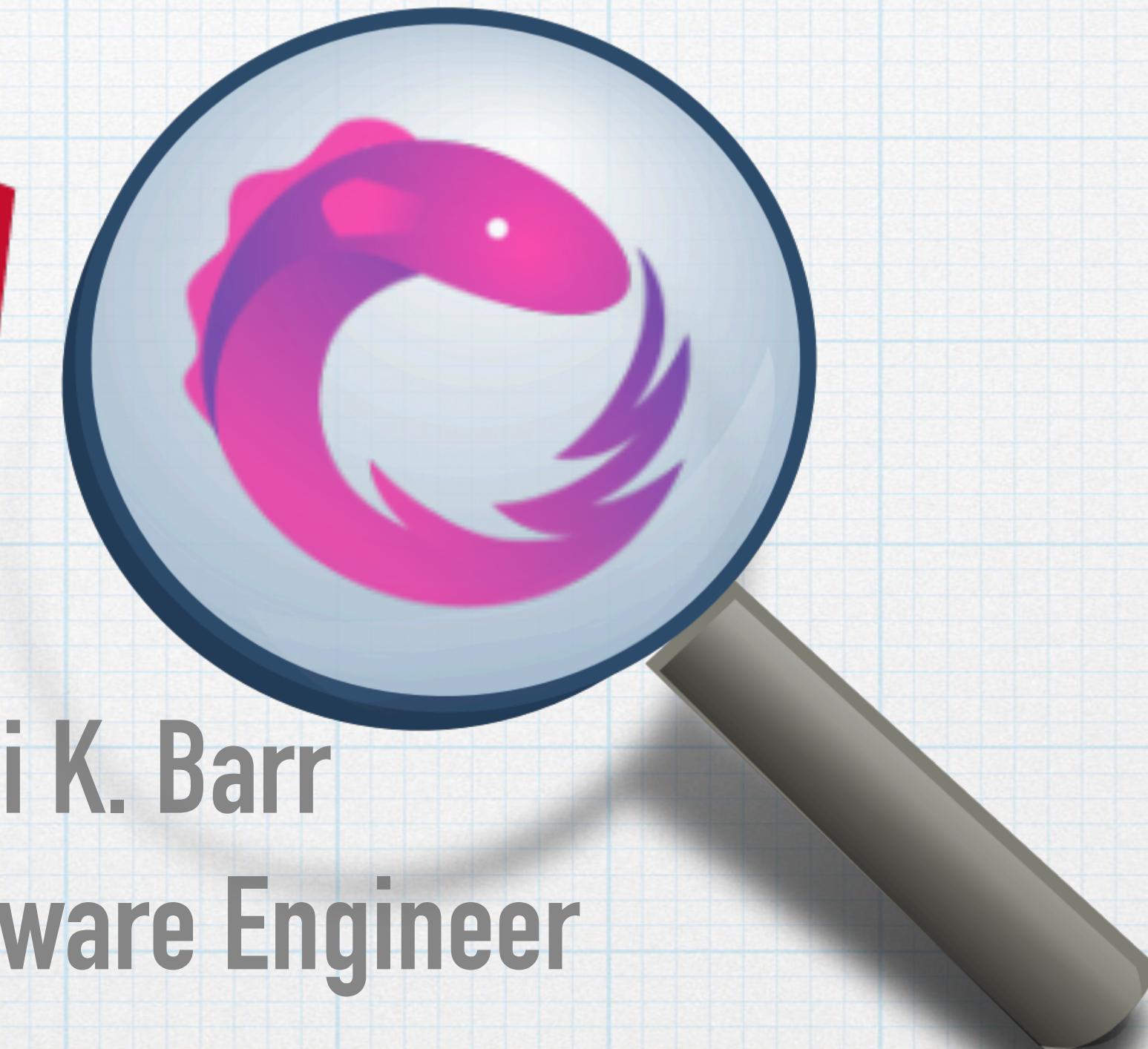
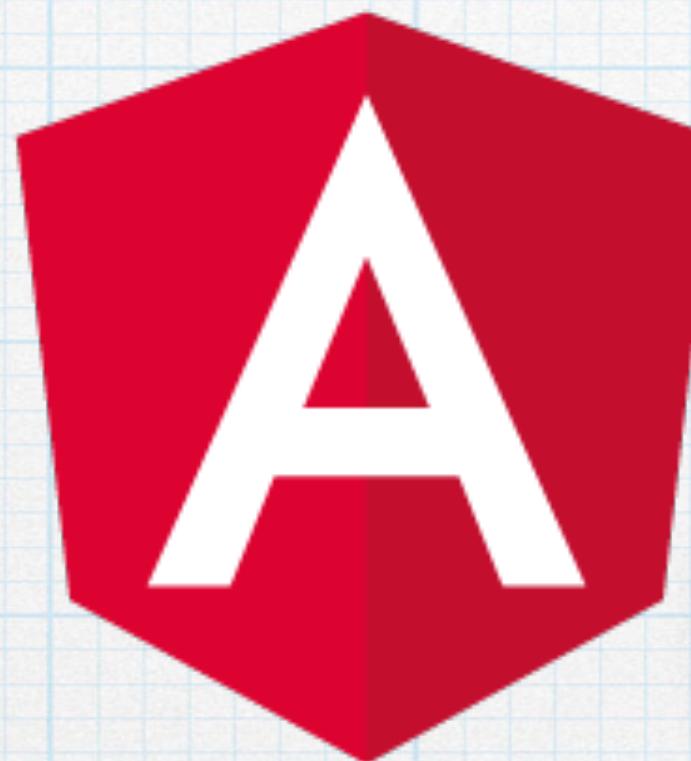


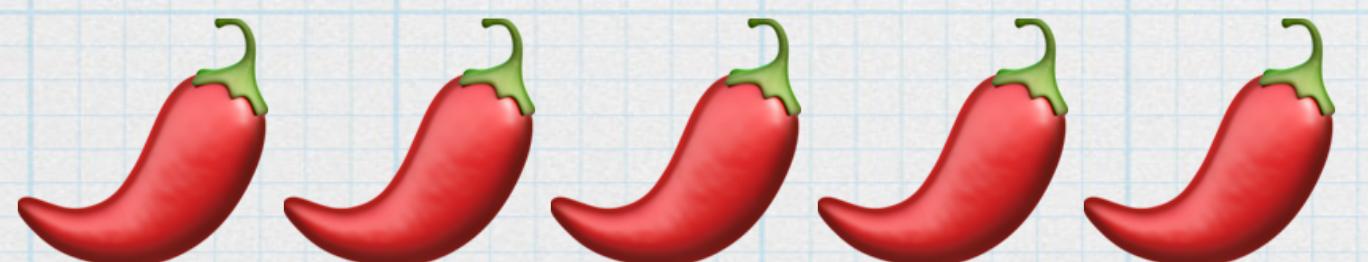
# Angular and The Case for RxJS



Sandi K. Barr  
Senior Software Engineer



# observable (not a spicy Promise)



- \* Multiple values over time
- \* Cancellable with unsubscribe
- \* Synchronous or asynchronous
- \* Declarative: what, not how or when
- \* Nothing happens without an observer
- \* Separate chaining and subscription
- \* Can be reused or retried





# observable

(lazy)

Create:

```
new Observable((observer) => observer.next(123));
```

Transform:

```
obs$.pipe(map((value) => value * 2));
```

Subscribe:

```
sub = obs$.subscribe((value) => console.log(value));
```

# Promise

(eager)

```
new Promise((resolve, reject) => resolve(123));
```

```
promise.then((value) => value * 2);
```

```
promise.then((value) => console.log(value));
```

*// implied by promise resolution*

# observable

```
const clicks$ = fromEvent(button, 'click');
const subscription = clicks$.subscribe(event => console.log('Clicked', event));
subscription.unsubscribe();
```

# Event

```
button.addEventListener('click', e => console.log('Clicked', e));
button.removeEventListener('click');
```

<https://angular.io/guide/comparing-observables#observables-compared-to-events-api>

# Observable



Subscription

# Observer

next()

error()

complete()

# Observable

## a function that takes an observer

```
const subscription = observable.subscribe(observer);  
subscription.unsubscribe();
```

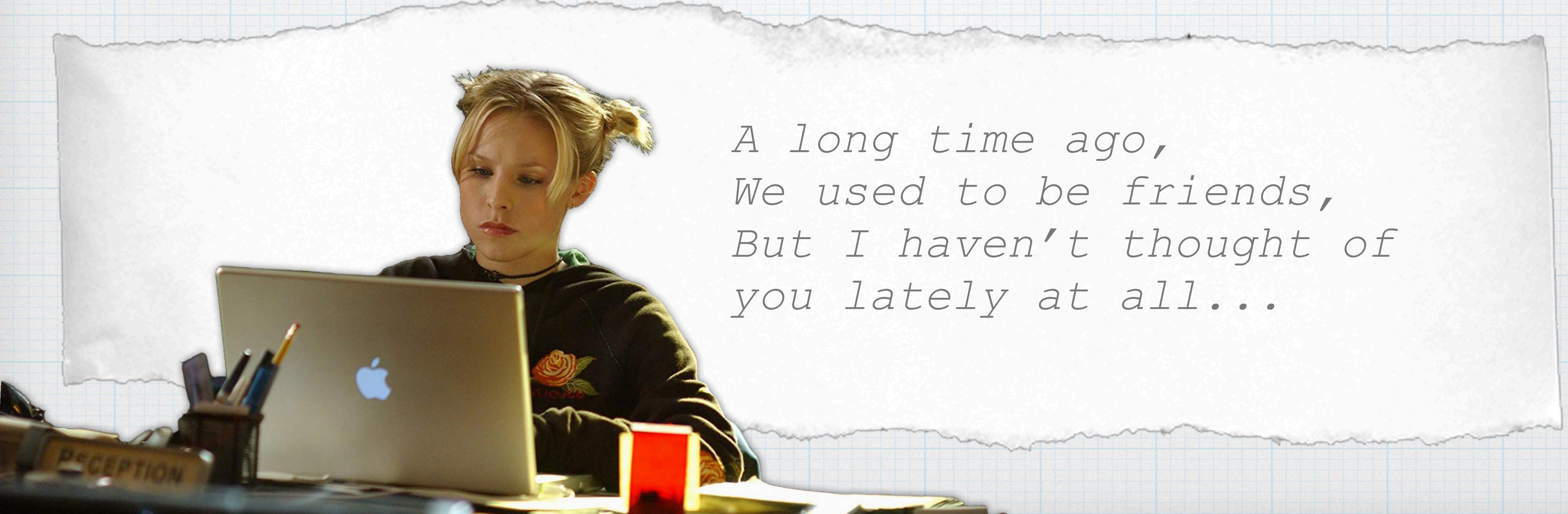
# Observer

## **an object with next, error, and complete methods**

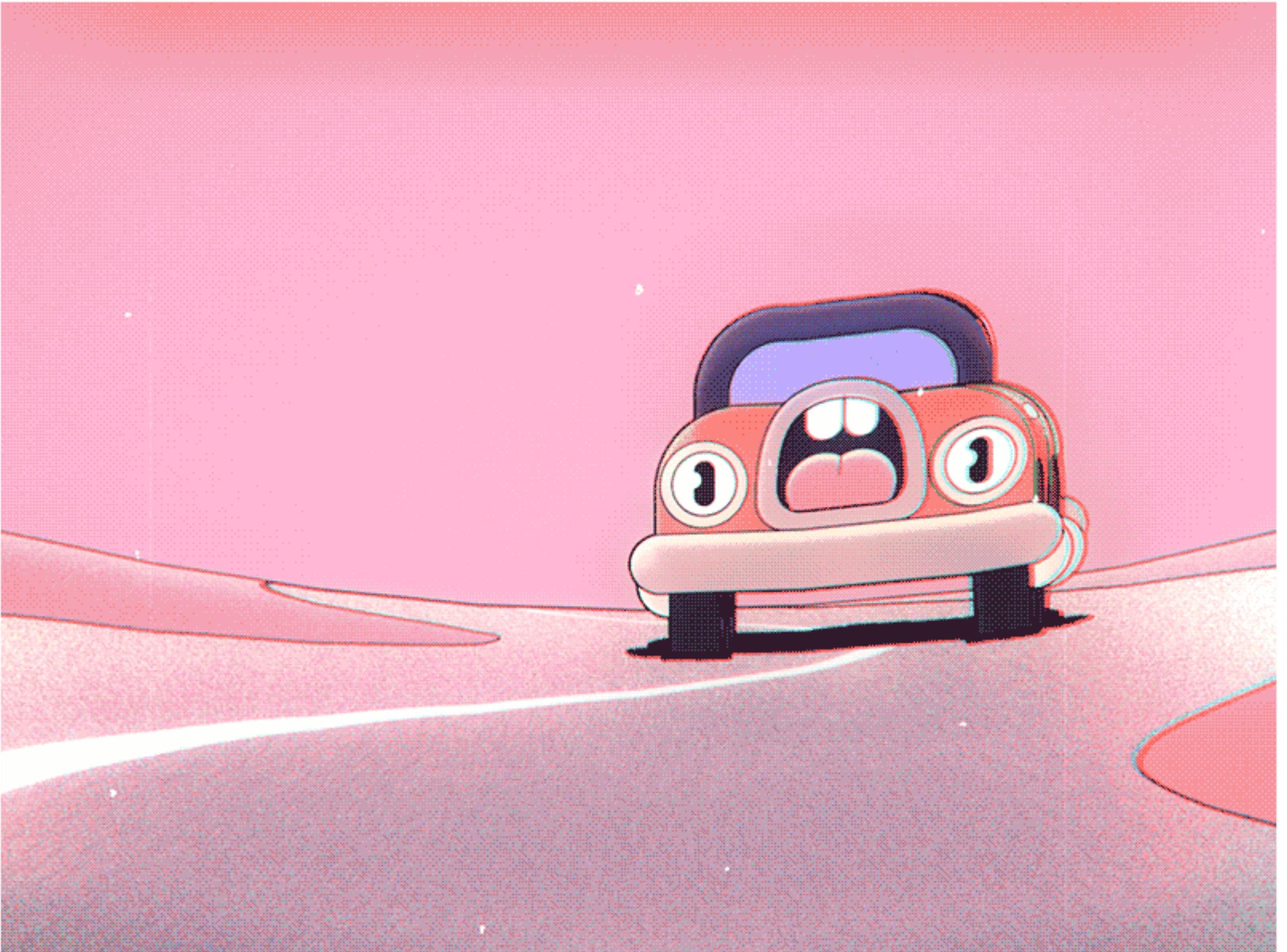
```
const observer = {  
  next: value => console.log('Next value:', value),  
  error: err => console.error('Error:', err),  
  complete: () => console.log('Complete')  
};  
  
const subscription = observable.subscribe(observer);
```

# Subscription

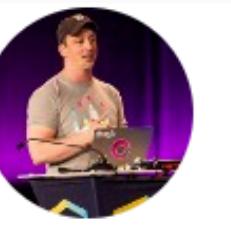
## **an observable only produces values on subscribe ()**



*A long time ago,  
We used to be friends,  
But I haven't thought of  
you lately at all...*



TERMINATION IS NOT GUARANTEED



Ben Lesh [Follow](#)

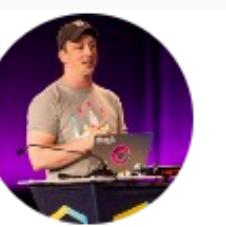
Mar 28, 2016 · 6 min read

## COLD is when your observable creates the producer

```
// COLD
var cold = new Observable(observer) => {
  var producer = new Producer();
  // have observer listen to producer here
};
```

## HOT is when your observable closes over the producer

```
// HOT
var producer = new Producer();
var hot = new Observable(observer) => {
  // have observer listen to producer here
};
```



Ben Lesh

Follow

Mar 28, 2016 · 6 min read

...

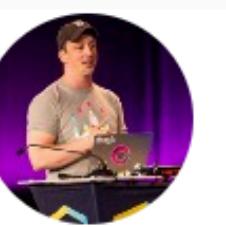
```
const source = new Observable(observer) => {
  const socket = new WebSocket('ws://someurl');
  socket.addEventListener('message', (e) => observer.next(e));
  return () => socket.close();
});
```

---

<https://medium.com/@benlesh/hot-vs-cold-observables-f8094ed53339>

# COLD OBSERVABLES

The producer is created during the subscription



Ben Lesh [Follow](#)

Mar 28, 2016 · 6 min read

...

```
const socket = new WebSocket('ws://someurl');

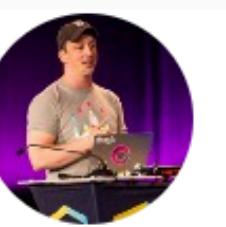
const source = new Observable((observer) => {
  socket.addEventListener('message', (e) => observer.next(e));
});
```

---

<https://medium.com/@benlesh/hot-vs-cold-observables-f8094ed53339>

# HOT OBSERVABLES

The producer is created outside the subscription



Ben Lesh

Follow

Mar 28, 2016 · 6 min read

...

```
function makeHot(cold) {  
  const subject = new Subject();  
  cold.subscribe(subject);  
  return new Observable((observer) => subject.subscribe(observer));  
}
```

<https://medium.com/@benlesh/hot-vs-cold-observables-f8094ed53339>

# SUBJECT: both an observable and an observer

Make a cold Observable hot

# be

[ bee; unstressed bee, bi ] [SHOW IPA](#)

[WORD ORIGIN](#)

[SEE MORE SYNONYMS FOR \*be\* ON THESAURUS.COM](#)

*verb (used without object), present singular 1st person am, 2nd are or (Archaic) art, 3rd is, present plural are; past singular 1st person was, 2nd were or (Archaic) wast or wert, 3rd was, past plural were; present subjunctive be; past subjunctive singular 1st person were, 2nd were or (Archaic) wert, 3rd were; past subjunctive plural were; past participle been; present participle be·ing.*

- 1 to exist or live:

*Shakespeare's "To be or not to be" is the ultimate question.*

- 2 to take place; happen; occur:

*The wedding was last week.*

- 3 to occupy a place or position:

*The book is on the table.*

- 4 to continue or remain as before:

*Let things be.*



## SUBJECT HAS STATE

Keeps a list of observers and sometimes also a number of the values that have been emitted

trying to learn any  
programming language 100%

come on

just a little  
bit more

almost there

oh crap...

RXJS

STEEP LEARNING CURVE

```
@Injectable()
export class TodoStoreService {

  private _todos: BehaviorSubject<Todo[]> = new BehaviorSubject([]);

  constructor(private todoHTTP: TodoHttpService) {
    this.loadData();
  }

  get todos(): Observable<Todo[]> {
    return this._todos.asObservable();
  }

  loadData() {
    this.todoAPI.getTodos()
      .subscribe(
        todos => this._todos.next(todos),
        err => console.log('Error retrieving Todos')
      );
  }
}
```

<https://github.com/sandikbarr/rxjs-todo/blob/master/src/app/todo/store/todo-store.service.ts>



## STORE: OBSERVABLE DATA SERVICE

Provide data to multiple parts of an application

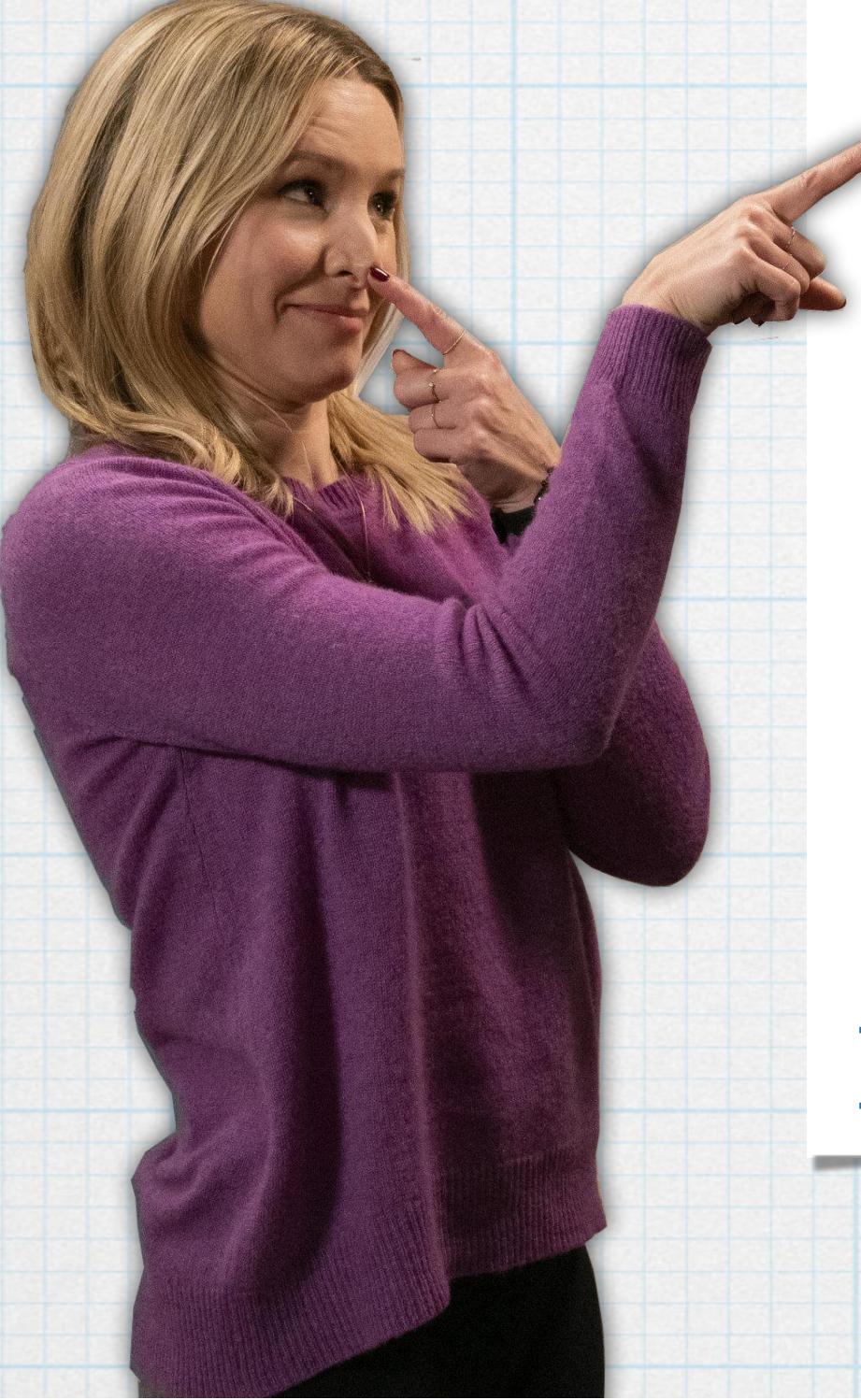


```
const subject = new BehaviorSubject(initialValue);  
  
// some time later...  
  
const value = subject.getValue();
```

<https://gist.github.com/sandikbarr/36bb0bf0b99a82a74be92aba1b1d0482#file-behavior-subject-getvalue-ts>

# BehaviorSubject

Just because you can access the value directly...



```
@Injectable()
export class TodoStoreService {

  private _todo: BehaviorSubject<Todo[]> = new BehaviorSubject([]);

  constructor(private todoAPI: TodoHttpService) {
    this.loadData();
  }

  get todos(): Observable<Todo[]> {
    return this._todos.asObservable();
  }
}
```

<https://github.com/sandikbarr/rxjs-todo/blob/master/src/app/todo/store/todo-store.service.ts>

# PROTECT THE BehaviorSubject WITH asObservable()

Components can subscribe after the data arrives

```
@Component({
  selector: 'app-todo-list',
  template: `
    <ul>
      <app-todo-list-item
        *ngFor="let todo of todos"
        [todo]="todo">
      </app-todo-list-item>
    </ul>
  `
})
export class TodoListComponent implements OnInit, OnDestroy {
  todos: Todo[];
  todosSub: Subscription;

  constructor (private todoStore: TodoStoreService) {}

  ngOnInit() {
    this.todosSub = this.todoStore.todos.subscribe(todos => {
      this.todos = todos;
    });
  }

  ngOnDestroy() {
    this.todosSub.unsubscribe();
  }
}
```

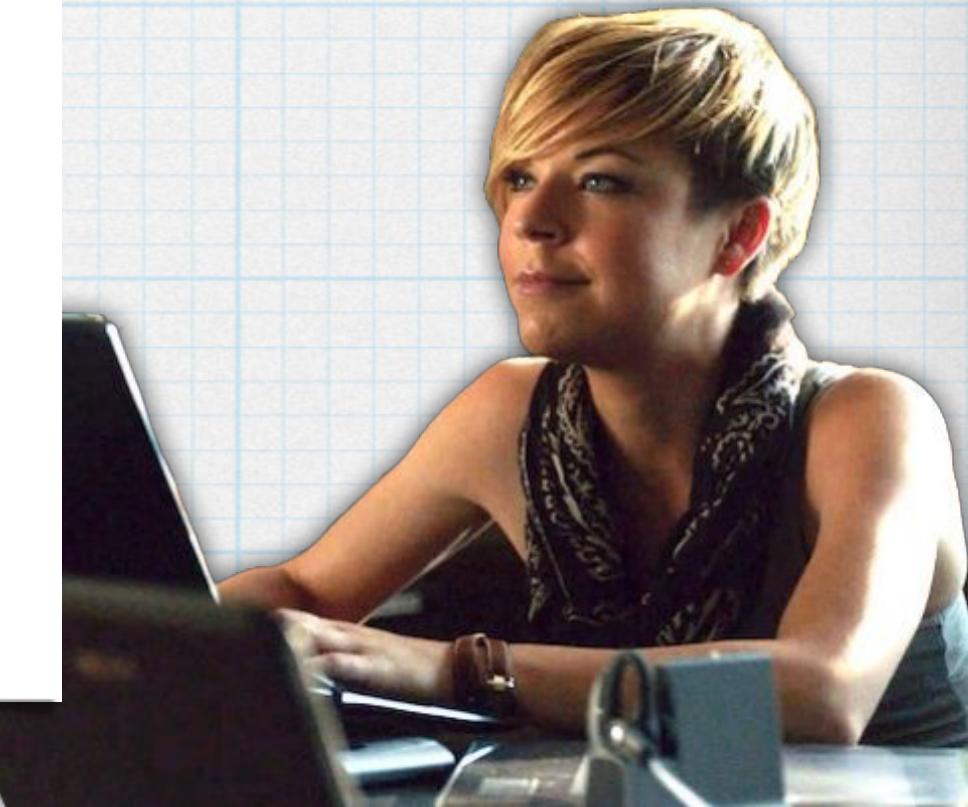
LET ME  
HAVE THAT  
TODO LIST

CLEAN UP  
SUBSCRIPTIONS



```
@Component({
  selector: 'app-todo-list',
  template: `
    <ul>
      <app-todo-list-item
        *ngFor="let todo of todos$ | async"
        [todo]="todo">
        </app-todo-list-item>
    </ul>
  `
})
export class TodoListComponent {
  todos$: Observable<Todo[]> = this.todoStore.todos;

  constructor (private todoStore: TodoStoreService) {}
}
```



<https://github.com/sandikbarr/rxjs-todo/blob/master/src/app/todo/components/todo-list.component.ts>

## USE THE ASYNC PIPE TO AVOID MEMORY LEAKS

Subscriptions live until a stream is completed or until they are manually unsubscribed

retweeted by Dan Abramov



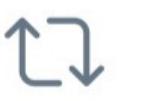
Justin Falcone  
@modernserf

separation of concerns



4:06 PM · 6/6/19 · Twitter for iPad

17 Retweets 124 Likes



Justin Falcone @modernserf · 6d  
Replying to @modernserf

though I guess the react version of this is  
a little bit of meatloaf + green bean +  
brownie in each cell of a pillminder



- \* Decouple responsibilities
- \* Observable Data Service
  - \* Provide data to multiple parts of an application
  - \* Not the same as centralized state management

```
addTodo(newTodo: Todo): Observable<Todo> {
  const observable = this.todoAPI.saveTodo(newTodo);

  observable.pipe(
    withLatestFrom(this.todos),
  ).subscribe(([savedTodo, todos]) => {
    this._todos.next(todos.concat(savedTodo));
  });

  return observable;
}
```

<https://github.com/sandikbarr/rxjs-todo/blob/master/src/app/todo/store/todo-store.service.ts>

## STORE: OBSERVABLE DATA SERVICE

Action method updates the store on success

```
addTodo(newTodo: Todo): Observable<Todo> {
  const observable = this.todoAPI.saveTodo(newTodo);

  observable.pipe(
    withLatestFrom(this.todos),
  ).subscribe(([savedTodo, todos]) => {
    this._todos.next(todos.concat(savedTodo));
  });

  return observable;
}
```

<https://github.com/sandikbarr/rxjs-todo/blob/master/src/app/todo/store/todo-store.service.ts>

## HTTP OBSERVABLES ARE COLD

Avoid duplicating HTTP requests!

```
@Injectable()
export class TodoHttpService {
  base = 'http://localhost:3000/todos';

  constructor(private http: HttpClient) { }

  getTodos(): Observable<Todo[]> {
    return this.http.get<Todo[]>(this.base);
  }

  saveTodo(newTodo: Todo): Observable<Todo> {
    return this.http.post<Todo>(this.base, newTodo, {headers}).pipe(share());
  }
}
```

---

<https://github.com/sandikbarr/rxjs-todo/blob/master/src/app/todo/http/todo-http.service.ts>

# HTTP SERVICE

**share () makes a cold Observable hot**

# OPERATORS

**compose complex  
asynchronous code in a  
declarative manner**



Pipeable Operators: take an input Observable and generate a resulting output Observable

Examples: `filter`, `map`, `mergeMap`

Creation Operators: standalone functions to create a new Observable

Examples: `interval`, `of`, `fromEvent`, `concat`

```
@Injectable()
export class TodoHttpService {
  base = 'http://localhost:3000/todos';

  constructor(private http: HttpClient) { }

  getTodos(): Observable<Todo[]> {
    return this.http.get<Todo[]>(this.base);
  }

  saveTodo(newTodo: Todo): Observable<Todo> {
    return this.http.post<Todo>(this.base, newTodo, {headers}).pipe(share());
  }
}
```

---

<https://github.com/sandikbarr/rxjs-todo/blob/master/src/app/todo/http/todo-http.service.ts>

**share () : refCount**  
share () also makes Observables retry-able



HOT: SHARE THE EXECUTION



COLD: INVOKE THE EXECUTION

```
@Component({
  selector: 'app-todo',
  template: `
    Total #: {{ totals$ | async }}
    <app-todo-list [todos]="todos$ | async"></app-todo-list>
  `
})
export class DuplicateHttpTodoComponent {
  todos$ = this.http.get<Todo[]>('http://localhost:3000/todos');
  total$ = this.todos$.pipe(map(todos => todos.length));

  constructor(private http: HttpClient) {}
}
```

---

<https://gist.github.com/sandikbarr/36bb0bf0b99a82a74be92aba1b1d0482#file-duplicate-http-component-ts>

## async : Unsubscribes automatically

But still be sure to avoid duplicating HTTP requests!

```
@Component({
  selector: 'app-todo',
  template: `
    <ng-container *ngIf="todos$ | async as todos">
      Total #: {{ todos.length }}
      <app-todo-list [todos]="todos"></app-todo-list>
    </ng-container>
  `
})
export class TodoComponent {
  todos$ = this.http.get<Todo[]>('http://localhost:3000/todos');

  constructor(private http: HttpClient) {}
}
```

---

<https://gist.github.com/sandikbarr/36bb0bf0b99a82a74be92aba1b1d0482#file-async-as-component-ts>

# ngIf with | async as

Assign Observable values to a local variable

```

@Component({
  selector: 'app-todo',
  template: `
    <ng-container *ngIf="todos$ | async as todos; else loading">
      Total #: {{ todos.length }}
      <app-todo-list [todos]="todos"></app-todo-list>
    </ng-container>
    <ng-template #loading><p>Loading...</p></ng-template>
  `
})
export class TodoComponent {
  todos$ = this.http.get<Todo[]>('http://localhost:3000/todos');

  constructor(private http: HttpClient) {}
}

```

---

<https://gist.github.com/sandikbarr/36bb0bf0b99a82a74be92aba1b1d0482#file-async-as-else-component-ts>

# ngIf with | async as with ; ngElse

Show alternate block when Observable has no value



🤘 REACTIVE TEMPLATES 🤘

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

const source$ = range(0, 10);

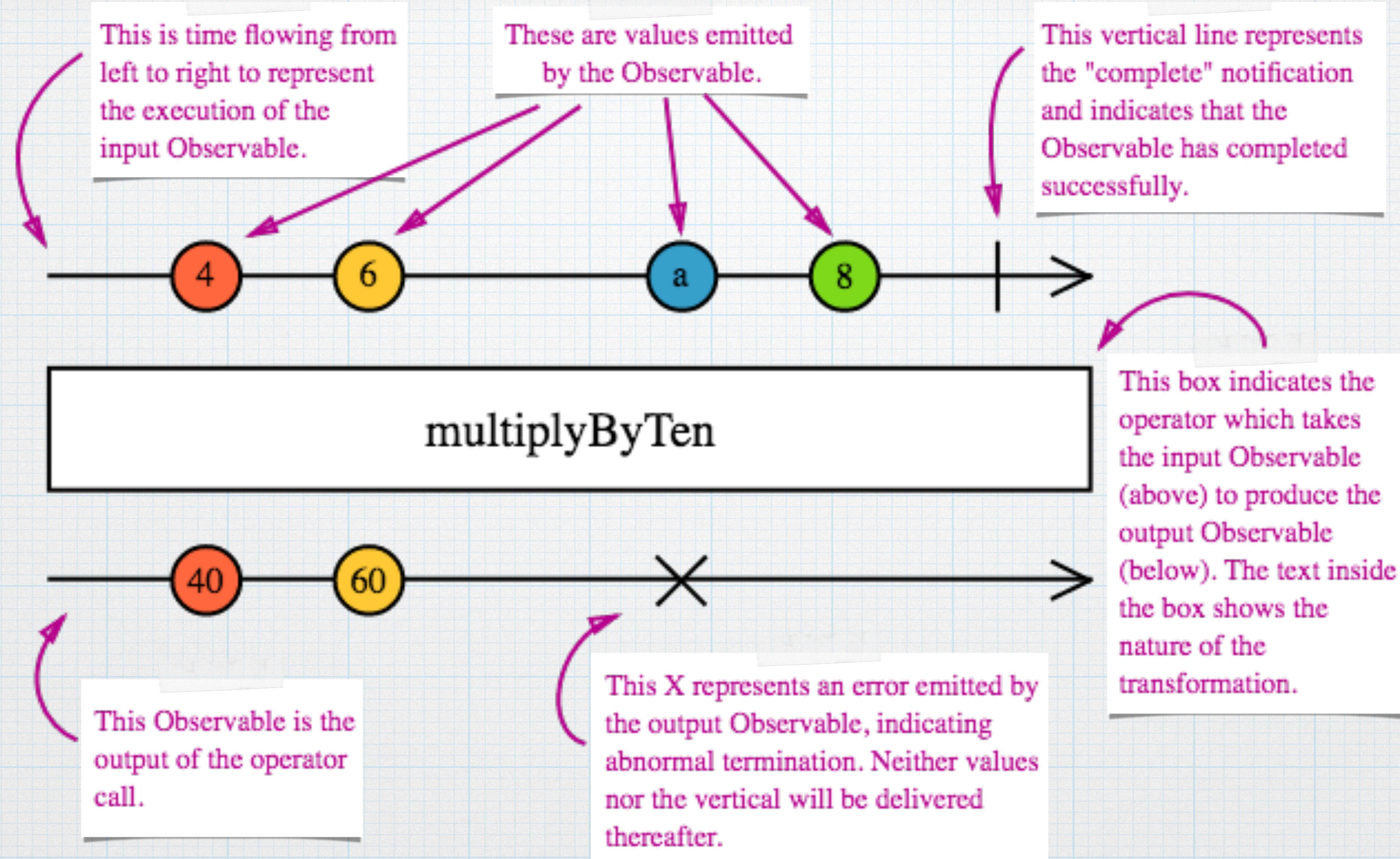
source$.pipe(
  filter(x => x % 2 === 0),
  map(x => x + x),
  scan((acc, x) => acc + x, 0)
).subscribe(x => console.log(x));
```

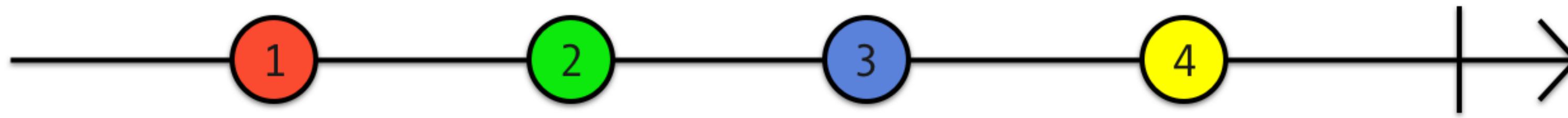
---

<https://gist.github.com/sandikbarr/36bb0bf0b99a82a74be92aba1b1d0482#file-pipe-ts>

# Observable.prototype.pipe()

## Compose a series of operators





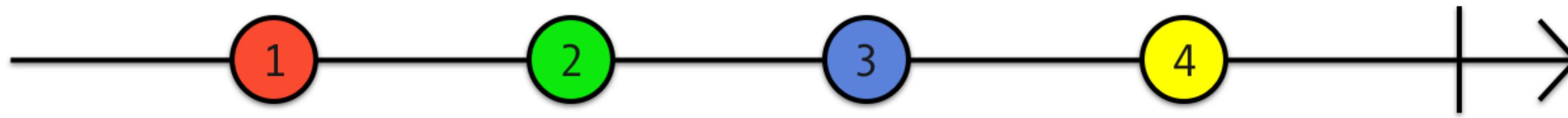
```
map(x => x * 10)
```



## map()

### A TRANSFORMATIONAL OPERATOR

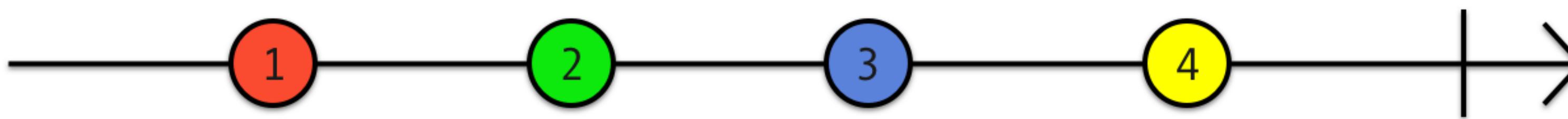
Applies a projection to each value



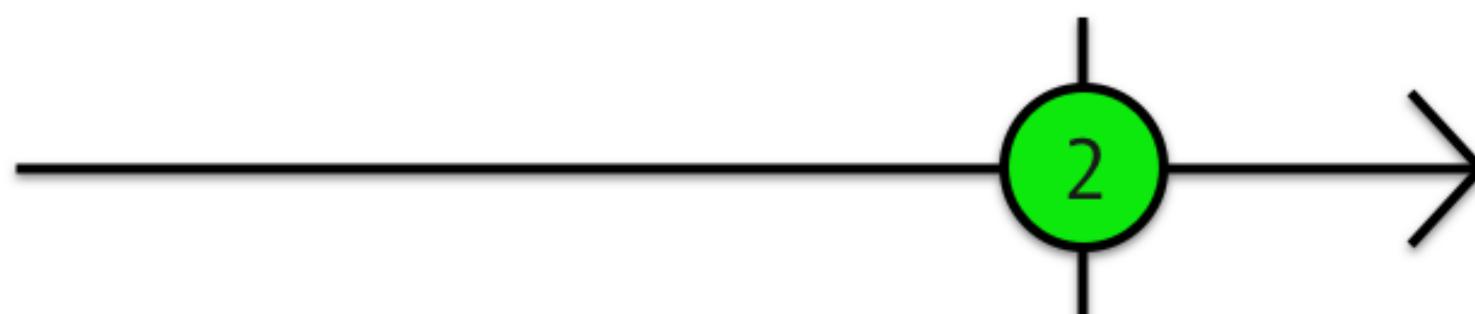
```
filter(x => x % 2 === 0)
```



**filter()**  
ONE OF MANY FILTERING OPERATORS  
Filters items emitted from source



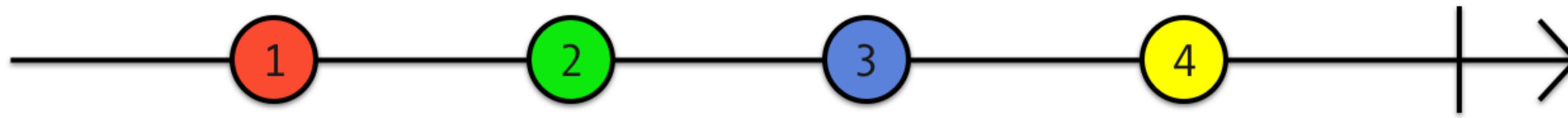
```
find(x => x % 2 === 0)
```



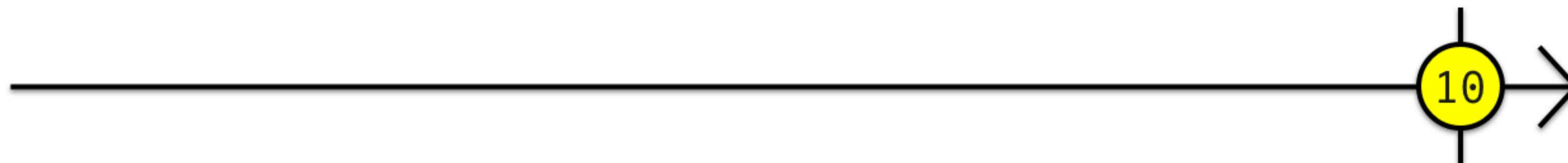
**find()**

**A CONDITIONAL OPERATOR**

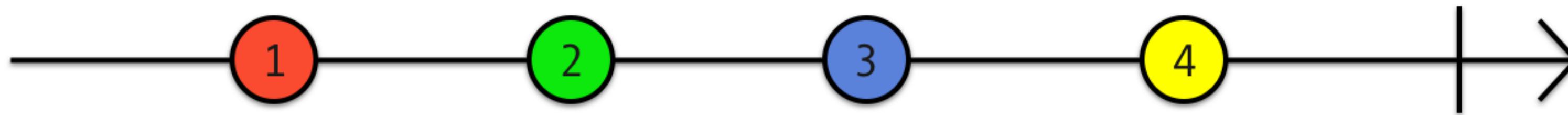
**Finds the first match then completes**



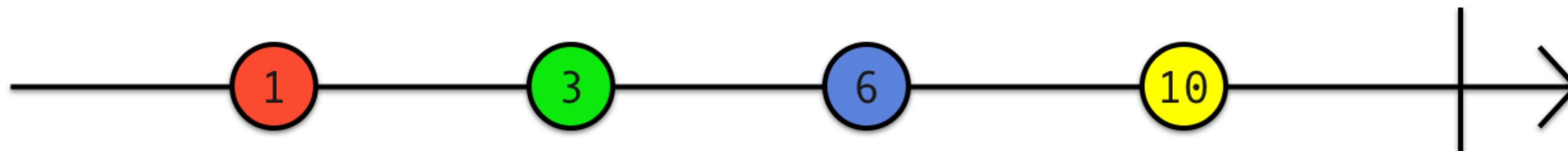
```
reduce((x, y) => x + y)
```



**reduce()**  
AN AGGREGATE OPERATOR  
Emits aggregate result on completion



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



## scan()

A TRANSFORMATIONAL OPERATOR

Emits accumulated result at each interval



```
@Component({
  selector: 'app-todo-search',
  template: `
    <label for="search">Search: </label>
    <input id="search" (keyup)="onSearch($event.target.value)"/>
  `
})
export class TodoSearchComponent implements OnInit, OnDestroy {
  @Output() search = new EventEmitter<string>();

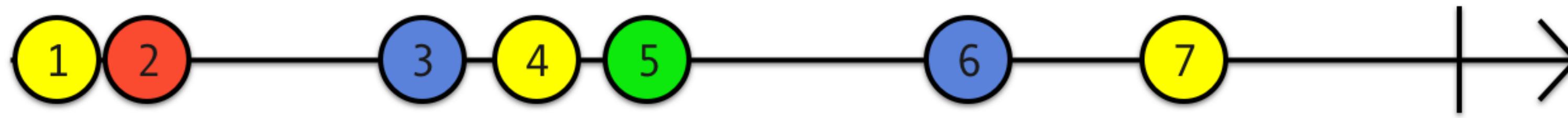
  changeSub: Subscription;
  searchStream = new Subject<string>();

  ngOnInit() {
    this.changeSub = this.searchStream.pipe(
      filter(searchText => searchText.length > 2), // min length
      debounceTime(300), // wait for break in keystrokes
      distinctUntilChanged() // only if value changes
    ).subscribe(searchText => this.search.emit(searchText));
  }

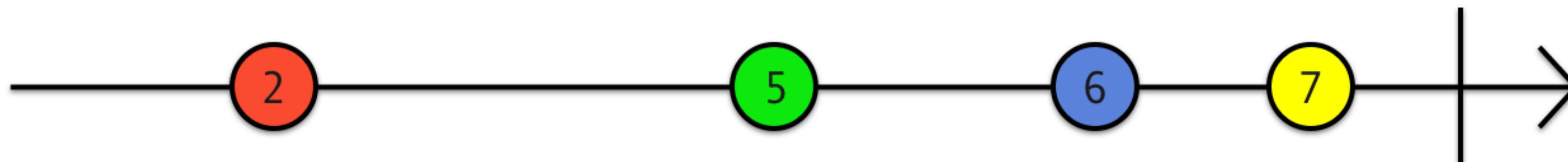
  ngOnDestroy() {
    if (this.changeSub) { this.changeSub.unsubscribe(); }
  }

  onSearch(searchText: string) {
    this.searchStream.next(searchText);
  }
}
```

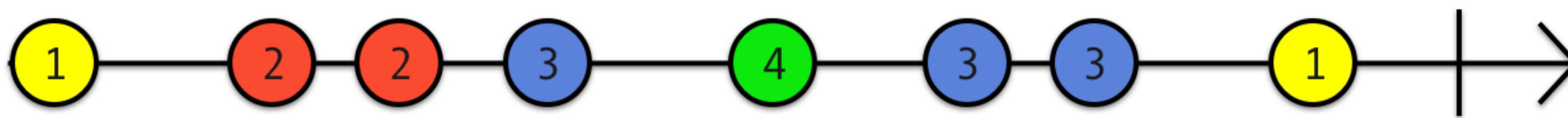
<https://github.com/sandikbarr/rxjs-todo/blob/master/src/app/todo/components/todo-search.component.ts>



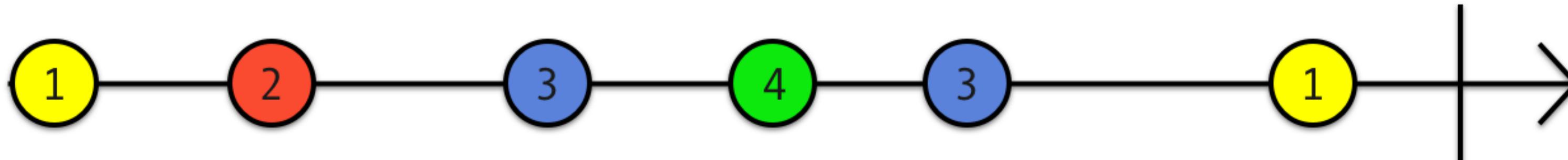
debounceTime(10)



**debounceTime ()**  
A FILTERING OPERATOR  
Rate-limit the input and delay the output



distinctUntilChanged()



distinctUntilChanged()

A FILTERING OPERATOR

Emits values that are distinct from the previous value

```
export class TodoEditComponent implements OnInit {  
  todo: Todo;  
  
  constructor(private todoStore: TodoStoreService,  
              private route: ActivatedRoute,  
              private router: Router) {}  
  
  ngOnInit() {  
    this.route.params.subscribe(params => {  
      const id = +params['id'];  
      this.todoStore.todos.subscribe((todos: Todo[]) => {  
        this.todo = todos.find(todo => todo.id === id);  
      });  
    });  
  }  
}
```

## AVOID NESTED SUBSCRIPTIONS



pyramid shaped callback hell



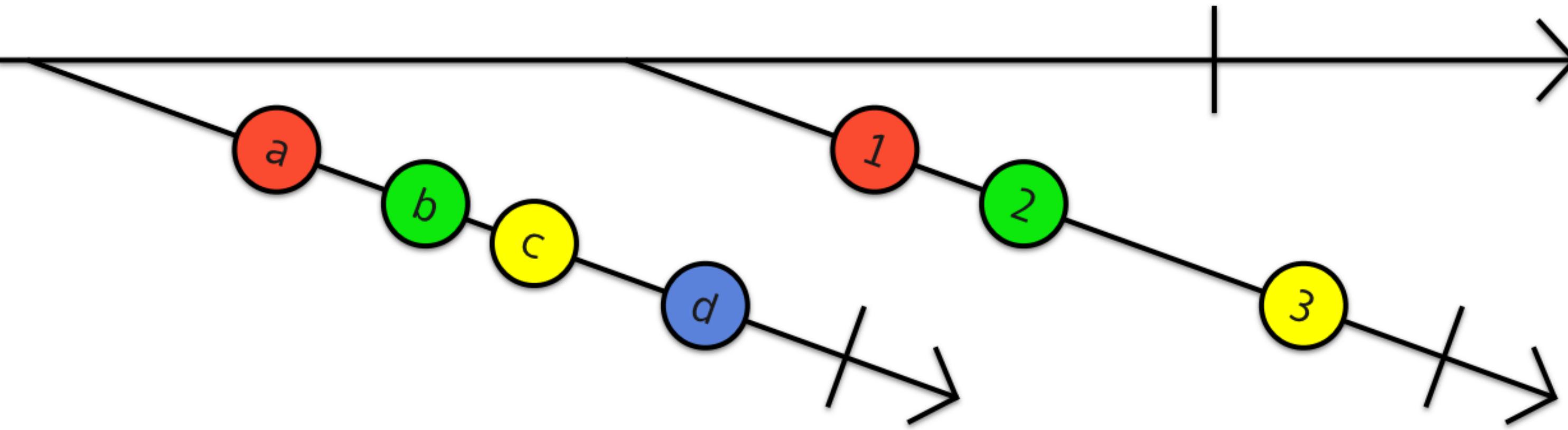
# HIGHER-ORDER OBSERVABLES

Observables that emit other Observables

```
export class TodoEditComponent {  
  todo$: Observable<Todo> = this.route.params.pipe(  
    map(params => +params['id']),  
    switchMap(id => this.todoStore.getTodoById(id))  
  );  
  
  constructor(private todoStore: TodoStoreService,  
             private route: ActivatedRoute,  
             private router: Router) {}  
}
```

<https://github.com/sandikbarr/rxjs-todo/blob/master/src/app/todo/components/edit/todo-edit.component.ts>

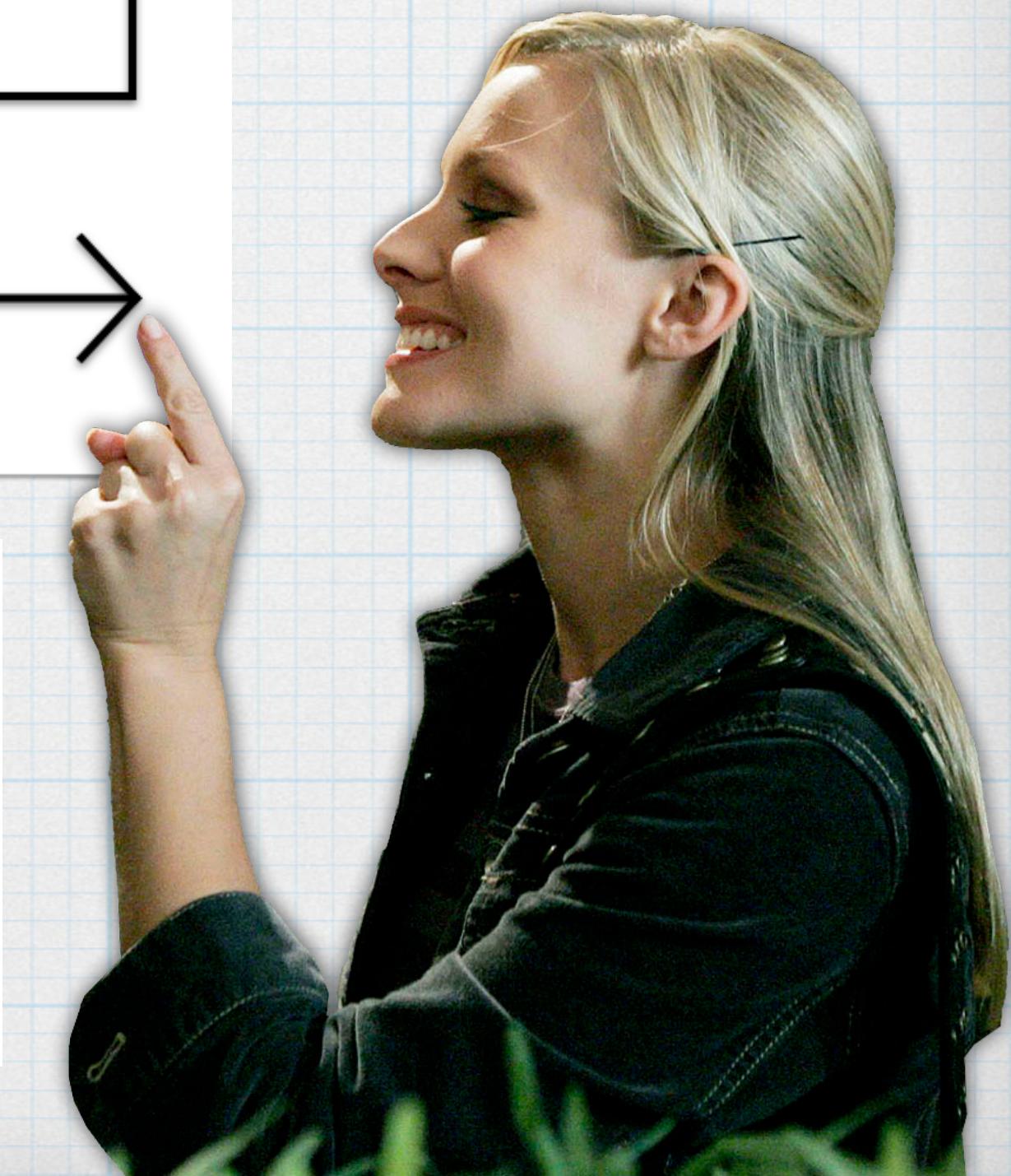


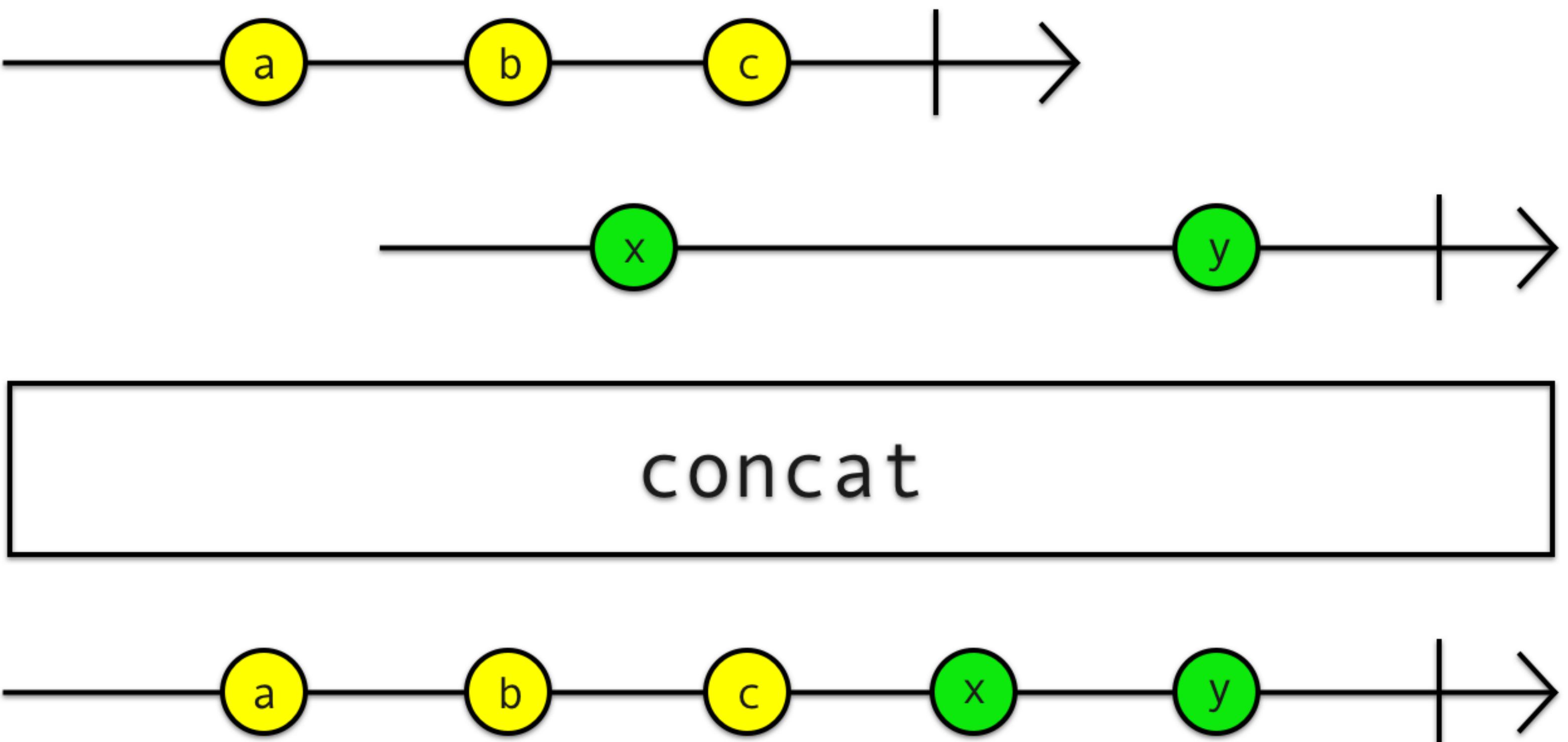


switch



**SWITCH**  
Cancels inner Observables

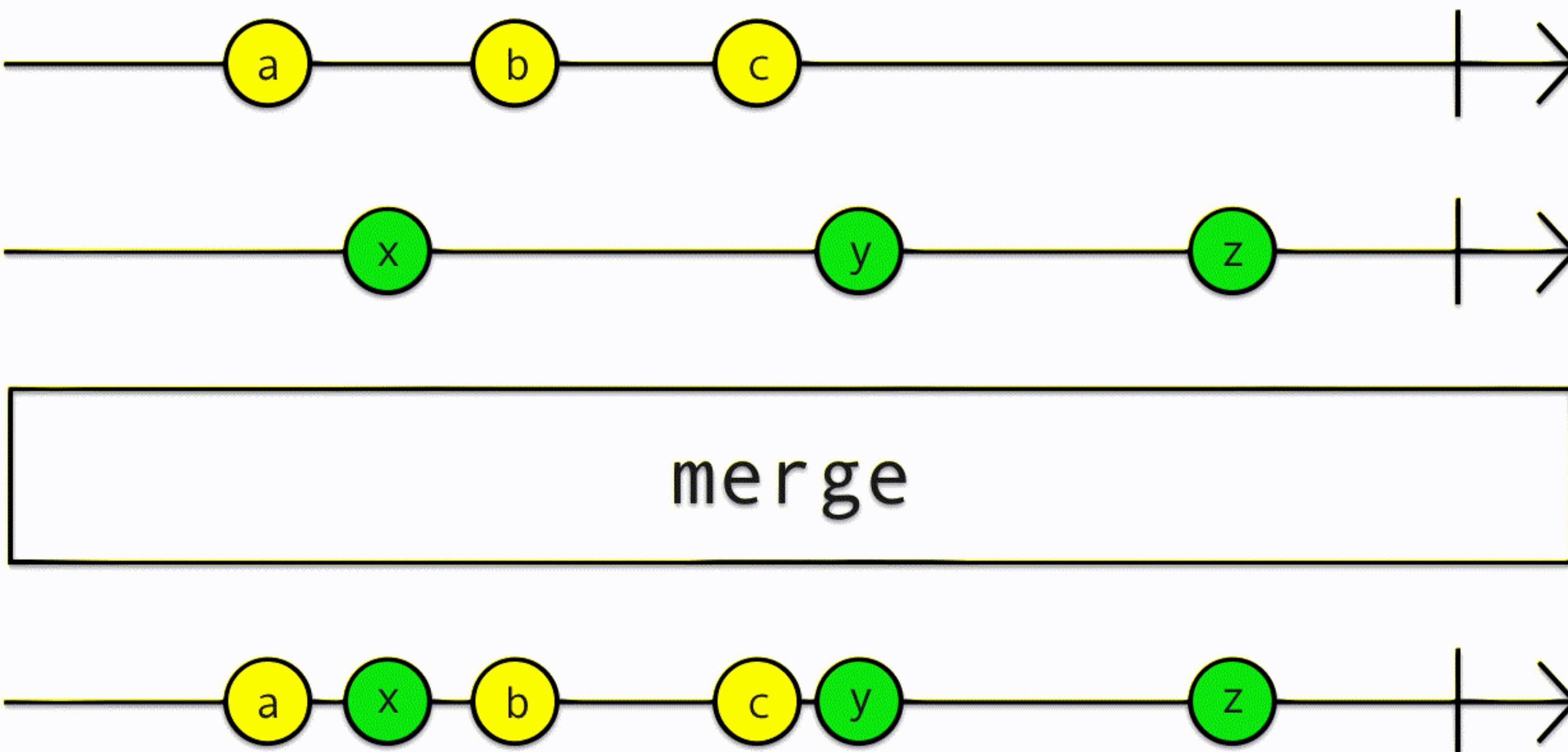




## CONCAT

Waits for inner Observables to complete





## MERGE

Subscribe to multiple inner Observables at a time



```
// using map with nested subscribe
from([1, 2, 3, 4]).pipe(
  map(param => getData(param))
).subscribe(val => val.subscribe(data => console.log(data)));

// using map and mergeAll
from([1, 2, 3, 4]).pipe(
  map(param => getData(param)),
  mergeAll()
).subscribe(val => console.log(val));

// using mergeMap
from([1, 2, 3, 4]).pipe(
  mergeMap(param => getData(param))
).subscribe(val => console.log(val));
```

Luuk Gruis: <https://medium.com/@luukgruijs/understanding-rxjs-map-mergemap-switchmap-and-concatmap-833fc1fb09ff>

## mergeMap() HIGHER-ORDER TRANSFORMATION OPERATOR

Projects each source value into an Observable that is merged into the output Observable

```
combineLatest( [notifications$, otherPrimaryActivities$] )
  .subscribe({
    next: ( [notification, otherPrimaryActivity] ) => {
      // fires whenever notifications$ _or_ otherPrimaryActivities$ updates
      // but not until all sources have emitted at least one value
    }
  });

notifications$.pipe( withLatestFrom(mostRecentUpdates$) )
  .subscribe({
    next: ( [notification, mostRecentUpdate] ) => {
      // fires only when notifications$ updates and includes latest from mostRecentUpdates$
    }
  });

```

<https://gist.github.com/sandikbarr/36bb0bf0b99a82a74be92aba1b1d0482#file-combinelatest-withlatestfrom-ts>

## combineLatest()

Updates from the latest values of each input Observable

## withLatestFrom()

Also provide the latest value from another Observable



```

@Component({
  selector: 'app-many-subscriptions',
  template: `<p>value 1: {{value1}}</p>
              <p>value 2: {{value2}}</p>
              <p>value 3: {{value3}}</p>`
})
export class SubscriberComponent implements OnInit, OnDestroy {
  value1: number;
  value2: number;
  value3: number;

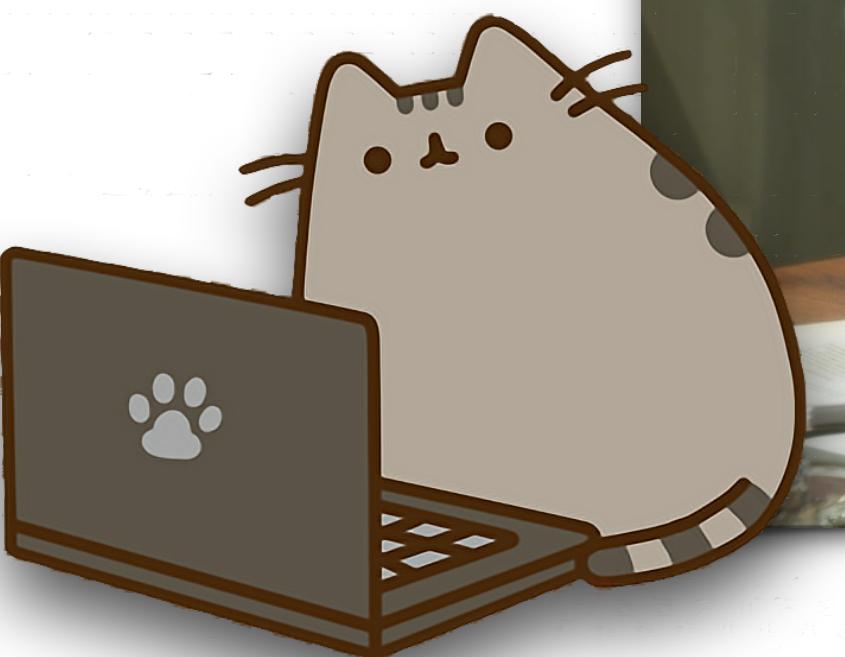
  destroySubject$: Subject<void> = new Subject();

  constructor(private service: MyService) {}

  ngOnInit() {
    this.service.value1.pipe(
      takeUntil(this.destroySubject$)
    ).subscribe(value => {
      this.value1 = value;
    });
    this.service.value2.pipe(
      takeUntil(this.destroySubject$)
    ).subscribe(value => {
      this.value2 = value;
    });
    this.service.value3.pipe(
      takeUntil(this.destroySubject$)
    ).subscribe(value => {
      this.value3 = value;
    });
  }

  ngOnDestroy() {
    this.destroySubject$.next();
  }
}

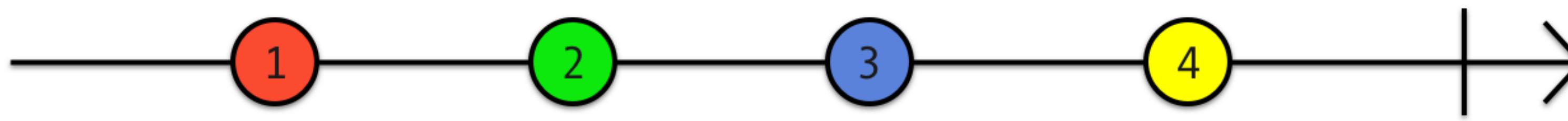
```



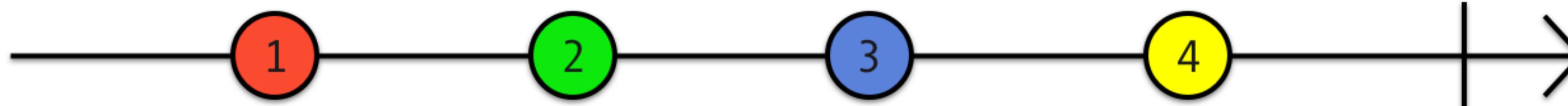
USE A SUBJECT  
TO COMPLETE  
STREAMS  
with this  
one weird trick



Always  
Bring  
Backup



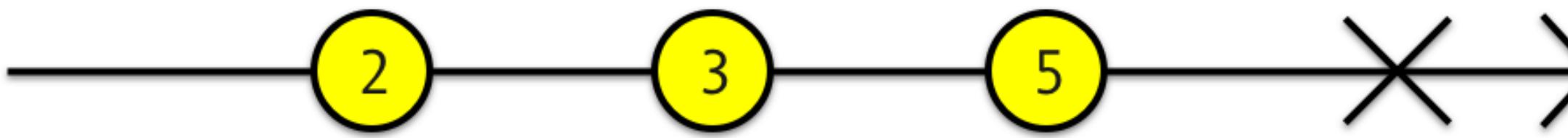
```
tap(console.log)
```



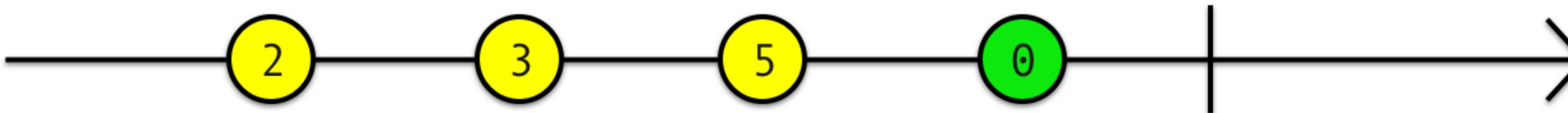
## tap()

### A UTILITY OPERATOR

Perform side effects and return the stream unchanged



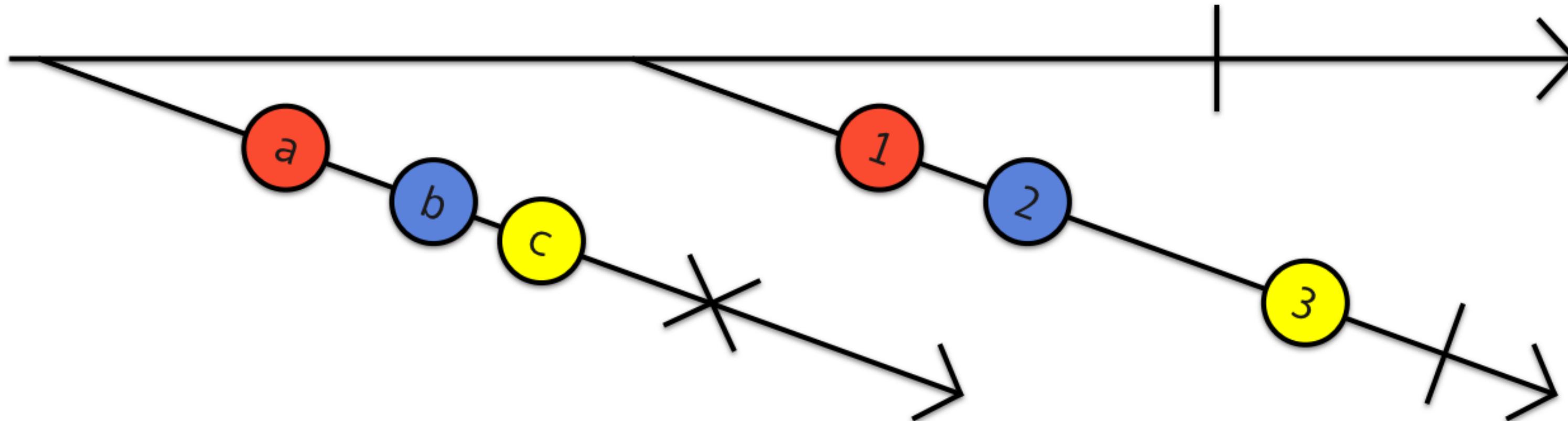
```
catchError(()=> 0((0)) --- End2[ ] )
```



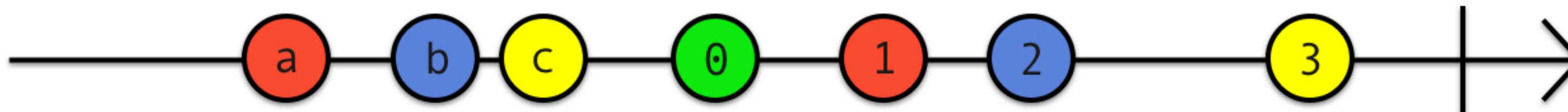
## catchError()

### AN ERROR HANDLING OPERATOR

Catch errors in the stream and return a new Observable



```
catchError(()=> — 0 — | —> )
```



## catchError()

An error can terminate a stream and send the error to the `error()` callback, or the stream can be allowed to continue if piped through `catchError()`.

## CREATION

fromEvent  
interval  
of

## JOIN CREATION

merge  
concat  
combineLatest

## TRANSFORMATION

map  
mergeMap  
scan

## FILTERING

filter  
debounceTime  
distinctUntilChanged

## JOIN

mergeAll  
withLatestFrom

## MULTICASTING

share  
publish  
publishReplay

## ERROR HANDLING

catchError  
retry

## UTILITY

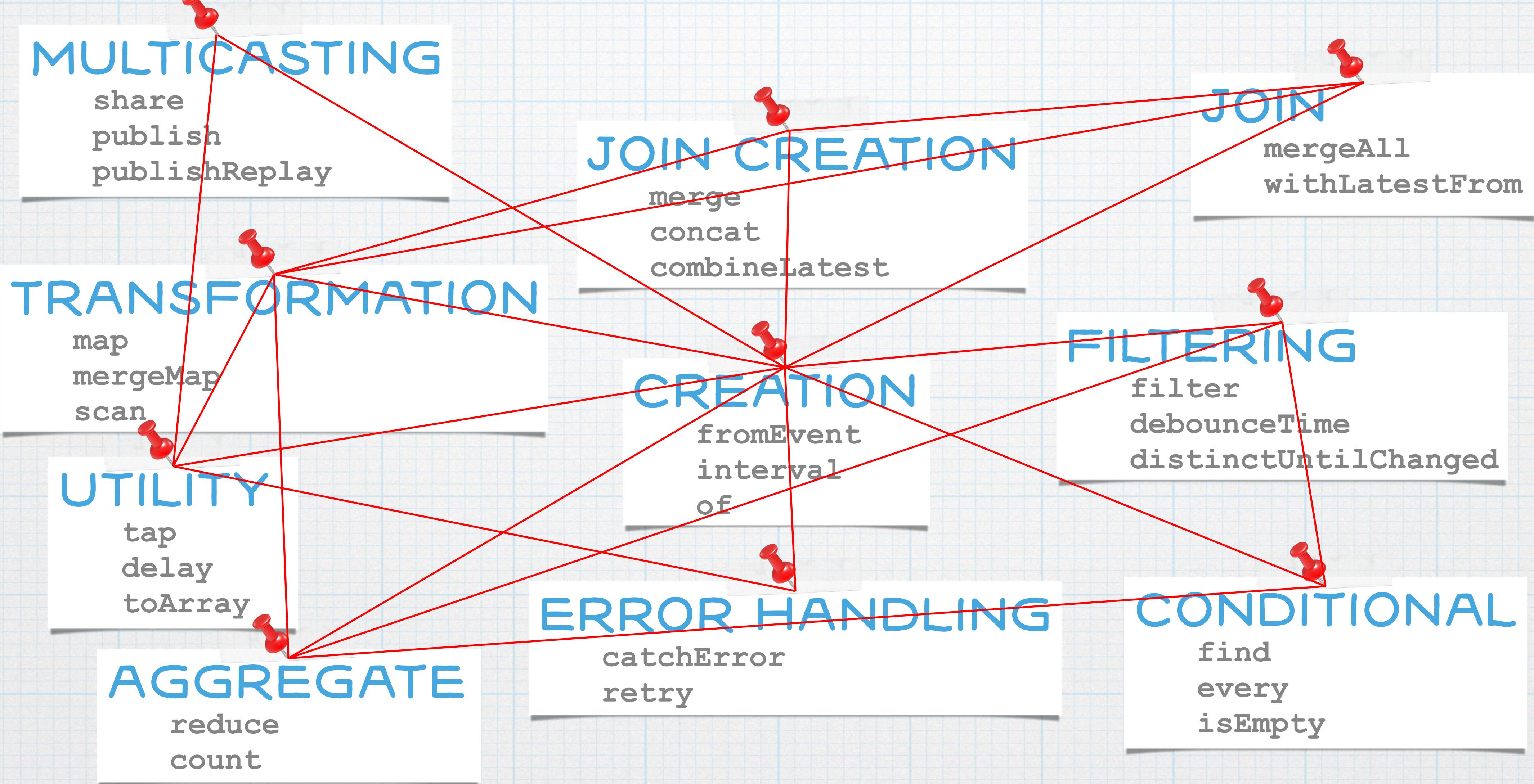
tap  
delay  
toArray

## CONDITIONAL

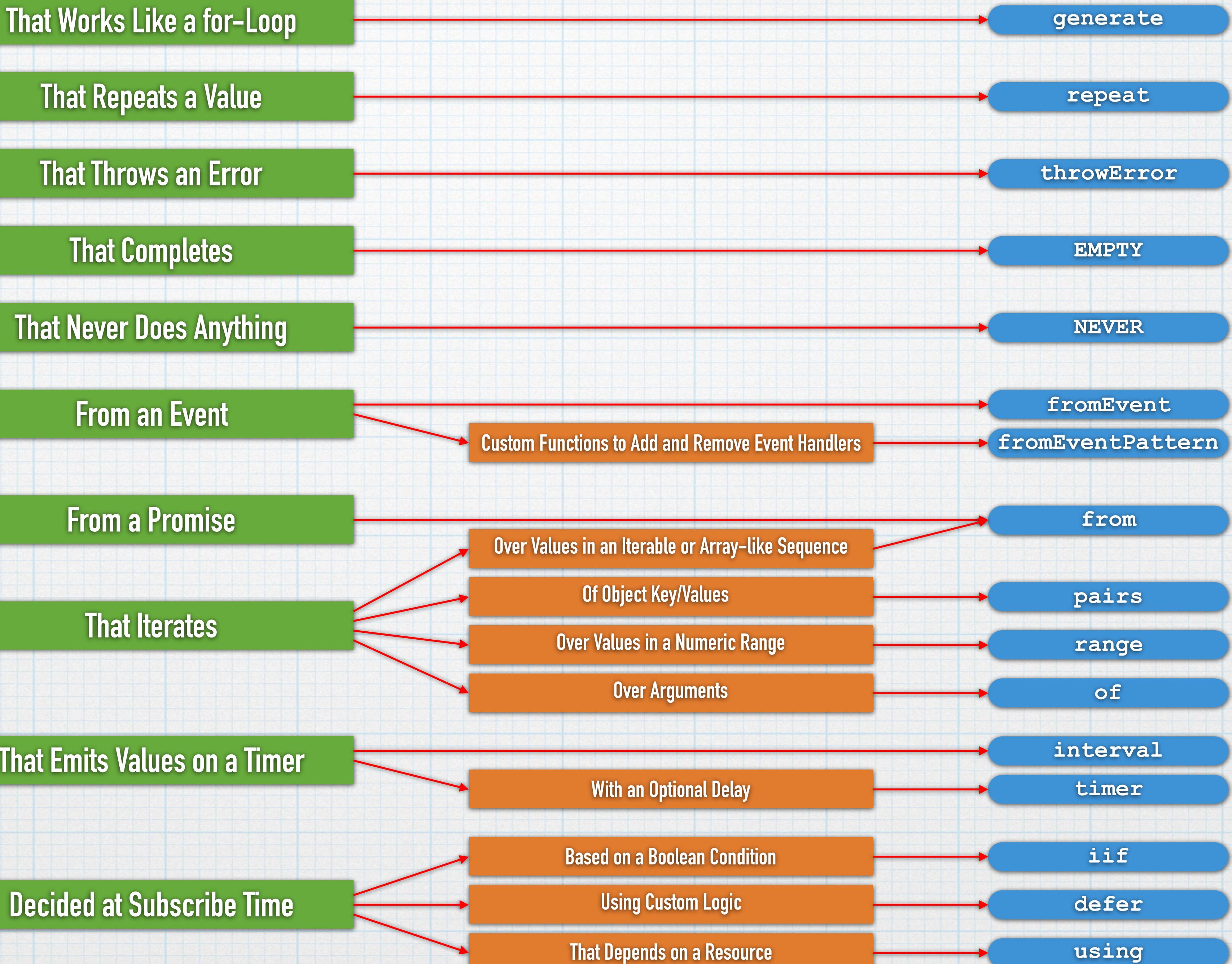
find  
every  
isEmpty

## AGGREGATE

reduce  
count



# Create a New Observable Sequence



# TROUBLESHOOTING

- \* Has this Observable been subscribed to?
- \* How many Subscriptions does this Observable have?
- \* When does this Observable complete? Does it complete?
- \* Do I need to unsubscribe from this Observable?



# THANKS!



example code: <https://github.com/sandikbarr/rxjs-todo>  
slides: <https://www.slideshare.net/secret/FL6NONZJ7DDAkf>