Web Performance
A Schrödingers Cat

Does perfect performance even exist?

Performance
Schrödinger's
Cat
SLO

Wie geht Performance

- Server Konfiguration
- Protokolle
- Kompression
- Lazy Loading & Loading Indicator
- SSR / SSG

Fertig 🥳

…. oder doch nicht?

TCP

Network

Caching

Tight mode

Prefetching

CDN

HTTP Technologie

MPA

Multiplexing

Web Vitals

SPA

Resource Prioritization

User hardware

Compression

Prerendering

Worker

SSG

BF-cache

Browser behaviour

Main Thread

Header

SSR

Preloading

Throttling

Adaptive Loading

Unbeeinflussbare Faktoren

Browser Verhalten

Netzwerk

User Hardware

Icons from Flaticon

Zu mir

🔢 23 Jahre alt

🧑‍💻 Fullstack Software Engineer & Product Owner

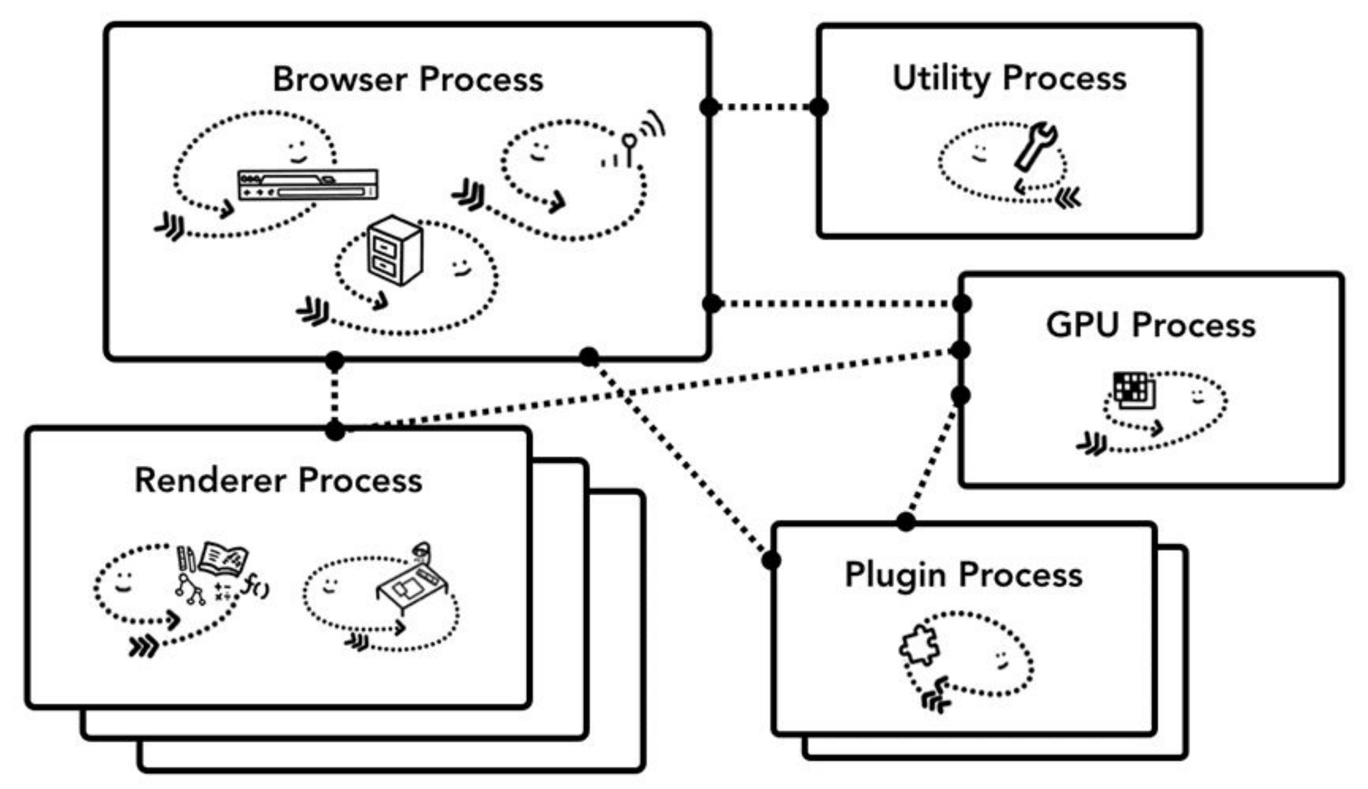
Dynatrace - Real User Monitoring

🚒 Enthusiastischer Feuerwehrmann

📃 Survey Time 📃

Grundlegendes

Browser Architektur



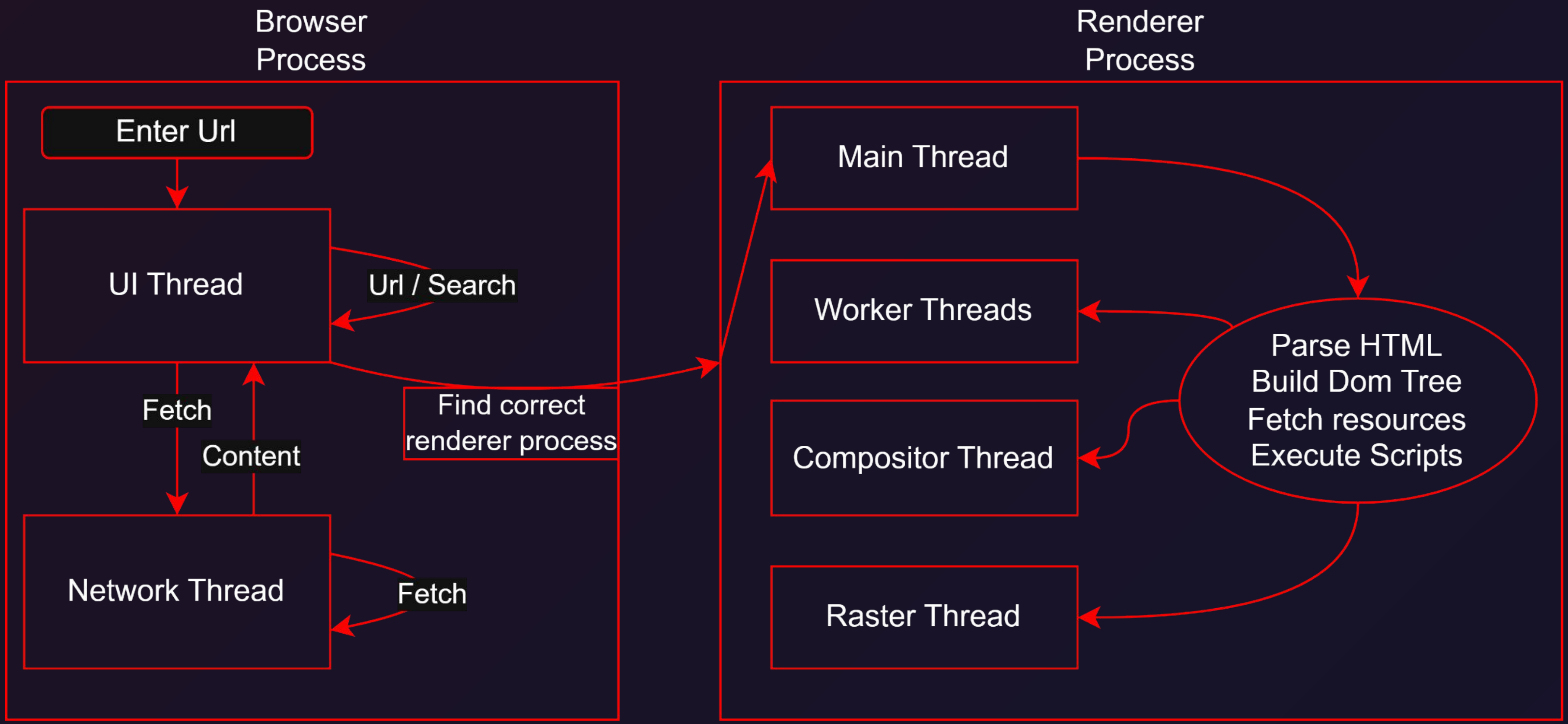
Browser Process

Utility Process

GPU Process

Renderer Process

Plugin Process



Browser Process

Enter Url

UI Thread

Url / Search

Fetch

Content

Network Thread

Fetch

Find correct renderer process

Renderer Process

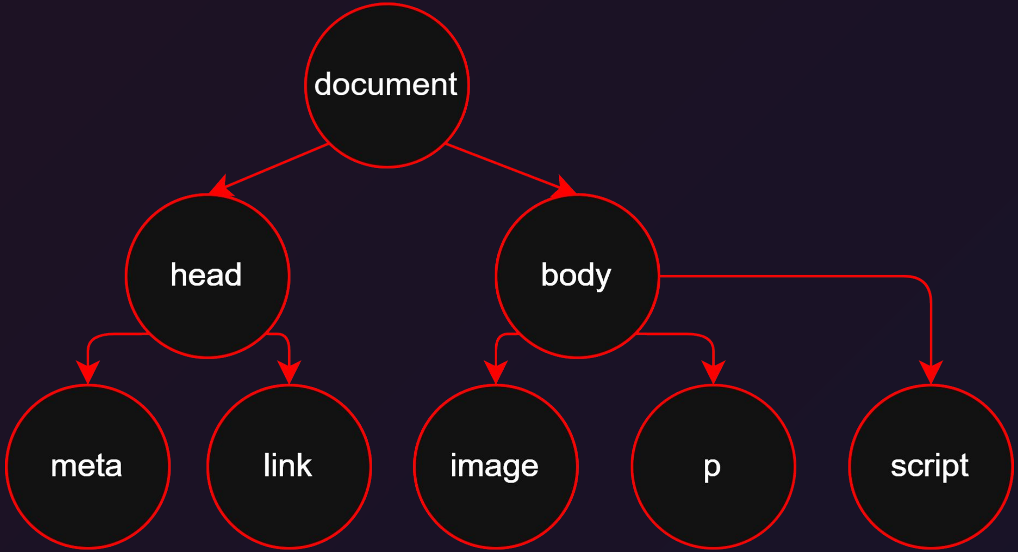
Main Thread

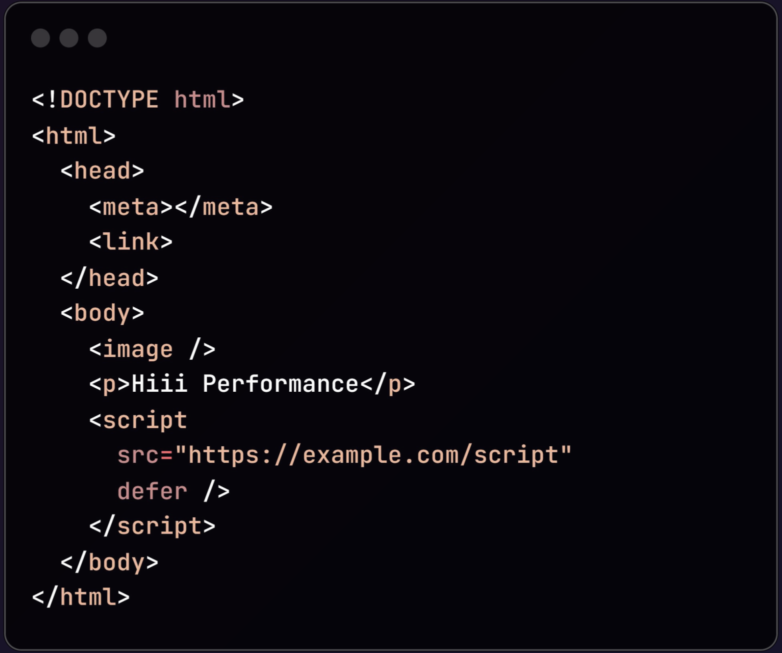
Worker Threads

Compositor Thread

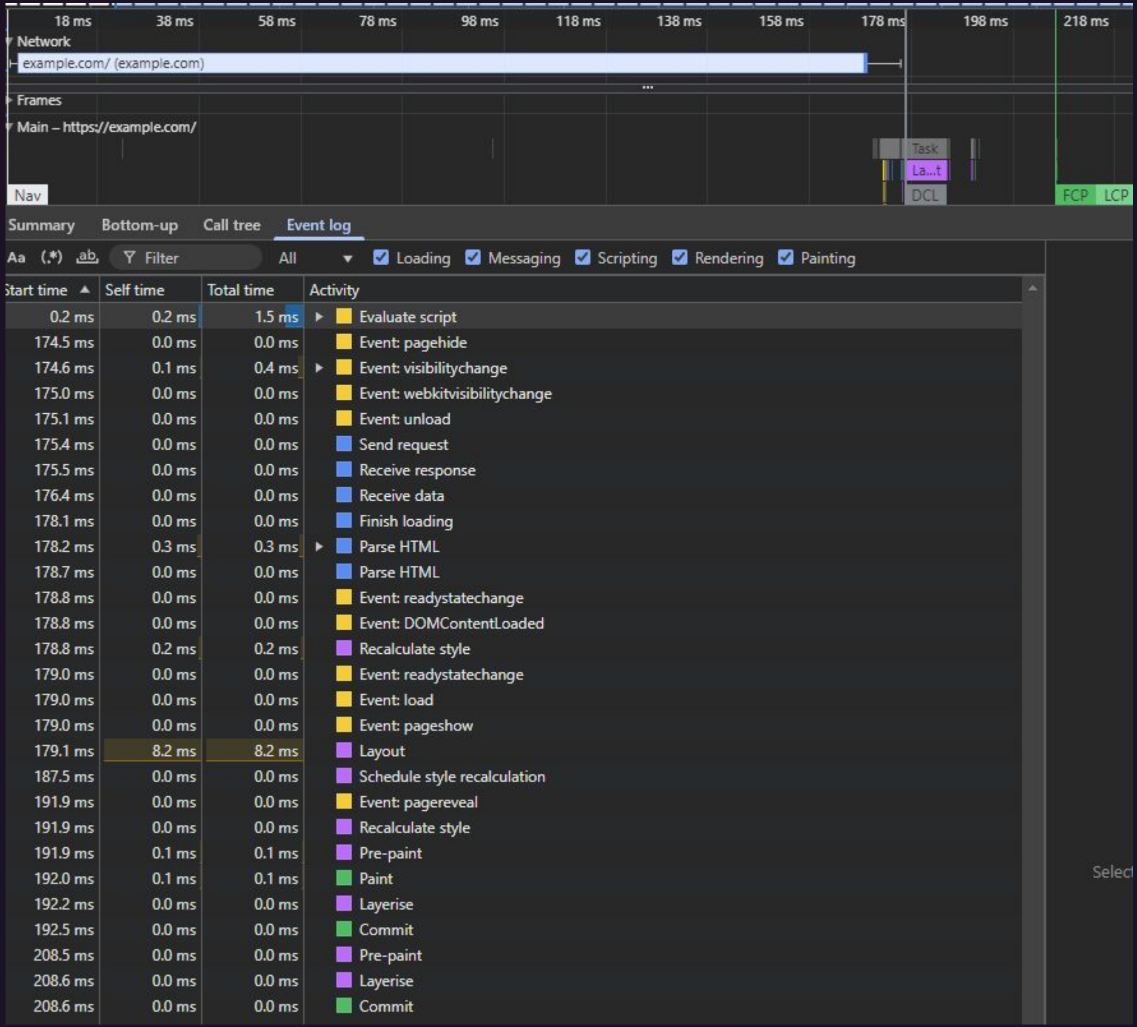
Raster Thread

Parse HTML
Build Dom Tree
Fetch resources
Execute Scripts


document
head
body
meta
link
image
p
script

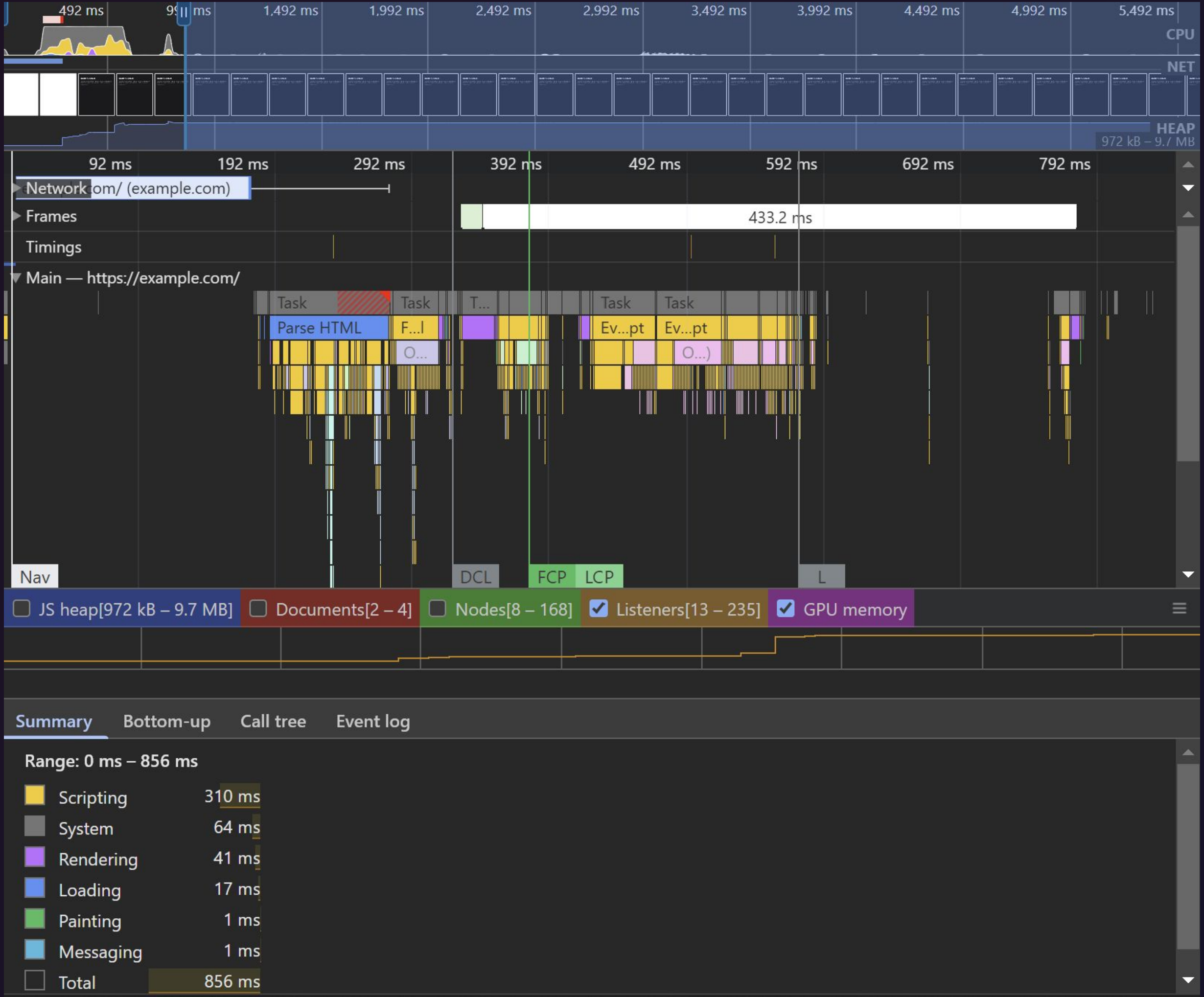


```
<!DOCTYPE html>
<html>
  <head>
    <meta></meta>
    <link>
  </head>
  <body>
    <image />
    <p>Hiii Performance</p>
    <script
      src="https://example.com/script"
      defer />
    </script>
  </body>
</html>
```


18 ms
38 ms
58 ms
78 ms
98 ms
118 ms
138 ms
158 ms
178 ms
198 ms
218 ms
Network
example.com/ (example.com)
Frames
Main – https://example.com/
Task
La...t
Nav
DCL
FCP
LCP
Summary
Bottom-up
Call tree
Event log
Aa
(.*)
ab
Filter
All
Selec

☑ Loading ☑ Messaging ☑ Scripting ☑ Rendering ☑ Painting

Start time ▲	Self time	Total time	Activity
0.2 ms	0.2 ms	1.5 ms	▶ 🟨 Evaluate script
174.5 ms	0.0 ms	0.0 ms	🟨 Event: pagehide
174.6 ms	0.1 ms	0.4 ms	▶ 🟨 Event: visibilitychange
175.0 ms	0.0 ms	0.0 ms	🟨 Event: webkitvisibilitychange
175.1 ms	0.0 ms	0.0 ms	🟨 Event: unload
175.4 ms	0.0 ms	0.0 ms	🟦 Send request
175.5 ms	0.0 ms	0.0 ms	🟦 Receive response
176.4 ms	0.0 ms	0.0 ms	🟦 Receive data
178.1 ms	0.0 ms	0.0 ms	🟦 Finish loading
178.2 ms	0.3 ms	0.3 ms	▶ 🟦 Parse HTML
178.7 ms	0.0 ms	0.0 ms	🟦 Parse HTML
178.8 ms	0.0 ms	0.0 ms	🟨 Event: readystatechange
178.8 ms	0.0 ms	0.0 ms	🟨 Event: DOMContentLoaded
178.8 ms	0.2 ms	0.2 ms	🟪 Recalculate style
179.0 ms	0.0 ms	0.0 ms	🟨 Event: readystatechange
179.0 ms	0.0 ms	0.0 ms	🟨 Event: load
179.0 ms	0.0 ms	0.0 ms	🟨 Event: pageshow
179.1 ms	8.2 ms	8.2 ms	🟪 Layout
187.5 ms	0.0 ms	0.0 ms	🟪 Schedule style recalculation
191.9 ms	0.0 ms	0.0 ms	🟨 Event: pagereveal
191.9 ms	0.0 ms	0.0 ms	🟪 Recalculate style
191.9 ms	0.1 ms	0.1 ms	🟪 Pre-paint
192.0 ms	0.1 ms	0.1 ms	🟩 Paint
192.2 ms	0.0 ms	0.0 ms	🟪 Layerise
192.5 ms	0.0 ms	0.0 ms	🟩 Commit
208.5 ms	0.0 ms	0.0 ms	🟪 Pre-paint
208.6 ms	0.0 ms	0.0 ms	🟪 Layerise
208.6 ms	0.0 ms	0.0 ms	🟩 Commit


492 ms
9|| ms
1,492 ms
1,992 ms
2,492 ms
2,992 ms
3,492 ms
3,992 ms
4,492 ms
4,992 ms
5,492 ms
CPU
NET
HEAP
972 kB – 9.7 MB
92 ms
192 ms
292 ms
392 ms
492 ms
592 ms
692 ms
792 ms
Network om/ (example.com)
Frames
433.2 ms
Timings
Main — https://example.com/
Task
Task
T...
Task
Task
Parse HTML
F...l
Ev...pt
Ev...pt
O...
O...)
Nav
DCL
FCP
LCP
L
Summary
Bottom-up
Call tree
Event log

☐ JS heap[972 kB – 9.7 MB] ☐ Documents[2 – 4] ☐ Nodes[8 – 168] ☑ Listeners[13 – 235] ☑ GPU memory

Range: 0 ms – 856 ms

🟨	Scripting	310 ms
⬛	System	64 ms
🟪	Rendering	41 ms
🟦	Loading	17 ms
🟩	Painting	1 ms
🟦	Messaging	1 ms
☐	Total	856 ms

Was ist Performance?

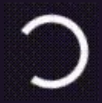
Web performance ist

- die objektive Messung
- und wahrgenommene Nutzererfahrung

einer Webseite/-applikation

Perceived Performance

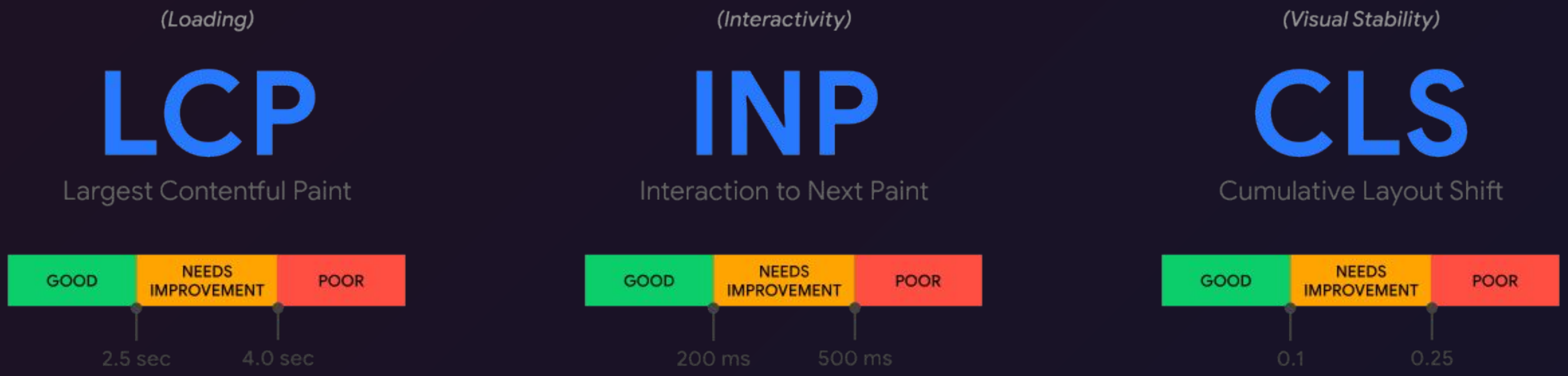
“Wie schnell fühlt sich die Seite an?”

- Lädt die Seite schnell und zeigt mir dann an das sich etwas tut oder dauert es ewig bis überhaupt mal etwas kommt.
- Verkürzen der Ladezeiten mit Ladeanzeigen
    - Bildern (siehe Videospiele) und Texten (Tipps/Witze)
    - 

Objektive Messung - Einflüsse

- Benutzerseite
  - Bandbreite, Hardware (CPU/GPU)
  - Code Architektur
  - Caching
- Serverseitig
  - Bandbreite/Durchsatz, IO
  - Garbage Collection
  - Code Architektur
  - Datenanbindung (Datenbank, andere Quellen)

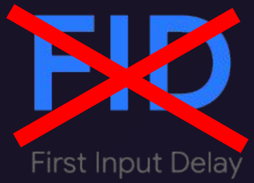
Core Web Vitals


(Loading)
LCP
Largest Contentful Paint
GOOD
NEEDS IMPROVEMENT
POOR
2.5 sec
4.0 sec
(Interactivity)
INP
Interaction to Next Paint
GOOD
NEEDS IMPROVEMENT
POOR
200 ms
500 ms
(Visual Stability)
CLS
Cumulative Layout Shift
GOOD
NEEDS IMPROVEMENT
POOR
0.1
0.25

Wann ist der größte Abschnitt (im sichtbaren Bereich) seit dem Ladebeginn der Seite fertig?

Durchschnittliche Dauer zwischen Interaktion und dem nächsten Paint

Wie stark verändert sich der Inhalt nach dem Laden


FID
First Input Delay

Ersetzt durch INP im März 24

Other Metrics

- TTFB - Time to first byte: Zeit bis zum ersten Byte
- Visually complete: Wie lange dauert es bis sich im sichtbaren Bereich nichts mehr ändert
- Speed index: VC - Nimmt auch die Größe des nicht fertigen Bereichs als Gewichtung
- DOMInteractive: Zeit, bis der Benutzer mit der Seite interagieren kann
- Total blocking time: Zeit, die die Seite nicht interaktiv war
- FCP: First contentful paint - Wann war der erste Paint

… many more

Wie berechnet man objektive Performance

- Lokal
  - Browser Devtools: Web Vitals, Waterfall, CPU/Netzwerk drosselung
- Extern
  - Manuell
    - https://pagespeed.web.dev/ (Google)
    - https://webpagetest.org (Catchpoint)
  - Automatisierte Techniken
    - RUM
    - Synthetic

Real User Monitoring

- Performance, Errors
- Page Usage/User journeys
  - Session Replay
  - Rage Clicks
  - Neue/rückkehrende Benutzer
  - Woher kamen die Benutzer
- Welche Ressourcen werden angefordert
- Fullstack Monitoring:
  - Verlinkung mit Backend Daten (Service Level)

Synthetic

- Agenten, die eine HTTP-Anfrage durchführen oder einen Headless Browser verwenden
  - Antwort verifizieren
  - Visuelle kontrolle
- Konfigurierbar
  - Ort
  - Interval
  - Bandbreite
- Verfügbarkeit
- Alarmierung bei Problemen (baselining)

Performance Verbesserungsmöglichkeiten

(ein paar zumindest 😀)

Bilder

Grundlagen

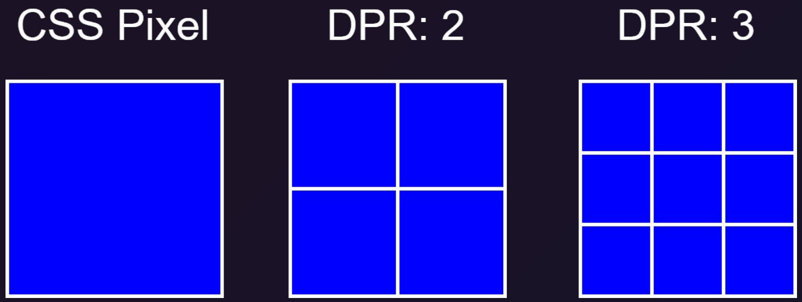
- LCP-Image nicht lazy laden

```html
<head>
    <link rel="preload" href="/hero.png" as="image" />
</head>
<img src="/hero.png" fetchpriority="high" />
```

- Verschiedene Bildgrößen anbieten (aber auch nicht zu viele)
- Verschiedene Bildformate

DPR

- Unterschied zwischen CSS & Physischen Pixel
- “Device pixel ratio” (DPR) bedenken
    - 500 CSS pixels * 2 DPR = 1000px image



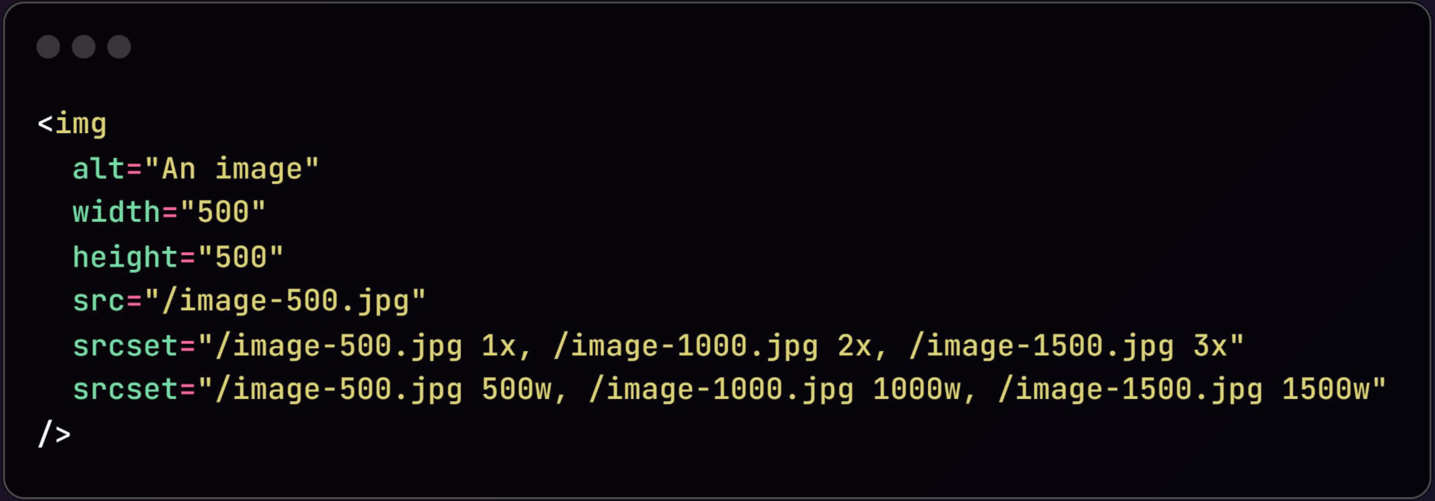
CSS Pixel

DPR: 2

DPR: 3

Bild Größen

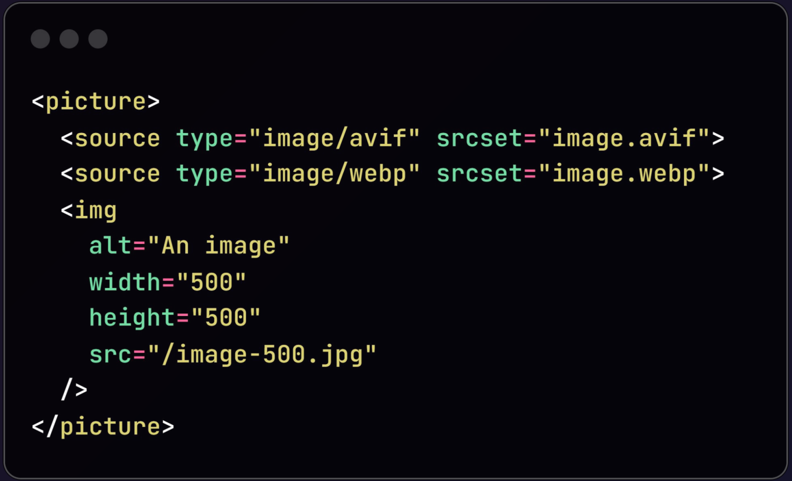
- `srcsets` benutzen



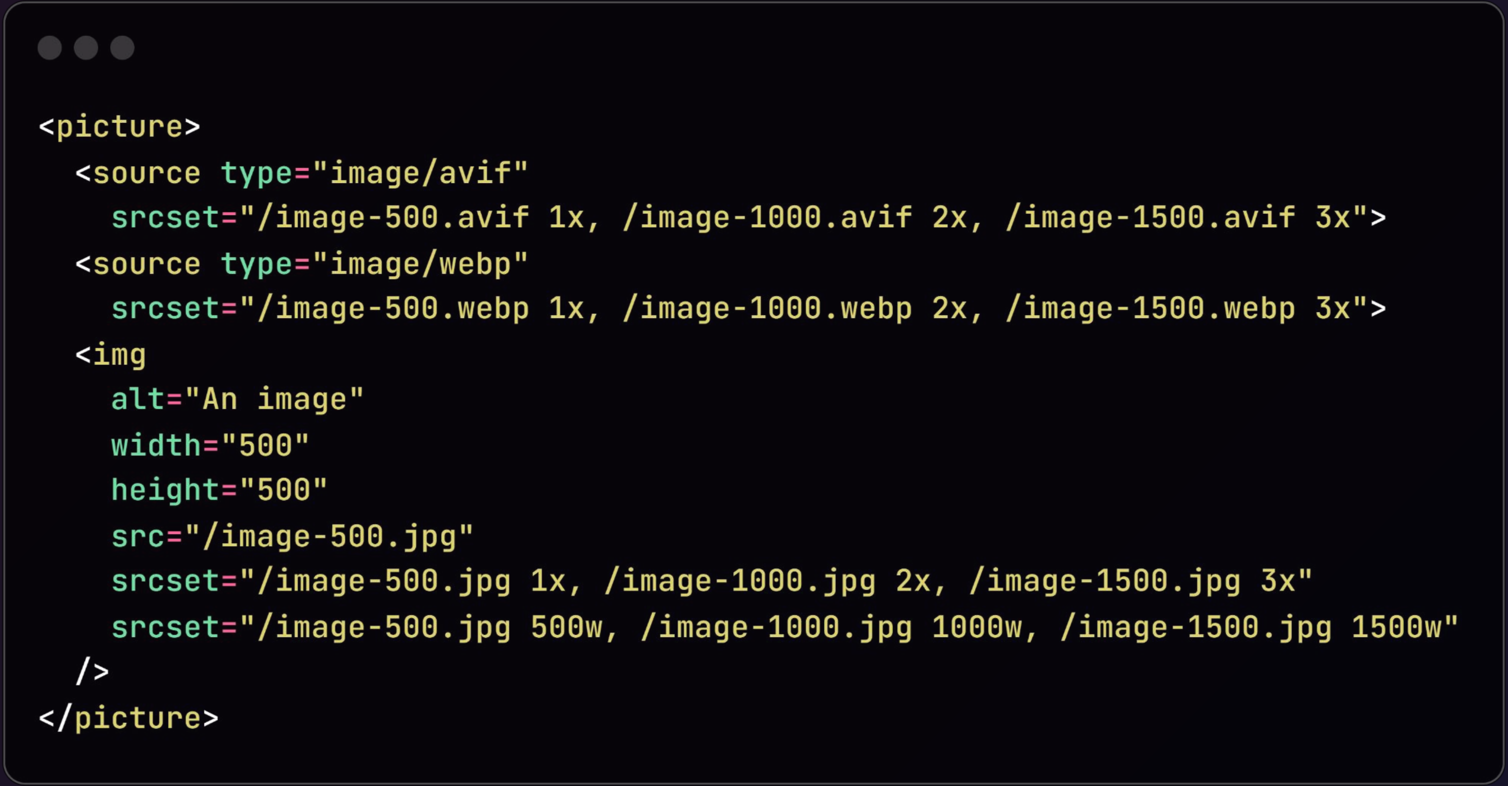
```
<img
  alt="An image"
  width="500"
  height="500"
  src="/image-500.jpg"
  srcset="/image-500.jpg 1x, /image-1000.jpg 2x, /image-1500.jpg 3x"
  srcset="/image-500.jpg 500w, /image-1000.jpg 1000w, /image-1500.jpg 1500w"
/>
```

Bild Formate

- Recherchieren Sie, welche Formate für Ihre Bedürfnisse am besten geeignet sind
- Kompressions benutzen (lossless or lossy)
- Serverseitige Bestimmung des Formats
- Zu großes HTML



```
<picture>
  <source type="image/avif" srcset="image.avif">
  <source type="image/webp" srcset="image.webp">
  <img
    alt="An image"
    width="500"
    height="500"
    src="/image-500.jpg"
  />
</picture>
```

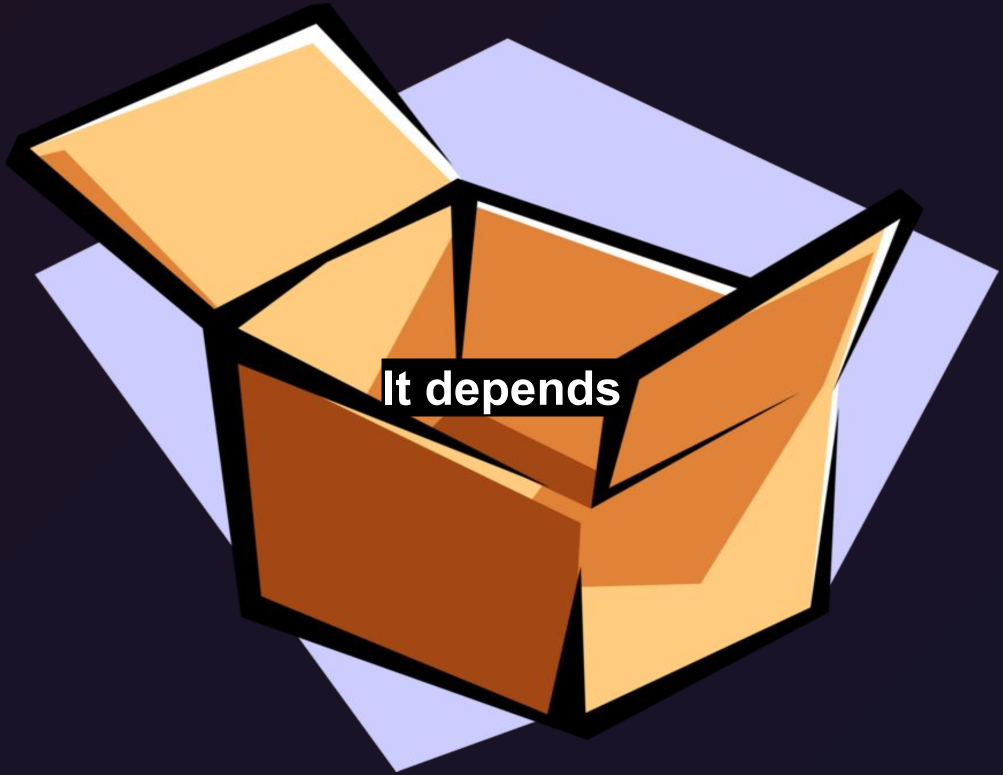


```
<picture>
  <source type="image/avif"
    srcset="/image-500.avif 1x, /image-1000.avif 2x, /image-1500.avif 3x">
  <source type="image/webp"
    srcset="/image-500.webp 1x, /image-1000.webp 2x, /image-1500.webp 3x">
  <img
    alt="An image"
    width="500"
    height="500"
    src="/image-500.jpg"
    srcset="/image-500.jpg 1x, /image-1000.jpg 2x, /image-1500.jpg 3x"
    srcset="/image-500.jpg 500w, /image-1000.jpg 1000w, /image-1500.jpg 1500w"
  />
</picture>
```

465 bytes (UTF-8)

für ein Bild

Bilder



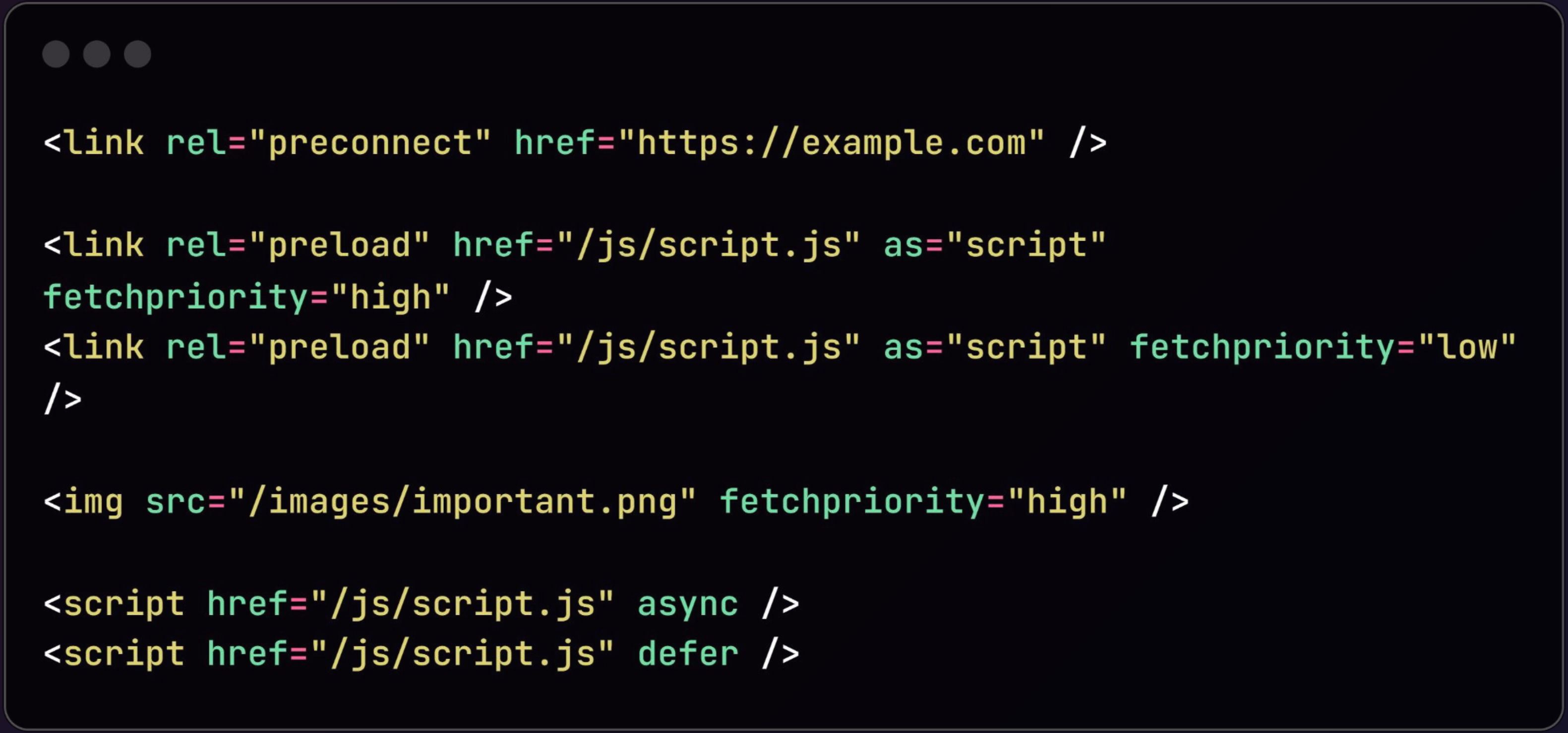
It depends

Fetch Priorities

Fetch priorities

- Zeigt dem Browser wichtige ressourcen (CSS, JS, Bilder…)
- Browser entscheiden grundsätzlich selbst was wichtig ist. Aber man kann versuchen nachzuhelfen

Wie konfiguriert man Fetch Priorities

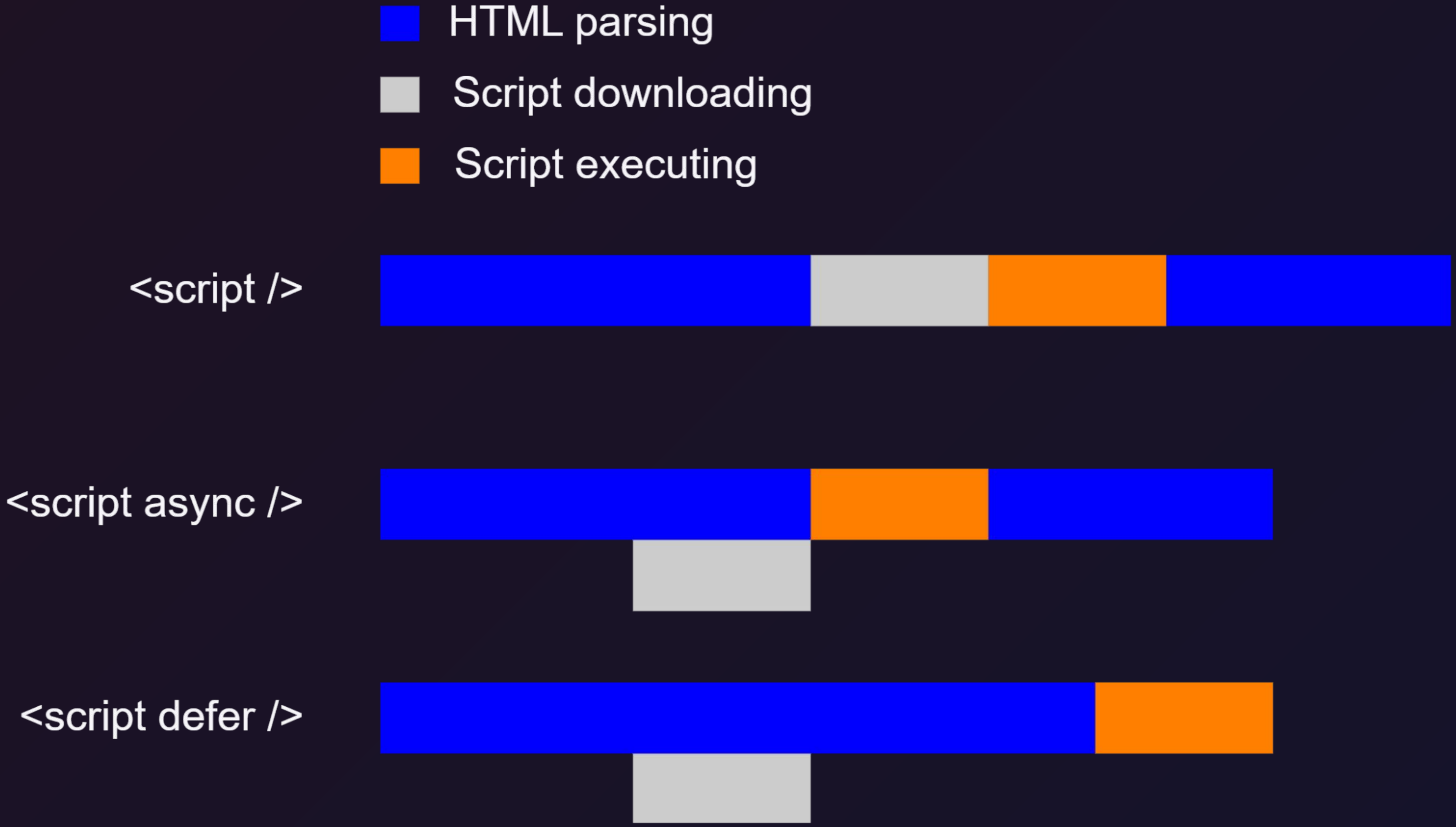


```
<link rel="preconnect" href="https://example.com" />

<link rel="preload" href="/js/script.js" as="script"
fetchpriority="high" />
<link rel="preload" href="/js/script.js" as="script" fetchpriority="low"
/>

<img src="/images/important.png" fetchpriority="high" />

<script href="/js/script.js" async />
<script href="/js/script.js" defer />
```



🟦 HTML parsing

⬜ Script downloading

🟧 Script executing

`<script />`

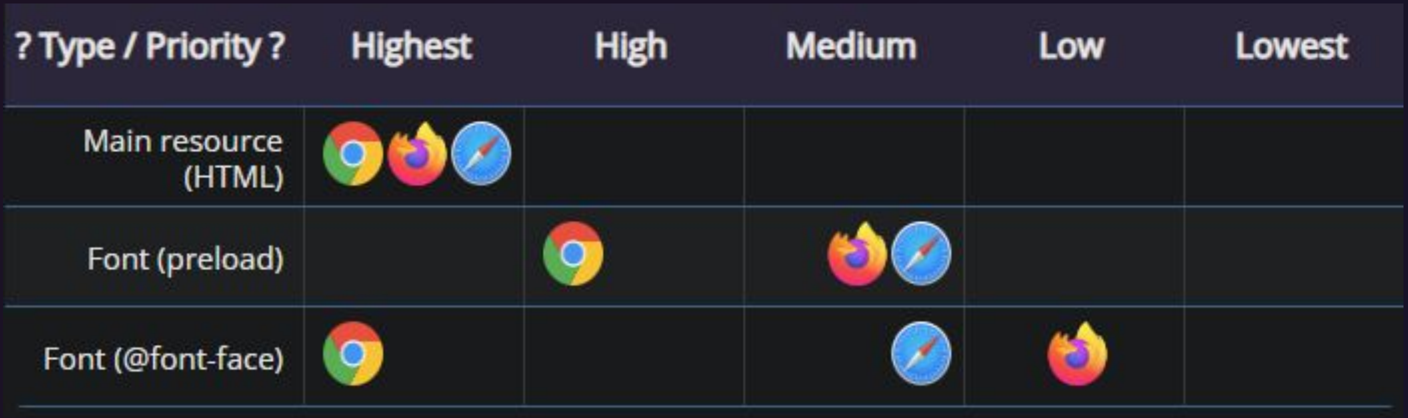
`<script async />`

`<script defer />`

Das heißt:
Browsers verhalten sich bei Ressourcen alle gleich…

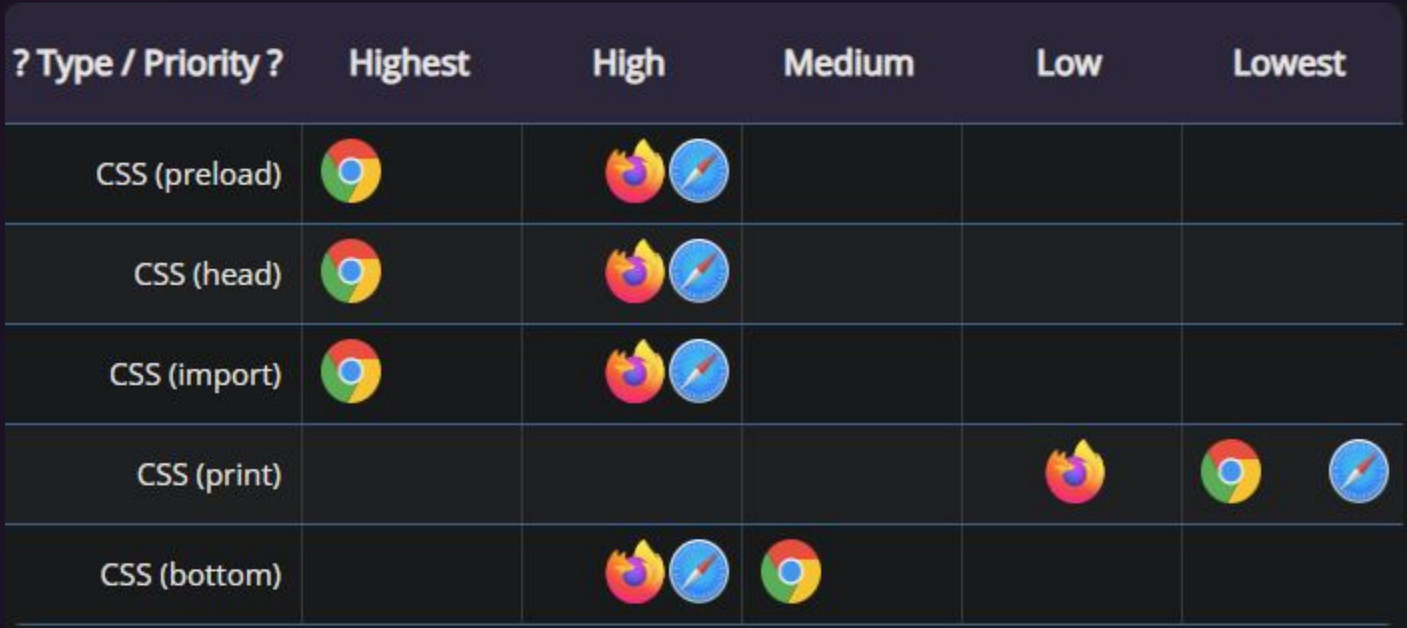
oder?🤔

Main Resource & Font



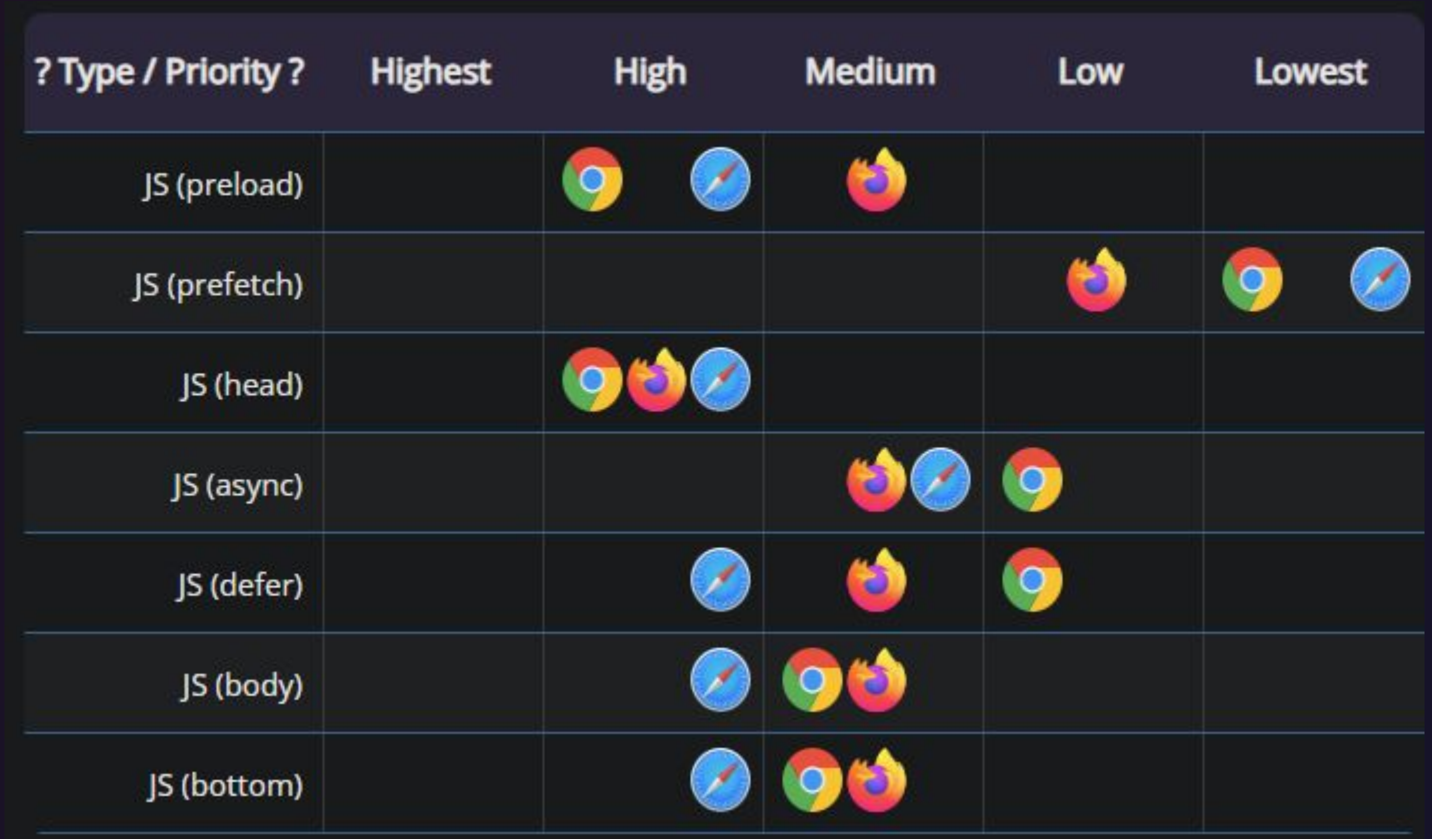
? Type / Priority ?	Highest	High	Medium	Low	Lowest
Main resource (HTML)	Chrome Firefox Safari				
Font (preload)		Chrome	Firefox Safari		
Font (@font-face)	Chrome		Safari	Firefox	

CSS



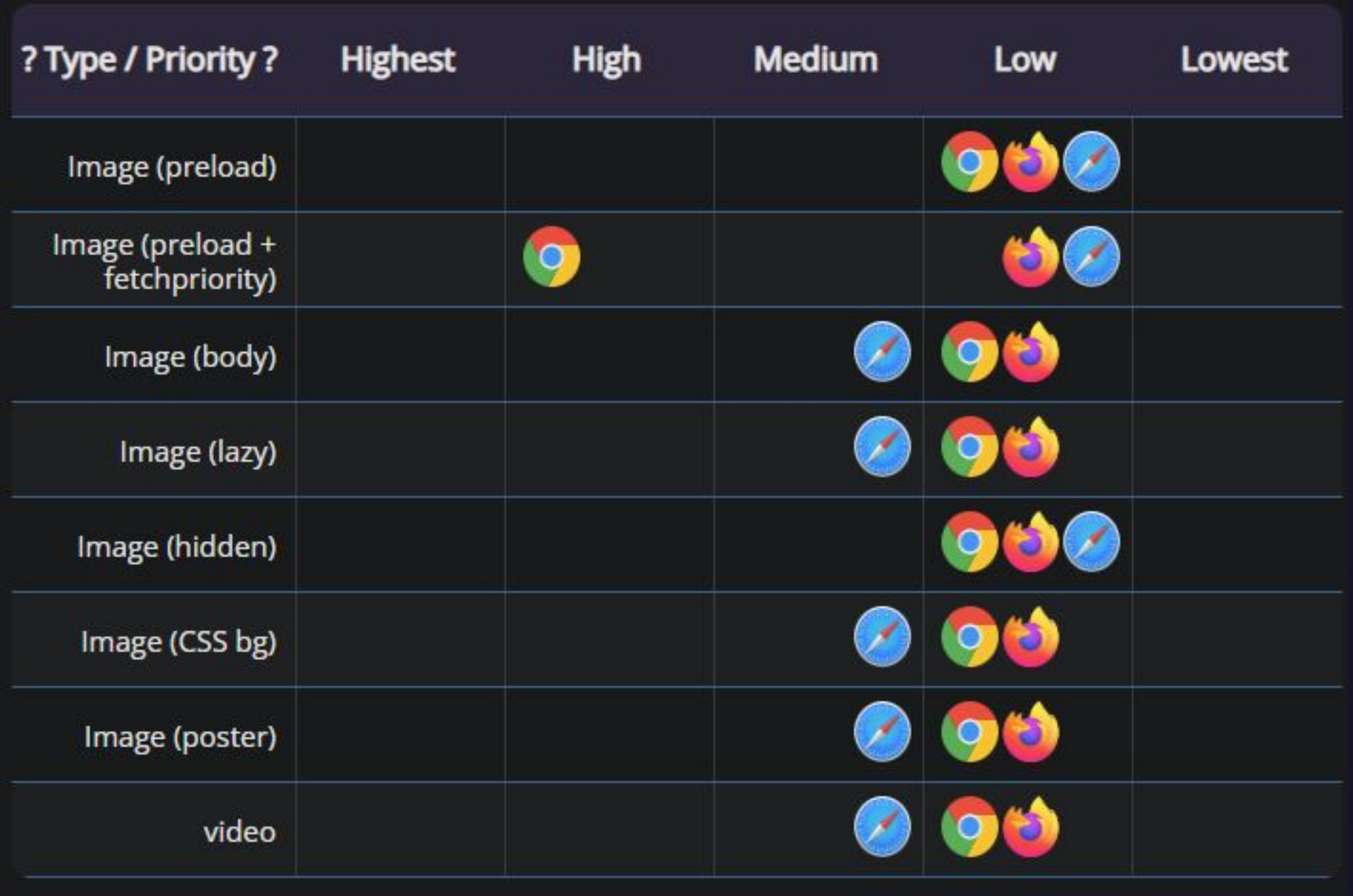
? Type / Priority ?	Highest	High	Medium	Low	Lowest
CSS (preload)	Chrome	Firefox Safari			
CSS (head)	Chrome	Firefox Safari			
CSS (import)	Chrome	Firefox Safari			
CSS (print)				Firefox	Chrome Safari
CSS (bottom)		Firefox Safari	Chrome		

JS



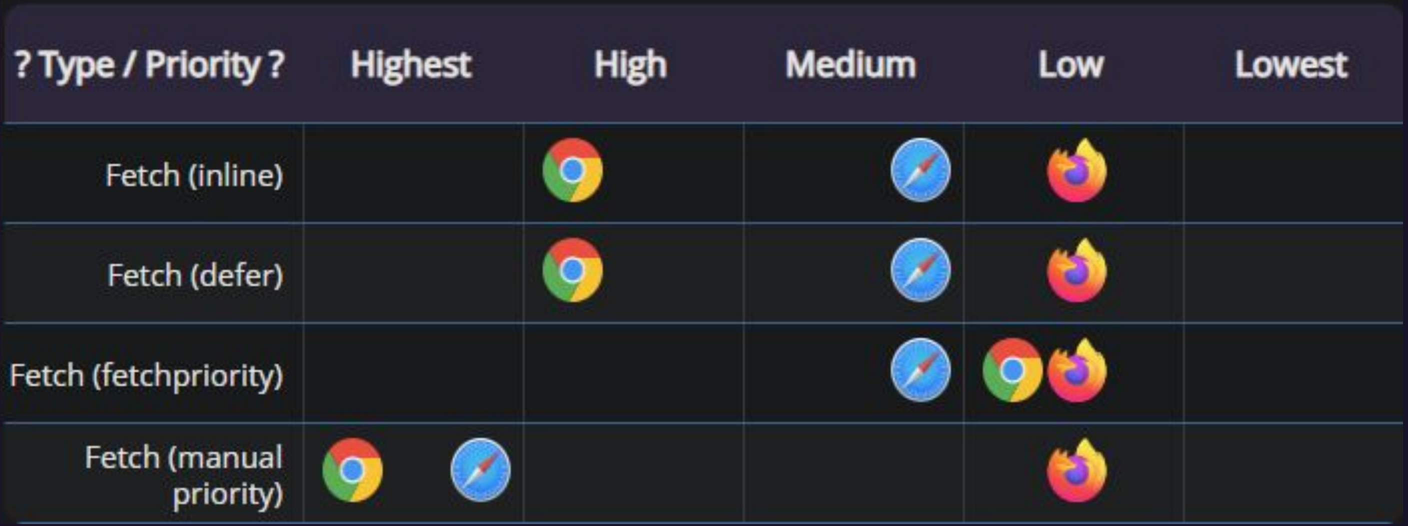
? Type / Priority ?	Highest	High	Medium	Low	Lowest
JS (preload)		Chrome Safari	Firefox		
JS (prefetch)				Firefox	Chrome Safari
JS (head)		Chrome Firefox Safari			
JS (async)			Firefox Safari	Chrome	
JS (defer)		Safari	Firefox	Chrome	
JS (body)		Safari	Chrome Firefox		
JS (bottom)		Safari	Chrome Firefox		

Image



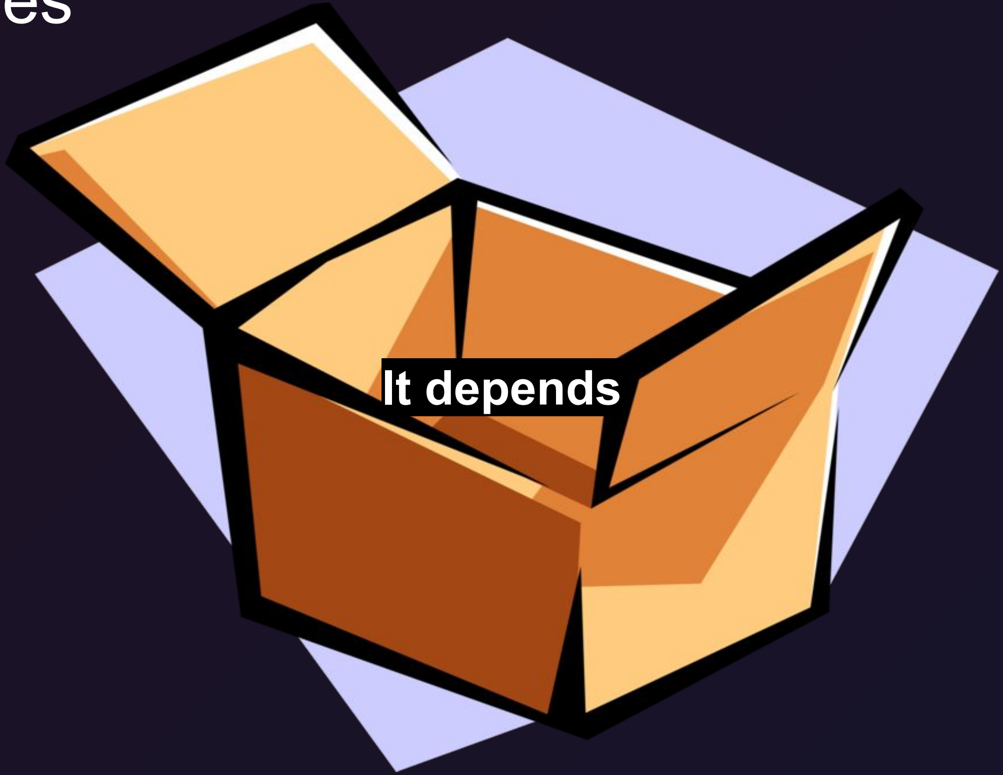
? Type / Priority ?	Highest	High	Medium	Low	Lowest
Image (preload)				◉ 🦊 🧭	
Image (preload + fetchpriority)		◉		🦊 🧭	
Image (body)			🧭	◉ 🦊	
Image (lazy)			🧭	◉ 🦊	
Image (hidden)				◉ 🦊 🧭	
Image (CSS bg)			🧭	◉ 🦊	
Image (poster)			🧭	◉ 🦊	
video			🧭	◉ 🦊	

Fetch



? Type / Priority ?	Highest	High	Medium	Low	Lowest
Fetch (inline)		●	●	●	
Fetch (defer)		●	●	●	
Fetch (fetchpriority)			●	● ●	
Fetch (manual priority)	● ●			●	

Fetch Priorities



It depends

Main Thread & Long Tasks

Main Thread & Long tasks

- JS ist Single Threaded
- Aufgaben
  - Skript Ausführung
  - Inputs/Events verarbeiten
  - Paints/Layout-Berechnung
- Frame duration: 1s (1000) / 60Hz = 16ms
- Long Tasks: Blockieren Main Thread >= 50ms (=> 3 frames)

Long task Verbesserungen

Tasks unterbrechen:

- *setTimeout*

```
return new Promise(resolve => {
  setTimeout(resolve, 0);
});
```

- Scheduler Api (eingeschränkte Verfügbarkeit)

Auslagern:

- Web workers
  - Komplexer, wann haben sie Sinn?

Web Worker

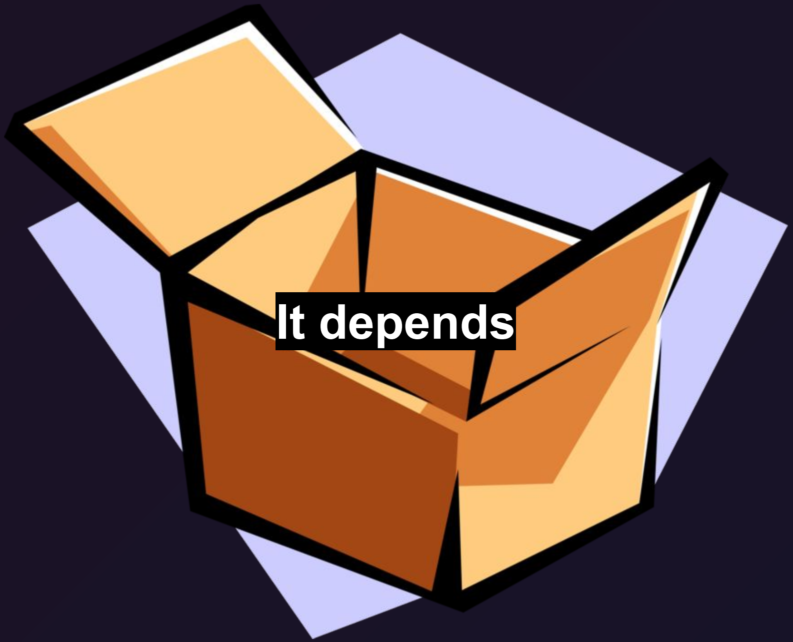
- Tasks in einem separaten Thread laufen lassen
- Typen:
  - Workers: führen ein Skript aus
    - lange/große Berechnungen auslagern (z.B. Rendering)
  - SharedWorkers
    - können von mehreren Skripten ausgeführt werden, solang nur eine Domain verwendet wird
    - Laufen über einen Port
  - ServiceWorker
    - Proxy Server (Browser ⇔ Netzwerk)
    - => Siehe PWAs
    - => Offline usage, Hintergrund aktualisierungen, Netzwerkanfragen intercepten…

Operation	
Instantiation of a web worker	40 ms
Instantiation of a broadcast channel	1 ms
Communication latency with postMessage	0.5 ms
Communication latency with broadcast channel	1.5 ms
Communication speed with postMessage	80 kB/ms
Communication speed with broadcast channel	12 kB/ms
Maximum message size with postMessage	1300 kB (16ms)
Maximum message size with broadcast channel	200 kB (16ms)

Worker Beispiele (main thread => worker)

- JSON parsing
    - 15 kb: 30-50ms => 5-10ms
    - 1.5 mb: 40-50ms => 15-20ms
    - 15 mb: 90-110ms => 70-95ms
- Arrays sortieren
    - 100k: 10ms => 1ms
    - 1m: 200ms => 10ms
    - 10m: 2.s => 100ms
- Bildmanipulation
- Andere komplexe Kalkulationen

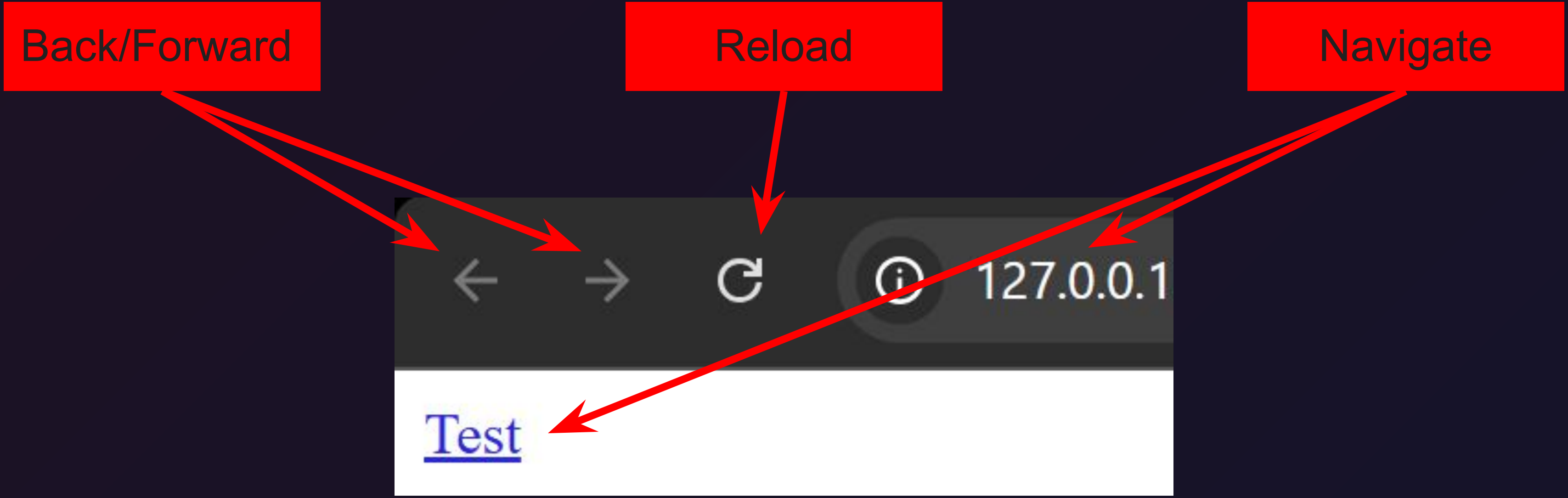
Main Thread & Workers



It depends

Back Forward Cache

Navigation types



Back/Forward

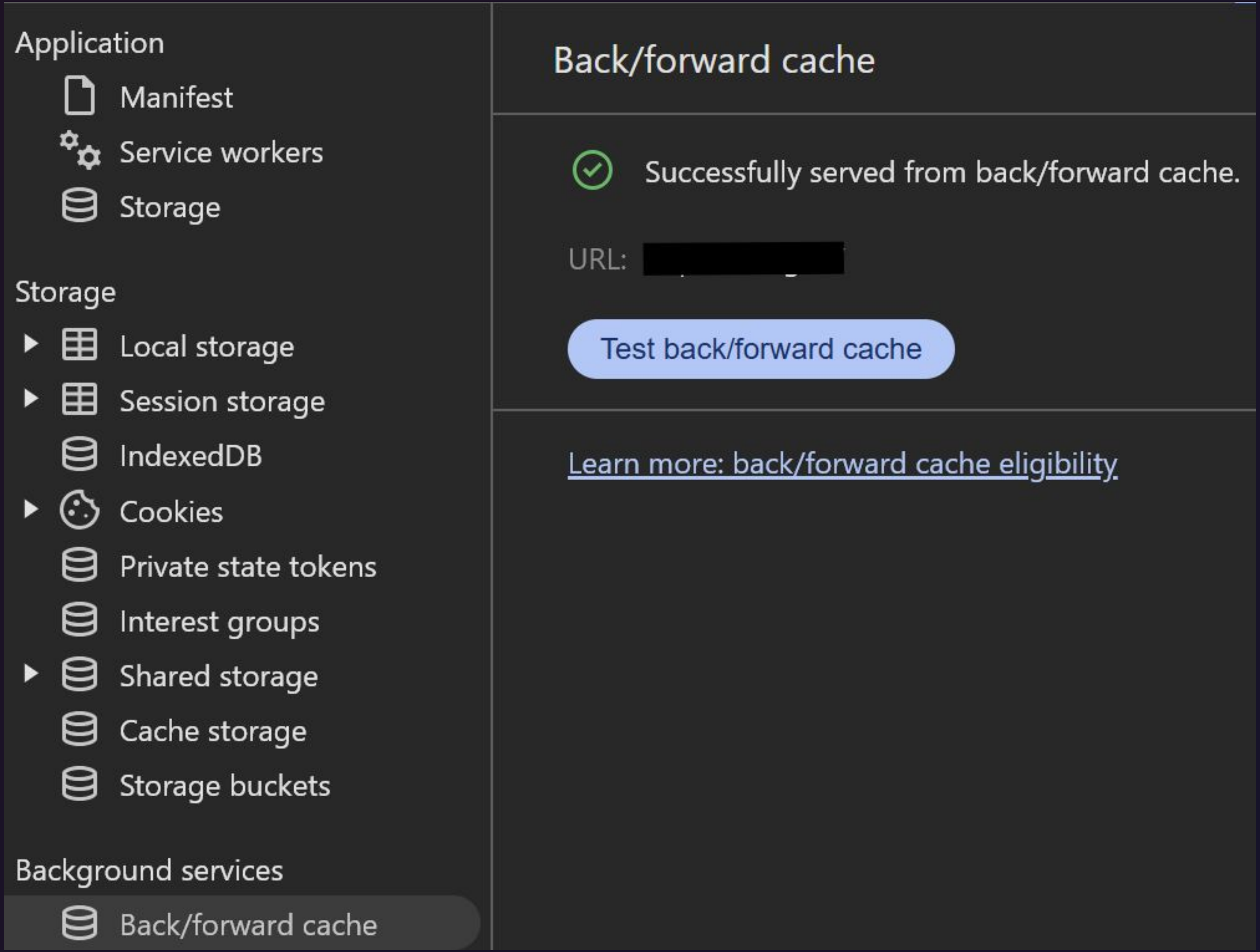
Reload

Navigate

127.0.0.1
Test

BF Cache

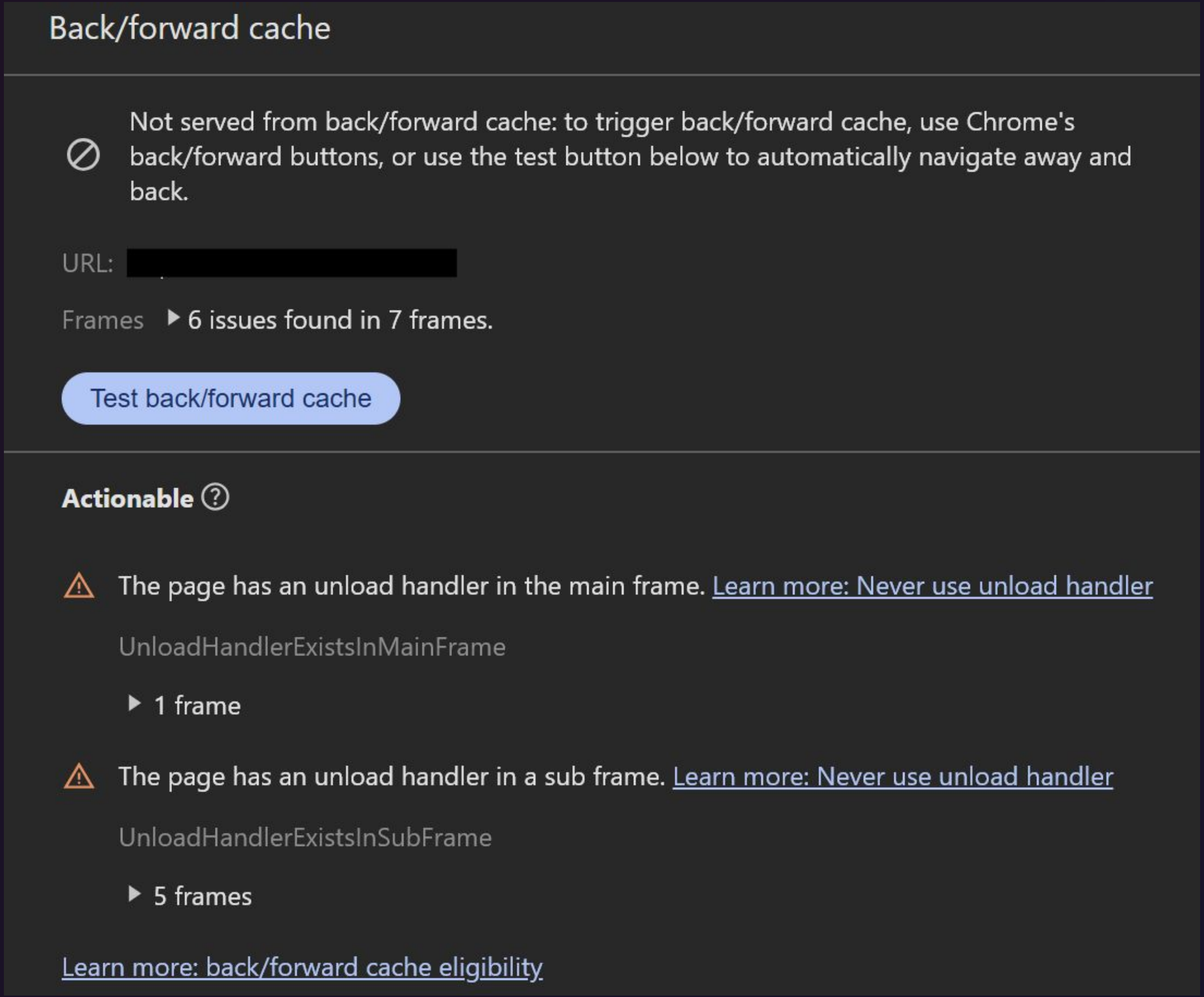
- Seiten cachen
- Back/Forward Navigation => Seite sofort darstellen


Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Storage buckets
Background services
Back/forward cache
Test back/forward cache
Learn more: back/forward cache eligibility

Back/forward cache

✓ Successfully served from back/forward cache.

URL: [redacted]



⊘ Not served from back/forward cache: to trigger back/forward cache, use Chrome's back/forward buttons, or use the test button below to automatically navigate away and back.

URL: [redacted]

Frames ▸ 6 issues found in 7 frames.

Actionable ⓘ

⚠ The page has an unload handler in the main frame. Learn more: Never use unload handler

UnloadHandlerExistsInMainFrame

▸ 1 frame

⚠ The page has an unload handler in a sub frame. Learn more: Never use unload handler

UnloadHandlerExistsInSubFrame

▸ 5 frames

Learn more: back/forward cache eligibility

Back/Forward Cache Probleme

- Mögliche Probleme
  - Header: `Cache control: no-store`
  - Javascript: `unload` event handler
- Serverseitige Lösung:
  - Header: `Permission Policy: unload=()`
  - Unload event => zu `pagehide` Event wechseln

Abschluss

Abschließend

- Es gibt immer Raum für Verbesserung
- Perfekte Performance existiert nicht wirklich
  - jedoch kann man viel verbessern
- “Ich weiß, dass ich nichts (wenig) weiß”

Generelle Empfehlungen

- Web-Performance zum Bestandteil des Entwicklungszyklus machen
- Reguläre checks (manuell und/oder automatisiert)
- Am neuesten Stand der Zeit bleiben

Danke

Die Katze ist aus der Box