WEBSTART
L'ECOLE DE LA
PROPULSION
DIGITALE

# Pourquoi faire du Node.js ?



*Vincent Caronnet // 2020-2021*
*Formateur web & mobile*

## Ce cours vous est présenté par...

**Vincent Caronnet**, infographiste et développeur web depuis 1996, j'interviens à **Webstart** et **MJM** depuis 2015 en tant que :

- **Graphiste print** (InDesign, Illustrator, Photoshop)
- **Webdesigner** (Adobe XD, HTML5, CSS3, JS)
- **Développeur web *full stack*** (Node.js, Express, Angular, Wordpress)
- **Développeur mobile** (Swift, Ionic, Cordova)

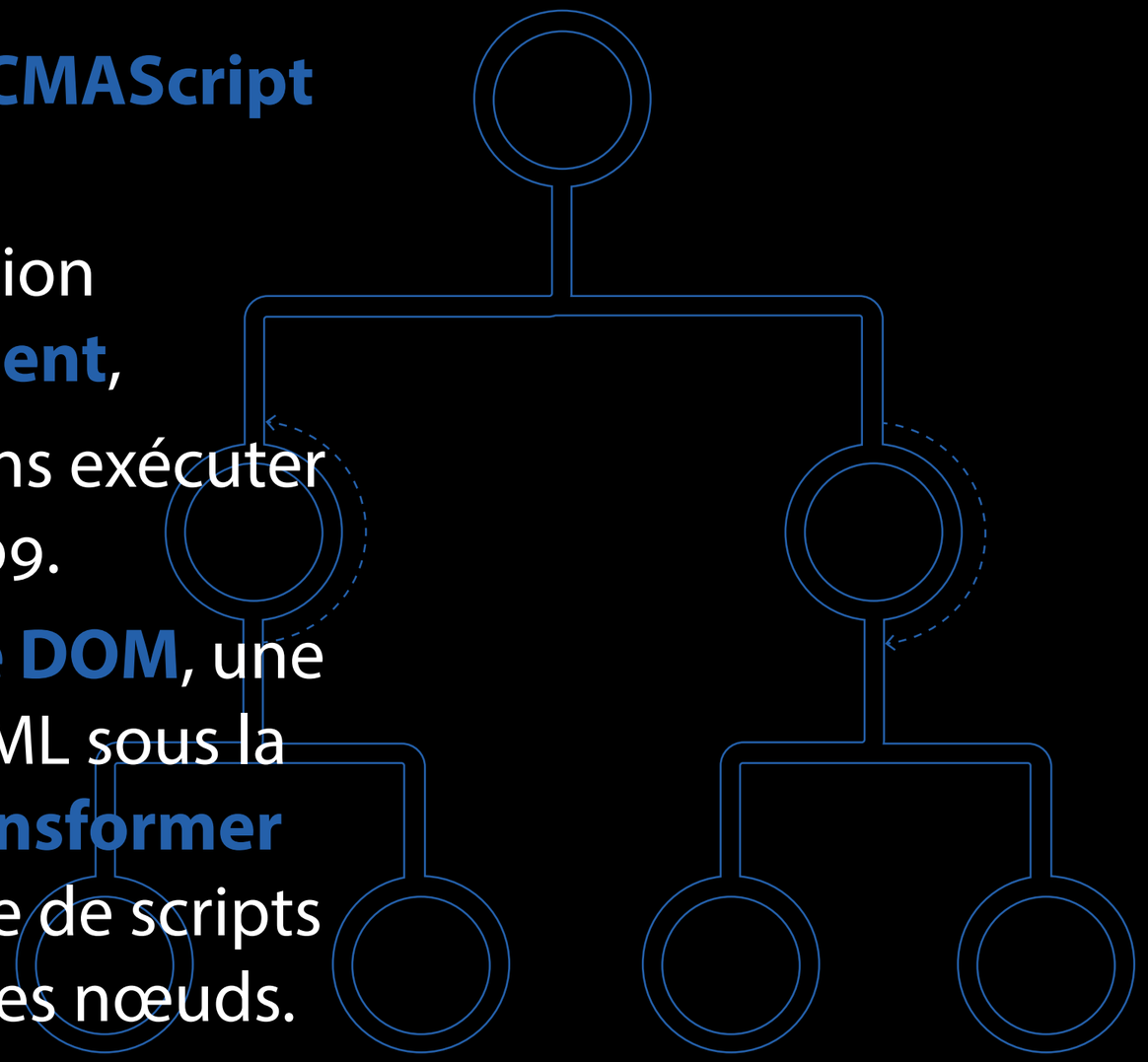
120@120.design @120design

# L'ECMAScript

Le JavaScript, dont le vrai nom est **l'ECMAScript** (ES), a été développé en 1995 :

- est **le seul langage** de programmation que nous pouvons exécuter **côté client**,
- est un langage que nous ne pouvions exécuter **que du côté du client**, jusqu'en 2009.

Pour cela, **les navigateurs pilotent le DOM**, une API qui cartographie le document HTML sous la forme de nœuds et qui permet de **transformer à la volée ce document HTML** à l'aide de scripts pour ajouter, supprimer ou modifier ces nœuds.



# Node.js, c'est...

En 2009, **Ryan Dahl présente Node.js** :

- **un environnement d'exécution JS** performant et pouvant accéder aux ressources systèmes de la machine qui l'exécute,
- **non bloquant**,
- **incluant son propre serveur HTTP** permettant de se passer d'Apache ou NGINX pour servir des ressources,
- **simple de fonctionnement** pour traiter les requêtes HTTP.

# Node.js est un environnement d'exécution JS

Pour arriver à ses fins, Ryan Dahl a d'abord utilisé C, Python et d'autres langages. **Il choisit finalement le JS** quand V8, le moteur de rendu JS développé par Google pour son navigateur Chrome, est devenu *open source*.

Le JS est compilé en langage machine **par le très performant V8**, et **Ryan Dahl lui adjoint des API** pour lui faire ce qu'il ne fait pas pour des raisons de sécurité quand il est exécuté dans un navigateur :

- **accéder au fichiers de l'ordinateur** qui l'exécute, en créer et les modifier,
- **maîtriser le transfert de données** et **les connexions réseau** de l'ordinateur qui l'exécute.

# Node.js est non bloquant

Si Ryan Dahl a choisi le JS, c'est parce qu'**il traite l'asynchronicité depuis toujours**, notamment grâce aux **événements** et aux **callbacks**.

Ce qui fait **la légèreté d'exécution** de Node.js par rapport aux autres plateformes (Java, .NET, PHP, etc.) est qu'**il n'est pas bloqué** quand il :

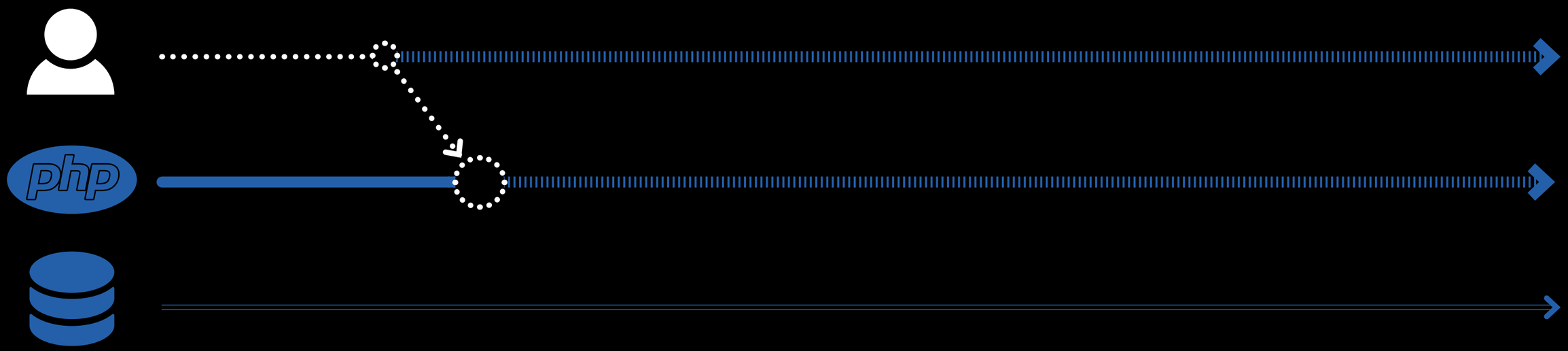
- traite une requête HTTP,
- lit ou écrit un fichier sur le disque dur,
- réalise un appel à une base de données.

Grâce aux fonctions *callbacks* ou aux promesses qui récupèrent la réponse une fois qu'elle leur parvient, ces opérations s'exécutent les unes après les autres **sans avoir à attendre le résultat** de la précédente.

# Node.js est non bloquant

Voici comment sont traitées ces opérations par **les autres plateformes** (PHP, Java, .NET, etc.) :

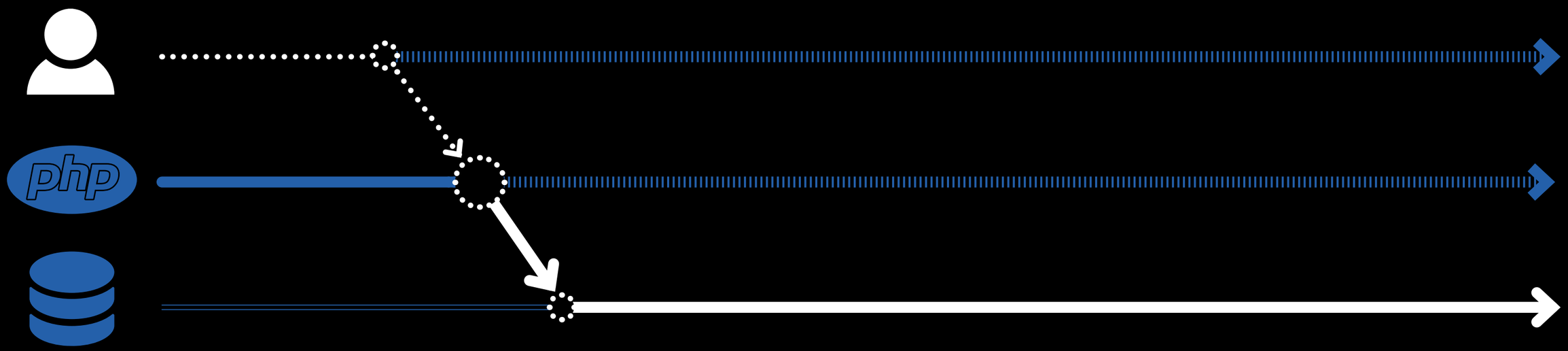
- **le client interroge le serveur**,
- le serveur doit interroger la base donnée **est bloqué en attendant**,...



# Node.js est non bloquant

Avec les autres plateformes :

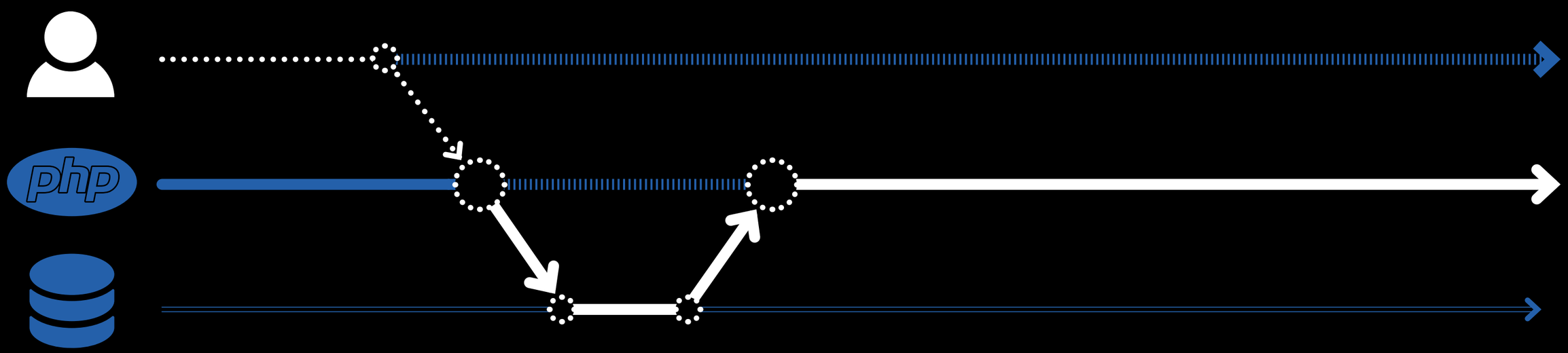
- le client interroge le serveur,
- le serveur doit interroger la base donnée est bloqué en attendant,
- **le serveur interroge la base de données**,...



# Node.js est non bloquant

Avec les autres plateformes :

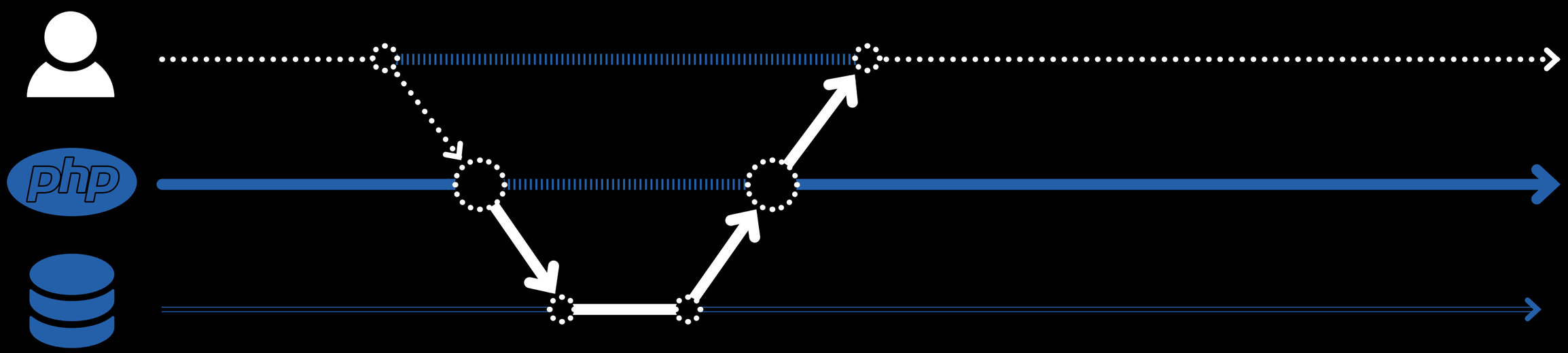
- le client interroge le serveur,
- le serveur doit interroger la base donnée est bloqué en attendant,
- le serveur interroge la base de données,
- **le serveur reçoit la réponse** de la base de données **et se débloque**,...



# Node.js est non bloquant

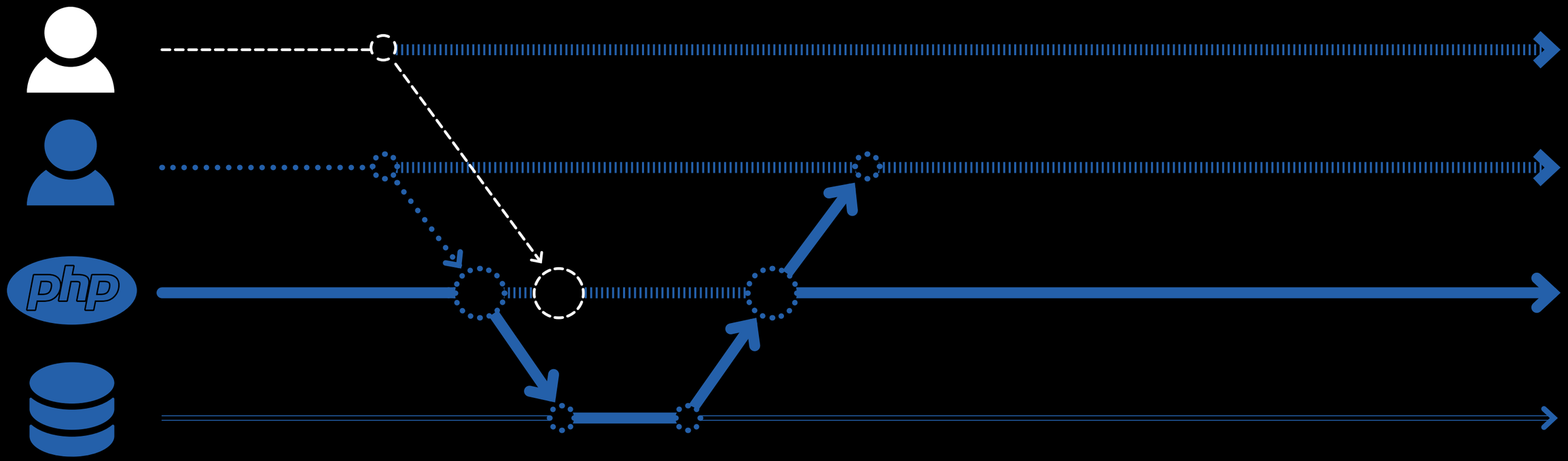
Avec les autres plateformes :

- le client interroge le serveur,
- le serveur doit interroger la base donnée est bloqué en attendant,
- le serveur reçoit la réponse de la base de données et se débloque,
- **le serveur envoie sa réponse** au client.



# Node.js est non bloquant

**Si un deuxième client** interroge le serveur
pendant ce temps, le serveur est **bloqué**
**par la requête du précédent client**,...

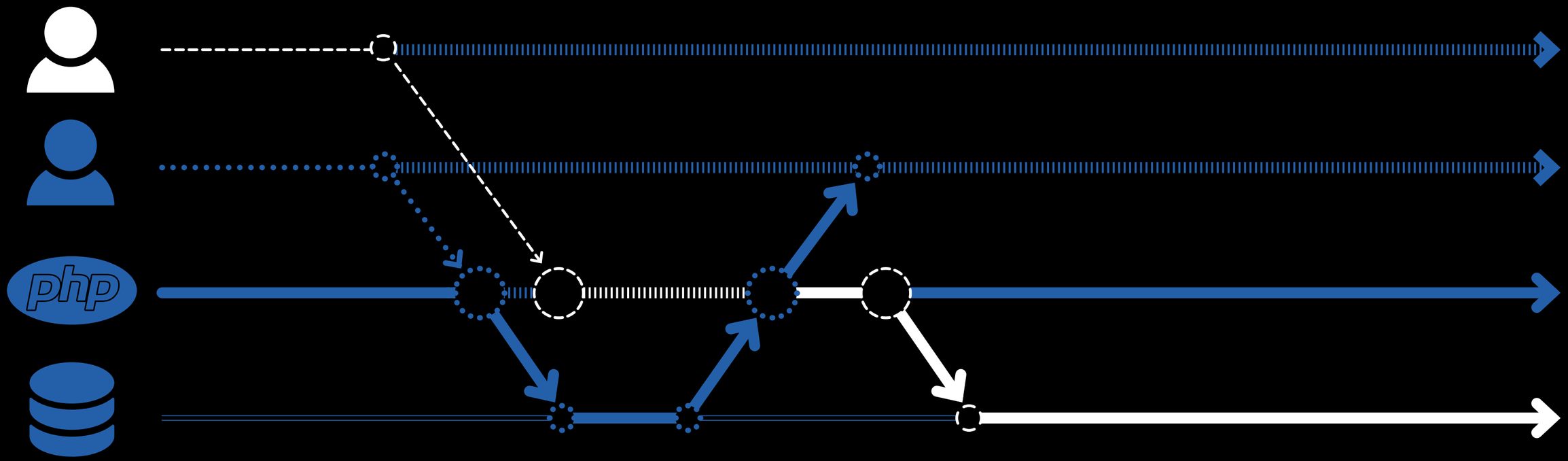





*Vincent Caronnet // 2020-2021*
*Formateur web & mobile*

# Node.js est non bloquant

... la deuxième requête ne sera traitée **que quand le serveur sera libéré**,...

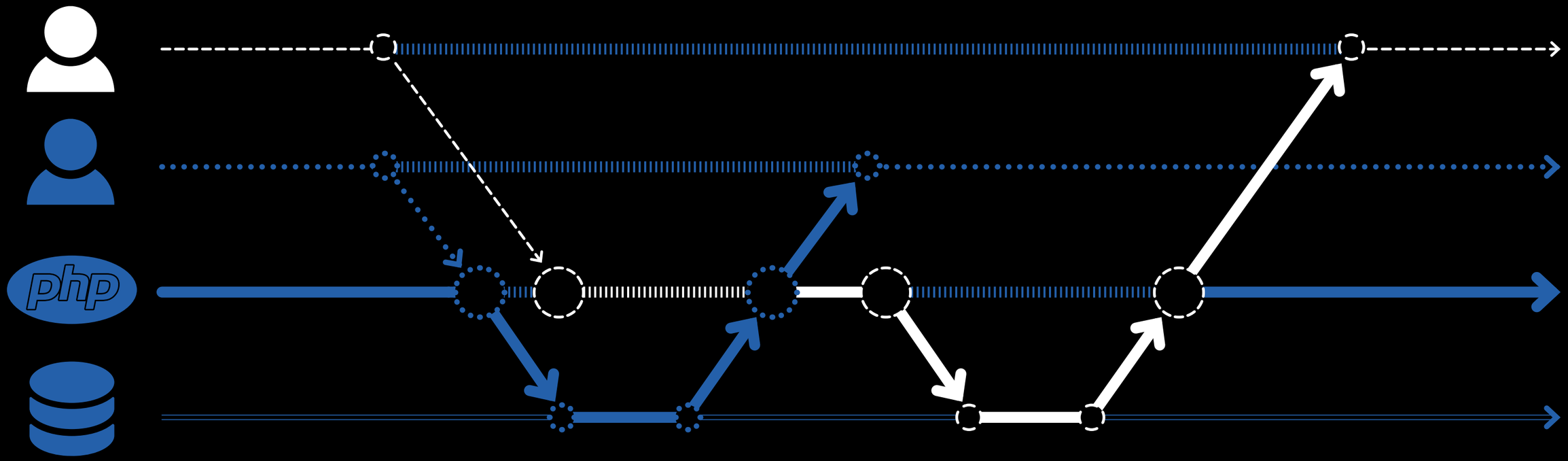





*Vincent Caronnet // 2020-2021*
*Formateur web & mobile*

# Node.js est non bloquant

... la deuxième client ne recevra donc **la réponse du seveur que bien plus tard**.



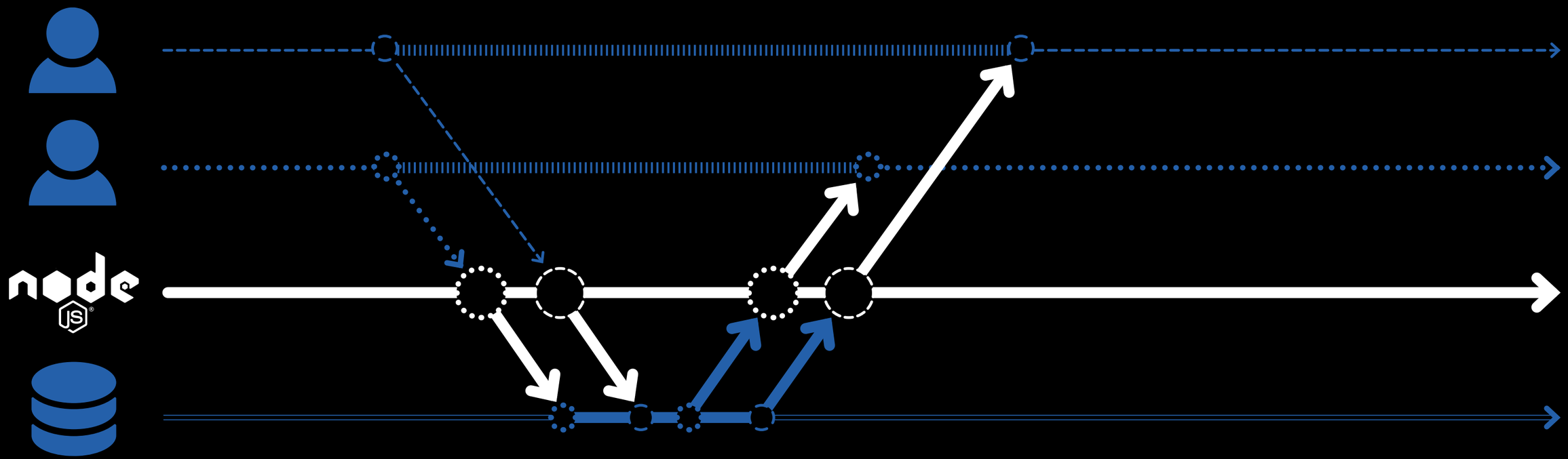



*Vincent Caronnet // 2020-2021*
*Formateur web & mobile*

*Pourquoi faire du Node.js ?*

# Node.js est non bloquant

Pour exécuter les mêmes opéraions, **Node.js ne bloque pas le serveur**, il peut **en traiter d'autres** en attendant leurs résultats et **son temps de réponse est plus rapide**.

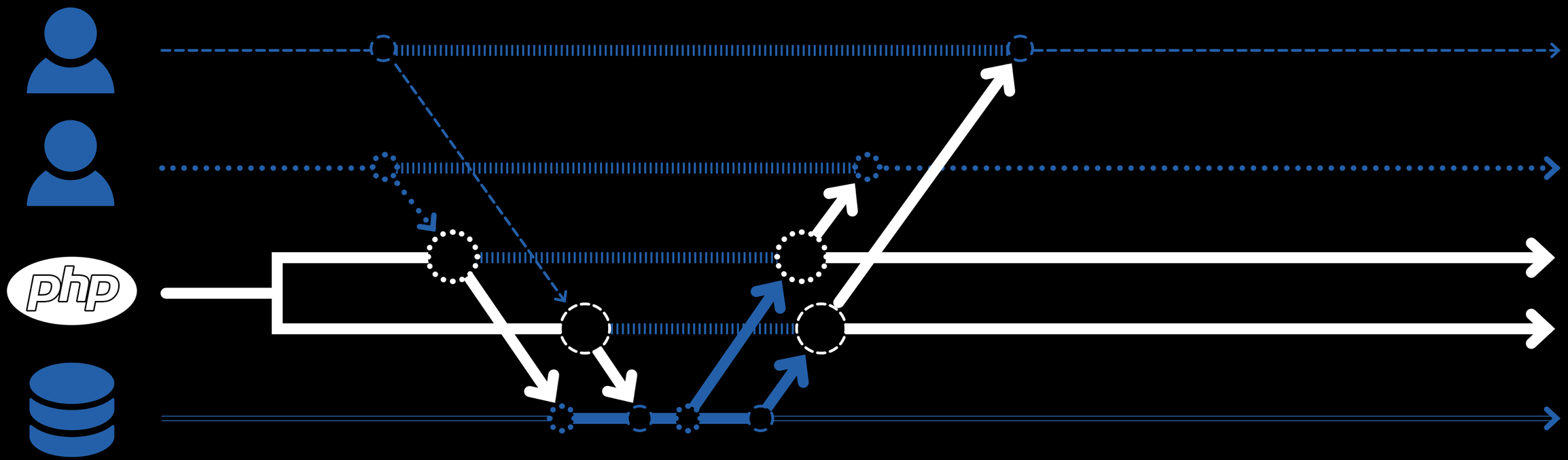





*Vincent Caronnet // 2020-2021*
*Formateur web & mobile*

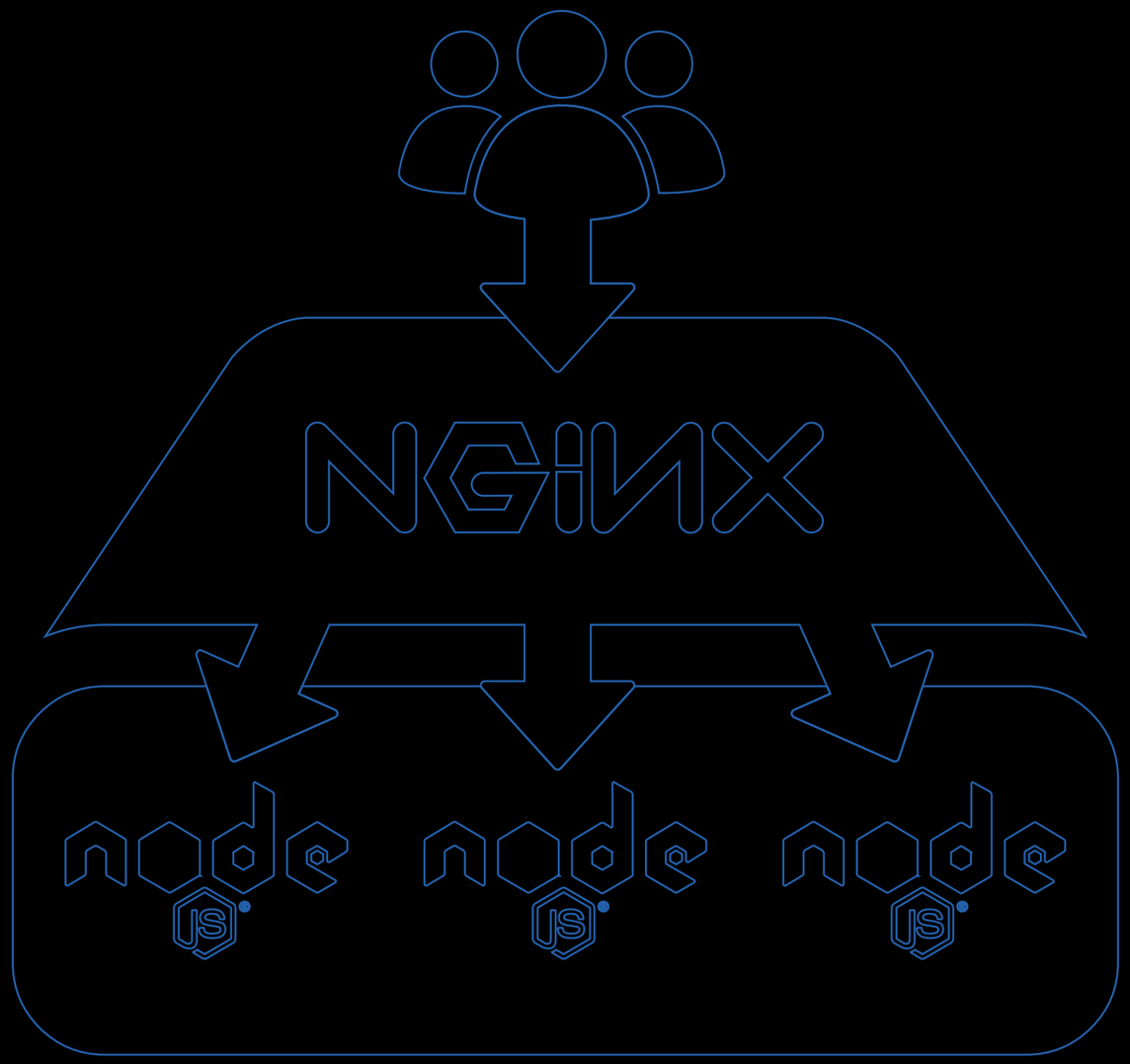
# Node.js est non bloquant

Pour arriver au même niveau de performance, **les autres plateformes** lancent **plusieurs processus en parallèle**, ce qui demande **des serveurs plus puissants** que ceux nécessaires à Node.js.



*Pourquoi faire du Node.js ?*

# Node.js inclut son propre serveur HTPP




Contrairement au PHP, **Node.js n'a pas besoin d'un serveur** Nginx ou Apache pour fonctionner : il lance **son propre serveur HTTP** et traite lui-même les requêtes qui lui sont adressées.

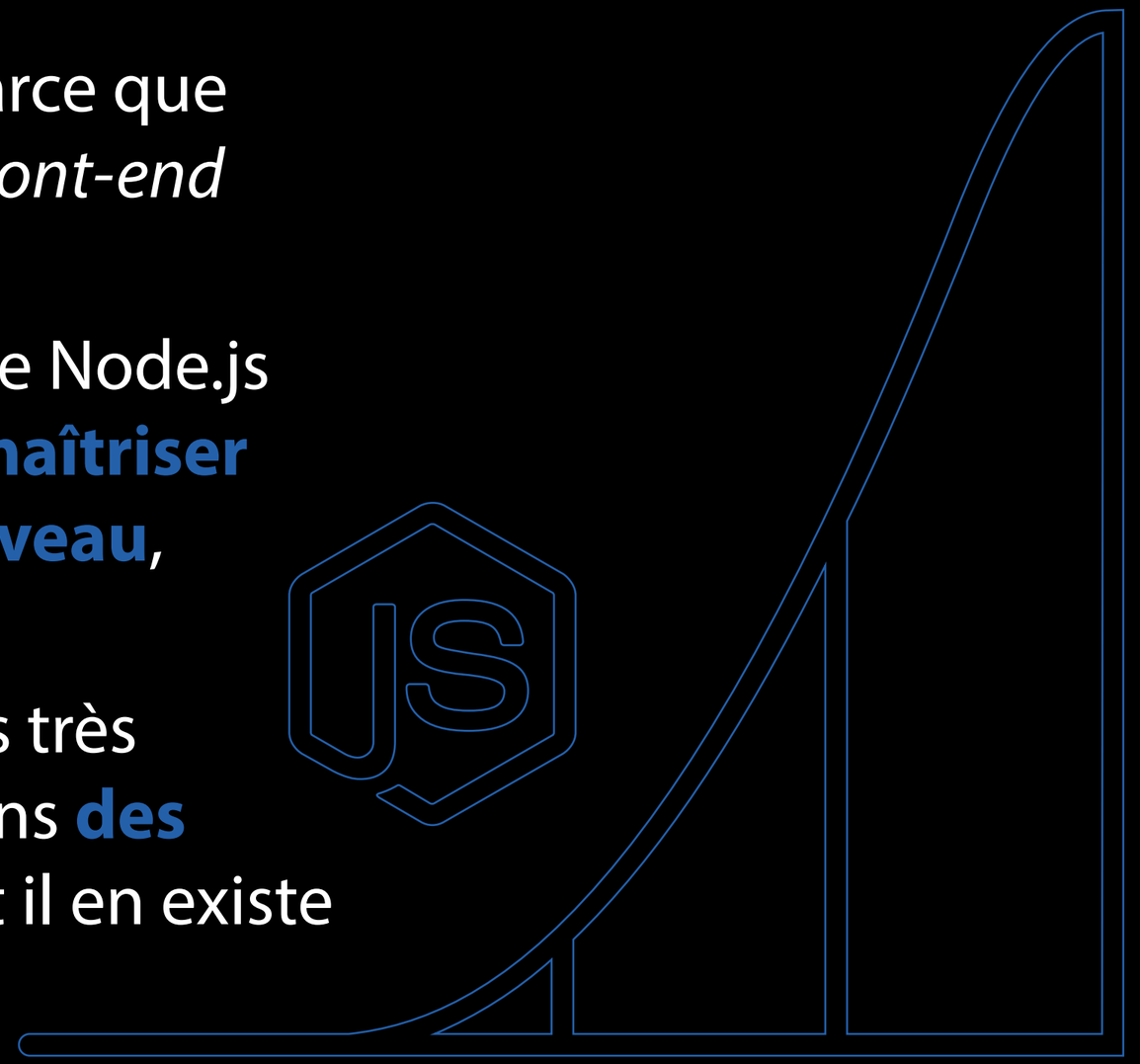
Cependant, Nginx ou Apache **restent nécessaires** pour **diriger un nom de domaine** vers une application Node.js hébergée sur une machine.

# Node.js est simple de fonctionnement

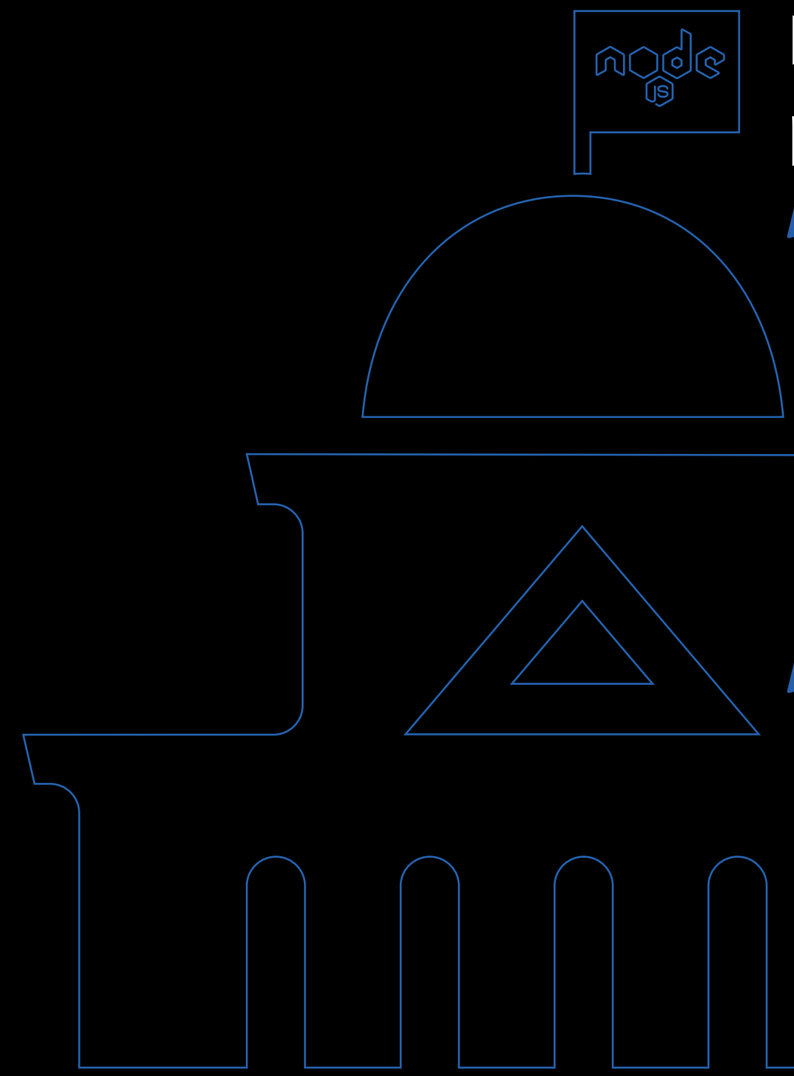
Le JS est **un langage peu exigeant** parce que faiblement typé et les développeurs *front-end* **l'utilisent couramment**.

De ce fait, la courbe d'apprentissage de Node.js est douce et permet de **rapidement maîtriser** des notions de programmation **bas niveau**, relativement proche du matériel.

**Node.js est modulaire**, nous pouvons très facilement inclure dans nos applications **des modules développés par des tiers** et il en existe des dizaines de milliers.



# L'écosystème de Node.js



Le projet Node.js est *open source* et est gouverné par deux entités :

- la **Node.js Foundation**, organisation à but non lucratif elle-même membre de la Linux Foundation, est composée d'entreprises telles que Google, IBM, Microsoft, Intel et PayPal, et assure **la communication** du projet,
- le **Technical Steering Committee** (TSC), composé de contributeurs et de collaborateurs individuels, dirige le projet, prend **les décisions techniques** et en rend compte à la Node.js Foundation.

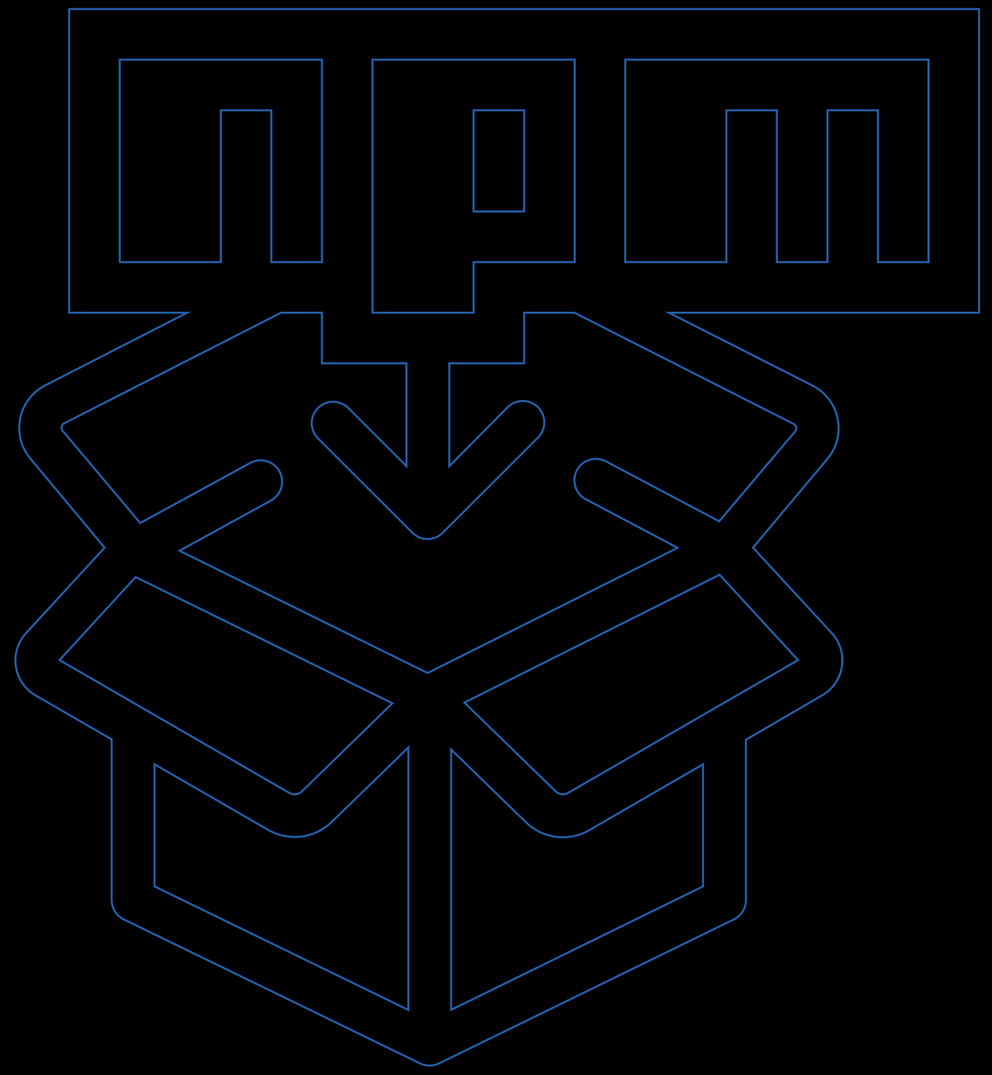
# L'écosystème de Node.js

Le TSC fixe le calendrier des mises à jour de Node.js et sort environ **deux nouvelles versions** majeures par an :

- **une version intermédiaire** dont le numéro est impair (ex. : `v9.0.0`), elle n'est développée et maintenue que pendant un peu plus de six mois,
- **une version Long Time Support** (LTS) dont le numéro est pair (ex. : `v10.0.0`), elle est développée et continue d'être maintenue pendant presque trois ans.

Il est préférable d'**utiliser les versions LTS en production** et de réserver les versions intermédiaires au test de nouvelles fonctionnalités ou pour migrer une application vers la LTS suivante.

# L'écosystème de Node.js




Au cœur de Node.js, il y a le **Node Package Manager** (NPM).

Grâce à NPM qui **est installé par défaut** avec Node.js, nous incluerons dans nos applications des modules développés par d'autres développeurs.

Il existe des dizaines de milliers de *packages* et ils sont référencés sur https://www.npmjs.com/.

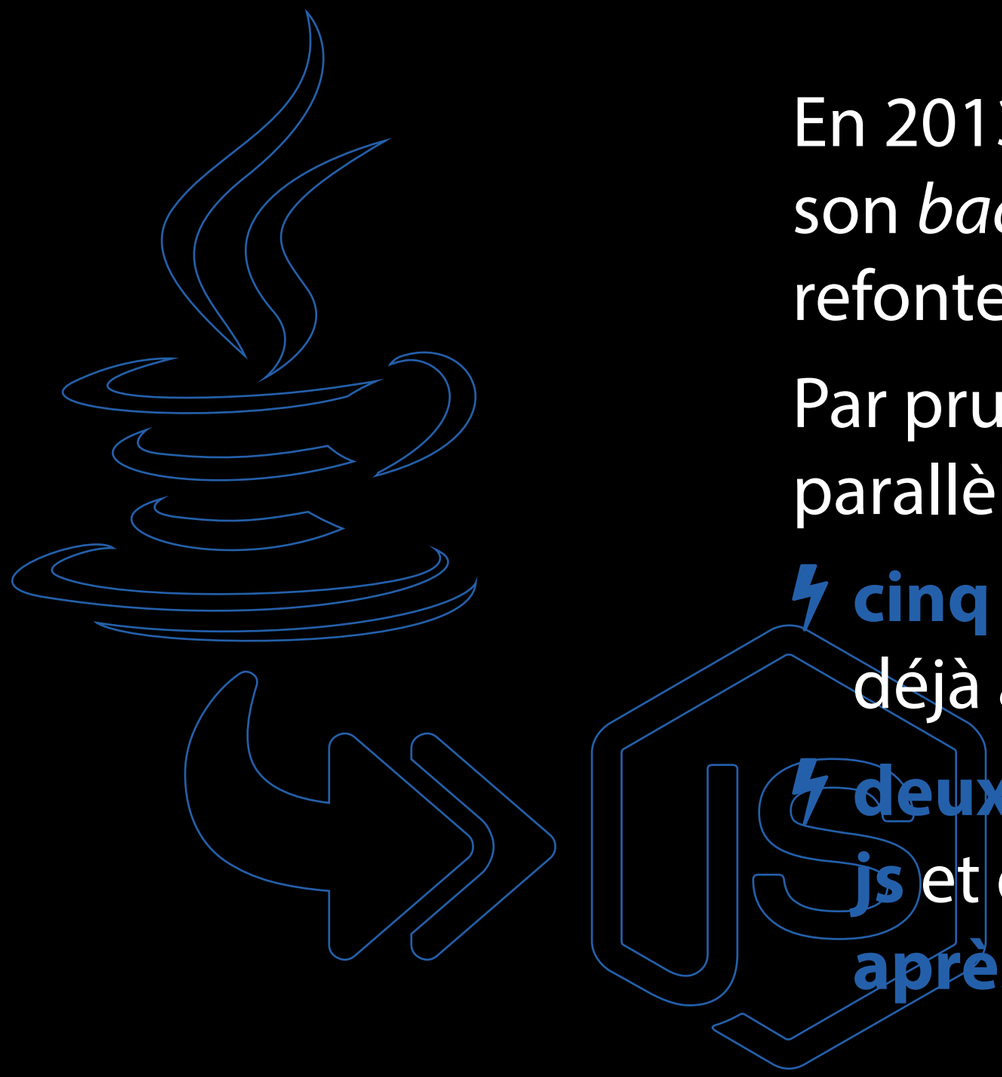
# Les success-stories de Node.js

Quand elle a lancé son application mobile en 2011, **Linkedin** a changé les **trente serveurs Ruby on Rails** qui hébergeaient son API pour **trois serveurs Node.js**.

Avec ces trois serveurs Node.js, Linkedin a alors constaté **des performances allant jusqu'à vingt fois celles de Ruby on Rails**.

Pour couronner le tout, Linkedin a aussi constaté que ses nouveaux serveurs ne fonctionnaient qu'à **10 % de leurs capacités** et auraient pu traiter dix fois plus de tâches.

# Les success-stories de Node.js



En 2013, **PayPal** a raconté comment elle a migré son *backend* de Java à Node.js à l'occasion de la refonte des pages du *backoffice* de ses utilisateurs.

Par prudence, **deux équipes** ont travaillé en parallèles pour coder les même fonctionnalités :

- **cinq développeurs Java confirmés** travaillant déjà avec ce langage dans l'entreprise,
- **deux développeurs qui découvrent Node.js** et qui ont commencé le travail **deux mois après les développeurs Java**.

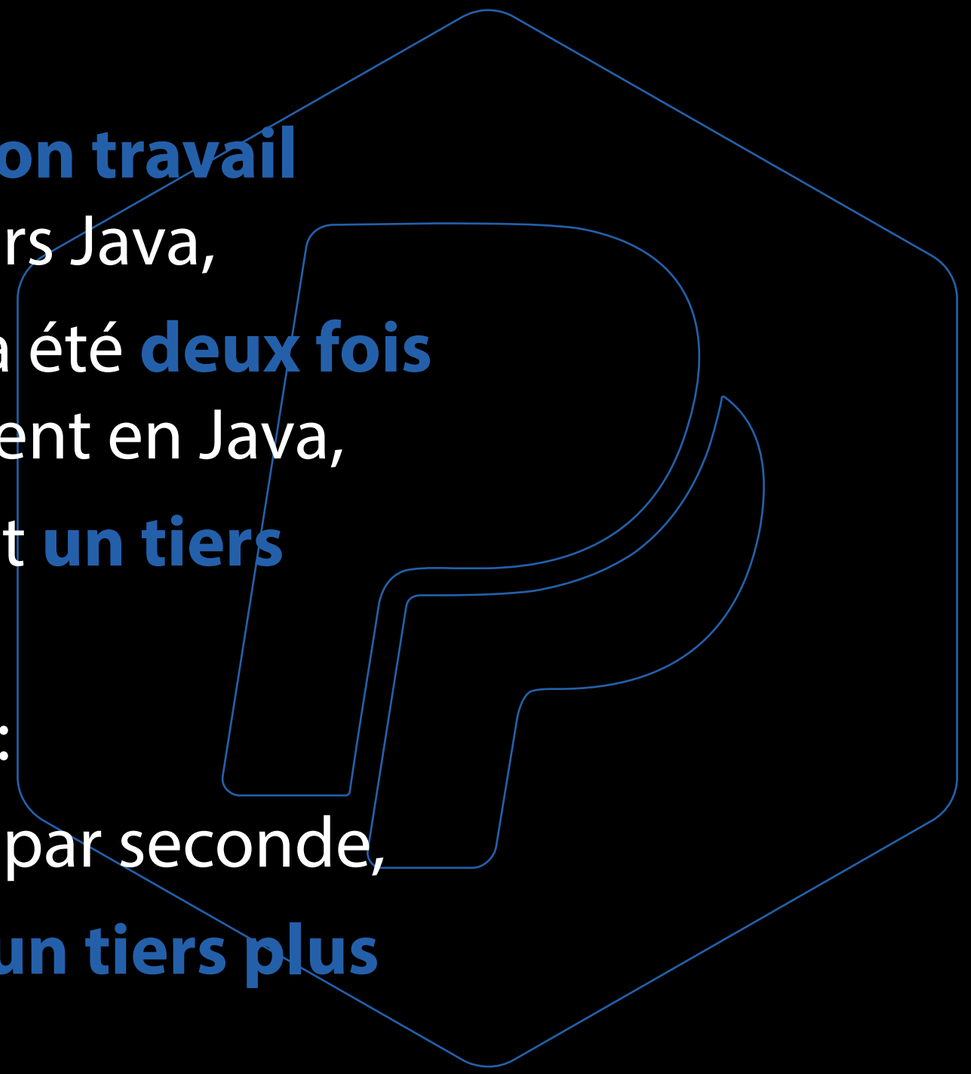
# Les success-stories de Node.js

Malgré ces gros handicaps :

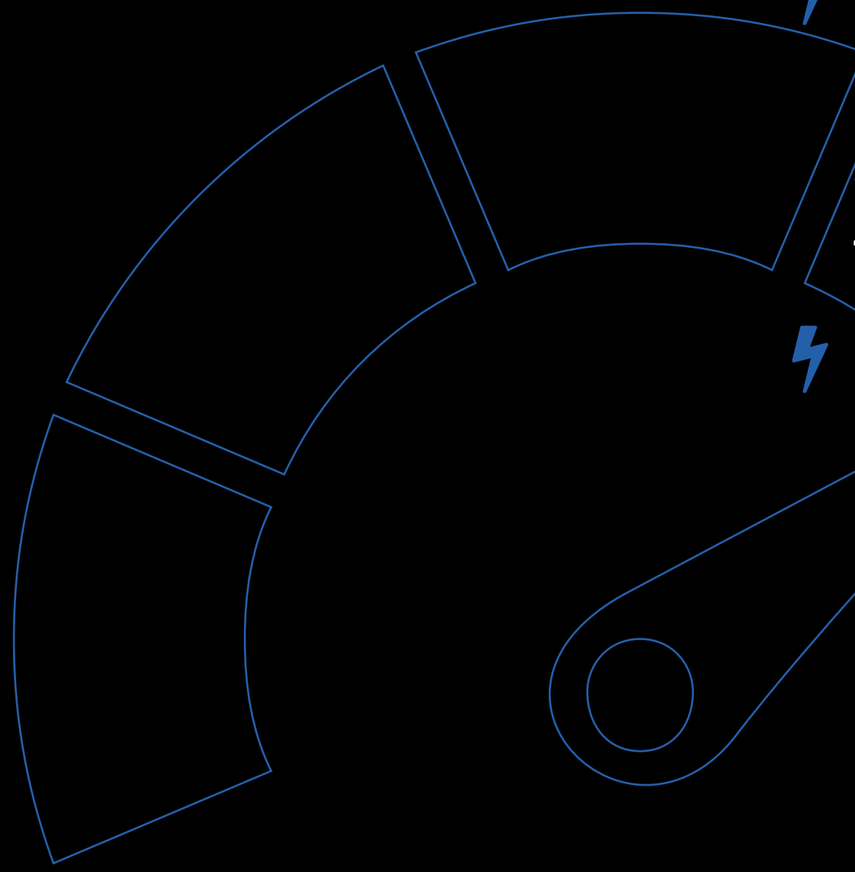
- l'équipe de développeurs Node.js **a livré son travail en même temps** que celle de développeurs Java,
- **le temps de développement** en Node.js a été **deux fois plus rapide** que le temps de développement en Java,
- l'équipe de développeurs Node.js a produit **un tiers de lignes de code en moins**.

Du côté des performances, **comparé à Java** :

- Node.js traite **deux fois plus de requêtes** par seconde,
- **le temps de réponse** de ces requêtes est **un tiers plus court**.



# Les success-stories de Node.js



De même :

- **Groupon** affirme avoir **réduit de manière importante le nombre de ses serveurs** après avoir migré vers Node.js, tout en réduisant le temps de chargement de ses pages de moitié,
- un jour de *Black friday*, **Wallmart** a réalisé des tests A/B avec Node.js d'un côté et ses serveurs de l'autre, il a alors été constaté que les processeurs des serveurs Node.js ne consommaient **que 1 % de leurs capacités** et qu'ils ne tombaient pas en panne.

# Quand ne faut-il pas utiliser Node.js ?

Nous ne devrions pas utiliser Node.js :

- si notre appliccation nécessite **une puissance de calcul importante** parce ce n'est pas là que Node.js excelle,
- si nous devons faire la refonte web qui **fonctionne déjà très bien** avec du PHP ou du .NET, il n'est pas nécessaire detout réécrire en Node.js,
- si nous travaillons **dans une équipe qui est déjà à l'aise avec une autre technologie**, elle perdrait du temps à prendre de nouvelles habitudes,
- s'il existe **une technologie avec une meilleure communauté** de développeurs que celle de Node.js, nous gagnerons du temps à utiliser cette technologie, par exemple : faire des mathématiques avec Python, proposer un CMS en PHP avec Wordpress ou Drupal, etc.

# Quand faut-il utiliser Node.js ?

Mais nous devrions utiliser Node.js :

- si notre appliccation nécessite **de nombreuses actions concurrentes**, par exemple : recevoir et 'envoyer des requêtes en permanence,
- si nous créons **un site web from scratch**, puisque nous n'aurons pas à déconstruire une solution existante,
- si nous souhaitons **unifier les équipes de développement** *front-end* et *back-end*, le JS devient alors un langage commun,
- si nous souhaitons **améliorer nos connaissances du JavaScript** et pratiquer **les dernières évolutions** du langage.