Node.js: JavaScript on the server

Claudia Hauff
cse1500-ewi@tudelft.nl

TU Delft

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 …

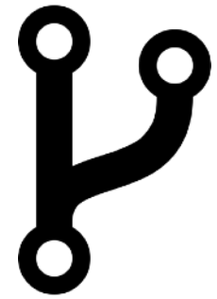
"… intended for C++ programmers who want to embed the V8 JavaScript engine within a C++ application."

"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.



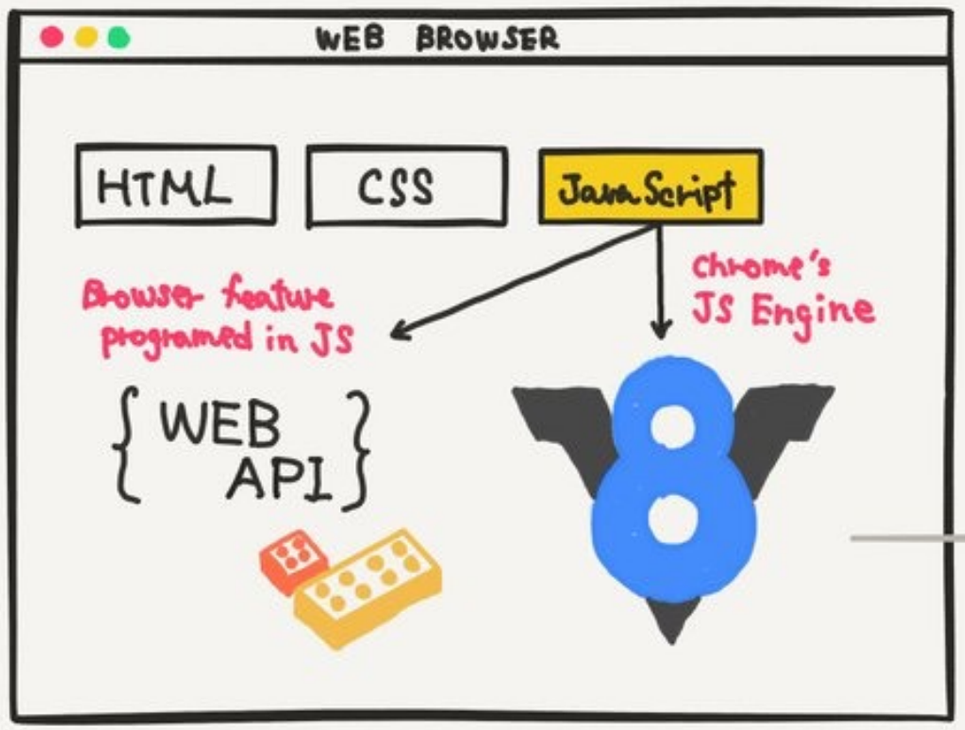
- 2017: Node.js becomes a **first-class citizen of V8**.

drawing by Mariko Kosaka (@kosamari)

@kosamari
March 3, 2017

What is Node.JS ??

What's browser JS ??

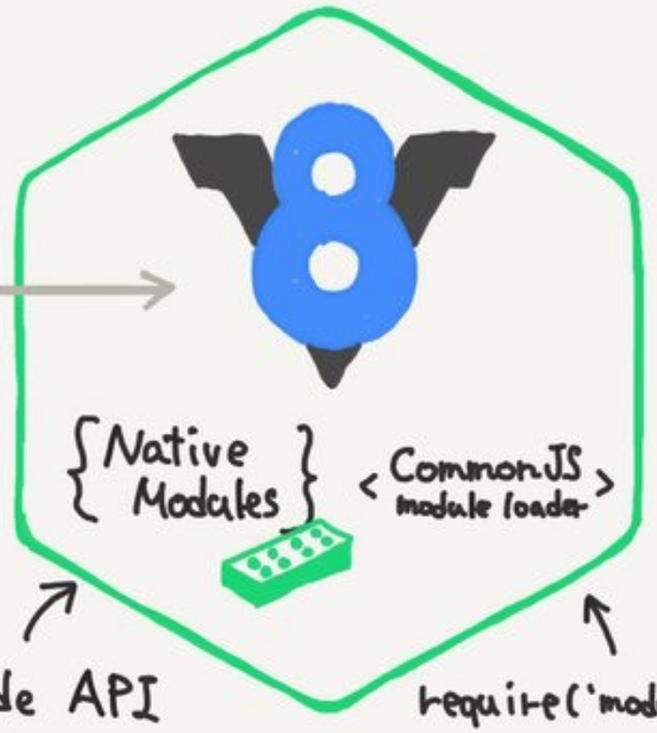


Web Platform
- Window
- Fetch
- Web Audio etc...

Language featurs
- 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 API
- http
- fs
- process etc...

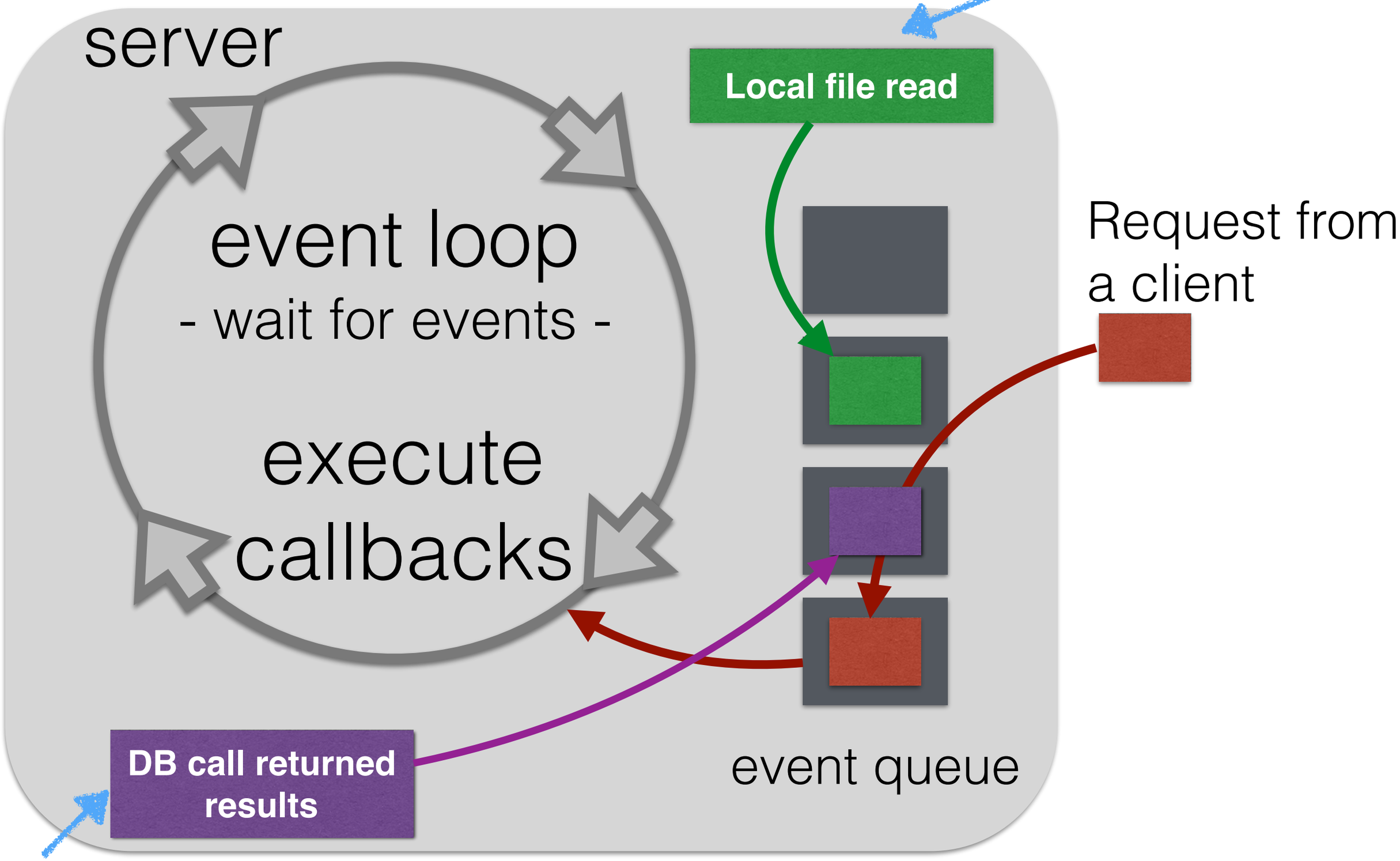

terminal


Server

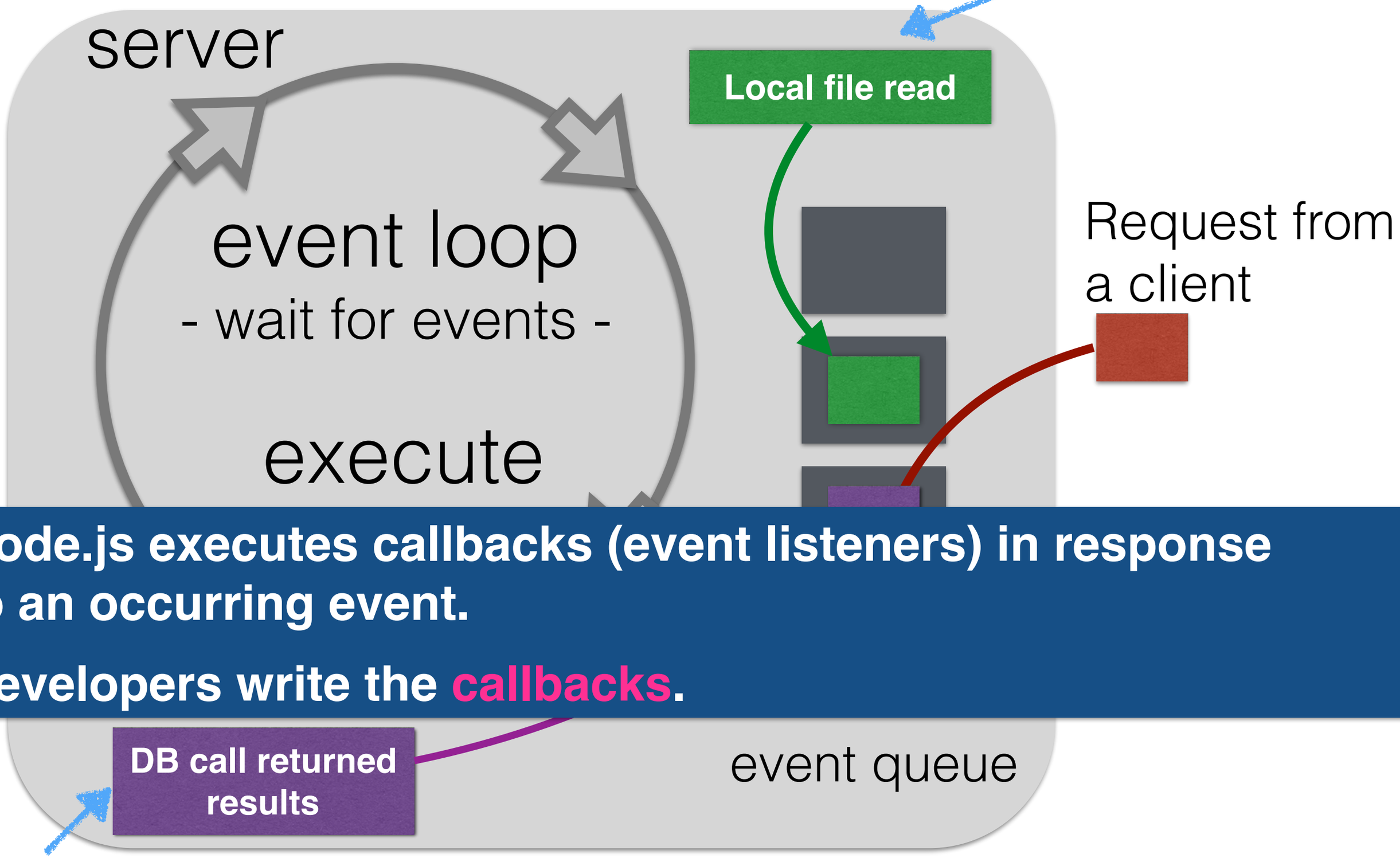

Hardware

Node runs on EVERYWHERE.

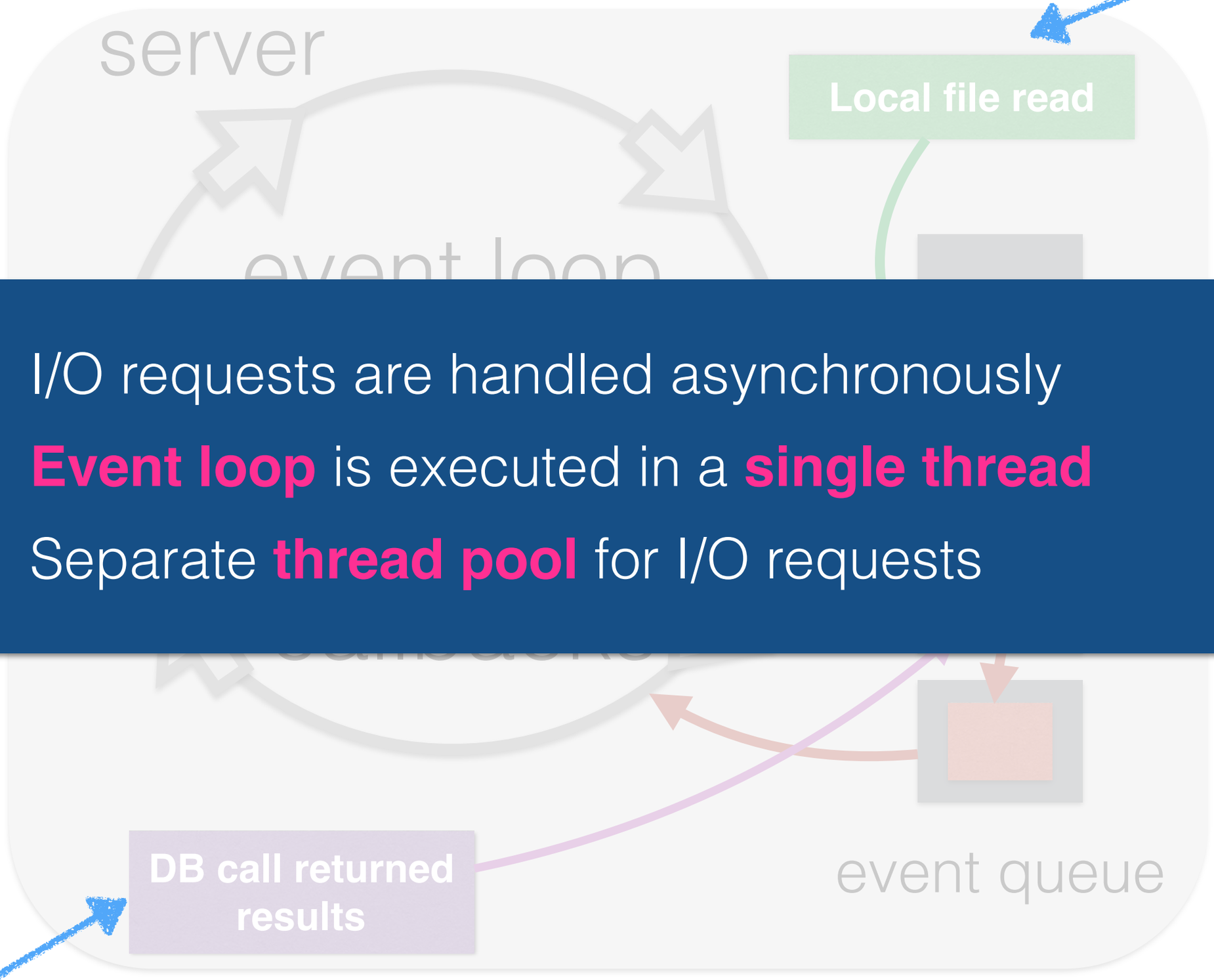
Node.js is event-driven



Node.js is event-driven



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

done in parallel

server

Local file read

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

callback: executed when file reading is complete

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

execute

event queue

DB call returned results

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**

Node.js `fs` **module***

Self-contained piece of code that provides reusable functionality.

```
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);
```

polls 'todos.txt' **for changes**

callback: defines what should happen when the file changes

anonymous function

asynchronous

Executed immediately after the **setup** of the callback

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.

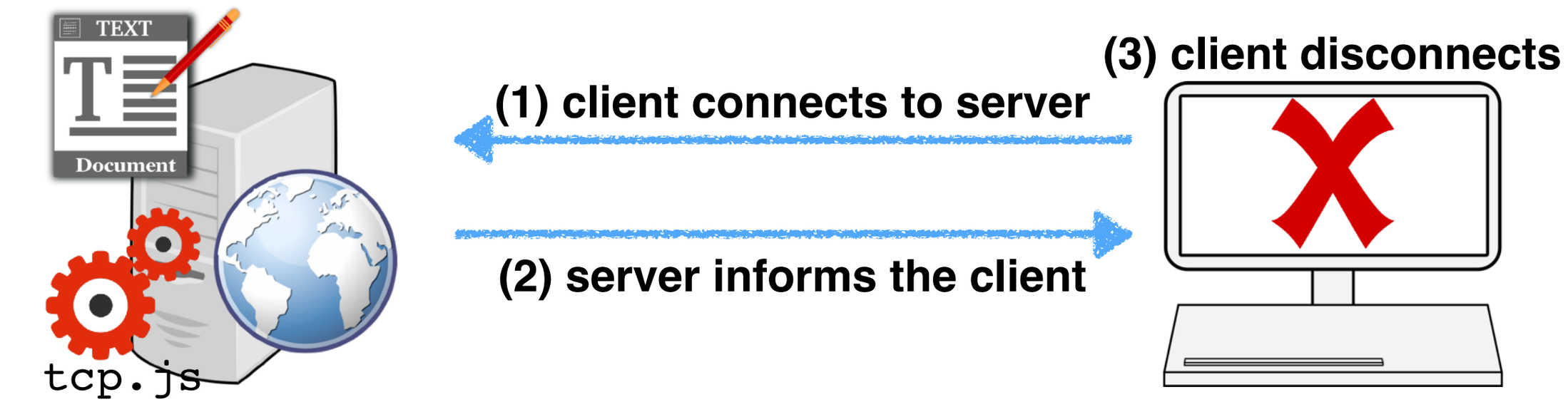
One phone binds to a phone number.

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`.




```
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
5 const                    = process.argv[2];
6 const                    ocess.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 });
```

server **object** is returned

callback function is invoked when another endpoint connects

bind to a 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
10     //what to do on connect
11     console.log("Subscriber connected");
12     connection.write("Now watching " + filename +
13         " for changes\n");
14
15     var watcher = fs.watch(filename, function () {
16         connection.write("File " + filename +
17             " has changed: " + Date.now() + "\n");
18     });
19
20     //what to do on disconnect
21     connection.on('close', function () {
22         console.log("Subscriber disconnected");
23         watcher.close();
24     });
25 });
26
27 server.listen(port, function () {
28     console.log("Listening to subscribers...");
29 });
```

client-side output

server-side output

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 });
```

A callback: what to do if a request comes in

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

HTTP **request** object

create a web server

HTTP **response** object

A callback: what to do es in

```
1  var http = require("http");
2
3  var port = process.argv
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 });
```

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 });
```

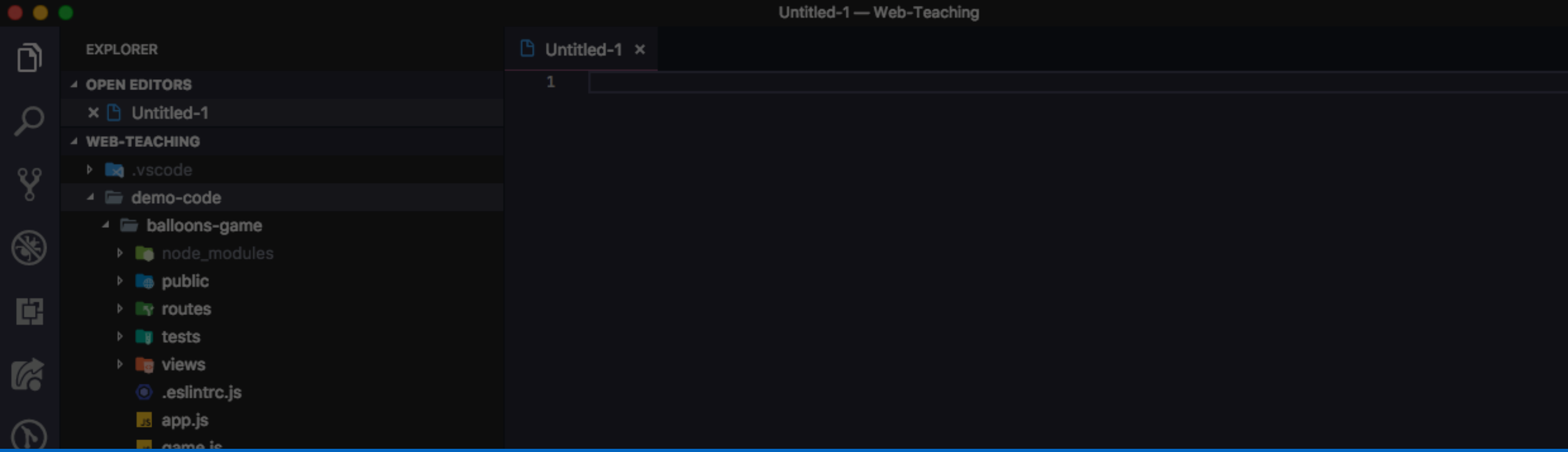
Using URLs for routing

```
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

node package manager

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

"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)**

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
4
5  var
6  var app = express();
7  http.createServer(app).listen(port);
8
9  var htmlPrefix = "<!DOCTYPE html><html><head></head><body><h1>";
10 var htmlSu
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.se                                tings "+name+htmlSuffix);
18 })
19
20 app.get("/goodbye", function(req, res){
21     res.                                  Goodbye to you too!" + htmlSuffix);
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

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

URL "route" set up

another route

all other routes

Express creates HTTP headers for us

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"));`

globally available string containing the name of the current module

a static files are contained in this directory

- Express always **first** checks the static files for a given route - if not found, the dynamic routes are checked

How to build a Web application

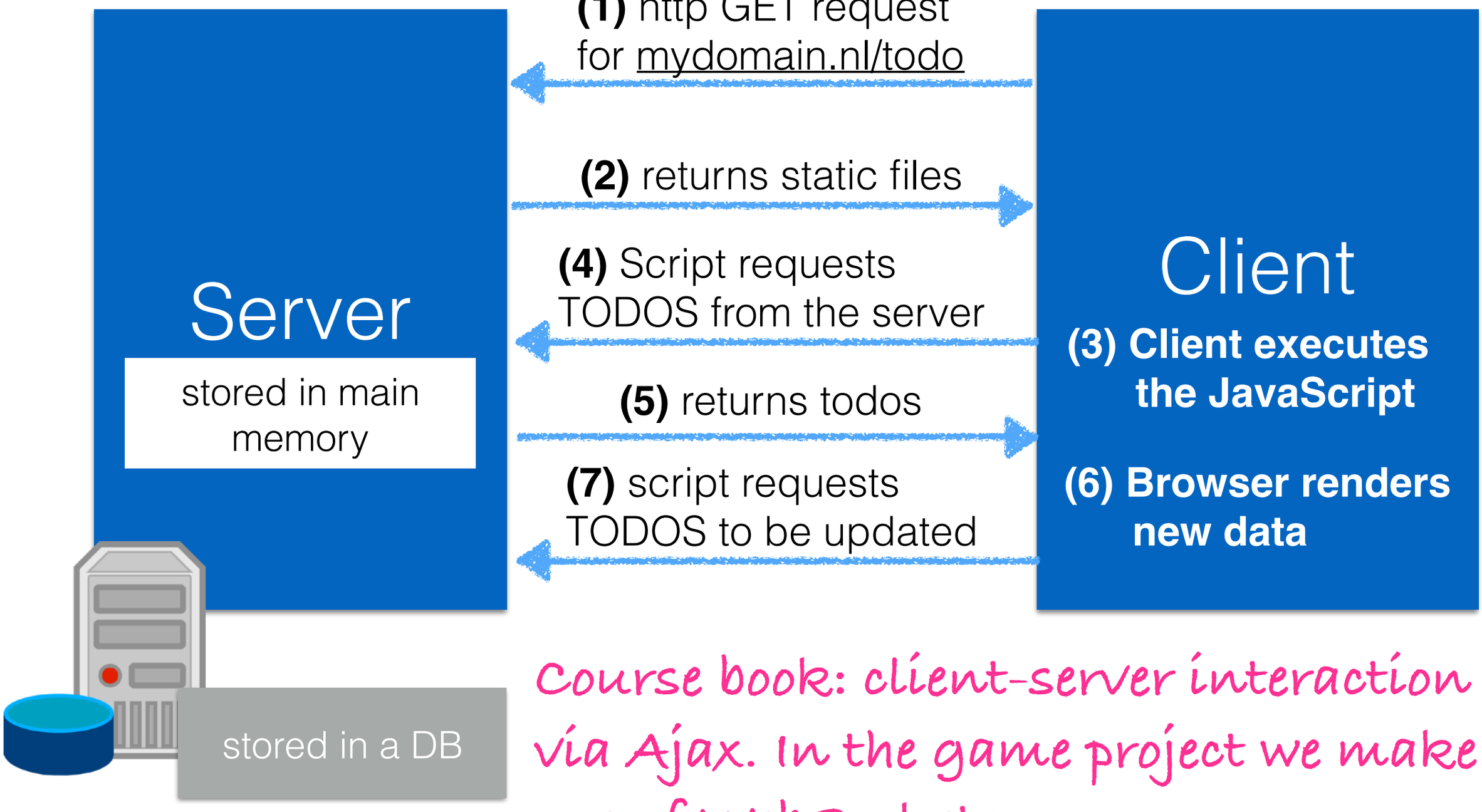
Development strategy

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)



Course book: client-server interaction via Ajax. In the game project we make use of WebSockets.

JSON: exchanging data between the client and server

XML vs. JSON

```xml
<!--?xml version="1.0"?-->
<timezone>
  <location></location>
  <offset>1</offset>
  <suffix>A</suffix>
  <localtime>20 Jan 2014 02:39:51</localtime>
  <isotime>2014-01-20 02:39:51 +0100</isotime>
  <utctime>2014-01-20 01:39:51</utctime>
  <dst>False</dst>
</timezone>
```

XML

```json
{
  "timezone": {
    "offset": "1",
    "suffix": "A",
    "localtime": "20 Jan 2014 02:39:51",
    "isotime": "2014-01-20 02:39:51 +0100",
    "utctime": "2014-01-20 01:39:51",
    "dst": "False"
  }
}
```

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 = [];
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 });
```

Array

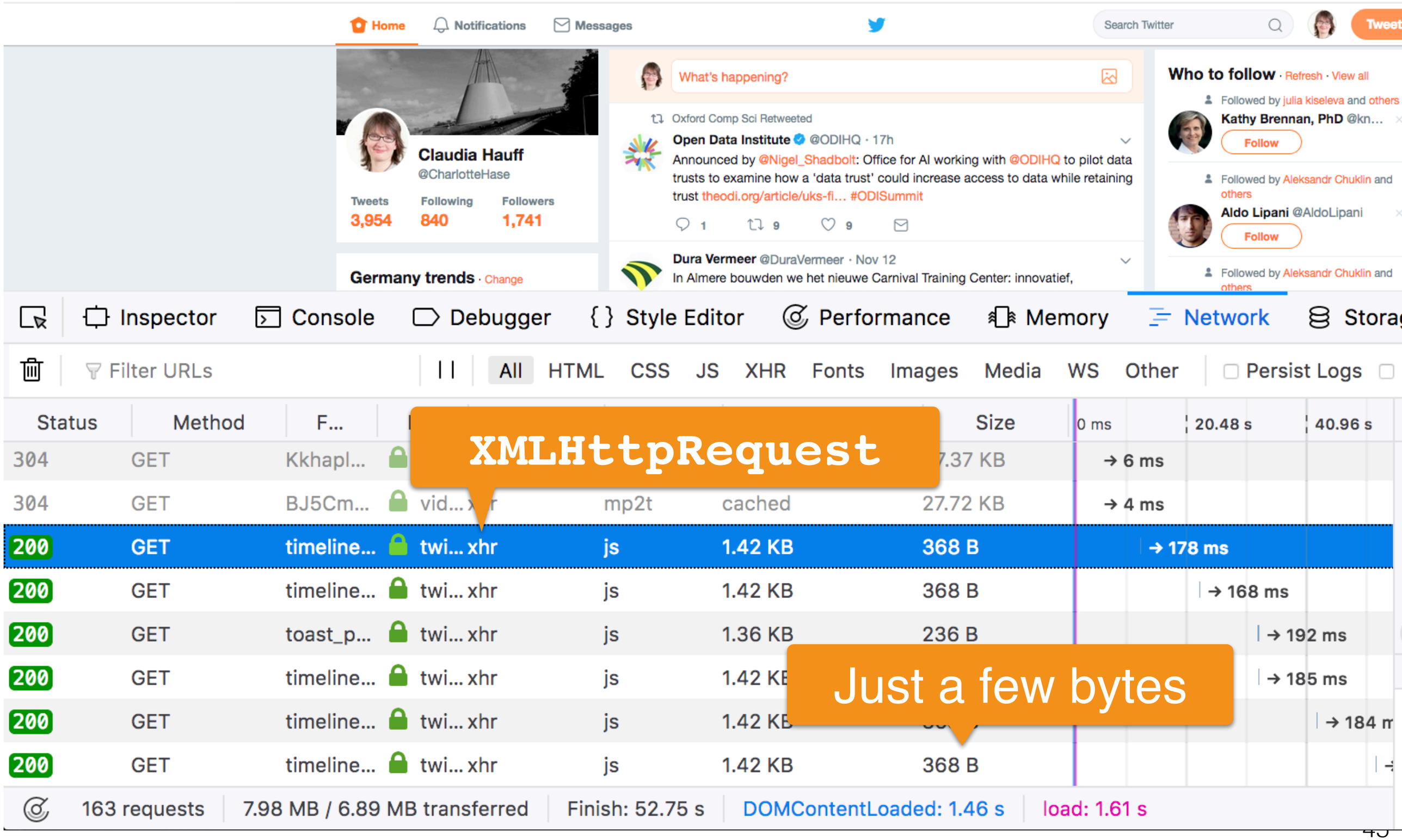
converted on the fly

Ajax: dynamic updating on the client

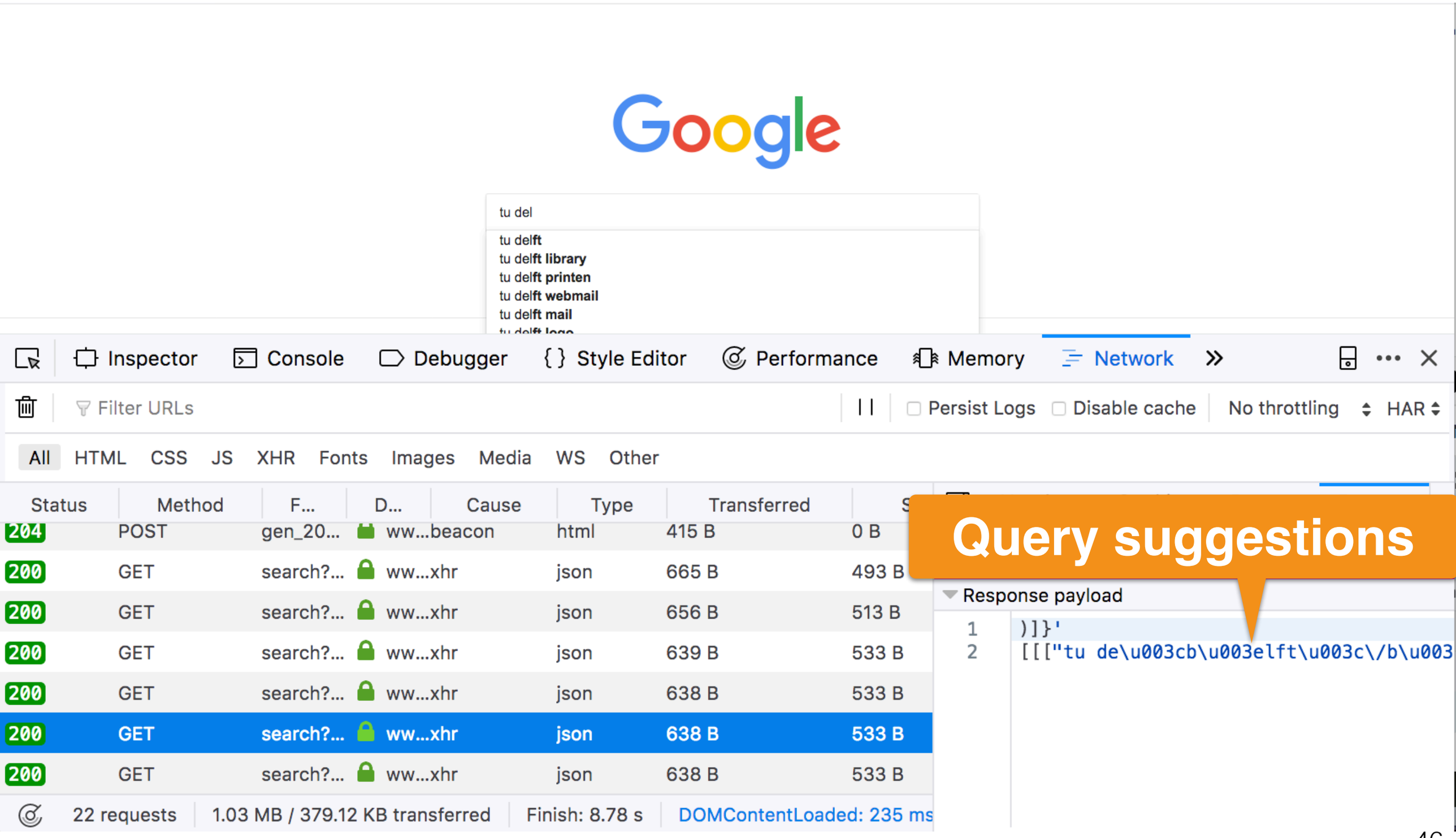
What is Ajax?

- 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?


Home
Notifications
Messages
Search Twitter
Tweet
Claudia Hauff
@CharlotteHase
Tweets 3,954
Following 840
Followers 1,741
Germany trends · Change
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/uks-fi… #ODISummit
1
9
9
Dura Vermeer @DuraVermeer · Nov 12
In Almere bouwden we het nieuwe Carnival Training Center: innovatief,
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
Inspector
Console
Debugger
Style Editor
Performance
Memory
Network
Filter URLs
All
HTML
CSS
JS
XHR
Fonts
Images
Media
WS
Other
Persist Logs
Status
Method
F…
Size
0 ms
20.48 s
40.96 s
304 GET Kkhapl… 7.37 KB → 6 ms
304 GET BJ5Cm… vid… mp2t cached 27.72 KB → 4 ms
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 → 185 ms
200 GET timeline… twi… xhr js 1.42 KB → 184 m
200 GET timeline… twi… xhr js 1.42 KB 368 B
XMLHttpRequest
Just a few bytes
163 requests
7.98 MB / 6.89 MB transferred
Finish: 52.75 s
DOMContentLoaded: 1.46 s
load: 1.61 s

What is Ajax?


Google
tu del
tu delft
tu delft library
tu delft printen
tu delft webmail
tu delft mail
Inspector
Console
Debugger
Style Editor
Performance
Memory
Network
Filter URLs
Persist Logs
Disable cache
No throttling
HAR
All
HTML
CSS
JS
XHR
Fonts
Images
Media
WS
Other
Status
Method
F...
D...
Cause
Type
Transferred
204 POST gen_20... ww...beacon html 415 B 0 B
200 GET search?... ww...xhr json 665 B 493 B
200 GET search?... ww...xhr json 656 B 513 B
200 GET search?... ww...xhr json 639 B 533 B
200 GET search?... ww...xhr json 638 B 533 B
200 GET search?... ww...xhr json 638 B 533 B
200 GET search?... ww...xhr json 638 B 533 B
22 requests
1.03 MB / 379.12 KB transferred
Finish: 8.78 s
DOMContentLoaded: 235 ms
Query suggestions
Response payload
1)]}'
2 [["tu de\u003cb\u003elft\u003c\/b\u003

On the client: HTML & JS

jQuery way

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

```
<!DOCTYPE html>
<head>
    <title>Plain text TODOs</title>
    <script src="http://code.jquery.com/jquery-3.3.1.min.js" type="text/javascript">
    </script>
    <script src="js/client-app.js" type="text/javascript"></script>
</head>

<body>
    <main>
        <section id="todo-section">
            <p>My list of TODOS:</p>
            <ul id="todo-list">
            </ul>
        </section>
    </main>
</body>
</html>
```

Define where the TODOs will be added. Empty list.

On the client: HTML & JS

jQuery way

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

```
<!DOCTYPE html>
<head>
    <title>Plain text TODOs</title>
    <script src="http://code.jquery.com/jquery-3.3.1.min.js" type="text/javascript">
    </script>
    <script src="js/client-app.js" type="text/javascript"></script>
</head>

<body>
    <main>
        <section id="todo-section">
            <p>My list of TODOS:</p>
            <ul id="todo-list">
            </ul>
        </section>
    </main>
</body>
</html>
```

Define where the TODOs will be added. Empty list.

Callback: define what happens when a todo object is available

```
var main = function () {
    "use strict";

    var addTodosToList = function (todos) {
        console.log("Loading todos from server");
        var todolist = document.getElementById("todo-list");
        for (var key in todos) {
            var li = document.createElement("li");
            li.innerHTML = "TODO: " + todos[key].message;
            todolist.appendChild(li);
        }
    };

    /*
     * This request retrieves the todo list once, to make this a regular
     * "event", make use of setInterval()
     */
    $.getJSON("../todos", addTodosToList)
        .done( function(){ console.log("Ajax request successful.");})
        .fail( function(){ console.log("Ajax request failed.");});
};
$(document).ready(main);
```

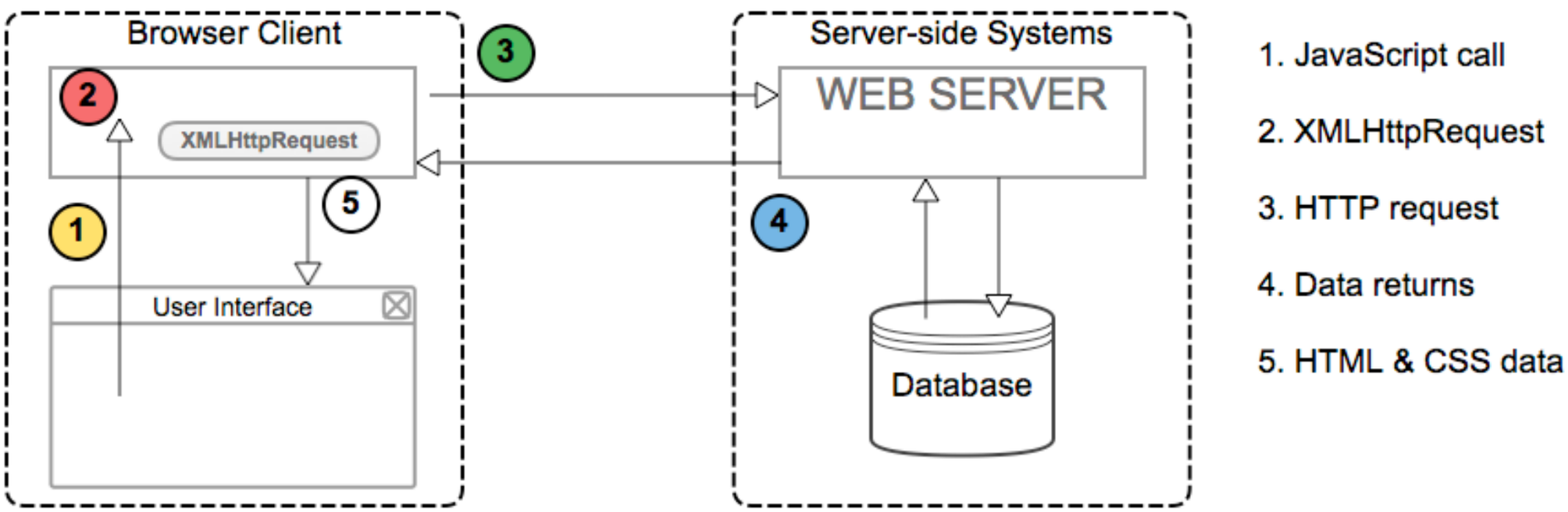
Dynamic insert of list elements into the DOM

Ajax

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

Ajax: how does it work?

1. Web browser creates a **`XMLHttpRequest`** object
2. **`XMLHttpRequest`** **requests data** from a Web server
3. **Data is sent back** from the server
4. On the client, **JavaScript code injects the data** into the page



WebSockets

WebSockets

Lecture 1

- **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

Hello World! example

```
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
                       the server ...";
20             };
21         </script>
22     </body>
23 </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 });
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: 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;

```
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);
```

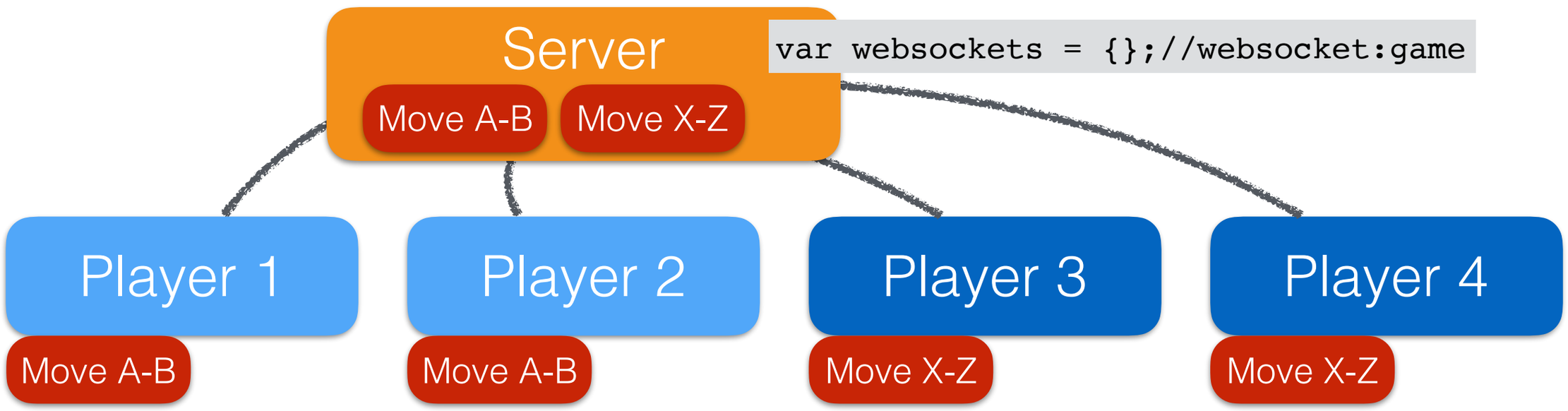
WebSocket server object

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:




```
var game = function (gameID) {
    this.playerA = null;
    this.playerB = null;
    this.id = gameID;
    this.wordToGuess = null;
    this.gameState = "0 JOINT";
};
```

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

```
var currentGame = new Game(gameStatus.gamesInitialized++);
var connectionID = 0;//each websocket receives a unique ID

wss.on("connection", function connection(ws) {

    /*
     * two-player game: every two players are added to the same game
     */
    let con = ws;
    con.id = connectionID++;
    let playerType = currentGame.addPlayer(con);
    websockets[con.id] = currentGame;

    /*
     * inform the client about its assigned player type
     */
    con.send((playerType == "A") ? messages.S_PLAYER_A : messages.S_PLAYER_B);

    ...
}
```

WebSockets for multi-player games

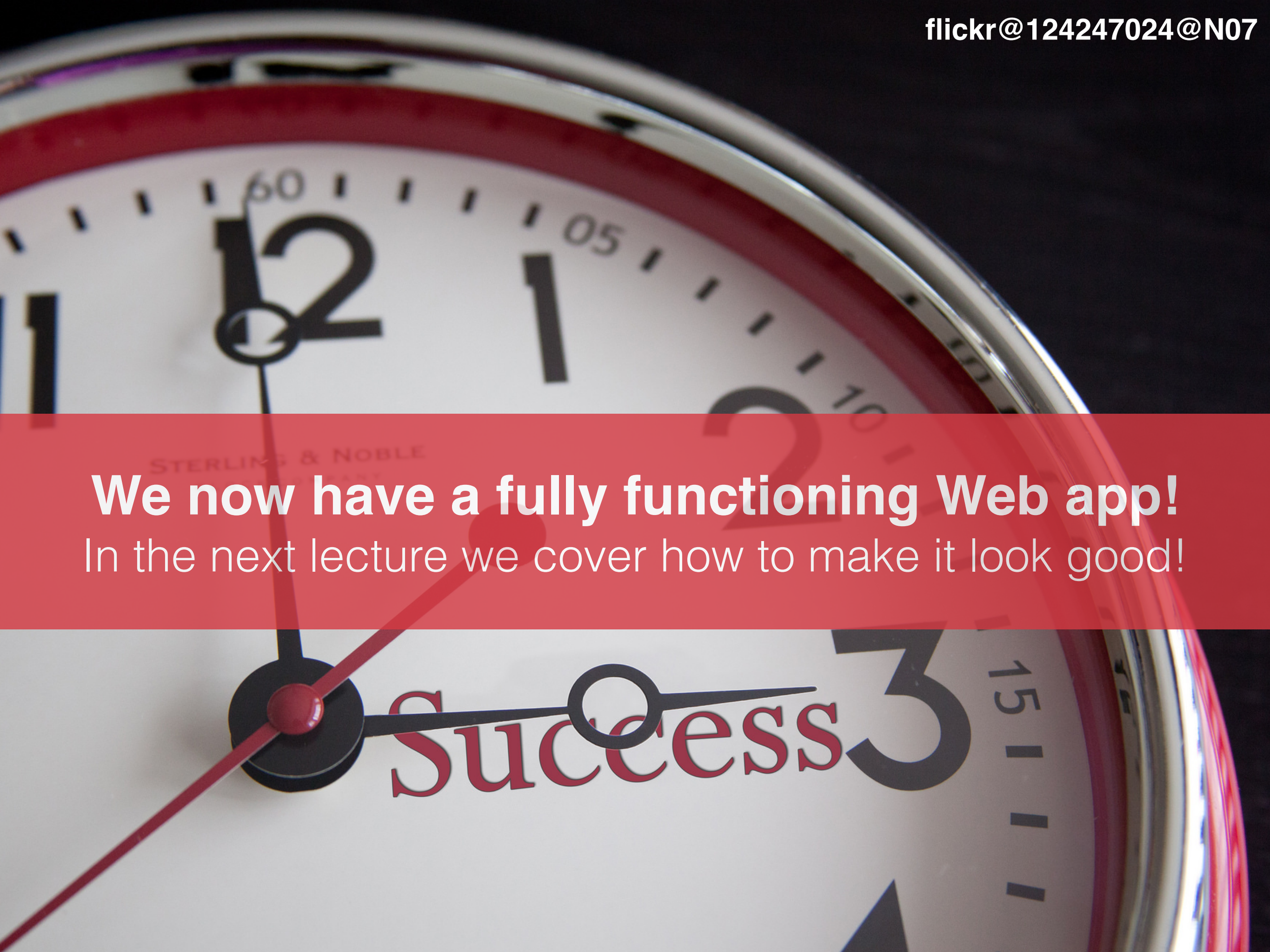
- Once a client establishes a WebSocket connection, the server-side script has several tasks:
  - Determine which game to add the player to
  - Inform the player about the game status
  - Request information from the player if necessary

```
var currentGame = new Game(gameStatus.gamesInitialized++);
var connectionID = 0;//each websocket receives a unique ID

wss.on("connection", function connection(ws) {

    /*
     * two-player game: every two pl
     */
    let con = ws;
    con.id = connectionID++;
    let playerType = currentGame.addPlayer(con);
    websockets[con.id] = currentGame;

    /*
     * inform the client about its assigned player type
     */
    con.send((playerType == "A") ? messages.S_PLAYER_A : messages.S_PLAYER_B);

    ...
}
```

unique identifier per WebSocket connection

add player to latest game

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).

flickr@courtneyrian