NODE JS

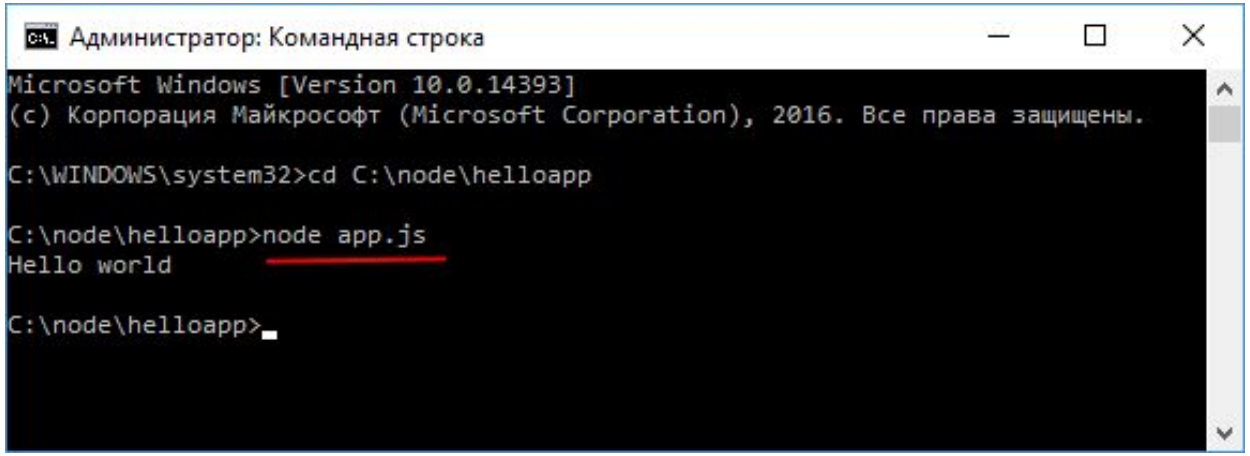
Что такое Node JS

- Node.js представляет среду выполнения кода на JavaScript
- Предназначен для создания серверных приложений на языке JavaScript
- Платформа для создания веб-приложений

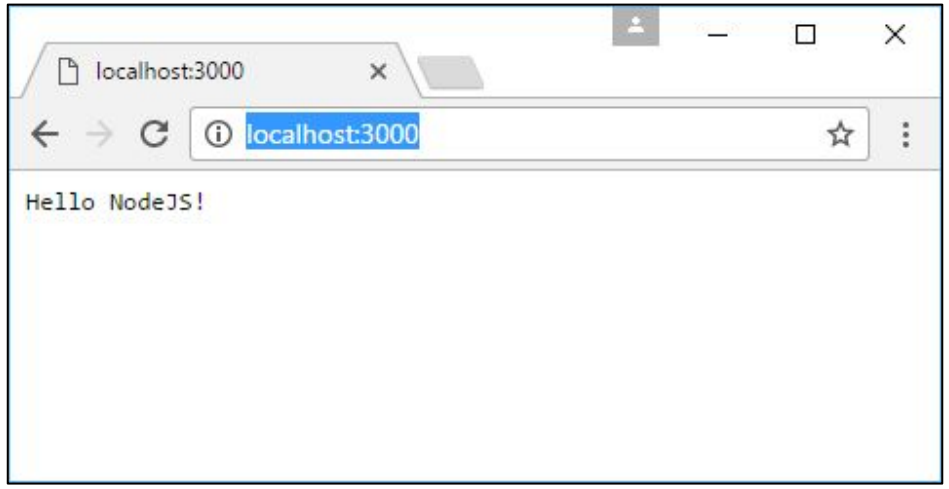
Скачать Node JS можно на официальном сайте https://nodejs.org/en/

Инструменты разработки

- WebStorm
- VS Code
- Atom
- REPL(Read Eval Print Loop) представляет возможность запуска выражений на языке JavaScript в командной строке или терминале.
- Console.log(“Hello world!”);

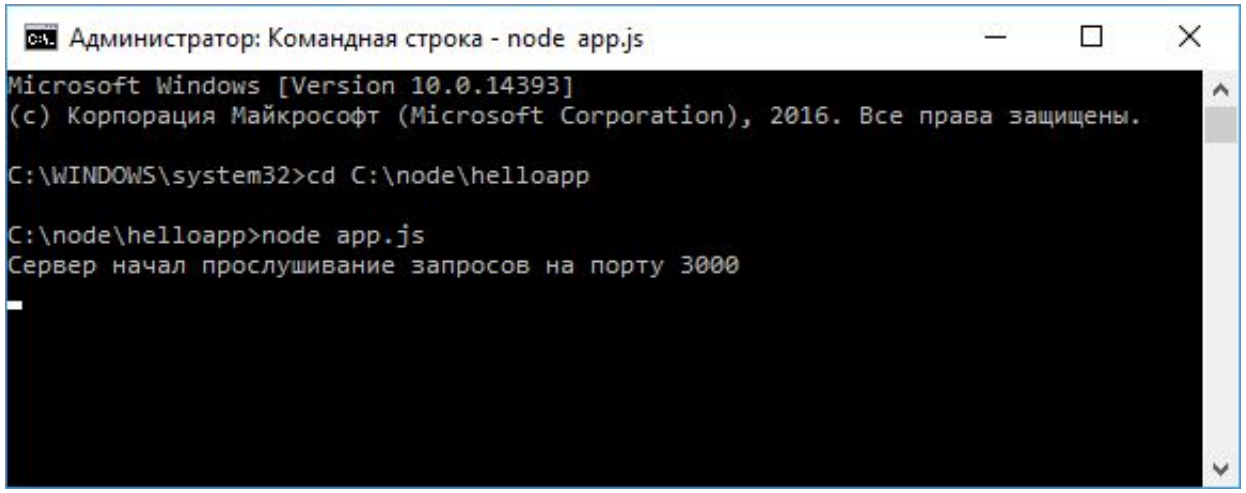


Простейшее Приложение




```
1const http = require("http");
2http.createServer(function(request,response){
3
4    response.end("Hello NodeJS!");
5
6}).listen(3000, "127.0.0.1",function(){
7    console.log("Сервер начал прослушивание запросов на порту
83000"); });
```

```
cd C:\node\helloapp
```



Основы работы с Node JS

- Модуль представляет блок кода, который может использоваться повторно в других модулях.
- Для загрузки модулей применяется функция require(). К примеру, в первом приложении из предыдущей темы для получения и обработки запроса был необходим модуль **http.**
- Подобным образом мы можем загружать и использовать другие встроенные модули. Например, используем модуль os, который предоставляет информацию об окружении и операционной системе:

```
1const os = require("os");
2// получим имя текущего пользователя
3let userName = os.userInfo().username;
4
5console.log(userName);
```

Основы работы с Node JS

- NPM(Node Package Manager) – менеджер для автоматизации установки и обновления пакетов
- Для нас менеджер npm важен в том плане, что с его помощью легко управлять пакетами.

```
npm install express
```

- Express представляет легковесный веб-фреймворк для упрощения работы с Node.js.
- После установки express в папке проекта modulesapp появится подпапка **node_modules**, в которой будут хранится все установленные внешние модули.

Package.json

```
{
  "name": "modulesapp",
  "version": "1.0.0"
}
```

- Для более удобного управления конфигурацией и пакетами приложения в npm применяется файл конфигурации **package.json**.
- Удалить node_modules.

```
npm install express --save
```

- --save указывает, что информацию о пакете нужно добавить в package.json

```
{
  "name": "modulesapp",
  "version": "1.0.0",
  "dependencies": {
    "express": "^4.14.0"
  }
}
```

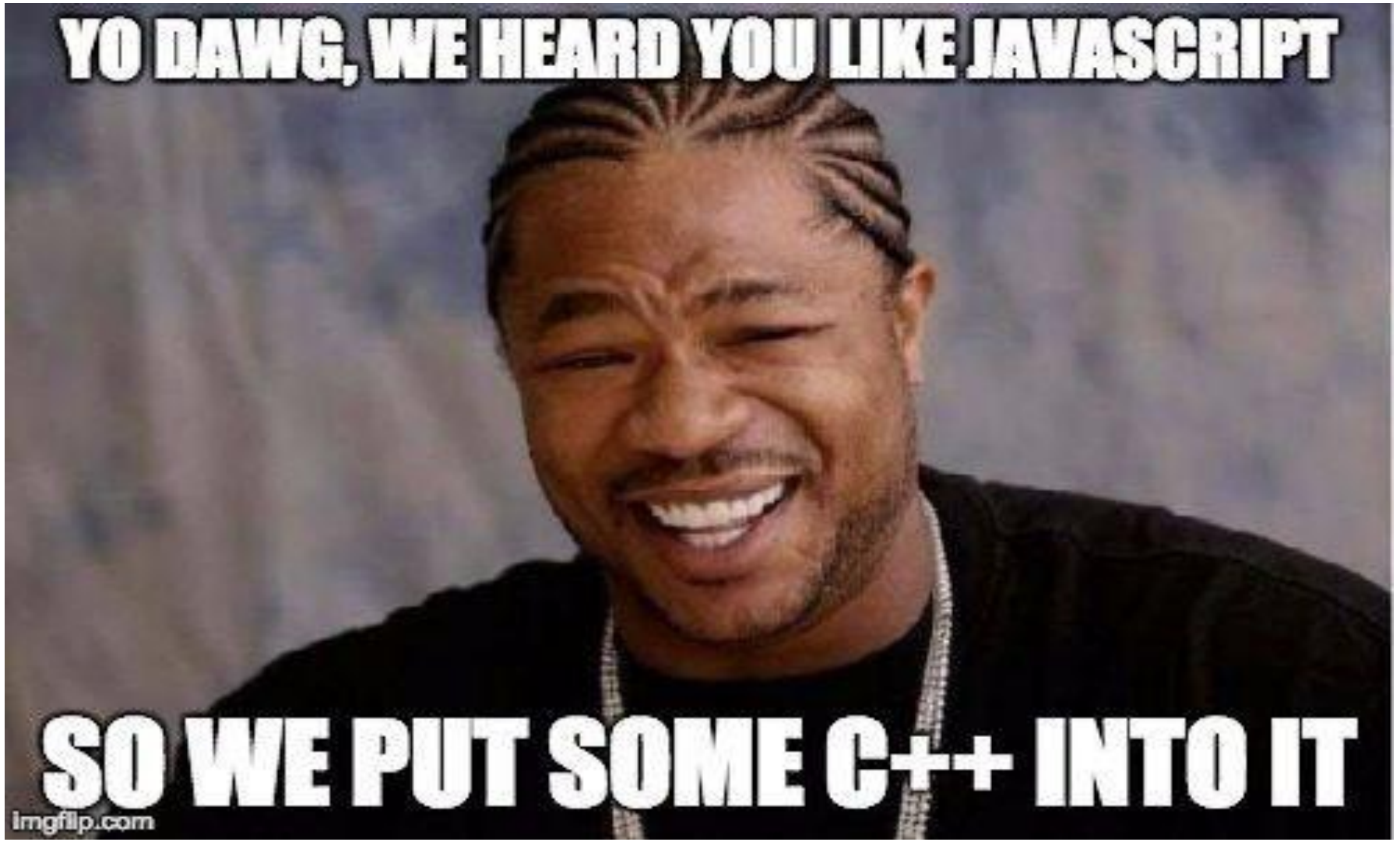
Файл *package.json* играет большую роль и может облегчить разработку в различных ситуациях. Например, при размещении в разных репозиториях нередко мы ограничены выделяемым дисковым пространством, тогда как папка *node_modules* со всеми загруженными пакетами может занимать довольно приличное пространство. В этом случае удобнее разместить основной код проекта без node_modules. В этом случае мы можем определить все пакеты в файле package.json, а затем для загрузки всех пакетов выполнить команду

```
npm install
```

C++ и Node JS

Или как ~~освятить~~ ускорить JS

КДПВ


YO DAWG, WE HEARD YOU LIKE JAVASCRIPT
SO WE PUT SOME C++ INTO IT
imgflip.com

Зачем такое делать?

1. Получить доступ к ~~нормальным~~ нативным библиотекам из JS (биндинги)
2. Повысить производительность отдельных участков кода
3. Просто пофаниться

Как это сделать?

- Напрямую использовать V8 API
- Использовать NAN
- Использовать N-API + C++

V8 API

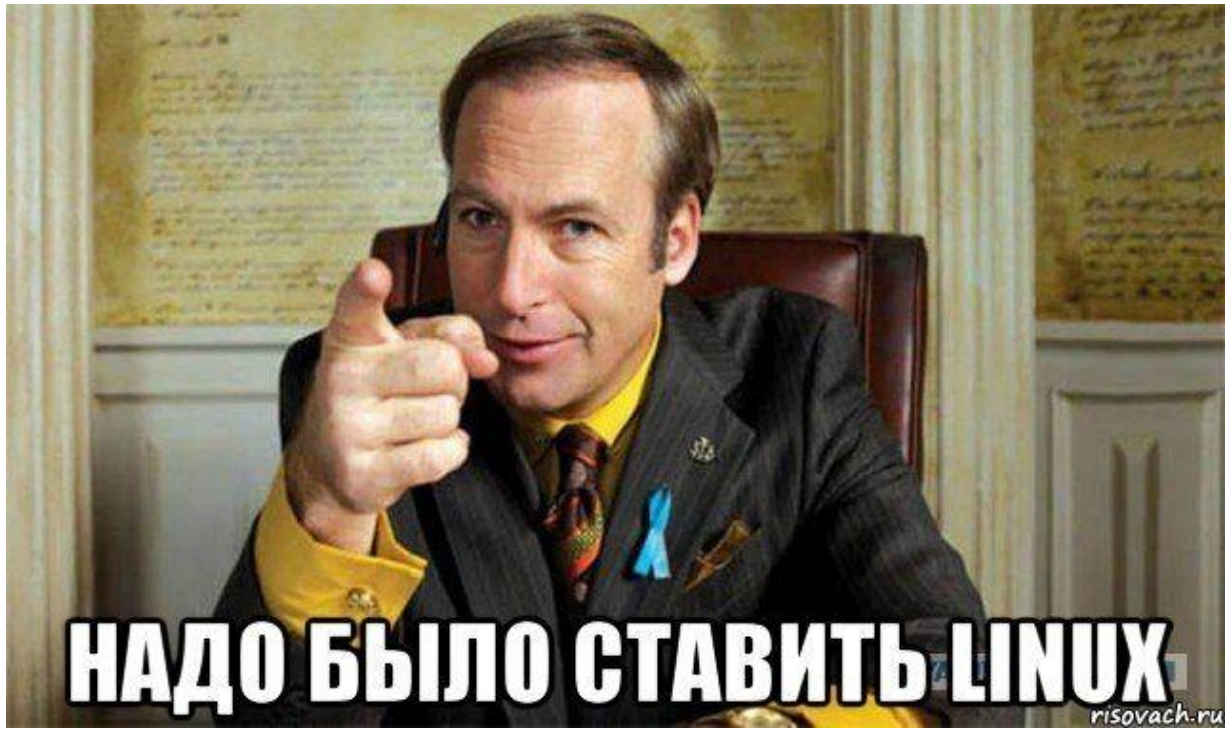
- Наиболее старый и примитивный способ
- Сложности совместимости между версиями
- #include <node.h> и всё. К сожалению.
- Никто в здравом уме не использует

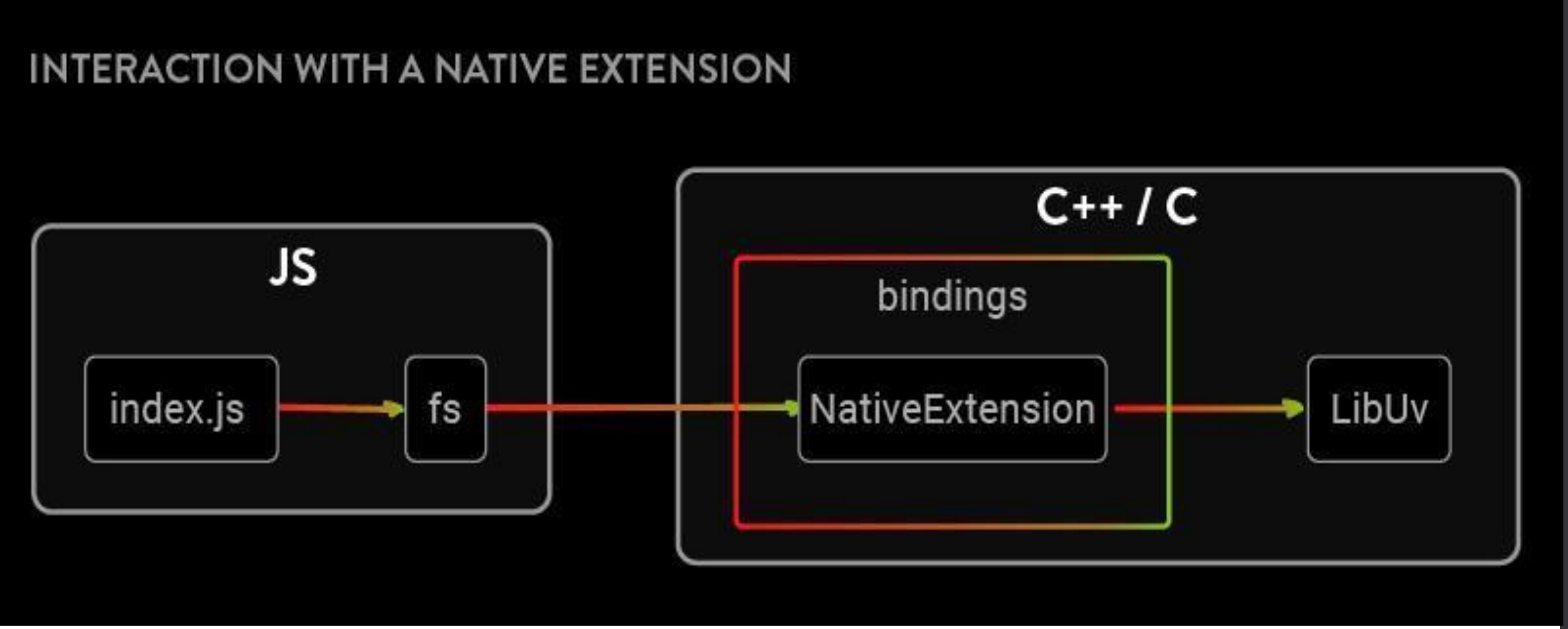
Native Abstractions for Node.js (NAN)

- npm install nan
- NAN != NaN
- Поддерживают совместимость с разными версиями V8
- Наиболее используемый способ в настоящее время

Требования к системе

1. Компилятор C++ (gcc, clang, msvc, etc.)
2. Python 2.7
3. make
4. node-gyp
5. NAN





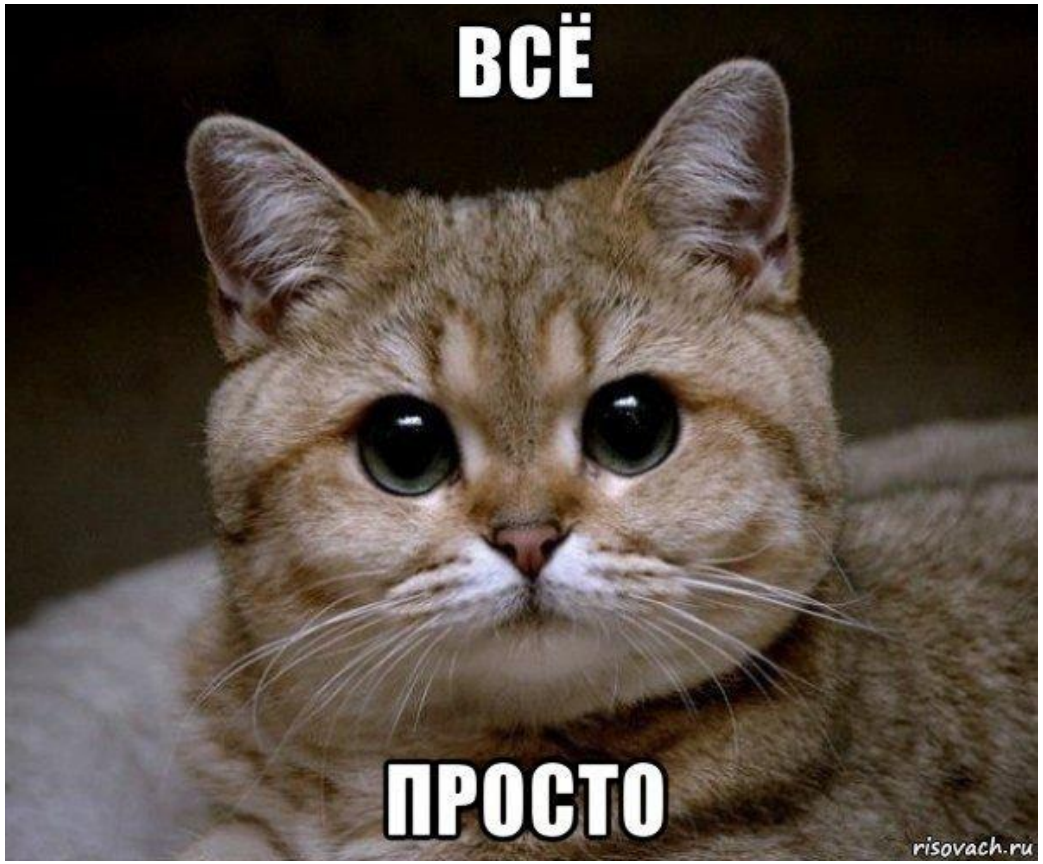
Доступ к дисковой подсистеме не входит в возможности JavaScript или V8. Libuv даёт возможности асинхронного выполнения кода. Однако, пользуясь Node.js, можно писать данные на диск и читать их. Именно здесь на помощь приходят нативные расширения. Модуль fs реализован средствами C++ (у него имеется доступ к диску), он даёт нам методы, вроде writeFile и readFile, которые можно вызывать из JavaScript.

bindings.gyp

```
{
  "targets": [
    {
      "include_dirs": [
        "<!(node -e
        \"require('nan')\")"
      ],
      "target_name": "addon",
      "sources": [ "main.cpp"
      ]
    }
  ]
}
```

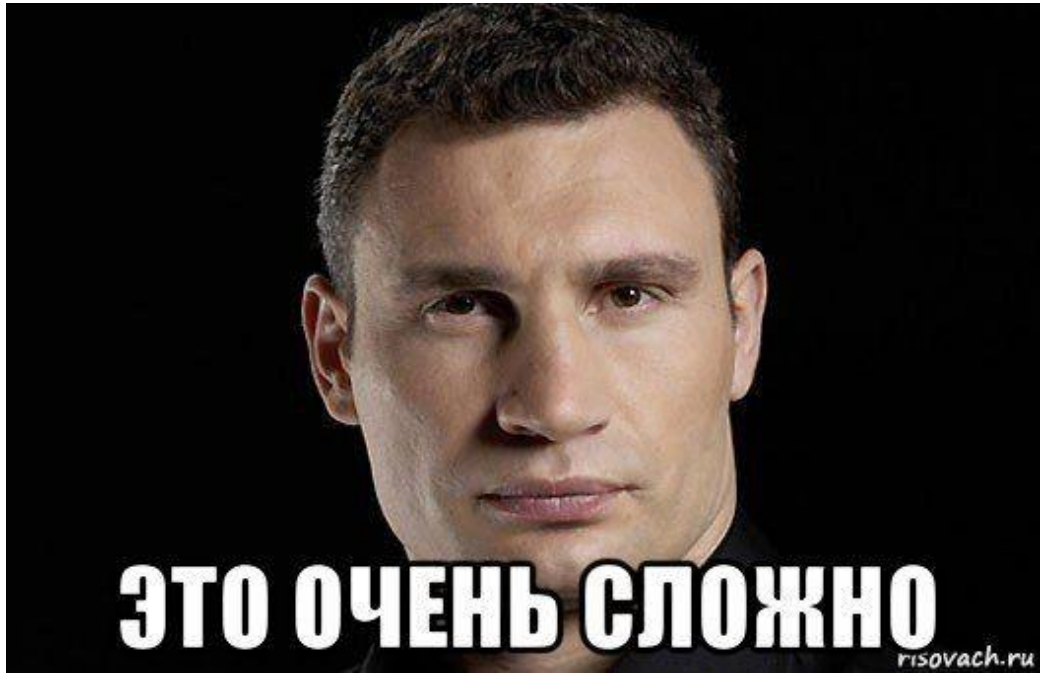
Сборка && запуск

- npm run compile
- node index.js
- …
- Profit!



Бенчмарки? А не будет!

- Сильно зависит от кейса
- Результаты могут разниться очень сильно: от тотального проигрыша JS до выигрыша (при определённых условиях)
- Бенчмаркинг - краааааааааааайне сложная штука



N-API

- Новый подход (с Node 10)
- Цель: сделать единое API и ABI, которое не будет ломаться от релиза к релизу
- Цель покруче: сделать эту вещь не просто для V8, а сразу для всех движков!

Вывод

- На C++ ~~нужно~~ можно писать везде :-)
- Иногда это даже может пригодиться
- Приходится писать много boilerplate (но это дело исправляется)
- Всё больше и больше стабильности в API и ABI с каждым днём
- **Можем** выиграть в производительности
- Не стоит заниматься таким без крайней необходимости

Ссылки

- https://nodejs.org/api/addons.html
- https://github.com/nodejs/node-gyp
- https://github.com/nodejs/nan
- Best OS in the world:
  - https://gentoo.org/
  - https://www.ubuntu.com/
  - https://getfedora.org/
  - http://www.linuxfromscratch.org/lfs/
  - https://www.archlinux.org/





Спасибо за внимание!

И используйте только православные технологии.