



Erwin de Gier

The definite guide to reactive programming in Java



github.com/erwindeg



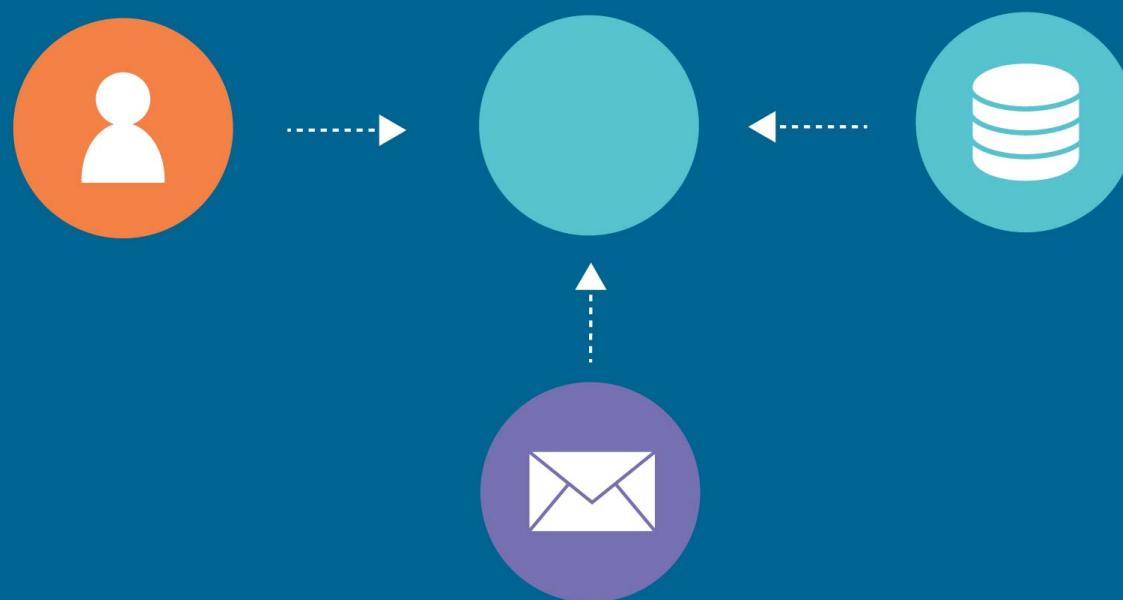
@erwindeg

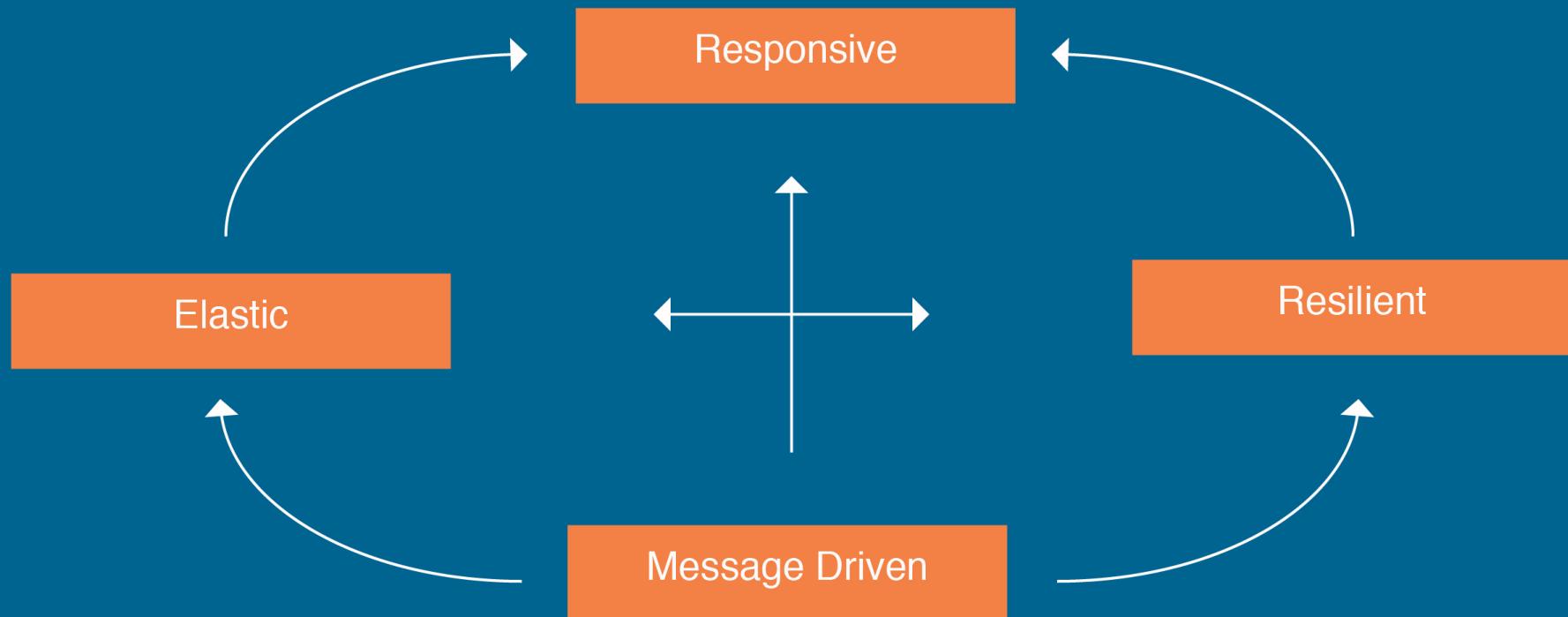


edegier.nl

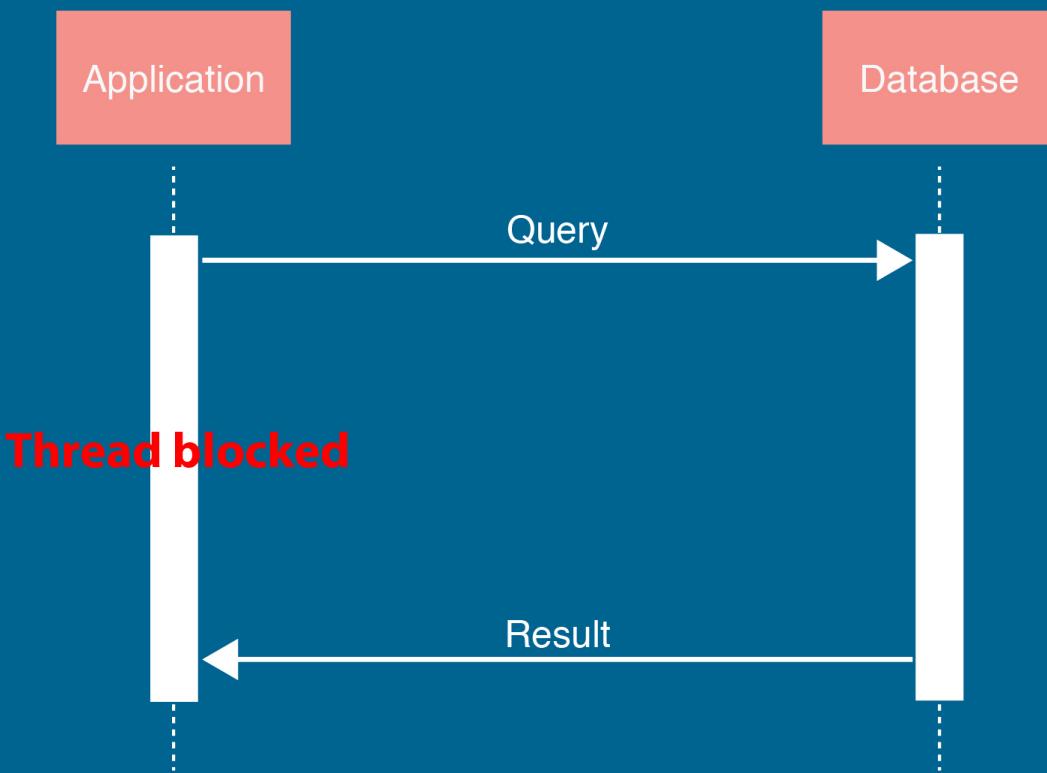
TRIFORK.
...think software

Why Reactive?

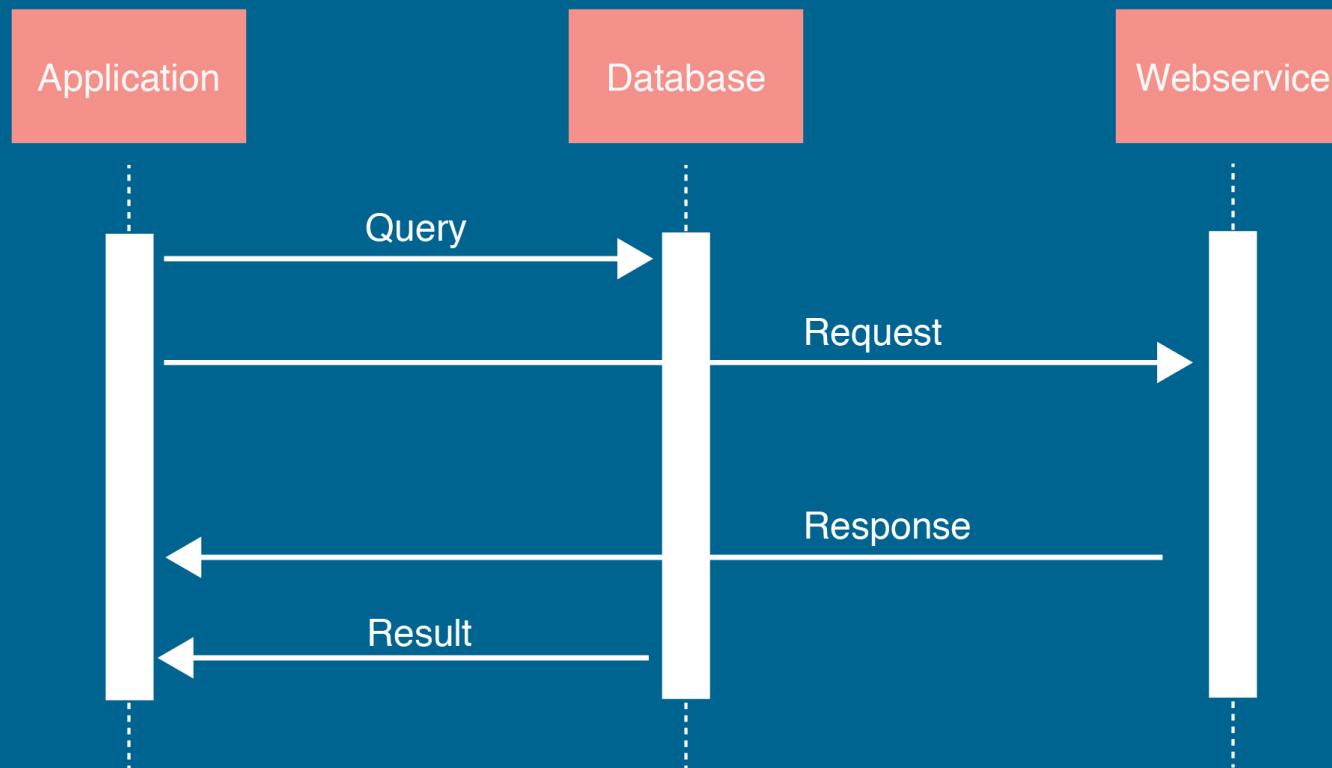




Synchronous



Asynchronous



```
//PersonRepository sync
public List<Person> findByName(String name);

public BigDecimal getIncome(String name);

//PersonRepository async
public void findByName(String name, Callback<List<Person>> persons);

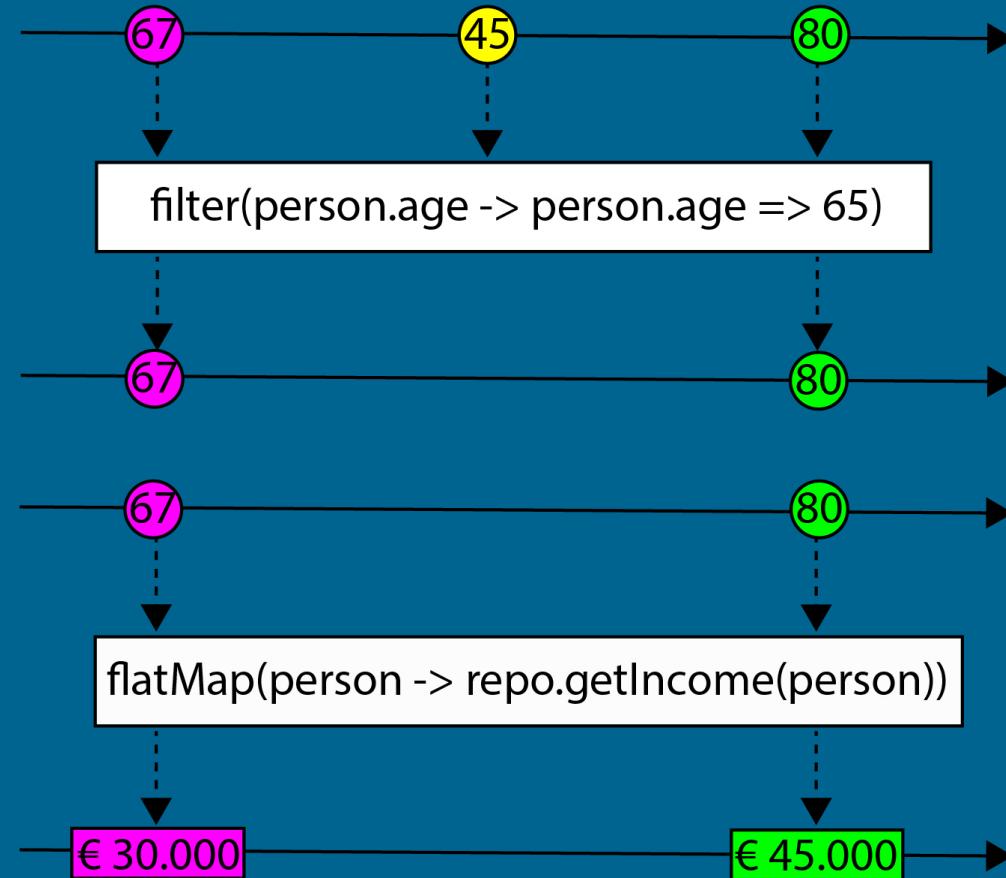
public void getIncome(String name, Callback<BigDecimal> income);
```

```
//Client call
repository.findByName("Erwin",
    persons -> {
        persons.stream().filter(person -> person.getAge() >= 65)
            .forEach(person -> {
                repository.getIncome(person, income ->
                    totalIncome = totalIncome.add(income));
            });
    }
);
```

```
//PersonRepository
public Observable<Person> findByName(String name);
public Observable<BigDecimal> getIncome(Person person);

//Client call
repository.findByName("Erwin")
    .filter(person -> person.getAge() >= 65)
    .flatMap(person -> repository.getIncome(person))
    .subscribe(income -> totalIncome = totalIncome.add(income));
```

RxJava



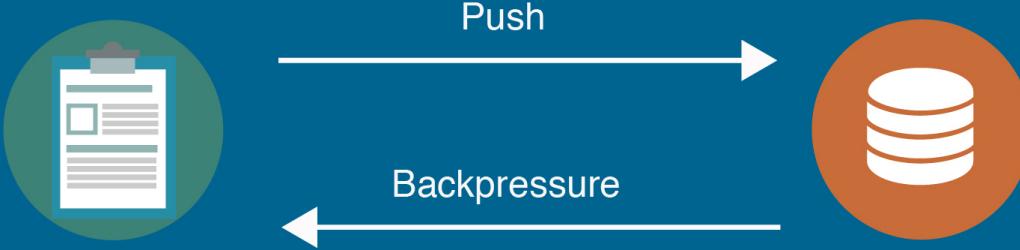
Java 8 Streams vs. RX Observables

Pull vs. Push

Finite vs. Infinite

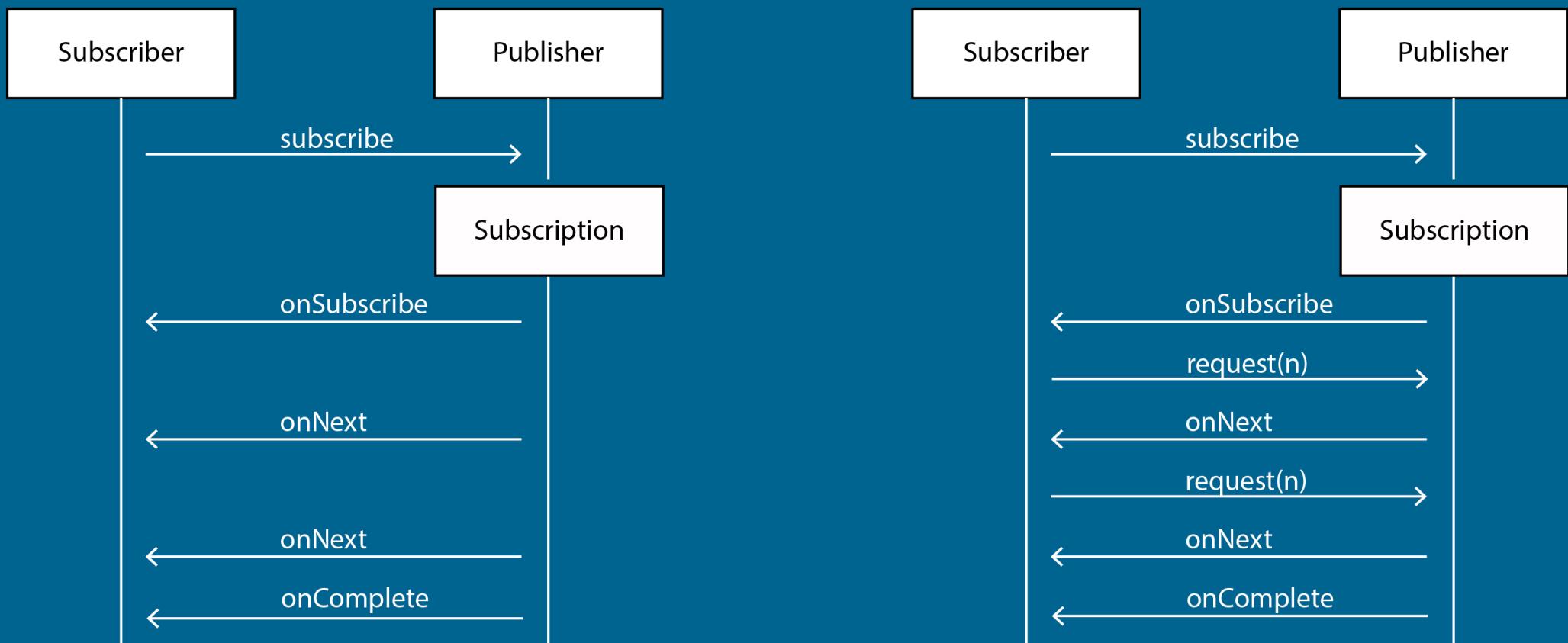
Sync vs. Async

RxJava 2



<http://www.reactive-streams.org>

Backpressure



Reactive Streams

✓ RxJava 2

✓ Project Reactor

✓ Akka Streams

✓ Java 9 Flow API

Java 9

✓ Flow API

✓ Interfaces copied from reactive streams

✓ Connecting different Rx implementations

✓ Easier to use Reactive Frameworks

Java vs. Reactive Streams

	No Value	Single Value	Multiple Values
Java Blocking	void	T	Iterable<T>
Java Non-blocking	CompletableFuture<Void>	CompletableFuture<T>	CompletableFuture<List<T>>
Reactive Streams	Publisher<Void>	Publisher<T>	Publisher<T>
RxJava	Observable<Void>	Single<T>	Observable<T>
Project Reactor	Mono<Void>	Mono<T>	Flux<T>
Akka Streams	Source<Void>	Source<T>	Source<T>
Java 9 Flow	Flow.Publisher<Void>	Flow.Publisher<T>	Flow.Publisher<T>

Java 8 Streams vs. Reactive Streams

```
Stream<Integer> j = Arrays.asList(1, 2, 3, 4, 5).stream();

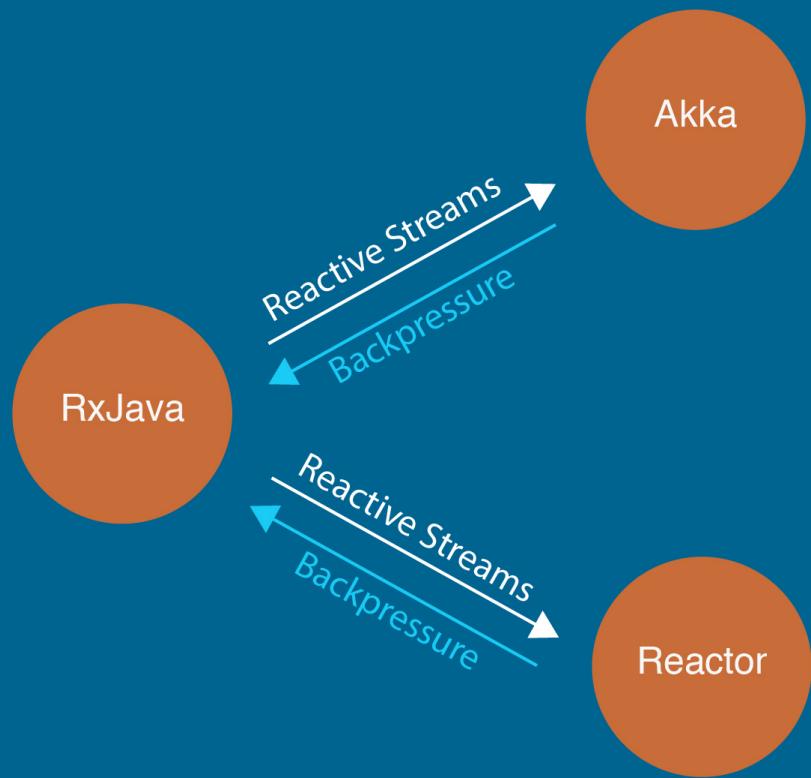
j.map(i -> i * 10)
  .forEach(System.out::println);

j.map(i -> i + 5)
  .forEach(System.out::println); //IllegalStateException
```

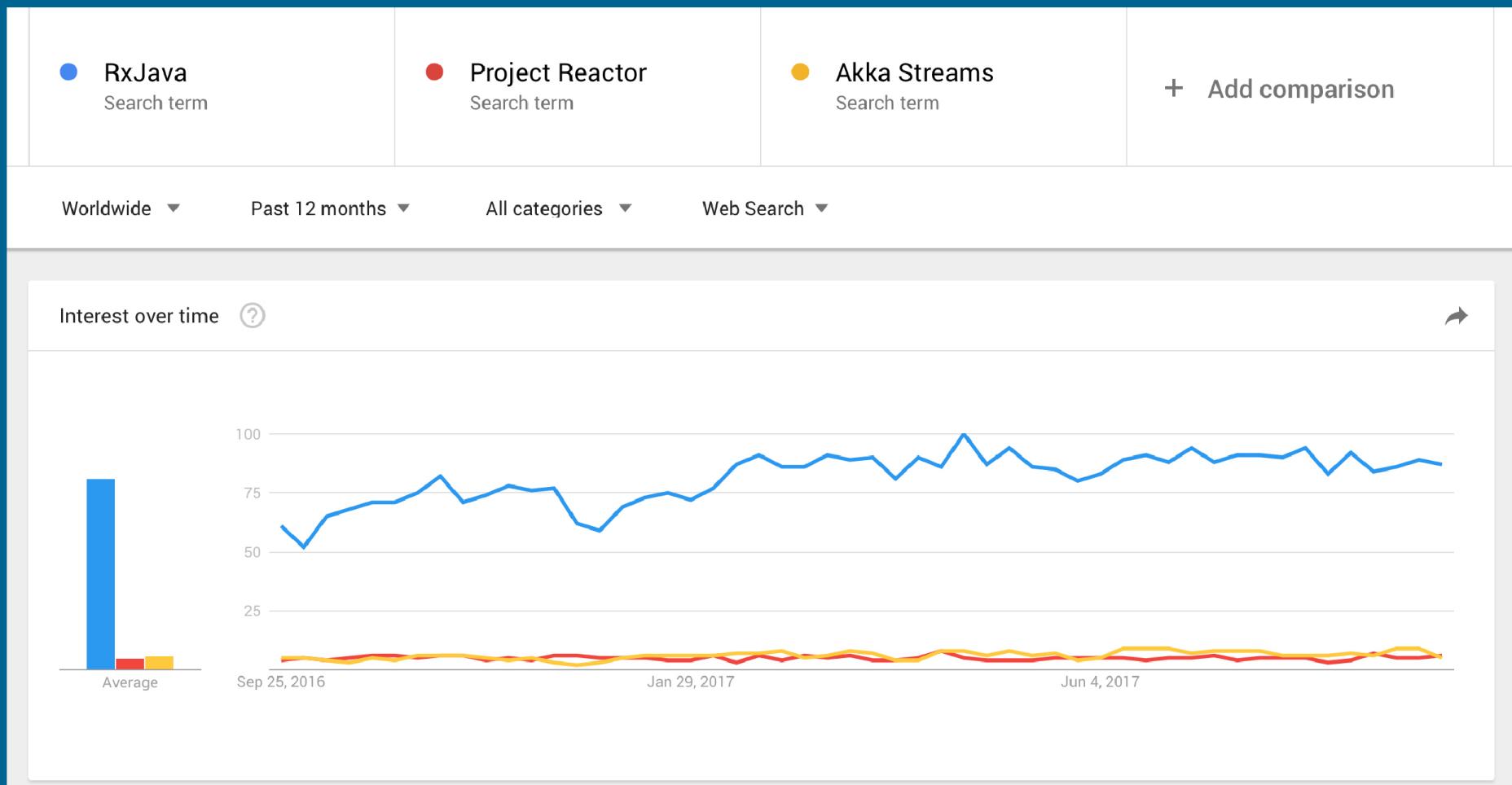
```
Flux<Integer> j = Flux.just(1, 2, 3, 4, 5);

j.map(i -> i * 10)
  .subscribe(System.out::println);

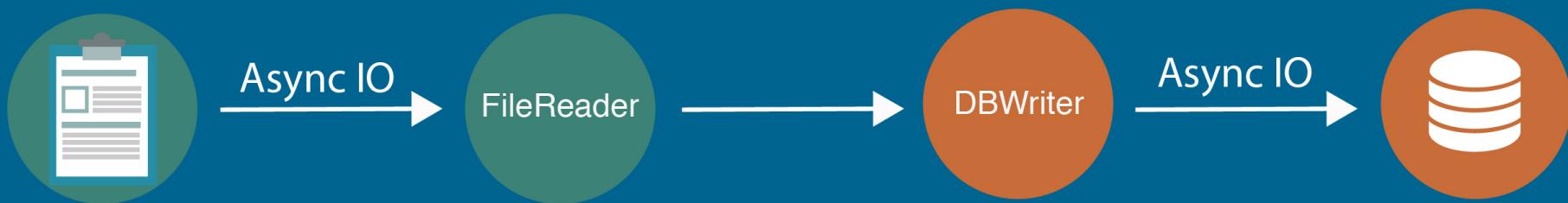
j.map(i -> i + 5)
  .subscribe(System.out::println);
```



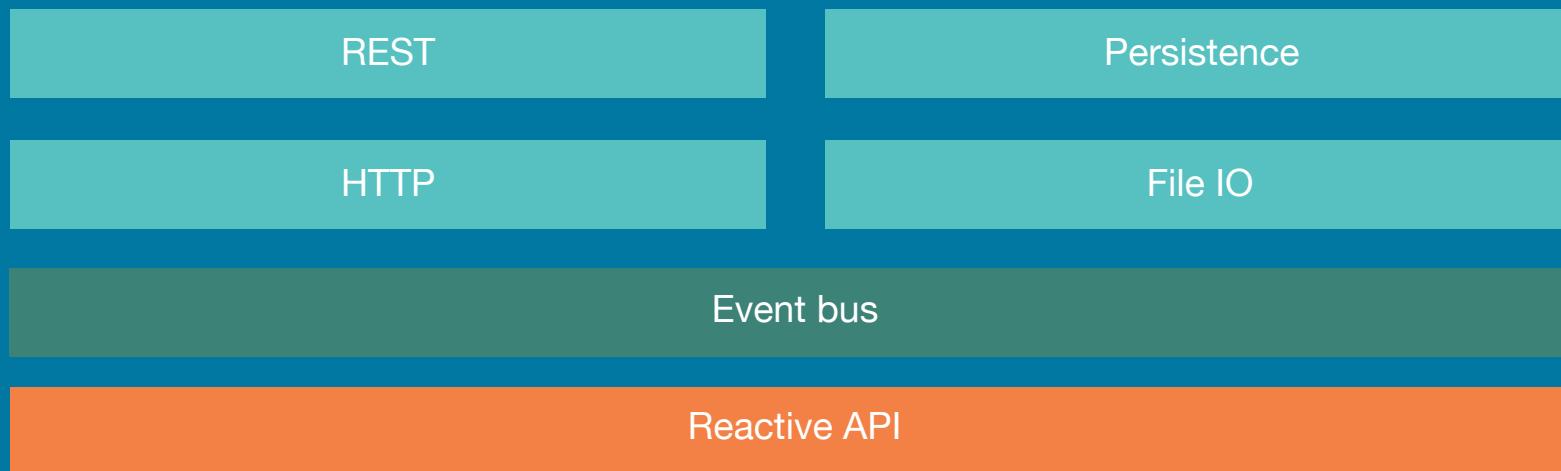
Popularity



Async operations



Reactive Stack



Reactive Frameworks

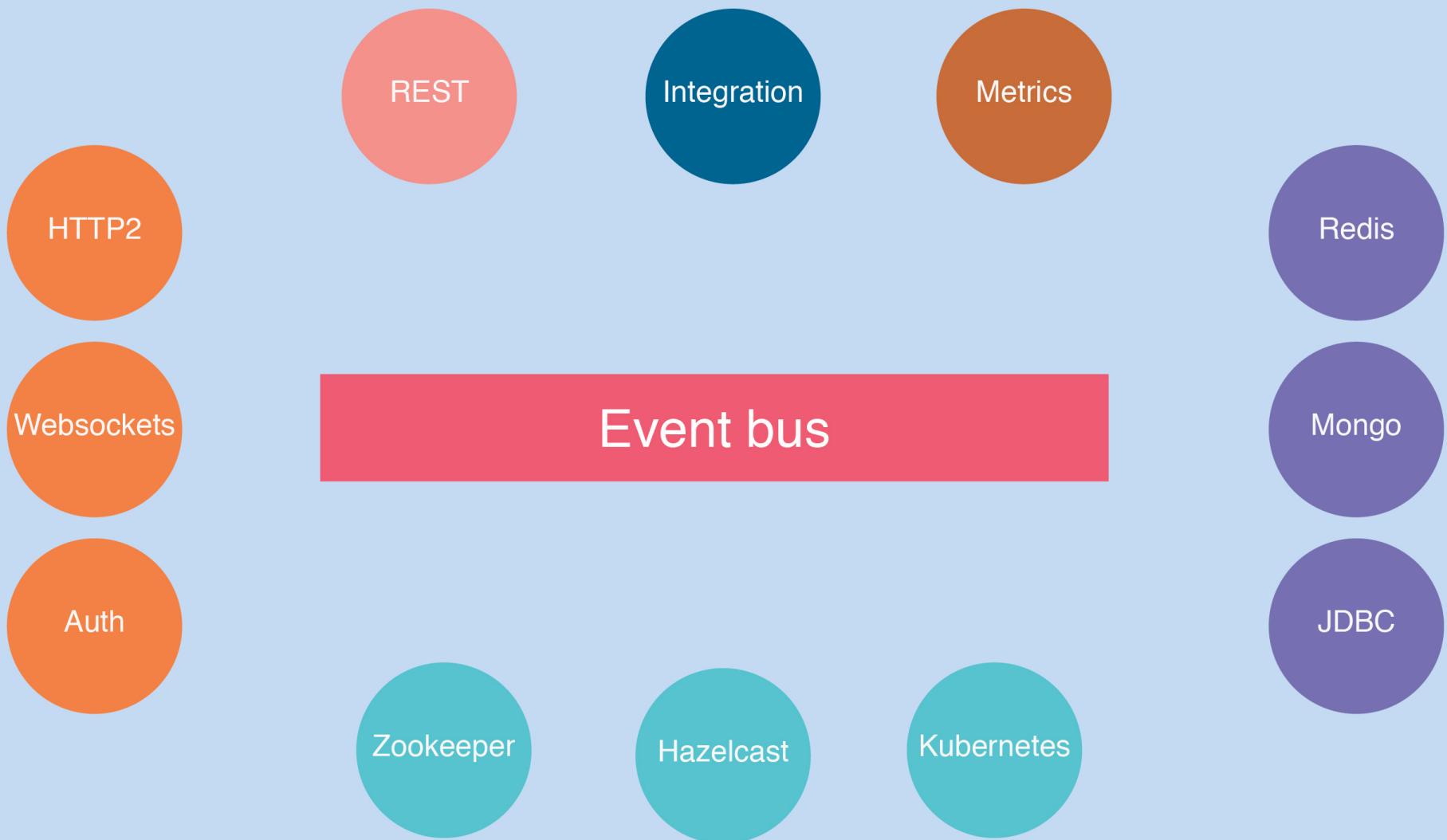
✓ Vert.x

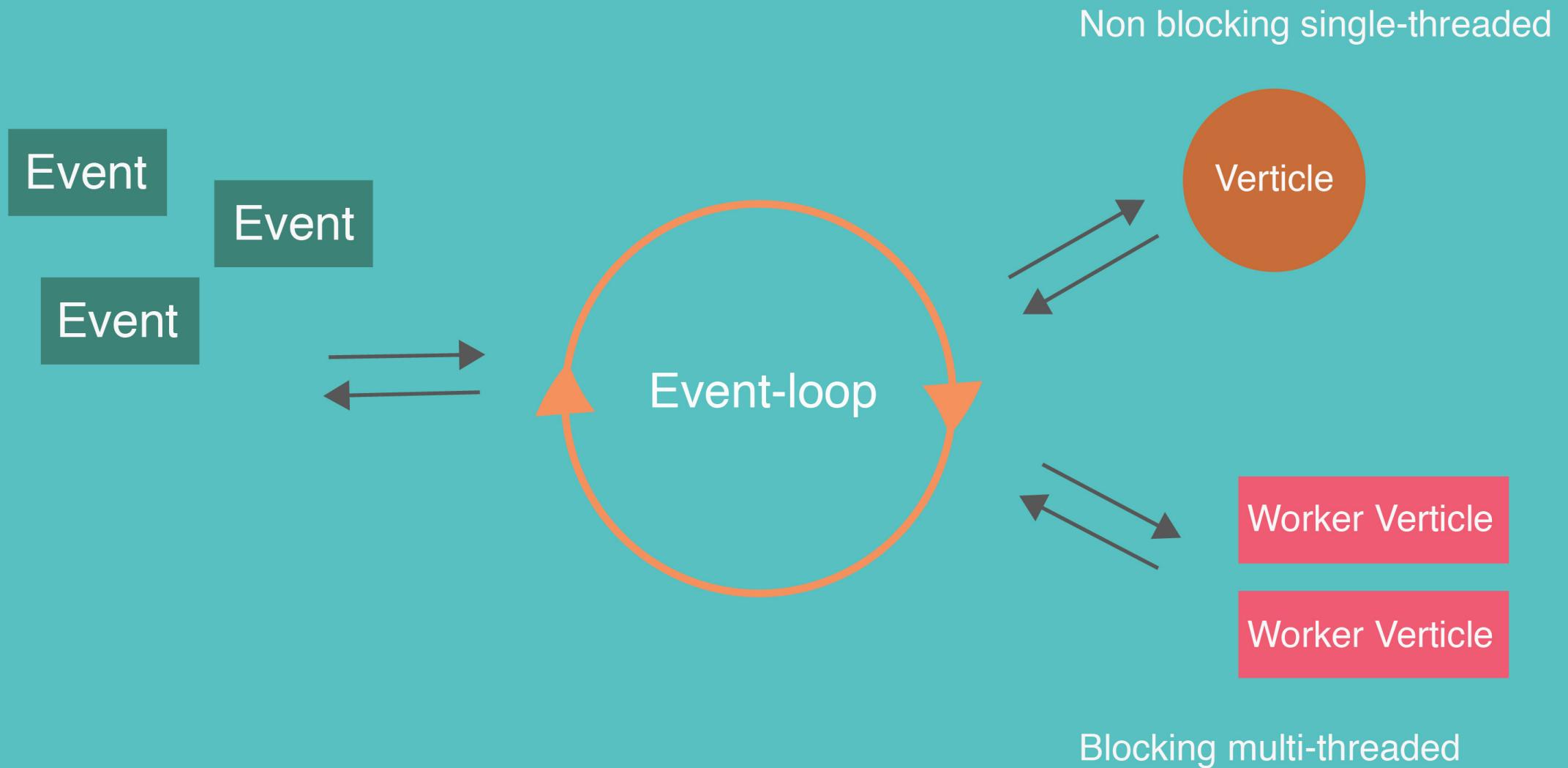
✓ Spring 5

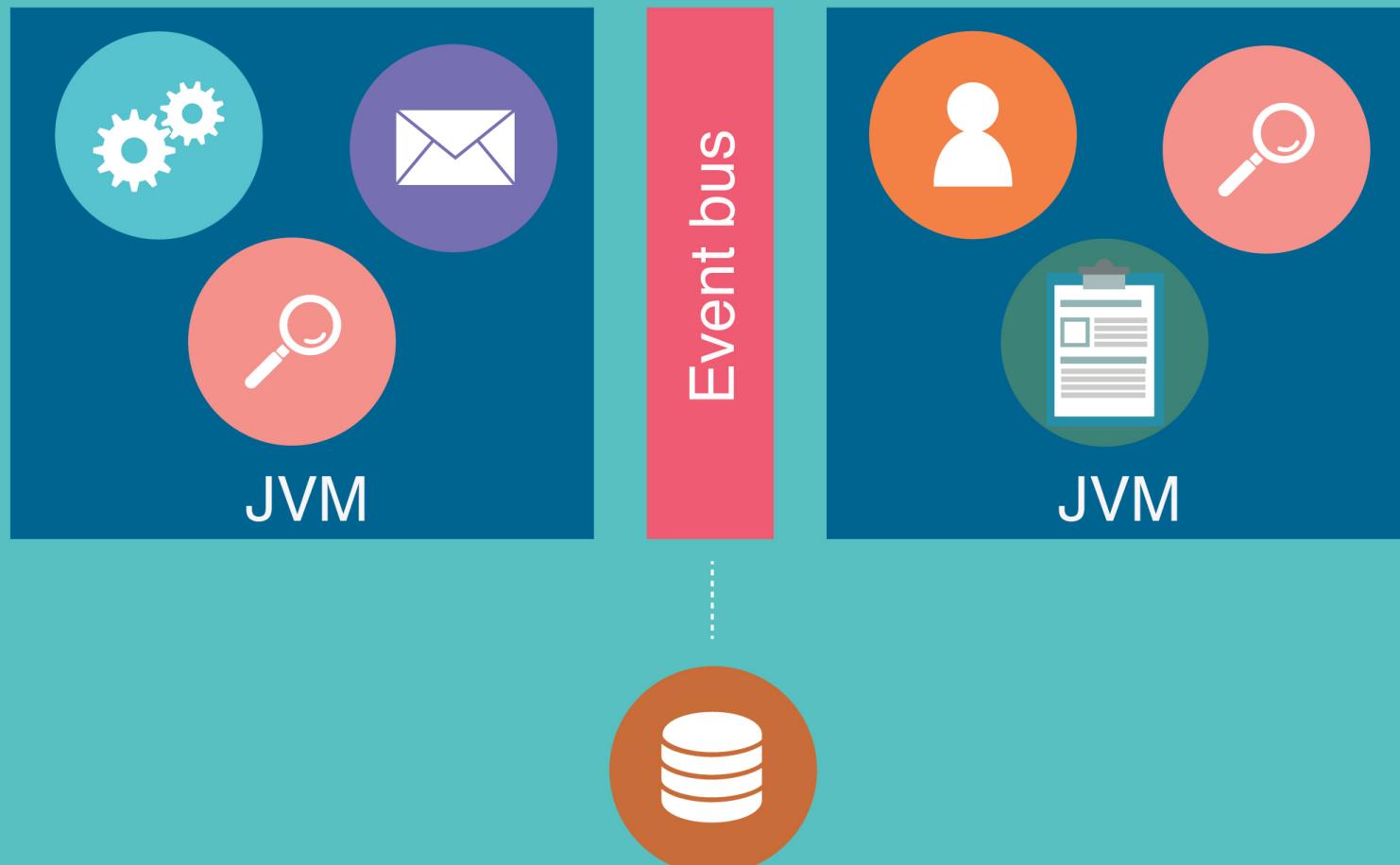
✓ Akka

VERT.X

- ✓ Runnable Jar
- ✓ Reactive
- ✓ Polyglot
- ✓ Distributed







```
public class HelloWorldVerticle extends AbstractVerticle{

@Override
public void start() throws Exception {
    vertx.eventBus().consumer("hello-channel",message -> System.out.println(message.body()));

    vertx.eventBus().send("hello-channel","Hello world!");
}

}
```

```
public class HelloWorldRestVerticle extends AbstractVerticle{

@Override
public void start() {
    Router router = Router.router(vertx);
    router.get("/hello").handler(routingContext -> {
        routingContext.response()
            .end(new JsonObject().put("message", "Hello World").encode());
    });
}

vertx.createHttpServer().requestHandler(router::accept).listen(8080);
}
```

Spring 5

✓ Spring Webflux

✓ Project Reactor

✓ Reactive Data Repositories

✓ Project Reactor event bus

```
@RestController("hello")
public class HelloController {

    @GetMapping
    Mono<String> hello(){
        return Mono.just("Hello World");
    }
}

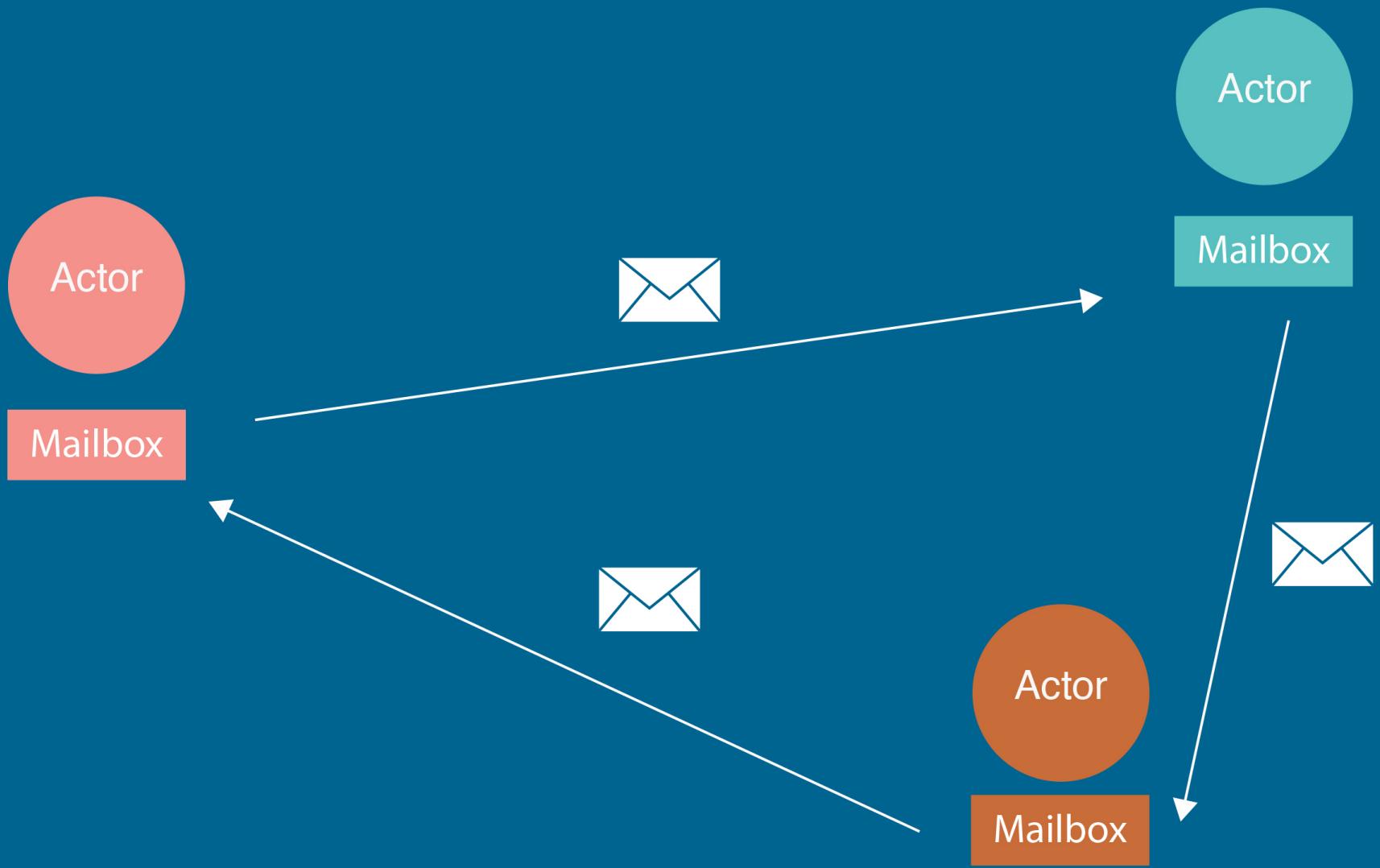
@RestController("person")
public class PersonController {
    @Autowired
    private ReactivePersonRepository personRepository;

    @GetMapping
    Flux<Person> getPersons() {
        return this.personRepository.findAll();
    }

    @PostMapping
    Mono<ResponseEntity<Person>> savePerson(@RequestBody Person person) {
        return this.personRepository.save(person)
            .map(result -> new ResponseEntity<>(result, HttpStatus.CREATED));
    }
}
```

AKKA

- ✓ Actor model
- ✓ Akka HTTP
- ✓ Scala
- ✓ Message driven



Actor

- State
- Behavior
- Mailbox
- Supervision of child actors

```
public class HelloWorld extends UntypedActor {

    @Override
    public void preStart() {
        // create the greeter actor
        final ActorRef greeter = getContext().actorOf(Props.create(Greeter.class), "greeter");
        // tell it to perform the greeting
        greeter.tell(Greeter.Msg.GREET, getSelf());
    }

    @Override
    public void onReceive(Object msg) {
        getContext().stop(getSelf());
    }
}

public class Greeter extends UntypedActor {

    @Override
    public void onReceive(Object msg) {
        System.out.println("Hello World!");
        getSender().tell(Msg.DONE, getSelf());
    }
}
```

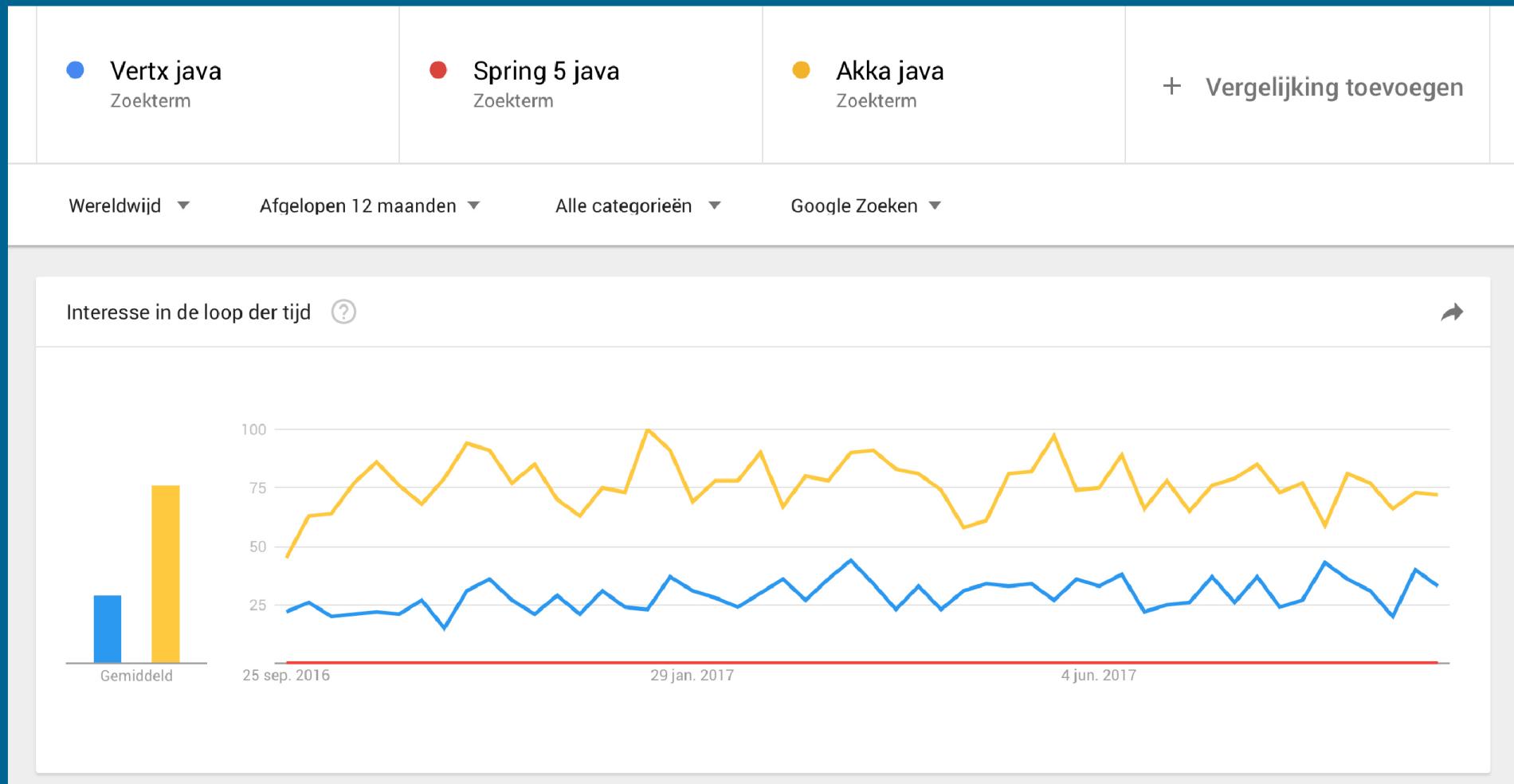
```
public class HttpServer extends HttpApp {

    public static void main(String[] args) throws IOException {
        ActorSystem system = ActorSystem.create();

        new HttpServer().bindRoute("localhost", 8080, system);
    }
    @Override
    public Route createRoute() {
        Route helloRoute = handleWith((ctx)
            -> ctx.complete("Hello World!"));

        return route(get(path("hello")).route(helloRoute)));
    }
}
```

Popularity



Vert.x vs Spring vs Akka



Landscape overview

	Event model	Annotations	Actor model
Framework	VERT.X	 spring by Pivotal™	
API	 RxJava		

“Unless you can model your entire system synchronously, a single asynchronous source breaks imperative programming”

Jake Wharton