resilience system



Need better system
Need better principles

Building robust system

Reactive System/Architecture

Scalable

Robust

Responsive

Elasticity

Reactive System

Responsive

Resilient

Elastic

Message driven

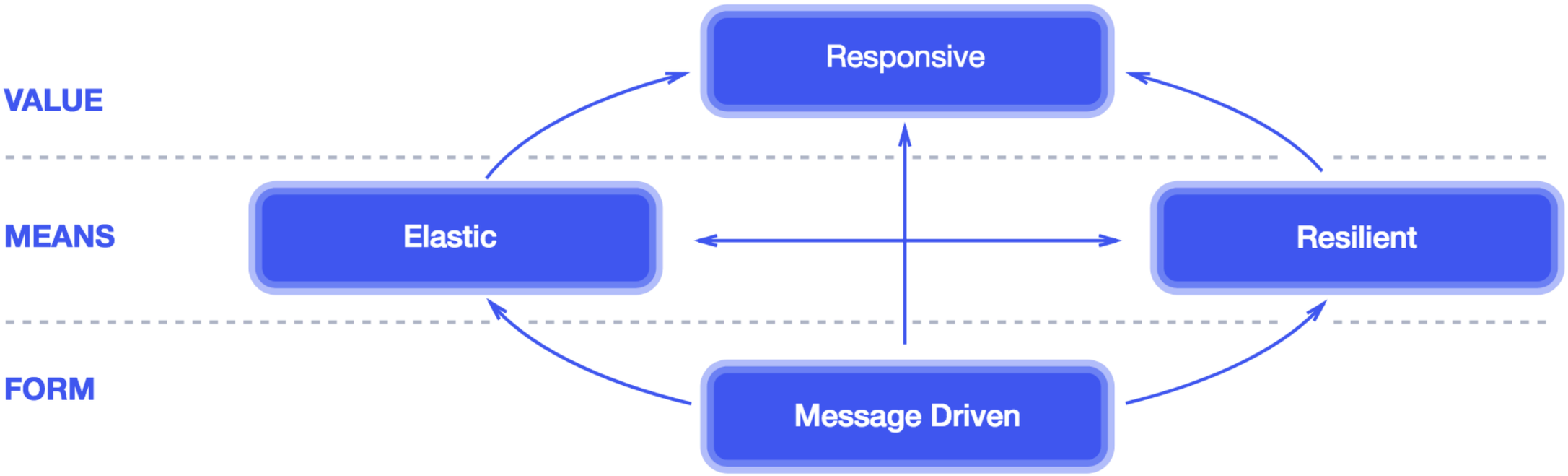
Async messaging

https://www.reactivemanifesto.org/

Reactive …

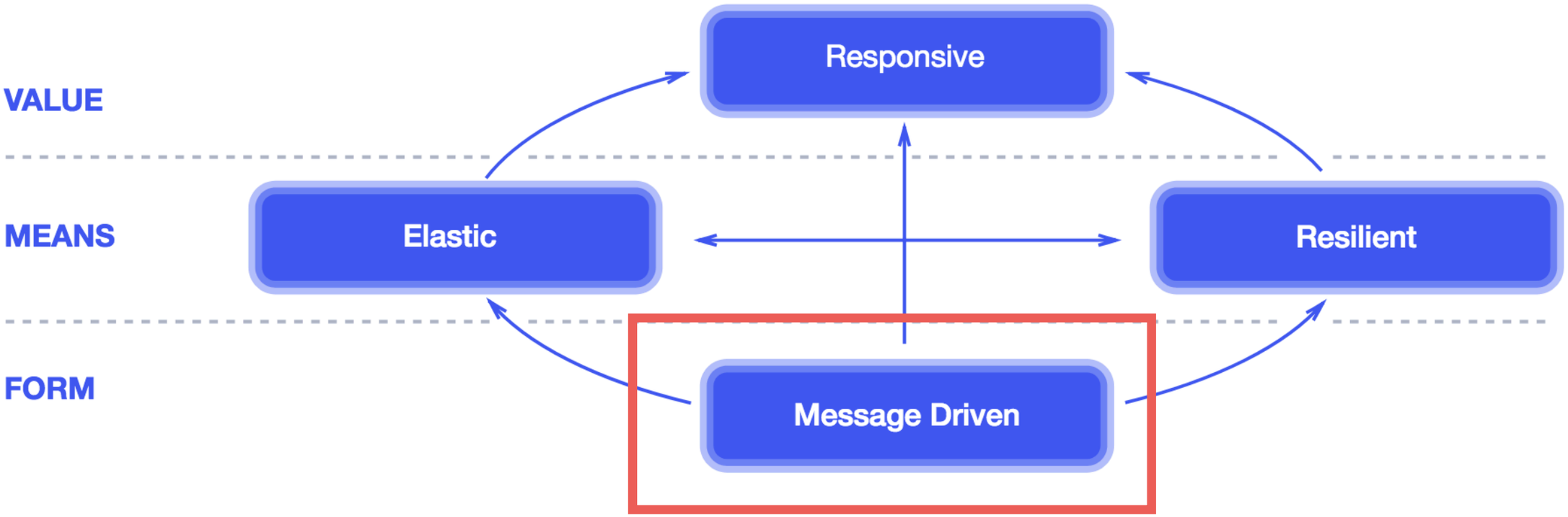
React to events => **Event/message-driven**
React to load => **Elastic**
React to failure => **Resilient**
React to users => **Responsive**

Reactive Manifesto



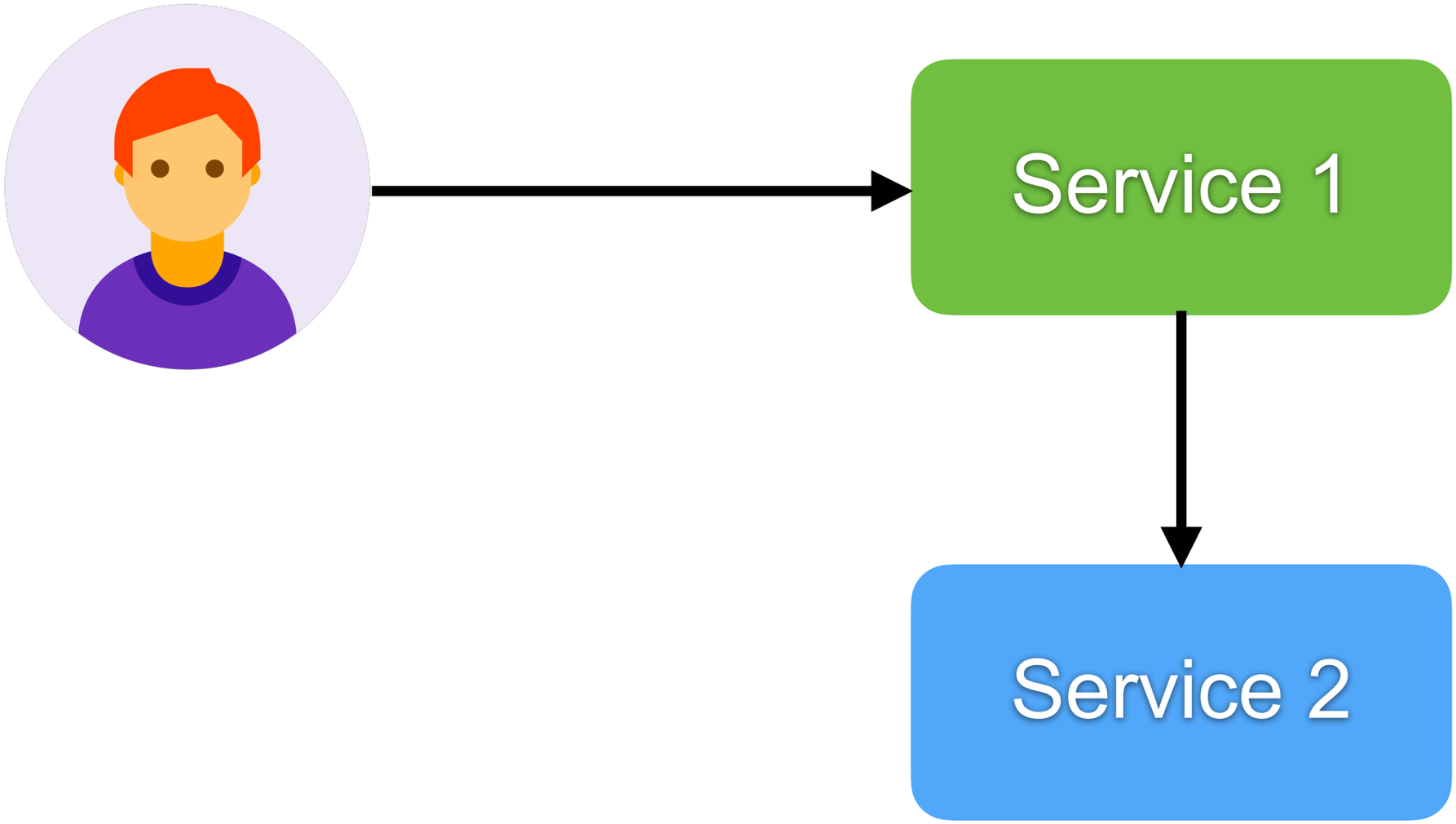
https://www.reactivemanifesto.org/

Reactive Manifesto

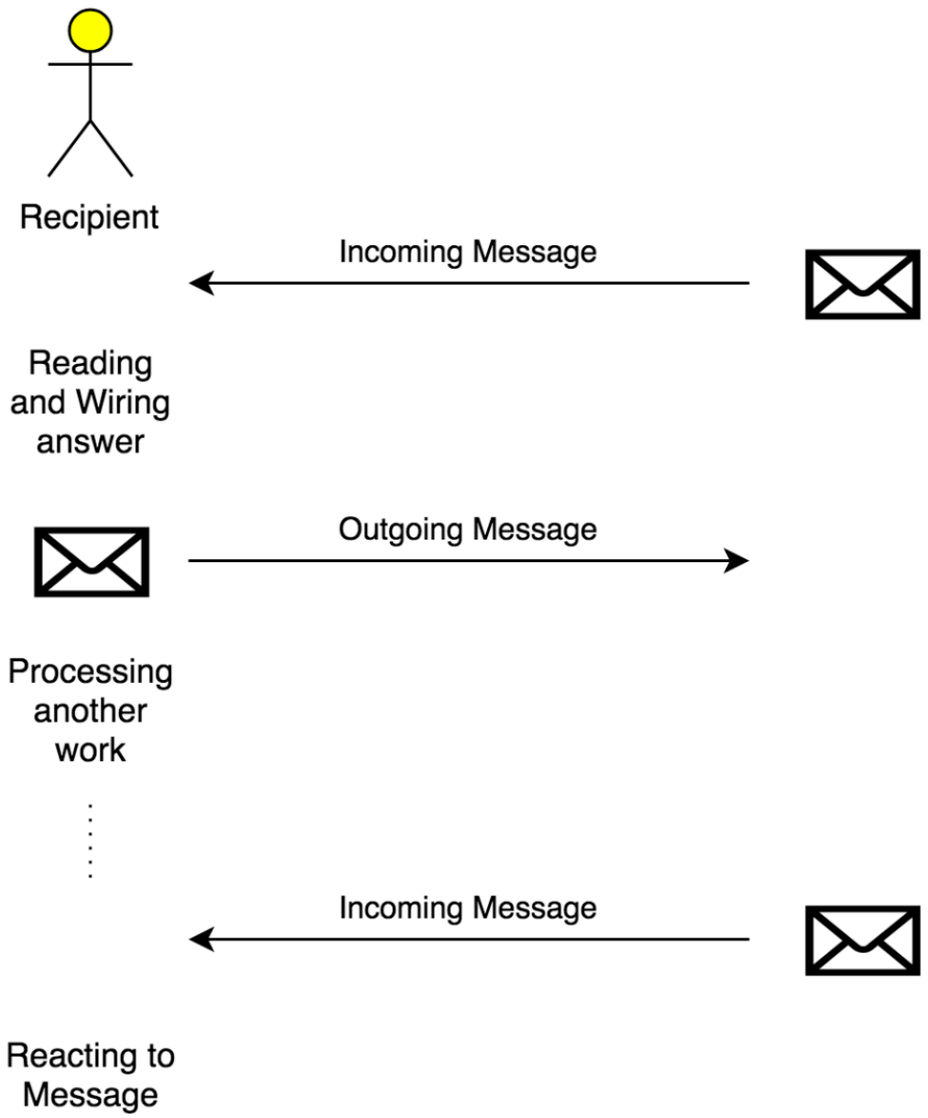


https://www.reactivemanifesto.org/

Message-driven communication

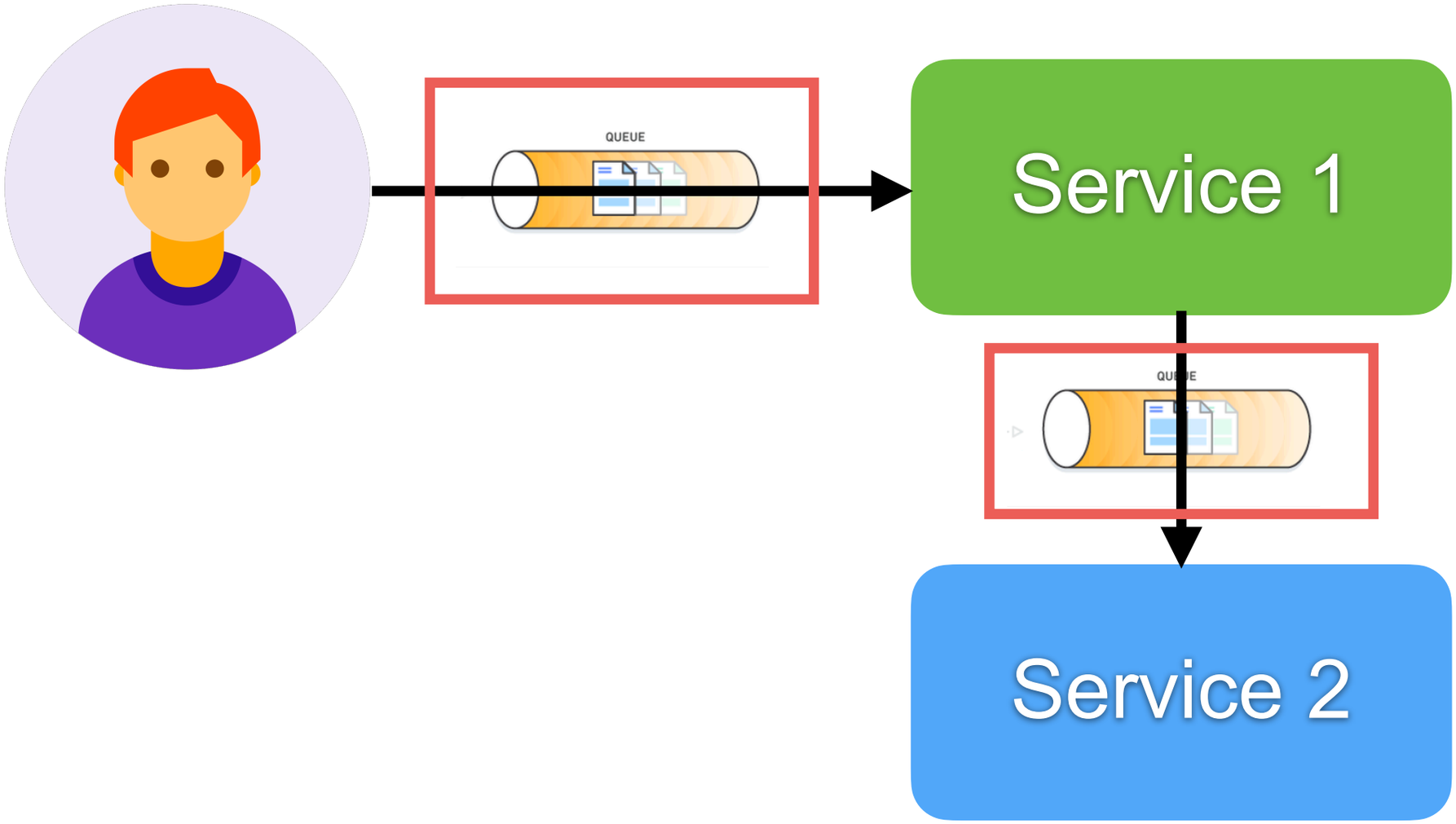


Non-blocking message communication

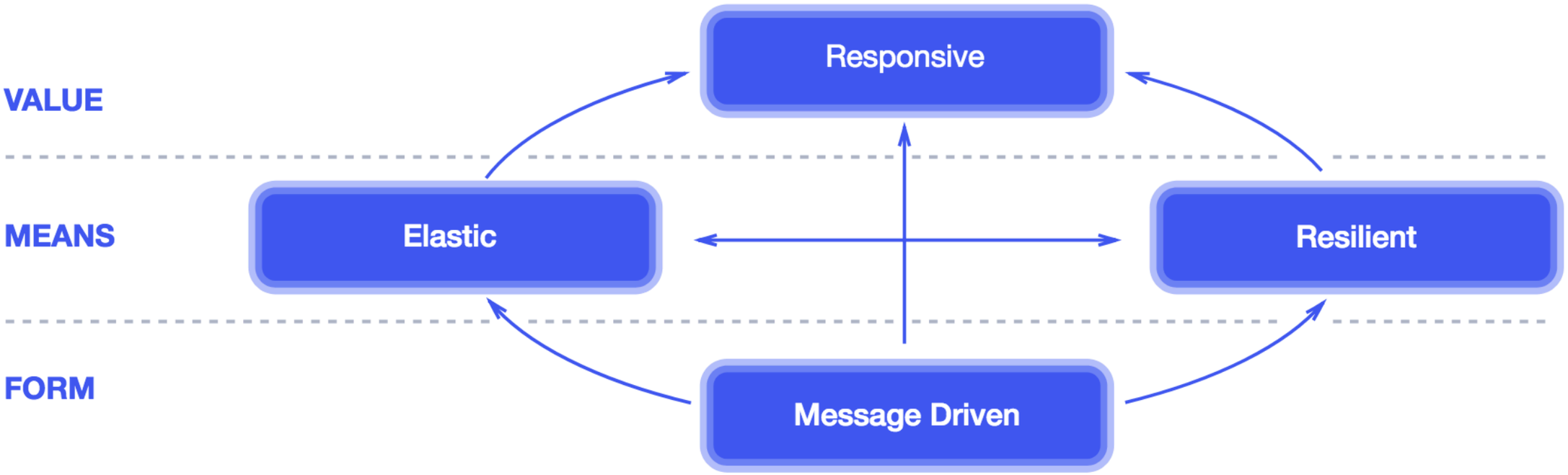


Efficient resource utilization when communicating between services in distributed system

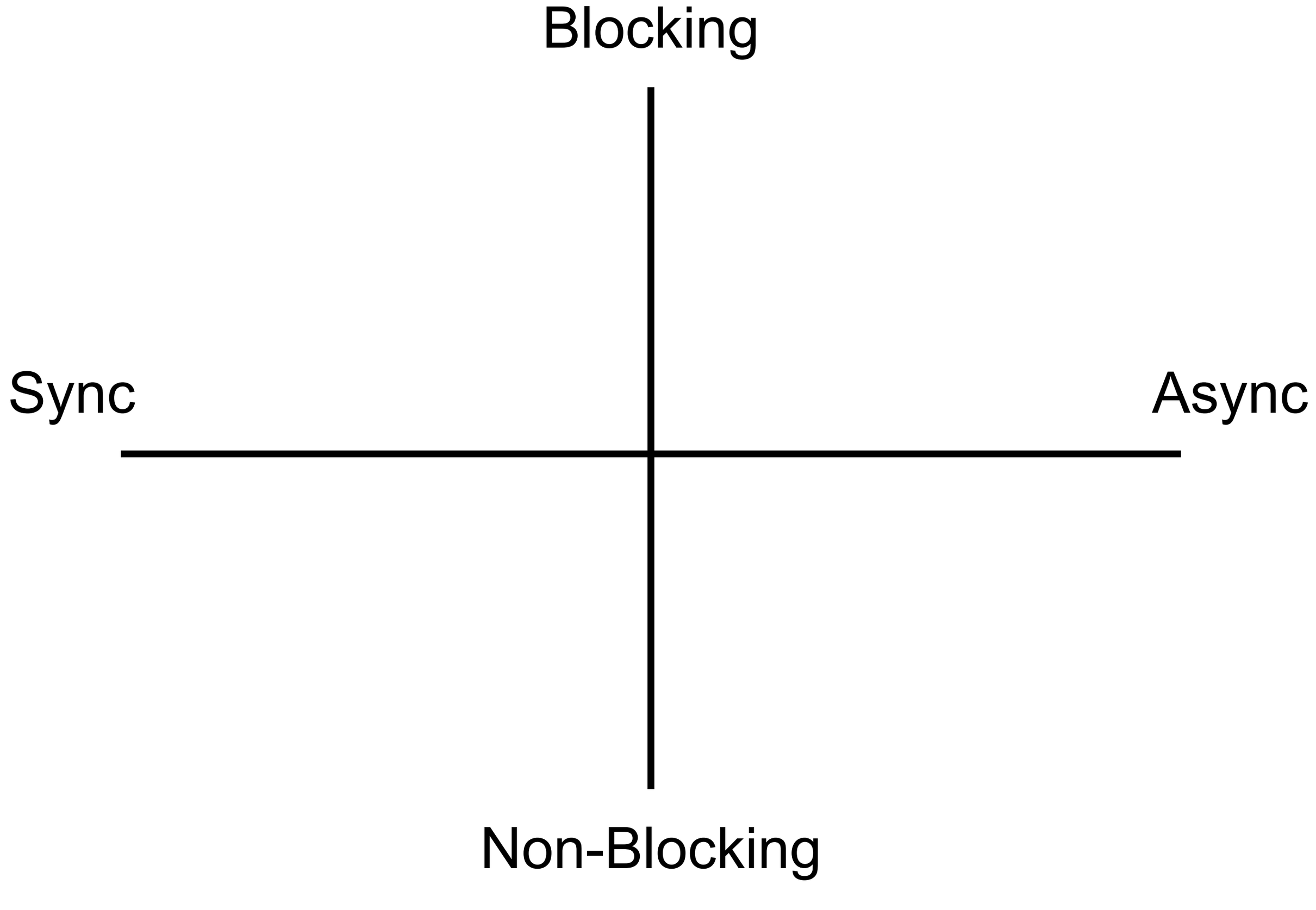
Using message broker



Reactive System



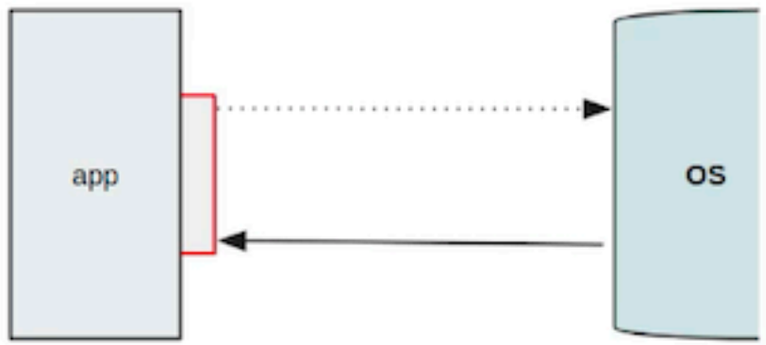
https://www.reactivemanifesto.org/



sync + blocking

async

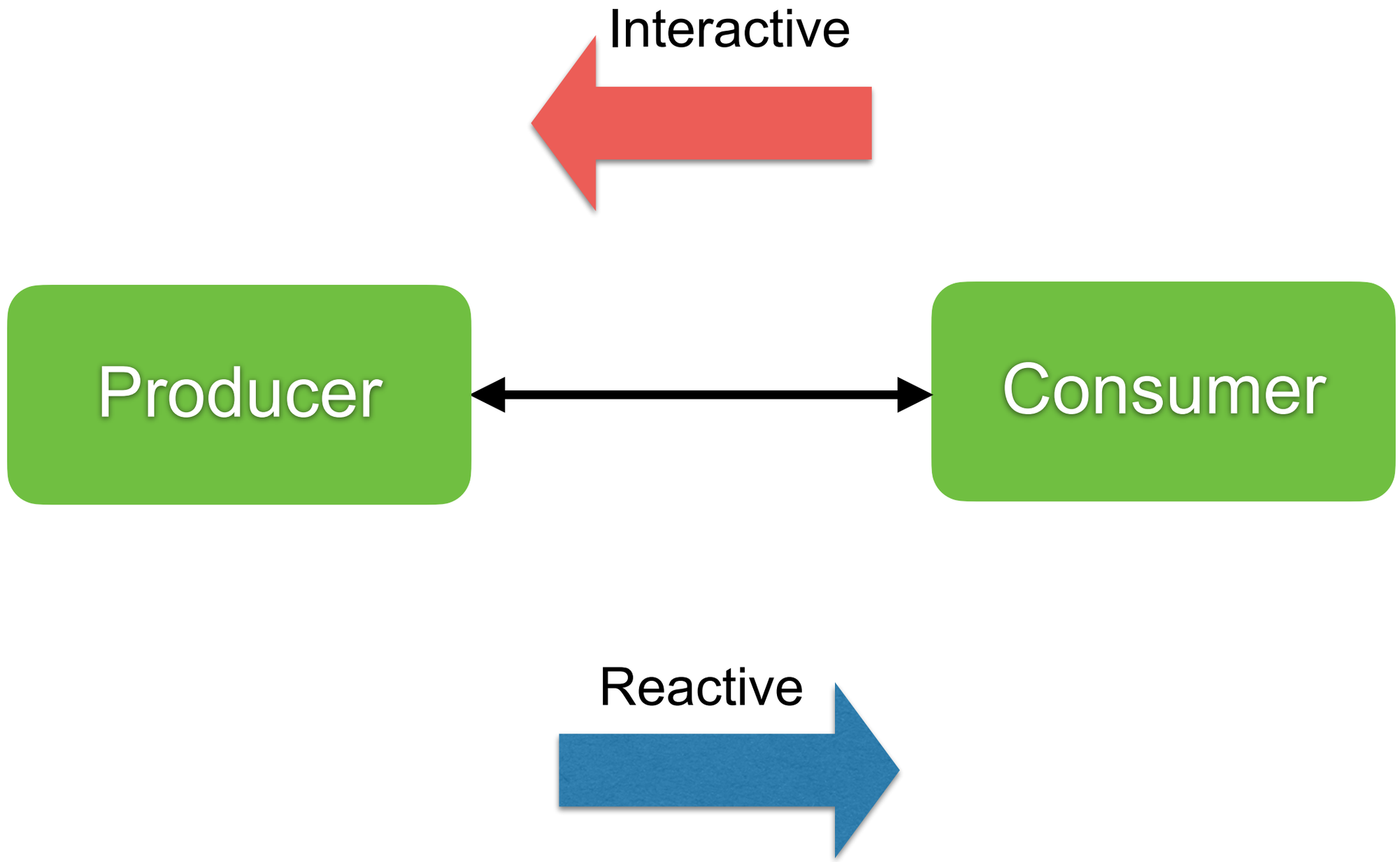
non-blocking



non-blocking + async

4 fundamental effects

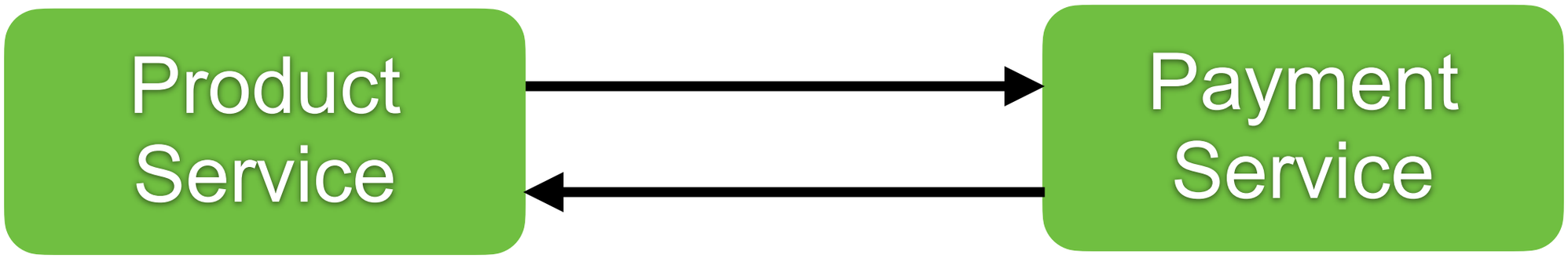
	One	Many
Sync	A	Iterator A
Async	Future A	Reactive programming Observable A

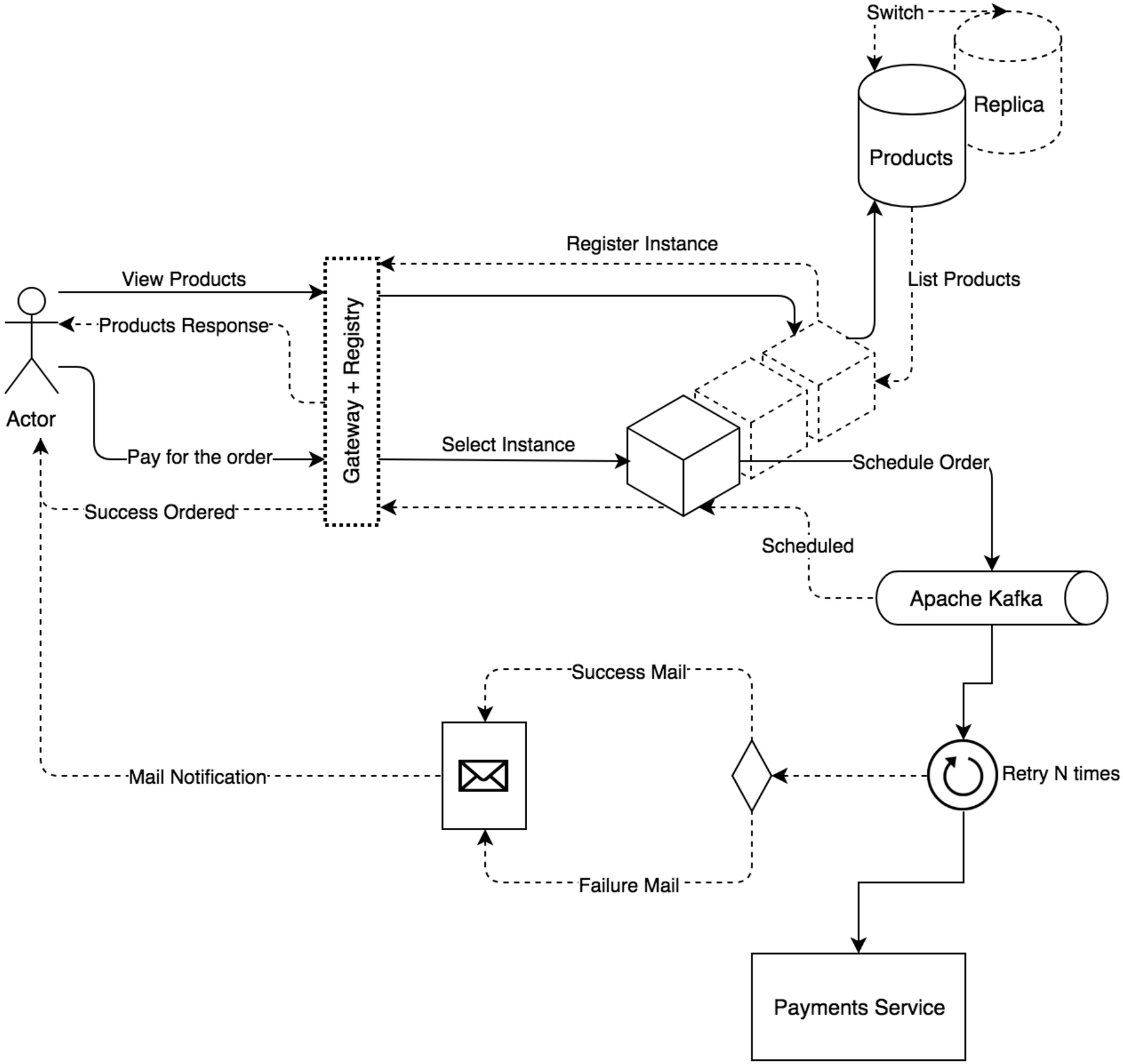


Reactive use cases

Reactive use cases

About **architecture**, can apply anywhere

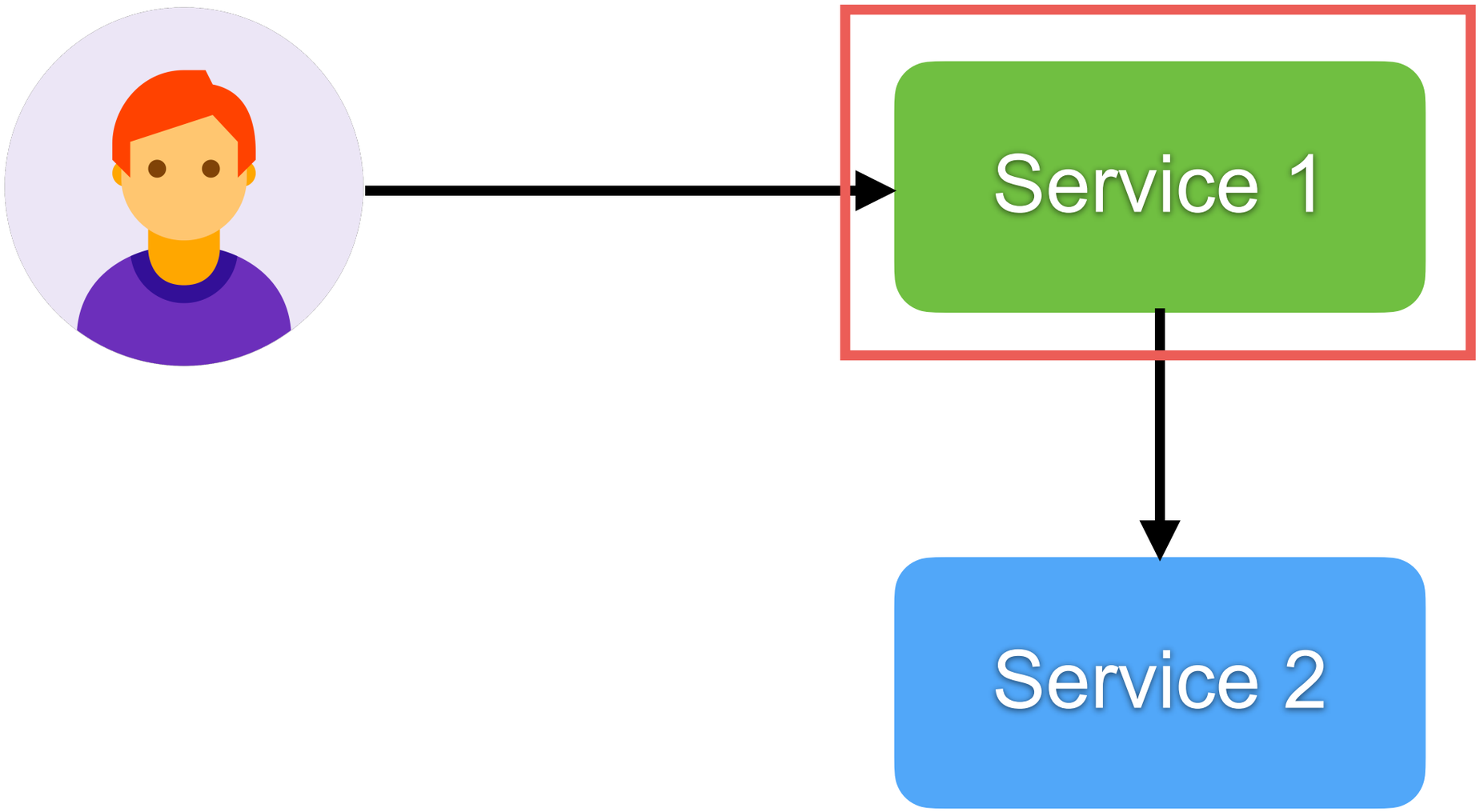




Reactive programming

https://en.wikipedia.org/wiki/Reactive_programming

Reactive programming



Reactive programming

Implementation technique
Subset of Asynchronous programming
Decompose the problem into multiple discrete step
Asynchronous and non-blocking

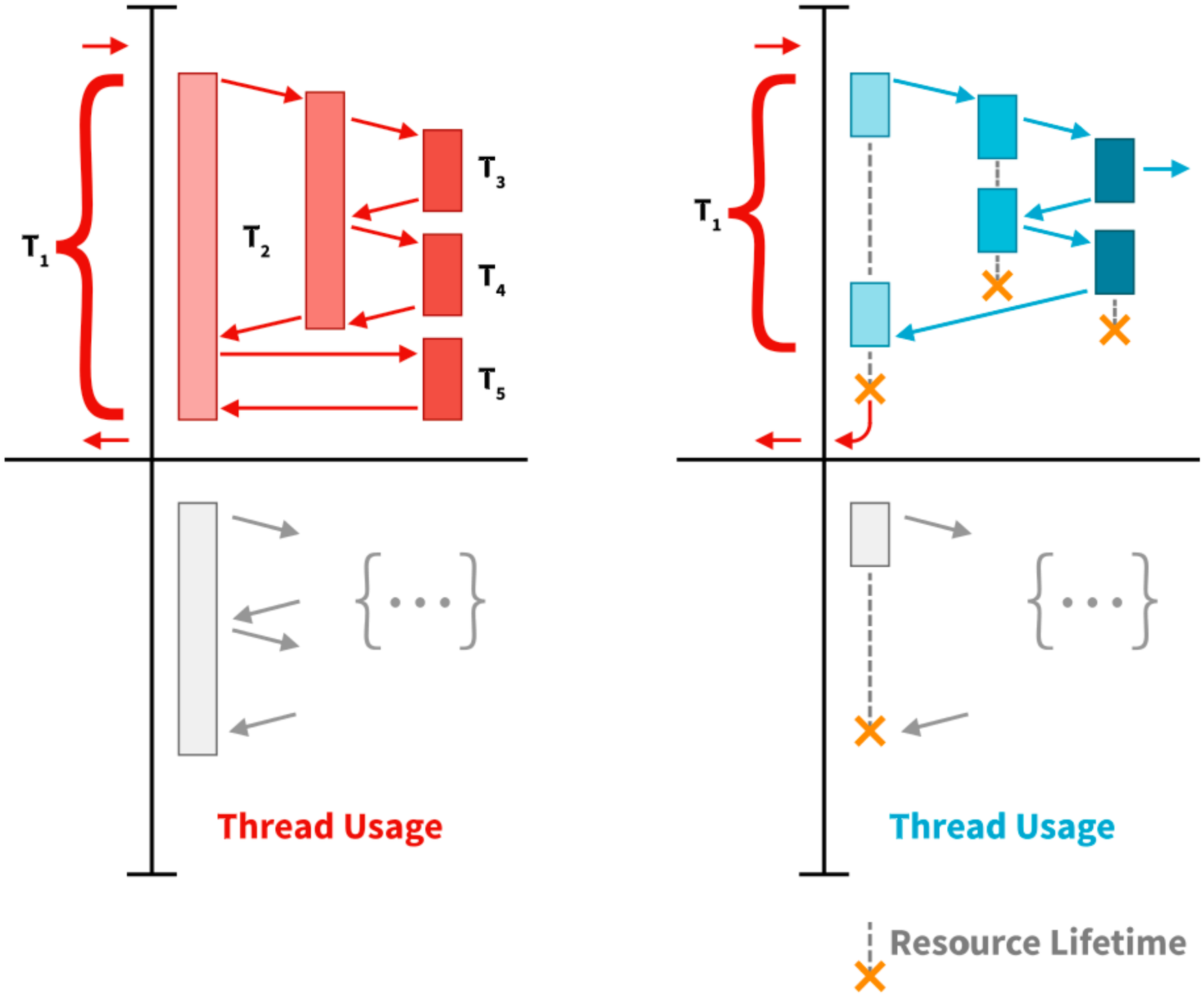
Dataflow / Stream programming

Reactive programming

Async data processing

Non-blocking

Functional / Declarative Programming



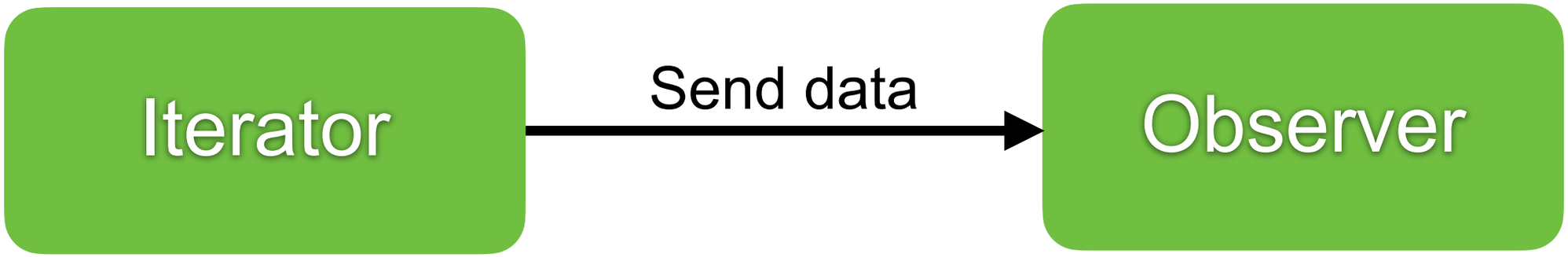
Synchronous, blocking communication (left) is resource inefficient and easily bottlenecked. The Reactive approach (right) reduces risk, conserves valuable resources, and requires less hardware/infrastructure.

https://mostafa-asg.github.io/post/reactive-systems-vs-reactive-programming/

API for Reactive programming

Callback

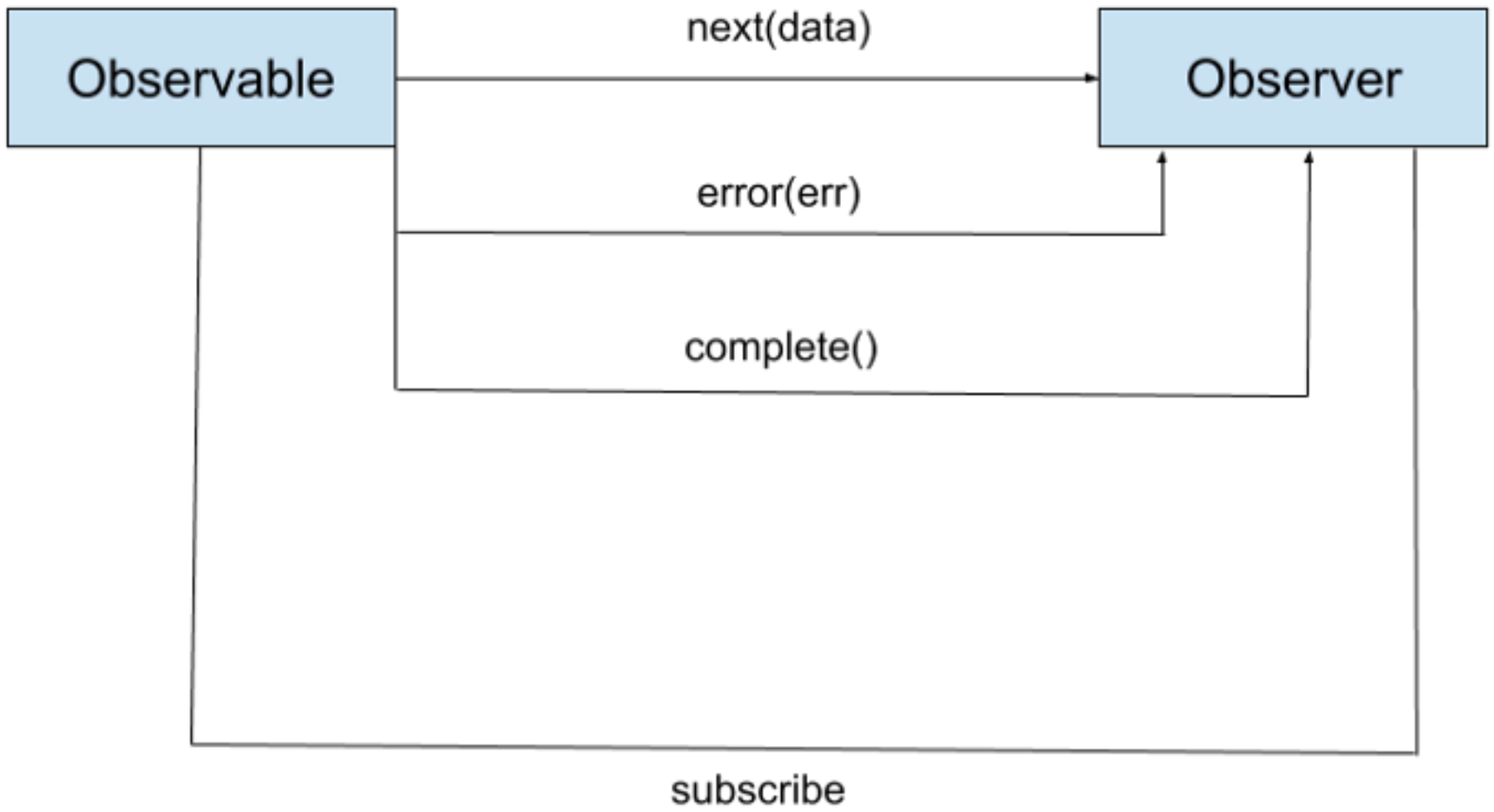
Declarative with functional composition



Design patterns

Iterator pattern
Observer pattern

Components



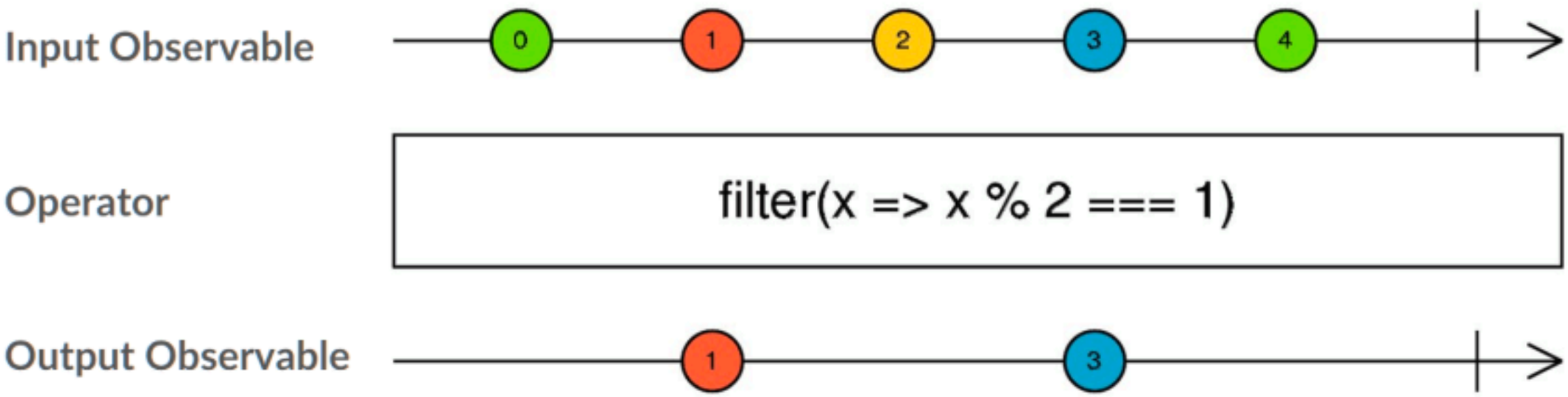
Reactive Streams

```
public interface Publisher<T> {
    public void subscribe(Subscriber<? super T> s);
}
public interface Subscriber<T> {
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}
```

```
public interface Subscription {
    public void request(long n);
    public void cancel();
}
```

Stream everywhere

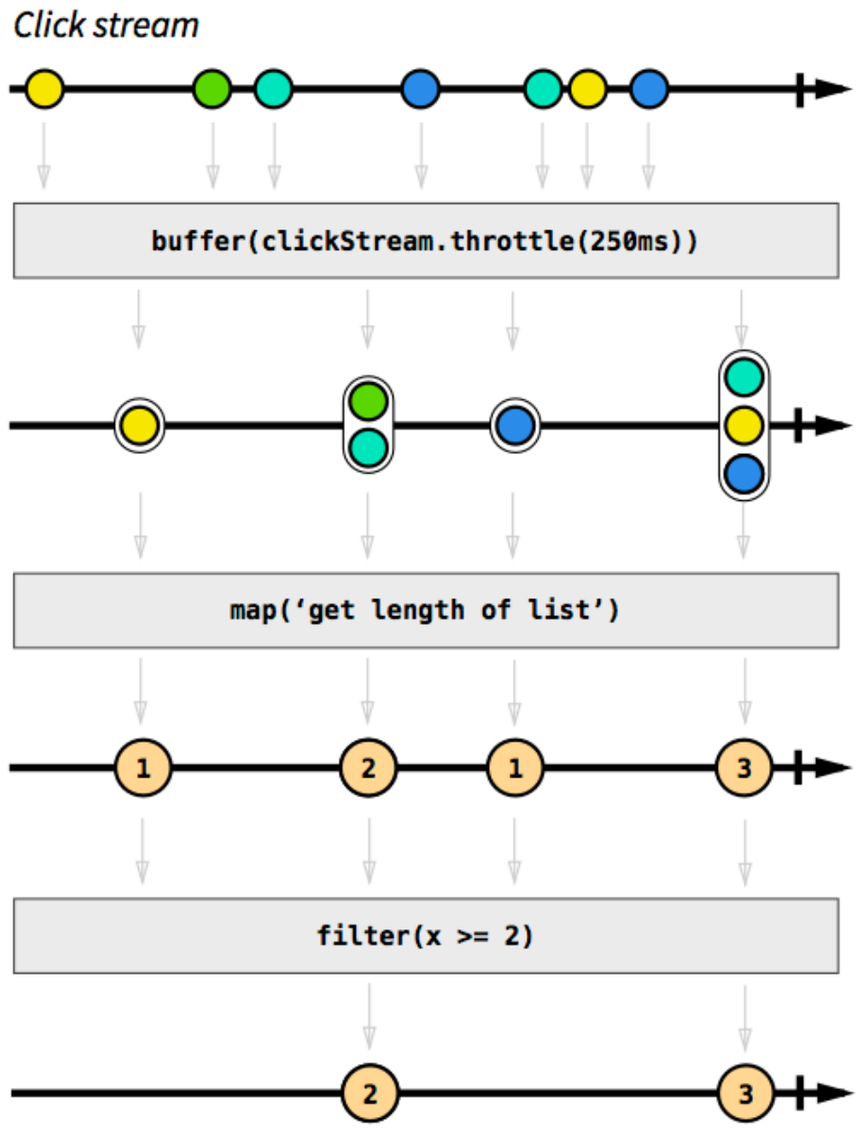
Many processes can be modeled as a stream



https://reactive.how/

Stream everywhere

Many processes can be modeled as a stream



JVM Reactive Libraries

RxJava

Vert.X

Akka

Ratpack

Reactor + WebFlux

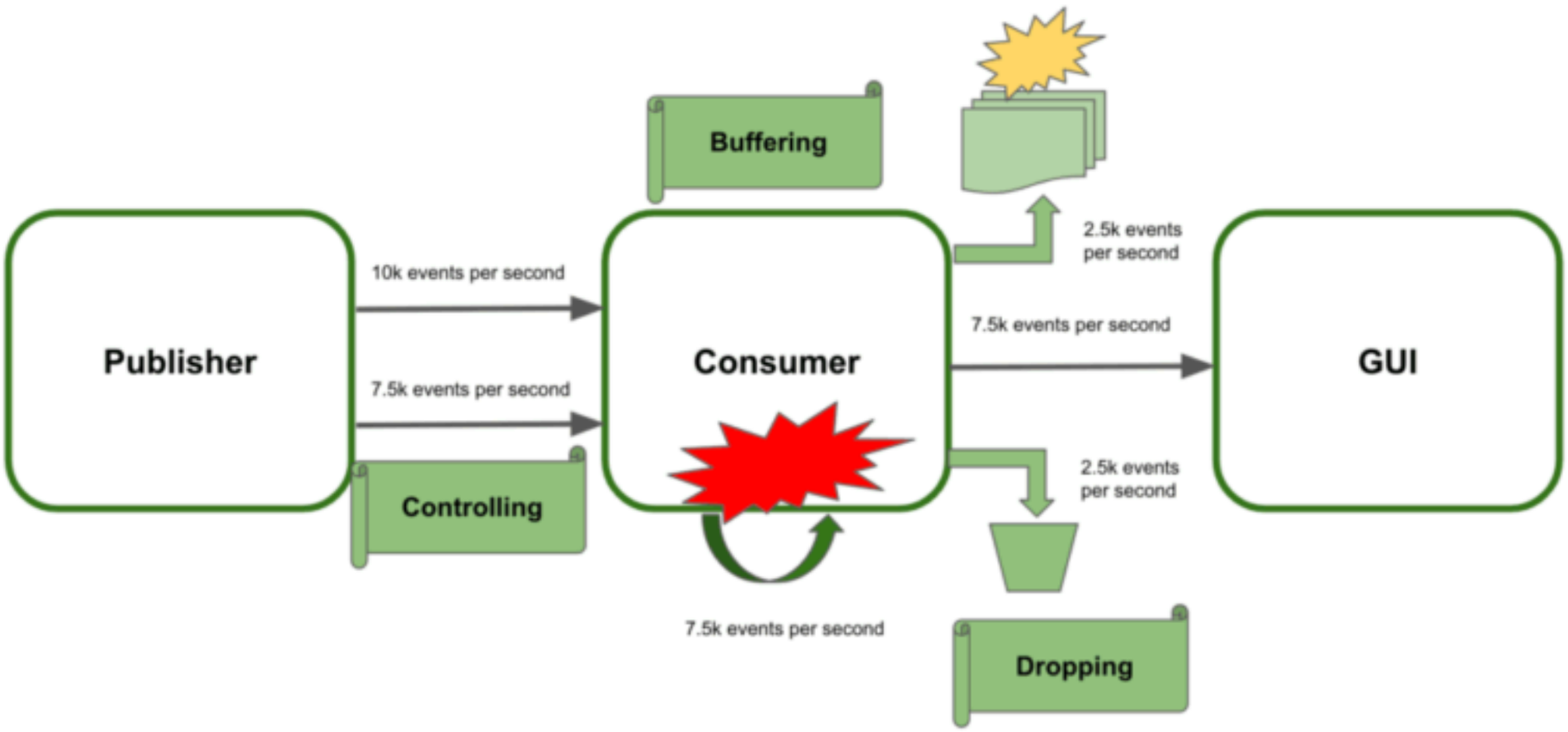
http://reactivex.io/languages.html

Benefits Reactive programming

Increase utilization of computing resources
Increase performance by reduce serialization point
Developer productivity
Avoid over-utilization (back-pressure)

Back-pressure

To prevent systemic failures



https://www.baeldung.com/spring-webflux-backpressure

Reactive programming

!=

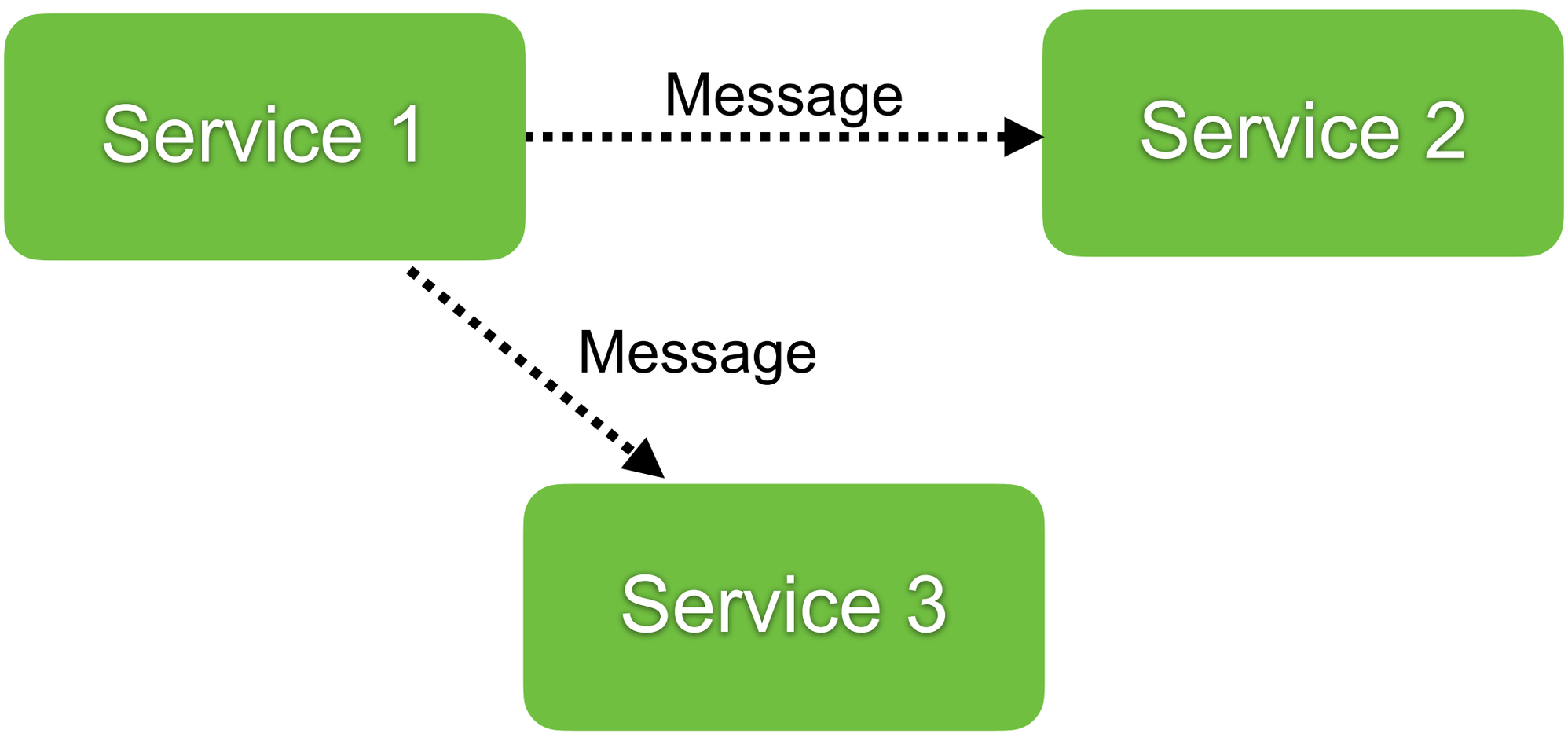
Reactive system

Reactive programming

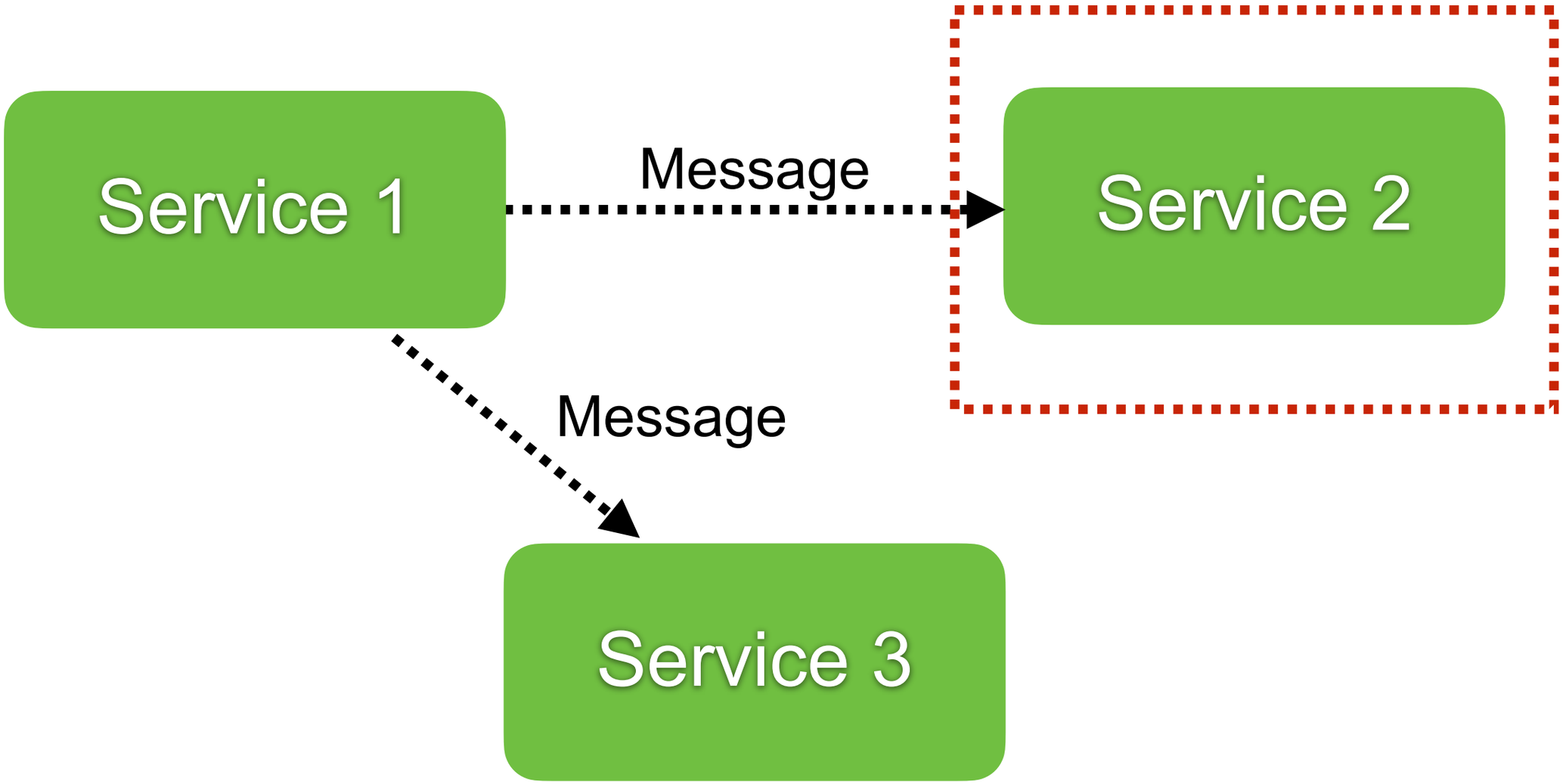
Provide a development model with async concept
Readable and understandable

Not transform your system into a Reactive system

Reactive system

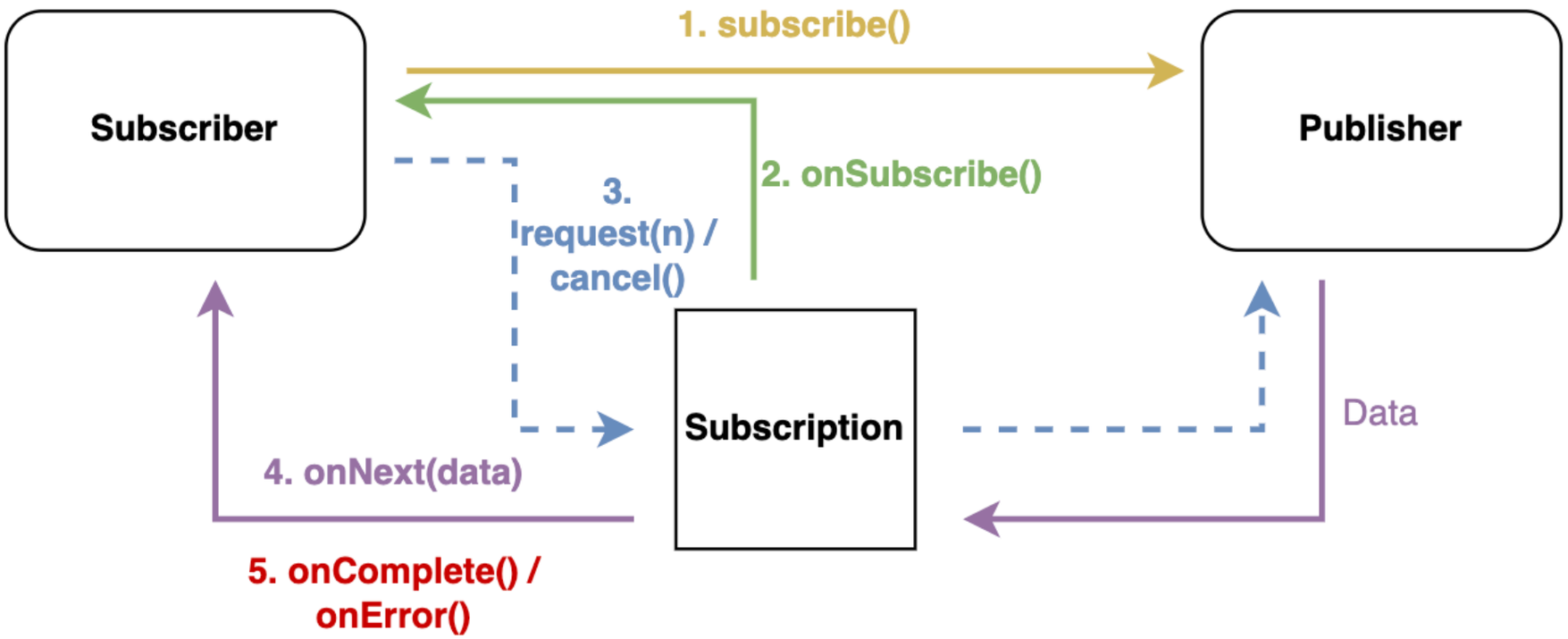


Reactive programming



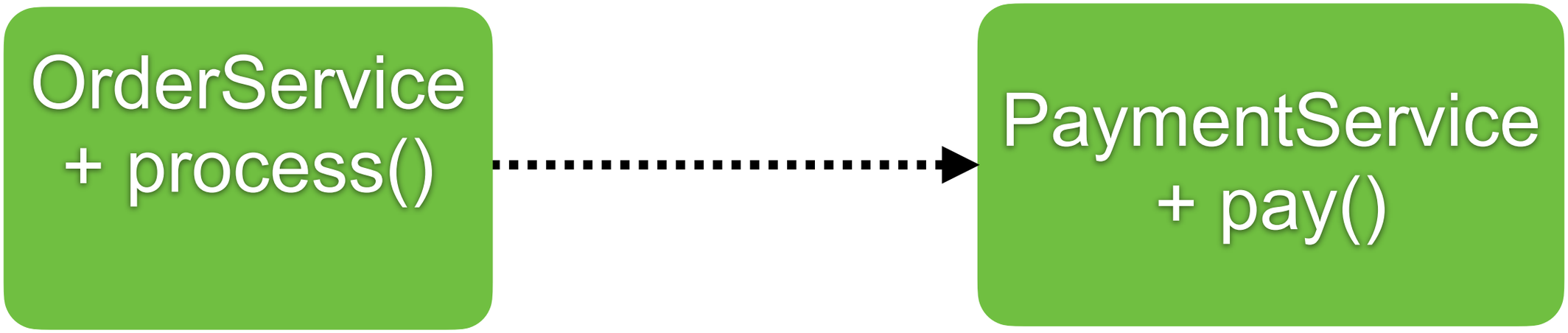
Publisher/Subscriber

Communication in same process



Coding time …

Example with Java (in-process)

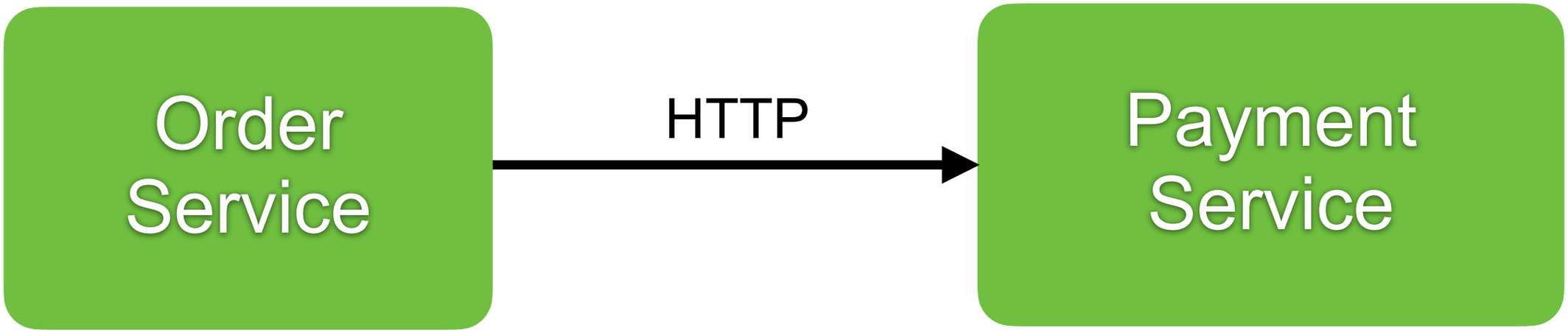


Slow process !!!

Solutions

Imperative
Callback
Future
CompletionStage

Service-to-Service



Slow process !!!

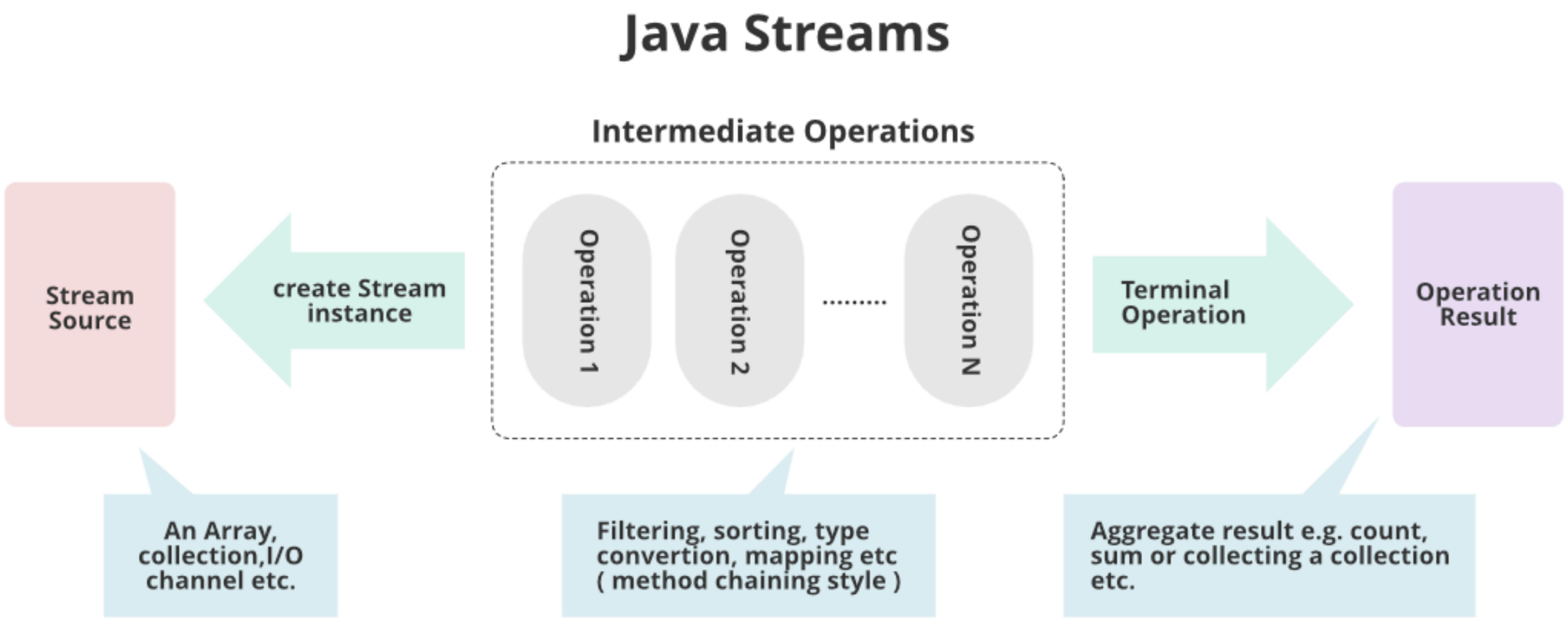
Java Stream

Java Stream

Represent a sequence of data
Focus on computation
Internal iteration

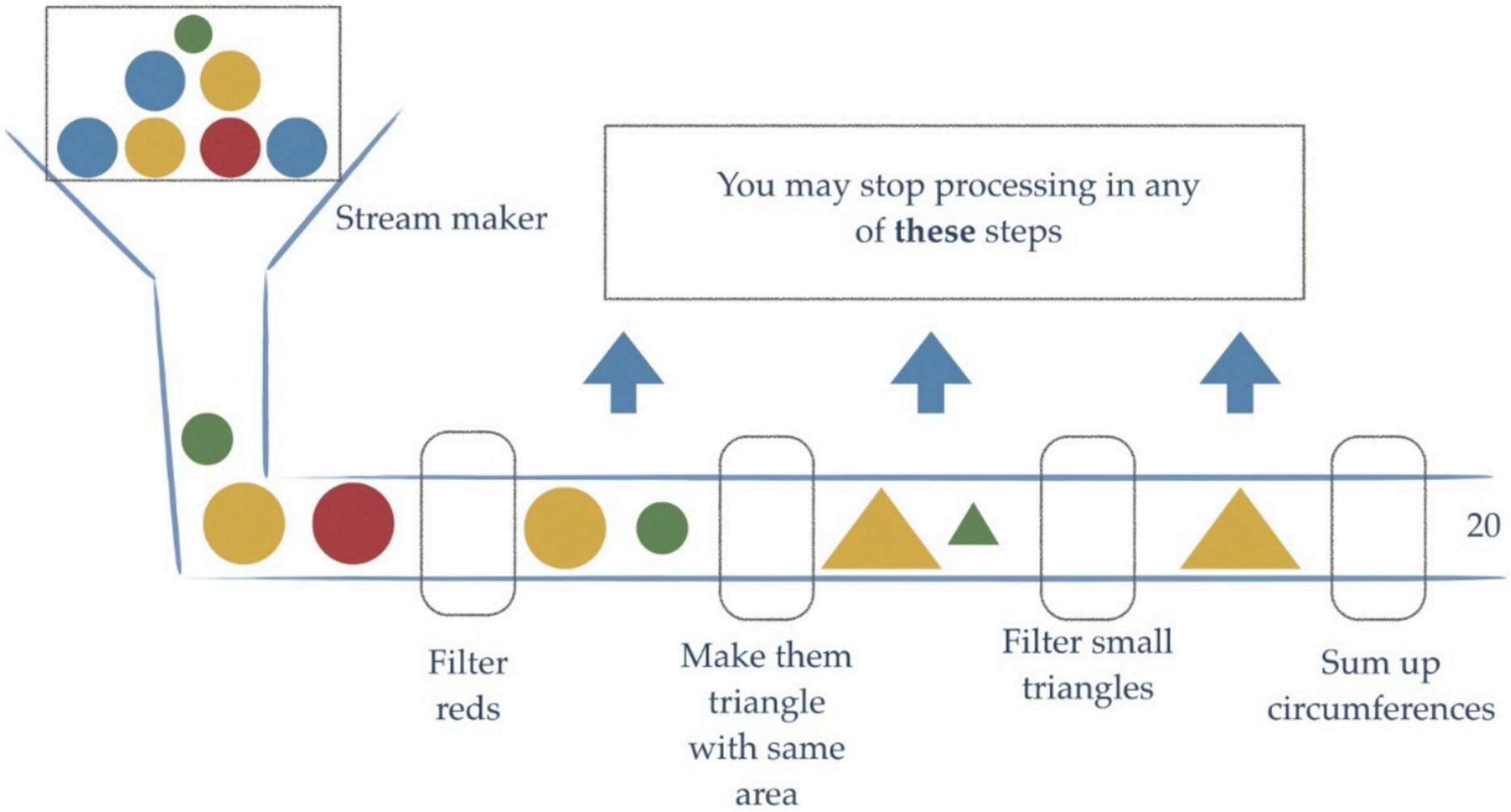
https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html

Working with Stream



https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html

Working with Stream

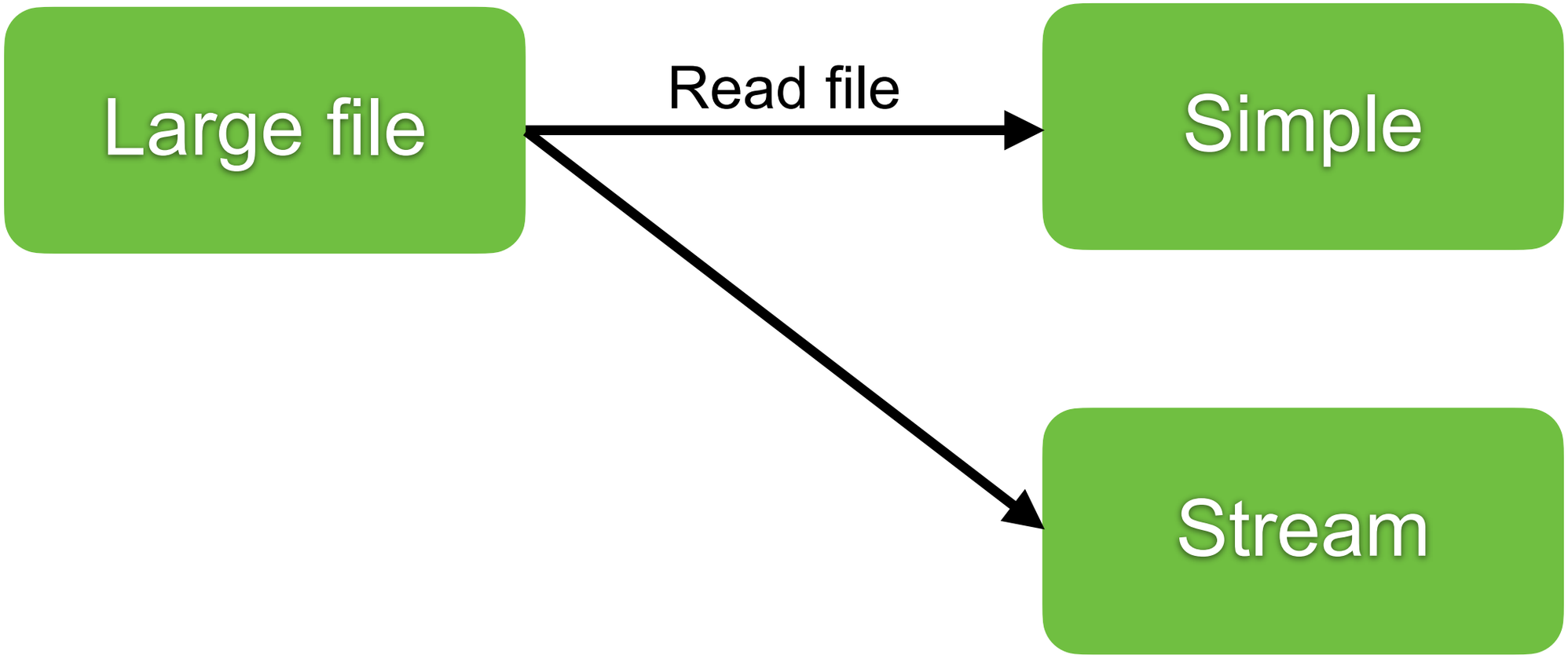


Working with Stream

```java
List<String> datas = Arrays.asList("A1", "A2", "B1", "B2", "C1");
// Imperative
for (String data : datas) {
    if(data.startsWith("A")) {
        out.println(data);
    }
}
// Declarative with Stream API
datas.stream()
        .filter( data -> data.startsWith("A") )
        .forEach(out::println);
```

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html

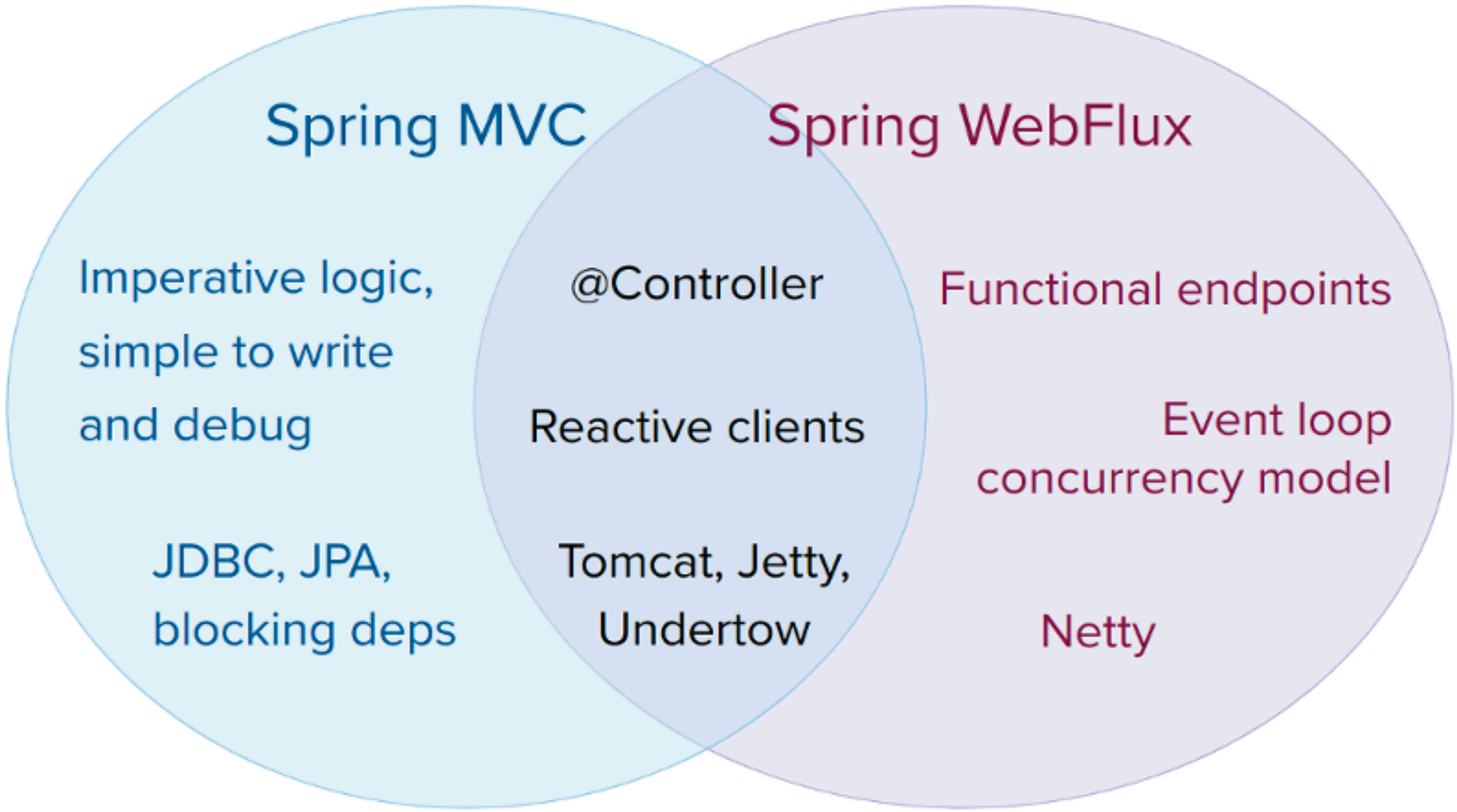
How to read a large file ?



Spring Reactive

https://spring.io/reactive

https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html#spring-webflux



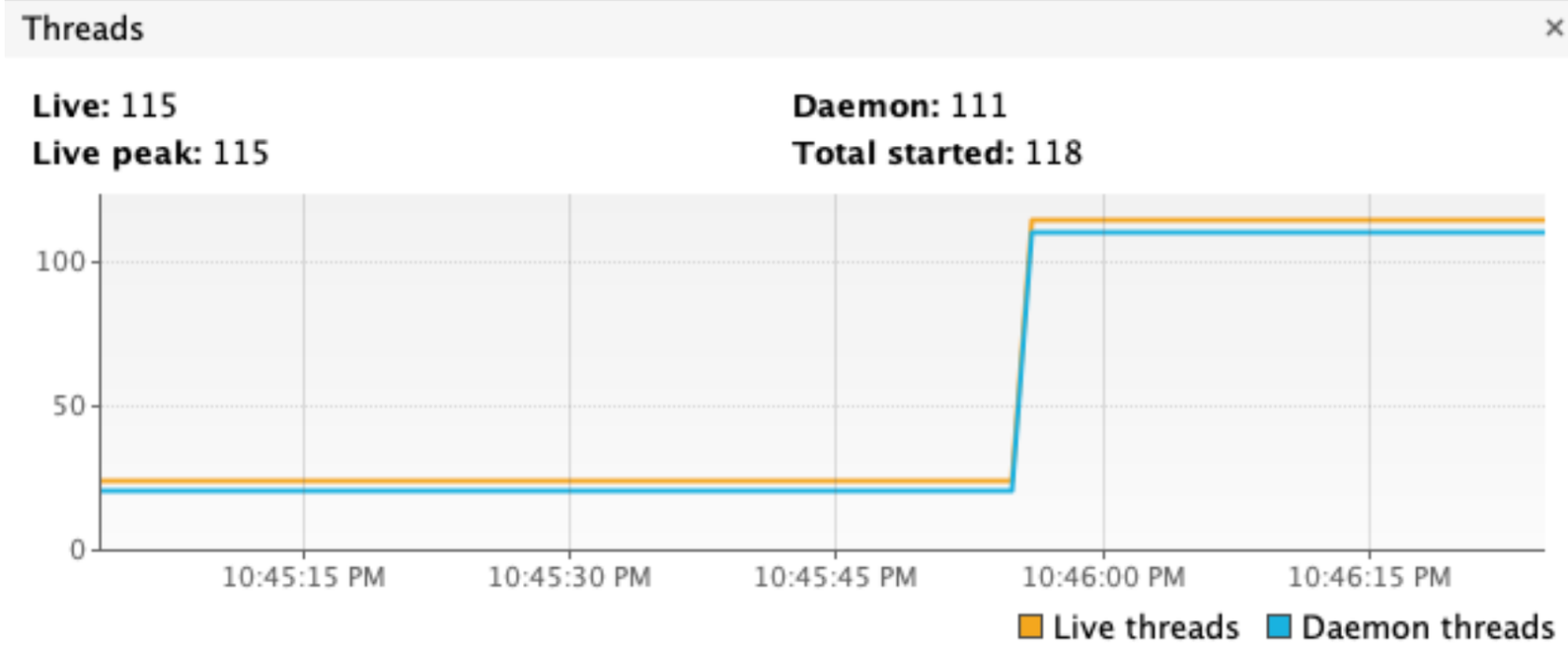
Spring MVC
Sync

Spring MVC
Async

Spring Webflux

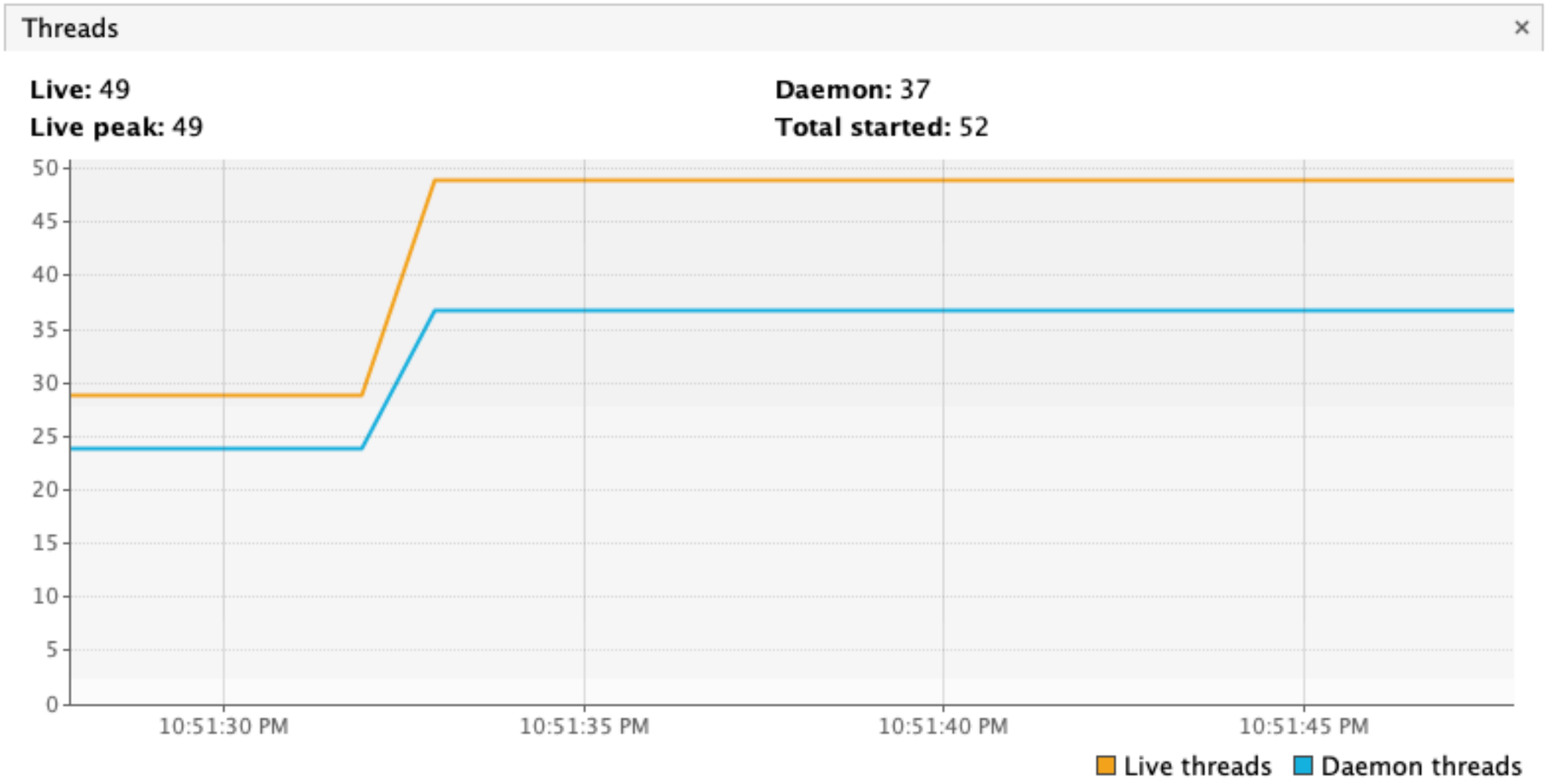
Spring MVC

$wrk -t 5 -c 100 -d 10s http://localhost:8080/demo



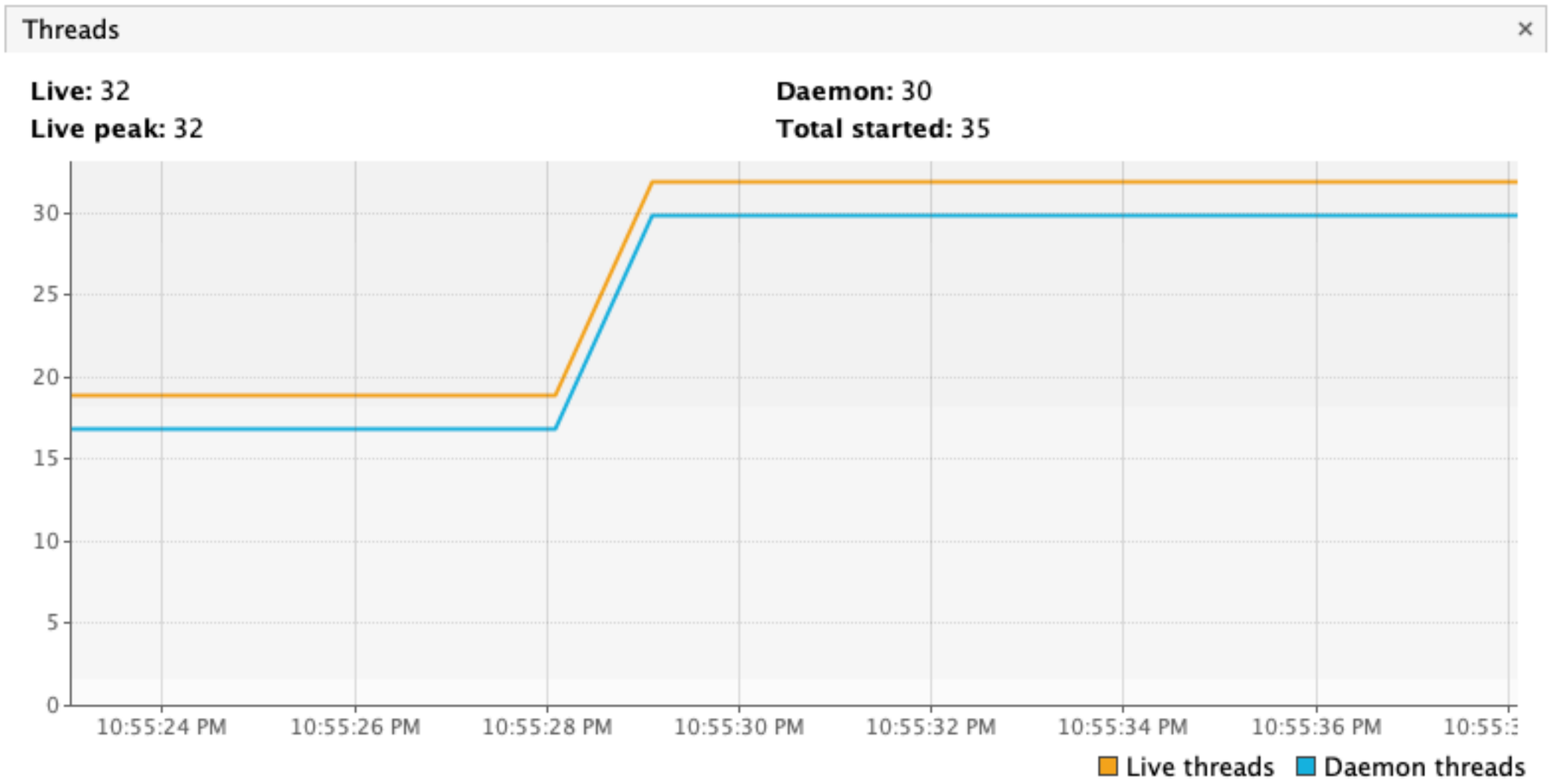
Spring MVC Async

$wrk -t 5 -c 100 -d 10s http://localhost:8080/demo



Spring Webflux

$wrk -t 5 -c 100 -d 10s http://localhost:8080/demo



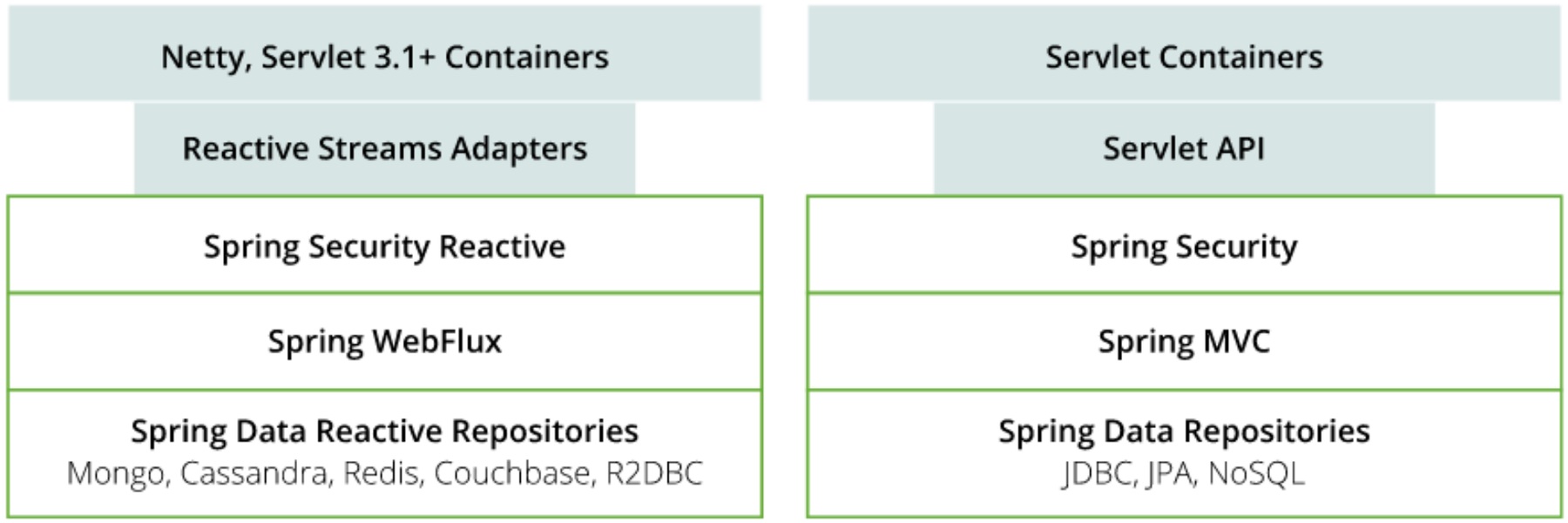
Spring Reactive

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.

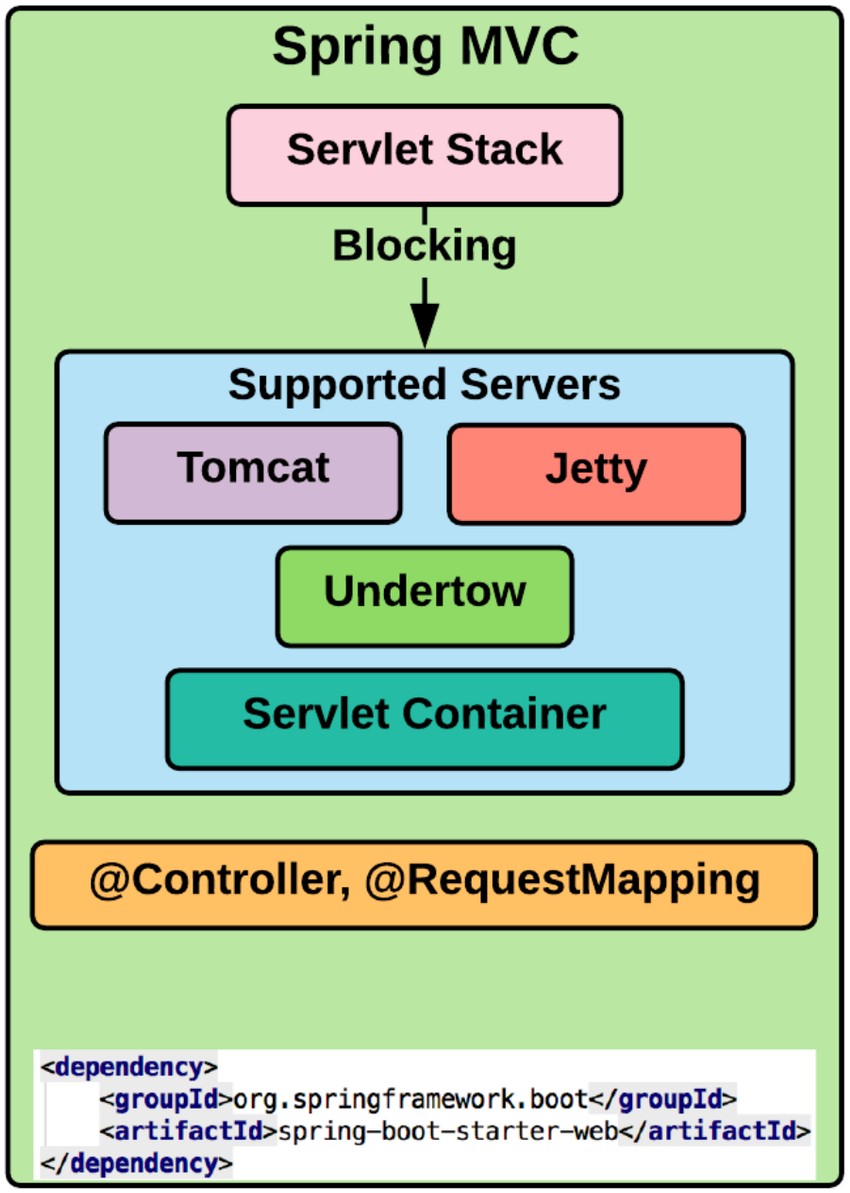


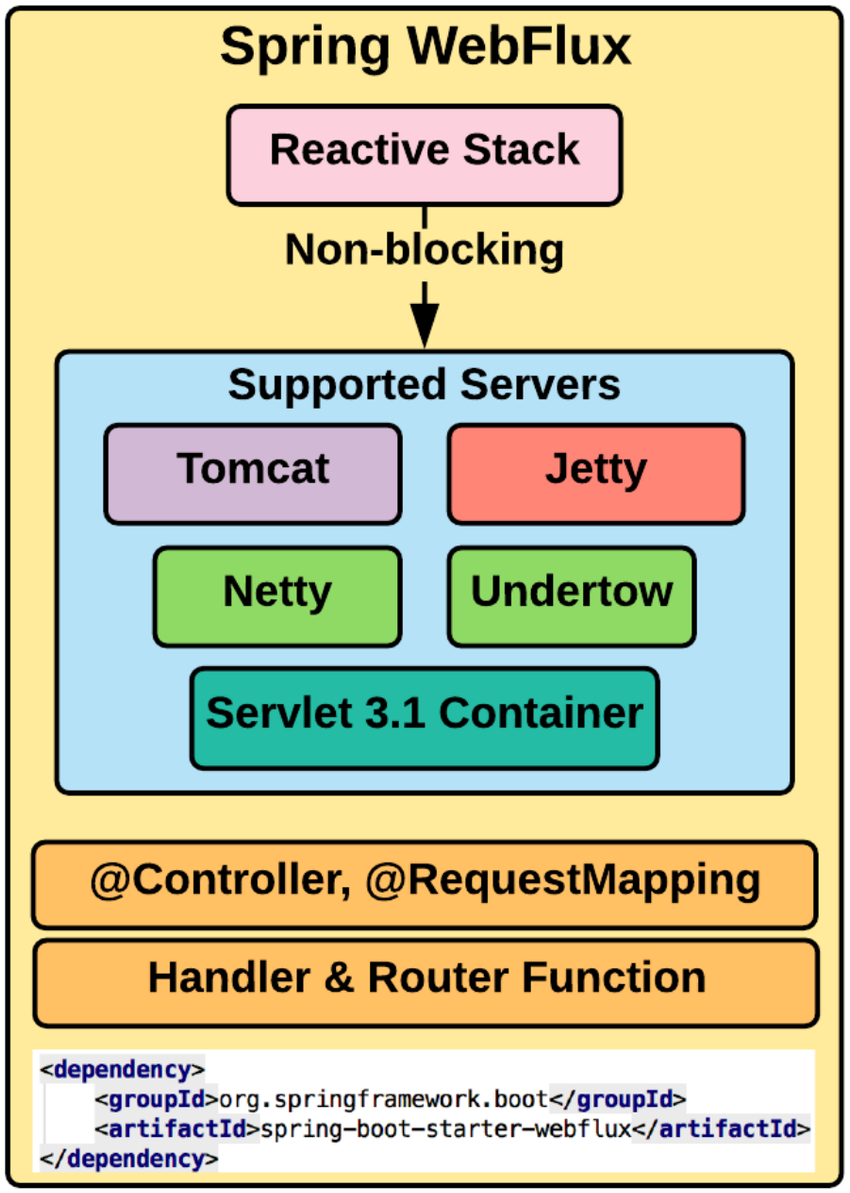
https://spring.io/reactive

Spring MVC Async
Spring WebFlux

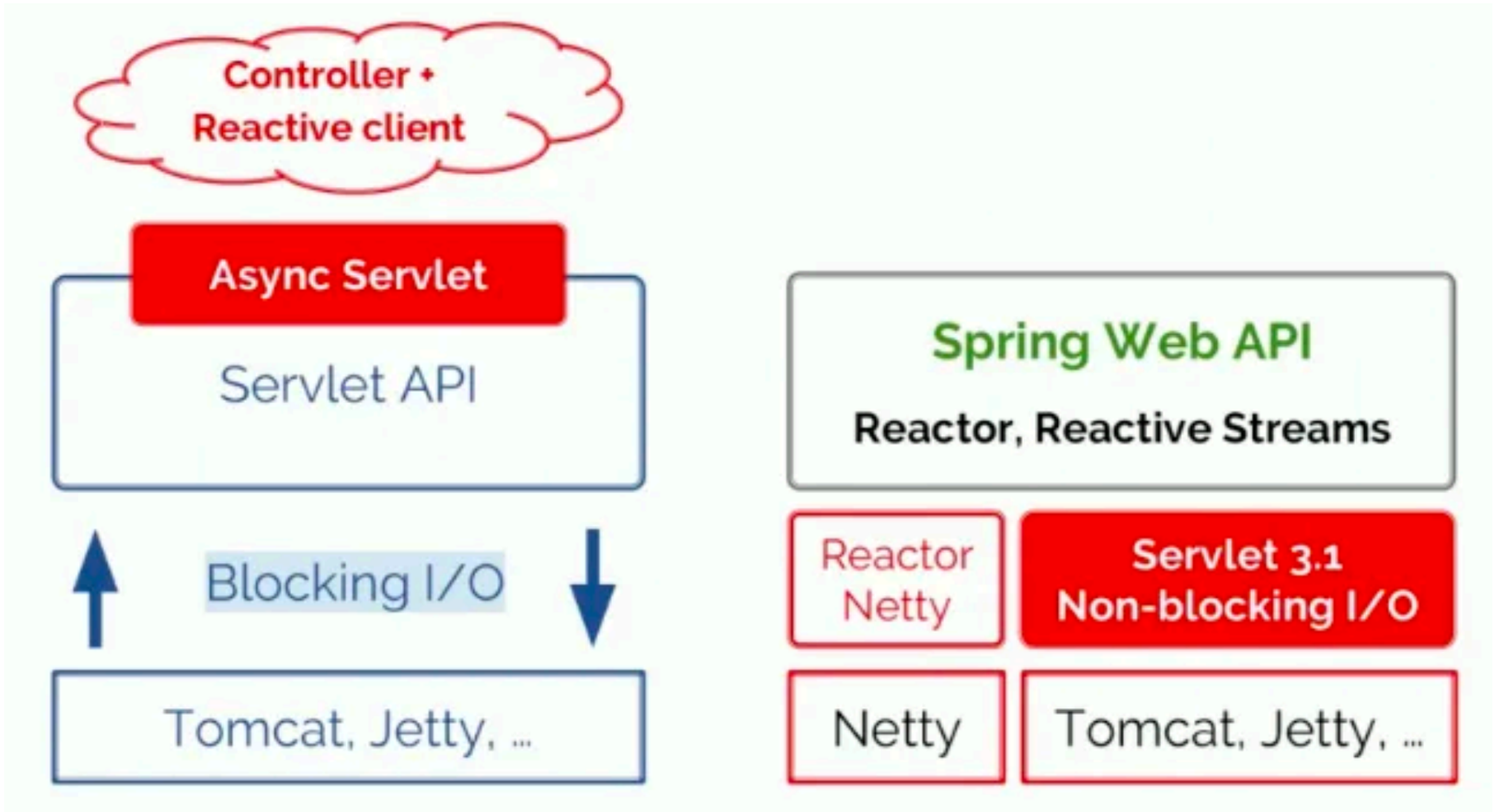
https://spring.io/reactive

Compare ...





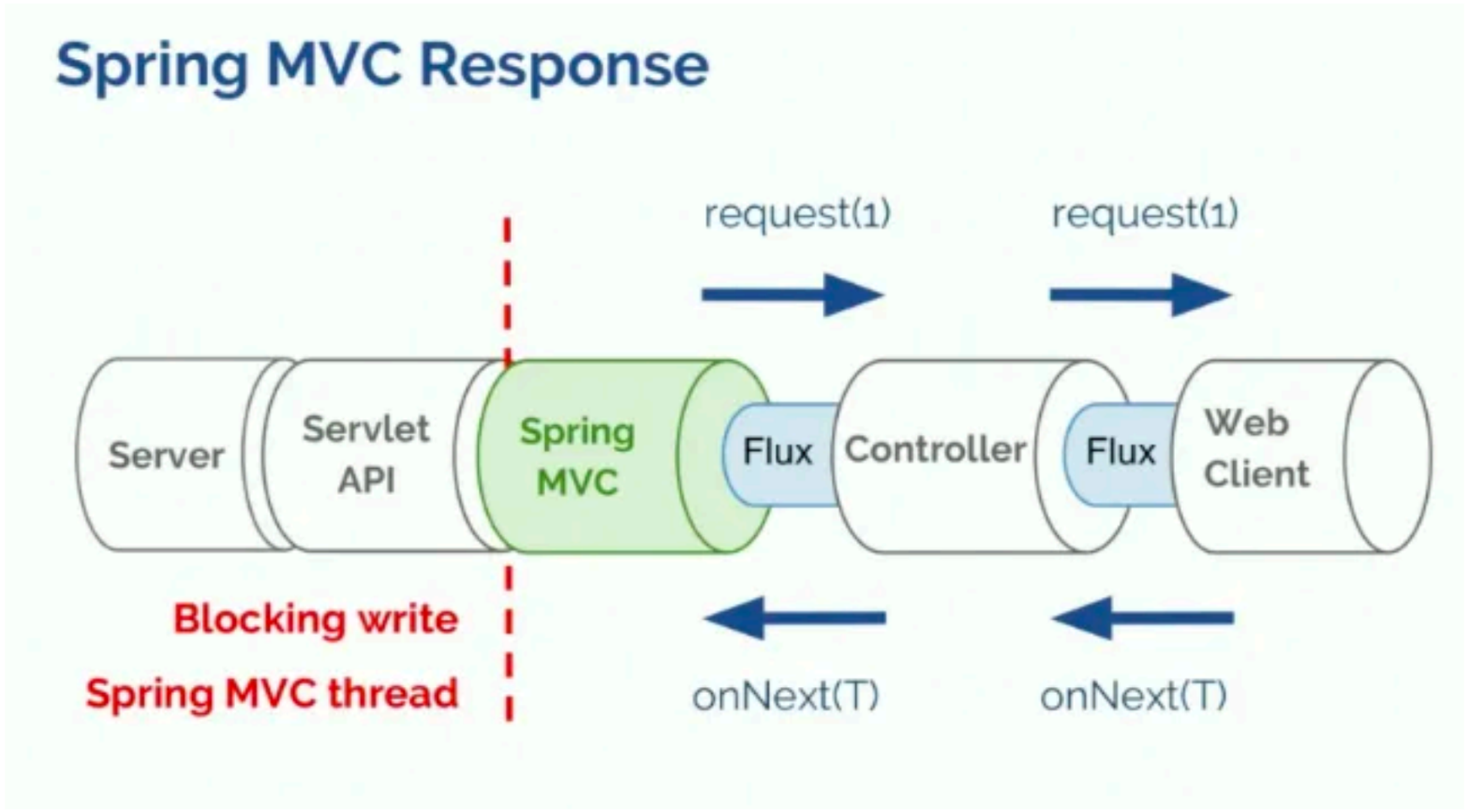
Compare ...



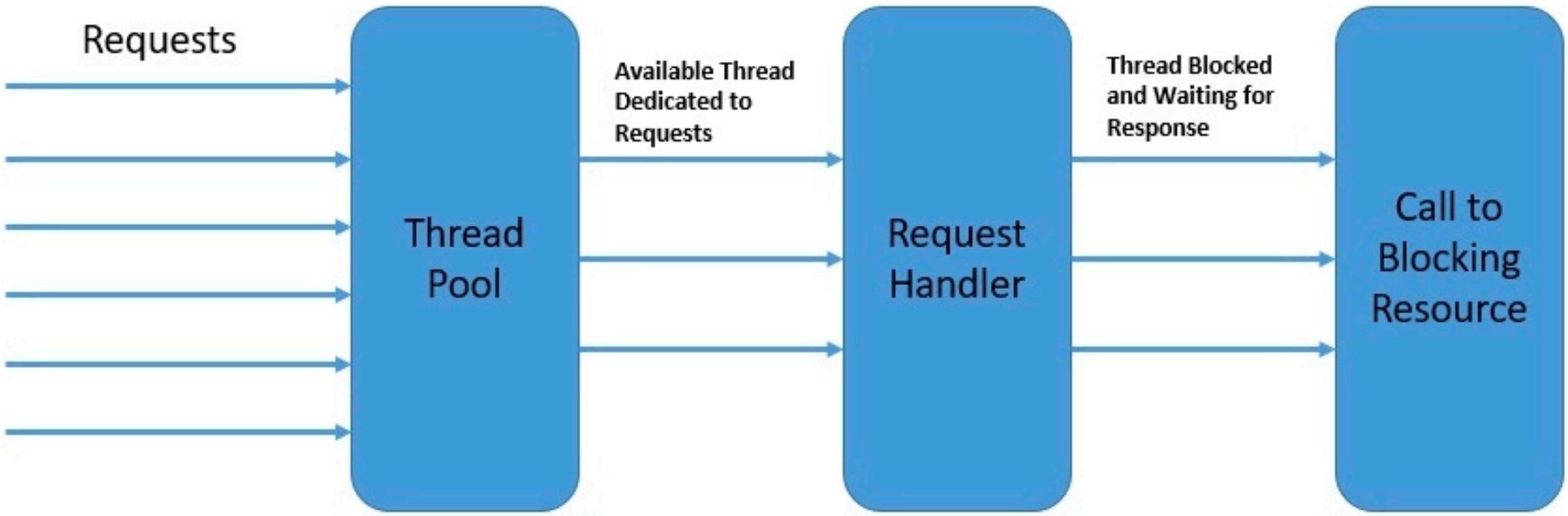
Compare ...

Spring MVC Async	Spring WebFlux
Servlet 3.0	Servlet 3.1+
Communication with the client is blocking	Non-blocking
Read the request body is blocking	Non-blocking
Filters and Servlets are working sync	Async
-	Support back-pressure
Imperative style	DecIrative and functional style

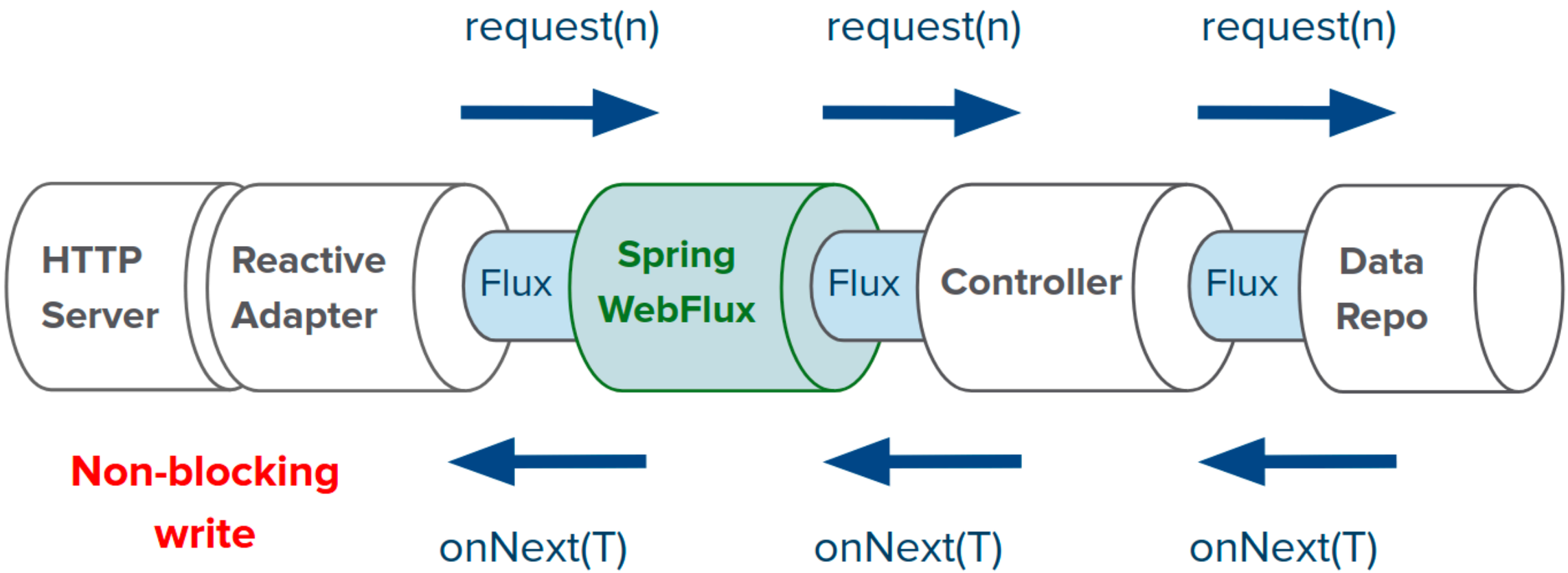
Spring MVC



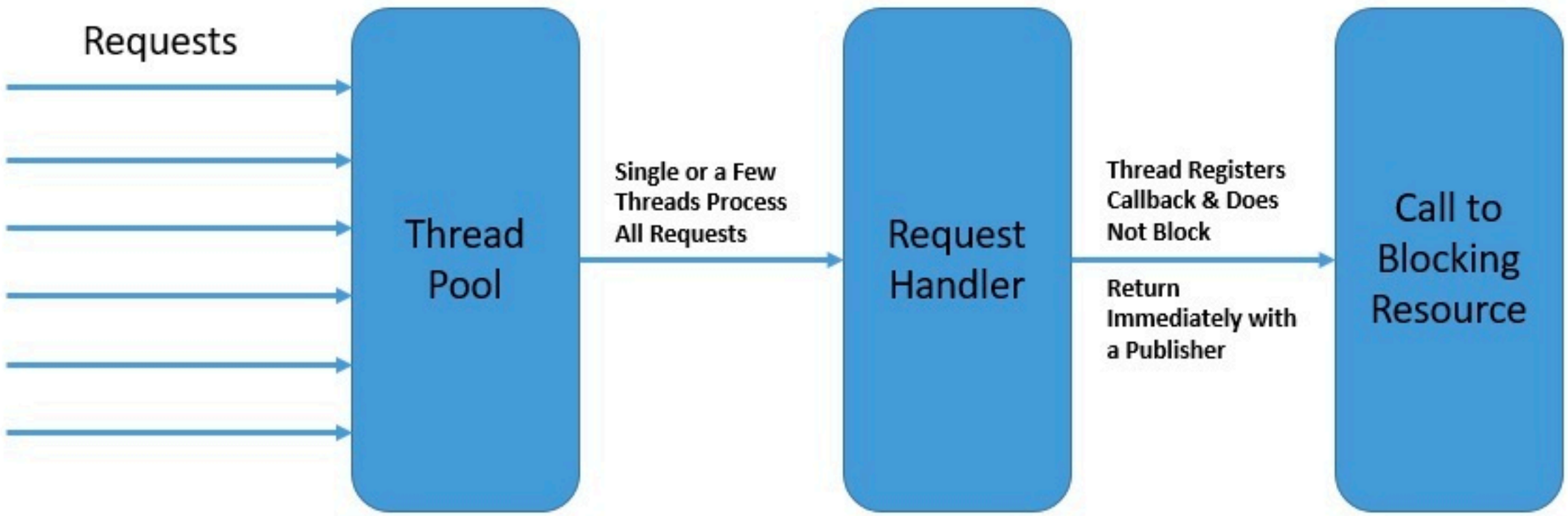
Spring MVC



Spring WebFlux

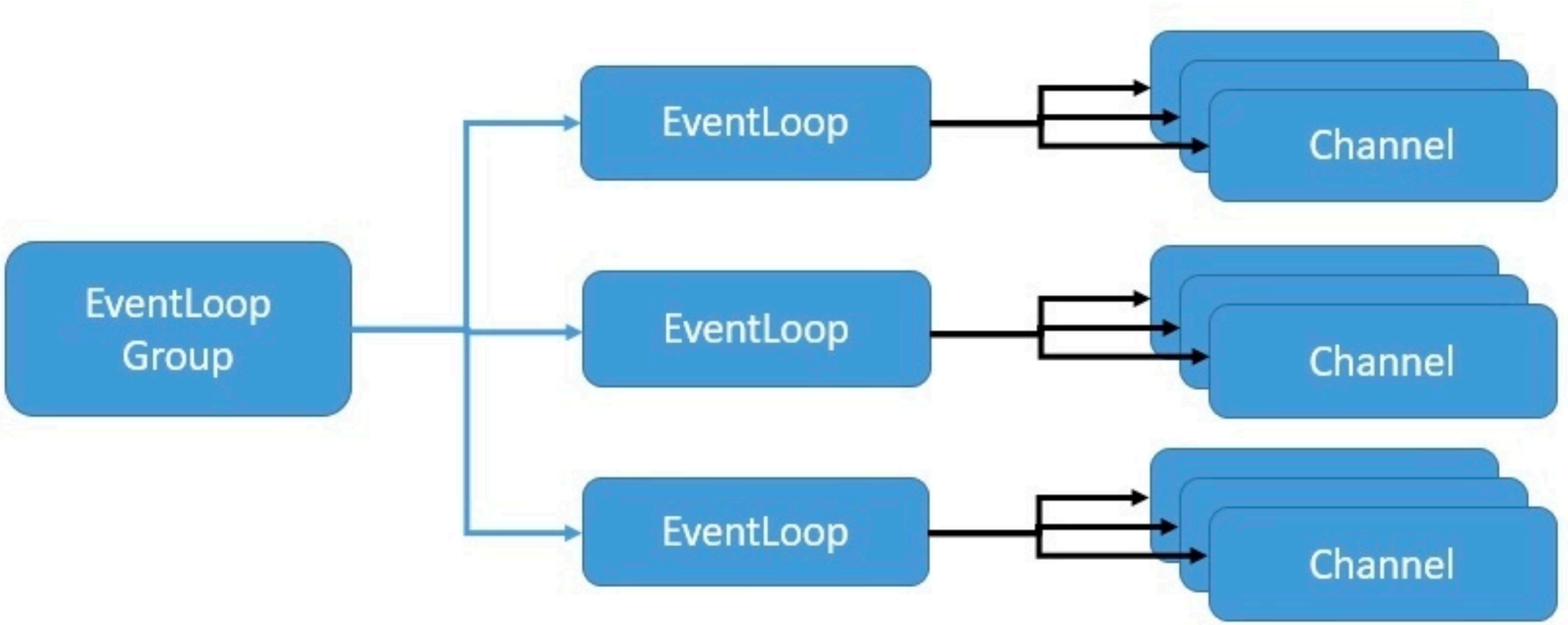


Spring WebFlux



Using Netty

Use event loop model to provide highly scalable concurrency



https://netty.io/

Choose right answer to your question/problem

Spring WebFlux

Reactive programming support for web app
Use project **Reactor**
Publisher implementations

Mono = 0 or 1 item
Flux = 0 or N items (infinity)

https://projectreactor.io/

Flux vs Mono

Mono

Handling zero(optional) or one result

Flux

Handling zero(optional) or many results (infinite)

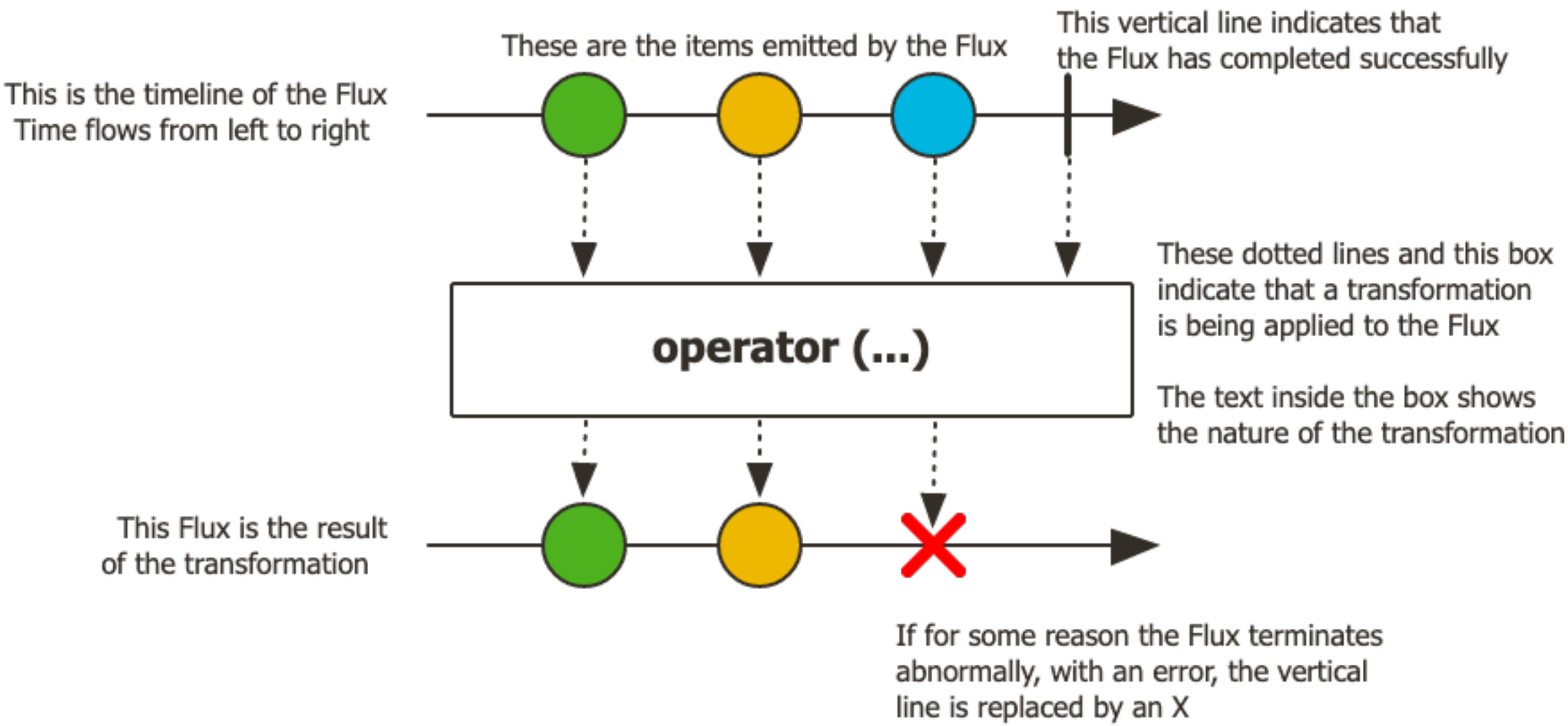
https://projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html
https://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html

Lirarary	Types
RxJS	Observable
Spring WebFlux	Mono or Flux
Java 9+	Flow

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

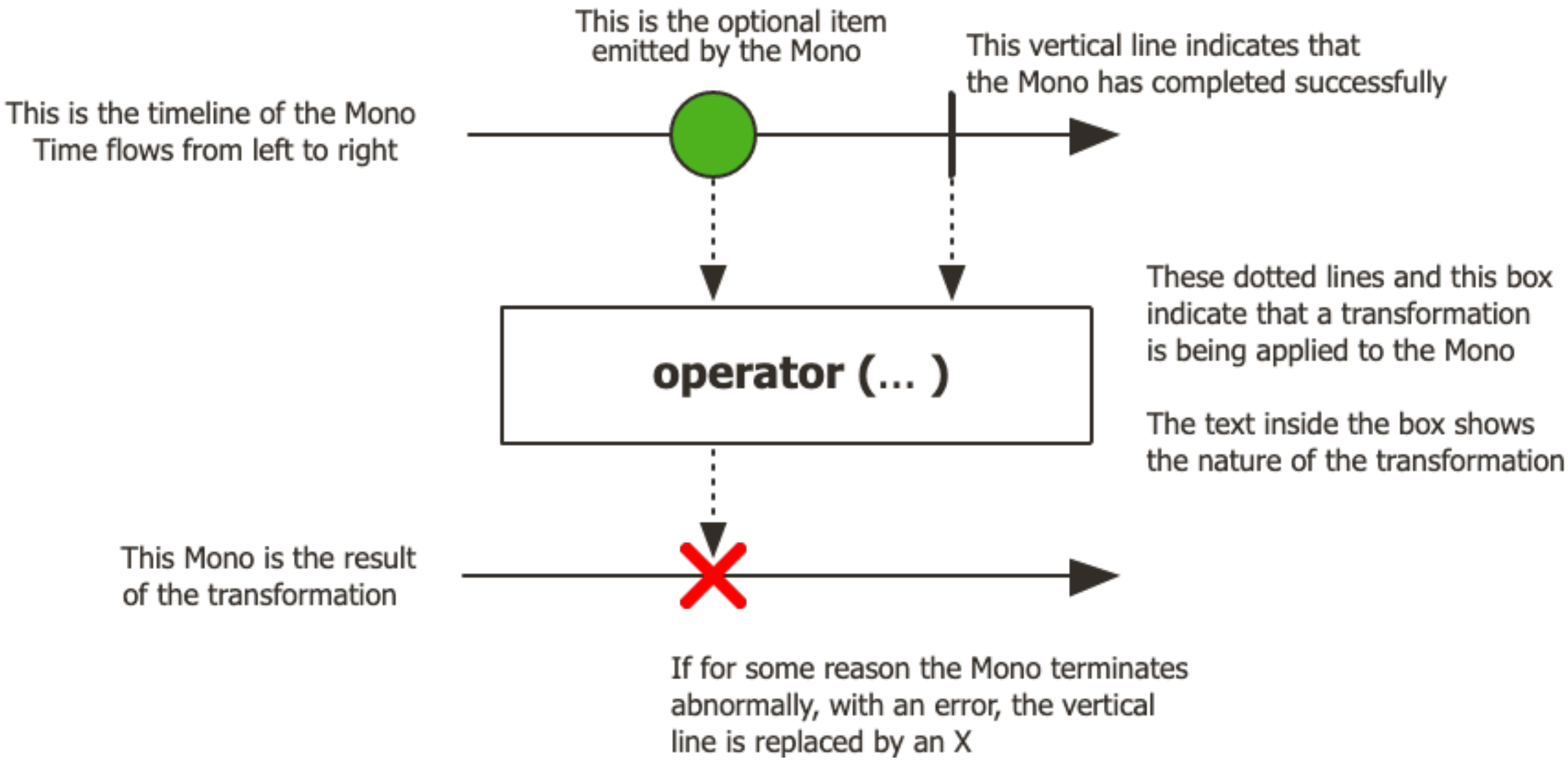
Reactor Operator

Flux :: 0-N items



https://projectreactor.io/docs/core/release/reference/index.html#flux

Mono :: 0-1 result



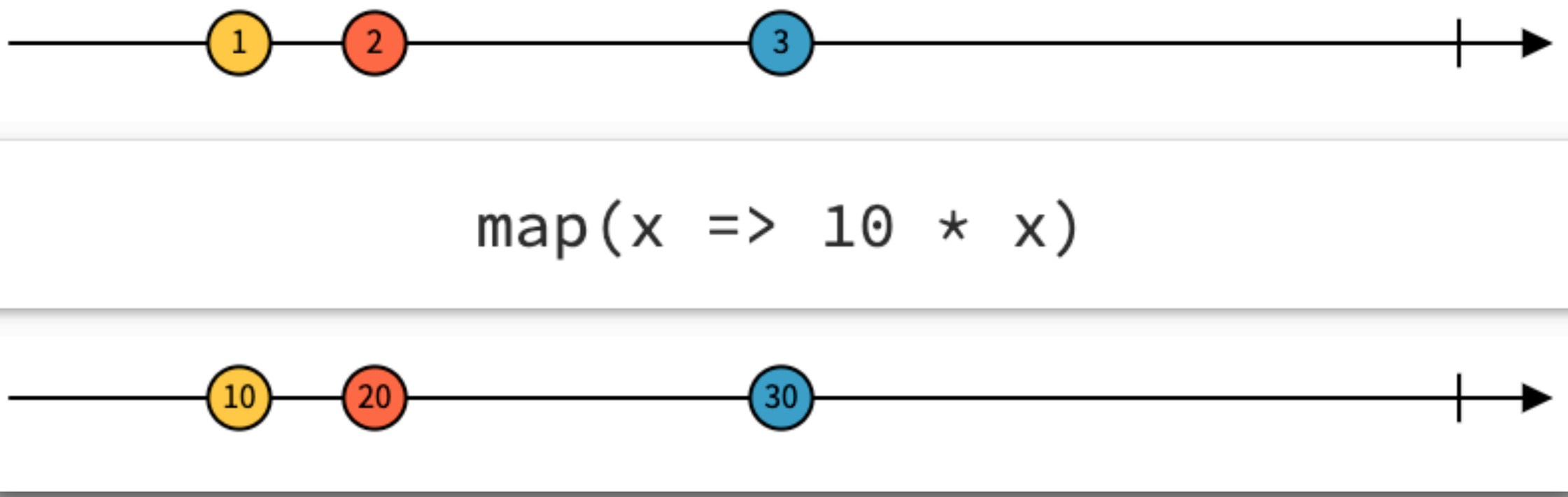
https://projectreactor.io/docs/core/release/reference/index.html#mono

Operators

Map
ZipWith
Concat
Merge

...

Map



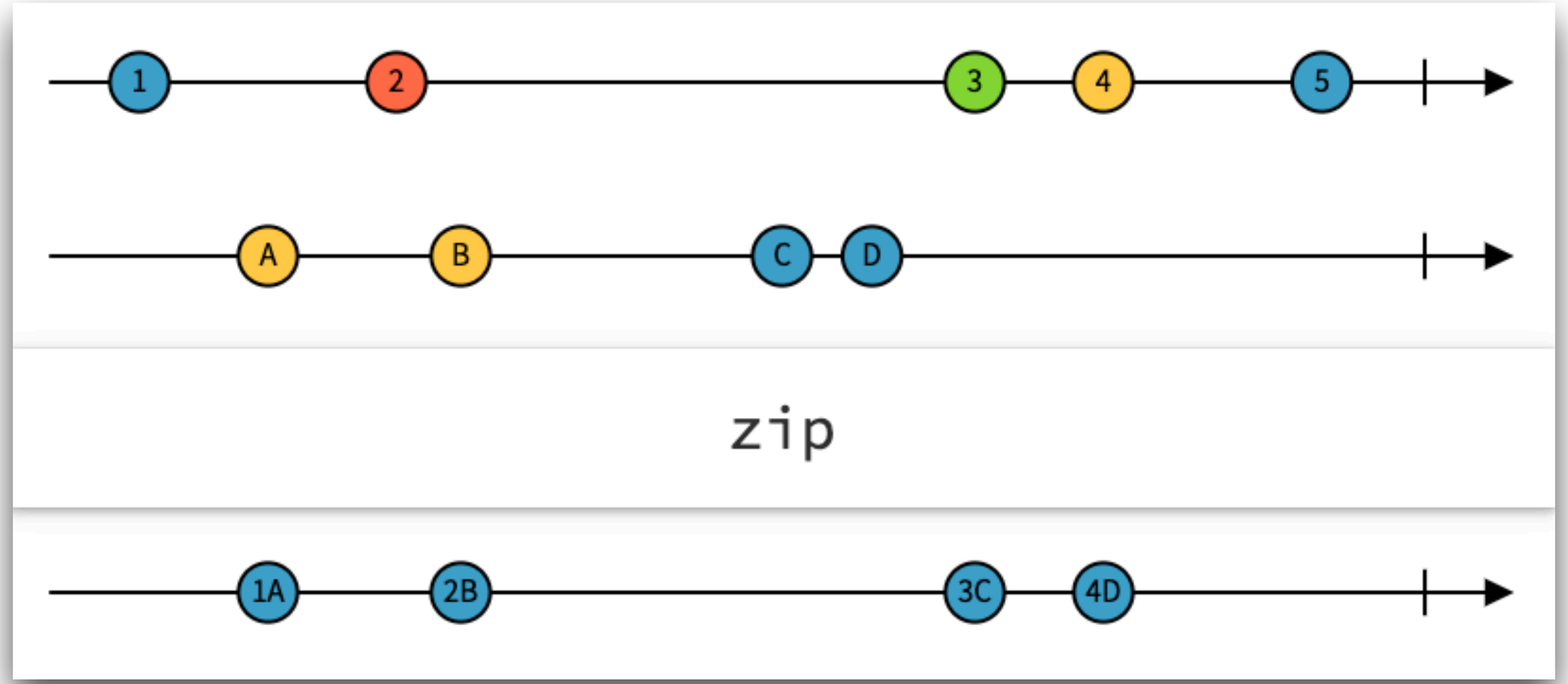
https://rxmarbles.com/#map

Map

```
Flux.just("first", "second", "third")
        .map(String::toUpperCase)
        .filter(s -> s.contains("S"))
        .subscribe(System.out::println);
```

https://rxmarbles.com/#map

ZipWith



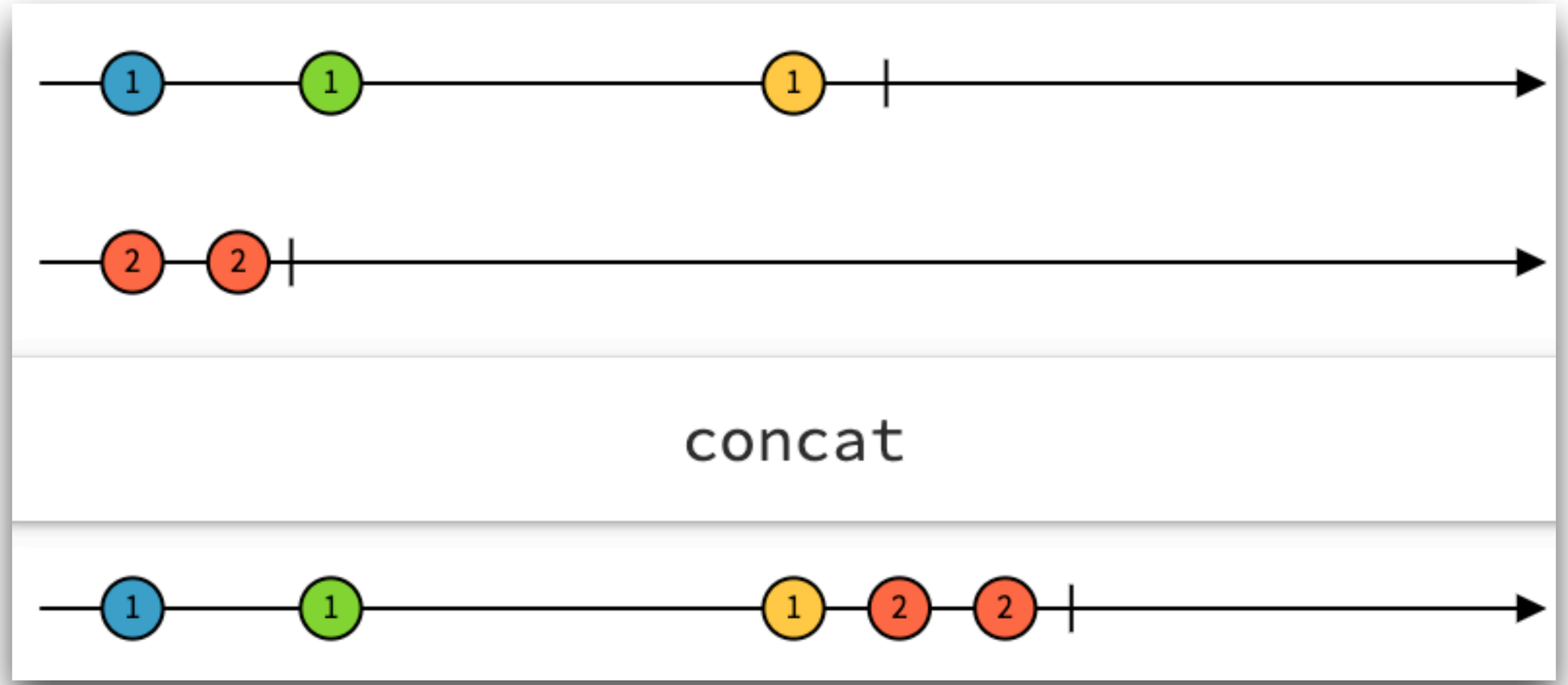
https://rxmarbles.com/#zip

ZipWith

```
var charFlux = Flux.just("a", "b", "c", "d");
var intFlux = Flux.just(1, 2, 3, 4);
intFlux.map(Object::toString)
        .zipWith(charFlux, String::concat)
        .subscribe(System.out::println);
```

https://rxmarbles.com/#zip

Concat



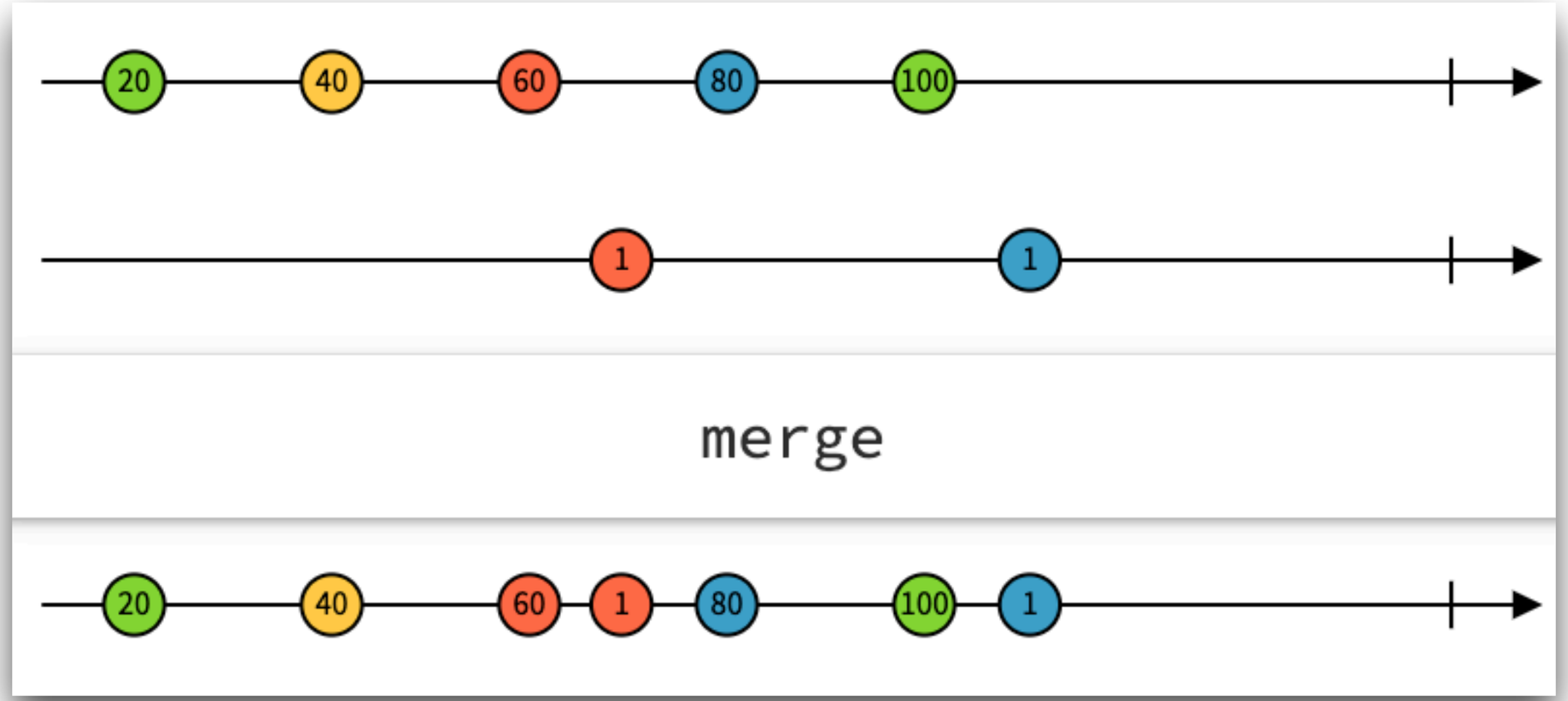
https://rxmarbles.com/#concat

Concat

```
var even = Flux.just(2, 4, 6, 8)
        .delayElements(Duration.ofMillis(100));
var odd = Flux.just(1, 3, 5, 7)
        .delayElements(Duration.ofMillis(75));
even.concatWith(odd).subscribe(System.out::print);
```

https://rxmarbles.com/#concat

Merge



https://rxmarbles.com/#merge

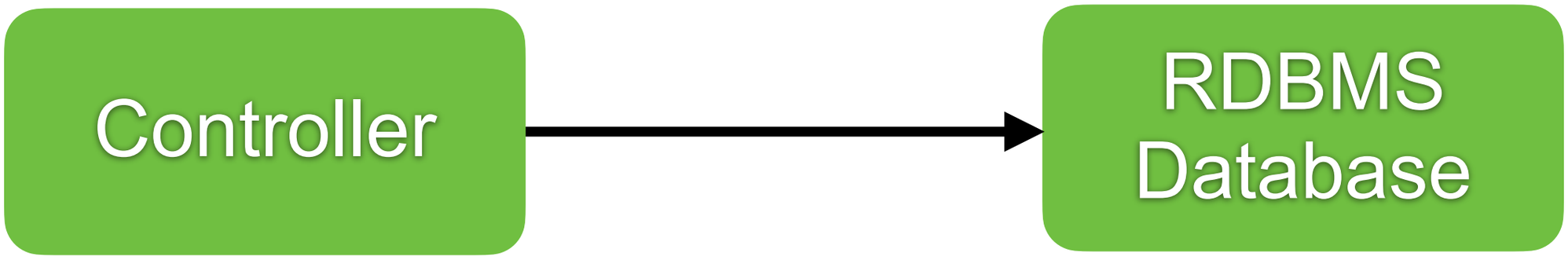
Merge

```
var even = Flux.just(2, 4, 6, 8)
        .delayElements(Duration.ofMillis(100));
var odd = Flux.just(1, 3, 5, 7)
        .delayElements(Duration.ofMillis(75));
even.mergeWith(odd).subscribe(System.out::print);
```

https://rxmarbles.com/#merge

Workshop with REST APIs

Spring WebFlux



Spring Webflux

Spring Data R2DBC

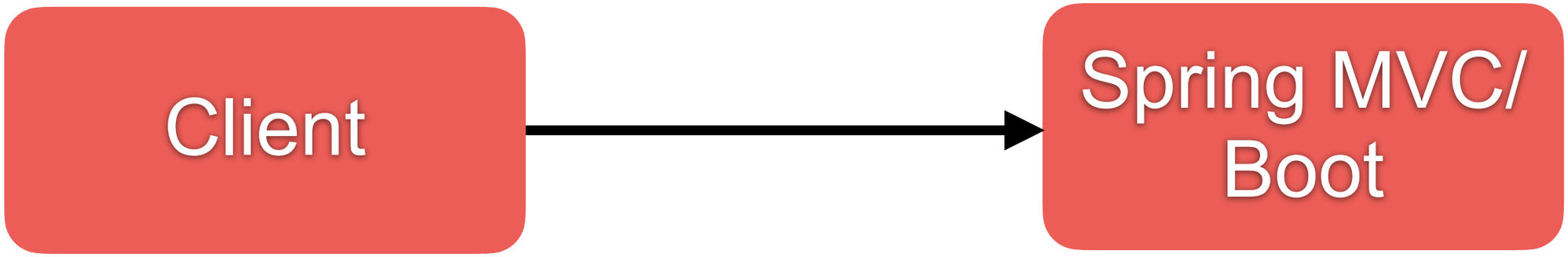
https://spring.io/projects/spring-data-r2dbc

Reactive Relational Database Connectivity

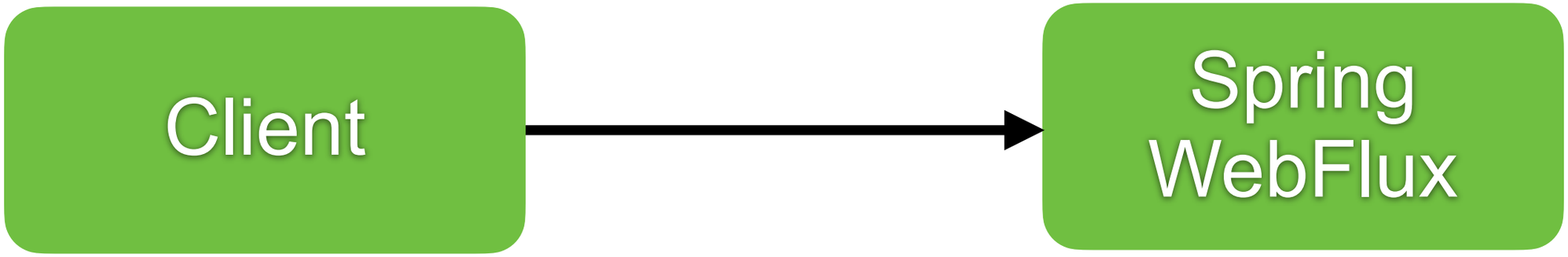
H2
MySQL, MariaDB
PostgreSQL
SQL Server
Oracle

https://spring.io/projects/spring-data-r2dbc

Spring WebFlux



RestTemplate



WebClient

https://spring.io/projects/spring-data-r2dbc

Demo
Reactive REST application

Create Project

spring initializr

Project
- Gradle - Groovy
- Gradle - Kotlin
- Maven

Language
- Java
- Kotlin
- Groovy

Spring Boot
- 3.2.0 (SNAPSHOT)
- 3.2.0 (M2)
- 3.1.4 (SNAPSHOT)
- 3.1.3
- 3.0.11 (SNAPSHOT)
- 3.0.10
- 2.7.16 (SNAPSHOT)
- 2.7.15

Project Metadata

Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo
Packaging	Jar / War
Java	20 / 17 / 11 / 8

Dependencies — ADD DEPENDENCIES... ⌘ + B

Spring Reactive Web WEB
Build reactive web applications with Spring WebFlux and Netty.

H2 Database SQL
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

https://start.spring.io/

Flux and Mono

Reactive REST

```
@GetMapping("/{id}")
public Mono<User> getUserById(@PathVariable String id) {
    return this.userRepository.findUserById(id);
}

@GetMapping
private Flux<User> getAllEmployees() {
    return this.userRepository.findAllUsers();
}
```

Testing

Testing

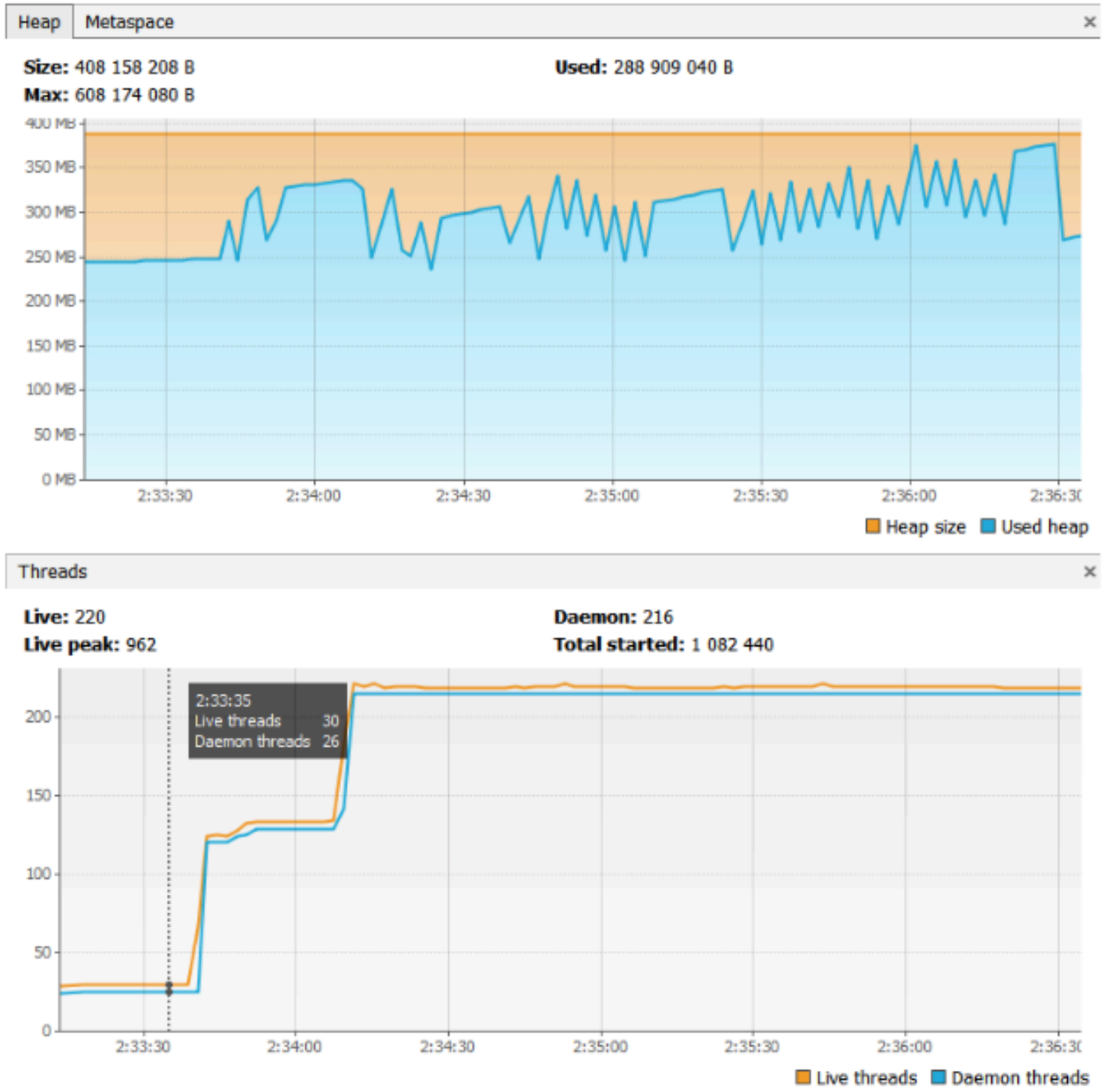
SpringBootTest + WebTestClient
WebFluxTest for Controller

https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.testing

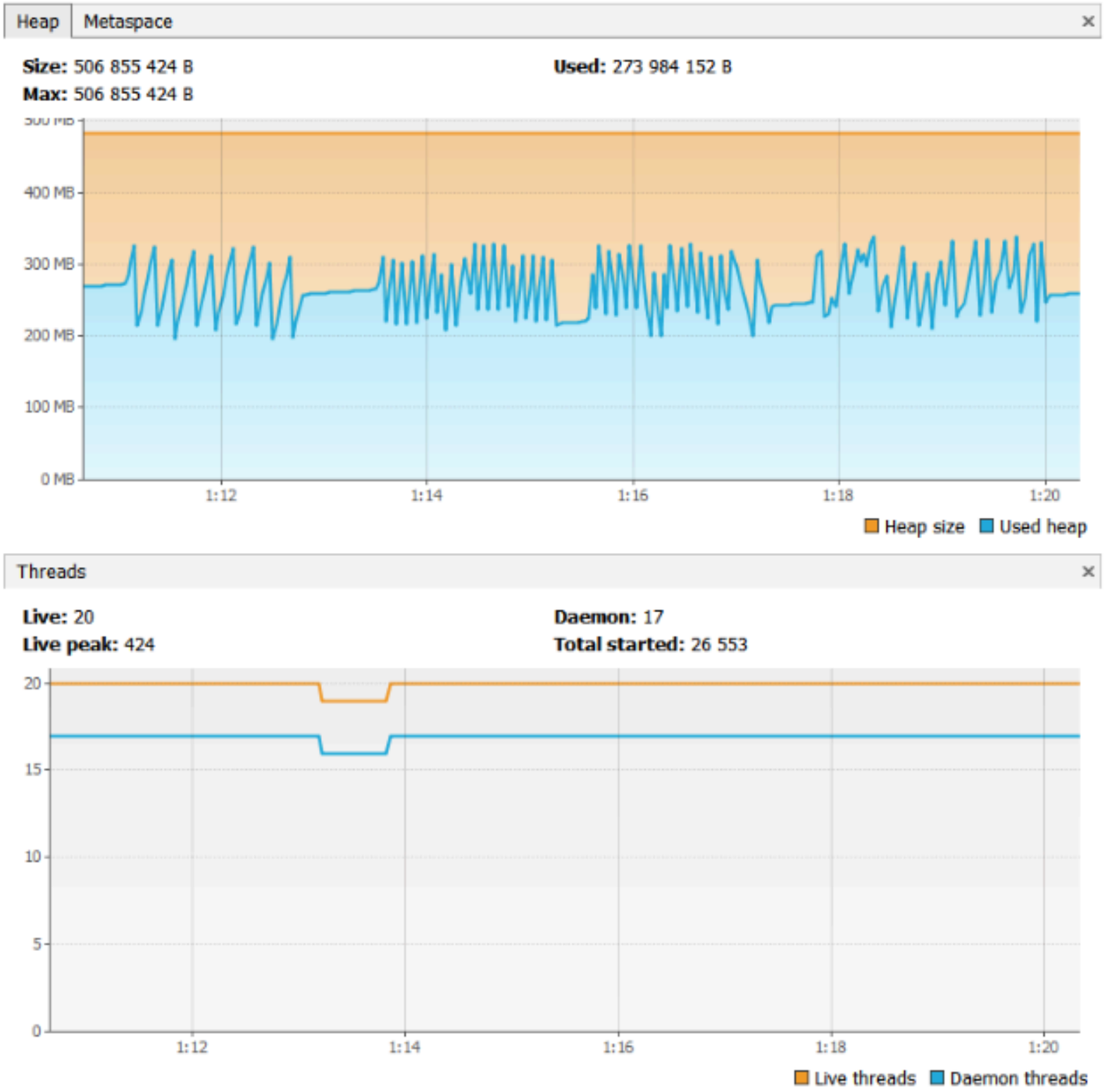
Performance testing

Profiling

Simple REST (default = 200)



WebFlux



REST vs WebFlux

Use delay = 3 seconds

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	600	451	149	25%	4.348	5	20111	44147	60001	60003	60005	28864	15541
rest	300	152	148	49%	2.174	3088	20113	45139	60001	60004	60005	30158	17380
webflux	300	299	1	0%	2.174	5	19102	44147	44159	54132	54156	27570	13327

https://gatling.io/

Q/A