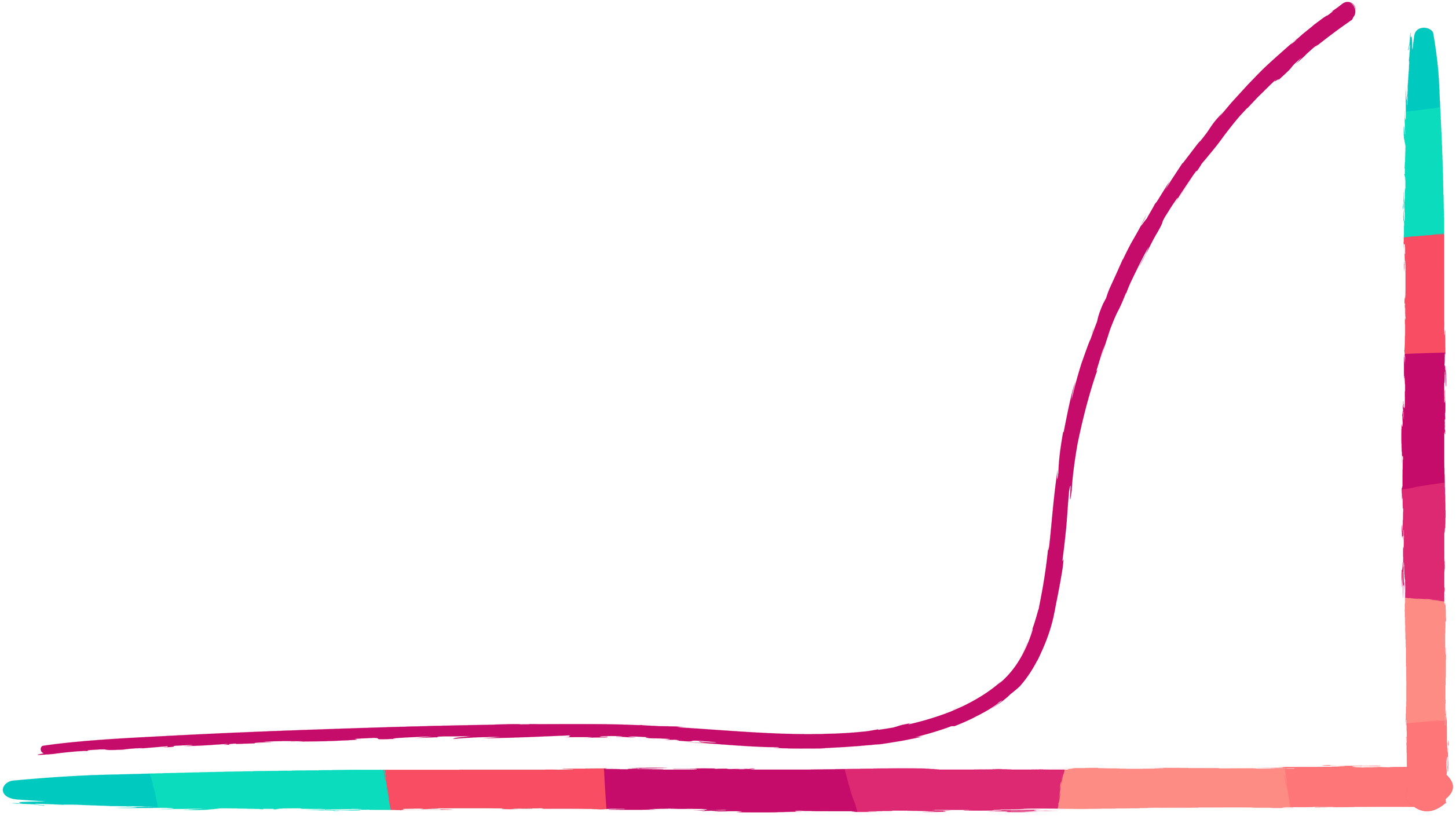
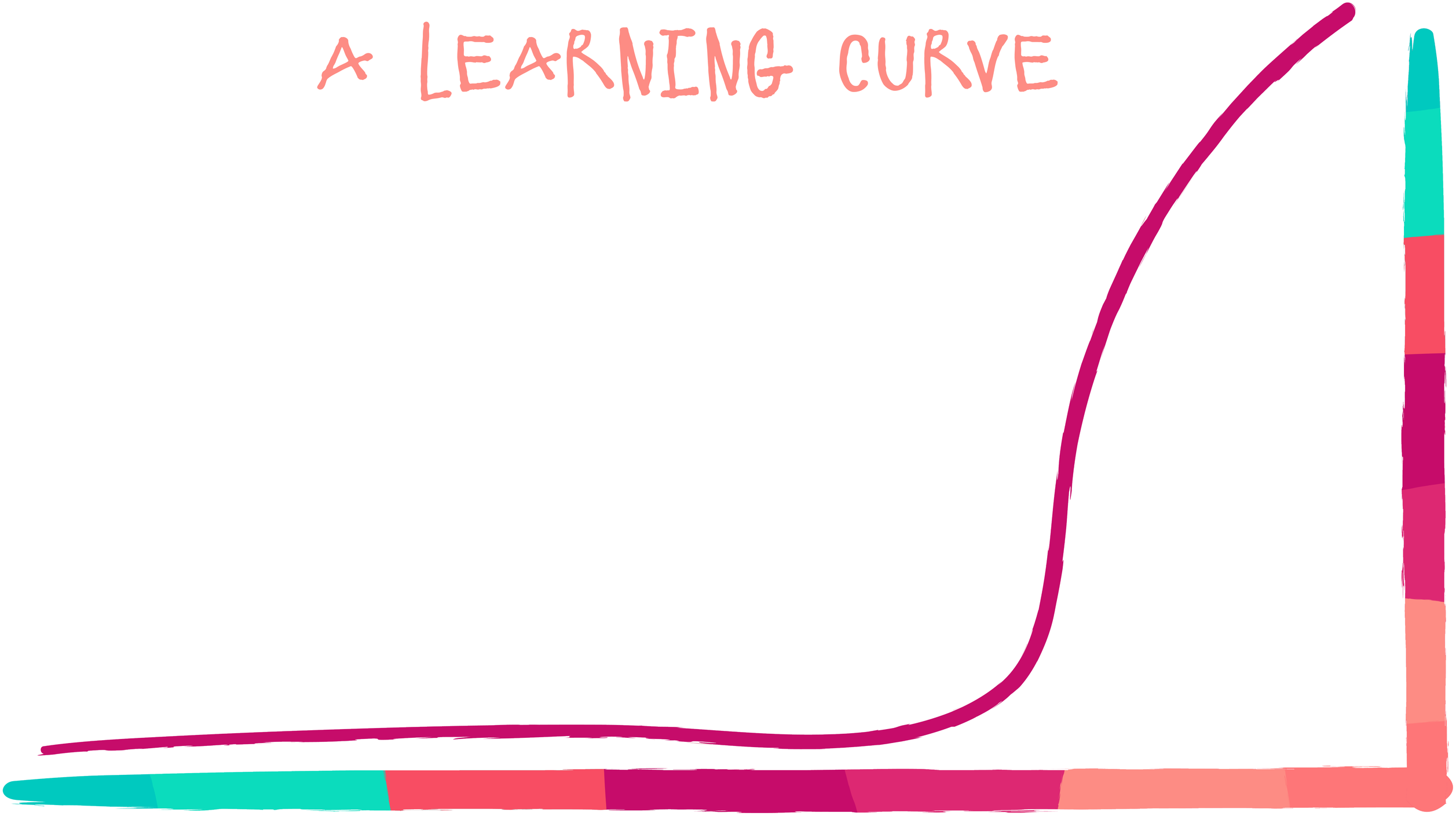
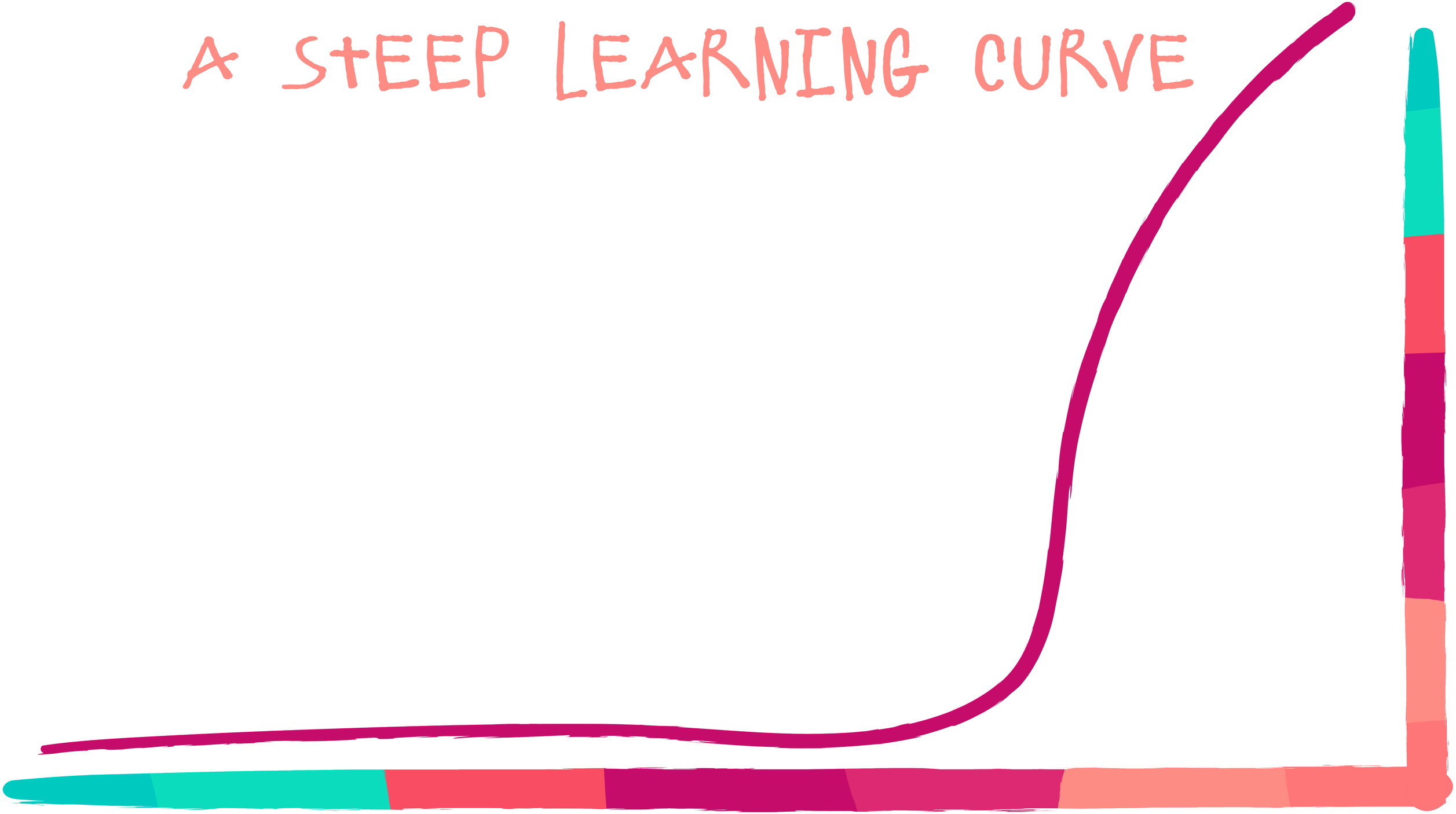


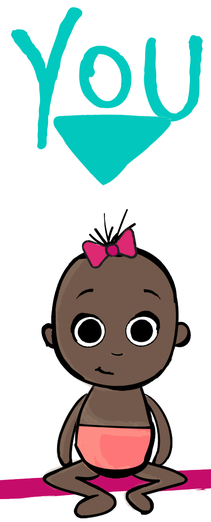
A LEARNING CURVE



A STEEP LEARNING CURVE



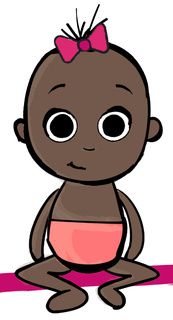
A STEEP LEARNING CURVE



A STEEP LEARNING CURVE

SUPERSTAR

YOU



A STEEP LEARNING CURVE

SUPERSTAR

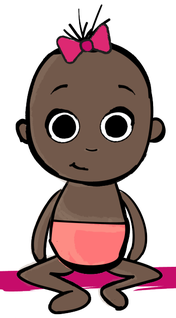
YOU

A STEEP LEARNING CURVE

SUPERSTAR

ME

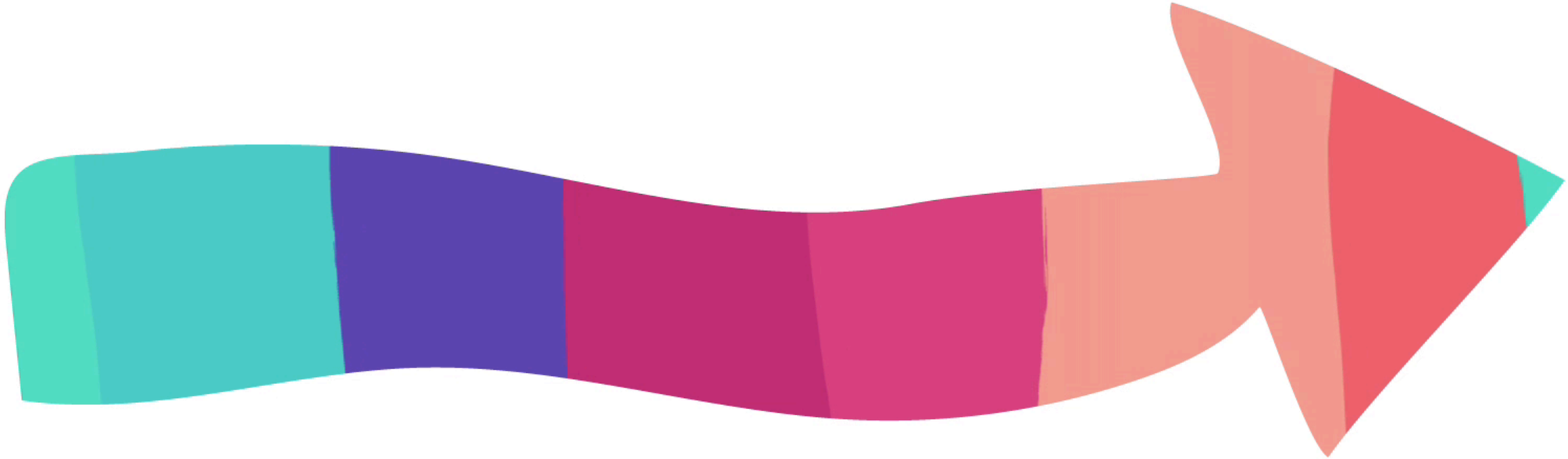
YOU



RXJAVA IN BABY STEPS

@BRWNGRLDEV

ASYNCHRONOUS DATA STREAMS



CLICK EVENTS

PUSH NOTIFICATIONS

KEYBOARD INPUT

READING A FILE

DATABASE ACCESS

DEVICE SENSOR UPDATES


GPS UPDATES
TIME

GPS UPDATES

TIME

-36.34543, 3.23445

GPS UPDATES

TIME

-36.34543, 3.23445

-36.24543, 3.23425

GPS UPDATES

TIME

-36.34543, 3.23445

-36.24543, 3.23425

-35.34543, 3.13445


SERVER RESPONSE
TIME

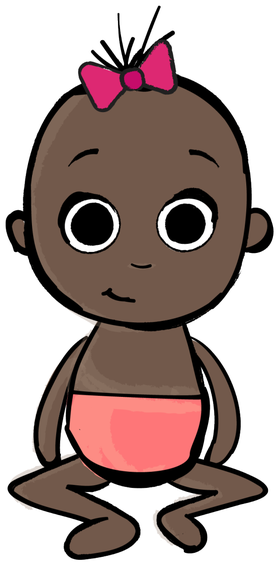
SERVER RESPONSE

TIME

SERVER RESPONSE

toys

TIME

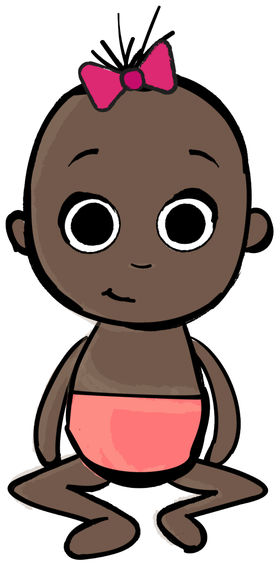


SERVER RESPONSE

TIME

SERVER RESPONSE

TIME

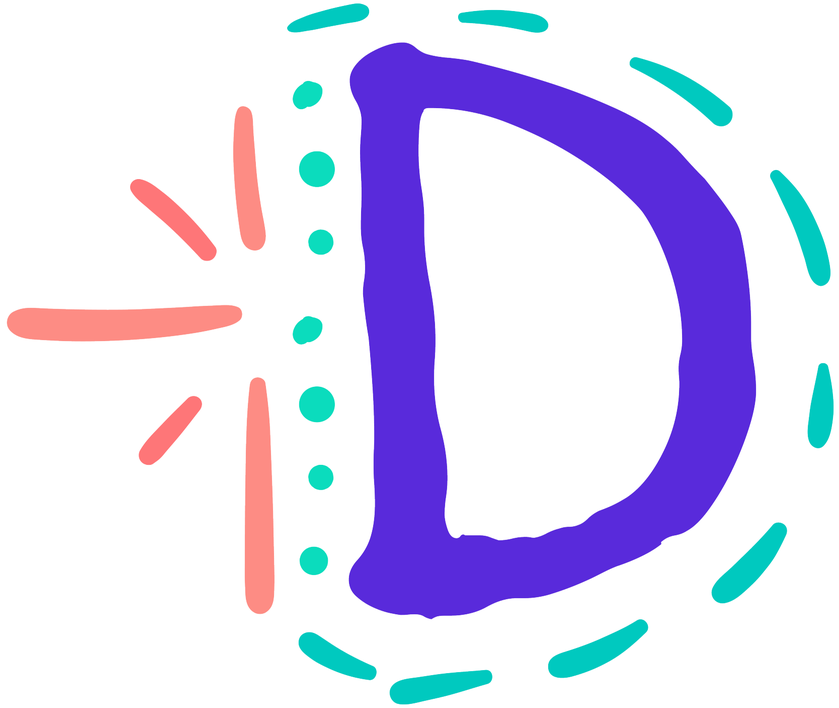


WHICH OF THE FOLLOWING IS AN ASYNCHRONOUS DATA STREAM?

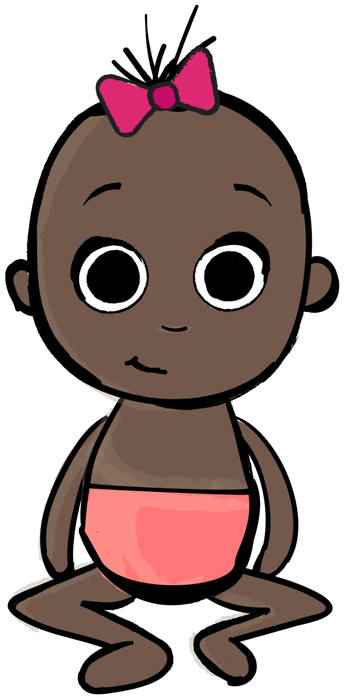
- A: CLICK EVENTS
- B: DATABASE ACCESS
- C: SERVER RESPONSE
- D: ALL OF THE ABOVE

WHICH OF THE FOLLOWING IS AN ASYNCHRONOUS DATA STREAM?

- A: CLICK EVENTS
- B: DATABASE ACCESS
- C: SERVER RESPONSE
- D: ALL OF THE ABOVE




WHY?



SCIENTIFIC RESEARCH

SCIENTIFIC RESEARCH


Following
Lucia
@LuciaP_86
Replying to @brwngrldev
Readability, higher level of abstraction, async
Ben Oberkfell
@benlikestocode
Replying to @brwngrldev
It saved my soul from c
Robert Anizoba
@bobtinkernaut
Replying to @brwngrldev @moyheen
Follow
Make multiple api calls in parallel (thank you
'zip'), threading made easy, operators to
transform/filter data
Eric Cochran
@Eric_Cochran
Replying to @brwngrldev @yogurtearl
FP: The predictability of functional
programming is what we prog
already strive for.
Perfect
@Perfect
Replying to @brwngrldev
Functional style, Thread management,
testability...
Matt Clarke
@kiwiandroiddev
Replying to @brwngrldev
Following
I've found it makes it easier to
follow/visualise a sequence of async
operations making it easier to adapt it or
spot errors
Yuliya Kaleda
@YuliyaKaleda
Following
Replying to @brwngrldev
Convenient data processing though various
chain operators

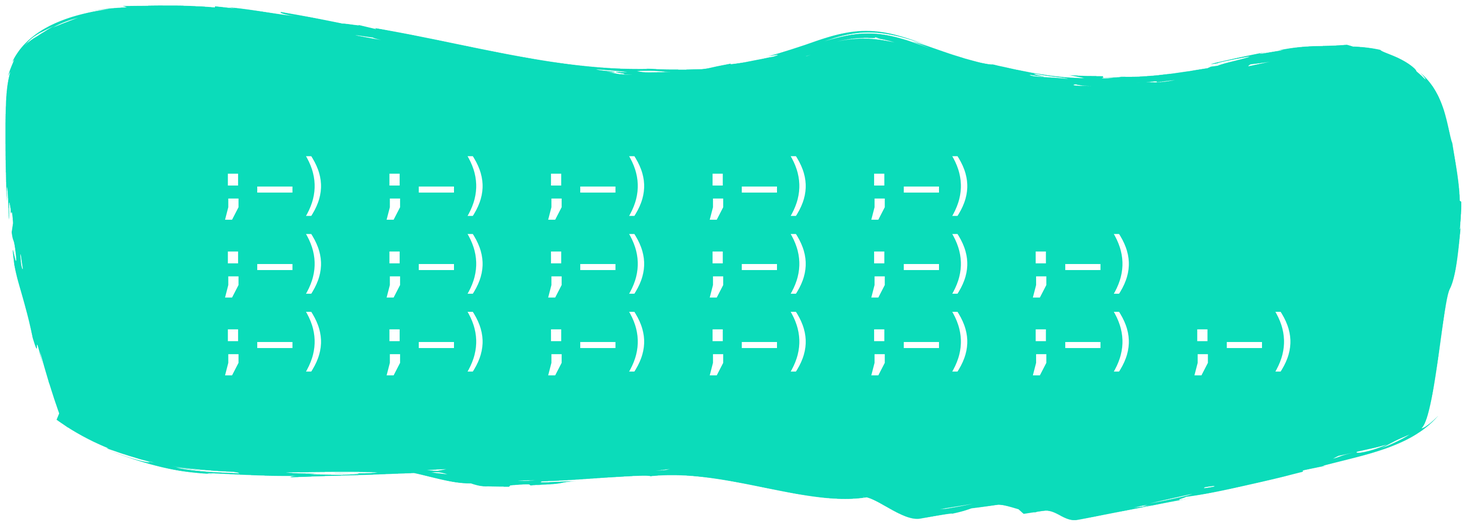

CHAINING
THREADING
COMPOSABLE
ABSTRACTION
NON-
BLOCKING
AVOID
CALLBACKS
DATA
TRANSFORMATION

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) }
```

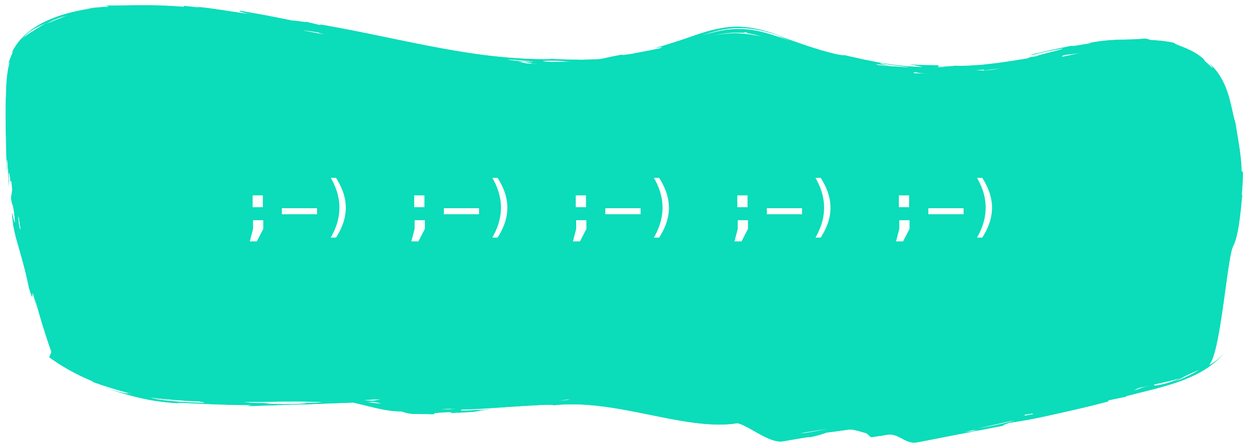


CHAINING

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

CHAINING

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



ABSTRACTION

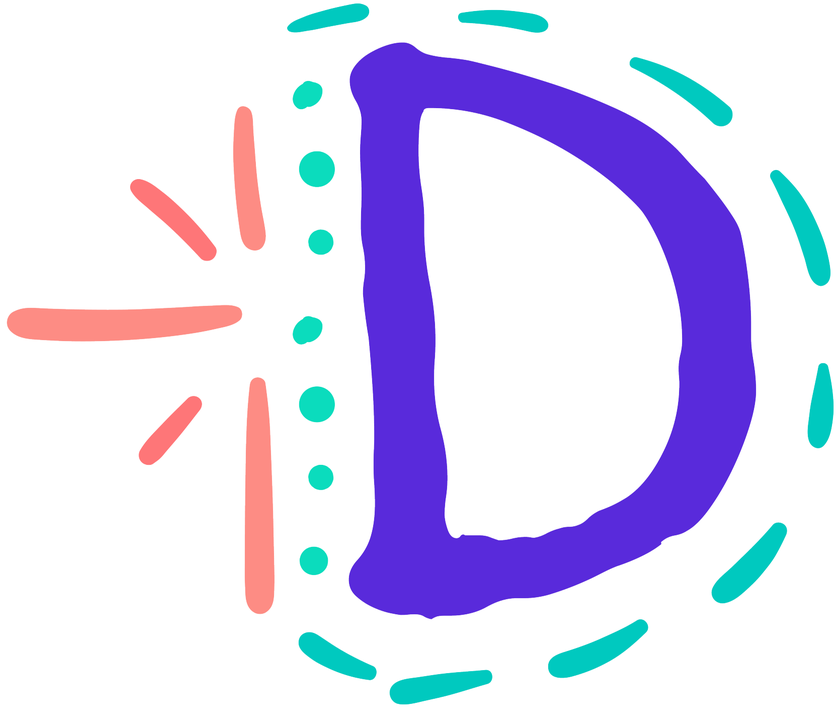
```kotlin
@Test
fun transformWithMap() {
    Observable.just( item1: 5,  item2: 6,  item3: 7)
            .map { ";-) ".repeat(it) }
            .filter { it.length < 24 }
            .subscribe { println(it) }
}
```

RXJAVA IS...?

- A: THE SILVER BULLET
- B: SIMPLY MAGIC
- C: PURE VOODOO
- D: A LIBRARY

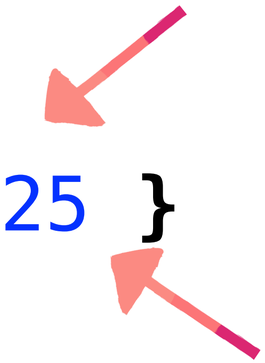
RXJAVA IS...?

- A: THE SILVER BULLET
- B: SIMPLY MAGIC
- C: PURE VOODOO
- D: A LIBRARY

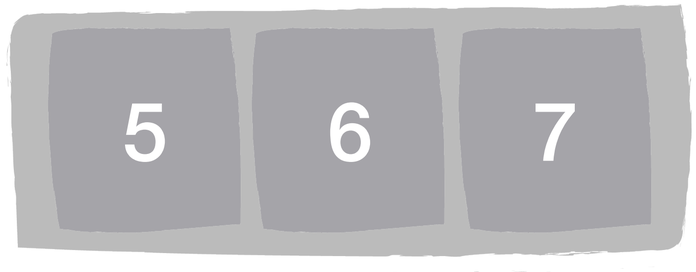


KotLIN COLLECTIONS

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



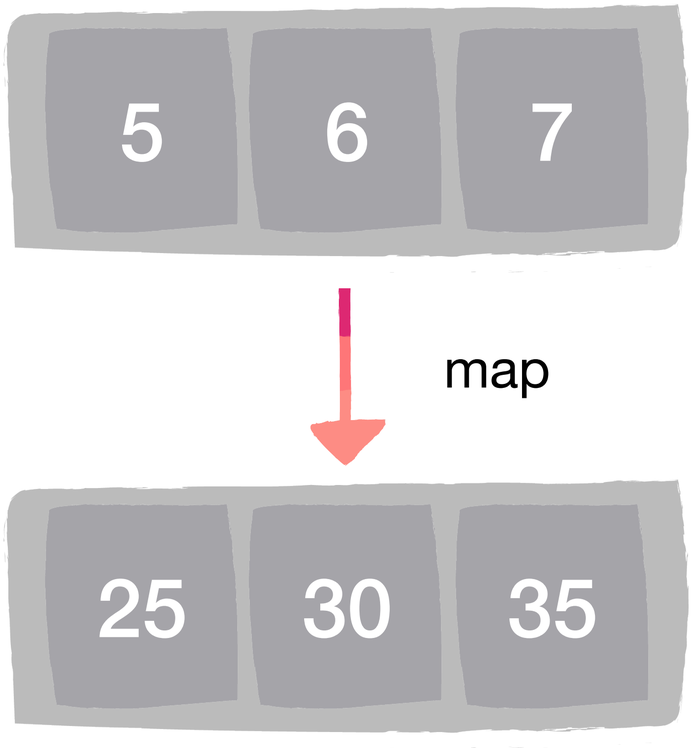
Kotlin collections




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

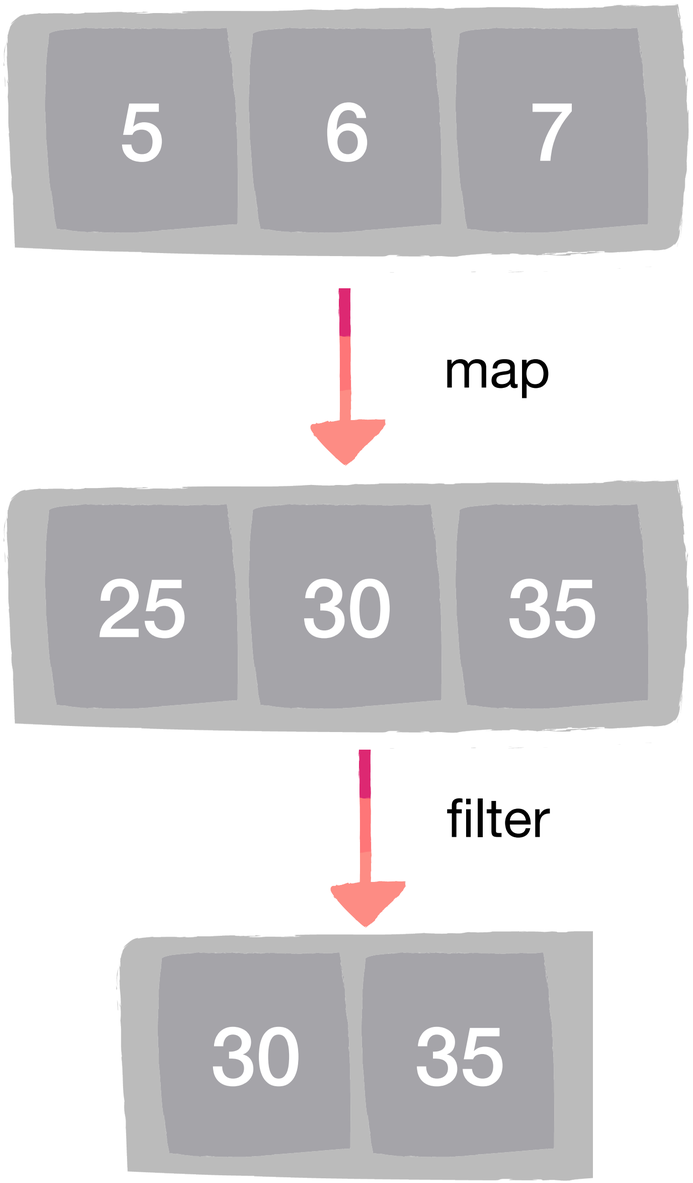
Kotlin collections

```
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

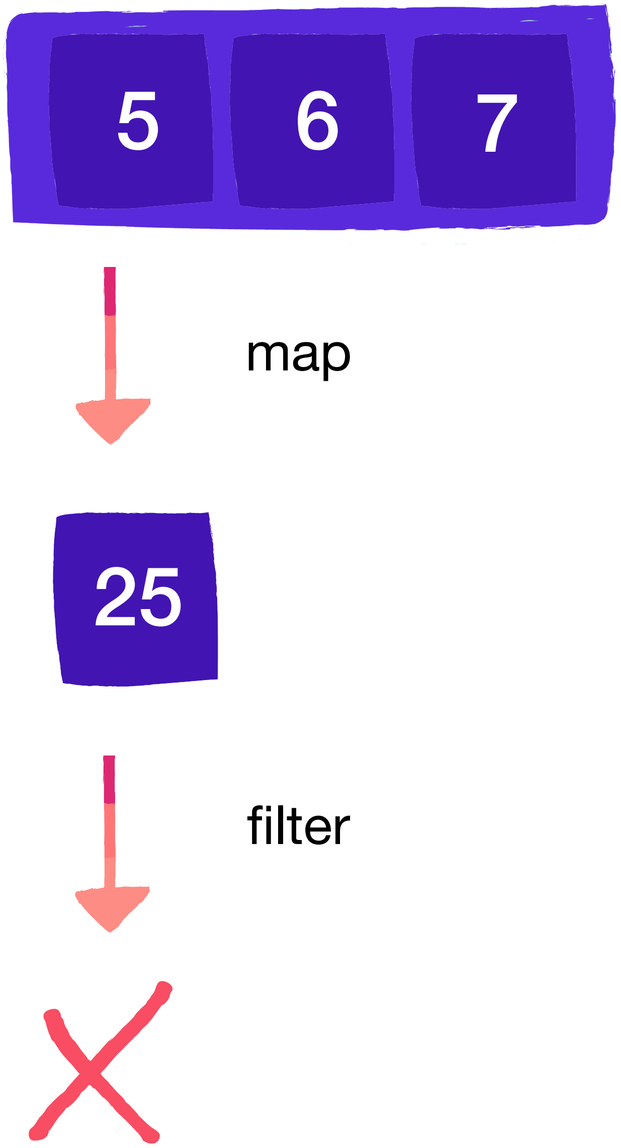
```
listOf(5, 6, 7)
    .asSequence()
    .map { it * 5 }
    .filter { it > 25 }
    .toList()
```

KotLIN SEQUENCES


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

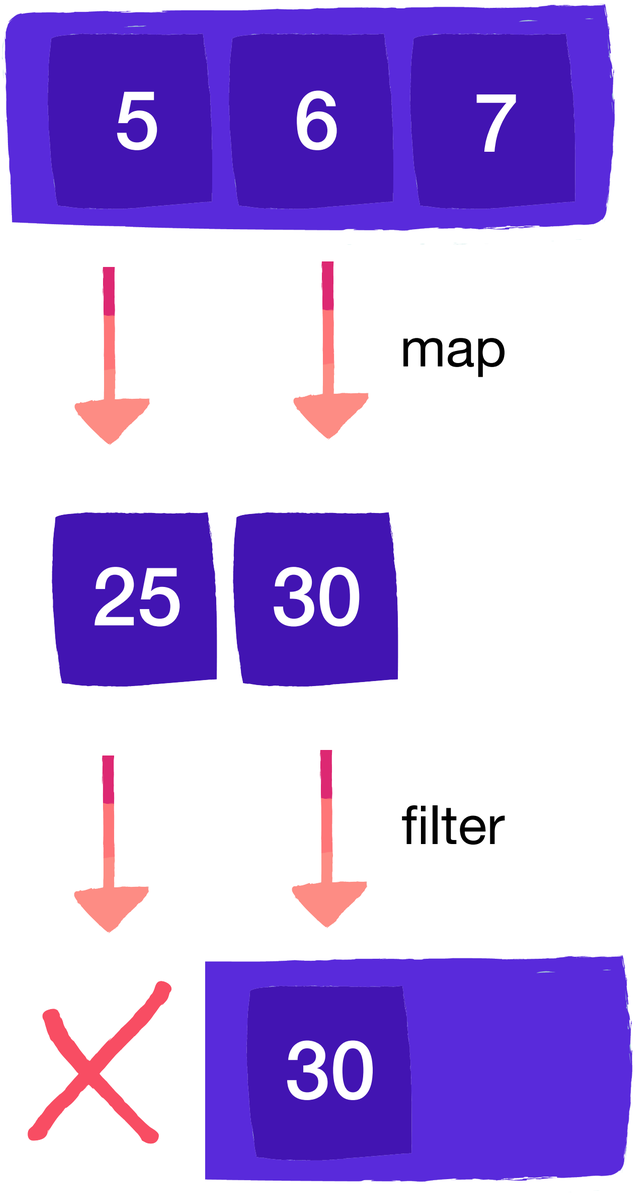
Kotlin sequences

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



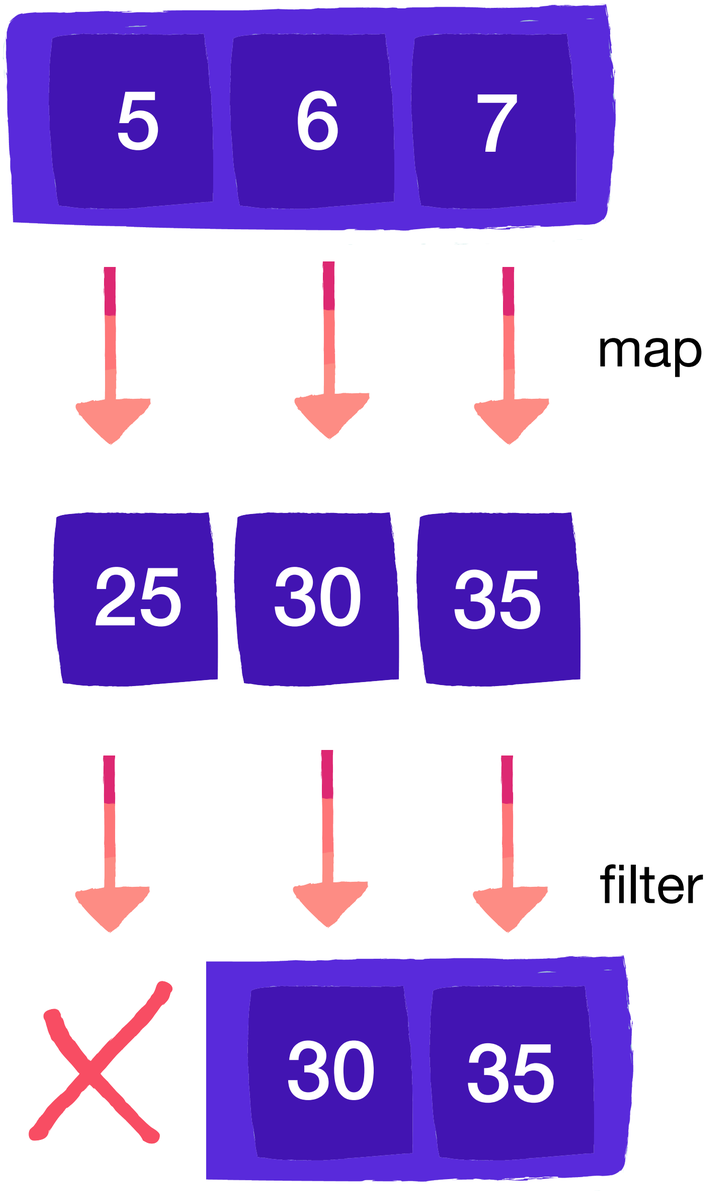
KotLIN SEQUENCES

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



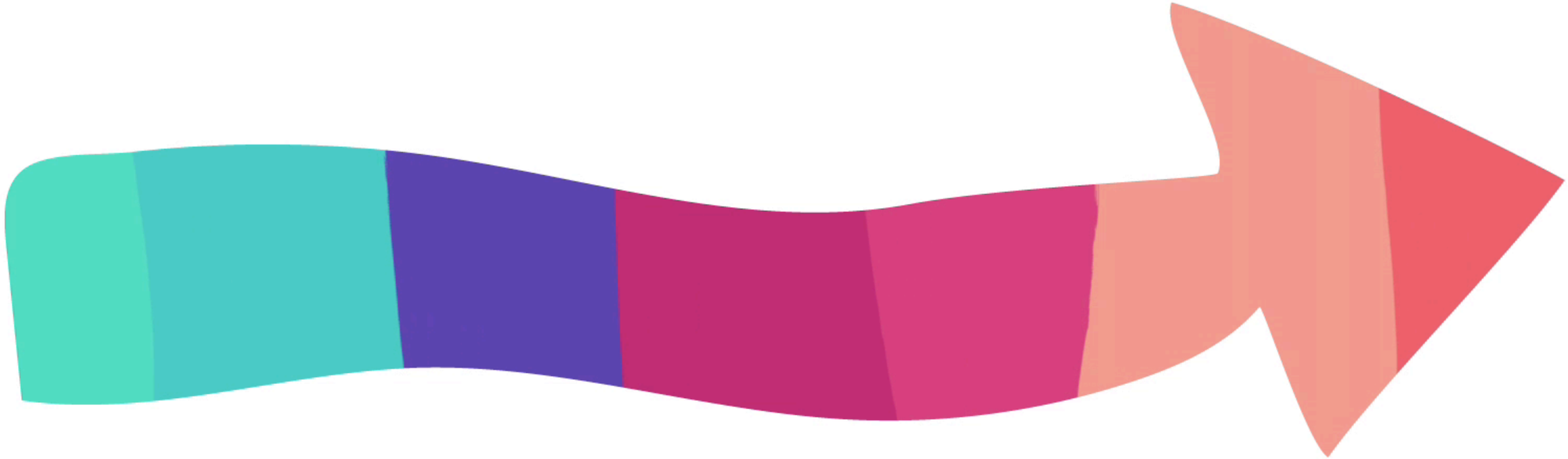
KotLIN SEQUENCES

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




WHY?

ASYNCHRONOUS DATA STREAMS

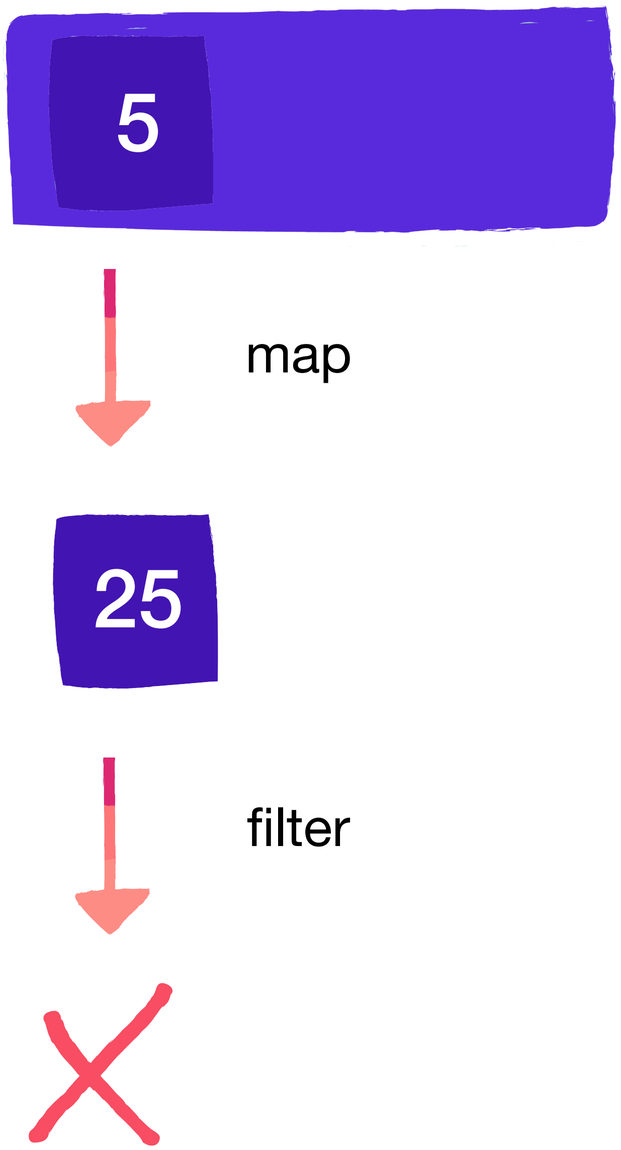


RXJAVA

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

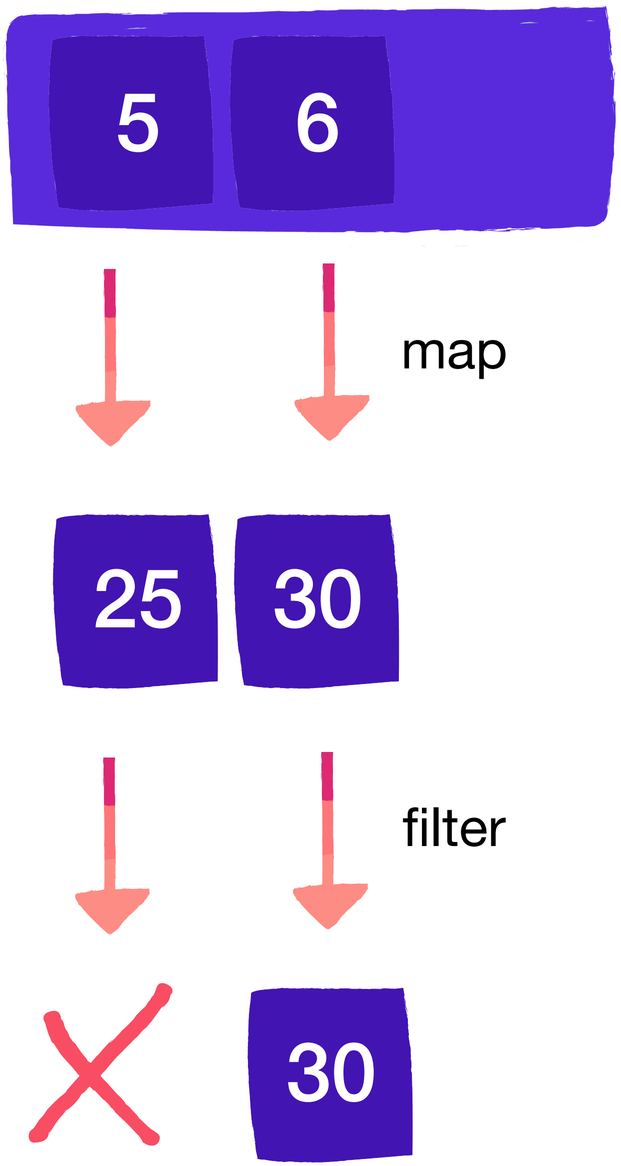
RXJAVA

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



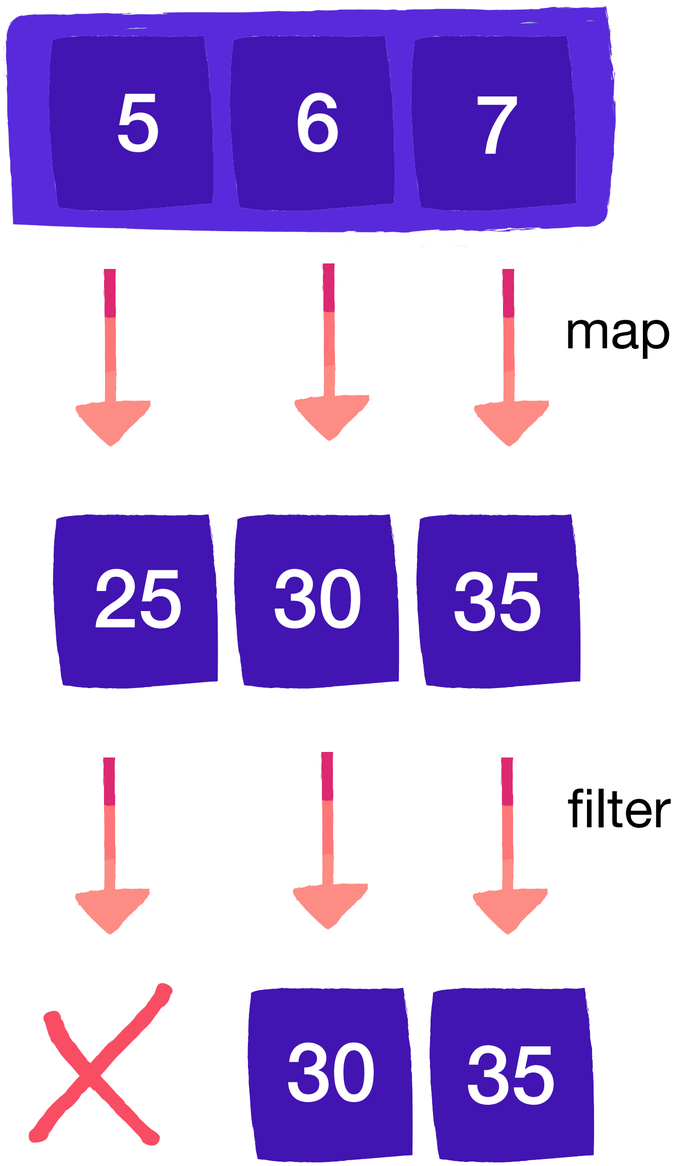
RXJAVA

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



RXJAVA

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

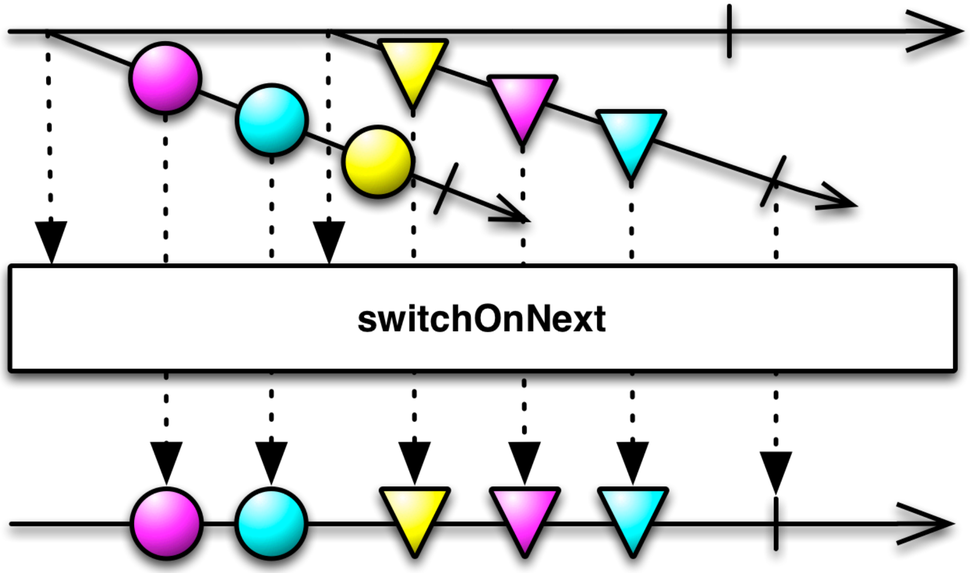


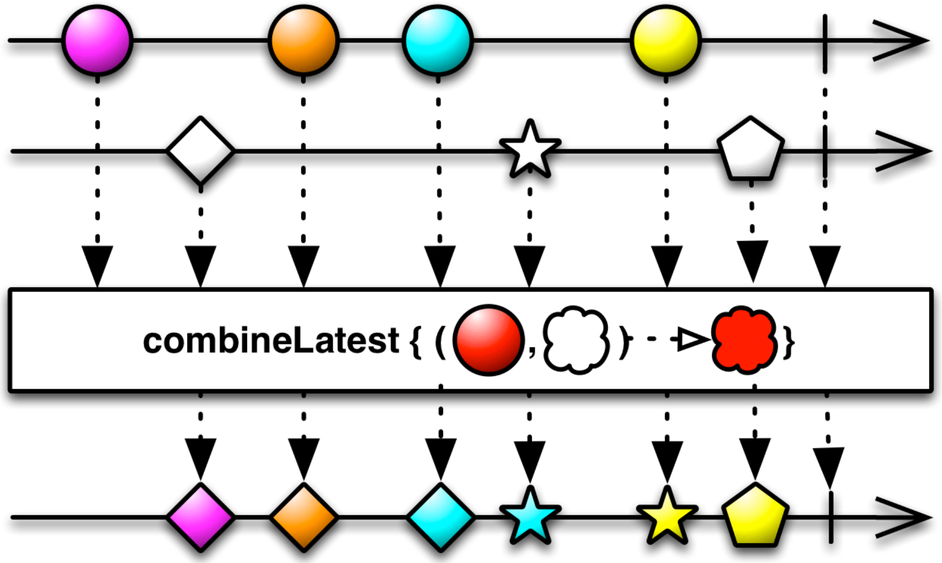
FLEXIBLE THREADING

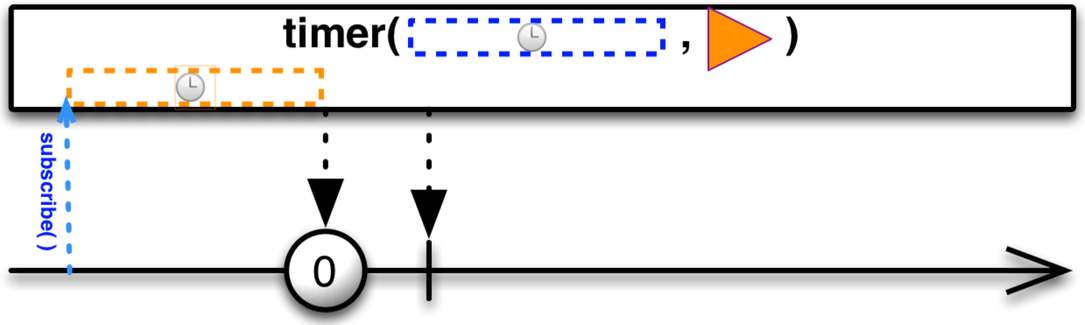
SCHEDULERS

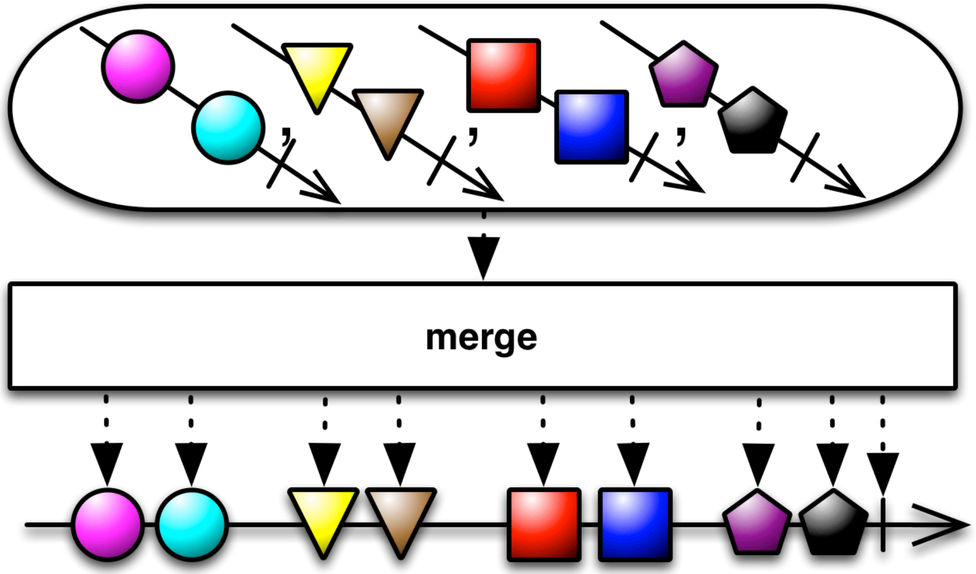
```
Observable.just(5, 6, 7)
          .map { ";-) ".repeat(it) }
          .subscribe { println(it) }
```

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


switchOnNext

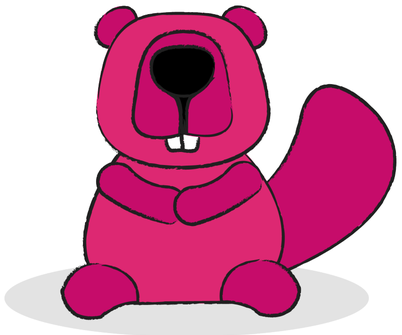

combineLatest { (,) } -


timer(,)
subscribe( )
0


merge

THE BASICS

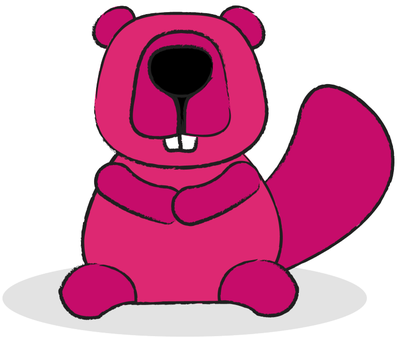
OBSERVABLE



THE BASICS

OBSERVER

OBSERVABLE

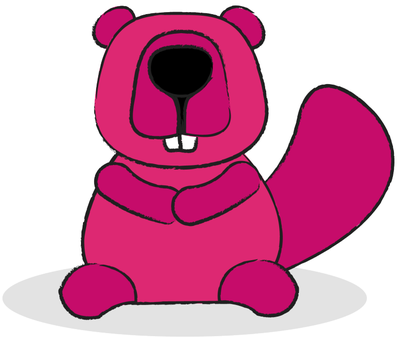


THE BASICS

OPERATORS

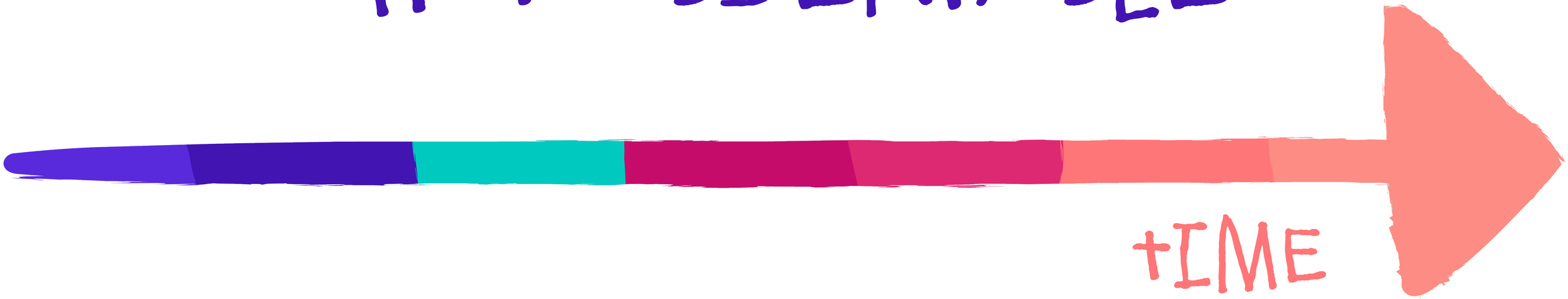
OBSERVER

OBSERVABLE

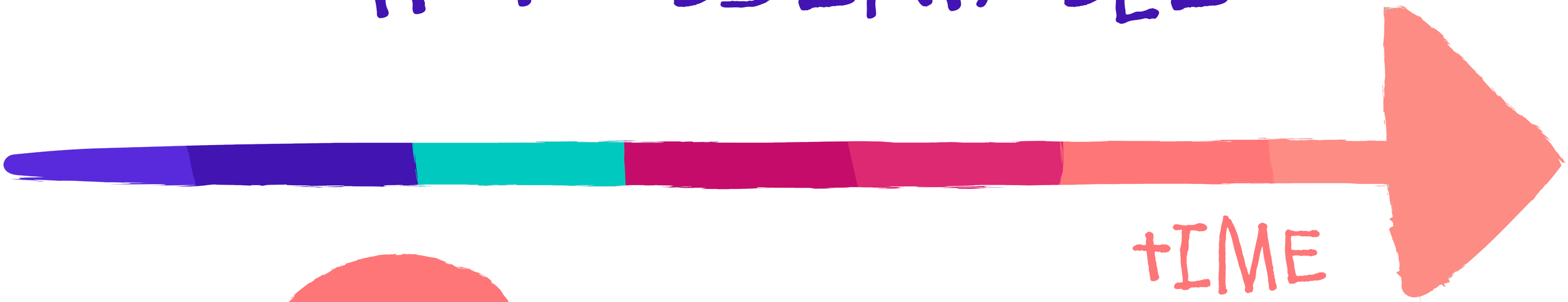


OBSERVABLE

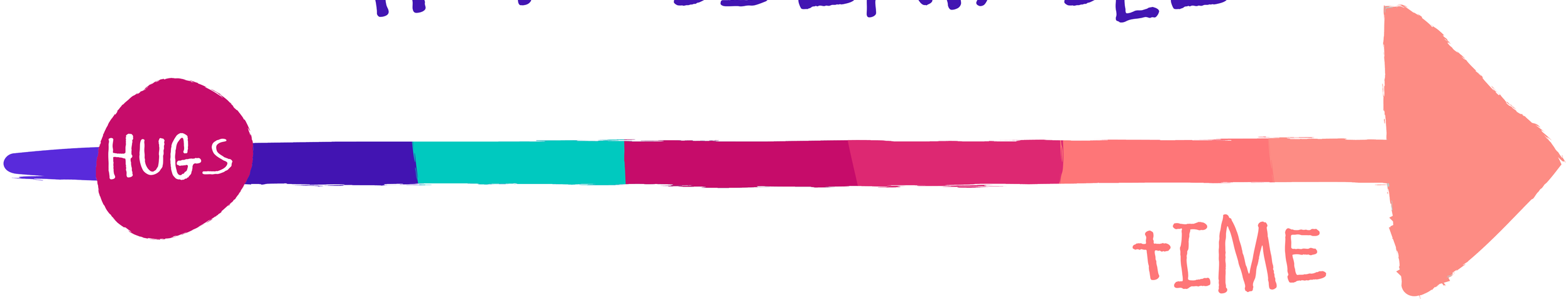
Hot OBSERVABLE

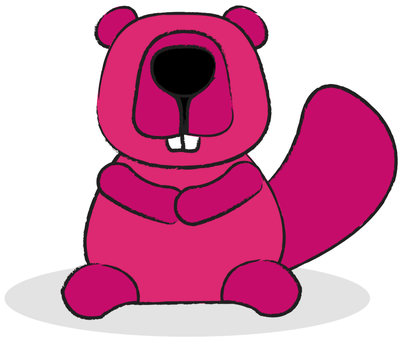


Hot OBSERVABLE

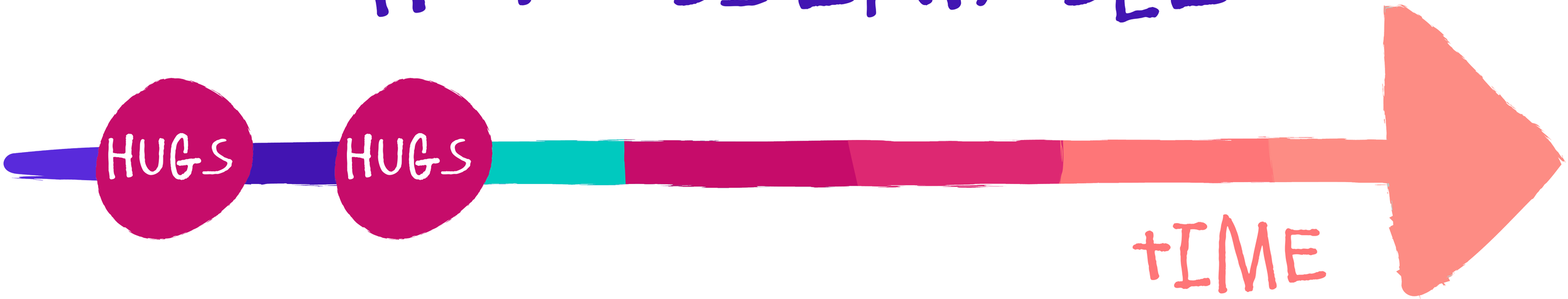


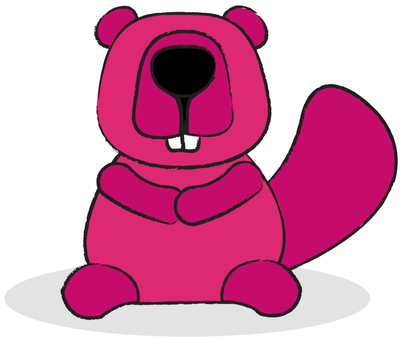
Hot OBSERVABLE



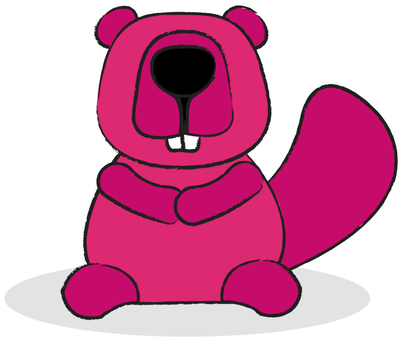


Hot OBSERVABLE





Hot OBSERVABLE



CLICK EVENTS

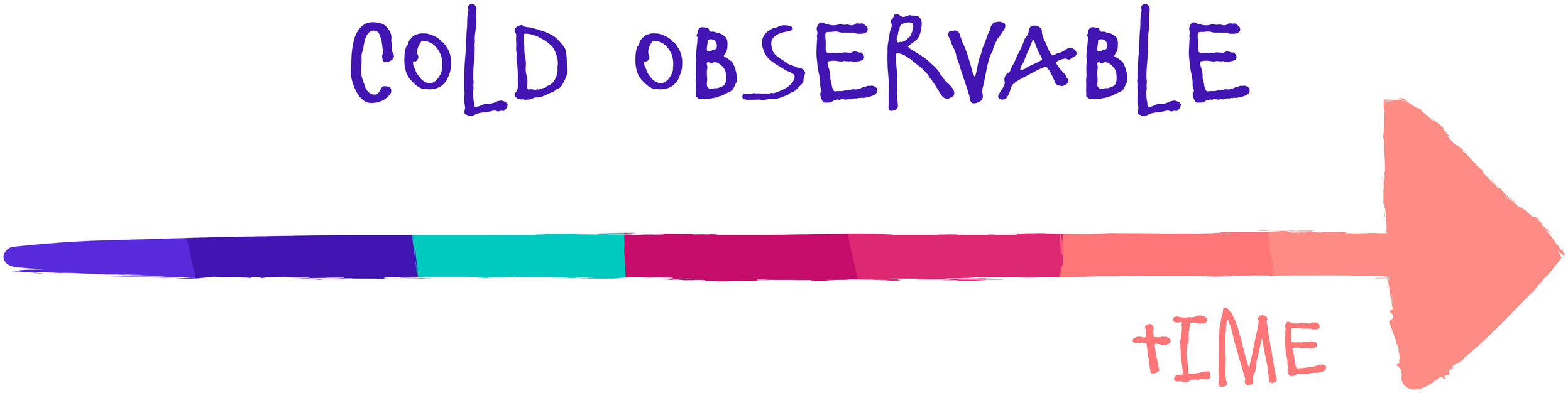
PUSH NOTIFICATIONS

KEYBOARD INPUT

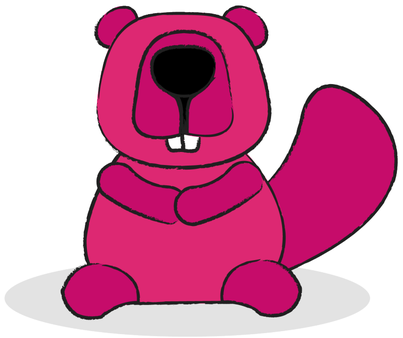
READING A FILE

DATABASE ACCESS

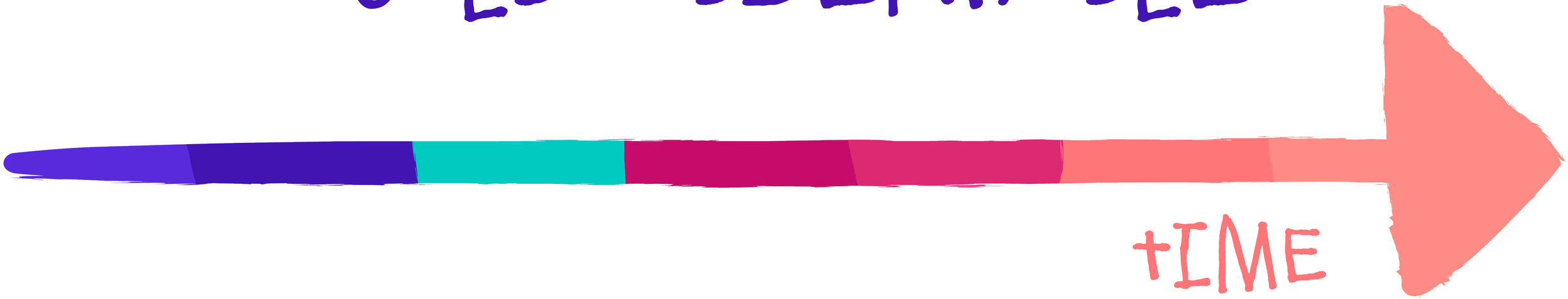
DEVICE SENSOR UPDATES

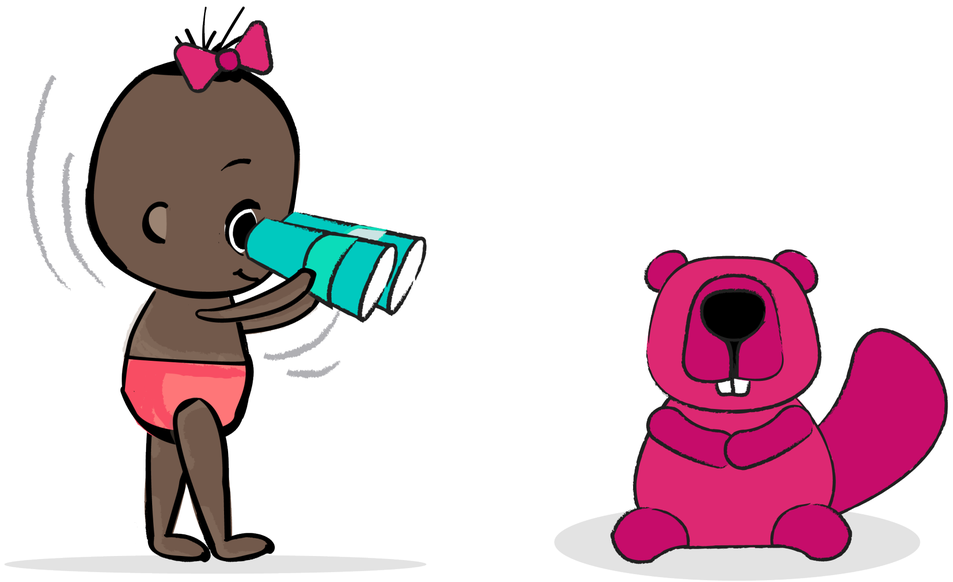

COLD OBSERVABLE
TIME

COLD OBSERVABLE



COLD OBSERVABLE

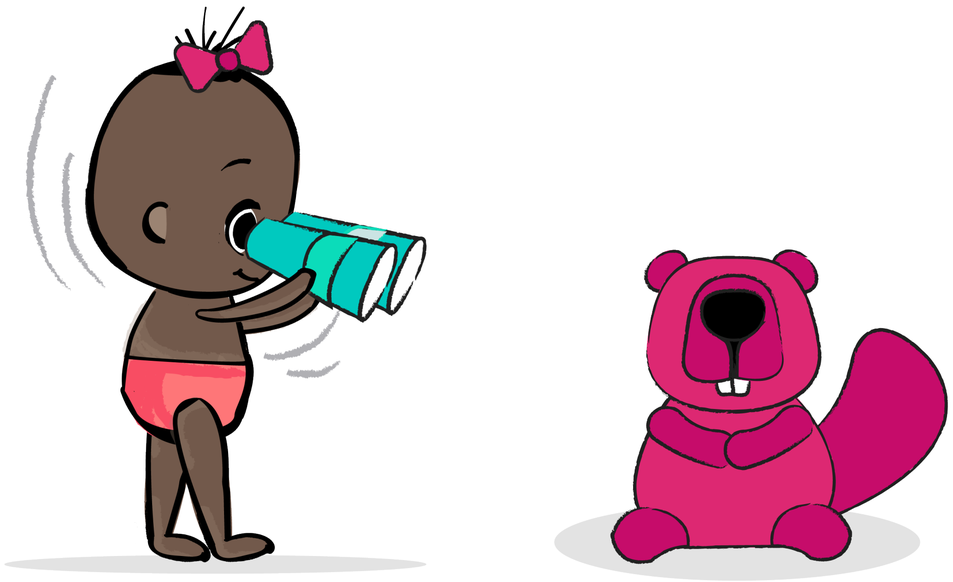




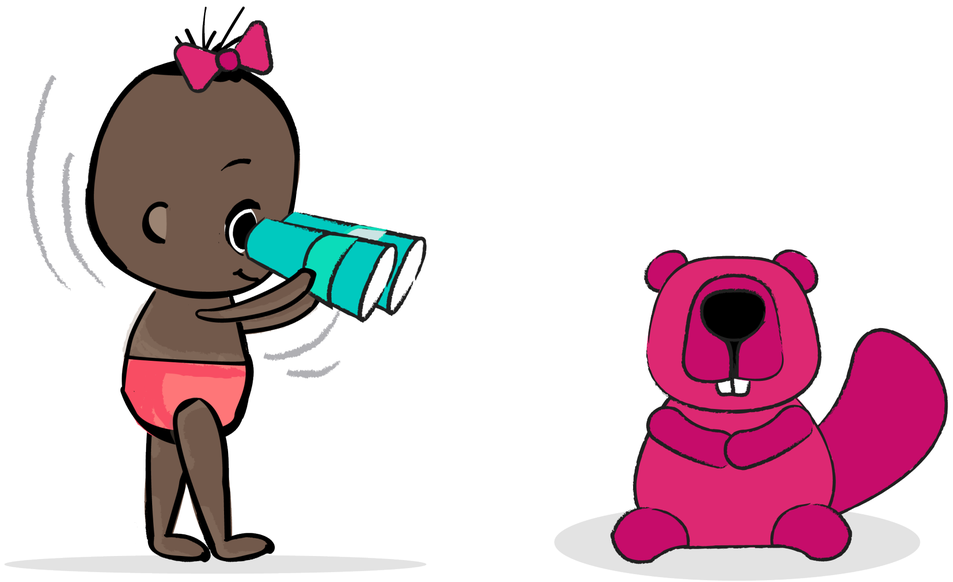
COLD OBSERVABLE

HUGS

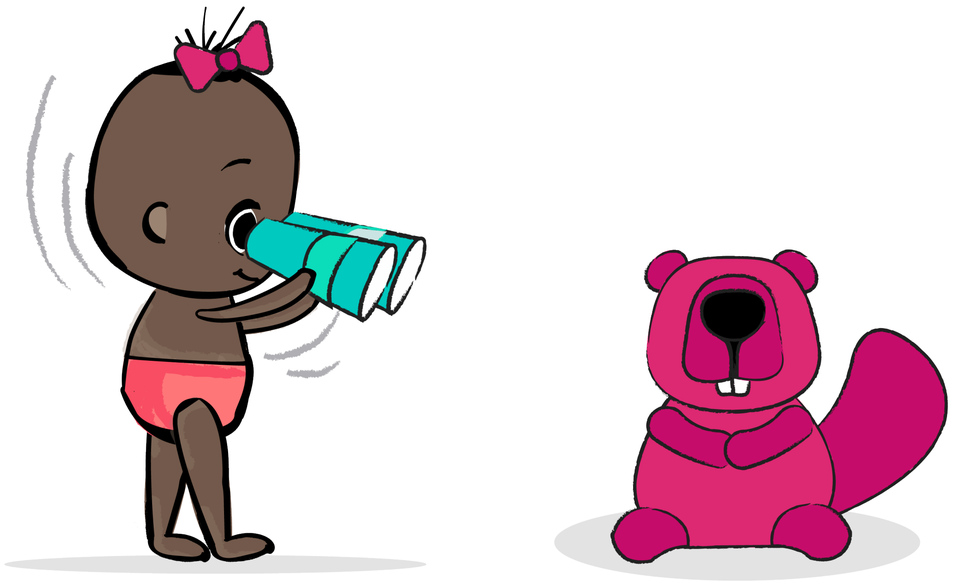
TIME



COLD OBSERVABLE



COLD OBSERVABLE



CLICK EVENTS

PUSH NOTIFICATIONS

KEYBOARD INPUT

READING A FILE

DATABASE ACCESS

DEVICE SENSOR UPDATES

WHERE?

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

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

Observable.just(item1, item2, item3)
```

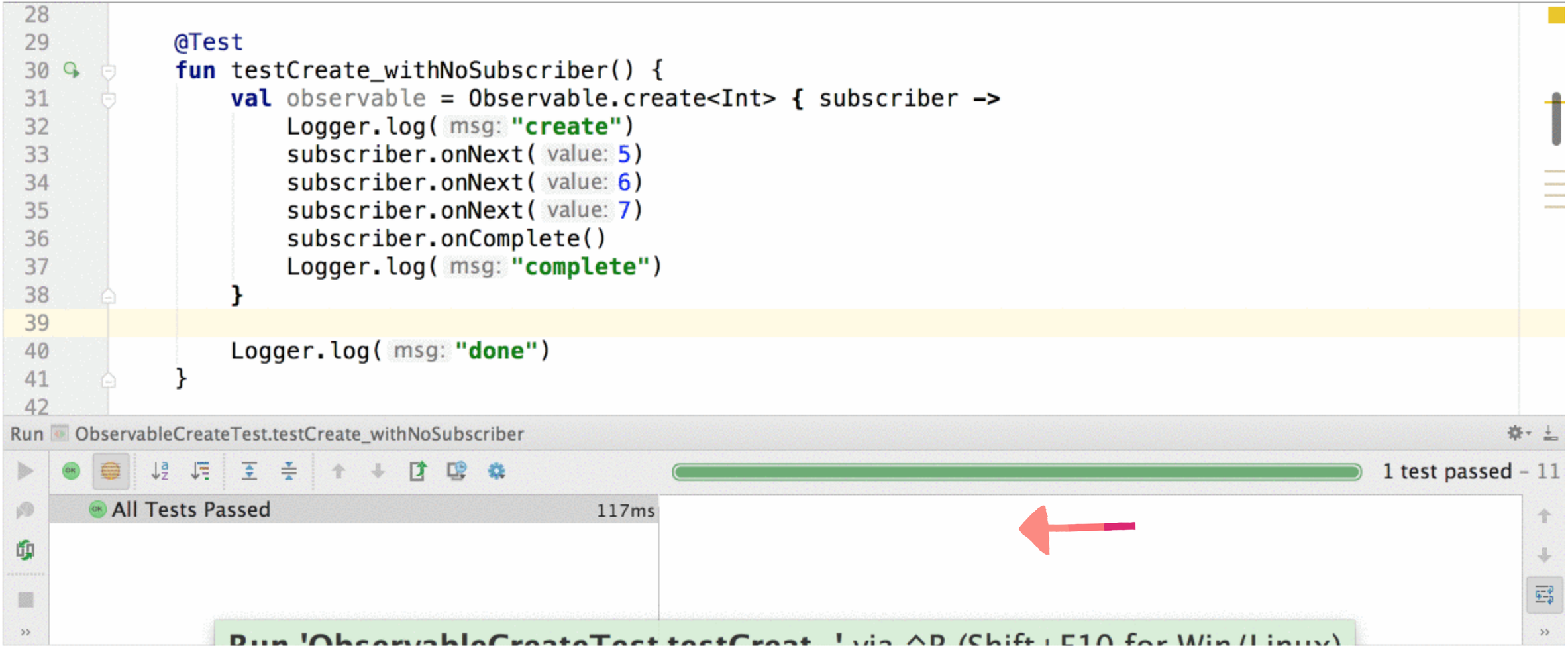
```
Observable.create<Int> { subscriber -> }

Observable.just(item1, item2, item3)

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

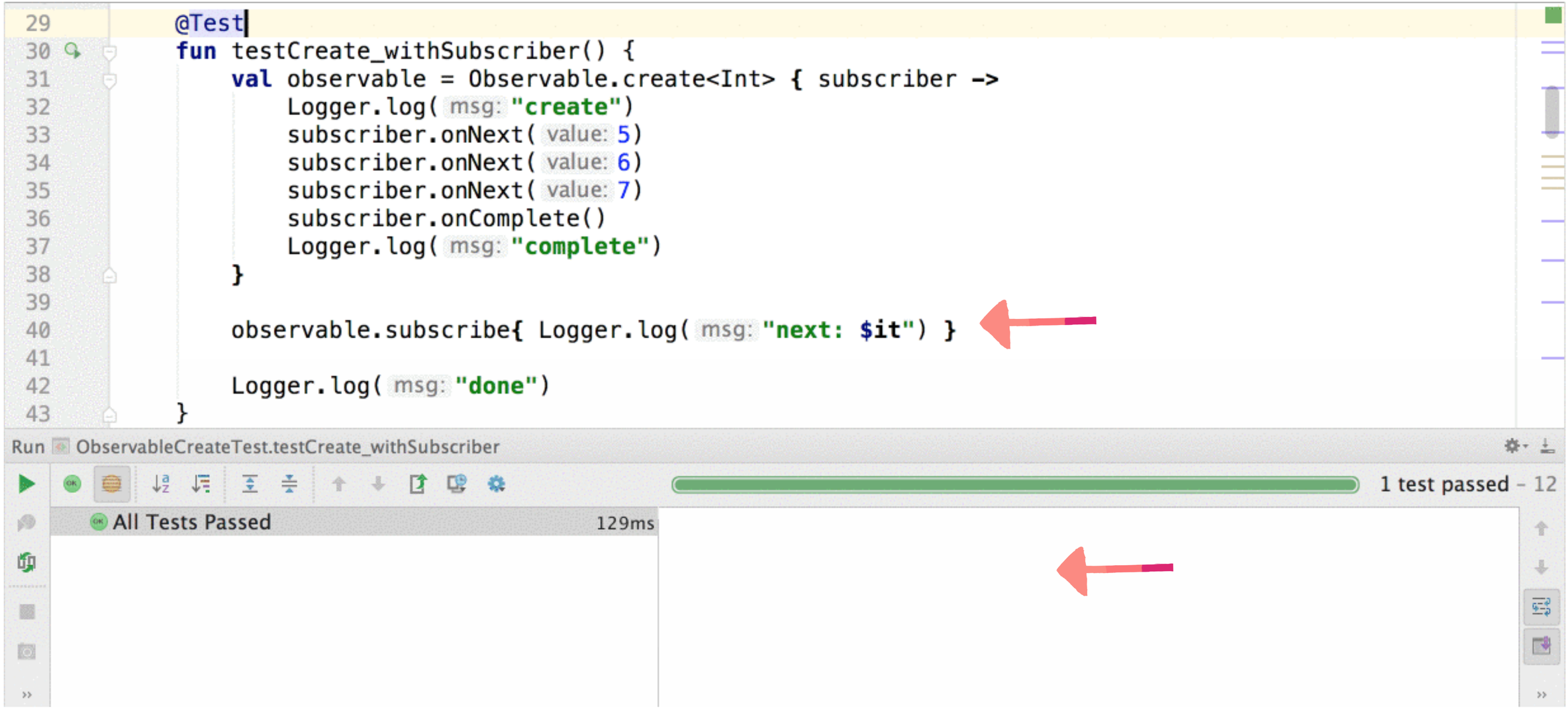
```
Observable.create<Int> { subscriber ->



}
```

```
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 – 11
All Tests Passed 117ms

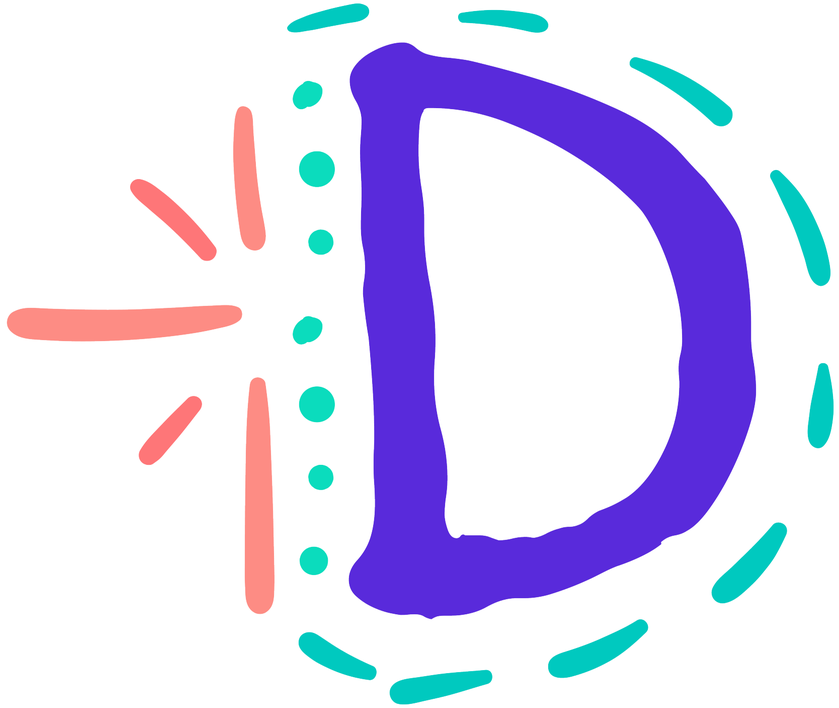

29 @Test
30 fun testCreate_withSubscriber() {
31 val observable = Observable.create<Int> { subscriber ->
32 Logger.log(msg: "create")
33 subscriber.onNext(value: 5)
34 subscriber.onNext(value: 6)
35 subscriber.onNext(value: 7)
36 subscriber.onComplete()
37 Logger.log(msg: "complete")
38 }
39
40 observable.subscribe{ Logger.log(msg: "next: $it") }
41
42 Logger.log(msg: "done")
43 }
Run ObservableCreateTest.testCreate_withSubscriber
1 test passed - 12
All Tests Passed 129ms

OBSERVABLES CAN. . .

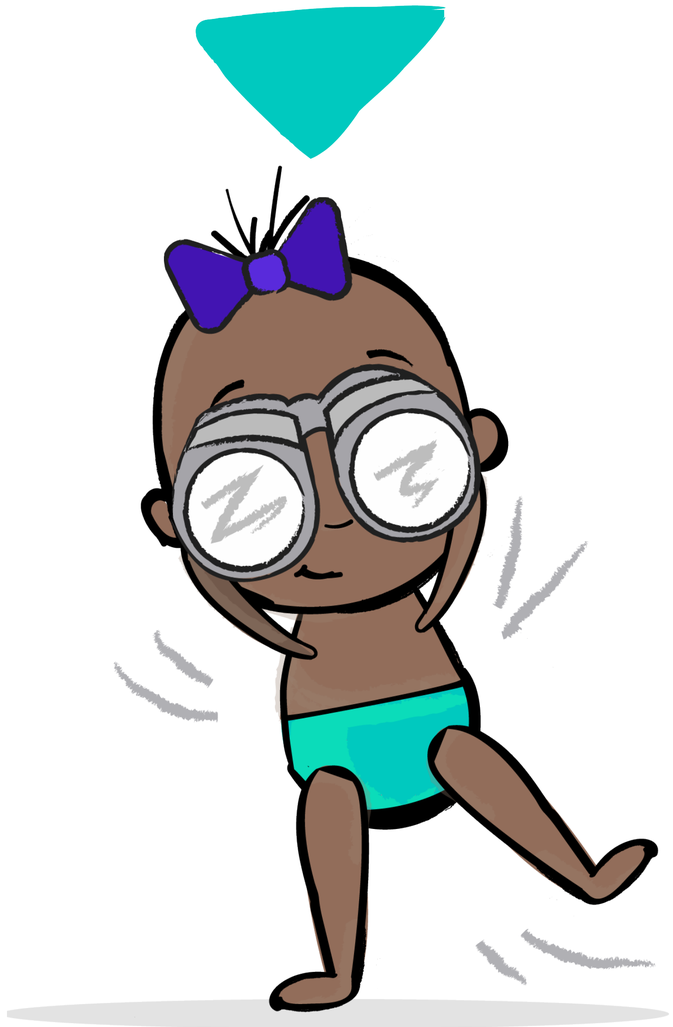
- A: EMIT ITEMS
- B: BE COLD
- C: BE HOT
- D: ALL OF THE ABOVE

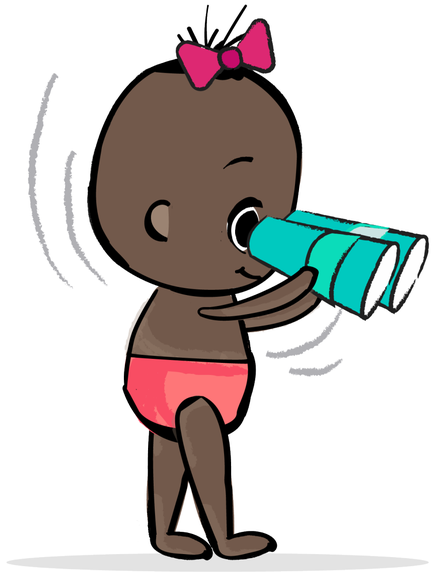
OBSERVABLES CAN. . .

- A: EMIT ITEMS
- B: BE COLD
- C: BE HOT
- D: ALL OF THE ABOVE

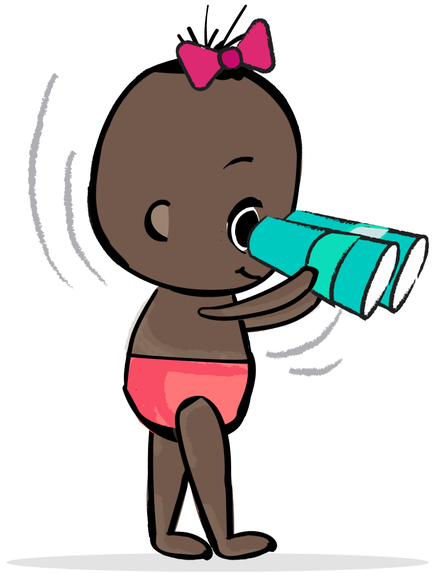


OBSERVER

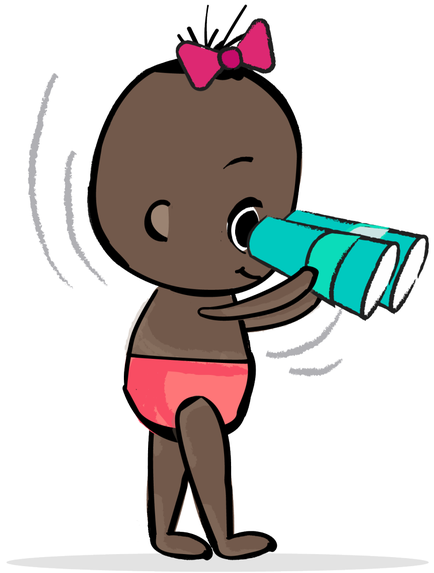




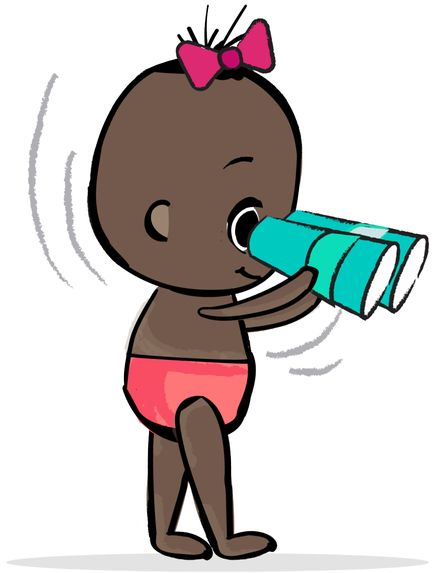
```
interface Observer<T> {
    fun onError(e: Throwable)

    fun onComplete()

    fun onNext(t: T)

    fun onSubscribe(d: Disposable)
}
```



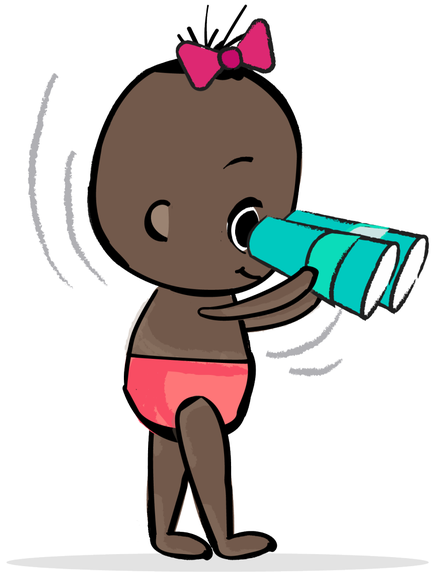
```
interface Observer<T> {
    fun onError(e: Throwable)

    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)
}
```

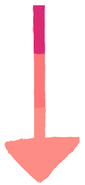


```
interface Observer<T> {
    fun onError(e: Throwable)

    fun onComplete()

    fun onNext(t: T)

    fun onSubscribe(d: Disposable)
}
```



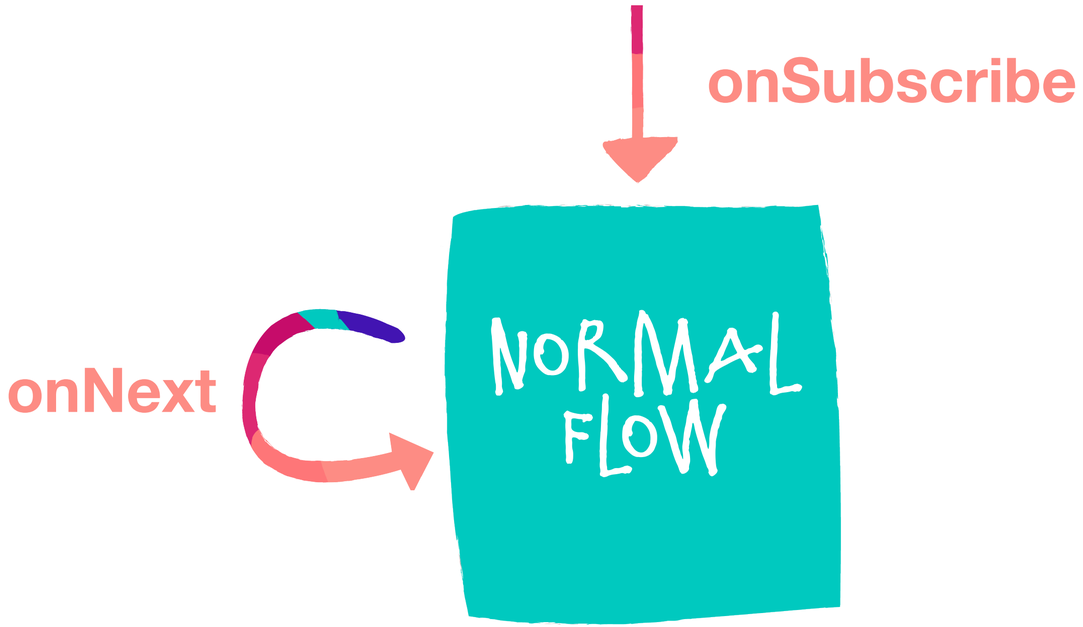
```
interface Observer<T> {
    fun onError(e: Throwable)

    fun onComplete()

    fun onNext(t: T)

    fun onSubscribe(d: Disposable)
}
```

OBSERVER'S LIFECYCLE

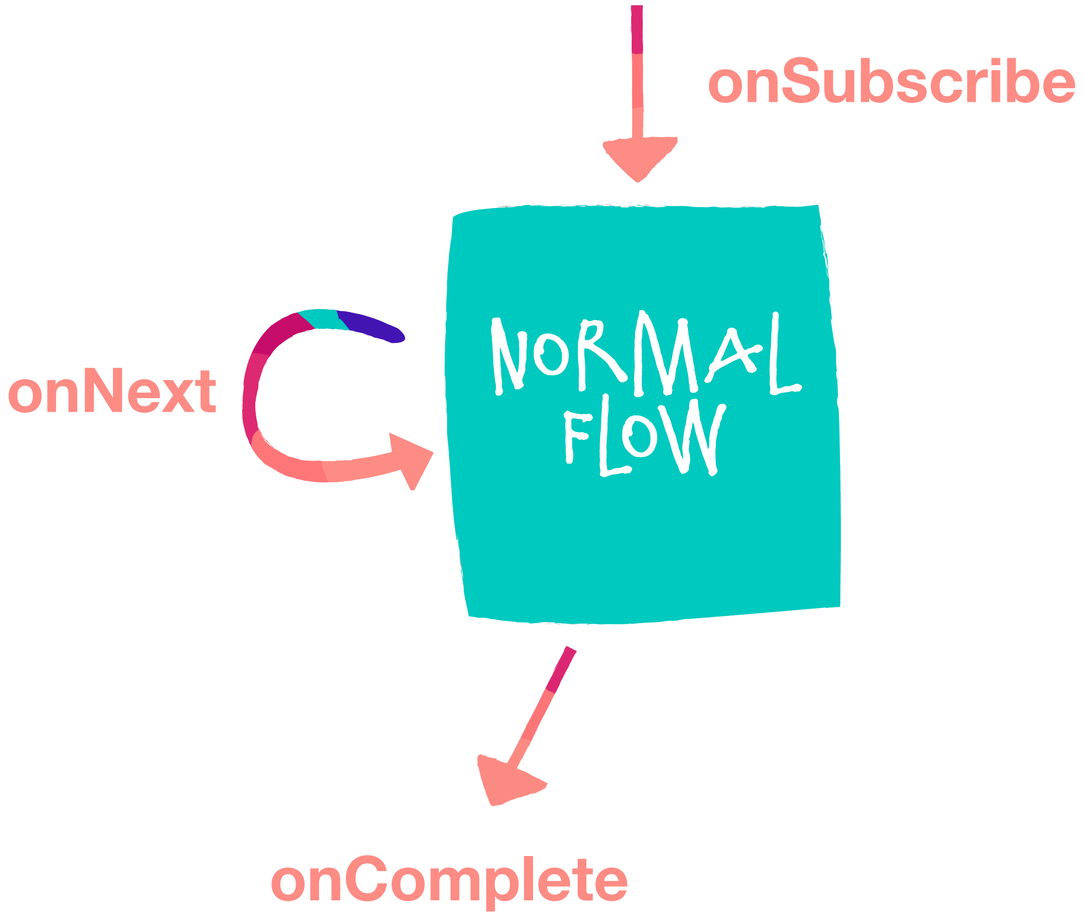


onSubscribe

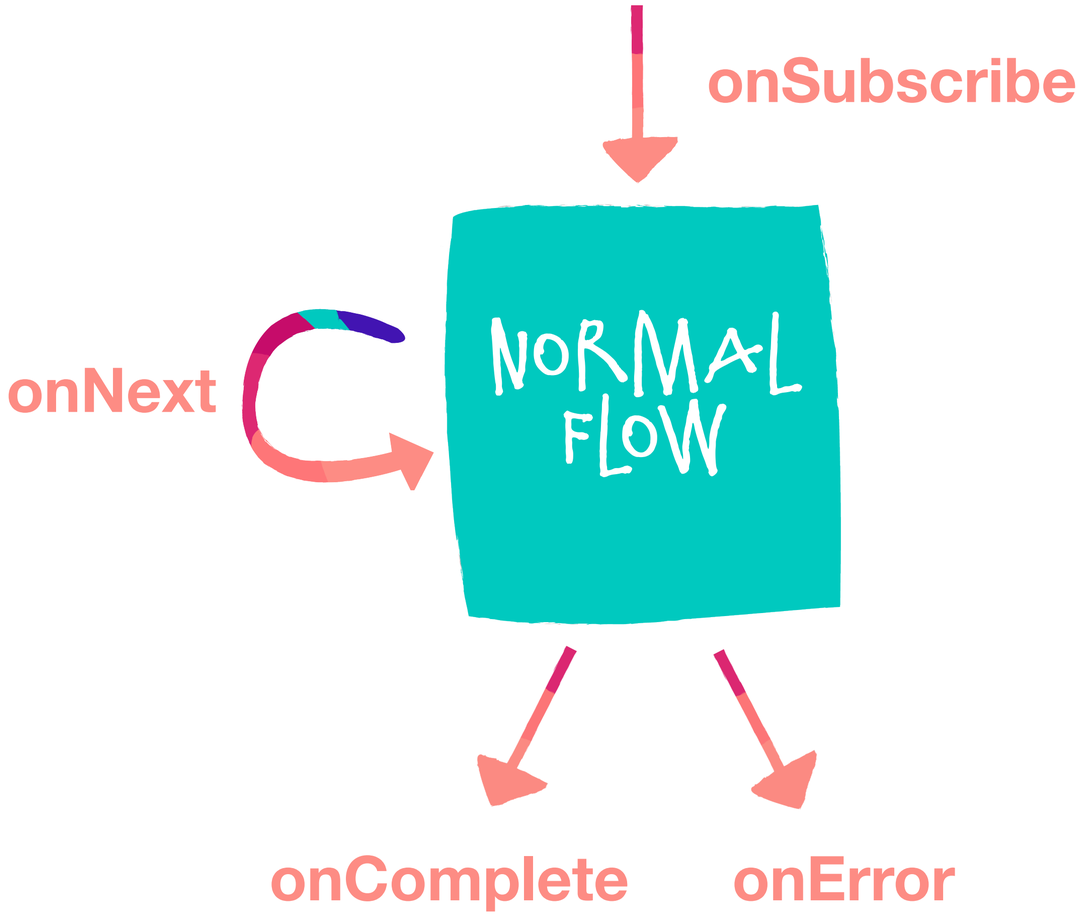
OBSERVER'S LIFECYCLE



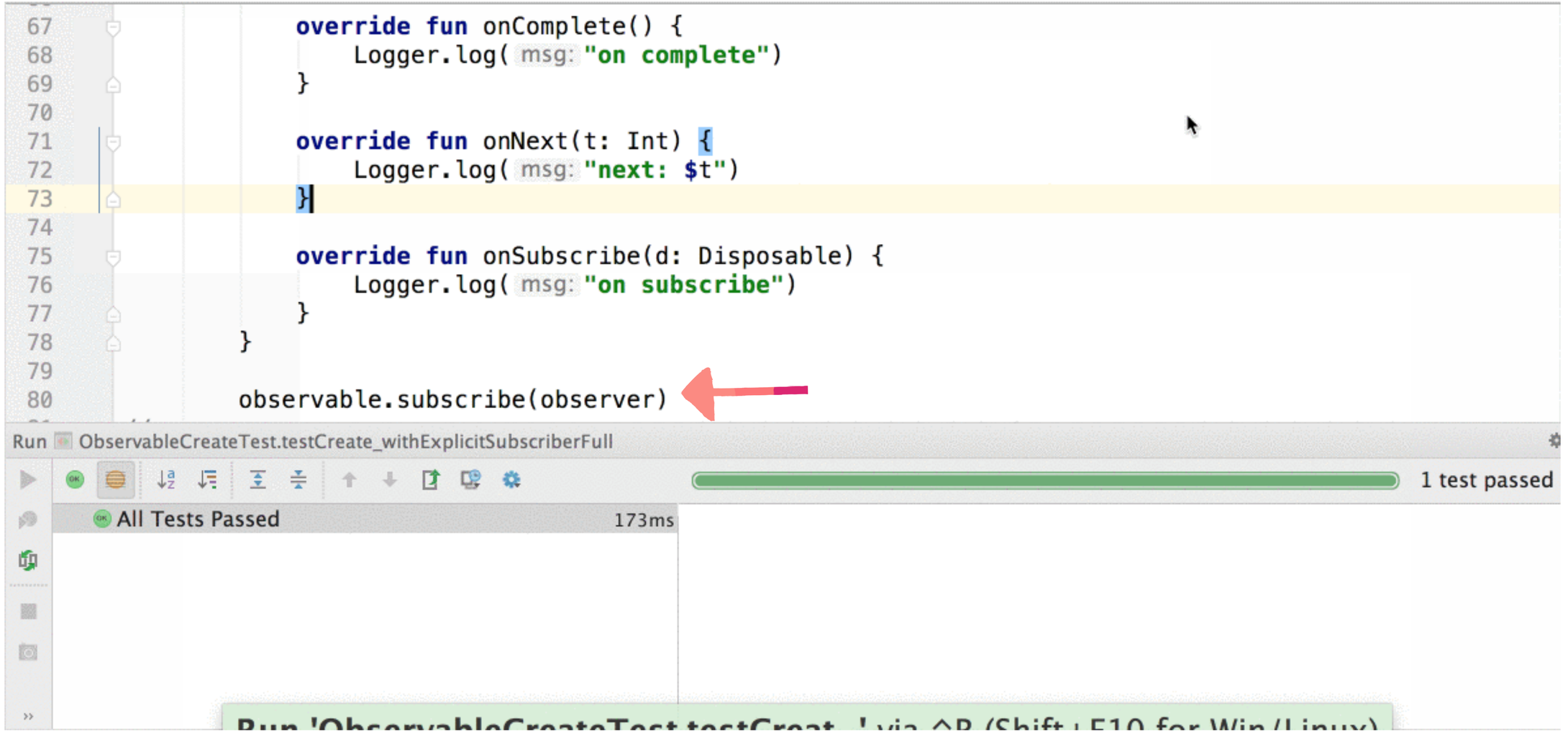
OBSERVER'S LIFECYCLE



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 onNext(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
1 test passed
All Tests Passed
173ms

```
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
```

```
            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)
    }
```

ervableCreateTest.testCreate_withExplicitSubscriberFull

All Tests Passed 148ms

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

Consumer

```
val consumer = 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. . .

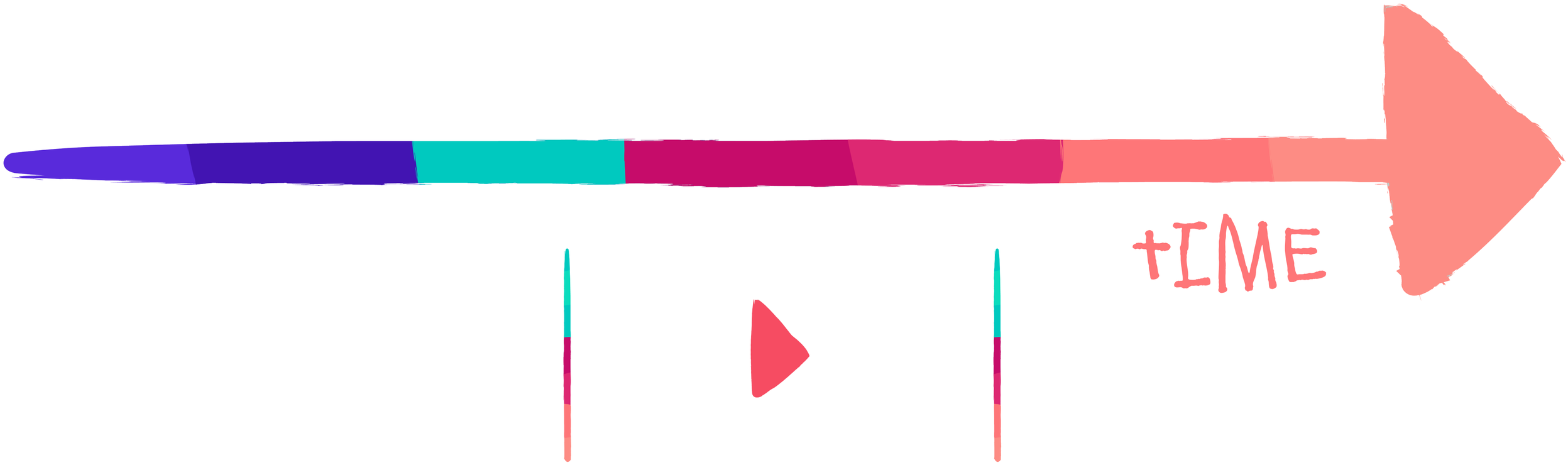
- A: HAVE A LIFECYCLE
- B: ALWAYS COMPLETE
- C: NEVER ERROR
- D: ALL OF THE ABOVE

OBSERVERS. . .

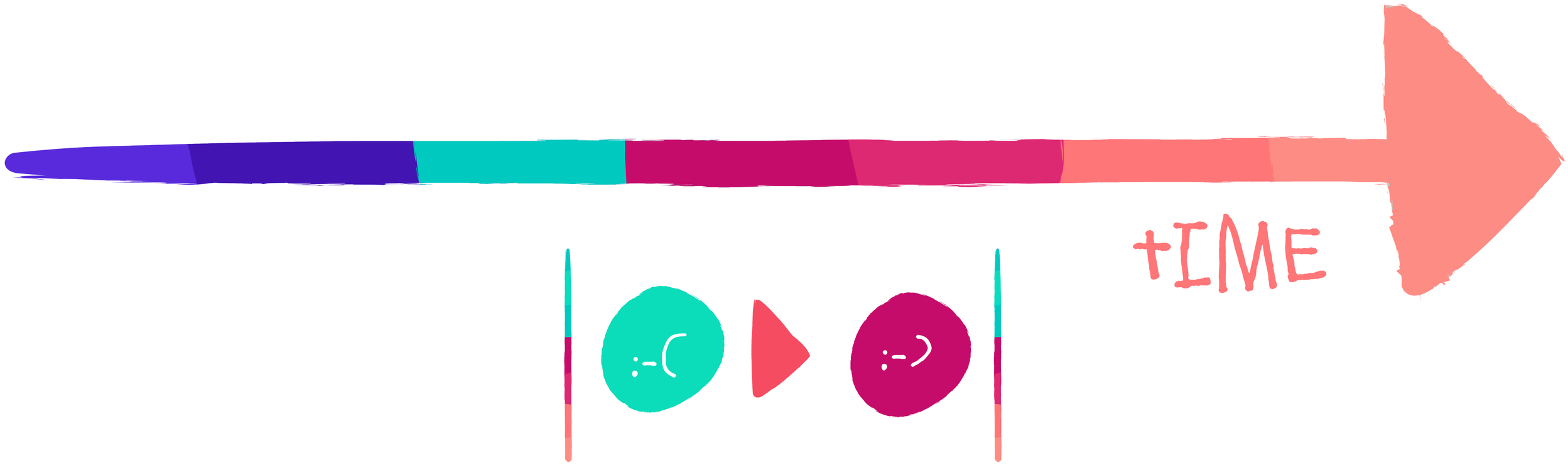
- A: HAVE A LIFECYCLE
- B: ALWAYS COMPLETE
- C: NEVER ERROR
- D: ALL OF THE ABOVE

OPERATOR

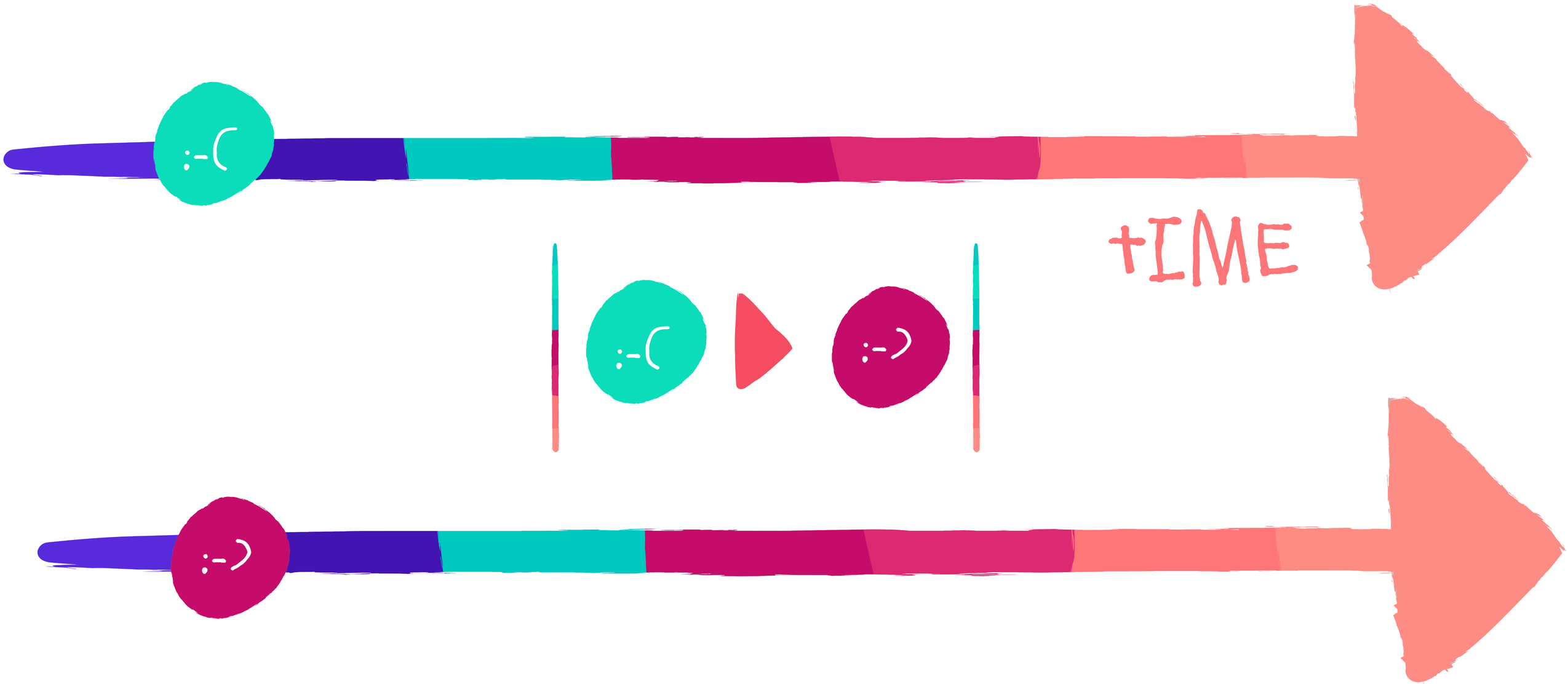
OPERATOR: MAP()



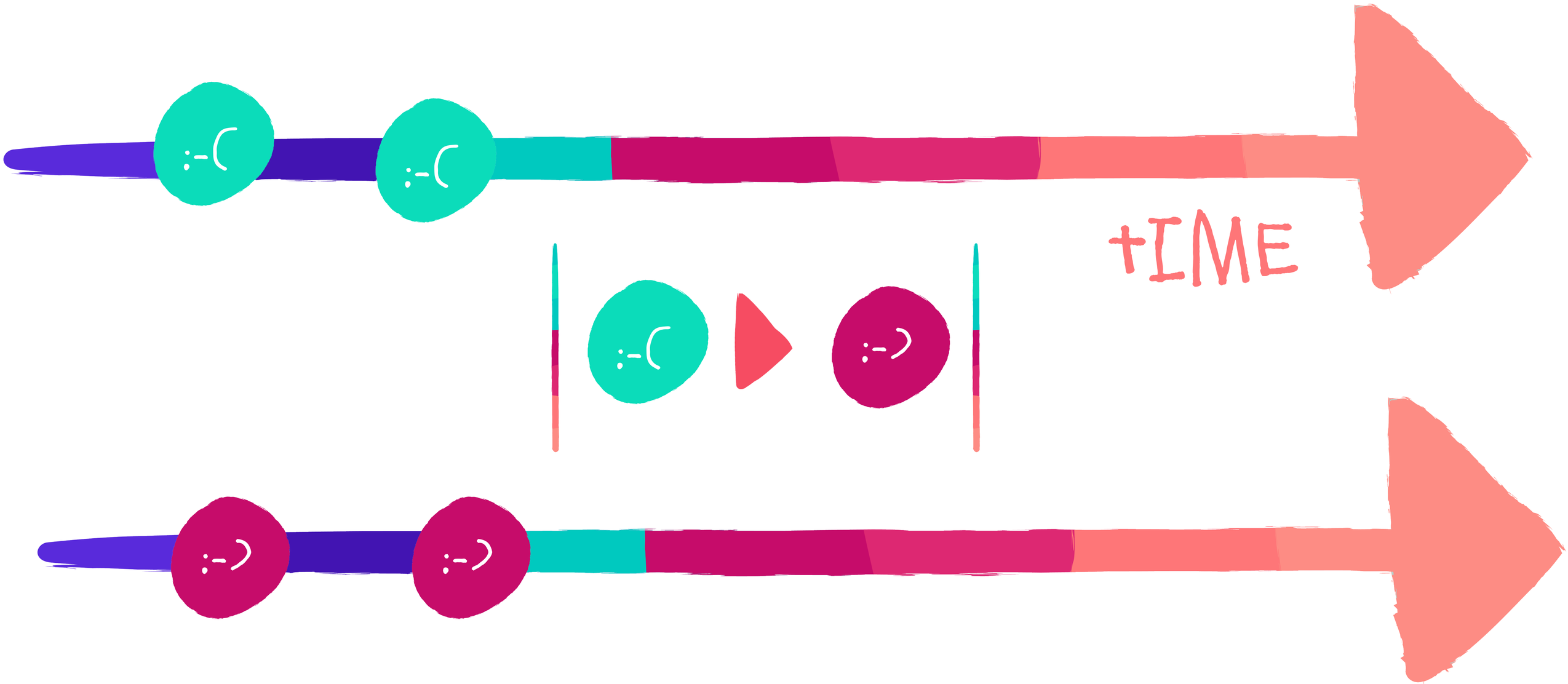
OPERATOR: MAP()



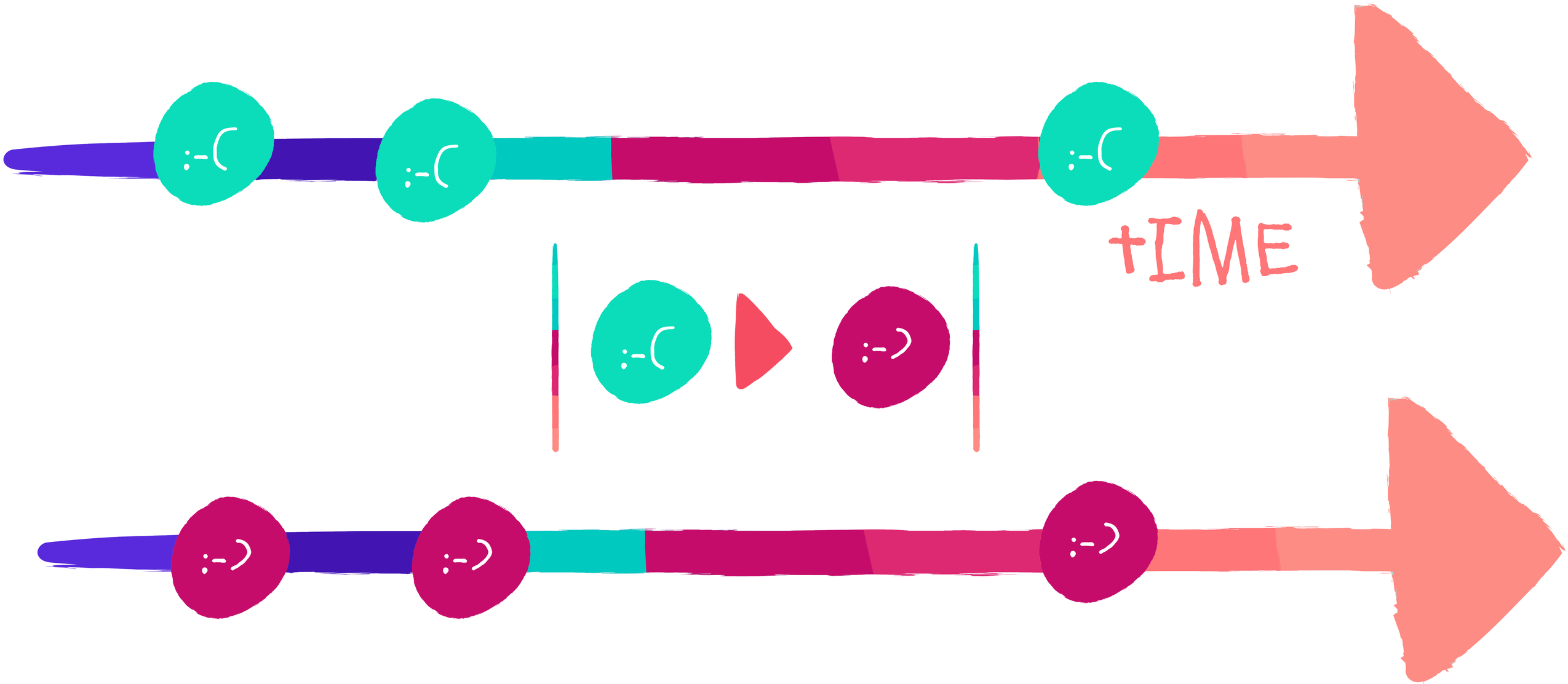
OPERATOR: MAP()



OPERATOR: MAP()



OPERATOR: MAP()



Operator: map()

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

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) }
```

```
.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) }
```

WHAT'S THE OUTPUT?

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


A: 1, 2, 3


B: A, B, C

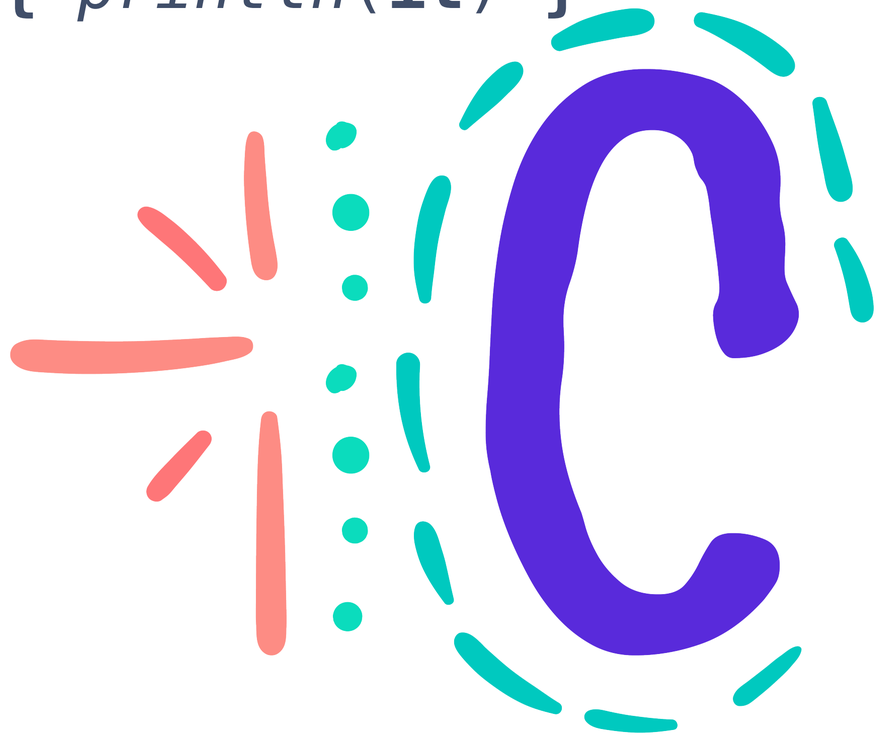

C: 2, 4, 6


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



WHAT'S THE OUTPUT?

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

WHAT'S THE OUTPUT?

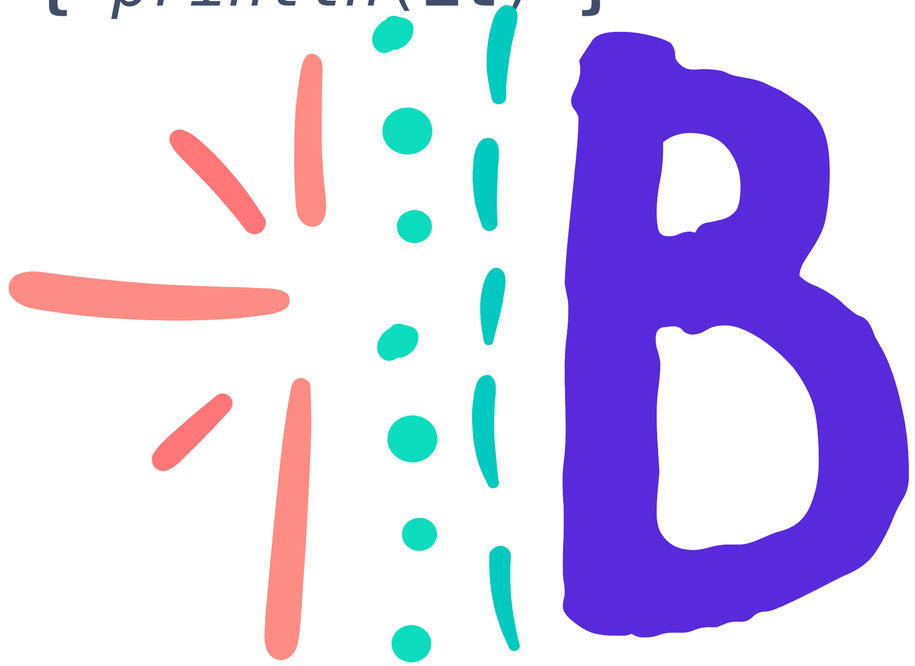
```
Observable.just(1, 2, 3)
        .map { it * 2 }
        .filter { it < 6 }
        .subscribe { println(it) }
```

- A: 2, 3, 1
- B: 2, 4
- C: 1, 2, 3
- D: 6, 4

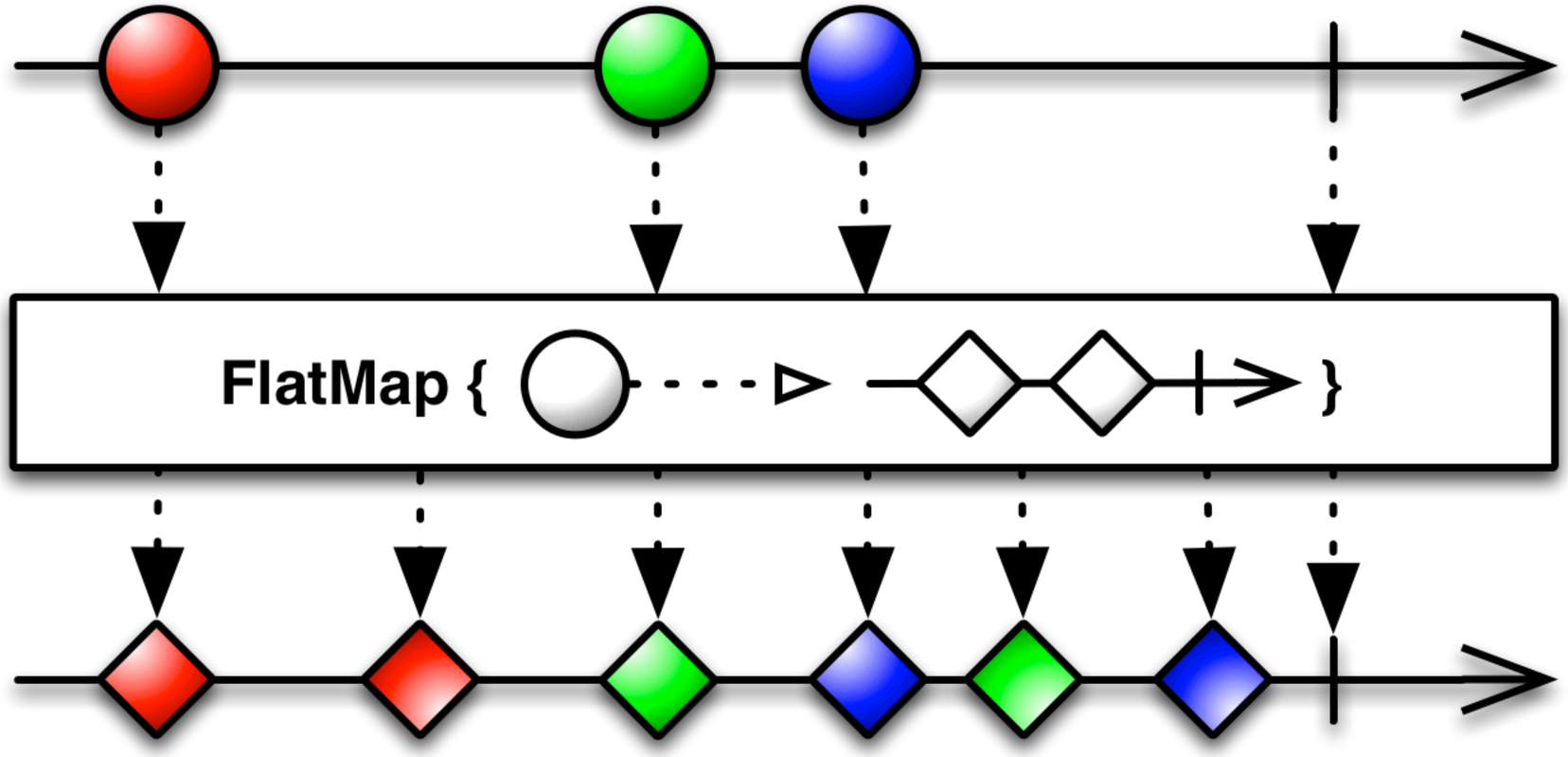
WHAT'S THE OUTPUT?

```
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

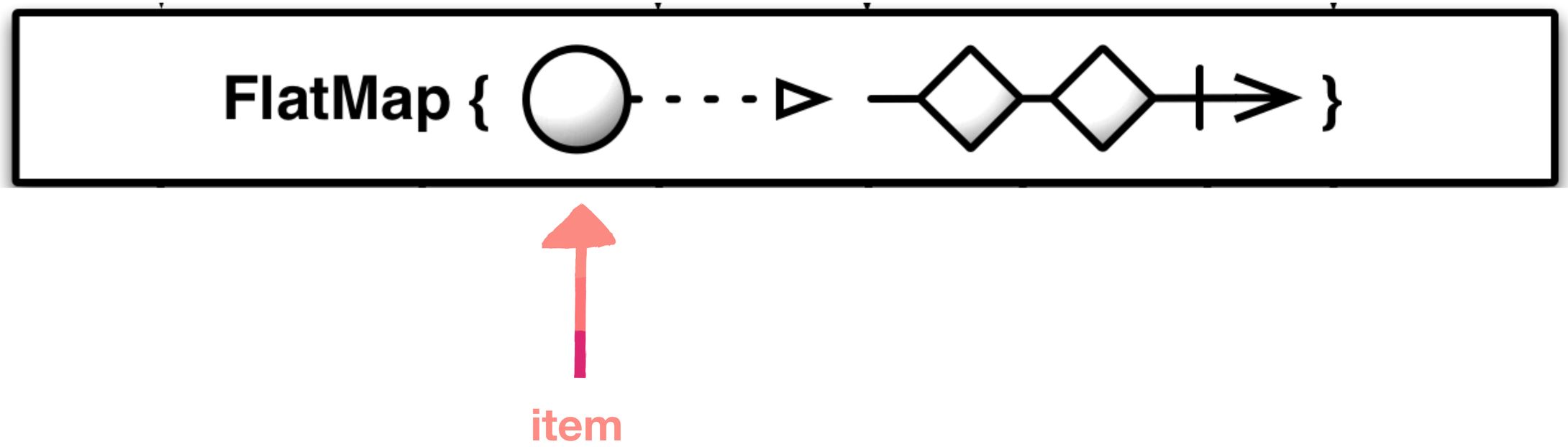


OPERATOR: FLATMAP()



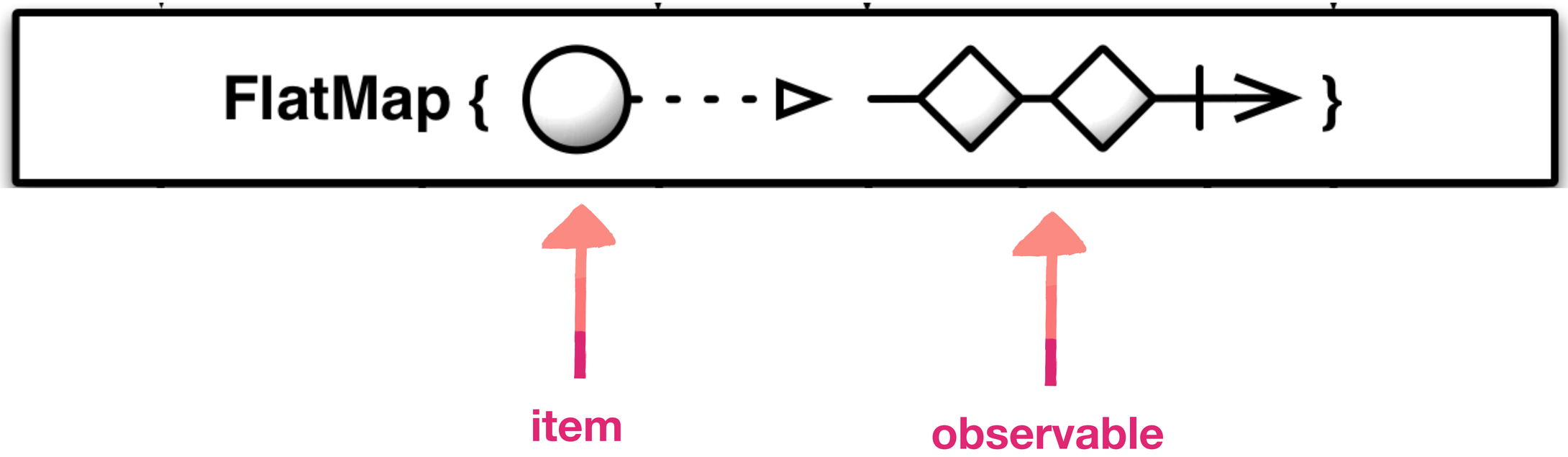
via reactivex.io

OPERATOR: FLATMAP()



via reactivex.io

OPERATOR: FLATMAP()

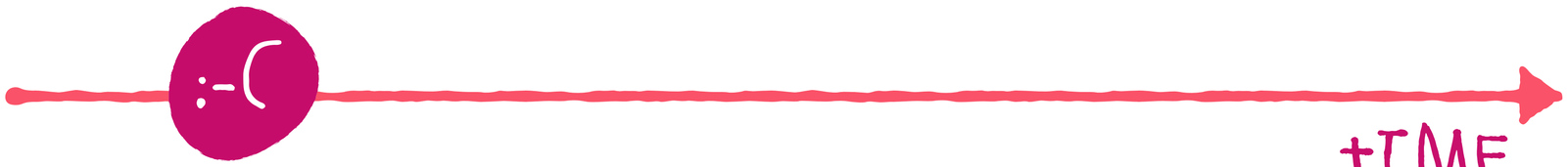


via reactivex.io

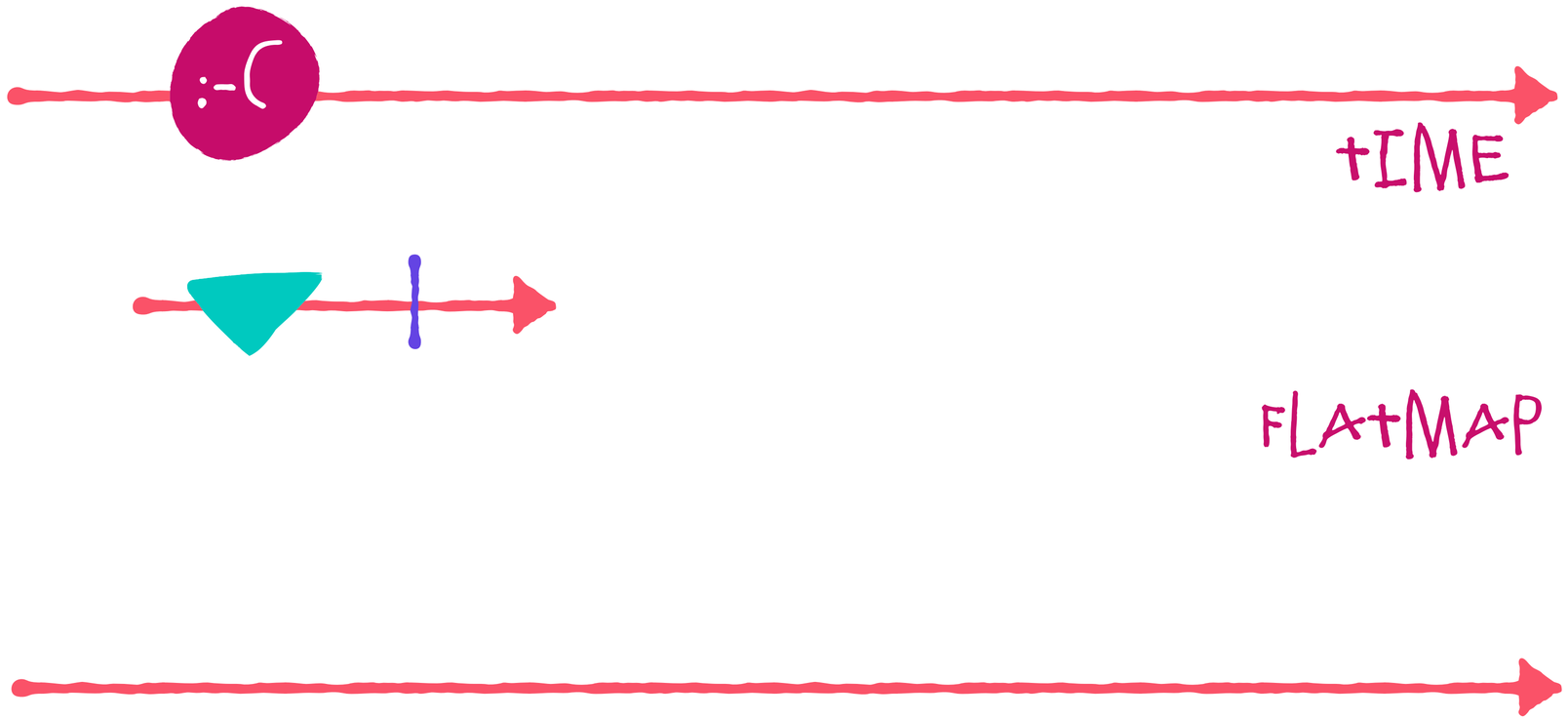
OPERATOR: FLATMAP()

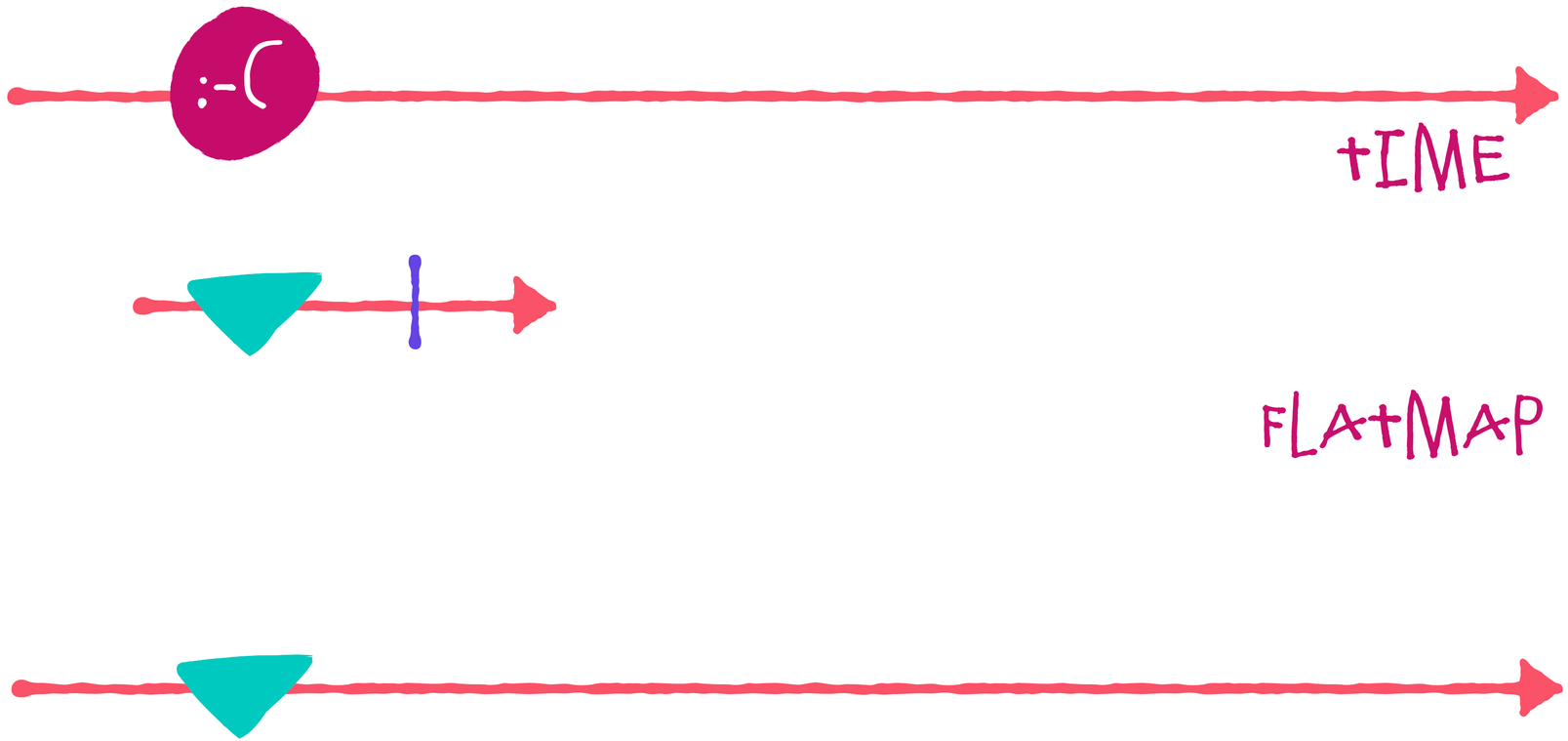
```
Observable.just( :-( , :-( )
        .flatMap( { Observable.just( 🔻 ) } )
        .subscribe { println(it) }
```

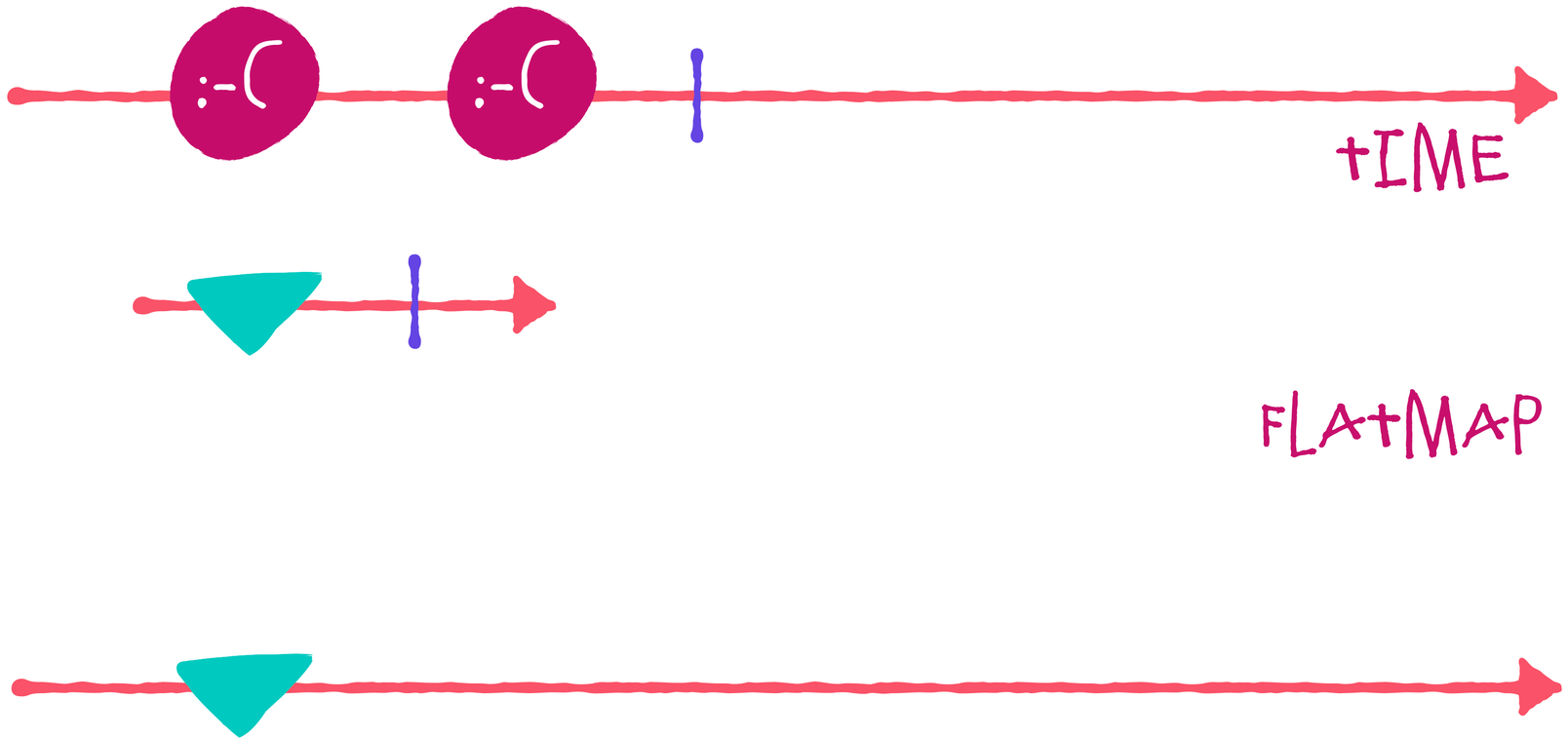

TIME
FLATMAP

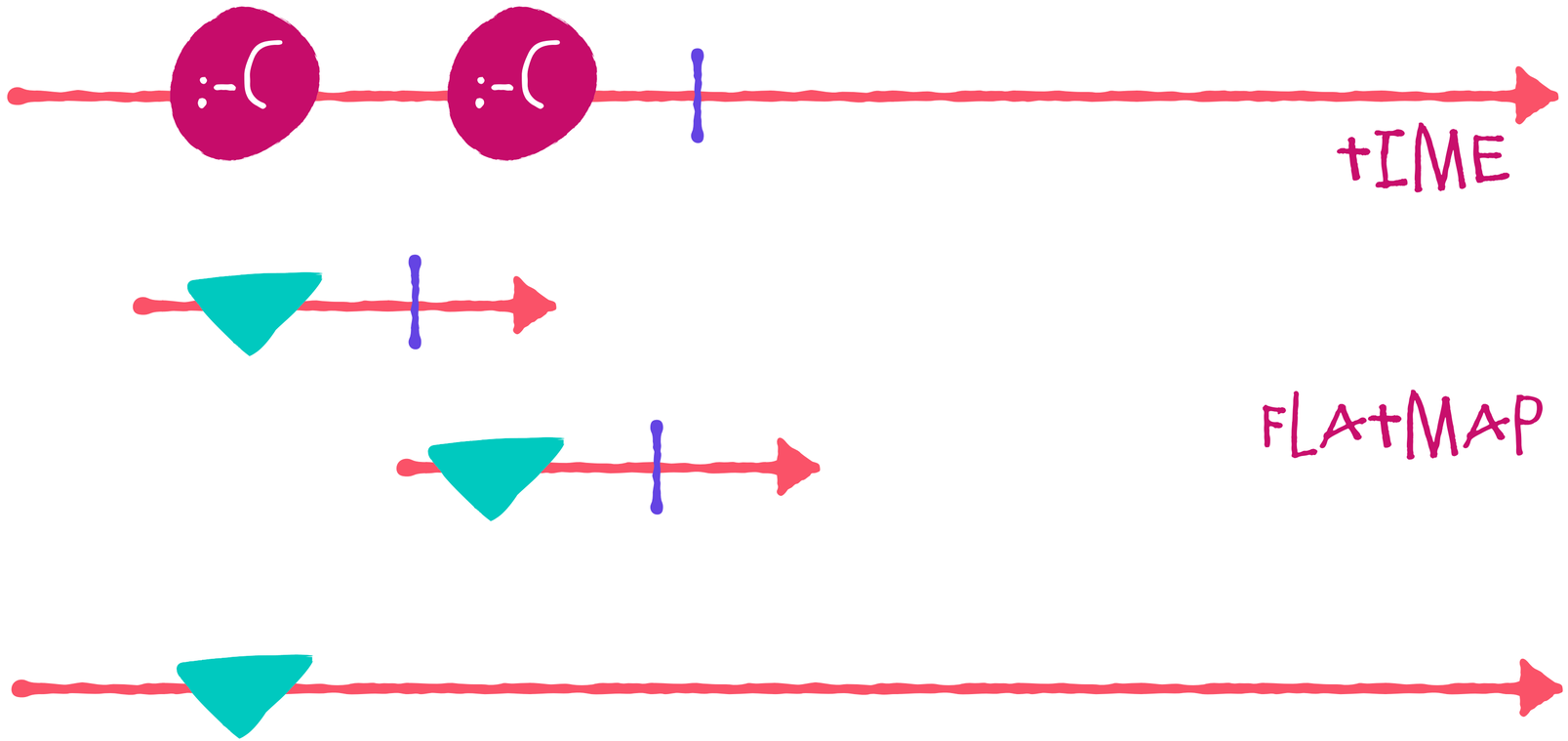


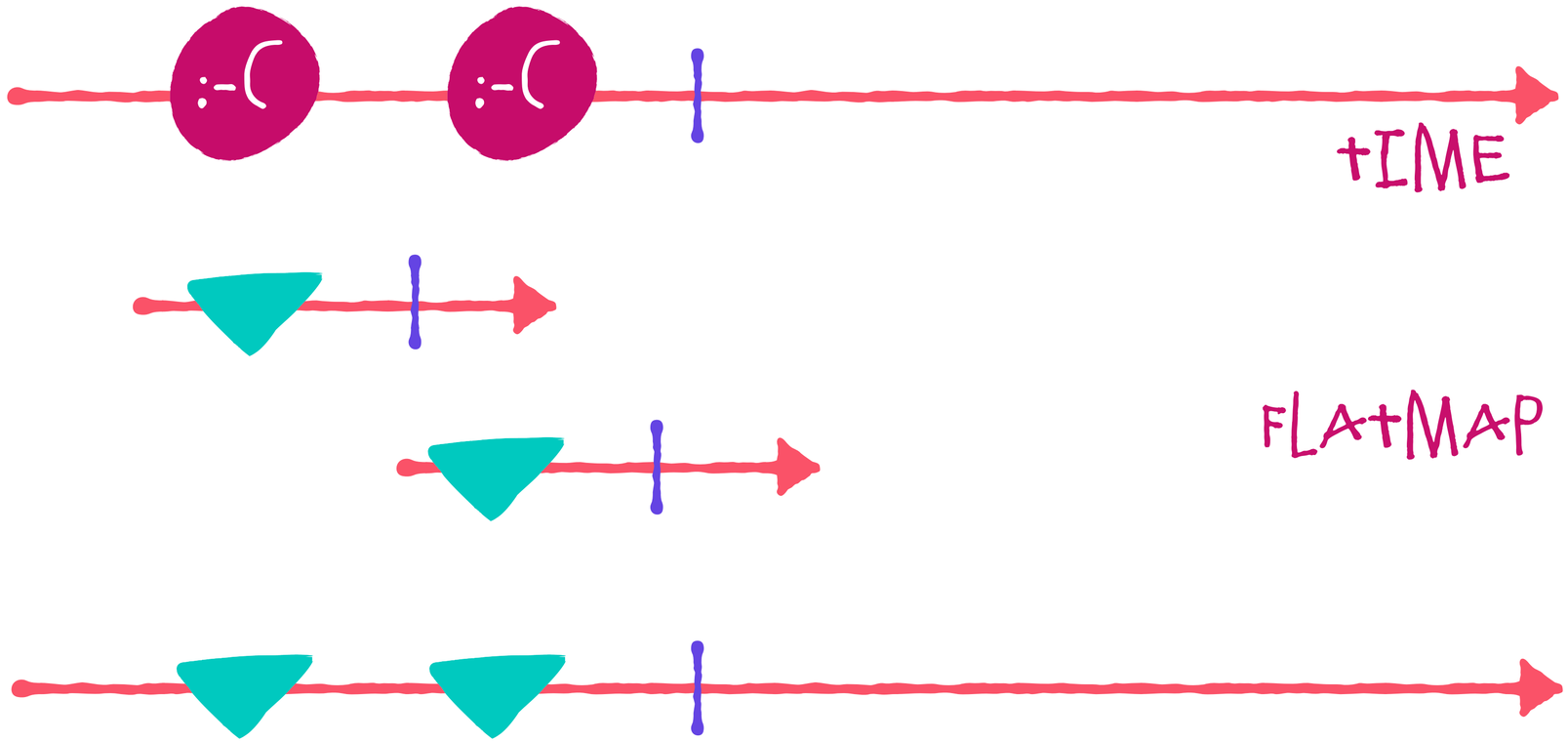
FLATMAP

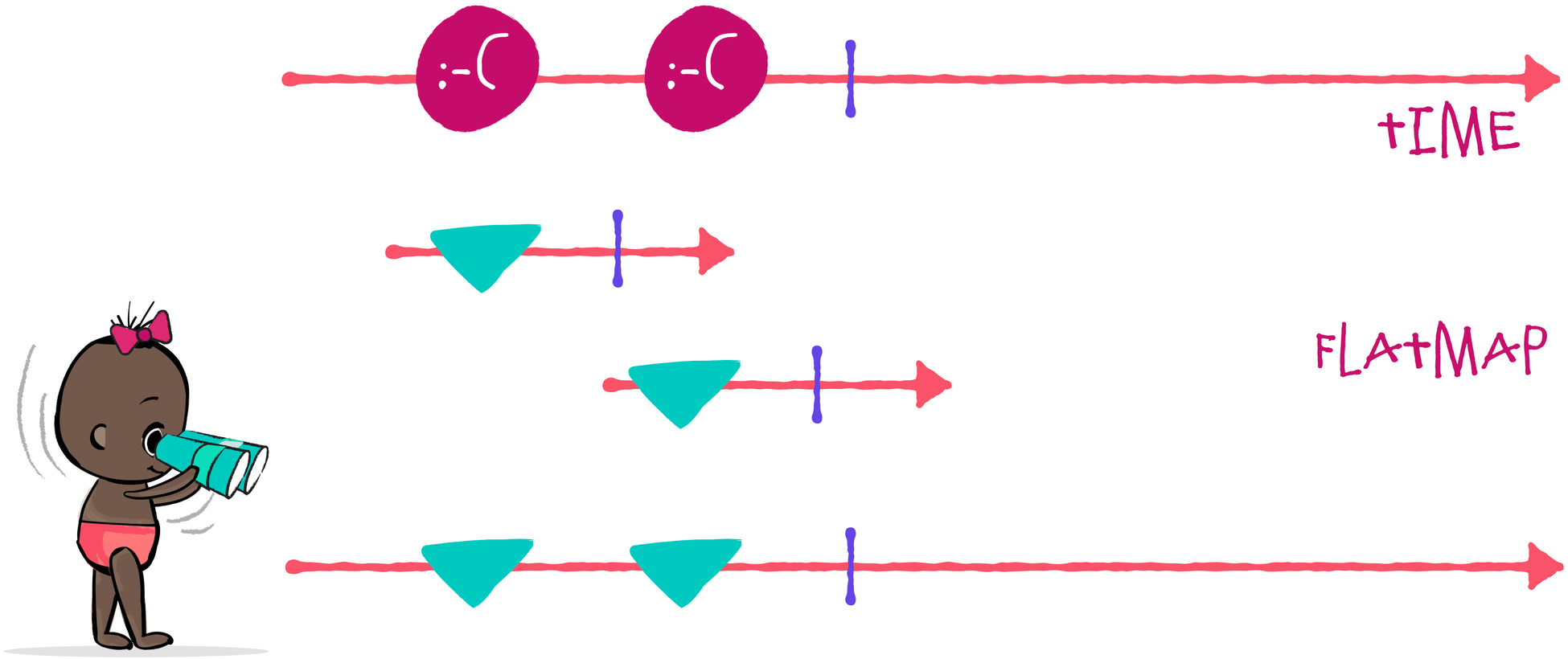

:-(
TIME
FLATMAP


:-(
TIME
FLATMAP


:-(
:-(
TIME
FLATMAP


:-(
:-(
TIME
FLATMAP


:-(
:-(
TIME
FLATMAP


:-(
:-(
TIME
FLATMAP

LONG RUNNING

ASYNCHRONOUS

OPERATOR: FLATMAP()

```
val users // Observable<User>
```

OPERATOR: FLATMAP()

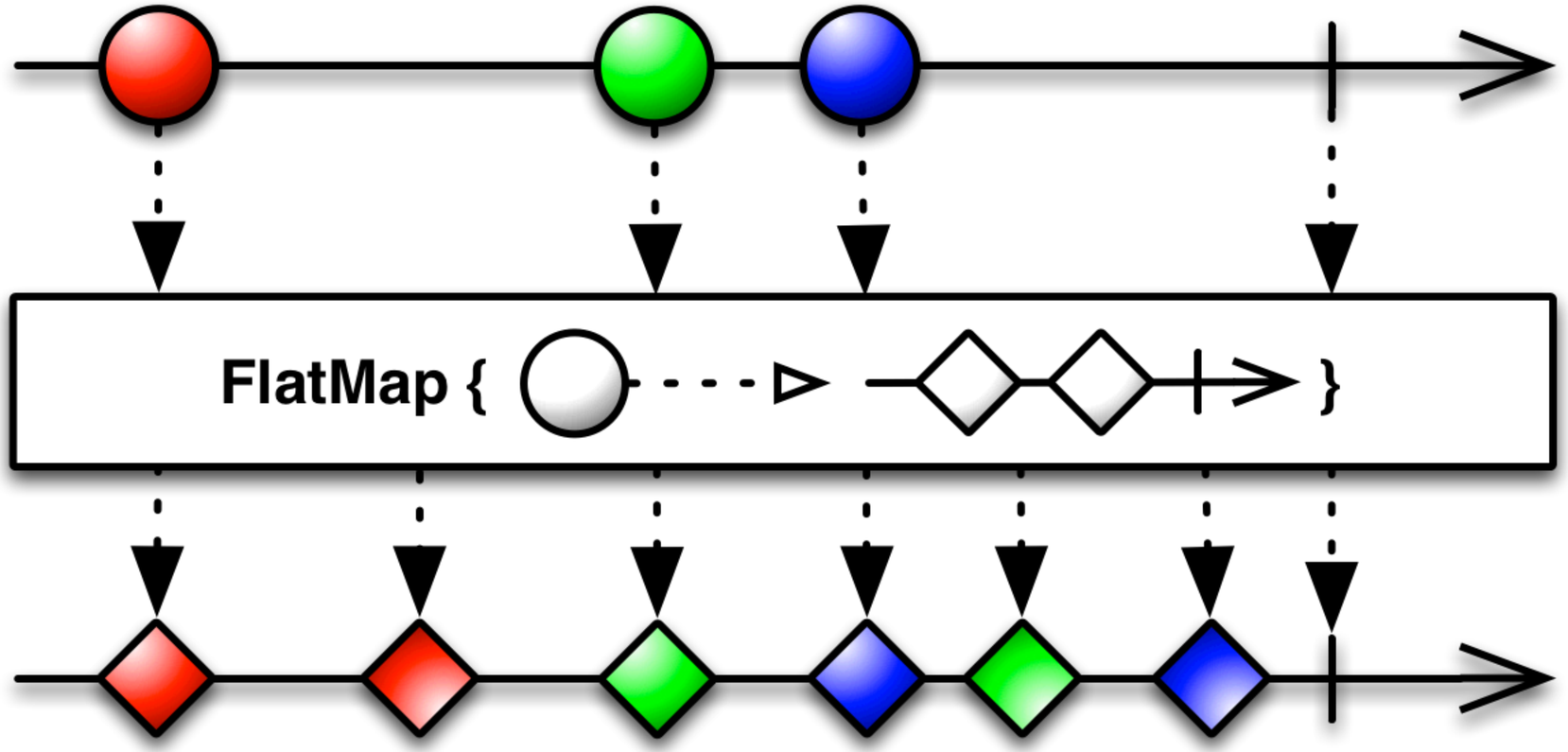
```
val users // Observable<User>
val posts: Observable<Post>
```

OPERATOR: FLATMAP()

```
val users // Observable<User>
val posts: Observable<Post>

posts = users.flatMap { getUsersPosts(it.id) }
```

OPERATOR: FLATMAP()

```
val users // Observable<User>
val posts: Observable<Post>

posts = users.flatMap { getUsersPosts(it.id) }
posts.subscribe { println(it) }
```



via reactivex.io

FLOWABLE

MAYBE

BACKPRESSURE

DISPOSABLE

COMPLETABLE

SINGLE

SHOULD YOU USE RXJAVA?

SHOULD YOU USE RXJAVA?

- LIKE FUNCTIONAL PROGRAMING?
- PROCESS ITEMS ASYNCHRONOUSLY?
- COMPOSE DATA?
- HANDLE ERRORS GRACEFULLY?

SHOULD YOU USE RXJAVA?

- LIKE FUNCTIONAL PROGRAMING?
- PROCESS ITEMS ASYNCHRONOUSLY?
- COMPOSE DATA?
- HANDLE ERRORS GRACEFULLY?

YES!

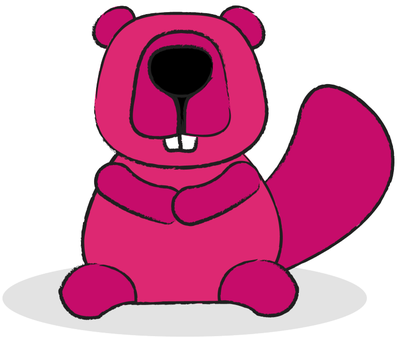
It
DEPENDS

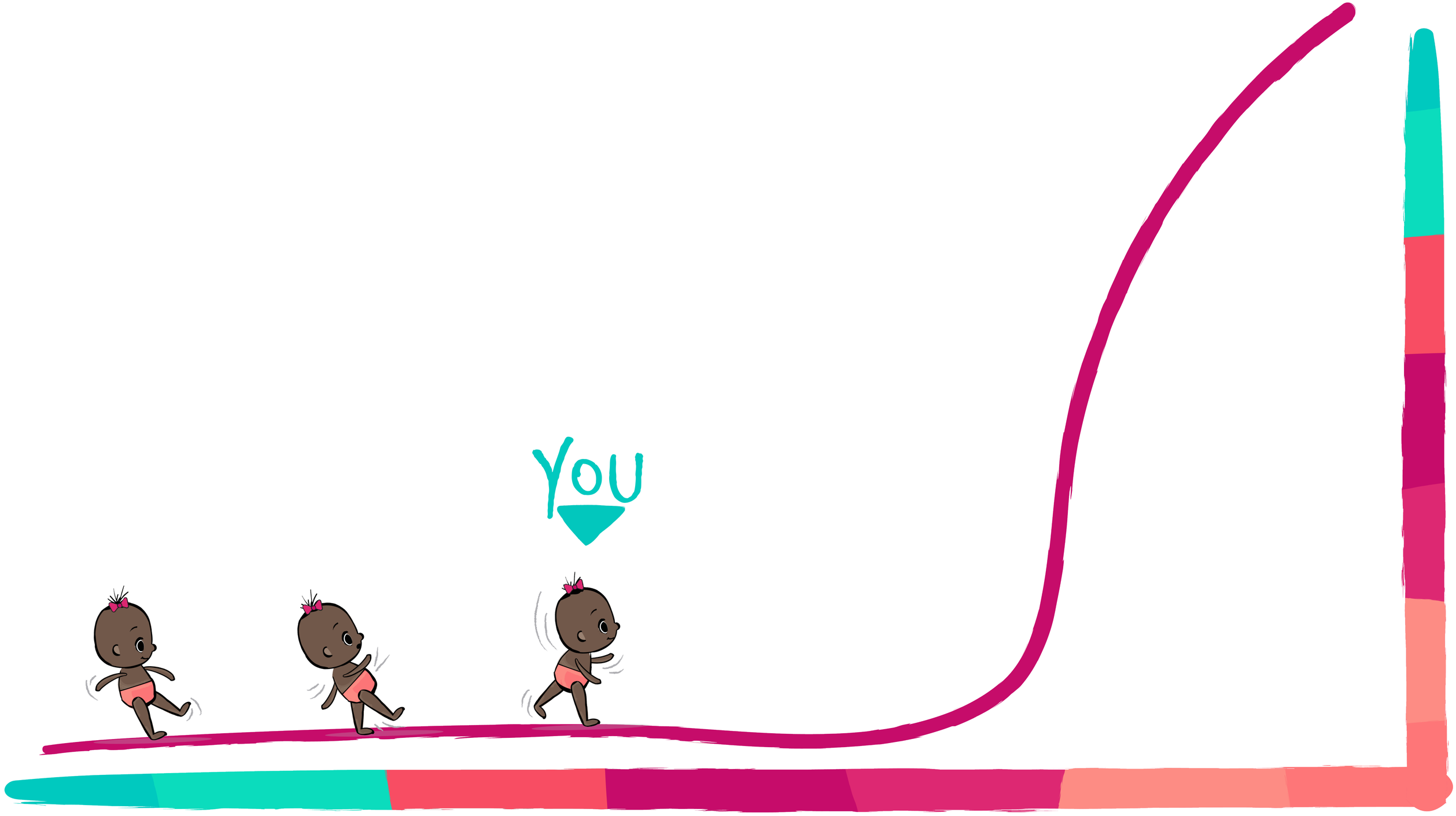
THE BASICS

OPERATORS

OBSERVER

OBSERVABLE




YOU

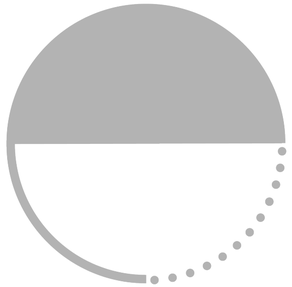
WWW.ADAVIS.INFO

@BRWNGRLDEV

RESOURCES

- Reactive Programming on Android with RxJava (http://amzn.to/2yOAkxn)
- Reactive Programming with RxJava (http://amzn.to/2zQtqb5)
- RxJava Playlist (https://goo.gl/9fw1Zv)
- Learning RxJava for Android Devs (https://goo.gl/VWxFLK)
- RxJava Video Course (https://caster.io/courses/rxjava)

SLIDE DESIGN:



@LAURAEMILYILLUSTRATION

FONT: ELLIOT 6, FONTSQUIRREL.COM