INTRODUCTION à Node.js

Partie 1

- **Introduction**
- Pourquoi Node.js ?
- Node.js est performant, mais ...
- Installation de Node.js, MongoDB
- Gestionnaire de paquet NPM,

Introduction

Définition : *« Node.js, en abrégé Node est une plate-forme logicielle libre et évènementielle en* ***JavaScript*** *orienté vers les applications réseau qui doivent pouvoir monter en charge ».*

Node n'est donc pas un Framework,

On pourrait plutôt comparer Node à la JVM pour Java.

C'est un interpréteur JavaScript basé sur le moteur JS de Google V8.

Pourquoi Node.js

Sa simplicité : de part l'utilisation du langage JavaScript : pas de multi-thread, mono-thread d'exécution, langage de script dynamique orienté Objets, langage évènementiel.

Ses performances : Node.js utilise V8 le moteur JavaScript de Google, reconnu pour ses performances et surtout sa capacité à compiler le JavaScript en langage machine (x86, ARM ou MIPS)

Unification du langage : Utiliser qu'un seul langage de développement côté client et serveur.

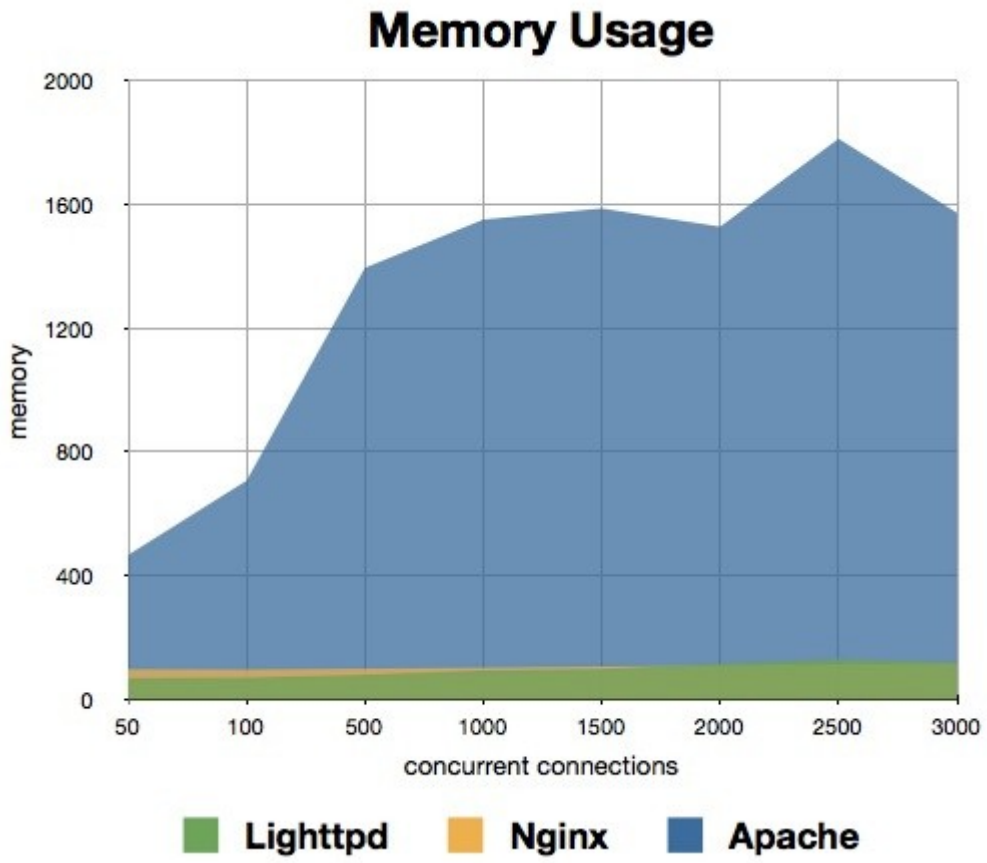
Node.js est performant, mais ...

- Oui il est performant, son architecture Single-Thread / Callback au dessus de V8 (code machine) lui permet de répondre très très vite à une demande sur un PORT.
- Mais … il y a des concepts à comprendre ...
- Attention, la programmation asynchrone, conséquence du Single-Thread* n'est pas triviale, et l'imbrication des callbacks peut perturber les programmeurs habitués à du code synchrone.

** la bouche d'événement d'exécution est non bloquante, rien ne doit l'arrêter, c'est pour cela qu'il existe des callbacks, afin de reprendre le cours de l'exécution du programme après une Entrée/Sortie (Thread noyau Linux).*

Node.js est performant, mais ...

Voyons une illustration graphique de la réduction des ressources nécessaires sur une architecture asynchrone
Apache (Synchrone), LightHttpd & Nginx (Asynchrone)

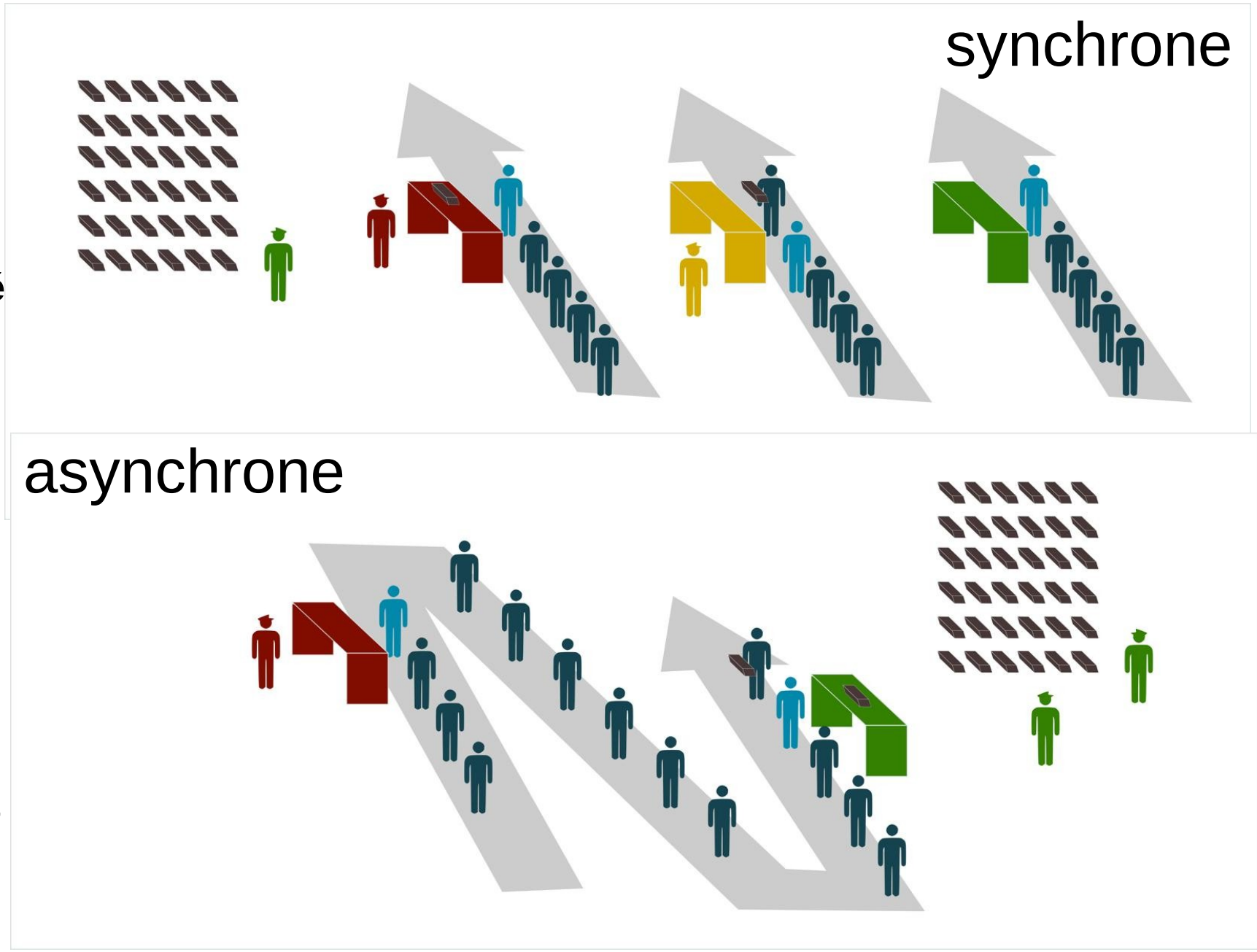


Node.js est performant, mais ...

Une image simple a comprendre est la différence entre le **Supermarché** (ci-contre)

et le

Drive, qui permet de lancer une demande de courses et d'avoir un « callback » par SMS pour les récupérer.



Installation de Node.js

Suivre la méthode, pour son système d'exploitation, sur le site de nodejs.org , ici pour ubuntu :

```
~$ sudo apt install nodejs
~$ sudo apt install npm
(…)
```

Nous venons d'installer la version « system » de NodeJS.

On peut connaître la version de NodeJS installée en ligne de commande, ainsi que la version de « npm » pour vérifier

```
~# nodejs -v
V8.10.0
~# npm -v
3.5.2
```

Notre environnement d'exécution est prêt !

Installation de Node.js (suite)

Il est aussi possible d'installer « nvm » un gestionnaire de version de NodeJS, permettant d'installer plusieurs versions :

```
~$ curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.34.0/install.sh | bash
(…)
```

On peut ensuite installer d'autres version de NodeJS, afin de pouvoir tester ses développements et les rendre compatibles avec les dernières versions de NodeJS. Il suffit d'utiliser l'option « install » de « nvm » :

```
~# nvm install <version nodejs> // ex : 8.12
~# nvm install 10.15.1
Downloading and installing node v10.15.1...
Downloading https://nodejs.org/dist/v10.15.1/node-v10.15.1-linux-x64.tar.xz...
######################################################################## 100,0%
Computing checksum with sha256sum
Checksums matched!
Now using node v10.15.1 (npm v6.4.1)
```

Gestionnaire de paquet NPM

- L'intérêt de Node.js est sa **modularité**, pour pouvoir l'exploiter pleinnement il est nécessaire d'avoir un outil performant (ex. apt-get Debian).
- Le gestionnaire de paquets « **npm** » permet d'installer, rechercher et même diffuser les modules (addons de Node.js).
- Différence entre un **paquet** et un **module :** un **paquet** est un dossier contenant les ressources décrites dans un fichier **package.json.** Un paquet peut contenir des modules (JavaScript) mais également des exécutable (écrit en C/C++ et compilé).

Gestionnaire de paquet NPM

- Exemple d'utilisation

```
~# npm install <nom_dupaquet>
```

Par exemple pour installer le module « express »

```
~# npm install express
```

Ce module nous est utile pour simplifier le développement sur le protocole HTTP, il nous met à disposition des sevices proches de ce d'une « Servlet » en Java (request, response).

On verra son utilisation un peut plus loin dans ce cours.

INTRODUCTION Node.js

Partie 2

- **Les bases de Node.js,**
- **Require** : importer des modules,
- Serveur **HTTP** basique en Node.js,
- **Synchrone** et **Asynchrone**,
- l'API **HTTP** de Node.js

Les bases de Node.js

Dans cette partie on va réaliser une simple application Node.js pour comprendre comment on créé un serveur.

On va créer un dossier dans lequel on va créer un fichier nommé « app.js ». Dans ce fichier on va écrire :

```
1 var http = require('http');
2
3 var server = http.createServer(function(req, res) {
4      res.writeHead(200);
5      res.end('Salut tout le monde !');
6 });
7
8 server.listen(8080);
```

Pour lancer notre serveur `~# node ./app.js` et http://localhost:8080

Require : importer des modules

- Les modules sont les unités de base dans le développement d'application Node.
- Cela fait de Node un environnement de programmation orientée composants.
- Pour chaque module on a un objet global qui le décrit (module object) :
  - **id** : identifiant du module (en général le filename)
  - **filename** : chemin absolu du module
  - **loaded** : booléen qui précise si le module est chargé.
  - **parent** : le module qui a requis le module chargé
  - **children** : la liste des autres modules requis par le module.

Require : importer des modules

- La propriété « **exports** » contient la valeur qui est retournée lorsque le module est requis, créons un fichier nommé « **math.js** » pour créer un module :

```
 1 function gcd (a,b) {
 2     return b ? gcd(b, a % b) : a ;
 3 }
 4
 5 module.exports.gcd = gcd ;
 6
 7 function factorielle(n, acc) {
 8     acc || (acc = 1) ;
 9     return n ? factorielle(n-1, acc * n) : acc ;
10 }
11
12 module.exports.factorielle = factorielle ;
```

```
 1 var math = require('./math');
 2 var gcd_12_3 = math.gcd(12,3) ;
 3 console.log('gdc pour 12 et 3 : ' + gcd_12_3 ;
```

- Dans un fichier nommé « app2.js » on va écrire les 3 lignes de code ci-dessus.

Un exemple complet

- Modifiez le programme math.js pour que l'on puisse lire les données en entrée au clavier.

```
var math = require('./math');
var util = require('util');
process.stdin.setEncoding('utf8');
console.log('entrez 2 valeurs entières: X,Y <return>');

process.stdin.on('data', function(d) {
    console.log('received data:', util.inspect(d));
    var gcd = math.gcd(d.toString().split(',')[0],
                       d.toString().split(',')[1]) ;
    console.log('Pour '+ d.toString().split(',')[0]
                  +' et '+ d.toString().split(',')[1]
                  +' gcd : ' + gcd);
    console.log('vous avez saisie : [' + d.toString().trim() + ']');
    console.log('entrez 2 valeurs entières: X,Y <return>');
    if (d === 'quit\n') {
        done();
    }
});
function done() {  console.log('programme terminé !');
    process.exit();
}
```

Un serveur HTTP plus évolué

- Voici le code d'un serveur Node.js qui gère le format de retour via « ContentType » et le « pathname » :

```
var http = require('http');
var url = require('url');
var server = http.createServer(function(req, res) {
    var page = url.parse(req.url).pathname;
    console.log(page);
    res.writeHead(200, {"Content-Type": "text/plain"});
    if (page == '/') {
        res.write('Vous êtes à l\'accueil');
    }
    else if (page == '/sous-sol') {
        res.write('Vous êtes dans la cave à vins, ces bouteilles
                   sont à moi !');
    }
    else if (page == '/etage/1/chambre') {
        res.write('Hé ho, c\'est privé ici !');
    }
    res.end();
});

server.listen(8080);
```

Synchrone et Asynchrone

- Le principe des appels asynchrones en Nodejs et JavaScript, regardons la lecture d'un fichier une action d'Entrée/Sortie :

```
var readFile = require('fs').readFile;

readFile('app2.js', function(error, result) { //lecture Asynchrone
  if (!error) {
    console.log('contenu du fichier : \n' + result);
  } else { console.log('Erreur accès fichier : ', error) ;
});
```

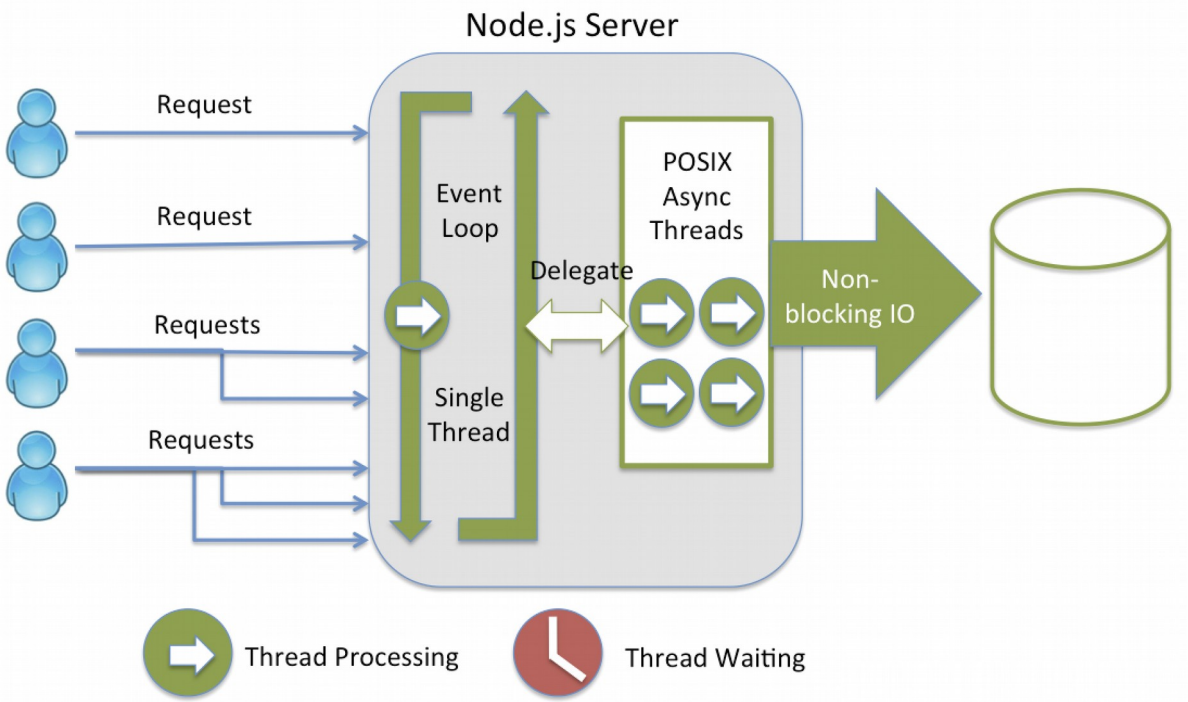
- Dans cet exemple ci-dessus, la fonction « readFile(...) » prend en paramètres :
  - **Filename** : le nom du fichier a lire,
  - **Callback** : Une fonction qui sera exécuté à la fin de l'opération d'Entrée/Sortie avec en params « error » et « result ». Elle est appelé fonction de Callback.
- Les limites en programmation sont les E/S pas le CPU !

Synchrone / Asynchrone (suite)

- Pour une partie des développeurs, un langage qui ne possède pas de « Threads », est considéré comme archaïque. Pourtant il est plus risqué et complexe de mettre en œuvre un programme multithreadé :
  - Il faut des verrous pour l'accès aux données.
  - Chaque bug peut induire une corruption des données.
  - Un mauvais verrouillage peut faire des interblocages.
- D'un point de vue performance, on constate d'après le tableau vu en début de cours sur la charge des serveur HTTP entre **Apache** et **Nginx**, **Lighthttp :**
  - Activier, désactiver des verrous est couteux en RAM / CPU

NB : En cas d'utilisation d'hébergement cloud préférez Node

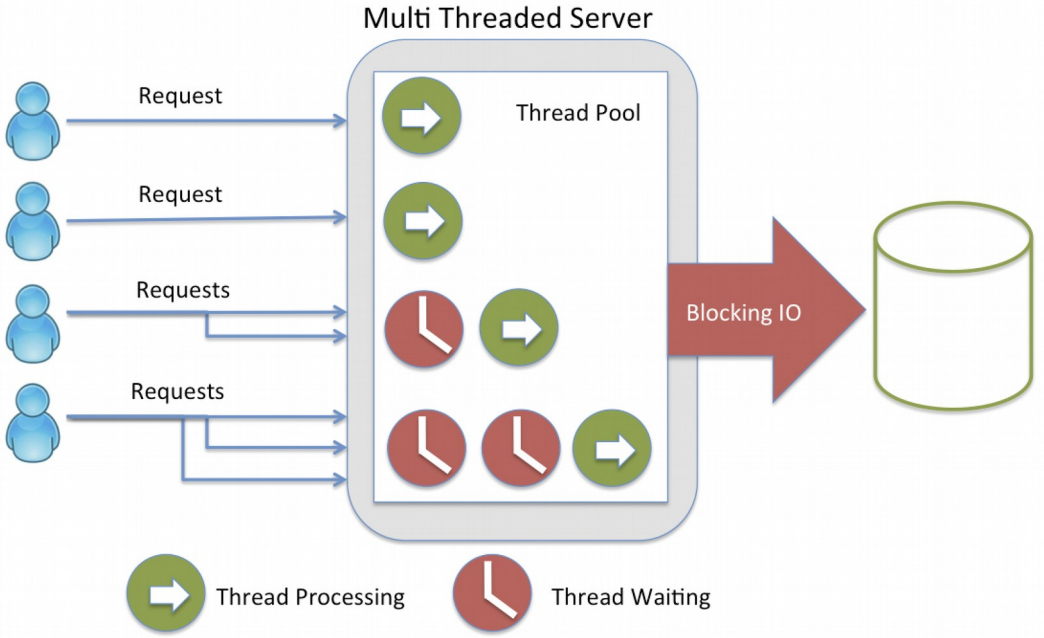
Synchrone (Thread Pool) et Asynchrone (Event Loop)



- **Questions de haute concurrence**

  Mais il y a une chose avec laquelle nous pouvons tous être d'accord: À des niveaux élevés de concurrence (en milliers de connexions) votre serveur a besoin d'utiliser le non-blocage asynchrone. J'aurais fini cette phrase avec E/S, mais le problème est que si une partie quelconque de vos serveurs bloque du code, alors vous allez avoir besoin d'un fil d'exécution (thread). Et à ces niveaux de concurrence, vous ne pouvez pas créer de threads pour chaque connexion. Donc, l'ensemble du code de votre application doit être non-bloquant et asynchrone, pas seulement la couche E/S. C'est là où Node excelle.

- Même si Java, Node ou autre chose peut gagner un comparatif de performance, aucun serveur d'application ne dispose de l'écosystème de non-blocage (event-loop) de Node.js aujourd'hui. Plus de 50 000 modules, tous écrit dans le style asynchrone, prêt à l'emploi. Il existe des exemples de code innombrables éparpillés sur le web ; leçons et tutoriels, tous utilisent le style asynchrone. Débogueurs, moniteurs, enregistreurs, managers de clusters, frameworks de tests et plus attendent votre code pour être asynchrone et non-bloquant.



Exercice Node.js

- Vous allez à partir des exemples précédents écrire un programme node.js qui réagit comme un serveur Web :
  - Il devrait sur un pathname particulier (/, sous-sol, etage/1/chambre) lire le fichier HTML correspondant
  - Pour chaque cas de pathname, votre serveur devra renvoyer dans la page du navigateur le contenu (result) de la lecture du fichier HTML.
  - On essaiera de ne pas dupliquer le code de lecture du fichier.
  - Réfléchir à des améliorations permettant de rendre atomique le développement des traitements.