Reactive Applications with Eclipse Vert.x

Julien Viet

QCon Shangai

Julien Viet

Open source developer for 16+ years

@vertx_project lead

Principal software engineer at  redhat.

Marseille JUG Leader

https://www.julienviet.com/

http://github.com/vietj

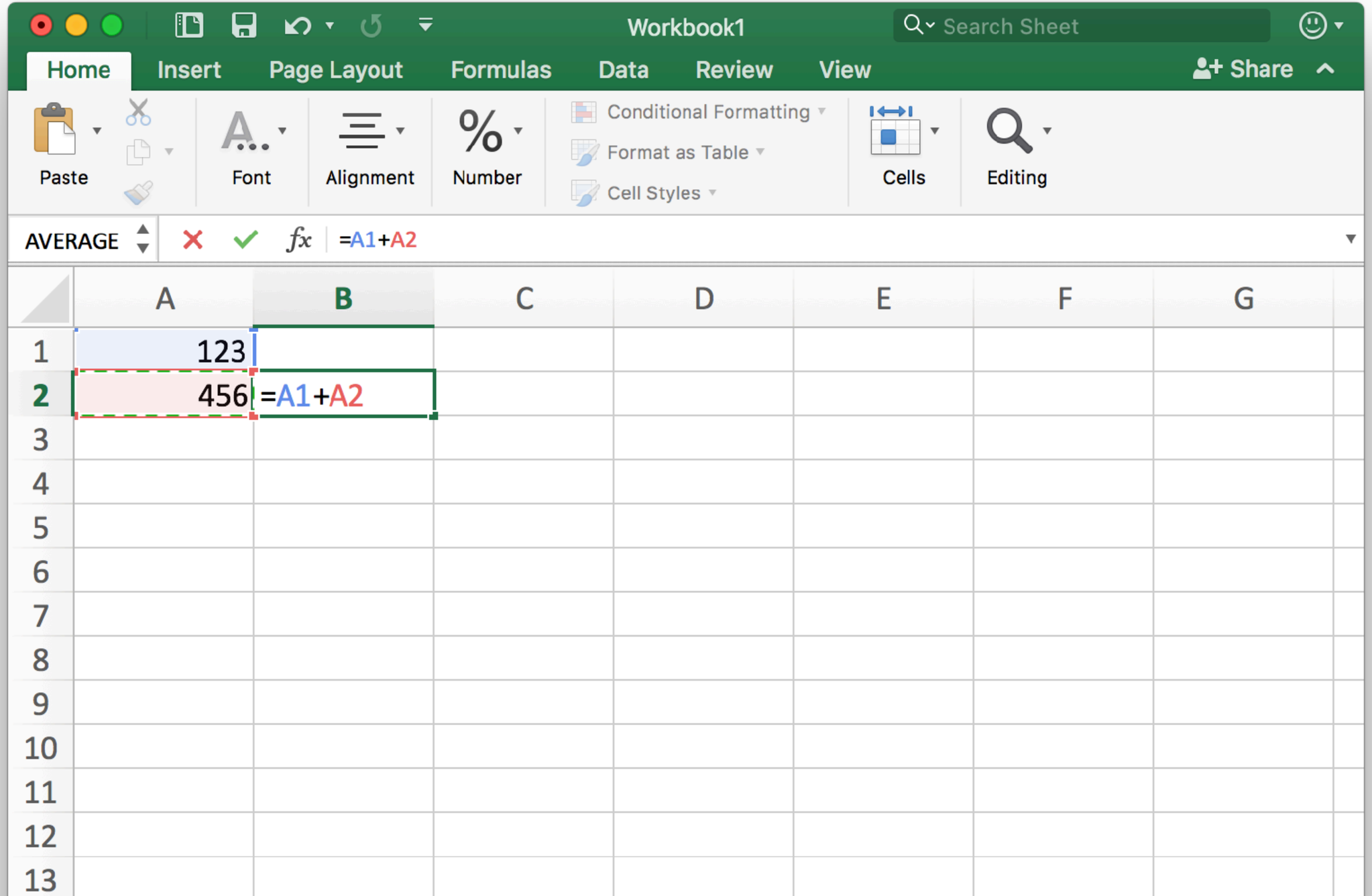
@julienviet

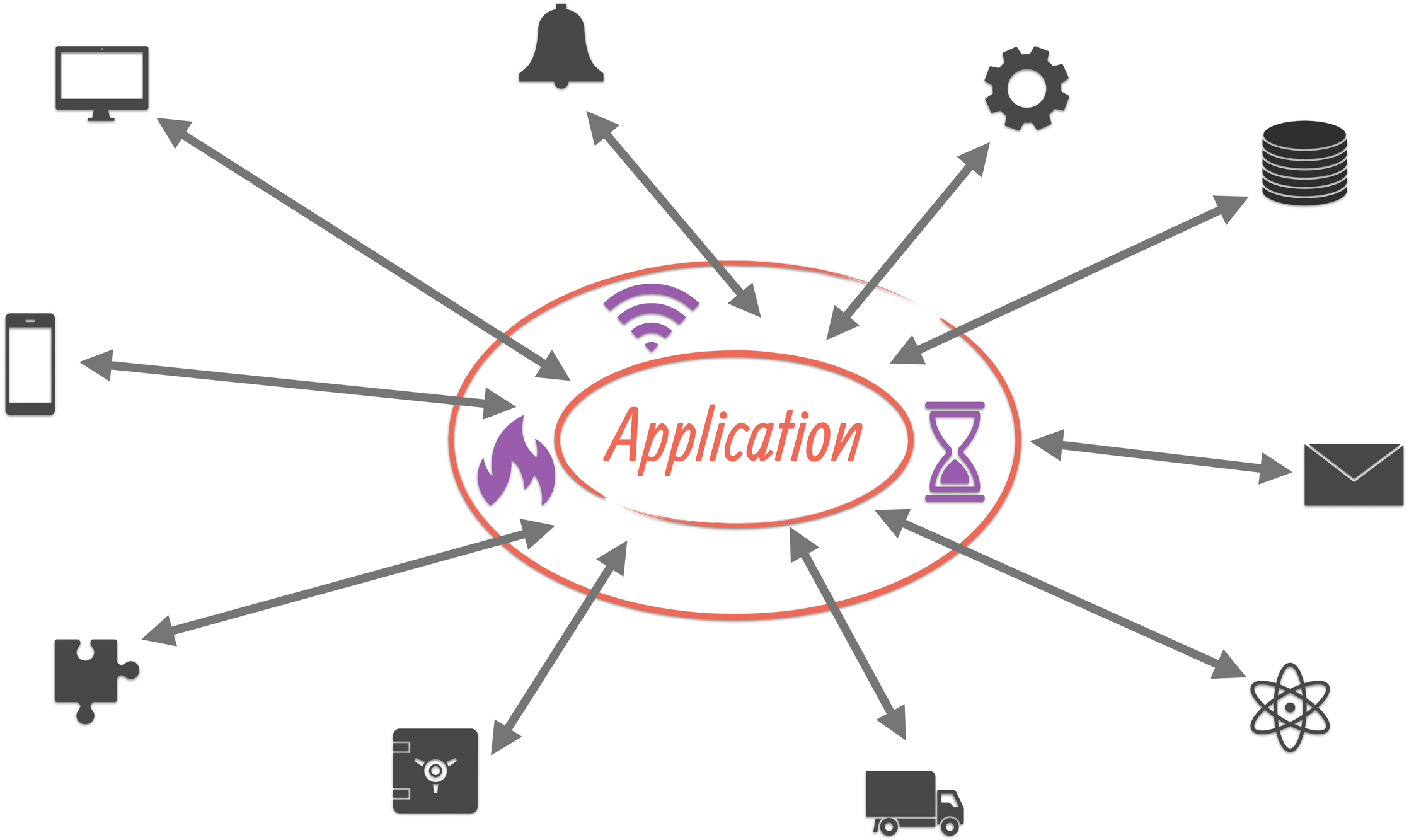
https://www.mixcloud.com/cooperdbi/

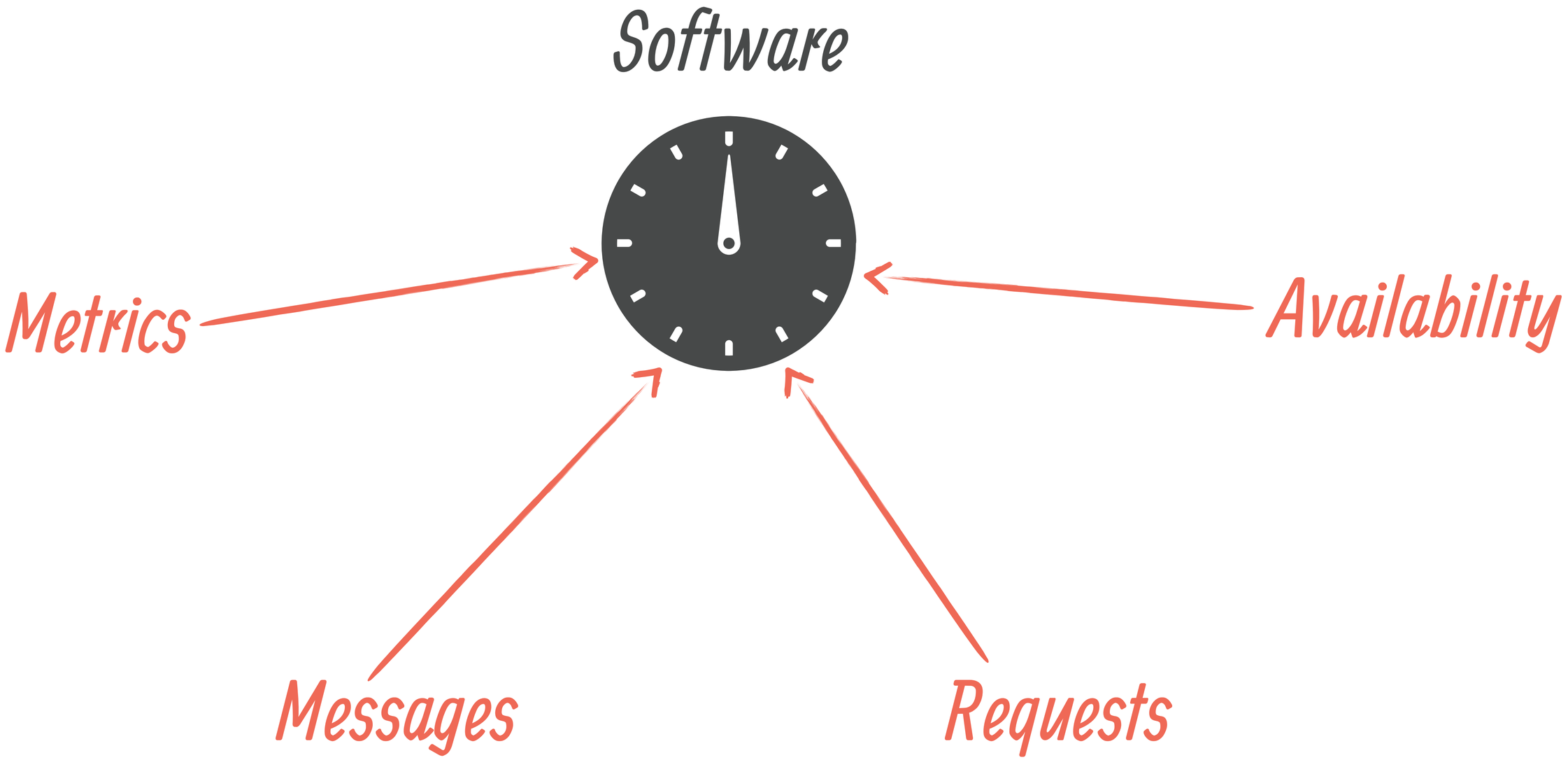
Outline

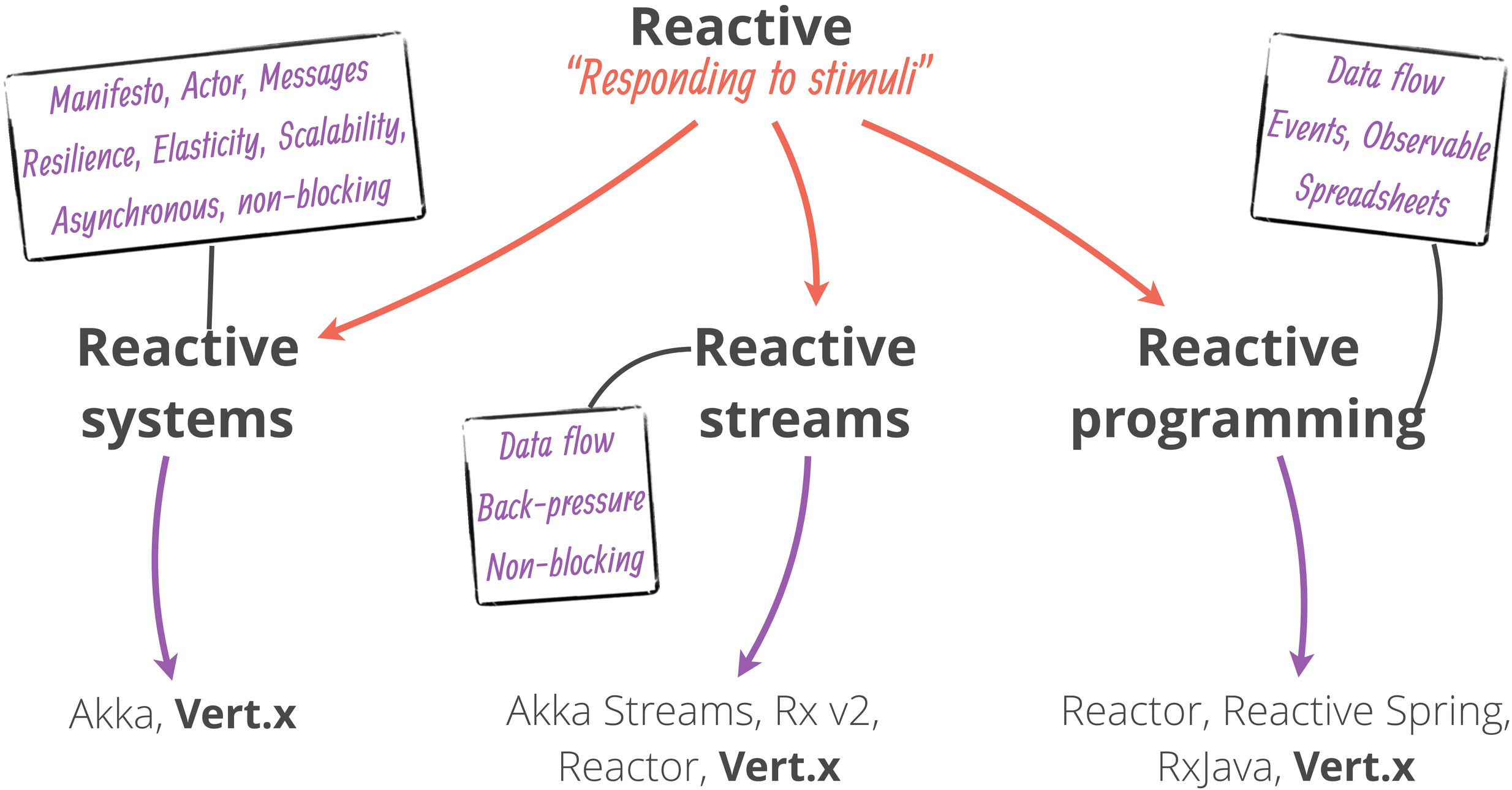
- ✓ Reactive? Vert.x?
- ✓ Foundations
- ✓ Reactive Programming
- ✓ Ecosystem

Reactive?
Vert.x?


Workbook1
Search Sheet
Home
Insert
Page Layout
Formulas
Data
Review
View
Share
Paste
Font
Alignment
Number
Conditional Formatting
Format as Table
Cell Styles
Cells
Editing
AVERAGE
=A1+A2
A
B
C
D
E
F
G
1
123
2
456
=A1+A2
3
4
5
6
7
8
9
10
11
12
13


Application


Software
Metrics
Availability
Messages
Requests


Reactive
"Responding to stimuli"
Manifesto, Actor, Messages
Resilience, Elasticity, Scalability,
Asynchronous, non-blocking
Data flow
Events, Observable
Spreadsheets
Reactive systems
Reactive streams
Reactive programming
Data flow
Back-pressure
Non-blocking
Akka, Vert.x
Akka Streams, Rx v2,
Reactor, Vert.x
Reactor, Reactive Spring,
RxJava, Vert.x

Eclipse Vert.x

Open source project started in 2012

Eclipse / Apache licensing

A toolkit for building reactive applications for the JVM

8K ★ on

Built on top of 

https://vertx.io

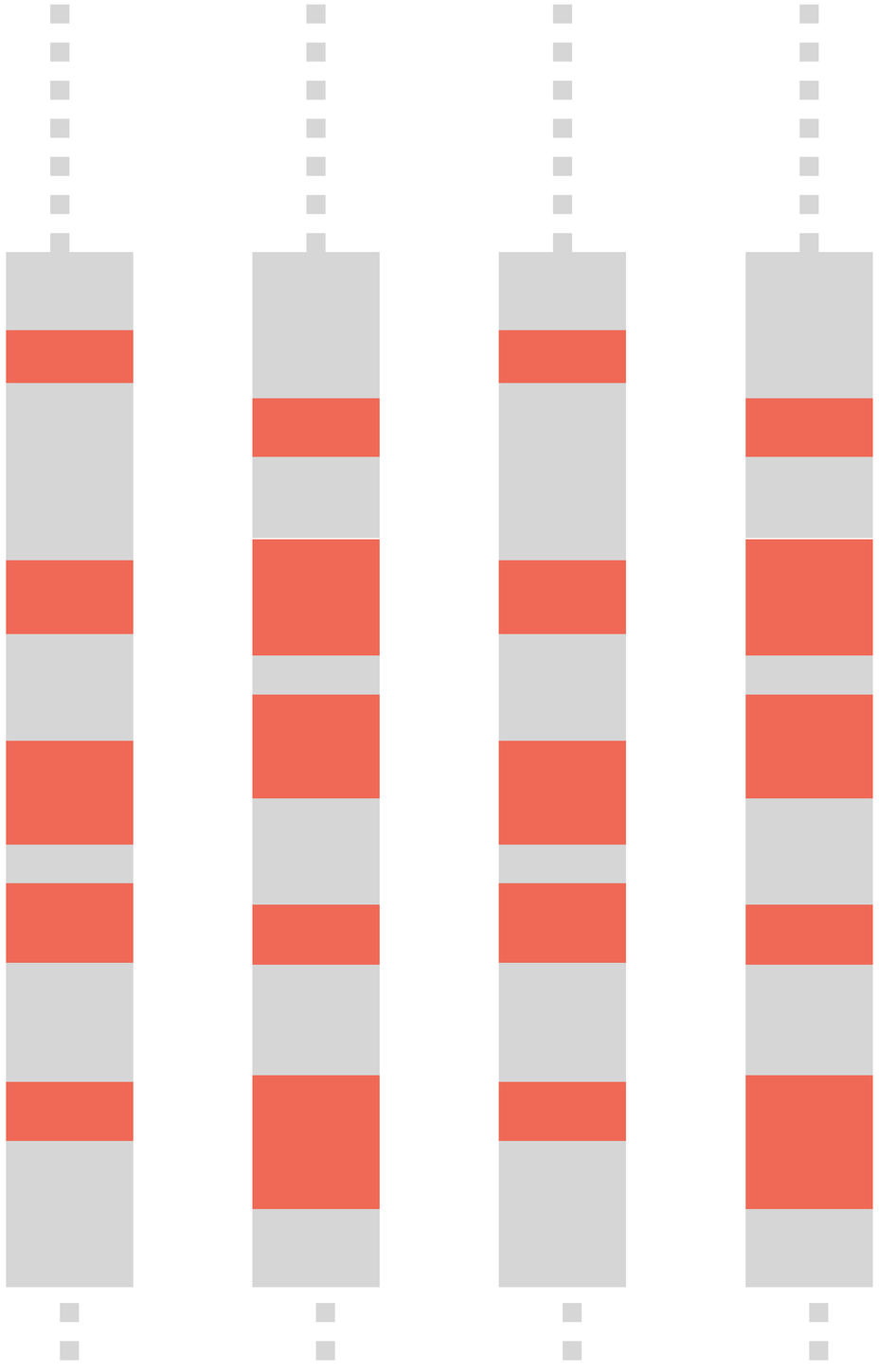
@vertx_project

Reactive foundations

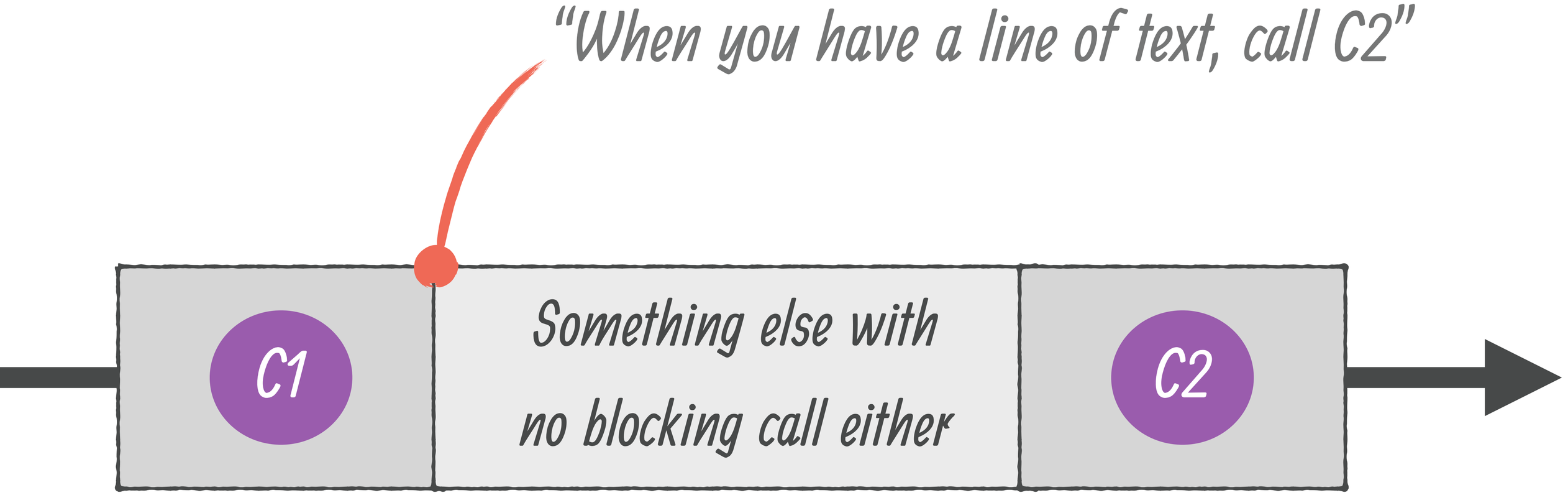
Reactive foundations

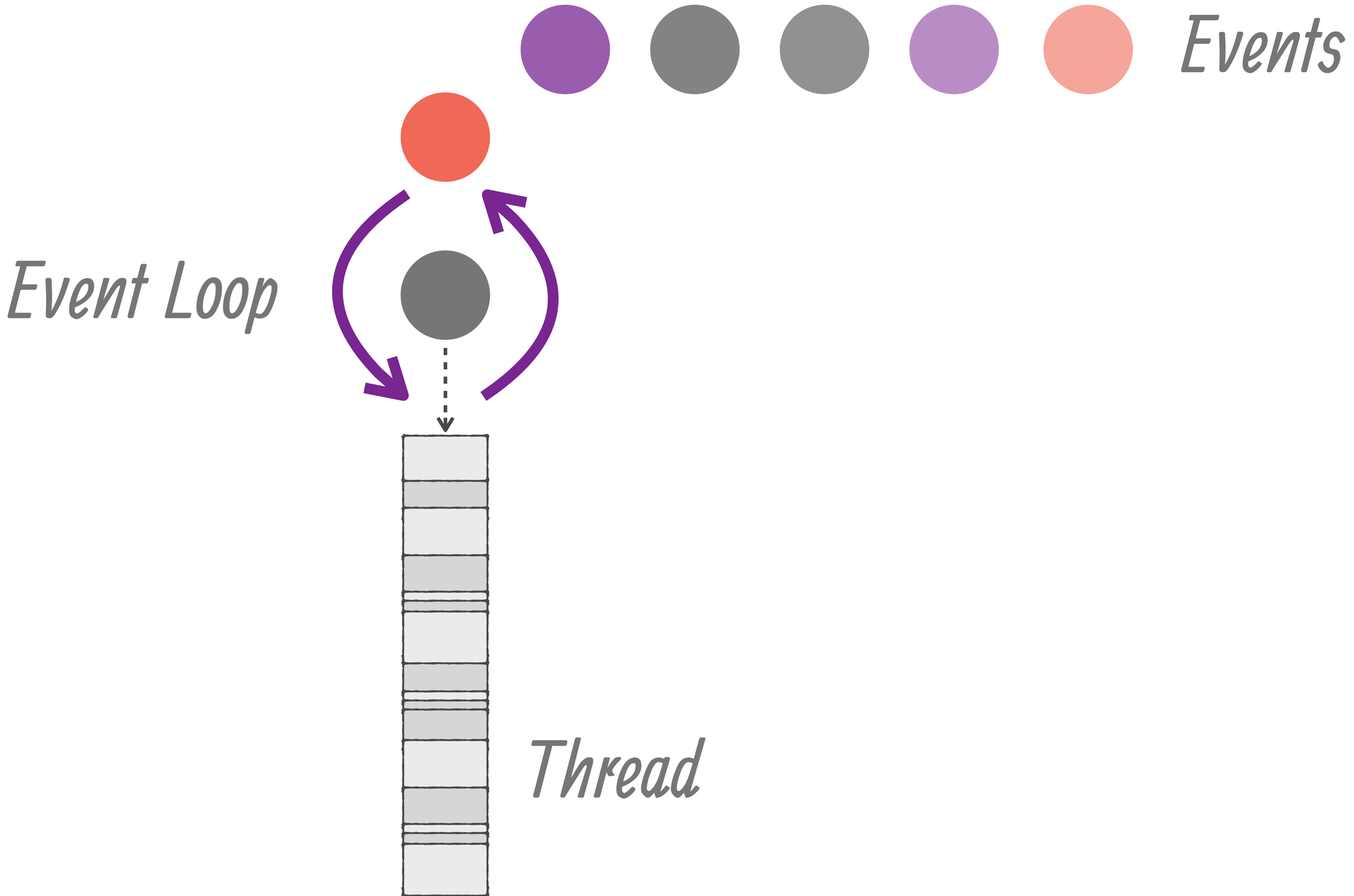
- ✓ Concurrency model
- ✓ Scalability model
- ✓ High performance networking
- ✓ Vert.x Event Bus

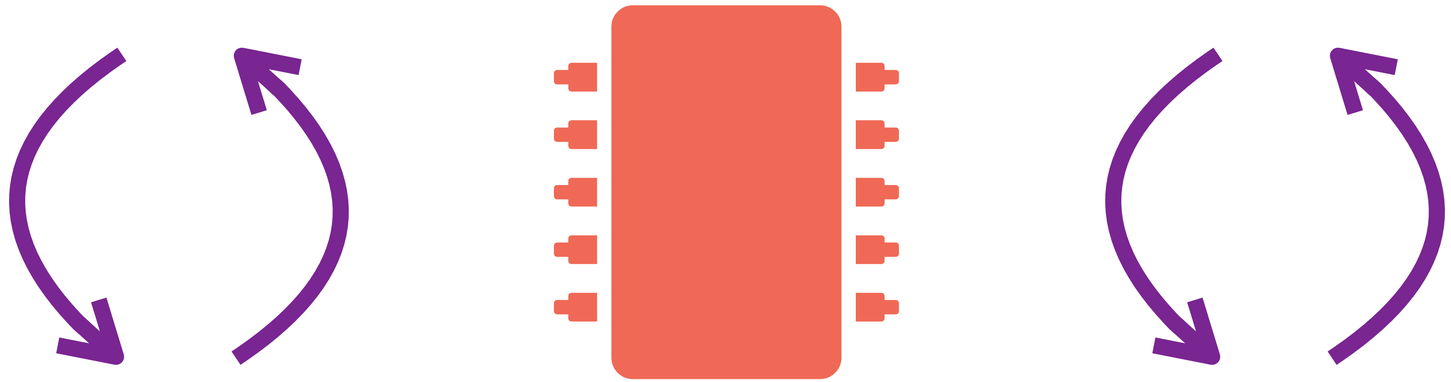
```
while (isRunning) {
  String line = bufferedReader.readLine();
  switch (line.substring(0, 4)) {
    case "ECHO":
      bufferedWriter.write(line);
      break
    //  ...
    // other cases ( ... )
    //  ...
    default:
      bufferedWriter.write("Unknown command");
  }
}
```



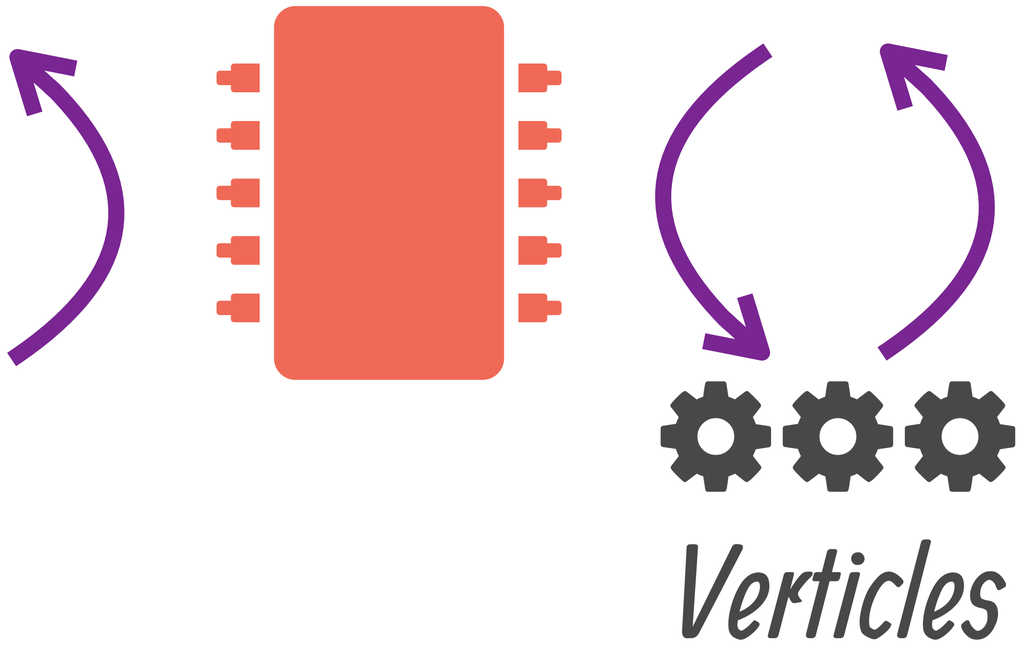
x 1000 =


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

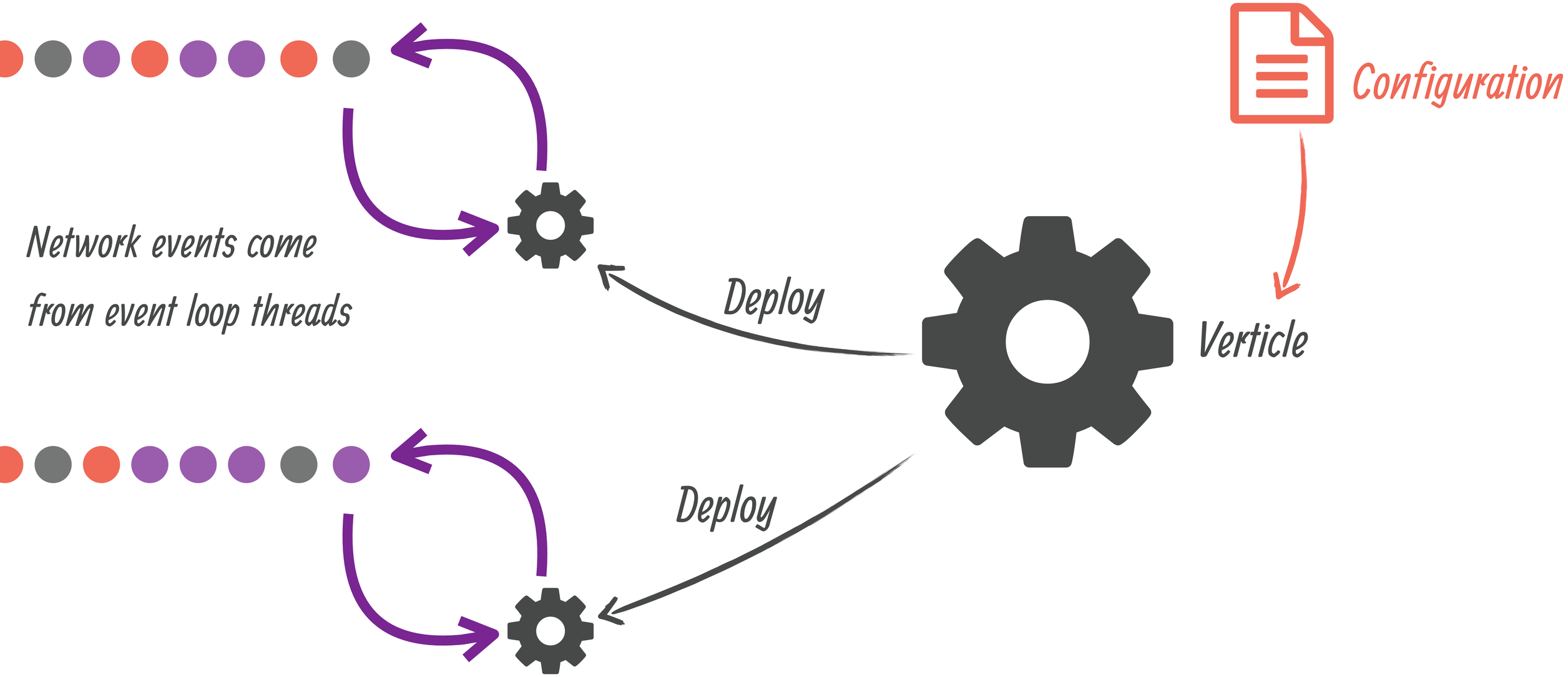

Events
Event Loop
Thread



2 event-loops per CPU core by default




```java
public class HttpVerticle extends AbstractVerticle {

  public static void main(String[] args) {
    Vertx vertx = Vertx.vertx();
    vertx.deployVerticle(HttpVerticle.class.getName());
    System.in.read();
    vertx.close();
  }

  @Override
  public void init(Vertx vertx, Context context) {
    // Associates this verticle with
    // its deployment context
    super.init(vertx, context);
  }

  @Override
  public void start() throws Exception {
    HttpServer server = vertx.createHttpServer();

    server.requestHandler(request -> {
      request
          .response()
          .end("Hello World");
    }).listen(8080);
  }
}
```

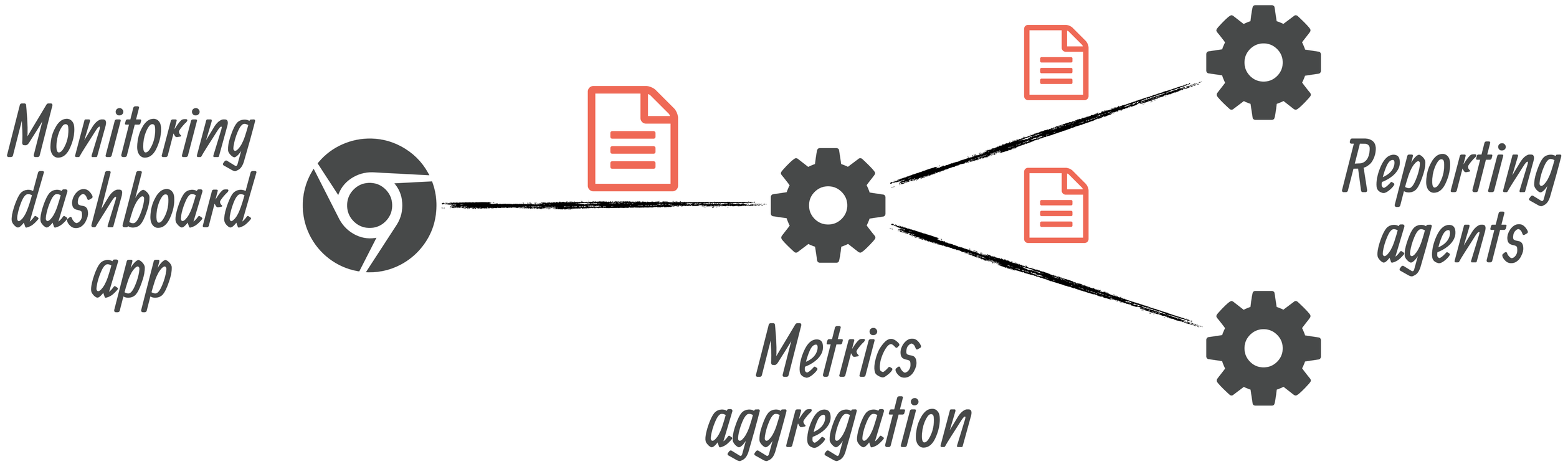

Configuration
Network events come
from event loop threads
Deploy
Verticle
Deploy

Simplified concurrency model

```
public class ChatVerticle extends AbstractVerticle {

  private Set<ServerWebSocket> webSockets = new HashSet<>();

  @Override
  public void start() throws Exception {
    HttpServer server = vertx.createHttpServer();

    server.websocketHandler(webSocket -> {

      webSocket.textMessageHandler(msg -> {
        webSockets.forEach(other -> other.writeTextMessage(msg));
      });


      webSocket.closeHandler(v -> webSockets.remove(webSocket));

      webSockets.add(webSocket);
    }).listen(8080);
  }
}
```

(demo)

Polling dashboard


Monitoring dashboard app
Metrics aggregation
Reporting agents

High performance networking

Performance

- ✓ Building bottom up
- ✓ Non blocking IO
- ✓ Reactive back-pressure

Pay the right price

- ✓ Very small footprint and startup time
- ✓ Do one thing and do it well
- ✓ Does not solve other (non) problems such as class loading or IoC
- ✓ Modular set of extensions

Non-blocking IO benefits

- ✓ Handle many connections with a few threads
  - focus on protocol concurrency
  - minimise system calls
  - more efficient for pipelined/multiplexed protocols

Non-blocking IO benefits

- ✓ Get away from thread pools
  - removes unnecessary bottlenecks
  - easier capacity planning
  - concurrency is protocol intrinsic

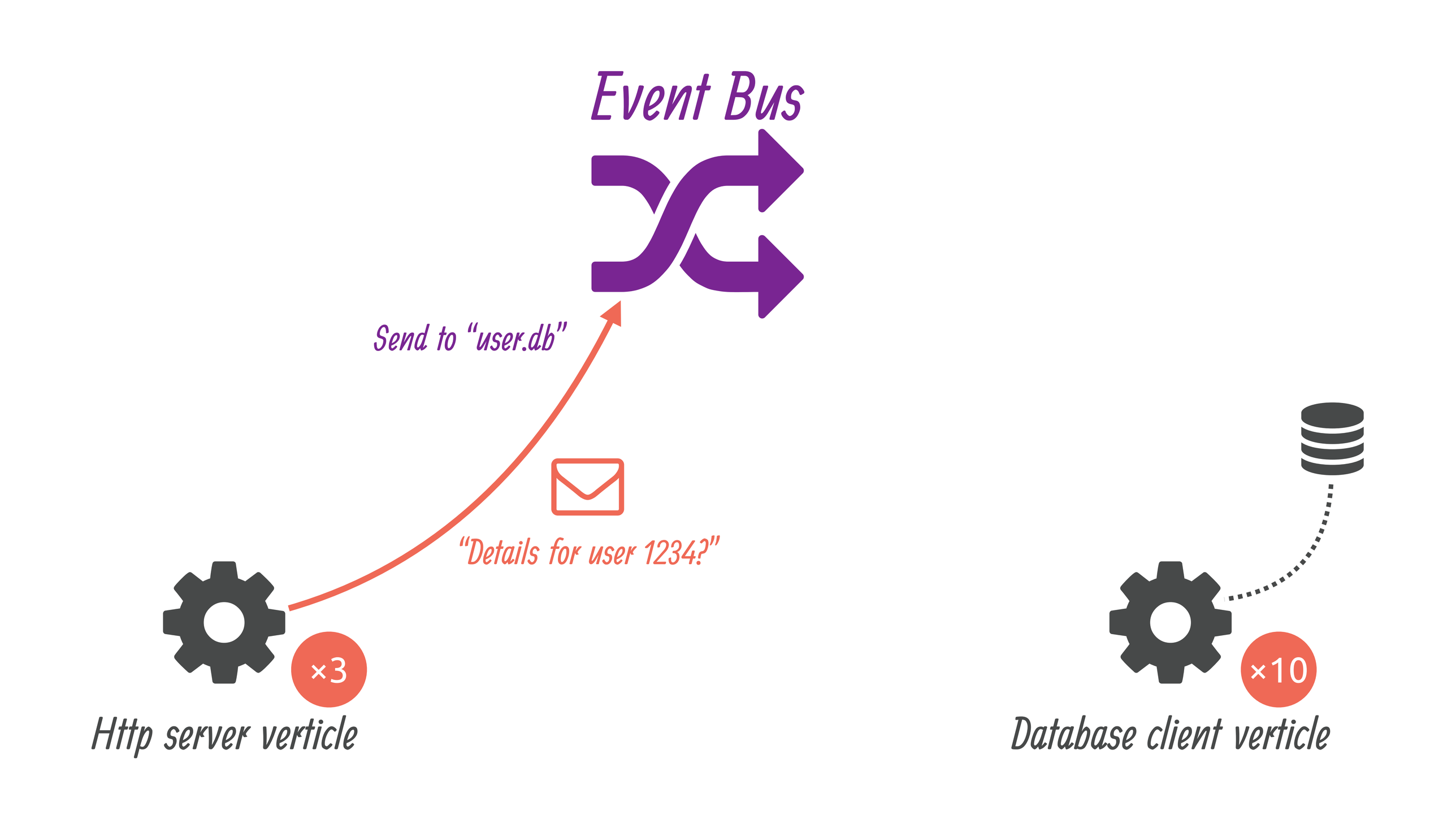
Non-blocking IO benefits

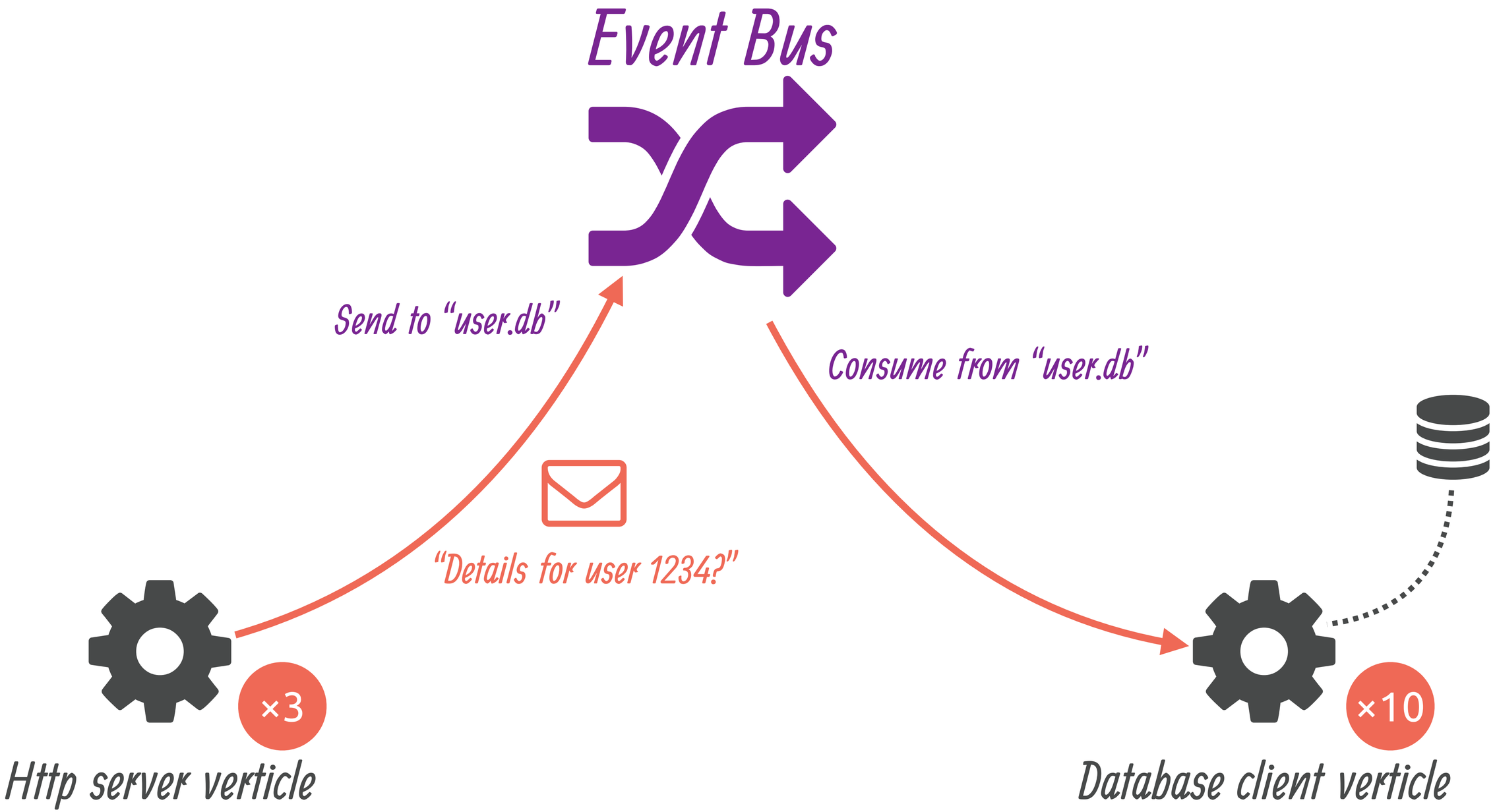
- ✓ Gracefully handle slow connection
  - remain responsive
  - don’t impact other connections

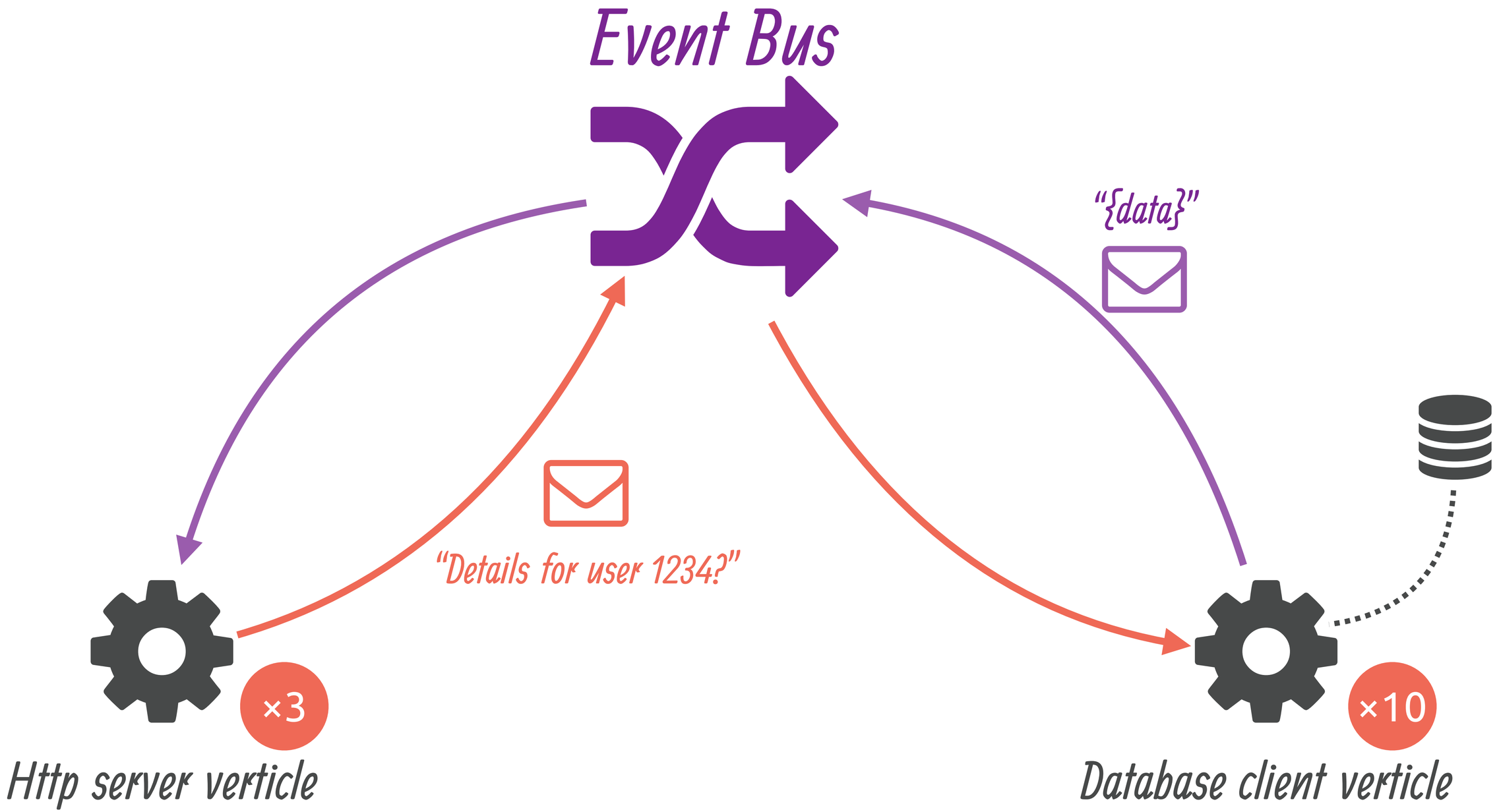
Demo: Polling dashboard

Message passing on the event bus


Http server verticle
Database client verticle


Event Bus
Send to "user.db"
"Details for user 1234?"
×3
Http server verticle
×10
Database client verticle


Event Bus
Send to "user.db"
Consume from "user.db"
"Details for user 1234?"
×3
×10
Http server verticle
Database client verticle


Event Bus
"{data}"
"Details for user 1234?"
×3
×10
Http server verticle
Database client verticle

Asynchronous messaging

`“foo.bar”, “foo-bar”, “foo/bar”, …`

Point to point (possible response back)
Publish / subscribe

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

Demo: push dashboard

Distributed across Vert.x nodes

Hazelcast, Ignite, Infinispan, ...

TCP bridge interface

Go, Python, C, JavaScript, Swift, C#, ...

SockJS bridge

Seamless frontend / backend messaging

Demo: bridge dashboard

Reactive Programming with Vert.x and RxJava

RxJava

Data and **events** **flows**

Organising **transformation** of **data** and **coordination** of **events**

Makes most sense with many **sources of events**

Motivation

`Future<List<T>>` is not always appropriate

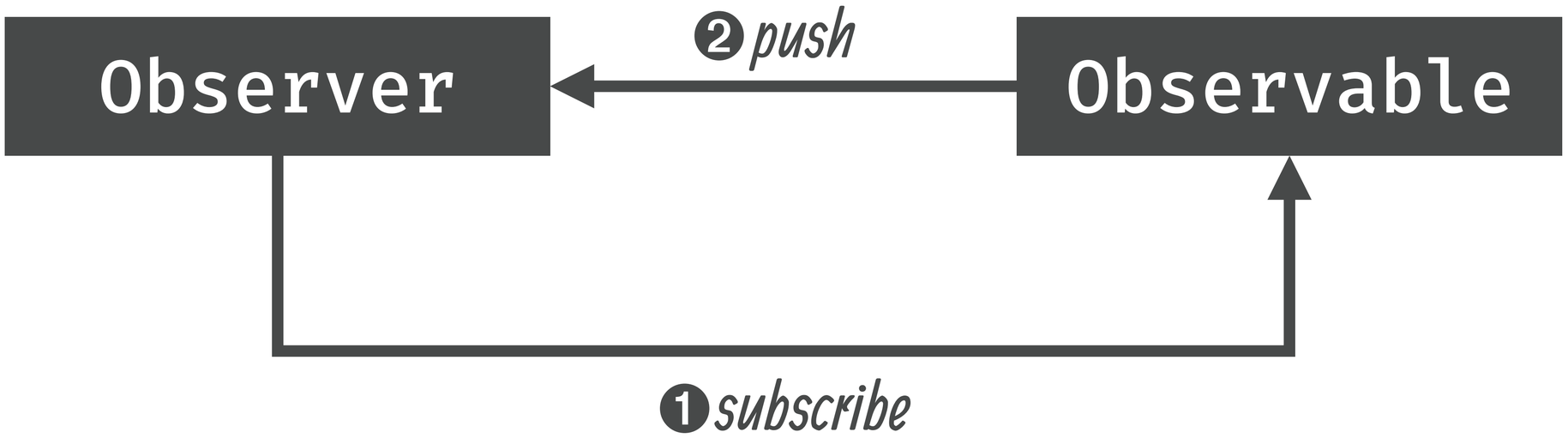
Dealing with **latencies**

Functional programming influence

RxJava 2 types

Reactive	`Completable`	`Maybe<T>`	`Single<T>`	`Observable<T>`
Interactive	`void`	`Optional<T>`	`T`	`Iterable<T>`

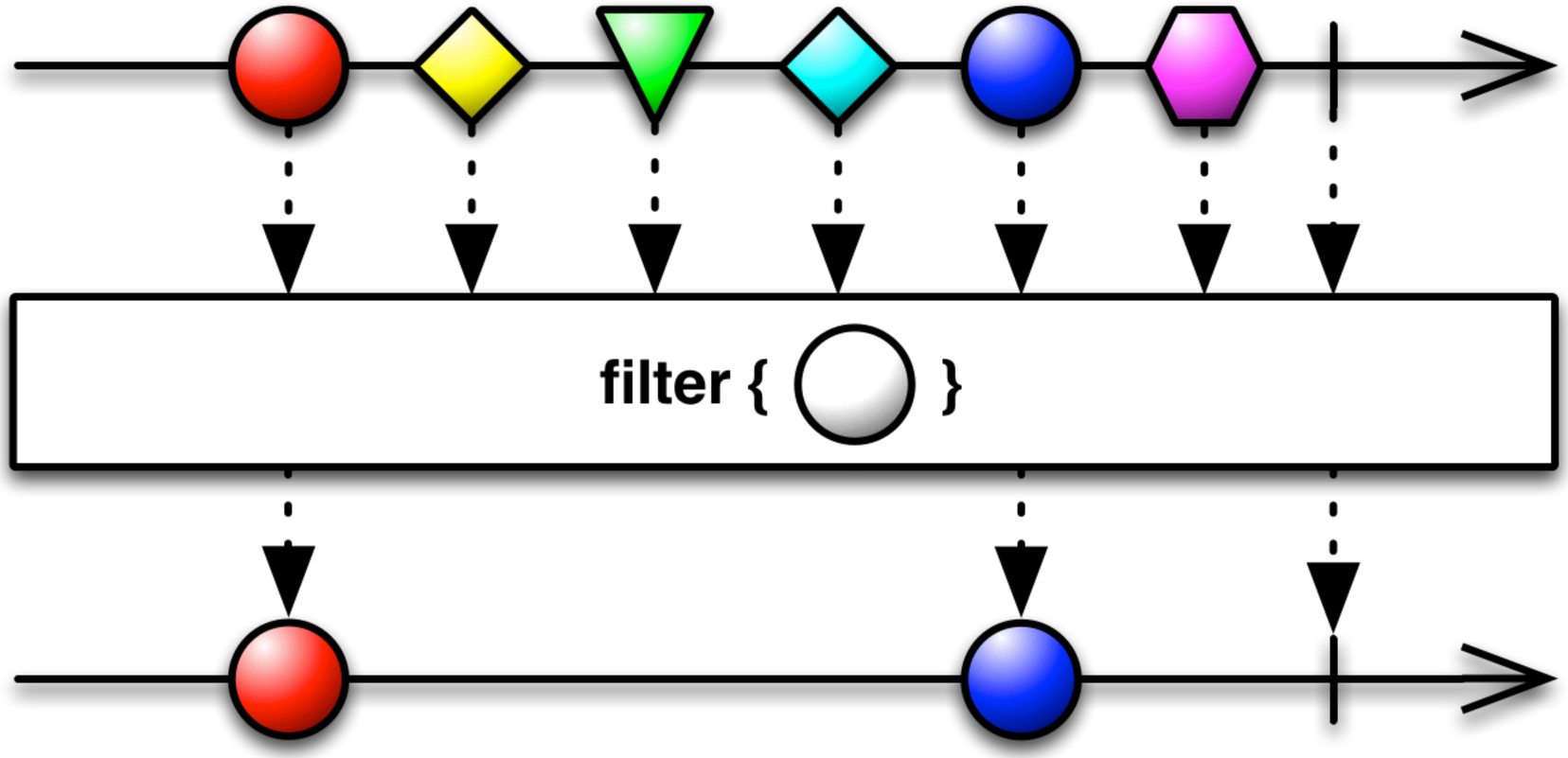
Event push

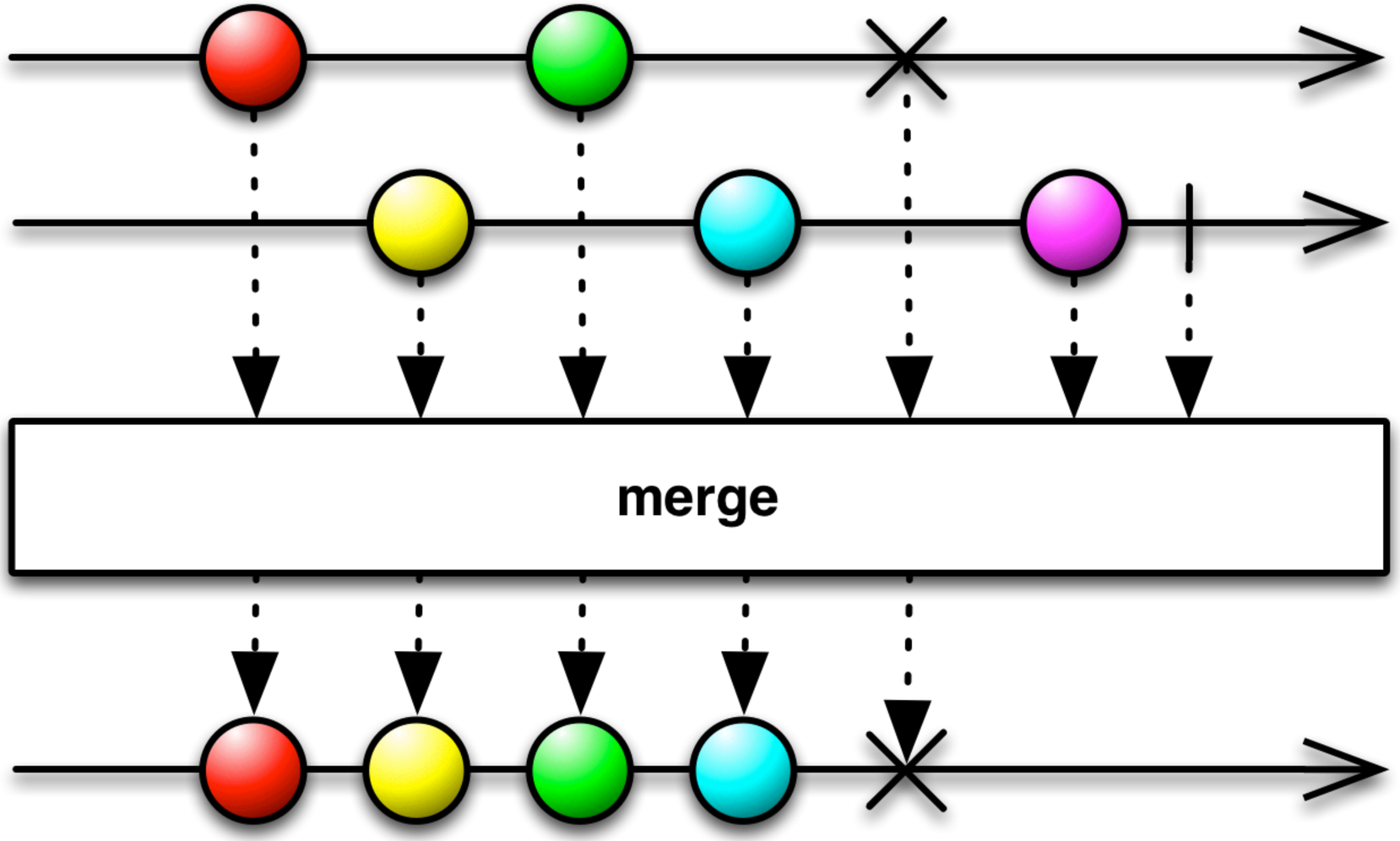


Iterable / Observable

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

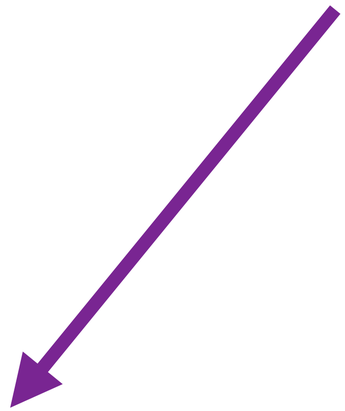
```
observable.subscribe(item ->
{
  ❶ // onNext
}, error -> {
  ❷ // onError
}, () -> {
  ❸ // onCompleted
});
```


filter { }


merge

Rxified API

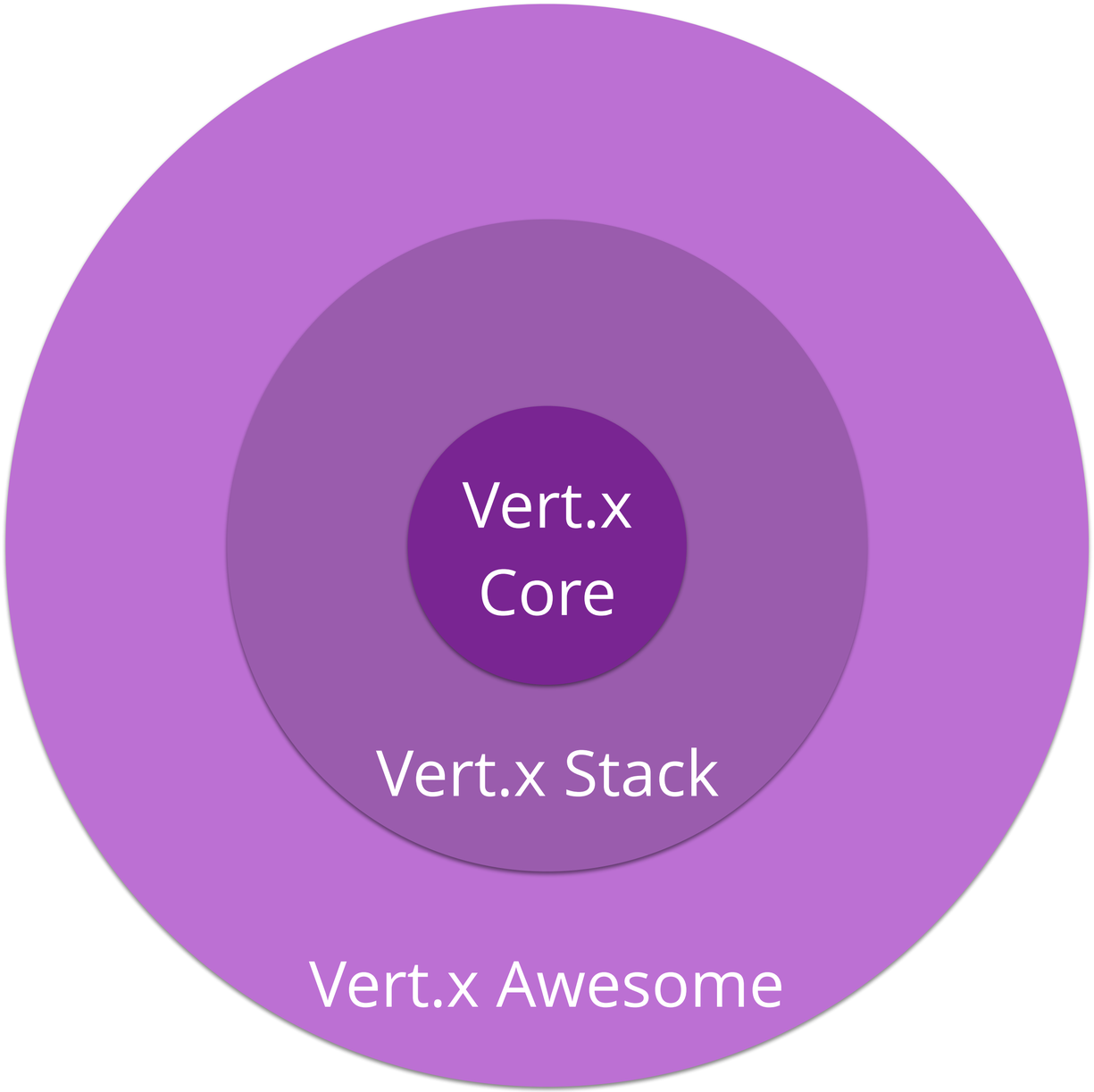
```
void listen(int port, Handler<AsyncResult<HttpServer>> ar)
```



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

Demo (RxJava)

Outro


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

Unified end-to-end reactive model + ecosystem

(not just APIs…)

For *all* kinds of distributed applications

(even the small-scale ones)

Flexible toolkit, not a framework

(your needs, your call)

Guide to async programming with Vert.x for Java developers

https://goo.gl/AcWW3A

Building Reactive Microservices in Java

https://goo.gl/ep6yB9