

# RxJS : les clefs pour comprendre les observables

REVOLUTION

Thierry Chatel @ThierryChatel

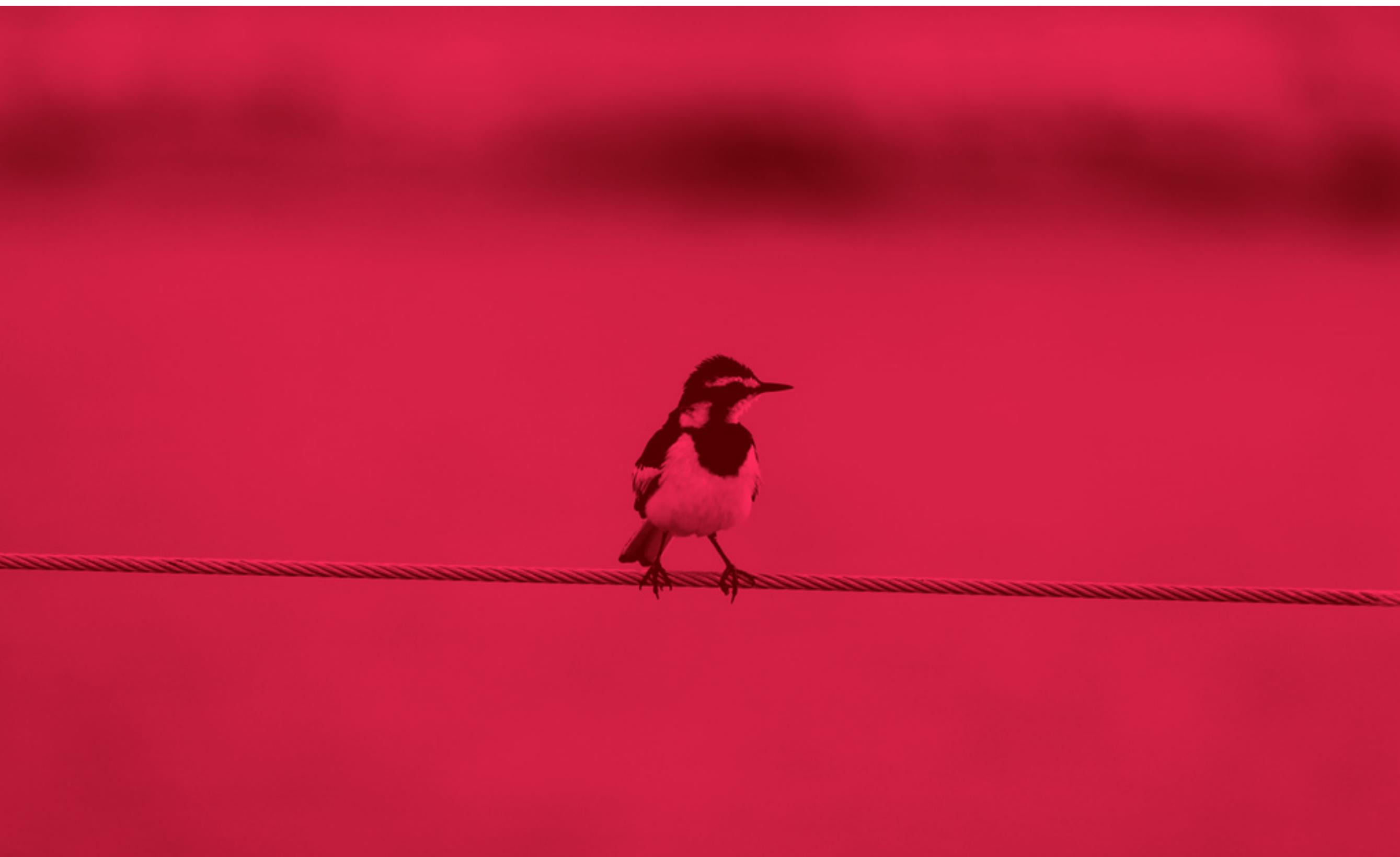
# RxJS : les clefs pour comprendre les observables

## 1. observables

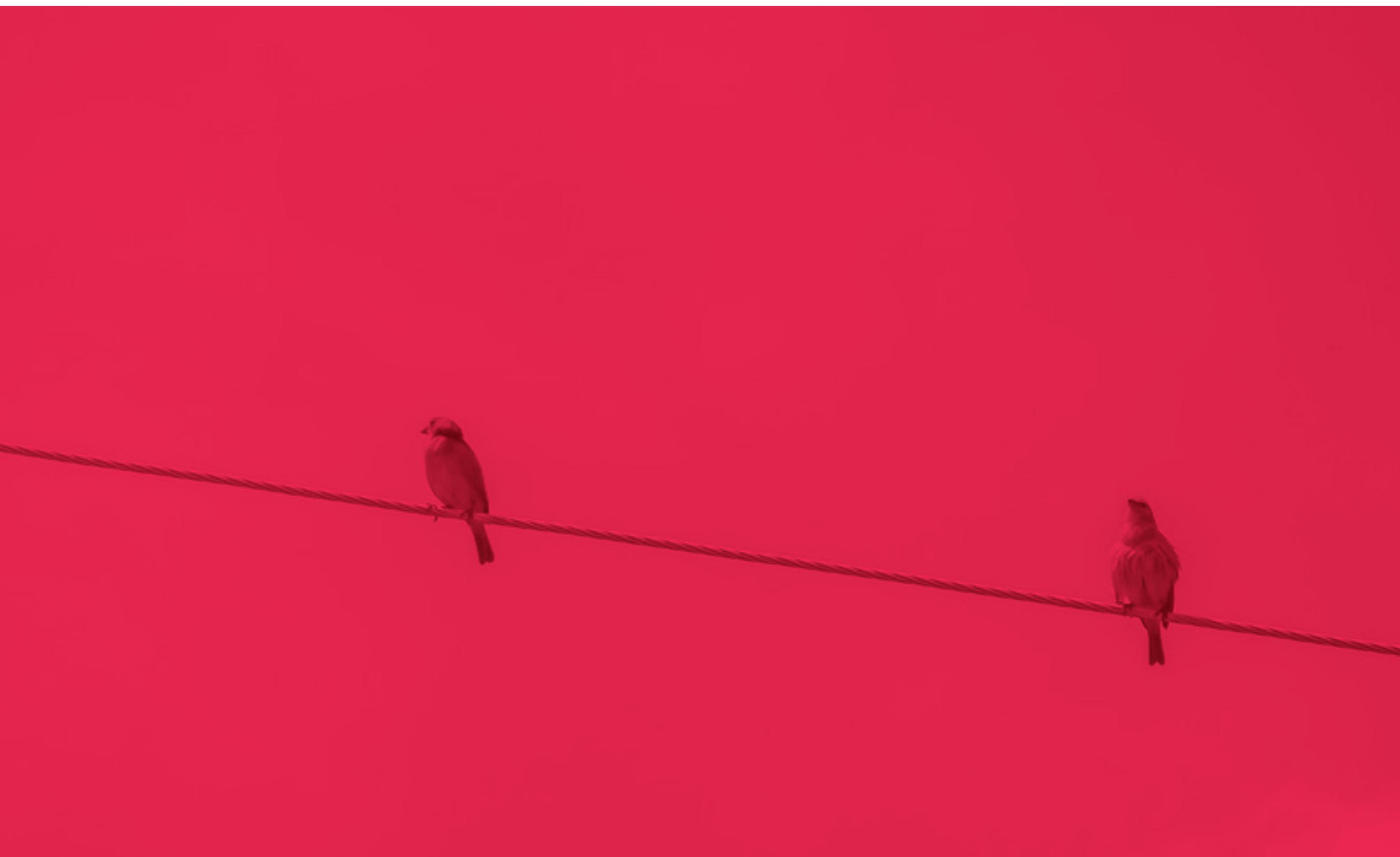
# Observable



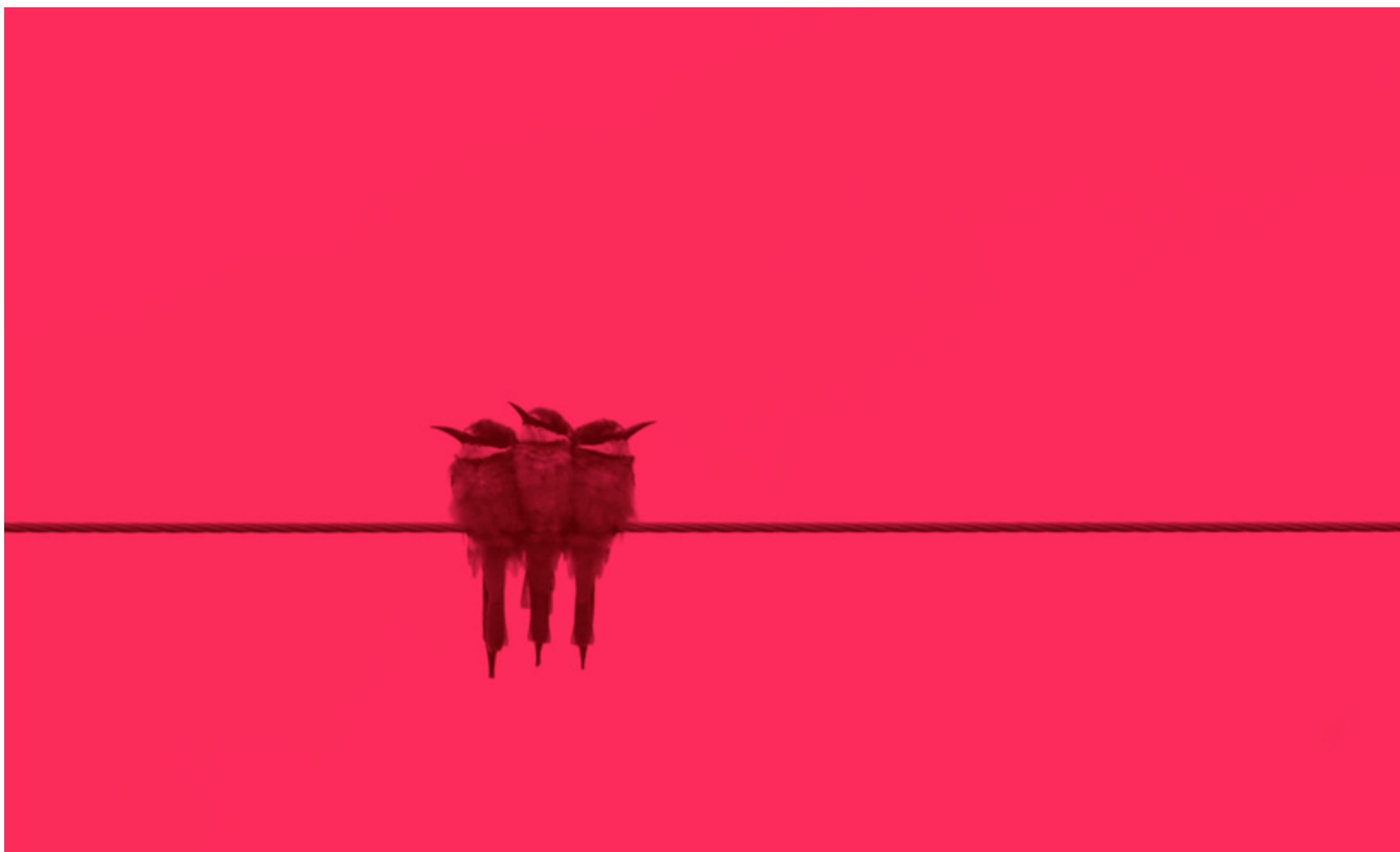
# Observable



# Observable

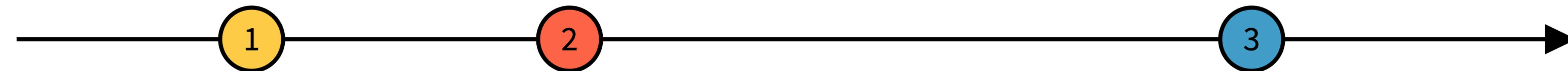


# Observable



# Observable

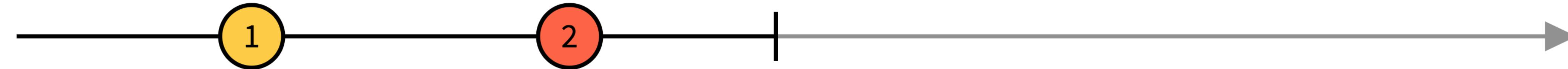
next



error



complete



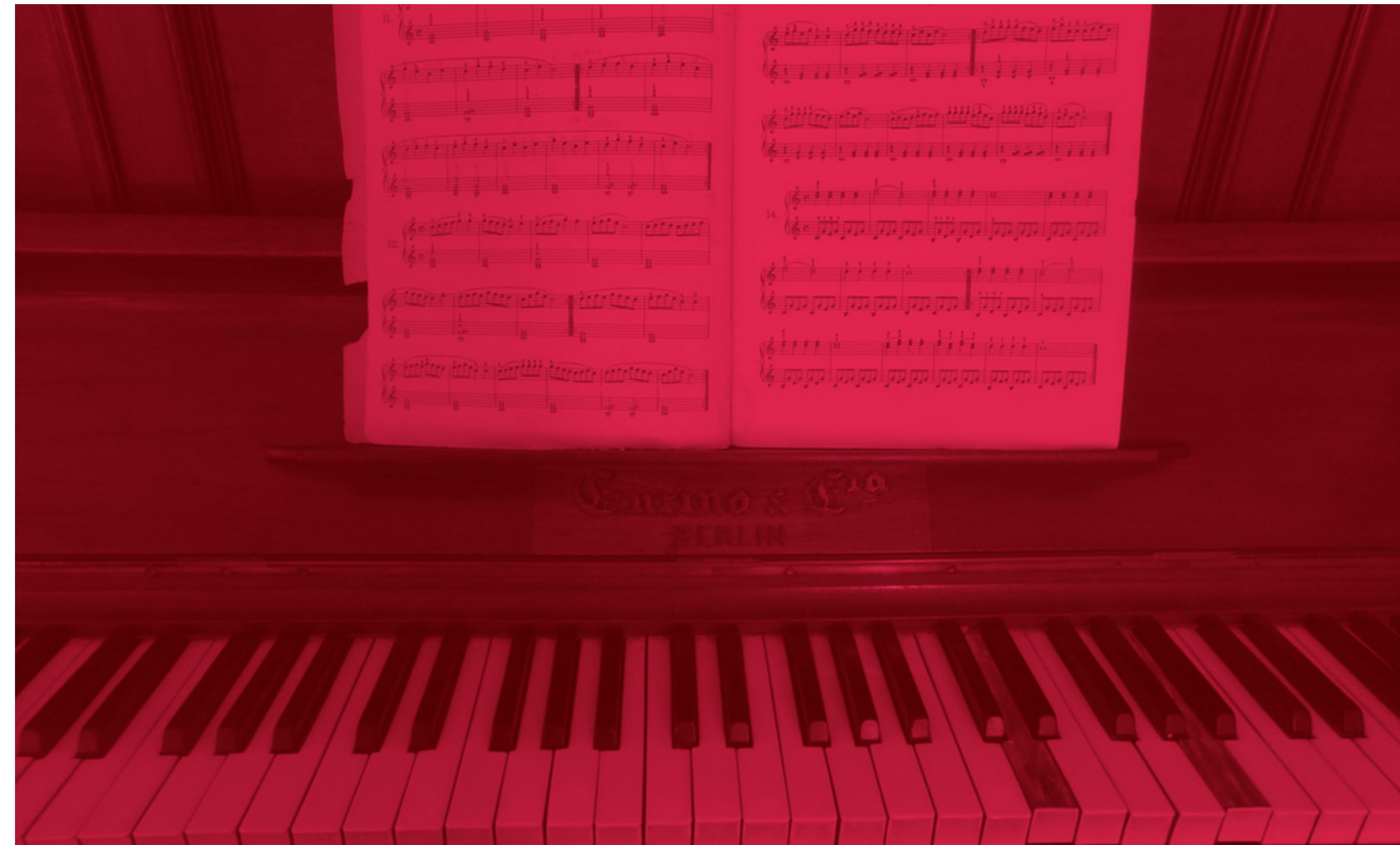


# RxJS : les clefs pour comprendre les observables

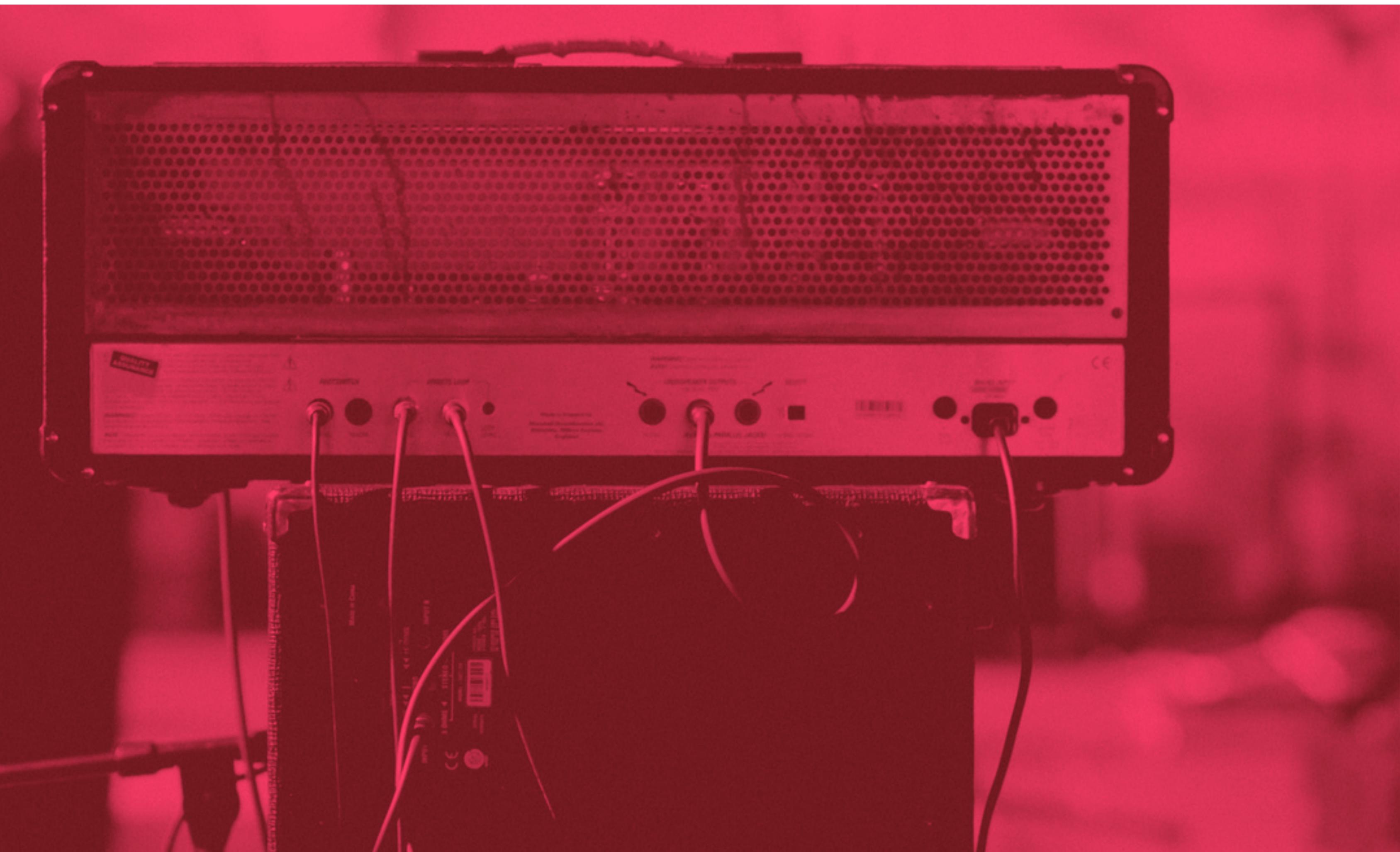
REVOLUTION

## 2. opérateurs

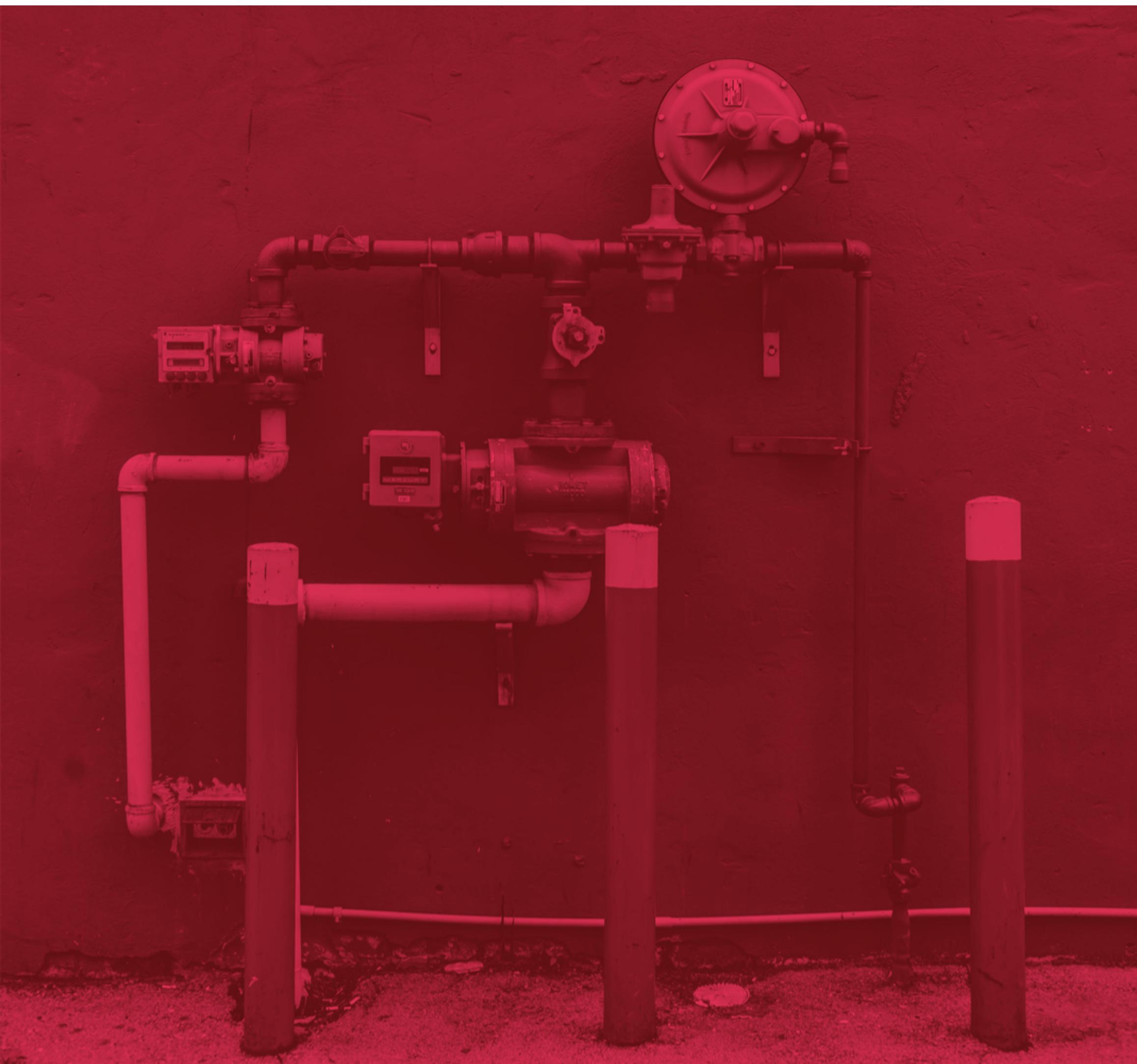
# Opérateurs



# Opérateurs



# Opérateurs



# Opérateurs

```
import {Observable} from 'rxjs';
import {filter, map} from 'rxjs/operators';

getAlertMessageStream(): Observable<string> {
  const notif$: Observable<Notification> =
    this.getNotificationStream();

  return notif$.pipe(
    filter(notif => notif.type === 'ALERT'),
    map(notif => notif.code + ':' + notif.message)
  );
}
```

# Opérateurs

RxJS Home Manual Reference Source Test dark theme light theme

Method Summary

**Public Methods**

public	<code>[Symbol_observable](): Observable</code> An interop point defined by the es7-observable spec <a href="https://github.com/zenparsing/es-observable">https://github.com/zenparsing/es-observable</a>
public	<code>audit(durationSelector: function(value: T): SubscribableOrPromise): Observable&lt;T&gt;</code> Ignores source values for a duration determined by another Observable, then emits the most recent value from the source Observable, then repeats this process.
public	<code>audit(durationSelector: function(value: T): SubscribableOrPromise): Observable&lt;T&gt;</code> Ignores source values for a duration determined by another Observable, then emits the most recent value from the source Observable, then repeats this process.
public	<code>auditTime(duration: number, scheduler: Scheduler): Observable&lt;T&gt;</code> Ignores source values for <code>duration</code> milliseconds, then emits the most recent value from the source Observable, then repeats this process.
public	<code>auditTime(duration: number, scheduler: Scheduler): Observable&lt;T&gt;</code> Ignores source values for <code>duration</code> milliseconds, then emits the most recent value from the source Observable, then repeats this process.
public	<code>buffer(closingNotifier: Observable&lt;any&gt;): Observable&lt;T[]&gt;</code> Buffers the source Observable values until <code>closingNotifier</code> emits.
public	<code>buffer(closingNotifier: Observable&lt;any&gt;): Observable&lt;T[]&gt;</code> Buffers the source Observable values until <code>closingNotifier</code> emits.
public	<code>bufferCount(bufferSize: number, startBufferEvery: number): Observable&lt;T[]&gt;</code> Buffers the source Observable values until the size hits the maximum <code>bufferSize</code> given.
public	<code>bufferCount(bufferSize: number, startBufferEvery: number): Observable&lt;T[]&gt;</code> Buffers the source Observable values until the size hits the maximum <code>bufferSize</code> given.
public	<code>bufferTime(bufferTimeSpan: number, bufferCreationInterval: number, maxBufferSize: number, scheduler: Scheduler): Observable&lt;T[]&gt;</code> Buffers the source Observable values for a specific time period.
public	<code>bufferTime(bufferTimeSpan: number, bufferCreationInterval: number, maxBufferSize: number, scheduler: Scheduler): Observable&lt;T[]&gt;</code> Buffers the source Observable values for a specific time period.
public	<code>bufferToggle(openings: SubscribableOrPromise&lt;O&gt;, closingSelector: function(value: O): SubscribableOrPromise): Observable&lt;T[]&gt;</code> Buffers the source Observable values starting from an emission from <code>openings</code> and ending when the output of <code>closingSelector</code> emits.
public	<code>bufferToggle(openings: SubscribableOrPromise&lt;O&gt;, closingSelector: function(value: O): SubscribableOrPromise): Observable&lt;T[]&gt;</code> Buffers the source Observable values starting from an emission from <code>openings</code> and ending when the output of <code>closingSelector</code> emits.
public	<code>bufferWhen(closingSelector: function(): Observable): Observable&lt;T[]&gt;</code> Buffers the source Observable values, using a factory function of closing Observables to determine when to close, emit, and reset the buffer.
public	<code>bufferWhen(closingSelector: function(): Observable): Observable&lt;T[]&gt;</code> Buffers the source Observable values, using a factory function of closing Observables to determine when to close, emit, and reset the buffer.
public	<code>catch(selector: function): Observable</code> Catches errors on the observable to be handled by returning a new observable or throwing an error.
public	<code>combineAll(project: function): Observable</code> Converts a higher-order Observable into a first-order Observable by waiting for the outer Observable to complete, then applying <code>combineLatest</code> .
public	<code>combineLatest(other: ObservableInput, project: function): Observable</code> Combines multiple Observables to create an Observable whose values are calculated from the latest values of each of its input Observables.
public	<code>combineLatest(other: ObservableInput, project: function): Observable</code> Combines multiple Observables to create an Observable whose values are calculated from the latest values of each of its input Observables.
public	<code>concat(other: ObservableInput, scheduler: Scheduler): Observable</code> Creates an output Observable which sequentially emits all values from every given input Observable after the current Observable.

# Opérateurs des tableaux

[2, 30, 22, 5, 60, 1]

# Opérateurs des tableaux

```
[2, 30, 22, 5, 60, 1]
.filter(x => x > 10)           // [30, 22, 60]
```

# Opérateurs des tableaux

```
[2, 30, 22, 5, 60, 1]
.filter(x => x > 10)           // [30, 22, 60]
```

```
[1, 2, 3, 4, 5]
.every(x => x < 10)          // true
```

# Opérateurs des tableaux

```
[2, 30, 22, 5, 60, 1]
  .filter(x => x > 10)           // [30, 22, 60]
```

```
[1, 2, 3, 4, 5]
  .every(x => x < 10)           // true
```

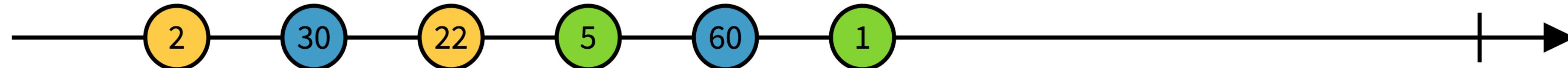
```
[1, 2, 3, 4, 5]
  .map(x => 10 * x)             // [10, 20, 30, 40]
  .reduce((t, x) => t + x, 0)    // 55
```

# Opérateurs des observables

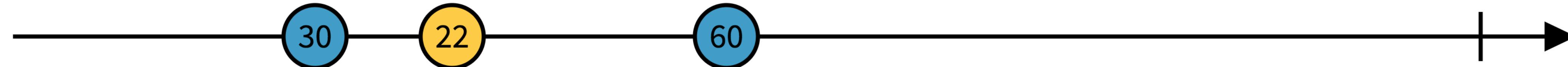
```
[2, 30, 22, 5, 60, 1]
.filter(x => x > 10)           // [30, 22, 60]
```

# Opérateurs des observables

```
[2, 30, 22, 5, 60, 1]  
.filter(x => x > 10) // [30, 22, 60]
```



**filter(x => x > 10)**

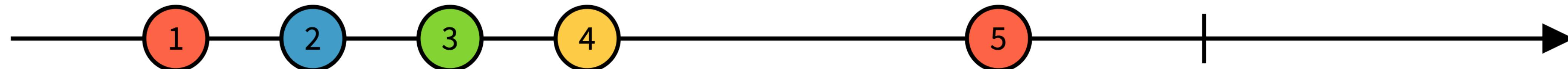


# Opérateurs des observables

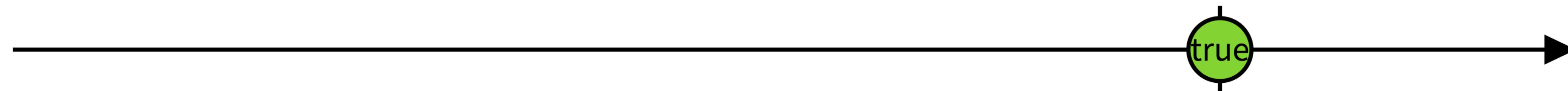
```
[1, 2, 3, 4, 5]
  .every(x => x < 10)           // true
```

# Opérateurs des observables

```
[1, 2, 3, 4, 5]  
.every(x => x < 10) // true
```



**every(x => x < 10)**

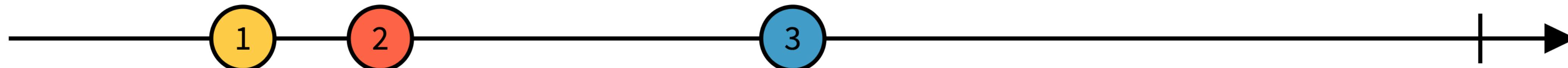


# Opérateurs des observables

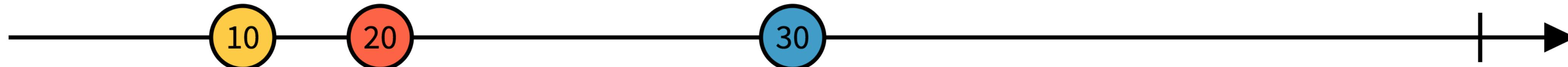
```
[1, 2, 3, 4, 5]
  .map(x => 10 * x)          // [10, 20, 30, 40]
  .reduce((t, x) => t + x, 0) // 55
```

# Opérateurs des observables

```
[1, 2, 3, 4, 5]
  .map(x => 10 * x)          // [10, 20, 30, 40]
  .reduce((t, x) => t + x, 0) // 55
```



**map(x => 10 \* x)**

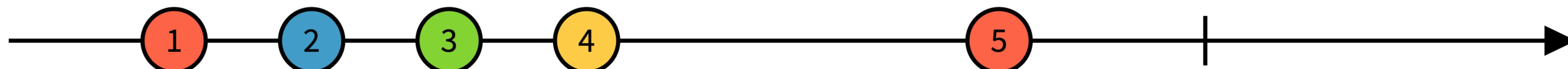


# Opérateurs des observables

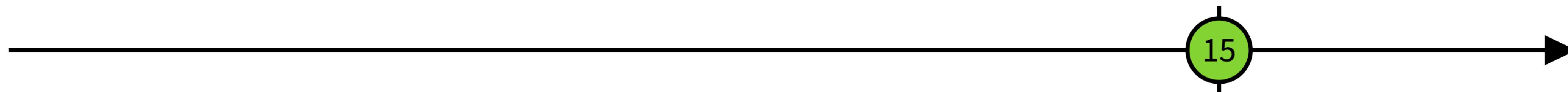
```
[1, 2, 3, 4, 5]
  .map(x => 10 * x)          // [10, 20, 30, 40]
  .reduce((t, x) => t + x, 0) // 55
```

# Opérateurs des observables

```
[1, 2, 3, 4, 5]
  .map(x => 10 * x)          // [10, 20, 30, 40]
  .reduce((t, x) => t + x, 0) // 55
```



**reduce((t, x) => t + x, 0)**

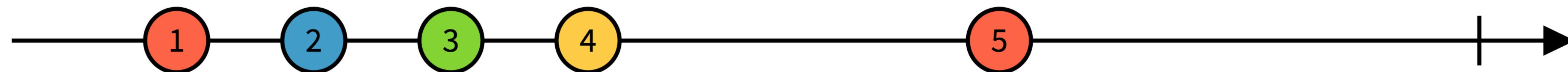


# Opérateurs des observables

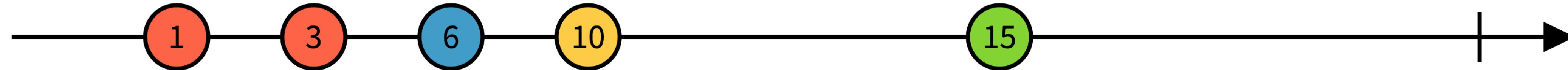
```
[1, 2, 3, 4, 5]
  .map(x => 10 * x)          // [10, 20, 30, 40]
  .reduce((t, x) => t + x, 0) // 55
```

# Opérateurs des observables

```
[1, 2, 3, 4, 5]
  .map(x => 10 * x)          // [10, 20, 30, 40]
  .reduce((t, x) => t + x, 0) // 55
```



**scan((t, x) => t + x, 0)**



# Opérateurs des observables

## Opérateur RxJS :

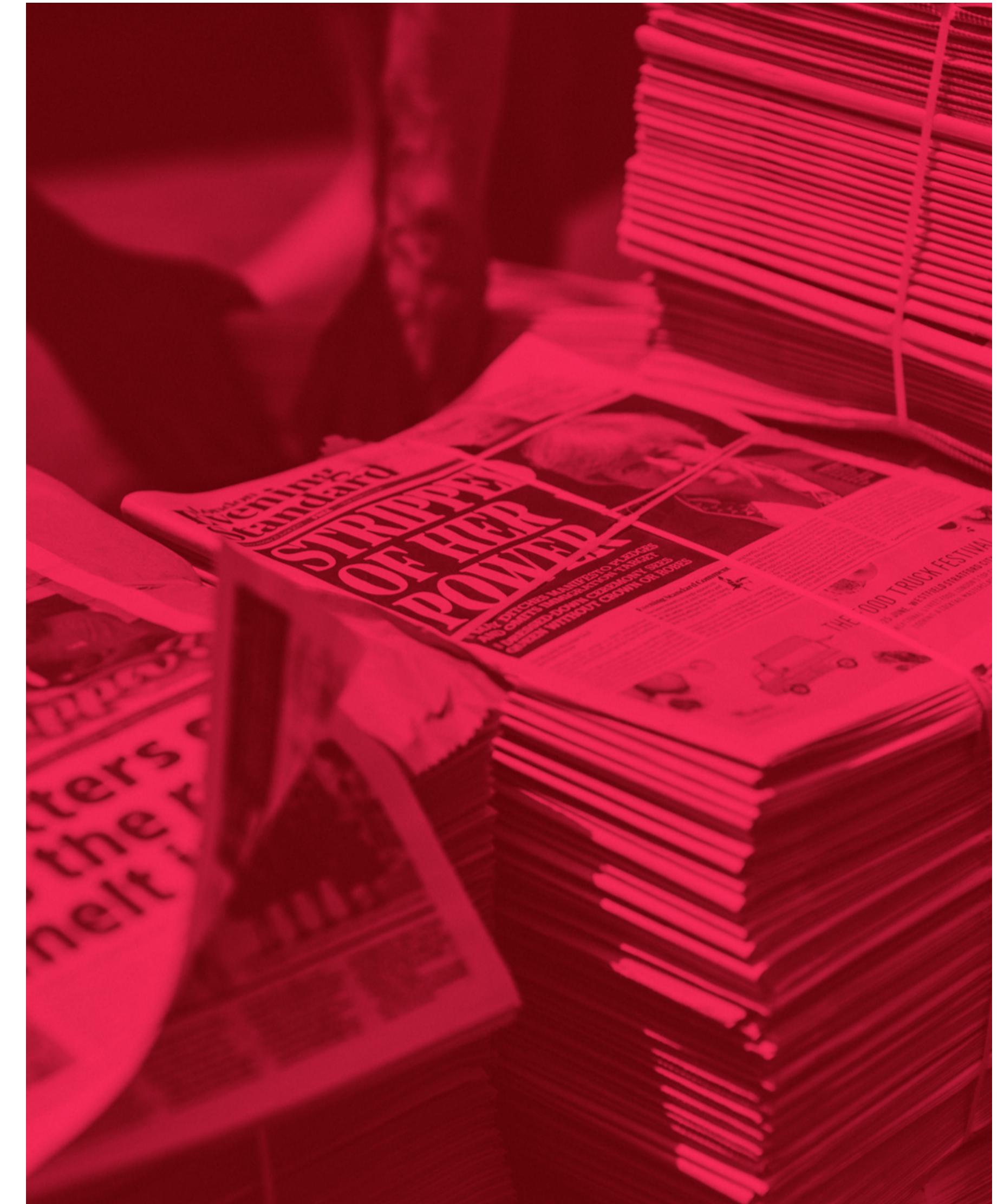
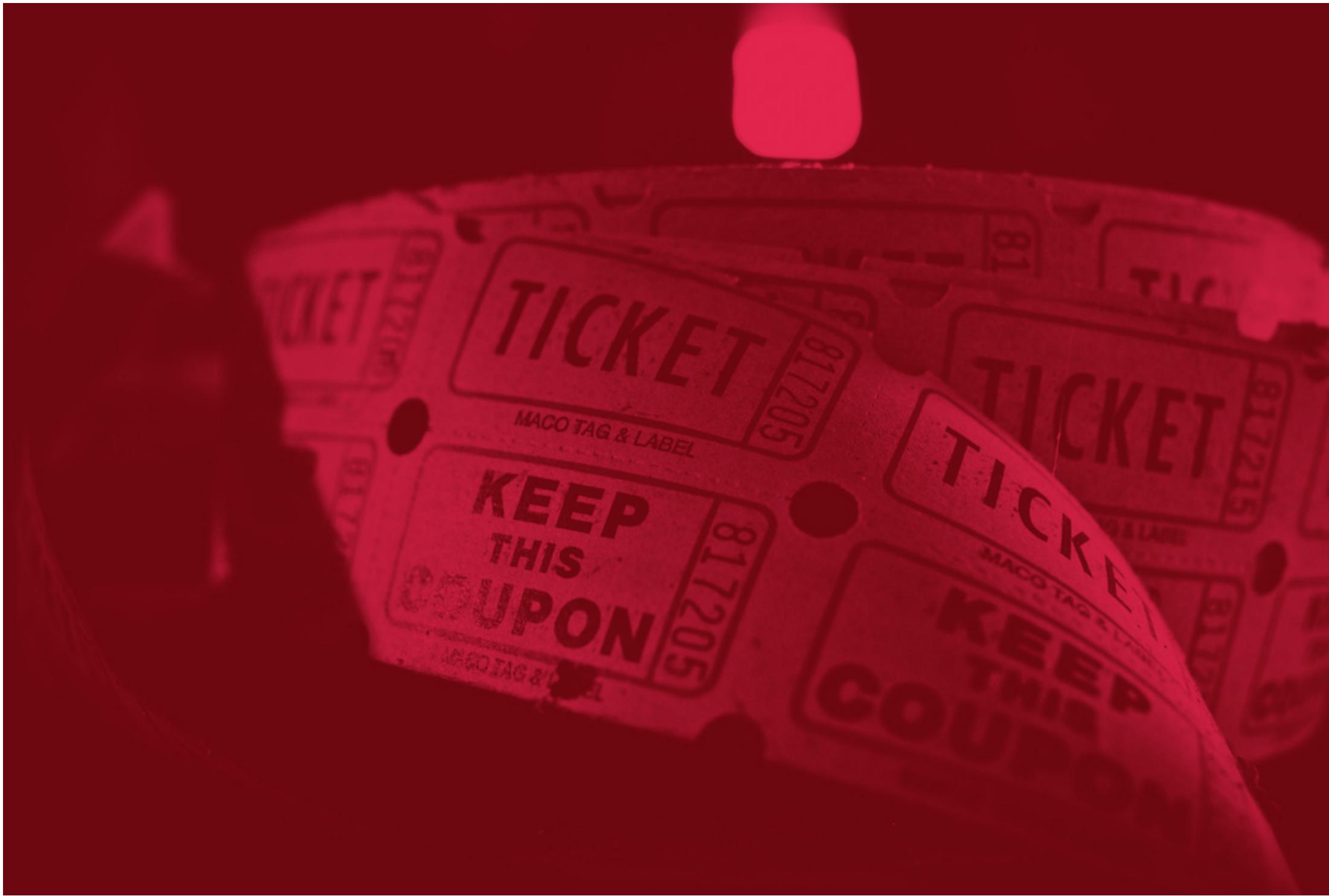
- fonction *pure*
- renvoie un nouvel observable
- sans modifier celui d'origine

# RxJS : les clefs pour comprendre les observables

REVOLUTION

## 3. souscription

# Souscription



# Souscription

```
const subscription = observable.subscribe(  
    next: value => console.log(value),  
    error: error => console.log(error),  
    complete: () => console.log('complete')  
) ;
```

```
const subscription = observable.subscribe( observer: {  
    next: value => console.log(value),  
    error: error => console.log(error),  
    complete: () => console.log('complete')  
});
```

# Souscription



```
subscription.unsubscribe();
```

# Souscription



# RxJS : les clefs pour comprendre les observables

REVOLUTION

## 4. cold & hot

# Comportement

COLD

VS

HOT



# Comportement

COLD



HOT

# Comportement

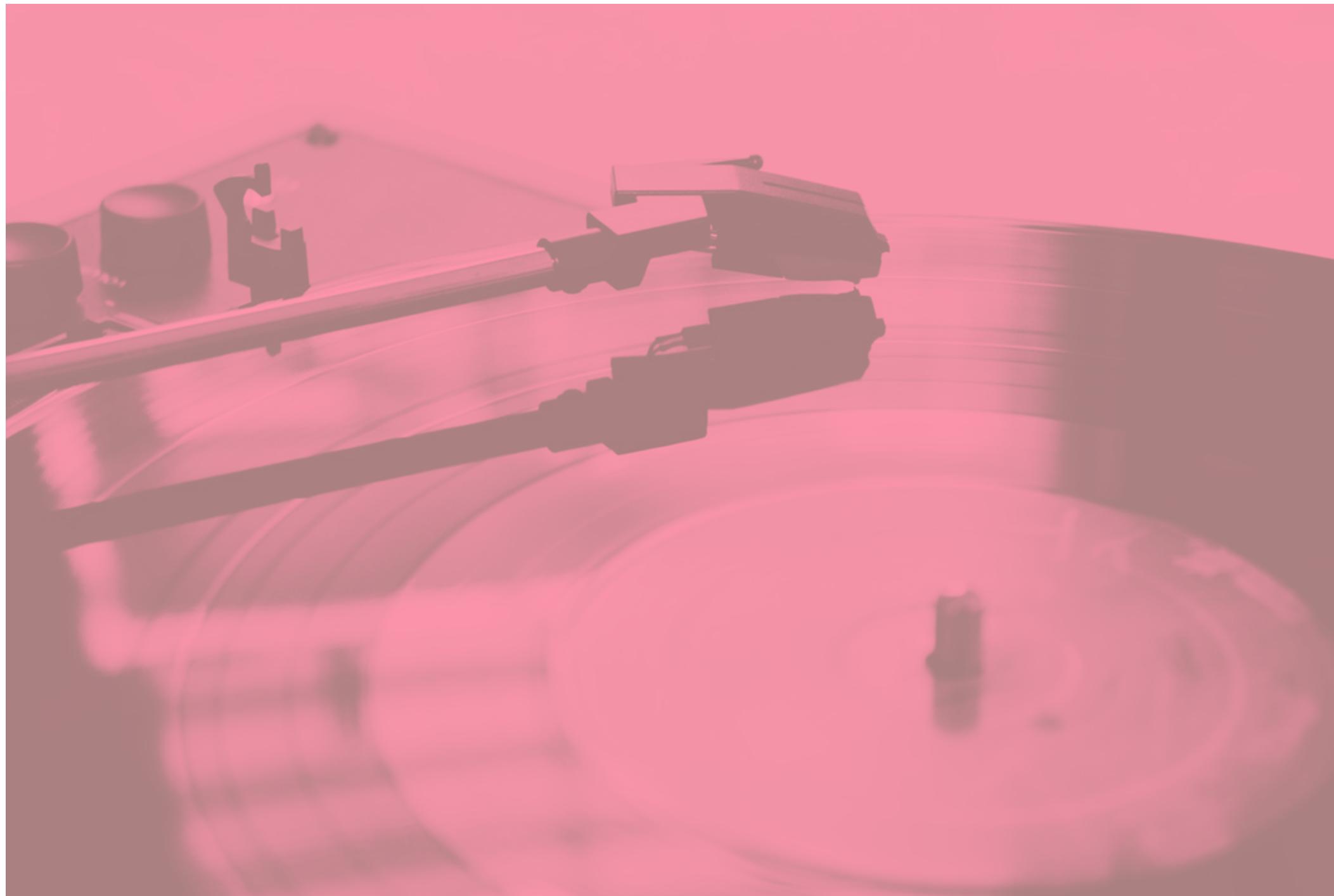
COLD



HOT

# Comportement

**COLD** = unicast



**HOT** = multicasted

# Comportement



# Comportement



# Comportement



**HTTP**

# RxJS : les clefs pour comprendre les observables

REVOLUTION

## 5. création (et transformation)

# Création : COLD

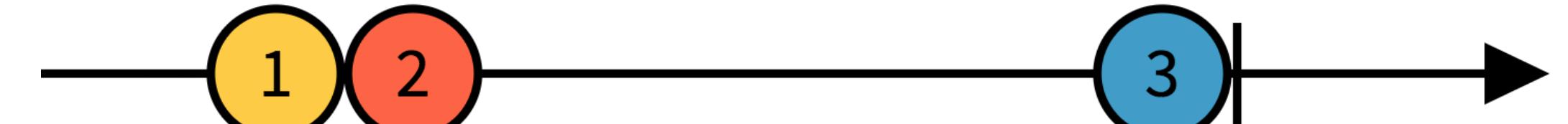
## create()

```
import {Observable, create} from 'rxjs';

const observable: Observable<number> =
  create(observer => {
    observer.next(1);
    observer.next(2);
    setTimeout(() => {
      observer.next(3);
      observer.complete();
    }, 1000);
  });

```

```
const subscription = observable.subscribe( observer: {
  next: value => console.log(value),
  error: error => console.log(error),
  complete: () => console.log('complete')
});
```



# Création : COLD

**of(1, 2, 3)**

**empty()**

**never()**

**throwError(error)**

```
import {Observable, of} from 'rxjs';
```

```
const numbers: Observable<number> = of(1, 2, 3);
```

# Création : HOT

**Subject** = Observer & *multicasted Observable*

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

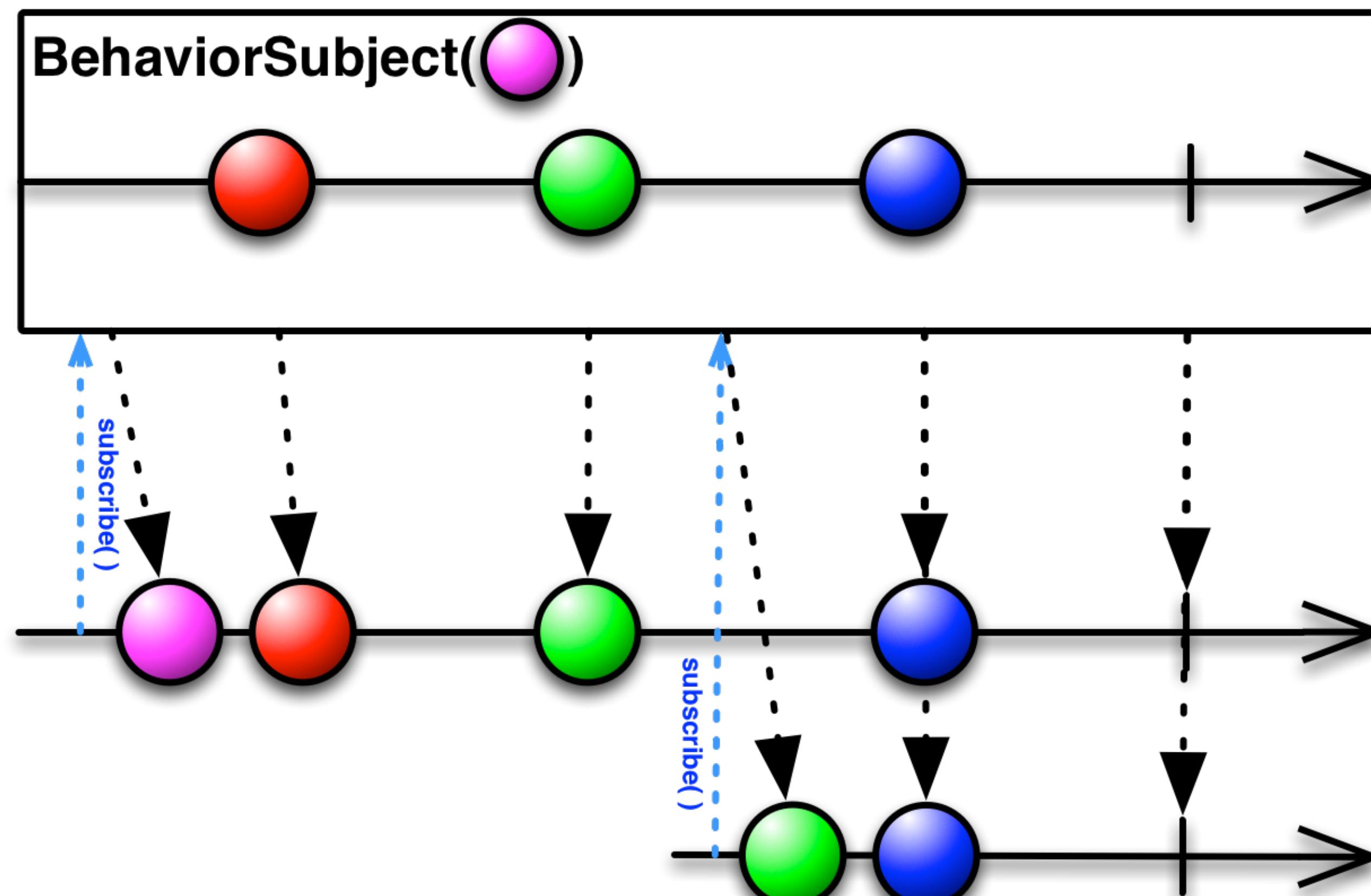
subject.subscribe(v => console.log('observerA: ' + v));
subject.next(1);
// observerA: 1

subject.subscribe(v => console.log('observerB: ' + v));
subject.next(2);
// observerA: 2
// observerB: 2
```

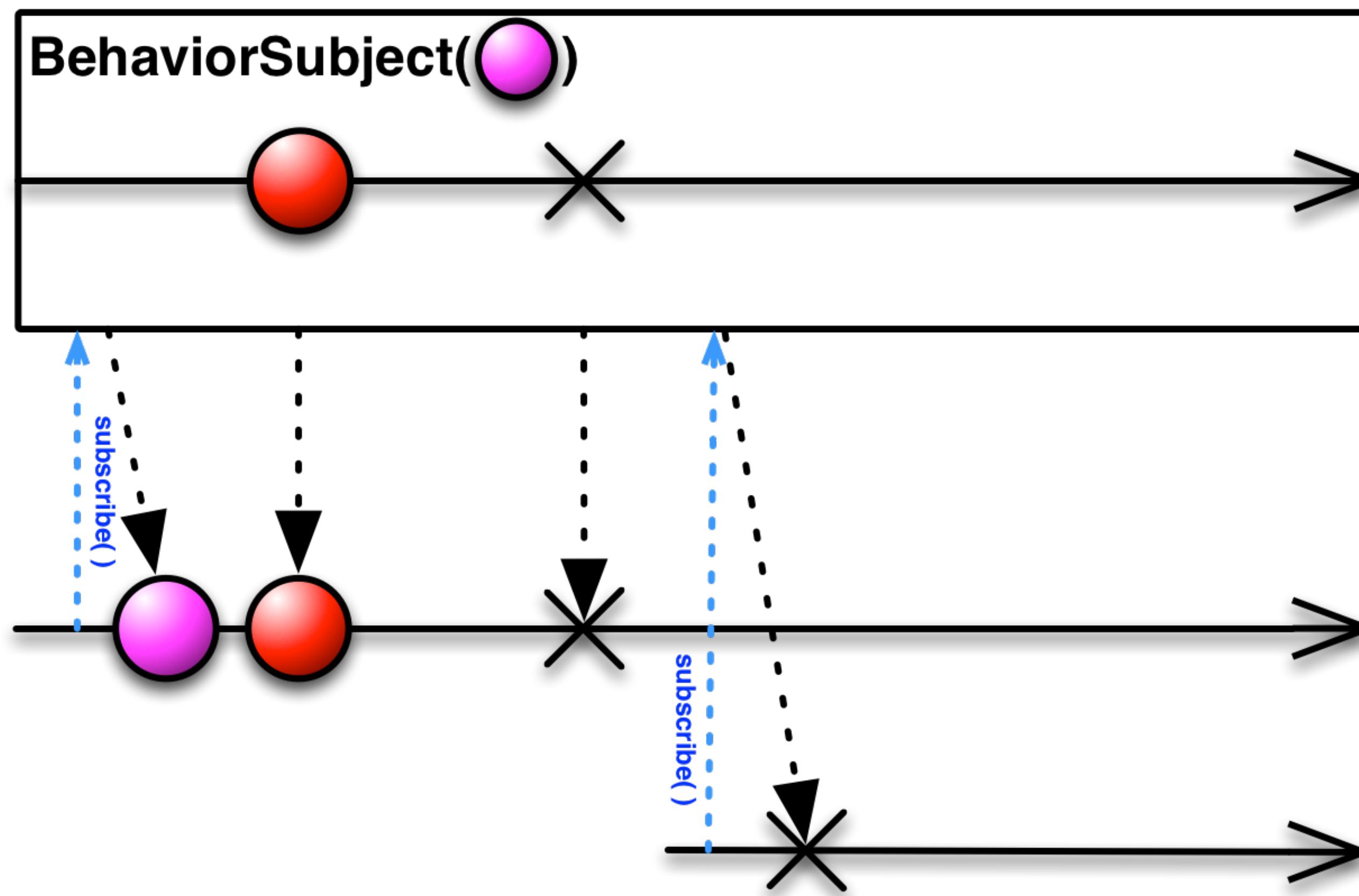
# Création : HOT

```
buildEventEmitter(): Observable<number> {  
    const subject = new Subject<number>();  
  
    setTimeout(() => {  
        subject.next(1);  
    }, 1000);  
  
    setTimeout(() => {  
        subject.next(2);  
    }, 2000);  
  
    return subject.asObservable();  
}
```

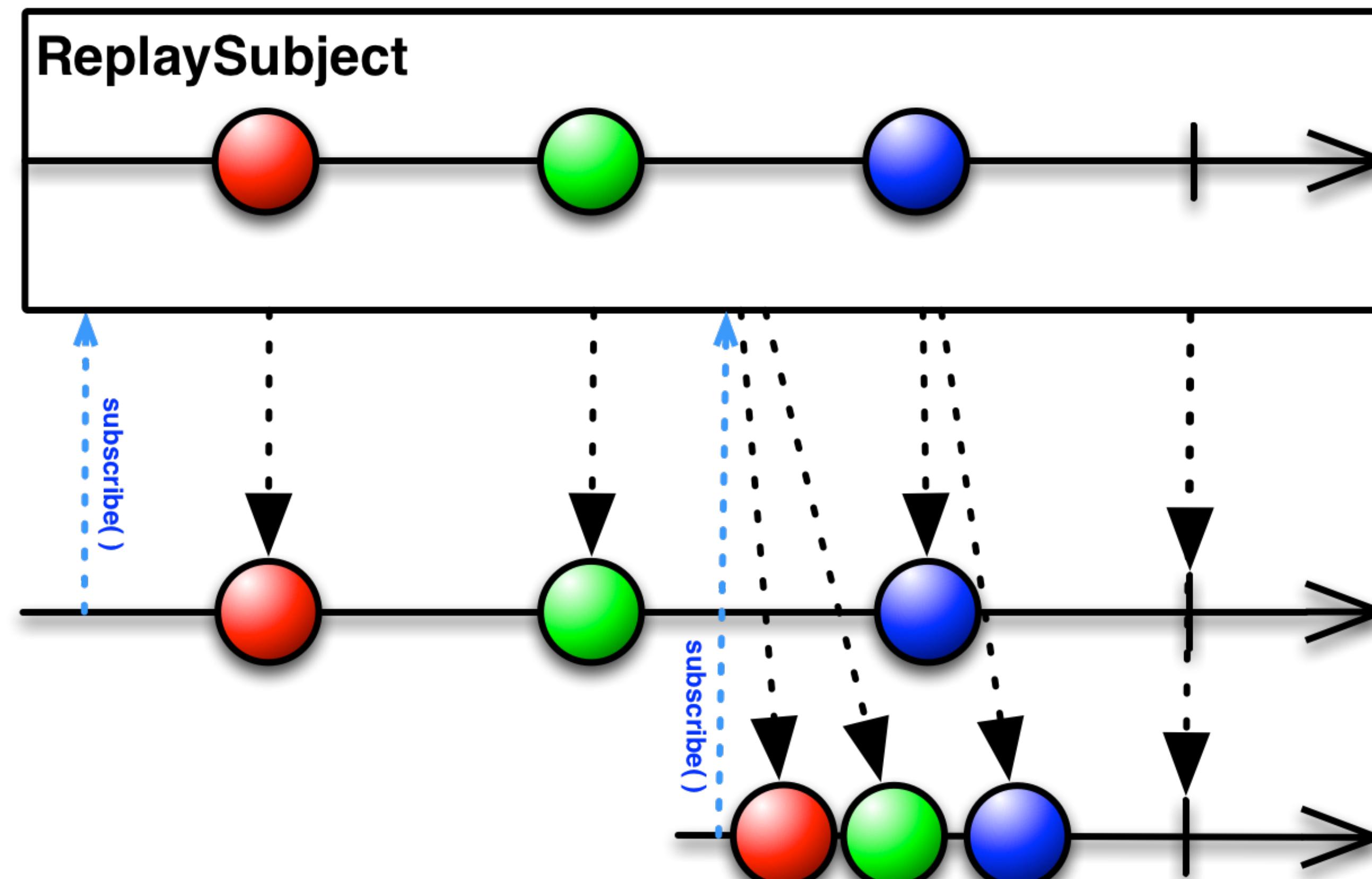
**BehaviorSubject** : conserver un état (valeur courante)



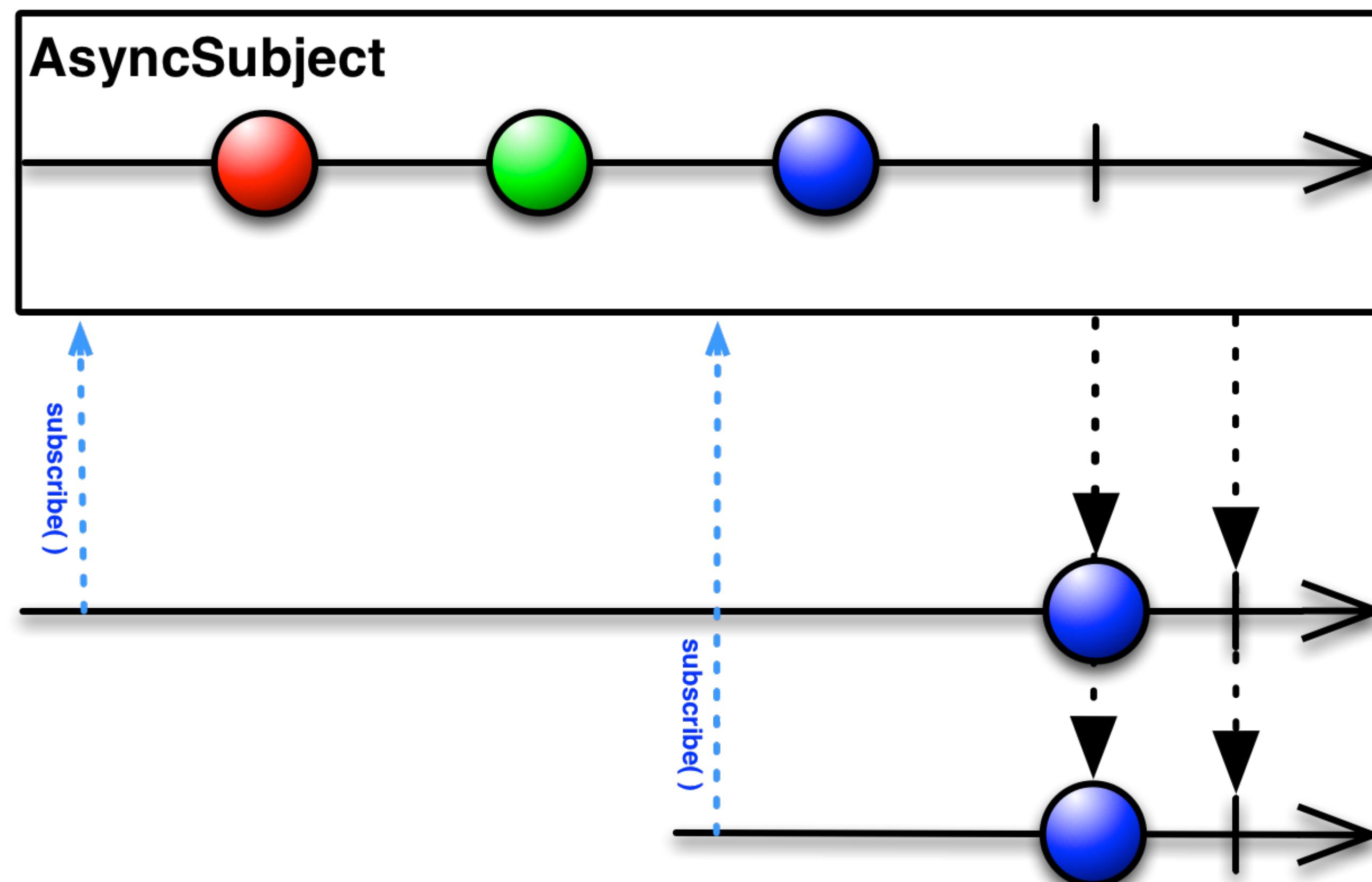
## BehaviorSubject : conserver un état (valeur courante)



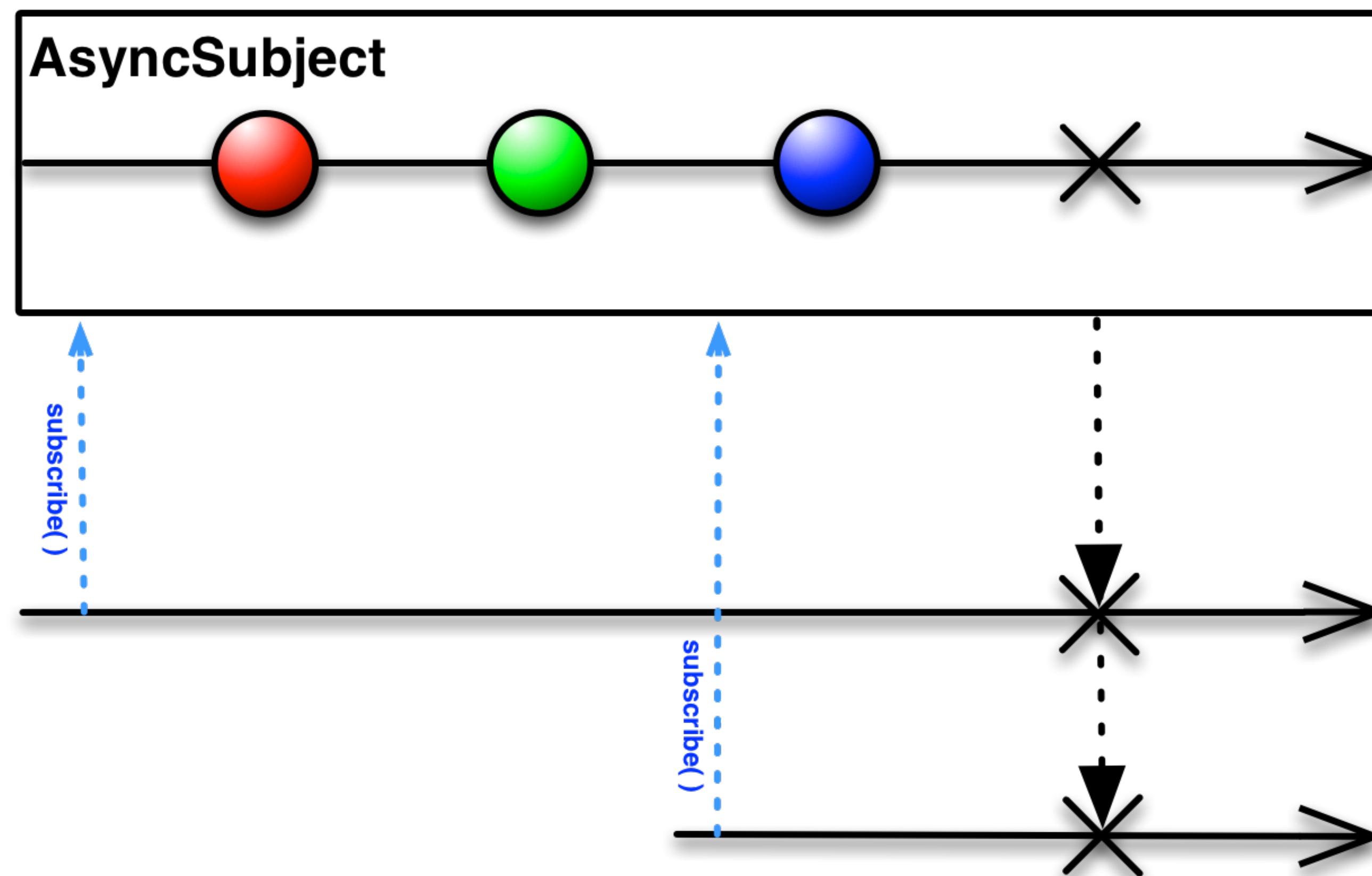
**ReplaySubject** : dernières valeurs en buffer



## AsyncSubject : résultat d'une opération asynchrone



## AsyncSubject : résultat d'une opération asynchrone



**fromPromise(p)**

**fromEvent(element, 'click')**

# Transformation : COLD → HOT

**share()**

**shareReplay(5)**

```
import {share, shareReplay} from 'rxjs/operators';
```

```
const hot$ = cold$.pipe(  
  share()  
);
```

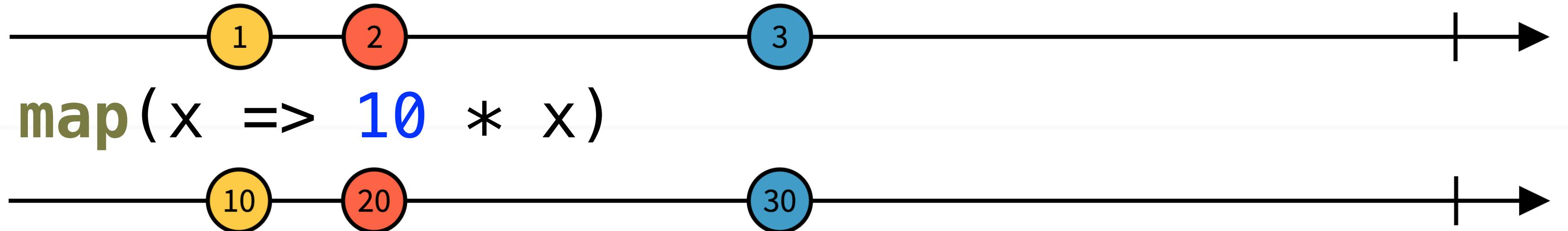
```
const hot$ = cold$.pipe(  
  shareReplay(1)  
);
```

# RxJS : les clefs pour comprendre les observables

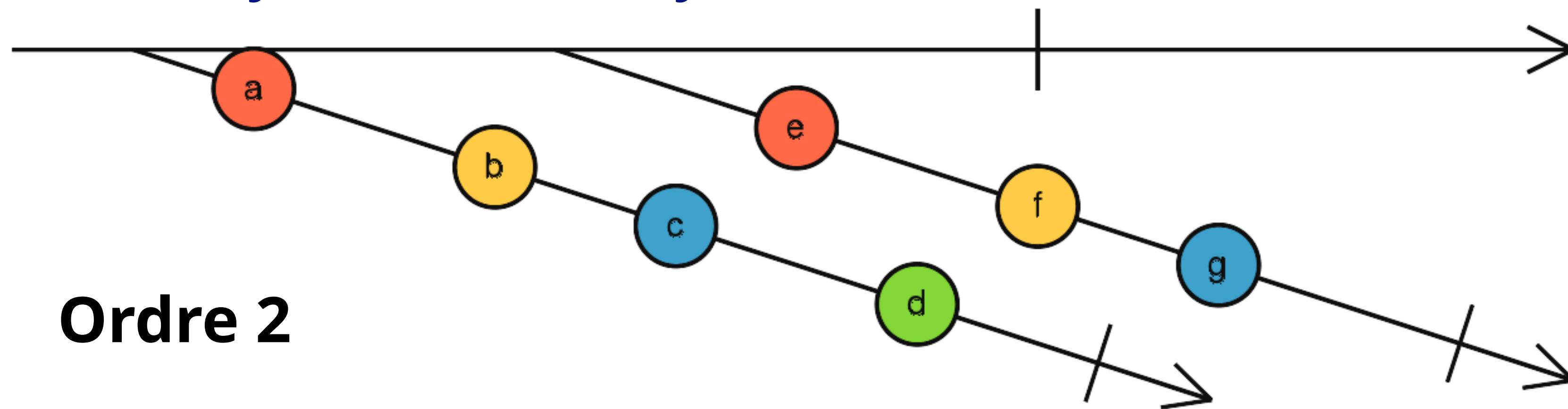
6. ordre 2

# Ordre 2

Tableaux de tableaux : `const array = [[1, 2], [3, 4]];`

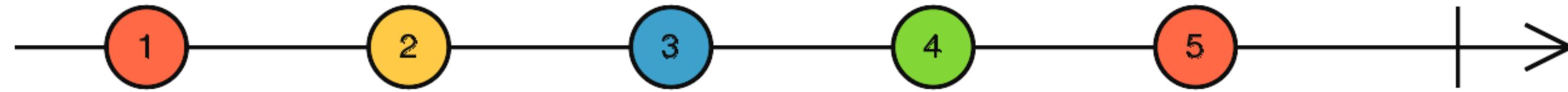


*calcul asynchrone renvoyant un observable :*

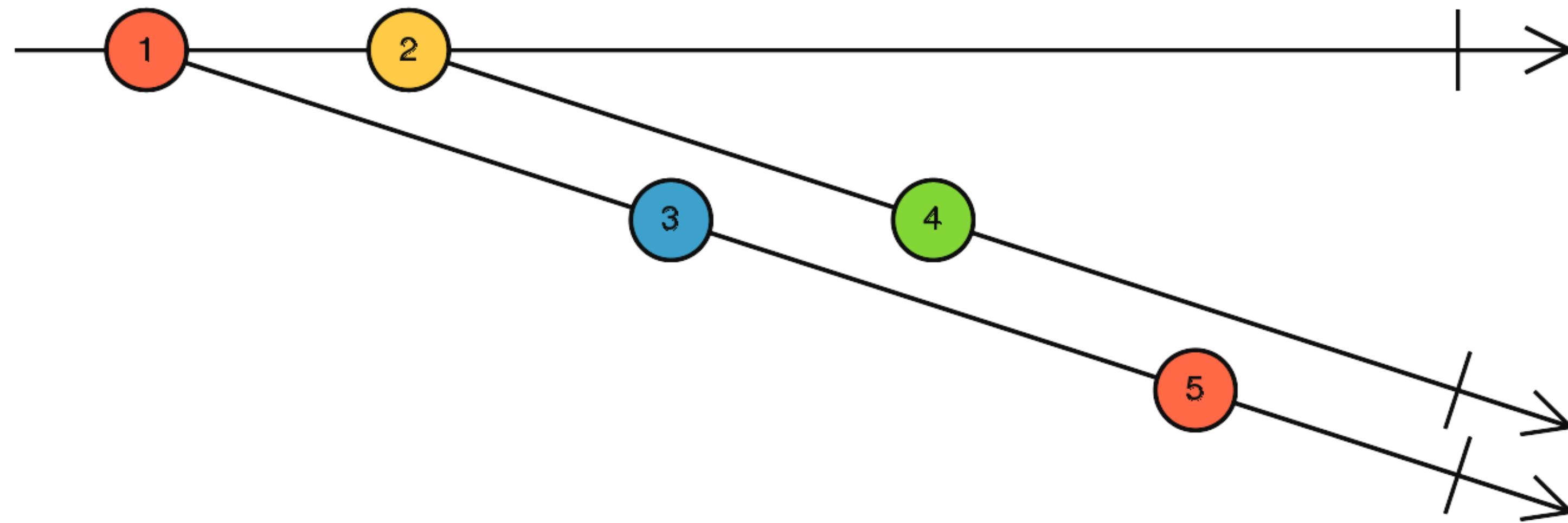


Ordre 2

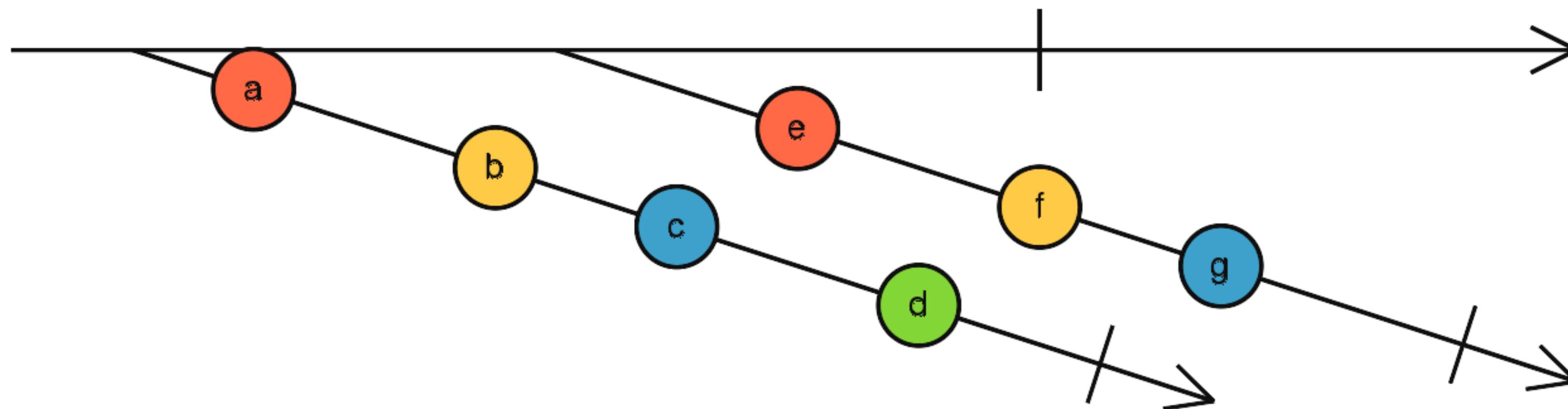
# Ordre 2



**groupBy(x => x % 2)**



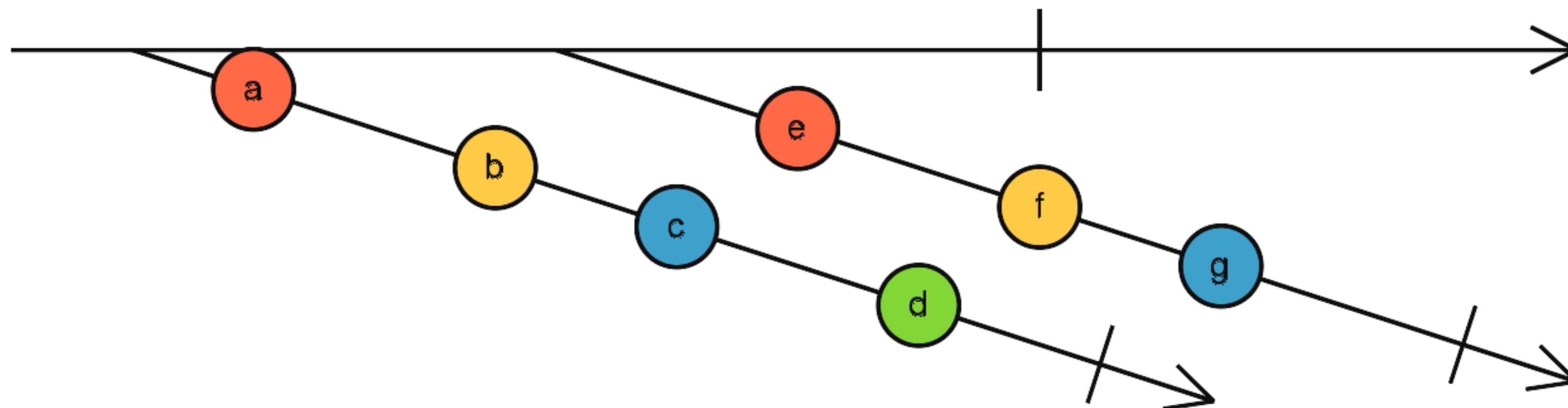
# Ordre 2 : aplatisir



*Requêtes simultanées :*

- exécuter en parallèle
- exécuter à la suite
- annuler la précédente
- annuler la nouvelle

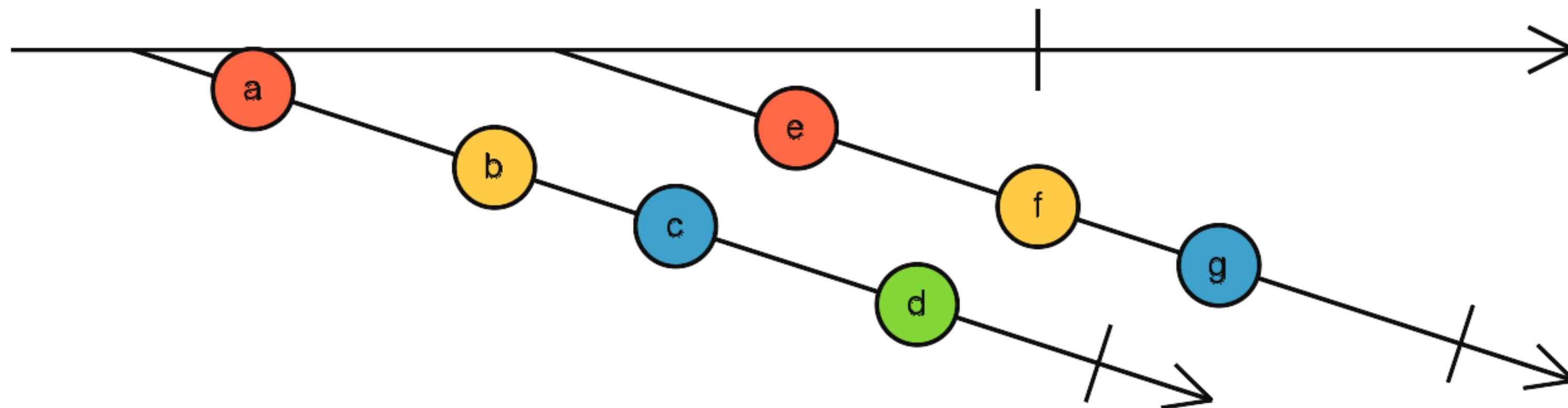
# Ordre 2 : aplatisir



*Requêtes simultanées :*

- exécuter en parallèle      `map()`, `mergeAll()`
- exécuter à la suite      `map()`, `concatAll()`
- annuler la précédente      `map()`, `switch()`
- annuler la nouvelle      `map()`, `exhaust()`

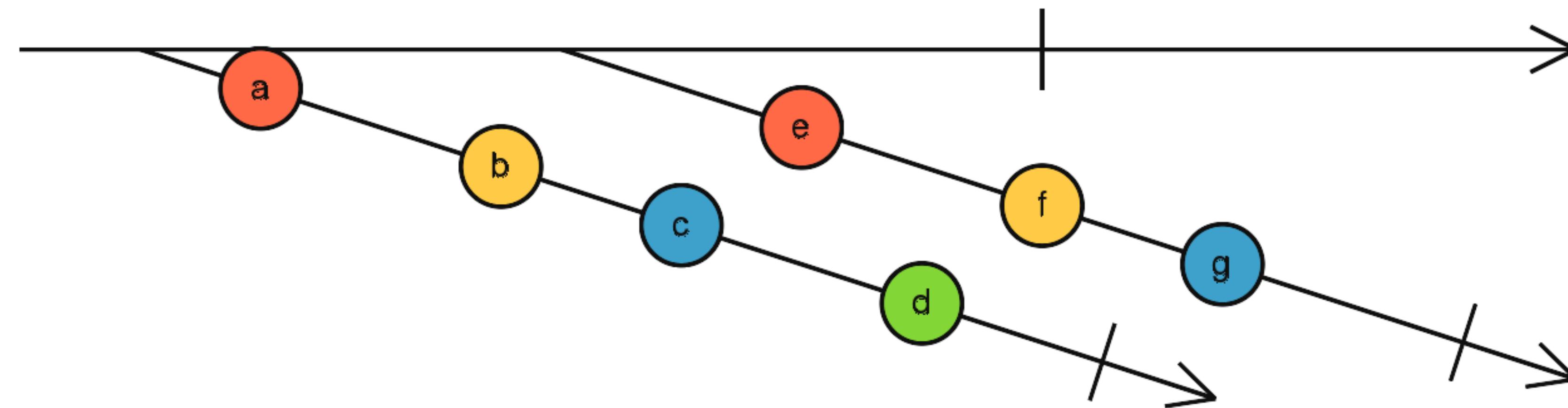
# Ordre 2 : aplatisir



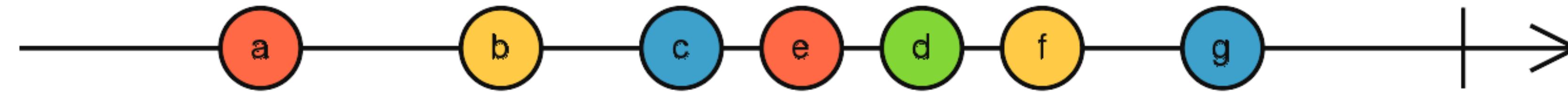
*Requêtes simultanées :*

- |                         |   |                           |
|-------------------------|---|---------------------------|
| • exécuter en parallèle | <code>map()</code> , <code>mergeAll()</code>  | <code>mergeMap()</code>   |
| • exécuter à la suite   | <code>map()</code> , <code>concatAll()</code> | <code>concatMap()</code>  |
| • annuler la précédente | <code>map()</code> , <code>switch()</code>    | <code>switchMap()</code>  |
| • annuler la nouvelle   | <code>map()</code> , <code>exhaust()</code>   | <code>exhaustMap()</code> |

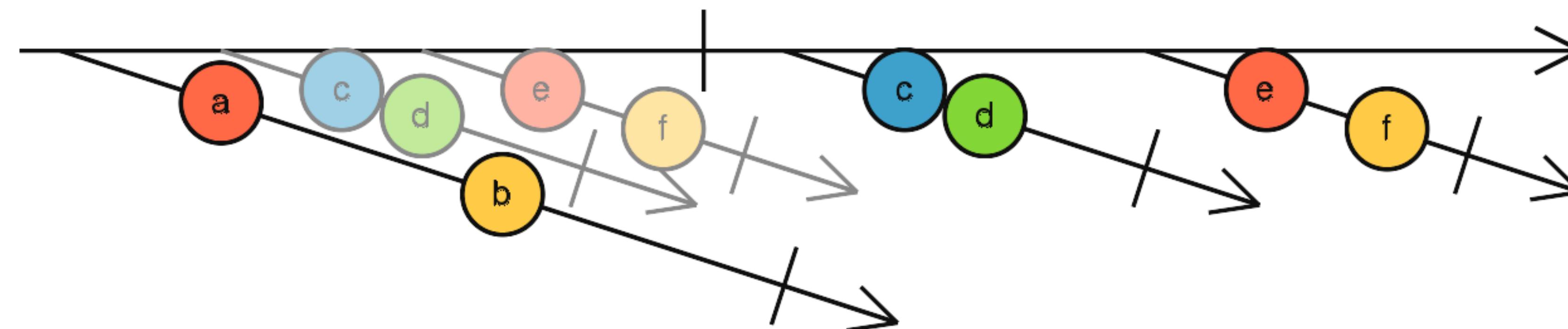
# Ordre 2 : aplatisir



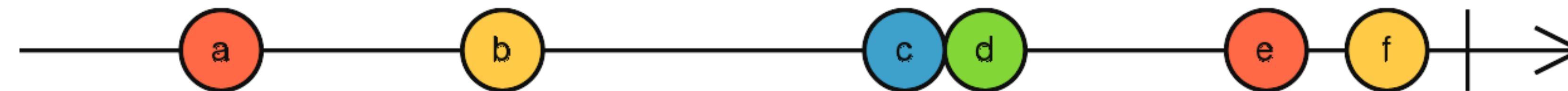
`mergeAll / mergeMap`



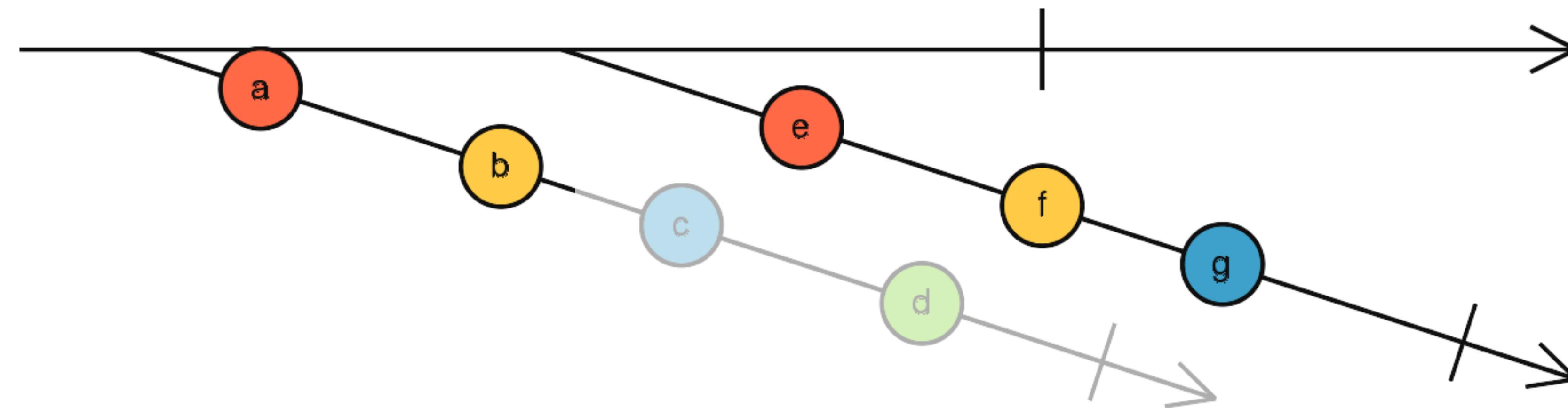
# Ordre 2 : aplatisir



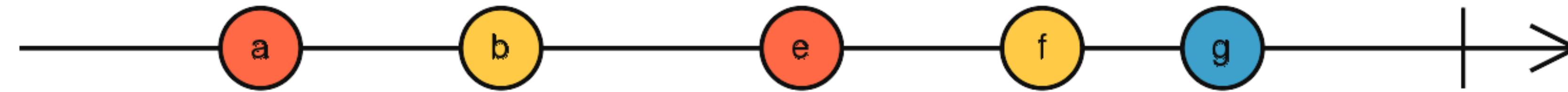
**concatAll / concatMap**



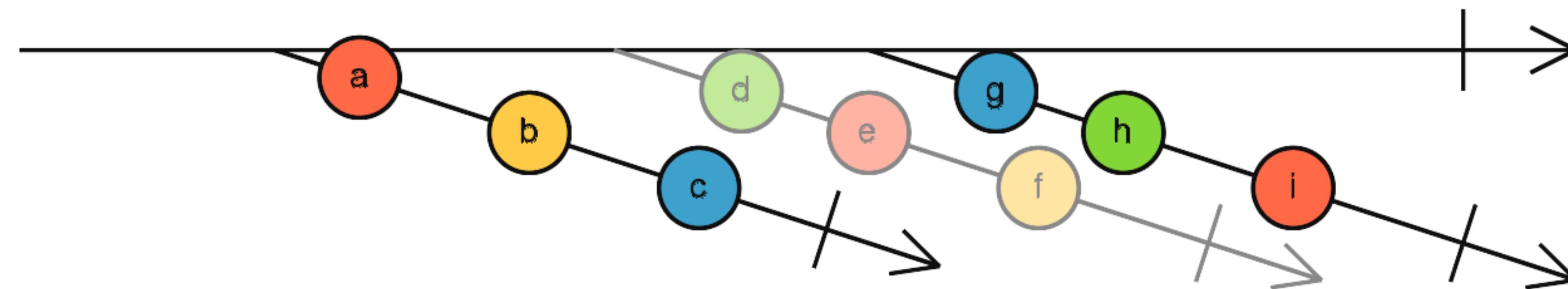
# Ordre 2 : aplatisir



**switch / switchMap**



# Ordre 2 : aplatisir



**exhaust / exhaustMap**



# RxJS : les clefs pour comprendre les observables

REVOLUTION

## 7. how-to

# HOW-TO : services asynchrones

**TOUJOURS** renvoyer un observable

**NE PAS** souscrire pour fournir les vraies données

*car on ne saura pas quand elles arrivent*

```
// OUI
export class PhotosService {
    constructor(private httpClient: HttpClient) { }
    list(): Observable<Photo[]> {
        const url = 'http://localhost:3390/photos';
        return this.httpClient.get<Photo[]>(url);
    }
}
```

# HOW-TO : services asynchrones

**TOUJOURS** renvoyer un observable

**NE PAS** souscrire pour fournir les vraies données  
*car on ne saura pas quand elles arrivent*

// *NON*



```
export class PhotosService {
    photos: Photo[];
    constructor(private httpClient: HttpClient) { }
    load(): void {
        const url = 'http://localhost:3390/photos';
        this.httpClient.get<Photo[]>(url)
            .subscribe(photos => this.photos = photos);
    }
}
```

# HOW-TO : erreurs

**PRINCIPE** : Remonter les erreurs au niveau où on peut les (comme les exceptions)

Déclencher une erreur : **throw 'fail'**;  
ou : **throwError('fail')**;

Intercepter une erreur : **catchError**

Réessayer : **retry** ou **retryWhen**  
(ou **retryWhen + scan**)

# HOW-TO : erreurs

```
import {Observable, of} from 'rxjs';
import {catchError} from 'rxjs/operators';

find(id: string): Observable<Resource> {
    const url = `http://.../resources/${id}`;
    return this.httpClient.get<Resource>(url)
        .pipe(
            catchError(error => of(null))
        );
}
```

# HOW-TO : erreurs

```
import {Observable, of} from 'rxjs';
import {catchError} from 'rxjs/operators';

find(id: string): Observable<Resource> {
    const url = `http://.../resources/${id}`;
    return this.httpClient.get<Resource>(url)
        .pipe(
            catchError(error => {
                if (error && error.status === 404) {
                    return of(null);
                }
                throw error;
            })
        );
}
```

# RxJS : les clefs pour comprendre les observables

## 8.1 exemple double clic

# Exemple : double-clicks

```
import {Observable, fromEvent} from 'rxjs';
import {buffer, debounceTime, map, filter} from 'rxjs/operators';

buildDoubleClicksObservable(button: HTMLButtonElement,
                           delay: number): Observable<MouseEvent> {

  const clicks$: Observable<MouseEvent> =
    fromEvent(button, 'click');

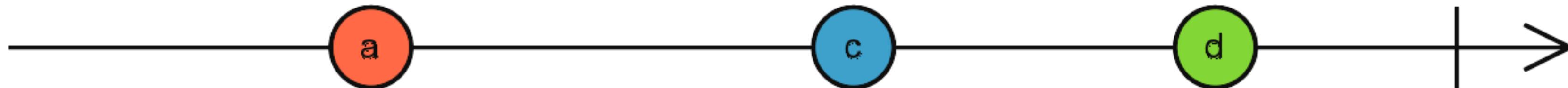
  const doubleClicks$: Observable<MouseEvent> =
    clicks$.pipe(
      buffer(clicks$.pipe(debounceTime(delay))),
      filter(clickGroup => clickGroup.length === 2),
      map(clickGroup => clickGroup[0])
    );

  return doubleClicks$;
}
```

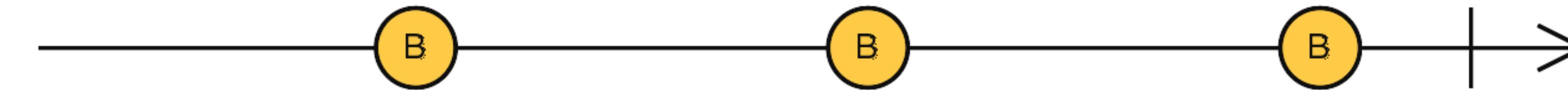
# Exemple : double-clicks



`debounceTime( 200 )`



# Exemple : double-clicks



buffer



# Exemple : double-clicks

```
import {Observable, fromEvent} from 'rxjs';
import {buffer, debounceTime, map, filter} from 'rxjs/operators';

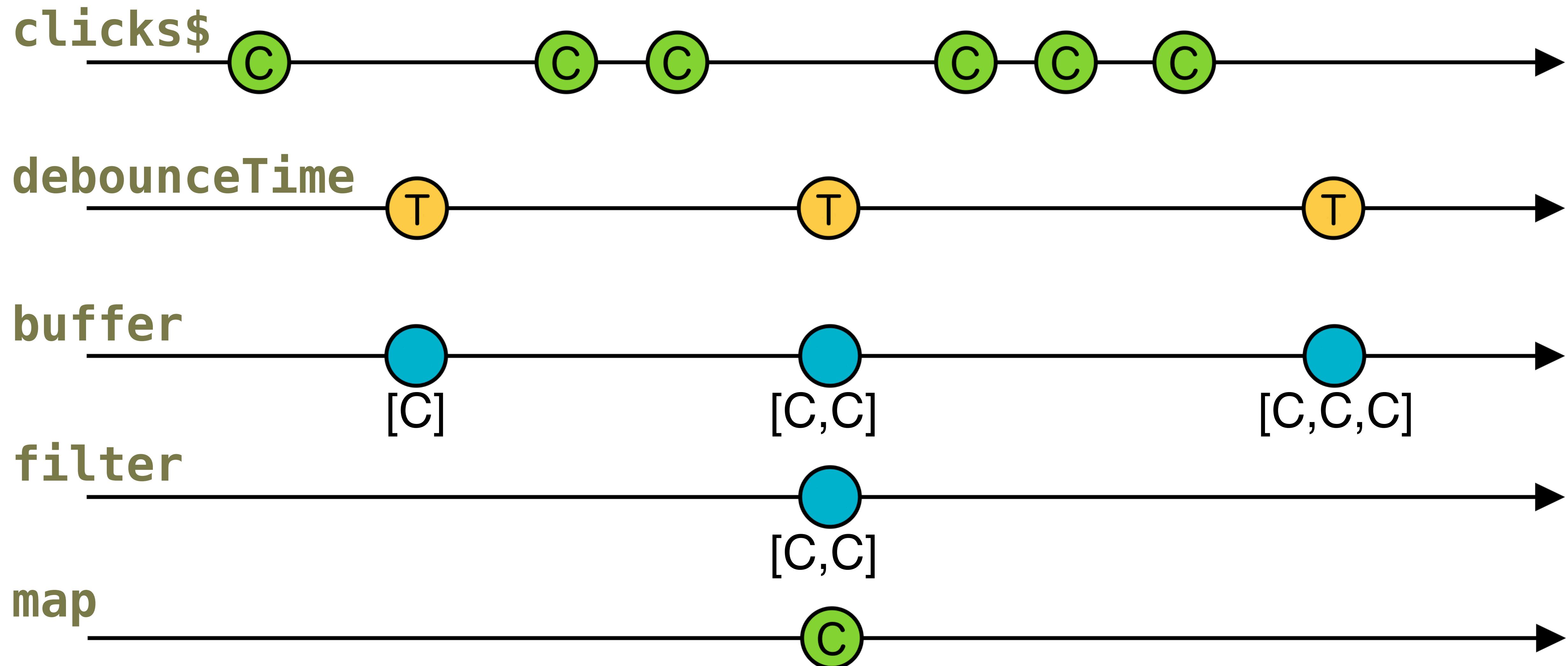
buildDoubleClicksObservable(button: HTMLButtonElement,
                            delay: number): Observable<MouseEvent> {

  const clicks$: Observable<MouseEvent> =
    fromEvent(button, 'click');

  const doubleClicks$: Observable<MouseEvent> =
    clicks$.pipe(
      buffer(clicks$.pipe(debounceTime(delay))),
      filter(clickGroup => clickGroup.length === 2),
      map(clickGroup => clickGroup[0])
    );

  return doubleClicks$;
}
```

# Exemple : double-clicks



# RxJS : les clefs pour comprendre les observables

## 8.2 exemple conserver les données (d'une requête HTTP)

# Exemple : conserver les données

```
list$: Observable<Book[]>;  
  
constructor(private httpClient: HttpClient) {  
  this.list$ = this.buildRequestObservable();  
}  
  
buildRequestObservable() {  
  return this.httpClient  
    .get<Book[]>(catalogUrl, {params: catalogUrlParams});  
}  
  
getList(): Observable<Book[]> {  
  return this.list$;  
}
```

$n$  requêtes HTTP

# Exemple : conserver les données

```
list$: Observable<Book[]>;  
  
constructor(private httpClient: HttpClient) {  
  this.list$ = this.buildRequestObservable();  
}  
  
buildRequestObservable() {  
  return this.httpClient  
    .get<Book[]>(catalogUrl, {params: catalogUrlParams})  
    .pipe(  
      shareReplay(1)  
    );  
}  
  
getList(): Observable<Book[]> {  
  return this.list$;  
}
```

1 requête HTTP

# Exemple : conserver les données

```
list$: Observable<Book[]>;  
  
constructor(private httpClient: HttpClient) {  
    this.list$ = this.buildRequestObservable();  
    setInterval(() => {  
        this.list$ = this.buildRequestObservable()  
    }, 3600 * 1000);  
}  
  
buildRequestObservable() {  
    return this.httpClient  
        .get<Book[]>(catalogUrl, {params: catalogUrlParams})  
        .pipe(  
            shareReplay(1)  
        );  
}  
  
getList(): Observable<Book[]> {  
    return this.list$;  
}
```

1 requête HTTP max / 1h

# Exemple : conserver les données

```
list$: Observable<Book[]>;  
  
constructor(private httpClient: HttpClient, private ngZone: NgZone) {  
  this.list$ = this.buildRequestObservable();  
  this.ngZone.runOutsideAngular(() => {  
    setInterval(() => {  
      this.ngZone.run(() => {  
        this.list$ = this.buildRequestObservable();  
      });  
    }, 3600 * 1000);  
}  
  
buildRequestObservable() {  
  return this.httpClient.get<Book[]>(catalogUrl, {params: catalogUrlParams})  
    .pipe(  
      shareReplay(1)  
    );  
}  
  
getList(): Observable<Book[]> {  
  return this.list$;  
}
```

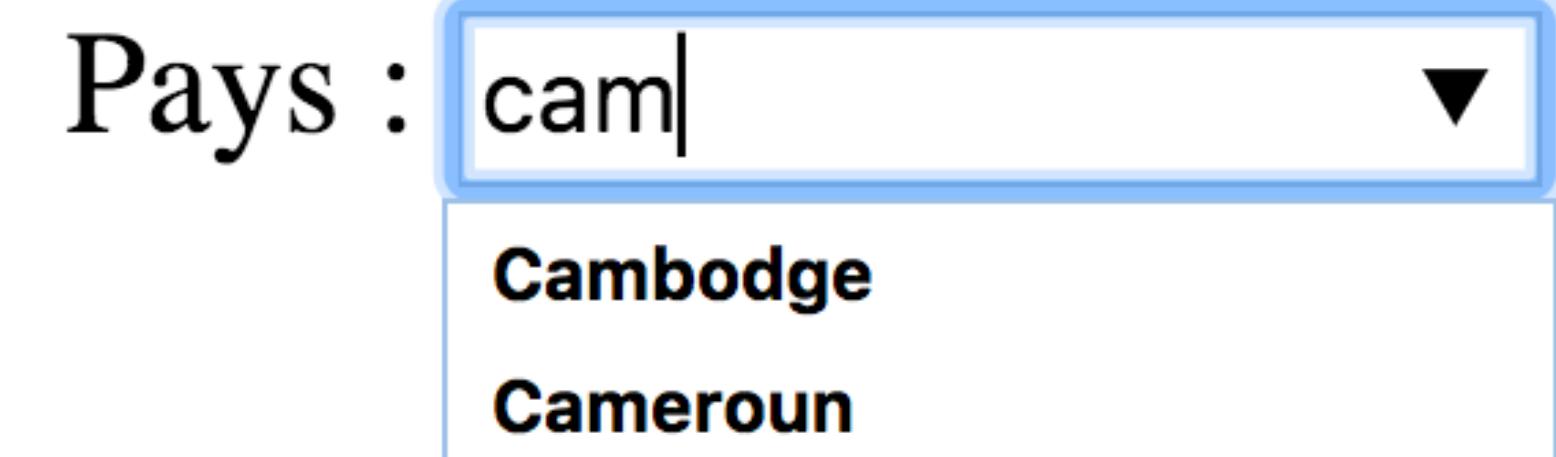
*Angular : NgZone*

# RxJS : les clefs pour comprendre les observables

## 8.3 exemple auto-complete

# Exemple : auto-complete

Auto-complete sur la saisie d'un pays



```
this.countryList$ = this.countryControl.valueChanges
  .pipe(
    map(name => name.trim()),
    filter(name => length >= 2),
    debounceTime(200),
    distinctUntilChanged(),
    switchMap(name => this.countryService.search(name))
  );
```

# Exemple : auto-complete

```
search(name: string): Observable<string[]> {
    name = name && name.trim();
    if (name) {
        const url = 'https://restcountries.eu/rest/v2/name/';
        return this.httpClient.get<Country[]>(url + name)
            .pipe(
                map(countries =>
                    countries.map(country => country.translations.fr)
                ),
                catchError(error => of([]))
            );
    }
    return of([]);
}
```

# RxJS : les clefs pour comprendre les observables



Liens : Rx Vizualizer  
Rx Marbles

<https://rxviz.com/>  
<http://rxmarbles.com/>

Slides : <https://tchatel.github.io/conf-rxjs/RxJS.pdf>

@ThierryChatel