INFINUM

RxJS Workshop

LECTURE 1 – THE BASICS

INFINUM JS TEAM, 2020

01

WHAT IS RX?

ReactiveX

An API for asynchronous programming with observable streams

– HTTP://REACTIVEX.IO/

ReactiveX

ReactiveX is a combination of the best ideas from the Observer pattern, the Iterator pattern, and functional programming

– HTTP://REACTIVEX.IO/

REACTIVEX IS UBIQUITOUS

RxJava

RxJS

Rx.NET

RxSwift

RxPHP

RxKotlin

Rx.rb

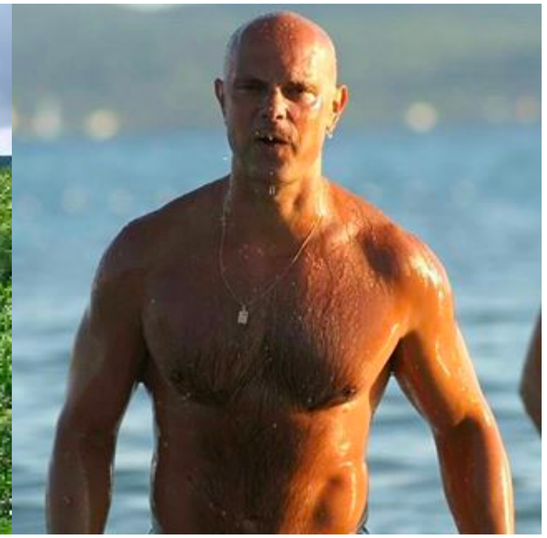
Others...

RxJS

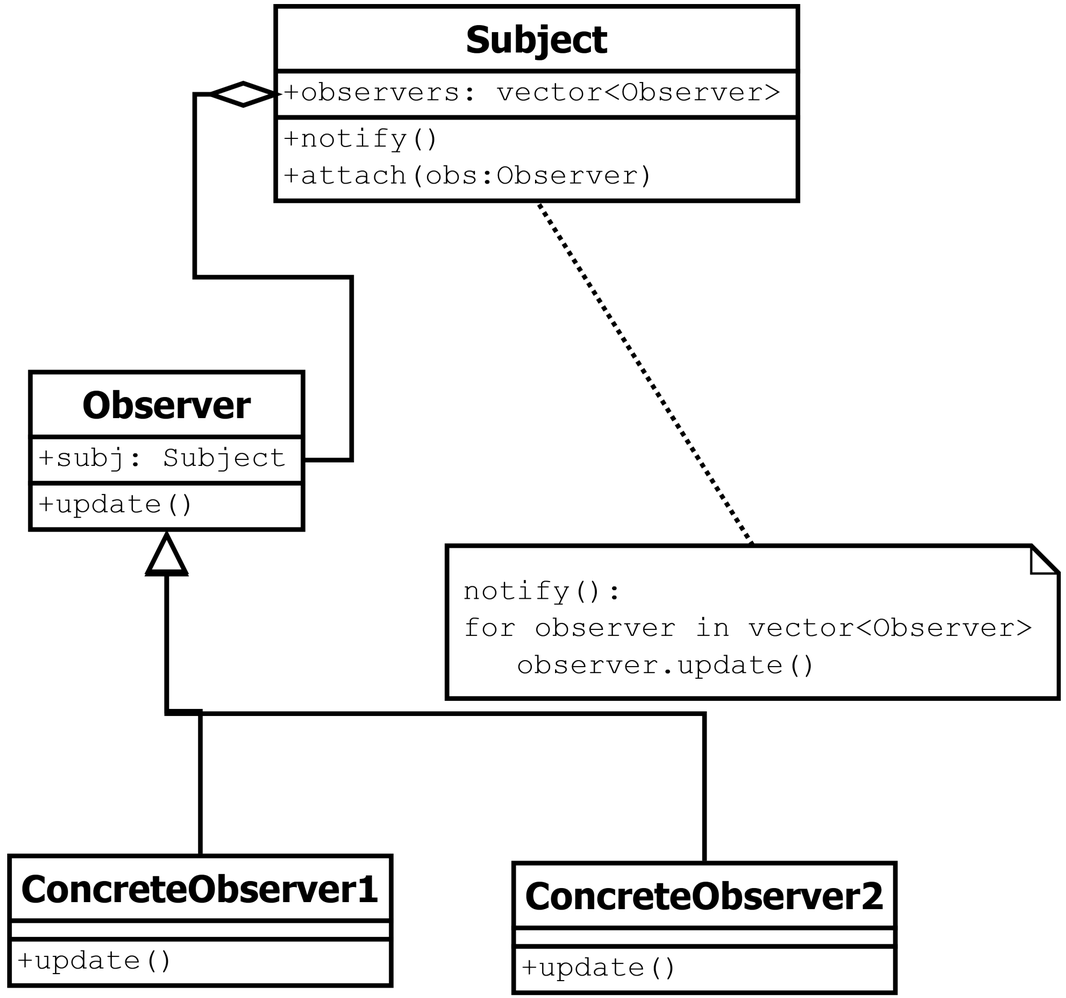
02

OBSERVABLE

DESIGN PATTERNS

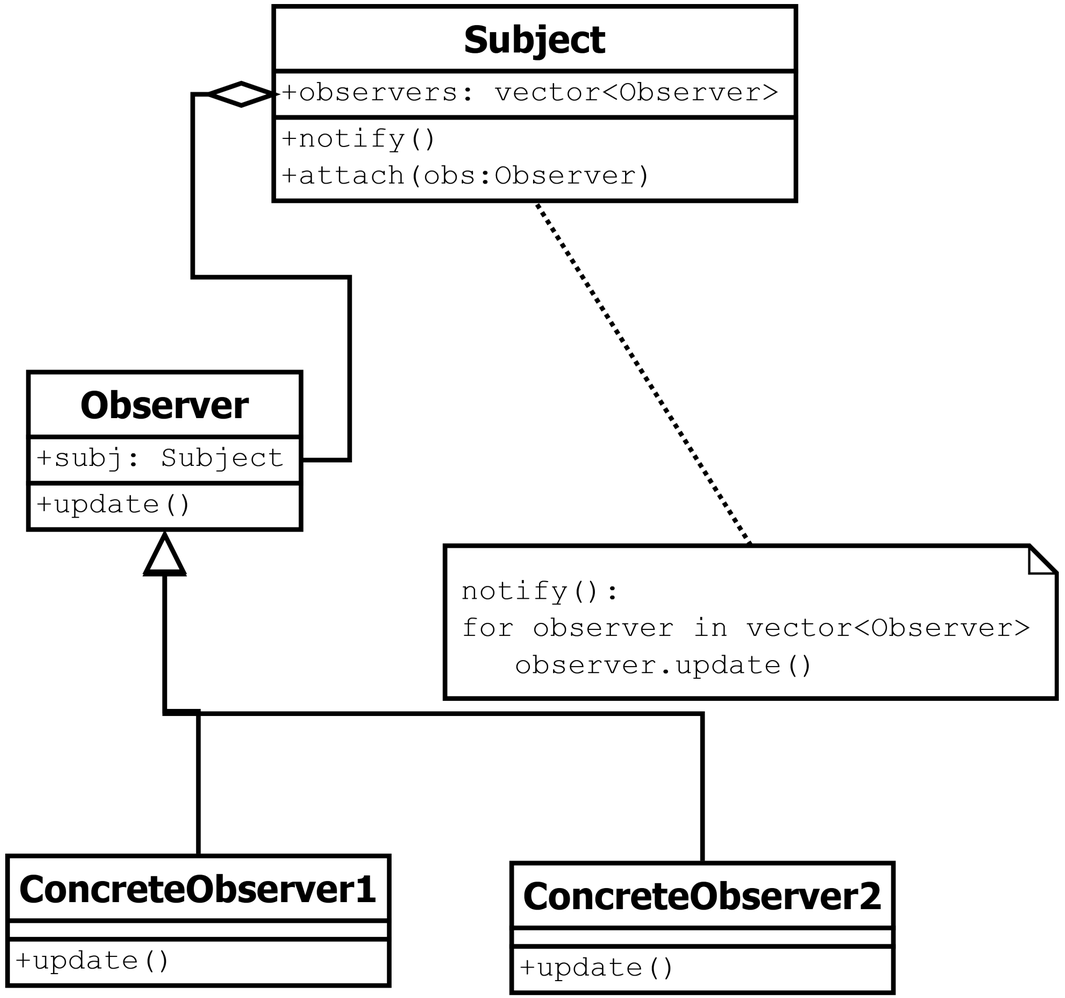


OBSERVER PATTERN



https://medium.com/datadriveninvestor/design-patterns-a-quick-guide-to-observer-pattern-d0622145d6c2

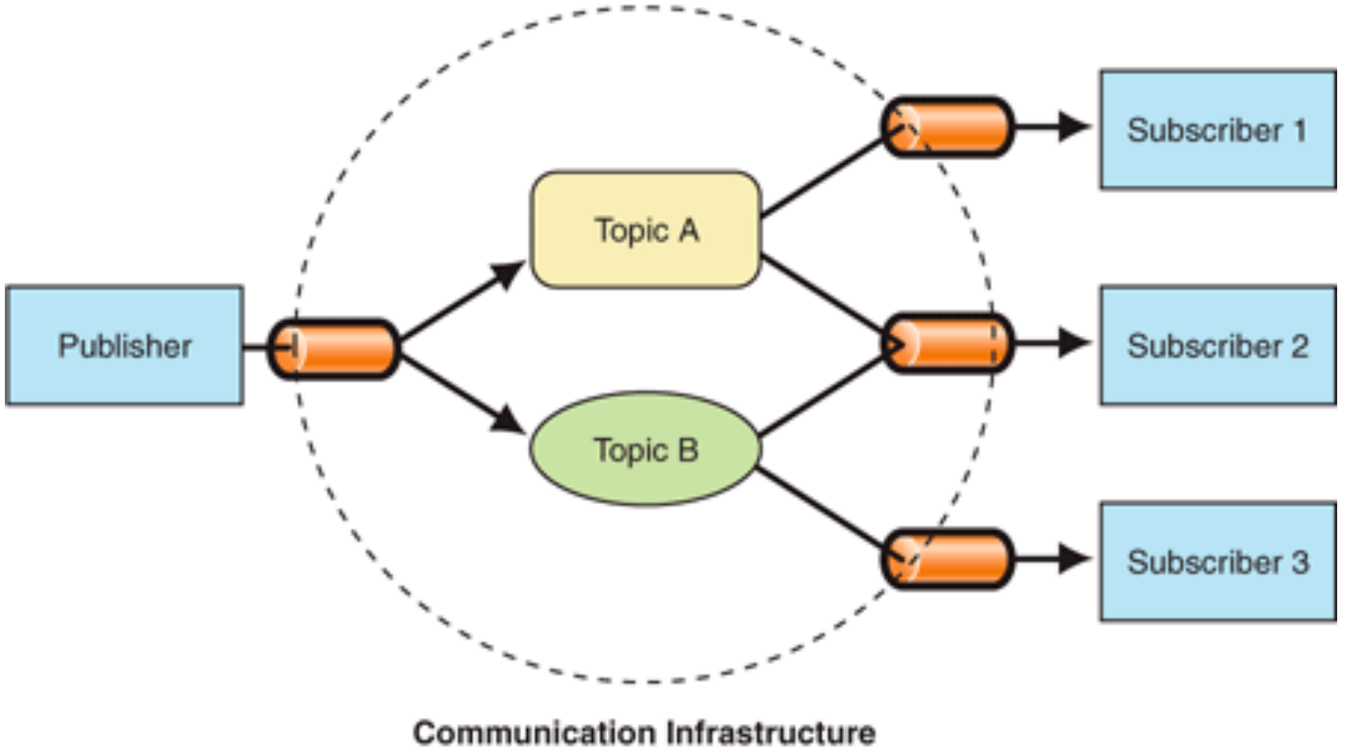
OBSERVER PATTERN



- **Subject:** It is considered as the keeper of information, of data or of business logic.
- **Register/Attach**: Observers register themselves to the subject because they want to be notified when there is a change.
- **Event:** Events act as a trigger in the subject such that all the observers are notified.
- **Notify:** Depending on the implementation, the subject may “push” information to the observers, or, the observers may “pull” if they need information from the subject.
- **Update:** Observers update their state independently from other observers however their state might change depending on the triggered event.

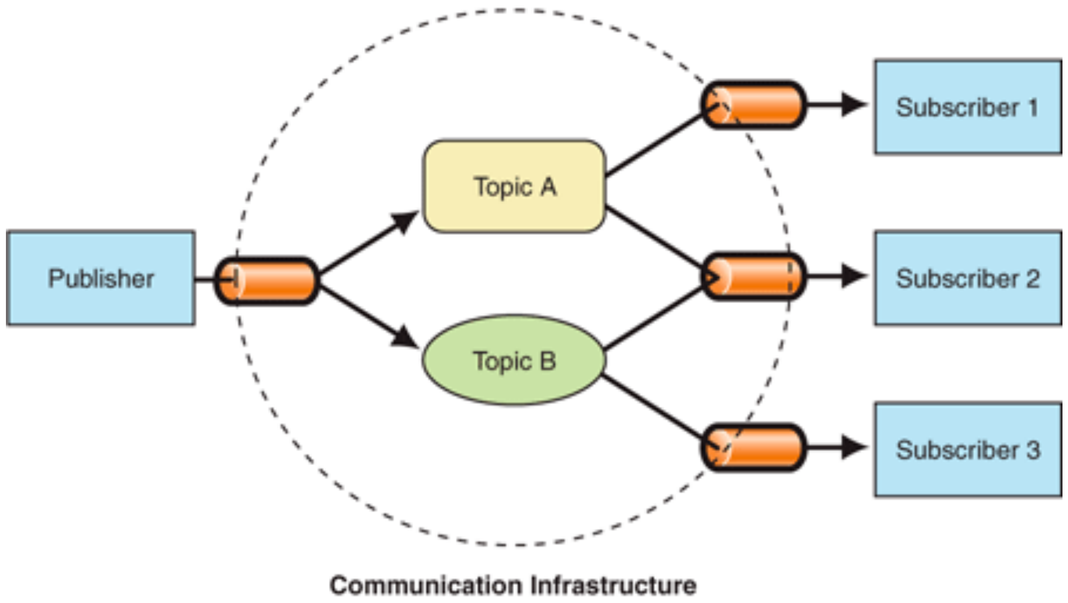
https://medium.com/datadriveninvestor/design-patterns-a-quick-guide-to-observer-pattern-d0622145d6c2

PUBLISHER-SUBSCRIBER PATTERN



https://hackernoon.com/observer-vs-pub-sub-pattern-50d3b27f838c

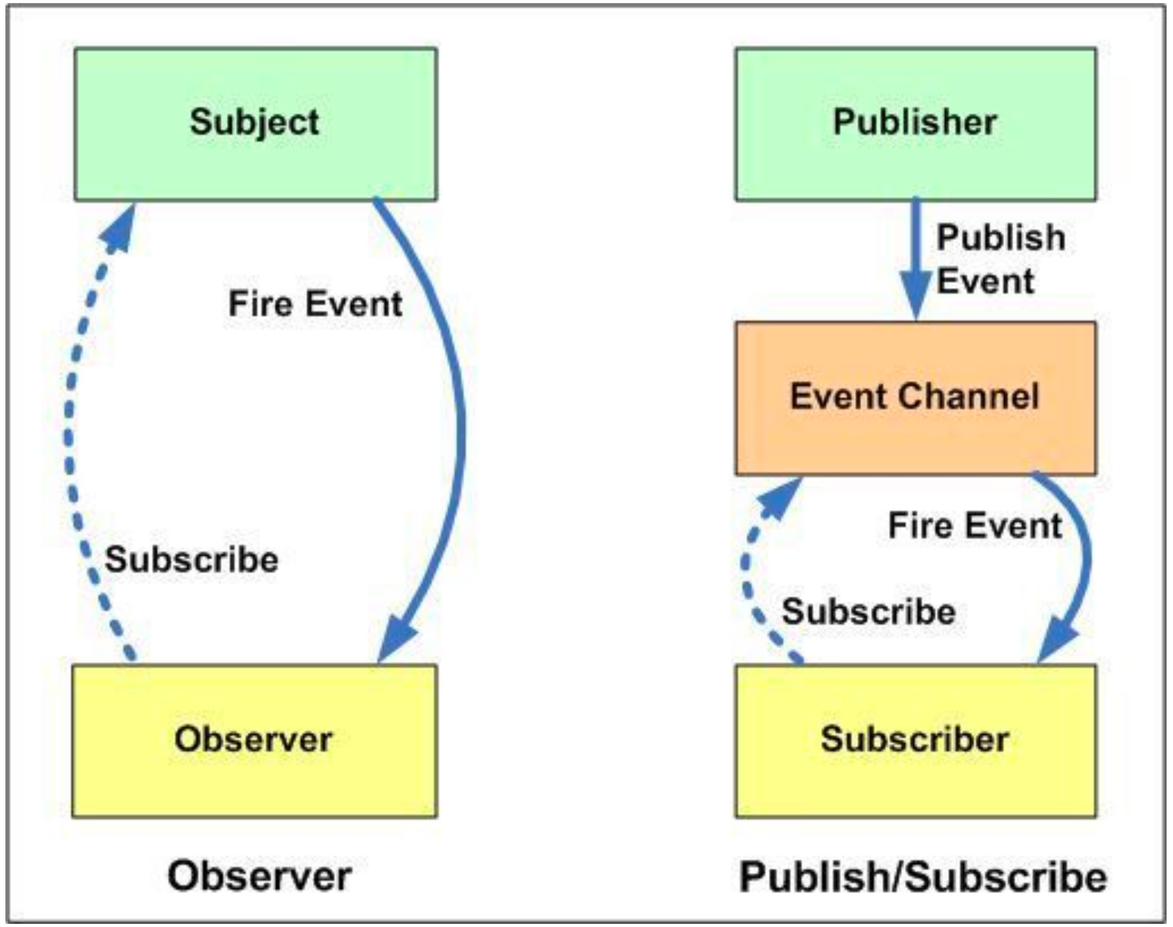
PUBLISHER-SUBSCRIBER PATTERN



Pub-sub is a pattern used to communicate messages between different system components without these components knowing anything about each other's identity.

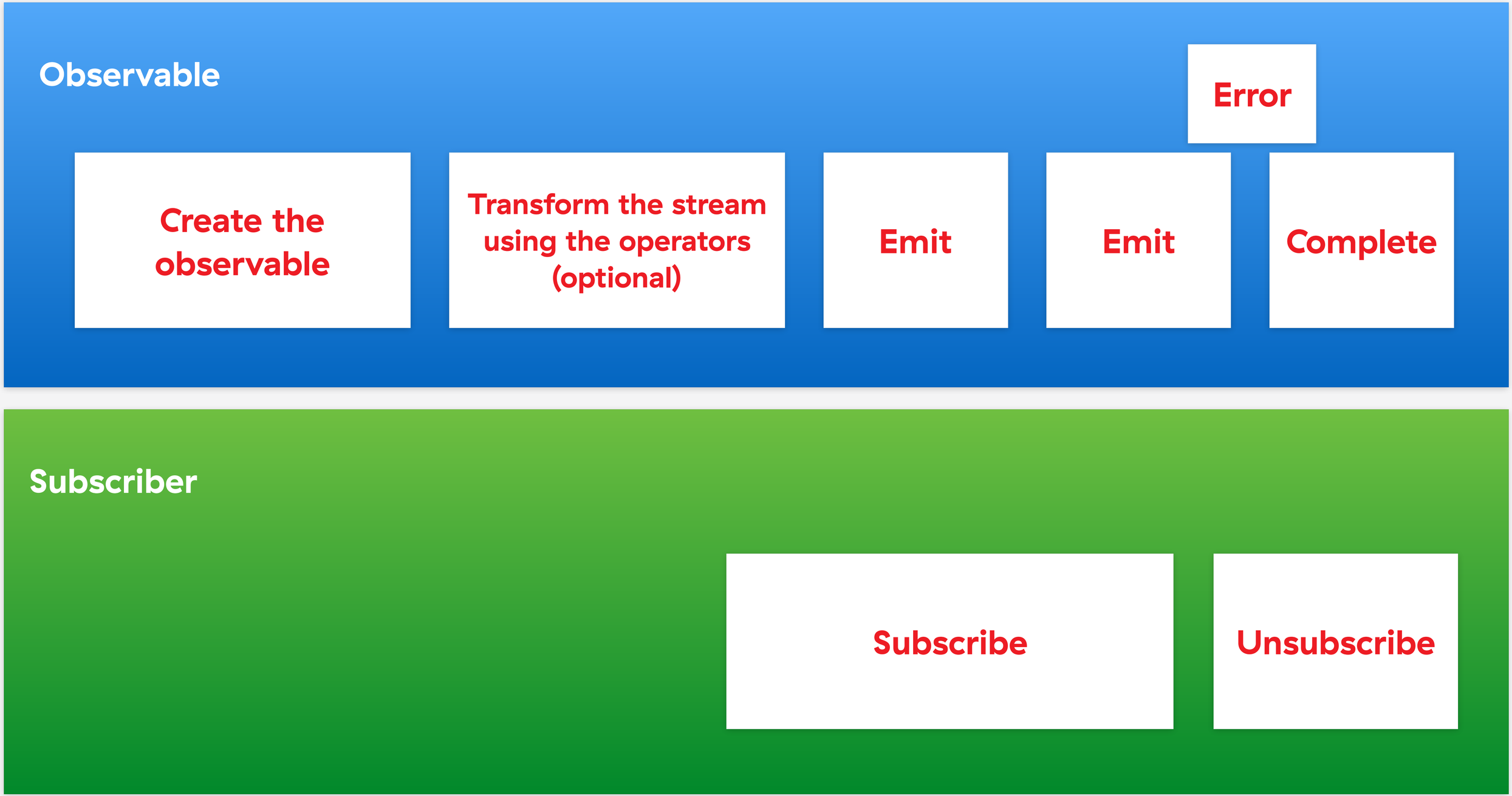
https://hackernoon.com/observer-vs-pub-sub-pattern-50d3b27f838c

OBSERVER VS PUB-SUB PATTERN



https://hackernoon.com/observer-vs-pub-sub-pattern-50d3b27f838c

03 OBSERVABLE LIFECYCLE


Observable
Error
Create the observable
Transform the stream using the operators (optional)
Emit
Emit
Complete
Subscriber
Subscribe
Unsubscribe

OBSERVABLE CREATION

```
1 import { fromEvent } from 'rxjs';
2
3 const button = document.getElementById('.some-button');
4
5 const buttonClicks$ = fromEvent(button, 'click');
```

SUBSCRIPTION

```
1 import { fromEvent } from 'rxjs';
2
3 const button = document.getElementById('.some-button');
4
5 const buttonClicks$ = fromEvent(button, 'click');
6
7 const subscription = buttonClicks$.subscribe((event) {
8   console.log(event);
9 });
```

SUBSCRIPTION OPTIONS

```
1 const subscription = buttonClicks$.subscribe({
2   next: event ⇒ console.log(event),
3   error: error ⇒ console.log(error),
4   complete: () ⇒ console.log('complete!')
5 });
```

SUBSCRIPTION OPTIONS

```
const subscription = buttonClicks$.subscribe({
  next: event ⇒ console.log(event),
  error: error ⇒ console.log(error),
  complete: () ⇒ console.log('complete!')
});
```

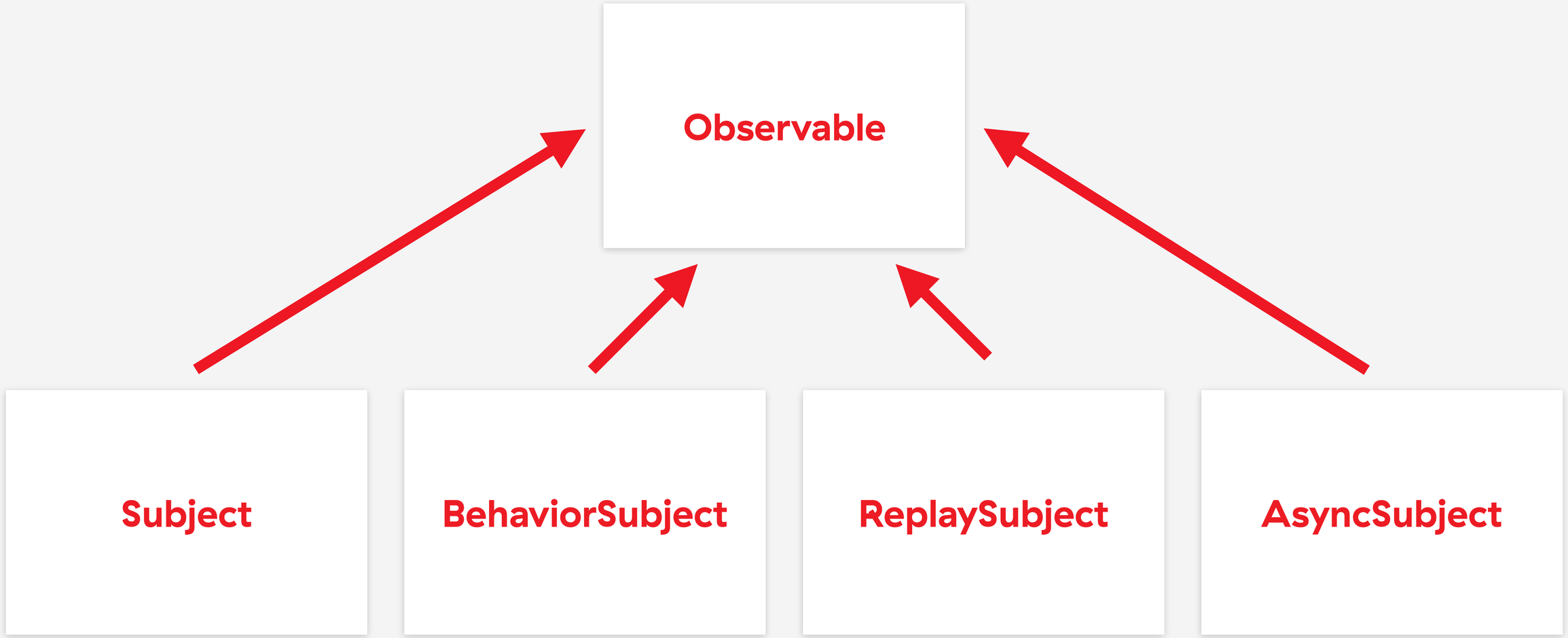
```
const subscription = buttonClicks$.subscribe(
  event ⇒ console.log(event),
  error ⇒ console.log(error),
  () ⇒ console.log('complete!')
);
```

UNSUBSCRIBE

```
1 const subscription = buttonClicks$.subscribe((event) ⇒ {
2   console.log(event);
3 });
4
5 // click #1 - logs
6
7 subscription.unsubscribe()
8
9 // click #2 - no logs
```

04

TYPES OF OBSERVABLES

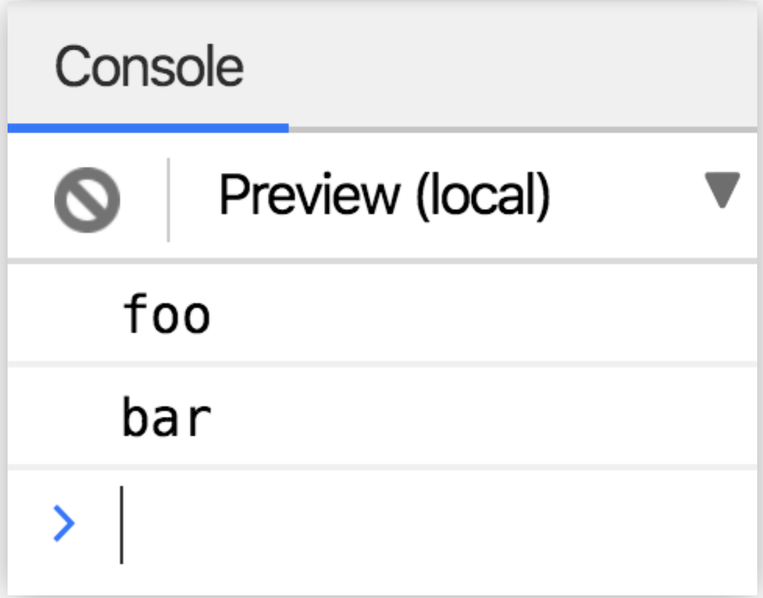

Observable
Subject
BehaviorSubject
ReplaySubject
AsyncSubject

Subject

```
1  import { Subject } from 'rxjs';
2
3  const source$ = new Subject();
4
5  source$.subscribe(value ⇒ console.log(value));
6
7  source$.next('foo');
8  source$.next('bar');
9
10 source$.unsubscribe();
11
12 source$.next('lost travolta meme');
```

Subject

```
1  import { Subject } from 'rxjs';
2
3  const source$ = new Subject();
4
5  source$.subscribe(value ⇒ console.log(value));
6
7  source$.next('foo');
8  source$.next('bar');
9
10 source$.unsubscribe();
11
12 source$.next('lost travolta meme');
```

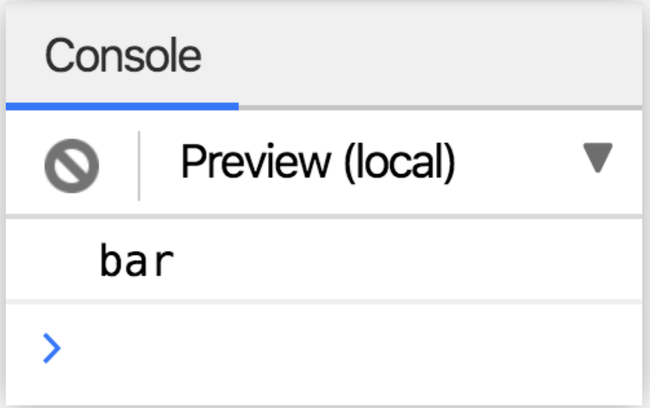


Subject

```
1 import { Subject } from 'rxjs';
2
3 const source$ = new Subject();
4
5 source$.next('foo');
6
7 source$.subscribe(value ⇒ console.log(value));
8
9 source$.next('bar');
```

Subject

```
1 import { Subject } from 'rxjs';
2
3 const source$ = new Subject();
4
5 source$.next('foo');
6
7 source$.subscribe(value ⇒ console.log(value));
8
9 source$.next('bar');
```

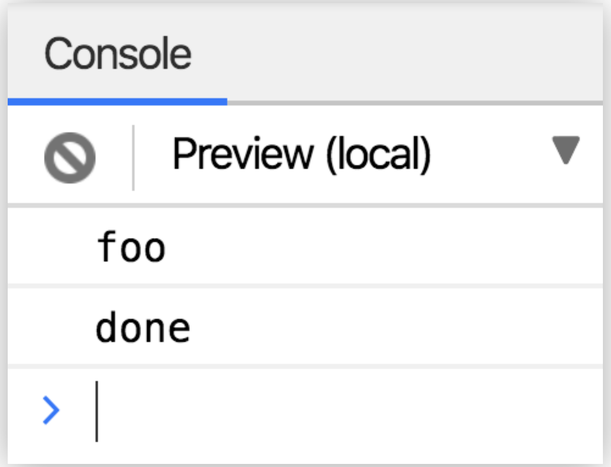


Subject

```
1 import { Subject } from 'rxjs';
2
3 const source$ = new Subject();
4
5 source$.subscribe({
6   next: console.log,
7   complete: () ⇒ console.log('done!')
8 });
9
10 source$.next('foo');
11 source$.complete();
12 source$.next('bar');
```

Subject

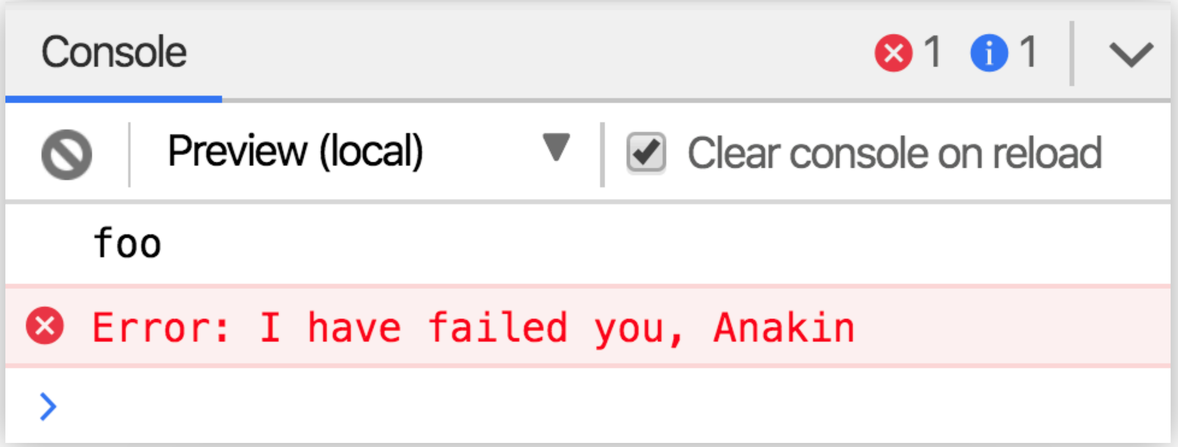
```
1 import { Subject } from 'rxjs';
2
3 const source$ = new Subject();
4
5 source$.subscribe({
6   next: console.log,
7   complete: () ⇒ console.log('done!')
8 });
9
10 source$.next('foo');
11 source$.complete();
12 source$.next('bar');
```



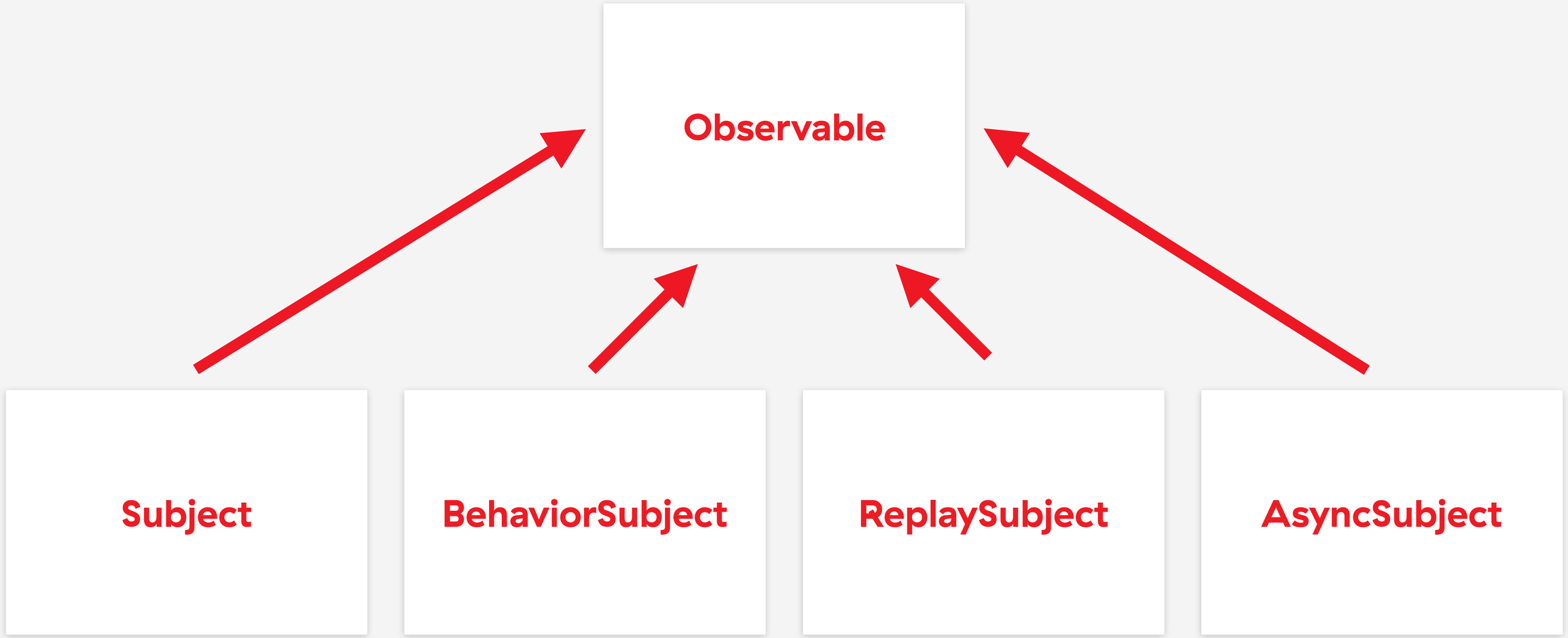
Subject

```
1 import { Subject } from 'rxjs';
2
3 const source$ = new Subject();
4
5 source$.subscribe({
6   next: console.log,
7   error: console.error,
8   complete: () ⇒ console.log('done!')
9 });
10
11 source$.next('foo');
12 source$.error(new Error('I have failed you, Anakin'));
13 source$.next('bar');
```

Subject




```
1 import { Subject } from 'rxjs';
2
3 const source$ = new Subject();
4
5 source$.subscribe({
6   next: console.log,
7   error: console.error,
8   complete: () ⇒ console.log('done!')
9 });
10
11 source$.next('foo');
12 source$.error(new Error('I have failed you, Anakin'));
13 source$.next('bar');
```

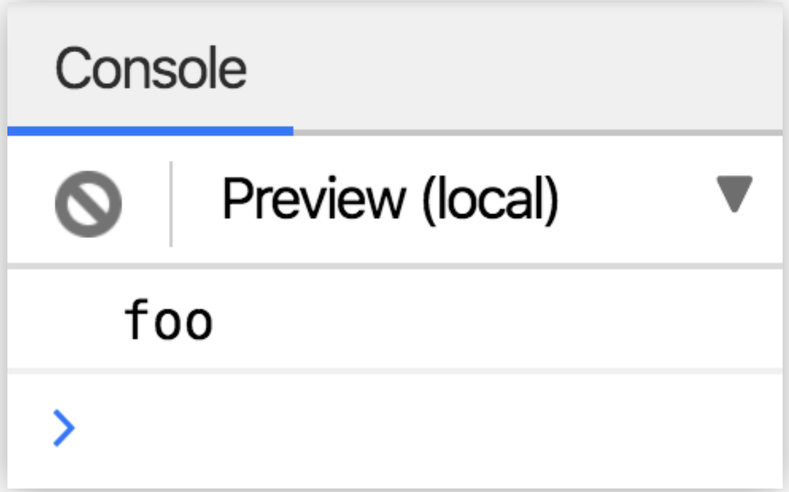

Observable
Subject
BehaviorSubject
ReplaySubject
AsyncSubject

BehaviorSubject

```
1 import { BehaviorSubject } from 'rxjs';
2
3 const source$ = new BehaviorSubject('foo');
4
5 source$.subscribe(console.log);
```

BehaviorSubject

```
1 import { BehaviorSubject } from 'rxjs';
2
3 const source$ = new BehaviorSubject('foo');
4
5 source$.subscribe(console.log);
```

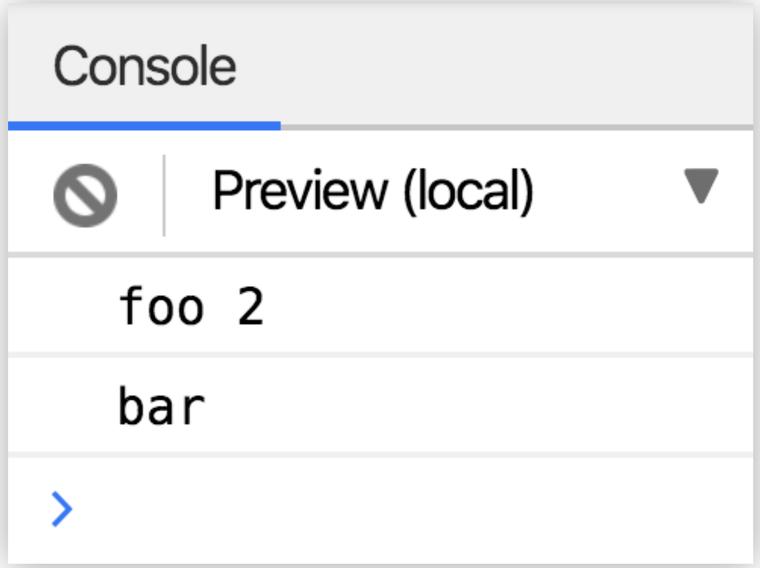


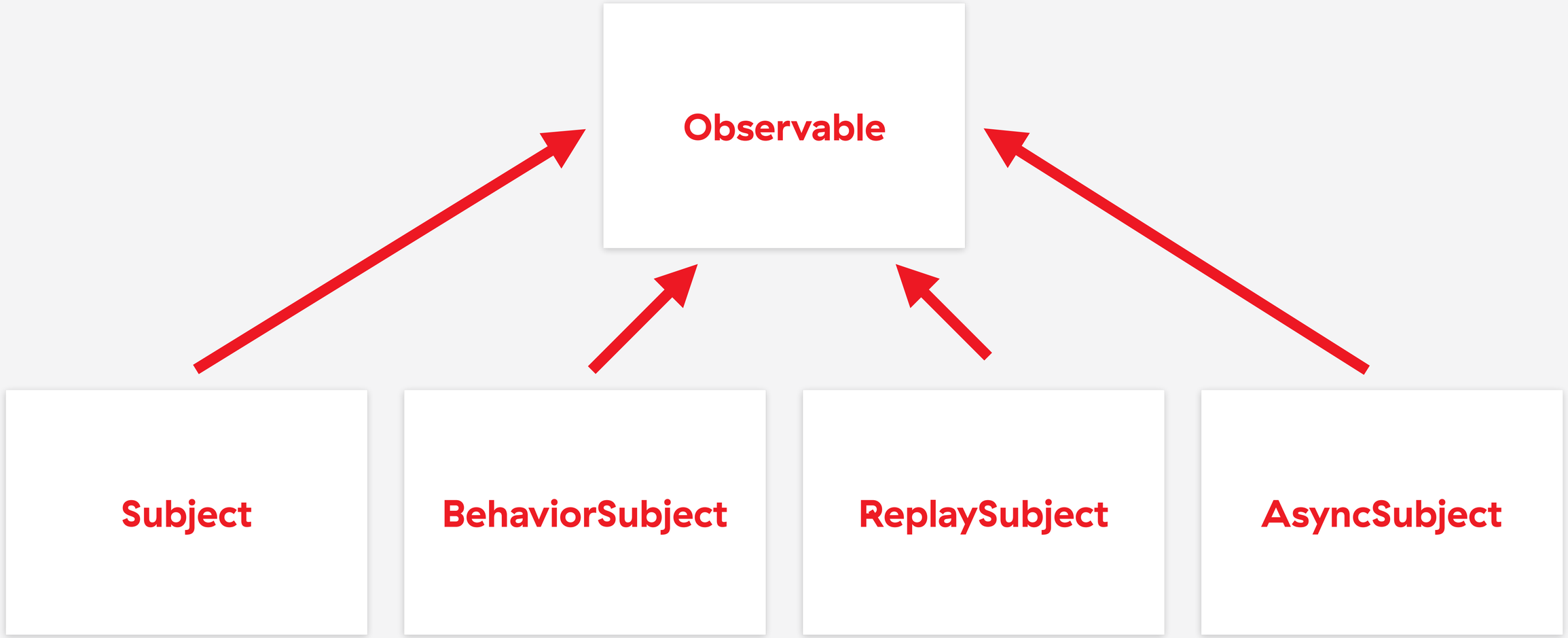
BehaviorSubject

```
1  import { BehaviorSubject } from 'rxjs';
2
3  const source$ = new BehaviorSubject('foo 0');
4
5  source$.next('foo 1');
6  source$.next('foo 2');
7
8  source$.subscribe(console.log);
9
10 source$.next('bar');
```

BehaviorSubject

```
1  import { BehaviorSubject } from 'rxjs';
2
3  const source$ = new BehaviorSubject('foo 0');
4
5  source$.next('foo 1');
6  source$.next('foo 2');
7
8  source$.subscribe(console.log);
9
10 source$.next('bar');
```



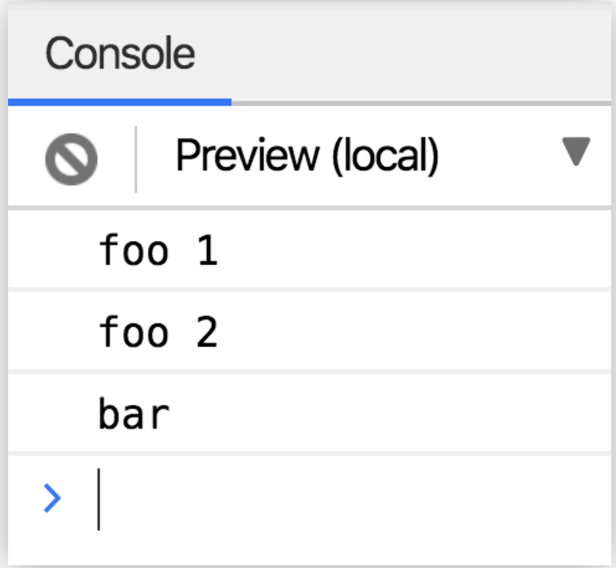

Observable
Subject
BehaviorSubject
ReplaySubject
AsyncSubject

ReplaySubject

```
1 import { ReplaySubject } from 'rxjs';
2
3 const source$ = new ReplaySubject();
4
5 source$.next('foo 1');
6 source$.next('foo 2');
7
8 source$.subscribe(console.log);
9
10 source$.next('bar');
```

ReplaySubject

```
1  import { ReplaySubject } from 'rxjs';
2
3  const source$ = new ReplaySubject();
4
5  source$.next('foo 1');
6  source$.next('foo 2');
7
8  source$.subscribe(console.log);
9
10 source$.next('bar');
```

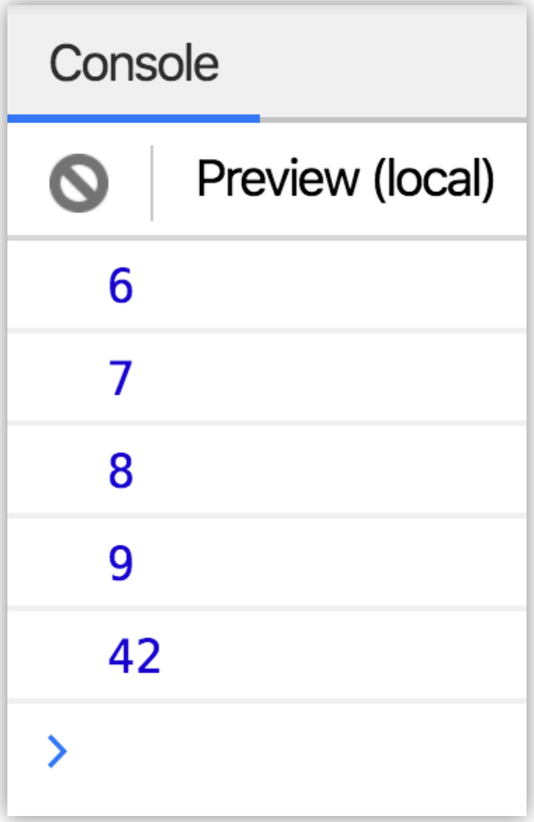


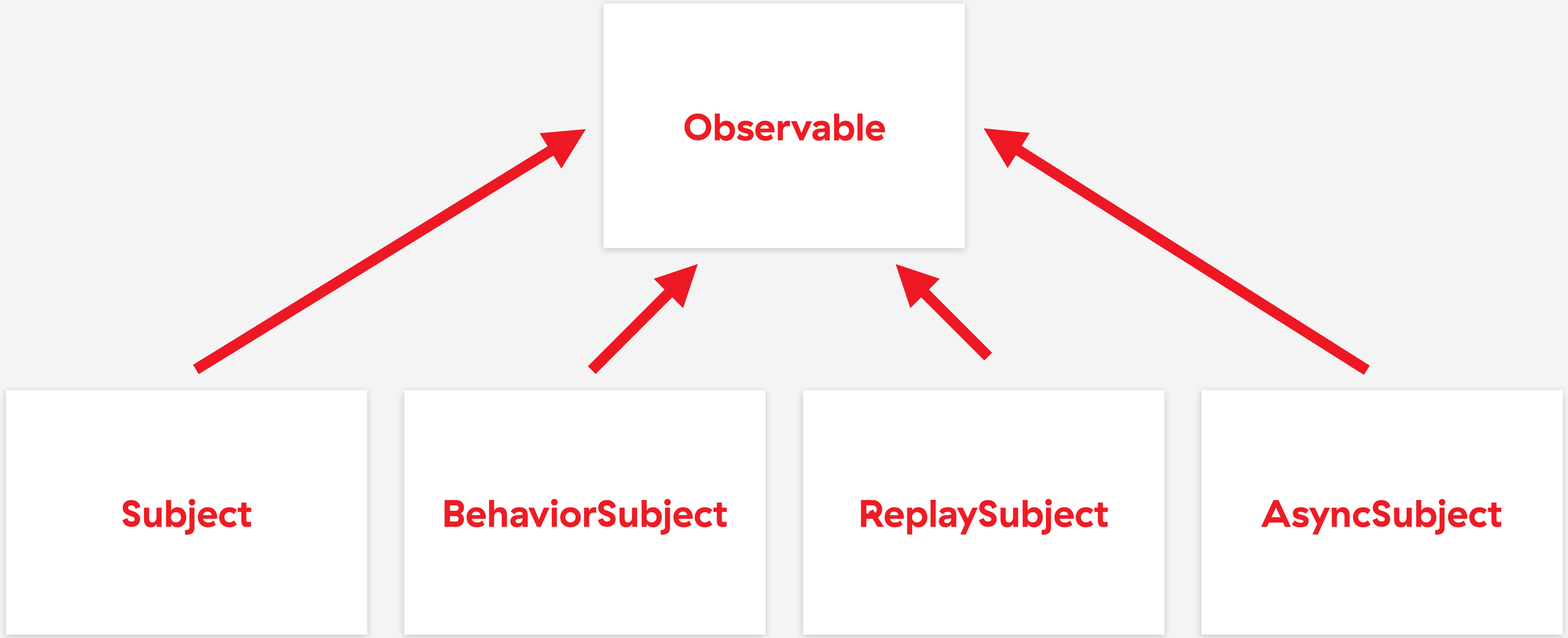
ReplaySubject

```
1 import { ReplaySubject } from 'rxjs';
2
3 const source$ = new ReplaySubject(4);
4
5 for (let i = 0; i < 10; i++) {
6   source$.next(i);
7 }
8
9 source$.subscribe(console.log);
10
11 source$.next(42);
```

ReplaySubject

```
1  import { ReplaySubject } from 'rxjs';
2
3  const source$ = new ReplaySubject(4);
4
5  for (let i = 0; i < 10; i++) {
6    source$.next(i);
7  }
8
9  source$.subscribe(console.log);
10
11 source$.next(42);
```



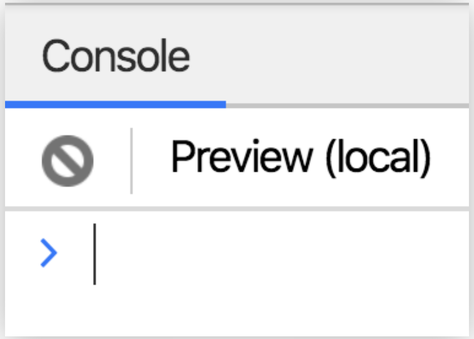

Observable
Subject
BehaviorSubject
ReplaySubject
AsyncSubject

AsyncSubject

```
1 import { AsyncSubject } from 'rxjs';
2
3 const source$ = new AsyncSubject();
4
5 source$.subscribe(console.log);
6
7 source$.next('foo');
8 source$.next('bar');
```

```
1 import { AsyncSubject } from 'rxjs';
2
3 const source$ = new AsyncSubject();
4
5 source$.next('foo');
6 source$.next('bar');
7
8 source$.subscribe(console.log);
```

AsyncSubject




```
1 import { AsyncSubject } from 'rxjs';
2
3 const source$ = new AsyncSubject();
4
5 source$.subscribe(console.log);
6
7 source$.next('foo');
8 source$.next('bar');
```

```
1 import { AsyncSubject } from 'rxjs';
2
3 const source$ = new AsyncSubject();
4
5 source$.next('foo');
6 source$.next('bar');
7
8 source$.subscribe(console.log);
```

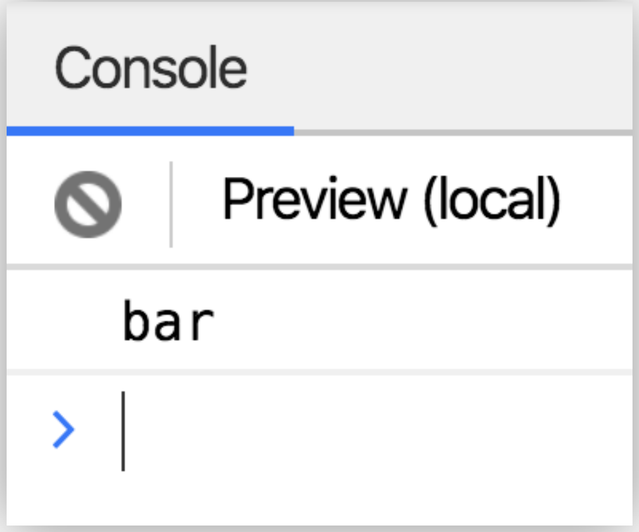
AsyncSubject

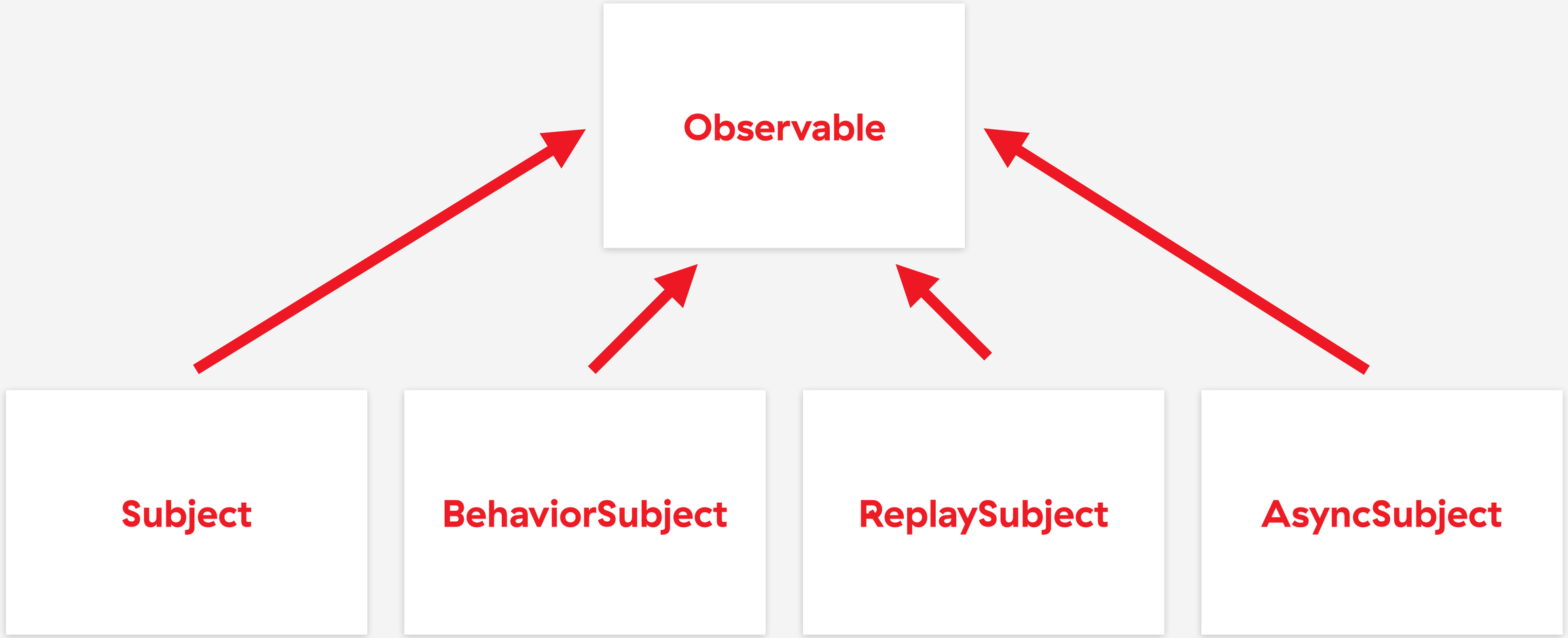
```
1 import { AsyncSubject } from 'rxjs';
2
3 const source$ = new AsyncSubject();
4
5 source$.next('foo');
6 source$.next('bar');
7
8 source$.subscribe(console.log);
```

```
1 import { AsyncSubject } from 'rxjs';
2
3 const source$ = new AsyncSubject();
4
5 source$.next('foo');
6 source$.next('bar');
7
8 source$.subscribe(console.log);
9 source$.complete();
```

AsyncSubject

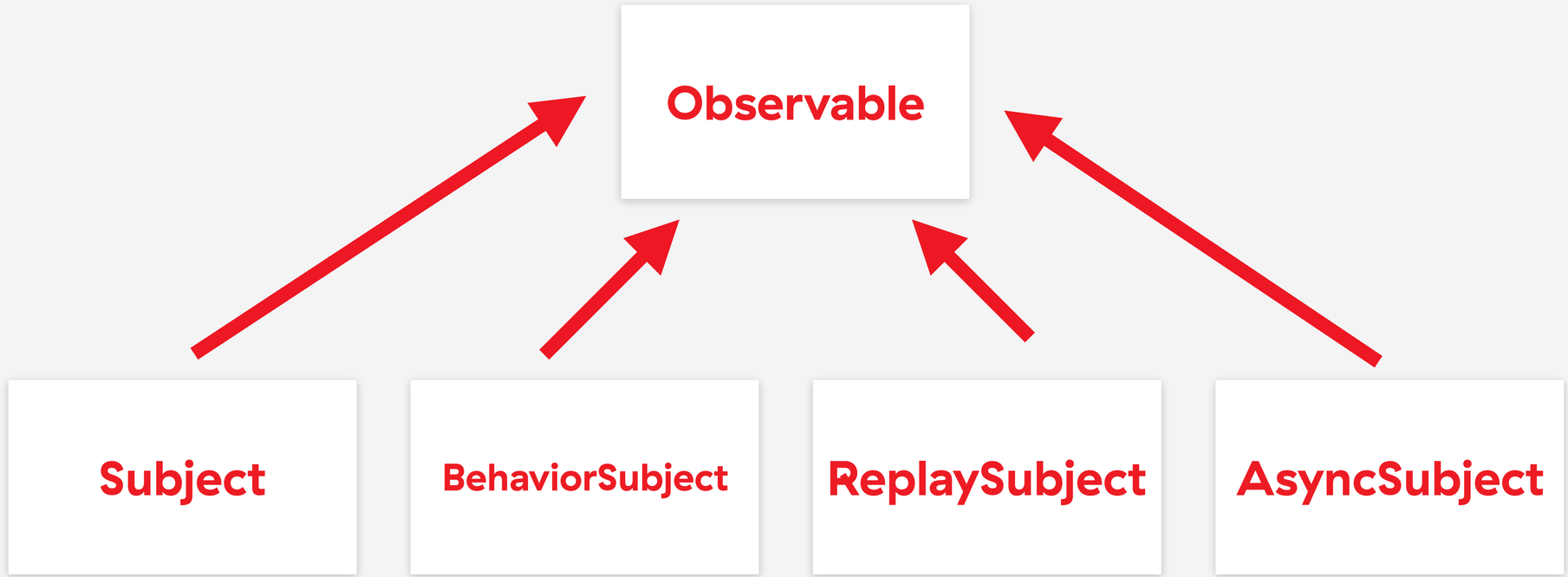
```
1 import { AsyncSubject } from 'rxjs';
2
3 const source$ = new AsyncSubject();
4
5 source$.next('foo');
6 source$.next('bar');
7
8 source$.subscribe(console.log);
9 source$.complete();
```




Observable
Subject
BehaviorSubject
ReplaySubject
AsyncSubject

RESOURCES

- Introduction
- RxJS Primer
- Observable



05

HOT & COLD OBSERVABLES

COLD

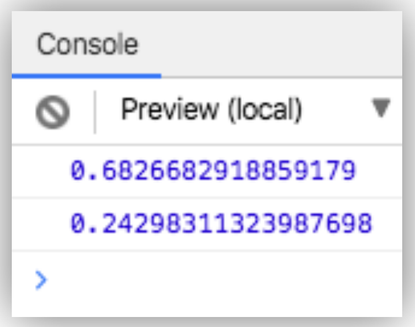
- Data is produced inside the observable
- Subscriptions start the data production

HOT

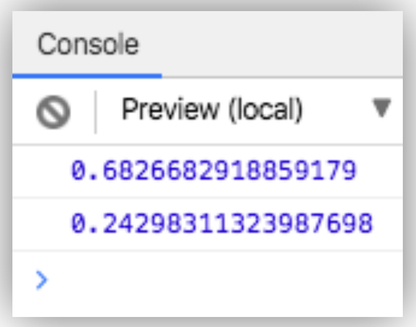
- Data is produced outside the observable
- Observable creation starts data production

```
1 import { Observable } from 'rxjs';
2
3 function createRng() {
4   return new Observable((observer) ⇒ {
5     const randomNumber = Math.random();
6     observer.next(randomNumber);
7   });
8 }
9
10 const rng$ = createRng();
11
12 rng$.subscribe(console.log);
13 rng$.subscribe(console.log);
```

```
1  import { Observable } from 'rxjs';
2
3  function createRng() {
4    return new Observable((observer) ⇒ {
5      const randomNumber = Math.random();
6      observer.next(randomNumber);
7    });
8  }
9
10 const rng$ = createRng();
11
12 rng$.subscribe(console.log);
13 rng$.subscribe(console.log);
```

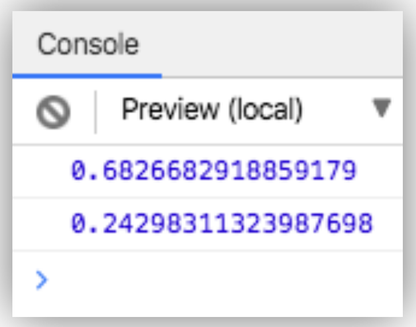


```
1 import { Observable } from 'rxjs';
2
3 function createRng() {
4   return new Observable((observer) ⇒ {
5     const randomNumber = Math.random();
6     observer.next(randomNumber);
7   });
8 }
9
10 const rng$ = createRng();
11
12 rng$.subscribe(console.log);
13 rng$.subscribe(console.log);
```



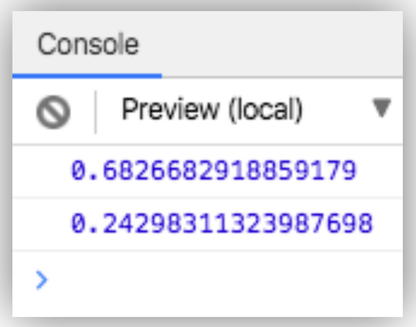

```
1 import { Observable } from 'rxjs';
2
3 function createRng() {
4   const randomNumber = Math.random();
5
6   return new Observable((observer) ⇒ {
7     observer.next(randomNumber);
8   });
9 }
10
11 const rng$ = createRng();
12
13 rng$.subscribe(console.log);
14 rng$.subscribe(console.log);
```

```
1  import { Observable } from 'rxjs';
2
3  function createRng() {
4    return new Observable((observer) ⇒ {
5      const randomNumber = Math.random();
6      observer.next(randomNumber);
7    });
8  }
9
10 const rng$ = createRng();
11
12 rng$.subscribe(console.log);
13 rng$.subscribe(console.log);
```

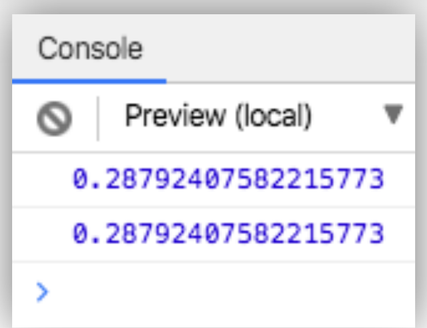



```
1  import { Observable } from 'rxjs';
2
3  function createRng() {
4    const randomNumber = Math.random();
5
6    return new Observable((observer) ⇒ {
7      observer.next(randomNumber);
8    });
9  }
10
11 const rng$ = createRng();
12
13 rng$.subscribe(console.log);
14 rng$.subscribe(console.log);
```

```
1  import { Observable } from 'rxjs';
2
3  function createRng() {
4    return new Observable((observer) ⇒ {
5      const randomNumber = Math.random();
6      observer.next(randomNumber);
7    });
8  }
9
10 const rng$ = createRng();
11
12 rng$.subscribe(console.log);
13 rng$.subscribe(console.log);
```




```
1  import { Observable } from 'rxjs';
2
3  function createRng() {
4    const randomNumber = Math.random();
5
6    return new Observable((observer) ⇒ {
7      observer.next(randomNumber);
8    });
9  }
10
11 const rng$ = createRng();
12
13 rng$.subscribe(console.log);
14 rng$.subscribe(console.log);
```



```
import { Observable } from 'rxjs';

function createRng() {
  return new Observable((observer) => {
    const randomNumber = Math.random();
    observer.next(randomNumber);
  });
}

const rng$ = createRng();

rng$.subscribe(console.log);
rng$.subscribe(console.log);
```

```
import { Observable } from 'rxjs';

function createRng() {
  const randomNumber = Math.random();

  return new Observable((observer) => {
    observer.next(randomNumber);
  });
}

const rng$ = createRng();

rng$.subscribe(console.log);
rng$.subscribe(console.log);
```

Console

Preview (local)

0.28792407582215773

0.28792407582215773

>

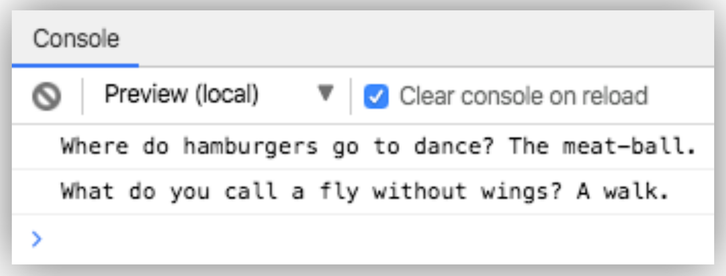
```
import { Observable } from 'rxjs';

function createRng() {
  return new Observable((observer) => {
    const randomNumber = Math.random();
    observer.next(randomNumber);
  });
}

const rng$ = createRng();

rng$.subscribe(console.log);
rng$.subscribe(console.log);
```

```
import { Observable } from 'rxjs';

function createRng() {
  const randomNumber = Math.random();

  return new Observable((observer) => {
    observer.next(randomNumber);
  });
}

const rng$ = createRng();

rng$.subscribe(console.log);
rng$.subscribe(console.log);
```

COLD OBSERVABLE EXAMPLE

```
1 import { ajax } from 'rxjs/ajax';
2 import { map } from 'rxjs/operators';
3
4 const joke$ = ajax({
5   url: 'https://icanhazdadjoke.com',
6   headers: { accept: 'application/json' },
7 }).pipe(
8   map(({ response }) => response.joke),
9 );
10
11 joke$.subscribe(console.log);
12 joke$.subscribe(console.log);
```

COLD OBSERVABLE EXAMPLE




```
1 import { ajax } from 'rxjs/ajax';
2 import { map } from 'rxjs/operators';
3
4 const joke$ = ajax({
5   url: 'https://icanhazdadjoke.com',
6   headers: { accept: 'application/json' },
7 }).pipe(
8   map(({ response }) => response.joke),
9 );
10
11 joke$.subscribe(console.log);
12 joke$.subscribe(console.log);
```

HOT OBSERVABLE EXAMPLE

```
1  import { webSocket } from "rxjs/webSocket";
2
3  const socket$ = webSocket("ws://localhost:8081");
4
5  socket$.subscribe(
6    msg => console.log('message received: ' + msg),
7    err => console.log(err),
8    () => console.log('complete')
9  );
10
11 socket$.subscribe(
12   msg => console.log('message received: ' + msg),
13   err => console.log(err),
14   () => console.log('complete')
15 );
```

06

OBSERVABLES VS PROMISES VS MOBX

RETURNING DATA IN JAVASCRIPT

	Single	Multiple
Synchronous	Primitive	Array, Object, …
Asynchronous	Promise	**?**

https://github.com/infinum/js-talks/blob/master/talks-10/rxjs_luka_skukan.pdf

RETURNING DATA IN JAVASCRIPT

	Single	Multiple
Synchronous	Primitive	Array, Object, …
Asynchronous	Promise	**RxJS**

https://github.com/infinum/js-talks/blob/master/talks-10/rxjs_luka_skukan.pdf

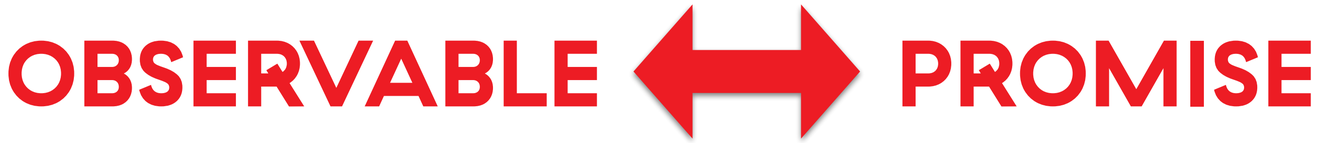
COMPARISON BETWEEN PROMISES AND RXJS

```
1 new Promise((resolve) ⇒ {
2   setTimeout(() ⇒ {
3     resolve(42)
4   }, 500);
5 }).then(console.log);
```

COMPARISON BETWEEN PROMISES AND RXJS

```
1 new Promise((resolve) ⇒ {
2   setTimeout(() ⇒ {
3     resolve(42)
4   }, 500);
5 }).then(console.log);
```

```
1 const source$ = new Subject();
2
3 setTimeout(() ⇒ {
4   source$.next(42);
5   source$.complete();
6 }, 500);
7
8 source$.subscribe(console.log)
```


OBSERVABLE
PROMISE

CONVERT OBSERVABLE TO A PROMISE

```
new Promise((resolve) => {
  setTimeout(() => {
    resolve(42)
  }, 500);
}).then(console.log);
```

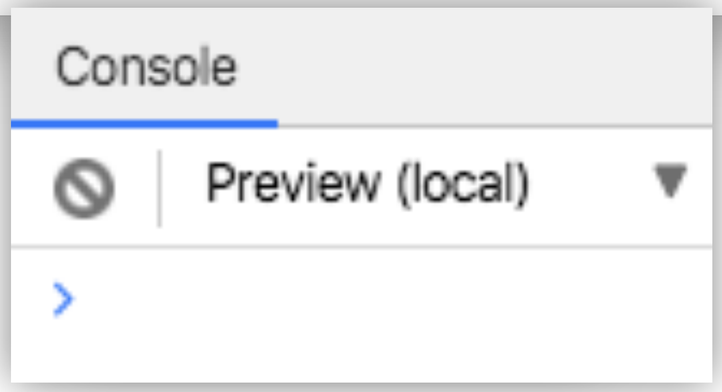
```
const source$ = new Subject();

setTimeout(() => {
  source$.next(42);
  source$.complete();
}, 500);

source$.subscribe(console.log)
```

CONVERT OBSERVABLE TO A PROMISE

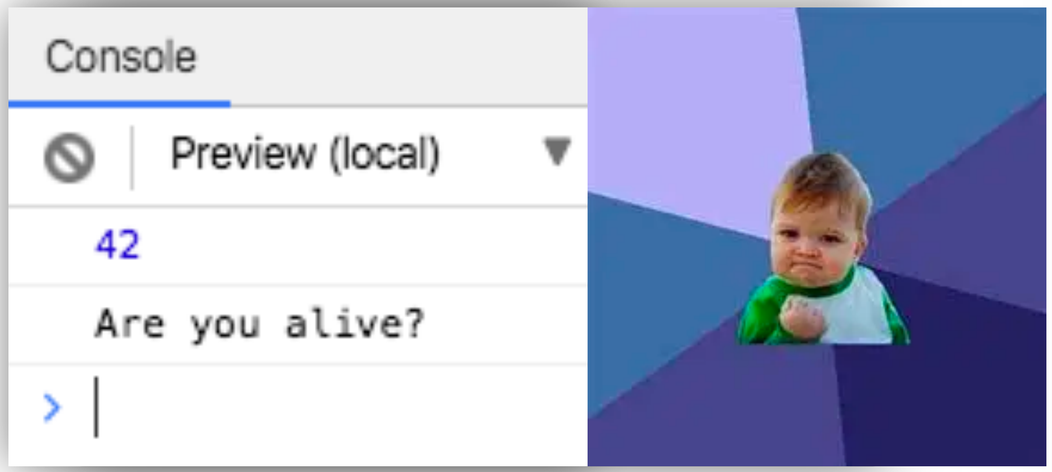
```
1 const p = new Promise((resolve) ⇒ {
2   setTimeout(() ⇒ {
3     resolve(42);
4   }, 500);
5 });
6
7 const result = await p;
8 console.log(result);
```

```
1 new Promise((resolve) ⇒ {
2   setTimeout(() ⇒ {
3     resolve(42)
4   }, 500);
5 }).then(console.log);
```

```
1 const source$ = new Subject();
2
3 setTimeout(() ⇒ {
4   source$.next(42);
5 }, 500);
6
7 const result = await source$.toPromise();
8 console.log(result);
9 console.log('Are you alive?');
```



CONVERT OBSERVABLE TO A PROMISE



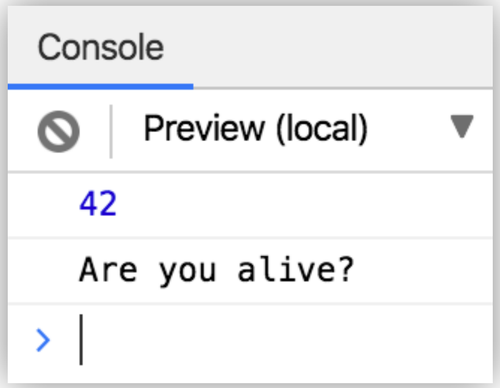

```
1 const source$ = new Subject();
2
3 setTimeout(() ⇒ {
4   source$.next(42);
5   source$.complete();
6 }, 500);
7
8 const result = await source$.toPromise();
9 console.log(result);
10 console.log('Are you alive?');
```

HOW MANY VALUES?

```
1 const source$ = new Subject();
2
3 setTimeout(() => {
4   source$.next(40);
5   source$.next(41);
6   source$.next(42);
7   source$.complete();
8 }, 500);
9
10 const result = await source$.toPromise();
11 console.log(result);
12 console.log('Are you alive?');
```

HOW MANY VALUES?

```
1  const source$ = new Subject();
2
3  setTimeout(() => {
4    source$.next(40);
5    source$.next(41);
6    source$.next(42);
7    source$.complete();
8  }, 500);
9
10 const result = await source$.toPromise();
11 console.log(result);
12 console.log('Are you alive?');
```



VS. MOBX

VS. MOBX

- RxJS has Observables
- MobX has Observables
- Not same at all
- RxJS observables
  - Explicit subscriptions
  - Have pipes
- MobX observable
  - A value that can be observed
  - Subscriptions are automated (observer / autorun)
  - MobX decides which observables are relevant for the observer

07

¿PREGUNTAS?

08

HOMEWORK

GENERAL

- Fork, pull and npm install **https://github.com/infinum/JS-RxWorkshop**
- Write the implementation in placeholder files in **src/lecture-X/task-Y/*.ts**
- Run tests to check if your implementation passes the tests
  - **npm test** runs all tests
  - **npm run test:lecture-1** runs all Lecture 1 tests
  - **npm run test:lecture-1:task-X** runs Lecture 1 – Task X tests
  - Write additional tests if you see the need or just want to further verify your solution (we might have missed some edge cases)
- You can also start the devserver (**npm start**) and write whatever you want in **src/index.ts** and **src/index.html**

TASK #1 – DOM EVENTS

- Wrap **DOM events on an element** into **Observable**
- Implement **fromEvent** function:
  - Receives parameters **element: HTMLElement** and **eventType: string**
  - Returns **Observable<Event>** which **emits** the event object when the event triggers

Notes:

- RxJS has a built-in function **fromEvent** which does basically the same thing
  - Be careful not to import it from RxJS instead of your own implementation
  - **DO NOT USE THIS** for homework solution (it would be a one-liner), but you can use it to test things in the devserver and see how it works (without looking at the source of RxJS implementation)
- Think about whether we want/need this observable to complete at some point?
- Will you implement this as a cold or as a hot observable? Try both ways, it will help you to better understand the the difference between hot and cold observable if you try both ways. Then think – do you want it to be cold or hot?
- What should happen when you unsubscribe?

TASK #2 – SETTIMEOUT

- Wrap **setTimeout** into **Observable**
- Implement **timeout** function:
  - Receives **time: number** in milliseconds as its only parameter
  - Returns **Observable<void>** which **emits** and **completes** after the specified timeout

Notes:

- Think about why we would want the observable to complete immediately after emission
- Will you implement this as a cold or as a hot observable? Try both ways, it will help you to better understand the the difference between hot and cold observable if you try both ways. Then think – do you want it to be cold or hot?
- What should happen when you unsubscribe?

TASK #3 – SETINTERVAL

- Wrap **setInterval** into **Observable**
- Implement **interval** function:
  - Receives **time: number** in milliseconds as its only parameter
  - Returns **Observable<number>** which periodically **emits** incrementing integers (starting at 0)

Notes:

- Think about why we do not need to complete the observable in this case
- Will you implement this as a cold or as a hot observable? Try both ways, it will help you to better understand the the difference between hot and cold observable if you try both ways. Then think – do you want it to be cold or hot?
- What should happen when you unsubscribe?

TASK #4 – REQUEST

- Wrap **fetch** or **XHR** (your choice) into **Observable**
- Implement **req<TBody>** function:
  - Receives **options: IRequestOptions<TBody>** as its only parameter
  - Returns **Observable<IReqResponse>** which **emits** the value and **completes**

Notes:

- Function type/interface is the same for both fetch and XHR, only the implementation is different
- Request should be aborted when unsubscribing
  - If using XHR: **link**
  - If using fetch: **link**
- Tests also check if you implementation works with **await req(...).toPromise()**
- Same question as in other tasks – will you implement this as a cold or as a hot observable?
- Response will always be a string, even if it is a stringified JSON (for simplicity)