



ManfredSteyer



Reactive Extensions for JavaScript

Manfred Steyer

SOFTWARE*architekt.at*

Contents

- Overview to Observables
- Generating Observables
- Hot vs. Cold Observables
- Piping operators (lookahead)
- Subjects (Pub / Sub)
- Closing Observables

Overview

What are observables?

- Represents (asynchronous) data that is published over time

Observable
„Source“



Operator
(z. B. map)

Observer
„Destination“

Observer

```
myObservable.subscribe(  
    (result) => { ... },  
    (error) => { ... },  
    () => { ... }  
);
```

Option with more than one
parameter is now deprecated!

Observer

```
myObservable.subscribe(  
  (result) => { ... }  
);
```

Observer

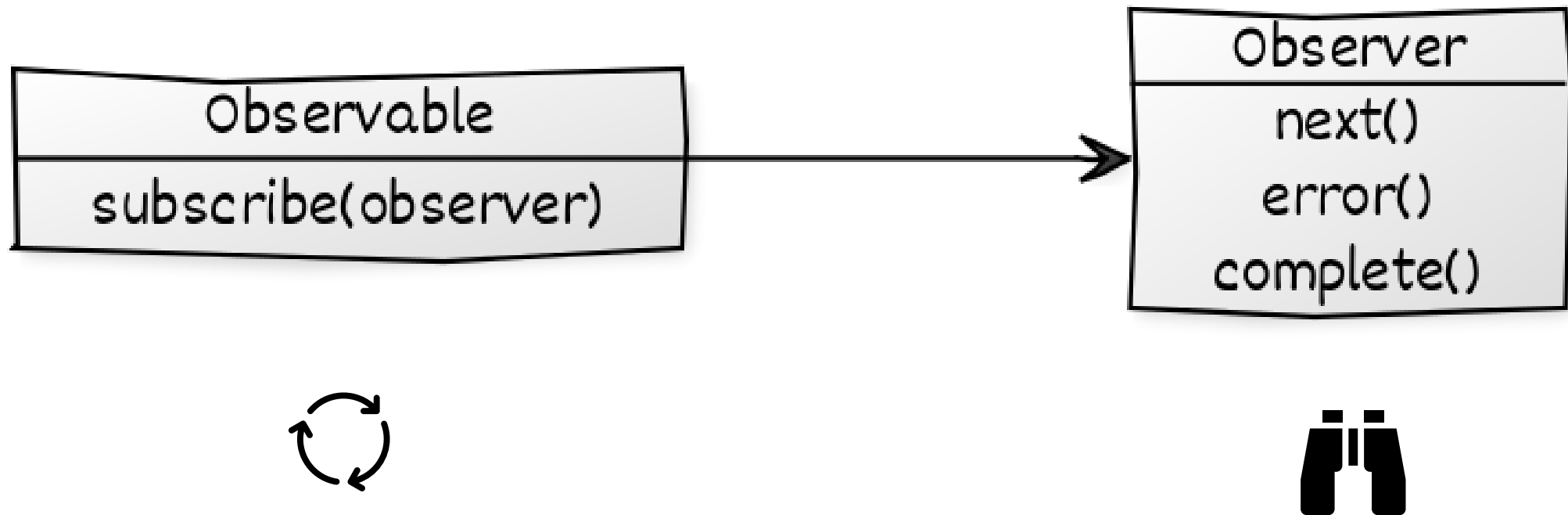
```
myObservable.subscribe(  
  next: (result) => { ... },  
  error: (error) => { ... },  
  complete: () => { ... }  
));
```


Example with Pipeable Operators

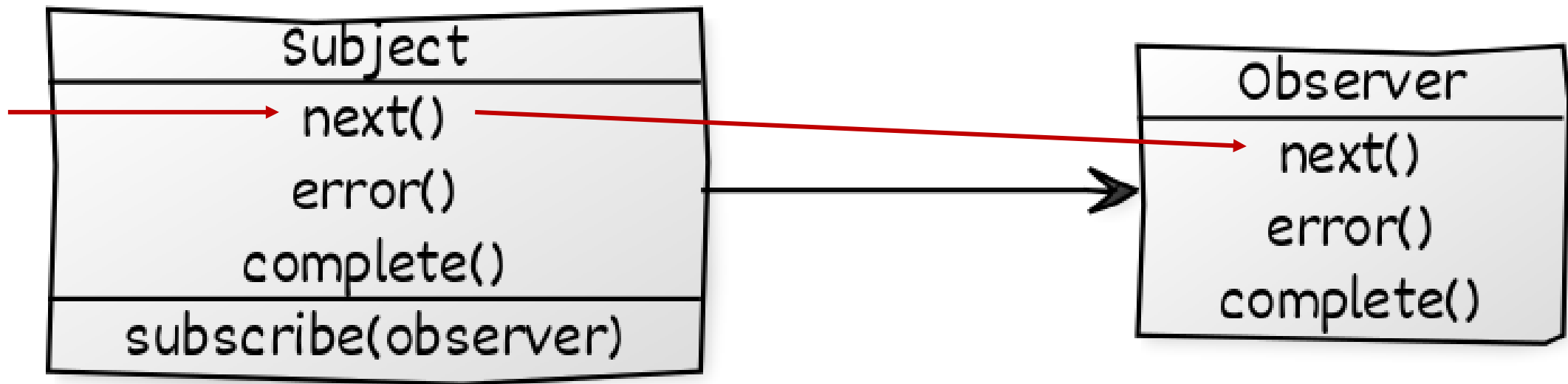
```
import { map } from 'rxjs/operators';

this
  .http
  .get("http://www.angular.at/api/...")
  .pipe(map(flightDateStr => new Date(flightDateStr)))
  .subscribe({
    next: (bookings) => { ... },
    error: (err) => { console.error(err); }
  });
```

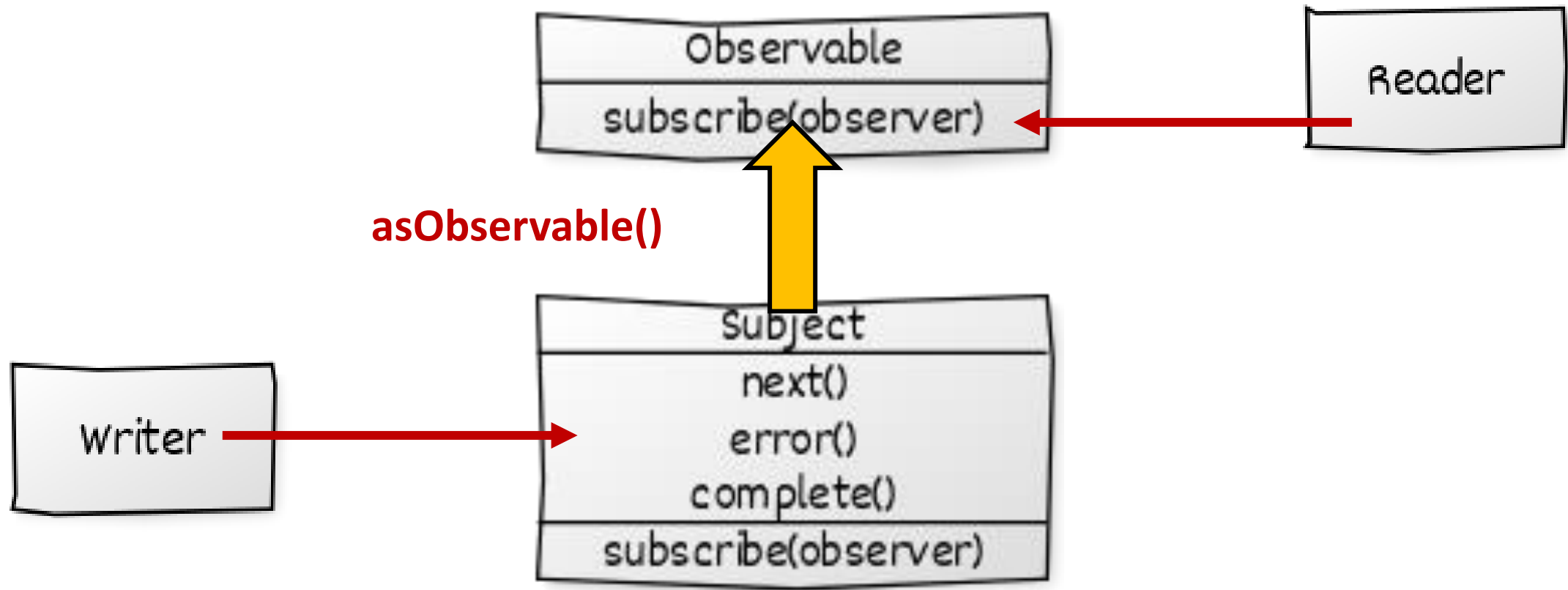
Observable und Observer



Subjects: Special Observables



Convert Subject into Observable



asObservable

```
private subject = new Subject<Flight>();  
readonly observable = subject.asObservable();
```

```
[...]  
this.observable.subscribe(...)
```

```
[...]  
this.subject.next(...)
```

Why Observables?


Asynchronous
operations

Interactive
(reactive)
behavior

Creating Observables

Creating an Observable

```
let observable = new Observable((observer) => {  
    observer.next(4711);  
    observer.next(815);  
  
    // observer.error("err!");  
  
    observer.complete();  
  
    return () => { console.debug('Bye bye'); };  
});
```



Sync & Async, Event-driven

```
let subscription = observable.subscribe(observer);
```

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


Creation Operators (Factories)

[<https://www.learnrxjs.io>]

fromEvent

of

throwError

interval

timer

Cold vs. Hot Observables

Cold vs. Hot Observables

Cold

- Default
- Point to point
- One Sender per consumer
- Lazy: Only starts at subscription

Hot

- Multicast
- Eager: Sender starts without subscriptions

Create Hot Observable

```
let o = this.find(from, to).pipe(share());
```

```
o.subscribe(...);
```



```
o.subscribe(...);
```

Sender starts with first subscription

Sender stops after all receiver have
been unsubscribed

Create Hot Observable

```
let o = this.find(from, to)
    .pipe(shareReplay({ bufferSize:1, refCount: true }));

o.subscribe(...);

[...];

o.subscribe(...);
```

DEMO

Operators

Transformation Operators

Operators

[<http://rxmarbles.com/#map>]



`map(x => 10 * x)`





`pluck("a")`

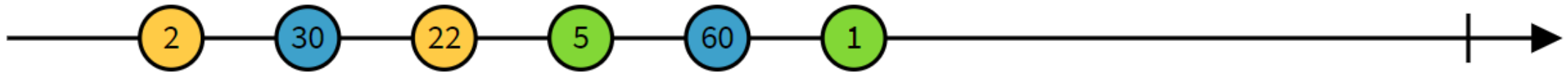




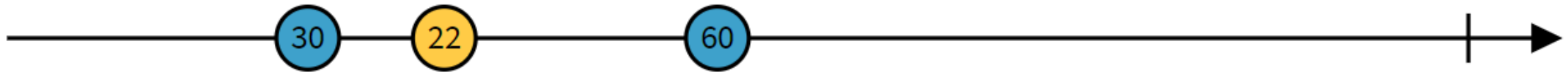
pairwise



Filtering Operators

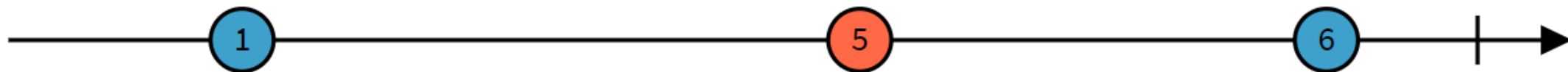


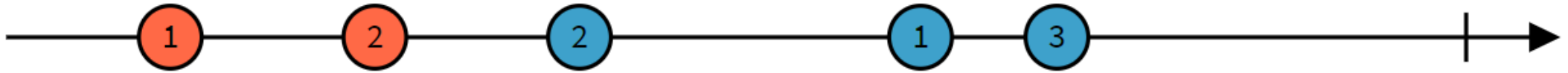
`filter(x => x > 10)`



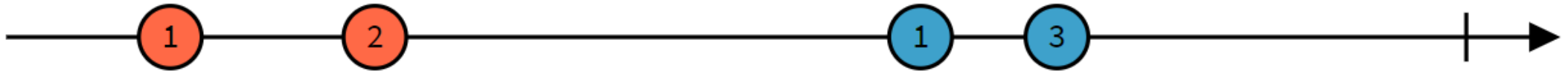


`debounceTime(10)`





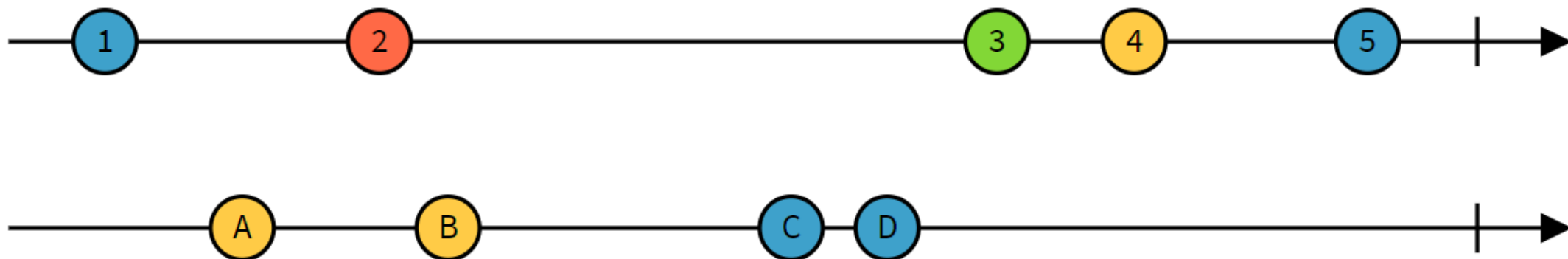
`distinctUntilChanged`



DEMO: Lookahead

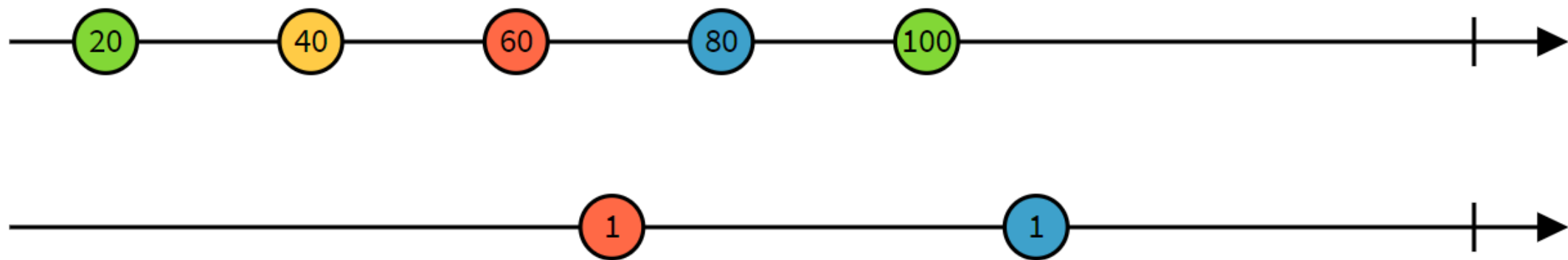
LAB

Combination Operators



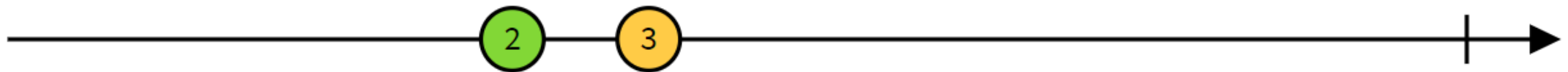
```
combineLatest((x, y) => "" + x + y)
```



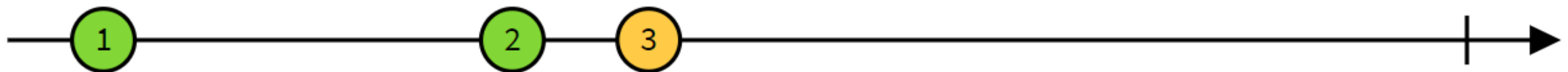


merge





`startWith(1)`



DEMO

Labs

Higher Order Observables

Operators for Higher Order Observables

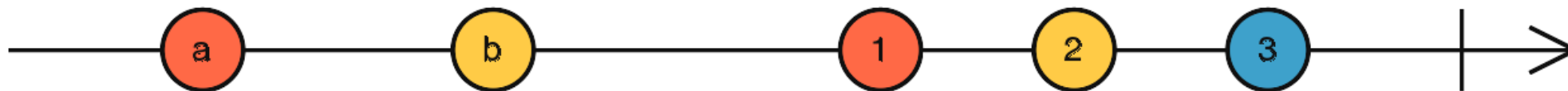
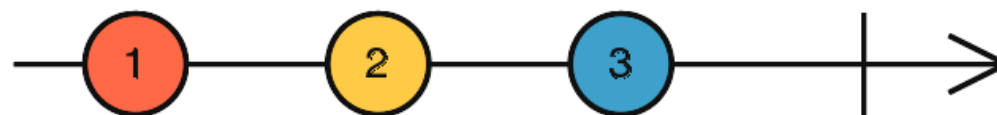
- switchMap
- mergeMap
- concatMap
- exhaustMap

DEMO

Error Handling

Operators for Error Handling

- catchError
- retry
- retryWhen
- throwError

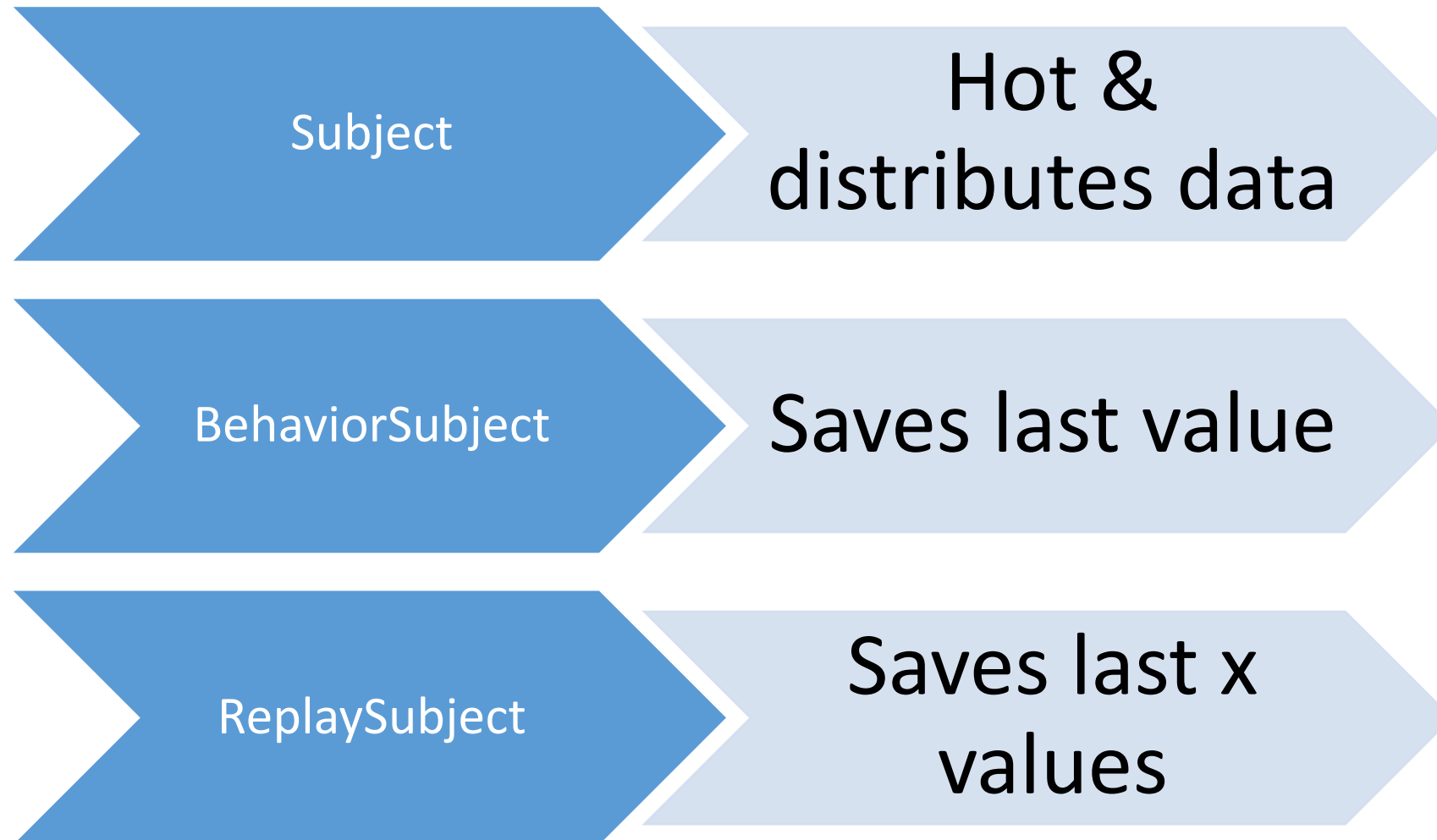


DEMO

LAB

Subjects

Subjects



DEMO: Pub/Sub with Subjects

Closing Observables

Closing Observables

- Explicitly
 - let subscription = observable\$.subscribe(...);
subscription.**unsubscribe()**;
- Implicitly
 - observable\$.pipe(**take(2)**).subscribe(...);
 - observable\$.pipe(**first()**).subscribe(...);
 - observable\$.pipe(**takeUntil(otherSubject)**).subscribe(...);
- Implicitly with async-Pipe in Angular
 - {{ observable\$ | **async** }}
- Automatic by Angular
 - Everything, Angular opens is also closed by it

DEMO