Conception Et Programmation Web

LIFWEB - http://lifweb.pages.univ-lyon1.fr/

Semestre printemps 2024-2025
L3 - UCBL

Aurélien Tabard - https://tabard.fr/, basé sur les cours de Romuald Thion

Serveur & JS

Node.js - fetch et requêtage - gestion de paquets

Aurélien Tabard - 2024-2025 - basé sur les cours de Romuald Thion

Serveur & JS

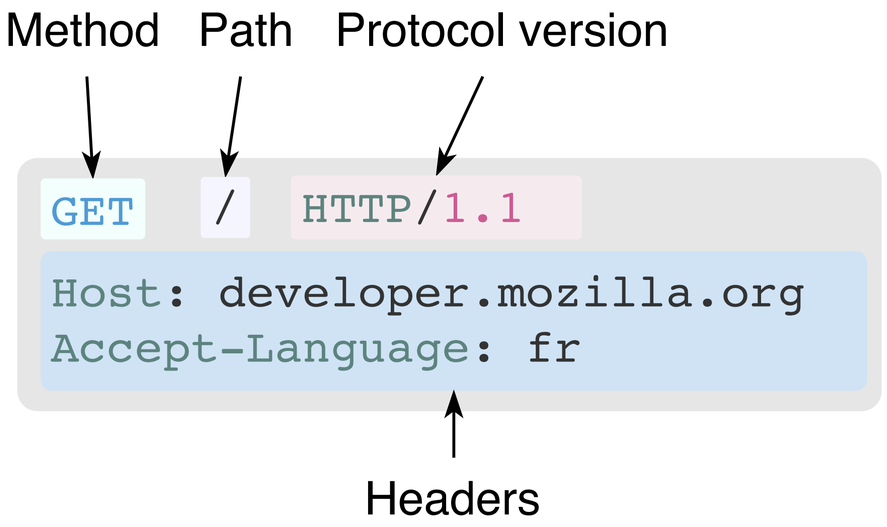
Plan

1. Rappels
2. Requêtage
3. Node.js
4. Node package manager (npm)

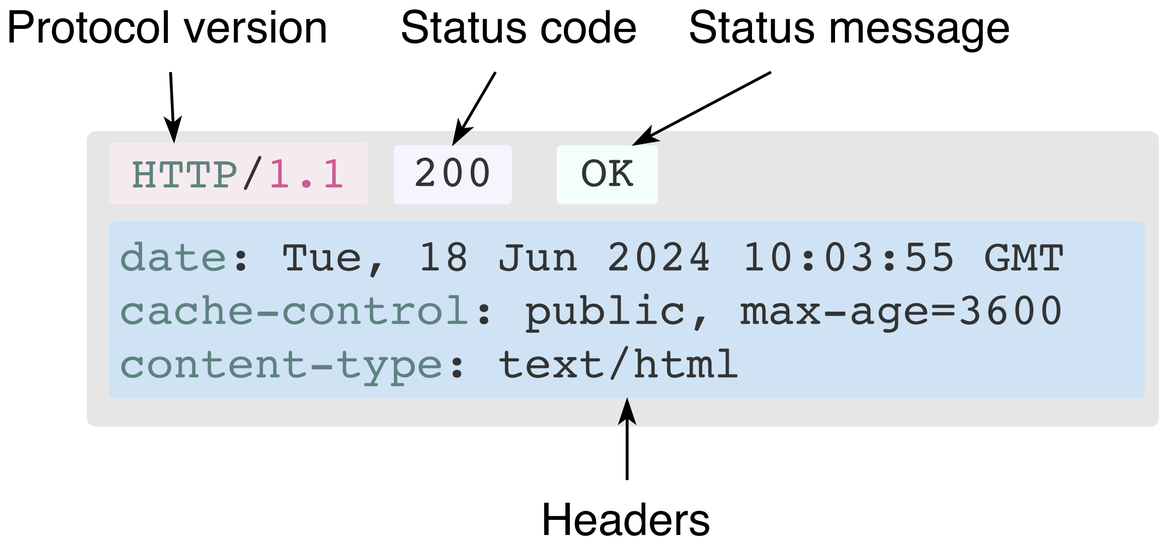
HTTP

[MSDN: HTTP Overview]

Requête



Réponse



Méthodes des requêtes HTTP

Appelées aussi verbes

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

GET : get a specific resource

POST: create a new resource

PUT: update an existing resource (or create)

DELETE: delete the specified resource

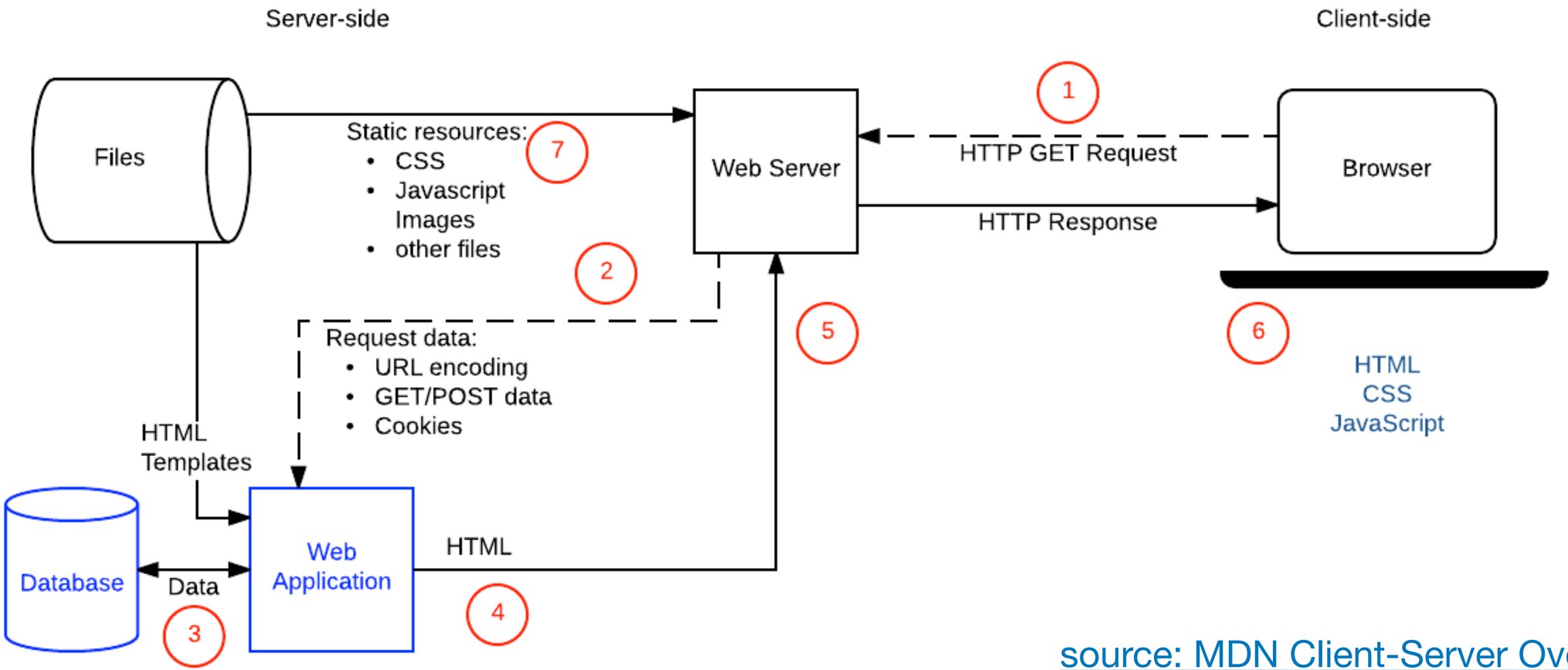
HEAD: get the metadata information

comme GET mais sans contenu (en-tête seulement)

TRACE, OPTIONS, CONNECT, PATCH: avancées

Architecture d'un site Web dynamique

ex: velov.grandlyon.com architecture n-tiers



source: MDN Client-Server Overview

Serveur & JS

Plan

1. Rappels
2. Requêtage
3. Node.js
4. Node package manager (npm)

API Web (1)

Des APIs Web comme https://thecatapi.com/ :

- logique métier, données, etc. ;
- éventuellement présentation HTML ;
- en Node.js (mais peut-être Java, Python, Rust).

👉 la suite de LIFWEB

Des clients qui les utilisent :

- un squelette HTML/CSS ;
- alimenté à partir des APIs via JS.

👉 précédemment dans LIFWEB.

Utiliser une API

Exemple sur https://lifweb.univ-lyon1.fr/

📄 Fichier JSON de définition de l'API openapi.json

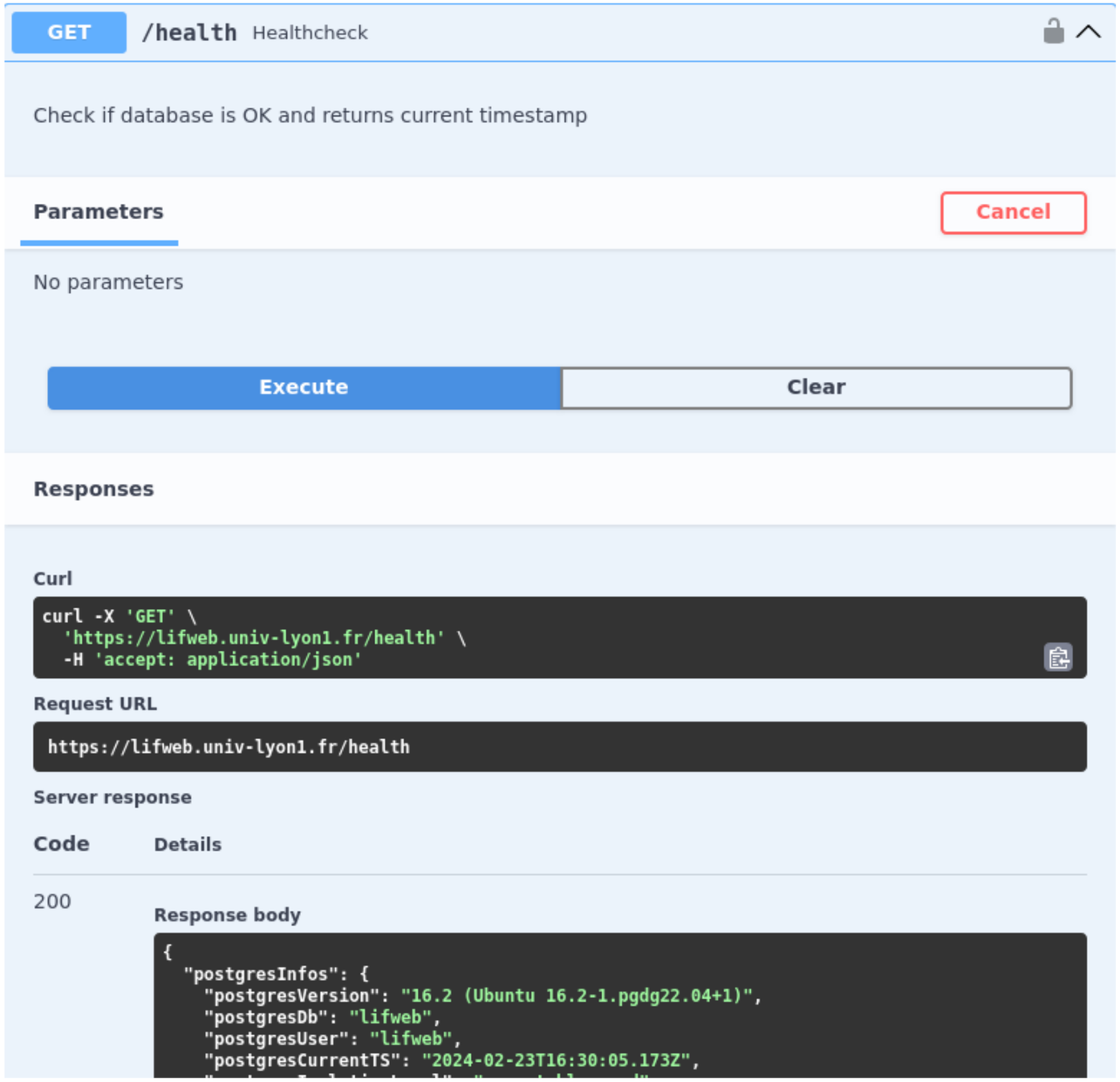
- Généré automatiquement par https://hapi.dev/
- Partie de la spécification du service
- Standard https://www.openapis.org/

🖥️ Interface https://swagger.io/

- Point d'entrée de la documentation
- Permet de tester l'API en direct
  - Génère des XHR (style fetch) pour vous

Démo

https://lifweb.univ-lyon1.fr/documentation#/health/getHealth



curl

https://curl.se/

```
curl https://lifweb.univ-lyon1.fr/health -H 'accept: application/json'
```

À combiner avec jq pour le rendu :

```
{
  "postgresInfos": {
    "postgresVersion": "16.2 (Ubuntu 16.2-1.pgdg22.04+1)",
    ...
    "driver": "https://github.com/porsager/postgres"
  },
  "serverId": "lifweb:369857:lsuib8s3",
  "serverStartedAt": "2024-02-20T15:14:54.519Z",
  "serverUri": "http://localhost:8001",
  ...
  "title": "lif-web-challenge-server",
  "version": "1.1.1"
}
```

Httpie

https://httpie.io/

```
https lifweb.univ-lyon1.fr/health accept:application/json
```

Méthode (non programmable) préférée 🤩

- Version CLI et GUI ;
- Plus intuitif que curl ;
- Complet, voir exemples.

Démo

```
https "api.openweathermap.org/data/2.5/weather?q=Lyon\
&appid=API_KEY” accept:application/json
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST
Access-Control-Allow-Origin: *
…
```

```
{
    "base": "stations",
    "clouds": {
        "all": 6
    },
    "id": 2996943,
    "main": {
        "feels_like": 271.59,
       "temp": 271.59,
        "temp_max": 272.98,
        "temp_min": 271.2
    }
…
}
```

https://curlconverter.com/ traduit les commandes curl

```
fetch("https://lifweb.univ-lyon1.fr/health", {
  headers: {
    accept: "application/json",
  },
});
```

Requêtage depuis le navigateur

Usage fetch + promise

exemple-check-ok-http.js

```
fetch("https://httpbin.org/status/418")
  .then(checkOK)
  .then((resp) => console.log(resp.status))
  .catch(console.error);
```

Usage async/await

```
try {
  const resp = checkOK(await fetch("https://httpbin.org/status/418"));
  console.log(resp.status);
} catch (error) {
  console.error(error);
}
```

Réponse

Propriété ok (MDN)

```
function checkOK(response) {
  if (!response.ok) {
    throw new Error(`[${response.status}] ${response.statusText}`);
  }
  return response;
}
```

🗒️ Remarques :

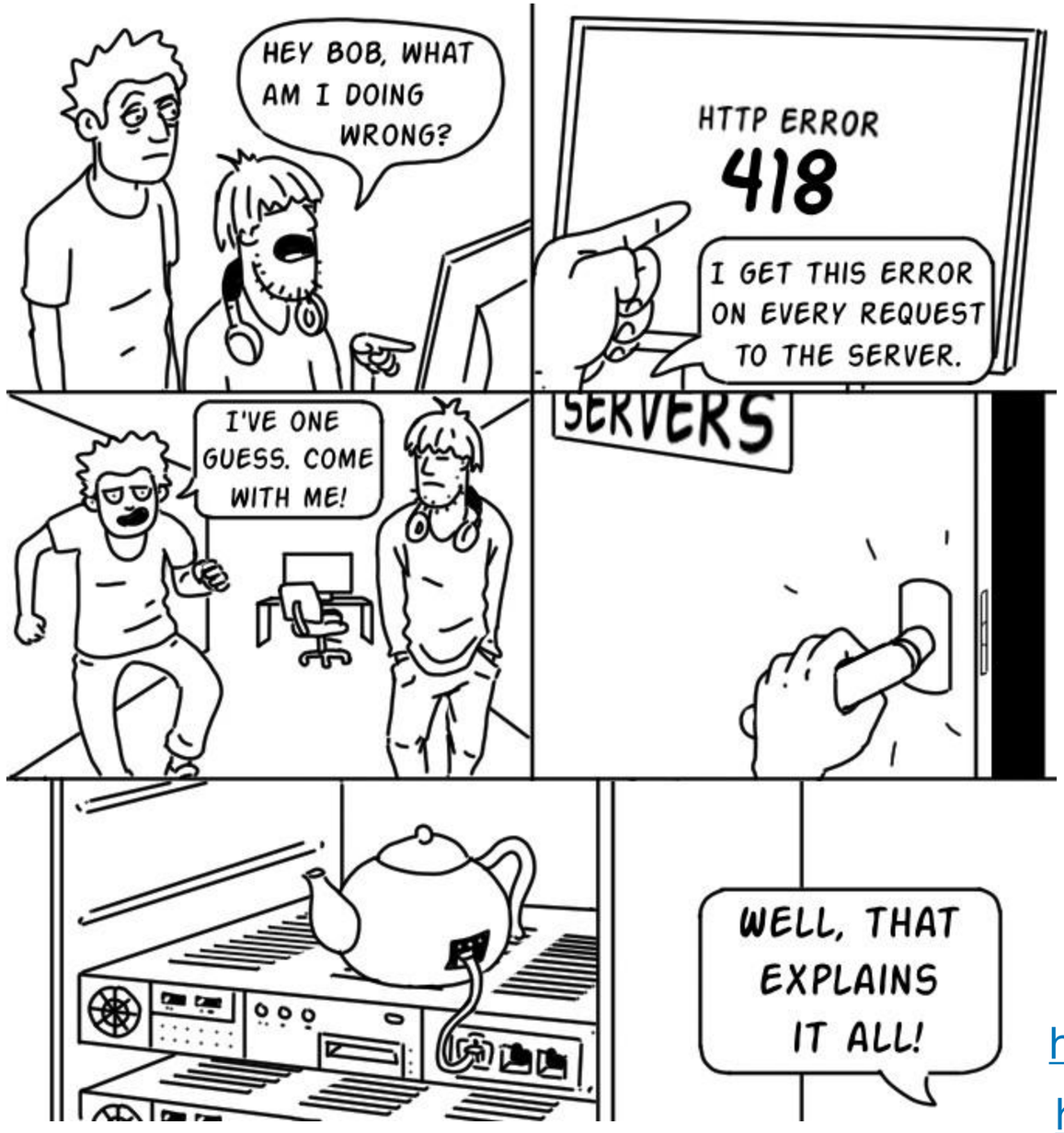
- On crée toujours un objet Error (MDN).
- On laisse remonter l'erreur à l'appelant.
- Si OK, on renvoie response pour chaînage.

📄 exemple-check-ok-http.js

Vérifier une réponse HTTP

💡 On a souvent besoin d’un handler pour vérifier la réponse HTTP pour faire la différence entre :

- Une erreur de réseau
  - pas d’accès internet, nom DNS non résolu, etc.
- Une réponse HTTP avec un code 4xx ou 5xx
  - une erreur applicative client ou serveur.



https://save418.com/

https://www.google.com/teapot

Serveur & JS

Plan

1. Rappels
2. Requêtage
3. Node.js
4. Node package manager (npm)

Node vs. Navigateur

Serveur / back-end

- moteur V8
- accès système complet
  - sockets, threads…
- event loop + pool de threads
- modules
  - CommonJS (CJS)
  - EcmaScript (ESM) 👈

Client / front-end

- moteur du browser
- limitations de sécurité
  - CORS…
- events loop du navigateur
- modules EcmaScript

🚧 Node.js s’aligne progressivement sur EcmaScript
et les APIs Web, mais les APIs historiques restent. 🚧

Exécution des programmes node.js

S’utilise comme Python ou OCaml (sans compilation) :

soit en interpréteur interactif REPL

- REPL = Read Eval Print Loop

soit en interpréteur non interactif

- node file.js

💡 La majorité des outils de l’écosystème JS utilisent Node.js.

Différence avec ocaml ou python

🔁 Node.js intègre nativement une boucle d’événements

- programmation asynchrone/événementielle native
- applications orientées serveur

🚀 les applications I/O intensive sont performantes

🐢 Les applications CPU intensive sont lentes

- JS est un langage interprété et lent
- On interface JS à du code natif (C/C++/Rust)

Runtime js en 2025

Alternatives à node.js

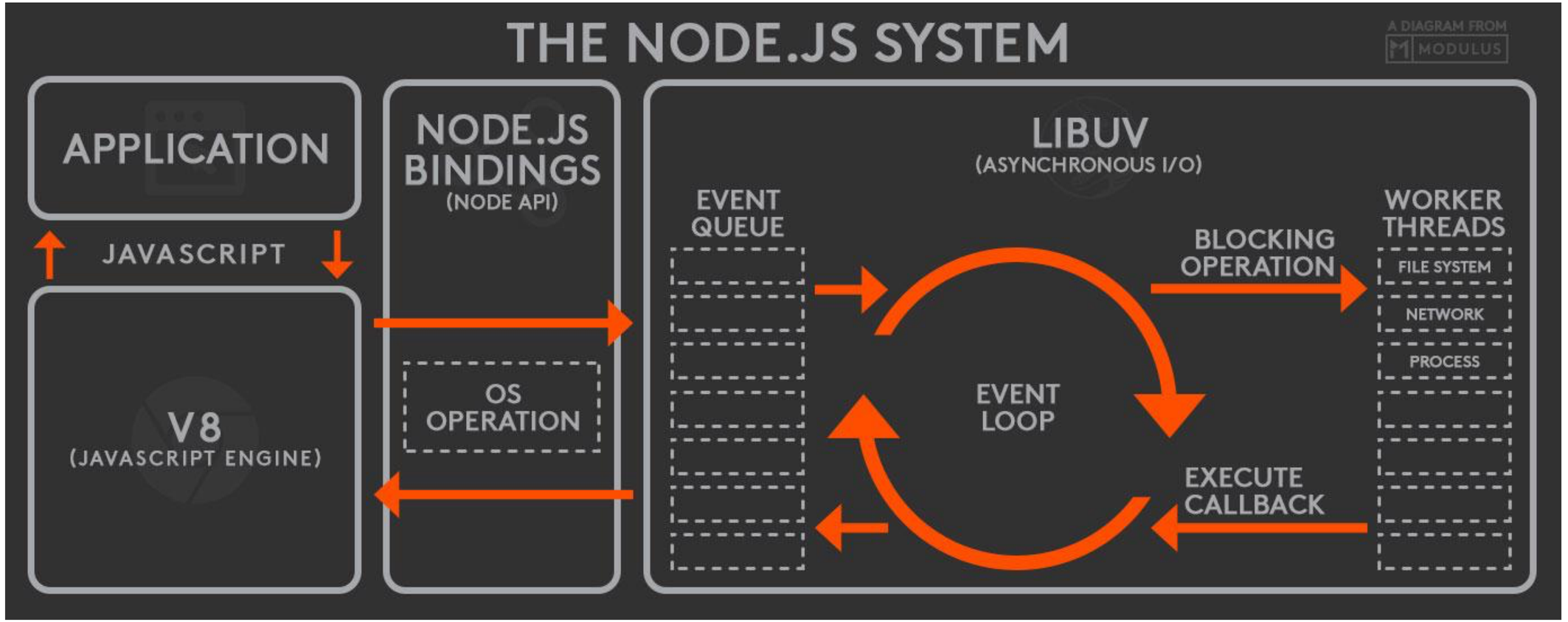
https://github.com/errilaz/awesome-js-runtimes

https://bestofjs.org/projects?tags=runtime

Runtime	Engine	Library	Stars	Activity
Node.js	V8	libuv	stars 110k	last commit february
Deno	V8	tokio	stars 102k	last commit today
Bun	JavaScriptCore		stars 76k	last commit today
Just	V8		stars 3.7k	last commit november 2023
Txiki.js	QuickJS	libuv	stars 2.7k	last commit january
Napa.js	V8		stars 9.2k	last commit october 2018
LLRT	QuickJS	tokio	stars 8.3k	last commit last thursday
WinterJS	SpiderMonkey, Spiderfire	hyper	stars 3.1k	last commit july 2024
Elsa	QuickJS		stars 2.8k	last commit november 2022
Window.js	V8	libuv, GLFW	stars 2.3k	last commit march 2023
Kaluma	JerryScript		stars 728	last commit february
Bare	V8	libuv	stars 261	last commit february

🔮 Standardisation JS backend https://wintercg.org/. 🔮

Architecture de node



source

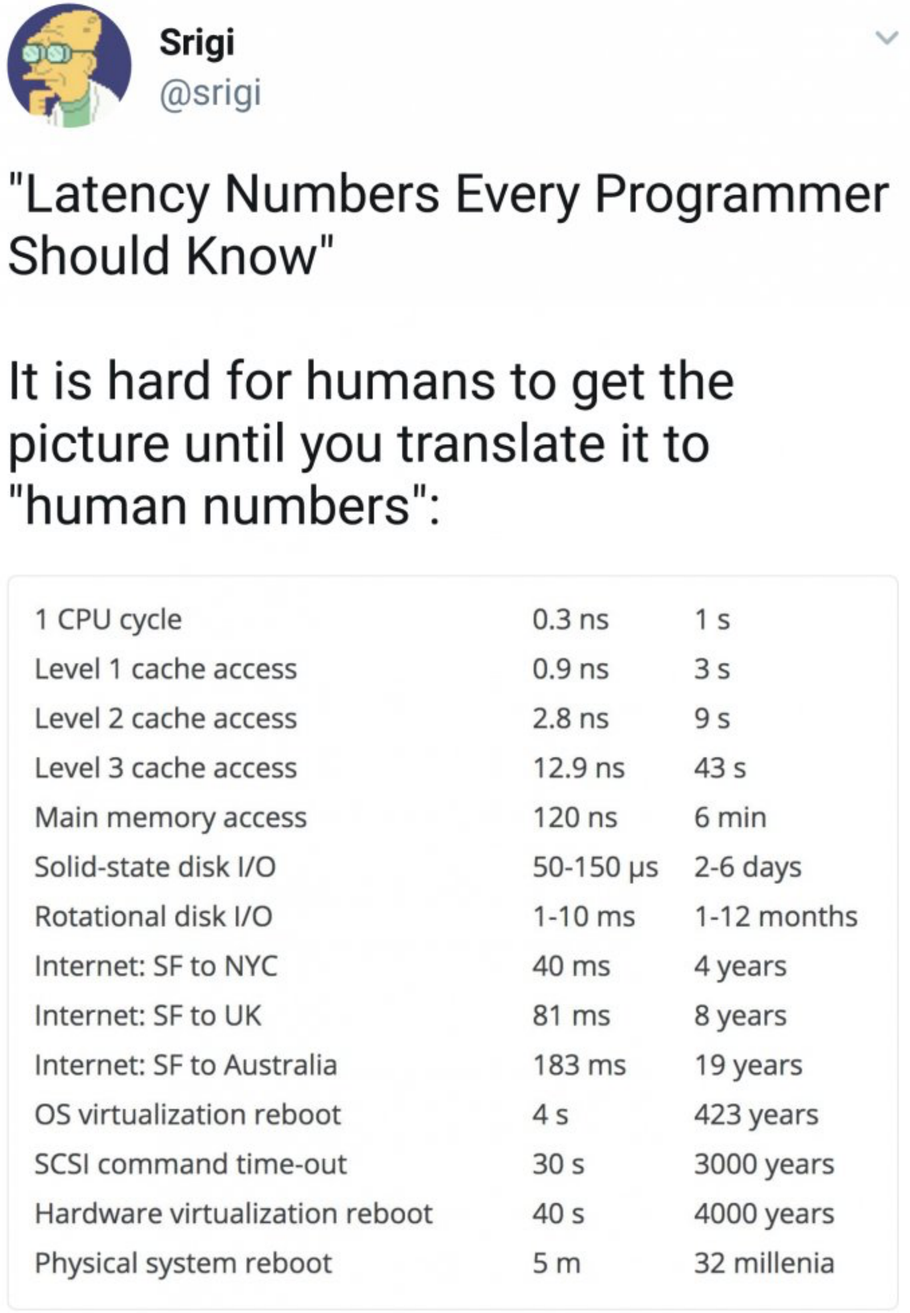
I/O vs CPU

Une requête HTTP :
-> des 100aines de millions de cycles !

Une multiplication : 10 à 20 cycles

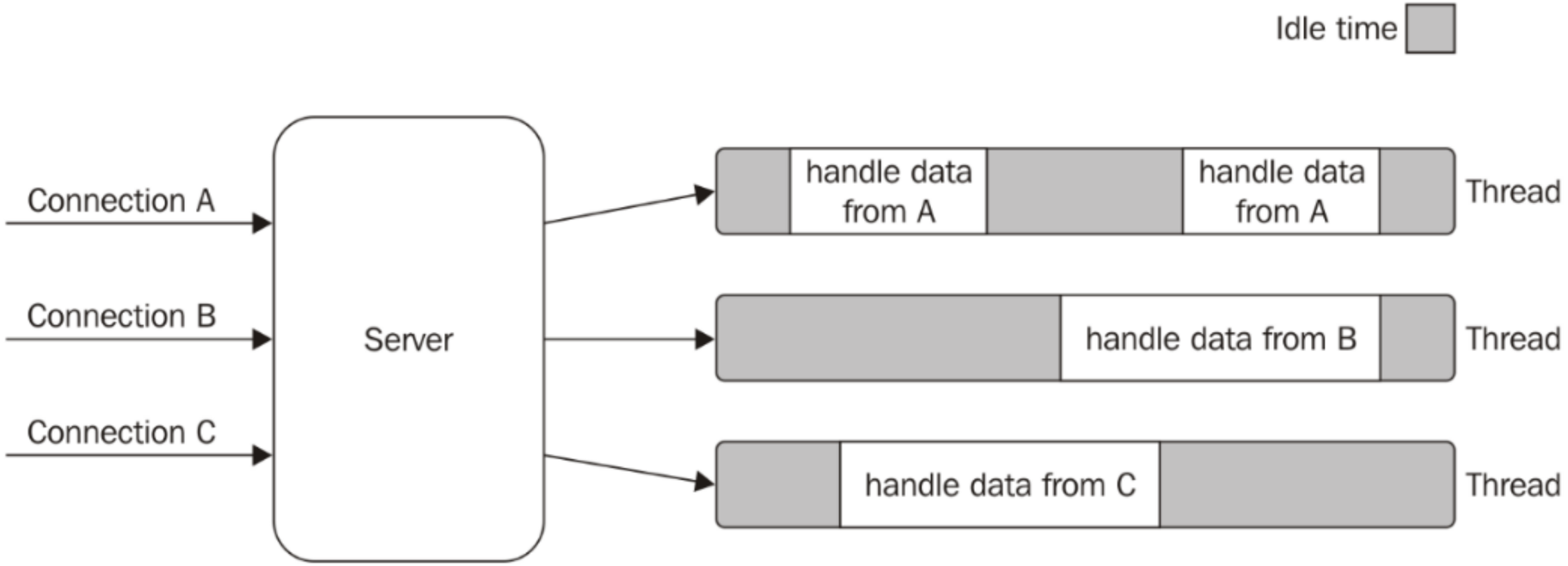
Là où Node.js brille : I/O concurrentes. 🏆

[What is the meaning of I/O intensive in Node.js]



Gérer la concurrence

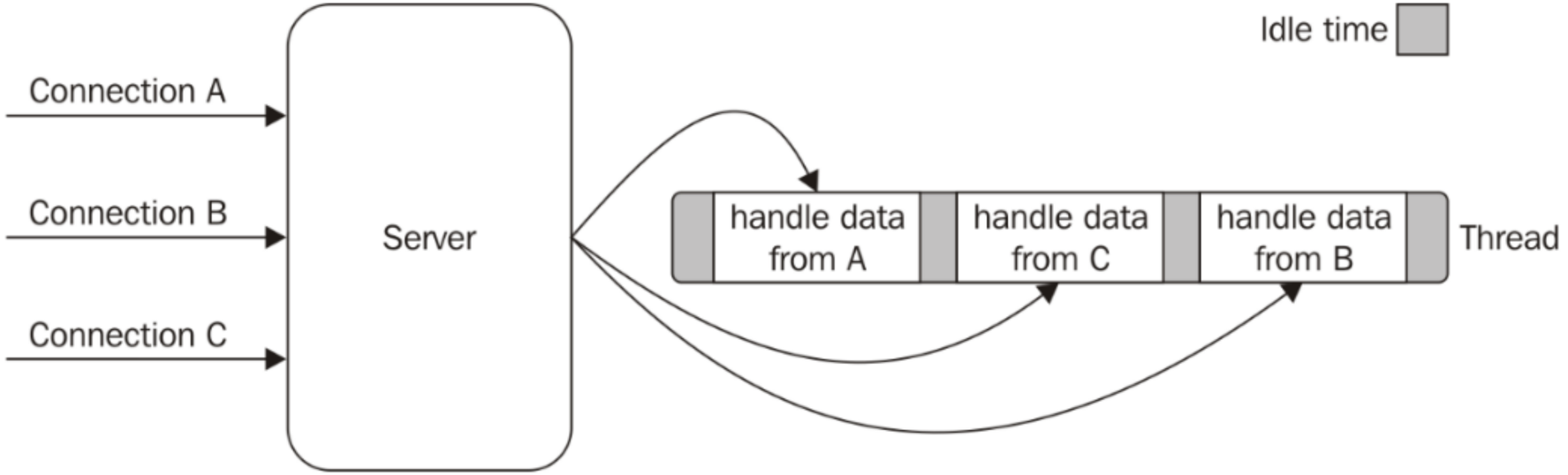
un processus/thread par client



Architecture thread based (source)

🤔 Voir le TP11 de LIF - Système d'Exploitation

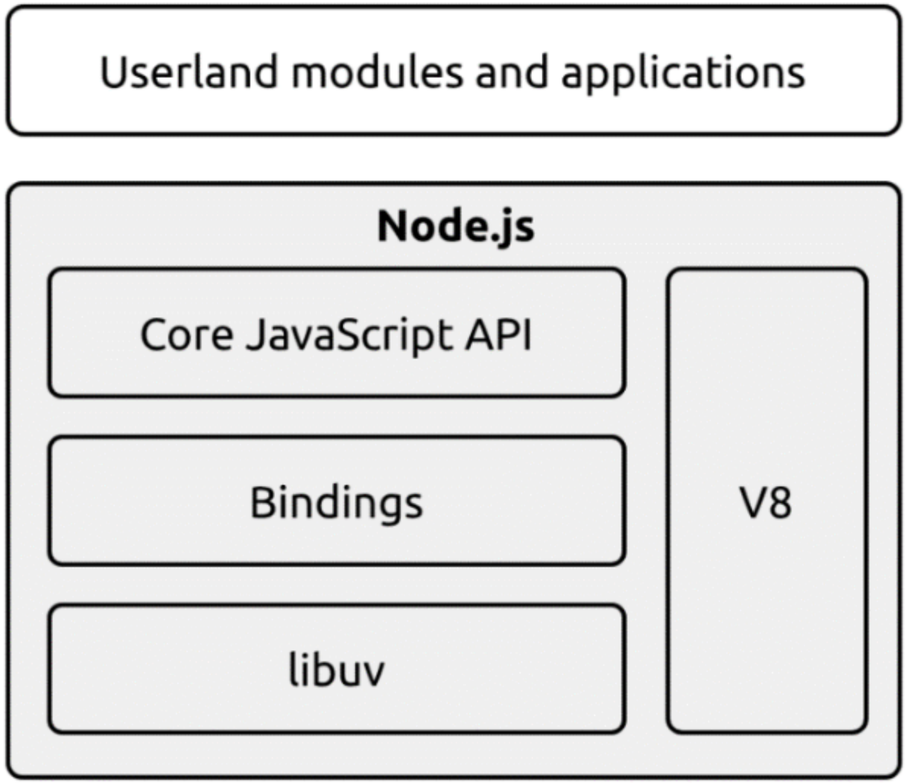
Architecture événementielle



Architecture événementielle worker (source)

💡 Processus/threads de travail auquel sont délégués les traitements via un démultiplexeur.

Boucle d’événements node.js : libuv



libuv.org composant **clef** de Node.js (source)

Principe de libuv

Initialement conçue comme la boucle d'événements de Node.js, la libuv est utilisée comme boucle d'événements en Python (uvloop), Lua (Luvit), …

```
while there are still events to process:
    e = get the next event
    if there is a callback associated with e:
        call the callback
```

- 🔗 Asynchronous TCP/UDP sockets
- 🌐 Asynchronous DNS resolution
- 📁 Asynchronous file and file system operations
- 🧵 Threads, processes
- 🕛 Timers, …

En savoir plus sur les boucles d’événements

📌 Voir aussi :

- Microtasks - javascript.info,
- The Node.js Event Loop, Timers, and process.nextTick() - Node.js,
- When to use queueMicrotask() vs. process.nextTick() - Node.js.

Tâches et micro-tâches

🔁 Il y a plusieurs queues d'événements :

- Les tâches à chaque tour de boucle.
- Les micro-tâches en fin de chaque tour :
  - On ne passe à la tâche suivante que lorsque la queue des micro-tâches est vide.

⚙️ API queueMicrotask - MDN, queueMicrotask - Node.js, setImmediate - Node.js, timerPromises.setImmediate - Node.js, et process.nextTick - Node.js.

Exemple tâches / micro-tâches

```
import { setTimeout, setImmediate } from "node:timers";
import { setTimeout as setTimeoutPromise } from “node:timers/promises";

console.info("Start");
setImmediate(() => console.log(0));
queueMicrotask(() => console.log(1));
setTimeout(console.log, 0, 2);
setTimeoutPromise(0, 3).then(console.log);
setTimeout(console.log, 0, 4);
Promise.resolve(5).then(console.log);
console.info("End");
```

💡 Affichera dans l’ordre `Start`, End, 1, 5, `0`, 2, 3 puis 4.

Les phases de la boucle

[The Node.js Event Loop, Timers, and process.nextTick() - Node.js]

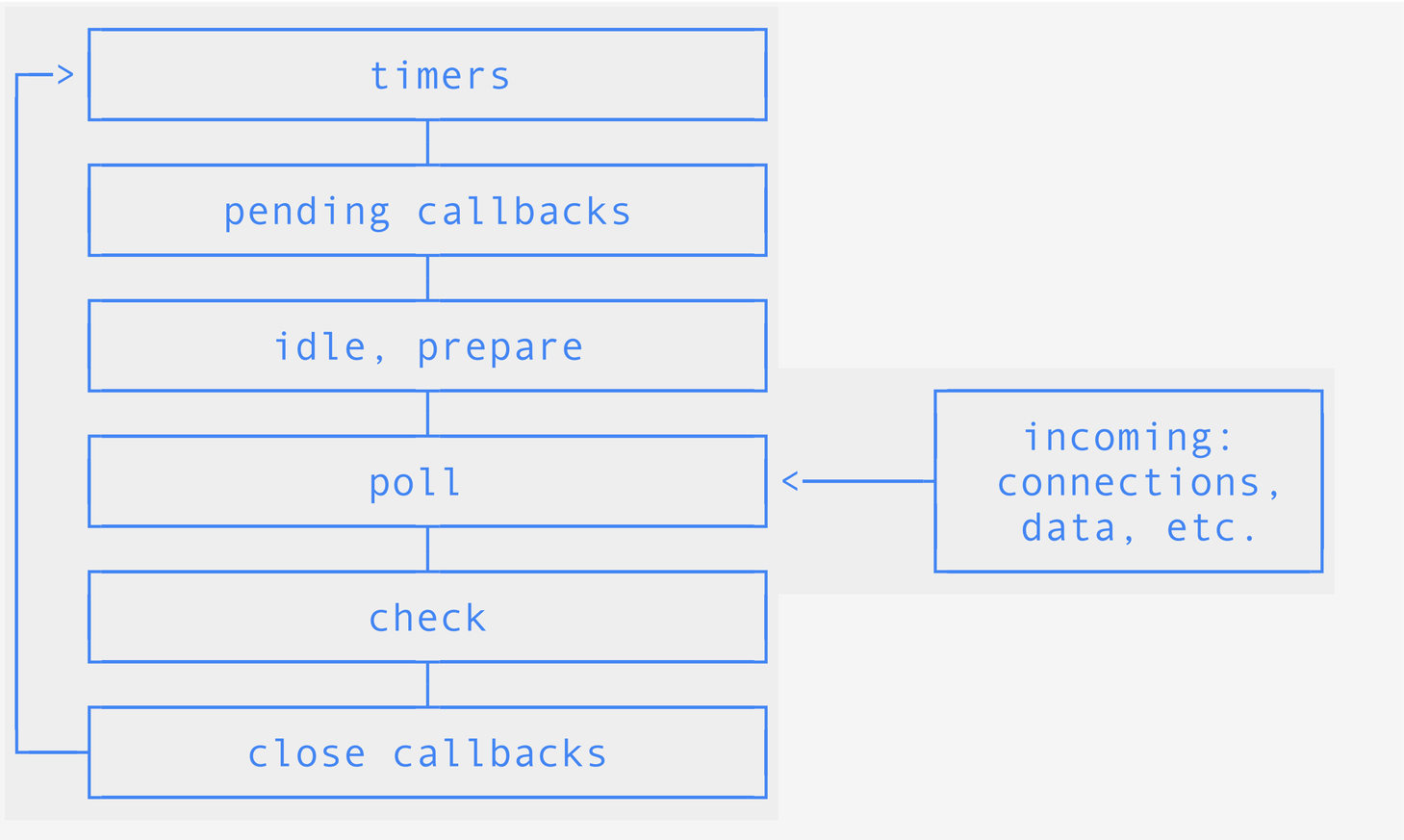


Diagramme de libuv

Event driven architecture

https://nodejs.org/api/events.html#events

"Much of the Node.js core API is built around an **idiomatic asynchronous event-driven architecture** in which certain kinds of objects (called "emitters") emit named events that cause Function objects ("listeners") to be called."

Event driven architecture

Toute l’architecture Node.js est asynchrone, orientée événements :

Concurrence coopérative native (et non préemptive)

Performant pour les applications I/O intensive

- Dont les serveurs Web

Valable pour toute l’API Node.js

- fichiers, streams, socket, crypto, process
- 💡 tout est asynchrone

Différence avec LIF- prog. Concurrente

On choisit quand suspendre les exécutions

- Programmatiquement, ce n’est pas l’OS

Programmation multithread limitée :

- Le moteur de la boucle est mono-thread
- Les worker threads sont cachés par la libuv
- Threads possibles via Web Workers (MDN)
  - Hors périmètre LIFWEB

Exemple ping-pong

exemple-ping-pong.js

```
import { EventEmitter } from "node:events";
/* ... */
const emitter1 = new EventEmitter();
emitter1.last = hrtime.bigint();

emitter1.on("ping", async (value, time) => {
  console.info(`[1] received ${value}@+${time - emitter1.last}`);
  emitter1.last = time;
  emitter2.emit("ping", value, hrtime.bigint());
});
/* ... idem emitter1 */
emitter1.emit("ping", 0, hrtime.bigint());
```

```
emitter1.emit("ping", 0, hrtime.bigint());
// [1] received 0@+18809
// [2] received 0@+295990871
// ...
```

Design pattern Observer

The Observer pattern defines an object (called subject) that can notify a set of observers (or listeners) when a change in its state occurs. (Node.js Design Patterns)

📖 Un des 23 designs patterns du Gang of Four, classé behavioural pattern, central de la conception des événements Node.js (et navigateur).

Design pattern Observer

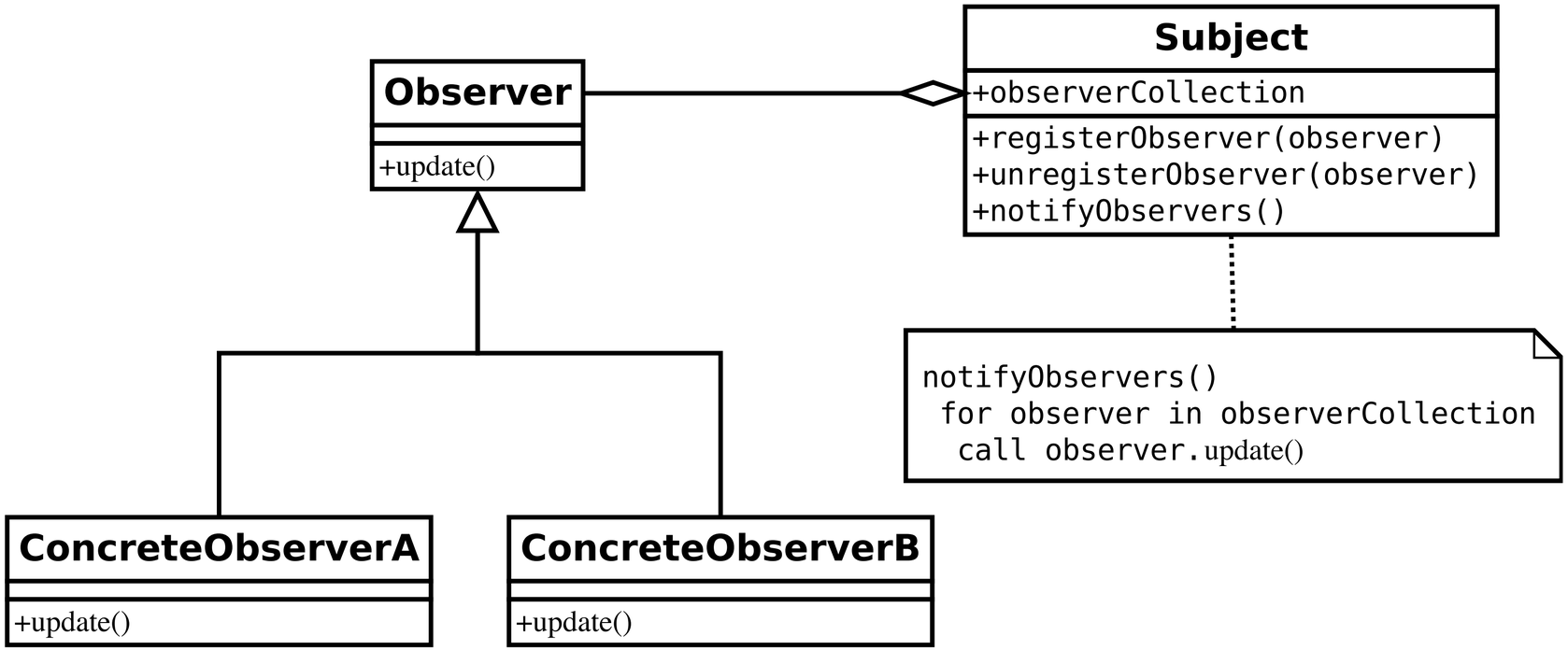


Diagramme de classes UML (Wikipedia)

EventEmitter / EventTarget

Node.js implémente nativement le patron Observer avec les classes

- `EventEmitter` (doc)
- `EventTarget` (doc)

Correspondance avec l'UML

- On enregistre `update()` via un *callback*
  - `registerObserver = addEventListener`
  - `notifyObservers = dispatchEvent`

EventEmitter

- natif Node.js
- v0.1.26
- e.on(...)
- e.emit(...)
- ❎ héritage
- ❎ bubbling
- event : string

EventTarget

- Web API (DOM/navigateurs)
- v14.5.0
- t.addEventListener(...)
- t.dispatchEvent(...)
- ✅ héritage
- ✅ bubbling
- Event : classe

📌 Node.js EventTarget vs. DOM EventTarget.

```
const ee = new EventEmitter();
const listenEmit = (id) => (value) => console.log(`[${id}]: ${value}`);
ee.on("ping", listenEmit("A"));
ee.emit("ping", 42);

const et = new EventTarget();
const listenTarget = (id) => (event) => console.log(`[${id}]: $
{event.detail.value}`);
et.addEventListener("ping", listenTarget("A"));
// ici avec un CustomEvent
et.dispatchEvent(new CustomEvent("ping", { detail: { value: 42 } }));
```

📄 node-target-vs-emitter.js.

👉 `EventTarget` s'aligne sur le navigateur et le standard.
Voir Prefer EventTarget over EventEmitter.

CPS ou EventEmitter

📄 node-callback-vs-emitter.js

```
// callback style CPS
function helloCallback(callback) {
  setTimeout(() => callback(undefined, "hello world"), 100);
}
helloCallback((error, message) => console.log(message));

// callback style event
import { EventEmitter } from "node:events";
function helloEvents() {
  const eventEmitter = new EventEmitter();
  setTimeout(() => eventEmitter.emit("complete", "hello world"), 100);
  return eventEmitter;
}
helloEvents().on("complete", (message) => console.log(message));
```

💡 `helloCallback()` et `helloEvents()` sont fonctionnellement équivalents :

- CB moins élégant si différents événements ;
- CB naturellement appelé une seule fois ;
- Plusieurs handlers possibles sur le même événement.

👉 EventEmitter apporte une abstraction sur Continuation Passing Style (CPS).

Exemple serveur TCP echo

📄 node-tcp-echo.js

```
import net from "node:net";
function handleConnection(socket) {
  socket.on("error", (error) => console.error(`error...`));
  socket.on("end", () => console.debug(`closed...`));
  socket.on("data", (chunk) => socket.write(`server ${chunk}`));
  socket.write("Bonjour !\n");
}

const server = net.createServer();
server
  .on("connection", handleConnection)
  .on("listening", () => console.debug(`listening...`))
  .listen(1337);
```

Serveur & JS

Plan

1. Rappels
2. Requêtage
3. Node.js
4. Node package manager (npm)

NPM

npm : Node Packet Manager.

pNpM et Yarn sont des successeurs.

npm est à Node.js ce que *pip* est à Python, avec un support natif des environnements type venv .

```
npm --version
# 8.16.0
npm init
# question interactives
npm install slugify
# added 1 package in 2s
cat main.js
# import slugify from "slugify"
# console.log(slugify("C'est un test ♥ !"));
node main.js
# C'est-un-test-love-!
```

Décrire son environnement

package.json

```
{
  "name": "cm5_exemples",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "DEBUG=app nodemon server.js"
  },
  "dependencies": {
    "dotenv": “^10.0.0"
  },
  "devDependencies": {
    "eslint": "^7.32.0",
    "nodemon": "^2.0.12"
  }
}
```

La définition du projet et dépendances, voir docs.npmjs.com dépendances.

Avec le fichier package.json précédent :

```
npm install
# added 296 packages in 2s

npm run dev
# > cm4_exemples@1.0.0 dev
# > cross-env DEBUG=app nodemon server.mjs

# [nodemon] 2.0.19
# [nodemon] to restart at any time, enter `rs`
# [nodemon] watching path(s): *.*
# [nodemon] watching extensions: js,mjs,json
# [nodemon] starting `node server.mjs`
#   app Server listening at http://127.0.0.1:5000/ +0ms
```

npx the package runner

https://www.npmjs.com/package/npx

npx permet d'exécuter des commandes locales au dossier (dans `node_modules/`) sans les installer globalement

> **Executes *`command`* either from a local *`node_modules/.bin`*, or from a central cache, installing any packages needed in order for *`command`* to run.**

`npx` est implicitement utilisé par les scripts du `package.json`.

Exemple

```
/tmp/npx❯ npx cowsay "Hello world"
Need to install the following packages:
  cowsay@1.5.0
Ok to proceed? (y) y

 _____________
< Hello world >
 -------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
/tmp/npx❯ ll
/tmp/npx❯
```

Systèmes de modules

Plusieurs co-existent

Common JS (CJS)

- historique
- `const maLib = require('laLib');`
- `module.exports = { ... };`
- par défaut pour `.js`

EcmaScript (ESM) :

- standard JS
- `import maLib from 'laLib';`
- `export default { ... };`
- par défaut pour `.mjs`

💡 Préférer les modules ESM 💡

Serveur & JS

Plan

1. Rappels
2. Requêtage
3. Node.js
4. Node package manager (npm)

Bonnes pratiques

https://12factor.net/

Configuration

- Fichier .env et cross-env (GitHub)

Logging

- Exemple https://getpino.io/

Reverse-proxy (a.k.a. front)

- https://www.nginx.com/

Sécurité

- HTTPS, compte non privilégié