hillel IT school

Lesson 39

План заняття

- Node js.
- Node modules.
- Debug.
- ENV/ARGV.
- Nodemon.
- Event Loop.
- Timers.
- Event emitter.

Node js

Node js (https://nodejs.org/en) це середовище для виконання js на вашої машині або на сервері. Виконання js на сервері це основна мета nodejs. У 2009 році це було розроблено командою на чолі з Раеном Далем. Вони взяли двигун v8 та зробили огортку над ним щоб комп'ютери розуміли цю мову.

Біль людиноподібна документація з nodejs - https://nodejsdev.ru/

Для роботи з асинхронними операціями Nodejs використовує бібліотеку libyvi (https://docs.libuv.org/en/v1.x/). libuv — це багатоплатформна бібліотека підтримки з фокусом на асинхронному вводі-виводі. В основному він був розроблений для використання Node.js, але його також використовують Luvit, Julia, uvloop та інші. Написана на c++.

Nodejs надає:

- Середу запуска js за допомогою v8
- Модель подій вводу та виведу за допомогою libuv
- Готові модулі для роботи за середовищем за допомогою npm

Node js

Nodejs з почтку створення не використовувала стандартне версіонуваня на принципами semver (https://semver.org/lang/uk/). Але з 2013 вони перейши на стандарт.

Це привело до відділеня дочірньої компаніі тому сьогодні є 2 відділеня роброки:

- Nodejs (https://uk.wikipedia.org/wiki/Node.js) - працював над стабілізацією
- Io.js (https://ru.wikipedia.org/wiki/Io.js) - працював над фічами

Але у 2014 прийшов Nodejs foundation який почав процювати над повною розробкою Nodejs. Ця організація почала працювати над обєдненням io.js та Node js. У 2015 io.js повернувся до Noddejs і стали існувати версію nodejs 4 і т.д.

Так створився стандарт у Node js.

Коли ви перейдете до офіційного сайту Nodejs ви побачите 2 версії lts та Current.

LTS – long term support - довга підтримка

Current - поточна

Подробиці тут https://nodejs.org/en/about/previous-releases

Node js

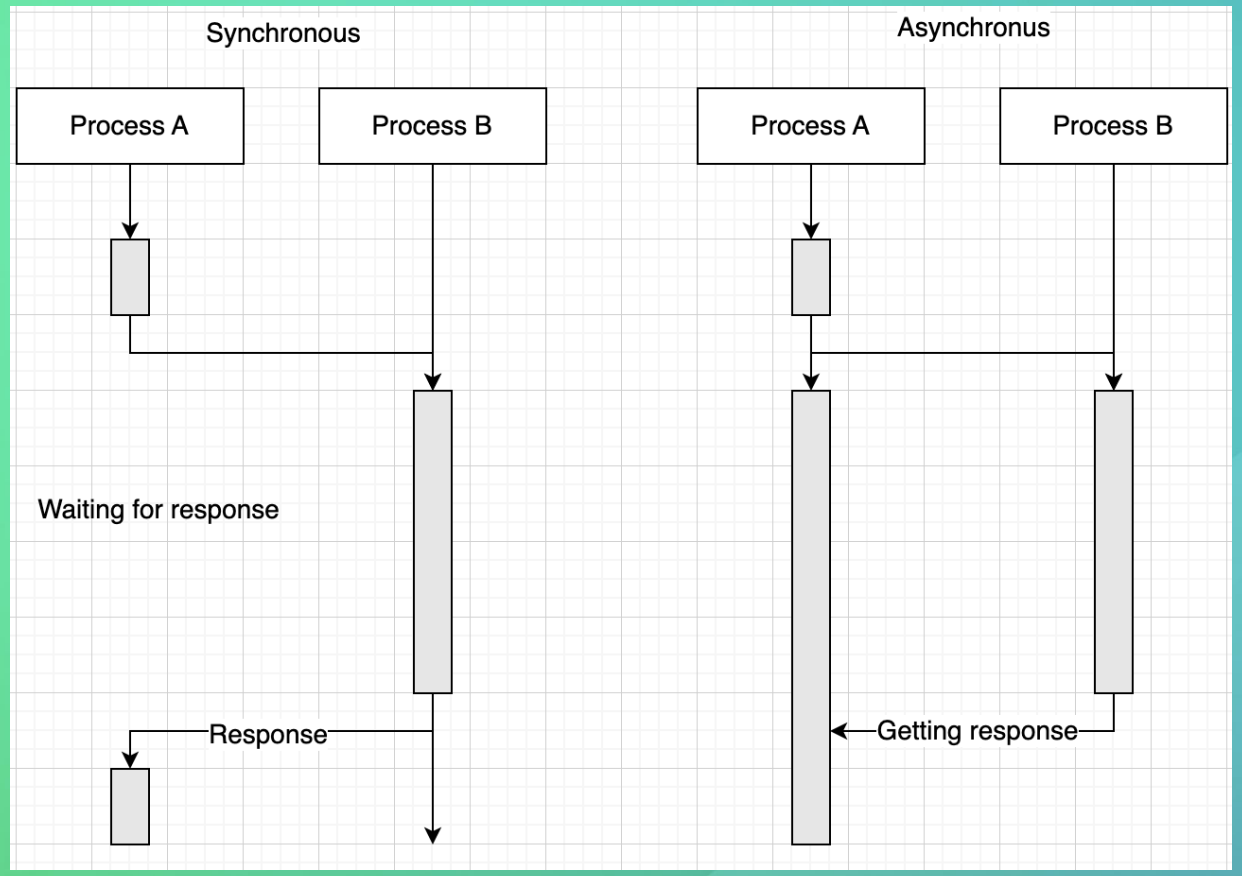
ECMAScript - це стандарт як повинен поводитись js. Двигун v8 розробляється Google для Chrome та Chromium але не завжди туди потрапляють нові фічи. Побачити що зараз підтримується можна тут https://node.green/

Розробкою стандартом займається комітет tc39 https://tc39.es/ru/ . Ви можете запропонувати свою фічу до комітету та можливо вона зайде до специфікації.

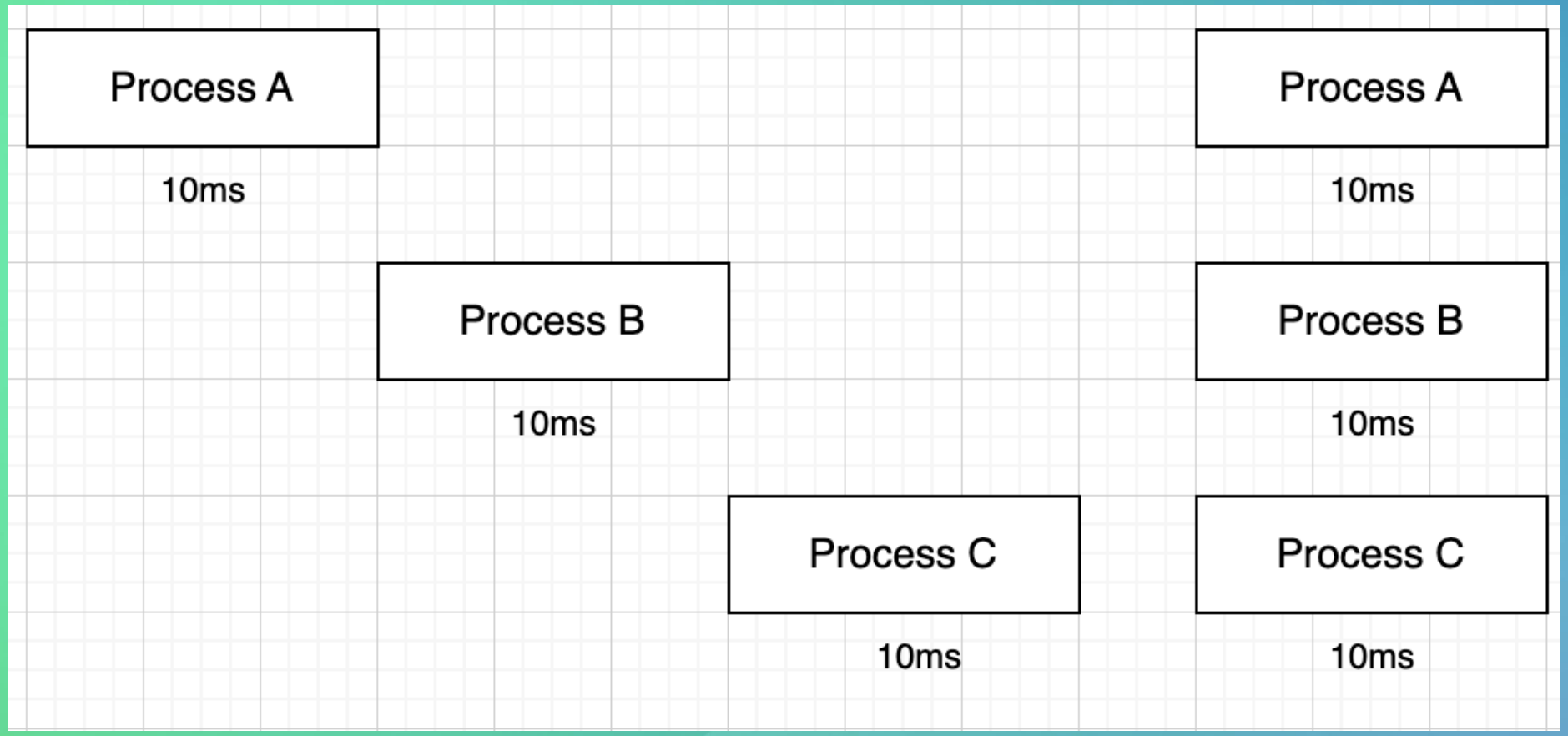
Gihub https://github.com/tc39

Node js event loop

Node js є однопоточним



Node js event loop



Node js

Можна встановити одну версію за офіційно сайту (я рекомендую LTS)

Або встановіть собі на машину nvm https://github.com/nvm-sh/nvm . Це дасть вам можливість мати декілько версій nodejs на машині та переключатись між ними

Якщо ви на працюєте mac то встановіть собі спочатку homebrew https://brew.sh/

Потім вже встановлюйте nvm https://formulae.brew.sh/formula/nvm#default

Якщо ви використовуєте oh-my-zsh то вам потрібно буде сконфігорувати .zshrc

Перевірити версію ноди можна у терміналі за допомогою команди node –v

Для тих хто не хоче робити усього цього є більш складний підхід. Ви можете за дпомогою Docker створювати свої власні середовища у середині яких розгортати оточення яке вам потрібно та займатись оркестрацією.

Node js modules

Будь який проект складається з багатьох файлів. У ноді такий підхід називається модульним. Ви можете використовувати сучасний модульний підхід за допомогою розширення mjs але це ще не стандарт для базового використання, тому ми будем використовувати common js. Різниця тільки в синтаксисі імпортування.

Node js package.jon

Для обслуговування вашого проекта вам потрібно ініціалізувати package.json. Можна його прописати вручну, або визвати у терміналі

npm init

Важливо! entry point це файл який буде головною точкою входу до застосунку. Ц нашому випадку це app.js

Node js debug

Ви можете дебажити ноду двома шляхами:

Перший варіант через Crome (не рекомендований підхід)

Для цього:

- Йдемо о браузеру та пишемо chrome://inspect
- Зупиняємо код за допомогою breackpoint. Для цього використовуємо ключьове слово debugger;
- Запускаємо ноду у режимі debug деталі тут https://nodejs.org/en/learn/getting-started/debugging;

Другий за допомогою IDE (рекомендований підхід)

- Виставляємо breackpoints
- Запускаємо IDE у режимі debug

Node js env

Об’єкт process є глобальним, який надає інформацію про поточний процес Node.js і контролює його. Як глобальний, він завжди доступний для програм Node.js без використання require(). Деталі тут https://nodejs.org/dist/latest-v8.x/docs/api/process.html

Властивість process.env повертає об’єкт, що містить середовище користувача. Деталі тут https://nodejs.org/dist/latest-v8.x/docs/api/process.html#process_process_env

Щоб додати якусь змінну до вашої поточної системи (юніт системі) вам потрібно у терміналі написати ключ цюєї змінної та значення. Наприклд:

USER=iKot

Якщо ви використаєе команду export USER=iKot то ця змінна запишиться до системи і буде існуваи там доки система не перезавантажитсья

Для ого щоб забрати значення цієї змінної звертаємось до process за ключем. Наприклад

process.env.USER

Node js env

Щоб записати набір якихось env змінних для проєкту ми можемо використовувати файл .env. Потім для використання йього файлу нам необходно завантажити бібліотеку dotenv (https://www.npmjs.com/package/dotenv) та завантажити її до проєкту.

Також її потрібно буде сконфігорувати у методі .config().

Щоб ваші змінні існували та ви могли їх використовувати на будь яких системах існує бібіотека cross-env https://www.npmjs.com/package/cross-env

Потім для сету env змінної (наприклад у scripts) ми використовуємо приклад

"start": "cross-env MODE=dev node app.js",

Node js env

Якщо вам потрібно запустити файл з якимись аргументами. Вам потрібно вказати їх при запуску його файлу. Наприклад у scripts:

"start": "cross-env MODE=dev node app.js --arg=1"

Забрати фргументи виконання можна з process.argv. Зетультатом буде масив де ви знайдете усі аргументи виконання. Деталі тут https://nodejs.org/dist/latest-v8.x/docs/api/process.html#process_process_argv

Для того що гарно використовувати аргументи завантажде бібліотеку yargs (https://www.npmjs.com/package/yargs)

Вантажимо це у файл

```
const yargs = require('yargs/yargs')

const { hideBin } = require('yargs/helpers')

const argv = yargs(hideBin(process.argv)).argv
```

Далі в нас буде можливість доставати аргумети з argv просто за ключем

Event loop

Всі синхронні операції та виконая коду в нас виконується за допомогою двигуна v8, також він обслуговує callStack.

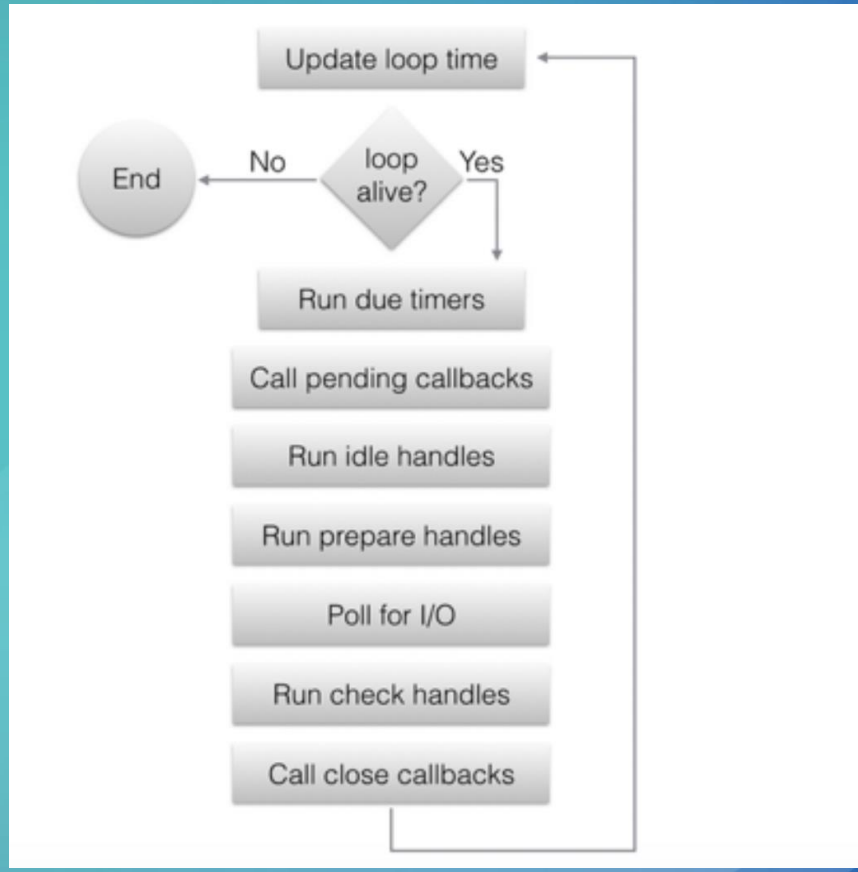
Все що стосується асинхроного вводу та виводу обслуговується libuv

Event loop — це "сутність яка перехоплює та обробляє зовнішні події та конвертує їх у функції зворотного виклику". Тобто, виклики I/O – це певні точки, в яких Node.js може перемикатися від одного запиту до іншого. При зверненні до I/O ваш код зберігає callback, і повертає контроль назад, в середу виконання Node.js. Збережений callback буде викликано пізніше, коли всі необхідні дані будуть отримані.

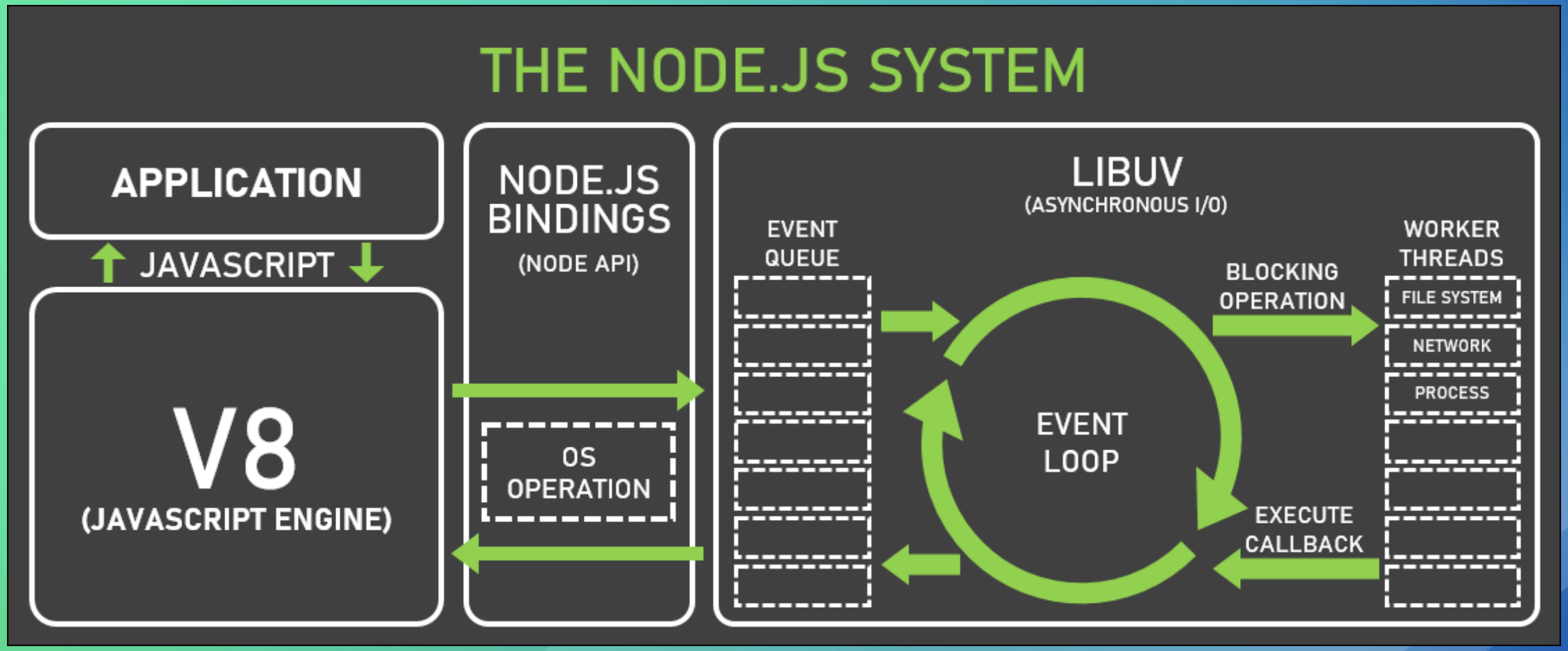
Event loop (libuv)

1. З купи витягується обробник таймера з найменшим часом і порівнює це значення з часом виконання подійного циклу, якщо час таймера менше, цей таймер зупиняється.
2. Запускаються функції очікуваних callbacks
3. Запускаються функції-запуску callbacks, але вони не мають жодного відношення до I/O. Фактично, це якісь внутрішні підготовчі дії, які відбуваються перед тим, як розпочинати виконання зовнішніх операцій (мається на увазі I/O).
4. Зовнішні операції I/O
5. Те саме, що і на 3, але є можливість вплинути з NodeJS (setImmediate - негайне виконання після зовнішньої операції I/O)
6. Функція обходить список обробників, що закриваються, і намагається завершити закриття для кожного. Якщо обробник має спеціальний callback на закриття, то, по закінченню, запускається цей зворотний виклик.
7. Запустіть цикл ще раз, якщо він живий.

Тут можете подивитись презентацію http://latentflip.com/loupe



Event loop


THE NODE.JS SYSTEM
APPLICATION
JAVASCRIPT
V8
(JAVASCRIPT ENGINE)
NODE.JS
BINDINGS
(NODE API)
OS
OPERATION
LIBUV
(ASYNCHRONOUS I/O)
EVENT
QUEUE
EVENT
LOOP
BLOCKING
OPERATION
WORKER
THREADS
FILE SYSTEM
NETWORK
PROCESS
EXECUTE
CALLBACK

Timers

У nodejs існують setInterval, setTimeout, setImmediate.

Вони виконуються лише після основного потоку!

libuv не може завершити процес, доки є активний таймер.

Event emitter

EventEmitter (https://nodejs.org/en/learn/asynchronous-work/the-nodejs-event-emitter) - це модуль, що сприяє комунікації між об'єктами в Node. Він є ядром асинхронної подієво-керованої архітектури та її суть у генеруванні іменованих подій, які призводять до виклику раніше зареєстрованих слухачів.

Багато вбудованих в Node модулів успадковують від EventEmitter.

EventEmitter – це не завжди асинхронність!

Для того щоб розпочати роботу з евентамі, потрібно шмпортуввати модуль та створити сутність EventEmitter

```
const EventEmitter = require('events')
```

```
const eventEmitter = new EventEmitter();
```

Для того щоб підписатись на подію нам потрібно навісити прослуховувач події. Наприклад

```
eventEmitter.on(id, () => {})
```

Щоб визвати подію нам потрібно викликати emit у нашого eventEmitter. Наприклад:

```
eventEmitter.emit(id, payload)
```

Дякую за увагу