Introduction à NODEJS

Sogelink So easy

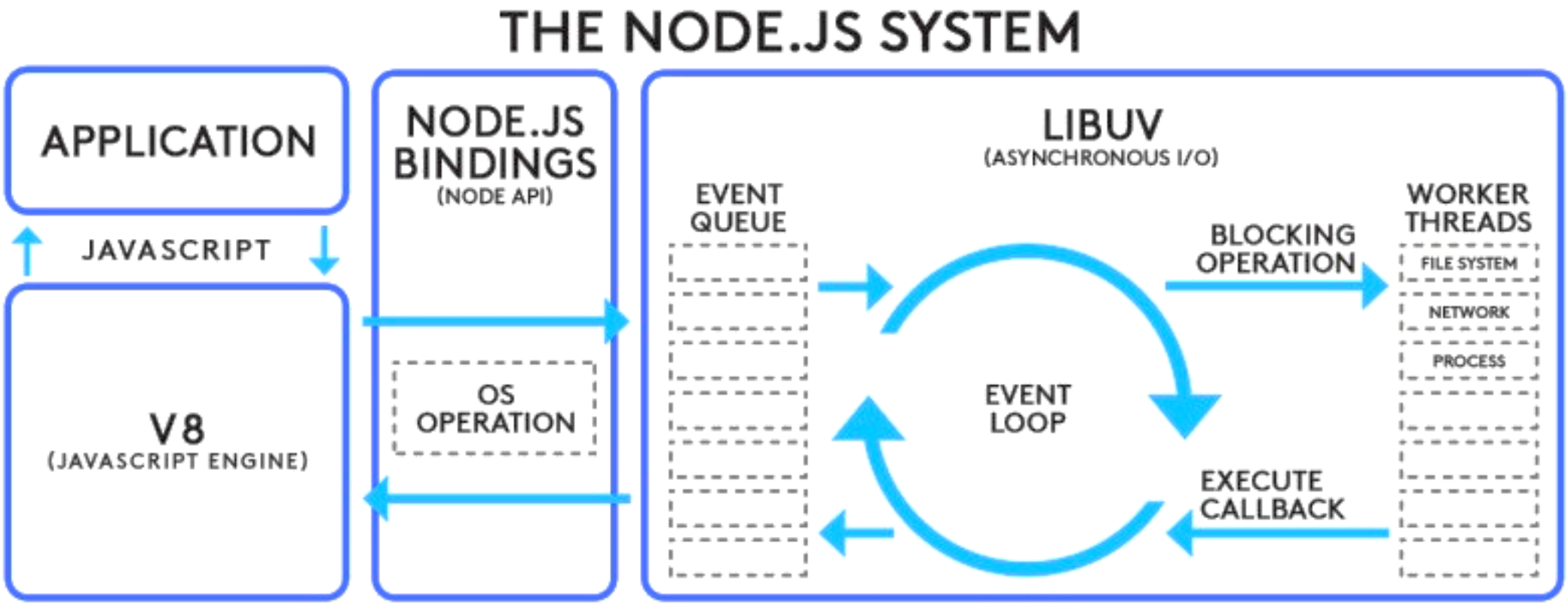
Historique

	NodeJS
1995	Naissance du Javascript dans Netscape
2008	Première version de Google Chrome et de V8
2009	Création de NodeJS par Ryan Dahl
2010	
2012	Départ de Ryan Dahl du projet
2014	'The Big Fork' : Création du fork io.js
2015	Naissance de la Fondation Node.js NodeJS 0.12 et io.js 3 fusionnent NodeJS 4
2016	NodeJS 6
2017	V8 inclut Node.js dans sa suite de tests

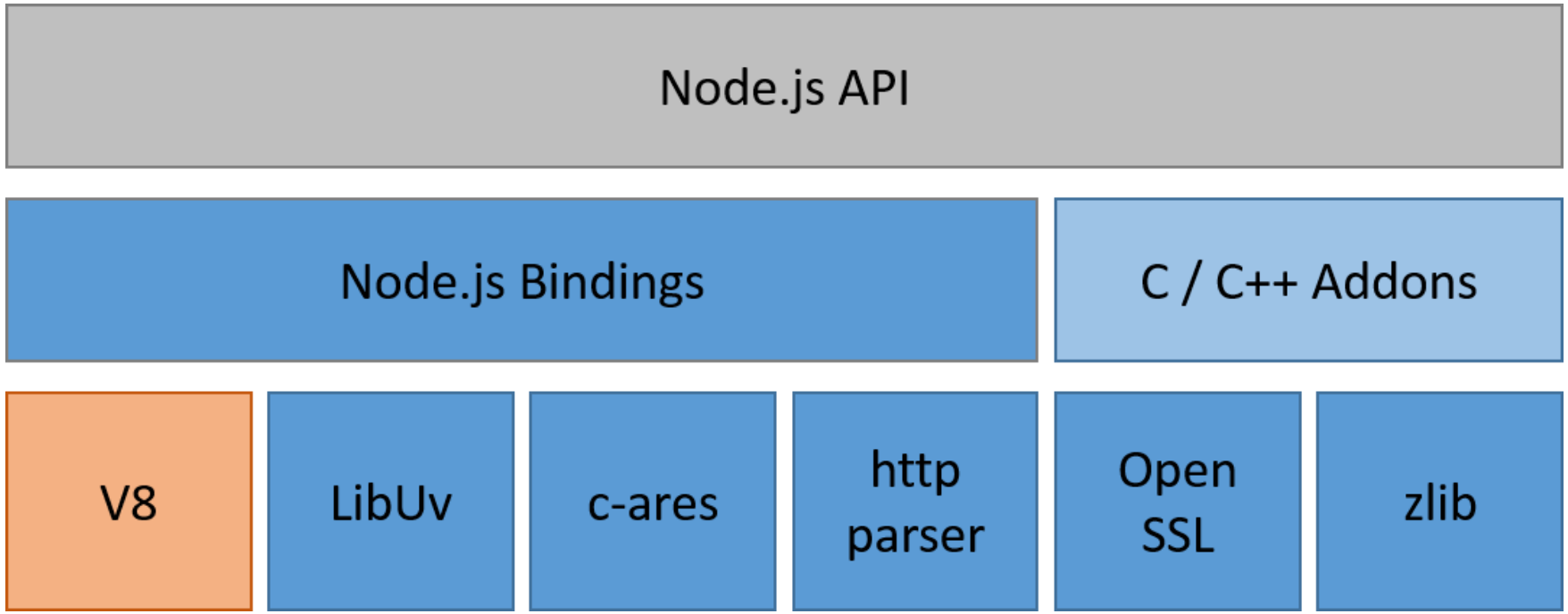
Présentation

- C’est une ligne de commande !
- C’est un Framework Javascript côté serveur
- Multi-environnements et open-source
- Codé en C++ (70%) et en Javascript (30%)
- Pensé pour construire des applications réseaux qui doivent supporter des montées en charge
- C’est « single-thread »
- Node.js est la combinaison du moteur Javascript V8 de Chrome, d’une API d’E/S de bas niveau, et d’une boucle d’événements

Architecture



Architecture



Le cas d'une requête en base de donnée...

- Beaucoup d'applications web utilise du code comme ceci:

```
result = query('select * from T');

// use result
```

Le cas d'une requête en base de donnée...

> Beaucoup d'applications web utilise du code comme ceci:

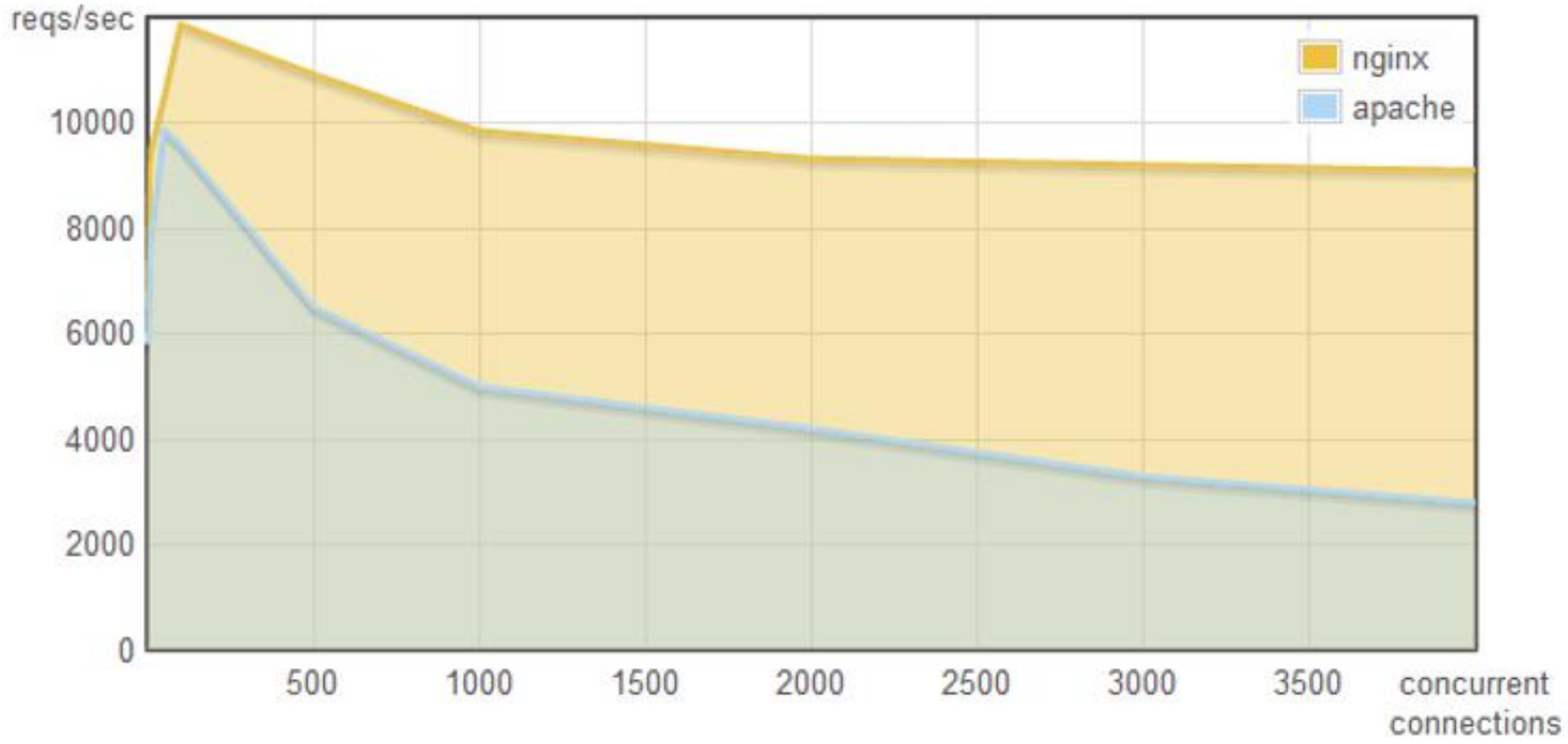
```
result = query('select * from T');

// use result
```

Que fait le serveur en attendant les résultats de sa requête ?

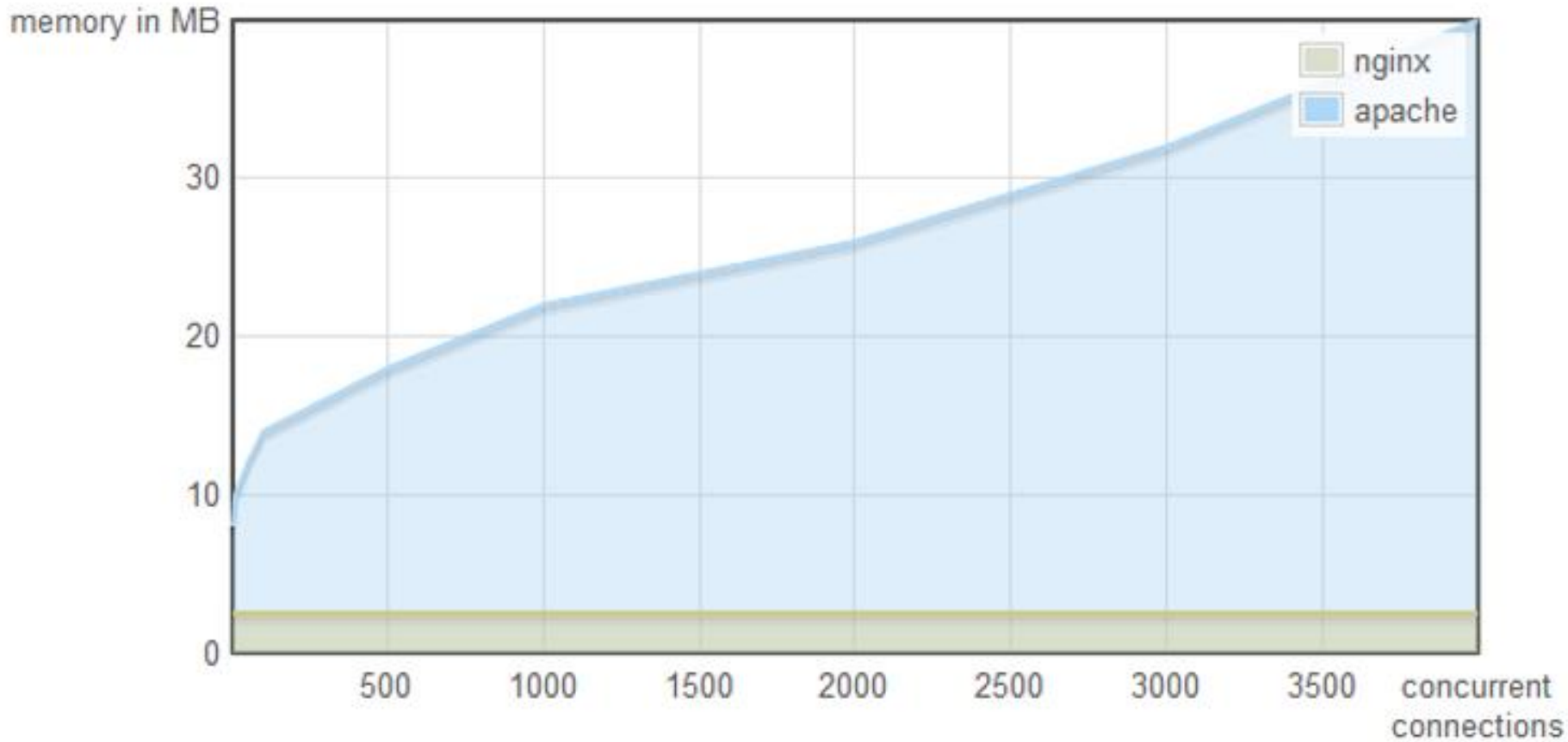
Apache vs. NGINX

> Requêtes par secondes



Apache vs. NGINX

> Utilisation de la mémoire



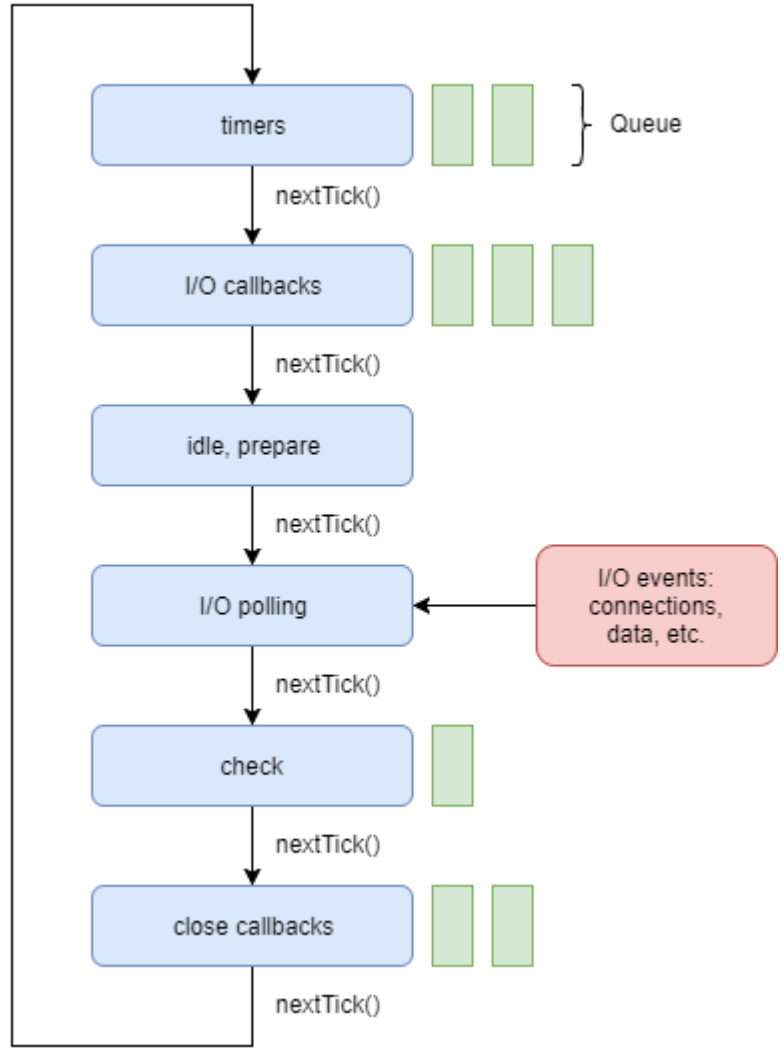
Apache vs. NGINX

- Pour Apache: 1 connexion = 1 thread

- NGINX n'utilise pas de thread. Il utilise une boucle d'évènements (**event loop**)

Event loop

- Permet d'effectuer les traitements asynchrones
- Chacune de ces phases possède une file (FIFO) de callback a exécuter. Dès que la file est vide, on passe à la phase suivante
  - **timers** : Exécute les callbacks des fonctions setTimeout et setInterval;
  - **I/O callbacks** : Phase de traitement des callbacks concernant les traitements asynchrones (sauf les callbacks close, les timers et les setImmediate)
  - **idle, prepare** : Phase utilisée en interne
  - **I/O polling** : Récupération ou attente de nouveaux évènements I/O.
    - C'est ici que les traitements synchrones sont réalisés ce qui peut potentiellement bloquer l'event loop.
    - Si l'application est inactive, la boucle d'événements reste dans cette phase en attentes de nouveaux événements externes
  - **check** : Exécute les callbacks de la fonction setImmediate
  - **close callback** : Exécute les callbacks de fermeture (.on('close'))
  - **nextTick** :
    - N'est pas une phase de l'event loop
    - Elle exécute les callbacks de la fonction nextTick à la fin de chaque phase



Classe EventEmitter

- Tous les objets qui émettent des événements étendent la classe EventEmitter
  - eventEmitter.on() permet d'attacher des fonctions à des événements
  - eventEmitter.emit() permet de déclencher des événements
- Lorsqu'un objet EventEmitter émet un événement :
  - Les fonctions attachées à cet événement sont appelées de manière synchrone
  - Les retours de ces fonctions sont ignorés

```
const EventEmitter = require('events');

class MyEmitter extends EventEmitter {}

const myEmitter = new MyEmitter();
myEmitter.on('event', () => {
  console.log('an event occurred!');
});
myEmitter.emit('event')
```

Le cas d'une requête en base de donnée...

- Version synchrone

```
result = query('select * from T');

// use result
```

Le cas d'une requête en base de donnée...

> Version synchrone

```
result = query('select * from T');

// use result
```

> Version asynchrone

```
query('select * from T', function(result) {

    // use result
});
```

Le cas d'une requête en base de donnée...

> Version synchrone

```
result = query('select * from T');

// use result
```

- Exemple d'implémentation

```
function query(queryString) {
  let result = executeQuery(queryString);
  // waiting...
  return result;
}
```

Le cas d'une requête en base de donnée...

> Version asynchrone

```
query('select * from T', function(result) {

    // use result
});
```

- Exemple d'implémentation

```
function query(queryString, callback) {
  executeQuery(queryString)
      .on('result', function (result) {

      callback(result);
  });
}
```

Convention des callbacks dans NodeJS

```
query('select * from T', function(error, result) {
    if (error) {
        // handle error
    }
    // use result
});
```

- 1er argument : Erreur
  - null si pas d'erreur
- 2ème argument (facultatif) : Données

Convention des callbacks dans NodeJS

```
query('select * from T', function(error, result) {
    if (error) {
        // handle error
    }
    // use result
});
```

- Exemple d'implémentation

```
function query(queryString, callback) {
  executeQuery(queryString)
      .on('result', function (result) {

      if (!result) {
        callback(new Error(`Error in query: result is null`));
      }
      callback(null, result);
  });
}
```

Code asynchrone en JS

	OK	KO
sync	`return result`	`throw error`
callback	`cb(null, result)`	`cb(error)`

Code asynchrone en JS

- Callbacks
- Promises (ES 2015)
- `async` / `await` (ES 2017)
- Observables

Promises

```
query('select * from T')
  .then(result => {
    // use result
  })
  .catch(error => {
    // handle error
  });
```

Promises

```
query('select * from T')
  .then(result => {
    // use result
  })
  .catch(error => {
    // handle error
  });
```

- Exemple d'implémentation

```
function query(queryString) {
  return new Promise((resolve, reject) => {
    executeQuery(queryString)
        .on('result', result => {

        if (!result) {
          return reject(new Error(`Error in query: result is null`));
        }
        return resolve(result);
    });
  });
}
```

Promises

- Rend le code plus lisible
- Remplacement des 'Callback Hell' par des chaînage de Promises
- `utils.promisify` permet de convertir une fonction avec callback en une Promise
  - depuis NodeJS 10

Promises

> Remplacement des '**Callback Hell**' par des chaînage de Promises

```
function asyncTask(i, cb) {
  let result = i+1;
  cb(null, result);
}

function runAsyncTasks(cb) {
  asyncTask(0, (err, a) => {
    asyncTask(a, (err, b) => {
      asyncTask(b, (err, c) => {
        asyncTask(c, (err, d) => {
          cb(null, d);
        });
      });
    });
  });
}

runAsyncTasks(function (err, result) {
  console.log(result)
});
```

Promises

> Remplacement des '**Callback Hell**' par des chaînage de Promises

```
function asyncTask(i) {
  let result = i+1;
  return Promise.resolve(result);
}

function runAsyncTasks() {
  return asyncTask(0)
    .then(a => { return asyncTask(a); })
    .then(a => asyncTask(a))
    .then(asyncTask);
}

runAsyncTasks()
  .then(result => console.log(result));
```

Code asynchrone en JS

	OK	KO
sync	`return result`	`throw error`
callback	`cb(null, result)`	`cb(error)`
Promises	`Promise.resolve(result)`	`Promise.reject(error)`

Async / Await

- Autre syntax pour utiliser des Promises
- Ressemble à du code synchrone
- Async : Une fonction définie avec le mot clé async renvoie systématiquement une promesse
  - Si une erreur est levée pendant l'exécution de la fonction, la promesse est rejetée
  - Si une valeur est retournée, la promesse est résolue avec cette valeur
- Await : Permet d'attendre la résolution d'une promesse et retourner sa valeur
  - Ne peut être utilisé que dans une fonction async

Async / Await

```
function asyncTask(i) {
  let result = i+1;
  return Promise.resolve(result);
}

async function runAsyncTasks() {
  const a = await asyncTask(0);
  const b = await asyncTask(a);
  const c = await asyncTask(b);
  const d = await asyncTask(c);
  return d;
}

runAsyncTasks()
  .then(result => console.log(result));
```

Observables

- 'Nouvelle' librairie (RxJs)
- Programmation Reactive
- Pas compatible par défaut avec NodeJS

Observable	Source/Flux de données (stream)
Subscriber	Ecoute une source de données
Operators	Transforme un Observable en Observable Modifie les données d'un Observable

Observables

	Single Value	Multiple Value
Synchrone	Get	Iterable
Asynchrone	Promise	Observable

Observables

```
import {Observable} from "rxjs";
import {map} from "rxjs/operators";

const observable = new Observable(observer => {

    setInterval(() => {
        observer.next(0);
    }, 1000);
});

observable.pipe(
    map(value => value+1),
    map(value => value+1),
    map(value => value+1),
    map(value => value+1)
).subscribe(value => console.log(value));
```

Modules

- Module ?
  - Bibliothèque, fichier, répertoire, dépendance
- En HTML, on utilise plusieurs balises **<script>** pour inclure des librairies.

Comment inclure du code Javascript dans du code Javascript ?

Modules

> Depuis le début, NodeJS implémente l'API **CommonJS** qui défini un module

- Déclaration d'un module:

```
module.exports = /* something */
```

- Inclusion d'un module:

```
require(/* something */)
```

- Exemple d'utilisation d'un module de base:

```
const fs = require('fs');

let content = fs.readFileSync('/etc/passwd');
console.log(content.toString());
```

Modules

> Depuis NodeJS v14, il est possible d'utiliser l'API **ES6+** pour définir ses modules

- Déclaration d'un module:

```
export /* something */
//  ou
export default /* something */
```

- Inclusion d'un module:

```
import something from ‘./something’
```

- Exemple d'utilisation d'un module de base:

```
import { readFileSync } from 'fs';

let content = readFileSync('/etc/passwd');
console.log(content.toString());
```

Modules de base

Name	Description
assert	provides a set of assertion functions useful for testing
buffer	provides the ability to handle buffers containing binary data
child_process	provides the ability to spawn child processes
console	provides a simple debugging console
cluster	allows to split a Node.js process into multiple workers to take advantage of multi-core systems
crypto	provides cryptographic functionality
dgram	provides an implementation of UDP Datagram sockets
dns	provides name resolution and DNS lookups
events	provides an API for managing events
fs	provides an API for interacting with the file system
http	provides an HTTP client/server implementation
http2	provides an HTTP/2 client/server implementation
https	provides an HTTPS client/server implementation
net	provides an asynchronous network API
os	provides operating system-related utility methods and properties
path	provides utilities for working with file and directory paths
perf_hooks	to enable the collection of performance metrics

Name	Description
process	provides information about, and control over, the current Node.js process
querystring	provides utilities for parsing and formatting URL query strings
readline	provides an interface for reading data from a Readable stream
repl	provides a Read-Eval-Print-Loop (REPL) implementation that is available both as a standalone program or includible in other applications
stream	an abstract interface for working with streaming data
string_decoder	provides an API for decoding Buffer objects into strings
timers	provide functions to schedule functions to be called at some future period of time
tls	provides an implementation of the Transport Layer Security (TLS) and Secure Socket Layer (SSL) protocols
tty	provides functionality used to perform I/O operations in a text terminal
url	provides utilities for URL resolution and parsing
util	supports the needs of Node.js internal APIs, useful for application and module developers as well
v8	exposes APIs that are specific to the version of V8 built into the Node.js binary
vm	enables compiling and running code within V8 Virtual Machine contexts
wasi	provides an implementation of the WebAssembly System Interface specification
worker	enables the use of threads that execute JavaScript in parallel
zlib	provides compression functionality

Modules de base

- Documentation:

  https://nodejs.org/dist/latest-v18.x/docs/api/

Définition d'un module CommonJS (Fonction)

> Module

```
// hello.js
module.exports = () => console.log('Hello :) (function)');
```

> Main

```
// app.js
const hello = require('./hello');

hello();
```

Définition d'un module ES6 (Fonction)

> Module

```
// hello.mjs
export default () => console.log('Hello =) (function)')
```

> Main

```
// app.mjs
import hello from './hello.mjs'

hello()
```

Définition d'un module CommonJS (Objet / Map)

> Module

```
// helloUtils.js
module.exports.hello = () => console.log('Hello :) (object)');
```

> Main

```
// app.js
const helloUtils = require('./helloUtils');

helloUtils.hello();
```

Définition d'un module ES6 (Objet / Map)

- Module

```
// helloUtils.mjs
export function hello() {
    console.log('Hello =) (object)')
}

export function hello2() {
    console.log('Hello 2')
}
```

- Main

```
// app.mjs
import * as helloUtils from './helloUtils.mjs'
helloUtils.hello()
helloUtils.hello2()
```

Définition d'un module CommonJS (Classe)

- Module

```
// HelloClass.js
module.exports = class HelloClass {
    constructor({name}) {
        this.name = name;
    }

    hello() { console.log(`Hello ${this.name}`); }
}
```

- Main

```
const HelloClass = require('./HelloClass');

new HelloClass({name: 'Adrien'}).hello();
```

Définition d'un module ES6 (Classe)

- Module

```
// HelloClass.mjs
export default class HelloClass {
    constructor({name}) {
        this.name = name;
    }

    hello() { console.log(`Hello ${this.name} (class)`) }
    static hello() { console.log(`Hello nobody`) }
}
```

- Main

```
// app.mjs
import HelloClass from './HelloClass.mjs'
new HelloClass({name: 'Adrien'}).hello();
HelloClass.hello();
```

CommonJS et Modules ES6

> Modules ES6 ≠ Modules CommonJS

> Par défaut, NodeJS utilise des modules CommonJS

> Pour utiliser les modules ES6:

- On utilise l'**extension de fichier** '**.mjs**' à la place de '.js' pour les fichiers de modules
- On spécifie '**type: "module"**' dans le package.json

	Création	Utilisation
CommonJS	module.exports	const fs = require('fs')
Module ES6	export (default)	import * as fs from "fs"

NPM

- NPM = NodeJS Package Manager
- Installé avec NodeJS depuis la version 0.6.3
- NodeJS: 3 342 873 paquets sur le dépôt central en 07/2023 (http://npmjs.org)
  - Debian: 64 419 paquets pour la version 12 (https://packages.debian.org)
  - Maven: 576 330 paquets (http://search.maven.org)

Historique

	NodeJS	NPM
1995	Naissance du Javascript dans Netscape	
2008	Première version de Google Chrome et de V8	
2009	Création de NodeJS par Ryan Dahl	
2010		Intégration de NPM dans NodeJS 0.6.3 Naissance d'Express Naissance de Socket.io
2012	Départ de Ryan Dahl du projet	
2014	'The Big Fork' : Création du fork io.js	
2015	Naissance de la Fondation Node.js NodeJS 0.12 et io.js 3 fusionnent NodeJS 4	
2016	NodeJS 6	The leftpad incident Naissance de Yarn
2017	V8 inclut Node.js dans sa suite de tests	3 milliards de téléchargements par semaine

NPM: package.json

> Fichier de description d'une application

```
{
    "name": "myApp",
    "version": "0.0.0",
    "private": true,
    "scripts": {
        "start": "node server.js"
    },
    "dependencies": {
        "express": "~4.9.0",
        "ejs": "~0.8.5",
        "underscore" : "*",
        "body-parser": "~1.0.0",
        "multer": "*",
        "q": "*"
    }
}
```

NPM: commandes de bases

- Initialisation d'une application

```
npm init
```

- Installation d'un module (socket.io)

```
npm install socket.io
```

- Installation d'un module et mise a jour du fichier package.json

```
npm install socket.io --save
```

- Installation d'une application

```
npm install
```

Express

- Framework pour faciliter la création d'application web
- Installation:

```
npm install express [--save]
```

- Documentation: http://expressjs.com/

Express

> Utilisation

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});

var server = app.listen(3000, function () {
  var host = server.address().address;
  var port = server.address().port;

  console.log('Example app listening at http://%s:%s', host, port);
});
```

Socket.io

- Module qui permet une communication bi-directionnel en temps réel
- Permet d'utiliser des websockets dans NodeJS
- Installation:

```
npm install socket.io [--save]
```

- Documentation: https://socket.io/

Socket.io : Utilisation côté serveur

```
const app = require('express')();
const server = require('http').createServer(app);
const io = require('socket.io');
const ioServer = io(server);

app.get('/', function(req, res){
  res.sendfile('index.html');
});

ioServer.on('connection', function(socket){
  console.log('a user connected');
  socket.on('myEvent1', function(data) {
    // Do stuff

    socket.emit('myEvent2', data);
  });
});

server.listen(3000);
```

Socket.io : Utilisation côté client

```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io();
  socket.emit('myEvent1', "Hello World");
  socket.on('myEvent2', function(data){
    alert(data);
  });
</script>
```

Stompit

- Module qui implémente un client au protocol STOMP
- Permet de communiquer avec un message broker (ActiveMQ, RabbitMQ...)
- Installation:

```
npm install stompit [--save]
```

- Documentation: http://gdaws.github.io/node-stomp/api/

Stompit - Producer

```
const stompit = require('stompit');

const connectOptions = {
    'host': 'localhost',
    'port': 61613
};

const headers = {
    'destination': '/queue/cpe-test'
};

stompit.connect(connectOptions, (error, client) => {
    if (error) {
        return console.error(error);
    }

    const frame = client.send(headers);
    frame.write('hello');
    frame.end();

    client.disconnect();
});
```

Stompit - Consumer

```
stompit.connect(connectOptions, (error, client) => {
    if (error) {
        return console.error(error);
    }

    client.subscribe(headers, (error, message) => {
        if (error) {
            return console.error(error);
        }

        message.readString('utf-8', (error, body) => {
            if (error) {
                return console.error(error);
            }

            console.log('received message: ' + body);
        });
    });
});
```


Merci pour votre attention