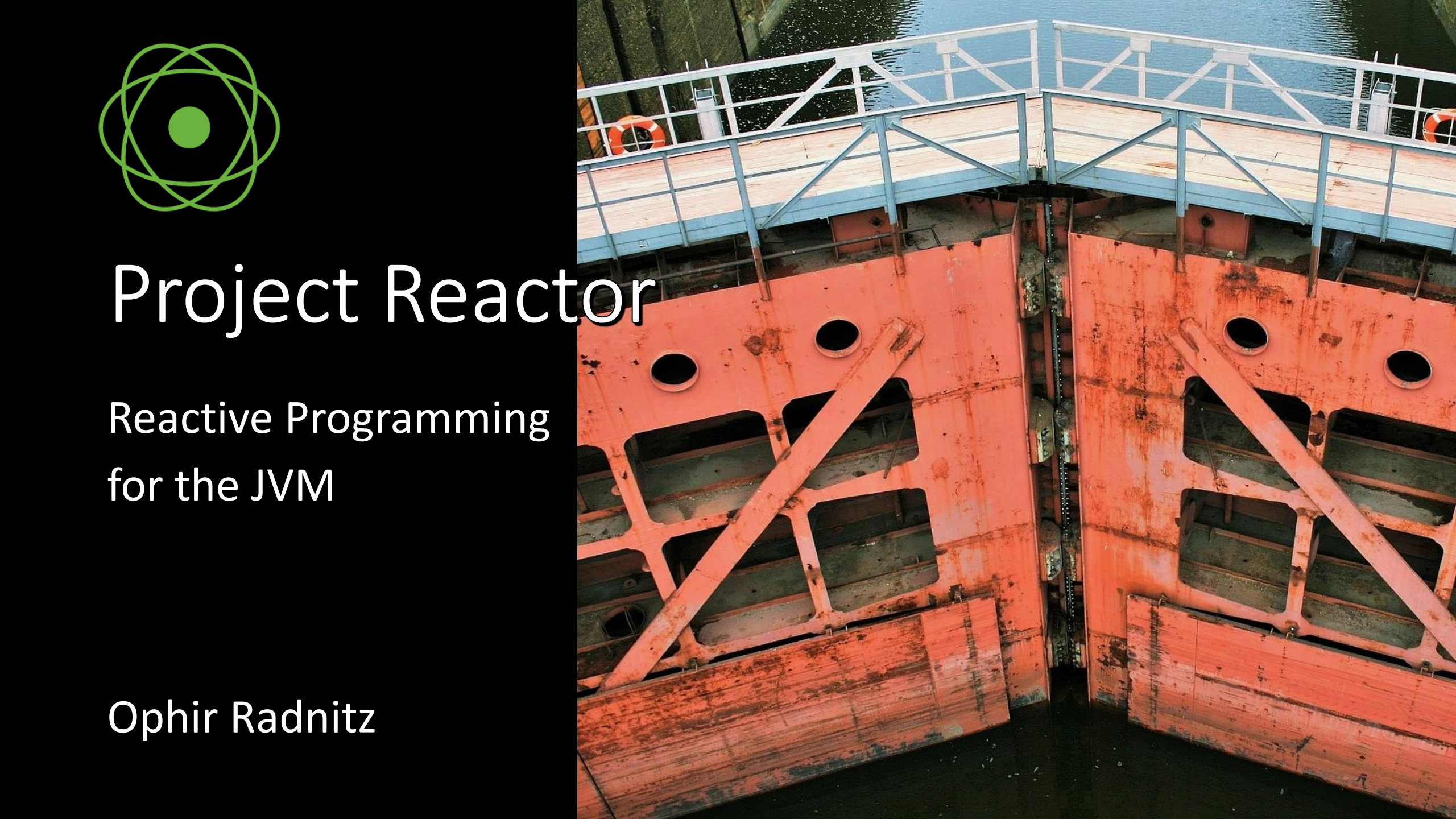


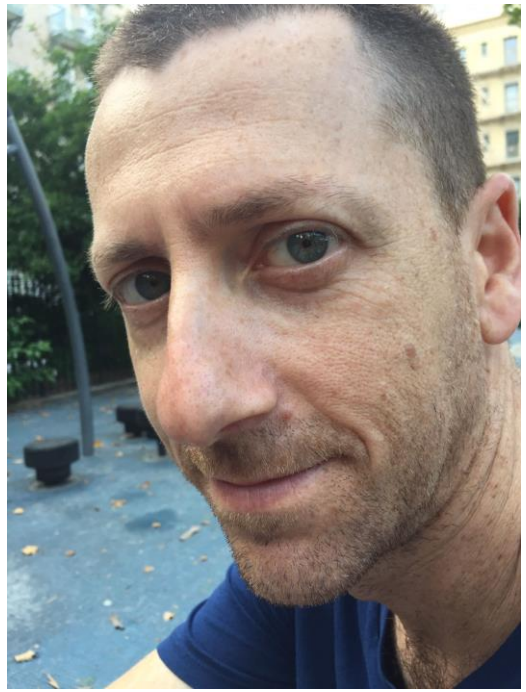
Project Reactor

Reactive Programming
for the JVM

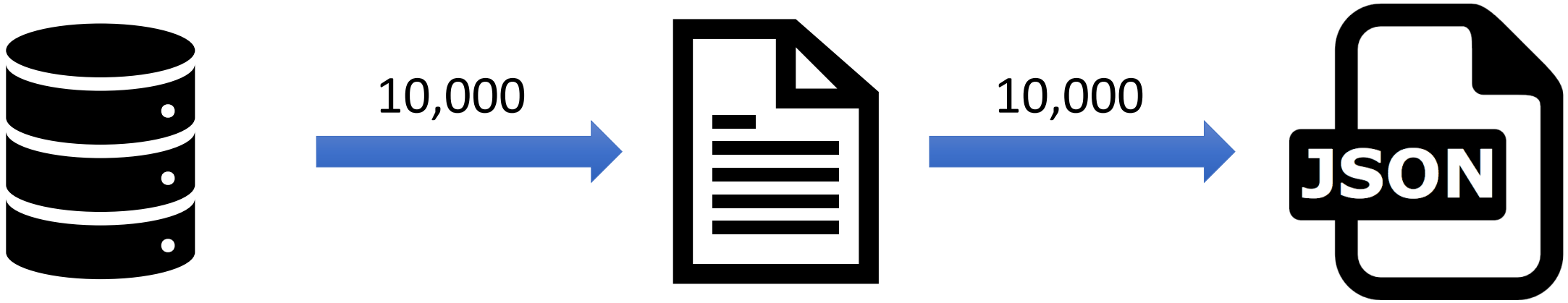
Ophir Radnitz

Who Am I

- Chief Software Architect at tufin
- Coding > 15 years



Common Use Case

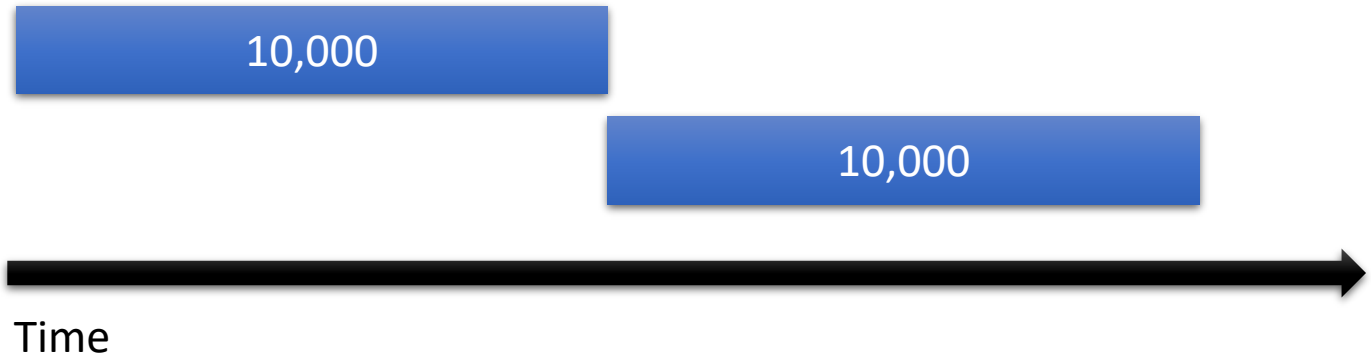


Common Code

```
@RequestMapping(path = "/{userId}", method = RequestMethod.GET)
public List<ReservationDTO> getReservations(@PathVariable("userId") String userId) {

    List<ReservationDTO> list = new ArrayList<>();
    for (Reservation reservation : repository.fetchReservations(userId)) {
        if (isRecent(reservation, 1, HOURS)) {
            ReservationDTO reservationDTO = toReservationDTO(reservation);
            list.add(reservationDTO);
        }
    }
    return list;
}
```

Common Code, stream-like

```
@RequestMapping(path = "/{userId}", method = RequestMethod.GET)
public List<ReservationDTO> getReservations(@PathVariable("userId") String userId) {

    return repository.fetchReservations(userId).stream()
            .filter(reservation -> isRecent(reservation, 1, HOURS))
            .map(this::toReservationDTO)
            .collect(Collectors.toList());
}
```



Going Reactive!

```
@GetMapping("/{userId}")
public Flux<ReservationDTO> getReservations(@PathVariable("userId") String userId) {

    return repository.fetchReservations(userId)
            .filter(reservation -> isRecent(reservation, 1, HOURS))
            .map(this::toReservationDTO);
}
```

Agenda

1

Introduction to Reactive Programming

2

Project Reactor drilldown

- Backpressure
- Error handling
- Debugging

3

Code Examples

- Web
- DB

Agenda

1

Introduction to Reactive Programming

2

Project Reactor drilldown

- Backpressure
- Error handling
- Debugging

3

Code Examples

- Web
- DB

Agenda

1

Introduction to Reactive Programming

2

Project Reactor drilldown

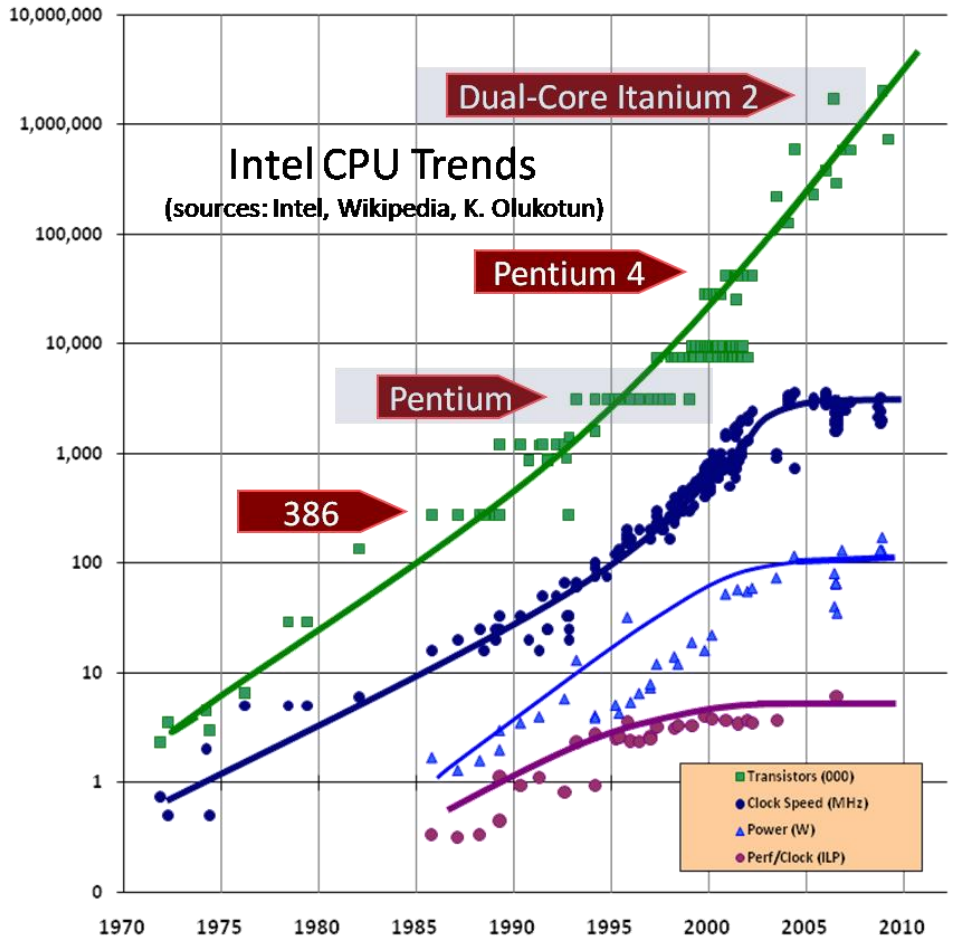
- Backpressure
- Error handling
- Debugging

3

Code Examples

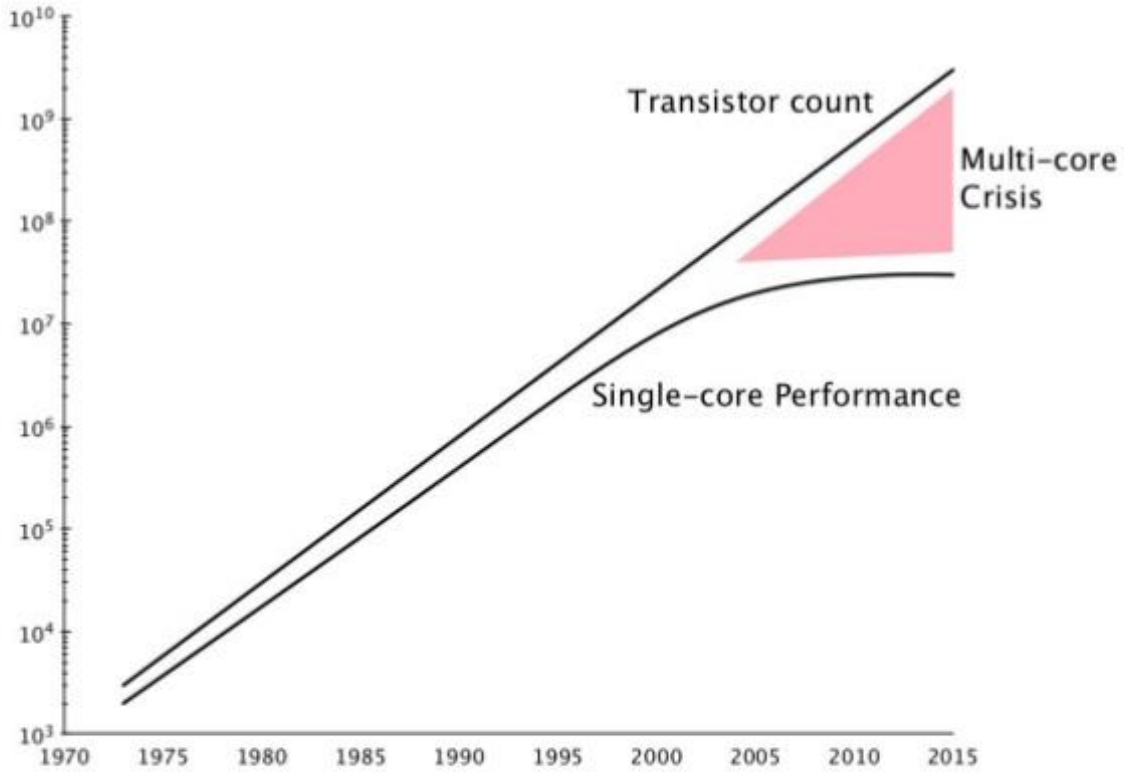
- Web
- DB

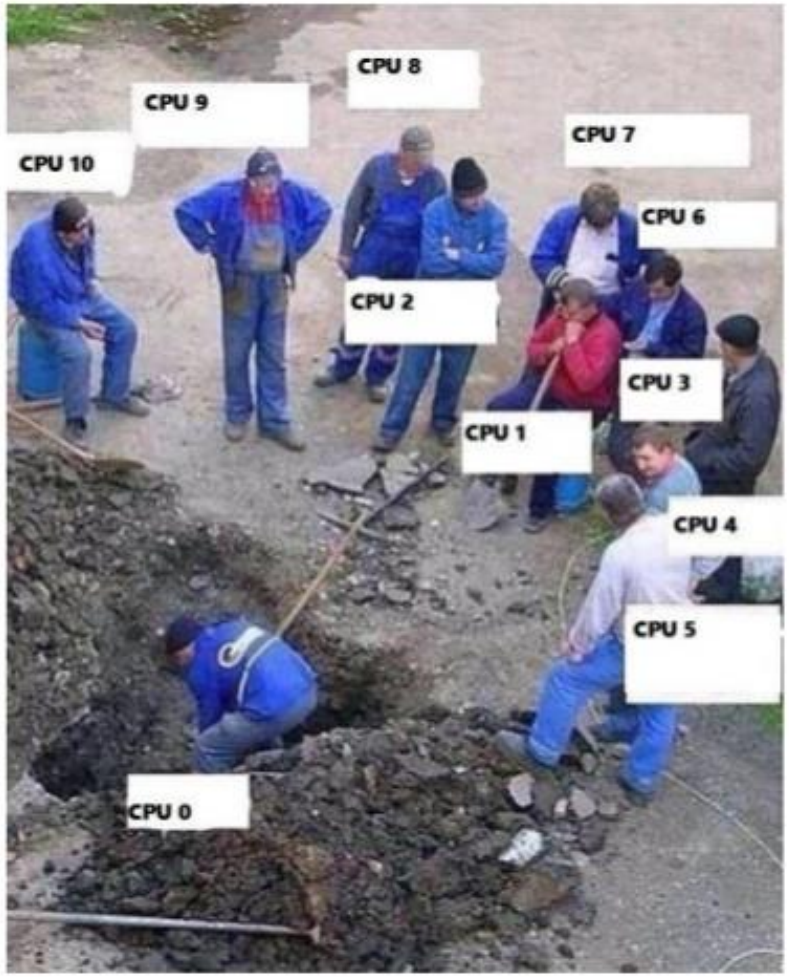
The Free Lunch Is Over



http://www.gotw.ca/publications/concurrency-ddj.htm

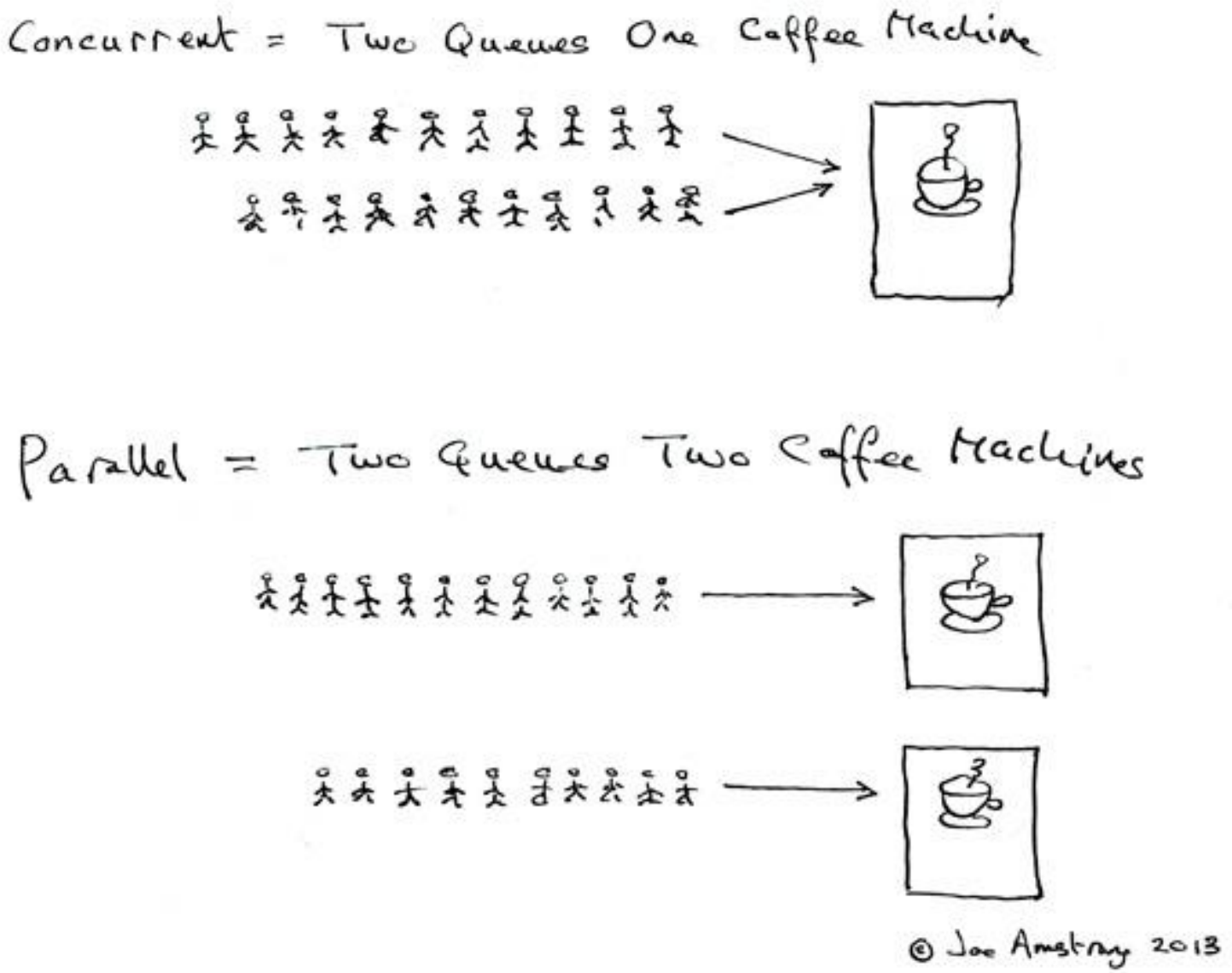
The Multi-core Crisis

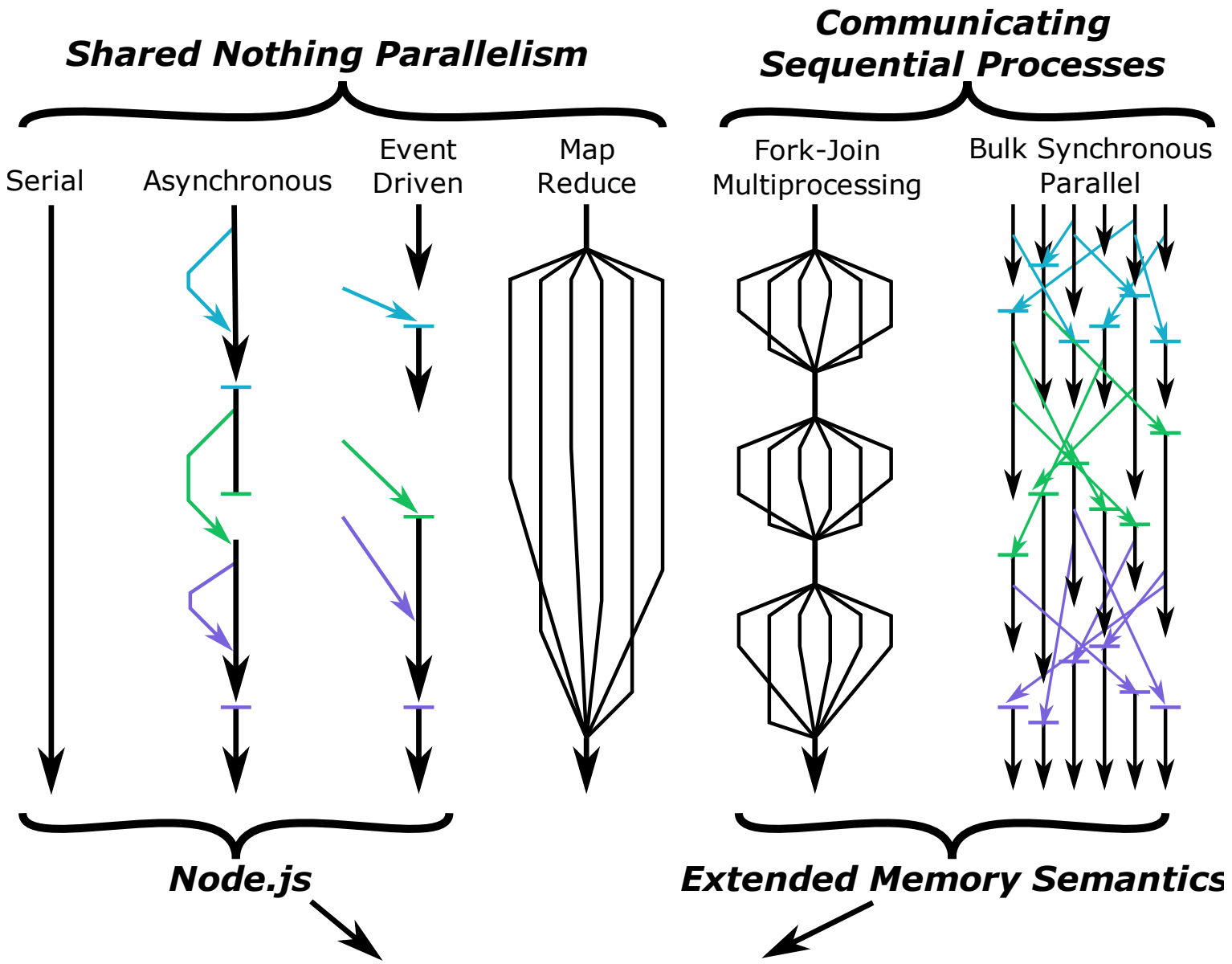




https://www.slideshare.net/mariofusco/comparing-different-concurrency-models-on-the-jvm

Concurrency / Parallelism



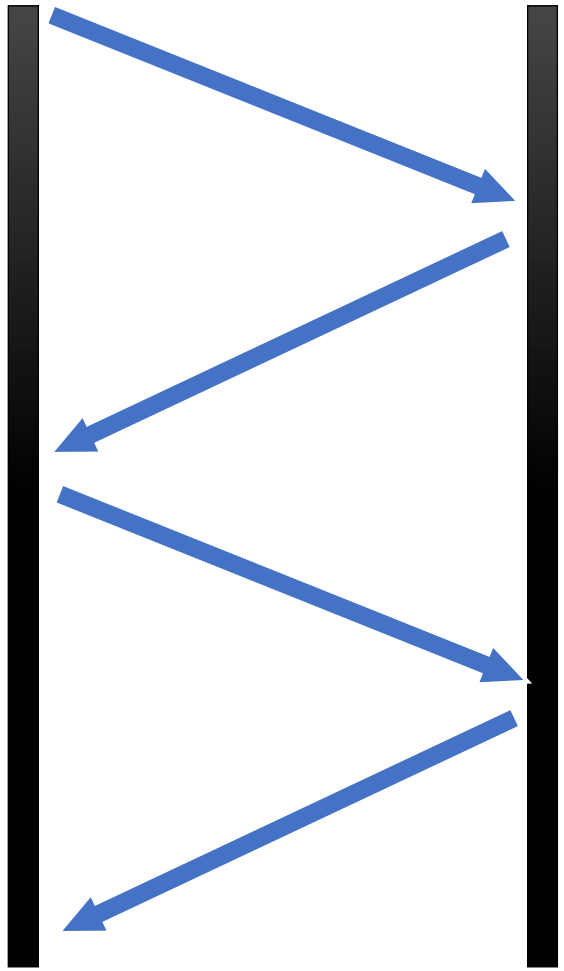


https://github.com/syntheticsemantics/ems

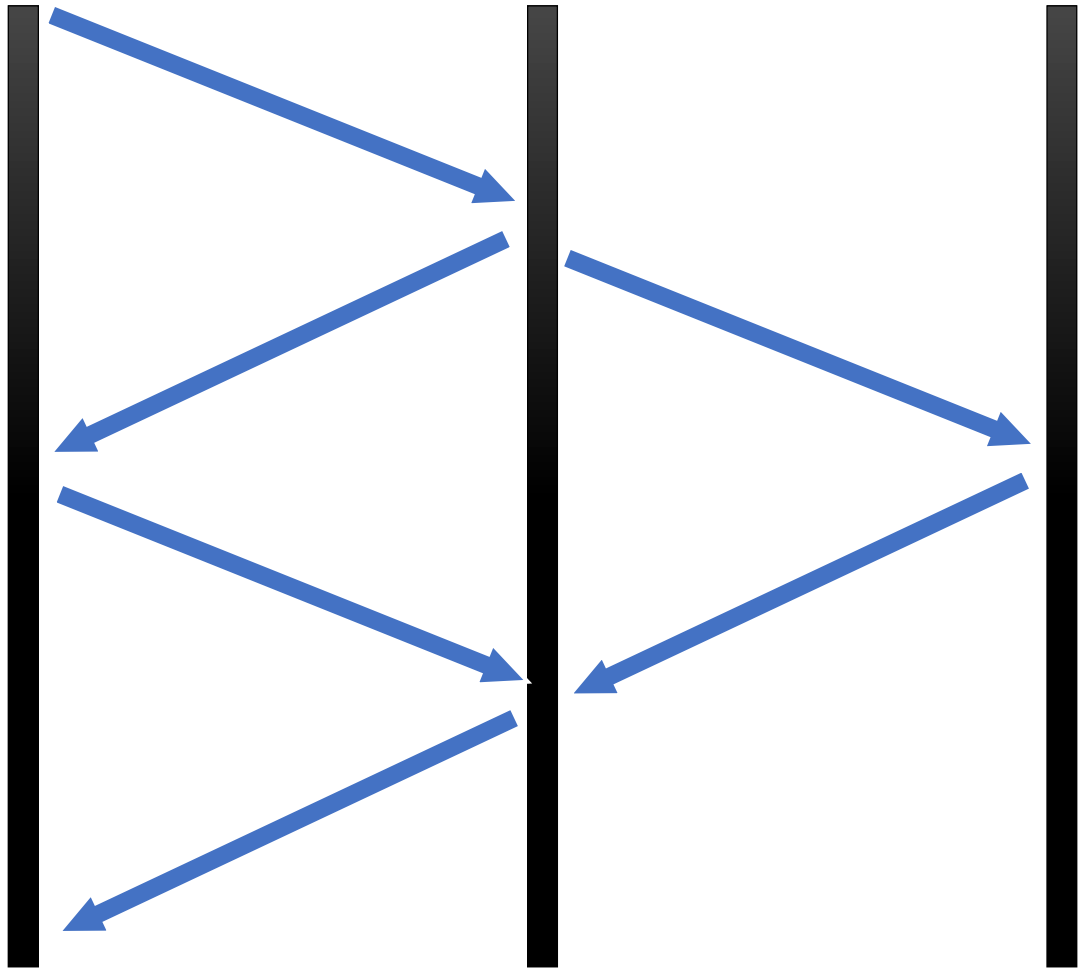
Many Concurrency Models for the JVM

- Threads, locks
- **`java.util.concurrent`**
  - ExcutorService, Future, CountdownLatch, CyclicBarrier, Workers…
- CompletableStage/CompletableFuture
- Fibers (event loop)
  - Actors
  - Vert.x Verticles (on top of Quasar)
- Kotlin Coroutines

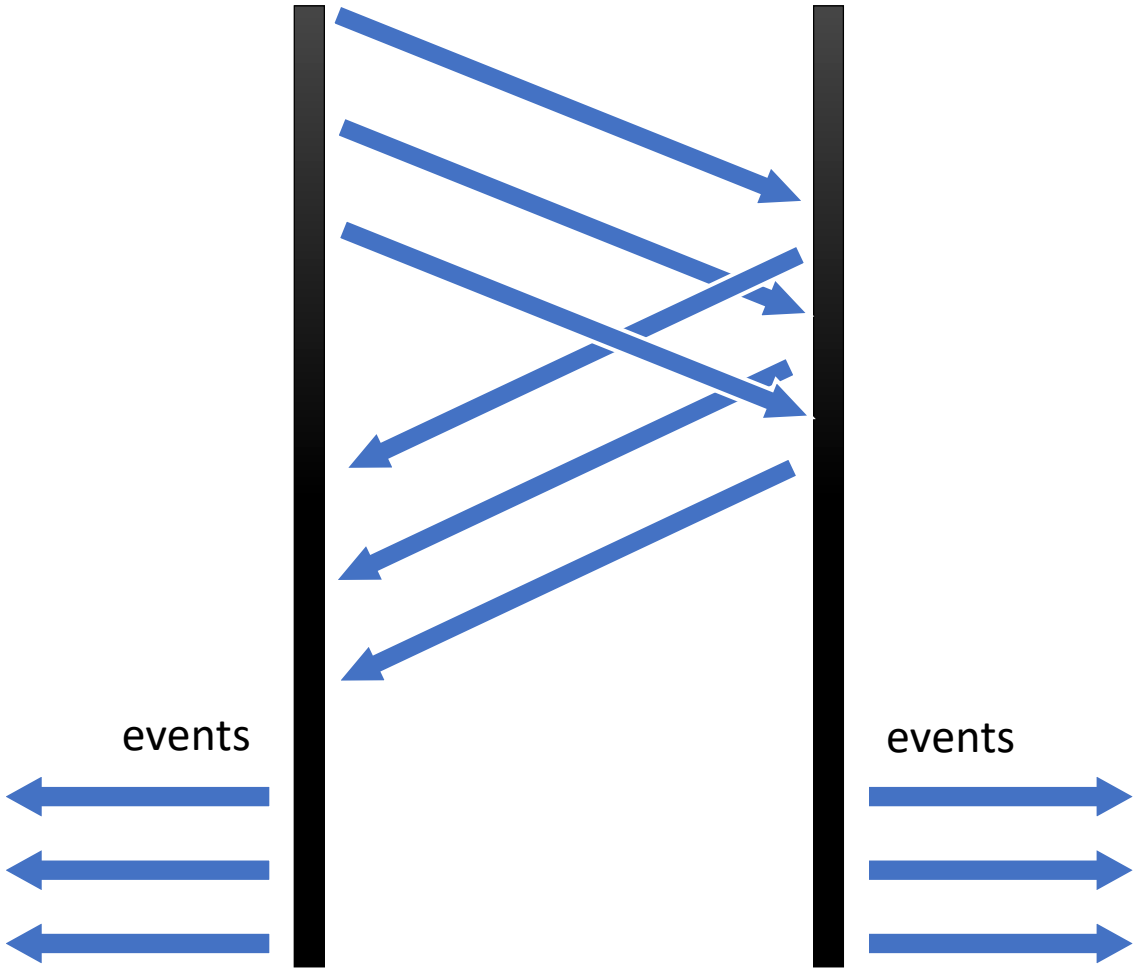
IO: Go Async



IO: Go Async



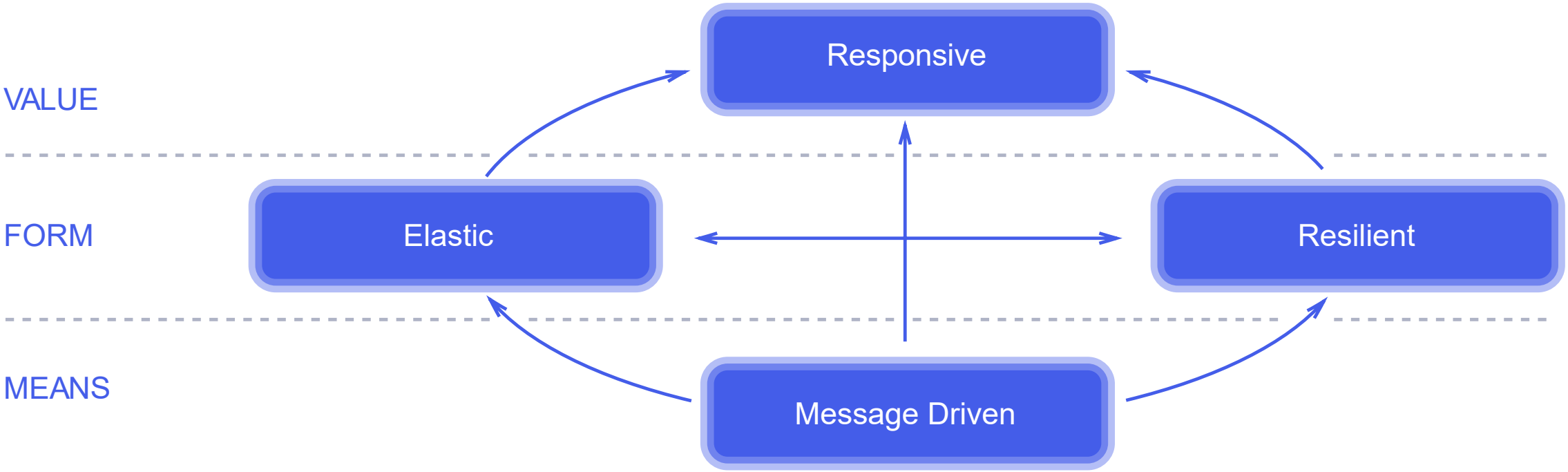
IO: Go Async



Use Cases

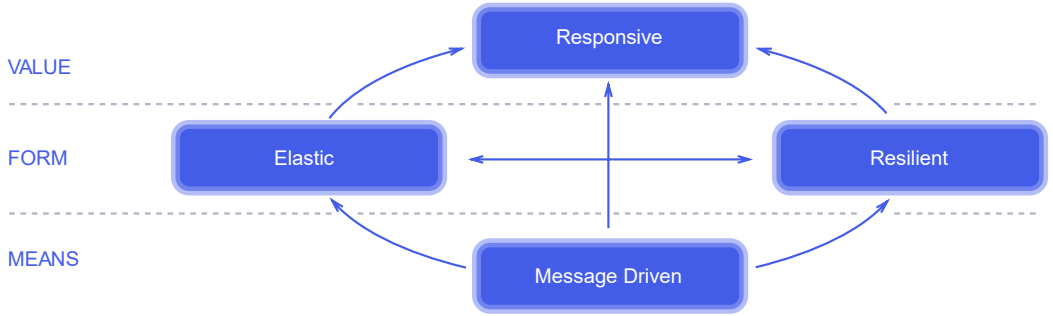
- Network access
- HTTP calls
- Message Broker
- Database access
- Mobile
- Browser events

Reactive Manifesto



https://www.reactivemanifesto.org/

Reactive Manifesto



- **Responsive**
  - The system responds in a timely manner if at all possible.
- **Resilient**
  - Stays responsive in the face of failure.
- **Elastic**
  - Stays responsive under varying workload. Can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs.
- **Event-Driven**
  - Relies on asynchronous message passing to establish a boundary between components that ensures loose coupling, isolation and location transparency.

Reactive Streams

Asynchronous programming paradigm concerned with data streams and the propagation of change.

High level way of writing **concurrent** programs

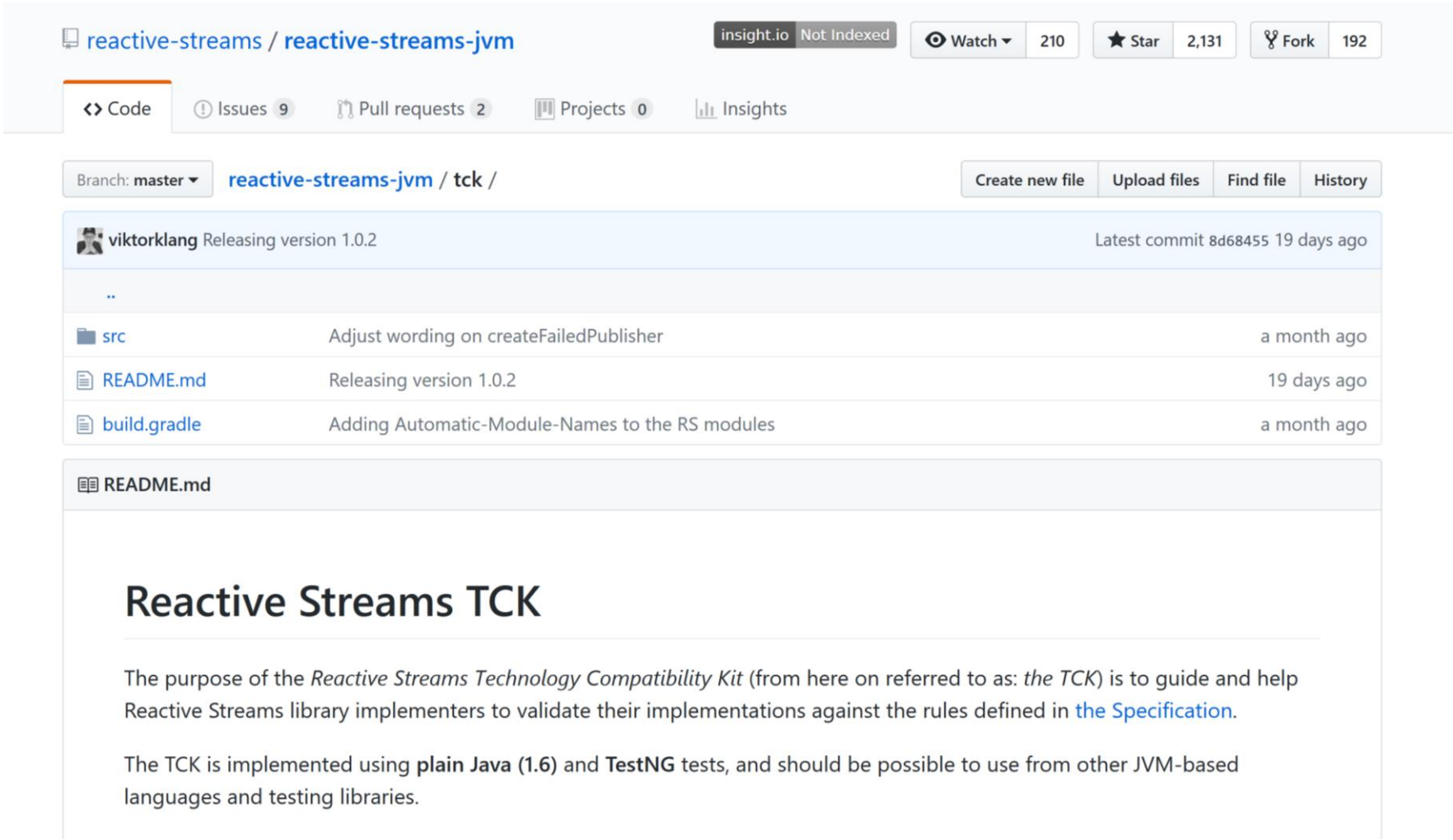
- Composable
- Flexible

Reactive Streams

- Contract for async stream processing
  - **Non-blocking**
  - **Backpressure**
- Interoperability layer
- Co-designed by Netflix, Pivotal, Lightbend, Twitter, RedHat, Kaazing

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

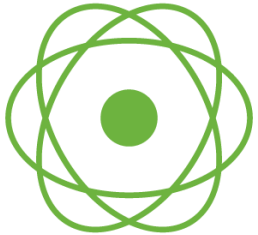
Technology Compatibility Kit



https://github.com/reactive-streams/reactive-streams-jvm/tree/master/tck

Reactive Streams

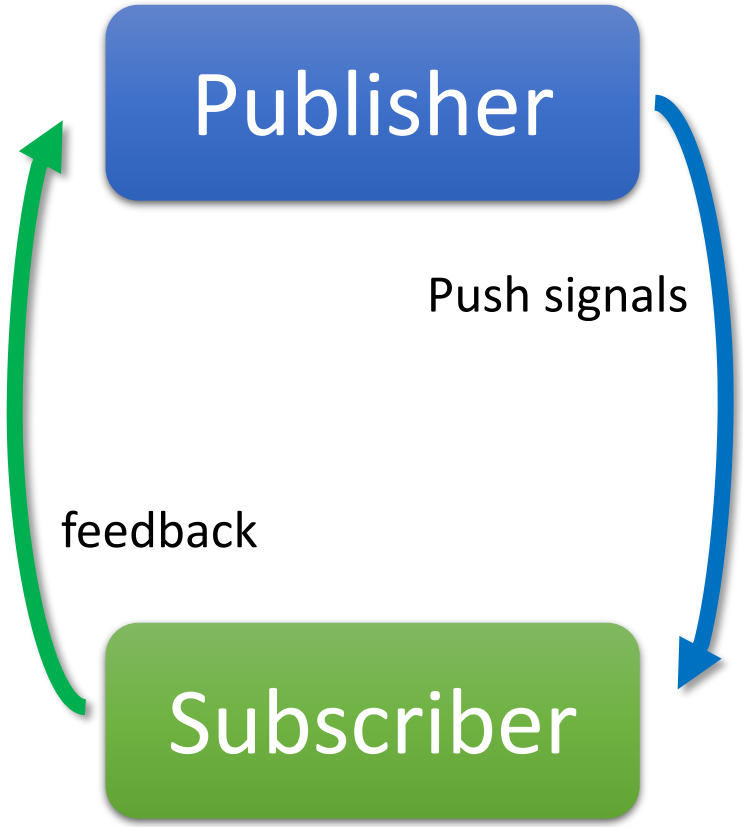
- Implemented by
  - **RxJava**
  - **Project Reactor**
  - **Akka Streams**
  - **Vert.x**
  - **Ratpack**



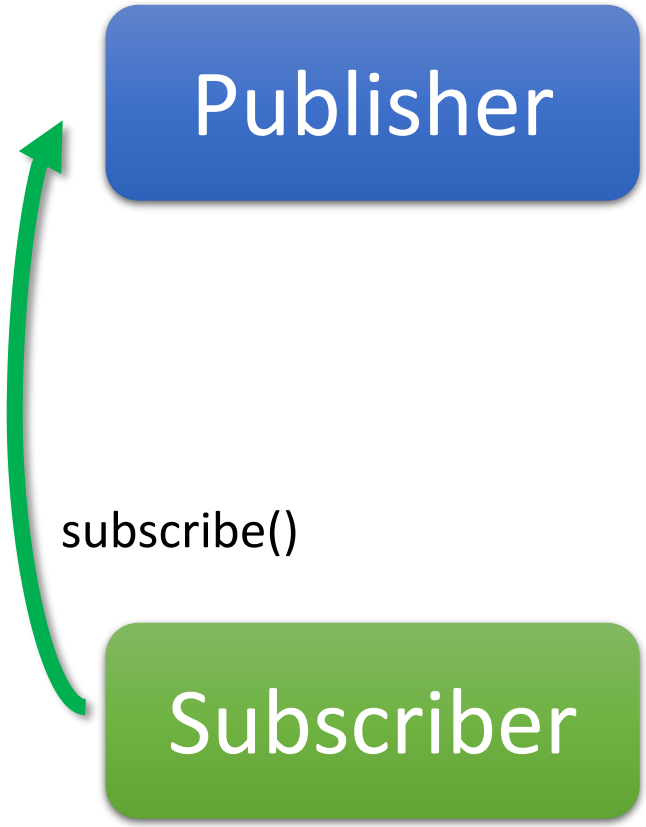
http://www.reactive-streams.org/

Backpressure

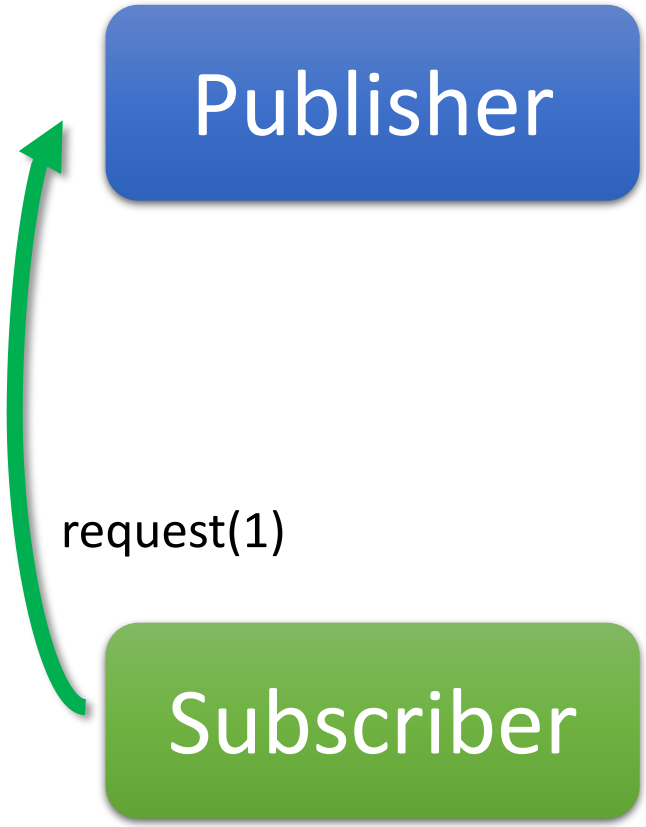
Control in-flight data



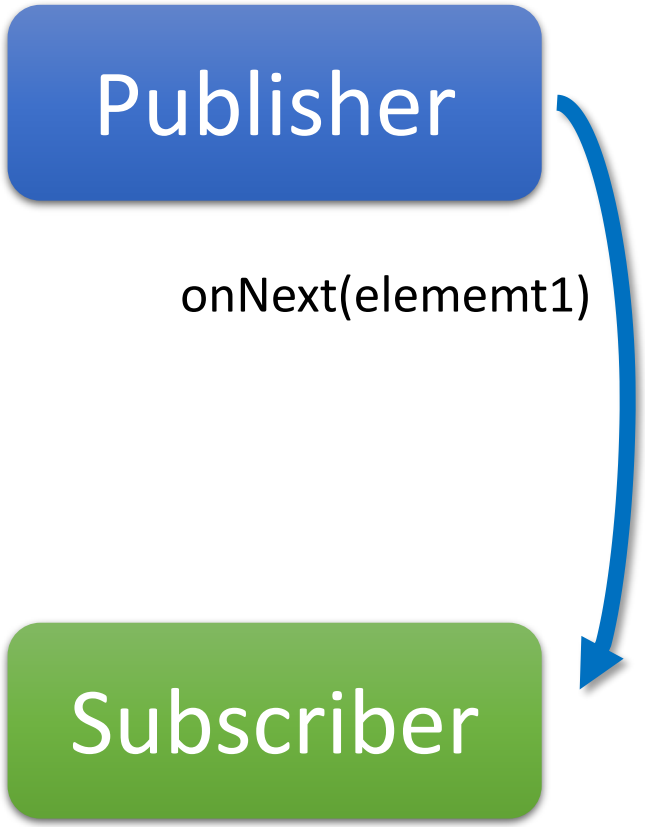
Backpressure



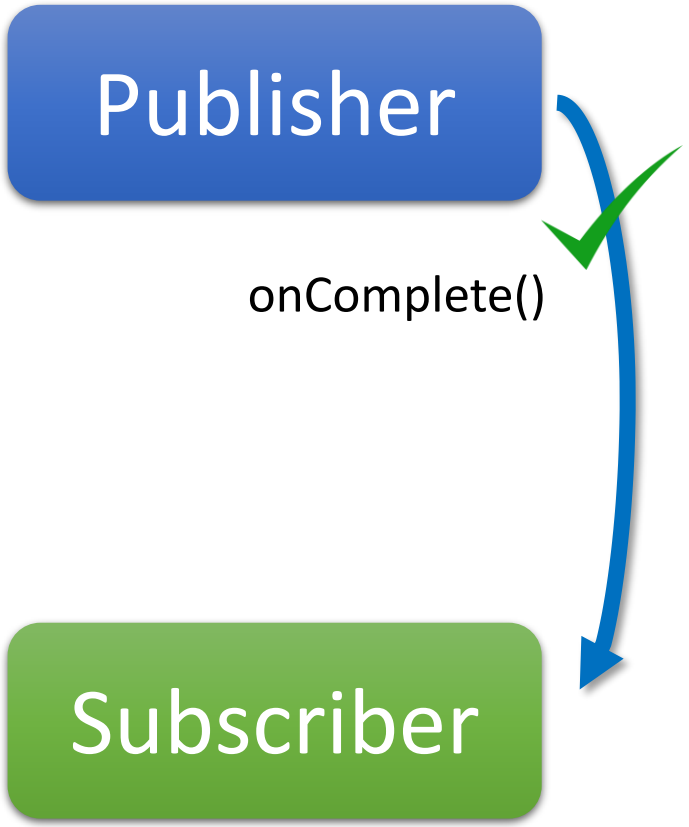
Backpressure



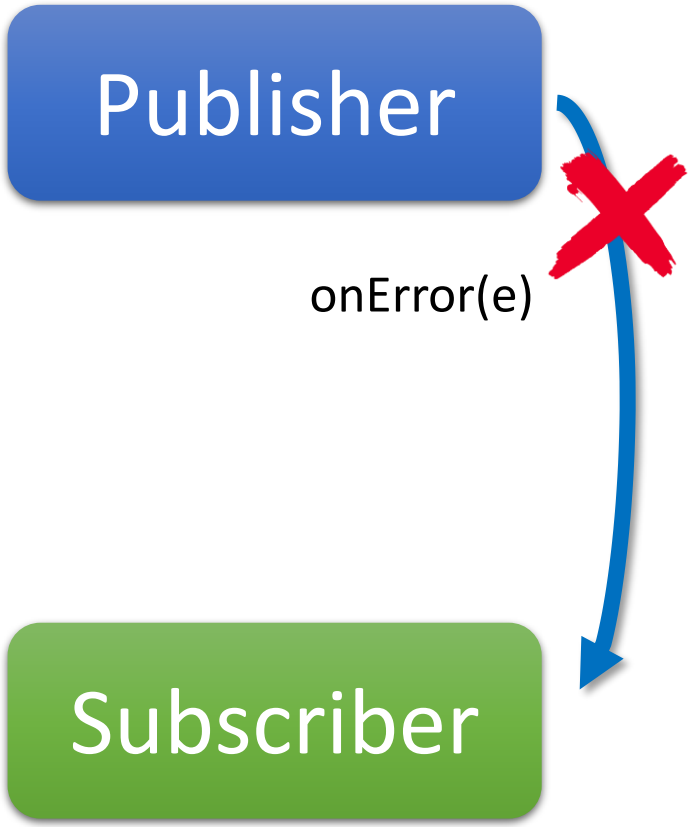
Backpressure

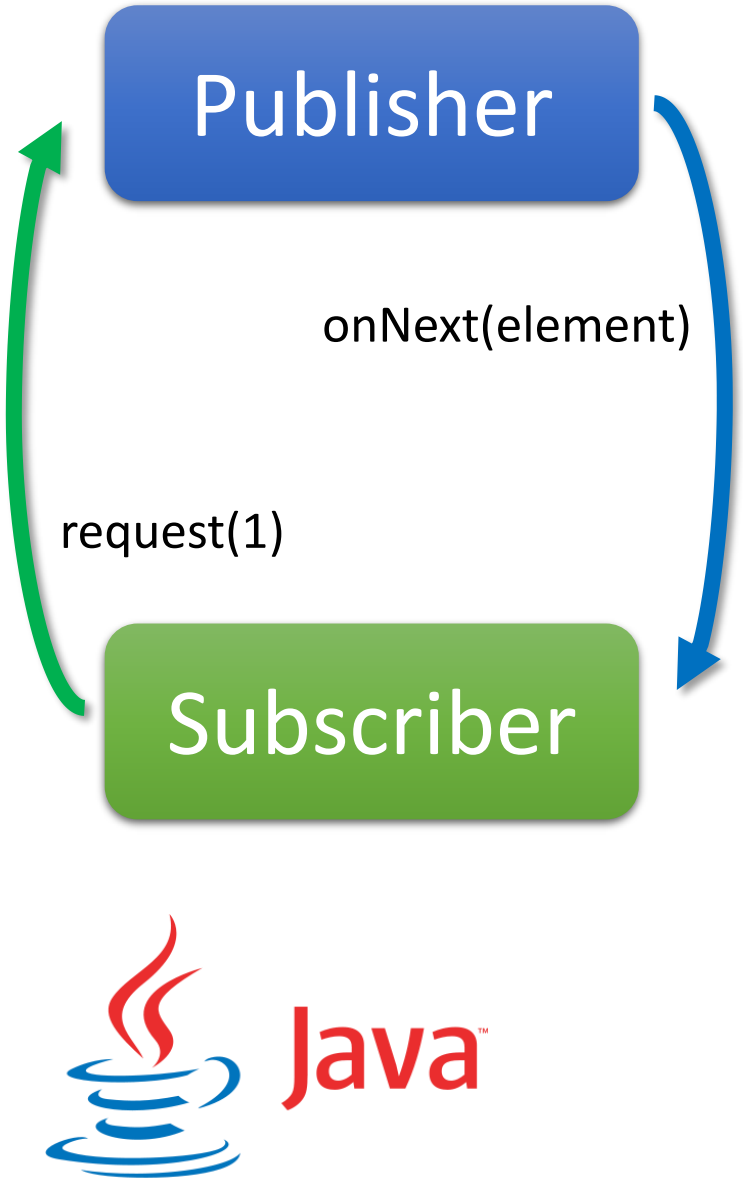


Backpressure



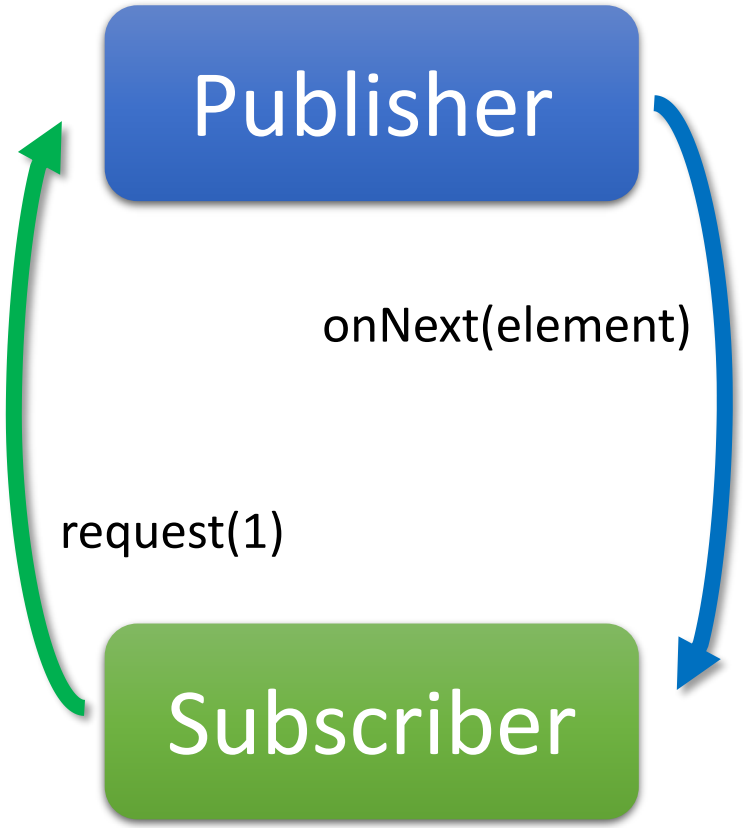
Backpressure





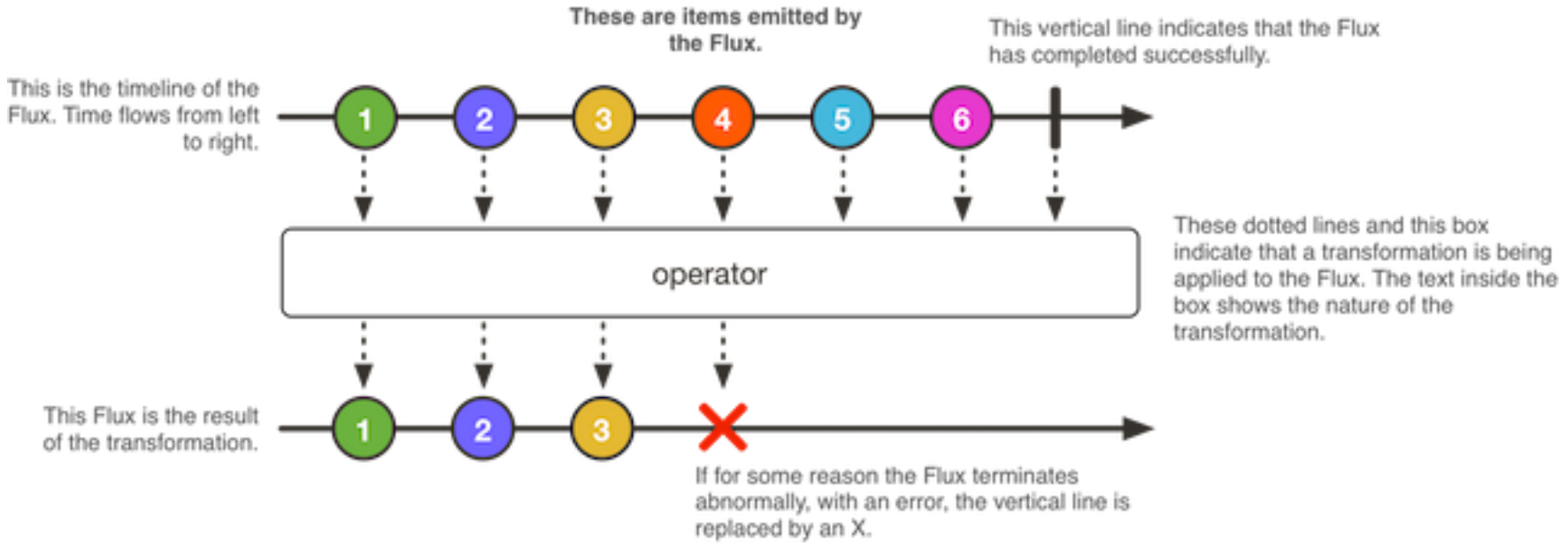

```java
interface Publisher<T> {

    public void subscribe(Subscriber<? super T> subscriber);
}
```

```java
interface Subscriber<T> {

    public void onNext(T item);

    public void onError(Throwable throwable);

    public void onComplete();

    public void onSubscribe(Subscription subscription);
}
```



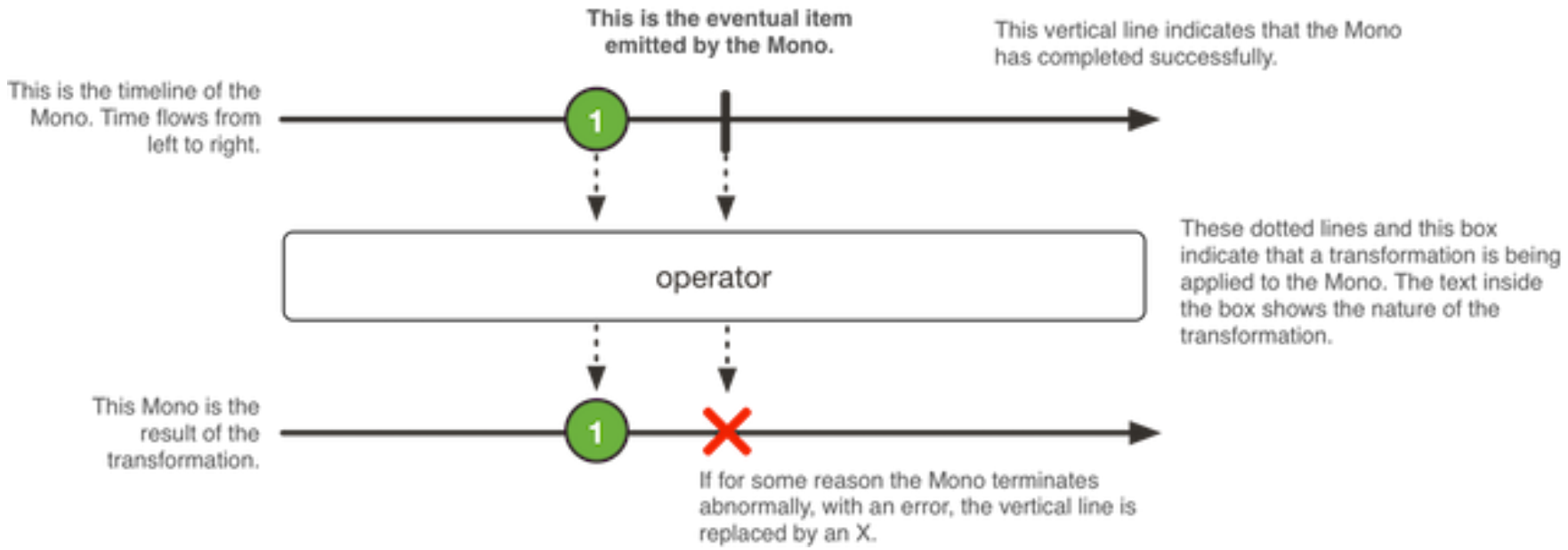

```
public static interface Subscription {

    public void request(long n);

    public void cancel();
}
```

Flux



http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html

Mono



http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html

Reactive Idioms

```
User fetchUser(String name);
```

```
Mono<User> fetchUser(String name);
```

Reactive Idioms

```
void persistReservation(Reservation reservation);
```

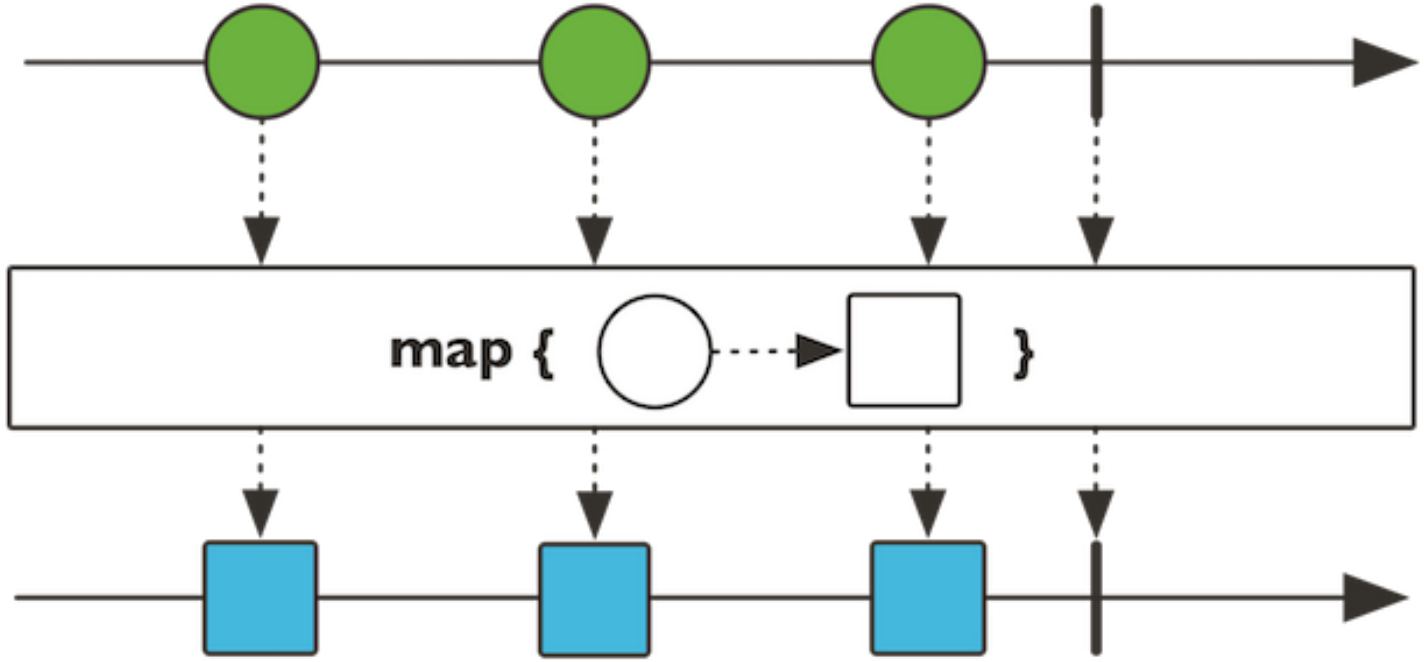
```
Mono<Void> persistReservation(Reservation reservation);
```

Reactive Idioms

```
Collection<Reservation> fetchReservations(User user);
```

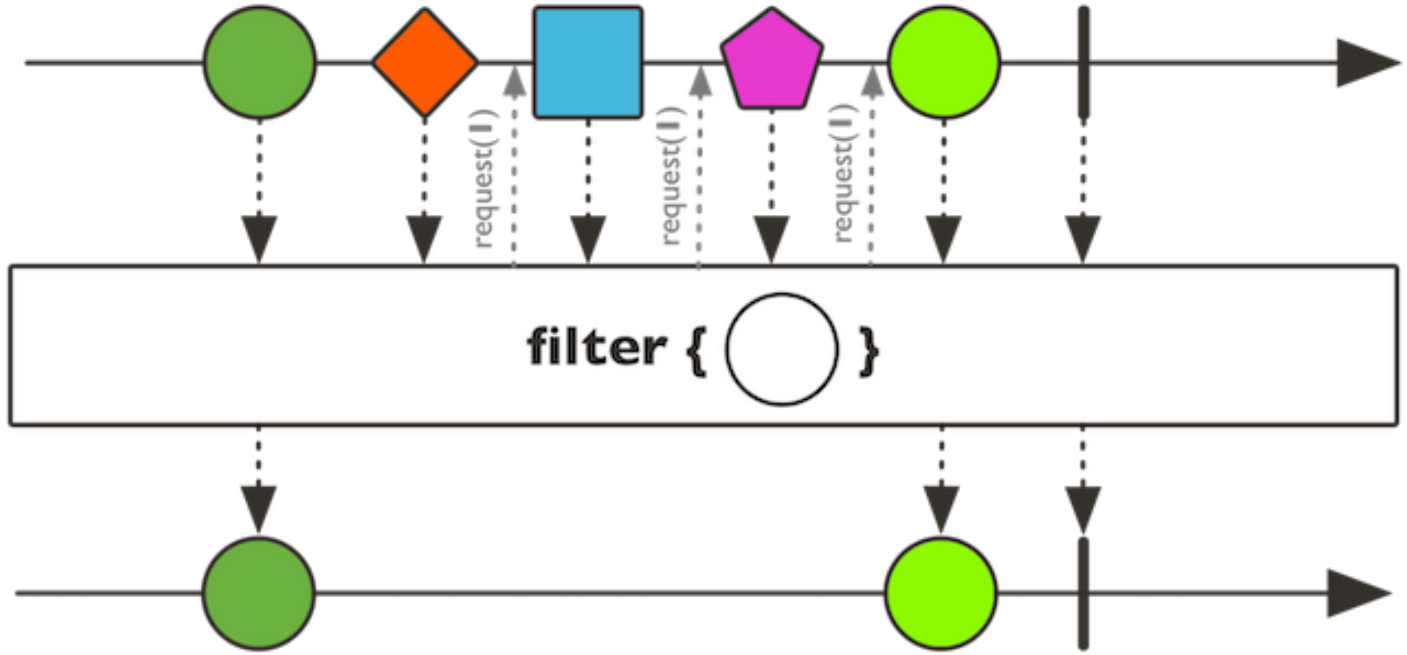
```
Flux<Reservation> fetchReservations(User user);
```

Functional style



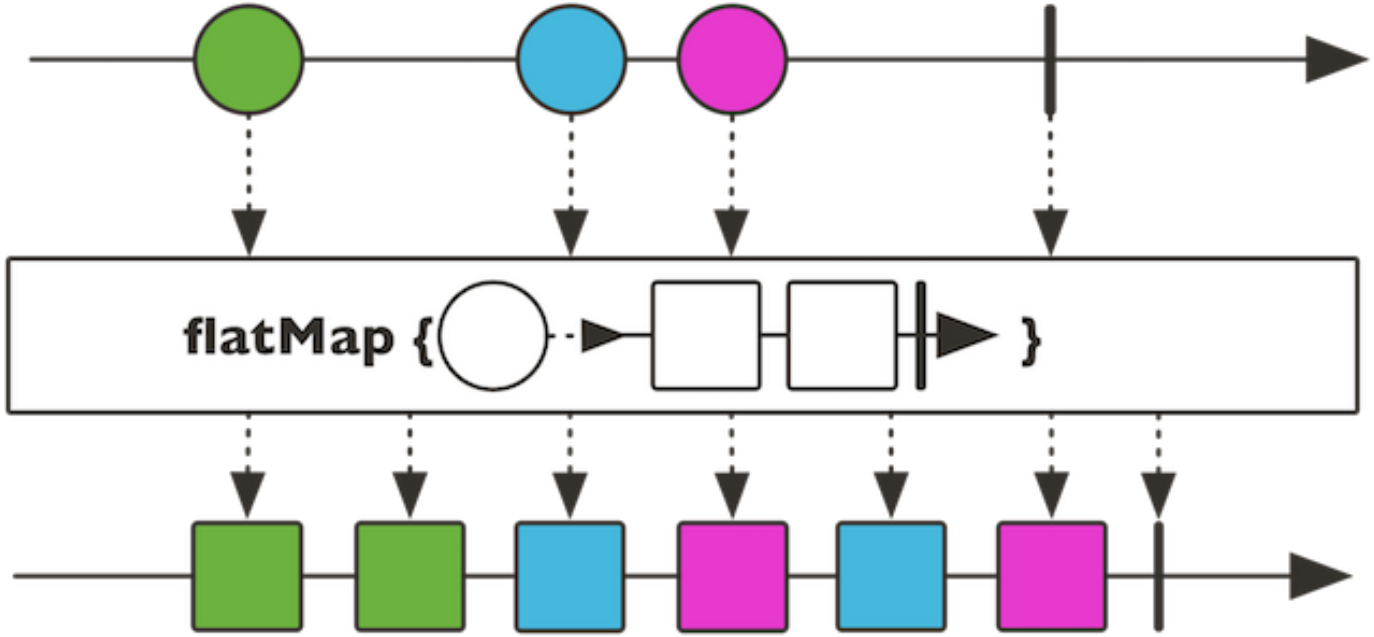

```
Flux.range(1, 10)
        .map(i -> (i % 2 == 0) ? "odd" : "even")
```

Functional style



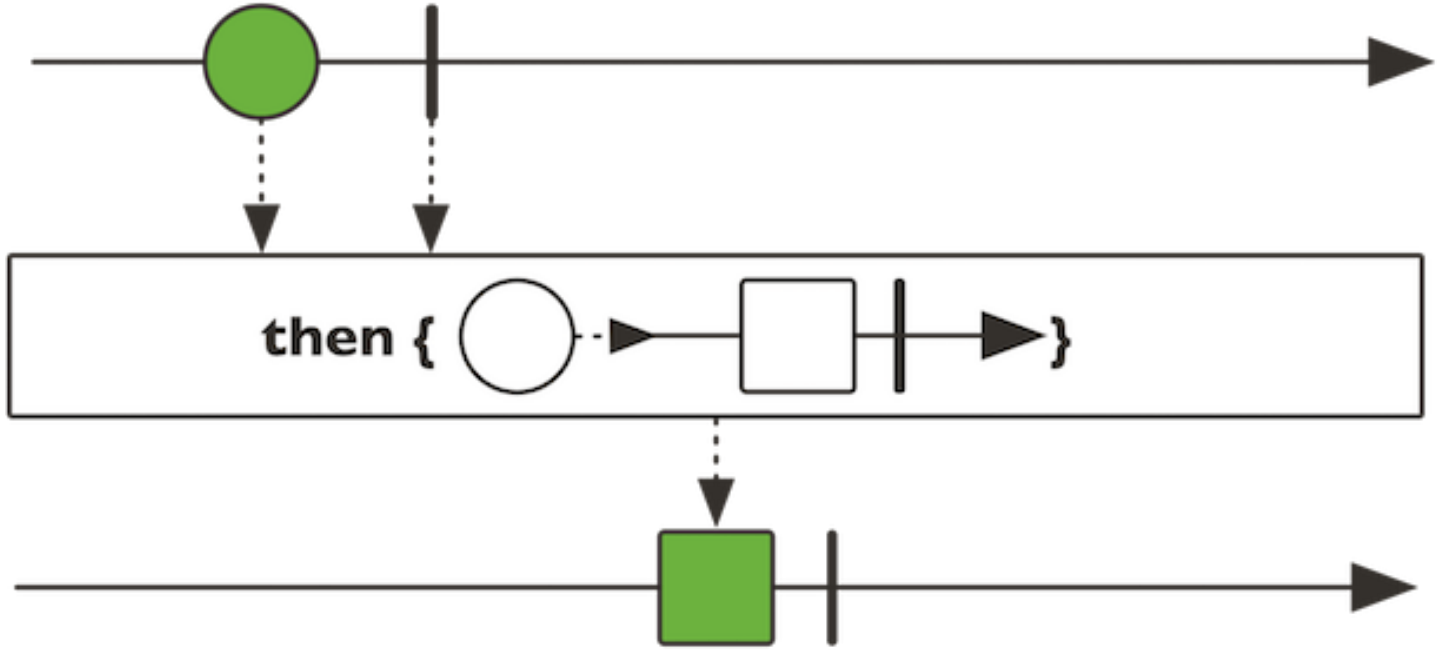

```
Flux.range(1, 10)
        .filter(i -> (i % 2 == 0))
```

Functional style



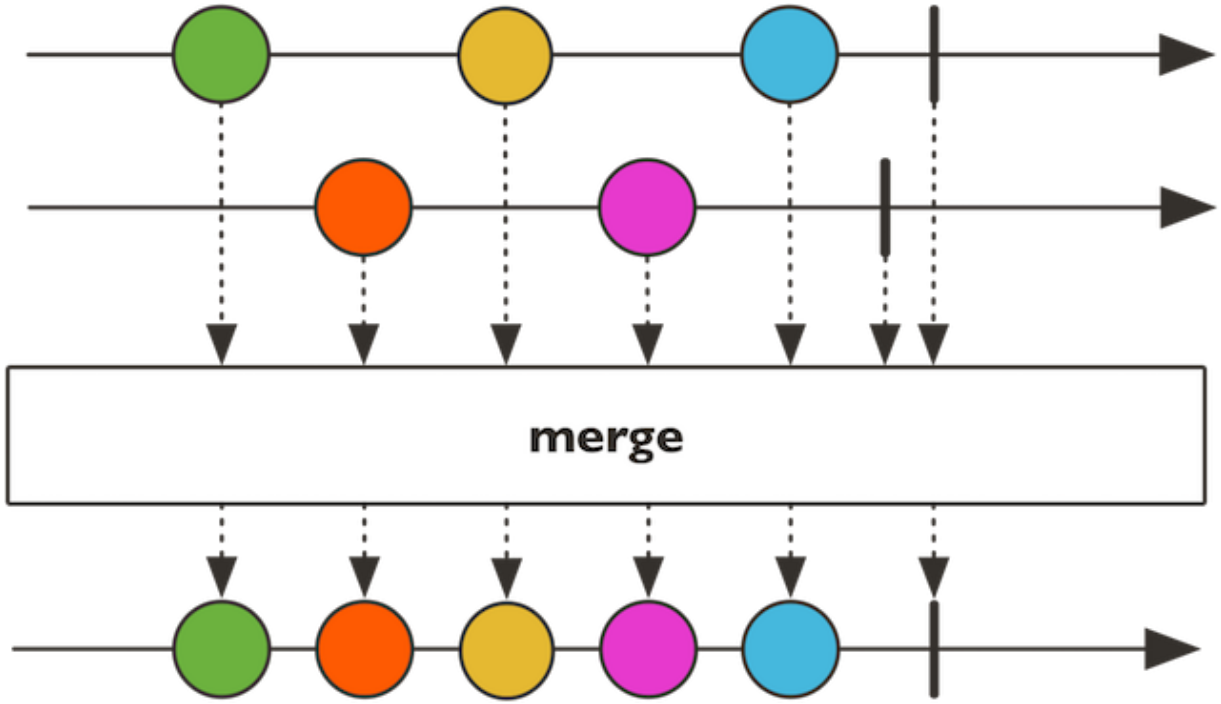

```
repository.insert(reservation)
    .flatMap(aVoid -> vehiclesService.getAvailableVehicles(location))
```

Functional style



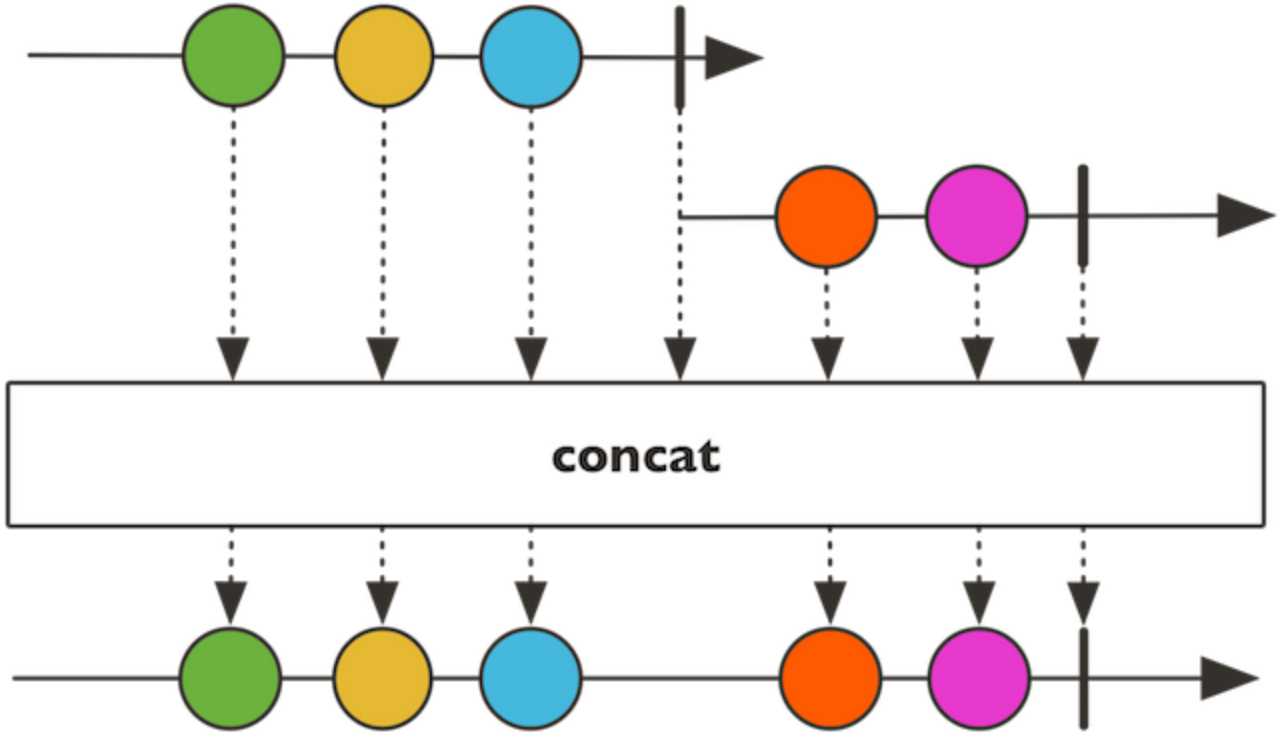

```
repository.insert(reservation)
    .then()
```

Functional style



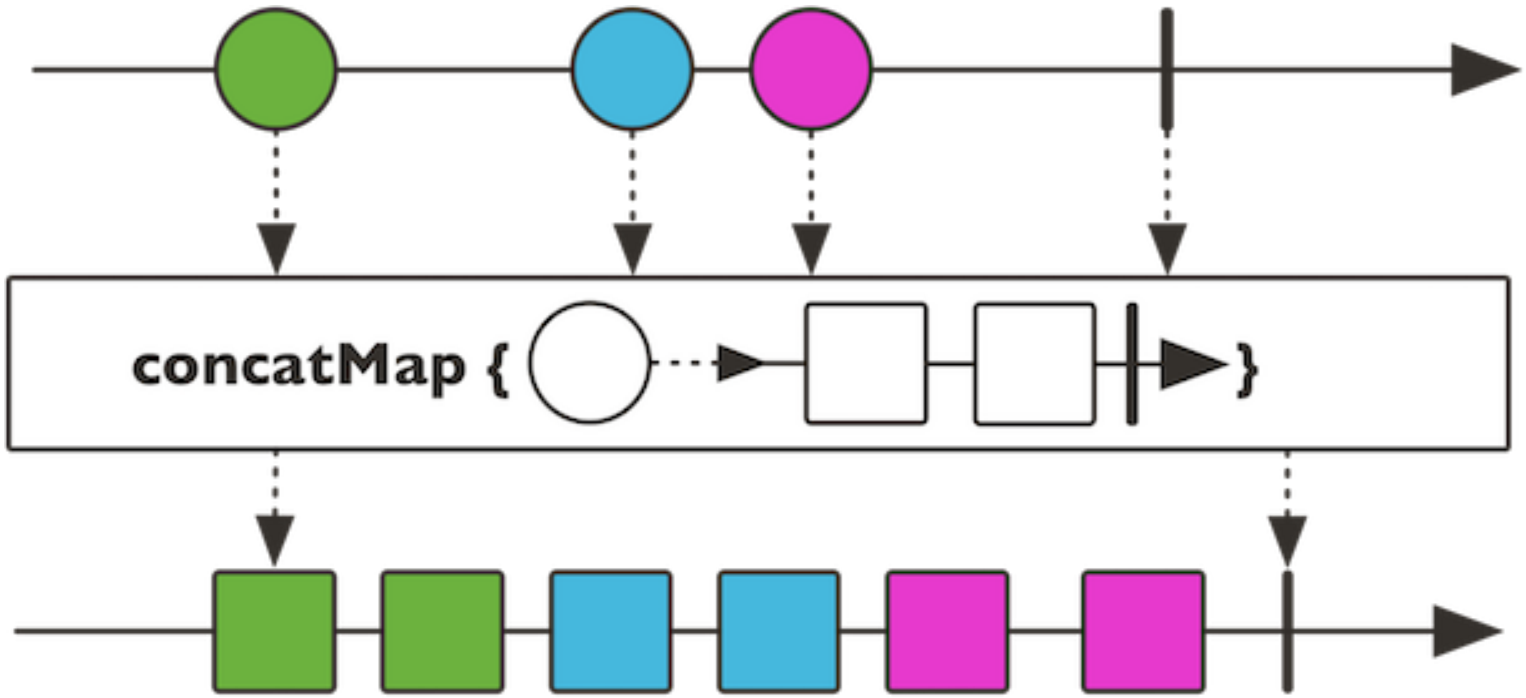

```
Flux.range(1, 10)
        .mergeWith(Flux.range(11, 20))
```

Functional style

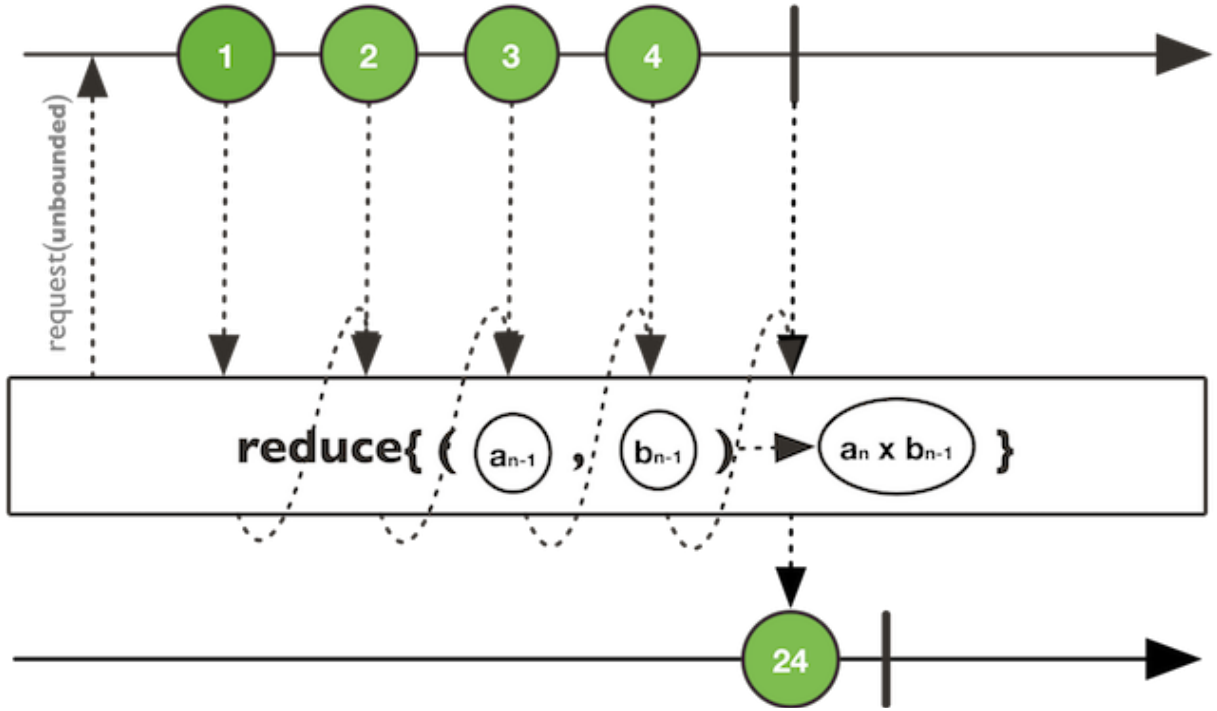



```
Flux.range(1, 10)
        .concatWith(Flux.range(11, 20))
```

Functional style

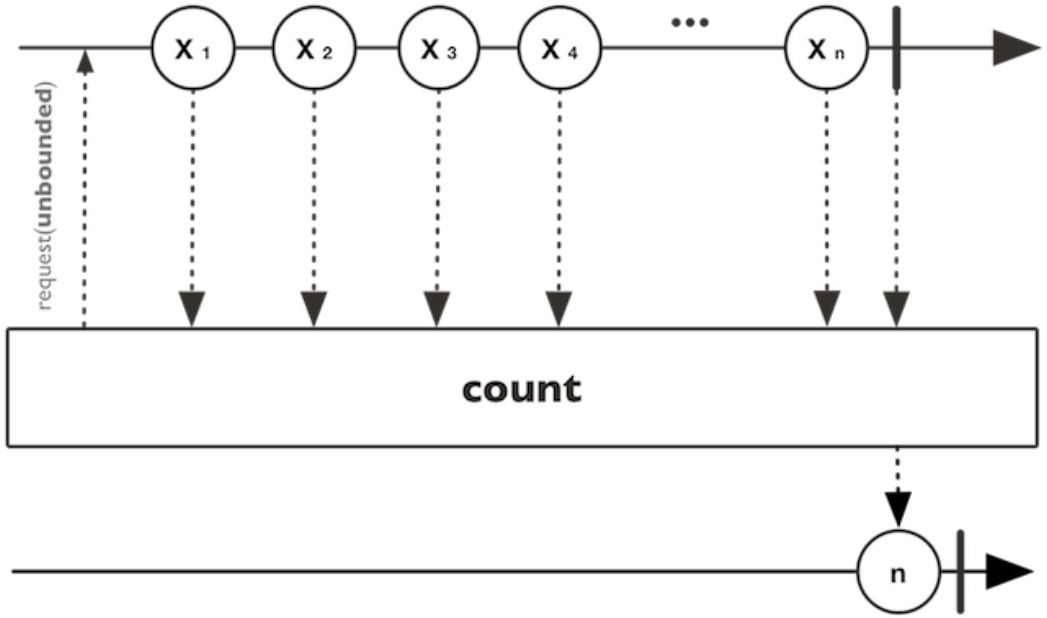


Functional style



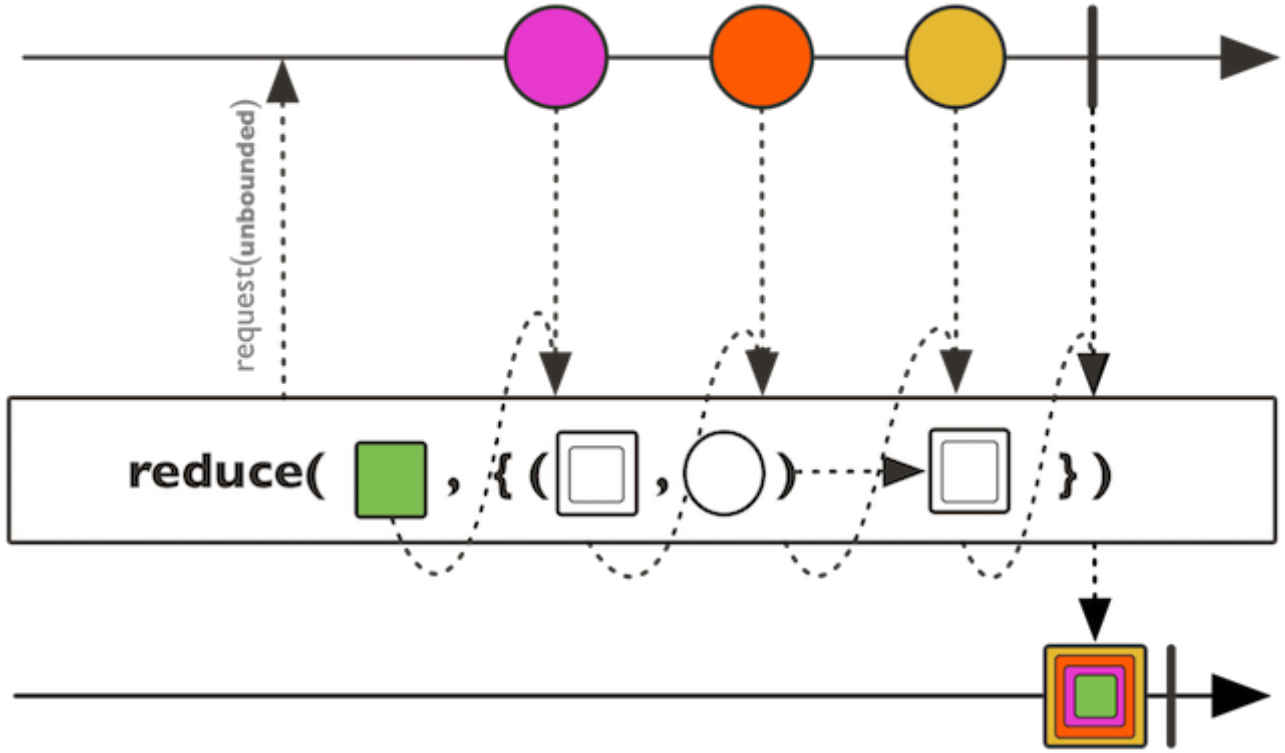

```
locateVehicles(l, radius)
    .take(Duration.ofSeconds(10))
    .reduce(0L, (count, vehicle) -> count + 1);
```

Functional style

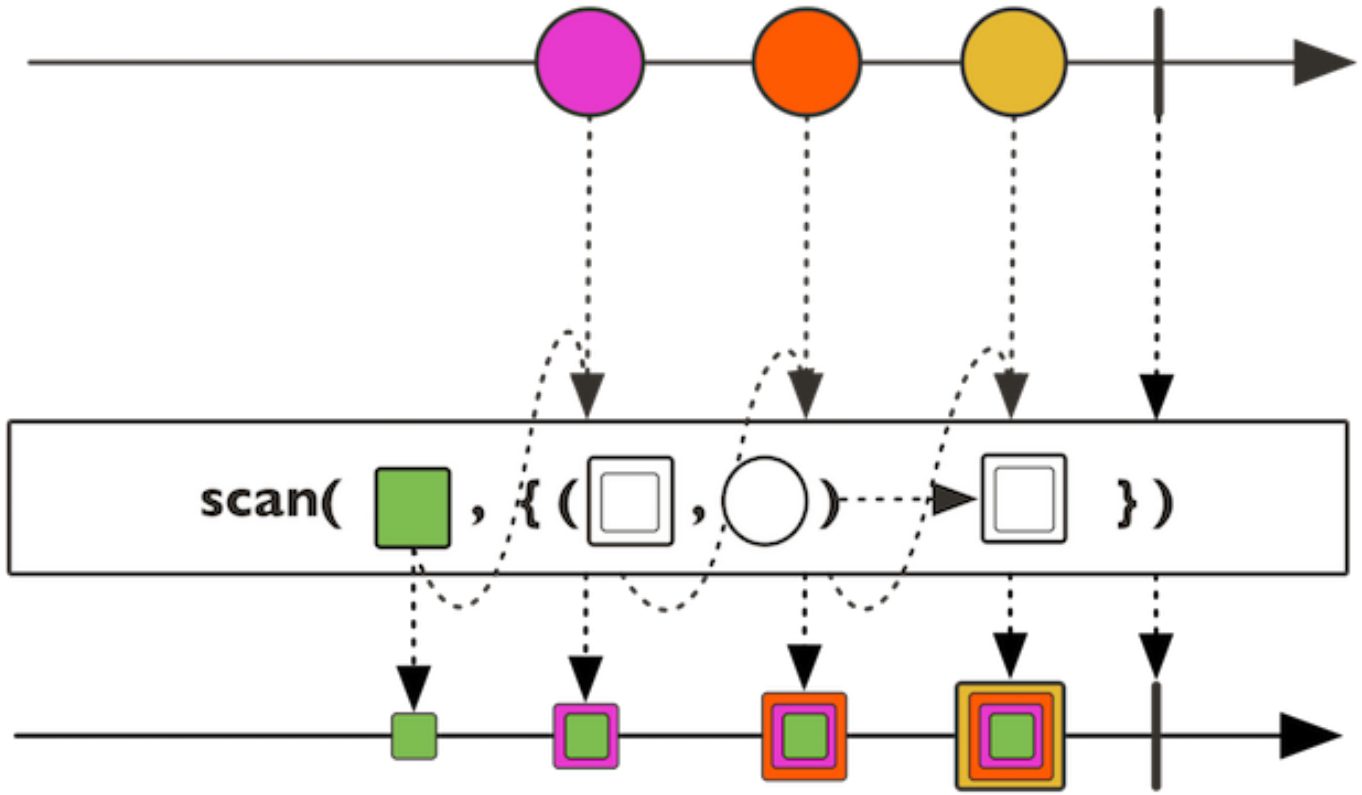



```
locateVehicles(l, radius)
    .take(Duration.ofSeconds(10))
    .count();
```

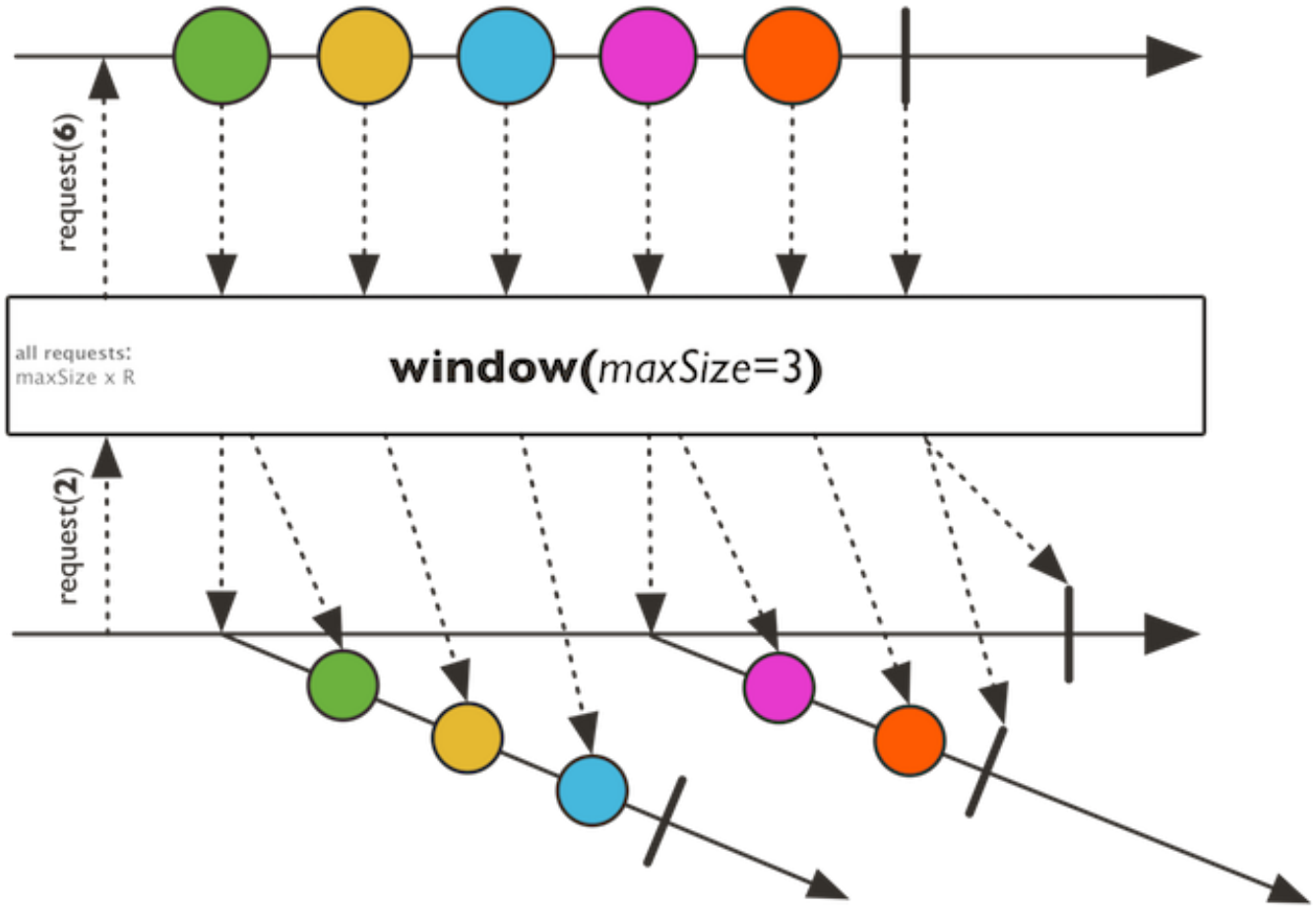
Functional style



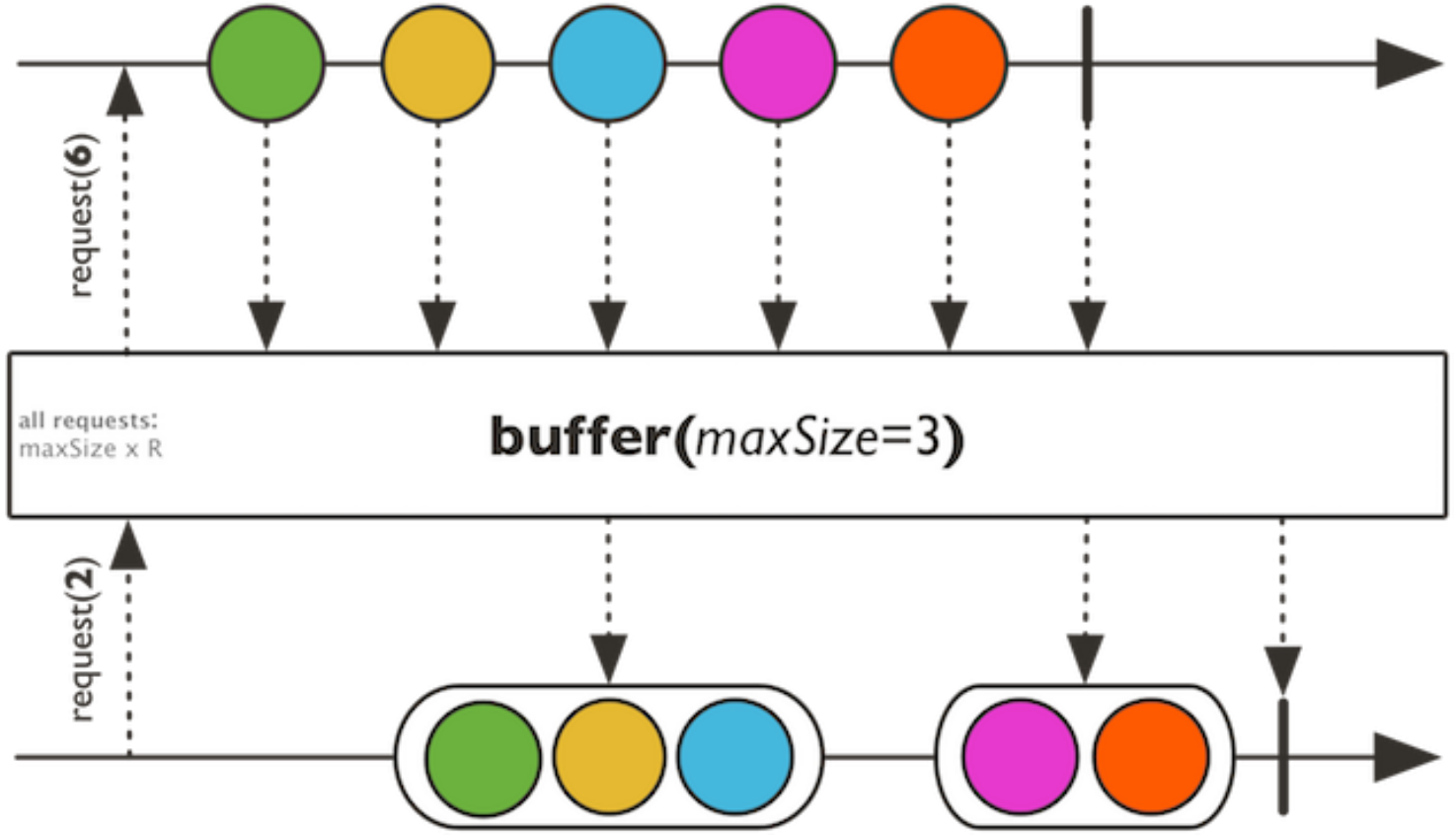
Functional style



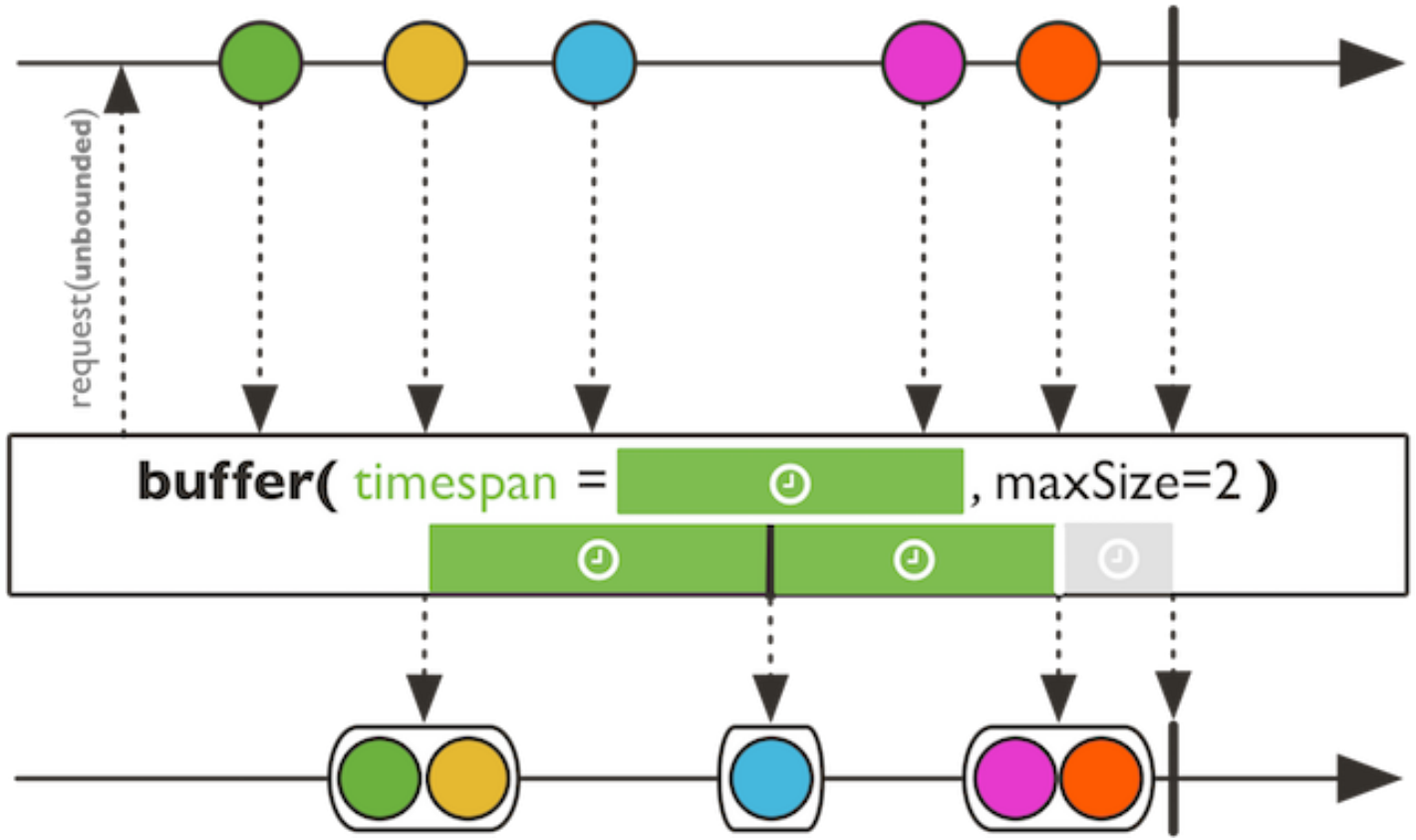
Batching: window



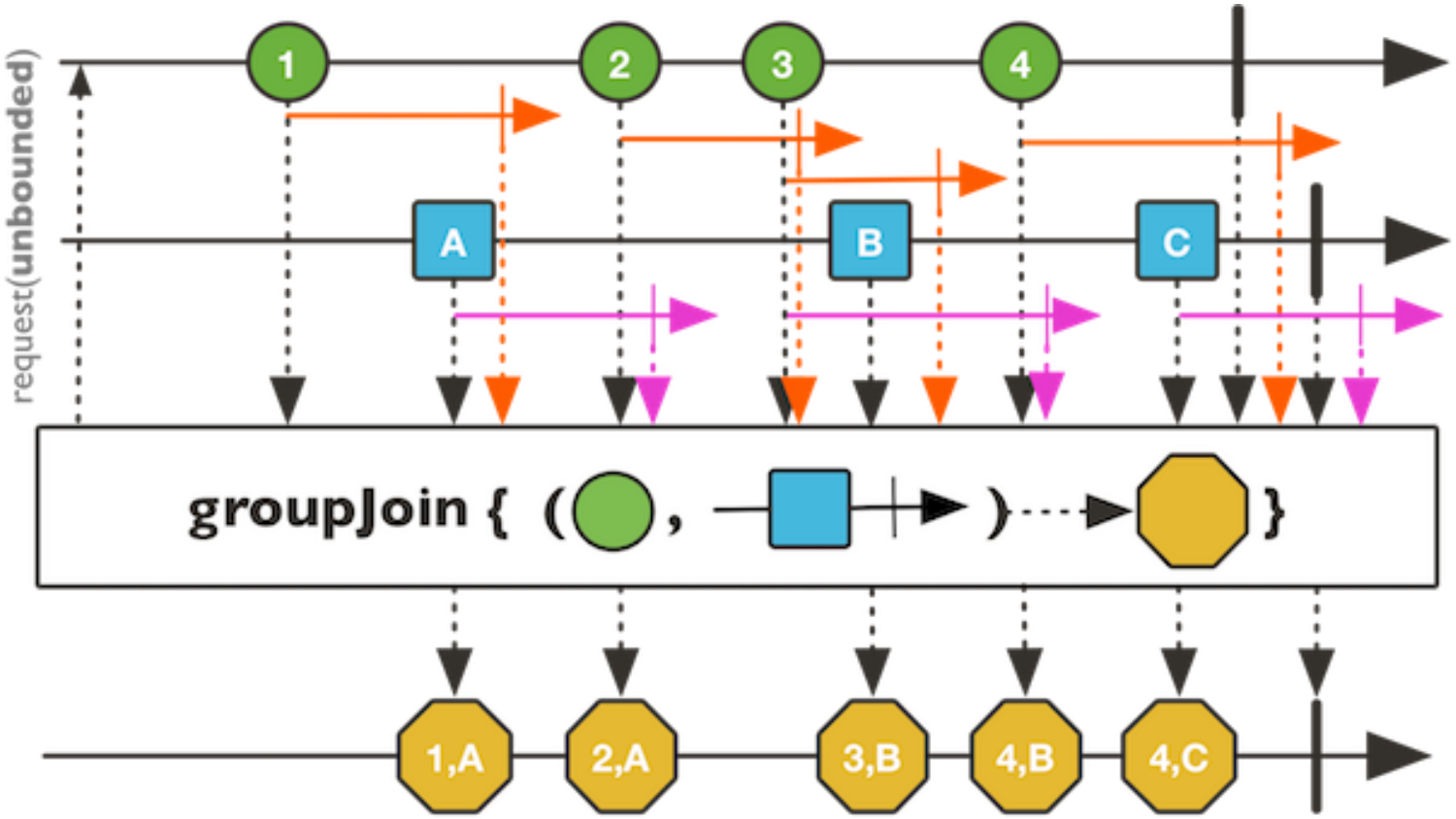
Functional style



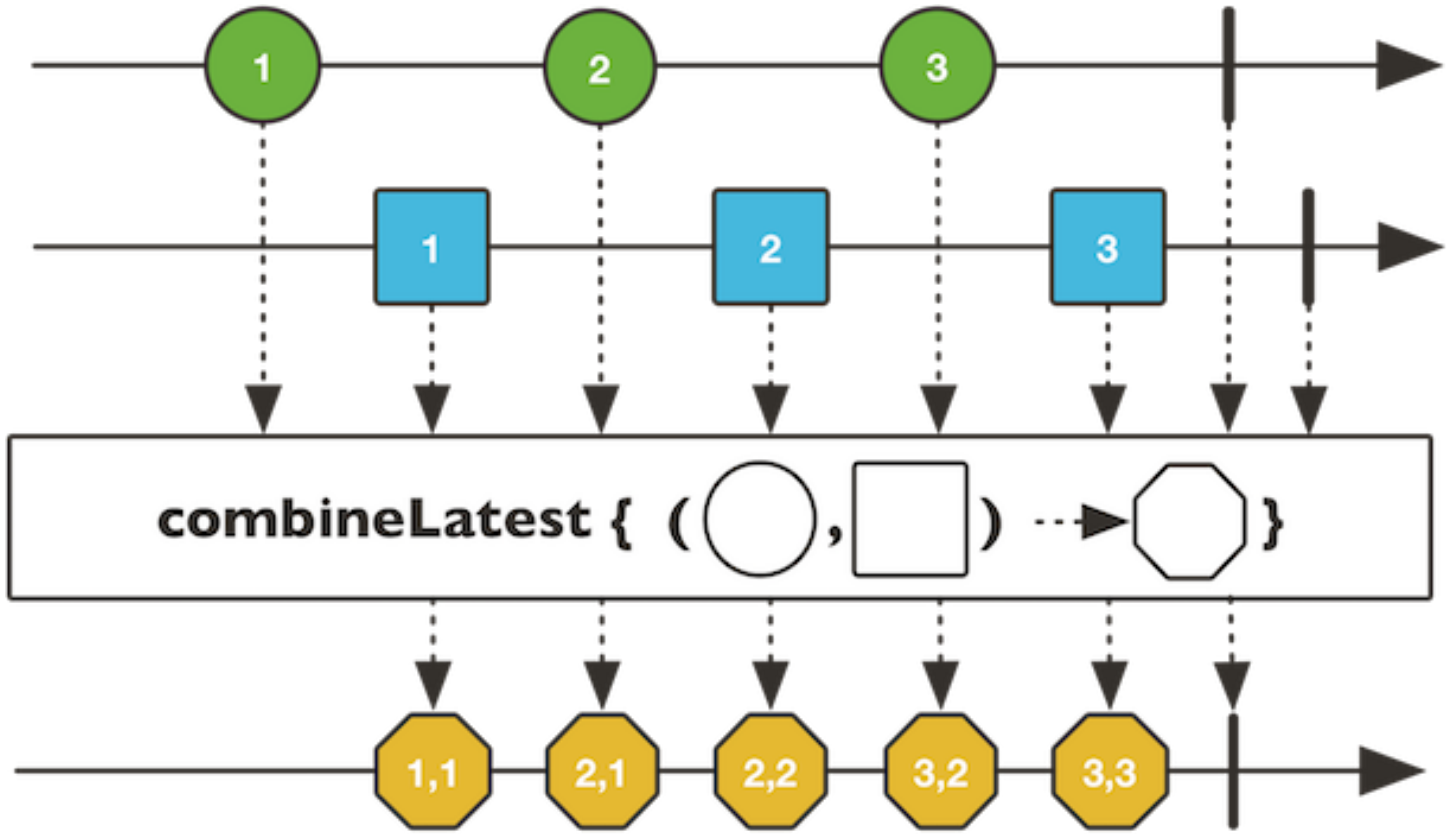
Functional style



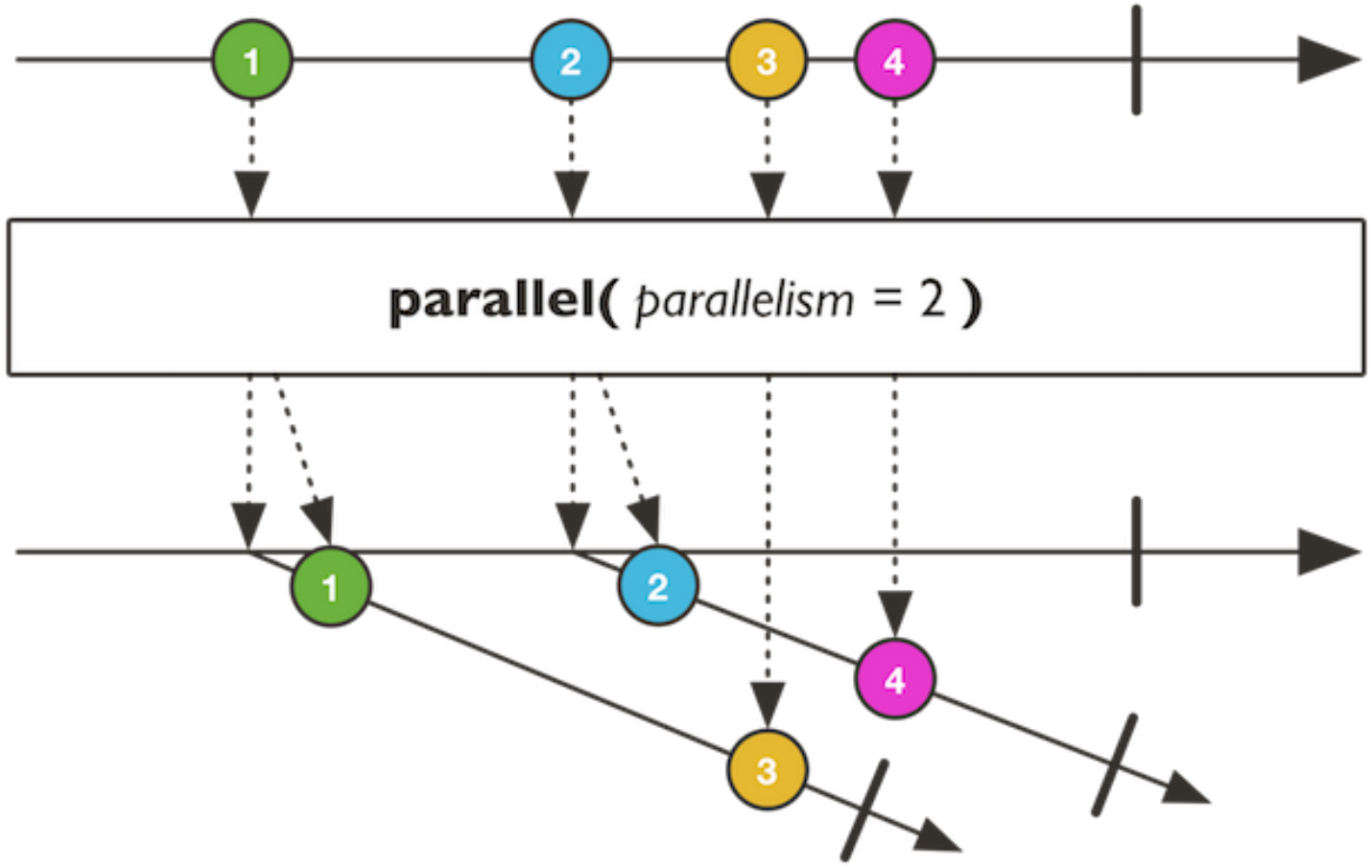
Functional style



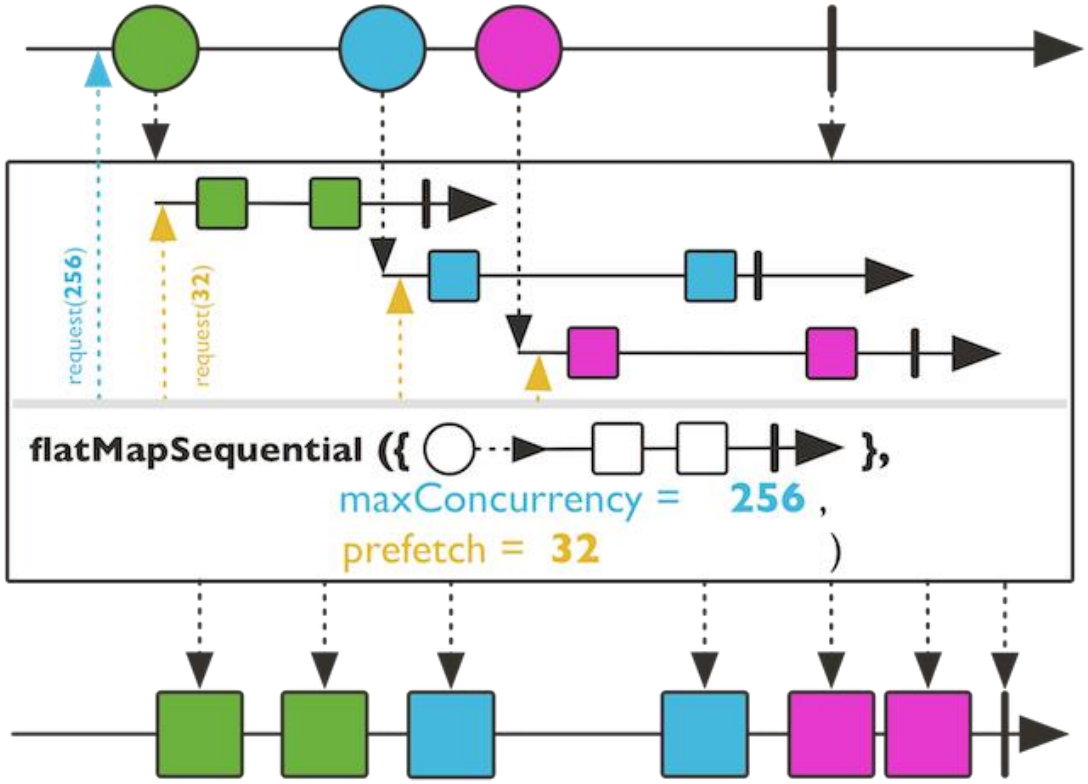
Functional style



Functional style



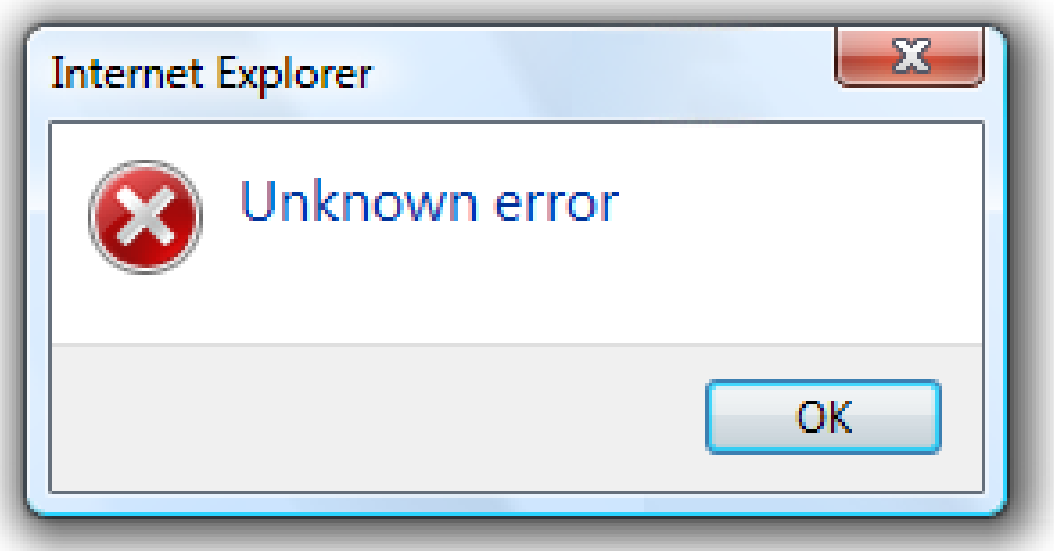
Functional style




```
Flux.fromIterable(Stream.rangeBy(0, count, pageSize))
    .flatMapSequential(offset -> fetchPaged(offset, pageSize), 10, 500)
```

```
return Mono.just(Tuples.of(userId, location))
    .map(t -> new Reservation(id, time, t.getT1(), t.getT2()))
    .flatMap(repository::persistReservation).flux()
    .flatMap(aVoid -> vehiclesServiceApi.getAvailableVehicles(location))
    .take(20)
    .map(vehicle -> new RideOffer(vehicle, time, location))
    .flatMap(repository::offerRide)
    .timeout(Duration.ofSeconds(30))
    .onErrorMap(e -> new RidesException("Failed to handle user " + userId, e))
    .doOnComplete(() -> log.info("Handled reservation for user '{}'", userId))
    .then();
}
```

Error Handling



Error Handling

```
public Flux<Integer> divide100By(Flux<Integer> dividers) {
    try {
        return dividers.flatMap(div -> Mono.just(100 / div));
    } catch (ArithmeticException e) {
        log.warn("Can't divide", e);
    }
}
```

Error Handling

```
public Flux<Integer> divide100By(Flux<Integer> dividers) {
    return dividers.flatMap(div ->
        Mono.just(100 / div)
            .onErrorMap(e -> new MyHandlingException("Never again!", e)));
}
```

ResumeOnError

```
public Flux<Integer> divide100By(Flux<Integer> dividers) {
    return dividers.flatMap(div ->
        Mono.just(100 / div)
            .doOnError(e -> {
                if (e instanceof ArithmeticException) log.warn("0 again?!", e);
            })
            .onErrorResume(ArithmeticException.class, e -> Mono.empty())
    );
}
```

Error Strategy (Coming in v3.2.0)

```
public Flux<Integer> divide100By(Flux<Integer> dividers) {
    return dividers.map(div -> 100 / div)
        .errorStrategyContinue(ArithmeticException.class,
            (error, value) -> log.warn("0 again?!", e));
}
```

Nothing happens until subscribe()

```
handleReservation(reservation)
    .subscribe();
```

Nothing happens until subscribe()

```
Disposable disposable = handleReservation(reservation)
        .subscribe();

disposable.dispose();
```

Nothing happens until subscribe()

```
handleReservation(reservation)
    .subscribe(aVoid -> log.info("Reservation handled"));
```

Also, there's block()…

```
RiderOffer offer = handleReservation(reservation)
    .block();
```

State Propagation with Context

```
Context a = Context.of("key", "value");
a.get("key"); // ⇒ “value”
```

State Propagation with Context

```
Mono<String> mono = Mono.just("Hello")
      .flatMap(s -> Mono.subscriberContext()
            .map(ctx -> s + " " + ctx.getOrDefault("key", "Stranger")))
      .subscriberContext(ctx -> ctx.put("key", "World"));


  StepVerifier.create(mono)
     .expectNext("Hello World")
     .verifyComplete();
```

Threading Contexts

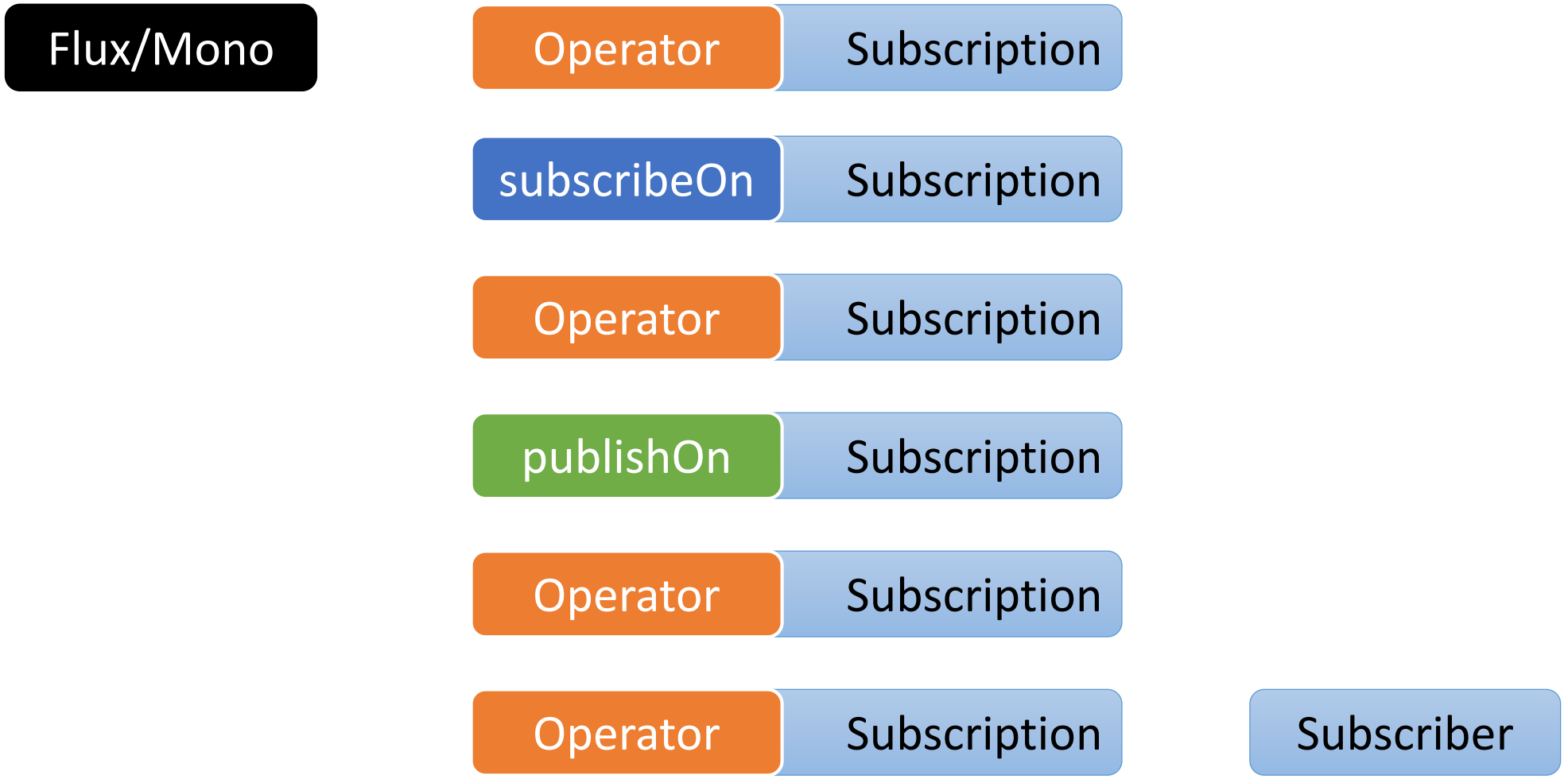
Threading Contexts

- Flux declaration is orthogonal to the thread context it operates in
- Scheduler choice:
  - Single, elastic, parallel, timer
- Reactor allows switching contexts

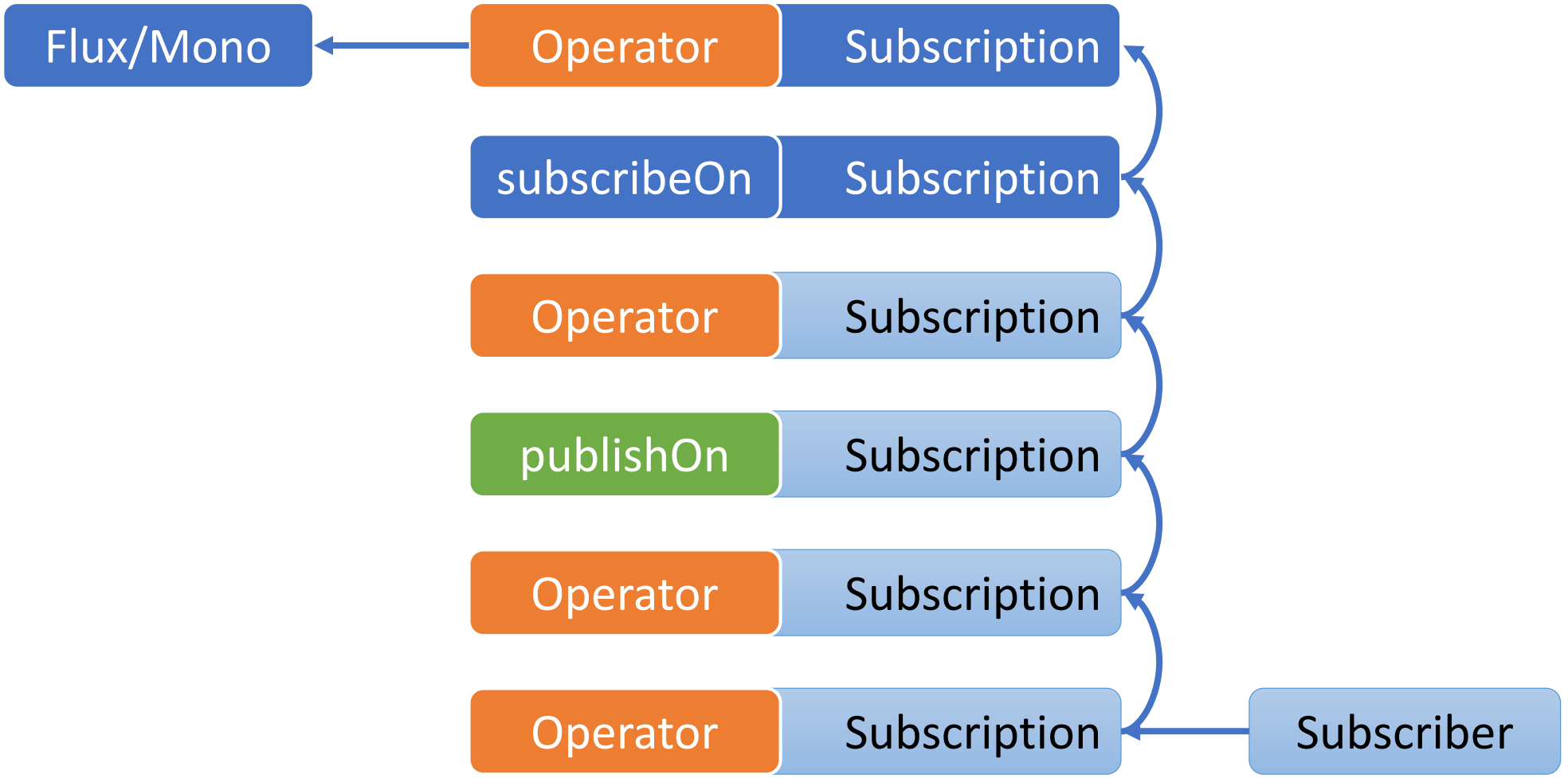
Switching Threading Contexts

- **publishOn**
  - Switch rest of the Flux (downstream) to a particular thread
- **subscribeOn**
  - Switch the subscription and request to a particular thread

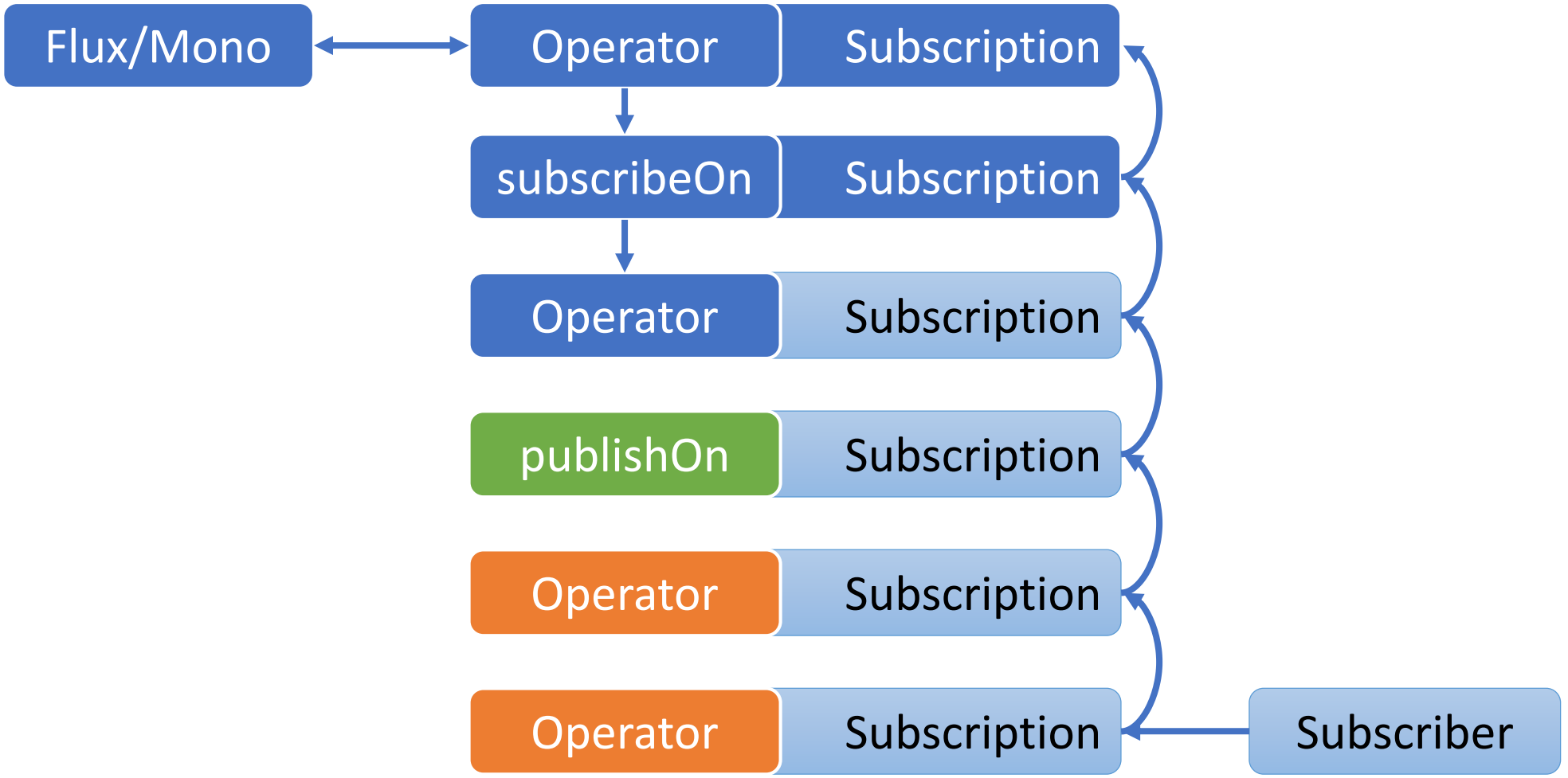
Switching Threading Contexts



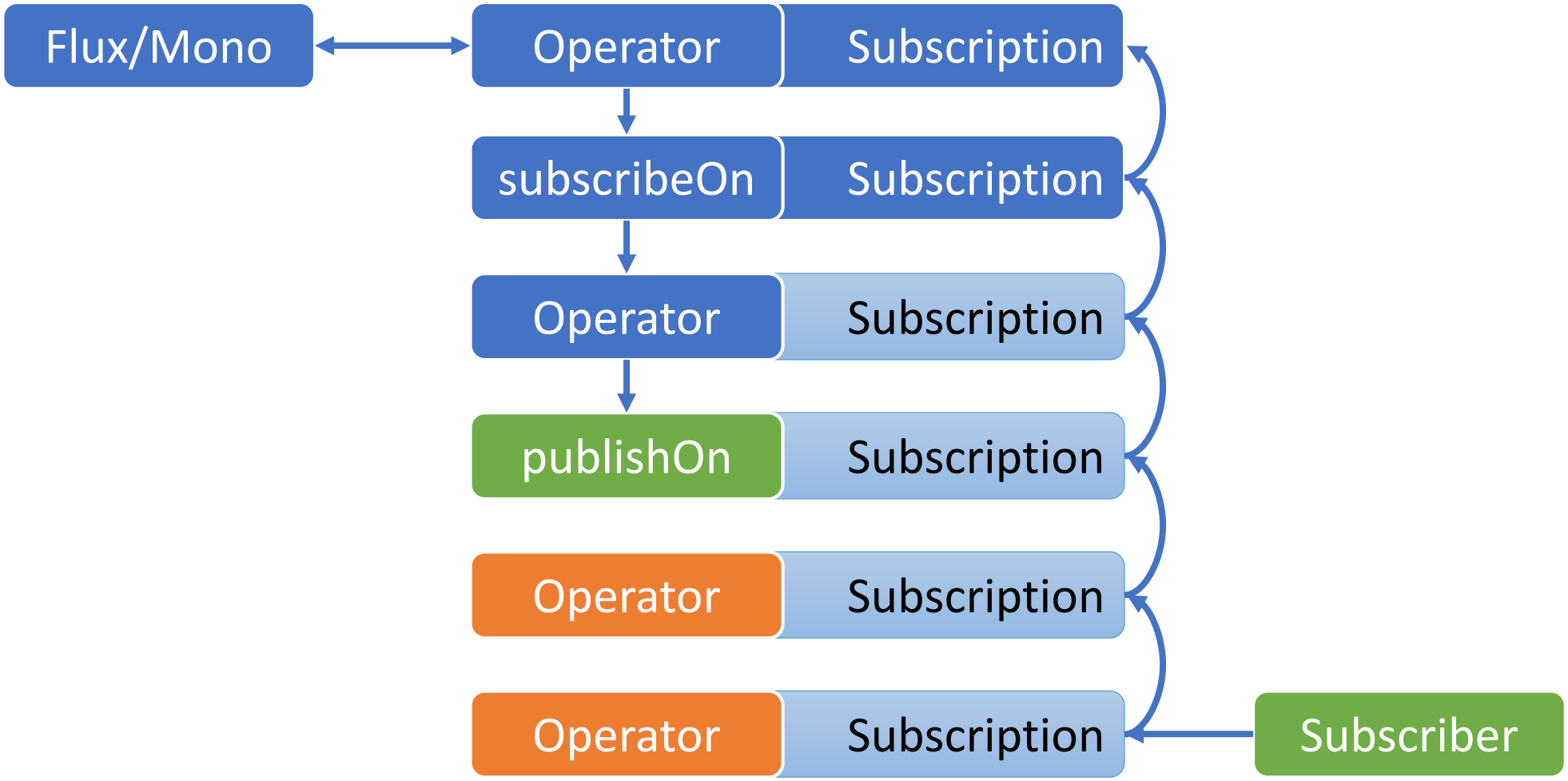
Switching Threading Contexts



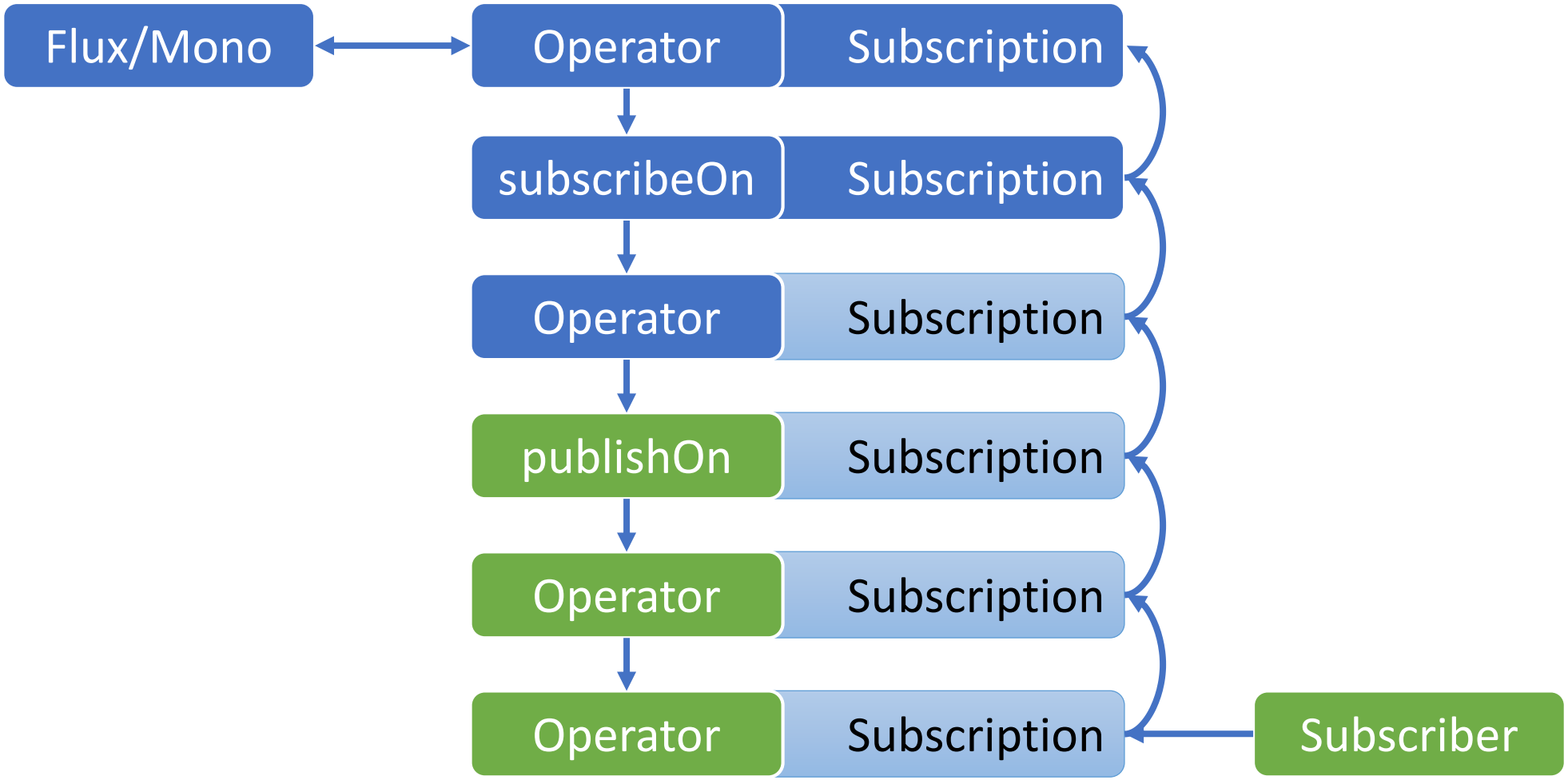
Switching Threading Contexts



Switching Threading Contexts



Switching Threading Contexts



Testing

Testing with StepVerifier

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

StepVerifier.create(publisher)
        .expectNext(1)
        .assertNext(v -> assertThat(v).isBetween(1, 3))
        .expectNextMatches(item -> item == 3)
        .expectNextCount(2)
        .expectComplete()
        .verify();
```

Testing Errors

```
Flux<Integer> publisher = Flux.just(1, 2, 3, 4, 5)
        .concatWith(Flux.error(new IllegalStateException("wat?!")));

StepVerifier.create(publisher)
        .expectNext(1)
        .expectNextCount(4)
        .expectErrorMessage("wat?!")
        .verify();
```

Testing with StepVerifier - Virtual Time!

```
Flux<Long> publisher = Flux.interval(Duration.ofSeconds(45)).take(2);

StepVerifier.withVirtualTime(() -> publisher)
        .thenAwait(Duration.ofSeconds(50))
        .expectNext(0L)
        .thenAwait(Duration.ofSeconds(40))
        .assertNext(v -> assertThat(v).isGreaterThan(0L))
        .expectComplete()
        .verify();
```

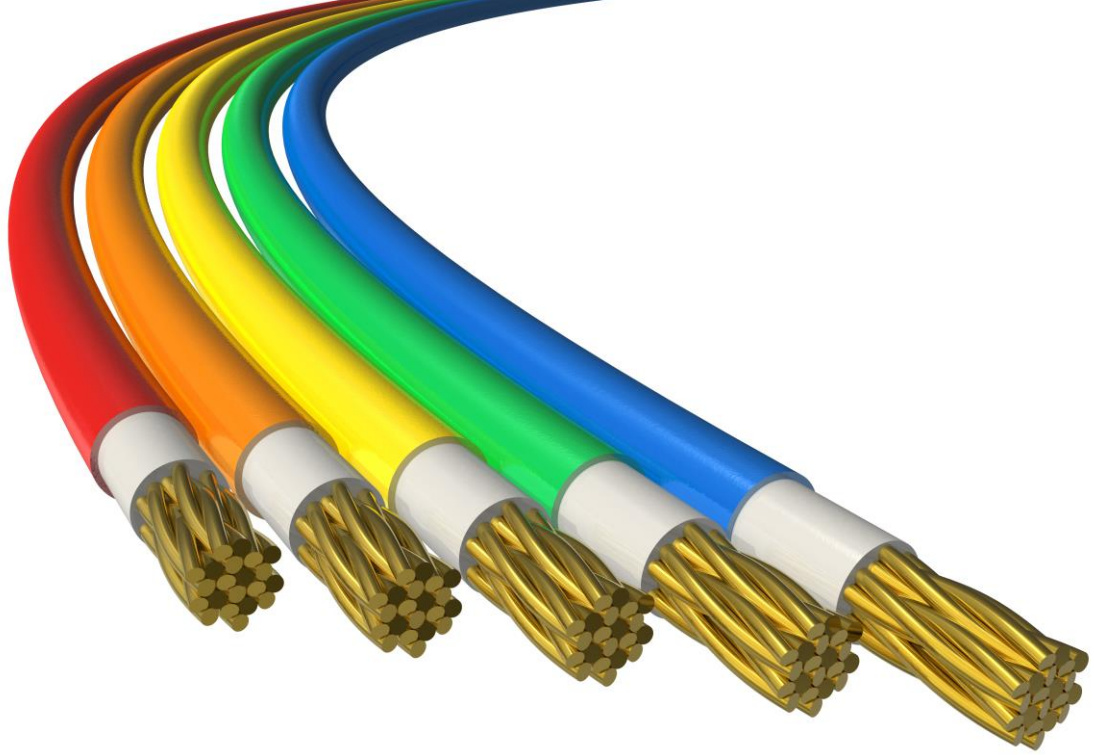
TestPublisher for testing compliance

```
TestPublisher.create().next(1, 2, 3, 4)
        .complete();

TestPublisher.createNoncompliant(TestPublisher.Violation.ALLOW_NULL)
        .emit(1, 2, 3, null, 4);
```

PublisherProbe for advanced flow testing

Disadvantages

- Learning curve
- Debuggability



Debugging & Visibility

Debugging via Logging

```
flux
    .switchIfEmpty(flux2)
    .retry(2)
    .timeout(Duration.ofSeconds(10))
    .log();
```

```
22:54:27.363 [main] INFO reactor.Flux.Take.1 -
onSubscribe(FluxTake.TakeSubscriber)
22:54:27.373 [main] INFO reactor.Flux.Take.1 - request(5)
22:54:27.394 [main] INFO reactor.Flux.Take.1 - onNext(Vehicle(id=v2))
22:54:27.394 [main] INFO reactor.Flux.Take.1 - onNext(Vehicle(id=v3))
22:54:27.395 [main] INFO reactor.Flux.Take.1 - onComplete()
```

Stacktraces are not informative

```
java.lang.IndexOutOfBoundsException: Source emitted more than one item
        at reactor.core.publisher.MonoSingle$SingleSubscriber.onNext(MonoSingle.java:120)
        at reactor.core.publisher.FluxFlatMap$FlatMapMain.emitScalar(FluxFlatMap.java:380)
        at reactor.core.publisher.FluxFlatMap$FlatMapMain.onNext(FluxFlatMap.java:349)
        at reactor.core.publisher.FluxMapFuseable$MapFuseableSubscriber.onNext(FluxMapFuseable.java:119)
        at reactor.core.publisher.FluxRange$RangeSubscription.slowPath(FluxRange.java:144)
        at reactor.core.publisher.FluxRange$RangeSubscription.request(FluxRange.java:99)
        at reactor.core.publisher.FluxMapFuseable$MapFuseableSubscriber.request(FluxMapFuseable.java:172)
        at reactor.core.publisher.FluxFlatMap$FlatMapMain.onSubscribe(FluxFlatMap.java:316)
        at
reactor.core.publisher.FluxMapFuseable$MapFuseableSubscriber.onSubscribe(FluxMapFuseable.java:94)
        at reactor.core.publisher.FluxRange.subscribe(FluxRange.java:68)
        at reactor.core.publisher.FluxMapFuseable.subscribe(FluxMapFuseable.java:67)
        at reactor.core.publisher.FluxFlatMap.subscribe(FluxFlatMap.java:98)
        at reactor.core.publisher.MonoSingle.subscribe(MonoSingle.java:58)
        at reactor.core.publisher.Mono.subscribeWith(Mono.java:2668)
        at reactor.core.publisher.Mono.subscribe(Mono.java:2629)
        at reactor.core.publisher.Mono.subscribe(Mono.java:2604)
        at reactor.core.publisher.Mono.subscribe(Mono.java:2582)
        at reactor.guide.GuideTests.debuggingCommonStacktrace(GuideTests.java:722)
```

Debug Mode

- For much more helpful stacktraces:

```
Hooks.onOperatorDebug();
```

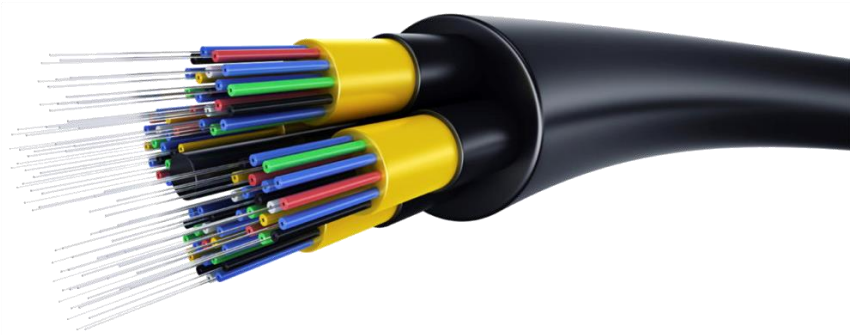
Debug Mode Stacktraces

```
java.lang.IndexOutOfBoundsException: Source emitted more than one item
        at reactor.core.publisher.MonoSingle$SingleSubscriber.onNext(MonoSingle.java:120)
        at reactor.core.publisher.FluxOnAssembly$OnAssemblySubscriber.onNext(FluxOnAssembly.java:314) (1)
...
(2)
...
        at reactor.core.publisher.Mono.subscribeWith(Mono.java:2668)
        at reactor.core.publisher.Mono.subscribe(Mono.java:2629)
        at reactor.core.publisher.Mono.subscribe(Mono.java:2604)
        at reactor.core.publisher.Mono.subscribe(Mono.java:2582)
        at reactor.guide.GuideTests.debuggingActivated(GuideTests.java:727)
        Suppressed: reactor.core.publisher.FluxOnAssembly$OnAssemblyException: (3)
Assembly trace from producer [reactor.core.publisher.MonoSingle] : (4)
        reactor.core.publisher.Flux.single(Flux.java:5335)
        reactor.guide.GuideTests.scatterAndGather(GuideTests.java:689)
        reactor.guide.GuideTests.populateDebug(GuideTests.java:702)
        org.junit.rules.TestWatcher$1.evaluate(TestWatcher.java:55)
        org.junit.rules.RunRules.evaluate(RunRules.java:20)
Error has been observed by the following operator(s): (5)
        |_      Flux.single(TestWatcher.java:55) (6)
```

Naming & Tagging operators

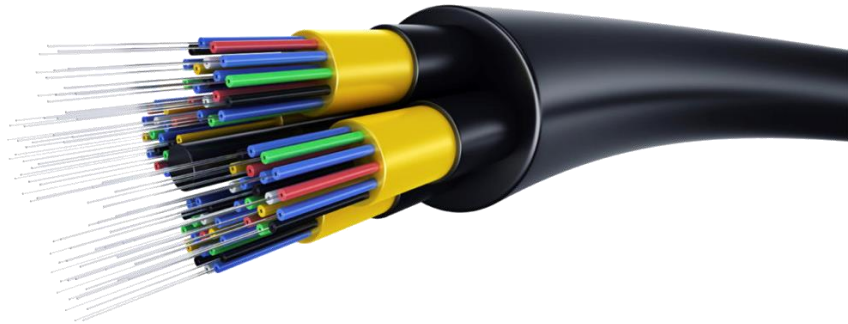
```
public Flux<Integer> nameAndTag() {
    return Flux.range(1, 10)
            .name("intRange")
            .tag("audience", "java.IL");
}
```

Hooks!

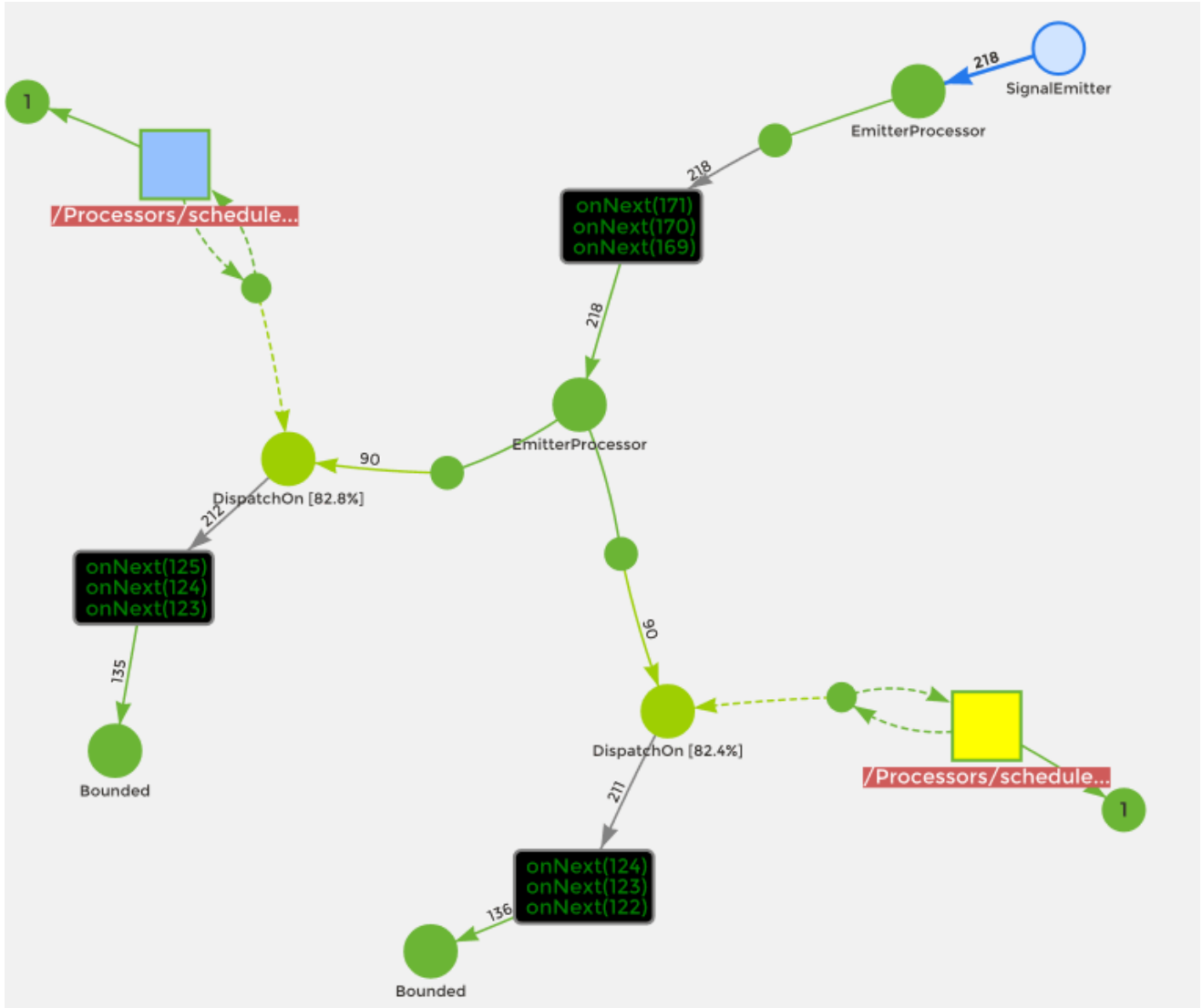


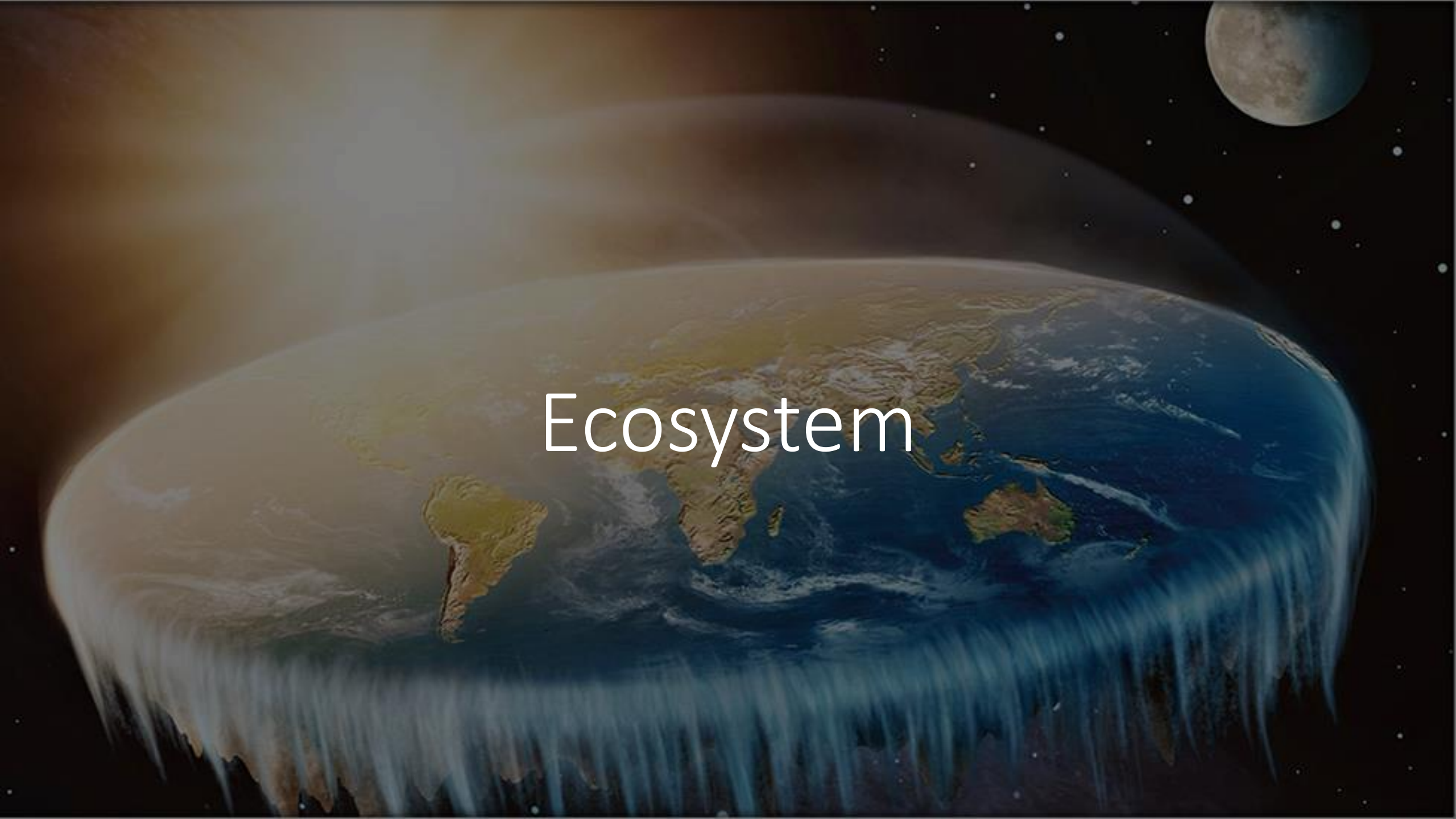
```
// last operator before subscribe
Hooks.onLastOperator(
    Operators.lift(scannable ->
        scannable.tags()
            .anyMatch(tag -> tag.getT1().contains("createdBy")),
                (scannable, subscriber) -> {
                    log.info("{} subscribed to {}", subscriber, scannable.name());
                    return subscriber;
}));
```

Hooks!



- onEachOperator
- onLastOperator
- onOperatorError
- onNextDropped
- onErrorDropped
- onNextDroppedFail
- onOperatorDebug
- resetOnEachOperator
- resetOnLastOperator
- resetOnOperatorError
- resetOnErrorDropped
- resetOnOperatorDebug
- resetOnNextDropped

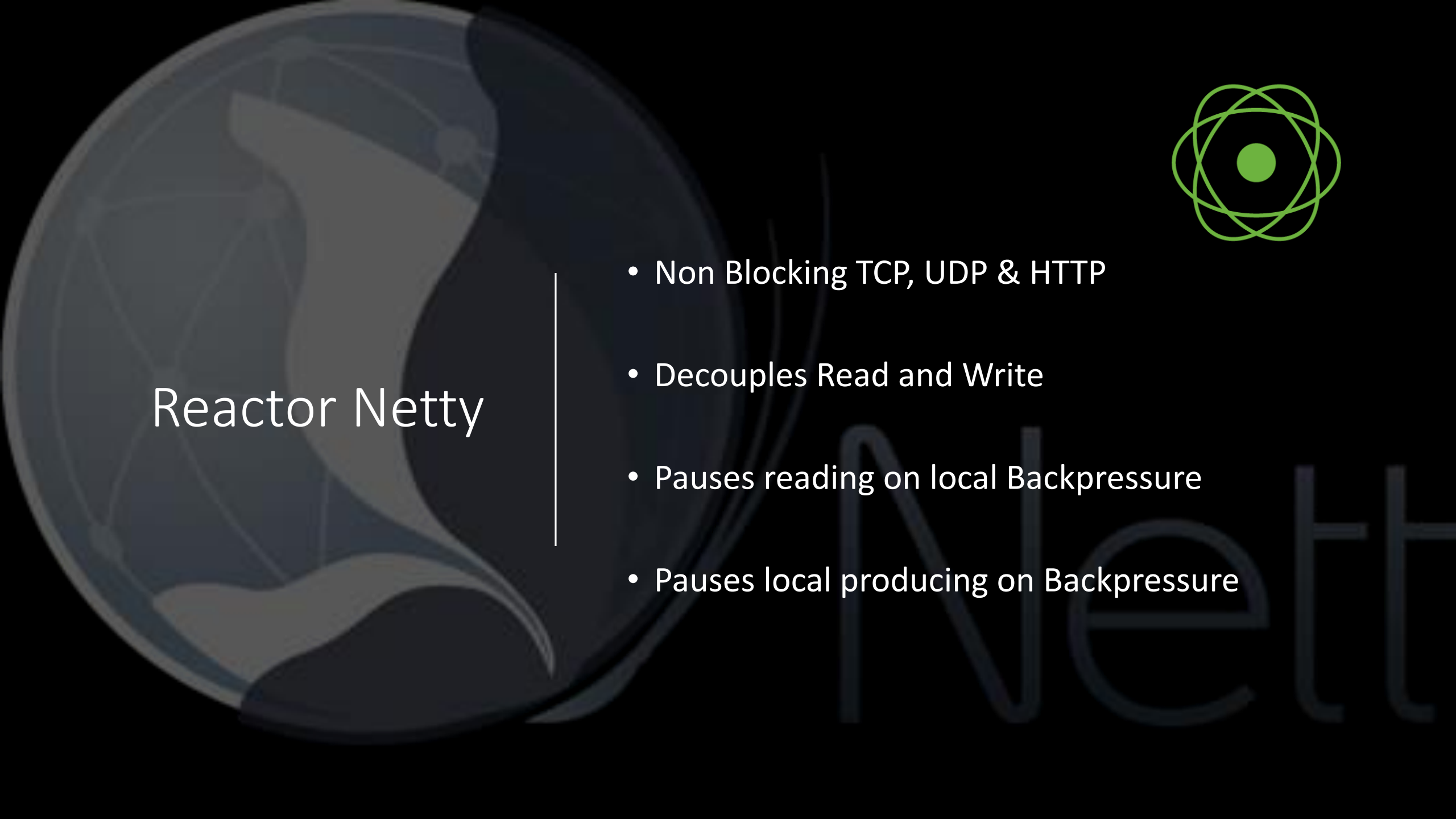

1
/Processors/schedule...
SignalEmitter
218
EmitterProcessor
218
onNext(171)
onNext(170)
onNext(169)
218
EmitterProcessor
90
DispatchOn [82.8%]
212
onNext(125)
onNext(124)
onNext(123)
135
Bounded
90
DispatchOn [82.4%]
/Processors/schedule...
1
211
onNext(124)
onNext(123)
onNext(122)
136
Bounded


Ecosystem

Kotlin

- Extensions to Flux & Mono
  - toMono(), toFlux(), test() etc.
- Nullability support

Kotlin



Reactor Netty

- Non Blocking TCP, UDP & HTTP
- Decouples Read and Write
- Pauses reading on local Backpressure
- Pauses local producing on Backpressure

Reactive Ecosystem

- Spring
  - Spring Webflux
  - Spring Data
  - Spring Data Cloud
  - Spring Integration
  - Spring Batch
- Kafka
- reactive-grpc

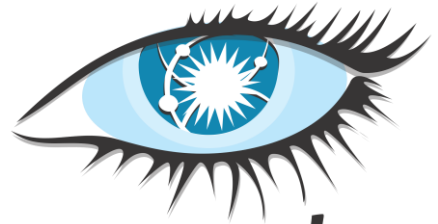
Reactive Data Access

- MongoDB
- Cassandra
- Redis
- CouchBase
- SQL: Slick / rdbc.io
    - ADBA - Asynchronous Database Access API

Couchbase

mongoDB

redis



cassandra

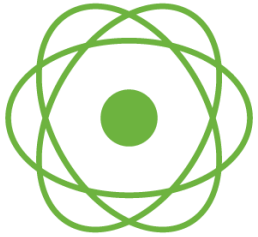
RSocket

- Binary protocol for use on byte stream transports (TCP, WebSockets, Aeron).
- Async message passing over a single connection
- Modes:
  - request/response (stream of 1)
  - request/stream (finite stream of many)
  - fire-and-forget (no response)
  - channel (bi-directional streams)
- Session resumption across different transport connections

http://rsocket.io/

Reactive Streams Alternatives

Reactive Streams



- Implemented by
  - **RxJava**
  - **Project Reactor**
  - **Akka Streams**
  - **Vert.x**
  - **Ratpack**

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

Reactive Libraries Implementations

Generation	Library	
0th	java.util.Observable	
1st	Rx.NET, RxJava	
2nd	RxJava 1.x	Limited backpressure
3rd	ProjectReactor, Akka Streams 2.x	Reactive streams, Actor fusion, Backpressure
4th	ProjectReactor 2.5, RxJava 2.x	Operator-fusion

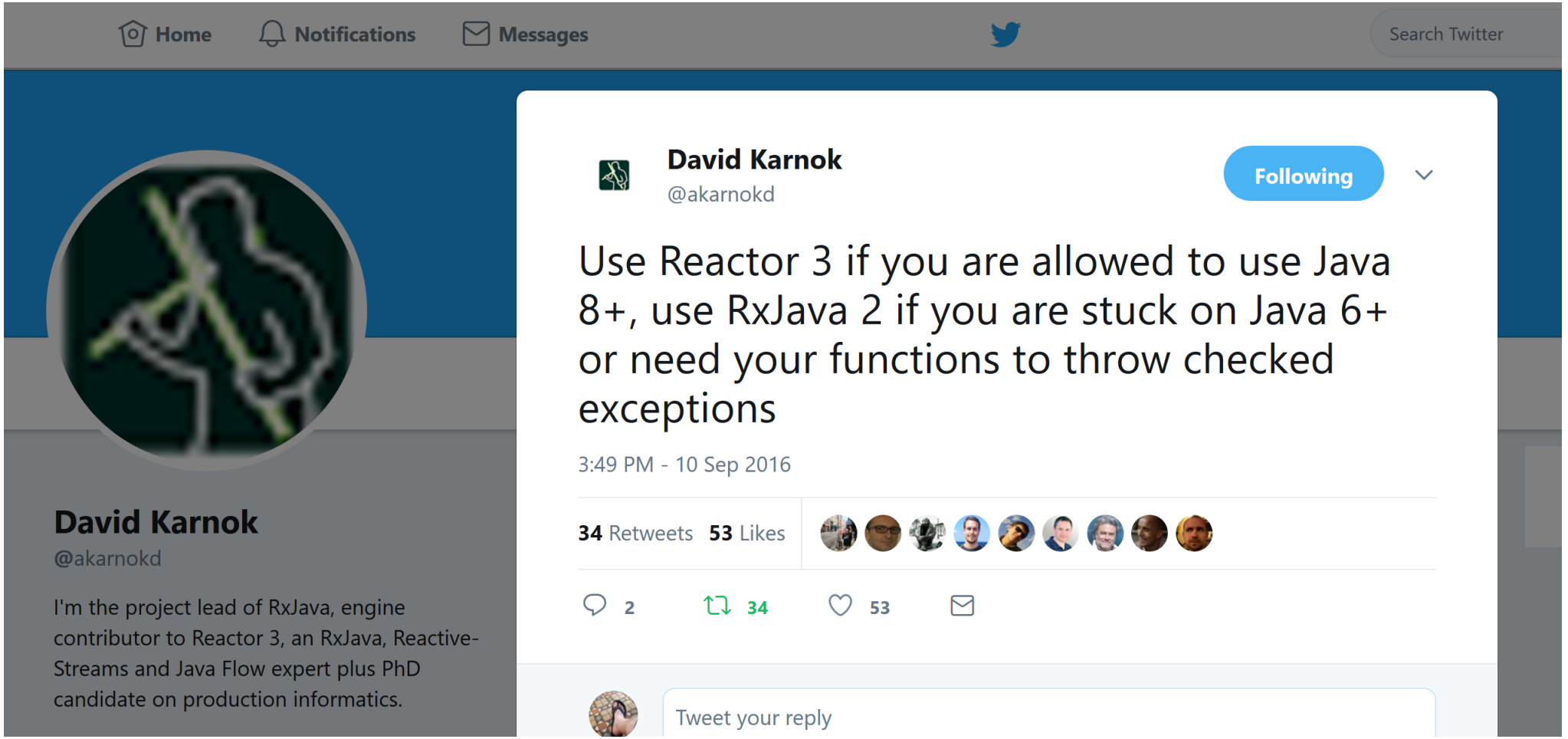
http://akarnokd.blogspot.co.il/2016/03/operator-fusion-part-1.html

Reactive Libraries Implementations

Library	0..n elements	0..1 elements
RxJava 1.x	`Observable`	`Single`, `Completable`
RxJava 2.x	`Observable`, `Flowable`	`Single`, `Completable`, `Maybe`
Akka Streams	`Source`, `Sink`, `Flow`	
Reactor	`Flux`	`Mono`

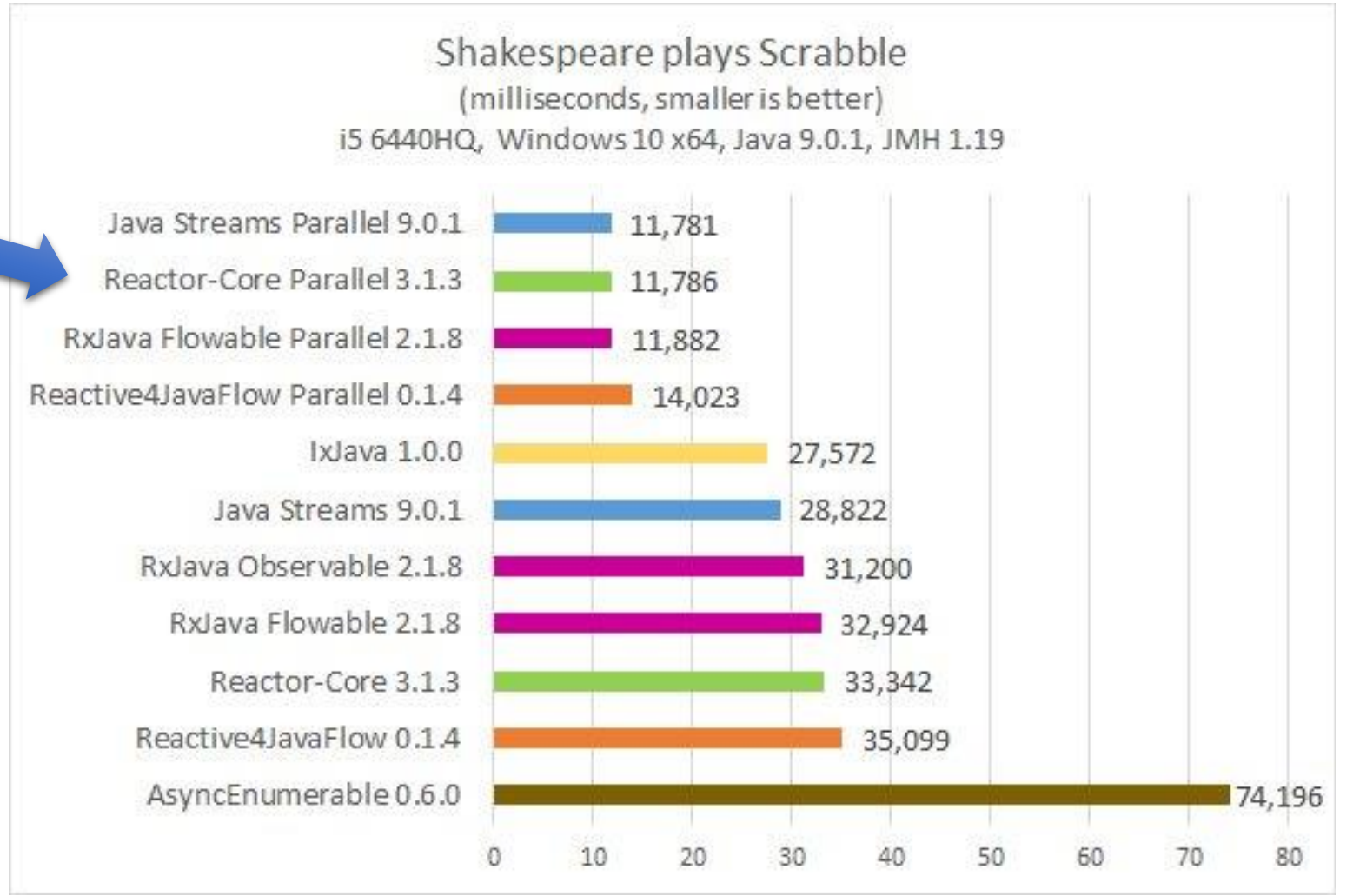
Async Monads in Java

Reactor	RxJava	Akka Streams	CompletableFuture	Future
block()	toBlocking()	toCompletionStage()	get()	get()
map()	map()	map()	thenApply()	¯_(ツ)_/¯
flatMap()	flatMap()	flatMapMerge()	thenCompose()	¯_(ツ)_/¯
doOnNext()	doOnNext()	runForeach()	thenAccept()	¯_(ツ)_/¯
doOnError()	doOnError()	recover()	exceptionally()	¯_(ツ)_/¯
reduce()	reduceWith()	reduce() fold()	thenCombine()	¯_(ツ)_/¯



https://twitter.com/akarnokd/status/774590596740685824

Shakespeare Scrabble Benchmarks (2018-01)



Interoperable

- Accept **Publisher** params
- Return specific types (Flux/Mono)

Interchangeable

- reactor.adapter.rxjava.RxJava2Adapter
  - `io.reactivex.Flowable<T> fluxToFlowable(Flux<T> source)`
  - `Flux<T> flowableToFlux(io.reactivex.Flowable<T> source)`
  - `Mono<Void> completableToMono(io.reactivex.Completable source)`
  - …
- reactor.adapter.akka.ActorScheduler

https://projectreactor.io/docs/adapter/release/api/reactor/adapter/rxjava/RxJava2Adapter.html

Code!

Didn’t Cover…

- ConnectableFlux
  - connect / autoConnect(n) / refCount(int, Duration)
- Processors/Sinks (Hot/Cold)
  - Direct, synchronous, asynchronous

Roadmap

- Nicer error handling
- Monitoring & Tracing
- More Spring reactive libraries
- reactor-netty
- RSocket

Learn More

- The reference documentation is great!
  - http://projectreactor.io/docs/core/release/reference/docs/index.html
- Reactor by Example [InfoQ]
  - https://www.infoq.com/articles/reactor-by-example
- David Karnok's (@akarnokd) blog: Advanced Reactive Java
  - http://akarnokd.blogspot.co.il/
- Code in this preso
  - https://github.com/liqweed/reactor-example-preso

Thank you!