Reacting to Actions: Subject and BehaviorSubject

Deborah Kurata
Developer

https://www.youtube.com/@deborah_kurata

Acme Product Management Home Product List Cart

Products

- Leaf Rake
- Garden Cart
- Hammer
- Saw
- Video Game Controller

Product Detail for: Leaf Rake

Name: Leaf Rake

Add to Cart

Code: GDN-0011
Description: Leaf rake with 48-inch wooden handle
Price: $19.95
In Stock: 15

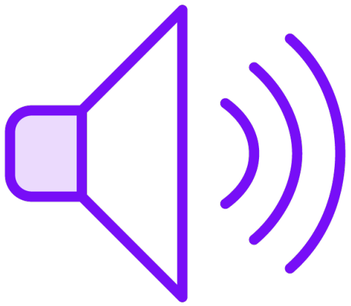
Review	Username	Text
Has no evil purpose	hama	This rake is worthy of honor
More than a tool	hama	The rake in the hand of a wizard may be more than a tool for the garden
A necessity!	eowyn	Those without rakes can still die upon them

```
export class ProductListComponent {
  pageTitle = 'Products';
  errorMessage = '';

  // Selected product id to highlight the entry
  selectedProductId: number = 0;

  onSelected(productId: number): void {
    this.selectedProductId = productId;
  }
```

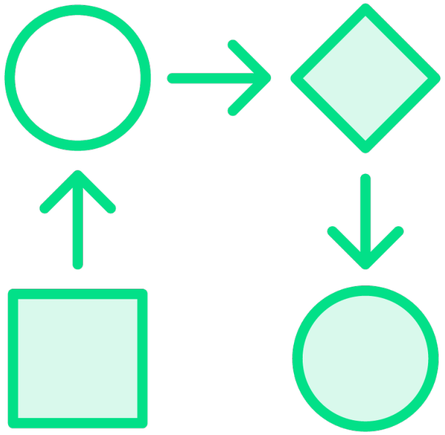
```
<div class='col-md-8'>
  <pm-product-detail [productId]="selectedProductId"></pm-product-detail>
</div>
```

```
export class ProductDetailComponent implements OnChanges, OnDestroy {
  @Input() productId: number = 0;
  errorMessage = '';

  // Product to display
  product: Product | null = null;

  // Set the page title
  pageTitle = this.product ?
    `Product Detail for: ${this.product.productName}` : 'Product Detail';
```

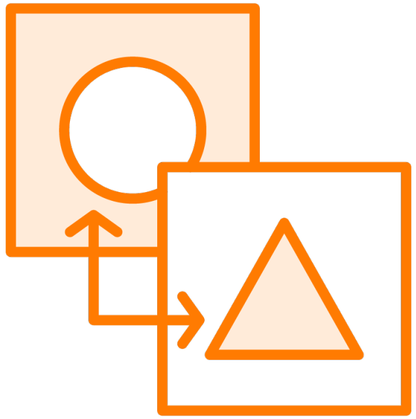
Reacting to User Actions with Observables



Emit notification



React to that notification

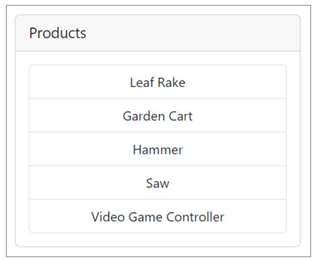


Share data between components

Improved code that is more explicit

Reacting to User Actions with Observables

Product List



Product Detail

I want a notification

I notify when the user selects a product

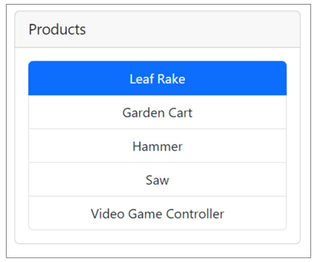
Templates

Components

Services

Reacting to User Actions with Observables

Product List



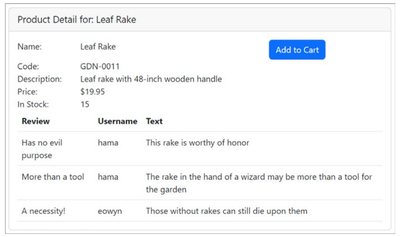
I'll style the selected row blue

I'll call the service to get the product

I notify when the user selects a product

Id

Product Detail

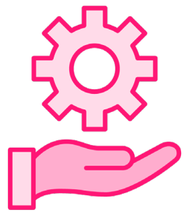


Templates

Components

Services

Creating an Observable



Work with an observable Angular creates for us

Use creation functions

Use a type of Subject

Overview

Observable vs. Subject

BehaviorSubject

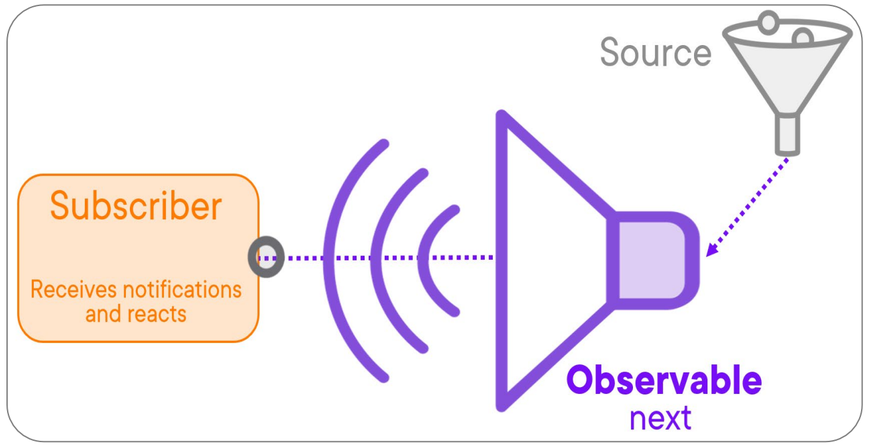
Creating a Subject and emitting notifications

Using a Subject to react to user actions:

- To communicate between components
- And get data using a variable URL

Combining observables to work with multiple sets of data

Observable Is Read-only



From the point of view of the subscriber

- An Observable is read-only
- Subscribe to react to its notifications and receive emissions
- Can't emit anything into it

Only the creator of the Observable can emit items into that Observable

Only the Observable Creator Can Emit Items

```
const apples$ = of('Apple1', 'Apple2');
```

```
const apples$ = from(['Apple1', 'Apple2']);
```

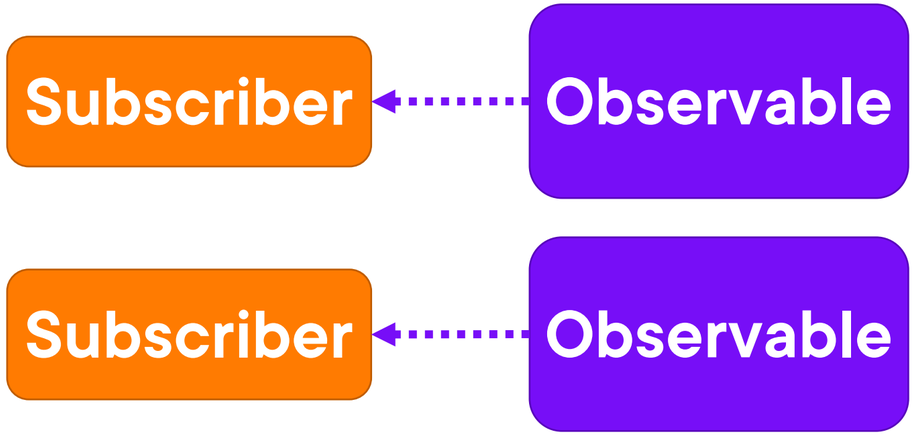
```
products$ = this.http.get<Product[]>(this.productsUrl)
  .pipe(
    catchError(err => this.handleError(err))
);
```

Subject

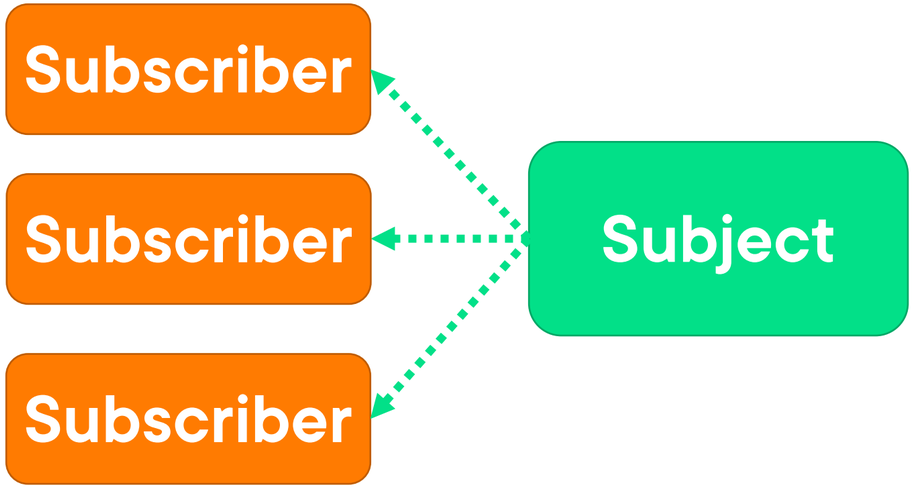
A Subject is a special type of Observable that allows values to be multicast to many Observers.

RxJS documentation: https://rxjs.dev/api/index/class/Subject

Unicast vs. Multicast

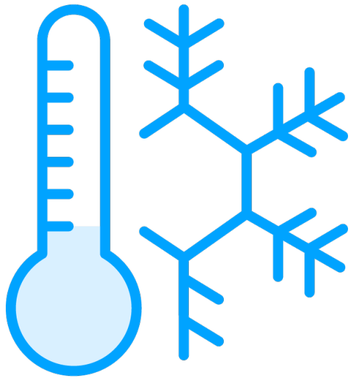


Observable is generally unicast



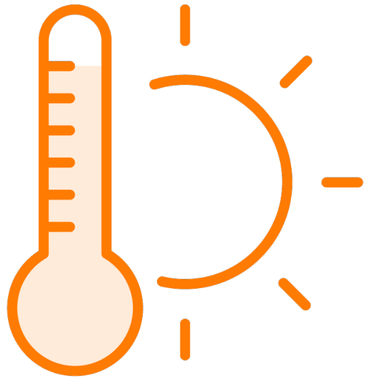
Subject is multicast

Hot and Cold Observables



Doesn't emit until subscribed to
Cold observable
Subscription activates the source
Unicast

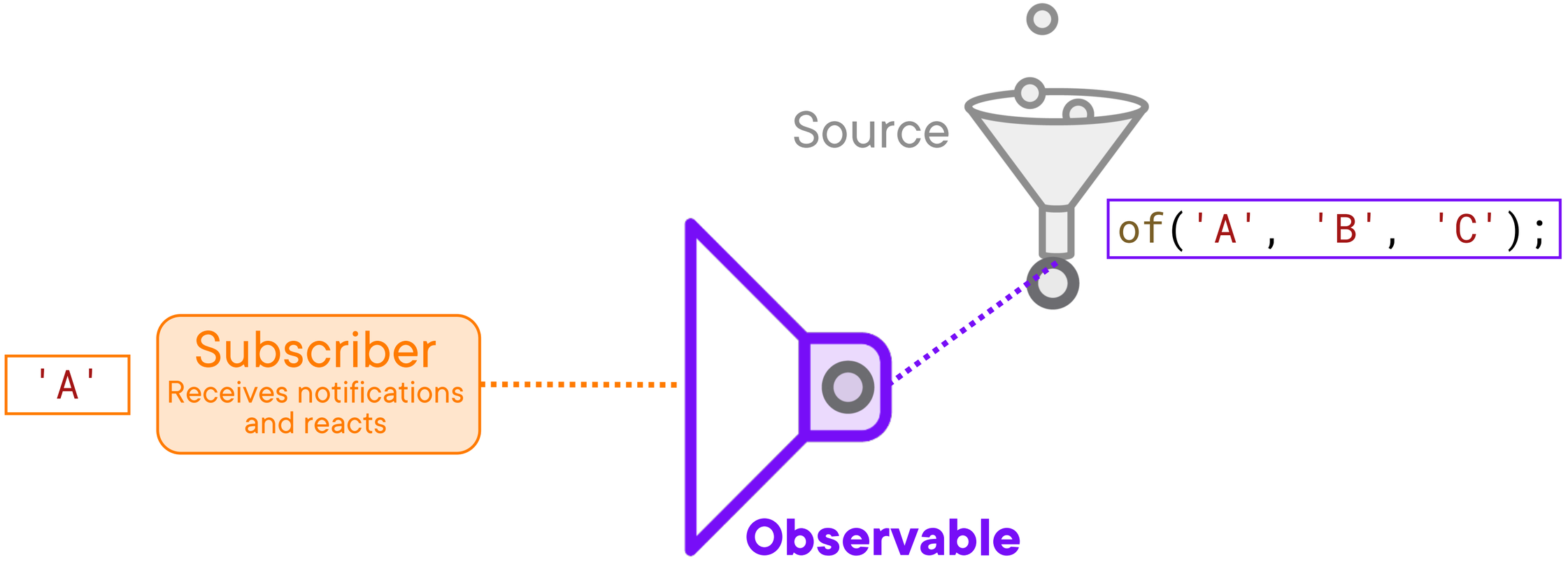
```
products$ = this.http.get<Product[]>(url);
```



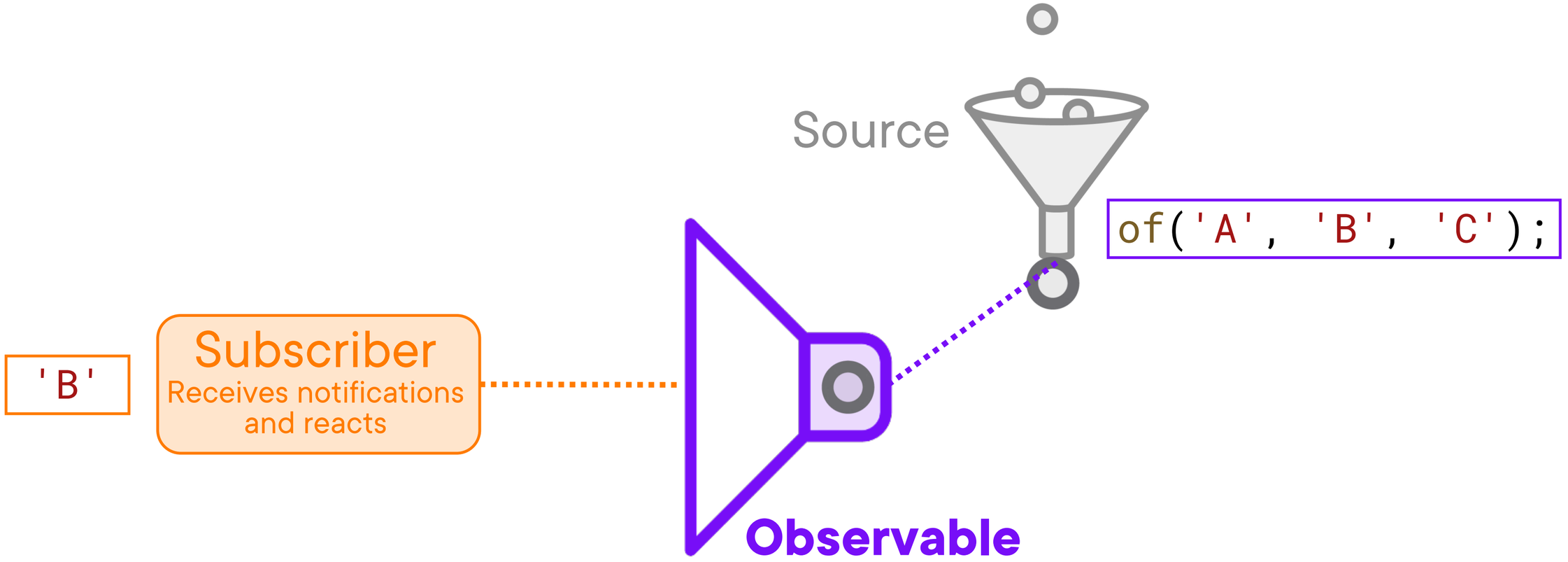
Creation activates the source
Emits without subscribers
Hot observable
Multicast

```
productSubject = new Subject<number>();
this.productSubject.next(12)
```

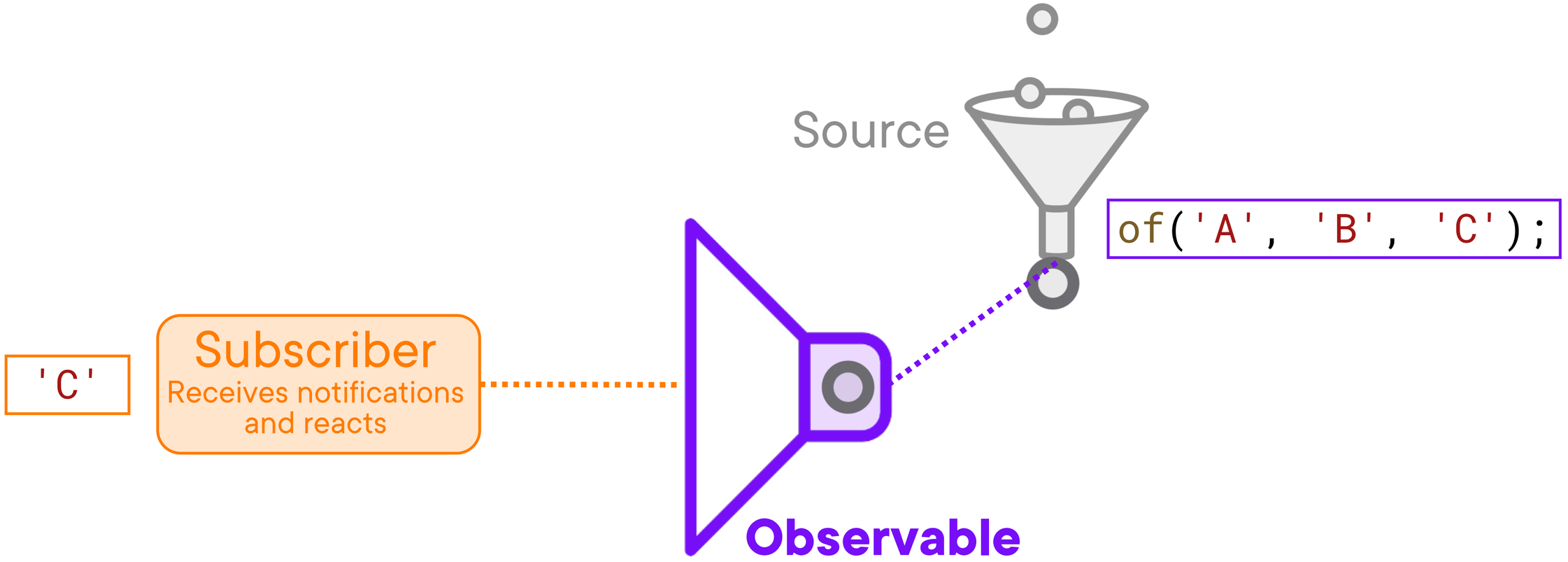
Cold/Unicast



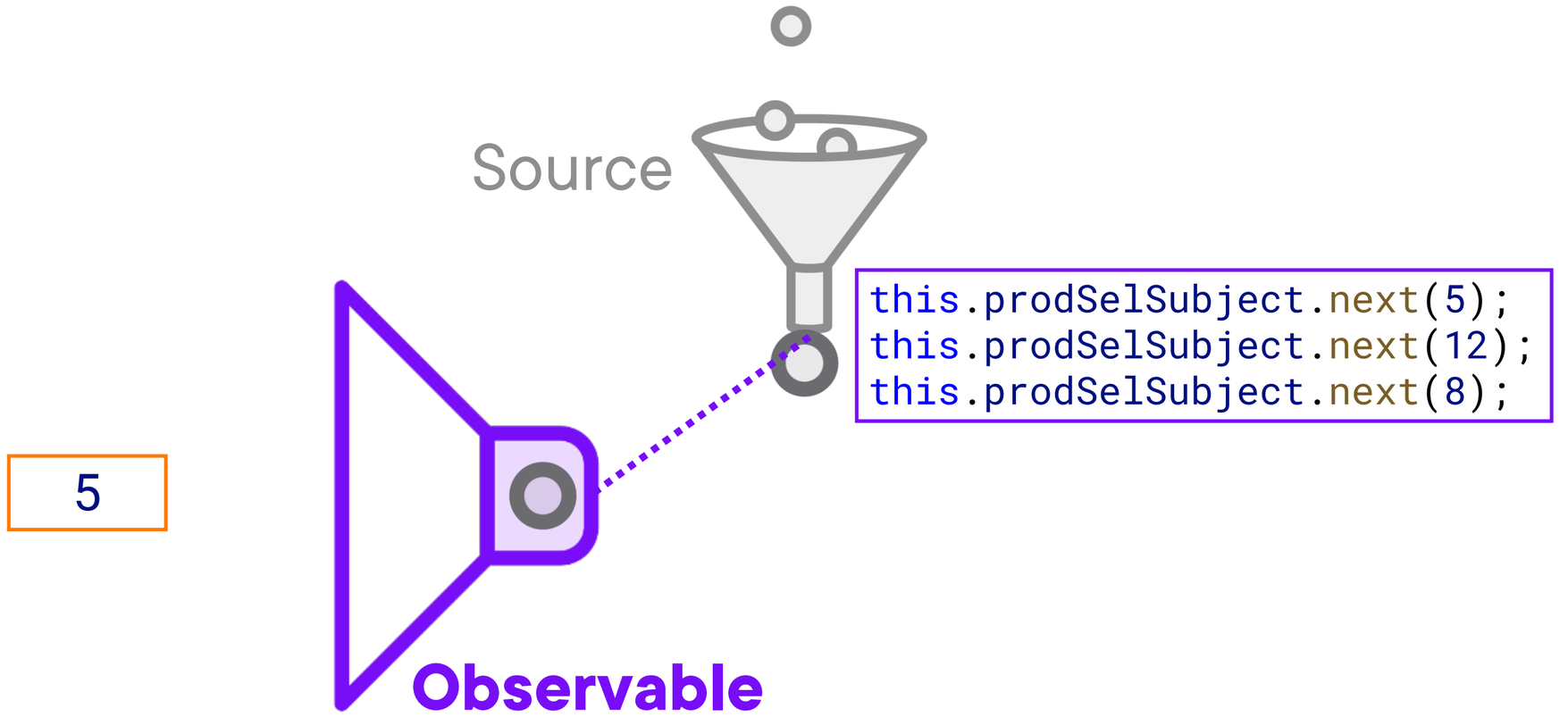
Cold/Unicast



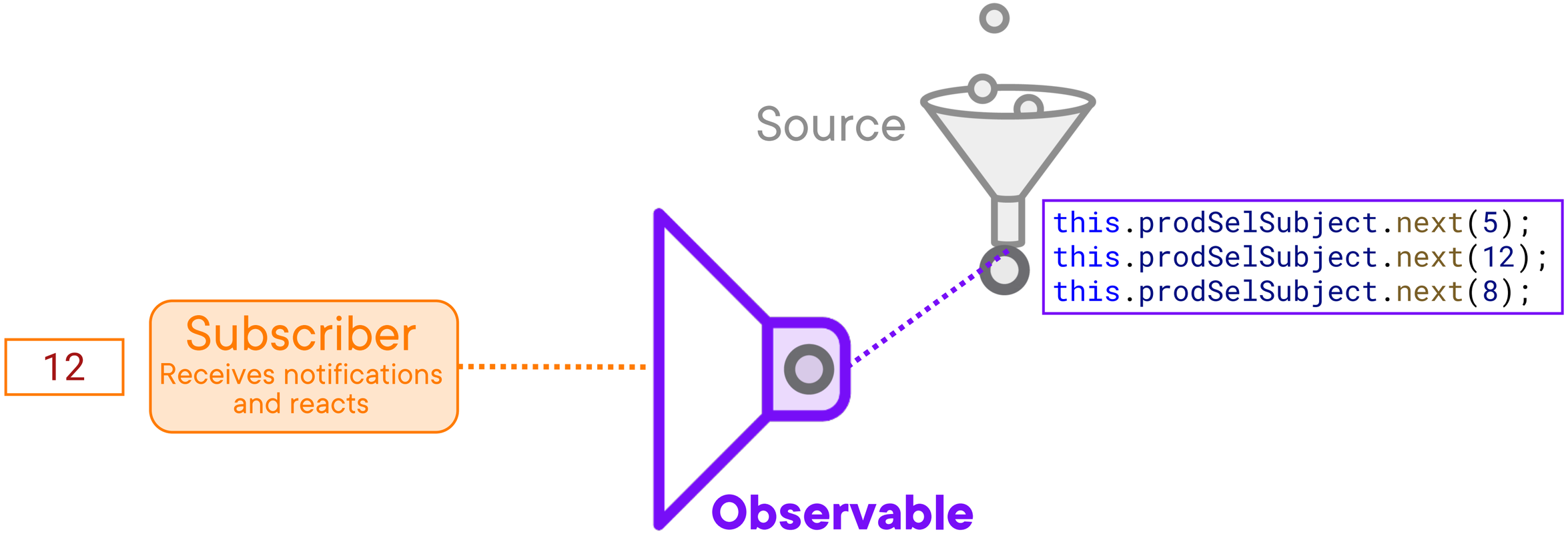
Cold/Unicast



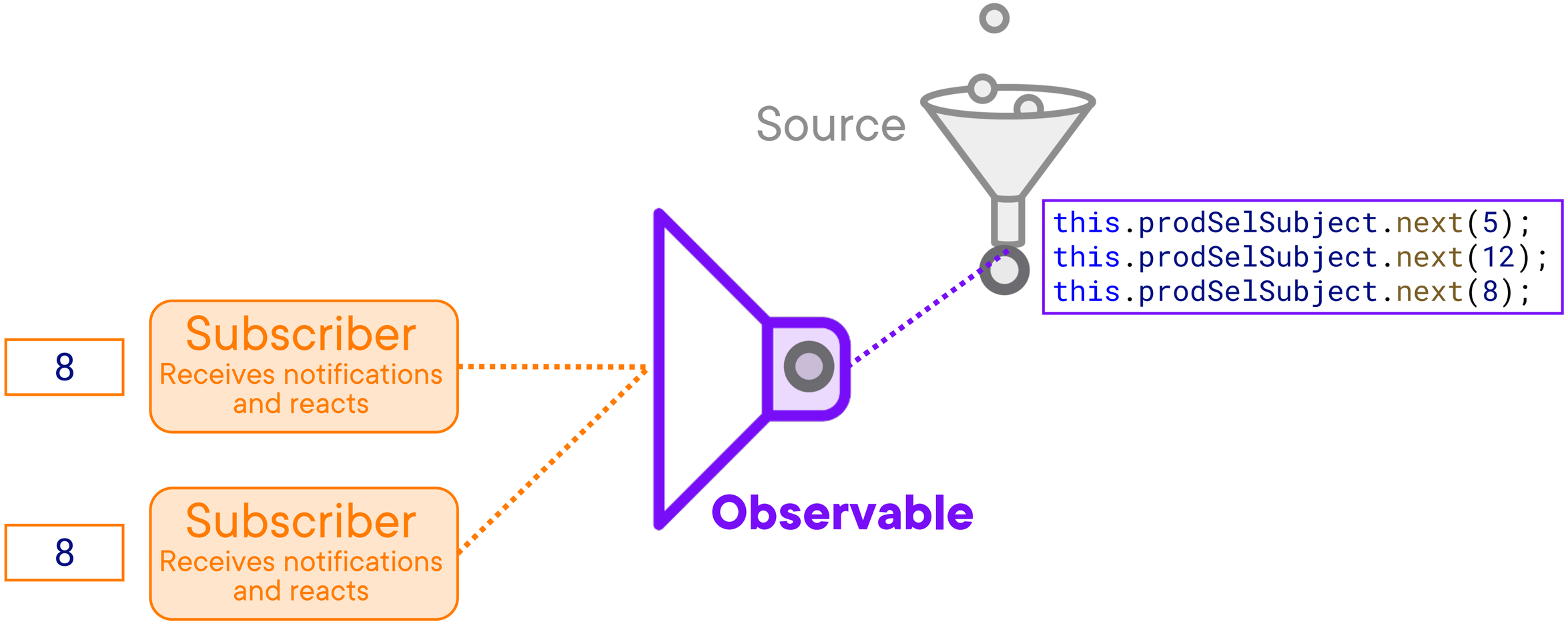
Hot/Multicast: Subject



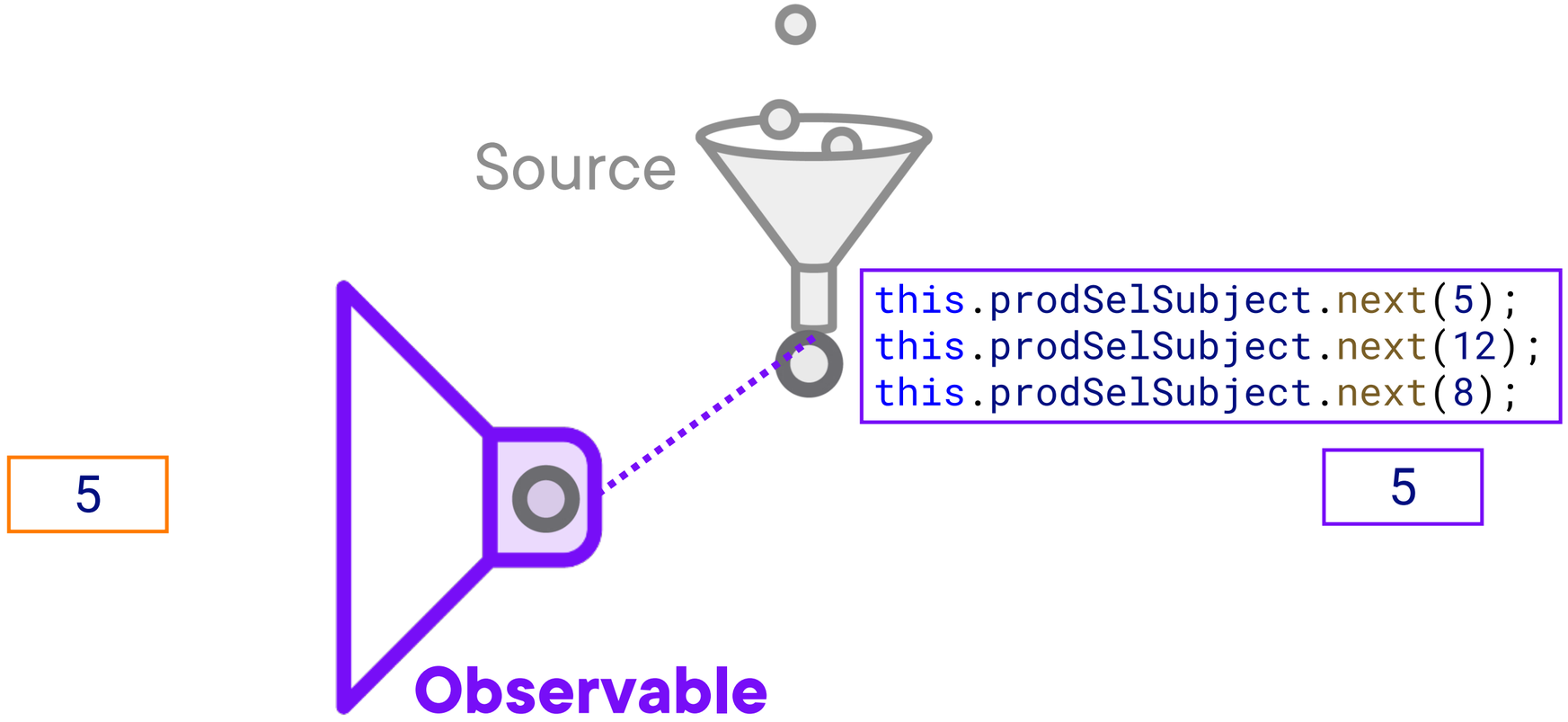
Hot/Multicast: Subject



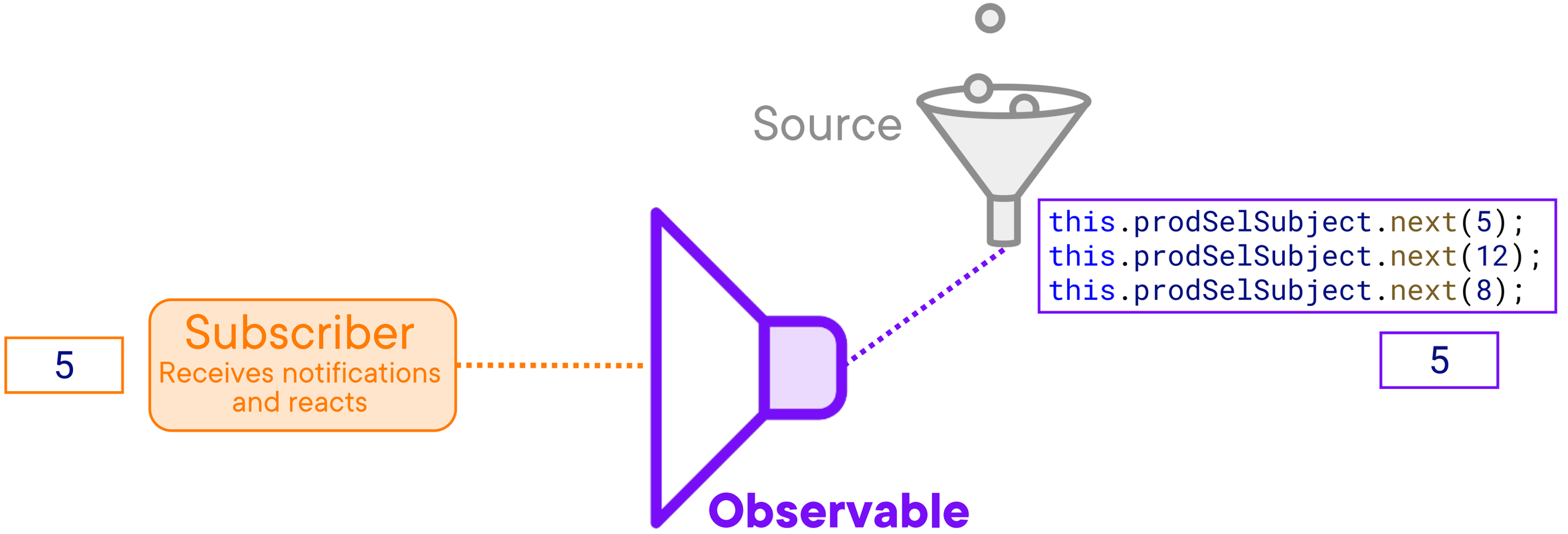
Hot/Multicast: Subject



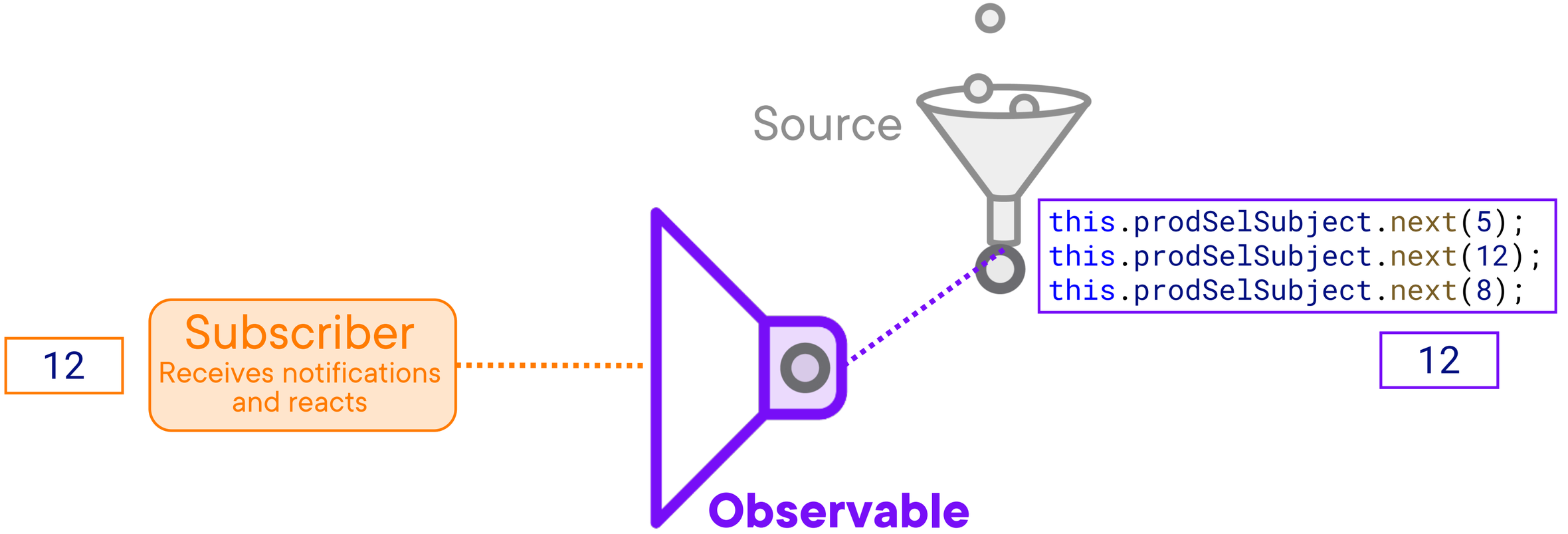
Hot/Multicast: BehaviorSubject



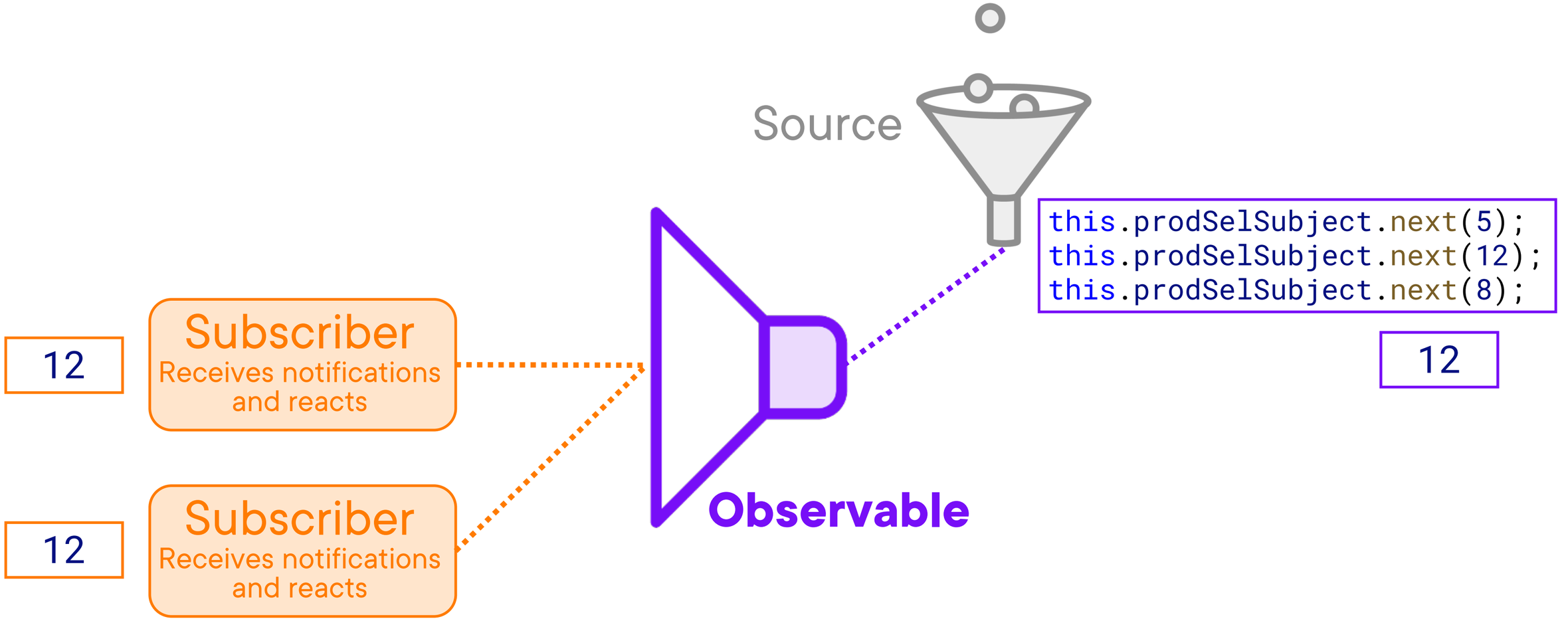
Hot/Multicast: BehaviorSubject



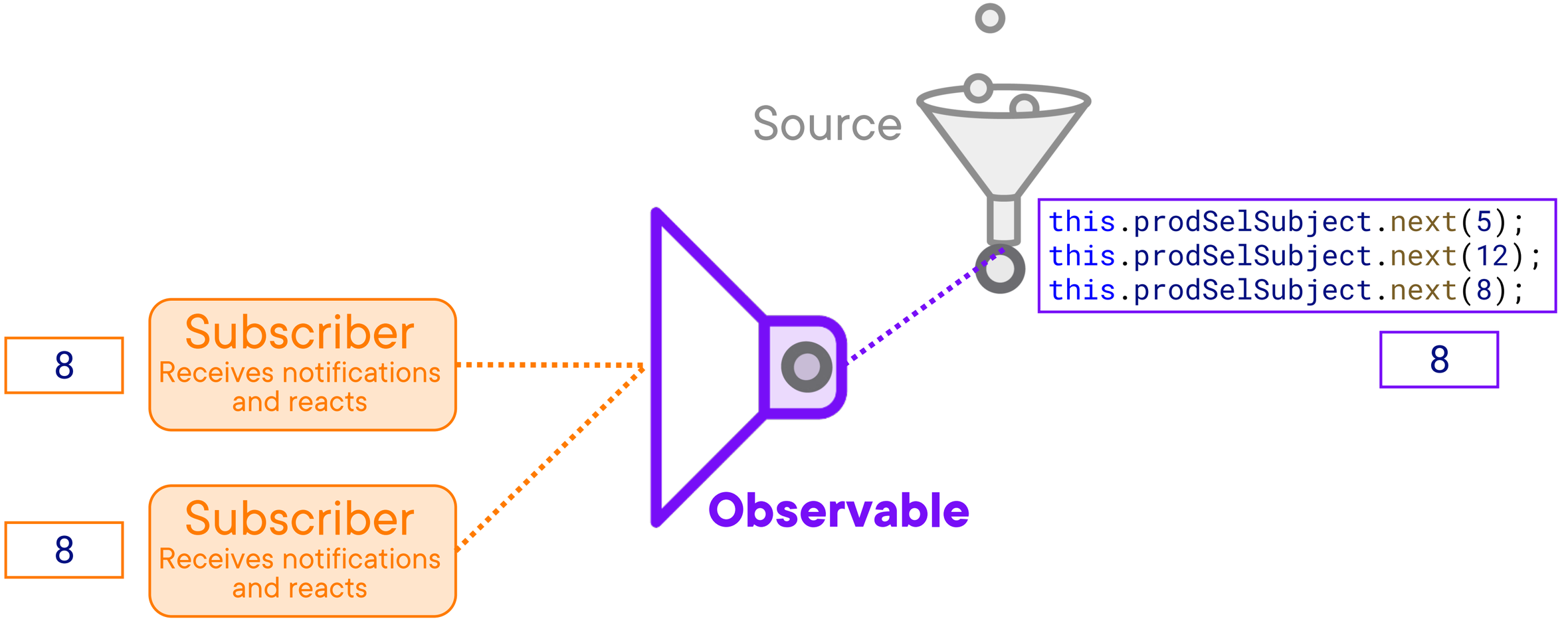
Hot/Multicast: BehaviorSubject



Hot/Multicast: BehaviorSubject



Hot/Multicast: BehaviorSubject



Creating a Subject

I'm not concerned about losing an emission

```
private productSelectedSubject = new Subject<number>();
productSelected$ = this.productSelectedSubject.asObservable();
```

I will always emit an initial value or the prior emission

```
private productSelectedSubject = new BehaviorSubject<number>(0);
productSelected$ = this.productSelectedSubject.asObservable();
```

Emitting a Notification with a Value

```
private productSelectedSubject = new Subject<number>();
productSelected$ = this.productSelectedSubject.asObservable();
```

```
this.productSelectedSubject.next(productId);
```

```
this.productSelectedSubject.error(errorText);
```

```
this.productSelectedSubject.complete();
```

Demo

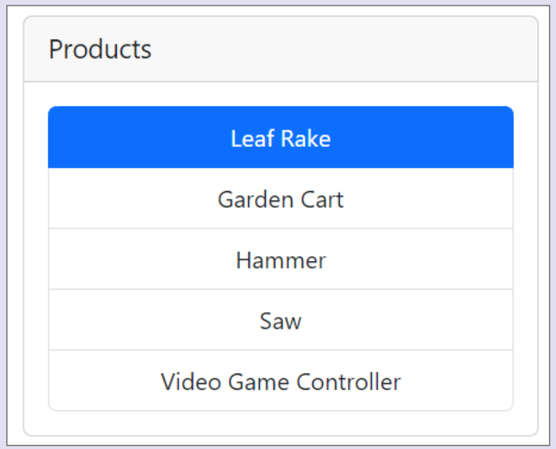
Create a Subject

Call its next method to emit a notification with a value

Reacting to Actions

```
private productSelectedSubject = new Subject<number>();
productSelected$ = this.productSelectedSubject.asObservable();
```

```
productSelected(selectedProductId: number): void {
  this.productSelectedSubject.next(selectedProductId);
}
```



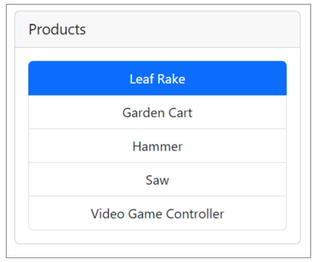

```
this.sub = this.productService.productSelected$.subscribe(…)
```

Demo

React to notifications from a Subject

Reacting to User Actions with Observables

Product List



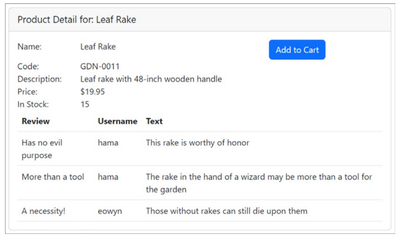
I'll style the selected row blue

I'll call the service to get the product

I notify when the user selects a product

Id

Product Detail



Templates

Components

Services

Demo

React to notifications and get data

When you have a variable URL ...

Key Point

```
private productSelectedSubject = new Subject<number>();
productSelected$ = this.productSelectedSubject.asObservable();
```

```
productSelected(selectedProductId: number): void {
 this.productSelectedSubject.next(selectedProductId);
}
```

```
product$ = this.productSelected$
   .pipe(
     switchMap(id => {
       const url = this.productsUrl + '/' + id;
       return this.http.get<Product>(url)
         .pipe(
           catchError(err => this.handleError(err))
         );
     })
   );
```

Demo

React to the retrieved product

- Display the product details

Combining Observables

Combine observables

Product Detail for: Leaf Rake

Name:	Leaf Rake
Code:	GDN-0011
Description:	Leaf rake with 48-inch wooden handle
Price:	$19.95
In Stock:	15

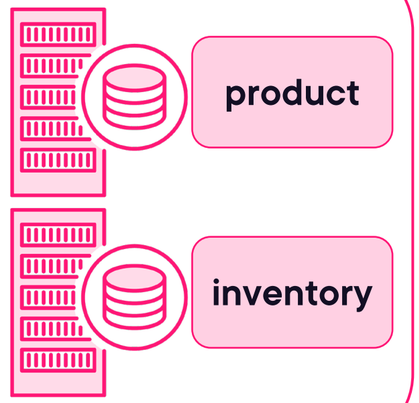
Template

product with inventory

Component

product with inventory

Service



Backend Server

Combining Observables

```
product$ = combineLatest([
  this.productSelected$,
  this.products$
]).pipe(
  map(([selectedProductId, products]) =>
    products.find(product => product.id === selectedProductId)
  )
)
```

RxJS Creation Function: `combineLatest`

Combines a set of observables

```
combineLatest([a$, b$, c$])
```

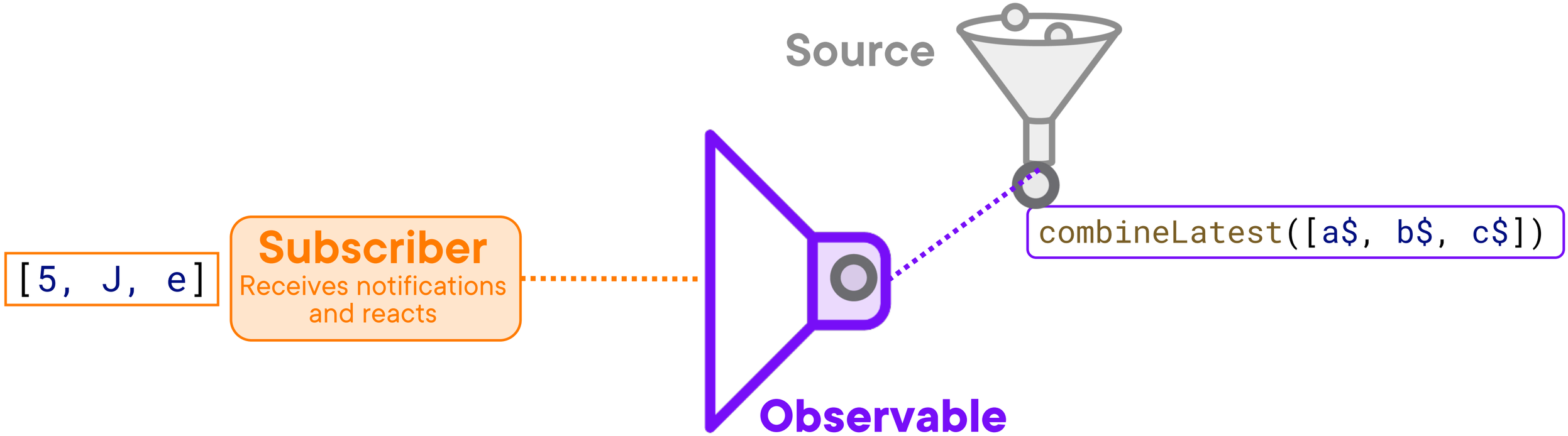
Creates an observable whose values are defined:

- Using the latest values from each input observable

Won't emit anything until each of the input observables emit at least once

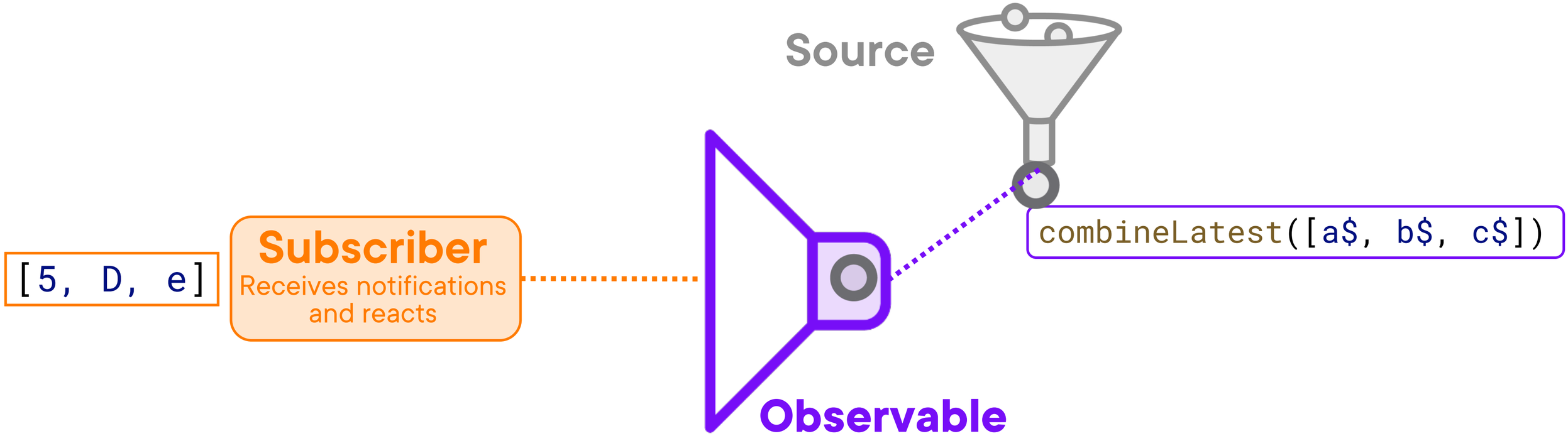
combineLatest()

a$	5
b$	J
c$	e

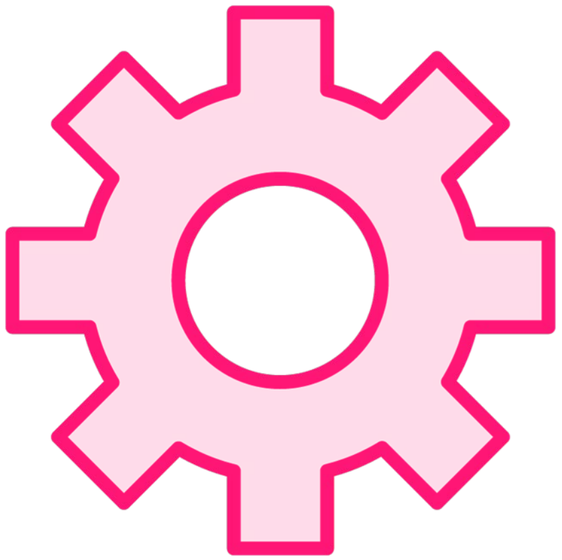


combineLatest()

a$	5
b$	D
c$	e



RxJS Creation Function: `combineLatest`



`combineLatest` **is a combination function**

When an item is emitted from any observable

- If all observables have emitted at least once
- Emits a value to the output observable

Emitted value combines the latest emitted value from each input observable into an array

Completes when all input observables complete

Demo

Combine observables to

- React to emissions from any of them
- And work with all the emissions in a single pipeline

RxJS Subject

Special type of observable we use to react to user actions and other events

```
private productSelectedSubject = new Subject<number>();
productSelected$ = this.productSelectedSubject.asObservable();
```

Use a Subject to emit notifications:

- next, error, complete

```
 productSelected(selectedProductId: number): void {
  this.productSelectedSubject.next(selectedProductId);
}
```

With the next notification, emit custom data:

- Numbers, strings, arrays, objects, anything

Communicate data to any components or services that subscribe

Any values emitted before a subscriber subscribes are lost

RxJS Behavior Subject

Special type of Subject that emits an initial value

```
private productSelSubject = new BehaviorSubject<number>(0);
productSelected$ = this.productSelSubject.asObservable();
```

- If the BehaviorSubject hasn't emitted, it emits the **value provided in the constructor** to any new subscribers
- Otherwise, it **buffers its latest emitted** value and **emits that** to any new subscribers

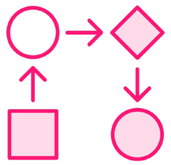
Use a BehaviorSubject to ensure every new subscriber receives an initial value

React to User Actions and Events

Declare a Subject

Emit a notification from that Subject when the action or event occurs

Subscribe to the observable (read-only) part of that Subject to start receiving notifications



React to those notifications

How Do We React?

Any way we want!

Modify something in the template

Change state (data)

Get data, using the notification as a trigger

Get data, using the emitted value in the URL

```
products$ = this.refresh$
  .pipe(
    tap(() => this.isLoadingSubject.next(true)),
    mergeMap(() => this.http.get<Product[]>(this.productsUrl)
      .pipe(
        catchError(this.handleError)
      )),
    tap(() => this.isLoadingSubject.next(false)),
    shareReplay(1)
  );
```

RxJS
combineLatest

Combine multiple observables and emit when any of those observables emit

```
products$ = combineLatest([
  this.products$,
  this.currentPage$,
  this.pageSizeAction$
])
  .pipe(
    map(([products, pageNumber, pageSize]) =>
      products.slice((pageNumber - 1) * pageSize,
        pageNumber * pageSize)
    )
  );
```

Use `combineLatest`

- To combine multiple sets of data for use in the template
- To work with multiple observables in the pipeline
- To wait for all observables to emit before performing an operation

"Passing" Data to Use in a URL

```
private productSelectedSubject = new Subject<number>();
productSelected$ = this.productSelectedSubject.asObservable();
```

```
product$ = this.productSelected$
  .pipe(
    switchMap(id => {
      const productUrl = this.productsUrl + '/' + id;
      return this.http.get<Product>(productUrl)
        .pipe(
          catchError(err => this.handleError(err))
        );
    })
  );
```

For More Information

Demo code

- https://github.com/DeborahK/angular-rxjs-signals-fundamentals

RxJS Documentation

- https://rxjs.dev/api/index/class/Subject
- https://rxjs.dev/api/index/class/BehaviorSubject
- https://rxjs.dev/api/index/function/combineLatest

"4 Wicked Pipelines for RxJS in Angular"

- https://youtu.be/wQ8jXlWMoCo

Up Next:

Introduction to Angular Signals