Reactive Programming -

Observables

RxJs - Observables

Synchronous programming, one line of your code is executed after the previous one

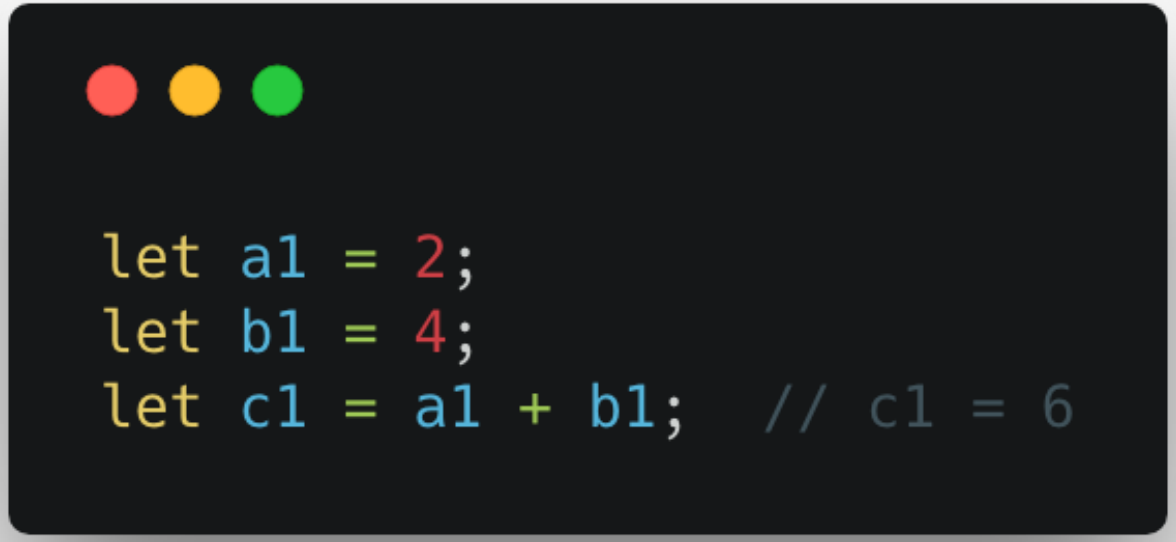
Asynchronous programming dramatically increases code complexity

RxJS - Observables

RxJS , any JavaScript-based apps

Angular uses **RxJS library** internally

RxJs - Observables



RxJs - Observables

```
let a1 = 2;
let b1 = 4;
let c1 = a1 + b1;  // c1 = 6

a1 = 55;          // c1 = 6 but should be 59
b1 = 20;          // c1 = 6 but should be 75
```

RxJs - Observables

In the **reactive** style of coding
(as opposed to **imperative** one)

changes in data **drive**
the invocation of your code

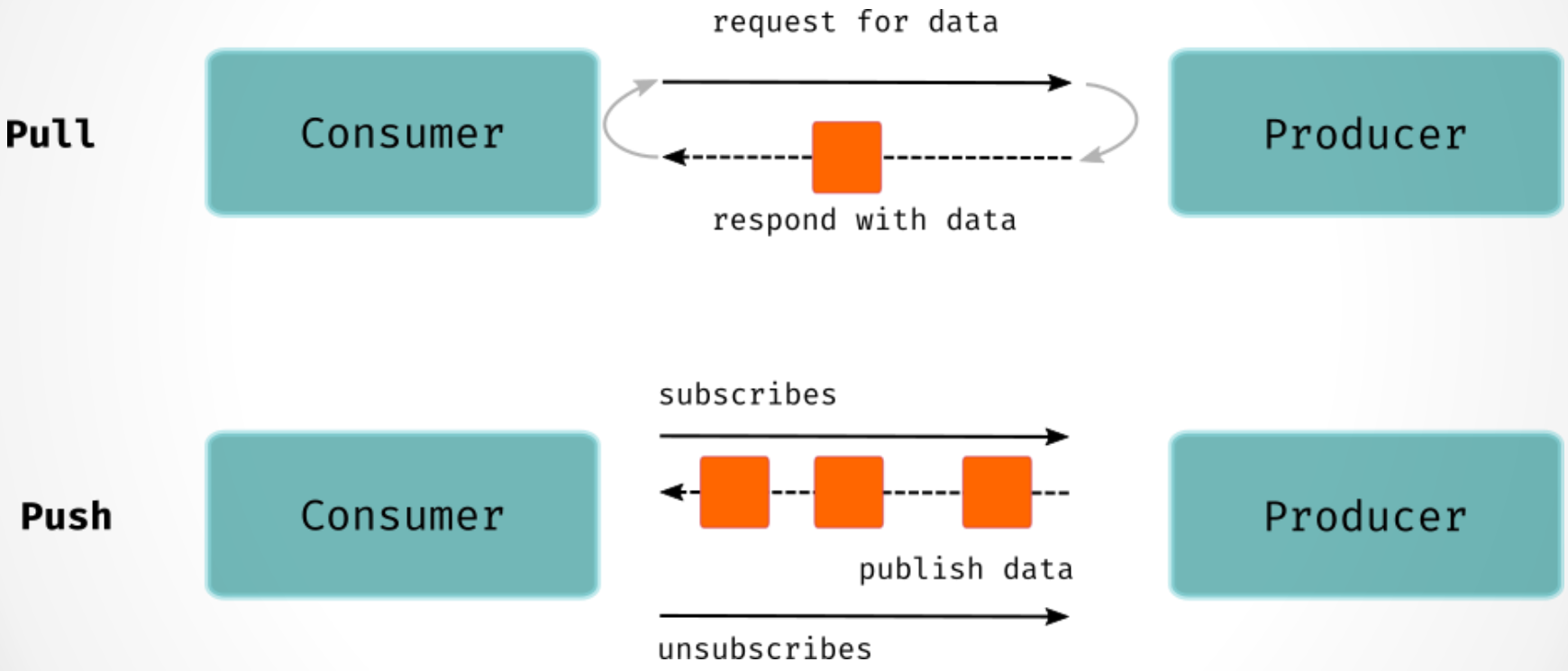
RxJs - Observables

Reactive programming is about creating **responsive event-driven** applications

observable event **stream** is pushed to **subscribers**

subscribers **observe** and **handle the events**

Pull vs Push


Pull
Consumer
request for data
respond with data
Producer
Push
Consumer
subscribes
publish data
unsubscribes
Producer

RxJs - Observables

An **observable** is a **function (or an object)** on the client that **gets the producer's data** and **pushes** them to the **subscriber(s)**

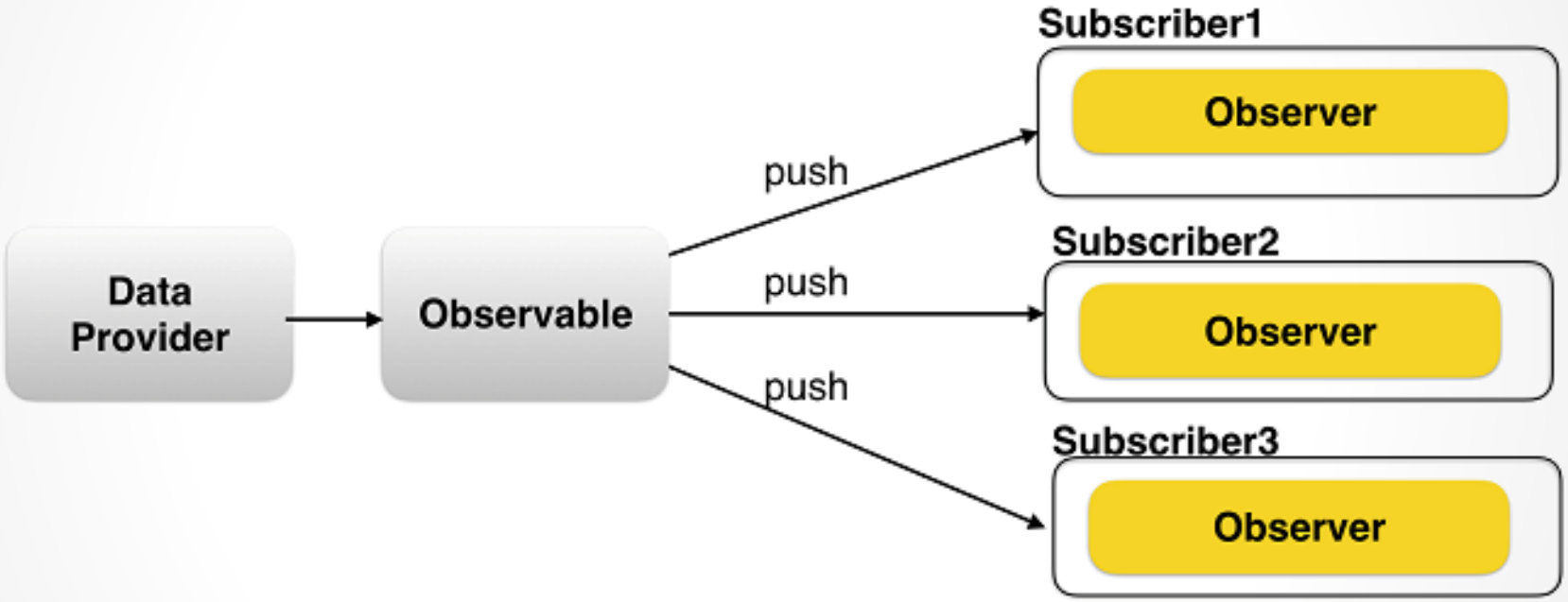
RxJs - Observables

An **observer** is an **object (or a function)** that knows how to **handle the data** elements pushed by the **observable**

RxJs - Observables

- **Observable** - data stream that pushes data over time
- **Observer** - consumer of an observable stream
- **Subscriber** - connects observer with observable

RxJs - Observables



RxJs - Observables

Observable:

- **Emit** the **next element** to the observer
- **Throw an error** on the observer
- Inform the observer that the **stream is over**

RxJs - Observables

Observer:

- The **function to handle** the next **element emitted** by the observable
- The **function to handle errors** thrown by the observable
- The **function** to be handle the **end of stream**

RxJs - Observables

subscriber connects an **observable** and **observer** by invoking the method **subscribe()** and **disconnects** them by invoking **unsubscribe()**

CREATING OBSERVABLES

Observable.of(1,2,3) - turns the sequence of "elements" into an Observable

Observable.create(myObserver) - returns an Observable that can invoke methods on myObserver that you will create and supply as an argument

CREATING OBSERVABLES

Observable.from(myArray) - converts an array(iterable, array-like, ...) into an Observable

Observable.fromEvent(myInput, 'keyup') - converts the keyup event from some HTML element represented by myInput into an Observable.

Observable.interval(1000) - emits a sequential integer (0,1,2,3...) every second

RxJs – Observables Dependencies

```
// RxJS 6

// creation and utility methods
import { Observable, interval, of, fromevent, pipe } from 'rxjs';
// operators all come from `rxjs/operators`
import { map, take, filter } from 'rxjs/operators';
```

RxJs - Observables

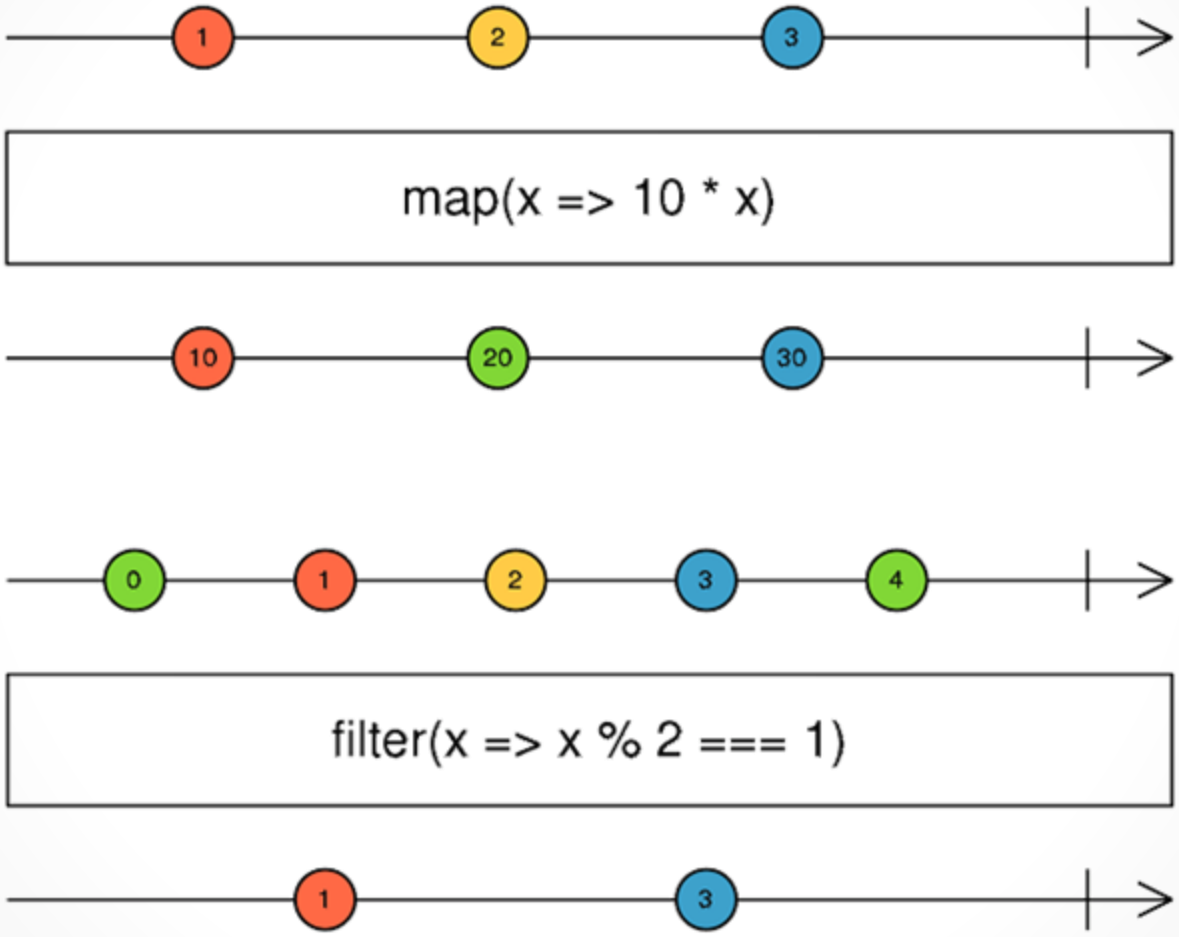
```
Rx.Observable.of(1,2,3)
     .subscribe(
          value => console.log(value),
          err => console.error(err),
          () => console.log("Streaming is over")
);

/// result:
1
2
3
Streaming is over
```

Exemplos…

Operators...

Operators...



```
// RxJS 6

// creation and utility methods
import { Observable, interval, of, fromevent, pipe } from 'rxjs';
// operators all come from `rxjs/operators`
import { map, take, filter } from 'rxjs/operators';
```

```
let beers = [
    {name: "Stella", country: "Belgium", price: 9.50},
    {name: "Sam Adams", country: "USA", price: 8.50},
    {name: "Bud Light", country: "USA", price: 6.50},
    {name: "Brooklyn Lager", country: "USA", price: 8.00},
    {name: "Sapporo", country: "Japan", price: 7.50}
];

from(beers)
    .pipe(
        filter(beer => beer.price < 8) ,
        map(beer => beer.name + ": $" + beer.price)
    ).subscribe(
        beer => console.log(beer),
        err => console.error(err),
        () => console.log("Streaming is over")
);

console.log("This is the last line of the script");
```

Observables Operators..

RXMARBLES.com