Javascript – reactive programming

Veerle Ongenae

Doel

UNIVERSITEIT GENT

- Basisconcepten Reactive Programming kennen en kunnen toepassen met behulp van RxJS
  - Observable
  - Observer
  - Subscription
  - Operators
  - Hot vs Cold Observables
- Verschillende programmeerparadigma's kennen en kunnen gebruiken.

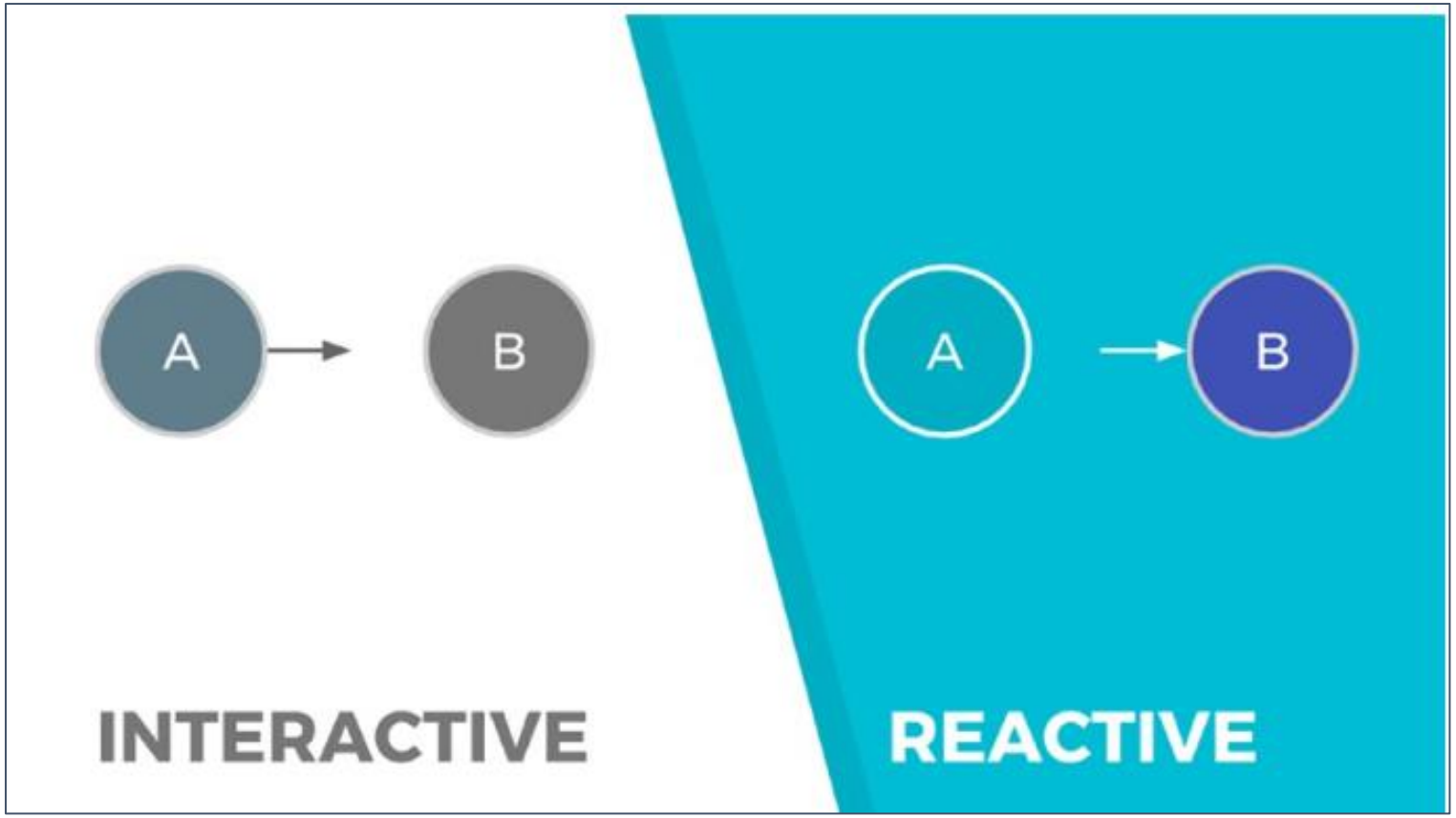
Overzicht

UNIVERSITEIT GENT

- Wat is reactive programming?

Wat is “reactive programming”?

Interactive versus reactive programming



bron: Front-End Reactive Architectures, Luca Mezzalira

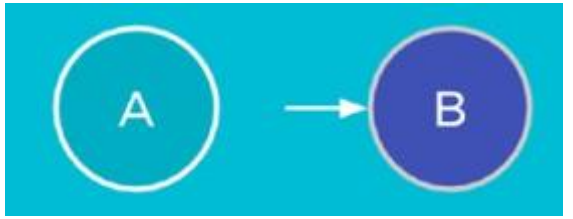
Interactive programming

UNIVERSITEIT GENT

- Object A kent de interface van object B

```
1   class Calculator {
2       sum(a, b) {
3           return a + b;
4       }
5   }
6
7   class Receipt {
8       constructor(calculator) {
9           this.calc = calculator;
10      }
11      print(itemA, itemB) {
12          const total = this.calc.sum(itemA, itemB);
13          console.log(`total receipt £${total}`);
14      }
15  }
16
17  const pizza = 6.00;
18  const beer = 5.00;
19  const calc = new Calculator();
20  const receipt = new Receipt(calc);
21
22  receipt.print(pizza, beer);
```

Reactive programming



UNIVERSITEIT GENT

- Object A reageert op veranderingen in object B
  - zonder de interface van B te kennen
  - kent enkel de methode **subscribe**
- Rxjs: veel gebruikte bibliotheek voor reactive programming

```
import { of } from 'rxjs';
import { reduce } from 'rxjs/operators';

class Calculator {
    constructor(itemA, itemB) {
        const obs = of(itemA, itemB);
        const sum = obs.pipe(reduce((acc, item) => (acc + item)));
        return { observable: sum };
    }
}

class Receipt {
    constructor(observable) {
        observable.subscribe(value => console.log(`total receipt: €${value}`));
    }
}

const pizza = 6.00;
const beer = 5.00;
const calc = new Calculator(pizza, beer);
const receipt = new Receipt(calc.observable);
```

Overzicht

UNIVERSITEIT GENT

- Wat is reactive programming?
- Verschillende programmeerparadigma's

Programmeerparadigma's

UNIVERSITEIT GENT

- **Imperative** Programming
  - Stap voor stap beschrijven wat het programma moet doen

```
1   class Calculator {
2       constructor() {
3           this.VAT = 22;
4       }
5
6       sum(...items) {
7           let total = 0;
8           for (let i = 0; i < items.length;i++) {
9               total += (1+this.VAT/100)*items[i];
10          }
11          return total;
12      }
13  }
```

```
15  class Receipt {
16      constructor(calculator) {
17          this.calc = calculator;
18      }
19
20      print(...items) {
21          let total = this.calc.sum(...items);
22          console.log(`total receipt €${total}`);
23      }
24  }
25
26  const JEANS = 80.00;
27  const SHIRT = 35.00;
28  const SHOES = 90.00;
29  const COAT = 140.00;
30  const HAT = 29.00;
31  const calc = new Calculator();
32  const receipt = new Receipt(calc);
33  receipt.print(JEANS, SHIRT, SHOES, COAT, HAT);
```

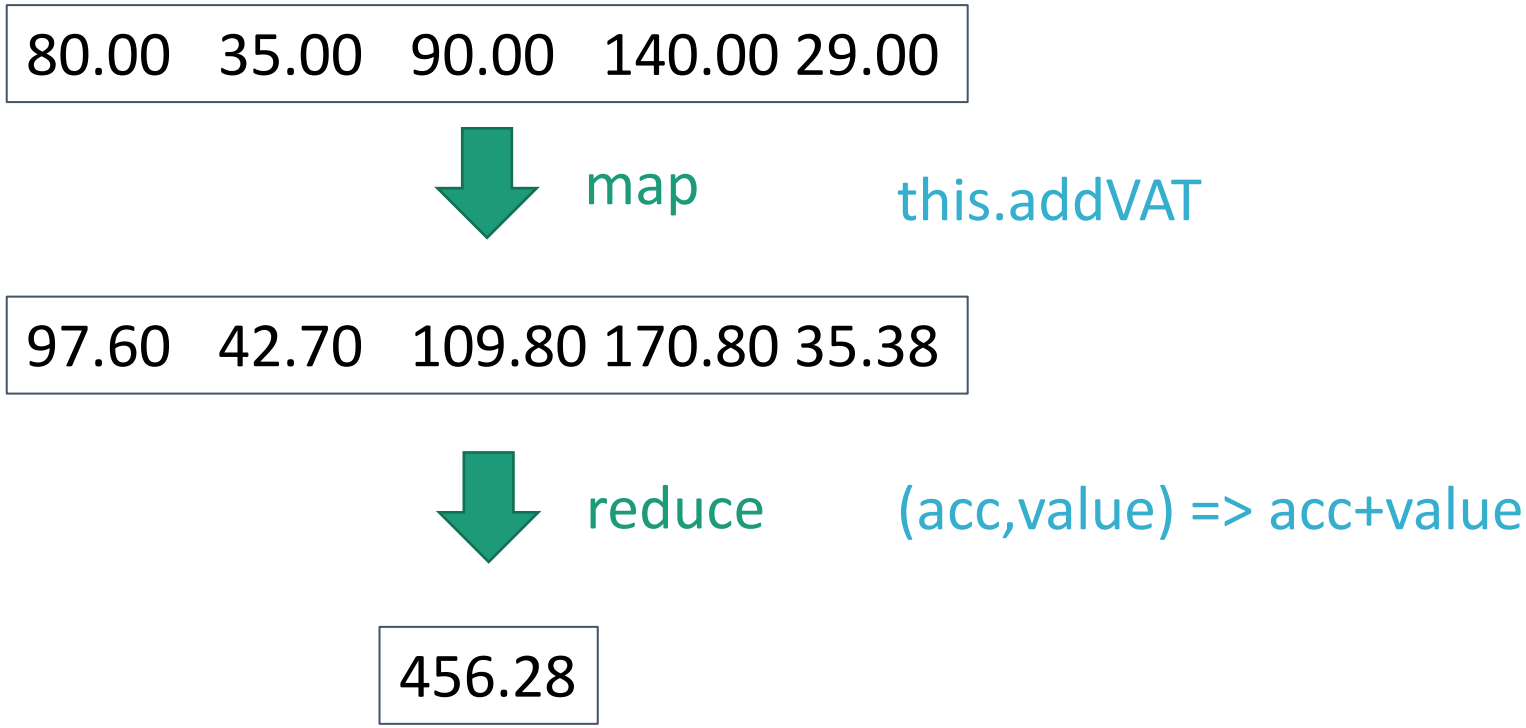
Programmeerparadigma's

UNIVERSITEIT

- **Functional** Programming
  - Het programma beschrijft de data flow
  - Functies gebruiken om op basis van bestaande waarden nieuwe waarden te maken
  - Waarden zijn immutable, onveranderlijk
  - Geen toestandsinformatie - instantievariabelen

```
1   class Calculator {
2
3       addVAT(item) {
4           return (1+22/100)*item;
5       }
6
7       sum(...items) {
8           return items.map(this.addVAT).reduce((acc,value) => acc+value);
9       }
10  }
11
12  class Receipt {
13      print(total) {
14          console.log(`total receipt €${total}`);
15      }
16  }
17
18  const JEANS = 80.00;
19  const SHIRT = 35.00;
20  const SHOES = 90.00;
21  const COAT = 140.00;
22  const HAT = 29.00;
23  const calc = new Calculator();
24  const receipt = new Receipt();
25  receipt.print(calc.sum(JEANS, SHIRT, SHOES, COAT, HAT));
```

Programmeerparadigma's: functioneel programmeren



Overzicht

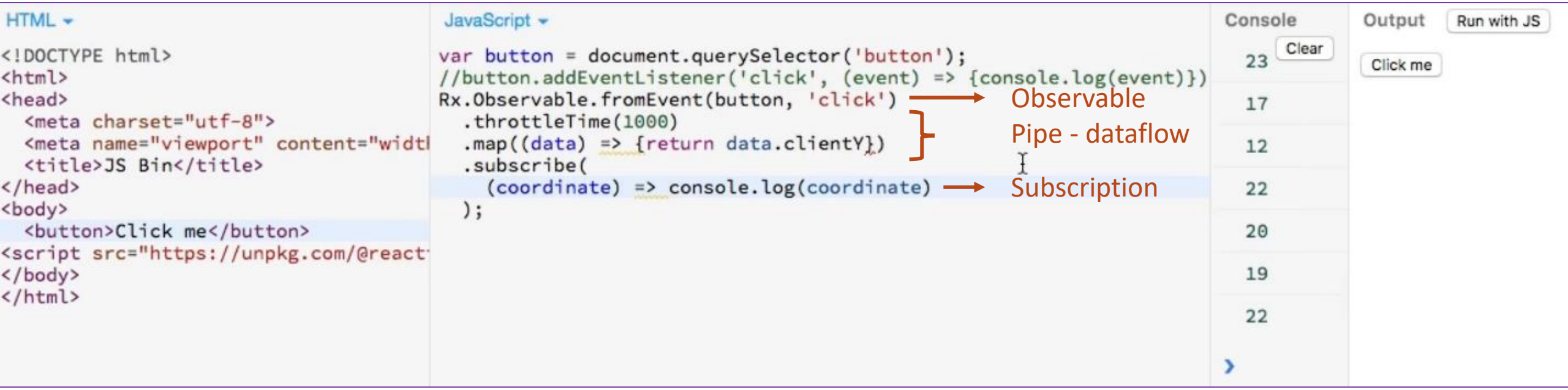
UNIVERSITEIT GENT

- Wat is reactive programming?
- Verschillende programmeerparadigma's
- Principe reactive programming

Reactive Programming - voorbeeld

UNIVERSITEIT GENT

- http://www.youtube.com/watch?v=T9wOu11uU6U



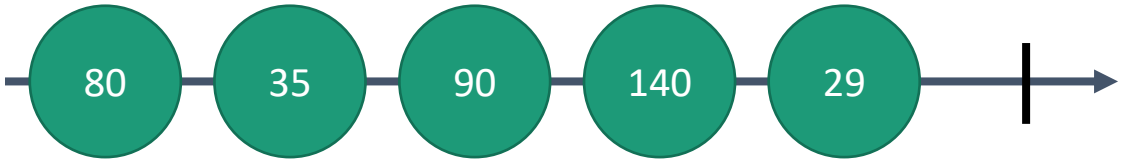
Programmeerparadigma's

UNIVERSITEIT GENT

- **Reactive** Programming
  - Een object, functie, stukje code, … luistert en reageert op een veranderlijke data flow

Reactive Programming is a paradigm based on asynchronous data streams that propagate changes during the application life cycle.

Programmeerparadigma's – reactive programming



Observable

```
map(value => (1 + this.VAT / 100) * value))
```

Observable

```
reduce((acc, value) => acc + value))
```

Observable

Programmeerparadigma's – reactive programming

UNIVERSITEIT GENT

```
1  import { from } from 'rxjs';
2  import { map, reduce } from 'rxjs/operators';
3
4  class Calculator {
5      constructor() {
6          this.VAT = 22;
7      }
8
9      sum(items) {
10         return from(items)
11             .pipe(
12                 map(value => (1 + this.VAT / 100) * value),
13                 reduce((acc, value) => acc + value));
14     }
15 }
```

Array → Observable

pijplijn

```
28 const JEANS = 80.00;
29 const SHIRT = 35.00;
30 const SHOES = 90.00;
31 const COAT = 140.00;
32 const HAT = 29.00;
33 const calc = new Calculator();
34 const receipt = new Receipt(calc);
35 receipt.print(JEANS, SHIRT, SHOES, COAT, HAT);
```

```
17 class Receipt {
18     constructor(calculator) {
19         this.calc = calculator;
20     }
21
22     print(...items) {
23         this.calc.sum(items)
24             .subscribe(total => console.log(`total receipt: €${total}`));
25     }
26 }
```

Observable + pijplijn

subscription

FACULTEIT EN ARCH

Samengevat

- Wat is reactive programming?
- Verschillende programmeerparadigma's
- Principe reactive programming
- Basisconcepten reactive programming

Doel

UNIVERSITEIT GENT

- Basisconcepten van reactive programming in JS kennen en kunnen gebruiken
  - Observable
  - Observer
  - Subscription
  - Operators
  - Hot vs Cold Observables

Reactive programming: Observable

UNIVERSITEIT GENT

- Een collectie van toekomstige waarden of events
  - Er kunnen waarden later toegevoegd worden
- Observer/Consumer
  - Reageert telkens er een waarde toegevoegd wordt (emit)
- Subscription
  - Voegt een observer (=functie) toe die uitgevoerd bij een nieuwe waarde
  - Functie `subscribe`

Observable: voorbeeld op basis van functie

parameter = observer
= argument subscribe-functie

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

let observable = new Observable( subscribe: (observer) => {
    observer.next( value: 1);
    observer.next( value: 2);
    observer.next( value: 3);
    setTimeout( handler: () => {
        observer.next( value: 4);
        observer.complete();
    }, timeout: 1000);
});

console.log('just before subscribe');
observable.subscribe( observer: {
    next: x => console.log('got value ' + x),
    error: err => console.error('something wrong occurred: ' + err),
    complete: () => console.log('done'),
});
console.log('just after subscribe');
```

functie

```
just before subscribe
got value 1
got value 2
got value 3
just after subscribe
got value 4
done
```

Reactive programming: methode subscribe van een Observable

UNIVERSITEIT GENT

- Subscribing to an Observable is analogous to calling a Function

- Pas bij het oproepen van de methode `subscribe` worden de waarden in de Observable verwerkt

Reactive programming: voorbeeld Observable op basis van Array

```
1   import { from } from 'rxjs';
2
3   function onData(value) {
4       console.log(value);
5   }
6   function onError(err) {
7       console.error(err);
8   }
9   function onComplete() {
10      console.log("stream complete!");
11  }
12
13  const lijst = Array.from(new Array(10), (x,i) => i+1);
14  const observable = from(lijst);
15  const observer = observable.subscribe(onData, onError, onComplete);
```

```
1
2
3
4
5
6
7
8
9
10
stream complete!
```

Reactive programming: voorbeeld Observable op basis van interval

UNIVERSITEIT GENT

```
1   import { interval } from 'rxjs';
2
3   const source = interval(1000);
4   source.subscribe(value => console.log("eerste subscription", value));
5   source.subscribe(value => console.log("tweede subscription", value));
6   source.subscribe(value => console.log("derde subscription", value));
```

```
eerste subscription 0
tweede subscription 0
derde subscription 0
eerste subscription 1
tweede subscription 1
derde subscription 1
eerste subscription 2
tweede subscription 2
derde subscription 2
```

Hot vs Cold Observables

- Cold Observable
  - Data geproduceerd in de “Observable”
- Hot Observable
  - Data geproduceerd buiten de “Observable”

bron: https://medium.com/@luukgruijs/understanding-hot-vs-cold-observables-62d04cf92e03

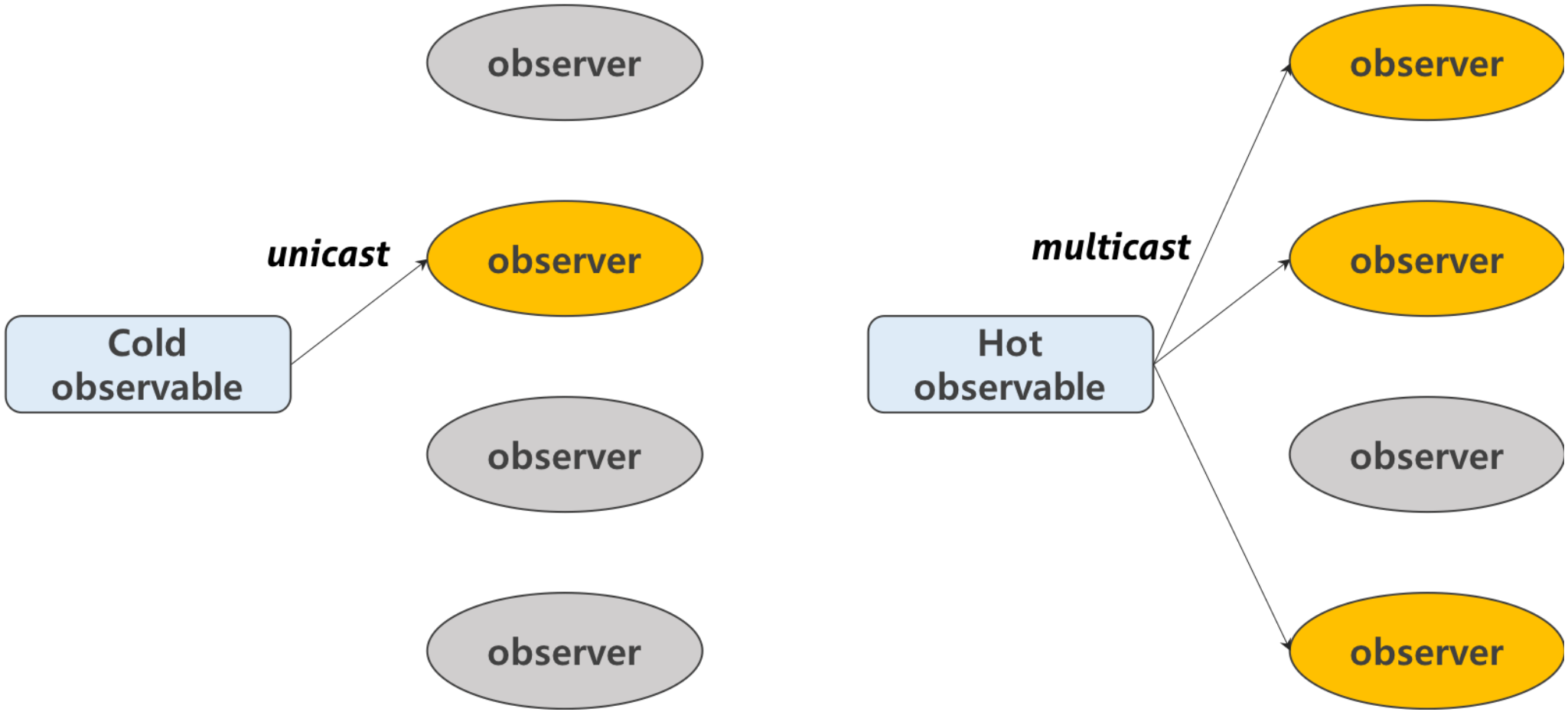
Cold Observable

```
1   import {Observable} from "rxjs";
2
3   const observable = new Observable( subscribe: (observer) => {
4       observer.next(Math.random());
5   });
6
7   // subscription 1
8   observable.subscribe( next: (data) => {
9     console.log(data); // 0.24957144215097515 (random number)
10  });
11
12  // subscription 2
13  observable.subscribe( next: (data) => {
14     console.log(data); // 0.004617340049055896 (random number)
15  });
```

Hot Observable

```
1  import {Observable} from "rxjs";
2
3  const random = Math.random();
4
5  const observable =  new Observable( subscribe: (observer) => {
6      observer.next(random);
7  });
8
9  // subscription 1
10 observable.subscribe( next: (data) => {
11     console.log(data); // 0.11208711666917925 (random number)
12 });
13
14 // subscription 2
15 observable.subscribe( next: (data) => {
16     console.log(data); // 0.11208711666917925 (random number)
17 });
```

Hot vs Cold Observable



bron: https://poiemaweb.com/angular-rxjs-observable

Overzicht

UNIVERSITEIT GENT

- Wat is reactive programming?
- Verschillende programmeerparadigma's
  - Imperative programming
  - Functional programming
  - Reactive programming
- Basisconcepten reactive programming
  - Observable – Observer - Subscription
- Operators RxJS
  - of: parameters → Observable
  - from: array, list, … → Observable
  - fromEvent: target, event → Observable
  - interval: tijdsinterval → Observable
  - map
  - reduce
- Methodes Observable
  - pipe
  - subscribe
- Hot vs Cold Observable