
plain concepts

# PROGRAMACIÓN REACTIVA

- *Paradigma de programación*
- *Trabajamos con flujos de datos finitos o infinitos de manera asíncrona.*
- *Escuchamos los cambios en esos flujos de datos y reaccionamos a ellos, como si fuesen un evento.*

# PROGRAMACIÓN FUNCIONAL

- *Patrón de programación*
- *Todo está orientado al uso de funciones.*
- *Podemos concatenar funciones y pasar funciones como argumento.*

RXJS 

*“RxJS es una librería de programación reactiva cuyo fin es simplificar la composición de códico asíncrono basado en eventos a traves de secuencias observables”*

https://rxjs-dev.firebaseapp.com/guide/overview

# OBSERVABLES

- Estructura de datos básica de RxJS.

```
interface Observable {
  subscribe(observer: Observer): Subscription
}
interface Observer {
  next(v: any): void;
  error(e: Error): void;
  complete(): void;
}
interface Subscription {
  unsubscribe(): void;
}
```

# OBSERVABLES: ANALOGÍA

```
const myTurnObservable = interval(1000);
const myTurn: number = 5;
const isMyTurn = myTurnObservable.pipe(filter(t => t > myTurn));
//filter emite valores que cumplan la condicion

const subscription = myTurnObservable
.pipe(takeUntil(isMyTurn))
.subscribe({
  next: (x: number) => { console.log("Next number is " + x);
},
  error: (err: Error) => console.error('Observer got an error: ' + err),
  complete: () => console.log("Completed")
})
```


https://stackblitz.com/edit/rxjs-uoadhs

# RXJS – CREAR OBSERVABLES

```
  //emit array as a sequence of values
const arraySource = from([1, 2, 3, 4, 5]);
//output: 1,2,3,4,5
const subscribe = arraySource.subscribe(val => console.log(val));

const ofSource = of(1, 2, 3, 4, 5);
//output: 1,2,3,4,5
ofSource.subscribe(val => console.log(val));

const intervalSource = interval(1000);
//output: 1,2,3,4,5....
intervalSource.subscribe(val => console.log(val));
```

# RXJS – CREAR OBSERVABLES

```
const observable = new Observable(function subscribe(subscriber) {
  const id = setInterval(() => {
    subscriber.next('hi')
  }, 1000);
});
```

```
const githubUsers = `https://api.github.com/users?per_page=2`;

const users = ajax(githubUsers);

const ajaxSubscription = users.subscribe({
  next: (res) => console.log(res),
  error: (error) => console.log(error),
  complete: () => console.log("Completed")
});
```

# RXJS – CREAR OBSERVABLES

```
//create observable that emits click events
const source = fromEvent(document, 'click');
//map to string with given event timestamp
const example = source.pipe(map(event => `Event time: ${event.timeStamp}`));
//output (example): 'Event time: 7276.390000000001'
example.subscribe(val => console.log(val));
```

# OBSERVABLE

- *Cada ejecución es exclusiva a un único observer. (unicast).*
- *Si un observable emite un error o un completed, se detiene el flujo.*

```
const myObservable = of(1, 2, 3);

const myObservable2 = throwError(() => new Error());

const myObservable3 = new Observable<number>((observer) => {
  observer.next(1);
  observer.next(2);
  observer.next(3);
  observer.error('err');
  observer.complete();
});
```

```
myObservable3.subscribe({
 next: (x: number) => console.log(x),
 error: (err) => console.log('Error'),
 complete: () => console.log('Completed')
});
```

# OBSERVERS

```
const observer = {
  next: (x: any) => console.log('Observer got a next value: ' + x),
  error: (err: Error) => console.error('Observer got an error: ' + err),
  complete: () => console.log('Observer got a complete notification'),
};

myObservable.subscribe(observer);
```

# SUBSCRIPTION

- Un observable no hace nada hasta que nos suscribimos a el.
- Al subscribirnos quedamos *ligados a la ejecución, cuando llegan nuevos valores nos enteramos*
- *Los flujos de datos pueden ser infinitas, necesitamos des-suscribirnos para pararla.*

```
const infiniteObservable = interval(1000);
const mySubscription = infiniteObservable.subscribe({
  next: (x: number) => console.log('Observer got a next value: ' + x),
  error: (err: Error) => console.error('Observer got an error: ' + err),
  complete: () => console.log('Observer got a complete notification'),
});
// later
mySubscription.unsubscribe();
```

# SUBJECT

- *Multicast.*
- *Utilizamos los Subjets para enviar datos a varios observers.*
- *Funcionan como los Observables.*

```
const subject = new Subject<number>();

subject.subscribe({
  next: (v) => console.log(`observerA: ${v}`),
});
subject.subscribe({
  next: (v) => console.log(`observerB: ${v}`),
});

subject.next(1);
subject.next(2);

// Logs:
// observerA: 1
// observerB: 1
// observerA: 2
// observerB: 2
```

# BEHAVIOURSUBJECT

- *Es una variante de Subject.*
- *Sabe cual es el valor actual*
- *Cuando un observer se subscribe, recibe el último valor*

# BEHAVIOURSUBJECT

```
const behaviourSubject = new BehaviorSubject(0); // 0 is the initial value
behaviourSubject.subscribe({
  next: (v) => console.log(`observerA: ${v}`),
});

behaviourSubject.next(1);
behaviourSubject.next(2);

behaviourSubject.subscribe({
  next: (v) => console.log(`observerB: ${v}`),
});

behaviourSubject.next(3);
```

```
// Logs
// observerA: 0
// observerA: 1
// observerA: 2
// observerB: 2
// observerA: 3
// observerB: 3
```

# RXJS - OPERADORES

- *Dos tipos:*
  - *Creación*
  - *Pipeables*

| AREA | OPERATORS |
| --- | --- |
| Creation | `from`,`fromEvent`, `of` |
| Combination | `combineLatest`, `concat`, `merge`, `startWith` , `withLatestFrom`, `zip` |
| Filtering | `debounceTime`, `distinctUntilChanged`, `filter`, `take`, `takeUntil` |
| Transformation | `bufferTime`, `concatMap`, `map`, `mergeMap`, `scan`, `switchMap` |
| Utility | `tap` |
| Multicasting | `share` |

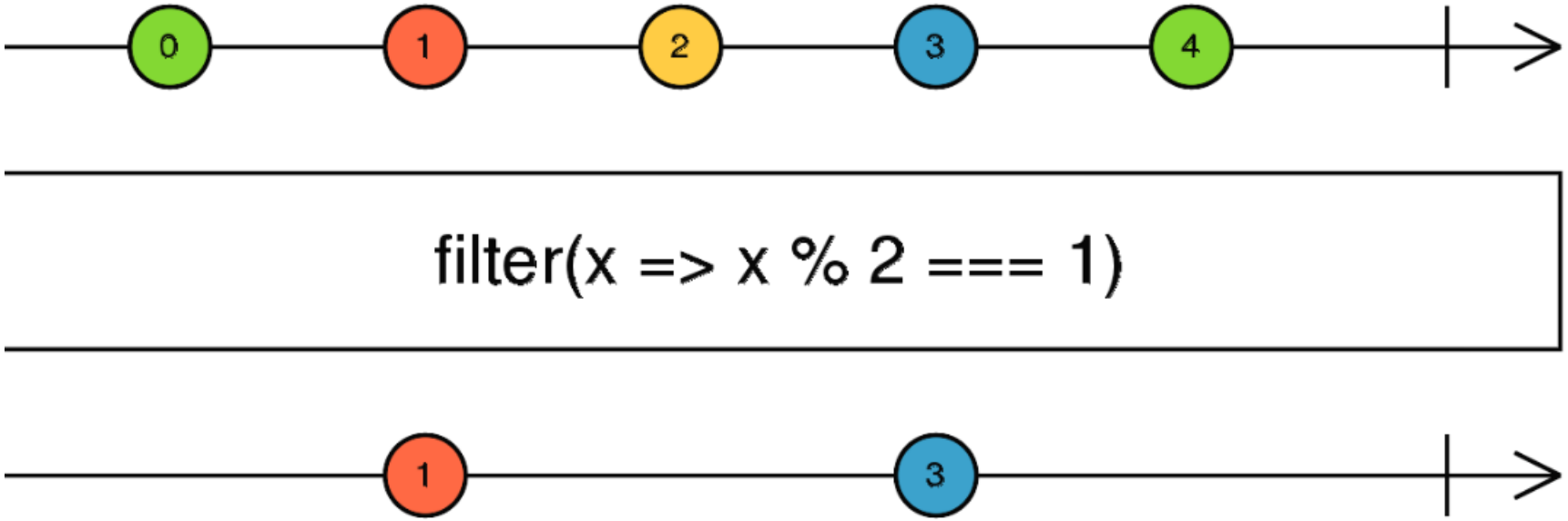
# PIPING

- *Podemos concatenar operadores pipeables*

```
myObservable.pipe(op1(), op2(), op3(), op4());
```

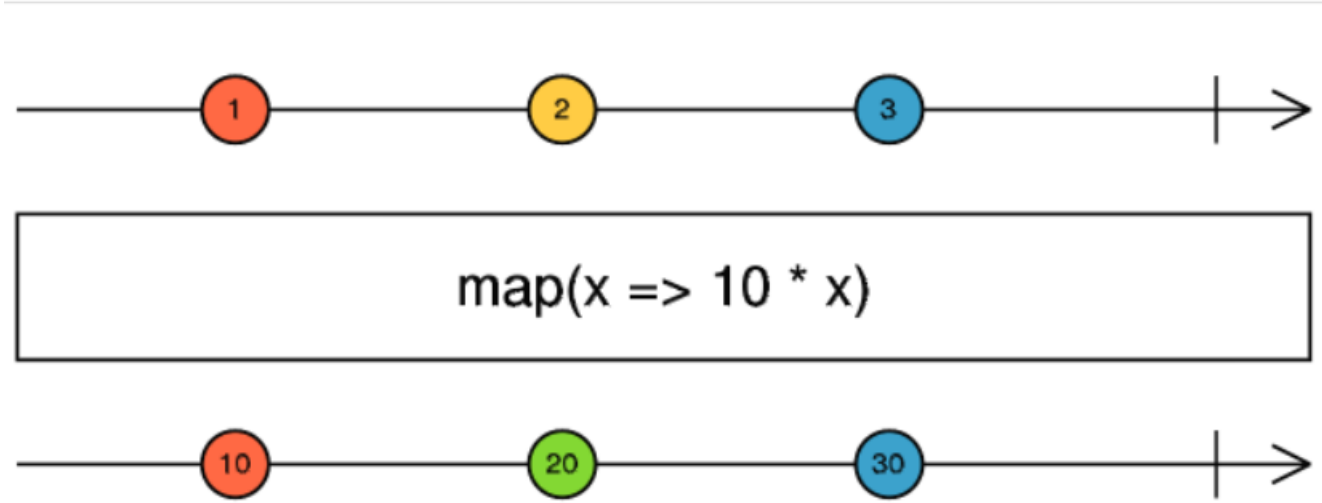
```
const squareOdd = of(1, 2, 3, 4, 5)
  .pipe(
    filter(n => n % 2 !== 0),
    map(n => n * n)
  );
```

# FILTER



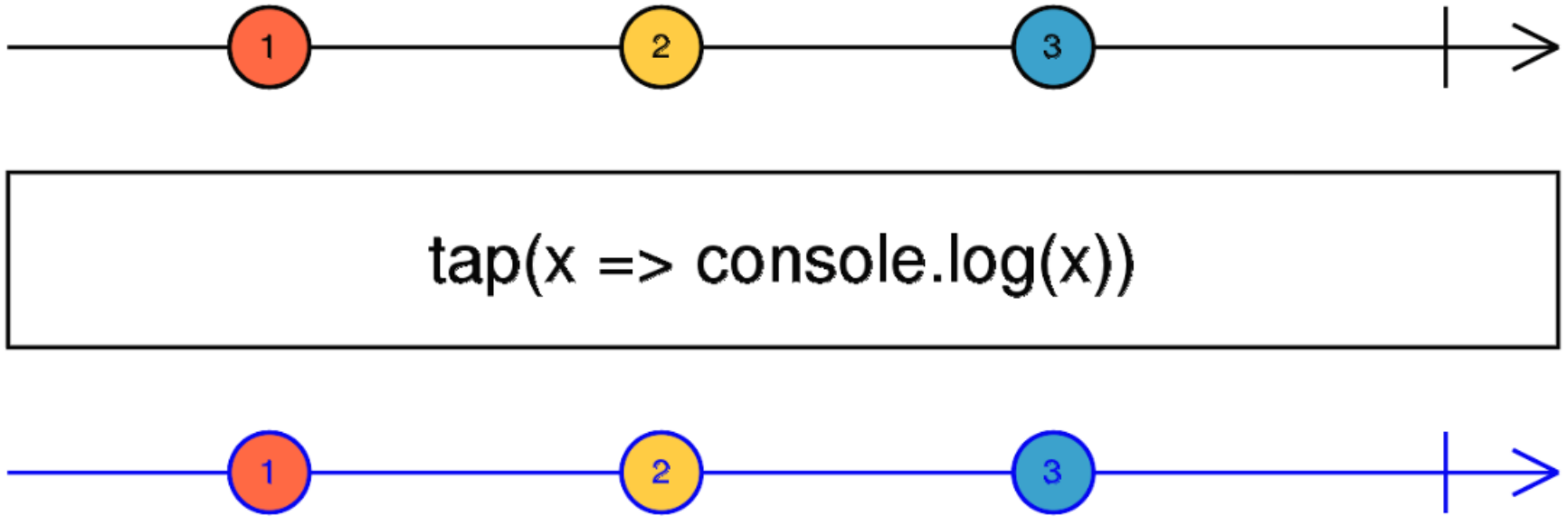

https://rxjs.dev/api/operators/filter

# MAP



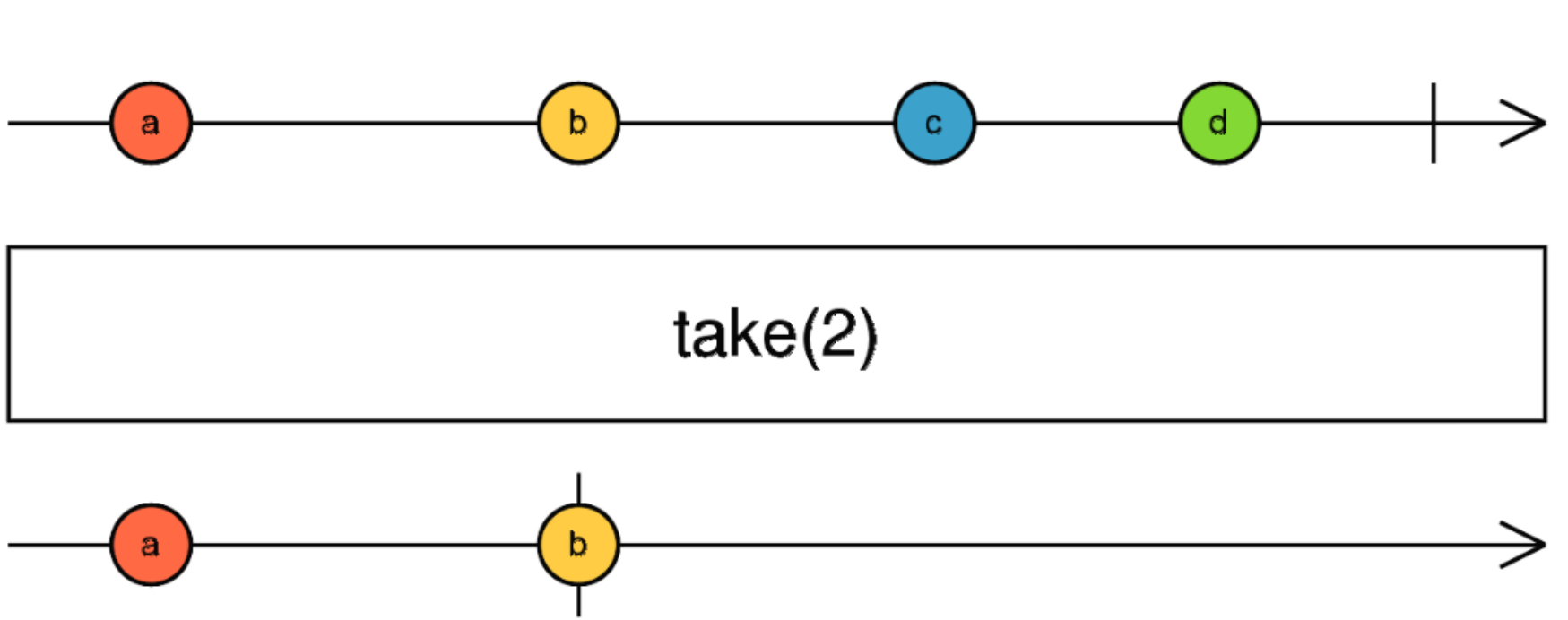

https://rxjs.dev/api/operators/map

# TAP



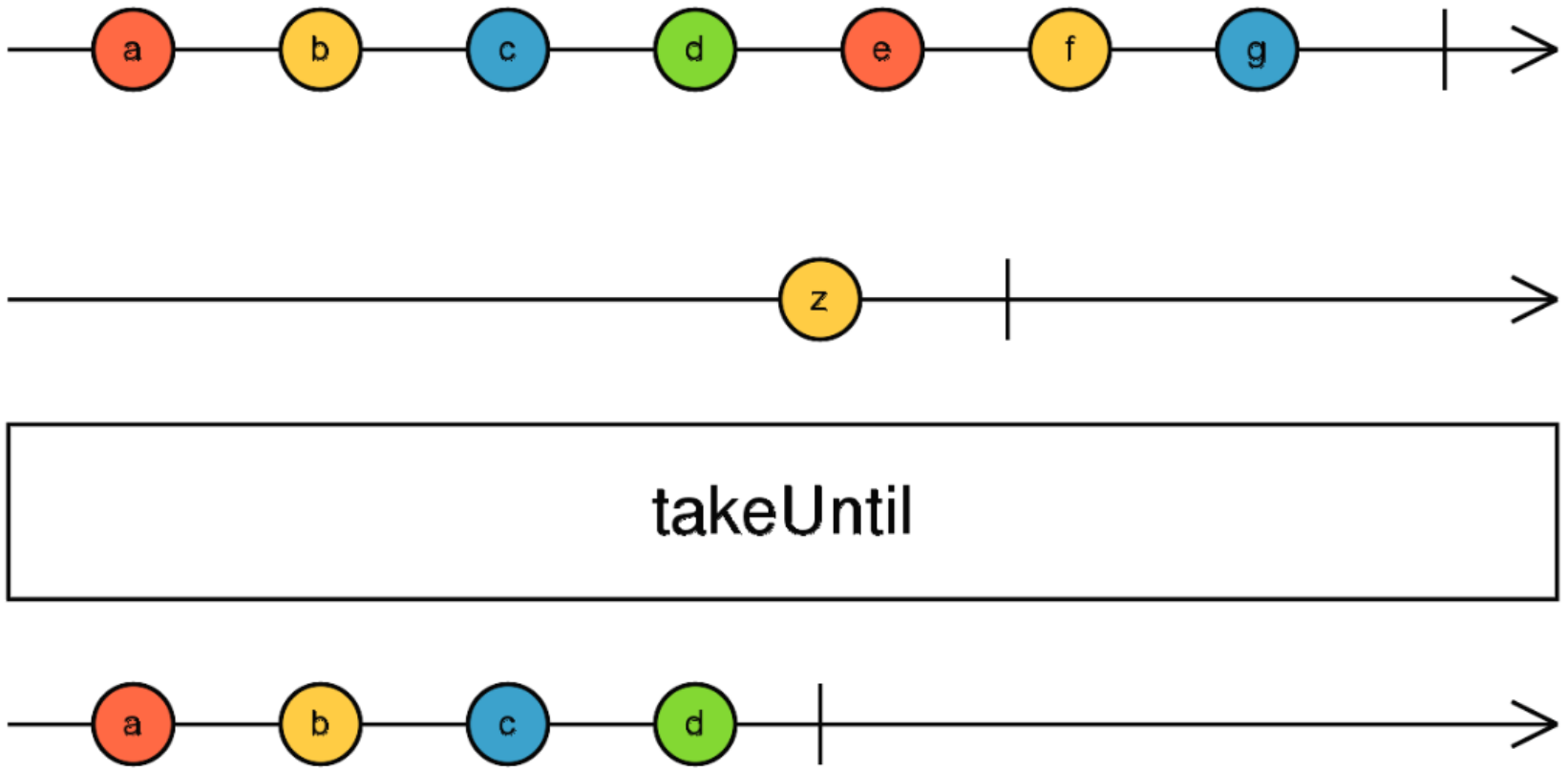

https://rxjs.dev/api/operators/tap

# TAKE



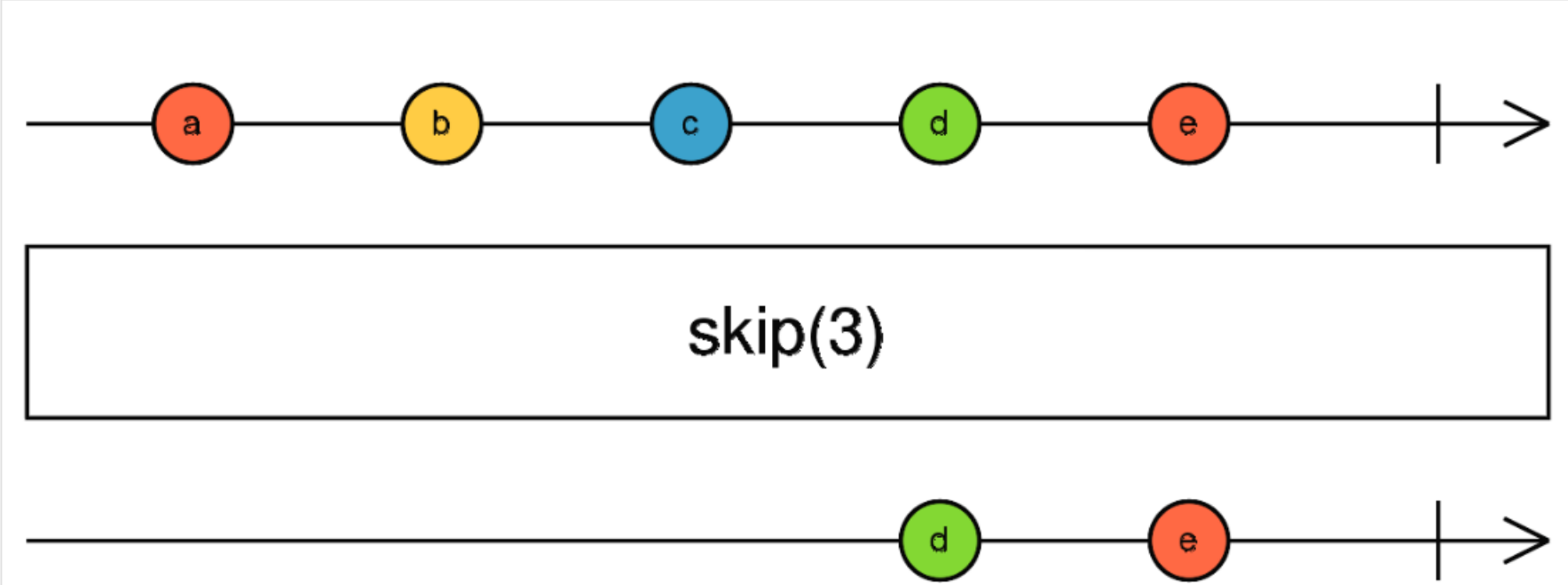

https://rxjs.dev/api/operators/take

# TAKEUNTIL



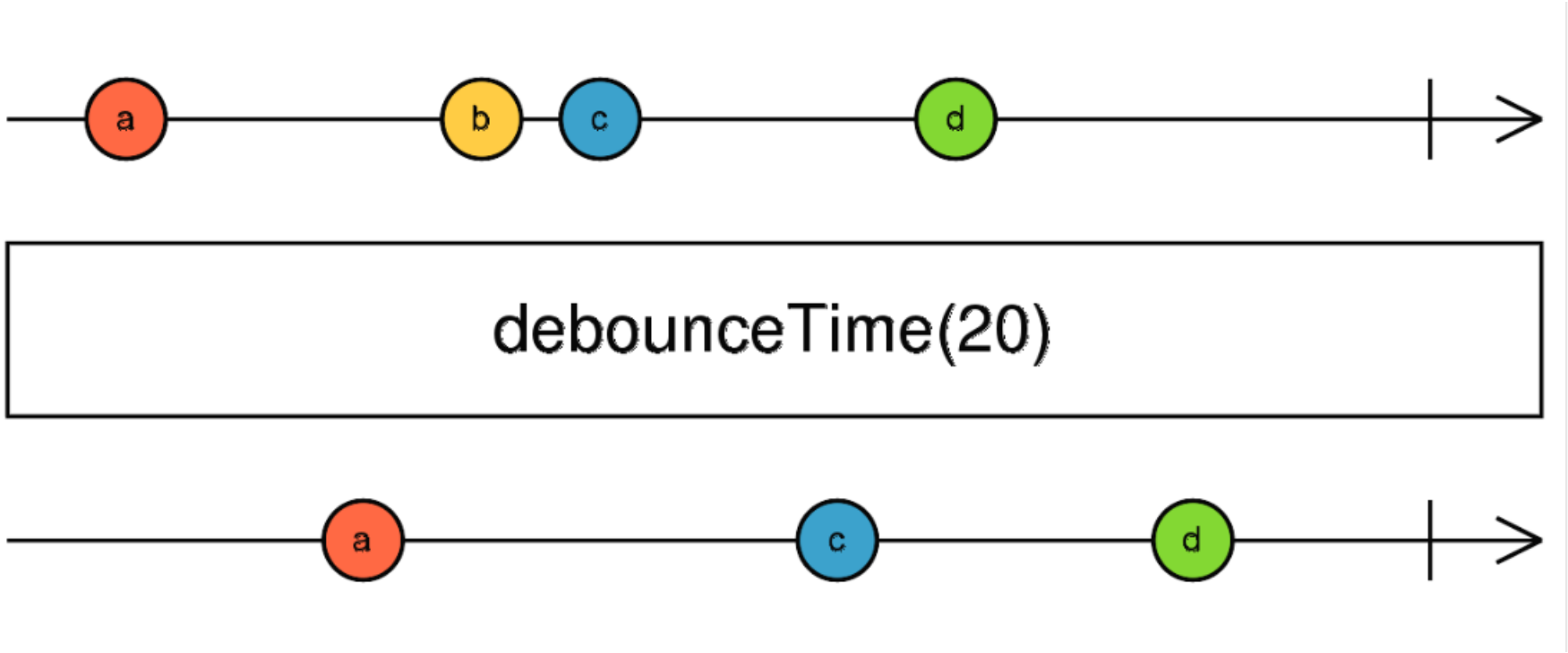

https://rxjs.dev/api/operators/takeUntil

# SKIP



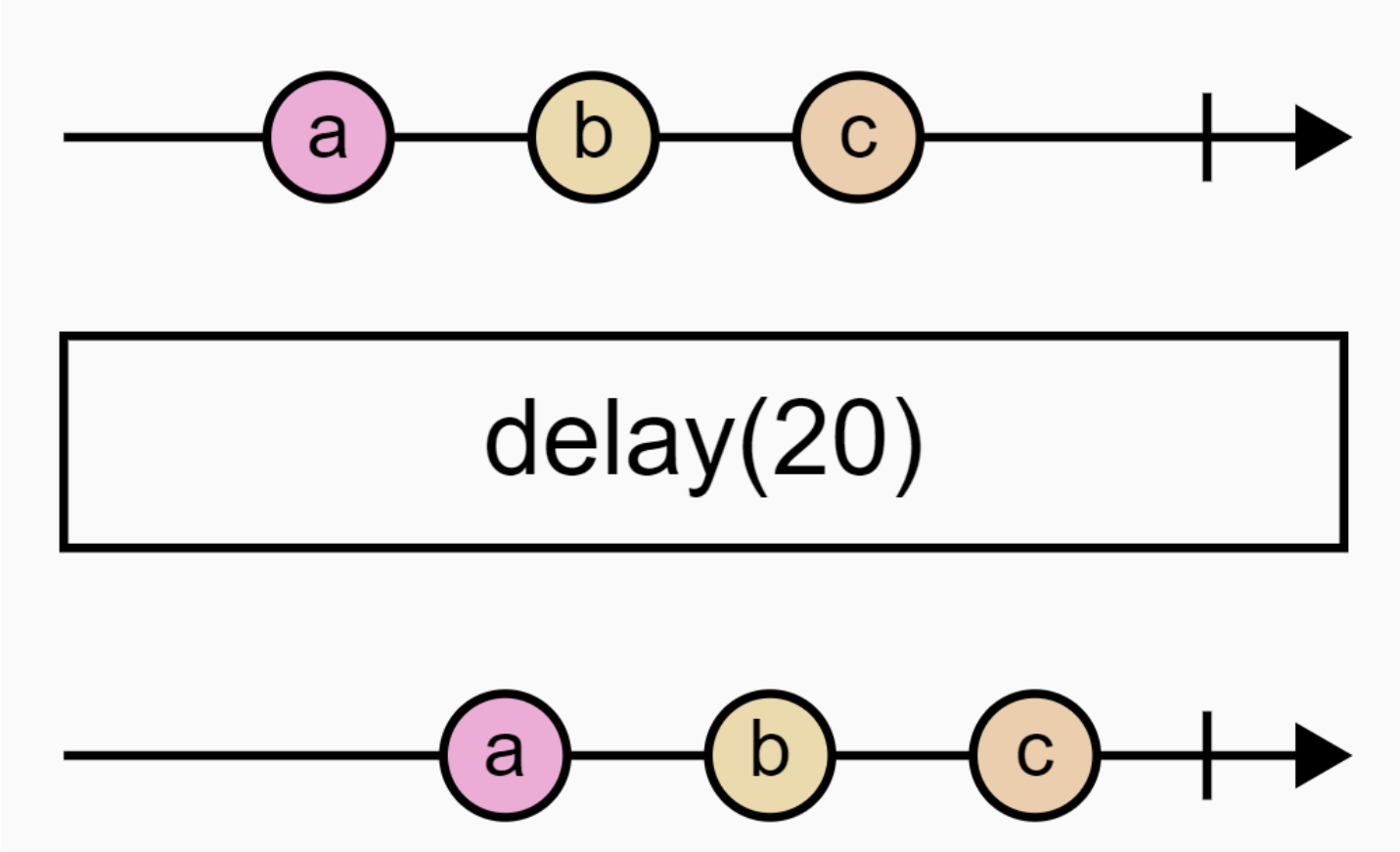

https://rxjs.dev/api/operators/skip

# DEBOUNCETIME




https://rxjs.dev/api/operators/debounceTime

# DELAY




https://rxjs.dev/api/operators/delay

# OBSERVABLES EN ANGULAR

- Angular utiliza observables para manejar operaciones asíncronas:
  - Enviar información entre componentes
  - Módulo HTTP usa observables para manejar las peticiones a servidor
  - Router y Forms para escuchar y responder a los eventos del usuario

- Angular trae por defecto RxJS
- Podemos usar los métodos de RxJS con los observables de Angular

# OBSERVABLES COMUNICACIÓN COMPONENTS

```
export class BroadcastService {

  public countSubject: Subject<number> = new Subject<number>();

  constructor() { }

  public changeCount(count: number){
    this.countSubject.next(count);
  }
}
```

```
ngOnInit(): void {
  this.broadcastService.countSubject
  .subscribe((count: number) => this.count = count);
}
```

```
  this.broadcastService.changeCount(this.count + 1);
```

# OBSERVABLES HTTP

```
public ngOnInit(): void {
  this.getAllSub = this.heroesService.getAll()
    .subscribe((heroes: Hero[]) => {
      this.heroes = heroes;
    });
}

public ngOnDestroy(): void {
  this.addSub?.unsubscribe();
  this.getAllSub?.unsubscribe();
  this.deleteSub?.unsubscribe();
}
```

```
this.heroService.delete(hero).subscribe({
  next: (res) => {
  this.heroes = this.heroes.filter(h => h !== hero);
  if (this.selectedHero === hero) {
    this.selectedHero = null;
  }
}, error: (error) => (this.error = error)
});
```

# OBSERVABLES ROUTING

```
this.router.events.subscribe(e => {
  if (e instanceof NavigationStart) {
    console.log('starts =>', e.url);
  }
  if (e instanceof NavigationEnd) {
    console.log('ends =>', e.url);
  }
});
```

```
this.route.params.subscribe(params => {
  this.product = this.products.find(p => p.id === Number(params.id));
});
```

```
this.activatedRoute.paramMap.subscribe(params => {
  this.id = +params.get('id');
});
```

# OBSERVABLES FORMULARIOS

```
this.heroForm.controls['name'].valueChanges
.pipe(debounceTime(500))
.subscribe( value => {
  console.log(value);
})
```

# ¡GRACIAS!

plain concepts

Barcelona

Bilbao

Madrid

Sevilla

Dubai

London

Seattle

plain concepts Bilbao

afole@plainconcepts.com

Ledesma 10 bis, 2ª planta

48001 Bilbao, España

+34 94 6008 168