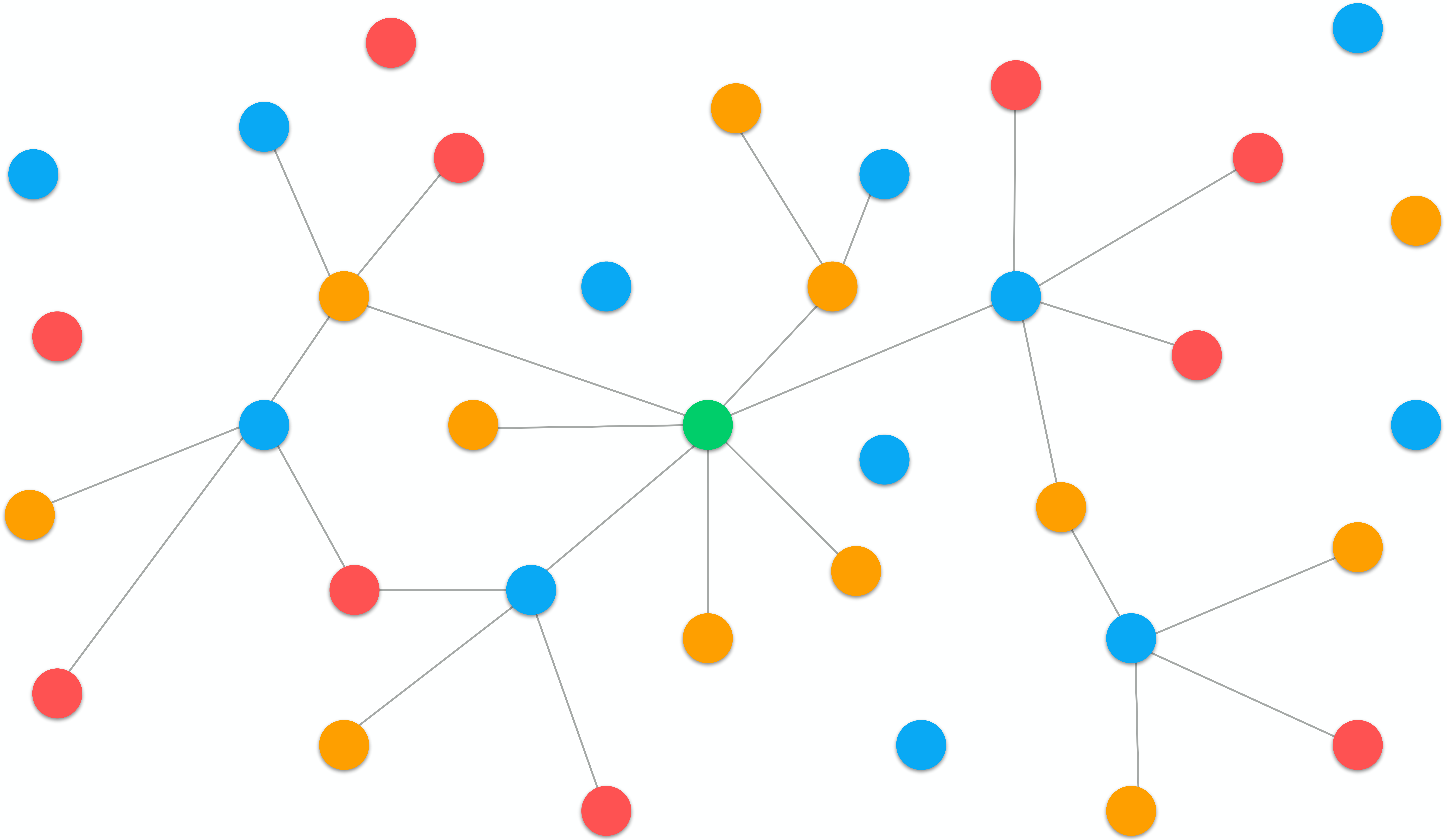
Got a question during this session? Post it on sli.do (#K100)

RxJava, RxJava 2, Reactor

State of the art of Reactive Streams on the JVM

David Wursteisen



Writing asynchronous code:

it sucks

Future

```
ExecutorService ex = Executors.newCachedThreadPool();
Future<String> future = ex.submit(() -> longProcessing());
String result = future.get();
```

Blocking call

Future

```
Future<?> future1 = /* ... */
Future<?> future2 = /* ... */
Future<?> future3 = /* ... */
Future<?> future4 = /* ... */
Future<?> future5 = /* ... */
```

Optimal Orchestration ?

Callback

```
RemoteService  service = buildRemoteService();

service.getUser(id -> {
    service.getData(id, data -> {
        service.getSomething(data, whut -> {
            service.neverEndingCallBack(whut, () -> {
            });
        });
    });
});
```

Callback Hell

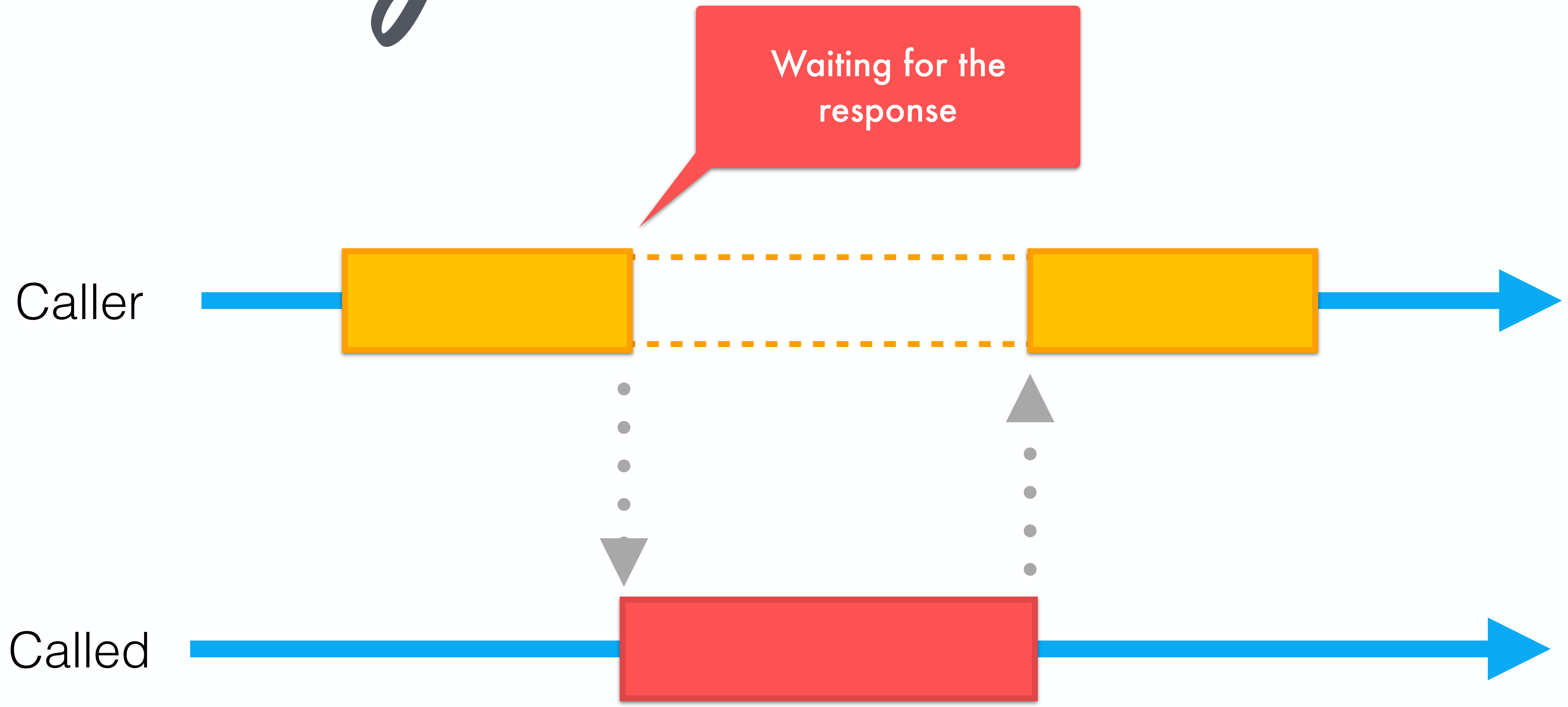
Relationship Status:

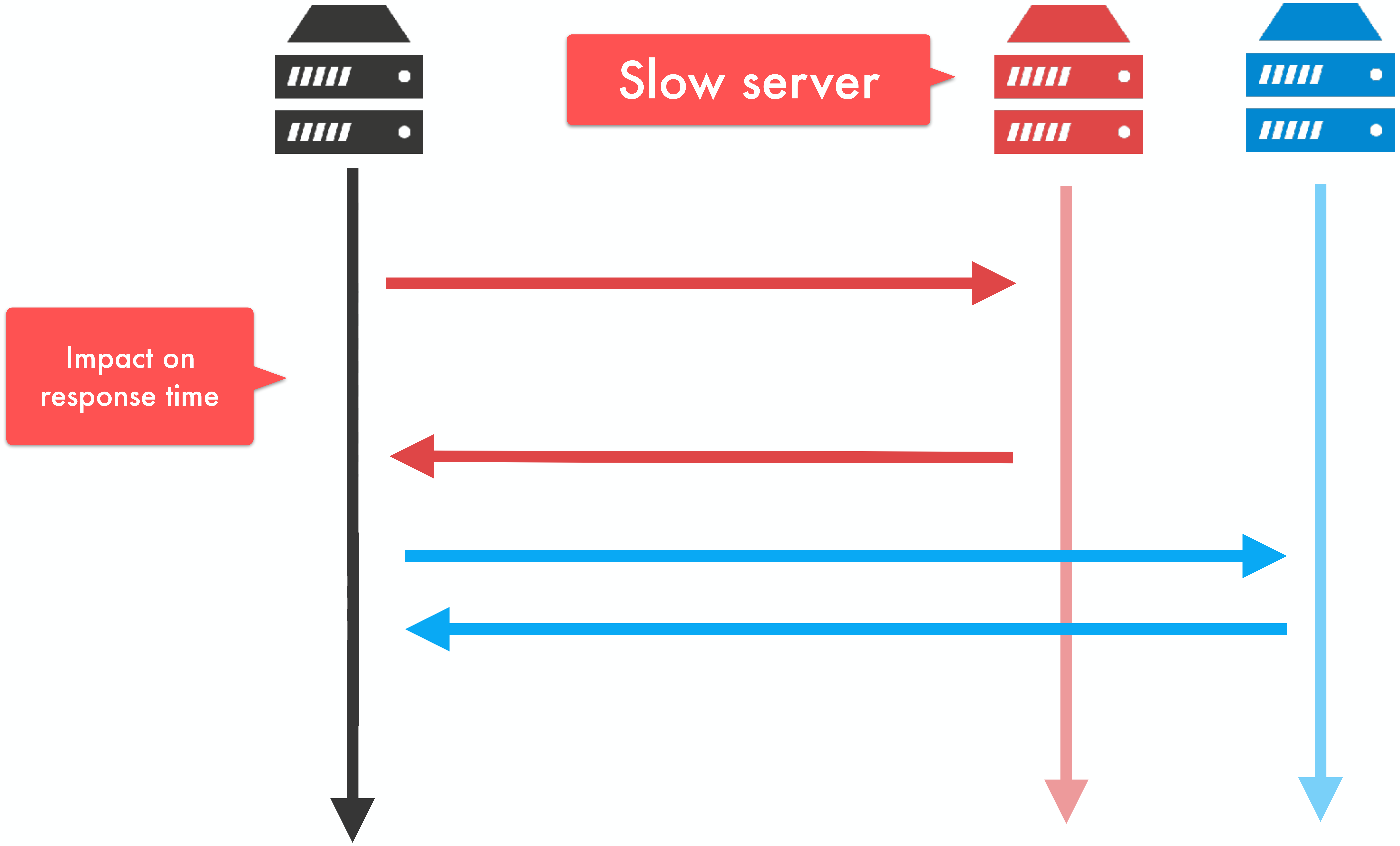
it's complicated

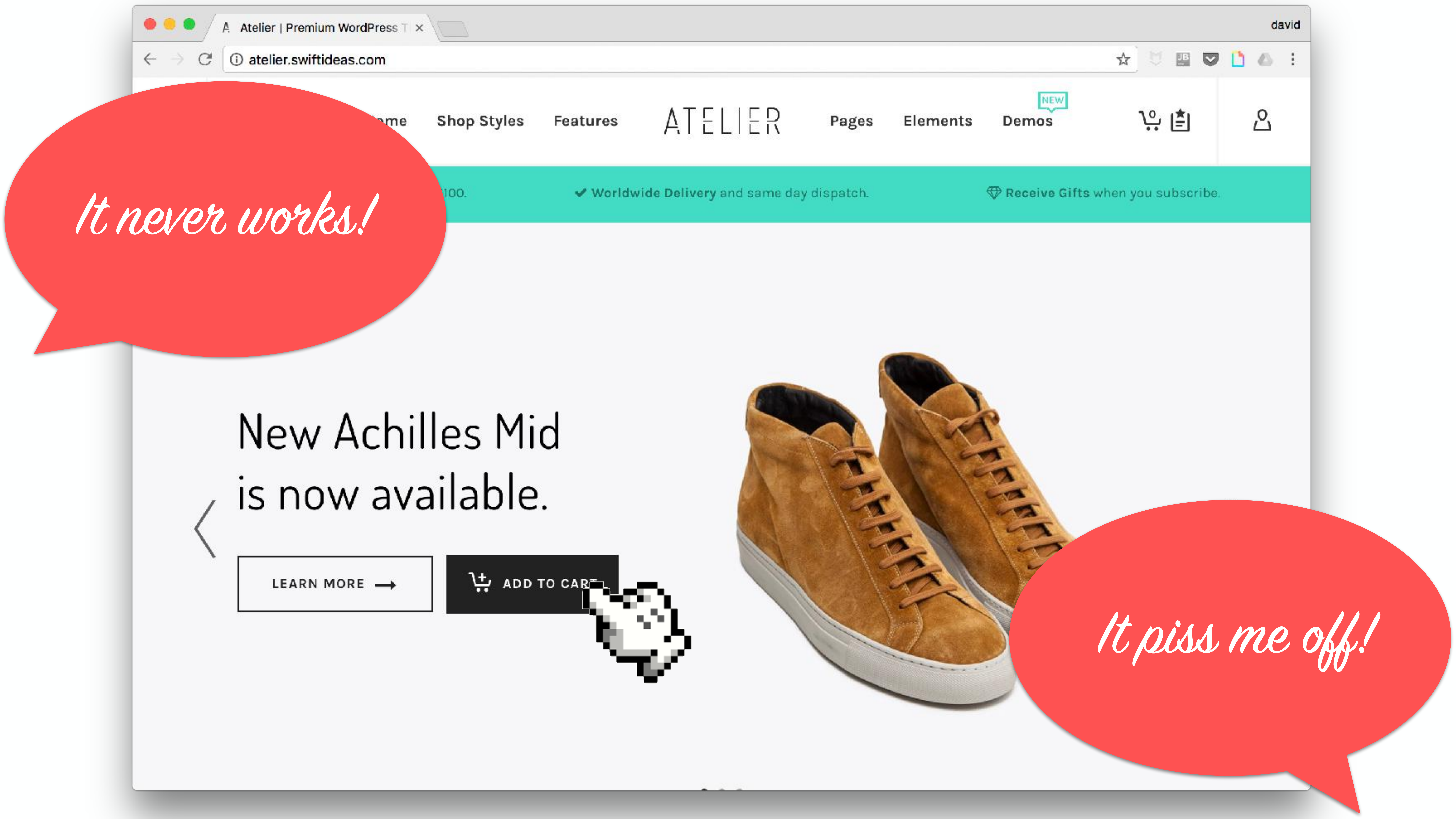
The problem

of synchronous code

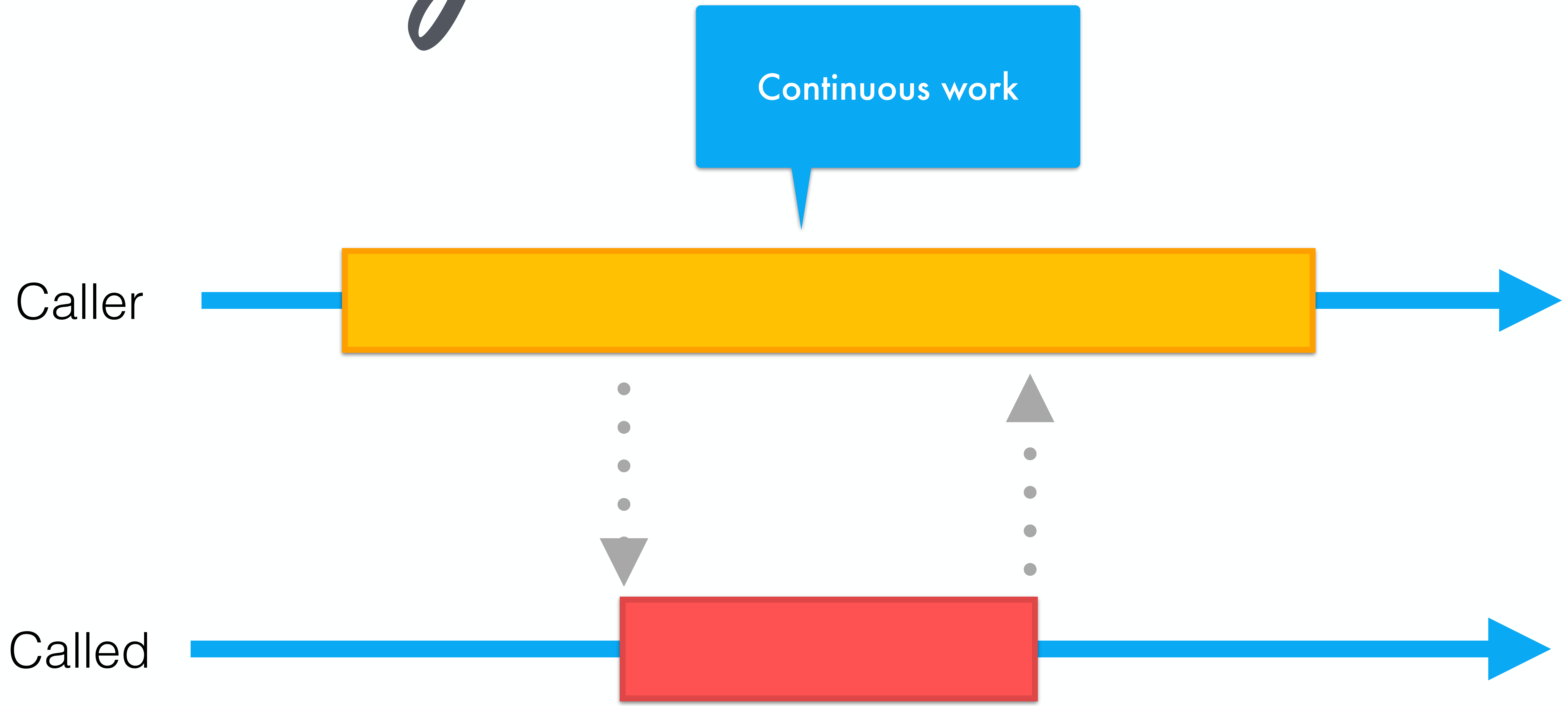
Synchronous




Slow server
Impact on response time

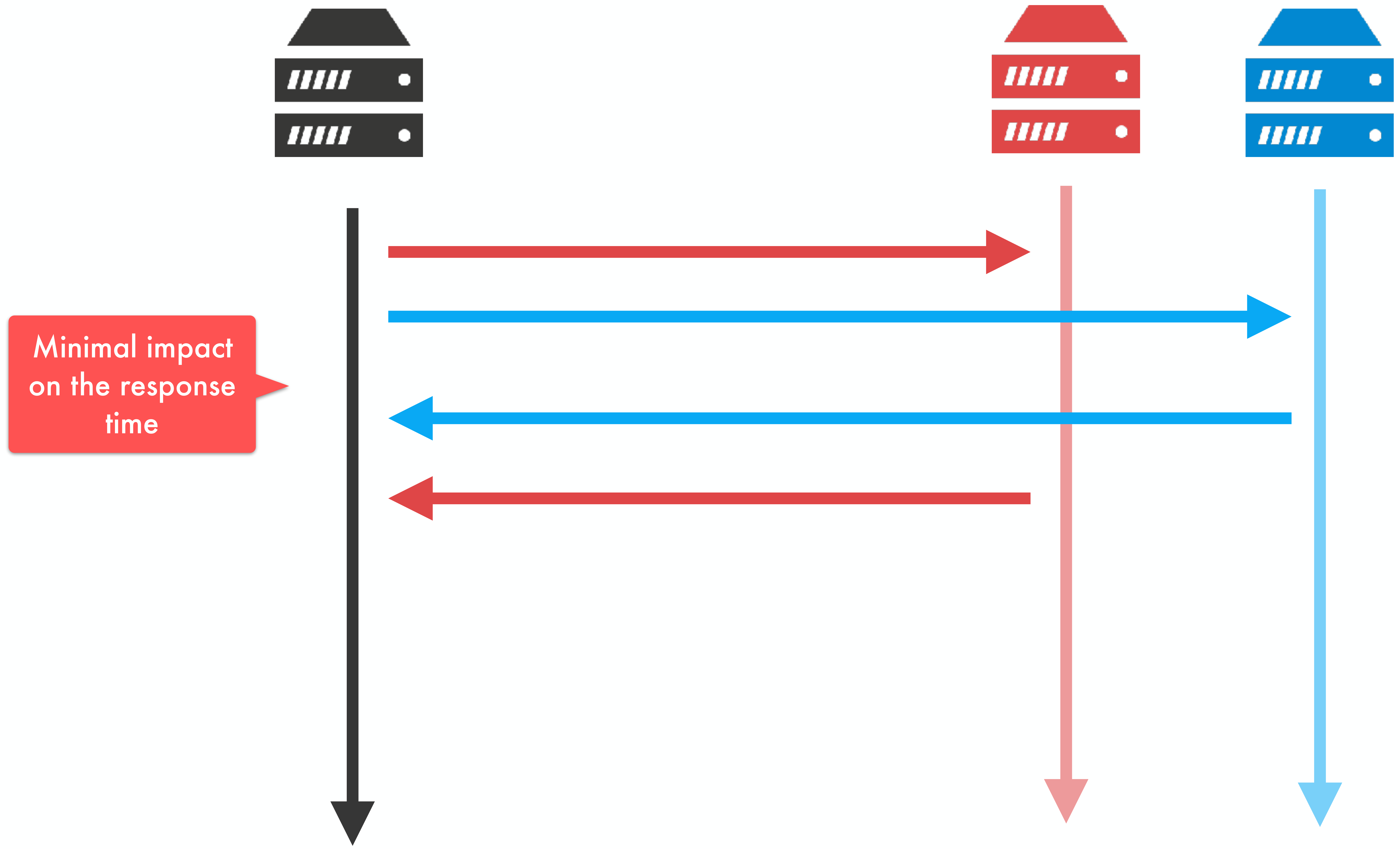

Atelier | Premium WordPress T
atelier.swiftideas.com
david
Shop Styles
Features
ATELIER
Pages
Elements
Demos
NEW
100.
Worldwide Delivery and same day dispatch.
Receive Gifts when you subscribe.
New Achilles Mid is now available.
LEARN MORE
ADD TO CART
It never works!
It piss me off!

Asynchronous



Asynchronous allow to take advantage of

the waiting time


Minimal impact on the response time

Write asynchronous code

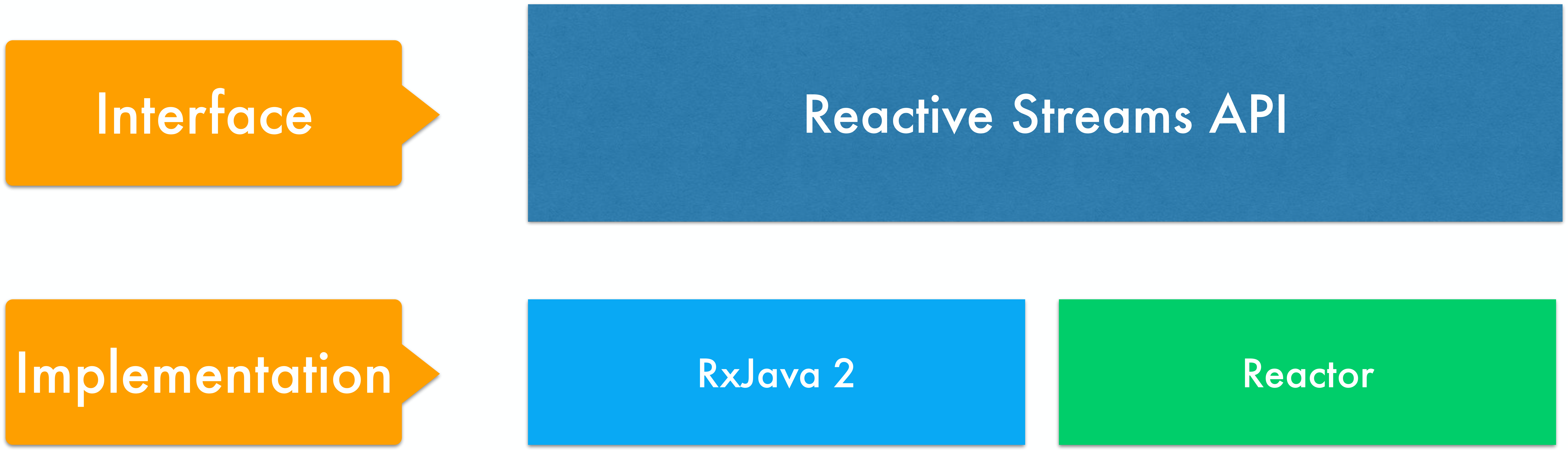
easily?

Emergence

of different approaches

Reactive Streams

Reactive Streams

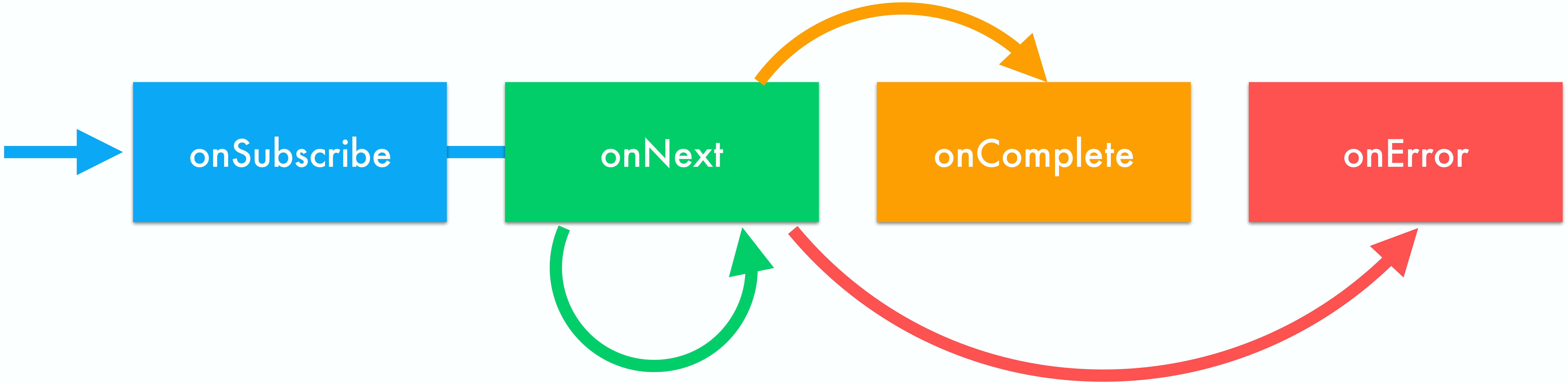


Reactive Streams API is

a bridge

between implementations

Reactive Streams contract



RxJava is *not compatible* with Reactive Streams

(You'll have to use an adapter: RxJavaReactiveStreams)

https://github.com/ReactiveX/RxJavaReactiveStreams

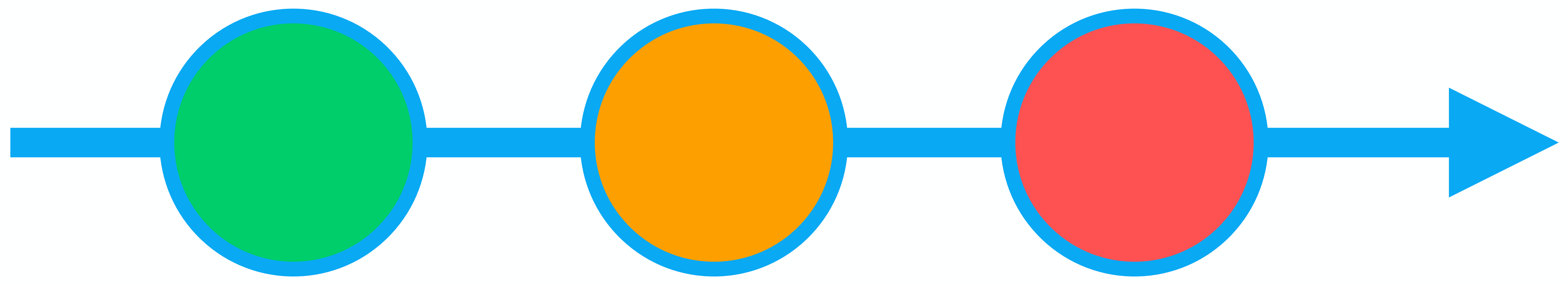
Reactive Streams	onNext * (onError | onComplete)
RxJava	onNext * (onError | onCompleted)

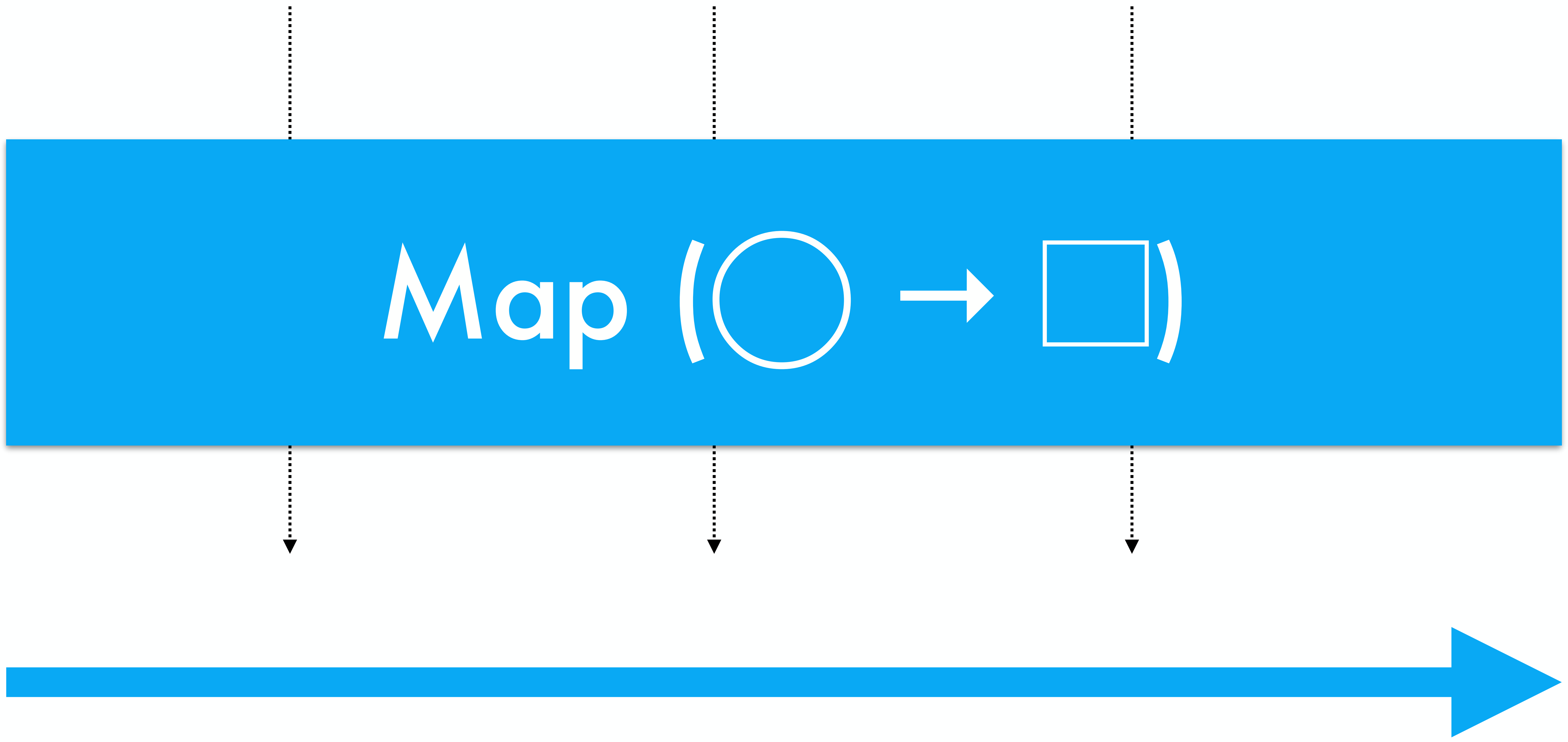
Reactive Streams	onNext * (onError | onComplete)
RxJava	onNext * (onError | onCompleted)

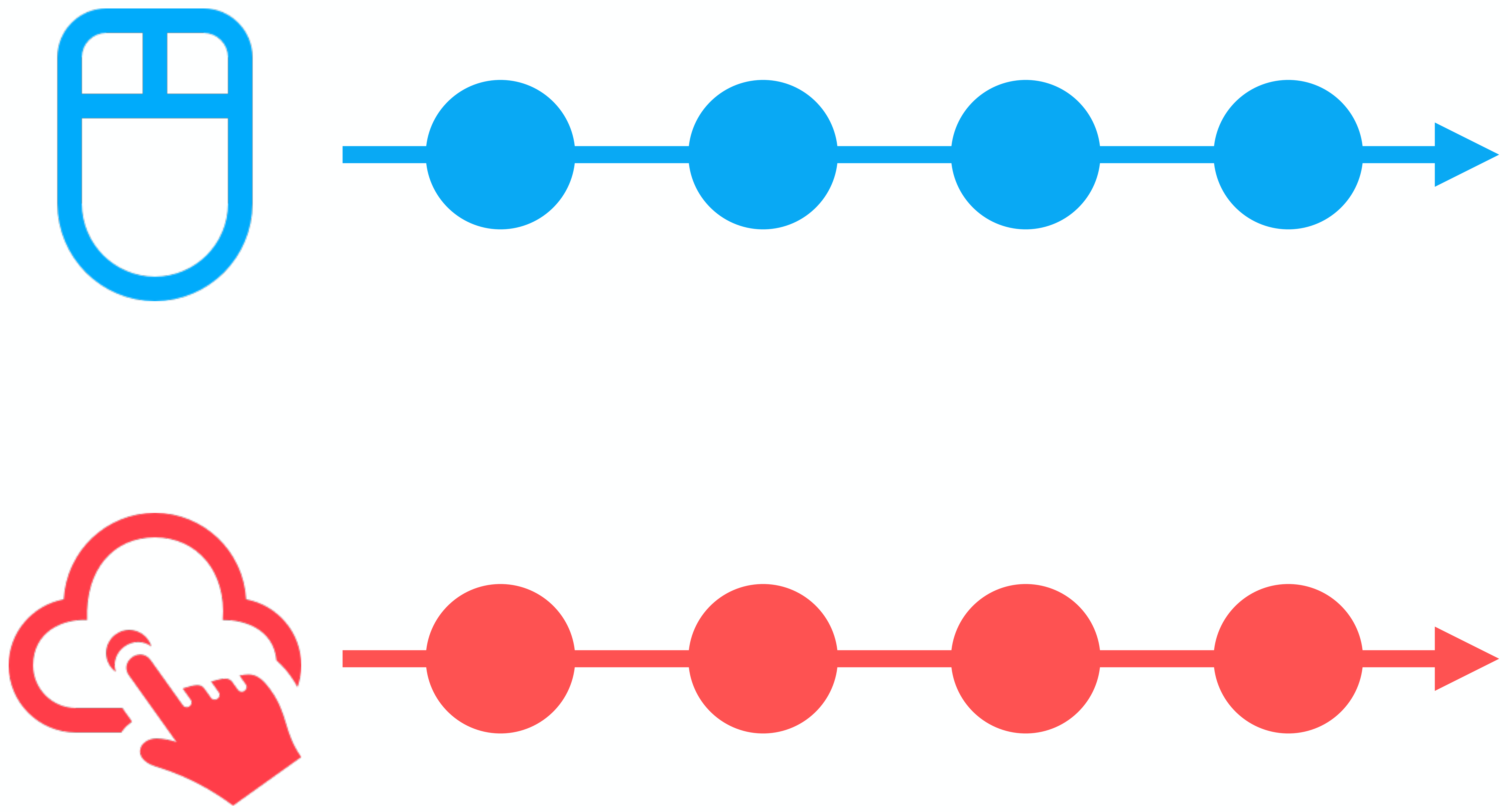
Different name

Common approach

API to handle events synchronously or asynchronously through a flow of events




Map (◯ → □)



```
remoteApi.people(1).flatMap(luke -> {

    Observable<String> vehicles = Observable.from(luke.getVehiclesIds())
            .flatMap(remoteApi::vehicle)
            .map(vehicle -> luke.getName() + " can drive " + vehicle.getName());

    Observable<String> starships = Observable.from(luke.getStarshipsIds())
            .flatMap(remoteApi::starship)
            .map(starship -> luke.getName() + " can fly with " + starship.getName());

    return Observable.merge(vehicles, starships);

}).subscribe(System.out::println);
```

```
remoteApi.people(1).flatMap(luke -> {

    Observable<String> vehicles = Observable.from(luke.getVehiclesIds())
            .flatMap(remoteApi::vehicle)
            .map(vehicle -> luke.getName() + " can drive " + vehicle.getName());

    Observable<String> starships = Observable.from(luke.getStarshipsIds())
            .flatMap(remoteApi::starship)
            .map(starship -> luke.getName() + " can fly with " + starship.getName());

    return Observable.merge(vehicles, starships);

}).subscribe(System.out::println);
```

Push of the result

Get Luke's vehicles

Get Luke's starships

```
remoteApi.people(1).flatMap(luke -> {

    Observable<String> vehicles = Observable.from(luke.getVehiclesIds())
            .flatMap(remoteApi::vehicle)
            .map(vehicle -> luke.getName() + " can drive " + vehicle.getName());

    Observable<String> starships = Observable.from(luke.getStarshipsIds())
            .flatMap(remoteApi::starship)
            .map(starship -> luke.getName() + " can fly with " + starship.getName());

    return Observable.merge(vehicles, starships);

}).subscribe(System.out::println);
```

```
remoteApi.people(1).flatMap(luke -> {

    Observable<String> vehicles = Observable.from(luke.getVehiclesIds())
            .flatMap(remoteApi::vehicle)
            .map(vehicle -> luke.getName() + " can drive " + vehicle.getName());

    Observable<String> starships = Observable.from(luke.getStarshipsIds())
            .flatMap(remoteApi::starship)
            .map(starship -> luke.getName() + " can fly with " + starship.getName());

    return Observable.merge(vehicles, starships);

}).subscribe(System.out::println);
```

Merge of two flows

```
remoteApi.people(1).flatMap(luke -> {

    Observable<String> vehicles = Observable.from(luke.getVehiclesIds())
            .flatMap(remoteApi::vehicle)
            .map(vehicle -> luke.getName() + " can drive " + vehicle.getName());

    Observable<String> starships = Observable.from(luke.getStarshipsIds())
            .flatMap(remoteApi::starship)
            .map(starship -> luke.getName() + " can fly with " + starship.getName());

    return Observable.merge(vehicles, starships);

}).subscribe(System.out::println);
```

Really execute the code

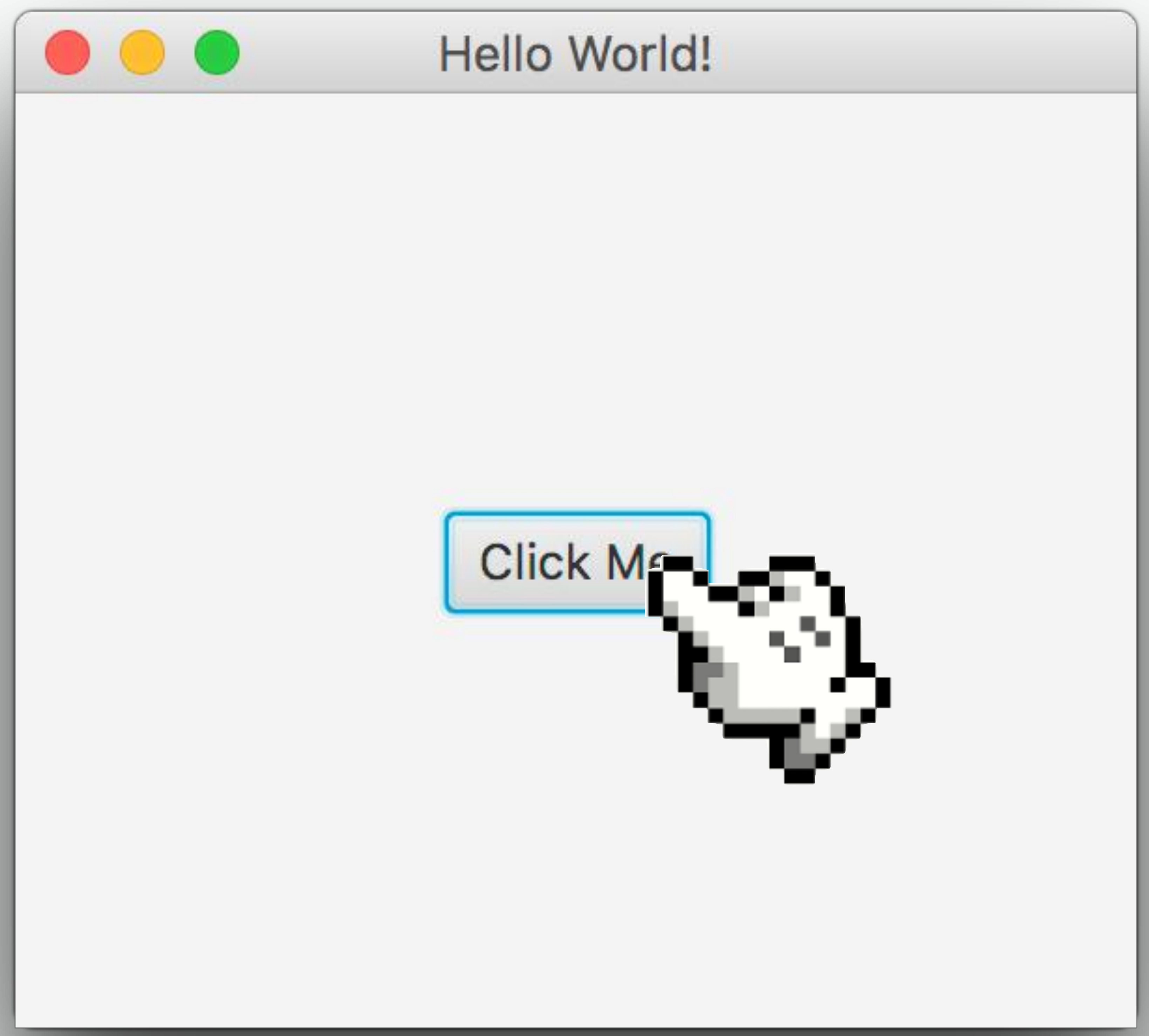
Flow of events

```
remoteApi.people(1).flatMap(luke -> {

    Observable<String> vehicles = Observable.from(luke.getVehiclesIds())
            .flatMap(remoteApi::vehicle)
            .map(vehicle -> luke.getName() + " can drive " + vehicle.getName());

    Observable<String> starships = Observable.from(luke.getStarshipsIds())
            .flatMap(remoteApi::starship)
            .map(starship -> luke.getName() + " can fly with " + starship.getName());

    return Observable.merge(vehicles, starships);

}).subscribe(System.out::println);
```

Flow of events

Flow of events

```
remoteApi.people(1).flatMap(luke -> {

    Observable<String> vehicles = Observable.from(luke.getVehiclesIds())
            .flatMap(remoteApi::vehicle)
            .map(vehicle -> luke.getName() + " can drive " + vehicle.getName());

    Observable<String> starships = Observable.from(luke.getStarshipsIds())
            .flatMap(remoteApi::starship)
            .map(starship -> luke.getName() + " can fly with " + starship.getName());

    return Observable.merge(vehicles, starships);

}).subscribe(System.out::println);
```


Hello World!
Click Me

```
Button btn = new Button();
btn.setText("Click Me");

JavaFx.fromClick(btn)
        .observeOn(Schedulers.io())
        .switchMap(evt -> remoteApi.getData())
        .observeOn(javaFx())
        .doOnNext(value -> btn.setText("Data: " + value))
        .subscribe();
```

```
Button btn = new Button();
btn.setText("Click Me");

JavaFx.fromClick(btn)                                          // Observable<Event>
        .observeOn(Schedulers.io())
        .switchMap(evt -> remoteApi.getData())
        .observeOn(javaFx())
        .doOnNext(value -> btn.setText("Data: " + value))
        .subscribe();
```

Listen for clicks

```
Button btn = new Button();
btn.setText("Click Me");

JavaFx.fromClick(btn)                              // Observable<Event>
    .observeOn(Schedulers.io())
    .switchMap(evt -> remoteApi.getData())
    .observeOn(javaFx())
    .doOnNext(value -> btn.setText("Data: " + value))
    .subscribe();
```

Execution context switch

```
Button btn = new Button();
btn.setText("Click Me");

JavaFx.fromClick(btn)                                  // Observable<Event>
    .observeOn(Schedulers.io())
    .switchMap(evt -> remoteApi.getData())    // Observable<Data>
    .observeOn(javaFx())
    .doOnNext(value -> btn.setText("Data: " + value))
    .subscribe();
```

Asynchronous call to a web service

```
Button btn = new Button();
btn.setText("Click Me");

JavaFx.fromClick(btn)                                   // Observable<Event>
        .observeOn(Schedulers.io())
        .switchMap(evt -> remoteApi.getData())    // Observable<Data>
        .observeOn(javaFx())
        .doOnNext(value -> btn.setText("Data: " + value))
        .subscribe();
```

Execution context switch

```
Button btn = new Button();
btn.setText("Click Me");

JavaFx.fromClick(btn)                                   // Observable<Event>
        .observeOn(Schedulers.io())
        .switchMap(evt -> remoteApi.getData())          // Observable<Data>
        .observeOn(javaFx())
        .doOnNext(value -> btn.setText("Data: " + value))
        .subscribe();
```

Update on the UI

```
Button btn = new Button();
btn.setText("Click Me");

JavaFx.fromClick(btn)                                   // Observable<Event>
        .observeOn(Schedulers.io())
        .switchMap(evt -> remoteApi.getData())          // Observable<Data>
        .observeOn(javaFx())
        .doOnNext(value -> btn.setText("Data: " + value))
        .subscribe();
```

Flow of events

Thanks to RxJava

and Reactor*...*

Writing asynchronous code:

~~it sucks~~

Once upon a time...

Creation of Reactive Extensions

Microsoft

Creation of RxJava

NETFLIX

facebook

Resumption of RxJava & RxJava 2

Work on Reactor

Creation of Reactor

Pivotal™

RxJava

is a

proved

technologie

Reactor

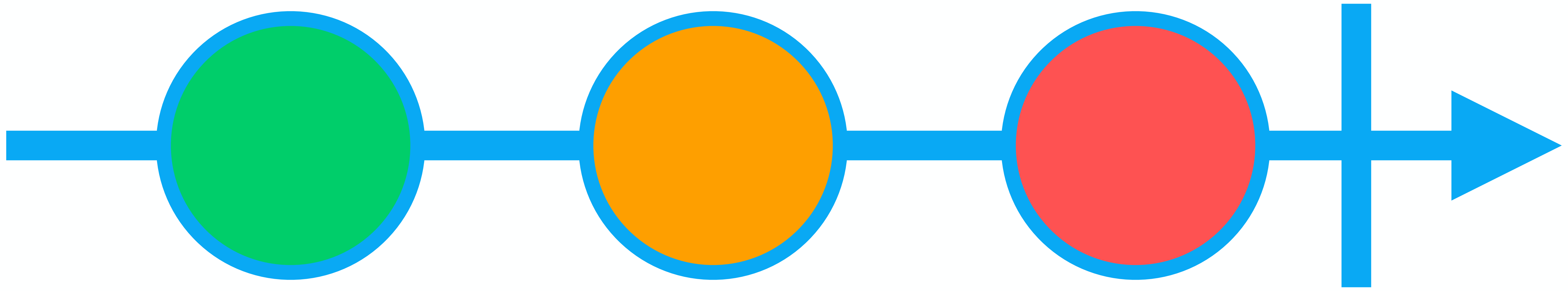
benefits from the experience of

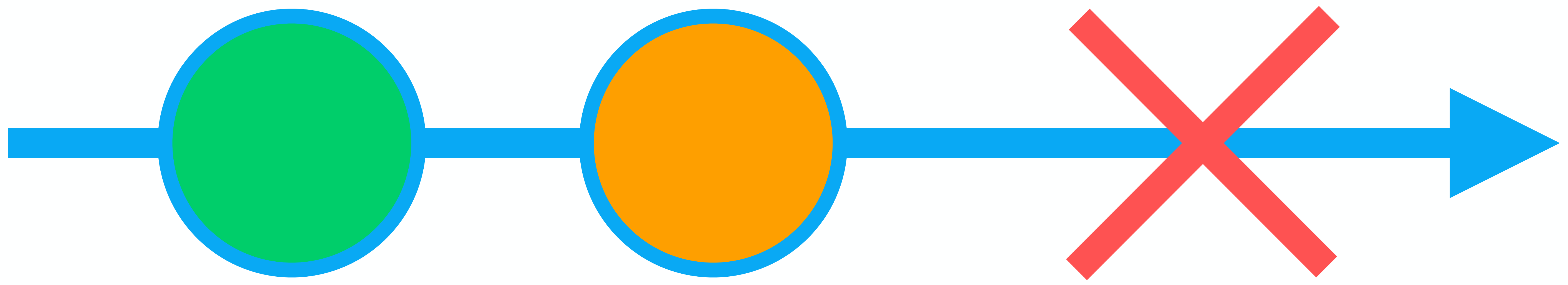
RxJava

(and vice versa)

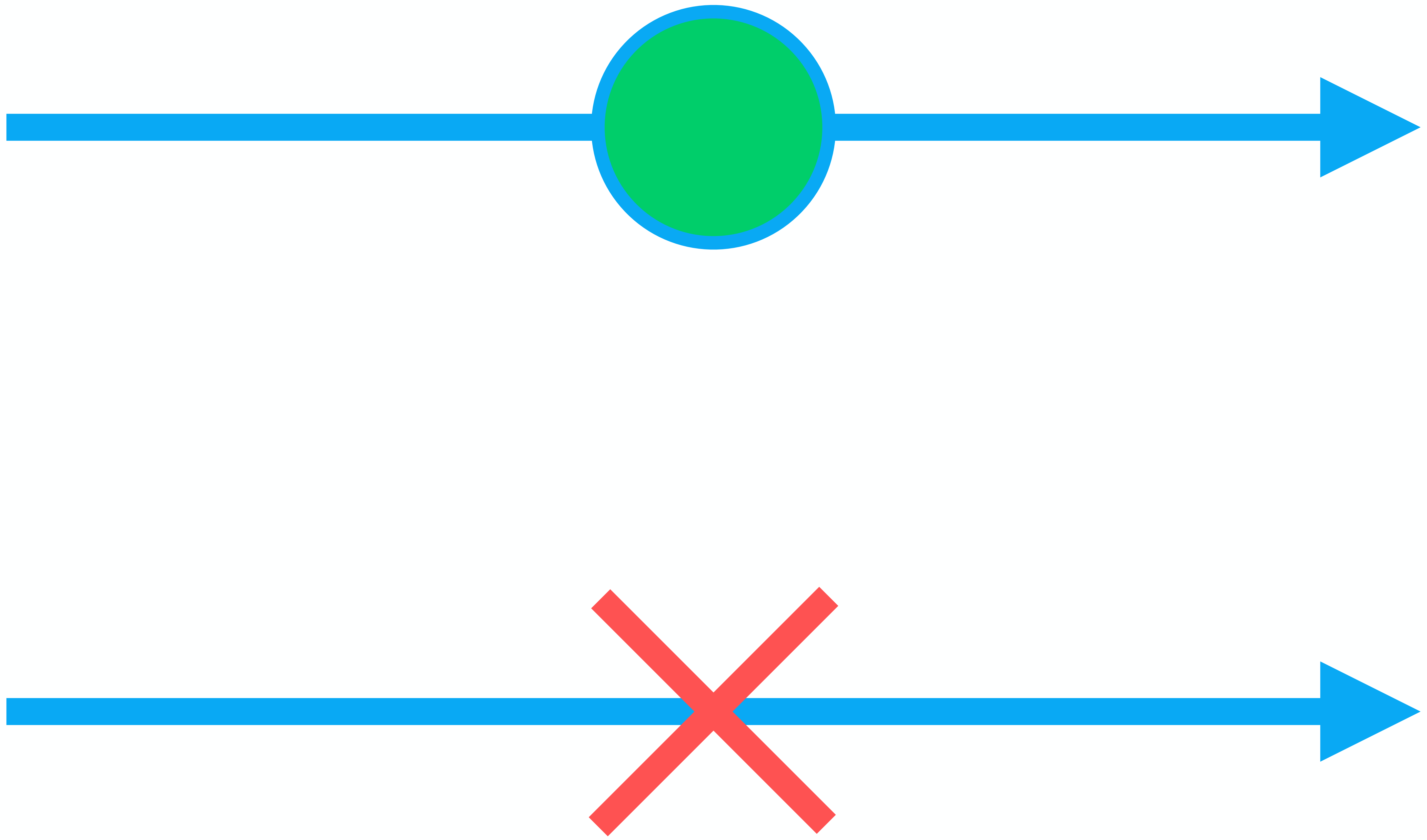
Object types

Observable

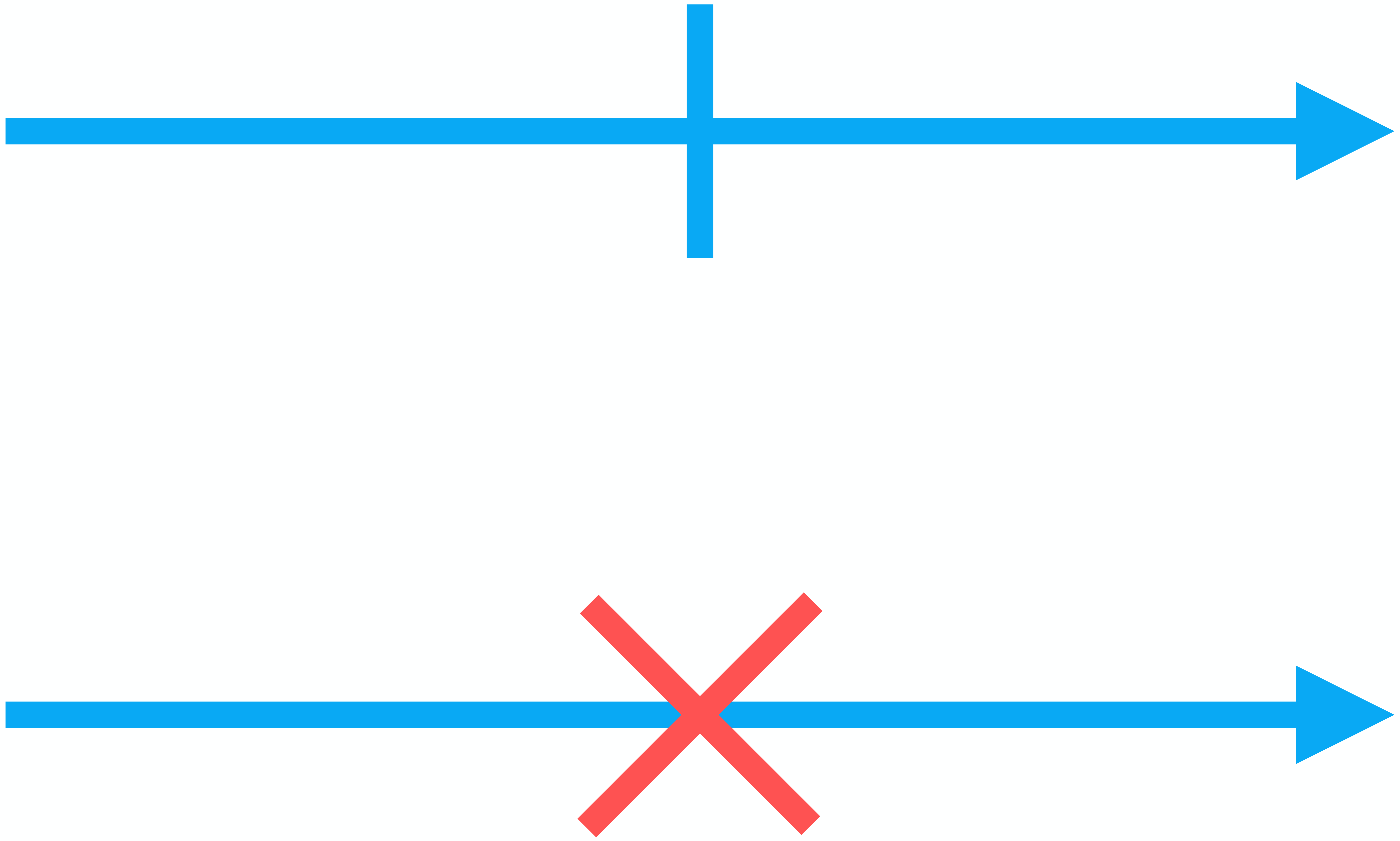




Single



Completable



RxJava

	Contrat	Backpressure
Observable	[N]	Yes
Single	[1]	No
Completable	[0]	No

Added afterward

Web service call

Background process

Listen a websocket, for each received command, compose a response by calling 3 different webservices, then execute 2 jobs sequentially ?

Listen a websocket, for each received command, compose a response by calling 3 different webservices, then execute 2 jobs sequentially ?

Listen a websocket, for each received command, compose a response by calling 3 different webservices, then execute 2 jobs sequentially ?

Listen a websocket, for each received command, compose a response by calling 3 different webservices, then execute 2 jobs sequentially ?

```
websocket("/topics/cmd")
        .observeOn(Schedulers.io())
        .switchMap(cmd ->
                Single.zip(
                        api.getActions(),
                        api.getScore(),
                        api.getUserData(),
                        this::composeResult).toObservable())
        .observeOn(Schedulers.computation())
        .concatMap(result -> updateDb(result).andThen(getLastResults()))
        .subscribe(last -> System.out.println("last results -> " + last));
```

Listen a websocket

Observable

```
websocket("/topics/cmd")
        .observeOn(Schedulers.io())
        .switchMap(cmd ->
                Single.zip(
                        api.getActions(),
                        api.getScore(),
                        api.getUserData(),
                        this::composeResult).toObservable())
        .observeOn(Schedulers.computation())
        .concatMap(result -> updateDb(result).andThen(getLastResults()))
        .subscribe(last -> System.out.println("last results -> " + last));
```

Webservices composition

Single

```
websocket("/topics/cmd")
        .observeOn(Schedulers.io())
        .switchMap(cmd ->
                Single.zip(
                        api.getActions(),
                        api.getScore(),
                        api.getUserData(),
                        this::composeResult).toObservable())
        .observeOn(Schedulers.computation())
        .concatMap(result -> updateDb(result).andThen(getLastResults()))
        .subscribe(last -> System.out.println("last results -> " + last));
```

```
websocket("/topics/cmd")
        .observeOn(Schedulers.io())
        .switchMap(cmd ->
                Single.zip(
                        api.getActions(),
                        api.getScore(),
                        api.getUserData(),
                        this::composeResult).toObservable())
        .observeOn(Schedulers.computation())
        .concatMap(result -> updateDb(result).andThen(getLastResults()))
        .subscribe(last -> System.out.println("last results -> " + last));
```

Completable

2 jobs executions

Listen a websocket

Observable

Webservices composition

Single

2 jobs executions

Completable

```
websocket("/topics/cmd")
        .observeOn(Schedulers.io())
        .switchMap(cmd ->
                Single.zip(
                        api.getActions(),
                        api.getScore(),
                        api.getUserData(),
                        this::composeResult).toObservable())
        .observeOn(Schedulers.computation())
        .concatMap(result -> updateDb(result).andThen(getLastResults()))
        .subscribe(last -> System.out.println("last results -> " + last));
```

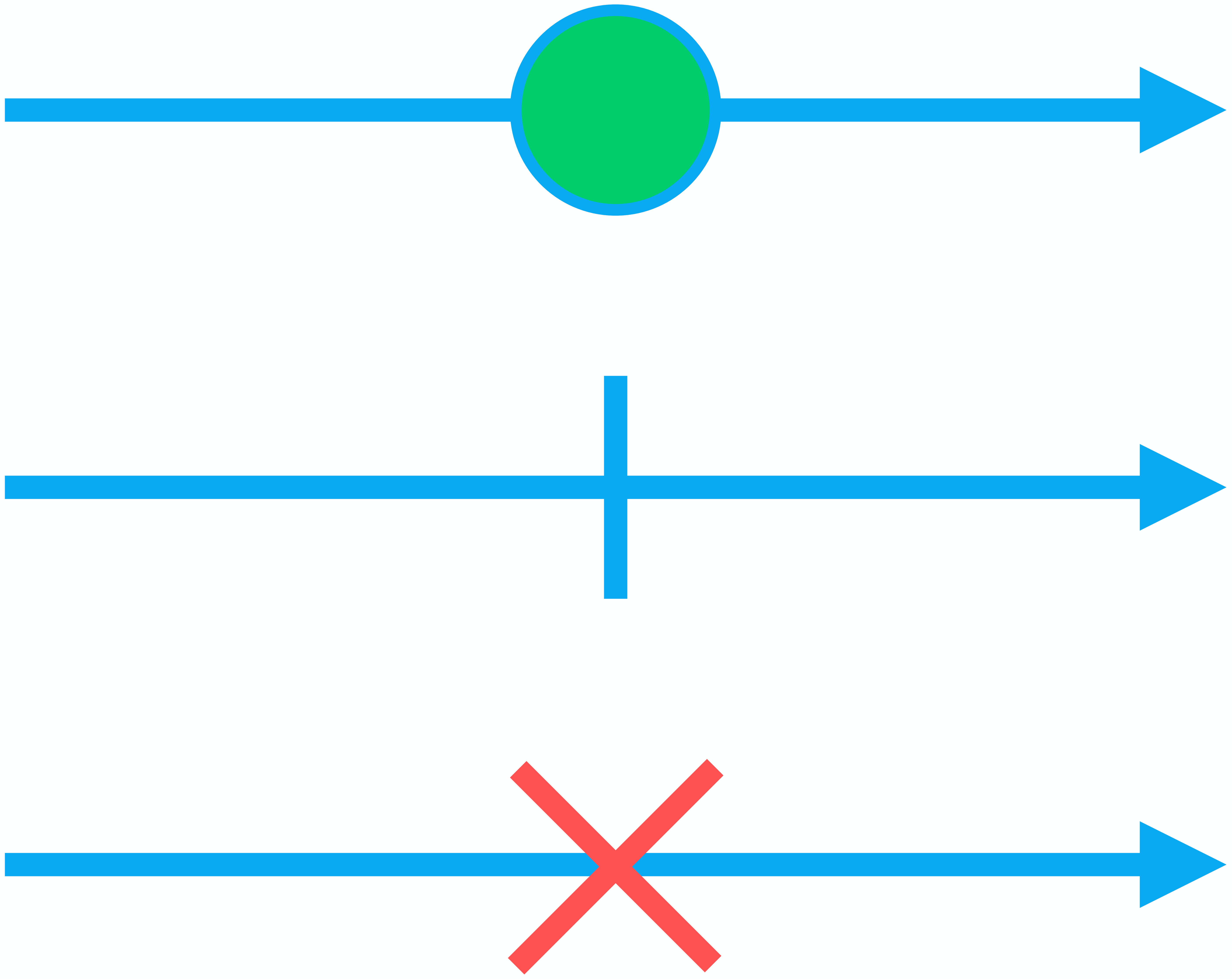
RxJava 2

	Contrat	Backpressure
Observable	[N]	No
Single	[1]	No
Completable	[0]	No
Maybe	[0|1]	No

Close to Java 8 Optional

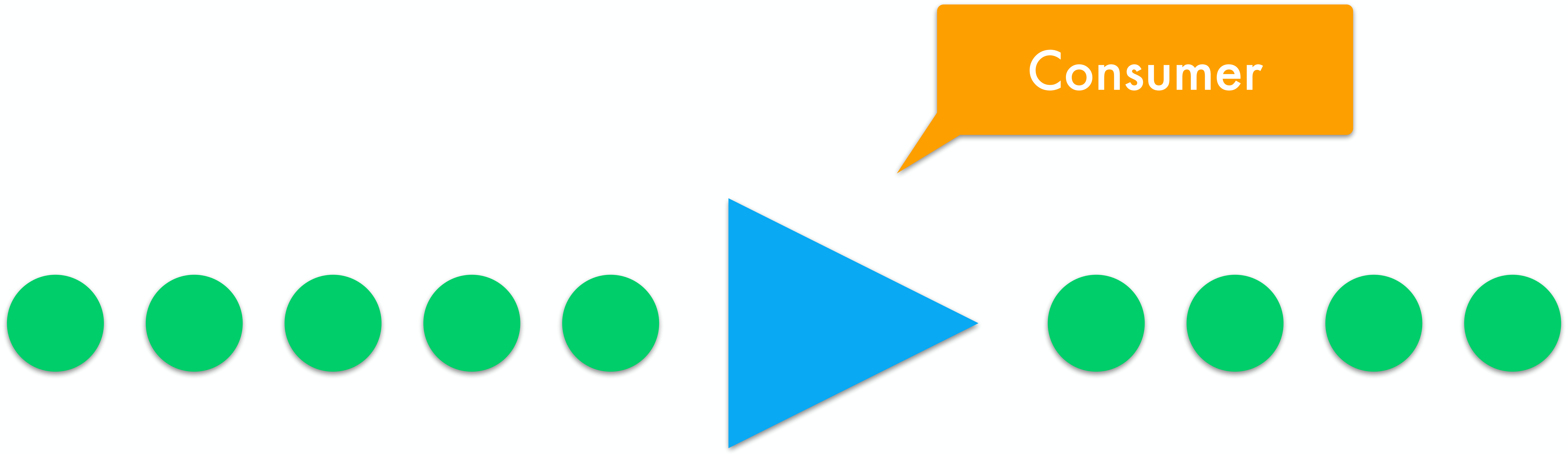
New!

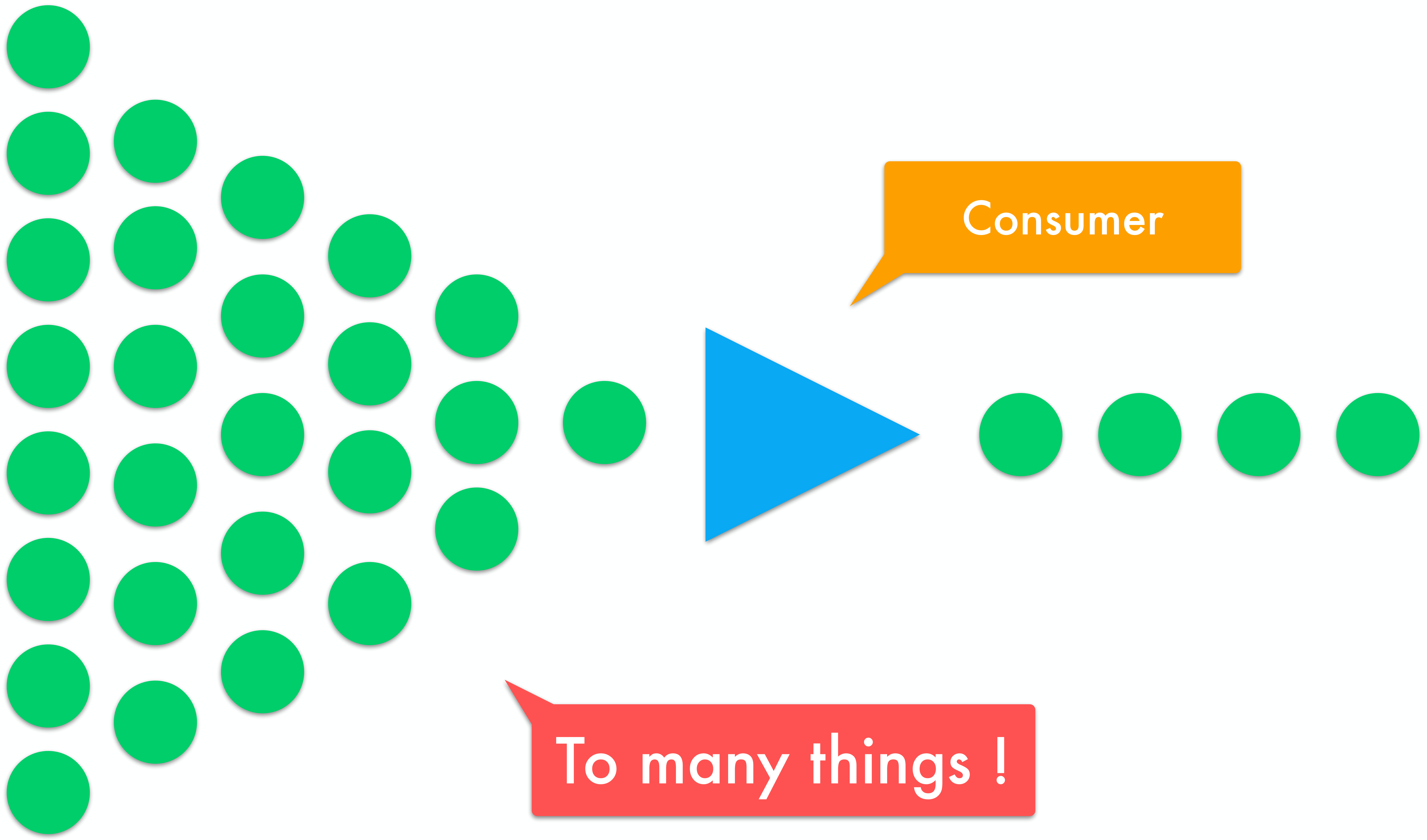
Maybe

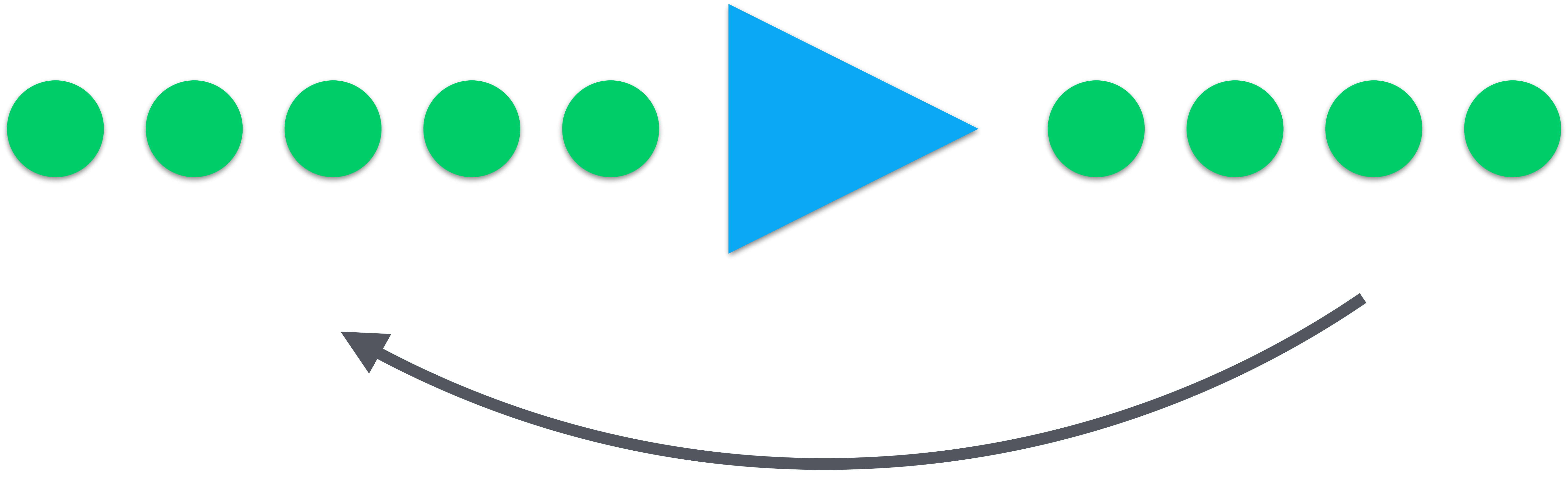


Backpressure

using RxJava 2


Consumer


Consumer
To many things !


backpressure

MissingBackpressureException ?

```
acceptedIntent
        .filter(intent -> !intent.getBooleanExtra("UpdatePhoneMode", false))
        .concatMap(intent -> approximatedEngine.detectCurrentPlace())
        .doOnNext(score -> Log.info(TAG, "Scan completed with result " + score))
        .concatMap(this::detectSleepMode)
        .concatMap((score) -> isNewPlace(score.getScore().getPlace()).map(p -> score))
        .doOnNext((p) -> Log.info(TAG, "Current place found is : " + p))
        .subscribe()
```

Added while I panicked

```
acceptedIntent
        .filter(intent -> !intent.getBooleanExtra("UpdatePhoneMode", false))
        .onBackpressureDrop()
        .concatMap(intent -> approximatedEngine.detectCurrentPlace())
        .doOnNext(score -> Log.info(TAG, "Scan completed with result " + score))
        .onBackpressureDrop()
        .concatMap(this::detectSleepMode)
        .onBackpressureDrop()
        .concatMap((score) -> isNewPlace(score.getScore().getPlace()).map(p -> score))
        .doOnNext((p) -> Log.info(TAG, "Current place found is : " + p))
        .subscribe()
```

RxJava 2

	Contrat	Backpressure
Observable	[N]	No
Single	[1]	No
Completable	[0]	No
Maybe	[0|1]	No

Close to Java 8 Optional

New!

RxJava 2

	Contrat	Backpressure
Observable	[N]	**No**
Single	[1]	**No**
Completable	[0]	**No**
Maybe	[0|1]	**No**
Flowable	[N]	Yes

Close to Java 8 Optional

Observable with back pressure

New!

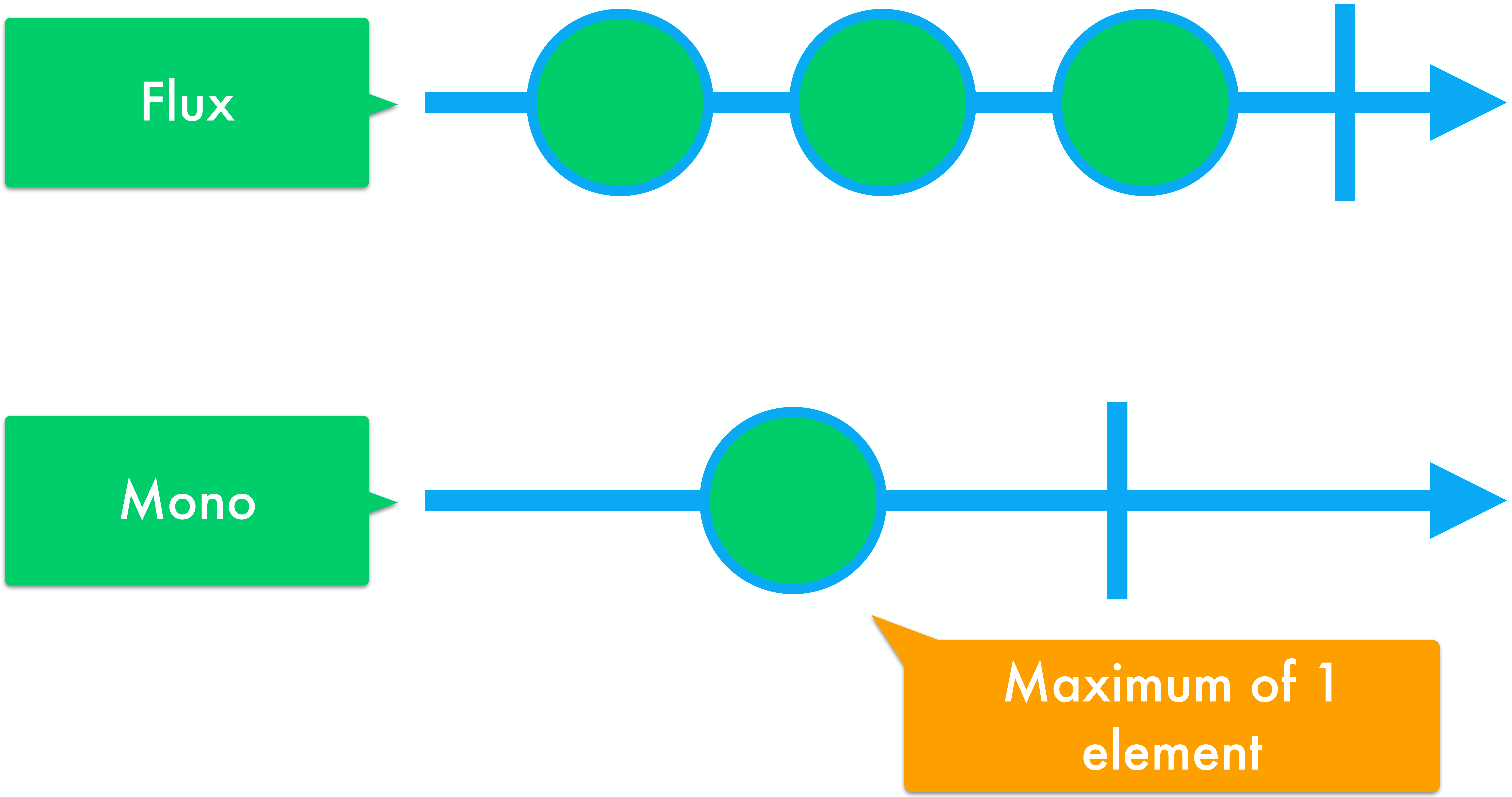
Observable

- ➞ less than 1000 events
- ➞ User interface management
- ➞ To be used instead of Java 8 Streams

Flowable

- ➞ more than 10 000 events
- ➞ Control the data flow
- ➞ Network stream with flow management

What does Reactor *offer ?*


Flux
Mono
Maximum of 1 element

Reactor

	Contrat	Backpressure
Flux	[N]	Yes
Mono	[0|1]	Yes

Identical to Flowable

Flux with only 1 element

Object types and Reactive Streams

Reactive Streams

Publisher
Flux
Flowable

```
Flux.range(1, 10)
        .flatMap(i -> Flux.just(1))
        .subscribe();
```

Publisher

```
Flux.range(1, 10)
         .flatMap(i -> Flux.just(1))
         .subscribe();
```

```
Flux.range(1, 10)
    .flatMap(i -> Flowable.just(1))
    .subscribe();
```

Publisher

RxJava 2

RxJava 2

Reactor

```
Flowable.defer(() -> Flux.range(1, 10))
        .subscribe(System.out::println);
```

Reactor

RxJava 2

```
Flux.defer(() -> Flowable.range(1, 10))
        .subscribe(System.out::println);
```

You can use a library which use RxJava 2 in your Reactor project

(and vice versa)

Operators

Consequent and homogenous

Catalogue

all amb ambArray ambWith any as awaitOnSubscribe blockFirst blockFirstMillis blockLast blockLastMillis blockingFirst blockingForEach blockingIterable blockingLast blockingLatest blockingMostRecent blockingNext blockingSingle blockingSubscribe buffer bufferMillis bufferSize bufferTimeout bufferTimeoutMillis bufferUntil bufferWhile cache cacheWithInitialCapacity cancelOn cast checkpoint collect collectInto collectList collectMap collectMultimap collectSortedList combineLatest combineLatestDelayError compose concat concatArray concatArrayDelayError concatArrayEager concatDelayError concatEager concatMap concatMapDelayError concatMapEager concatMapEagerDelayError concatMapIterable concatWith contains count create debounce defaultIfEmpty defer delay delayElements delayElementsMillis delayMillis delaySubscription delaySubscriptionMillis dematerialize distinct distinctUntilChanged doAfterNext doAfterTerminate doFinally doOnCancel doOnComplete doOnEach doOnError doOnLifecycle doOnNext doOnRequest doOnSubscribe doOnTerminate elapsed elementAt elementAtOrError empty equals error filter first firstElement firstEmitting firstEmittingWith firstOrError flatMap flatMapCompletable flatMapIterable flatMapMaybe flatMapSequential flatMapSingle forEach forEachWhile from fromArray fromCallable fromFuture fromIterable fromPublisher fromStream generate getClass getPrefetch groupBy groupJoin handle hasElement hasElements hashCode hide ignoreElements interval intervalMillis intervalRange isEmpty join just last lastElement lastOrError lift limitRate log map mapError materialize merge mergeArray mergeArrayDelayError mergeDelayError mergeSequential mergeWith never next notify notifyAll observeOn ofType onBackpressureBuffer onBackpressureDrop onBackpressureError onBackpressureLatest onErrorResumeNext onErrorResumeWith onErrorReturn onErrorReturnItem onExceptionResumeNext onTerminateDetach parallel publish publishNext publishOn range rangeLong rebatchRequests reduce reduceWith repeat repeatUntil repeatWhen replay replayMillis retry retryUntil retryWhen safeSubscribe sample sampleFirst sampleFirstMillis sampleMillis sampleTimeout scan scanWith sequenceEqual serialize share single singleElement singleOrEmpty singleOrError skip skipLast skipMillis skipUntil skipUntilOther skipWhile sort sorted startWith startWithArray strict subscribe subscribeOn subscribeWith switchIfEmpty switchMap switchMapDelayError switchOnError switchOnNext switchOnNextDelayError take takeLast takeMillis takeUntil takeUntilOther takeWhile test then thenEmpty thenMany throttleFirst throttleLast throttleWithTimeout timeInterval timeout timeoutMillis timer timestamp to toFuture toIterable toList toMap toMultimap toObservable toSortedList toStream toString transform unsafeCreate unsubscribeOn using wait window windowMillis windowTimeout windowTimeoutMillis windowUntil windowWhile withLatestFrom zip zipArray zipIterable zipWith zipWithIterable

RxJava	RxJava 2	Reactor	
flatMap	flatMap	flatMap	Emit Noe, one or more events
amb	amb	firstEmitting	Emit events from the first emitting stream
…	…	…	…
debounce	debounce	N/A	Ignore events during a time laps

RxJava	RxJava 2	Reactor	
flatMap	flatMap	flatMap	Emit Noe, one or more events
amb	amb	firstEmitting	Emit events from the first emitting stream
…	…		…
debounce	debounce	N/A	Ignore events during a time laps

Renamed

Operators

cover

a lot of scenarios

Nota bene

writing operators is hard

when one writes an operator, the Observable protocol, unsubscription, backpressure and concurrency have to be taken into account and adhered to the letter

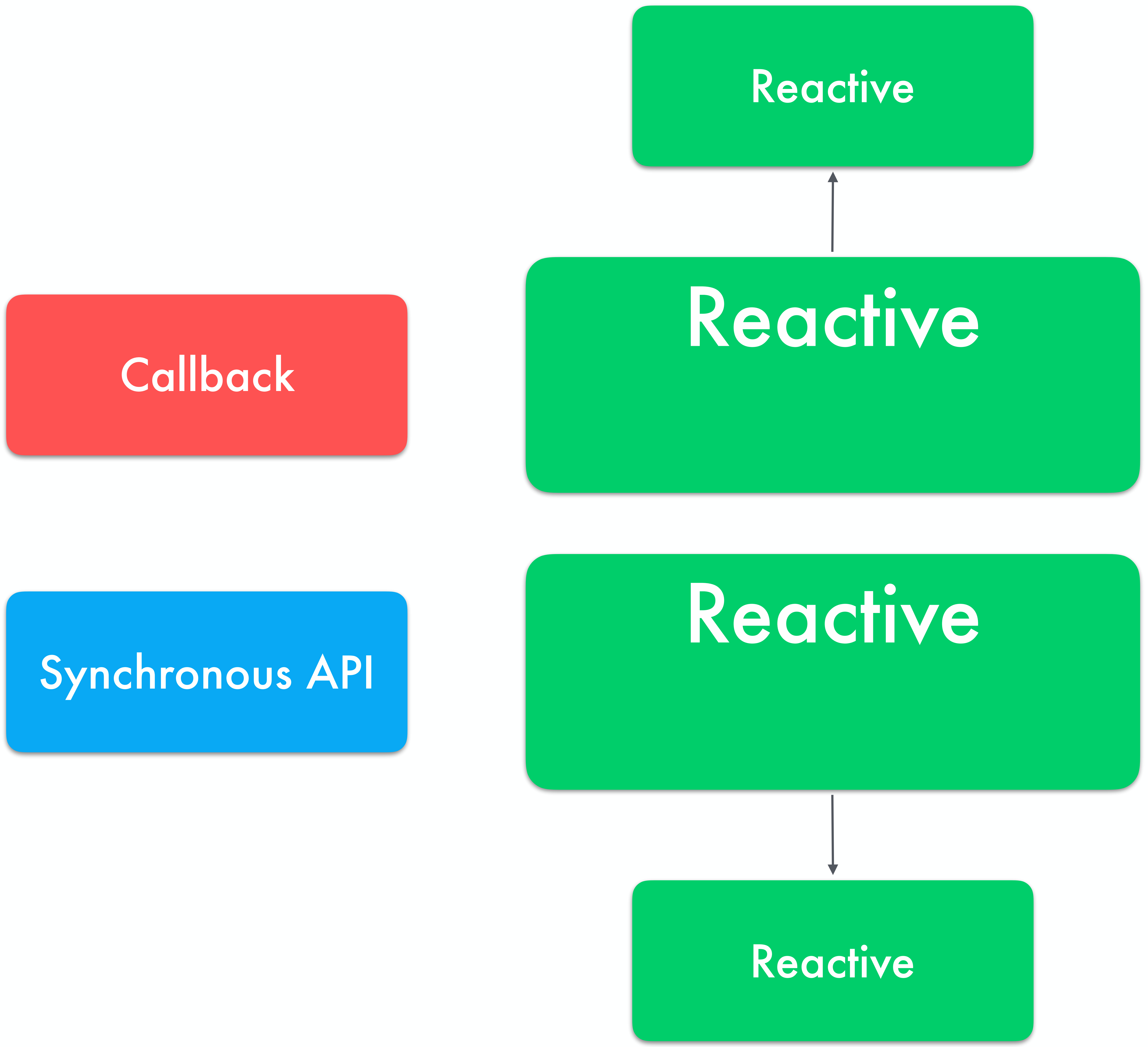
https://github.com/ReactiveX/RxJava/wiki/Implementing-custom-operators-(draft)

Writing a new operator with RxJava 2

is more complex

than with RxJava

Make a application Reactive


Reactive
Callback
Reactive
Synchronous API
Reactive
Reactive

Factory

RxJava RxJava 2	Reactor	
Flowable.just	Flux.just	Emitting existing value
Flowable.defer	Flux.defer	Lazy emitting
Flowable.fromCallable	Mono.fromCallable	Lazy emitting, computed from a method call
Flowable.create	Flux.create	Manual emitting
Flowable.using	Flux.using	Resource management
Flowable.fromPublisher	Flux.from	Using a Publisher (Reactive Streams)
Flowable.generate	Flux.generate	Using a value generator

RxJava RxJava 2	Reactor	
Flowable.just	**Flux.just**	Emitting existing value
Flowable.defer	**Flux.defer**	Lazy emitting
Flowable.fromCallable	**Mono.fromCallable**	Lazy emitting, computed from a method call
Flowable.create	**Flux.create**	Manual emitting
Flowable.using	**Flux.using**	Resource management
Flowable.fromPublisher	**Flux.from**	Using a Publisher (Reactive Streams)
Flowable.generate	**Flux.generate**	Using a value generator

RxJava RxJava 2	Reactor	
Flowable.just	Flux.just	Emitting existing value
Flowable.defer	Flux.defer	Lazy emitting
Flowable.fromCallable	Mono.fromCallable	Lazy emitting, computed from a method call
Flowable.create	Flux.create	Manual emitting
Flowable.using	Flux.using	Resource management
Flowable.fromPublisher	Flux.from	Using a Publisher (Reactive Streams)
Flowable.generate	Flux.generate	Using a value generator

RxJava RxJava 2	Reactor	
Flowable.just	**Flux.just**	Emitting existing value
Flowable.defer	**Flux.defer**	Lazy emitting
Flowable.fromCallable	**Mono.fromCallable**	Lazy emitting, computed from a method call
Flowable.create	**Flux.create**	Manual emitting
Flowable.using	**Flux.using**	Resource management
Flowable.fromPublisher	**Flux.from**	Using a Publisher (Reactive Streams)
Flowable.generate	**Flux.generate**	Using a value generator

example of

wrapping

```java
@RestController
public class HelloController {

    private static final byte[] TOPIC_NAME = "topic".getBytes();

    @RequestMapping(value = "/redis")
    private String redis() throws InterruptedException {
        CountDownLatch latch = new CountDownLatch(1);
        AtomicReference<String> result = new AtomicReference<>();
        this.connection.subscribe((message, pattern) -> {
            result.set(message.toString());
            latch.countDown();
        }, TOPIC_NAME);
        latch.await();
        return result.get();
    }
}
```

```
@RestController
public class HelloController {

    private static final byte[] TOPIC_NAME = "topic".getBytes();

    @RequestMapping(value = "/redis")
    private String redis() throws InterruptedException {
        CountDownLatch latch = new CountDownLatch(1);
        AtomicReference<String> result = new AtomicReference<>();
        this.connection.subscribe((message, pattern) -> {
            result.set(message.toString());
            latch.countDown();
        }, TOPIC_NAME);
        latch.await();
        return result.get();
    }
}
```

```
@RestController
public class HelloController {

    private static final byte[] TOPIC_NAME = "topic".getBytes();

    @RequestMapping(value = "/redis")
    private String redis() throws InterruptedException {
        CountDownLatch latch = new CountDownLatch(1);
        AtomicReference<String> result = new AtomicReference<>();
        this.connection.subscribe((message, pattern) -> {
            result.set(message.toString());
            latch.countDown();
        }, TOPIC_NAME);
        latch.await();
        return result.get();
    }
}
```

```
@RestController
public class HelloController {

    private static final byte[] TOPIC_NAME = "topic".getBytes();

    @RequestMapping(value = "/redis")
    private String redis() throws InterruptedException {
        CountDownLatch latch = new CountDownLatch(1);
        AtomicReference<String> result = new AtomicReference<>();
        this.connection.subscribe((message, pattern) -> {
            result.set(message.toString());
            latch.countDown();
        }, TOPIC_NAME);
        latch.await();
        return result.get();
    }
}
```

Code for synchronisation

Code for synchronisation

Step 1

Wrapping

```
@RestController
public class HelloController {

    private static final byte[] TOPIC_NAME = "topic".getBytes();

    @RequestMapping(value = "/redis")
    private String redis() throws InterruptedException {

        String result = Flowable.create(sub -> {
                              this.connection.subscribe((message, pattern) -> {
                                      sub.onNext(message.toString());
                                      sub.onComplete();
                              }, TOPIC_NAME);
                }, BackpressureStrategy.BUFFER)
                .blockingFirst();
        return result;
    }
}
```

```
@RestController
public class HelloController {

    private static final byte[] TOPIC_NAME = "topic".getBytes();

    @RequestMapping(value = "
    private String redis() throws

        String result = Flowable.create(sub -> {
                            this.connection.subscribe((message, pattern) -> {
                                    sub.onNext(message.toString());
                                    sub.onComplete();
                            }, TOPIC_NAME);
                }, BackpressureStrategy.BUFFER)
                .blockingFirst();
        return result;
    }
}
```

Wrapping

Reactive Contract

Synchronisation

Step 2

Asynchronous

```
@RestController
public class HelloController {

    private static final byte[] TOPIC_NAME = "topic".getBytes();

    @RequestMapping(value = "/redis")
    private DeferredResult<String> redis() throws InterruptedException {

        DeferredResult<String> result = new DeferredResult<>(10_000l);

        Flowable.create(sub -> {
                               this.connection.subscribe((message, pattern) -> {
                                         sub.onNext(message.toString());
                                         sub.onComplete();
                               }, TOPIC_NAME);
            }, BackpressureStrategy.BUFFER)
            .subscribe(result::setResult);

        return result;
    }
@RestController
public class HelloController {

  private static final byte[] TOPIC_NAME = "topic".getBytes();

  @RequestMapping(value = "/redis")
  private DeferredResult<String> redis() throws InterruptedException {

    DeferredResult<String> result = new DeferredResult<>(10_000l);

    Fl
                                  cribe((message, pattern) -> {
                      sub.onNext(message.toString());
                      sub.onComplete();
               }, TOPIC_NAME);
        }, BackpressureStrategy.BUFFER)
        .subscribe(result::setResult);

    return result;
  }
```

Use of DeferredResult

Lazy result

Step 3

Reactive Streams

```java
@RestController
public class HelloController {

    private static final byte[] TOPIC_NAME = "topic".getBytes();

    @RequestMapping(value = "/redis", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
    private Flux<String> redis() throws InterruptedException {

        Flowable<String> rxjava = Flowable.create(sub -> {
            this.connection.subscribe((message, pattern) -> sub.onNext(message.toString()),
                                                                TOPIC_NAME);
        }, BackpressureStrategy.BUFFER);

        return Flux.defer(() -> rxjava);
    }
}
```

Return a Flux

Flux → SSE

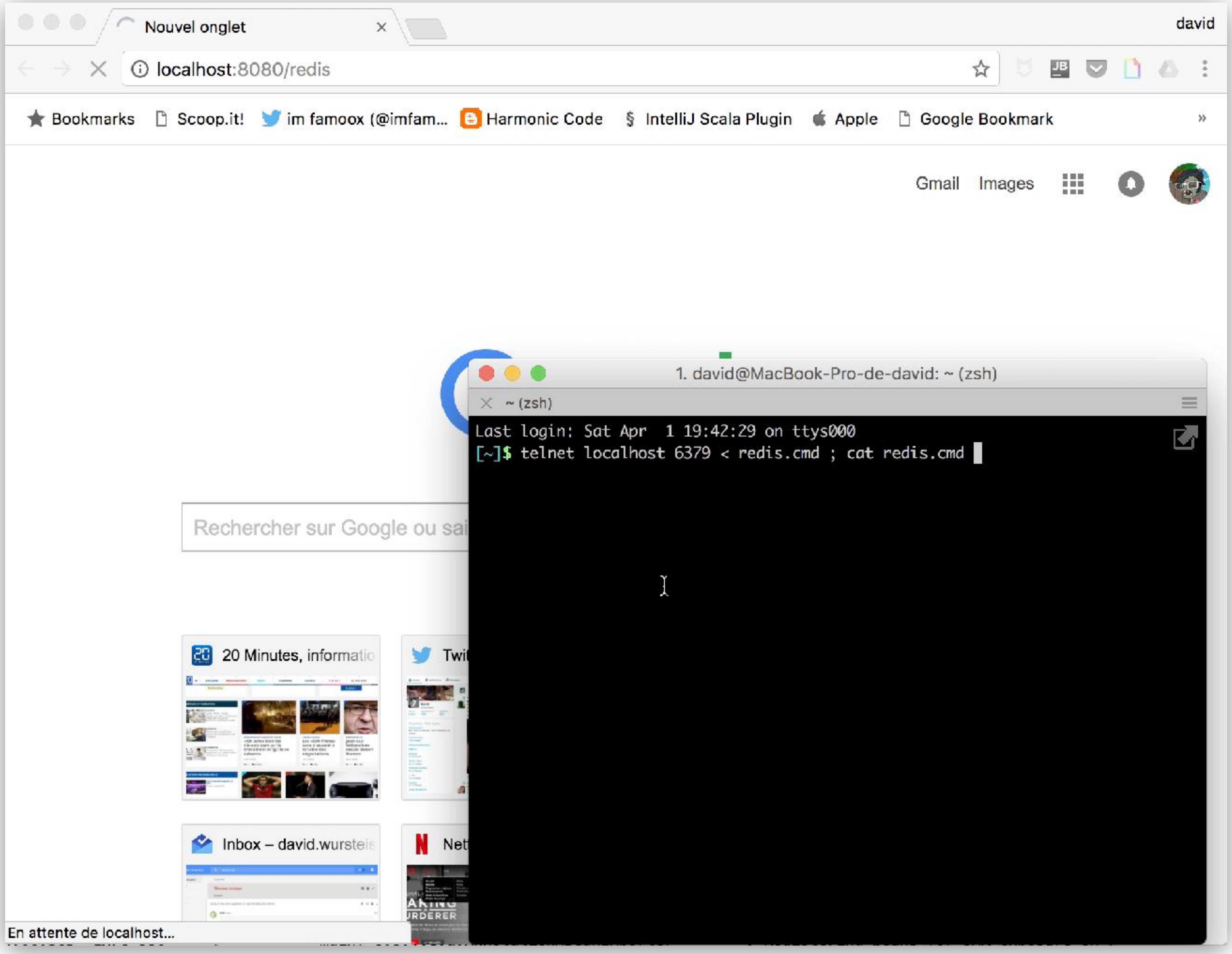
```
@RestController
                          ontroller {

  private st    al byte[] TO         =  topic  .get        ();

  @RequestMapping(value = "/redis", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
  private Flux<String> redis() throws InterruptedException {

    Flowable<String> rxjava = Flowable.create(sub -> {
      this.connection.subscribe((message, pattern) -> sub.onNext(message.toString()),
                                                                    TOPIC_NAME);
    }, BackpressureStrategy.BUFFER);

    return Flux.defer(() -> rxjava);
  }
}
```

RxJava 2 → Flux

```java
@RestController
public class HelloController {

    private static final byte[] TOPIC_NAME = "topic".getBytes();

    @RequestMapping(value = "/redis", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
    private Publisher<String> redis() throws InterruptedException {
        return Flowable.create(sub -> {
            this.connection.subscribe((message, pattern) -> sub.onNext(message.toString()),
                                                                TOPIC_NAME);
        }, BackpressureStrategy.BUFFER);

    }
```

Publisher

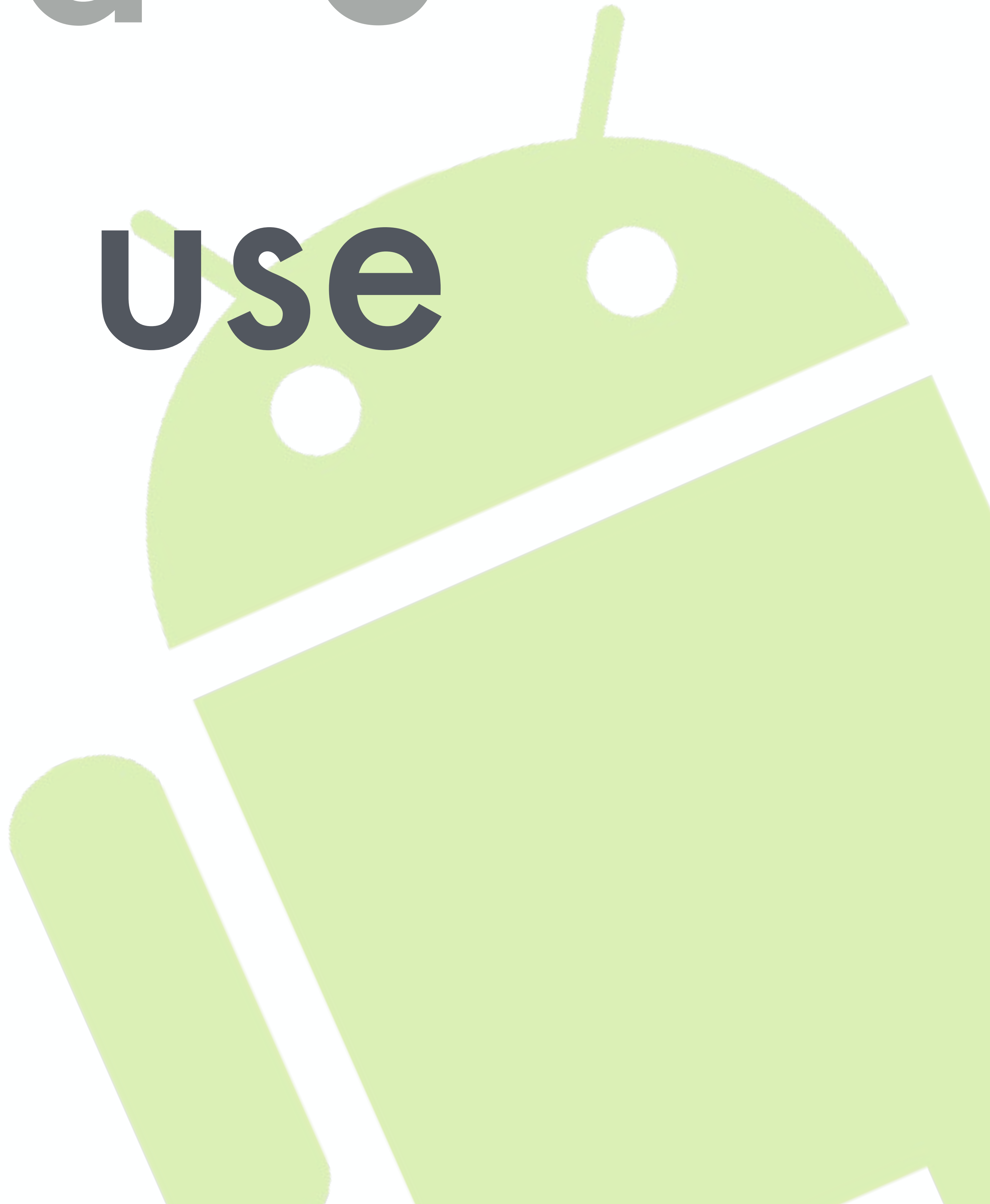
```
@RestController
public class HelloController {

    private static final byte[] TOPIC_NAME = "topic".getBytes();

    @RequestMapping(value = "/redis", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
    private Publisher<String> redis() throws InterruptedException {
        return Flowable.create(sub -> {
            this.connection.subscribe((message, pattern) -> sub.onNext(message.toString()),
                                                                  TOPIC_NAME);
        }, BackpressureStrategy.BUFFER);

    }
```



```
Last login: Sat Apr  1 19:42:29 on ttys000
[~]$ telnet localhost 6379 < redis.cmd ; cat redis.cmd
```

Reactor use Java 8 while RxJava 2 use Java 6

Reactor use Java 8 while RxJava 2 use Java 6

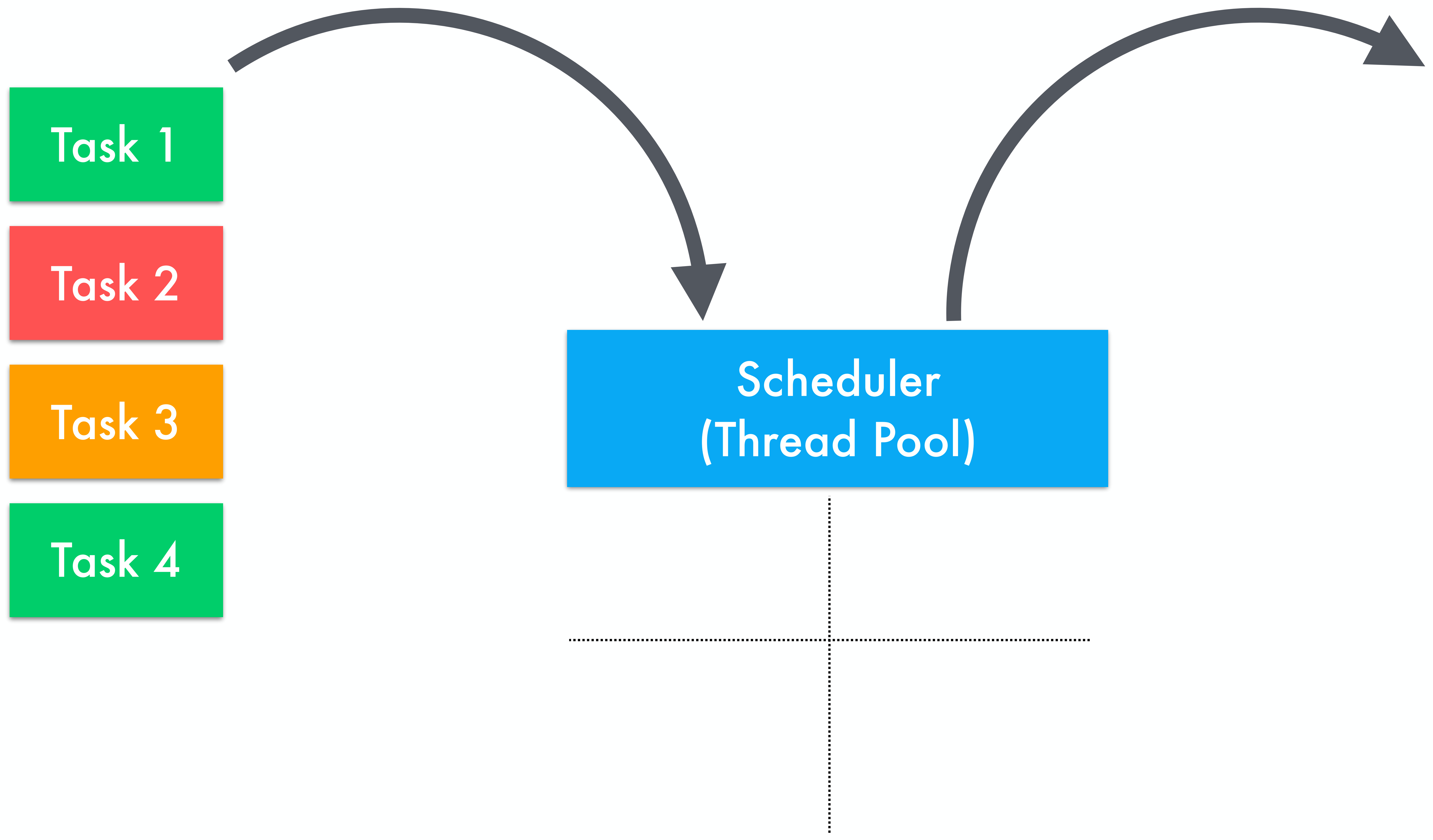


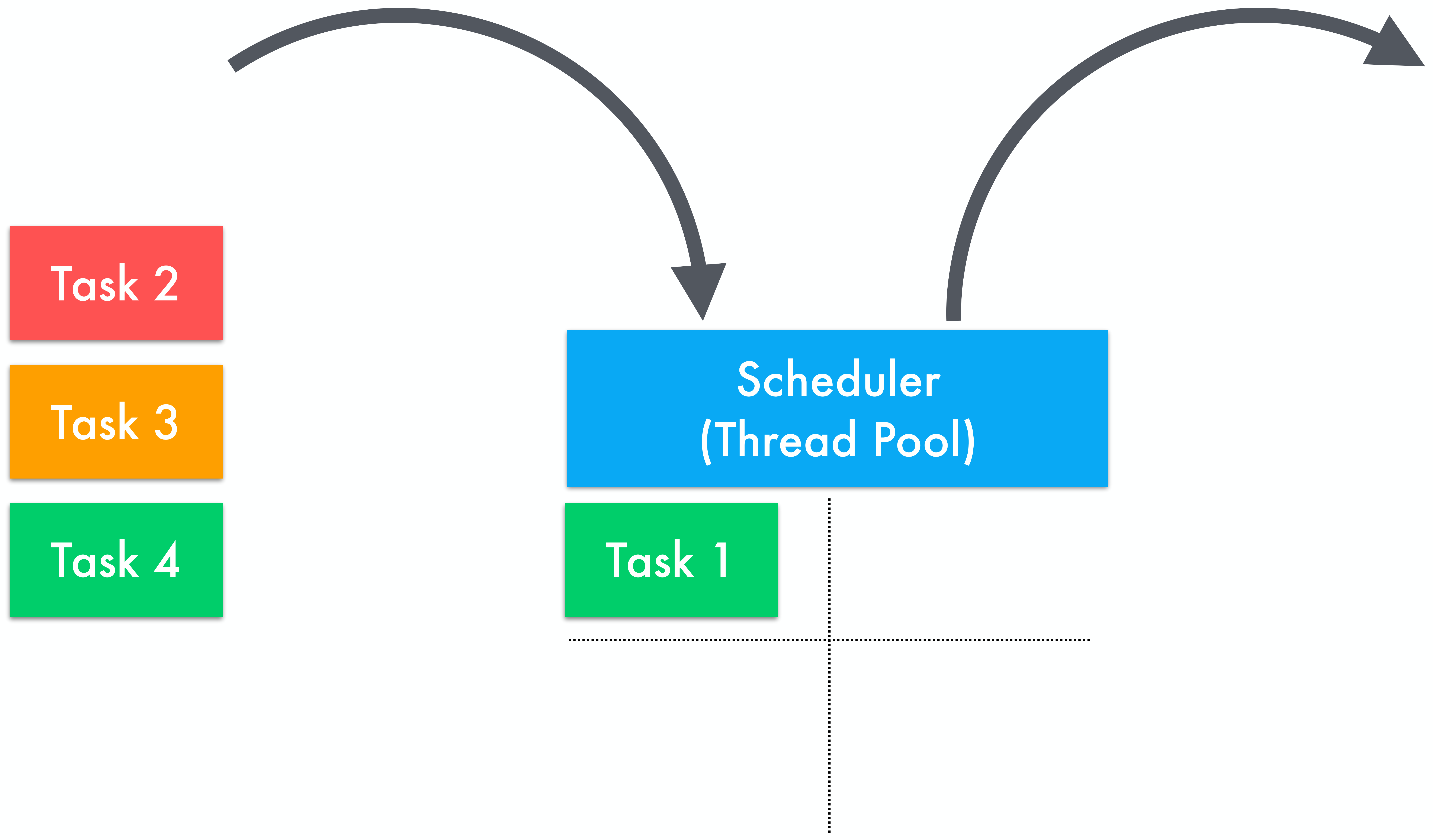
Asynchronous

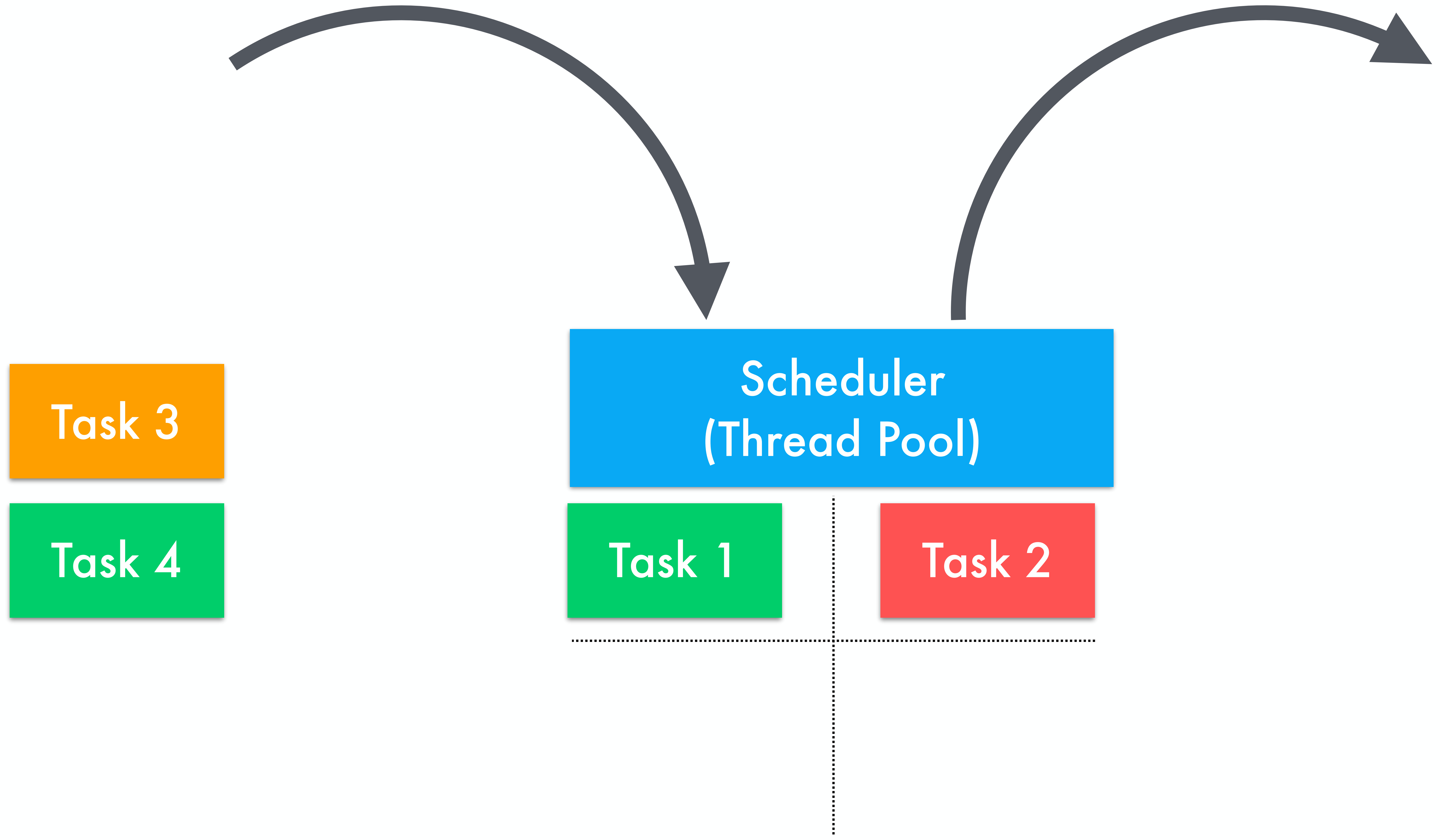
Execution context management

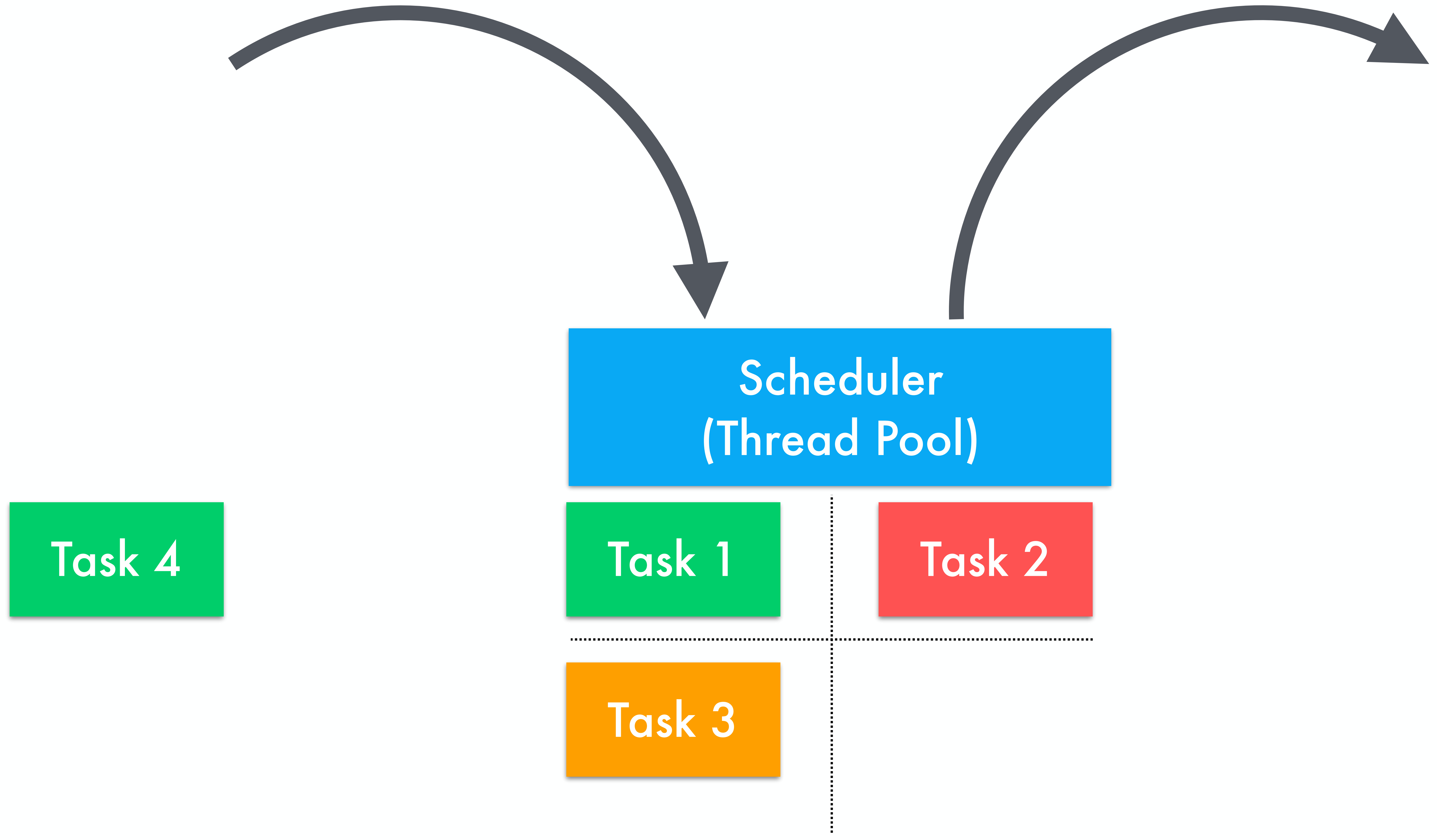
Rethink cette partie là

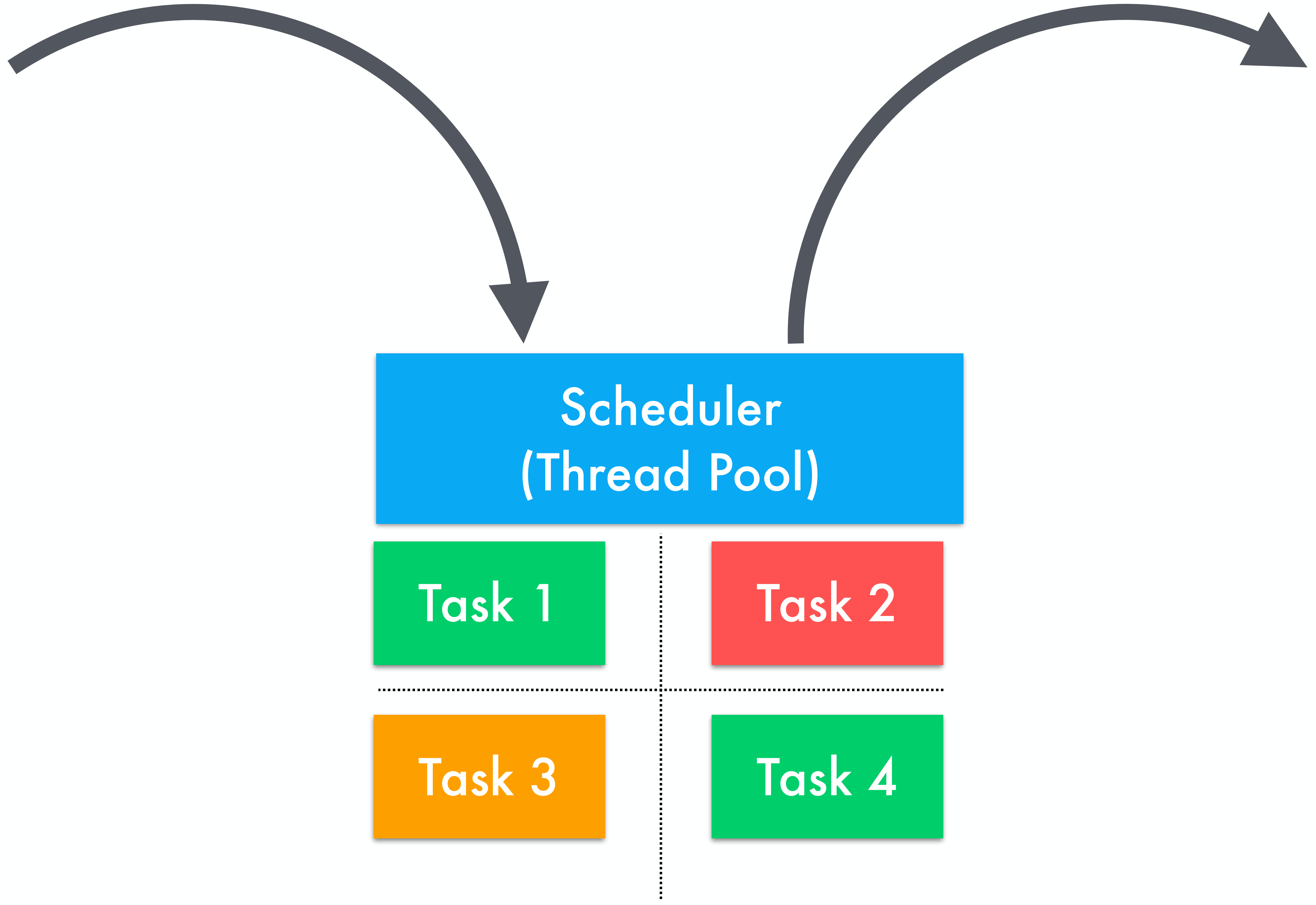
Schedulers

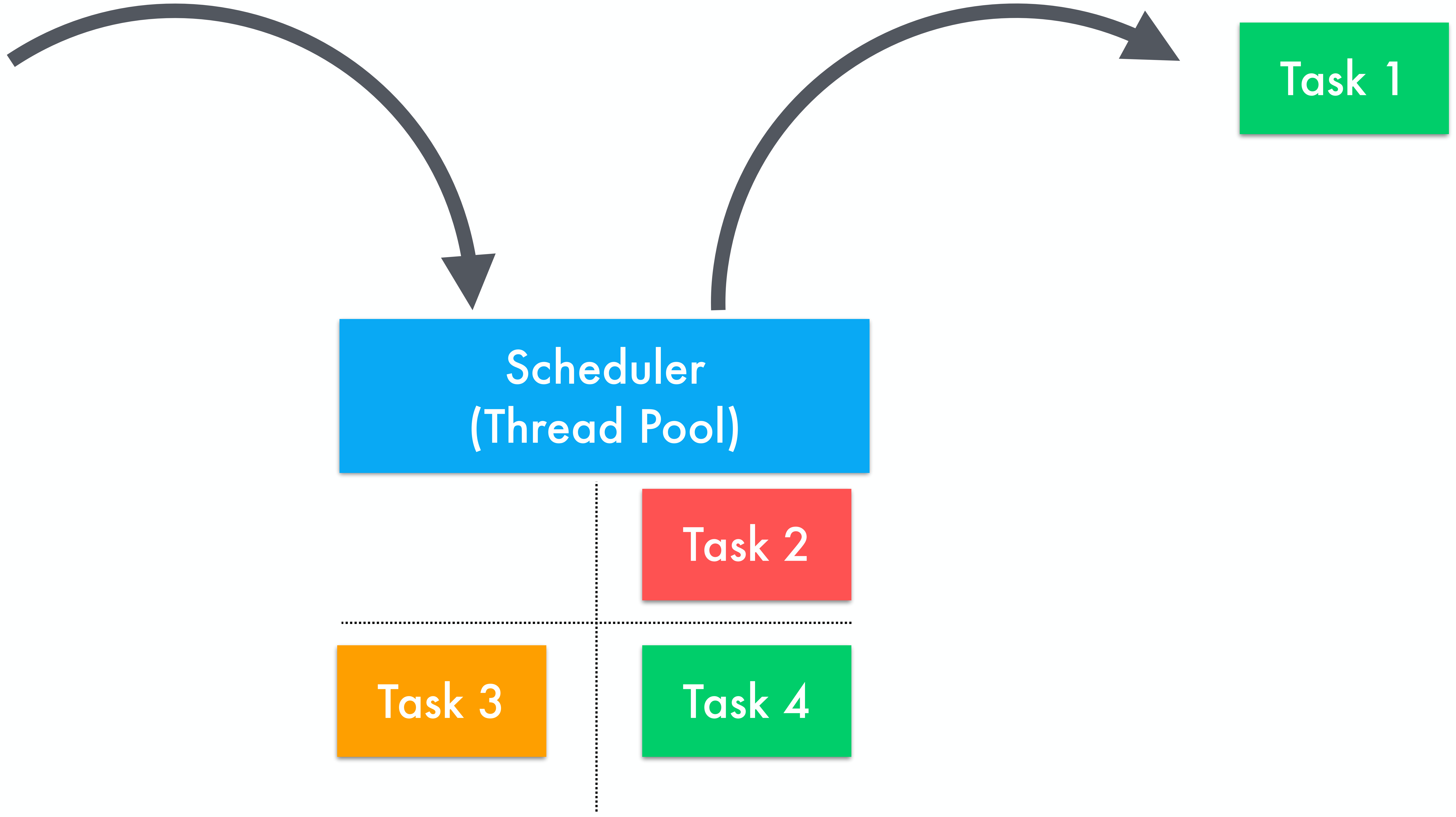

Task 1
Task 2
Task 3
Task 4
Scheduler
(Thread Pool)

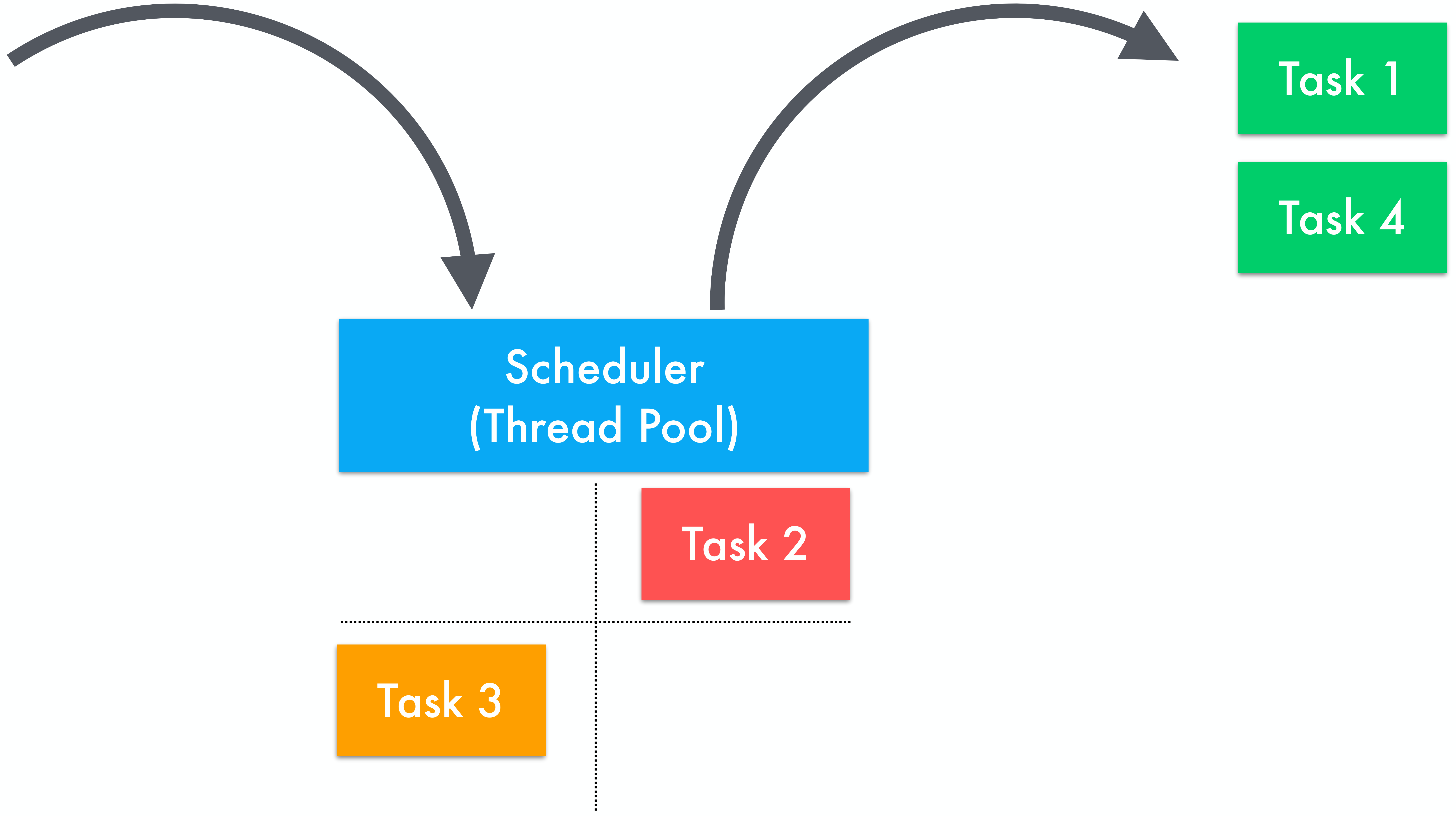

Task 2
Task 3
Task 4
Scheduler
(Thread Pool)
Task 1

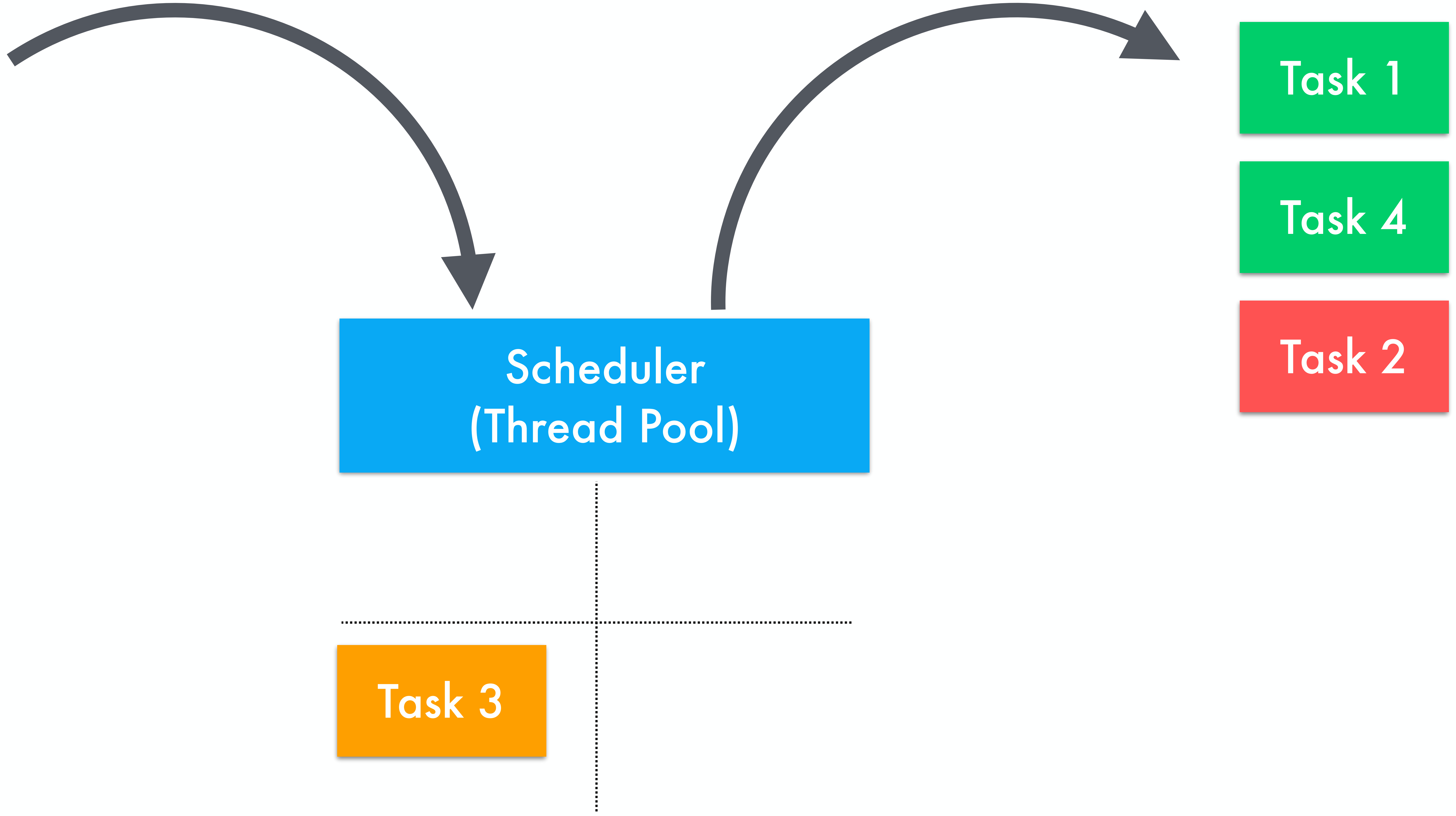

Task 3
Task 4
Scheduler
(Thread Pool)
Task 1
Task 2

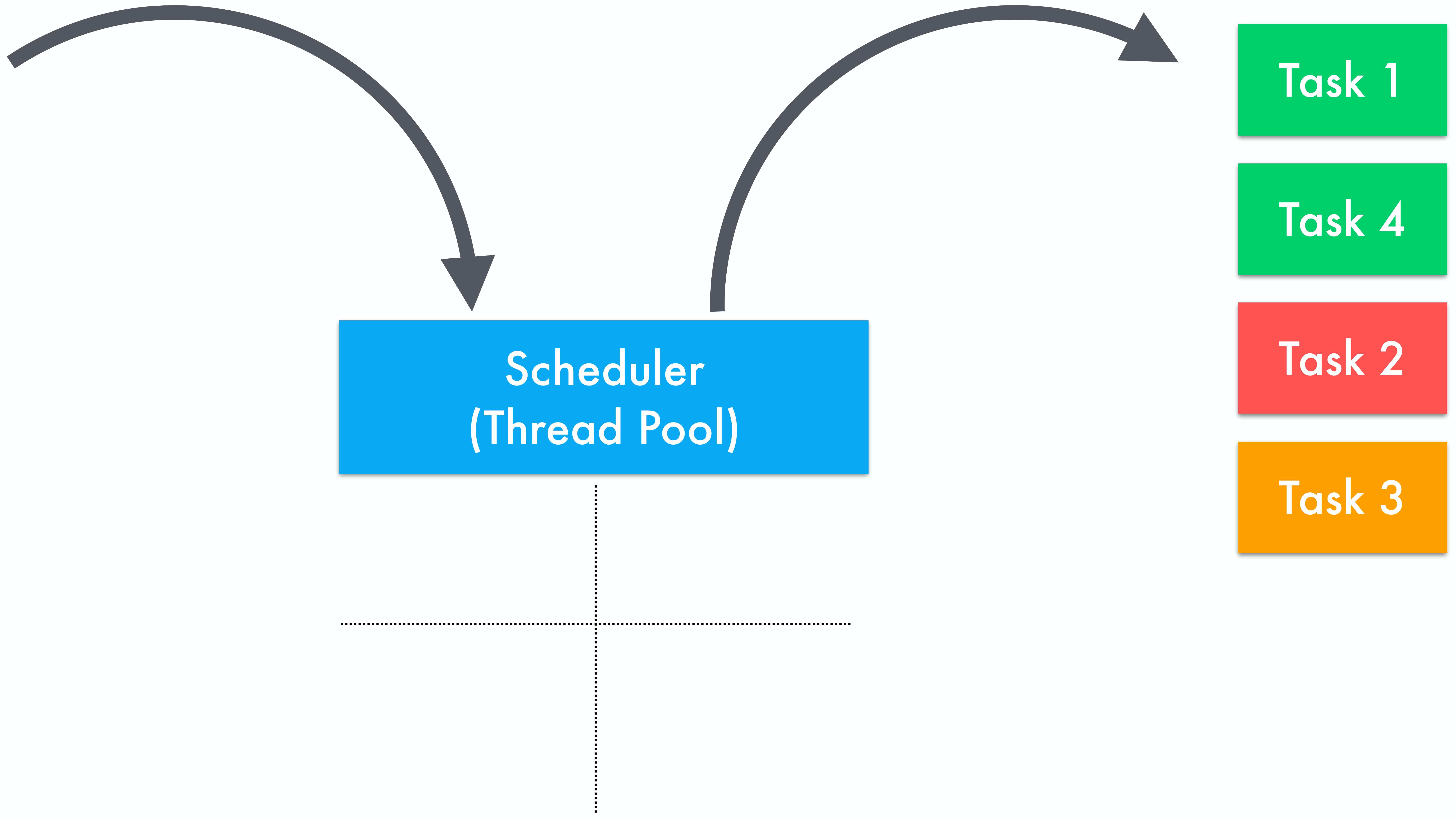

Scheduler
(Thread Pool)
Task 4
Task 1
Task 2
Task 3


Scheduler
(Thread Pool)
Task 1
Task 2
Task 3
Task 4


Task 1
Scheduler
(Thread Pool)
Task 2
Task 3
Task 4


Task 1
Task 4
Scheduler
(Thread Pool)
Task 2
Task 3


Task 1
Task 4
Task 2
Scheduler
(Thread Pool)
Task 3


Scheduler
(Thread Pool)
Task 1
Task 4
Task 2
Task 3

java.lang.IllegalStateException:
Not on the main thread

NetworkOnMainThreadException

```
JavaFx.fromClick(btn)
        .observeOn(Schedulers.io())
        .switchMap(evt -> remoteApi.getData())
        .observeOn(Schedulers.computation())
        .flatMap(data -> intensiveComputation(data))
        .observeOn(javaFx())
        .doOnNext(value -> btn.setText("Data: " + value))
        .subscribe();
```

```
JavaFx.fromClick(btn)
        .observeOn(Schedulers.io())
        .switchMap(evt -> remoteApi.getData())
        .observeOn(Schedulers.computation())
        .flatMap(data -> intensiveComputation(dat
        .observeOn(javaFx())
        .doOnNext(value -> btn.s                        ue))
        .subscribe();
```

i/o

computation

UI Thread

RxJava	RxJava 2	Reactor	Description
io()	io()	elastic()	Thread pool which grow up if needed
computation()	computation()	parallel()	Limited thread pool
single()	single()	single()	Pool of 1 thread
immediate()		immediate()	Execute the task immediately
trampoline()	trampoline()		Queue the current task

RxJava	RxJava 2	Reactor	Description
io()	io()	elastic()	Thread pool which grow up if needed
computation()	computation()	parallel()	Limited thread pool
single()	single()	single()	Pool of 1 thread
immediate()		immediate()	Execute the task immediately
trampoline()	trampoline()		Queue the current task

RxJava	RxJava 2	Reactor	Description
io()	io()	elastic()	Thread pool which grow up if needed
computation()	computation()	parallel()	Limited thread pool
single()	single()	single()	Pool of 1 thread
immediate()		immediate()	Execute the task immediately
trampoline()	trampoline()		Queue the current task

Reactor

Technical naming

RxJava

Functional naming

Performance

Generation

RxJava 2
Reactor
3
4
5+
Fusion

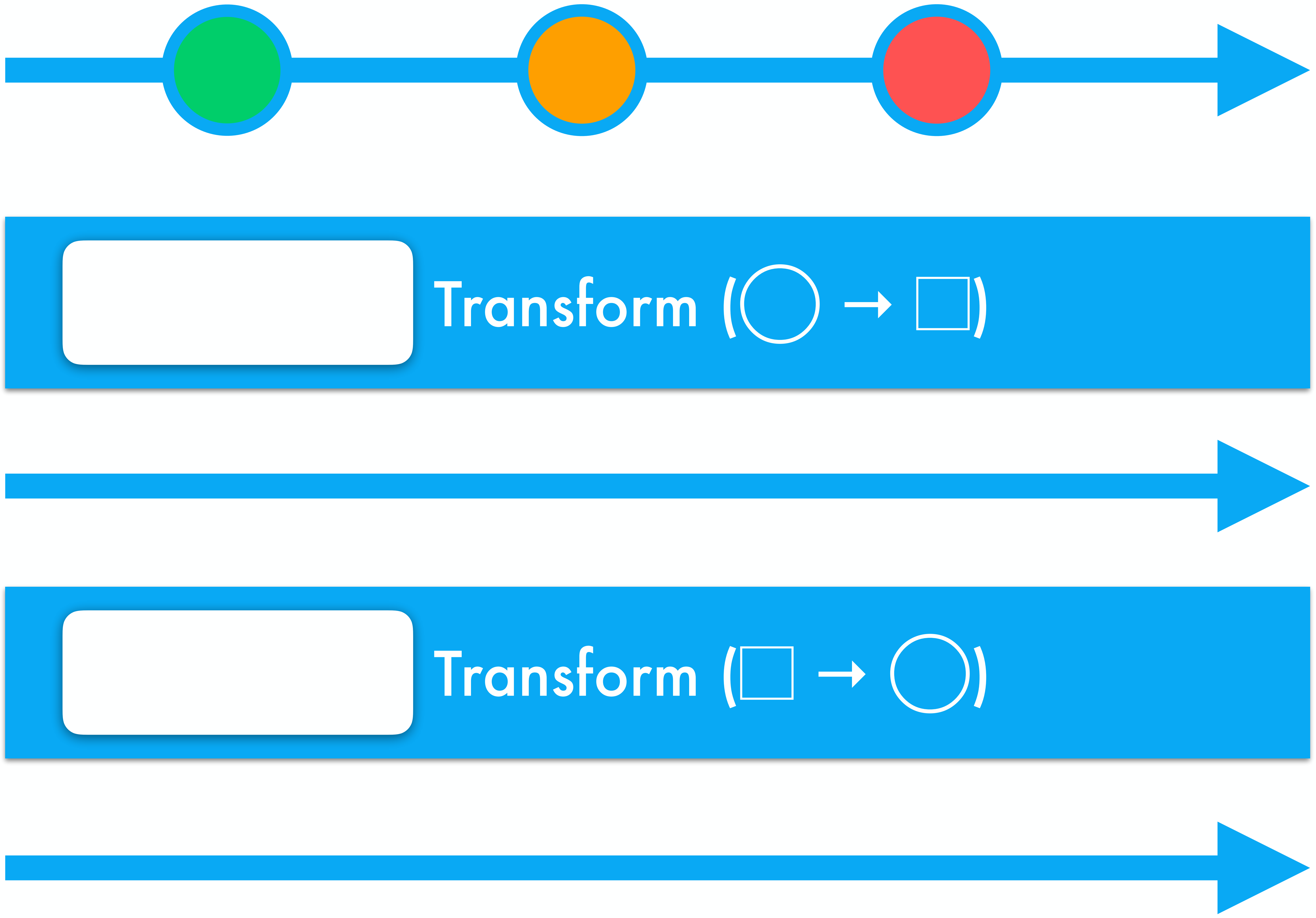
Warning

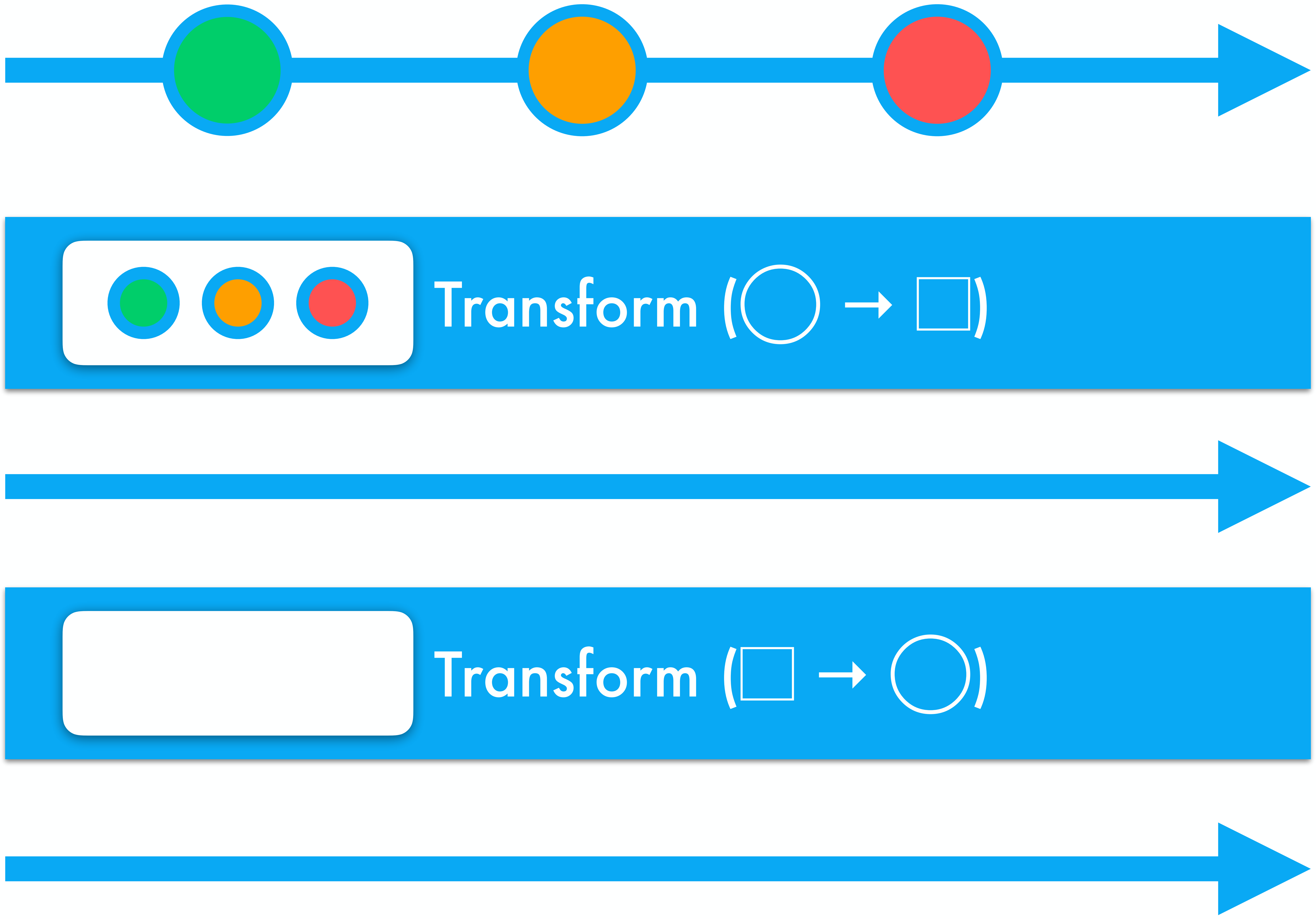
Conceptuel slides

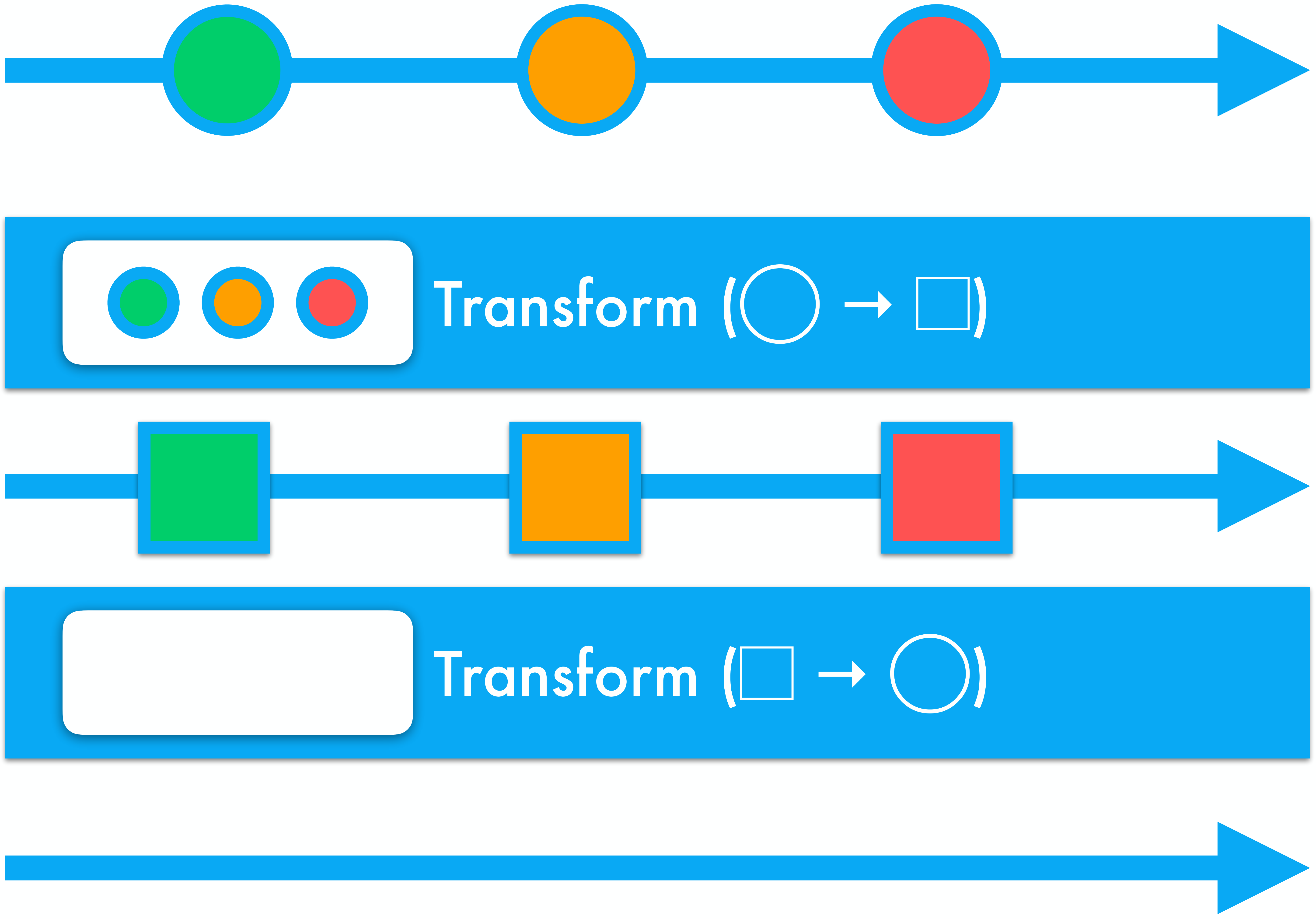
Without fusion

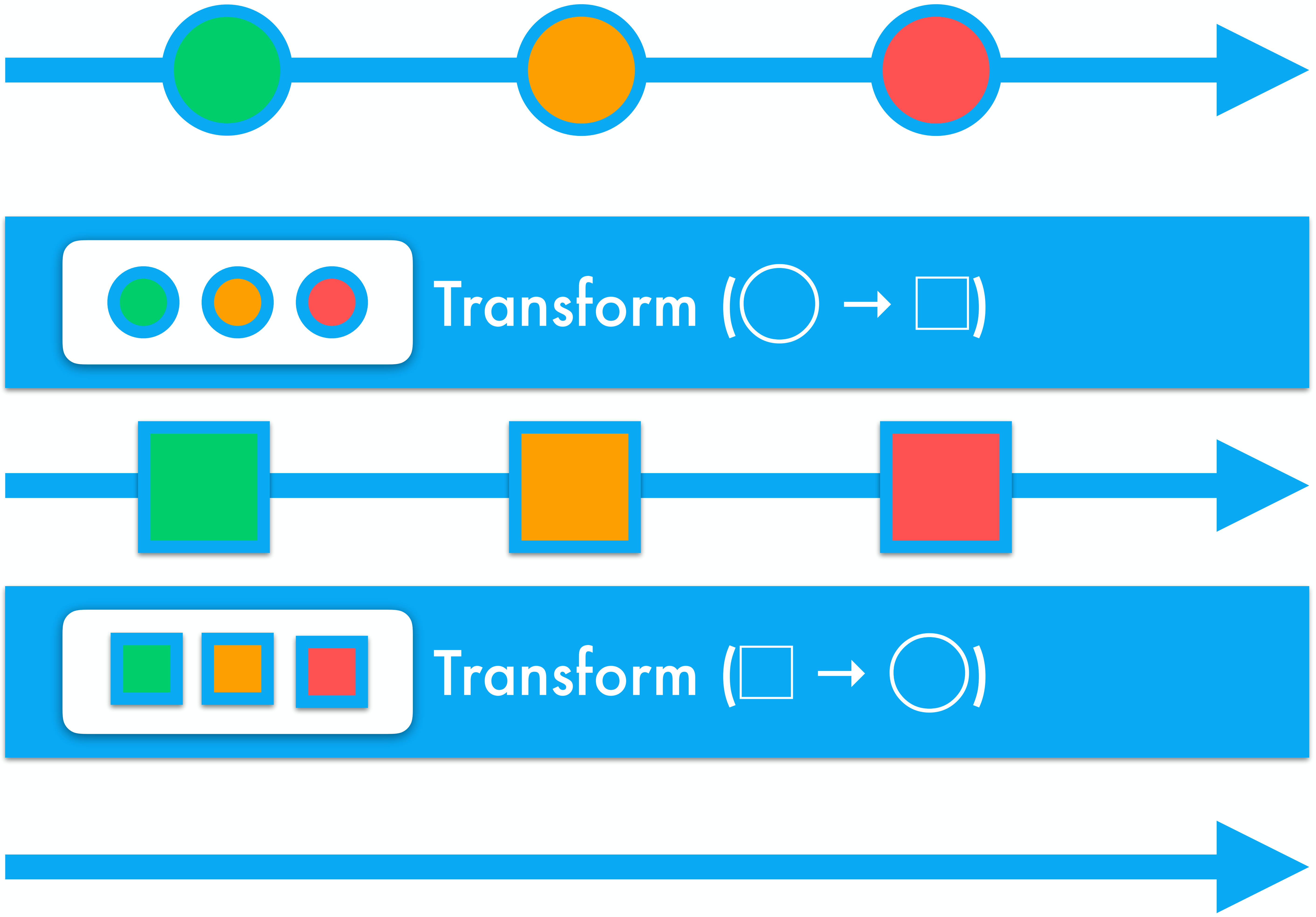
Transform (○ → □)

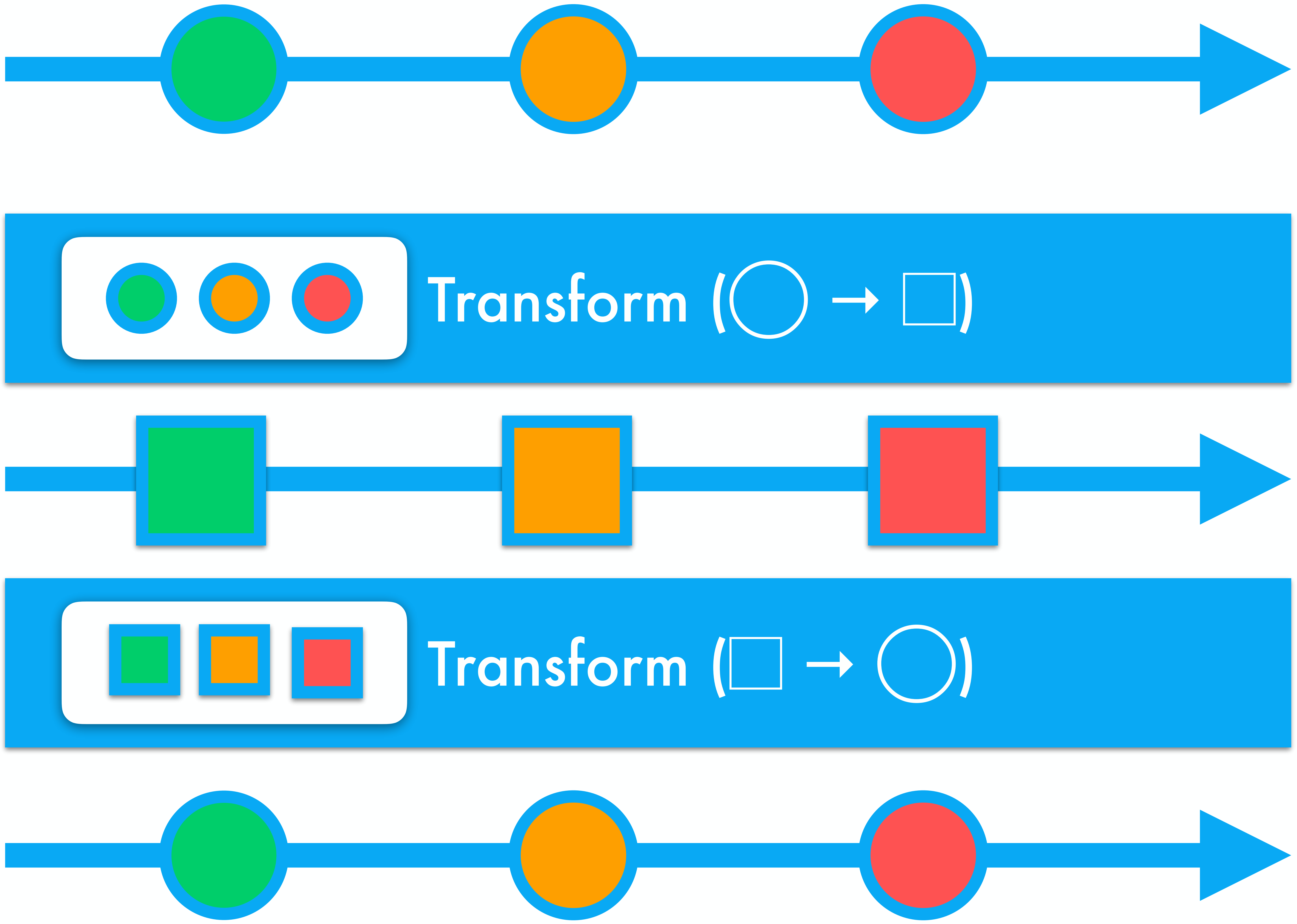
Transform (□ → ○)


Transform (○ → □)
Transform (□ → ○)

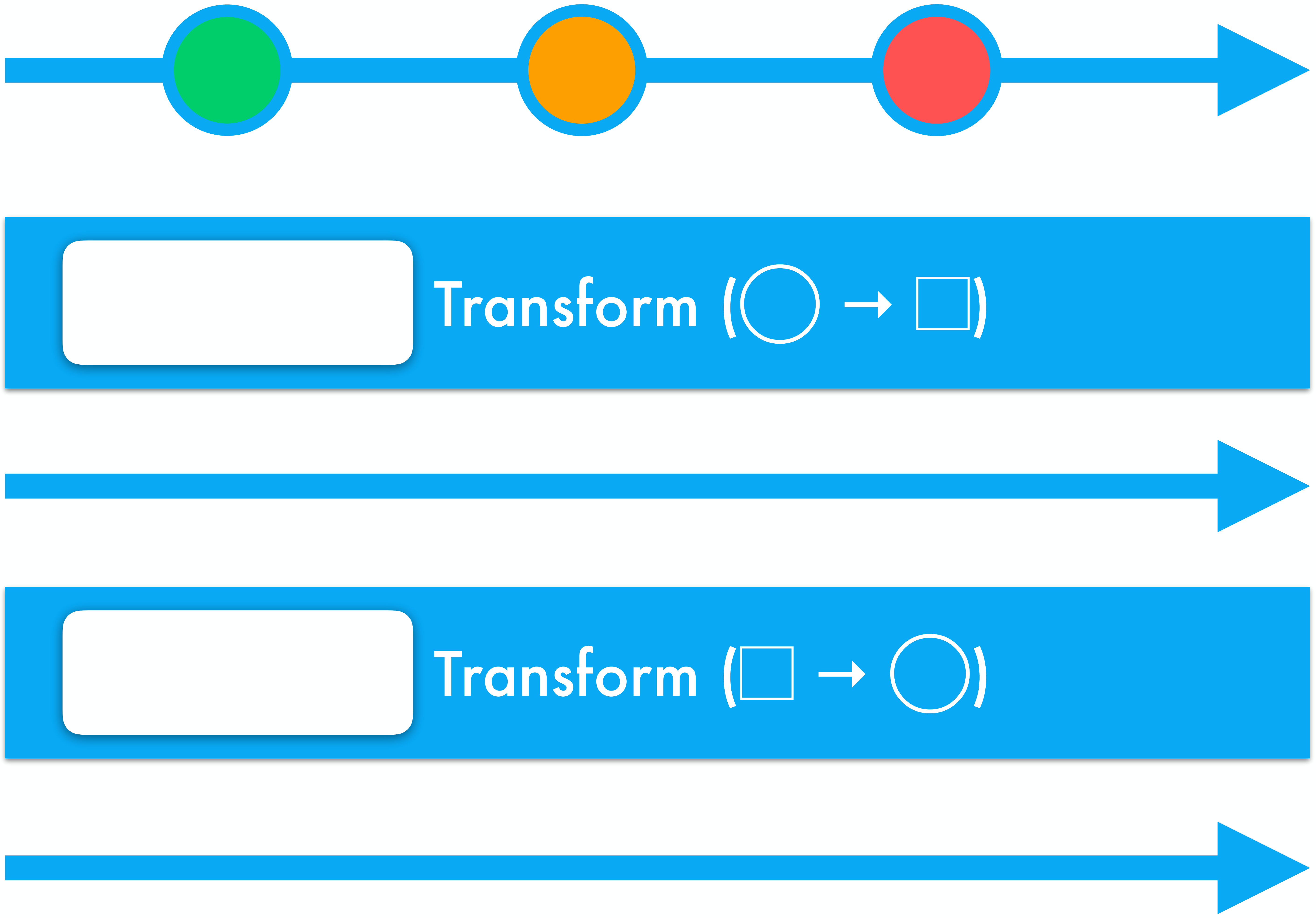

Transform (○ → □)
Transform (□ → ○)

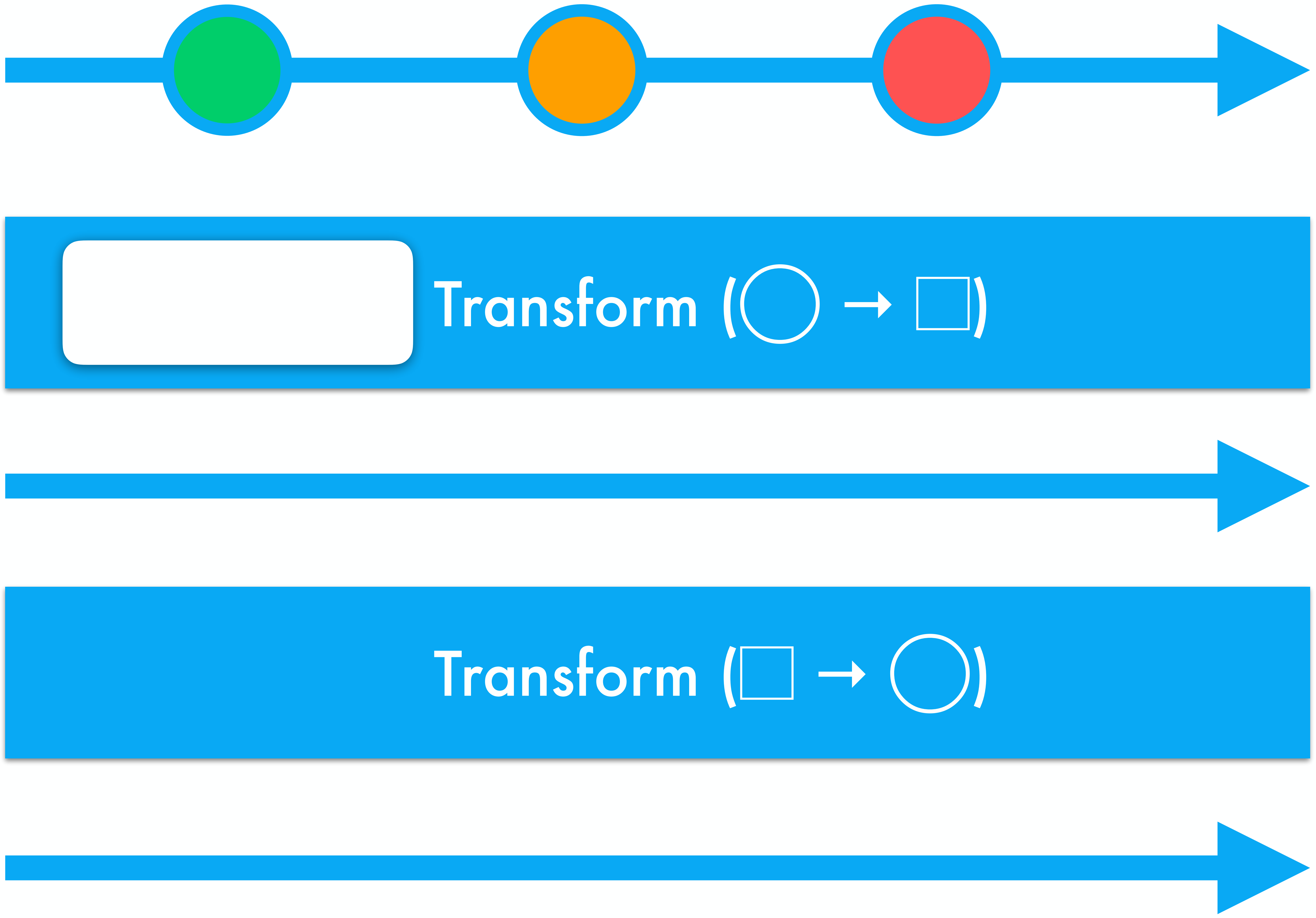

Transform (○ → □)
Transform (□ → ○)

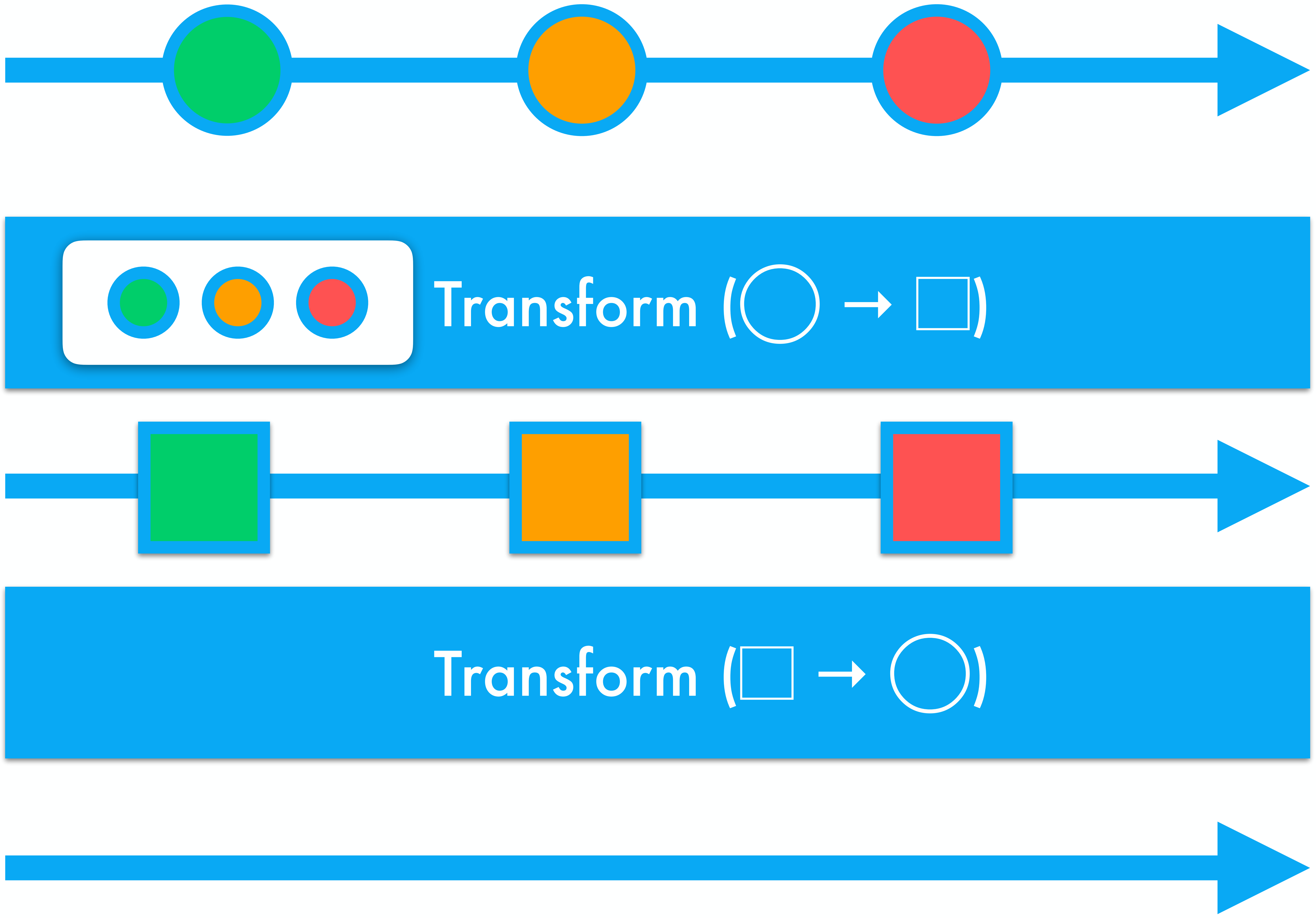

Transform (○ → □)
Transform (□ → ○)

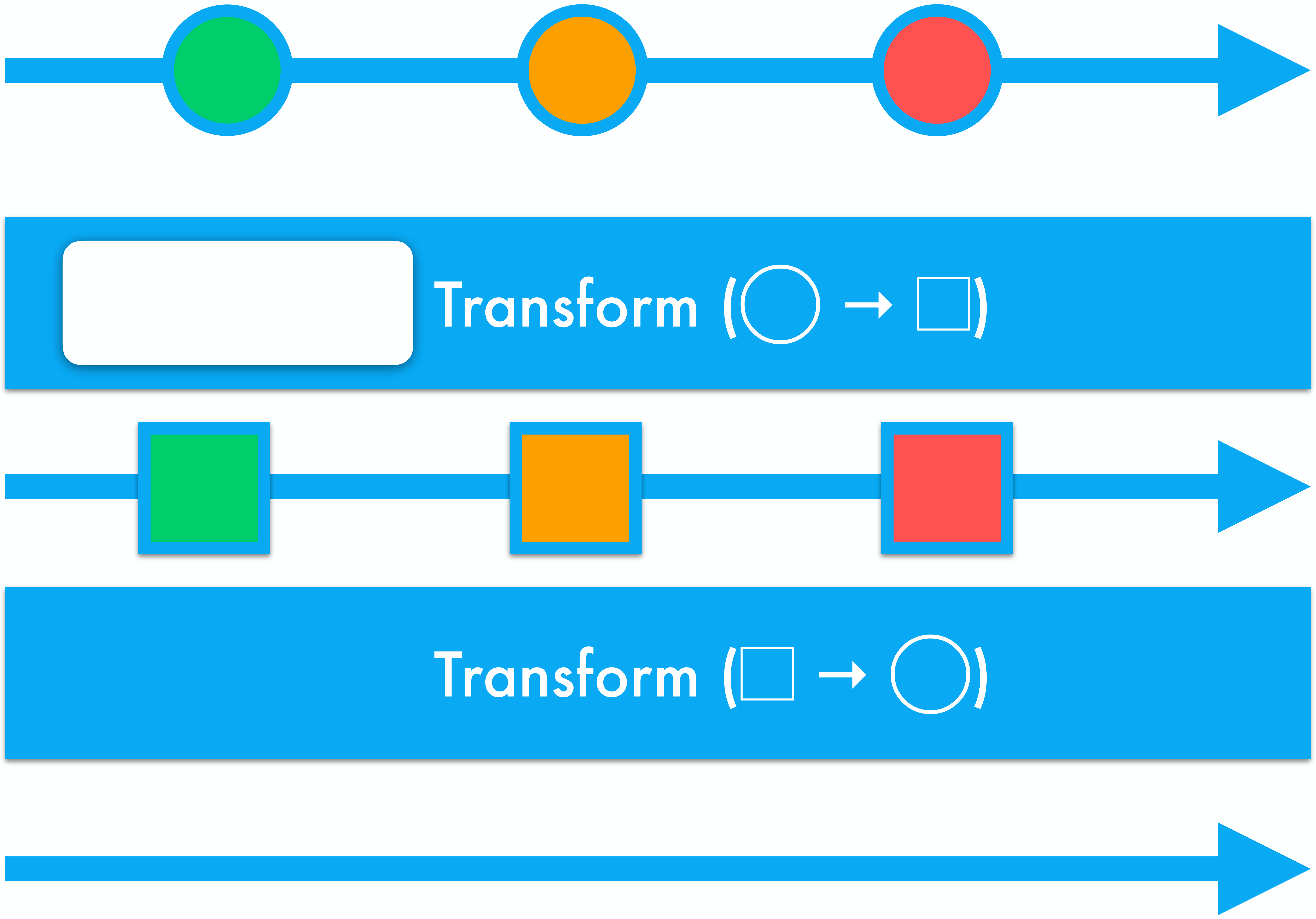

Transform (○ → □)
Transform (□ → ○)

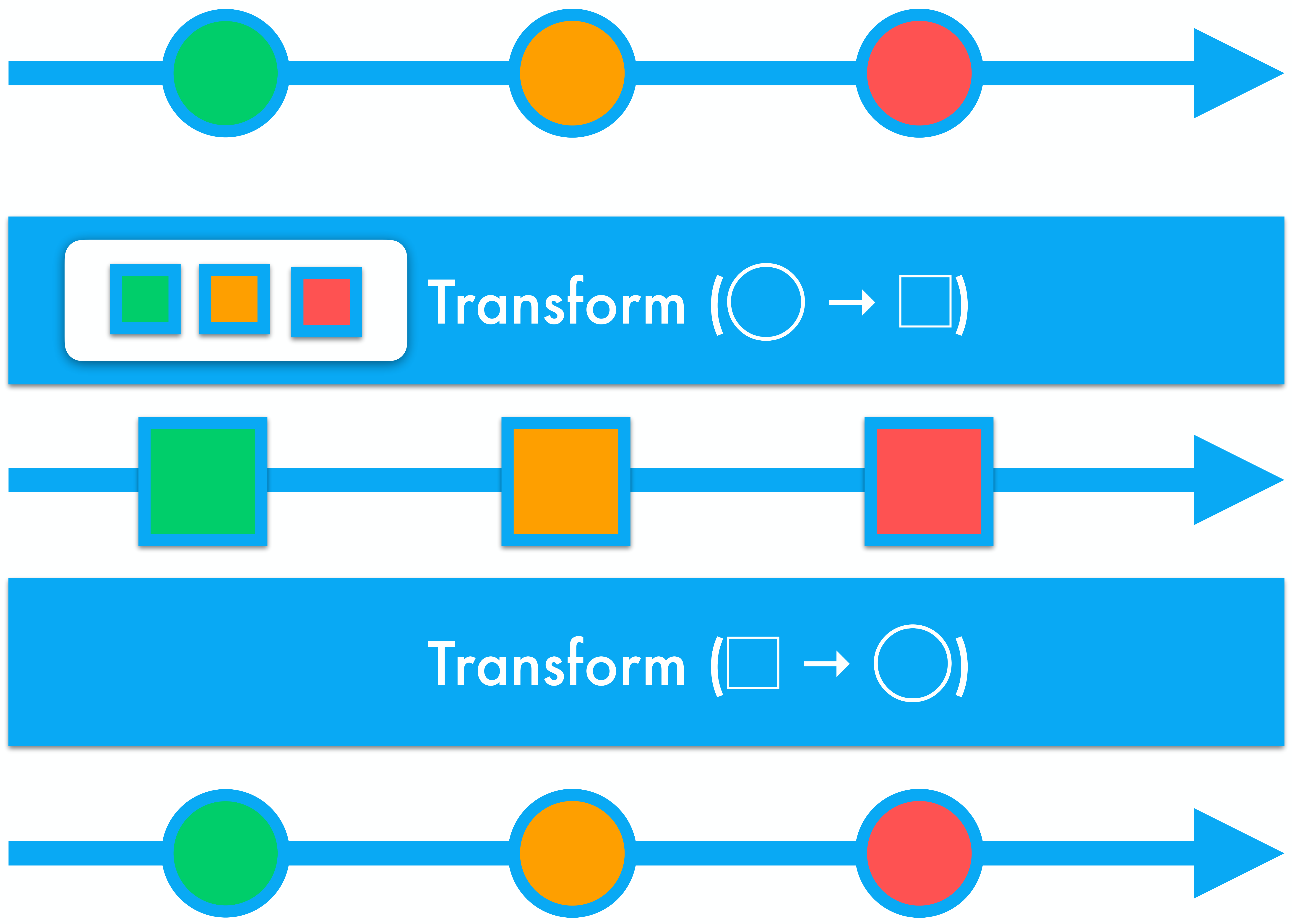
With fusion


Transform (○ → □)
Transform (□ → ○)


Transform (◯ → □)
Transform (□ → ◯)

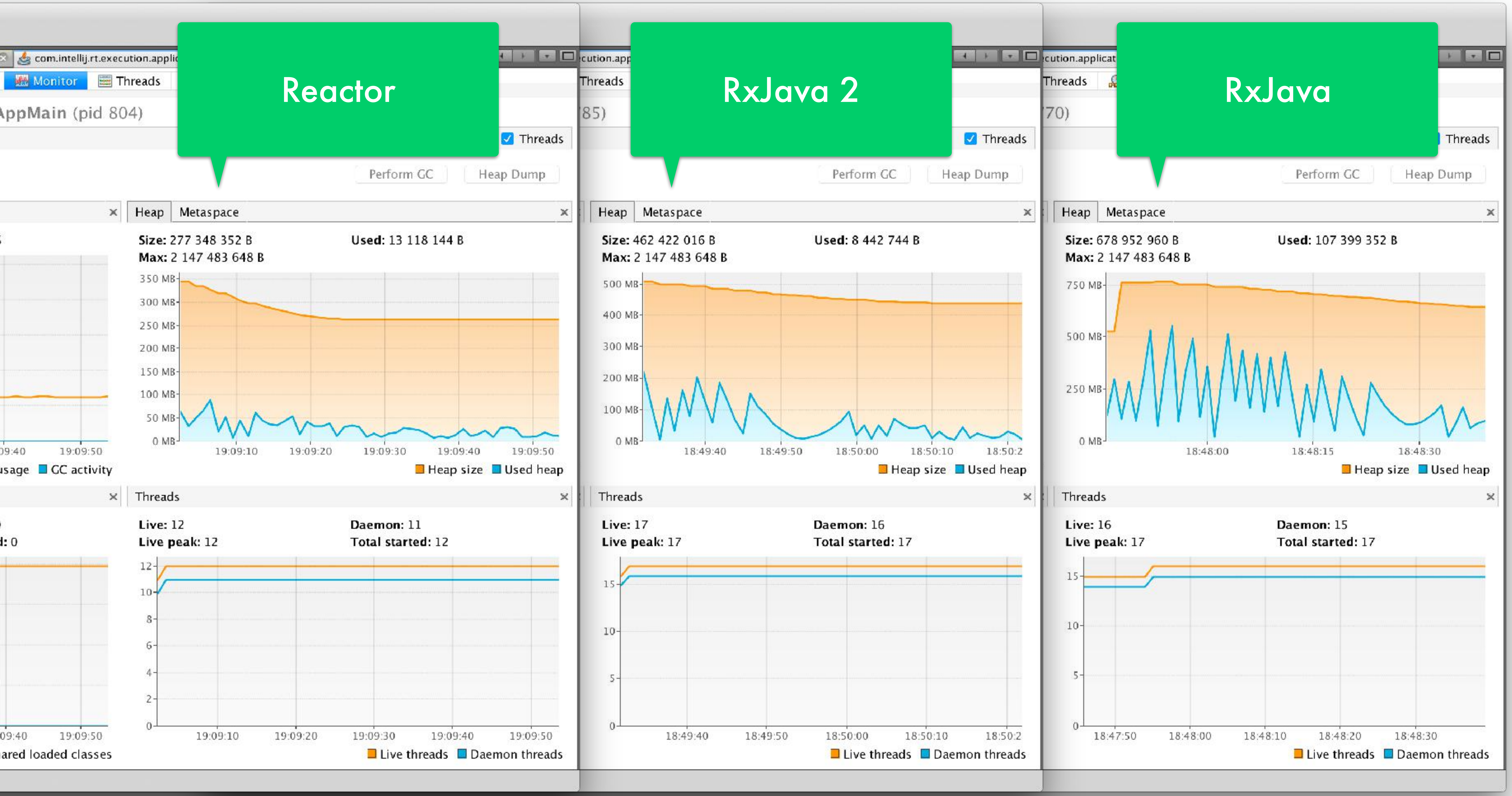

Transform (○ → □)
Transform (□ → ○)


Transform (○ → □)
Transform (□ → ○)


Transform (○ → □)
Transform (□ → ○)

Fusion *decreases* memory consumption and *increases* performances

```
for (int x = 0; x < 10_000; x++) {

  Observable.interval(10, TimeUnit.MILLISECONDS)
        .takeWhile(i -> take.get())
        .flatMap(i -> Observable.range(1, 100))
        .subscribe();
}
```



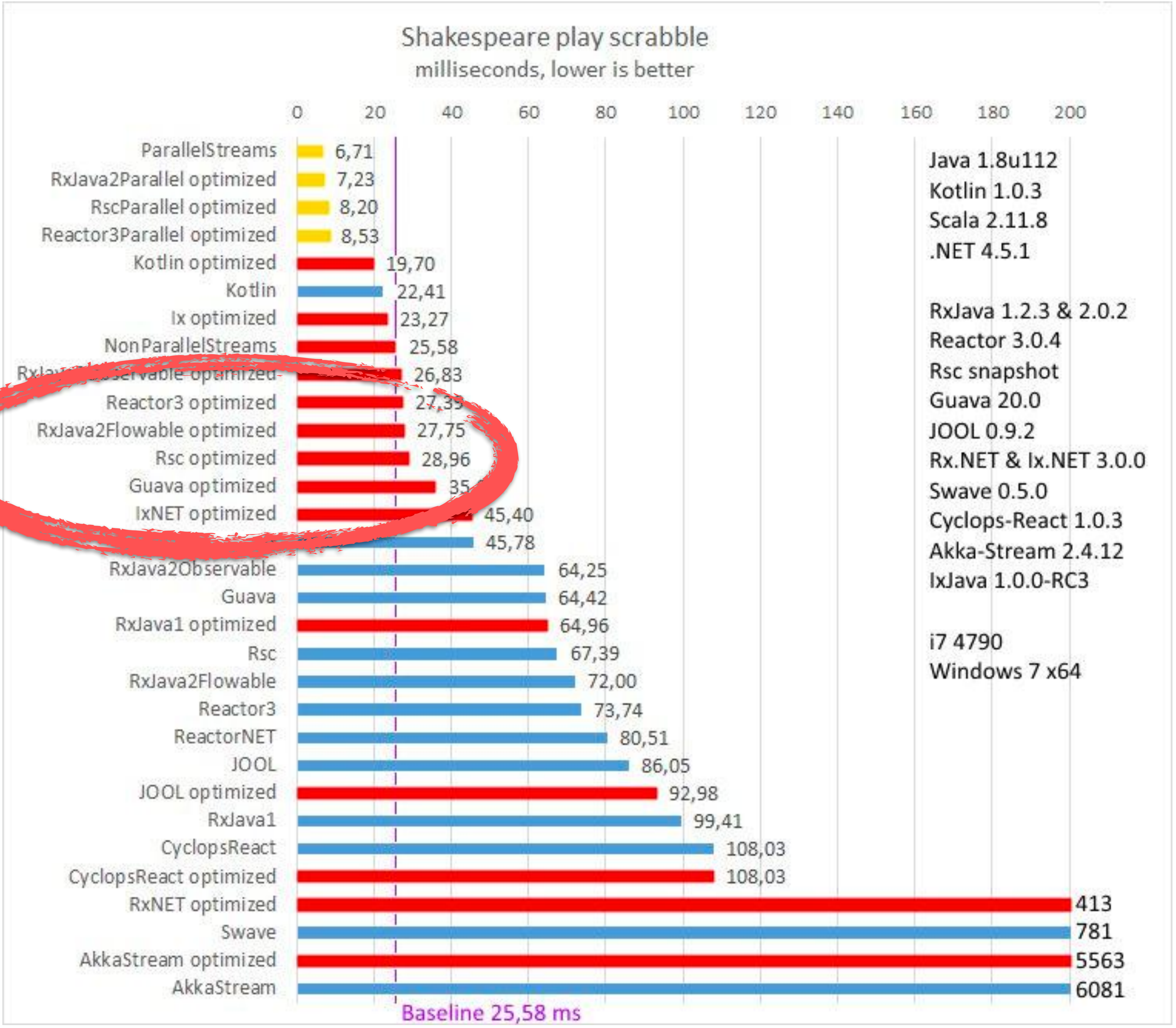
```
Reactor
Size: 277 348 352 B        Used: 13 118 144 B
Max: 2 147 483 648 B
Live: 12                   Daemon: 11
Live peak: 12              Total started: 12

RxJava 2
Size: 462 422 016 B        Used: 8 442 744 B
Max: 2 147 483 648 B
Live: 17                   Daemon: 16
Live peak: 17              Total started: 17

RxJava
Size: 678 952 960 B        Used: 107 399 352 B
Max: 2 147 483 648 B
Live: 16                   Daemon: 15
Live peak: 17              Total started: 17
```

December 2016

Java 8 Stream

RxJava 2 / Reactor

RxJava



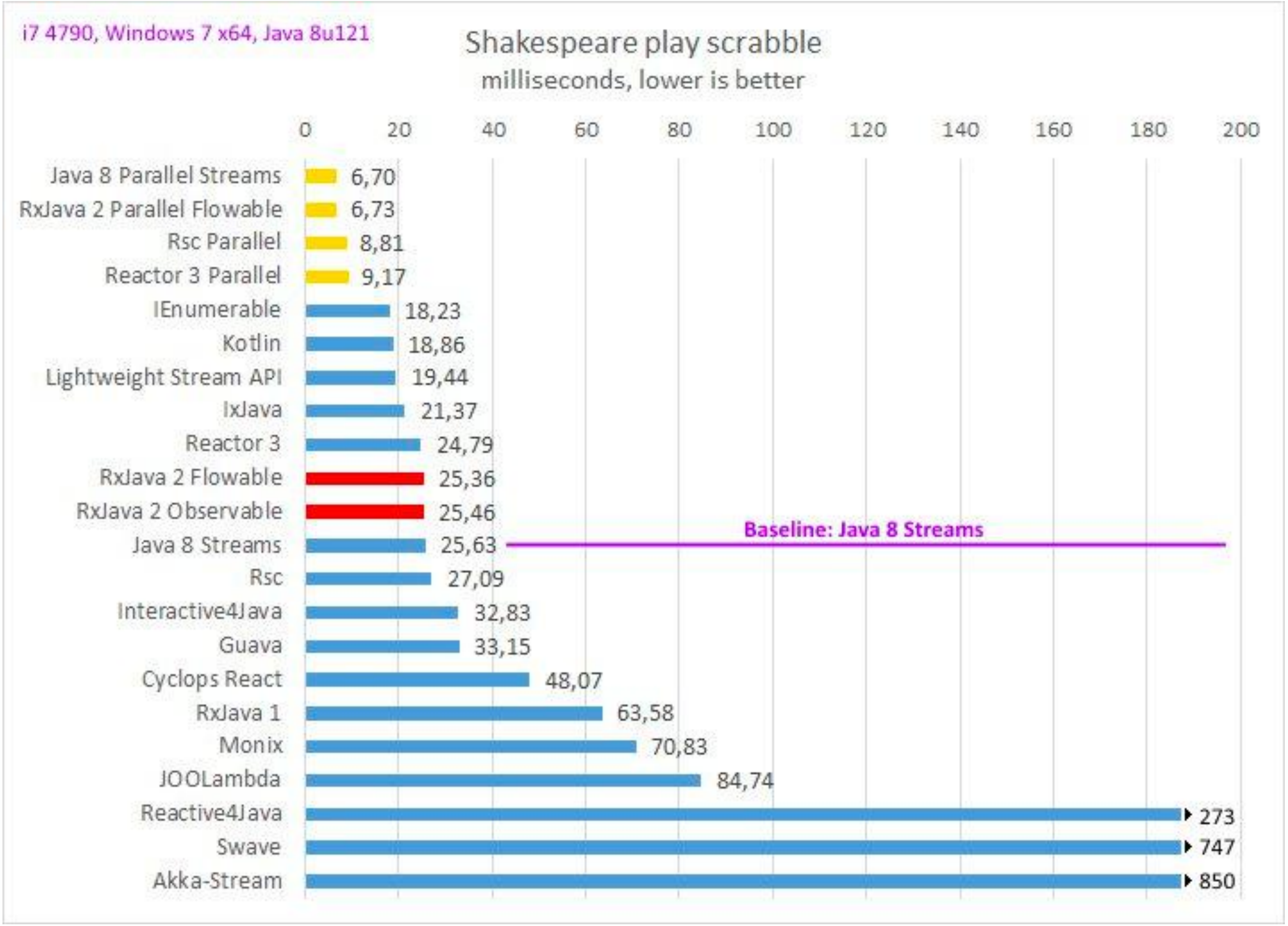
http://akarnokd.blogspot.fr/2016/12/the-reactive-scrabble-benchmarks.html

March 2017

RxJava 2 / Reactor

Java 8 Stream

RxJava



https://twitter.com/akarnokd/status/844555409012740096

Ecosystem

	RxJava	RxJava 2	Reactor	
Retrofit	Yes	Yes	No	Android
RxAndroid	Yes	Yes	No	
Realm	Yes	No	No	
Hystrix	Yes	No	No	
Couchbase	Yes	No	No	
MongoDB	Yes	No	No	
Spring Data 2.0	Yes	No	Yes	Spring
Reactor IPC	No	No	Yes	
WebFlux	No	Yes	Yes	

	RxJava	RxJava 2	Reactor
Retrofit	Yes	Yes	No
RxAndroid	Yes	Yes	No
Realm	Yes	No	No
Hystrix	Yes	No	No
Couchbase	Yes	No	No
MongoDB	Yes	No	No
Spring Data 2.0	Yes	No	Yes
Reactor IPC	No	No	Yes
WebFlux	No	Yes	Yes

Android

Spring

We are aggressively

migrating our internal code to

RxJava 2

https://github.com/uber/AutoDispose

	RxJava	RxJava 2	Reactor
Retrofit	Yes	Yes	No
RxAndroid	Yes	Yes	No
Realm	Yes	No	No
Hystrix	Yes	No	No
Couchbase	Yes	No	No
MongoDB	Yes	No	No
Spring Data 2.0	Yes	No	Yes
Reactor IPC	No	No	Yes
WebFlux	No	Yes	Yes

Android

Spring

Inertia

to migrate

	RxJava	RxJava 2	Reactor	
Retrofit	Yes	Yes	No	Android
RxAndroid	Yes	Yes	No	Android
Realm	Yes	No	No	
Hystrix	Yes	No	No	
Couchbase	Yes	No	No	
MongoDB	Yes	No	No	
Spring Data 2.0	Yes	No	Yes	Spring
Reactor IPC	No	No	Yes	Spring
WebFlux	No	Yes	Yes	Spring

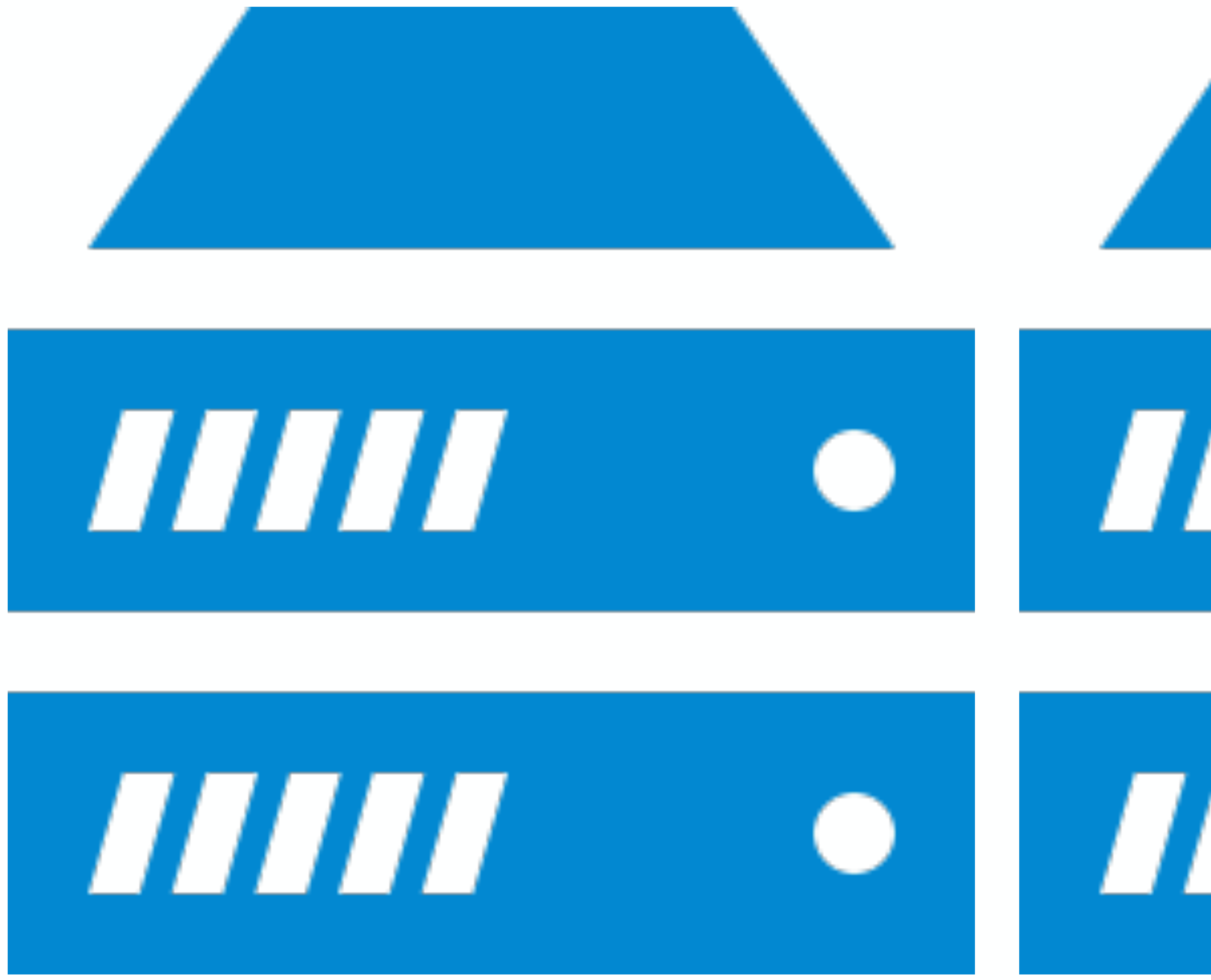
Spring 5 will *accelerate the adoption* of Reactor

Android

RxJava 2

Backend

RxJava 2



Spring

Reactor

Thanks for your attention

We stay in touch?

@dwursteisen

david.wursteisen@soat.fr

http://blog.soat.fr

Post your question on sli.do
(#K100)