UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

WEB TECHNOLOGIES — LECTURE 10

JAVASCRIPT IN A SERVER ENVIRONMENT: NODE.JS

Luigi Libero Lucio Starace, PhD

luigiliberolucio.starace@unina.it

https://docenti.unina.it/luigiliberolucio.starace

https://luistar.github.io

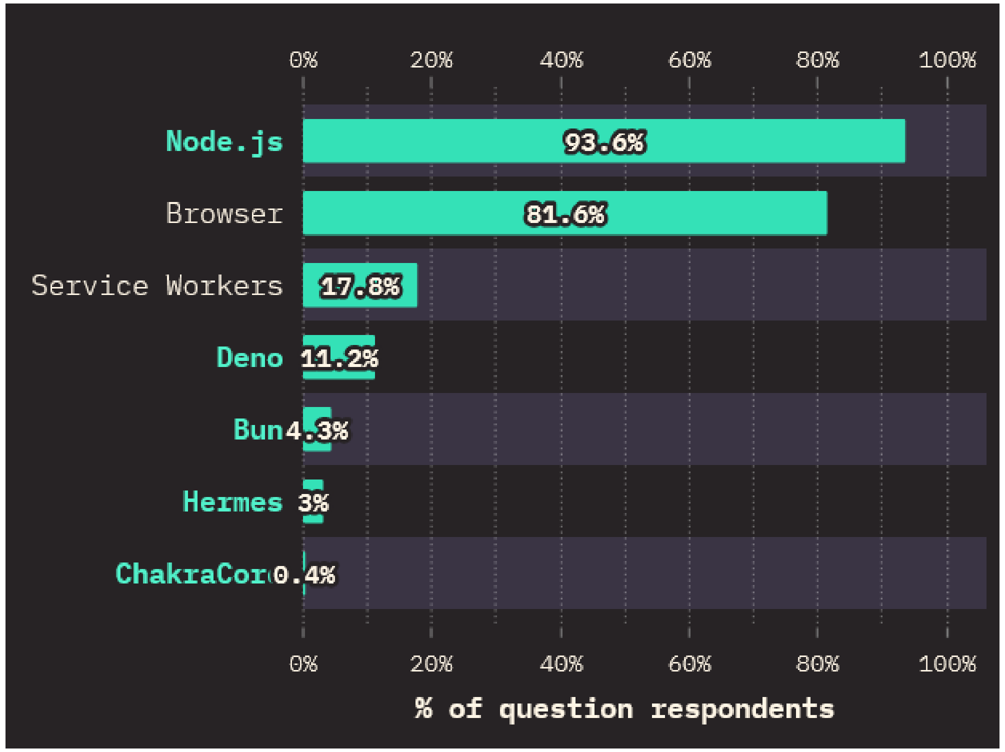
PREVIOUSLY, ON WEB TECHNOLOGIES

- We now know **server-side scripting**
  - Web pages can be generated by programs on-the-fly
- We've learned a good deal about JavaScript in the first lectures
  - So far, we've only used it in a browser environment
  - Well, we can also use it in a **server environment**!
  - Several different runtimes allow us to do so…

Node.js Deno Bun

NODE.JS

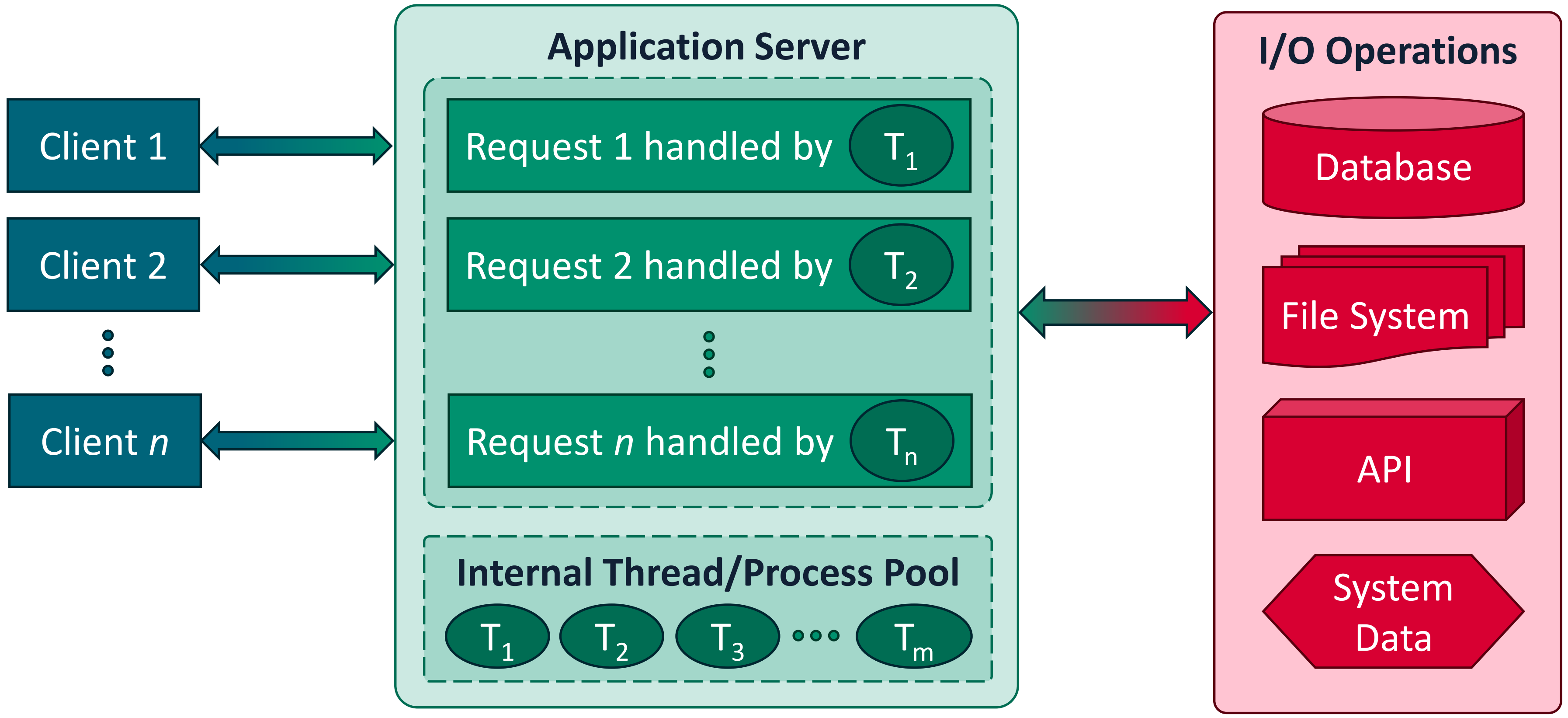
- **Open-source** and **cross-platform** JavaScript Runtime Environment
- Runs Google's **V8 JavaScript engine** from the Chromium project
- **Event-based**, **asynchronous-by-default**, **non-blocking I/O** model
- By far the most popular JS execution environment



State of JavaScript 2022 Report

MULTI–THREADED REQUEST HANDLING

In other execution environments/languages **one request** is handled by **one dedicated thread/process**



Client 1
Client 2
Client *n*
Application Server
Request 1 handled by T_1
Request 2 handled by T_2
Request *n* handled by T_n
Internal Thread/Process Pool
T_1 T_2 T_3 … T_m
I/O Operations
Database
File System
API
System Data

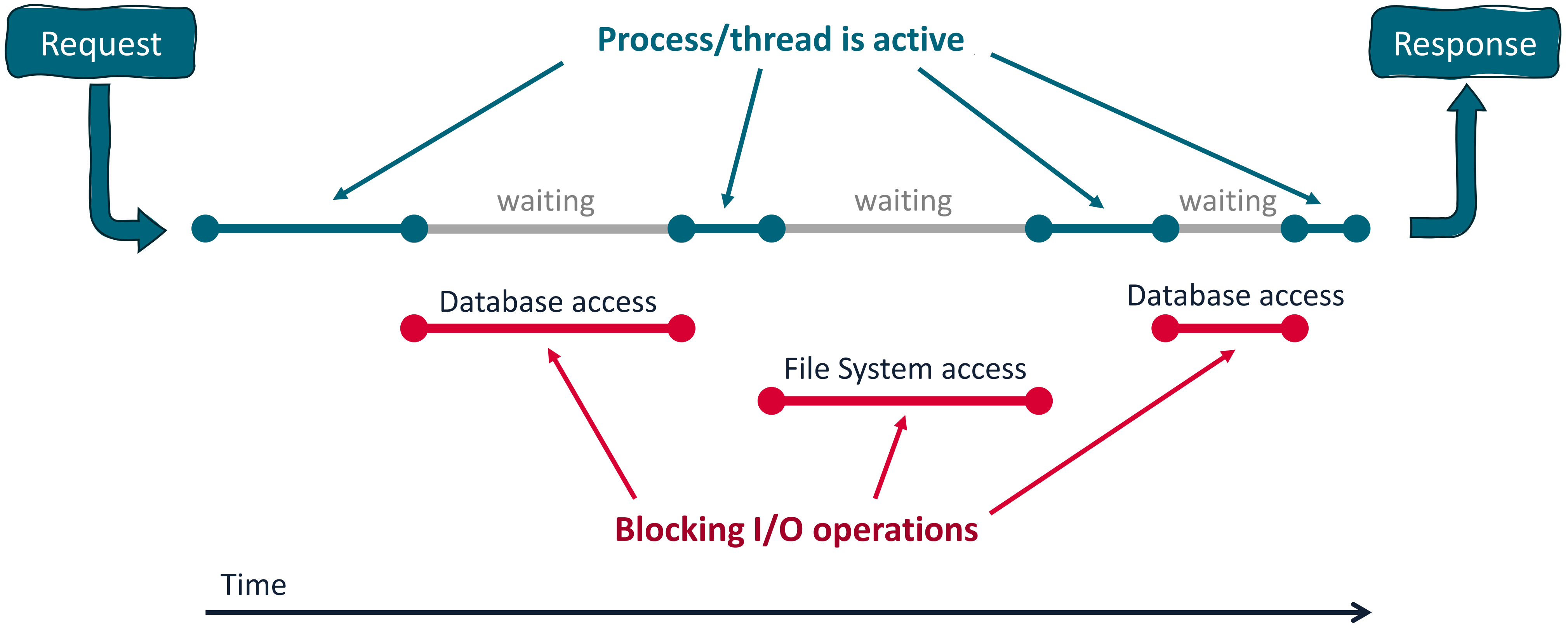
EXPENSIVE, BLOCKING I/O OPERATIONS

When serving Requests, Web Applications often need to perform time-consuming, blocking I/O operations

- Access a database
- Access an external API (e.g.: using `fetch()`)
- Read a file from the local file system
- Read the request body

As a result, the threads/processes handling a request spend a lot of time **waiting** for these operations to complete

BLOCKING I/O EXAMPLE



MULTI–THREADED REQUEST HANDLING

- Can be **inefficient** (lots of thread time spent waiting)
- **Limited scalability** (number of concurrent requests handled)
  - Thread pool is pre-allocated at the pre-defined size. Cannot manage more concurrent request than the thread pool allows.

Let’s think of it with an example.

- A Restaurant has **10 tables**. Management decides to hire **10 waiters**. Each waiter is assigned to exactly one table.

MULTI–THREADED REQUEST HANDLING

- When a customer arrives, they are seated to the first free table and given a menu by their waiter.
- Customers take 5 to 10 minutes to decide what they want. **During this time, the waiter remains idle**.
- When a customer decided what they want, the waiter takes their order and bring it to the chef.
- The chef takes 15 to 30 minutes to prepare the food. **During this time, the waiter remains idle**.

MULTI–THREADED REQUEST HANDLING

- When the food is ready, the waiter brings it to the customer. The customer takes 15 to 30 minutes to eat the food. **During this time, the waiter remains idle**.
- Once the customer is done eating, the waiter cleans their table and asks a manager to prepare their bill. Bill preparation takes 2 to 5 minutes. **During this time, the waiter remains idle**.
- Once the bill is ready, the waiter takes it to the customer, who pays and leaves the establishment.
- A new customer can now be served by the same waiter at the same table.

MULTI–THREADED REQUEST HANDLING

- In our example, waiters are threads serving customers (requests).
- At most 10 customers (requests) can be served at any time.
- A customer deciding what to order, a cook preparing the food, a customer eating it, and a manager preparing the bill are **blocking operations**. The waiter remains idle during these operations.
- Resources are not being used in a very efficient way…
- What if we want to serve an additional request at a time? We need to hire a new waiter.

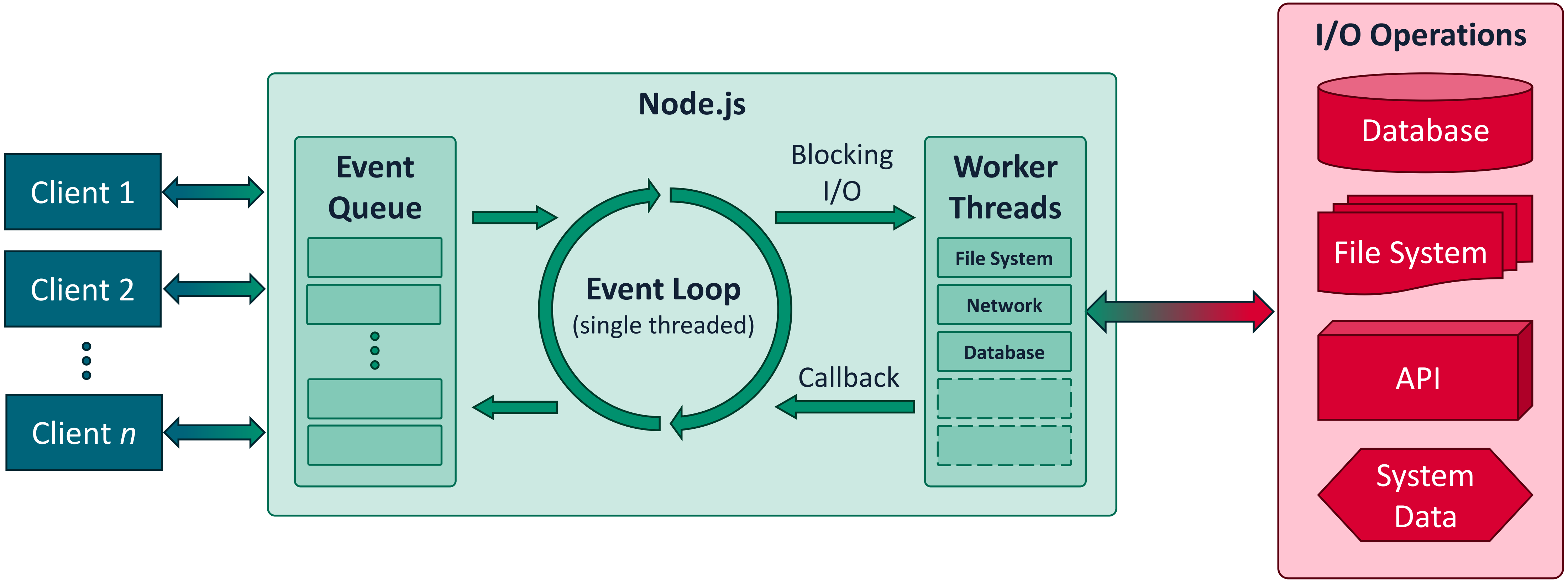
IMPROVING EFFICIENCY AT OUR RESTAURANT

- Restaurant hires only one waiter to attend its 10 tables
- When a customer arrives, the waiter greets them and sits them to their table, giving them a menu. While the customer is choosing, the waiter is available to attend the needs of other customers.
- Once the customer decides their order, the waiter brings it to the cook. While the cook prepares the food, the waiter returns back to the dining hall and is available to attend to other customers.
- Once the food is ready (and the kitchen bell rings), the waiter bring the food to the customer. While the customer eats, the waiter is available to attend to other customers.

NODE.JS: PHILOSOPHY

- Node.js is based on a philosophy that is quite similar to our more efficient restaurant example
- A Node.js program runs in a single process
- No need to create new threads for every request
- The Node.js runtime provides a set of **asynchronous I/O primitives** in its standard library that prevent JavaScript code from blocking
  - Most libraries leverage non-blocking paradigm whenever possible
  - Wide adoption of the **callback** pattern
  - Generally, synchronous operations are an exception rather than the norm

THE SINGLE–THREADED EVENT LOOP



INSTALLING NODE.JS

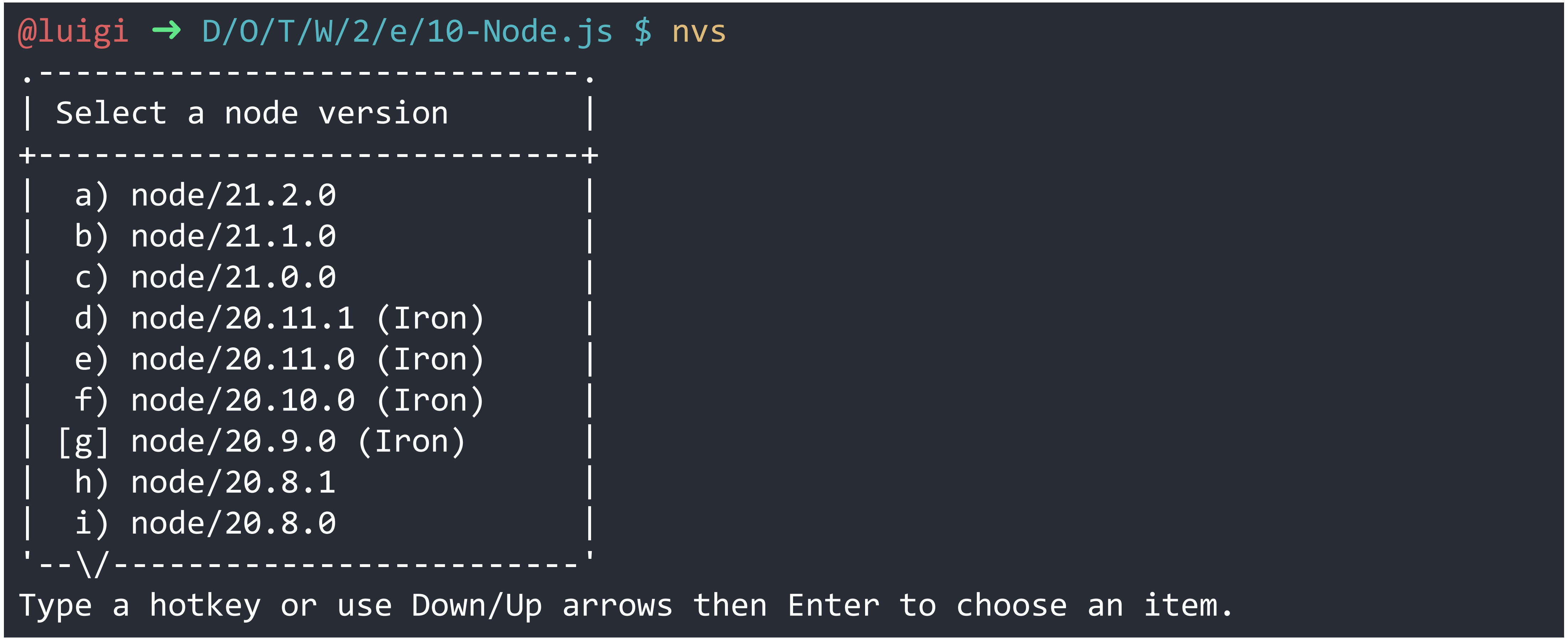
To install Node.js you can:

- use the installers available on the official website
- use your favourite package manager
- use a Node.js version manager such as nvm, nvs, nvm4w
  - Nice if you want to experiment with different versions and switch frequently

In the Web Technologies course (2023/24), we'll use **Node.js 20.9.0**

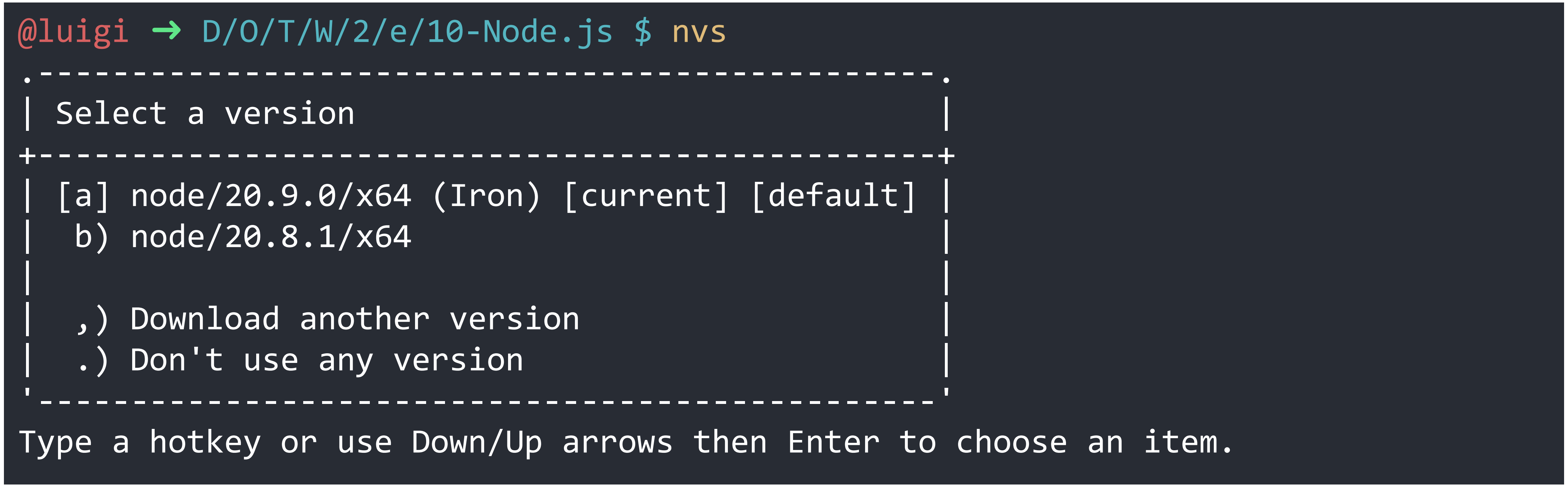
- As long as you use a supported version, there should be no issues

INSTALLING NODE.JS



```
@luigi ➜ D/O/T/W/2/e/10-Node.js $ nvs
.-----------------------------.
| Select a node version       |
+-----------------------------+
|  a) node/21.2.0             |
|  b) node/21.1.0             |
|  c) node/21.0.0             |
|  d) node/20.11.1 (Iron)     |
|  e) node/20.11.0 (Iron)     |
|  f) node/20.10.0 (Iron)     |
| [g] node/20.9.0 (Iron)      |
|  h) node/20.8.1             |
|  i) node/20.8.0             |
'--\/-------------------------'
Type a hotkey or use Down/Up arrows then Enter to choose an item.
```

INSTALLING NODE.JS



```
@luigi ➜ D/O/T/W/2/e/10-Node.js $ nvs
.--------------------------------------------------.
| Select a version                                 |
+--------------------------------------------------+
| [a] node/20.9.0/x64 (Iron) [current] [default] |
|  b) node/20.8.1/x64                              |
|                                                  |
|  ,) Download another version                     |
|  .) Don't use any version                        |
'--------------------------------------------------'
Type a hotkey or use Down/Up arrows then Enter to choose an item.
```

INSTALLING NODE.JS

- To check that everything is ok, you can run: **`node --version`**

```
@luigi → D/O/T/W/2/e/10-Node.js $ node --version
v20.9.0

@luigi → D/O/T/W/2/e/10-Node.js $
```

NODE.JS: HELLO WORLD

```
//hello-world.js
"use strict"
console.log("Hello Node.js!");
```

```
@luigi → D/O/T/W/2/e/10-Node.js $ node .\hello-world.js
Hello Node.js!

@luigi → D/O/T/W/2/e/10-Node.js $
```

NODE.JS VS BROWSER ENVIRONMENT

- In a browser environment, we mainly use JavaScript to manipulate the DOM. We had **window**, **document**, Web Platform APIs such as **Cookies**, **LocalStorage**, etc…
- In Node.js, these objects and APIs are not available
- On the other hand, in Node.js, we can leverage the standard library and its many modules for **file system** access and **network** access
  - You can read and write files on the filesystem
  - You can listen on a socket for incoming requests

NPM: THE NODE PACKAGE MANAGER

- One of the main drivers of the success of Node.js is **npm**
- In September 2022, **2.1+ million** packages were available
- The biggest single-language repository on Earth
- There is a package for (almost!) everything
- **npm** provides ways to **download** and **manage** dependecies for Node.js projects

NPM: GETTING STARTED

- **npm** should be installed by default with Node.js
- You can check that npm is available using: **npm --version**

```
@luigi → D/O/T/W/2/e/10-Node.js $ npm --version
10.2.2

@luigi → D/O/T/W/2/e/10-Node.js $
```

- To create an npm package in the current directory, run: **`npm init`**
- To create an ES compatible module, use **`npm init es6`**

NPM: CREATING A NEW PACKAGE

- **`npm init`** will ask for some information about your package/project
  - Package name, version, repository (if any), licence, author,…
  - Entry point (the first javascript file to run), test command (if any)
- At the end, npm creates a **project.json** file in the same directory
- The project.json file contains all the information about the package, including its **dependencies** (if any) and **commands** to run it.
- Think of it as npm’s version of Maven’s **`pom.xml`**
- Since we will work with ES modules, use **`npm init es6`** when creating a new package

THE PACKAGE.JSON FILE

```
{
  "name": "hello-web-technologies",
  "version": "0.0.1",
  "description": "Web Technologies' first npm package",
  "main": "app.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "greetings"
  ],
  "author": "Luigi Libero Lucio Starace",
  "license": "MIT"
}
```

PACKAGE.JSON: MAIN FILE AND TASKS

- The main file (**`app.js`** in our example) is executed when client code imports our package. It generally exports public variables/functions
- The **`scripts`** property can be used to specify command-line tasks
  - The **`test`** task was defined by default and does not do much
  - Tasks can be executed by running: **`npm <taskname>`**
  - We can add a **`start`** task by modifying the scripts property as follows:

```
"scripts": {
  "start": "node app.js",
  "test": "echo \"Error: no test specified\" && exit 1"
},
```

NPM: RUNNING TASKS

```
//from package.json
"scripts": {
  "start": "node app.js",
  "test": "echo \"Error: no test specified\" && exit 1"
},
```

```
//app.js file
"use strict";
console.log("Hello Web Technologies!");
```

```
@luigi → D/O/T/W/2/e/10-Node.js $ npm start

> hello-web-technologies@0.0.1 start
> node app.js

Hello Web Technologies!

@luigi → D/O/T/W/2/e/10-Node.js $
```

NPM: INSTALLING DEPENDENCIES

- Let’s make our package more interesting
- We search the npm library, and find this interesting package: cowsay
- Let’s use this package
- To install a dependency, run **npm install <packagename>**

NPM: INSTALLING DEPENDENCIES

- In our case, **npm install cowsay**
- Two things happen:
  1. npm keeps track of the new dependency, by adding a new section in the **package.json** file

```
"dependencies": {
  "cowsay": "^1.5.0"
}
```

  2. Dependencies are downloaded in the **node_modules** directory
     - Notice that npm downloaded not only the package we requested, but also its (recursive) dependencies!

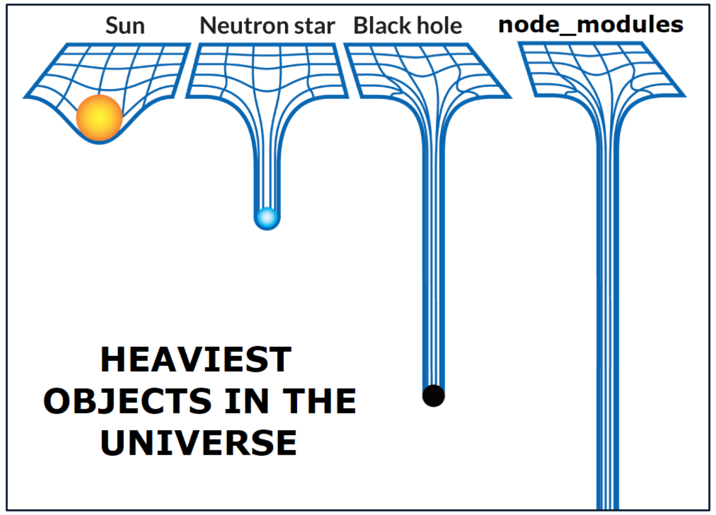
NPM: IMPORTING DEPENDENCIES

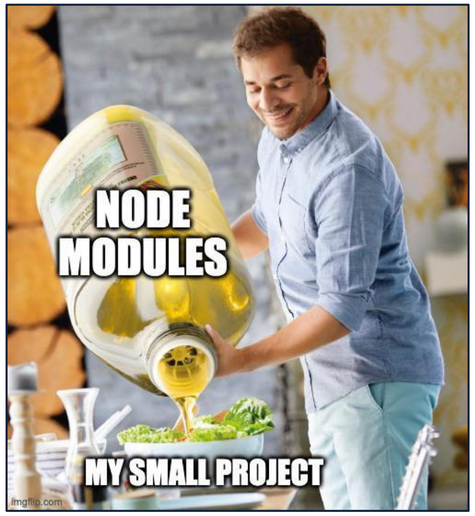
```
//app.js file
import cowsay from 'cowsay';

console.log(cowsay.think({
  text: "Hello Web Tech!"
}));
```

```
@luigi → D/O/T/W/2/e/10-Node.js $ npm start

> hello-web-technologies@0.0.1 start
> node app.js

 _________________
( Hello Web Tech! )
 -----------------
        o   ^__^
         o  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
@luigi → D/O/T/W/2/e/10-Node.js $
```

DISTRIBUTING AN NPM PACKAGE

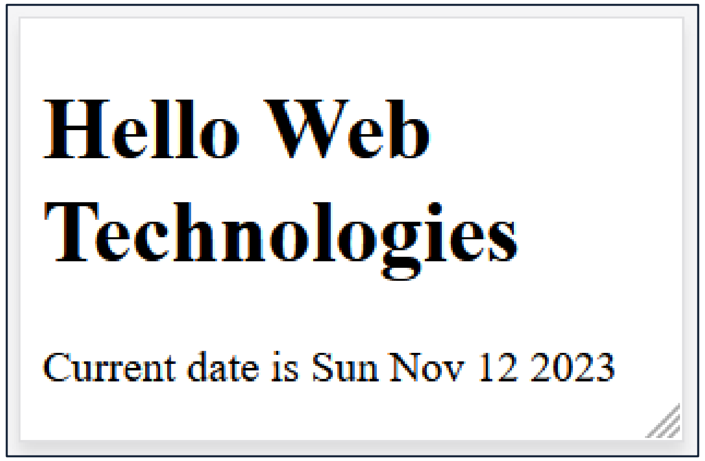
- When distributing an npm package, we just need to provide our own code, and a **package.json** descriptor.
  - No need to add **/node_modules/** to versioning!
- The command **npm install** can be used to install all the dependencies listed in the **package.json** file





A FIRST WEB APPLICATION WITH NODE.JS

```
import http from 'http';

const PORT = 3000;

let server = http.createServer(function(request, response){
  let course = "Web Technologies";
  response.writeHead(200, {"Content-Type": "text/html"});
  response.write(`<!DOCTYPE html>
  <html><body>
    <h1>Hello ${course}</h1>
    <p>Current date is ${new Date().toDateString()}</p>
  </body></html>`);
  response.end();
}).listen(PORT);

console.log(`Web app listening on port ${PORT}`);
```



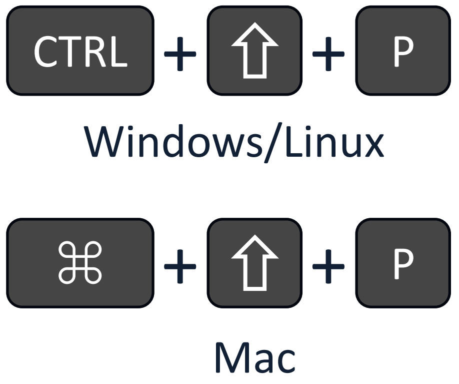
Hello Web Technologies

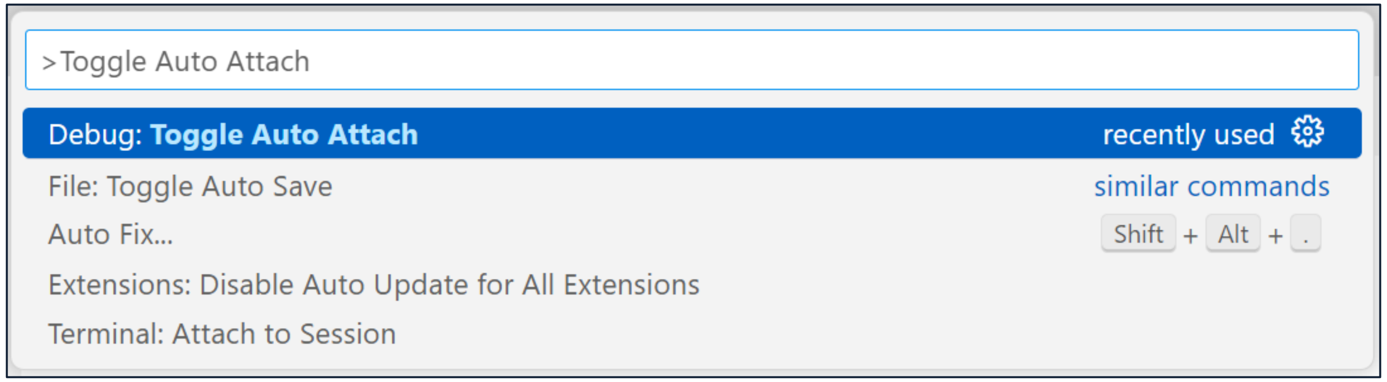
Current date is Sun Nov 12 2023

DEBUGGING A NODE.JS APP IN VSCODE

- The easiest way is to enable the **Auto Attach** feature
- This will attach a debugger to Node.js processes started in VSCode
- To enable Auto Attach, use CTRL+SHIFT+P to show VSCode command palette and then search for «**Toggle Auto Attach**»
- I suggest you then select the «**Smart**» mode, which automatically attaches only to scripts that are **outside /node_modules/**

DEBUGGING A NODE.JS APP IN VSCODE




>Toggle Auto Attach

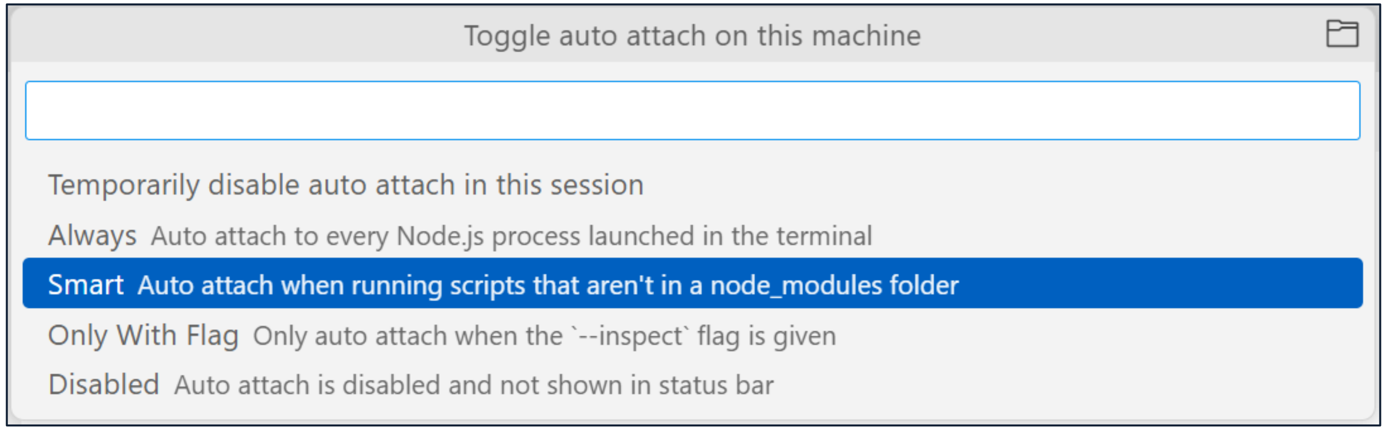
Debug: **Toggle Auto Attach** recently used

File: Toggle Auto Save similar commands

Auto Fix... Shift + Alt + .

Extensions: Disable Auto Update for All Extensions

Terminal: Attach to Session


Toggle auto attach on this machine

Temporarily disable auto attach in this session

Always Auto attach to every Node.js process launched in the terminal

Smart Auto attach when running scripts that aren't in a node_modules folder

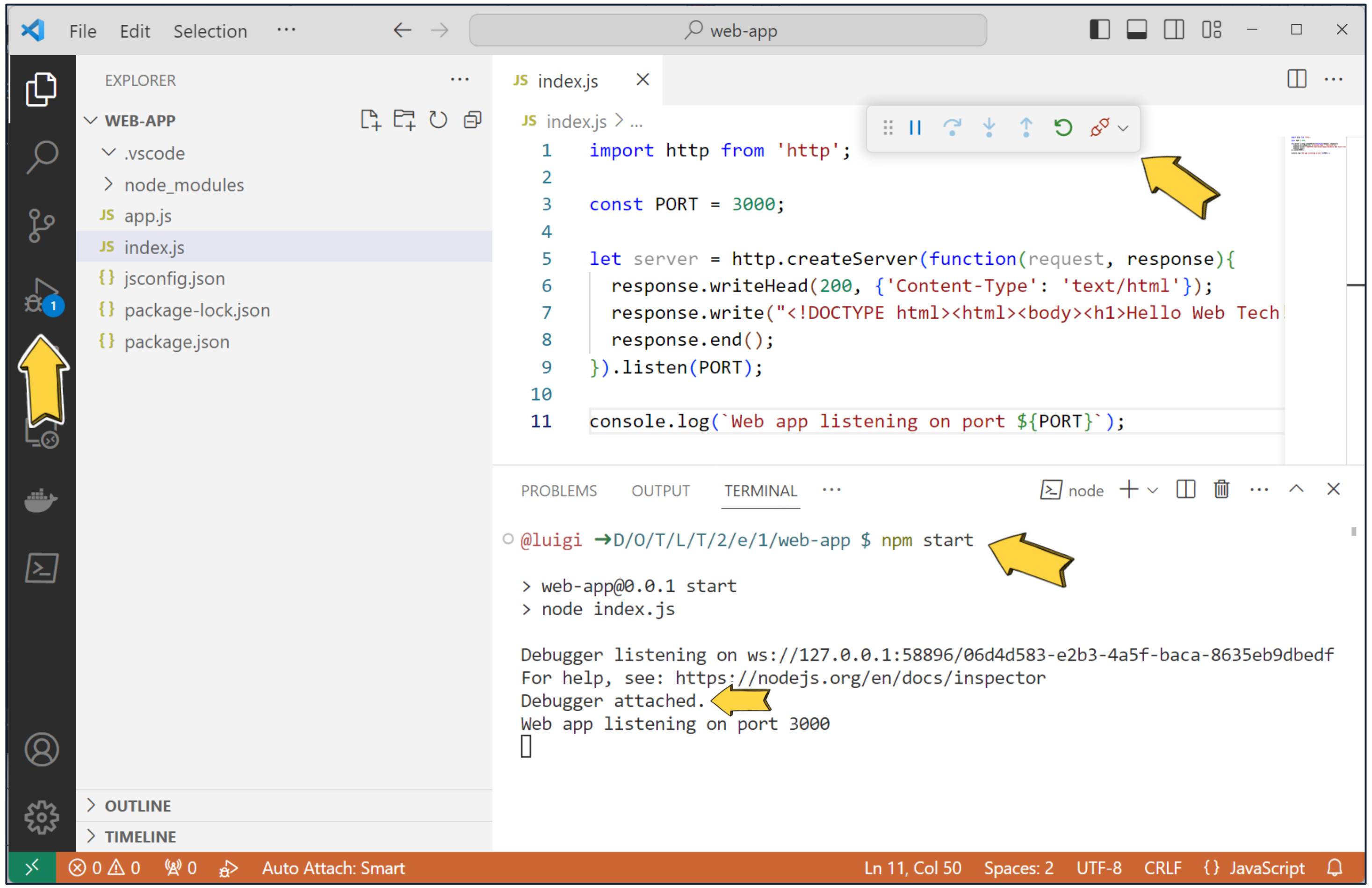
Only With Flag Only auto attach when the `--inspect` flag is given

Disabled Auto attach is disabled and not shown in status bar

DEBUGGING A NODE.JS APP IN VSCODE

When you start your Node.js app from the VSCode terminal, a debugger will attach

The Run and Debug panel will activate, and you can start properly debugging your app

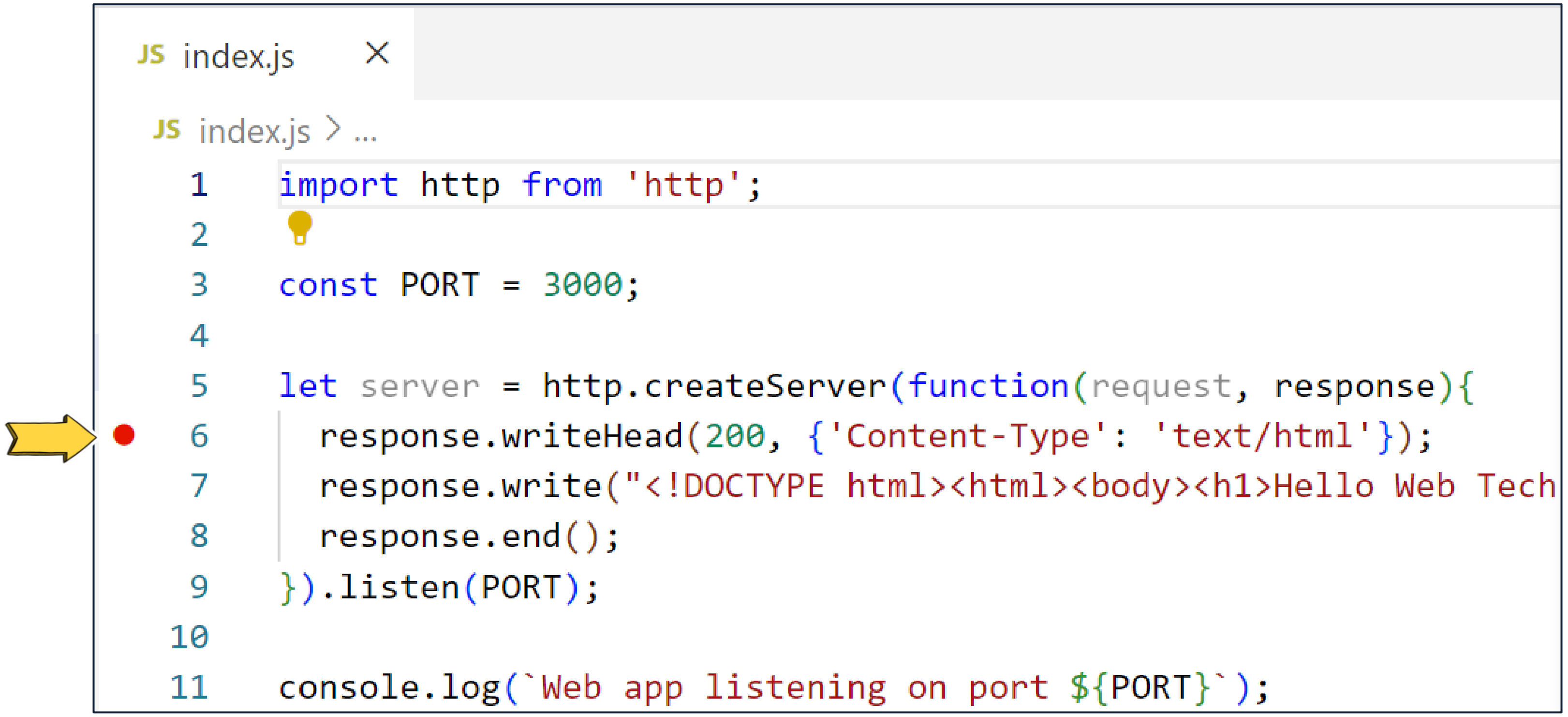


```javascript
import http from 'http';

const PORT = 3000;

let server = http.createServer(function(request, response){
  response.writeHead(200, {'Content-Type': 'text/html'});
  response.write("<!DOCTYPE html><html><body><h1>Hello Web Tech
  response.end();
}).listen(PORT);

console.log(`Web app listening on port ${PORT}`);
```

```
@luigi ➜D/O/T/L/T/2/e/1/web-app $ npm start

> web-app@0.0.1 start
> node index.js

Debugger listening on ws://127.0.0.1:58896/06d4d583-e2b3-4a5f-baca-8635eb9dbedf
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
Web app listening on port 3000
```

DEBUGGING A NODE.JS APP IN VSCODE

- You can add breakpoints as usual, by clicking next to the line number of the line of code on which you want to place the breakpoint

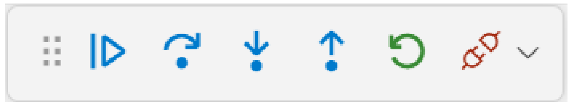


```
import http from 'http';

const PORT = 3000;

let server = http.createServer(function(request, response){
  response.writeHead(200, {'Content-Type': 'text/html'});
  response.write("<!DOCTYPE html><html><body><h1>Hello Web Tech
  response.end();
}).listen(PORT);

console.log(`Web app listening on port ${PORT}`);
```

DEBUGGING A NODE.JS APP IN VSCODE

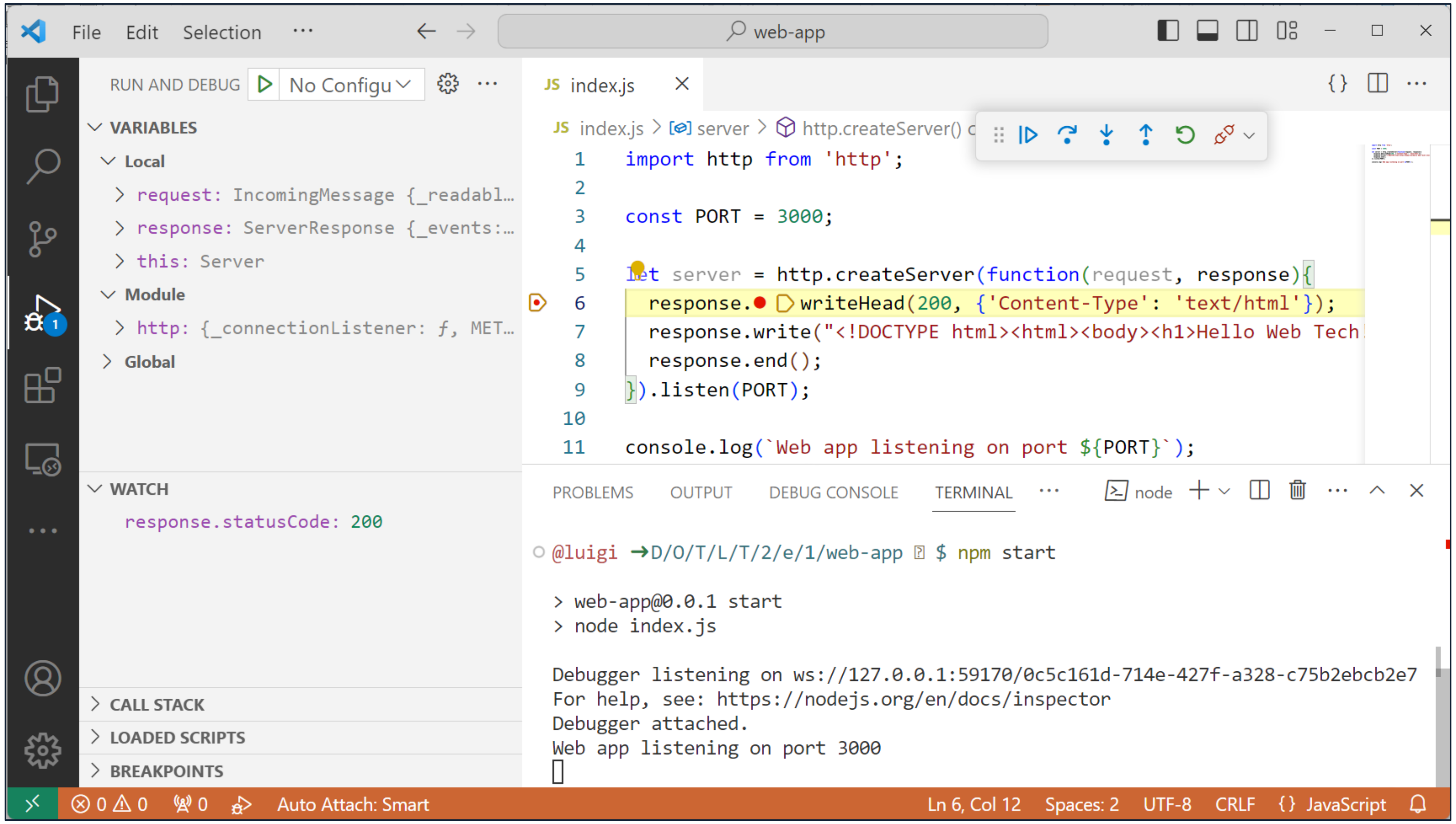
- Code execution **pauses** when it reaches a breakpoint
- The Run and Debug panel allows us to inspect variables, to watch expressions, to look at the call stack and much more
- Definitely more effective than console.logging our way out of the bug!
- We can also exercise fine-grained control the execution flow with **Debugging Actions**

DEBUGGING: ACTIONS



	Action	Description
▷	**Resume**	Resume the normal execution flow (until the next breakpoint)
↷	**Step Over**	Execute the next statement as a single unit, without inspecting its inner component steps
↓	**Step Into**	Enter the next statement to follow its execution line-by-line.
↑	**Step Out**	When inside a method or subroutine, return to the earlier execution context by completing remaining lines of the current method as though it were a single command.
↺	**Restart**	Terminate the current program execution and start debugging again using the current run configuration (does not work with auto attach!).
🔌	**Disconnect**	Terminate the current debugging session.

DEBUGGING A NODE.JS APP IN VSCODE



```
request: IncomingMessage {_readabl…
response: ServerResponse {_events:…
this: Server
http: {_connectionListener: ƒ, MET…
```

```
response.statusCode: 200
```

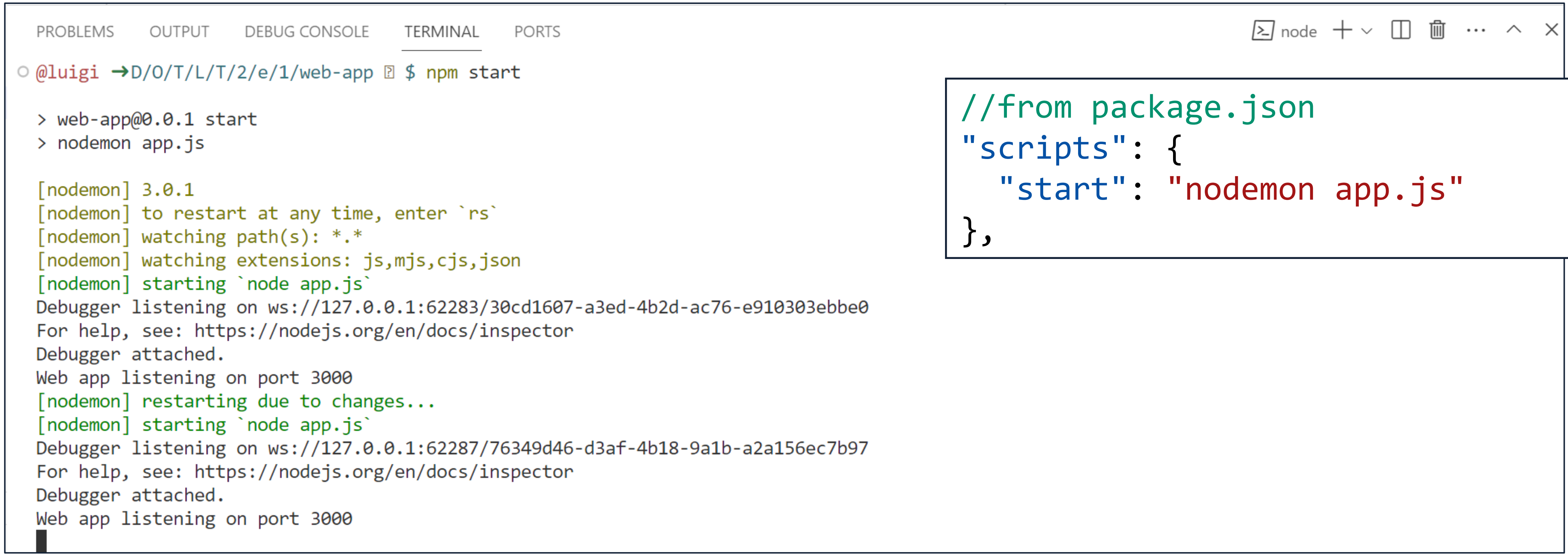
```javascript
import http from 'http';

const PORT = 3000;

let server = http.createServer(function(request, response){
  response.writeHead(200, {'Content-Type': 'text/html'});
  response.write("<!DOCTYPE html><html><body><h1>Hello Web Tech
  response.end();
}).listen(PORT);

console.log(`Web app listening on port ${PORT}`);
```

```
@luigi →D/O/T/L/T/2/e/1/web-app $ npm start

> web-app@0.0.1 start
> node index.js

Debugger listening on ws://127.0.0.1:59170/0c5c161d-714e-427f-a328-c75b2ebcb2e7
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
Web app listening on port 3000
```

LIVE RELOADING IN NODE.JS

- Everytime we make a change to our code, we need to restart the entire server (**npm start**, or **node app.js**, etc…)
- To make development more enjoyable, we can use utilities that monitor our codebase for changes, and restart the development server when needed (a.k.a. **live reloading**).
- One such utility is **nodemon**
- You can install it using npm (**npm install nodemon**)
- Then you just need to replace **node app.js** with **nodemon app.js**

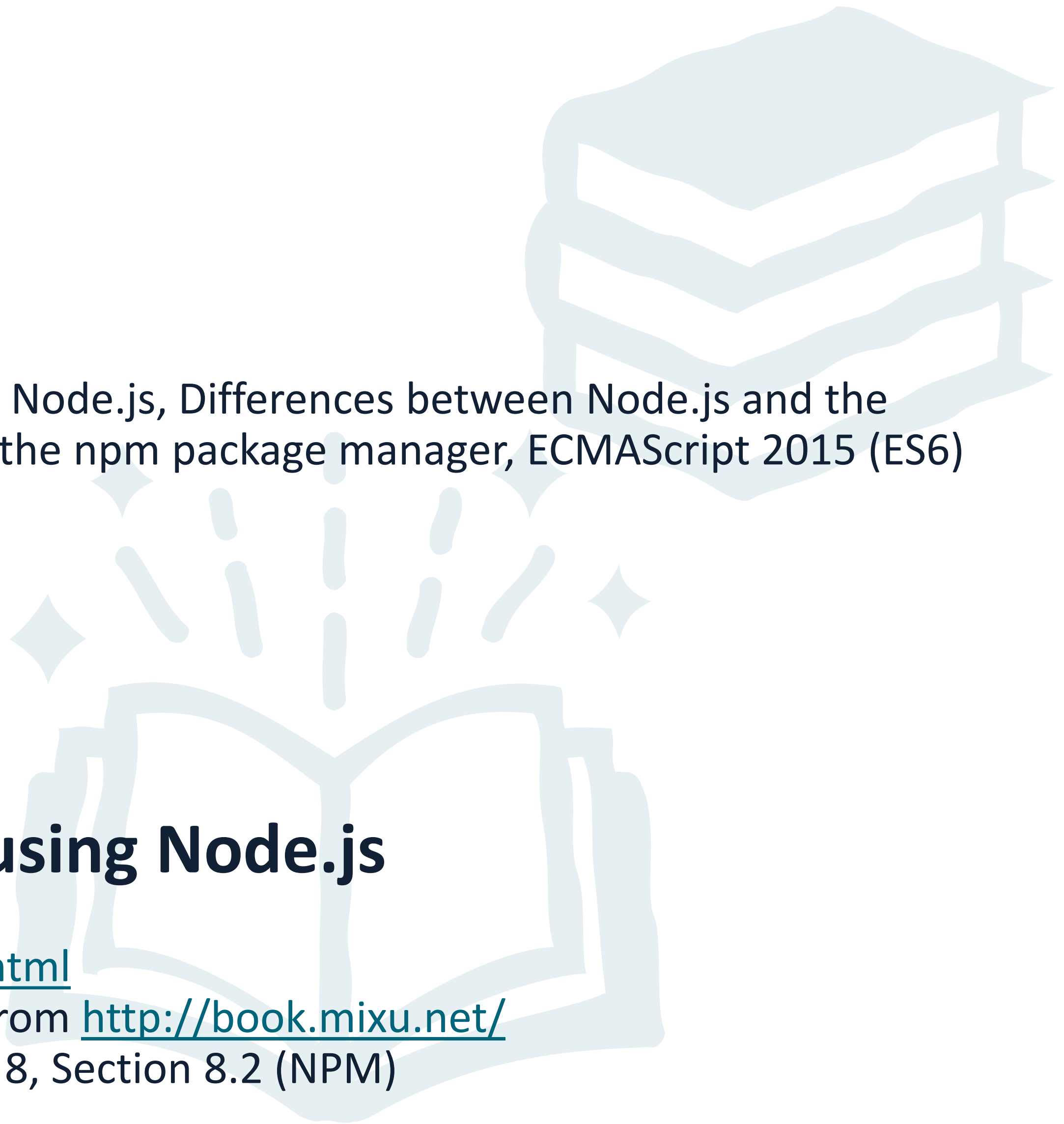
LIVE RELOAD WITH NODEMON (EXAMPLE)



```
○ @luigi →D/O/T/L/T/2/e/1/web-app ⍰ $ npm start

  > web-app@0.0.1 start
  > nodemon app.js

  [nodemon] 3.0.1
  [nodemon] to restart at any time, enter `rs`
  [nodemon] watching path(s): *.*
  [nodemon] watching extensions: js,mjs,cjs,json
  [nodemon] starting `node app.js`
  Debugger listening on ws://127.0.0.1:62283/30cd1607-a3ed-4b2d-ac76-e910303ebbe0
  For help, see: https://nodejs.org/en/docs/inspector
  Debugger attached.
  Web app listening on port 3000
  [nodemon] restarting due to changes...
  [nodemon] starting `node app.js`
  Debugger listening on ws://127.0.0.1:62287/76349d46-d3af-4b18-9a1b-a2a156ec7b97
  For help, see: https://nodejs.org/en/docs/inspector
  Debugger attached.
  Web app listening on port 3000
```

```
//from package.json
"scripts": {
  "start": "nodemon app.js"
},
```

REFERENCES (1/2)



- **Getting started**
  Node.js Docs
  https://nodejs.org/en/learn/
  Relevant parts: Introduction to Node.js, How to install Node.js, Differences between Node.js and the Browser, The V8 JavaScript Engine, An introduction to the npm package manager, ECMAScript 2015 (ES6) and beyond.
- **Eloquent JavaScript (3rd edition)**
  By Marijn Haverbeke
  Freely available at https://eloquentjavascript.net/
  Chapter 20
- **Mixu's Node book: A book about using Node.js**
  By Mikito Takada
  Freely available at http://book.mixu.net/node/single.html
  You can also download it in PDF, epub, mobi formats from http://book.mixu.net/
  Relevant parts: Chapter 2 (What is Node.js?), Chapter 8, Section 8.2 (NPM)

REFERENCES (2/2)

- **Nodemon**
  Nodemon official website
  https://nodemon.io/
- **Node.js debugging in VS Code**
  Visual Studio Code Docs
  https://code.visualstudio.com/docs/nodejs/nodejs-debugging