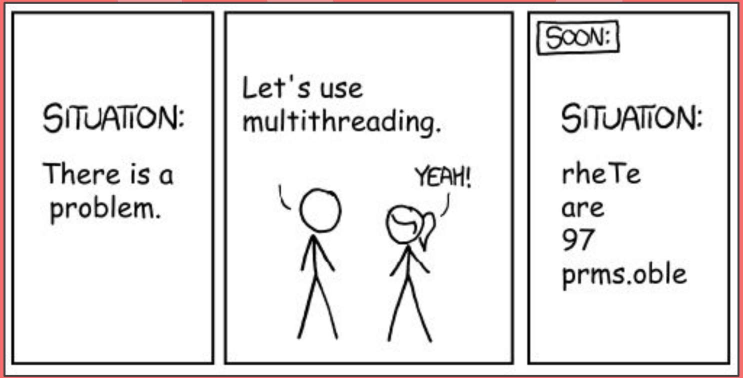
Reactive Streams

Async for the Masses



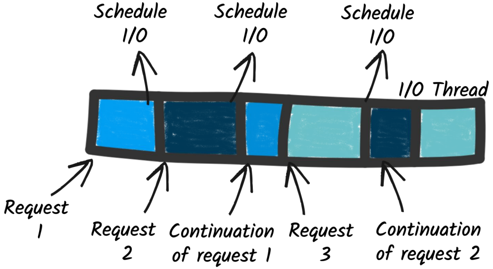
WILD
CODE
SCHOOL

- ❖ Reactive Programming, Streams and Systems
- ❖ Demo: Reactive Programming in three flavours
- ❖ Reactive Programming in Quarkus and Spring Boot
- ❖ How-to decide on imperative vs reactive programming

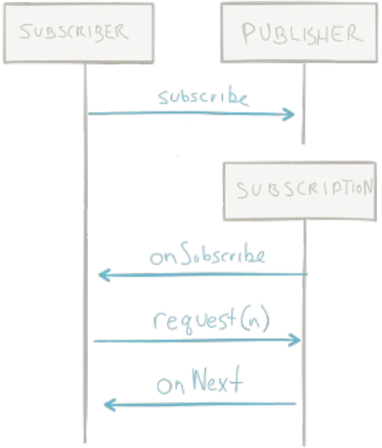
Reactive Programming, Streams & Systems (and not React!)

Reactive is a **set of design principles** to build **robust, efficient, and concurrent applications and systems**. These principles let you **handle more load** than traditional approaches while **using the resources** (CPU and memory) **more efficiently**.

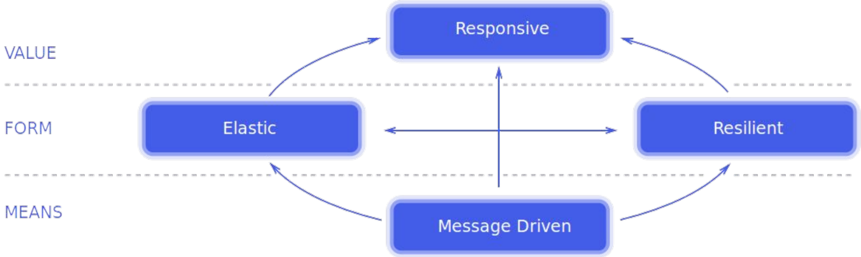
Reactive Programming



Reactive Streams



Reactive Systems



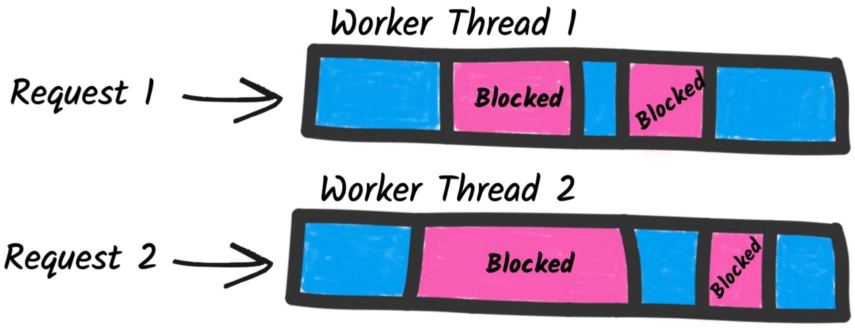
https://quarkus.io/guides/quarkus-reactive-architecture

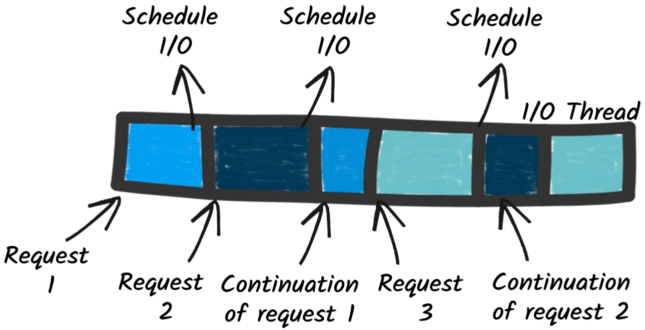
Reactive Programming - the Design Patterns

Reactive programming is programming with **asynchronous data streams.**

In technical terms, **reactive programming** is a paradigm in which declarative code is issued to construct **asynchronous processing pipelines.**

In other words, it's **programming with asynchronous data streams** that **sends** data to a **consumer** as it becomes available, which enables developers to write code that can **react to these state changes** quickly and asynchronously.

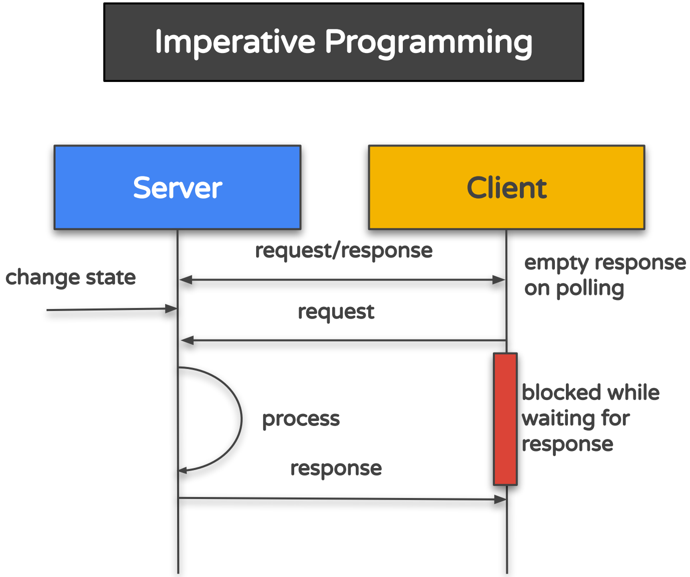


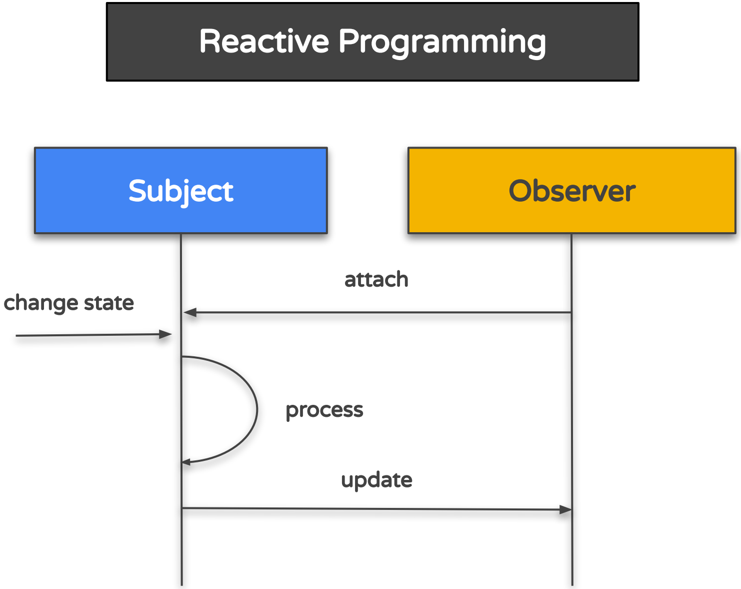


Wikipedia: Observer Pattern, Iterator Pattern
https://developer.ibm.com/articles/defining-the-term-reactive/

Reactive Programming - the Design Patterns

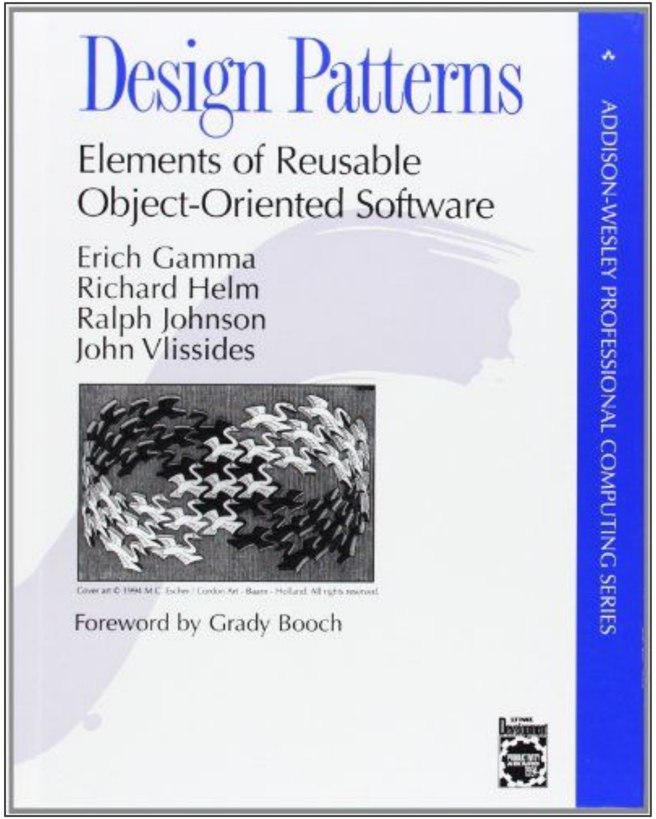
Reactive Programming combines **functional programming, the observer pattern, and the iterable pattern.**

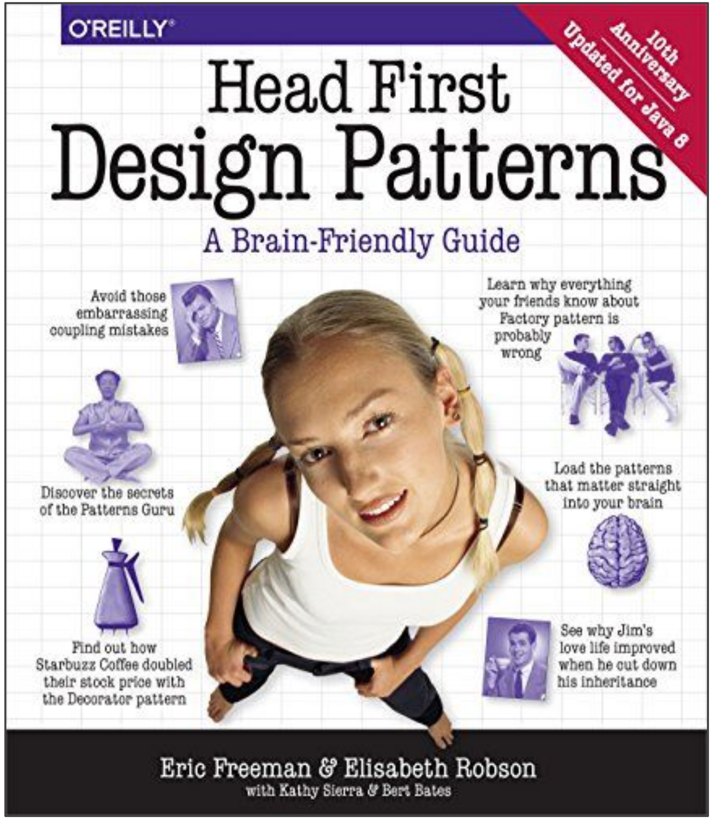




Wikipedia: Observer Pattern, Iterator Pattern
https://smallrye.io/smallrye-mutiny/pages/philosophy#events

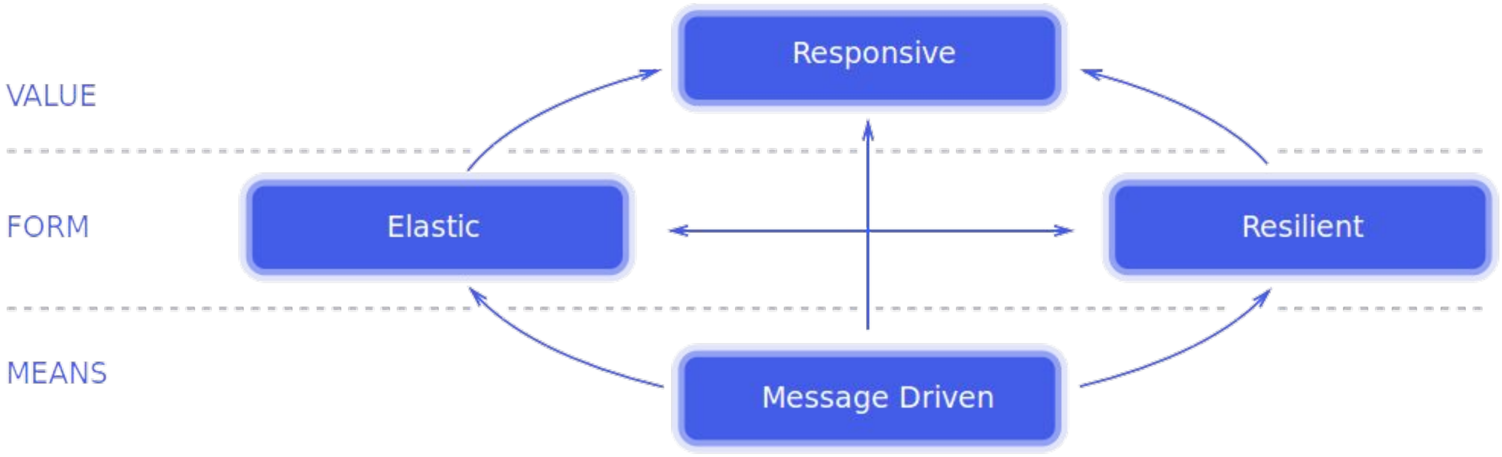
GoF Patterns: Observer & Iterator


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


O'REILLY
10th Anniversary
Updated for Java 8
Head First
Design Patterns
A Brain-Friendly Guide
Avoid those embarrassing coupling mistakes
Learn why everything your friends know about Factory pattern is probably wrong
Discover the secrets of the Patterns Guru
Load the patterns that matter straight into your brain
Find out how Starbuzz Coffee doubled their stock price with the Decorator pattern
See why Jim's love life improved when he cut down his inheritance
Eric Freeman & Elisabeth Robson
with Kathy Sierra & Bert Bates

Reactive Programming, Streams & Systems

Systems built as **Reactive Systems** are **more flexible, loosely-coupled and scalable**. This makes them **easier to develop** and **amenable to change**.



https://www.reactivemanifesto.org/

How do Reactive Programming and Systems relate?

Reactive Programming is a distinct **subset** of Reactive Systems
at the implementation level.

Reactive Programming offers productivity for Developers—through performance and resource efficiency—at the component level for **internal logic** and **dataflow management.**

It is highly beneficial to use **Reactive Programming within the components of a Reactive System.**

It is highly beneficial to **use Reactive Systems** to create the **system around the components** written **using Reactive Programming.**

https://www.lightbend.com/white-papers-and-reports/reactive-programming-versus-reactive-systems

Reactive Streams & Reactive Extensions

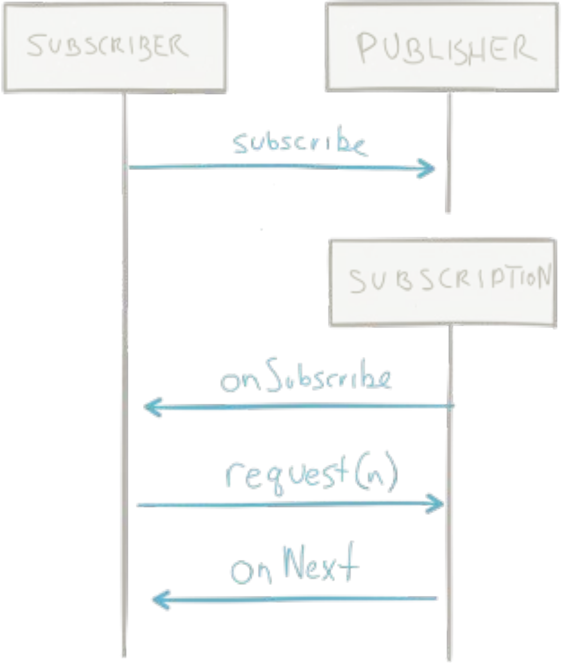
A stream is **“a sequence of data elements made available over time”**

Reactive Streams is an initiative to provide a standard for **asynchronous** stream processing **with non-blocking back pressure.**

Reactive extensions enables **imperative programming languages to compose asynchronous and event-based programs by using observable sequences.** Reactive extensions **combine the observer and iterator patterns** and functional idioms to give you a sort of toolbox, enabling your application to create, combine, merge, filter, and transform data streams.

Reactive Streams is an initiative that was created to provide a standard to **unify reactive extensions and deal with asynchronous stream processing with non-blocking backpressure,** which encompasses efforts aimed at runtime environments as well as network protocols.

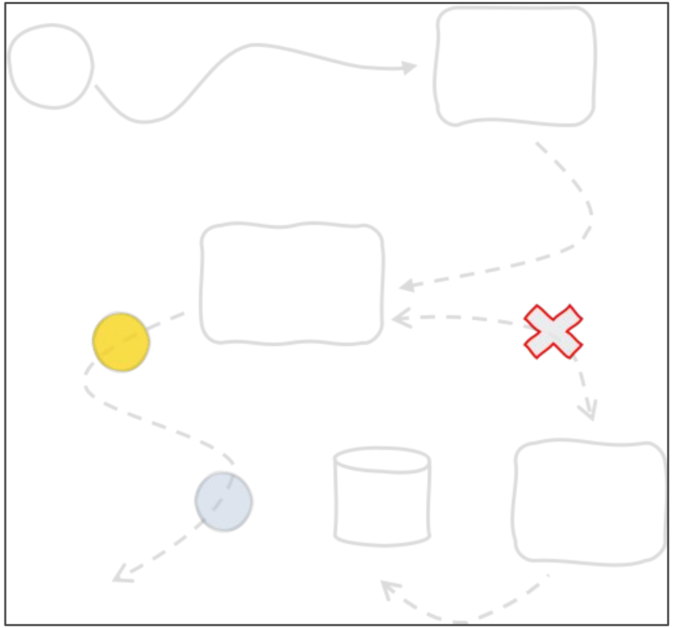
Note*: While the* ***asynchronous boundary*** *is about* ***decoupling in time,*** *what* ***Reactive Streams does not yet give us is decoupling in space — distribution.*** *This would allow us to distribute load across nodes and clusters, ideally with* ***location transparency****.*



https://blog.redelastic.com/a-journey-into-reactive-streams-5ee2a9cd7e29#.2wqcc3cja

Gotchas with Reactive Programming

Communications in distributed systems are inherently asynchronous and unreliable. Anything can go wrong, anytime, and often with no prior notice.

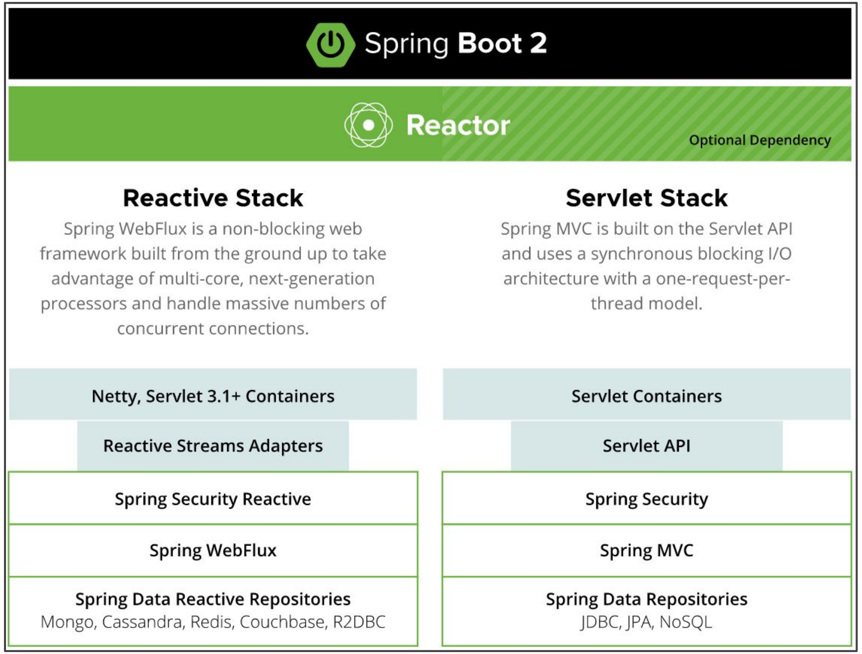


Most **classic applications** use a **synchronous development model**. Synchronous code is **easy to reason about, more comfortable to write and read than asynchronous code**, but it has some **hidden cost**. This cost emerges when building I/O intensive applications, quite common in distributed applications.

While applications using **non-blocking I/O are more efficient** and better suited for the Cloud's distributed nature, they come with a considerable constraint: **you must never block the I/O thread.** Thus, you need to implement your **business logic** using an **asynchronous development model.**

https://smallrye.io/smallrye-mutiny/pages/philosophy

Spring Boot with Project Reactor: no unified experience



Spring Boot 2

Reactor

Optional Dependency

Reactive Stack

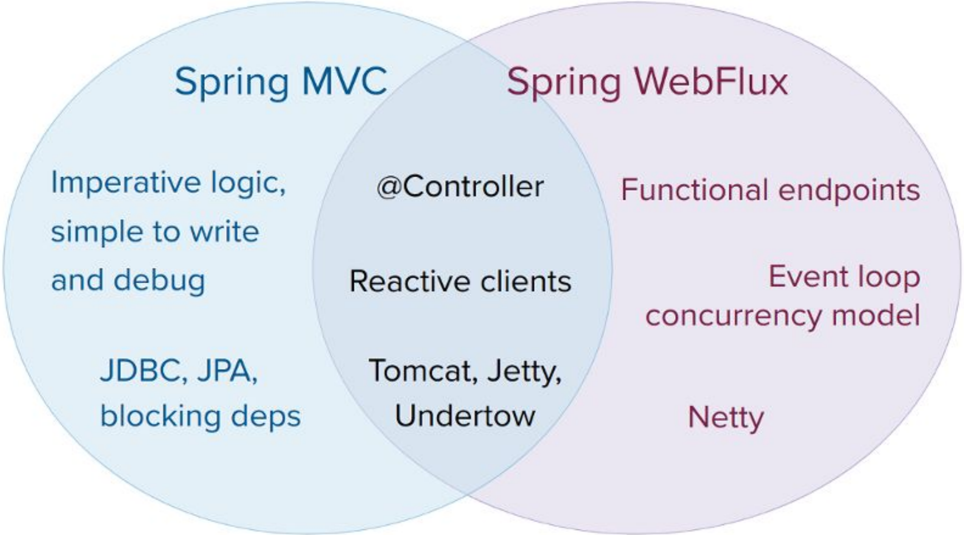
Spring WebFlux is a non-blocking web framework built from the ground up to take advantage of multi-core, next-generation processors and handle massive numbers of concurrent connections.

- Netty, Servlet 3.1+ Containers
- Reactive Streams Adapters
- Spring Security Reactive
- Spring WebFlux
- Spring Data Reactive Repositories
  Mongo, Cassandra, Redis, Couchbase, R2DBC

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.

- Servlet Containers
- Servlet API
- Spring Security
- Spring MVC
- Spring Data Repositories
  JDBC, JPA, NoSQL



Spring MVC

Imperative logic, simple to write and debug

JDBC, JPA, blocking deps

Spring WebFlux

Functional endpoints

Event loop concurrency model

Netty

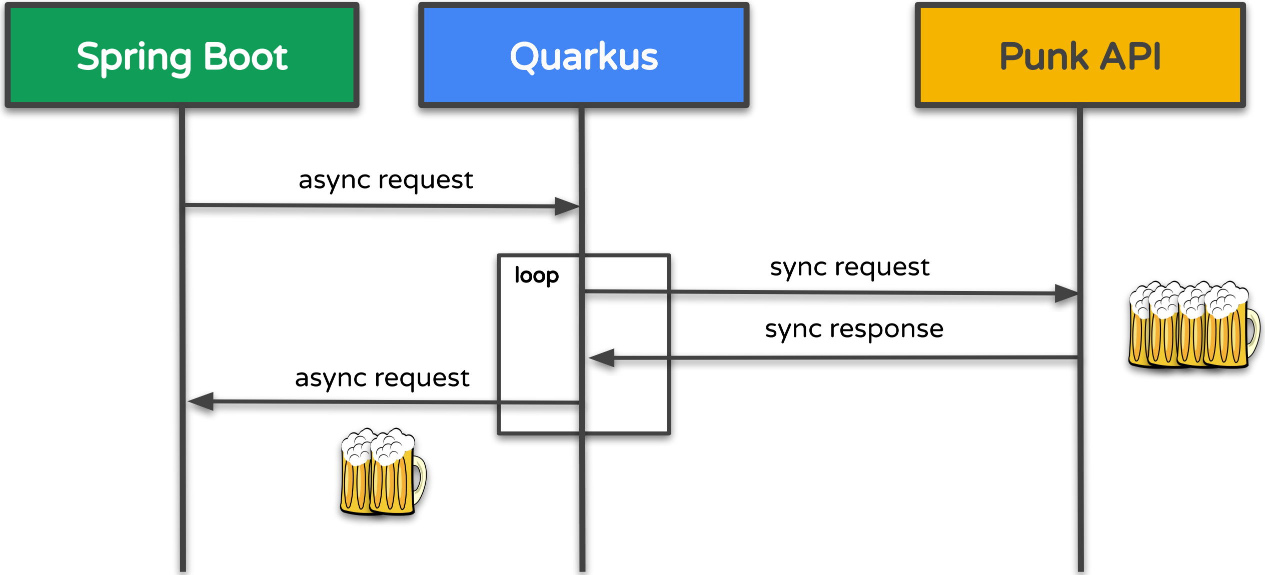
@Controller

Reactive clients

Tomcat, Jetty, Undertow

https://spring.io/reactive

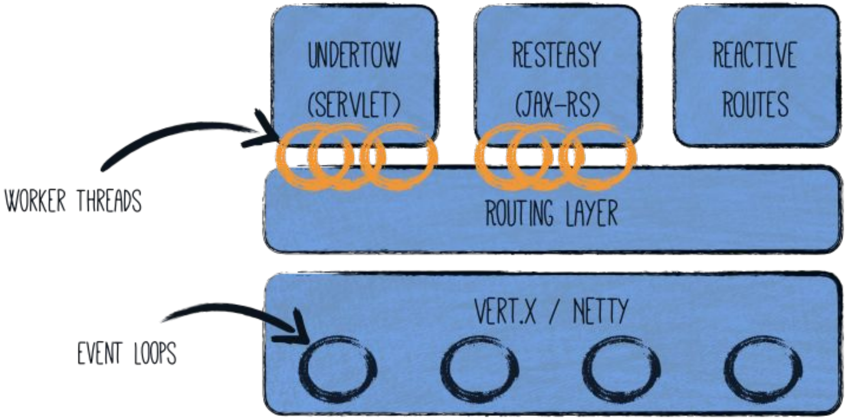
Microservices Compatibility: Spring Boot Beer Consumer



Quarkus Reactive Service **pulls** beer from Punk API **synchronously** and **pushes asynchronously** to Spring Reactive Consumer.

What is special about Quarkus and Reactive Programming?

Quarkus is a **Reactive framework**. Since the beginning, Reactive has been an **essential tenet of the Quarkus architecture**. It includes **many reactive features** and offers a broad ecosystem.
This is what Quarkus is about: **unifying reactive and imperative in a single runtime**.

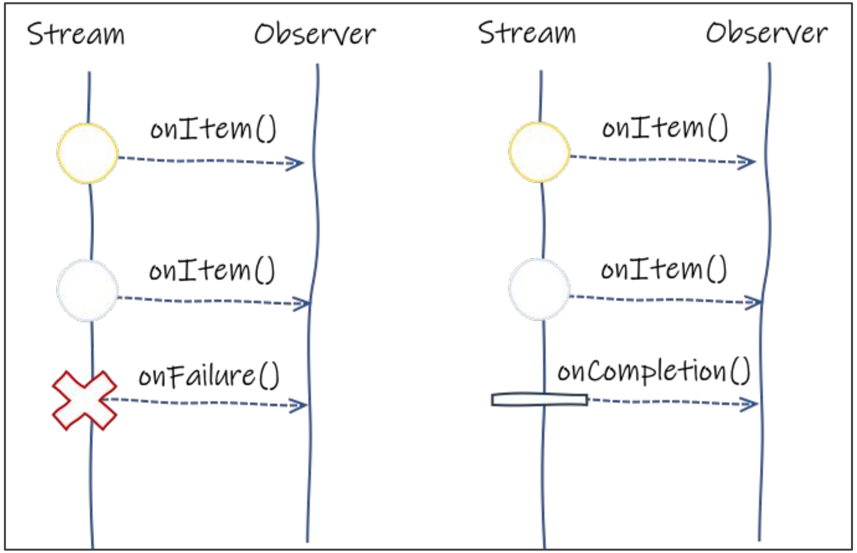


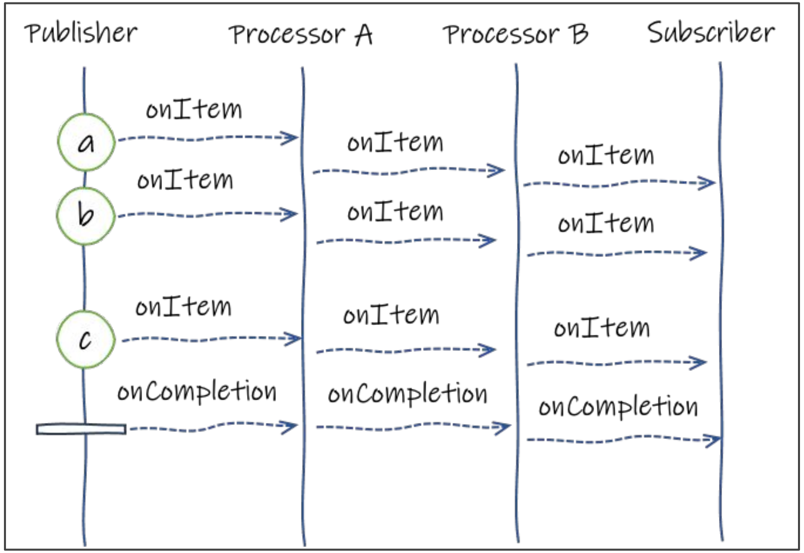
Quarkus HTTP support is based on a non-blocking and reactive engine (Eclipse Vert.x and Netty). **All the HTTP requests your application receive are handled by event loops (IO Thread) and then are routed towards the code that manages the request.** Depending on the destination, it can invoke the code managing the request on a **worker thread (servlet, Jax-RS)** or use the **IO thread (reactive route)**.

https://developers.redhat.com/blog/2019/11/18/how-quarkus-brings-imperative-and-reactive-programming-together#enter_quarkus

Unis and Multis (Reactive Streams with Mutiny)

A Uni represents a *stream* that can only **emit** either **an item** or a **failure event**. You rarely create instances of Uni yourself, but, instead, **use a reactive client exposing a Mutiny API** that provides Unis. Quarkus with its unified (imperative & reactive) programming model exposes the Mutiny API.





https://smallrye.io/smallrye-mutiny/getting-started/creating-unis

Links and other information

Reactive Design: System, Streams & Programming

- **What is Reactive Design (Systems, Streams & Programming):** https://www.lightbend.com/white-papers-and-reports/reactive-programming-versus-reactive-systems
- **Reactive Manifesto:** https://www.reactivemanifesto.org/
- **Reactive Future in 2051:** https://paulstovell.com/reactive-programming/
- **IBM on Reactive Definitions:** https://developer.ibm.com/articles/defining-the-term-reactive/

Reactive Streams & Reactive Extensions

- **Reactive Streams Spec:** https://www.reactive-streams.org/

Quarkus Reactive Concepts

- **Getting Started with Reactive:** https://quarkus.io/guides/getting-started-reactive
- **Quarkus Reactive Architecture:** https://quarkus.io/guides/quarkus-reactive-architecture
- **Smart Dispatch:** https://quarkus.io/blog/resteasy-reactive-smart-dispatch/

Links and other information

Tutorial

- **Getting Started (RESTEasy & Panache):** https://quarkus.io/guides/getting-started-reactive
- **Reactive Beer:** https://redhat-developer-demos.github.io/quarkus-tutorial/quarkus-tutorial/reactive.html