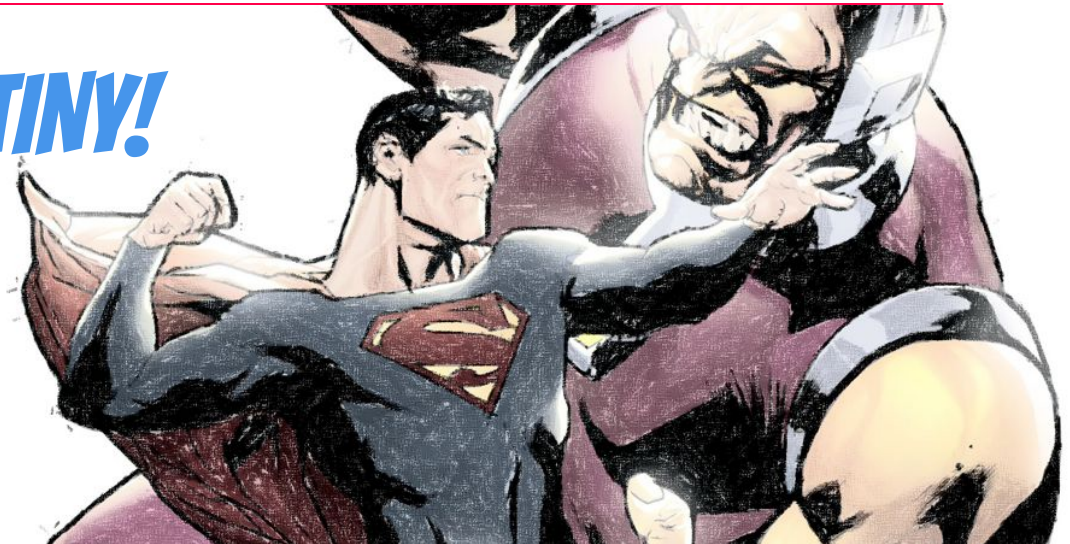
QUARKUS

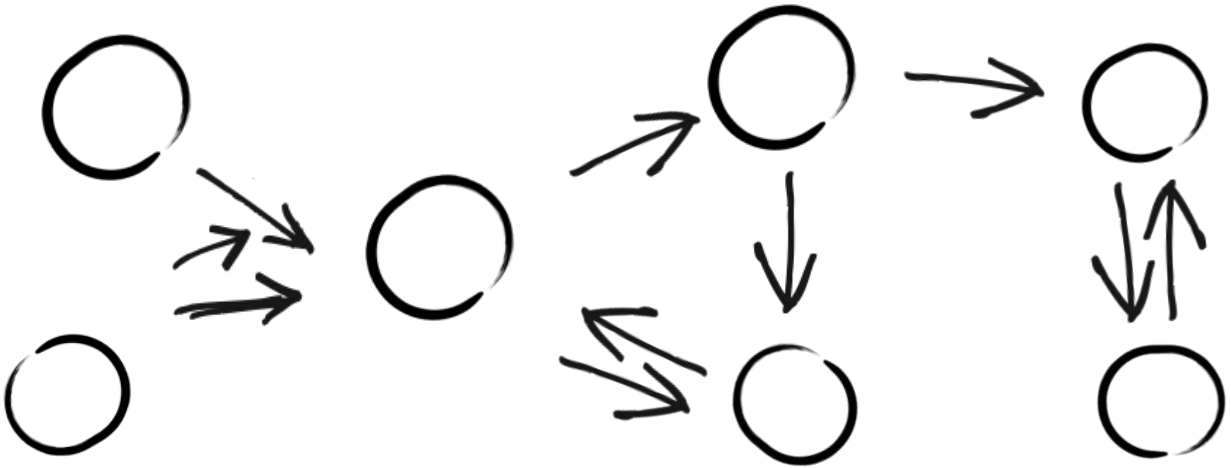
A *Reactive* MUTINY!

Clement Escoffier, Red Hat

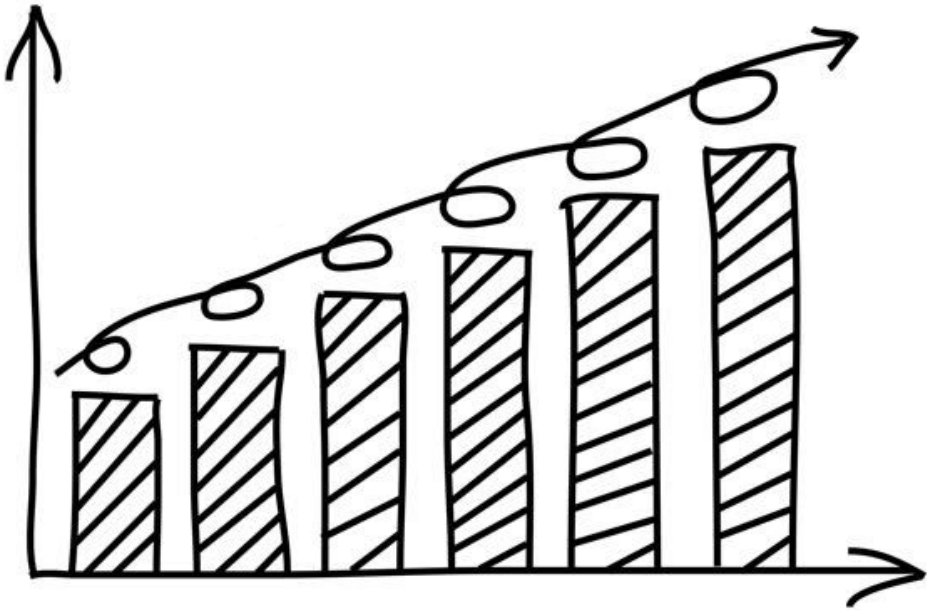


Why does reactive matter?

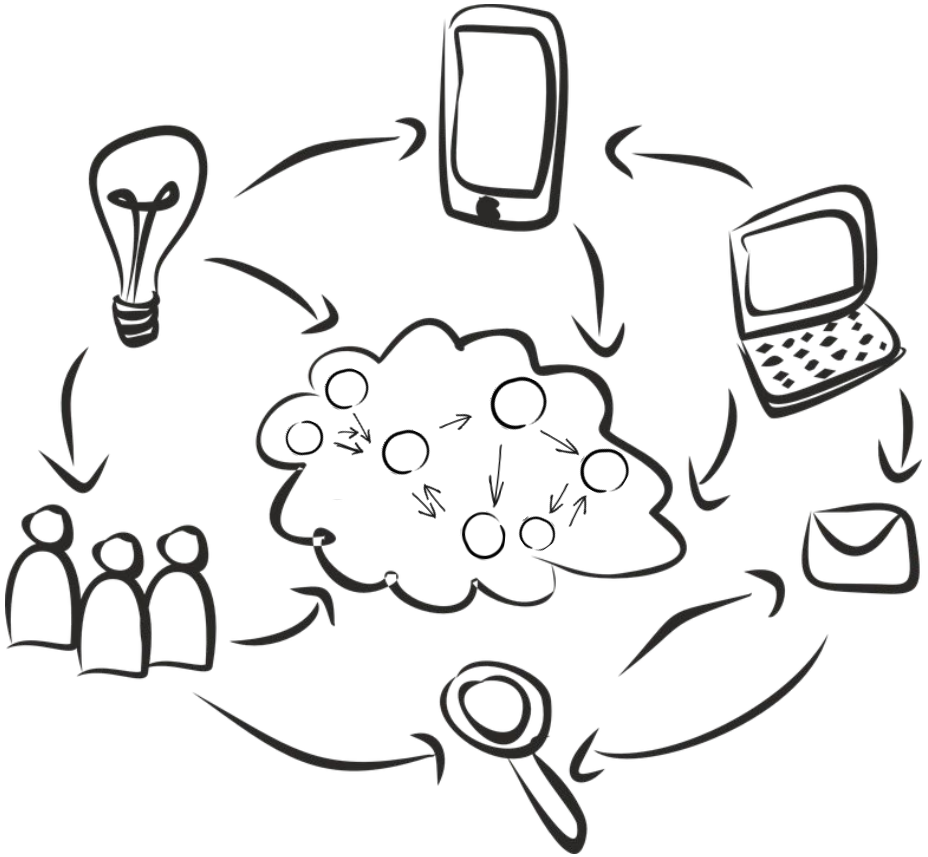
Because of distributed systems



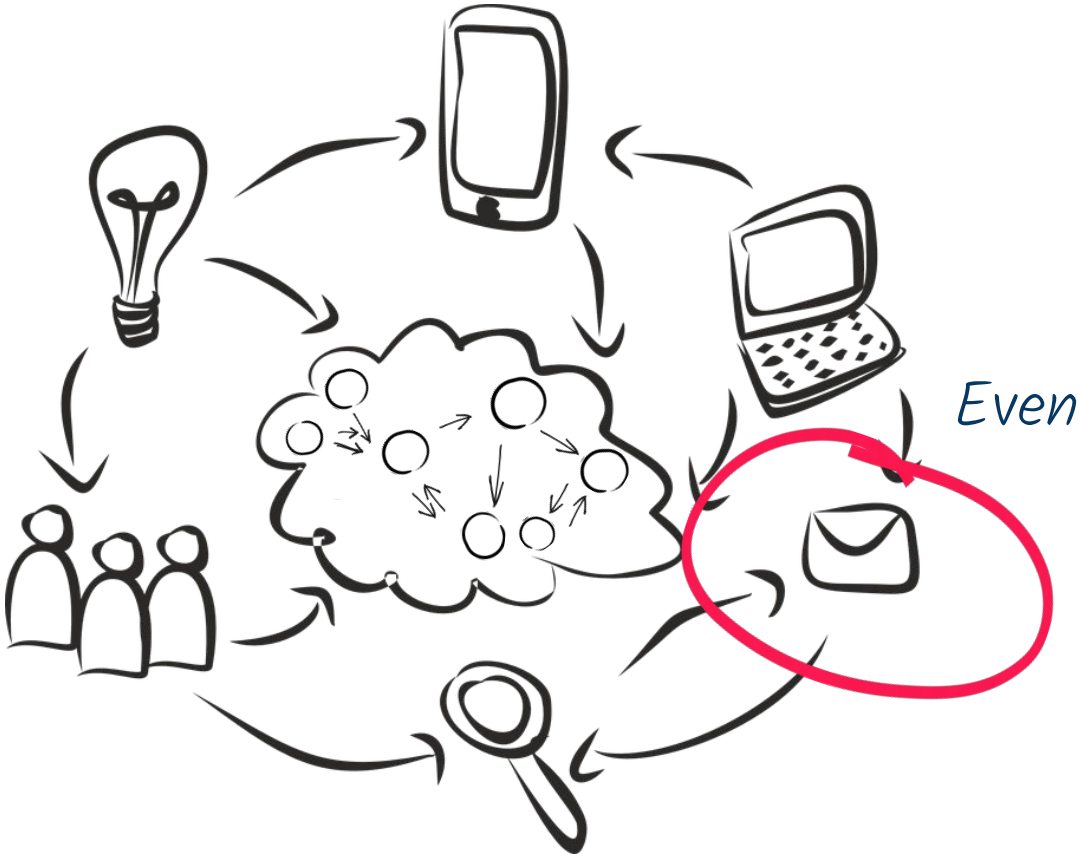
Req/sec are growing



Evolution of the usages => Concurrency



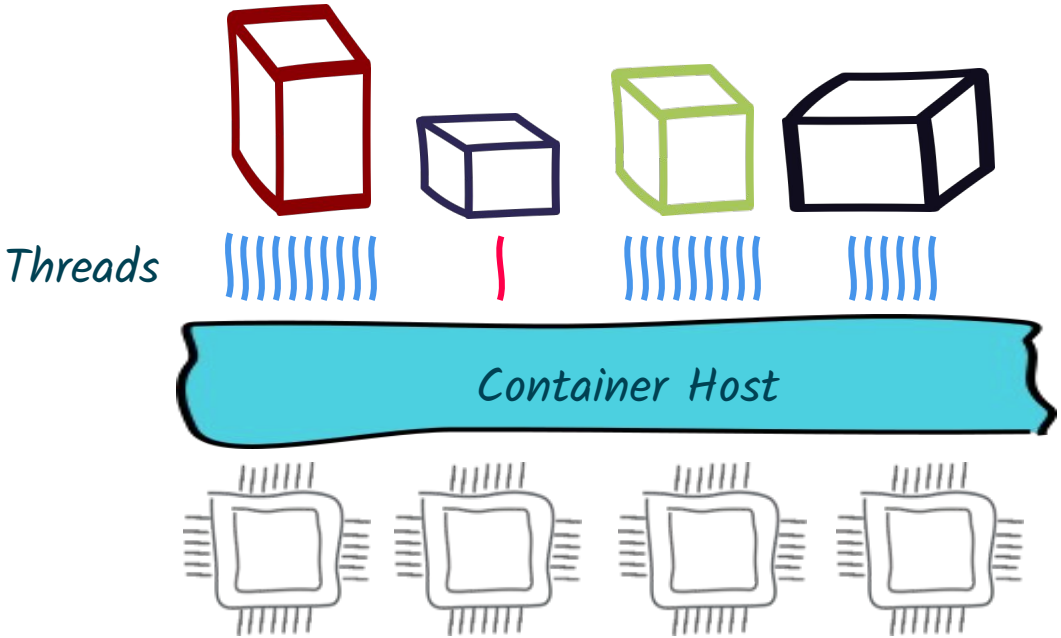
Evolution of the usages



Events, a **LOT** of Events

The hidden truth of containers

Containers are about **sharing** and **deployment density**



costs memory

cost lots of memory

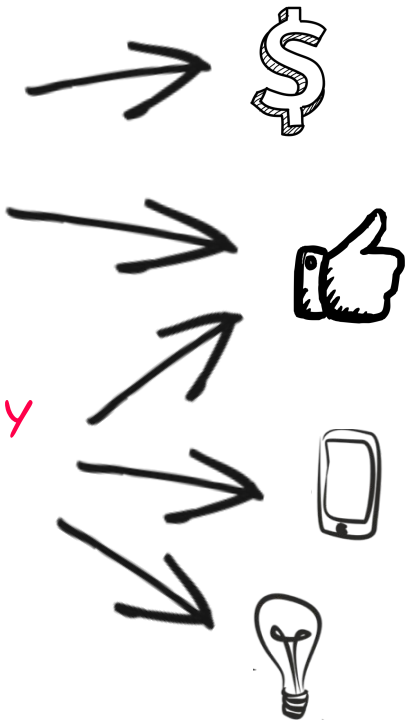
cost CPU cycles

\+ quotas = BOOM

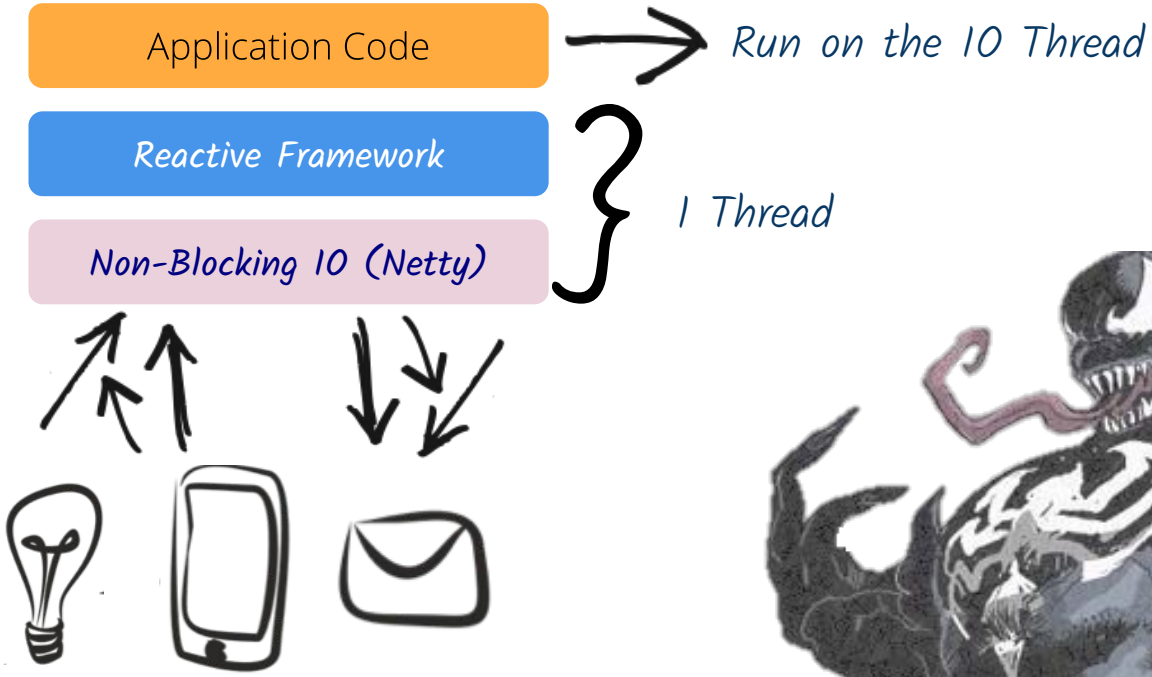
Benefits of Reactive

Non-Blocking IO → Use resources efficiently

Event-driven → On event X Do Y



Reactive brings a different concurrency model



How do we tame the asynchronous beast

Callbacks

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

… 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) {  System.out.println(new Product(l)); }
                   connection.close(
                      done -> {
                       if (done.failed()) {/* failure handling */}
                   });
           }
       });
   }
});
```

Reactive eXtension & 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);
```

MUTINY!

https://smallrye.io/smallrye-mutiny

EVENT-DRIVEN

onItem()

onFailure()

onCancellation()

onCompletion()

NAVIGABLE

The API guides you

Number of methods per class

EXPRESSIVENESS

Express your logic

Avoid maths

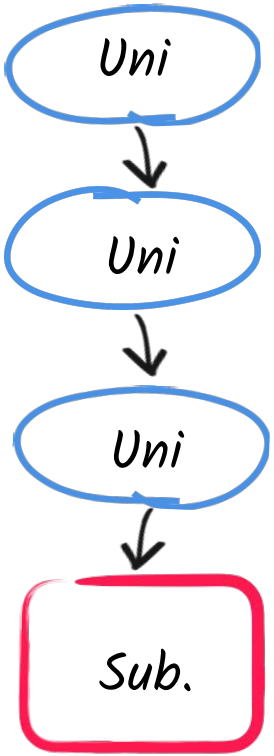
MUTINY!

```
service.order(order)

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

   .onFailure().recoverWithItem(fallback)

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



MUTINY!

UNI

1 item or failure

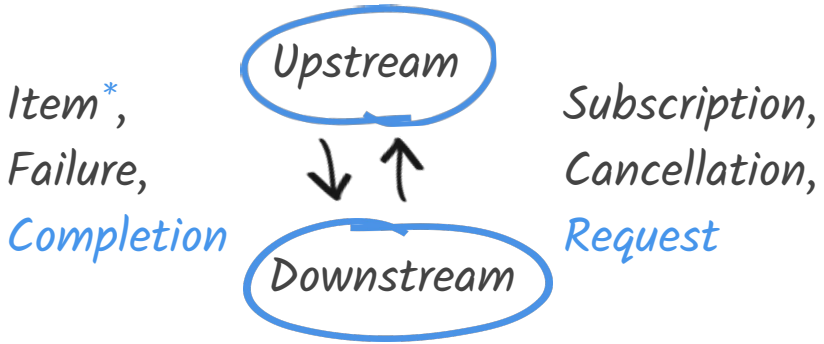
Async operation

MULTI

0..n item or 1 failure

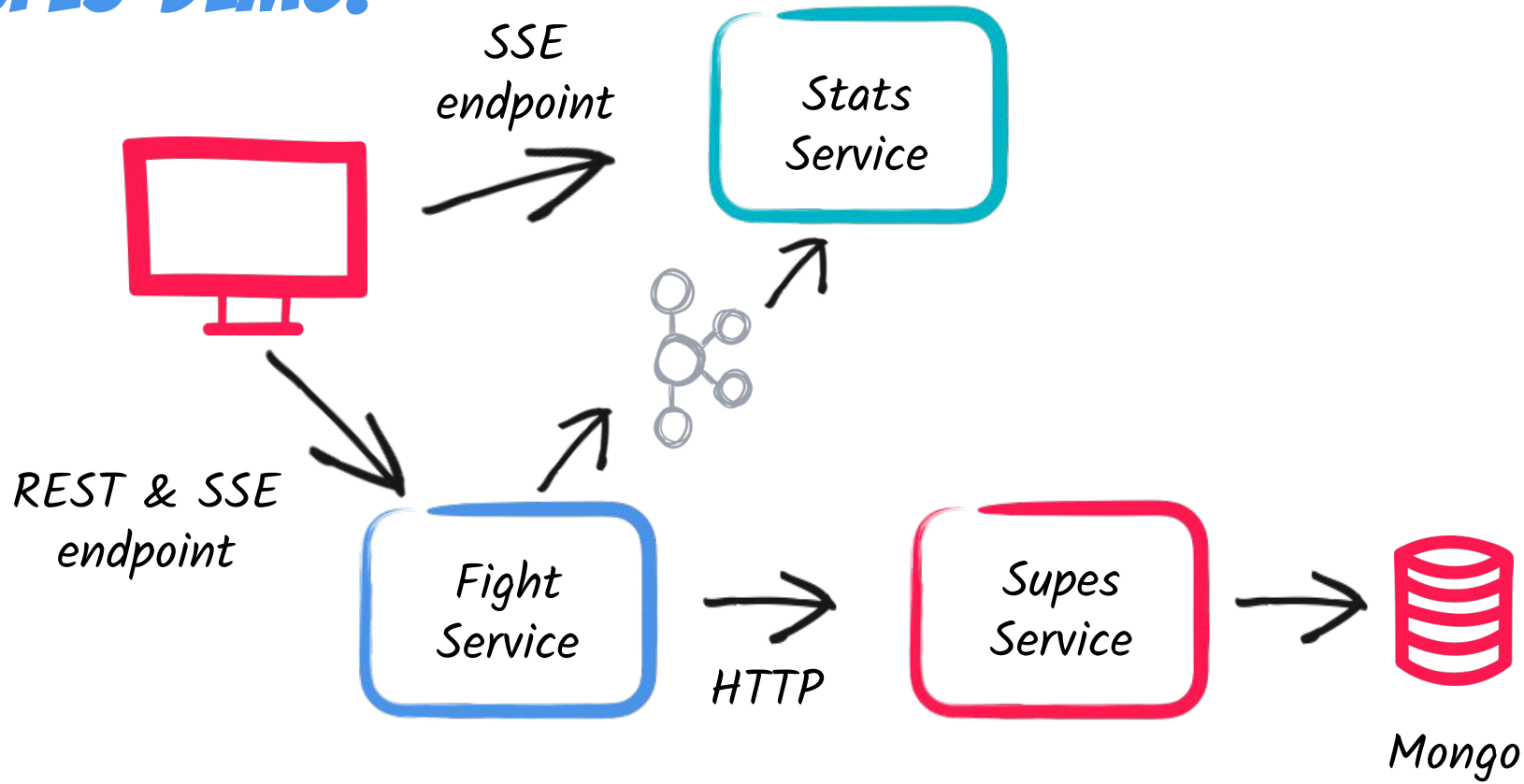
Back-pressure

EVENTS



Show me
some CODE!

SUPES-DEMO!


SSE
endpoint
Stats
Service
REST & SSE
endpoint
Fight
Service
HTTP
Supes
Service
Mongo

QUARKUS

https://quarkus.io & https://code.quarkus.io

https://quarkusio.zulipchat.com

@quarkusio

https://github.com/quarkusio/quarkus