

A thick red vertical bar is positioned on the left side of the slide, extending from the top to the bottom.

Webbasierte Anwendungen

AJAX/Web 2.0

Prof. Dr. Ludger Martin

Gliederung

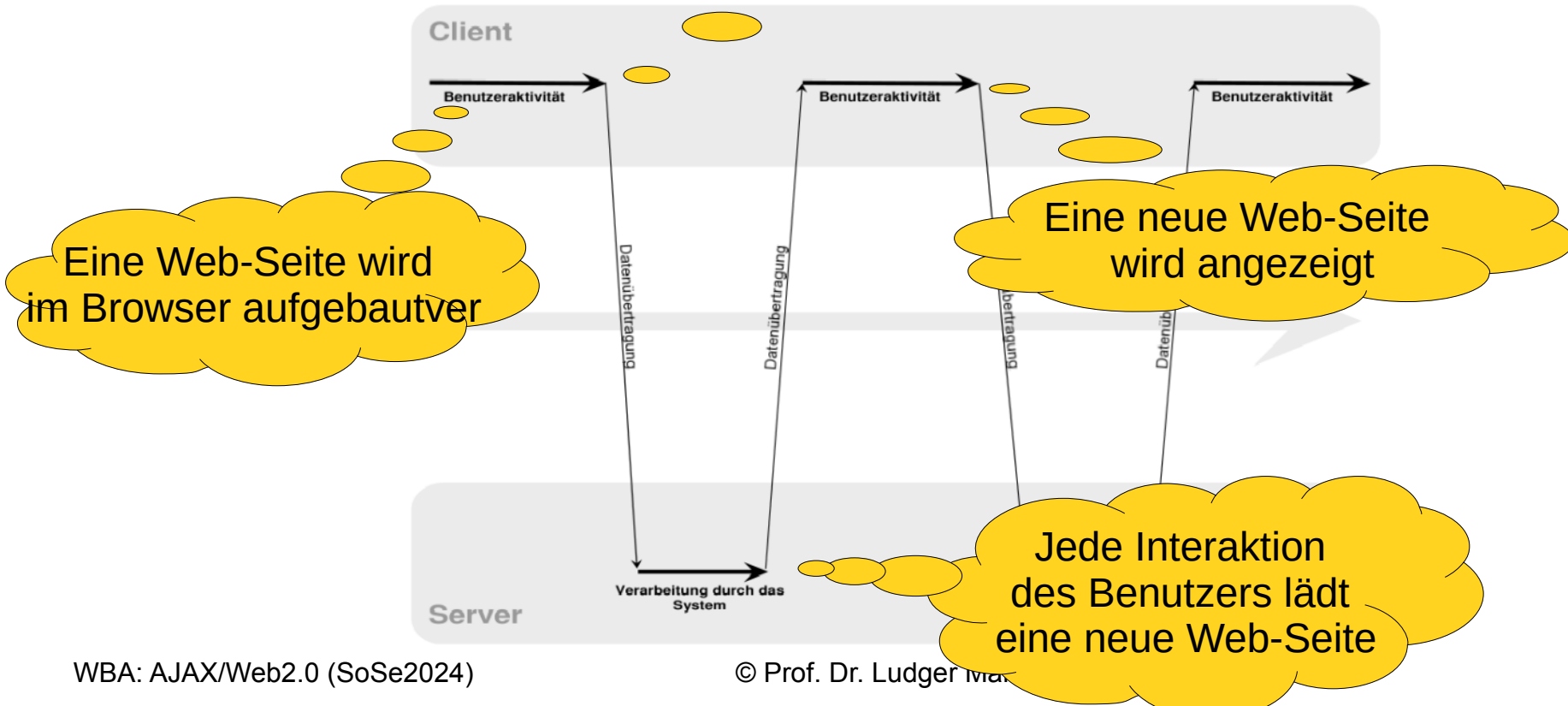
- ◆ Einführung
- ◆ Same-Origin-Policy
- ◆ XMLHttpRequest Objekt
- ◆ Fetch API

Einführung

- ◆ AJAX: **A**synchronous **J**avaScript and **X**ML
- ◆ Auch Web 2.0 genannt
- ◆ Web Dokumente in HTML und CSS
- ◆ DOM für Anzeige und Interaktion
- ◆ Asynchroner Datenaustausch per XMLHttpRequest (W3C seit 2006)
- ◆ Ausgiebige Nutzung von JavaScript

Sie wird angeschaut und
es werden ggf.
Formularfelder ausgefüllt

Der klassische Web-Seite-Zyklus:



Einführung

Der klassische Web-Seiten Zyklus:

- ◆ Eine Web-Seite wird im Browser aufgebaut.
- ◆ Sie wird angeschaut und es werden ggf. Formularfelder ausgefüllt.
- ◆ Jede Interaktion des Benutzers lädt eine neue Web-Seite.
- ◆ Eine geänderte Web-Seite wird angezeigt.
- ◆ usw.

Einführung

Von normalen Desktop-Anwendungen sind wir gewohnt:

- ◆ Eine Benutzungsoberfläche wird geladen.
- ◆ Ein Benutzer tätigt seine Eingaben.
- ◆ Gleichzeitig reagiert die Anwendung auf die Eingaben, ohne dass eine Blockade stattfindet.
- ◆ usw.

Einführung

Der AJAX/Web 2.0-Ansatz:

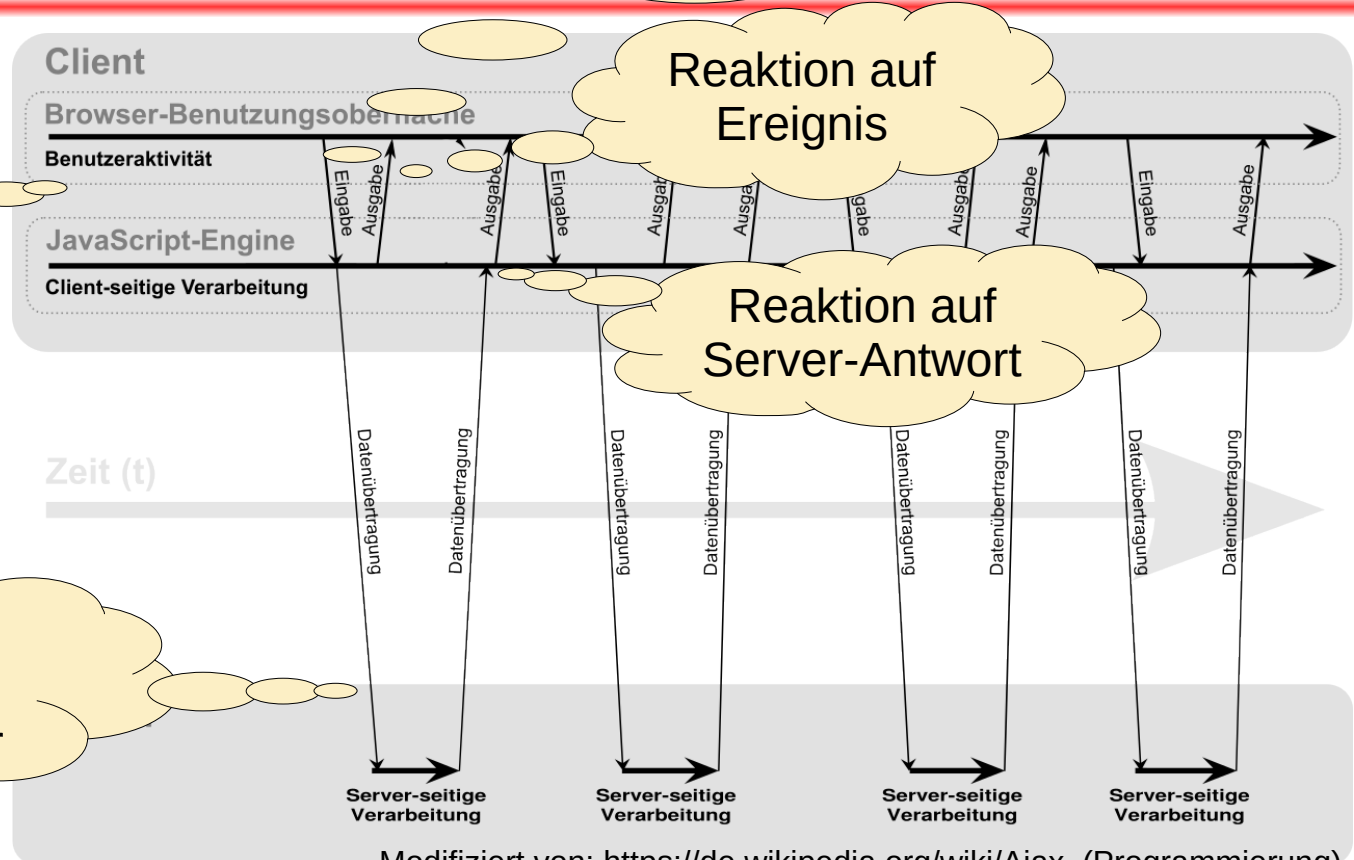
- ◆ Eine Seite wird im Browser aufgebaut.
- ◆ Sie wird angeschaut und es werden ggf. Formularfelder ausgefüllt.
- ◆ Gleichzeitige Kommunikation mit einem Server und dem darauf installierten Service.
- ◆ Die Benutzungsoberfläche wird dynamisch angepasst.
- ◆ USW.

Ein

Ereignis, z.B.
Formularfelder ausgefüllt,
Mausbewegung, ...

AJAX/Web 2.0:

Eine Seite wird im
Browser aufgebaut

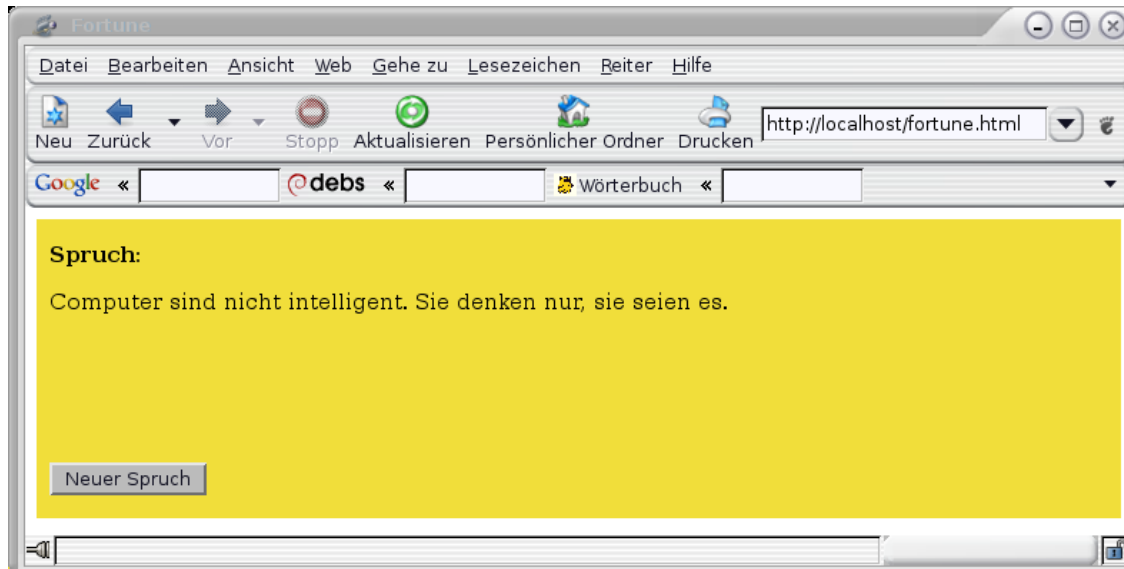


Gleichzeitige
Kommunikation
mit einem Server

Einführung

♦ AJAX/Web 2.0-Beispiel:

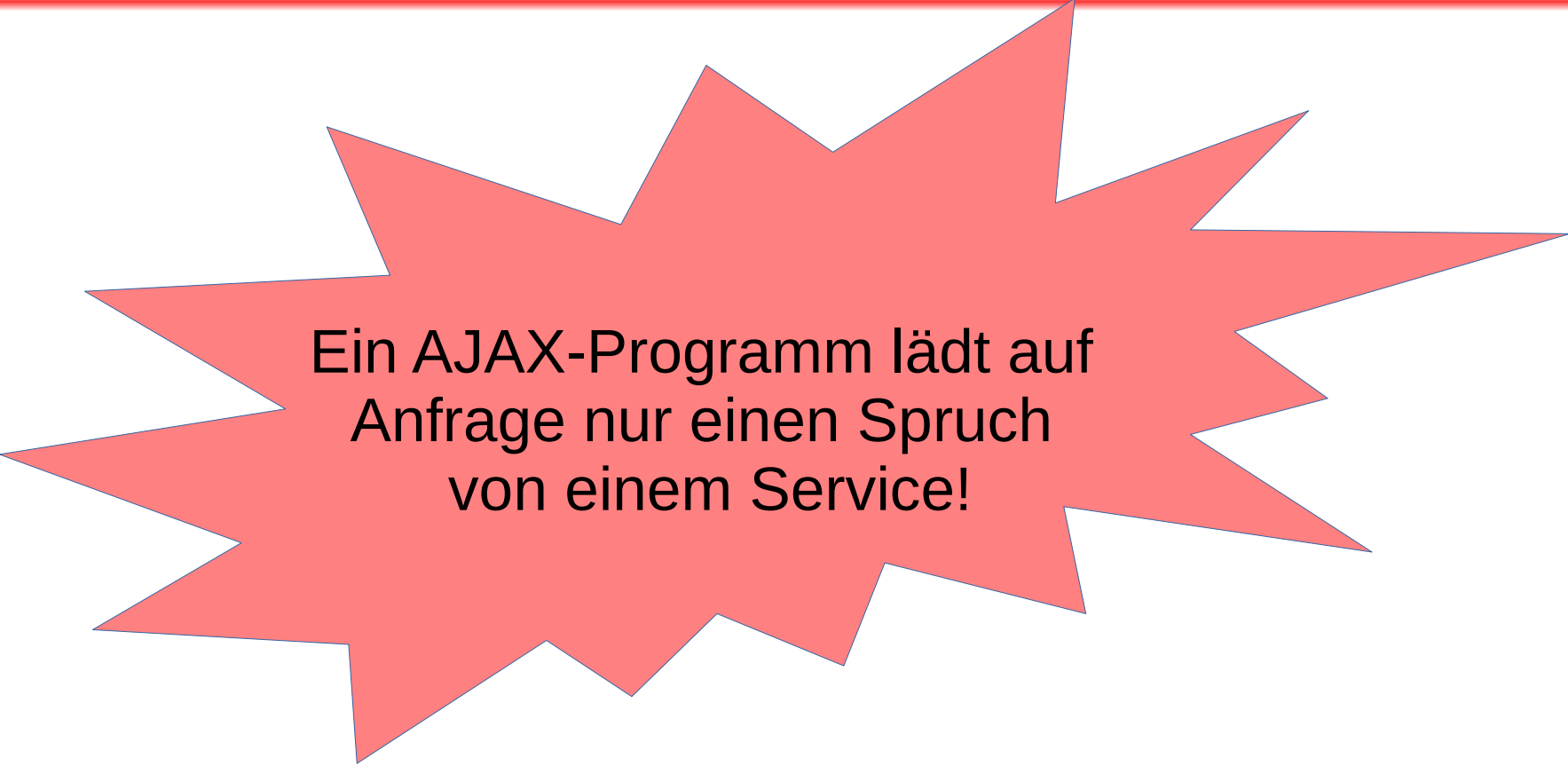
- ★ Laden eines Spruchs auf einer Web-Seite, ohne dass sie neu geladen werden muss.



Einführung

- ◆ Lässt sich ohne viel Aufwand mit einem reinen JavaScript-Programm lösen:
 - ★ Array mit allen Sprüchen
 - ★ Auf Knopfdruck wird eine JavaScript-Funktion aufgerufen, die einen neuen Spruch aus dem Array sucht und darstellt.
- ➡ **Nachteil:** Alle Sprüche müssen in der JavaScript-Datei übertragen werden, was u.U. lange Ladezeiten mit sich bringt!

Einführung



Ein AJAX-Programm lädt auf
Anfrage nur einen Spruch
von einem Service!

Einführung

♦ Vorteile AJAX/Web 2.0:

- ★ Es wird keine neue Seite geladen, sondern die bestehende Seite wird verändert.
- ★ Auslöser für das Ändern oder Neuladen von Seiten sind nicht nur Mausklicks, sondern beliebige Ereignisse.
- ★ Die Kommunikation zwischen Client und Server läuft asynchron ab.
- ★ Die Steuerung wird mittels JavaScript auf dem Client realisiert.
- ★ Die Nachrichten zwischen Client und Server sind im XML-Format.

Einführung

- ◆ Neues Paradigma AJAX/Web 2.0:
 - ★ Eine Web 2.0-Anwendung hat nichts mit dem herkömmlichen Web mit verknüpften Seiten zu tun.
 - ★ Eine Web 2.0-Anwendung läuft verteilt auf dem Client und einem Server. HTML ist die grafische Benutzungsoberfläche auf dem Client.
 - ★ JavaScript stellt die Verbindung zwischen Benutzungsoberfläche und Anwendung dar.
 - ★ Fürs Web 2.0 existieren viele Bibliotheken, die dem Programmierer die Arbeit leichter machen.

Einführung

- ◆ AJAX muss nicht in seiner ursprünglichen Idee benutzt werden.
 - ★ AJAX ohne A: Es kann durchaus viele Gründe geben, um eine Kommunikation synchron zu gestalten.
 - ★ AJAX ohne X: AJAX muss nicht ausschließlich nur mit XML-Nachrichten arbeiten. Es kann z.B. auch Text, HTML oder JSON per AJAX übertragen werden. Je nach Wahl muss der Content-Type-Header gesetzt werden und der Body geparsed werden (`JSON.parse()`).

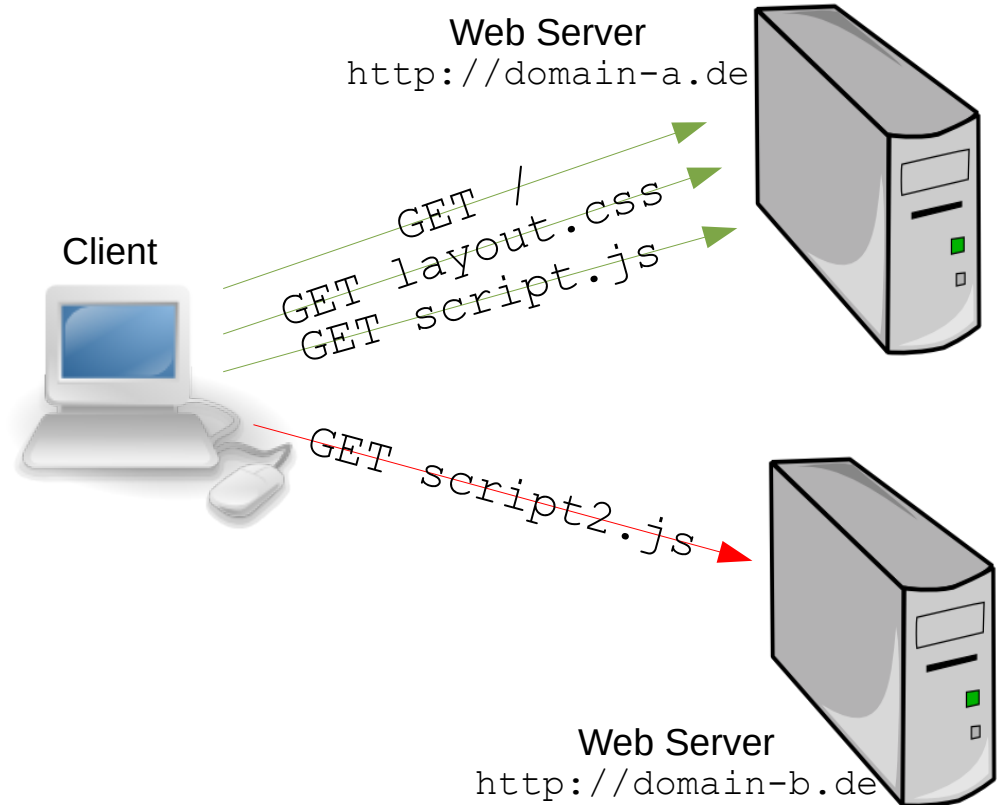
Einführung

Was benötigt AJAX?

- ◆ AJAX ist eine Zusammenstellung von verschiedenen Techniken:
 - ★ Clientseitig JavaScript
 - ★ DOM (Document Object Model)
 - ★ XML, JSON oder ...
 - ★ Server (Node.js, PHP, Python, Java, ...)

Same-Origin-Policy

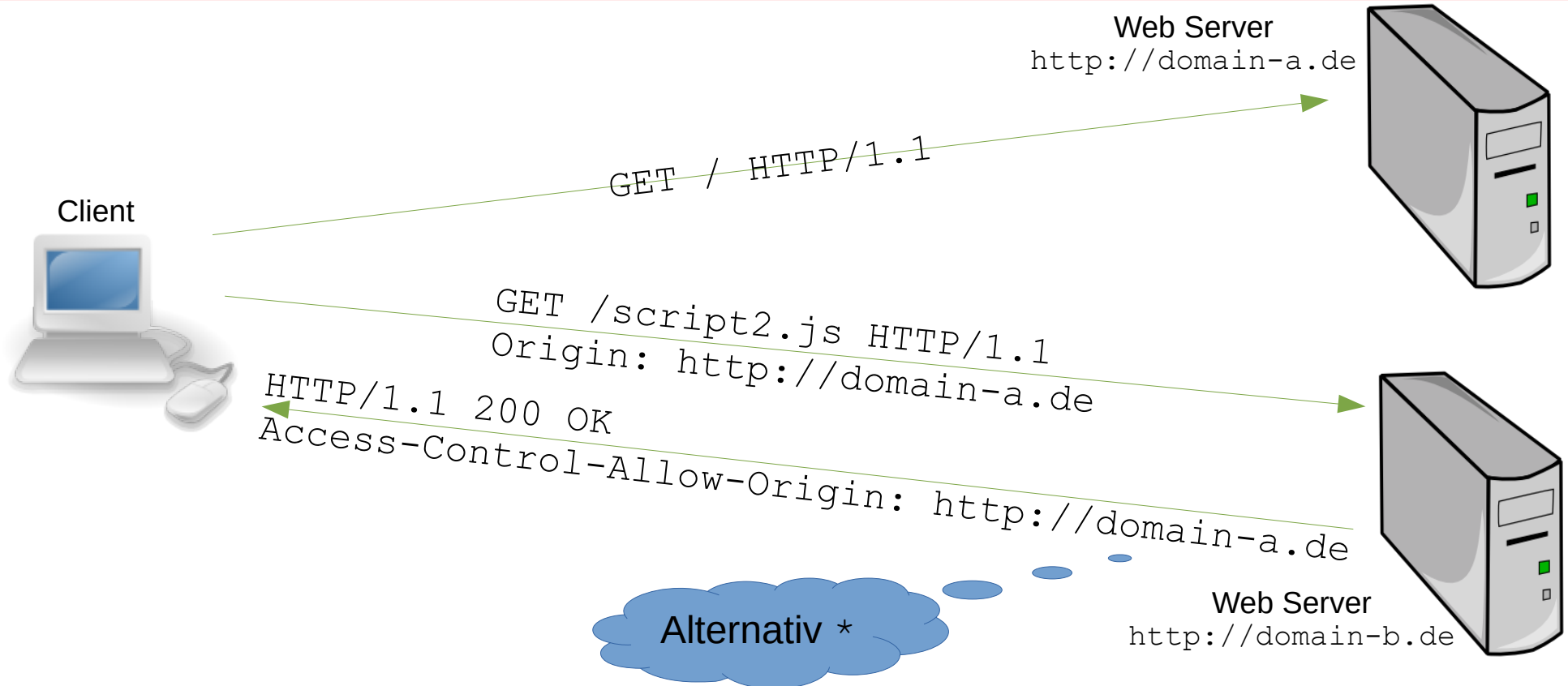
- ◆ 1996 von Netscape eingeführt.
- ◆ Sicherheitsregel, die es untersagt JavaScript Inhalte zu laden, die von einem anderen Web-Server stammen.
- ◆ Herkunft wird anhand von Protokoll, Domain und Port definiert. Nur wenn alle drei Merkmale übereinstimmen, wird das Laden erlaubt.



Same-Origin-Policy

- ◆ Achtung, kann durch spezielle Angriffsmethoden (z.B. DNS Rebinding) ausgehebelt werden.
- ◆ Insbesondere in AJAX/Web 2.0 Anwendungen möchte man die Same-Origin-Policy ausschalten.
- ◆ Cross-Origin Resource Sharing
 - ★ Der Client setzt seine Quelle: `Origin`:
 - ★ Der angefragte Server muss bei seiner Antwort den Zugriff durch entsprechende HTTP-Header erlauben.
 - `Access-Control-Allow-Origin`:
 - `Access-Control-Allow-Methods`:

Same-Origin-Policy



XMLHttpRequest Objekt

- ◆ Das native AJAX besteht nur aus einem einzigen Objekt, das mit JavaScript angesprochen wird.
- ◆ Das XMLHttpRequest-Objekt erlaubt:
 - ★ HTTP/HTTPS Anfrage
 - ★ Abfrage der Antwort
 - ★ Abbrechen der Anfrage

XMLHttpRequest Objekt

1. Request-Objekt ausprägen

- ★ Für alle modernen Browser außer

`new XMLHttpRequest ()`

- ★ Beispiel:

```
let request = new XMLHttpRequest ( ) ;
```

XMLHttpRequest Objekt

2. Request initialisieren

Die [] gehören nicht zur Syntax!

```
★ void XMLHttpRequest::open(String method,  
                             String url, boolean async [,  
                             String username, String password])
```

- `method`: Zu benutzende Methode (GET, POST, ...)
- `url`: anzufragende URL (Same-Origin-Policy)
- `async`: asynchron oder synchron
- `username, password`: optional, falls Authentifizierung notwendig

★ Beispiel:

```
request.open("get", "fortune_x.php", true);
```

XMLHttpRequest Objekt

3. Für asynchrone Kommunikation eine Event-Funktion dem Event-Handler `onreadystatechange` zuweisen.

Beispiel:

```
request.onreadystatechange = zeigeSpruch;
```

4. Anfrage stellen

```
void XMLHttpRequest::send(Object body)
```

- ★ `body`: Wenn die Methode POST/PUT spezifiziert ist, enthält der `body` eine Zeichenkette bzw. ein XML-Dokument, andernfalls null.

- ★ Zusätzlich muss bei POST/PUT Content-Type gesetzt werden

```
request.setRequestHeader('Content-Type', 'application/xml');
```

Beispiel:

```
request.send(null);
```

XMLHttpRequest Objekt

5. Event-Handler-Funktion implementieren:

a) Die Eigenschaft `readyState` auf erfolgreiche Ausführung abfragen

```
readonly short XMLHttpRequest::readyState
```

Zustand	Name	Beschreibung
0	UNSENT	Dies ist der Anfangszustand. Das <code>XMLHttpRequest</code> Objekt wurde ausgeprägt oder mit <code>abort()</code> zurück gesetzt.
1	OPENED	Die Verbindung ist mit <code>open()</code> geöffnet, aber die Methode <code>send()</code> ist noch nicht aufgerufen worden.
2	HEADERS RECEIVED	Die Methode <code>send()</code> wurde aufgerufen und die HTTP(S) Anfrage wurde gesendet. Eine Antwort ist aber noch nicht eingetroffen.
3	LOADING	Die HTTP(S) Antwort wird gerade empfangen. Der <code>body</code> ist aber noch nicht vollständig.
4	DONE	Die HTTP(S) Antwort ist vollständig empfangen.

XMLHttpRequest Objekt

5. Event-Handler-Funktion implementieren:

b) Die Eigenschaft status abfragen

```
readonly short XMLHttpRequest::status
```

Der HTTP-Statuscode des Servers wird zurückgegeben.
Diese Eigenschaft darf nur gelesen werden, wenn
`readyState >= 2` ist.

- 200 OK
- 400 Bad Request
- 404 Not Found
- 500 Internal Server Error

Nur eine kleine
Auswahl!

XMLHttpRequest Objekt

5. Event-Handler-Funktion implementieren:

c) Die Antwort kann mit der Eigenschaft

- `readonly String XMLHttpRequest::responseText` abgefragt werden, solange `readyState >= 3` ist und mit
- `readonly Document XMLHttpRequest::responseXML`, wenn es sich um ein XML-Dokument handelt und `readyState === 4` ist.

XMLHttpRequest Objekt

Beispiel (HTML-Teil):

```
<div id="fortune" >
  <p><strong>Spruch:</strong></p>
  <!-- Spruch wird im div ausgetauscht -->
  <div id="fortuneMessage">&nbsp;</div>
  <!-- Event zum Auslösen der Aktion -->
  <p><input type="button"
        value="Neuer Spruch"
        onclick="ladeSpruch();"
      </p>
</div>
```

XMLHttpRequest Objekt

Aufruf `fortune_x.php` liefert XML zurück:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<spruch>
```

Drei Mitarbeiter eines Softwarehauses sind mit dem Auto unterwegs. Auf einer Gebirgsstraße versagen die Bremsen, der Wagen stürzt einen Abhang runter und landet in einem Bach. Was tun? Der Marketing-Manager: "Wir benennen das Problem, formulieren eine Lösung und nähern uns so der Problemlösung." Der Leiter der Hotline: "Wir rufen einen Techniker, der die Bremse ersetzt." Der Software-Entwickler: "Unsinn, wir schieben den Wagen auf die Straße zurück fahren weiter und schauen erst mal, ob sich der Vorfall wiederholt."

```
</spruch>
```

XMLHttpRequest Objekt

Beispiel (JS-Teil 1):

```
// HTTP-Request Objekt ausprägen
let request = new XMLHttpRequest();

// Event handler
function zeigeSpruch() {
    // Prüfe, dass Kommunikation beendet
    if (request.readyState === 4) {
        // Prüfe Server Status-Code: Erfolgreich
        if (request.status === 200) {
            // XML Antwort abfragen
```

XMLHttpRequest Objekt

Beispiel (JS-Teil 2):

```
// XML Antwort abfragen
let xml = request.responseXML;
let spruch = xml.getElementsByTagName
    ('spruch')[0].textContent;
// Text in HTML setzen
document.getElementById
    ('fortuneMessage').firstChild.
        nodeValue = spruch;
    }
}
}
```

XMLHttpRequest Objekt

Beispiel (JS-Teil 3):

```
// Eventhandler zuweisen
request.onreadystatechange = zeigeSpruch;

// Event-Funktion
function ladeSpruch() {
    // URL für Anfrage setzen
    request.open("get", "fortune_x.php", true);
    // Anfrage ausführen
    request.send(null);
}
```

XMLHttpRequest Objekt

- ◆ Mehrere Anfragen können mittels JavaScript parallel abgesetzt werden.
- ◆ Es ist nicht vorhersehbar, in welcher Reihenfolge diese antworten.
- ◆ Sind Abhängigkeiten im Programm notwendig, so müssen diese per Hand geprüft werden:
 - ★ Setzen von Flags
 - ★ Abfragen von `readyState` einer vorhergehenden Anfrage
 - ★ Verwenden von synchronen Anfragen

XMLHttpRequest Objekt

- ◆ Eventuell ist es notwendig eine frühere Anfrage abubrechen, die noch nicht geantwortet hat:

```
void XMLHttpRequest::abort( )
```

- ◆ Auslösen eines Timeouts, wenn der Server nicht auf eine Anfrage antwortet.

```
request.timeout = 5000; // in ms  
request.ontimeout = function (e) {  
    ...  
}
```


XMLHttpRequest Objekt

- ◆ Sollen Daten per POST übertragen werden, muss ein POST-Body bei Methode `request.send(body)` angegeben werden.
- ◆ Zusätzlich muss zuvor der Header für die Anfrage mit dem passenden Content-Type gesetzt werden:

```
request.setRequestHeader(  
    'Content-Type',  
    'application/x-www-form-urlencoded');
```

Fetch API

- ◆ Ab ECMAScript 2015 (ES6)
- ◆ Leistungsfähiger und flexibler als XMLHttpRequest
- ◆ Nutzt Request und Response Objekte
- ◆ Gibt einen Promise zurück um an das Response Objekt zu kommen

Fetch API

◆ `fetch(resource[, options])`

Die `[]` gehören
nicht zur Syntax!

- ★ Globale Funktion, gibt `Promise` zurück
- ★ reject aufgrund von URL- oder Netzwerkfehler
- ★ Status 404 u.a. sind kein Grund für reject
- ★ Options-Objekt
 - `method` – Anfragemethode
 - `body` – Daten bei POST/OUT
 - `headers` – Header-Object

```
const myHeaders = new Headers();  
myHeaders.append("Content-Type", "application/json");
```

Fetch API

◆ Response-Objekt

- ★ `Response.ok` – `true` wenn Status zwischen 200 und 299
- ★ `Response.headers` – Header Object
- ★ `Response.json()` – JSON encoded data
- ★ `Response.text()` – text data
- ★ `Response.blob()` – blob data

Fetch API

◆ Beispiel (JS-Teil 1):

```
// Event handler
function ladeSpruch () {
    // do a request
    let p1 = fetch ('fortune.php');
    // there was a response
    let p2 = p1.then(function (response) {
        if (response.ok) {
            return response.text();
        } else {
            throw new Error('Status not 2xx');
        }
    });
}
```

Fetch API

◆ Beispiel (JS-Teil 2):

```
// there was no response, e.g. wrong domainname
p1.catch(function (error) {
    console.log(error);
});
// process data
let p3 = p2.then(function (text) {
    document.getElementById ('fortuneMessage')
        .firstChild.nodeValue = text;
});
// error procession data
p3.catch(function (error) {
    console.log(error);
});
}
```

Fetch API

◆ Beispiel `async/await`

```
◆ async function ladeSpruch () {  
    // do a request  
    let response = await fetch ('fortune.php');  
    // decode the response  
    let text = await response.text();  
    // use the response  
    document.getElementById ('fortuneMessage')  
        .firstChild.nodeValue = text;  
}
```

Literatur

- ◆ mdn: Fetch API,
https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API,
21.05.2024
- ◆ Flanagan, D.: JavaScript - The Definitive Guide,
Auflage 5, O'Reilly, 2006
- ◆ Mintert, Stefan und Christoph Leisegang: Ajax –
Grundlagen, Frameworks und Praxislösungen, dpunkt
Verlag, 2007