Advanced Web Technologies

Web Technologies Basics & Web Media

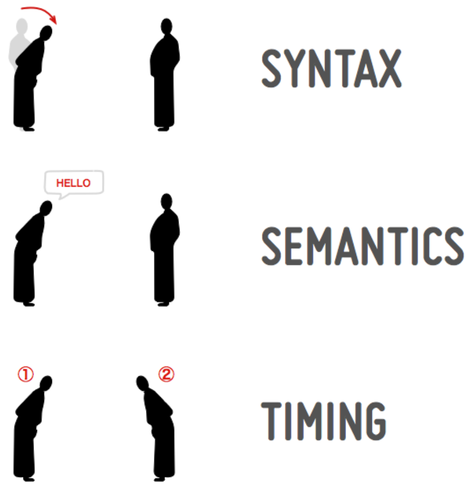
Dr. Louay Bassbouss | Open Distributed Systems | lecture winter term 2023/24

Schedule

No	Week	Date	Topic
1	42	16.10.2023	Introduction and Framework
2	43	23.10.2023	Web Technologies Basics / Media Entertainment for the Web
3	44	30.10.2023	Foundations of Media Streaming
4	45	06.11.2023	Advanced Media Streaming
5	46	13.11.2023	Multiscreen Technologies and Standards
6	47	20.11.2023	HbbTV and Smart TV
7	48	27.11.2023	Media Players - dash.js, Exoplayer
8	49	04.12.2023	Dynamic Advertisement
9	50	11.12.2023	Context-Aware Media Streaming & Encoding
	51	18.12.2023	Holiday break
	52	25.12.2023	Holiday break
	1	01.01.2024	Holiday break
10	2	08.01.2024	Media Delivery in 5G Networks (1)
11	3	15.01.2024	Media Delivery in 5G Networks (2)
12	4	22.01.2024	Metaverse Platforms and Technologies
13	5	29.01.2024	Securing Content-Provenance and Authenticity
14	6	05.02.2024	Interoperable Web-supported Learning Technologies
15	7	12.02.2024	Exercise and Test Preparation
16	8	19.02.2024	Written Test (60min) first slot

Web Technologies Basics

- W3C Intro
- Web Architecture Basics
  - DNS, URL, HTTP
- Web Application Basics
  - HTML, CSS, JavaScript
  - APIs: Web Storage, Workers, DOM
- Web Media APIs
  - HTML5 Video/Audio, MSE, EME, …
- Communication APIs
  - XHR, Fetch, WebSocket, WebRTC



World Wide Web Consortium - W3C

- https://www.w3.org/
- ***The*** Web standardisation organisation
- Mission: Leading the Web to its Full Potential
- Formed ~~and Directed~~ by Web inventor Tim Berners-Lee since 1994

- 30+ years track record of success
- Does not (formally) create binding standards, just "recommendations"…
  - … which are however strong de-facto standards
  - HTML5, CSS, DOM, Web APIs, XML, Web Services / SOAP, SVG, RDF, VoiceXML, ...
- Consortium with ~380 members
- International organization
  - Europe, US, Japan, China, W3C offices (Brazil, …)

World Wide Web Consortium - W3C

Process is (used to be ?) quite slow
Standard example: HTML5

- HTML 4 was standardized in 1997, updated to 4.01 in 200
- Work on HTML5 started in 2004
- Four stages
  - Working Draft
  - Candidate Recommendation
  - Proposed Recommendation
  - W3C Recommendation
- HTML5 Recommendation completed in October 2014

But specifications are quite stable and usable (and used) early on

World Wide Web Consortium - W3C

- Working Groups typically produce deliverables (e.g., standards track technical reports, software, test suites, and reviews of the deliverables of other groups).
- The primary goal of an Interest Group is to bring together people who wish to evaluate potential Web technologies and policies. An Interest Group is a forum for the exchange of ideas.
- There are several W3C Working Groups working on Web API standardization, e.g.,
  - Web Payments WG
  - Web Authentication WG
  - Accessibility Guidelines WG
  - Media and Entertainment IG
  - Cascading Style Sheets (CSS) WG
  - Devices and Sensors WG
  - Web Real-Time Communications WG
  - Web Application Security WG
  - Web of Things WG/IG

Technische Universität Berlin

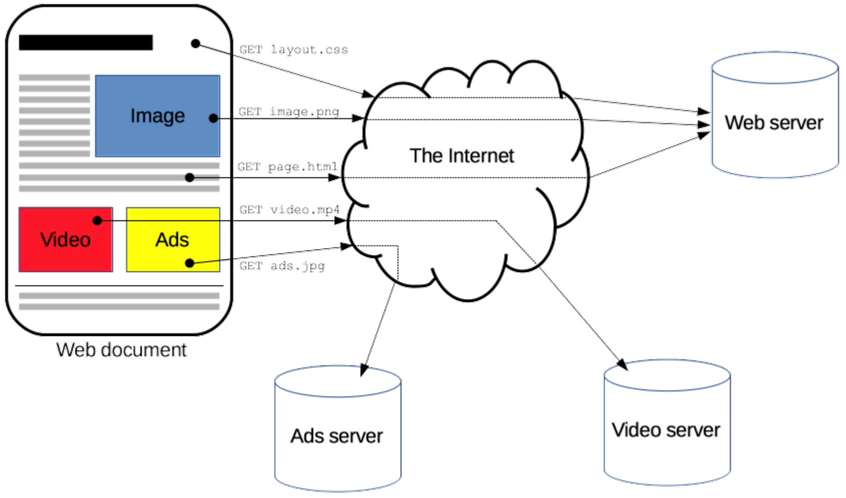
WEB ARCHITECTURE BASICS

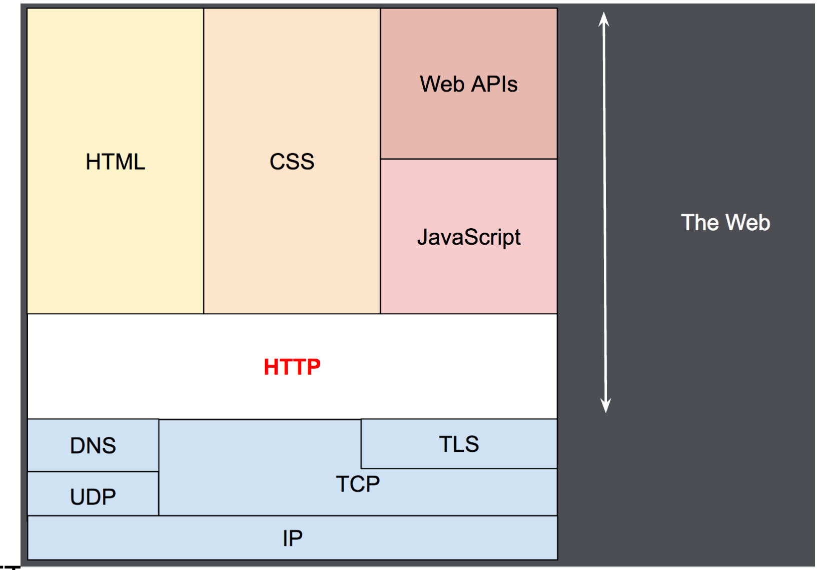
DNS – Domain Name System

- DNS stands for "domain name system"
  - Domain names are the human-readable website addresses we use every day
  - For example, Google's domain name is google.com
- But computers don't understand where "google.com" is.
  - Behind the scenes, the Internet use numerical IP addresses ("Internet protocol" addresses).
  - Example IPv4 address 205.251.242.103
  - Example IPv6 address 2a00:1450:4005:80b::200e
- DNS translates domain names into IP addresses (like a phone book)
- We use google.com instead of 173.194.113.120 because domain names are more meaningful and easier to remember.

Hypertext Transfer Protocol - HTTP

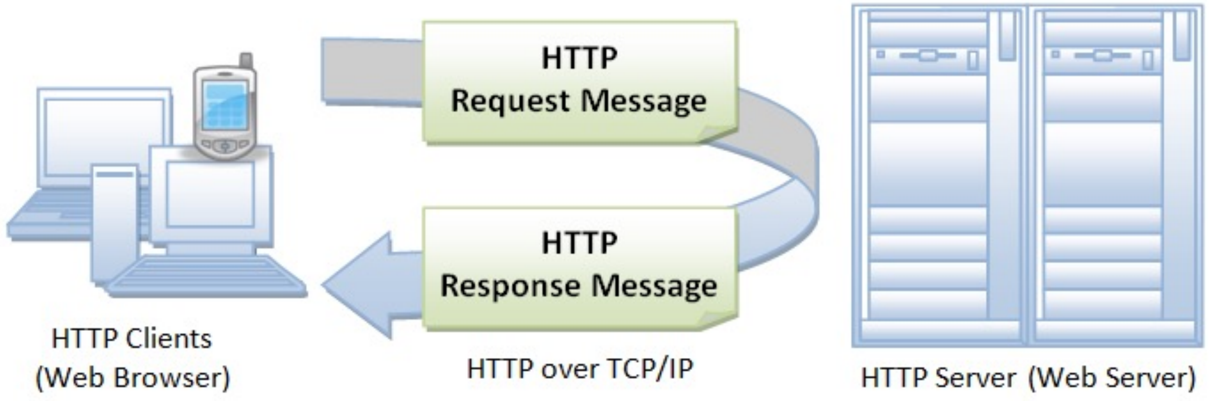
- HTTP is a application protocol based on TCP/IP (not for HTTP/3) which allows to fetch resources, such as HTML documents, images, media, JSON, XML from a Web Server
- It is a Client-server protocol, which means requests are initiated by the Client e.g. Web Browser
- The browser may open multiple HTTP connections to fetch resources of a single Web page (Script, Images, CSS, ...)





More details https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

Hypertext Transfer Protocol - HTTP



HTTP is an ***asymmetric request-response client-server*** protocol

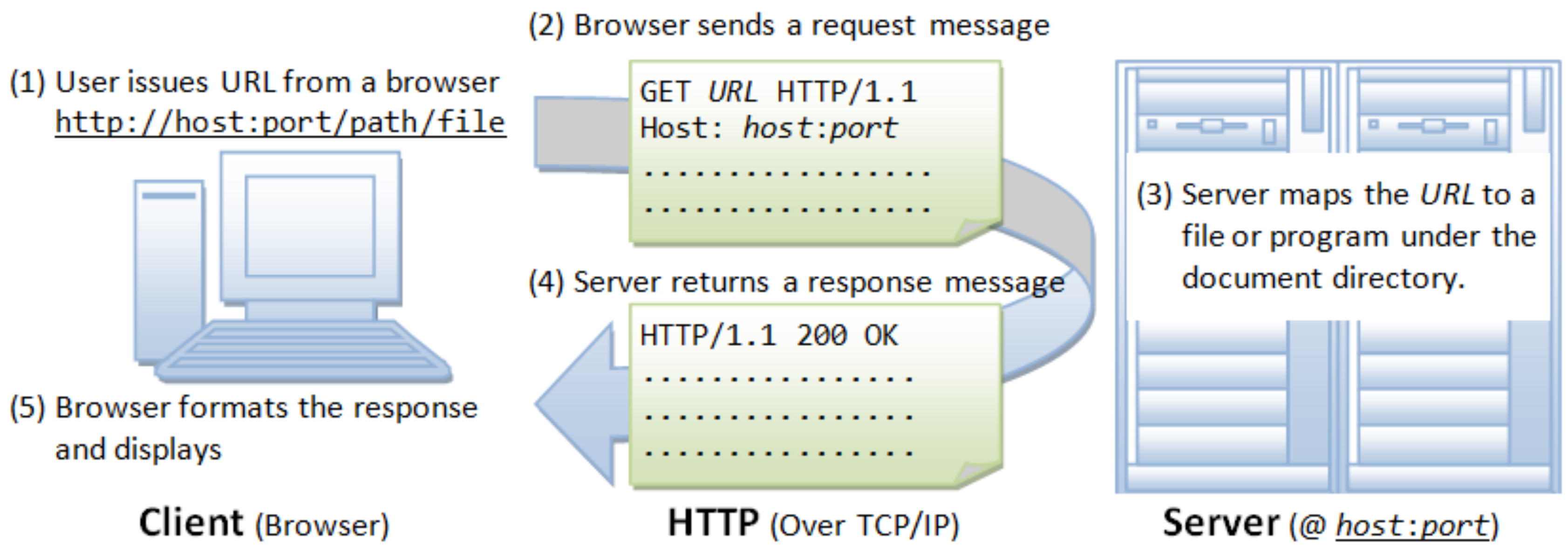
- In other words, HTTP is a *pull protocol*, the client *pulls* information from the server (instead of server *pushes* information down to the client)

HTTP is a **stateless** protocol

- In other words, the current request does not know what has been done in the previous requests

https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

HTTP - Example


(1) User issues URL from a browser
http://host:port/path/file
(2) Browser sends a request message
GET URL HTTP/1.1
Host: host:port
(3) Server maps the URL to a file or program under the document directory.
(4) Server returns a response message
HTTP/1.1 200 OK
(5) Browser formats the response and displays
Client (Browser)
HTTP (Over TCP/IP)
Server (@ host:port)

https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

URL - Uniform Resource Locator

- A URL (Uniform Resource Locator) is used to uniquely identify a resource on the web.
- URL has the following syntax:

```
scheme://hostname:port/path?query#fragment
```

- There are 6 parts in a URL:
  - **Scheme**: The application-level scheme name to identify how to access the specified resource e.g., HTTP, FTP, and telnet.
  - **Hostname**: The DNS domain name (e.g., www.test101.com) or IP address (e.g., 192.128.1.2) of the server.
  - **Port**: The TCP port number that the server is listening for incoming requests from the clients (16bit/2bytes).
  - **Path**: The name and location of the requested resource, under the server document base directory.
  - **Query**: optional additional data as query string preceded by a question mark, typically a key-value pair list with an ampersand (&) as delimiter
  - **Fragment**: optional fragment identifier preceded by a hash to identify a secondary resource within the main resource, e.g., a heading

URL - Example

```
http://www.example.com/docs/index.html?lang=en&session=2fn258v7#News
```

- **Protocol** is http
- **Hostname** is www.example.com.
- The **port** number is not specified in the URL, but default http port is 80
- The **path** and file name for the resource to be located is "/docs/index.html".
- The **query** string contains to web site specific key value pairs
- The **fragment** name is **news**, the browser scrolls the web page to bring a specific html fragment into the visible area
- Other examples:
  - ftp://www.ftp.org/docs/test.txt,
  - mailto:user@test101.com

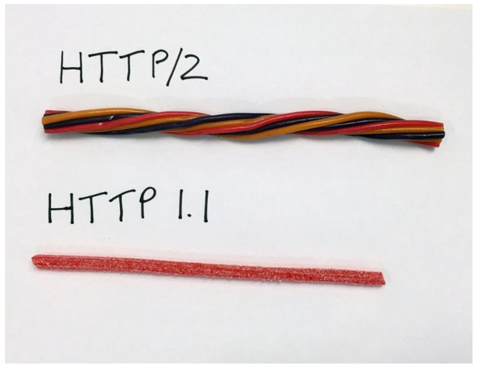
HTTP Protocol

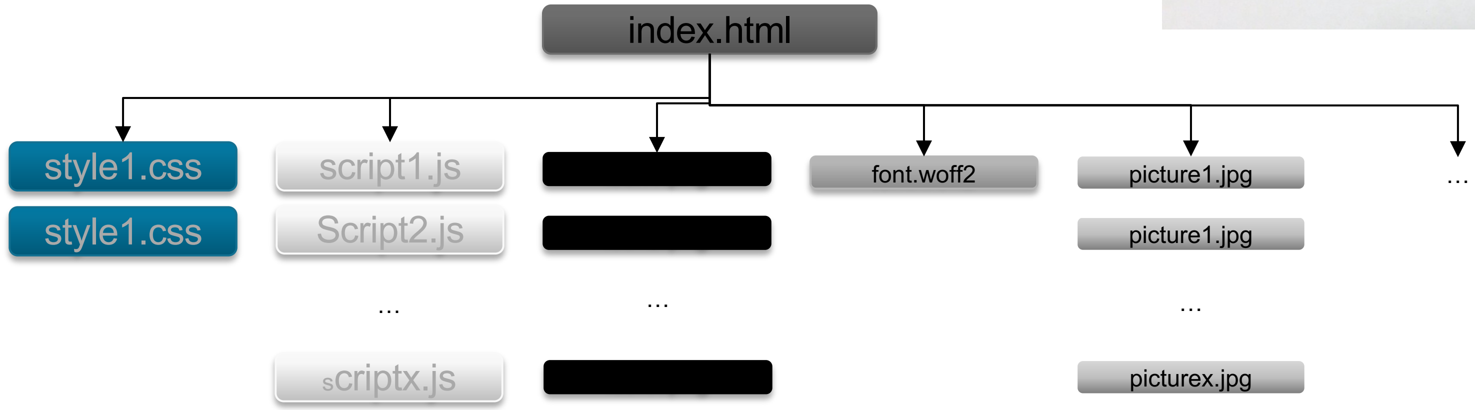
- Whenever you enter a URL so a website in the address box of the browser, the browser translates the URL into a HTTP request message and sends the request to the server.
- For example, the HTTP request message for the URL http://www.example.com/doc/index.html may look like:

```
GET /docs/index.html HTTP/1.1
Host: www.example.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

HTTP/2

- In May 2015 IETF standardized HTTP/2
  - RFC7540 and 7541
- Main goal to reduce transport overhead
  - combine requests (multiplex)
  - better data compression that include header information
  - server-push (PUSH_PROMISE)
  - binary encoding



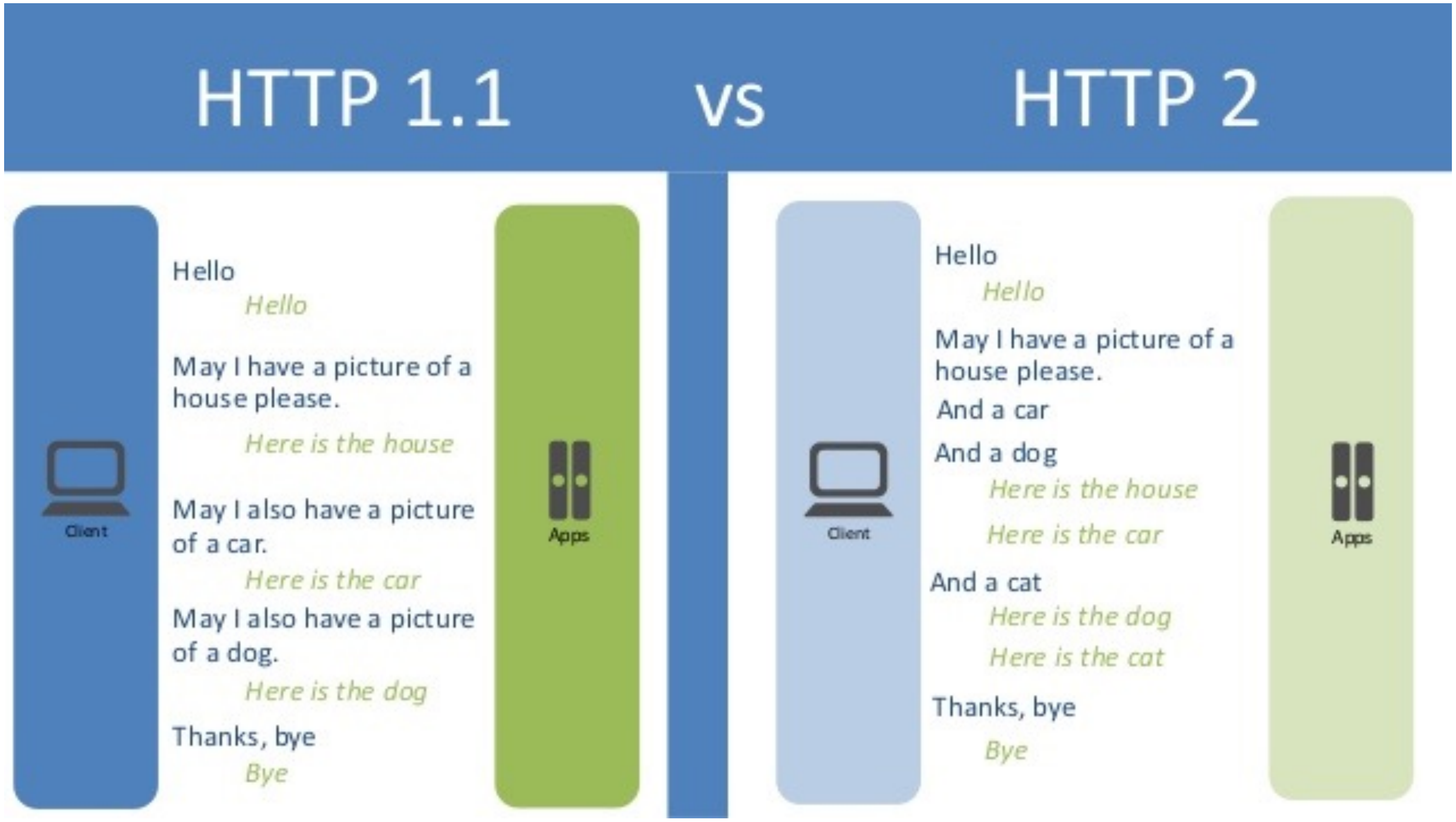


With HTTP/1.1 one HTTP and thus TCP request must be made for each resource

HTTP/2

- Originally TLS was planned as being mandatory for HTTP/2 but it was dropped
  - But anyway Google und Mozilla state that they only support HTTP/2 with transport layer security (TLS with ALPN (Application Layer Protocol Negotiation)
- HTTP/2 is already there
  - Google Chrome (incl. Chrome for iOS and Android) since version 41, Mozilla Firefox since version 36, Internet Explorer 11, Opera since version 28 (Opera Mobile since version 24) and Safari since version 8

HTTP/2



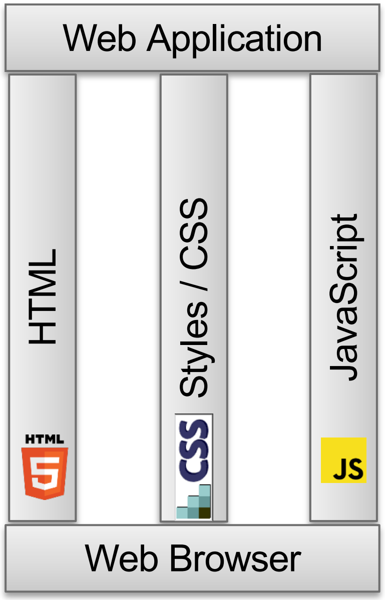
http://www.slideshare.net/lmacvittie/http2-changes-everything

WEB APPLICATION BASICS

Web Application Development Fundamentals

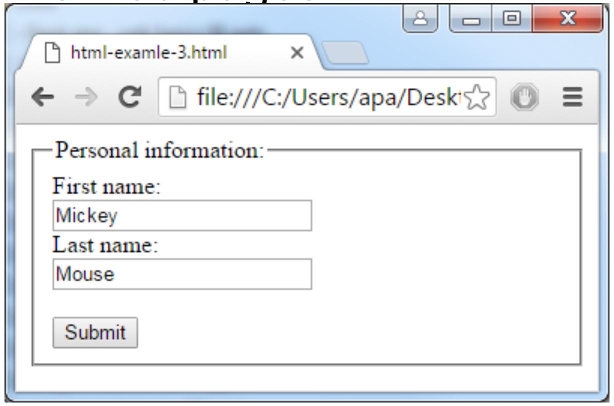
- Three client-side technologies for Web Application development
  - HTML
  - CSS
  - JavaScript

- Only HTML is mandatory for the Web Browser to render a web page.
  - But CSS makes it “nicer”
  - And JavaScript can make the page more interactive and dynamic



HTML – HyperText Markup Language

- The HyperText Markup Language or HTML is the standard markup language for web pages
- Web browsers read HTML files and render them into web pages
- HTML…
  - describes the **structure** of a website
  - provides cues for presentation
  - allows images and objects to be embedded
  - can be used to create interactive forms



Personal information:
First name: Mickey
Last name: Mouse

- HTML **elements** form the building blocks of all websites, they provide means to create structured documents by denoting structural semantics for text such as
  - Headings, Paragraphs, Lists, Links, Forms and its input elements
  - and more…

More details https://developer.mozilla.org/en-US/docs/Learn/HTML

HTML5

- HTML4 was standardized in 1997
  - Mainly only displaying content was possible, web sites where far away from what native applications could do
  - many modern features where only possible via plug-ins like Adobe Flash
- HTML5 development was started in 2006 and finished 2014 and brought new elements:
  - semantic elements like <header>, <footer>, <article>, and <section>
  - input form control attributes like number, date, time, calendar, and range
  - graphic elements <svg> and <canvas>
  - multimedia elements <audio> and <video>
- Some new API's are:
  - HTML Application Cache, HTML Geolocation, HTML Drag and Drop, HTML Local Storage, HTML Web Workers, HTML SSE
- Removed elements like <strike> <center> <font>

CSS - Cascading Style Sheets

- What is CSS?
  - CSS stands for Cascading Style Sheets
  - CSS describes how HTML elements are to be displayed
  - CSS saves a lot of work. It can control the layout of multiple Web pages all at once
  - External Style Sheets are stored in CSS files

- Why Use CSS?
  - CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

More details https://developer.mozilla.org/en-US/docs/Web/CSS

CSS Solved a Big Problem

- HTML was NEVER intended for formatting text. HTML was created to describe content
  - e.g, <h1>This is a heading</h1> as primary headline
  - or <p>This is a paragraph.</p> for paragraphs
- When tags like <font>, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers
  - Development of large web sites, where fonts and color information were added to every single page, became a long and expensive process.
- To solve this problem, the World Wide Web Consortium (W3C) created CSS
  - CSS was created **to specify the document's style, not its content**
- CSS can save a lot of work
  - style definitions can be put into external .css files
  - With an external style sheet, the look of an entire Web site can be changed by changing just one file!

JavaScript

⇩

how web sites are build using HTML
how web sites can be styled using CSS

⇩

make web sites dynamic
make web sites behave like „real“ applications and not just like a nice „book“

JavaScript

- JavaScript is a high-level, dynamic, untyped, and interpreted programming language
- It has been standardized in the **ECMAScript** language specification
  - Latest version is ECMAScript2020
- Alongside HTML and CSS, it is one of the three essential technologies of World Wide Web content production
  - JavaScript is prototype-based with first-class functions, making it a multi-paradigm language, supporting
    - **object-oriented**
    - **Imperative**
    - and **functional** programming styles

More details https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

Modern JavaScript APIs

- Web Storage
- Workers
    - Web Workers
    - Shared Workers
    - Service Workers
- Geolocation
- DOM
- Communication

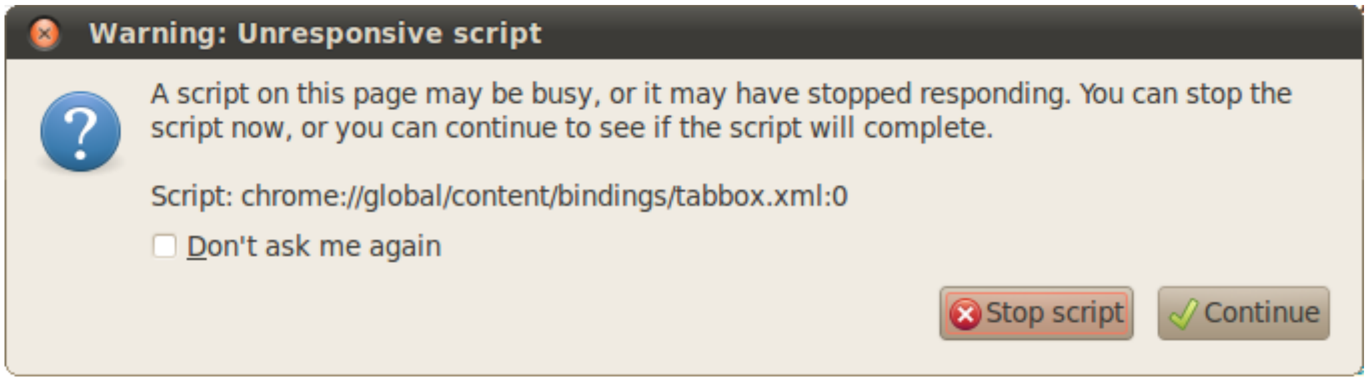
Web Storage APIs

- Store basic key/value data locally within the user's browser
- Before HTML5, application data had to be stored in cookies
  - invented to solve the problem "how to remember information about the user"
  - included in every server request => HTTP is stateless
- without affecting a website’s performance large amounts of data can be stored locally
- Unlike cookies, the storage limit is larger (at least 5MB)
  - information is not transferred to the server
- Web storage is per origin (per domain and protocol)
  - All pages, from one origin, can store and access the same data

More details https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

Web Workers: Why?

- JavaScript is a single-threaded environment
  - multiple scripts cannot run at the same time
  - imagine a site that needs to handle UI events
  - query and process large amounts of API data
  - and manipulate the DOM
- Also Web Apps becoming more and more powerful like real desktop applications


Warning: Unresponsive script

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

Script: chrome://global/content/bindings/tabbox.xml:0

Web Workers

- When executing scripts in an HTML page, the page becomes unresponsive until the script is finished
  - single threaded, event based
- A web worker is a JavaScript that runs in the background
  - independently of other scripts
  - without affecting the performance of the page
    ➔You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background
- A Web Worker script is defined in a separate JavaScript file
- Creating a Web Worker is simple, the code in worker.js is executed directly after creating the Worker

```
new Worker("worker.js");
```

- As “extension” service workers allow scripts to react on events without the web page even being active (e.g., Web Push)

More details https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API

Web Workers

- Web workers are linked to their creator in a 1:1 relationship
- Since web workers are independently from the web applications execution context, communication between the Worker and the main applications is done using messaging
- Web workers do not have access to the following JavaScript objects
  - The window object
  - The document object
  - The parent object

Worker Script

```
self.addEventListener('message', function(e) {
   console.log(‚Main said: ', e.data);
   self.postMessage(e.data);
});
```

Main Script

```
var worker = new Worker(‚worker.js');

worker.addEventListener('message', function(e) {
  console.log('Worker said: ', e.data);
});

worker.postMessage('Hello World');
```

Shared Workers

- Shared web workers allow any number of scripts to communicate with a single worker

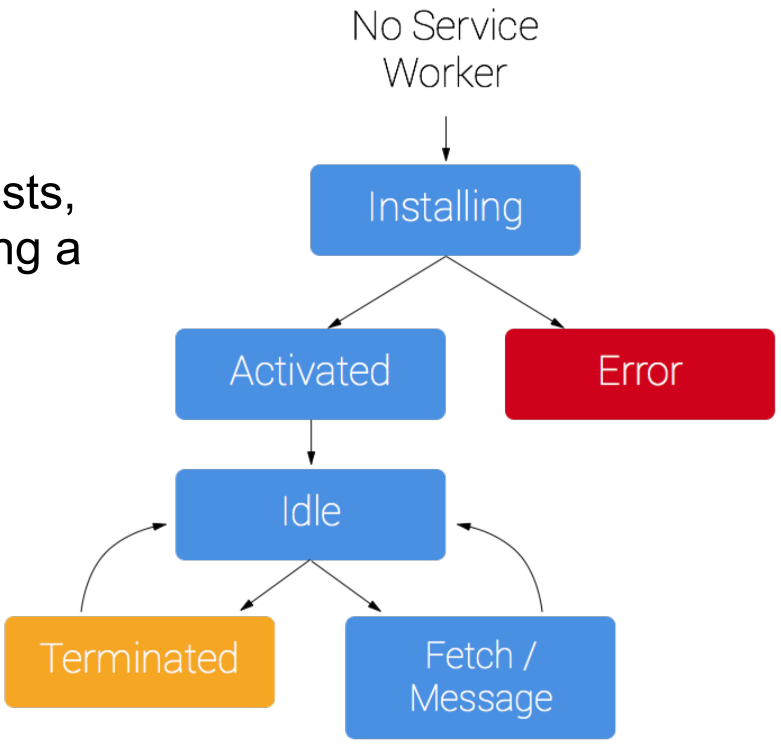
```
new SharedWorker(„sharedworker.js");
```

- Any of your page scripts can communicate with the shared web worker
- Unlike dedicated web workers, you must communicate via a ‘port’ object and attach a message event handler
- In addition, you must call the port’s start() method before using the first postMessage():

More details https://developer.mozilla.org/en-US/docs/Web/API/SharedWorker

Service Workers

- Service worker can be used for
  - push notifications
  - background sync (offline usage)
    - intercept and handle network requests, including programmatically managing a cache of responses
- A service worker has a lifecycle that is completely separate from your web page
  - either the service worker is terminated to save memory
  - or it handles fetch and message events that occur when a network request or message is made from your page
- service worker must be provided using a HTTPS connection



More details https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorker

Dynamic Web: Document Object Model (DOM)

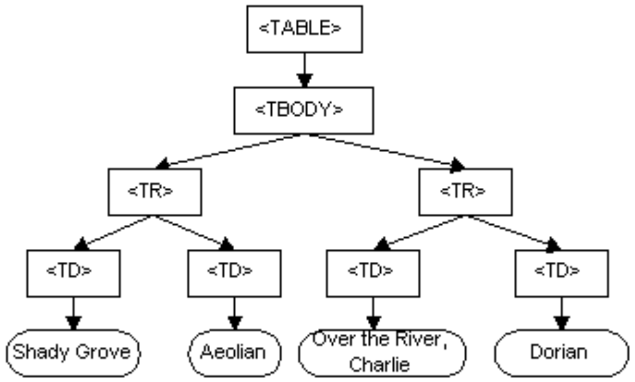
- Allows to view/modify page elements in script code after the page was loaded
  - client side = highly responsive interactions, not needed to request new HTML pages from a server
  - browser-independent
- a representation of the **current** web page (not the HTML source) as a **tree** of JavaScript objects
- The **DOM is a programming API for documents**
  - based on an object structure that closely resembles the structure of the documents
  - allows to **build documents**, **navigate** their structure, and **add**, **modify**, **or delete elements** and content
  - In JavaScript the DOM API is available via the global “document” object

Dynamic web: DOM

```
<TABLE>
   <TBODY>
      <TR>
         <TD>Shady Grove</TD>
         <TD>Aeolian</TD>
      </TR>
      <TR>
         <TD>Over the River, Charlie</TD>
         <TD>Dorian</TD>
       </TR>
   </TBODY>
</TABLE>
```

Shady Grove	Aeolian
Over the River, Charlie	Dorian

Initial DOM Tree

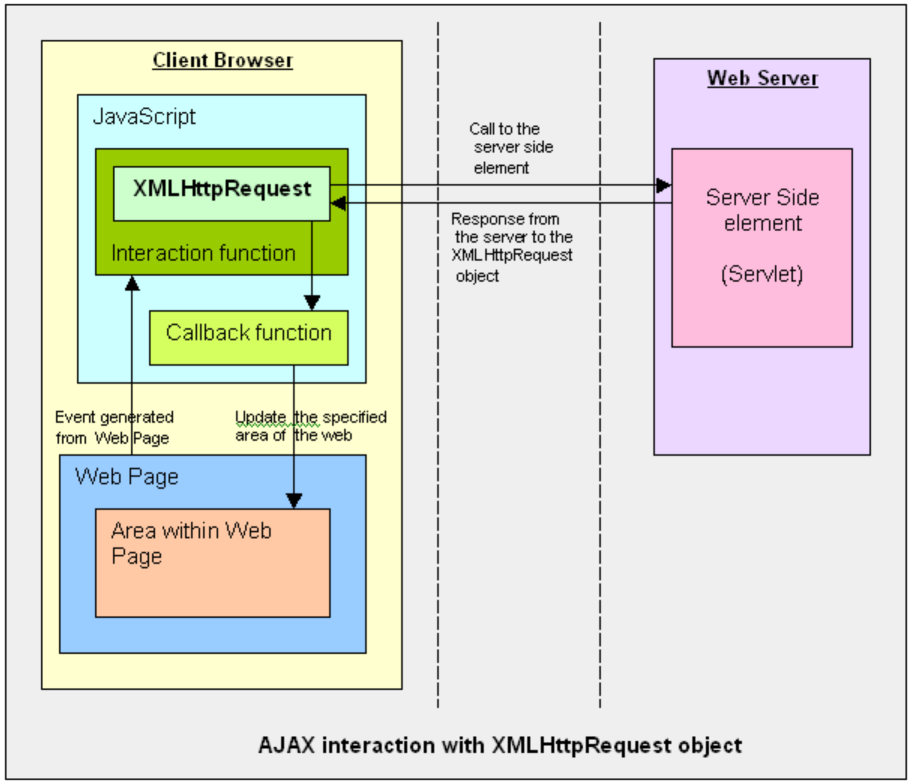


Dynamic Web: AJAX process

- AJAX is a concept for creating dynamic web pages.
  - AJAX stands for Asynchronous JavaScript and XML (but not limited to XML)
  - AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes
  - reloading the whole page to get new or updated data is not needed anymore
    ➔Classic web pages, (which do not use AJAX) must reload the entire page if the content should change
    - avoids the "click-wait-refresh" pattern
  - a dynamic web site can mimic the feel of a desktop app by presenting a continuous user experience rather than disjoint pages => “Single Page Web App”

- AJAX is based on internet standards, and typically uses a combination of:
  - **XMLHttpRequest** object (to retrieve data from a web server) and newer async communication APIs
  - **JavaScript/DOM** (to display/use the data)

Dynamic Web: AJAX process


Client Browser
JavaScript
XMLHttpRequest
Interaction function
Callback function
Event generated from Web Page
Update the specified area of the web
Web Page
Area within Web Page
Call to the server side element
Response from the server to the XMLHttpRequest object
Web Server
Server Side element
(Servlet)
AJAX interaction with XMLHttpRequest object

http://www.wilsonmar.com/ajax_rec.htm

WEB MEDIA APIS

W3C Web Media APIs

- Media-related APIs play a central role in W3C
- A brief introduction in the most relevant APIs will be provided in this part of the lecture
- Some of the APIs like MSE, EME, Remote Playback, Second Screen, etc. will be covered in more details in other slots later on during the Semester
- https://caniuse.com/ can help you to check if a specific API or feature is supported or not

HTML Video element (<video>)

- Embeds a media player within a web page via the <video> HTML element
- Media formats and video/audio codecs support may differ between browsers.
  - MP4 format and H264 video and AAC audio codecs are the most supported format and codecs across browsers
- Most relevant attributes:
  - src, autoplay, controls, currentTime, duration, loop, muted, playsinline
- Most relevant methods:
  - play(), pause(),
- Most relevant events:
  - canplay, play, pause, playing, ended, timeupdate

More details →→ https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video

HTML Video element Example

```
<video controls id=”myVid"  src="https://example.org/video.mp4"
        poster=" https://example.org/poster.jpeg "
        width="620">
    Sorry, your browser doesn't support embedded videos, but don't worry, you can
</video>
<button id=”myBtn" ></button>

var vid = document.querySelector(" #myVid ");
vid.addEventListener('pause', (event) => {
    console.log(’The video is pause’);
});

var btn = document.querySelector(" #myBtn ");
btn.addEventListener(click', (event) => {
    vid.pause();
});
```

HTML Audio element (<audio>)

- Embeds an sound content within a web page via the <audio> HTML element
- Similar to video element, media formats and audio codecs support may differ between browsers.
- List of attributes, methods and events are similar to video element (without the video specific elements e.g. poster attribute)
- Example:

```
<audio controls>
        <source src="myAudio.mp3" type="audio/mpeg">
        <source src="myAudio.ogg" type="audio/ogg">
        Your browser doesn't support HTML5 audio.
</audio>
```

More details →→ https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio

HTML Canvas Element <canvas> and API

- The Canvas JavaScript API and the HTML <canvas> element provides the capability to draw graphics within a web page
- The Canvas API largely focuses on 2D graphics. The WebGL API, which also uses the <canvas> element, draws hardware-accelerated 2D and 3D graphics
- Example:

```
const canvas = document.getElementById('canvas’);
const ctx = canvas.getContext('2d’);
ctx.fillStyle = 'green’;
ctx.fillRect(10, 10, 150, 100);
```

More details →→ https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

WebGL

- WebGL (Web Graphics Library) provides a JavaScript API for rendering high-performance interactive 3D and 2D graphics within any compatible web browser without the use of plug-ins
- It works with the canvas element
- Supported on most modern Browser on Desktop, mobile and some TVs
- WebGL API closely conforms to OpenGL ES 2.0, WebGL 2 API introduces support for much of the OpenGL ES 3.0 feature set
- Use Cases:
  - Web-based Games
  - 360° Video Playback in Browser
  - AR/VR Application
- There are 3D JavaScript Libs build on top of WebGL like three.js that makes the use of WebGL much easier

More details →→ https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

Media Source Extensions API (MSE)

- Media Source Extensions MSE provides an API that enables plugin-free media streaming media on the Web.
- The Media Streams can be created via JavaScript and played using the <audio> or <video> elements
- The main use case is to support Adaptive Bitrate (ABR) Streaming on the Web without any plugin
- The most 2 adopted ABR formats are DASH and HLS
- There are many Open Source players like dash.js and hls.js that are built on top of MSE and support ABR playback via an easy to use JavaScript Lib
- The MSE API will be discussed in more details in a dedicated slot during the semester

More details →→ https://developer.mozilla.org/en-US/docs/Web/API/Media_Source_Extensions_API

Minimal MSE Example

```
<video controls id="myVid"> </video>
var video = document.querySelector('#myVid');
var mimeCodec = 'video/mp4; codecs="avc1.42E01E, mp4a.40.2"';
var mediaSource = new MediaSource();
video.src = URL.createObjectURL(mediaSource);
mediaSource.addEventListener('sourceopen', sourceOpen);
function sourceOpen (_) {
    var mediaSource = this;
    var sourceBuffer = mediaSource.addSourceBuffer(mimeCodec);
    fetchSegment('http://example.org/path/to/segment/seg_1.m4s', function (buf) {
        sourceBuffer.addEventListener('updateend', function (_) {
            mediaSource.endOfStream();
            video.play();
        });
        sourceBuffer.appendBuffer(buf);
    };
};
```

Encrypted Media Extensions API (EME)

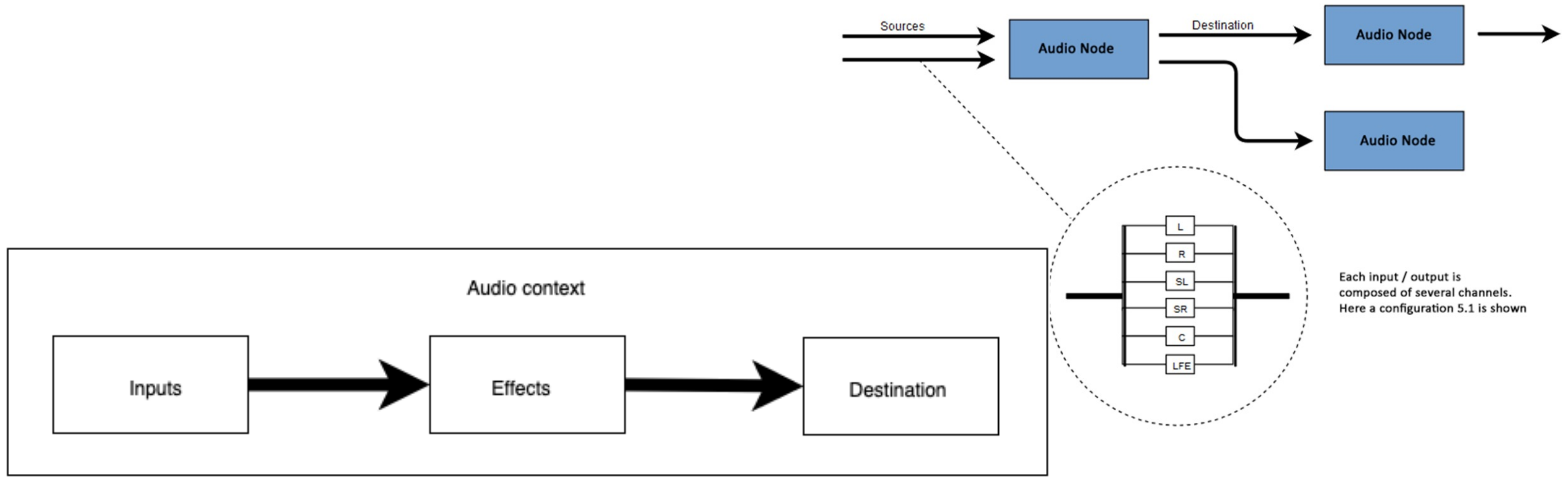
- The Encrypted Media Extensions (EME) API provides interfaces for controlling the playback of encrypted content on the Web
- It is available on most modern Browsers on Desktop, Mobile and TV, but the support of DRM Systems () vary between Browsers and Platforms. Example DRM Systems:
  - PlayReady DRM → Microsoft
  - Widevine DRM → Google

- EME and DRM will be explained in more details in a dedicated slot during the Semster

More details →→ https://developer.mozilla.org/en-US/docs/Web/API/Encrypted_Media_Extensions_API

Web Audio API

- Web Audio API provides a powerful API for controlling audio on the Web by allowing developers to choose audio sources, add effects to audio, create audio visualizations, etc.
- The API involves handling audio operations inside an audio context using audio nodes, which are linked via their inputs and outputs



More details →→ https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

Media Capture and Streams API

- The Media Capture and Streams API is related to WebRTC and provides the interfaces and methods for working with the streams.
- The **MediaStream** object represents a muxed audio- or video data and consists of zero or more **MediaStreamTrack** objects.

- Example: Access Camera Stream and display it in a video element

```
<video id=”myVid">waiting for camera</video>
video = document.getElementById(‘myVid’);

navigator.mediaDevices.getUserMedia({ video: true, audio: true })
.then(function(stream) {
        video.srcObject = stream;
        video.play();
}) .catch(function(err) {
        console.log("An error occurred: " + err);
});
```

More details →→ https://developer.mozilla.org/en-US/docs/Web/API/Media_Streams_API

Screen Capture API

- The Screen Capture API provides extends the Media Capture and Streams API to capture a screen or portion of a screen (such as a window) as a media stream. This stream can then be recorded or shared with others over the network e.g. using the WebRTC API
- Example:

```
<video id=”myVid">waiting for camera</video>
video = document.getElementById(‘myVid’);
getDisplayMedia = navigator.mediaDevices.getDisplayMedia;
getDisplayMedia({video: {cursor:"always"}, audio: false})
        .then(function(stream) {
                video.srcObject = stream;
                video.play();
        }) .catch(function(err) {
                console.log("An error occurred: " + err);
        });
```

More details →→ https://developer.mozilla.org/en-US/docs/Web/API/Screen_Capture_API

MediaStream Recording API

- The MediaStream Recording API is related to the Media Capture and Streams API and the WebRTC API
- The MediaStream Recording API captures the data generated by a MediaStream, HTMLMediaElement object or HTMLCanvasElement for further processing (save to disk, send it over the network, etc.)
- MediaRecorder is the major interface which takes the data from a MediaStream and delivers it to the web App for further processing. Example

```
var canvas = document.querySelector("canvas");
var stream = canvas.captureStream(25); // Optional frames per second argument.
var options = { mimeType: "video/webm; codecs=vp9" };
var mediaRecorder = new MediaRecorder(stream, options);
mediaRecorder.ondataavailable = function(event) {
        // Use event.data to access recorded data
};
mediaRecorder.start();
```

More details →→ https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Recording_API

COMMUNICATION APIS

XMLHttpRequest (XHR)

- The XMLHttpRequest (XHR) API is used retrieve data from a Web Server without having to do a full page refresh
- it can be used to retrieve any type of data and not just XML!!!
- Limitation: the requested data can be accessed only when it is fully downloaded
- Example:

```
var req = new XMLHttpRequest();
 req.addEventListener("load", function () {
          console.log(this.responseText);
);
req.open("GET", "http://www.example.org/example.txt");
req.send();
```

For accessing binary data → req.responseType = "arraybuffer";

More details →→ https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest

Fetch API

- Similar API to XMLHttpRequest but provides a more powerful and flexible feature set
- It provides a generic definition of Request and Response objects, which can be used in other APIs like Service Workers, Cache API etc.
- An important advantage of the Fetch API compared to XHR is that it allows access to chunks of retrieved data even before the content is completely downloaed. This is important for applications that need to access stream of data
- Example:

```
 fetch('https://example.com/file.json’)
.then(response => response.json())
.then(data => { console.log('Success:', data); })
.catch((error) => { console.log.error('Error:', error); });
```

More details https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

The WebSocket API

- The **WebSocket API** makes it possible to open a two-way (bidirectional) interactive communication session between the client/browser and a server
- XHR and Fetch APIs uses the HTTP protocol (request/response) which closes the connection when the response the downloaded .
- With WebSocket API, the client can send messages to a server and receive event-driven responses without having to poll the server for a reply.
- Node.js based WebSocket Server → https://github.com/websockets/ws
- Example:

```
var socket = new WebSocket("ws://example.org/socketserver");
socket.onopen = function (event) {
   socket.send(”Ping");
};
socket.onmessage = function (event) {
    console.log(event.data);
}
```

More details https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

WebRTC – Web Real Time Communication

WebRTC

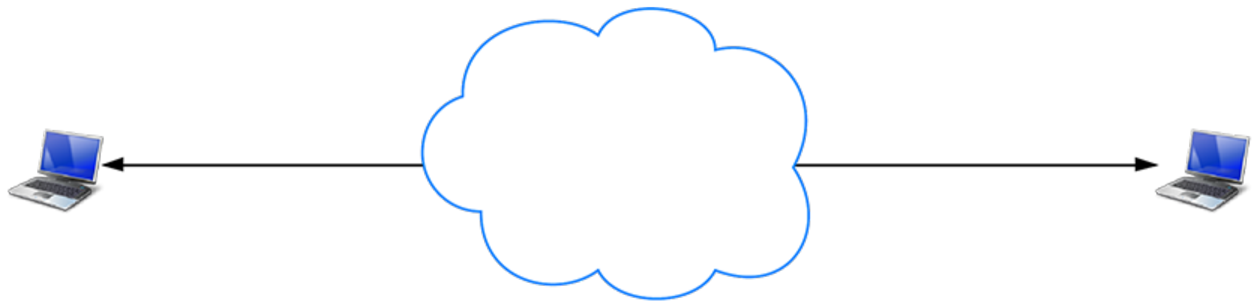
Why?

- No software downloads and no Plug-Ins
  - No installations like Skype, WebEx or GotoMeeting
- Solves incompatibilities of browsers for real time communication
- Real Peer-2-Peer Connection between Browsers
  - Direct exchange of audio, video, and data between Browsers

For what?

- Communication in general (Telcos)
- (Existing) Video/Web conference system providers
- Peer 2 Peer Data exchange
- Training Providers, Customer Support

WebRTC

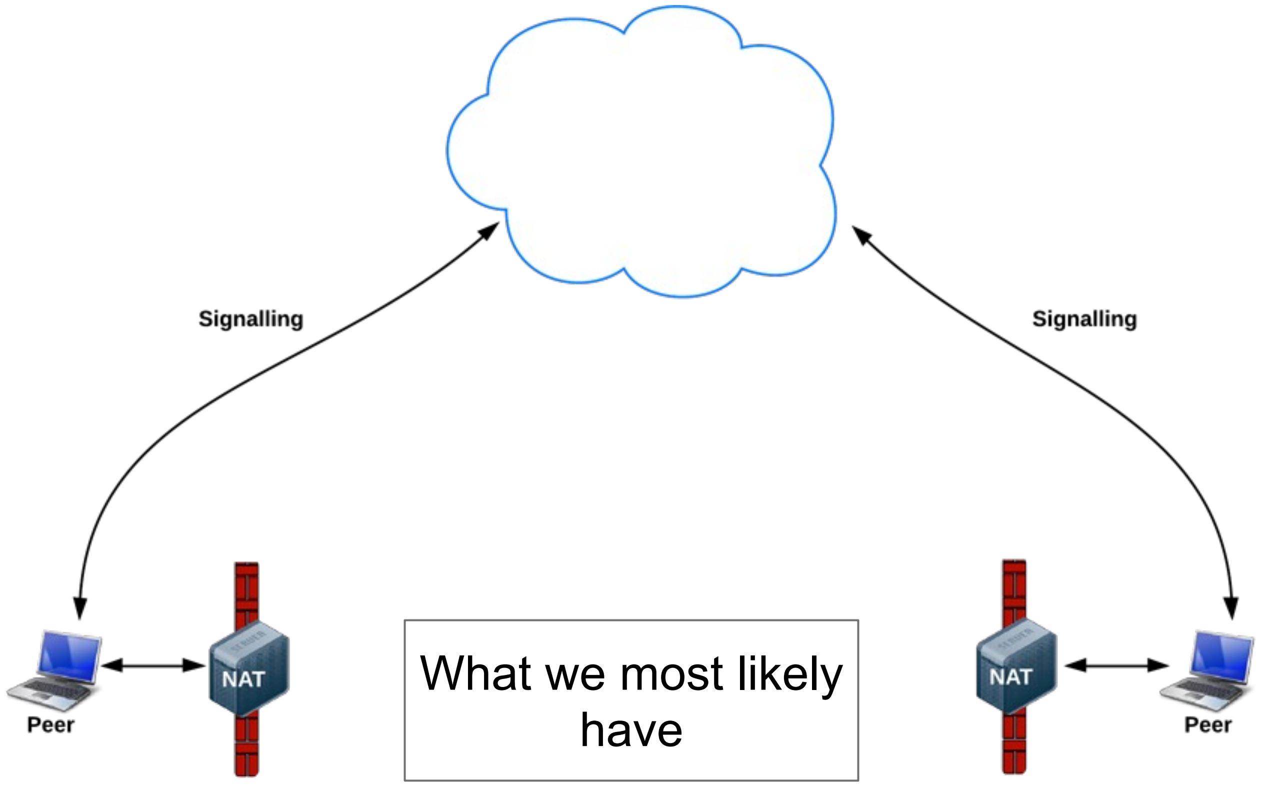


What we need:

direct communication between peers

Image: http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/

WebRTC



What we most likely have

Image: http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/

WebRTC

Using STUN servers to get the public IP to be communicated via signaling and trying to create a UDP session between the clients

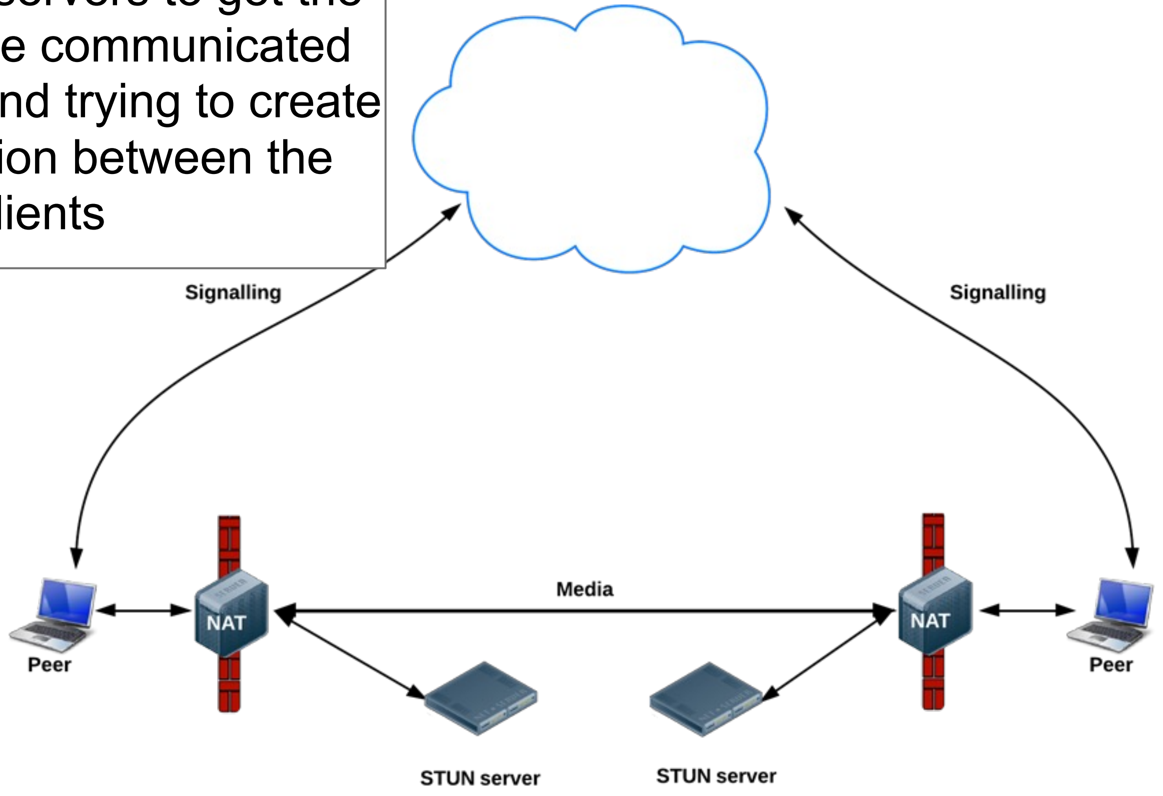


Image: http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/

WebRTC

If a direct Peer-2-Peer connection cannot be established, a TURN media stream relay server can be used.

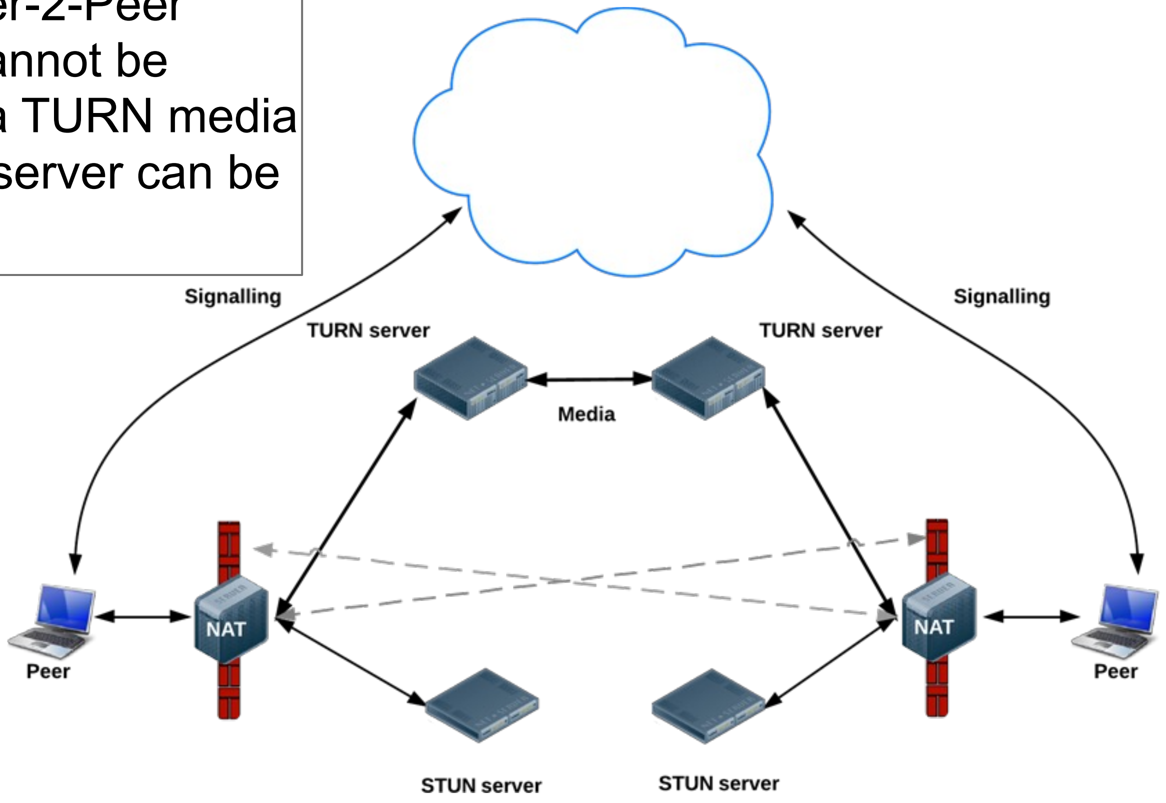
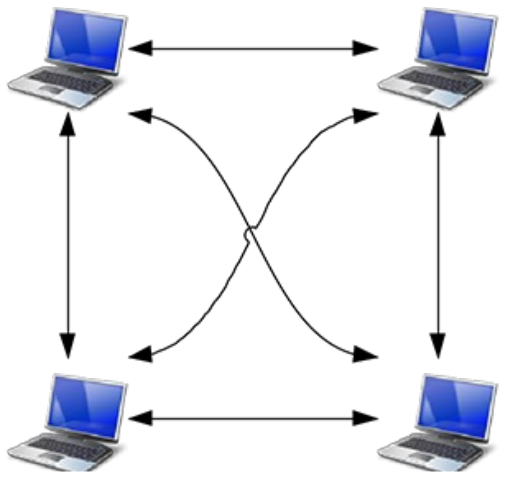


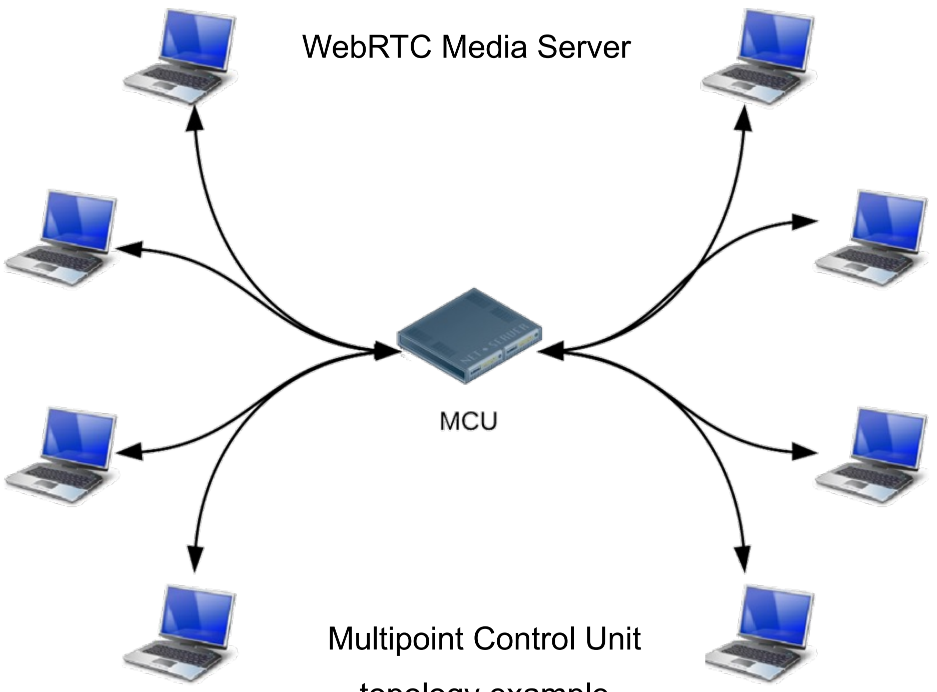
Image: http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/

WebRTC

WebRTC based audio/video conferencing



Fully-meshed



Multipoint Control Unit topology example

https://www.html5rocks.com/en/tutorials/webrtc/basics/

WebRTC API

- Provides interfaces to use WebRTC in the Browser
- Enables streaming any kind of media streams and data between 2 browsers, but can be also used to communicate with a WebRTC server
- Media Stream Sources: Camera, Microphone, Screen Capture, Canvas
- Example:

```
// Local Peer
localConnection = new RTCPeerConnection();
sendChannel = localConnection.createDataChannel("sendChannel");
sendChannel.onopen = handleSendChannelStatusChange; sendChannel.onclose =
handleSendChannelStatusChange;
// wait for connection open
sendChannel.send(“Hello WebRTC”);
```

```
// Remote Peer
remoteConnection = new RTCPeerConnection();
remoteConnection.ondatachannel = function (event) {
      receiveChannel = event.channel;
      receiveChannel.onmessage = function (event) {
            console.log("Receive message: " + event.data);
      };
};
```

More details https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API

Technische Universität Berlin

berlin

Thank you for your attention.