INTRODUCTION À

NODEJS

As an asynchronous event driven JavaScript runtime, Node is designed to build scalable network applications.

Source: nodejs.org

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser.

Source: wikipedia.org

HISTORIQUE

- 1995: Naissance du Javascript
- 1998: Arrivée de WebKit (Apple, Nokia, Blackberry…)
- 2008: Première version de Google Chrome et de V8
- 2009: Création de NodeJS par Ryan Dahl
- 2010: Intégration de NPM dans NodeJS
- 2012: Départ de Ryan Dahl du projet
- 2014: Création du fork io.js
- 2015: NodeJS 0.12 et io.js 3.3 fusionnent pour donner NodeJS 4

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 montée en charge
- C’est « single-thread »
- Utilise un modèle E/S non-bloquant, régit par des évènements

LE CAS D'UNE REQUÊTE EN BASE DE DONNÉE...

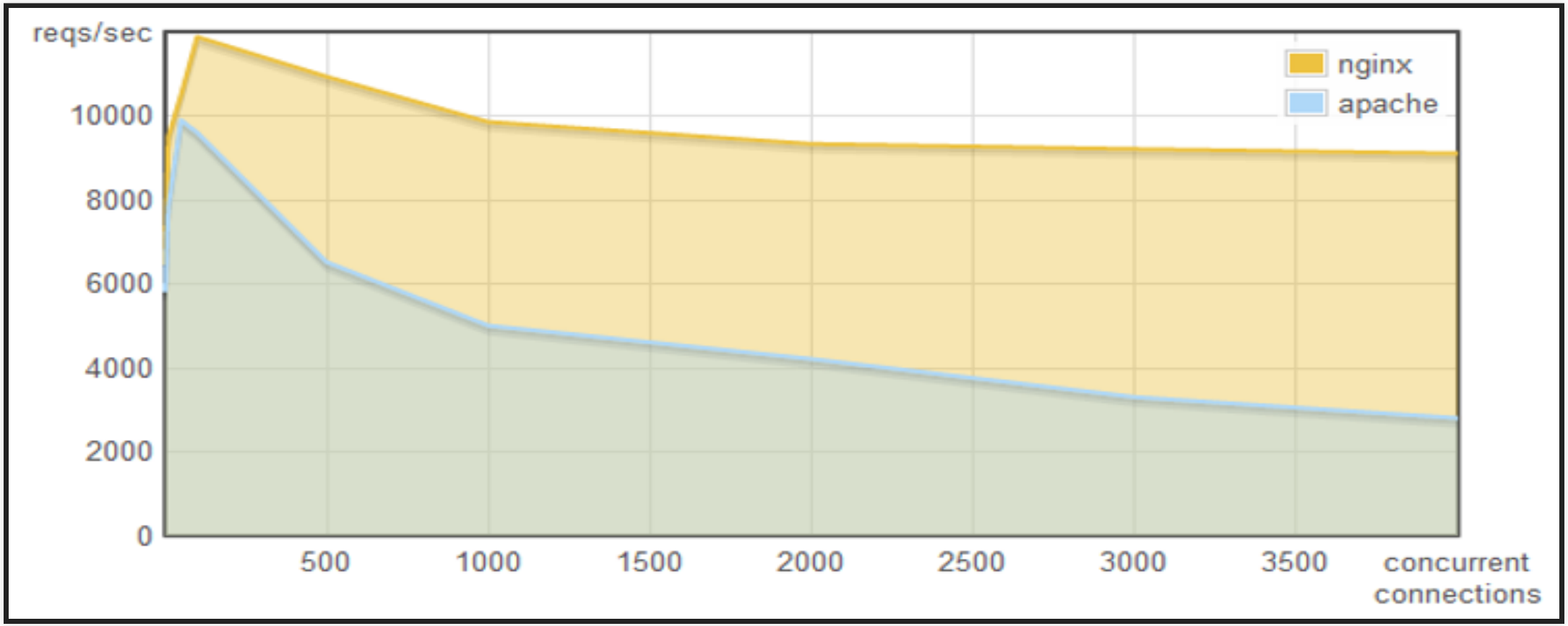
Beaucoup d'applications web utilise du code comme ceci:

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

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

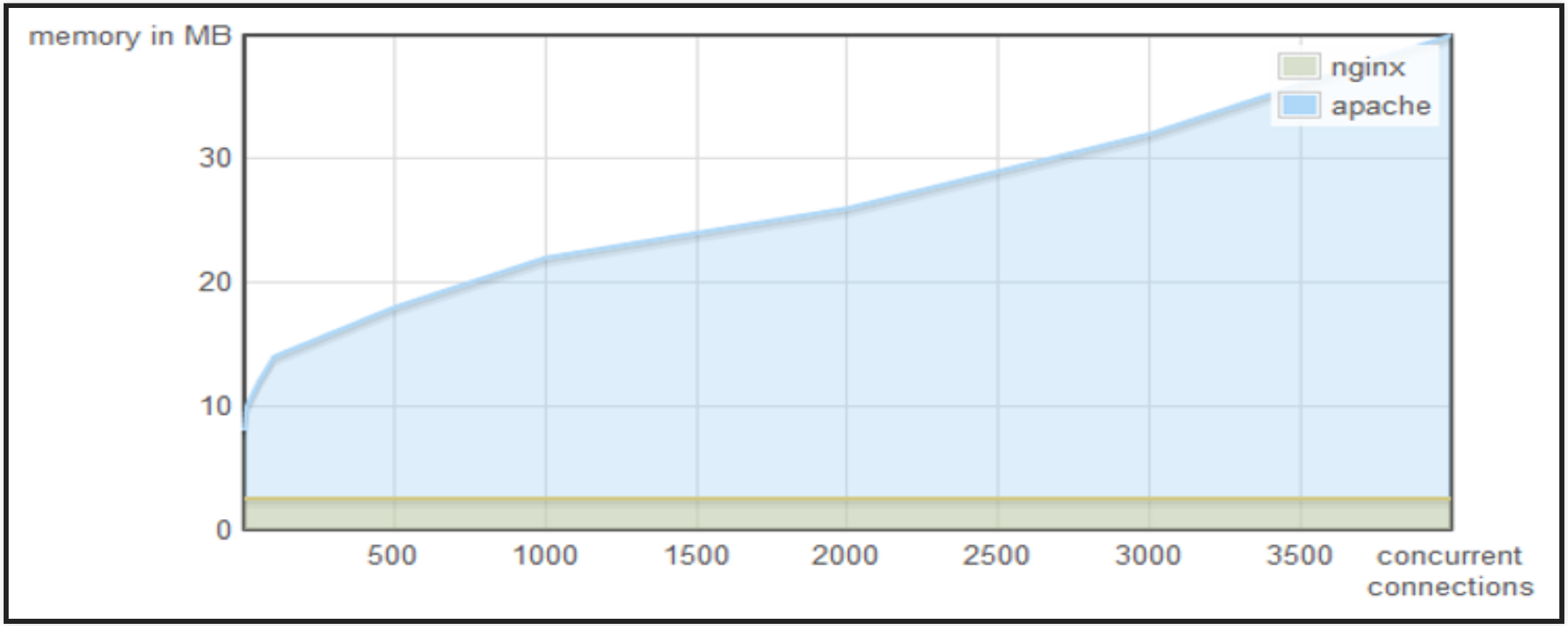
APACHE VS. NGINX

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

- C’est une boucle qui attend des évènements (disponibilité des E/S, timeout, interaction de l’utilisateur...)

- Lorsqu’un évènement est émis, on exécute une « fonction de retour » (**callback**)

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) {
  var 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) {
  var request = executeQuery(queryString);

  request.on('result', function (result) {
      callback(result);
  });
}
```

MODULES

- Module ?
  - Bibliothèque, fichier, répertoire, dépendance
- NodeJS implémente l'API **CommonJS** qui défini un module
- Appel d'un module:

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

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

MODULES DE BASES

'async_hooks' 'assert' 'buffer' 'child_process' 'console' 'constants' 'crypto' 'cluster' 'dgram' 'dns' 'domain' 'events' 'fs' 'http' 'http2' 'https' 'inspector' 'module' 'net' 'os' 'path' 'perf_hooks' 'process'

'punycode' 'querystring' 'readline' 'repl' 'stream' 'string_decoder' 'sys' 'timers' 'tls' 'tty' 'url' 'util' 'v8' 'vm' 'zlib'

- Documentation:

https://nodejs.org/dist/latest-v8.x/docs/api/documentation.html

DÉFINITION D'UN MODULE

Module

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

Main

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

hello();
```

DÉFINITION D'UN MODULE

Module

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

Main

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

helloUtils.hello();
```

DÉFINITION D'UN MODULE

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();
```

NPM

- NPM = NodeJS Package Manager
- Installé avec NodeJS depuis la version 0.6.3
- NodeJS: 706 094 paquets sur le dépôt central (npmjs.org)
  - Debian: 68 956 paquets pour la version 9 (packages.debian.org)
  - Maven: 251 221 paquets (search.maven.org)

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

```
var app = require('express')();
var server = require('http').createServer(server);
var io = require('socket.io');
var 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>
```

NODEJS

EXERCICES

EXERCICE 1

Écrire un programme qui dit « Hello world » dans la console.

EXERCICE 2

Écrire un programme qui accepte un ou plusieurs nombres comme arguments de la ligne de commande, et affiche la somme de ces nombres sur la console

EXERCICE 3

Ecrire un programme qui utilise une opération synchrone sur le système de fichiers pour lire un fichier et afficher son nombre de lignes sur la console (comme si vous faisiez `cat file | wc -l`).

Le chemin du fichier sera passé en argument de la ligne de commande

EXERCICE 4

Écrire le même programme que précédemment en asynchrone

EXERCICE 5

Créez un programme qui affiche une liste de fichiers au sein d'un répertoire donné, filtrés en fonction de leur extension.

- Le premier argument de la ligne de commande est le chemin du répertoire.
- Le deuxième argument est une extension de fichier à utiliser pour le filtrage.

La liste des fichiers sera affichée sur la console (un fichier par ligne)

Vous devez utiliser des E/S asynchrones.

EXERCICE 6

Rendre Modulaire l’exercice précédant

Le module en question doit exporter une unique fonction qui prendra trois arguments : le chemin du répertoire, l’extension de filtrage et un callback.

L’affichage se fait depuis le programme principal.

Pensez à traiter les erreurs remontées du module.

EXERCICE 7

Écrivez un programme qui fait une requête HTTP GET sur une URL fournie comme premier argument de la ligne de commande.

Affichez le contenu de chaque événement `'data'` de la réponse sur sa propre ligne dans la console.

EXERCICE 8

Ce problème est le même que le précédent, mais cette fois-ci vous allez recevoir trois URLs sur la ligne de commande.

Vous devrez collecter le contenu complet qui vous sera envoyé pour chaque URL, et l'afficher sur la console

La difficulté réside dans le fait que vous devez les afficher dans le même ordre que celui des URLs transmises sur la ligne de commande.

EXERCICE 9

Écrivez un serveur de temps TCP

Votre serveur devrait attendre des connexions TCP entrantes sur le port dont le numéro vous sera fourni comme premier argument en ligne de commande. À chaque connexion, vous écrirez sur la socket la date et l'heure courante selon le format suivant :

```
YYYY-MM-DD hh:mm
```

EXERCICE 10

Créer un serveur web de fichier statique simple.

Le port d’écoute et la liste de répertoires exposés par le serveur seront stockés dans un fichier de configuration au format JSON.

Penser à gérer les erreurs 404 et 500.