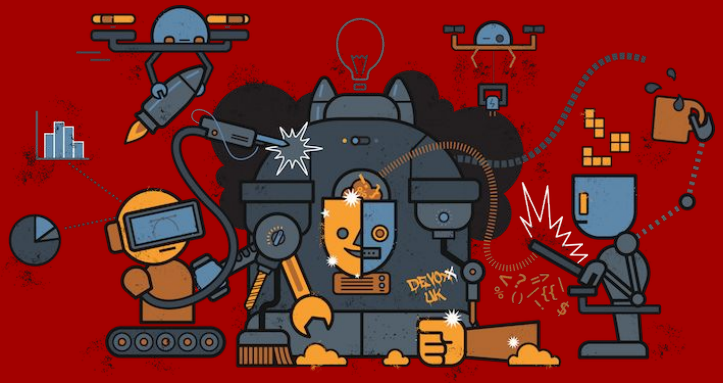
The Reactive Landscape

Clement Escoffier, Vert.x Core Developer, Red Hat



Reactive Fashionista

The new gold rush ?

Reactive System, Reactive Manifesto,
Reactive Extension, Reactive Programming,
Reactive Spring, Reactive Streams...

Scalability, Asynchronous, Back-Pressure,
Spreadsheet, Non-Blocking, Actor, Agent...

redhat.

Reactive ?

Oxford dictionary

1 - Showing a response to a stimulus

1.1 (*Physiology)* Showing an immune response to a specific antigen

1.2 (of a disease or illness) caused by a reaction to something: *'reactive depression'*

2 - Acting in response to a situation rather than creating or controlling it

Reactive Architecture / Software

Application to software

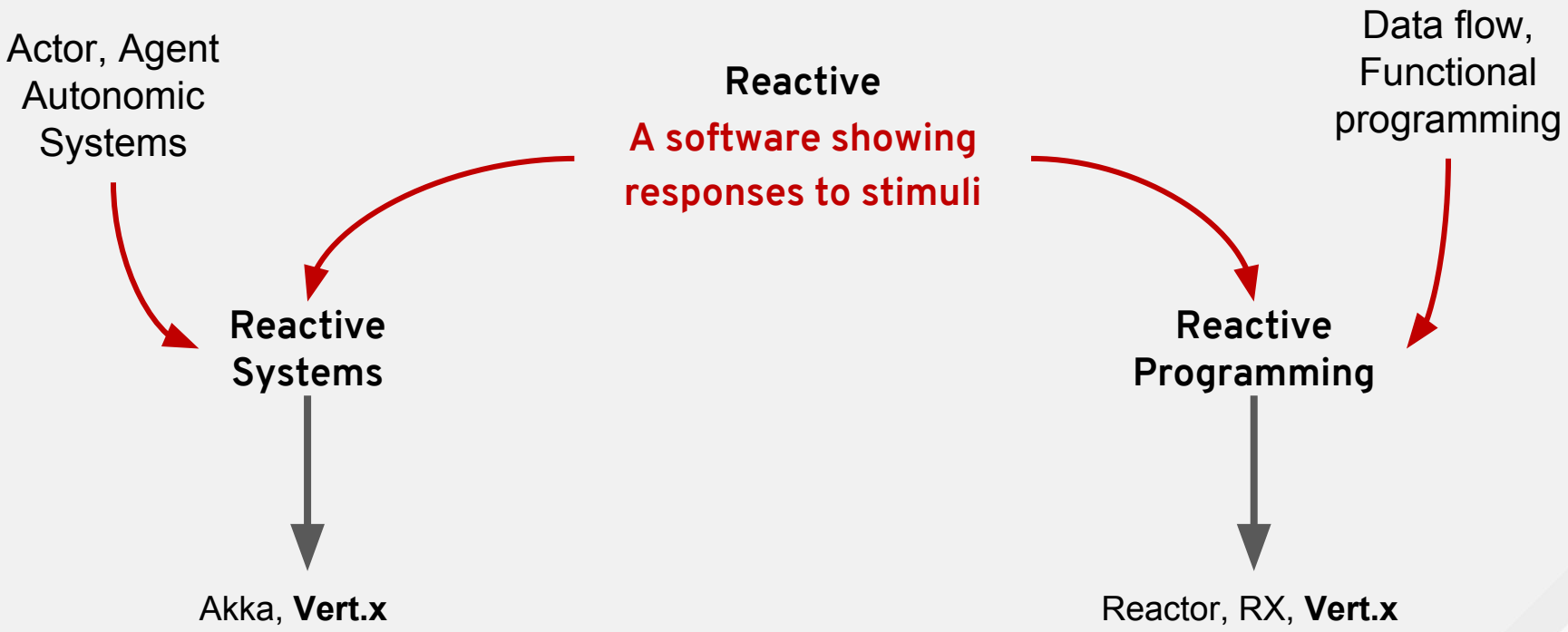
A software showing responses to stimuli

- Events, Messages, Requests, Failures, Measures, Availability…
- The end of the flow of control ?

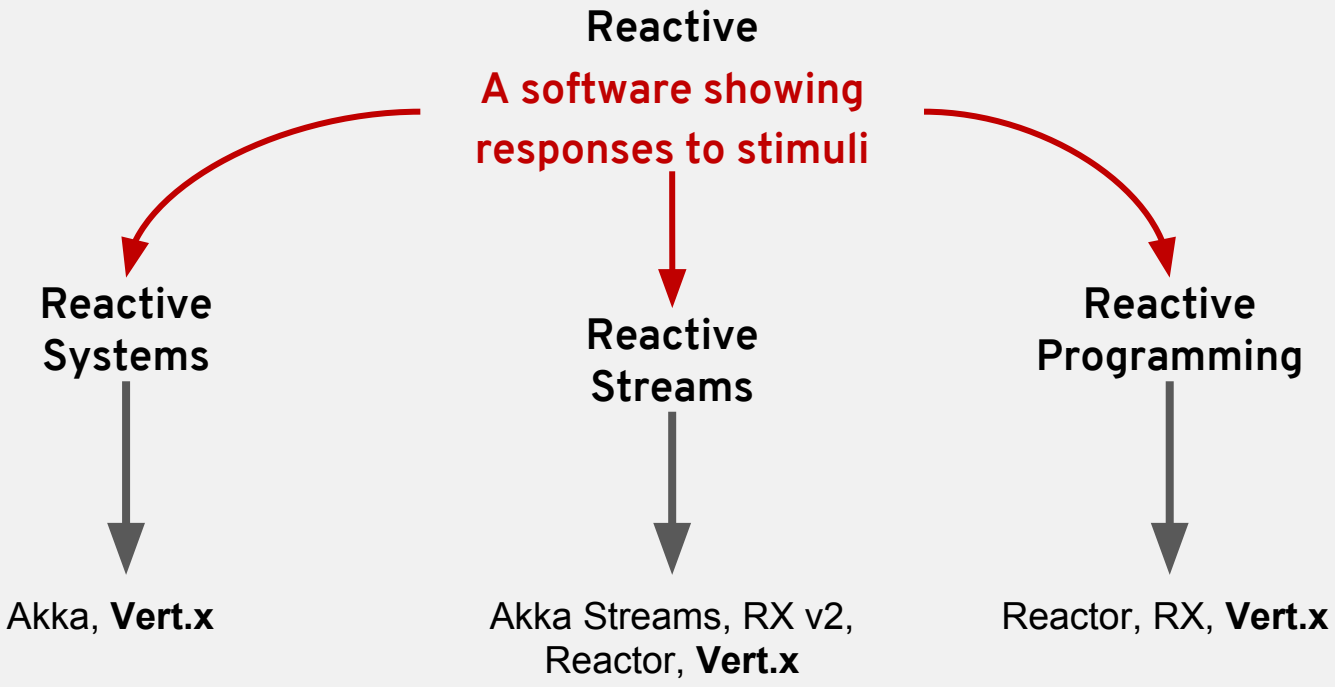
Is it new?

- Actors, Object-oriented programming…

The 2+1* parts of the reactive spectrum



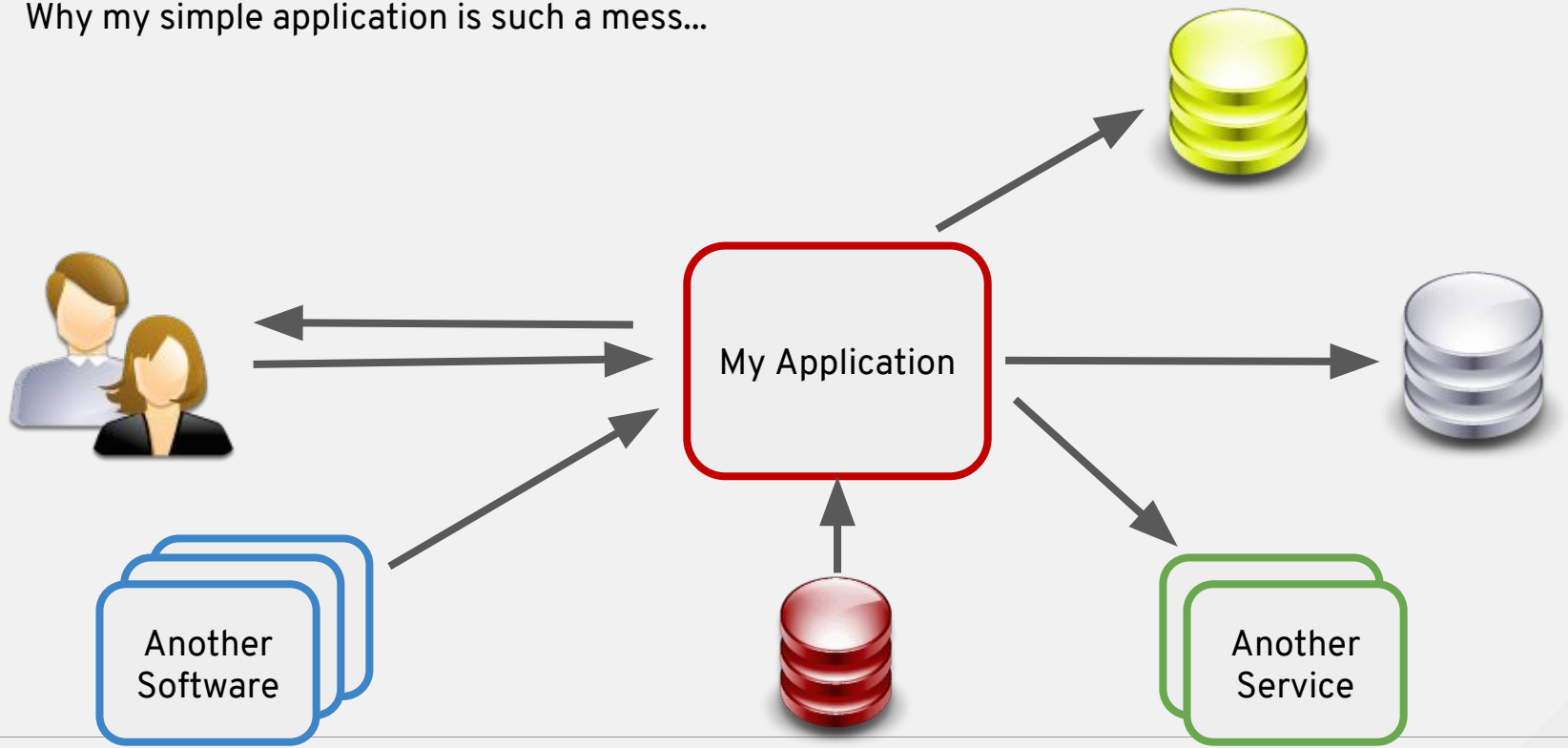
The 2+1* parts of the reactive spectrum



redhat.

Modern software is not autonomous

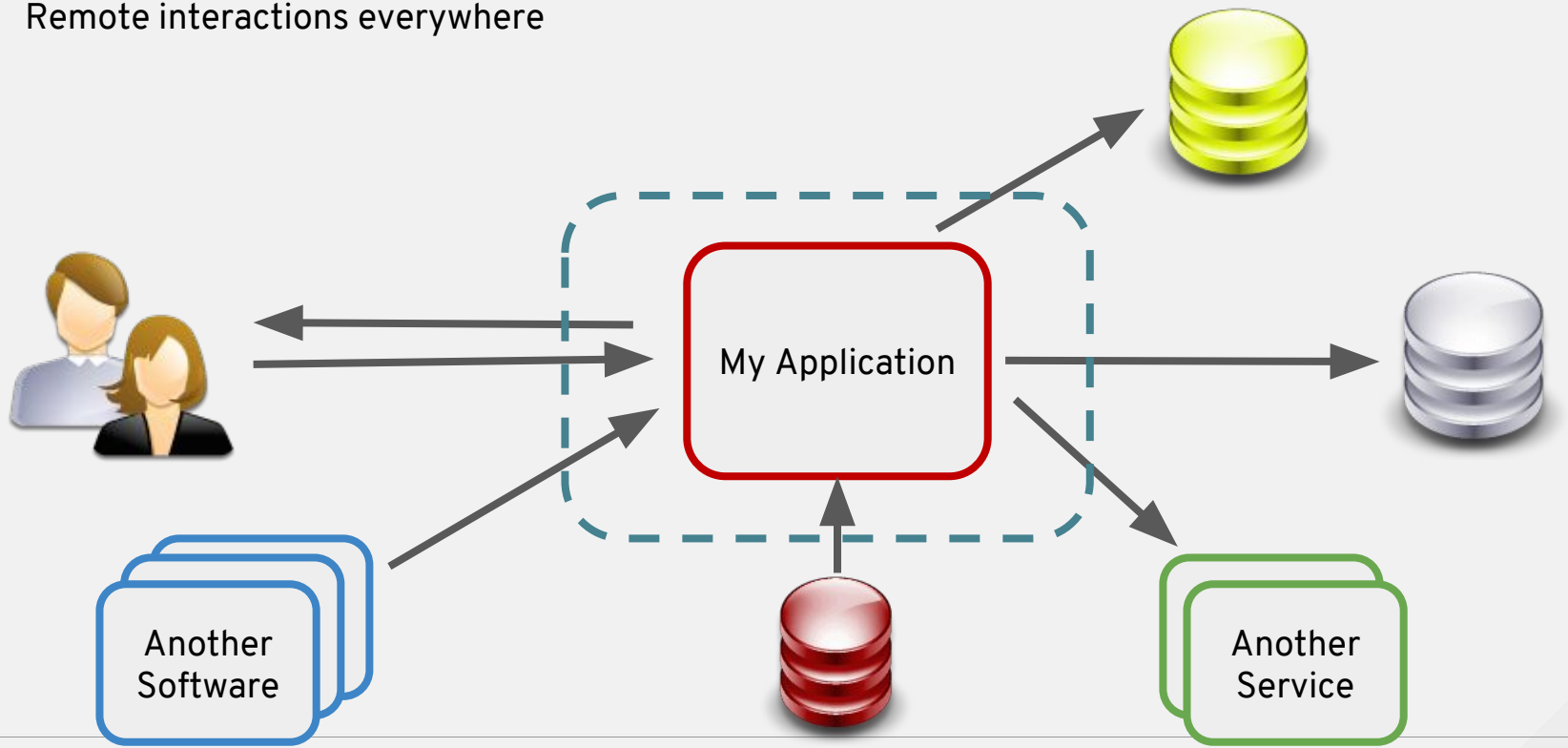
Why my simple application is such a mess...



redhat.

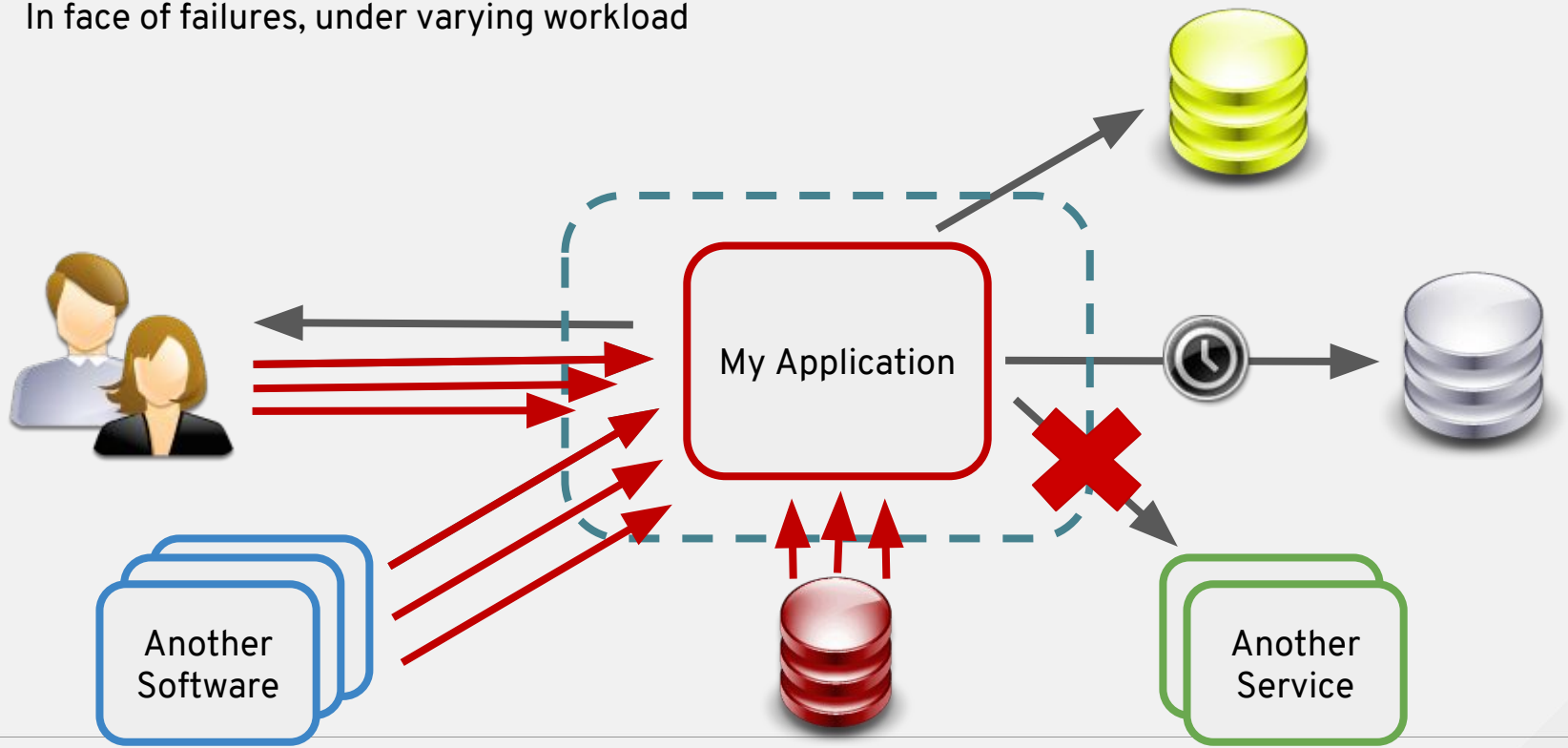
Modern software is not autonomous

Remote interactions everywhere

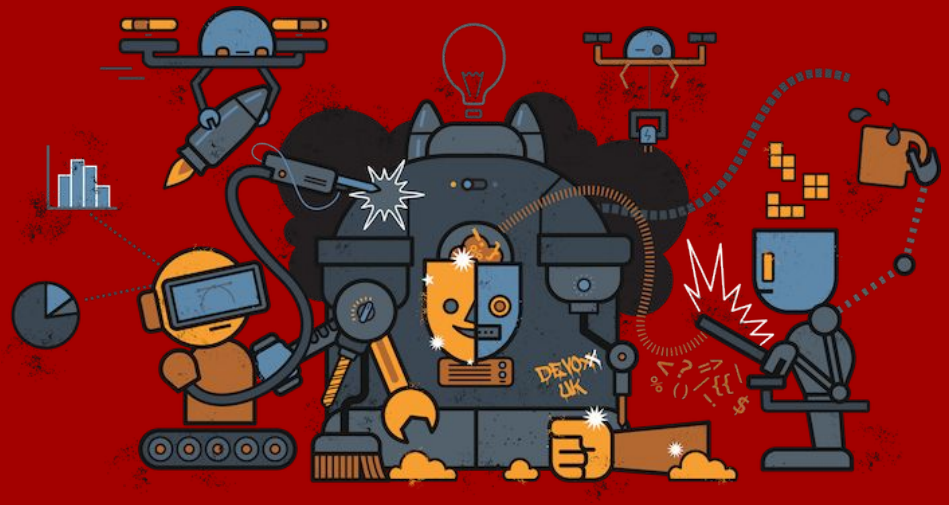


Modern software is not autonomous

In face of failures, under varying workload



Reactive Systems => Responsive Systems

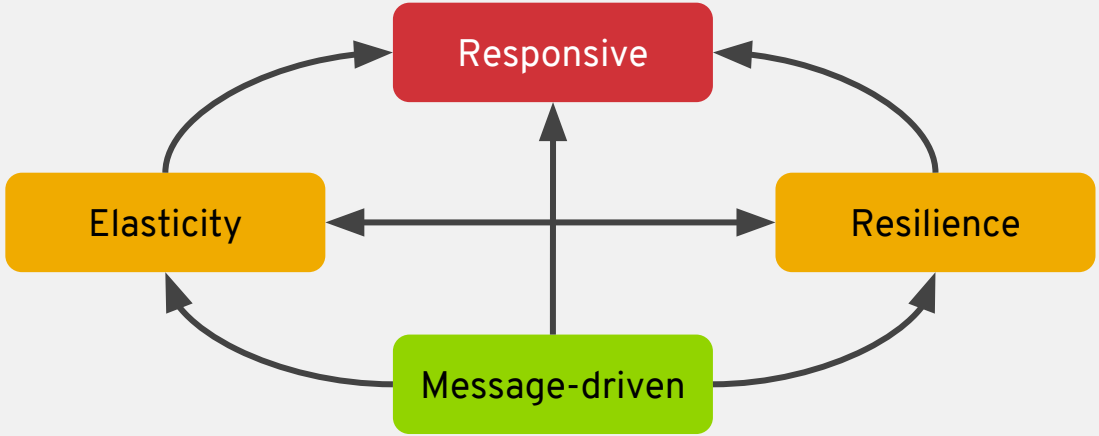


redhat.

Reactive Manifesto

http://www.reactivemanifesto.org/

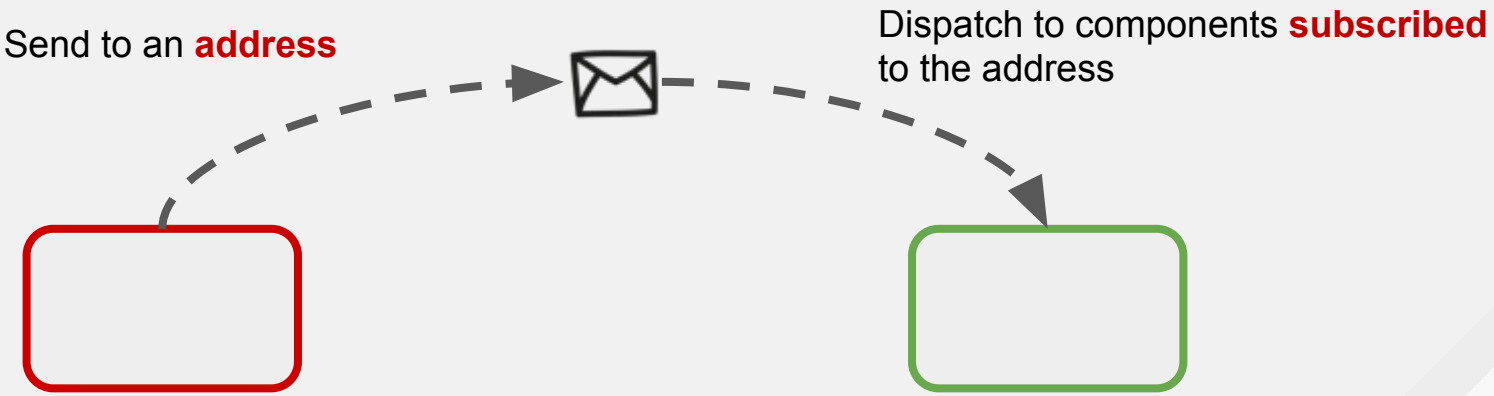
Reactive Systems are an architecture style focusing on **responsiveness**



Asynchronous message passing

Components interacts using **messages**

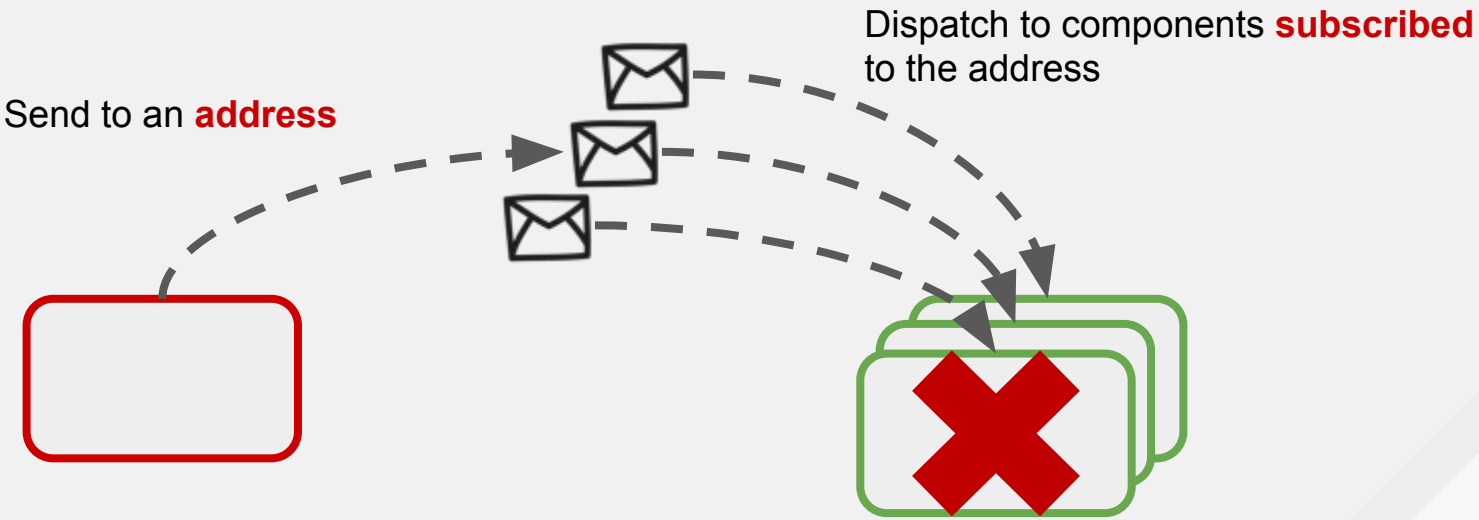
Allow decoupling



Asynchronous message passing

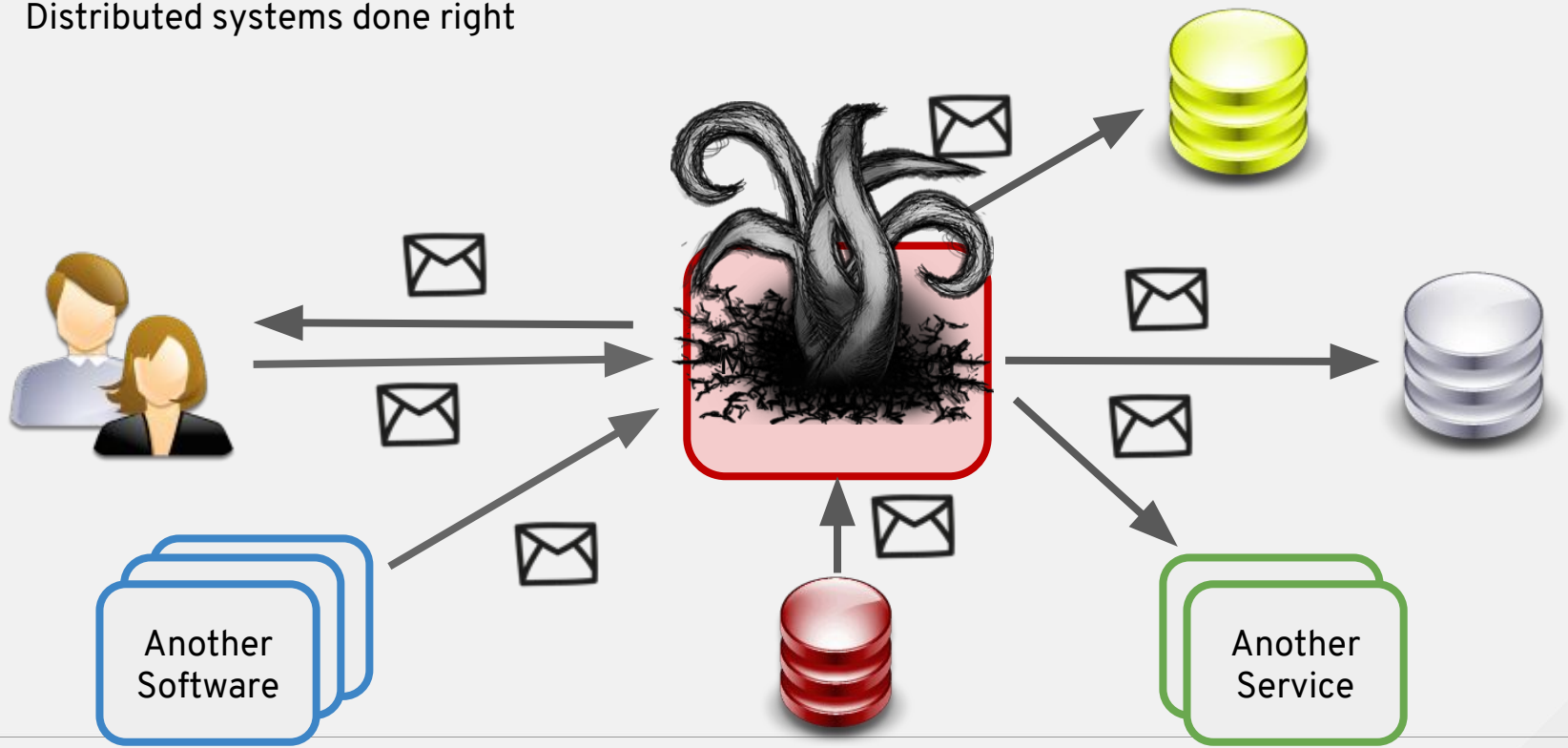
Messages allows **elasticity**

Resilience is not only about failures, it's also about **self-healing**



Reactive Systems

Distributed systems done right



redhat.

Pragmatic reactive systems

And that's what Eclipse Vert.x offers to you

Development model => Embrace **asynchronous**

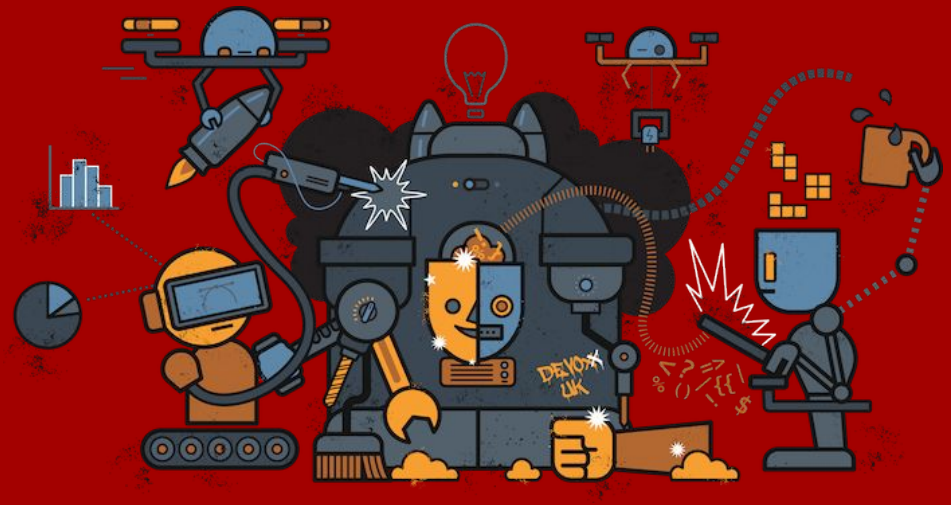
Simplified concurrency => **Event-loop**, not thread-based

I/O

- **Non-blocking I/O**, if you can't isolate

Asynchronous non-blocking development model

Asynchronous development model



redhat.

Asynchronous development models

Async programming

- **Exists since the early days of computing**
- Better usage of hardware resource, avoid blocking threads

Approaches

- **Callbacks**
- Future / Promise (single value, many read, single write)
- Data flow variables (cell)
- Data streams
- Continuation, Co-Routines

Don’t wait, we will call you…

The Hollywood way

Synchronous

```
public int compute(int a, int b) {
    return ...;
}
```

```
int res = compute(1, 2);
```

Asynchronous

```
public void compute(int a, int b,
  Handler<AsyncResult<Integer>> h){
    // ...
    handler.handle(i);
}
```

```
compute(1, 2, res -> {
    // Called with the
    // async result
});
```

Asynchronous development models

Web server example

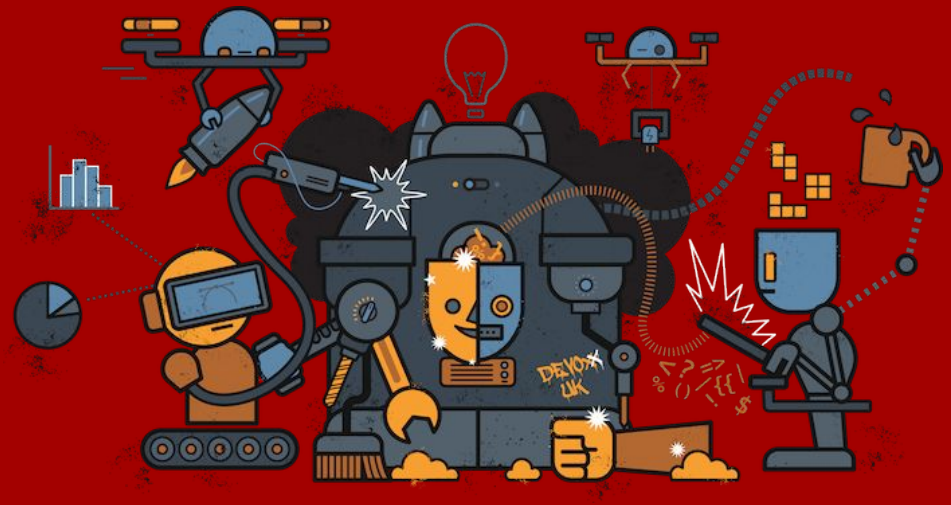
```
vertx.createHttpServer()
    .requestHandler(req ->  // Async reaction
        req.response().end(Thread.currentThread().getName())
    )
    .listen(8080, hopefullySuccessful -> { // Async operation
        //...
    });
```

Callbacks lead to…

Reality check

```
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 Programming - Tame the asynchronous



redhat.

Do we have “Excel” users in the room ?

My Expense Report	
Lunch	15£
Coffee	25£
Drinks	45£
Total	85£

Do we have “Excel” users in the room ?

My Expense Report	
Lunch	15£
Coffee	25£
Drinks	45£
Total	=sum(B2:B4)

Observe

redhat.

Observables / Streams

My Expense Report	
Lunch	15£
Coffee	0£
Drinks	0£
Total	15£

My Expense Report	
Lunch	15£
Coffee	25£
Drinks	0£
Total	40£

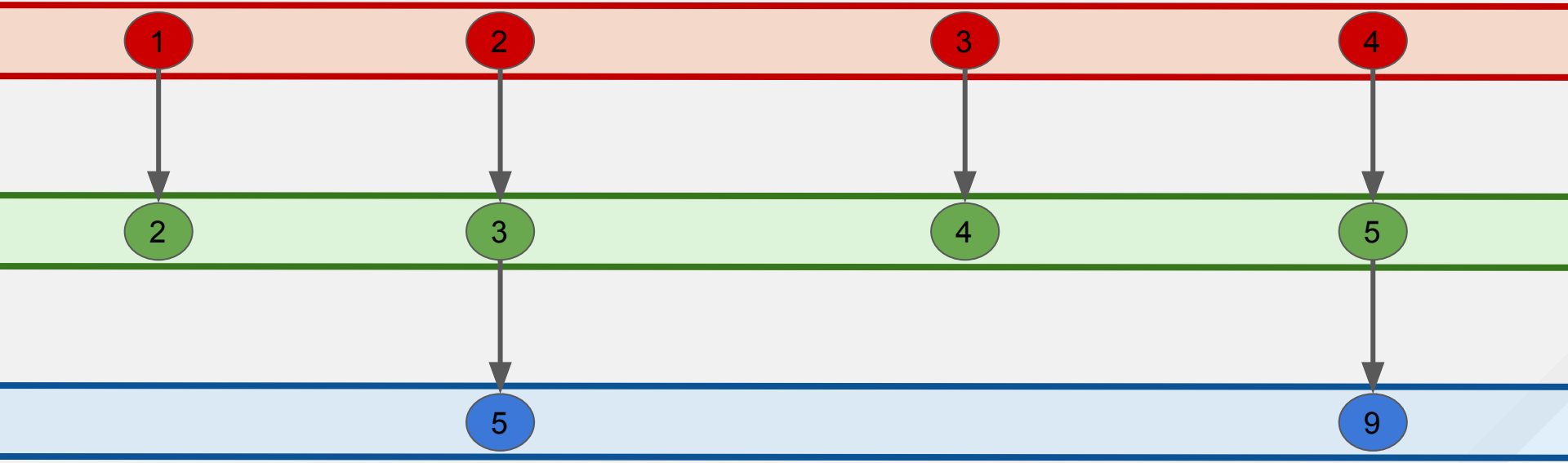
My Expense Report	
Lunch	15£
Coffee	25£
Drinks	45£
Total	85£

time

redhat.

Reactive Programming

Observable and Subscriber



Reactive Extension - RX Java

```
Observable<Integer> obs1 = Observable.range(1, 10);

Observable<Integer> obs2 = obs1.map(i -> i + 1);

Observable<Integer> obs3 = obs2.window(2)
  .flatMap(MathObservable::sumInteger);

obs3.subscribe(
  i -> System.out.println("Computed " + i)
);
```

Reactive types & Asynchronous

Observables - Stream of data, Async reaction

- Bounded or unbounded stream of values
- Data, Error, End of Stream

```
observable.subscribe(
   val -> { /* new value */ },
   error -> { /* failure */ },
   () -> { /* end of data */ }
);
```

Singles - Async operation

- Stream of one value
- Data, Error

```
single.subscribe(
   val -> { /* the value */ },
   error -> { /* failure */ }
);
```

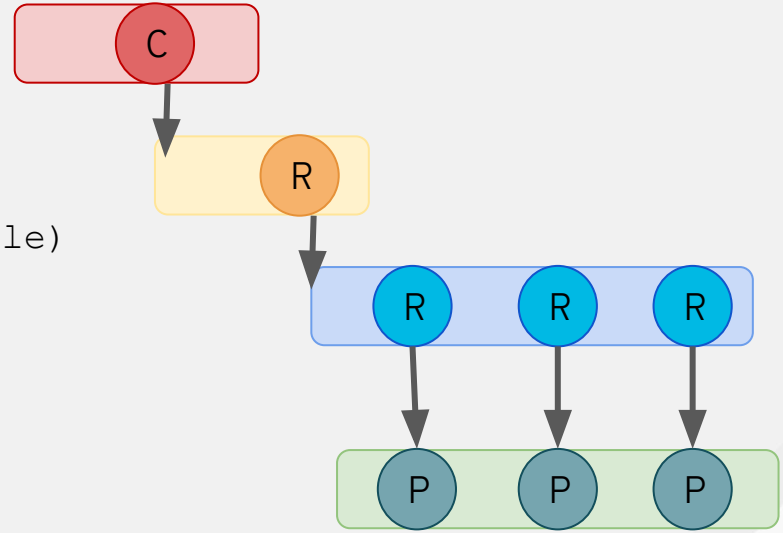
Completables - Async operation (no return)

- Stream without a value
- Completion, Error

```
completable.subscribe(
   () -> { /* completed */ },
   error -> { /* failure */ }
);
```

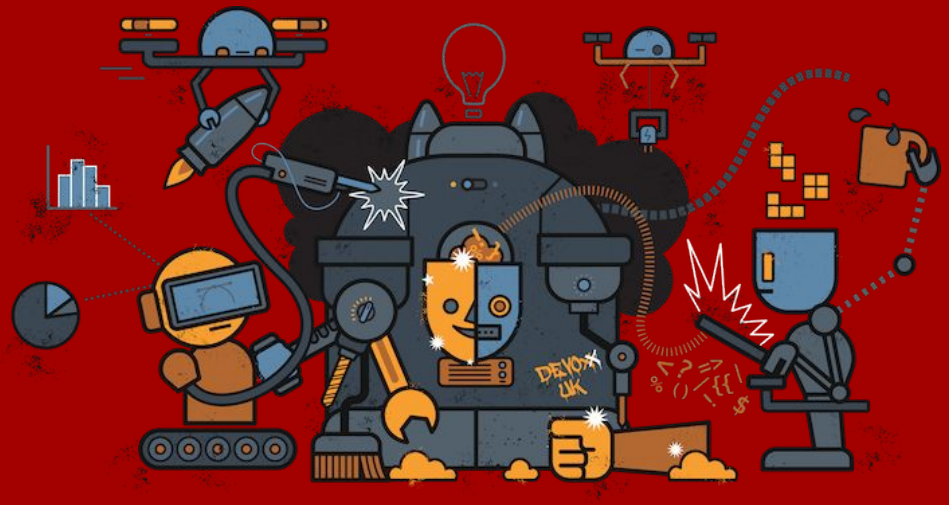
Reactive types & Asynchronous

```
client.rxGetConnection()   // Single(async op)
 .flatMapObservable(conn ->
   conn
     .rxQueryStream("SELECT * from PRODUCTS")
     .flatMapObservable(SQLRowStream::toObservable)
     .doAfterTerminate(conn::close)
 ) // Observable (rows)
 .map(Product::new) // Observable (products)
 .subscribe(System.out::println);
```



redhat.

Back pressure & Reactive streams

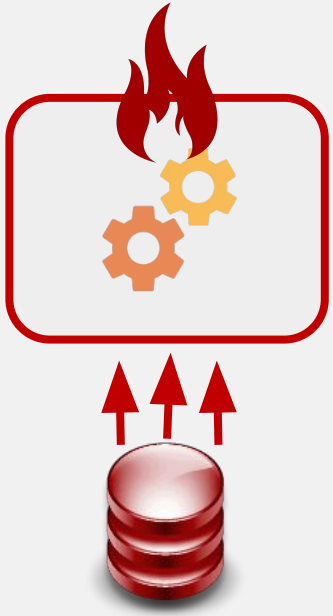


redhat.

What if … the processing can’t keep up

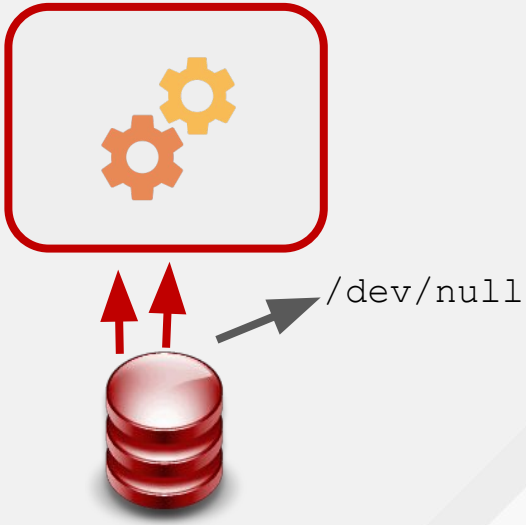
```
public void start() {
  Observable<JsonObject> source = …
  source.
   .concatMap(this::process)
   .subscribe(
    System.out::println,
    System.err::println);
}
```

process takes a long time to return.



When data loss is acceptable

```
public void start() {
  Observable<JsonObject> source = …
  source.
    .onBackpressureDrop()
    .concatMap(this::process)
    .subscribe(...);
}
```



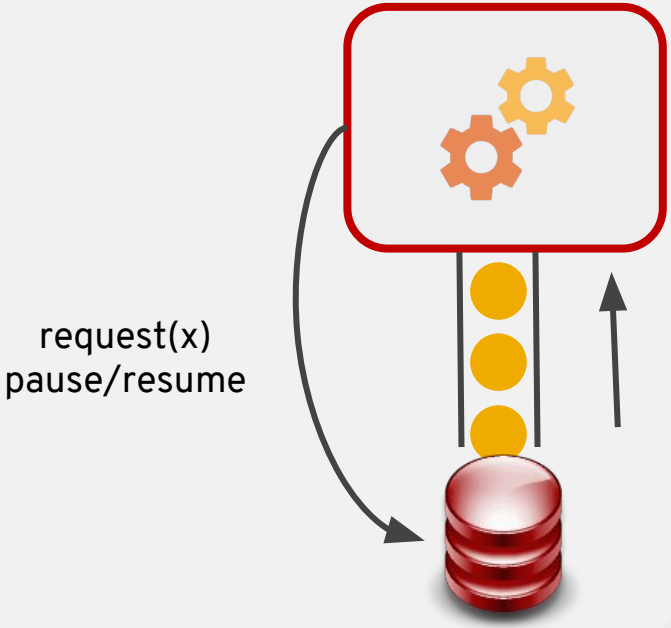
When you control the source

Control protocol between the consumer and the source

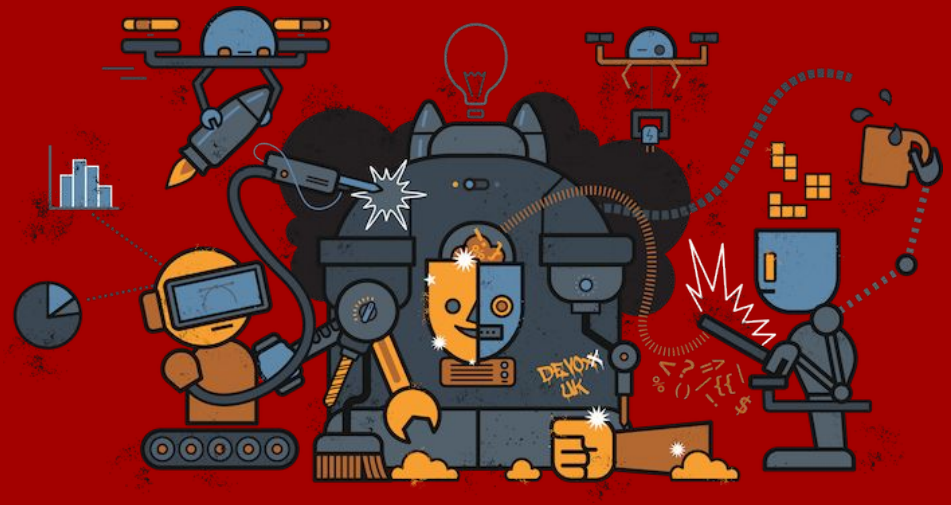
- **The source send data only when requested by the consumer**
- Reverse the control

Approaches

- Reactive Streams (request / subscription) -> Java 9 `Flow`
- Pause / Resume (Vert.x)



Vert.x - Unleash your reactive superpowers



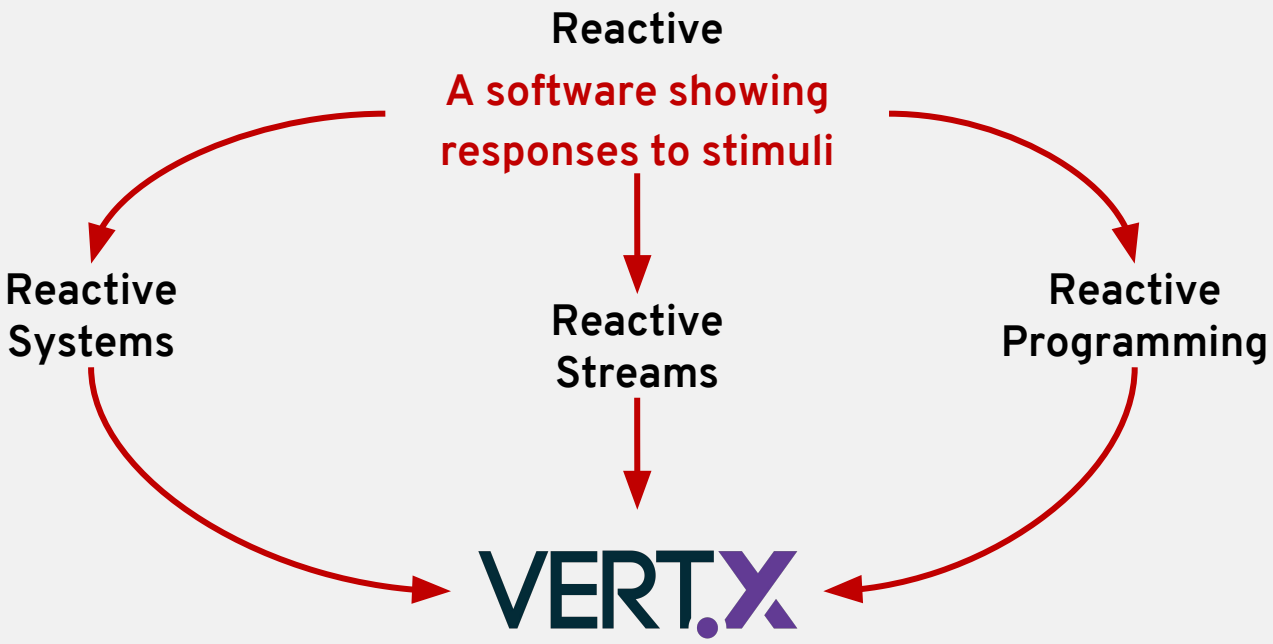
redhat.

Eclipse Vert.x

Vert.x is a toolkit to build distributed and reactive systems

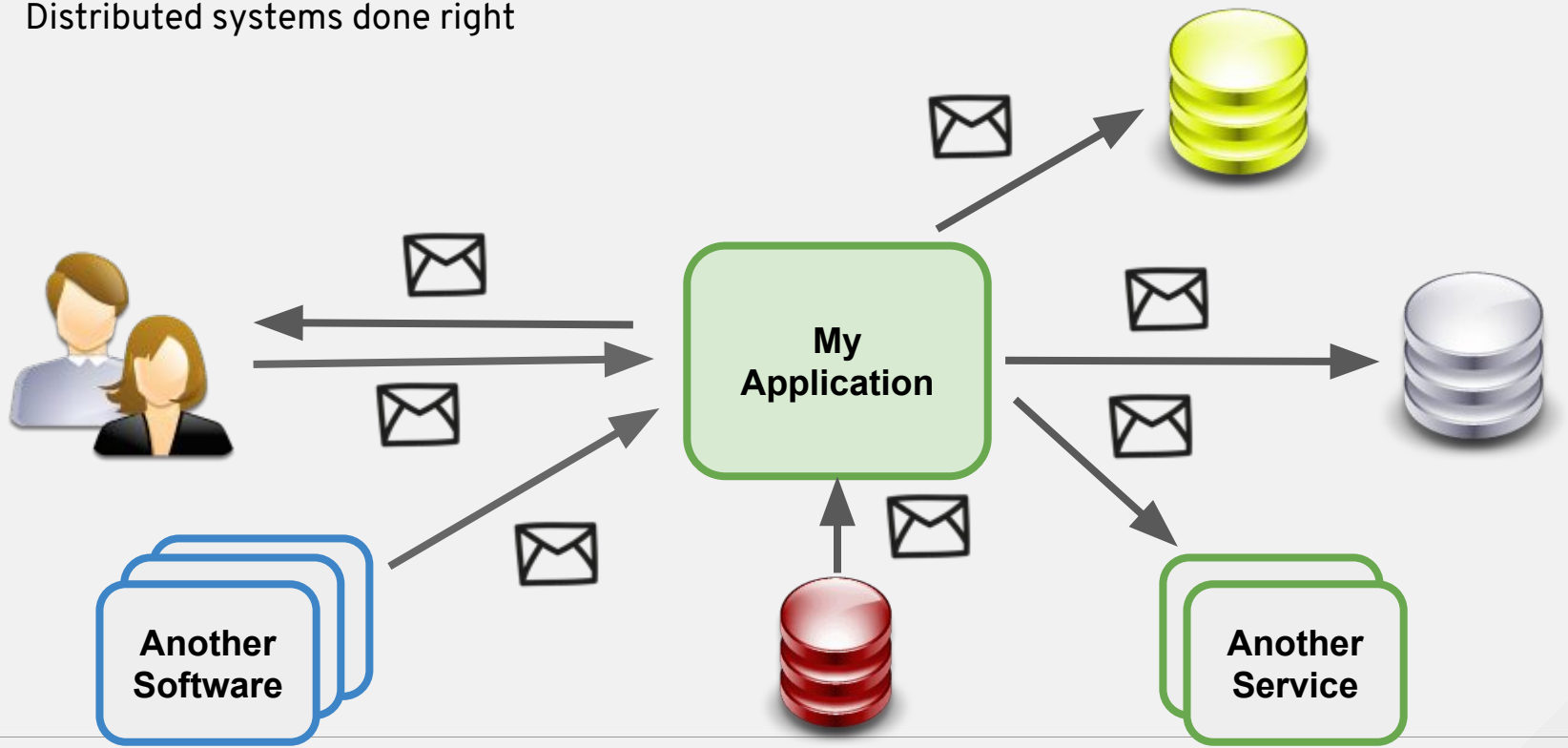
- Designed with **reactive** in mind
- **Asynchronous Non-Blocking development model**
- Simplified concurrency (**event loop**)
- Microservice, Web applications, IOT, API Gateway, High-volume event processing, Full-blown backend message bus

Vert.x - the all-in-one toolkit



Taming the asynchronous

Distributed systems done right



redhat.

Reactive Web Application

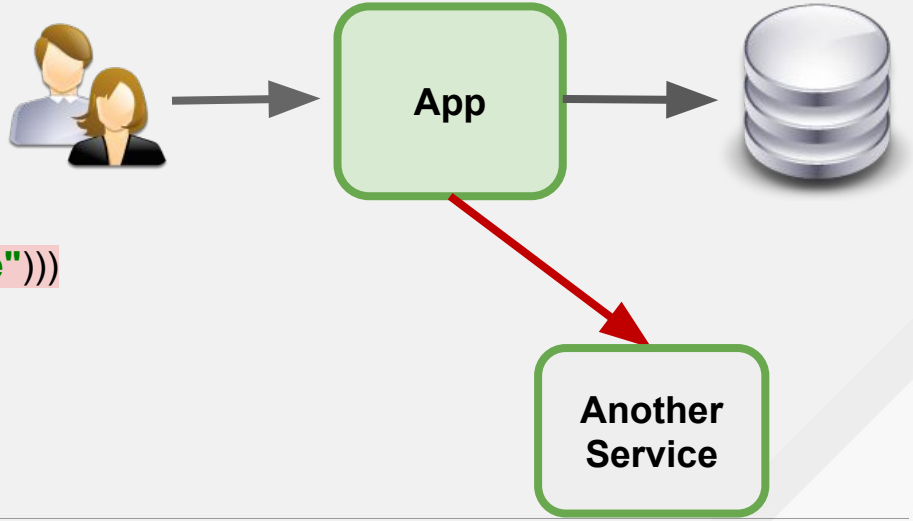
```
private void add(RoutingContext rc) {
  String name = rc.getBodyAsString();
  database.insert(name) // Single (async)
    .subscribe(
      () -> rc.response().setStatusCode(201).end(),
      rc::fail
    );
}

private void list(RoutingContext rc) {
  HttpServerResponse response = rc.response().setChunked(true);
  database.retrieve() // Observable (async)
    .subscribe(
      p -> response.write(Json.encode(p) +" \n\n"),
      rc::fail,
      response::end);
}
```

Orchestrating remote interactions

Sequential composition

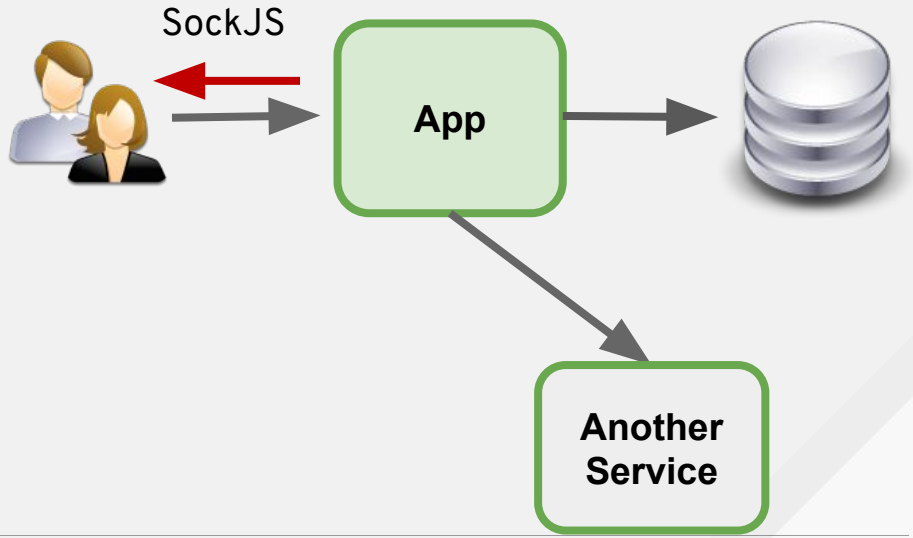
```
WebClient pricer = ...
HttpServerResponse response = rc.response().setChunked(true);
database.retrieve()
  // For each row call...
  .flatMapSingle(p ->
      webClient
          .get("/prices/" + p.getName())
          .rxSend()
          .map(HttpResponse::bodyAsJsonObject)
          .map(json -> p.setPrice(json.getDouble("price")))
  )
  .subscribe(
      p -> response.write(Json.encode(p) + " \n\n"),
      rc::fail,
      response::end);
```



Push data using event bus bridges

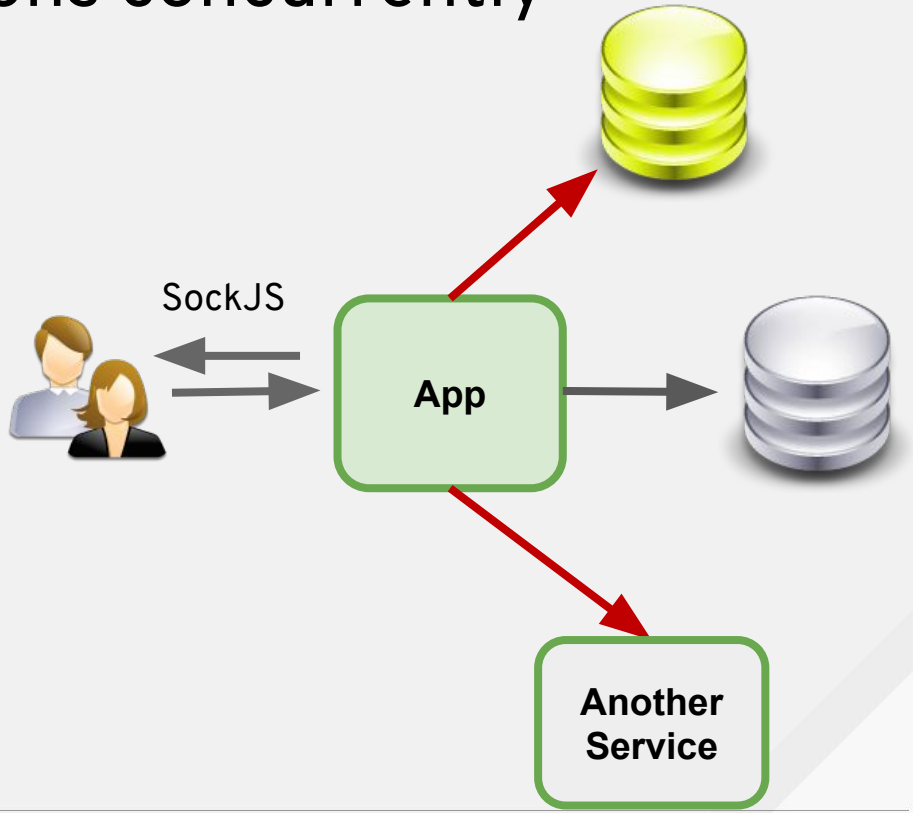
Web Socket, SSE...

```
String name = rc.getBodyAsString().trim();
database.insert(name)
  .flatMap(...)
  .subscribe(
    p -> {
      String json = Json.encode(p);
      rc.response().setStatusCode(201).end(json);
      vertx.eventBus().publish("products", json);
    },
    rc::fail);
```



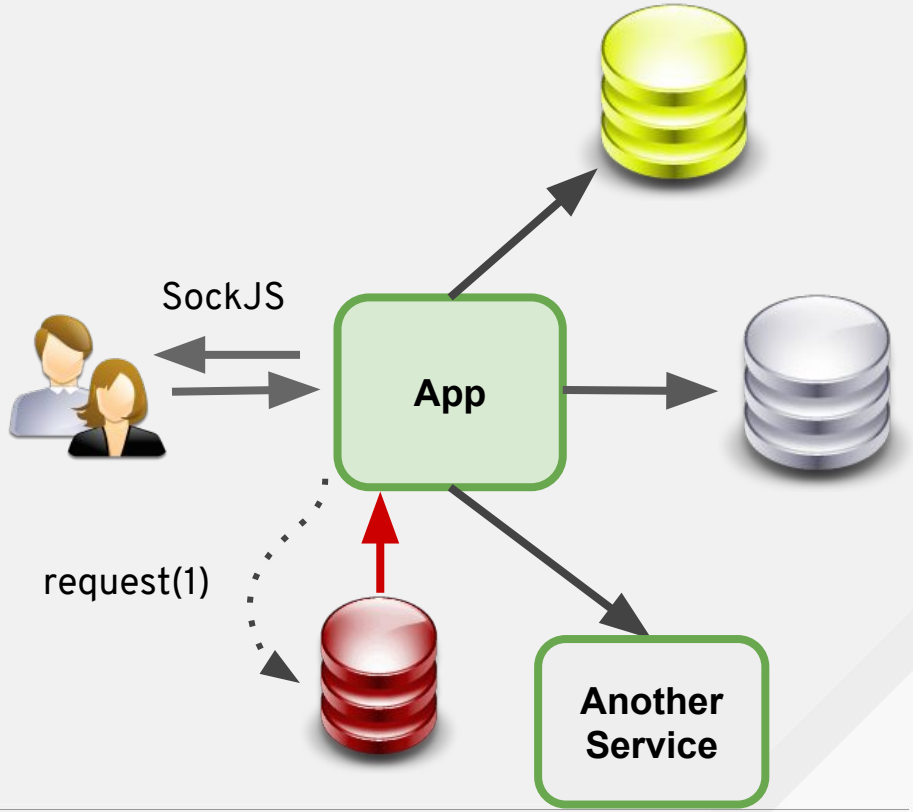
Executing several operations concurrently

```
database.insert(name)
  .flatMap(p -> {
      Single<Product> price = getPriceForProduct(p);
      Single<Integer> audit = sendActionToAudit(p);
      return Single.zip(price, audit, (pr, a) -> pr);
  })
  .subscribe(
      p -> {
          String json = Json.encode(p);
          rc.response().setStatusCode(201).end(json);
          vertx.eventBus().publish("products", json);
      },
      rc::fail);
```

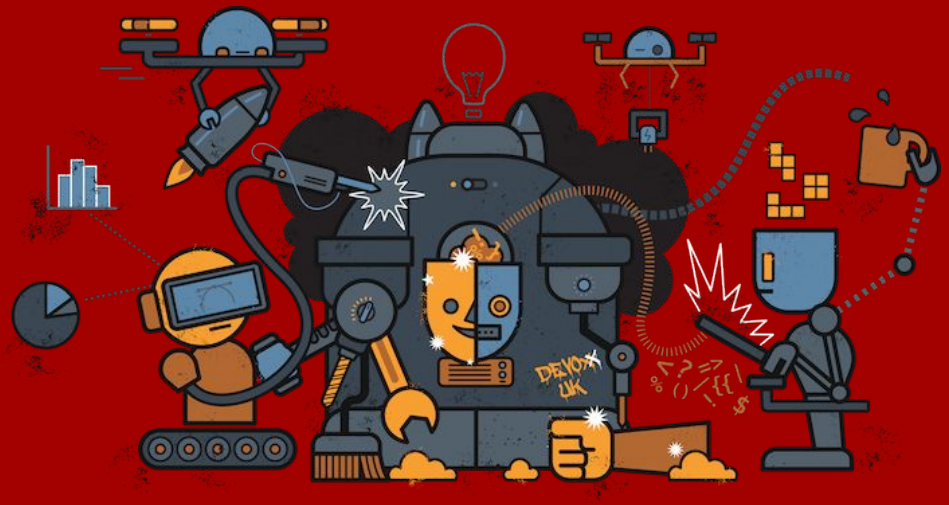


Reactive Streams

```
Publisher<Integer> publisher =
        new RandomIntegerPublisher();
Subscriber<Integer> subscriber =
        new RandomIntegerSubscriber(vertx);
publisher.subscribe(subscriber);
```

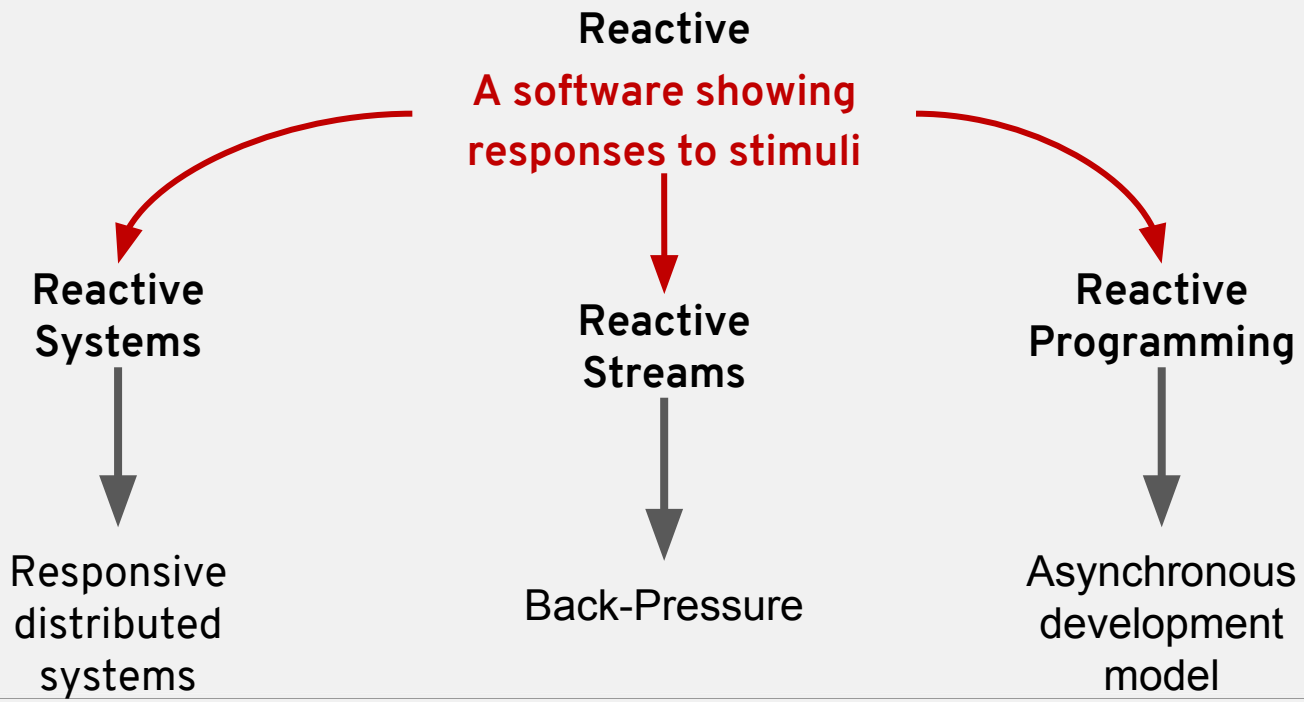


Reactive => Build better systems



redhat.

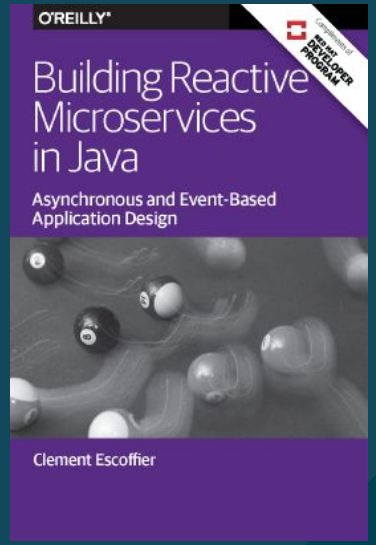
The reactive landscape



All you need is (reactive) love

redhat.

Don’t let a framework lead, you are back in charge



clement.escoffier@redhat.com

@clementplop

@vertx_project

https://groups.google.com/forum/#!forum/vertx

https://developers.redhat.com/promotions/building-reactive-microservices-in-java