

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

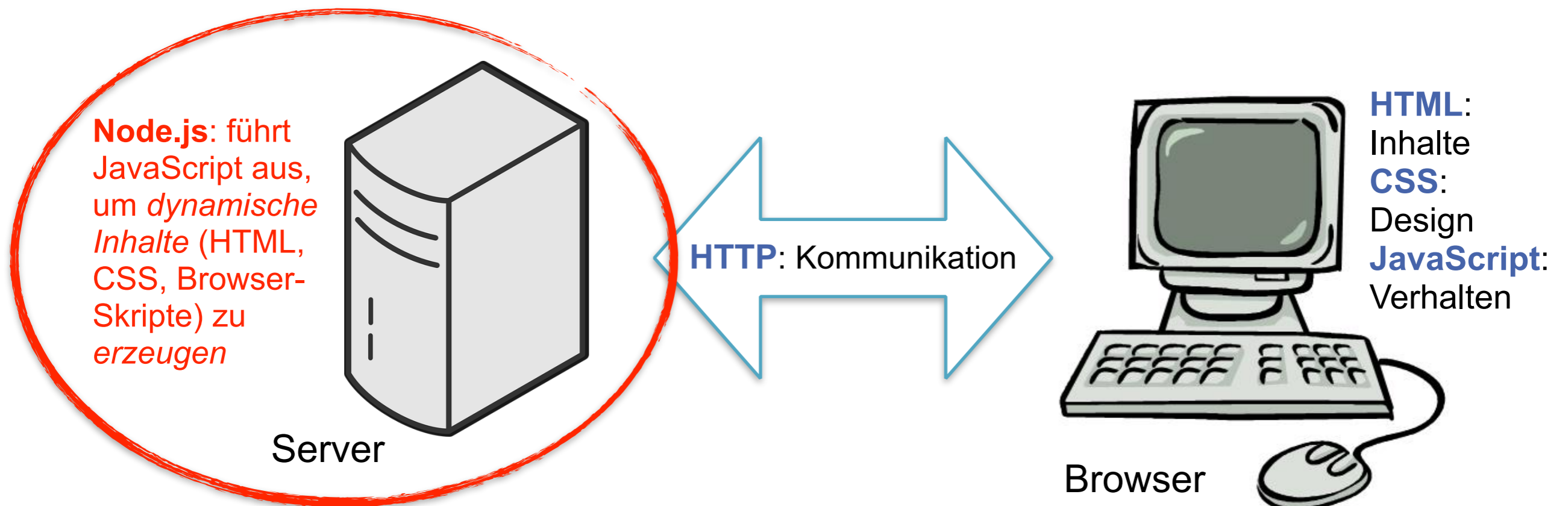
UNIVERSITY OF APPLIED SCIENCES



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
- ...Single-Page Apps mit **Ajax** und **JSON** implementieren



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."

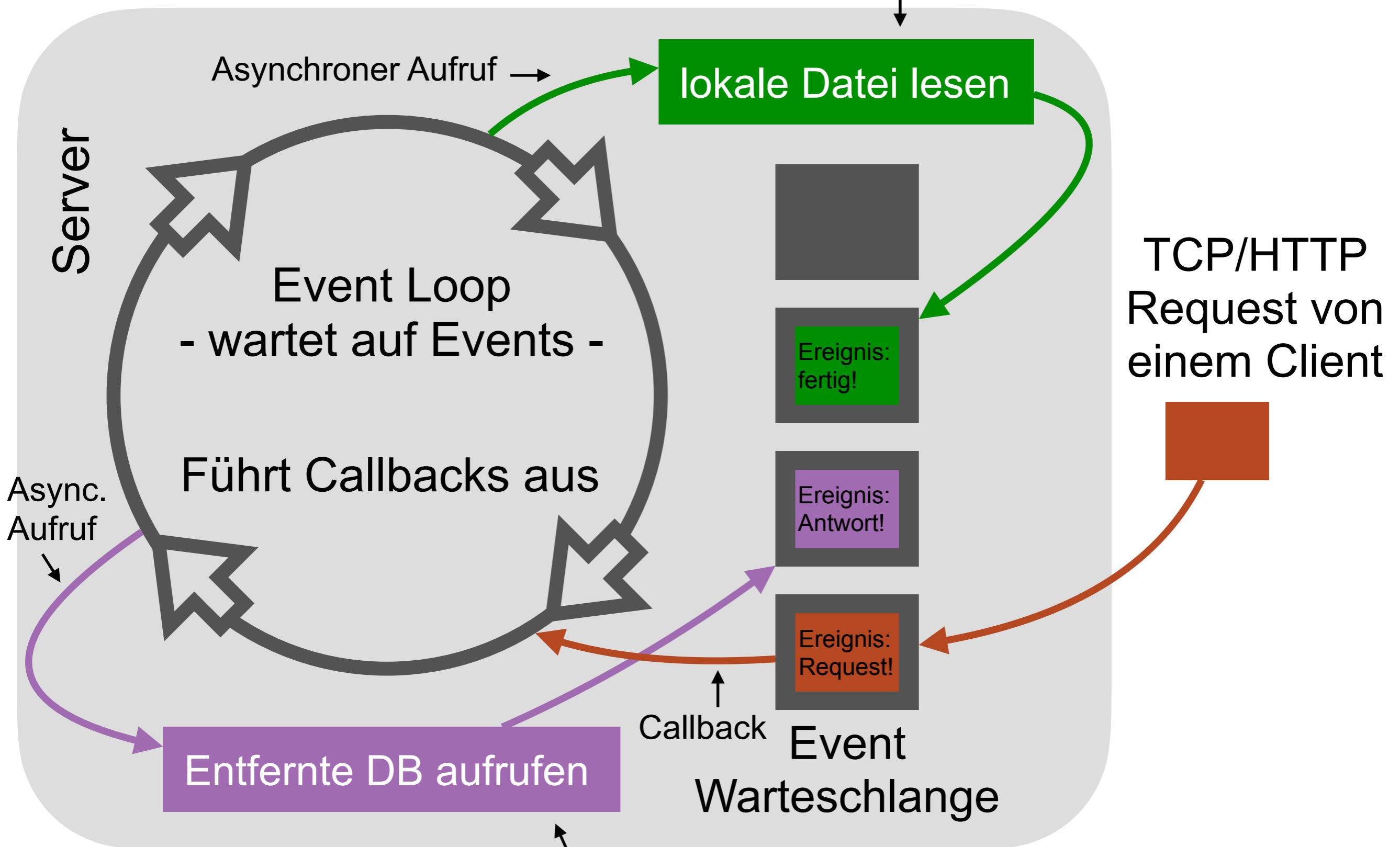
Geschichte von node.js

- Mittlerweile eine **etablierte** 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** und **Software Repository „NPM“** wurde 2011 veröffentlicht
- Native Unterstützung für **Windows** im Jahr 2011



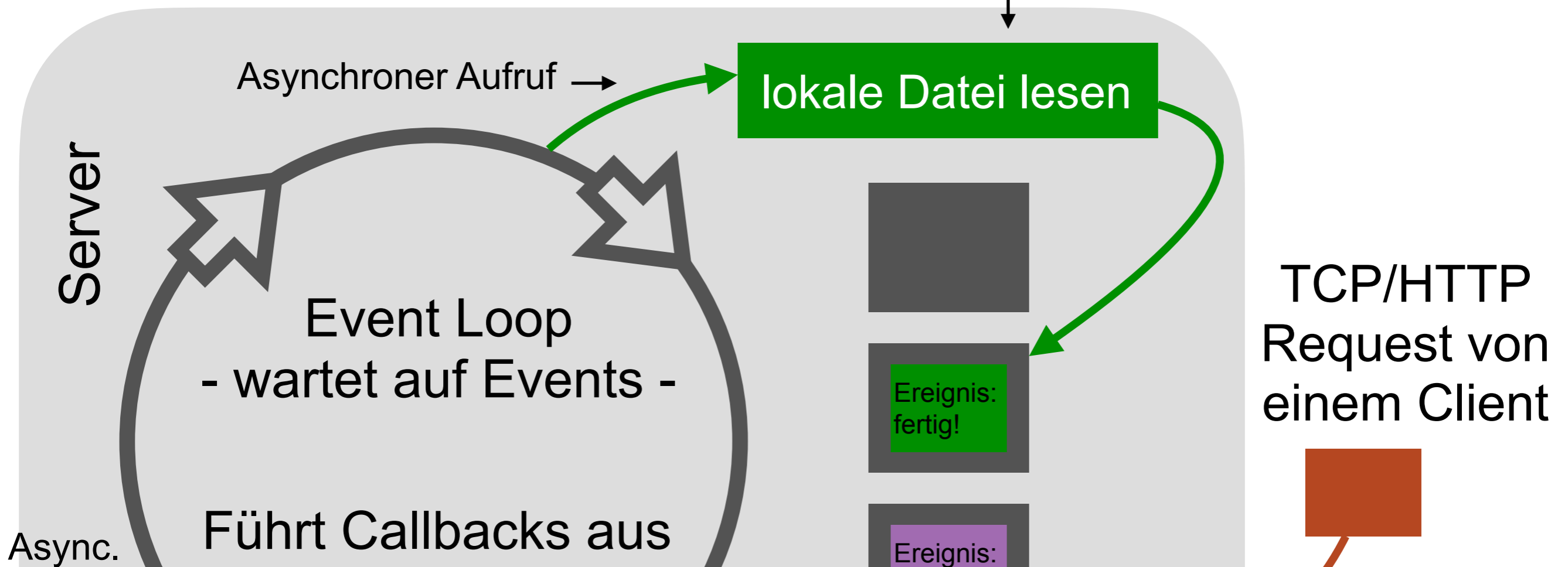
node.js ist ereignisgesteuert

Nebenläufige Ausführung



node.js ist ereignisgesteuert

Nebenläufige Ausführung



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 Client Request

Node.js ist für diese Anwendungsfälle konzipiert

Node.js: single-threaded aber **hoch parallel**

Blockierendes E / A (Datenbankbeispiel)

- (1) Client Request annehmen und auslesen
- (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) Client Request annehmen und auslesen
- (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

Eine konstante

Node.js fs
Modul

```
const fs = require("fs");  
fs.watch("todos.txt", function() {  
  console.log("File 'todos.txt' has just changed");  
});  
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:**
 - Die zu beobachtende Datei muss vorhanden sein

Netzwerkprogrammierung mit node.js

- Node.js ist speziell für **verteilte Programmierung** geeignet (nicht nur Webprogrammierung!)
- Node.js verfügt über integrierte Unterstützung für **low-level** Socket-Verbindungen (TCP-Sockets)
- TCP-Socket-Verbindungen haben **zwei Endpunkte!**
 - 1. Server **bindet** sich an einen dauerhaften nummerierten Port
 - 2. Server **verbindet** jeden Client über je einen temporären Port

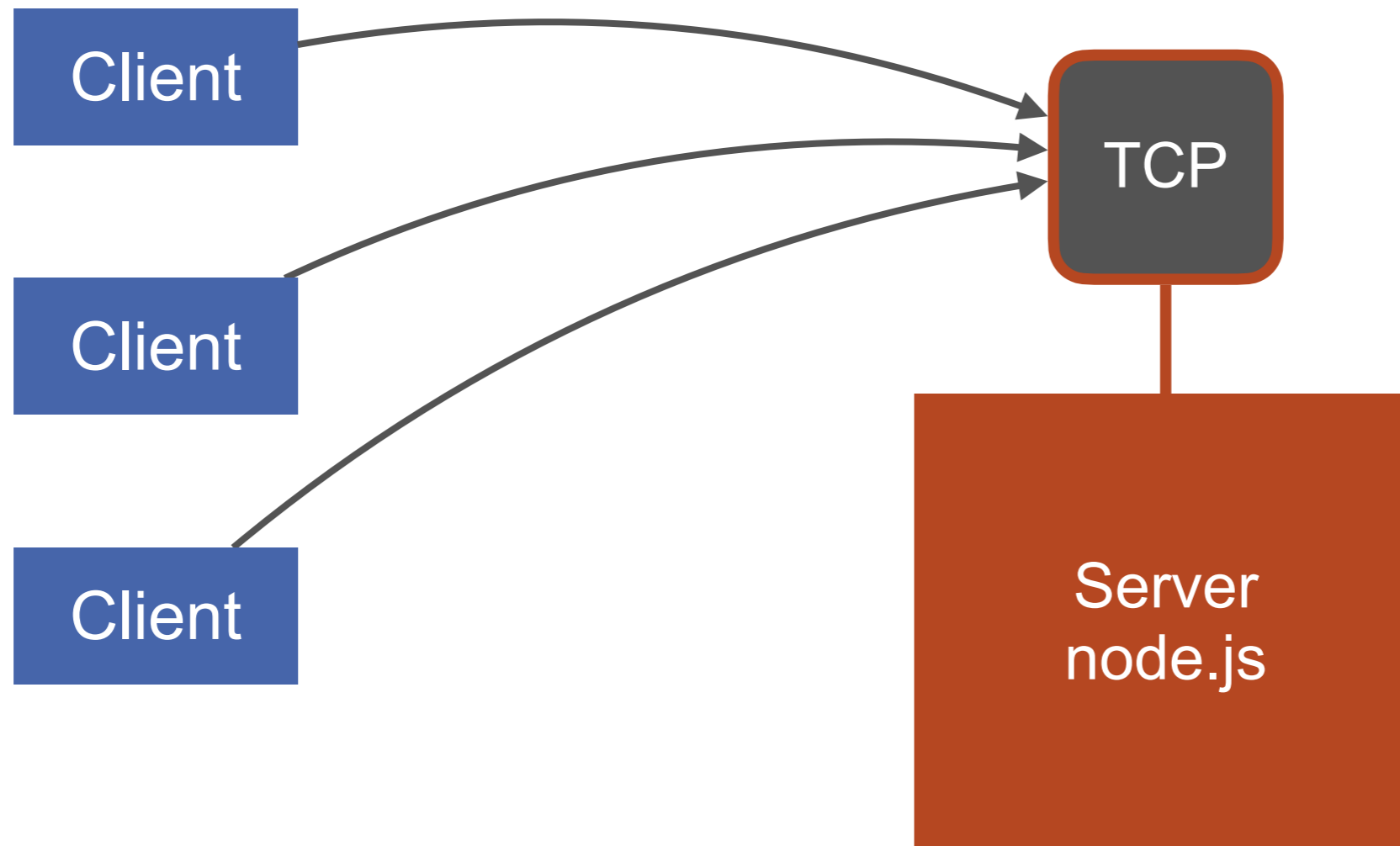
Analoges Beispiel: Telefonleitungen

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



Low-level Netzwerkprogrammierung mit node.js

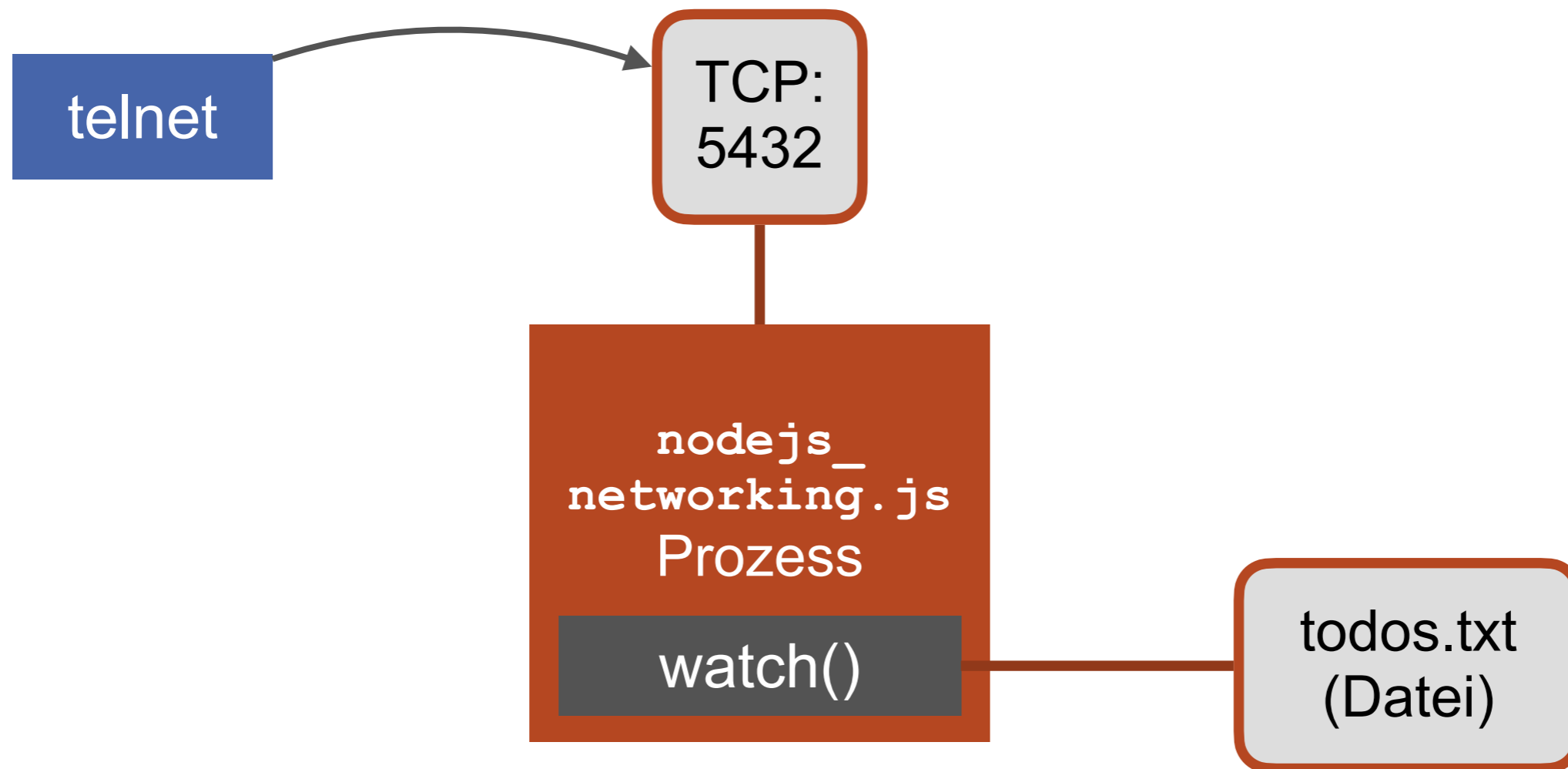
Server Objekt wird
zurückgegeben

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

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.');
```

Log auf Server

```
    connection.write("Now watching todos.txt for
                      changes...\n");
```

Log auf Client

```
    // watcher setup
    var watcher = fs.watch(filename, function() {
      connection.write("File '" + filename + "'
                       changed: " + Date.now() + "\n");
    });
```

Dateiänderungen werden zum Client gesendet

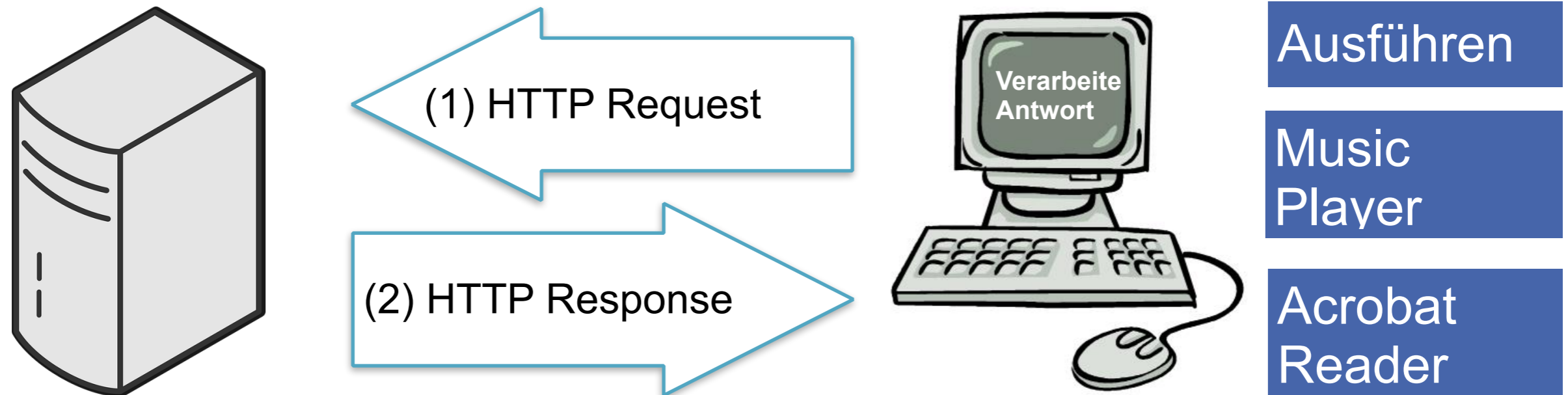
```
    // cleanup
    connection.on('close', function() {
      console.log('Subscriber disconnected.');
```

Client beendet Verbindung

```
      watcher.close();
    });
  });
server.listen(5432, function() {
  console.log('Listening for subscribers...');
});
```

Kurzer **Rückblick** auf die erste Vorlesung

Web Server und Clients



- 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.

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
```

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)

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

node.js http Modul

basic-server.js

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

```
var server;
```

Web Server erstellen

Callback: was bei einem Request getan werden soll

```
server = http.createServer(function(req, res) {
```

```
  res.writeHead(200, {  
    "Content-Type": "text/plain"
```

Erstelle HTTP Response & sende zurück

```
  });
```

```
  res.end("Hello World!");
```

```
  console.log("HTTP response sent");
```

```
});
```

Web Server starten

```
server.listen(3000);
```

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

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;
```

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

HTTP Request Objekt

```
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");  
});
```

HTTP Response Objekt

Setze HTTP-Status-Code und -Header als Objekt

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

Verschiedene Ports möglich

'Hello World', node.js Style

basic-server3.js

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

Response Funktion

```
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");  
}
```

Response Funktion als Parameter

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

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

Kommandozeilenparameter

URLs für das "Routing"

```
basic-server4.js
var http = require("http");
var server;

var simpleHTTPResponder = function(req, res) {
  var url_parts = new URL(req.url, `http://${req.headers.host}`);
  if (url_parts.pathname == "/greetme") {
    res.writeHead(200, {"Content-Type": "text/plain"});
    var query = url_parts.searchParams;
    if (query.has("name") {
      res.end("Greetings " + query.get("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);
```

Template String

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 server;  
  
var simpleHTTPResponder = function(req, res) {  
  var url_parts = new URL(req.url, `http://${req.headers.host}`);  
  if (url_parts.pathname == "/greetme") {
```

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

```
  }  
  } else {  
    res.writeHead(404, {"Content-Type": "text/plain"});  
    res.end("Only /greetme is implemented.");  
  }  
}
```

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 http = require("http");
var app;

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

app.get("/greetme", function(req, res) {
  var url_parts = new URL(req.url, `http://${req.headers.host}`);
  var query = url_parts.searchParams;

  var name = query.has("name") ? query.get("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 http = require("http");  
var app;
```

app Objekt kapselt
Express Funktionen

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

URL "Route" Konfiguration

```
app.get("/greetme", function(req, res) {  
  var url_parts = new URL(req.url, `http://${req.headers.host}`);  
  var query = url_parts.searchParams;  
  
  var name = query.has("name") ? query.get("name") : "Anonymous";  
  res.send("Greetings " + name);  
});
```

Noch eine Route

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

Express erzeugt die
HTTP Header

'Hello World', Express Style

```
var express = require("express");
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 url_parts = new URL(req.url, `http://${req.headers.host}`);
  var query = url_parts.searchParams;

  var name = query.has("name") ? query.get("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.

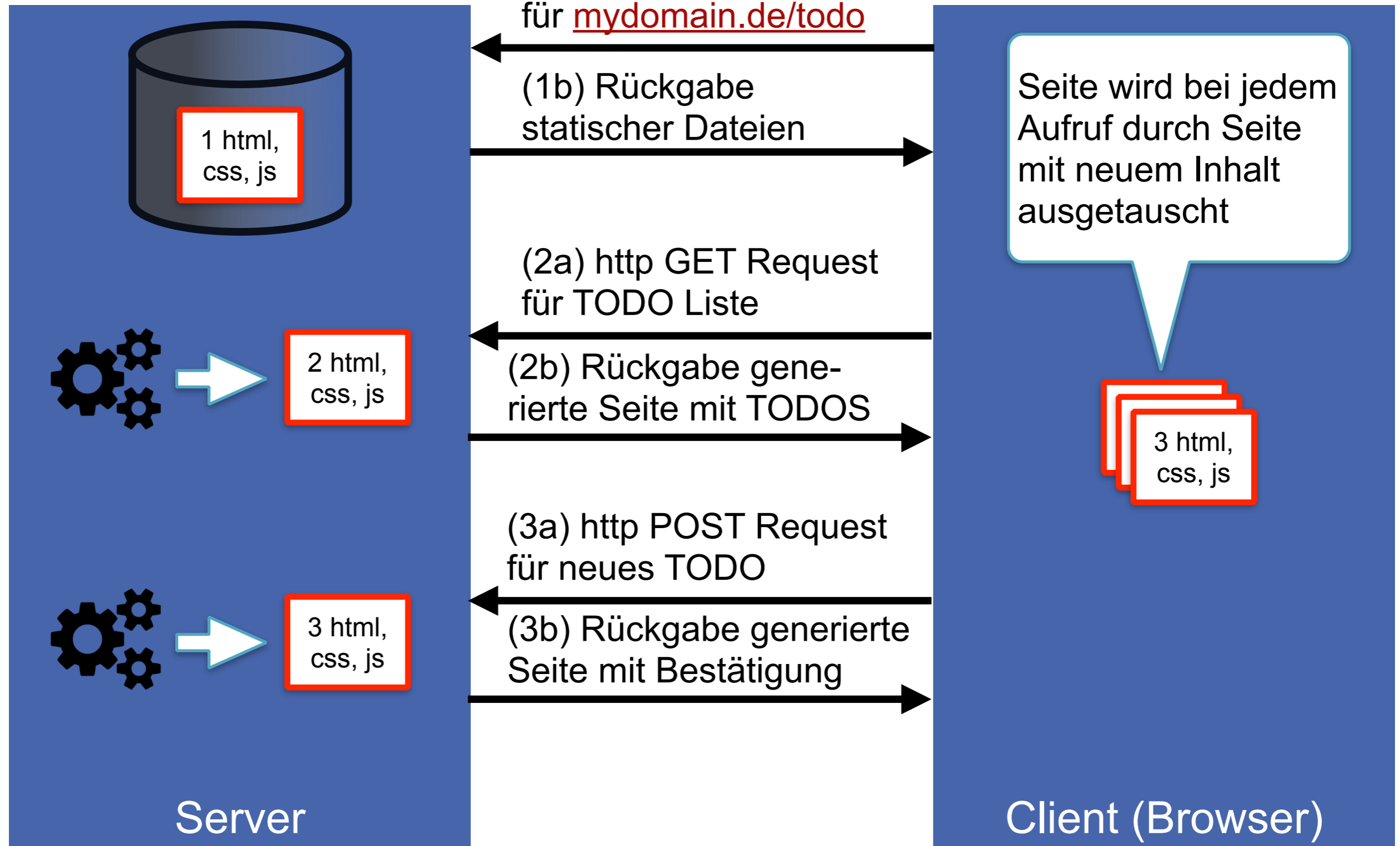
Wie baut man eine **Web Anwendung**?

Entwicklungsstrategie

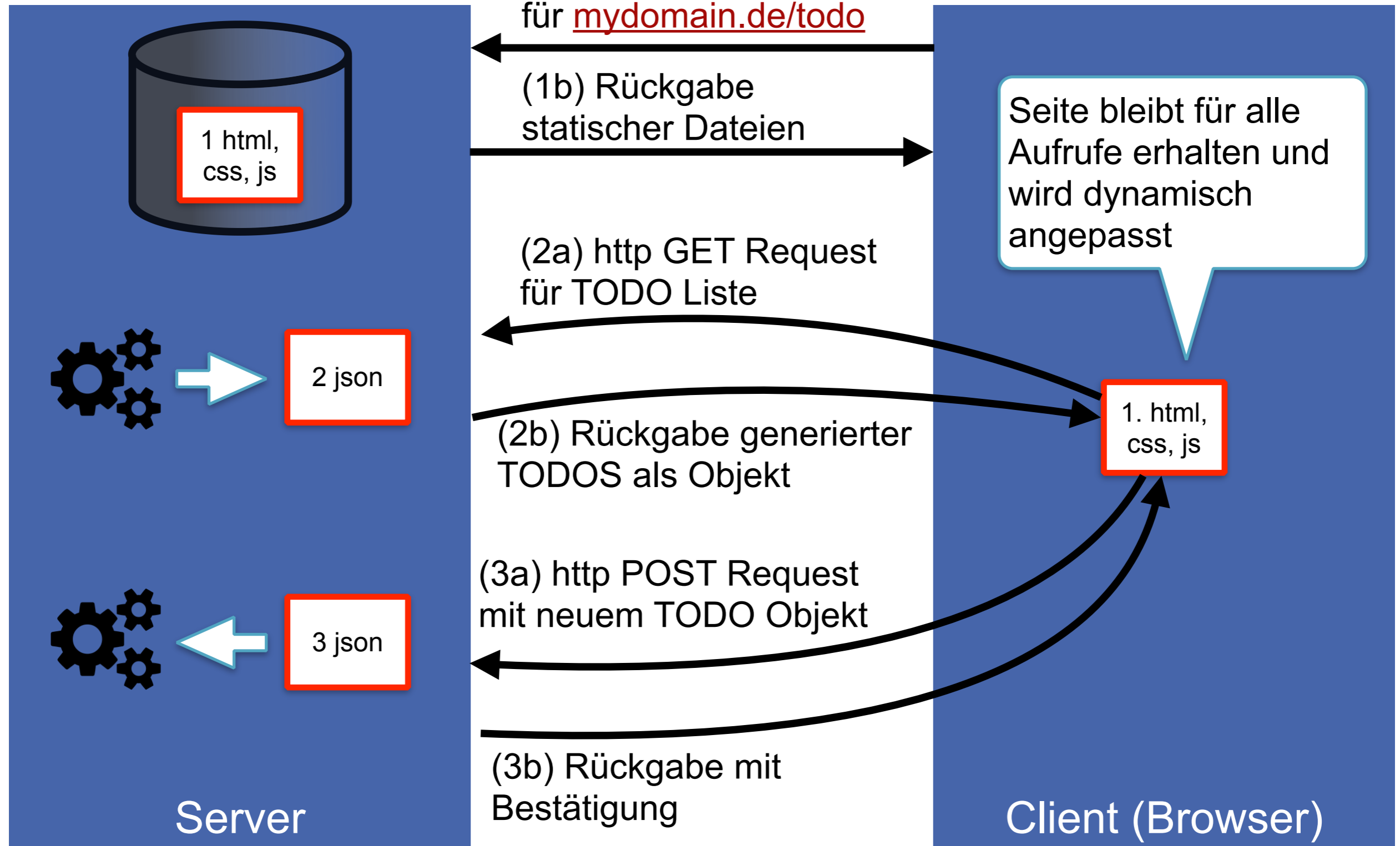
- Entwickele den **clientseitigen Code** (HTML, CSS, JavaScript)
- Platziere alle Dateien in ein Verzeichnis (z.B. `/client`) **auf dem Server**
- Definiere den **node.js Servercode** in einer `*.js` Datei mit Express
- Setze den **statischen Dateipfad** auf das Verzeichnis aus Schritt 2
- Füge Interaktionen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

```
{
  "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 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 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 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

Client erfragt Kopie der
TODOs beim Server

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

JSON-formatierte TODOs
werden gesendet

Auf dem Server: todo aktualisieren

```
app.get("/addtodo", function(req, res) {
  var url_parts = new URL(req.url, `http://${req.headers.host}`);
  var query = url_parts.searchParams;
  if (query.has("message")) {
    var tx = {
      message: query.get("message"),
      type: query.get("type"),
      deadline: query.get("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 = new URL(req.url, `http://${req.headers.host}`);  
  var query = url_parts.searchParams;  
  if (query.has("message")) {  
    var tx = {  
      message: query.get("message"),  
      type: query.get("type"),  
      deadline: query.get("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);  
  
}
```

```
$(main);
```

mit jQuery

Auf dem Client: JavaScript

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

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

```
  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);  
    }  
  };
```

Füge
Listenelemente
dynamisch in
das DOM ein

```
$.getJSON("todos", addTodosToList);
```

Dies ist AJAX

```
}
```

Wenn der Aufruf an /todos beendet
ist führe addTodosToList aus

```
$(main);
```

Nach Laden des Dokuments
Funktion "main" ausführen

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 JavaScript*", 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)

