Modern Reactive Java

Julien Ponge, Red Hat

Ingolstadt JUG January 2021

In this session...

Reactive, and why it matters

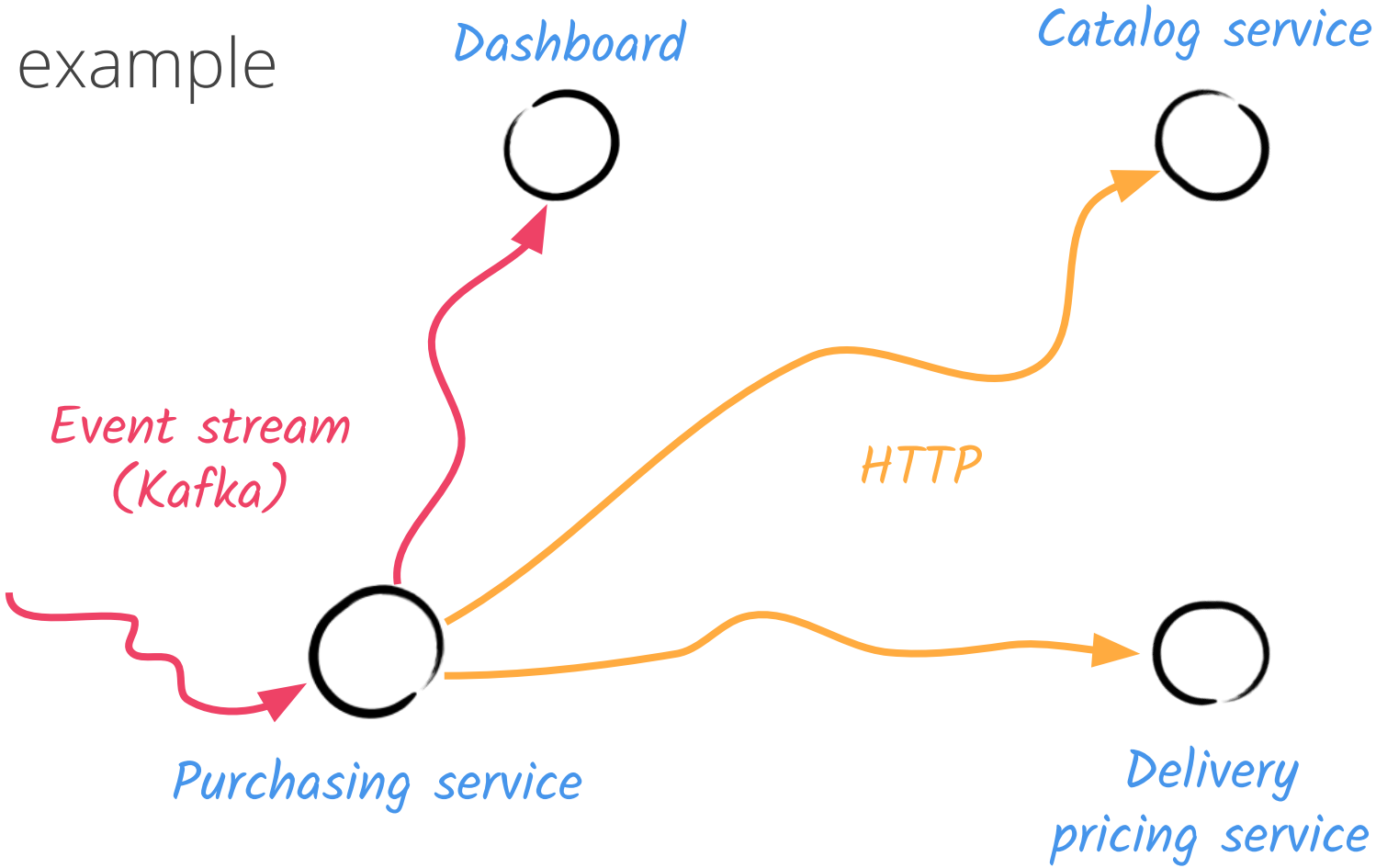
Vert.x: a toolkit for writing reactive applications

Mutiny: intuitive event-driven reactive programming library for Java

Mutiny and Vert.x inside **Quarkus**

(Credits to Clément Escoffier for some of the content)

Running example



https://github.com/jponge/ingolstadt-jug-january-2021

Why does reactive matter?

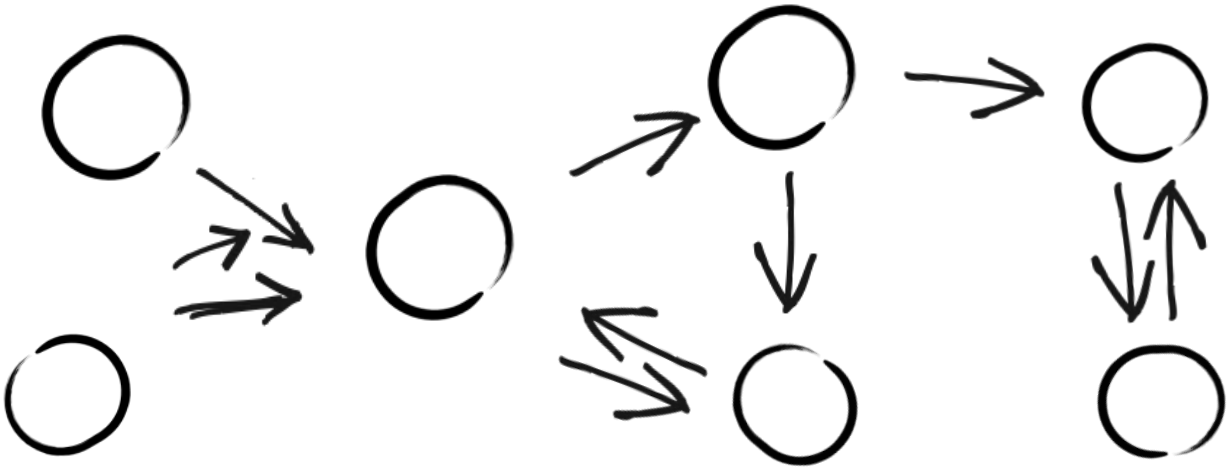
Reactive?

Systems → Reactive Manifesto, resiliency, responsiveness, elasticity, ...

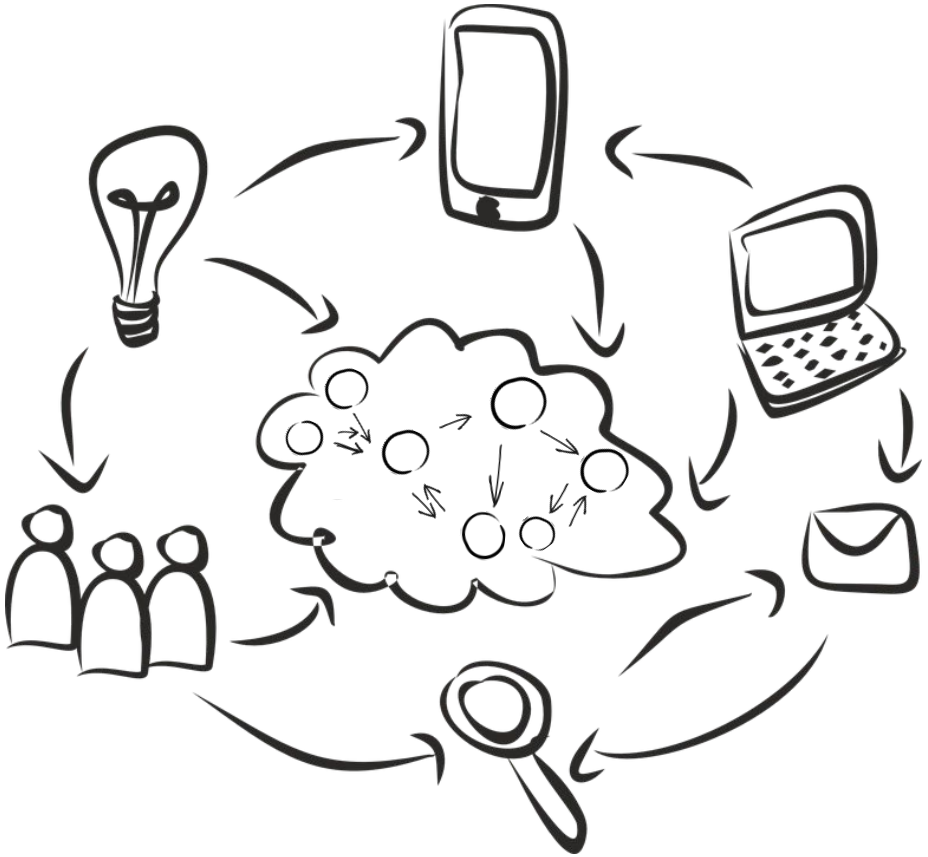
Streams → Modern producer/consumer and back-pressure

Programming → Spreadsheets, reactive extensions, modern UIs, etc

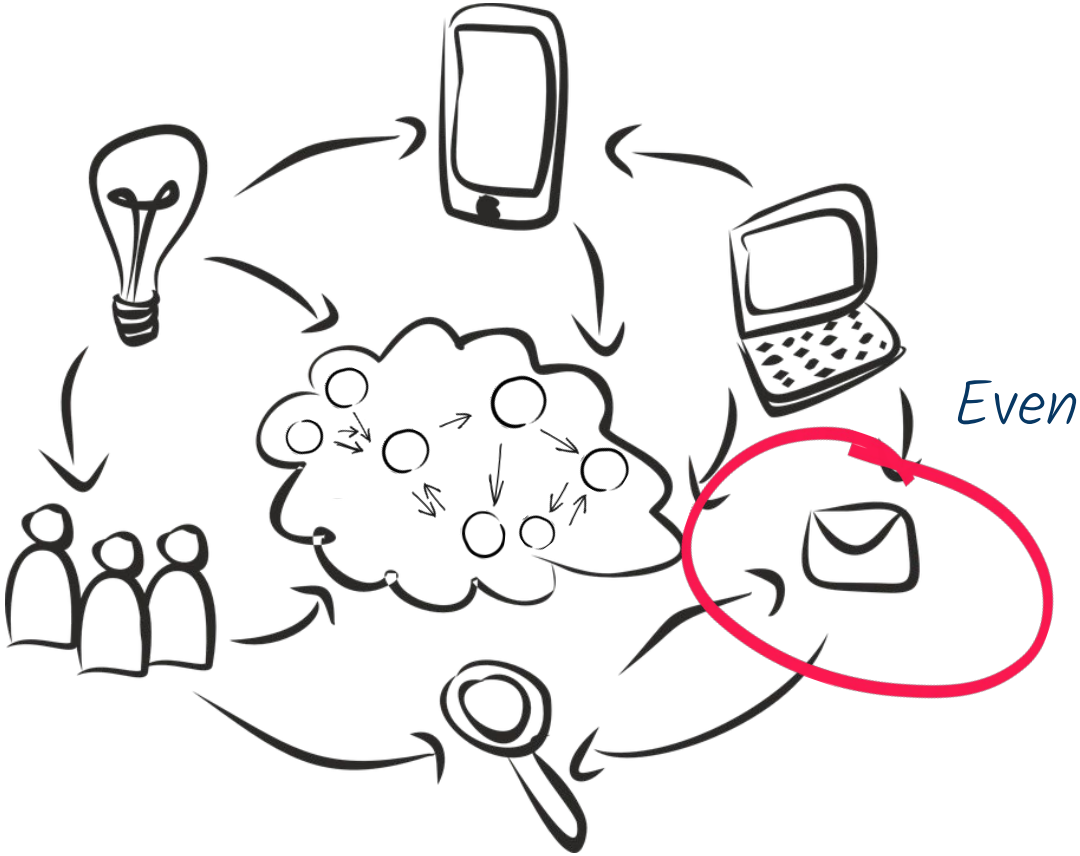
Because of Distributed Systems



Evolution of the Usages => Concurrency



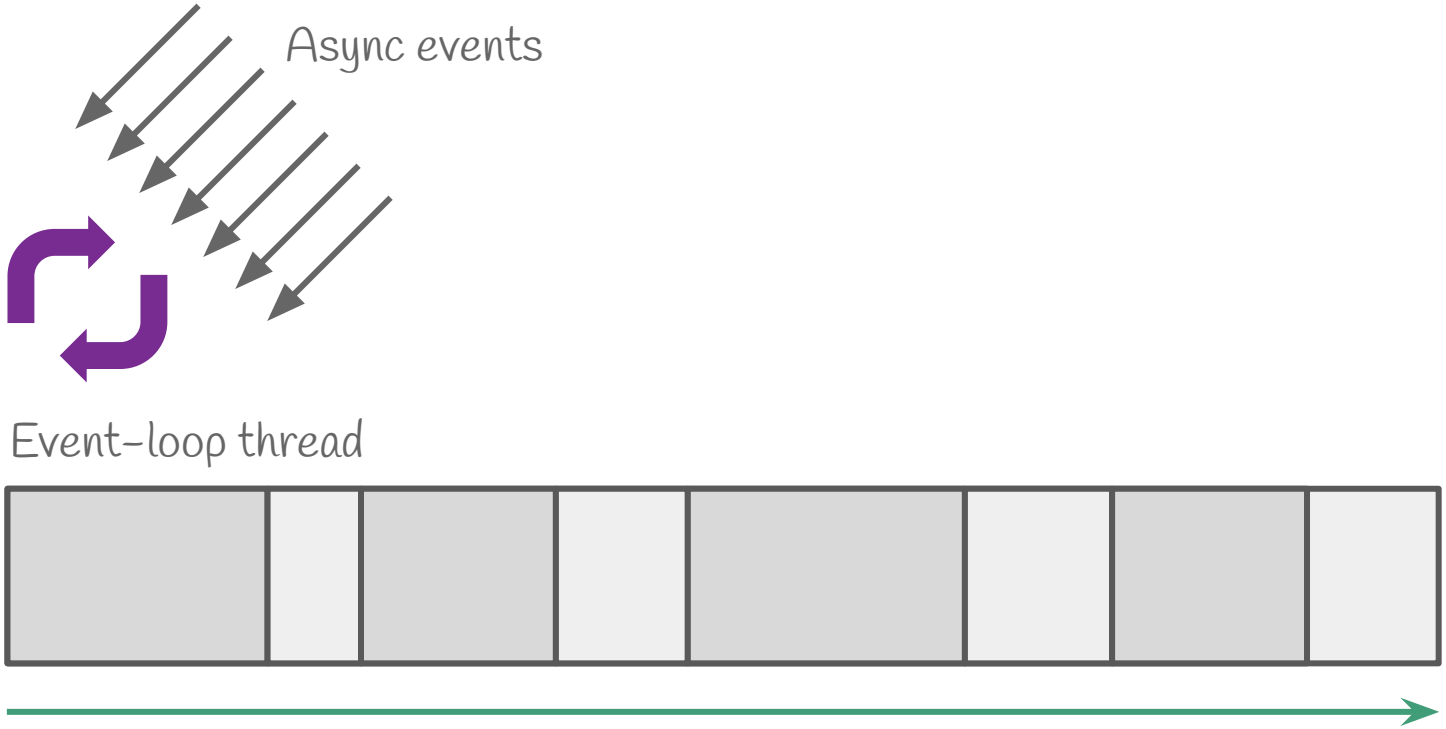
Events, Events everywhere!



Thread request #1

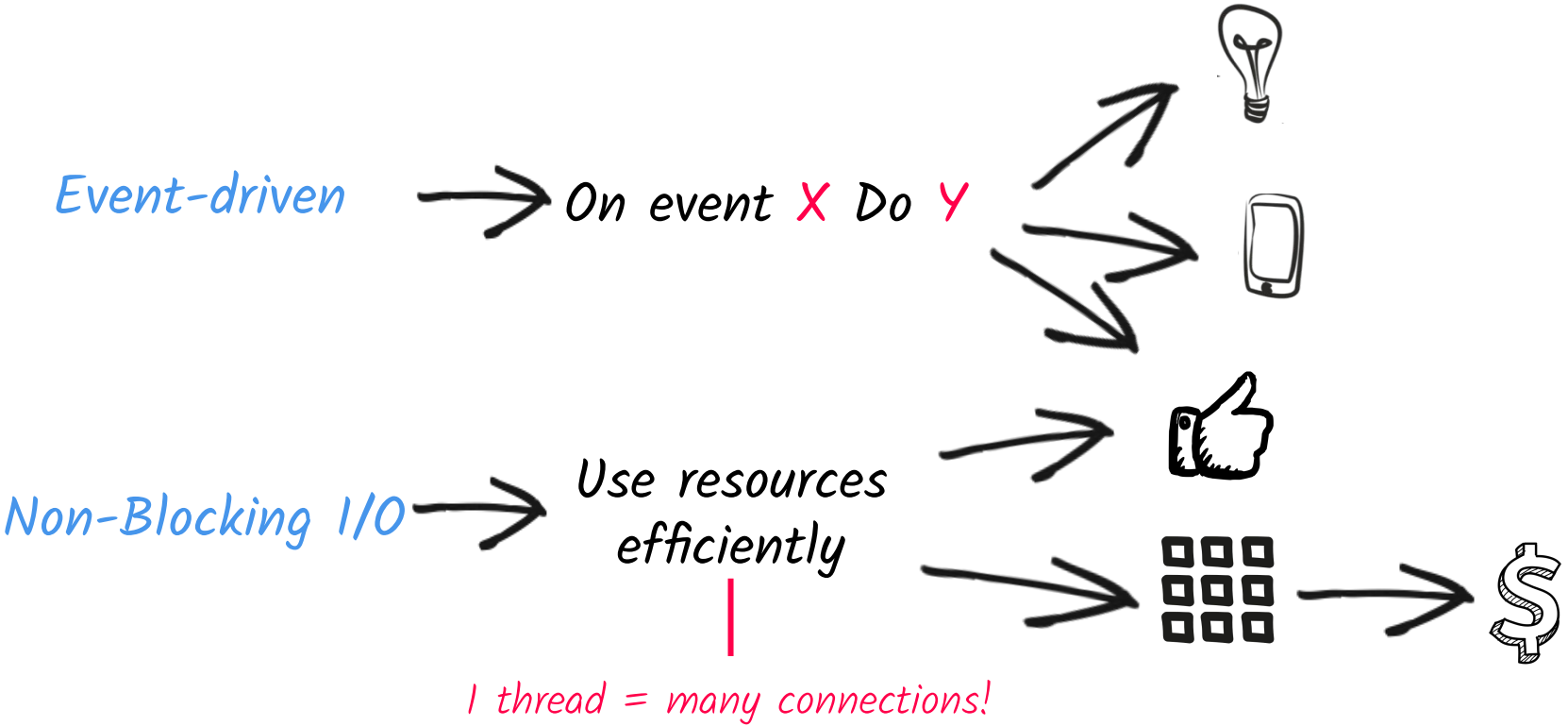
Blocking I/O don’t scale!

Thread request #2

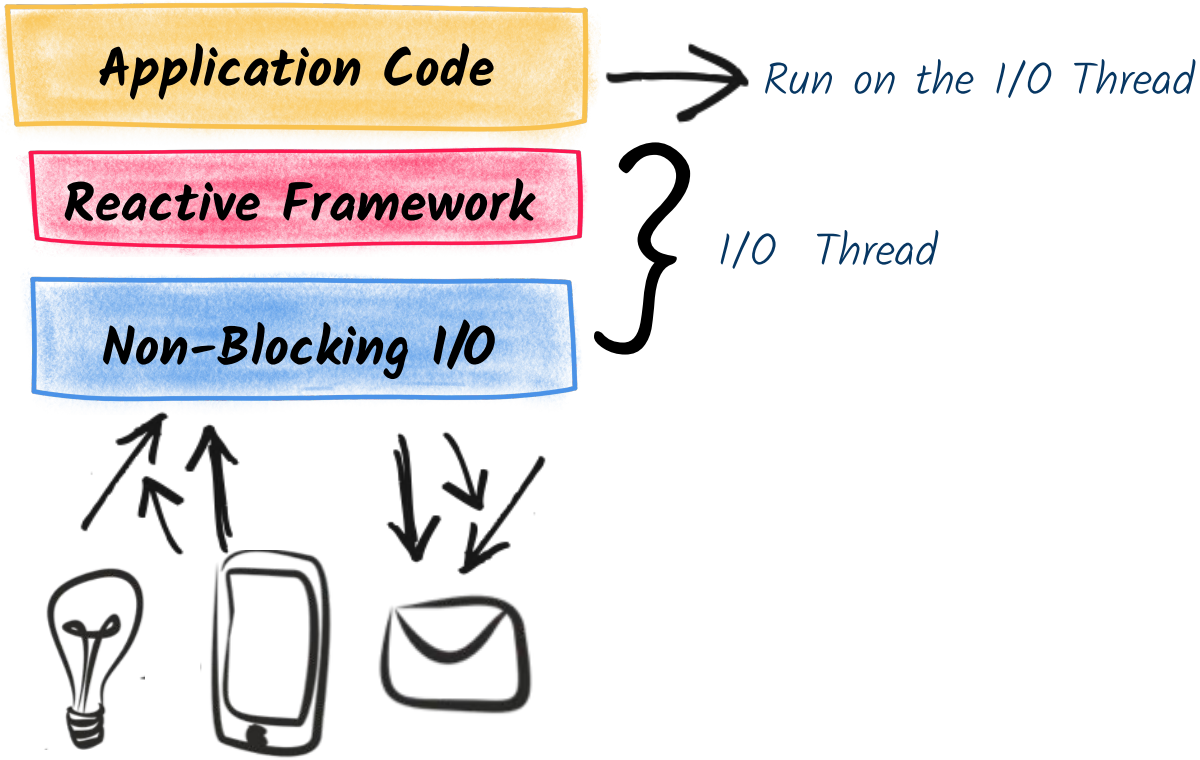


Async I/O do scale!

Benefits of Reactive



Reactive: a different concurrency model



Vert.x

ECLIPSE FOUNDATION

Versatile

Asynchronous

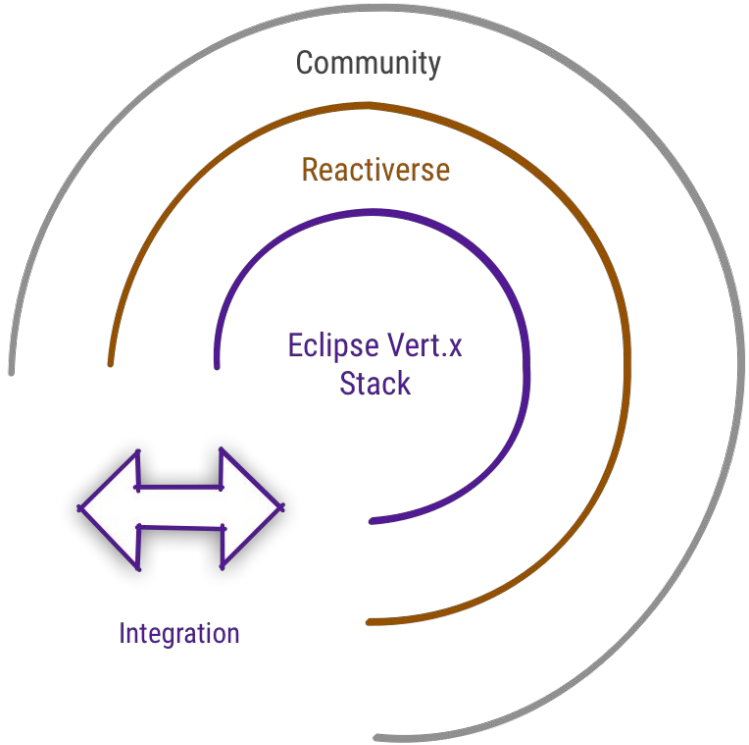
Polyglot

Reactive

Fast

Event-driven

Toolkit

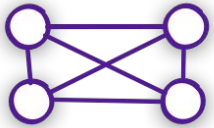


Reactive database clients

Messaging and
event streams

Web APIs and clients

Integration

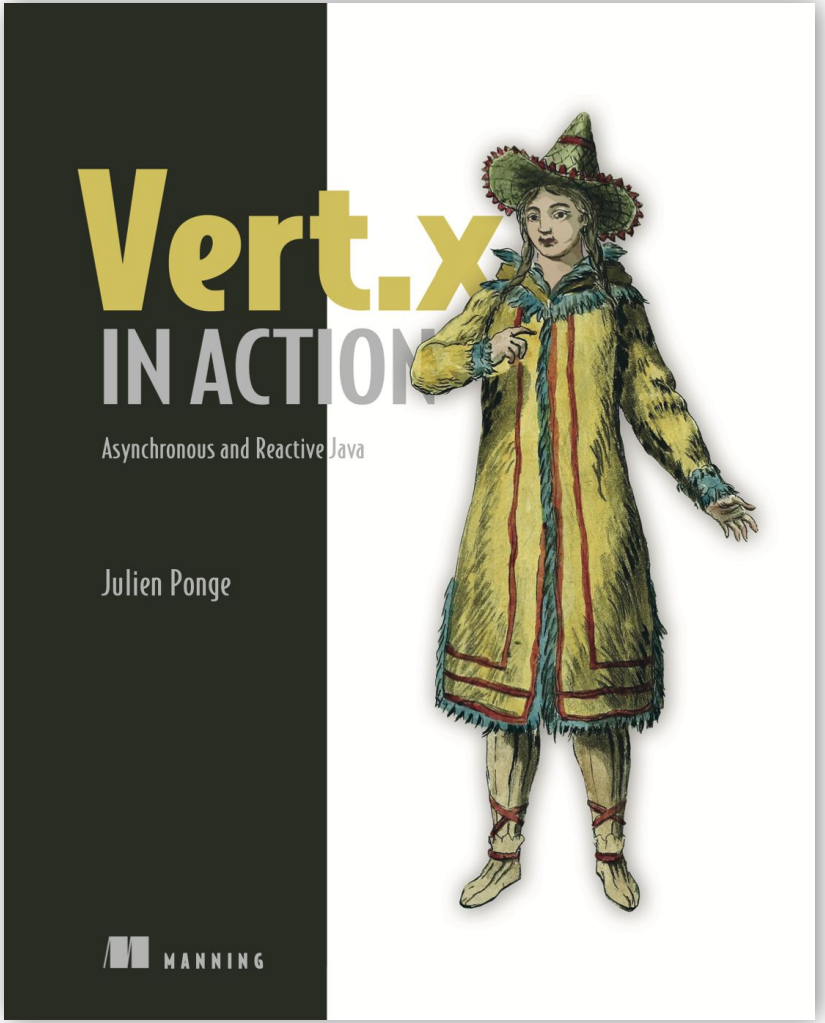


Clustering

Micro-services and
cloud native

Security and
authentication

Show me
some CODE!



PART 1: FUNDAMENTALS OF ASYNCHRONOUS PROGRAMMING WITH VERT.X

PART 2: DEVELOPING REACTIVE SERVICES WITH VERT.X

QUARKUS

Developer Joy

Supersonic Subatomic Java

Best of breed

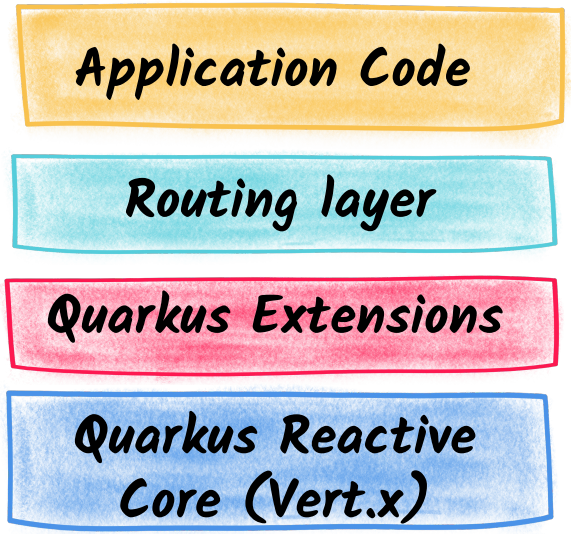
libraries and standards

VERT.X

Unifies

Reactive and Imperative

SUBATOMIC SUPERSONIC REACTIVE!



DATA

- Mongo, Redis, PostGreSQL, MySQL, DB2...
- Panache and Hibernate Reactive

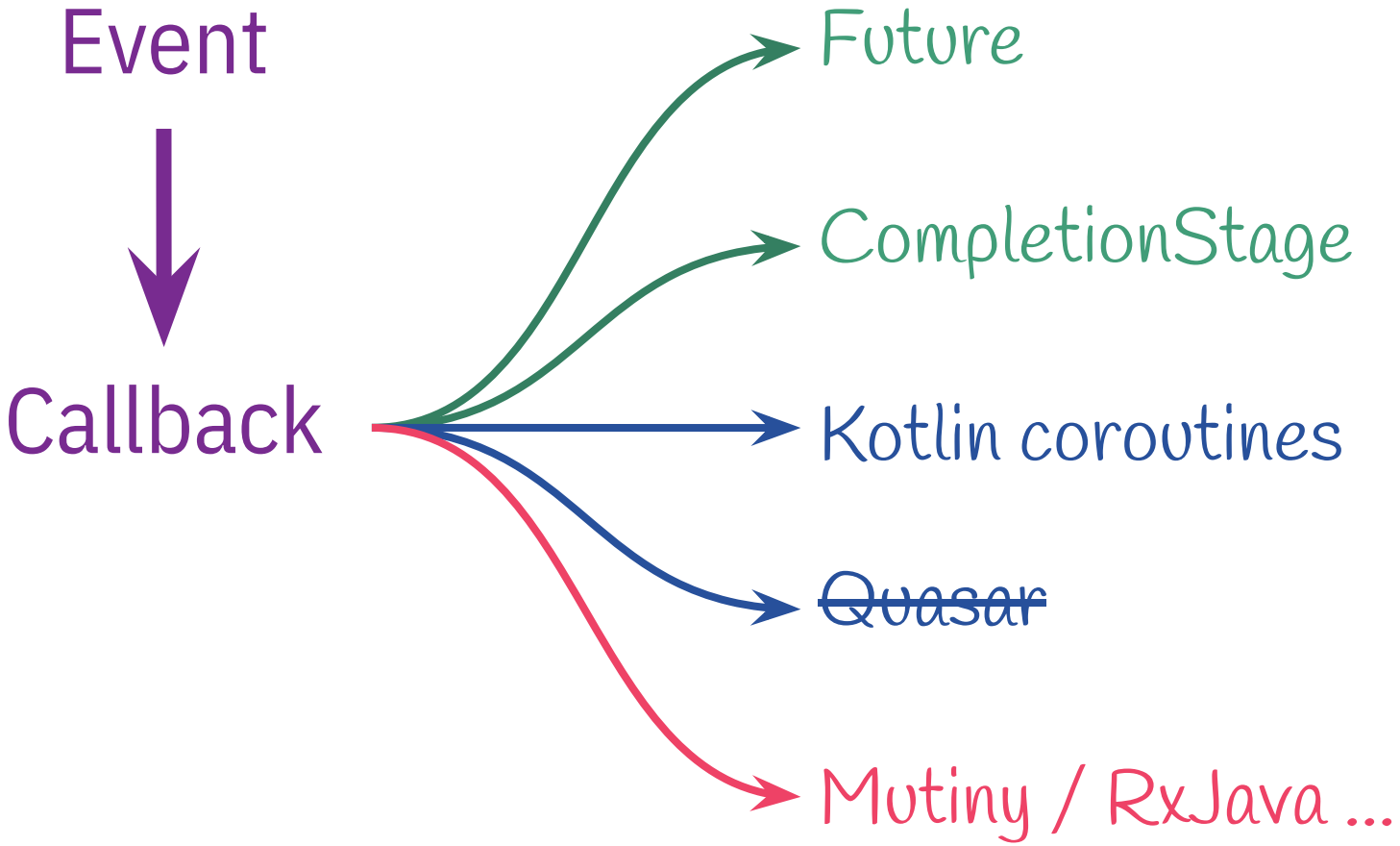
COMMUNICATION

- gRPC, Kafka, AMQP, MQTT, Apache Camel...
- MicroProfile Reactive Messaging

WEB

- Mail, Templating, HTTP / REST, GraphQL...

How do we tame the asynchronous beast?


Event
Callback
Future
CompletionStage
Kotlin coroutines
~~Quasar~~
Mutiny / RxJava ...

Callbacks

```
vertx.createHttpServer()
    .requestHandler(req ->  // Async reaction
        req.response().end("Reactive Greetings")
    )
    .listen(8080, ar -> { // Async operation
        // Continuation...
    });
```

... Callback HELL!

```
client.getConnection(conn -> {
    if (conn.failed()) {/* failure handling */}
    else {
        SQLConnection connection = conn.result();
        connection.query("SELECT * from PRODUCTS",
            rs -> {
                if (rs.failed()) {/* failure handling */}
                else {
                    List<JsonArray> lines = rs.result().getResults();
                    for (JsonArray l : lines) {  log(new Product(l)); }
                    connection.close(done -> {
                        if (done.failed()) {/* failure handling */}
                    });
                }
        });
    }
});
```

Reactive eXtensions & Programming

```
client.rxGetConnection()  // Single(async op)
 .flatMapPublisher(conn ->
   conn
     .rxQueryStream("SELECT * from PRODUCTS")
     .flatMapPublisher(SQLRowStream::toFlowable)
     .doAfterTerminate(conn::close)
 ) // Flowable of Rows
 .map(Product::new) // Flowable of Products
 .subscribe(System.out::println);
```

... Map / flatMap JUNGLE

```
client.rxGetConnection()  // Single(async op)
 .flatMapPublisher(conn ->
   conn
     .rxQueryStream("SELECT * from PRODUCTS")
     .flatMapPublisher(SQLRowStream::toFlowable)
     .doAfterTerminate(conn::close)
 ) // Flowable of Rows
 .map(Product::new) // Flowable of Products
 .subscribe(System.out::println);
```

Why MUTINY!

No more "Monad-hell"!

```
WebClient webClient = WebClient.create(vertx);
webClient
  .get(3001, "localhost", "/ranking-last-24-hours")
  .as(BodyCodec.jsonArray())
  .rxSend()
  .delay(5, TimeUnit.SECONDS, RxHelper.scheduler(vertx))
  .retry(5)
  .map(HttpResponse::body)
  .flattenAsFlowable(Functions.identity())
  .cast(JsonObject.class)
  .flatMapSingle(json -> whoOwnsDevice(webClient, json))
  .flatMapSingle(json -> fillWithUserProfile(webClient, json))
  .subscribe(
    this::hydrateEntryIfPublic,
    err -> logger.error("Hydratation error", err),
    () -> logger.info("Hydratation completed"));
```

```
eventConsumer
  .subscribe("incoming.steps")
  .toFlowable()
  .
```

```
concatMapMaybeDelayError(Function<? super KafkaConsumerRe…   Flowable<R>
concatMapMaybeDelayError(Function<? super KafkaConsumerRe…   Flowable<R>
concatMapSingle(Function<? super KafkaConsumerRecord<Stri…   Flowable<R>
concatMapSingle(Function<? super KafkaConsumerRecord<Stri…   Flowable<R>
concatMapSingleDelayError(Function<? super KafkaConsumerR…   Flowable<R>
concatMapSingleDelayError(Function<? super KafkaConsumerR…   Flowable<R>
concatMapSingleDelayError(Function<? super KafkaConsumerR…   Flowable<R>
concatWith(Completa…   Flowable<KafkaConsumerRecord<String, JsonObject>>
concatWith(Publishe…   Flowable<KafkaConsumerRecord<String, JsonObject>>
concatWith(MaybeSou…   Flowable<KafkaConsumerRecord<String, JsonObject>>
concatWith(SingleSo…   Flowable<KafkaConsumerRecord<String, JsonObject>>
contains(Object item)                                  Single<Boolean>
count()                                                   Single<Long>
debounce(long timeo…   Flowable<KafkaConsumerRecord<String, JsonObject>>
debounce(long timeo…   Flowable<KafkaConsumerRecord<String, JsonObject>>
debounce(Function<?…   Flowable<KafkaConsumerRecord<String, JsonObject>>
Results might be incomplete while indexing is in progress
```

```
}
pr
}
pr
JsonObject data = record.value();
```

We need navigable APIs!

Two types: Uni & Multi

Uni

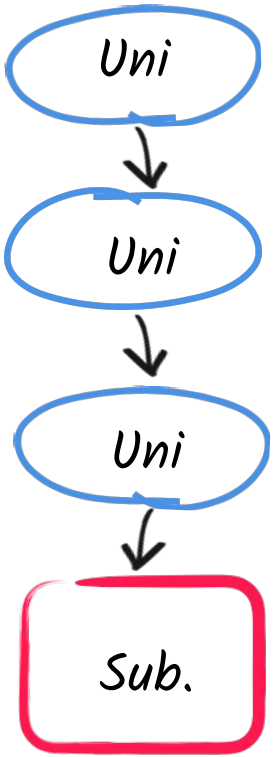
- Models asynchronous operations
- Emits 1 item or failure events
- Not *reactive streams* compliant! (no back-pressure, *null* is permitted, etc)

Multi

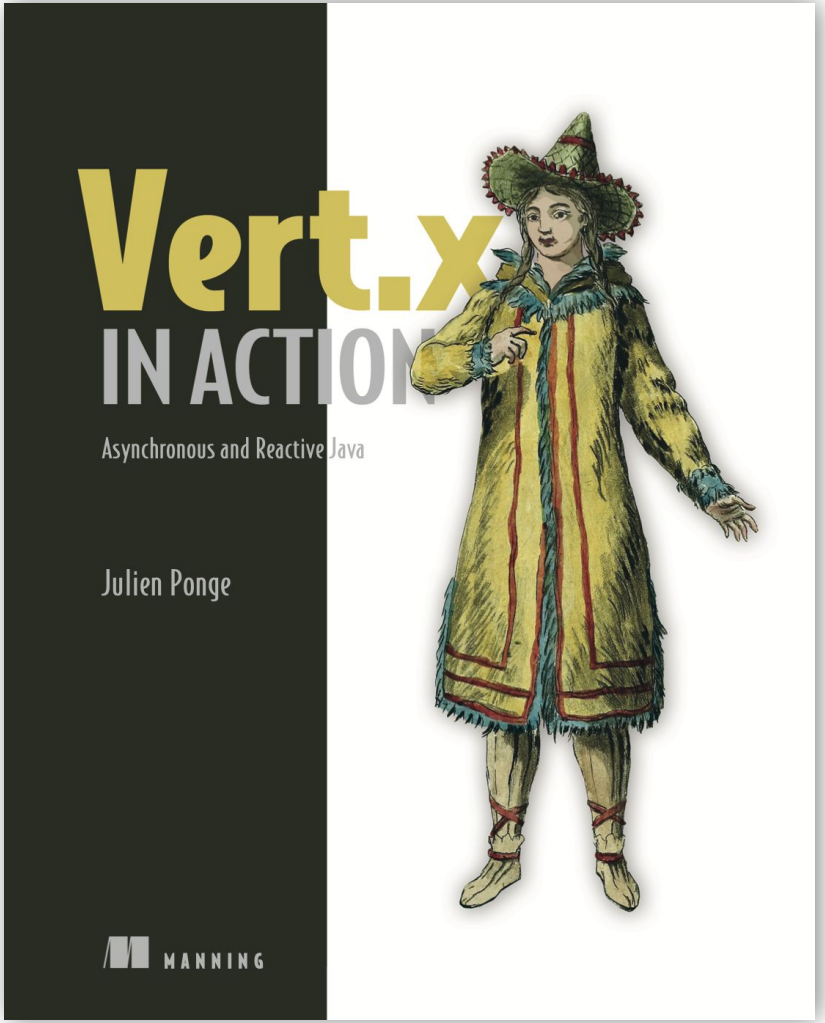
- Models a stream of events
- Emits n items, failure or completion events
- Implements reactive streams

MUTINY!

```
service.order(order)

   .onItem().transform(i -> process(i))

   .onFailure().recoverWithItem(fallback)

   .subscribe().with(
      item -> ...
);
```



Show me
some CODE!


Vert.x
IN ACTION
Asynchronous and Reactive Java
Julien Ponge
MANNING

Q&A