Enhance Web Development

Angular 2

Observables et RxJS

- Reactive avec des Observables
- Qu'est-ce que RxJS ?
- Les opérateurs les plus communs
- Pipes Async

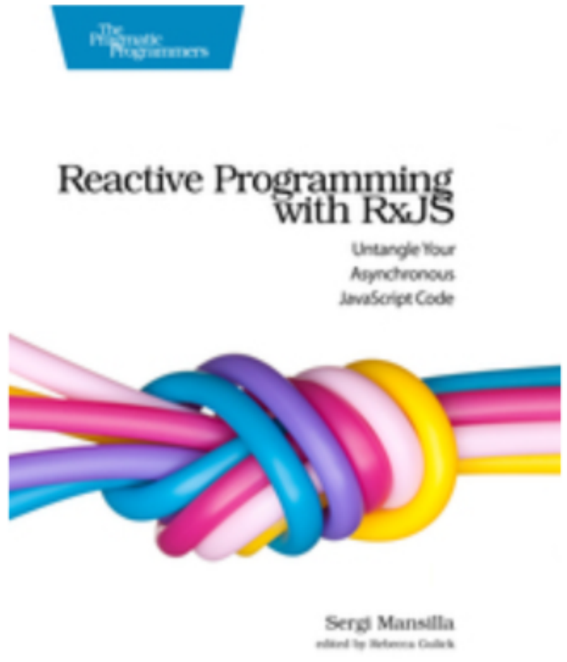
Reactive avec des Observables

- Dans le pattern Observer, un "*object*" (appelé le sujet), conserve une liste de ses "*dependants*" (appelé "*observers*") et les notifie automatiquement à chaque changement de "*state*"
- C'est ce qu'on appelle une "*push strategy*" (vs "*pull/polling strategy*")

Qu'est-ce que RxJS ?

- Une librairie pour composer des programmes asynchrone et basés sur les évènements en utilisant des collections d'*Observable*
- Un nombre **énorme** d'opérateur pour transformer le flux de donnée

Un peu de lecture :-)



https://pragprog.com/book/smreactjs/reactive-programming-with-rxjs

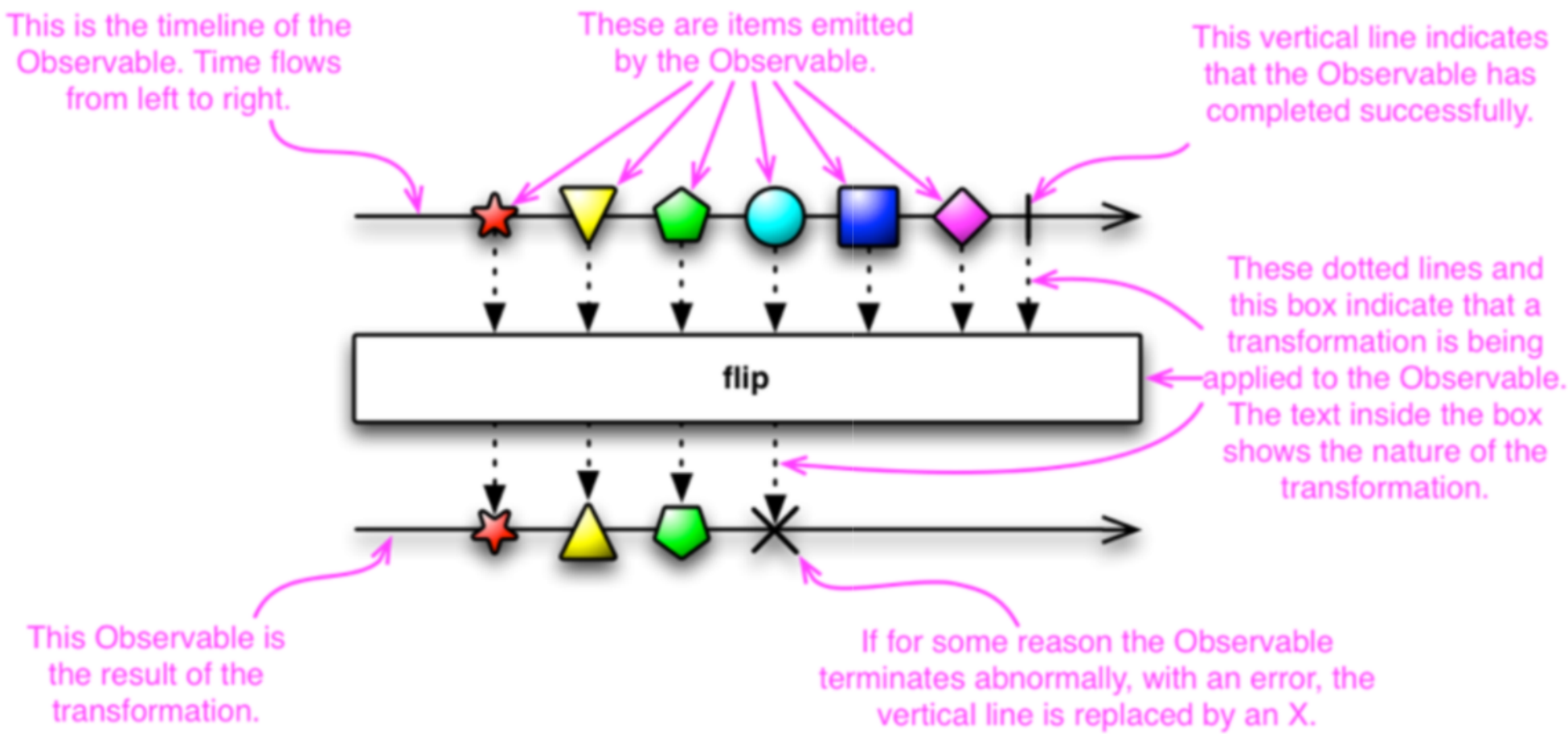
```
/* Get stock data somehow */
const source = getAsyncStockData()
const subscription = source
  .filter(quote => quote.price > 30)
  .map(quote => quote.price)
  .subscribe(
    price => console.log(`Prices higher than $30: ${price}`),
    err => console.log(`Something went wrong: ${err.message}`)
  )

/* When we're done */
subscription.dispose()
```

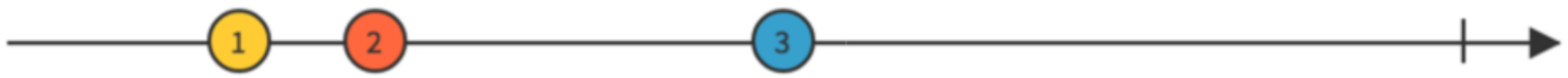
Exemple basique

Les opérateurs les plus communs

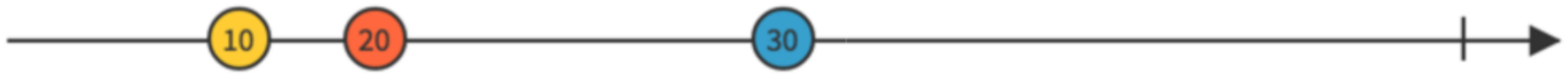
- map
- filter
- scan
- debounce
- distinctUntilChanged
- comineLatest
- flatMap


This is the timeline of the Observable. Time flows from left to right.
These are items emitted by the Observable.
This vertical line indicates that the Observable has completed successfully.
These dotted lines and this box indicate that a transformation is being applied to the Observable. The text inside the box shows the nature of the transformation.
flip
This Observable is the result of the transformation.
If for some reason the Observable terminates abnormally, with an error, the vertical line is replaced by an X.

Marbles




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



map

```
// Array
var numbers = [1, 2, 3]
var roots = numbers.map(Math.sqrt)

// roots is now [1, 4, 9], numbers is still [1, 2, 3]
// Observable
var source = Observable.range(1, 3)
  .map(x => x * x)

var subscription = source.subscribe(
  x => console.log('Next: ' + x),
  err => console.log('Error: ' + err),   () => console.log('Completed')
)

// => Next: 1
// => Next: 4
// => Completed
```

map


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

filter

```
// Array
var filtered = [12, 5, 8, 130, 44]
  .filter(x => x >= 10)
// filtered is [12, 130, 44]

// Observable
var source = Observable.range(0, 5)
  .filter(x => x % 2 === 0)

var subscription = source.subscribe(
  x => console.log('Next: ' + x),
  err => console.log('Error: ' + err),
  () => console.log('Completed')
)
// => Next: 0
// => Next: 2
// => Next: 4
// => Completed
```

filter


```
scan((x, y) => x + y)
```

scan

```
var source = Observable.range(1, 3)
  .scan((acc, x) => acc + x)

var subscription = source.subscribe(
  x => console.log('Next: ' + x),
  err => console.log('Error: ' + err),
  () => console.log('Completed')
)

// => Next: 1
// => Next: 3
// => Next: 6
// => Completed
```

scan

debounce

debounce

```
var array = [
  800,
  700,
  600,
  500
]
let source = Observable.for(array, function (x) { return Observable.timer(x)
  .map(function(x, i) { return i; })
  .debounce(function (x) { return Observable.timer(700); })

var subscription = source.subscribe(
  x => console.log('Next: ' + x),
  err => console.log('Error: ' + err),
  () => console.log('Completed')
)
// => Next: 0
// => Next: 3
// => Completed
```

debounce

distinctUntilChanged

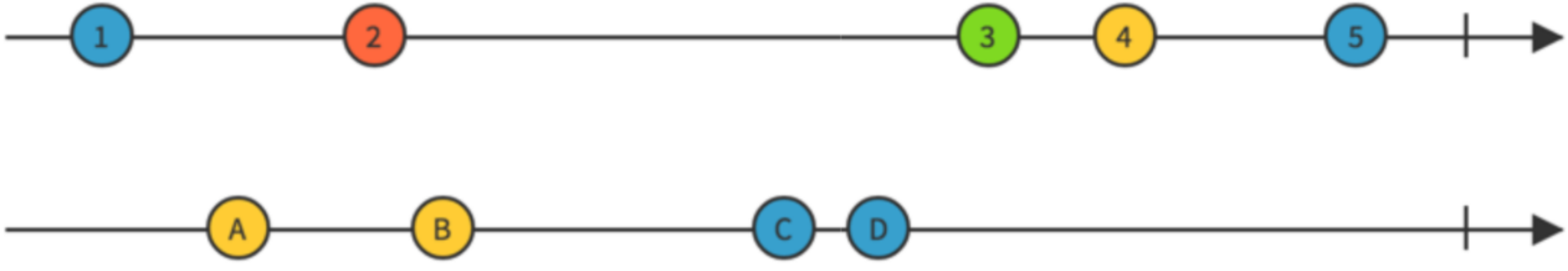
distinctUntilChanged

```
var source = Observable.of(42, 42, 24, 24)
  .distinctUntilChanged();

var subscription = source.subscribe(
  x => console.log('Next: ' + x),
  err => console.log('Error: ' + err),
  () => console.log('Completed')
)

// => Next: 42
// => Next: 24
// => Completed
```

distinctUntilChanged




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

distinctUntilChanged

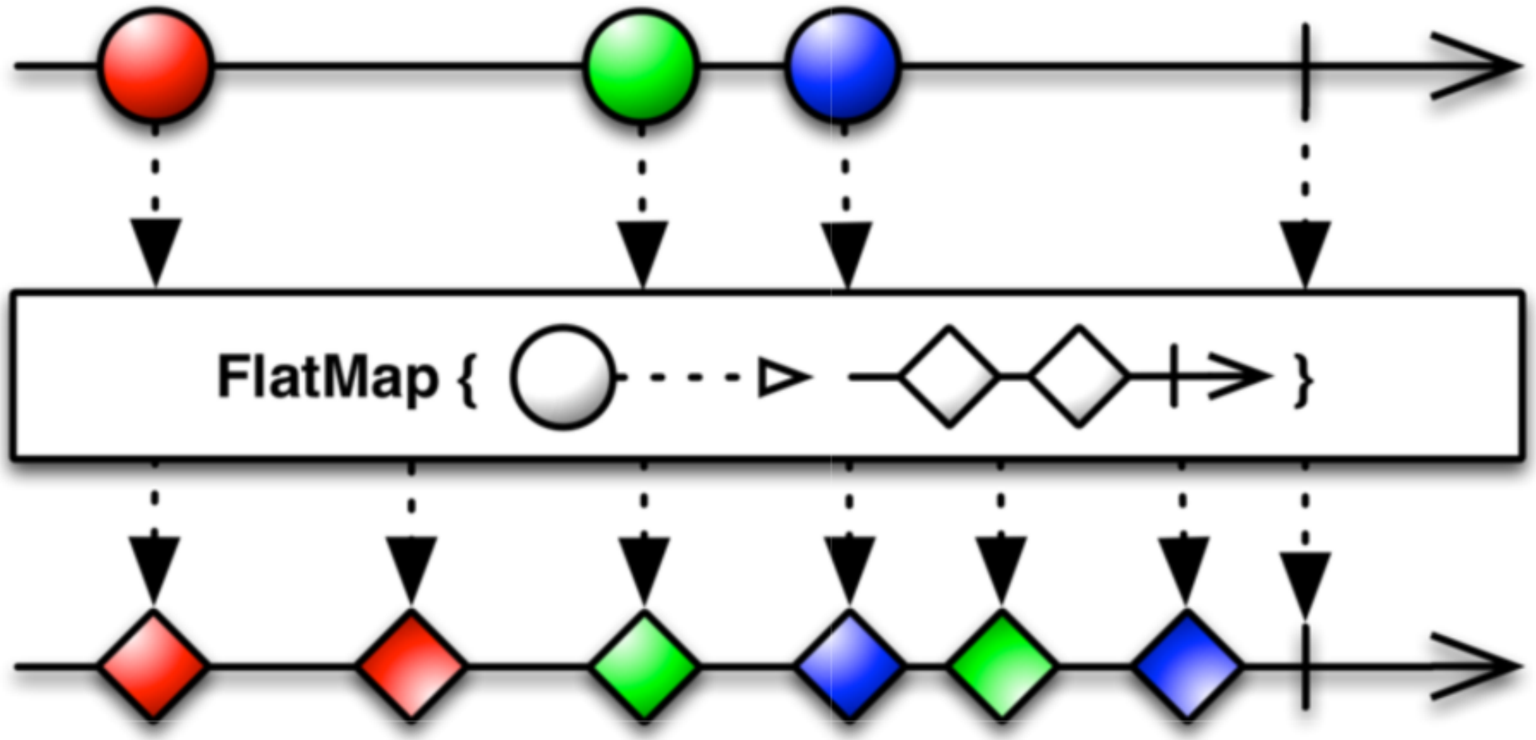
```
var source1 = Observable.interval(100)
  .map(function (i) { return 'First: ' + i; })

var source2 = Observable.interval(150)
  .map(function (i) { return 'Second: ' + i; })

// Combine latest of source1 and source2 whenever either gives a value
var source = Observable.combineLatest(
  source1,
  source2
).take(4)

var subscription = source.subscribe(
  x => console.log('Next: ' + JSON.stringify(x)),
  err => console.log('Error: ' + err),
  () => console.log('Completed')
)
// => Next: ["First: 0","Second: 0"]
// => Next: ["First: 1","Second: 0"]
// => Next: ["First: 1","Second: 1"]
// => Next: ["First: 2","Second: 1"]
// => Completed
```

distinctUntilChanged



flatMap

```
var source = Observable.range(1, 2)
  .flatMap(function (x) {
    return Observable.range(x, 2)
  })

var subscription = source.subscribe(
  x => console.log('Next: ' + x),
  err => console.log('Error: ' + err),
  () => console.log('Completed')
)

// => Next: 1
// => Next: 2
// => Next: 2
// => Next: 3
// => Completed
```

flatMap

Les pipes async

- Permet de résoudre directement des données asynchrones dans la vue (observable/promises)
- Evite le processus manuel de se subscribe à une méthode asynchrone dans le composant
- On peut donc chainer les opérateur sur l'observable dans le composant et laisser la vue subscribe

More about change detection

http://blog.thoughtram.io/angular/2016/02/22/angular-2-change-detection-explained.html

```
@Component({
  selector: 'my-app',
  template: `
  <div>
    <items-list [items]="items | async"
                (selected)="selectItem($event)"
                (deleted)="deleteItem($event)">
    </items-list>
  </div>
  `,
  directives: [ItemList],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class App {
  items: Observable<Array<Item>>;

  constructor(private itemsService: ItemsService) {
    this.items = itemsService.items;
  }
}
```

Les pipes async