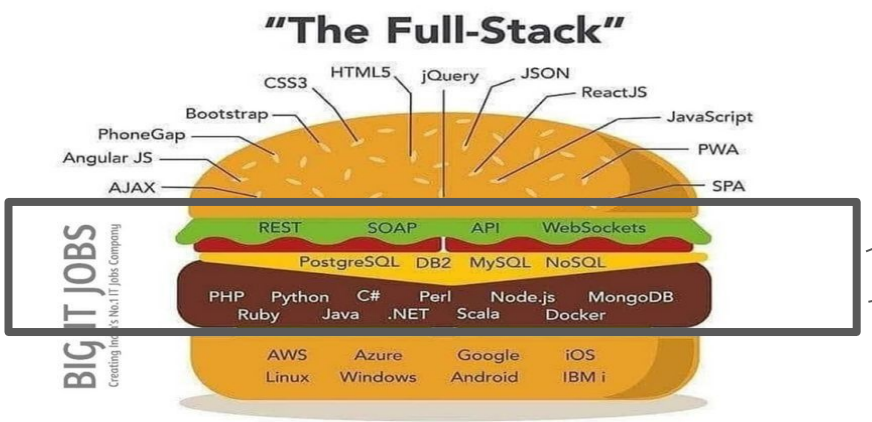
Dev Web

côté serveur

S3 - R3.01 - 2023/2024

Qu'allons nous faire?

- Dev Web back-end (Joseph Azar)
  - 8 semaines
- Dev Web front-end (Stephane Domas)



API REST, Serveur Web

Javascript, NodeJS

JSON

Quels outils/langages allons-nous utiliser ?

Semaine 1

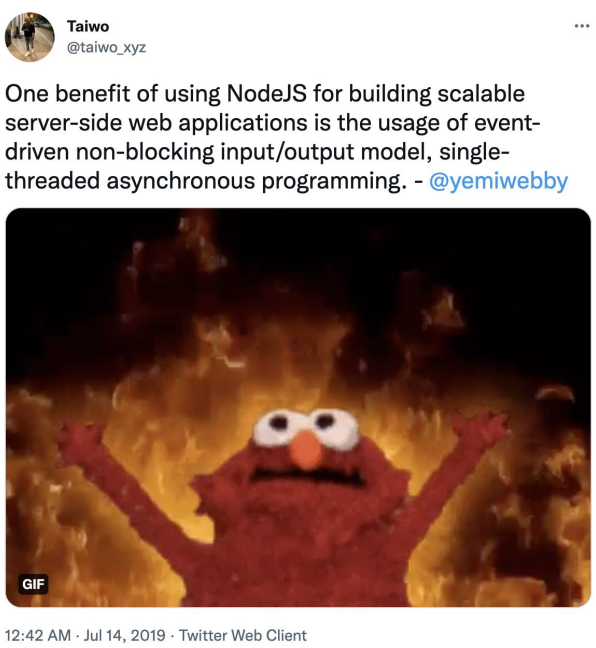
- Qu'est-ce que Node.js ?
- Fonctionnalités
- Pourquoi choisir Node.js ?
- Programmation événementielle asynchrone & Callbacks
- Système de fichiers NodeJS

Qu'est-ce que Node.js ?

- Il s'agit d'un environnement d'exécution JavaScript open source asynchrone piloté par les événements.
- Il est construit sur le moteur JavaScript de Chrome (moteur V8).
- Il permet aux développeurs d'utiliser Javascript côté serveur.

Qu'est-ce que Node.js ?

Asynchronous event-driven JavaScript

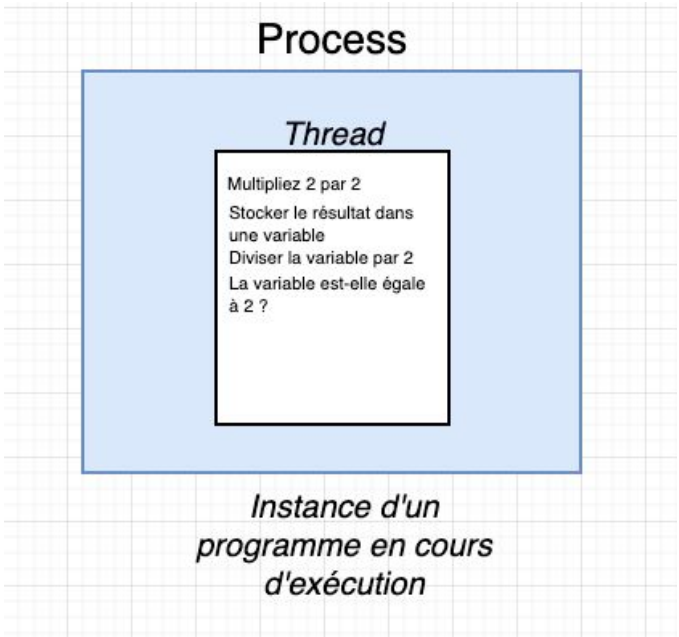


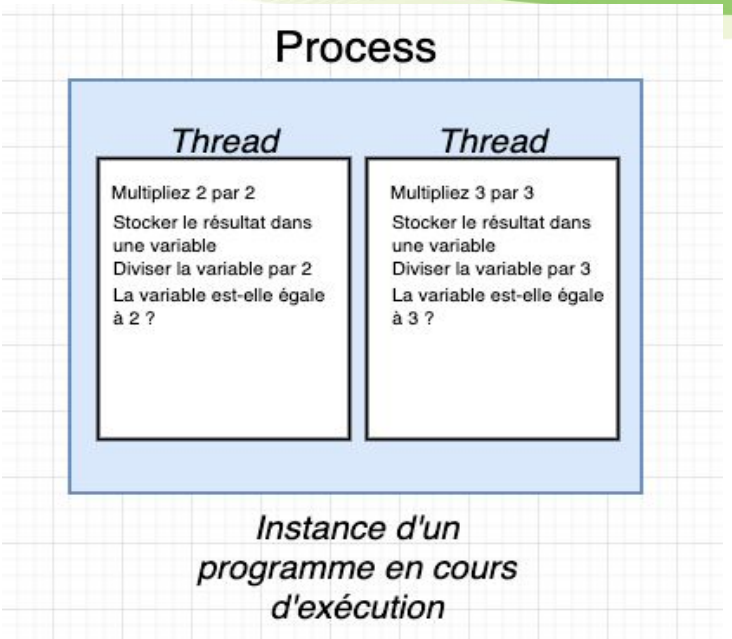
“L'un des avantages de l'utilisation de NodeJS pour la création d'applications Web évolutives côté serveur est l'utilisation d'un modèle d'entrée/sortie non bloquant piloté par les événements et d'une programmation asynchrone à un seul thread.”

“L'élimination des processus de blocage grâce à l'utilisation d'E/S asynchrones pilotées par les événements est le principal principe organisationnel de Node.” – Mastering Node.js - Second Edition

Qu'est-ce que Node.js ?

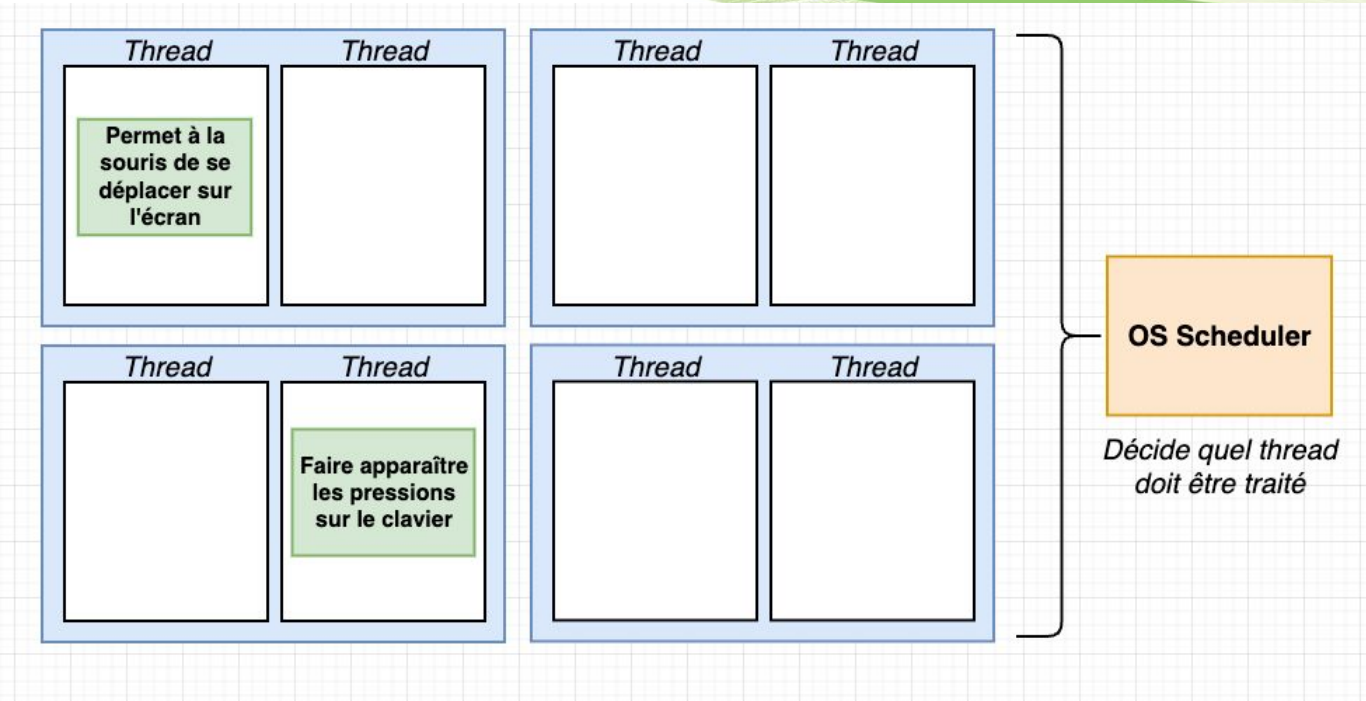
Threads





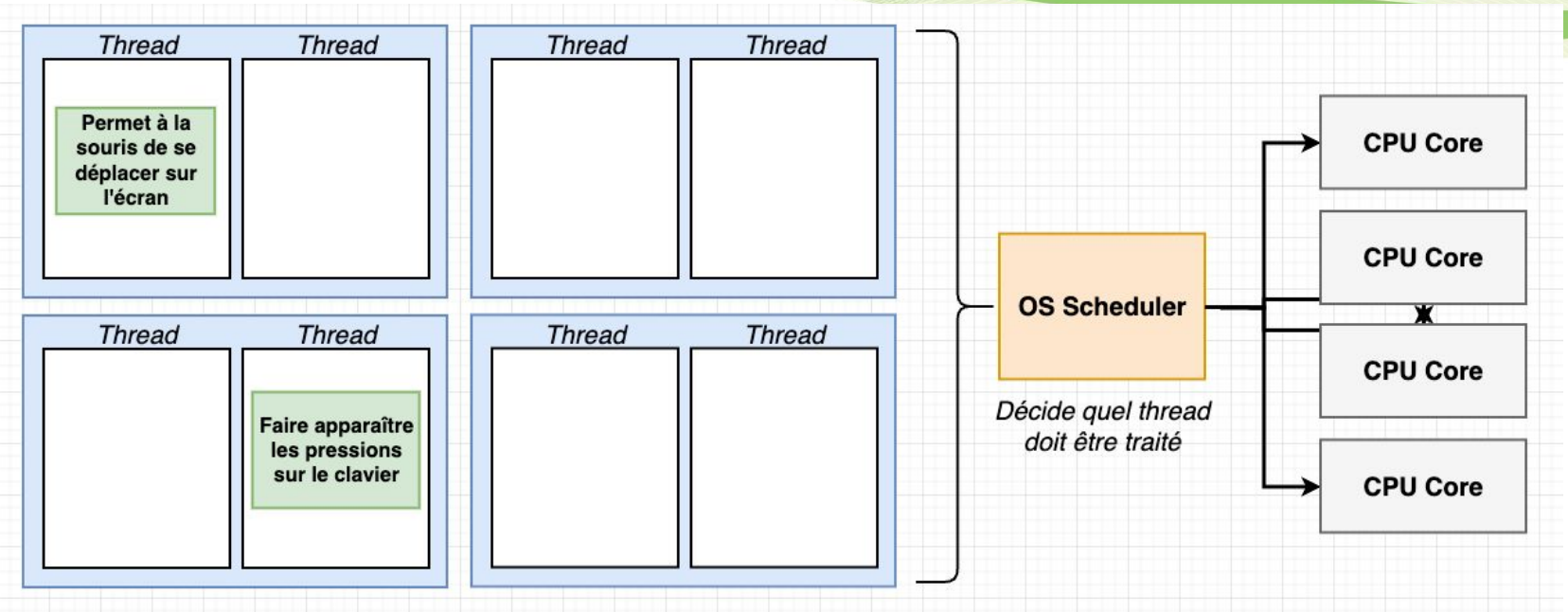
Qu'est-ce que Node.js ?

Threads



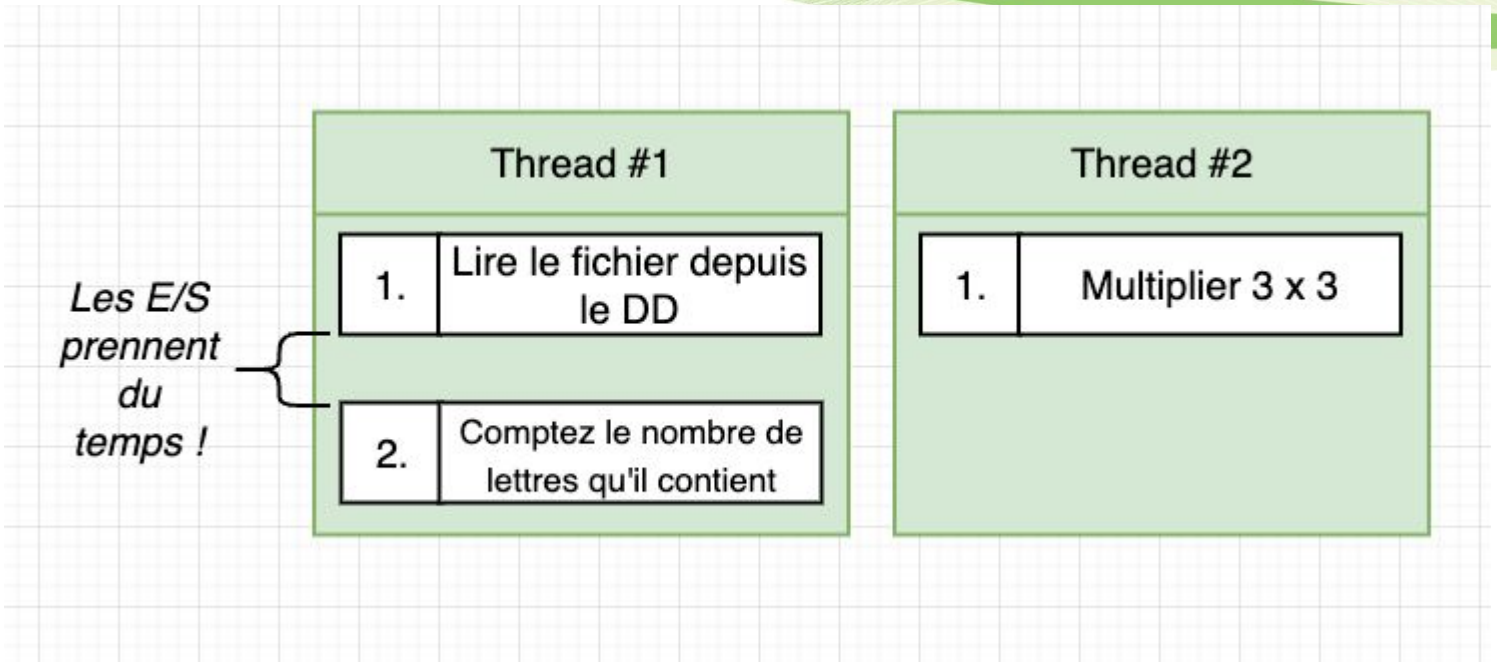
Qu'est-ce que Node.js ?

Threads



Qu'est-ce que Node.js ?

Threads



Qu'est-ce que Node.js ?

Threads

- Les threads sont des unités d'instruction en attente d'exécution par le CPU.
- La décision de l'ordre d'exécution de ces threads est appelée ordonnancement (**Scheduling**).
- L'ordonnancement est géré par le système d'exploitation.
- Pour améliorer la vitesse de traitement des threads, on peut :
  - soit ajouter plus de cœurs CPU à notre machine,
  - soit permettre à notre ordonnanceur OS de détecter de longues pauses dues à des opérations d'entrée/sortie coûteuses.

Qu'est-ce que Node.js ?

Asynchronous event-driven JavaScript

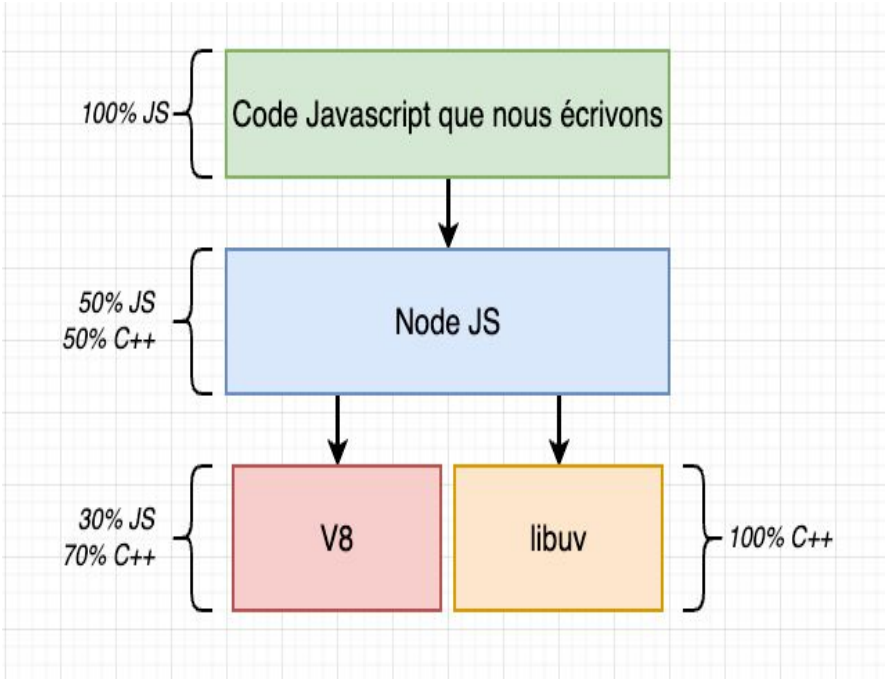
Threads versus événements utilisant des rappels (Callbacks)

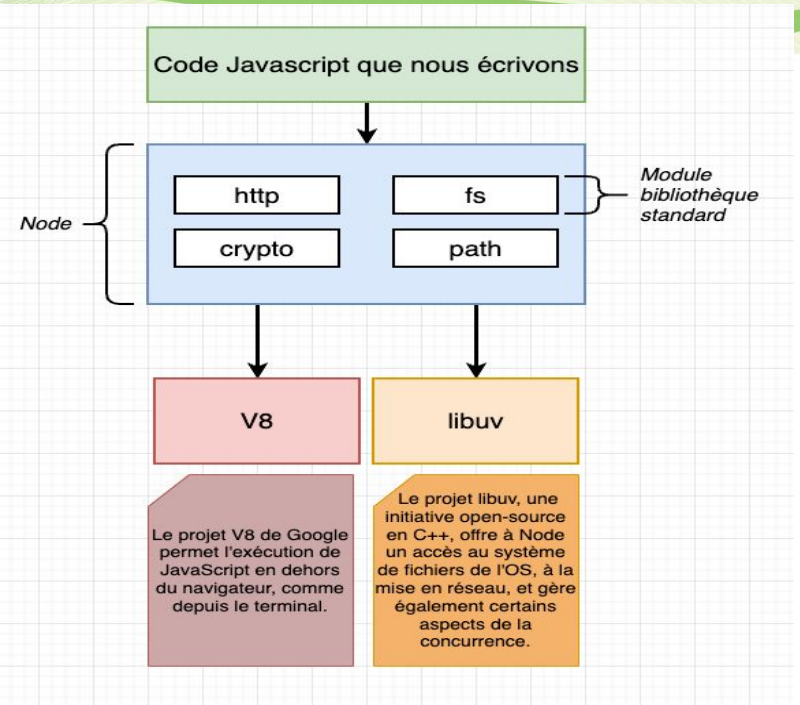
```
request = readRequest(socket);
reply = processRequest(request);
sendReply(socket, reply);
moreWork(); // will run after sendreply()
```

```
readRequest(socket, function(request) {
 processRequest(request,
    function (reply) {
        sendReply(socket, reply);
    });
});
moreWork(); // will run before readRequest
```

Qu'est-ce que Node.js ?

Éléments internes de NodeJS





Qu'est-ce que Node.js ?

Moteur Javascript V8

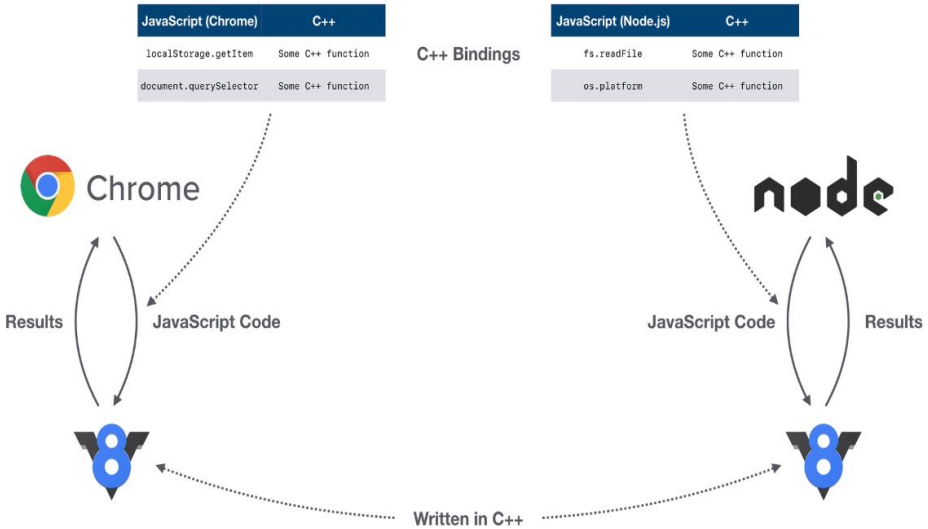
La bibliothèque V8 fournit à Node.js un moteur JavaScript (un programme qui convertit le code Javascript en code machine ou de niveau inférieur que les microprocesseurs peuvent comprendre), que Node.js contrôle via l'API V8 C++. V8 est maintenu par Google, pour une utilisation dans Chrome.

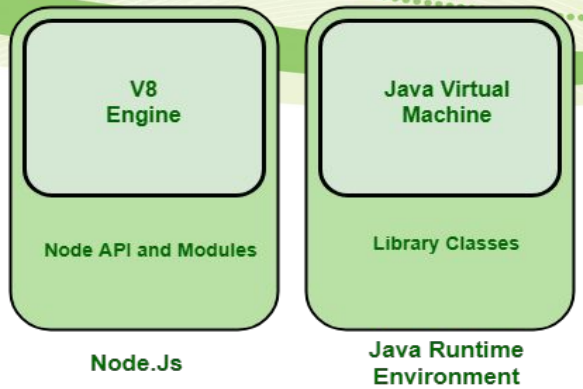
Le moteur Chrome V8 :

- Le moteur V8 est écrit en C++ et utilisé dans Chrome et Nodejs.
- Il implémente ECMAScript comme spécifié dans ECMA-262.
- Le moteur V8 peut fonctionner de manière autonome, nous pouvons l'intégrer à notre propre programme C++.

Qu'est-ce que Node.js ?

Moteur Javascript V8





- V8 est essentiellement une bibliothèque C++ autonome.
- Vous pouvez intégrer V8 dans votre programme C++ et exécuter un programme JavaScript à partir de celui-ci.

Disons que nous voulons ajouter la possibilité d'avoir des déclarations comme print('hello world') en plus de console.log('Hello World') dans notre code JavaScript. Avec le moteur V8, qui est déjà open source, nous pouvons ajouter notre propre implémentation C++ de la fonction d'impression.

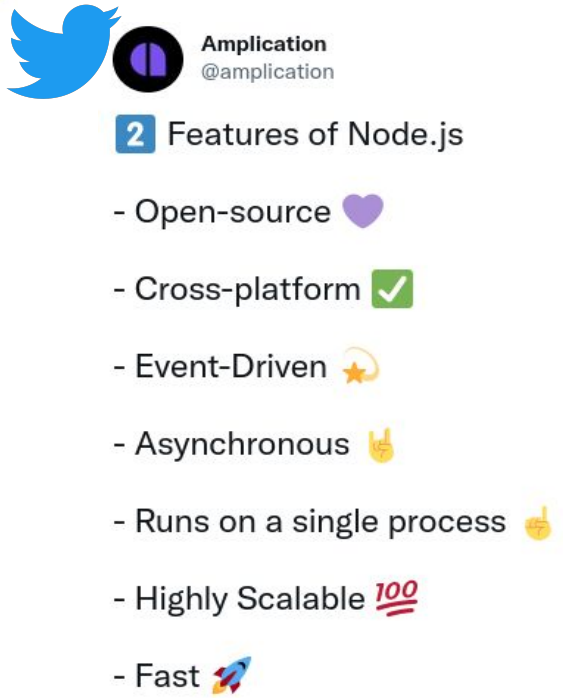
Qu'est-ce que Node.js ?

Javascript côté serveur

“Lorsqu'il a conçu Node, JavaScript n'était pas le choix de langue d'origine de Ryan Dahl. Pourtant, lors de l'exploration, il a trouvé un bon langage moderne sans opinions sur les **flux**, le **système de fichiers**, la **gestion des objets binaires**, les **processus**, la **mise en réseau** et d'autres capacités que l'on s'attendrait à voir exister dans un langage système. JavaScript, strictement limité au navigateur, n'avait pas implémenté, ces fonctionnalités.”

— *Mastering Node.js - Second Edition*

Fonctionnalités



Les capacités de Node.js

Node.js est une plate-forme pour écrire des applications JavaScript en dehors des navigateurs Web. Ce n'est pas l'environnement JavaScript que nous connaissons dans les navigateurs Web ! Bien que Node.js exécute le même langage JavaScript que nous utilisons dans les navigateurs, il ne possède pas certaines des fonctionnalités associées au navigateur. Par exemple, il n'y a pas de DOM HTML intégré dans Node.js.

Au-delà de sa capacité native à exécuter JavaScript, les modules intégrés offrent les fonctionnalités suivantes :

- Outils de ligne de commande (dans le style de script shell)
- Un programme de type terminal interactif , c'est-à-dire une **boucle de lecture-évaluation-impression (REPL)**
- Excellentes fonctions de contrôle de processus pour superviser les processus enfants
- Un objet tampon pour traiter les données binaires
- Sockets TCP ou UDP avec rappels complets basés sur les événements
- Recherche DNS
- Un serveur client HTTP, HTTPS et HTTP/2 superposé à l'accès au système de fichiers de la bibliothèque TCP
- Prise en charge intégrée des tests unitaires rudimentaires via des assertions

– Node.js Web Development, Fifth Edition

Pourquoi choisir Node.js ?

Node.js, un langage utile et efficace

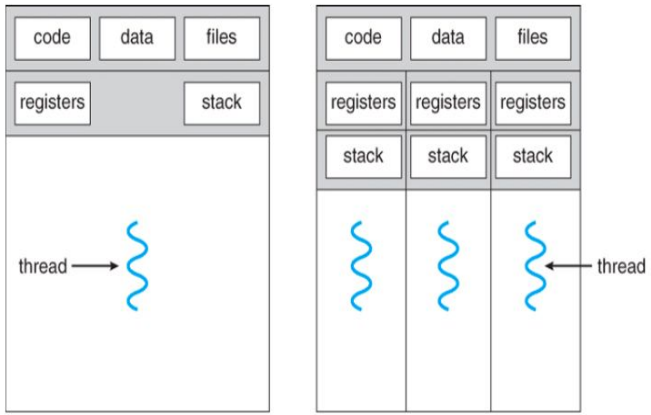
- Le moteur JavaScript V8 développé par Google, qui permet d'exécuter du code JavaScript à l'intérieur de Google Chrome et, grâce à Node, directement sur le serveur
- Open-Source avec une communauté de développeurs active
- Conçu pour une programmation efficace et asynchrone côté serveur
- Une boucle d'événements, appelée aussi EVENT LOOP NodeJS, permettant d'exécuter plusieurs opérations simultanées de façon asynchrone et non bloquante en tirant profit des multiples fils d'exécution (multithreading) des noyaux des processeurs modernes

Pourquoi choisir Node.js ?

Node est très adapté pour les serveurs web

Node JS est single-threaded: il ne s'exécute que sur une seule instance (un thread ou fil d'exécution) du système d'exploitation, à l'inverse d'un serveur fonctionnant en PHP (avec Apache HTTP Server par exemple) qui est, lui, multi-threaded

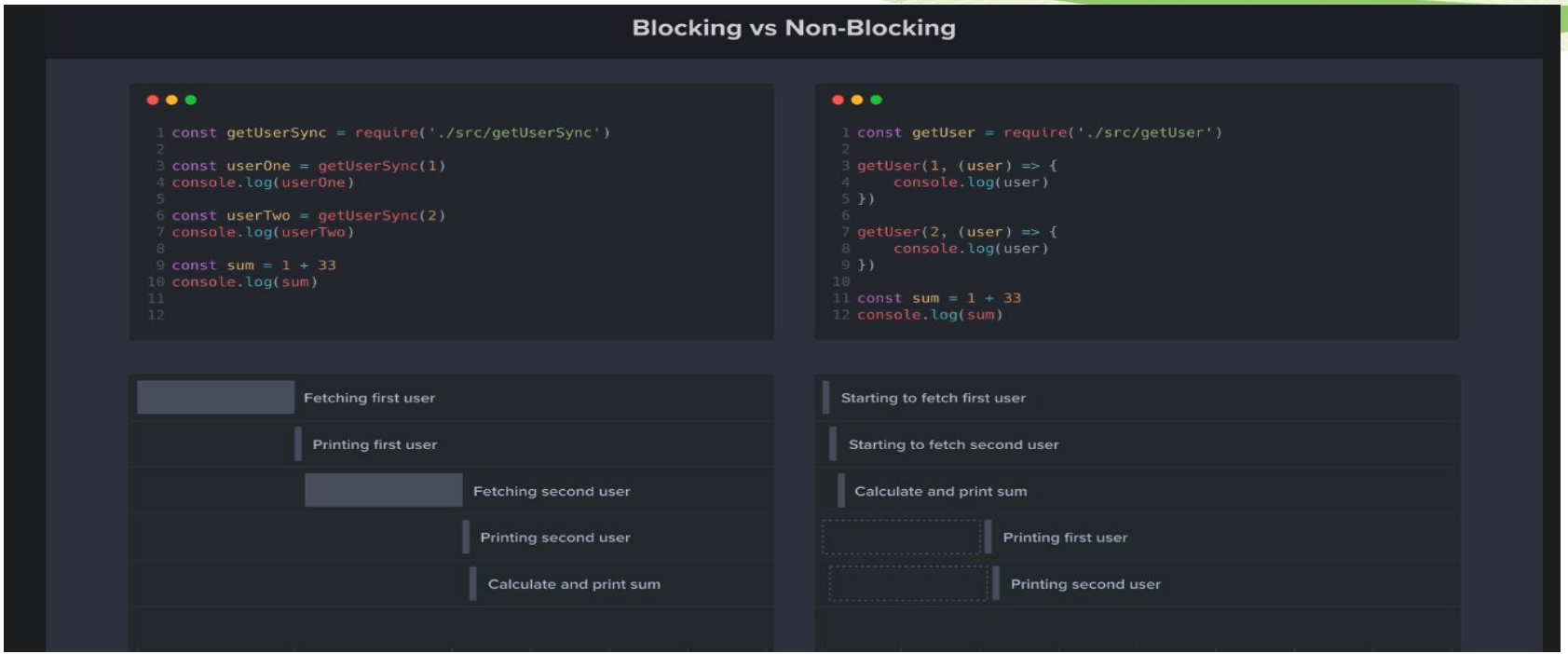
Cette caractéristique de Node lui permet de gérer les requêtes de façon non bloquantes. Là où un serveur PHP mobilise un thread pour réceptionner une requête et attendre qu'elle passe à travers tout le code pour retourner une réponse au client, le serveur Node va gérer chaque requête de manière asynchrone au sein d'un seul et unique thread. La requête est ainsi réceptionnée puis envoyée dans la callback queue où le thread de Node sert les réponses une par une



Différence entre un process mono thread et un process multi threds

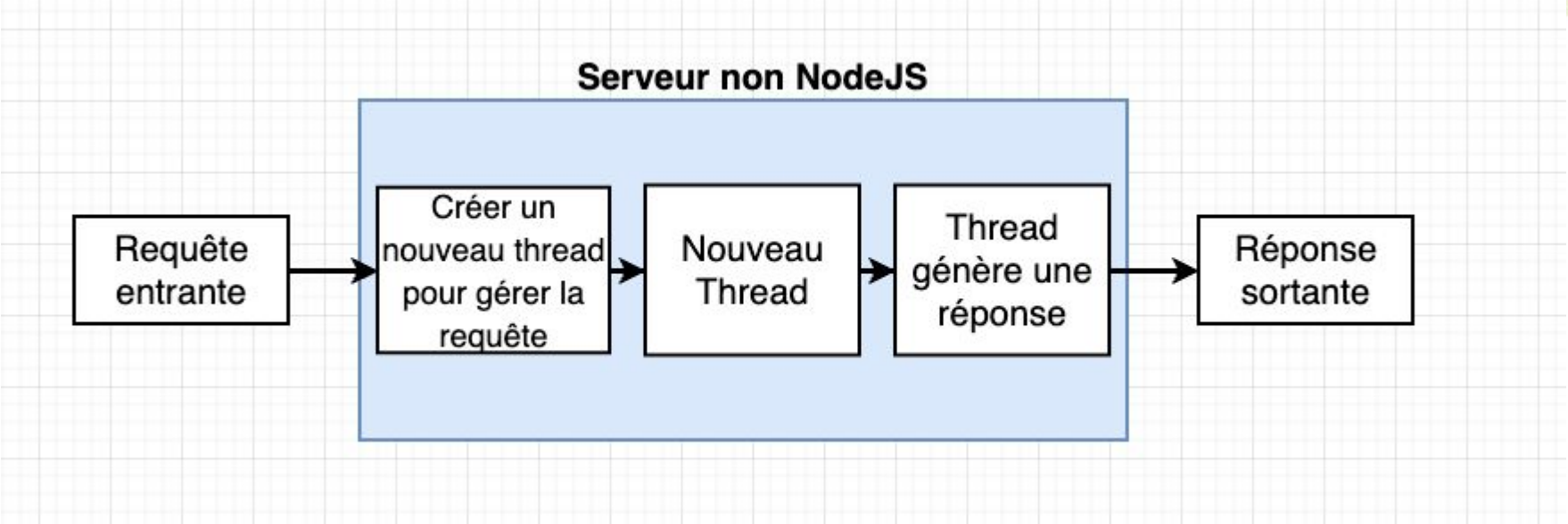
Pourquoi choisir Node.js ?

Bloquant vs Non bloquant



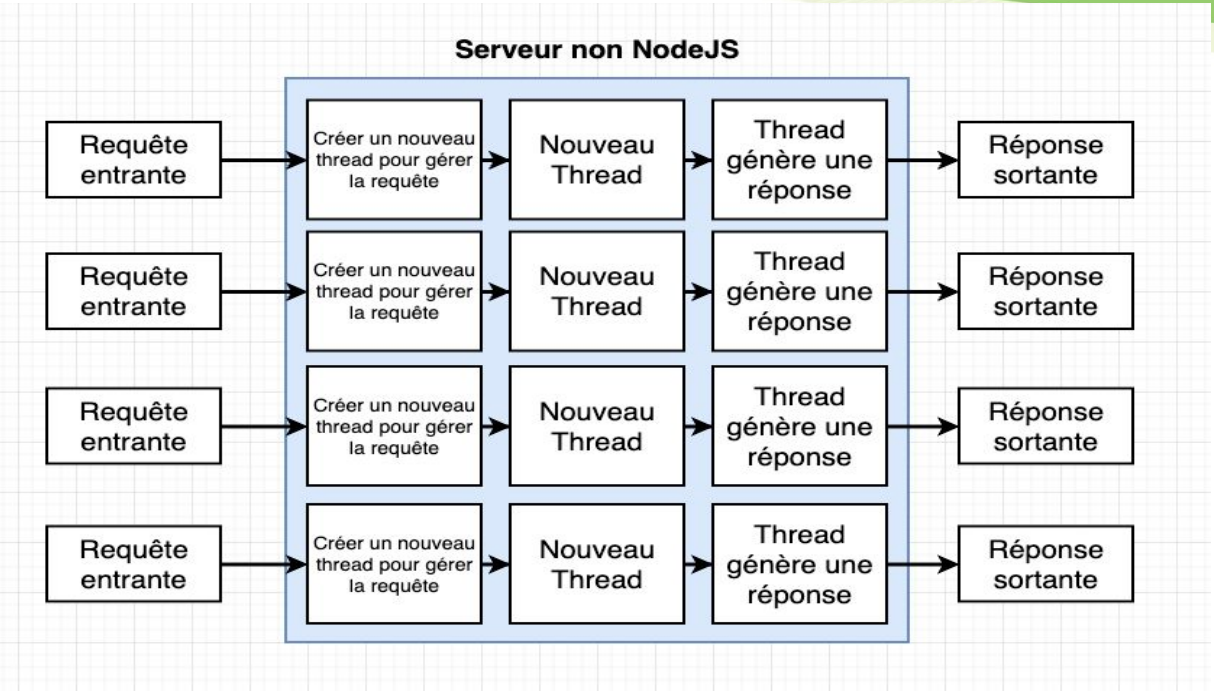
Pourquoi choisir Node.js ?

Bloquant vs Non bloquant



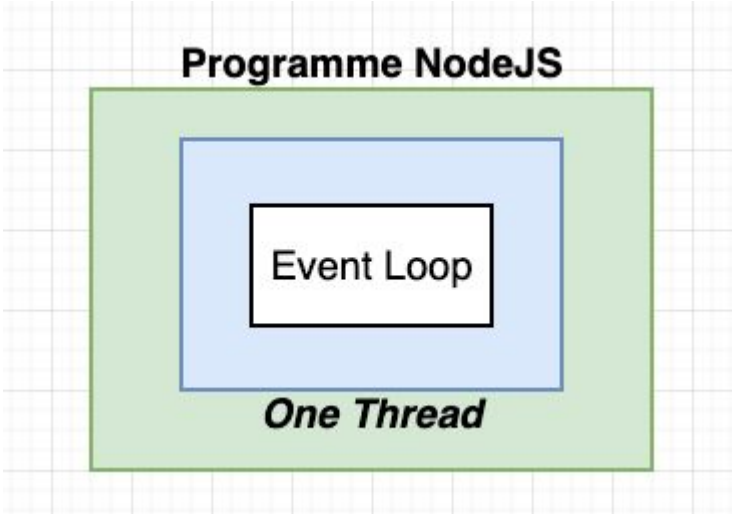
Pourquoi choisir Node.js ?

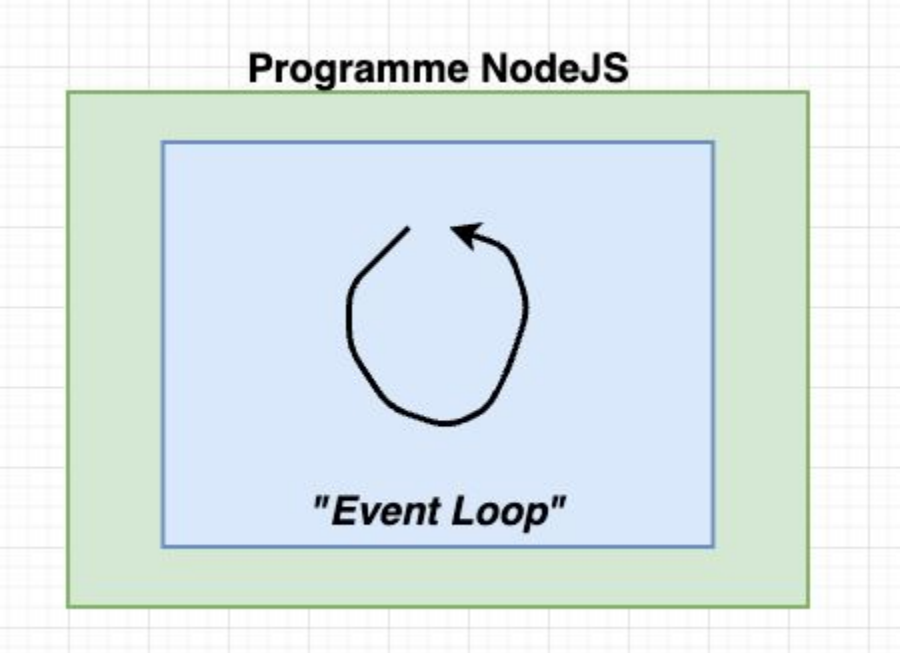
Bloquant vs Non bloquant



Pourquoi choisir Node.js ?

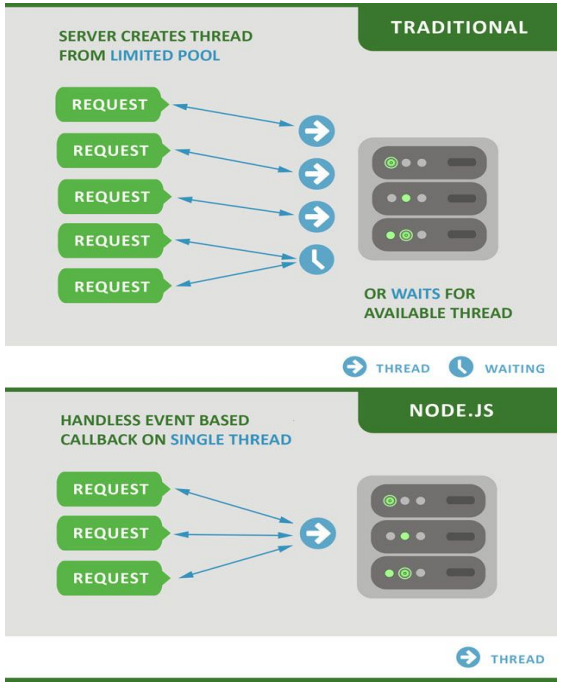
Bloquant vs Non bloquant

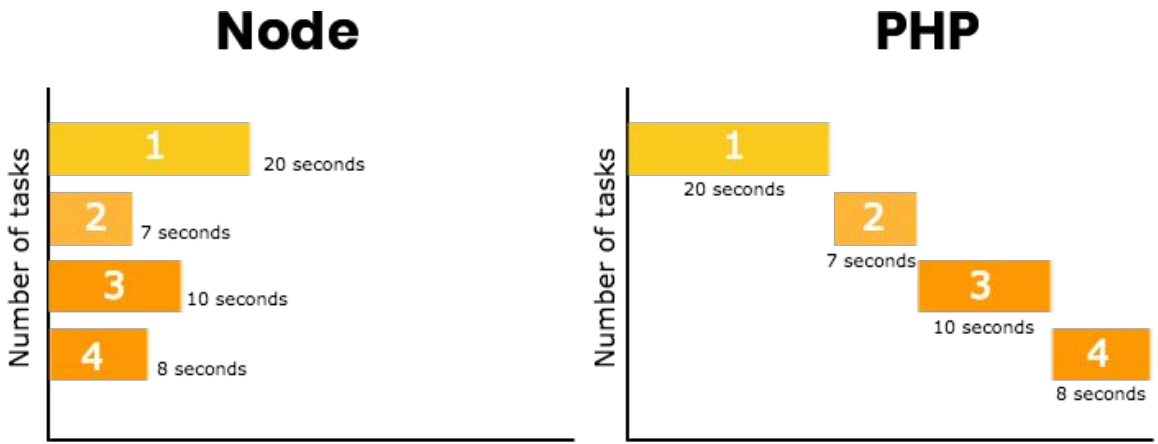




Pourquoi choisir Node.js ?

Bloquant vs Non bloquant





Pourquoi choisir Node.js ?

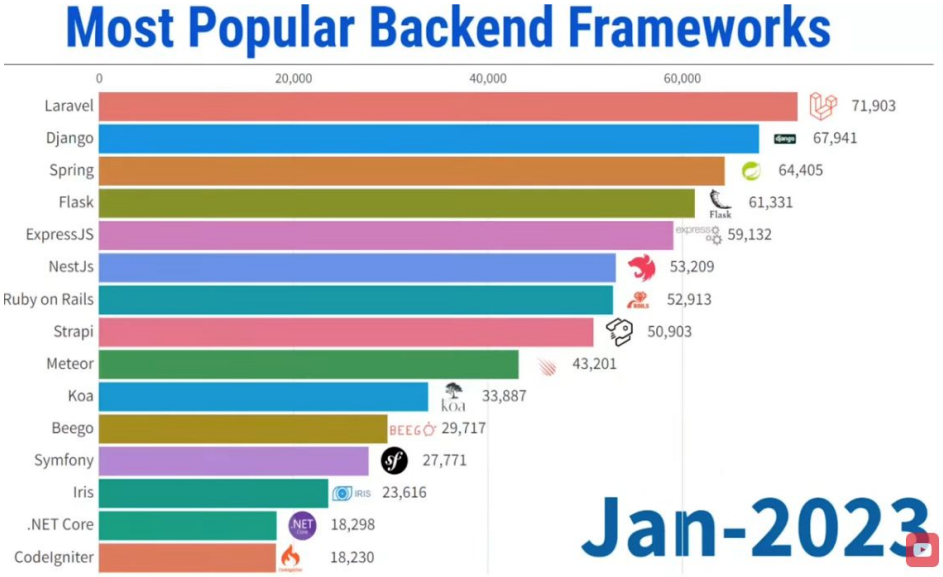
Node est basé sur JavaScript

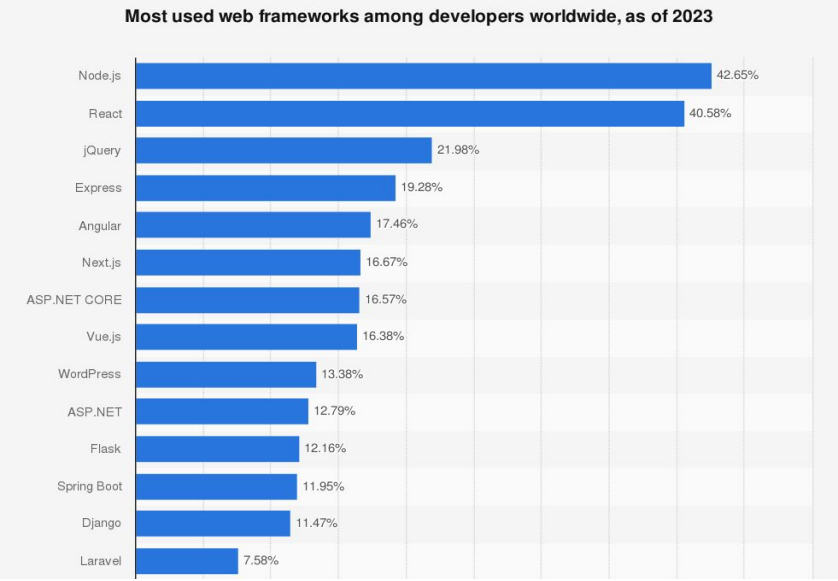
Créé en 1995 par Netscape et longtemps cantonné à réaliser de petites animatio
les sites web, JavaScript est aujourd'hui le langage de développement web le plus populaire au monde. Cela peut s'illustrer par plusieurs statistiques dont les suivantes :

- JavaScript est le langage le plus utilisé par les répondants au Stack Overflow Developers Survey de 2019.
- C'est également le langage le plus présent dans les contributions GitHub depuis au moins cinq ans.
- L'utilisation de JavaScript comme langage de programmation côté serveur permet également d'abolir les barrières entre front-end et back-end, permettant de monter une équipe fullstack plus facilement.
- Enfin, Javascript est le langage utilisé par une écrasante majorité de sites web pour le front-end mais est également de plus en plus populaire pour le back-end.

Pourquoi choisir Node.js ?

Le marché de l'emploi est en demande





Référence : https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2023/

Qui utilise Node.js?

- Paypal – De nombreux sites au sein de Paypal ont également commencé la transition vers Node.js.
- LinkedIn - LinkedIn utilise Node.js pour alimenter ses serveurs mobiles, qui alimentent les produits iPhone, Android et Web mobile.
- Mozilla a implémenté Node.js pour prendre en charge les API de navigateur qui compte un demi-milliard d'installations.
- Ebay héberge son service d'API HTTP à l'aide de Node.js .

Différents types de programme Node

- Des applications Web
- Outils de ligne de commande
- Applications de bureau
- Applications IoT (Node-RED)

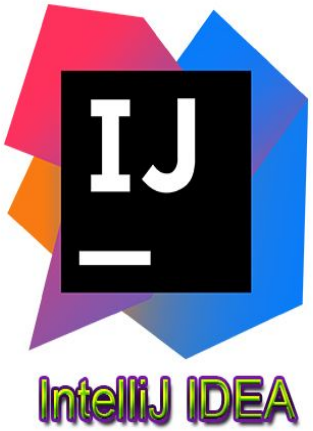
Express

yargs

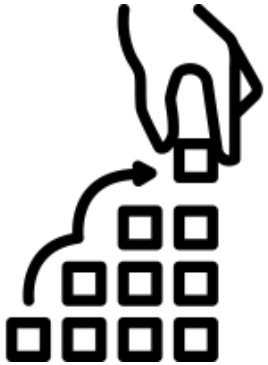
ELECTRON

Node-RED

Environnement de développement



Fondamentaux de Node JS



Qu'est-ce que la fonction de rappel (Callback) ? 📞🔙

En JavaScript, les fonctions sont généralement exécutées de manière séquentielle. Une fonction s'exécute dans l'ordre dans lequel vous l'appelez, et non dans l'ordre dans lequel vous la définissez. Exemple 👉

JavaScript

```
//functions
function cat(){
console.log("This is a cat");
}
function dog(){
console.log("This is a dog.");
}

//function call
cat();
dog();

//output will be
This is a cat.
This is a dog.
```

Qu'est-ce que la fonction de rappel (Callback) ? 📞🔙

Par exemple, si nous utilisons la fonction '**setTimeout()**' sur la fonction **pet1**, **setTimeout()** est une fonction qui exécute une fonction après un temps donné. En attendant, JavaScript exécute d'autres fonctions, quelle que soit la séquence. Exemple 👉

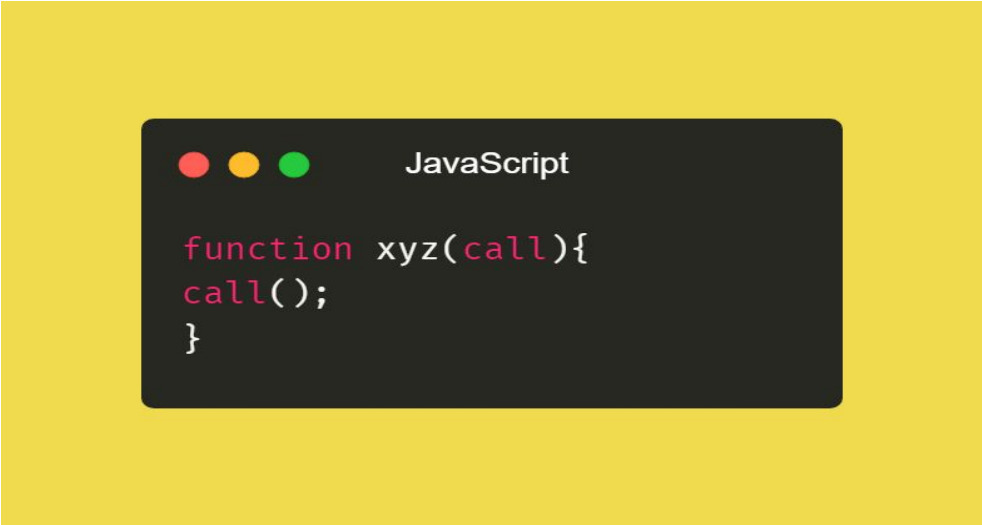
JavaScript

```
function pet1(){
console.log("This is a cat");
}
function pet2(){
console.log("This is a dog.");
}
setTimeout(pet1, 5000);
pet2();

//output will be
This is a dog.
This is a cat.
```

Qu'est-ce que la fonction de rappel (Callback) ? 📞🔙

La fonction que nous passons en argument à une autre fonction s'appelle une fonction de rappel. Exemple 👇🏻



Qu'est-ce que la fonction de rappel (Callback) ? 📞🔙

Contrôle de séquence

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2) {
  let sum = num1 + num2;
  return sum;
}

let result = myCalculator(5, 5);
myDisplayer(result);
```

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2) {
  let sum = num1 + num2;
  myDisplayer(sum);
}

myCalculator(5, 5);
```

Qu'est-ce que la fonction de rappel (Callback) ? 📞🔙

Contrôle de séquence

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2) {
  let sum = num1 + num2;
  return sum;
}

let result = myCalculator(5, 5);
myDisplayer(result);
```

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2) {
  let sum = num1 + num2;
  myDisplayer(sum);
}

myCalculator(5, 5);
```

Le problème avec le premier exemple ci-dessus, c'est que vous devez appeler deux fonctions pour afficher le résultat.
Le problème avec le deuxième exemple, c'est que vous ne pouvez pas empêcher la fonction myCalculator d'afficher le résultat.

Qu'est-ce que la fonction de rappel (Callback) ? 📞🔙

Contrôle de séquence avec 📞🔙

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2, Callback) {
  let sum = num1 + num2;
  Callback(sum);
}

myCalculator(5, 5, myDisplayer);
```

Dans l'exemple ci-dessus, **myDisplayer** est le nom d'une fonction. Il est passé à **myCalculator()** en tant qu'argument.

Lorsque vous passez une fonction en argument, n'oubliez pas de ne pas utiliser de parenthèses.

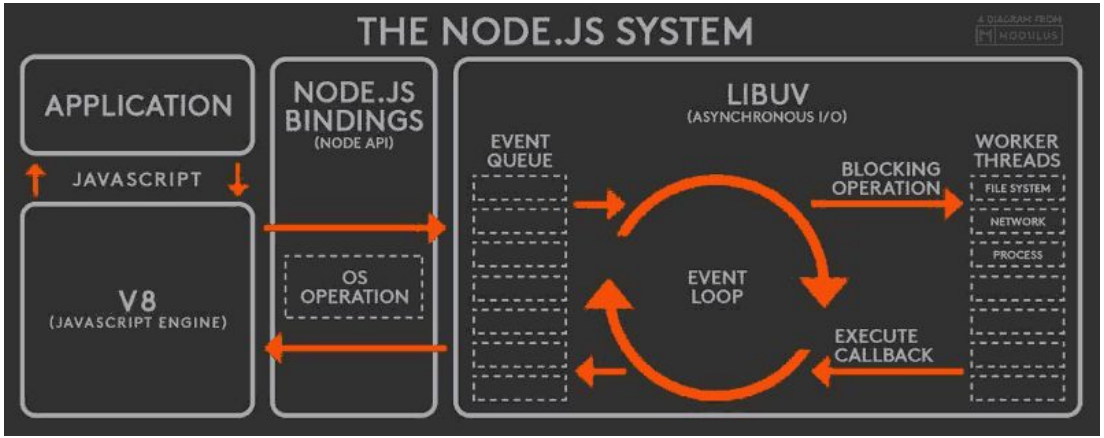
Correcte : myCalculator(5, 5, myDisplayer) ;
Faux : myCalculator(5, 5, myDisplayer());

Là où les rappels brillent vraiment, ce sont dans les fonctions asynchrones, où une fonction doit attendre une autre fonction (comme attendre le chargement d'un fichier).

Vue d'ensemble sur NodeJS Event Loop

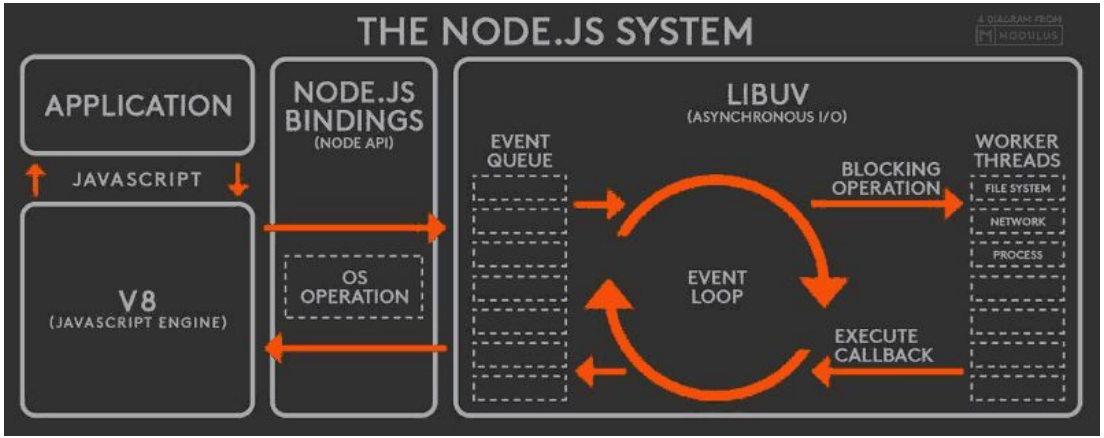
Q: Qu'est-ce que la boucle d'événement **Event Loop** ?

R: La boucle d'événements est ce qui permet à Node.js d'effectuer des opérations d'E/S non bloquantes - malgré le fait que JavaScript est à thread unique - en déchargeant les opérations sur le noyau du système chaque fois que possible.



Vue d'ensemble sur NodeJS Event Loop

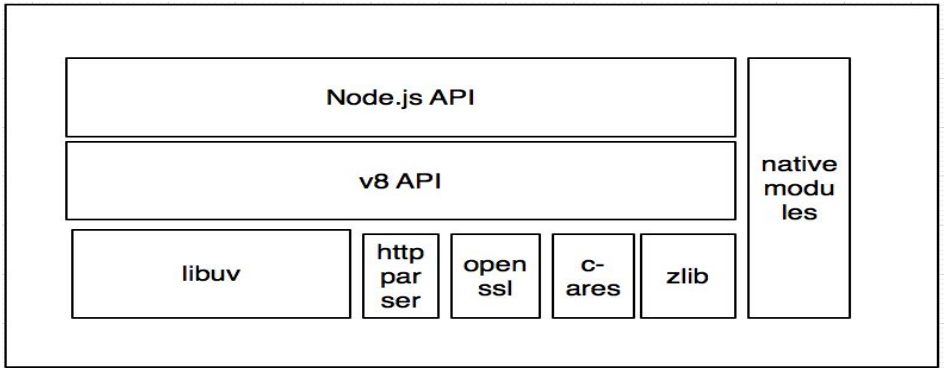
Chaque E / S nécessite un rappel - une fois qu'elles sont terminées, elles sont poussées dans la boucle d'événements pour exécution. Étant donné que la plupart des noyaux modernes sont multithreads, ils peuvent gérer plusieurs opérations exécutées en arrière-plan. Lorsque l'une de ces opérations est terminée, le noyau indique à Node.js que le rappel approprié peut être ajouté à la file d'attente d'interrogation pour être éventuellement exécuté.

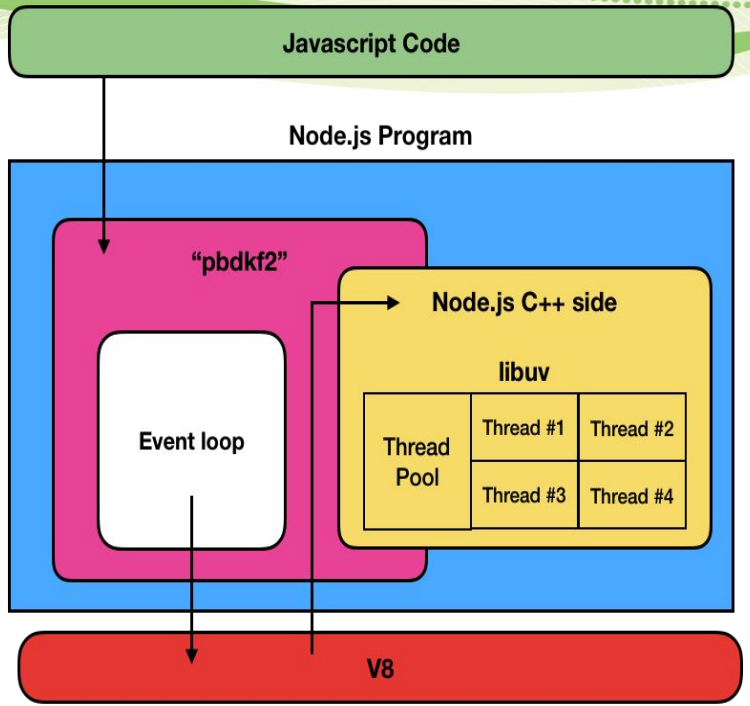


Vue d'ensemble sur NodeJS Event Loop

Q: Qu'est-ce que **libuv** ?

R: libuv est une bibliothèque C utilisée pour résumer les opérations d'E/S non bloquantes en une interface cohérente sur toutes les plates-formes prises en charge. libuv fournit des mécanismes pour gérer le système de fichiers, le DNS, le réseau, les processus enfants, la gestion du signal, et le streaming. Elle comprend également un pool de threads pour décharger le travail de certaines choses qui ne peuvent pas être effectuées de manière asynchrone au niveau du système d'exploitation.

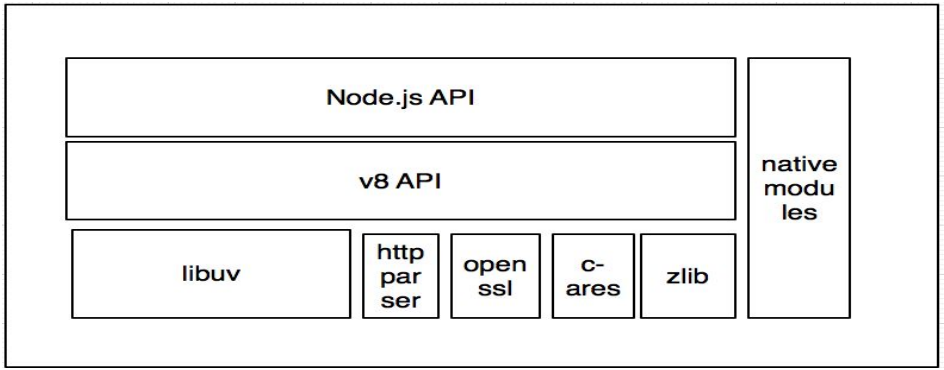


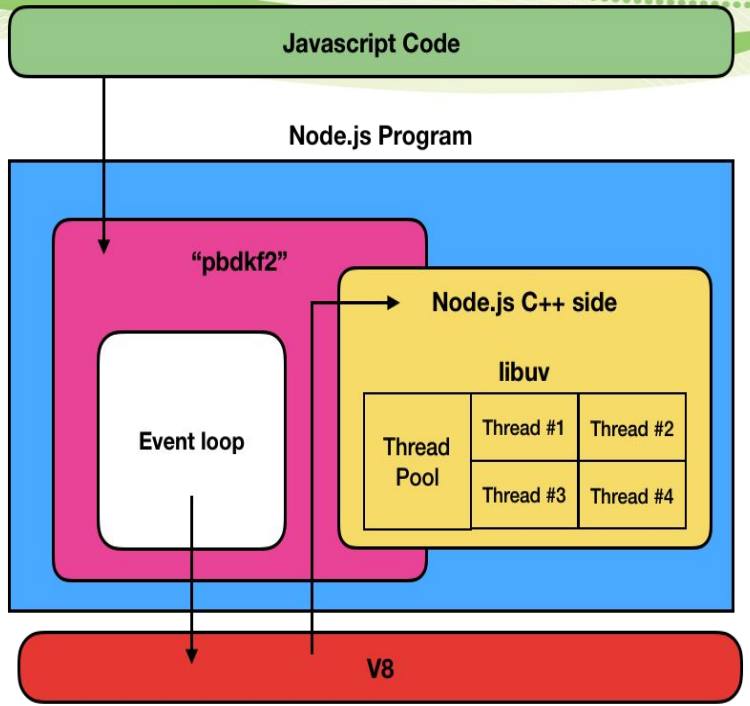


Vue d'ensemble sur NodeJS Event Loop

Q: Quelle est la différence entre **libuv** et **v8** ?

R: **V8** est responsable de l'exécution du code JavaScript en dehors du navigateur, tandis que **libuv** est responsable de l'accès au système d'exploitation et au système de fichiers sous-jacents de notre système et résout également une certaine mesure de la concurrence.

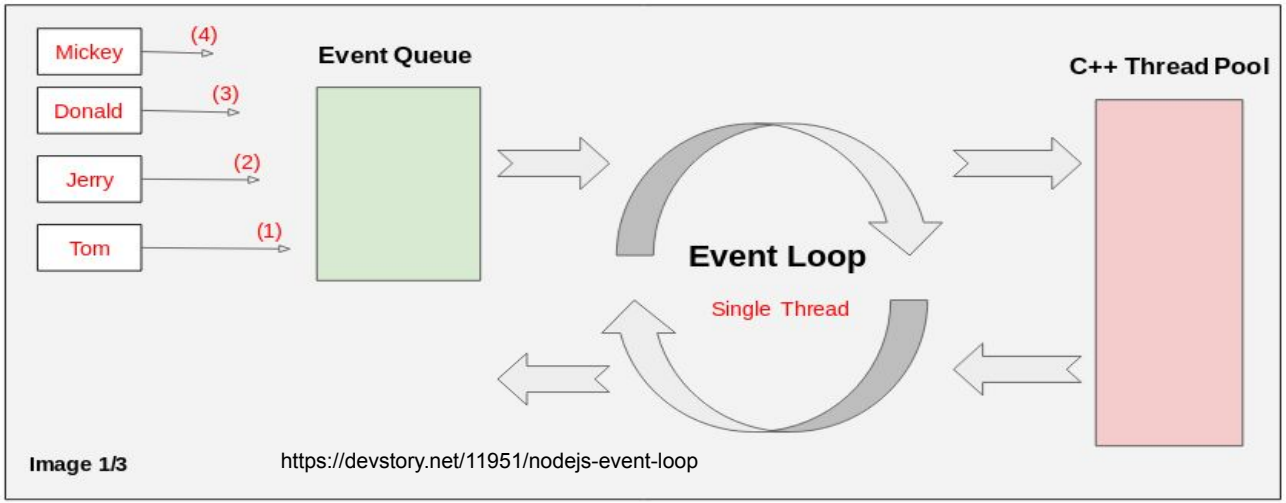




Vue d'ensemble sur NodeJS Event Loop

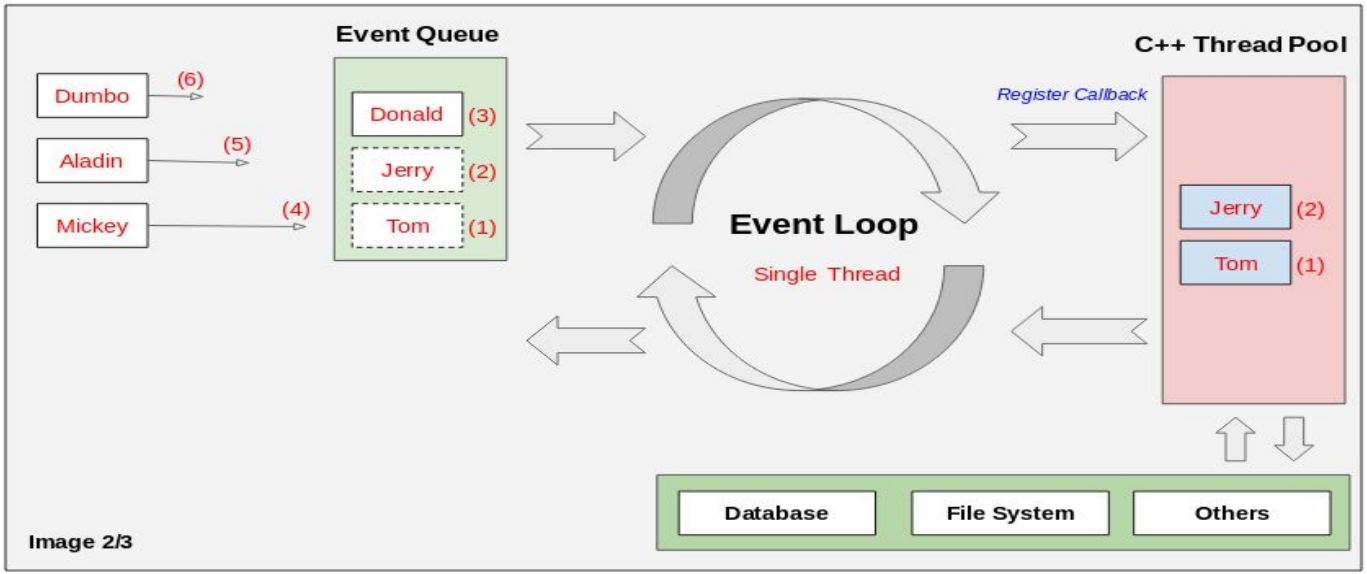
NodeJS est une application de thread unique (Single Thread), qui fonctionne sur une plateforme écrite en C++. Cette plateforme utilise le multi-thread pour effectuer des tâches en même temps.

La figure suivante illustre des demandes (request) envoyés à part des utilisateurs au serveur de NodeJS.



Vue d'ensemble sur NodeJS Event Loop

Chaque demande (request) de l'utilisateur est traitée comme un événement (event) par le NodeJS. Elles sont placées dans un Event Queue (La liste des événements). Le NodeJS utilise le principe FIFO (First In First Out), ce qui signifie que les premières demandes arrivant sera traités avant tout.

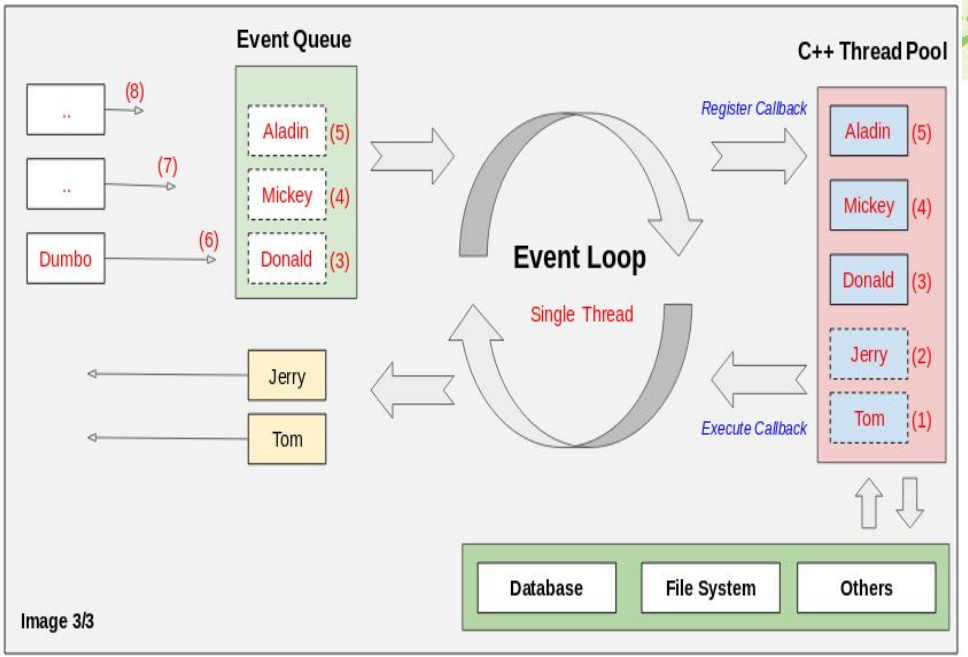


Vue d'ensemble sur NodeJS Event Loop

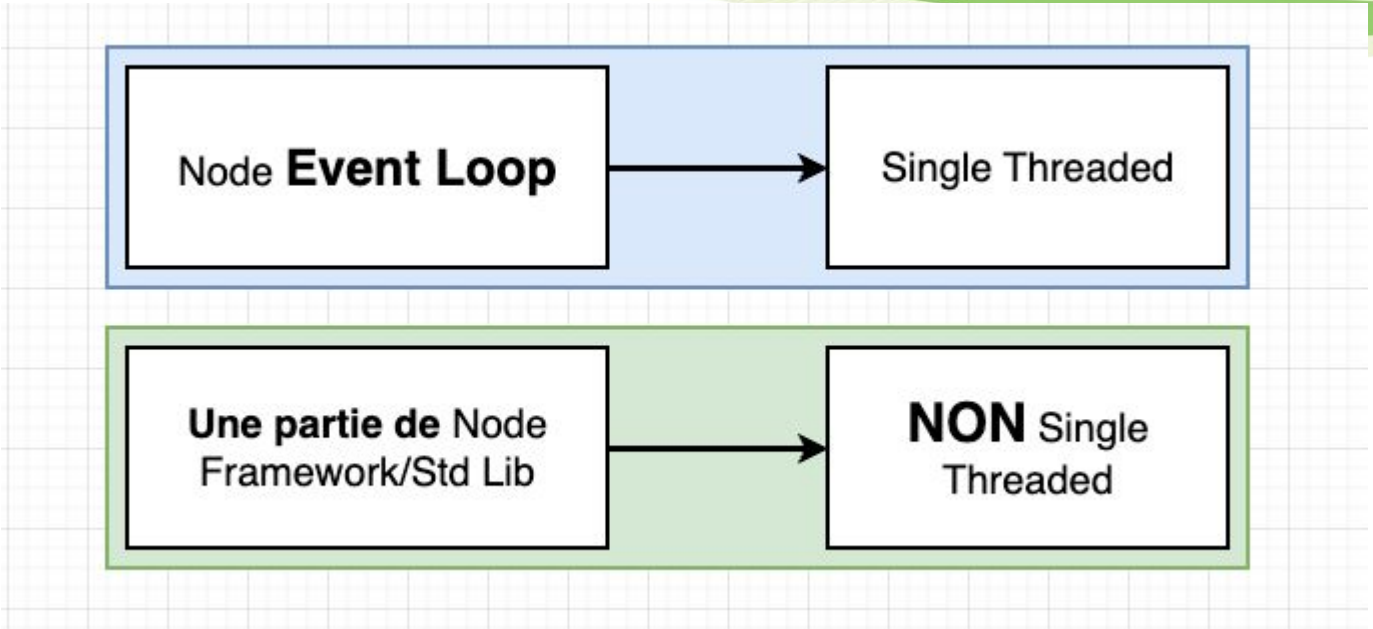
Comme une boucle infinie, elle transmet les demandes au Thread Pool (Pool de threads) et chaque requête est enregistrée une fonction Callback. Quand une requête est terminée, la fonction Callback correspondante sera appelée à être exécuté.

Lorsque le traitement d'une demande est terminé, le NodeJS appelera la fonction Callback (Enregistré pour cette demande) pour l'exécuter.

Démo : https://www.jsv9000.app/



Vue d'ensemble sur NodeJS Event Loop

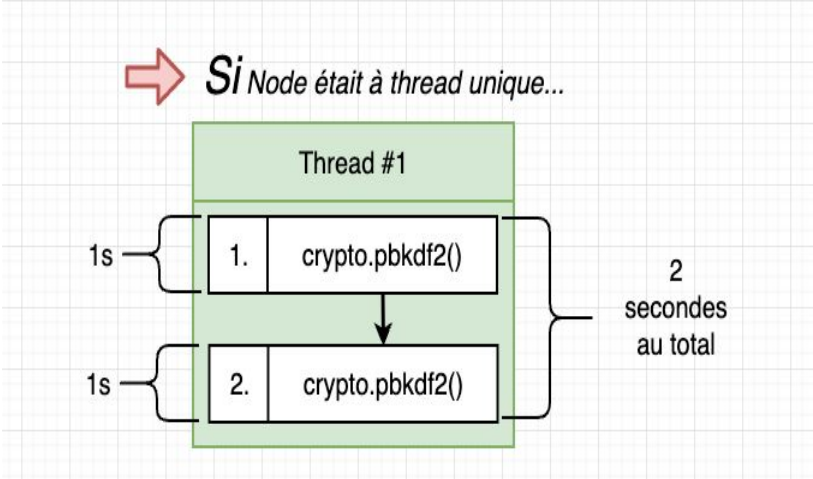


Vue d'ensemble sur NodeJS Event Loop

```
const crypto = require("crypto");

const start = Date.now();
crypto.pbkdf2('a','b',100000,512,'sha512', ()=>{
   console.log('1: ', Date.now()-start);
});

crypto.pbkdf2('a','b',100000,512,'sha512', ()=>{
    console.log('2: ', Date.now()-start);
});
```



Installation de NodeJS