Applications réactives avec Eclipse Vert.x

Devoxx France 2017

Julien Ponge
Julien Viet

Julien Ponge

Maitre de Conférences

“Delegated consultant to Red Hat” **on Vert.x** (part-time)

Eclipse Golo + extensive F/OSS background

https://julien.ponge.org/

@jponge

@jponge

https://www.mixcloud.com/hclcast/

Julien Viet

Open source developer for 15+ years

Current **@vertx_project** lead

Principal software engineer at  redhat.

Marseille Java User Group Leader

https://www.julienviet.com/

http://github.com/vietj

@julienviet

https://www.mixcloud.com/cooperdbi/

Agenda

Introduction

Vert.x core 101

Networking with Vert.x

Boiler Vroom 

Guests 

Ecosystem

RxJava

Outro

Eclipse Vert.x

Open source project started in 2012

Created by Tim Fox

Eclipse / Apache licensing

A toolkit for building reactive applications for the JVM

https://vertx.io

vertx_project

Installation

Java 8

Vert.x is a set of jars on Maven Central

Unopinionated : your build, your IDE

CLI `vertx` tool

Let’s build our first reactive app!

Reactive

Non blocking

Event driven

Distributed

Reactive programming

Vert.x

Modular

Minimum dependencies

Composable

Embedded


Java


JS


Groovy


JRuby


ceylon


Scala


Kotlin

Vert.x

Un-opinionated

Classloading free, IoC free

Simple

Flows

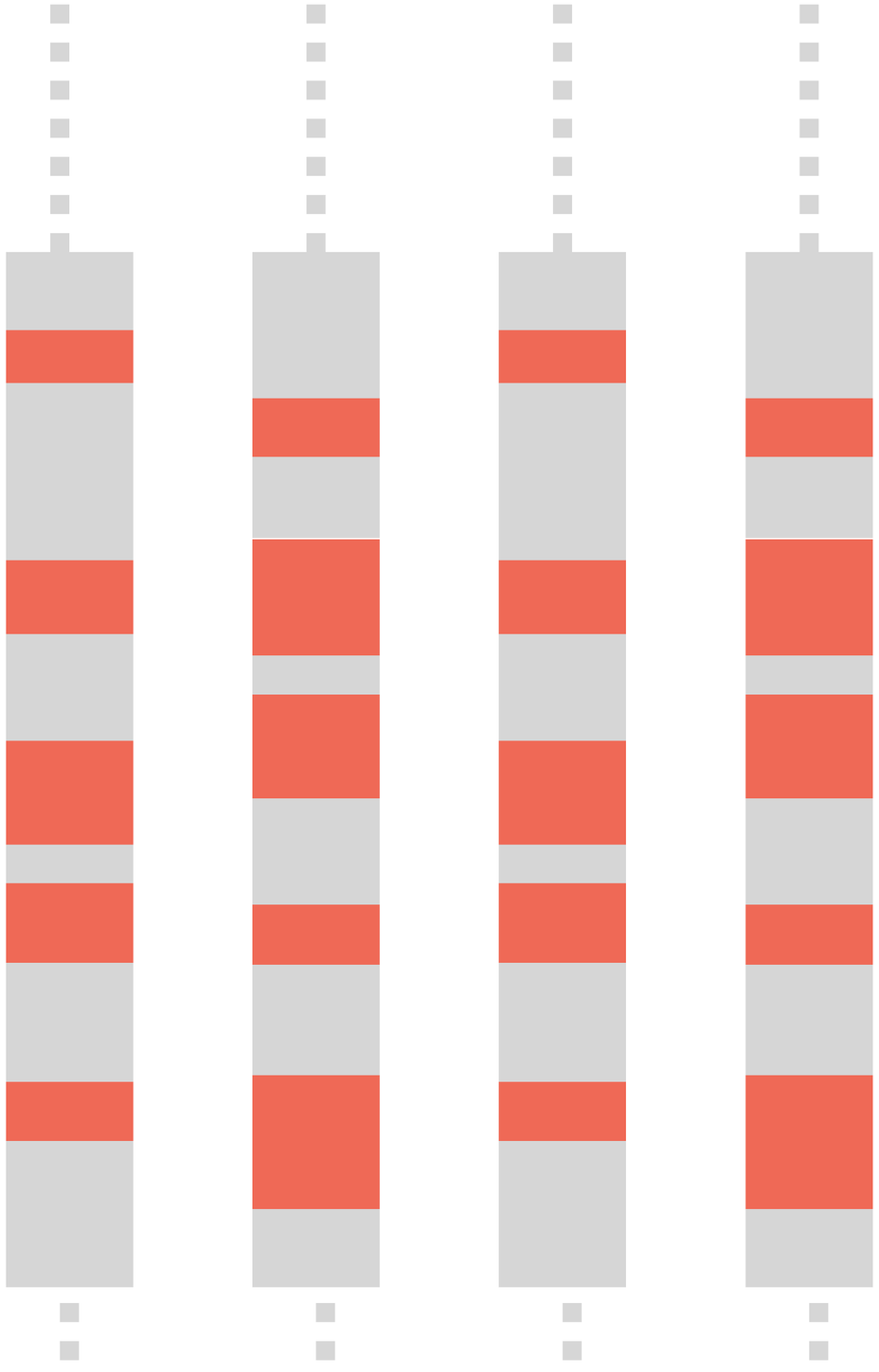
Vert.x core — 101

Outline

- ✓ Vert.x concurrency model
- ✓ Dealing with asynchronous events
- ✓ Message passing on the event bus
- ✓ Failover demo

Vert.x Concurrency Model

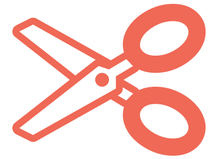
```
while (isRunning) {
  String line = bufferedReader.readLine();
  switch (line.substring(0, 4)) {
    case "ECHO":
      bufferedWriter.write(line);
      break
    //  ...
    // other cases ( ... )
    //  ...
    default:
      bufferedWriter.write("UNKW Unknown command");
  }
}
```



x 1000 =

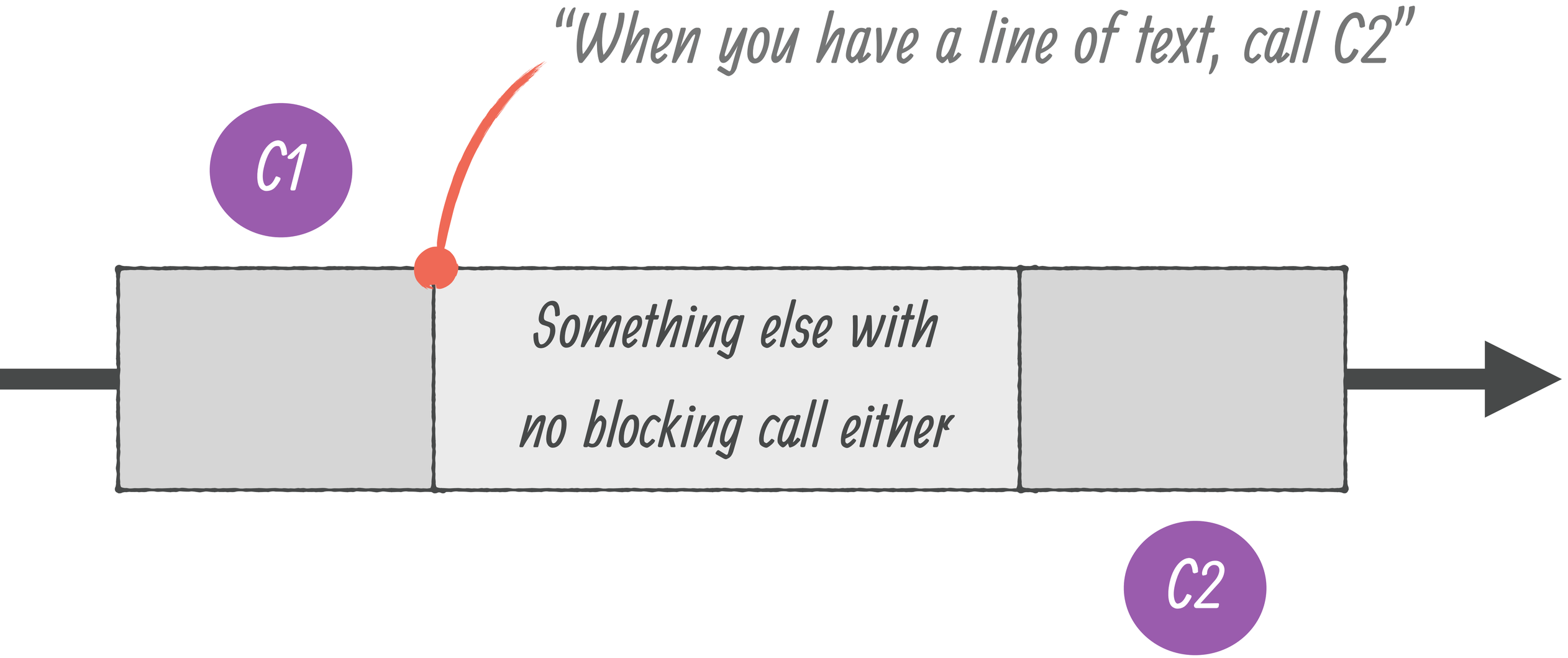
C1

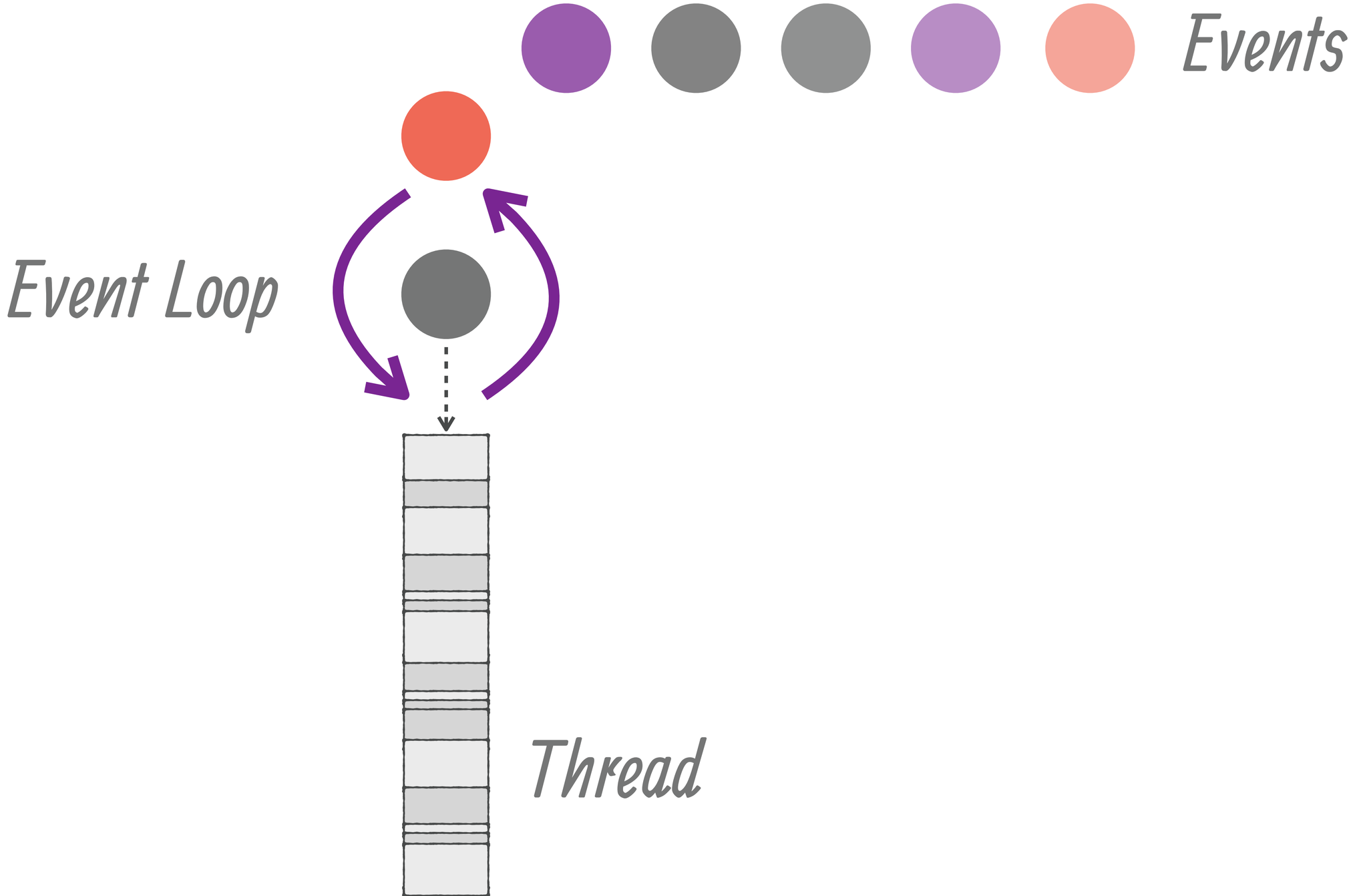
```
String line = bufferedReader.readLine();
```

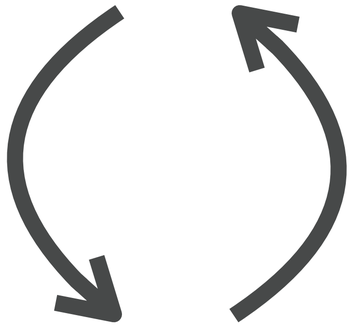


C2

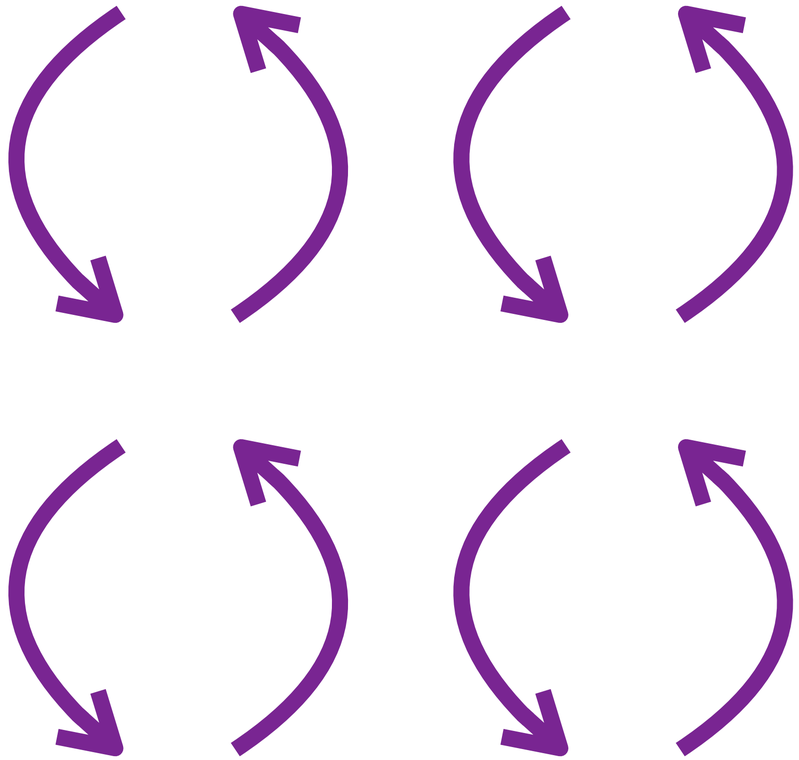
```
switch (line.substring(0, 4)) {
  case "ECHO":
    bufferedWriter.write(line);
    break
  // (…)
```


"When you have a line of text, call C2"
C1
Something else with
no blocking call either
C2

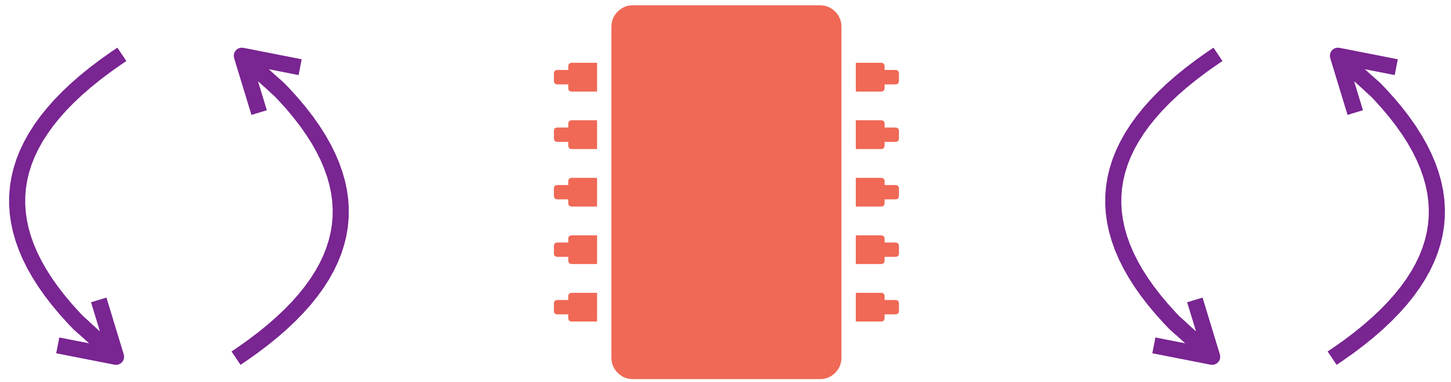

Events
Event Loop
Thread



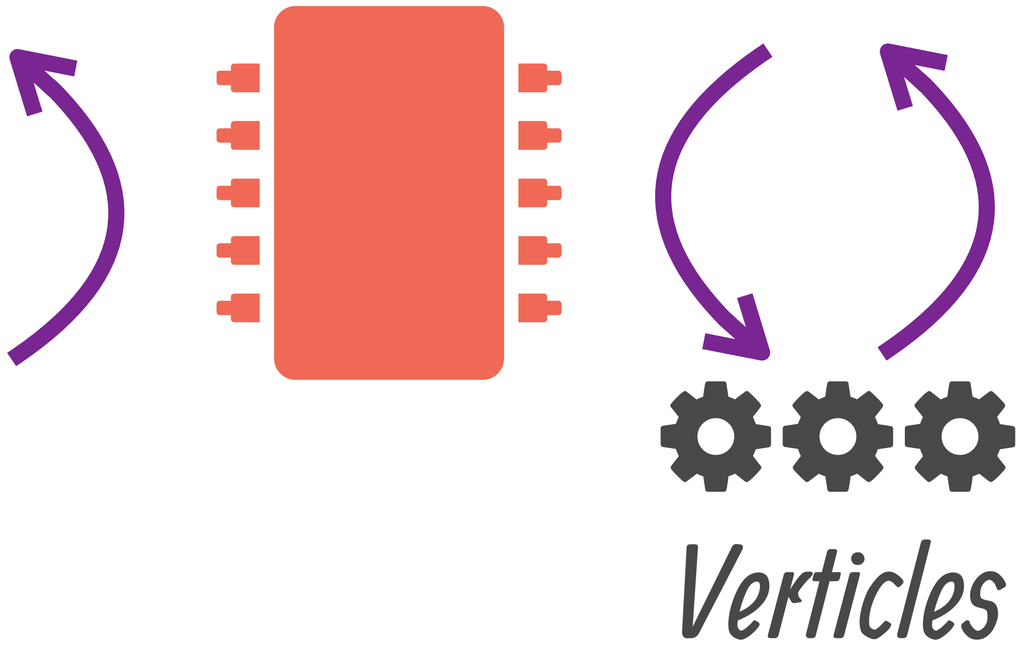
Reactor



Multi-reactor



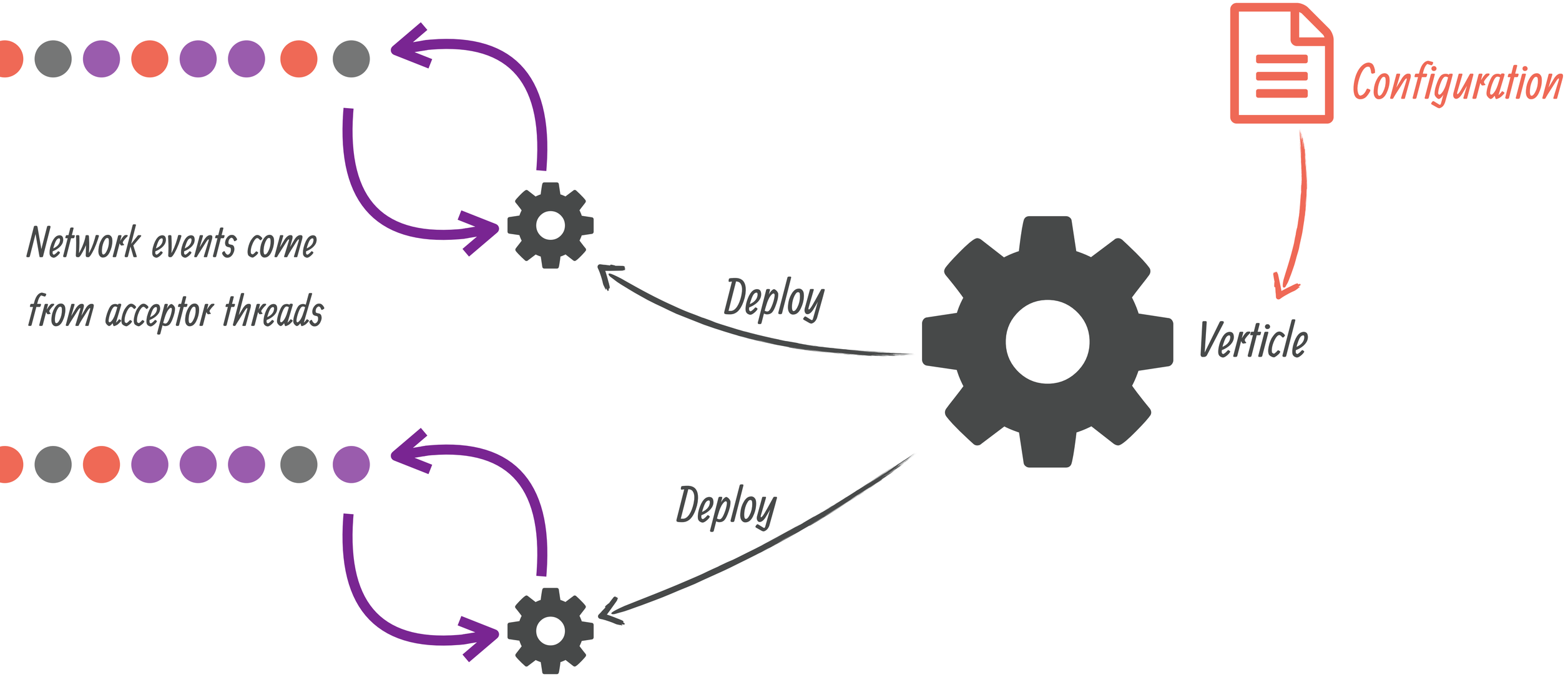
2 event-loops per CPU core by default

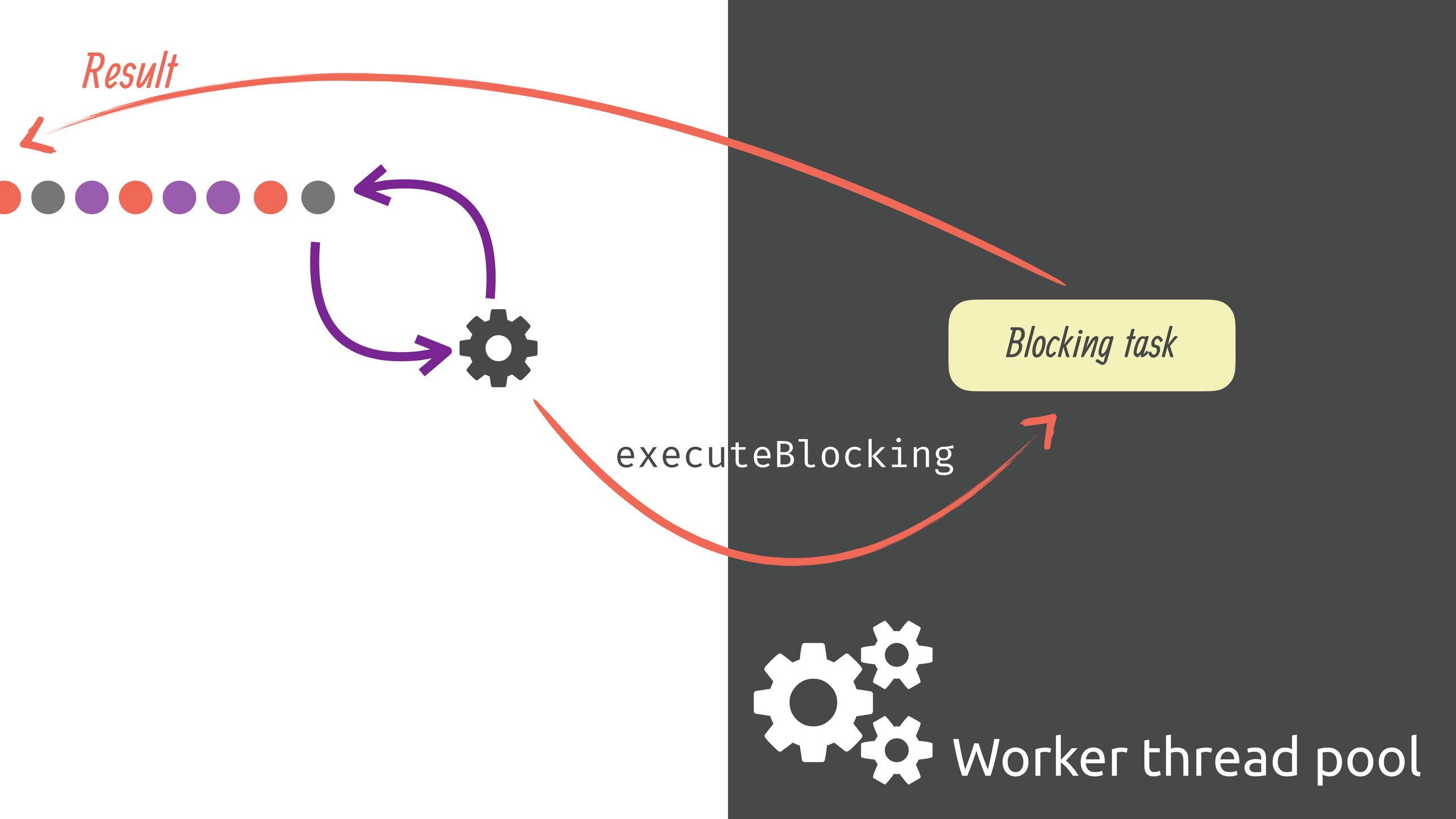



```java
public class SomeVerticle extends AbstractVerticle {

  @Override
  public void start() throws Exception { }

  @Override
  public void stop() throws Exception { }
}
```

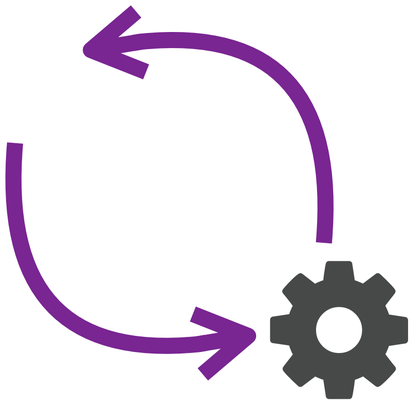
```javascript
exports.vertxStart = function() { }
exports.vertxStop  = function() { }
```

```kotlin
class SomeVerticle : AbstractVerticle() {

    override fun start() { }

    override fun stop() { }
}
```


Configuration
Network events come
from acceptor threads
Deploy
Verticle
Deploy

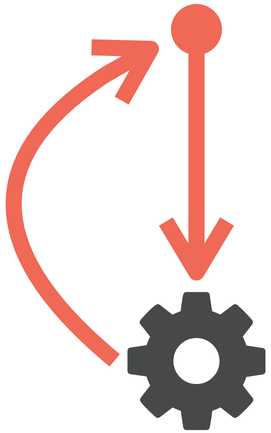

Result
Blocking task
executeBlocking
Worker thread pool

Worker pool



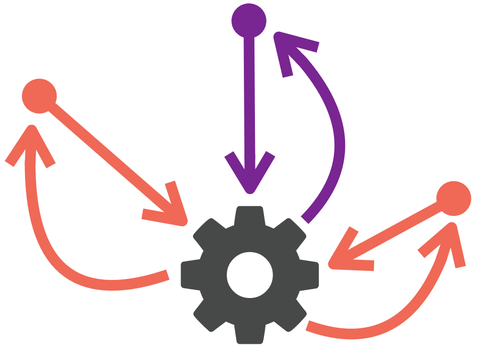
"Regular verticle"

(on event-loop)



Worker verticle

(1 thread)



Multi-thread

worker verticle

Dealing with asynchronous events

```java
@FunctionalInterface
public interface Handler<E> {
  void handle(E event);
}
```

```java
doSomething("localhost", 8080, value → {
  // ( .. )
});
```

AsyncResult<T>

```
if (asyncResult.succeeded()) {
  Object result = asyncResult.result();
  // ( ... )
```

```
} else {
  Throwable t = asyncResult.cause();
  // ( ... )
}
```

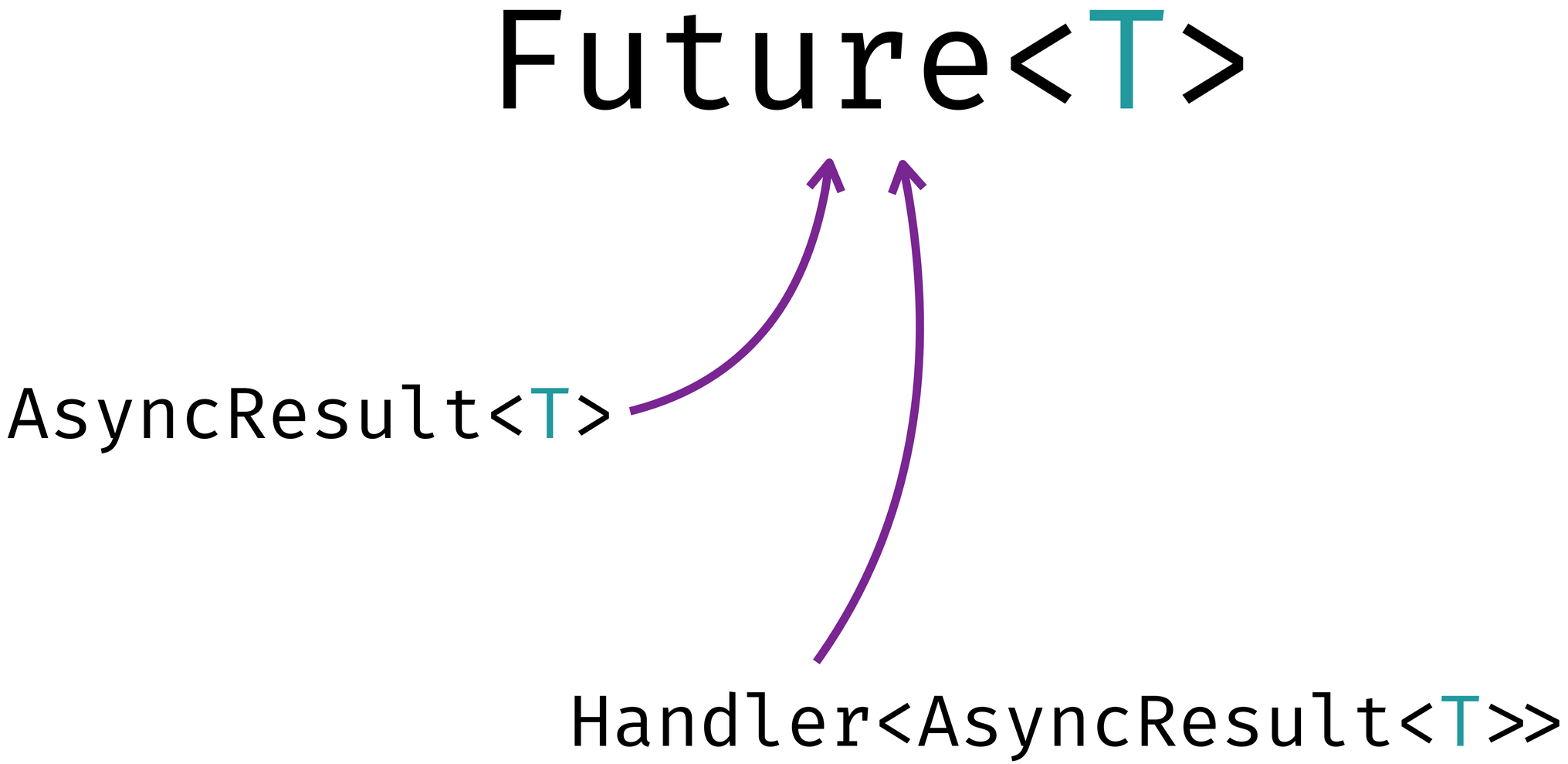
```
vertx.createHttpServer()
  .requestHandler(req → req.response().end("Yo!"))
  .listen(8080, ar → {
    if (ar.failed()) {
      System.err.println("Wooops!");
    }
  });
```

Handler<HttpServerRequest>

```
vertx.createHttpServer()
  .requestHandler(req → req.response().end("Yo!"))
  .listen(8080, ar → {
    if (ar.failed()) {
      System.err.println("Wooops!");
    }
  });
```

Handler<AsyncResult<HttpServer>>

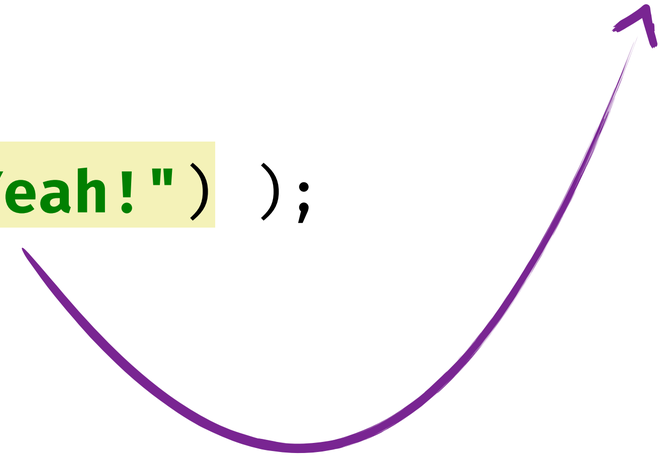
```
vertx.createHttpServer()
  .requestHandler(req → req.response().end("Yo!"))
  .listen(8080, ar → {
    if (ar.failed()) {
      System.err.println("Wooops!");
    }
  });
```


Future<T>
AsyncResult<T>
Handler<AsyncResult<T>>

```
Future<String> aFuture = Future.future();
// ( ... )

// Much much later, in another galaxy
aFuture.complete(“Awesome!");
```

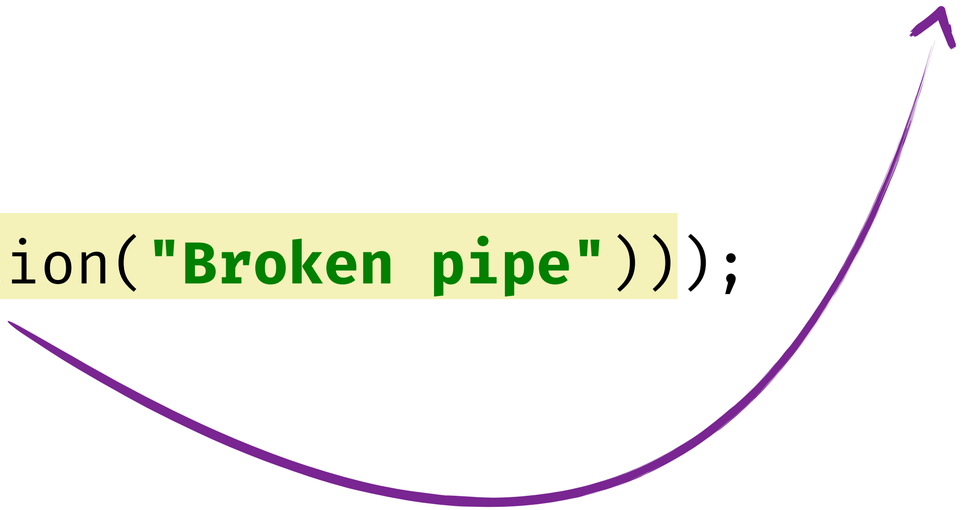
```
Future<String> aFuture = Future.future();
// ( ... )

// Much much later, in another galaxy
aFuture.fail("Wooops");
```

```
public void doSomethingAsync(Handler<AsyncResult<String>> handler) {
  // ( ... )

  handler.handle( Future.succeededFuture("Yeah!") );

  // ( ... )
}
```



Passing a success result

```
public void doSomethingAsync(Handler<AsyncResult<String>> handler) {
  // ( ... )

  handler.handle(
    Future.failedFuture(new IOException("Broken pipe")));

  // ( ... )
}
```

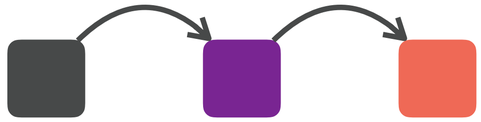


Passing a failure result

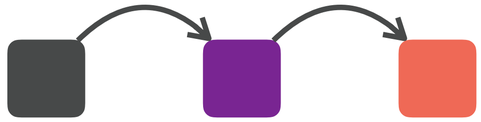
```
Handler<AsyncResult<String>> completer = aFuture.completer();
```

An async result handler

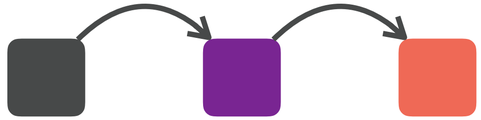
that completes a future



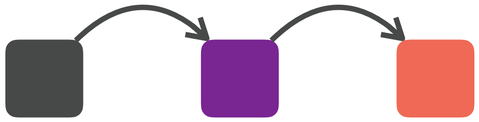
```
Future<String> f1 = Future.future();

Future<Integer> f2 = f1
  .compose(str → Future.succeededFuture(str.length()))
  .map(n → n * 2)
  .otherwise(0)
  .setHandler(ar → {
    if (ar.succeeded()) {
      System.out.println(ar.result());
    } else {
      ar.cause().printStackTrace();
    }
  });
```



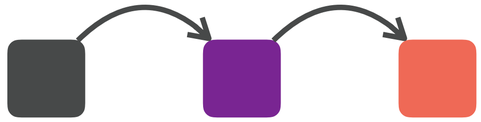
```
Future<String> f1 = Future.future();

Future<Integer> f2 = f1
  .compose(str → Future.succeededFuture(str.length()))
  .map(n → n * 2)
  .otherwise(0)
  .setHandler(ar → {
    if (ar.succeeded()) {
      System.out.println(ar.result());
    } else {
      ar.cause().printStackTrace();
    }
  });
```



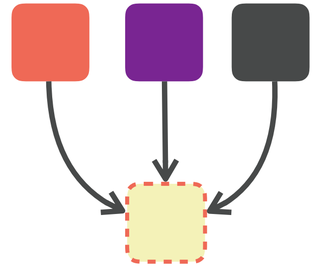
```
Future<String> f1 = Future.future();

Future<Integer> f2 = f1
  .compose(str → Future.succeededFuture(str.length()))
  .map(n → n * 2)
  .otherwise(0)
  .setHandler(ar → {
    if (ar.succeeded()) {
      System.out.println(ar.result());
    } else {
      ar.cause().printStackTrace();
    }
  });
```



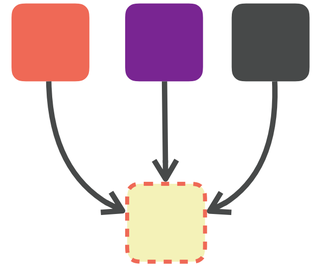
```
Future<String> f1 = Future.future();

Future<Integer> f2 = f1
  .compose(str → Future.succeededFuture(str.length()))
  .map(n → n * 2)
  .otherwise(0)
  .setHandler(ar → {
    if (ar.succeeded()) {
      System.out.println(ar.result());
    } else {
      ar.cause().printStackTrace();
    }
  });

f1.complete("abc");
```



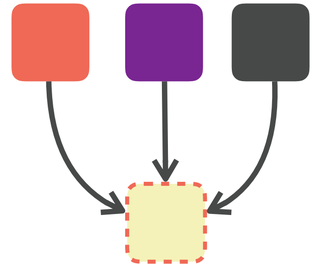
```
Future<String> f1 = Future.future();

Future<Integer> f2 = f1
  .compose(str → Future.succeededFuture(str.length()))
  .map(n → n * 2)
  .otherwise(0)
  .setHandler(ar → {
    if (ar.succeeded()) {
      System.out.println(ar.result());
    } else {
      ar.cause().printStackTrace();
    }
  });

f1.fail(new IllegalStateException("Just won't do"));
```



```
List<Future> futures = Arrays.asList(
  Future.succeededFuture("Ok"),
  Future.failedFuture("Bam"),
  Future.succeededFuture("Great")
);

CompositeFuture.all(futures);
```



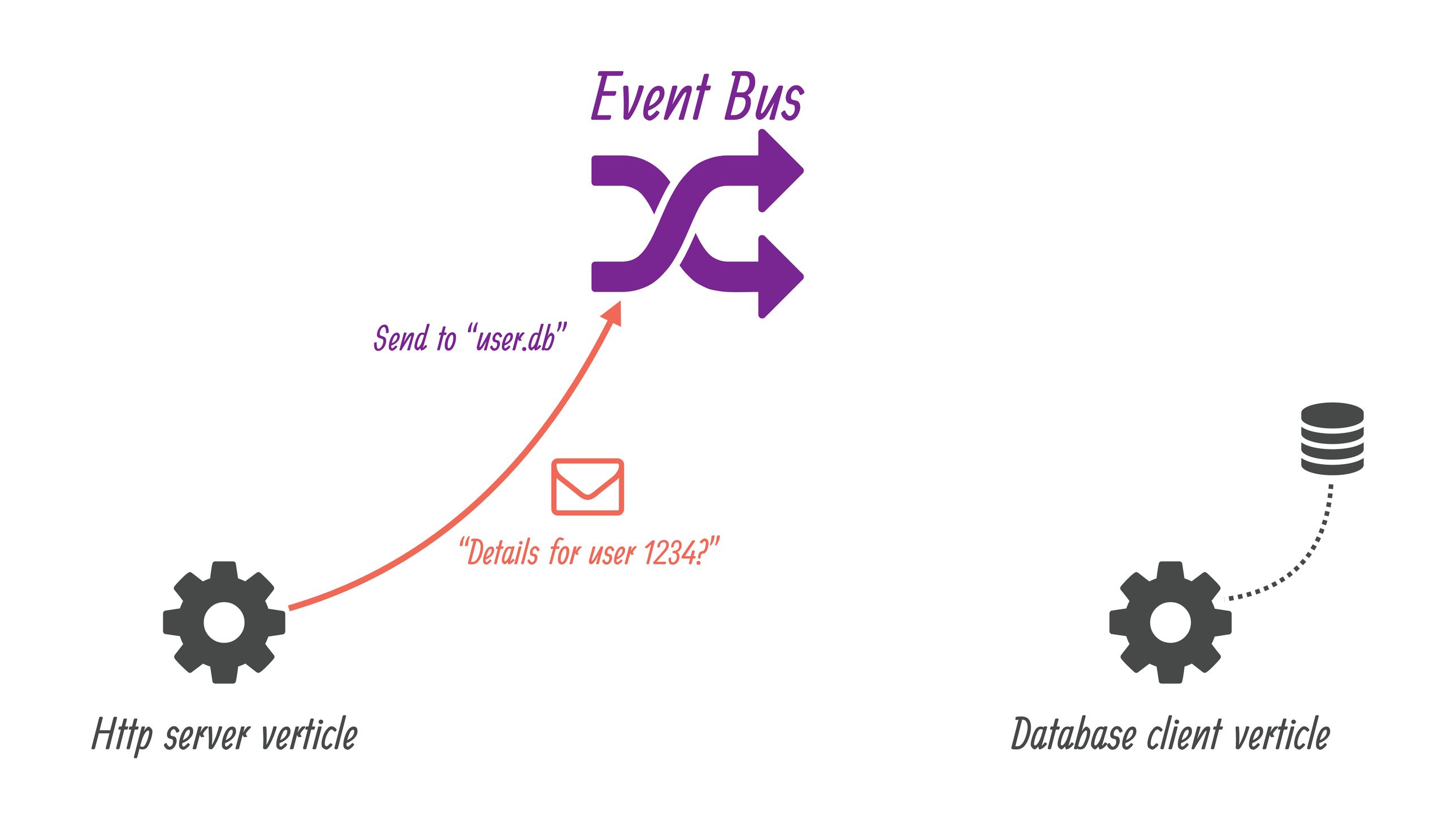
```
List<Future> futures = Arrays.asList(
  Future.succeededFuture("Ok"),
  Future.failedFuture("Bam"),
  Future.succeededFuture("Great")
);

CompositeFuture.join(futures);
```

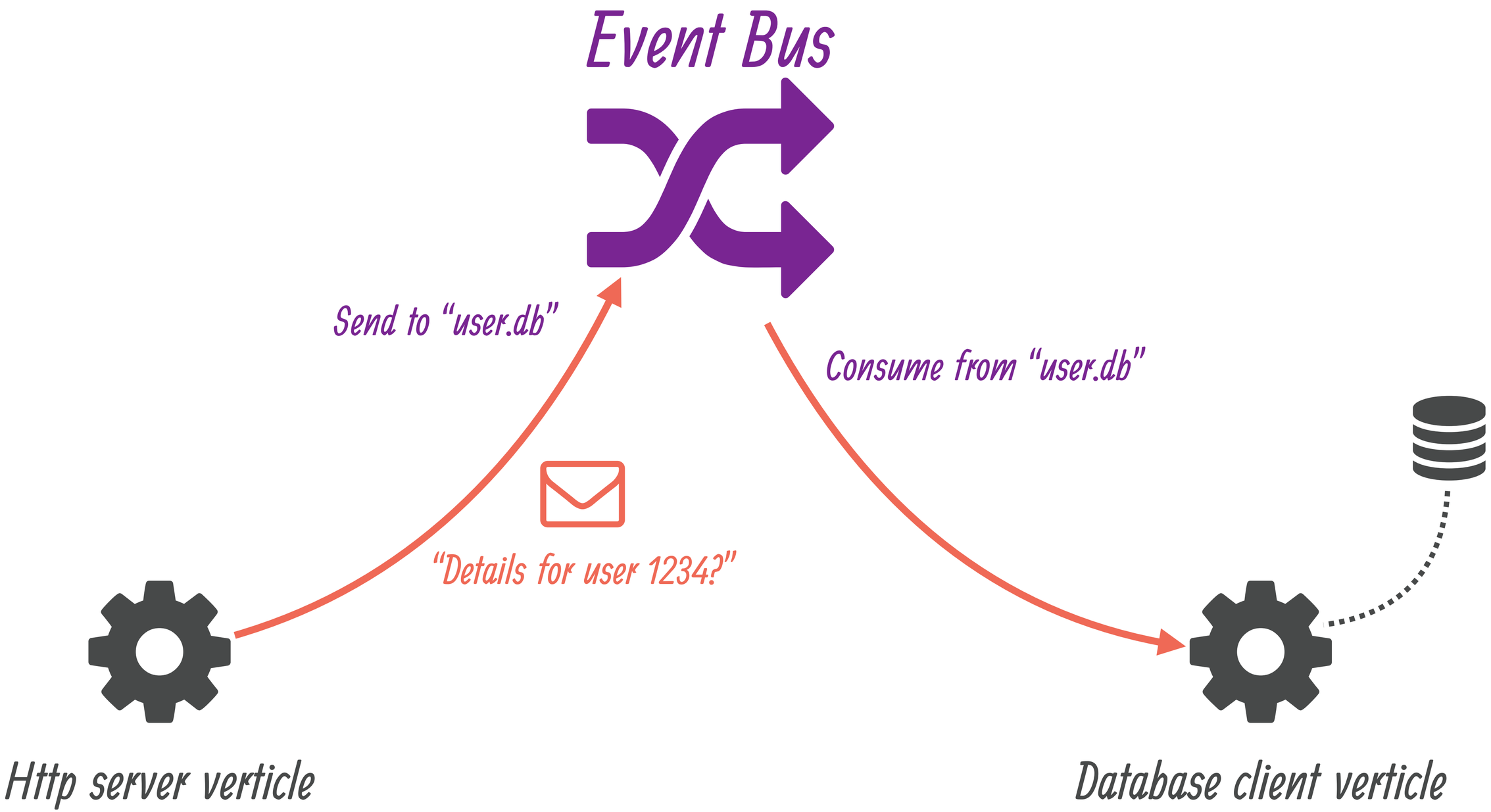


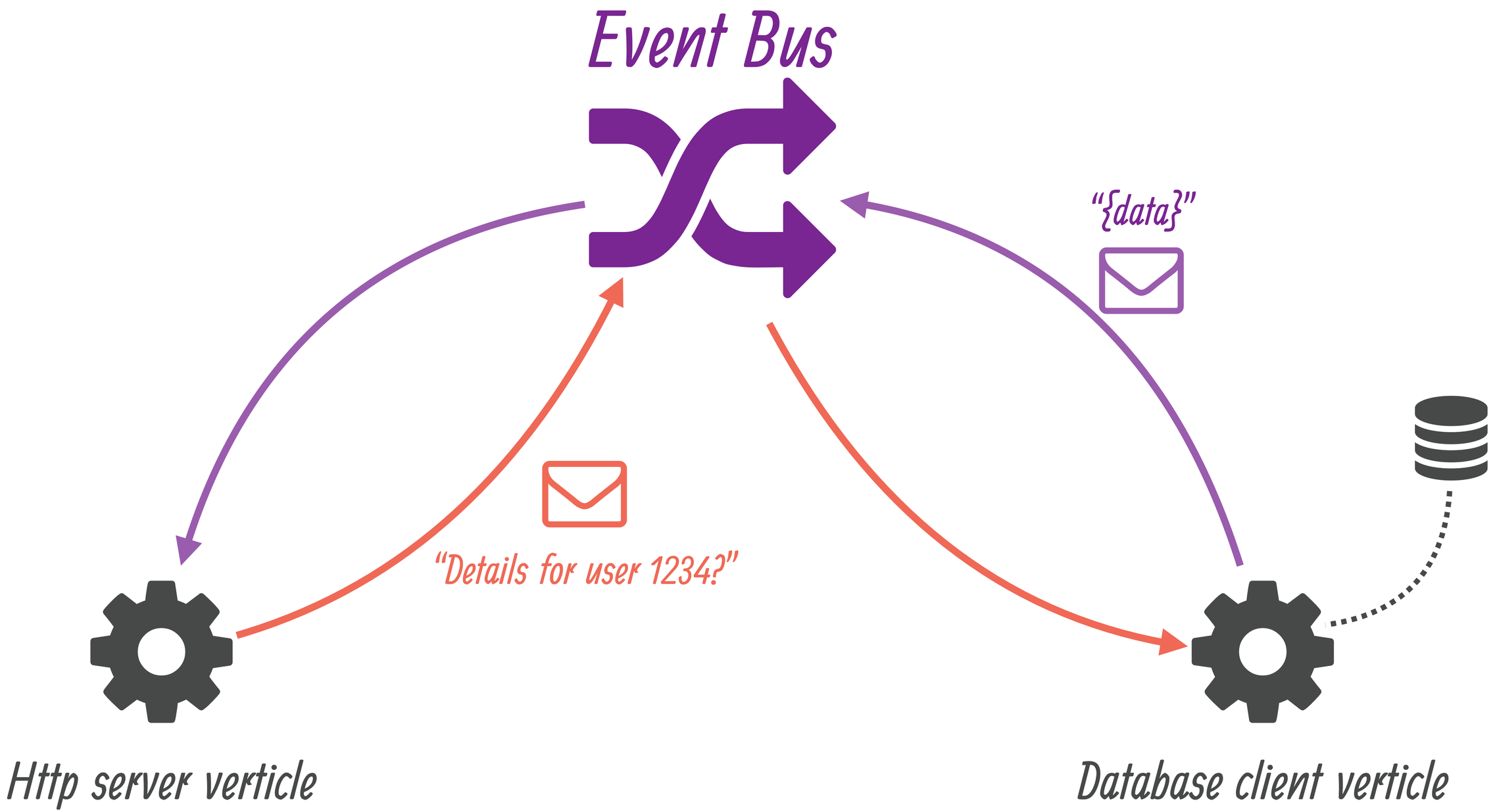
```
List<Future> futures = Arrays.asList(
  Future.succeededFuture("Ok"),
  Future.failedFuture("Bam"),
  Future.succeededFuture("Great")
);

CompositeFuture.any(futures);
```

Message passing on the event bus


Http server verticle
Database client verticle


Event Bus
Send to "user.db"
"Details for user 1234?"
Http server verticle
Database client verticle


Event Bus
Send to "user.db"
Consume from "user.db"
"Details for user 1234?"
Http server verticle
Database client verticle


Event Bus
"{data}"
"Details for user 1234?"
Http server verticle
Database client verticle

Asynchronous messaging

`“foo.bar”, “foo-bar”, “foo/bar”, …`

Point to point (with possible response back)
Publish / subscribe

Distributed across Vert.x nodes

Hazelcast, Ignite, Infinispan, …

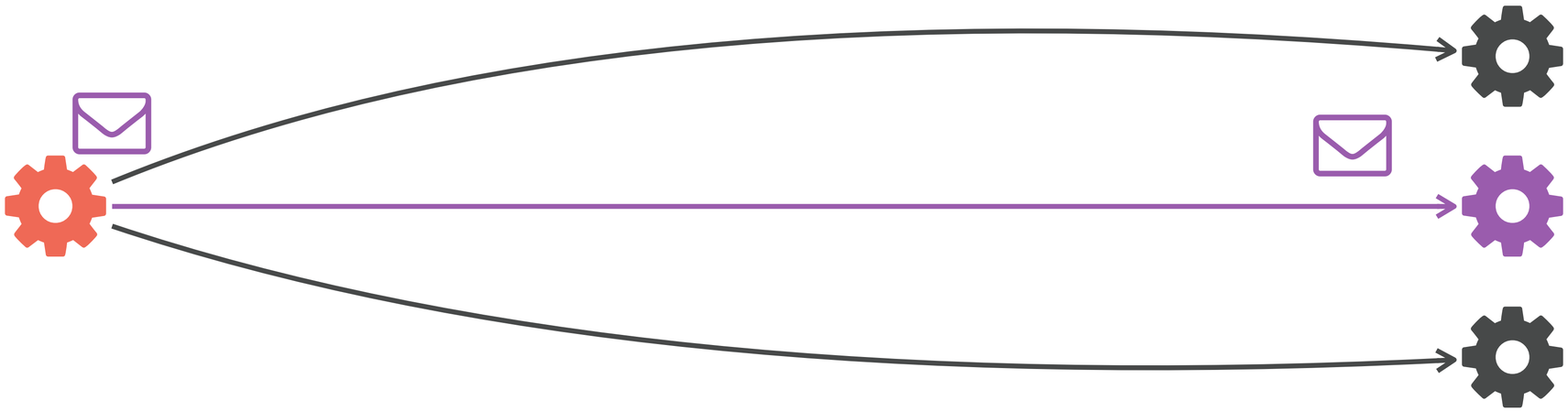
TCP bridge interface

Go, Python, C, JavaScript, Swift, C#, …

SockJS bridge

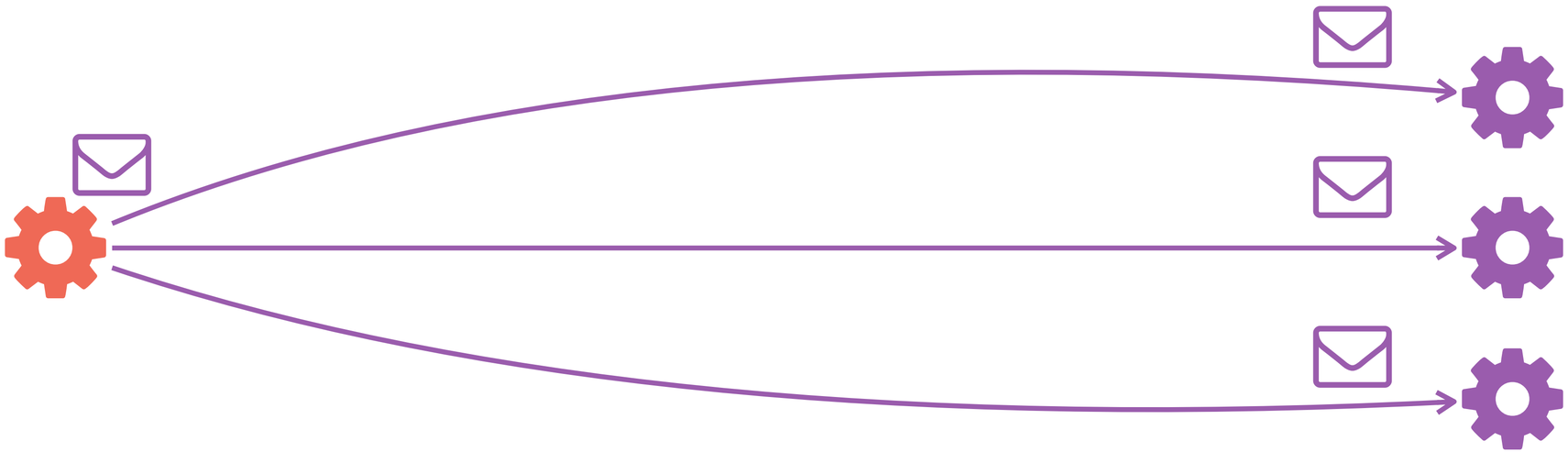
Seamless frontend / backend messaging

```
vertx.setPeriodic(1000, id → {
  vertx.eventBus().send("a.b.c", "tick");
});
```



```
vertx.eventBus().consumer("a.b.c", message → {
  System.out.println(message.body());
});
```

```
vertx.setPeriodic(1000, id → {
  vertx.eventBus().publish("a.b.c", "tick");
});
```



```
vertx.eventBus().consumer("a.b.c", message → {
  System.out.println(message.body());
});
```

Headers
DeliveryOptions (e.g., timeouts)

Body

Address
Reply address

“Primitive” types
String, int, double, …

JSON Object/Array
Polyglot applications, clean boundaries

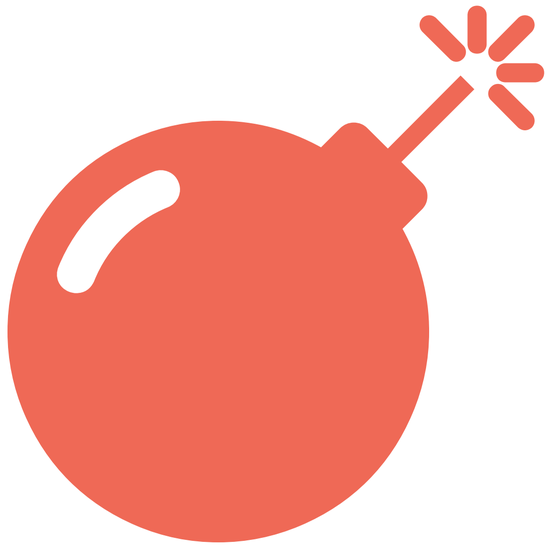
Custom codecs
For advanced usages

```
switch (message.headers().get("action")) {
  case "all-pages":
    fetchAllPages(message);
    break;
  case "get-page":
    fetchPage(message);
    break;
  case "create-page":
    createPage(message);
    break;
  case "save-page":
    savePage(message);
    break;
  case "delete-page":
    deletePage(message);
    break;
  default:
    message.fail(ErrorCodes.BAD_ACTION.ordinal(), "Bad action: " + action);
}
```

```
private void deletePage(Message<JsonObject> message) {
  dbClient.getConnection(car → {
    if (car.succeeded()) {
      SQLConnection connection = car.result();
      JsonArray data = new JsonArray().add(message.body().getString("id"));
      connection.updateWithParams(sqlQueries.get(SqlQuery.DELETE_PAGE), data, res → {
        connection.close();
        if (res.succeeded()) {
          message.reply(new JsonObject().put("result", "ok"));
        } else {
          reportQueryError(message, res.cause());
        }
      });
    } else {
      reportQueryError(message, car.cause());
    }
  });
}
```

```
switch (message.headers().get("action")) {
  case "all-pages":
    fetchAllPages(message);
    break;
  case "get-page":
    fetchPage(message);
    break;
  case "create-page":
    createPage(message);
    break;
  case "save-page":
    savePage(message);
    break;
  case "delete-page":
    deletePage(message);
    break;
  default:
    message.fail(ErrorCodes.BAD_ACTION.ordinal(), "Bad action: " + action);
}
```

If lots of actions...



Proxy + handler source code will be generated

```
@ProxyGen
public interface WikiDatabaseService {

// ( ... )

  @Fluent
  WikiDatabaseService savePage(int id, String markdown,
                               Handler<AsyncResult<Void>> resultHandler);

  @Fluent
  WikiDatabaseService deletePage(int id,
                                 Handler<AsyncResult<Void>> resultHandler);


  static WikiDatabaseService createProxy(Vertx vertx, String address) {
    return new WikiDatabaseServiceVertxEBProxy(vertx, address);
  }
// ( ... )
}
```

Parameters from a JSON document

Handlers for replies

Generated proxy

```
dbService = WikiDatabaseService.createProxy(vertx, "wikidb.queue");
```

```
private void pageDeletionHandler(RoutingContext context) {
  dbService.deletePage(Integer.valueOf(context.request().getParam("id")), reply → {
    if (reply.succeeded()) {
      context.response().setStatusCode(303);
      context.response().putHeader("Location", "/");
      context.response().end();
    } else {
      context.fail(reply.cause());
    }
  });
}
```

(demo — failover)

Networking with Vert.x

Networking with Vert.x

Vert.x Core

- TCP, HTTP/1, HTTP/2, WebSocket, UDP, DNS

Vert.x Web

- SockJS

MQTT Server

Networking with Vert.x

SSL/TLS

Native SSL

Asynchronous DNS resolver

Proxy socks4, socks5, HTTP connect

Metrics

HTTP with Vert.x

Server, client

HTTP/1, HTTP/2

Websockets

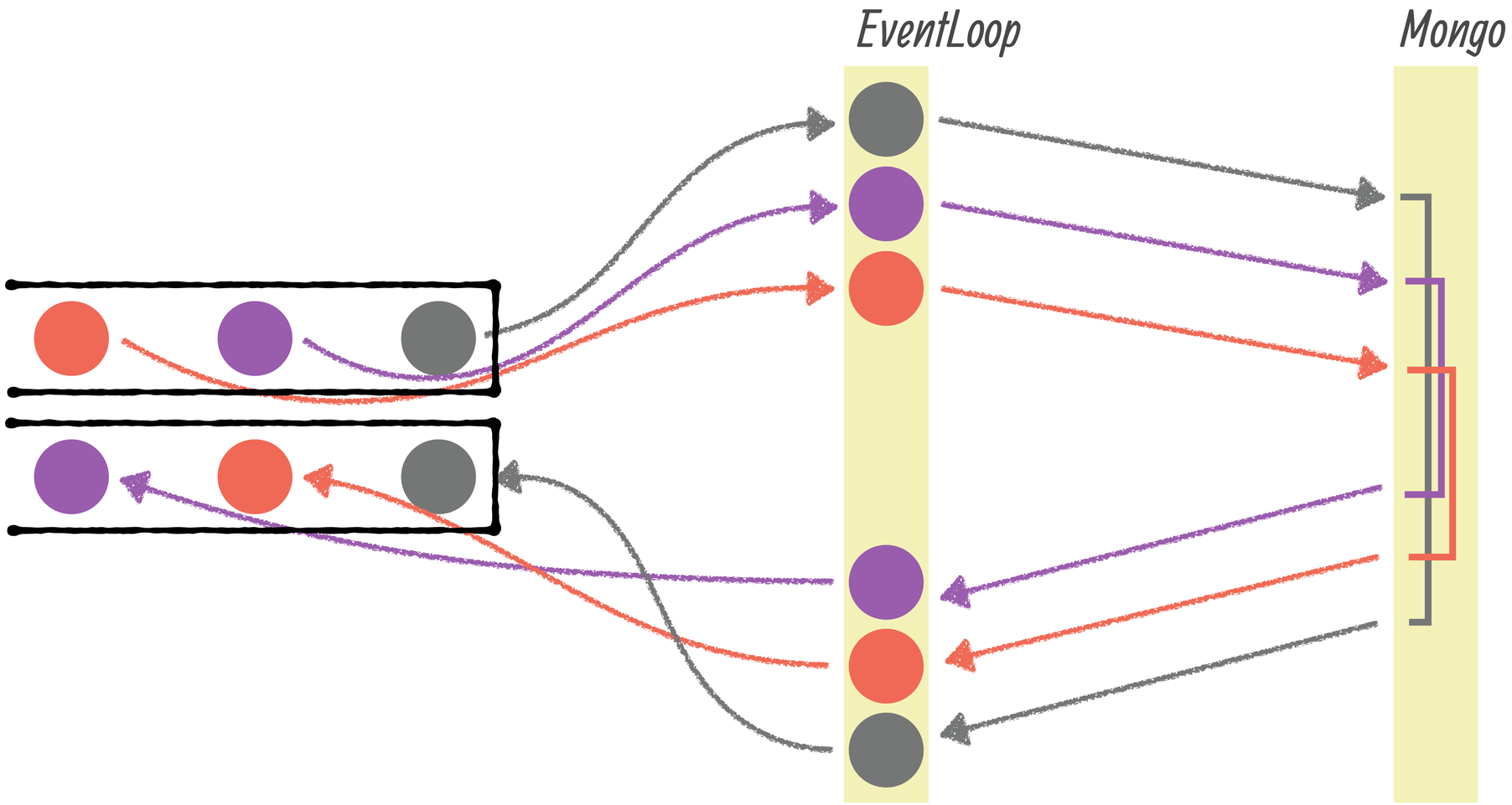
SockJS

Persistent connections

Typical HTTP server

```
MongoClient client = MongoClient.createNonShared(vertx, getConfig());
HttpServer server = vertx.createHttpServer();

server.requestHandler(request -> {
  if (request.path().equals("/document")) {
    client.findOne("docs", QUERY, fieldsOf(request), ar -> {
      if (ar.succeeded()) {
        String json = ar.result().encode();
        request.response().end(json);
      } else {
        request.response().setStatusCode(500).end();
      }
    });
  } else { /* … */ }
});

server.listen(8080);
```

Server concurrency

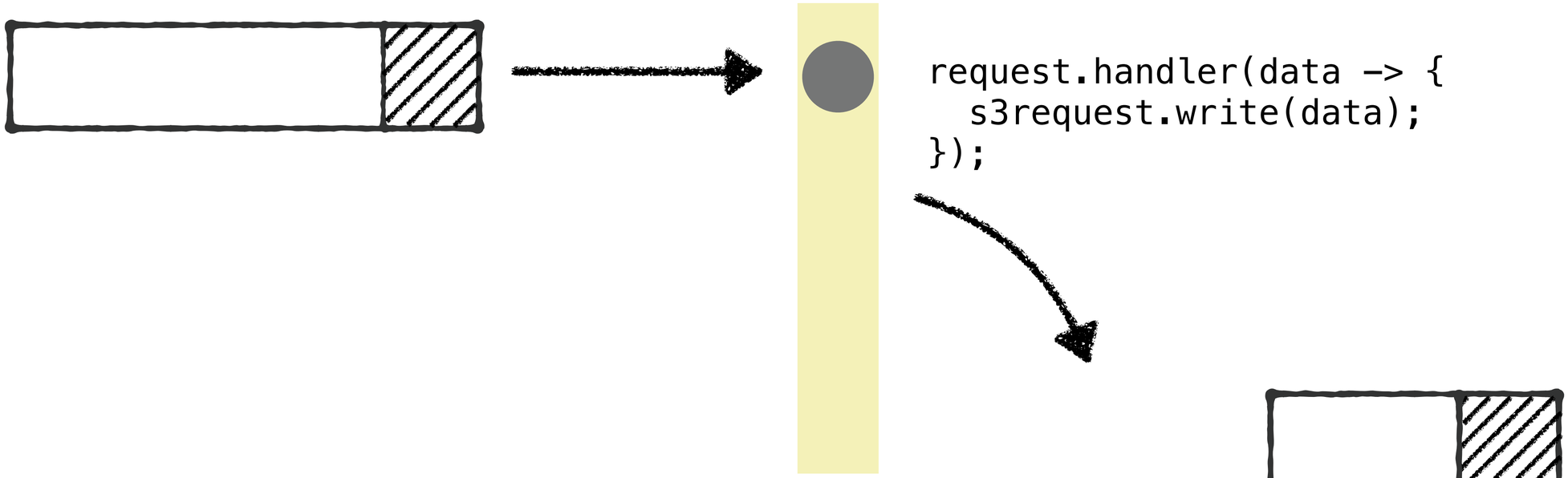


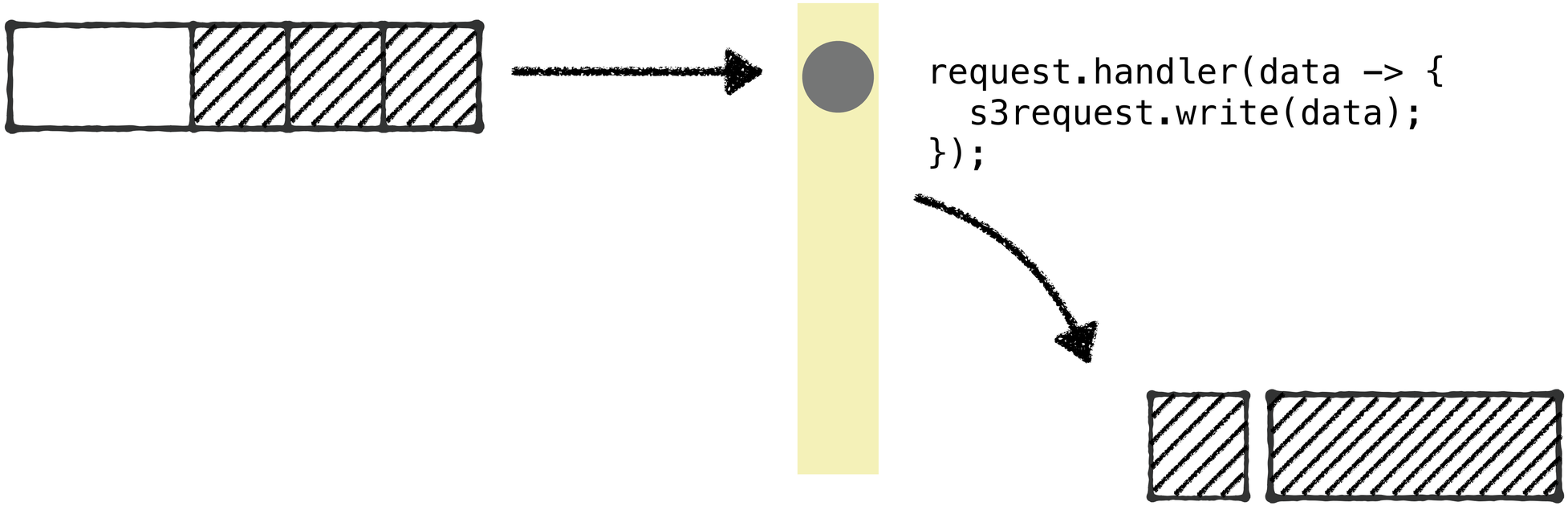
HTTP service with an S3 backend

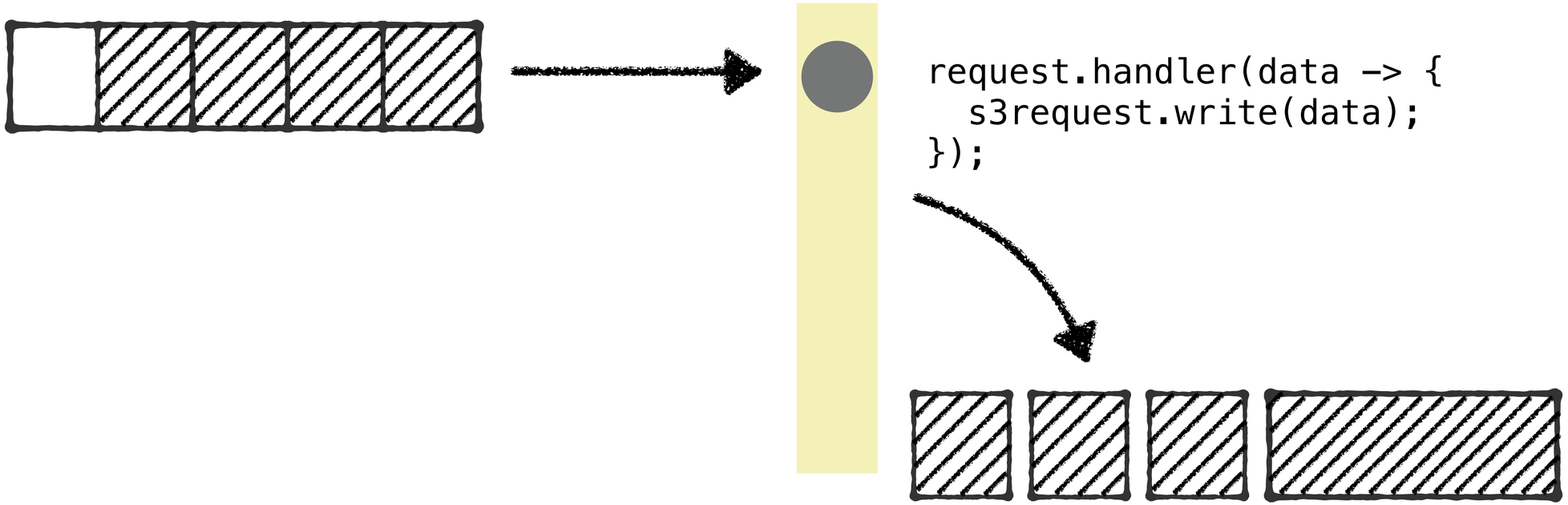
```
HttpServer server = vertx.createHttpServer();
S3Client s3Client = new S3Client();

server.requestHandler(request -> {

  if (request.method() == PUT && request.path().equals("/upload")) {
    request.bodyHandler(data -> {
      s3Client.put("the-bucket", "the-key", data, s3Response -> {
        HttpServerResponse response = request.response();
        response
           .setStatusCode(s3Response.statusCode())
           .end();
      });
    });
  }
});

server.listen(8080);
```

Streaming to S3

```
server.requestHandler(request -> {
  if (request.method() == PUT && request.path().equals("/upload")) {

    S3ClientRequest s3Request = s3Client.createPutRequest(
        "the-bucket", "the-key", s3Response -> {
          HttpServerResponse response = request.response();
          response
            .setStatusCode(s3Response.statusCode())
            .end();
    });

    request.handler(chunk -> s3Request.write(chunk));

    request.endHandler(v -> s3Request.end());
  }
});
```


request.handler(data -> {
s3request.write(data);
});


request.handler(data -> {
s3request.write(data);
});


request.handler(data -> {
s3request.write(data);
});

Transfering data

Back-pressure signal

back-pressure propagates as a signal between systems in protocols

- network
- Inter Process communication pipes
- threads
- etc…

Back-pressure

mechanism in protocols to slow down the data flow between a producer and a consumer

when demand> capacity : queue or drop packets

Back-pressure in protocols

TCP

HTTP/2

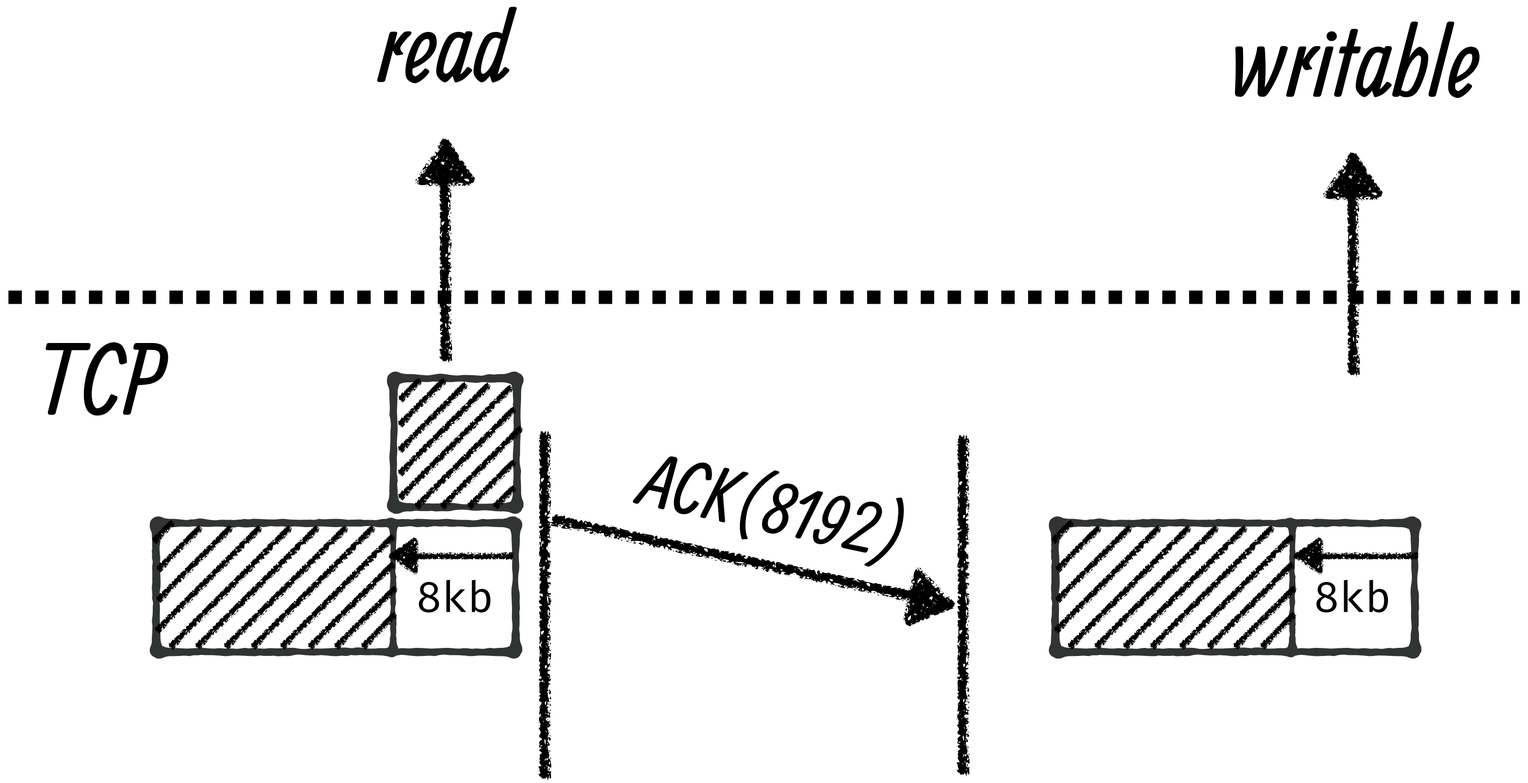
Websockets / SockJS

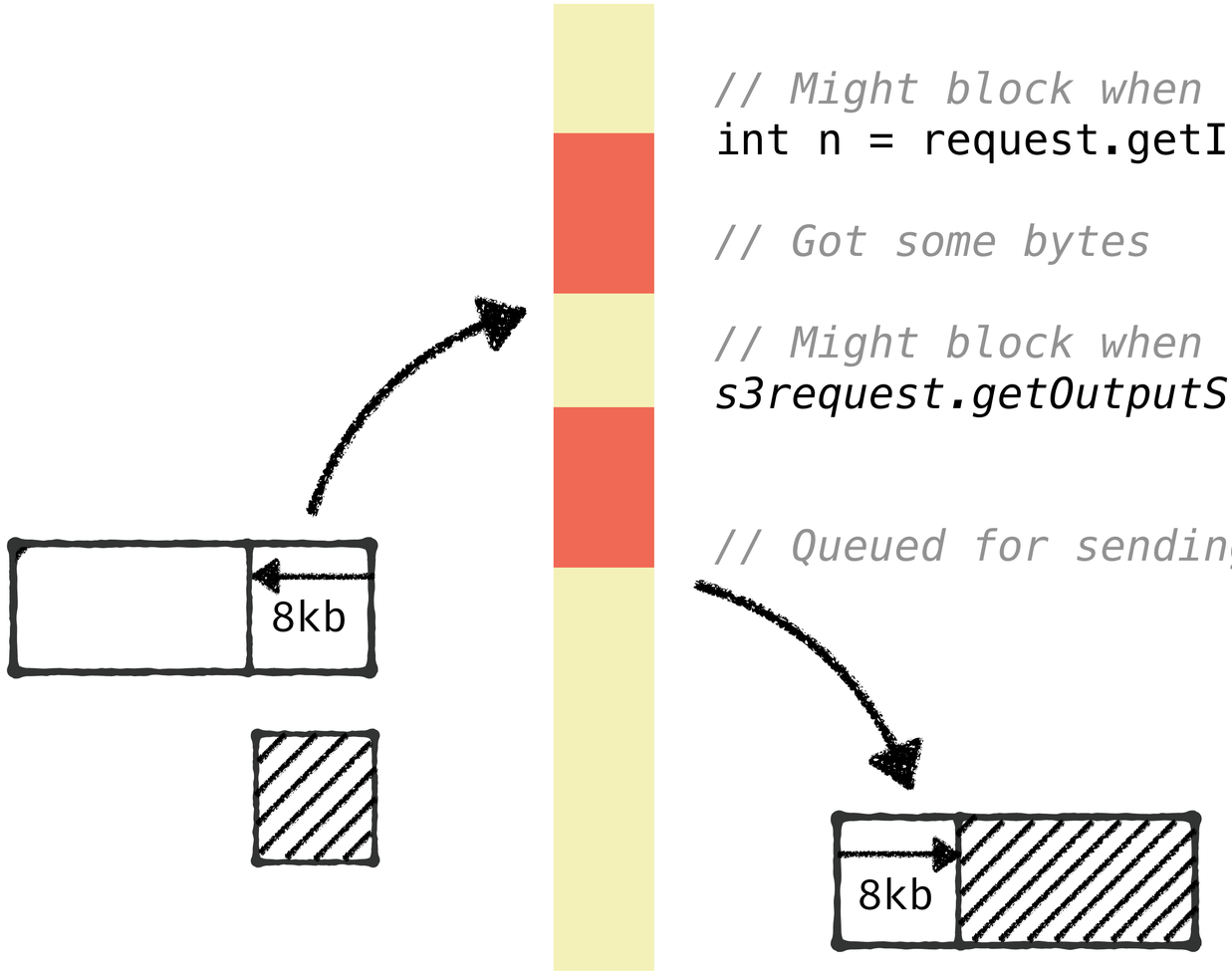
AMQP

OS pipes

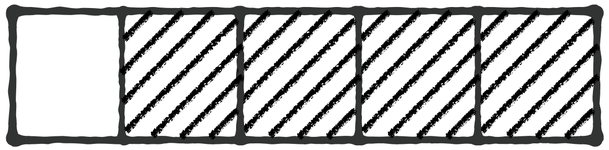
etc…

Back-pressure in TCP

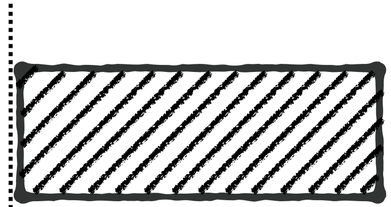


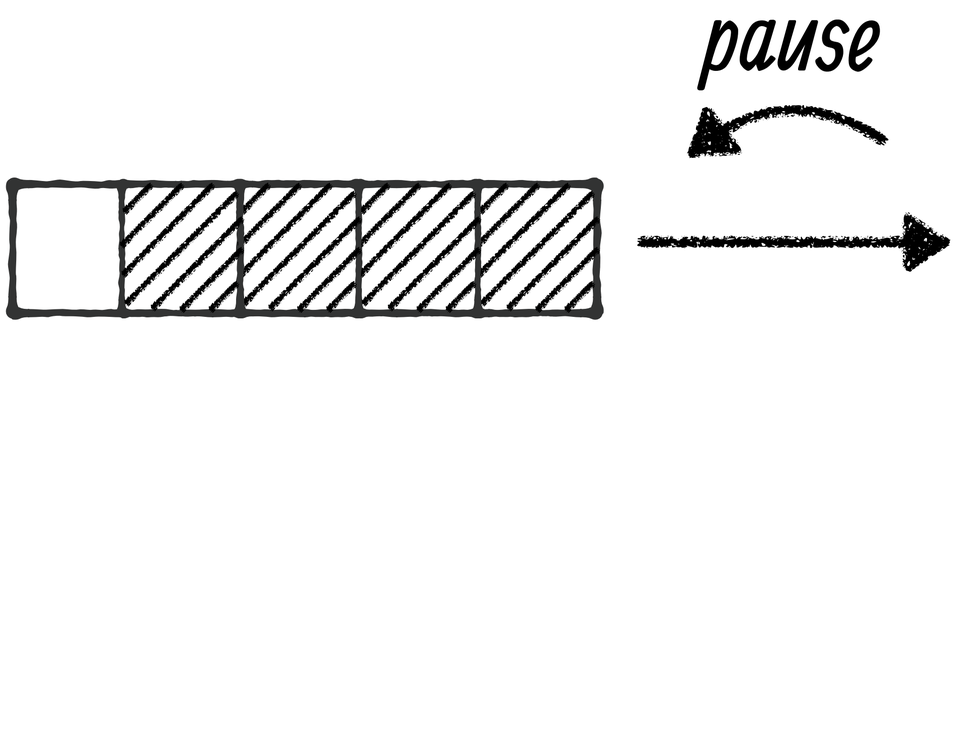



```
// Might block when the buffer is empty
int n = request.getInputStream().read(data);

// Got some bytes

// Might block when the buffer is full
s3request.getOutputStream().write(data, 0, n);


// Queued for sending
```

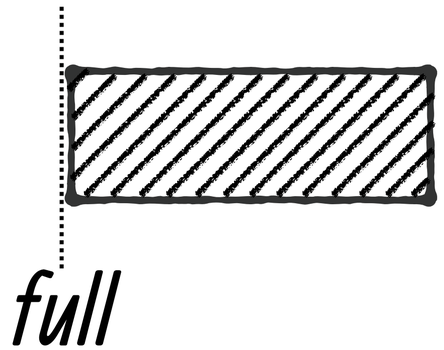


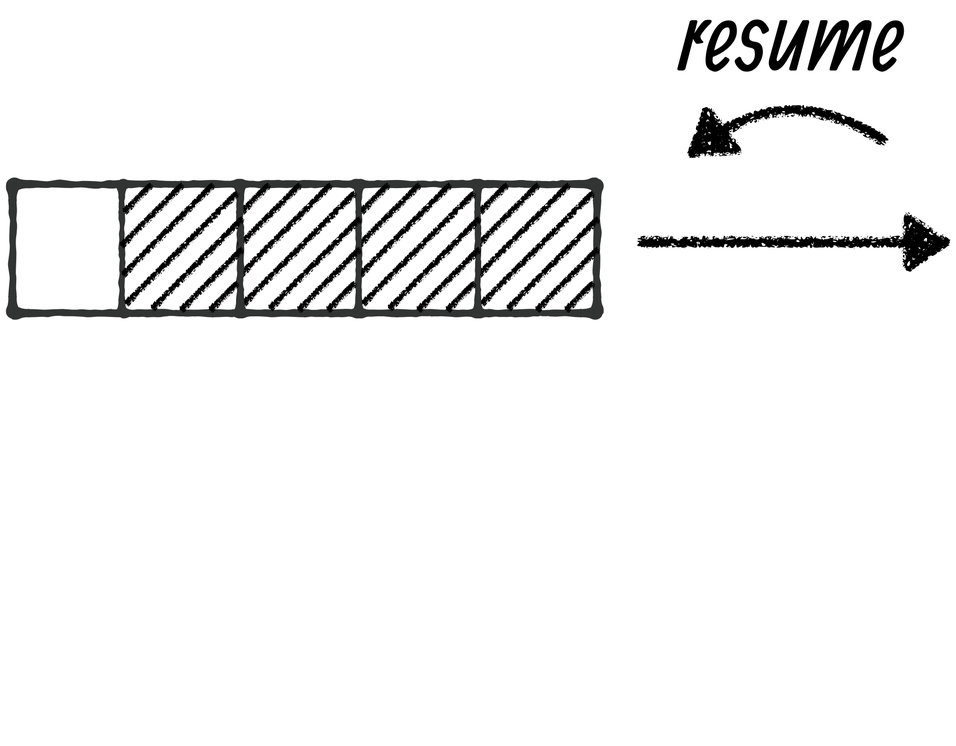
```
request.handler(data -> {
  if (!s3request.writeQueueFull()) {
     s3request.write(data);
  }
});
```



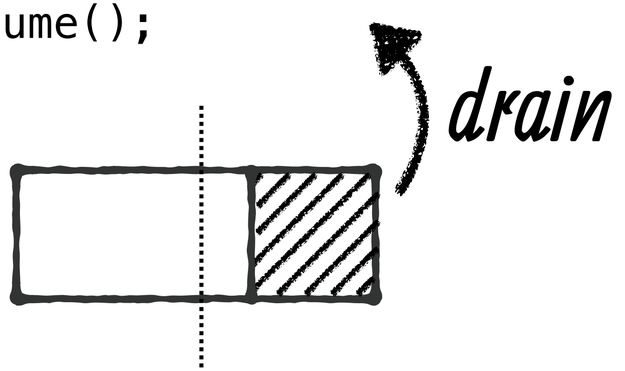



```
request.handler(data -> {
  s3request.write(data);
  if (s3request.writeQueueFull()) {
    // stop reading from the request
     request.pause();
  }
});
```






```
request.handler(data -> {
  s3request.write(data);
  if (s3request.writeQueueFull()) {
    request.pause();
    s3request.drainHandler(v -> {
      socket.resume();
    });
  }
});
```



Read stream

```
public interface ReadStream<T> {

  void pause();

  void resume();

  void handler(Handler<T> handler);

  void endHandler(Handler<Void> handler);

}
```

Write stream

```
public interface WriteStream<T> {

  void write(T data);

  boolean writeQueueFull();

  void drainHandler(Handler<Void> handler);

  void end();

}
```

Pumping

```
Pump pump = Pump.pump(request, s3request);
pump.start();

request.endHandler(v -> {
  pump.stop();
  s3Request.end();
});
```

Unified back-pressure

TCP streams

HTTP client/server request/responses

WebSocket

Async file

Kafka client consumer/producer

SockJS

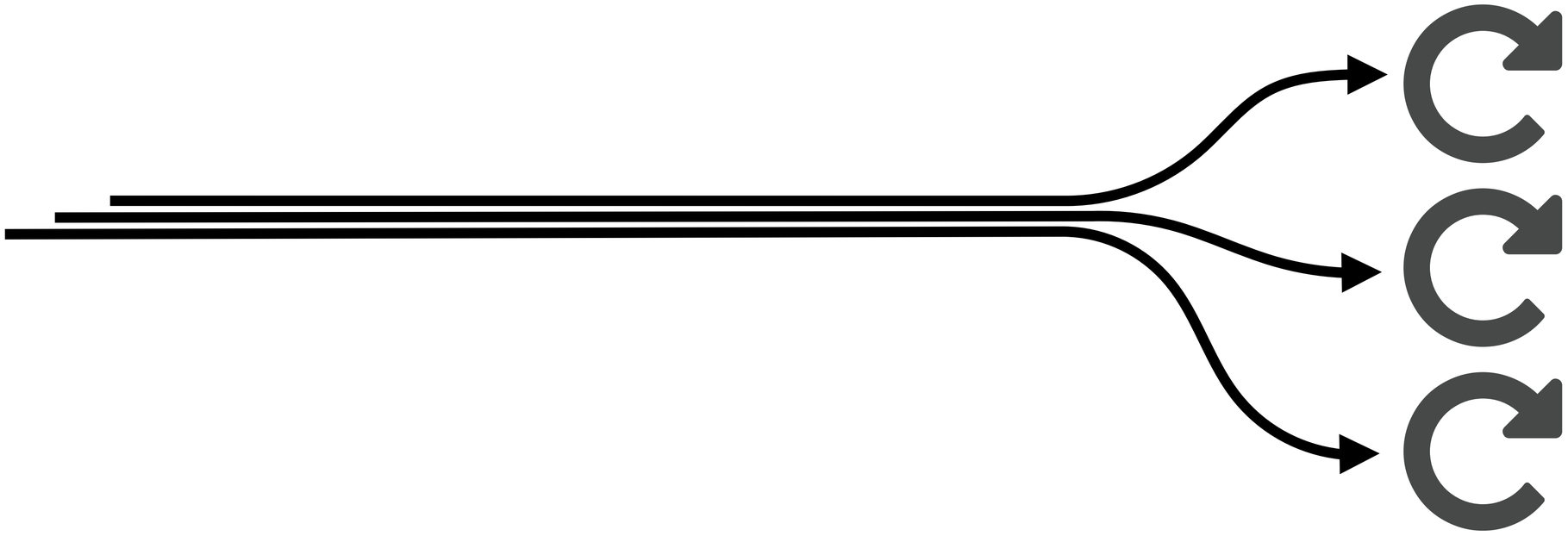
OS pipes

Vert.x provides an unified model for dealing with back-pressured streams

Reactive-stream for interoperability in Java ecosystem

Scaling servers

```
class Server extends AbstractVerticle {
  @Override
  public void start() throws Exception {
    HttpServer server = vertx.createHttpServer();
    server.requestHandler(req -> {
      req.response().end("Hello from");
    });
    server.listen(8080);
  }
}

// Will it bind ?
vertx.deployVerticle(
    Server.class.getName(),
    new DeploymentOptions().setInstances(3)
);
```



Boiler Vroom
(wifi “Boiler Vroom”)

Boiler Vroom, unplugged

HTTP Client

MP3

VLC

VERT.X

Event bus → Chunked HTTP

MP3

Clients

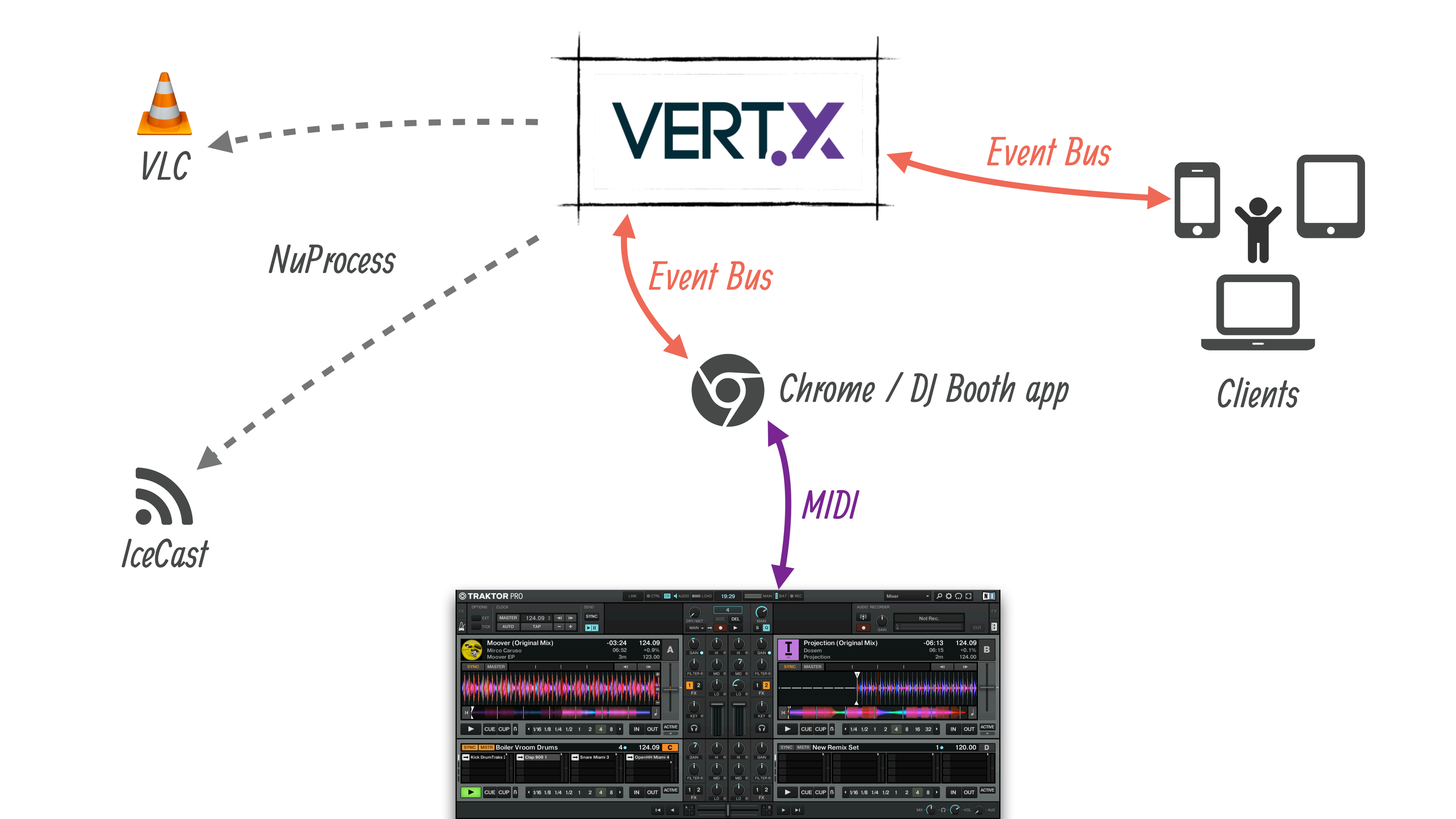
OGG/Vorbis

HTTP 1.1

IceCast

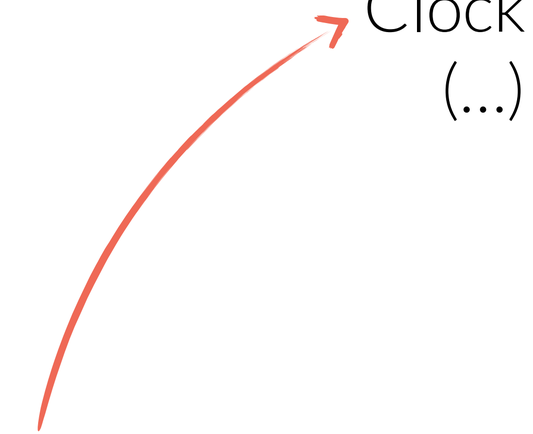
OGG/Vorbis

HTTP 1.0


VERT.X
VLC
Event Bus
NuProcess
Event Bus
Chrome / DJ Booth app
Clients
IceCast
MIDI
TRAKTOR PRO

MIDI Signals

Channel control
Note on / off
Start / Stop
Pitchbend
Clock
(…)



https://webaudio.github.io/web-midi-api/

https://github.com/cotejp/webmidi

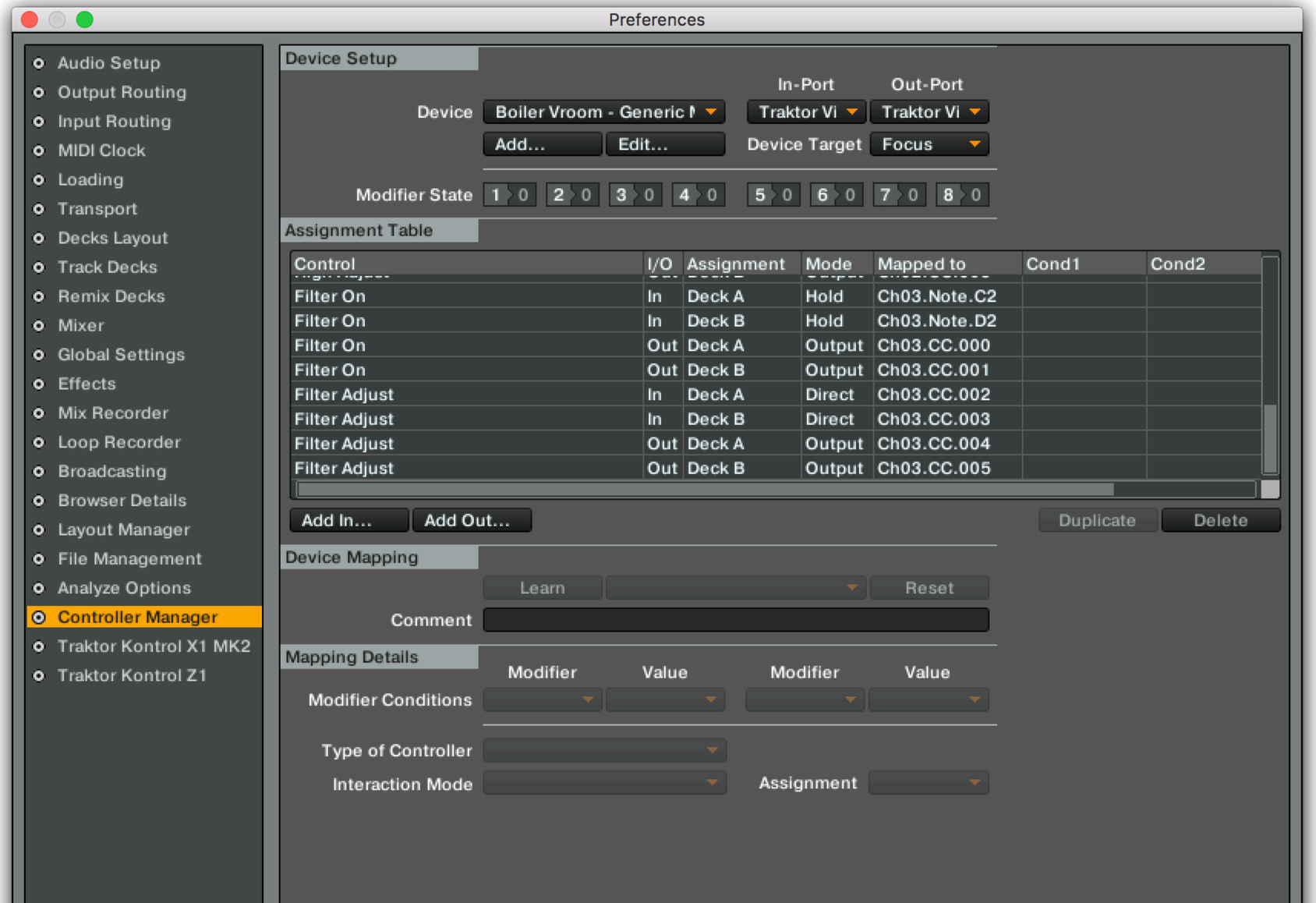
https://github.com/jponge/webmidi-sequencer

https://github.com/jponge/boiler-vroom

WebMidi

JSON / Event Bus

MIDI In / Out mappings




Gradle


yarn


webpack


VERT.X


WebMidi.js


TRAKTOR PRO

(Code walkthrough)

Guest 1

About me

- Sébastien Blanc
- Red Hat, Keycloak team
- @sebi2706

The vertx TCP eventbus bridge

- Clients don’t need to be part of the cluster.
- Several clients available : Go, C++, Python, JS, JS (Browser and NodeJS) , Java ...
- Strict protocol but easy to implement :
  - 4bytes int32 message length (big endian encoding)
  - json string (encoded in UTF-8)

The vertx java TCP eventbus bridge Client

- Small library
- 0 dependency
- Language level >= 7
- Simple API :

```
EventBus eventBus = new EventBus("localhost",7000, new MessageHandler() {
  public void handle(Message message) {
     //Handle error
  }
});
```

Register to an address

```
eventBus.register("yourAddress", headers, new MessageHandler() {
  public void handle(Message responseMessage) {
     //handle response message;
  }
});
```

Create a message

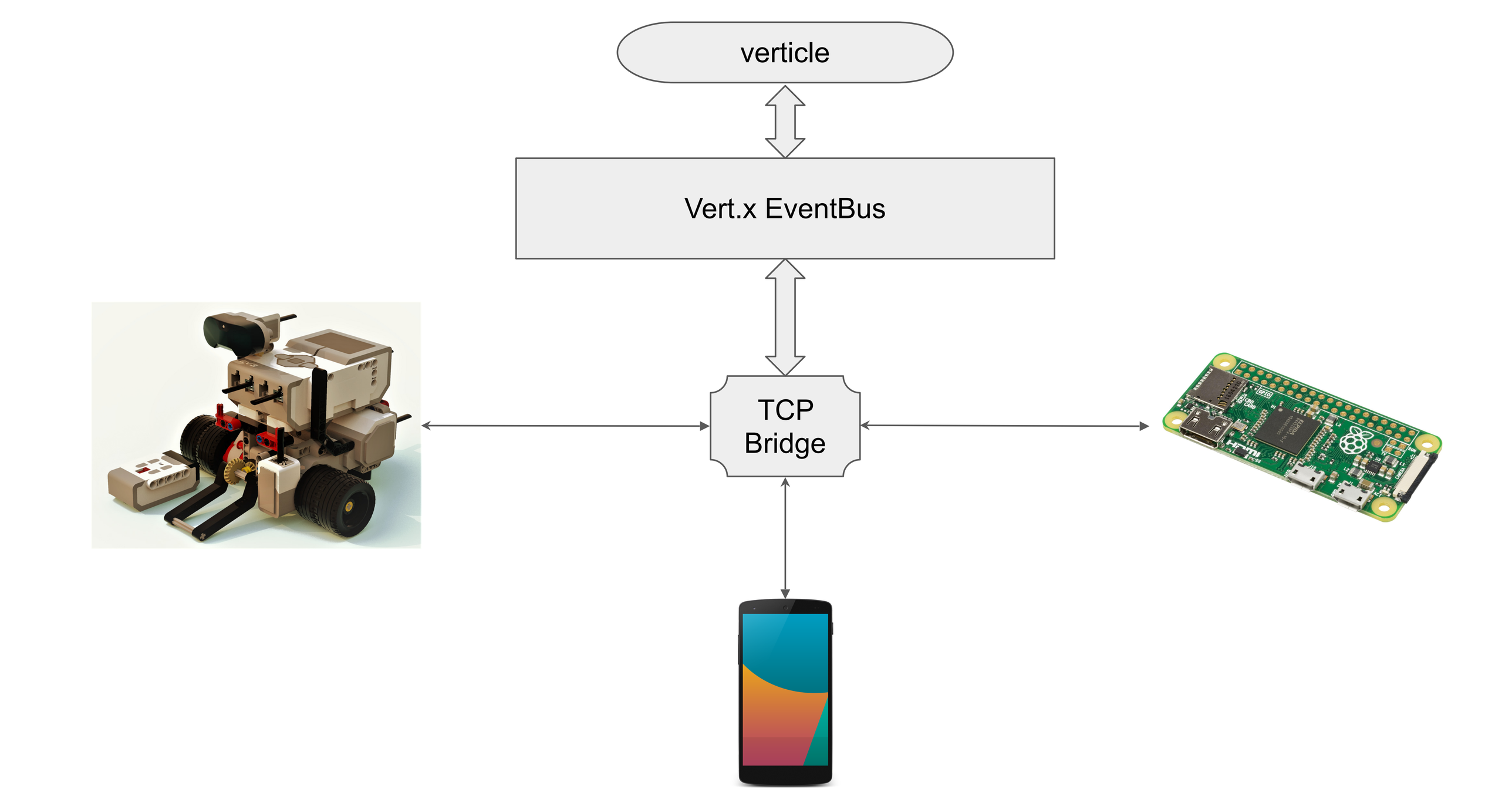
```
Message message = new Message();
message.setAddress("yourAddress");
Map bodyMap = new LinkedHashMap();
bodyMap.put("action","forward");
message.setBody(bodyMap);
```

Publish to an Address

```
eventBus.publishMessage(message);
```

Send to an Address

```
eventBus.sendMessage(message, new MessageHandler() {
    public void handle(Message responseMessage) {
        //handle response message;
    }
});
```


verticle
Vert.x EventBus
TCP
Bridge

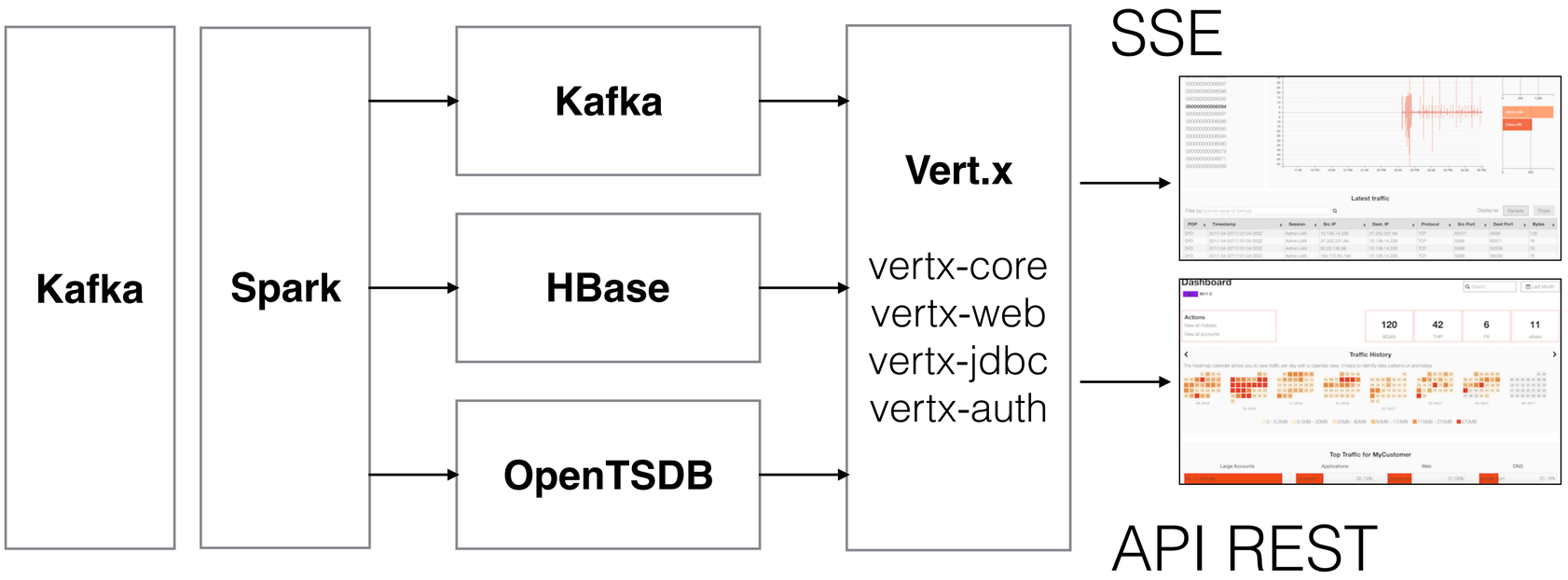
Guest 2

@raphaelluta

- Consultant Technique Indépendant
  Web & Data
- Master 2 Commerce Electronique
  Université Paris Est Créteil
- Membre
  Apache Software Foundation
- Adepte de Pareto
- Disciple de Little et Gunther
- Speaker occasionnel


ESTELLE MAERSK
MAERSK LINE

Architecture



Pourquoi Vert.x ?

- Réactif bien adapté pour le streaming
- Développement de services indépendants, communiquant via l'event bus
- …mais déploiement dans la même JVM possible

Guest 3

Vert-x, Polyglot + IOT? 🤣

Philippe Charrière aka @k33g_org

- Previously: solution engineer @GitHub
  - *raising source code* 🐑
- **Currently: CSO & tech evangelist @Clever_Cloud** 😄
  - *deploying apps like a God* ⚡
- And **especially**, core committer on the well known **Eclipse Golo** project 😛
- 2 passions:
  - *blinking leds (I ❤️ IOT) & REST applications (with any languages)*
  - *I create a web framework every day (it's a legend)*

3 use cases

- Blinking LEDs
- Quick “microservices”
- Playing with MQTT

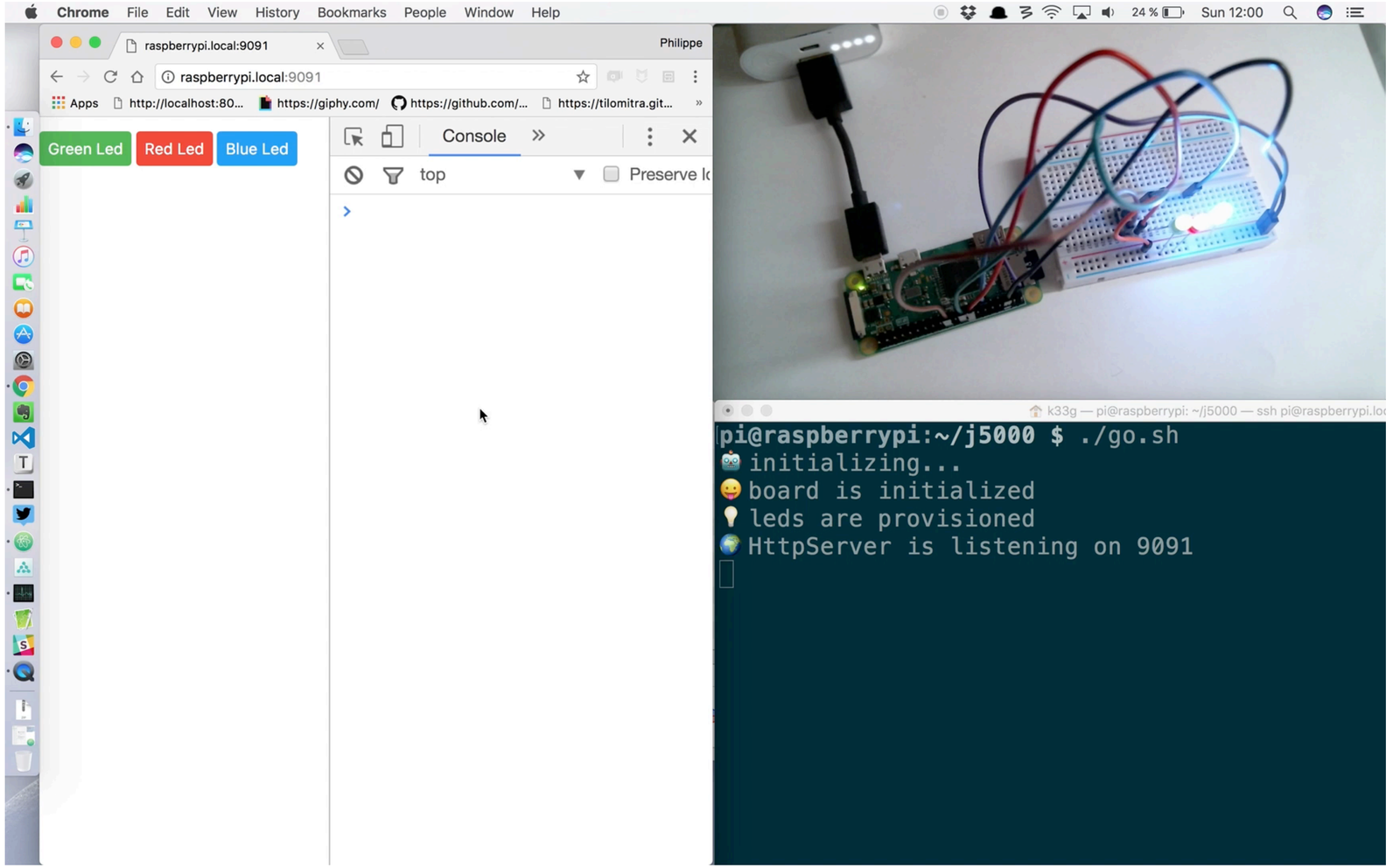
1st use case

- Blinking LEDs with RPI 0
- Using a dynamic language (for the JVM)
  - *Editing code and compiling in ssh it's boring*
- “Running blink” from my browser

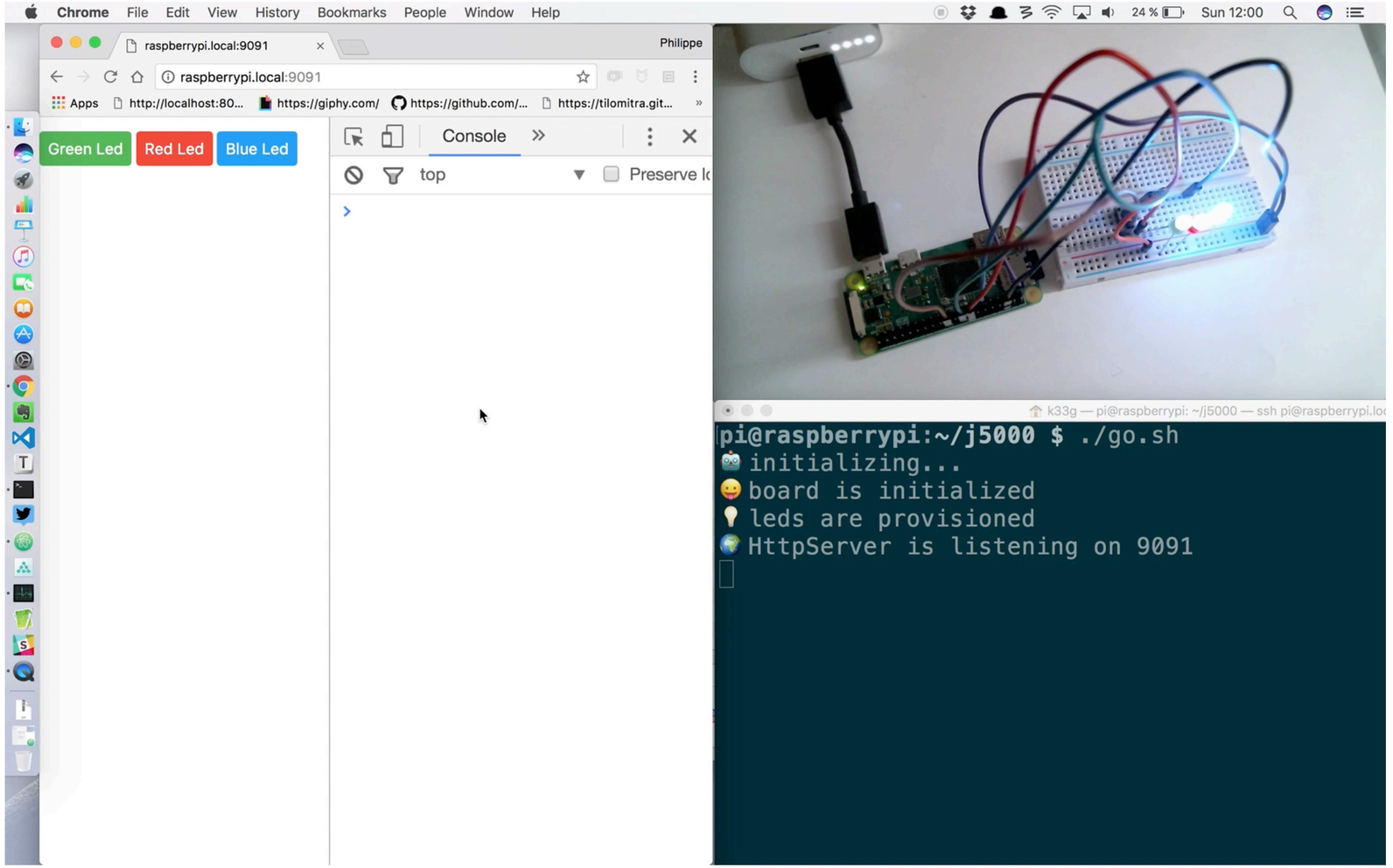
My weapons

- **Golo** - Eclipse Foundation project
  - *A dynamic language for the JVM build with invoke dynamic*
  - *Light (~700 kb) and fast (in a dynamic context)*
- **PI4J** - Java I/O Library for RPI
- **Vert-x**
  - *io.vertx.ext.web + io.vertx.core.http*

The expected result



The expected result



Golo augment Vert-x

```
augment io.vertx.ext.web.Router {
  function get = |this, uri, handler| {
    return this: get(uri): handler(handler)
  }
  function post = |this, uri, handler| {
    return this: post(uri): handler(handler)
  }
}

augment io.vertx.core.http.HttpServerRequest {
  function param = |this, paramName| -> this: getParam(paramName)
}

augment io.vertx.core.http.HttpServerResponse {
  function djson = |this| {
    this: putHeader("content-type", "application/json;charset=UTF-8")
    return DynamicObject(): define("send", |self| {
      this: end(JSON.stringify(self), "UTF-8")
    })
  }
}
```

Golo augment Vert-x

```
let server = createHttpServer()
let router = getRouter()

router: get("/blink/:led", |context| {

  trying({
    let selectedLed = context: request(): param("led")

    match {
      when selectedLed: equals(“red") then redWorker: send(5)
      when selectedLed: equals(“green") then greenWorker: send(5)
      when selectedLed: equals(“blue") then blueWorker: send(5)
      otherwise raise("😡 Huston!")
    }
    return selectedLed
  })
  : either(
    recover = |error| →
      context: response(): djson(): error(error: message()): send(),
    mapping = |selectedLed| →
      context: response(): djson(): led(selectedLed): send()
  )
})

startHttpServer(server=server, router=router, port=9091, staticPath="/*")
```

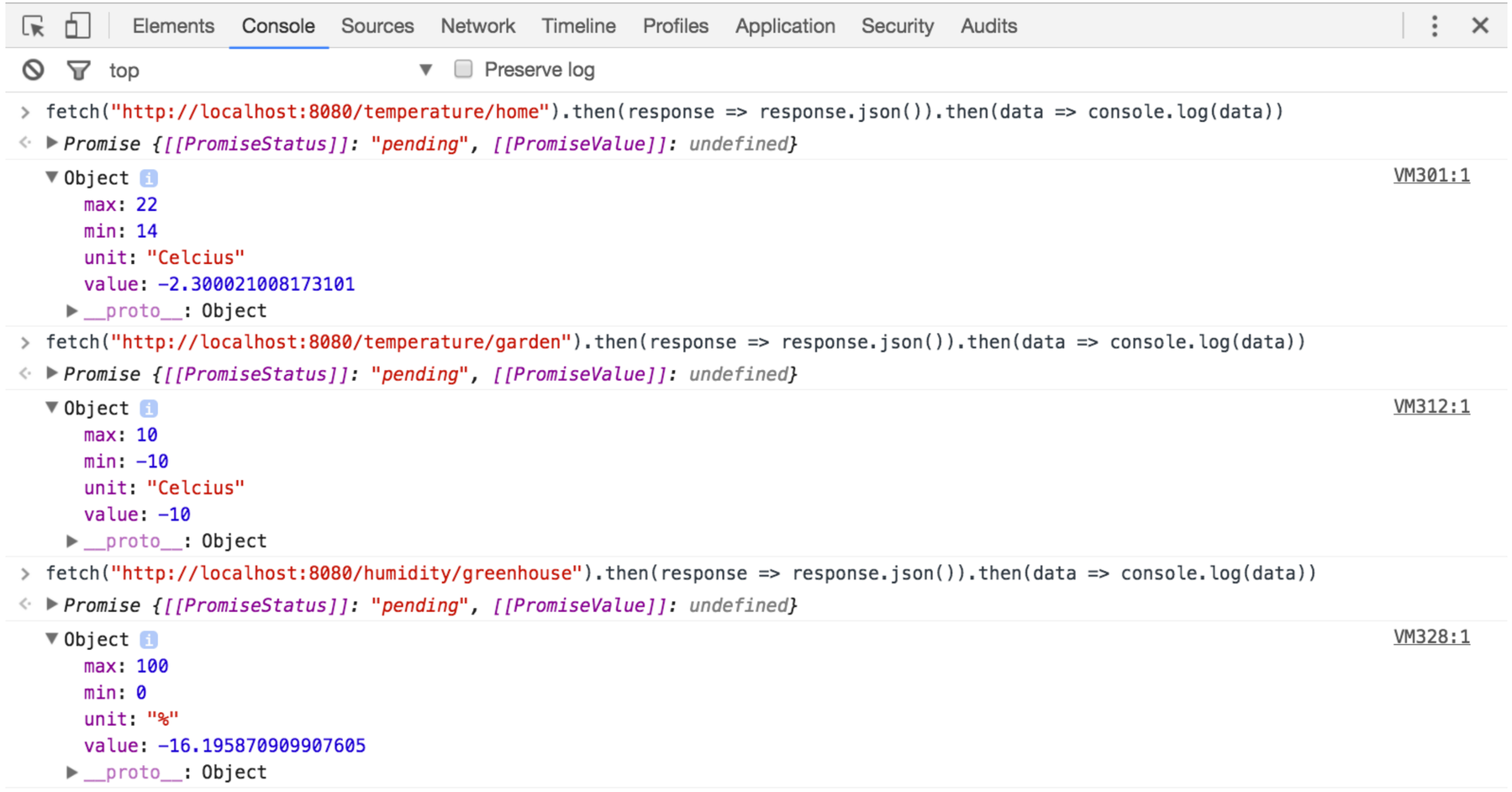
2nd use case

- Simulate sensors gateways
- Using a less dynamic and more typed language (for the JVM) to write some kind of “microservice”
  - *POC for my current company*

My weapons

- **Scala - 🙀⚠️👶 steps**
  - *In fact, Scala is simple as JavaScript, with a little effort 🤔*
- **Vert.x**
  - *io.vertx.scala.ext.web + io.vertx.scala.core*
  - ❤️ *vertx.setPeriodic*

The expected result



vertx.setPeriodic is 💖

```
class Component(val min:Double, val max:Double) {
  var value:Double = 0.0

  private val vertx = Vertx.vertx()
  private val getRandomInt = (min:Int, max:Int) ⇒
     Math.floor(Math.random()*(max - min +1)) + min
  private val B = Math.PI / 2.0
  private val unitsTranslatedToTheRight = getRandomInt(0, 5)
  private val amplitude = () ⇒ (this.max - this.min)/2.0
  private val unitsTranslatedUp = () ⇒ this.min + amplitude()

  private val getLevel = (t: Double) ⇒
    amplitude() * Math.cos(B*(t-unitsTranslatedToTheRight))
      + unitsTranslatedUp()

  private val timer = vertx.setPeriodic(1000, (v) ⇒ {
    val now = LocalDateTime.now()
    val t:Double = now.getMinute + now.getSecond / 100.0
    value = getLevel(t)
  })

  def cancel() = {
    vertx.cancelTimer(timer)
  }
}
```

io.vertx.scala.ext.web, it's like Express.js 😉

```
val server = vertx.createHttpServer()
val router = Router.router(vertx)
val jsonStr = (obj: Any) ⇒ jsonMapper.writeValueAsString(obj)

val temperatureSensorHome = new Temperature(14.0, 22.0)
val temperatureSensorGarden = new Temperature(-10.0, 10.0)
val humidityGreenHouse = new Humidity(0.0, 100.0)

router.route("/temperature/home").handler(context ⇒ {
  context.response().end(jsonStr(temperatureSensorHome))
})

router.route("/temperature/garden").handler(context ⇒ {
  context.response().end(jsonStr(temperatureSensorGarden))
})

router.route("/humidity/greenhouse").handler(context ⇒ {
  context.response().end(jsonStr(humidityGreenHouse))
})

server.requestHandler(router.accept _).listen(8080)
```

3rd use case

- Simulate sensors gateways
- So, using Scala
- and **MQTT**
  - *POC for my current company*

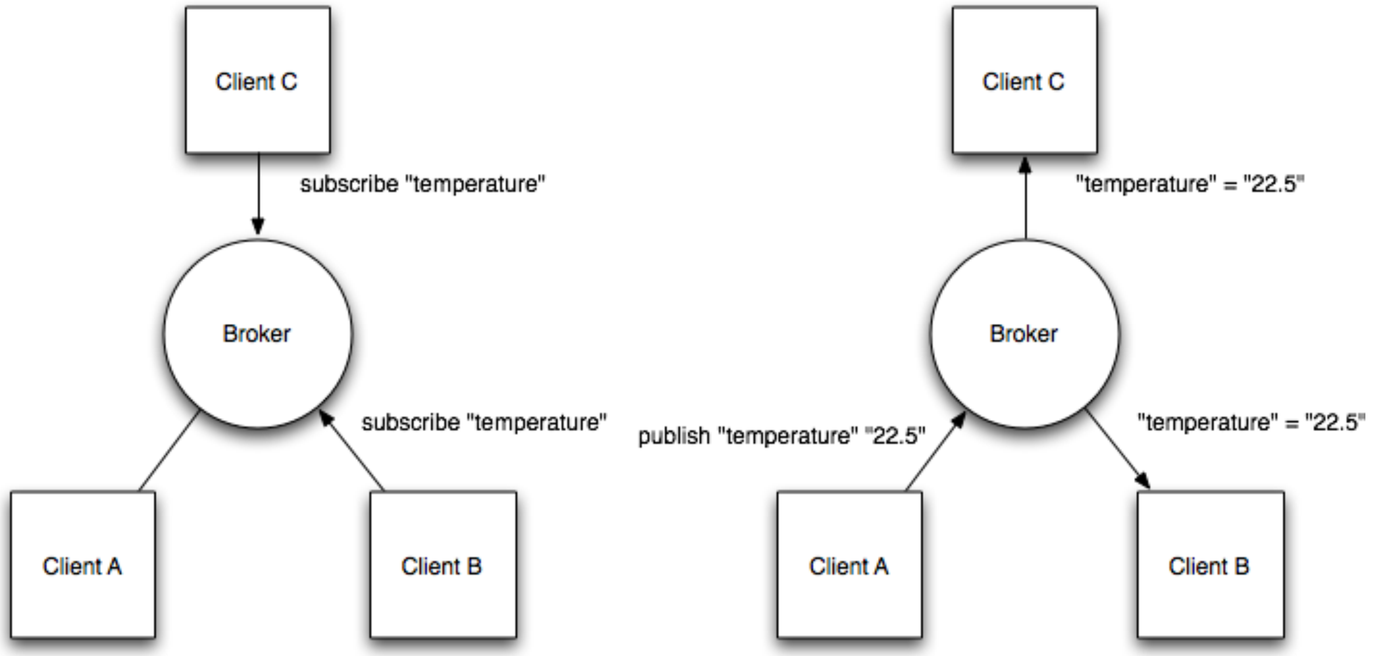
My weapons

- **Scala**
- **Vert/x**
  - *io.vertx.scala.**mqtt** + io.vertx.scala.core + vertx.setPeriodic*
- **Paho - Java version + MQTT.js**
  - *MQTT Clients*

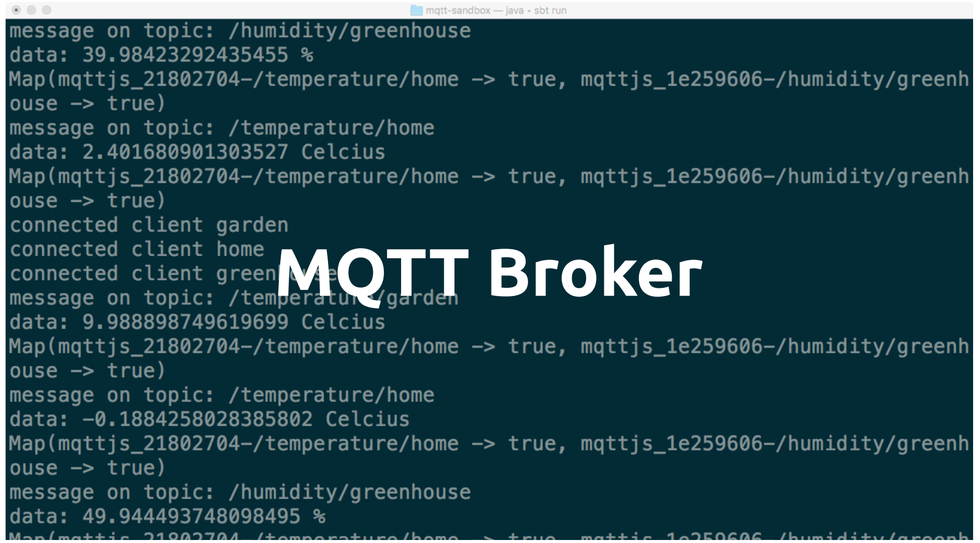
MQTT?

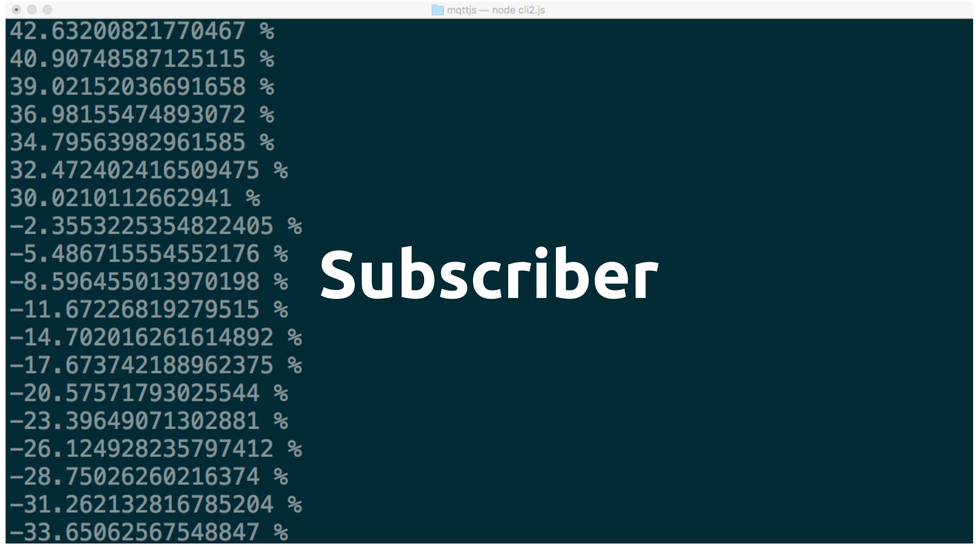
Message Queue Telemetry Transport

https://eclipse.org/community/eclipse_newsletter/2014/february/article2.php

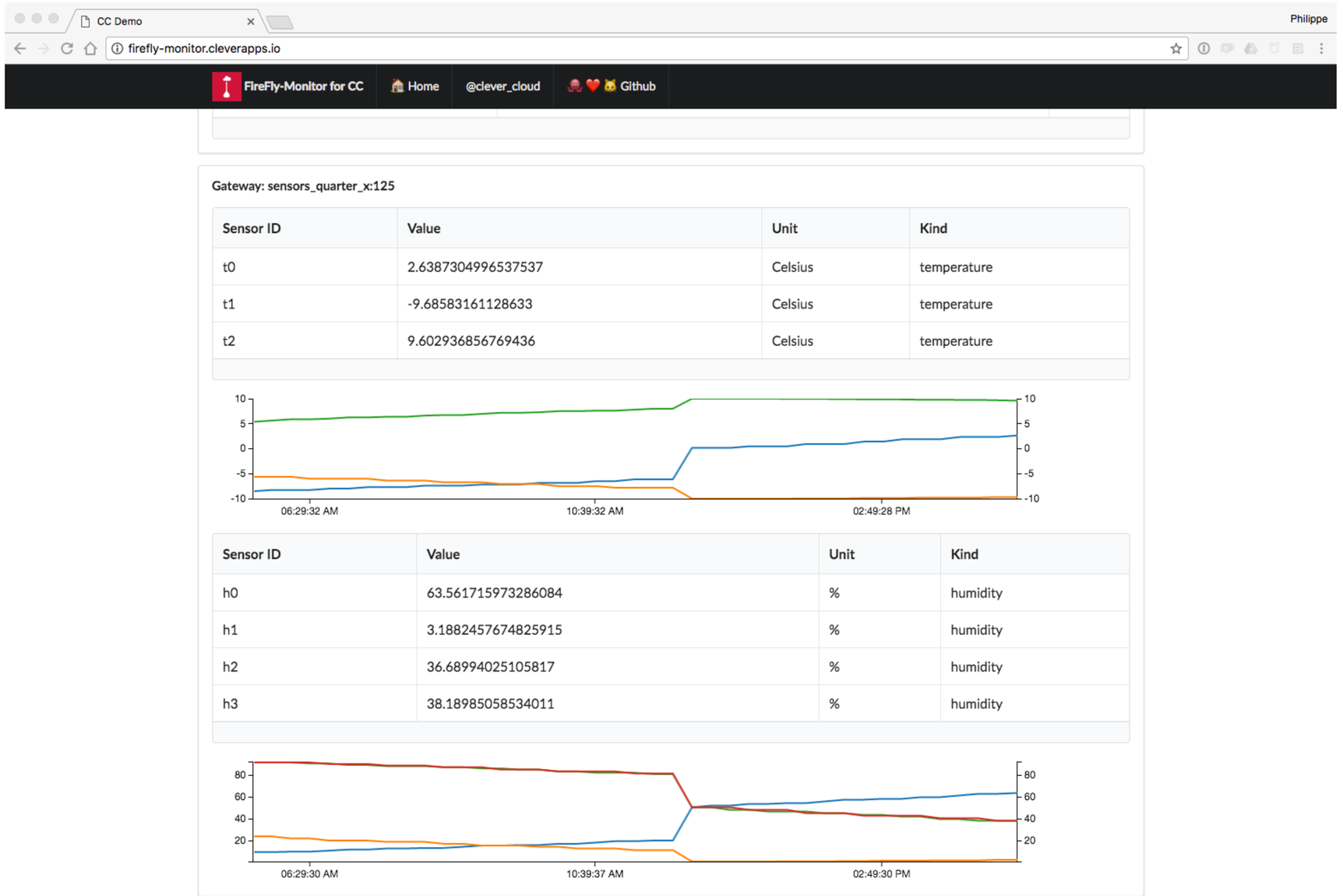


The expected result





The expected result



Some MQTT publisher(s) with Paho

```
val sensorGreenHouse = new Humidity(0.0, 100.0)

def getClient(id: String):MqttClient = {
  val brokerUrl = "tcp://localhost:1883"
  val persistence = new MemoryPersistence
  val client = new MqttClient(brokerUrl, id, persistence)
  client.connect()
  client
}

vertx.setPeriodic(4000, (v) ⇒ {
  Try({
    getClient("greenhouse")
      .getTopic("/humidity/greenhouse")
      .publish(
         new MqttMessage(
           s"${sensorGreenHouse.value} ${sensorGreenHouse.unit}”.getBytes(“utf-8")
         ))
  })
  match {
    case Success(value) ⇒ println(s"😄 Data published")
    case Failure(cause) ⇒ println(s"😡 Huston? ${cause.getMessage}")
  }
})
```

We need a MQTT broker! ⚠️ Don’t use it in production

```
mqttServer.endpointHandler(endpoint ⇒ {
  endpoint.accept(false)

  clientsMap(endpoint.clientIdentifier()) = endpoint // Add the endpoint to the clients map

  endpoint.subscribeHandler(subscribe ⇒ { // update subscriptions
    subscribe.topicSubscriptions.foreach(subscription ⇒ {
      subscriptionsMap(endpoint.clientIdentifier()+"-"+subscription.topicName()) = true
    })
  })

  endpoint.publishHandler(message ⇒ { // You've got a 📫
    clientsMap.values.foreach((client) ⇒ { // if 👨 has subscribed to the current topic, then send 💌
      subscriptionsMap.get(client.clientIdentifier()+"-"+message.topicName()) match {
        case Some(b) ⇒
         client.publish(message.topicName(), Buffer.buffer(message.payload().toString()),
         MqttQoS.AT_LEAST_ONCE, false, false)
        case None ⇒ { /* 😝 */ }
      }
    })
  })
})

mqttServer.listen()
```

Some MQTT subscribers(s) with MQTT.js

```
var mqtt = require('mqtt')
var client  = mqtt.connect('mqtt://localhost:1883')

client.on('connect', () ⇒ {
  client.subscribe('/temperature/home')
  client.publish('/presence', 'yo mqtt server')
})

client.on('message', (topic, message) ⇒ {
  console.log(message.toString())
})
```

(Reactive) Ecosystem

Databases

- MySQL, PostgreSQL
- JDBC
- MongoDB
- Redis

Authentication / Security

- Auth:
  - Apache Shiro (LDAP, properties, …)
  - JDBC
  - MongoDB
- OAuth (+ providers)
- Json Web Tokens (JWT)

Metrics

- DropWizard
- Hawkular
- Health Checks

Messaging / Integration

- AMQP 1.0
- STOMP
- SMTP
- Kafka
- RabbitMQ
- Camel
- JCA

Microservices

- Discovery
- Circuit breaker
- Config
- Consul
- Kubernetes
- GRPC

Clustering

- Hazelcast
- Apache Ignite
- Infinispan
- Apache Zookeeper

Last but not least…

- vertx-unit
- Because testing async code is a bit different…
- Integrates with JUnit

Reactive Programming with Vert.x and RxJava

RxJava

Data and events flows

Great at organizing transformation of data and coordination of events

It makes most sense when many sources of events are involved

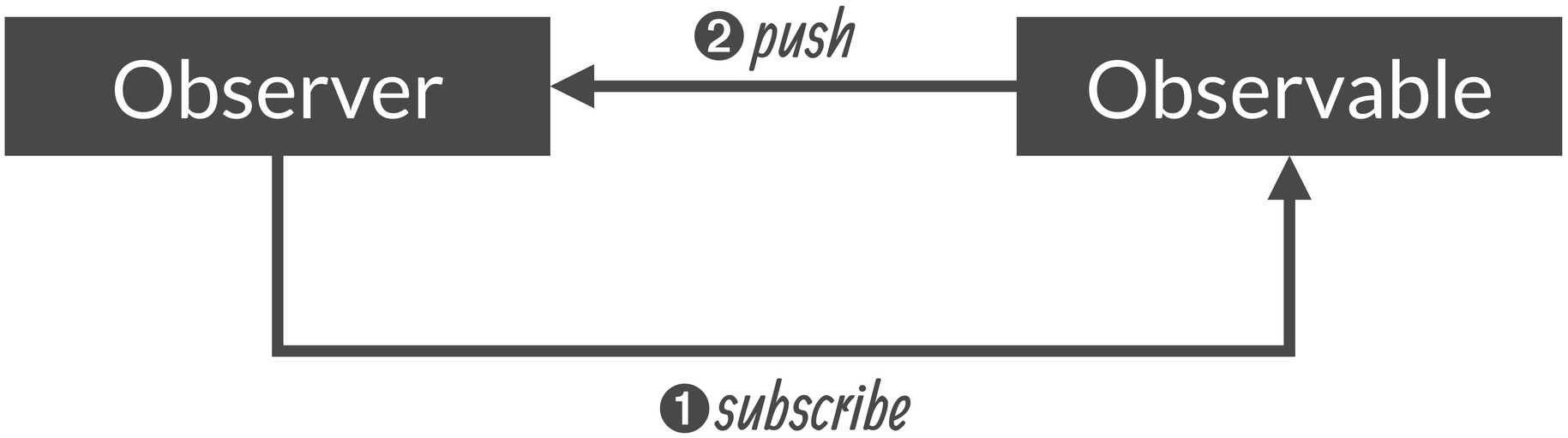
Motivation

Future<List<T>> is not appropriate

Dealing with latencies

Functional programming influence

Push based



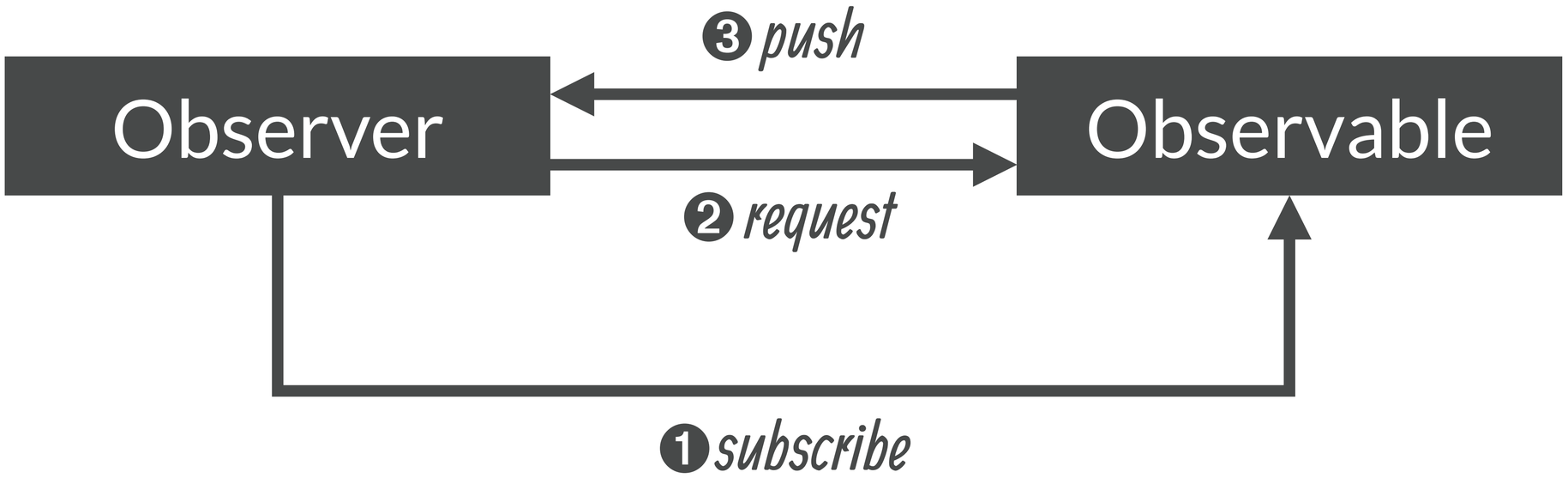
Iterable / Observable

```
try {
  for (String item : it) {
    ❶
  }
  ❸
} catch (Throwable e) {
  ❷
}
```

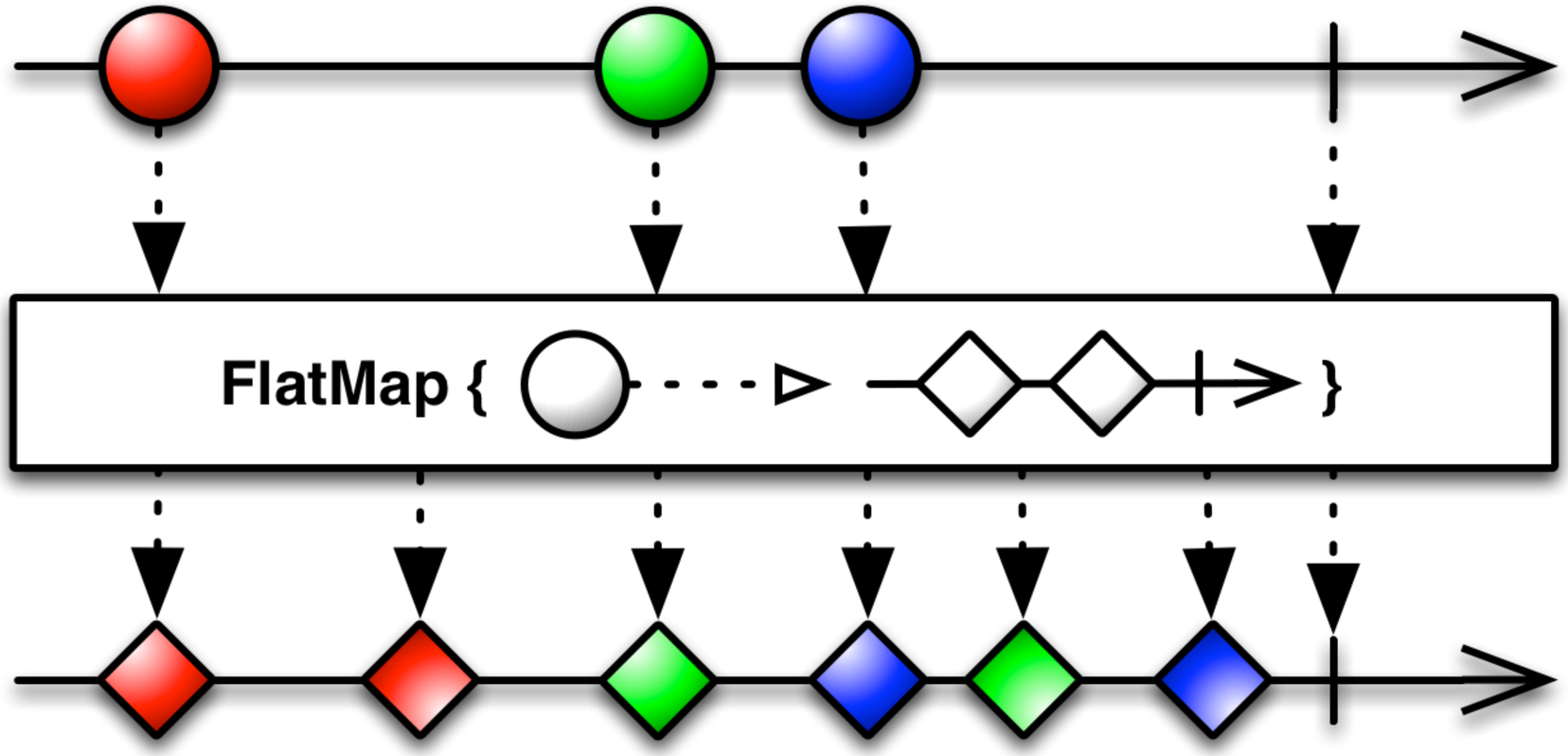
```
observable.subscribe(item ->
{
  ❶ // onNext
}, error -> {
  ❷ // onError
}, () -> {
  ❸ // onCompleted
});
```

	0	1	0..n
Reactive	`Completable`	`Single<T>`	`Observable<T>`
Interactive	`void`	`T`	`Iterable<T>`

Reactive pull back pressure



Operators



Rxified APIs

Each API type (annotated with @VertxGen) has its prefix `io.vertx` replaced by `io.vertx.rxjava`

`io.vertx.core.Vertx` → `io.vertx.rxjava.Vertx`

etc…

Rxified ReadStream

```
package io.vertx.rxjava.core.streams;

interface ReadStream<T> {

  /**
   * @return an unicast and back-pressurable Observable,
   *         hot or not depends on the stream
   */
  Observable<T> toObservable();
}
```

Rxified HttpServerRequest

```
server.requestHandler(request -> {

  Observable<Buffer> obs = request.toObservable();

  obs.subscribe(buffer -> {
    // A new buffer
  }, err -> {
    // Something wrong happened
  }, () -> {
    // Done
  });
});
```

Rxified Handler<AsyncResult>

```
void listen(int port, Handler<AsyncResult<HttpServer>> ar)

Single<HttpServer> rxListen(int port);
```

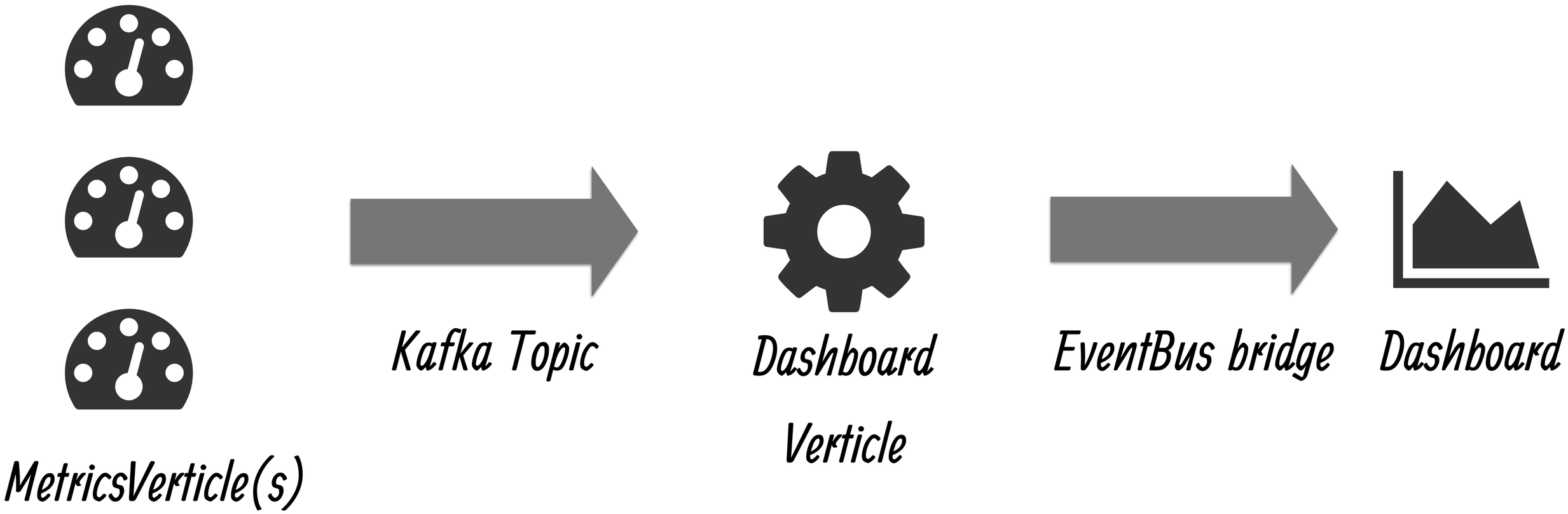
```
server.rxListen(8080).subscribe(
    s -> {
      // Server started
    },
    err -> {
      // Could not start server
    }
);
```

```
HttpRequest<Buffer> request = client.put(8080, "example.com", "/");

Observable<Buffer> stream = getSomeObservable();

// Create a single, the request is not yet sent
Single<HttpResponse<Buffer>> single = request.rxSendStream(stream);

for (int i = 0;i < 5;i++) {
  // Actually send the request and subscribe to the observable
  single.subscribe(response → {
    // Handle the response
  }, error → {
    // Handle the error
  });
}
```

Composing singles

```
Single<HttpResponse<JsonObject>> req1 = createRequest1();
Single<HttpResponse<JsonObject>> req2 = createRequest2();

Single<JsonObject> single = Single.zip(
    req1.retry(3),
    req2,
    (buf1, buf2) ->
        new JsonObject().mergeIn(buf1.body()).mergeIn(buf2.body()));
single.subscribe(json -> {
    // Got merged result
}, err -> {
    // Got error
});
```

Observable demo



RxJava 2

Better performances

Based on reactive-streams

Null values forbidden

More reactive types

- Observable / Flowable
- Single / Maybe / Completable

Prototype at https://github.com/vert-x3/vertx-rx

Outro

Vert.x stack


mongoDB.


Stomp


Infinispan


Apache
Camel
™


RabbitMQ


kafka


apache
Ignite

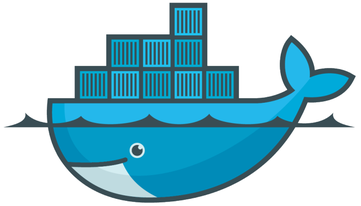

AMQP


MQTT.ORG


hazelcast

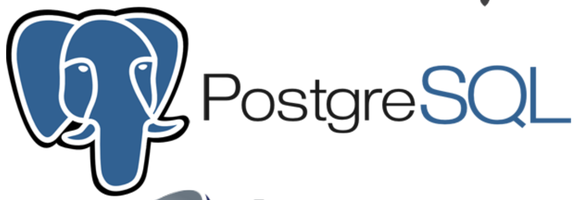
MySQL®


RxJava

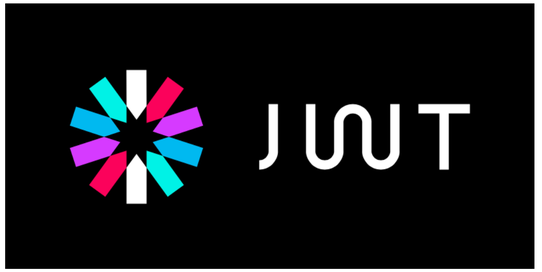

docker


CONSUL


kubernetes

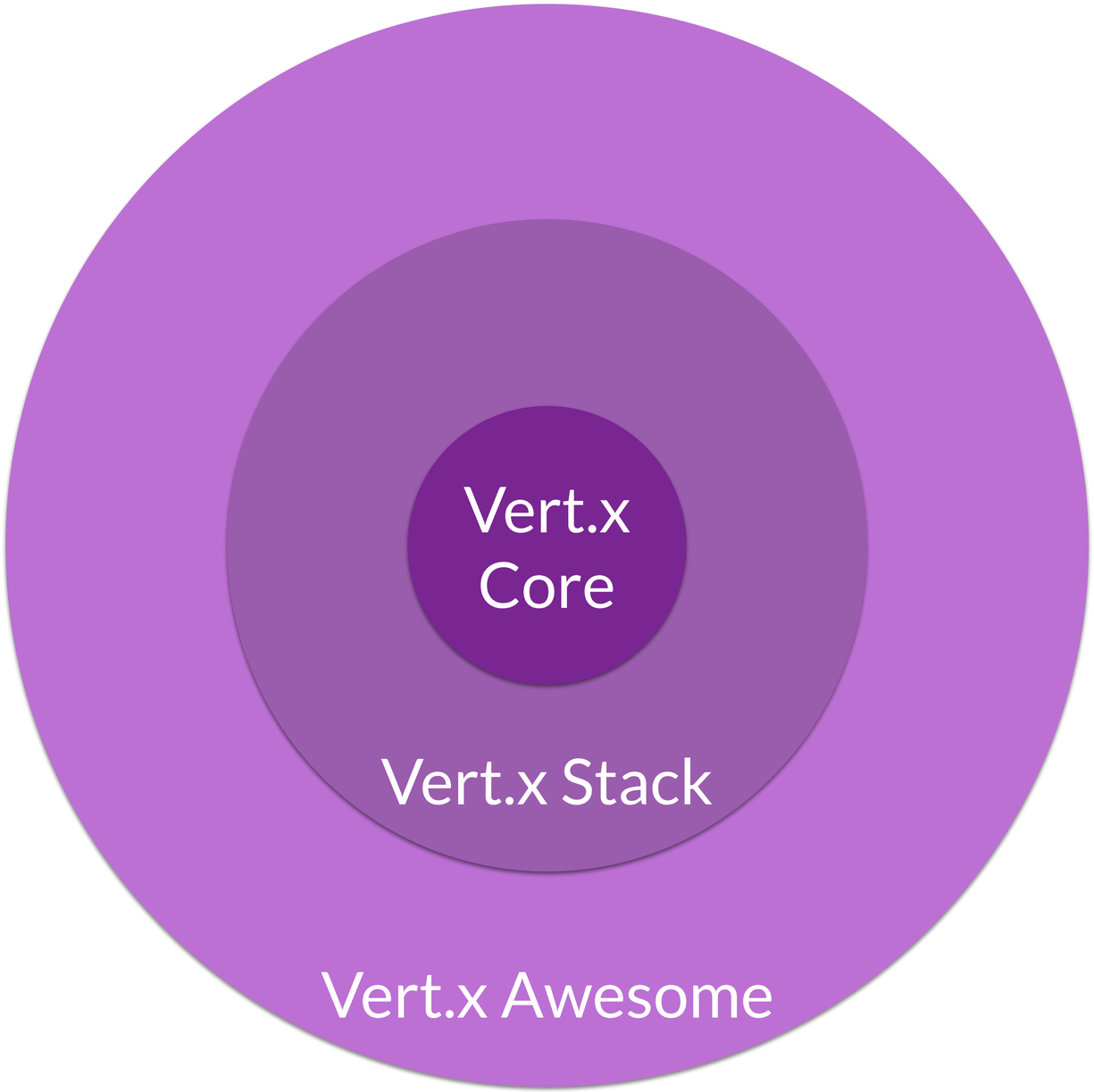

PostgreSQL


redis
GRPC


JWT


OPENSHIFT


APACHE
SHIRO


Vert.x
Core
Vert.x Stack
Vert.x Awesome

That’s all folks

Enjoy the benefits of Reactive with the Vert.x ecosystem

Want more?

- *Microservices réactifs avec Eclipse Vert.x et Kubernetes*
  vendredi 7 avril 14h55