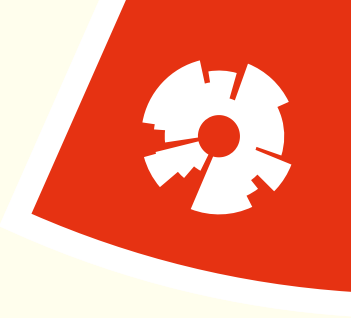
T-213-VEFF: Web Programming I

L7/L8: JavaScript

Grischa Liebel

This Lecture

- Terminology: Client-/Server-side
- The DOM
- JavaScript
  - History/Versions
  - Principles
  - Basic Syntax
  - Hoisting



This Lecture

- JavaScript
  - Callbacks
  - JavaScript Execution (Event Loop)
  - AJAX

Learning Outcomes

- define and contrast client-side and server-side web applications
- define and explain key language concepts of **HTML, CSS, and JavaScript**
- predict the behaviour and look of a web application based on its source code
- predict the behaviour of asynchronous JavaScript code
- develop basic client-side web applications using **HTML, CSS, and JavaScript**

Learning Outcomes

- make use of AJAX to enrich web applications with asynchronous behaviour
- analyse web application source code for errors
- propose improvements to web application source code
- improve existing web application source code

Literature

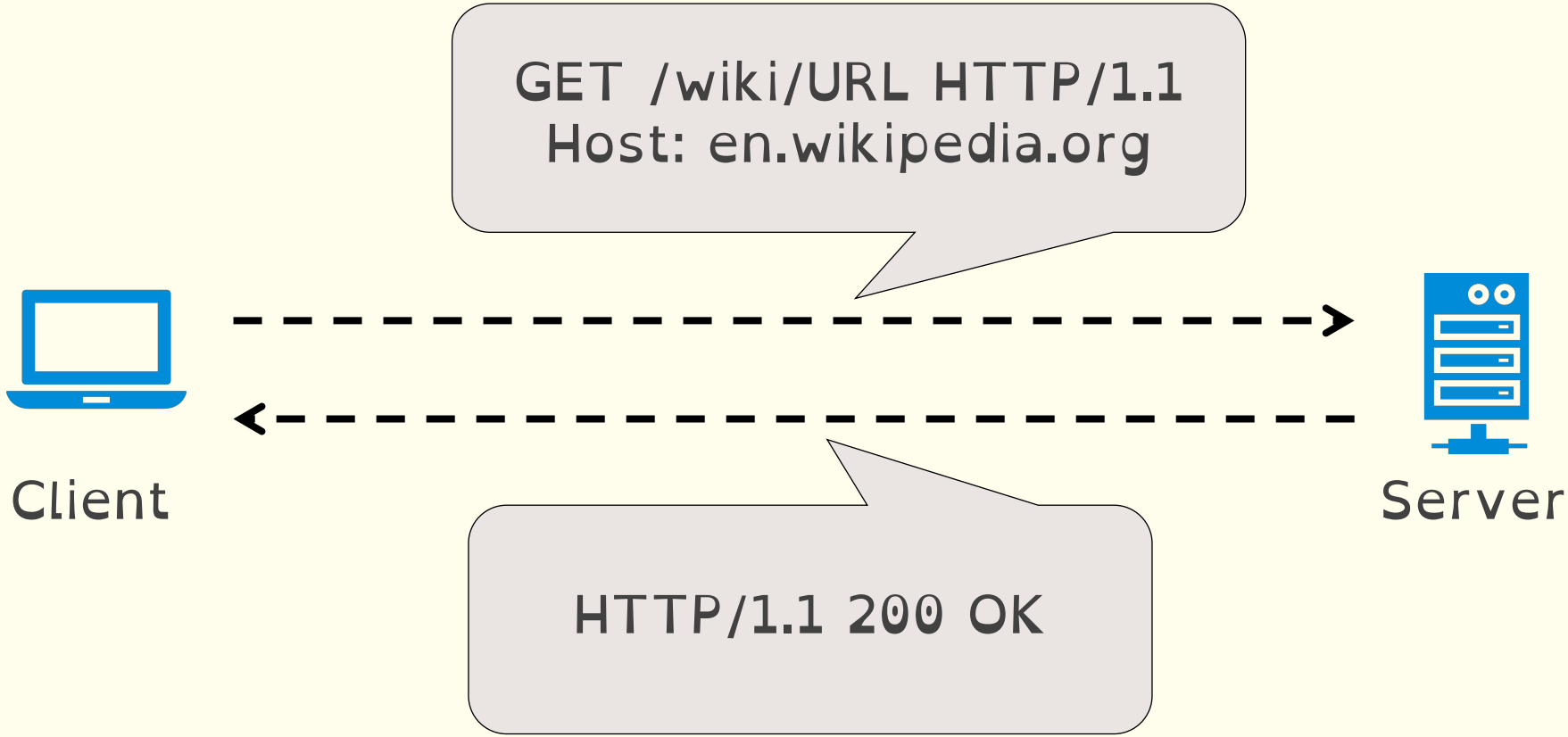
- [1] https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/introduction.html
- [2] https://www.w3schools.com/js/
- [3] https://www.w3schools.com/js/js_versions.asp
- [4] https://www.w3schools.com/tags/ref_eventattributes.asp
- [5] https://medium.com/front-end-hacking/javascript-event-loop-explained-4cd26af121d4
- [6] http://latentflip.com/loupe

Web App vs. Web Site?

> "The general distinction between a dynamic web page of any kind and a 'web application' is unclear. Web sites most likely to be referred to as 'web applications' are those which have similar functionality to a desktop software application [..]" - Wikipedia

- We use: "A web application is a dynamic web site with a functionality similar to desktop software"

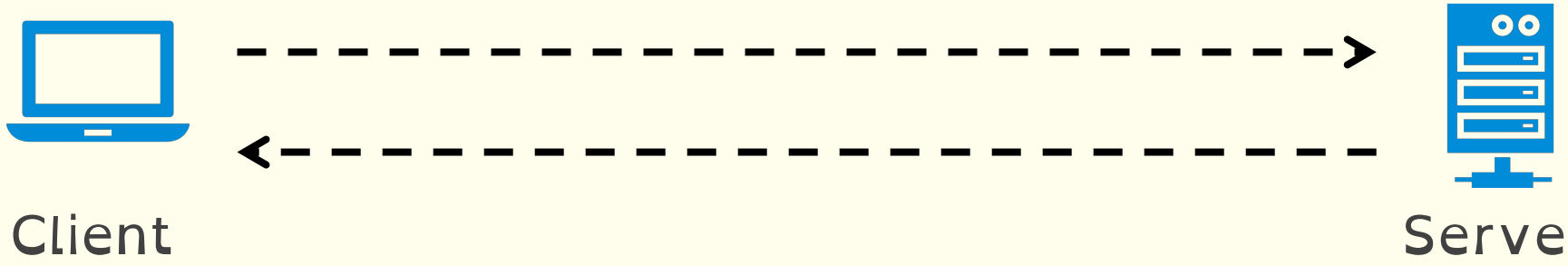
Client/Server side, Frontend/backend



"Frontend"
"Client side"

"Backend"
"Server side"

Client/Server side, Frontend/backend



- Logic executed on client (browser)
- After server serves page
- Mainly JavaScript

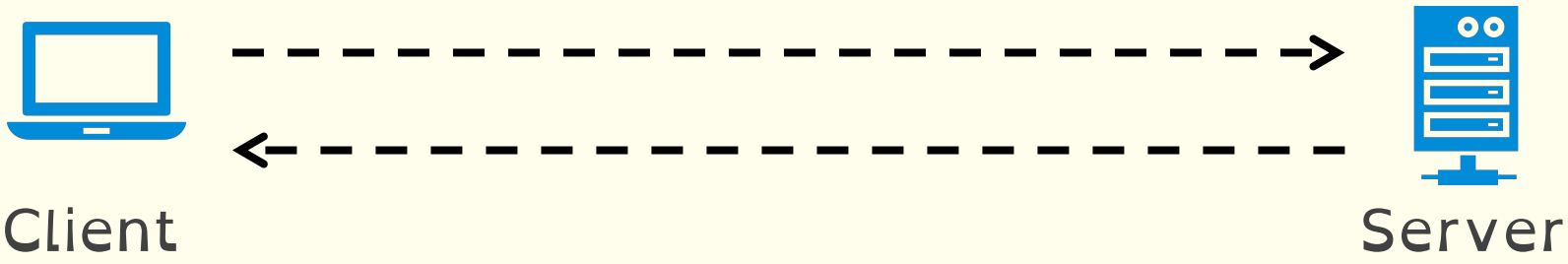
- Code executed on server
- Before server serves page
- E.g.: serving different HTML code depending on parameters
- PHP, Java, JavaScript, ...

"Frontend"
"Client side"

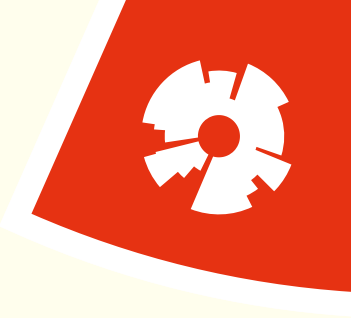
"Backend"
"Server side"

Client- vs Server-Side Web Applications

- A client-side (web) application would use client-side languages
- A server-side (web) application would use server-side languages
- You can of course use both at the same time
  - Therefore: Typically, you talk about where your code is **executed**, not what kind of application you have
  - Server-side/Client-side **execution**



Server-side Execution



- Code is private (on the server)
- Can enforce specific versions of the scripting language
  - E.g., we see later that JavaScript has many versions
- Can use non-standard language elements
- Can access server-side resources
  - E.g., database

Client-side Execution

- Code is public (client sends a GET request to receive the script)
- Can be changed by the user (similar to cookies, CSS, ...)
- Has to be compatible with different browsers/language versions
- Lowers the execution load on the server
  - Clients execute all the logic
- More network traffic
  - Scripts have to be sent to the client via HTTP(S)

Recap: Browser loads HTML

- Browsers interpret HTML
- Source can be viewed

More detailed:
Browsers parse the HTML code and create the **DOM**

The Document Object Model (DOM)

"The Document Object Model (DOM) is an application programming interface (API) for valid HTML [..] documents" - W3C
"With the DOM, programmers can build documents, navigate their structure, and add, modify, or delete elements and content." - W3C

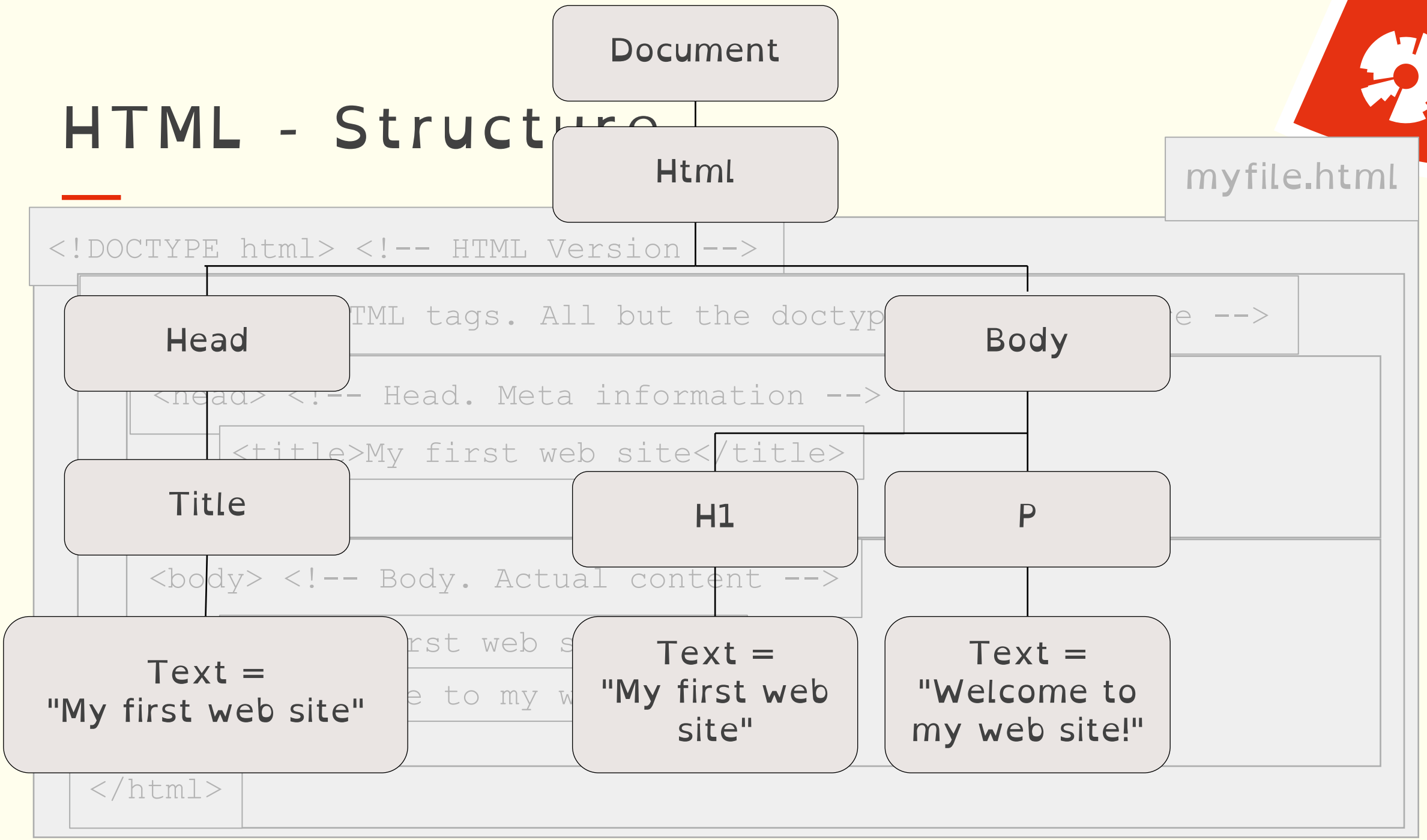
- DOM = Object tree of HTML tags + API to manipulate it
  - The way for scripting languages to access the web site (e.g., read fields, change elements, change CSS, insert/remove things)

HTML - Structure

myfile.html

```
<!DOCTYPE html> <!-- HTML Version -->
<html> <!-- HTML tags. All but the doctype should be here -->
<head> <!-- Head. Meta information -->
    <title>My first web site</title>
</head>
<body> <!-- Body. Actual content -->
    <h1>My first web site</h1>
    <p>Welcome to my web site!</p>
</body>
</html>
```

HTML - Structure



JavaScript

- “part of the triad of technologies that all Web developers must learn: HTML, CSS, and JavaScript” [Flanagan '11]
- Weakly typed, interpreted language
- Supports object-oriented, functional, event-driven styles
- Traditionally: client-side language, no I/O

D. Flanagan (2011), “JavaScript: The Definite Guide”, 6th ed., O’Reilly

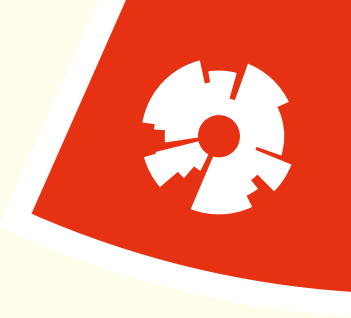
JavaScript

- “part of the triad of technologies that all Web developers must learn: HTML, CSS, and JavaScript” [Flanagan '11]
- Weakly typed, interpreted language
- Supports object-oriented, functional, event-driven styles
- Traditionally: client-side language, no I/O

The focus for now!

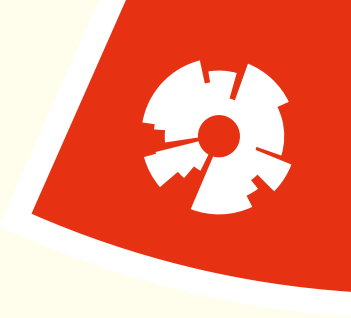
D. Flanagan (2011), “JavaScript: The Definite Guide”, 6th ed., O’Reilly

JavaScript



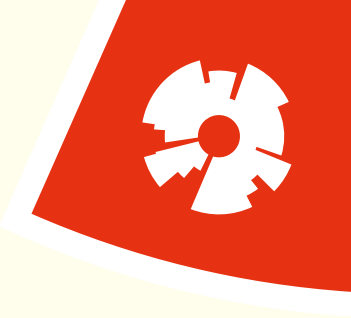
- Original version by Netscape: JavaScript
- Language standard: ECMAScript
- Different standard versions, different browser support
- ECMAScript 3 (ES3) fully supported in all browsers [3]
- ECMAScript 5 (ES5) fully supported in all modern browsers [3]
- ECMAScript 2015 (ES6) limited in older browsers
  - We cover only few concepts of ES6

JavaScript (client-side) Execution



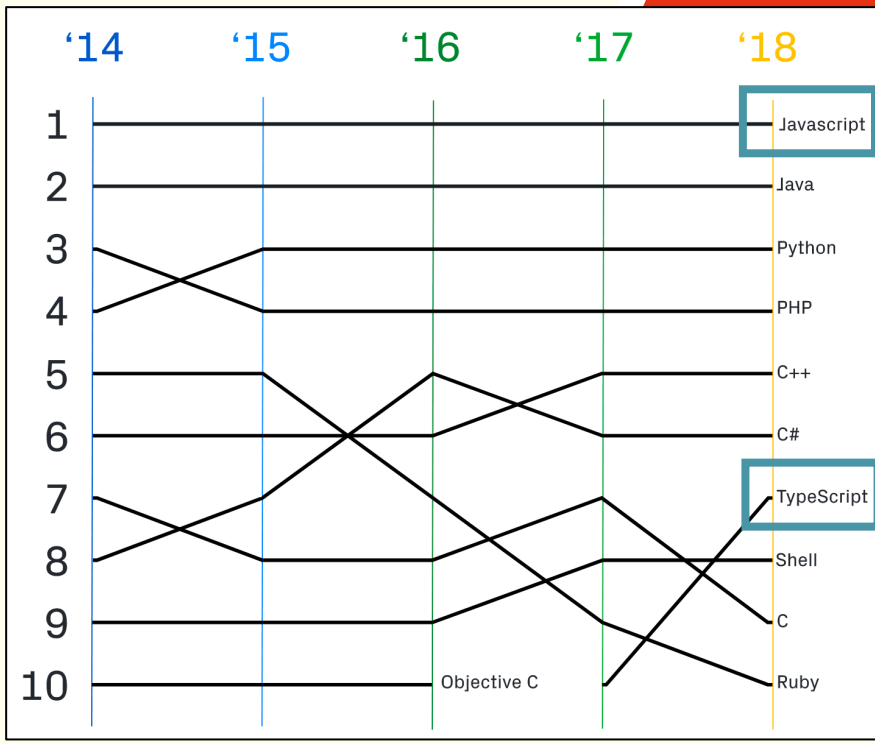
- Each browser has a JavaScript engine that interprets the code
  (Simplified. Typically, they have some internal representation)
- Different engines on the market
  - V8 (Google Chrome, but also for Node.js/backend)
  - SpiderMonkey (Firefox)
  - Chakra (Microsoft IE and Edge)

JavaScript (client-side) Execution



- Differences in speed, language support, different kind of optimisations
  - But "magically", they all work similarly well
  - Nowadays, for regular use, you notice very few differences!

Why JavaScript?



- Earlier in this course: C#
- We use JavaScript only, since
  - It's popular (https://octoverse.github.com/projects)
  - Both backend and frontend
  - Less context switch, tooling, knowledge
  - It's supported by all (common) browsers

Why JavaScript?

- JavaScript has a bad reputation
  - Mainly historical
  - But: Does lots of unintuitive things (https://www.destroyallsoftware.com/talks/wat)

JS - Example

- Let's start with a simple example!

```
<button type="button"
onclick="window.alert('Test')">
Click
</button>
```

React to an HTML event

JS - Example

- Internal definition
- Script tags!

```
<script> window.alert("Test"); </script>
```

- In head or body
- Executed once parsed!

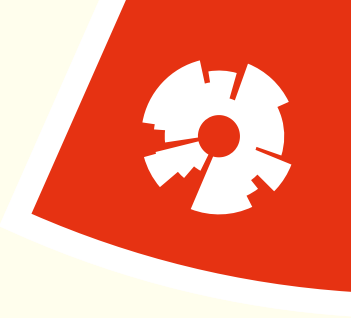
JS - Example

- External definition
- .js file
- Script tags!

```
<script src="script.js"></script>
```

- In head or body
- Executed once parsed!

JS - Specification



- Three different possibilities:
  - As part of HTML attribute (event)
  - Internal definition
  - External definition
- (Multiple definitions/files possible and common)
- Similar to CSS: For modularisation and readability, best to have external files
- For internal/external: Where should we place the script tags?

Where to put the Script tag?

- Can be in head and in body
- Is loaded once the browser parses it (Top-down)
- DOM construction stops until script is loaded
  - If script is large, website "freezes"
- Old recommendation: Put it in the bottom of body
  - Entire DOM is constructed before the scripts are loaded
- New: Put in head, and use **async** or **defer** attributes
- For this course: Not so relevant

Some Basic JavaScript

- Logging to the browser console (e.g., F12 in Firefox)

```
console.log("Hello World");
```

- Basic functions

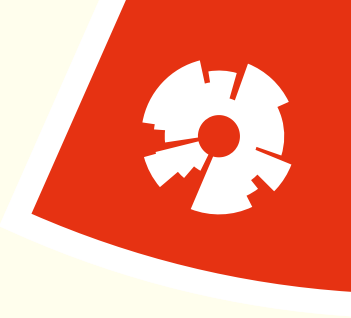
```
//Function definition
function showAlert (msg) {
    window.alert(msg);
}
...
//Function call
showAlert("Hello");
```

Curly braces define a block

Indentation/Whitespace is optional

Semicolons are also optional
(but strongly recommended)

Some Basic JavaScript - Variables



- `var x = 5;`
- `var y = 6;`
- `var z = x + y;`
- `var pi = 3.14;`
- `var person = "Grischa";`
- `var boolean = true;`

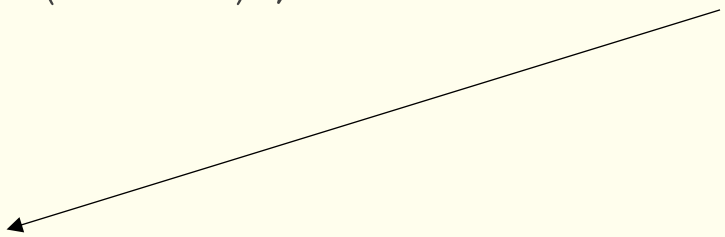
- **Untyped!**
  - Not quite - more later

Accessing the DOM

- One of the main uses of JavaScript in the Browser: read/modify the DOM
- Access existing elements:

```
document.getElementById(id);
document.getElementsByTagName(name);
document.getElementsByClassName(name);
document.body;
document.head;
//Get the first child of an HTML element
var child = element.firstChild;
```

Most methods work on any HTML element



Modifying the DOM

- **Modify existing elements:**

```
//Change text between start/end tag of el. with id header1
document.getElementById("header1").textContent = "New";

//Change the src attribute of myPic
document.getElementById("myPic").src = "cat.jpg";

//Change the color property of the style attribute to blue
document.getElementById("header1").style.color = "blue";

//Remove the first child element of element
element.removeChild(element.firstChild);
```

Modifying the DOM

- Replace the entire HTML:

```
document.write(text);
```

Modifying the DOM

- Insert new elements
- Complicated!

```
var para = document.createElement("p");
var paraId = document.createAttribute("id");
paraId.value = "newPara";
para.setAttributeNode(paraId);
var paraText = document.createTextNode("new text");


para.appendChild(paraText);
document.getElementById("myDiv").appendChild(para);
```

Create the element itself

Create and add attributes

Create and add text (between start and end tag)

Insert element somewhere in the DOM

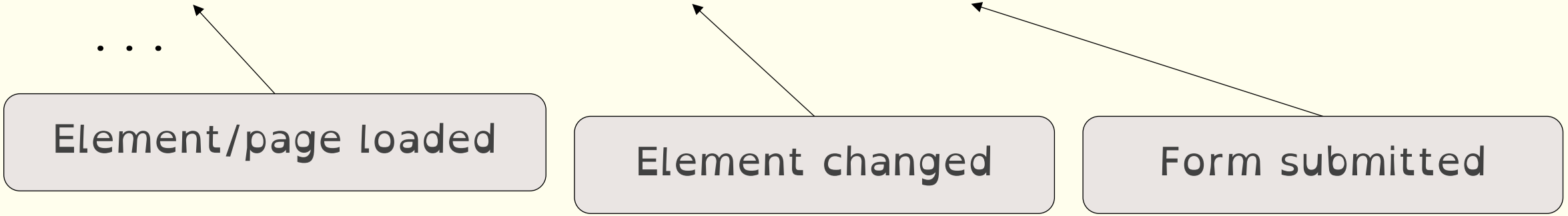
Events

- When should an action happen (e.g., DOM modification)?
- HTML events!

```
<button type="button"
onclick="callFunction()">Go!</button>
```

- There are a lot of them. Check [4] for a reference

```
onload, onclick, onchange, onsubmit, oncontextmenu,
...
```



More on Variables and Types

- `var x = 5; //number`
- `var y = 6; //number`
- `var z = x + y; //number`
- `var pi = 3.14; //number`
- `var person = "Grischa"; //string`
- `var boolean = true; //boolean`

- ~~Untyped!~~ **Weakly typed** - implicit and dynamic types!

More on Variables and Types

- JavaScript knows six data types
- Primitive
  - number
  - string
  - boolean
  - undefined
- Complex
  - object //Arrays are also of type object
  - function
- Can be checked using `typeof(var);`

Functions are variables

- As in Python

```
function functA () {
    ...
}

//Function call
functA();
//Function assignment as a variable
var b = functA;

b(); //calls functA
```

JavaScript Objects

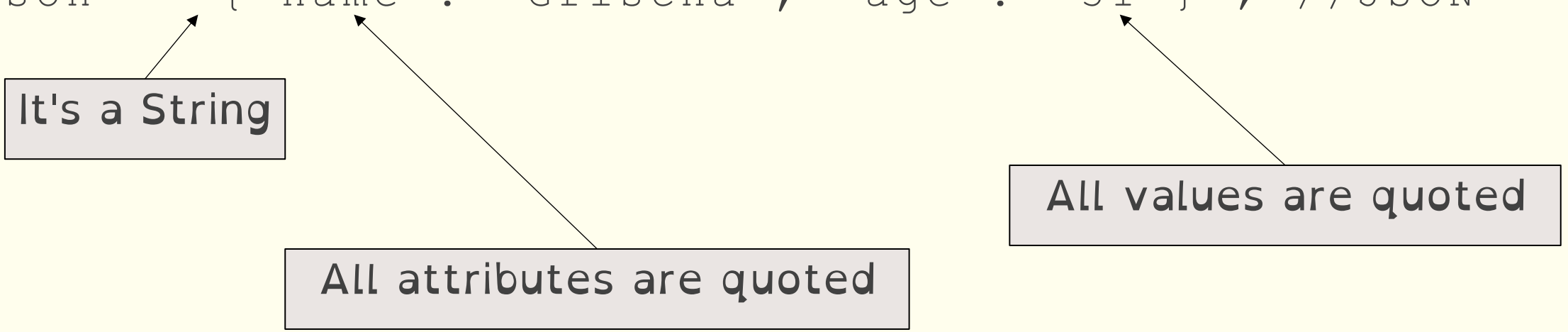
```
var obj = {name:"Grischa", age:31};

//Access via obj.variable or obj["variable"]
console.log(obj.name); //outputs "Grischa"
console.log(obj["name"]); //outputs "Grischa"
```

- No classes! (there are prototypes - not covered in this course)
- Object creation is incredibly simple
  - Origin of JSON, the **JavaScript Object Notation**

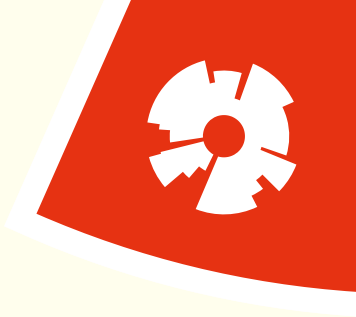
JSON

- JavaScript Object Notation = Common data exchange format
- Essentially the same as a JavaScript object:

```
var obj = {name: "Grischa", age: 31}; //Object

var json = '{"name": "Grischa", "age": "31"}'; //JSON
```



JSON ← → JavaScript object conversion



```
var obj = {name: "Grischa", age: 31}; //Object
var json = JSON.stringify(obj); //JSON


var json = '{"name": "Grischa", "age": "31"}'; //var
obj = JSON.parse(json); //Object
```

- This simplicity is one of the main reasons why JSON is so popular!
- JSON.stringify() removes functions!

JavaScript Arrays

- Square brackets, as in most languages

```
var cars = ["Saab", "Volvo", "BMW"];
```

- Access by index only (no associative arrays)

```
cars[0];
```

- Array in JSON: unquoted

```
var jsonObj = '{"name": "Grischa", "age": "31",
                "languages": ["German", "English"]}';
```

- Of course, you can have objects in arrays

```
var objArray = [{...},{...}];
```

JavaScript - Summary I

- How to include JS in HTML
- How to call/trigger JS code
  - Events
  - When parsed
- DOM access
- Basics: variables, types, functions, objects
- JSON

JavaScript

- What comes on the following 6 slides is the **source of many errors**
  - Make sure you understand these concepts!
- Part of the (bad) reputation of JavaScript
- Type conversion
- Comparators
- Scope
- Hoisting

JavaScript Type Conversion

- `var x = 5; //Number`
- `var y = 6; //Number`
- `var z = 'test'; //String`
- `z = x + y; //Number`
- `var txt = "5"; //String`
- `var m = x + y + txt //?`

`//m = "115"`

Types are dynamic
(can change over time)!

Type conversion from left to right!
(typeof(m) is "string")

JavaScript Comparison

```
var x = 5;
var y = "5";
if (x == y) { //True or false?
...
}
```

True!

- '==' compares only the value (it performs type conversion!)
- Inequality operator: '!='

JavaScript Comparison

```
var x = 5;
var y = "5";
if (x === y) { //True or false?
...
}
```

Convention:
Unless you absolutely need '==', use '==='

False!

- '===' compares both the type and value
- Inequality operator: '!=='

JavaScript Scopes

- Global and local scope, nothing else
  - From ES6: block scope

```
var carName = "Volvo";

function myFunction() {
     var licensePlate = "ABC234";

     x = 5;
}

if (carName === "Volvo") {
     var temp = 42;
}
```

Global

Local to the function

Implicitly global (!)

Global (!)

JavaScript Block Scope

- From ES6: block scope
  - Everything within curly braces is a **block** (also before ES6)
  - ES6 introduces special keywords to declare variables with block scope

```
if (carName === "Volvo") {
      var temp = 42;
      let temp2 = 42;
      const temp3 = 42; //no re-definition, re-assignment
}
console.log(temp); //Logs 42
console.log(temp2); //Logs undefined
console.log(temp3); //Logs undefined
```

JavaScript Hoisting

```
console.log(x);
x = 5;
console.log(x);
if (true) {
    var x;
}
x = 7;
console.log(x);
```

undefined

5

7

- No errors!
- JavaScript "hoists" (moves) variable declarations to the top of the scope!

JavaScript Strict Mode

```
"use strict"; //on top of your script/function
```

- Forces a number of conventions
  - Variables have to be declared (hoisting still works)
  - Deleting variables is not allowed
  - Prevents the use of keywords as variable names
- From version ES5
- Older browsers/JavaScript engines ignore the statement

JavaScript Callbacks

Extremely common in JavaScript

- A pattern/style to execute functionality after work is done
- "You don't call us, we call you!"

```
function doA(callback) {
    //do something
    callback();
});
doA(function() { console.log("done"); } );
```

But....why?

- Callback function is provided as a function parameter
- Callback function is called when `doA()` is finished.
- `doA()` "calls back"

JavaScript Callbacks

`setTimeout` delays the execution of a function (callback) by X ms

```
setTimeout( function(){
    console.log(1);
  }, 500 );
console.log(2);
```

Output:

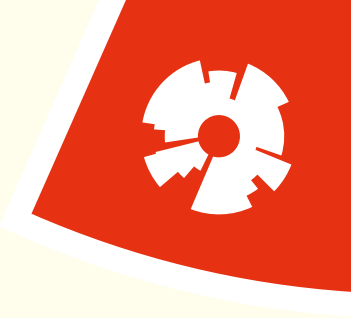
2

1

- `Console.log(1);` is only called after 500ms (roughly)
- Meanwhile, the execution continues

  (i.e., 2 is logged before 1)

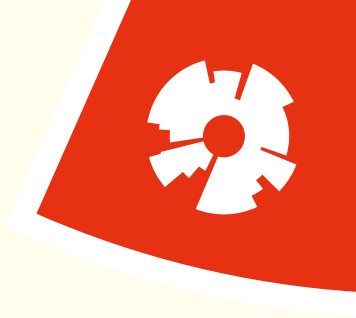
JavaScript Callbacks



- This is actually a great tool!
- In the internet, many things happen delayed
  - We don't know how long an HTTP response takes
- No polling needed (checking whether there is a result)
- No message passing/notification needed
- Non-blocking
  (while waiting for an answer, you can do other things)

JavaScript Callbacks

- But readability is so-so
  - Nested callbacks
  - Difficult to read
  - Prone to errors
- "Callback hell"



JavaScript Callbacks

```
fs.readdir(src, function (err, files) {
  if (err) {
      ...
  } else {
      files.forEach(function (...) {
            console.log(..);
            gm(...).size(function (err, ...) {
              if (err) {
               console.log(...);
            } else {
              console.log(...);
          }
      });
   });
 }
});
```

Still a pretty small example...

JavaScript Callbacks

- Define functions separately
- Give functions clear names
- (Use Promises)

```
function readCallback (err, files) {
  if (err) {
        ...
  } else {
        files.forEach(handleFile(...));
 }
}

function fileCallback (...) {
        console.log(..);
        gm(...).size(doSomething(err,...));
}

function: doSomething(err, ...) {
                if (err) {
              console.log(...);
            } else {
              console.log(...);
          }
}

fs.readdir(src,handleRead(err,files));
```

JavaScript - Asynchronicity

- JavaScript is single-threaded
  - Can only do one thing at a time
- But there is complicated asynchronous behaviour!

JavaScript - Asynchronicity

- (Yet another important source of errors)
- JavaScript does not execute everything in the way you think!

```
function log(){

    setTimeout(function cb(){
        console.log('1'); }
        ,0);

    console.log('2');
}

log();
```

Log "1" with timeout 0ms

Then log "2"

Output:
2
1

JavaScript - Execution

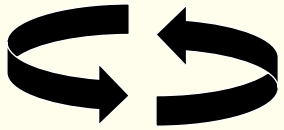
Call Stack (single-thread)

```
console.log(1);
```

```
cb();
```

Browser Web APIs

```
setTimeout(...);
(running timer)
```



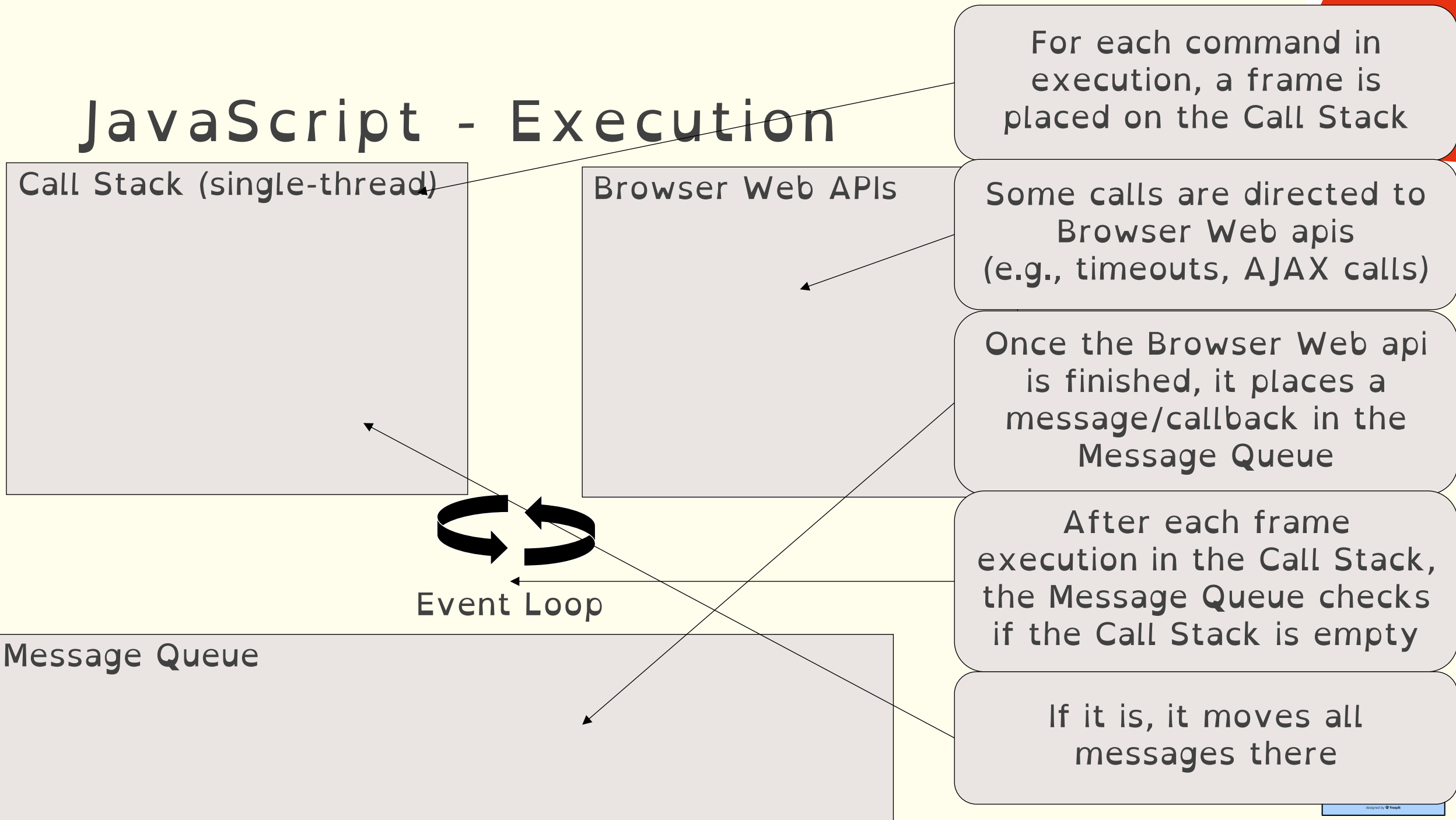
Event Loop

Message Queue

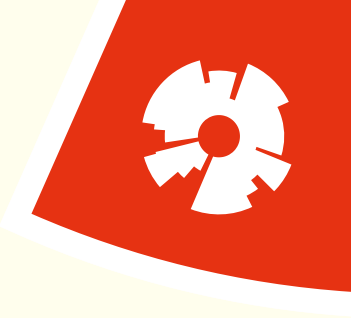
```
cb();
```

```
function log(){
 setTimeout(
  function cb(){
   console.log('1'); }
  ,0);
 console.log('2');
}
log();
```

JavaScript - Execution

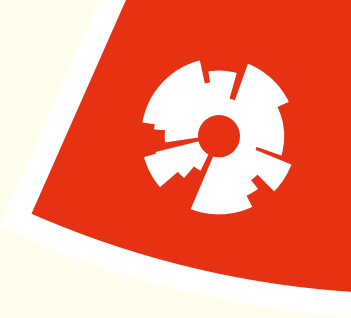


JavaScript - Summary II



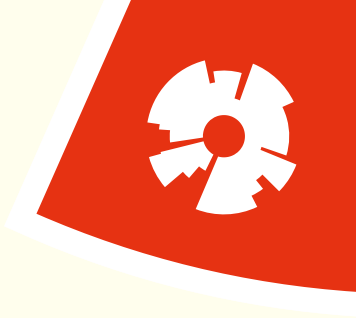
- You have now learned some of the 'quirks' of JavaScript
  - Type conversion
  - Variable comparison
  - Hoisting
  - Scopes
  - Callbacks
  - Asynchronicity
- With this, you are almost ready to understand **most** JavaScript libraries!

JavaScript - Dynamic Behaviour



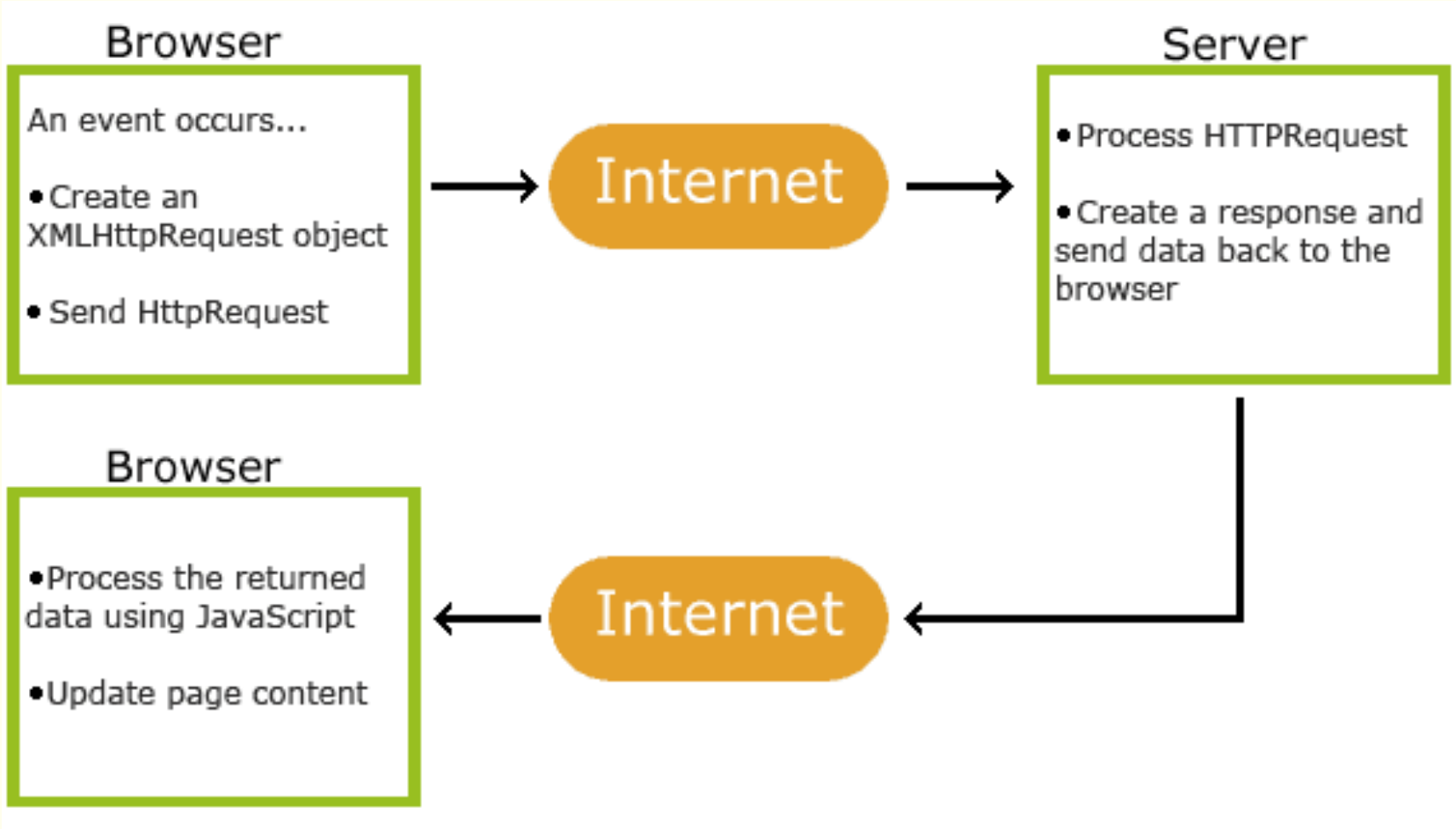
- We can now react to things that happen in the HTML page
  - User events
  - Timeouts
  - Loading of sites
- But: Sometimes we need to get new information after the page is loaded
  - Google suggests search terms while writing
  - But Google doesn't know what you are going to type!
  - Google uses **AJAX**

AJAX



- Asynchronous JavaScript And XML
  (Allows actually not only XML, but also text, JSON, ...)
- Allows HTTP requests/responses **after** the page has been served/loaded
  - Update the page with new data without reload
  - Load only what is needed, more later
  - Load different data depending on user actions/input
- Use the standard `XMLHttpRequest` object to make requests

AJAX


Browser
An event occurs...
• Create an XMLHttpRequest object
• Send HttpRequest
Internet
Server
• Process HTTPRequest
• Create a response and send data back to the browser
Browser
• Process the returned data using JavaScript
• Update page content
Internet

AJAX

```
var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {

    if (this.readyState == 4 && this.status == 200) {

        console.log(this.responseText);

    }

};

xhttp.open("GET", "ajaxTest.txt", true);

xhttp.send();
```

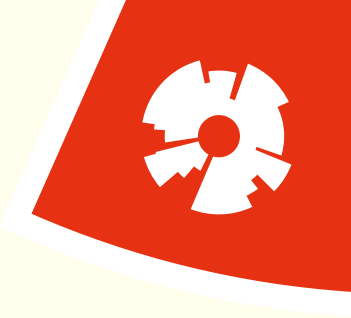
Callback function

HTTP method

URL

Asynchronous or blocking?

AJAX - Axios



- Using `XMLHttpRequest` is cumbersome
- Especially for more complicated requests (with several headers and a body)
- Many libraries simplify HTTP requests
- One example: Axios
  - You are free to use it here in the course
  - Uses some advanced concepts (Promises and arrow functions)
  - No need to understand all details - for simple usage here enough

AJAX - Axios

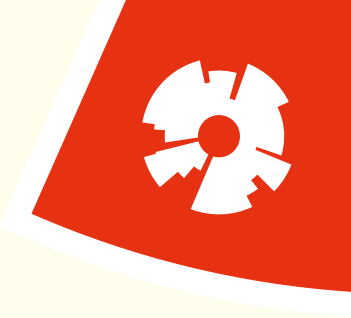
```
//Include the axios library
<script
src="https://unpkg.com/axios/dist/axios.min.js"></script>
...


axios.get('URL')
     .then(response => {
          for (var i=0; i<response.data.length; i++) {
               console.log("Element nr " + i + ": " +
response.data[i]);
          }
     }).catch(error => {
          console.log(error);
     });
```

HTTP method

Promise objects:
.then() = success case
.catch() = error case

Arrow functions: Short form for function definition (with some restrictions)
(here: instead of `function(error) {}`)

What have I not covered?



- (Intuitive things): Loops, string operations, Math, Random, Date, lots of operators
- Explicit type conversion & float accuracy
- Closures
- Arrow functions
- Promises
- Object creation, prototypes
- `this` keyword

What have I not covered?

- Cross-Origin Resource Sharing (CORS)
- Transpiled languages (CoffeScript, TypeScript)
- ...
- Some of these will be added later (testing/debugging, backend, buffer lectures)
- Web Programming II covers some advanced JavaScript

Check [2] if you're interested. Otherwise, be prepared to learn (also for the assignments)!

Summary

- When an HTML document is parsed, the **browser creates the DOM**, an API for accessing and manipulating the content
- JavaScript is originally a **client-side** language
  - **event-based**, reacts to user input
  - can **read/modify the DOM**

Summary

- While JavaScript is **similar** to many programming languages, it is **unintuitive**
  - Type conversion, comparison, hoisting, scope, asynchronicity
- **AJAX** is used to make HTTP requests (asynchronously) after a page has been served
  - Makes web sites/apps truly interactive/dynamic

Next Lecture (L9)

- Testing and Debugging
- A bit of theory
  - Testing levels, types
  - Testing processes: TDD, BDD, ATDD
- Practical stuff
  - Main focus: JavaScript
  - Some basic UI testing: Selenium and Sikuli
  - JavaScript debugging
  - JavaScript (unit) testing with Mocha and Chai


HÁSKÓLINN Í REYKJAVÍK
REYKJAVÍK UNIVERSITY

Sources

Warning: "Designed by Freepik"
AJAX flow: Screenshot from https://www.w3schools.com/js/js_ajax_intro.asp
Todo: Designed by Makyzz / Freepik
Megaphones: Designed by Freepik