node.js: JavaScript on the server

Claudia Hauff
TI1506: Web and Database Technology
ti1506-ewi@tudelft.nl

TU Delft

At the end of this lecture, you should be able to …

- **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 communication via `JSON`
- **Implement** client-side code using Ajax

A reminder before we start

Web servers and clients

Lecture 1

(1) HTTP request

(2) HTTP response

- display
- execute
- music player
- Acrobat Reader

- Wait for data requests
- Answer thousands of clients simultaneously
- Host **web resources**

- Clients are most often Web browsers
- **Telnet**

Web resource: any kind of content with an identity, including static files (e.g. text, images, video), software programs, Web cam gateway, etc.

HTTP response message

```
HTTP/1.1 200 OK
Date: Fri, 01 Aug 2014 13:35:55 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 5994
Connection: keep-alive
Set-Cookie: fe_typo_user=d5e20a55a4a92e0;
path=/; domain=tudelft.nl
[....]
Server: TU Delft Web Server

....
....
```

start line

header fields

name:value

body (optional)

```
1  var x = six();
2
3  //function declaration
4  function six(){
5    return 6;
6  }
7
8  var y = seven()
9
10 //function expression
11 var seven = function(){
12   return 7;
13 }
14
15 console.log(x+""+y);
```

Declarations are processed before any code is executed (the hoisting principle).

Declarations are hoisted to the top.

Expressions are not hoisted.

What is node.js?

node.js in its own words …

"This document is 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.

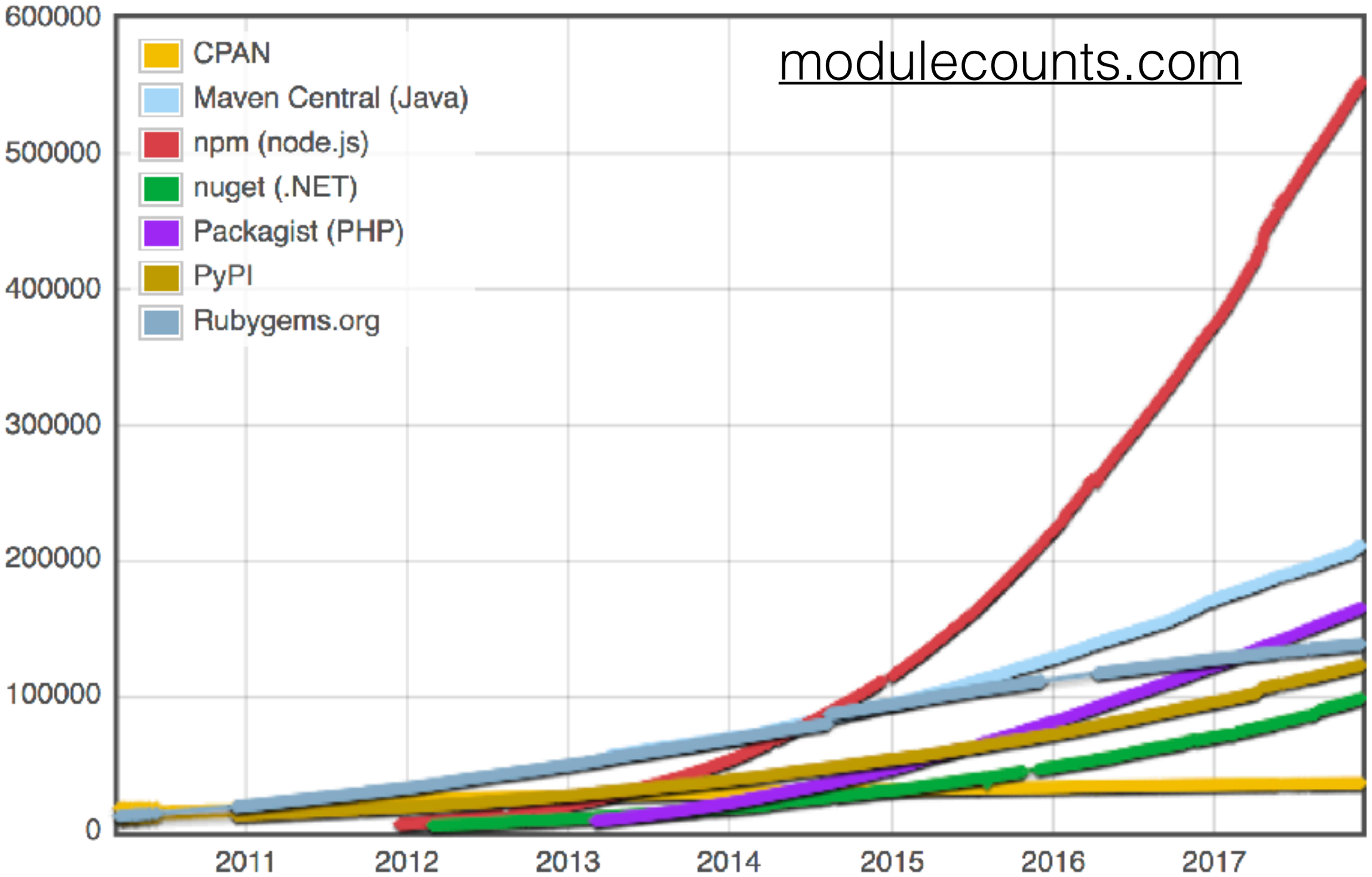
Node.js uses an **event-driven**, **non-blocking I/O** model that makes it lightweight and efficient, perfect for **data-intensive real-time applications** that run across distributed devices."

https://nodejs.org

History of node.js

- Google's JavaScript execution engine (**V8**) was open-sourced in 2008
- node.js builds on V8 and was released in **2009**
- **node.js' package manager** (`npm`) was released in 2011
- December 2014: **io.js**
- May 2015: **io.js merges back** with node.js
- Node.js Foundation steers development
- **2017**: node becomes a **first-class citizen of V8** (no V8 commit allowed to break node)

History of node.js



Packages do not have to do a lot

npm

find packages

Easy sharing. Manage teams and permissions with one click. Create a free org »

★ smallest public

Find the smallest number in a list

Install

```
$ npm install --save smallest
```

Stats

41 downloads in the last day

354 downloads in the last week

1,241 downloads in the last month

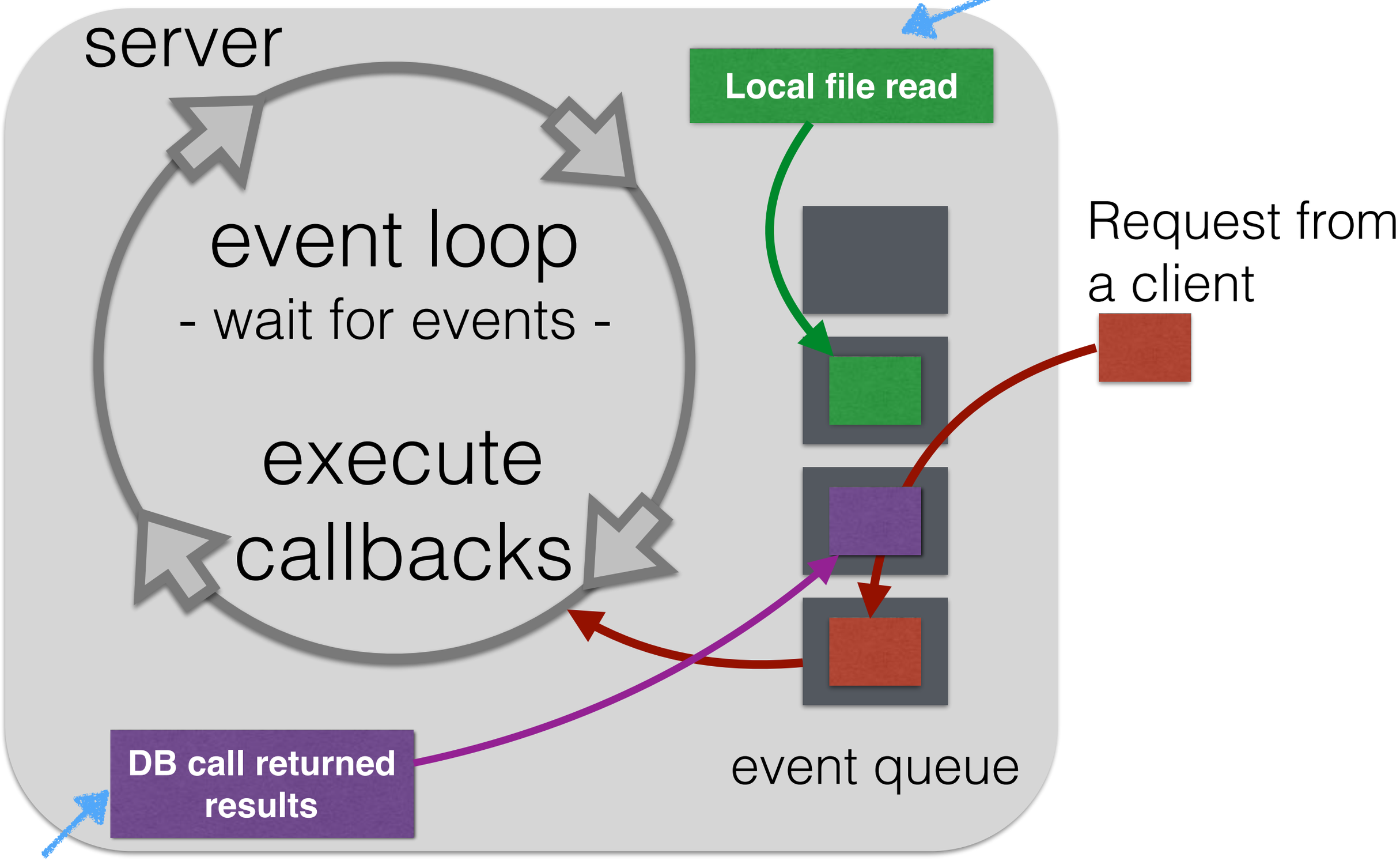
No open issues on GitHub

No open pull requests on GitHub

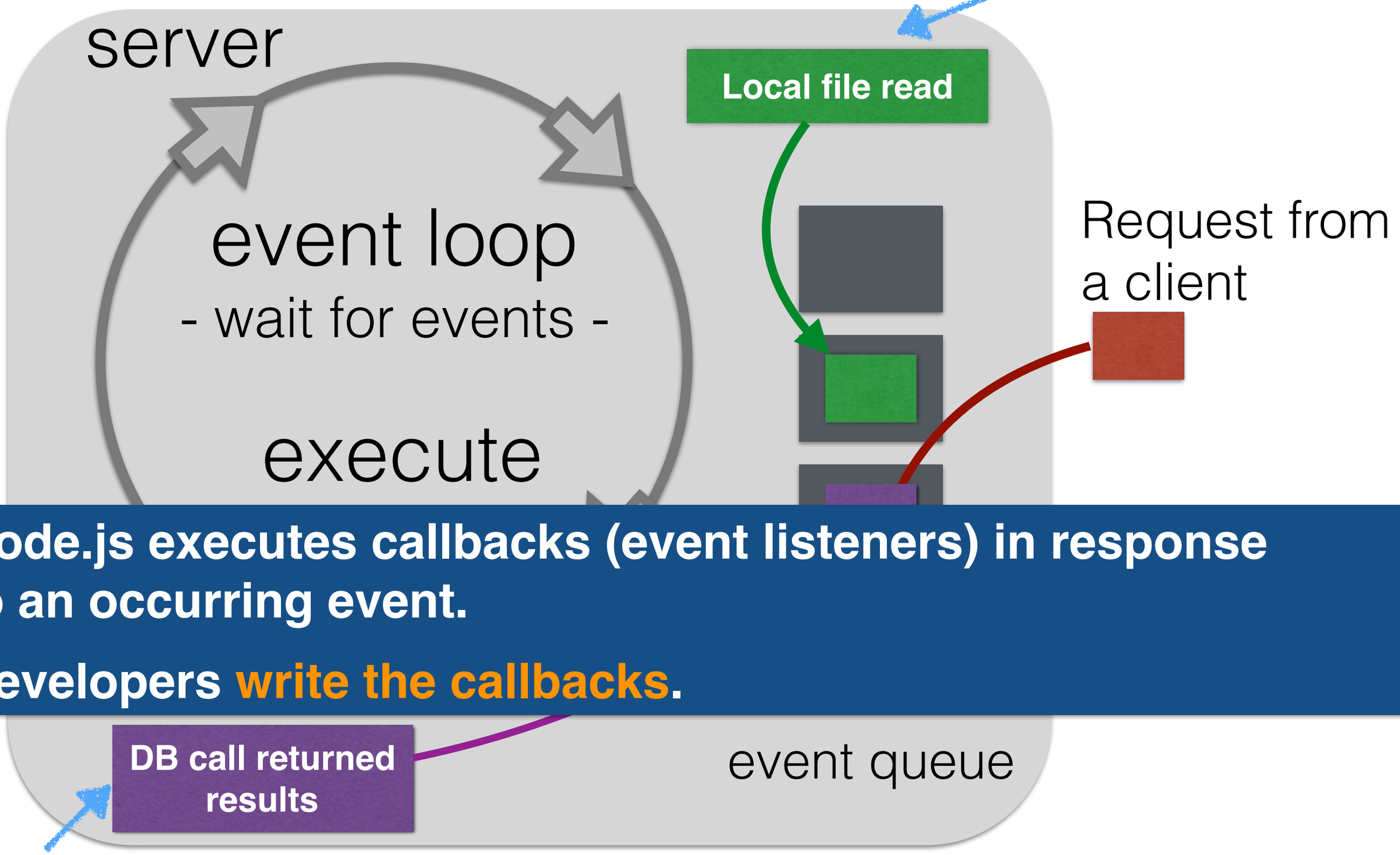
5 lines (4 sloc) | 135 Bytes

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

node.js is event-driven

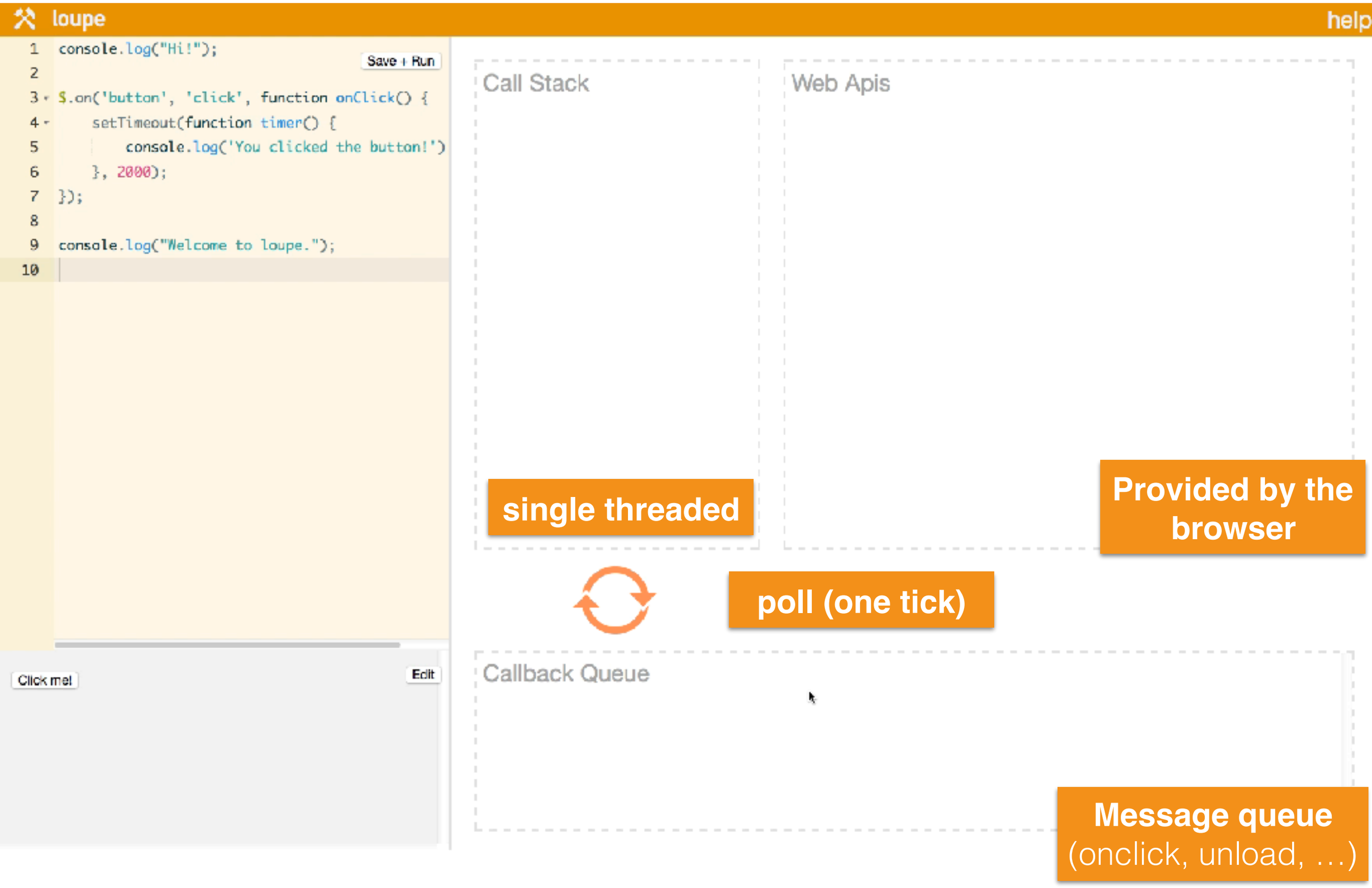


node.js is event-driven



It is actually a bit more complicated than that …

http://latentflip.com/loupe/



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: single-threaded but highly parallel

Blocking I/O (database example)

(1) read request

(2) process request & access the database

(3) **wait** for the database to return data and process it

(4) process the next request

Non-blocking I/O

(1) read request

(2) process request and make a **callback** to access the database

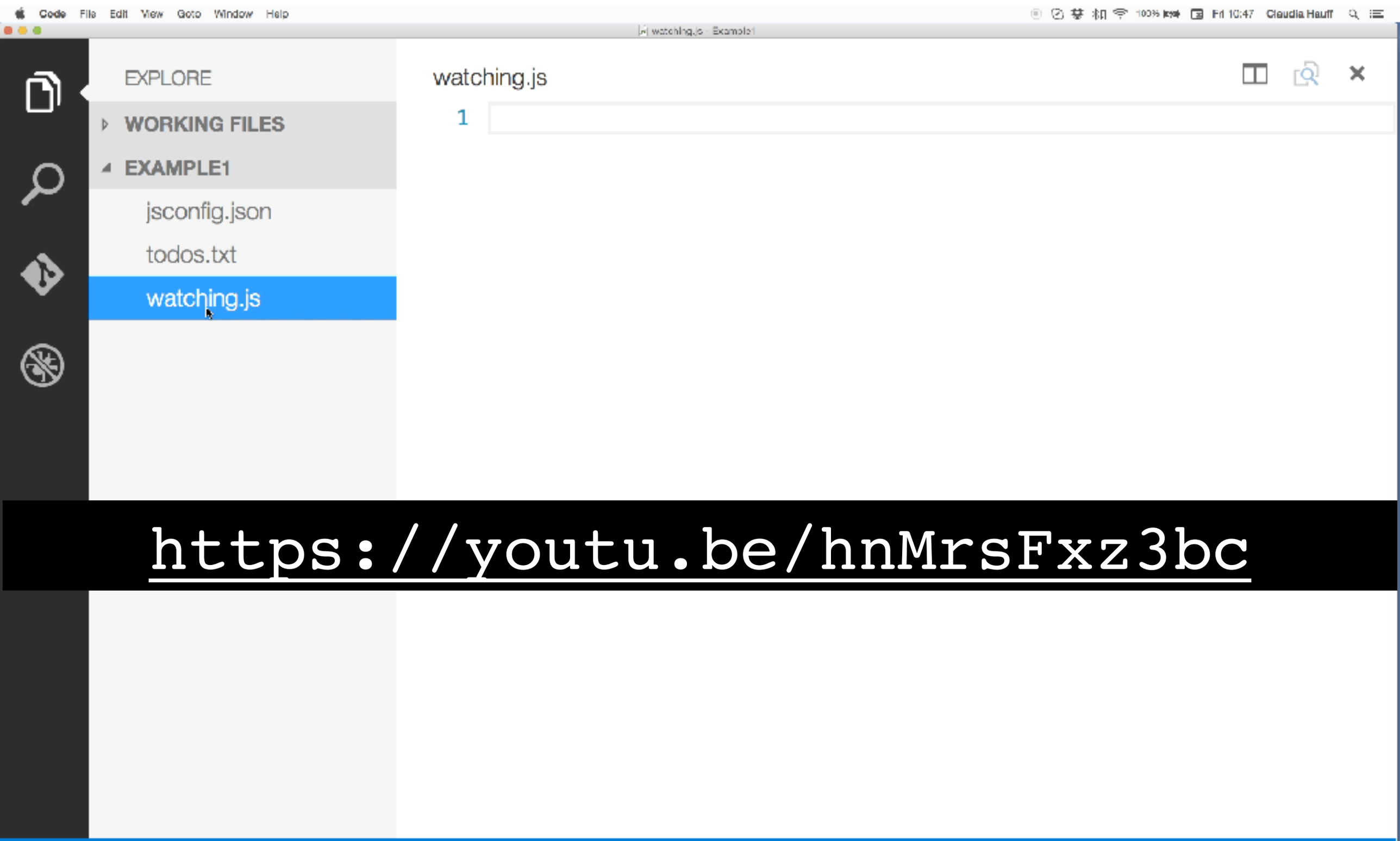
(3) do other things

(4) when the callback returns, process it

The first code examples

Example 1

Visual Studio Code (open-source, for all platforms)



https://youtu.be/hnMrsFxz3bc

Example: watch a file for changes

read-only reference to a value; **`var`** can be used too

`require()` **usually** returns a JavaScript **object**

node.js fs **module***

Self-contained piece of code that provides reusable functionality.

```
1 const fs = require('fs');
2 fs.watch('todos.txt', function() {
3     console.log("File 'todos.txt' has \
4                  just changed");
5 });
6 console.log("Now watching 'todos.txt'");
```

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

* **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)

Assumption: file to watch exists!

Example: watch a file for changes

read-only reference to a value; **`var`** can be used too

`require()` **usually** returns a JavaScript **object**

node.js fs **module***

Self-contained piece of code that provides reusable functionality.

callback: defines what should happen when the file changes

```
1 const fs = require('fs');
2 fs.watch('todos.txt', function() {
3     console.log("File 'todos.txt'
4                  just changed");
5 });
6 console.log("Now watching 'todos.txt'");
```

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

anonymous function

asynchronous

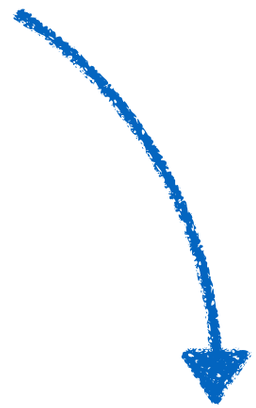
Executed immediately after the **setup** of the callback

* **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)

Assumption: file to watch exists!

Networking with node.js

- Built specifically for **networked programming** (not just Web programming!)
- node.js has built-in support for **low-level** socket connections (TCP sockets)
- TCP socket connections have **two endpoints**
  1. **binds** to a numbered port
  2. **connects** to a port



defined by IP address and port number

Networking with node.js

- Built specifically for **networked programming** (not just Web programming!)
- node.js has built-in support for **low-level** socket connections (TCP sockets)
- TCP socket connections have **two endpoints**
  1. **binds** to a numbered port
  2. **connects** to a port

defined by IP address and port number

Analogous example: phone lines.

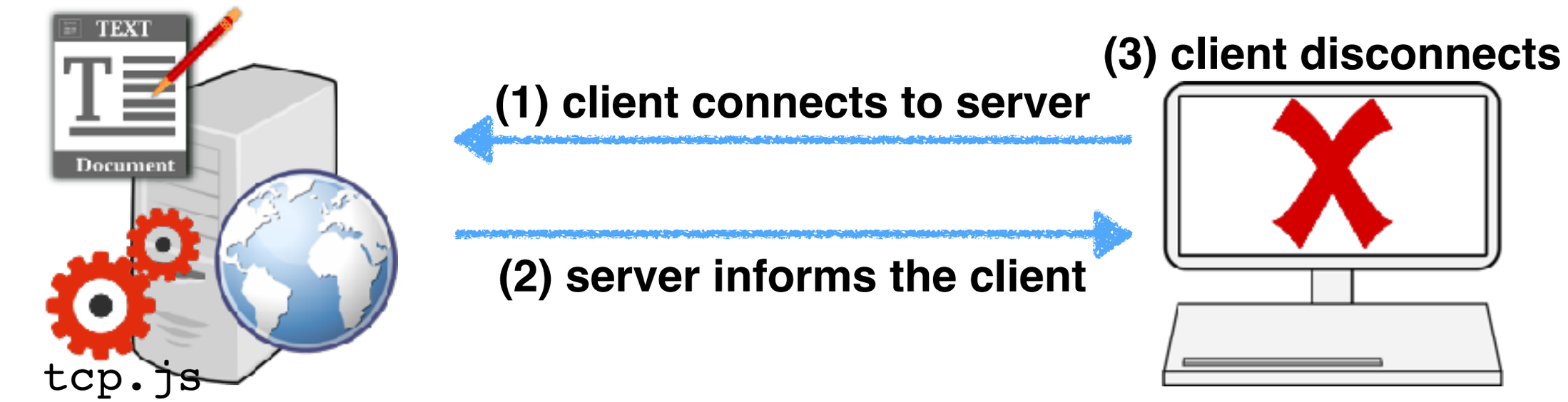
One phone binds to a phone number.

Another phone tries to call that phone.

If the call is answered, a connection is established.

Example: low-level networking with node.js

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

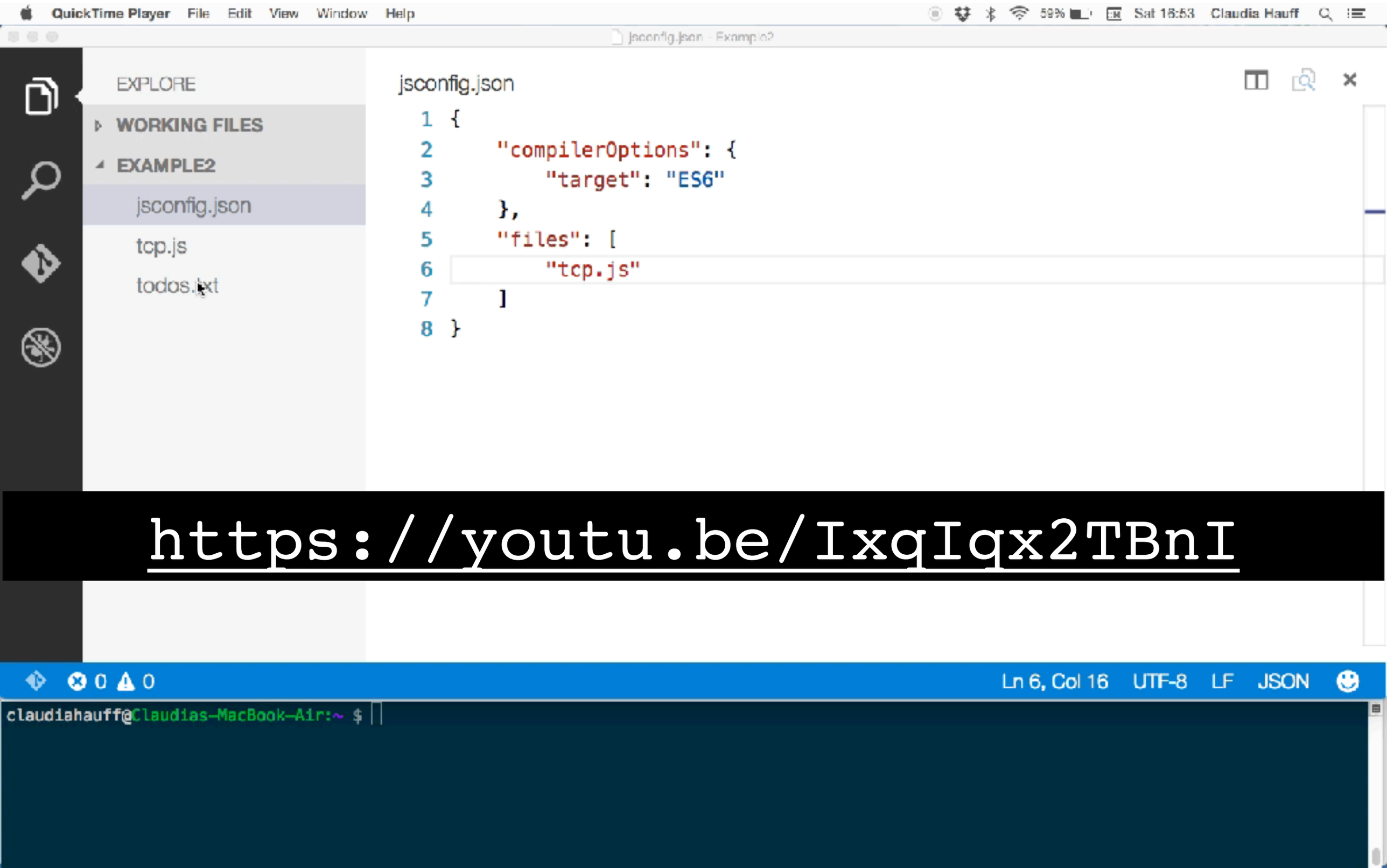



```
server:~ $
Listening for subscribers …
Subscriber connected.
Subscriber disconnected.
```

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

Example 2

Example: low-level networking with node.js



https://youtu.be/IxqIqx2TBnI

Example: low-level networking with node.js

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

```
1 "use strict";
2 const
3   net = require('net'),
4   server = net.createServer(function(connection)
5   {
6     // use connection object for data transfer
7   });
8
9 server.listen(5432);
```

server **object** is returned

callback function is invoked when another endpoint connects

bind to port 5432

Example: low-level networking with node.js

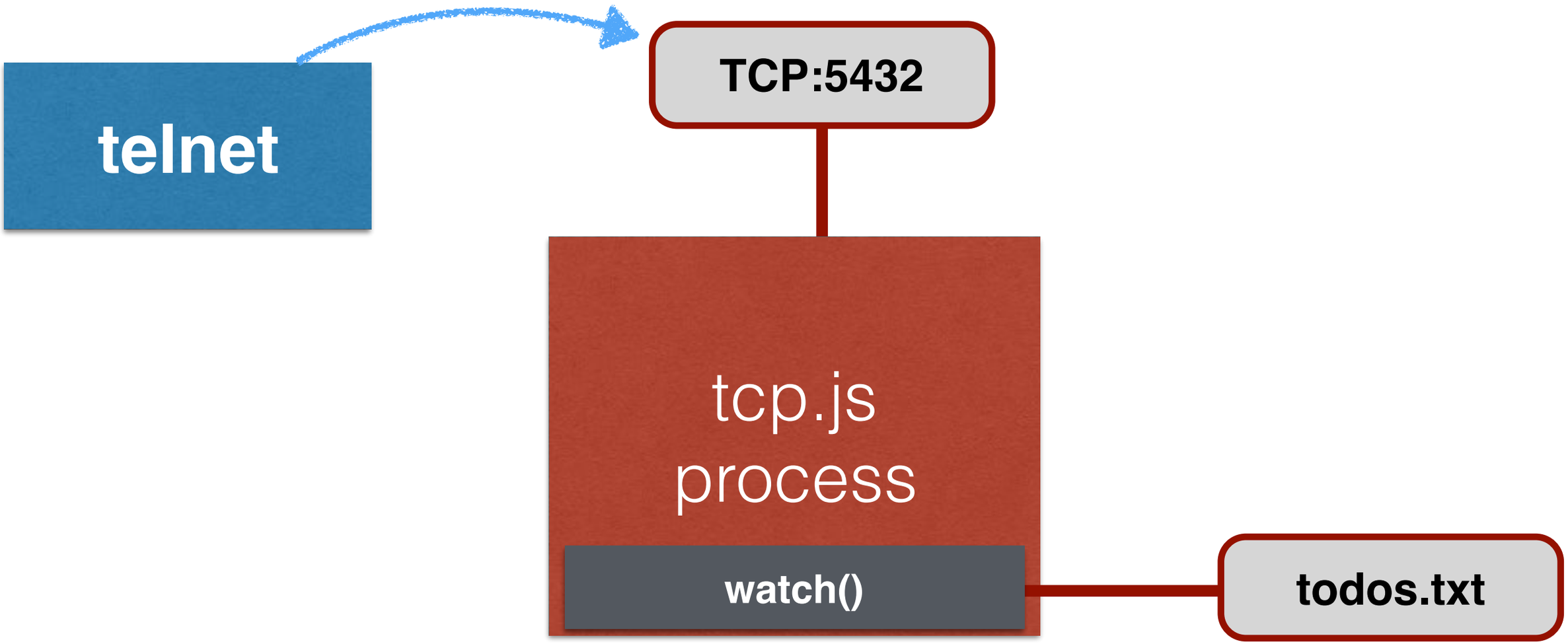
Start the **server** on the command line: **`$ node --harmony tcp.js`**
Client terminal: `$ telnet localhost 5432`

```
5    filename = "todos.txt",
6    server = net.createServer(function(connection)
7    {
8        console.log('Subscriber connected.');
9        connection.write("Now watching “ + filename +
10                            “ for changes...\n");
11       // watcher setup
12       var watcher = fs.watch(filename, function() {
13           connection.write("File '" + filename + "'
14                       changed: " + Date.now() + "\n");
15       });
16       // cleanup
17       connection.on('close', function() {
18           console.log('Subscriber disconnected.');
19           watcher.close();
20       });
21   });
22 server.listen(5432, function() {
23   console.log('Listening for subscribers...');
24 });
```

client-side output

server-side output

Low-level networking with node.js



Using node.js to create a Web server

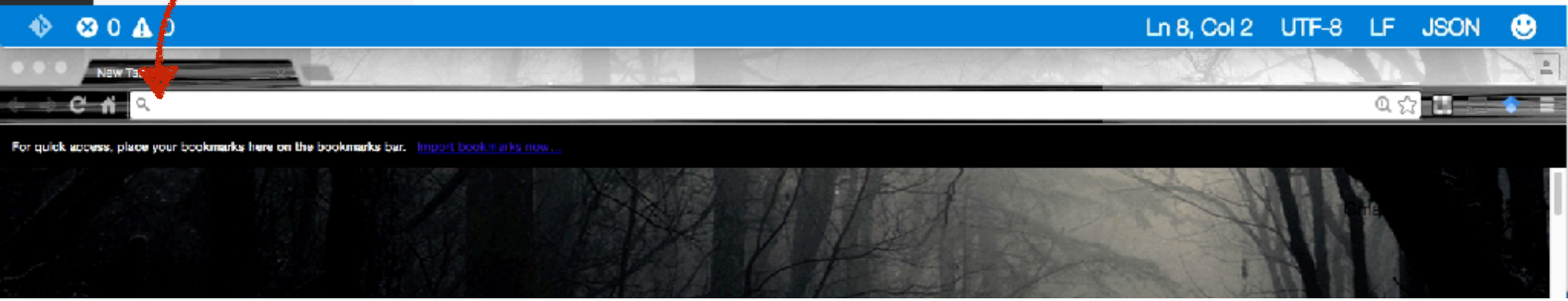
node.js is **not** a Web server. It provides **functionality** to implement one!

Example 3

“Hello World” in the browser

https://youtu.be/tPFQChnnGXg

`localhost:3000/Hallihallo` or any other path also works



The “Hello World” of node.js

node.js http module

Create a **Web server**

A callback: what to do if a request comes in

```
1  var http = require(“http");
2  var server;
3
4  server = http.createServer(function (req, res) {
5    res.writeHead(200, {"Content-Type":
6                                  "text/plain"});
7    res.end("Hello World!”);
8    console.log(“HTTP response sent”);
9  });
10
11 server.listen(3000);
12 console.log("Server listening on port 3000");
```

Create a **HTTP response** & send it

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

The “Hello World” of node.js

```
1  var http = require(“http");
2  var server;
3
4  var sentCounter = 0;
5
6  server = http.createServer(function (req, res) {
7      res.writeHead(200, {"Content-Type":
8                     "text/plain"});
9      res.end("Hello World!");
10     sentCounter++;
11     console.log(sentCounter+" HTTP responses sent
12                              in total");
13 });
14
15 var port = 2345;
16 server.listen(port);
17 console.log("Server listening on port “ + port);
```

This is standard JavaScript. We can add variables, functions, objects…

HTTP **response** object

HTTP **request** object

Are we sending an object? Yes and no (JSON)

Any port number between 1024 and 49151 is fine.

A little refactoring …

```
1  var http = require("http"),
2  var server;
3
4  var simpleHTTPResponder = function (req, res) {
5   res.writeHead(200, {"Content-Type":
6                       "text/plain"});
7   sentCounter++;
8   res.end("'Hello World' for the "+sentCounter+".
9           time!");
10   console.log(sentCounter+" HTTP responses sent
11               in total");
12 }
13
14 var sentCounter = 0;
15
16 server = http.createServer(simpleHTTPResponder);
17
18 var port = process.argv[2];
19 server.listen(port);
20 console.log("Server listening on port "+port);
```

function as parameter

command line parameter

Using URLs for routing

```
1  var http = require("http");
2  var url = require('url');
3  var server;
4
5  var simpleHTTPResponder = function (req, res) {
6    var url_parts = url.parse(req.url, true);
7    if(url_parts.pathname == "/greetme") {
8       res.writeHead(200, {"Content-Type":
9                     "text/plain"});
10      var query = url_parts.query;
11      if( query["name"]!=undefined) {
12         res.end("Greetings "+query["name"]);
13      }
14      else { res.end("Greetings Anonymous"); }
15   }
16   else {
17      res.writeHead(404,{"Content-Type":
18                    "text/plain"});
19      res.end("Only /greetme is implemented.");
20   }
21 }
22
23 server = http.createServer(simpleHTTPResponder);
24 var port = process.argv[2];
25 server.listen(port);
```

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

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

otherwise send back a **404 error**

Example 4

URL routing

https://youtu.be/S3hslG7am-4

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** (like http)
- Express is not one of them (but we have **NPM**)

```
$ npm install express
```

a collection of code with a public interface

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)**

The “Hello World” of Express

```
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 app.get("/greetme", function (req, res) {
11  var query = url.parse(req.url, true).query;
12  var name = ( query["name"]!=undefined) ?
13                                  query[“name"] : "Anonymous";
14  res.send("Greetings "+name);
15 });
16
17 app.get("/goodbye", function (req, res) {
18  res.send("Goodbye you!");
19});
```

Express creates HTTP headers for us

The “Hello World” of Express

```
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.create
9
10 app.get("/greetme", function (req, res) {
11  var query = url.parse(req.url, true).query;
12  var name = ( query["name"]!=undefined) ?
13                                  query[“name"] : "Anonymous";
14  res.send("Greetings "+name);
15 });
16
17 app.get("/goodbye", function (req, res) {
18  res.send("Goodbye you!");
19});
```

app object is our way to use Express’ abilities

URL “route” set up

another route

Express creates HTTP headers for us

Express and HTML …

```
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 app.get("/greetme", function (req, res) {
11  var query = url.parse(req.url, true).query;
12  var name = ( query["name"]!=undefined) ? query[
13                "name"] : "Anonymous";
14  res.send("<html><head></head><body><h1>
15            Greetings "+name+"</h1></body></html>
16            ");
17 });
18
19 app.get("/goodbye", function (req, res) {
20  res.send("Goodbye you!");
21 });
```

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

a static files are contained in this directory

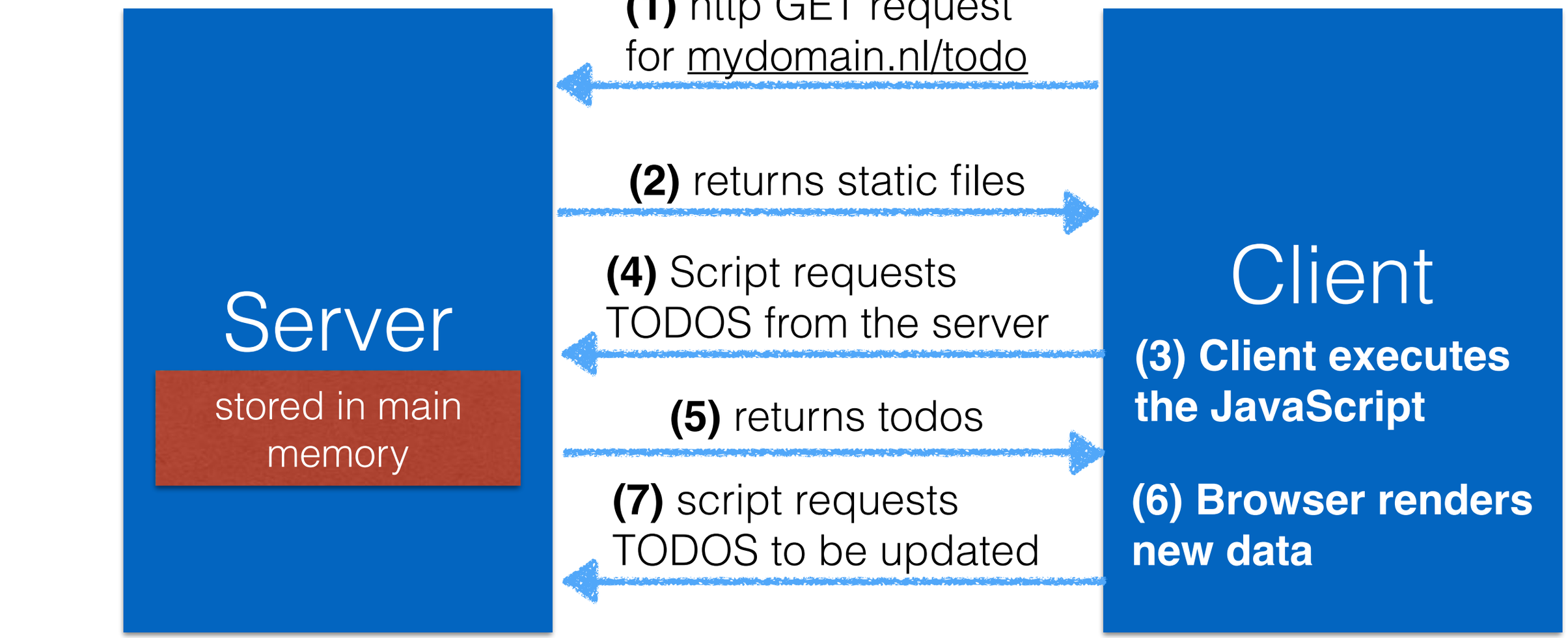
How to build a Web application

Development strategy

- Develop the **client-side code** (HTML, CSS, JavaScript)
- Place all files into some directory (e.g. `/client`) **on the server**
- Define the **node.js server code** in one or more *.js files using Express
- Set **Express' static file path** to the directory of step 2
- Add interactivity between client and server via **Ajax** and **JSON**

```
todo-server.js
client/
    index.html
    html/
        =>error.html
        =>addtodo.html
images/
    =>background.png
    =>logo.png
css/
    =>layout.css
    =>style.css
javascript/
    =>client-app.js
```

TODO Web app flow



JSON: exchanging data between the client and server

Exchanging data: JSON

JavaScript Object Notation

- In early (earlier) years, **XML** was used as data exchange format - well defined but not easy to handle
- Developed by **Douglas Crockford**
- XML is often too **bulky** in practice; JSON is much **smaller** than XML
- JSON can be fully parsed using **built-in JavaScript** commands
- JavaScript objects can be turned into JSON with a call

JSON vs. XML

```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

JSON vs. JavaScript Objects

- JSON: all object property names must be enclosed in quotes

- JSON objects **do not have functions** as properties

- Any JavaScript object can be transformed into JSON via `JSON.stringify`

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",
12             type : 1, deadline : "12/12/2014"};
13 var t2 = { message : "English homework due",
14             type : 3, deadline : "20/12/2014"};
15 todos.push(t1);
16 todos.push(t2);
17
18 app.get("/todos", function (req, res) {
19  res.json(todos);
20 });
```

in-memory todos

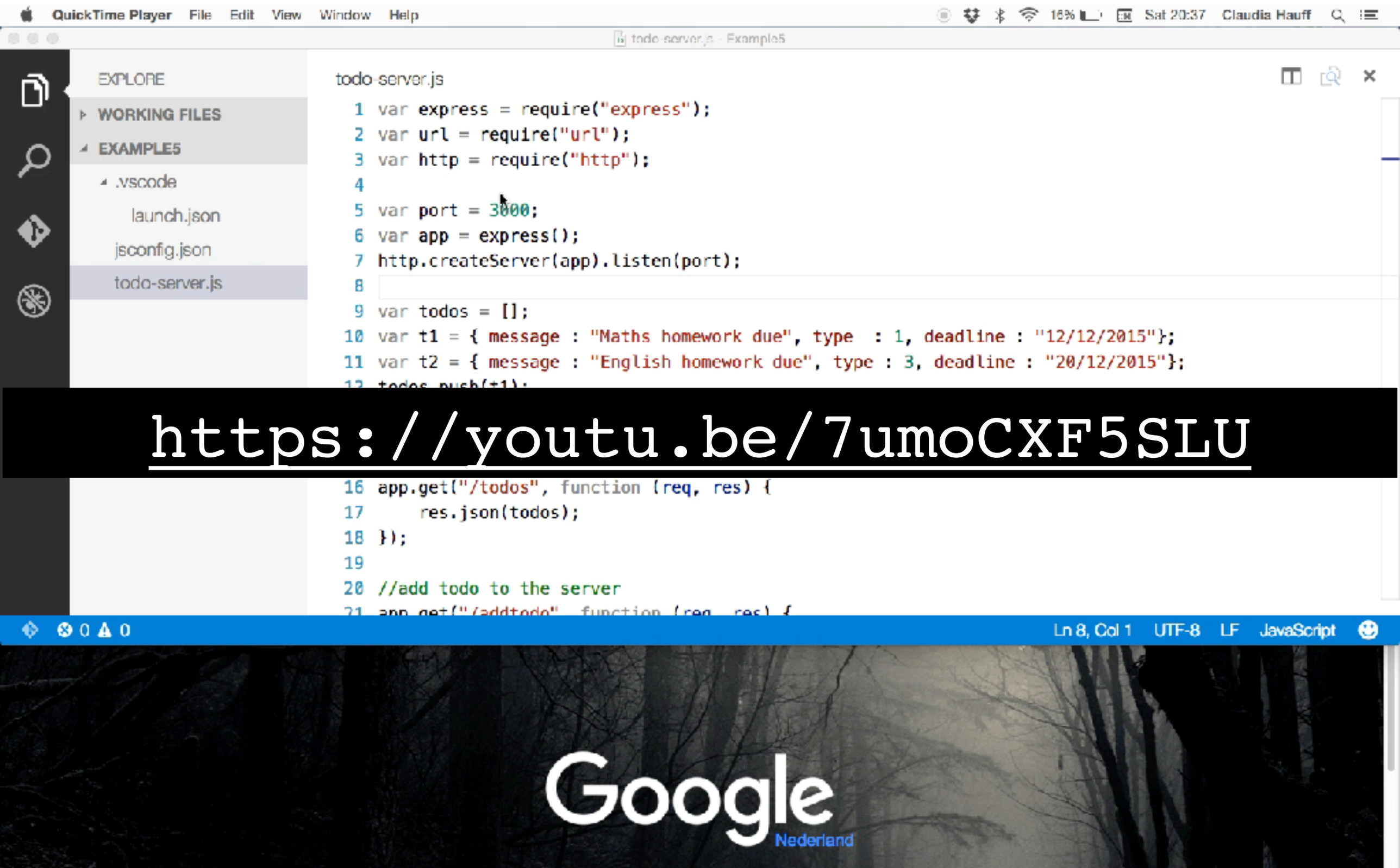
Send **JSON** to **client**

On the server: todo updating

```
1  app.get("/addtodo", function (req, res) {
2   var url_parts = url.parse(req.url, true);
3   var query = url_parts.query;
4   if(query["message"]!==undefined) {
5      var tx = { message: query["message"],
6                 type: query["type"],
7                 deadline : query[“deadline"]};
8      todos.push(tx);
9      res.end("Todo added successfully");
10     console.log("Added "+tx.message);
11  }
12  else
13  {
14     res.end("Error: missing message parameter");
15  }
16 });
```

Example 5

Sending json & todo updating



https://youtu.be/7umoCXF5SLU

Ajax: dynamic updating on the client

Ajax

XML only in the name

Asynchronous JavaScript and XML

- Ajax is a **JavaScript mechanism** that enables the dynamic loading of content without having to refetch/ reload the page manually
- Ajax: technology to **inject** new data into an existing web page (not a language or a product)
- You see this technology every day: chats, endless scrolling

Ajax

XML only in the name

Asynchronous JavaScript and XML

- Ajax is a **JavaScript mechanism** that enables the dynamic loading of content without having to refetch/ reload the page manually
- Ajax: technology to **inject** new data into an existing web page (not a language or a product)
- You see this technology every day: chats, endless scrolling
- Ajax revolves around **`XMLHttpRequest`**, a JavaScript object
- **jQuery hides all complexity, makes Ajax calls easy**

On the client: basic HTML

jQuery way

```
1  <!doctype html>
2  <html>
3      <head>
4          <title>Plain text TODOs</title>
5          <script src="http://code.jquery.
6                       com/jquery-1.11.1.js"
7                       type="text/javascript"></script>
9          <script src="javascript/client-app.js"
10                      type="text/javascript"></script>
12     </head>
13     <body>
14       <main>
15         <section id="todo-section">
16           <p>My list of TODOS:</p>
17           <ul id="todo-list">
18           </ul>
19         </section>
20       </main>
21     </body>
22 </html>
```

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

Define where the TODOs will be added.

On the client: JavaScript

jQuery way

Callback: define what happens when a todo object is available

```
1  var main = function () {
2   "use strict";
3
4   var addTodosToList = function (todos) {
5      var todolist = document.getElementById("todo-list");
6
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  $.getJSON("todos", addTodosToList);
15 }
16 $(document).ready(main);
```

this is Ajax

when the document is loaded, execute main()

On the client: JavaScript

jQuery way

Callback: define what happens when a todo object is available

Dynamic insert of list elements into the DOM

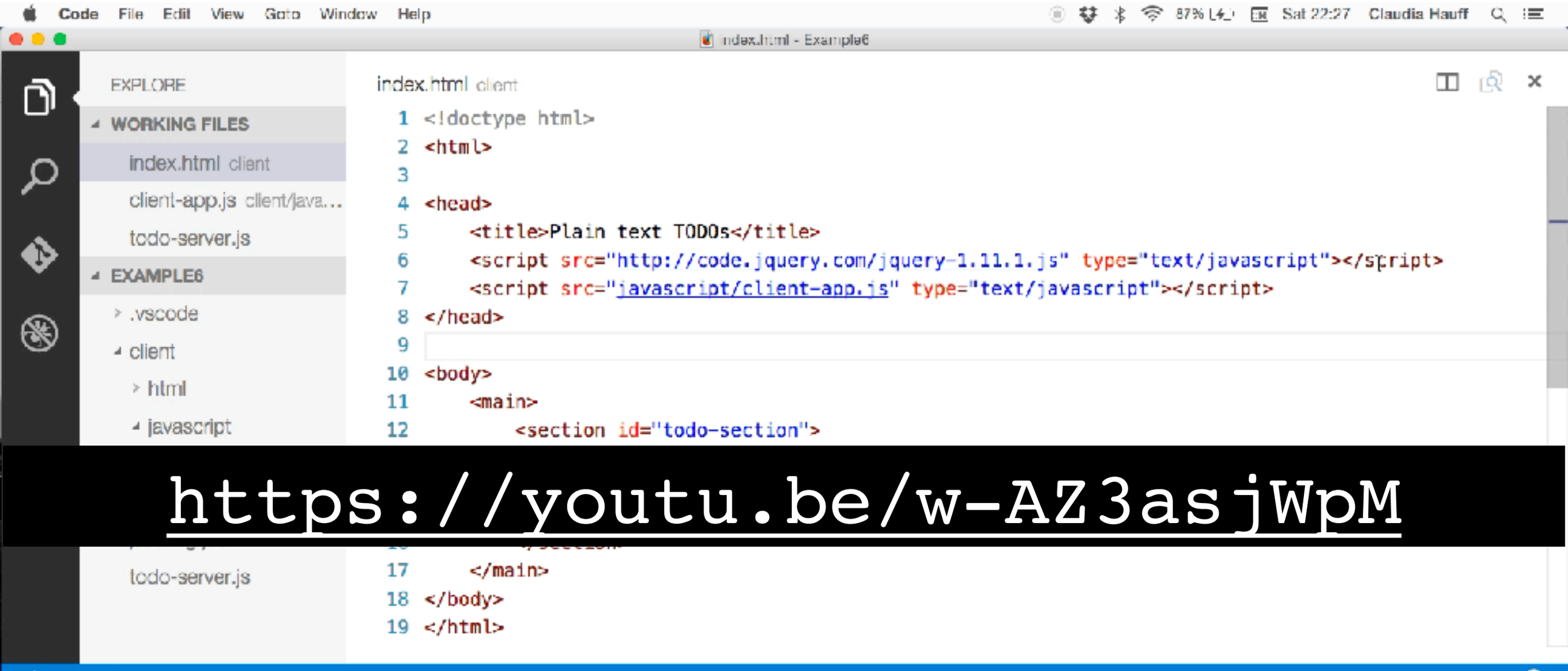
```
1  var main = function () {
2   "use strict";
3
4                          oList = function (todos) {
5                          t = document.getElementById("todo-list");
6
7       for (var key in todos) {
8           var li = document.createElement("li");
9           li.innerHTML = "TODO: “+todos[key].message;
10          todolist.appendChild(li);
11      }
12  };
13
14  $.getJSON("todos", addTodosToList);
15 }
16 $(document).ready(main);
```

this is Ajax

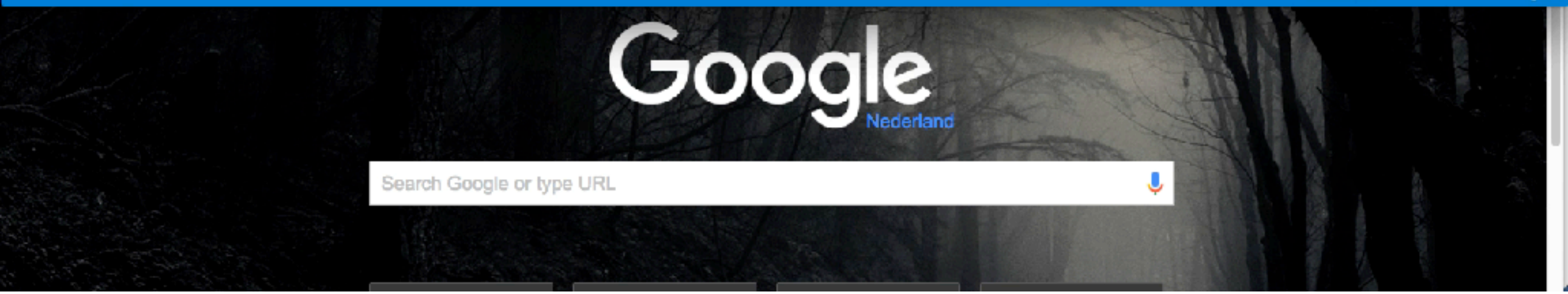
when the document is loaded, execute main()

Example 6

Ajax



https://youtu.be/w-AZ3asjWpM



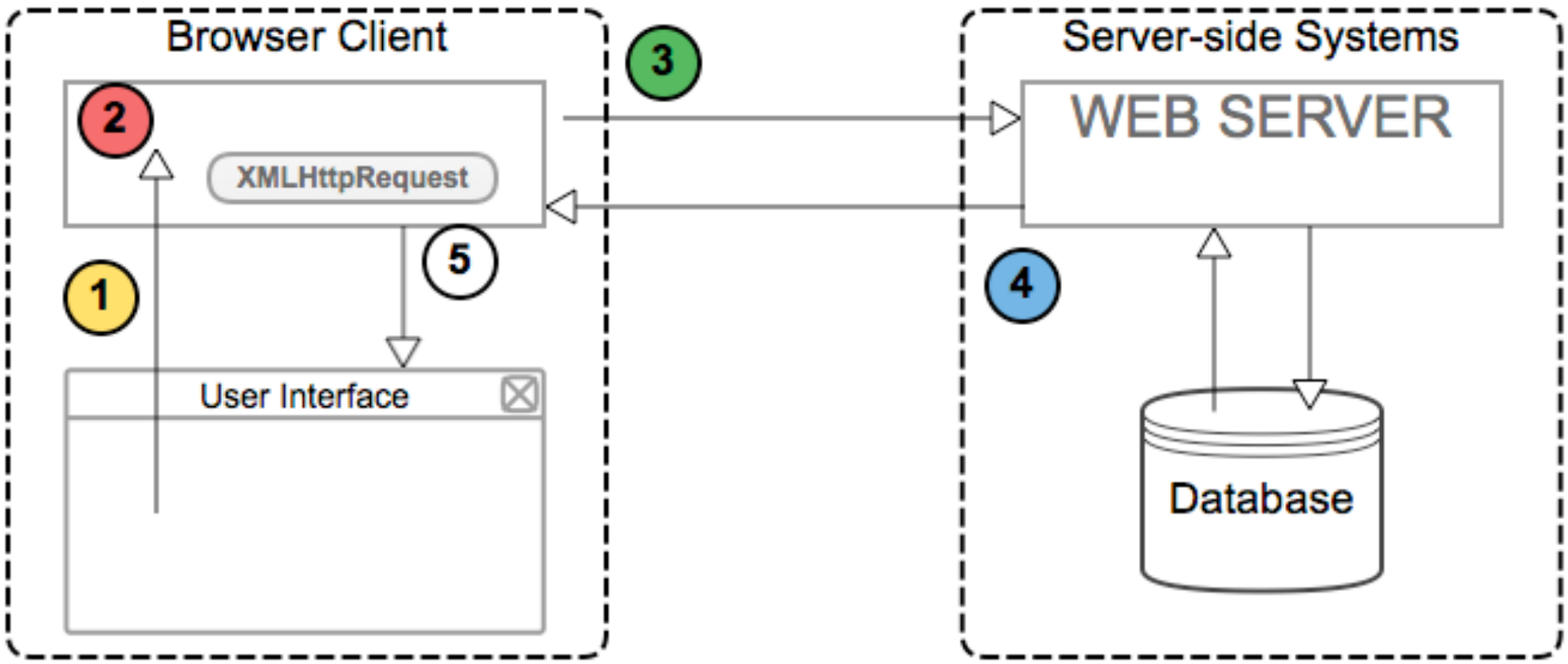
flickr@124247024@N07

We now have a fully functioning Web app!

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

Ajax: how does it work?



1. JavaScript call
2. XMLHttpRequest
3. HTTP request
4. Data returns
5. HTML & CSS data

Without Ajax …

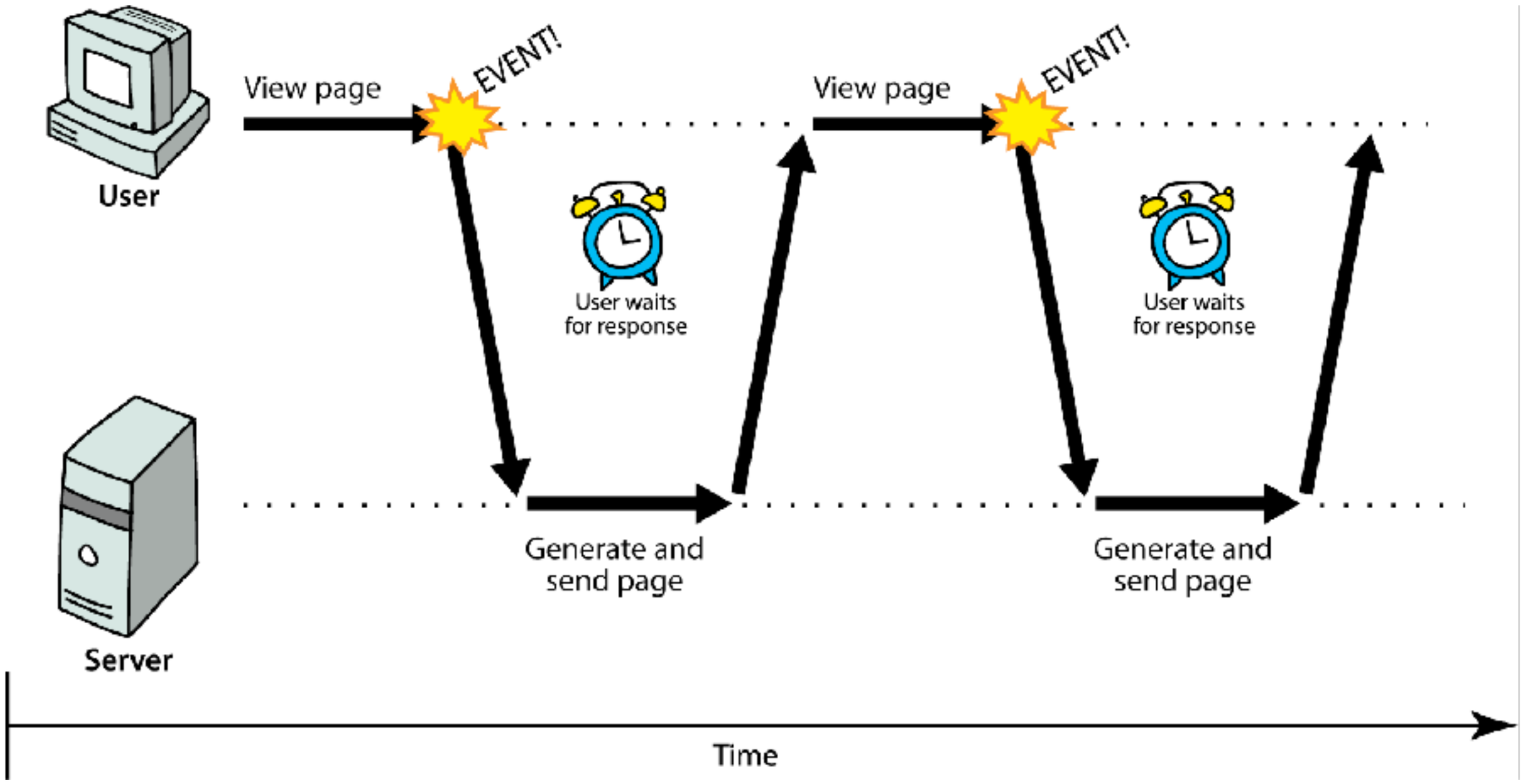


Image source: Web Programming Step by Step

Ajax works differently

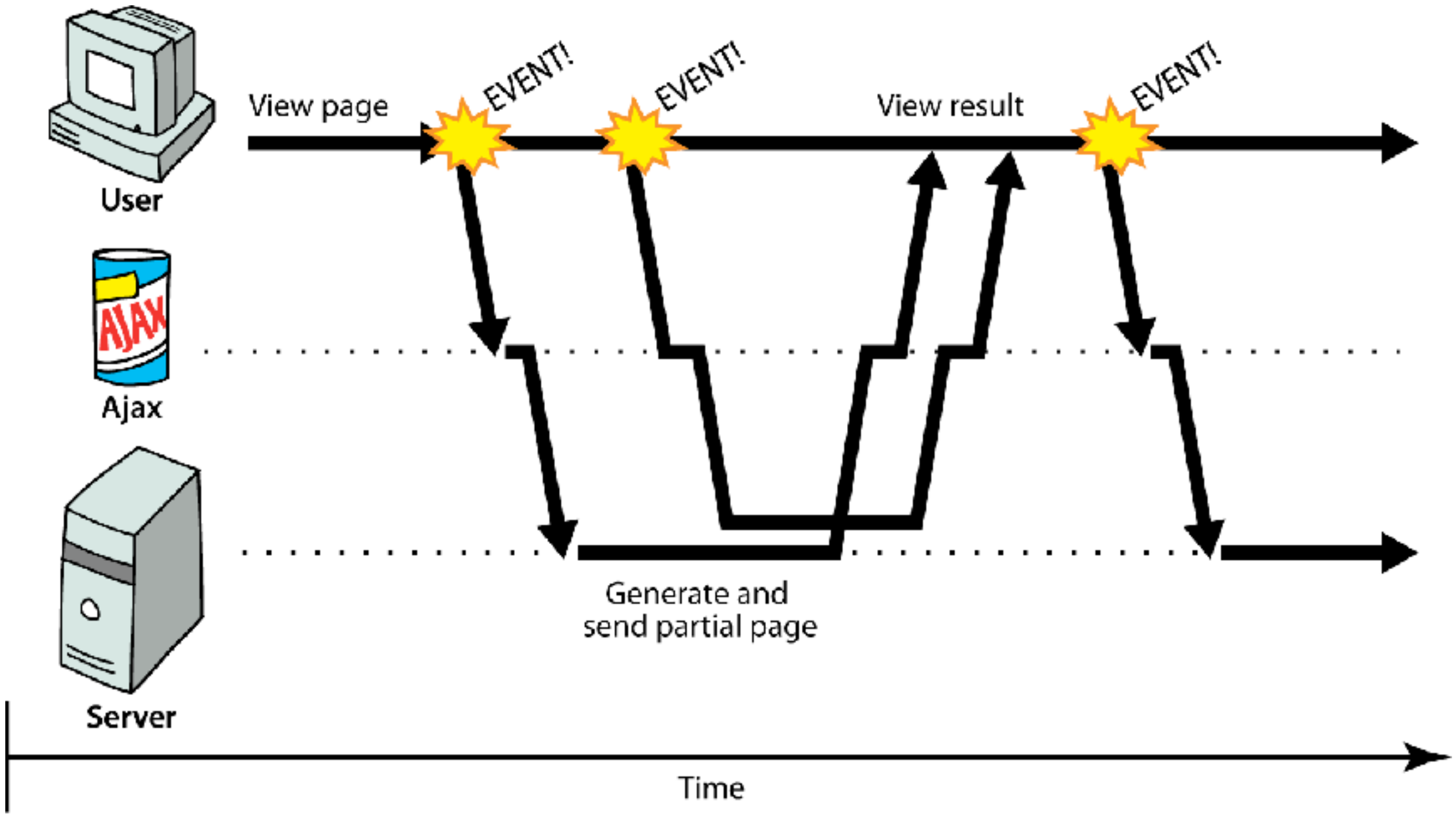


Image source: Web Programming Step by Step

Ajax security

- Conveniently we always requested data from "our" Web server
- **Security restriction of Ajax**: can only fetch files from the same Web server as the calling page (*Same-origin policy*)
  - Same origin when protocol, port, and host are the same for two pages
- Ajax **cannot** be executed from a Web page stored locally on disk

the course book explains how to get around this restriction (not recommended)