



# Sommaire

1. Qu'est-ce que Node.JS
2. Utilisations
3. Gestionnaire de paquet et package.json
4. EventLoop
5. Plan de travail

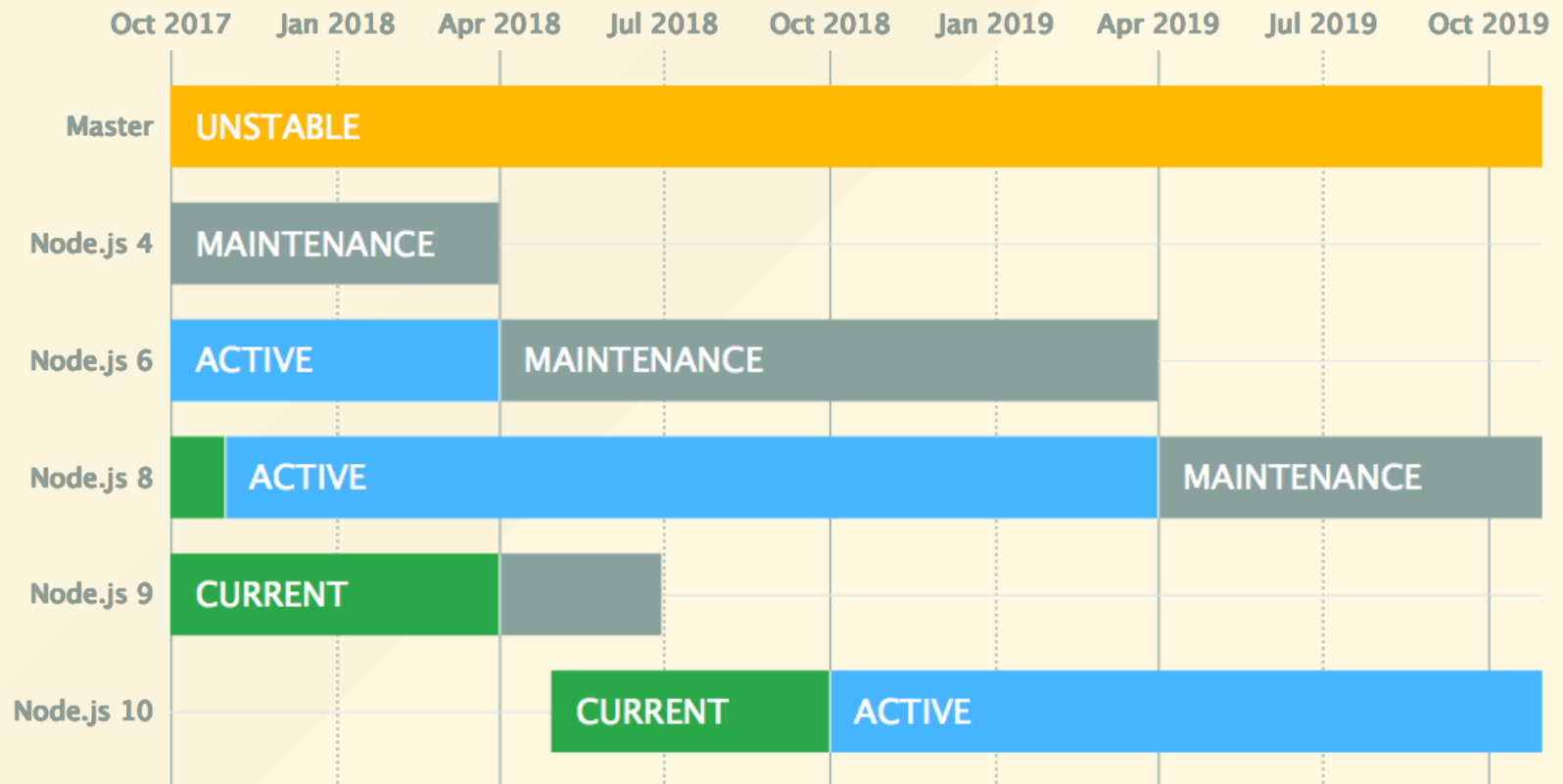
# Qu'est-ce que Node.JS

# Qu'est-ce que Node.JS

- Javascript côté server
- Moteur V8 de Webkit
- Open Source
- Multiplateforme (ou presque)
- Mono thread
- Asynchrone
- I/O non bloqués

# Qu'est-ce que Node.js

- Versions standards et LTS



# Qu'est-ce que Node.js

Release	LTS Status	Active LTS Start	Maintenance End
v0.12.x	<b>End-of-Life</b>	-	2016-04-01
4.x	<b>Maintenance</b>	2015-10-01	April 2018
5.x	No LTS		
6.x	<b>Active</b>	2016-10-18	April 2019
7.x	No LTS		
8.x	<b>Pending</b>	2017-10-31	December 2019

# Qu'est-ce que Node.js

- Deux types de version :
  - LTS : Support étendu : Ce sont les versions les plus stables
  - Les autres : versions dites "beta" (test de nouvelles fonctionnalités, ...)

# Qu'est-ce que Node.JS

- serveur http

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```



# Utilisation

- scripts bash
- compilations diverses (babel, ...)
- applications multiplateformes desktop (electron, NW)
- applications mobiles (react-native, ionic, native-script)
- serveurs web (express, hapi, koa)
- serverless (chromeless, ...)
- ...

# Gestionnaire de paquets

# Gestionnaire de paquets

- NPM
  - officiel
  - [npmjs.org](https://npmjs.org)

```
npm install -S hapi
```

- Yarn
  - facebook
  - plus rapide

```
yarn add hapi
```

# Gestionnaire de paquets

- NPX
  - Permet de lancer des binaires installés via npm
  - Peu importe s'ils sont au sein du projet ou global

```
./node_modules/.bin/eslint  
/usr/bin/eslint
```

```
npx eslint
```

# Package.json

- coeur de l'application
- paquets utilisés
- commandes de lancement
- version Node.JS requise

```
{
  "private": true,
  "name": "Test APP",
  "description": "This is an example package.json",
  "version": "0.0.0-this-does-not-matter",
  "license": "MIT",
  "scripts": {
    "dev": "npx poi",
  },
  "devDependencies": {
    "less": "2.7.2",
    "less-loader": "4.0.5",
  },
  "dependencies": {
    "lodash": "4.17.4",
    "moment": "2.18.1",
  }
}
```

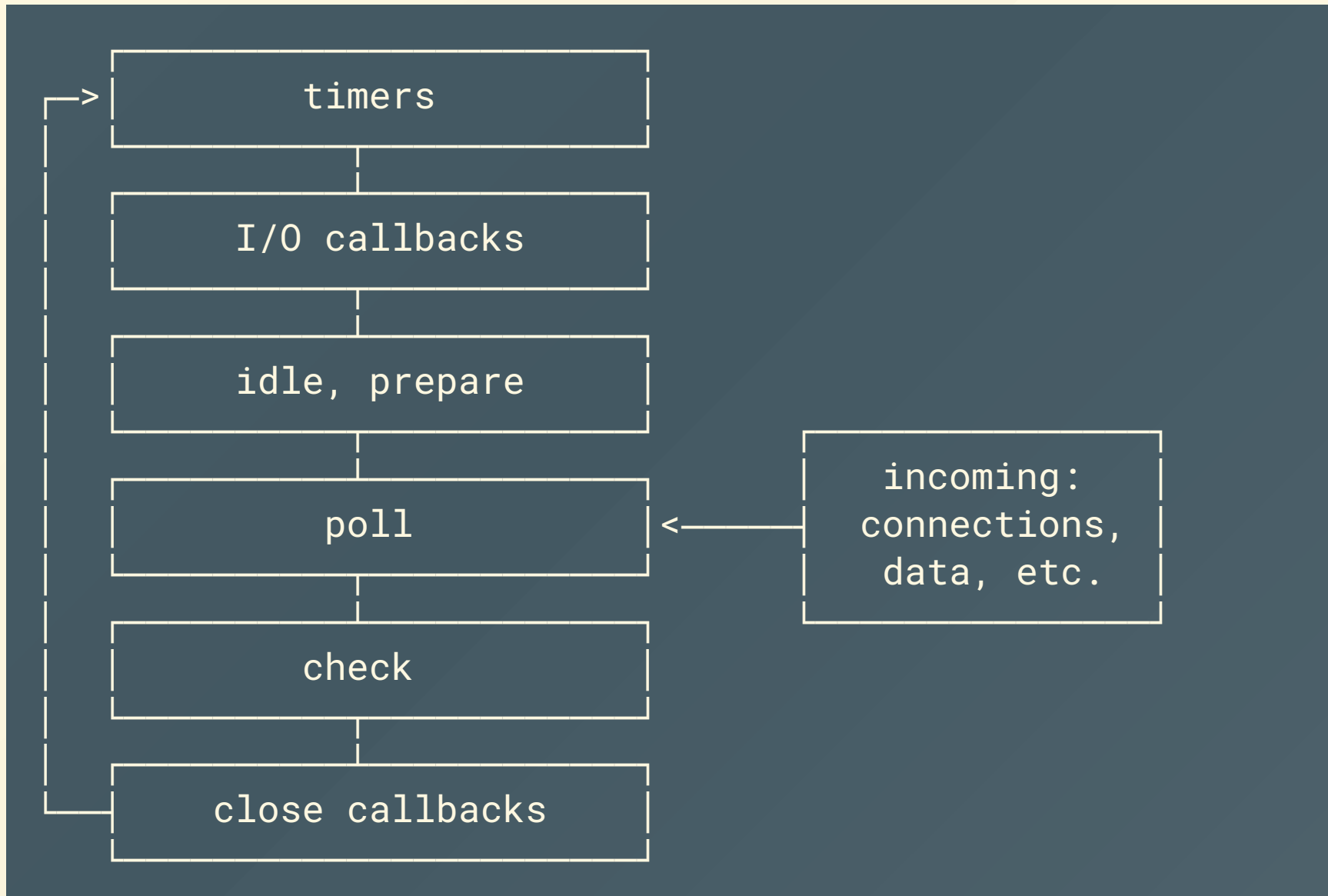
# Package.json

- **scripts** : scripts à lancer via `npm run`
- **devDependencies** : Dépendances servant uniquement pour la phase de développement
- **dependencies** : dépendances du projet
- **private** : Si à `true`, le projet ne sera jamais publié sur npm si vous lancez `npm publish` par mégarde

# EventLoop



# EventLoop



# EventLoop (détails)

- **timers** : lance les callbacks programmés par `setTimeout` et `setIntervale`
- **I/O callbacks** : lance tous les autres callbacks
- **idle, prepare** : utilisé par le coeur de Node.JS
- **poll** : Créer de nouveaux événements d'entrée/sortie. Le cas échéant, peut bloquer l'exécution du script.
- **check** : invoque les callbacks de `setImmediate`
- **close callbacks** : `socket.on('close')`, ...

# EventLoop (détails)

- Permet l'asynchrone
- Possibilité de forcer l'exécution au prochain passage avec `process.nextTick()`
- Fonctions avec callback pas toujours asynchrones selon ce qu'elles font
- Promesses toujours Asynchrones

# EventLoop (examples)

```
const funcSync = callback => {  
  let fields = {  
    foo : 'foo'  
  };  
  callback(fields);  
};  
  
console.log('a');  
funcSync(field => { console.log(field); });  
console.log('fin');
```

```
a  
{foo: "foo"}  
fin
```

# EventLoop (examples)

```
const funcSync = callback => {  
  let fields = { foo : 'foo' };  
  process.nextTick(() => {  
    callback(fields);  
  });  
};  
  
console.log('a');  
funcSync(field => { console.log(field); });  
console.log('fin');
```

```
a  
fin  
{foo: "foo"}
```

# EventLoop (exceptions)

- **Attention** : tous les callbacks ne sont pas asynchrones de base :

```
const a = [1, 2, 3, 4];  
  
a.forEach((value) => {  
    console.log(value);  
});  
  
console.log('after forEach');
```

- Les fonctions des types de base (String, Object, Array, ...) sont toujours synchrones.

# Les événements

# Les événements

- Utilisés au travers de la classe `EventEmitter`

```
const EventEmitter = require('events').EventEmitter;  
const myEventNamespace = new EventEmitter();
```

- Callbacks effectués à des moments précis
- Appels via `event.emit('eventName', ...params)`
- Interceptés via `event.on('eventName', callbackFunc)`



# **Les promesses**

# Les promesses

- Asynchrones
- Capturent les erreurs
- Transformation fonction callback en promesses via `util.promisify`

```
const {promisify} = require('util');
const fs          = require('fs');
const readFileAsync = promisify(fs.readFile);

readFileAsync(filePath, {encoding: 'utf8'})
  .then(text => console.log(text))
  .catch(err => console.error(err))
```

# Les promesses

- Norme A+ : Utilisation de libs compatibles
- Manipulation des Arrays en asynchrone :
  - `Promise.map`
  - `Promise.each`
- Fonctions étendues compatibles A+ au sein de la library `bluebird`

# Les promesses : Gestion des erreurs

```
myfunctionWithErrorsPrimisify()  
  .then(() => console.log('fini sans erreurs'))  
  .catch(TypeError, err => { console.error(err) })  
  .catch(ReferenceError, err => { console.error(err) })  
  .catch(err => {  
    console.error('Error non gérée');  
    console.error(err);  
  });
```

**Attention** : La stack d'erreur se base au catch et non le lieu réel de l'erreur si jamais vous oubliez de la capturer avec la promesse

# Socket.io

- Websocket : connexion "permanente"
- Communication client/serveur via les événements
- Aucun timeout
- L'événement appelé peut avoir un callback

```
socket.emit('myeventWithCallback', (result) => { });
```

- Gestion de Rooms
- Permet l'envoi d'events en broadcast

# Mise en production

- Serveurs Web :
  - PM2 : Permet de lancer le serveur comme un service et de le relancer automatiquement
  - Docker : via un système d'orchestration (Rancher, ...)
- Module :
  - NPM et Yarn
- Scripts :
  - Services functions (AWS lambda, ...)

# Plan de travail : Outils

- Utilisation de NVM avec la dernière version LTS de Node.JS en date (8.X)
- Utilisation de Webstorm (licences gratuites étudiants)
- Repository Git pour le suivi du projet
- TD et TP séparés en 2 parties :
  - Apprentissage via [nodeschool.io](https://nodeschool.io)
  - Mise en pratique

# Plan de travail

- TD : Introduction
- TP1 : Analyse, Debug et Eslint
- TP2 : GIT et NPM
- TP3 : Hapi
- TP4 : Hapi : Swagger et Joi
- TP5 : Mongoose et migrations
- TP6 : Envoi d'email
- TP7 : [Socket.io](https://socket.io/) et microservice d'email



# Plan de travail

- Objectif :
  - Comprendre Node.JS
  - Comprendre les websockets
  - Utiliser Hapi.JS
  - Sait faire une API REST
  - Interagir avec MongoDB
  - Approche microService