node.js: JavaScript on the server

TI1506: Web and Database Technology
Claudia Hauff

Lecture 4 [Web], 2014/15

Course overview [Web]

1. http: the language of Web communication
2. Web (app) design & HTML5
3. JavaScript: interactions in the browser
4. **node.js: JavaScript on the server**
5. CSS: Lets make things pretty
6. Ajax: asynchronous JavaScript
7. Personalization: Cookies & sessions
8. Securing your application

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-side code using Ajax
- **Implement** client/server communication via JSON

Lecture 1

A reminder before we start

Web servers and clients

(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 request message

plain text, line-oriented character sequences

```
GET / HTTP/1.1
Host: www.tudelft.nl
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9;
rv:31.0) Gecko/20100101 Firefox/31.0
Accept: text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Cookie:
__utma=1.2092357793611.16111.19805.2;utmcmd=(none);
```

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)

Why node.js?

A conscious choice

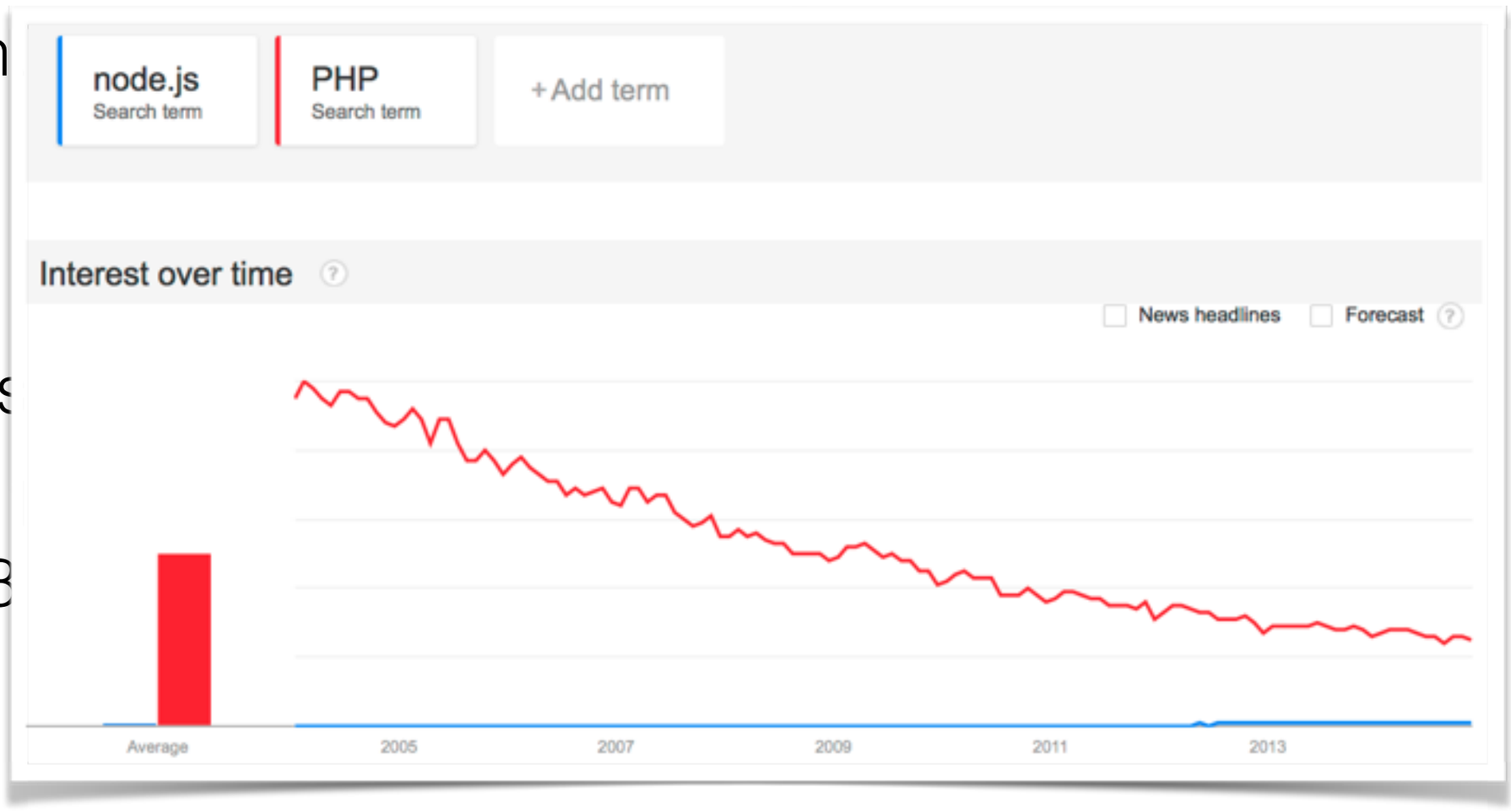
- Previous course years: PHP was taught as server-side technology
  - Established code base
  - A lot of support is available online
  - A huge number of libraries exist

- But:
  - Interest is declining
  - Tough to teach 4 languages (PHP, JavaScript, HTML, CSS) in 8 weeks

A conscious choice

- Previous course years: PHP was taught as server-side technology
  - Established code base
  - A lot of support is available online
  - A huge n

- But:
  - Interest is
  - Tough to
    CSS) in 8



What is node.js?

node.js in its own words …

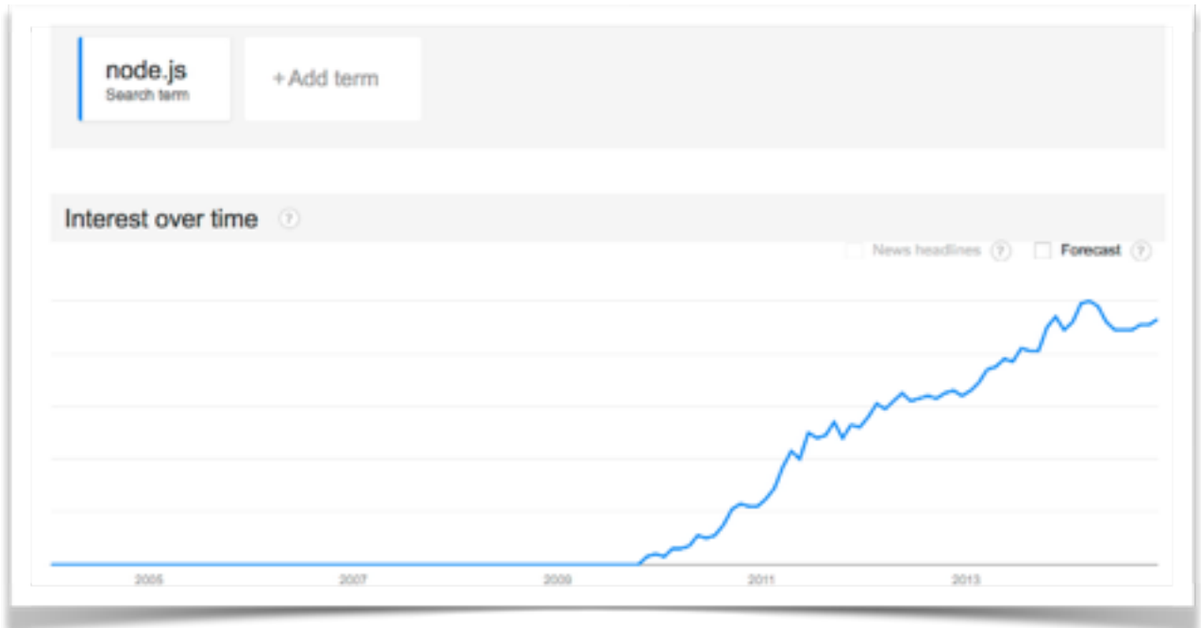
"Node.js® is a platform **built on Chrome's JavaScript runtime** for easily building **fast**, **scalable network** applications.

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."

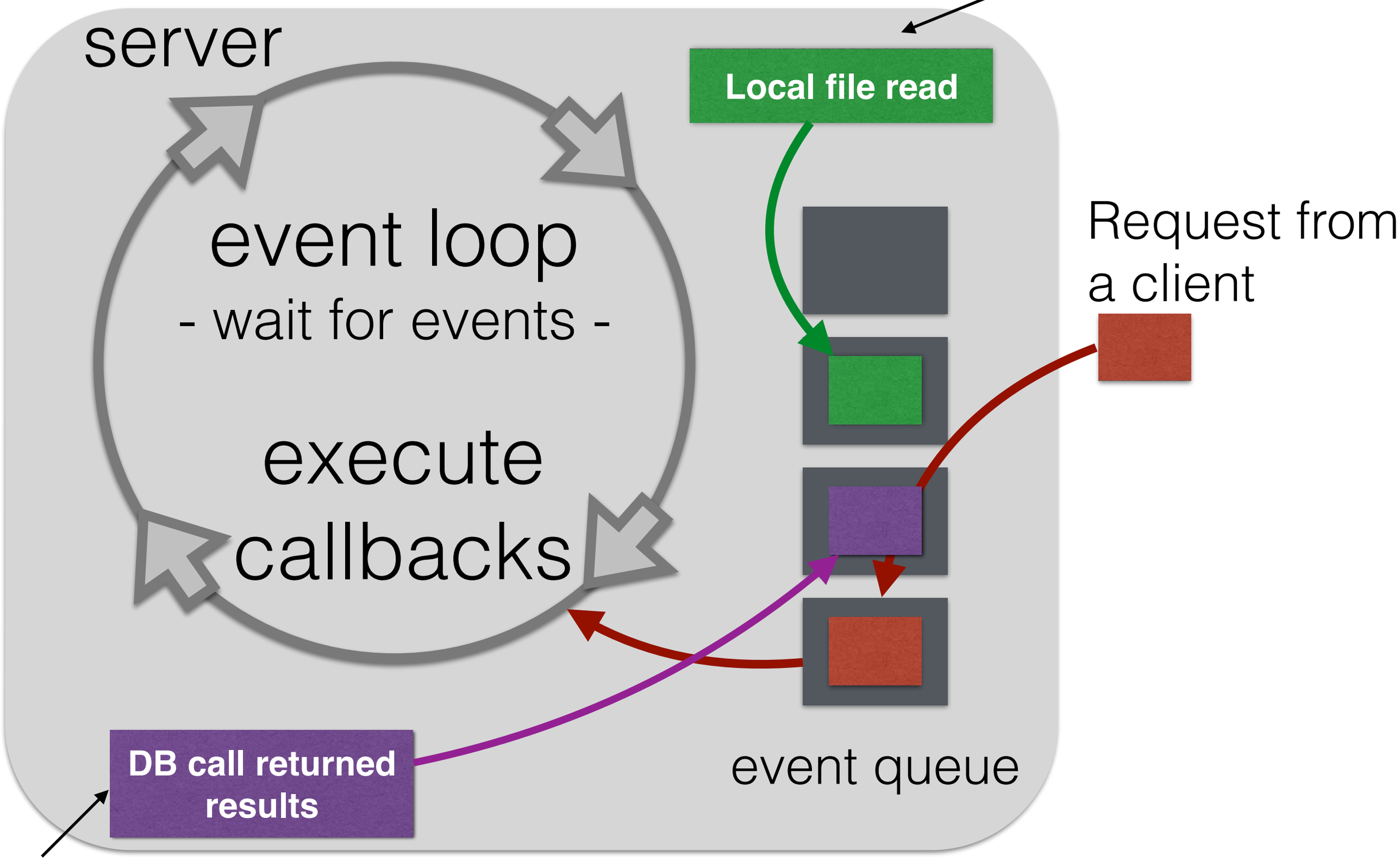
History of node.js

- A very young technology
- Google's JavaScript execution engine (V8) was open-sourced in 2008
- node.js builds on V8 and was first released in 2009
- Node.js' package manager (NPM) was released in 2011
- Native support for Windows in 2011
- Managed by a single project leader

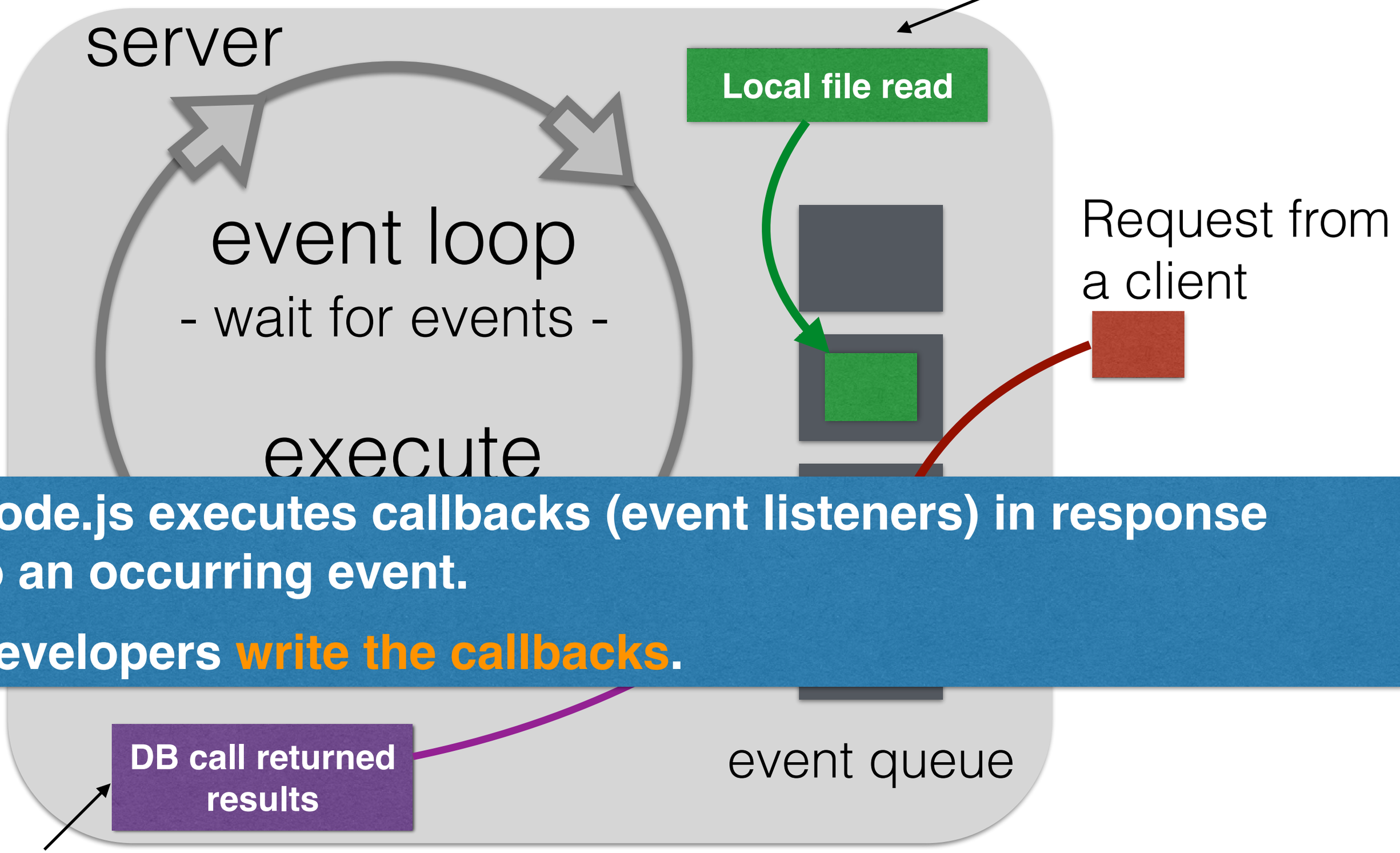
if we remove PHP from the trend line, things look better



node.js is event-driven



node.js is event-driven



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

Developers write the callbacks.

node.js: single-threaded but highly parallel

- **I/O bound programs**: programs constrained by data access (adding more CPUs or main memory will not lead to large speedups)

- Many tasks might require **waiting time**
  - Waiting for a database to return results
  - Waiting for a third party Web service
  - Waiting for connection requests

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

node.js: single-threaded but highly parallel

Blocking I/O (database example)

(1) read request

(2) process request & access the database

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

(4) process the next request

Non-blocking I/O

(1) read request

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

(3) do other things

(4) when the callback returns, process it

The first code examples

Lets start simple: our first node.js code

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

Lets start simple

part of the new ECMAScript Harmony

node.js fs module

polls ‘todos.txt’ for changes

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

- Node.js **module**: self-contained piece of code that provides reusable functionality
- `require()` usually returns a JavaScript object
- **Assumptions**:
  - Start node with —harmony option
  - File to watch must exist

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.

Low-level networking with node.js

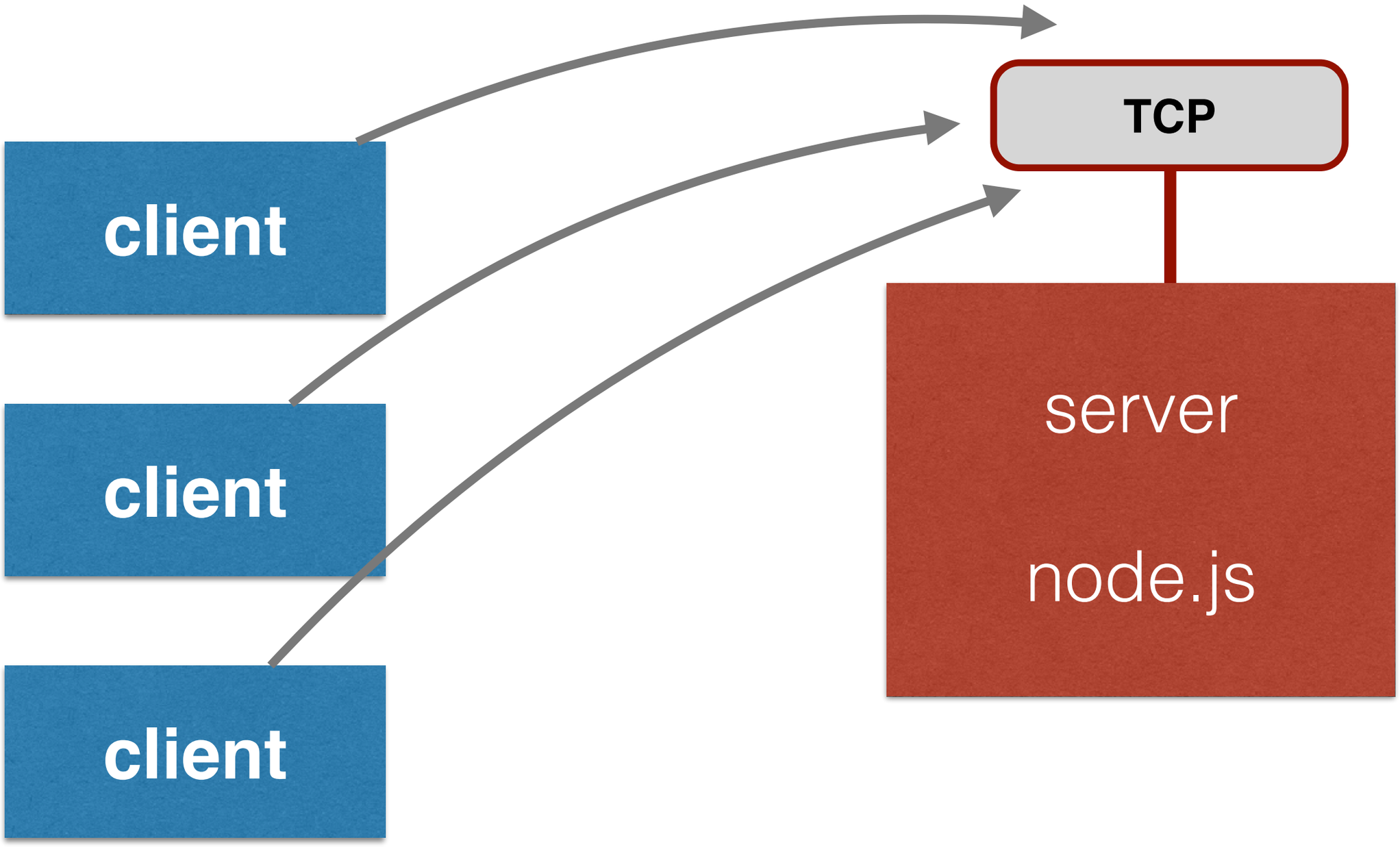
```
1 "use strict";
2 const
3   net = require('net'),
4   server = net.createServer(function(connection)
5   {
6       // use connection object for
7   });
8
9 server.listen(5432);
```

Server object is returned

callback function is invoked when another endpoint connects

bind to port 5432

Low-level networking with node.js



Low-level networking with node.js - lets add functionality

```
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 todos.txt for
10                       changes...\n");
11      // watcher setup
12      let watcher = fs.watch(filename, function() {
13         connection.write("File '" + filename + "'
14                          changed: " + Date.now() +
15                          "\n");
16      });
17      // cleanup
18      connection.on('close', function() {
19         console.log('Subscriber disconnected.');
20         watcher.close();
21      });
22 });
23 server.listen(5432, function() {
24   console.log('Listening for subscribers...');
25 });
```

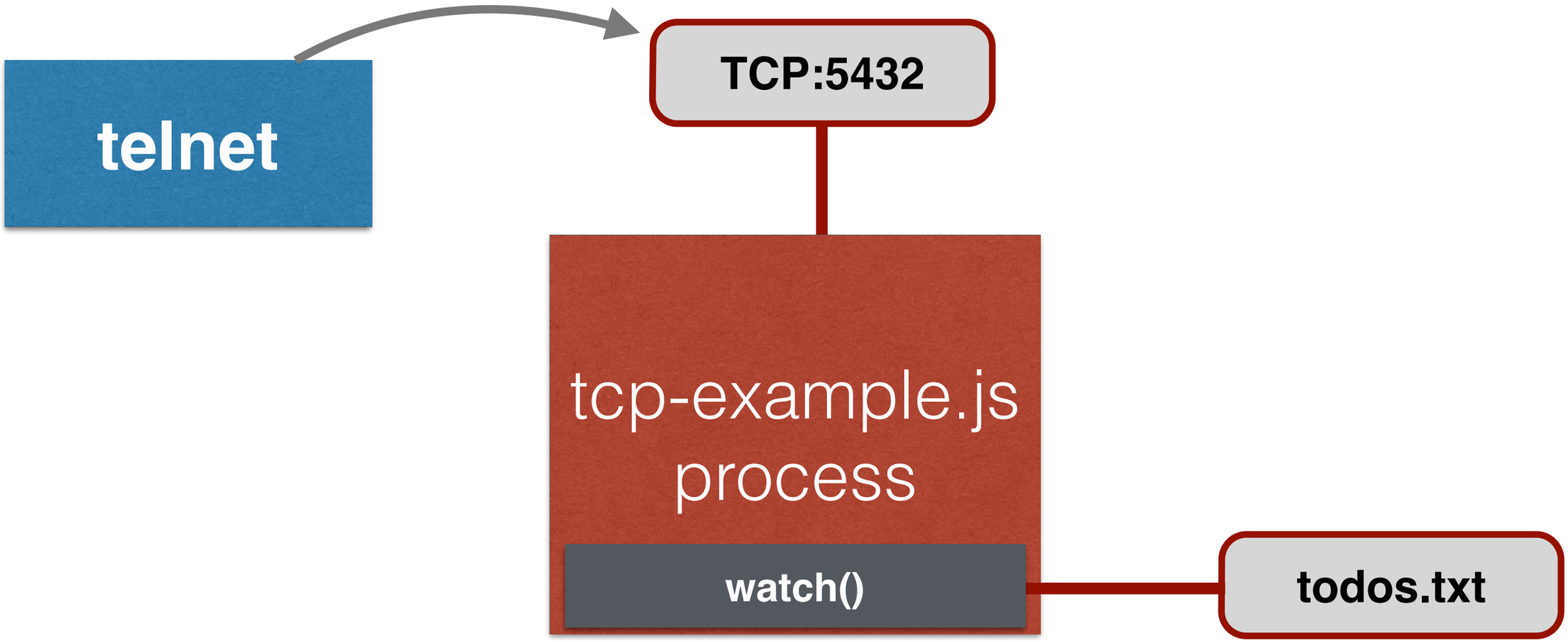
logs to the server

logs to the client

ile change info are sent to the client

client disconnects

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.

The “Hello World” of node.js

basic-server.js

node.js http module

A callback: what to do if a request comes in

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");
```

Create a HTTP response & send it

start the server

Start the server on the command line: **`$ node basic-server.js`**
Open the browser at: http://localhost:3000

The “Hello World” of node.js

basic-server2.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

basic-server2.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)

Nothing special about 3000

The “Hello World” of node.js

basic-server3.js

```
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);
```

our “response function”

response function as parameter

command line parameter

Using URLs for routing

basic-server4.js

```
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 say ok

we can extract params from the URL

otherwise send back a 404 error

Using URLs for routing

```
1  var http = require("http");
2  var url = require('url');
3  var server;
4
5  var simpleHTTPResponder = function (req, res) {
12          res.end("Greetings "+query["name"]);
13      }
21  }
22
23  server = http.createServer(simpleHTTPResponder);
24  var port = process.argv[2];
25  server.listen(port);
```

This is not getting any better, very tedious to write an HTTP server this way. It is too low-level.

How do you send CSS files and images?

Express

Express

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

```
$ npm install express
```

“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.crea
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 (r
20  res.send("Goodbye you!");
21 });
```

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

Express and its static file server

- **Static files**: files that are not created/changed on the fly
  - CSS
  - JavaScript (client-side)
  - HTML
  - Images, video, etc.

- A single line of code is sufficient to serve static files:
  `app.use(express.static(__dirname + "/static"));`

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

a static files are contained in this directory

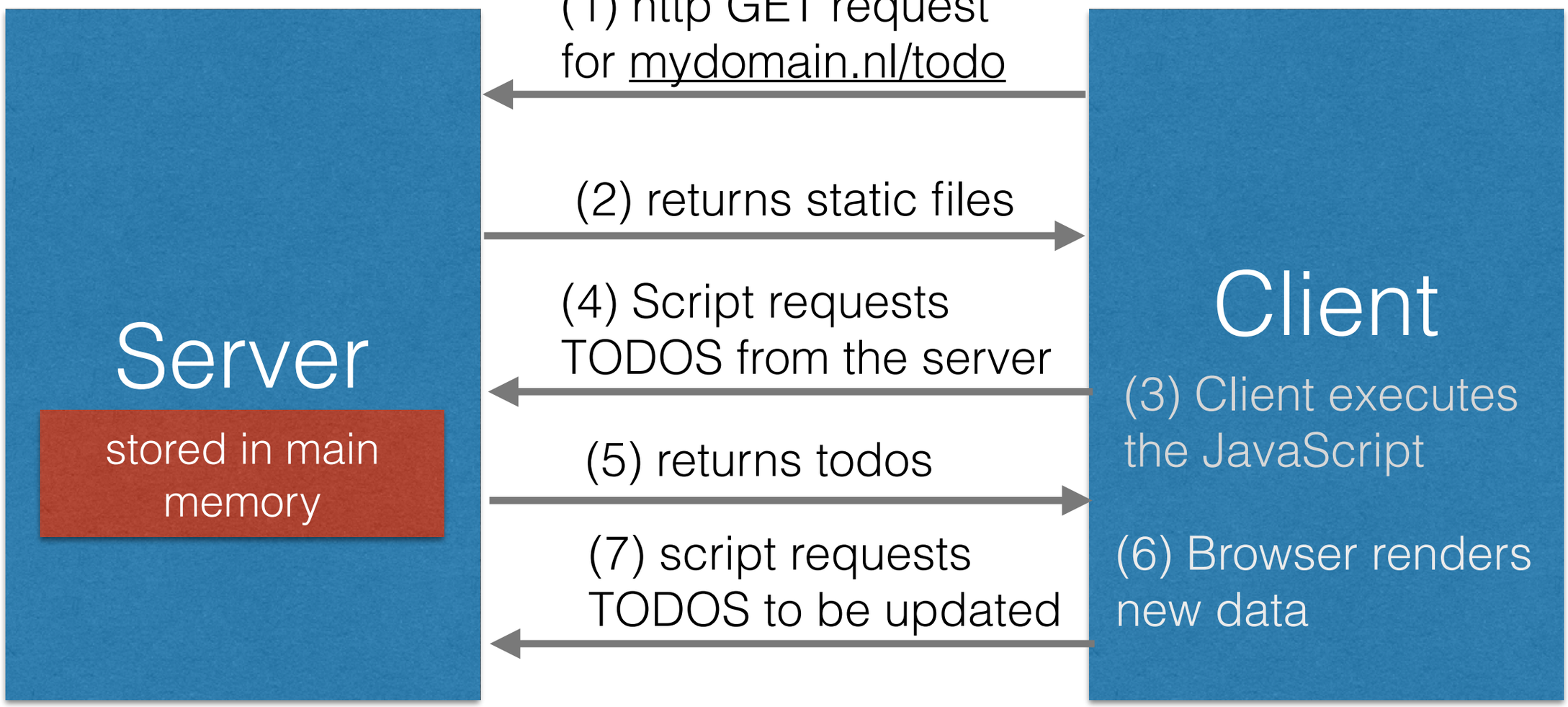
How to build a Web application

Development strategy

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

```
server.js
client/
    html/
        =>index.html
        =>error.html
    images/
        =>background.png
        =>logo.png
css/
    =>layout.css
    =>style.css
javascript/
    =>todos.js
```

TODO Web app flow



stored in a database
(next week)

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
- 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 ‘made’ into JSON with one 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`

Exchanging data: JSON

```
1 var todos = {};
2 var t1 = { message : "Maths homework due",
3            type : 1, deadline : "12/12/2014"};
4 var t2 = { message : "English homework due",
5            type : 3, deadline : "20/12/2014"};
6 todos[2] = t1;
7 todos[9] = t2;
8 JSON.stringify(todos);
```


```
{
   "31212":{
      "message":"Maths homework due",
      "type":1,
      "deadline":"12/12/2014"
   },
   "23232":{
      "message":"English homework due",
      "type":3,
      "deadline":"20/12/2014"
   }
}
```

On the server: sending JSON

basic-express4.js

```
1  var express = require("express");
2  var url = require("url");
3  var http = require("http");
4  var app;
5
6  var port = process.argv[2];
7  app = express();
8  http.createServer(app).listen(port);
9
10 var todos = {};
11 var t1 = { message : "Maths homework due", type
12           : 1, deadline : "12/12/2014"};
13 var t2 = { message : "English homework due",
14           type : 3, deadline : "20/12/2014"};
15 todos[31212] = t1;
16 todos[23232] = t2;
17
18 app.get("/todos", function (req, res) {
19  res.json(todos);
20 });
```

On the server: sending JSON

basic-express4.js

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

we store all TODOs on the server

Client requests the server's "copy" of the TODOs.

the client is sent the JSON formatted 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"], type :
6               query["type"], deadline : query[
7               "deadline"]};
8      var looping = 1;
9      while(looping>0) {
10        var r = getRInt(1,1000);//defined elsewhere
11        if( todos[r] == undefined) {
12           todos[r] = tx;
13           looping = -1;
14           console.log("Added “+tx.message);
15        }
16     }
17  }
18  res.end();
19 });
```

On the server: todo updating

```
1  app.get("/addtodo", function (req, res) {
2   var url_parts = url.parse(req.url, t
3   var query = url_parts.query;
4   if(query["message"]!=undefined) {
5      var tx = { message : query["message"], type :
6               query["type"], deadline : query[
7               "deadline"]};
8      var looping = 1;
9      while(looping>0) {
10        var r = getRInt(1,1000);//defined elsewhere
11        if( todos[r] == undefined) {
12           todos[r] = tx;
13           looping = -1;
14           console.log("Added “+tx.message);
15        }
16     }
17   }
18   res.end();
19 });
```

Is the key ‘message’ one of the URL’s parameters?

Rudimentary way of finding an index in our todo array.

Ajax: dynamic updating on the client

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="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-
6                                  list");
7      for (var key in todos) {
8           var li = document.createElement("li");
9           li.innerHTML = "TODO: "+todos[key].message;
10          todolist.appendChild(li);
11     }
12  };
13
14  $.getJSON("todos", addTodosToList);
15 }
16 $(document).ready(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-
6                                          list");
7      for (var key in todos) {
8         var li = document.createElement("li");
9         li.innerHTML = "TODO: "+todo.message;
10        todolist.appendChild(li);
11     }
12  };
13
14  $.getJSON("todos", addTodosToList);
15 }
16 $(document).ready(main);
```

Define what happens when a todo object is available (callback function)

Dynamic insert of list elements into the DOM

this is Ajax

when the call to /todos has returned, execute addTodosToList

when the document is loaded, execute main()

We now have a fully functioning Web app!

End of Lecture