Verteilte Systeme 1

Technologien des World Wide Web

christian.zirpins@hs-karlsruhe.de

JavaScript auf dem Server mit Node.js

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

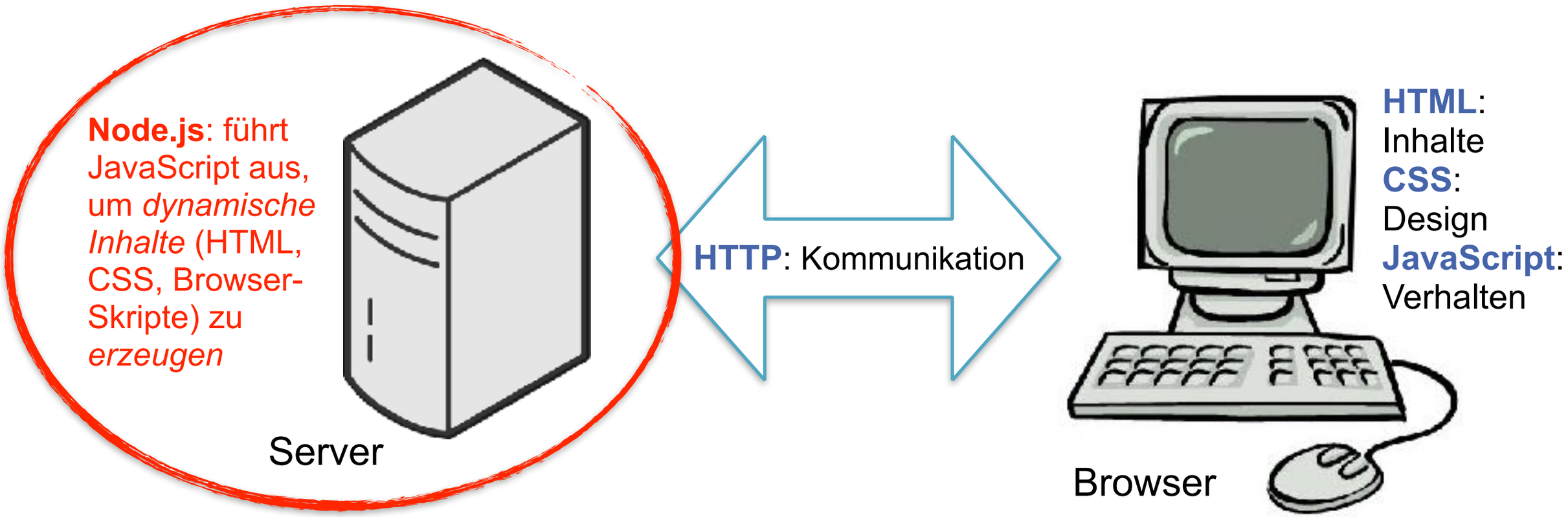
VS1 Termine im Sommer 2017

Termin (ca.)	Thema	Vorbereitung (Begleitbuch)	Raum
KW11: 13.03, 15.03	**HTTP**, die Sprache des Web		E 301/304
KW12: 20.03, 22.03	Web Apps mit **HTML5**	Web App Development Kapitel 2	E 301/304
KW13: 27.03, 29.03	Gestaltung von Web Apps mit **CSS3**	Web App Development Kapitel 3	E 301/304
KW14: 03.04, 05.04	***Übung**: Webseite mit HTML5/CSS3 erstellen*		LI 137
KW15: 10.04, 12.04	Browser Interaktion mit **JavaScript**	Web App Development Kapitel 4	E 301/304
KW16	Ostern		
KW17: 24.04, 26.04	***Übung**: Formulare mit JavaScript / HTML5 APIs*		LI 137
KW18	Maifeiertag		
KW19: 08.05, 10.05	JavaScript auf dem Server mit **Node.js**	Web App Development Kapitel 6	E 301/304
KW20: 15.05, 17.05	***Übung**: Node.js / Express Web App erstellen*		LI 137
KW21: 22.05, 24.05	Web Entwicklung mit **Ajax** & Co	Web App Development Kapitel 5	E 301/304
KW22: 29.05, 31.05	***Übung**: Web App mit REST und AJAX erweitern*		LI 137
KW23	Pfingsten		
KW24: 12.06, 14.06	Web Apps **Personalisieren**	Web App Development Kapitel 9	E 301/304
KW25: 19.06, 21.06	Web App **Sicherheit**		E 301/304
KW26: 26.06, 28.06	Klausurvorbereitung, Q&A		E 301/304

Heutige Lernziele

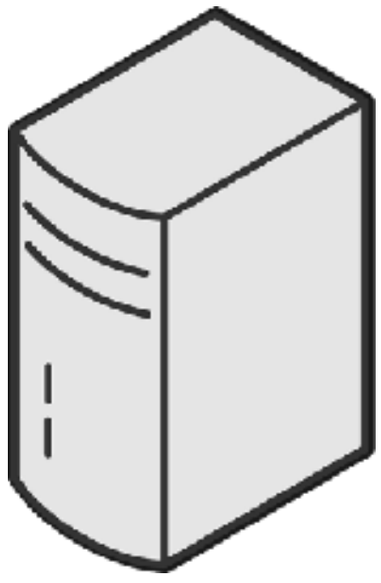
Nach dieser Vorlesung können Sie…

- …die wichtigsten Ideen hinter **node.js** erklären
- …grundlegende **Netzwerkfunktionalität** mit node.js implementieren
- …den Unterschied zwischen **node.js**, **NPM & Express** erläutern
- …eine voll funktionsfähige **Web-Anwendung** mit Client-und serverseitiger Interaktivität erstellen
- …clientseitigen Code mit **Ajax** implementieren
- …Client/Server-Kommunikation über **JSON** implementieren



Kurzer Rückblick auf die erste Vorlesung

Web Server und Clients



(1) HTTP Request

(2) HTTP Response

Anzeigen

Ausführen

Music Player

Acrobat Reader

- Server warten auf Daten-Anforderungen (Requests)
- Antworten tausenden Clients gleichzeitig
- Stellen **Web Ressourcen** bereit

- Clients sind meistens **Web Browser**
- Telnet

Web Ressource: jede Art von Inhalt mit einer Identität, wie statische Dateien (z.B. Text, Bilder, Video), Software Programme, Webcam etc.

Vorlesung 1

HTTP-Request Message

Nur Text: zeilenorientierte Zeichenfolgen

```
GET /home.html HTTP/1.1
Host: www.hs-karlsruhe.de
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
10.12; rv:47.0) Gecko/20100101 Firefox/47.0
Accept: text/html,application/xhtml+xml,
application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Cookie: _pk_id.1.ad89=0a5649e382c8b5e7…
Connection: keep-alive
Cache-Control: max-age=0
```


Vorlesung 1

HTTP-Response Message

```
HTTP/1.1 200 OK
```

Startzeile

```
Date: Thu, 22 Sep 2016 14:52:37 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.19
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Keep-Alive: timeout=5, max=500
Connection: Keep-Alive
Transfer-Encoding: chunked
```

Headerfelder

`name:wert`

```
.....
.....
```

Body (optional)

Was ist Node.js?

node.js in seinen eigenen Worten

"Node.js® ist eine Plattform, **basierend auf der JavaScript-Laufzeit von Chrome**, um schnell und einfach skalierbare Netzwerkanwendungen zu erstellen.

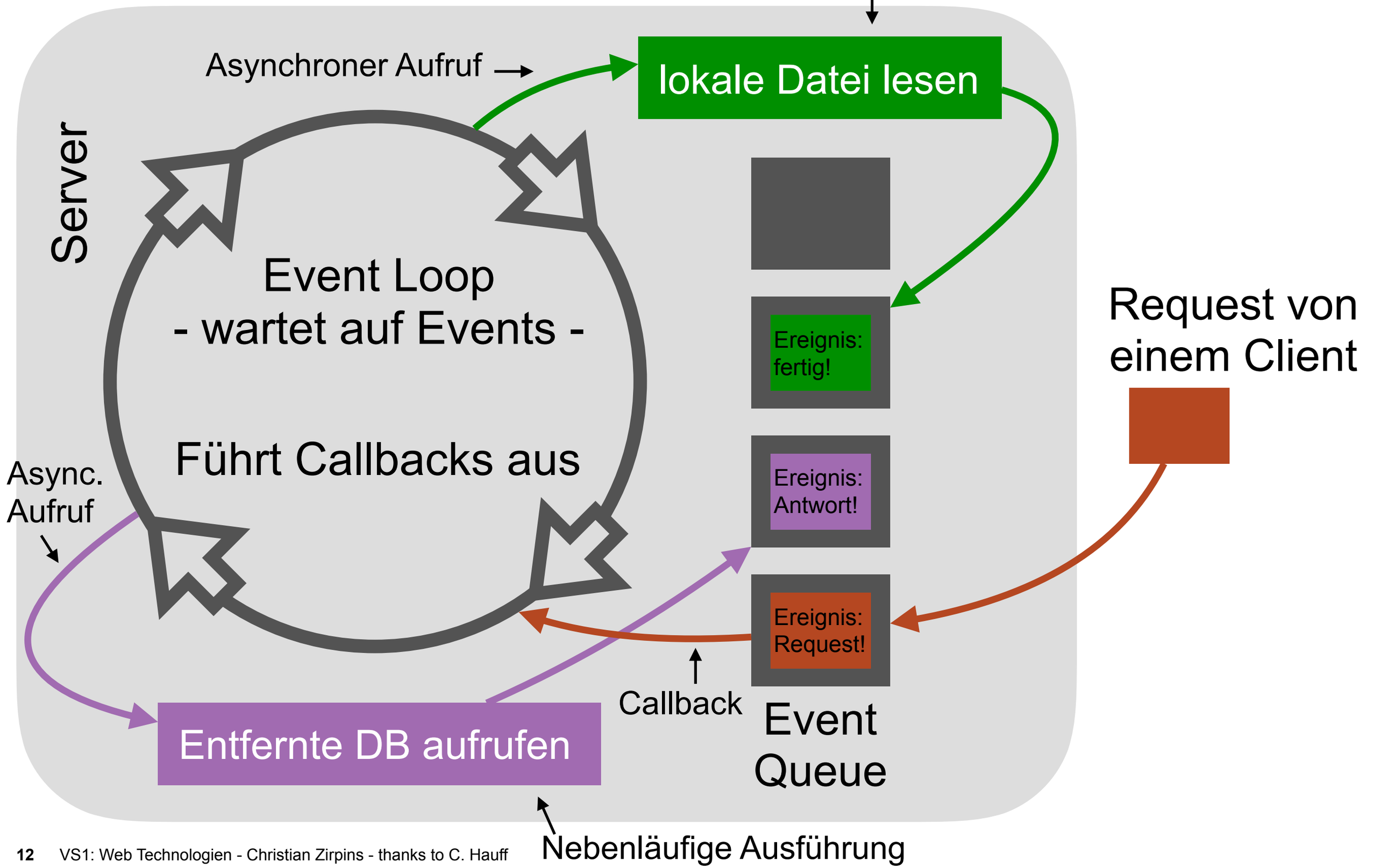
Node.js verwendet ein **ereignisgesteuertes**, **nicht-blockierendes E/A**-Modell, das es leichtgewichtig und effizient macht, ideal für **datenintensive Echtzeit-Anwendungen**, die über verteilte Geräte laufen."

Geschichte von node.js

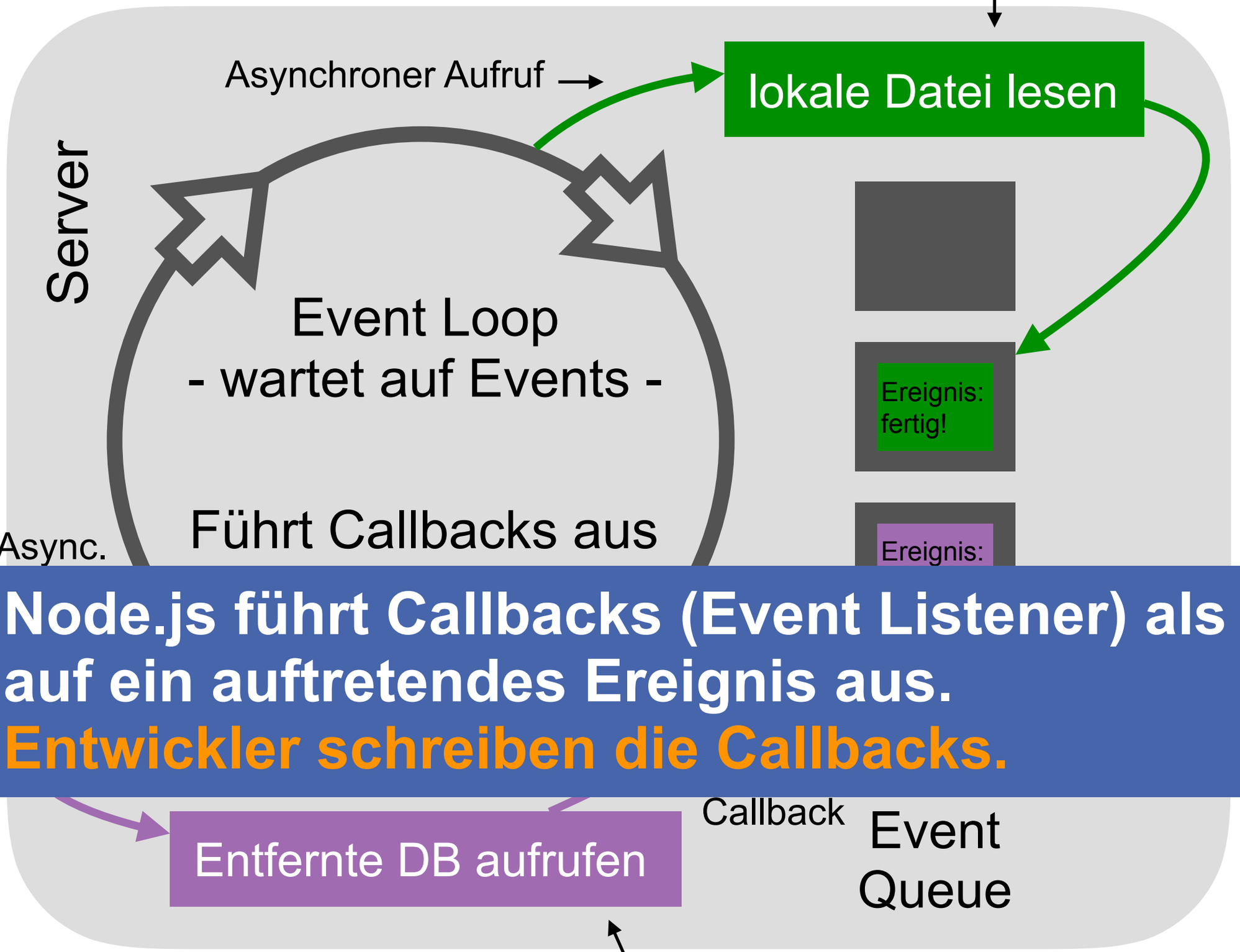
- Relativ junge Technologie
- JavaScript-Ausführungs-Engine von **Google (V8)** wurde 2008 als Open Source veröffentlicht
- **node.js** baut auf V8 auf und wurde erstmals 2009 von Ryan Dahl veröffentlicht
- Node.js **Paketmanager (NPM)** wurde 2011 veröffentlicht
- Native Unterstützung für **Windows** im Jahr 2011

"node.js" Interesse im zeitlichen Verlauf

node.js ist ereignisgesteuert



node.js ist ereignisgesteuert



Node.js führt Callbacks (Event Listener) als Reaktion auf ein auftretendes Ereignis aus.
Entwickler schreiben die Callbacks.

Node.js: single-threaded aber hoch parallel

E/A = Eingabe/Ausgabe
entspricht I/O = Input/Output

- **E/A-lastige Programme:** Programme, die durch Datenzugriff eingeschränkt sind (mehr CPUs oder Hauptspeicher führen nicht zu großer Beschleunigung)
- Viele Aufgaben erfordern **Wartezeit**!
  - Warten auf eine Datenbank, um Ergebnisse zurückzugeben
  - Warten auf einen Drittanbieter-Webdienst
  - Warten auf Verbindungsanforderungen

Node.js ist für diese Anwendungsfälle konzipiert

Node.js: single-threaded aber hoch parallel

Blockierendes E / A (Datenbankbeispiel)

(1) Request lesen
(2) Request verarbeiten & Zugriff auf die Datenbank durchführen
(3) warten, bis die Datenbank Daten zurückgibt und diese verarbeiten
(4) nächsten Request verarbeiten

Nicht blockierendes E / A

(1) Request lesen
(2) Request verarbeiten & Zugriff auf die Datenbank mit Callback durchf.
(3) andere Dinge tun
(4) wenn der Callback zurückkehrt, diesen verarbeiten

Erste Code Beispiele

Codebeispiele im Laborprojekt
https://github.com/zirpins/vs1lab/tree/master/Beispiele

Ganz einfach: unser erstes node.js Skript

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

Ganz einfach

Teil von ECMAScript Harmony

Node.js fs Modul

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

Überwacht 'todos.txt' auf Änderungen

- Node.js **Modul**: in sich geschlossener Code, der wiederverwendbare Funktionalität enthält (Bibliothek)
- `require()` gibt normalerweise ein JavaScript-Objekt zurück
- **Annahmen**:
  - Starten Sie node.js mit der Option `-harmony`
  - Die zu beobachtende Datei muss vorhanden sein

Netzwerkprogrammierung mit node.js

- Speziell für **verteilte Programmierung** (nicht nur Webprogrammierung!)
- Node.js verfügt über integrierte Unterstützung für **low-level** Socket-Verbindungen (TCP-Sockets)
- TCP-Socket-Verbindungen haben **zwei Endpunkte**!
  - 1. **bindet** an einen nummerierten Port
  - 2. **verbindet** sich mit einem Port

Analoges Beispiel: Telefonleitungen

Ein Telefon "bindet sich" an eine Telefonnummer.

Ein anderes Telefon versucht, dieses Telefon anzurufen.

Wenn der Anruf angenommen wird, wird eine Verbindung aufgebaut.

Low-level Netzwerkprogrammierung mit node.js

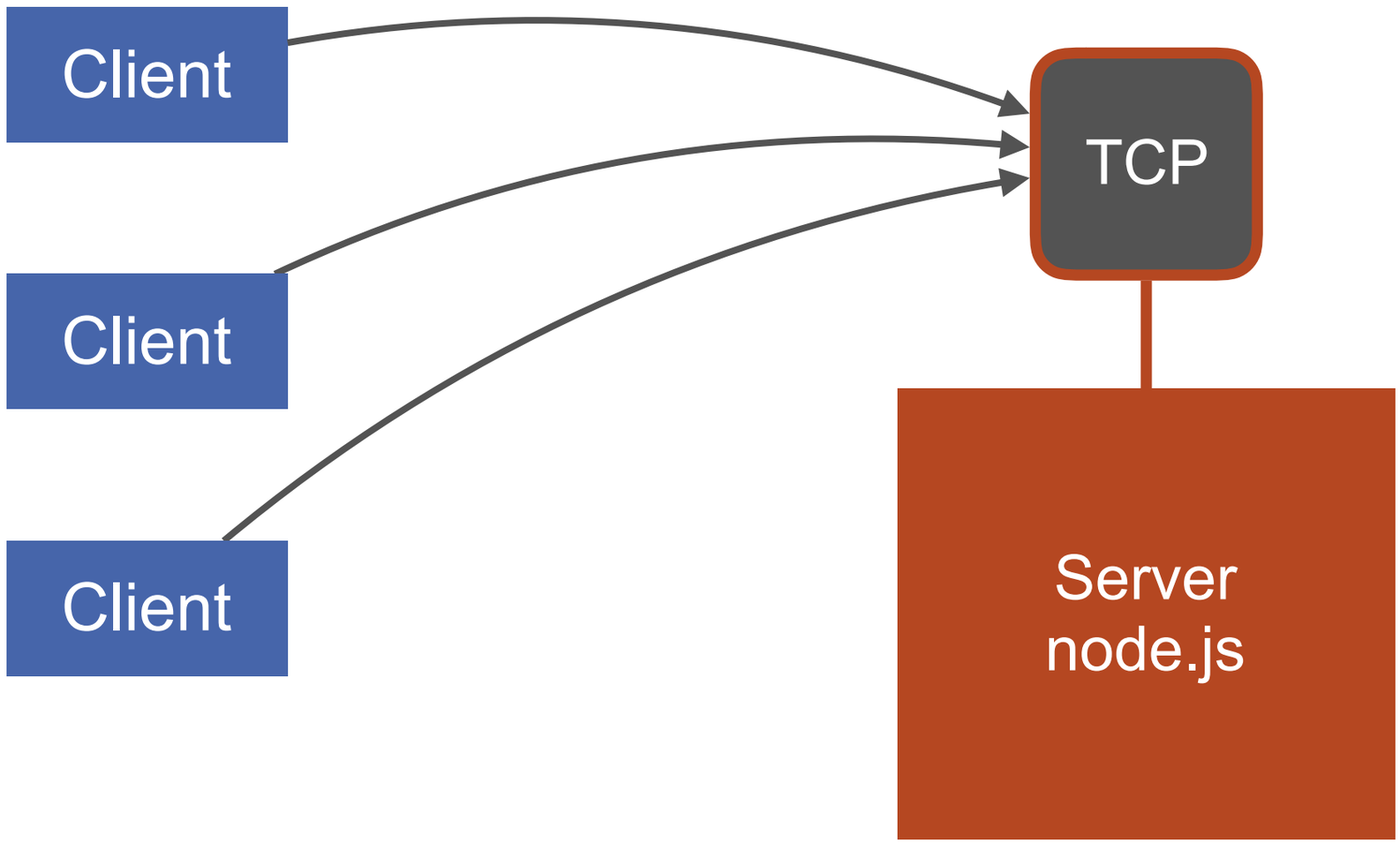
```
"use strict";
const
    net = require("net"),
    server = net.createServer(function(connection) {
        // nutze connection Objekt
    });
server.listen(5432);
```

Server Objekt wird zurückgegeben

An Port 5432 binden

Callback Funktion wird aufgerufen, wenn ein anderer Endpunkt sich verbindet

Low-level Netzwerkprogrammierung mit node.js



Low-level Netzwerkprogrammierung mit node.js

```
"use strict";
const
    fs = require('fs'),
    net = require('net'),
    filename = "todos.txt",
    server = net.createServer(function(connection) {
        console.log('Subscriber connected.');
        connection.write("Now watching todos.txt for
                          changes...\n");
        // watcher setup
        var watcher = fs.watch(filename, function() {
            connection.write("File '" + filename + "'
                             changed: " + Date.now() + "\n");
        });
        // cleanup
        connection.on('close', function() {
            console.log('Subscriber disconnected.');
            watcher.close();
        });
    });
server.listen(5432, function() {
    console.log('Listening for subscribers...');
});
```

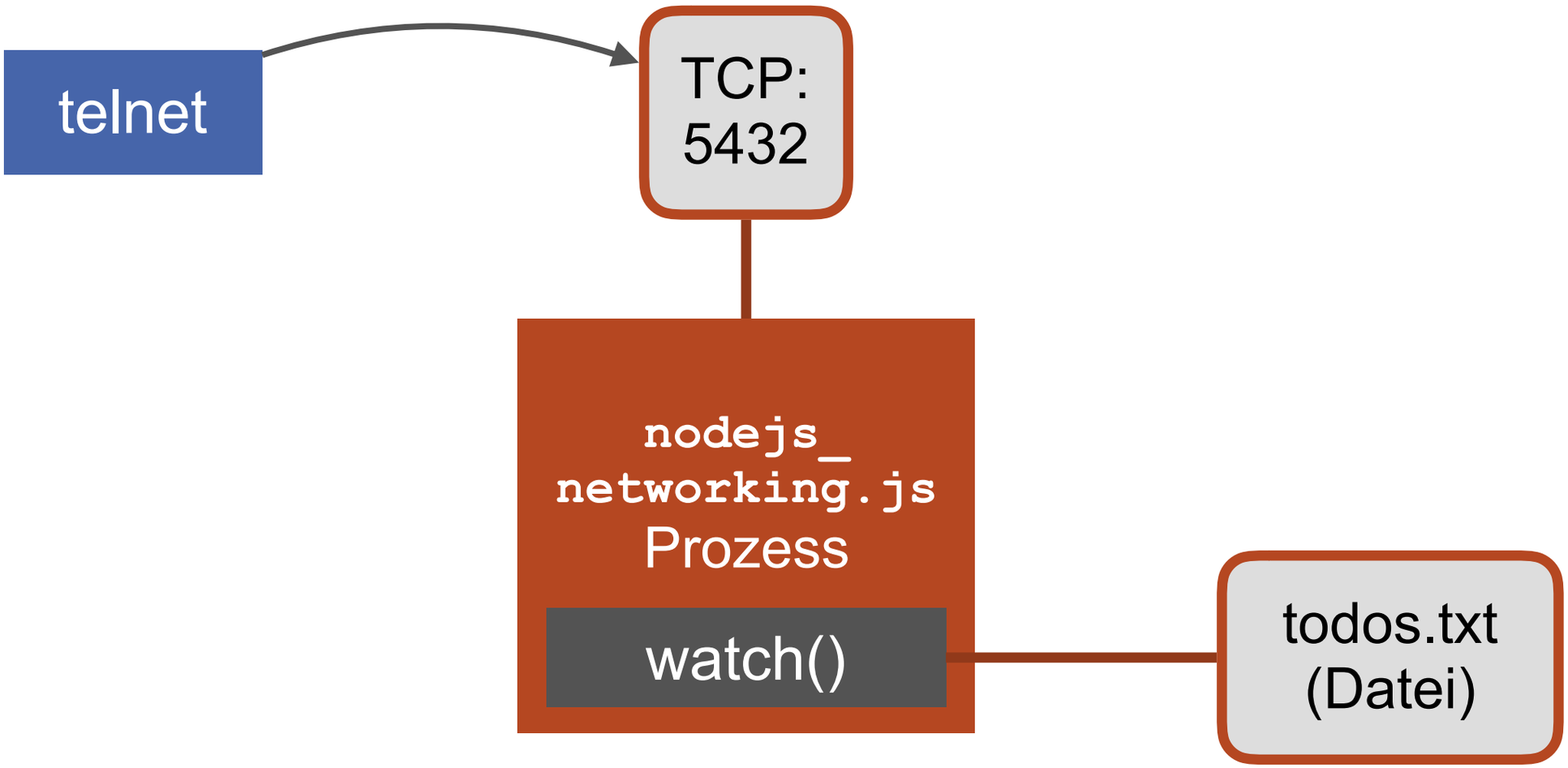
Log auf Server

Log auf Client

Dateiänderungen werden zum Client gesendet

Client beendet Verbindung

Low-level Netzwerkprogrammierung mit node.js



Web Server mit node.js erstellen

node.js ist selbst kein Web Server,
hat aber die Mittel, um einen zu erstellen

'Hello World', node.js Style

basic-server.js

```
var http = require("http");
var server;

server = http.createServer(function(req, res) {
    res.writeHead(200, {
        "Content-Type": "text/plain"
    });
    res.end("Hello World!");
    console.log("HTTP response sent");
});

server.listen(3000);
console.log("Server listening on port 3000");
```

node.js http Modul

Web Server erstellen

Callback: was bei einem Request getan werden soll

Erstelle HTTP Response & sende zurück

Web Server starten

Server starten: `$ node basic-server.js`
Browser öffnen unter http://localhost:3000

'Hello World', node.js Style

basic-server2.js

```
var http = require("http");
var server;
var sentCounter = 0;

 server = http.createServer(function(req, res) {
     res.writeHead(200, {
         "Content-Type": "text/plain"
     });
     res.end("HelloWorld!");
     sentCounter++;
     console.log(sentCounter + " HTTP responses
                 sent in total");
});

var port = 2345;
server.listen(port);
console.log("Server listening on port " + port);
```

'Hello World', node.js Style

basic-server2.js

```
var http = require("http");
var server;
var sentCounter = 0;

 server = http.createServer(function(req, res) {
     res.writeHead(200, {
         "Content-Type": "text/plain"
     });
     res.end("HelloWorld!");
     sentCounter++;
     console.log(sentCounter + " HTTP responses
                 sent in total");
});

var port = 2345;
server.listen(port);
console.log("Server listening on port " + port);
```

Standard JavaScript: Variablen, Funktionen, Objekte u.a. können hinzugefügt werden

HTTP Request Objekt

HTTP Response Objekt

Setze HTTP-Status-Code und -Header als Objekt

Verschiedene Ports möglich

'Hello World', node.js Style

basic-server3.js

```
var http = require("http"),
    server;

var simpleHTTPResponder = function(req, res) {
    res.writeHead(200, {
        "Content-Type": "text/plain"
    });
    sentCounter++;
    res.end("'Hello World' for the " + sentCounter
        + ". time!");
    console.log(sentCounter
        + " HTTP responses sent in total");
}

var sentCounter = 0;
server = http.createServer(simpleHTTPResponder);

var port = process.argv[2];
server.listen(port);
console.log("Server listening on port " + port);
```

Response Funktion

Response Funktion als Parameter

Kommandozeilenparameter

URLs für das "Routing"

basic-server4.js

```
var http = require("http");
var url = require("url");
var server;

var simpleHTTPResponder = function(req, res) {
    var url_parts = url.parse(req.url, true);
    if (url_parts.pathname == "/greetme") {
        res.writeHead(200, {"Content-Type": "text/plain"});
        var query = url_parts.query;
        if (query["name"] != undefined) {
            res.end("Greetings " + query["name"]);
        } else {
            res.end("Greetings Anonymous");
        }
    } else {
        res.writeHead(404, {"Content-Type": "text/plain"});
        res.end("Only /greetme is implemented.");
    }
}

server = http.createServer(simpleHTTPResponder);
var port = process.argv[2];
server.listen(port);
```

Für den Pfad "/greetme" antworten wir mit OK

Query Parameter extrahieren

Ansonsten senden wir einen 404 Fehler

URLs für das "Routing"

basic-server4.js

```
var http = require("http");
var url = require("url");
var server;

var simpleHTTPResponder = function(req, res) {
    var url_parts = url.parse(req.url, true);
    if (url_parts.pathname == "/greetme") {
        }
    } else {
        res.writeHead(404, {"Content-Type": "text/plain"});
        res.end("Only /greetme is implemented.");
    }
}
```

Dies wird nicht mehr besser… Sehr mühsam, einen HTTP-Server auf diese Weise zu schreiben. **Es ist zu "low-level".**

Was, wenn noch CSS-Dateien und Bilder hinzukommen?

Das Express Framework

Express

- Der node.js **Kern** hat eine kleine Code Basis
- node.js besitzt standardmäßig einige **Kern Module** (z.B. http)
- Express gehört nicht dazu (aber es gibt NPM)

```
$ npm install express
```

"Das Express-Modul erstellt eine Ebene über dem Kern-HTTP-Modul, die viele komplexe Dinge behandelt, die wir nicht selbst behandeln wollen, wie das Bereitstellen von statischen HTML-, CSS- und clientseitigen JavaScript-Dateien." (LWAD, Kap. 6)

'Hello World', Express Style

```
var express = require("express");
var url = require("url");
var http = require("http");
var app;

var port = process.argv[2];
app = express();
http.createServer(app).listen(port);

app.get("/greetme", function(req, res) {
    var query = url.parse(req.url, true).query;
    var name = (query["name"] != undefined) ?
        query["name"] : "Anonymous";
    res.send("Greetings " + name);
});

app.get("/goodbye", function(req, res) {
    res.send("Goodbye you!");
});
```

'Hello World', Express Style

```
var express = require("express");
var url = require("url");
var http = require("http");
var app;

var port = process.argv[2];
app = express();
http.createServer(app).listen(port);

app.get("/greetme", function(req, res) {
    var query = url.parse(req.url, true).query;
    var name = (query["name"] != undefined) ?
        query["name"] : "Anonymous";
    res.send("Greetings " + name);
});

app.get("/goodbye", function(req, res) {
    res.send("Goodbye you!");
});
```

app Objekt kapselt Express Funktionen

URL "Route" Konfiguration

Noch eine Route

Express erzeugt die HTTP Header

Express und HTML…

```
var express = require("express");
var url = require("url");
var http = require("http");
var app;

var port = process.argv[2];
app = express();
http.createServer(app).listen(port);

app.get("/greetme", function(req, res) {
    var query = url.parse(req.url, true).query;
    var name = (query["name"] != undefined) ?
        query["name"] : "Anonymous";
   res.send("<html><head></head><body><h1> Greetings "
          + name + "</h1></body></html>");
});

app.get("/goodbye", function(req, res) {
    res.send("Goodbye you!");
});
```

Fehleranfällig, nicht wartbar, scheitert bei jedem ernsthaften Projekt

Express und sein statischer Dateiserver

- **Statische Dateien**: Dateien die nicht zur Laufzeit erzeugt oder geändert werden
  - CSS
  - JavaScript (clientseitig)
  - HTML
  - Bilder, Videos, etc.
- Eine einzelne Zeile Code genügt, um statische Dateien bereitzustellen:

```
app.use(express.static(__dirname + "/static"));
```

Alle statischen Dateien liegen in diesem Ordner

- Express prüft immer **zuerst** die statischen Dateien für eine Route, erst wenn dies misslingt, werden die dynamischen Routen geprüft.

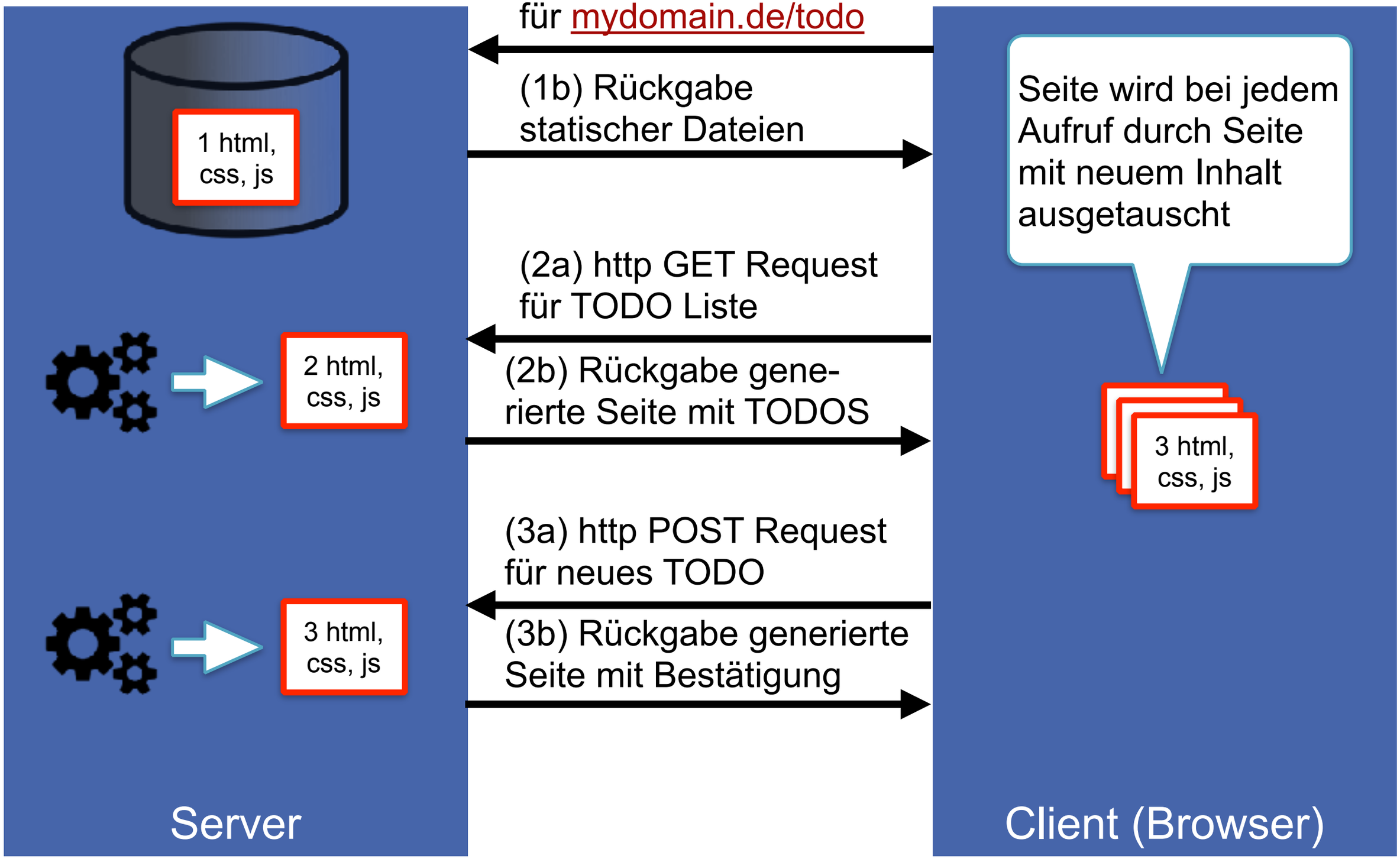
Wie baut man eine Web Anwendung?

Entwicklungsstrategie

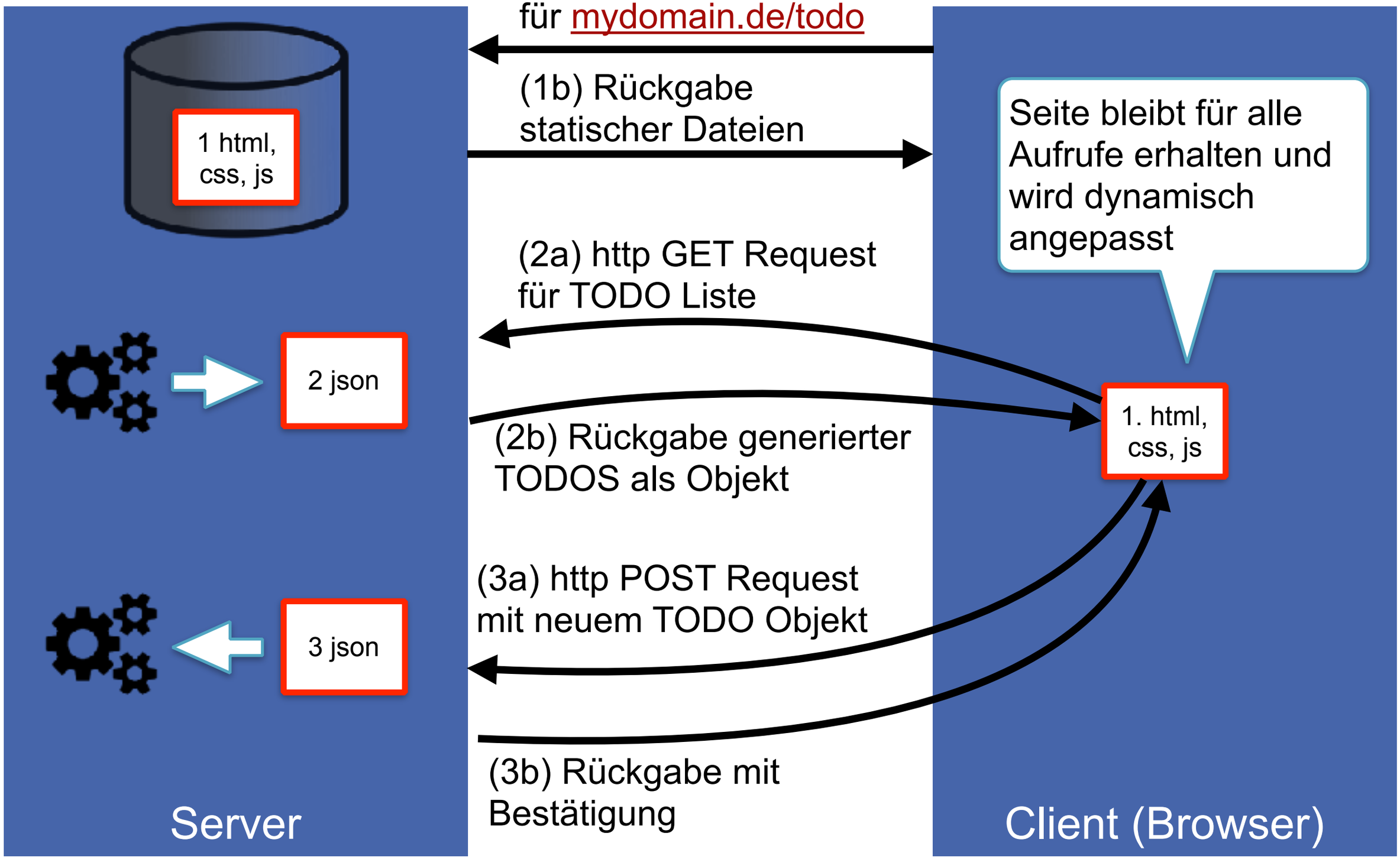
- Entwickele den **clientseitigen Code** (HTML, CSS, JavaScript)
- Platziere alle Dateien in ein Verzeichnis (z.B. `/client`) **auf dem Server**
- Definiere den **node.js Servercode** in einer *.js Datei mit Express
- Setze den **statischen Dateipfad** auf das Verzeichnis aus Schritt 2
- Füge Interaktionen zwischen Client und Server hinzu

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

Multi-Page Apps



Single-Page Apps



JSON: Daten zwischen Client und Server austauschen

Datenaustausch: JSON

JavaScript Object Notation

- In früheren Jahren wurde **XML** als Datenaustauschformat verwendet - solide, aber nicht einfach zu handhaben
- XML ist in der Praxis oft zu sperrig
- **JSON** ist viel kleiner als XML
- JSON kann vollständig mit eingebauten JavaScript-Befehlen analysiert werden
- JavaScript-Objekte können mit einem Aufruf in JSON gewandelt werden

JSON vs. XML

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>
```

JSON

```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 vs. JavaScript Objekte

- JSON: alle Objekt-Eigenschaftsnamen müssen in Anführungszeichen eingeschlossen werden
- JSON-Objekte **haben keine Funktionen** als Eigenschaften
- Jedes JavaScript-Objekt kann über **`JSON.stringify`** in JSON umgewandelt werden

Datenaustausch: JSON

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

```
{
    "2": {
        "message": "Maths homework due",
        "type": 1,
        "deadline": "12/12/2014"
    },
    "9": {
        "message": "English homework due",
        "type": 3,
        "deadline": "20/12/2014"
    }
}
```

Auf dem Server: JSON senden

```
var express = require("express");
var url = require("url");
var http = require("http");

var port = process.argv[2];
var app = express();
http.createServer(app).listen(port);

var todos = {};
var t1 = { message: "Maths homework due",
           type: 1,
           deadline: "12/12/2014"};
var t2 = { message: "English homework due",
           type: 3,
           deadline: "20/12/2014" };
todos[31212] = t1;
todos[23232] = t2;

app.get("/todos", function(req, res) {
    res.json(todos);
});
```

Auf dem Server: JSON senden

```
var express = require("express");
var url = require("url");
var http = require("http");

var port = process.argv[2];
var app = express();
http.createServer(app).listen(port);
```

```
var todos = {};
var t1 = { message: "Maths homework due",
           type: 1,
           deadline: "12/12/2014"};
var t2 = { message: "English homework due",
           type: 3,
           deadline: "20/12/2014" };
todos[31212] = t1;
todos[23232] = t2;
```

Wir speichern alle TODOs auf dem Server

```
app.get("/todos", function(req, res) {
    res.json(todos);
});
```

Client erfragt Kopie der TODOs beim Server

JSON-formatierte TODOs werden gesendet

Auf dem Server: todo aktualisieren

```
app.get("/addtodo", function(req, res) {
    var url_parts = url.parse(req.url, true);
    var query = url_parts.query;
    if (query["message"] != undefined) {
        var tx = {
            message: query["message"],
            type: query["type"],
            deadline: query["deadline"]
        };
        var looping = 1;
        while (looping > 0) {
            var r = getRInt(1, 1000);
            if (todos[r] == undefined) {
                todos[r] = tx;
                looping = -1;
                console.log("Added " + tx.message);
            }
        }
    }
    res.end();
});
```

Auf dem Server: todo aktualisieren

```
app.get("/addtodo", function(req, res) {
    var url_parts = url.parse(req.url, true);
    var query = url_parts.query;
    if (query["message"] != undefined) {
        var tx = {
            message: query["message"],
            type: query["type"],
            deadline: query["deadline"]
        };
        var looping = 1;
        while (looping > 0) {
            var r = getRInt(1, 1000);
            if (todos[r] == undefined) {
                todos[r] = tx;
                looping = -1;
                console.log("Added " + tx.message);
            }
        }
    }
    res.end();
});
```

Ist Query Parameter 'message' vorhanden?

Rudimentärer Ansatz um einen Index im TODO Array zu finden

Ajax: dynamische Aktualisierung des Clients

Auf dem Client: HTML

```
<doctype html>
    <html>
    <head>
        <title>Plain text TODOs</title>
        <script src="http://code.jquery.
                    com/jquery-1.11.1.js"
                    type="text/javascript"></script>
        <script src="client-app.js"
                    type="text/javascript"></script>
    </head>
    <body>
        <main>
            <section id="todo-section">
                <p>My list of TODOS:</p>
                <ul id="todo-list">
                </ul>
            </section>
        </main>
    </body>
    </html>
```

Lade JavaScript Dateien, starte mit jQuery

Definiere wo die TODOs hinzugefügt werden

mit jQuery

Auf dem Client: JavaScript

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

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

    $.getJSON("todos", addTodosToList);

}

$(document).ready(main);
```

mit jQuery

Auf dem Client: JavaScript

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

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

    $.getJSON("todos", addTodosToList);

}

$(document).ready(main);
```

Definiere was passiert, wenn ein TODO Objekt verfügbar ist (Callback)

Füge Listenelemente dynamisch in das DOM ein

Dies ist AJAX

Wenn der Aufruf an /todos beendet ist führe addTodosToList aus

Nach Laden des Dokuments main ausführen

mit jQuery

Das war's: Web App läuft!

Codebeispiele im Laborprojekt
https://github.com/zirpins/vs1lab/tree/master/Beispiele

Literatur

- **Learning Web App Development, Kapitel 5, 6**
- Empfehlung: Marijn Haverbeke, *"Eloquent JavaScrip"*, No Starch Press, 2014 (Online: http://eloquentjavascript.net)
- Ethan Brown, "Web development with Node and Express", O'Reilly, 2014
- Robert Prediger ; Ralph Winzinger, "Node.js", Hanser, 2015 (Online verfügbar im Hochschulnetz)