# Javascript – asynchroon programmeren

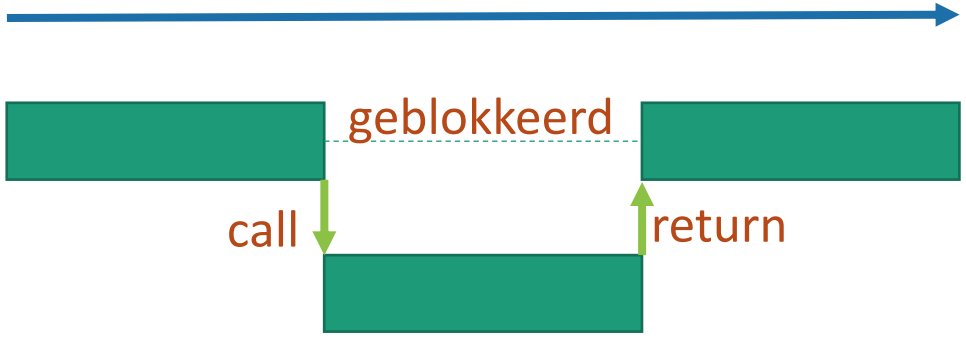
Veerle Ongenae

# Doel

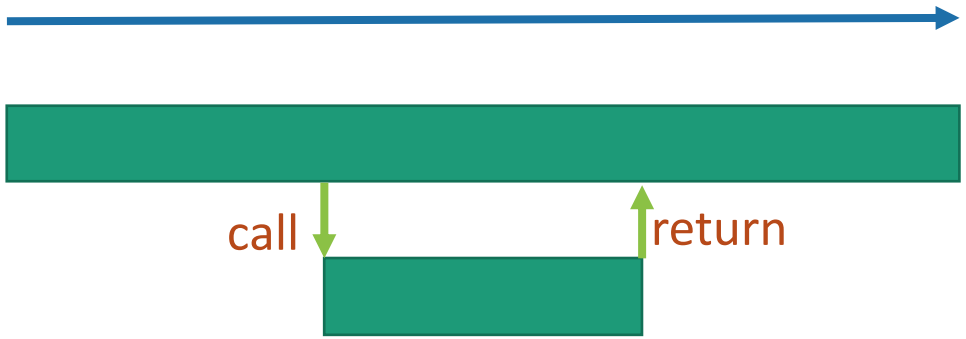
- Asynchroon versus synchroon programmeren
- Hoe asynchroon programmeren in javascript
  - Callback-functies
  - Promises
  - Asynchrone functies

# Synchrone en asynchrone functie

SYNCHROON




ASYNCHROON



# Synchroon vs asynchroon

UNIVERSITEIT GENT

Synchroon

```
result = query('SELECT * FROM posts WHERE id = 1');
do_something_with(result);
```

Asynchroon

```
query('SELECT * FROM posts WHERE id = 1', do_something_with);
```

Callback-functie

# Synchroon vs asynchroon

UNIVERSITEIT GENT

## Synchroon

```
let fs = require("fs"); // oude stijl modules

let data = fs.readFileSync('input.txt');

console.log(data.toString());
console.log("Program Ended");
```

```
De inhoud van het
bestand.

Program Ended
```

## Asynchroon

```
let fs = require("fs");

fs.readFile('input.txt', function (err, data) {
    if (err) return console.error(err);
    console.log(data.toString());
});

console.log("Program Ended");
```

```
Program Ended
De inhoud van het
bestand.
```



# Callback hell

- Veel functies met callbacks die genest zijn

```
request('http://www.somepage.com’,
  function (firstError, firstResponse, firstBody) {
    if(firstError){
        // Handle error.
    }
    else {
        request(`http://www.somepage.com/${firstBody.someValue}`,
          function (secondError, secondResponse, secondBody) {
            if(secondError){
                // Handle error.
            }
            else {
                // Use secondBody for something
            }
        });
    }
});
```

# Promises

UNIVERSITEIT GENT

- Oplossing voor callback hell
- Promise = object toekomstig resultaat asynchrone functie
  - Drie toestanden: nog niet afgerond, vervuld, fout opgetreden
- Asynchrone functie
  - Functie keert onmiddellijk terug
  - Caller wacht niet op volledig uitvoeren functionaliteit
- Promise laat toe om callback of foutafhandeling toe te voegen



# Syntax Promise

```
new Promise(executor);
```

- executor
  - Functie met twee parameters
    - resolve
    - reject
- De executor-functie wordt onmiddellijk uitgevoerd en start asynchrone opdrachten. Als die opdrachten klaar zijn roept hij de resolve-functie op. Als er een fout optrad, wordt de reject-functie opgeroepen.

# Promises

```
let p = new Promise((resolve, reject) => {
    console.log("In functie");
    let kans = Math.floor(Math.random() * 2);
    console.log("kans: "+kans);
    if (kans === 0) {
        resolve("Gelukt");
    } else {
        reject("Fout");
    }
}
);
p.then((tekst) => console.log(tekst))
        .catch ((fout) => console.log(fout));
console.log("Na functie");
```

callback = param then

functie = param catch

```
In functie
kans: 1
Na functie
Fout
```

```
In functie
kans: 0
Na functie
Gelukt
```

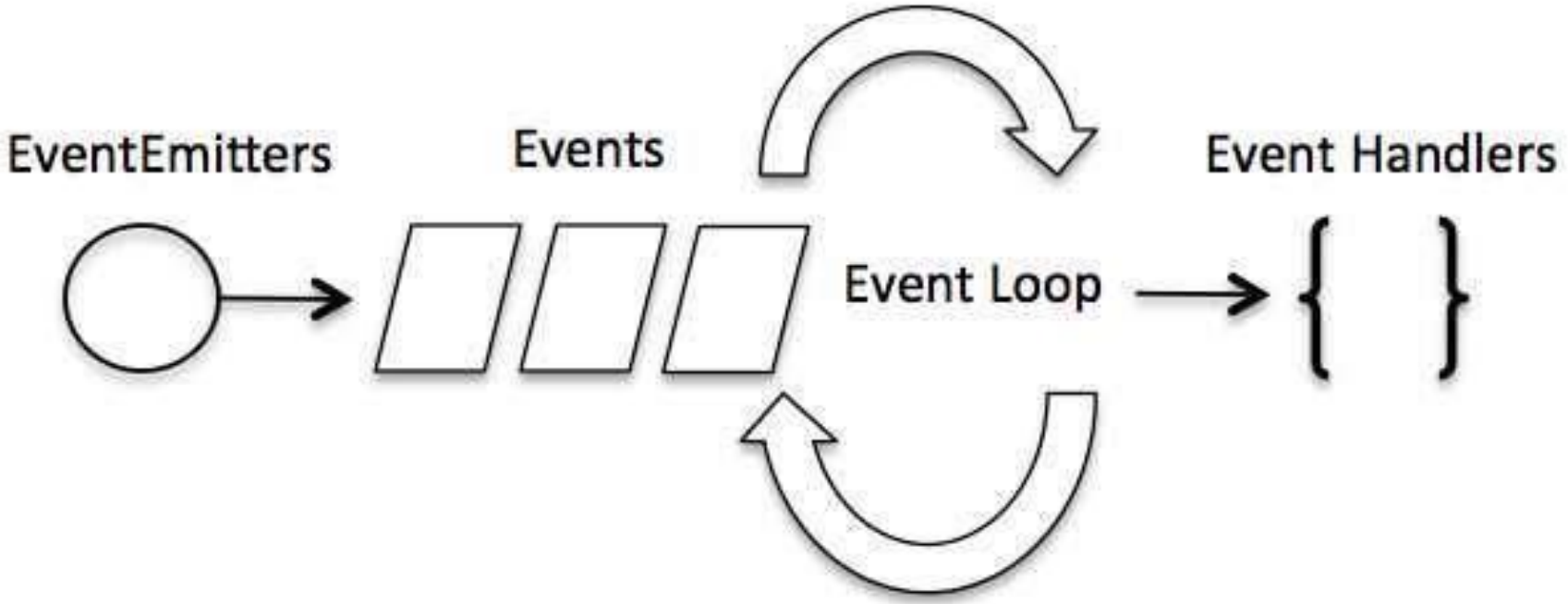
# Promises - chaining

```
function log(bericht) {
    console.log(bericht);
}

function logEnGok(bericht) {
    return new Promise((resolve, reject) => {
        log(bericht);
        if (Math.floor(Math.random() * 3) !== 0) {
            resolve("Gelukt");
        } else {
            reject("Fout");
        }
    }
    );
}

logEnGok("Start").then(logEnGok)
        .then(logEnGok).then(logEnGok).catch(log);
log("Na logEnGok");
```

```
Start
Na logEnGok
Gelukt
Gelukt
Fout
```

```
Start
Na logEnGok
Gelukt
Gelukt
Gelukt
```

# Asynchroon gerealiseerd




bron: https://medium.com/@tigranbs/concurrency-vs-event-loop-vs-event-loop-concurrency-eb542ad4067b

# Asynchrone functies

- Gebruik Promises vereenvoudigen
- Vanaf ECMA 7

```
async function name([param[, param[, ... param]]]) {
   statements
}
```

- Het resultaat van een asynchrone functie is een Promise
- In een asynchrone functie kan een await-opdracht gebruikt worden. In dat geval pauzeert de uitvoering van de functie, start een asynchrone opdracht, keert terug naar de caller en wacht totdat de opgeroepen promise klaar is.

# Voorbeeld Promise

```
1  function doubleAfter2Seconds(x) {
2      return new Promise(resolve => {
3          setTimeout(() => {
4              resolve(x * 2);
5          }, 2000);
6      });
7  }
```

```
na oproep doubleAfter2Seconds
20
```

```
28 doubleAfter2Seconds(10).then((r) => {
29     console.log(r);
30 });
31 console.log("na oproep doubleAfter2Seconds");
```

# Voorbeeld meerdere Promises

```
 9  function addPromise(x) {
10      return new Promise(resolve => {
11          doubleAfter2Seconds(10).then((a) => {
12              doubleAfter2Seconds(20).then((b) => {
13                  doubleAfter2Seconds(30).then((c) => {
14                      resolve(x + a + b + c);
15                  });
16              });
17          });
18      });
19  }
```

```
na oproep addPromise
Met promises 130
```

```
33  addPromise(10).then((sum) => {
34      console.log("Met promises",sum);
35  });
36  console.log("na oproep addPromise");
```

# Alternatief voorbeeld met asynchrone functies

UNIVERSITEIT GENT

```
21  async function addAsync(x) {
22      const a = await doubleAfter2Seconds(10);
23      const b = await doubleAfter2Seconds(20);
24      const c = await doubleAfter2Seconds(30);
25      return x + a + b + c;
26  }
```

```
38  addAsync(10).then((sum) => {
39      console.log("Met asynchrone functie", sum);
40  });
41  console.log("na oproep addAsync");
```

```
na oproep addAsync
Met asynchrone functie 130
```



# Doel

- Asynchroon versus synchroon programmeren
- Hoe asynchroon programmeren in javascript
  - Callback-functies
  - Promises
  - Asynchrone functies

# Consoleprogramma's in Javascript

UNIVERSITEIT GENT

- Node en nmp installeren
  - Node = javascript-runtime
  - Npm = node package manager
    - ➢ Modules installeren
    - ➢ In map node_modules
- Welke modules gebruik je in je programma?
  - packages.json
- Modules installeren

```
npm install
```

- Programma met modules uitvoeren

```
node -r esm vb2Observable.js
```

```
{
  "name": "vbreactiveprogramming",
  "version": "1.0.0",
  "description": "",
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "esm": "^3.2.25",
    "rxjs": "*"
  }
}
```