

# Node.js: JavaScript on the server

Claudia Hauff

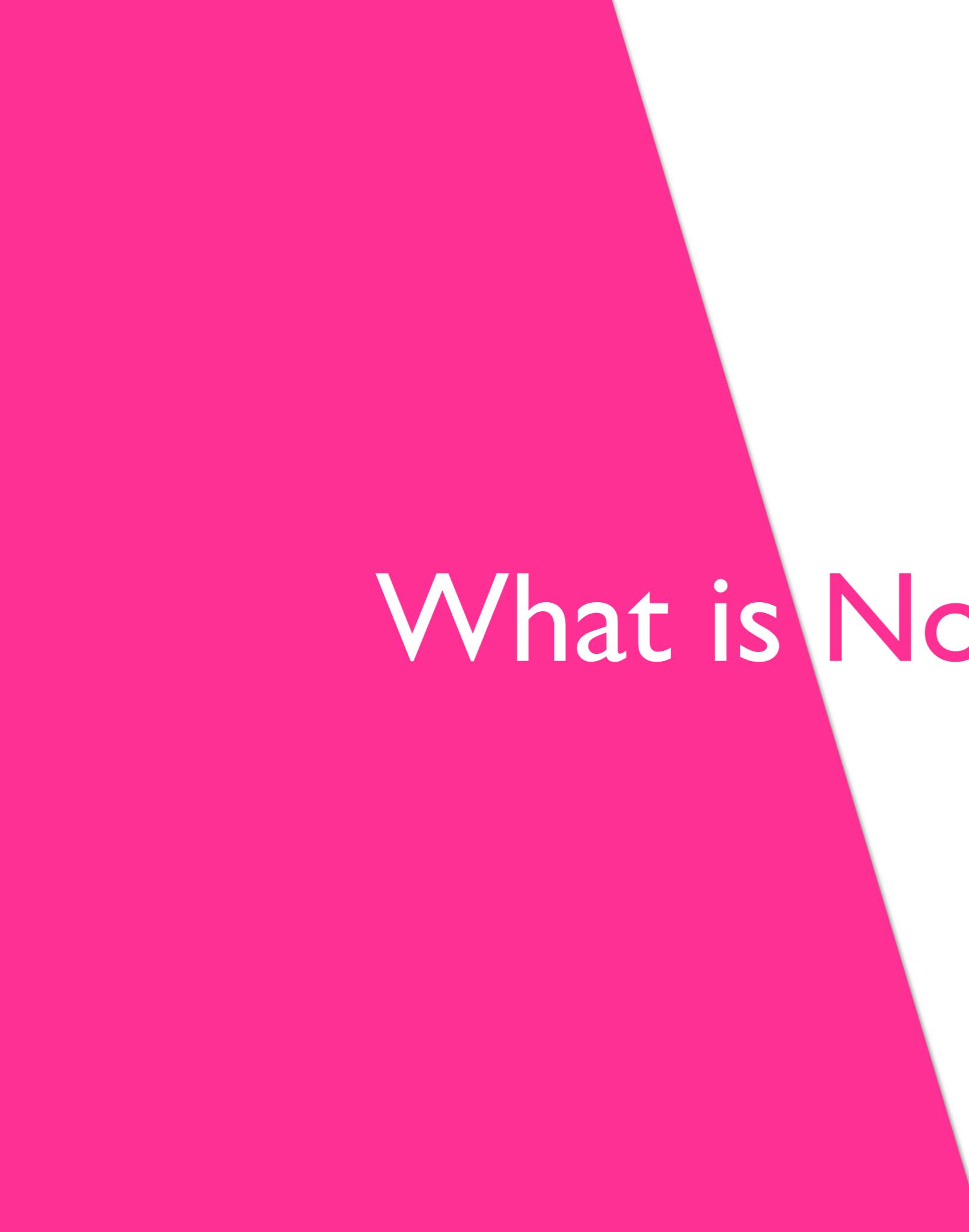
cse1500-ewi@tudelft.nl

# Web technology overview

1. HTTP: the language of web communication
2. HTML & web app design
3. **JavaScript**: interactions in the browser
4. **Node.js**: JavaScript on the server
5. **CSS**: adding style
6. Node.js: advanced topics
7. Cookies & sessions
8. Web security

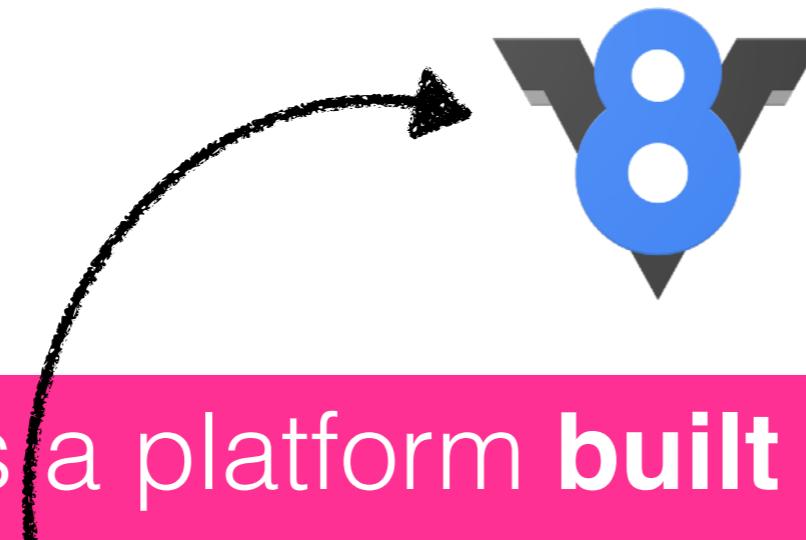
# Learning goals

- **Explain** the main ideas behind `Node.js`
- **Implement** basic network functionality with `Node.js`
- **Explain** the difference between `Node.js`, `npm` & `Express`
- **Create** a fully working Web application that has client- and server-side interactivity
- **Implement** client/server bidirectional communication through `WebSockets`



# What is Node.js?

# Node.js in its own words ...



“... intended for C++ programmers who want to embed the V8 JavaScript engine within a C++ application.”

“Node.js® is a platform **built on Google Chrome's JavaScript runtime** for easily building **fast, scalable network** applications.

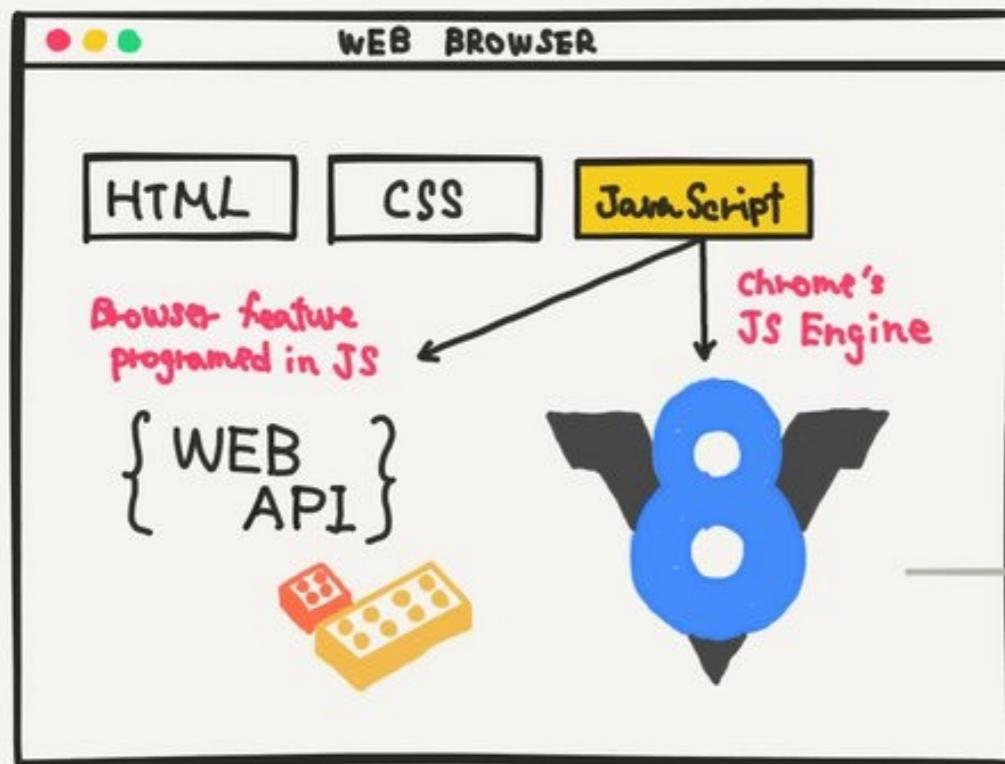
Node.js uses an **event-driven, non-blocking I/O** model that makes it lightweight and efficient, perfect for **data-intensive real-time applications** that run across distributed devices.”

# History of Node.js

- 2008: Google's **V8** engine is open-sourced.
- 2009: Node.js is released. It builds on V8.
- 2011: Node.js' package manager (npm) is released.
- December 2014: io.js is forked.
- May 2015: io.js merges back with Node.js. The **Node.js Foundation** steers the development.
- 2017: Node.js becomes a **first-class citizen of V8**.

## What is Node.js ??

What's browser JS ??

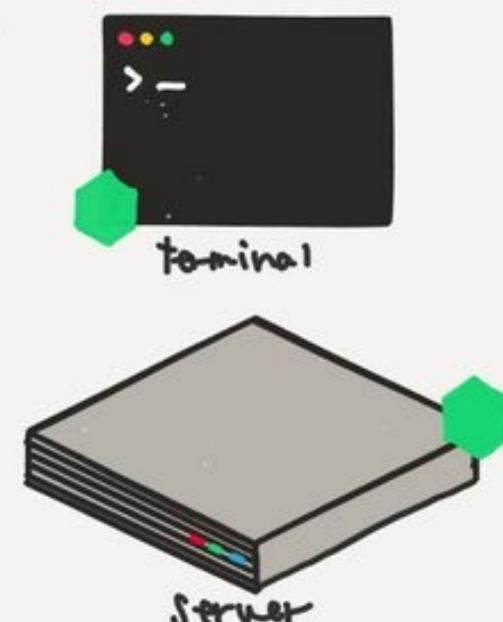
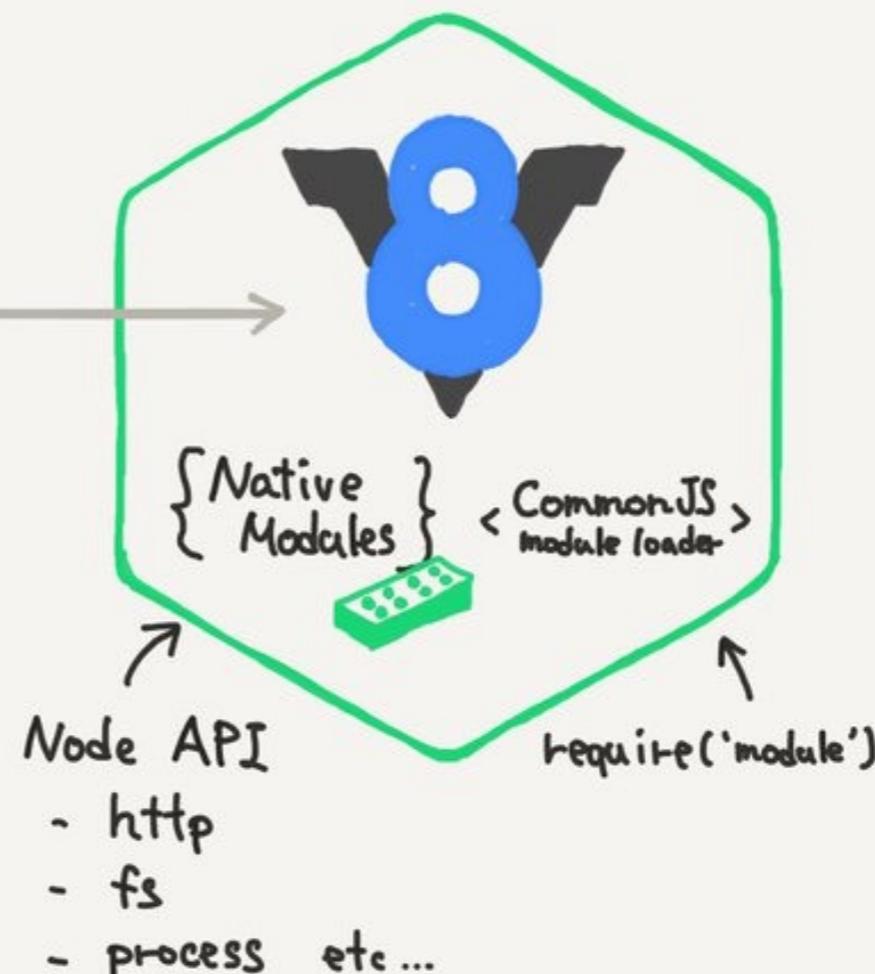


↑  
Web Platform  
- Window  
- Fetch  
- Web Audio etc...

↑  
Language features  
- ES6  
- promise  
- Typed Array etc ...

Web API & JavaScript Language are often described as the JavaScript (in intro books)

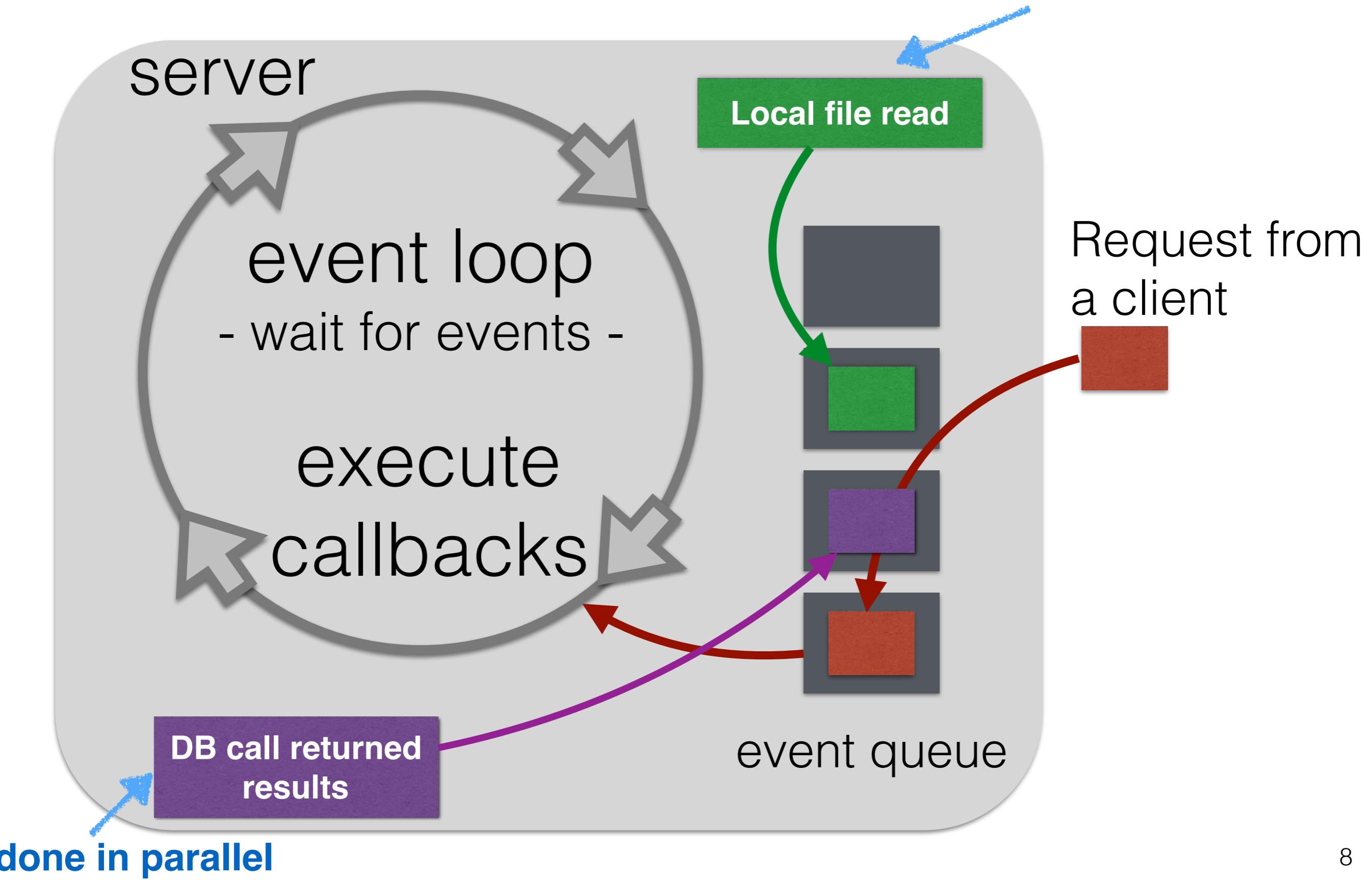
Node is a JavaScript ENVIRONMENT with Special API (like http) and default module loader.



↑  
Node runs on EVERYWHERE.

# Node.js is event-driven

done in parallel



# Node.js is event-driven

done in parallel



Node.js executes callbacks (event listeners) in response to an occurring event.

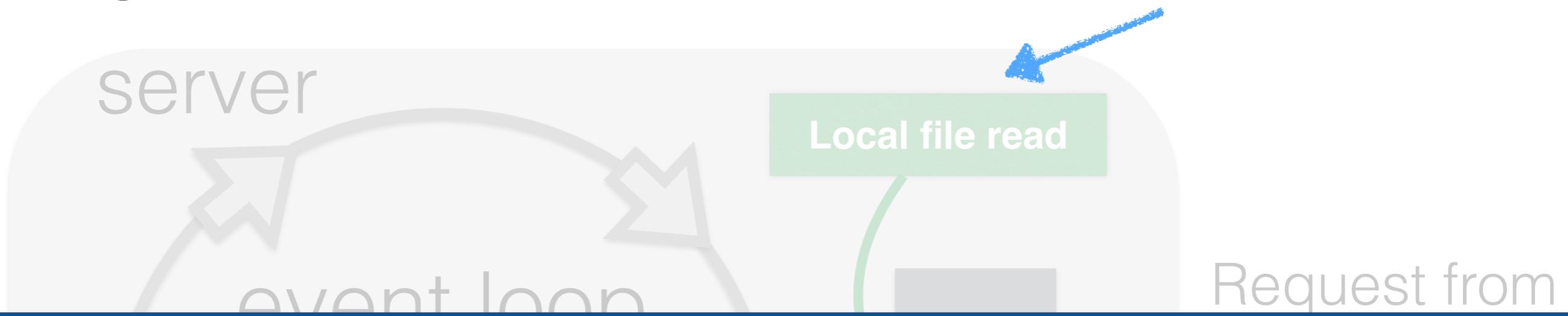
Developers write the **callbacks**.

DB call returned results

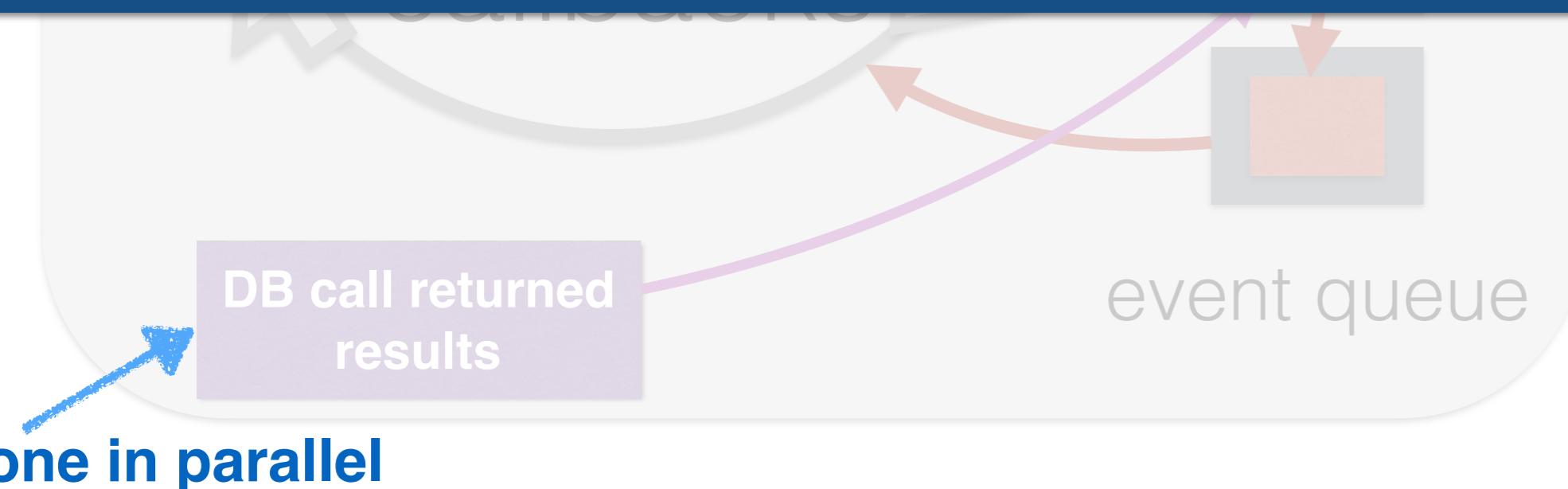
event queue

done in parallel

# Single-threaded?



- I/O requests are handled asynchronously
- **Event loop** is executed in a **single thread**
- Separate **thread pool** for I/O requests



# Async is the default

server

```
fs.readFile('/etc/passwd', function(err, data) {  
  if(err) throw err;  
  console.log(data);  
})
```

done in parallel

Local file read

callback: executed when file reading is complete

execute

```
let data = fs.readFileSync('/etc/passwd')
```

DB call returned  
results

event queue

done in parallel

# Node.js: single-threaded but highly parallel

- **I/O bound programs**: programs constrained by data access (adding more CPUs or main memory will not lead to large speedups)
- Many tasks might require **waiting time**
  - Waiting for a database to return results
  - Waiting for a third party Web service
  - Waiting for connection requests

Node.js is designed with these use cases in mind.

# Node.js in examples

Watching a file for  
changes ...

# Watching a file for changes

```
1 var fs = require('fs');
2
3 var file = process.argv[2];
4
5 fs.watch(file, function () {
6     console.log("File changed!");
7 });
8
9 console.log("Now watching " + file);
```

**Assumption:**

file to watch exists!

# Watching a file for changes

require()  
usually  
returns a  
JavaScript  
object

**Self-contained** piece of  
code that provides  
reusable functionality.

Node.js **fs module\***

```
1 var fs = require('fs');
2
3 fs.watch('todos.txt', function (err) {
4   if (err) throw err;
5   console.log('File changed');
6 })
7
8 // Executed immediately after the setup of the callback
9 console.log("Now watching " + file);
```

**callback**: defines what  
should happen when  
the file changes

**anonymous function**

**asynchronous**

**Assumption:**  
file to watch exists!

\* **module**: any file/directory that can be loaded by `require()`  
**package**: any file/directory described by a `package.json` file  
(most npm packages are modules)

# Low-level networking with Node.js

# Not just for web programming ...

- Built for **networked programming**
- Node.js has built-in support for **low-level** socket connections
- TCP socket connections have **two endpoints**
  1. **binds** to a numbered port
  2. **connects** to a port

## Analogous example: phone lines.

One phone binds to a phone number.

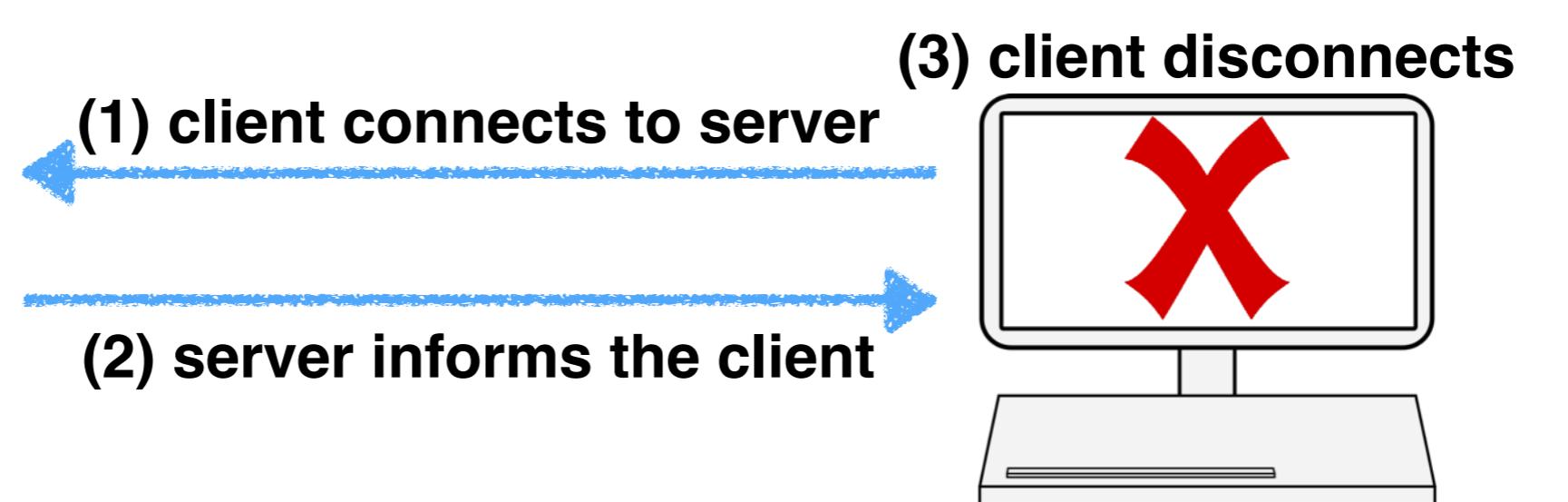
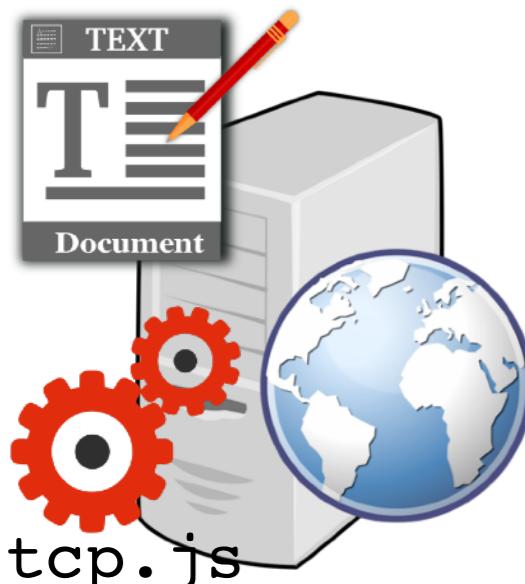
Another phone tries to call that phone.

If the call is answered, a connection is established.

# Client / server communication

**Task:** Server informs connected clients about changes to file todos.txt.

todos.txt



```
server:~ $  
Listening for subscribers ...  
Subscriber connected.  
Subscriber disconnected.
```

```
client:~ $  
Now watching todos.txt for  
changes ...  
File todos.txt changed: ...
```

# Callbacks all the way ...

**Task:** Server informs connected clients about changes to file todos.txt.

**Client:** telnet localhost <port>

```
1 const fs = require('fs');
2 const net = require('net');
3
4 //command line arguments: file to watch and port number
5 const filename = process.argv[2];
6 const port = process.argv[3];
7
8 var server = net.createServer(function (connection) {
9     // use connection object for data transfer
10 });
11
12 server.listen(port, function () {
13     console.log("Listening to subscribers...");
14 });
```

# Callbacks all the way ...

**Task:** Server informs connected clients about changes to file todos.txt.

```
Client: telnet localhost <port>
1 const fs = require('fs');
2 const net = require('net');
3
4 //command line arguments: file to watch and port to listen on
5 const serverObject = process.argv[2];
6 const port = process.argv[3];
7
8 var server = net.createServer(function (connection) {
9     // use connection object for data transfer
10 });
11
12 server.listen(port, function () {
13     console.log("Listening to subscribers...");
14 })
```

callback function is invoked when another endpoint connects

bind to a port

# Our first Hello World! web server

Node.js is **not** a Web server. It provides **functionality** to implement one!

# The “Hello World” of node.js

```
1 var http = require("http");
2
3 var port = process.argv[2];
4
5 var server = http.createServer(function (req, res) {
6   res.writeHead(200, { "Content-Type": "text/plain" });
7   res.end("Hello World!");
8   console.log("HTTP response sent");
9 })
10
11 server.listen(port, function () {
12   console.log("Listening on port " + port);
13 })
```

Start the **server** on the command line: **\$ node web.js 3000**  
Open the browser (**client**) at: <http://localhost:3000>

# The “Hello World” of node.js

node.js http module

```
1 var http = require("http");
2
3 var port = process.argv[2]; create a web server
4
5 var server = http.createServer(function (req, res) {
6   res.writeHead(200, { "Content-Type": "text/plain" });
7   res.end("Hello World!");
8   console.log("HTTP response sent");
9 }
10
11 server.listen(port, function () {
12   console.log("Listening on port " + port);
13 });
```

A **callback**: what to do  
if a request comes in

Create a HTTP  
response & send it

Start the **server** on the command line: **\$ node web.js 3000**  
Open the browser (**client**) at: <http://localhost:3000>

# The “Hello World” of node.js

node.js http module

```
1 var http = require("http");
2
3 var port = process.argv[2];
4
5 var server = http.createServer(function (req, res) {
6   res.writeHead(200, { "Content-Type": "text/plain" });
7   res.end("Hello World!");
8   console.log("HTTP response sent");
9 })
10
11 server.listen(port, function () {
12   console.log("Listening on port " + port);
13 })
```

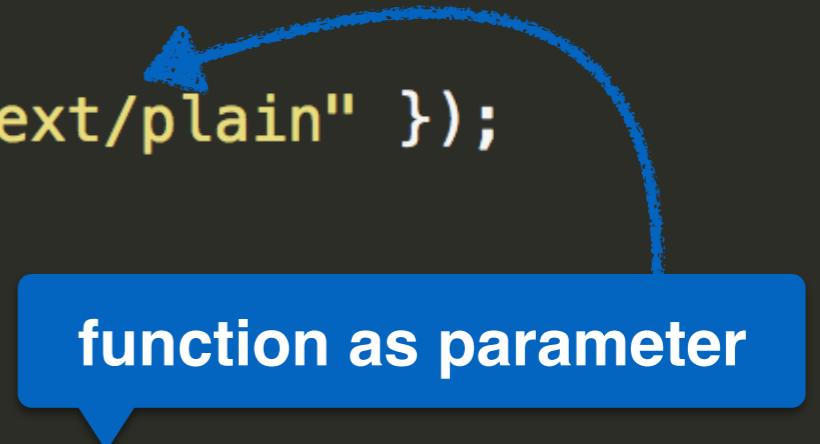
HTTP **request** object  
A **callback**: what to do  
create a web server  
HTTP **response** object

Create a HTTP response & send it

Start the **server** on the command line: **\$ node web.js 3000**  
Open the browser (**client**) at: <http://localhost:3000>

# A little refactoring ...

```
1 var http = require("http");
2
3 var port = process.argv[2];
4
5 function simpleHTTPResponder(req, res){
6     res.writeHead(200, { "Content-Type": "text/plain" });
7     res.end("Hello World!");
8     console.log("HTTP response sent");
9 }
10
11 var server = http.createServer(simpleHTTPResponder);
12
13 server.listen(port, function () {
14     console.log("Listening on port " + port);
15 });
```



function as parameter

# Using URLs for routing

```
1 var http = require("http");
2 var url = require("url");
3
4 var port = process.argv[2];
5
6 function simpleHTTPResponder(req, res) {
7
8     //parse the URL
9     var uParts = url.parse(req.url, true);
10
11    //implemented path
12    if (uParts.pathname == "/greetme"){
13        res.writeHead(200, { "Content-Type": "text/plain" });
14
15        //parse the query
16        var query = uParts.query;
17        var name = "Anonymous";
18
19        if (query["name"] != undefined){
20            name = query["name"];
21        }
22
23        res.end(" Greetings "+name);
24    }
25    //all other paths
26    else {
27        res.writeHead(404, { "Content-Type": "text/plain" });
28        res.end("Only /greetme is implemented.");
29    }
30}
31
32 var server = http.createServer(simpleHTTPResponder);
33
34 server.listen(port, function () {
35     console.log("Listening on port " + port);
36});
```

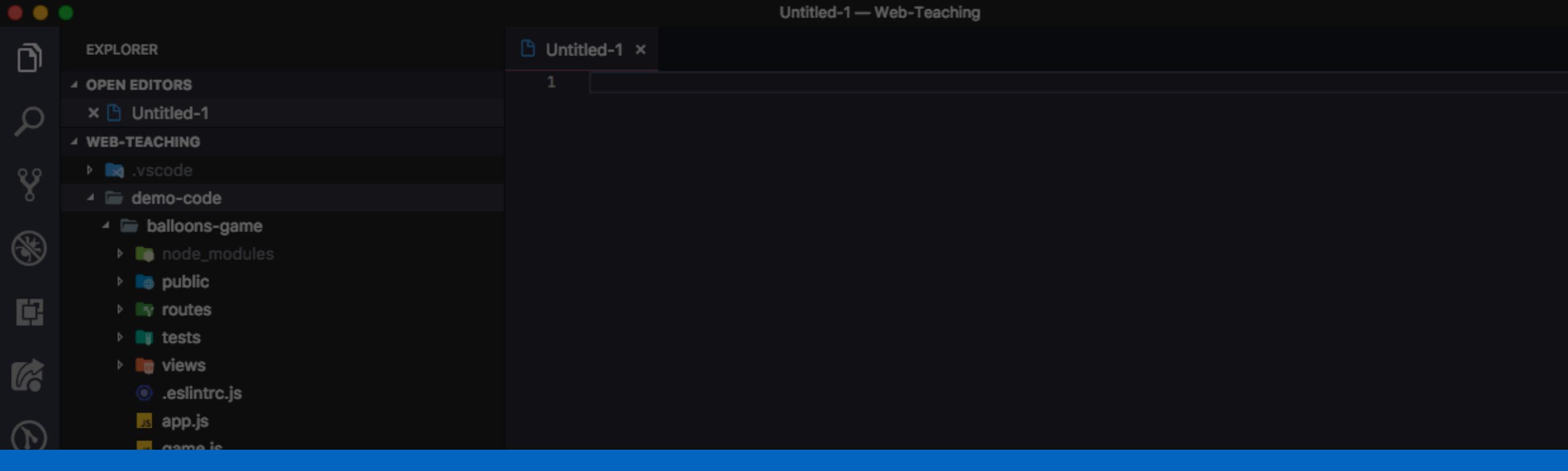
# Using URLs for routing

```
1 var http = require("http");
2 var url = require("url");
3
4 var port = process.argv[2];
5
6 function simpleHTTPResponder(req, res) {
7
8     //parse the URL
9     var uParts = url.parse(req.url, true);
10
11    //implemented path
12    if (uParts.pathname == "/greetme"){
13        res.writeHead(200, { "Content-Type": "text/plain" });
14
15        //parse the query
16        var query = uParts.query;
17        var name = "Anonymous";
18
19        if (query["name"] != undefined){
20            name = query["name"];
21        }
22
23        res.end(" Greetings "+name);
24    }
25    //all other paths
26    else {
27        res.writeHead(404, { "Content-Type": "text/plain" });
28        res.end("Only /greetme is implemented.");
29    }
30}
31
32 var server = http.createServer(simpleHTTPResponder);
33
34 server.listen(port, function () {
35     console.log("Listening on port " + port);
36});
```

if the **pathname** is  
**/greetme** we greet

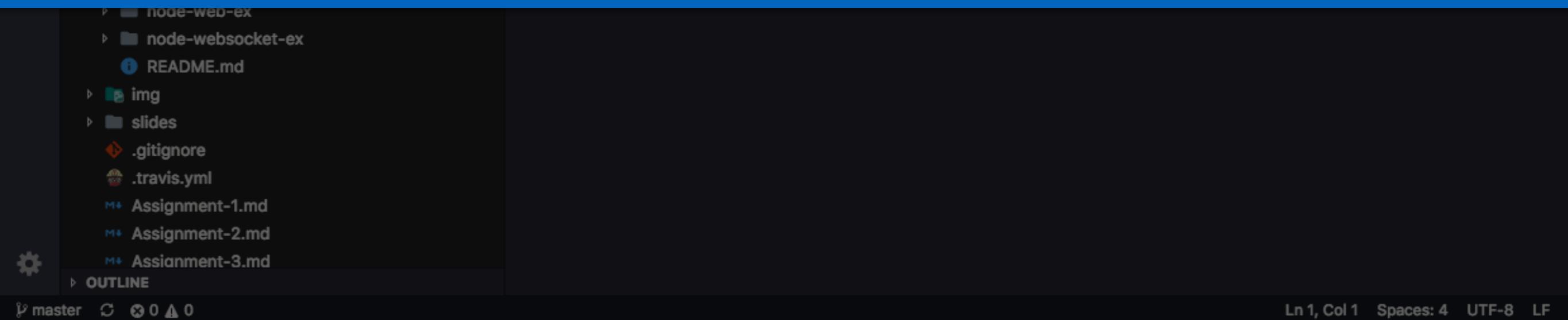
we can **extract params**  
from the **URL**

otherwise send  
back a **404 error**



Low-level node.js capabilities are important to know about  
(you don't always need a Web server), but ...

Tedious to write an HTTP server this way.  
How do you send CSS files and images?





Express

# Express

- Node.js has a **small core** code base
- Node.js comes with some **core modules included**
- Express is not one of them (but we have **NPM**)

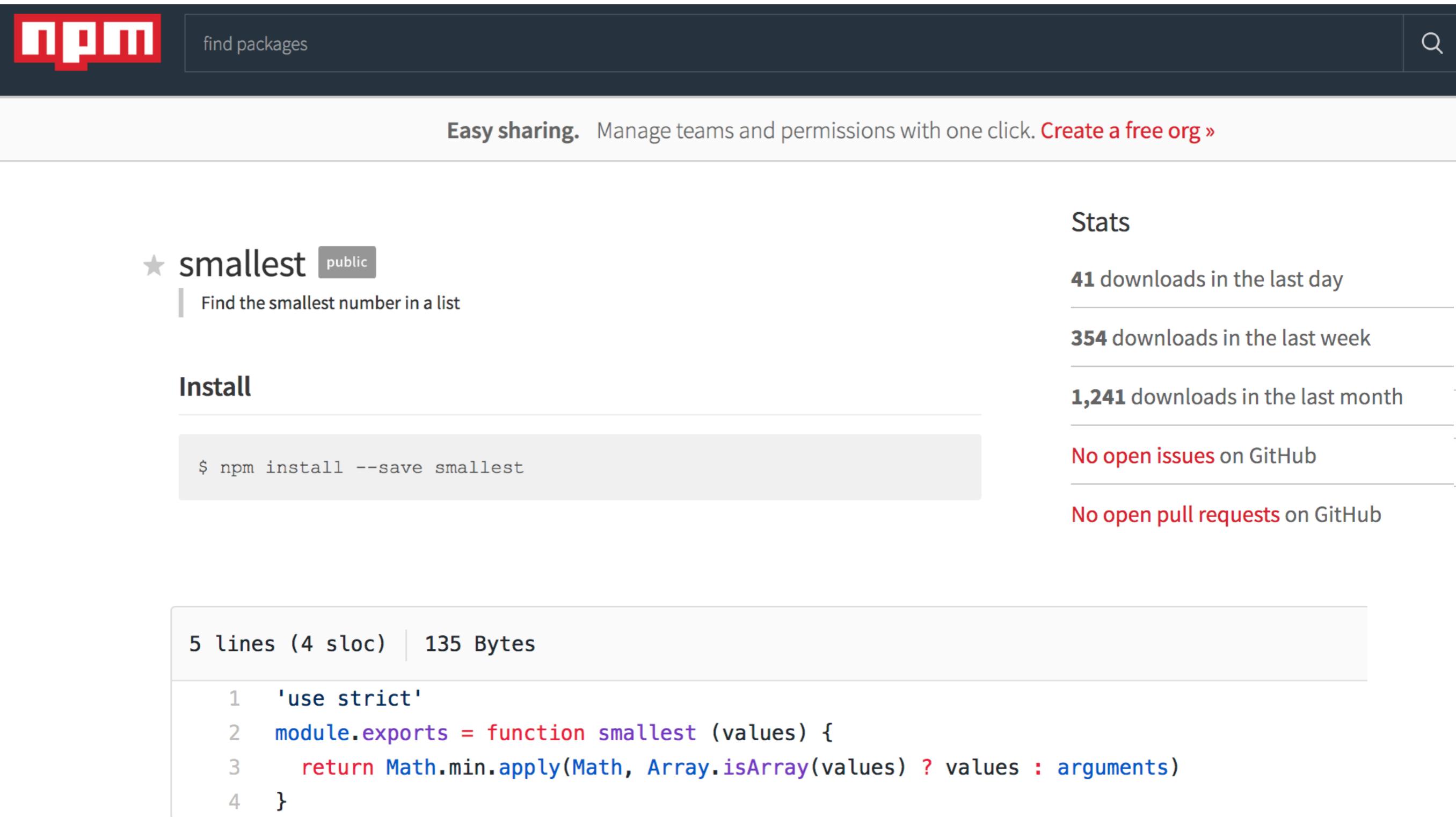
a collection of code with  
a public interface

```
$ cd my-project-folder  
$ npm init -y  
$ npm install express --save
```

node package manager

“The Express module creates a **layer on top of the core http module** that handles a lot of **complex things** that we don’t want to handle ourselves, like **serving up static** HTML, CSS, and client-side JavaScript files.” (**Web course book, Ch. 6**)

# Packages do not have to do a lot



The screenshot shows the npm package page for 'smallest'. The top navigation bar includes the npm logo, a search bar with 'find packages', and a magnifying glass icon. Below the header, a banner promotes 'Easy sharing' and 'Create a free org'. The main content area features the package name 'smallest' with a star icon and a 'public' badge. A brief description states 'Find the smallest number in a list'. The 'Install' section contains the command '\$ npm install --save smallest'. To the right, the 'Stats' section displays download counts: 41 in the last day, 354 in the last week, and 1,241 in the last month. It also shows 'No open issues' and 'No open pull requests' on GitHub. At the bottom, the code implementation is shown:

```
5 lines (4 sloc) | 135 Bytes
```

```
1 'use strict'
2 module.exports = function smallest (values) {
3     return Math.min.apply(Math, Array.isArray(values) ? values : arguments)
4 }
```

# The “Hello World” of Express

```
1 var express = require("express");
2 var url = require("url");
3 var http = require("http");
4
5 var port = process.argv[2];
6 var app = express();
7 http.createServer(app).listen(port);
8
9 var htmlPrefix = "<!DOCTYPE html><html><head></head><body><h1>";
10 var htmlSuffix = "</h1></body></html>";
11
12 app.get("/greetme", function(req, res){
13     var query = url.parse(req.url, true).query;
14
15     var name = ( query["name"] !=undefined) ? query["name"] : "Anonymous";
16
17     res.send(htmlPrefix+"Greetings "+name+htmlSuffix);
18 })
19
20 app.get("/goodbye", function(req, res){
21     res.send(htmlPrefix + "Goodbye to you too!" + htmlSuffix);
22 })
23
24 app.get("*", function(req, res){
25     res.send("Not a valid route ...");
26 })
```

# The “Hello World” of Express

```
1 var express = require("express");
2 var url = require("url");
3 var http = require("http");
4
5 var port = process.argv[2];
6 var app = express();
7 http.createServer(app).listen(port);
8
9 var htmlPrefix = "<!DOCTYPE html><html><head></head><body><h1>";
10 var htmlSuf
11
12 app.get("/greetme", function(req, res){
13     var query = url.parse(req.url, true).query;
14
15     var name = ( query["name"] !=undefined) ? query["name"] : "Anonymous";
16
17     res.send(htmlPrefix + name + htmlSuf);
18 })
19
20 app.get("/goodbye", function(req, res){
21     res.send("Goodbye to you too!" + htmlSuf);
22 })
23
24 app.get("*", function(req, res){
25     res.send("Not a valid route ..."));
26 })
```

**app object is our way to use Express' abilities**

**URL “route” set up**

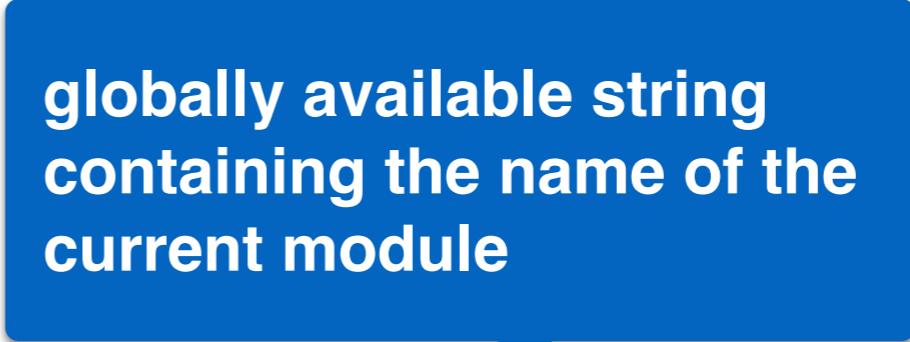
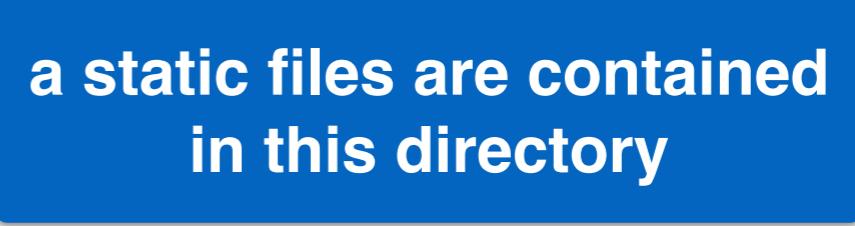
**another route**

**all other routes**

**Express creates HTTP headers for us**

**error-prone, not maintainable,**  
fails at anything larger than a  
toy project.

# Express and its static file server

- **Static files**: files that are not created/changed on the fly
    - CSS
    - JavaScript (client-side)
    - HTML
    - Images, video, etc.
  - A single line of code is sufficient to serve static files:  
`app.use(express.static(__dirname + "/static"));`
  - Express always **first** checks the static files for a given route - if not found, the dynamic routes are checked
- 
- globally available string containing the name of the current module
- 
- a static files are contained in this directory



# How to build a Web application

# Development strategy

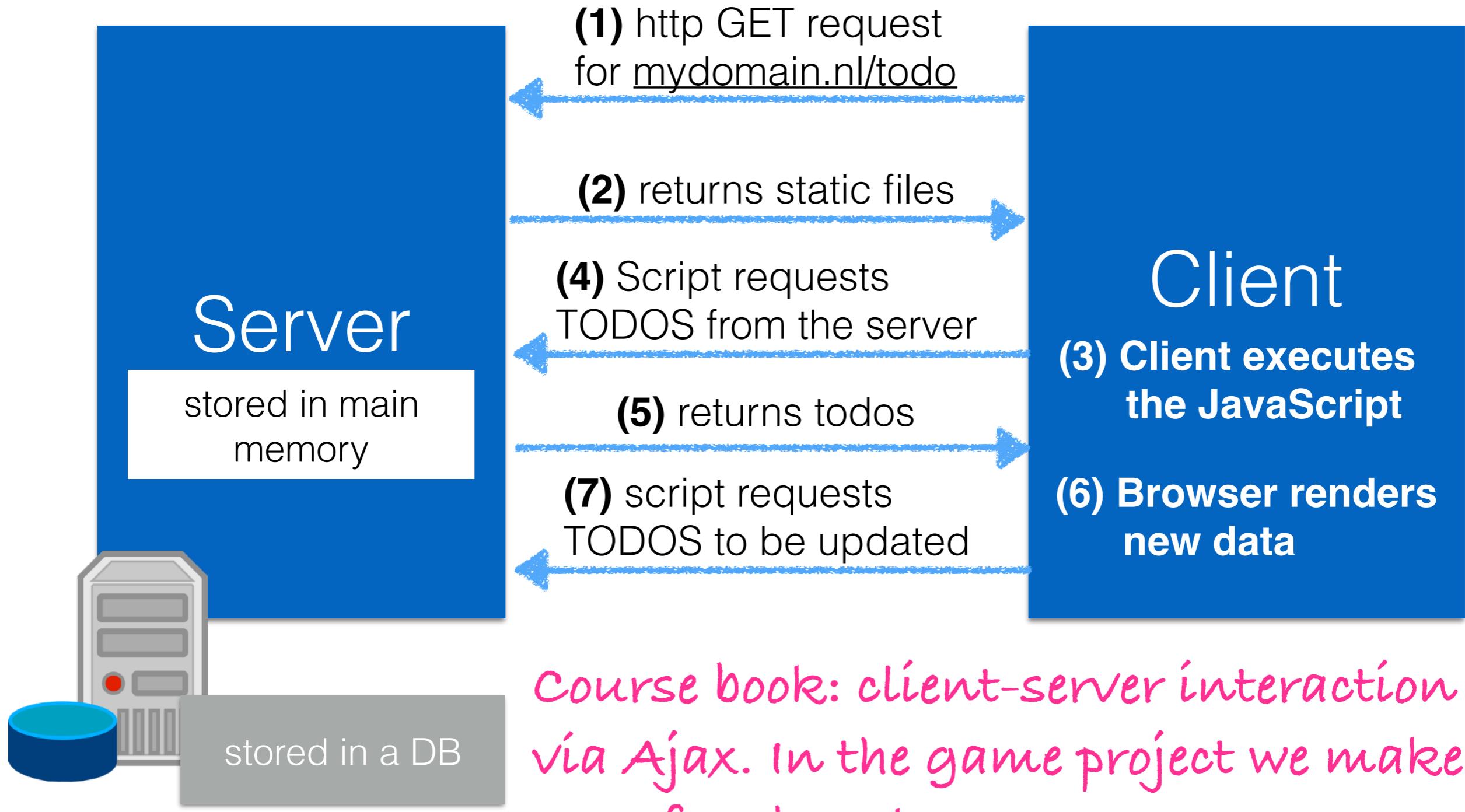
1. Develop the **client-side code** (HTML, CSS, JavaScript)
2. Place all files into some directory (e.g. /client) **on the server**
3. Write the **Node.js server code** using Express
4. Set **Express' static file path** to the directory of step 2
5. Add interactivity between client and server



```
app.js  
client/  
    index.html  
    html/  
        =>game.html  
        =>splash.html  
    images/  
        =>background.png  
        =>logo.png  
    css/  
        =>layout.css  
        =>style.css  
    javascript/  
        =>client-app.js
```

Boilerplate structure and code: code generators help (A2)!

# Typical web app flow (TODO app)





# JSON: exchanging data between the client and server

# XML vs. JSON

```
1 <!--?xml version="1.0"?-->
2 <timezone>
3   <location></location>
4   <offset>1</offset>
5   <suffix>A</suffix>
6   <localtime>20 Jan 2014 02:39:51</localtime>
7   <isotime>2014-01-20 02:39:51 +0100</isotime>
8   <utctime>2014-01-20 01:39:51</utctime>
9   <dst>False</dst>
10 </timezone>
```

XML

```
1 {
2   "timezone": {
3     "offset": "1",
4     "suffix": "A",
5     "localtime": "20 Jan 2014 02:39:51",
6     "isotime": "2014-01-20 02:39:51 +0100",
7     "utctime": "2014-01-20 01:39:51",
8     "dst": "False"
9   }
10 }
```

JSON

# Exchanging data

- Ten/twenty years ago **XML** was the standard data exchange format - well defined, not easy to handle
- JSON (**JavaScript Object Notation**) was developed in the early 2000s by **Douglas Crockford**
- XML is often too **bulky**; JSON has a smaller footprint
- JSON to JS object and the reverse is possible using **built-in JavaScript** functions (`JSON.parse`, `JSON.stringify`)

# JSON vs. JavaScript objects

- JSON: all property names must be enclosed in quotes
- Objects created from JSON **do not have functions** as properties (functions are stripped in the conversion to JSON)
- Express has a dedicated response object method to send JSON: `res.json(param)`

# On the server: sending JSON

```
1 var express = require("express");
2 var url = require("url");
3 var http = require("http");
4 var app;
5
6 var port = process.argv[2];
7 app = express();
8 http.createServer(app).listen(port);
9
10 var todos = [];   Array
11 var t1 = { message : "Maths homework due", type : 1, deadline : "12/12/2014"};
12 var t2 = { message : "English homework due", type : 3, deadline : "20/12/2014"};
13 todos.push(t1);
14 todos.push(t2);
15
16 app.get("/todos", function (req, res) {
17     res.json(todos);
18 });

converted on the fly
```

# Ajax: dynamic updating on the client

# What is Ajax?

- Asynchronous JavaScript and XML: XML is in the name only
- Ajax enables dynamic loading of content **without refetching the entire page**
- Ajax is a technology that **injects** new data into an existing document
- jQuery **hides a lot of complexity** and makes Ajax calls easy

# What is Ajax?

The screenshot shows a Twitter profile page for Claudia Hauff (@CharlotteHase) and a feed of tweets. Below the browser window is the developer tools Network tab, displaying a list of requests. An orange callout bubble labeled "XMLHttpRequest" points to a request for "timeline...". Another orange callout bubble labeled "Just a few bytes" points to a request for "368 B".

Home Notifications Messages

Search Twitter

Who to follow · Refresh · View all

Followed by julia kiseleva and others

Kathy Brennan, PhD @kn... Follow

Followed by Aleksandr Chuklin and others

Aldo Lipani @AldoLipani Follow

Followed by Aleksandr Chuklin and others

What's happening?

Oxford Comp Sci Retweeted

Open Data Institute @ODIHQ · 17h

Announced by @Nigel\_Shadbolt: Office for AI working with @ODIHQ to pilot data trusts to examine how a 'data trust' could increase access to data while retaining trust theodi.org/article/ufs-fi... #ODISummit

Dura Vermeer @DuraVermeer · Nov 12

In Almere bouwden we het nieuwe Carnival Training Center: innovatief,

Tweets 3,954 Following 840 Followers 1,741

Germany trends · Change

Inspector Console Debugger Style Editor Performance Memory Network Storage

Filter URLs

All HTML CSS JS XHR Fonts Images Media WS Other Persist Logs

| Status | Method | F...        |   |            |      | Size     | Time   | Time     |
|--------|--------|-------------|---|------------|------|----------|--------|----------|
| 304    | GET    | Kkhapl...   | 🔒 |            |      | 7.37 KB  | → 6 ms | 20.48 s  |
| 304    | GET    | BJ5Cm...    | 🔒 | vid... xhr | mp2t | 27.72 KB | → 4 ms | 40.96 s  |
| 200    | GET    | timeline... | 🔒 | twi... xhr | js   | 1.42 KB  | 368 B  | → 178 ms |
| 200    | GET    | timeline... | 🔒 | twi... xhr | js   | 1.42 KB  | 368 B  | → 168 ms |
| 200    | GET    | toast_p...  | 🔒 | twi... xhr | js   | 1.36 KB  | 236 B  | → 192 ms |
| 200    | GET    | timeline... | 🔒 | twi... xhr | js   | 1.42 KB  | 368 B  | → 185 ms |
| 200    | GET    | timeline... | 🔒 | twi... xhr | js   | 1.42 KB  | 368 B  | → 184 ms |
| 200    | GET    | timeline... | 🔒 | twi... xhr | js   | 1.42 KB  | 368 B  | → 184 ms |

163 requests 7.98 MB / 6.89 MB transferred Finish: 52.75 s DOMContentLoaded: 1.46 s load: 1.61 s

# What is Ajax?

The screenshot shows the Google homepage with a search bar containing 'tu del'. Below the search bar, a dropdown menu lists suggestions: 'tu delft', 'tu delft library', 'tu delft printen', 'tu delft webmail', 'tu delft mail', and 'tu delft logo'. The browser's developer tools are open, specifically the Network tab, which displays network requests for the current page. A large orange callout bubble points to the suggestion 'tu delft' in the dropdown, with the text 'Query suggestions' inside it. The Network tab table has columns for Status, Method, URL, Cause, Type, Transferred, and Response payload. The last row of the table, which corresponds to the 'tu delft' suggestion, is highlighted with a blue background and shows a response payload of `[[{"tu de": "tu delft"}]]`.

| Status | Method | F...       | D...            | Cause       | Type | Transferred | S...  |
|--------|--------|------------|-----------------|-------------|------|-------------|-------|
| 204    | POST   | gen_20...  | green lock icon | ww...beacon | html | 415 B       | 0 B   |
| 200    | GET    | search?... | green lock icon | ww...xhr    | json | 665 B       | 493 B |
| 200    | GET    | search?... | green lock icon | ww...xhr    | json | 656 B       | 513 B |
| 200    | GET    | search?... | green lock icon | ww...xhr    | json | 639 B       | 533 B |
| 200    | GET    | search?... | green lock icon | ww...xhr    | json | 638 B       | 533 B |
| 200    | GET    | search?... | green lock icon | ww...xhr    | json | 638 B       | 533 B |
| 200    | GET    | search?... | green lock icon | ww...xhr    | json | 638 B       | 533 B |
| 200    | GET    | search?... | green lock icon | ww...xhr    | json | 638 B       | 533 B |

Query suggestions

Response payload

```
1  }])
2  [[[{"tu de": "tu delft"}]]]
```

# On the client: HTML & JS

```
1 <!DOCTYPE html>
2 <head>
3   <title>Plain text TODOS</title>
4   <script src="http://code.jquery.com/jquery-3.3.1.min.js" type="text/javascript">
5     </script>
6   <script src="js/client-app.js" type="text/javascript"></script>
7 </head>
8
9 <body>
10  <main>
11    <section id="todo-section">
12      <p>My list of TODOS:</p>
13      <ul id="todo-list">
14        </ul>
15    </section>
16  </main>
17 </body>
18 </html>
```

Load the JavaScript files, **start with jQuery**

Define where the TODOs will be added. Empty list.

# On the client: HTML & JS

```

1 <!DOCTYPE html>
2 <head>
3   <title>Plain text TODOS</title>
4   <script src="http://code.jquery.com/jquery-3.3.1.min.js" type="text/javascript">
5   </script>
6   <script src="js/client-app.js" type="text/javascript"></script>
7 </head>
8
9 <body>
10 <main>
11   <section id="todo-section">
12     <p>My list of TODOS:</p>
13     <ul id="todo-list">
14       </ul>
15   </section>
16 </main>
17 </body>
18 </html>

```

when the document is loaded, execute `main()`

Load the JavaScript files, **start with jQuery**

Define where the TODOs will be added. Empty list.

**Callback:** define what happens when a todo object is available

Dynamic insert of list elements into the DOM

Ajax

```

1 var main = function () {
2   "use strict";
3
4   var addTodosToList = function (todos) {
5     console.log("Loading todos from server");
6     var todolist = document.getElementById("todo-list");
7     for (var key in todos) {
8       var li = document.createElement("li");
9       li.innerHTML = "TODO: " + todos[key].message;
10      todolist.appendChild(li);
11    }
12  };
13
14  /*
15   * This request retrieves the todo list once, to make this a regular
16   * "event", make use of setInterval()
17   */
18  $.getJSON("../todos", addTodosToList)
19    .done(function(){ console.log("Ajax request successful.");})
20    .fail(function(){ console.log("Ajax request failed.");});
21
22 $(document).ready(main);

```

# WebSockets

# WebSockets

- **Bidirectional communication** between client and server
- **Protocol upgrade** initiated by the client (status code 101)
- WebSocket servers can share a port with HTTP servers
- WebSocket protocol is simple, co-exists peacefully with HTTP
- ws: a Node.js WebSocket library (the most popular one)

# WebSockets: on the client

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>WebSocket test</title>
5   </head>
6   <body>
7     <main>
8       Status: <span id="hello"></span>
9     </main>
10
11    <script>
12      var socket = new WebSocket("ws://localhost:3000");
13      socket.onmessage = function(event){
14        document.getElementById("hello").innerHTML = event.data;
15      }
16
17      socket.onopen = function(){
18        socket.send("Hello from the client!");
19        document.getElementById("hello").innerHTML = "Sending a first message to
20          the server ...";
21      };
22    </script>
23  </body>
24 </html>
```

Initiate WebSocket connection (scheme ws!)

A received message should be displayed

Message sent from client to server once the connection is open

# WebSockets: on the server

```
1 var express = require("express");
2 var http = require("http");
3 var websocket = require("ws");
4
5 var port = process.argv[2];
6 var app = express();
7
8 app.use("/", function(req, res) {
9   res.sendFile("client/index.html", {root: "./"});
10 });
11
12 var server = http.createServer(app);
13
14 const wss = new websocket.Server({ server }); // WebSocket server object
15
16 wss.on("connection", function(ws) {
17   //let's slow down the server response time a bit
18   //to make the change visible on the client side
19   setTimeout(function() {
20     console.log("Connection state: " + ws.readyState);
21     ws.send("Thanks for the message. --Your server.");
22     ws.close();
23     console.log("Connection state: " + ws.readyState);
24   }, 2000);
25
26   ws.on("message", function incoming(message) { // Callback for the message event
27     console.log("[LOG] " + message);
28   });
29 });
30
31 server.listen(port);
```

WebSocket server object

We define what happens when a connection is established

Callback for the message event

# WebSockets: on the server

The WebSocket protocol as described in [RFC 6455](#) has four event types:

- `open` : this event fires once a connection request has been made and the handshake was successful; messages can be exchanged now;
- `message` : this event fires when a message is received;
- `error` : something failed;
- `close` : this event fires when the connection closes; it also fires after an `onerror` event;

```
11
12 var server = http.createServer(app);
13
14 const wss = new websocket.Server({ server });
15
16 wss.on("connection", function(ws) {
17   //let's slow down the server response time a bit
18   //to make the change visible on the client side
19   setTimeout(function() {
20     console.log("Connection state: " + ws.readyState);
21     ws.send("Thanks for the message. --Your server.");
22     ws.close();
23     console.log("Connection state: " + ws.readyState);
24   }, 2000);
25
26   ws.on("message", function incoming(message) {
27     console.log("[LOG] " + message);
28   });
29 });
30
31 server.listen(port);
```

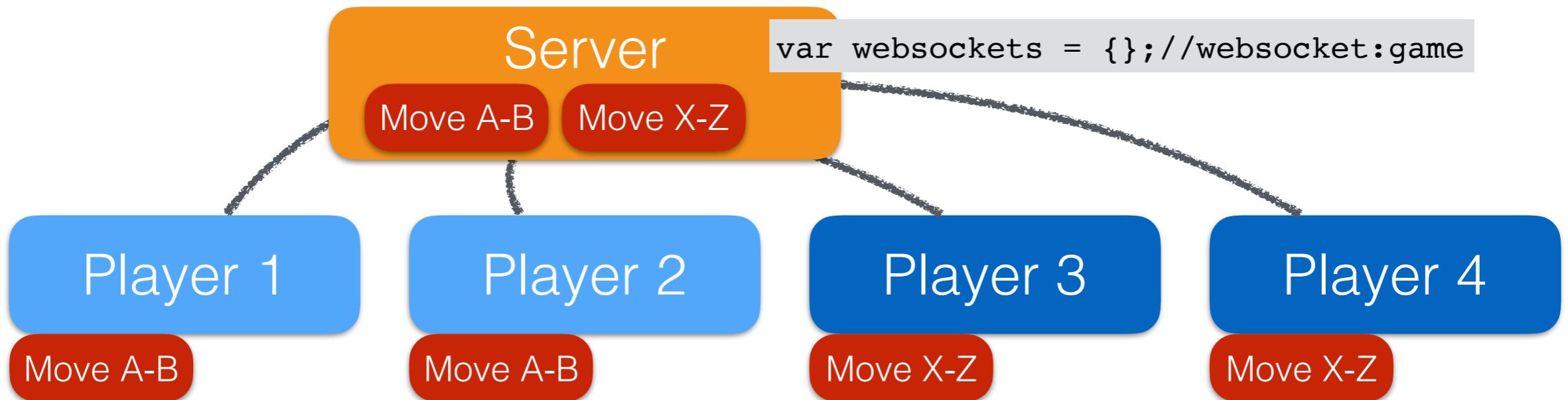
WebSocket server object

We define what happens when a connection is established

Callback for the message event

# WebSockets for multi-player games

- Every player establishes a WebSocket conn. to the server
- The server **tracks** the game each player belongs to:



```
1 var game = function (gameID) {  
2     this.playerA = null;  
3     this.playerB = null;  
4     this.id = gameID;  
5     this.wordToGuess = null;  
6     this.gameState = "0 JOINT";  
7 };
```

# WebSockets for multi-player games

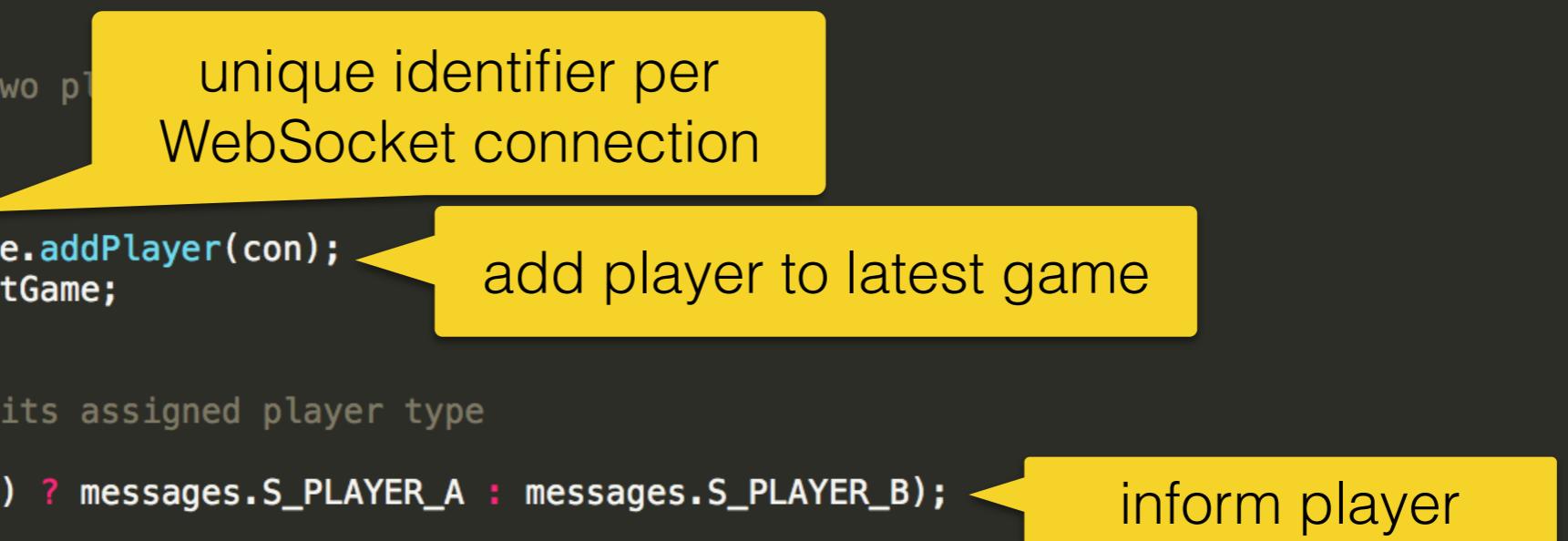
- Once a client establishes a WebSocket connection, the server-side script has several tasks:
  - Determine which game to add the player to
  - Inform the player about the game status
  - Request information from the player if necessary

```
1 var currentGame = new Game(gameStatus.gamesInitialized++);
2 var connectionID = 0;//each websocket receives a unique ID
3
4 wss.on("connection", function connection(ws) {
5
6     /*
7      * two-player game: every two players are added to the same game
8      */
9     let con = ws;
10    con.id = connectionID++;
11    let playerType = currentGame.addPlayer(con);
12    websockets[con.id] = currentGame;
13
14    /*
15     * inform the client about its assigned player type
16     */
17    con.send((playerType == "A") ? messages.S_PLAYER_A : messages.S_PLAYER_B);
18
19    ...
20}
```

# WebSockets for multi-player games

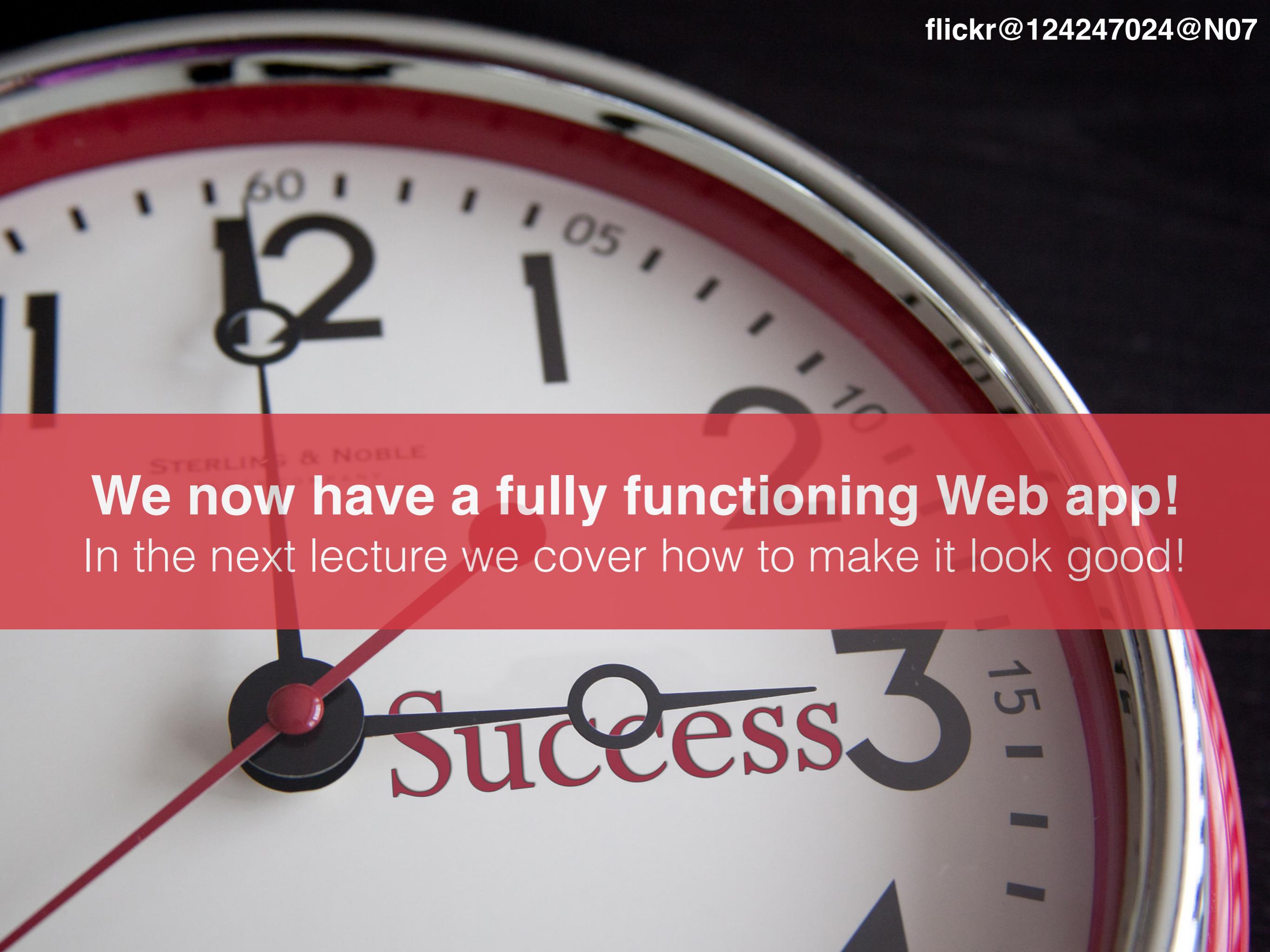
- Once a client establishes a WebSocket connection, the server-side script has several tasks:
  - Determine which game to add the player to
  - Inform the player about the game status
  - Request information from the player if necessary

```
1 var currentGame = new Game(gameStatus.gamesInitialized++);  
2 var connectionID = 0; //each websocket receives a unique ID  
3  
4 wss.on("connection", function connection(ws) {  
5  
6     /*  
7      * two-player game: every two players share one game  
8      */  
9     let con = ws;  
10    con.id = connectionID++;  
11    let playerType = currentGame.addPlayer(con);  
12    websockets[con.id] = currentGame;  
13  
14    /*  
15     * inform the client about its assigned player type  
16     */  
17    con.send((playerType == "A") ? messages.S_PLAYER_A : messages.S_PLAYER_B);  
18  
19    ...  
20}
```



The code is annotated with three yellow callout boxes:

- A callout box points to the line `con.id = connectionID++;` with the text "unique identifier per WebSocket connection".
- A callout box points to the line `currentGame.addPlayer(con);` with the text "add player to latest game".
- A callout box points to the line `con.send((playerType == "A") ? messages.S_PLAYER_A : messages.S_PLAYER_B);` with the text "inform player".



We now have a fully functioning Web app!

In the next lecture we cover how to make it look good!

- Read **chapter 3** (CSS) of the Web development book before the Monday lecture!
- Work through **Assignment 2** (the most difficult one - the bulk of the app is created).