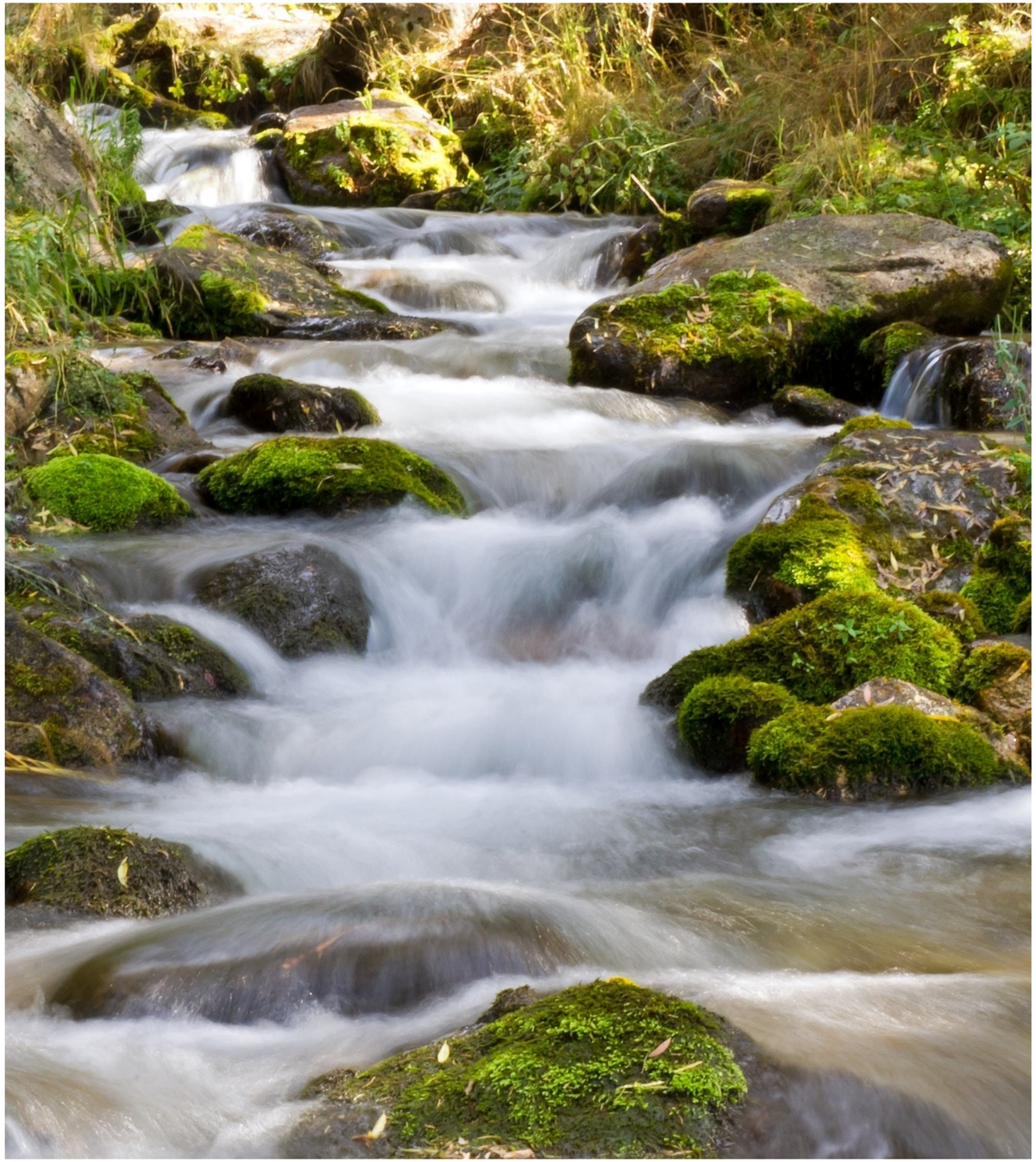
RxJS Terms and Syntax

Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata



Conveyor

Start

Emits items

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 - Emits items	`subscribe()` - Emits items
Item passes through a set of operations	`pipe()` **through a set of operators**
As an observer - Next item, process it - Error occurred, handle it - Complete, you're done	**Observer** - `next()` - `error()` - `complete()`
Stop	`unsubscribe()`

Module Overview

Observer/Subscriber

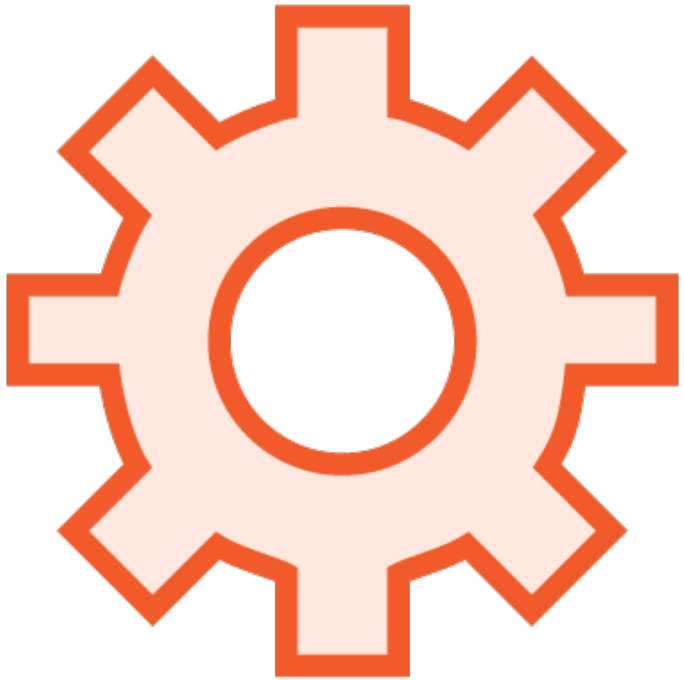
Observable

Subscribing

Unsubscribing

Creation functions

RxJS Features

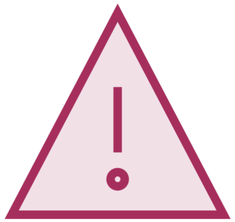


`of`

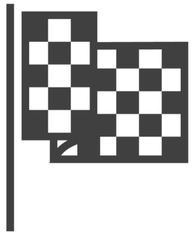
`from`

Observer

Next item, process it



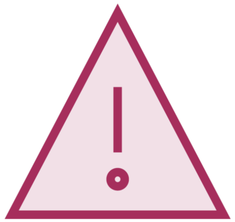
Error occurred, handle it



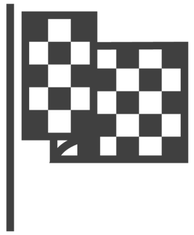
Complete, you're done

Observer

Next item, process it



Error occurred, handle it



Complete, you're done

Observer

`next()`

`error()`

`complete()`

Observes and responds to notifications

Observer

"a collection of callbacks that knows how to listen to values delivered by the Observable."

"An Observer is a consumer of values delivered by an Observable."

In RxJS, an Observer is also defined as an interface with next, error, and complete methods.

Observer

```
next()
error()
complete()
```

Observes and responds to notifications

Subscriber

Subscriber

```
next()
error()
complete()
```

Observer that can unsubscribe from an Observable

Observer

```
next()
error()
complete()
```

Observes and responds to notifications

Observer

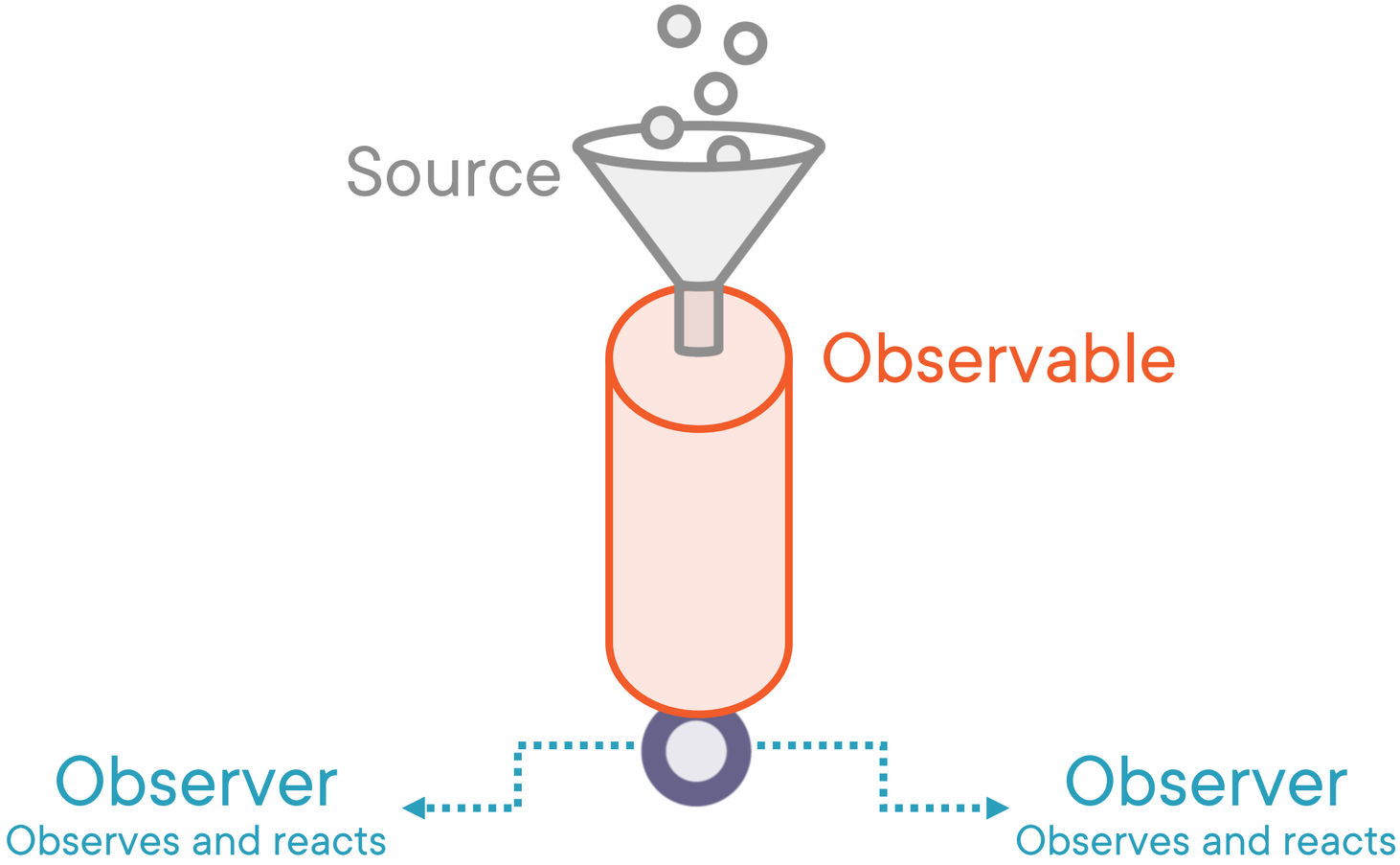
```
const observer = {
  next: apple => console.log(`Apple emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
};
```

Observable

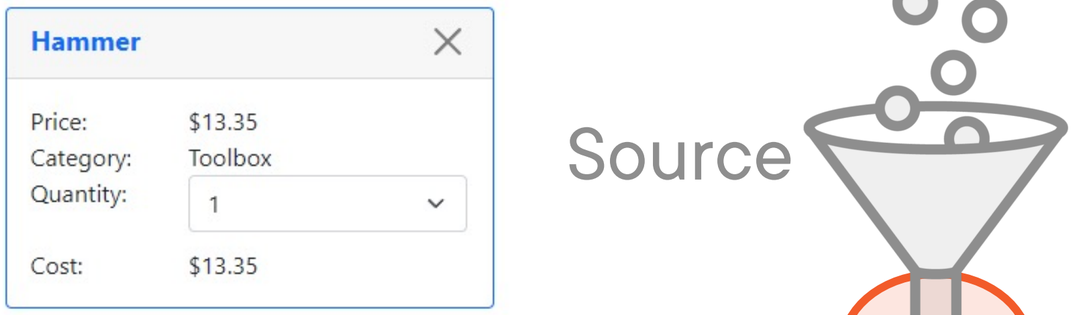
A collection of events or values emitted over time

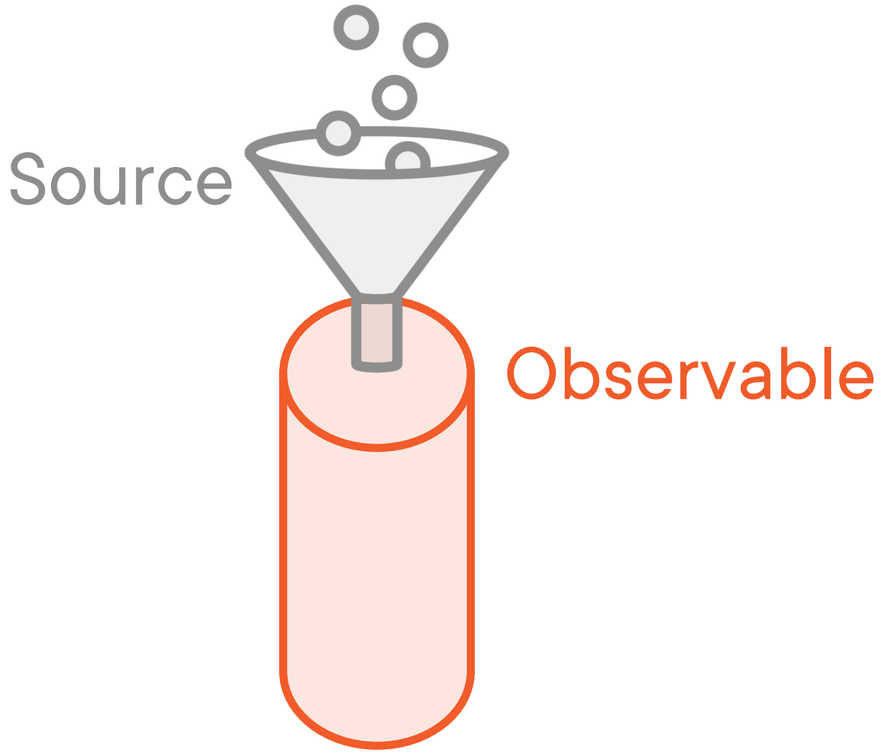
- User actions
- Application events (routing, forms)
- Response from an HTTP request
- Internal structures

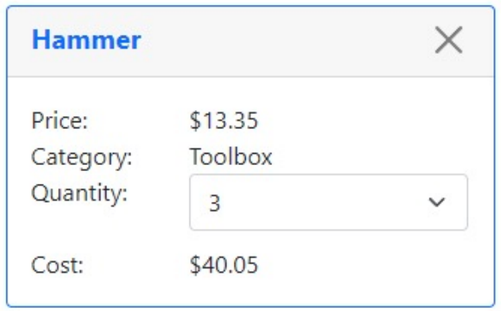
Observable/Observer



Observable/Observer

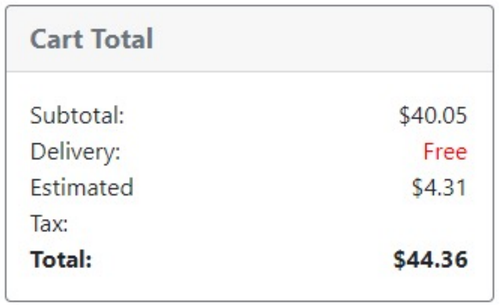






Observer
Observes and reacts

Observer
Observes and reacts



Observables Can Emit

Primitives: numbers, strings, dates

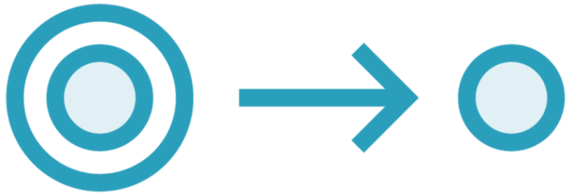
Events: mouse, key, valueChanges, routing

Objects: customers, products

Arrays

HTTP response

Observables



Synchronous

Asynchronous

[1, 2, 3]

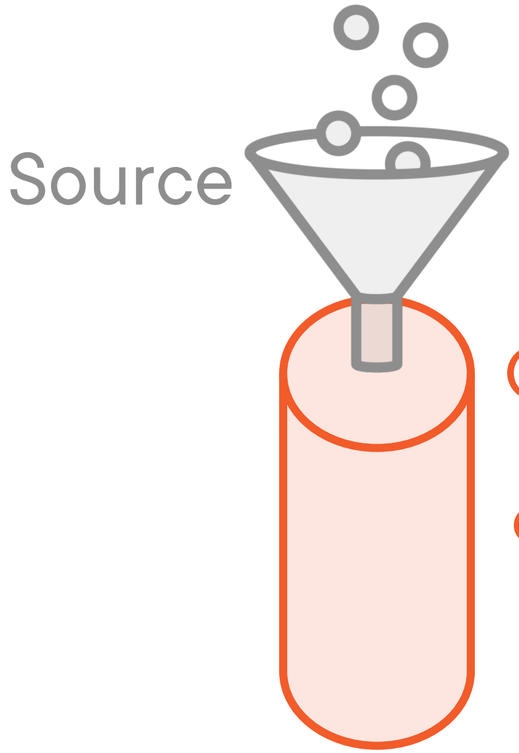
Finite emissions

Infinite emissions

Observable -> Collection of Items over Time

Array [a, b, c]

map, filter, concat



Observable

map, filter, concatMap, delay, timeout

Observer

Observes and reacts

Observer

Observes and reacts

Observable

```
const observer = {
  next: apple => console.log(`Apple emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
};
```

```
const apples$ = new Observable(appleSubscriber => {
  appleSubscriber.next('Apple 1');
  appleSubscriber.next('Apple 2');
  appleSubscriber.complete();
});
```

Subscribing

Start

Emits items

Item passes through a set of operations

As an observer

Next item, process it

Error occurred, handle it

Complete, you're done

Stop

Subscribing

Start

Emits items

Item passes through a set of operations

As an observer

Next item, process it

Error occurred, handle it

Complete, you're done

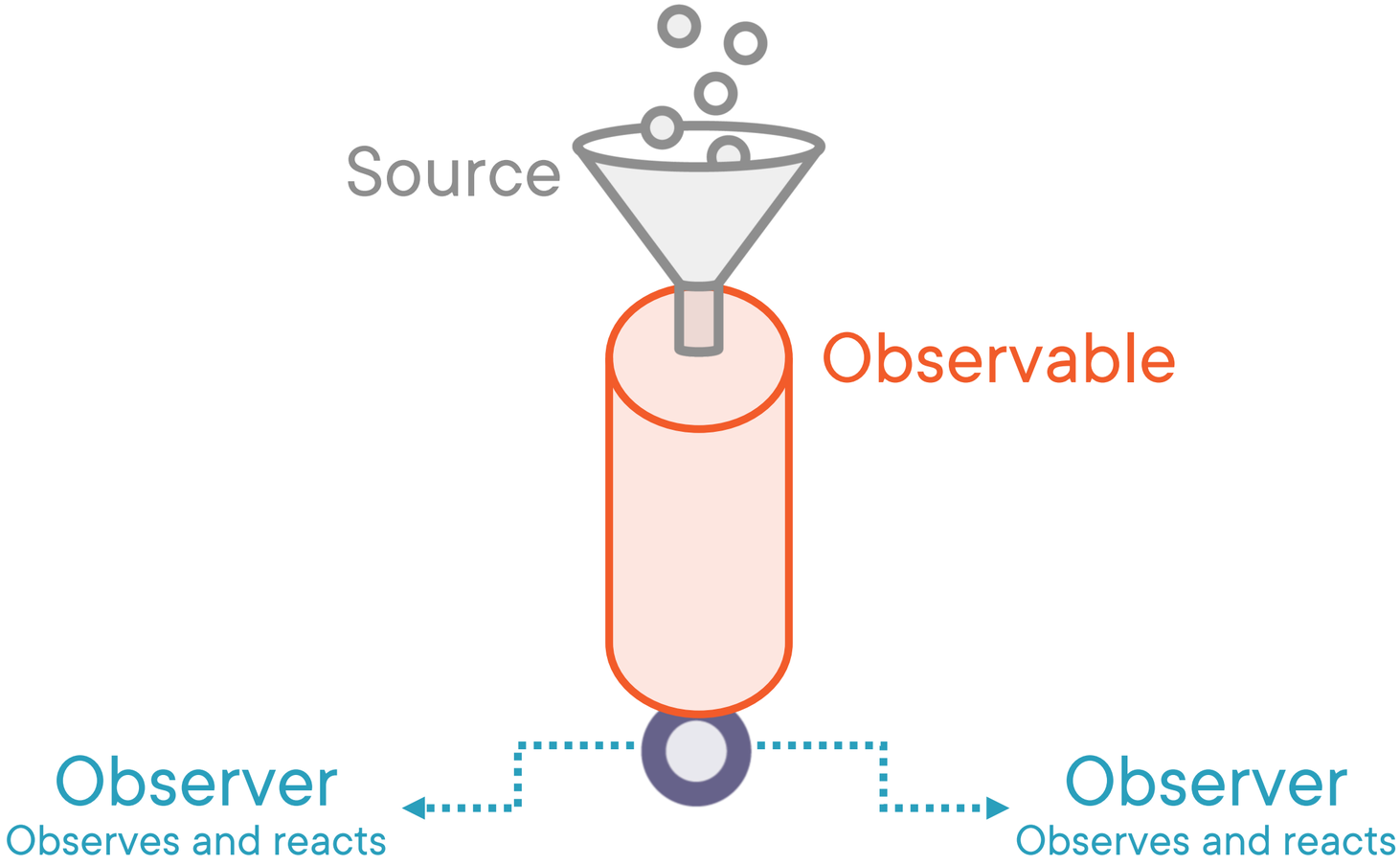
Stop

Call `subscribe()` on the Observable

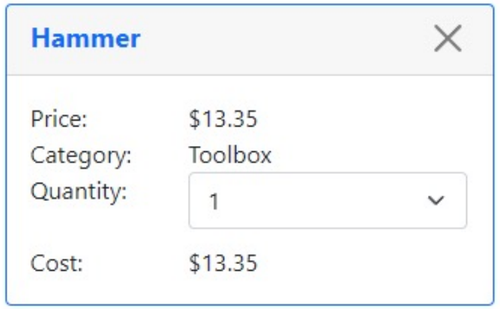
Pass in the Observer

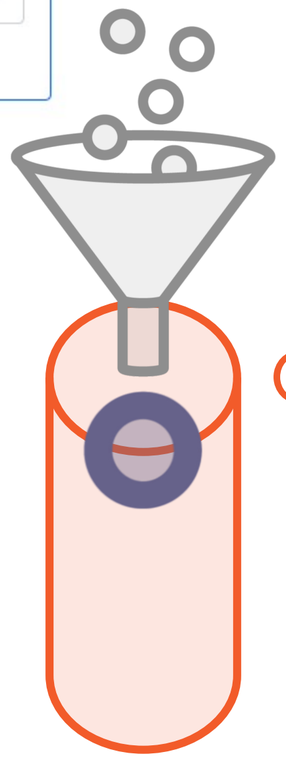
MUST subscribe to start receiving emissions

Subscribing



Subscribing



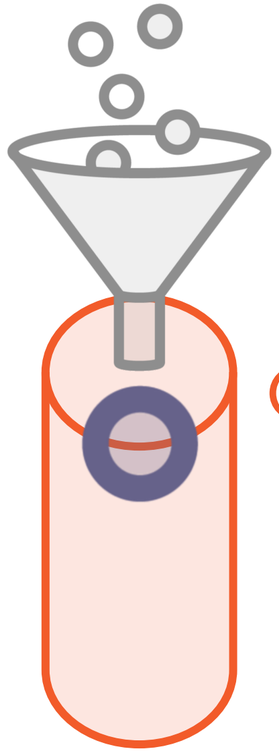


Observable

Observer
Observes and reacts

Observer
Observes and reacts

[a, b, c]



Observable

Observer
Observes and reacts

Observer
Observes and reacts

Subscribing

```
const apples$ = new Observable(appleSubscriber => {
  appleSubscriber.next('Apple 1');
  appleSubscriber.next('Apple 2');
  appleSubscriber.complete();
});
```

```
const observer = {
  next: apple => console.log(`Apple emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
};
```

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

Subscribing

```
const observer = {
  next: apple => console.log(`Apple emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
};
```

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

```
const sub = apples$.subscribe({
  next: apple => console.log(`Apple emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
});
```

Subscribing

```
const sub = apples$.subscribe({
  next: apple => console.log(`Apple emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
});
```

```
const sub = apples$.subscribe(
  apple => console.log(`Apple emitted ${apple}`)
);
```

Subscribing

```
const apples$ = new Observable(appleSubscriber => {
  appleSubscriber.next('Apple 1');
  appleSubscriber.next('Apple 2');
  appleSubscriber.complete();
});
```

```
const sub = apples$.subscribe({
  next: apple => console.log(`Apple emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
});
```

Unsubscribing

Start

Emits items

Item passes through a set of operations

As an observer

Next item, process it

Error occurred, handle it

Complete, you're done

Stop

Unsubscribing

Start

Emits items

Item passes through a set of operations

As an observer

Next item, process it

Error occurred, handle it

Complete, you're done

Stop

Every time we subscribe to start, we should unsubscribe to stop

Call `unsubscribe()` on the subscription

Stop Receiving Notifications

Call `complete()` on the subscriber

Use an operator that automatically completes

Throw an error

Call `unsubscribe()` on the subscription

Unsubscribing

```
const sub = apples$.subscribe({
  next: apple => console.log(`Apple emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
});
```

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

Properly unsubscribing from each Observable helps avoid memory leaks

Observable/Subscription/Observer

```
const apples$ = new Observable(appleSubscriber => {
  appleSubscriber.next('Apple 1');
  appleSubscriber.next('Apple 2');
  appleSubscriber.complete();
});
```

```
const sub = apples$.subscribe({
  next: apple => console.log(`Apple emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
});
```

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

In Angular, we often work with Observables that Angular creates for us.

Creating an Observable

```
const apples$ = new Observable(appleSubscriber => {
  appleSubscriber.next('Apple 1');
  appleSubscriber.next('Apple 2');
  appleSubscriber.complete();
});
```

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

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

# of vs. from

```
const apples = ['Apple 1', 'Apple 2'];
```

```
of(apples);
// [Apple1, Apple2]
```

```
from(apples);
// Apple1 Apple2
```

```
of(...apples);
// Apple1 Apple2
```

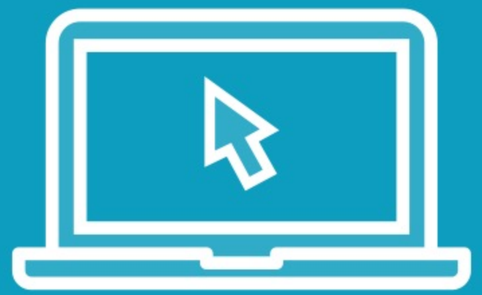
Creating an Observable

```
@ViewChild('para') par: ElementRef;

ngAfterViewInit() {
  const parStream = fromEvent(this.par.nativeElement, 'click')
                              .subscribe(console.log)
}
```

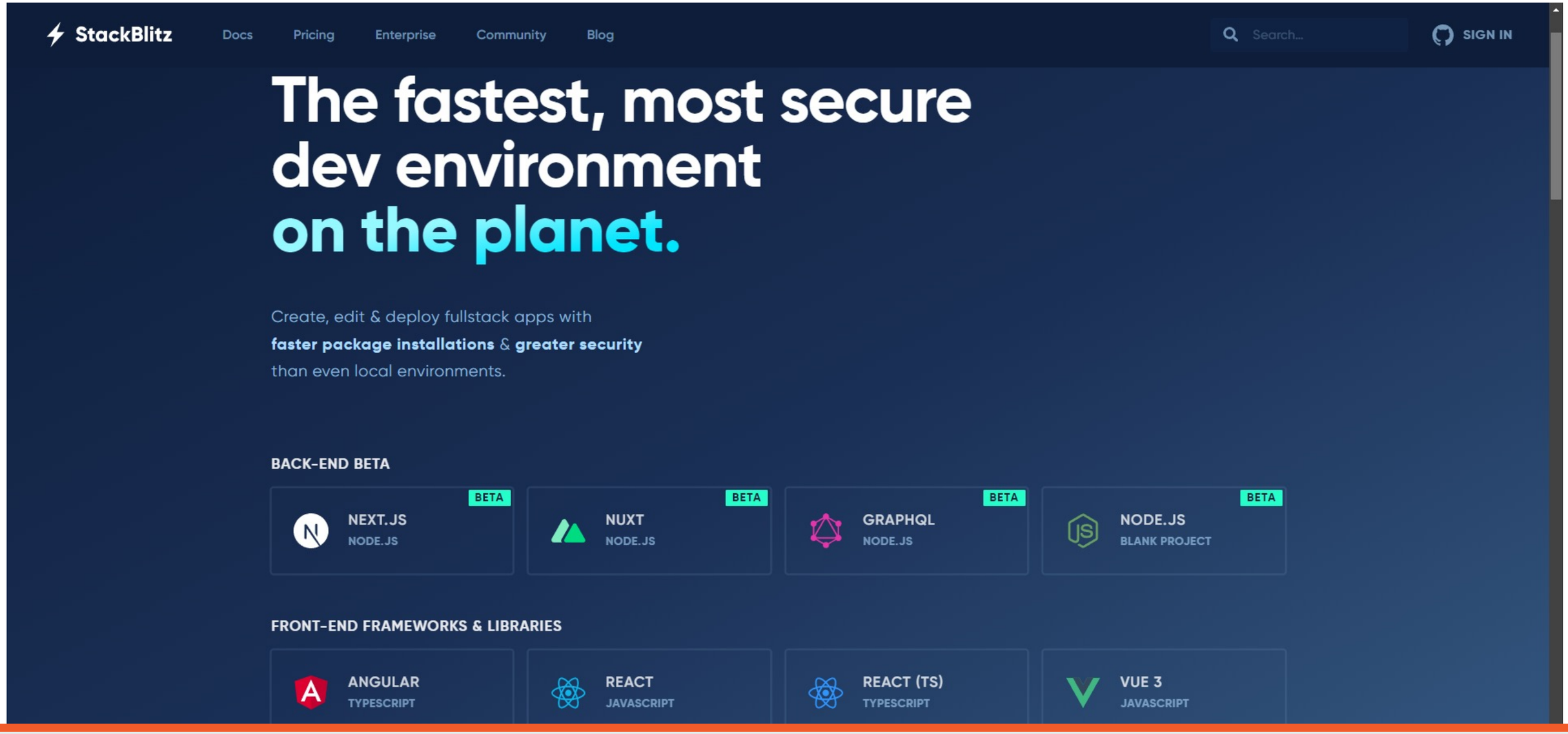
```
const num = interval(1000).subscribe(console.log);
```

Demo



Creation functions:

- `of`
- `from`



https://stackblitz.com

Checklist

Review module concepts

Provide additional use cases and tips

Code along assistance

Revisit as you build

RxJS Checklist: Terms

Observable

- Collection of events or values emitted over time

Observer

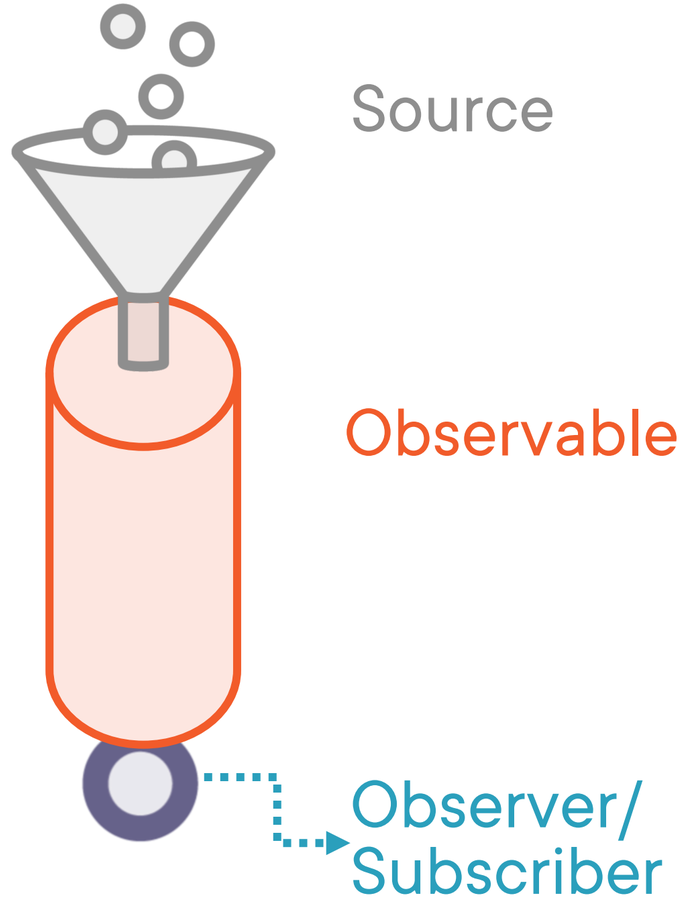
- Observes notifications from the Observable
- Methods to process notifications: `next()`, `error()`, `complete()`

Subscriber

- An Observer that can unsubscribe

Subscription

- Tells the Observable that the subscriber is ready for notifications
- `subscribe()` returns a Subscription
- Use Subscription to unsubscribe



RxJS Checklist: Creating an Observable

Constructor

Creation functions

- `of`, `from`, `fromEvent`, `interval`, ...
- Create an Observable from anything

Returned from an Angular feature

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

RxJS Checklist: Start Receiving Emissions

Call `subscribe()`

Pass in an Observer

- `next()`, `error()`, `complete()`

```
const sub = apples$.subscribe({
 next: apple => console.log(`Apple emitted ${apple}`),
 error: err => console.log(`Error occurred: ${err}`),
 complete: () => console.log(`No more apples, go home`)
});
```

RxJS Checklist: Stop Receiving Emissions

Use a creation function that completes

- `of`, `from`, ...

Use an operator that completes

- `take`, ...

Throw an error

Call `unsubscribe()` on the Subscription

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

Coming up next:

RxJS Operators