REACTIVE PROGRAMMING WITH OBSERVABLES

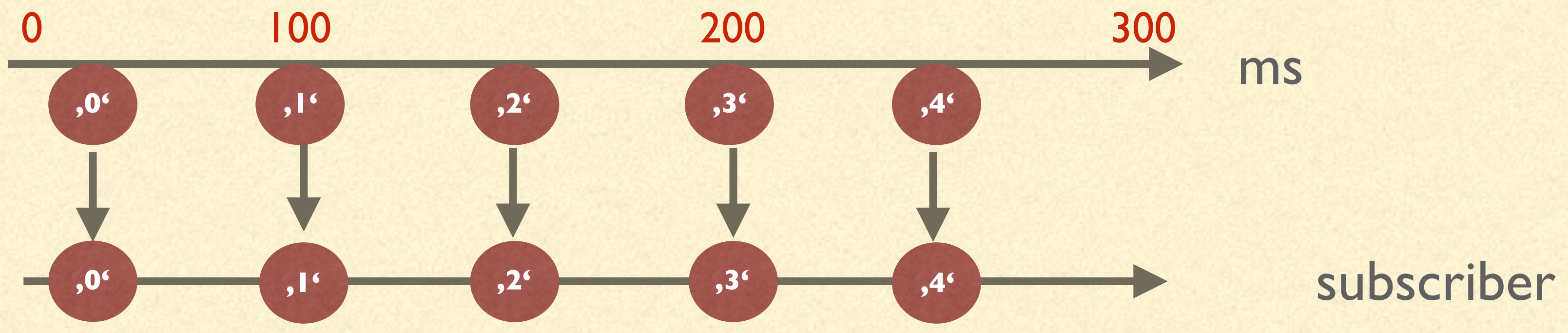
LARS WIEDEMANN, CHEMMEDIA

Github: github.com/gernsdorfer

Twitter: twitter.com/gernsdorfer

Events are Arrays

Simple Example



```
let myObservable$ = Rx.Observable.create(
    (observer) => {
        let counter = 0;
        setInterval(function () {
            observer.onNext(counter.toString());
            counter++
        }, 1000)
    });
myObservable$
    .subscribe(
        item=> console.log('Observer', item),
        err => console.log('error', err),
        complete => console.log('finish')
    );
```

Create an Observable

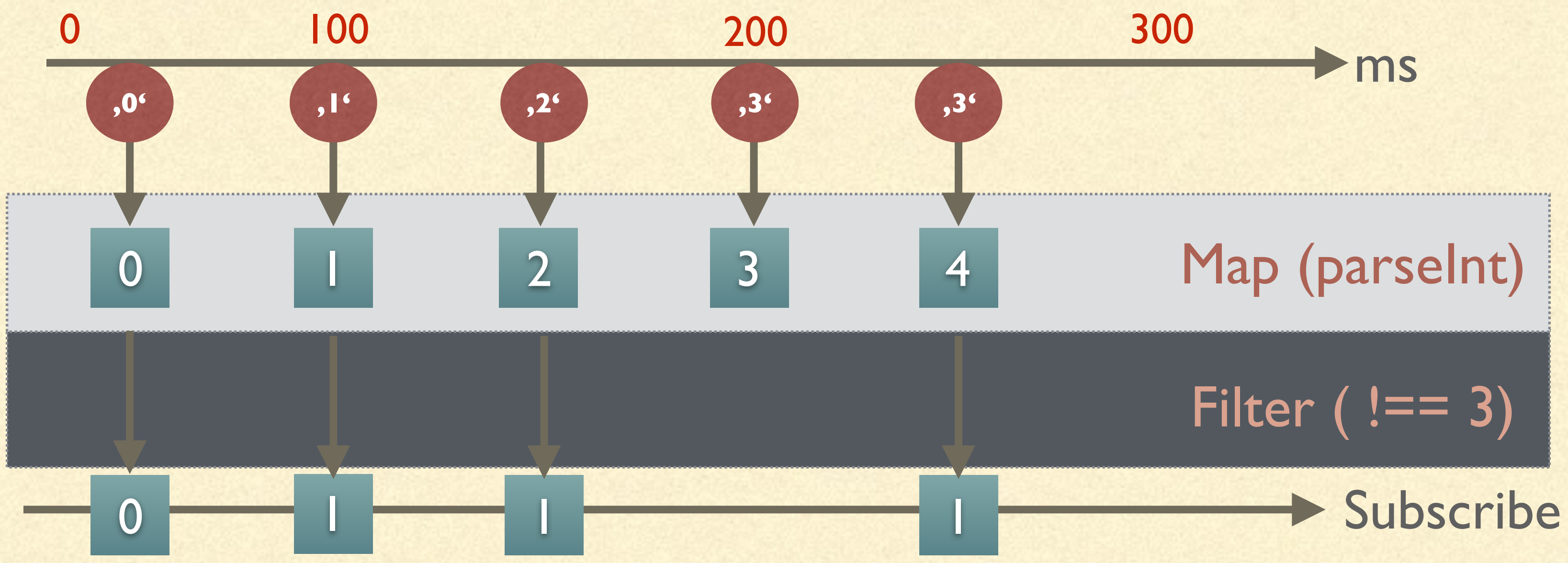
```
Rx.Observable.create(/* Custom */)
Rx.Observable.fromEvent(/* ELEMENT */,/* EVENT */)
Rx.Observable.off(/* VALUES */)
Rx.Observable.fromPromise(/*PROMISE */)

myObservable$.dispose()// remove observable
```

Listen to an Observable

```
let myObservable$ = Rx.Observable.create(/* .... */),
    mySubscribe = myObservable$.subscribe({
          (value)=> /* value */)
          (error)=> /* error */)
          ()=> /* finish */)
mySubscribe.dispose()//remove subscribe
```

Example with operator’s



```
let myObservable$ = Rx.Observable
    .create((observer) => {
        let counter = 0;
        let myTimer = setInterval(function () {
            observer.onNext(counter.toString());
            counter++
        }, 50);
        return () => {
            clearInterval(myTimer);
        }
    })
    .take(5);
myObservable$
    .map((item) => parseInt(item))
    .filter((item) =>item !== 3)
    .subscribe(item=> console.log(item));
```

Operator

transforming

```
myObservable$
      .map((item) => /* NEW VALUE */)
      .flatMap(item) => /* RETURN NEW OBSERVABLE */);
```

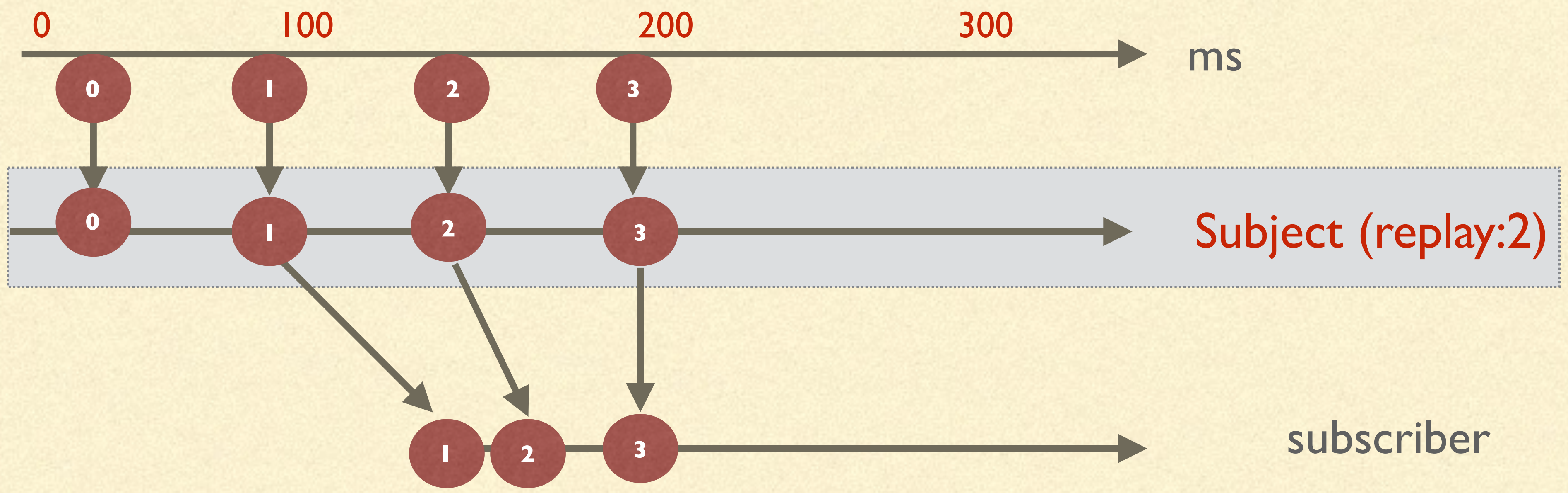
filter

```
myObservable$
     .filter((item) => /* FILTER */)
     .take(/* NUMBER*/);
```

combine

```
myObservable$
     .merge(/* OBSERVABLE A */,/* OBSERVABLE B */);
```

Subject Example



```
let myObservable$ = new Rx.Observable.interval(50).take(4);
let mySubject$ = new Rx.ReplaySubject(2);
myObservable$.subscribe(subject);
setTimeout(()=> {
    mySubject$.subscribe(
        (value) => console.log('observerA: ' + value)//1-2-3

    );
}, 150);
```

Subject

a proxy for Observers

```
new Rx.AsyncSubject()
new Rx.BehaviorSubject(/* Start value */)
new Rx.ReplaySubject(/*Buffersize*/,/*BufferTime*/)
let mySubject = new Rx.Subject();
subject.onNext(3);
```

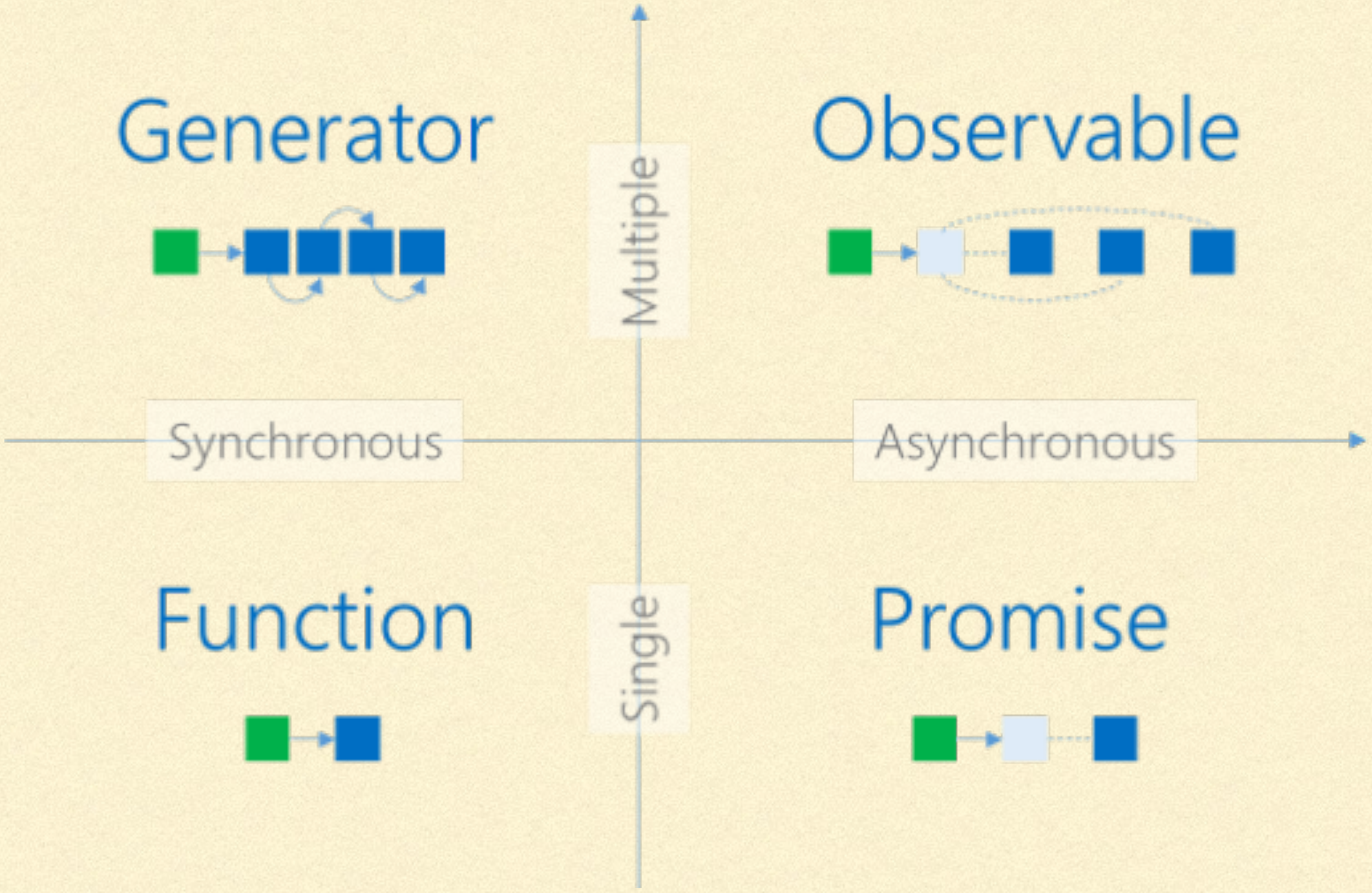
Use Cases

Suggest

```
let inputElement = document.querySelector('#item-name');
Rx.Observable.fromEvent(inputElement, 'keyup')
    .map(()=> inputElement.value) // get Value
    .filter((text) => text.length > 1) // filter empty text
    .debounce(500) //wait 500 ms
    .distinctUntilChanged()// record only change
    .subscribe((value)=> console.log(value));
```

Mouse Tracking

```
Rx.Observable.fromEvent(document, 'click')
    .buffer(Rx.Observable.interval(700))
    .where((click) => click.length > 0)
    .map((clickList)=> {
        return clickList.map((click)  => {
            return {
                x: click.offsetX,
                y: click.offsetY
            };
        });
    })
    .subscribe((value)=> console.log(value));
```


Generator
Observable
Multiple
Synchronous
Asynchronous
Function
Promise
Single

Links

- http://reactivex.io
- http://xgrommx.github.io/
- https://github.com/Reactive-Extensions/RxJS/
- https://www.youtube.com/watch?v=KOOT7BArVHQ