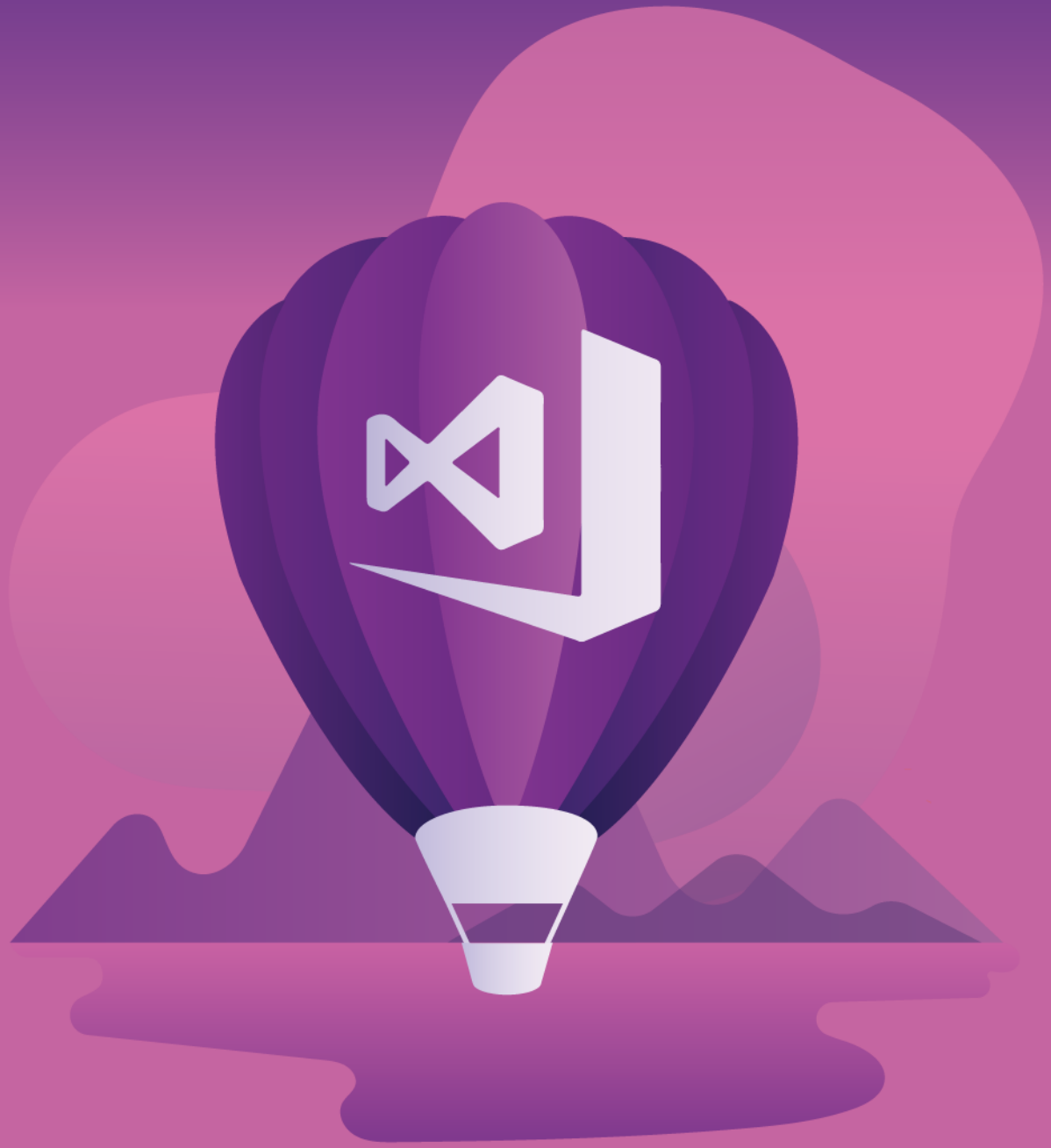


BIENVENIDOS
AL CURSO:

ASP NET CORE WEB APPLICATION: INTEGRACIÓN

SESIÓN 07

GALAXY
TRAINING

ÍNDICE

¿Qué es la programación reactiva?

¿Qué es la programación reactiva?

Reactivo significa actuar en respuesta a una situación en lugar de crearla o controlarla: reaccionar a ella. La programación reactiva es en realidad similar a esta definición, lo que significa escribir código que reacciona a los cambios.

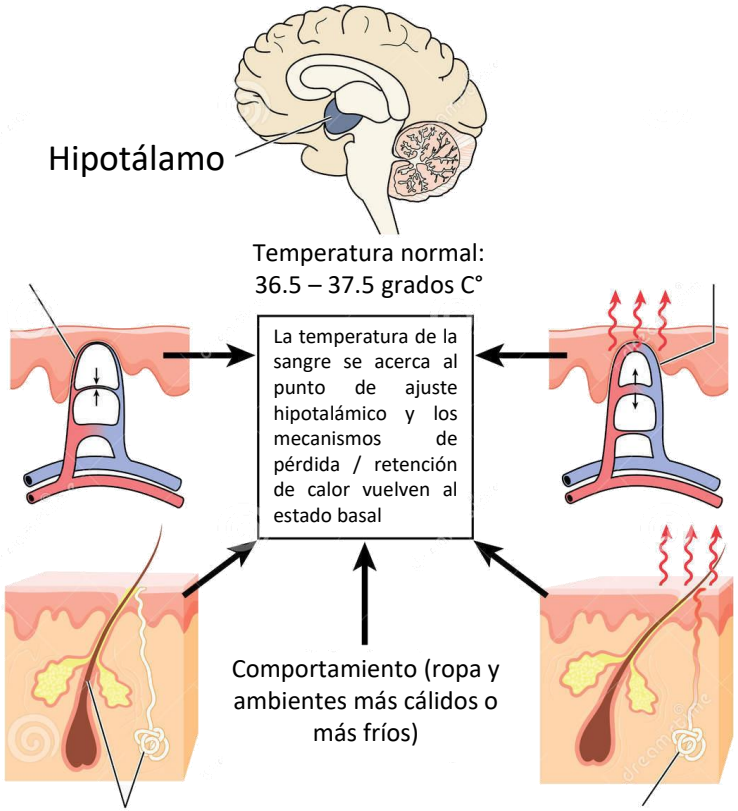
Es una forma de codificar con flujos de datos asincrónicos que nos facilitará el código de aplicaciones e interfaces que responden dinámicamente a los cambios en los datos.

Lo que significa que cada vez que tengamos cambios en los datos, nuestra aplicación responderá de forma reactiva a esos cambios.

¿Qué es la programación reactiva?

Periferias frías

Periferias calientes



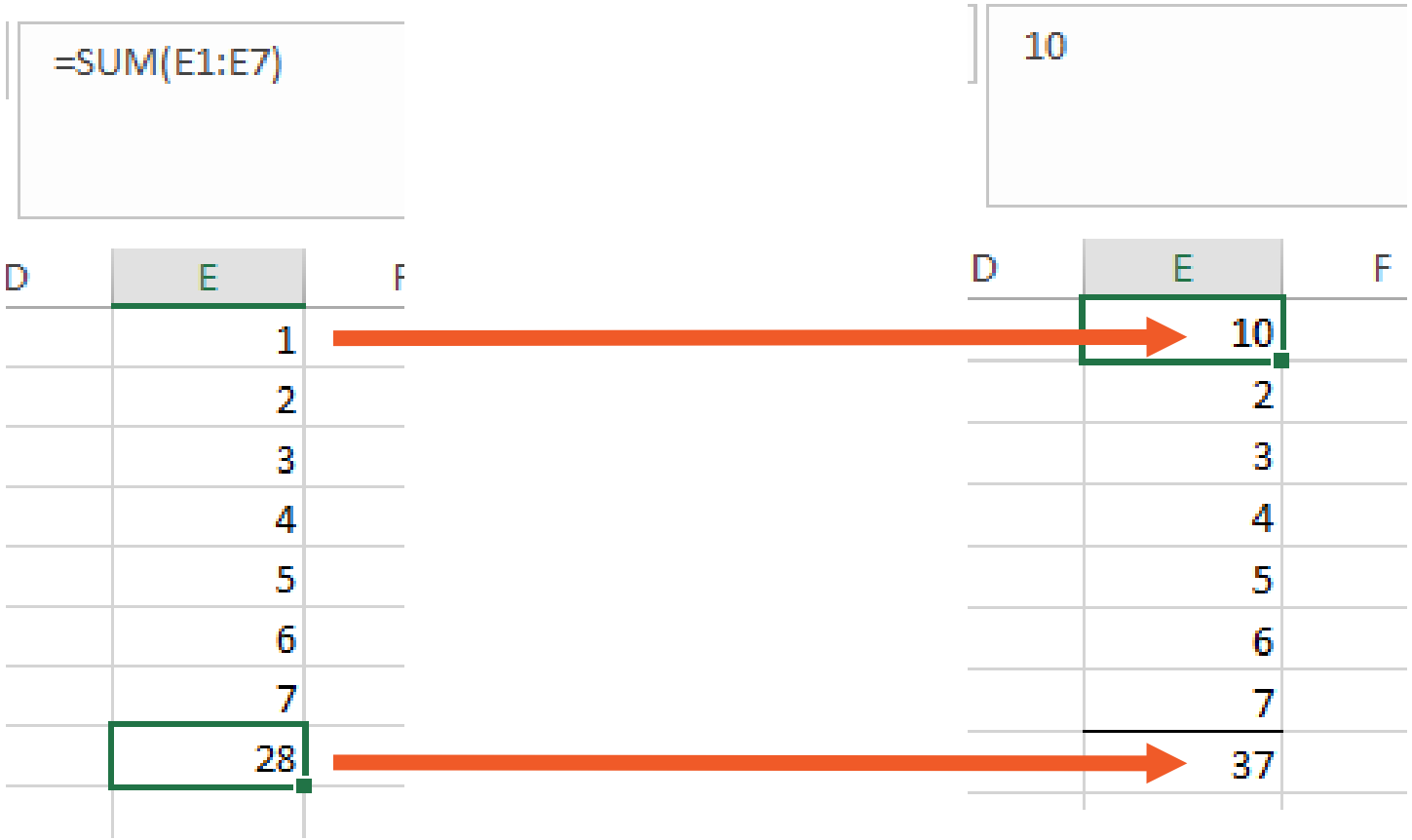
Los vasos capilares se constriñen cerca de la superficie de la piel para conservar el calor.

Los capilares se dilatan cerca de la superficie de la piel para perder calor al medio ambiente.

La secreción de sudor cesa y el vello se levanta, atrapando la capa aislante o el aire.

Las glándulas sudoríparas comienzan a secretar, causando pérdida de calor por evaporación.

¿Qué es la programación reactiva?



¿Qué es la programación reactiva?

Array of data

Stream of data

El hecho de pensar en **streams** en vez de valores aislados, nos permite pensar en una forma de programar en la cual podemos manipular secuencias enteras de valores que aún no han sido creados.

Para poder trabajar con programación reactiva, el punto importante aquí es imaginar el programa completo como un flujo de secuencias de datos

Reactive Extensions

¿Qué es la programación reactiva?

Array of data

Stream of data

Lo que se necesita es consultar una especie de base de datos de eventos, y obtener los 10 últimos.

Esto es lo que podemos realizar precisamente con la programación reactiva, podemos consultar los **streams de eventos**, filtrarlos, transformarlos, etc.

Iterator

Array of data

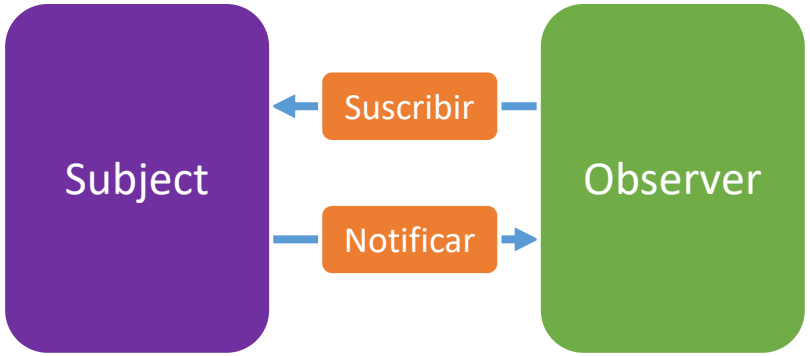
Stream of data

El **patrón iterator,** que nos provee de una forma de acceso a los elementos de un objeto agregado, de forma secuencial, sin exponer su representación interna.

El patrón de iterador tiene 2 métodos:

- **next():**devuelve el siguiente valor de la posición actual.
- **hasNext():** devuelve true si el elemento actual es el último

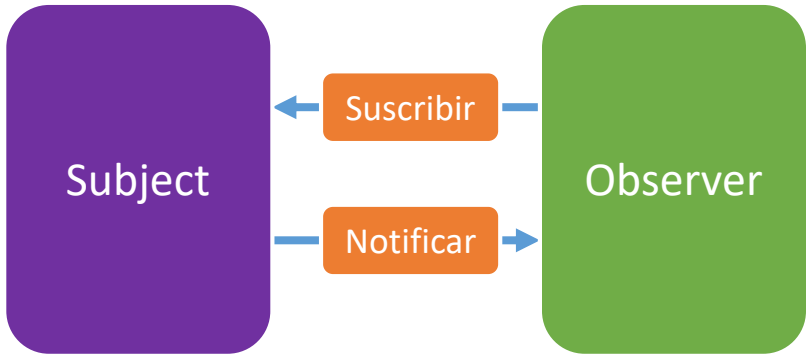
Observable



Lo que necesitamos para poder tratar los streams de eventos como “**bases de datos**” que pueden ser consultadas, es un objeto que abstraiga el concepto de stream, el cual es llamado : Observable

Comencemos por el observable que es un objeto en el que un observador se suscribe a él, de modo que este último reacciona a cualquier elemento o secuencia de elementos emitidos por el observable.

Observer

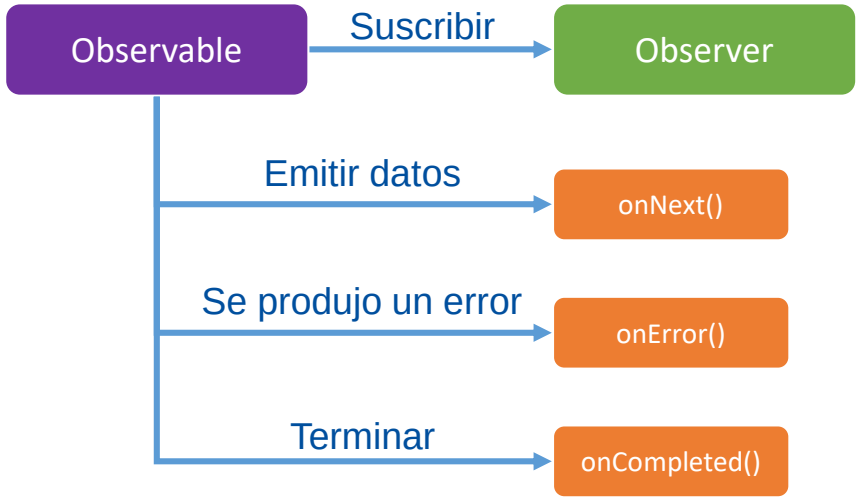


El otro concepto base es el **patrón observador**, también conocido como **publicación/subscripción**.

Un objeto observable puede ser vigilado por objetos observadores. De esta forma, un observable almacena referencias sus observadores y tendrá la capacidad de notificarles cambios.

Demo : Observable y Observer

Observer



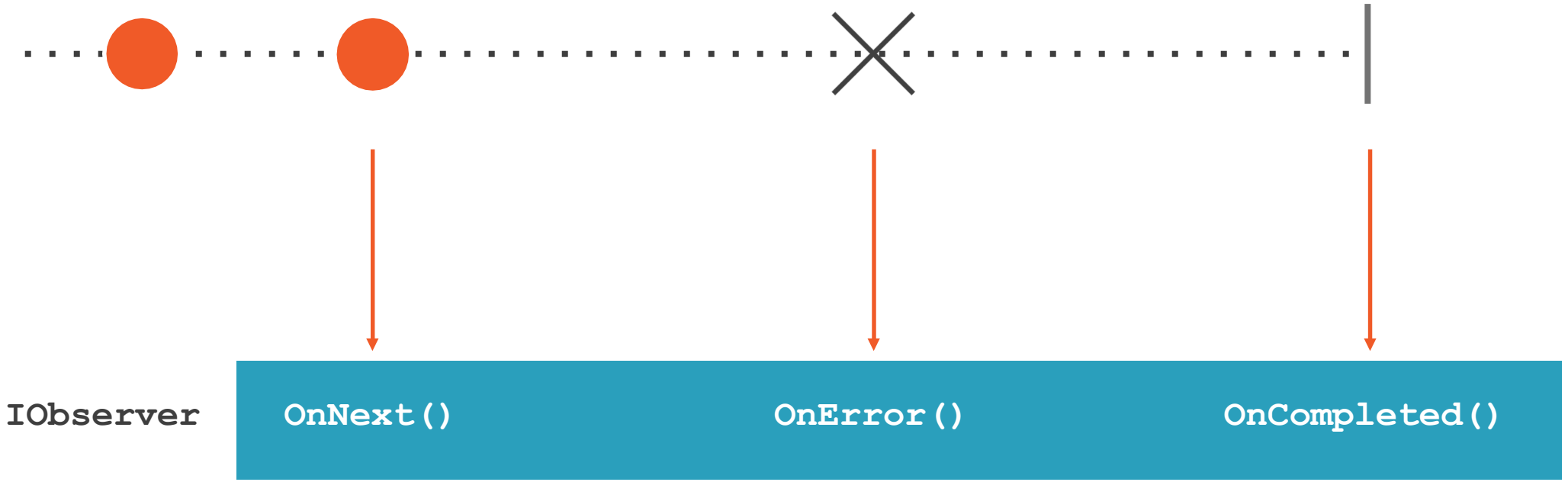
Un observador también llamado suscriptor está conectado al observable a través del método de suscripción. Este método proporciona tres subconjuntos:

• **onNext(value):**siempre que un observable emita un nuevo valor.

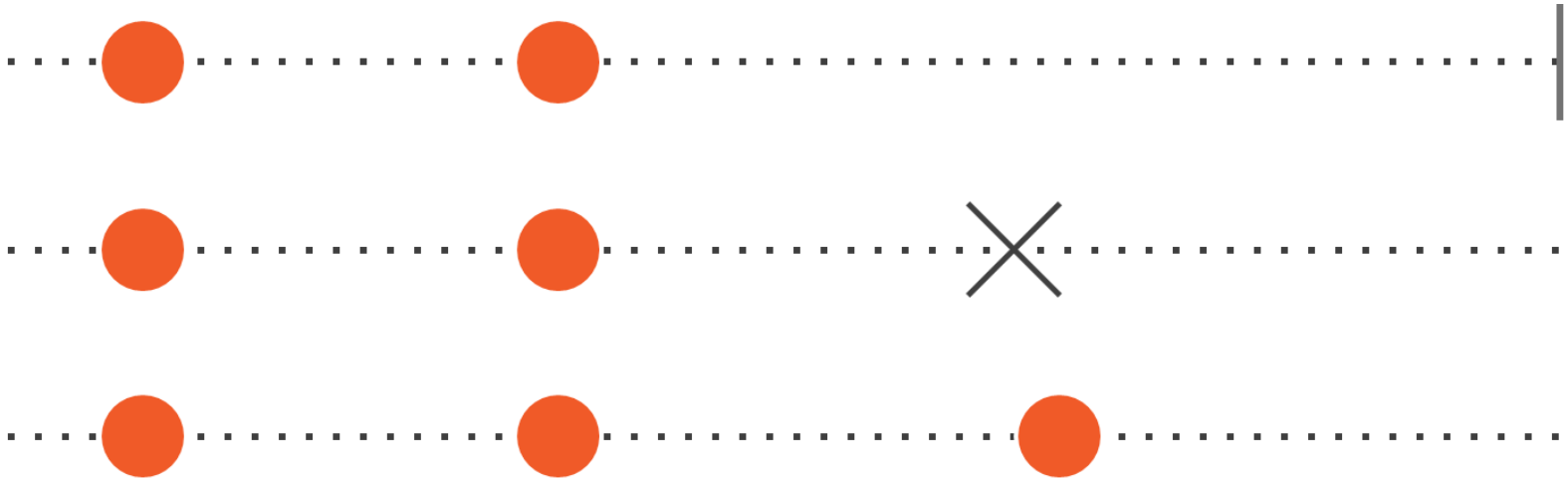
• **onError(error):**siempre que se produzca un error o el observable no emita el valor o evento esperado. Termina la secuencia con un error.

• **onCompleted():** cuando se llama al método onNext para la hora final y no se ha producido ningún error. Termina la secuencia con éxito.

Observer

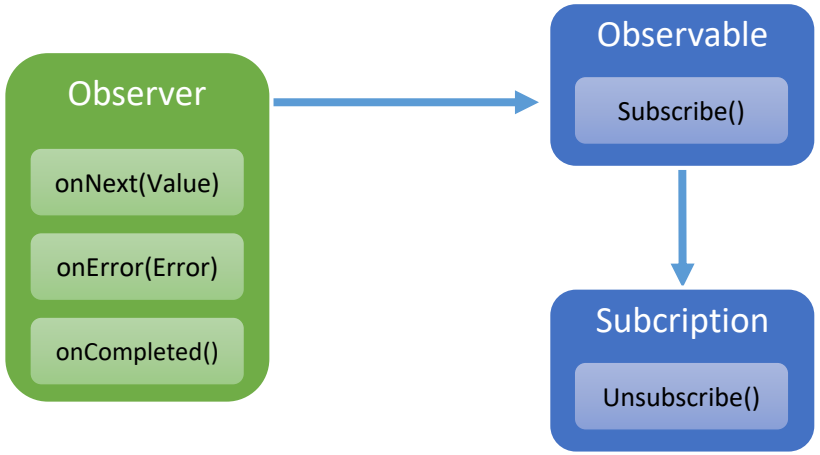


Observer



OnNext()	OnNext()	OnError()	OnCompleted()

Subscripción



El Observable implementa 2 métodos principales:

· subscribir

· anular la suscripción

Subject

Un sujeto es un tipo especial de reactivo que puede ser un observable y un observador. Puede emitir valores o eventos, aceptar suscripciones y agregar nuevos elementos a la secuencia.

Hay 4 tipos de temas, voy a proporcionar un ejemplo a cada tipo con el fin de aclarar el comportamiento de esos temas.

Subject

AsyncSubject

Emite el último valor, emitido por la fuente observable después de que se completa.

BehaviorSubject

Al suscribirse, comienza emitiendo el elemento emitido más recientemente por la fuente Observable.

PublishSubject

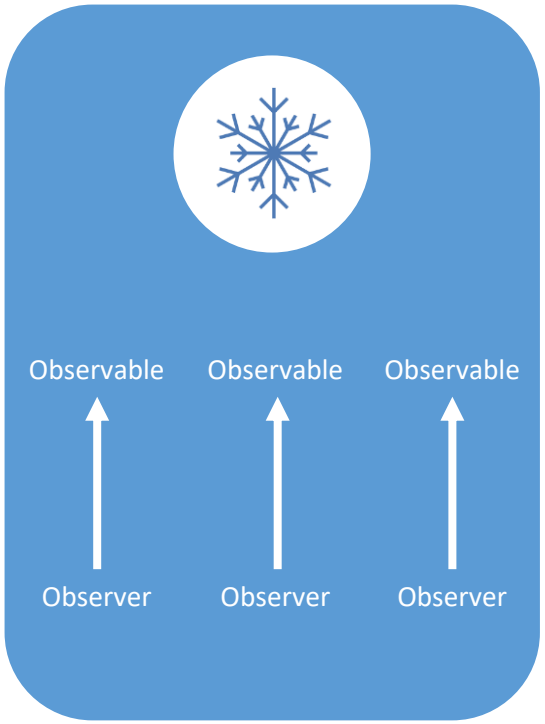
Emite solo los valores emitidos después de la suscripción

ReplaySubject

Emite todos los valores emitidos por el observable, independientement e de cuándo se realizó la suscripción

Demo : Subject

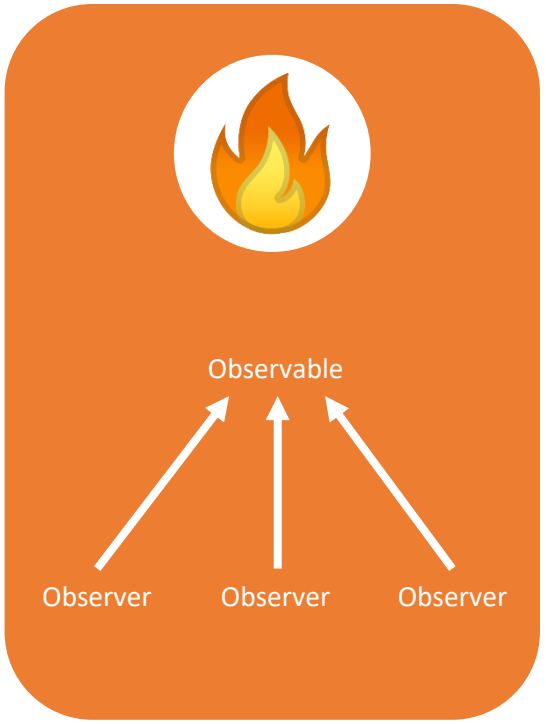
Cold Observable



Cold Observable son aquellos a los que se crean datos dentro de ellos, y comienzan a emitir valores solo cuando un observador se suscribe a ese tipo de observables, lo que significa que emite valores solo cuando se solicita.

Algunos ejemplos de casos de uso: consultas de bases de datos, servicios web, lectura o descarga de archivos...

Hot Observable

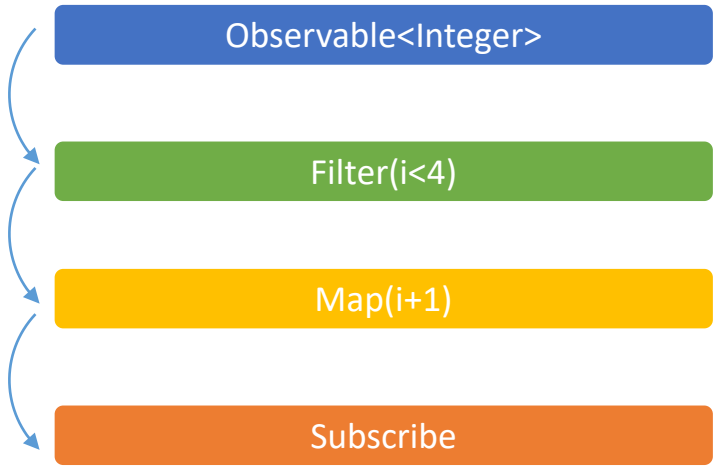


Hot Observables son aquellos que comienzan a emitir valores independientemente de si el observador está listo o no.

Lo que significa que estos observables comienzan a emitir valores antes de que se haga la suscripción no espera a que los observadores se suscriban (Sin control sobre la tasa de emisión).

Los datos se producen fuera del observable, por ejemplo: eventos de ratón y teclado, eventos de interfaz de usuario, eventos del sistema, tiempo...

Operadores



Los operadores son las herramientas que hacen que observables sea potente, lo que nos permite codificar soluciones elegantes y declarativas para tareas asíncronas complejas. También nos permiten encadenar operaciones, cada operación modifica el observable emitido por el operador anterior.

Estos operadores operan en un observable y devuelven un observable. Lo que nos permitirá encadenarlos uno tras otro.

Estas operaciones pueden transformar, combinar o filtrar los elementos emitidos por una secuencia observable, antes de que el suscriptor los reciba.

Operadores

Creando

- Create
- From
- Interval
- Just

Tranformando

- Map
- flatMap
- Scan

Filtrando

- Debounce
- Filter
- Skip
- Take

Combinando

- Merge
- Join
- Switch
- Zip

Error

- Catch
- Retry

Utilitario

- Delay
- Do
- Subscribe
- subscribeOn
- ObserveOn

ReactiveX (Reactive Extensions)

ReactiveX

ReactiveX o Reactive eXtension o simplemente Rx es una biblioteca para la programación asincrónica y basada en eventos mediante secuencias de secuencias de datos.

Asincrónico significa ejecutar varios bloques de código simultáneamente, cada bloque de código se ejecutará en su propio subproceso.

Se basa en eventos significa ejecutar código basado en los eventos generados mientras se ejecuta el programa (por ejemplo: Button clics eventos).

http://reactivex.io/

Reactive Extensions (ReactiveX)

Reactive Extensions (Rx) es una biblioteca para componer programas asincrónicos y basados en eventos mediante secuencias observables y operadores de consulta de estilo LINQ.

Las secuencias de datos pueden adoptar muchas formas, como una secuencia de datos de un archivo o servicio web, solicitudes de servicios web, notificaciones del sistema o una serie de eventos, como la entrada del usuario.

Extensiones reactivas representa todas estas secuencias de datos como secuencias observables. Una aplicación puede suscribirse a estas secuencias observables para recibir notificaciones asincrónicas a medida que llegan nuevos datos.

ReactiveX - Observables

```
Observable.Empty;
```

```
Observable.Never;
```

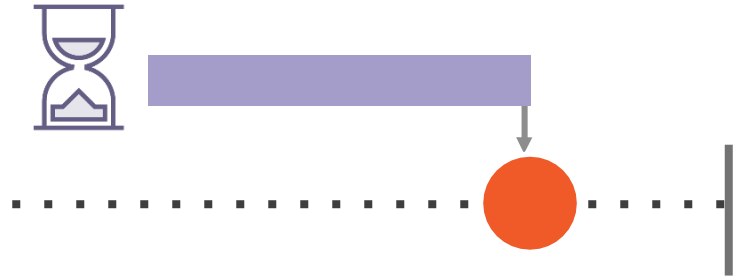
```
Observable.Throw(exception);
```

```
Observable.Return("42");
```

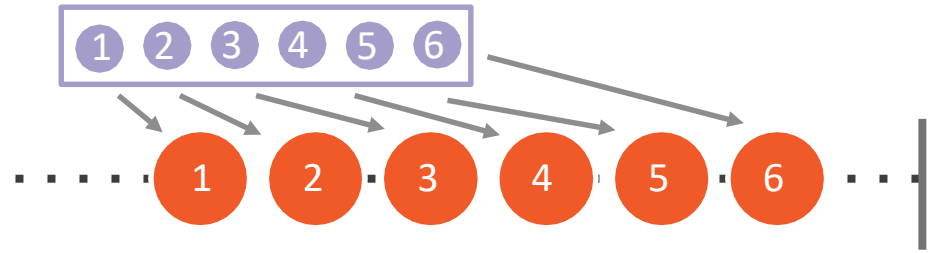
ReactiveX - Observables

```
Observable.Start(() => 42);
```

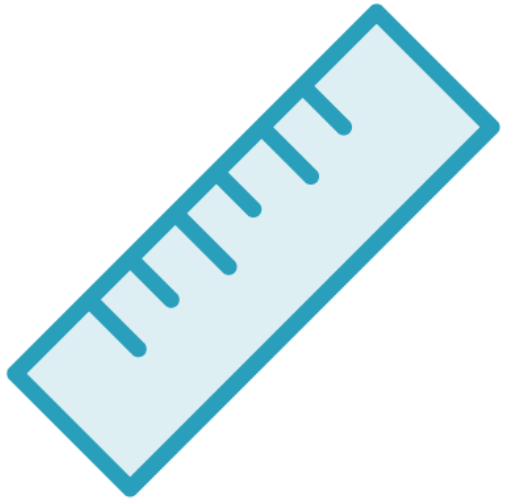
```
Task.ToObservable();
```



```
IEnumerable numbers = ...
numbers.ToObservable();
```



ReactiveX - Creadores



Range

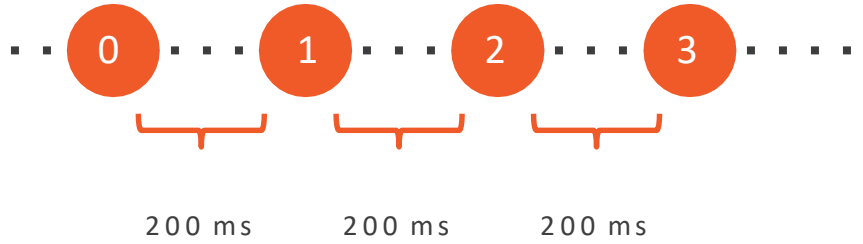
Interval, Timer

Create, Generate

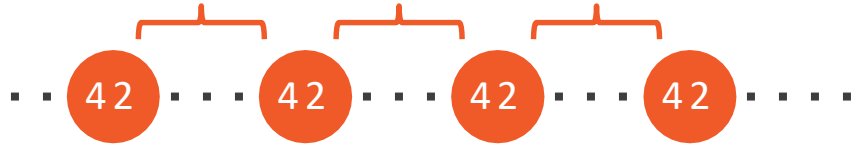
ReactiveX - Creadores

```
Observable.Range(1,4);
```

```
// timespan = 200ms
Observable.Interval(timespan);
```

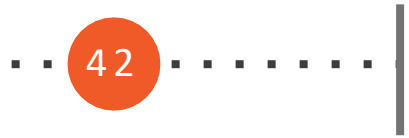


```
Observable.Timer(timespan, ()=>  42);
```



ReactiveX - Creadores

```
Observable.Create<int>(o =>
{
  o.OnNext(42);
  o.OnCompleted();
  return Disposable.Empty;
});
```




```
Observable.Generate(1,
value => value < 5,
value => value + 1,
value => value);
```

ReactiveX - Operadores

Filtrar elementos por condición

- Where
- OfType<T>

Filtrar elementos duplicados

- Distinct
- DistinctUntilChanged

Filter Head or Tail Elements

- Skip / Take
- SkipLast / TakeLast

Filtrar elementos por condición

- Where
- OfType<T>

Filtrar elementos duplicados

- Distinct
- DistinctUntilChanged

Filter Head or Tail Elements

- Skip / Take
- SkipLast / TakeLast

Secuencia

- SkipUntil / TakeUntil

Combinando secuencias del mismo tipo

- Merge
- Amb
- Concat
- Catch
- OnErrorResumeNext

Secuencias de emparejamiento

- CombineLatest
- Zip

Secuencias de secuencias

- Merging Sequences of Sequences
- Switch

Operadores de agregación

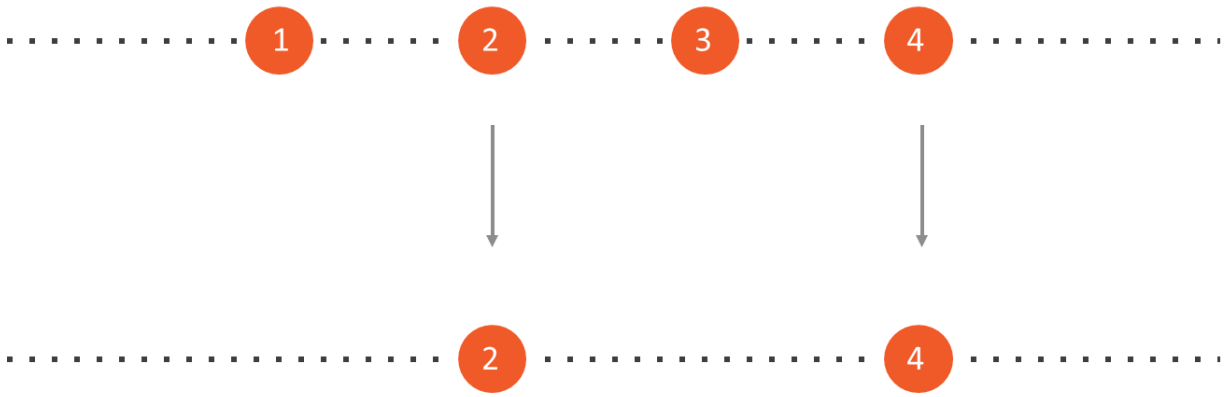
- Count / Sum
- Aggregate
- Scan

Utilitarios

- Do
- Timestamp
- Throttle

ReactiveX - Operadores

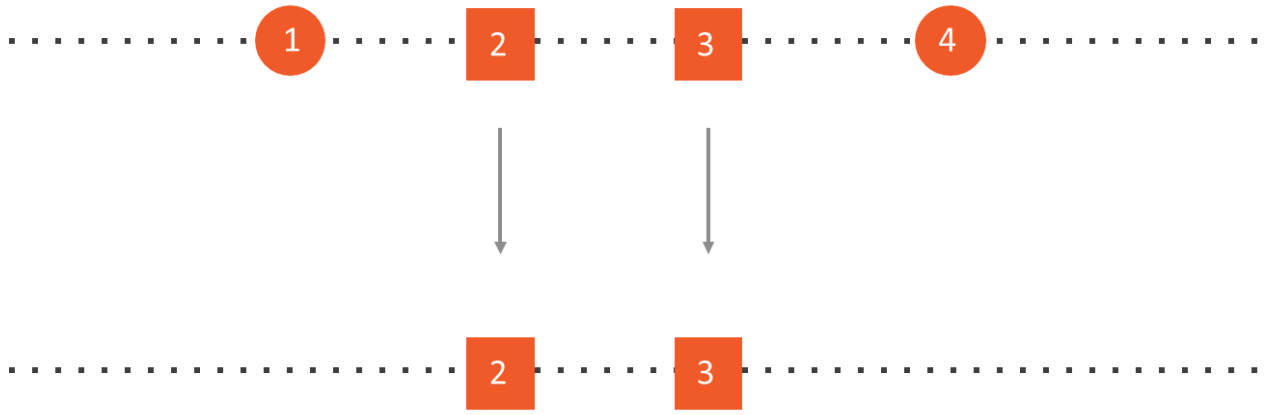
Where




```
.Where( x => x % 2 == 0);
```

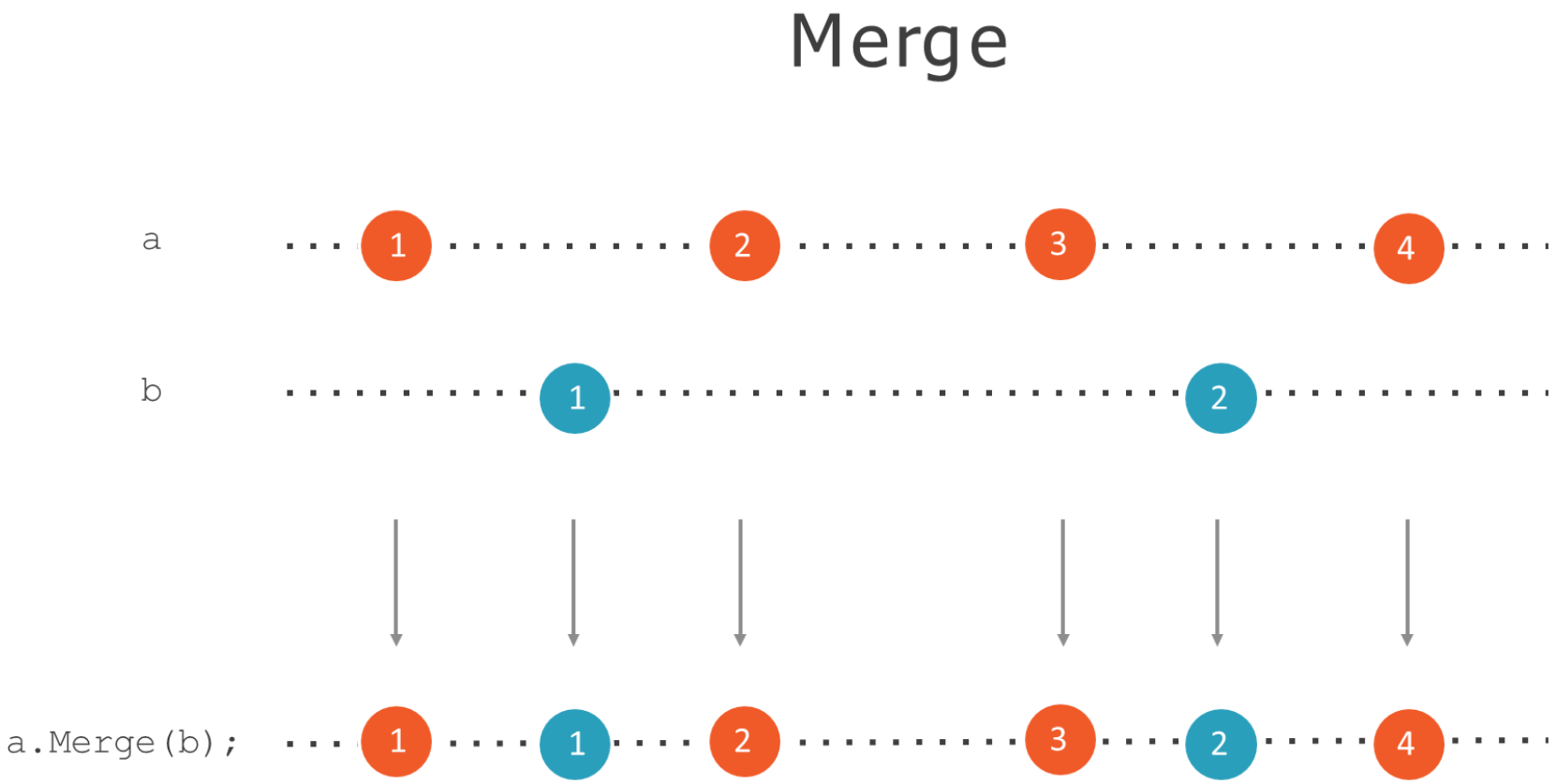
ReactiveX - Operadores

OfType<T>



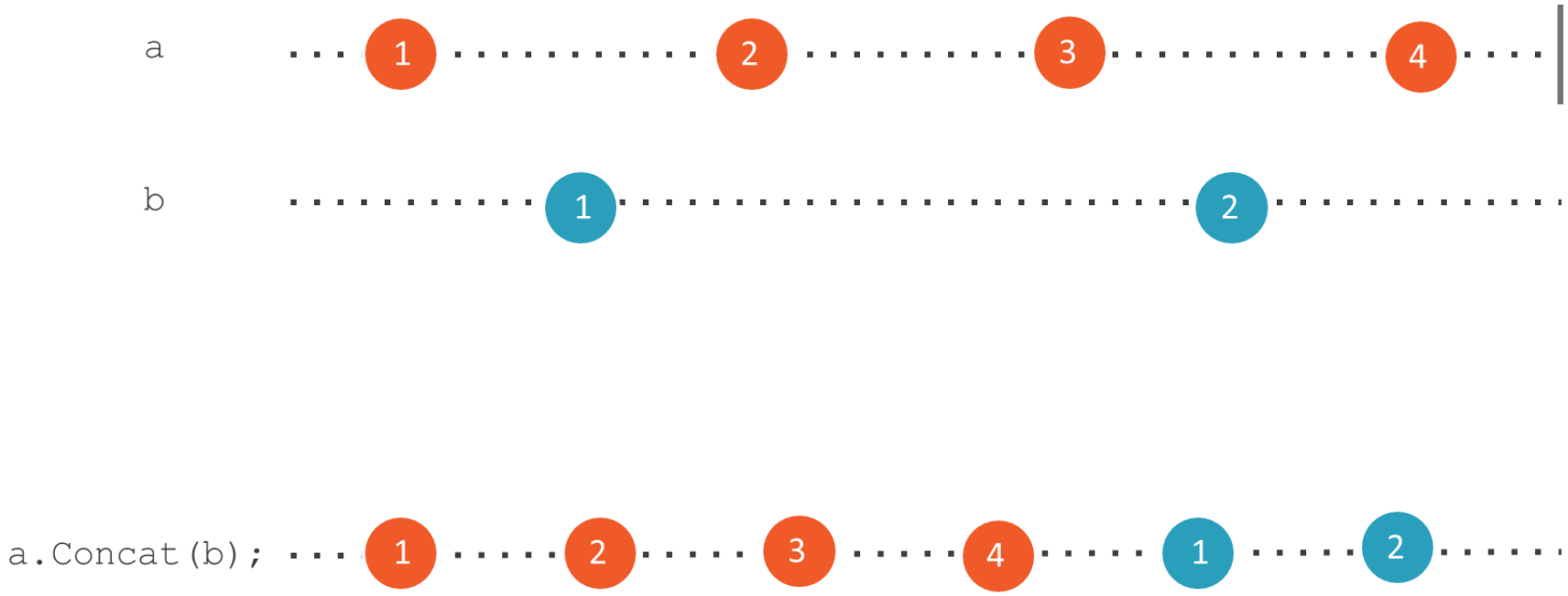

```
.OfType<Square>();
```

ReactiveX - Operadores

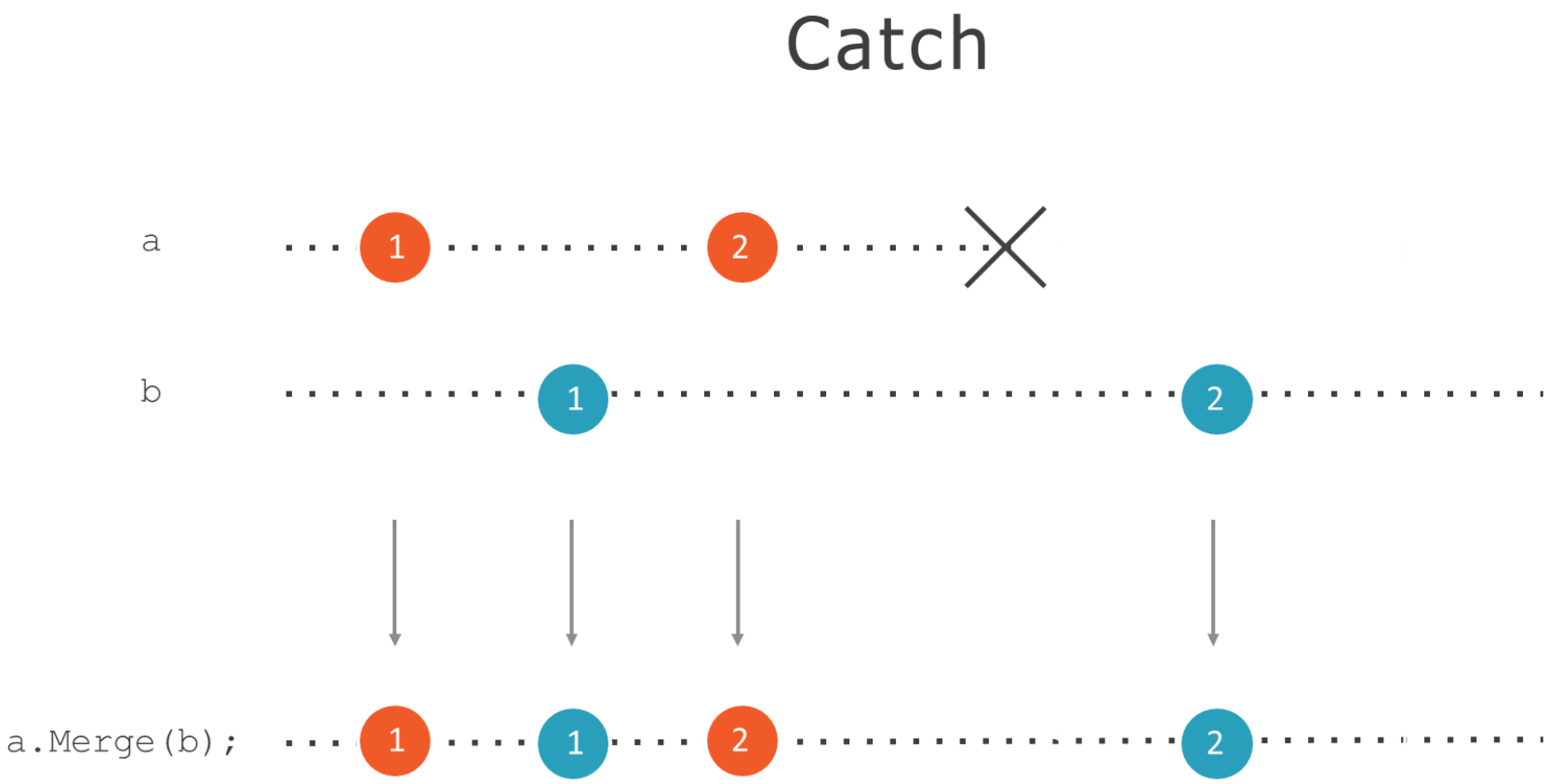


ReactiveX - Operadores

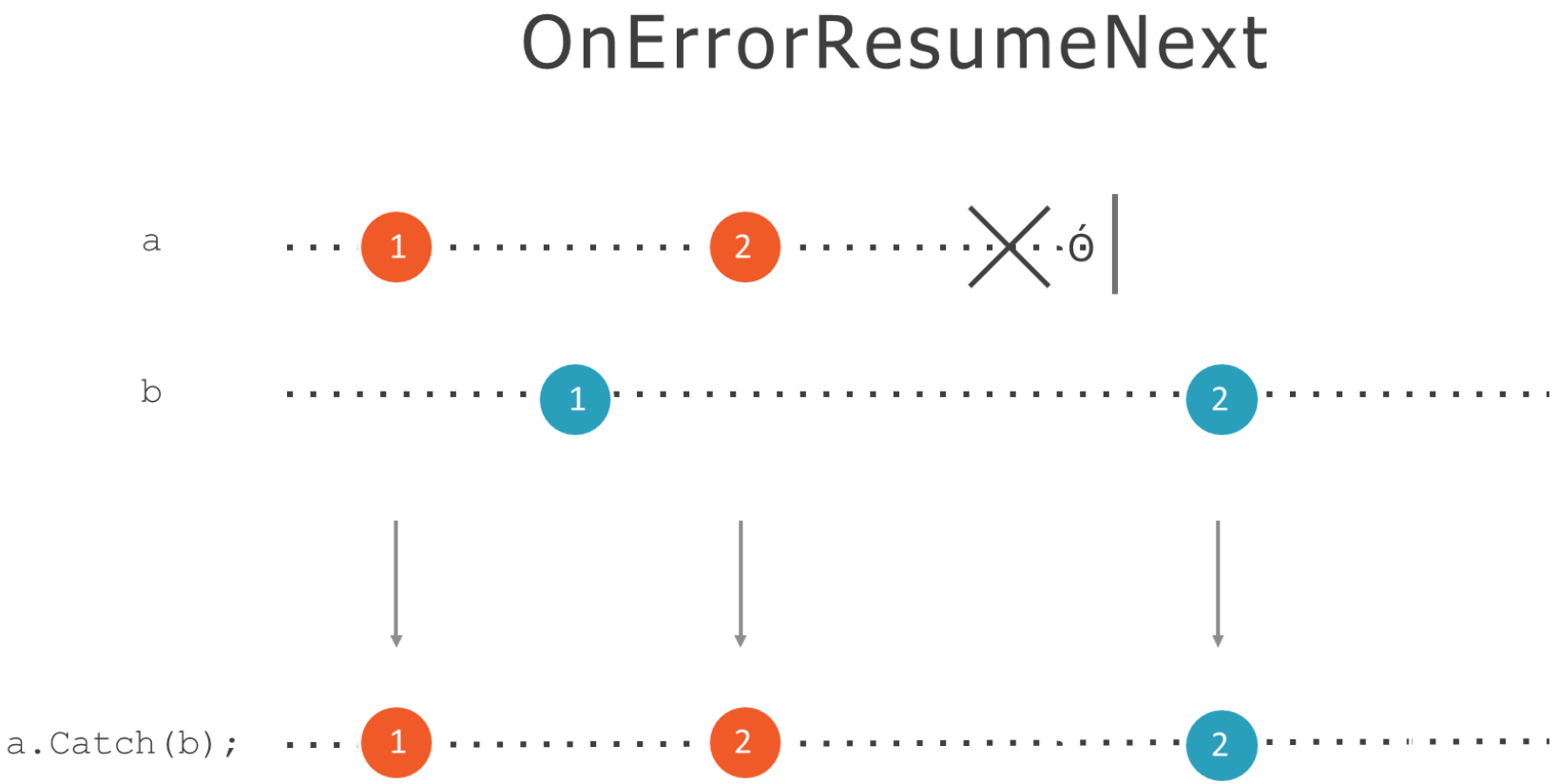
Concat



ReactiveX - Operadores



ReactiveX - Operadores



Demo : Operadores

GRACIAS

POR SU PREFERENCIA

GALAXY
TRAINING