

# **BROWSERAPIS 2**

**MMWP2024 - LV06**

## INHALTSVERZEICHNIS

- Organisation
- Storage- und IndexedDB-API
- File- und Webworker API
- Neuere APIs

## INHALTSSCHWERPUNKTE

- Vorstellung weiterer Browser-APIs
- Beispielhafte Benutzung bestimmter Browser-APIs
- Neuste Browser-APIs

## VORAUSSETZUNG

Der Ausgangspunkt dieser Vorlesungsreihe ist das Wissen über funktionsweise von BrowserAPIs

- Möglichkeit über Javascript Browser APIs anzusprechen
- Eventhandling in Javascript sowie Ausführungsreihenfolge von Events
- HTML5 Elemente wie Forms und Eingabefelder

# ZIELE

Vorstellung von:

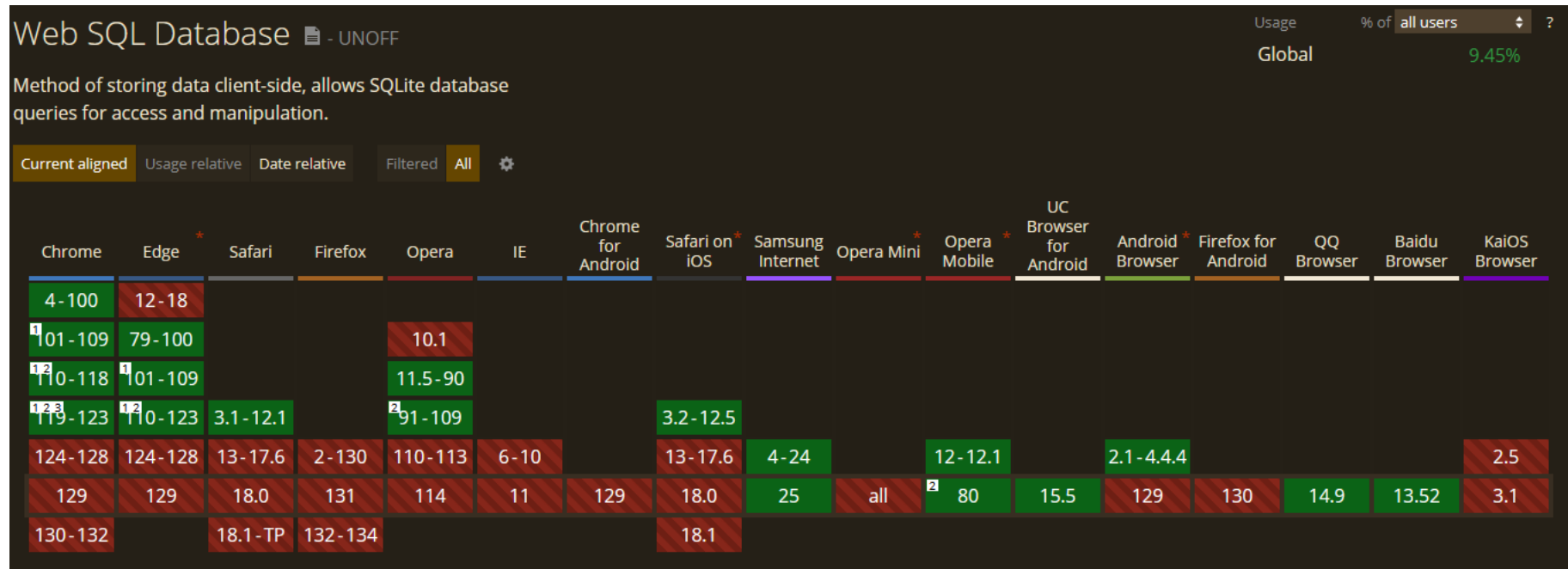
- Storage-APIs
- IndexedDB API
- File API
- Web Workers API
- Web Audio API
- WebGPU API
- WebXR Device API

## STORAGE-APIS

Es gibt zwei APIs in HTML5, die clientseitige Speicherung von Webseitendaten befördern

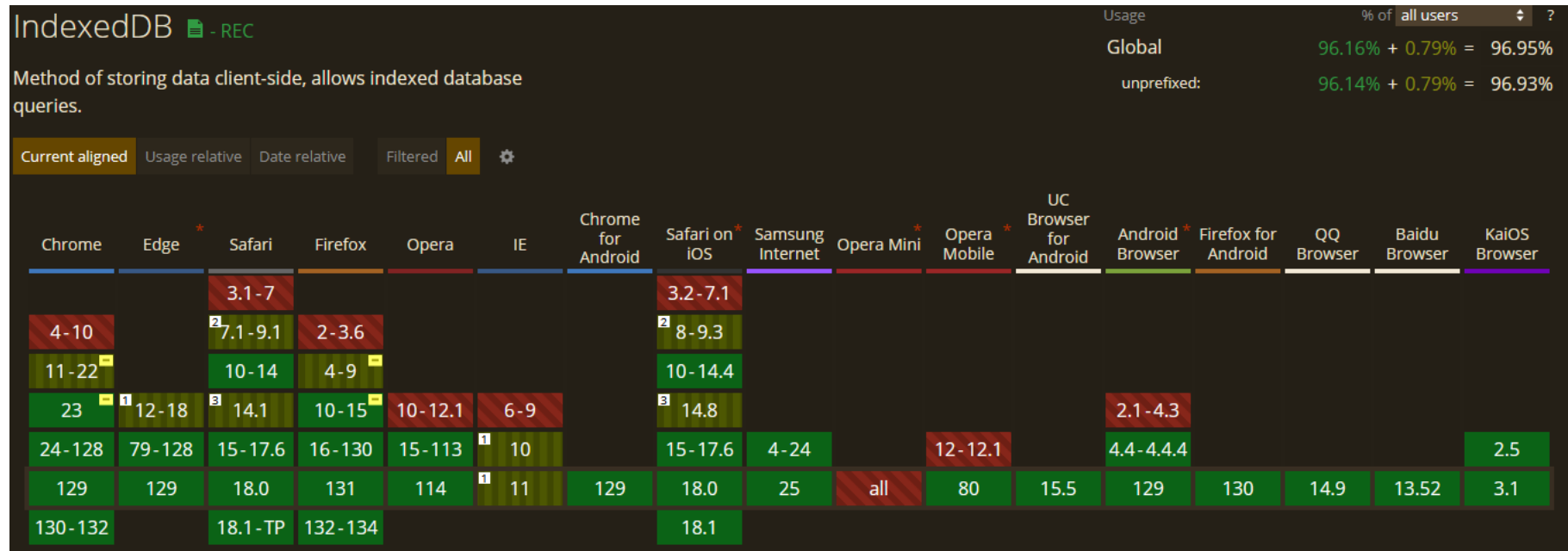
- Web-Storage-API: gute Unterstützung durch Webbrowser
- Indexed-Database-API (ersetzt die obsoleete Web-SQL-Database-API): benötigt zudem möglichst Framework-Unterstützung

# STORAGE-APIS - SUPPORT - 1



Quelle


# STORAGE-APIS - SUPPORT - 2



Quelle




# STORAGE-APIS - SUPPORT - 3

Web Storage - name/value pairs  - LS

Usage % of all users Global 97.07%

Method of storing data locally like cookies, but for larger amounts of data (sessionStorage and localStorage, used to fall under HTML5).

Current aligned Usage relative Date relative Filtered All 

Chrome	Edge *	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
		3.1-3.2	2-3	10.1	6-7											
4-128	12-128	4-17.6	3.5-130	11.5-113	8-10		3.2-17.6	4-24		12-12.1		2.1-4.4.4				2.5
129	129	18.0	131	114	11	129	18.0	25	all	80	15.5	129	130	14.9	13.52	3.1
130-132		18.1-TP	132-134				18.1									

Quelle

# WEB STORAGE API

Diese API hat zwei wichtige Attribute:

- sessionStorage
  - Es werden temporäre Daten während einer in einem Fenster laufenden Session sicher clientseitig verwahrt
  - Sie erlöschen, sobald Webseiten anderer Herkunft aufgerufen werden oder Reiter bzw. Fenster geschlossen werden
  - Die Speicherbereiche sind strikt getrennt
- localStorage
  - Es werden große Dateien clientseitig persistent gespeichert
  - Kann nur über Cache löschen vom Benutzer gelöscht werden
  - Die Daten stehen der Anwendung, welche sie angelegt hat, dauerhaft zur Verfügung
  - Die Web Storage API funktioniert im Allgemeinen nur, wenn sie dem Browser über einen Webserver geliefert wird
- Cookies sind an Browser gekoppelt

## WEB STORAGE API - NUTZUNGSWEISE

- Die von beiden Attributen genutzten Funktionen in JavaScript sind weitgehend die gleichen
- Pro Applikation/Domain im Webbrowserfenster oder -reiter stehen aktuell etwa 50 MB zur Verfügung, im IE/Edge 100 MB
- Da Löschen möglich ist, muss über Speichermanagement nachgedacht werden
- Browser fragen manchmal um Erweiterung des Speicherplatzes an

## WEB STORAGE API - SPEICHERBEREICHE

Jede Seite ergeben unterschiedliche Speicherbereiche, auch wenn die gleiche Seite geladen wird:

- <https://dogs.com/>
- <http://dogs.com/>
- <https://dogs.com:80>
- <https://www.dogs.com>

## WEB STORAGE API - WEITERE INFORMATIONEN

- Beide Speicherarten sind weiterhin browserspezifisch, d.h. Browser unterschiedlicher Hersteller haben für denselben Ursprung getrennte Speicherbereiche
- Web Storage heißt heute DOM-Storage (was aber mit dem DOM wenig zu tun hat)
- Im Gegensatz zu Cookies, auf die sowohl Server als auch Client zugreifen können, wird Web Storage vollständig vom Client gesteuert
- Über die Eingabe von (in Firefox) „about:config“ in der Adresszeile kann der Wert von `dom.storage.enabled` von `true` zu `false` geändert und Web-Storage-Objekte können somit abgeschaltet werden

# WEB STORAGE API - VERWALTEN UND LÖSCHEN VON DATEN

- Firefox: [Quelle](#)
- Chromium: [Quelle](#)
- [Firefox Umsetzung](#)
- Datenschutz sollte bei Nutzung der API bedacht werden!

## WEB STORAGE API - BEISPIEL - 1

```
1 <div id="result"></div>
2 <script>
3 // Store
4 localStorage.setItem("lastname", "Meyer");
5 // Retrieve
6 document.getElementById("result").innerHTML = localStorage.getItem("lastn
7 </script>
```

## WEB STORAGE API - BEISPIEL - 2

```
1 <p><button onclick="clickCounter()" type="button">Click me!</button></p>
2 <div id="result"></div>
3 <script>
4 function clickCounter() {
5     if (localStorage.clickcount) {
6         localStorage.clickcount = Number(localStorage.clickcount) + 1;
7     } else {
8         localStorage.clickcount = 1;
9     }
10    document.getElementById("result").innerHTML = "You have clicked the bu
11 }
12 </script>
```



## JSON STRUKTUR

- Objekt - beginnt mit { und endet mit }
  - Es enthält eine durch Kommata geteilte, ungeordnete Liste von Eigenschaften
  - Objekte ohne Eigenschaften („leere Objekte“) sind zulässig
- Array - beginnt mit [ und endet mit ]
  - Es enthält eine durch Kommata geteilte, geordnete Liste von Werten gleichen oder verschiedenen Typs
  - Leere Arrays sind zulässig

## JSON STRUKTUR

```
1 {  
2   "Herausgeber": "Xema",  
3   "Nummer": "1234-5678-9012-3456",  
4   "Deckung": 2e+6,  
5   "Waehrung": "EURO",  
6   "Inhaber":  
7   {  
8     "Name": "Muster",  
9     "Vorname": "Z",  
10    "Teilnahme am Bonusprogramm": true,  
11    "Hobbys": ["Reiten", "Golfen", "Lesen"],  
12    "Alter": 42,  
13    "Kinder": [],  
14    "Partner": null  
15  }
```

## WEB STORAGE API - SPEICHERN

```
1 localStorage.setItem('students', JSON.stringify({
2   name: 'participant',
3   list: ['Justus Bley', 'Peter Miller', 'Bob Black']
4 }));
5 var participants = JSON.parse(localStorage.getItem('students'));
6 console.log(participants.name, participants.list);
7 document.getElementById("result").innerHTML = participants.name + ": " +
8
9 localStorage.setItem('key', JSON.stringify({firstname: 'Peter', lastname:
10 alert(JSON.parse(localStorage.getItem('key')).firstname);
11 // Peter
```

## WEB STORAGE API - LÖSCHEN

```
1 localStorage.removeItem("key1");  
2 sessionStorage.removeItem("key2");  
3 localStorage.clear();  
4 sessionStorage.clear();
```

## WEB STORAGE API - WEITERES

- Spezifikation des W3C: <http://www.w3.org/TR/webstorage/#storage>
- Work-around für ältere Browser und Tutorial: [Quelle](#)
- Beispiele: [Quelle](#)
- Demo (gemeinsam aufrufen):
  - Auswahl tätigen, neu laden: Auswahl ist beständig: [Quelle](#)
  - sehen was passierte: [Quelle](#)

## INDEXEDDB API (2.0) - 1

- Es ist ein Datenbanksystem, das indizierte Informationen clientseitig speichert
- Anders als bei SQL werden Informationen als Objekte (Datensätze oder Records) in sogenannten Objektspeichern (Tabellen) gespeichert
- Jede Datenbank ist durch Computer und Website oder Anwendung eindeutig bestimmt durch Aufruf mit „(Name, Version)“
- Es ist eine API der untersten Ebene

## INDEXEDDB API (2.0) - 2

- Der Entwickler muss viel selbst programmieren und dabei die Bedingungen jedes Prozesses in jeder Operation prüfen
- Deshalb wird der Einsatz der IndexedDB API meist jQuery und ähnliche Frameworks bzw. Bibliotheken verwendet
- Diese nehmen viel Programmierarbeit ab

## INDEXEDDB API (2.0) - 3

- Im Firefox sind die IndexedDB-Datenbanken beispielsweise in einer ganz normalen SQLite-Datenbank gespeichert
- Chromium hingegen setzt auf LevelDB



## INDEXEDDB API (2.0) - BEISPIELE

- Tutorial: [Quelle](#)
- [Beispiel](#)

## INDEXEDDB API (2.0) - WEITERES

- W3C-Spezifikation: [Quelle](#)
- Kurzreferenz: [Quelle](#)
- Dexie.js (ein Wrapper für die Arbeit mit IndexedDB):  
[Quelle](#)

## FILE API

- Aktuell wird die File-API in der Web Workers API
- Früher waren File API (Grundvariante), File API: Directories & System, bzw. File API: Writer noch im Einsatz
- Ziele sind es, Dateien auf dem Computer des Benutzers über Webseiten zu lesen, zu verarbeiten und zu erstellen
- Das Hoch- und Herunterladen von Dateien funktionierte schon sehr viel früher


## FILE API - IDEE

- Die Grundidee ist isolierte lokale Speicherung in einer clientseitigen kleinen objektorientierten Datenbank bzw. in einem kleinen Dateisystem
- Beides würde anwendungsspezifisch funktionieren
- Es soll nur zwei „Lademöglichkeiten“ in den Verarbeitungsbereich geben (zu ladende Dateien müssen bereits existieren): das `<input>`-Element und Drag&Drop-Operation

## FILE API - BEISPIELE


- [CSS Tricks Beispiele](#)
- [jQuery Beispiel](#)
- [Google File API guide](#)

# STORAGE-APIS - SUPPORT

File API  - WD

Usage % of all users Global 96.33% + 0.64% = 96.96%

Method of manipulating file objects in web applications client-side, as well as programmatically selecting them and accessing their data.

Current aligned Usage relative Date relative Filtered All 

Chrome	Edge *	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
4-5		3.1-5										2.1-2.3				
<sup>1</sup> 6-12		<sup>1</sup> 5.1	2-3.5	10.1			3.2-5.1					<sup>1</sup> 3-4.3				
<sup>2</sup> 13-37	<sup>2</sup> 12-18	<sup>2</sup> 6-9.1	<sup>2</sup> 3.6-27	<sup>2</sup> 11.5-24	6-9		<sup>2</sup> 6-9.3					<sup>2</sup> 4.4				
38-128	79-128	10-17.6	28-130	25-113	<sup>2</sup> 10		10-17.6	4-24		<sup>2</sup> 12-12.1		4.4.4				2.5
129	129	18.0	131	114	<sup>2</sup> 11	129	18.0	25	all	80	15.5	129	130	14.9	13.52	3.1
130-132		18.1-TP	132-134				18.1									

Quelle

## STORAGE-APIS - WEITERES

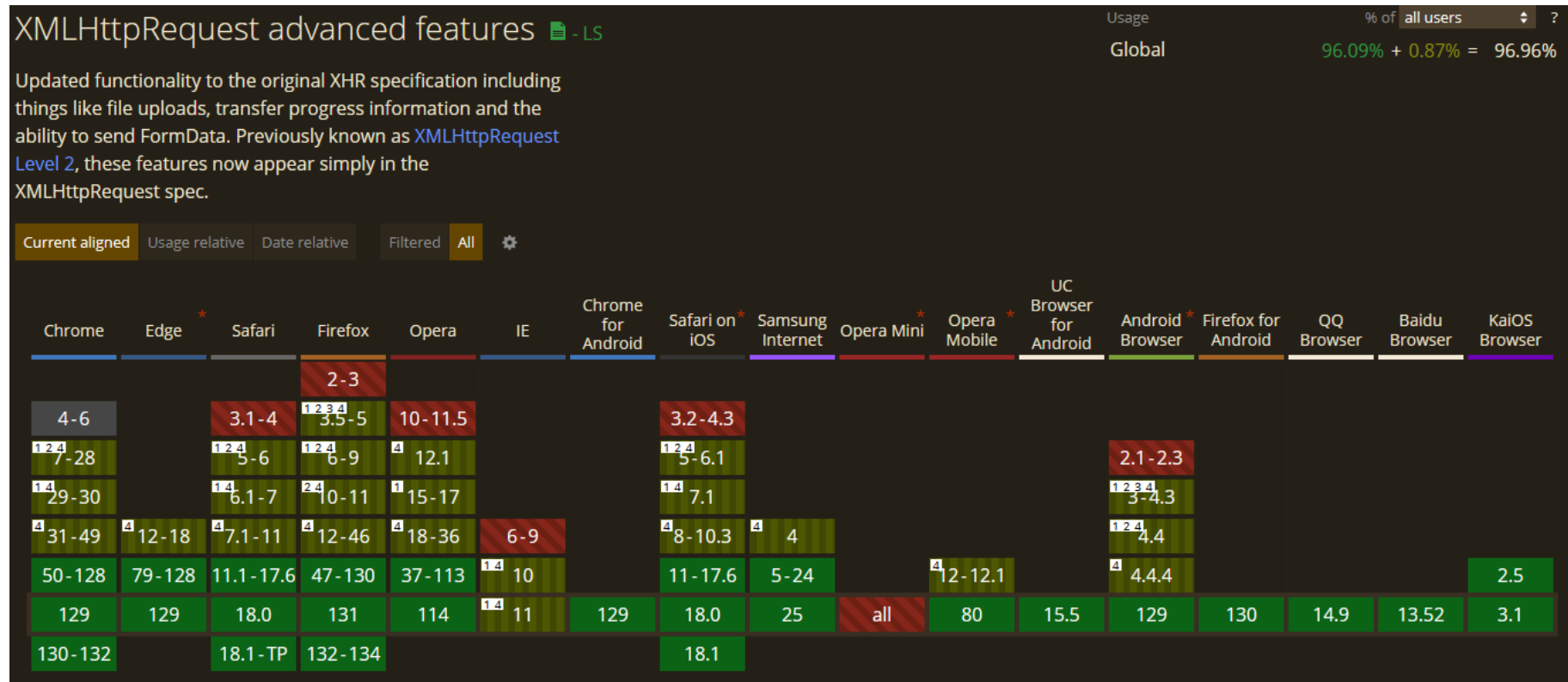
- W3C-Spezifikation: [Quelle](#)
- Datei-Upload (nur Dateiname, Dateigröße und MIME-Type sind ausgebenbar): [Quelle](#)
- Beschreibung dazu: [Quelle](#)

# ASYNCHRONE ANFRAGEN - XMLHTTPREQUEST ADVANCED FEATURES

- XMLHttpRequest advanced features (vorher: XMLHttpRequest-API Level2)
- Es ist eine API für AJAX-Anwendungen geschaffen worden, die vollständig und browserübergreifend funktioniert
- Neue Events, unter anderem solche, die den Ladefortschritt nachvollziehen oder einen Abbruch des Ladens ermöglichen; die eine Datei hochladen
- Cross-Origin Requests sind möglich, d.h. der Webserver muss per Serversprache des aufgerufenen Scripts oder per Konfiguration des Webserver selbst AJAX- Anfragen anderer Domains erlauben (spezifisch oder aller)
- [W3C Beispiel](#)



# XMLHttpRequest ADVANCED FEATURES - SUPPORT

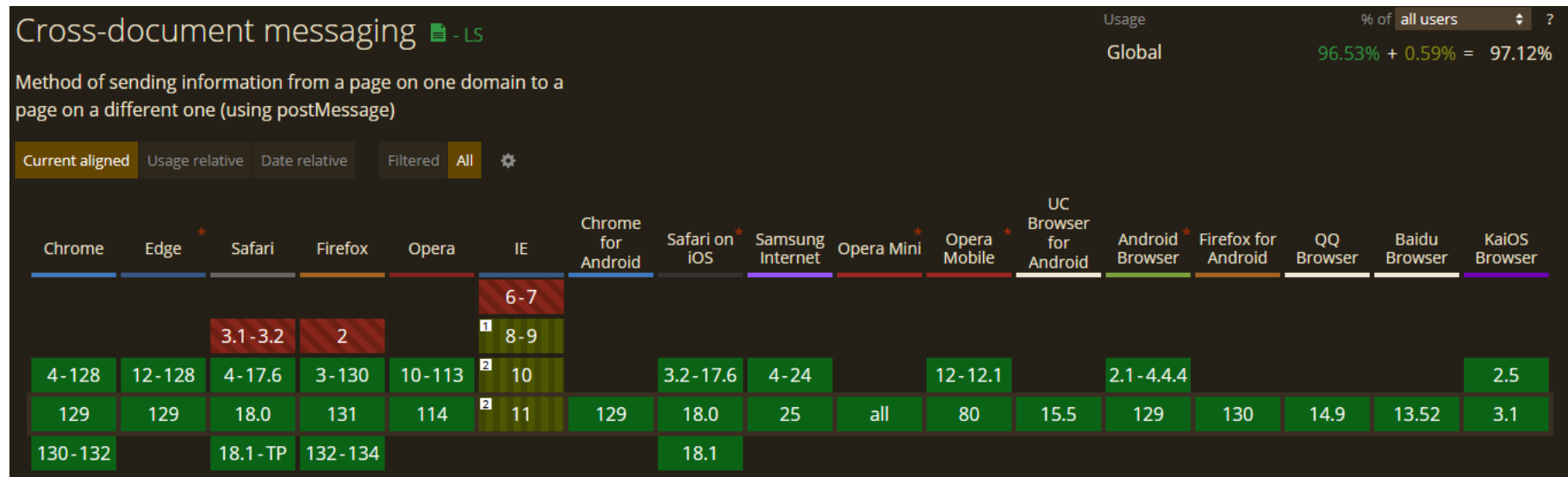


Quelle

## WEB MESSAGING API

- Cross Document Messaging API (kurz: Web Messaging API)
- Diese API soll Kommunikationsbarrieren zwischen verschiedenen Frames und Fenstern überwinden
- Auch Anwendungen mit verschiedenen Domänen als Ursprung können miteinander kommunizieren
- Als Kennungen dienen Hostname und Port der Domänen. Für Tests sind zwei Domänen und Webserver erforderlich
- Die Nachrichtenübermittlung ist asynchron
- [Mozilla Beispiel](#)

# WEB MESSAGING API - SUPPORT



Quelle

## WEB SOCKETS (API) / WEBTRANSPORT API

- Ziel ist es, in der Kommunikation von Client und Webserver Echtzeitanwendungen zu ermöglichen
- Web Sockets ist bereits in HTTP/2 integriert, WebTransport API arbeitet mit HTTP/3
- Die Verbindung wird über ein persistentes TCP-Socket ohne HTTP-Header hergestellt
- Es wird ein Websockets-Server-Script benötigt (z.B. in Rust, ASP.NET, PHP, Python, Node.js)

# WEBSOCKET - SUPPORT

WebSocket API

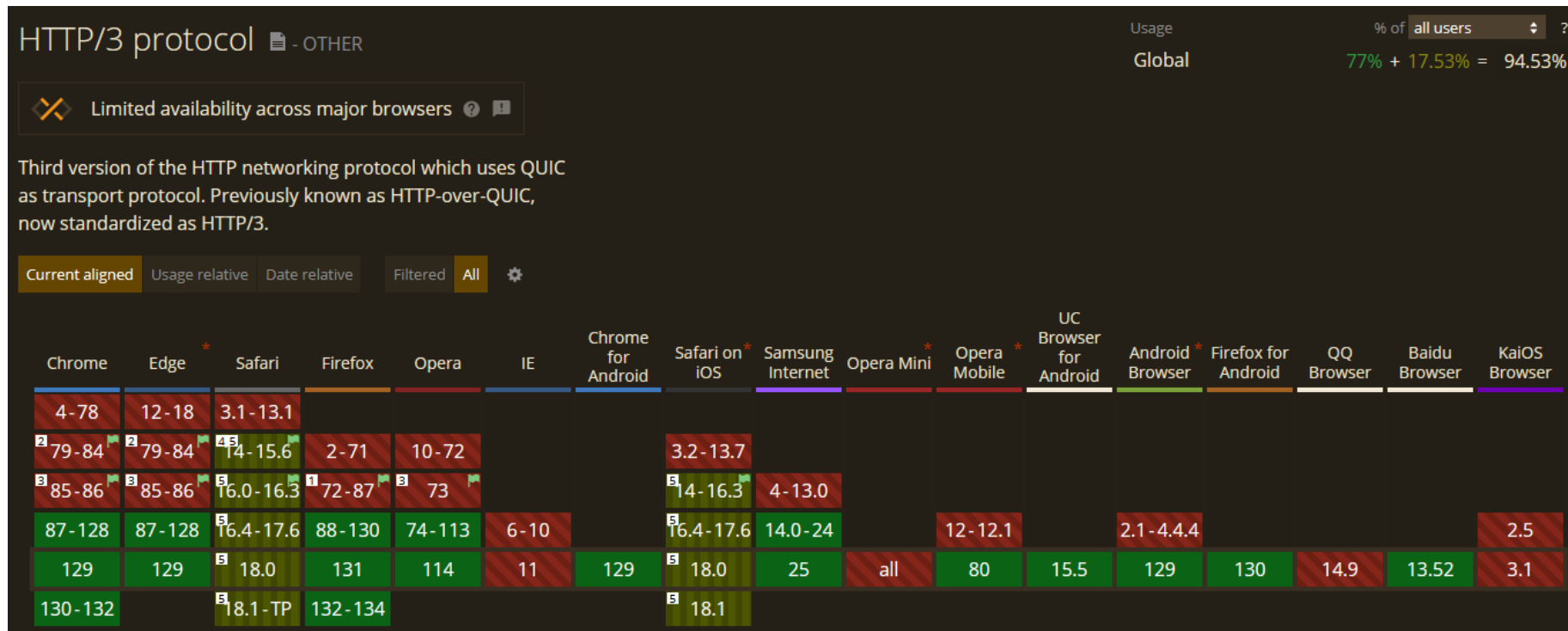
Usage % of all users 96.83%  
unprefixed: 96.83%

Current aligned Usage relative Date relative Filtered All

Chrome	Edge *	Safari	Firefox	Opera	IE ⚠ *	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
			2-6													
4		3.1-4	7-10	10-11.5	6-9		3.2-4.1			12		2.1-4.3				
5-130	12-130	5-18.0	11-131	12.1-113	10		4.2-18.0	4-25		12.1		4.4-4.4.4				2.5
131	131	18.1	132	114	11	131	18.1	26	all	80	15.5	131	132	14.9	13.52	3.1
132-134		18.2-TP	133-135				18.2									

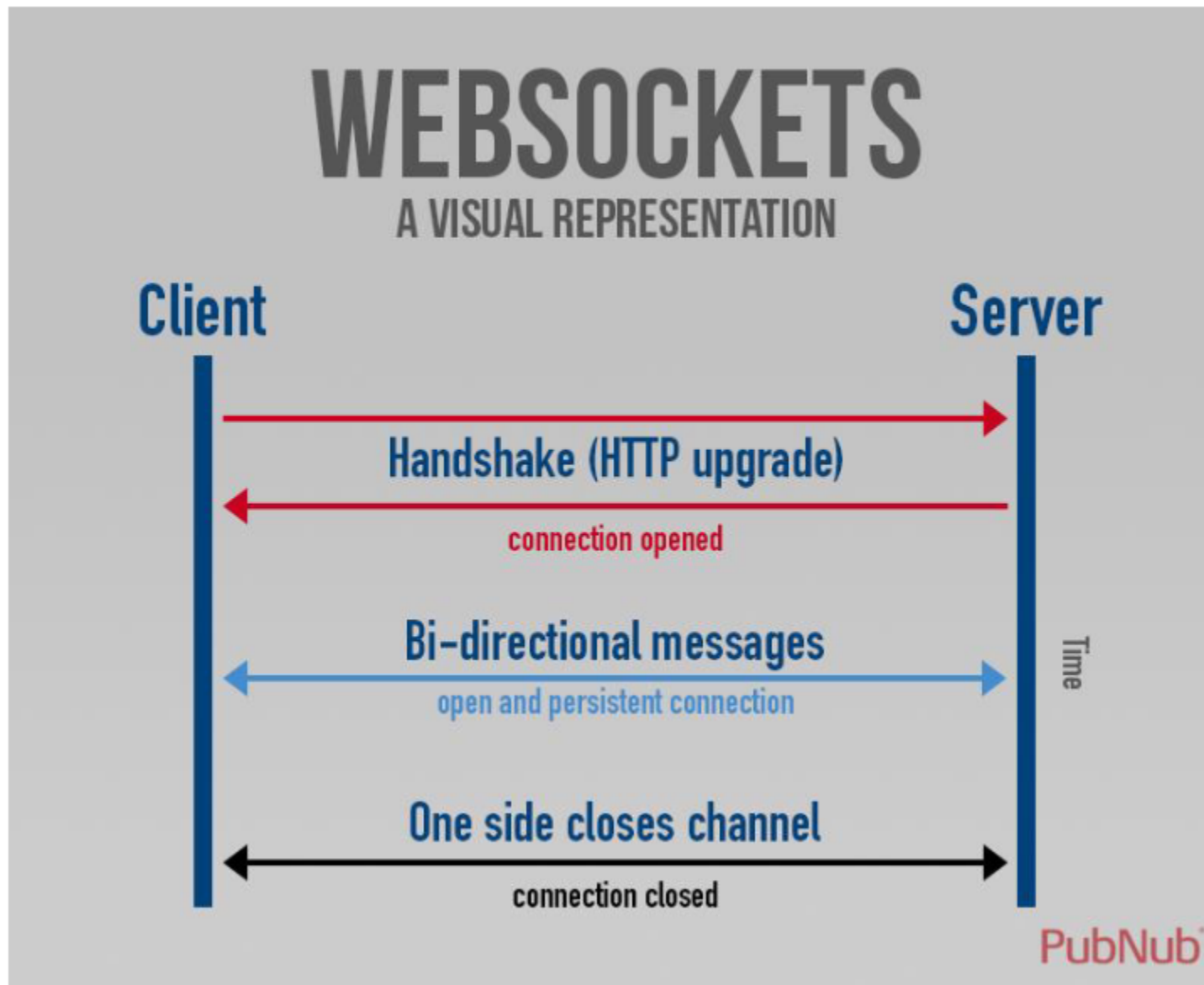
Quelle

# WEBTRANSPORT API - SUPPORT



Quelle

# WEB SOCKETS - FUNKTIONSWEISE - 1



Quelle

## WEB SOCKETS - FUNKTIONSWEISE - 2

- Web Sockets sind Transfer Control Protocol (TCP) basierte Netzwerkprotokolle zur bidirektionalen Kommunikation zwischen Web-Anwendungen und einem Web-Socket-Server
- Notify-Systeme (Push-Service) und Multiplexing wird von WebSockets unterstützt
- [Mozilla Websocket Beispiel](#)



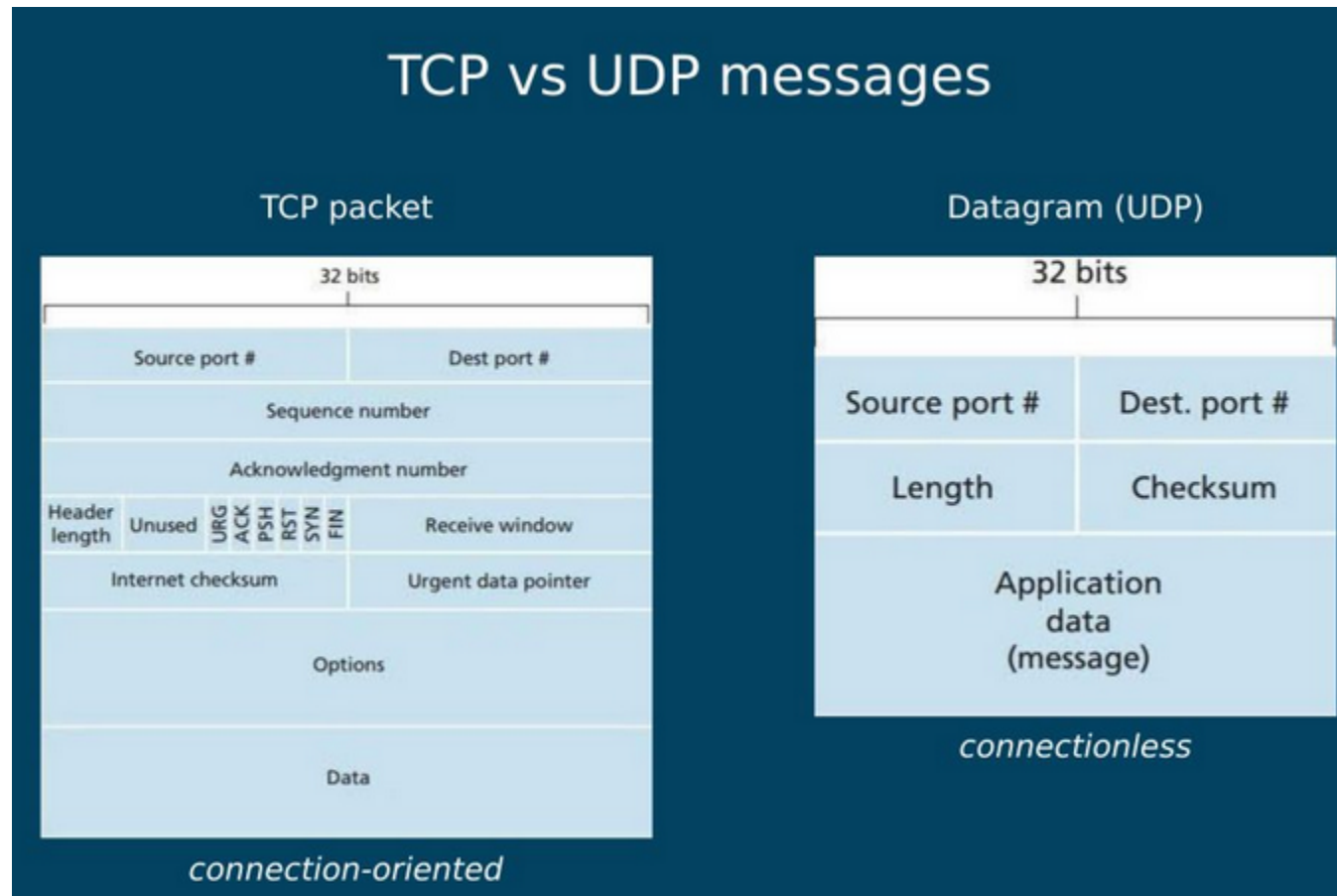
## WEB SOCKETS - BEISPIEL

```
1 var socket = new WebSocket(urlToWebsocketServer);
2
3 // callback-Funktion wird gerufen, wenn die Verbindung erfolgreich aufge
4 socket.onopen = function () {
5     console.log("Verbindung wurde erfolgreich aufgebaut");
6 };
7
8 // callback-Funktion wird gerufen, wenn eine neue Websocket-Nachricht ei
9 socket.onmessage = function (messageEvent) {
10     console.log(messageEvent.data);
11 };
12
13 // callback-Funktion wird gerufen, wenn ein Fehler auftritt
14 socket.onerror = function (errorEvent) {
15     console.log("Error! Die Verbindung wurde unerwartet geschlossen").
```

## WEBTRANSPORT API UND QUIC

- HTTP3 / QUIC [Quelle](#)
- Ein wichtiger Unterschied bei HTTP/3 ist, dass es auf QUIC, über UDP läuft
- QUIC ist so konzipiert, dass es schnell ist und einen schnellen Wechsel zwischen den Netzen ermöglicht
- QUIC eignet sich besser für die mobile Internetnutzung
- Faster connection establishment: QUIC kombiniert die kryptographischen und Transport-Handshakes
- Zum Nachlesen: [Quelle](#)

# TCP VS UDP



Quelle

# HTTP1/2 VS HTTP3

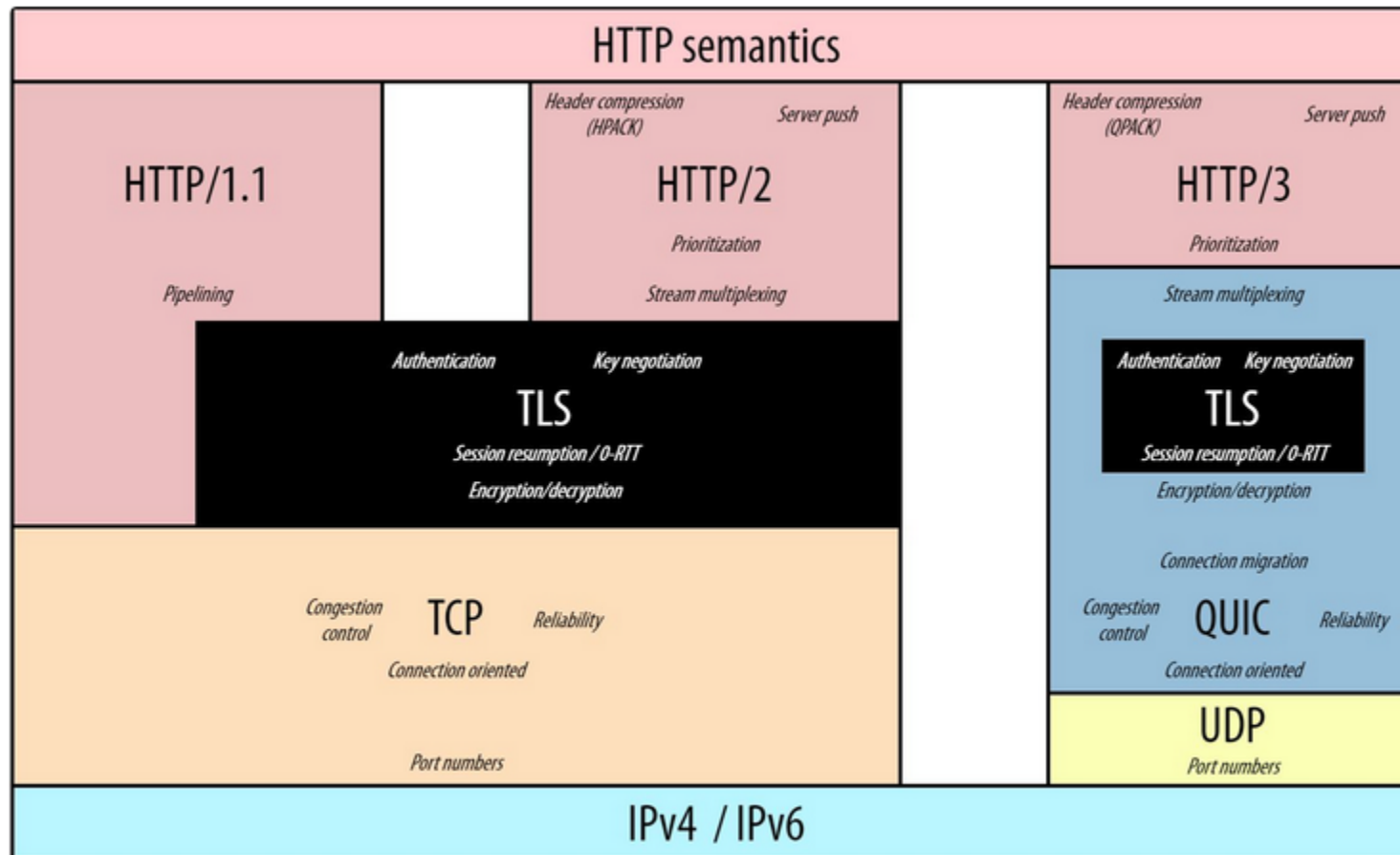
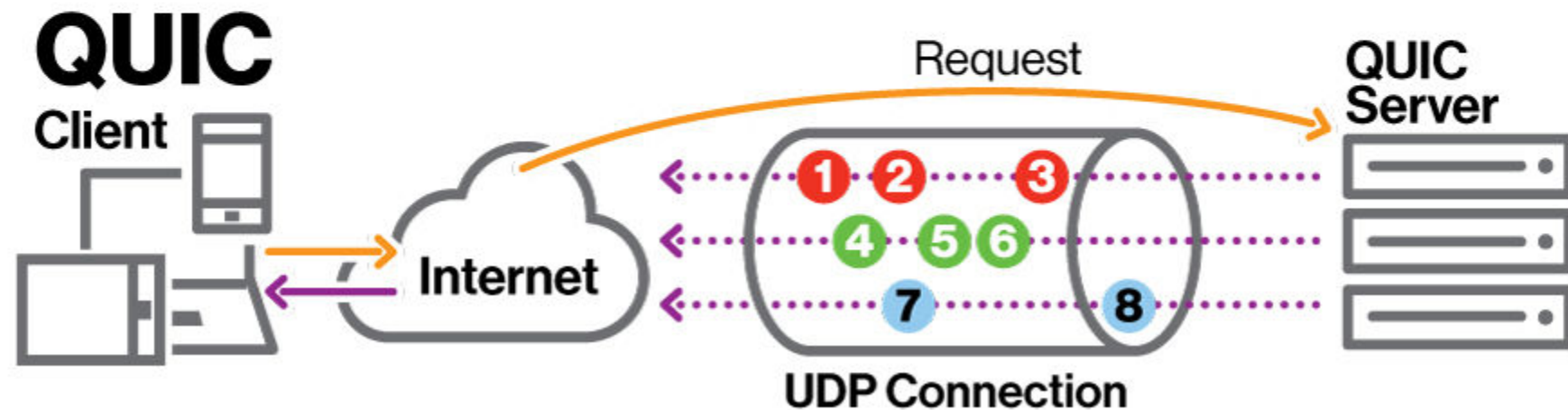
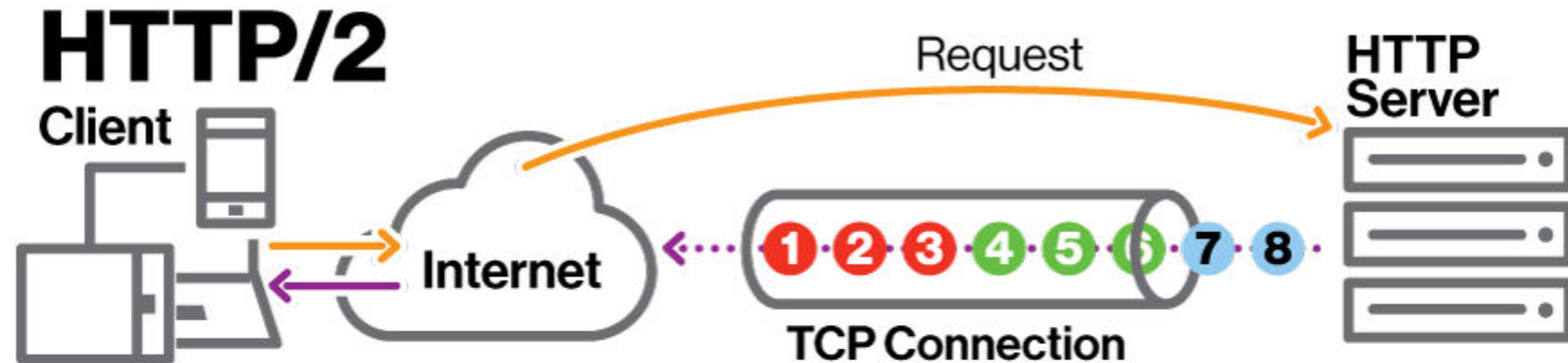


Image credit: Robin Marx: H3 Protocol Stack; GitHub

Quelle

# HTTP2 VS HTTP3



Quelle

## WEBSOCKETS, WEBTRANSPORT, XMLHTTPREQUEST - WEITERES - 1

- W3C-Spezifikation XMLHttpRequest: [Quelle](#)
- W3C-Spezifikation WebMessaging: [Quelle](#)
- W3C-Spezifikation WebTransport: [Quelle](#)
- W3C-Spezifikation WebSockets: [Quelle](#)

## WEBSOCKETS, WEBTRANSPORT, XMLHTTPREQUEST - WEITERES - 2

- Tutorial für postMessage: [Quelle](#)
- Tutorial und Screencast-Demo: [Quelle](#)
- Tutorial für HTTP3 [Quelle](#)

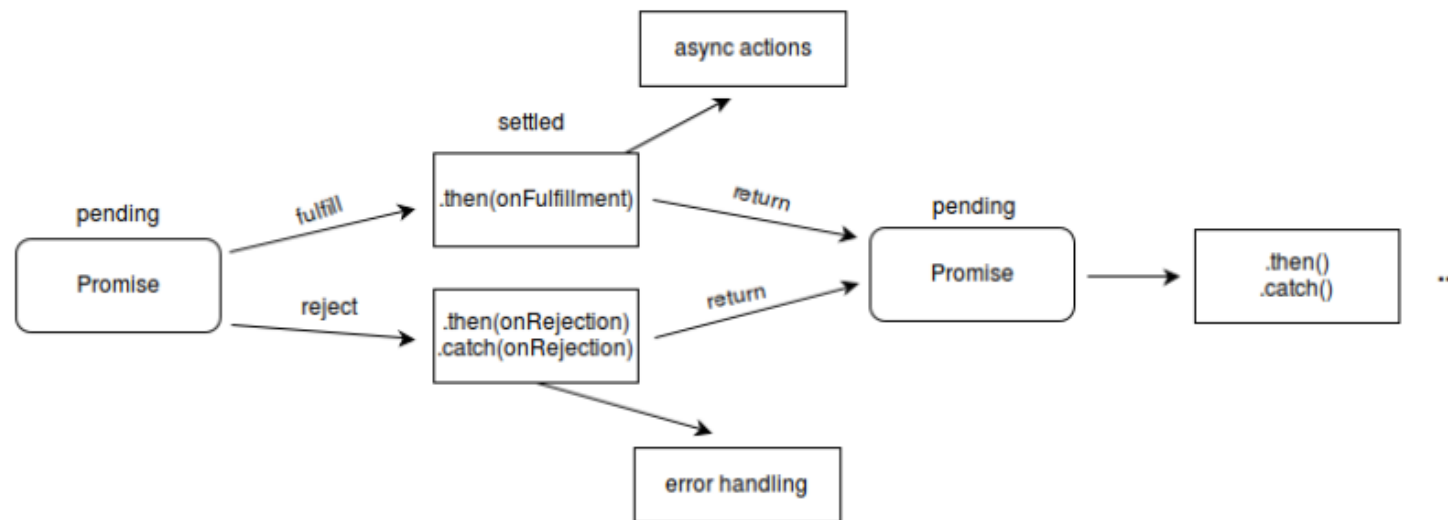
## WENN PAKETE LÄNGER BENÖTIGEN ZUM LADEN

- JavaScript ist keine Multithread-Sprache, alles muss nacheinander abgearbeitet werden
- Bei hohen Latenzen, oder großen Dateien wird der Renderthread z.B. blockiert
- Die Web Workers API bietet die Möglichkeit, Code im Hintergrund in separaten Threads zu verarbeiten
- Damit laufen Funktionen Multitaskingfähig
- Die aktive Webseite bleibt davon unbeeinflusst
- Das Grundmodell sind separate JavaScript-Dateien, die mit der Hauptdatei Nachrichten austauschen



# WEB WORKERS API - FUNKTIONSWEISE - 1

```
new Promise(function(resolve, reject) { ... });
```



Quelle

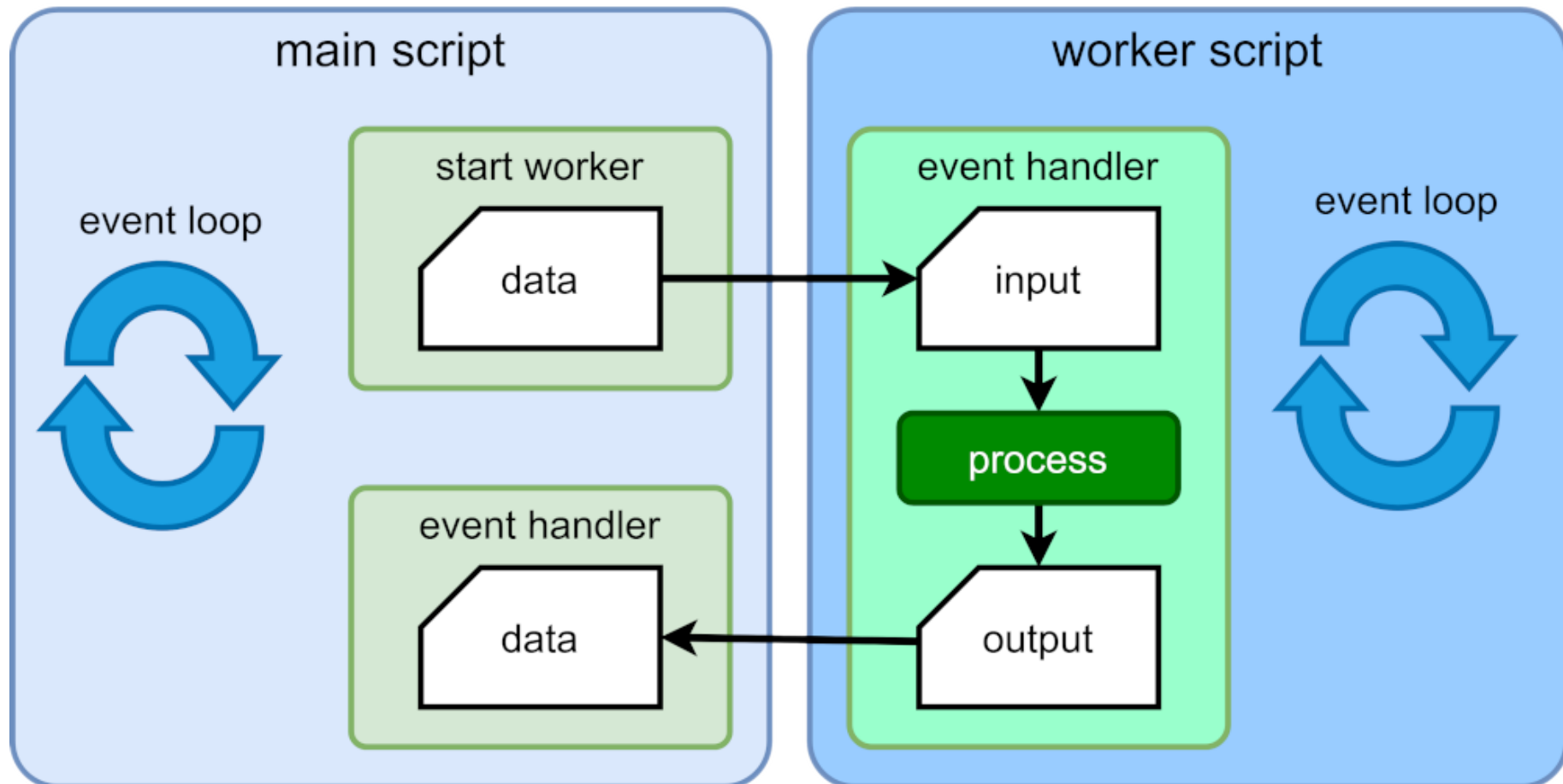
## WEB WORKERS API - FUNKTIONSWEISE - 2

- Nachrichten bestehen aus Strings oder aus JSON/XML-Objekten
- Andere Datenlasten sind derzeit nicht vorgesehen
- Elemente des Hauptdokuments wie HTML-Elemente, JavaScript-Code oder - Variable sind den Workers nicht zugänglich
- Fehlermeldungsroutrinen für die Worker müssen programmiert werden
- Worker können vom Hauptprogramm oder in sich selbst abgeschaltet werden

## WEB WORKERS API - SYNCHRONE UND ASYNCHRONE APIS

- Die HTML5-APIs liegen mitunter in einer asynchronen und in einer synchronen Version vor
- Bisher wurde mit der asynchronen Version gearbeitet, das heißt die APIs liefen in einem separaten Thread
- Die File API und die IndexedDB API gibt es aber z.B. auch als synchrone Versionen

# WEB WORKERS API - FUNKTIONSWEISE




Quelle

## WEB WORKERS API - DEDICATED UND SHARED WORKER


- Ein Dedicated Worker arbeitet nur mit der Hauptdatei zusammen, von der er angesprochen (und mit der er zusammen erstellt) wurde
- Ein Shared Worker kann mit mehreren Verbindungen zugleich arbeiten
- Zum Test eines Dedicated Worker werden mindestens drei Dateien gebraucht, für den eines Shared Worker mindestens fünf Dateien
- Laufzeitenvergleich

# WEB WORKERS API - SUPPORT

Web Workers  - LS

Usage % of all users Global 96.96%

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

Current aligned Usage relative Date relative Filtered All 

Chrome	Edge *	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
		3.1-3.2	2-3	10.1	6-9		3.2-4.3					2.1				
4-128	12-128	4-17.6	3.5-130	11.5-113	10		5-17.6	4-24		12-12.1		2.2-4.3				2.5
129	129	18.0	131	114	11	129	18.0	25	all	80	15.5	129	130	14.9	13.52	3.1
130-132		18.1-TP	132-134				18.1									


Quelle


# SHARED WEB WORKERS API - SUPPORT




Quelle


# DEDICATED WEB WORKERS API - SUPPORT

DedicatedWorkerGlobalScope API 

Usage % of all users  ?

Global 95.46%

Current aligned Usage relative Date relative Filtered All 

Chrome	Edge *	Safari	Firefox	Opera	IE  *	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
		3.1-3.2	2-3	10.1	6-9		3.2-4.3					2.1-4.3				
4-128	12-128	4-17.6	3.5-130	11.5-113	10		5-17.6	4-24		12-12.1		4.4-4.4.4				2.5
129	129	18.0	131	114	11	129	18.0	25	all	80	15.5	129	130	14.9	13.52	3.1
130-132		18.1-TP	132-134				18.1									

Quelle



## WEB WORKERS API - WEITERES

- W3C-Spezifikation: [Quelle](#)
- [Quelle](#)
- [Windows 98](#)
- [Webworker Live-PI Berechnung](#)
- [W3C Webworker Guide](#)
- [Angular Webworker](#)

## WEBGPU API

- Die WebGPU-API ermöglicht es Webentwicklern, den Grafikprozessor (GPU) des zugrunde liegenden Systems zu nutzen, um Hochleistungsberechnungen durchzuführen und komplexe Bilder zu zeichnen, die im Browser gerendert werden können
- WebGPU ist der Nachfolger von WebGL und bietet bessere Kompatibilität mit modernen GPUs, Unterstützung für allgemeine GPU-Berechnungen, schnellere Operationen und Zugang zu erweiterten GPU-Funktionen.

## WEBGPU API - BEISPIELE

- Firefox unterstützt WebGPU noch nicht, einzelne Funktionen sind in nightly nutzbar (2024)
- [KI-Netze und LLMs über WebGPU](#)
- [Webgpu compute](#)
- [Webgpu renderer](#)
- [WebGPU Samples](#)

## WEBXR DEVICE API

- WebXR ist eine Gruppe von Standards, die zusammen verwendet werden, um das Rendern von 3D-Szenen auf Hardware zu unterstützen
- Darstellung von virtueller Welten (Virtual Reality oder VR) oder für das Hinzufügen von grafischen Bildern zur realen Welt (Augmented Reality oder AR)
- Die WebXR-Geräte-API implementiert den Kern des WebXR-Standards
- Auswahl von Ausgabegeräten, Rendering auf dem ausgewählten Gerät mit der entsprechenden Bildrate und Bewegungsvektorenverwaltung
- Ein typisches XR-Gerät hat entweder 3 oder 6 Freiheitsgrade und kann mit oder ohne externen Positionssensor ausgestattet sein
- [Theorie zu WebXR](#)

## WEBXR DEVICE API - BEISPIELE

- Movements
- Inputs
- Targeting
- Performance
- Permissions

## ANDERE BROWSER APIS

- History API: [Quelle](#)
- MediaStream API (oder Stream API, siehe WebRTC API): [Quelle](#)
- Pointer Lock API: [Quelle](#)
- Page Visibility API: [Quelle](#)
- WebRTC API: [Quelle](#)
- Web Messaging API: [Quelle](#)
- Web Audio API: [Quelle](#)

## WEITERE QUELLEN

- [W3Schools](#)
- [Mozilla WebAPIs](#)
- [W3 standards](#)
- [W3 API standards](#)
- [OpenAPI](#)

## **ABSPANN**

Sechstes Level geschafft weitere Folgen!

Fragen und Feedback?