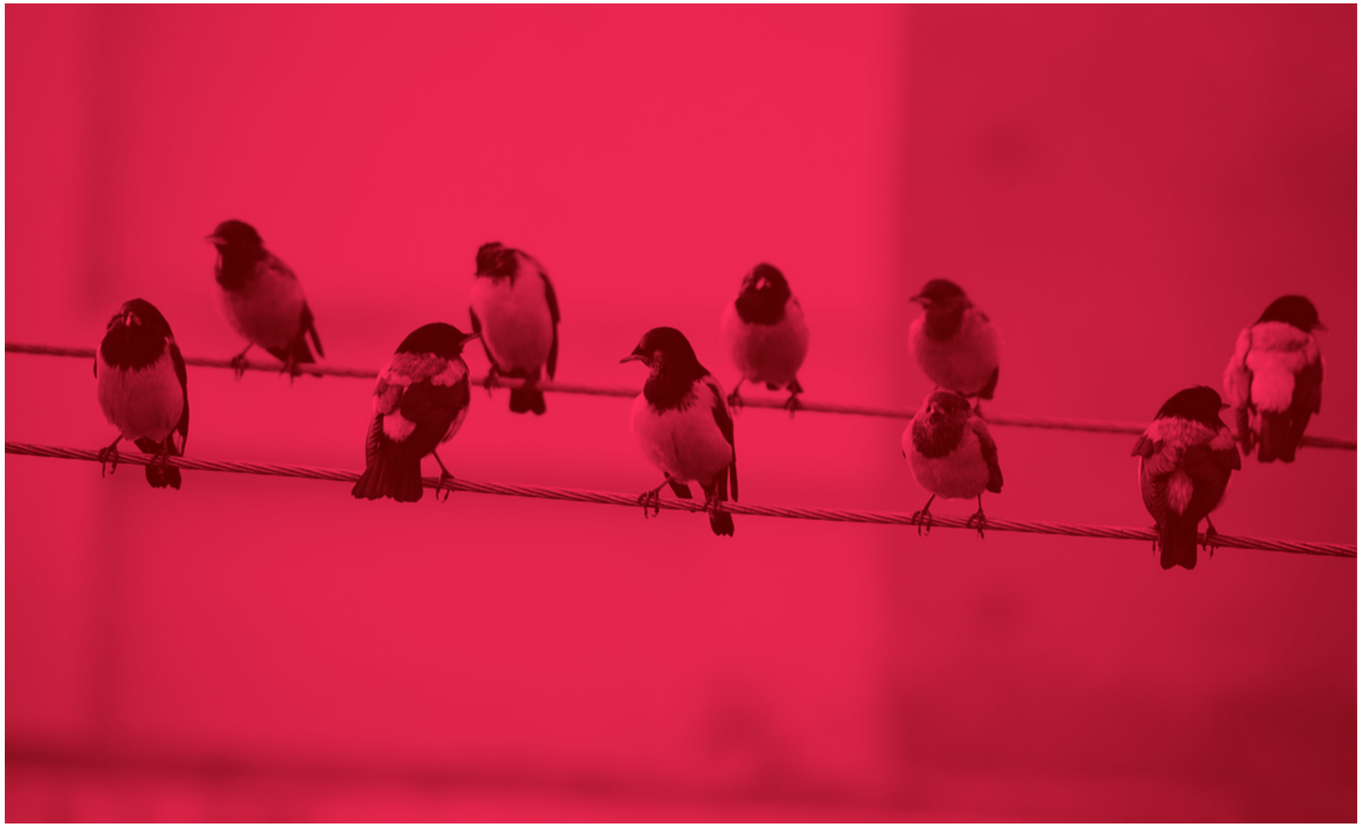
# RxJS : les clefs pour comprendre les observables

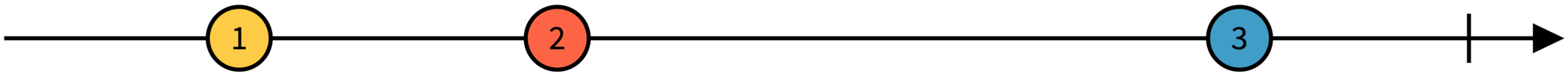

Thierry Chatel @ThierryChatel

# RxJS : les clefs pour comprendre les observables


## 1. présentation des observables

# Observable

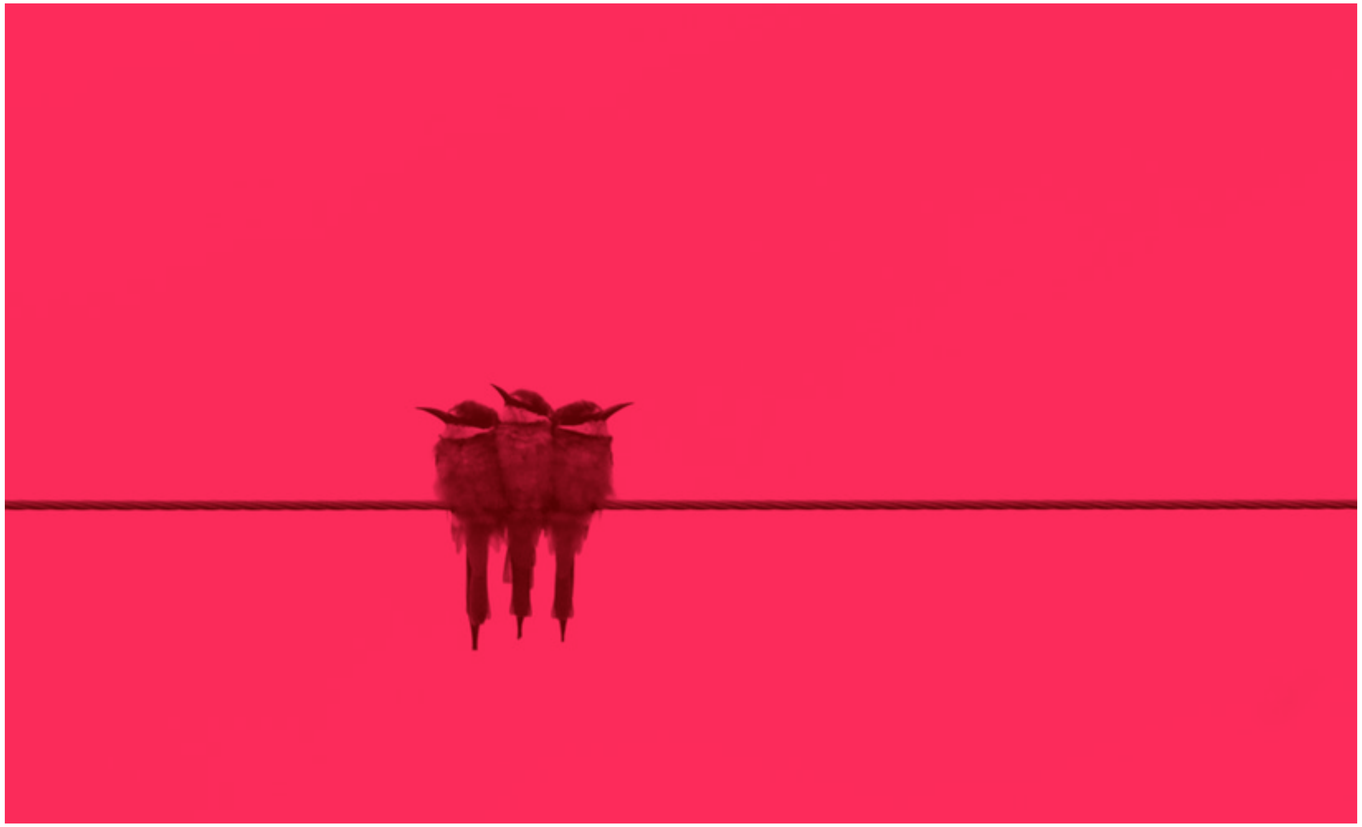


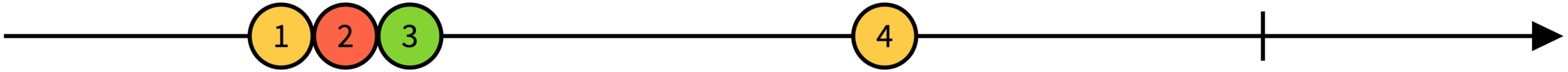


# Observable

# Observable

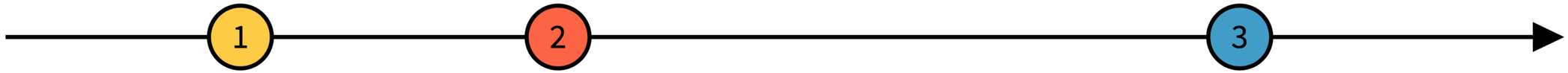
# Observable





# Observable

## next

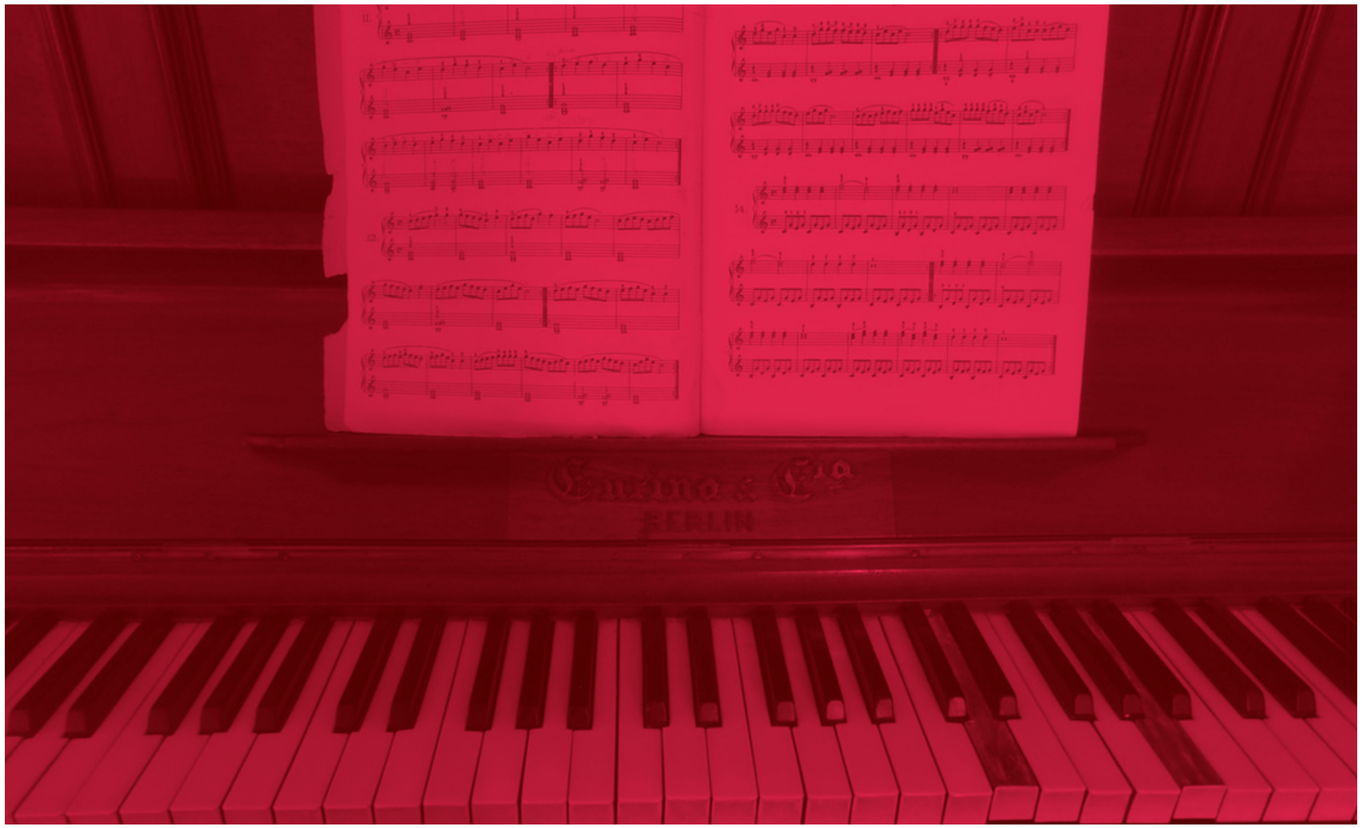


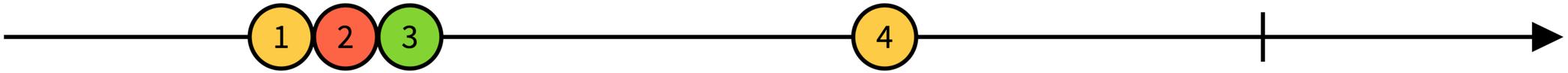

## error


## complete

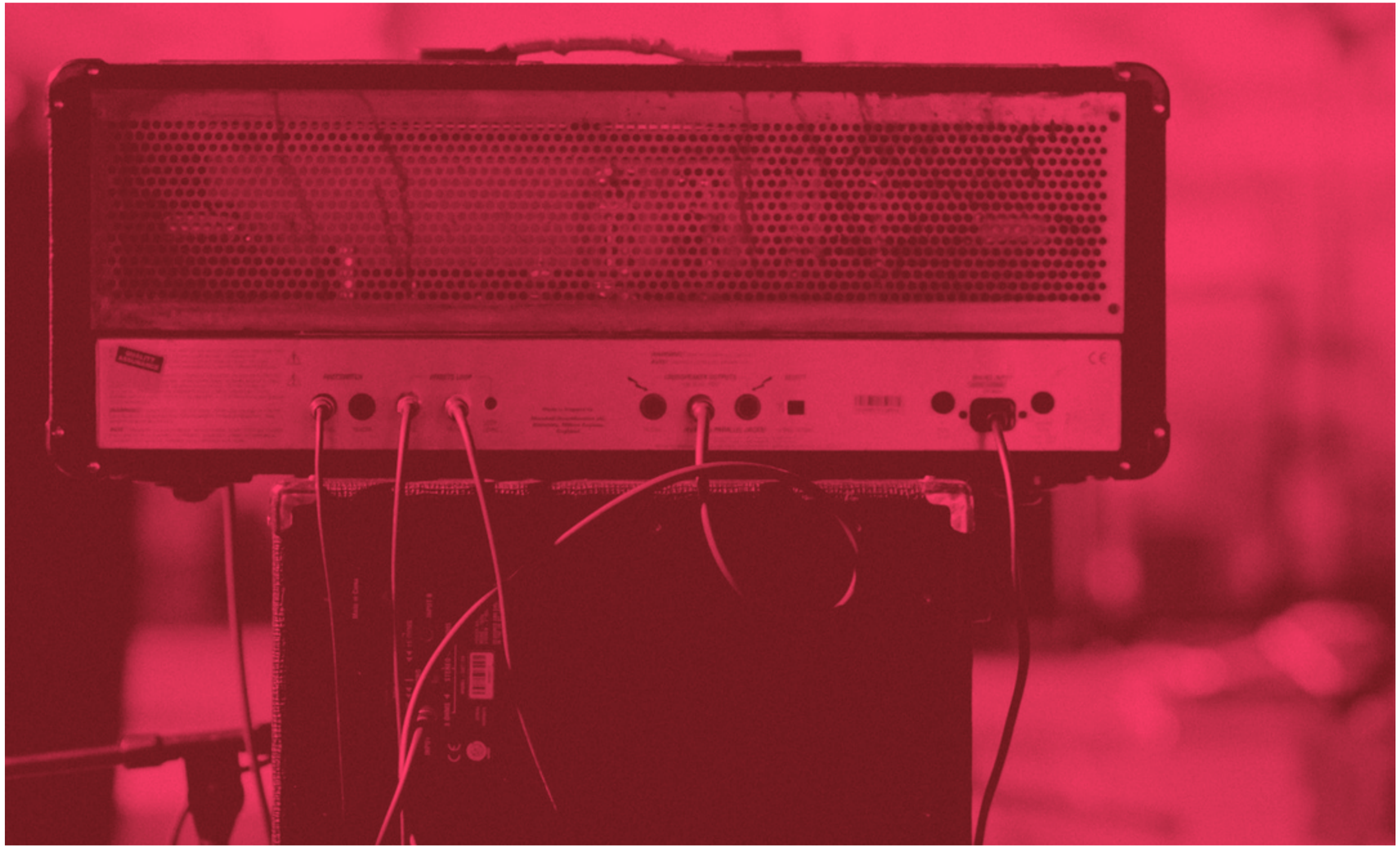

Impossible !

# Opérateurs

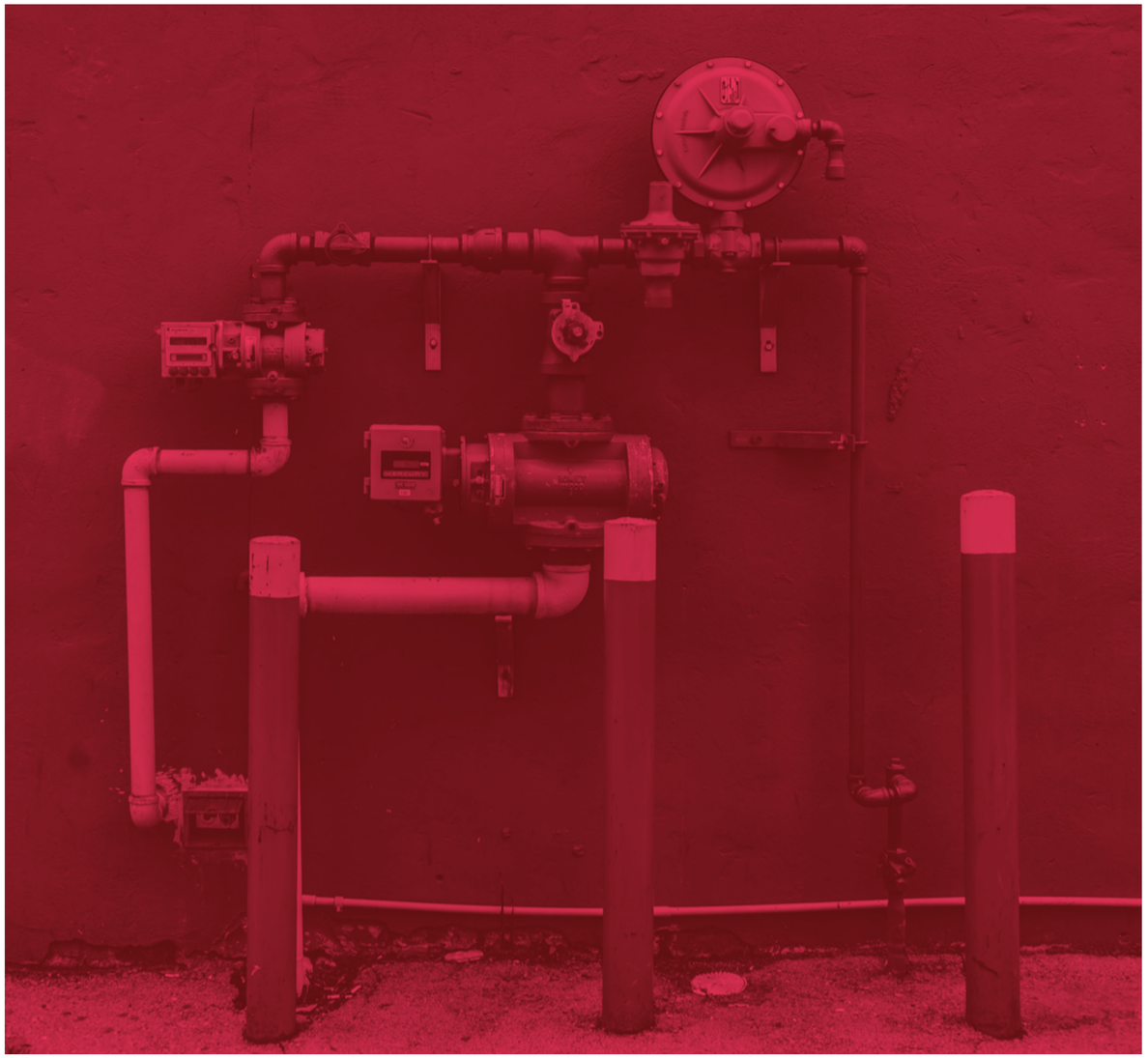




# Opérateurs



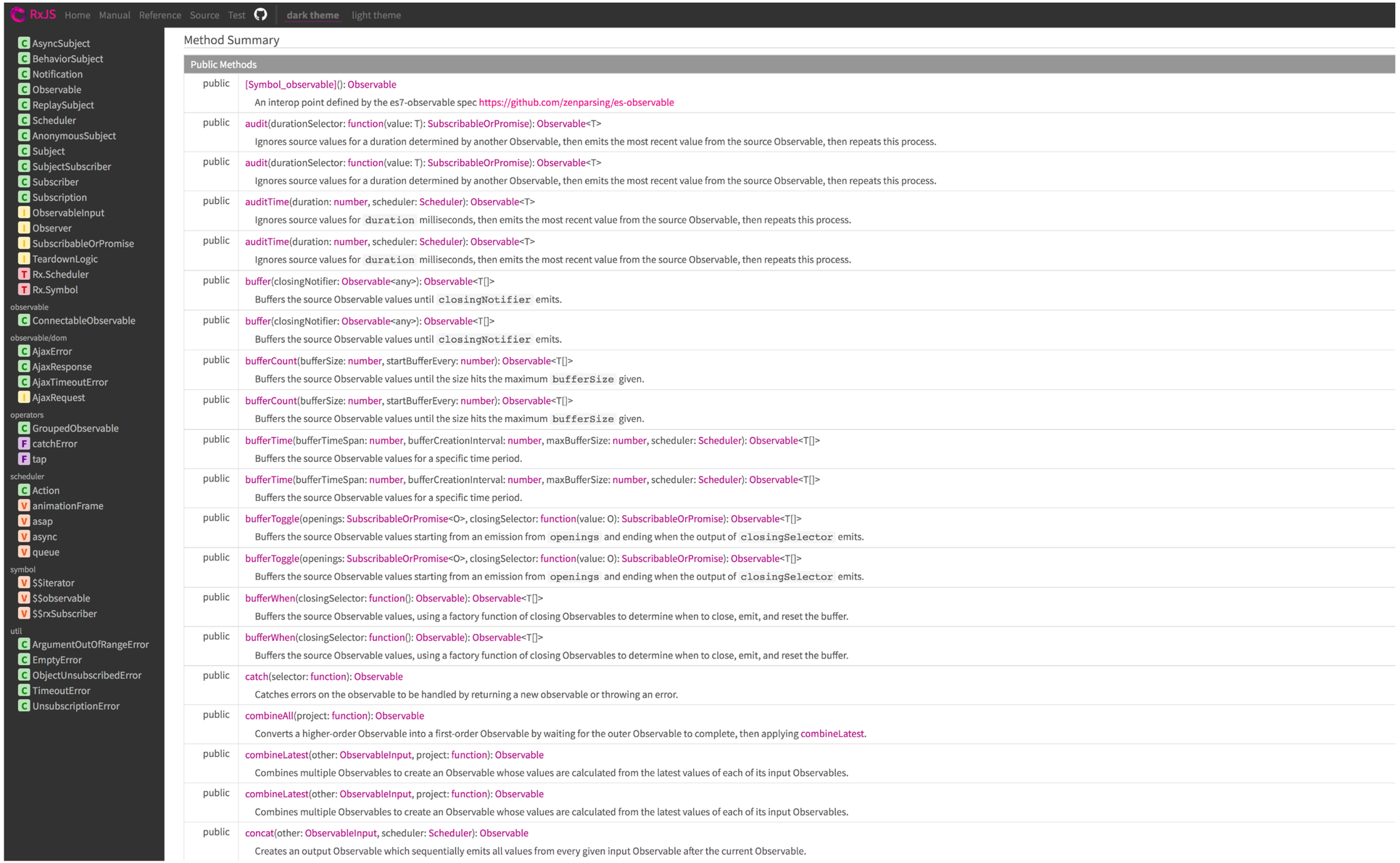
# Opérateurs



# 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
C AsyncSubject
C BehaviorSubject
C Notification
C Observable
C ReplaySubject
C Scheduler
C AnonymousSubject
C Subject
C SubjectSubscriber
C Subscriber
C Subscription
I ObservableInput
I Observer
I SubscribableOrPromise
I TeardownLogic
T Rx.Scheduler
T Rx.Symbol
observable
C ConnectableObservable
observable/dom
C AjaxError
C AjaxResponse
C AjaxTimeoutError
I AjaxRequest
operators
C GroupedObservable
F catchError
F tap
scheduler
C Action
V animationFrame
V asap
V async
V queue
symbol
V $$iterator
V $$observable
V $$rxSubscriber
util
C ArgumentOutOfRangeError
C EmptyError
C ObjectUnsubscribedError
C TimeoutError
C UnsubscriptionError
Method Summary
Public Methods
public [Symbol_observable](): Observable
An interop point defined by the es7-observable spec https://github.com/zenparsing/es-observable
public audit(durationSelector: function(value: T): SubscribableOrPromise): Observable<T>
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 audit(durationSelector: function(value: T): SubscribableOrPromise): Observable<T>
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 auditTime(duration: number, scheduler: Scheduler): Observable<T>
Ignores source values for duration milliseconds, then emits the most recent value from the source Observable, then repeats this process.
public auditTime(duration: number, scheduler: Scheduler): Observable<T>
Ignores source values for duration milliseconds, then emits the most recent value from the source Observable, then repeats this process.
public buffer(closingNotifier: Observable<any>): Observable<T[]>
Buffers the source Observable values until closingNotifier emits.
public buffer(closingNotifier: Observable<any>): Observable<T[]>
Buffers the source Observable values until closingNotifier emits.
public bufferCount(bufferSize: number, startBufferEvery: number): Observable<T[]>
Buffers the source Observable values until the size hits the maximum bufferSize given.
public bufferCount(bufferSize: number, startBufferEvery: number): Observable<T[]>
Buffers the source Observable values until the size hits the maximum bufferSize given.
public bufferTime(bufferTimeSpan: number, bufferCreationInterval: number, maxBufferSize: number, scheduler: Scheduler): Observable<T[]>
Buffers the source Observable values for a specific time period.
public bufferTime(bufferTimeSpan: number, bufferCreationInterval: number, maxBufferSize: number, scheduler: Scheduler): Observable<T[]>
Buffers the source Observable values for a specific time period.
public bufferToggle(openings: SubscribableOrPromise<O>, closingSelector: function(value: O): SubscribableOrPromise): Observable<T[]>
Buffers the source Observable values starting from an emission from openings and ending when the output of closingSelector emits.
public bufferToggle(openings: SubscribableOrPromise<O>, closingSelector: function(value: O): SubscribableOrPromise): Observable<T[]>
Buffers the source Observable values starting from an emission from openings and ending when the output of closingSelector emits.
public bufferWhen(closingSelector: function(): Observable): Observable<T[]>
Buffers the source Observable values, using a factory function of closing Observables to determine when to close, emit, and reset the buffer.
public bufferWhen(closingSelector: function(): Observable): Observable<T[]>
Buffers the source Observable values, using a factory function of closing Observables to determine when to close, emit, and reset the buffer.
public catch(selector: function): Observable
Catches errors on the observable to be handled by returning a new observable or throwing an error.
public combineAll(project: function): Observable
Converts a higher-order Observable into a first-order Observable by waiting for the outer Observable to complete, then applying combineLatest.
public combineLatest(other: ObservableInput, project: function): Observable
Combines multiple Observables to create an Observable whose values are calculated from the latest values of each of its input Observables.
public combineLatest(other: ObservableInput, project: function): Observable
Combines multiple Observables to create an Observable whose values are calculated from the latest values of each of its input Observables.
public concat(other: ObservableInput, scheduler: Scheduler): Observable
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

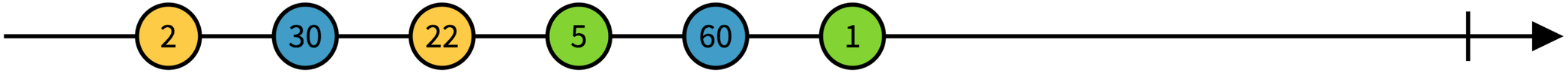
```
[2, 30, 22, 5, 60, 1]
  .filter(x => x > 10)                  // [30, 22, 60]


[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

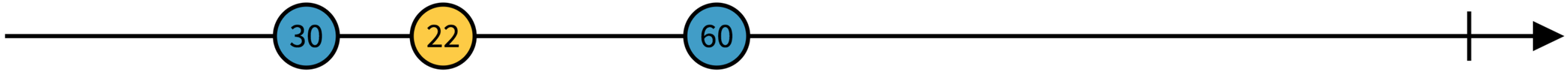
```
[2, 30, 22, 5, 60, 1]
  .filter(x => x > 10)                    // [30, 22, 60]
```

# Opérateurs des observables

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




```
filter(x => x > 10)
```

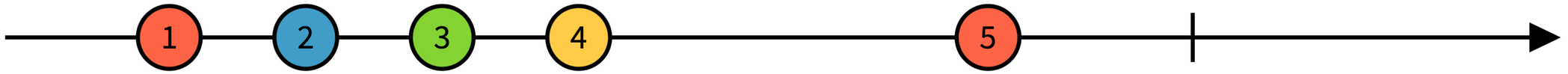


# Opérateurs des observables

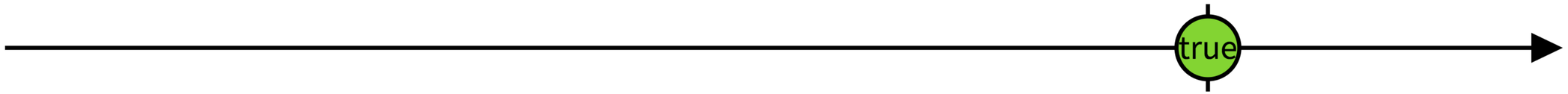
```
[1, 2, 3, 4, 5]
  .every(x => x < 10)                    // true
```

# Opérateurs des observables

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




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



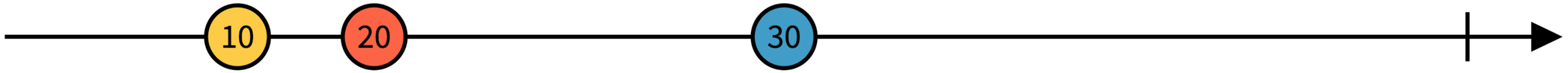
# Opérateurs des observables

```
[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)
```

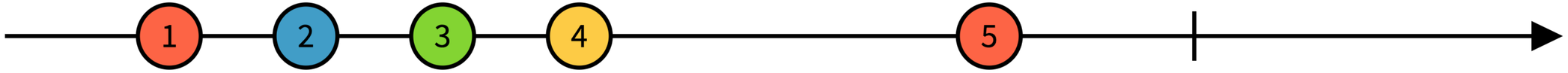


# Opérateurs des observables

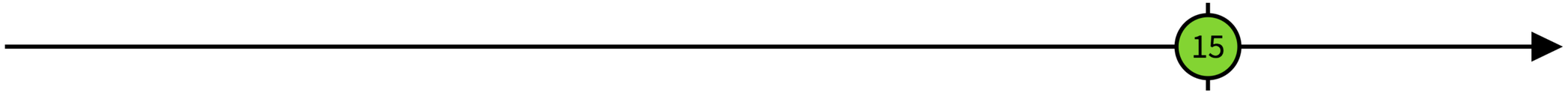
```
[1, 2, 3, 4, 5]
  .map(x => 10 * x)                    // [10, 20, 30, 40]
  .reduce((t, x) => t + x, 0)          // 55
```

# 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)
```

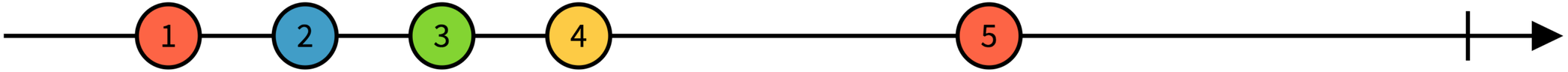


# Opérateurs des observables

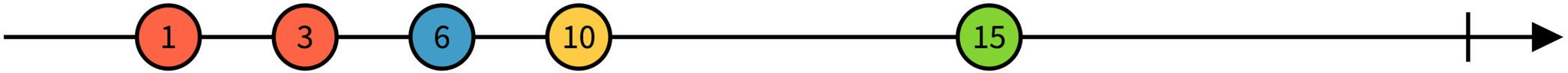
```
[1, 2, 3, 4, 5]
  .map(x => 10 * x)                    // [10, 20, 30, 40]
  .reduce((t, x) => t + x, 0)          // 55
```

# 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)
```

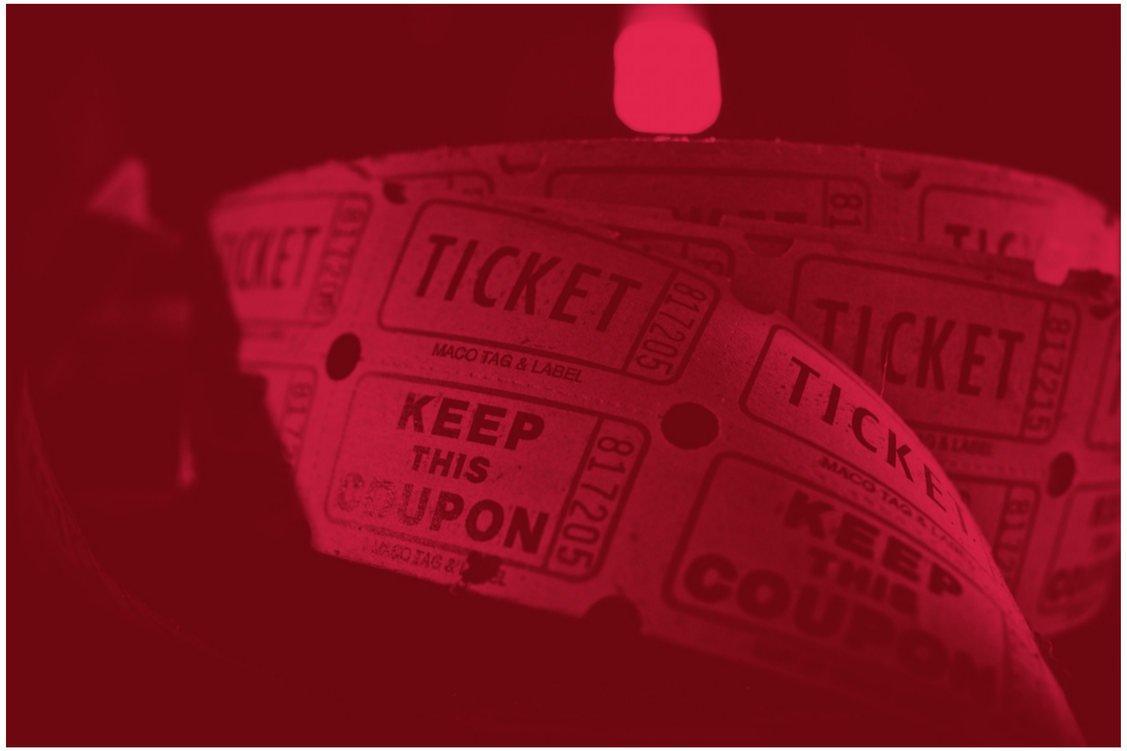


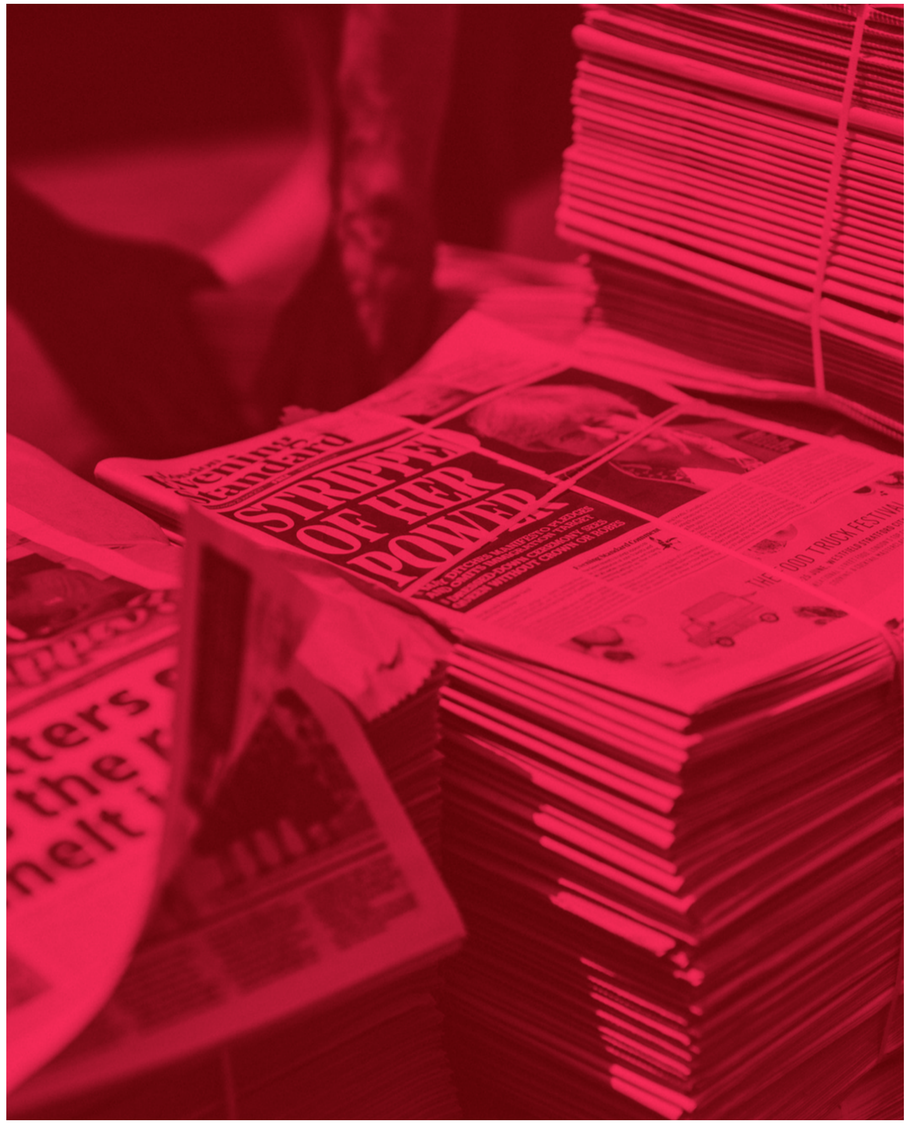
# Opérateurs des observables

Opérateur RxJS :

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

# 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();
```

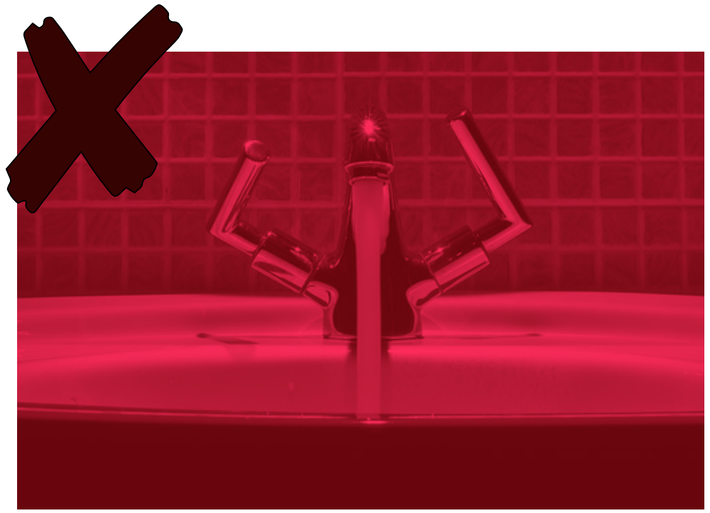
# Souscription

# Comportement

**COLD**

VS

**HOT**



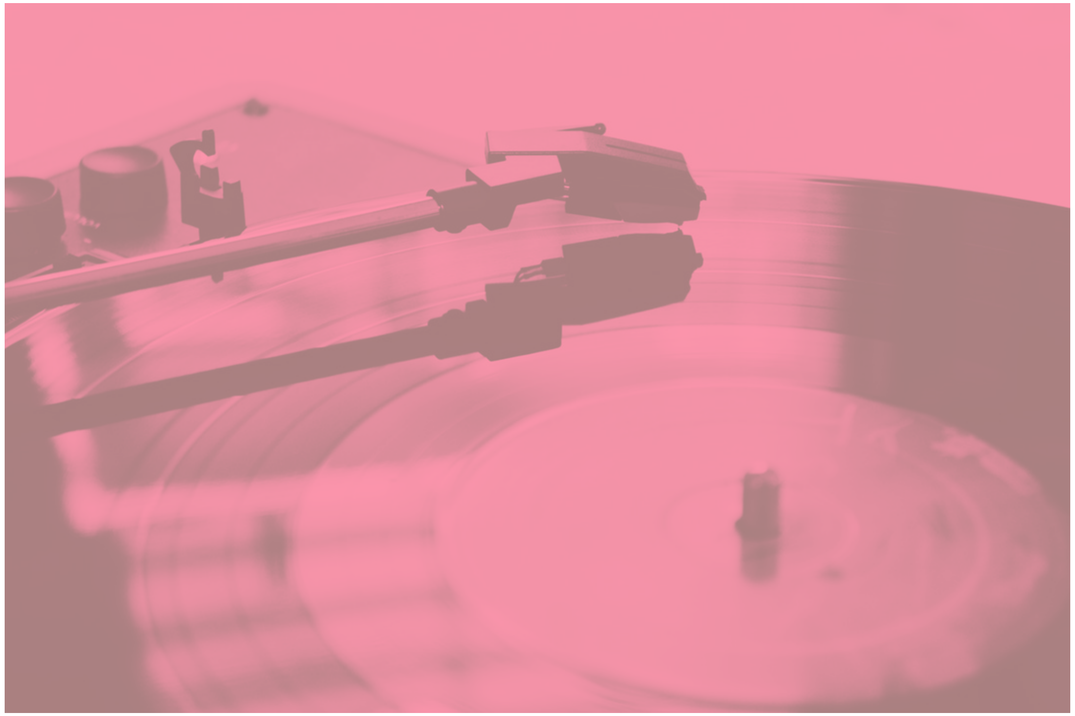
# Comportement

COLD

HOT

# Comportement

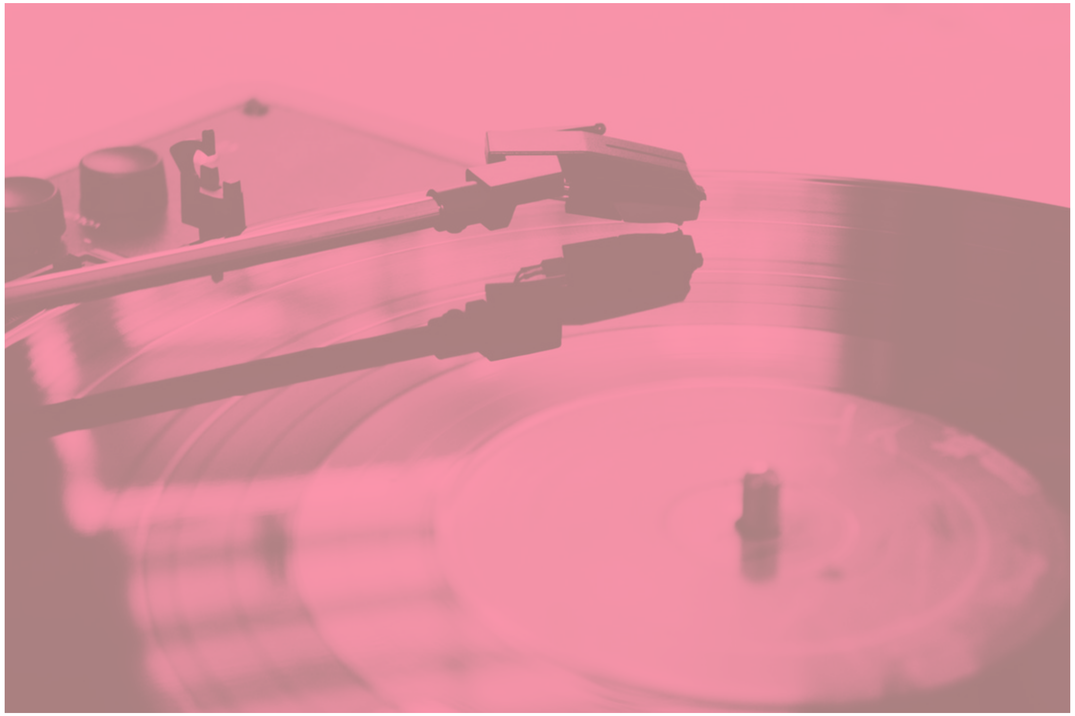
COLD



HOT

# Comportement

**COLD** = unicast



**HOT** = multicasted

# Comportement

# Comportement

# Comportement

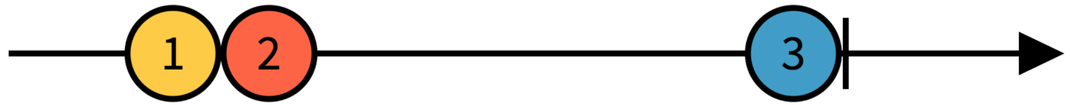
***HTTP***

# Création : COLD

## create()

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

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



# Création : COLD

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

**empty()**

**never()**

**throwError(error)**

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

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

# Création : HOT

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

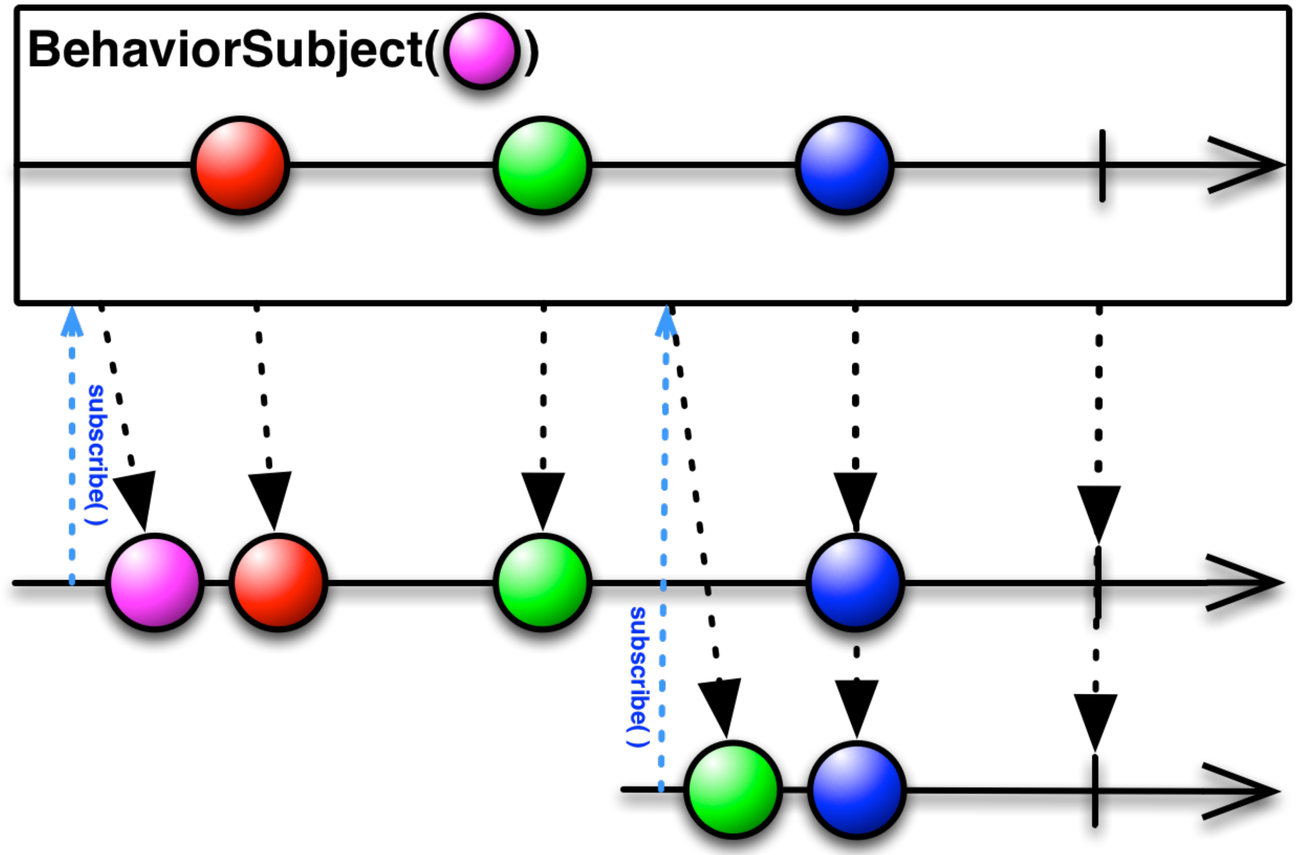
```
const subject = new Subject<number>();

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();
}
```

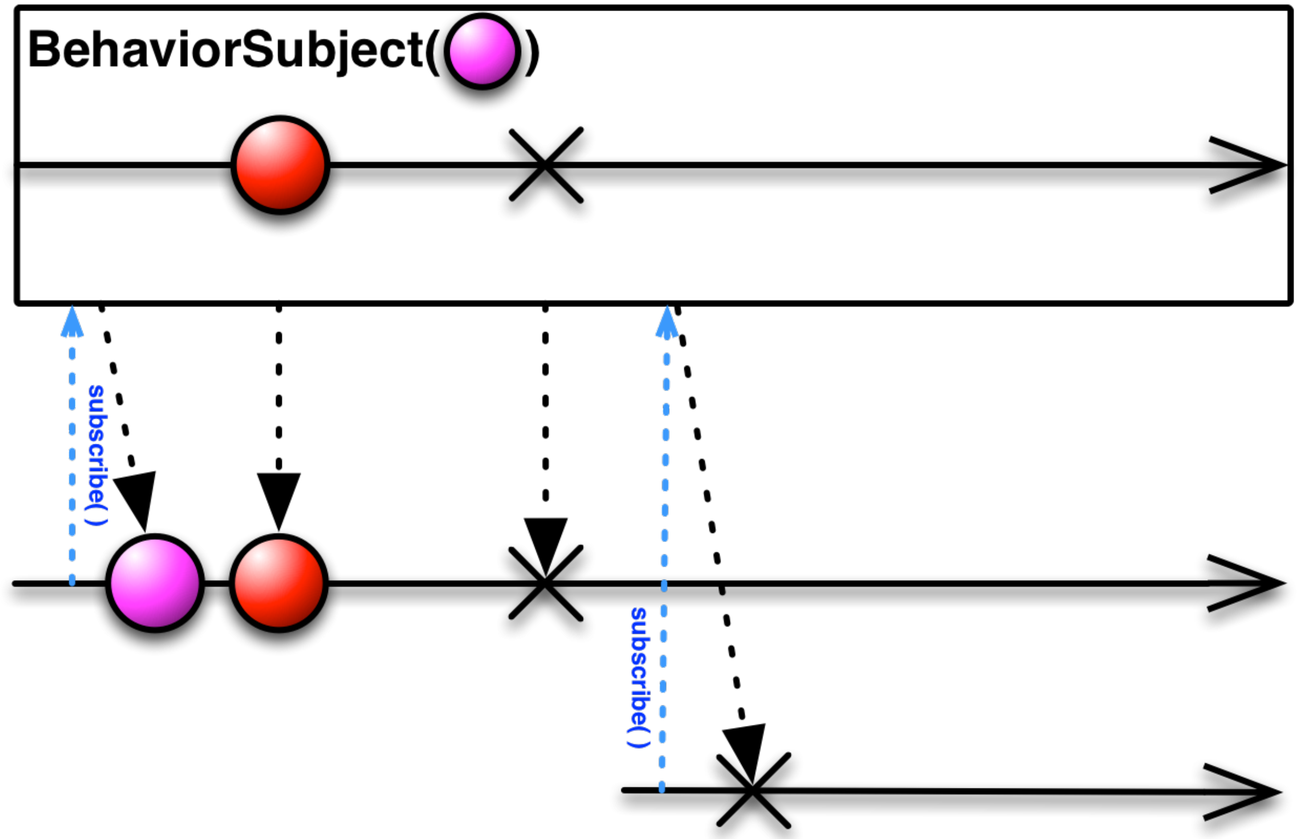
# Création : HOT

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



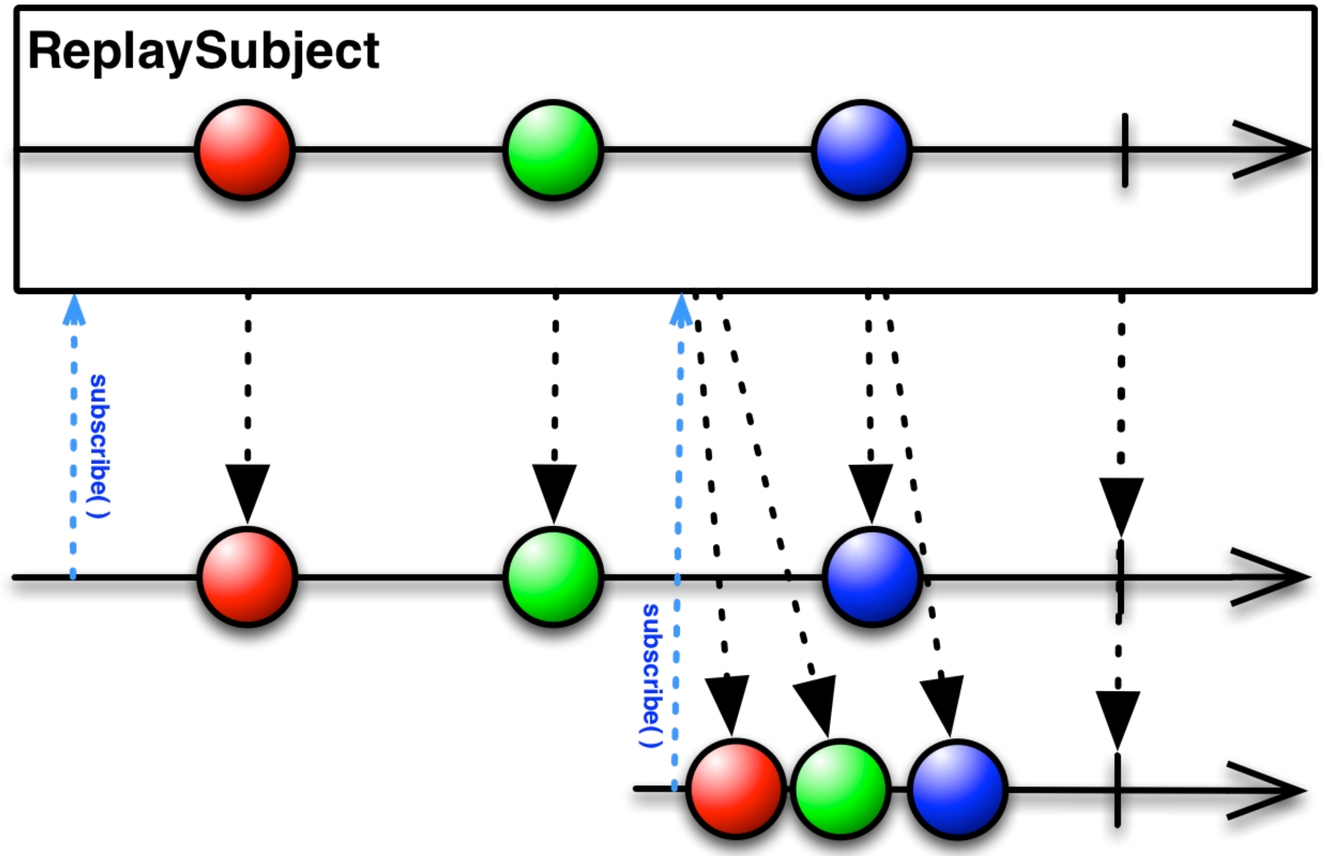
# Création : HOT

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



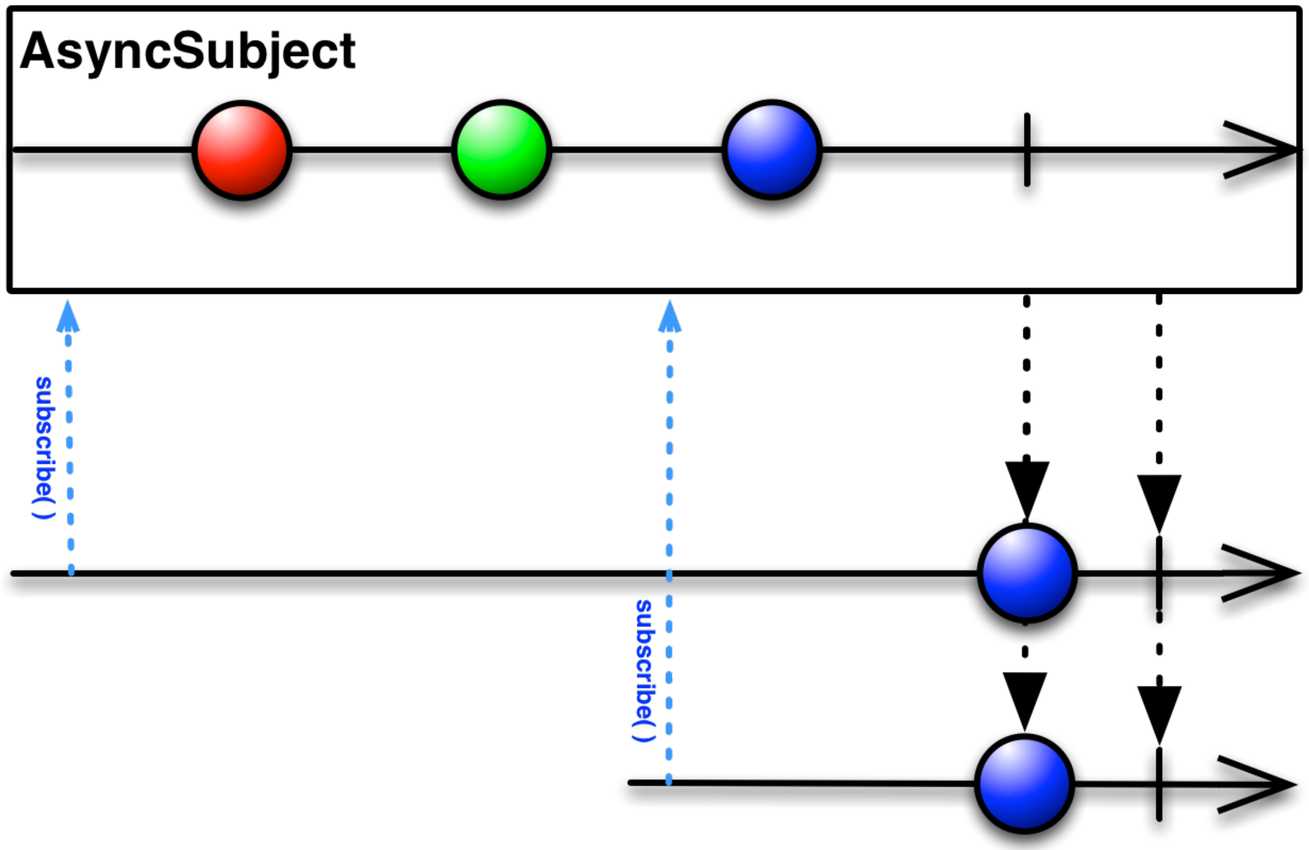
# Création : HOT

**ReplaySubject** : dernières valeurs en buffer



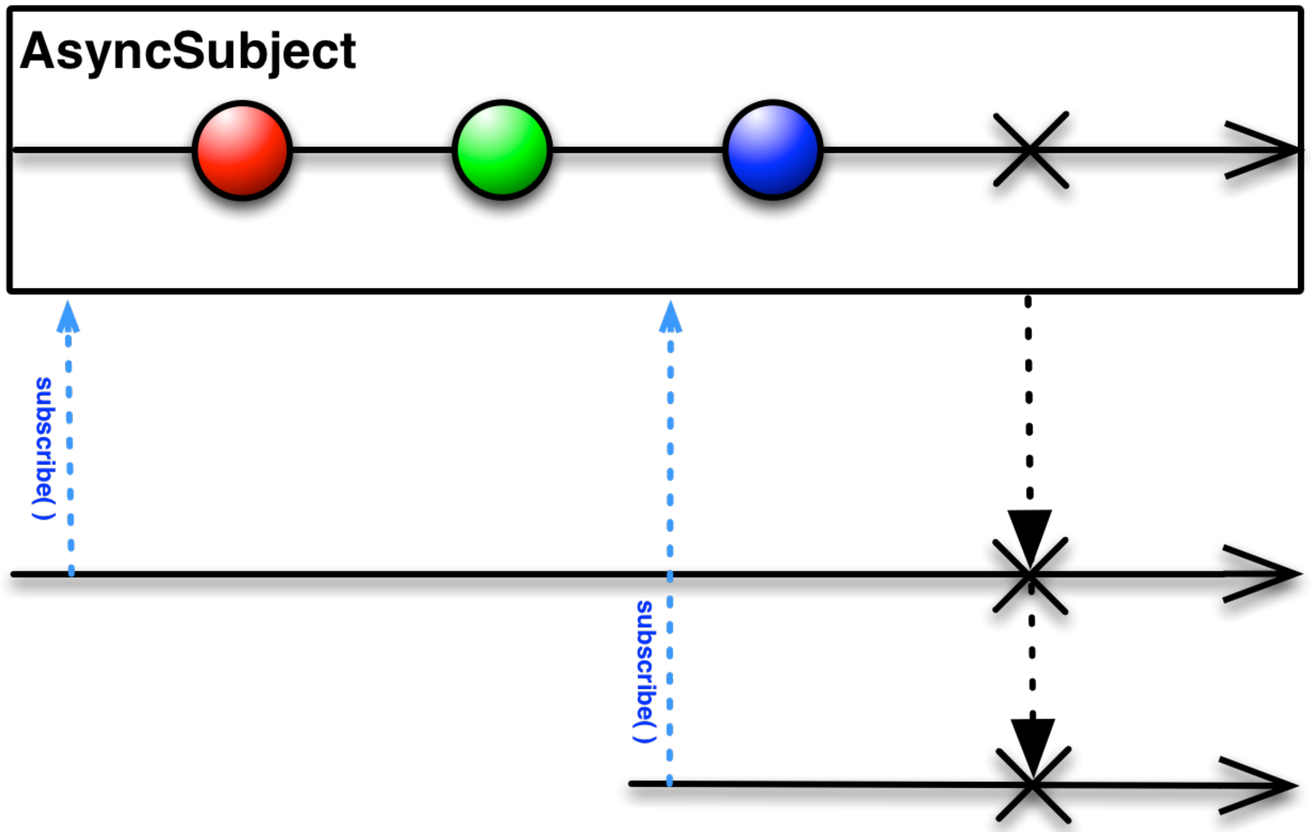
# Création : HOT

**AsyncSubject** : résultat d'une opération asynchrone



# Création : HOT

**AsyncSubject** : résultat d'une opération asynchrone



# Création : HOT

```
fromPromise(p)

fromEvent(element, 'click')
```

# Transformation : COLD → HOT

**share()**

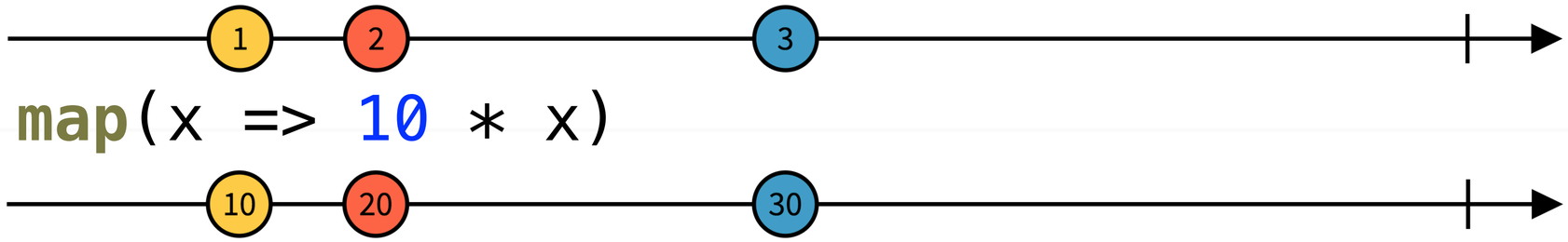
**shareReplay(5)**

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

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

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

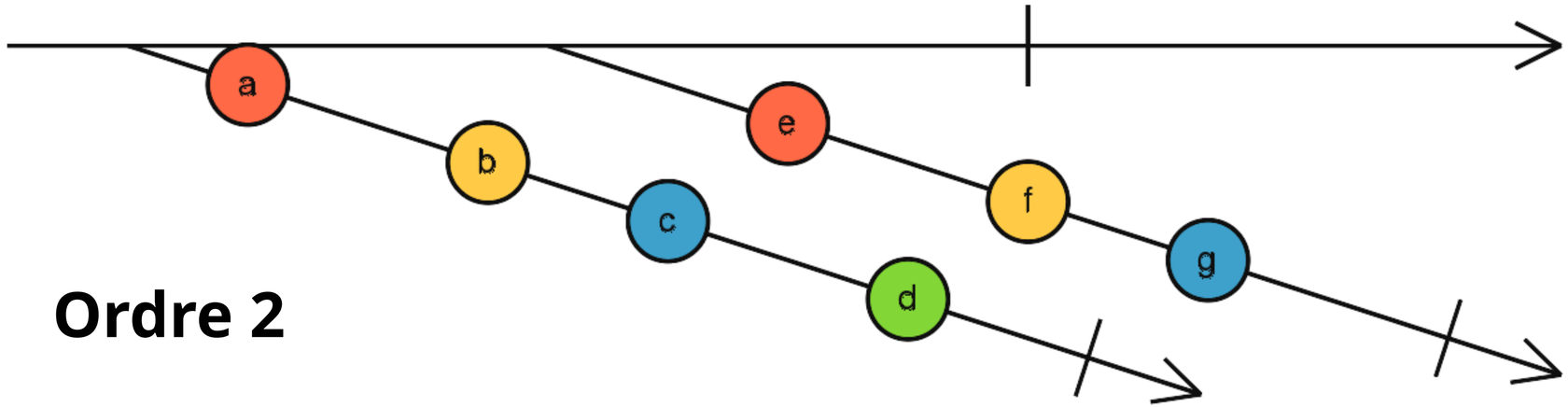
# Ordre 2

*Tableaux de tableaux :*

```
const array = [[1, 2], [3, 4]];
```

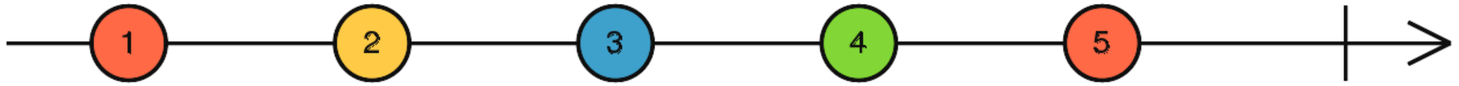



*calcul asynchrone renvoyant un observable :*

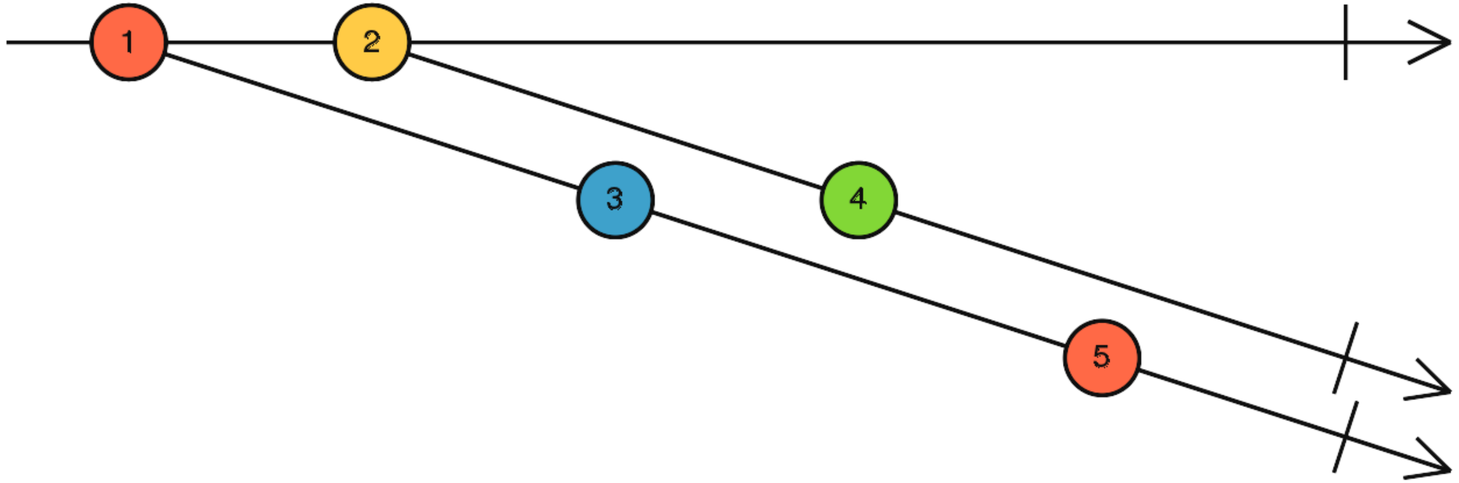



**Ordre 2**

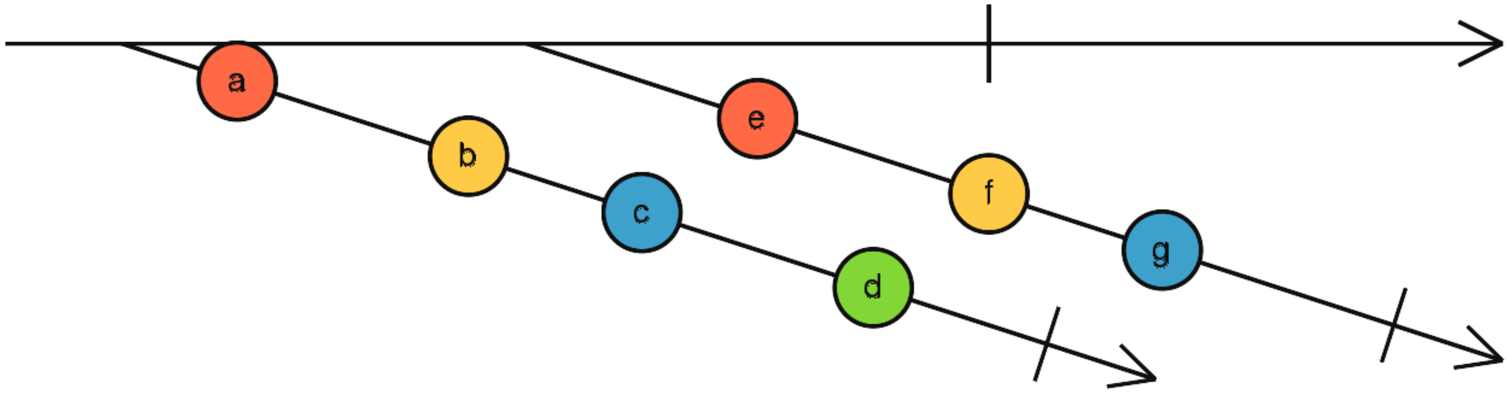
# Ordre 2




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



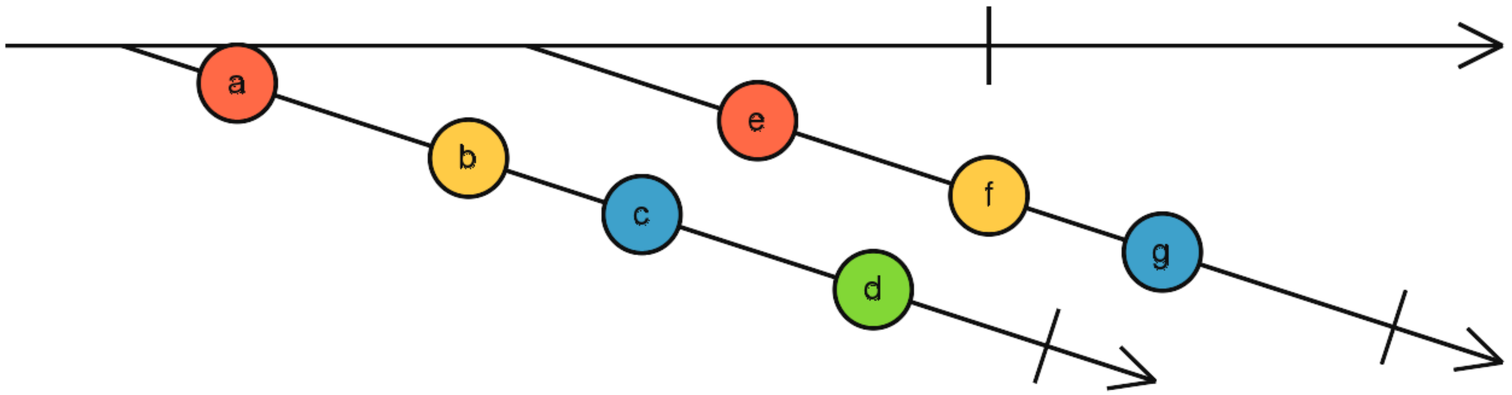
# Ordre 2 : aplatir




*Requêtes simultanées :*

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

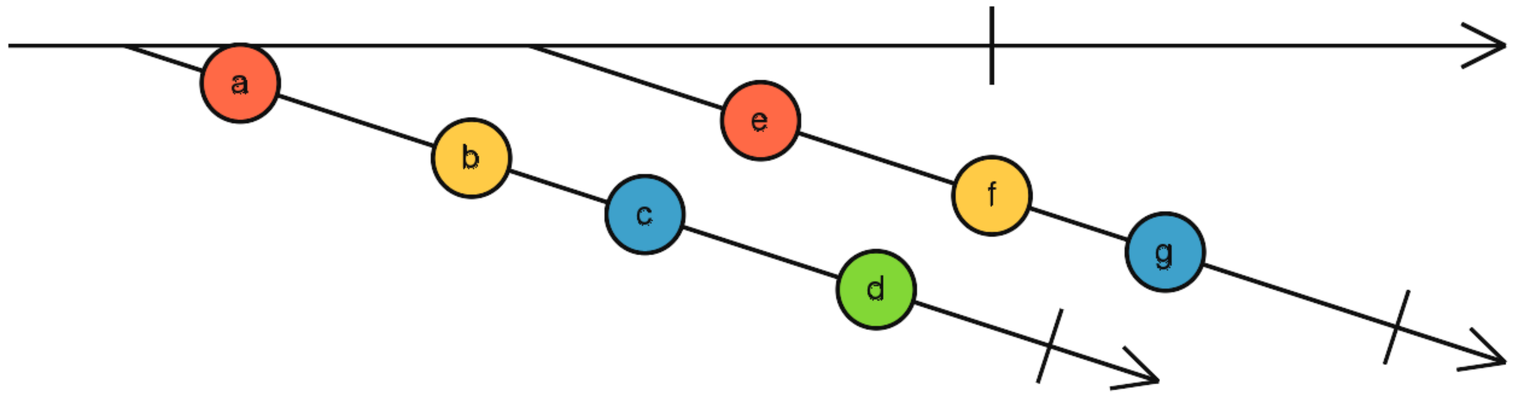
# Ordre 2 : aplatir




*Requêtes simultanées :*

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

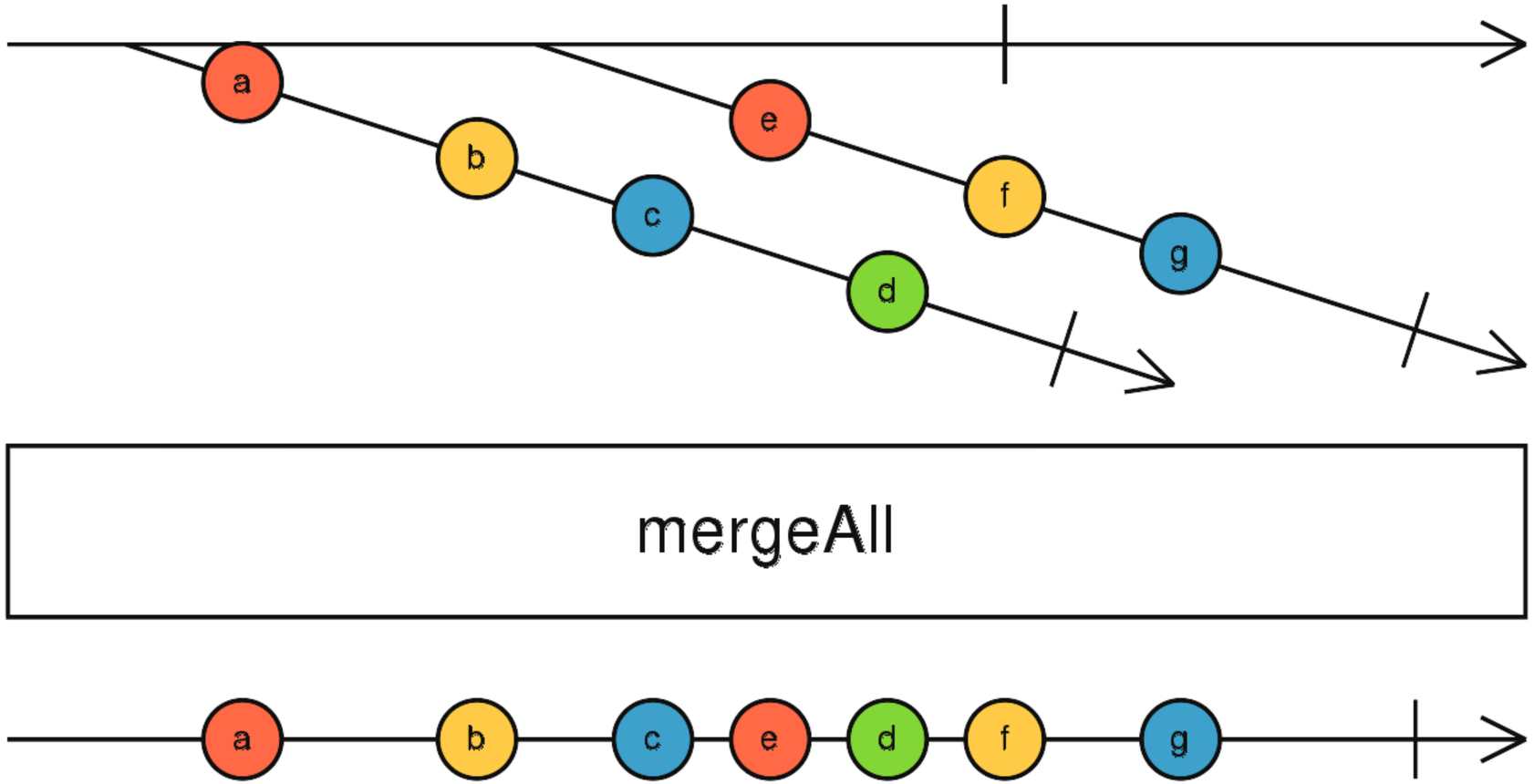
# Ordre 2 : aplatir



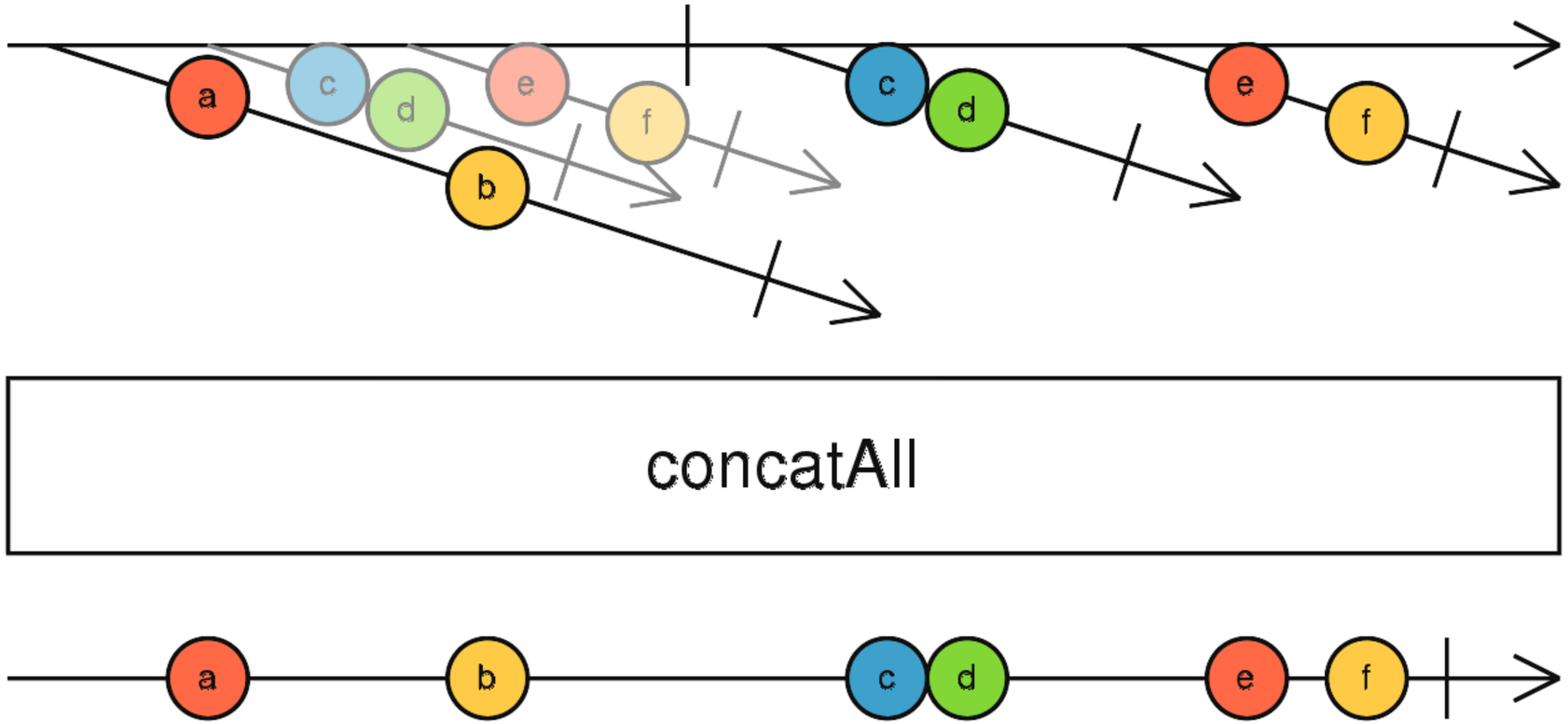

*Requêtes simultanées :*

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

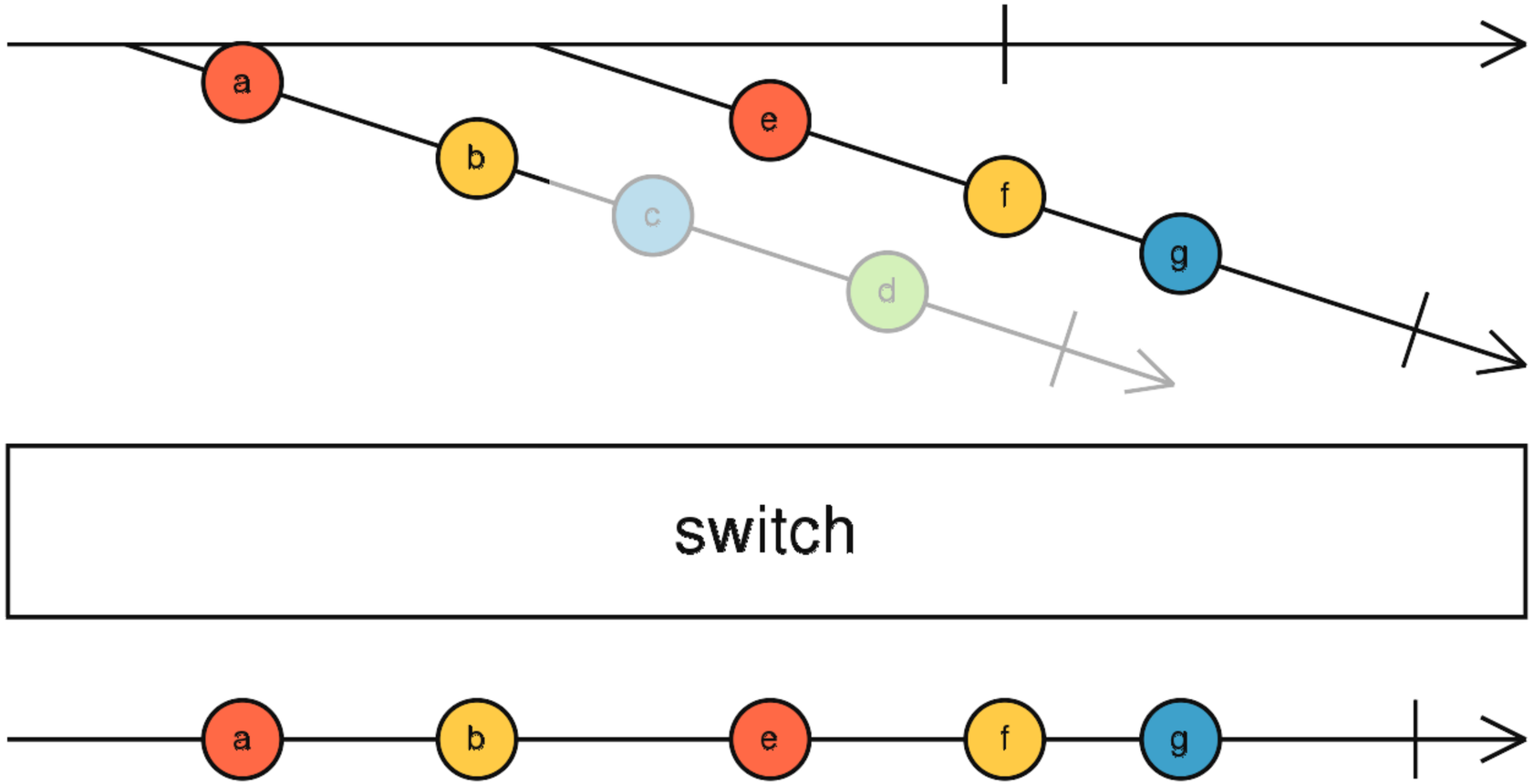
# Ordre 2 : aplatir



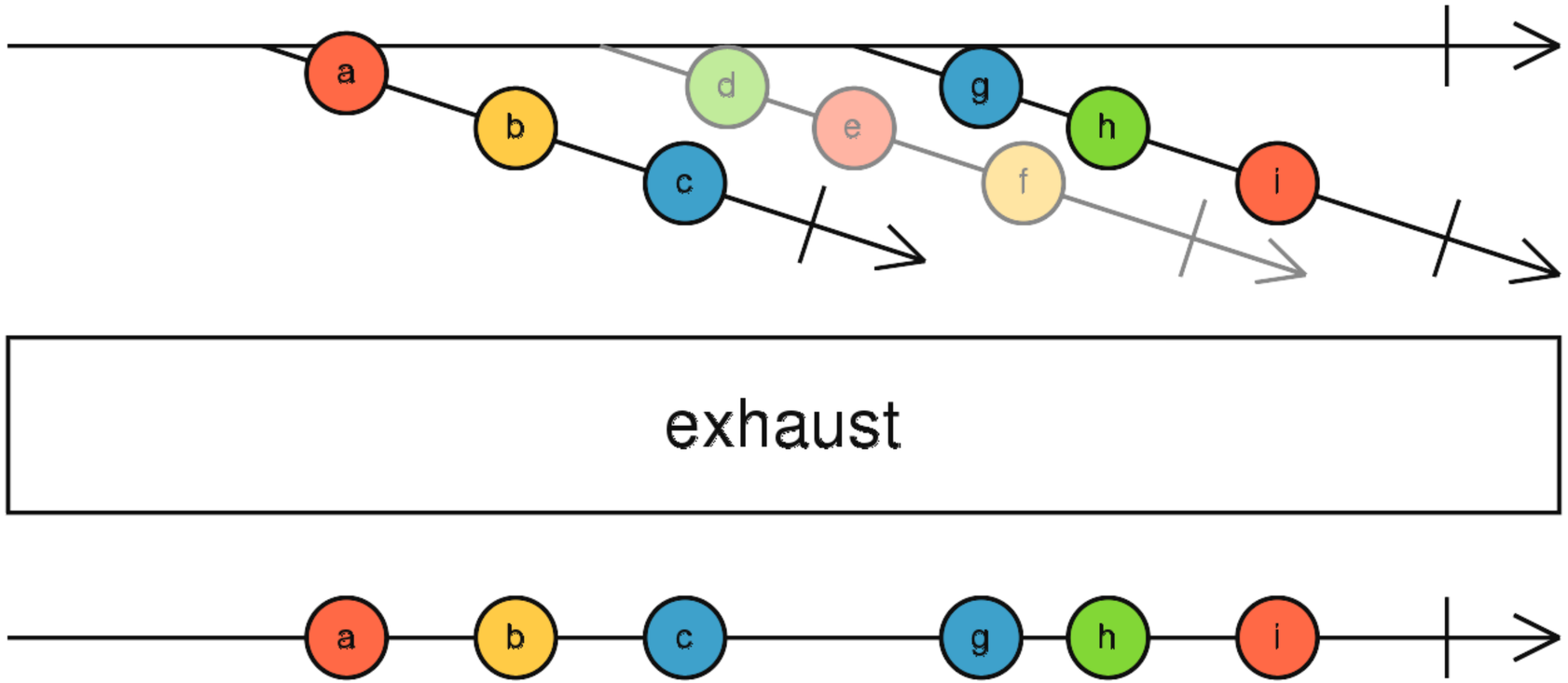
# Ordre 2 : aplatir



# Ordre 2 : aplatir



# Ordre 2 : aplatir



# HOW-TO : services

Ne pas souscrire, renvoyer un observable

# HOW-TO : erreurs

Gestion des erreurs :
comme exceptions, remonter au plus haut

catchError()
throwError()

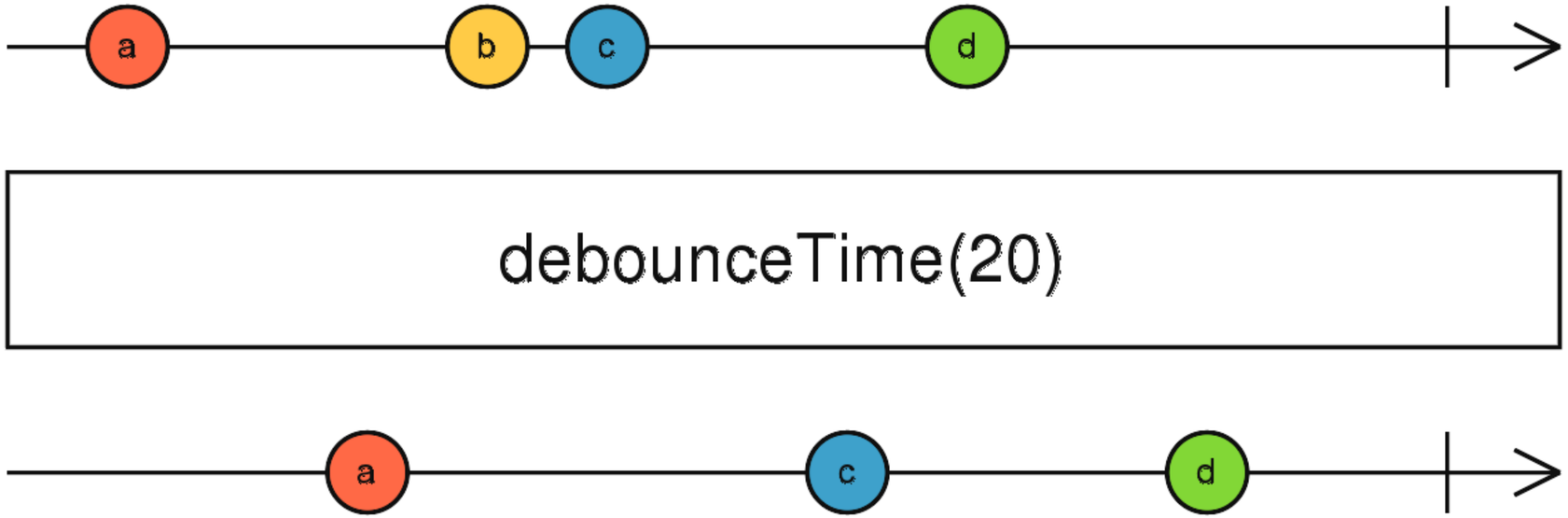
# RxJS : les clefs pour comprendre les observables

## Exemple : double clic

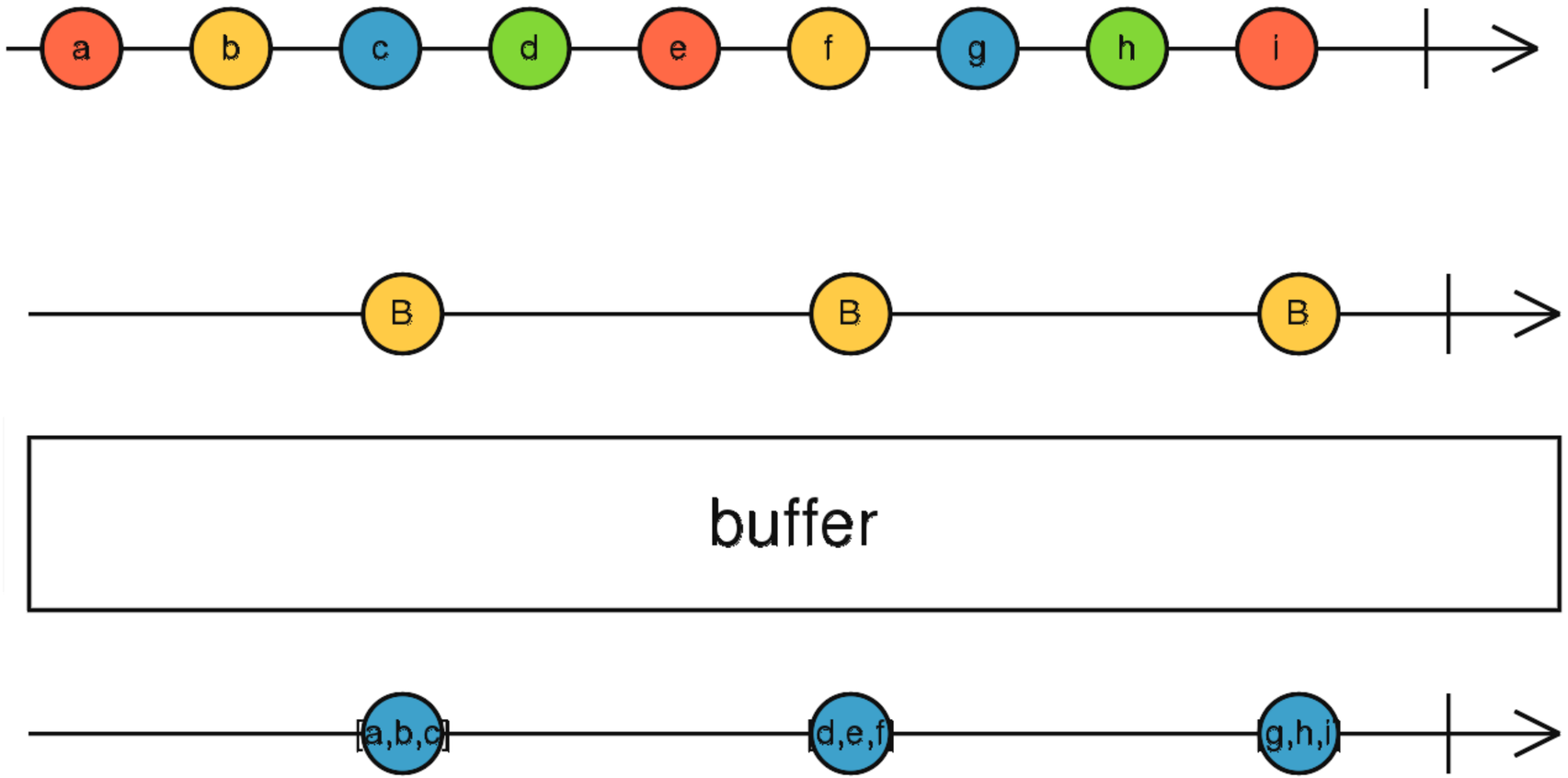
# Exemple : double-clicks

```
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



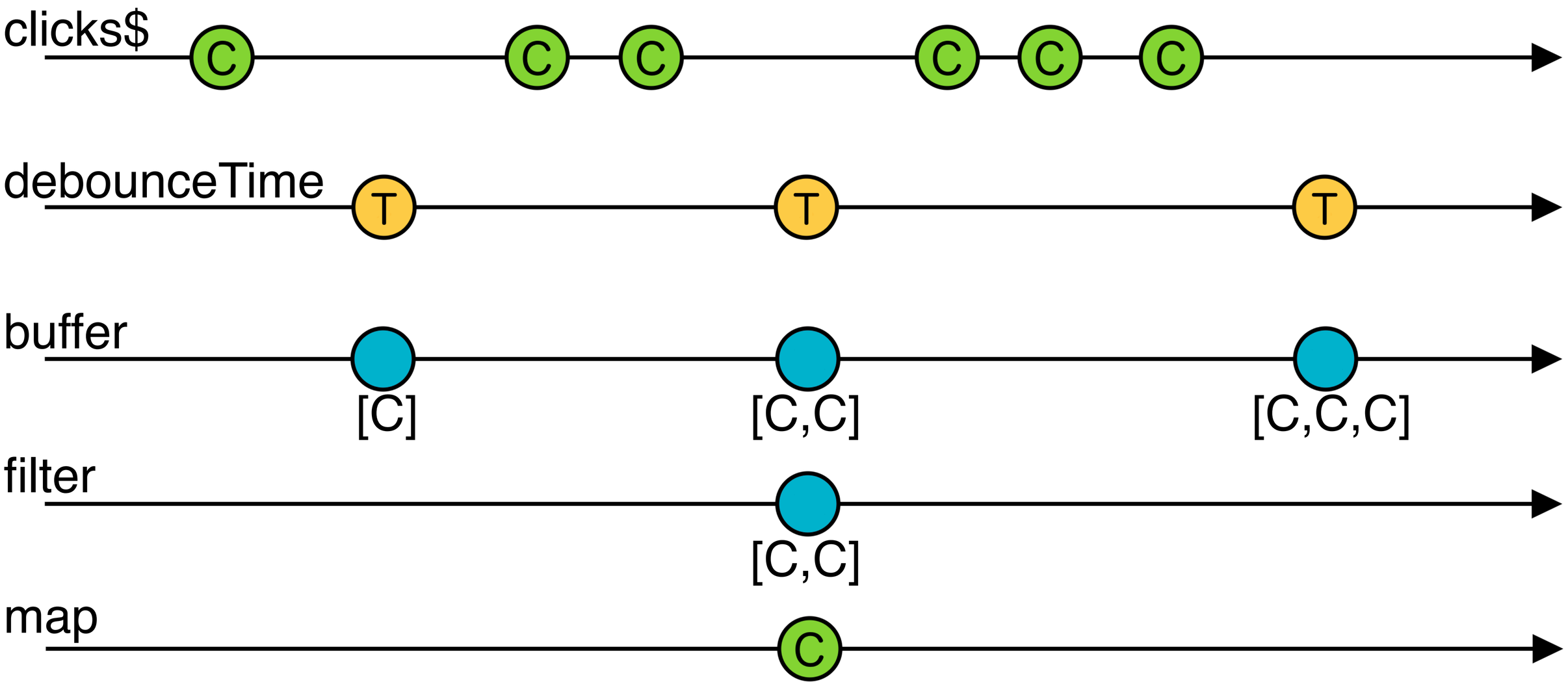
# Exemple : double-clicks



# 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

## Exemple : conserver les données d'une requête

# 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

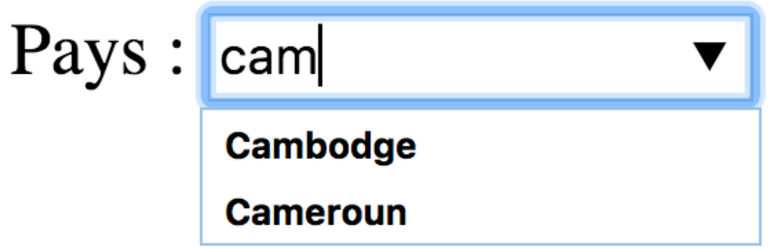
```
list$: Observable<Book[]>;

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$;
}
```

# RxJS : les clefs pour comprendre les observables

## 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 https://rxviz.com/
Rx Marbles http://rxmarbles.com/

Slides : ...

@ThierryChatel