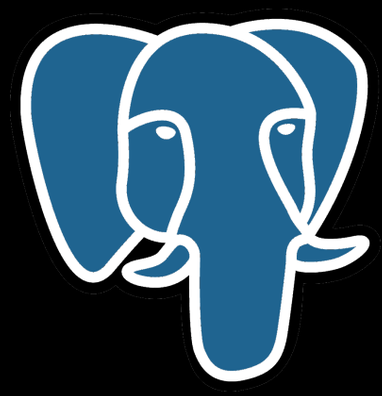
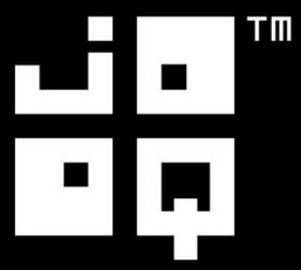
Tutorial de RxJava

Un tutorial de RxJava (2, obviamente)

@dovaleac **Pablo González Doval** dovaleac@gmail.com





VERT.X

ATLÁNTICA CONF

GRUPOMASMOVIL

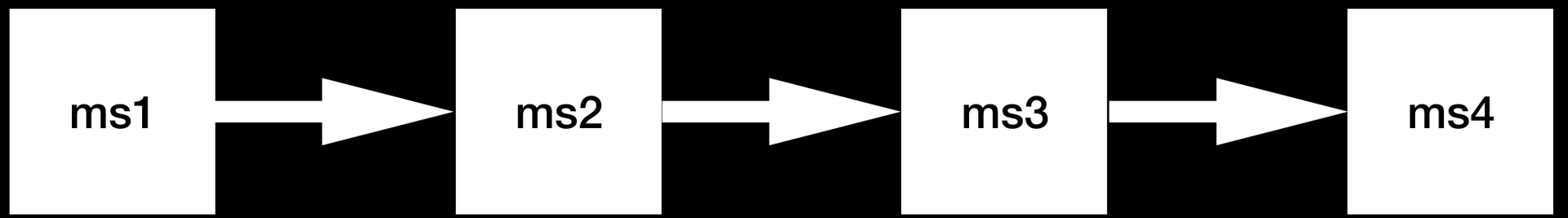
@dovaleac **Pablo González Doval** dovaleac@gmail.com

Qué es RxJava?

Un framework para construir aplicaciones reactivas con Java

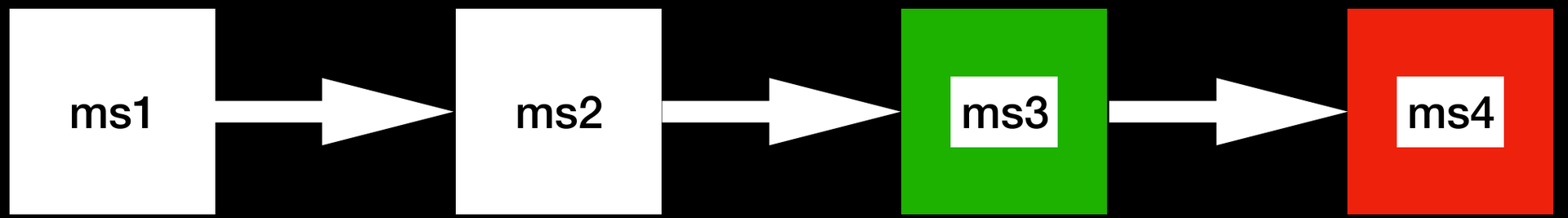
@dovaleac **Pablo González Doval** dovaleac@gmail.com

Porqué usamos RxJava?



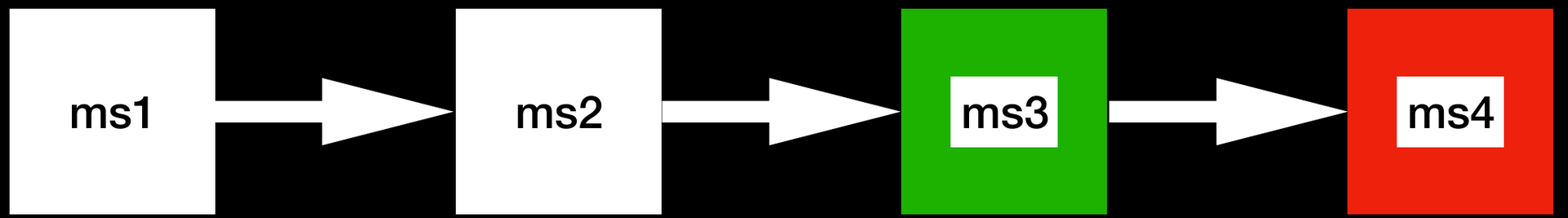
@dovaleac **Pablo González Doval** dovaleac@gmail.com

Porqué usamos RxJava?



@dovaleac **Pablo González Doval** dovaleac@gmail.com

Porqué usamos RxJava?



@dovaleac **Pablo González Doval** dovaleac@gmail.com

“Es el siguiente el que le pide datos al anterior, y es el anterior el que quiere que le pida datos el siguiente. MR”

Backpressure

@dovaleac **Pablo González Doval** dovaleac@gmail.com

¿Qué hace este código?

```
joinedFlowable
        .map(tuple -> tuple.acceptVisitor(tuplePrinter) + "\n")
        .sorted()
```

Operaciones terminales

- subscribe() / subscribe(***, Consumer<Throwable>)
- forEach(Consumer<T>)
- blockingGet()

Tipos de datos

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Tipos de datos

Estructura RxJava	Uso	Equivalente sin backpressure
Flowable<T>	Flujo de datos	Stream<T>
Single<T>	Un único elemento	CompletableFuture<T>
Completable	Consume tiempo pero no devuelve nada	CompletableFuture<Void>
Maybe<T>	Devuelve 0 o 1 elementos	CompletableFuture <Optional<T>>

Por qué Maybe?

- RxJava NO PERMITE EMITIR NULOS
- Artículo de Carlos Blé en kotlin: https://leanmind.es/es/blog/evitar-null-tambien-en-kotlin/
- Maybe ~ Single nullable
- Maybe ~ Single || Completable

Eventos

Estructura RxJava	onNext(t)	onSuccess(t)	onComplete()	onError(throwable)
Flowable<T>	N		1	0..1
Single<T>		1		0..1
Completable			1	0..1
Maybe<T>		0..1	0..1	0..1

Ejemplo práctico

anapioficeandfire.com

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Dominio

House

- id
- name
- region
- words
- lord
- overlord

Character

- id
- name
- titles
- playedBy

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Services

ReadService

- getAllCharacters()
- getAllHouses()
- getRandomCharacter()
- getRandomHouse()
- getCharacterById(int)
- getHouseById(int)
- getOverlordedByHouse(house)

WriteService

- addNewHouse
- addNewCharacter
- changeHouseRuler

Creando entidades

@dovaleac **Pablo González Doval** dovaleac@gmail.com

***.just()

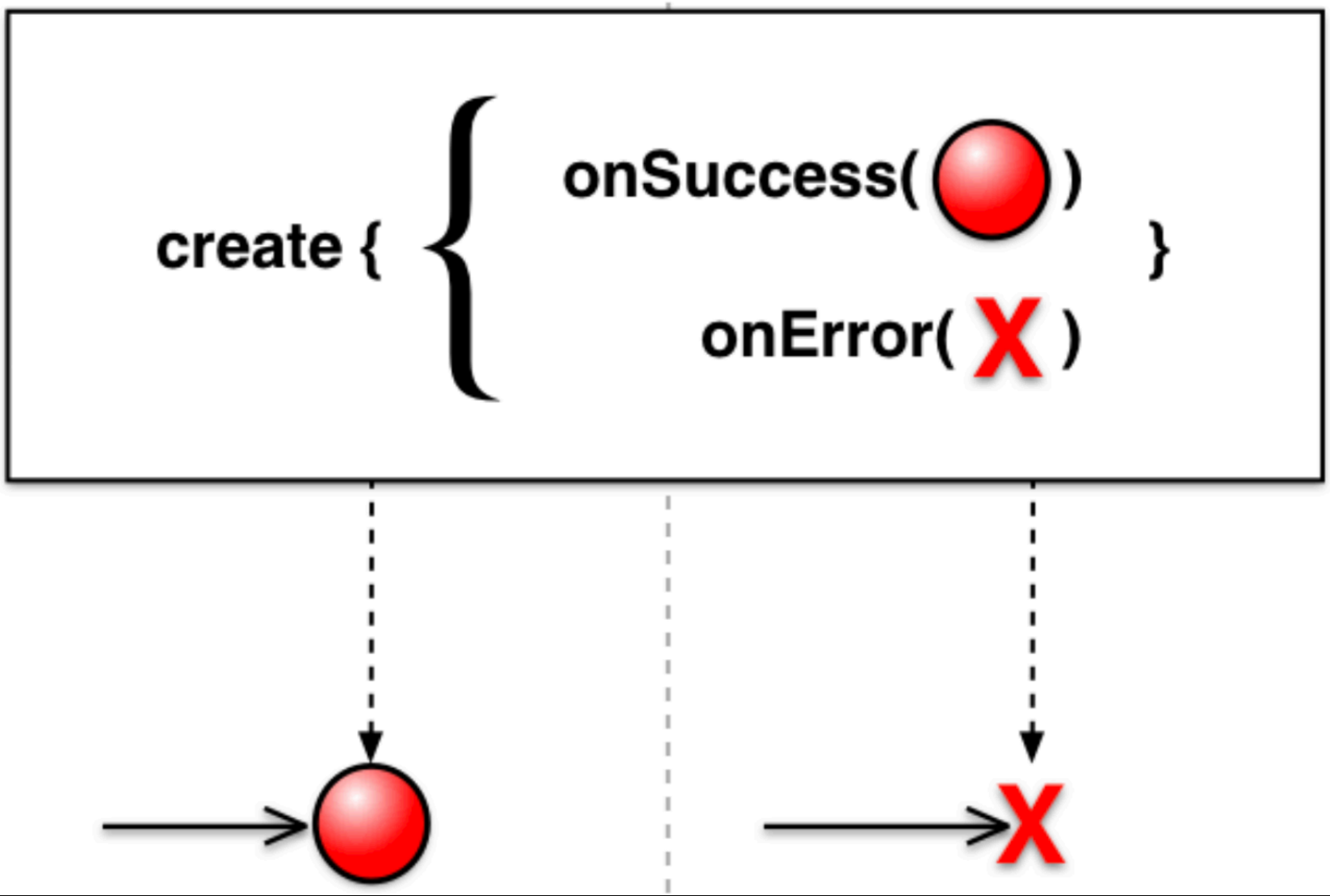
Encapsular un valor como un Single/Flowable/Maybe

```
public Single<String> sayHelloWithSingle() {
  return Single.just("Hello!");
}
```

***.create()

```
public static <T> Single<T> create(SingleOnSubscribe<T> source)
```

Provides an API (via a cold Single) that bridges the reactive world with the callback-style world.



Eventos

Estructura RxJava	onNext(t)	onSuccess(t)	onComplete()	onError(throwable)
Flowable<T>	N		1	0..1
Single<T>		1		0..1
Completable			1	0..1
Maybe<T>		0..1	0..1	0..1

***.create()

Emitir manualmente los eventos de un Single

```
public <T> Single<T> createSingleFromCompletableFuture(CompletableFuture<T> completableFuture) {
  return Single.create(singleEmitter -> {
    try {
      completableFuture.thenAcceptAsync(singleEmitter::onSuccess);
    } catch (Exception e) {
      singleEmitter.onError(e);
    }
  });
}
```

Eventos

Estructura RxJava	onNext(t)	onSuccess(t)	onComplete()	onError(throwable)
Flowable<T>	N		1	0..1
Single<T>		1		0..1
Completable			1	0..1
Maybe<T>		0..1	0..1	0..1

***.create()

Emitir manualmente los eventos de un Completable

```
public Completable createCompletableFromCompletableFuture(
    CompletableFuture<Void> completableFuture) {
  return Completable.create(completableEmitter -> {
    try {
      completableFuture.thenAcceptAsync(aVoid -> completableEmitter.onComplete());
    } catch (Exception e) {
      completableEmitter.onError(e);
    }
  });
}
```

Eventos

Estructura RxJava	onNext(t)	onSuccess(t)	onComplete()	onError(throwable)
Flowable<T>	N		1	0..1
Single<T>		1		0..1
Completable			1	0..1
Maybe<T>		0..1	0..1	0..1

***.create()

Emitir manualmente los eventos de un Maybe

```
public <T> Maybe<T> createMaybeFromCompletableFuture(
    CompletableFuture<Optional<T>> completableFuture) {
  return Maybe.create(maybeEmitter -> {
    try {
      completableFuture.thenAcceptAsync(t ->
          t.ifPresentOrElse(
              maybeEmitter::onSuccess,
              maybeEmitter::onComplete)
      );
    } catch (Exception e) {
      maybeEmitter.onError(e);
    }
  });
}
```

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Eventos

Estructura RxJava	onNext(t)	onSuccess(t)	onComplete()	onError(throwable)
Flowable<T>	N		1	0..1
Single<T>		1		0..1
Completable			1	0..1
Maybe<T>		0..1	0..1	0..1

***.create()

Emitir manualmente los eventos de un Flowable

```java
public <T> Flowable<T> createFlowableFromStream(Stream<T> stream) {
  return Flowable.create(flowableEmitter -> {
    try {
      stream.forEach(flowableEmitter::onNext);
    } catch (Exception e) {
      flowableEmitter.onError(e);
    }
    flowableEmitter.onComplete();
  }, BackpressureStrategy.BUFFER);
}
```

@dovaleac **Pablo González Doval** dovaleac@gmail.com

create vs just

Qué opción sería más preferible?

```
public abstract class CreateVsJust<T> {

  protected abstract T getT(); //potentially costly

  public Single<T> getSingleByJust() {
    return Single.just(getT());
  }

  public Single<T> getSingleByCreate() {
    return Single.create(emitter -> {
          try {
            T t = getT();
            emitter.onSuccess(t);
          } catch (Exception e) {
            emitter.onError(e);
          }
        }
    );
  }
}
```

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Consumiendo entidades

@dovaleac **Pablo González Doval** dovaleac@gmail.com

.subscribe()

Cómo reaccionar ante cada evento

Evento	Consumidor
onNext(t)	Consumer<T>
onSuccess(t)	Consumer<T>
onComplete()	Action
onError(throwable)	Consumer<Throwable>

@dovaleac **Pablo González Doval** dovaleac@gmail.com

.subscribe()

Sobrecargado pero fácil de comprender

Disposable	**subscribe**() Subscribes to a Single but ignore its emission or notification.
Disposable	**subscribe**(**BiConsumer**<? super **T**,? super **Throwable**> onCallback) Subscribes to a Single and provides a composite callback to handle the item it emits or any error notification it issues.
Disposable	**subscribe**(**Consumer**<? super **T**> onSuccess) Subscribes to a Single and provides a callback to handle the item it emits.
Disposable	**subscribe**(**Consumer**<? super **T**> onSuccess, **Consumer**<? super **Throwable**> onError) Subscribes to a Single and provides callbacks to handle the item it emits or any error notification it issues.
void	**subscribe**(**SingleObserver**<? super **T**> observer) Subscribes the given SingleObserver to this SingleSource instance.
protected abstract void	**subscribeActual**(**SingleObserver**<? super **T**> observer) Implement this method in subclasses to handle the incoming **SingleObserver**s.
Single<**T**>	**subscribeOn**(**Scheduler** scheduler) Asynchronously subscribes subscribers to this Single on the specified **Scheduler**.

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Transformando entidades

@dovaleac **Pablo González Doval** dovaleac@gmail.com

```java
protected abstract Single<T> getT();

protected abstract U processT(T t);

public U returnU() {
  Single<T> t = getT();
  return processT( t: null);
}
```

Cómo puedo llamar a processT?

@dovaleac **Pablo González Doval** dovaleac@gmail.com

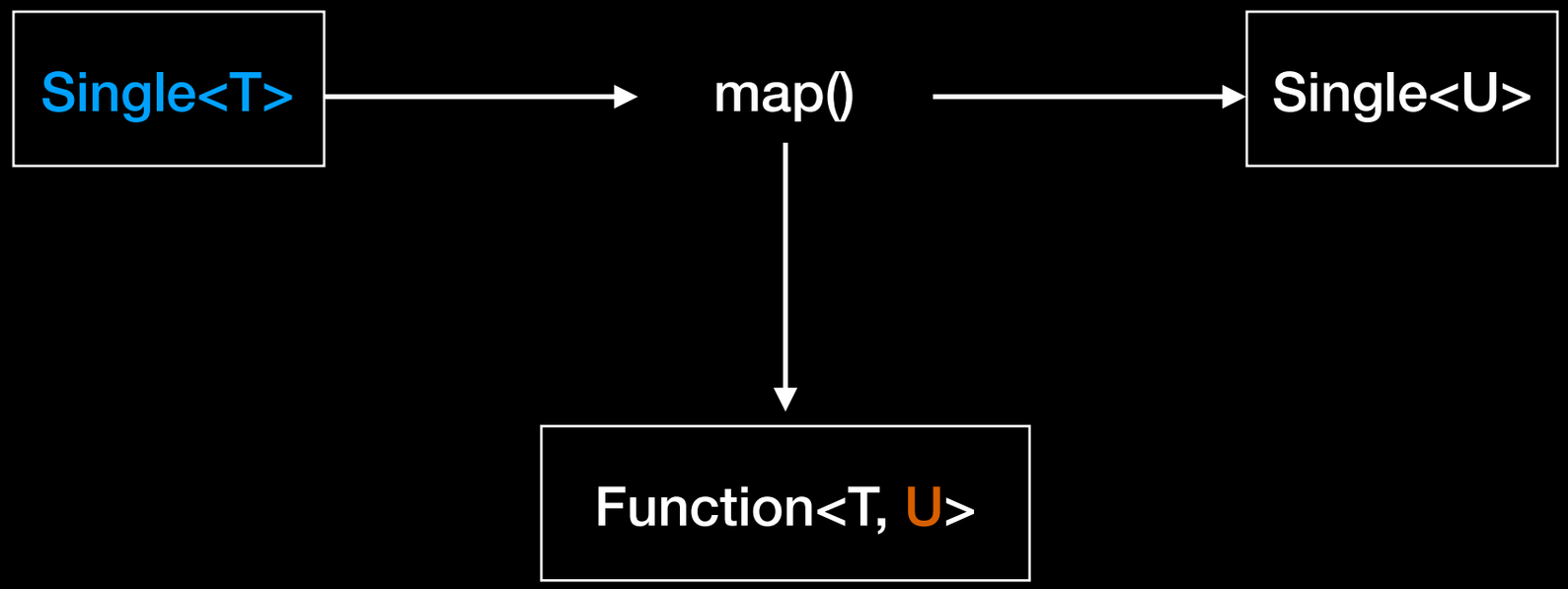
```java
protected abstract Single<T> getT();

protected abstract U processT(T t);

public Single<U> returnUSingle() {
    return getT()
            .map(this::processT);
}
```

Usamos map()

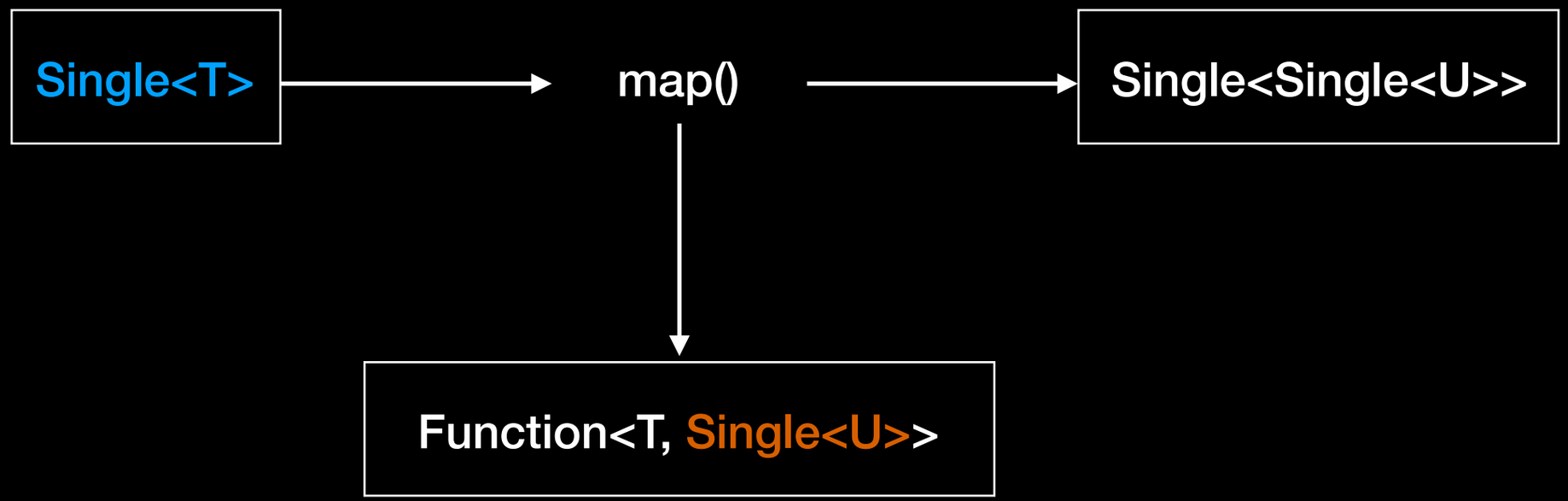
Dentro del map ejecutamos la operación

@dovaleac **Pablo González Doval** dovaleac@gmail.com

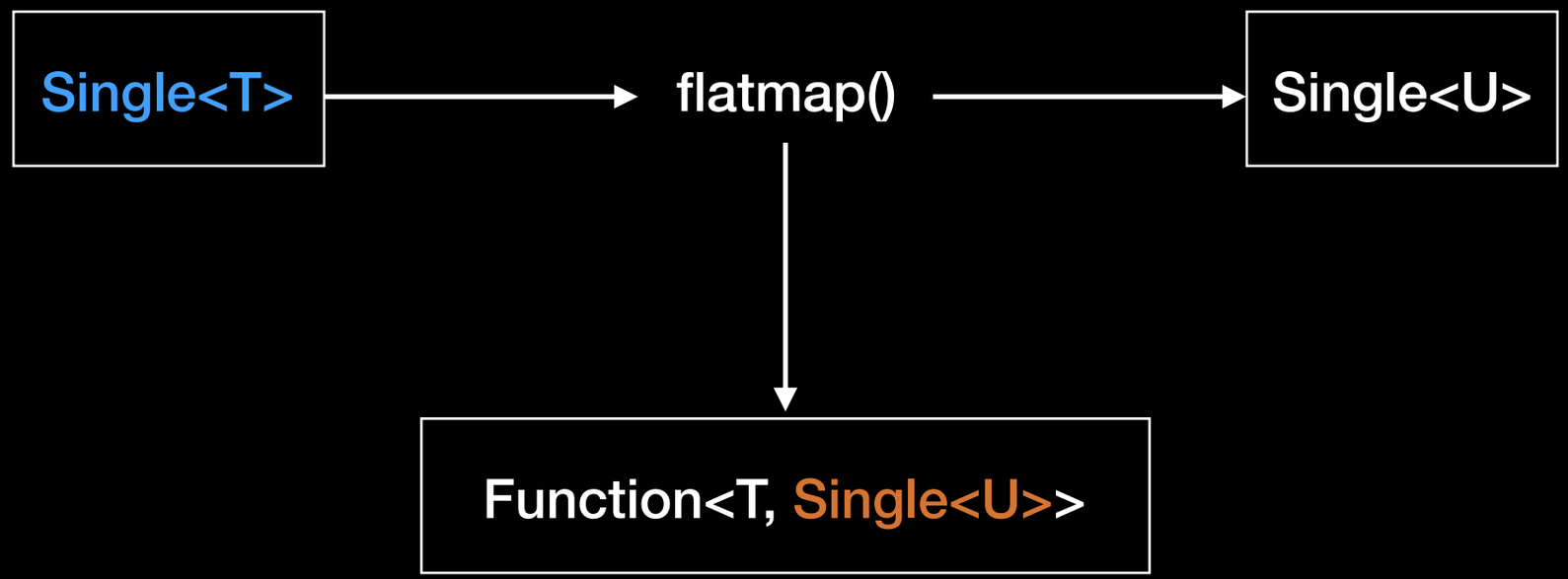
Map



Map

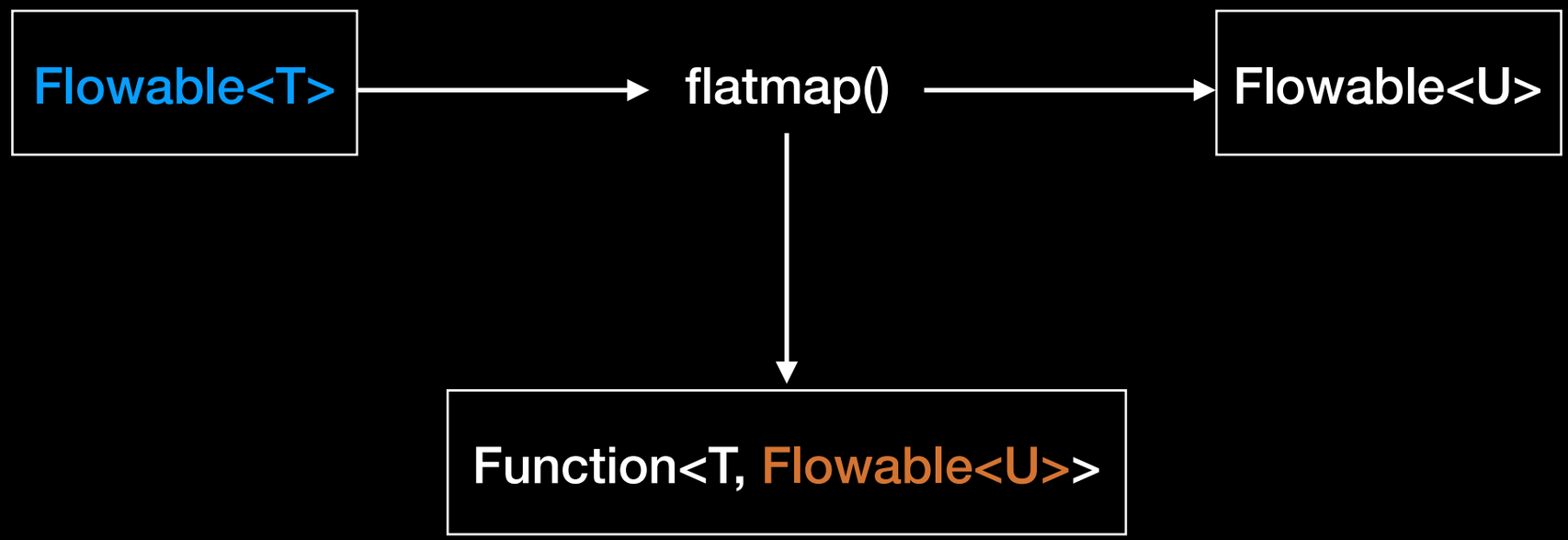


Flatmap

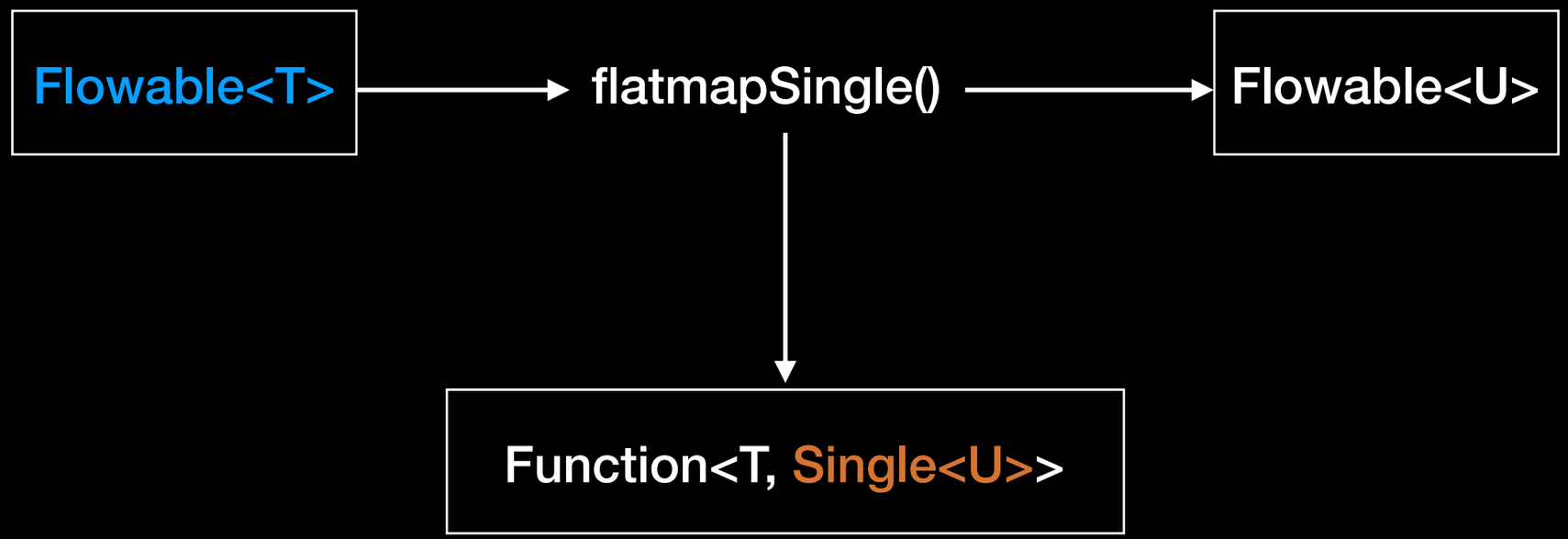


@dovaleac **Pablo González Doval** dovaleac@gmail.com

Flatmap



Flatmap



@dovaleac **Pablo González Doval** dovaleac@gmail.com

Flatmap & company

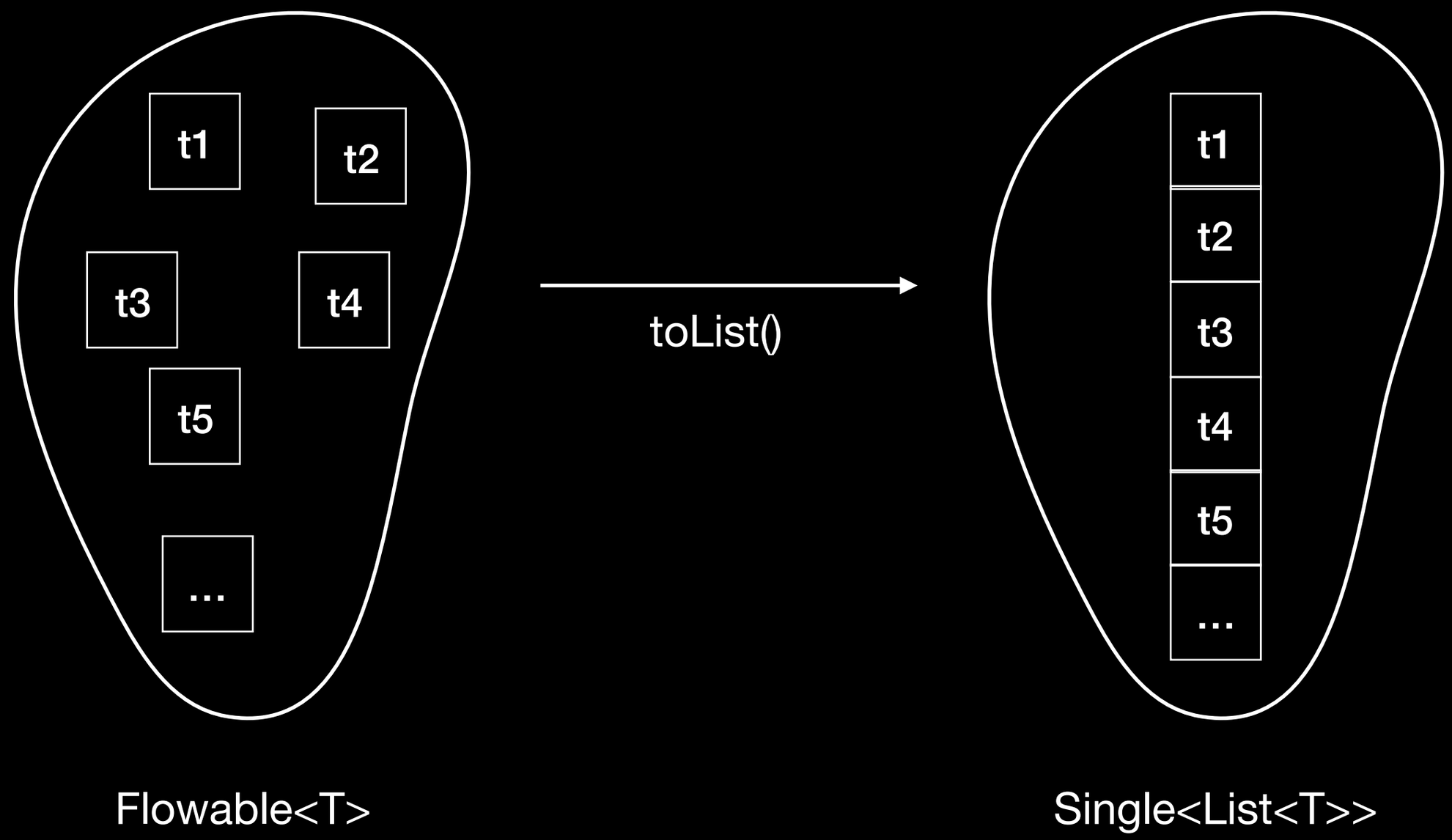
	Flowable<U>	Single<U>	Maybe<U>	Completable
Flowable<T>	flatMap -> Flowable<U>	flatMapSingle -> Flowable<U>	flatMapMaybe -> Flowable<U>	flatMapCompletable -> Completable
Single<T>	flatMapPublisher -> Flowable<U>	flatMap -> Single<U>	flatMapMaybe -> Maybe<U>	flatMapCompletable -> Completable
Maybe<T>	flatMapPublisher -> Flowable<U>	flatMapSingleElement -> Maybe<U>, flatMapSingle -> Single<U>*	flatMap -> Maybe<U>	flatMapCompletable -> Completable
Completable	andThen -> Flowable<U>	andThen -> Single<U>	andThen -> Maybe<U>	andThen -> Completable

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Otras transformaciones

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Flowables a Collections



Flowable<T>

Single<List<T>>

@dovaleac

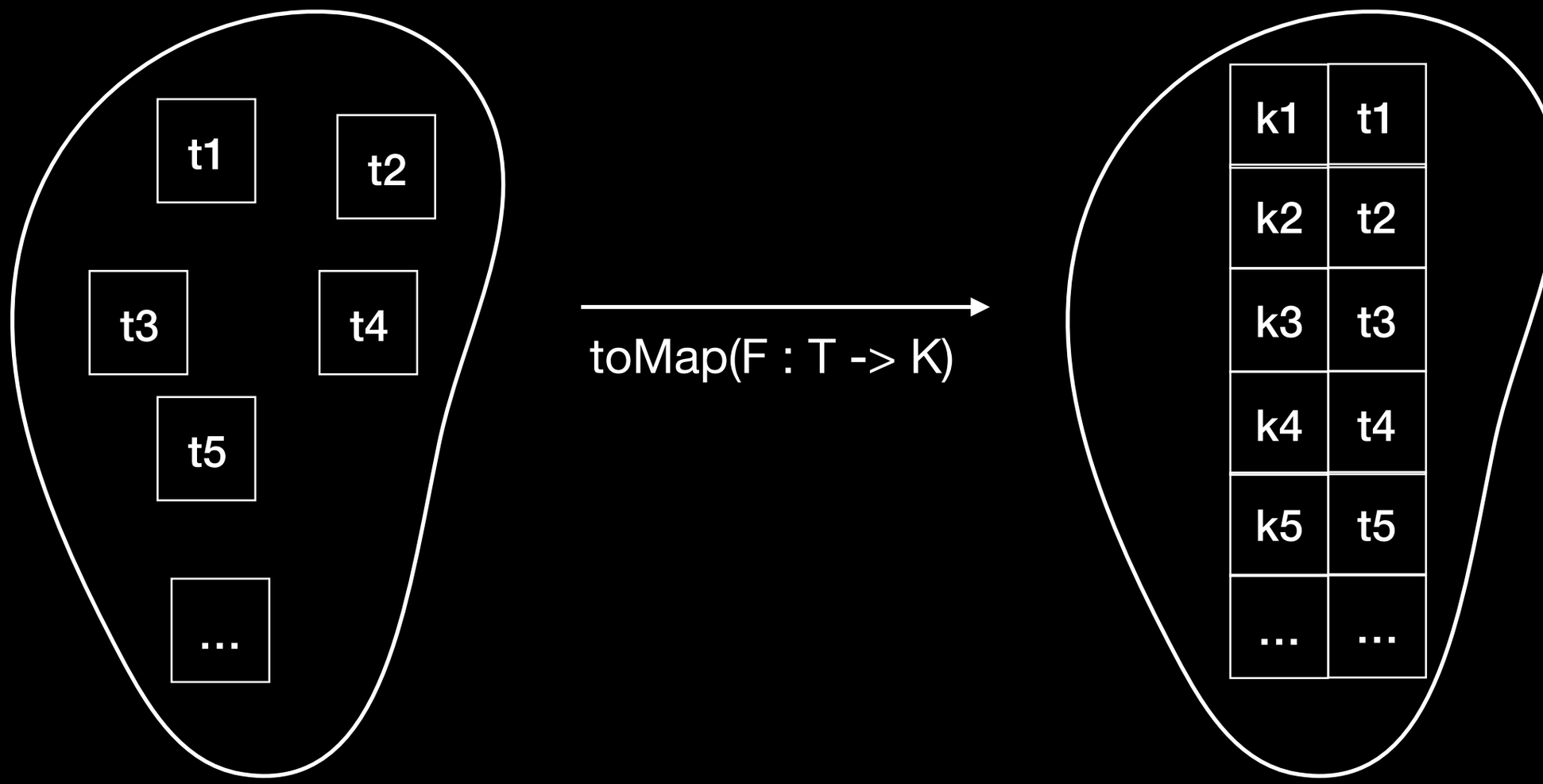
Pablo González Doval

dovaleac@gmail.com

Los Flowables, como los Streams,
no se pueden consumir 2 veces

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Flowables a Collections

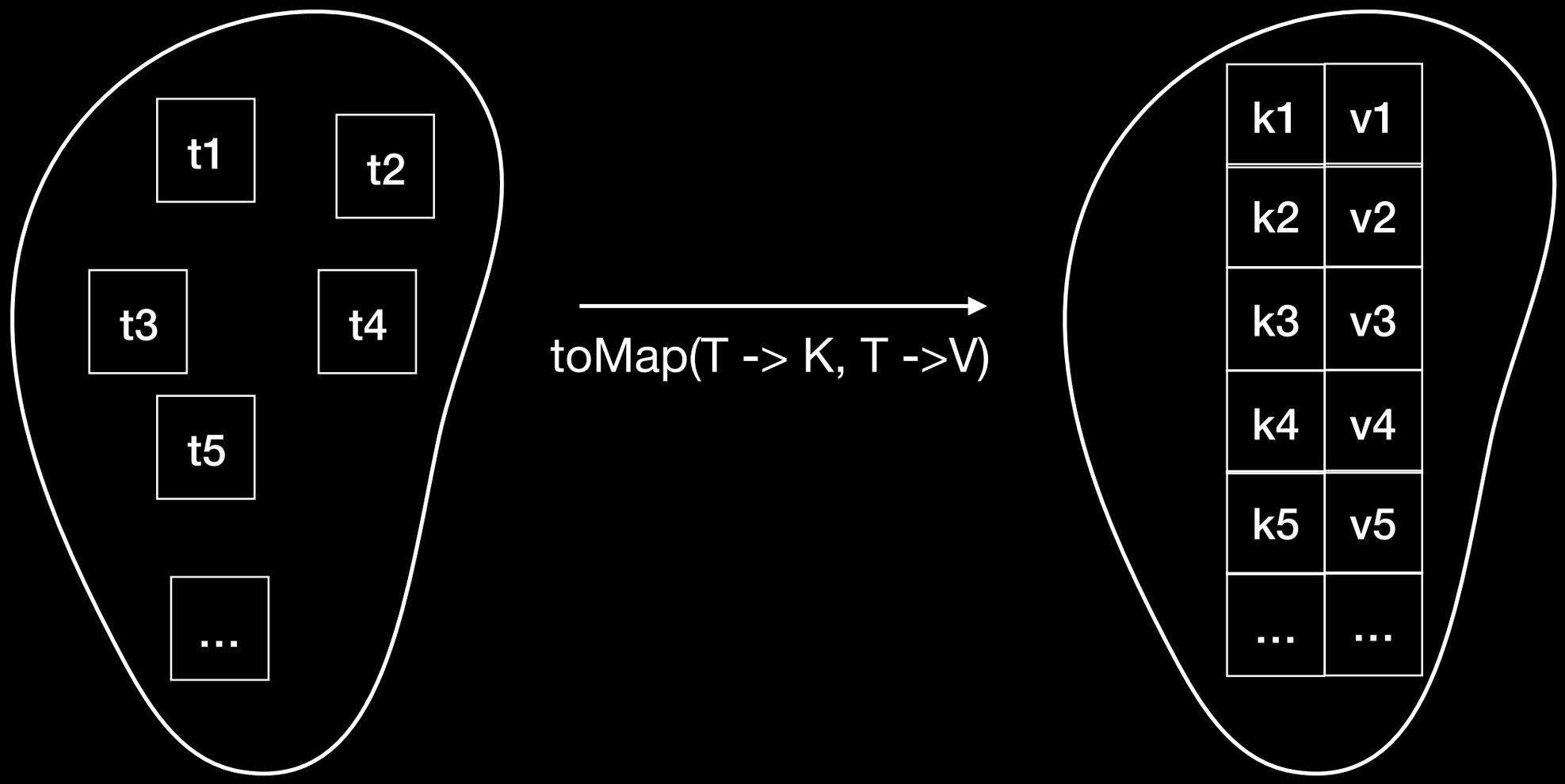


Flowable<T>

Single<Map<K, T>>

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Flowables a Collections

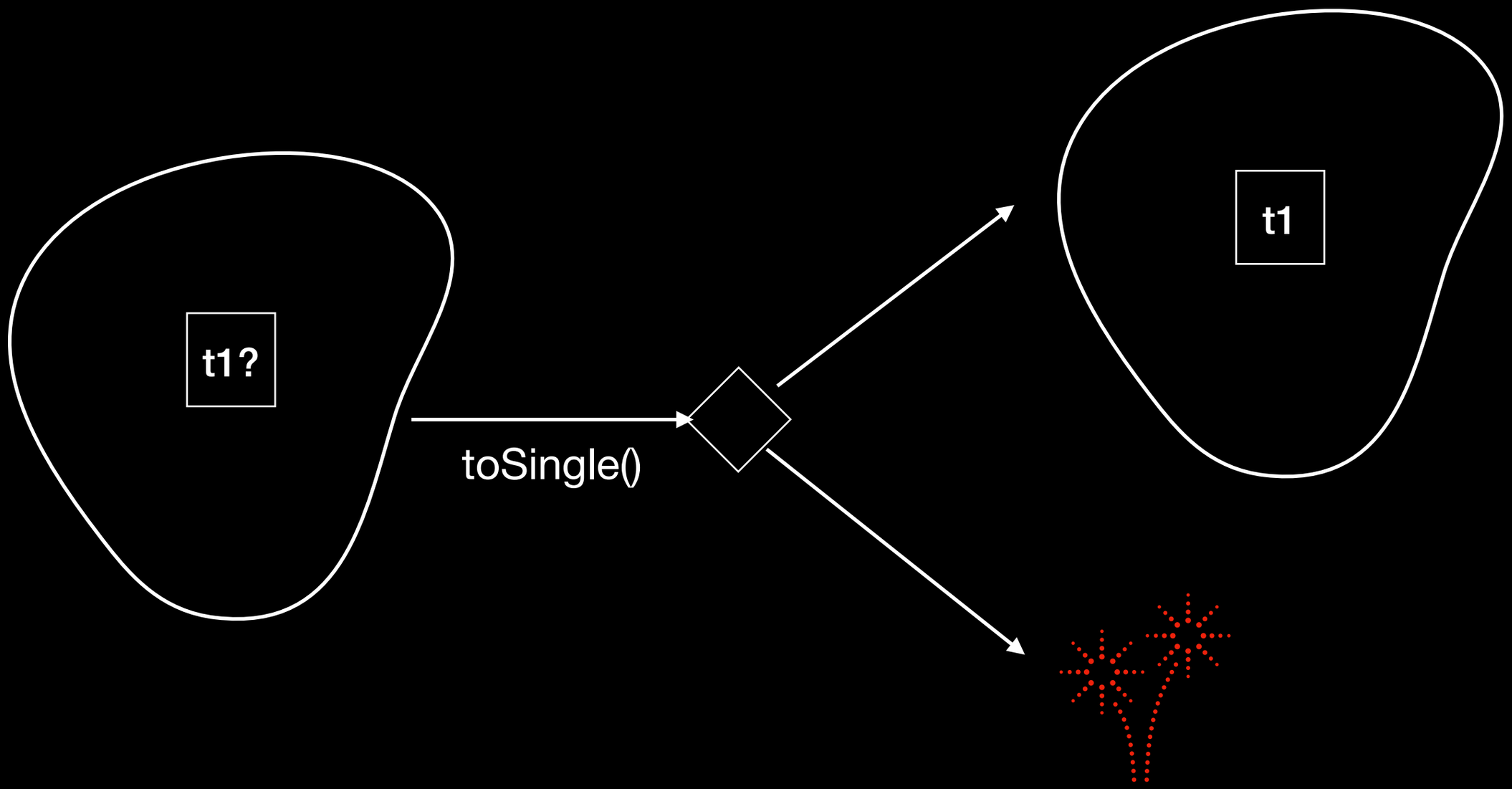


Flowable<T>

Single<Map<K, V>>

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Maybe a Single



Maybe<T>

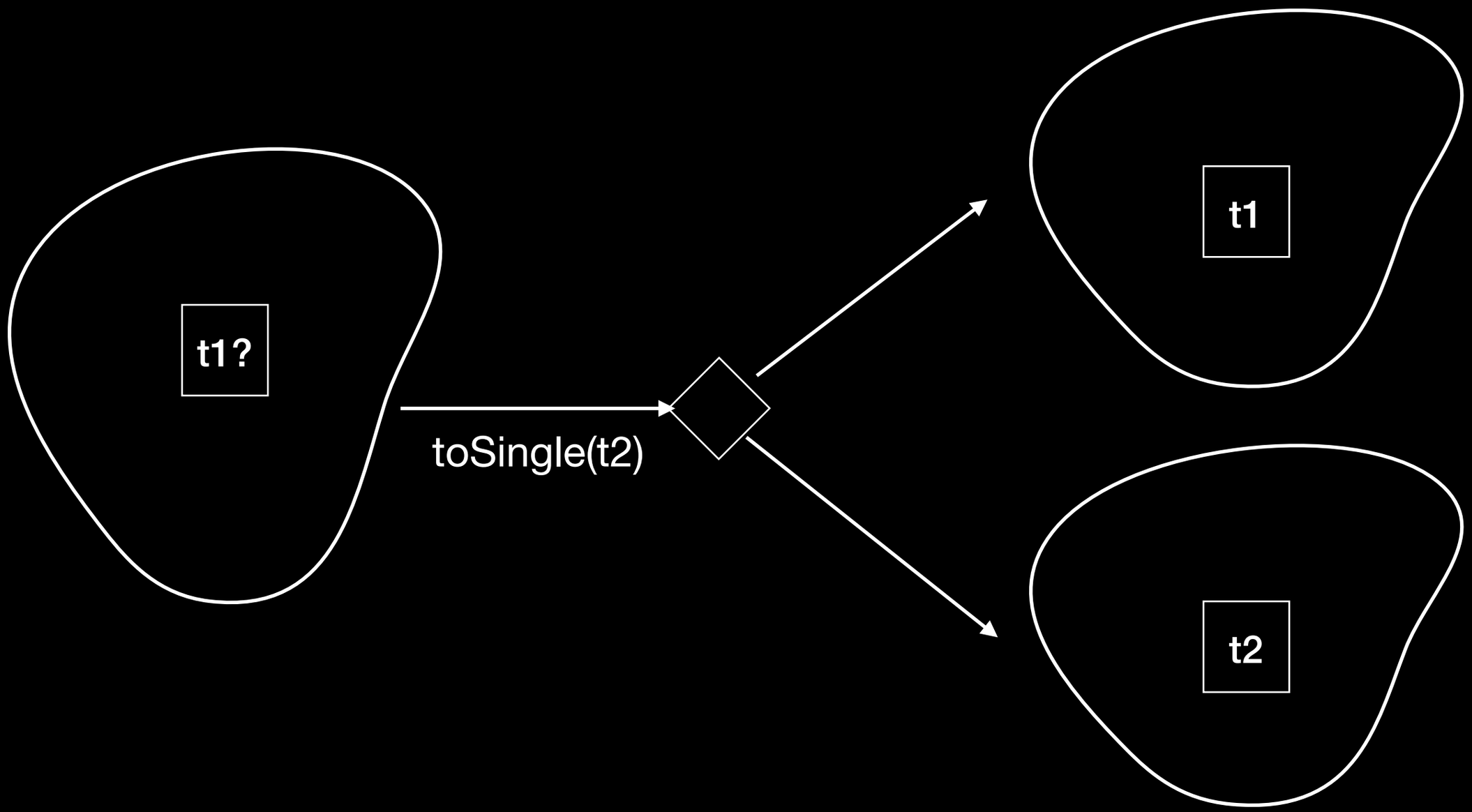
Single<T>

@dovaleac

Pablo González Doval

dovaleac@gmail.com

Maybe a Single

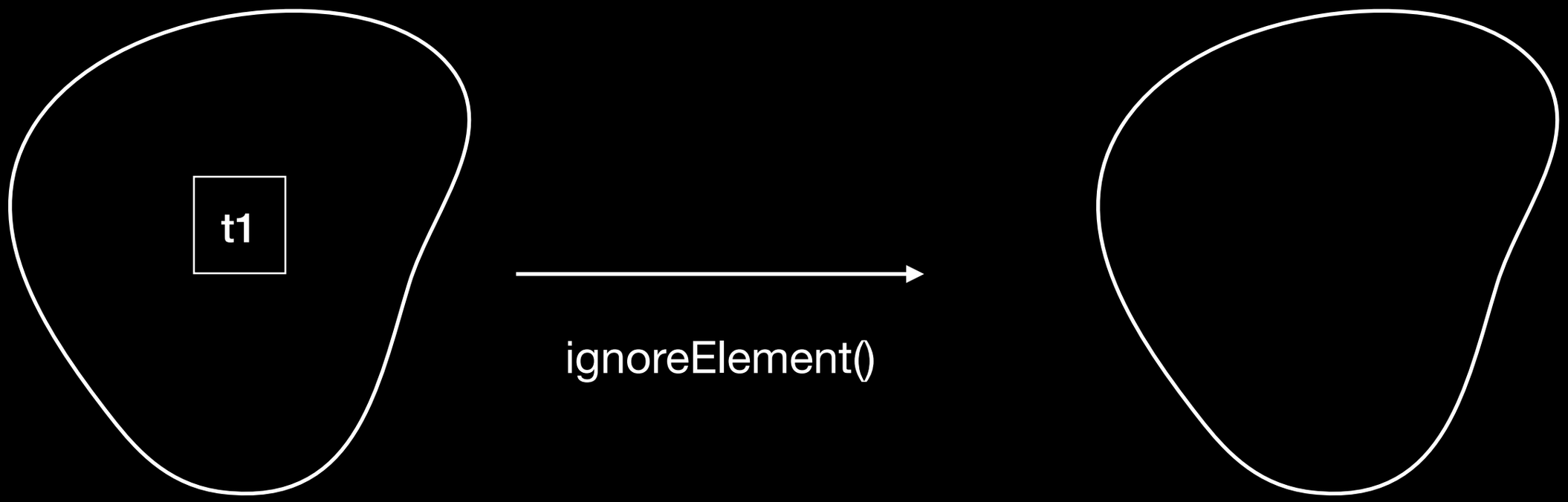


Maybe<T>

Single<T>

@dovaleac **Pablo González Doval** dovaleac@gmail.com

*** a Completable



Single<T>

Completable

@dovaleac

Pablo González Doval

dovaleac@gmail.com

Otras formas de crear

@dovaleac **Pablo González Doval** dovaleac@gmail.com

General

- *.empty()
- *.complete()
- *.never()

Flowables

- fromIterable(): a partir de una lista
- generate(): generación inductiva (caso 0 + f(N -> N + 1))
- range(): rango de números

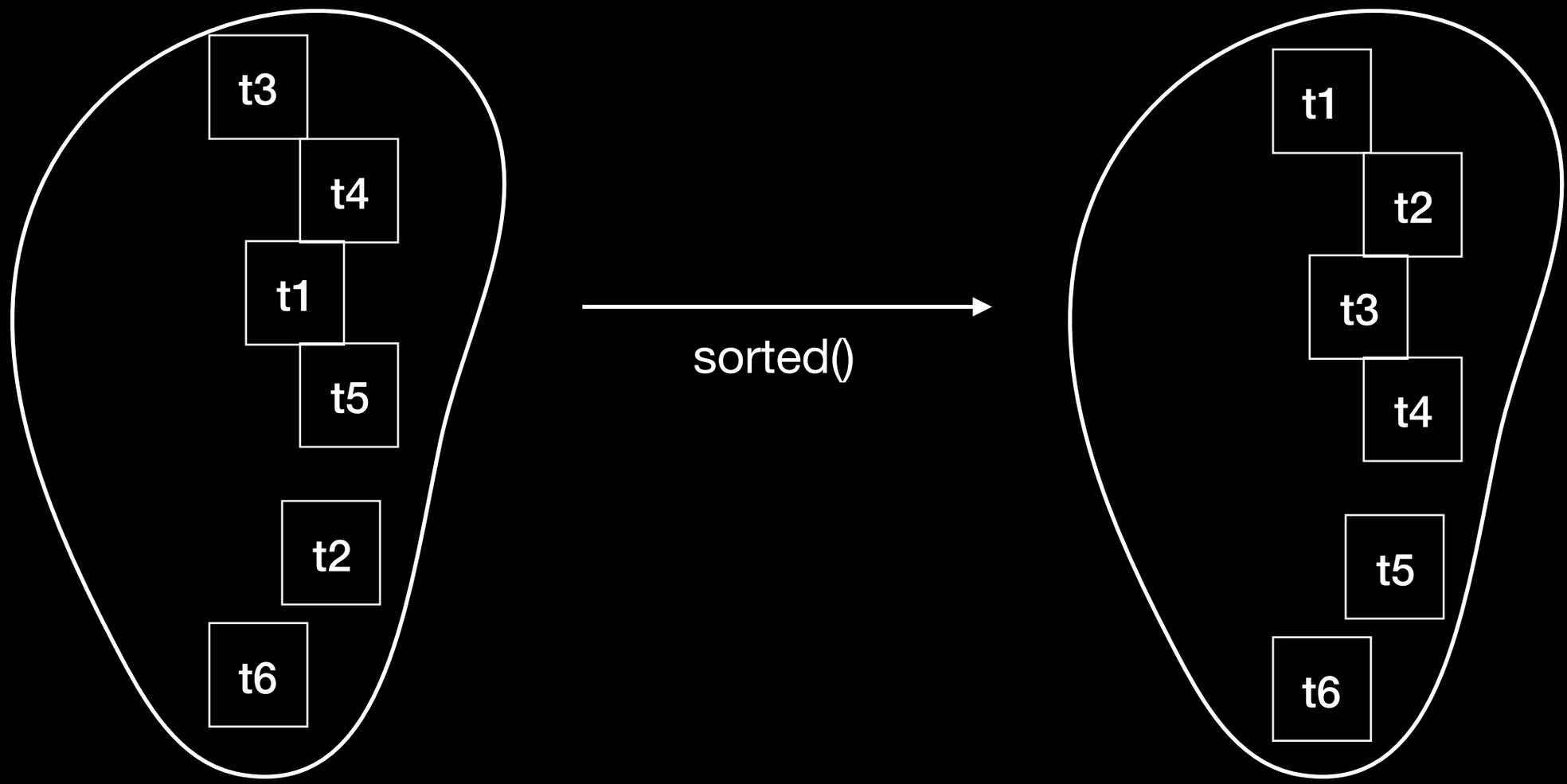
Completables

- andThen(): en serie
- merge(): varios en paralelo, todos deben acabar
- amb(): varios en paralelo, llega con que acabe uno

Otros

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Flowable.sorted()

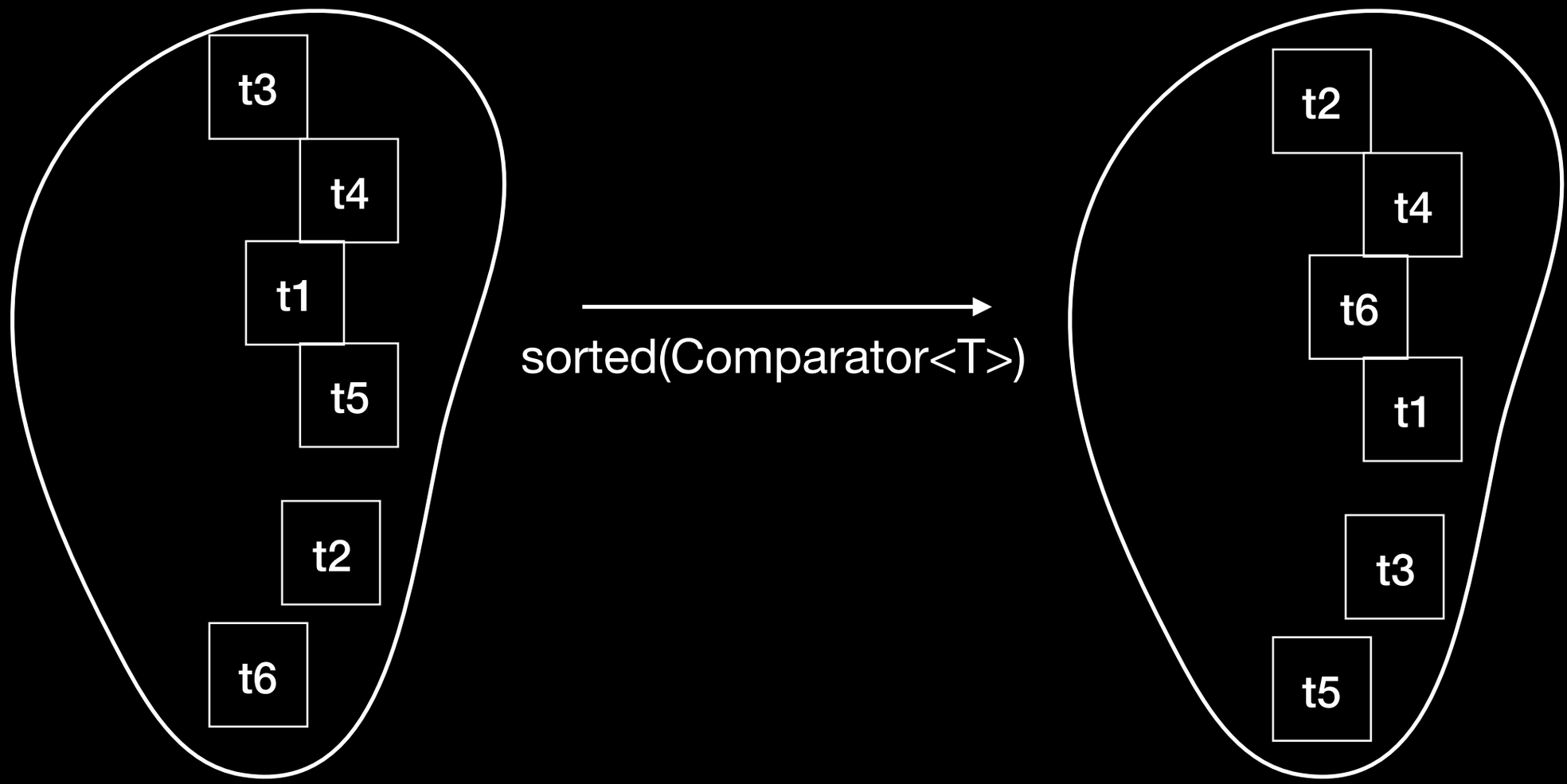


Flowable<T>

Flowable<List<T>>

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Flowable.sorted()

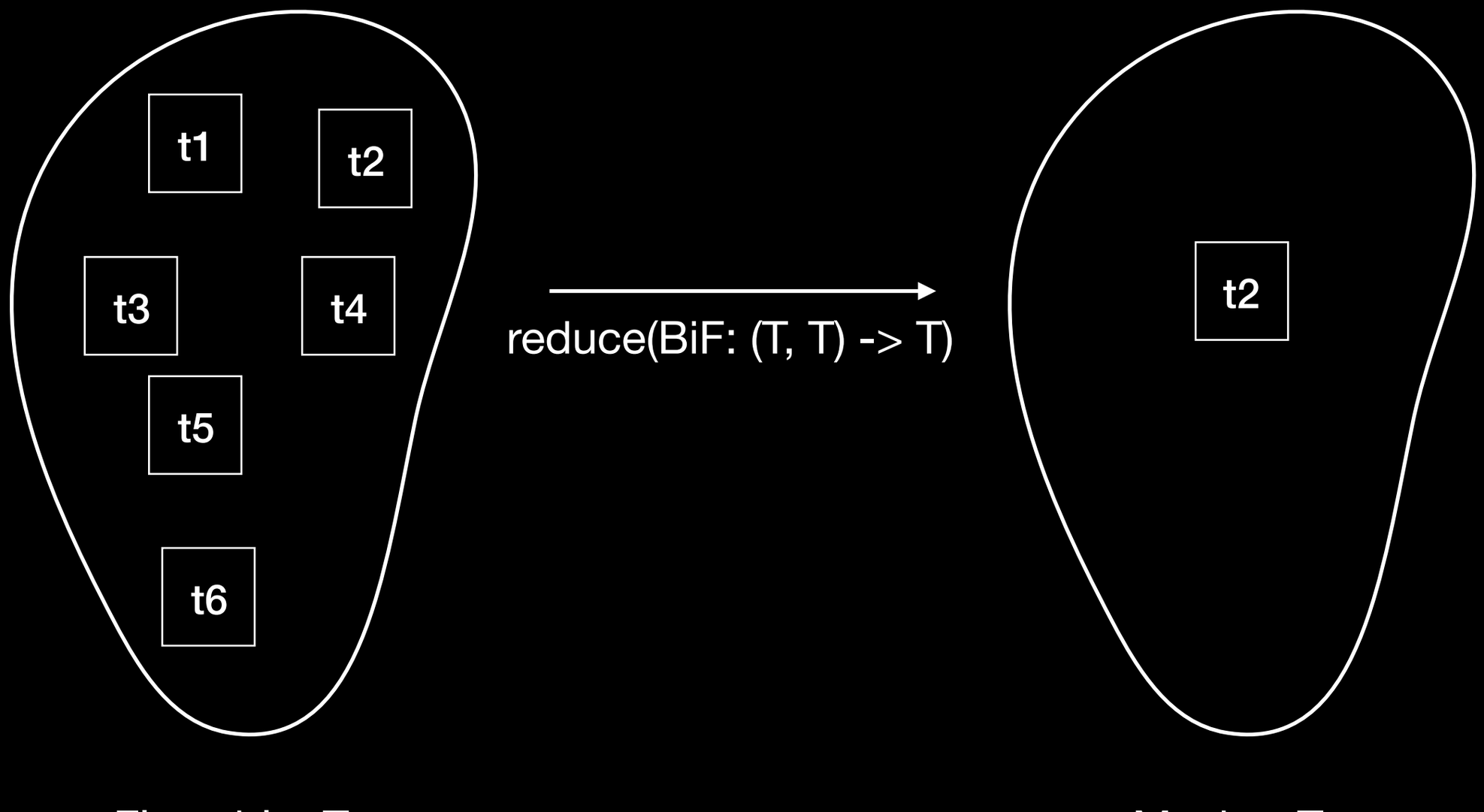


Flowable<T>

Flowable<List<T>>

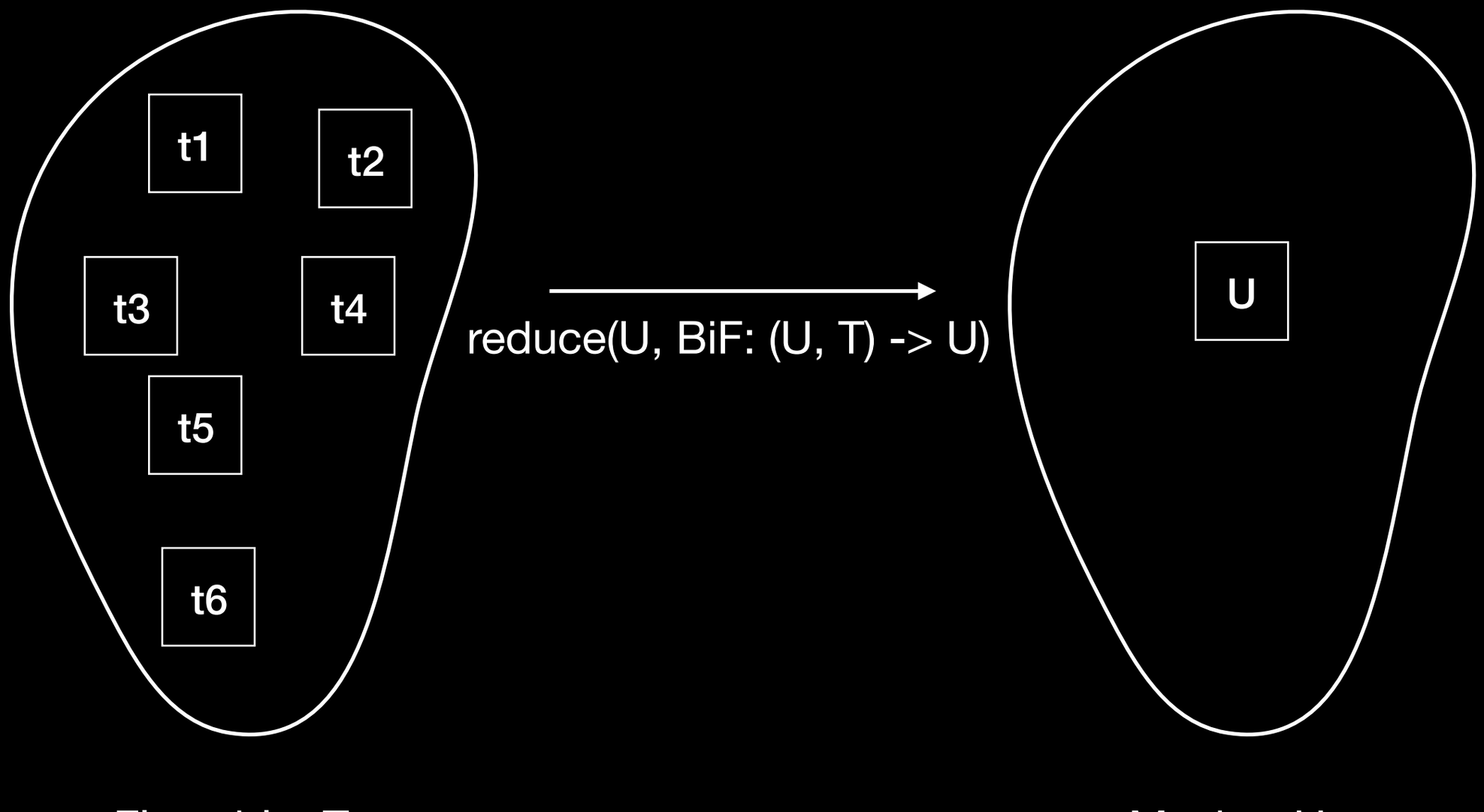
@dovaleac **Pablo González Doval** dovaleac@gmail.com

Flowable.reduce()



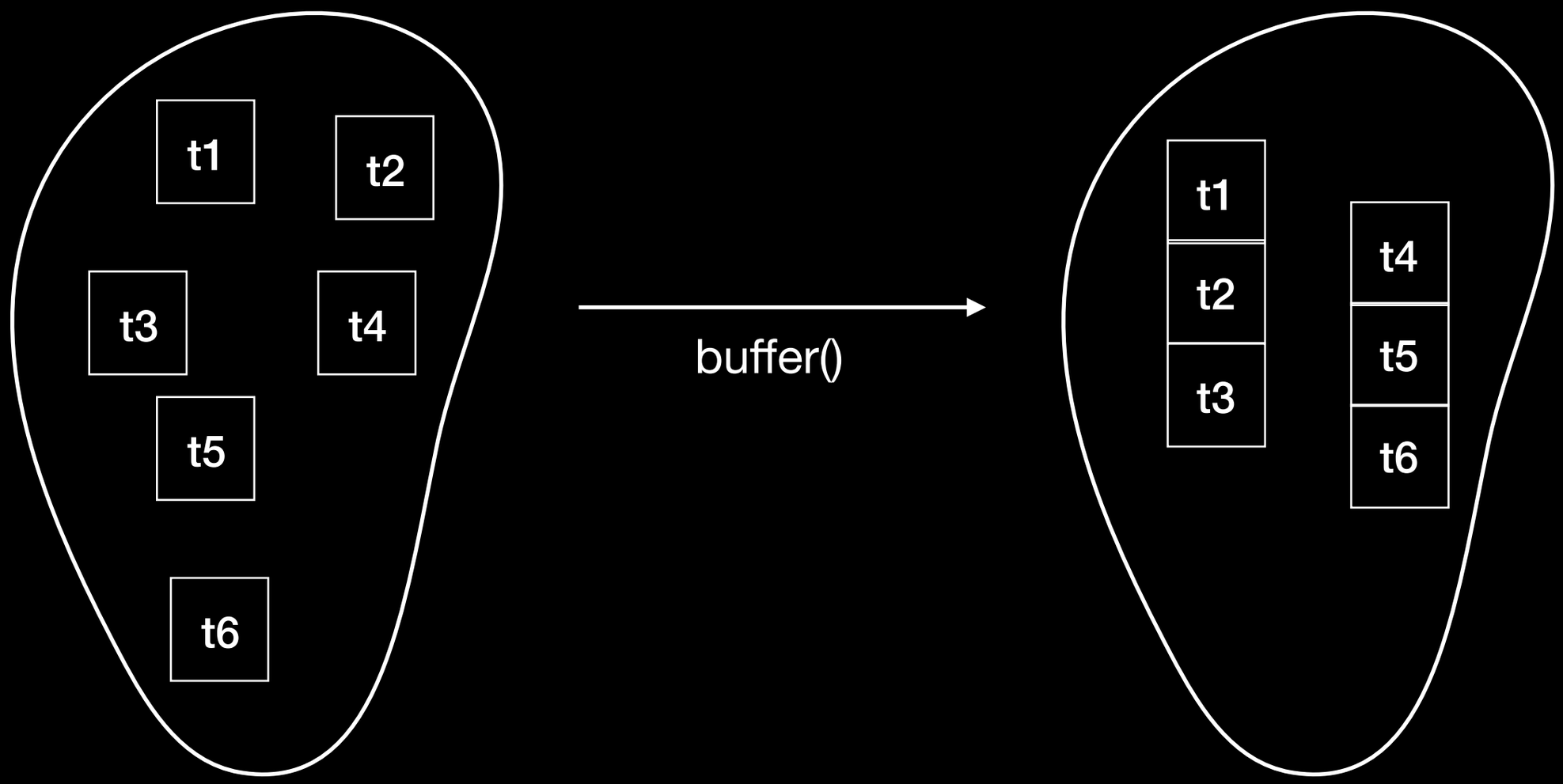
@dovaleac **Pablo González Doval** dovaleac@gmail.com

Flowable.reduce()



@dovaleac **Pablo González Doval** dovaleac@gmail.com

Flowable.buffer()

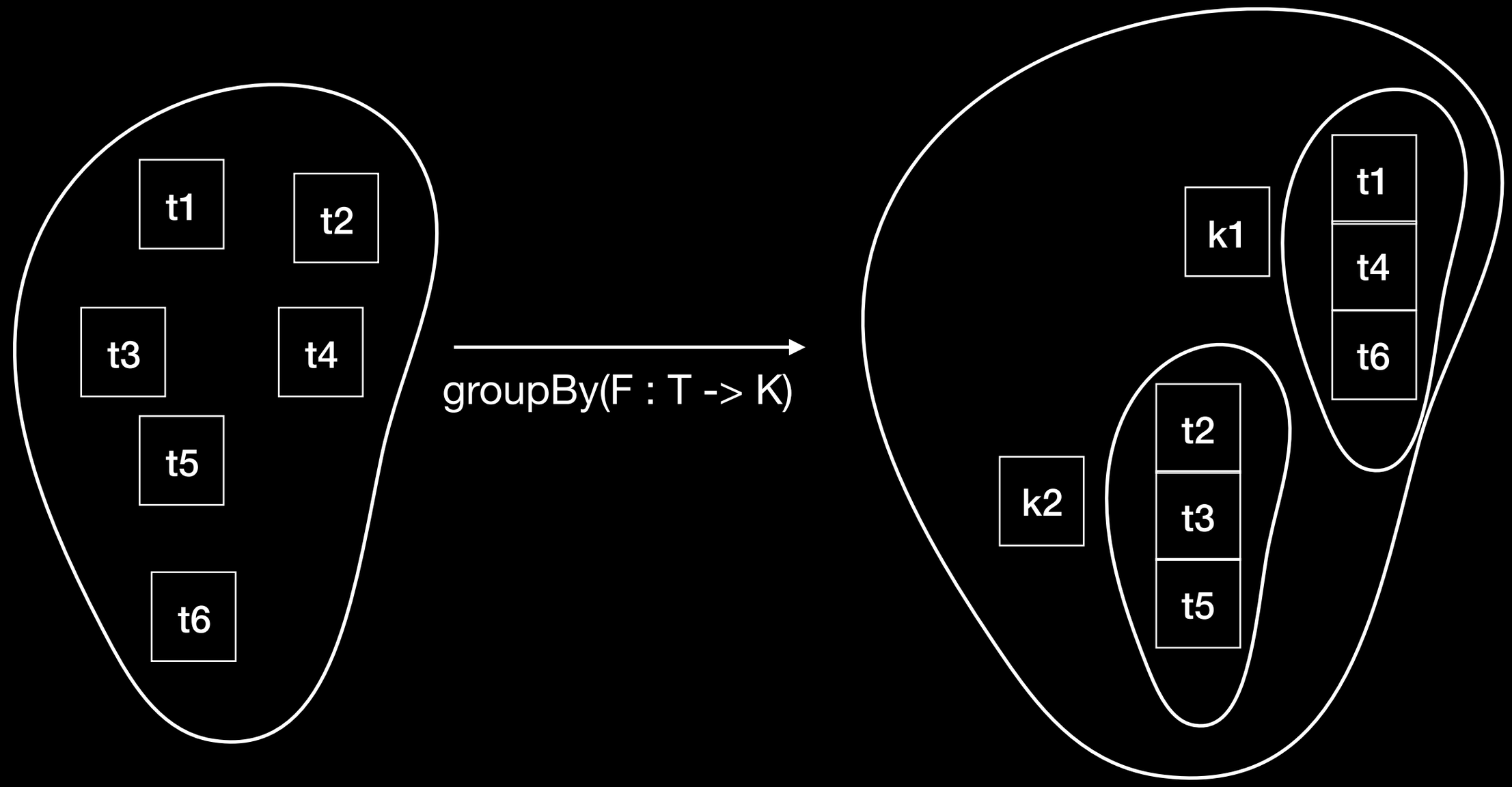


Flowable<T>

Flowable<List<T>>

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Flowable.groupBy()

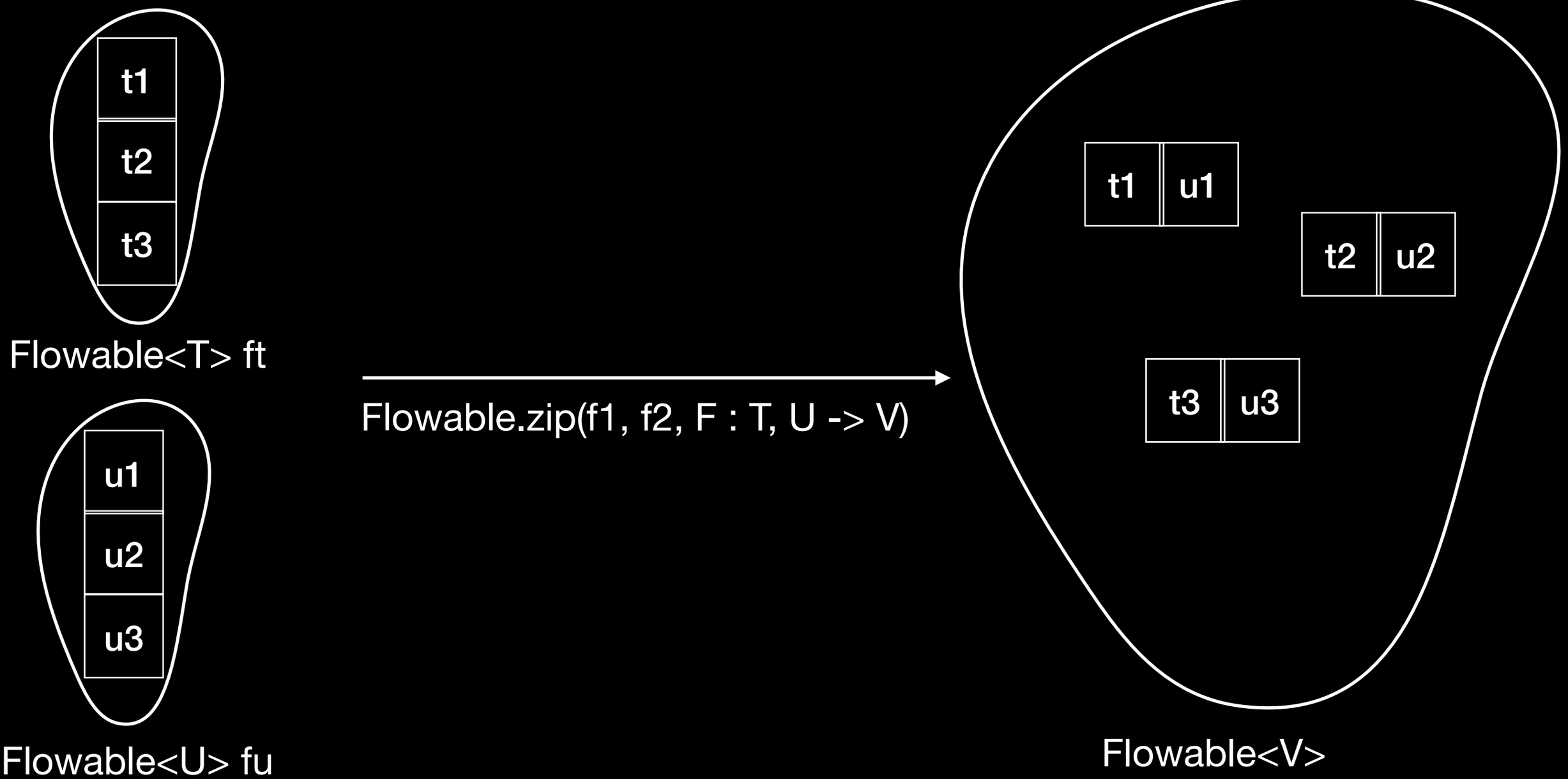


Flowable<T>

Flowable<GroupedFlowable<K, T>>

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Flowable.zip()



@dovaleac **Pablo González Doval** dovaleac@gmail.com

```
single.map(value -> {
    if (value == null) {
        return process(value);
    } else {
        throw new XException();
    }
});
```

```
single.flatMap(value -> {
    if (value == null) {
        return Single.just(process(value));
    } else {
        return Single.error(new XException());
    }
});
```

Cuidado con las excepciones

El manejo de excepciones de RxJava te sorprenderá

@dovaleac **Pablo González Doval** dovaleac@gmail.com

Y ahora... los ejercicios

- Ejercicios: **com.dovaleac.rxjava2tutorial.services.ExerciseService**
- Test: **com.dovaleac.rxjava2tutorial.services.ExerciseServiceTest**
- La variable *entryPoint* contiene el *ReadService* y el *WriteService*
- Leed el javadoc con cuidado

Muchas gracias!

@dovaleac **Pablo González Doval** dovaleac@gmail.com