RxJS Terms and Syntax

Deborah Kurata
Developer

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

Overview

Observable

Subscription

Observer

Observable creation functions

Observe and React to Events through Time

Observe events or data

- Apple emitted onto the conveyor

Subscribe to receive notifications

- Notified as each apple is emitted

React to each notification

- Transform
- Filter
- Modify

Conveyor

Start

Notified as each apple is emitted

Item passes through a set of operations

As an observer

Next item, process it

Error occurred, handle it

Complete, you're done

Stop

Conveyor -> Observable

Apple Factory	RxJS
Start	`subscribe()`
Notified as each apple is emitted	- Notified of emissions
Item passes through a set of operations	`pipe()` **through a set of operators**
As an observer	**Observer**
Next item, process it	- `next()`
Error occurred, handle it	- `error()`
Complete, you're done	- `complete()`
Stop	`unsubscribe()`

Observable

A collection of events or data values emitted over time

An observable is created from an event or data source

- User actions
- Application events (routing, forms)
- Response from an HTTP request
- Internal structures

Observable Emits Notifications

Next item is emitted

Error occurred, no more items are emitted

Complete, no more items are emitted

Observables Can Emit

- Primitives: numbers or strings
- Events: mouse, key, valueChanges, routing
- Objects: customers, products
- Other data structures: arrays, observables
- HTTP response

Observables

Synchronous

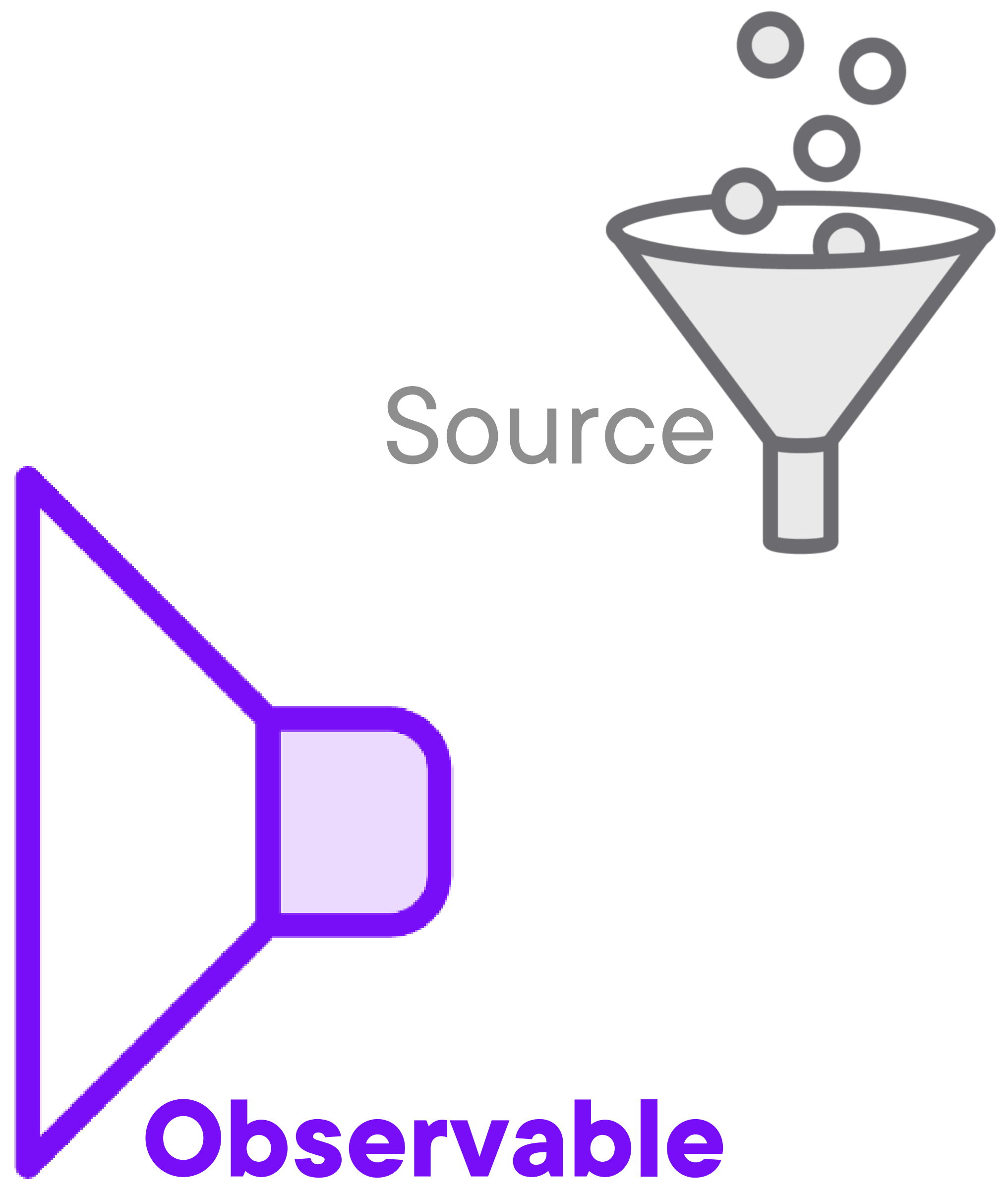
Asynchronous

[1,2,3]

Finite emissions

Infinite emissions

Observable



Subscription

Start

Notified as each apple is emitted

Item passes through a set of operations

As an observer

Next item, process it

Error occurred, handle it

Complete, you're done

Stop

Subscription

Start

Notified as each apple is emitted

Item passes through a set of operations

As an observer

Next item, process it

Error occurred, handle it

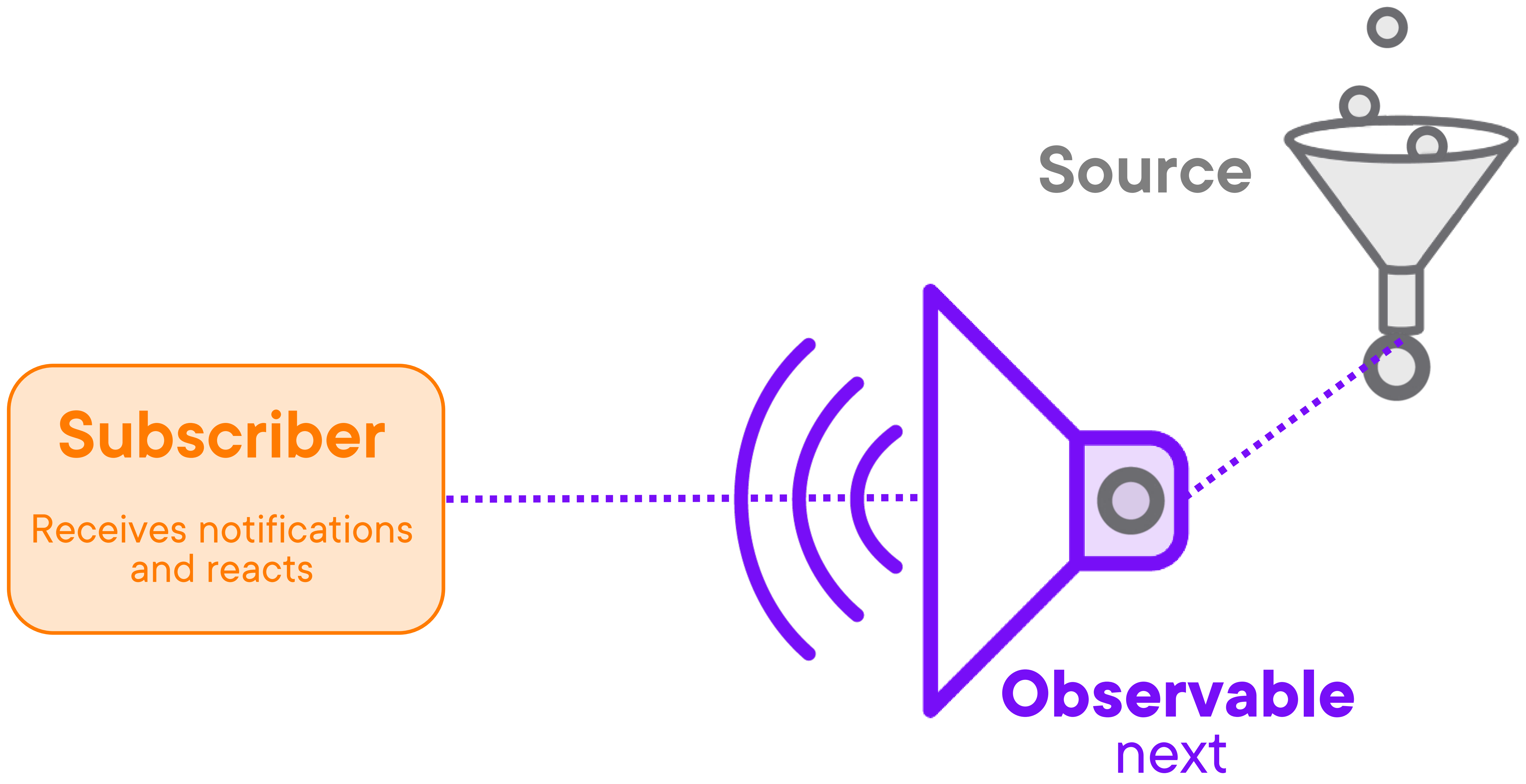
Complete, you're done

Stop

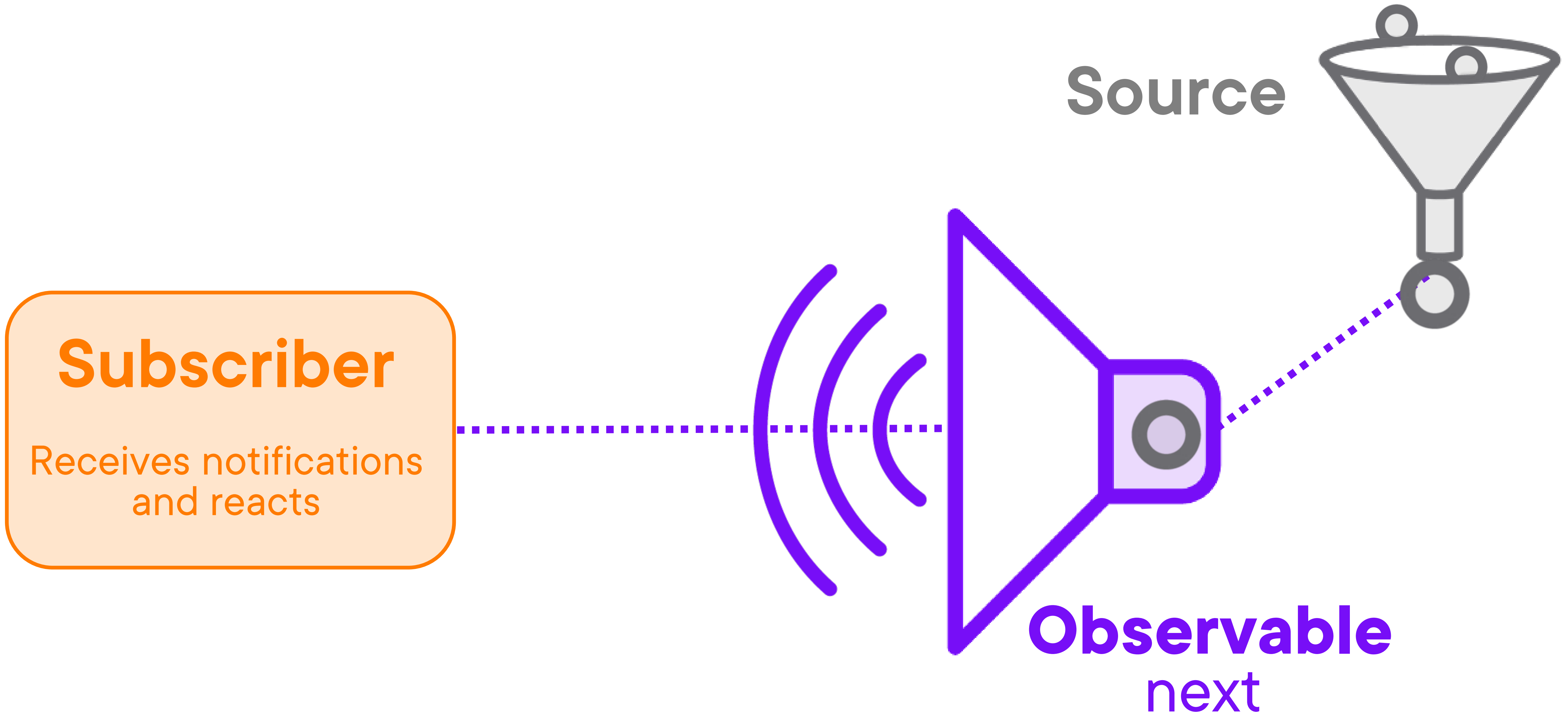
Call `subscribe()` on the observable

MUST subscribe to start receiving notifications

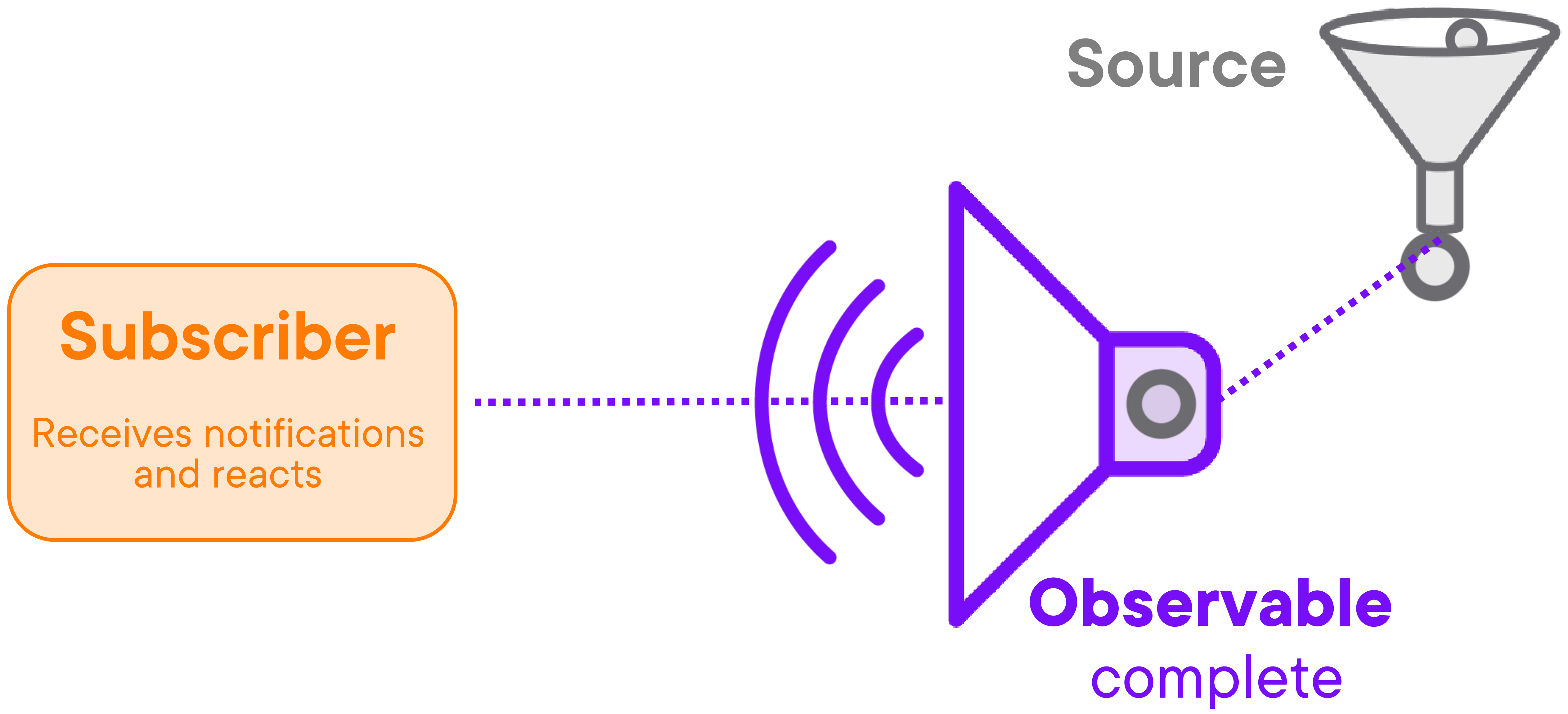
Subscribing to an Observable



Subscribing to an Observable



Subscribing to an Observable



Subscription

```
apples$.subscribe();
```

Key Point

Suffix an observable with a dollar sign ($)

```
apples$.subscribe();
```

Why?

Easier to recognize the variable as an observable that requires subscribing

Subscription

```
apples$.subscribe();
```

Calling `subscribe()` returns a subscription which represents the execution of the observable

```
this.sub = apples$.subscribe();
```

```
this.sub.unsubscribe();
```

Key Point

If you **subscribe** …
you should always **unsubscribe**

```
this.sub = apples$.subscribe();
```

```
this.sub.unsubscribe();
```

Why?

Properly unsubscribing helps avoid memory leaks and hard to find bugs

Subscription

```
sub!: Subscription;
```

```
this.sub = apples$.subscribe();
```

```
this.sub.unsubscribe();
```

Observer

Next item, process it

Error occurred, handle it

Complete, you're done

Observer

Observes and responds to notifications

Next item, process it	`next: item => …`
Error occurred, handle it	`error: err => …`
Complete, you're done	`complete: () => …`

Explicit Observer (Uncommon)

```
// Define an explicit observer (uncommon)
const observer = {
  next: apple => console.log(`Value emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples`)
};
```

```
const sub = apples$.subscribe(observer);
```

Pass Observer to Subscribe

```
// Pass the next callback function
this.sub = apples$.subscribe(
  apple => console.log(`Value emitted ${apple}`)
);
```

```
// Pass an observer object with one or more callbacks
this.sub = apples$.subscribe({
  next: apple => console.log(`Value emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples`)
});
```

Retaining Emitted Items

```
this.sub = keyPresses$.subscribe(
  event => console.log(`Value emitted ${event.key}`)
);
```

```
const keys: string[] = [];
this.sub = keyPresses$.subscribe(
  event => keys.push(event.key)
);
```

Creating an Observable

Work with an observable Angular creates for us

Use creation functions

Create a Subject

Creation Functions

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

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

```
const clicks$ = fromEvent(document, 'click');
```

```
const items$ = timer(initialDelay);
```

```
const items$ = timer(initialDelay, subsequentDelay);
```

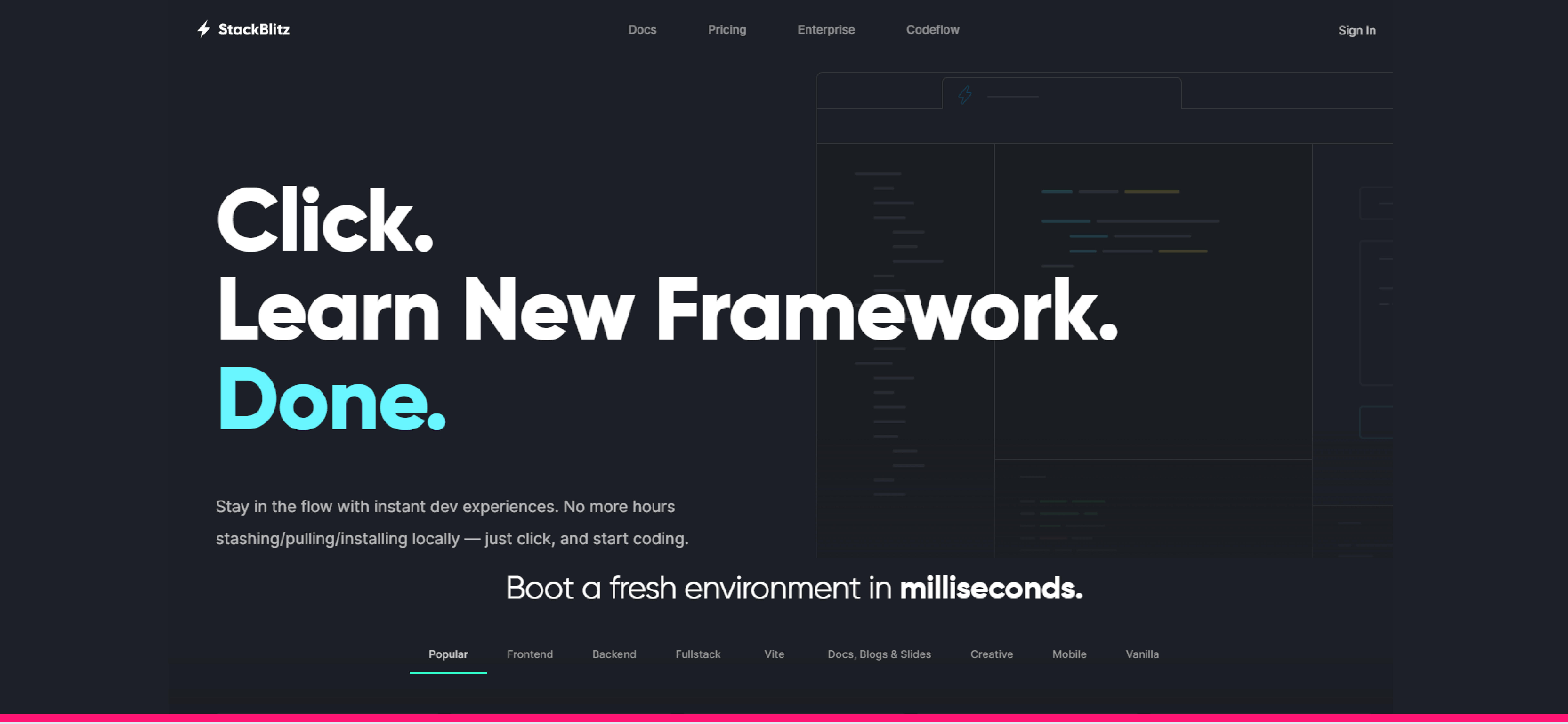
Demo

Create observables

- `of()`
- `from()`

Subscribe to each observable

Use an observer to react to notifications



Click.
Learn New Framework.
Done.

Stay in the flow with instant dev experiences. No more hours stashing/pulling/installing locally — just click, and start coding.

Boot a fresh environment in **milliseconds.**

https://stackblitz.com

Demo

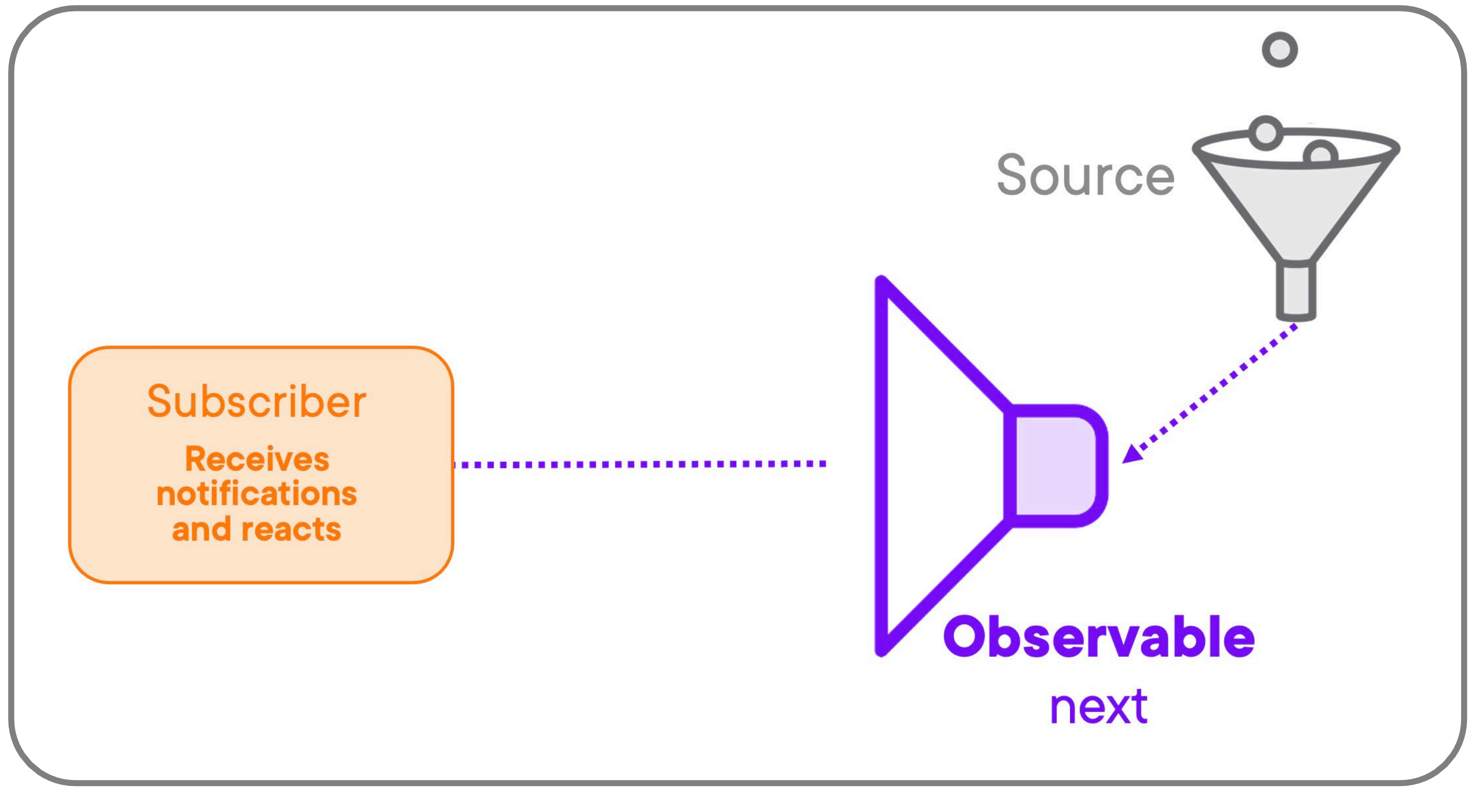
Create an observable

- `fromEvent()`

RxJS Terms

Observable

- Collection of events or values observed over time
- Connects to a source
- Emits notifications when it receives data or an event from that source



Subscription

- Tells the observable that we are ready for notifications
- `subscribe()` returns a Subscription
- Use that Subscription to unsubscribe

Observer

- Observes notifications from the observable
- Reacts to those notifications: `next()`, `error()`, `complete()`

Subscribing to an Observable

Subscription

Observable

Observer

```
this.sub = apples$.subscribe({
  next: apple => console.log(`Emitted: ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples`)
});
```

```
this.sub.unsubscribe();
```

Creating an Observable

Returned from an Angular feature

- Forms: `valueChanges`
- Routing: `paramMap`
- HTTP: `get`

Creation functions

- `of`, `from`, `fromEvent`, `timer`, ...
- Create an observable from anything

Subject

- Our code is the source
- We emit notifications and our own values

Best Practices

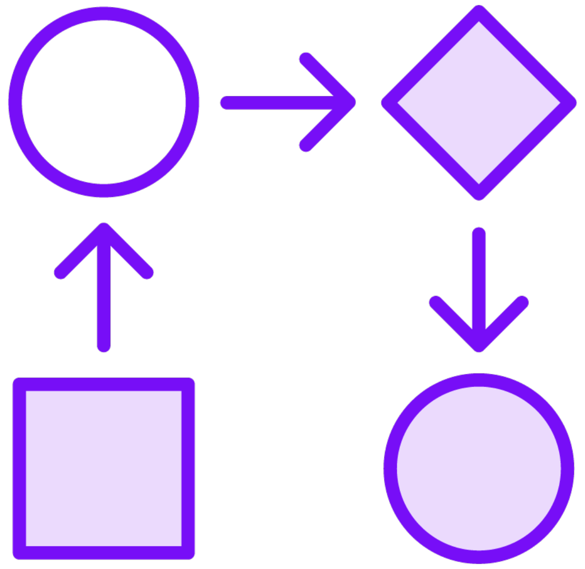
Append a dollar sign ($)

```
this.sub = apples$.subscribe(…);
```

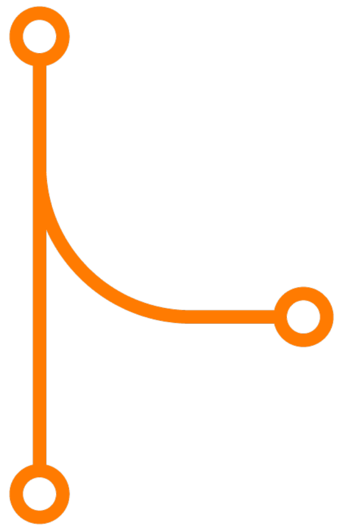
```
this.sub.unsubscribe();
```

If you subscribe, be sure to unsubscribe

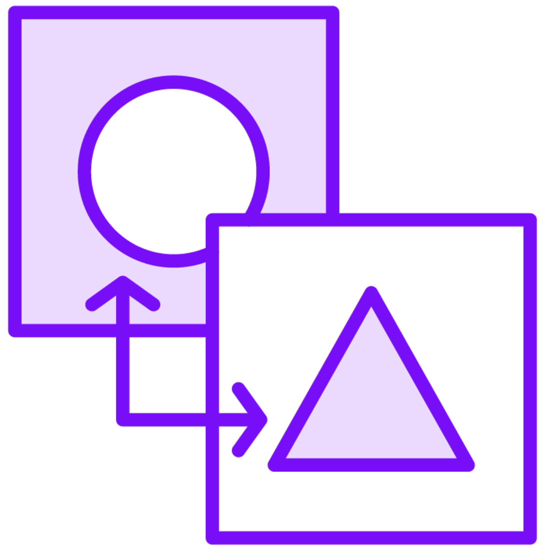
Benefits of Observables



Build more reactive code



Compose observables to combine sets of data



Pass data to other parts of the application

For More Information

RxJS documentation

- https://rxjs.dev/

"RxJS in Angular: Terms, Tips and Patterns"

- https://youtu.be/vtCDRiG_D4

Demo code

- https://stackblitz.com/edit/rxjs-signals-m3-deborahk

Up Next:

RxJS Operators