Angular and The Case for RxJS

Sandi K. Barr
Senior Software Engineer

PRX

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	Promise

Create:	(lazy)
	`new Observable((observer) =>`
	`  observer.next(123));`
		(eager)
		`new Promise((resolve, reject) =>`
		`  resolve(123));`

Transform:	`obs$.pipe(map((value) =>` `  value * 2));`	`promise.then((value) =>` `  value * 2);`
Subscribe:	`sub = obs$.subscribe((value) =>` `  console.log(value));`	`promise.then((value) =>` `  console.log(value));`
Unsubscribe:	`sub.unsubscribe();`	`// implied by promise resolution`

https://angular.io/guide/comparing-observables#cheat-sheet

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

```
 9
10   const subscription = observable.subscribe(observer);
11
12   subscription.unsubscribe();
13
```

Observer

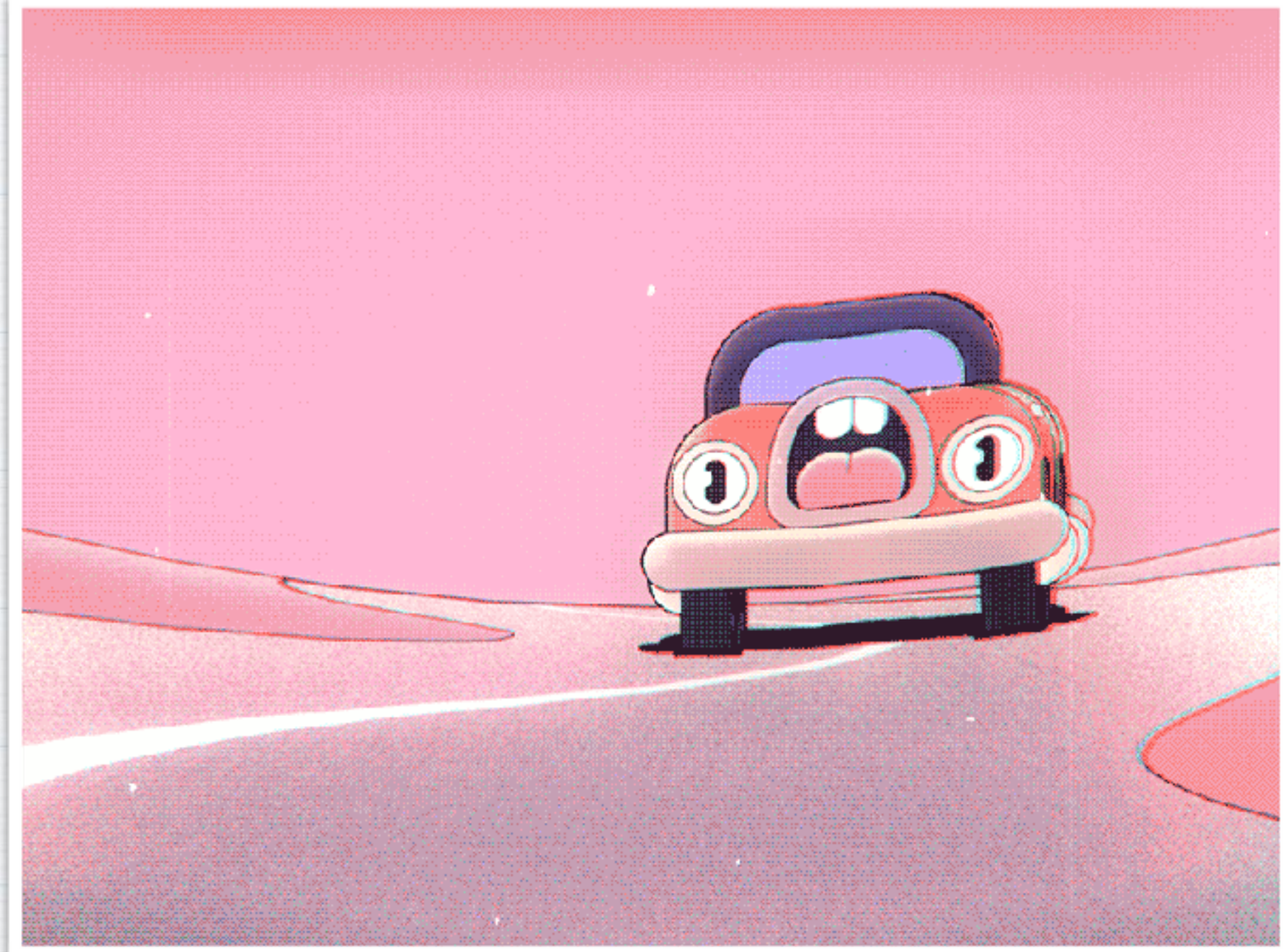
an object with `next`, `error`, **and** `complete` **methods**

```
4   const observer = {
5     next: value => console.log('Next value:', value),
6     error: err => console.error('Error:', err),
7     complete: () => console.log('Complete')
8   };
9
10  const subscription = observable.subscribe(observer);
11
```

Subscription

an Observable only produces values on `subscribe()`

One ringy dingy,
Two ringy dingies...



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
});
```

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

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

COLD OBSERVABLES

The producer is created during the subscription.

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

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

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

HOT OBSERVABLES

The producer is created outside the subscription.

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

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

SUBJECT: **both an observable and an observer**

Make a cold Observable hot.

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

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.

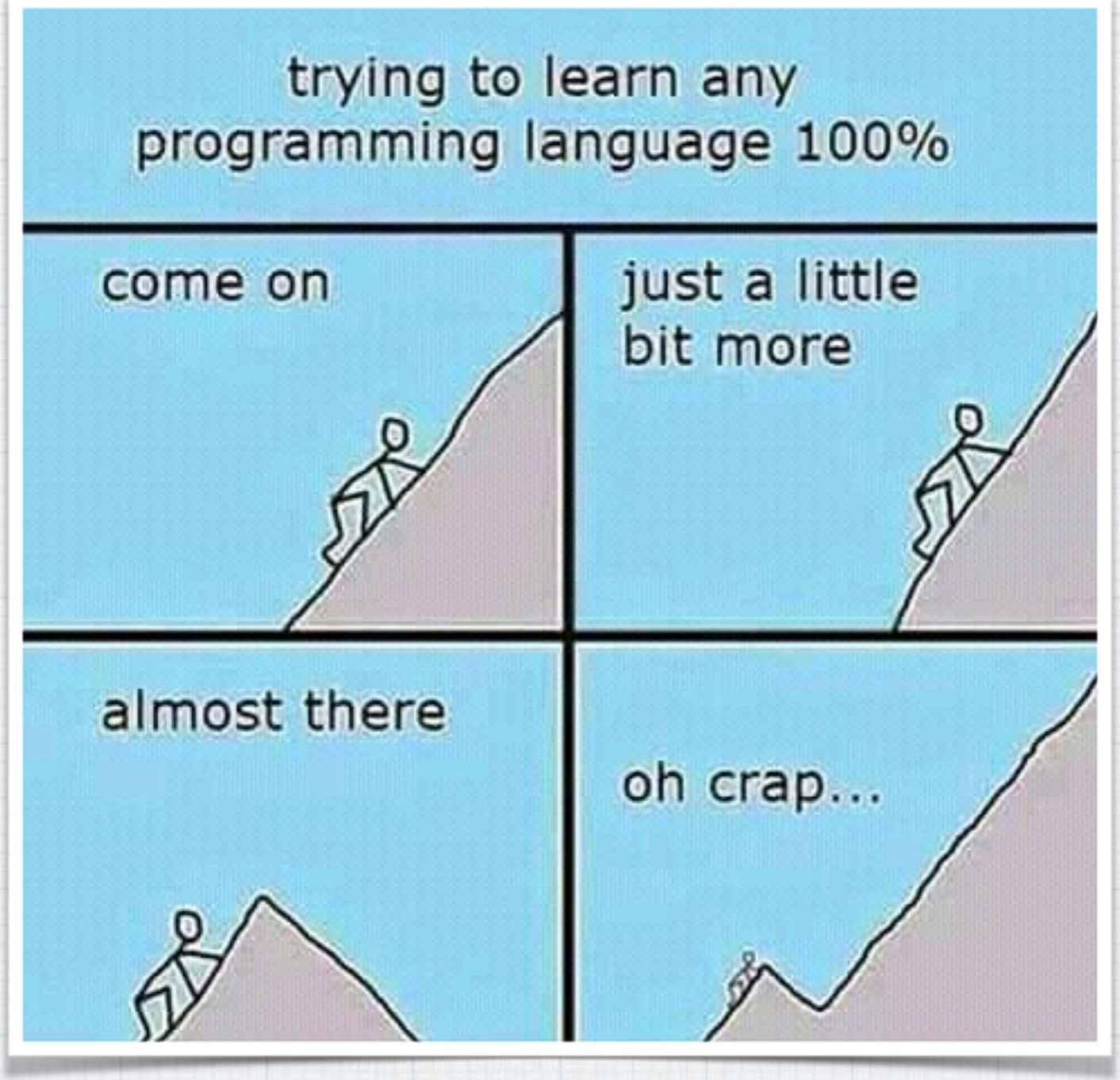
2 to take place; happen; occur:
The wedding was last week.

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.



RxJS

STEEP LEARNING CURVE

```typescript
@Injectable()
export class DobyStoreService {

  private _dobys: BehaviorSubject<Doby[]> = new BehaviorSubject([]);

  constructor(private dobyAPI: DobyHttpService) {
      this.loadInitialData();
  }

  get dobys(): Observable<Doby[]> {
      return this._dobys.asObservable();
  }

  loadInitialData() {
    this.dobyAPI.getAllDobys()
      .subscribe(
          dobys => {
              this._dobys.next(dobys);
          },
          err => console.log('Error retrieving Dobys')
      );
  }
```

https://github.com/sandikbarr/do-by/blob/master/src/app/doby/store/doby-store.service.ts

STORE: OBSERVABLE DATA SERVICE

Provide data to multiple parts of an application.

```
1  let BehaviourSubject = new BehaviorSubject(initialState);
2
```

https://gist.github.com/jhades/f2e97da5a4d977fabb0e7857b6420d10#file-behaviour-subject-ts

```
1  let currentValue = behaviorSubject.getValue();
2
```

https://gist.github.com/jhades/f2e97da5a4d977fabb0e7857b6420d10#file-current-value-ts

BehaviorSubject

Just because you can access the value directly...

```
6
7   @Injectable()
8   export class DobyStoreService {
9
10    private _dobys: BehaviorSubject<Doby[]> = new BehaviorSubject([]);
11
12    constructor(private dobyAPI: DobyHttpService) {
13        this.loadInitialData();
14    }
15
16    get dobys(): Observable<Doby[]> {
17        return this._dobys.asObservable();
18    }
19
```

https://github.com/sandikbarr/do-by/blob/master/src/app/doby/http/doby-http.service.ts

PROTECT THE `BehaviorSubject` WITH `asObservable()`

Components can subscribe after the data arrives.

```
@Component({
  selector: 'app-doby-list',
  template: `
    <ul>
      <app-doby-list-item
        *ngFor="let doby of dobys"
        [doby]="doby">
      </app-doby-list-item>
    </ul>
  `
})
export class DobyListComponent implements OnInit, OnDestroy {
  dobys: Doby[];
  dobysSubscription: Subscription;

  constructor (private dobyStore: DobyStoreService) {}

  ngOnInit() {
    this.dobysSubscription = this.dobyStore.dobys.subscribe((dobys: Doby[]) => {
      this.dobys = dobys;
    });
  }

  ngOnDestroy() {
    if (this.dobysSubscription) { this.dobysSubscription.unsubscribe(); }
  }
}
```

```
@Component({
  selector: 'app-doby-list',
  template: `
    <ul>
      <app-doby-list-item
        *ngFor="let doby of dobys$ | async"
        [doby]="doby">
      </app-doby-list-item>
    </ul>
  `
})
export class DobyListComponent {
  dobys$: Observable<Doby[]> = this.dobyStore.dobys;

  constructor (private dobyStore: DobyStoreService) {}
}
```

AVOID MEMORY LEAKS

Use the async pipe. Subscriptions live until a stream is completed or until they are manually unsubscribed.

- Decouple responsibilities
- Observable Data Service
  - Provide data to multiple parts of an application
  - Not centralized state management

```
29
30      addDoby(newDoby: Doby): Observable<Doby> {
31        const observable = this.dobyAPI.saveDoby(newDoby);
32
33        observable.subscribe((savedDoby: Doby) =>
34          this._dobys.next(this._dobys.getValue().concat([savedDoby]))
35        );
36
37        return observable;
38      }
39
```

https://github.com/sandikbarr/do-by/blob/master/src/app/doby/store/doby-store.service.ts

STORE: OBSERVABLE DATA SERVICE

Action method updates the store on success.

```
29
30    addDoby(newDoby: Doby): Observable<Doby> {
31      const observable = this.dobyAPI.saveDoby(newDoby);
32
33      observable.subscribe((savedDoby: Doby) =>
34        this._dobys.next(this._dobys.getValue().concat([savedDoby]))
35      );
36
37      return observable;
38    }
39
```

https://github.com/sandikbarr/do-by/blob/master/src/app/doby/store/doby-store.service.ts

HTTP OBSERVABLES ARE COLD

Avoid duplicating HTTP requests.

```
11
12  @Injectable()
13  export class DobyHttpService {
14
15    constructor(private http: HttpClient) { }
16
17    getAllDobys(): Observable<Doby[]> {
18      return this.http.get<Doby[]>('http://localhost:3000/dobys');
19    }
20
21    saveDoby(newDoby: Doby): Observable<Doby> {
22      return this.http.post<Doby>('http://localhost:3000/dobys', newDoby, {headers}).pipe(share());
23    }
24
```

https://github.com/sandikbarr/do-by/blob/master/src/app/doby/http/doby-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

```
11
12  @Injectable()
13  export class DobyHttpService {
14
15    constructor(private http: HttpClient) { }
16
17    getAllDobys(): Observable<Doby[]> {
18      return this.http.get<Doby[]>('http://localhost:3000/dobys');
19    }
20
21    saveDoby(newDoby: Doby): Observable<Doby> {
22      return this.http.post<Doby>('http://localhost:3000/dobys', newDoby, {headers}).pipe(share());
23    }
24
```

https://github.com/sandikbarr/do-by/blob/master/src/app/doby/http/doby-http.service.ts

share() : refCount

share() also makes Observables retry-able

🔥HOT: SHARE THE EXECUTION

⛄COLD: INVOKE THE EXECUTION

```
6   @Component({
7     selector: 'app-doby',
8     template: `
9       Total #: {{ total$ | async }}
10      <app-doby-list [dobys]="dobys$ | async"></app-doby-list>
11    `
12  })
13  export class DuplicateHttpDobyComponent {
14    dobys$ = this.http.get<Doby[]>('http://localhost:3000/dobys');
15    total$ = this.dobys$.pipe(map(dobys => dobys.length));
16
17    constructor(private http: HttpClient) {}
18  }
```

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

async : Unsubscribes automatically

But still be sure to avoid duplicating HTTP requests!

```
5   @Component({
6     selector: 'app-doby',
7     template: `
8       <ng-container *ngIf="dobys$ | async as dobys">
9         Total #: {{ dobys.length }}
10        <app-doby-list [dobys]="dobys"></app-doby-list>
11      </ng-container>
12    `
13  })
14  export class DuplicateHttpDobyComponent {
15    dobys$ = this.http.get<Doby[]>('http://localhost:3000/dobys');
16
17    constructor(private http: HttpClient) {}
18  }
```

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

ngIf with | async as

Assign Observable values to a local variable.

```
5   @Component({
6     selector: 'app-doby',
7     template: `
8       <ng-container *ngIf="dobys$ | async as dobys; else loading">
9         Total #: {{ dobys.length }}
10        <app-doby-list [dobys]="dobys"></app-doby-list>
11      </ng-container>
12      <ng-template #loading><p>Loading...</p></ng-template>
13    `
14  })
15  export class DuplicateHttpDobyComponent {
16    dobys$ = this.http.get<Doby[]>('http://localhost:3000/dobys');
17
18    constructor(private http: HttpClient) {}
19  }
```

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

ngIf with| async as with ;ngElse

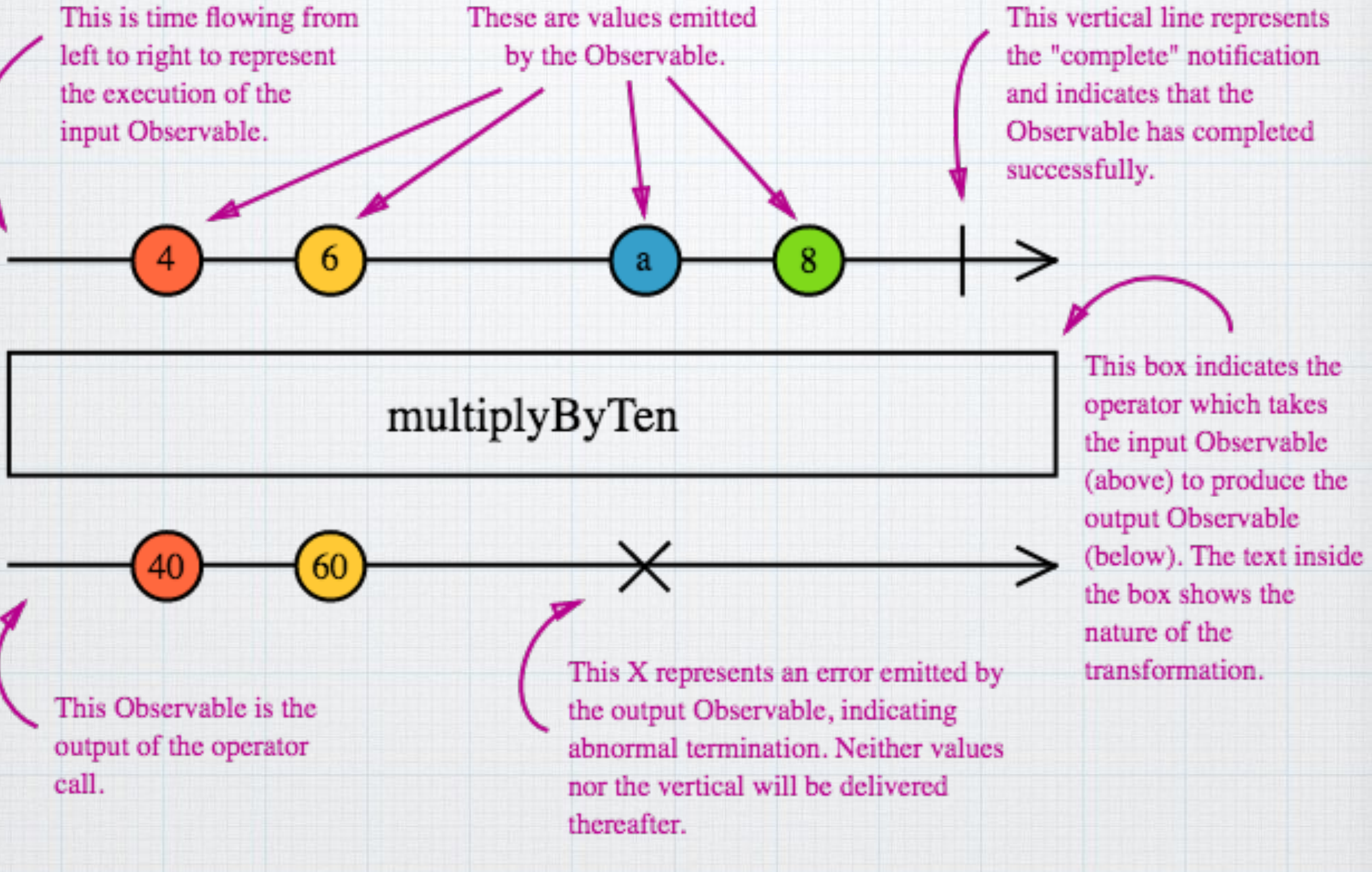
Show alternate block when Observable has no value.

🙌REACTIVE TEMPLATES🙌

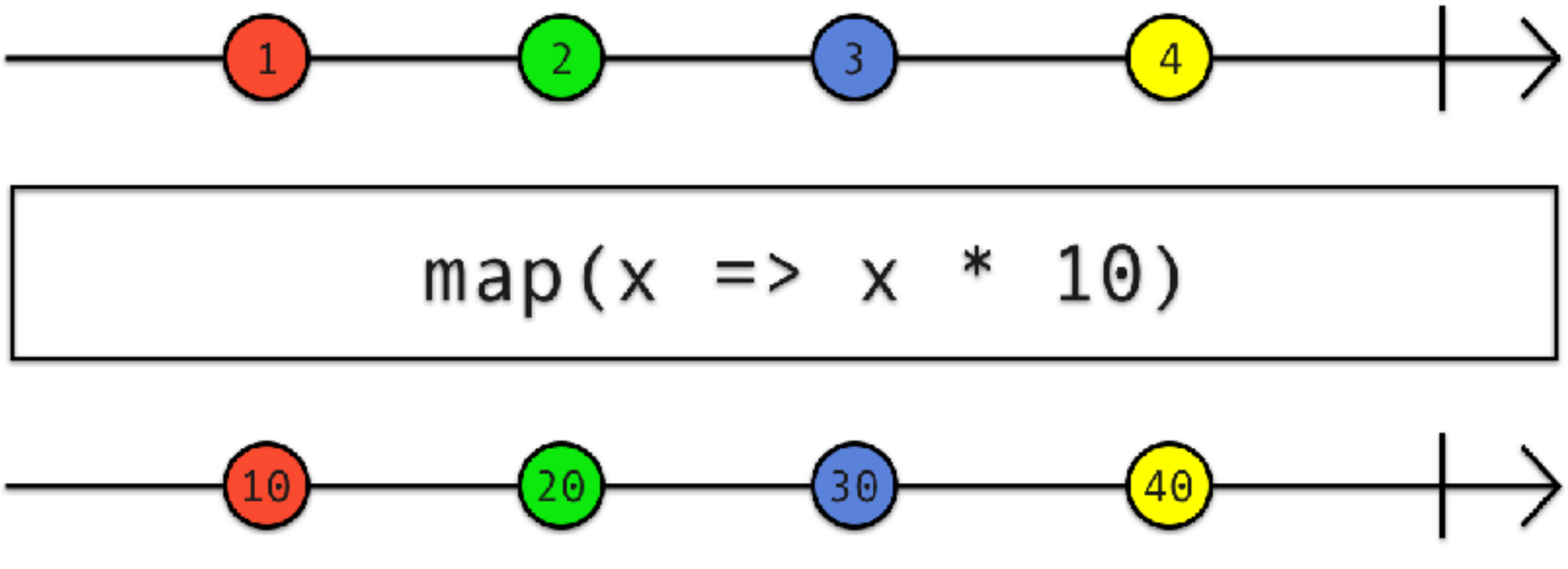
```
import { range } from 'rxjs';
import { map, filter, scan } from 'rxjs/operators';

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))
```

Observable.prototype.pipe()

Compose a series of operators



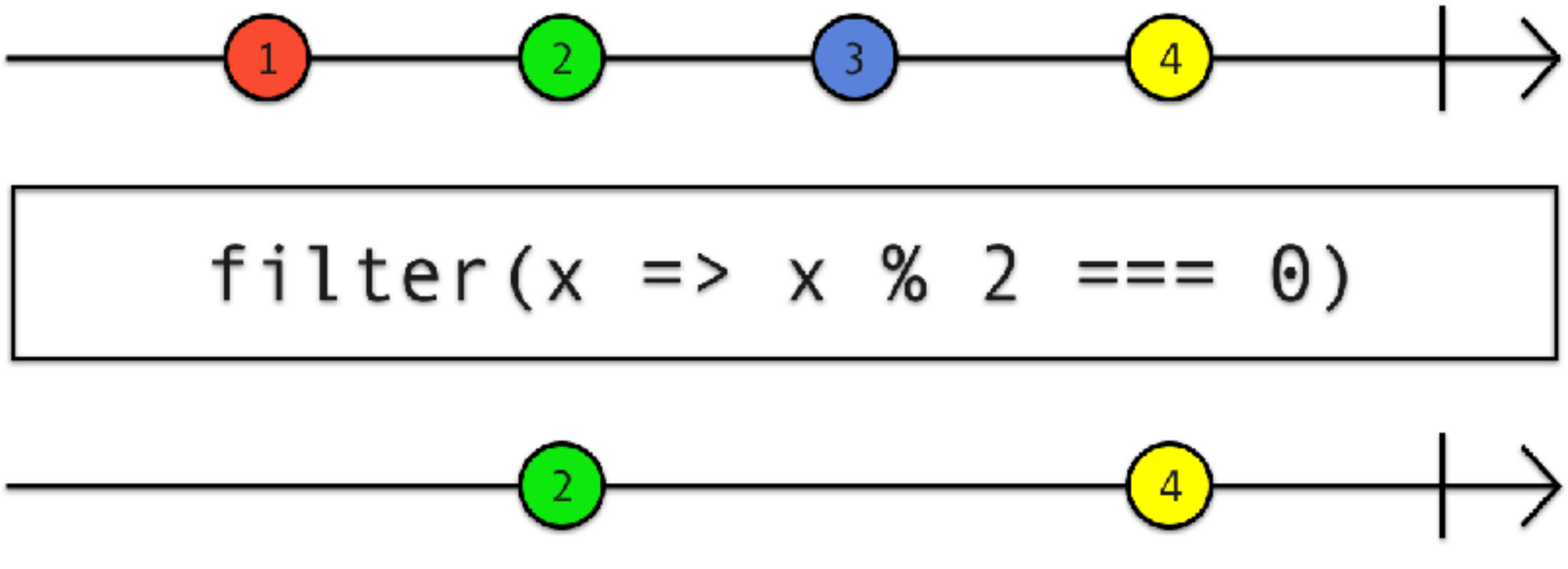
https://rxjs-dev.firebaseapp.com/assets/images/guide/marble-diagram-anatomy.svg



map()

A TRANSFORMATIONAL OPERATOR

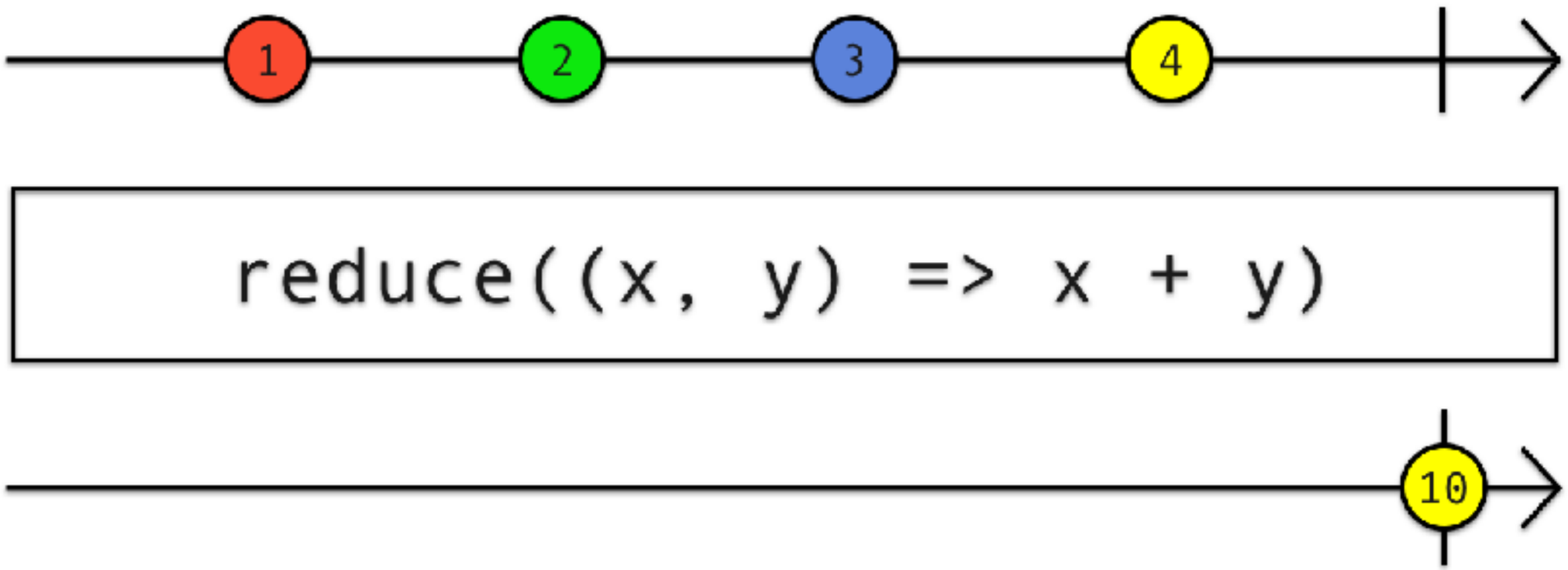
Applies a projection to each value



`filter()`

ONE OF MANY FILTERING OPERATORS

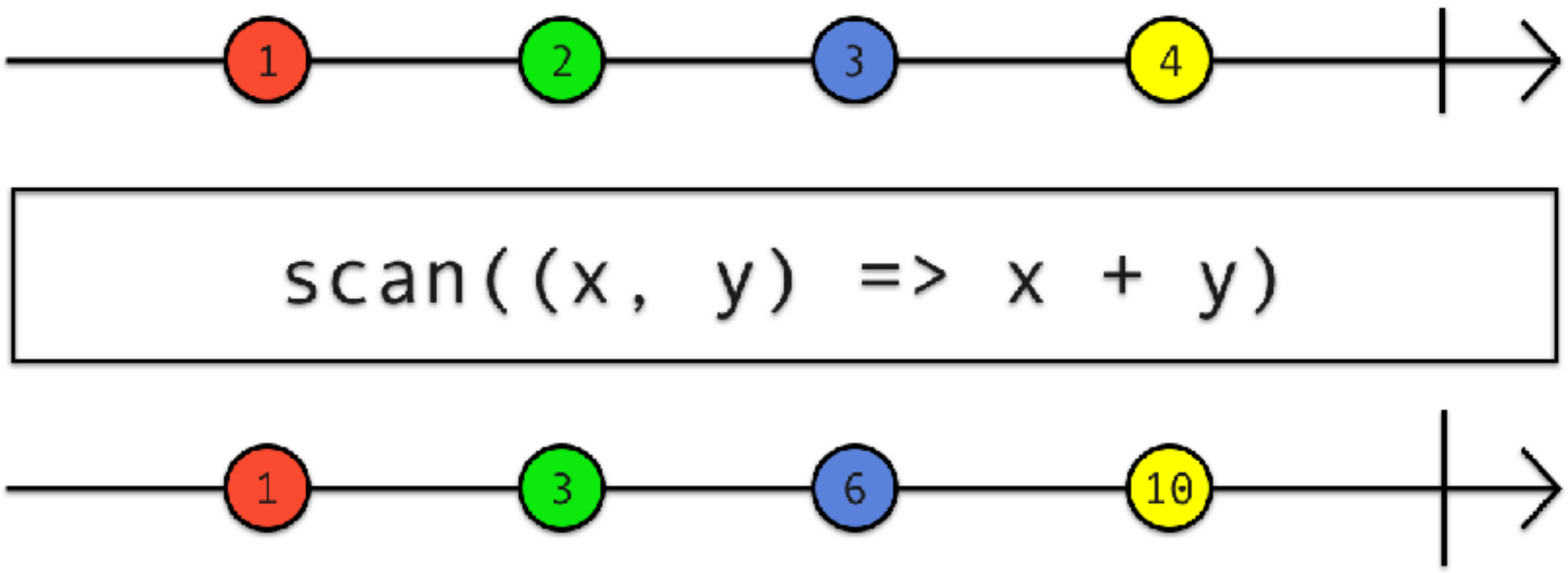
Filters items emitted from source



reduce()

AN AGGREGATE OPERATOR

Emits aggregate result on completion



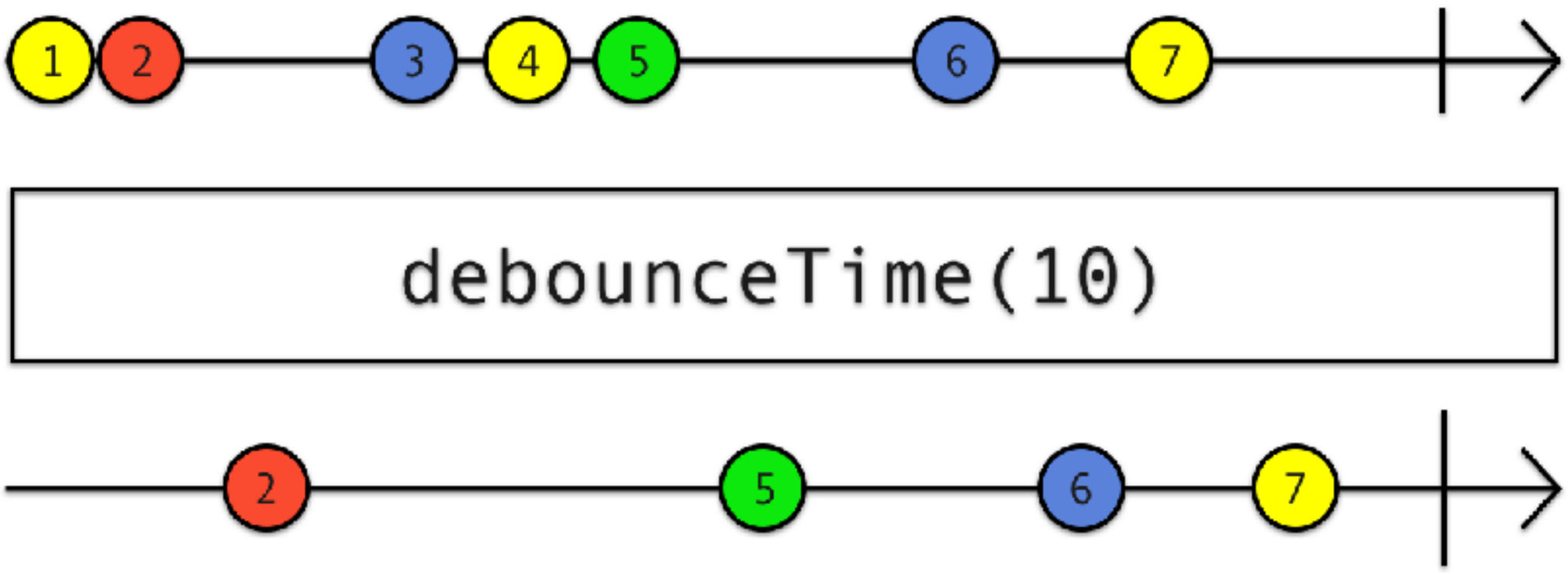
scan()

AN AGGREGATE OPERATOR

Emits accumulated result at each interval

```
@Component({
  selector: 'app-doby-search',
  template: `
    <label for="search">Search: </label>
    <input id="search" (keyup)="onSearch($event.target.value)"/>
  `
})
export class DobySearchComponent 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);
  }
}
```

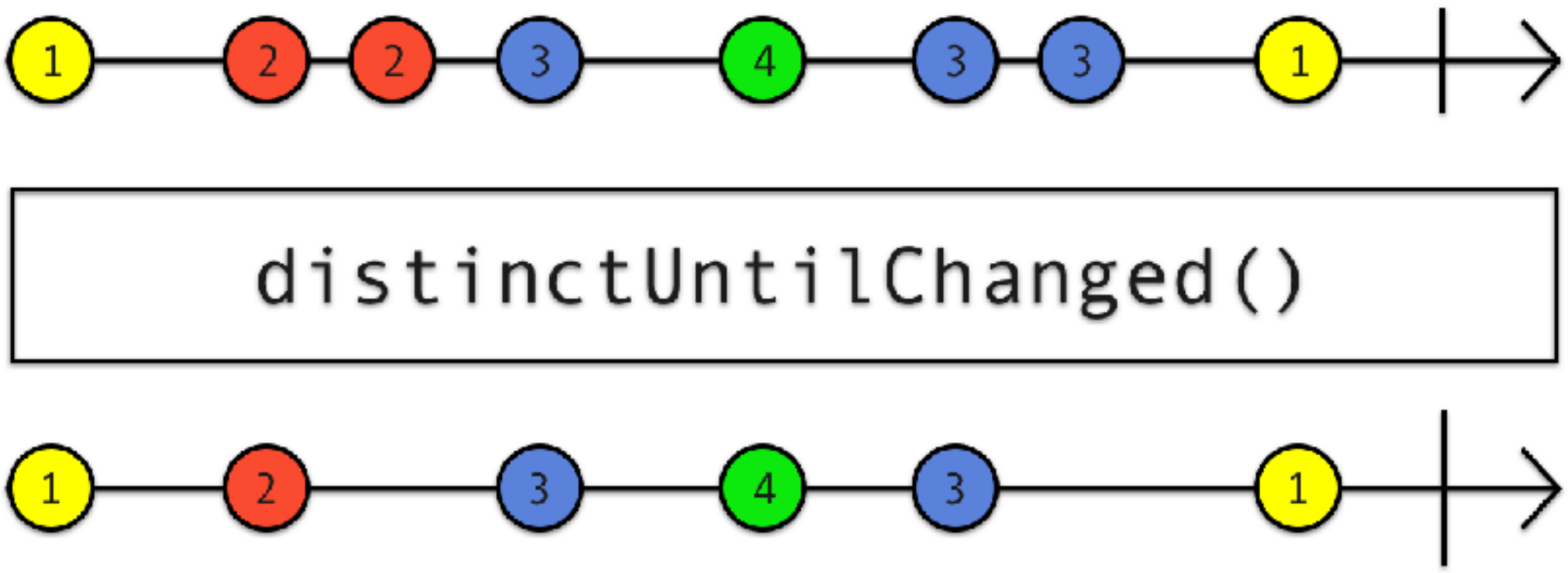
fancy



debounceTime()

A FILTERING OPERATOR

rate-limit and delay



distinctUntilChanged()

A FILTERING OPERATOR

Emits values that are distinct from the previous value

```
20  export class DobyEditComponent implements OnInit {
21    doby: Doby;
22
23    constructor(private dobyStore: DobyStoreService,
24                private route: ActivatedRoute,
25                private router: Router) {}
26
27    ngOnInit() {
28      this.route.params.subscribe(params => {
29        const id = +params['id'];
30        this.dobyStore.dobys.subscribe((dobys: Doby[]) => {
31          this.doby = dobys.find(doby => doby.id === id);
32        });
33      });
34    }
```

AVOID NESTED SUBSCRIPTIONS

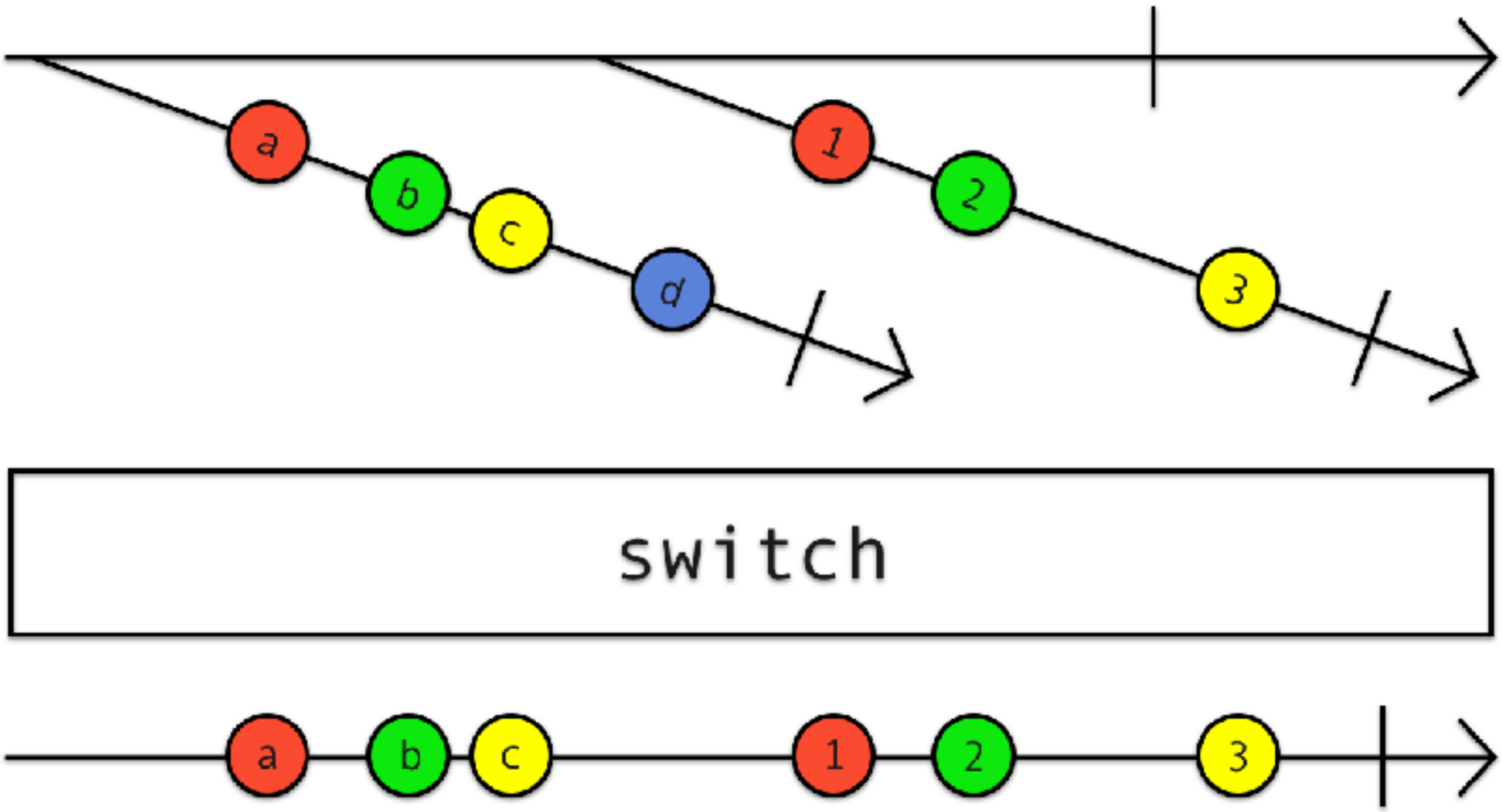

pyramid shaped callback hell


```
24  export class DobyEditComponent {
25    doby$: Observable<Doby> = this.route.params.pipe(
26      map(params => +params['id']),
27      switchMap(id => this.dobyStore.getDobyById(id))
28    );
29
30    constructor(private dobyStore: DobyStoreService,
31                private route: ActivatedRoute,
32                private router: Router) {}
```

https://github.com/sandikbarr/do-by/blob/master/src/app/doby/components/edit/doby-edit.component.ts

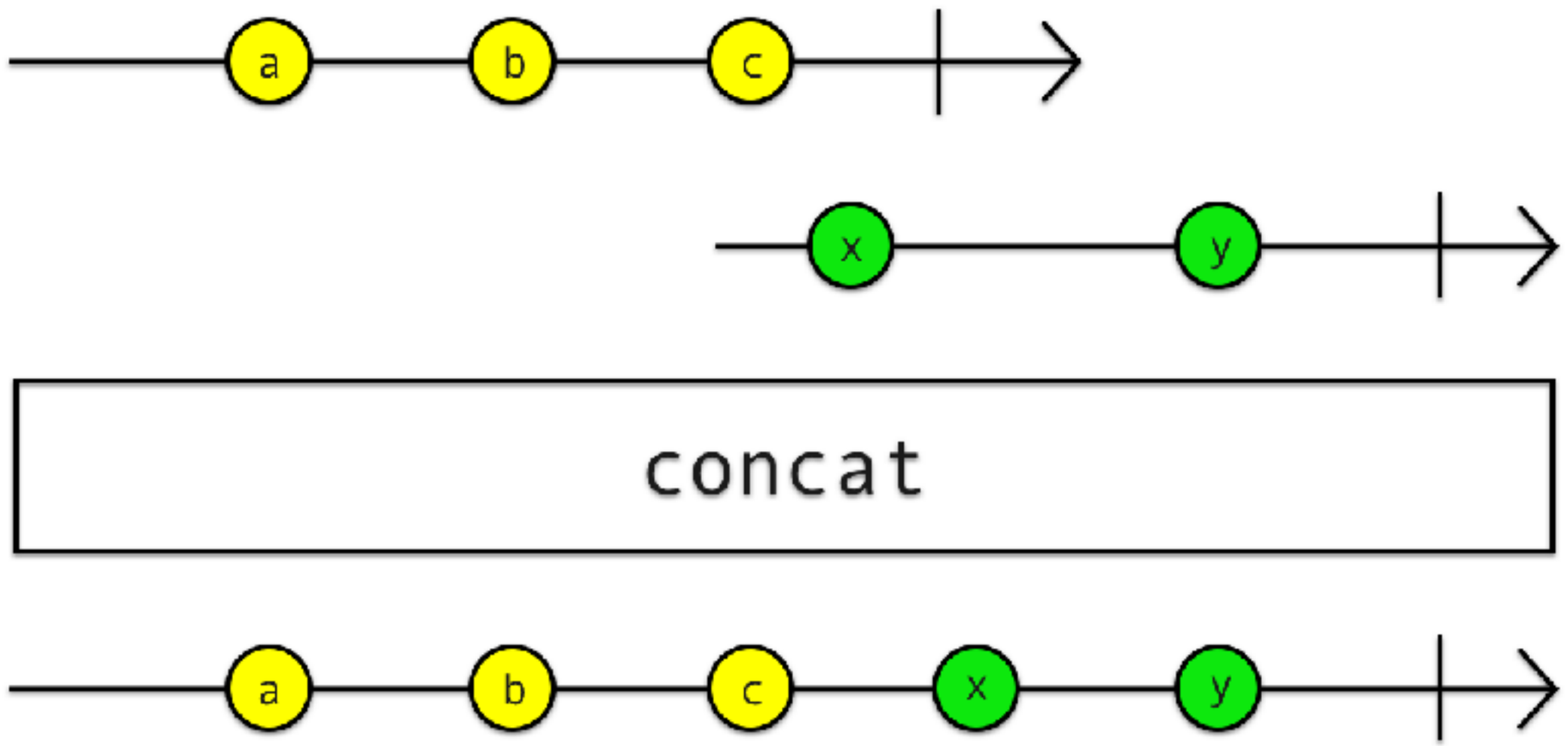
HIGHER-ORDER OBSERVABLES

Observables that emit other Observables



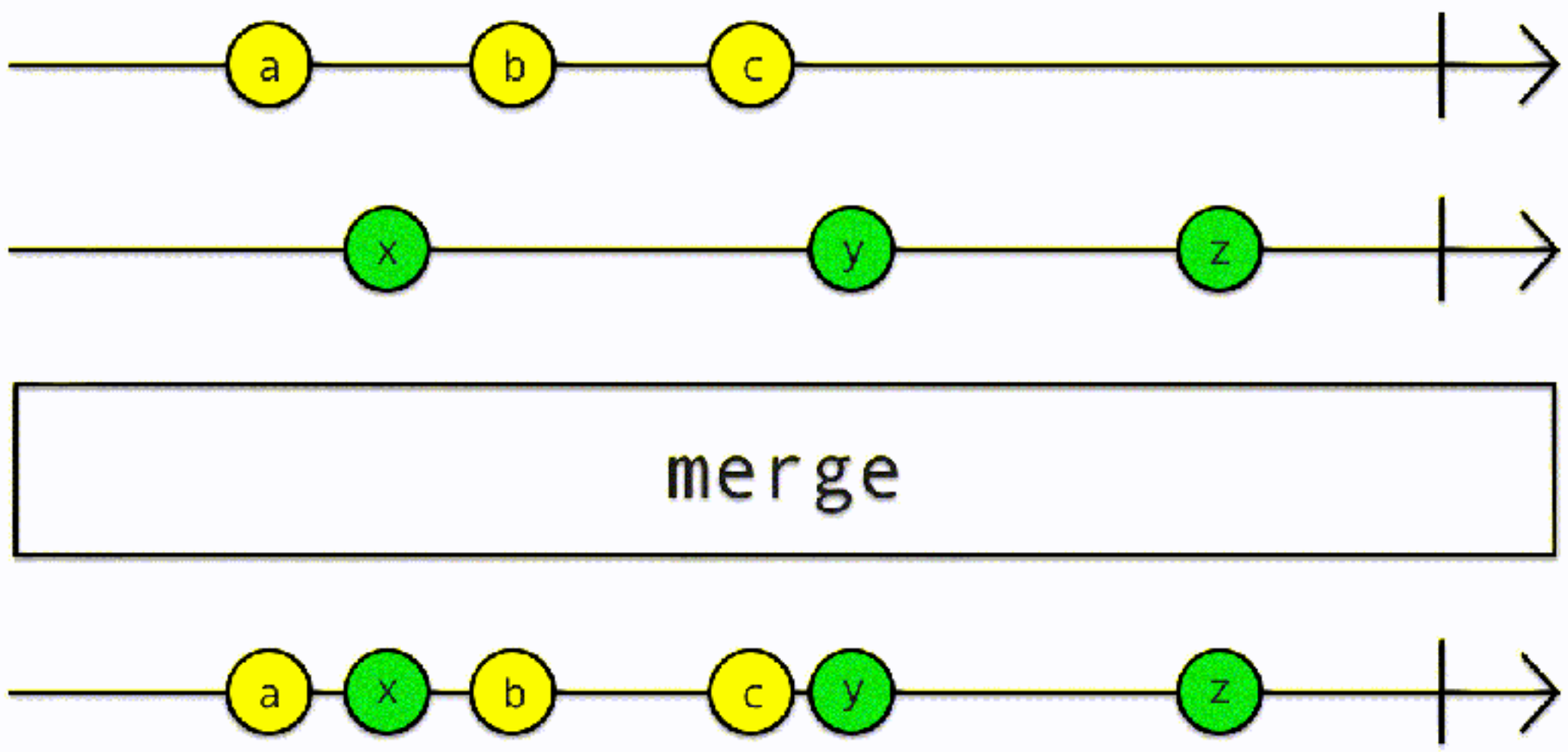
SWITCH

Cancels inner Observables



CONCAT

Waits for inner Observables to complete



MERGE

Subscribe to multiple inner Observables at a time

```
1   // using map with nested subscribe
2   from([1,2,3,4]).pipe(
3     map(param => getData(param))
4   ).subscribe(val => val.subscribe(data => console.log(data)));
5
6   // using map and mergeAll
7   from([1,2,3,4]).pipe(
8     map(param => getData(param)),
9     mergeAll()
10  ).subscribe(val => console.log(val));
11
12  // using mergeMap
13  from([1,2,3,4]).pipe(
14    mergeMap(param => getData(param))
15  ).subscribe(val => console.log(val));
```

https://gist.github.com/sandikbarr/36bb0bf0b99a82a74be92aba1b1d0482#file-map-mergeall-mergemap-ts

mergeMap()

HIGHER-ORDER TRANSFORMATION OPERATOR

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

```
1  import { combineLatest, withLatestFrom } from 'rxjs/operators';
2
3  combineLatest(notifications$, otherPrimaryActiities$)
4    .subscribe( ([notification, otherPrimaryActivity]) => {
5      // fires whenever notifications$ _or_ otherPrimaryActivities$ updates
6    }
7
8  notifications$.pipe( withLatestFrom(mostRecentUpdates$) )
9    .subcribe( ([notification, mostRecentUpdate]) => {
10     // fires only when notifications$ updates and includes latest from mostRecentUpdates$
11   }
```

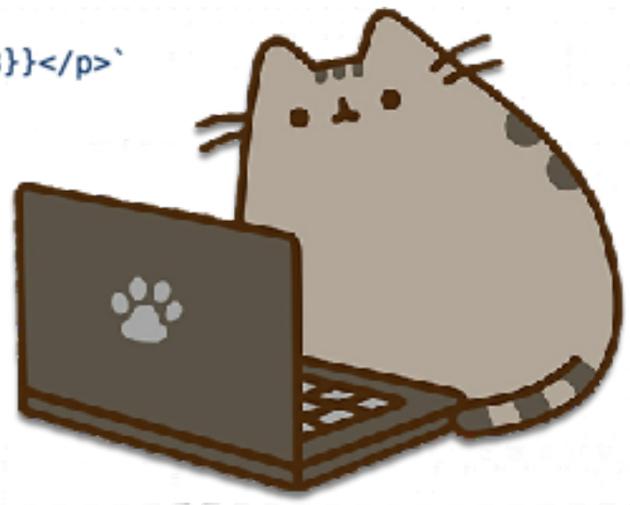
`combineLatest()`

Updates from the latest values of each input Observable

`withLatestFrom()`

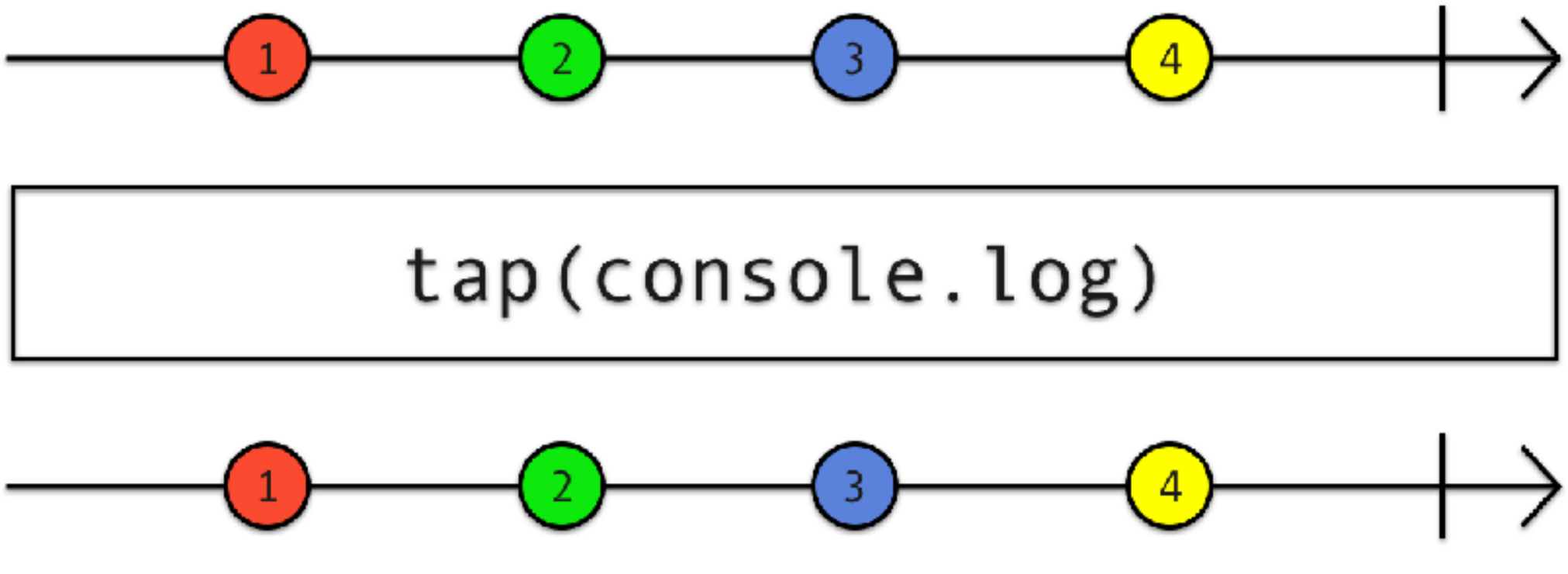
Also provide the latest value from another Observable

```
5
6   @Component({
7     selector: 'app-many-subscriptions',
8     template: `<p>value 1: {{value1}}</p><p>value 2: {{value2}}</p><p>value 3: {{value3}}</p>`
9   })
10  export class SubscriberComponent implements OnInit, OnDestroy {
11    value1: number;
12    value2: number;
13    value3: number;
14
15    destroySubject$: Subject<void> = new Subject();
16
17    constructor(private service: MyService) {}
18
19    ngOnInit() {
20      this.service.value1.pipe(
21        takeUntil(this.destroySubject$)
22      ).subscribe(value => {
23        this.value1 = value;
24      });
25      this.service.value2.pipe(
26        takeUntil(this.destroySubject$)
27      ).subscribe(value => {
28        this.value2 = value;
29      });
30      this.service.value3.pipe(
31        takeUntil(this.destroySubject$)
32      ).subscribe(value => {
33        this.value3 = value;
34      });
35    }
36
37    ngOnDestroy() {
38      this.destroySubject$.next();
39    }
40  }
```



USE A SUBJECT TO COMPLETE STREAMS with this one weird trick

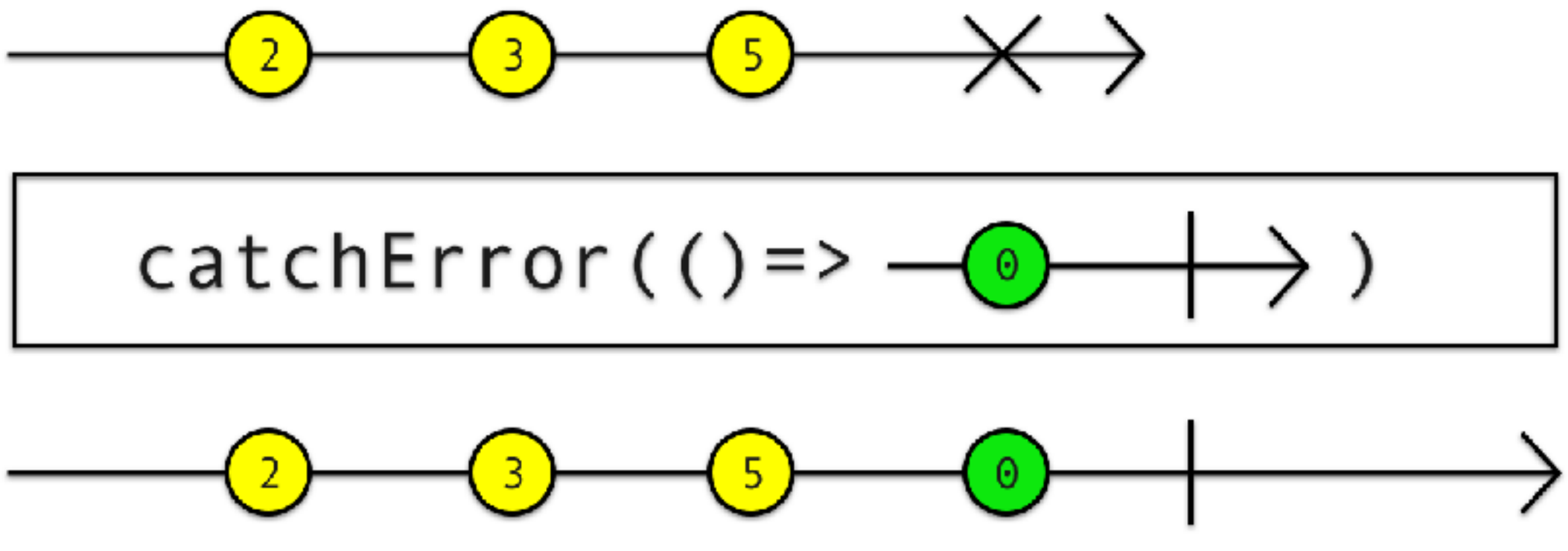
https://gist.github.com/sandikbarr/36bb0bf0b99a82a74be92aba1b1d0482#file-weird-trick-takeuntil-ts



tap()

A UTILITY OPERATOR

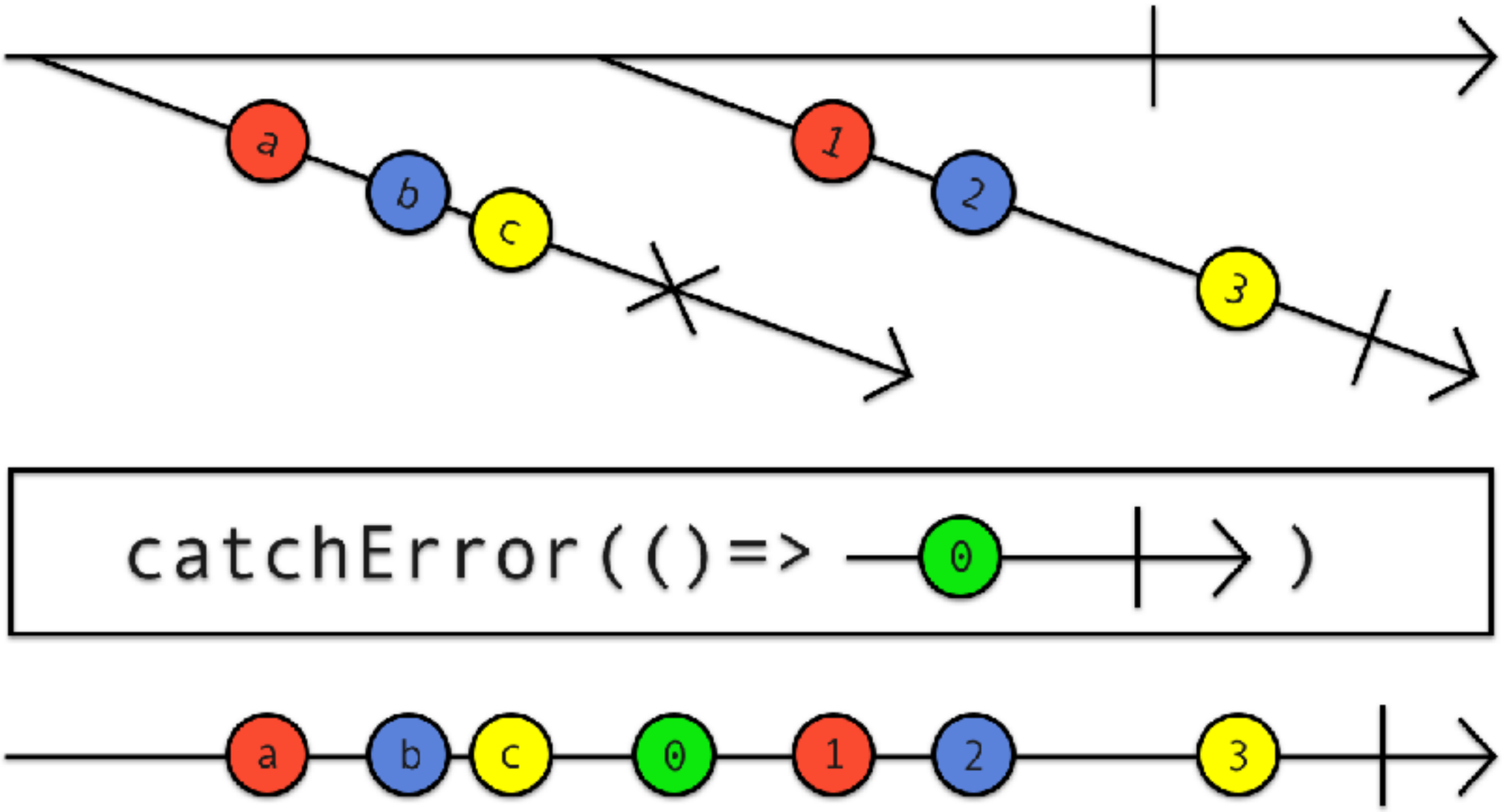
Perform side effects and return the stream unchanged



`catchError()`

AN ERROR HANDLING OPERATOR

Catch errors in the stream and return a new Observable



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
- internal
- 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

https://rxjs-dev.firebaseapp.com/guide/operators

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/do-by