Erwin de Gier

Real-world reactive programming in Java

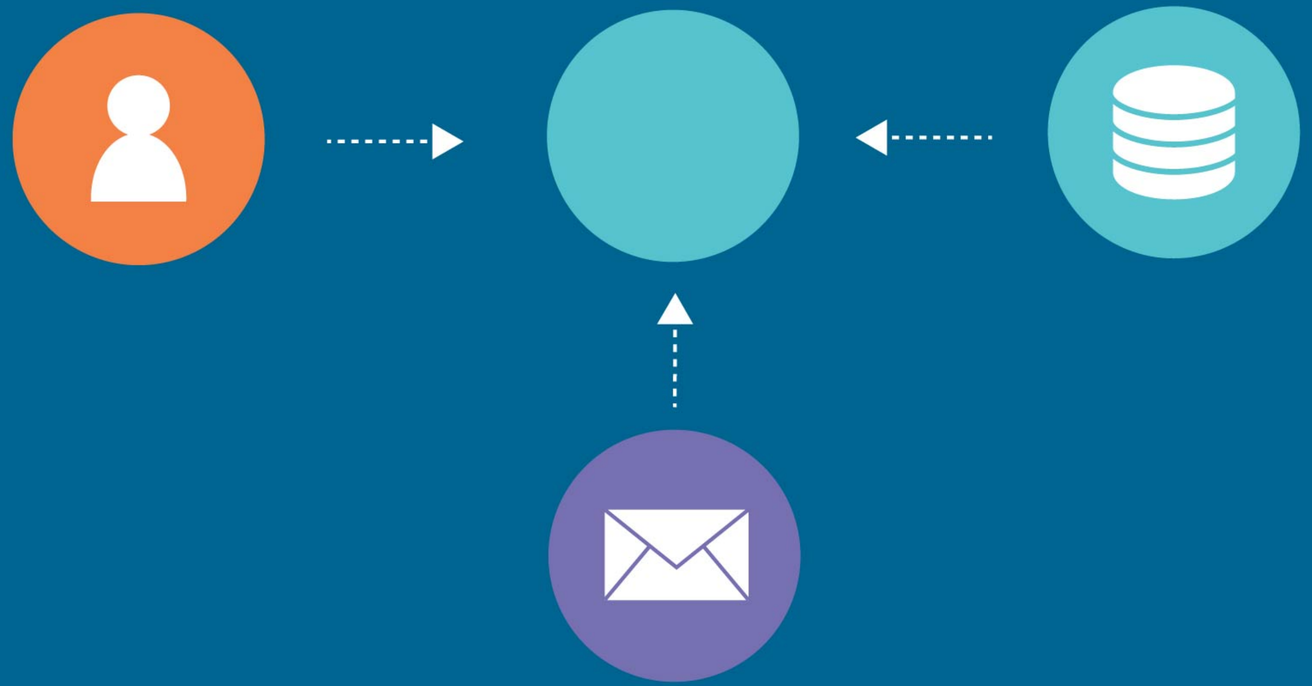
@erwindeg

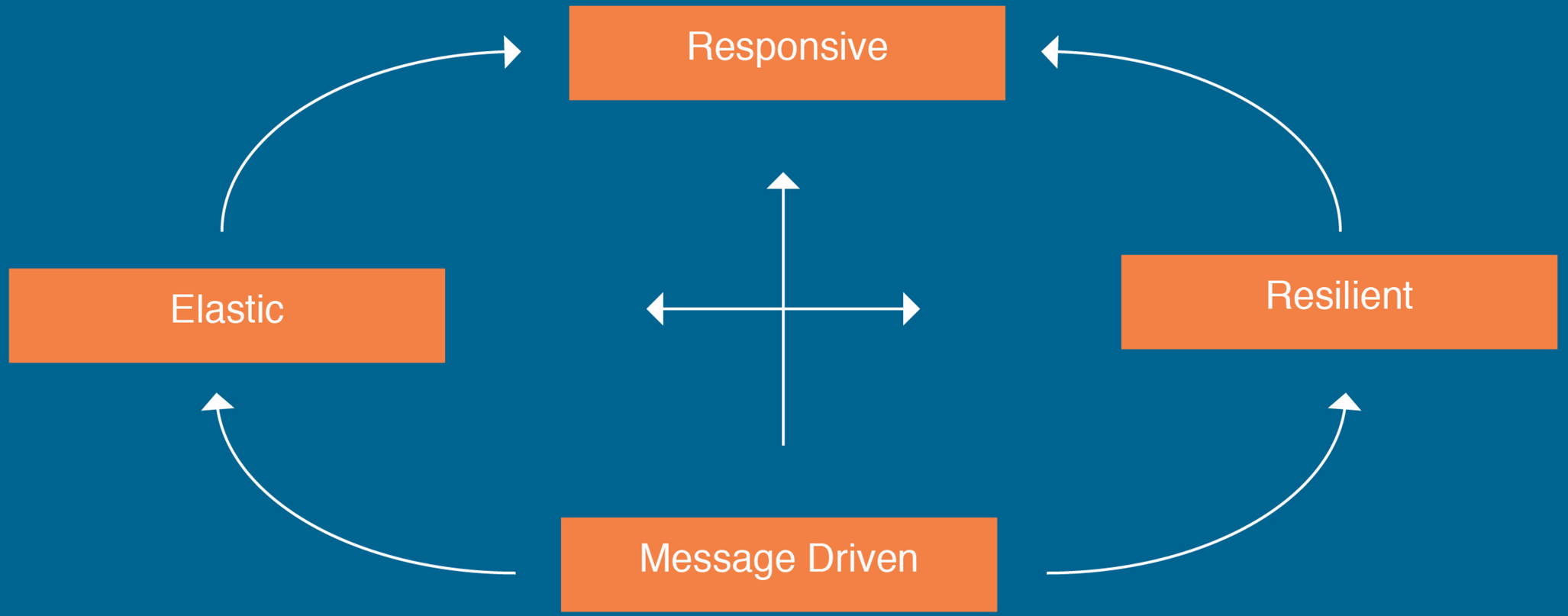
github.com/erwindeg

edegier.nl

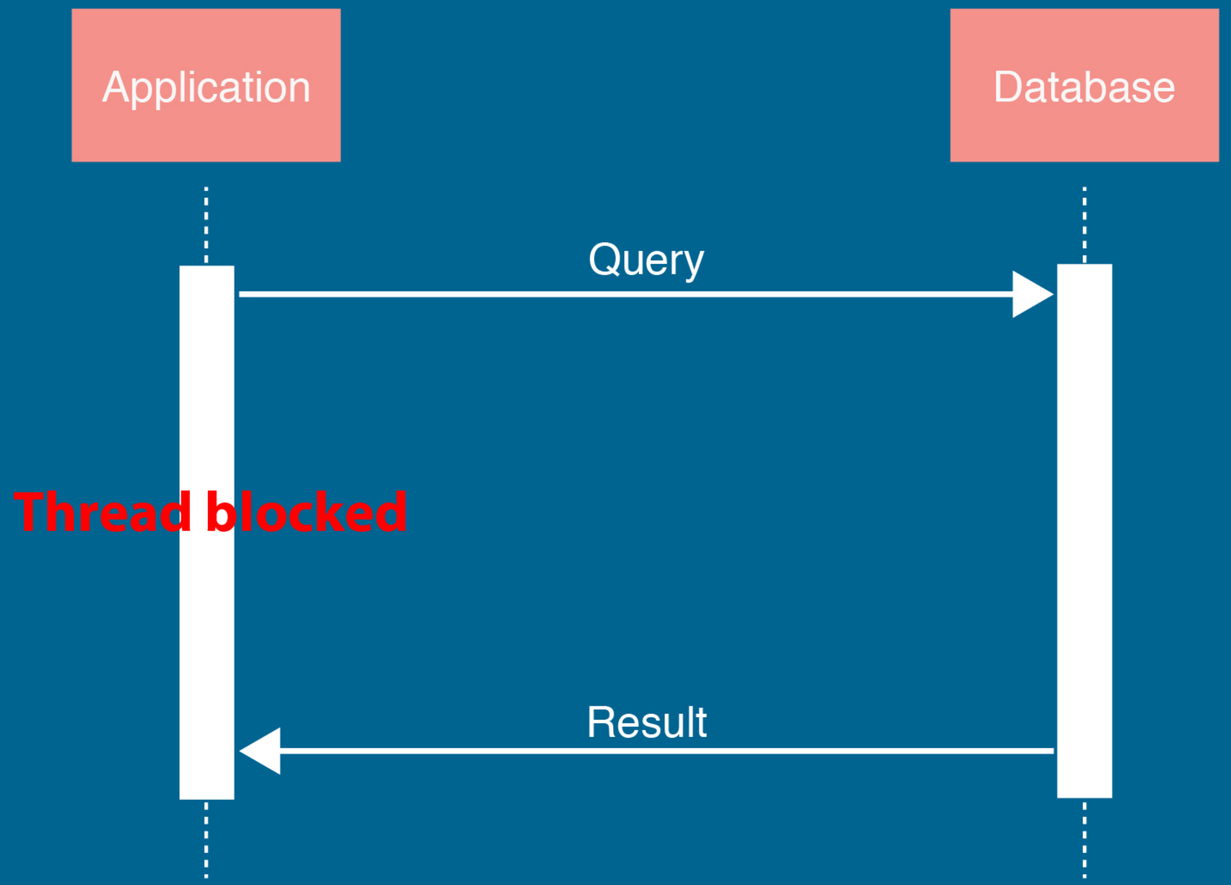
TRIFORK.
...think software

Why Reactive?

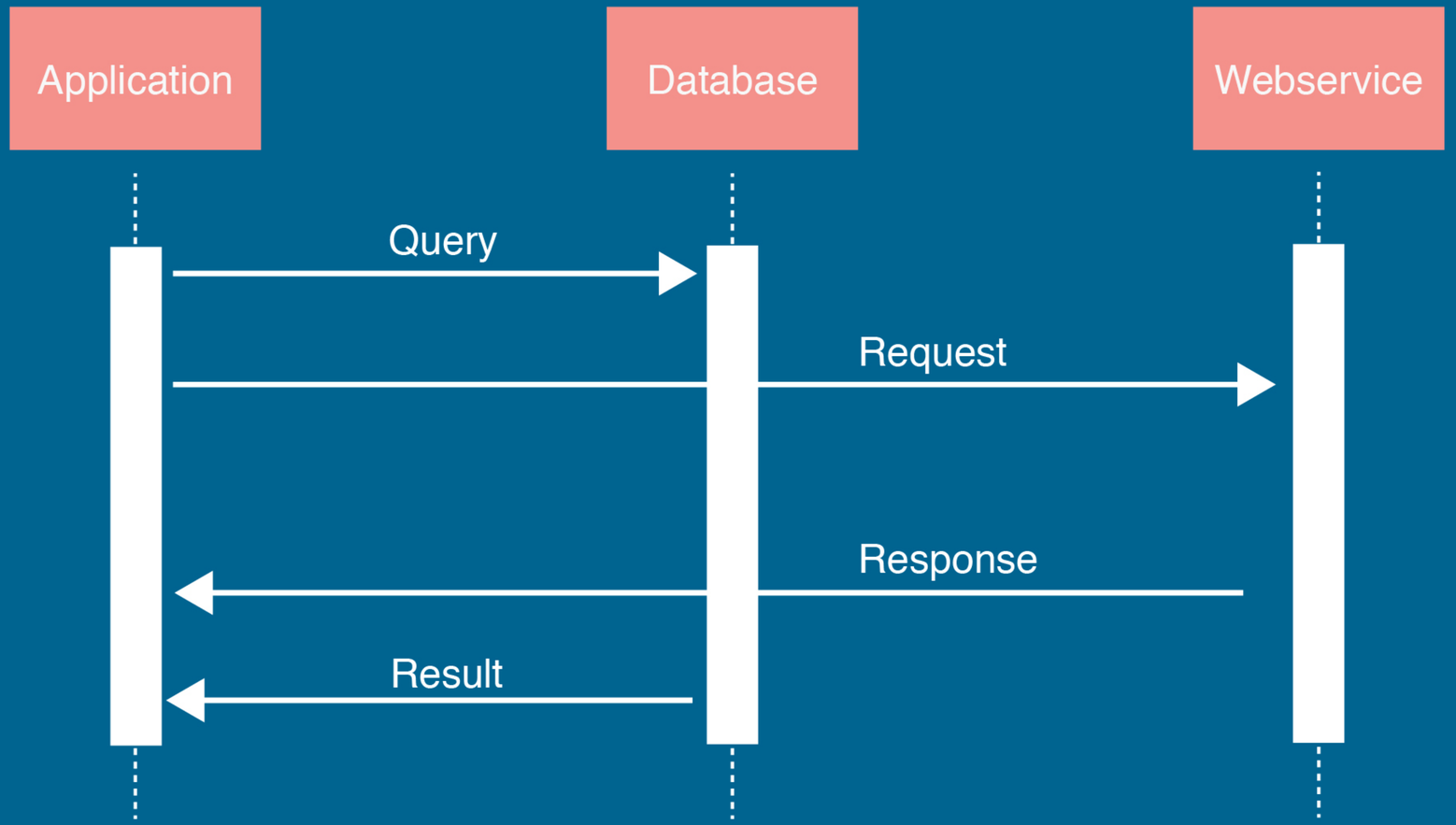



Responsive
Elastic
Resilient
Message Driven

Synchronous



Asynchronous

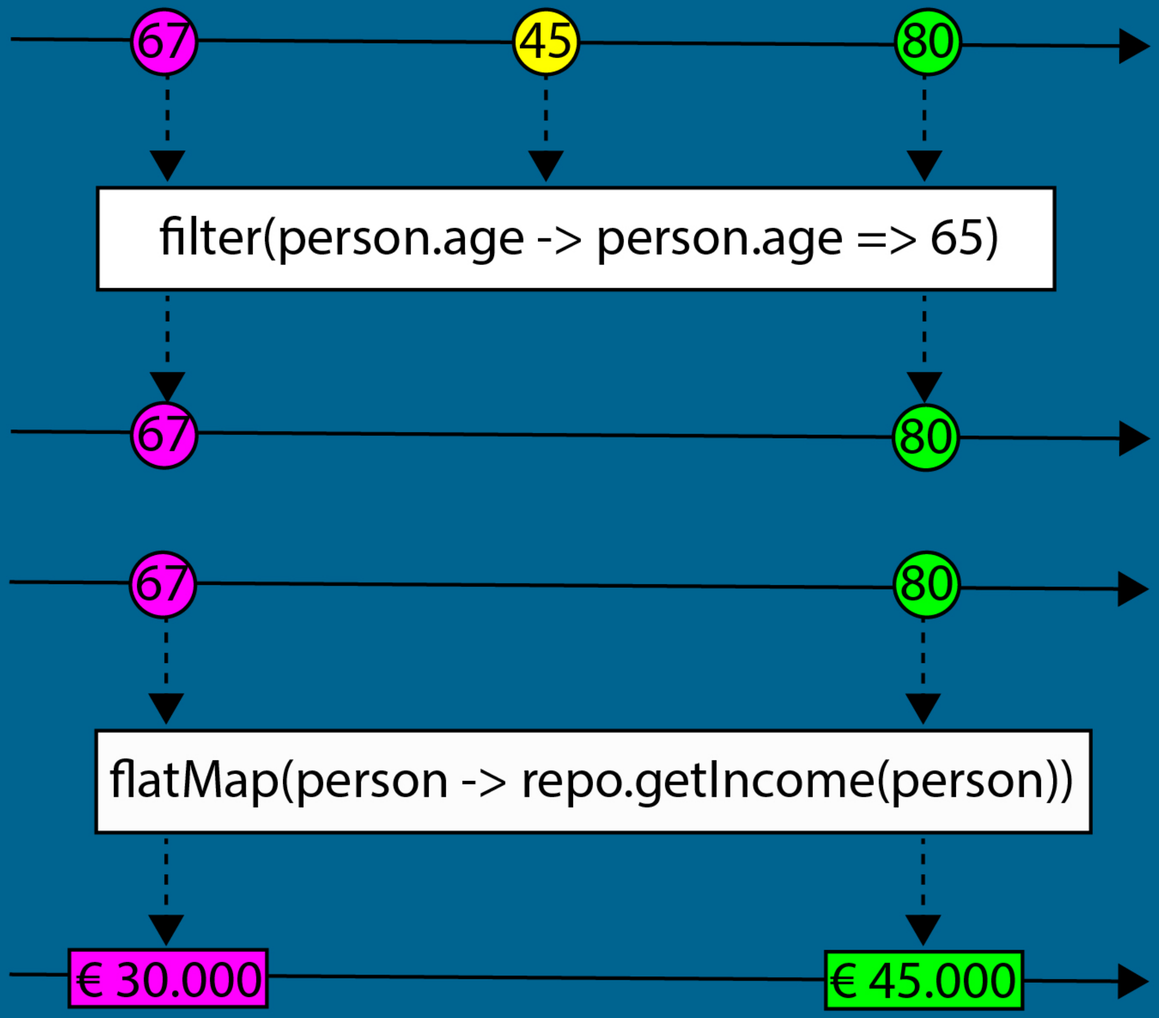


```java
//PersonRepository sync
public List<Person> findByName(String name);

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));
                       });
        }
);
```

```java
//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


67
45
80
filter(person.age -> person.age => 65)
67
80
67
80
flatMap(person -> repo.getIncome(person))
€ 30.000
€ 45.000

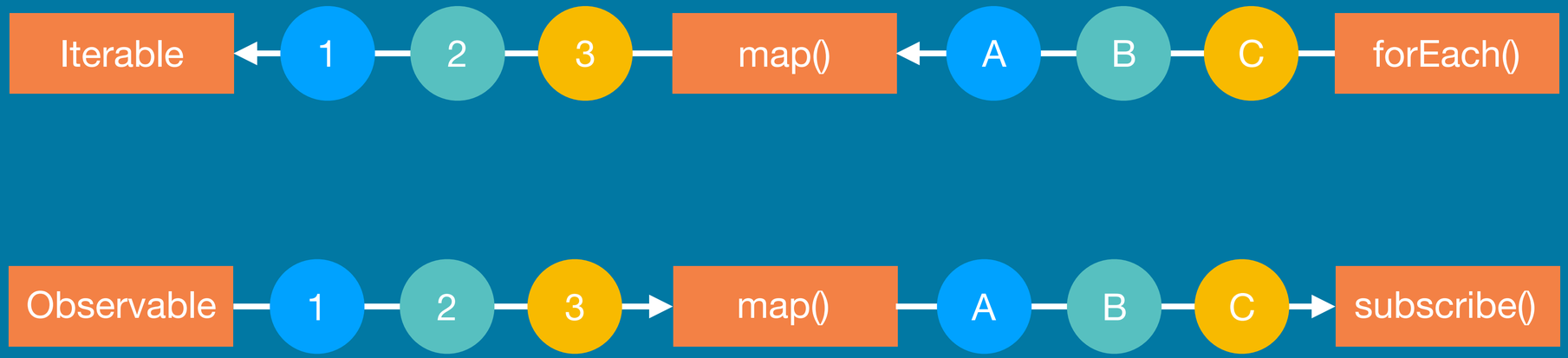
Java 8 Streams vs. RX Observables

Pull vs. Push

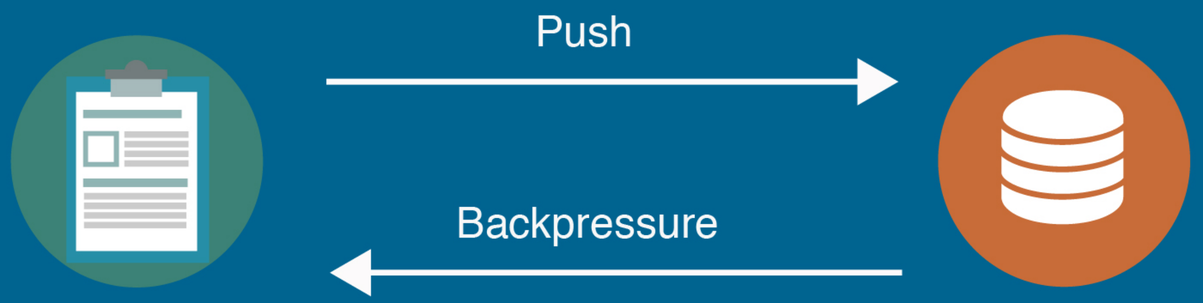
Finite vs. Infinite

Sync vs. Async

Java 8 Streams vs. RX Observables

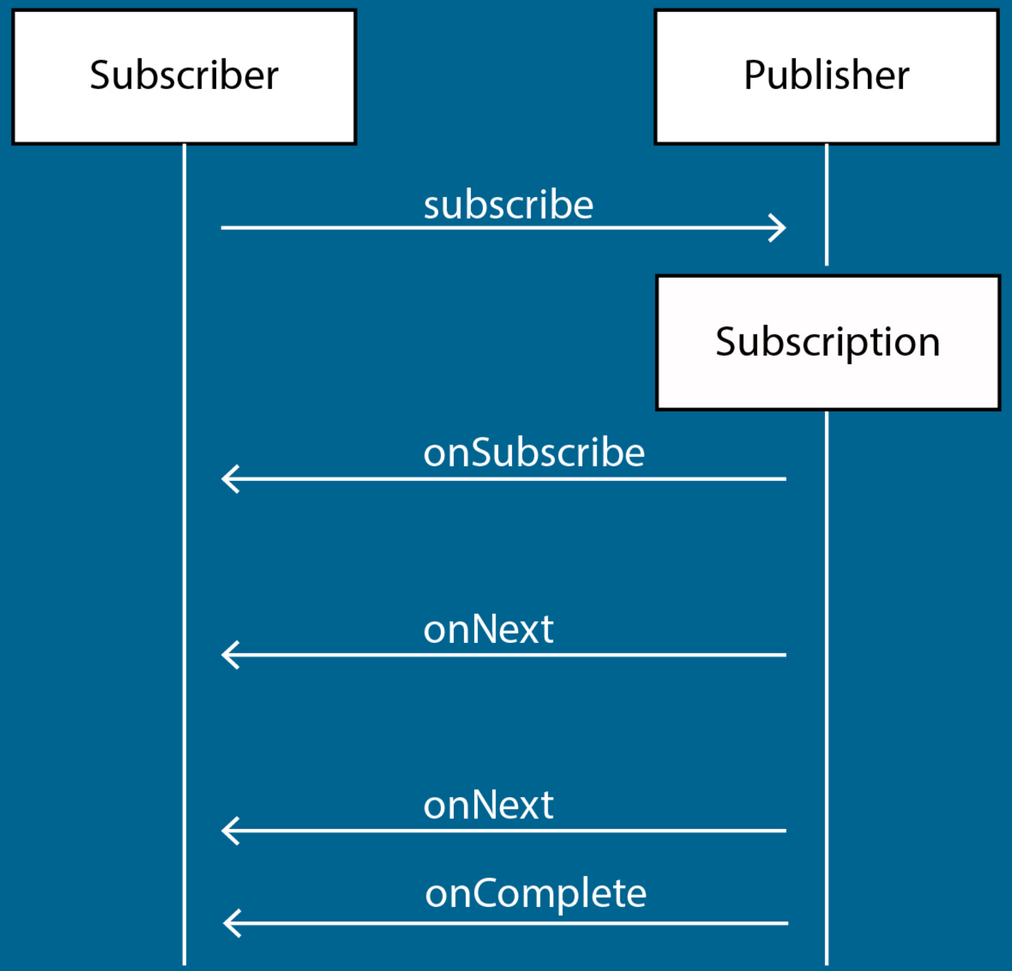


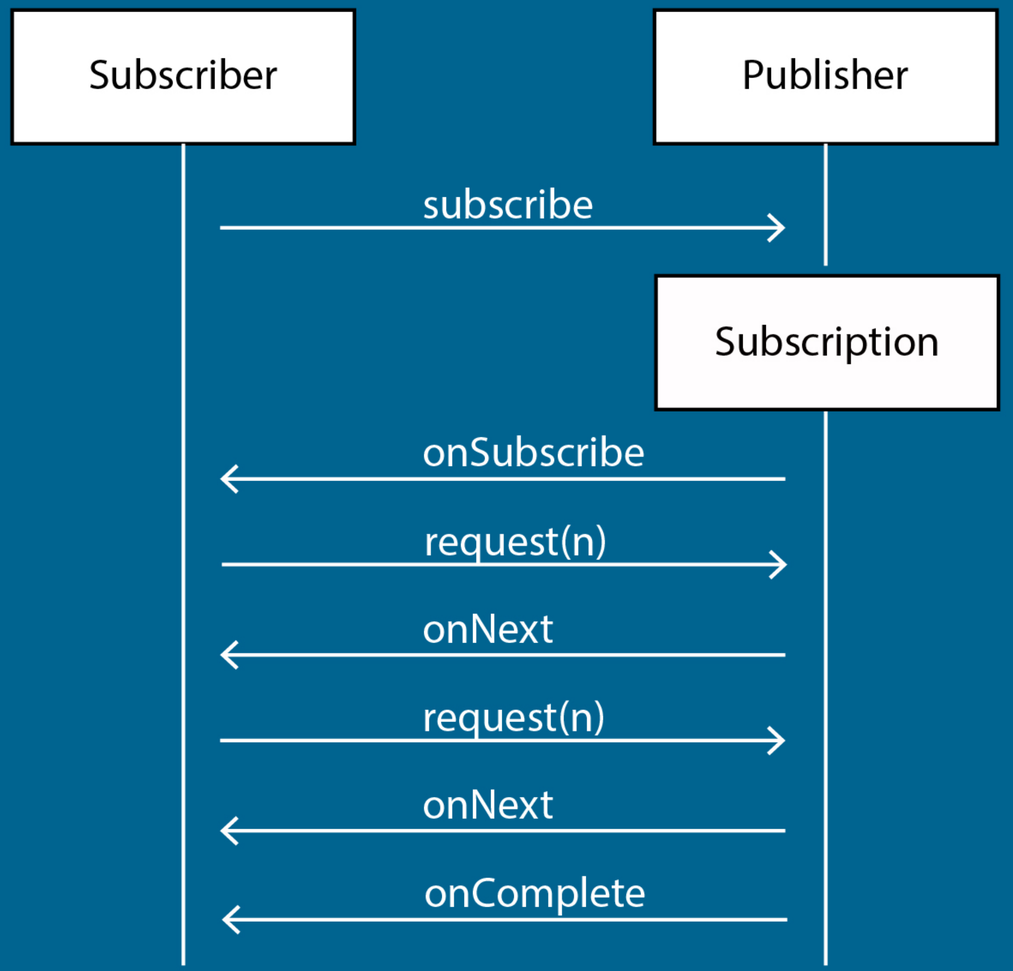
RxJava 2



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

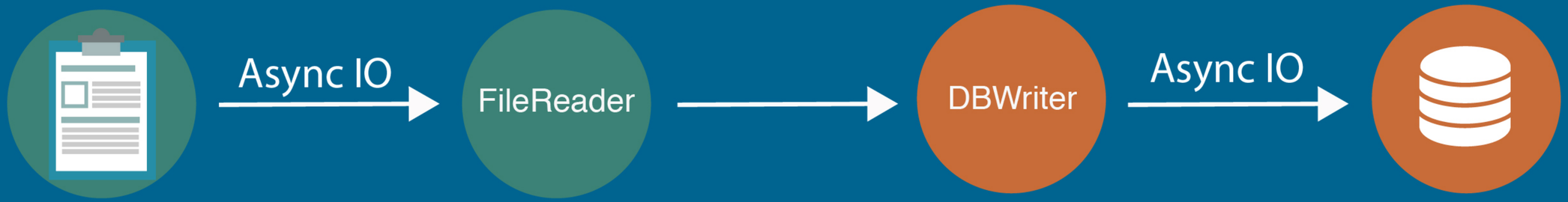
	No Value	Single Value	Multiple Values
Java Blocking	void	T	Iterable<T>
Java Non-blocking	CompletableFuture<Void>	CompletableFuture<T>	CompletableFuture<List<T>>
Reactive Streams	Publisher<Void>	Publisher<T>	Publisher<T>
RxJava	Observable<Void>	Single<T>	Observable<T>
Project Reactor	Mono<Void>	Mono<T>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

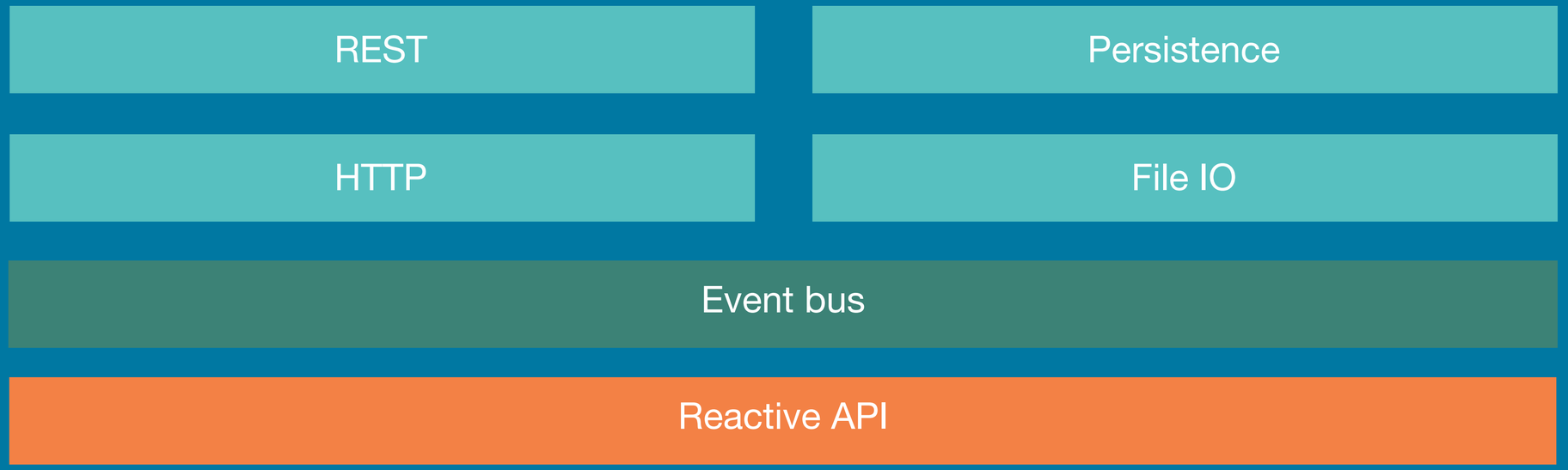
```
Stream<Integer> j = Arrays.asList(1, 2, 3, 4, 5).stream();

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

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

```
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);
```

Async operations



Reactive Stack



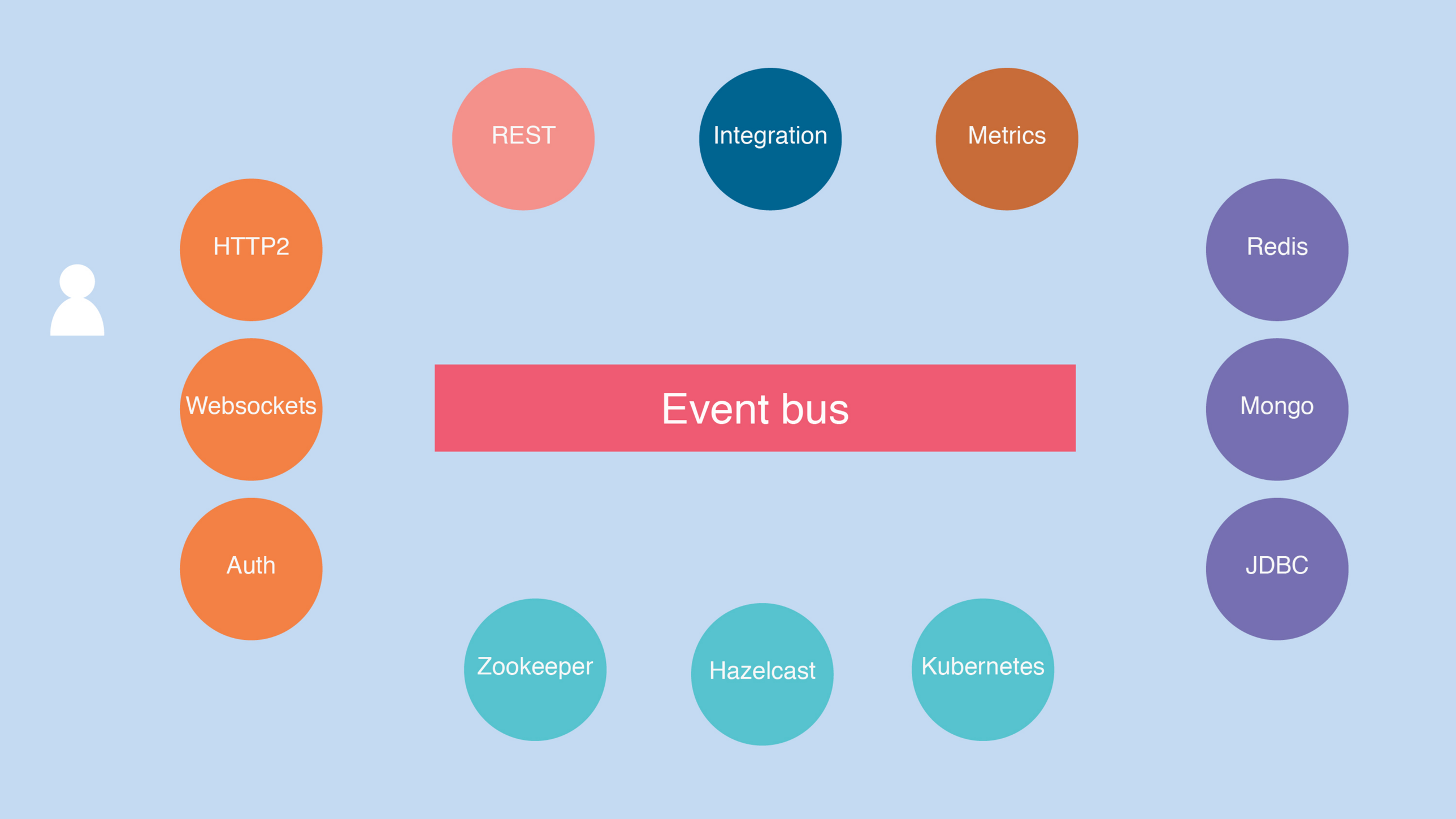
Reactive Frameworks

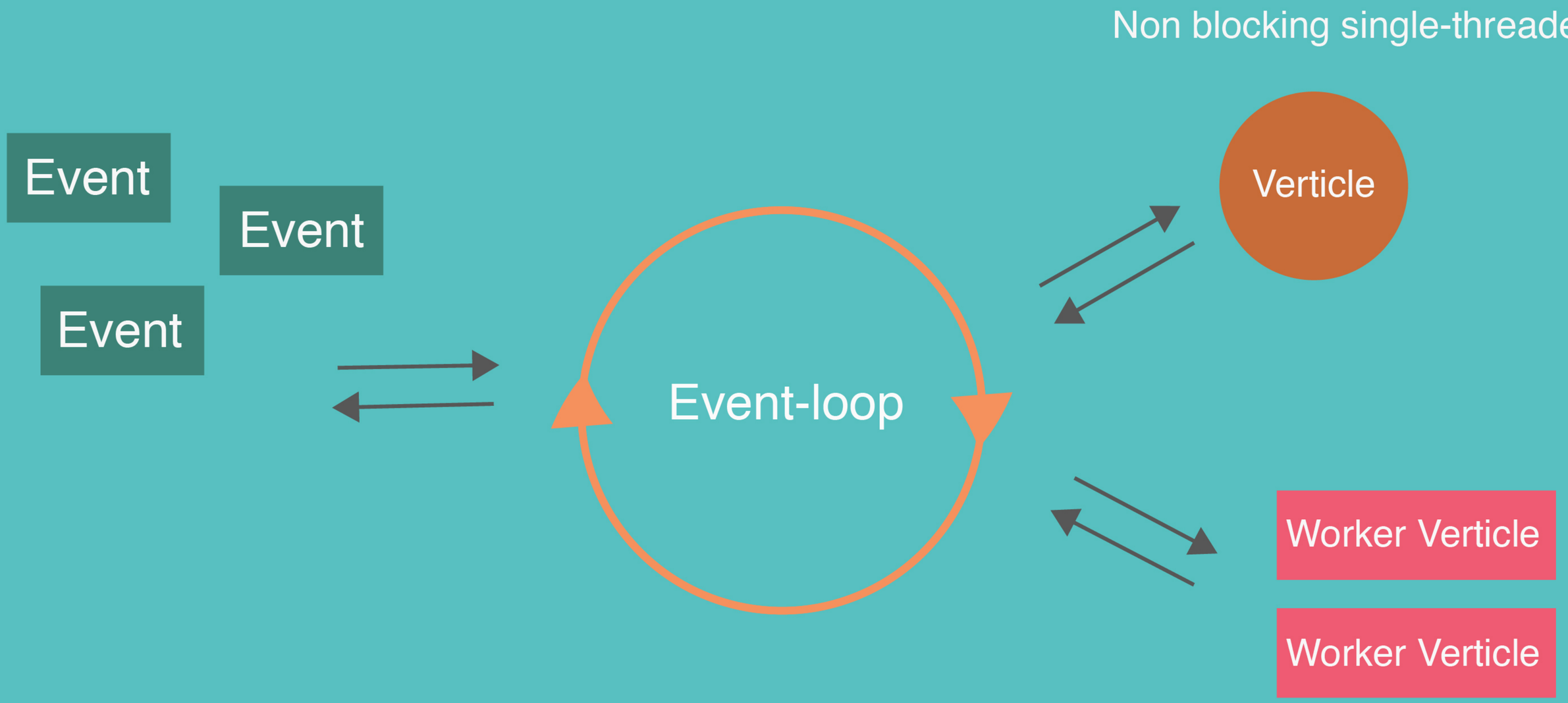
- Vert.x
- Spring 5
- Akka

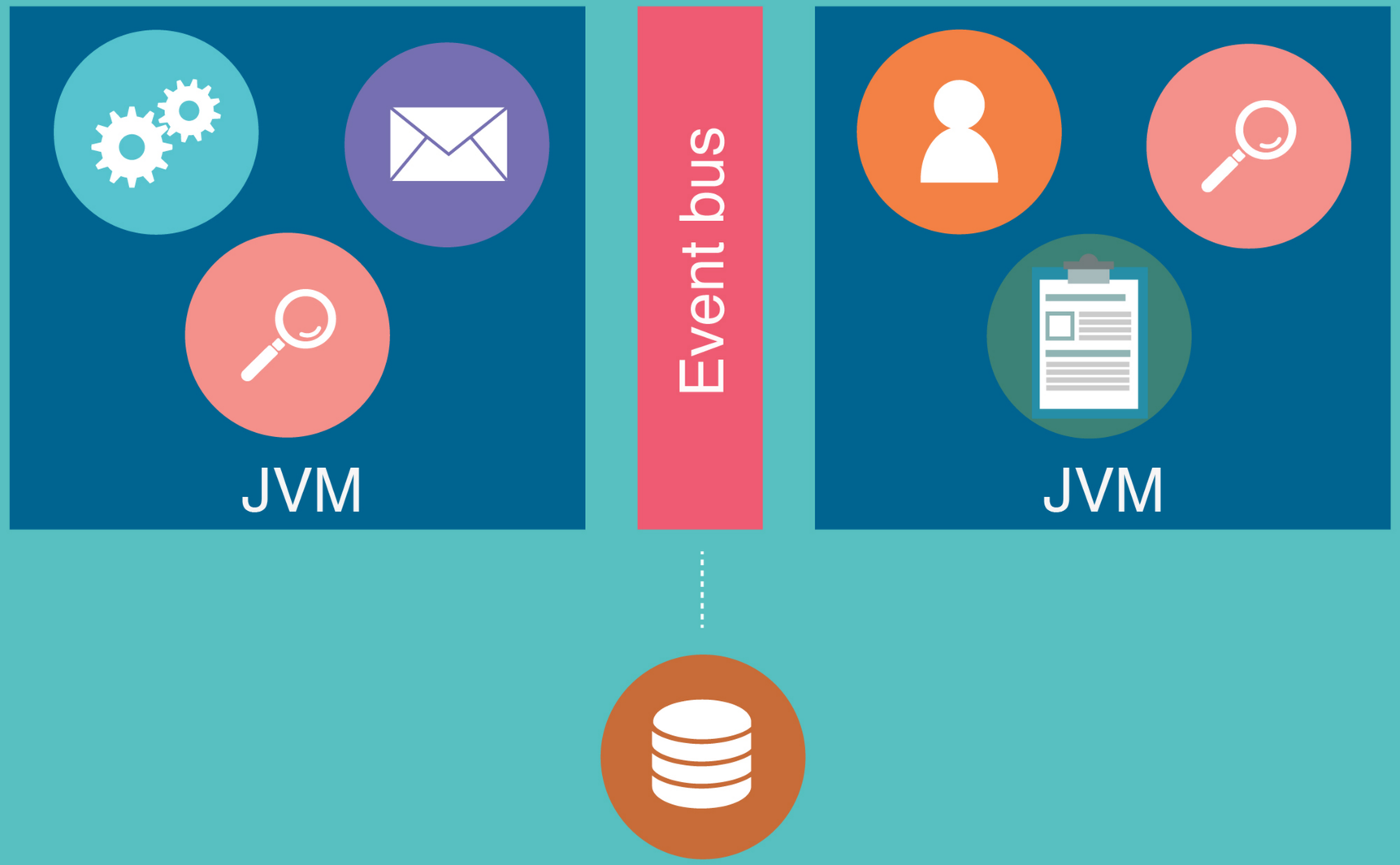
Reactive Frameworks

vs.

- Runnable Jar
- Reactive
- Polyglot
- Distributed


REST
Integration
Metrics
HTTP2
Redis
Websockets
Event bus
Mongo
Auth
JDBC
Zookeeper
Hazelcast
Kubernetes


Non blocking single-threaded
Event
Event
Event
Event-loop
Verticle
Worker Verticle
Worker Verticle
Blocking multi-threaded


JVM
Event bus
JVM

```java
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!");
  }
}
```

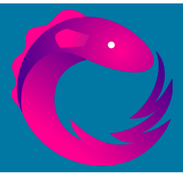
```java
public class HelloWorldRestVerticle extends AbstractVerticle{

  @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

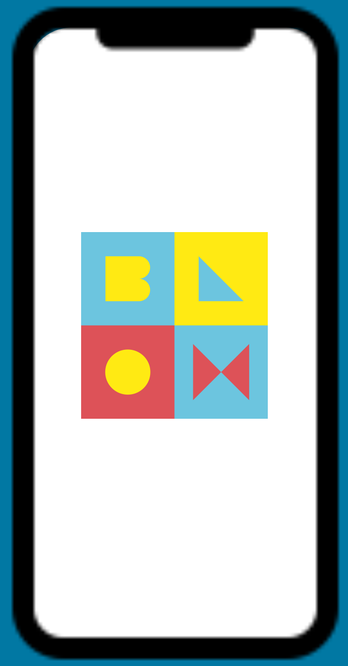
- Spring Webflux
- Project Reactor
- Reactive Data Repositories
- Project Reactor event bus

```java
@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));
    }
}
```

Reactive Overview

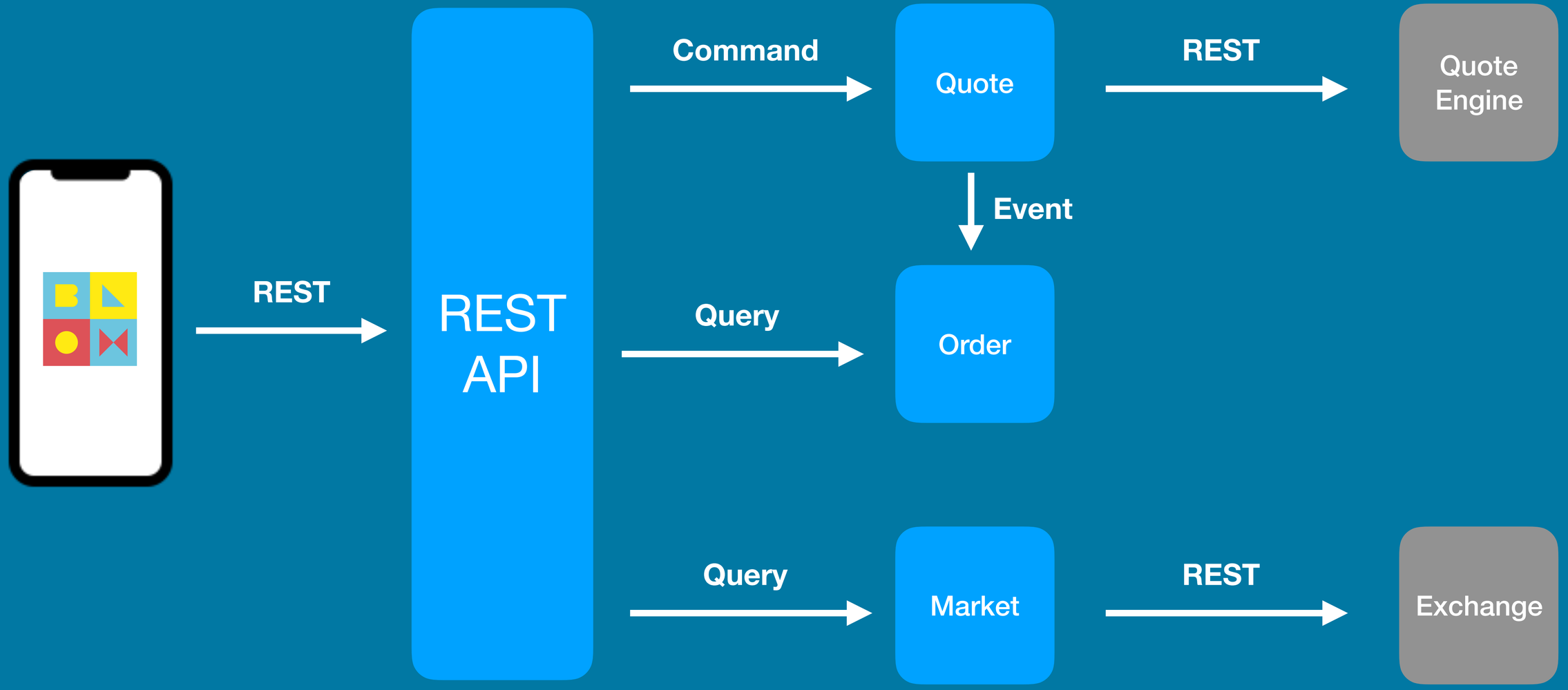
	Event Model	Annotations
Framework		
API	 RxJava	 Project Reactor

We are Blox



- Crypto positions
- Closed system
- Buy Crypto
- Sell Crypto
- Accounts

Architecture



```
@POST
public Mono<ResponseEntity<OrderResponse>> createOrder(CreateOrderRequest request){
  CreateOrderCommand command = CreateOrderCommand.fromRequest(request); //1
  return this.commandGateway.send(command) //2
         .flatMap(id -> queryGateway.send(new FindOrderSummaryQuery( id))) //3
         .retryWhen(errors -> errors.delayElements(Duration.of(100, MILLIS)) //4
         .take(10)).concatWith(Mono.error(new RuntimeException())).next() //5
         .onErrorReturn(new OrderResponse(orderID,OrderStatus.CREATED)) //6
         .map(orderResponse -> ResponseEntity.ok().body(orderResponse)); //7
}
```

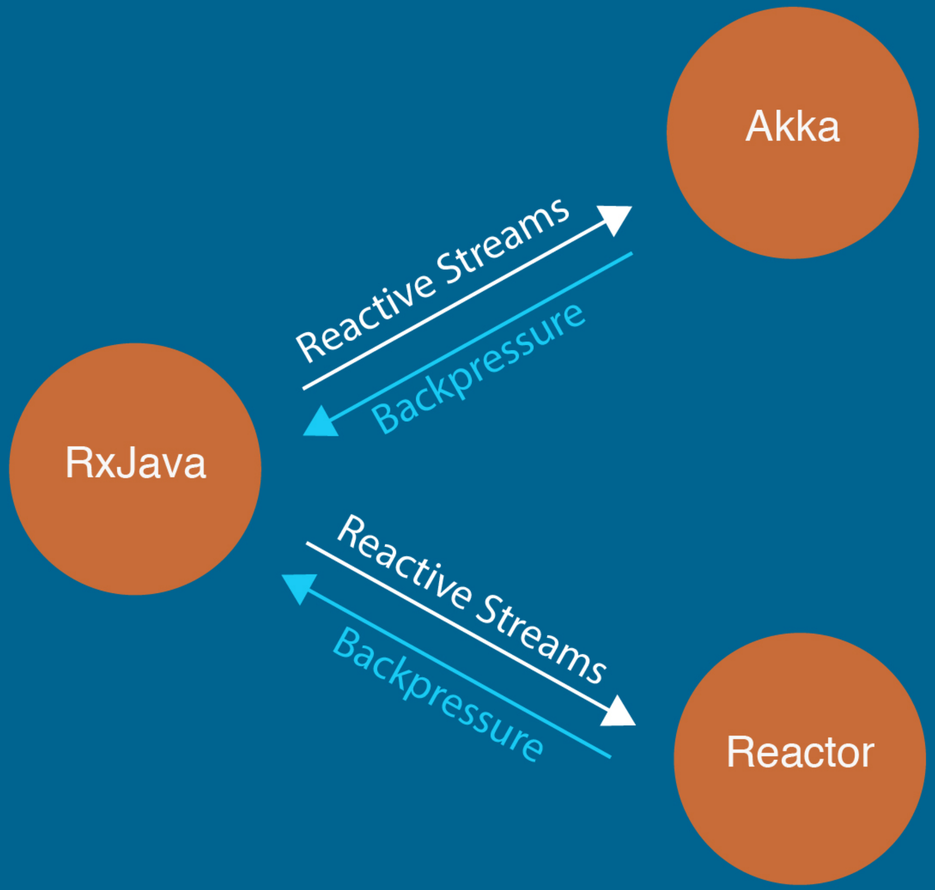
Spring 5 Webflux

- Testing
- Spring Security
- Spring Cloud Hystrix
- Axon API’s: toFuture & fromFuture

WebFlux Testing

- @WebFluxTest(controllers = …)
- WebTestClient for WebFlux controller tests
- @MockBean for injected services

Reactive Streams



```java
@Override
protected Observable<ClientResponse> construct() {

    Mono offer = this.webClient
        .method(this.request.method())
        .uri(this.request.url())
        .body(request.body())
        .exchange()
        .timeout(Duration.ofMillis(getProperties().executionTimeoutInMilliseconds().get()));

    return toSingle(offer).toObservable();
}

protected Mono<ClientResponse> exchange(ExchangeCommand command) {
    return Mono.from(toPublisher(command.toObservable().toSingle()));
}
```

Conclusion: state of reactive Java

- Libraries for reactive programming: Rx / Reactor
- Frameworks for async processes: Spring / Vert.x
- Expect to DIY
- Workarounds