



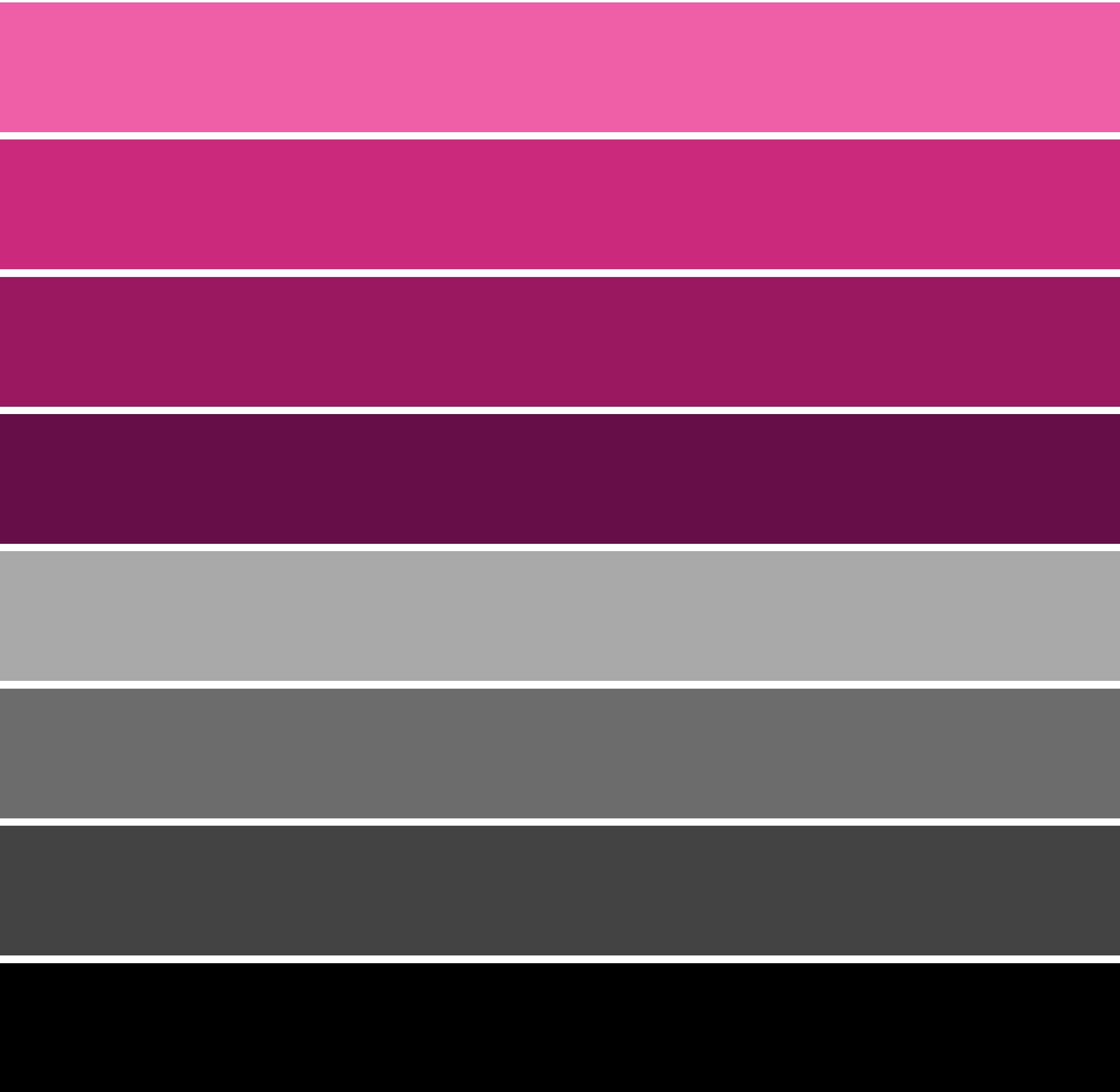
Chi ha paura degli Observable?

Michele Stieven



Michele Stieven

Front-end Developer & Consultant



Anatomia di un Observable

120+ Operatori

Higher Order Observables

Hot vs Cold

Subjects

Multicasting

Unsubscribing

Schedulers

HOT
VS
COLD

Definizione ufficiale:

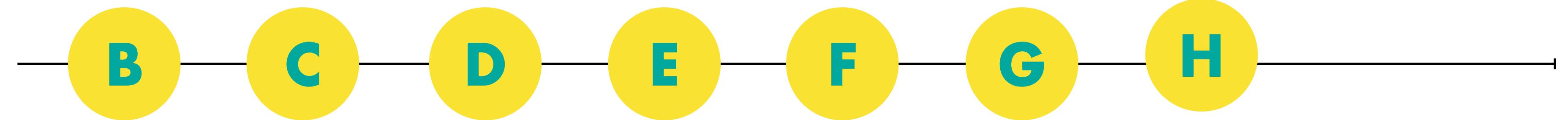
Un Observable è cold se inizia a produrre valori quando avviene il subscribe.

È hot se produce valori indipendentemente dalle sottoscrizioni.

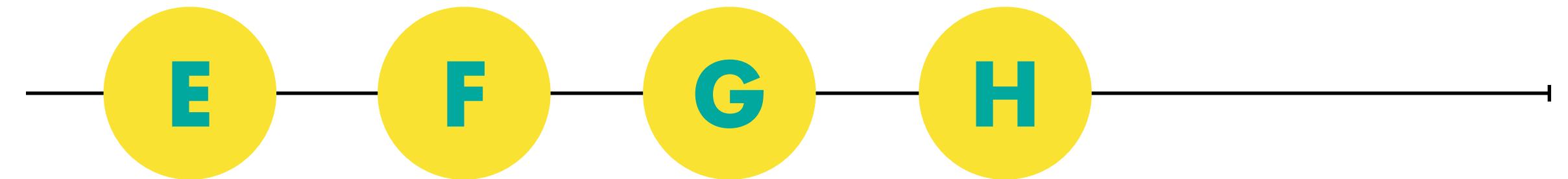
Hot



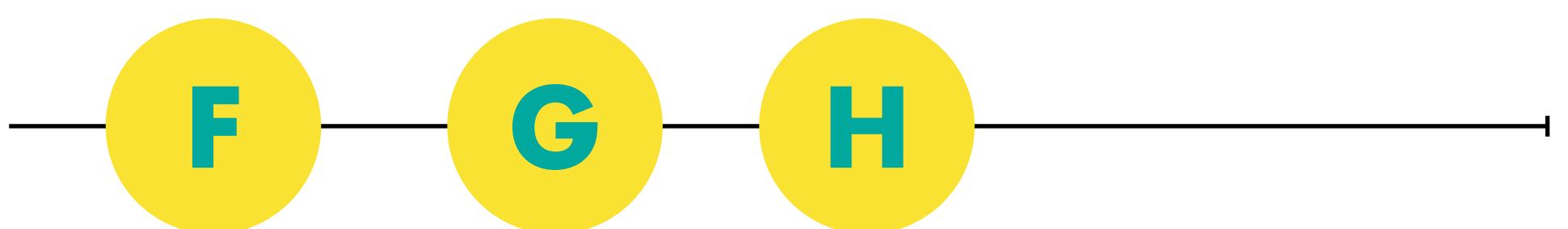
S1



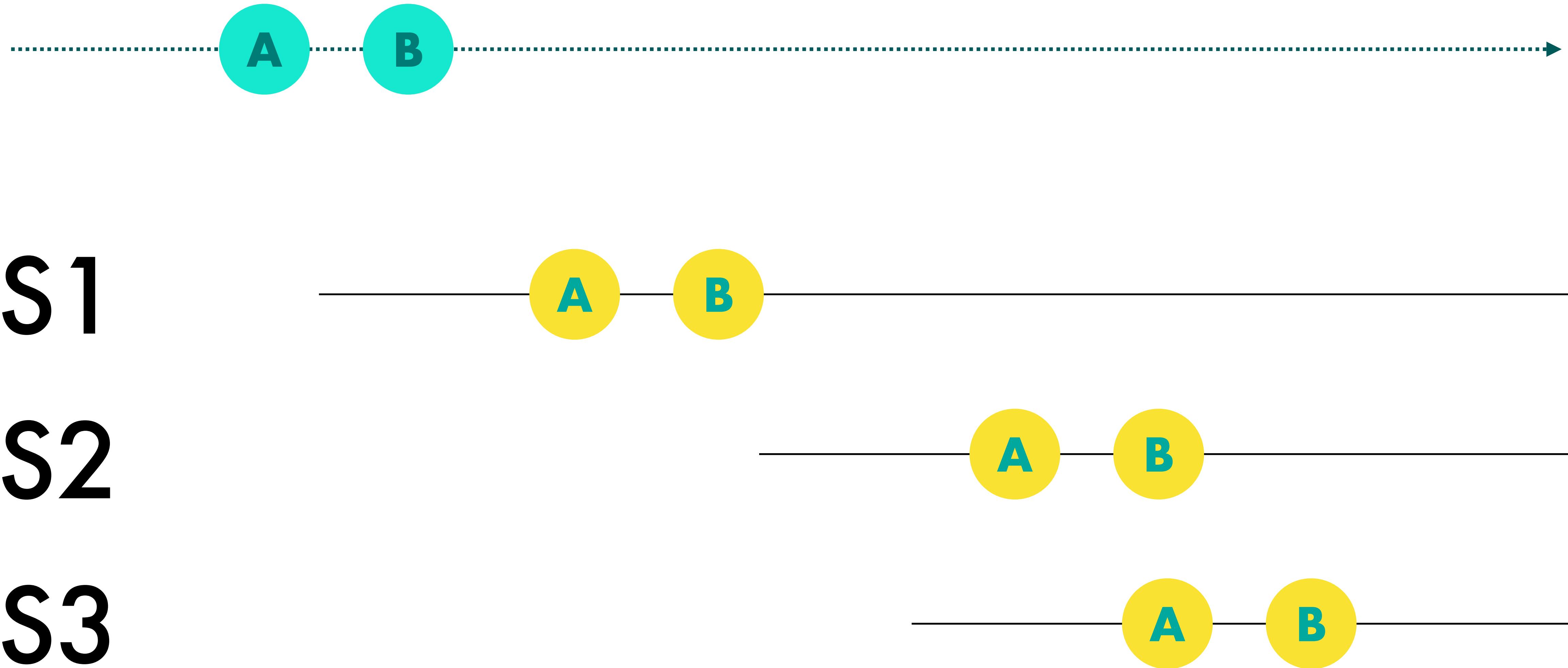
S2



S3



Cold



Gli Observable sono solo funzioni

...e una funzione non fa nulla finché non viene chiamata

```
const obs$ = new Observable(observer => {  
  observer.next(1);  
  
  observer.complete();  
});  
  
obs$.subscribe(console.log); // 1  
obs$.subscribe(console.log); // 1  
obs$.subscribe(console.log); // 1
```

DUBBIO #1

Se un Observable è cold di default,
cosa lo rende “osservabile”?

In natura, se osserviamo qualcosa, si muove indipendentemente dal fatto che lo stiamo guardando.



RISPOSTA

Se un observable è hot, qualcuno deve averlo avviato per noi.

C'è un'**entità nascosta** che ascolta l'observable e ci comunica i valori.

Spoiler: è un Subject

CONSEGUENZA #1

L'observable **hot** non riparte da zero ad ogni subscribe, si comporta come qualcosa di osservabile in natura.

CONSEGUENZA #2

I valori prodotti dal vero Observable
vengono condivisi con tutte le sottoscrizioni.

Perché è importante?

Poiché ogni valore viene generato 1 sola volta, il “costo” dell’emissione del valore non varia all’aumentare delle sottoscrizioni.

Esempio: Se un valore è il risultato di una chiamata Http, la chiamata viene eseguita una volta per tutti.



Definizione di Ben Lesh

(RxJS lead dev)

- Cold è quando l'Observable **crea** il PRODUTTORE
- Hot è quando l'Observable **utilizza** il PRODUTTORE (*senza crearlo*)

```
const source = new Observable((observer) => {
  const socket = new WebSocket('ws://someurl');
  socket.addEventListener('message', (e) => observer.next(e));
  return () => socket.close();
});
```

```
const socket = new WebSocket('ws://someurl');

const source = new Observable((observer) => {
  socket.addEventListener('message', (e) => observer.next(e));
});
```

Tip:

Pensate alla definizione di Ben Lesh come una conseguenza, non come punto di partenza. All'Observer non interessa del produttore, e pensarla in questi termini può trarvi in inganno.

Ricordate però che un Observable COLD genera nuovi valori per ogni subscription, anche se possono sembrare gli stessi.

Rendiamo HOT un Observable COLD

Introduciamo **publish** e **connect**

LIVE CODING

DUBBIO #2

**Ha senso iniziare a produrre valori se
nessuno sta ancora osservando?**

RISPOSTA

Dipende.

Un observable che non inizia a
produrre valori per conto proprio,
non è veramente hot.

È tiepido.

Rendiamo *tiepido* un Observable HOT

Introduciamo `RefCount`

LIVE CODING

DUBBIO #3

Come so se un observable è hot o cold?

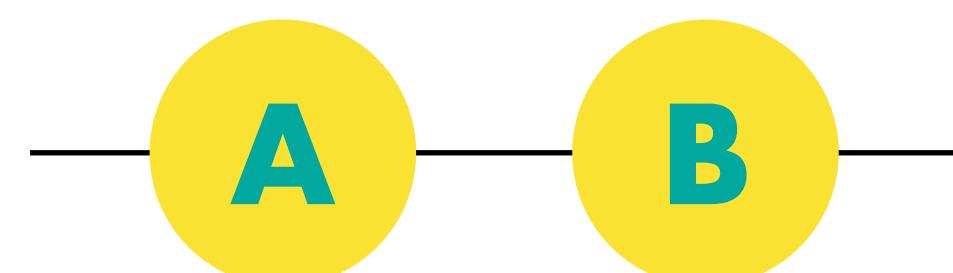
RISPOSTA

**Non lo sai. Qualcuno deve dirtelo, o
devi provare ad intuirlo.**

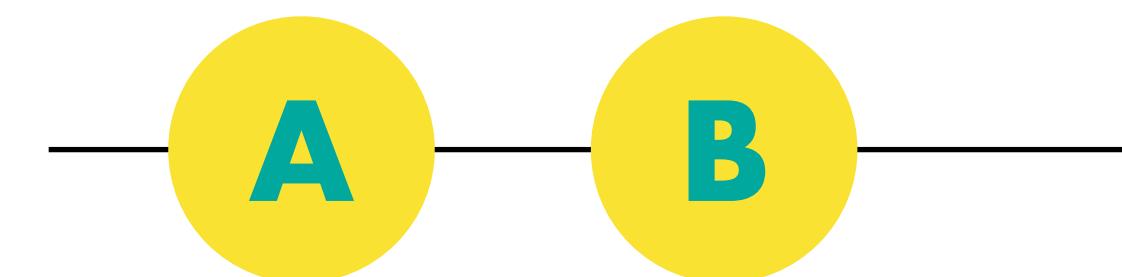
**Non possiamo sapere cosa è successo
prima del subscribe o dopo
l'unsubscribe.**

Quiz

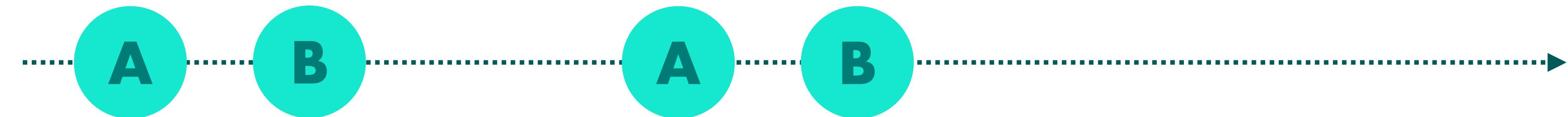
S1



S2



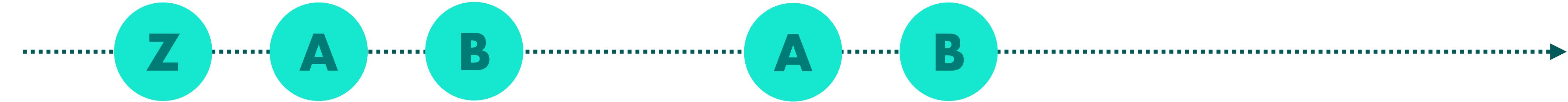
c 1



c 2



H 3



ESEMPIO: HttpClient, Angular

`http.get(...).subscribe();`

- Sappiamo che esegue una chiamata Http
- Sappiamo che la chiamata viene effettuata solo al subscribe
- Sappiamo che emette al massimo un valore, e poi completa

Quindi, presumiamo che:

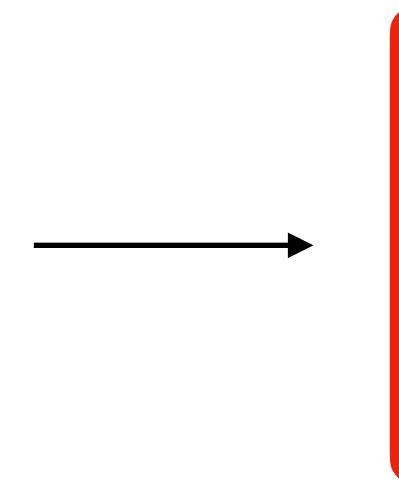
- Sia un observable cold
- Due subscribe allo stesso observable generino **due** chiamate distinte
- Dobbiamo evitare di usare una async pipe direttamente sull'observable, perché ogni ciclo di Change Detection farebbe ripartire la chiamata
- Se vogliamo condividere il risultato, dobbiamo usare 1 subscribe e salvare il risultato, oppure dobbiamo renderlo tiepido.

Multicasting HttpClient

```
http.get(...).pipe(  
  share()  
)
```

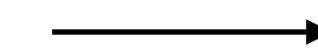


```
http.get(...).pipe(  
  multicast(() => new Subject()),  
  refCount()  
)
```



Di due subscribe, se la seconda arriva dopo l'emissione, la chiamata viene ripetuta

```
http.get(...).pipe(  
  shareReplay(1)  
)
```



```
http.get(...).pipe(  
  multicast(() => new ReplaySubject(1)),  
  refCount()  
)
```



Anche se la seconda subscribe arriva a chiamata terminata, viene ri-emesso, solo per lei, l'ultimo valore.
Ok!

Attenzione: utilizzatelo solo localmente e non a livello di servizi!

DUBBIO #4

Cos'è un Subject?

RISPOSTA

Un Subject è sia un Observable che un Observer. Vuol dire che possiamo:

- Effettuarci un subscribe
- Usarlo come observer da dare in pasto ad un subscribe
- Utilizzare le sue proprietà **next**, **error** e **complete** (ricordate: è un observer) per farlo emettere/crashare/completare manualmente

Tutti i tipi di Subject

Introduciamo `Subject`, `ReplaySubject`,
`BehaviorSubject`, `AsyncSubject`.

LIVE CODING

Il tempo è difficile

Gli Observable giocano con un concetto complesso: il tempo.
Ma i giochi più difficili sono quelli che danno più soddisfazione!



Grazie!

<https://stackblitz.com/edit/rxjs-multicasting-playground>