RX Java / RX Kotlin

asynchronous data streams

Click events

Push notifications

Keyboard input

Reading a file

Database access

Device sensor updates

GPS Updates

GPS Updates

-36.34543, 3.23445

GPS Updates

-36.34543, 3.23445

-36.24543, 3.23425

GPS Updates

-36.34543, 3.23445

-36.24543, 3.23425

-36.34543, 3.13445

Server response

I want toys !!!

Server response

😀

Server response

I want toys !!!

Server response

Which of the following is an asynchronous data stream?

A Click events

B Database access

C Server response

D All the above

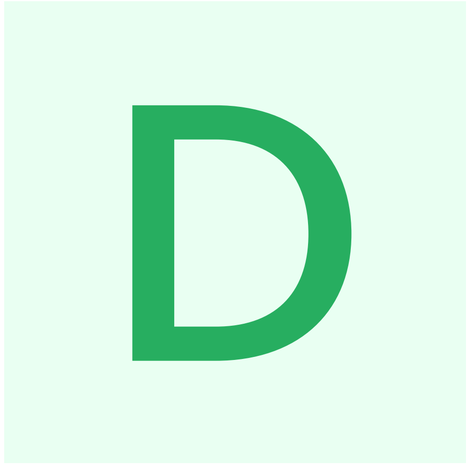
Which of the following is an asynchronous data stream?

A Click events

B Database access

C Server response

D All the above



Why?

Chaining

Threading

Composable

Abstraction

Data transformation

Non-blocking

Avoid callbacks

Data Transformation

```
observable.just (5,6,7)
        .map { ";-) ".repeat(it) }
        .subscribe { Println (it) }
```

Data Transformation

```
observable.just (5,6,7)
        .map { ";-) ".repeat(it) }
        .subscribe { Println (it) }
```

Data Transformation

```
observable.just (5,6,7)
        .map { ";-) ".repeat(it) }
        .filter { it.length < 24 }
        .subscribe { Println (it) }
```

Data Transformation

```
observable.just (5,6,7)
        .map { ";-) ".repeat(it) }
        .filter { it.length < 24 }
        .subscribe { Println (it) }
```

Abstraction

RX java is ....?

A The silver bullet

B Simply magic

C Pure voodoo

D A library

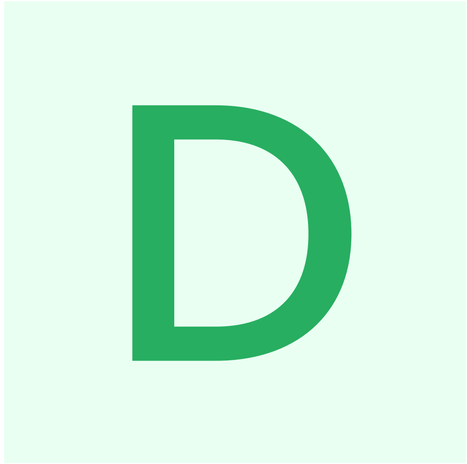
RX java is ....?

A The silver bullet

B Simply magic

C Pure voodoo

D A library

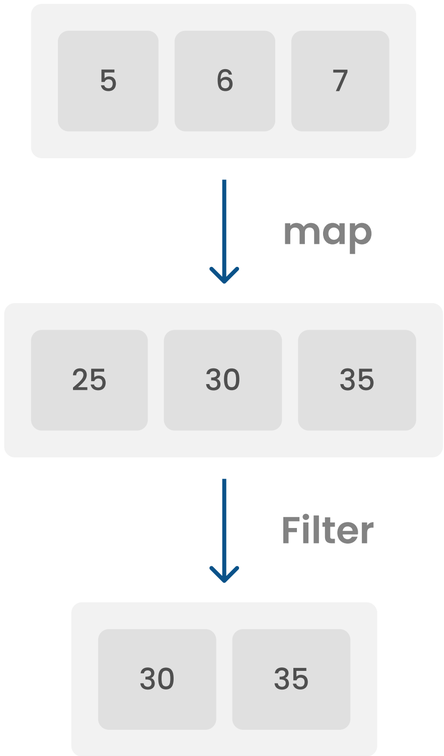


Kotlin collections

```
listof (5,6,7)
        .map { it * 5) }
        .filter { it > 25 }
```

Kotlin collections

```
listof (5,6,7)
        .map { it * 5) }
        .filter { it > 25 }
```



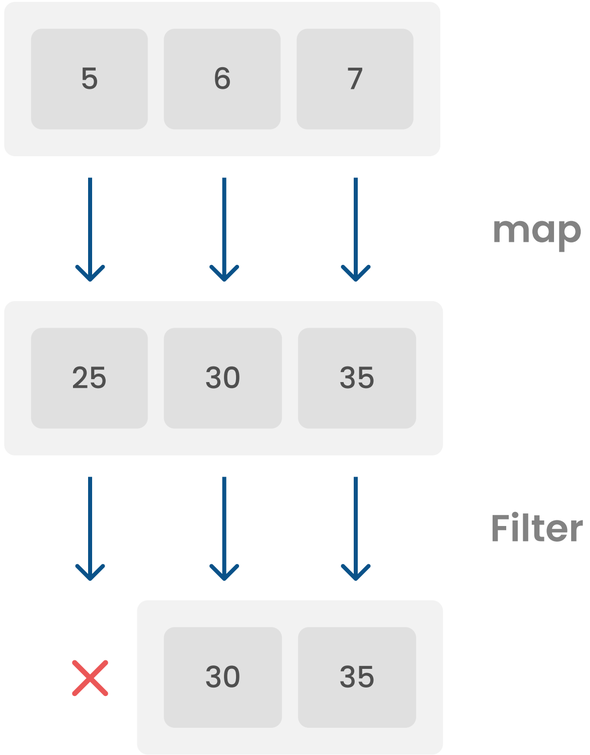
Lazy

Kotlin sequences

```
listof (5,6,7)
        .assequence()
        .map { it * 5) }
        .filter { it > 25 }
        .tolist()
```

Kotlin sequences

```
listof (5,6,7)
        .assequence()
        .map { it * 5) }
        .filter { it > 25 }
        .tolist()
```



Why?

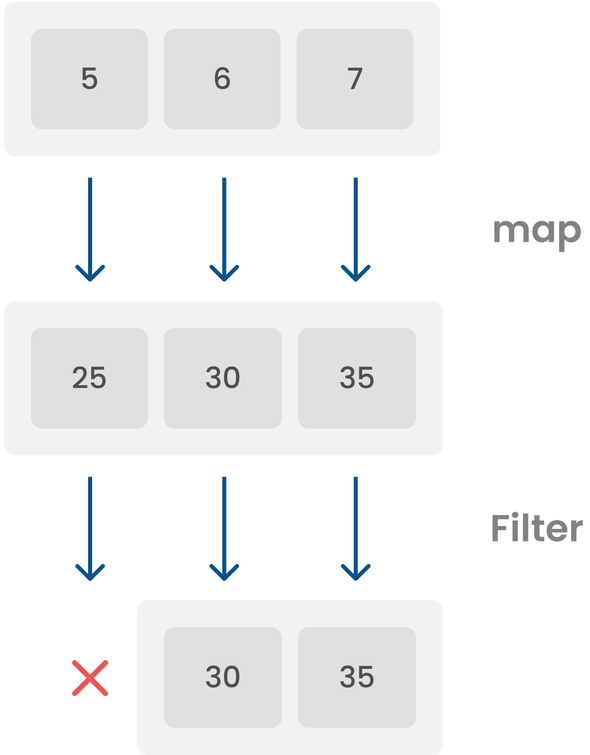
asynchronous data streams

RxJava

```
numbers
        .map { it * 5) }
        .filter { it > 25 }
        .subscribe()
```

RxJava

```
numbers
        .map { it * 5) }
        .filter { it > 25 }
        .subscribe()
```



Flexible Threading

Schedulers

```
observable.just (5,6,7)
      .map { “;-) “.repeat (it) }
      .subscribe { println(it) }
```

```
observable.just (5,6,7)
    .subscribeOn(Schedulers.io())
    .map { ";-) ".repeat (it) }
    .subscribe { println(it) }
```

The Basics

Operators

Observer

Observable

Observable

Hot observable

Hot observable

I want
hugs !!!

Hot observable

Click events

Push notifications

Keyboard input

Reading a file

Database access

Device sensor updates

Cold observable

Click events

Push notifications

Keyboard input

Reading a file

Database access

Device sensor updates

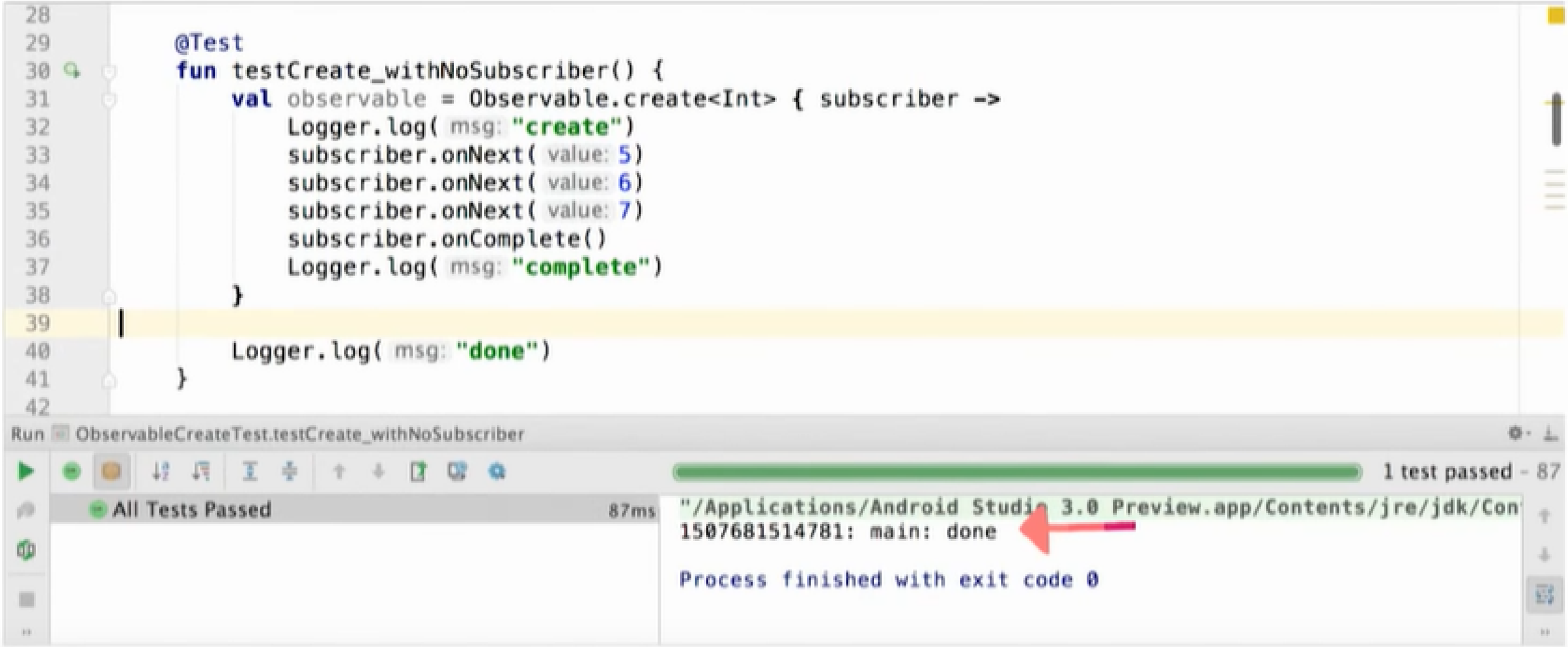
Where ?

```
Observable.create<Int> { subscriber -> }

Observable.just ( itemu, item2, item3 )

Observable.interval ( 2, TimeUnit.SECONDS)
```

```
Observable.create<Int> { subscriber ->
        Logger.log("create")
        Logger.log("complete")
}

Logger.log("done")
```

```
Observable.create<Int> { subscriber ->
        Logger.log("create")

        subscriber.onNext(5)
        subscriber.onNext(6)
        subscriber.onNext(7)

        Logger.log("complete")
}

Logger.log("done")
```

```
Observable.create<Int> { subscriber ->
        Logger.log("create")

        subscriber.onNext(5)
        subscriber.onNext(6)
        subscriber.onNext(7)
        subscriber.oncomplete( )
        Logger.log("complete)
}

Logger.log("done")
```


28
29 @Test
30 fun testCreate_withNoSubscriber() {
31 val observable = Observable.create<Int> { subscriber ->
32 Logger.log(msg: "create")
33 subscriber.onNext(value: 5)
34 subscriber.onNext(value: 6)
35 subscriber.onNext(value: 7)
36 subscriber.onComplete()
37 Logger.log(msg: "complete")
38 }
39
40 Logger.log(msg: "done")
41 }
42
Run ObservableCreateTest.testCreate_withNoSubscriber
1 test passed - 87
All Tests Passed 87ms
"/Applications/Android Studio 3.0 Preview.app/Contents/jre/jdk/Con
1507681514781: main: done
Process finished with exit code 0

Observables can...

A emit items

B be cold

C be hot

D all of the above

Observables can...

A emit items

B be cold

C be hot

D **all of the above**

observer

```
interface Observer<T> {
        fun onError(e: Throwable)

        fun onComplete( )

        fun onNext (t: T)

        fun onSubscribe(d: Disposable)

}
```

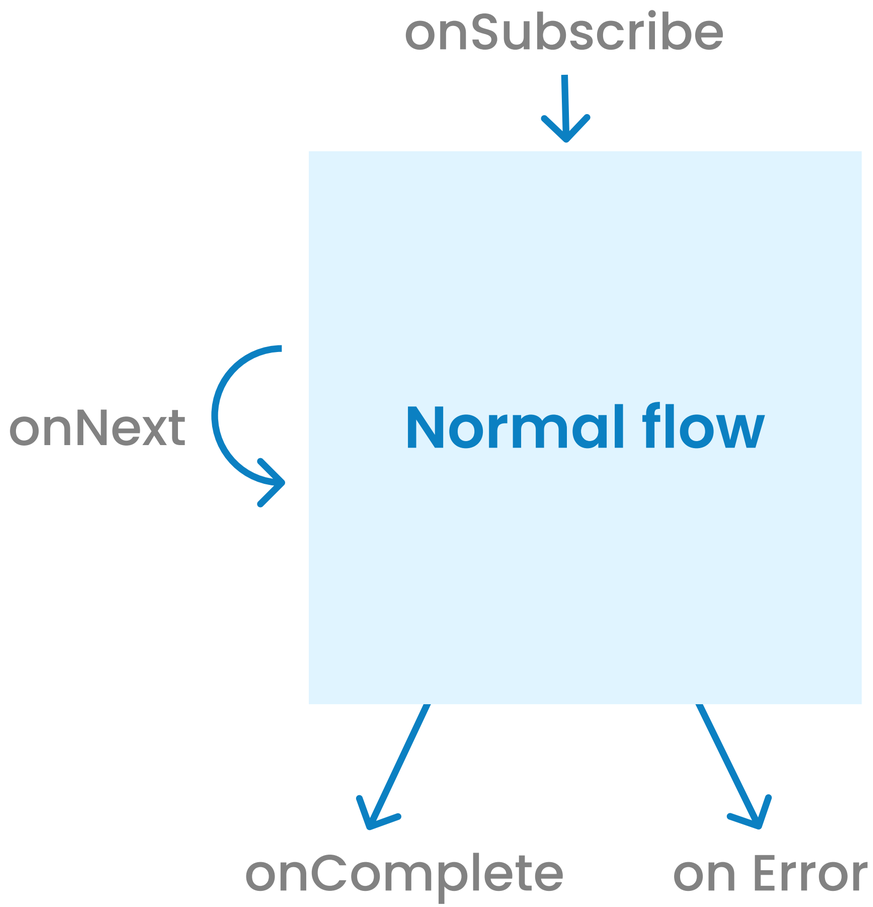
```
interface Observer<T> {
        fun onError(e: Throwable)

        fun onComplete( )
        fun onNext (t: T)
        fun onSubscribe(d: Disposable)
}
```

```
interface Observer<T> {
        fun onError(e: Throwable)

        fun onComplete( )

        fun onNext (t: T)

        fun onSubscribe(d: Disposable)

}
```

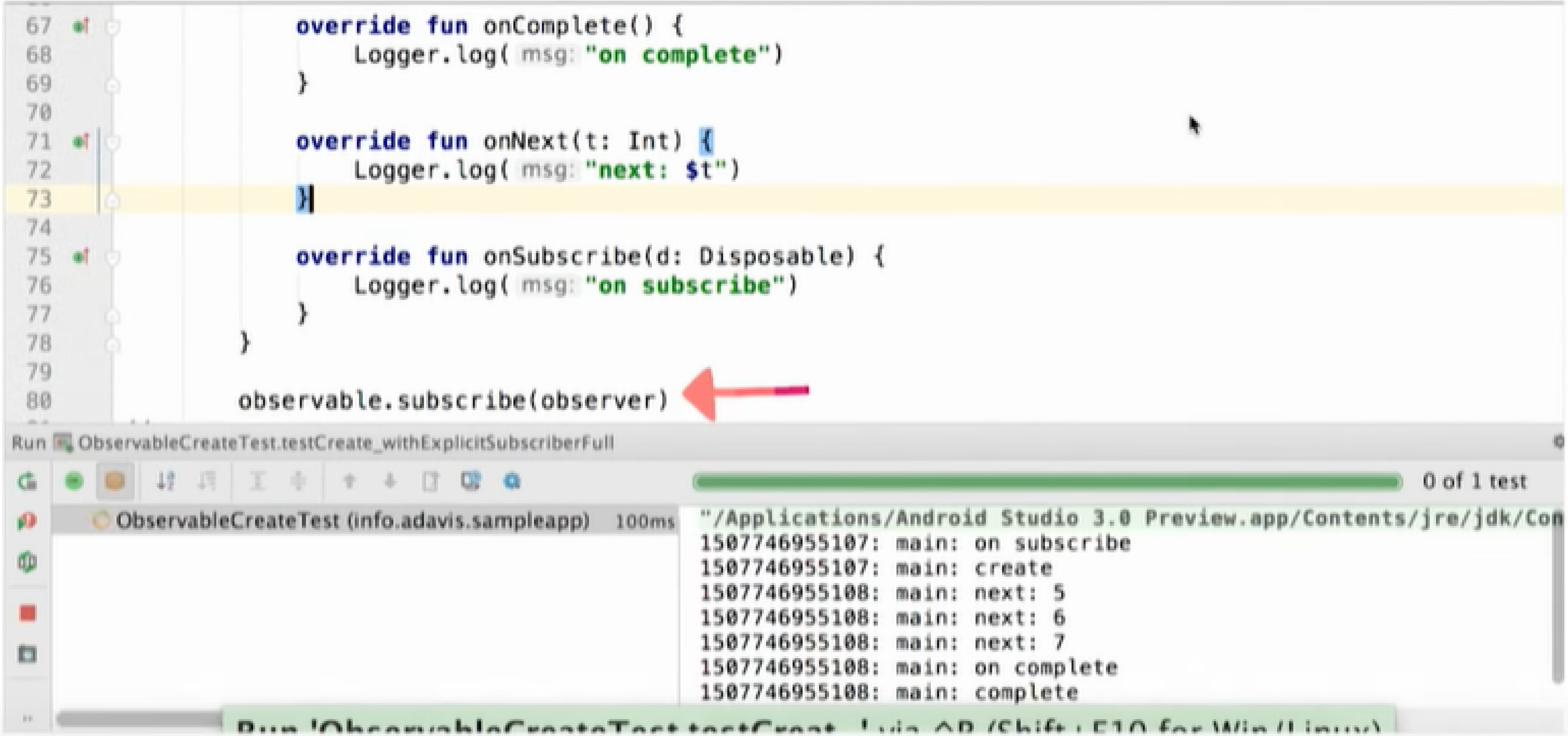
```
interface Observer<T> {
        fun onError(e: Throwable)

        fun onComplete( )
        fun onNext (t: T)
        fun onSubscribe(d: Disposable)

}
```

```
interface Observer<T> {
        fun onError(e: Throwable)

        fun onComplete( )

        fun onNext (t: T)

        fun onSubscribe(d: Disposable)

}
```

observer’s lifecycle



```
val observer = object : Observer<Int> {

        override fun onError(e: Throwable){
            Logger.log(e)
}

        override fun onComplete( ) {
            Logger.log("on complete")
}

        override fun onNect(t: Int ) {
            Logger.log("next: $t")
}

        override fun onSubscribe (d : Disposable) {
            Logger.log("on subscribe")
}
```


override fun onComplete() {
Logger.log(msg: "on complete")
}
override fun onNext(t: Int) {
Logger.log(msg: "next: $t")
}
override fun onSubscribe(d: Disposable) {
Logger.log(msg: "on subscribe")
}
}
observable.subscribe(observer)
Run ObservableCreateTest.testCreate_withExplicitSubscriberFull
0 of 1 test
ObservableCreateTest (info.adavis.sampleapp) 100ms
"/Applications/Android Studio 3.0 Preview.app/Contents/jre/jdk/Con
1507746955107: main: on subscribe
1507746955107: main: create
1507746955108: main: next: 5
1507746955108: main: next: 6
1507746955108: main: next: 7
1507746955108: main: on complete
1507746955108: main: complete

```
interface Consumer<T> {

    fun accept ( t:T )

}
```

```
val consumer = object : consumer<Int>  {

        override fun accept ( t: Int ) {
            Logger.log(“next: $t”)
        }
}
```

```
            subscriber.onNext( value: 6)
            subscriber.onNext( value: 7)
            subscriber.onComplete()
            Logger.log( msg: "complete")
        }

        val consumer = object : Consumer<Int> {
            override fun accept(t: Int) {
                Logger.log( msg: "next: $t")
            }
        }

        observable.subscribe(consumer)
    }
```

servableCreateTest.testCreate_withExplicitSubscriberFull

1 test passed

All Tests Passed 148ms

```
1507747777513: main: create
1507747777513: main: next: 5
1507747777513: main: next: 6
1507747777513: main: next: 7
1507747777513: main: complete

Process finished with exit code 0
```

Consumer

```
val observer = Consumer<Int> { t -> logger.log("next: $t") }

obs.subscribe(consumer)
```

Consumer

```
obs.subscribe(Consumer<Int> { t -> logger.log("next: $t") }
```

Consumer

```
obs.subscribe( { t -> logger.log(“next: $t”) }
```

Consumer

```
obs.subscribe{ t -> logger.log(“next: $t”) }
```

Consumer

```
obs.subscribe{ logger.log("next: $it") }
```

```kotlin
@Test
fun testCreate_withExplicitSubscriberFull() {
    val observable = Observable.create<Int> { subscriber ->
        Logger.log( msg: "create")
        subscriber.onNext( value: 5)
        subscriber.onNext( value: 6)
        subscriber.onNext( value: 7)
        subscriber.onComplete()
        Logger.log( msg: "complete")
    }

    observable.subscribe{ Logger.log( msg: "next: $it") }
}
```

ervableCreateTest.testCreate_withExplicitSubscriberFull

1 test passed

All Tests Passed 250ms

```
1507749694482: main: create
1507749694482: main: next: 5
1507749694482: main: next: 6
1507749694482: main: next: 7
1507749694482: main: complete

Process finished with exit code 0
```

Observers...

A have a lifecycle

B always complete

C Never error

D all of the above

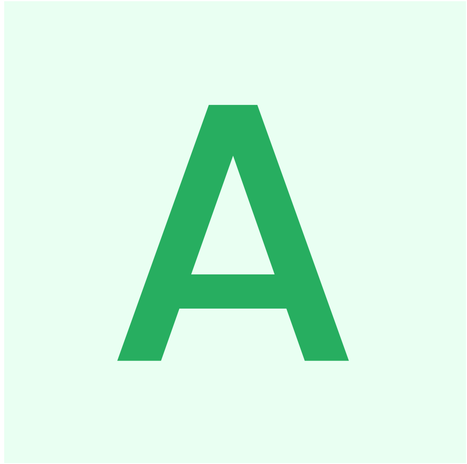
Observers...

A emit items

B be cold

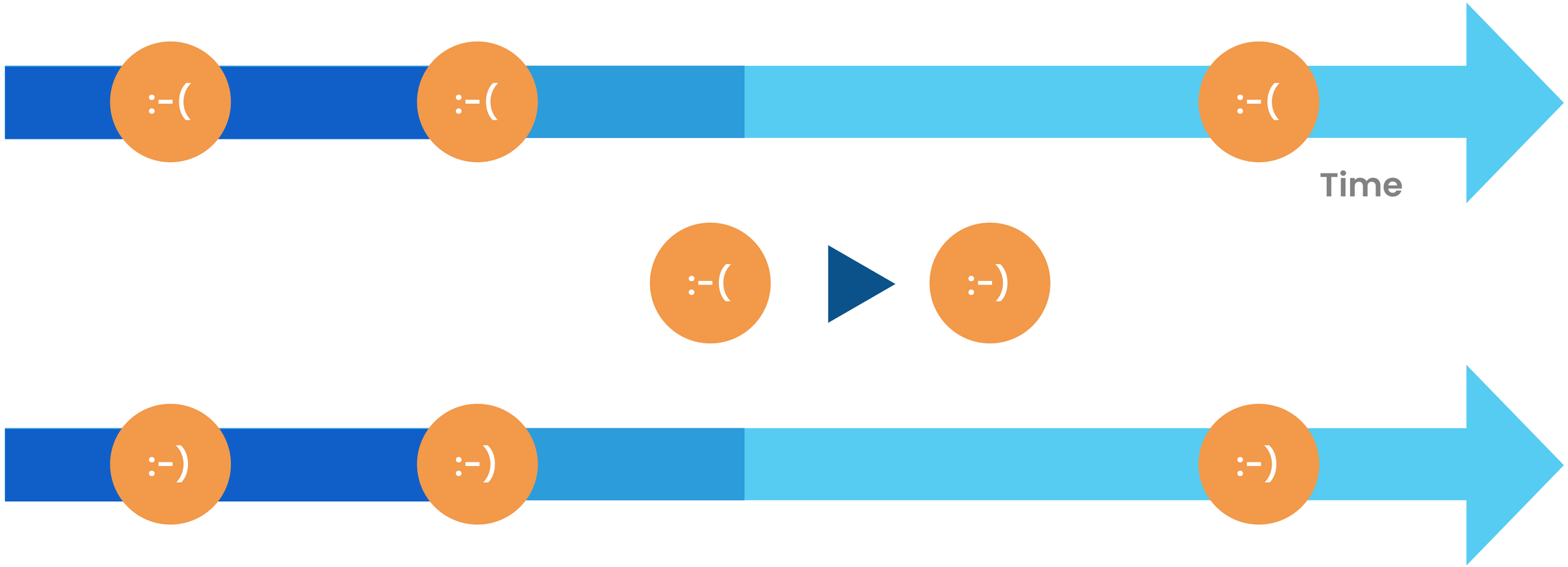
C be hot

D all of the above



Operator

Operator : map()



Operator : map()

```
Observable.just(5,6,7,)

        .map { ";-)".repeat(it) }
        .subscribe { println(it) }
```

Operator : map()

```
Observable.just(5,6,7,)
        .map {object: function<Int, String> {
            override fun apply(t: Int) : String {
                return ";-) ".repeat(t)
            }
        })
        .subscribe { println(it) }
```

Operator : map()

```
Observable.just(5,6,7,)
        .map {object: function<Int, String> {
            override fun apply(t: Int) : String {
                return ";-) ".repeat(t)
            }
        })
        .subscribe { println(it) }
```

Operator : map()

```
Observable.just(5,6,7,)
        .map {object: function<Int, String> {
            override fun apply(t: Int) : String {
                return ";-) ".repeat(t)
            }
        })
        .subscribe { println(it) }
```

Operator : map()

```
Observable.just(5,6,7,)
        .map {object: function<Int, String> {
            override fun apply(t: Int) : String {
                return ";-) ".repeat(t)
            }
        })
        .subscribe { println(it) }
```

Operator : map()

```
.map {object: function<Int, String> {
        override fun apply(t: Int) : String {
            return ";-) ".repeat(t)
        }
})
```

```
.map { ";-) ".repeat(it)}
```

What's the output?

```
Observable.just(1,2,3)
        .map {it * 2}
        .subscribe {println(it)}
```

A 1,2,3

B a,b,c

C 2,4,6

D 6,2,4

What's the output?

```
Observable.just(1,2,3)
        .map {it * 2}
        .subscribe {println(it)}
```

A 1,2,3

B a,b,c

C 2,4,6

D 6,2,4

C

What's the output?

```
Observable.just(1,2,3)
        .map {it * 2}
        .filter {it < 6}
        .subscribe { println (it)}
```

A 2,3,1

B 2,4

C 1,2,3

D 6,4

What's the output?

```
Observable.just(1,2,3)
        .map {it * 2}
        .filter {it < 6}
        .subscribe { println (it)}
```

A 2,3,1

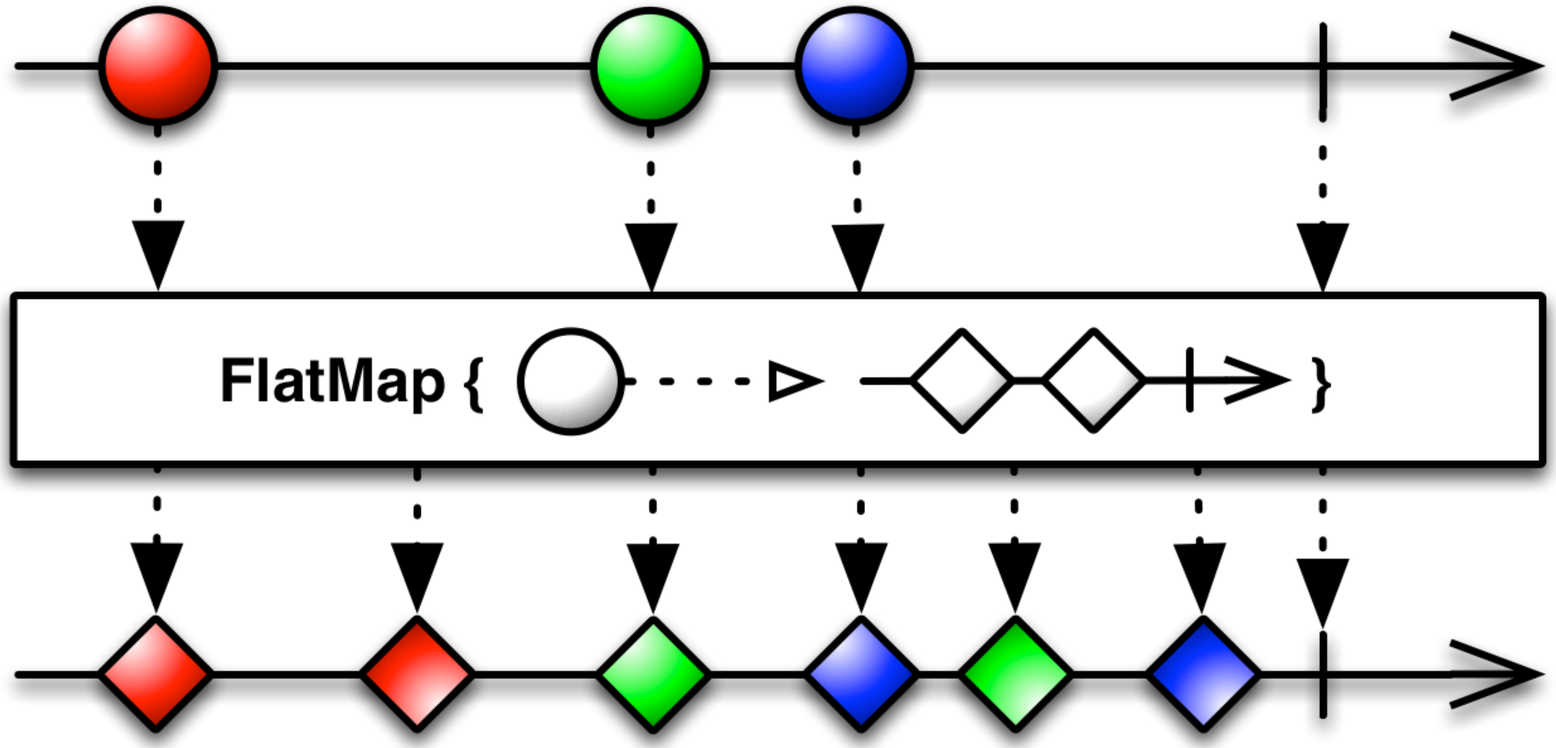
B 2,4

C 1,2,3

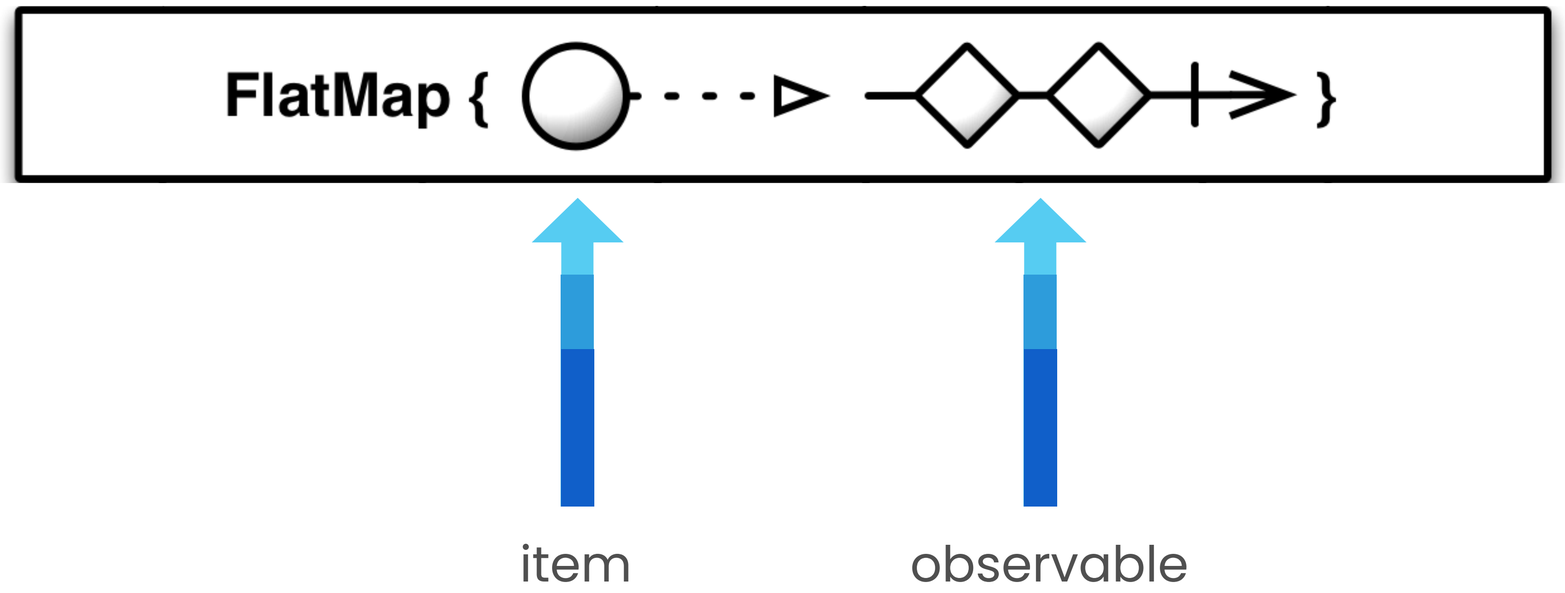
D 6,4

B

Operator: flat map()

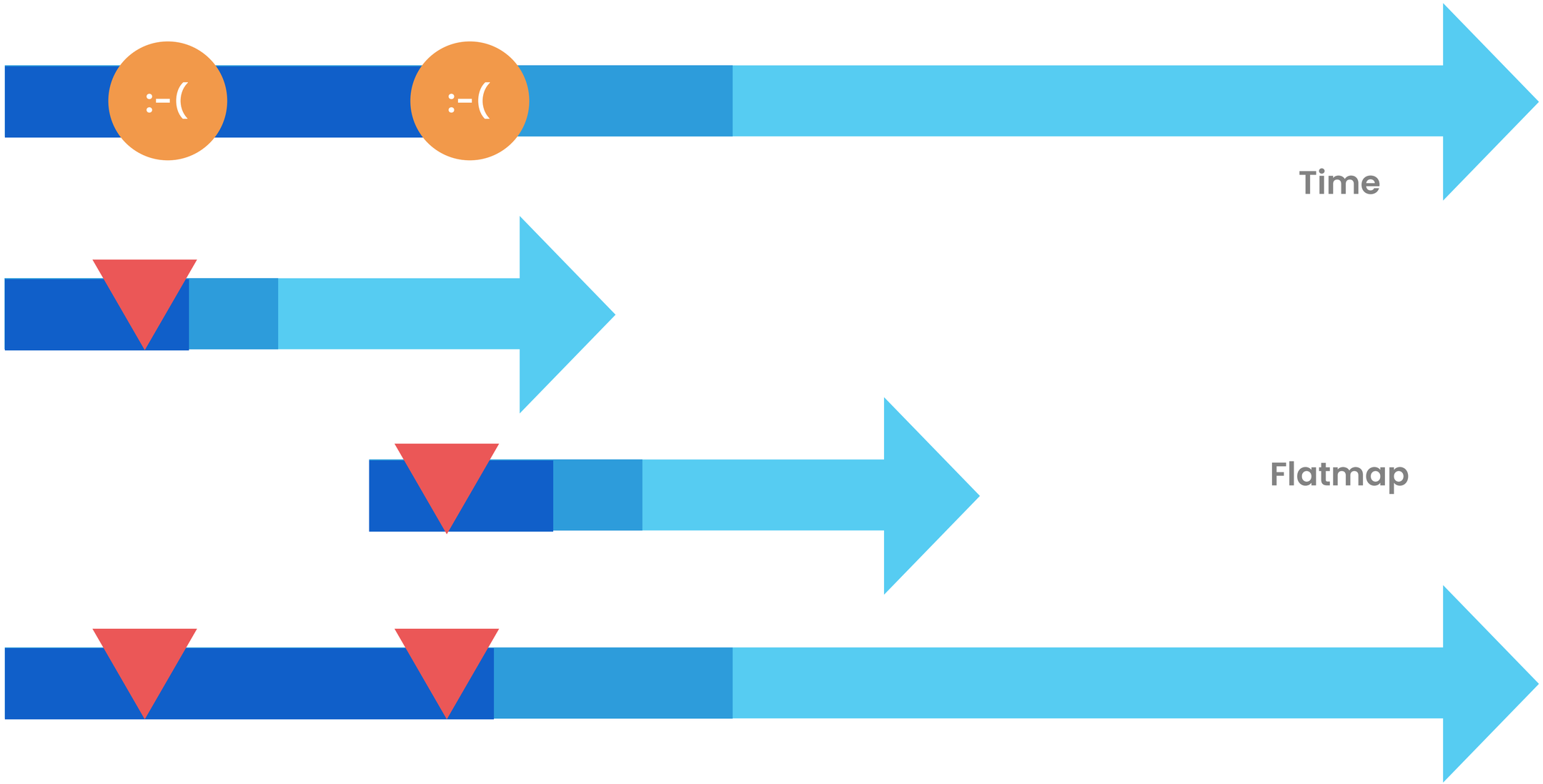


Operator: flat map()



Operator: flat map()

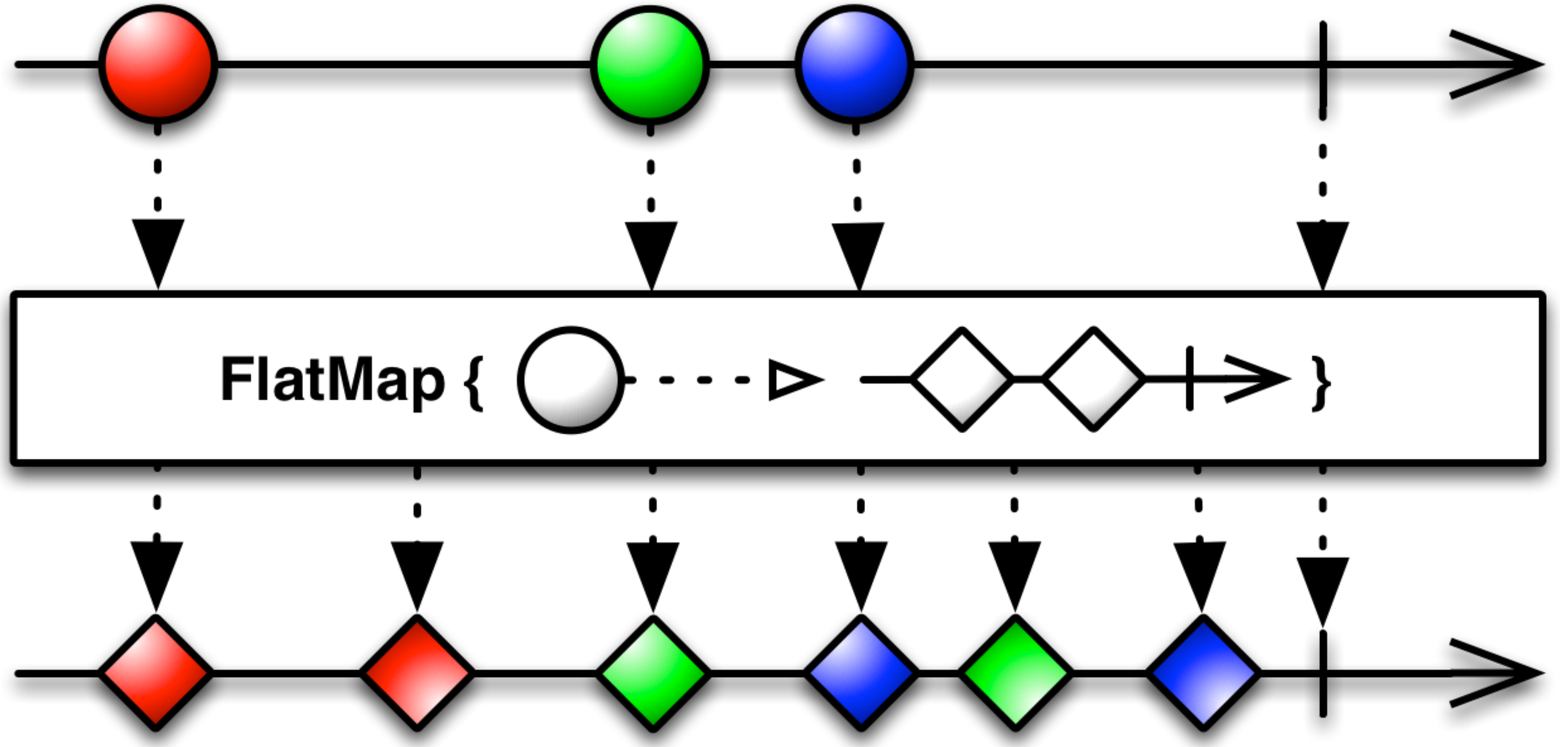
```
Observable.just( :-) , :-( )
        .flatMap( { Observable.just(▼) } )
            .subscribe { println(it) }
```


:-(
:-(
Time
Flatmap

Long running asynchronous

Operator : flatmap()

```
val user // observable<User>
val user: observable<post>

posts = user.flatmap { getusersposts(it.id) }
posts.subscribe {println(it)}
```


FlatMap {
}

Flowable

Maybe

Backpressure

Completable

Disposable

Single

should you use rxjava?

A Like functional programing?

B process items asynchronously?

C compose data?

D handle errors gracefully?

should you use rxjava?

- Like functional programing?
- process items asynchronously?
- compose data?
- handle errors gracefully?

Yes

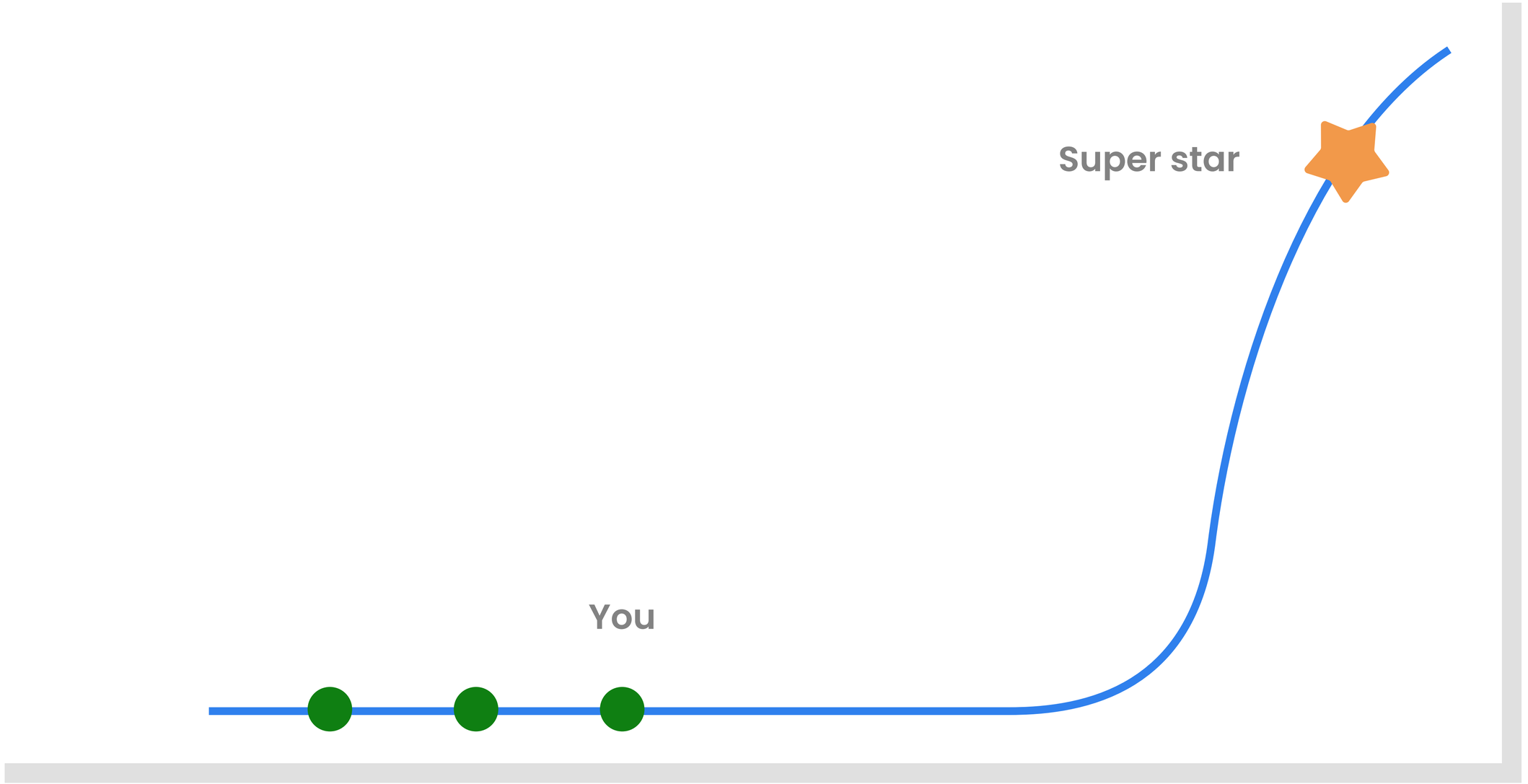
It depends

The Basics

Operator

Observer

Observable


Super star
You