

node.js: JavaScript on the server

Claudia Hauff

TI1506: Web and Database Technology

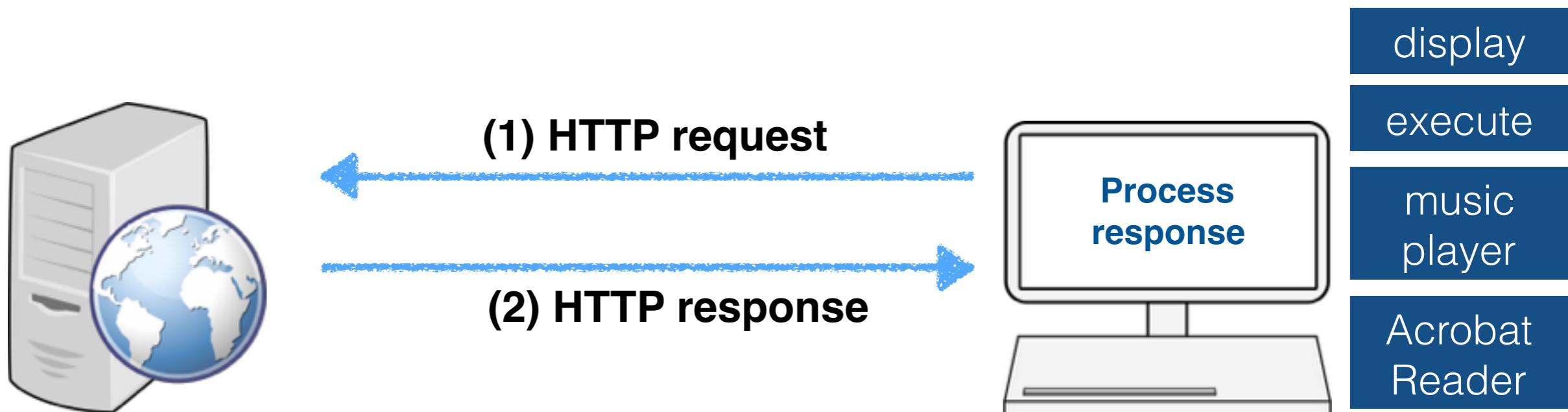
ti1506-ewi@tudelft.nl

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

A reminder before
we start

Web servers and clients



- 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

start line

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

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.

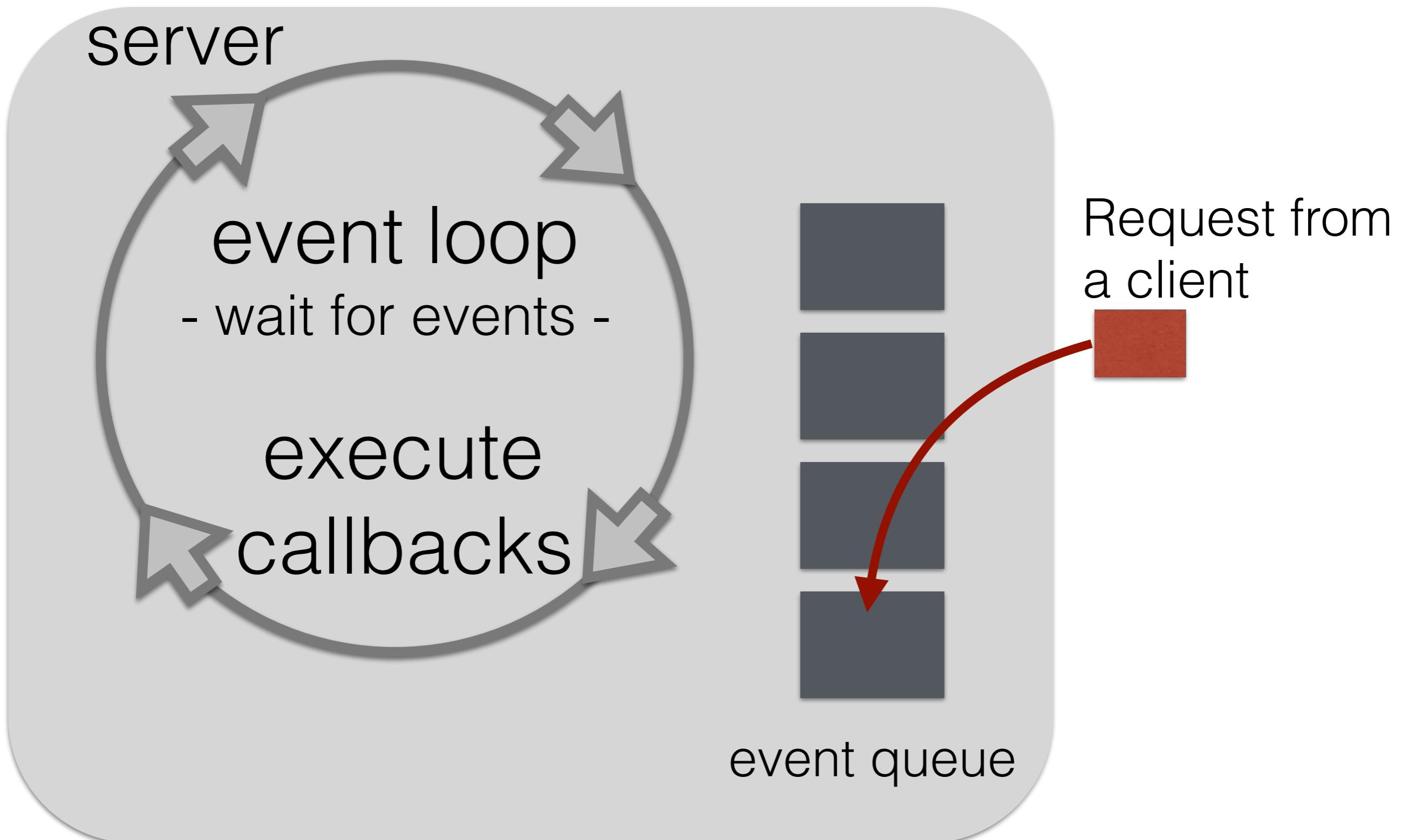
Node.js uses an **event-driven, non-blocking I/O** model that makes it lightweight and efficient, perfect for **data-intensive real-time applications** that run across distributed devices.”

History of node.js

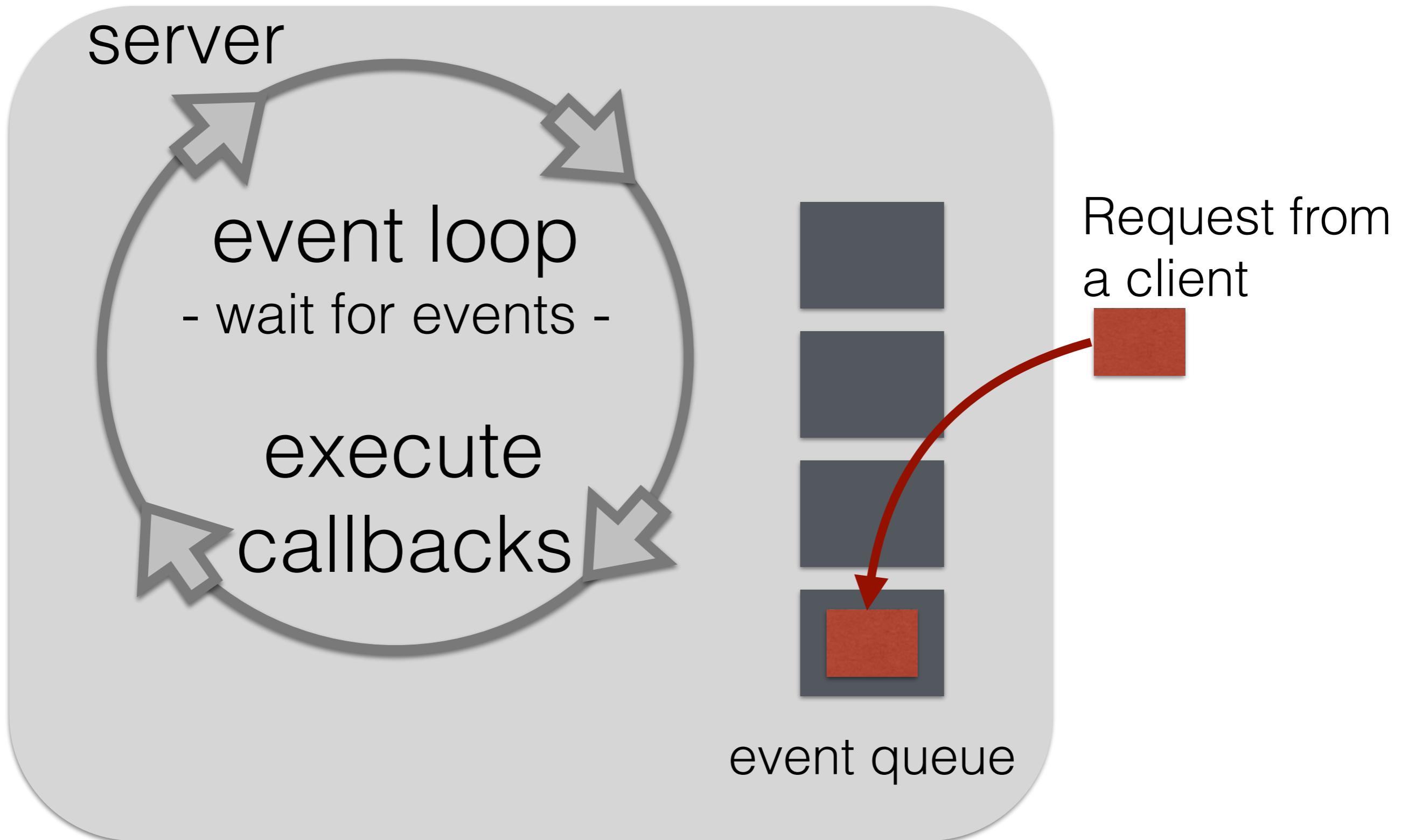
- Google's JavaScript execution engine (**V8**) was open-sourced in 2008
- node.js builds on V8 and was released in **2009**
- **node.js' package manager** (npm) was released in 2011
 - **200,000** packages [**npm**]
 - **158,000** packages [**CPAN**]
- December 2014: **io.js**
- May 2015: **io.js merges back** with node.js



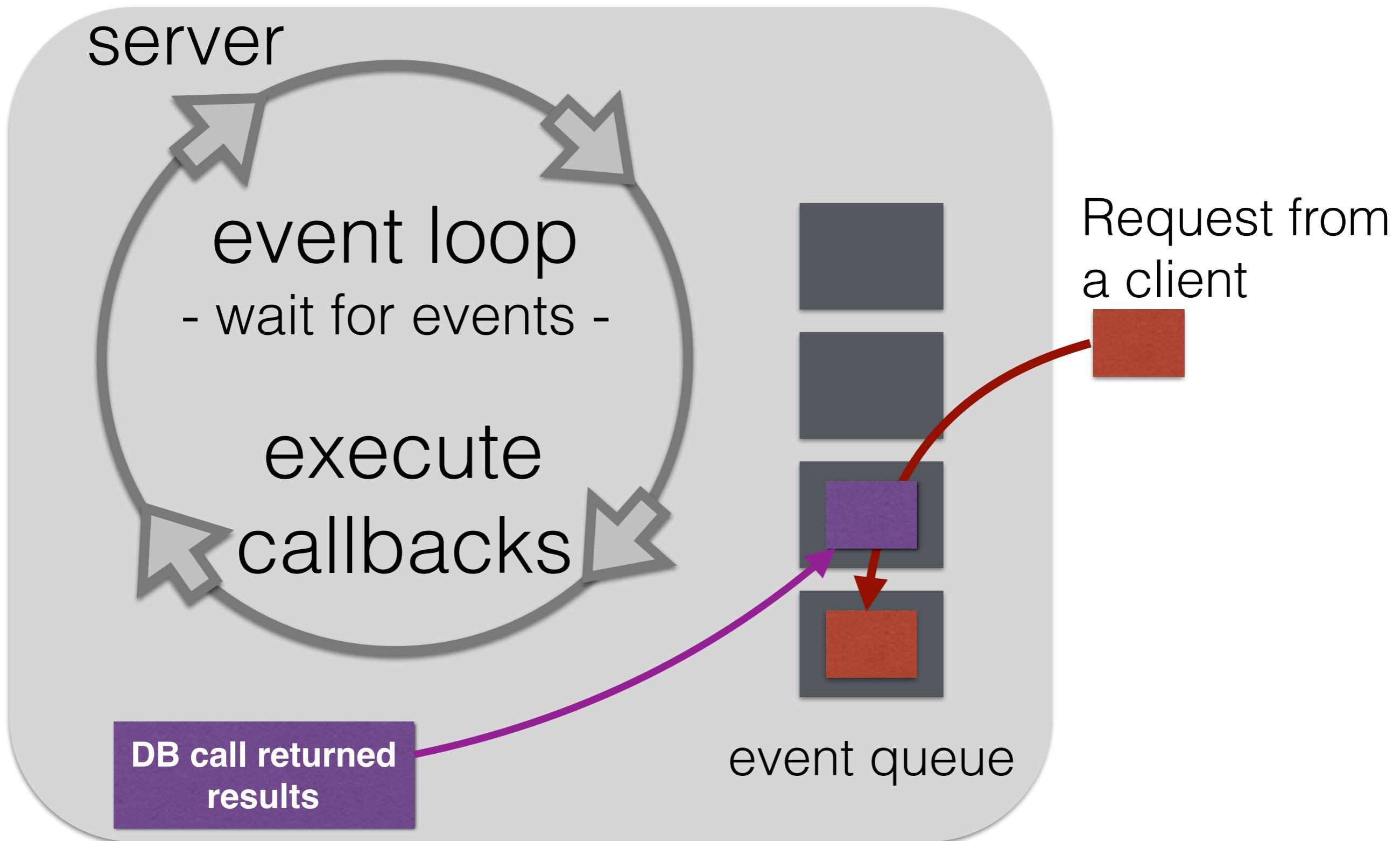
node.js is event-driven



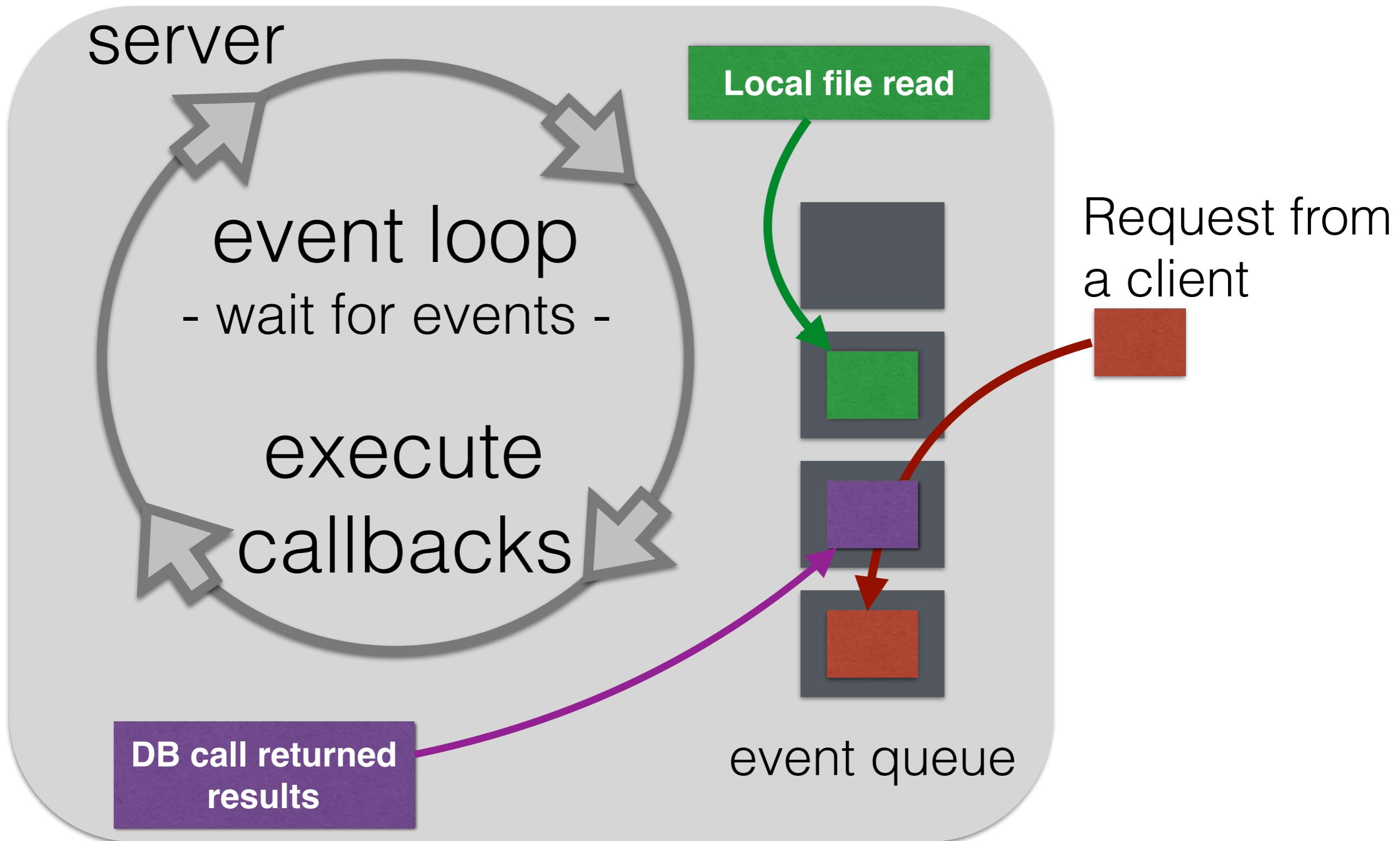
node.js is event-driven



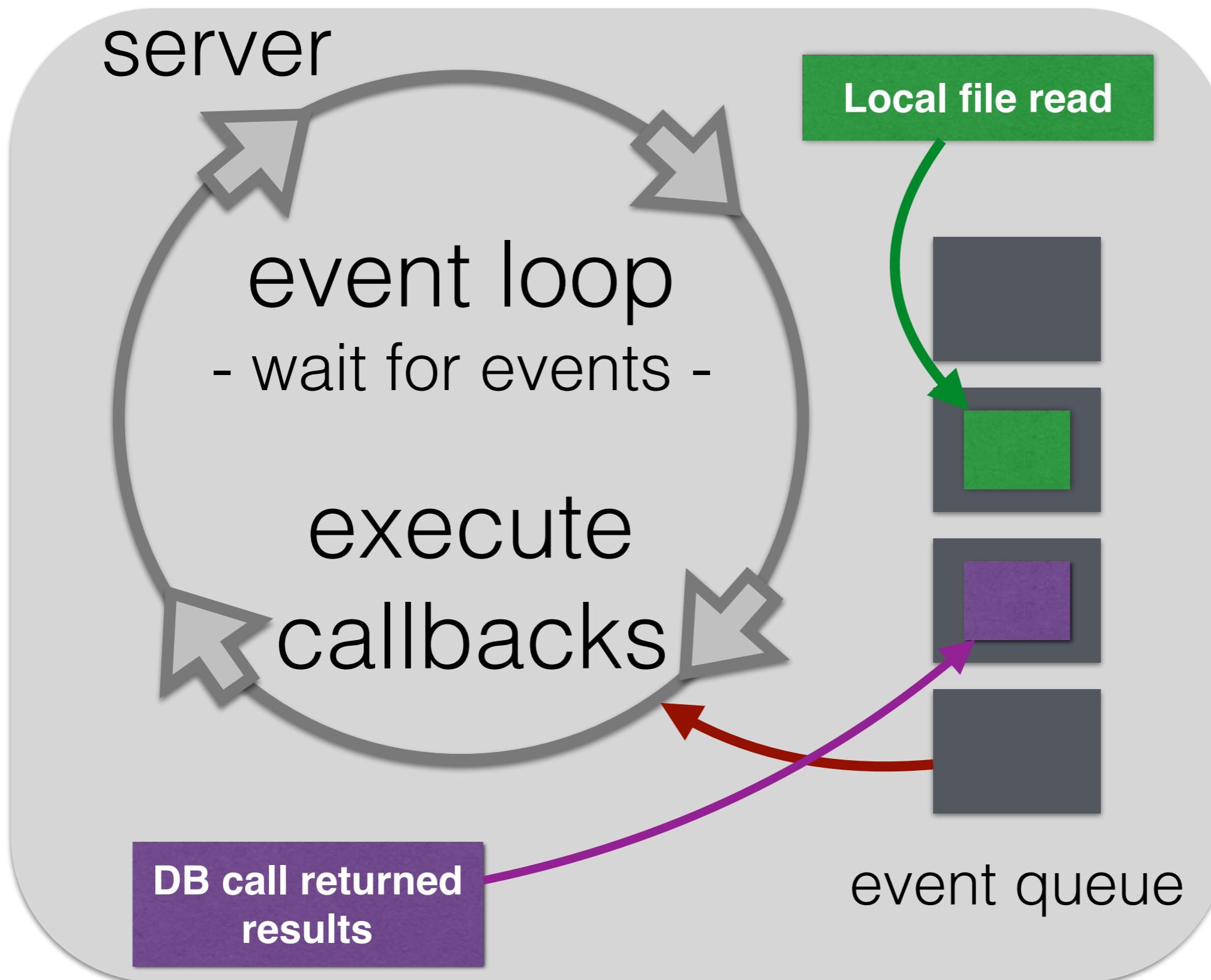
node.js is event-driven



node.js is event-driven

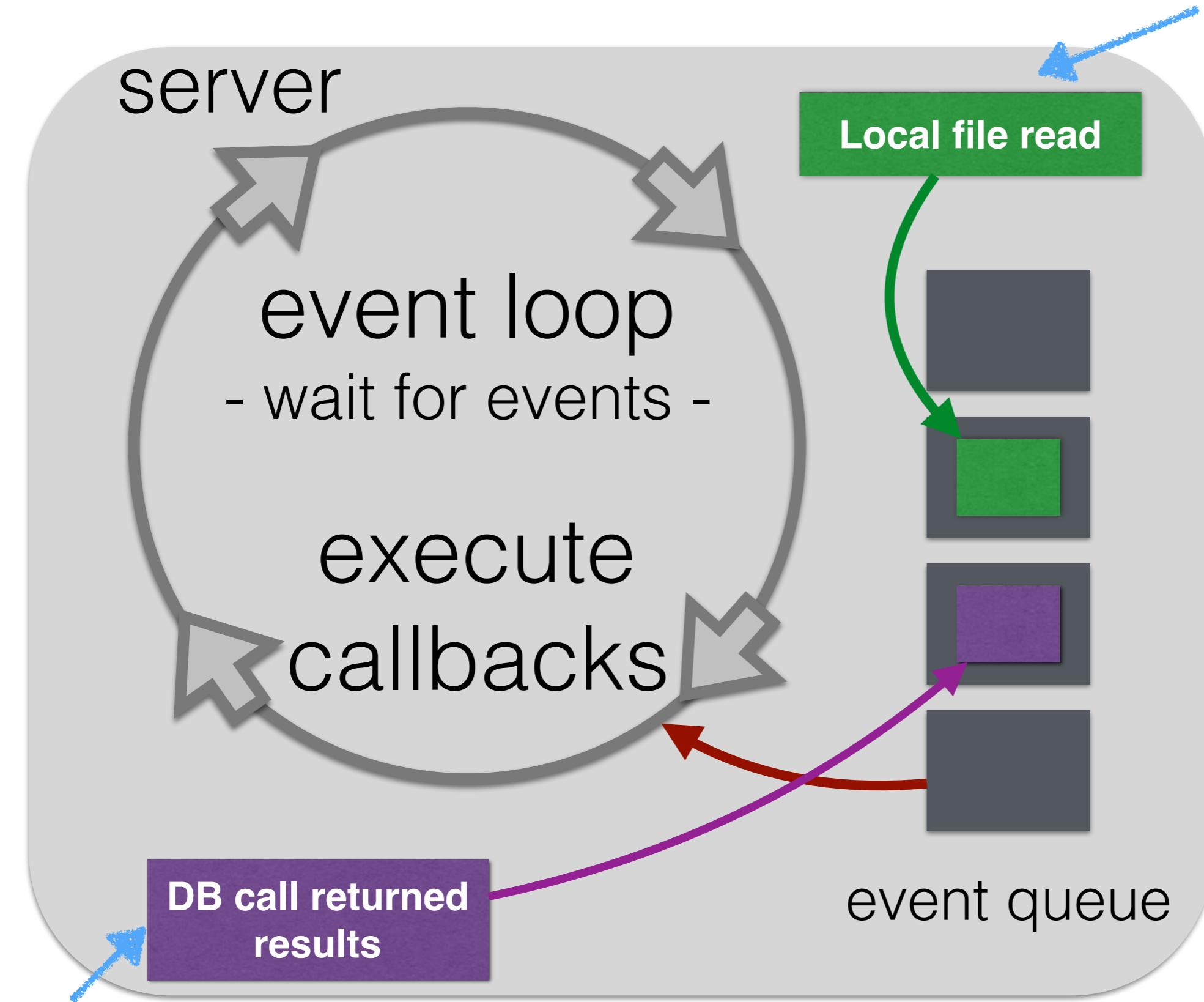


node.js is event-driven



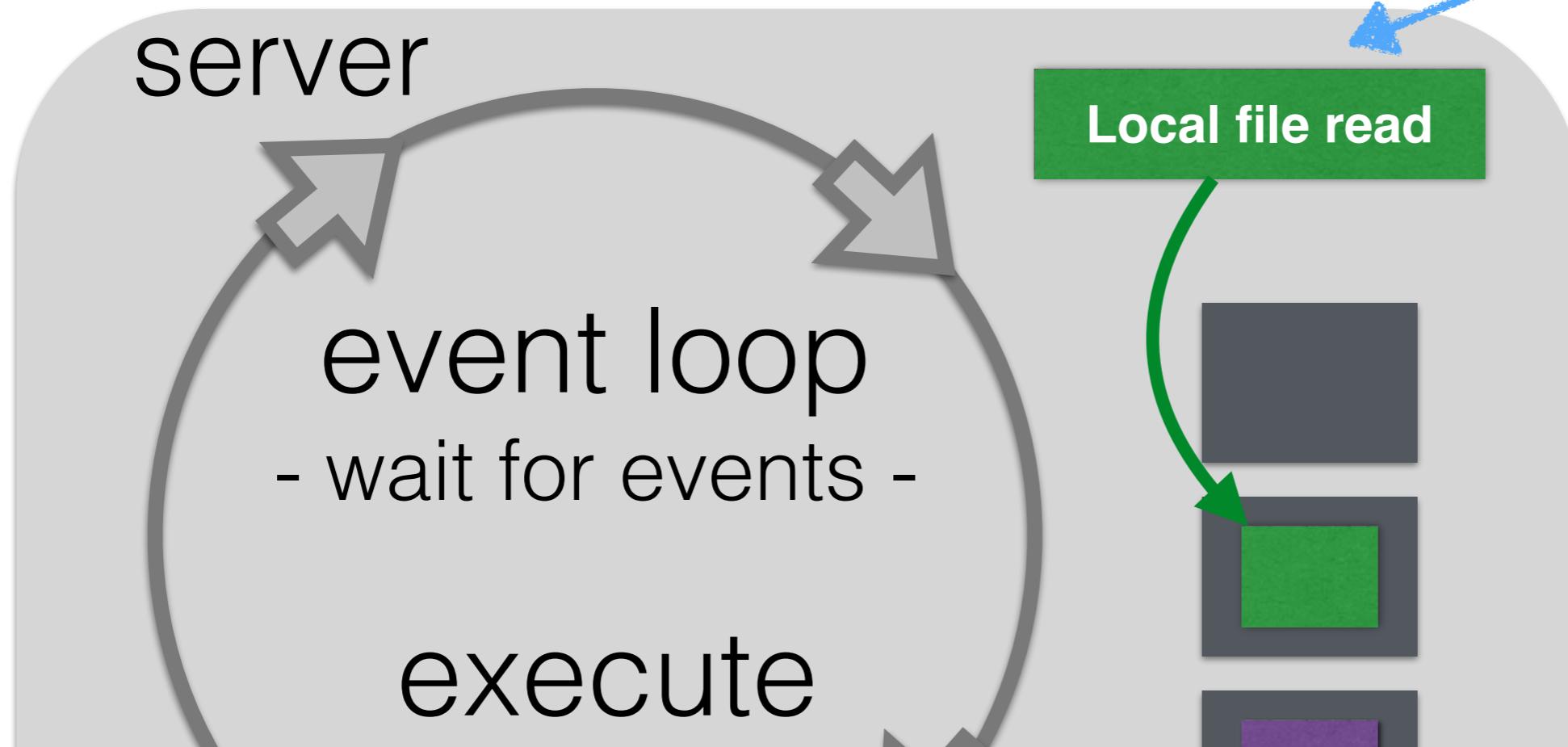
node.js is event-driven

done in parallel



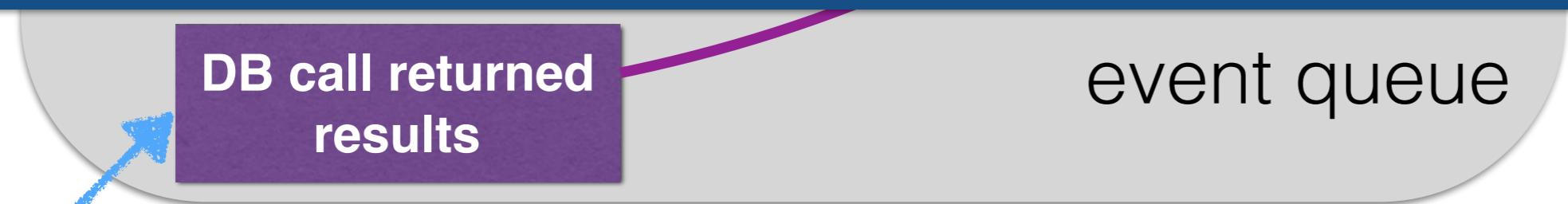
node.js is event-driven

done in parallel



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

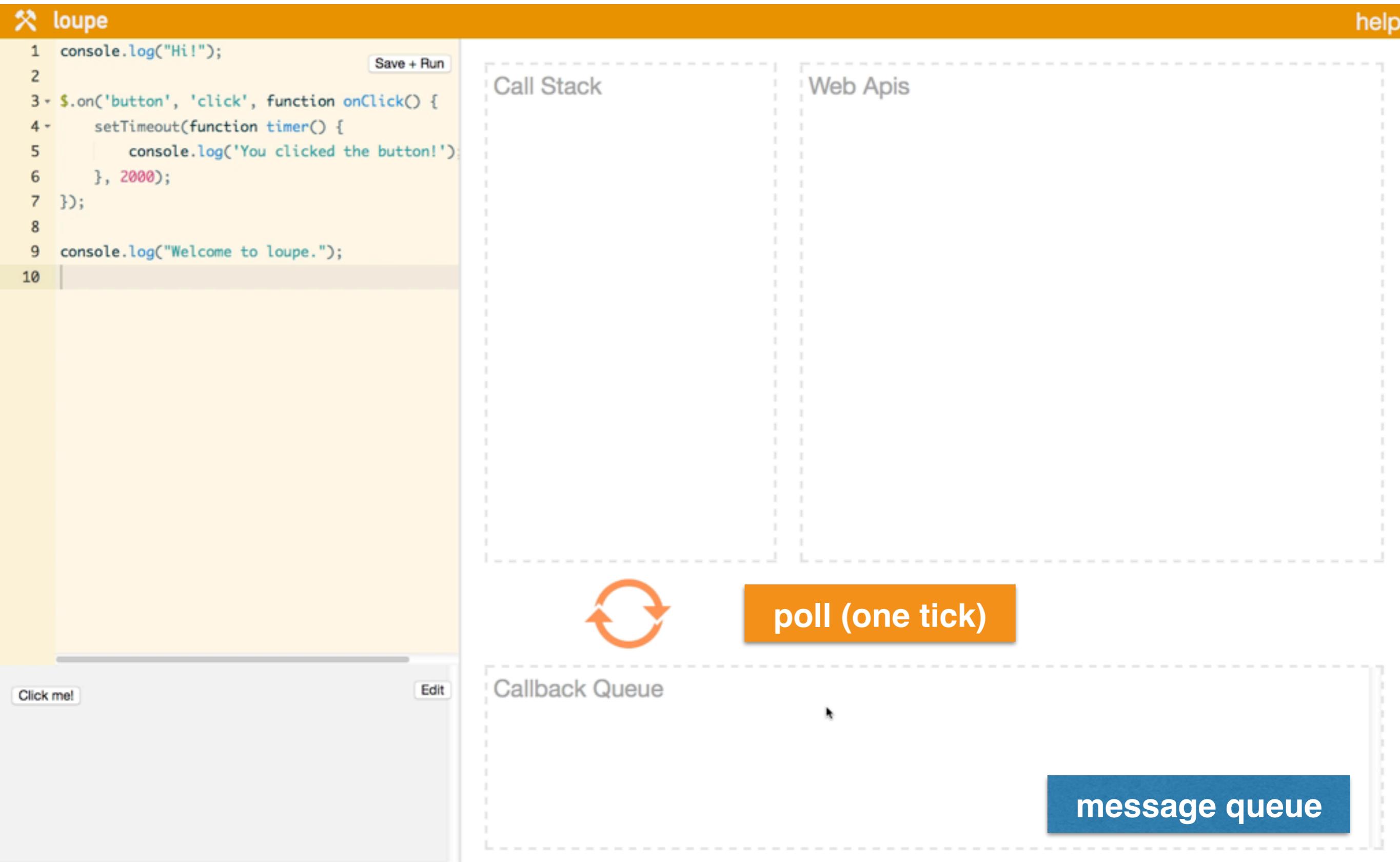
Developers write the callbacks.



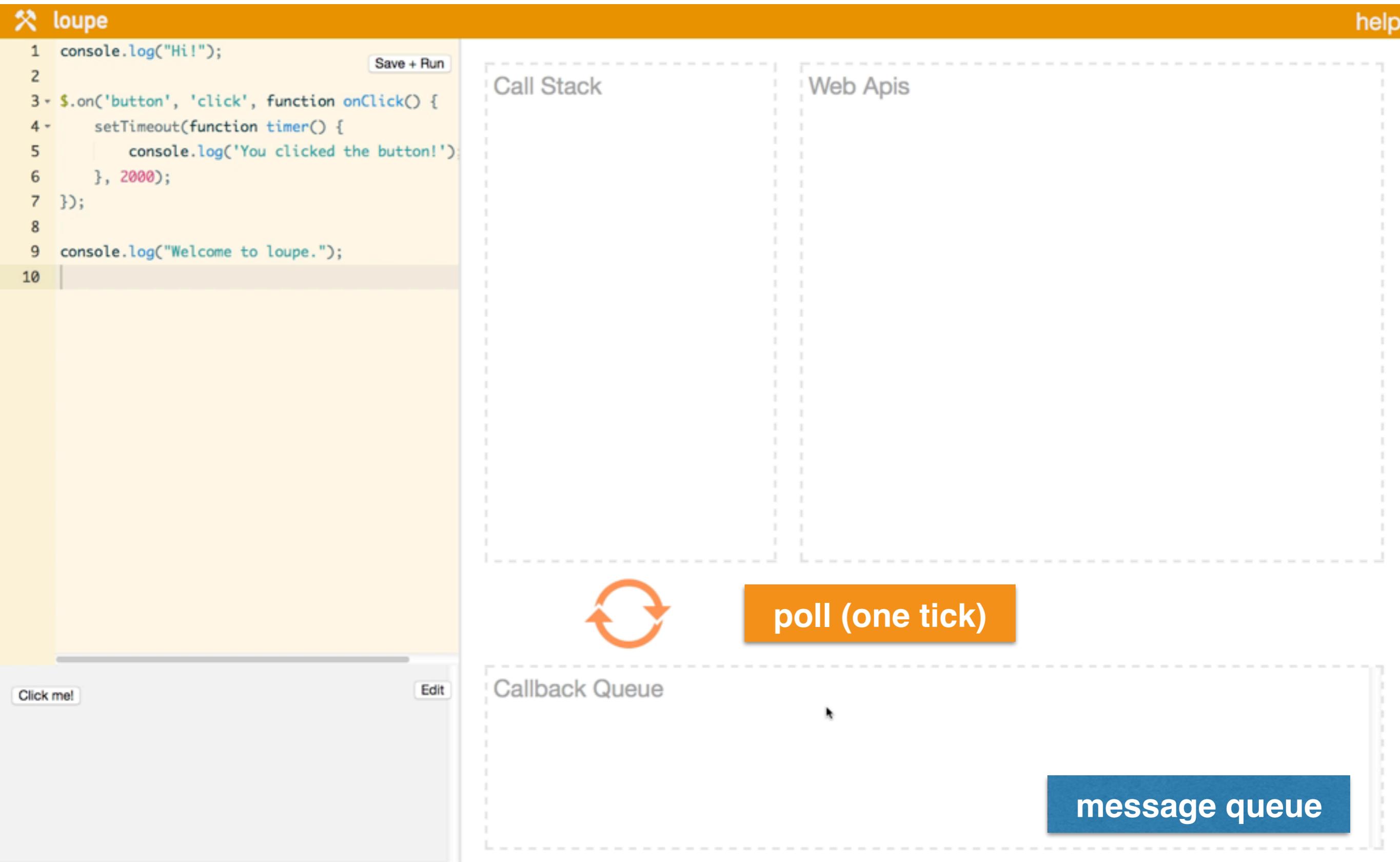
done in parallel

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)



A screenshot of the Visual Studio Code interface. The title bar shows "watching.js - Example1". The left sidebar has icons for File Explorer, Search, Problems, and Snippets. The "EXPLORER" section shows a tree view with "WORKING FILES" expanded, showing "EXAMPLE1" which contains "jsconfig.json", "todos.txt", and "watching.js", with "watching.js" selected and highlighted in blue. The main editor area shows the code for "watching.js":

```
1
```

The status bar at the bottom shows "Ln 1, Col 1" and "UTF-8 LF JavaScript".

Example 1

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



A screenshot of the Visual Studio Code interface. The title bar shows "watching.js - Example1". The left sidebar has icons for File Explorer, Search, Problems, and Snippets. The "EXPLORER" section shows a folder named "EXAMPLE1" containing "jsconfig.json", "todos.txt", and "watching.js", with "watching.js" selected and highlighted in blue. The main editor area shows the code for "watching.js":

```
1
```

The status bar at the bottom shows "Ln 1, Col 1" and "UTF-8 LF JavaScript".

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

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

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

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'");
```

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

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.

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'");
```

Example: watch a file for changes

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

require()
usually
returns a
JavaScript
object

Self-contained piece of
code that provides
reusable functionality.

node.js fs **module**

callback: defines
what should happen
when the file changes

anonymous function

```
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'");
```

Example: watch a file for changes

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

require()
usually
returns a
JavaScript
object

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

callback: defines
what should happen
when the file changes

anonymous function

asynchronous

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

callback: defines what should happen when the file changes

anonymous function

asynchronous

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

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);
```

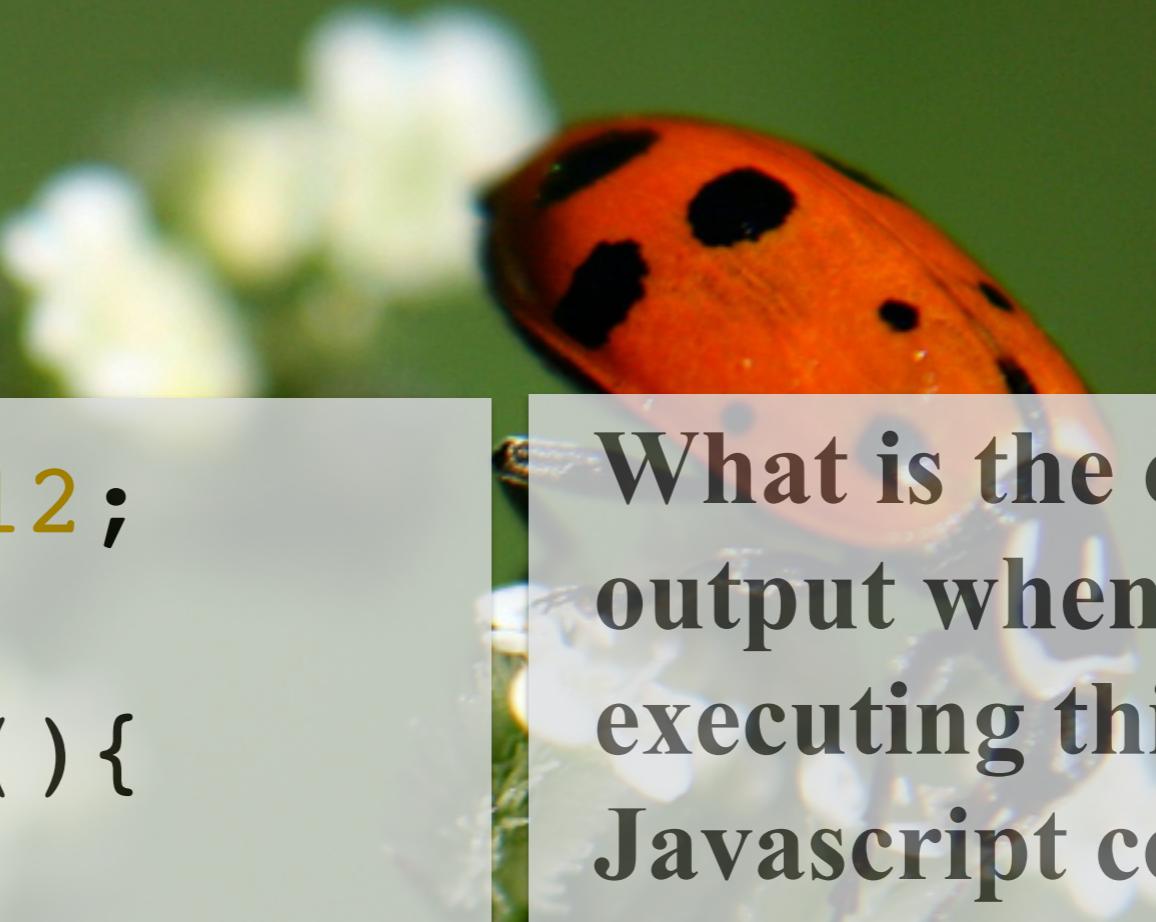
Declarations are processed before any code is executed (the **hoisting principle**).

Declarations are hoisted to the top.

Expressions are not hoisted.

0:31

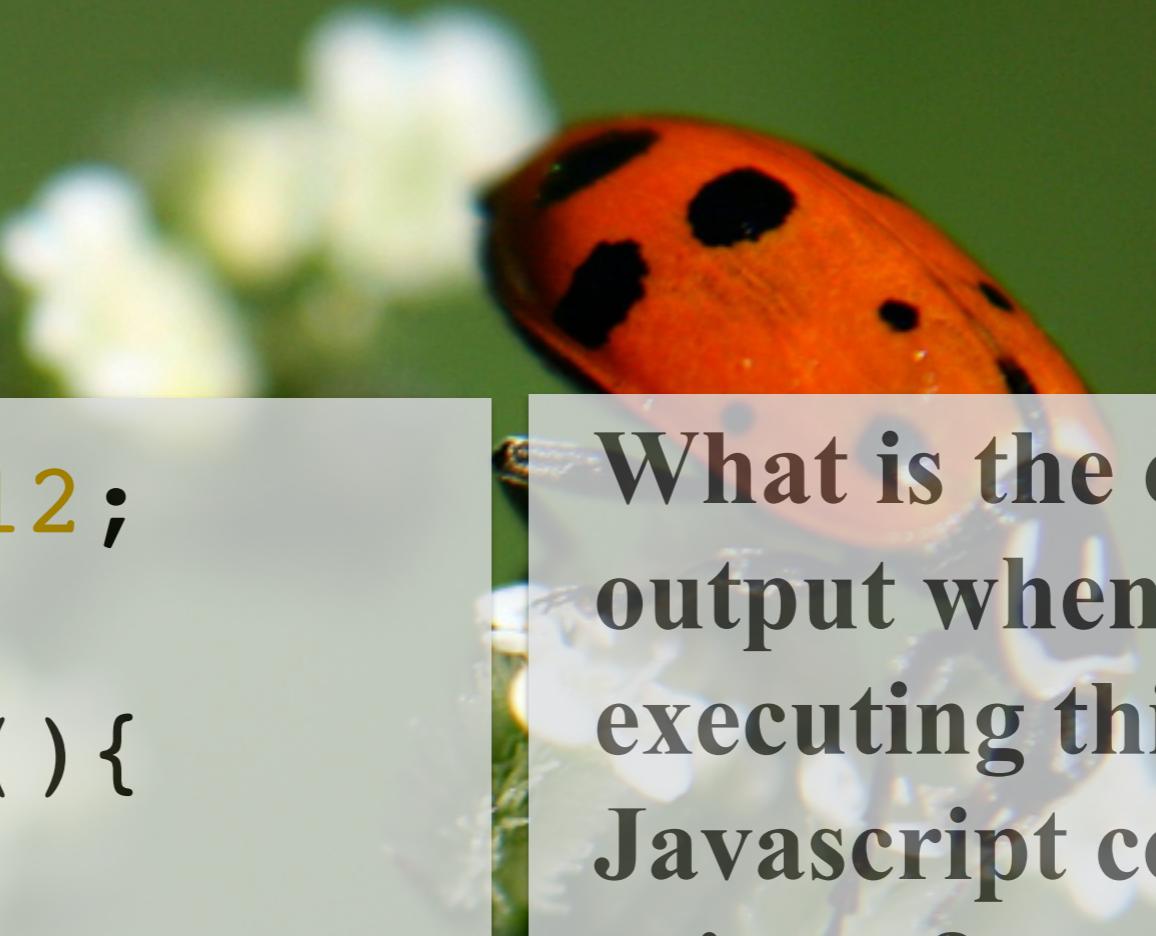
```
1 var bar = 12;  
2  
3 function X(){  
4   bar = 6;  
5   var bar;  
6 }  
7 console.log(bar);  
8 X();  
9 console.log(bar);
```



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

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?

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.

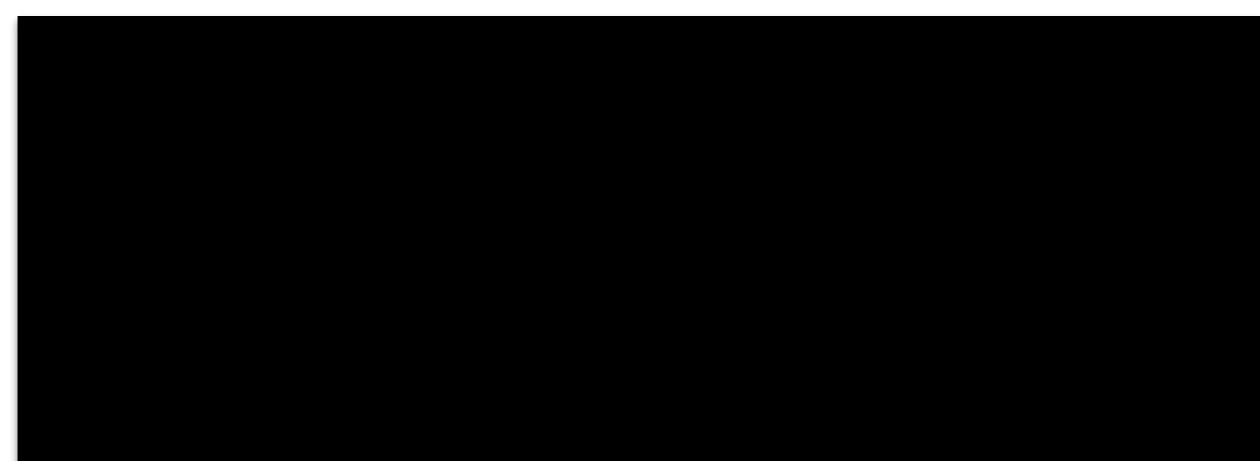
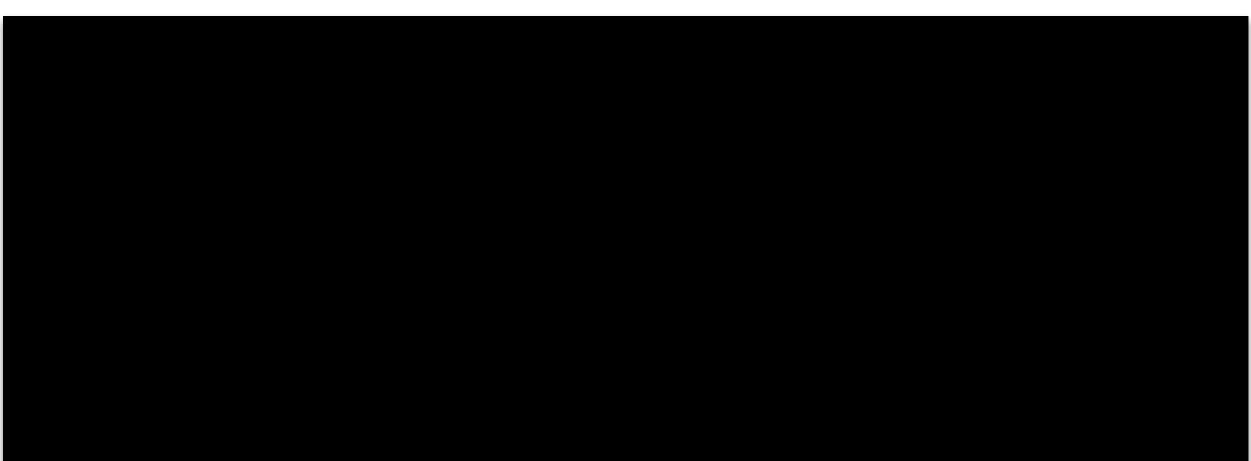
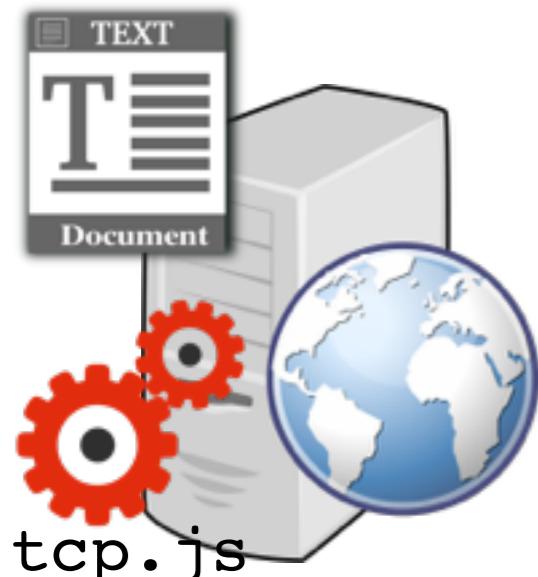
Another phone tries to call that phone.

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.

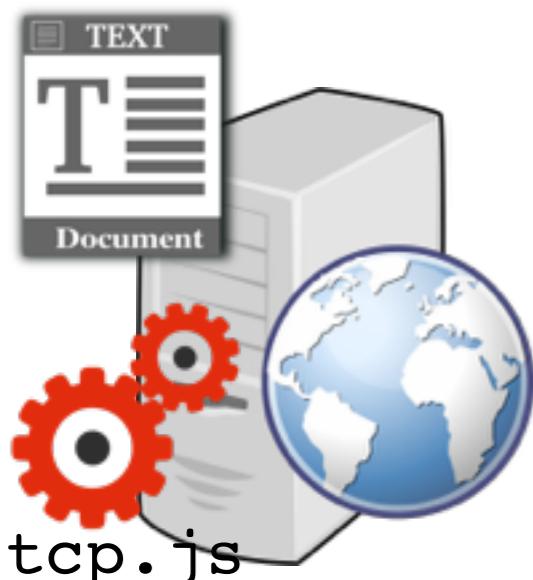
todos.txt



Example: low-level networking with node.js

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

todos.txt

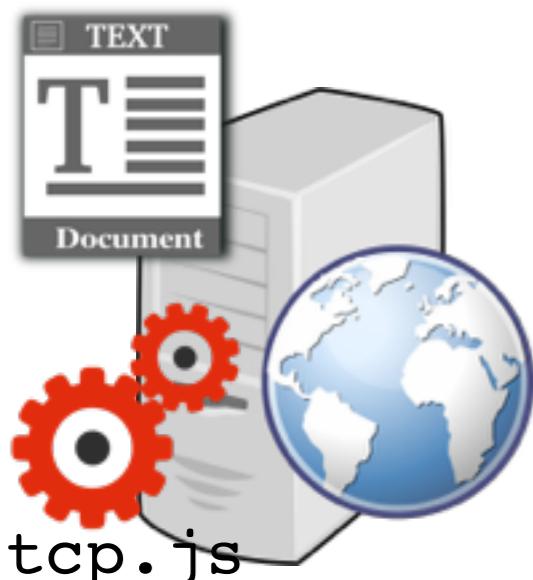


```
server:~ $  
Listening for subscribers ...
```

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

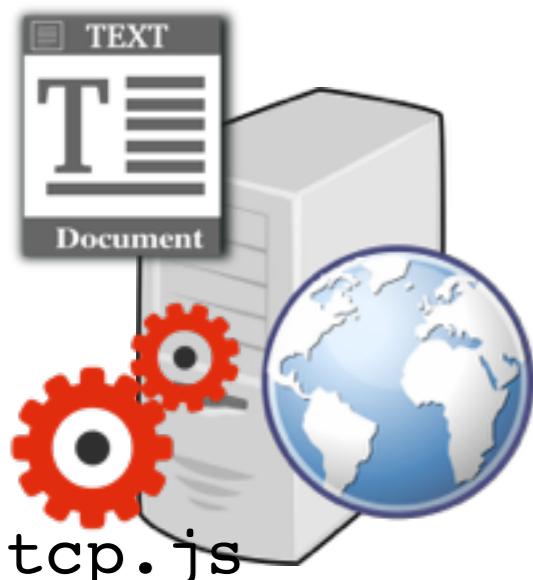


```
server:~ $  
Listening for subscribers ...
```

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

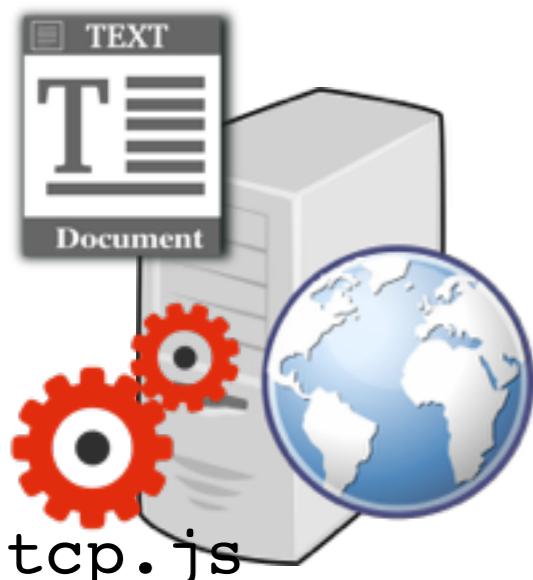


```
server:~ $  
Listening for subscribers ...  
Subscriber connected.
```

Example: low-level networking with node.js

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

todos.txt



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

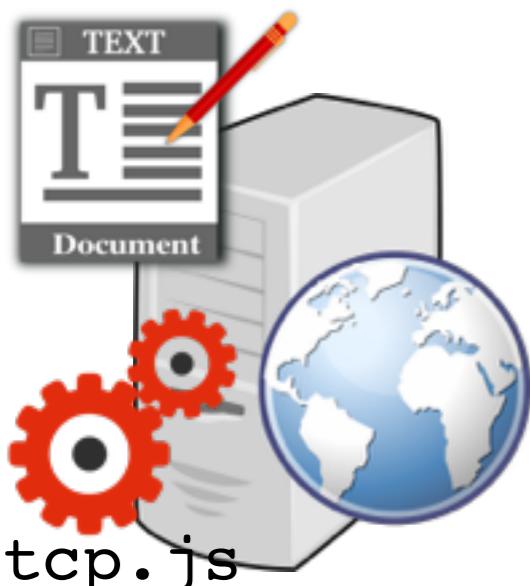


```
client:~ $  
Now watching todos.txt for  
changes ...
```

Example: low-level networking with node.js

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

todos.txt



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

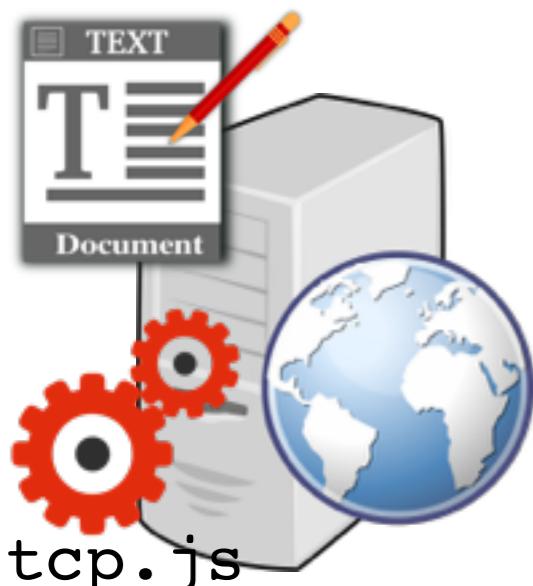


```
client:~ $  
Now watching todos.txt for  
changes ...
```

Example: low-level networking with node.js

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

todos.txt



- (1) client connects to server
- (2) server informs the client



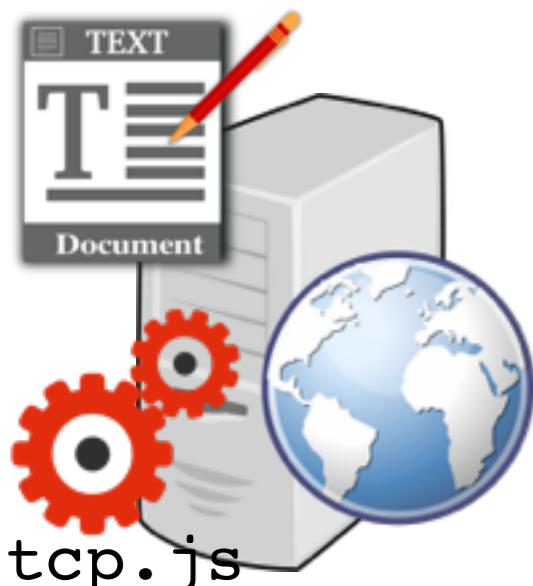
```
server:~ $  
Listening for subscribers ...  
Subscriber connected.
```

```
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



- (1) client connects to server
- (2) server informs the client



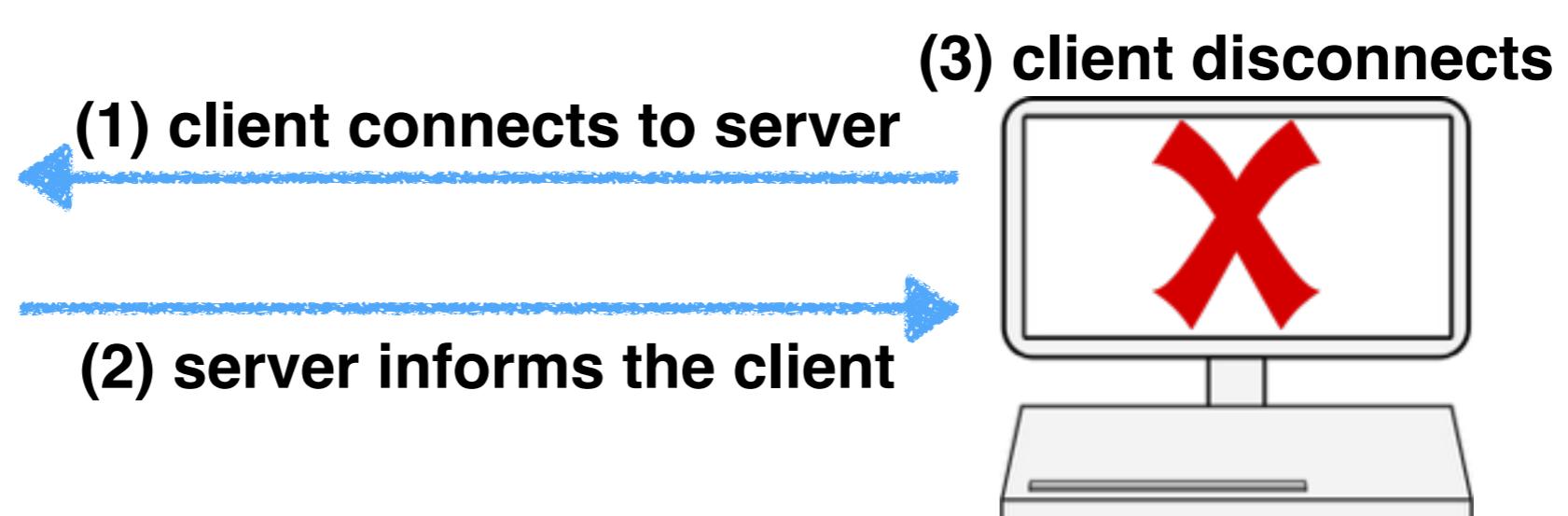
```
server:~ $  
Listening for subscribers ...  
Subscriber connected.
```

```
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.

todos.txt



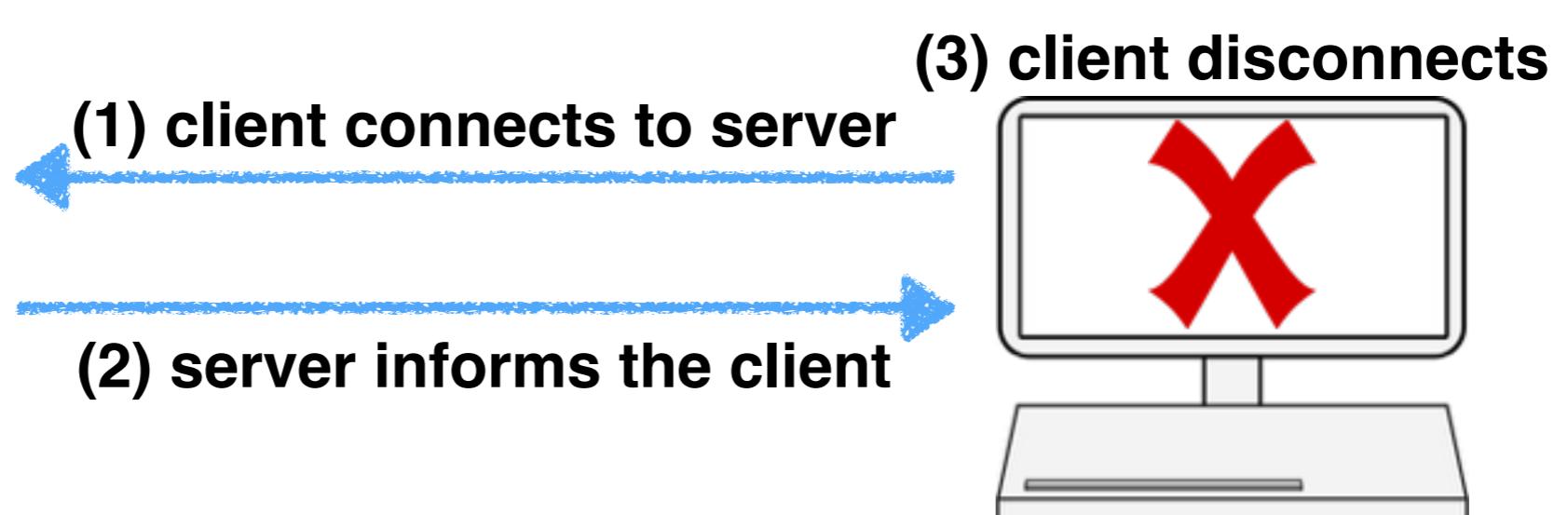
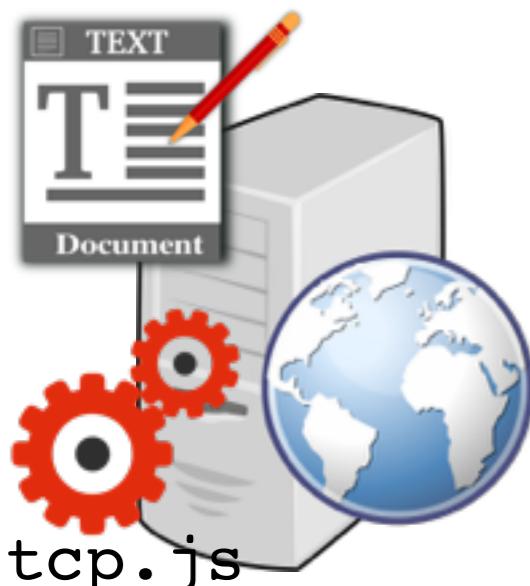
```
server:~ $  
Listening for subscribers ...  
Subscriber connected.
```

```
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.

todos.txt



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

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

Example: low-level networking with node.js

The screenshot shows a Mac OS X desktop environment. At the top, there's a purple header bar with the title "Example: low-level networking with node.js". Below it is a standard Mac menu bar with "QuickTime Player" selected. The main area contains a terminal window and a code editor.

The terminal window at the bottom left shows a command-line interface with the user "claudiahauff" at "Claudias-MacBook-Air:~".

The code editor window, titled "jsconfig.json - Example2", displays the following JSON configuration file:

```
1 {
2     "compilerOptions": {
3         "target": "ES6"
4     },
5     "files": [
6         "tcp.js"
7     ]
8 }
```

The "tcp.js" file is highlighted in the code editor. The status bar at the bottom right of the code editor indicates the file is "Ln 6, Col 16" and is encoded in "UTF-8" with "LF" line endings, and is a "JSON" file.

```
claudiahauff@Claudias-MacBook-Air:~ $
```

Example: low-level networking with node.js

The screenshot shows a Mac OS X desktop environment. At the top, there's a purple header bar with the title "Example: low-level networking with node.js". Below it is a standard Mac menu bar with "QuickTime Player" selected. The main area contains a terminal window and a code editor.

The terminal window at the bottom left shows a command-line interface with the user "claudiahauff" at "Claudias-MacBook-Air:~".

The code editor window, titled "jsconfig.json - Example2", displays the following JSON configuration file:

```
1 {
2     "compilerOptions": {
3         "target": "ES6"
4     },
5     "files": [
6         "tcp.js"
7     ]
8 }
```

The "tcp.js" file is highlighted in the code editor. The status bar at the bottom right of the code editor indicates the file is "Ln 6, Col 16" and is encoded in "UTF-8" with "LF" line endings, and is a "JSON" file.

```
claudiahauff@Claudias-MacBook-Air:~ $
```

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...');

client-side output
server-side output
24 });


```

Example: low-level networking with node.js

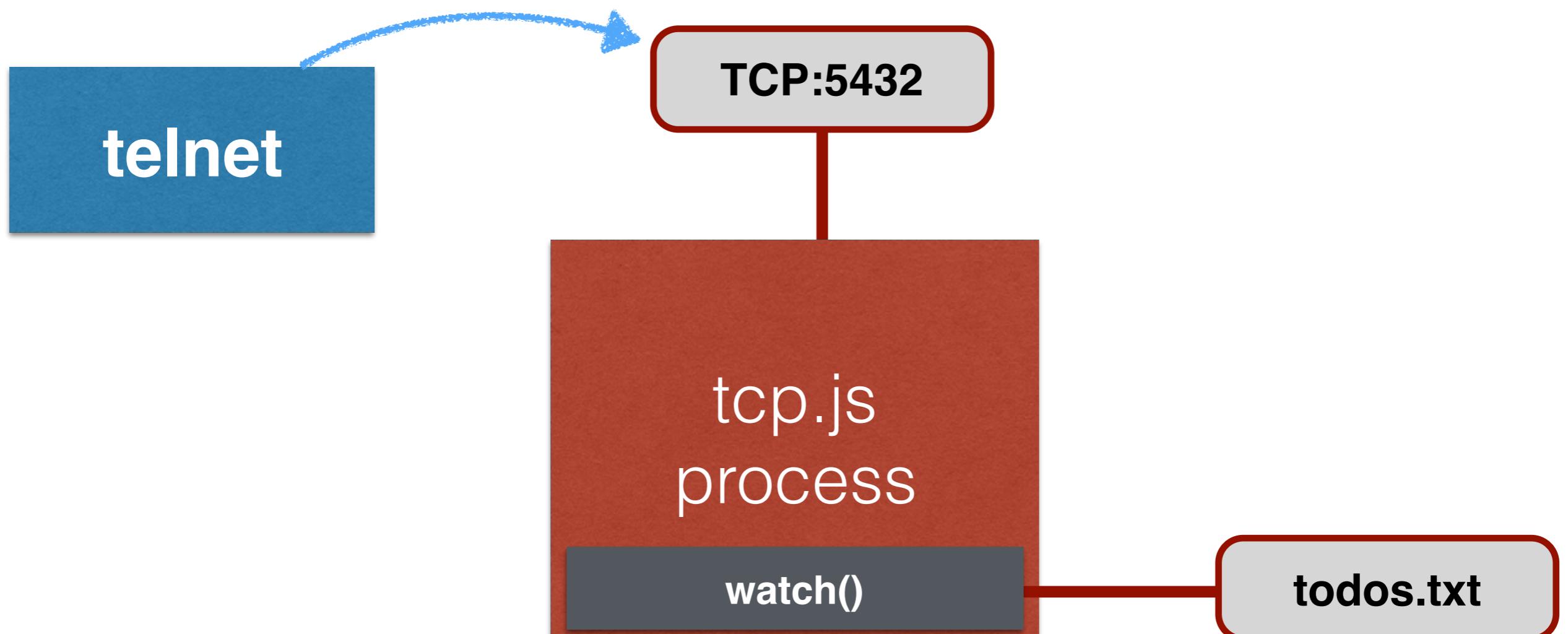
Start the **server** on the command line: `$ node --harmony tcp.js`
Client terminal: `$ telnet localhost 5432`

```
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

The screenshot shows a Mac OS X desktop environment. In the top left, there's a QuickTime Player window with a menu bar. The main area features a file browser window titled "jsconfig.json - Example3". The sidebar on the left lists "WORKING FILES" and "EXAMPLE3" sections, with "jsconfig.json" and "web.js" selected. The main pane displays the JSON configuration file:

```
1 {  
2   "compilerOptions": {  
3     "target": "ES6"  
4   },  
5   "files": [  
6     "web.js"  
7   ]  
8 }
```

Below the browser window is a browser interface with a blue header bar showing "Ln 8, Col 2" and "UTF-8". The main content area of the browser shows a dark image of bare trees.

For quick access, place your bookmarks here on the bookmarks bar. [Import bookmarks now...](#)

Example 3

“Hello World” in the browser

The screenshot shows a Mac OS X desktop environment. In the top left, there's a QuickTime Player window with a menu bar. The main area features a file browser window titled "jsconfig.json - Example3". The sidebar on the left lists "WORKING FILES" and "EXAMPLE3" sections, with "jsconfig.json" and "web.js" selected. The main pane displays the JSON configuration file:

```
1 {  
2   "compilerOptions": {  
3     "target": "ES6"  
4   },  
5   "files": [  
6     "web.js"  
7   ]  
8 }
```

Below the browser window is a browser interface with tabs, a search bar, and a status bar indicating "Ln 8, Col 2" and "UTF-8". A message at the bottom of the browser window says, "For quick access, place your bookmarks here on the bookmarks bar. [Import bookmarks now...](#)".

“Hello World” in the browser

The screenshot shows a Mac OS X desktop environment. At the top, there's a purple bar with the title "Example 3" and a large white font header "‘Hello World’ in the browser". Below this is a standard Mac menu bar with "QuickTime Player" selected. The main window is a terminal-like interface with a sidebar on the left labeled "EXPLORE" showing "WORKING FILES" and "EXAMPLE3" with files "jsconfig.json" and "web.js". The main pane displays the contents of "jsconfig.json":

```
1 {
2   "compilerOptions": {
3     "target": "ES6"
4   },
5   "files": [
6     "web.js"
7   ]
8 }
```

A red arrow points from the bottom-left towards the terminal window, indicating the source of the output. A red box highlights the URL "localhost:3000/Hallihallo" in the browser's address bar. The browser window shows the text "Hello World" in a large font.

localhost:3000/Hallihallo or any other path also works

Ln 8, Col 2 UTF-8 LF JSON ☺

New Tab

For quick access, place your bookmarks here on the bookmarks bar. [Import bookmarks now...](#)

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

```
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

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

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

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

HTTP **request** object

The “Hello World” of node.js

```
1 var http = require("http");
2 var server; This is standard JavaScript. We can
3 add variables, functions, objects...
4 var sentCounter = 0; HTTP response object
5
6 server = http.createServer(function (req, res) {
7     res.writeHead(200, {"Content-Type": "text/plain"}); HTTP request object
8     res.end("Hello World!");
9     sentCounter++;
10    console.log(sentCounter + " HTTP responses sent
11                                in total");
12 });
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; This is standard JavaScript. We can
3 add variables, functions, objects...
4 var sentCounter = 0; HTTP response object
5
6 server = http.createServer(function (req, res) {
7     res.writeHead(200, {"Content-Type": "text/plain"}); HTTP request object
8
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); Are we sending an object?
                                                    Yes and no (JSON)
```

The “Hello World” of node.js

```
1 var http = require("http");
2 var server; This is standard JavaScript. We can
3 add variables, functions, objects...
4 var sentCounter = 0; HTTP response object
5
6 server = http.createServer(function (req, res) {
7     res.writeHead(200, {"Content-Type": "text/plain"}); HTTP request object
8     res.end("Hello World!");
9     sentCounter++;
10    console.log(sentCounter + " HTTP responses sent
11        in total");
12
13 });
14 Any port number between 1024 and 49151 is fine.
15 var port = 2345;
16 server.listen(port);
17 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);
```



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;                                function as parameter  
15  
16 server = http.createServer(simpleHTTPResponder);  
17  
18 var port = process.argv[2];                          command line parameter  
19 server.listen(port);  
20 console.log("Server listening on port "+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);
```

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": "text/plain"});
9     var query = url_parts.query;
10    if( query["name"]!=undefined) {
11      res.end("Greetings "+query["name"]);
12    }
13    else { res.end("Greetings Anonymous"); }
14  }
15  else {
16    res.writeHead(404, {"Content-Type": "text/plain"});
17    res.end("Only /greetme is implemented.");
18  }
19
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**

URL routing

The screenshot shows a Mac OS X desktop environment. At the top, there's a purple bar with the title "Example 4" on the left and "URL routing" in large white letters in the center. Below the purple bar is a standard Mac menu bar with "QuickTime Player" selected. To the right of the menu bar are various system status icons. The main window is a "QuickTime Player" window showing a video preview of a forest scene. Overlaid on this window is a "VS Code" editor. The left sidebar of the VS Code editor shows a file tree with a folder named "EXAMPLE3" expanded, containing ".vscode", "launch.json", "jsconfig.json", and "routing.js". The "routing.js" file is the active tab, showing the following code:

```
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
```

The status bar at the bottom of the VS Code window indicates "Ln 1, Col 1" and "JavaScript".

URL routing

The screenshot shows a Mac OS X desktop environment. At the top, there's a purple bar with the title "Example 4" on the left and a search bar on the right. Below the purple bar, a QuickTime Player window is open, showing a video player interface with various controls and a status bar indicating "Sat 19:26 Claudia Hauff". The main content area of the screen is occupied by a Visual Studio Code (VS Code) window. The VS Code interface includes:

- File Explorer (Left):** Shows a tree view of files and folders. The "EXAMPLE3" folder is expanded, revealing ".vscode", "launch.json", "jsconfig.json", and "routing.js".
- Editor Area (Center):** Displays the content of the "routing.js" file. The code implements a simple HTTP responder using the Node.js "http" module. It checks if the pathname is "/greetme" and responds with a name if provided in the query string, or with "Anonymous" if not. If no pathname is provided, it returns a 404 error.
- Status Bar (Bottom):** Shows "Ln 1, Col 1" and "UTF-8" along with other standard status bar icons.

```
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
```

URL routing

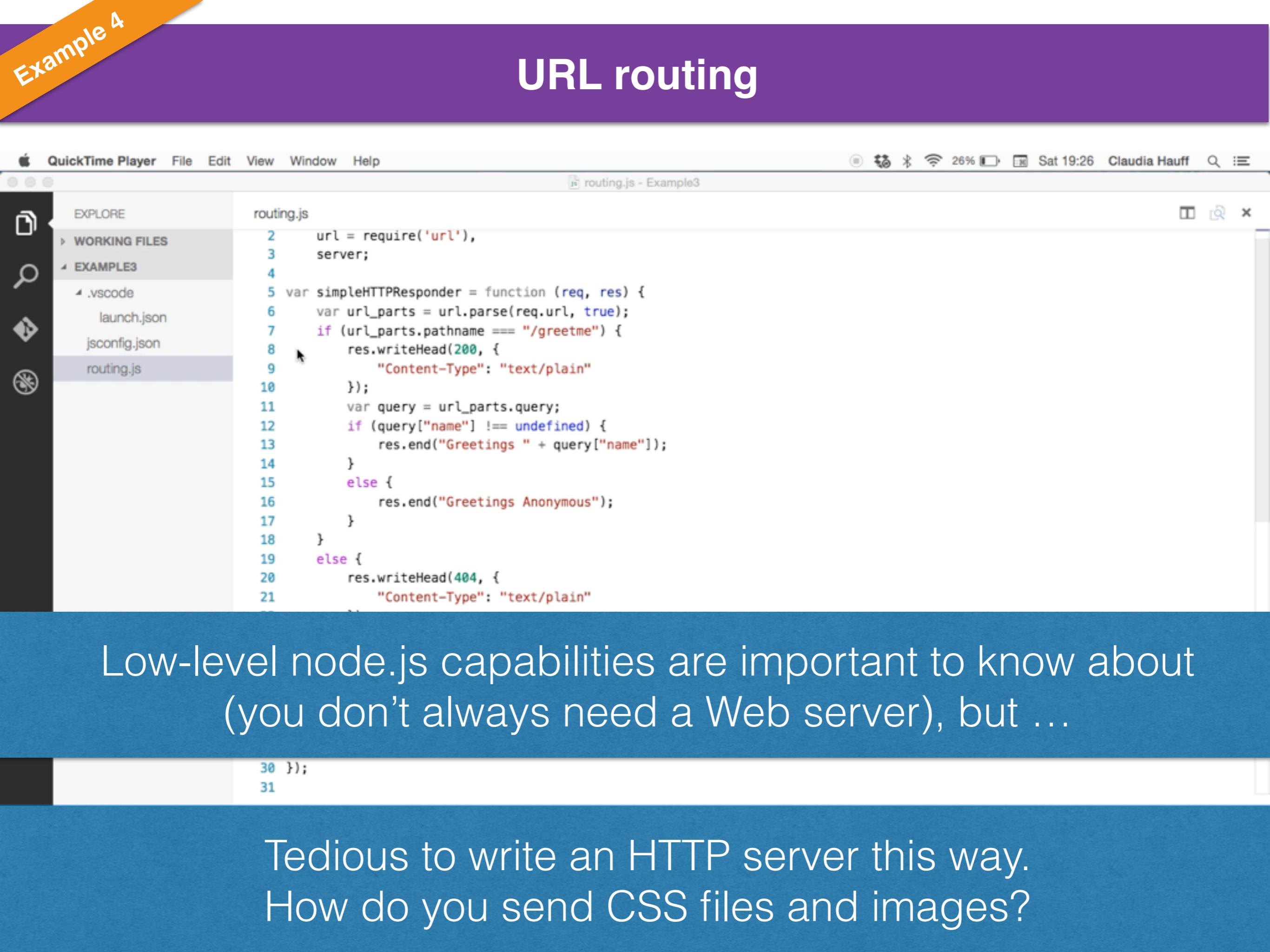
The screenshot shows a Mac OS X desktop environment. At the top, there's a purple bar with the title "Example 4" and a white bar with the title "URL routing". Below the purple bar is a menu bar for "QuickTime Player" with options: File, Edit, View, Window, Help. To the right of the menu bar are system status icons and the date/time "Sat 19:26 Claudia Hauff". The main window contains a "QuickTime Player" player bar at the top with controls like play/pause, volume, and a search field. Below it is a "ROUTER" tab and a "ROUTER" section with a tree view showing "WORKING FILES" and "EXAMPLE3" (which is expanded). Under "EXAMPLE3", there are files ".vscode", "launch.json", "jsconfig.json", and "routing.js" (which is selected and highlighted in blue). The central area is a code editor with syntax highlighting for JavaScript. The code in "routing.js" is as follows:

```
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 }
24
25 server.on('request', simpleHTTPResponder);
26
27 module.exports = server;
```

The bottom of the code editor shows status information: "Ln 1, Col 1" and "UTF-8 LF JavaScript".

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

URL routing



The screenshot shows a Mac OS X desktop with a QuickTime Player window open. The window title is "routing.js - Example3". The menu bar at the top includes "QuickTime Player", "File", "Edit", "View", "Window", "Help", and system status like battery level (26%), signal strength, and date/time (Sat 19:26). The desktop icons on the left include Finder, Spotlight, and others. The main content area shows the "routing.js" file code:

```
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 }
24
25
26
27
28
29
30 });
31
```

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.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

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.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

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.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

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, res) {
11   var query = url.parse(req.url, true);
12   var name = (query["name"] != undefined) ? query["name"] : "Anonymous";
13   res.send("<html><head></head><body><h1>
14               Greetings " + name + "</h1></body></html>
15             ");
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

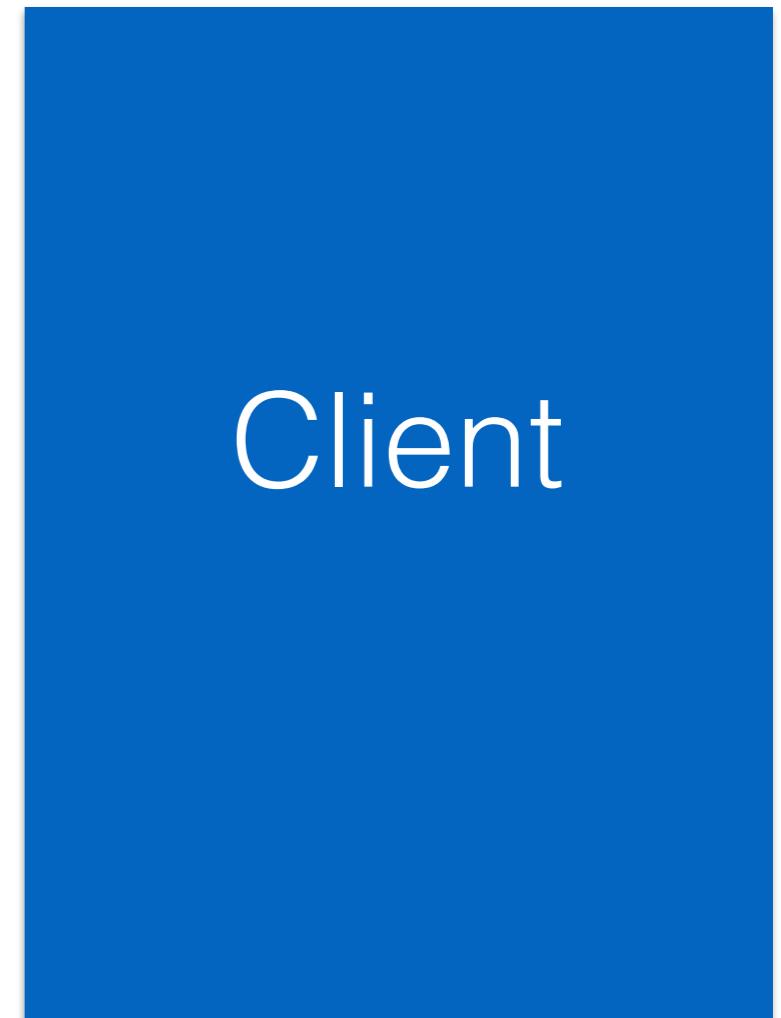
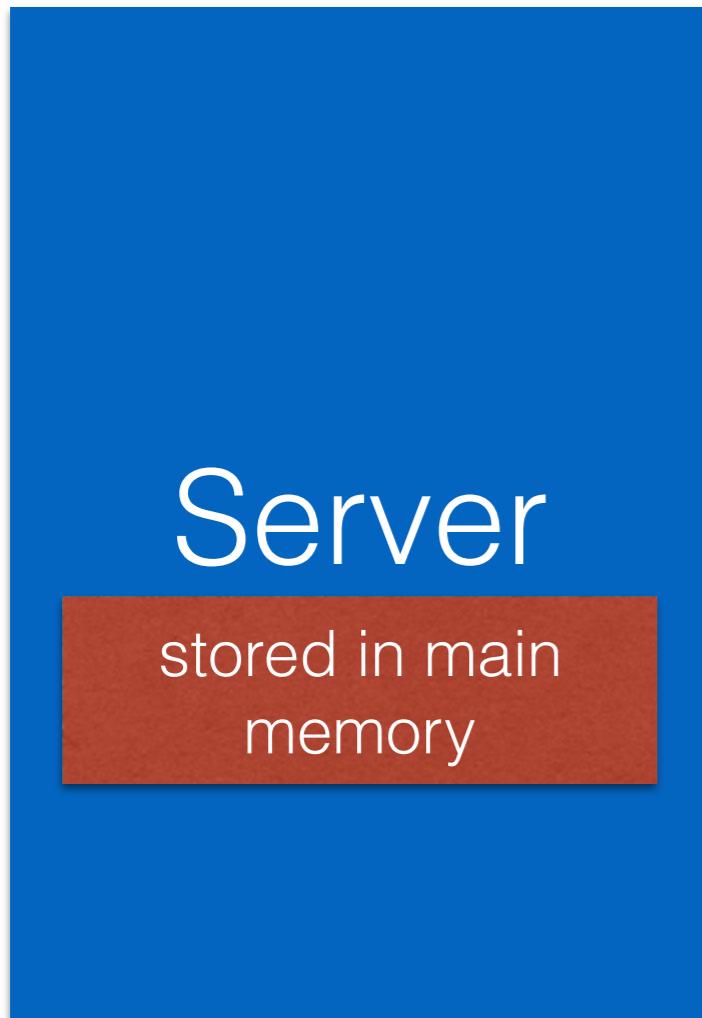
```
todo-server.js
client/
    index.html
    html/
        =>error.html
        =>addtodo.html
    images/
        =>background.png
        =>logo.png
    css/
        =>layout.css
        =>style.css
    javascript/
        =>client-app.js
```

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
```

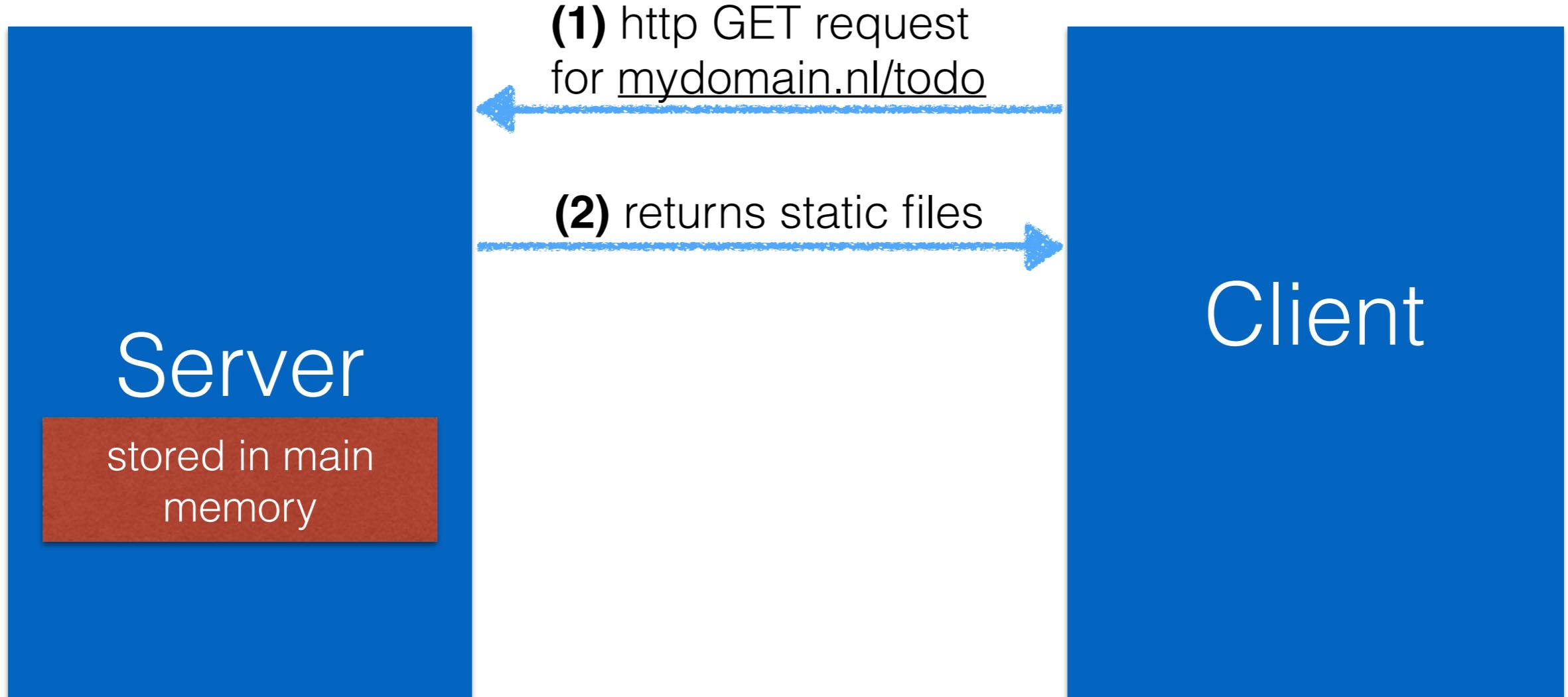
TODO Web app flow



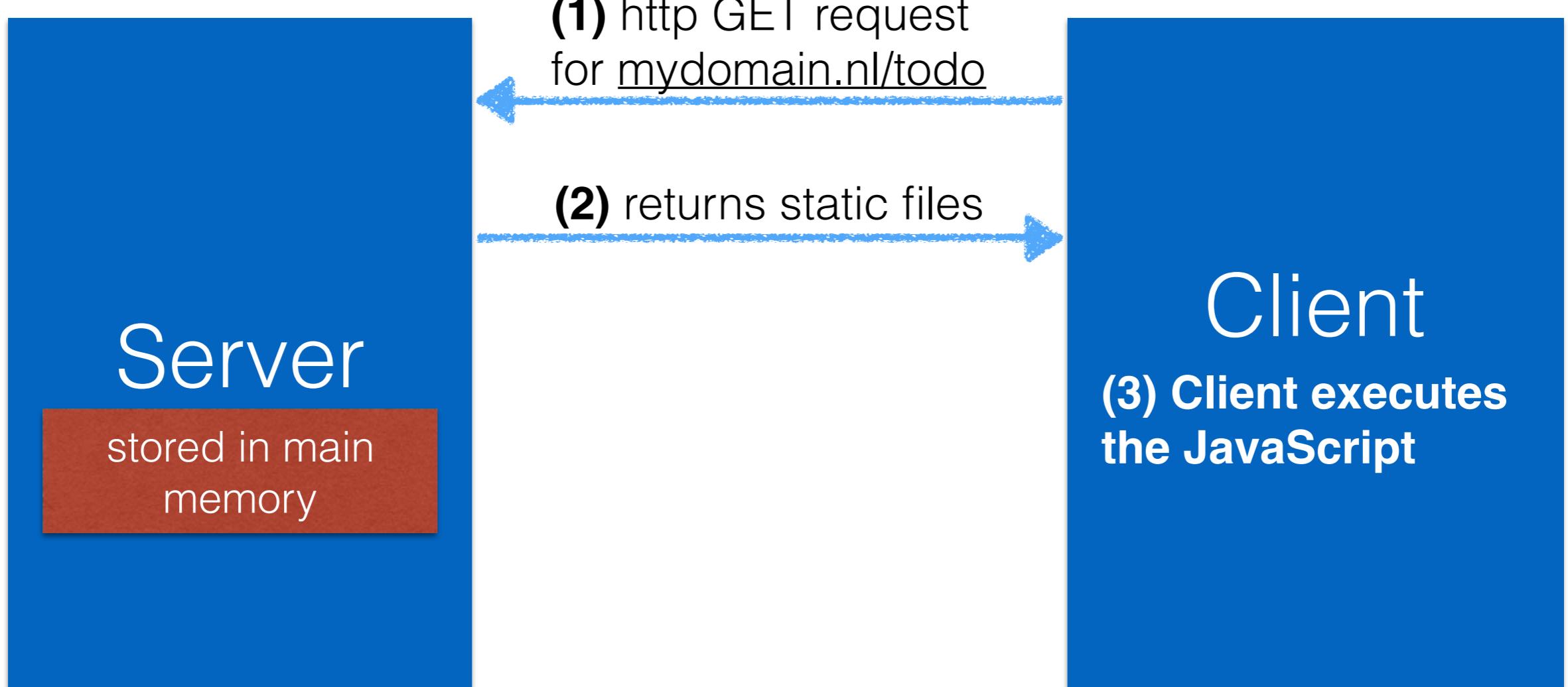
TODO Web app flow



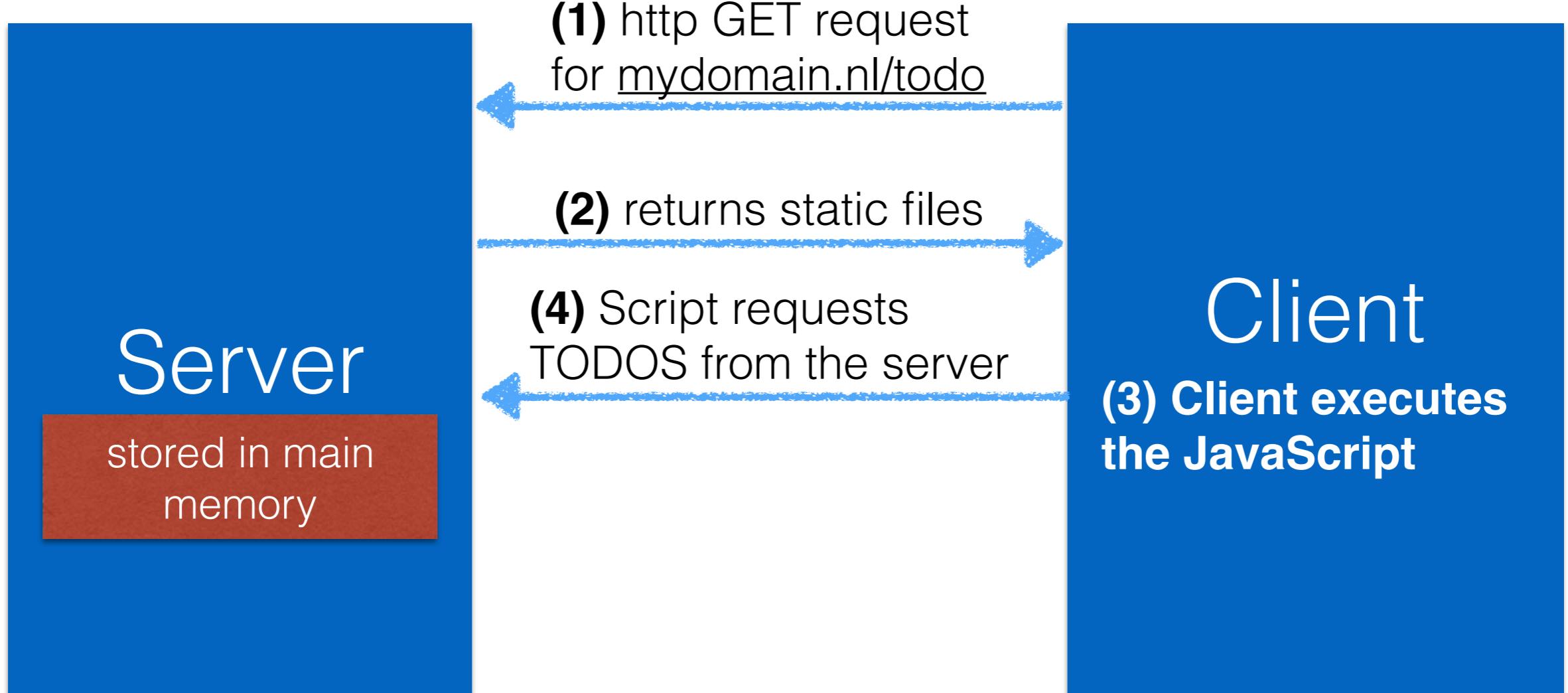
TODO Web app flow



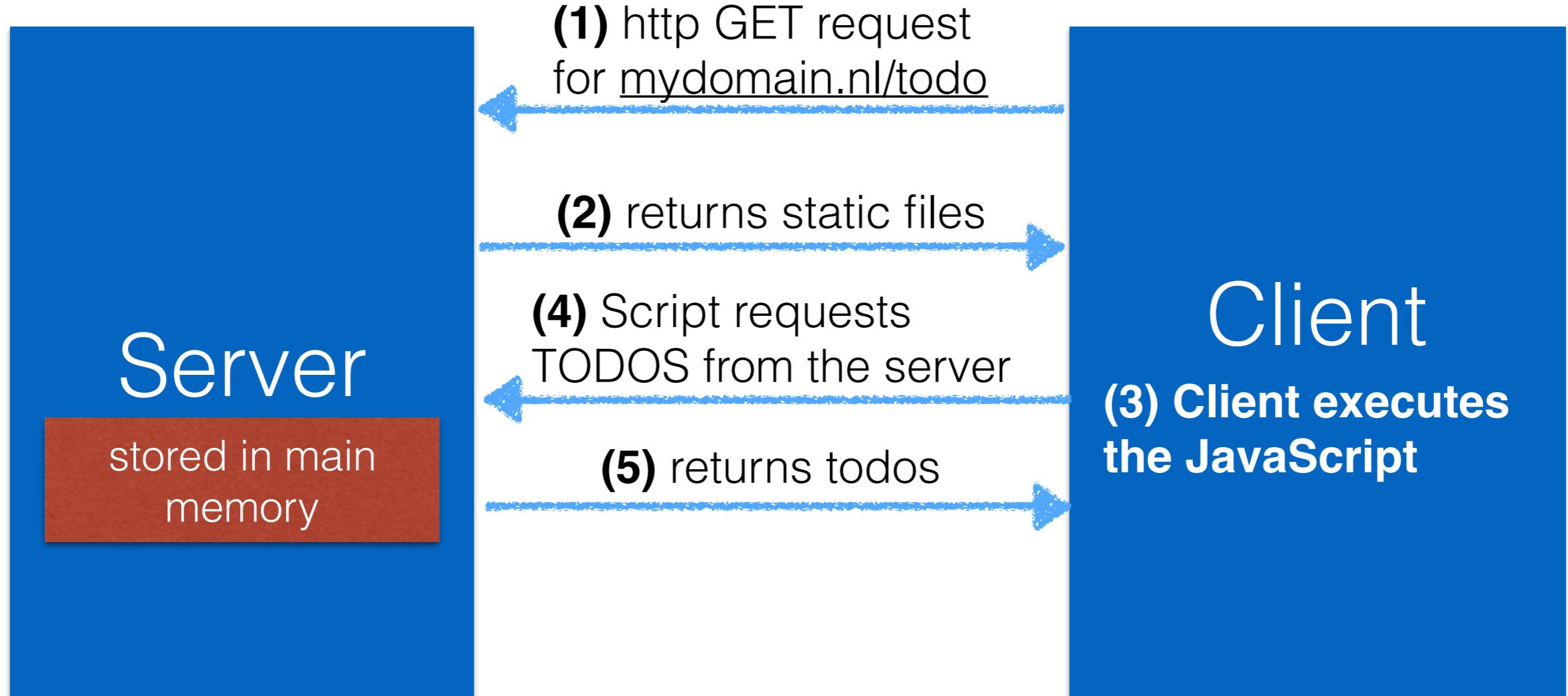
TODO Web app flow



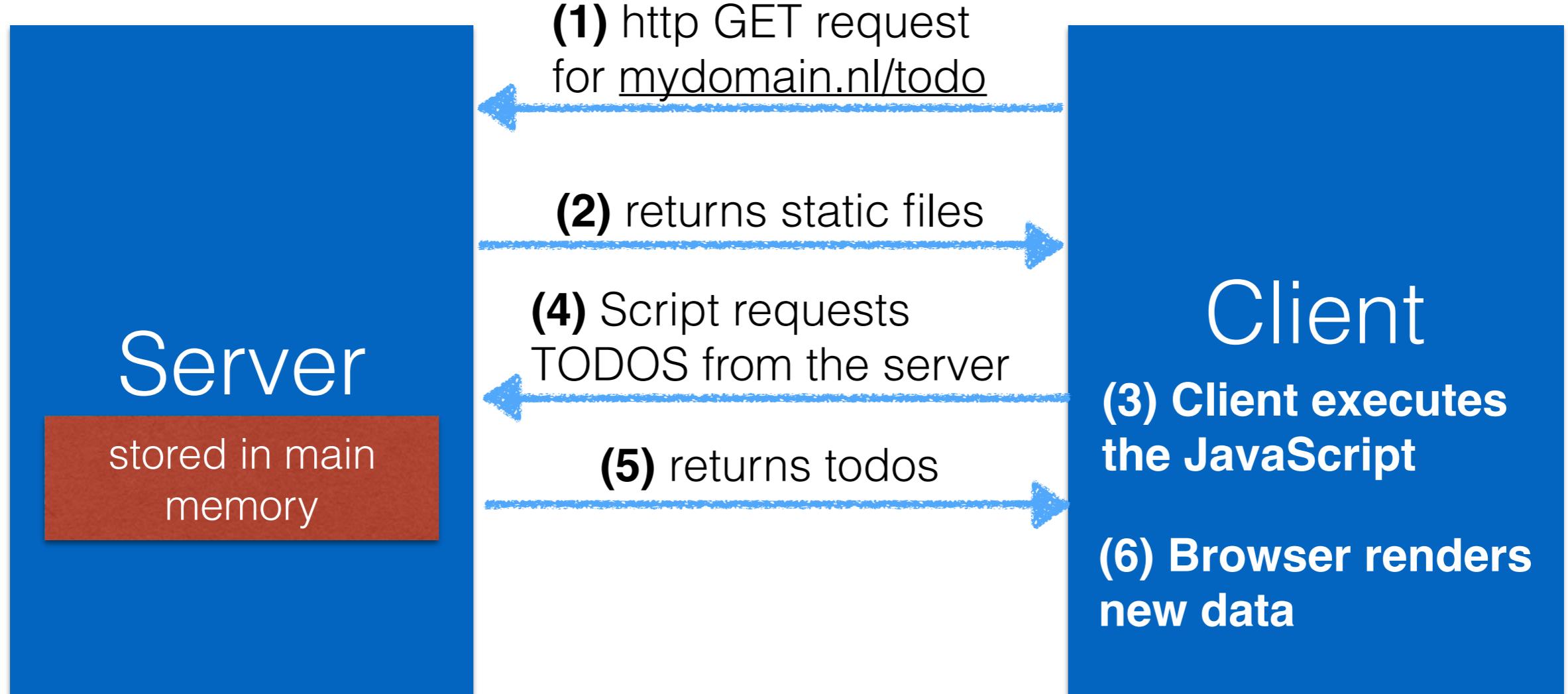
TODO Web app flow



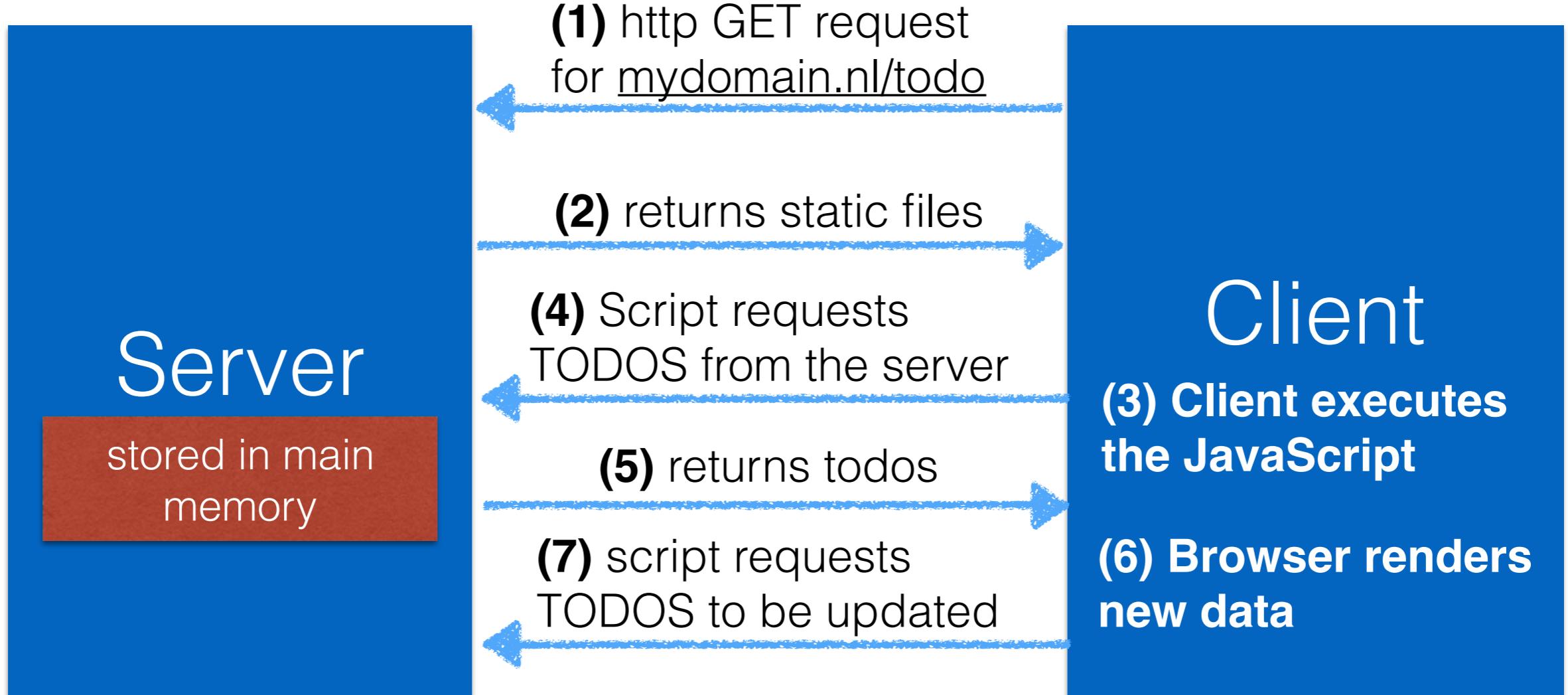
TODO Web app flow



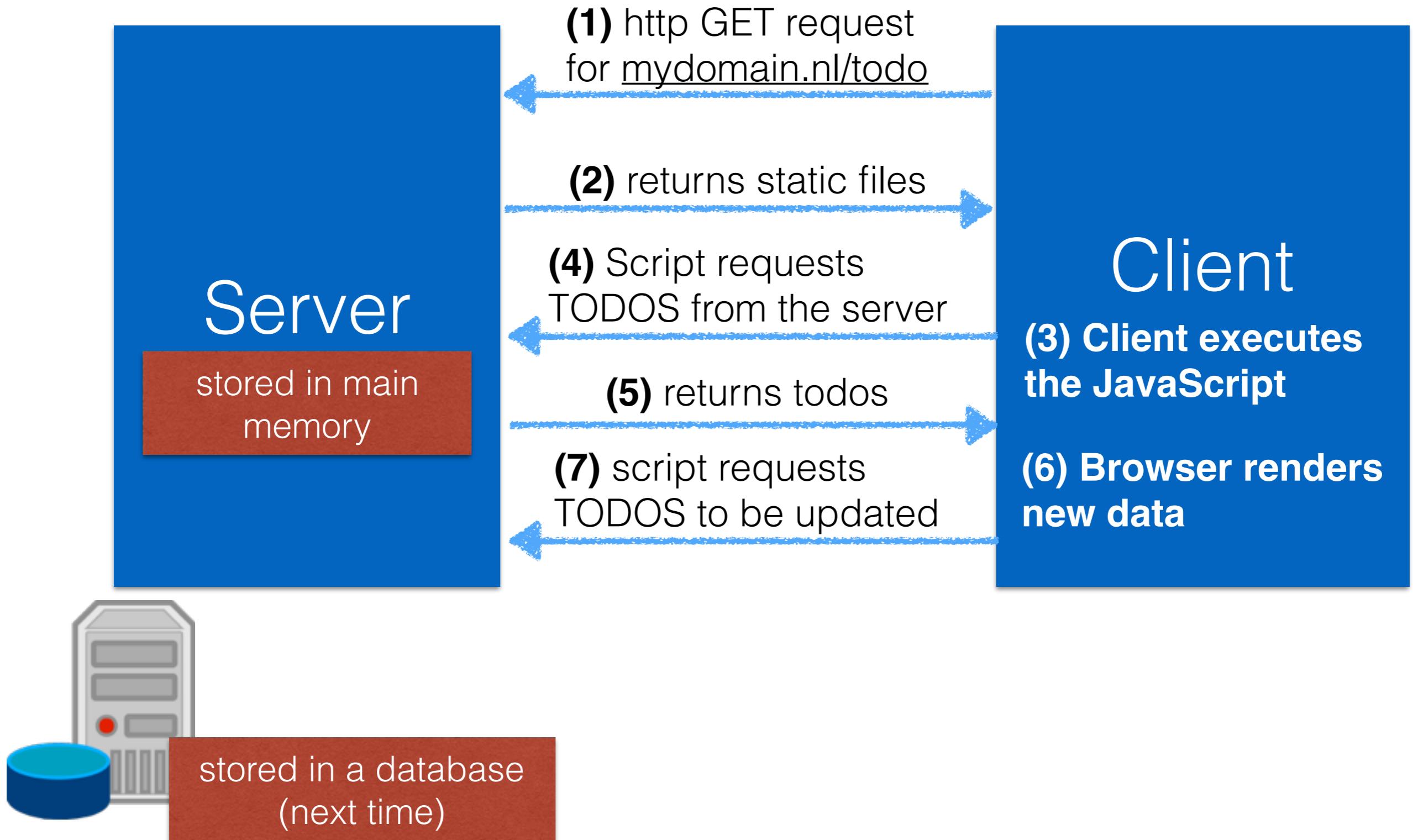
TODO Web app flow



TODO Web app flow



TODO Web app flow



**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

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

XML

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

JSON

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);                                Send JSON to client
20 });

in-memory todos
```

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) {           is message a key
2   var url_parts = url.parse(req.url, true);          in the query?
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});
```

Sending json & todo updating

The screenshot shows a Mac OS X desktop environment. At the top, there's a menu bar with 'QuickTime Player', 'File', 'Edit', 'View', 'Window', and 'Help'. To the right of the menu bar are system status icons. Below the menu bar is a window titled 'todo-server.js - Example5'. This window has a sidebar on the left labeled 'EXPLORE' with sections for 'WORKING FILES' and 'EXAMPLES'. Under 'EXAMPLES', there's a folder '.vscode' containing 'launch.json' and 'jsconfig.json', and a file 'todo-server.js' which is currently selected. The main area of the window shows the contents of 'todo-server.js':

```
1 var express = require("express");
2 var url = require("url");
3 var http = require("http");
4
5 var port = 3000;
6 var app = express();
7 http.createServer(app).listen(port);
8
9 var todos = [];
10 var t1 = { message : "Maths homework due", type : 1, deadline : "12/12/2015"};
11 var t2 = { message : "English homework due", type : 3, deadline : "20/12/2015"};
12 todos.push(t1);
13 todos.push(t2);
14
15 //clients requests todos
16 app.get("/todos", function (req, res) {
17     res.json(todos);
18 });
19
20 //add todo to the server
21 app.get("/adddtodo", function (req, res) {
```

At the bottom of the window, there's a status bar with icons for volume, battery, and signal strength, followed by 'Ln 8, Col 1' and 'UTF-8 LF JavaScript'.

Sending json & todo updating

The screenshot shows a Mac OS X desktop environment. At the top, there's a menu bar with 'QuickTime Player', 'File', 'Edit', 'View', 'Window', and 'Help'. To the right of the menu bar are system status icons. Below the menu bar is a window titled 'todo-server.js - Example5'. This window has a sidebar on the left labeled 'EXPLORE' with sections for 'WORKING FILES' and 'EXAMPLES'. Under 'EXAMPLES', there's a folder '.vscode' containing 'launch.json' and 'jsconfig.json', and a file 'todo-server.js' which is currently selected. The main area of the window shows the contents of 'todo-server.js':

```
1 var express = require("express");
2 var url = require("url");
3 var http = require("http");
4
5 var port = 3000;
6 var app = express();
7 http.createServer(app).listen(port);
8
9 var todos = [];
10 var t1 = { message : "Maths homework due", type : 1, deadline : "12/12/2015"};
11 var t2 = { message : "English homework due", type : 3, deadline : "20/12/2015"};
12 todos.push(t1);
13 todos.push(t2);
14
15 //clients requests todos
16 app.get("/todos", function (req, res) {
17     res.json(todos);
18 });
19
20 //add todo to the server
21 app.get("/adddtodo", function (req, res) {
```

At the bottom of the window, there's a status bar with icons for volume, battery, and signal strength, followed by 'Ln 8, Col 1' and 'UTF-8 LF JavaScript'.

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

```
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

```
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**

On the client: basic HTML

```
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

```
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

```
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

when the document is loaded, execute main()

On the client: JavaScript

```
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

this is Ajax

when the document is loaded, execute main()

On the client: JavaScript

```
1 var main = function () {  
2   "use strict";  
3  
4   // ...  
5   addTodosToList = function (todos) {  
6     t = document.getElementById("todo-list");  
Dynamic insert of list elements into the DOM  
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

this is Ajax

when the document is loaded, execute main()

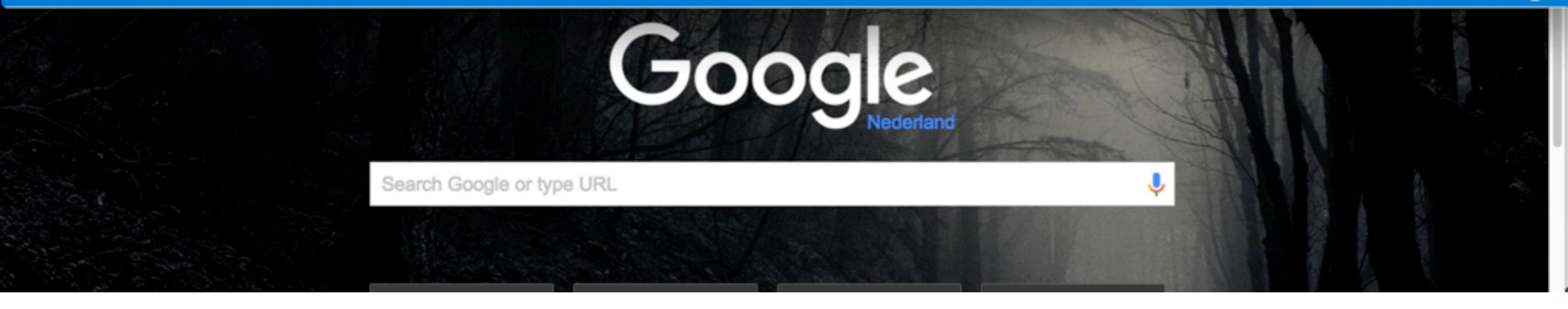
Example 6

Ajax

The screenshot shows a code editor window titled "index.html - Example6". The left sidebar has sections for "EXPLORE", "WORKING FILES", and "EXAMPLE6". Under "WORKING FILES", "index.html client" is selected. Under "EXAMPLE6", "index.html" is also selected. The main editor area contains the following HTML code:

```
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>
```

The status bar at the bottom shows "Ln 9, Col 1" and "UTF-8".



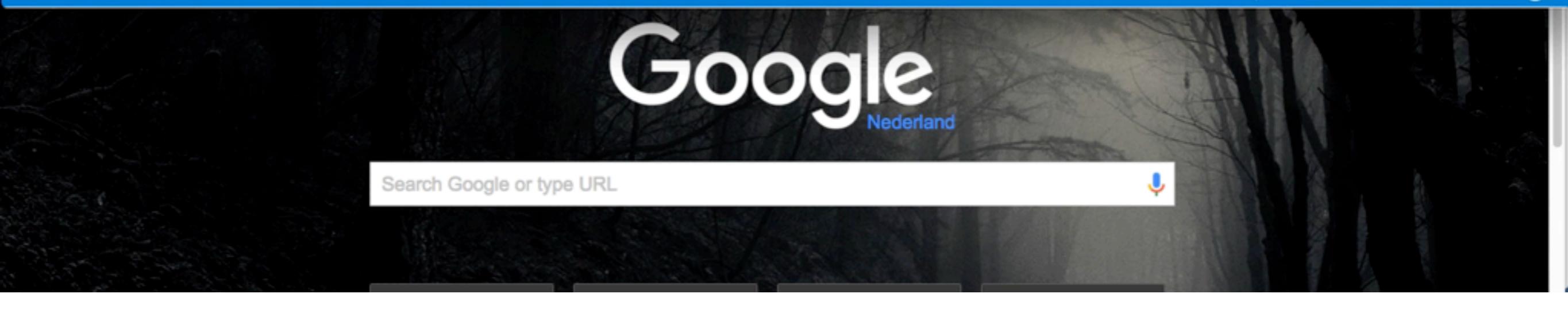
Example 6

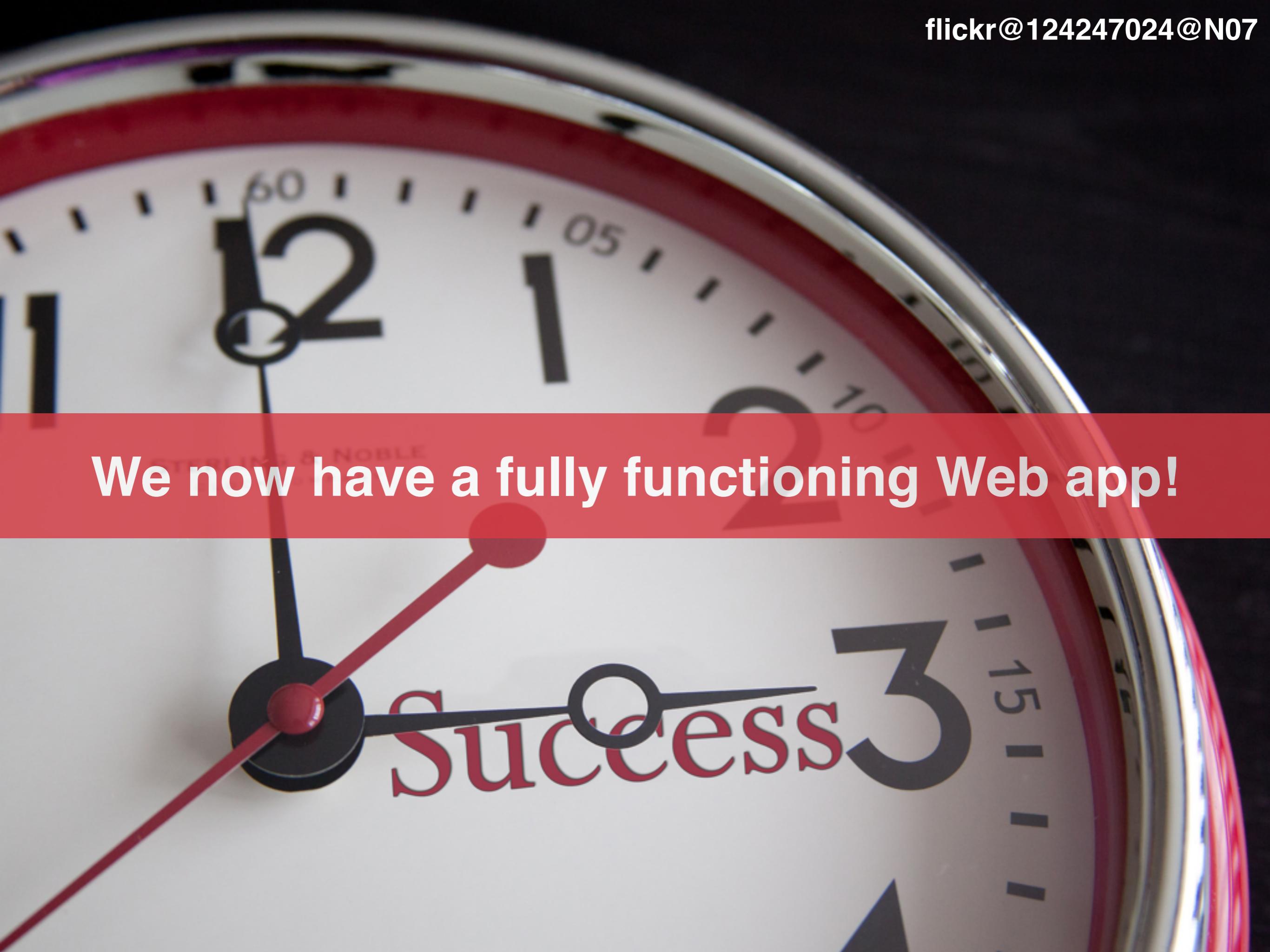
Ajax

The screenshot shows a code editor window titled "index.html - Example6". The left sidebar has sections for "EXPLORE", "WORKING FILES", and "EXAMPLE6". Under "WORKING FILES", "index.html client" is selected. Under "EXAMPLE6", "client" contains "html" and "javascript" subfolders, with "client-app.js" and "index.html" listed. The main editor area displays the following HTML code:

```
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>
```

The status bar at the bottom shows "Ln 9, Col 1" and "UTF-8".



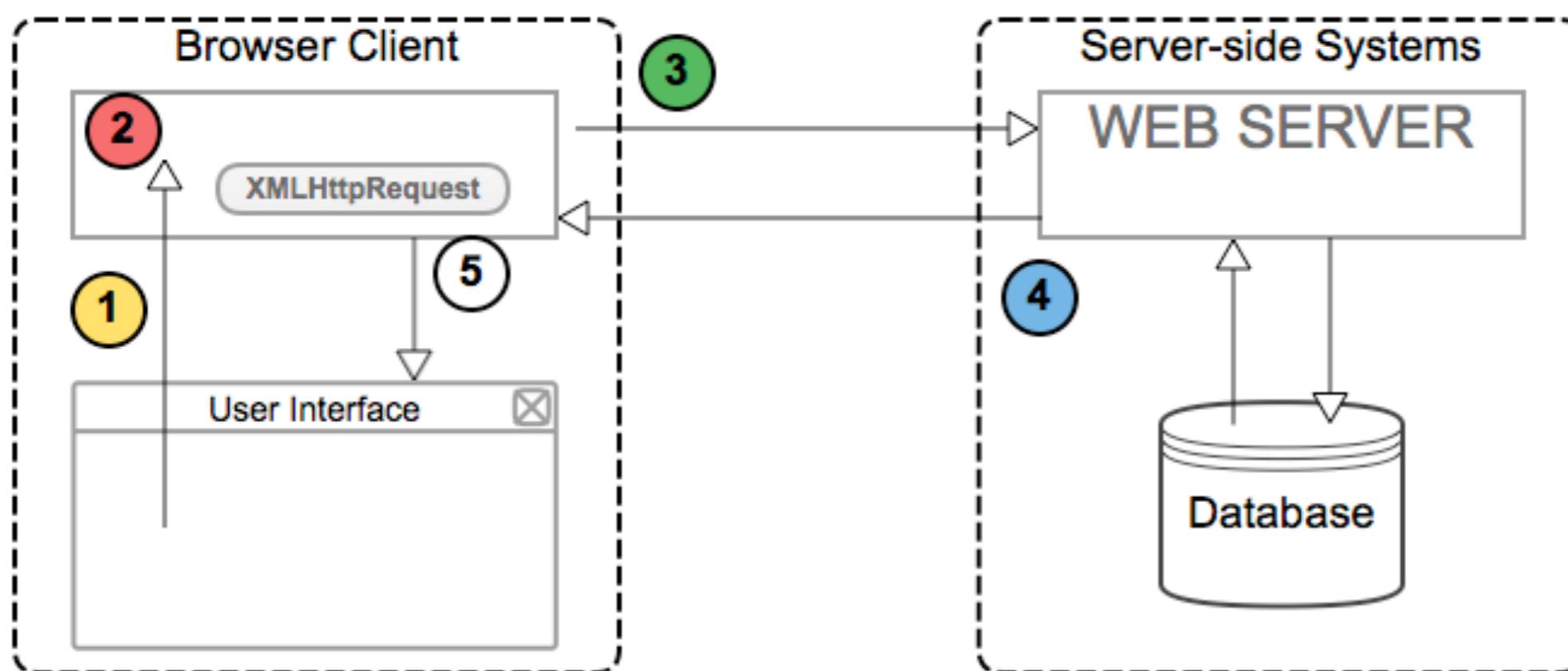


We now have a fully functioning Web app!

Ajax: how does it work?

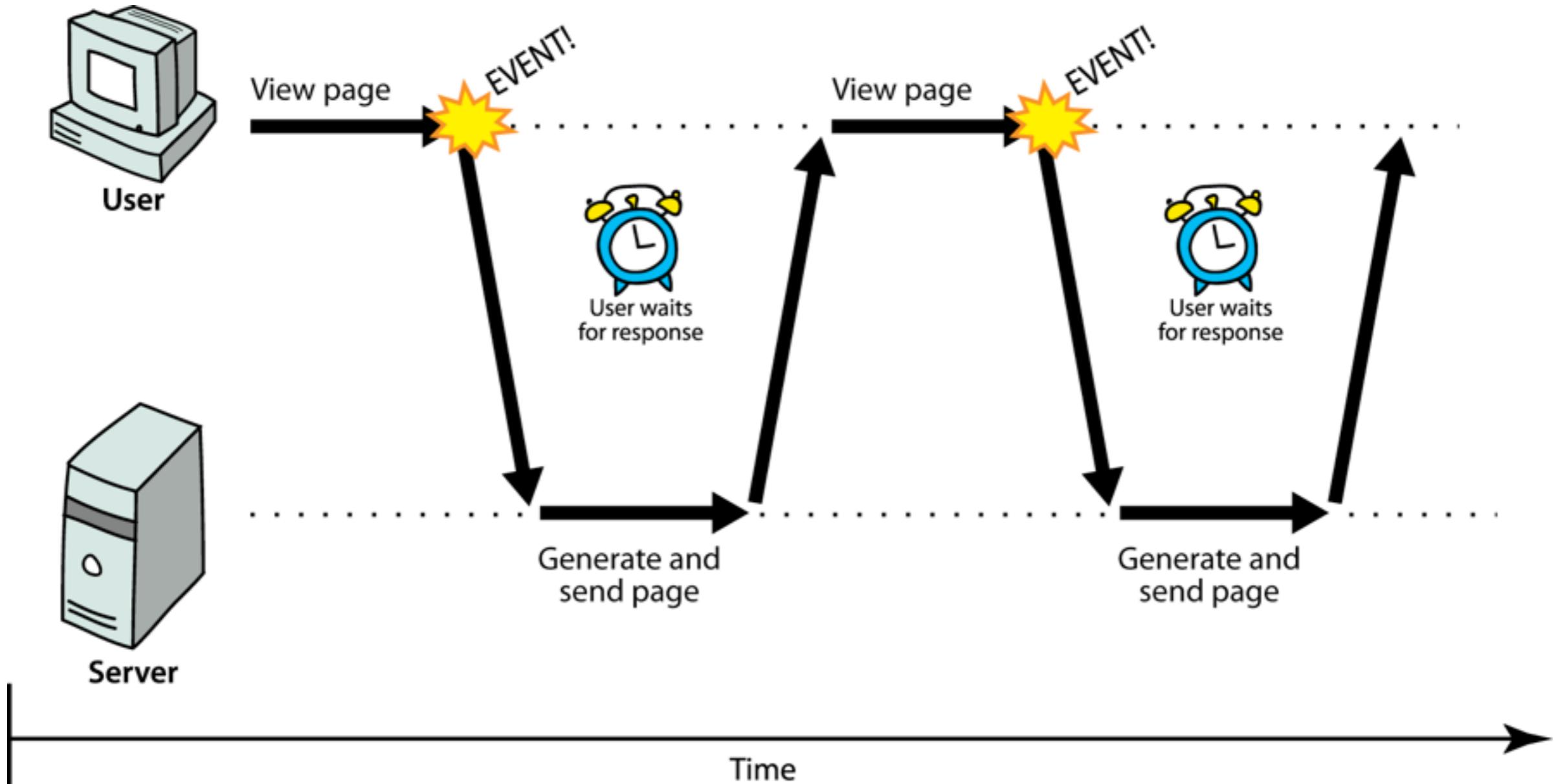
1. Web browser creates a **XMLHttpRequest** object
2. XMLHttpRequest **requests data** from a Web server
3. **Data is sent back** from the server
4. On the client, **JavaScript code injects the data** into the page

Ajax: how does it work?

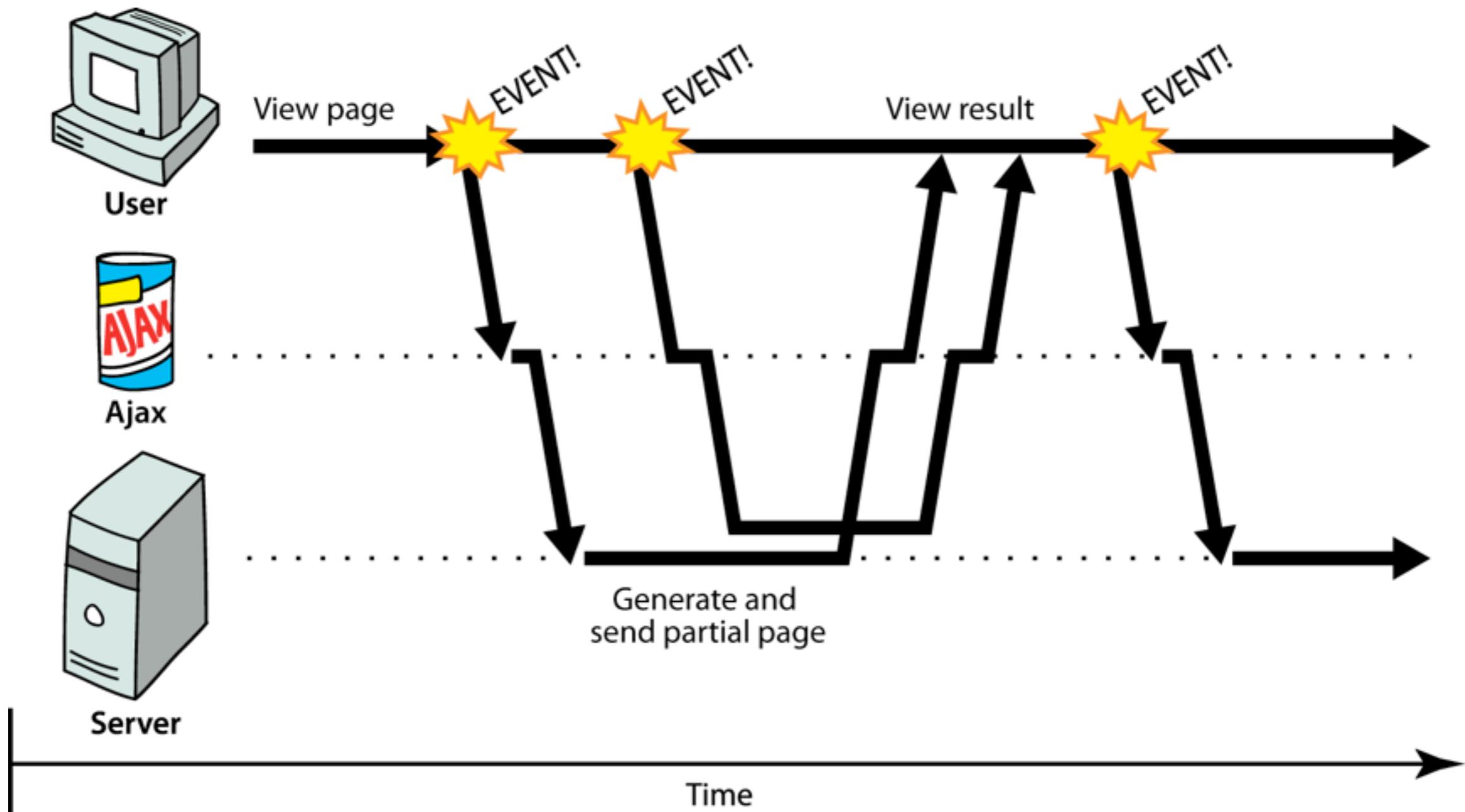


1. JavaScript call
2. XMLHttpRequest
3. HTTP request
4. Data returns
5. HTML & CSS data

Without Ajax ...



Ajax works differently



Ajax: synchronous request

```
1 //IE6 and prior IE versions use Microsoft.  
2 XMLHttpRequest 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

```
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

```
1 var ajax = new XMLHttpRequest();  
2 // event onreadystatechange is fired  
3 // when the status of the request changes. 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);
```

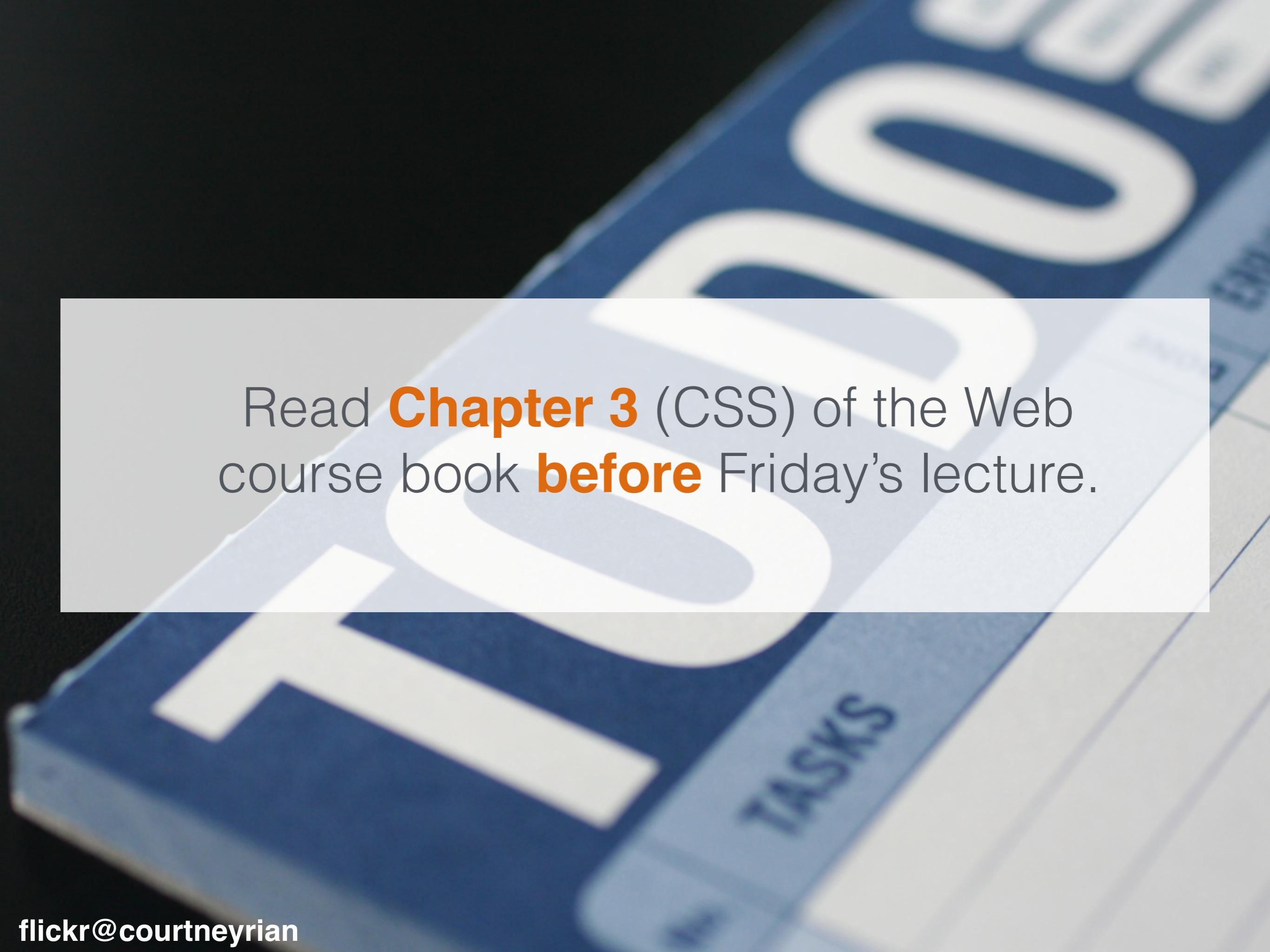
Ajax security

- 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.

End of Lecture