

JANUAR 2021

# ECHTZEITANWENDUNGEN MIT HTML5, WEBSOCKETS UND WEB WORKER

Grundlagen zur Realisierung von Webapplikationen

# ECHTZEITANWENDUNGEN MIT HTML5 WEBSOCKETS UND WEB WORKER

1. UND 2. TAG

## HTML5 Kurzeinführung

- ▶ Web-Client Technologien HTML, CSS, JS
- ▶ AJAX und Single Page Apps, fetch()

## Die WebSocket API

- ▶ Methoden
- ▶ Attribute
- ▶ Anwendung

## WebSocket Proxys

- ▶ unter node.js
- ▶ andere Möglichkeiten

## Realisierung eines Webservices

- ▶ unter node.js
- ▶ Datenbankzugriff
- ▶ Formate für Serverresponses

## Synchronisierung mehrerer Browserclients

- ▶ WebSocket-Server als Zentrale für mehrere Browser
- ▶ Programmieren eines **Multichats**
- ▶ Programmieren eines Multiuserpong - Spieles
- ▶ weitere Beispiele

## Die Web Worker API

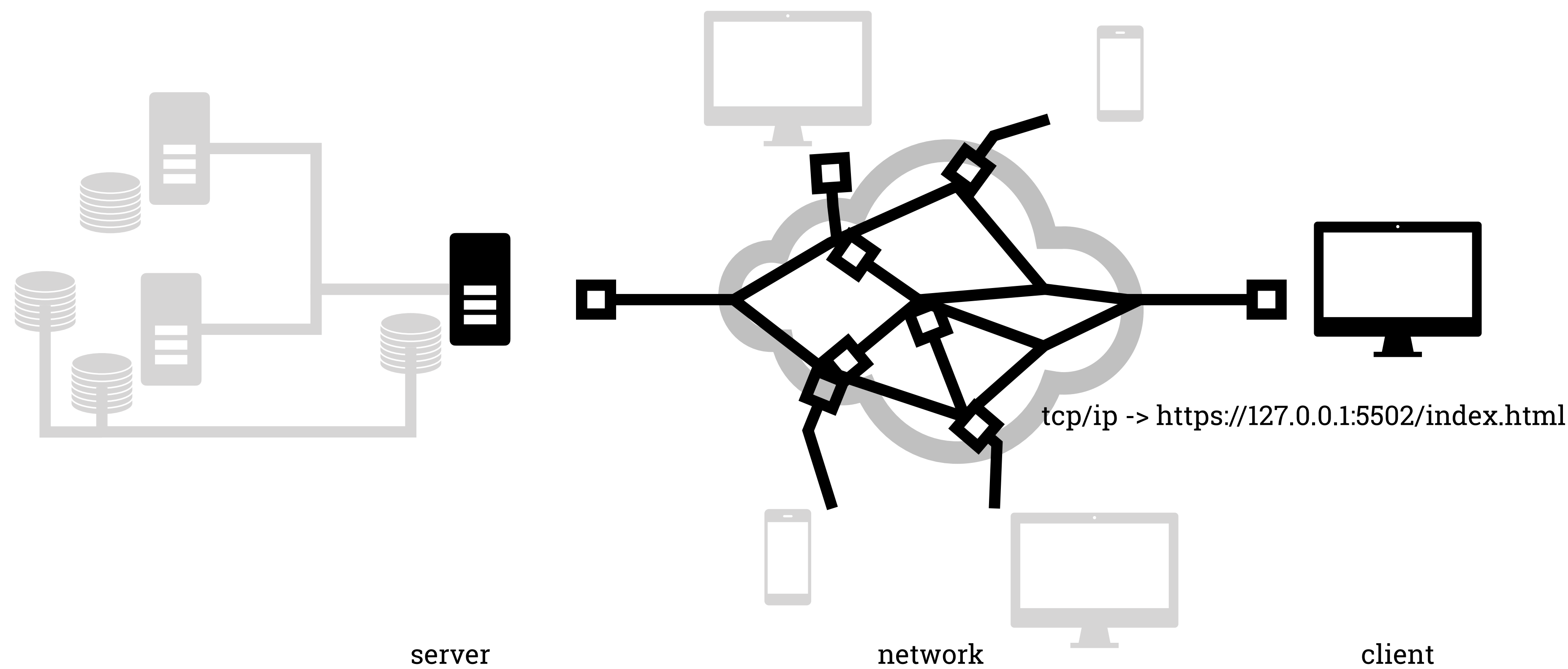
- ▶ Methoden und Attribute
- ▶ Service Worker, Inline Worker, Shared Worker

## Threads

- ▶ Starten und Beenden von Threads
- ▶ Verarbeiten der Ergebnisse
- ▶ Beispiele für Multithreadanwendungen

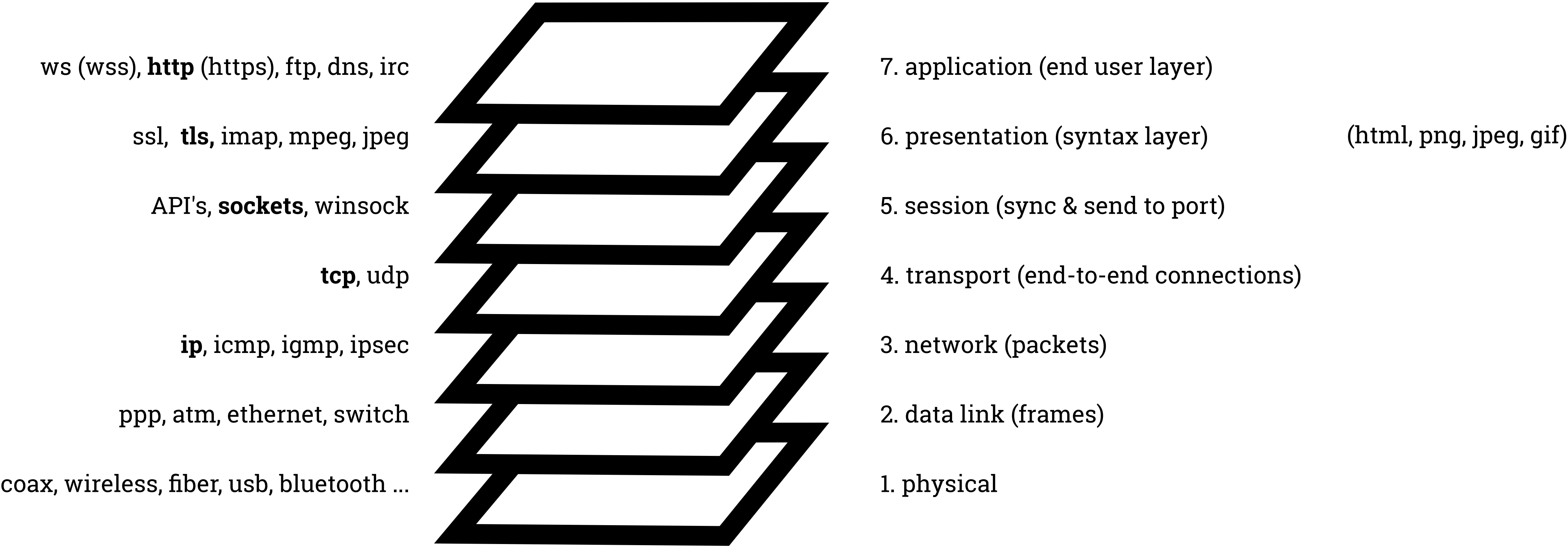
# WEBSERVER

---



OSI - OPEN SYSTEMS INTERCONNECTION

seven tiers of OSI



# OSI - OPEN SYSTEMS INTERCONNECTION

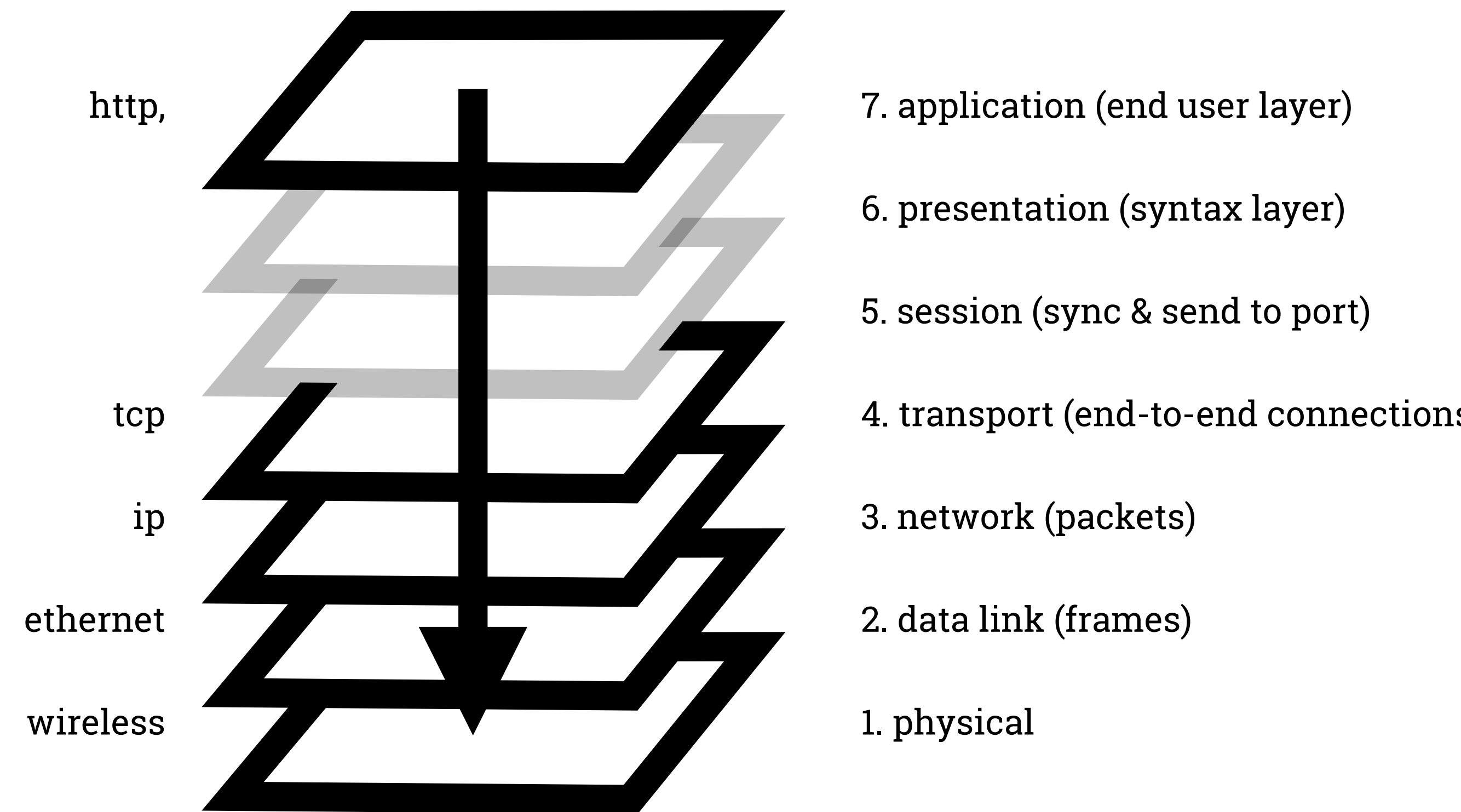
---

If you use your web browser to navigate to **http://www.gfu.net**, this communication uses the following protocols from each layer, starting at layer 7:

**HTTP → TCP → IP → Ethernet.**

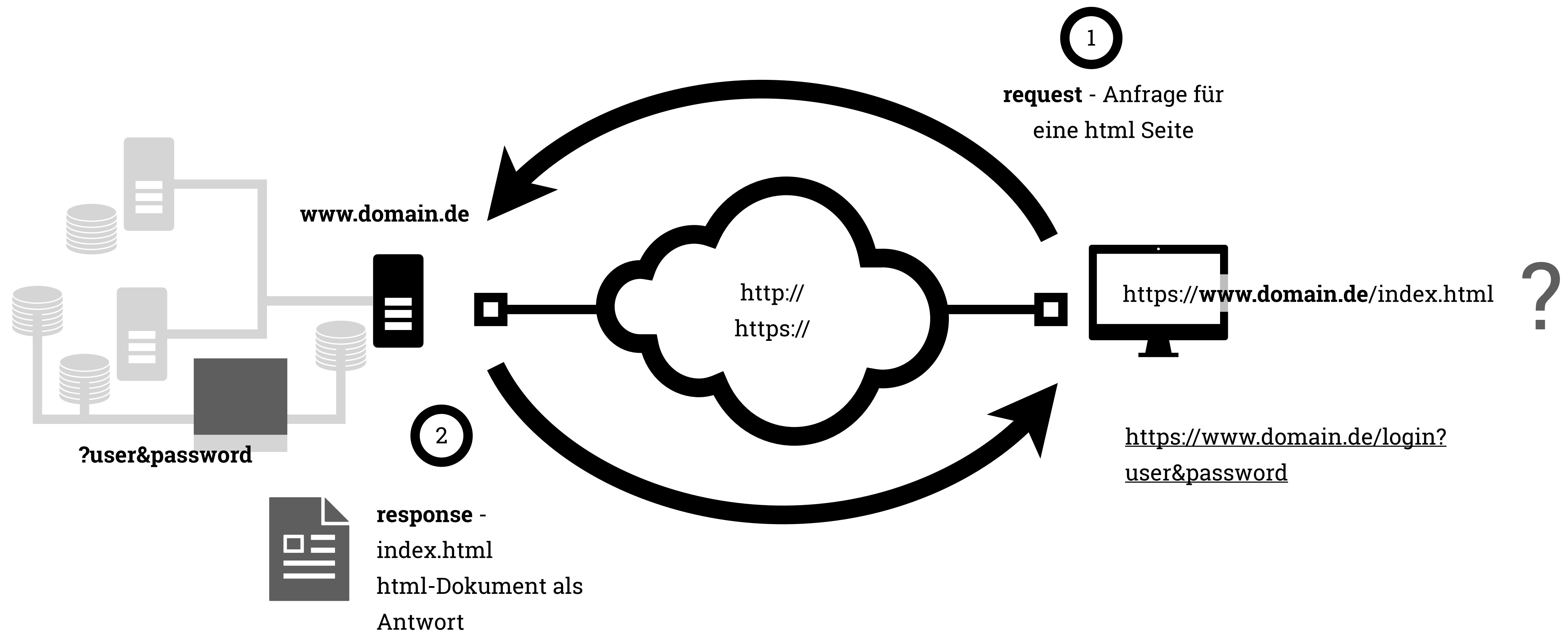
On the other hand, entering **https://www.gfu.net** would use

**HTTP → SSL → TCP → IP → Ethernet.**



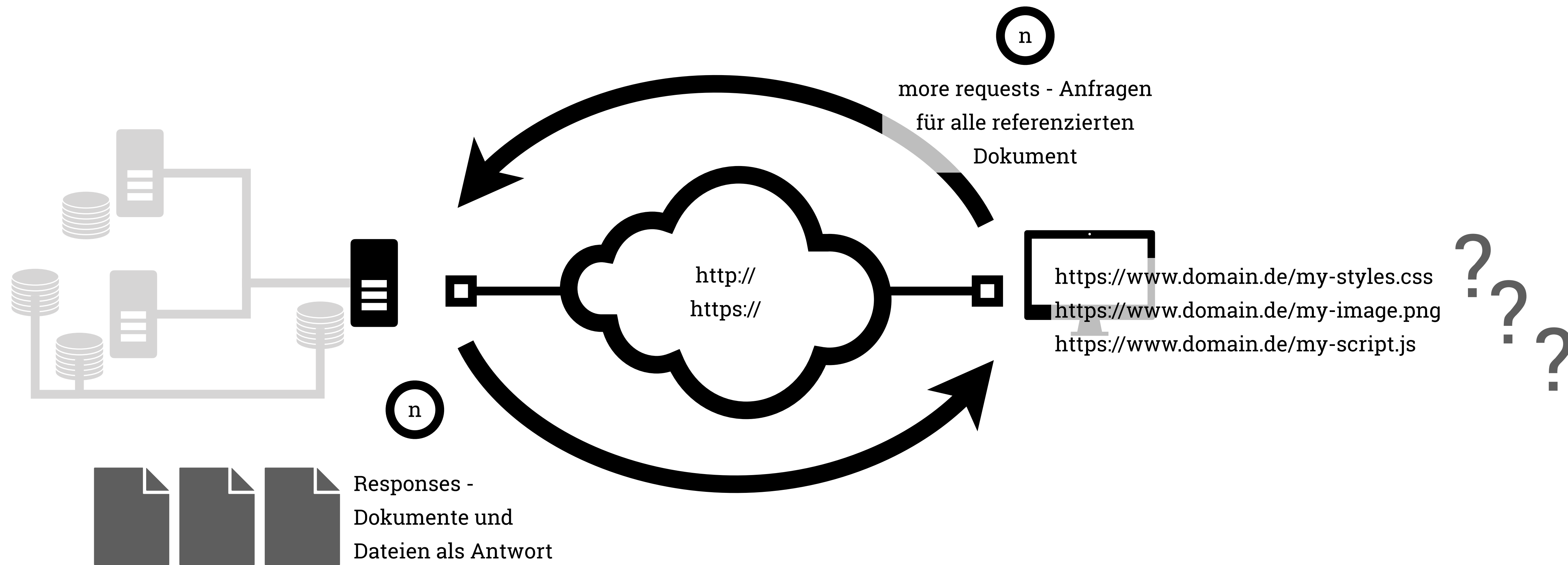
# REQUEST - RESPONSE

---

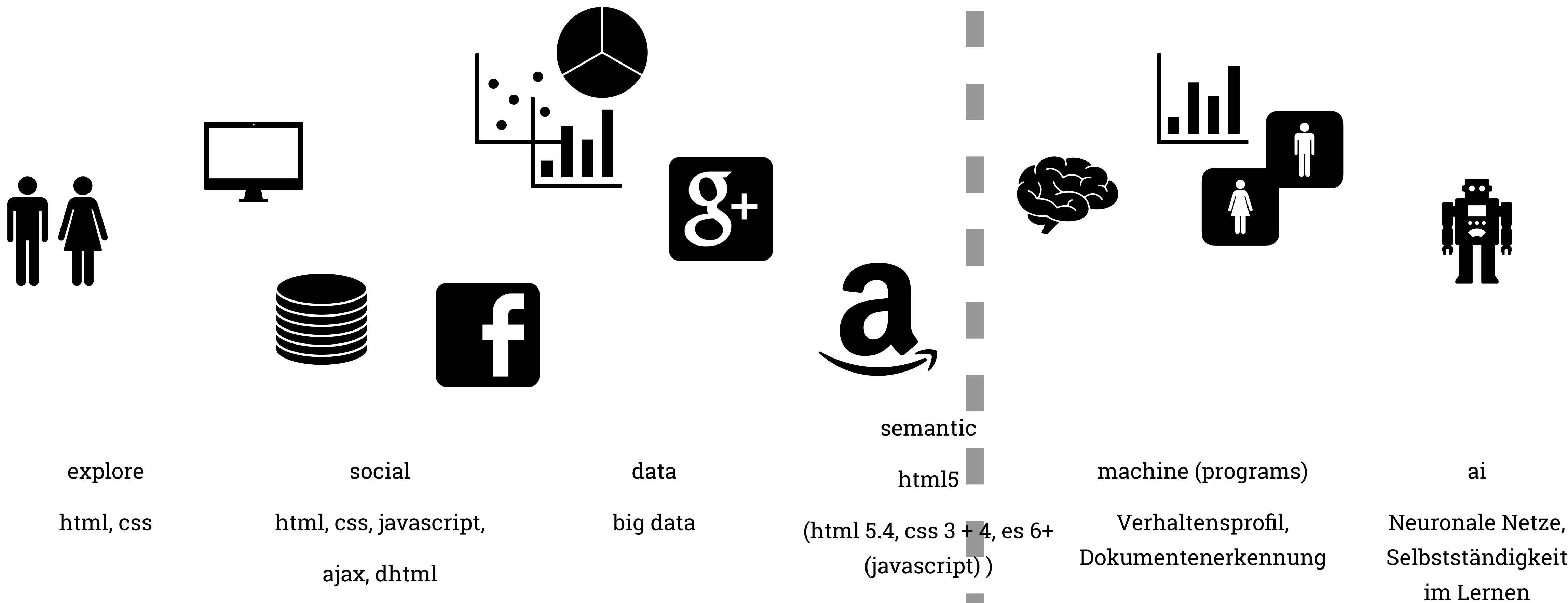


# REQUEST - RESPONSE

---



# WEB 1.0, WEB 2.0, WEB 3.0, INDUSTRIE 4.0 (DIGITALISIERUNG, AUTONOMISIERUNG)





# WEB-CLIENT TECHNOLOGIEN

---



- ▶ HTML - Hypertext Markup Language
- ▶ CSS - Cascading Style Sheets
- ▶ Javascript
- ▶ Ajax, Single Page Application (SPA)

# HTML - HYPERTEXT MARKUP LANGUAGE

---

Die Hypertext Markup Language (HTML, englisch für Hypertext-Auszeichnungssprache) ist eine textbasierte Auszeichnungssprache zur Strukturierung elektronischer Dokumente wie Texte mit Hyperlinks, Bildern und anderen Inhalten.

HTML-Dokumente sind die Grundlage des World Wide Web und werden von Webbrowsern dargestellt.

# HTML5

---

Das Paket HTML5 ist eine seit 2010/2014 existierende Bündelung von HTML, CSS und Javascript.

Es bildet die Grundlage für das Umsetzen von beliebigen Applikationsinterfaces.

Für den Browser erscheinen mehr und mehr Bibliotheksobjekte, die sämtliche Funktionalitäten einer Anwendungsoberfläche bereitstellen.

... websockets, local storage, audio/video, 2D/3D Grafik, bluetooth, usb devices ... sind nur einige der neuen Funktionalitäten.

# CONTENT: HTML - HYPERTEXT MARKUP LANGUAGE

---

```
<html>
<head>
  <title>The Document Title</title>
</head>
<body onload="my-field.focus()">
  <h1>Headline Ipsum Dolor Sit Amet</h1>
  <p>A paragraph consectetur ad piscit ...</p>
  
  <a href="other-file.html">to the other file</a>
  <form><input tabindex="1" ...><button type="submit">log in</button></form>
</body>
</html>
```

# HTML - HYPERTEXT MARKUP LANGUAGE

---

HTML hat die Aufgabe, eine Seite oder einen Inhalt semantisch zu strukturieren.

Eine Webseite soll kein CSS oder Javascript direkt enthalten, sondern sich den Inhalt samt seiner inhaltlichen Struktur begrenzen.

Dies macht Inhalte maschinenlesbar und durch Programme und assistive Systeme verarbeitbar.

# CSS - CASCADING STYLE SHEETS

---

Cascading Style Sheets (für gestufte Gestaltungsbögen; kurz: CSS) ist eine Stylesheet-Sprache für elektronische Dokumente und zusammen mit HTML und JavaScript eine der Kernsprachen des World Wide Webs.

# CSS - CASCADING STYLE SHEETS

---

CSS schreibt Regeln für die Visualisierung und das dynamisch-visuelle Verhalten von HTML Elementen fest.

Jede sichtbare Eigenschaft besitzt eine CSS Regeln und kann verändert werden

Der Browser selbst benutzt ein rein auf Semantik abzielendes CSS.

Eine Webapplikation kann durch CSS beliebig und in jedem Aspekt umgestaltet werden, ohne das der Inhalt seine Struktur verliert.

U. a. dies macht HTML/CSS zu einem universellen Oberflächenwerkzeug für Applikationen.

# CSS - CASCADING STYLE SHEETS

---

```
/* my-styles.css */  
html * { box-sizing: border-box;}  
body { background-color: white; color: black;}  
.responsive-image { height: auto; max-width: 100%}  
a[href]:hover { color: red; }
```



# HTML + CSS

---

```
<html>

<head>

  <title>The Document Title</title>

  <link type="stylesheet" href="my-styles.css"

</head>

<body>

  <h1>Headline Ipsum Dolor Sit Amet</h1>

  <p>A paragraph consectetur ad psicit ...</p>

  <a href="other-file.html">to the other file</a>

</body>

</html>
```

# JAVASCRIPT

---

JavaScript (kurz JS) ist eine Skriptsprache, die für dynamisches HTML in Webbrowsern entwickelt wurde, um Interaktionen des Benutzers auszuwerten, Inhalte zu verändern, nachzuladen oder zu generieren und so die Möglichkeiten von HTML und CSS zu erweitern.

Heute findet JavaScript auch außerhalb von Browsern Anwendung, so etwa auf Servern und in Microcontrollern.

Javascript ist im Grunde eine im Browser implementierte Bibliothek auf Basis von ECMA Script (ES).

Nodejs ist eine ES-Variante, die als Serversprache implementiert wurde.

Landläufig wird beides Javascript genannt.

# JAVASCRIPT

---

Javascript ist eine im Kern funktionale Sprache, die auf Objekten basiert und Objekte handhabt.

Javascript besitzt eine eigene Objektnotation namens JSON, die sich auch außerhalb von Javascript als Datenformat durchgesetzt hat.

Javascript beherrscht auch Klassen und Instanzenbildung, besitzt aber eine nur schwache Typisierung und keine Sichtbarkeiten.

Javascript wird im Browser kompiliert.

# JAVASCRIPT

---

```
/* my-script.js */  
  
let anchor = document.querySelector('a[href]');  
anchor.addEventListener('click', onMyAnchorClick);  
  
function onMyAnchorClick(event){  
    event.preventDefault();  
    loadData();  
}  
  
function loadData () { ... }
```

# HTML + CSS + JAVASCRIPT

---

```
<html>

<head>

  <title>The Document Title</title>

  <link type="stylesheet" href="my-styles.css"

</head>

<body>

  <h1>Headline Ipsum Dolor Sit Amet</h1>

  <p>A paragraph consectetur ad psicit ...</p>

  <a href="other-file.html">to the other file</a>

  <script src="any-library-or-framework.js">

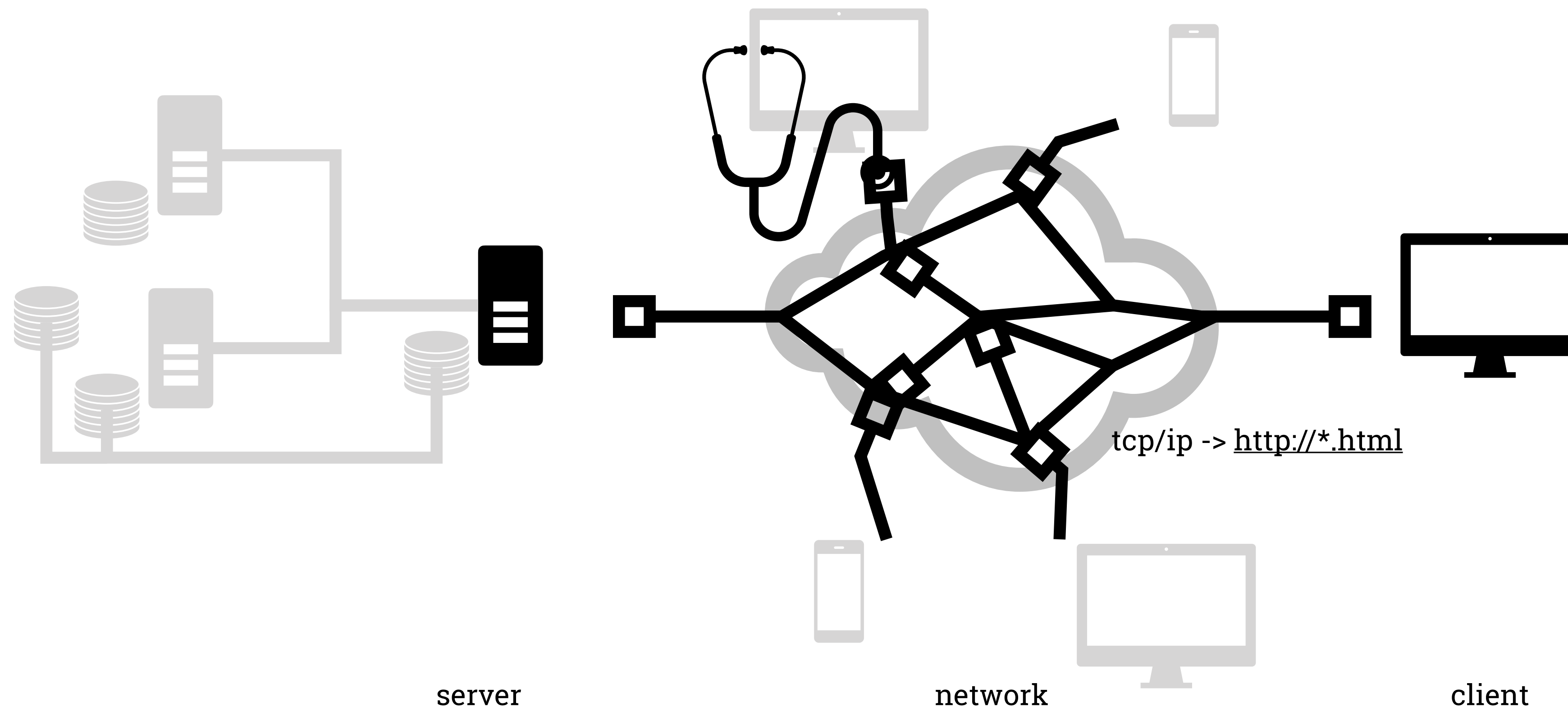
  <script src="my-script.js">

</body>

</html>
```

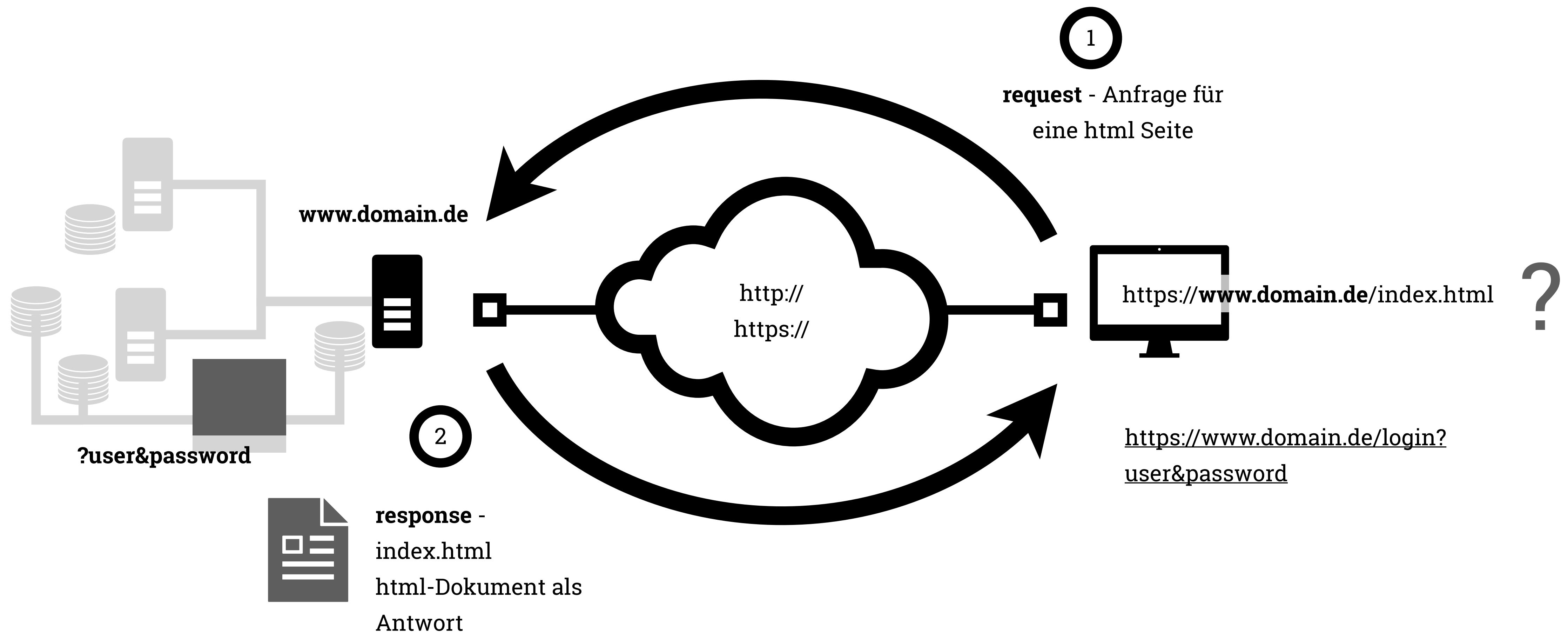
# WEBSERVER

---



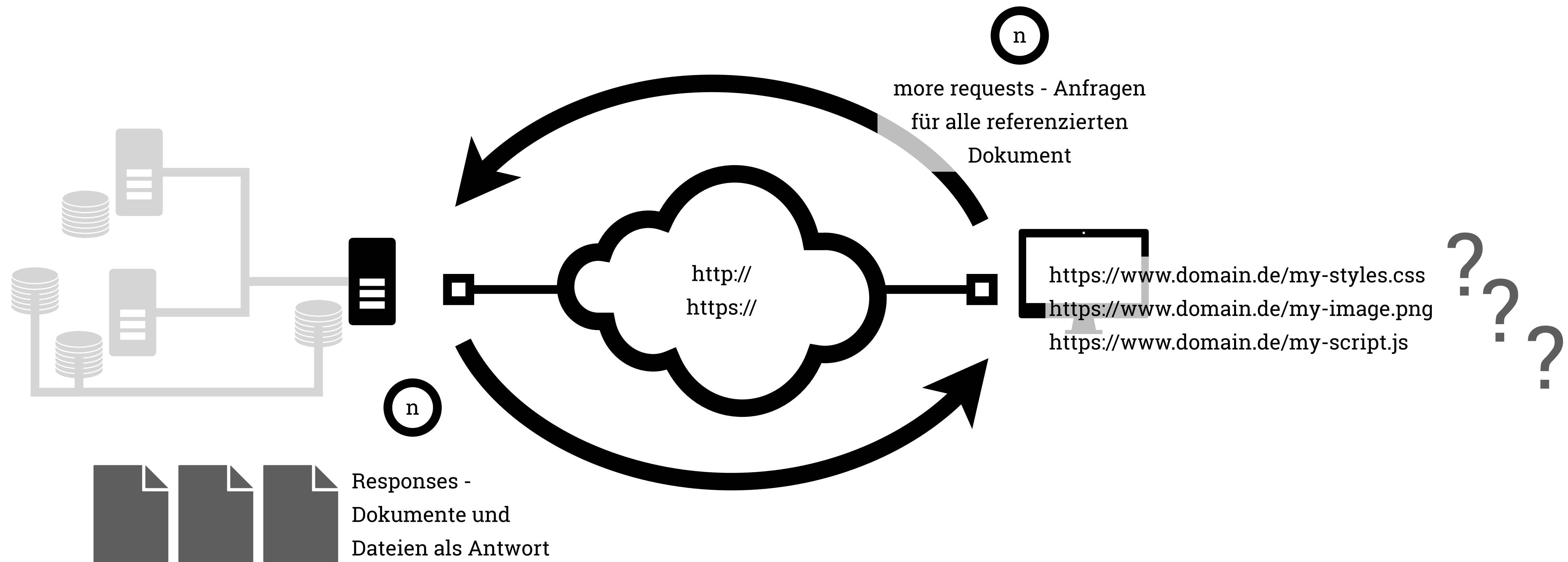
# REQUEST - RESPONSE

---



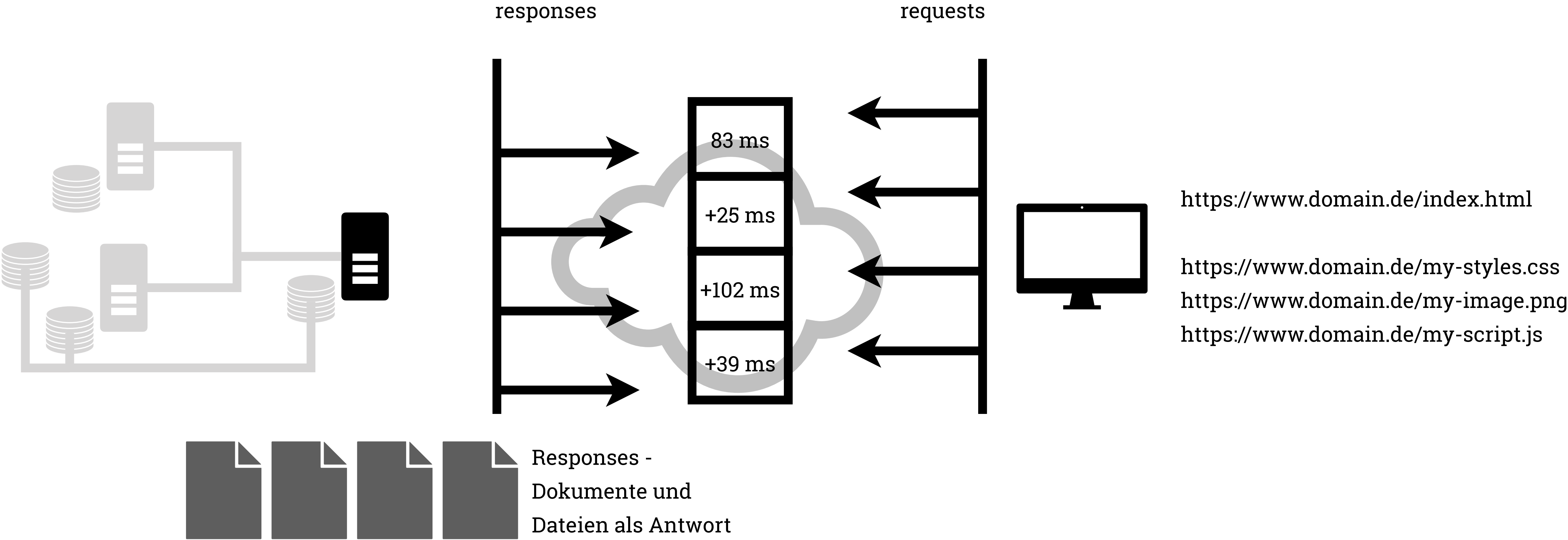
# REQUEST - RESPONSE

---





# ASYNCHRONOUS REQUEST - RESPONSE



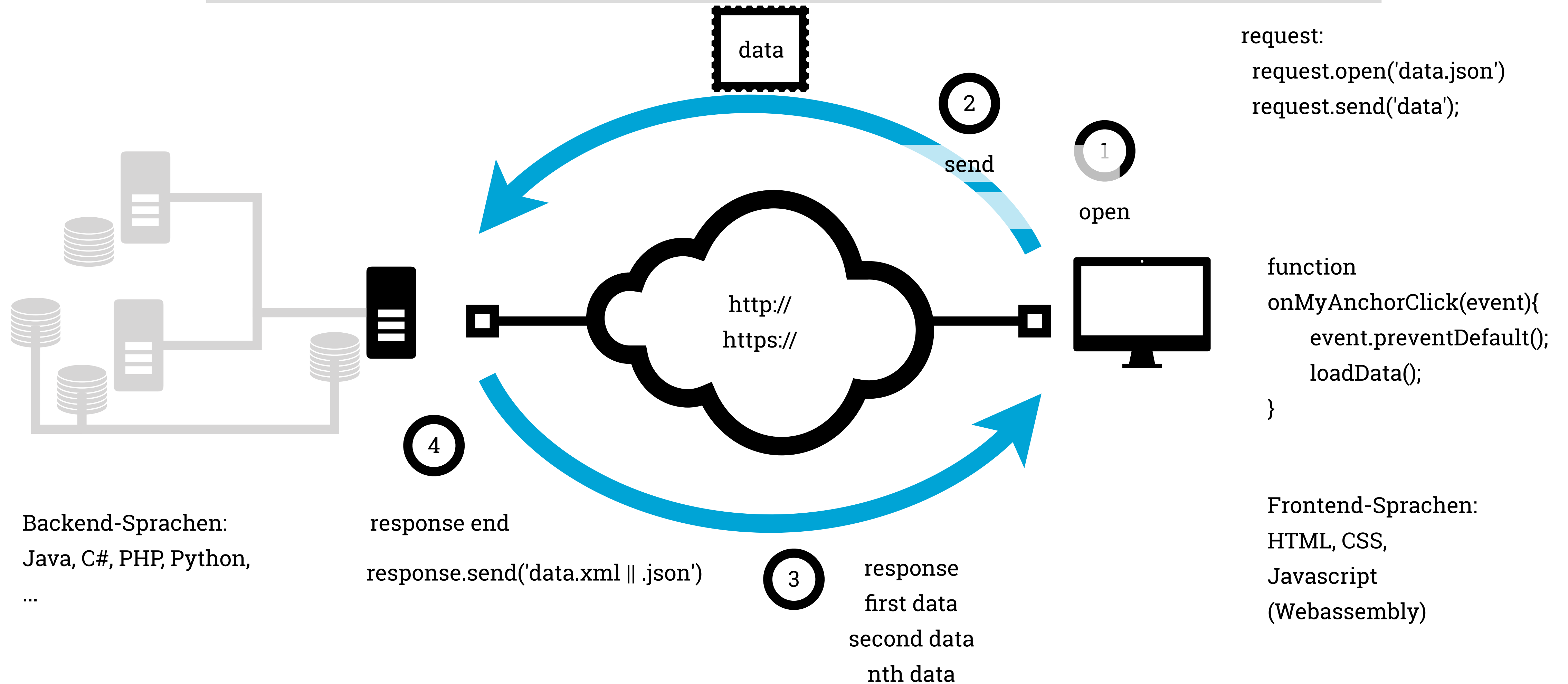
# AJAX - ASYNCHRONOUS JAVASCRIPT AND XML

---

AJAX (Asynchronous JavaScript and XML) bezeichnet ein Konzept der asynchronen **Datenübertragung** per Javascript zwischen einem Browser und dem Server.

Es ermöglicht es, HTTP-Anfragen durchzuführen, während eine HTML-Seite angezeigt wird, und die Seite zu verändern, ohne sie komplett neu zu laden.

# ASYNCHRONOUS REQUEST - RESPONSE PHASEN



# ASYNCHRONER REQUEST (XHR)

old-fashioned style

---

```
xhr = new XMLHttpRequest();
xhr.addEventListener('readystatechange', onReadyStateChange);
xhr.open("GET", 'data.xml'); // Request
xhr.send();

function onReadyStateChange() {
    switch (xhr.readyState) {
        case 0: console.log('there is no request'); break;
        case 1: console.log('request opened'); break;
        case 2: console.log('request sent');    break;
        case 3: console.log('response first part ...'); break;
        case 4: console.log('response more parts and finished!'); break;
    }
}

document.createElement('p') -> p.addChild('data')
```

# ASYNCHRONER REQUEST MIT EINEM PROMISE

ES6+

---

```
fetch( 'http://domain.de/data.json' )  
  .then(function (response) {  
    if (response.ok)  
      return response.json();  
    else  
      throw new Error('Daten konnten nicht geladen werden');  
  })  
  .then(function (json) { // Code zum Verarbeiten der  
Daten })  
  .catch(function (err) { // Hier Fehlerbehandlung });
```

# SPA - SINGLE PAGE APPLICATION

---

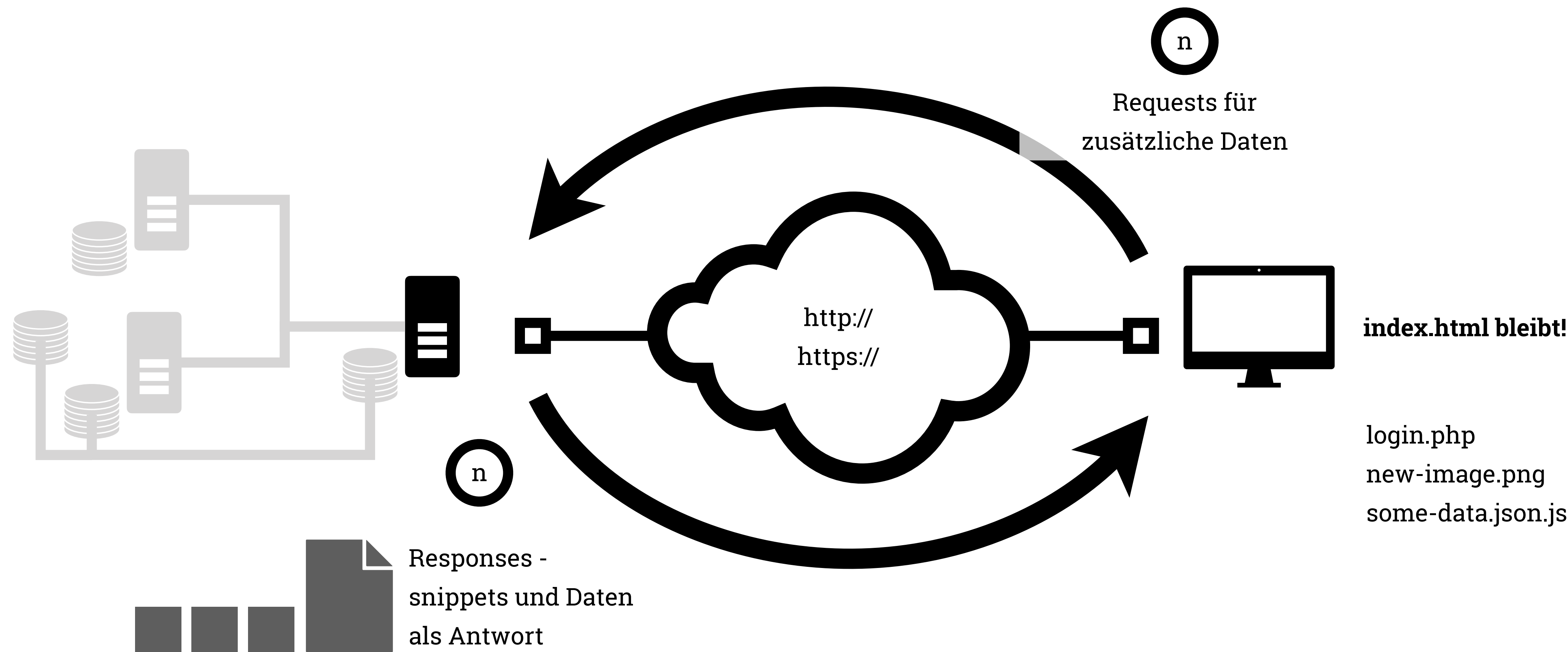
Sie besteht aus **einem einzigen HTML-Dokument**. Weitere Inhalte werden dynamisch nachgeladen.

Dies ist die Grundlage für sogenannte Rich-Clients bzw. Fat-Clients.

Eine verstärkte clientseitige Ausführung der Webanwendung ermöglicht eine Reduzierung der Serverlast sowie die Umsetzung von selbstständigen Webclients, die beispielsweise eine Offline-Unterstützung anbieten.

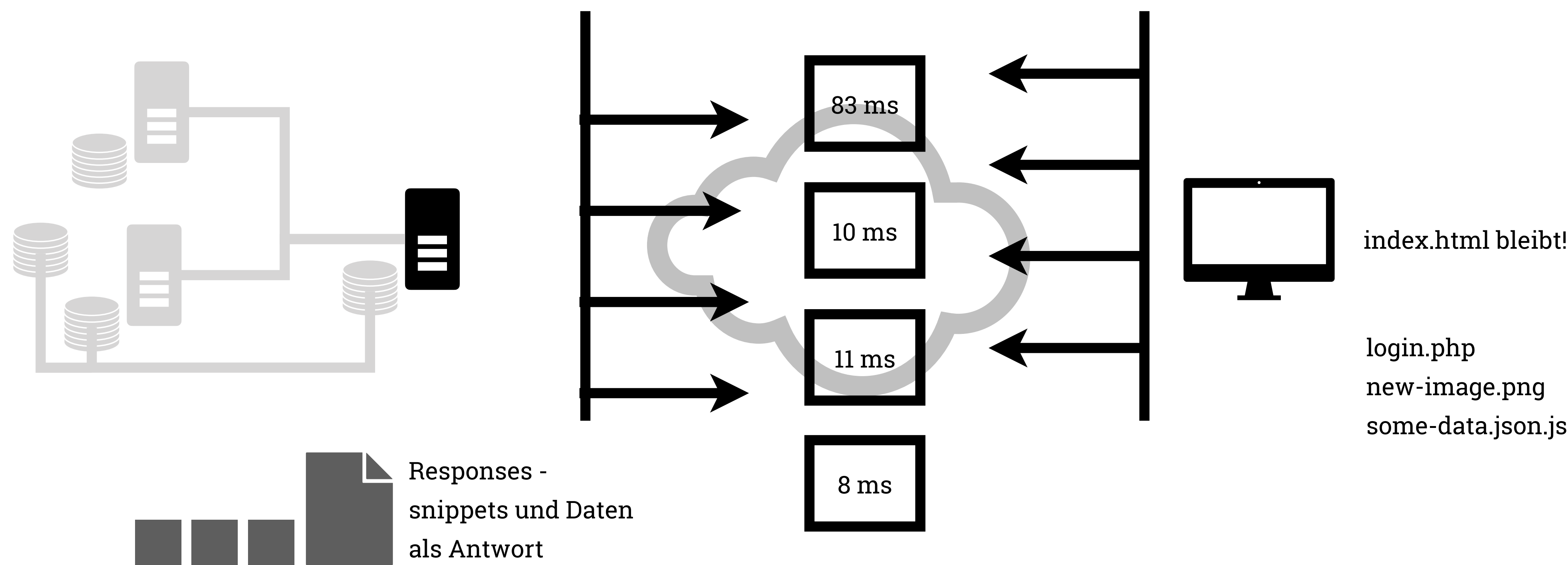
# SINGLE PAGE APPLICATION

---



# SINGLE PAGE APPLICATIONS MIT AJAX

---





## WEBSOCKET - ECHTZEITDATEN VOM SERVER

---

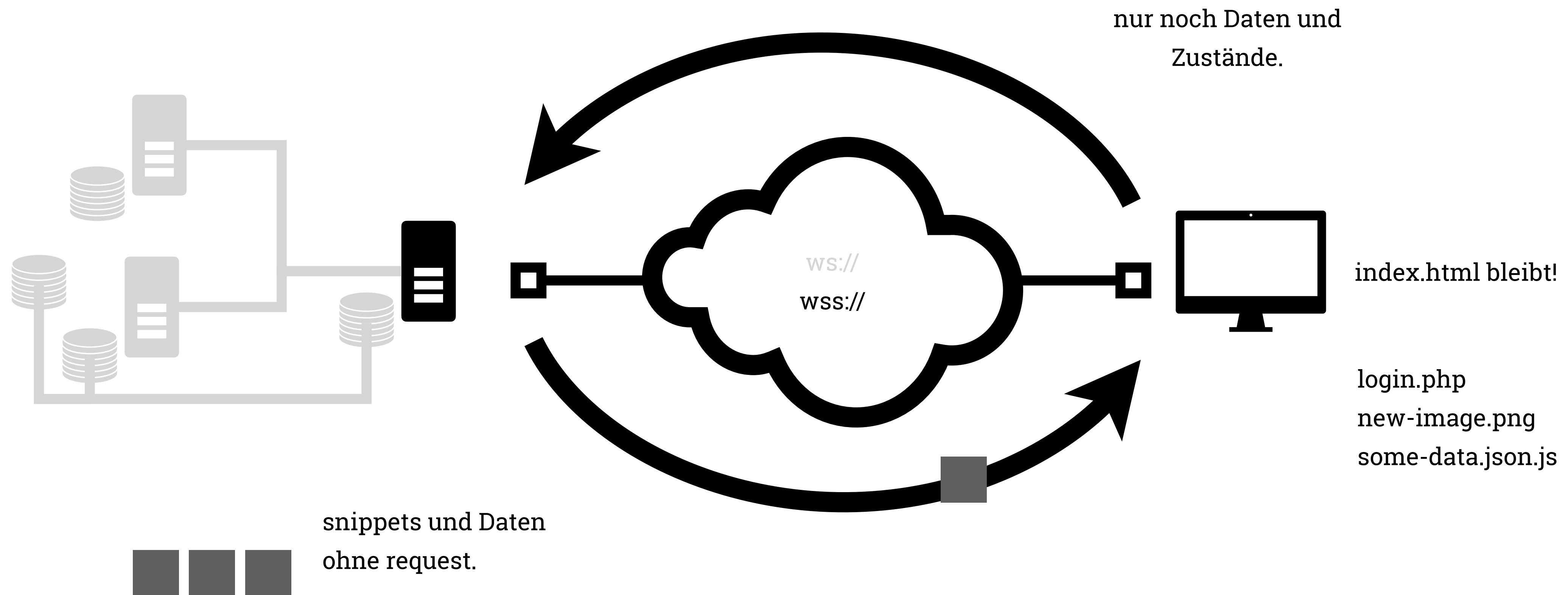
Über Websockets kann eine Echtzeitverbindung zwischen Client und Server aufgebaut werden.

Eine WebSocketverbindung ist konsistent und verzichtet auf die Einschränkungen von http.

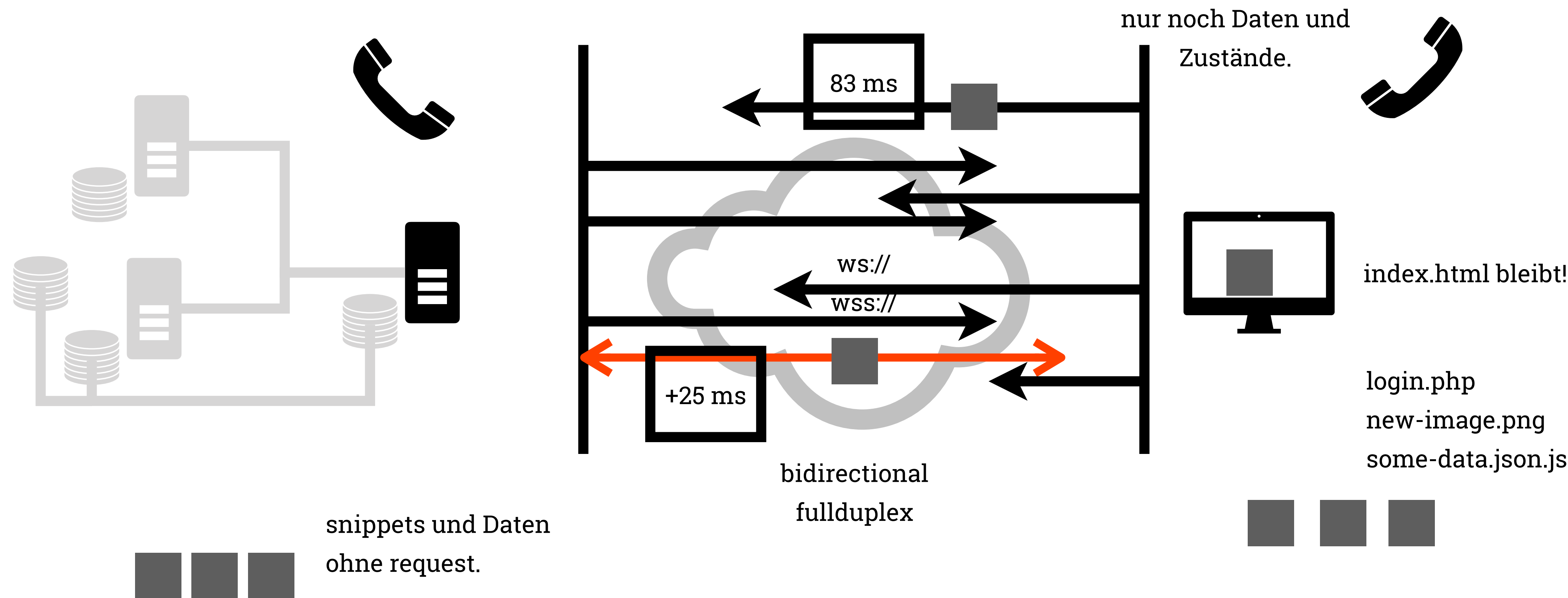
Über das Protokoll ws:// (wss://) können Daten beliebig hin und her geschickt werden.

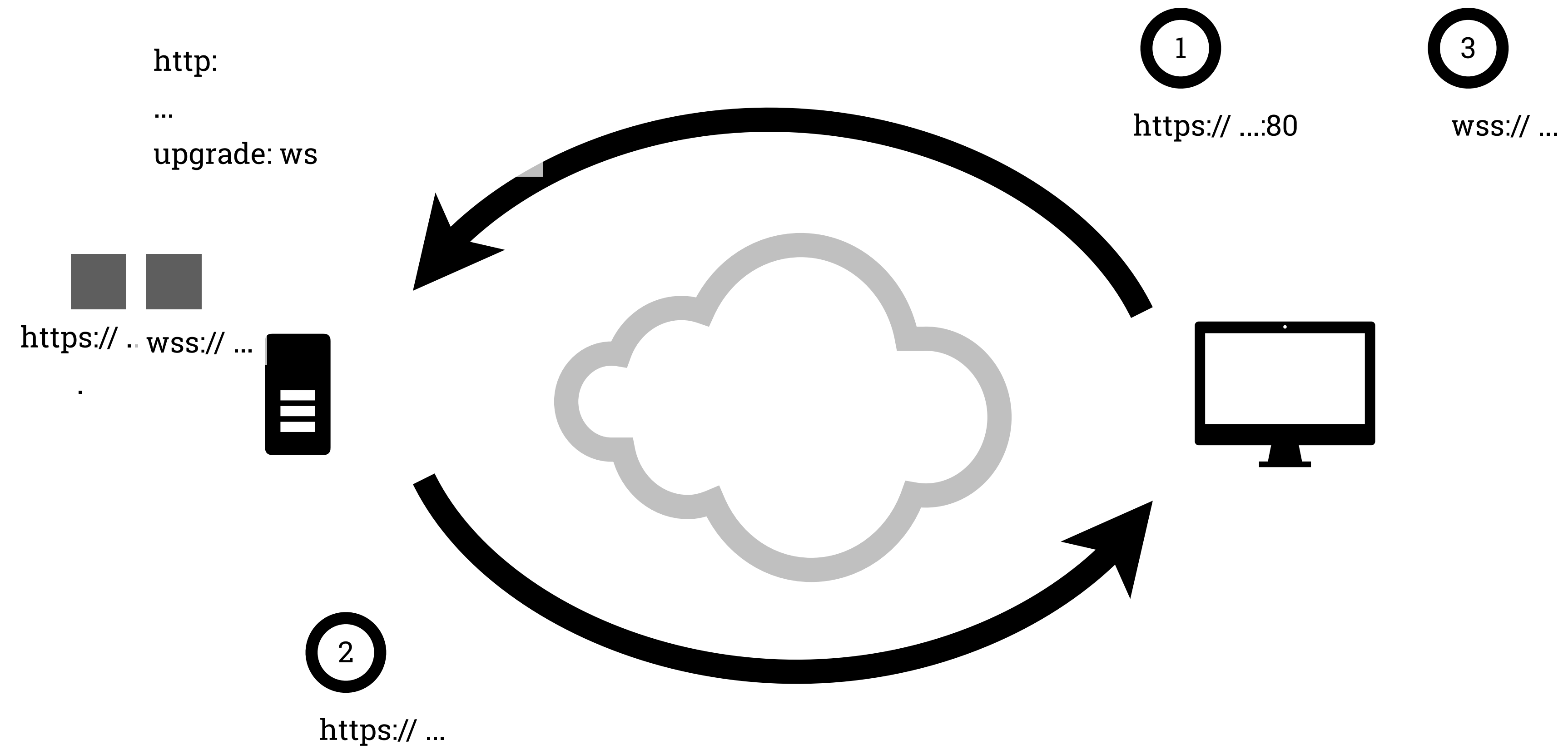
# WEBSOCKET VERBINDUNGEN

---



# WEBSOCKET VERBINDUNGEN





# DIE WEBSOCKET API (CLIENT)

```
ws = new WebSocket('wss://localhost:6969');

ws.onopen = () => {
  console.log('Connection opened!');
}

ws.onmessage = ({data}) => showMessage(data);

ws.onclose = function () {
  ws = null;
}

sendBtn.onclick = function () {
  if (!ws) { showMessage("No WebSocket connection :("); return; }

  ws.send(messageBox.value);
  showMessage(messageBox.value);
}
```

# Can I use

# websocket

? ⚙ Settings

21 results found


✓ Caniuse (1)      ✓ MDN (20)

# WebSocket API

Usage % of **all users** ▴ ▾ ?

Global	95.85%
--------	--------

```
unprefixed: 95.83%
```

Current aligned   Usage relative   Date relative   Filtered   All   

[illegible]

# **A VERY SIMPLE CHAT CLIENT BASED ON WEBSOCKETS**



## THE MOST SIMPLEST CHAT UI

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Simple Websocket Chat</title>
</head>
<body>
  <h1>Simple Chat</h1>
  <pre id="messages">no news so far ...</pre>
  <input type="text" id="messageBox">
  <button id="sendButton">send</button>
  <script src="client.js"></script>
</body>
</html>
```



A dark gray circle containing the white text 'JS'.

## CLIENT.JS - SCAFFOLDING

```
const sendButton = document.querySelector('#sendButton');
const messages = document.querySelector('#messages');
const messageBox = document.querySelector('#messageBox');

let ws;

function showMessage(message) { ... }           // *1
function setCursor() { ... }                     // *2

function init() {
  if (ws) {
    ws.onerror = ws.onopen = ws.onclose = null;
    ws.close();
  }
  ws = new WebSocket('ws://localhost:3000');

  ws.onopen = () => { ... }                       // *3
  ws.onmessage = ({ data }) => showMessage(data);

  ws.onclose = function () { ... }               // *4
}

sendButton.onclick = function (e) { ... }        // *5
init();
```

## \*1 - SHOW MESSAGES

```
// Show messages within the message box and scroll to top

function showMessage(message) {
    messages.textContent += `\n\n${message}`;
    messages.scrollTop = messages.scrollHeight;
    messageBox.value = '';
}
```

## \*2 - SETCURSOR()

```
// Set the cursor into the input field

function setCursor() {
  document.querySelector('input').focus();
}
```

### **\*3 - WS.ONOPEN()**

```
// Confirm open in console  
  
ws.onopen = () => {  
    console.log('Connection opened!');  
    setCursor();  
}
```

## **\*4 - WS.ONCLOSE()**

```
// Closing  
  
ws.onclose = function () {  
    ws = null;  
}
```

## **\*5 - SENDBUTTON.ONCLICK = ...**

```
// Send messages on click
sendButton.onclick = function (e) {
    if (!ws) {
        showMessage("No WebSocket connection :(");
        return;
    }

    ws.send(messageBox.value);
    showMessage(messageBox.value);
    setCursor();
}
```

A dark gray circle containing the white text 'JS'.

## SERVER.JS

```
const express = require('express');
const http = require('http');
const WebSocket = require('ws');

const port = 3000;
const server = http.createServer(express);
const wss = new WebSocket.Server({ server })

wss.on('connection', function connection(ws) {
  ws.on('message', function incoming(data) {
    wss.clients.forEach(function each(client) {
      if (client !== ws && client.readyState === WebSocket.OPEN) {
        client.send(data);
      }
    })
  })
})

server.listen(port, function () {
  console.log(`Server is listening on ${port}!`)
})
```

## SOCKET.IO

---



Tabs sind *single threaded*.

Keine gleichzeitige Ausführung von mehreren parallel arbeitenden Skripten.

Nebenläufige Hintergrundskripte über mehrere *threads*.

Rechenintensive Aufgaben blockieren nicht den sonstigen Ablauf der Webanwendung.



# SOCKET.IO 3.0 IS HERE

FEATURING THE FASTEST AND MOST RELIABLE  
REAL-TIME ENGINE



~/Projects/tweets/index.js

```
1. const io = require('socket.io')(80);  
2. const cfg = require('./config.json');  
3. const tw = require('node-tweet-stream')(cfg);  
4.  
5. tw.track('socket.io');  
6. tw.track('javascript');  
7.  
8. tw.on('tweet', (tweet) => {  
9.   io.emit('tweet', tweet);  
10. });
```

## SOCKET.IO - SERVER.JS

```
const http = require('http');
const express = require('express');
const socketio = require('socket.io');

const app = express();
const server = http.createServer(app);
const io = socketio(server);

io.on('connection', socket => {
  socket.emit('message', 'you joined a chat');
  socket.broadcast.emit('message', 'a new user joined the chat');
  io.emit('message', 'you are all good friends');
})
```

## SOCKET.IO - CLIENT.JS

```
<script src="/socket.io/socket.io.js"></script>
<script src="js/main.js"></script>

- - - -

const socket = io();

socket.on('message', message => {
  outputMessage(message); // do the nice dom stuff
  chatMessages.scrollTop = chatMessages.scrollHeight;
});

socket.emit('chatMessage', msg);
```

# A COMPLEX CHAT CLIENT BASED ON SOCKET.IO

**HTML**



## THE LOGIN PAGE HEAD (INDEX.HTML)

```
<div class="login-container">
  <header class="login-header">
    <h1><i class="far fa-comments"></i> Chat</h1>
  </header>
  <main class="login-main">
    <form action="chat.html">
      <div class="form-control">
        <label for="username">Username</label>
        <input type="text" name="username" id="username" placeholder="Enter username..."
          required autocomplete="off"/>
      </div>
      <div class="form-control">
        <label for="room">Room</label>
        <select name="room" id="room">
          <option value="London">London</option>
          <option value="Paris">Paris</option>
          <option value="Berlin">Berlin</option>
        </select>
      </div>
      <button type="submit" class="btn">Join Chat</button>
    </form>
  </main>
</div>
```



## THE LOGIN PAGE BODY (INDEX.HTML)

```
<div class="login-container">
  <header class="login-header">
    <h1><i class="far fa-comments"></i> Chat</h1>
  </header>
  <main class="login-main">
    <form action="chat.html">
      <div class="form-control">
        <label for="username">Username</label>
        <input type="text" name="username" id="username" placeholder="Enter username..."
          required autocomplete="off"/>
      </div>
      <div class="form-control">
        <label for="room">Room</label>
        <select name="room" id="room">
          <option value="London">London</option>
          <option value="Paris">Paris</option>
          <option value="Berlin">Berlin</option>
        </select>
      </div>
      <button type="submit" class="btn">Join Chat</button>
    </form>
  </main>
</div>
```



## THE CHAT PAGE BODY (INDEX.HTML)

```
<div class="chat-container">
  <header class="chat-header">
    <h1><i class="far fa-comments"></i> ChatCord</h1>
    <a href="index.html" class="btn">Leave Room</a>
  </header>
  <main class="chat-main">
    <div class="chat-sidebar">
      <h3><i class="fas fa-person-booth"></i> Room Name:</h3>
      <h2 id="room-name"></h2>
      <h3><i class="fas fa-users"></i> Users</h3>
      <ul id="users"></ul>
    </div>
    <div class="chat-messages"></div>
  </main>
  <div class="chat-form-container">
    <form id="chat-form">
      <input id="msg" type="text" placeholder="Enter Message" required autocomplete="off" />
      <button class="btn"><i class="fas fa-paper-plane"></i> Send</button>
    </form>
  </div>
</div>
```

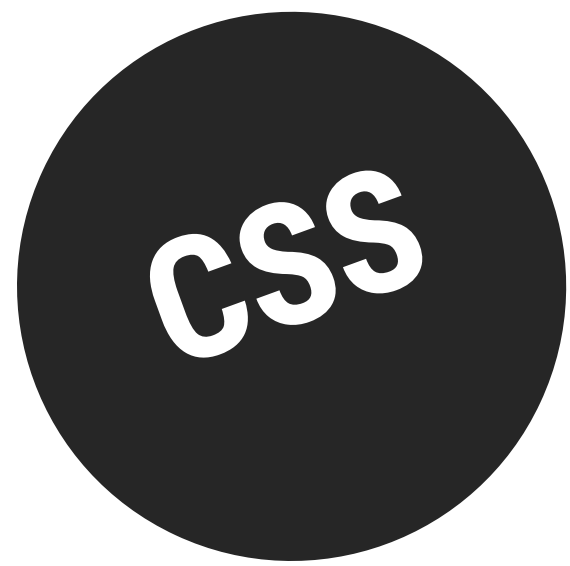




## THE CHAT PAGE SCRIPTS (INDEX.HTML)

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/qs/  
6.9.2/qs.min.js" crossorigin="anonymous"></script>  
<script src="/socket.io/socket.io.js"></script>  
<script src="js/main.js"></script>
```

css



## SOME VARIABLES

```
:root {  
  --dark-color-a: hsla(0, 0%, 15%, 1);  
  --dark-color-b: hsla(0, 0%, 25%, 1);  
  --light-color: hsla(0, 0%, 85%, 1);  
  --success-color: hsla(120, 40%, 50%, 1);  
  --error-color: hsla(0, 60%, 60%, 1);  
  --font-size: 16px;  
  --padding: var(--font-size);  
  --margin: calc(var(--font-size)  
    + var(--font-size) * 0.25)  
}
```

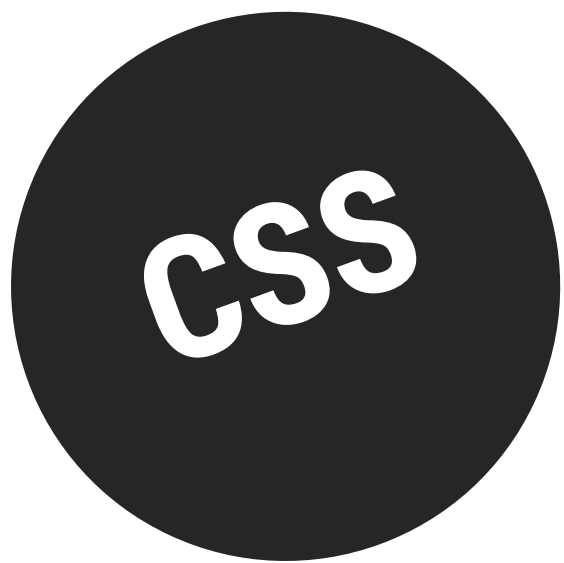


## THE STYLE CSS

```
@import url("https://fonts.googleapis.com/css2?
family=Roboto+Condensed:wght@300;400;700&display=swap");

@import url("https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.12.1/css/all.min.css");

@import url(variables.css);
@import url(base.css);
@import url(login.css);
@import url(chat.css);
```



## THE BASE STYLES

```
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}
body {
  font-family: 'Roboto Condensed',
               sans-serif;
  font-size: var(--font-size);
  background: var(--dark-color-a);
  margin: var(--margin);
}
ul {
  list-style: none;
}

a {
  text-decoration: none;
}
button {
  cursor: pointer;
  border: 0;
  padding: calc(var(--padding)
                / 3) var(--padding);
  color: var(--dark-color-a);
  background: var(--light-color);
  font-size: calc(var(--padding)
                  + var(--padding) * 0.125);
}
```



## THE LOGIN PAGE STYLES

```
.join-container {
  max-width: 500px;
  margin: 80px auto;
  color: #fff;
}

.join-header {
  text-align: center;
  padding: 20px;
  background: var(--dark-color-a);
  border-top-left-radius: 5px;
  border-top-right-radius: 5px;
}

.join-main {
  padding: 30px 40px;
  background: var(--dark-color-b);
}

.join-main p {
  margin-bottom: 20px;
}

.join-main .form-control {
  margin-bottom: 20px;
}

.join-main label {
  display: block;
  margin-bottom: 5px;
}

.join-main input[type='text'] {
  font-size: 16px;
  padding: 5px;
  height: 40px;
  width: 100%;
}

.join-main select {
  font-size: 16px;
  padding: 5px;
  height: 40px;
  width: 100%;
}

.join-main .btn {
  margin-top: 20px;
  width: 100%;
}
```

**SERVER JAVASCRIPT**

A dark gray circle containing the white text 'JS'.

## SERVER.JS - VARIABLES (OR CONSTANTS)

```
const path = require('path');
const http = require('http');
const express = require('express');
const socketio = require('socket.io');
const formatMessage = require('./utils/messages');

const { userJoin, getCurrentUser, userLeave, getRoomUsers }
    = require('./utils/users');

const app = express();
const server = http.createServer(app);
const io = socketio(server);
const botName = 'Chat Bot';
const PORT = process.env.PORT || 3001;
```



A dark gray circle containing the white text 'JS'.

## SERVER.JS - THE SCRIPT SCAFFOLDING

```
// Set a static folder
app.use(express.static(path.join(__dirname, 'public')));

// Run when client connects
io.on('connection', socket => {

    socket.on('joinRoom', ... );           // *1

    socket.on('chatMessage', ... );        // *2

    socket.on('disconnect', ... );         // *3
});

server.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

A dark gray circle containing the white text 'JS'.

## SERVER.JS - \*1 SOCKET.ON JOIN ROOM

```
socket.on('joinRoom', ({ username, room }) => {  
  const user = userJoin(socket.id, username, room);  
  socket.join(user.room);  
  
  // Welcome current user  
  socket.emit('message', formatMessage(botName, 'Welcome to the Chat!'));  
  
  // Broadcast when a user connects  
  socket.broadcast.to(user.room).emit(  
    'message',  
    formatMessage(botName, `${user.username} has joined the chat`)  
  );  
  
  // Send users and room info  
  io.to(user.room).emit('roomUsers', {  
    room: user.room,  
    users: getRoomUsers(user.room)  
  });  
});
```

A dark gray circle containing the white text 'JS'.

## SERVER.JS - \*2 SOCKET.ON CHAT MESSAGE

```
// Listen for chatMessage
socket.on('chatMessage', msg => {
  const user = getCurrentUser(socket.id);

  io.to(user.room).emit(
    'message',
    formatMessage(user.username, msg)
  );
});
```

A dark gray circle containing the white text 'JS'.

## SERVER.JS - \*2 SOCKET.ON CHAT MESSAGE

```
// Runs when client disconnects
socket.on('disconnect', () => {
  const user = userLeave(socket.id);

  if (user) {
    io.to(user.room).emit(
      'message',
      formatMessage(botName, `${user.username} has left the chat`)
    );

    // Send users and room info
    io.to(user.room).emit('roomUsers', {
      room: user.room,
      users: getRoomUsers(user.room)
    });
  }
});
```

A dark gray circle containing the white text 'JS'.

## UTILS/MESSAGES.JS

```
const moment = require('moment');

function formatMessage(username, text) {
  return {
    username,
    text,
    time: moment().format('h:mm a')
  };
}

module.exports = formatMessage;
```

A dark gray circle containing the white text "JS".

## UTILS/USERS.JS

```
const users = [];  
  
// Join user to chat  
function userJoin(id, username, room) {  
  const user = { id, username, room };  
  users.push(user);  
  return user;  
}  
  
// Get current user  
function getCurrentUser(id) {  
  return users.find(user => user.id === id);  
}  
  
// User leaves chat  
function userLeave(id) {  
  const index = users.findIndex(user => user.id === id);  
  if (index !== -1) return users.splice(index, 1)[0];  
}  
  
// Get room users  
function getRoomUsers(room) {  
  return users.filter(user => user.room === room);  
}  
  
module.exports = { userJoin, getCurrentUser, userLeave, getRoomUsers };
```

**CLIENT JAVASCRIPT**

A dark gray circle containing the white text 'JS'.

## MAIN.JS - VARIABLES (OR CONSTANTS)

```
const chatForm
    = document.getElementById('chat-form');
const chatMessages
    = document.querySelector('.chat-messages');
const roomName
    = document.getElementById('room-name');
const userList
    = document.getElementById('users');

// Destructuring of username and room from URL
const { username, room }
    = Qs.parse(location.search, { ignoreQueryPrefix: true });

const socket = io();
```



A dark gray circle containing the white text "JS".

## MAIN.JS - BROADCASTING "USER JOINS THE ROOM"

```
// Join chatroom
```

```
socket.emit('joinRoom', { username, room });
```

A dark gray circle containing the letters 'JS' in a white, bold, sans-serif font.

## MAIN.JS - EVENTHANDLING "USER JOINS THE ROOM" AND "MESSAGE"

```
// Get room and users
socket.on('roomUsers', ({ room, users }) => {
  outputRoomName(room);
  outputUsers(users);
});

// Message from server
socket.on('message', message => {
  outputMessage(message);

  // Scroll down
  chatMessages.scrollTop = chatMessages.scrollHeight;
});
```

A dark gray circle containing the white text 'JS'.

## MAIN.JS - SUBMIT A MESSAGE AND CLEAR THE INPUT

```
// Message submit
chatForm.addEventListener('submit', e => {
  e.preventDefault();

  // Get message text
  let msg = e.target.elements.msg.value;
  msg = msg.trim();

  if (!msg) return false;

  // Emit message to server
  socket.emit('chatMessage', msg);

  // Clear input
  e.target.elements.msg.value = '';
  e.target.elements.msg.focus();
});
```

A dark gray circle containing the white text 'JS'.

## MAIN.JS - THE FUNCTION TO PUT OUT A MESSAGE

```
// Output message to DOM
function outputMessage(message) {
  const div = document.createElement('div');
  div.classList.add('message');

  const p = document.createElement('p');
  p.classList.add('meta');
  p.innerText = message.username;
  p.innerHTML += ` <span>${message.time}</span>`;
  div.appendChild(p);

  const para = document.createElement('p');
  para.classList.add('text');
  para.innerText = message.text;
  div.appendChild(para);

  document.querySelector('.chat-messages').appendChild(div);
}
```

A dark gray circle containing the white text 'JS'.

## MAIN.JS - FUNCTIONS TO ADD ROOM AND USER TO THE DOM

```
// Add room name to DOM
function outputRoomName(room) {
  roomName.innerText = room;
}

// Add users to DOM
function outputUsers(users) {
  userList.innerHTML = '';
  users.forEach(user => {
    const li = document.createElement('li');
    li.innerText = user.username;
    userList.appendChild(li);
  });
}
```

**CLIENT JAVASCRIPT**

A dark gray circle containing the white text 'JS'.

## MAIN.JS - VARIABLES (OR CONSTANTS)

```
const chatForm
    = document.getElementById('chat-form');
const chatMessages
    = document.querySelector('.chat-messages');
const roomName
    = document.getElementById('room-name');
const userList
    = document.getElementById('users');

// Destructuring of username and room from URL
const { username, room }
    = Qs.parse(location.search, { ignoreQueryPrefix: true });

const socket = io();
```

A dark gray circle containing the white text "JS".

## MAIN.JS - BROADCASTING "USER JOINS THE ROOM"

```
// Join chatroom
```

```
socket.emit('joinRoom', { username, room });
```



A dark gray circle containing the white text "JS".

## MAIN.JS - EVENTHANDLING "USER JOINS THE ROOM" AND "MESSAGE"

```
// Get room and users
socket.on('roomUsers', ({ room, users }) => {
  outputRoomName(room);
  outputUsers(users);
});

// Message from server
socket.on('message', message => {
  outputMessage(message);

  // Scroll down
  chatMessages.scrollTop = chatMessages.scrollHeight;
});
```

A dark gray circle containing the white text 'JS' in a bold, sans-serif font.

## MAIN.JS - SUBMIT A MESSAGE AND CLEAR THE INPUT

```
// Message submit
chatForm.addEventListener('submit', e => {
  e.preventDefault();

  // Get message text
  let msg = e.target.elements.msg.value;
  msg = msg.trim();

  if (!msg) return false;

  // Emit message to server
  socket.emit('chatMessage', msg);

  // Clear input
  e.target.elements.msg.value = '';
  e.target.elements.msg.focus();
});
```

A dark gray circle containing the white text 'JS'.

## MAIN.JS - THE FUNCTION TO PUT OUT A MESSAGE

```
// Output message to DOM
function outputMessage(message) {
  const div = document.createElement('div');
  div.classList.add('message');

  const p = document.createElement('p');
  p.classList.add('meta');
  p.innerText = message.username;
  p.innerHTML += ` <span>${message.time}</span>`;
  div.appendChild(p);

  const para = document.createElement('p');
  para.classList.add('text');
  para.innerText = message.text;
  div.appendChild(para);

  document.querySelector('.chat-messages').appendChild(div);
}
```

A dark gray circle containing the letters 'JS' in a white, bold, sans-serif font.

## MAIN.JS - FUNCTIONS TO ADD ROOM AND USER TO THE DOM

```
// Add room name to DOM
function outputRoomName(room) {
  roomName.innerText = room;
}

// Add users to DOM
function outputUsers(users) {
  userList.innerHTML = '';
  users.forEach(user => {
    const li = document.createElement('li');
    li.innerText = user.username;
    userList.appendChild(li);
  });
}
```

## WEBWORKER - MULTITHREADING

---

Tabs sind *single threaded*.

Keine gleichzeitige Ausführung von mehreren parallel arbeitenden Skripten.

Nebenläufige **Hintergrundskripte** über mehrere *threads*.

Rechenintensive Aufgaben blockieren nicht den sonstigen Ablauf der Webanwendung.

**WARNING: UNRESPONSIVE SCRIPT**



A scripts on this page may be busy, or it may have stopped responsding.  
You can stop the script now, or you can continue see if the scripts  
will complete.

**STOP SCRIPT**

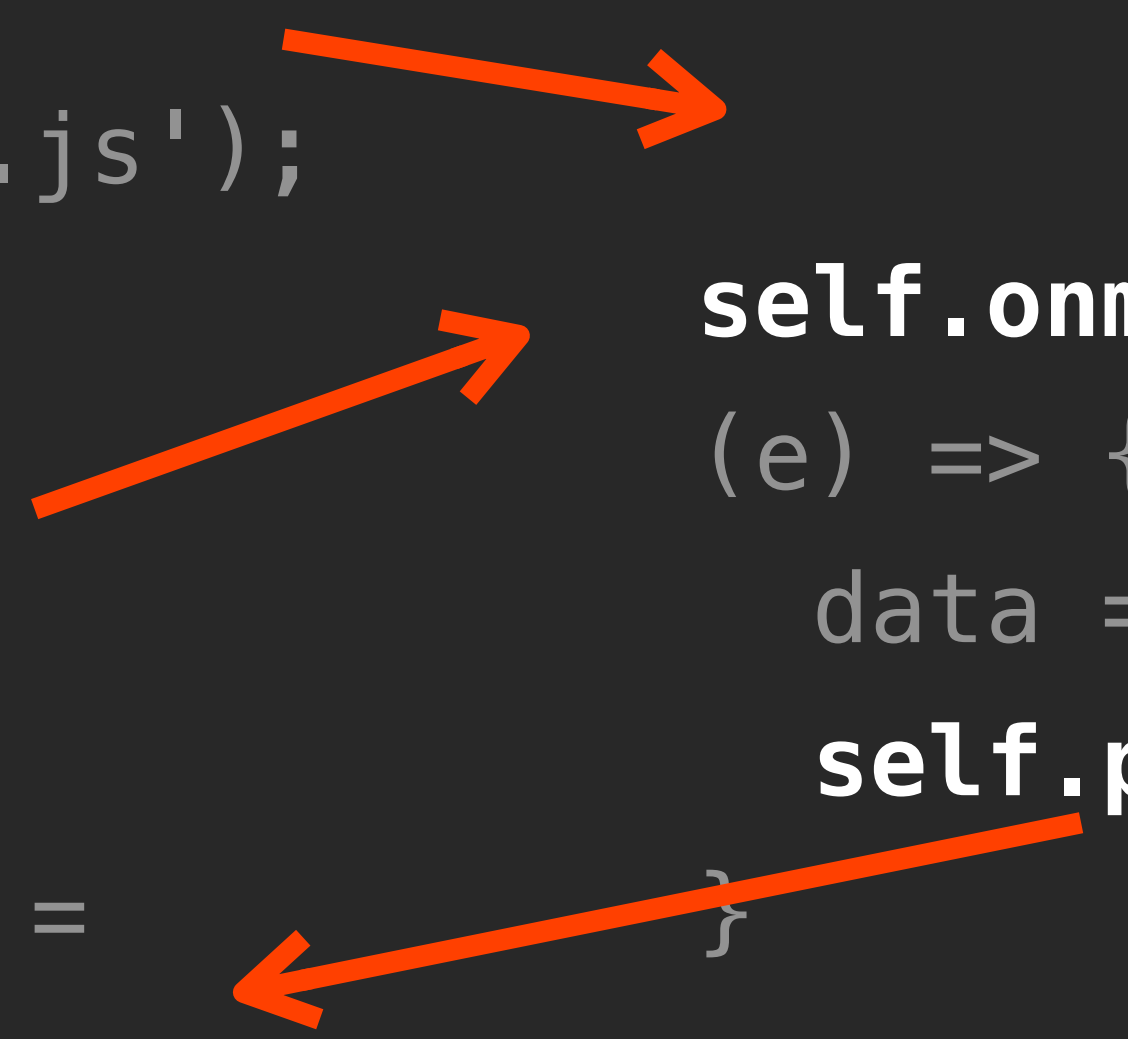
**CONTINUE**

## DIE WORKER API

```
// main.js:                                // task.js:
let worker =
  new Worker('task.js');

worker.onmessage =
  (e)=>e.data;

                                self.onmessage =
                                (e) => {
                                    data = resultOfWork;
                                    self.postMessage(data);
                                }
```



## MAIN.JS - EXAMPLE

```
(function () {

    let worker = new Worker('task.js');

    worker.onerror = () => {console.log('something went wrong.')}

    worker.onmessage = function (event) {
        console.log('worker started ...');
        console.dir(event.data);
    };

    worker.onclose = () => { console.log('worker closed!')};

    worker.postMessage({
        "message": "hello worker"
    });

    worker.terminate();
})();
```



## TASK.JS - EXAMPLE

```
onmessage = function (event) {  
  
    console.log('yes i\'m running');  
  
    console.dir(event.data);  
    console.dir(self);  
  
    postMessage({ message: "yes, hello" });  
  
    self.close();  
}
```

# Can I use

# webworker

? ⚙ Settings

1 result found

# Web Workers - LS

## Usage

% of **all users** ?

Global

97.73%

Method of running scripts in the background, isolated from the web page

Current aligned

Usage relative

Date relative

Filtered

All

[illegible]

# GELTUNGSBEREICH DES WORKERS

---

Aufgrund ihres Multithread-Verhaltens können Web Worker nur auf einen Teil der Funktionen von JavaScript zugreifen.

Worker haben **KEINEN** Zugriff auf:

- ▶ Das **navigator**-Objekt
  - ▶ Das **location**-Objekt (schreibgeschützt)
  - ▶ **XMLHttpRequest**
  - ▶ **setTimeout()/clearTimeout()** und **setInterval()/clearInterval()**
  - ▶ Den **Anwendungscache**
  - ▶ Import externer Skripts mithilfe der **importScripts()**-Methode
  - ▶ Erzeugen weiterer Web Worker
- ▶ Das DOM (nicht threadsicher)
  - ▶ Das window-Objekt
  - ▶ Das document-Objekt
  - ▶ Das parent-Objekt

## EXTERNE SKRIPTS LADEN

---

Mithilfe der `importScripts()`-Funktion können Sie externe Skriptdateien oder -bibliotheken in einen Worker laden. Bei dieser Methode können null oder mehr Zeichenfolgen als Dateinamen der zu importierenden Ressourcen angegeben werden.

```
// worker.js:
```

```
importScripts('script1.js');  
importScripts('script2.js');
```

Dies lässt sich auch in einer Importanweisung zusammenfassen:

```
importScripts('script1.js',  
             'script2.js');
```

# UNTERGEORDNETE WORKER

---

Worker können untergeordnete Worker erzeugen. So lassen sich große Aufgaben während der Laufzeit weiter unterteilen.

Nachteile:

- ▶ Beim Hosten untergeordneter Worker müssen Sie sich an der übergeordneten Seite orientieren.
- ▶ URIs in untergeordneten Workern werden in Abhängigkeit vom Speicherort des übergeordneten Workers aufgelöst (im Gegensatz zur Hauptseite).

Beachten Sie, dass die meisten Browser für jeden Worker separate Prozesse erzeugen. Bevor Sie eine Worker-Farm erzeugen, vergewissern Sie sich, dass Sie nicht zu viele Systemressourcen des Nutzers beanspruchen.

## SHARED WORKER

---

Ein gewöhnlicher Worker gehört immer zu der Seite, die ihn erzeugt hat.

Ruft ein Benutzer die Seite mehrfach in seinem Browser auf (oder verschiedene Seiten derselben Domain, die den gleichen Worker einsetzen), *so werden mehrere Worker mit dem gleichen Code initialisiert.*

Setzt man stattdessen einen SharedWorker ein, so wird dieser wiederverwendet.

# Can I use

shared worker

? ⚙ Settings

13 results found

☒ Caniuse (2)

☒ MDN (11)

## Shared Web Workers - LS

Method of allowing multiple scripts to communicate with a single web worker.

## Usage

% of

all users

35.11%

Current aligned

Usage relative

Date relative

Filtered

All

[illegible]

## SHARED-WORKER

```
var worker = new SharedWorker("shared-worker.js");  
  
// Alle Seiten, die sich eine Instanz des Shared Workers  
// erstellen, arbeiten im Hintergrund mit demselben  
Worker.
```



## SICH MIT EINEM SHARED WORKER VERBINDEN

```
/* Ein Shared Worker besitzt Ports für die verschiedenen
Seiten, über die er kommuniziert. Diese API ist der HTML5
Messaging API ähnlich. */

var worker = new SharedWorker("/html5/web-worker-shared.js");
worker.port.addEventListener("message",
    function(event) {
        alert(event.data);
    }
    , false
);
worker.port.start();
```

## EINE NACHRICHT AN DEN SHARED WORKER SCHICKEN

```
/* Wenn der Port gestartet ist und die Seite auf Messages  
hört, lassen sich Nachrichten an den Shared Worker  
schicken */  
  
worker.port.postMessage(  
    "Meine Nachricht"  
);
```

## INLINE-WORKER

---

Nehmen wir an, Sie möchten Ihr Worker-Skript spontan erstellen oder eine eigenständige Seite erzeugen, ohne separate Worker-Dateien erstellen zu müssen.

Mit der neuen BlobBuilder-Oberfläche können Sie Ihren Worker in die gleiche HTML-Datei wie Ihre Hauptlogik "einbetten". Dazu erstellen Sie einen BlobBuilder und hängen den Worker-Code als Zeichenfolge an.

## INLINE-WORKER

```
let
  bb = new BlobBuilder(),
  string = "onmessage = function(e) { postMessage('msg from worker'); }"

bb.append(string);

let blobURL = window.URL.createObjectURL(bb.getBlob());
let worker = new Worker(blobURL);

worker.onmessage = function(e) {
  // e.data == 'msg from worker'
};

worker.postMessage(); // Start the worker
```

# Can I use

blob

? ⚙ Settings

23 results found

✓ Caniuse (3)      ✓ MDN (20)


## Blob URLs - WD

Method of creating URL handles to the specified File or Blob object.

Usage % of **all users** ▴ ▾ ?

Global	97.55%
--------	--------

```
unprefixed: 97.32%
```

Current aligned Usage relative Date relative Filtered All 

[illegible]

# SERVICE-WORKER

---

Ein Service-Worker kann vom Server gesendete Benachrichtigungen (selbst dann) empfangen, wenn gerade keine Seite der entsprechenden Domain geöffnet ist. Damit können **beispielsweise Push-Benachrichtigungen** realisiert werden, die auch ohne geöffnete Seite funktionieren.

Für seine zweite Aufgabe als **Proxy** steht im Worker das fetch-Event zur Verfügung, das immer ausgelöst wird, wenn der Browser Daten der überwachten Domain anfordert, unabhängig davon, ob dies durch AJAX, Benutzernavigation oder andere Ursachen ausgelöst wird.


Der Worker fängt das Event ab und liefert bei Bedarf die angeforderten Daten auf andere Weise.

Die Spezifikation ist aber so flexibel, dass **zahlreiche weitere Möglichkeiten** über einen einfachen Cache hinaus möglich sind.

? ⚙ Settings

✓ Caniuse (1)      ✓ MDN (49)

Usage	% of	all users	?
Global	94.7%	+ 0.1%	= 94.8%

Current aligned Usage relative Date relative Filtered All 

[illegible]

## SERVICE-WORKER BEISPIEL: REGISTRIEREN

```
if ("serviceWorker" in navigator) {  
  navigator  
    .serviceWorker  
    .register("service-worker.js")  
    .then(function (reg) {  
      console.log("Registered with scope: ", reg.scope);  
    })  
    .catch(function (err) {  
      console.log("ServiceWorker registration failed: ",  
                  err);  
    });  
}
```



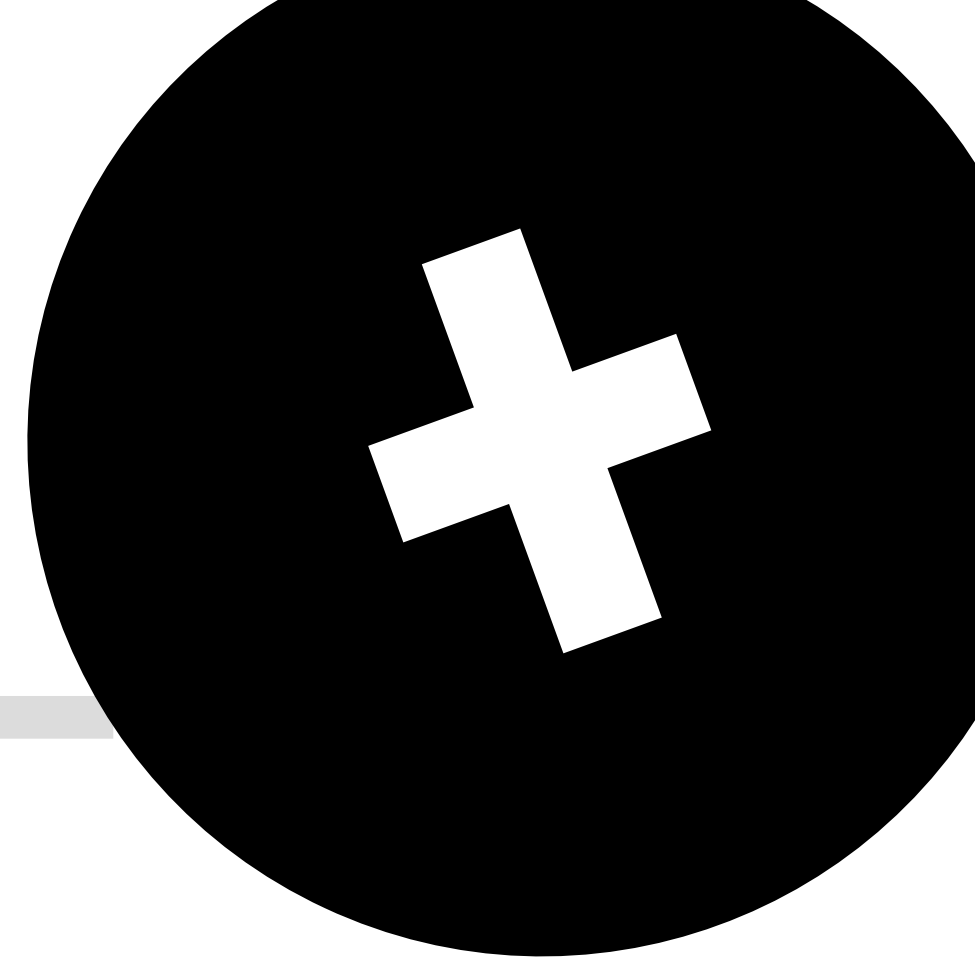
## SERVICE-WORKER BEISPIEL: DATEIEN AUS LISTE CACHEN

```
let cacheName = "myappCache",
    filesToCache = [
      "index.html",
      "js/site.js",
      "css/style.css", ...
    ];

self.addEventListener("install", function (e) {
  e.waitUntil(
    caches.open(cacheName).then(function (cache) {
      return cache.addAll(filesToCache);
    })
  );
});
```

# DATENAUSTAUSCH TECHNOLOGIEN

---



- ▶ RDF
- ▶ XML
- ▶ **JSON**
- ▶ ODATA

# JSON - JAVASCRIPT OBJECT NOTATION

---

Die JavaScript Object Notation ist ein kompaktes Datenformat in einer einfach lesbaren Textform und dient dem Zweck des Datenaustausches zwischen Anwendungen.

JSON ist von der Programmiersprache unabhängig. Parser und Generatoren existieren in allen verbreiteten Sprachen.

JSON wurde ursprünglich von Douglas Crockford spezifiziert. Stand Ende 2017 wird es durch zwei unterschiedliche, inhaltlich aber gleiche, Standards spezifiziert – RFC 8259[1] sowie ECMA-404.[2]

# JSON - BEISPIEL

---

```
{
  "name": "Georg",      <- key value pairs
  "alter": 47,
  "verheiratet": false,
  "beruf": null,
  "kinder": [
    {
      "name": "Lukas",
      "alter": 19,
      "schulabschluss": "Gymnasium"
    },
    {
      "name": "Lisa",
      "alter": 14,
      "schulabschluss": null
    }
  ]
}
```

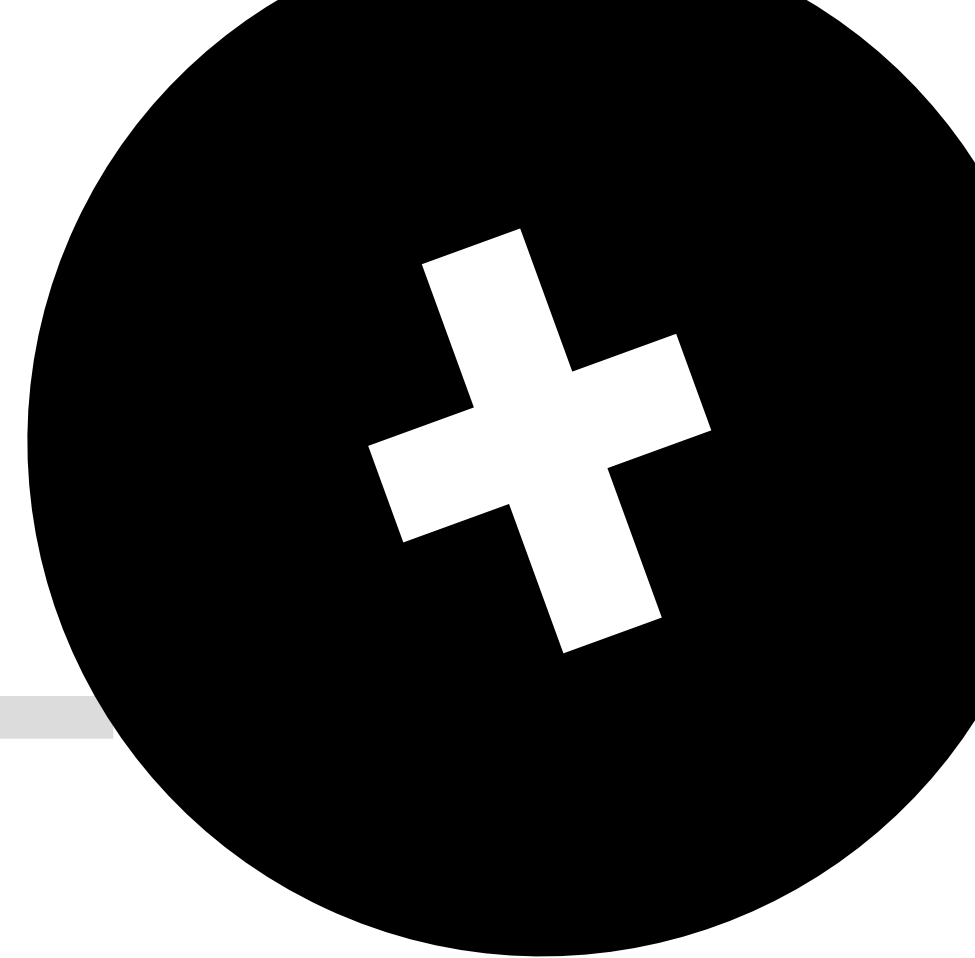
# JSON - JAVASCRIPT OBJECT NOTATION

---

<https://www.json.org/json-de.html>

# VUEJS GRUNDLAGEN

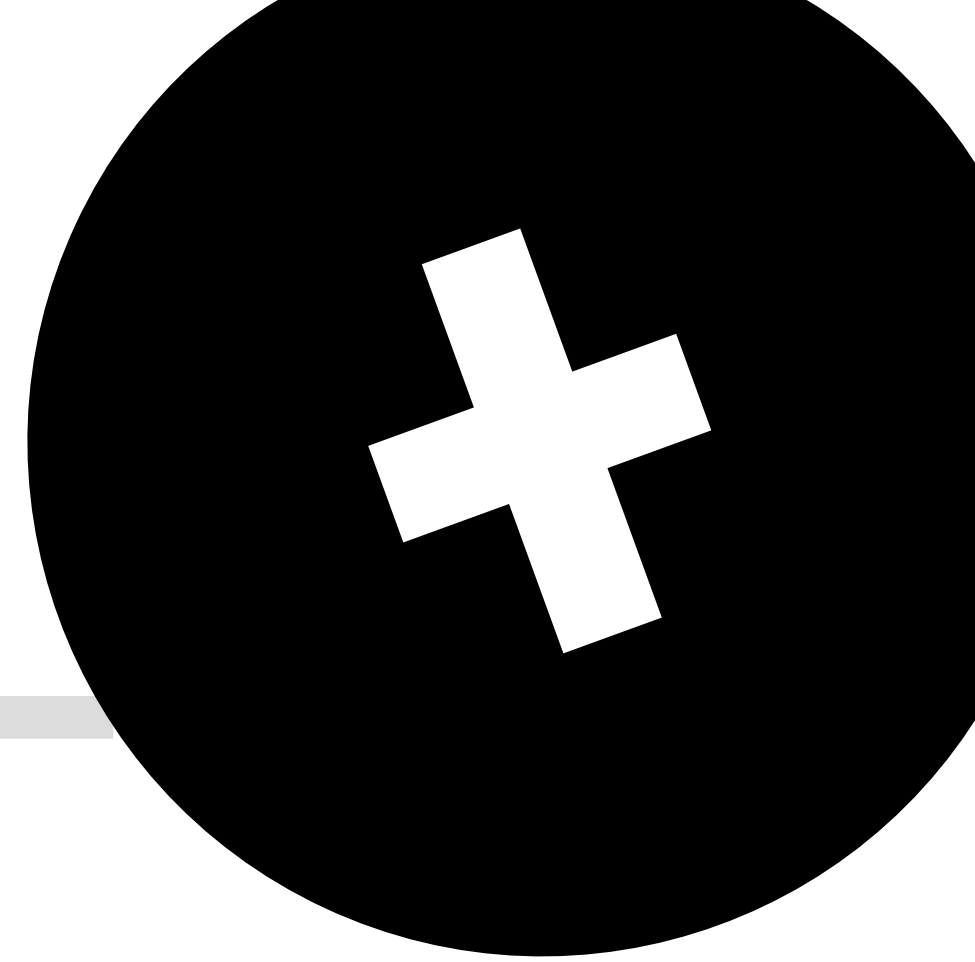
---



- ▶ Vue - Grundbegriffe
  - ▶ Die Vue Instanz
  - ▶ Direktiven
  - ▶ Methoden
  - ▶ Data Binding
  - ▶ Events
  - ▶ Filter
  - ▶ Computed Properties
  - ▶ Components
  - ▶ Vue 2
- ▶ Vue CLI
  - ▶ Vue dev tools
  - ▶ Props & slots
  - ▶ Router
  - ▶ ...

## WAS IST VUE?

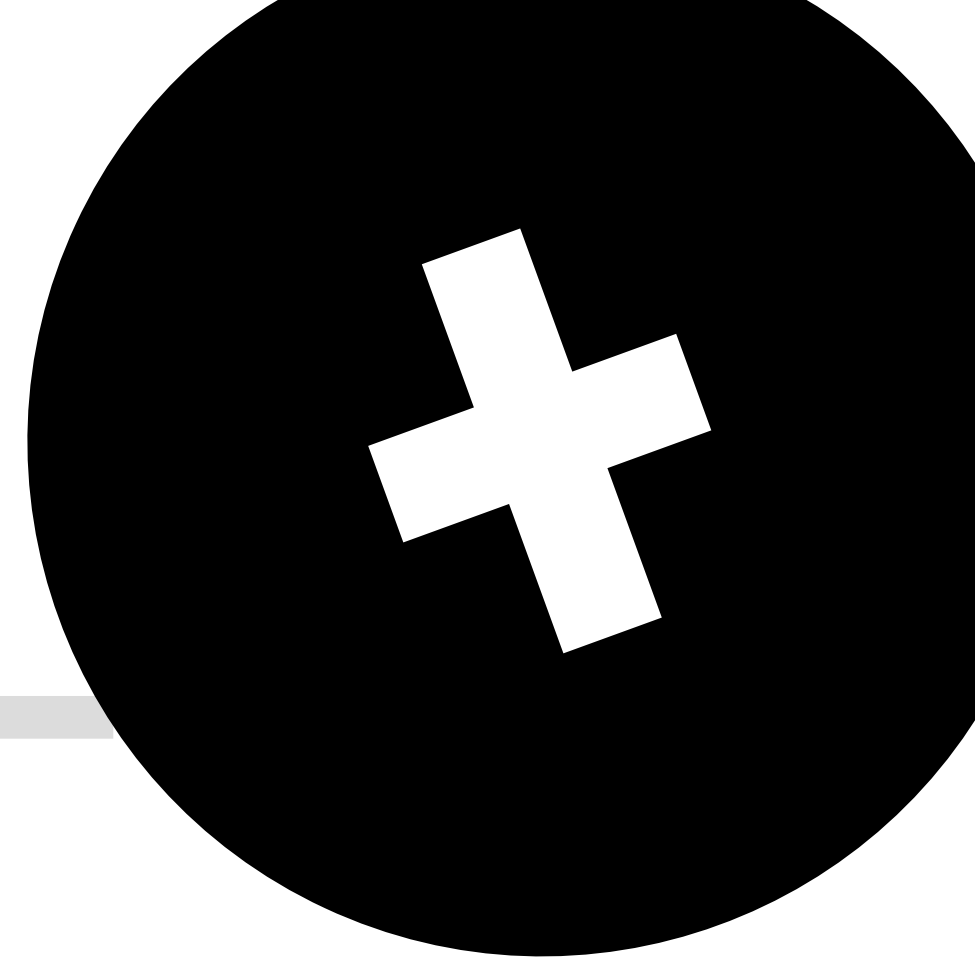
---



- ▶ Vue (oder Vue.js) ist ein Open-Source-Front-End-JavaScript-Framework.
- ▶ Vue ist die View-Schicht einer MVC-Anwendung (Model-View-Controller)
- ▶ Vue ist derzeit eine der populärsten JavaScript-Bibliotheken/Frameworks
- ▶ Im Gegensatz zu anderen populären JavaScript-Projekten wird Vue nicht von einem großen Unternehmen wie React (Facebook) oder Angular (Google) unterstützt.  
Vue wurde ursprünglich von Evan You und der Open-Source-Community geschrieben.

# EINRICHTUNG UND INSTALLATION

---



Es gibt zwei Wege, Vue einzurichten:

- ▶ in einem Node-Projekt
- ▶ in einer statischen HTML-Datei



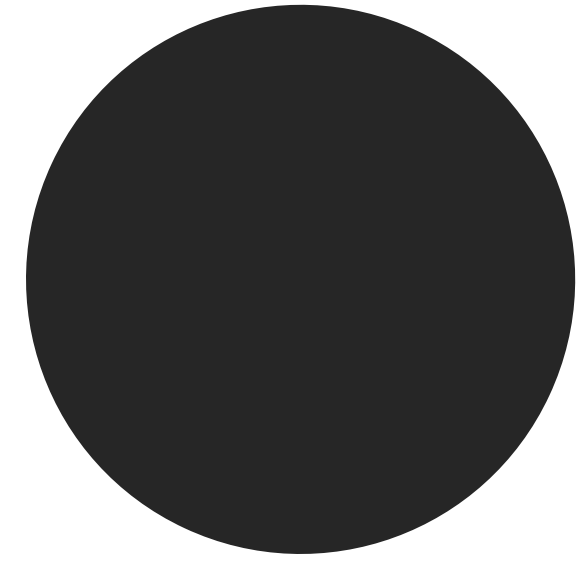


# STATISCHES HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,initial-scale=1.0" />
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

    <title>Vue App</title>
  </head>

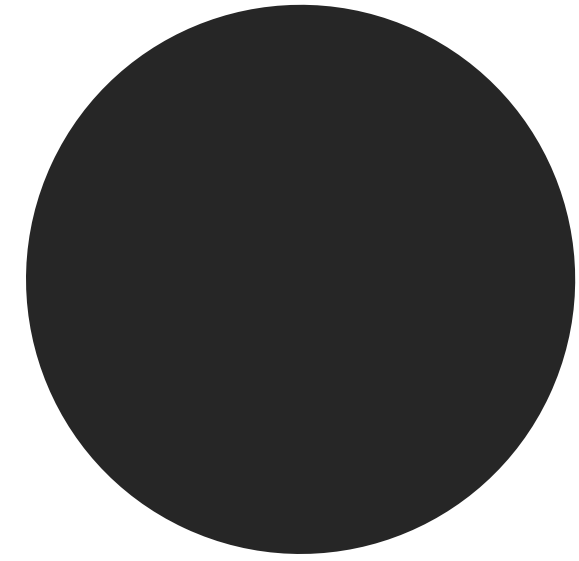
  <body>
    <div id="app"></div>
  </body>
</html>
```



## DATEN AUSGEBEN

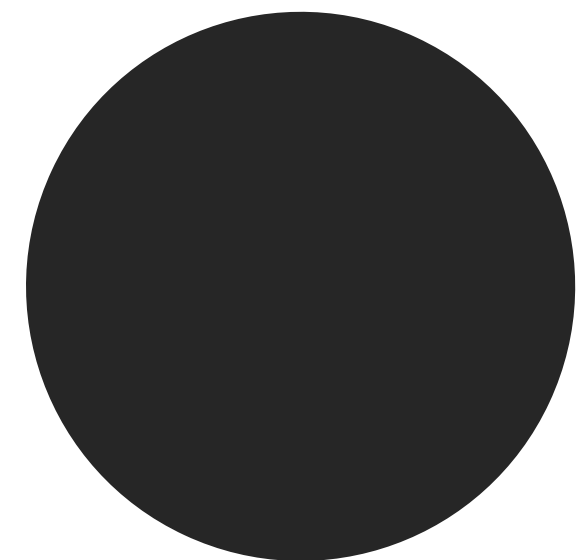
```
<div id="app">{{message}}</div>

<script>
  const App = new Vue({
    el: '#app',
    data: {
      message: 'Hello Vue!',
    },
  })
</script>
```



## VUE CLI GLOBAL INSTALLIEREN

```
# install with npm  
npm install -g @vue/cli  
  
npm i -g @vue/cli @vue/cli-service-global
```



# VUE BOILERPLATE

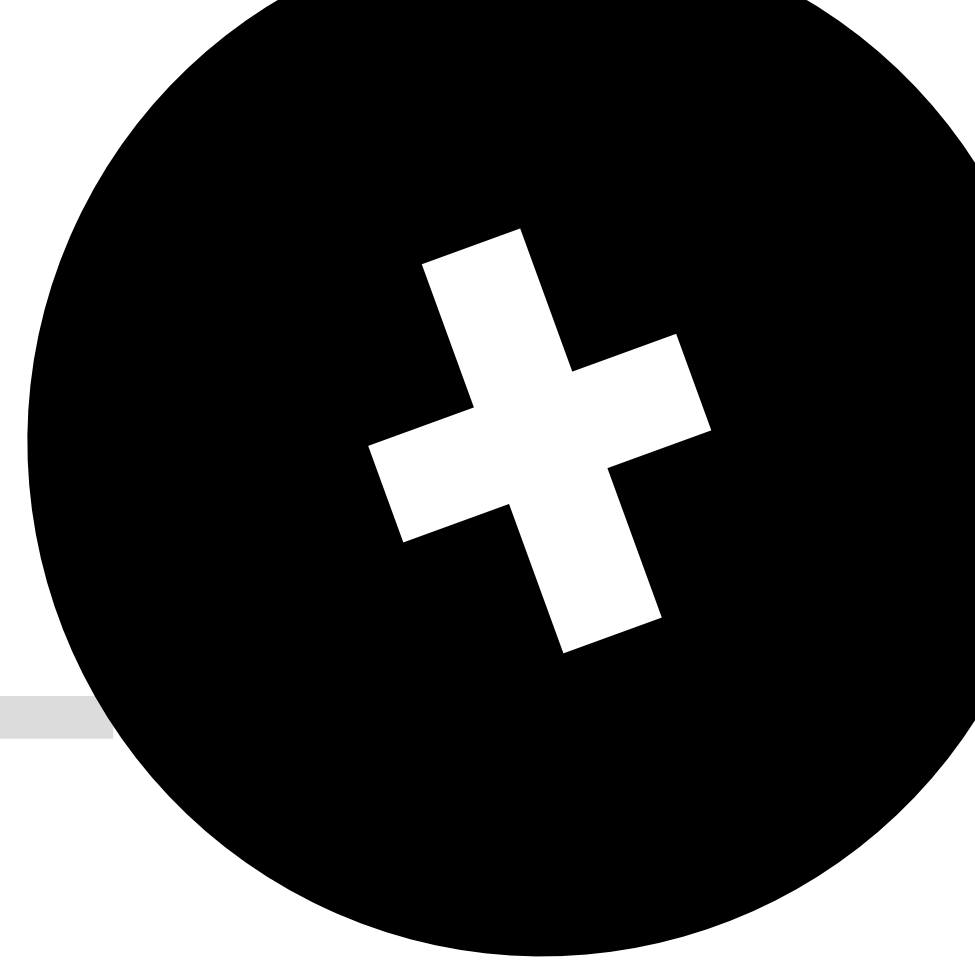
```
vue create vue-app
```

```
cd vue-app
```

```
npm run serve
```

## TOOLS & TUTORIAL

---



- ▶ vetur als Extension für VS Code
- ▶ vue-beautify2 als Extension für VS Code
- ▶ Vue DevTools für Chrome oder Firefox (geht aktuell nicht mit Vue3)
- ▶ <https://www.taniarascia.com/getting-started-with-vue/>