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 (focus on TODO app) that has client- and server-side interactivity
- **Implement** client/server communication via `JSON`
- **Implement** client-side code using Ajax

Lecture 1

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)

What is node.js?

node.js in its own words …

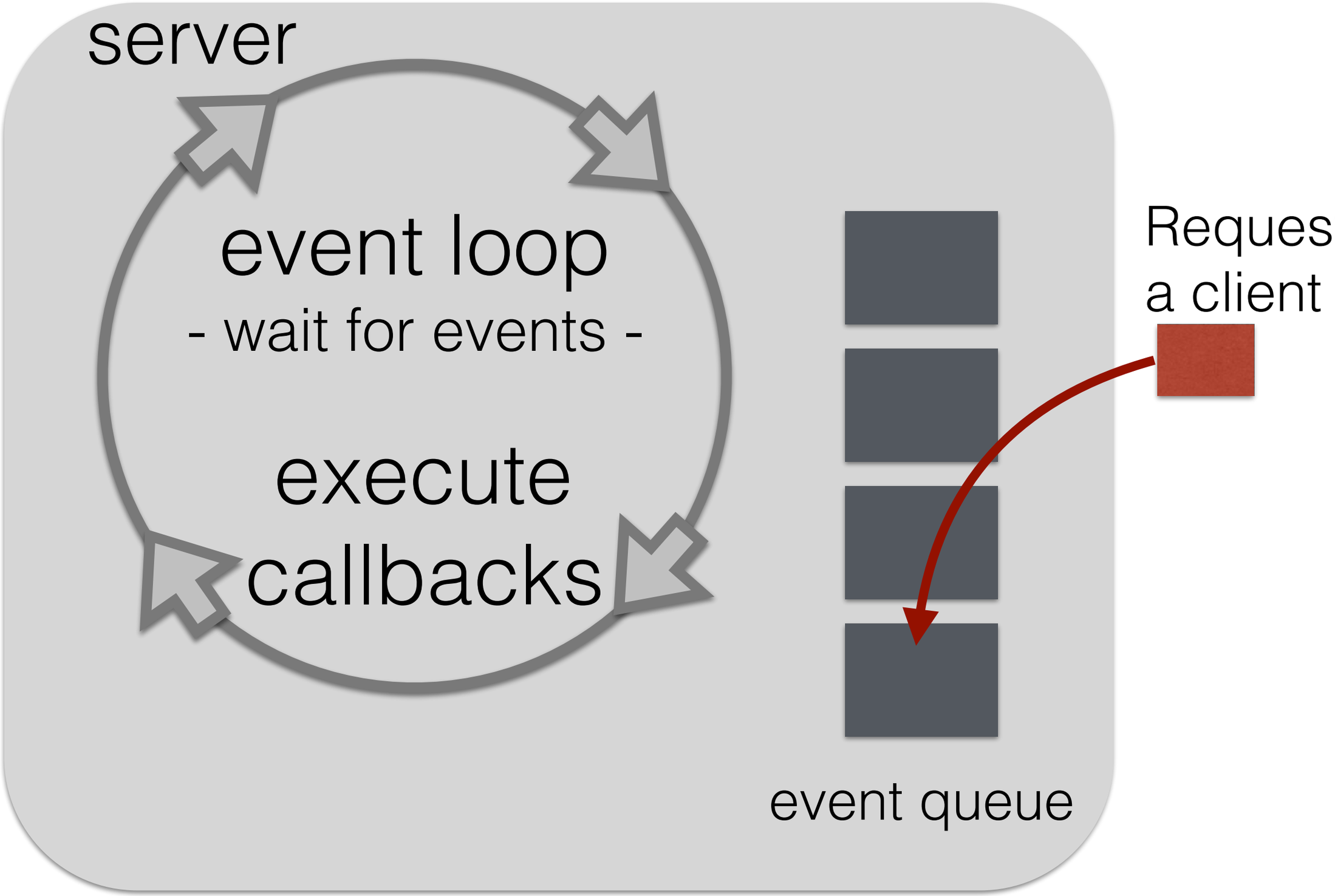
"Node.js® is a platform **built on Chrome's JavaScript runtime** for easily building **fast**, **scalable network** applications.

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."

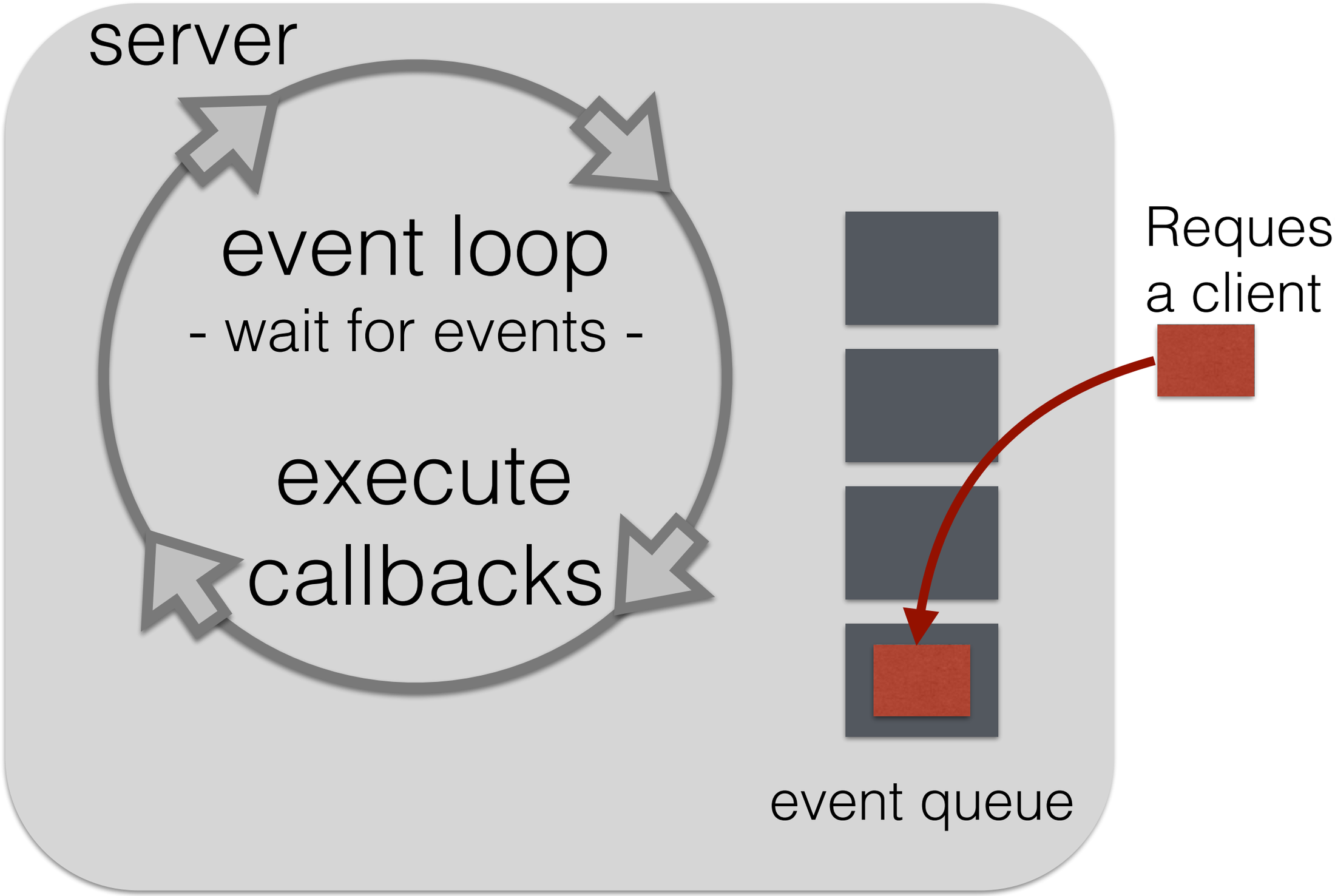
History of node.js

- 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
  - **200,000** packages [**npm**]
  - **158,000** packages [**CPAN**]
- December 2014: **io.js**
- May 2015: **io.js merges back** with node.js

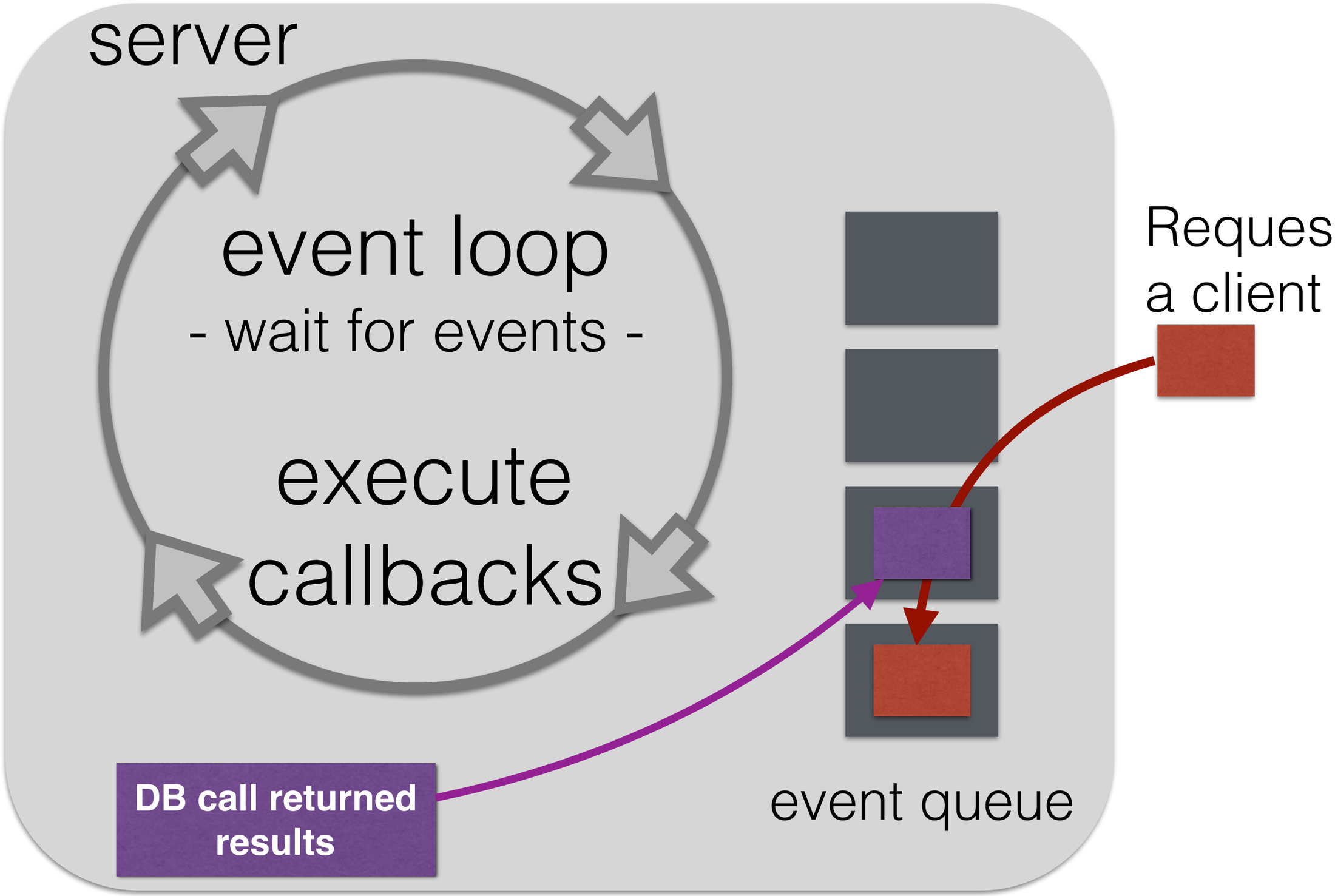
node.js is event-driven



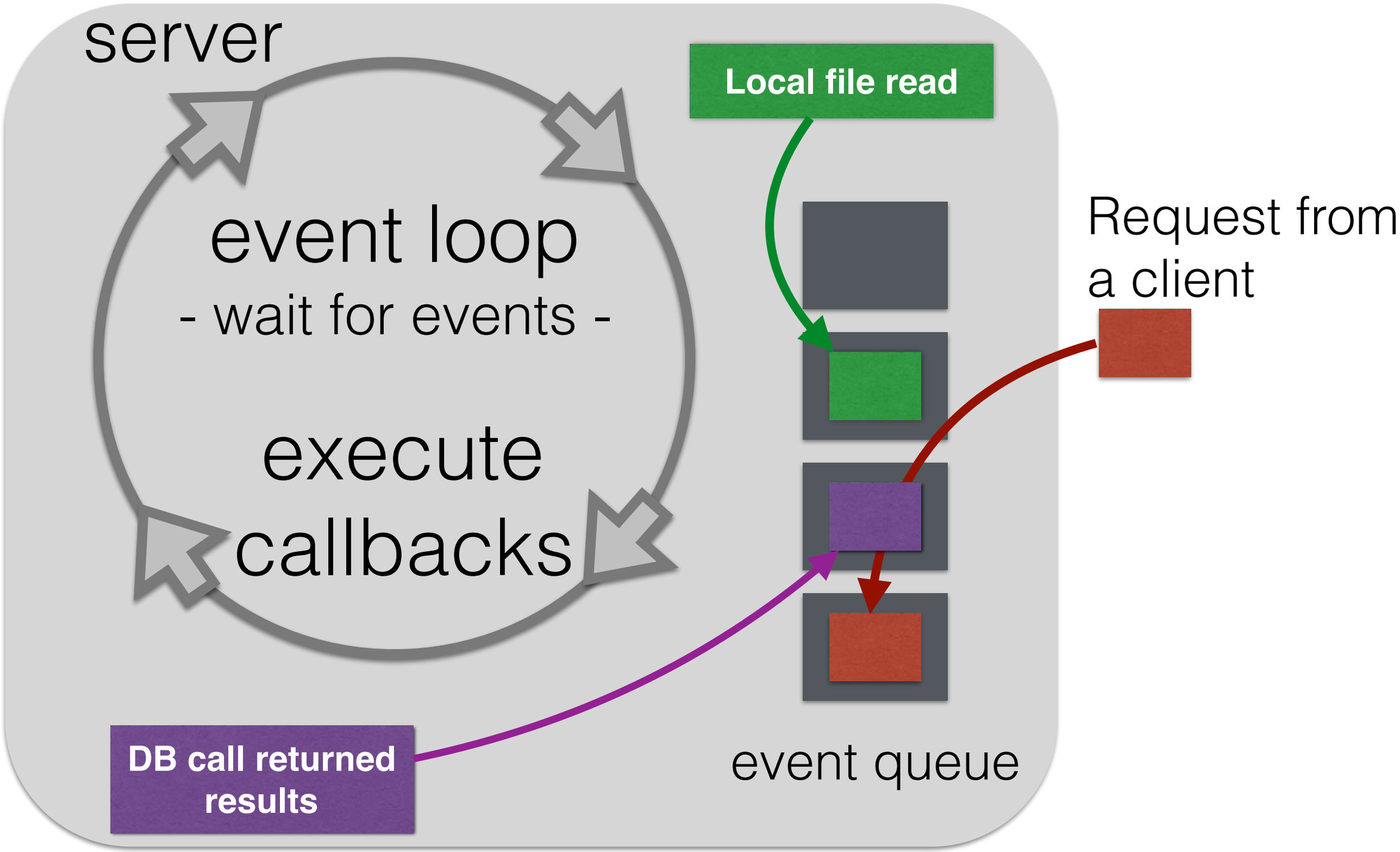
node.js is event-driven



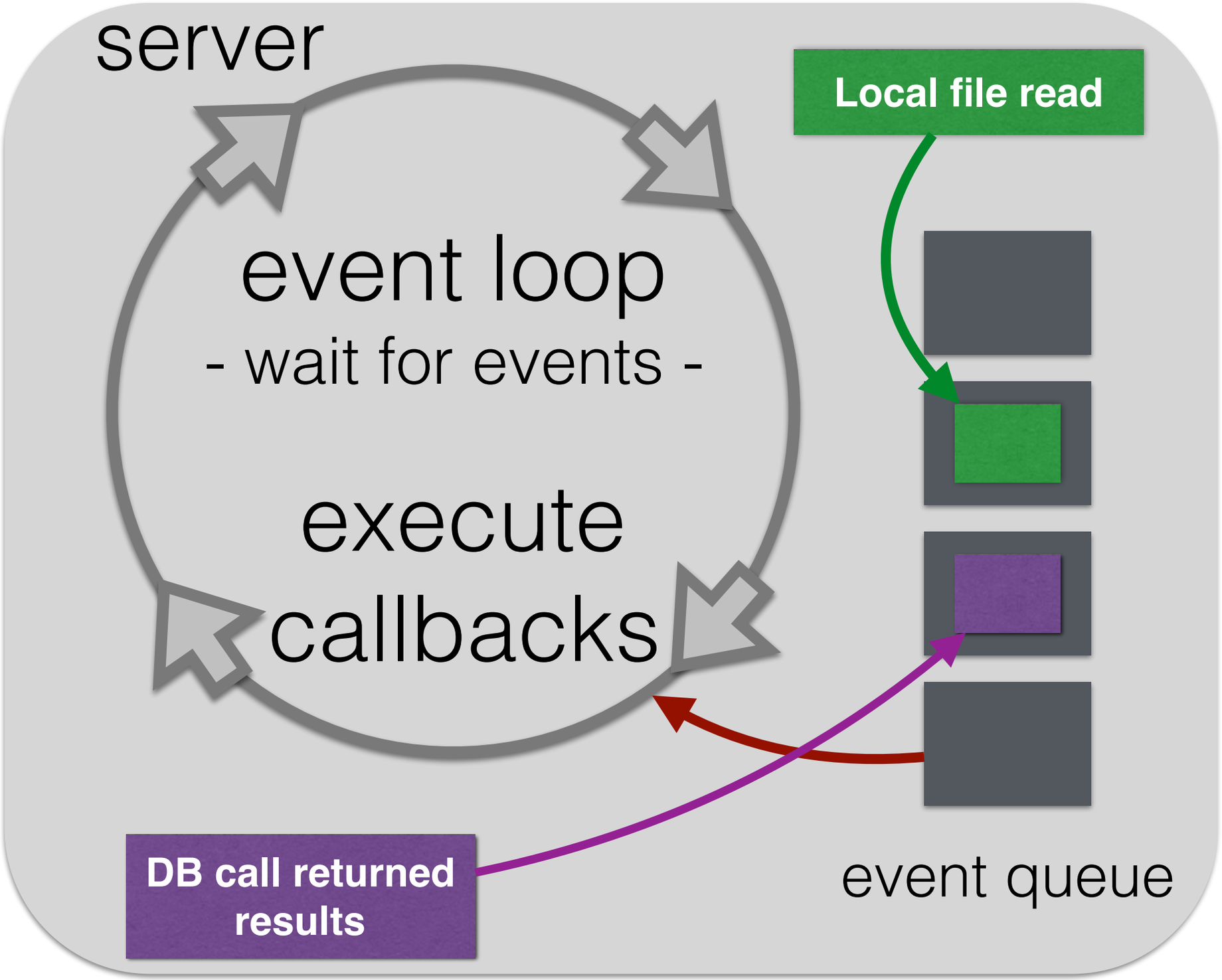
node.js is event-driven



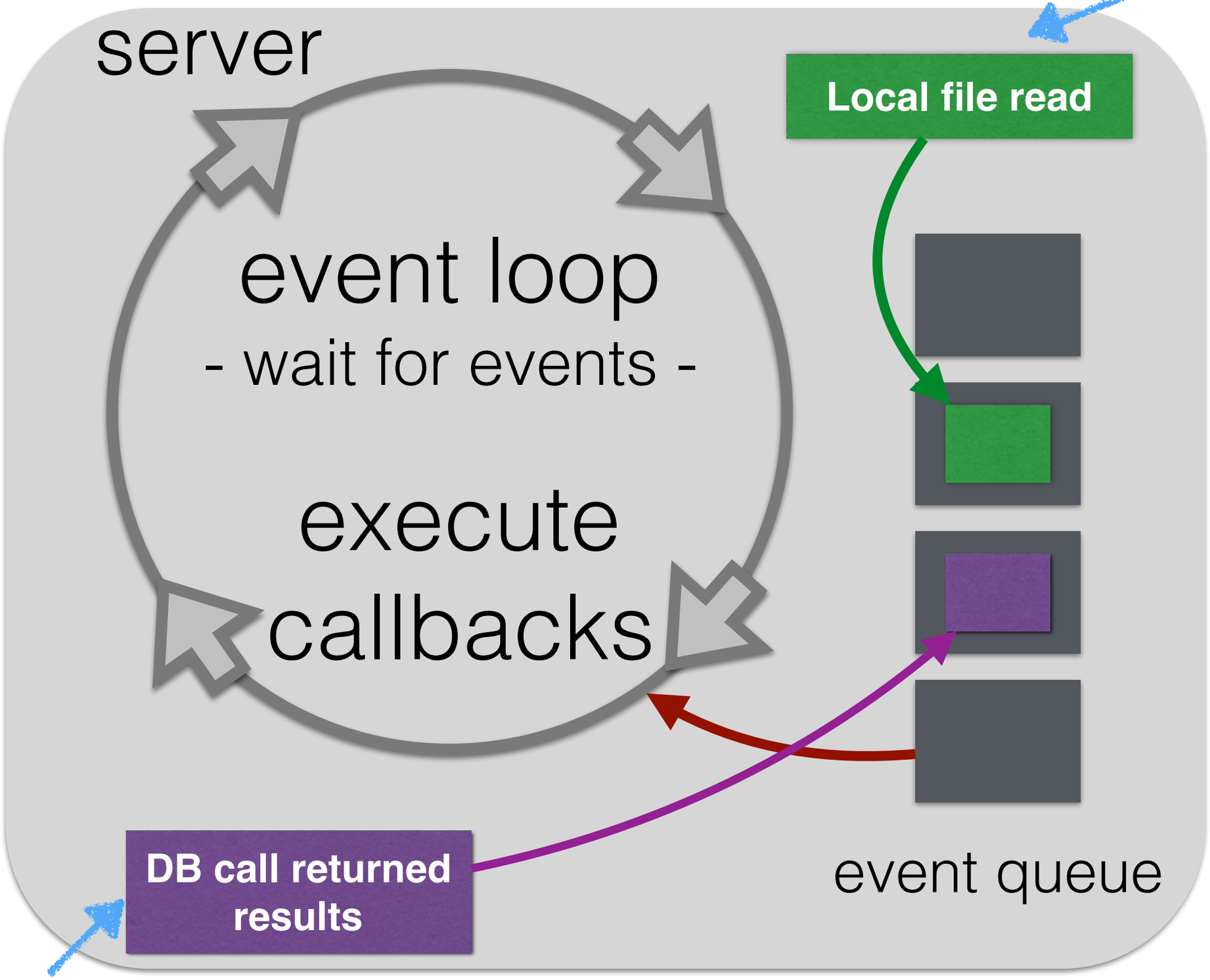
node.js is event-driven



node.js is event-driven

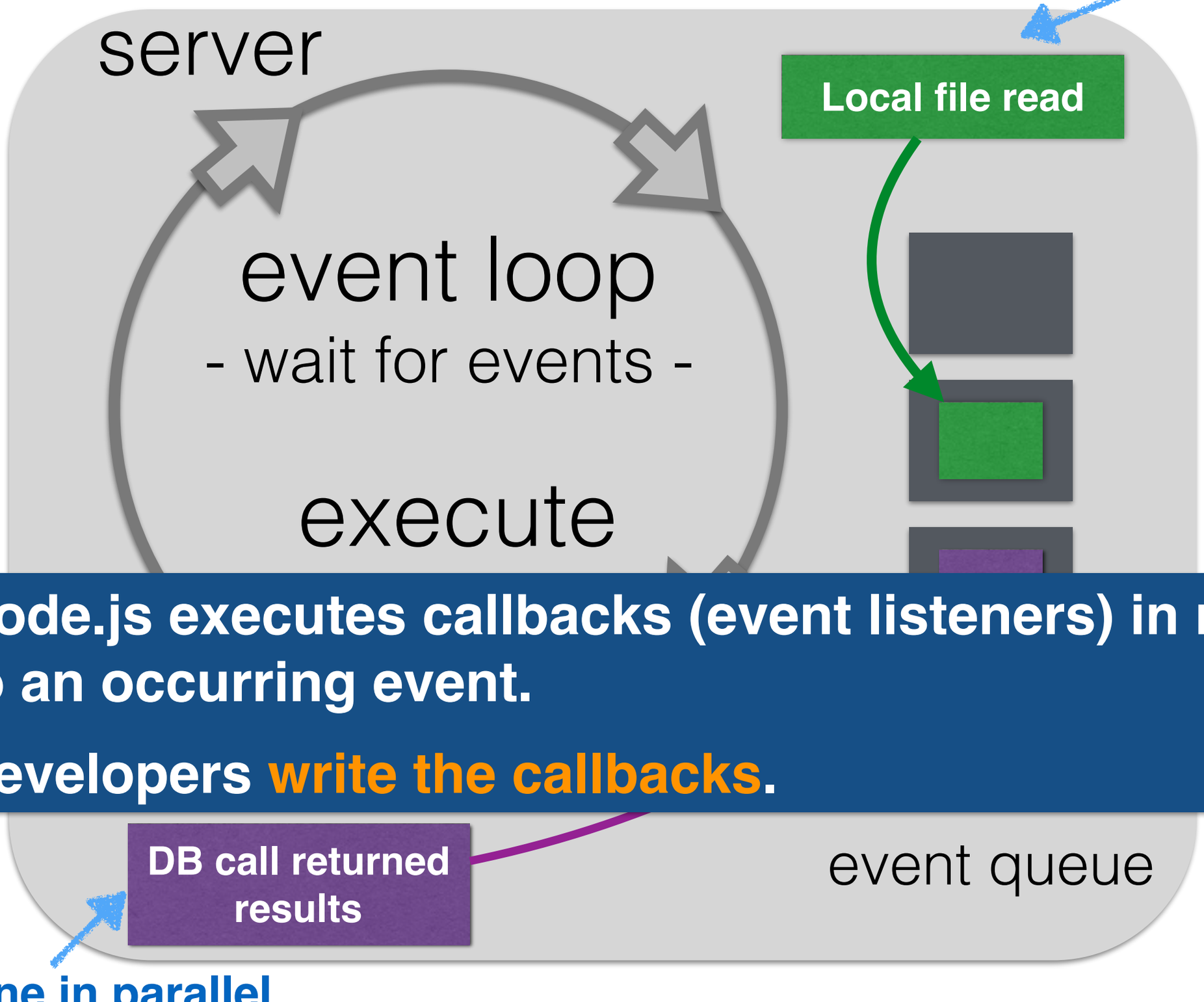


node.js is event-driven



node.js is event-driven

done in parallel



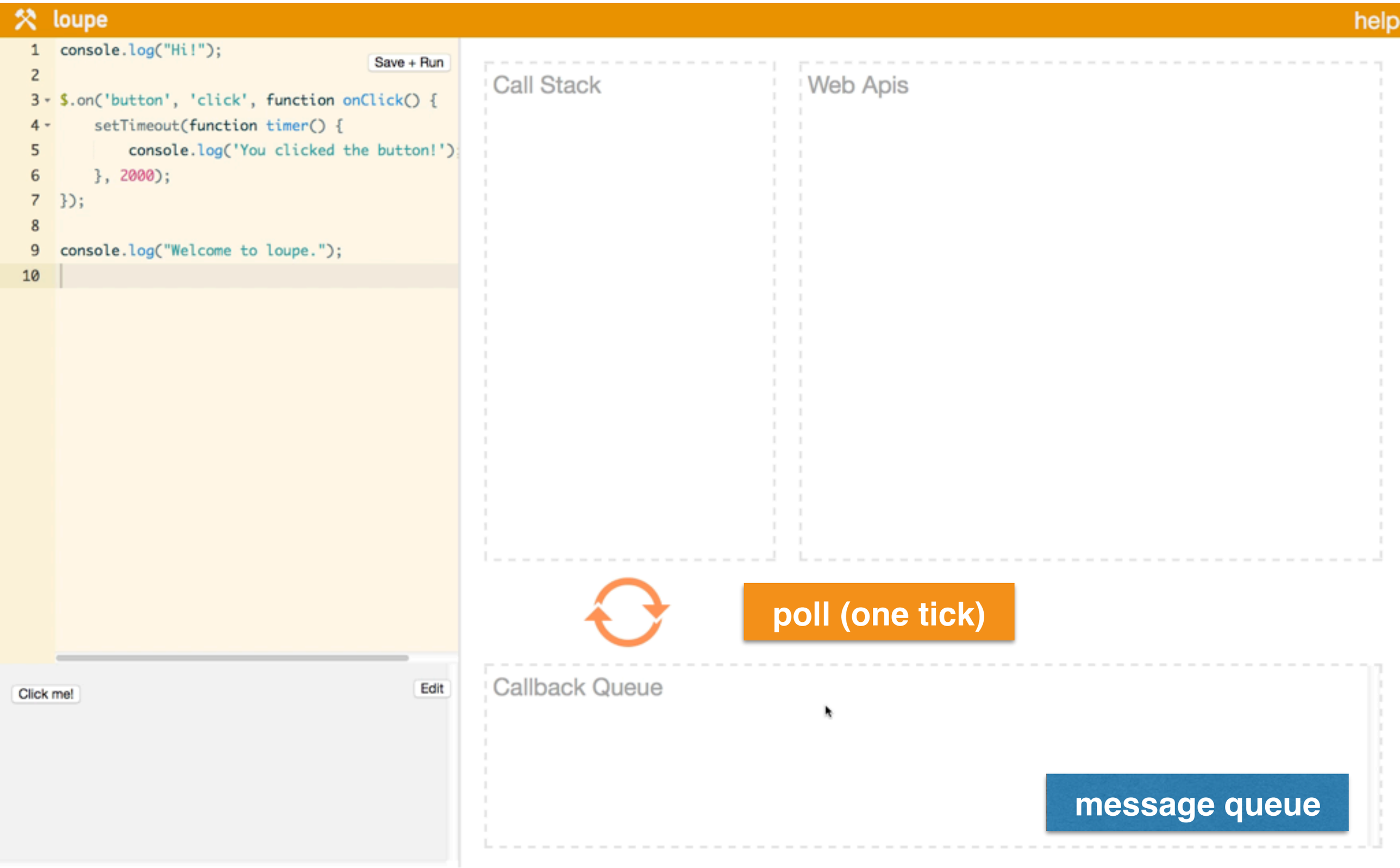
Node.js executes callbacks (event listeners) in response to an occurring event.

Developers write the callbacks.

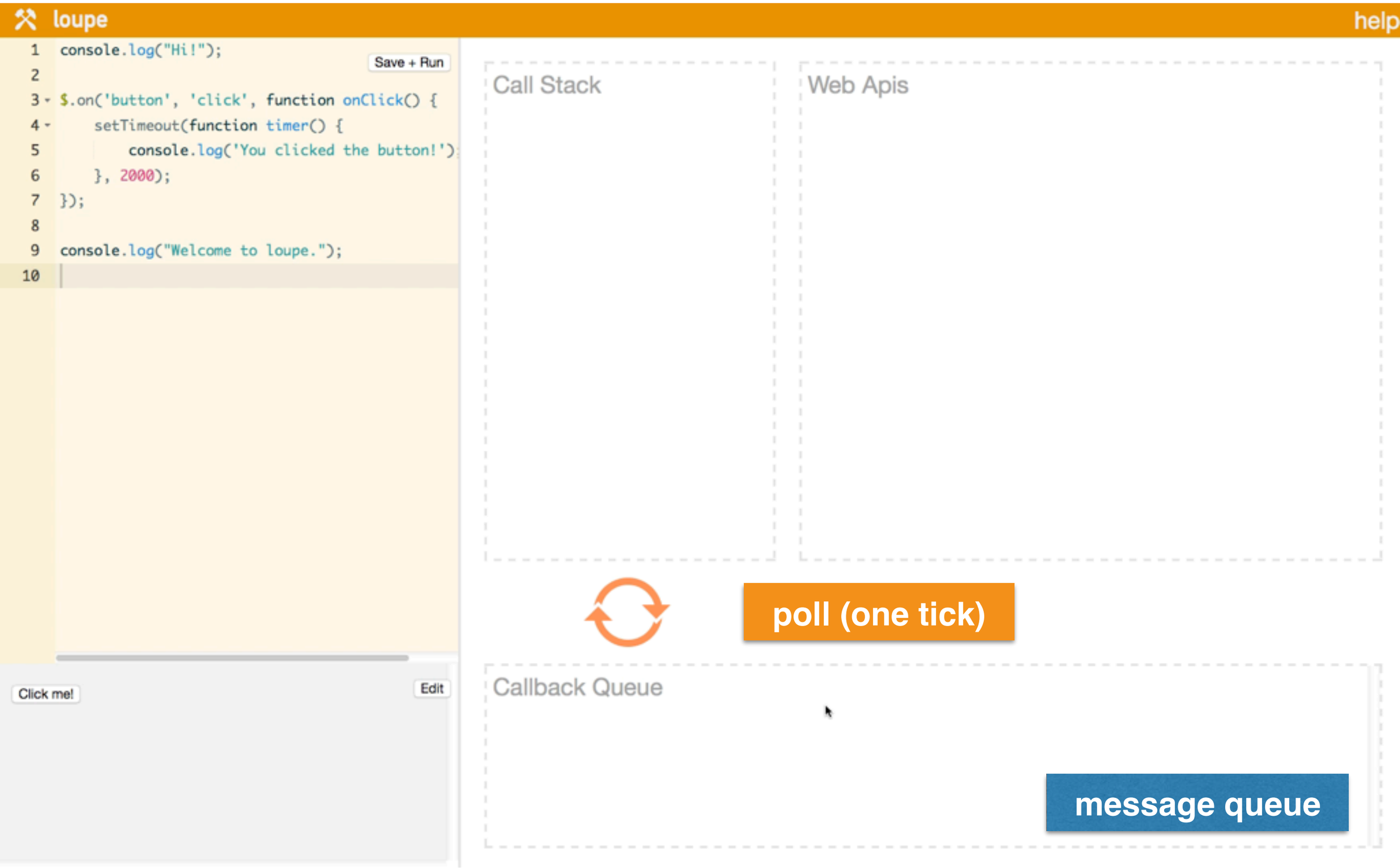
done in parallel

It is actually a bit more complicated than that …

http://latentflip.com/loupe/



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: 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)

Non-blocking I/O

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

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

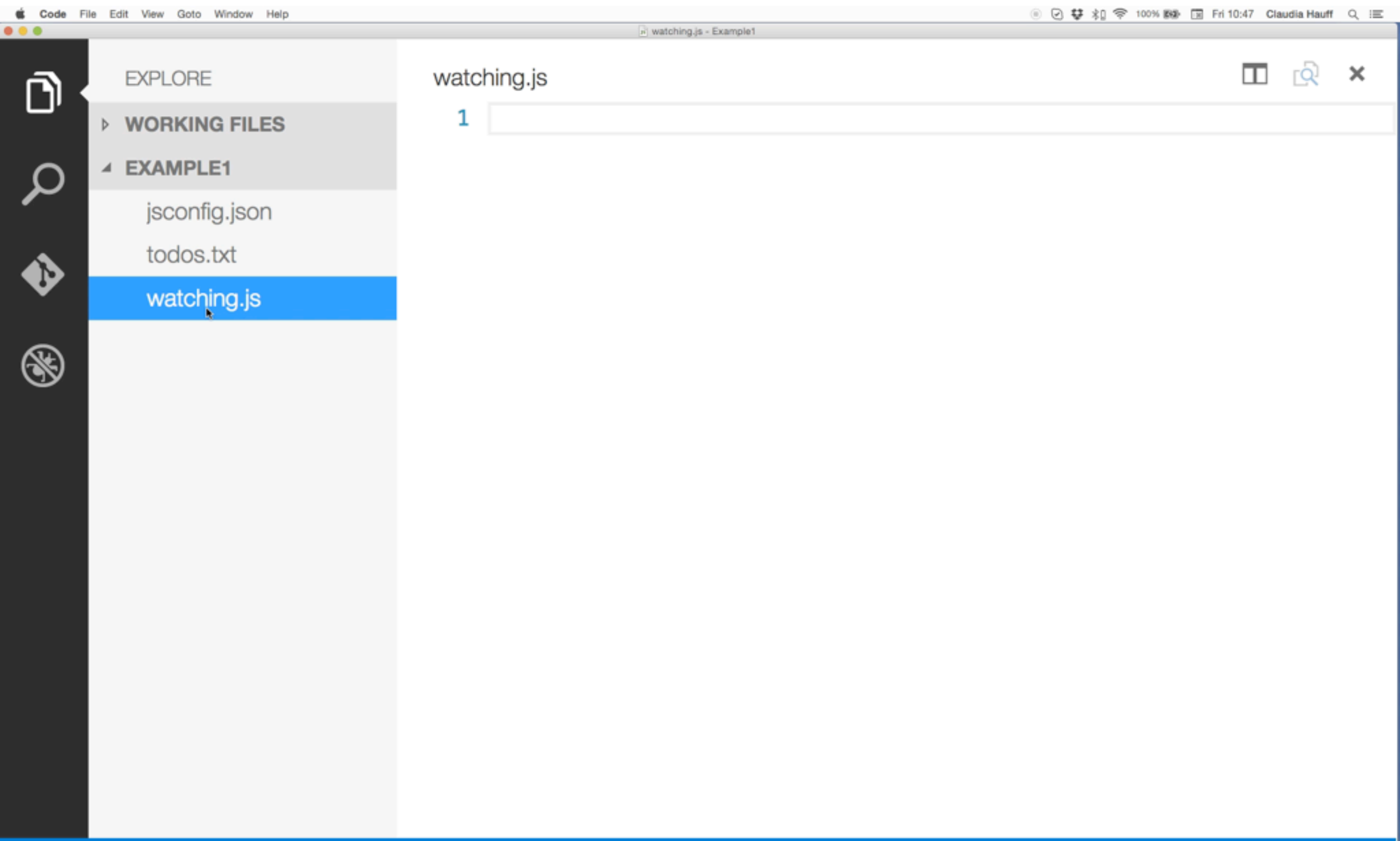
(3) do other things

(4) when the callback returns, process it

The first code examples

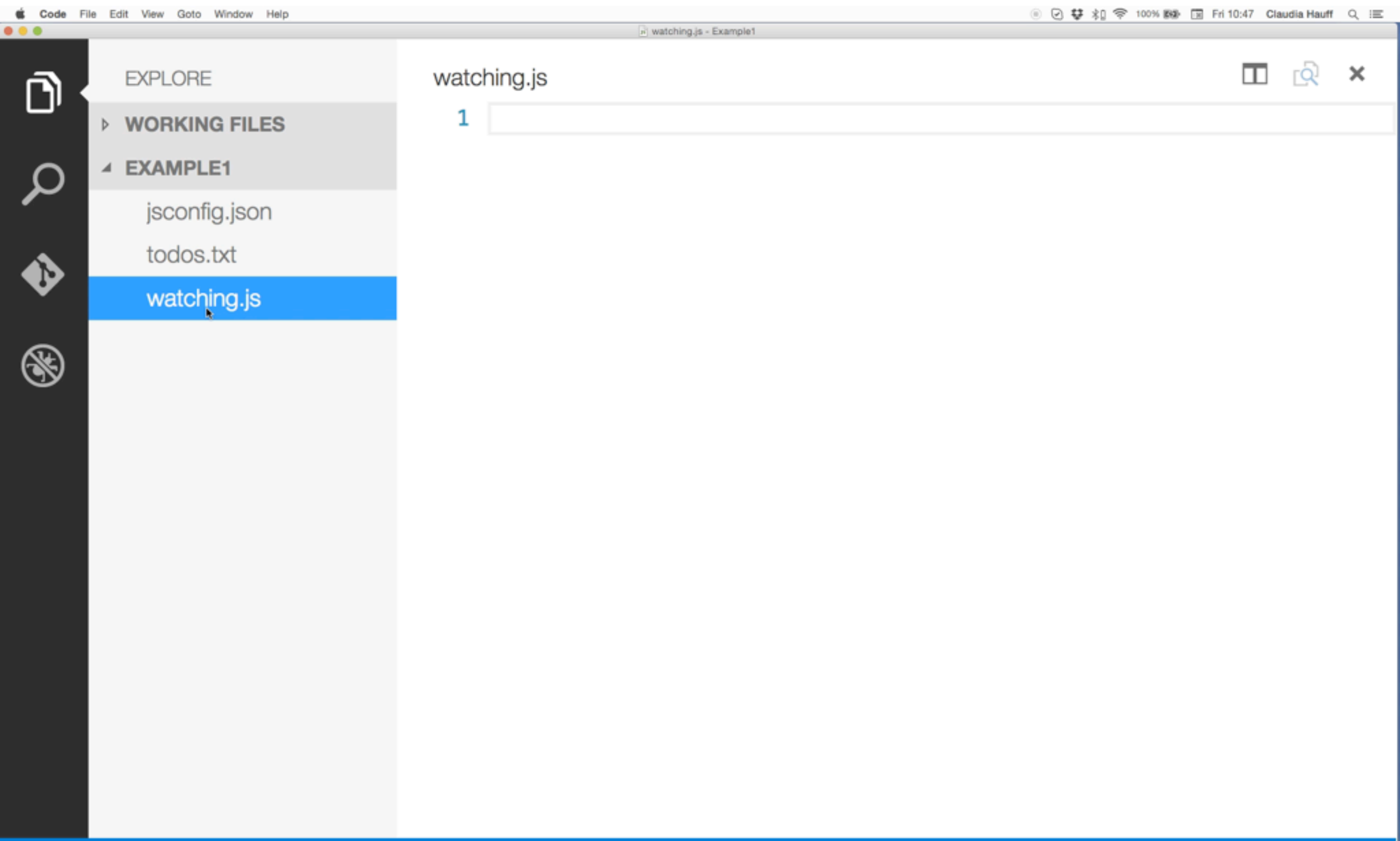
Example 1

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



Example 1

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



Example: watch a file for changes

```
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'");
```

Assumption: file to watch exists!

Example: watch a file for changes

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

```
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'");
```

Assumption: file to watch exists!

Example: watch a file for changes

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

node.js fs **module**

```
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'");
```

Assumption: file to watch exists!

Example: watch a file for changes

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

Self-contained piece of code that provides reusable functionality.

node.js fs **module**

```
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'");
```

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.

```
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'");
```

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.

```
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'");
```

polls ‘todos.txt’ **for changes**

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.

```
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'");
```

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' has \
4                  just changed");
5 });
6 console.log("Now watching 'todos.txt'");
```

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'");
```

anonymous function

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

anonymous function

asynchronous

```
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'");
```

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'");
```

anonymous function

asynchronous

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

Assumption: file to watch exists!

0:31

Does `require()` use synchronous or asynchronous access?

```
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'");
```

0:31

Does `require()` use synchronous or asynchronous access?

```
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'");
```

0:35

```
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);
```

What is the console output when executing this Javascript code snippet?

0:35

```
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);
```

What is the console output when executing this Javascript code snippet?

```
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.

0:31

```
1 var bar = 12;
2
3 function X(){
4  bar = 6;
5  var bar;
6 }
7 console.log(bar);
8 X();
9 console.log(bar);
```

What is the console output when executing this Javascript code snippet?

flickr@isolino

0:31

```
1 var bar = 12;
2
3 function X(){
4  bar = 6;
5  var bar;
6 }
7 console.log(bar);
8 X();
9 console.log(bar);
```

What is the console output when executing this Javascript code snippet?

flickr@isolino

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

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

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`.

`todos.txt`

`tcp.js`

Example: low-level networking with node.js

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

`todos.txt`

`tcp.js`

```
server:~ $
Listening for subscribers …
```

Example: low-level networking with node.js

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

`todos.txt`

(1) client connects to server

```
server:~ $
Listening for subscribers …
```

Example: low-level networking with node.js

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

`todos.txt`

(1) client connects to server

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

Example: low-level networking with node.js

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

`todos.txt`

(1) client connects to server

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

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

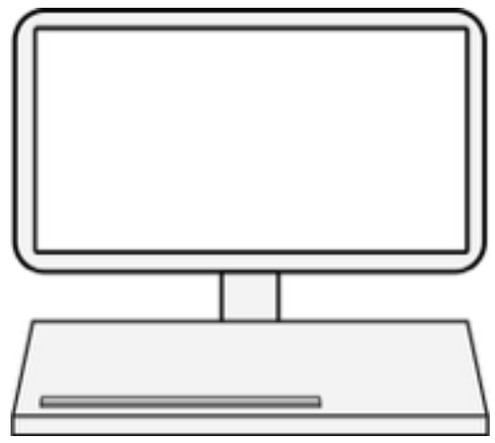
Example: low-level networking with node.js

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

`todos.txt`

`tcp.js`

(1) client connects to server

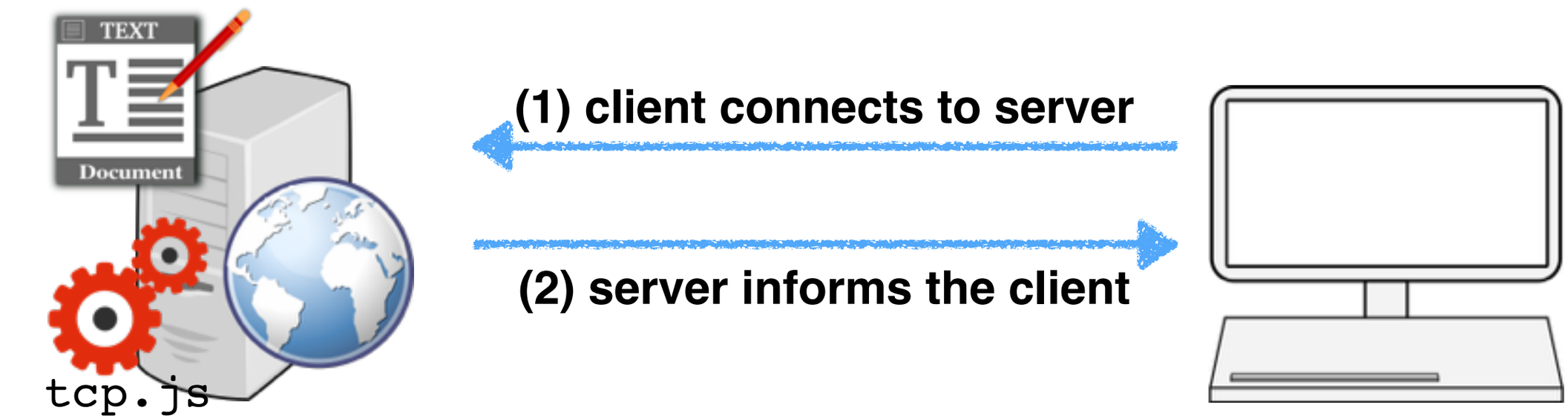


```
server:~ $
Listening for subscribers …
Subscriber connected.
```

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

Example: low-level networking with node.js

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

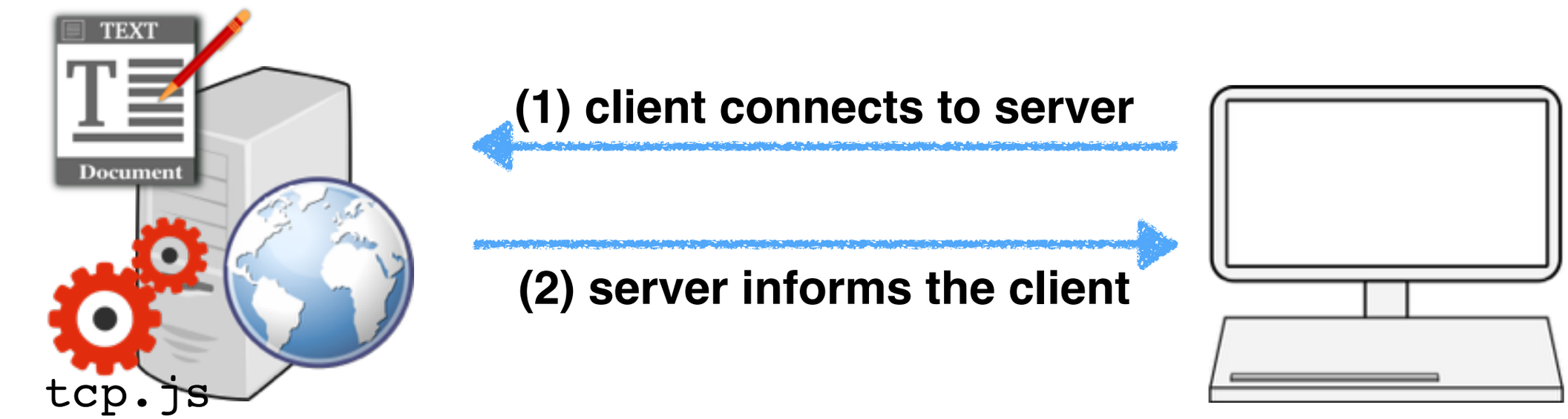



```
server:~ $
Listening for subscribers …
Subscriber connected.
```

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

Example: low-level networking with node.js

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

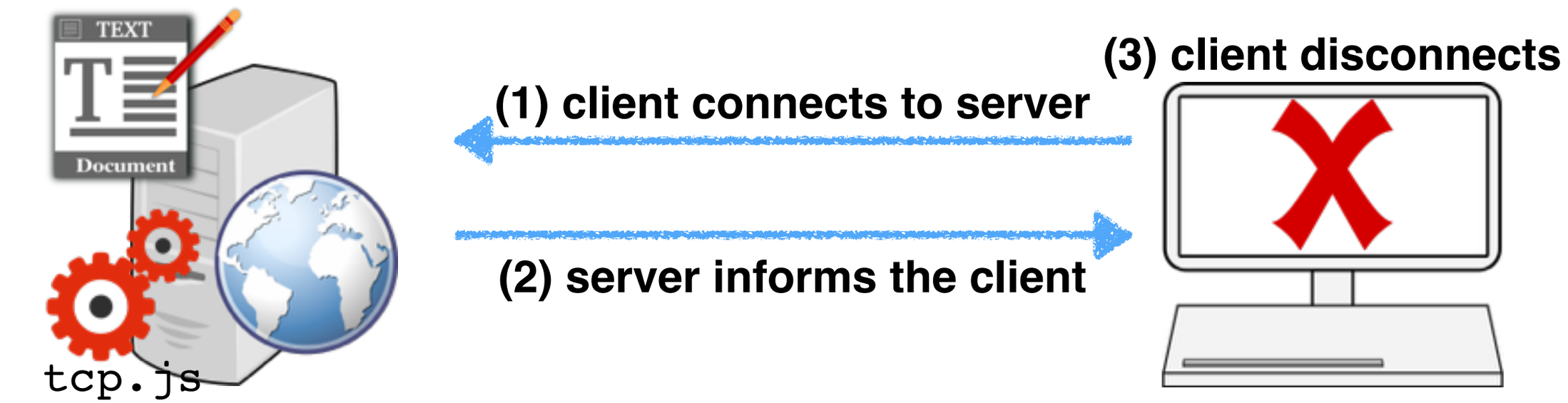



```
server:~ $
Listening for subscribers …
Subscriber connected.
```

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

Example: low-level networking with node.js

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

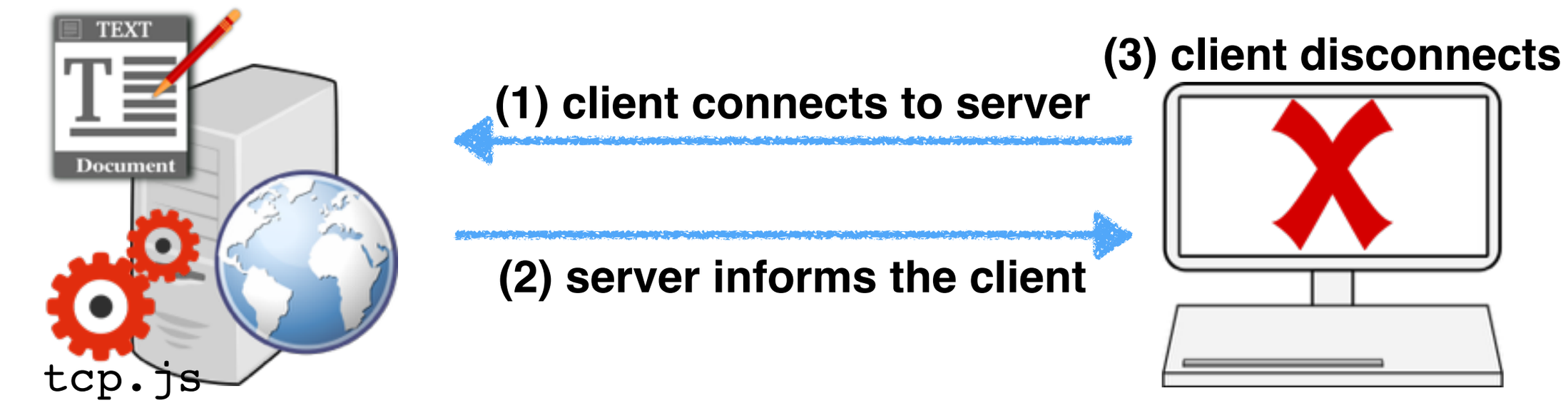



```
server:~ $
Listening for subscribers …
Subscriber connected.
```

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

Example: low-level networking with node.js

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

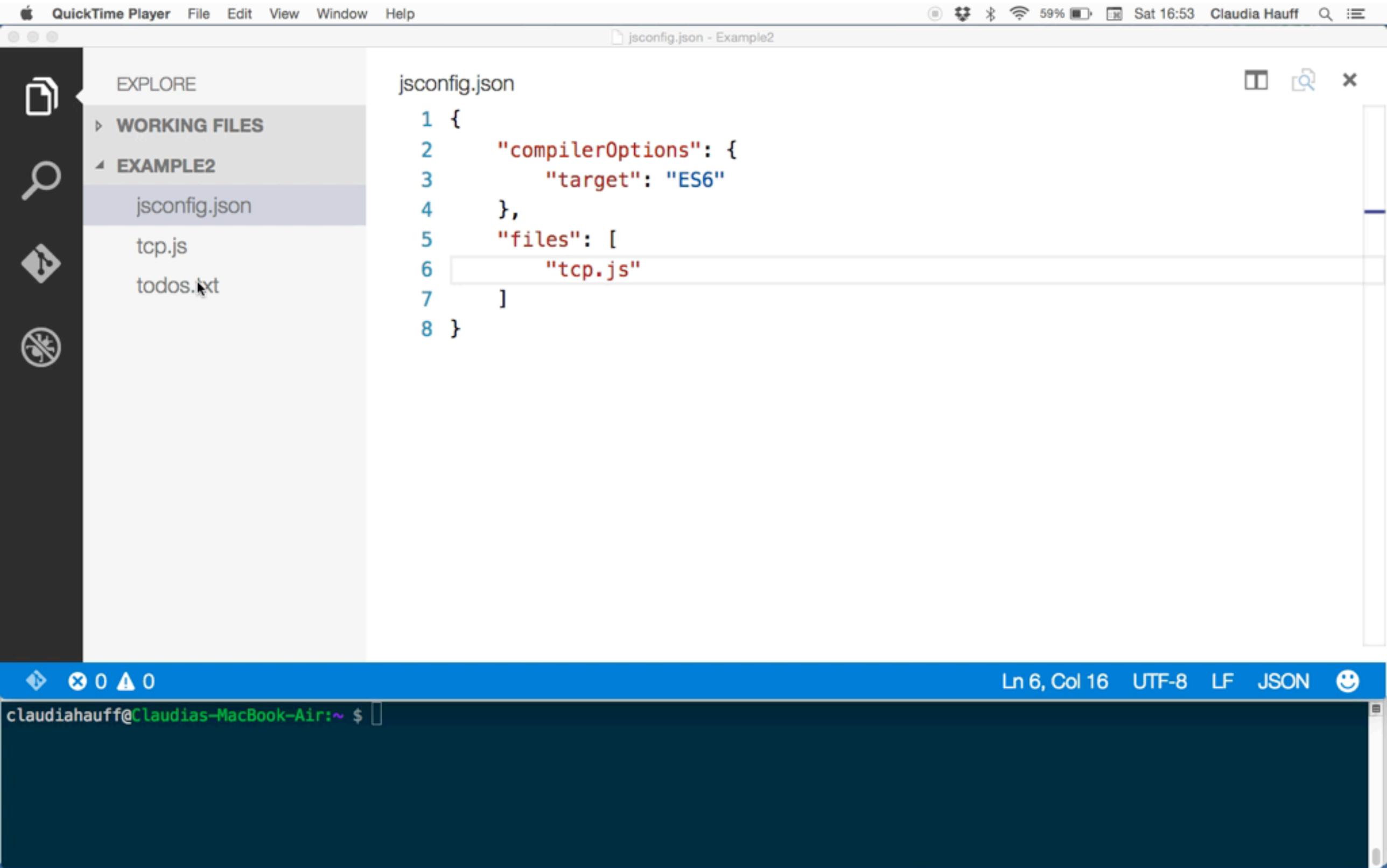



```
server:~ $
Listening for subscribers …
Subscriber connected.
Subscriber disconnected.
```

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

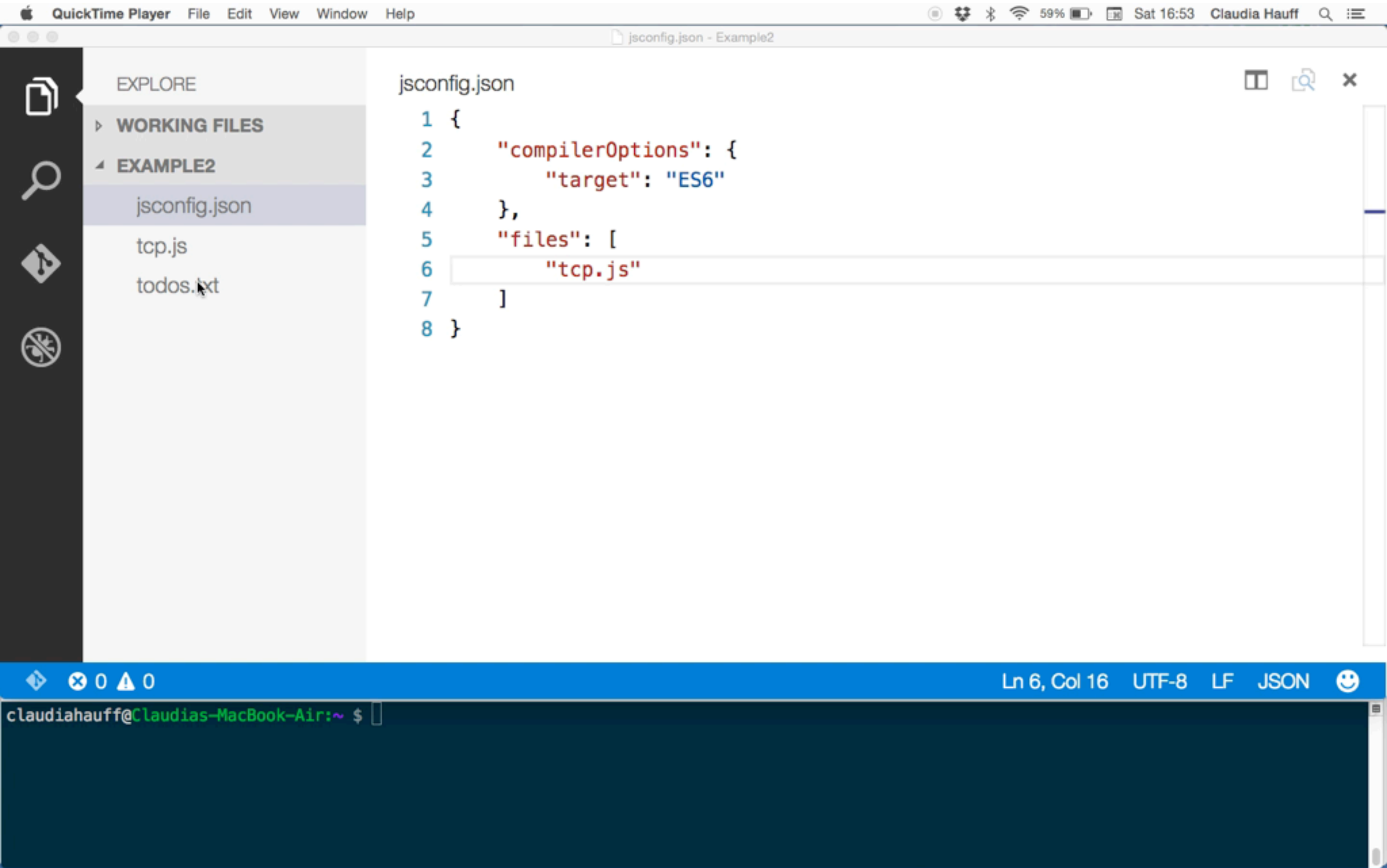
Example 2

Example: low-level networking with node.js



Example 2

Example: low-level networking with node.js



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);
```

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

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

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

```
1  'use strict';
2  const
3     fs = require('fs'),
4     net = require('net'),
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

Example: low-level networking with node.js

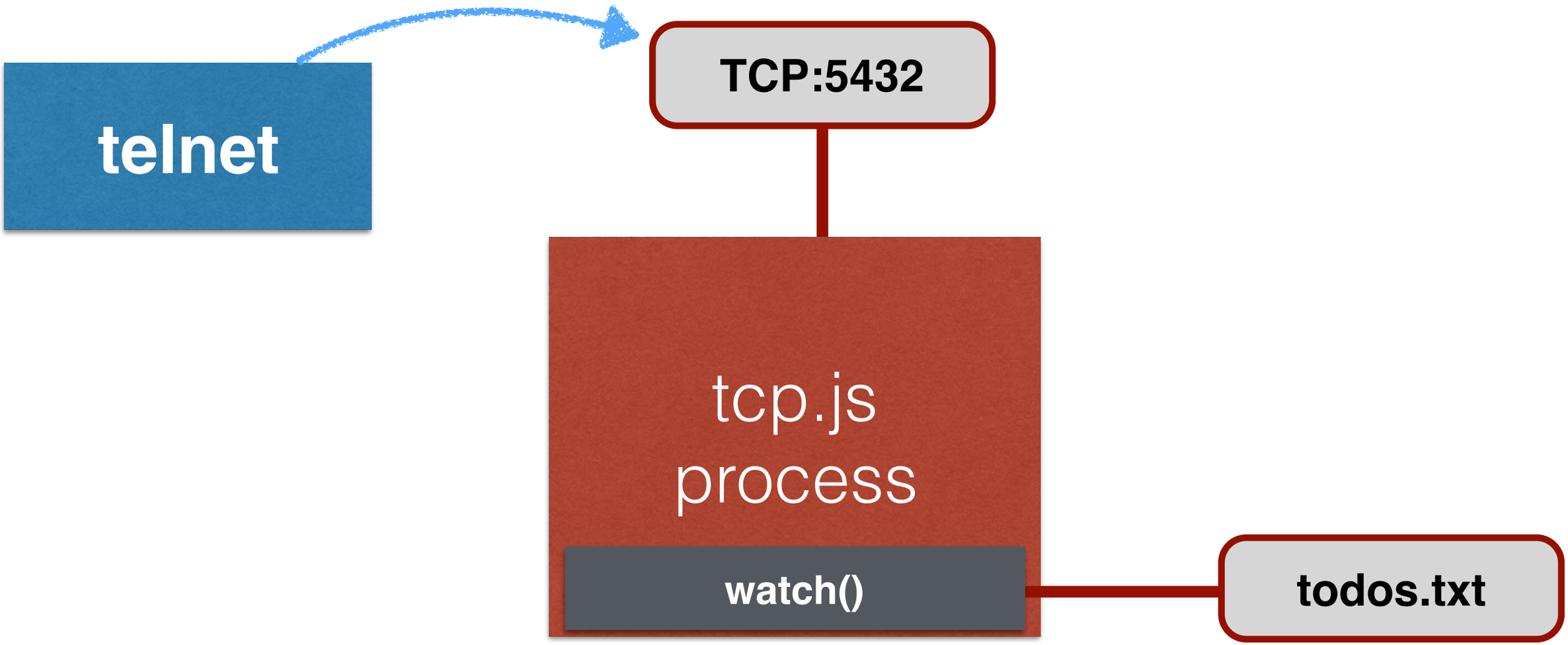
Start the **server** on the command line: **`$ node --harmony tcp.js`**
Client terminal: `$ telnet localhost 5432`

```
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

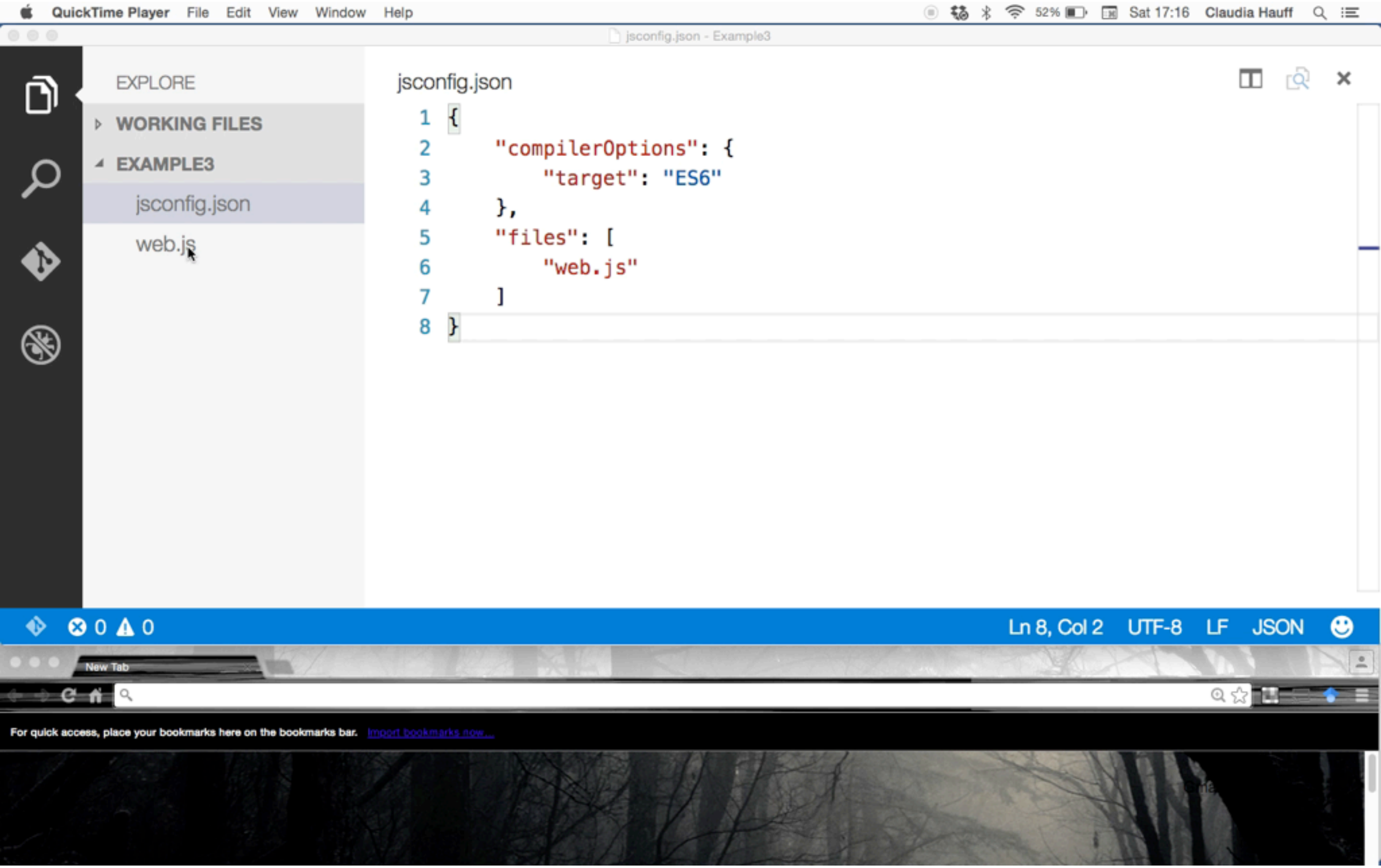


Using node.js to create a Web server

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

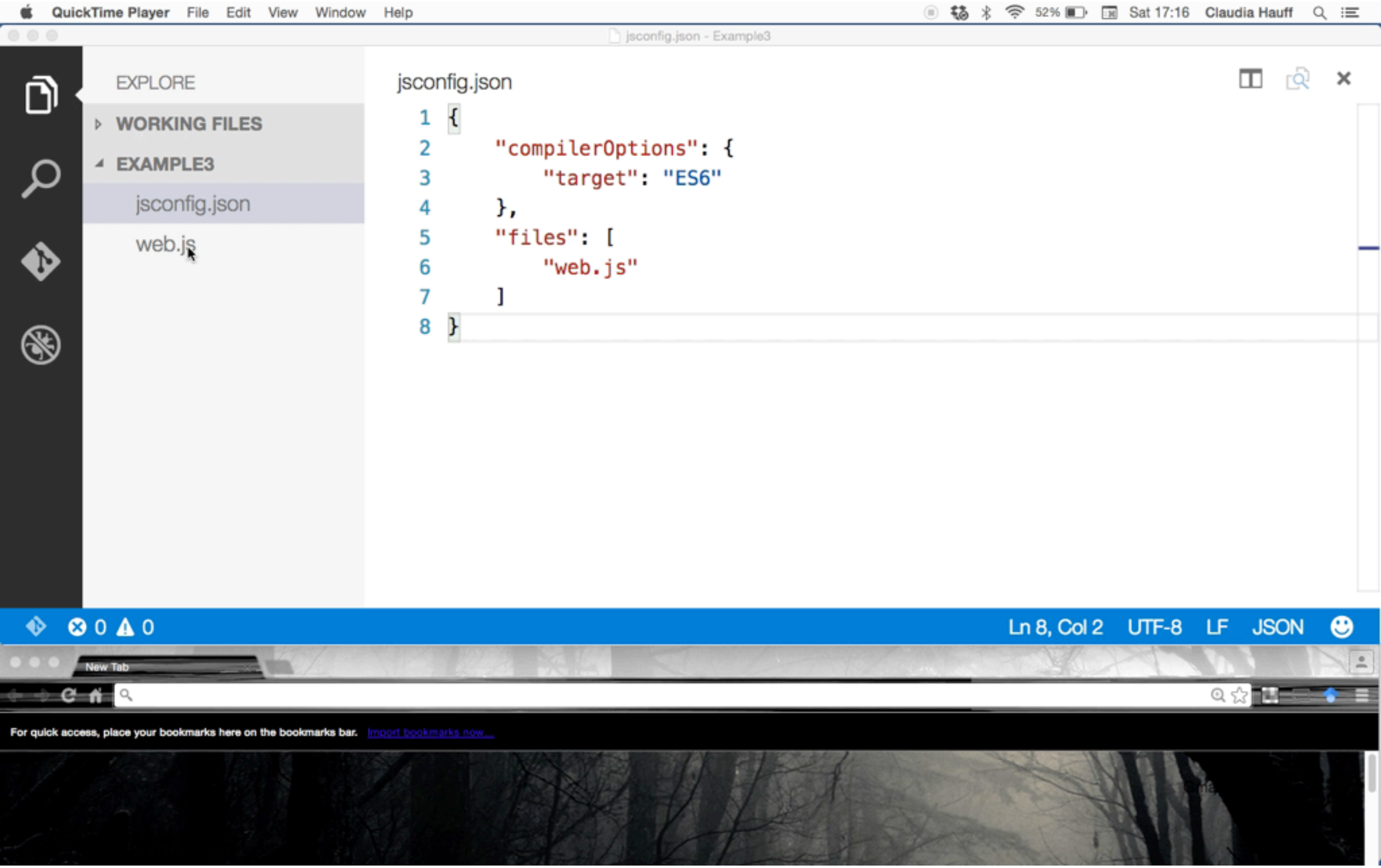
Example 3

“Hello World” in the browser



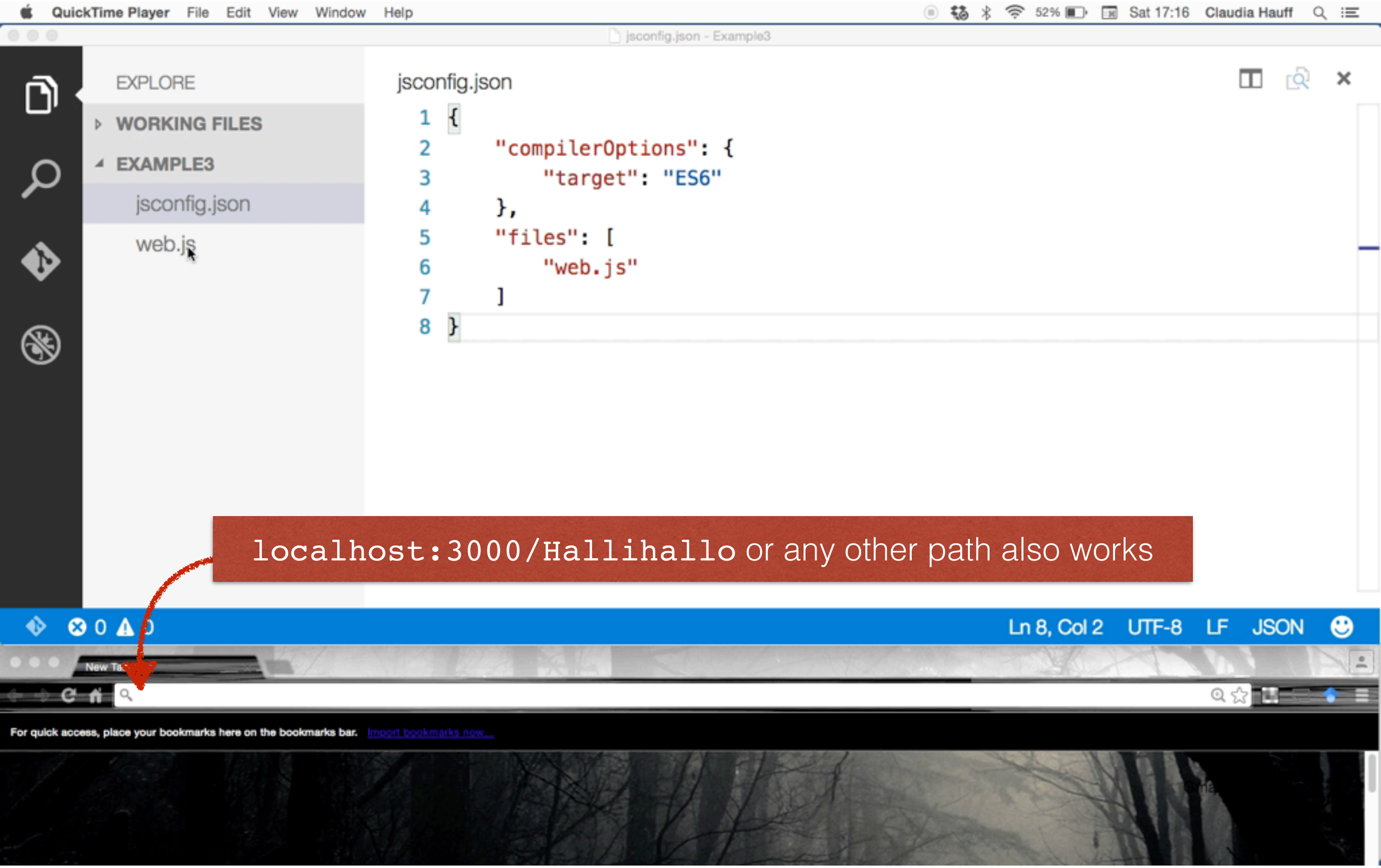
Example 3

“Hello World” in the browser



Example 3

“Hello World” in the browser



The “Hello World” of node.js

```
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");
```

The “Hello World” of node.js

node.js http module

```
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");
```

The “Hello World” of node.js

node.js http module

Create a Web server

```
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");
```

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");
```

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

The “Hello World” of node.js

node.js http module

```
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 **Web server**

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`**
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);
```

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…

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 **request** object

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

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)

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);
```

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);
```

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

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**

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

QuickTime Player File Edit View Window Help — 26% — Sat 19:26 — Claudia Hauff

routing.js - Example3

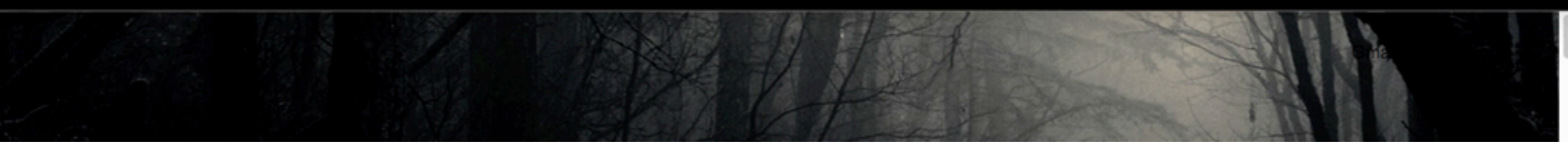
EXPLORE

- WORKING FILES
- EXAMPLE3
  - .vscode
    - launch.json
  - jsconfig.json
  - routing.js

routing.js

```
2     url = require('url'),
3     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, {
9             "Content-Type": "text/plain"
10        });
11        var query = url_parts.query;
12        if (query["name"] !== undefined) {
13            res.end("Greetings " + query["name"]);
14        }
15        else {
16            res.end("Greetings Anonymous");
17        }
18    }
19    else {
20        res.writeHead(404, {
21            "Content-Type": "text/plain"
22        });
23        res.end("Only /greetme is implemented!");
24    }
25 };
26 server = http.createServer(simpleHTTPResponder);
27 var port = 3000;
28 server.listen(port, function () {
29     console.log("Server listening on port " + port);
30 });
31
```

0 0 — Ln 1, Col 1 — UTF-8 — LF — JavaScript



Example 4

URL routing

QuickTime Player File Edit View Window Help — 26% — Sat 19:26 — Claudia Hauff

routing.js - Example3

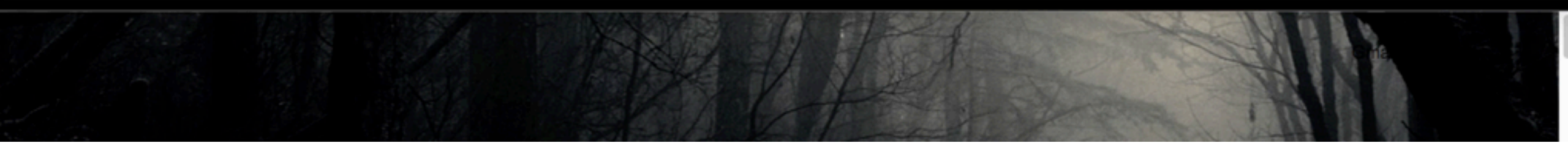
EXPLORE

- WORKING FILES
- EXAMPLE3
  - .vscode
    - launch.json
  - jsconfig.json
  - routing.js

routing.js

```
2     url = require('url'),
3     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, {
9             "Content-Type": "text/plain"
10        });
11        var query = url_parts.query;
12        if (query["name"] !== undefined) {
13            res.end("Greetings " + query["name"]);
14        }
15        else {
16            res.end("Greetings Anonymous");
17        }
18    }
19    else {
20        res.writeHead(404, {
21            "Content-Type": "text/plain"
22        });
23        res.end("Only /greetme is implemented!");
24    }
25 };
26 server = http.createServer(simpleHTTPResponder);
27 var port = 3000;
28 server.listen(port, function () {
29     console.log("Server listening on port " + port);
30 });
31
```

0 0 — Ln 1, Col 1 — UTF-8 — LF — JavaScript



Example 4

URL routing

QuickTime Player File Edit View Window Help — 26% — Sat 19:26 — Claudia Hauff

routing.js - Example3

EXPLORE

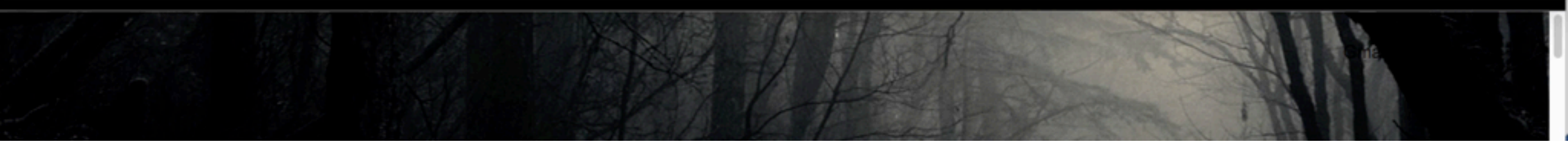
- WORKING FILES
- EXAMPLE3
  - .vscode
    - launch.json
  - jsconfig.json
  - routing.js

routing.js

```
2     url = require('url'),
3     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, {
9             "Content-Type": "text/plain"
10        });
11        var query = url_parts.query;
12        if (query["name"] !== undefined) {
13            res.end("Greetings " + query["name"]);
14        }
15        else {
16            res.end("Greetings Anonymous");
17        }
18    }
19    else {
20        res.writeHead(404, {
21            "Content-Type": "text/plain"
30 });
31
```

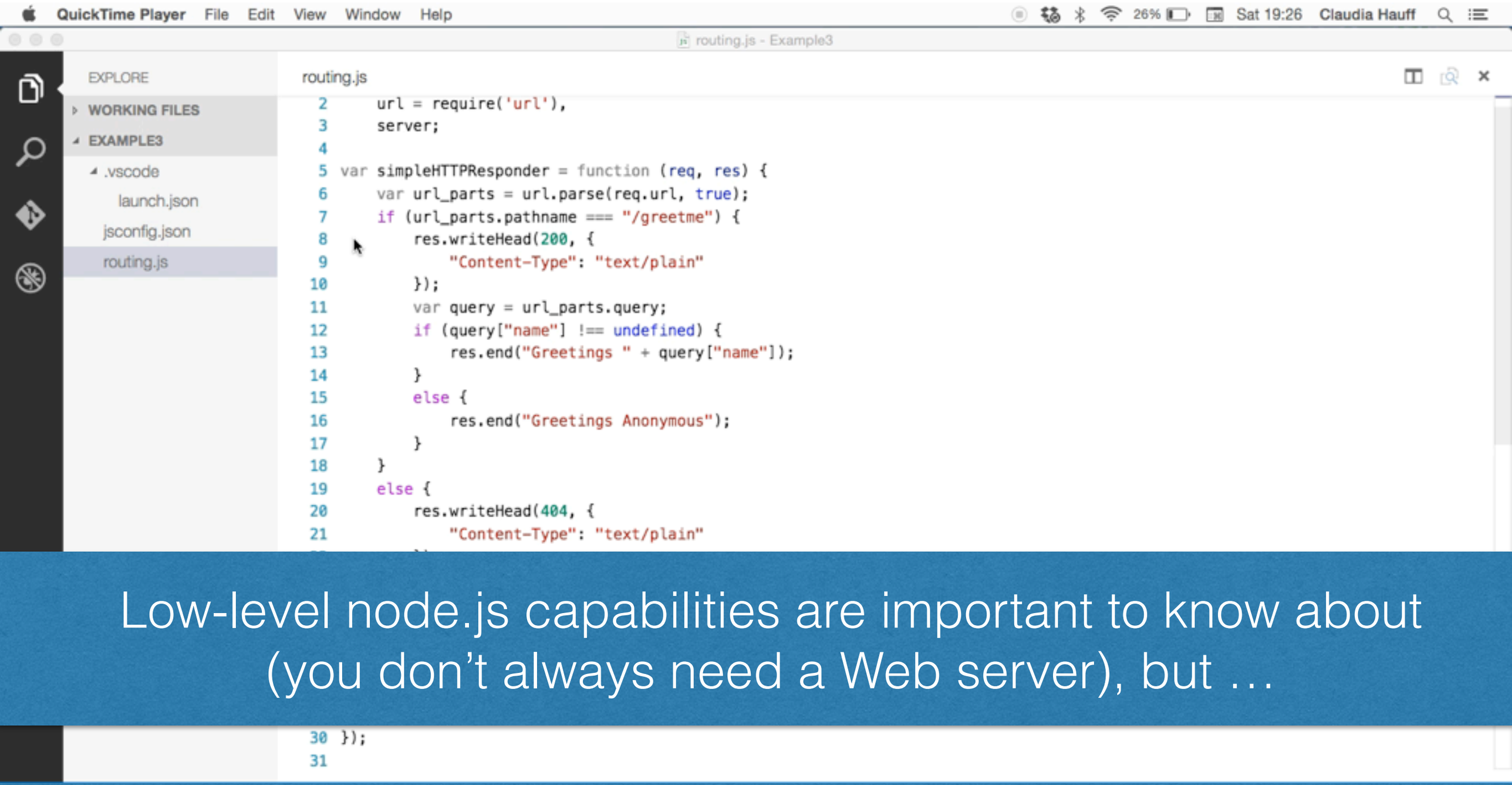
Low-level node.js capabilities are important to know about (you don't always need a Web server), but …

0 0 — Ln 1, Col 1 — UTF-8 — LF — JavaScript



Example 4

URL routing



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?

0:35

Executing this node.js code yields which console output?

```
1  var fs = require('fs');
2  var n;
3
4  function f() {
5    // asynchronous
6    fs.readFile('n.txt', function(err, content) {
7      n = parseInt(content); //n.txt contains 42
8      n++;
9    })
10 }
11
12 f();
13 console.log(n);
```

0:35

Executing this node.js code yields which console output?

```
1  var fs = require('fs');
2  var n;
3
4  function f() {
5    // asynchronous
6    fs.readFile('n.txt', function(err, content) {
7      n = parseInt(content); //n.txt contains 42
8      n++;
9    })
10 }
11
12 f();
13 console.log(n);
```

0:35

Executing this node.js code yields which console output?

```
1  var fs = require('fs');
2
3  function f(done) {
4      fs.readFile('n.txt', function(err,
5                  fileContents) {
6          var n = parseInt(fileContents); //n.txt
7                                          //contains 42
8          n++;
9          done(n);
10     });
11 }
12
13 f(function(n){
14     console.log(n);
15 });
```

0:35

Executing this node.js code yields which console output?

```
1  var fs = require('fs');
2
3  function f(done) {
4      fs.readFile('n.txt', function(err,
5                  fileContents) {
6          var n = parseInt(fileContents); //n.txt
7                                          //contains 42
8          n++;
9          done(n);
10     });
11 }
12
13 f(function(n){
14     console.log(n);
15 });
```

0:35

Executing this node.js code yields which console output?

```
1 const fs = require('fs');
2 var file1 = "file1";//assume the file exists
3 var file2 = "file2";//assume the file exists
4
5 var exists1;
6 var exists2;
7
8 fs.exists(file1, function (exists) {
9  if (exists)
10     exists1 = true;
11 });
12
13 fs.exists(file2, function (exists) {
14  if(exists)
15     exists2 = true;
16 });
17
18 console.log(exists1+"/"+exists2);
```

`fs.exists()` is asynchronous; it tests whether the given file exists; callback argument contains either `true` (exists) or `false` (does not exist)

0:35

Executing this node.js code yields which console output?

```
1 const fs = require('fs');
2 var file1 = "file1";//assume the file exists
3 var file2 = "file2";//assume the file exists
4
5 var exists1;
6 var exists2;
7
8 fs.exists(file1, function (exists) {
9  if (exists)
10     exists1 = true;
11 });
12
13 fs.exists(file2, function (exists) {
14  if(exists)
15     exists2 = true;
16 });
17
18 console.log(exists1+"/"+exists2);
```

`fs.exists()` is asynchronous; it tests whether the given file exists; callback argument contains either `true` (exists) or `false` (does not exist)

0:35

Executing this node.js code yields which console output?

```
1 const fs = require('fs');
2 var file1 = "file1";//assume the file exists
3 var file2 = "file2";//assume the file exists
4
5 var exists1;
6 var exists2;
7
8 if(fs.existsSync(file1)){
9  exists1 = true;
10 };
11
12 if(fs.existsSync(file2)) {
13  exists2 = true;
14 };
15
16 console.log(exists1+"/"+exists2);
```

`fs.existsSync()`: synchronous version of `fs.exists()`. Returns `true` if the file exists, `false` otherwise.

0:35

Executing this node.js code yields which console output?

```
1 const fs = require('fs');
2 var file1 = "file1";//assume the file exists
3 var file2 = "file2";//assume the file exists
4
5 var exists1;
6 var exists2;
7
8 if(fs.existsSync(file1)){
9  exists1 = true;
10 };
11
12 if(fs.existsSync(file2)) {
13  exists2 = true;
14 };
15
16 console.log(exists1+"/"+exists2);
```

`fs.existsSync()`: synchronous version of `fs.exists()`. Returns `true` if the file exists, `false` otherwise.

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

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});
```

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});
```

app object is our way to use Express’ abilities

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

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

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 });
```

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,
11   var query = url.parse(req.url, tr
12   var name = ( query["name"]!=undef
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

```
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
```

Development strategy

- Develop the **client-side code** (HTML, CSS, JavaScript)

```
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
```

Development strategy

- Develop the **client-side code** (HTML, CSS, JavaScript)
- Place all files into some directory (e.g. `/client`) **on the server**

```
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
```

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 a *.js file using Express

```
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
```

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 a *.js file using Express
- Set **Express' static file path** to the directory of step 2

```
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
```

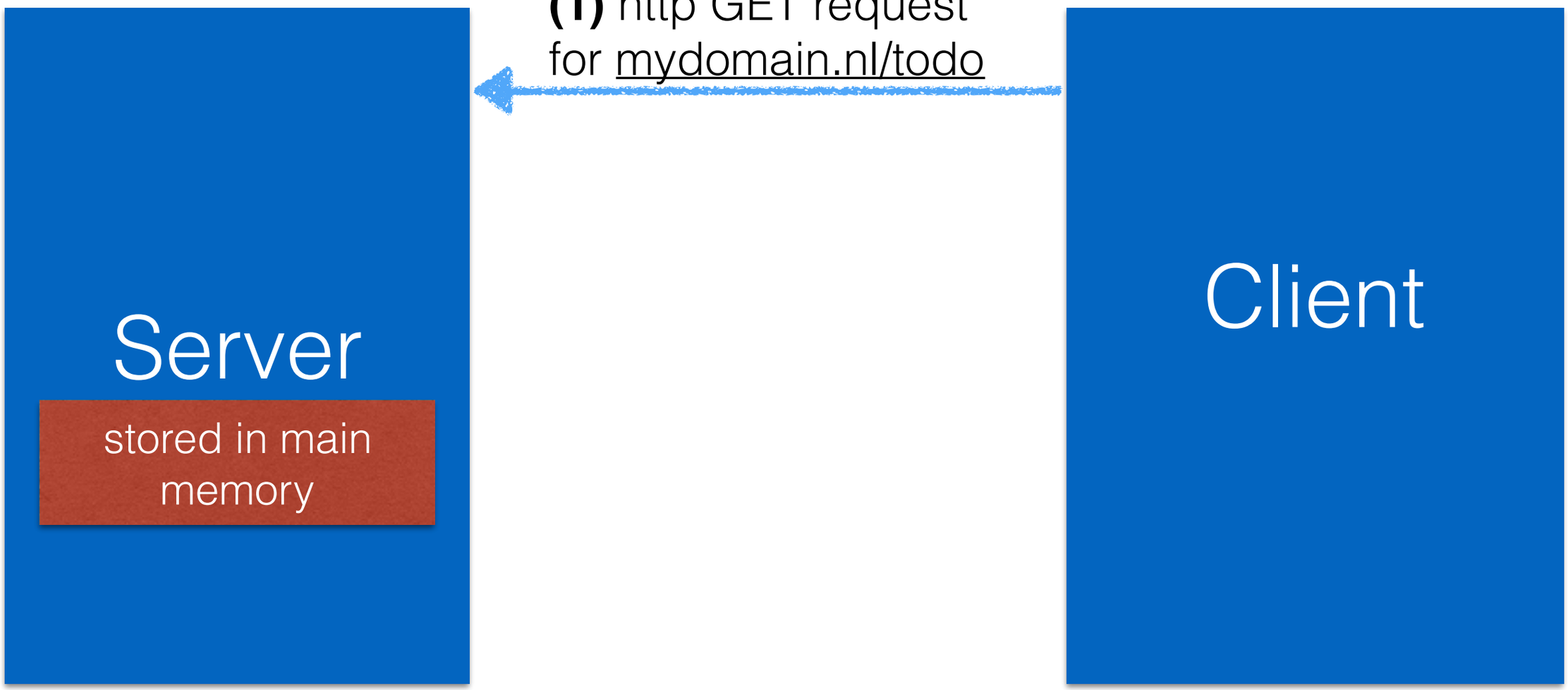
Development strategy

- 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 a *.js file using Express
- Set **Express' static file path** to the directory of step 2
- Add interactivity between client and server via **Ajax** and **JSON**

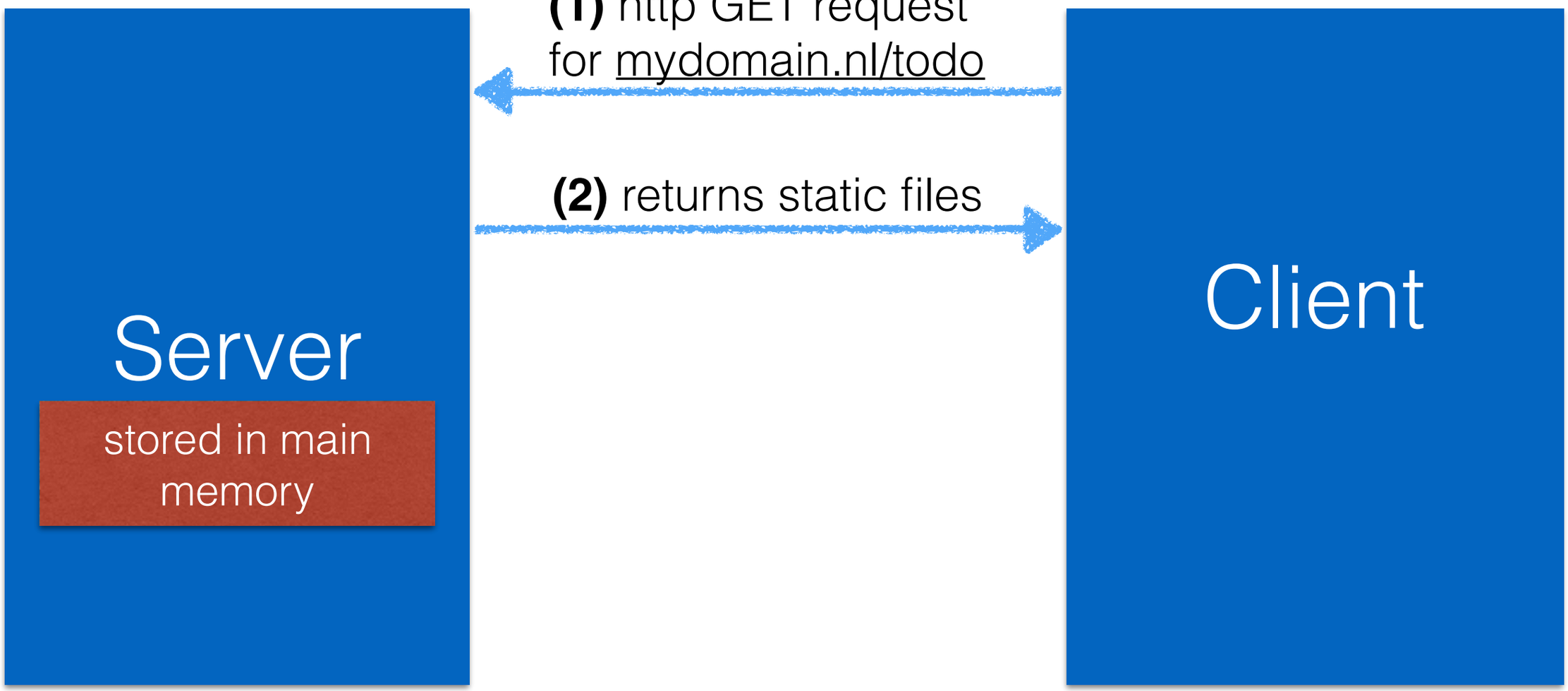
```
todo-server.js
client/
    index.html
    html/
        =>error.html
        =>addtodo.html
images/
    =>background.png
    =>logo.png
css/
    =>layout.css
    =>style.css
javascript/
    =>client-app.js
```

TODO Web app flow

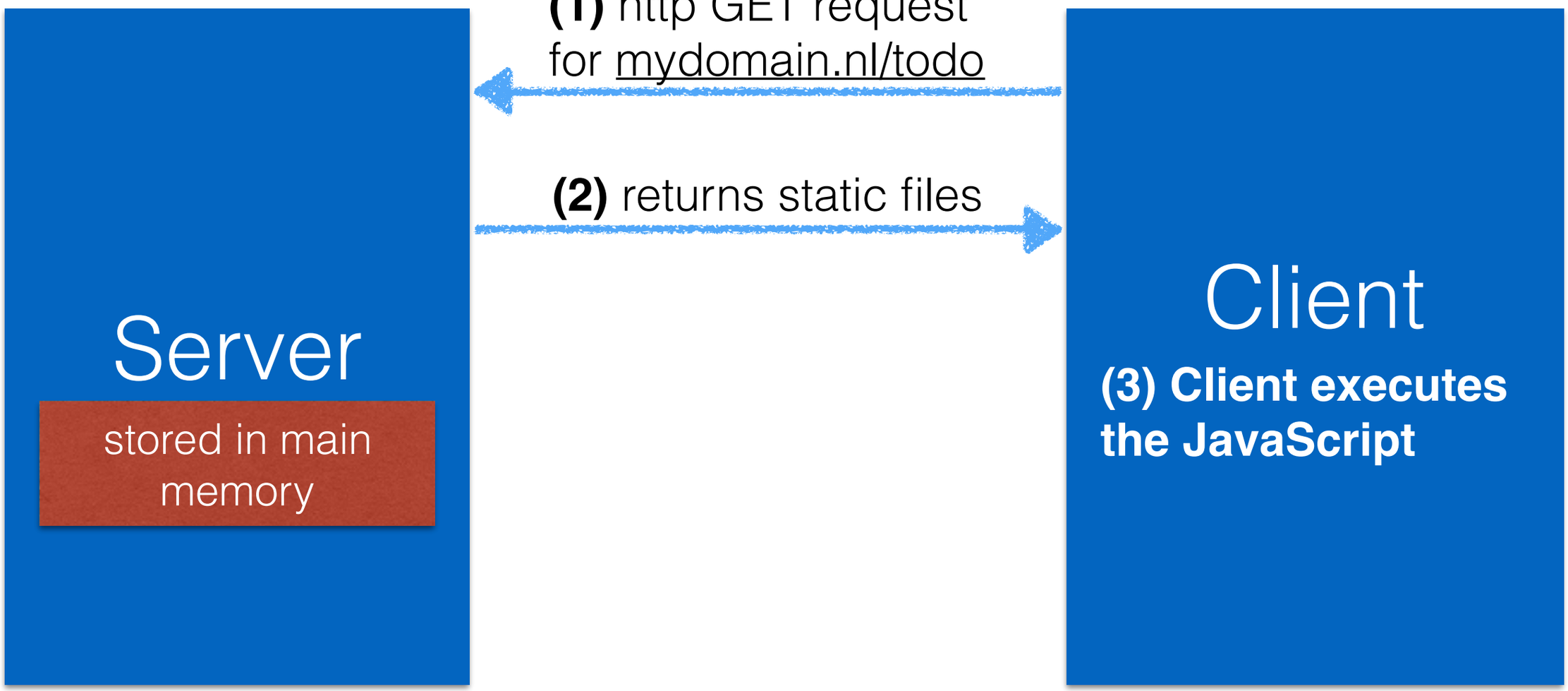
TODO Web app flow



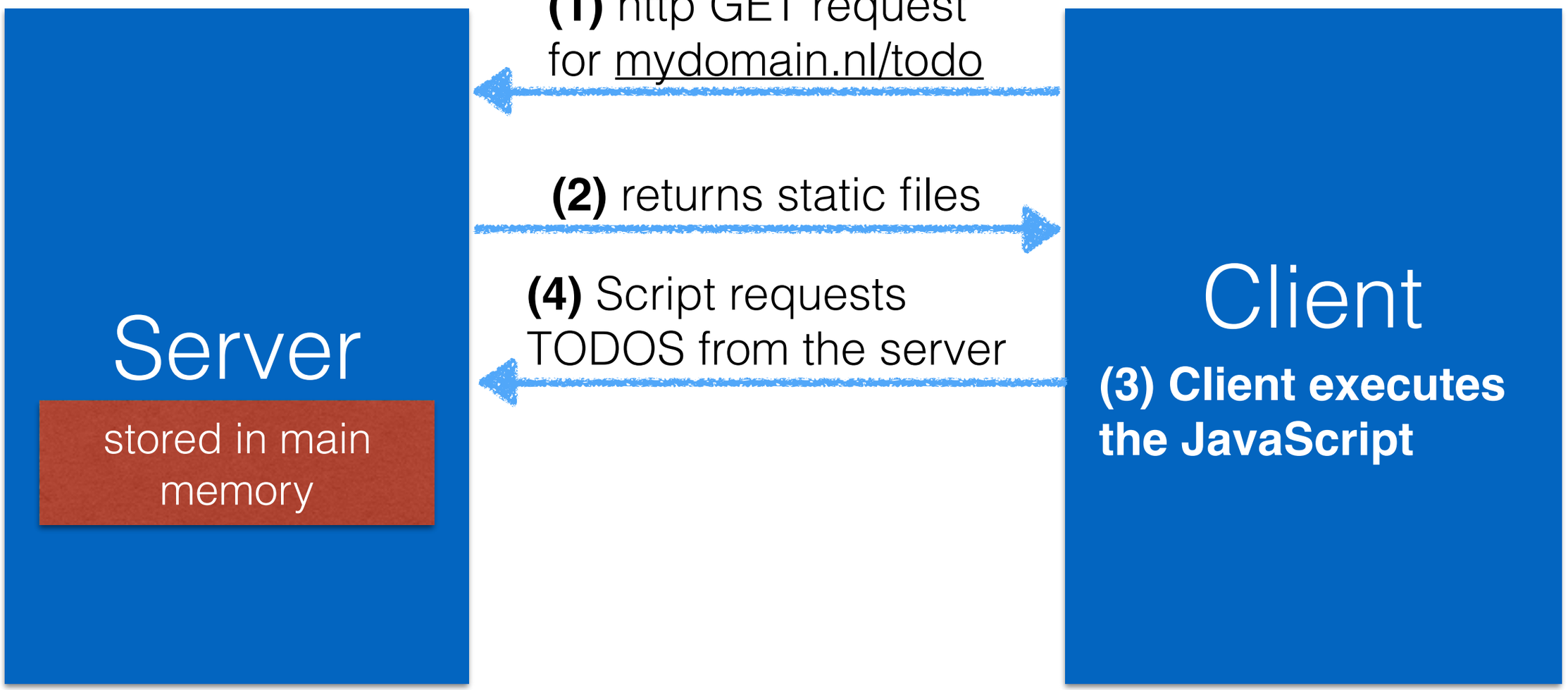
TODO Web app flow



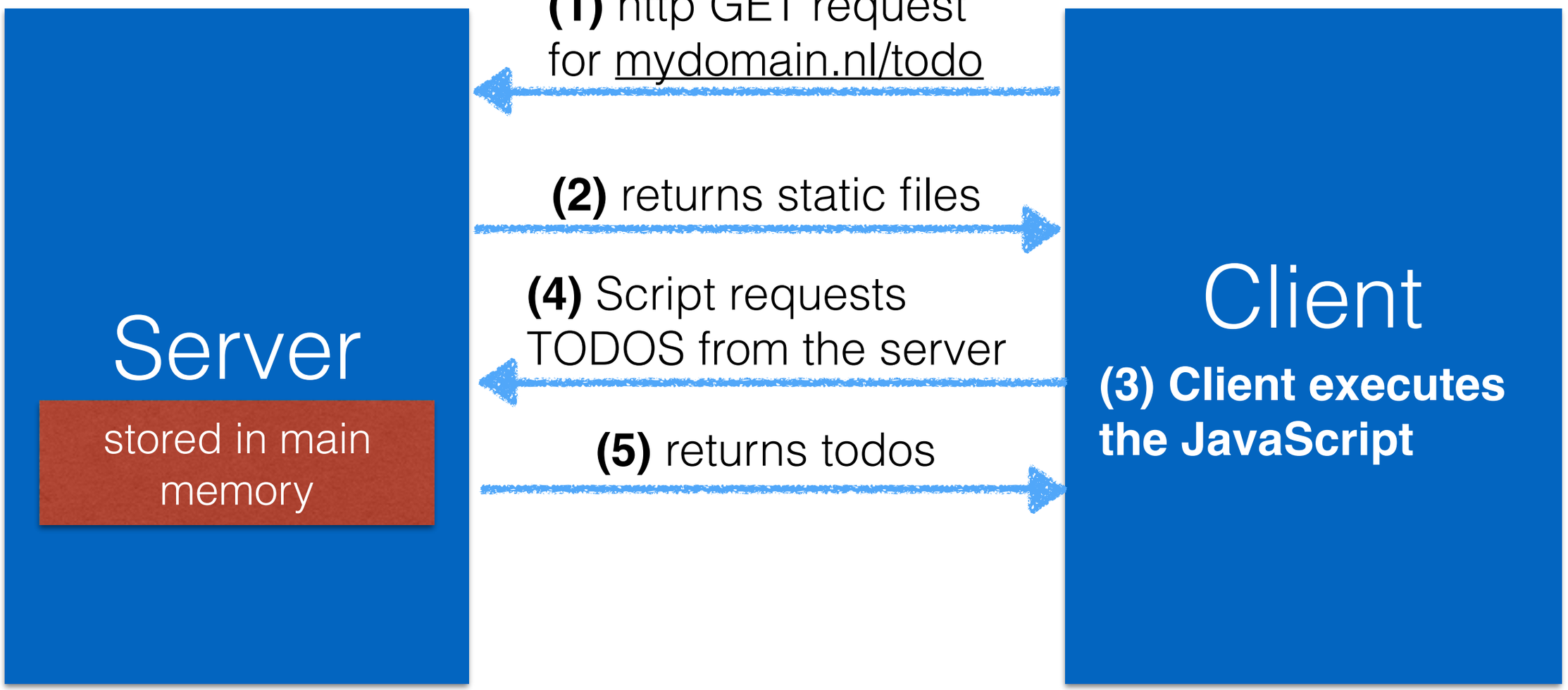
TODO Web app flow



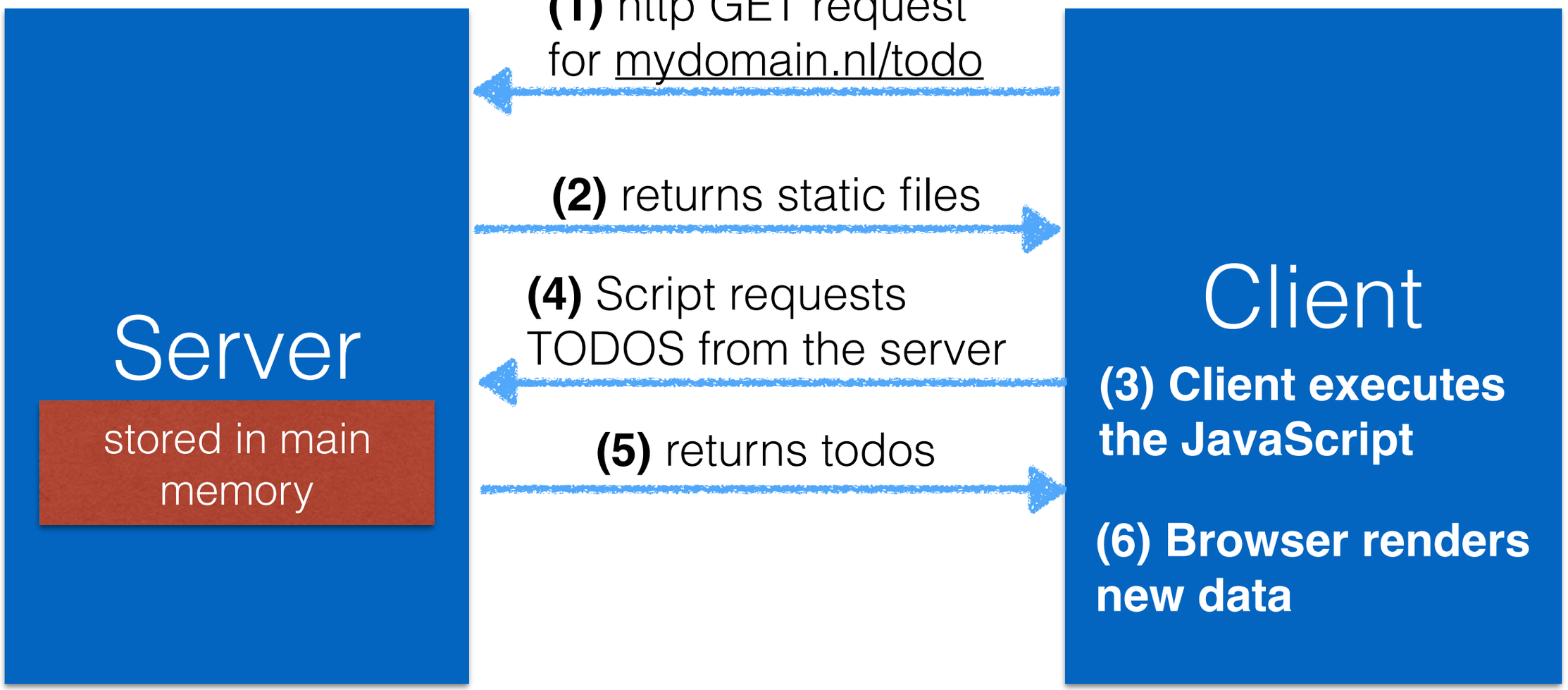
TODO Web app flow



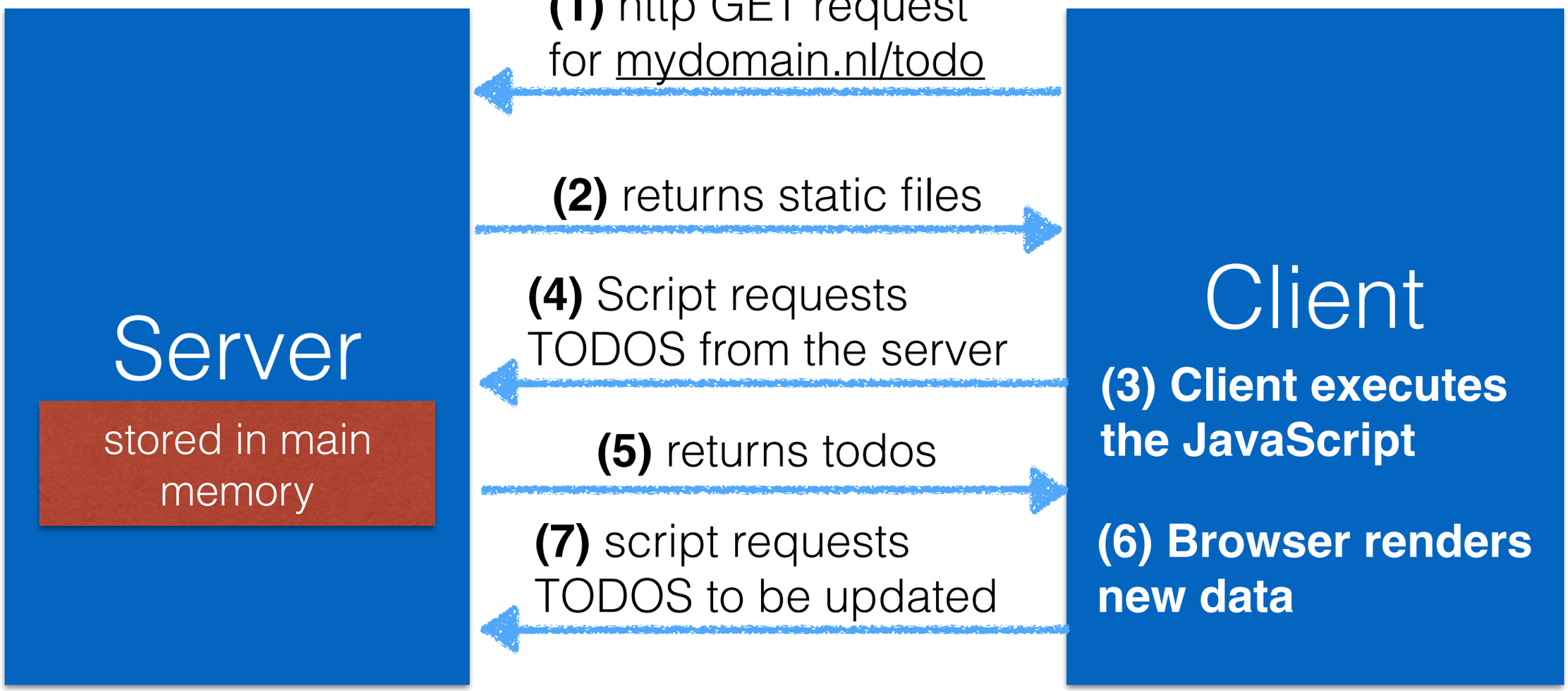
TODO Web app flow



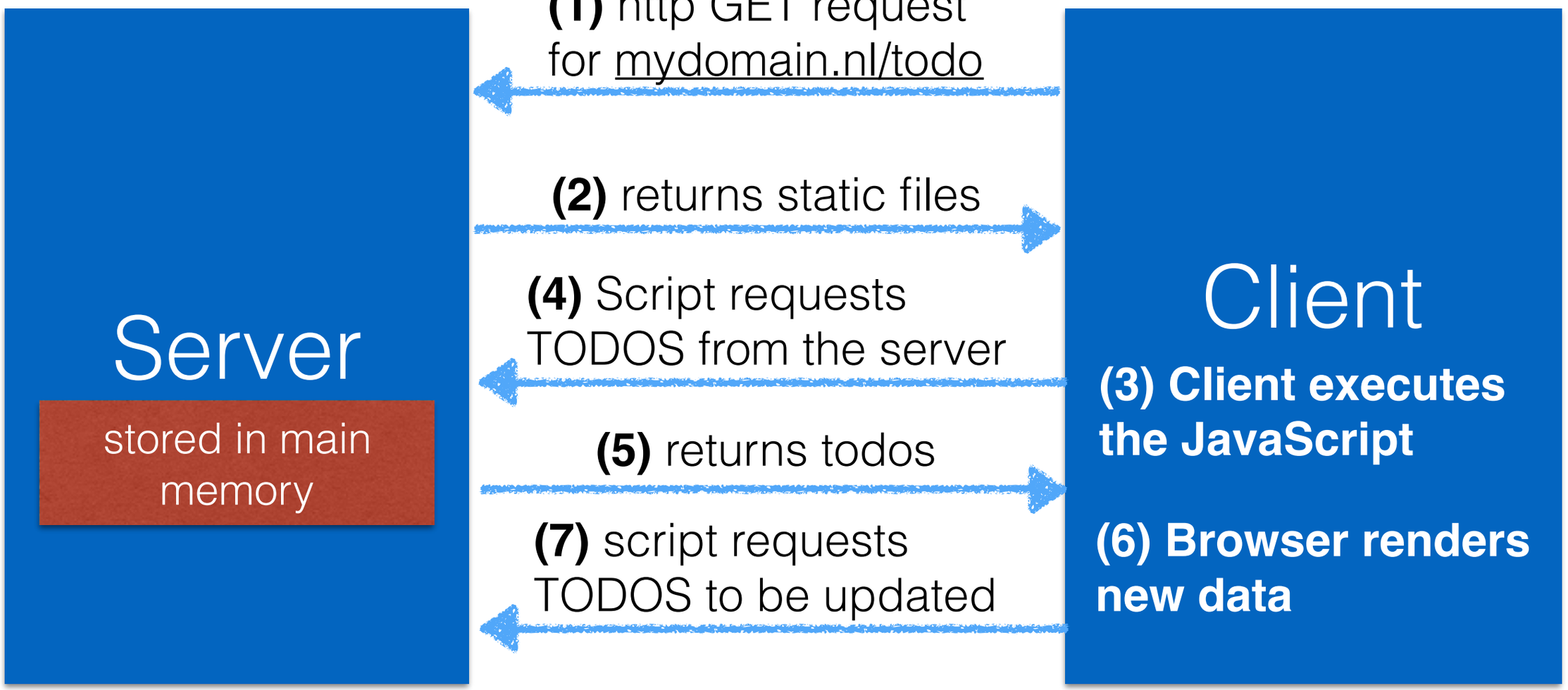
TODO Web app flow



TODO Web app flow



TODO Web app flow



stored in a database (next time)

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 });
```

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

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 });
```

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 });
```

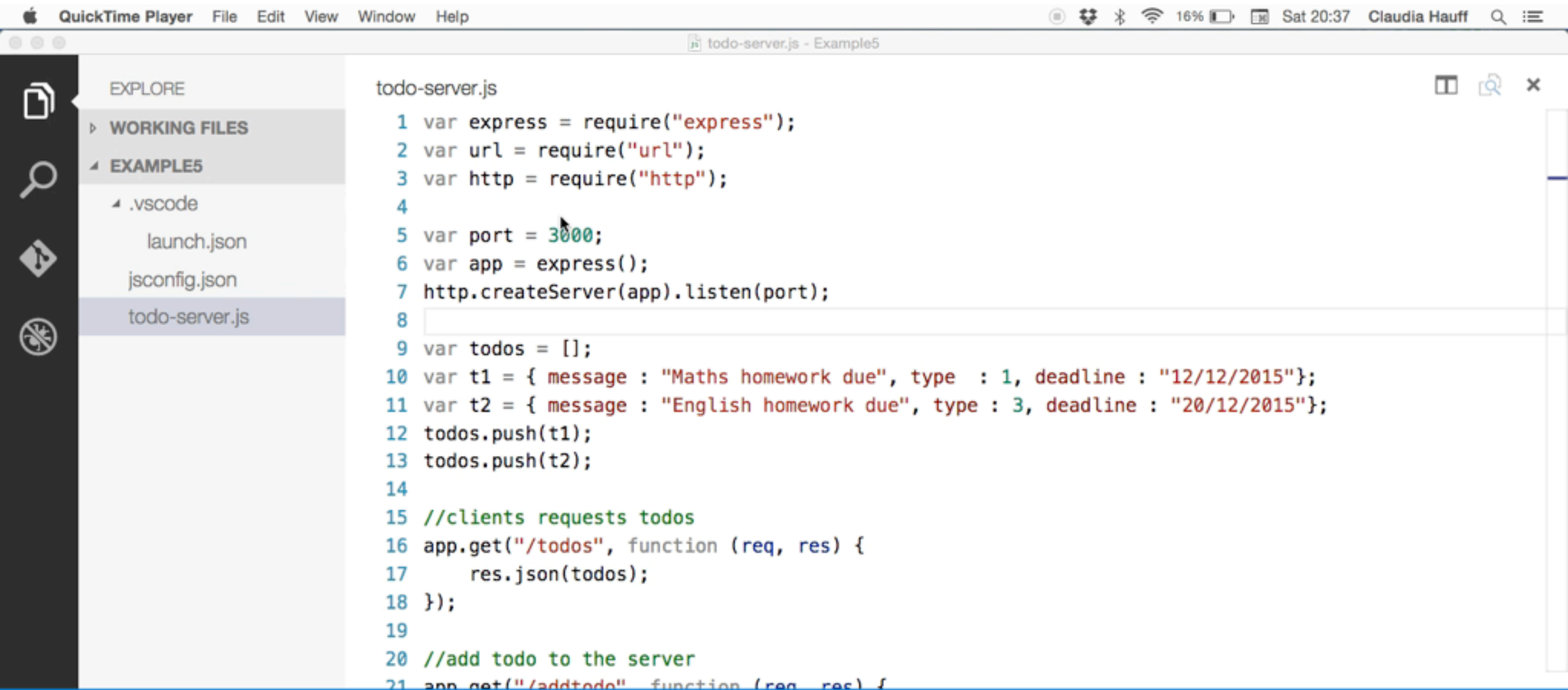
On the server: todo updating

```
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 });
```

is message a key in the **query**?

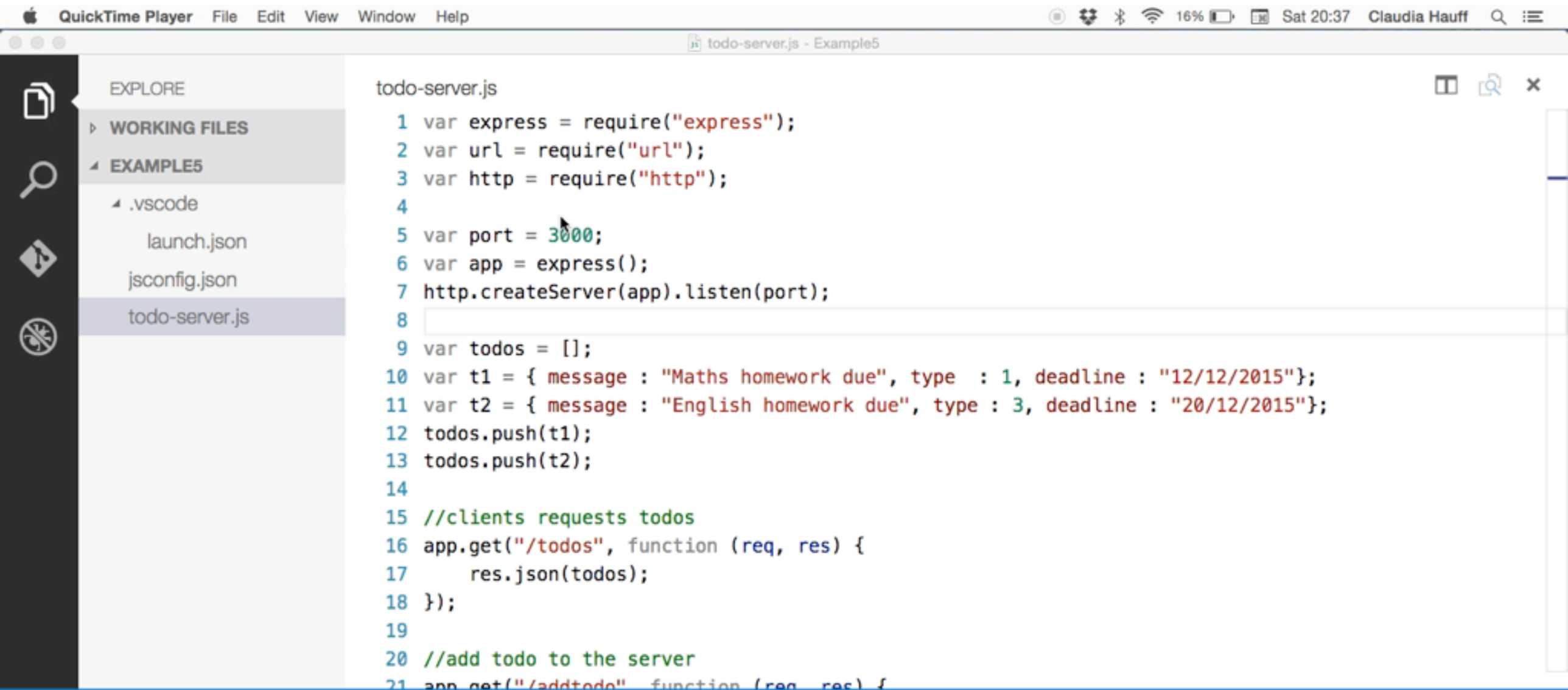
Example 5

Sending json & todo updating



Example 5

Sending json & todo updating



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

0:35

How does an Ajax request appear to a Web server?

A. An Ajax request looks like any other http request.

B. An Ajax request is not sent via http, but instead via atp, the *Ajax transfer protocol*.

C. An Ajax request always has to be sent as part of an HTML `<form>`.

D. An Ajax request is never sent to a Web server, the reply is generated by the browser cache.

0:35

How does an Ajax request appear to a Web server?

A. An Ajax request looks like any other http request.

B. An Ajax request is not sent via http, but instead via atp, the *Ajax transfer protocol*.

C. An Ajax request always has to be sent as part of an HTML `<form>`.

D. An Ajax request is never sent to a Web server, the reply is generated by the browser cache.

0:35

Imagine building a chat application using Ajax. How is the browser notified of new messages to display?

A. Ajax allows the server to push http responses to the client.
B. The browser has to poll the server for message updates in short time intervals.

0:35

Imagine building a chat application using Ajax. How is the browser notified of new messages to display?

A. Ajax allows the server to push http responses to the client.
B. The browser has to poll the server for message updates in short time intervals.

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>
```

On the client: basic HTML

jQuery way

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

```
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>
```

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

```
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);
```

when the document is loaded, execute main()

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);
```

when the document is loaded, execute main()

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

```
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);
```

Callback: define what happens when a todo object is available

Dynamic insert of list elements into the DOM

this is Ajax

when the document is loaded, execute main()

Example 6

Ajax

Code File Edit View Goto Window Help — 87% — Sat 22:27 — Claudia Hauff

index.html - Example6

EXPLORE

WORKING FILES
- index.html client
- client-app.js client/java...
- todo-server.js

EXAMPLE6
- .vscode
- client
  - html
  - javascript
    - client-app.js
    - index.html
  - jsconfig.json
  - todo-server.js

index.html client

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

2 0 — Ln 9, Col 1 — UTF-8 — LF — HTML

Example 6

Ajax

Code File Edit View Goto Window Help — 87% — Sat 22:27 — Claudia Hauff

index.html - Example6

EXPLORE

WORKING FILES
- index.html client
- client-app.js client/java...
- todo-server.js

EXAMPLE6
- .vscode
- client
  - html
  - javascript
    - client-app.js
    - index.html
  - jsconfig.json
  - todo-server.js

index.html client

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

2 0 — Ln 9, Col 1 — UTF-8 — LF — HTML

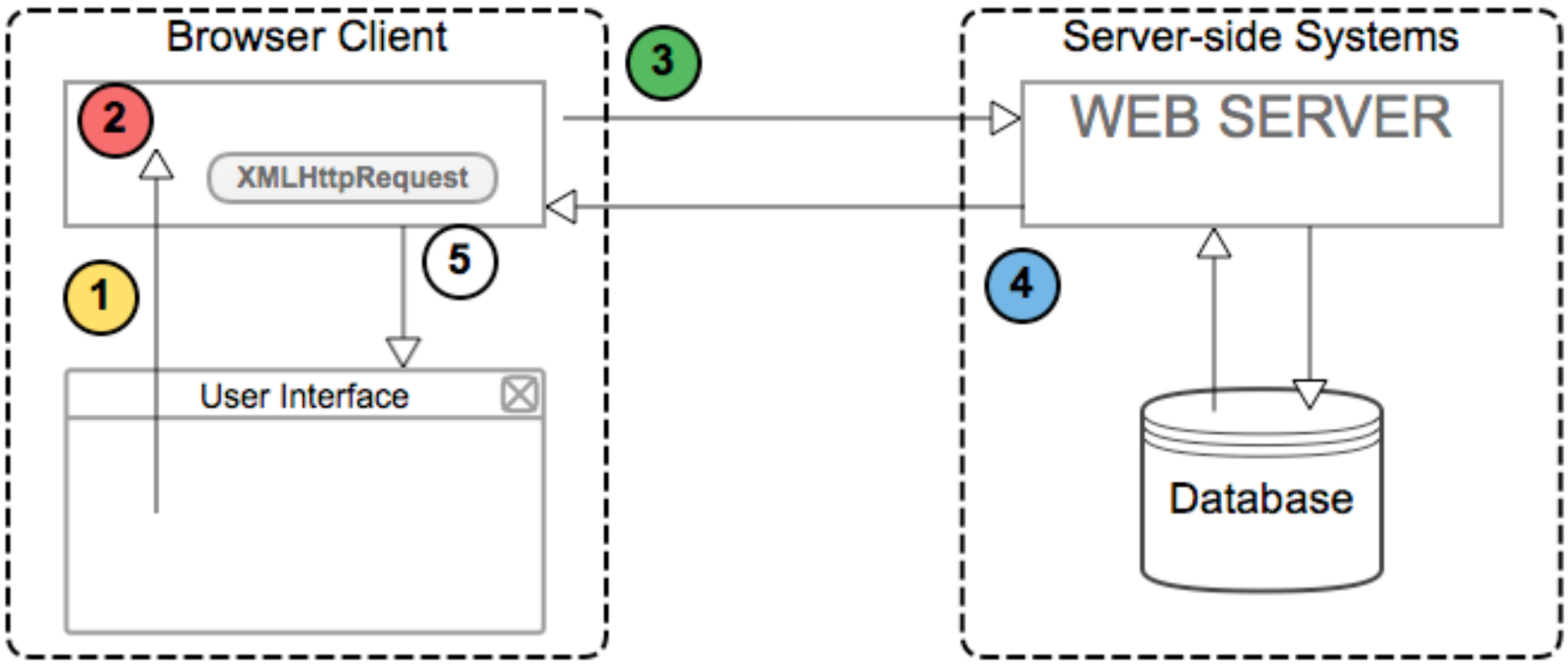
flickr@124247024@N07

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 …

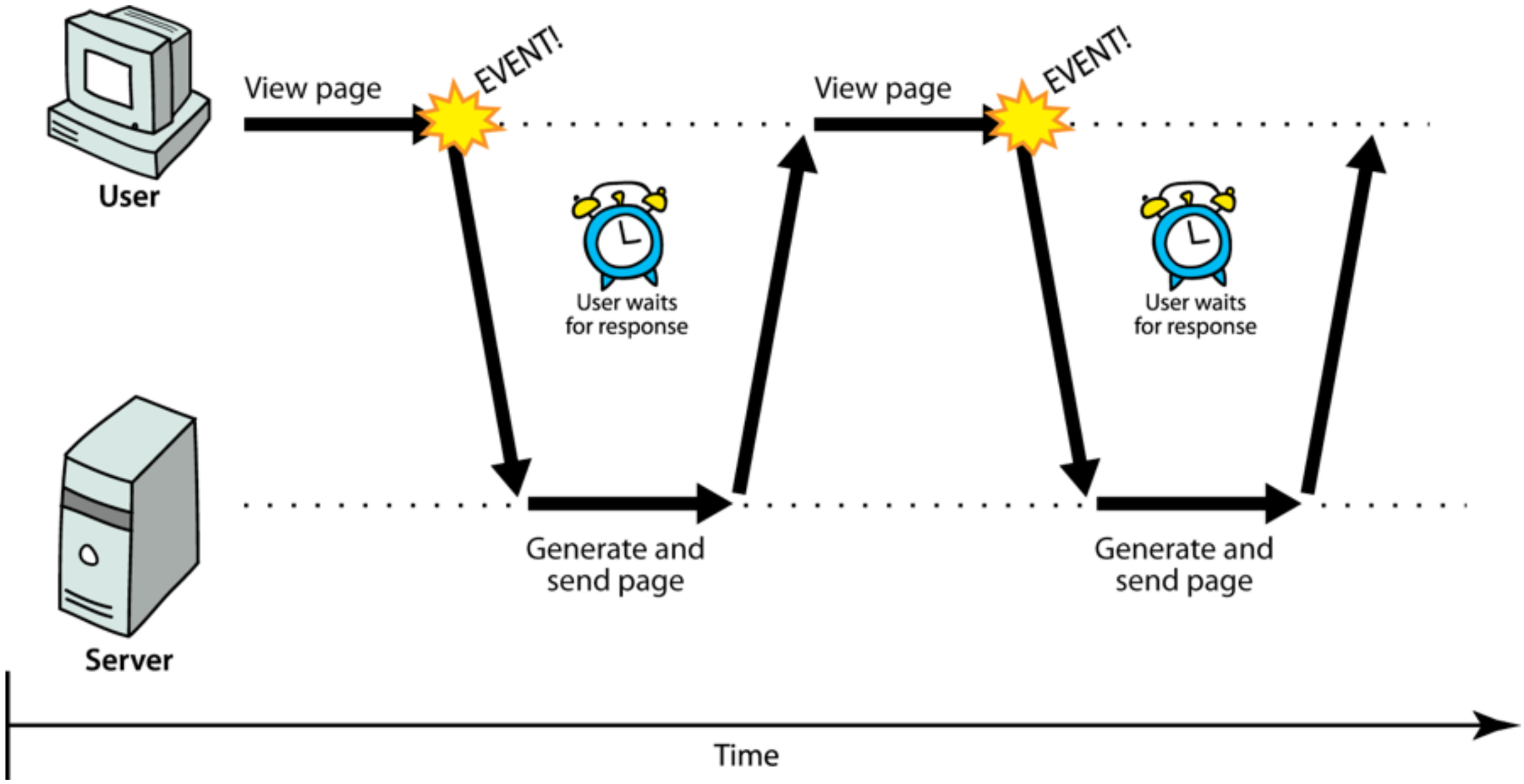


Image source: Web Programming Step by Step

Ajax works differently

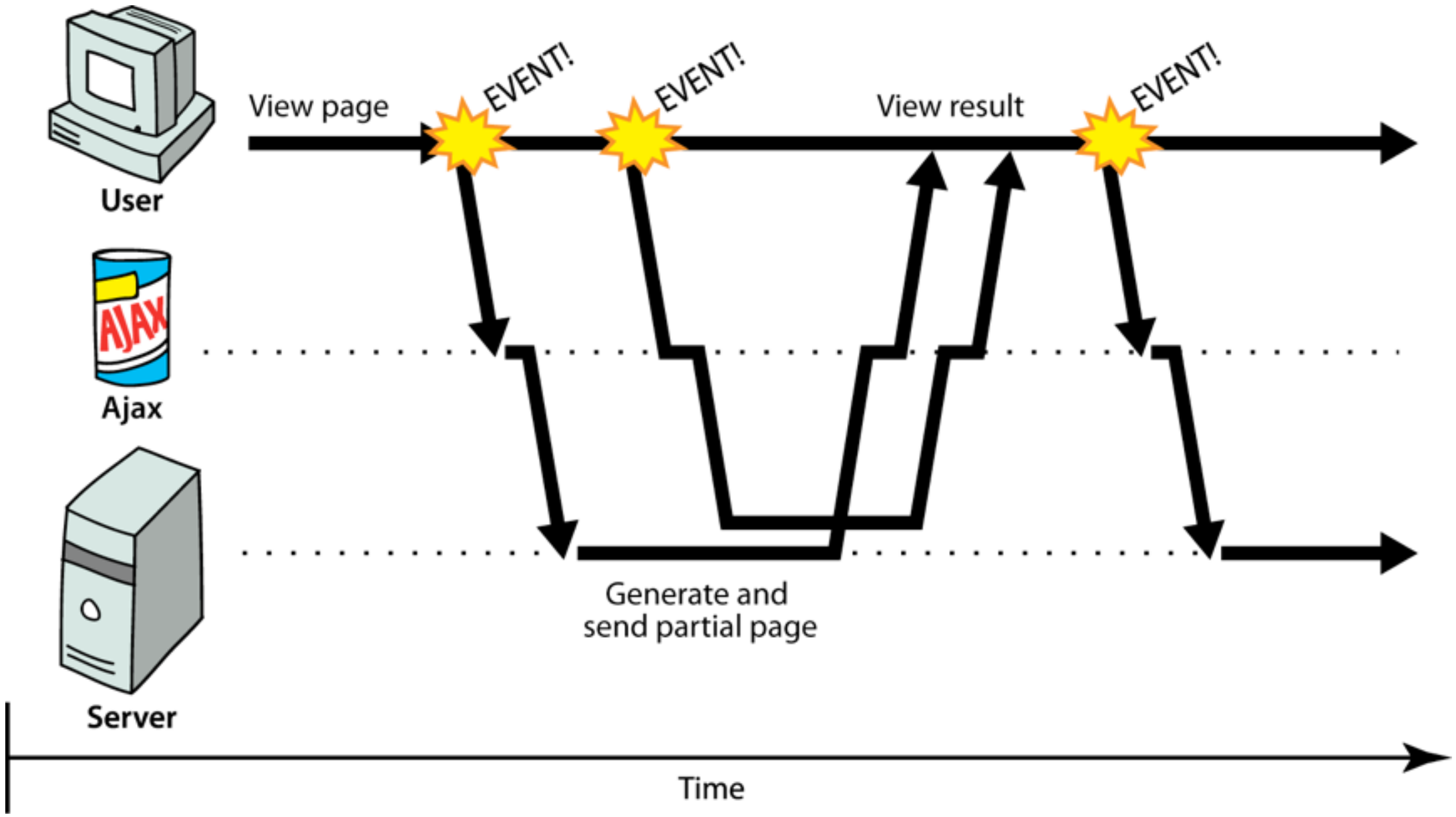


Image source: Web Programming Step by Step

Ajax: synchronous request

plain JS

```
1 //IE6 and prior IE versions use Microsoft.
2 XMLHTTP instead
3 var ajax = new XMLHttpRequest();
4
5 //retrieve data from URL (file) of interest
6 //false parameter: synchronous request
7 ajax.open('GET', 'example.txt', false);
8 ajax.send(null);
9
10 //response data in ajax.responseText
11 document.getElementById('ttExampleText').value
12                                 = ajax.responseText;
```

line of code is executed after line 7/8 are executed.

Ajax: an asynchronous request

plain JS

```
1  var ajax = new XMLHttpRequest();
2
3  //function to be called when the data has
4  arrived
5  ajax.onreadystatechange = function() {
6
7     //the only state we care about
8     if(ajax.readyState==4) {
9        /*
10        * process received data
11        */
12    }
13 }; //end of function
14
15 ajax.open("GET", "url", true); //true indicates
16                                 //asynchronous request
17
18 ajax.send(null);
```

Ajax: an asynchronous request

plain JS

```
1  var ajax = new XMLHttpRequest();
2
3  /                                    data has
4  a
5  ajax.onreadystatechange = function() {
6
7      //the only state we care about
8      if(ajax.readyState==4) {
9          /*
10          * process received data
11          */
12     }
13 }; //end of function
14
15 ajax.open("GET", "url", true); //true indicates
16                                 //asynchronous request
17
18 ajax.send(null);
```

event `onreadystatechange` is fired when the status of the request changes.

Ajax security

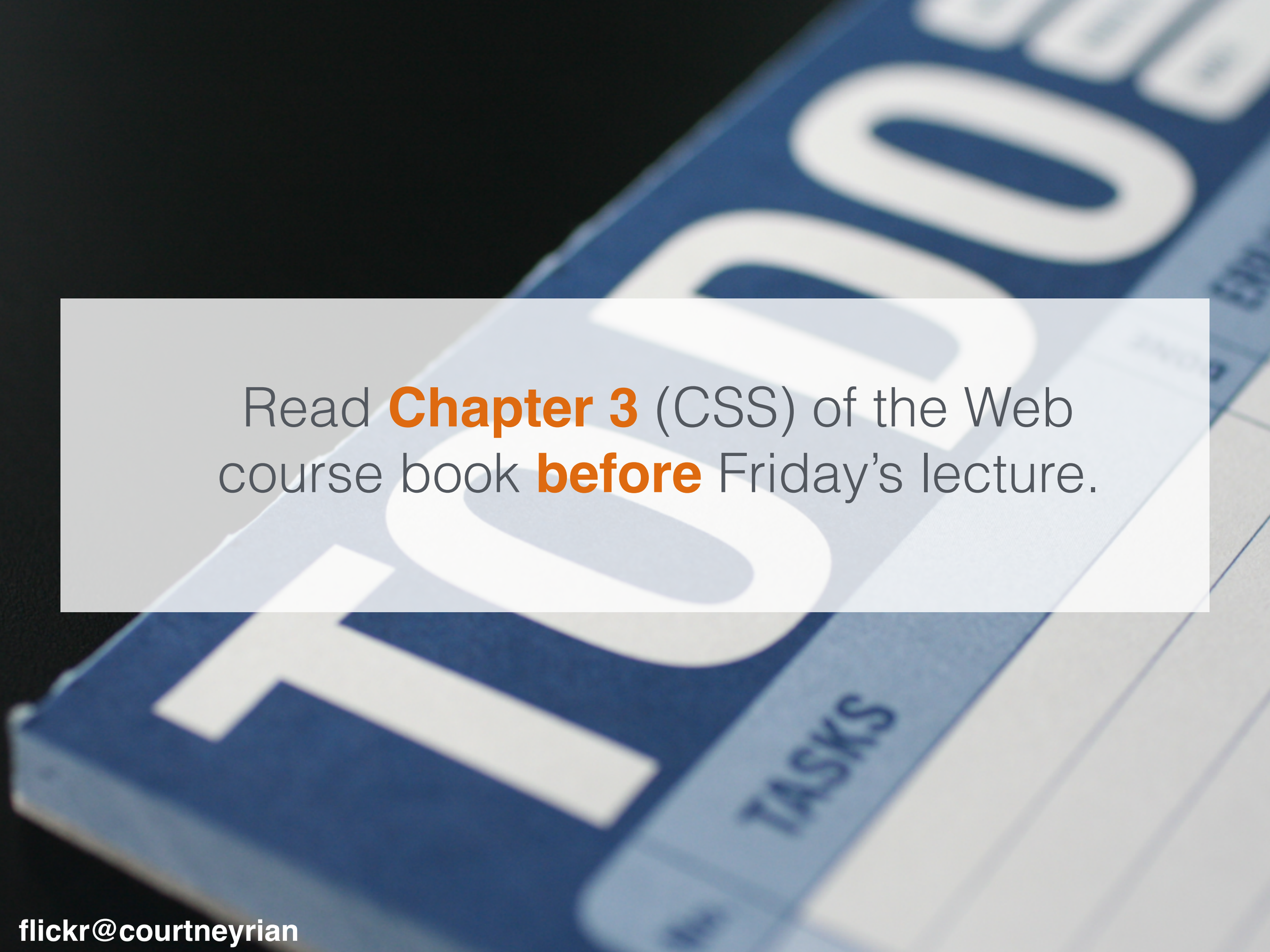
- Conveniently we always requested data from "our" Web server
- **Security restriction of Ajax**: can only fetch files from the same Web server as the calling page (*Same-origin policy*)
  - Same origin when protocol, port, and host are the same for two pages
- Ajax **cannot** be executed from a Web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

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)

Read **Chapter 3** (CSS) of the Web course book **before** Friday's lecture.



flickr@courtneyrian

End of Lecture