

INTRODUCING NODE.JS



Agenda

2

- Install Node.js
- Understand the power of Node.js
- Discuss Node.js architecture
- Use nvm & npm
- Understand module system
- Control flow strategies
- General topics

What

3

- ❑ Node.js is a server side JavaScript platform
- ❑ Built on Chrome's V8 engine
- ❑ Is open source
- ❑ Single threaded
- ❑ Event-driven, non blocking I/O
- ❑ Developed in 2009 by Ryan Dahl
- ❑ Supported by Joyent

Node.js as a Web server

4

- According to w3techs.com Node.js has only 0.4% market share
- Still, it gains more and more popularity
 - ▣ ~1 million web sites world wide
 - ▣ PHP is ~40 million
- Top web sites
 - ▣ Aliexpress.com
 - ▣ bbc.com
 - ▣ outbrain.com
 - ▣ flickr.com

Node.js as BFF

5

- Node.js cannot easily replace existing server side infra written in Java/.NET
- Common scenario is to put Node.js at the front of Java/.NET → Backend for Frontend
- Usually is controlled by Front End engineers
 - ▣ Thus allowing the developer to push JavaScript code to the server
 - ▣ Improve client side performance

Node.js as Development Tools

6

- This is where Node.js really shines
- Extreme echo system of development tools
 - ▣ Build tools – Webpack, Gulp, Grunt
 - ▣ Compilers – Typescript, Babel
 - ▣ Testability – Selenium, Jasmine, Mocha
 - ▣ Desktop applications – VSCode, GithubDesktop

When should we use?

7

- Node.js is great when most work is I/O
- Think of a web server. The “hard” work relates to
 - ▣ HTTP → Networking I/O
 - ▣ Database → Networking I/O
 - ▣ File system
- The server is more of a controller/facade

When NOT to use

8

- ❑ Heavy server-side computation
 - ❑ Can offload the “hard” work to background processes
 - ❑ Can use threads (not common)
- ❑ Direct access to OS API is required
 - ❑ Can integrate C/C++ code

Installation

9

- Depends on the OS
- Starts with <https://nodejs.org>
- Amazingly you can just download Node.js as a tar/zip file and start using it
 - ▣ <https://nodejs.org/dist/latest-v8.x/>
- On Windows you may execute **nodevars.bat** which fixes the PATH with
 - ▣ node
 - ▣ npm

NVM

10

- Each Node.js project may be dependent on different Node.js version
- Can resolve that by installing Node.js per project
 - ▣ Less common
- NVM allows managing multiple versions of Node.js at the machine level while having only ONE active version at a time

NVM

11

- Ensure you don't have any previous installation of Node.js
- **nvm list** – Get a list of all installed versions
- **nvm install latest** – Installs latest Node.js version
- **nvm use 9.8.0** – Configure machine to use the specified version

Hello World Sample

12

- Create new main.js file
- Paste the following

```
console.log("Hello Node.js");
```

- From the command line execute

```
node main.js
```

- Can it be simpler ?

Http Server Sample

13

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

const requestHandler = (req, res) => {
  res.end('Hello Node.js Server!');
}

const server = http.createServer(requestHandler);

server.listen(3000, (err) => {
  if (err) {
    return console.log('something bad happened', err);
  }

  console.log(`server is running`);
});
```

Better abstraction with Express

14

□ npm install express

```
const express = require("express");

const app = express();

app.get("/api/contact", function (req, res) {
  res.json([
    {id: 1, name: "Ori"},
    {id: 2, name: "Roni"}
  ]);
});

app.listen(3000, function() {
  console.log("Server is running");
});
```

Toolings

15

- But what if we just need a simple web server that returns static content from current directory
- No need to re-implement that
- `npm install http-server`
- `node_modules/.bin/http-server`
- A web server is up and running on port 8080 ...

Typescript

16

- ❑ Adds type safety to Node.js
- ❑ `npm install typescript`
- ❑ `npx tsc -init`
- ❑ `npm install @types/node`
- ❑ `npx tsc`

Typescript
generates
compilation error



```
import * as fs from "fs";
```

```
fs.readFile("main.ts", function(err, data: string) {  
  console.log(data);  
});
```


Quick Exercise

17

- ❑ Install nvm
- ❑ Install Node.js using nvm
- ❑ Create a simple HTTP echo server using Express
- ❑ `/api/echo/hello` → Returns “hello”

NODE.JS ARCHITECTURE



Agenda

19

- Discuss Node.js architecture
- Understand main characteristics
- Write some code

Characteristics

20

- ❑ Built on Chrome's **V8** engine
- ❑ Uses **libuv**
- ❑ Single threaded
- ❑ Event-driven
- ❑ Non blocking I/O

V8

21

- ❑ JavaScript engine
- ❑ Compiles JS to native machine code
- ❑ Written in C++
- ❑ Used in Chrome & Node.js
- ❑ Supports Windows, macOS, Linux
- ❑ Can be embedded into C++
- ❑ [Hello world sample](#)

V8 vs. The World

22

- ❑ Same role as Java's JVM or .NET's CLR
- ❑ However, JavaScript is dynamic language
- ❑ Therefore less optimization opportunities
- ❑ V8 profiles code at runtime and optimizes it
 - ▣ Same as Java HotSpot technique
 - ▣ Has two compilers Full-Codegen & Crankshaft
 - ▣ Therefore can be faster than GCC
 - ▣ Shouldn't be faster than Java/.NET
 - ▣ See some [benchmarks](#)

libuv

23

- Multi platform library with focus on asynchronous I/O
- Was developed for use by Node.js
 - ▣ But is now used by others
- Supports all the goodies of Node.js
 - ▣ Event loop
 - ▣ Async TCP & UDP sockets
 - ▣ Async file system operations
 - ▣ IPC
 - ▣ More ...
- Create thread sample

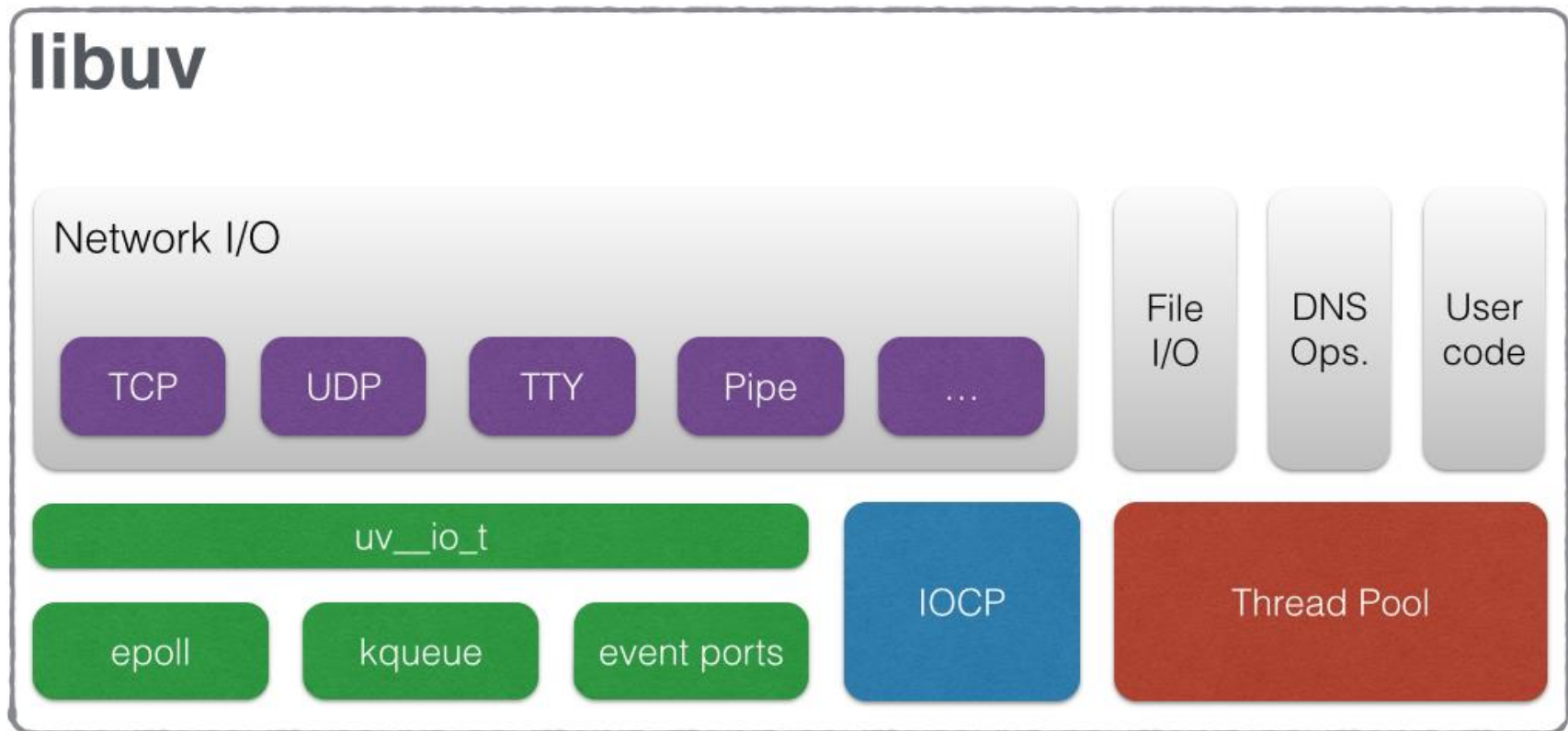
libuv

24

- When possible uses OS asynchronous API
- Surprisingly does not use asynchronous file I/O
 - ▣ Code complexity
 - ▣ Poor APIs
 - ▣ Poor implementation
- Uses thread pool instead

libuv

25



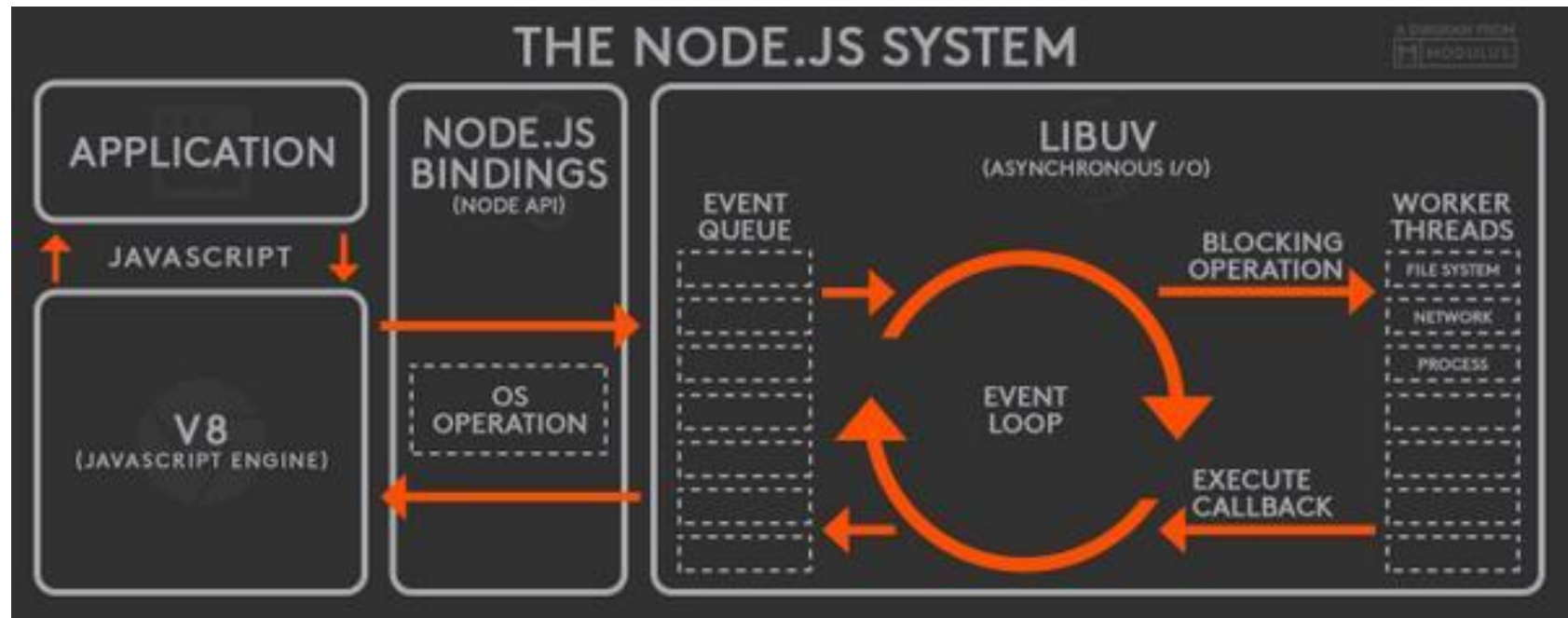
Integrating

26

- Take V8
- Combine it with libuv
- Implement some JavaScript API to be consumed by the application
- And voila ... Node.js

Node.js Architecture

27



Traditional Web Server

28

- ❑ Spawns new thread for each request
 - ▣ May use some kind of thread pool
- ❑ Each thread consumes memory and increases context switching
- ❑ Thread blocks when accessing file system/networking
- ❑ Programmer must synchronize access to shared/static data
 - ▣ Thus increasing even more blocking time

Single Threaded

29

- Only JavaScript code is Single Threaded
 - ▣ NodeJS has multiple worker threads

```
setTimeout(function() {  
  console.log("timeout");  
}, 1000);  
  
console.log("Before");  
sleep(2000);  
console.log("After");  
  
function sleep(ms) {  
  const before = new Date();  
  
  while(new Date() - before < ms);  
}
```



Before
After
timeout

Event Queue

30

- ❑ Continuing with our previous sample
- ❑ What happens after 1000 milliseconds ?
- ❑ A worker thread handles the timer event by putting an appropriate event inside the queue
- ❑ Only when our JavaScript code completes it returns to the **event loop** and fetches the next waiting event

Asynchronous I/O

31

- Node.js uses callbacks to handle async operations
- The function returns immediately and the “real work” executes at the background
- Once completes, an event is pushed to the event queue waiting to be processed by the main thread

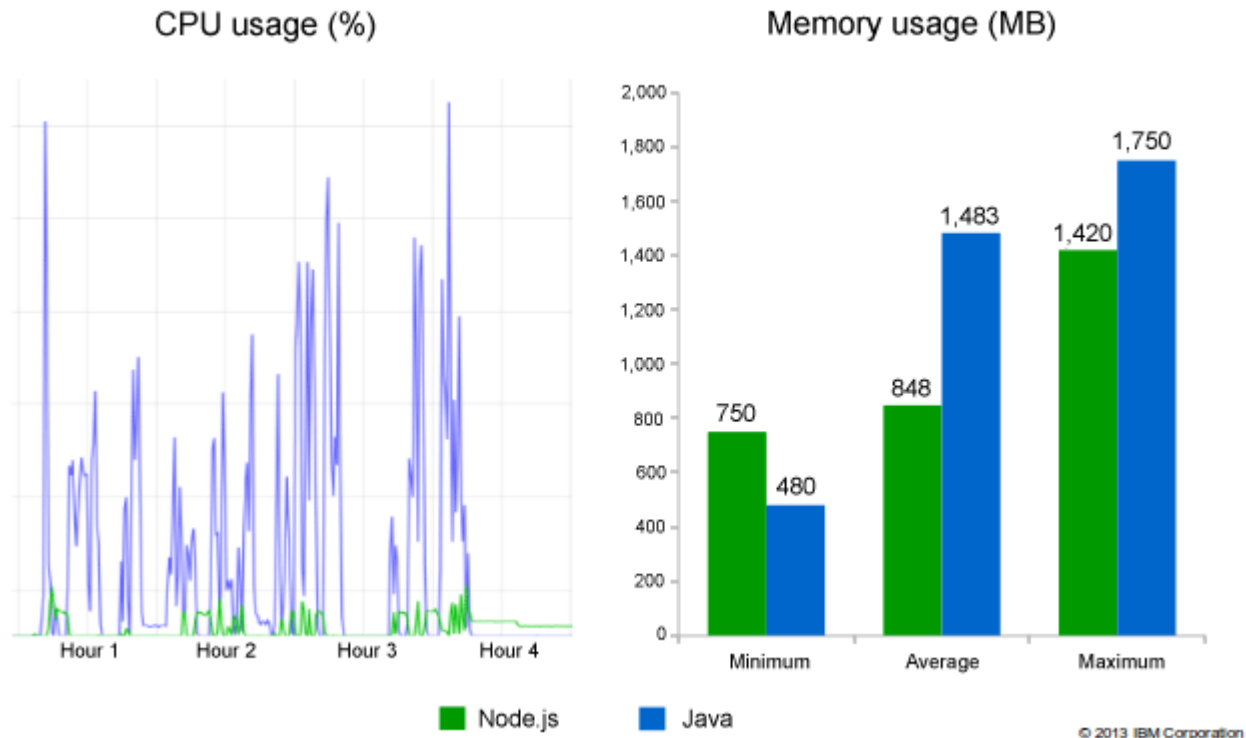
```
const fs = require("fs");

fs.readFile("main.js", function(err, buffer) {
  if(err) {
    return;
  }

  console.log(buffer.toString());
});
```

Performance

4



REPL

33

- ❑ Execute “node” and then enter
- ❑ Interactive mode
- ❑ Write and evaluate JavaScript code

```
> node  
> 8 + 5  
13  
>
```

Debugging

34

- ❑ `node --inspect --inspect-brk main.js`
- ❑ Open Chrome at `chrome::/inspect`
- ❑ Wait for remote target list to refresh
- ❑ Click inspect
- ❑ Use Console/Sources/Memory tabs

Attach Debugger

35

- Find the PID of the running process
- Send **SIGUSR1** signal

```
kill -s SIGUSR1 nodejs-pid
```

- Windows does not support the SIGUSR1 signal
 - ▣ Can use **process._debugProcess(pid)** instead
- Once the relevant process is signaled can use Chrome as usual

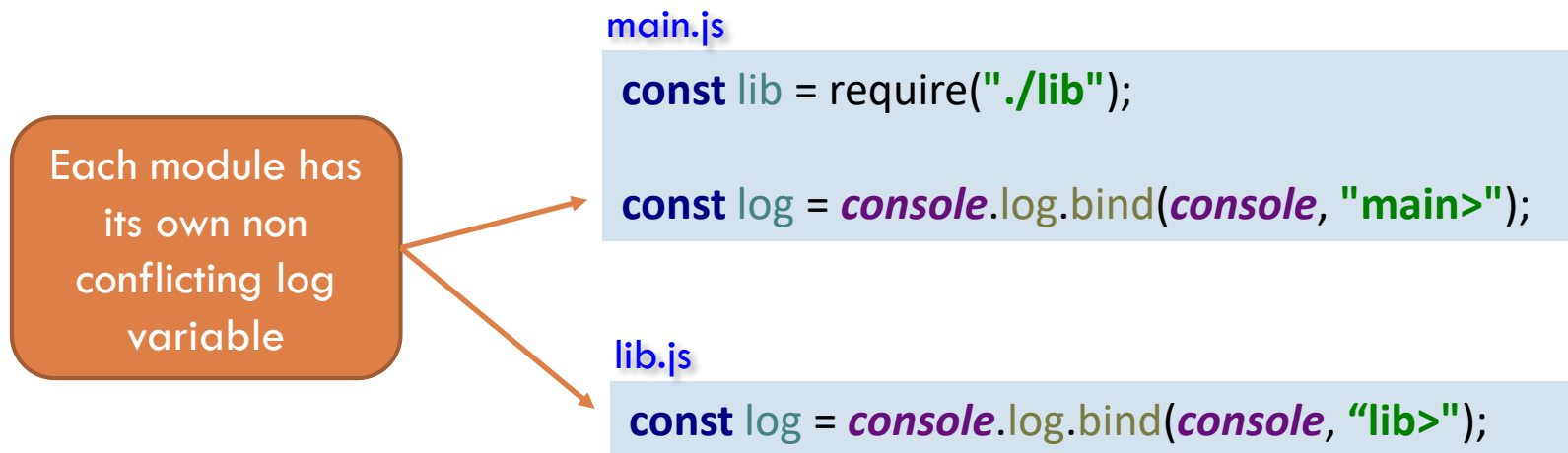
MODULES



Module System

37

- Each file is treated as a separate module
- Variables local to the module are private and cannot be accessed by other modules

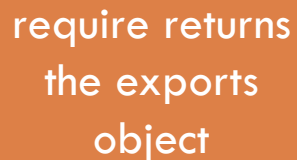


Exporting

38

- Exporting a variable/function is done through the **exports** object

require returns
the exports
object



```
const log = console.log.bind(console, "lib>");  
  
function doSomething() {  
  log("doSomething");  
}  
  
exports.doSomething = doSomething;
```

```
const {doSomething} = require("./lib");
```

Module Wrapper

39

- Each file has its own **exports**, **module** and other variables
- Node.js achieves that by wrapping your code inside a function

```
(function(exports, require, module, __filename, __dirname) {  
  // Module code actually lives in here  
});
```

Module Scope

40

- **__dirname** – Full path of the directory containing the current module
- **__filename** – Full path of the current module
- **exports** – We saw that already
- **module** – Reference to the module object
- **require** – We saw that already

module.exports

41

- The exports object is created by the module system
- Sometimes you want to control the exports instance
 - ▣ For example, exporting a class

```
class Logger {  
  constructor(prefix) {  
    this.log = console.log.bind(console, prefix + ">");  
    this.warn = console.warn.bind(console, prefix + ">");  
    this.error = console.error.bind(console, prefix + ">");  
  }  
}
```

```
module.exports = Logger;
```

```
const {Logger} = require("./logger");
```

```
const logger = new Logger("main");
```

```
logger.log("In the beginning");
```

Cyclic Dependencies

42

- In case of cyclic dependency Node.js returns the original **module.exports** object
 - ▣ Thus, think twice before overwriting it

lib1.js

```
const lib2 = require("./lib2");

function run() {
  console.log("lib2", lib2);
}

module.exports = {
  run
};
```

lib2.js

```
const lib1 = require("./lib1");

function run() {
  console.log("lib1", lib1);
}

module.exports = {
  run
};
```



Might be
empty
object

Resolving require

43

- When using a relative path Node.js follows exactly that path

```
require("./lib");
```

- Same for absolute path

```
require("/lib");
```

- Supported extensions are: .js, .json, .node

Resolving require

44

- When specifying non relative/none absolute path
Node.js looks for the following locations
 - ▣ Core module: http, fs, path
 - ▣ ./node_modules/lib1.js
 - ▣ ../node_modules/lib1.js
 - ▣ ../../node_modules/lib1.js
- Up until the root folder

```
require("lib1");
```

Require a folder

45

- It is valid to require a folder. In that case Node.js looks for **index.js** file
- You can change the default name by using a **package.json** file

Node.js looks
for main.js
instead of
index.js

```
{  
  "main": "main.js"  
}
```

module.paths

46

- Allows for dynamic modification of the locations Node.js uses when resolving a dependency

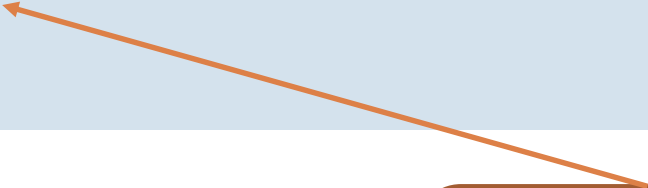
```
module.paths.push("c:\\1");  
require("lib");
```

require.cache

47

- Holds a map of all loaded modules
- Can manipulate it and force module reload

```
const path = require("path");  
  
require("./lib");  
  
delete require.cache[path.resolve(__dirname, "./lib.js")];  
  
require("./lib");
```



Delete reference to
module object from
require.cache

Native Modules

48

- When Node.js public API is not enough you may implement native modules which access OS directly
- Not straight forward 😞
- Need to write cross platform C++ code
 - ▣ May use libuv to achieve that
- Must use V8 APIs to interact with JavaScript code
 - ▣ V8 changes a lot over time
 - ▣ Thus, native module tend to break cross Node.js versions

C++ Addon

49

```
#include <node.h>

namespace demo {

using v8::FunctionCallbackInfo;
using v8::Isolate;
using v8::Local;
using v8::Object;
using v8::String;
using v8::Value;

void Method(const FunctionCallbackInfo<Value>& args) {
    Isolate* isolate = args.GetIsolate();
    args.GetReturnValue().Set(String::NewFromUtf8(isolate, "world"));
}

void init(Local<Object> exports) {
    NODE_SET_METHOD(exports, "hello", Method);
}

NODE_MODULE(NODE_GYP_MODULE_NAME, init)

} // namespace demo
```

Compile the Addon

50

- Create **binding.gyp** file

```
{
  "targets": [
    {
      "target_name": "addon",
      "sources": [ "hello.cc" ]
    }
  ]
}
```

- npm install -g node-gyp
- node-gyp configure → Makefile/vcxproj file is created
- node-gyp build → addon.node is created

```
const addon = require('./build/Release/addon');
console.log(addon.hello());
```

CONTROL FLOW



The Challenge

52

- Node.js asynchronous nature impose non intuitive programming model

Callback hell



```
function readFileSync(filePath, callback) {  
  fs.stat(filePath, function (err, stat) {  
    if (err) {  
      callback(err);  
      return;  
    }  
  
    if (stat.isFile()) {  
      fs.readFile(filePath, function (err, data) {  
        if (err) {  
          callback(err);  
          return;  
        }  
  
        callback(null, data.toString());  
      });  
    }  
  });  
}
```

async package

53

- Async utilities for node and the browser
- **npm install async**

```
function readFileSyncExists(filePath, cb) {  
  async.seq(  
    fs.stat,  
    (stat, cb) => stat.isFile() ? fs.readFile(filePath, cb) : cb(new Error("Not a file")),  
  )(filePath, cb);  
}
```

Promise Flow

54

- Convert each function to promise based

```
function readFile(filePath) {  
  return new Promise((resolve, reject)=> {  
    fs.readFile(filePath, function(err, data) {  
      if(err) {  
        reject(err);  
        return;  
      }  
  
      resolve(data);  
    });  
  });  
}
```

- Can wrap that logic inside a **promisify** helper

promisify

55

es6-promisify
package offers
an almost
identical
function

```
function promisify(func) {  
  return function (...args) {  
    return new Promise((resolve, reject) => {  
      args.push(callback);  
      func.apply(this, args);  
  
      function callback(err, res) {  
        if(err) {  
          reject(err);  
          return;  
        }  
  
        resolve(res);  
      }  
    });  
  }  
}
```

Use the new functions

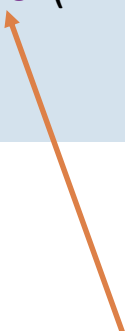
56

```
function readFileIfExists(filePath) {  
  return stat("1.txt").then(stat => {  
    if (stat.isFile()) {  
      return readFile(filePath);  
    }  
  
    throw new Error("Not a file");  
  });  
}
```

Return a
promise to
allow
“continuation”



Must throw
exception to
signal an error



Callback hell ?

57

- The promise flow simplifies code since middle layers does not have to deal with errors
- However, the code still suffers from the callback hell

```
function readFileIfExists(filePath) {  
  return stat("1.txt").then(stat => {  
    if (stat.isFile()) {  
      return readFile(filePath);  
    }  
  
    throw new Error("Not a file");  
  });  
}
```

async/await

58

- Code feels almost synchronous

```
async function readFileIfExists(filePath) {  
  const info = await stat(filePath);  
  if(!info.isFile()) {  
    throw new Error("Not a file");  
  }  
  
  return await readFile(filePath);  
}
```

```
async function main() {  
  try {  
    const data = await readFileIfExists("1.txt");  
    console.log(data.toString());  
  }  
  catch(err) {  
    console.error(err);  
  }  
}
```

Promise Flow

59

- ❑ Unfortunately most Node.js APIs are callback based
- ❑ Need to manually wrap the code
- ❑ Be careful when wrapping instance methods
 - ❑ Must keep the correct **this**

```
const obj = {  
  id: 123,  
  oldStyle: function(callback) {  
    callback(null, this.id);  
  }  
};  
  
const newStyle = promisify(obj.oldStyle.bind(obj));
```

Promise Limitation

60

- ❑ Promise can be resolved only once
- ❑ Therefore, it cannot represent a reoccurring event
 - ❑ Stream
 - ❑ Button clicks
- ❑ Runs immediately

Rxjs

61

- ❑ ReactiveX library for JavaScript
- ❑ A big concept
- ❑ Some love it, some hate it
- ❑ Out of scope for us
- ❑ However, lets take a simple look

Rxjs

62

- Observable generates a stream of values

```
const filePath = "1.txt";
const source = stat(filePath).switchMap(stat => {
  if (!stat.isFile()) {
    throw new Error("Not a file");
  }

  return readFile(filePath);
});

source.subscribe(res => {
  console.log(res.toString());
}, err => {
  console.error(err);
});
```

GENERAL TOPICS



Auto Restart

64

- When developing a web server it is convenient that the server is automatically restarted with each code modification
- `npm install nodemon`
- `node_modules/.bin/nodemon main.js`
- Other alternatives
 - ▣ `forever`
 - ▣ `pm2`

Yarn

65

- ❑ <https://yarnpkg.com/en/>
- ❑ Yarn follows the same NPM rules but is considered faster
- ❑ Has better caching strategic
- ❑ Automatically creates package.json
- ❑ Supports workspace

Summary

66

- Node.js is a lean platform
 - ▣ Less Than 20MB of installation
- Easily installed and getting started
- Lot's of open source packages
 - ▣ Some time its hard to choose the right one