TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Dr. Lucas Vogel
Fakultät Informatik | Chair of Distributed and Networked Systems | TU Dresden

Lecture Internet and Web Applications
The Fast Web: Foundations of Performant Websites

TECHNISCHE UNIVERSITÄT DRESDEN

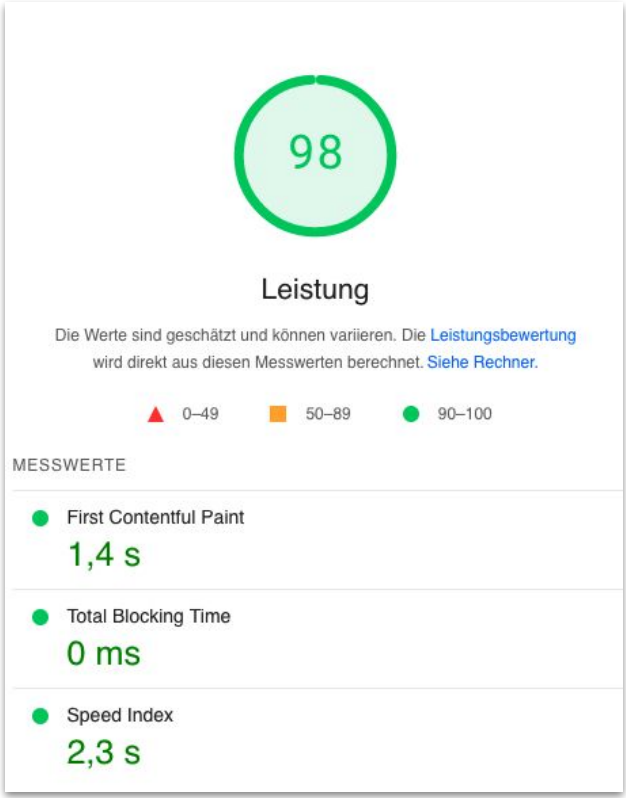
DRESDEN concept

What happens in this lecture?

Web development and performance

Structure of this lecture

- **1: Introduction: current state of web applications**
  - Why we need high-performance web apps
- **2: Basics about websites and measurements**
  - Overview of CSS, JavaScript, and HTML
  - Bundling
  - Measurement types
- **3: Optimization strategies**
  - Types and relevance
- **4: Streaming content**
- **Summary**



Source: https://pagespeed.web.dev, 31.05.2024

Who here had this problem recently?

- Network timeouts on a train?
- Phone carrier dropping service due to faulty software update?
- Reduced network speed due to bandwidth limits?
- Spotty cell service at a large event?

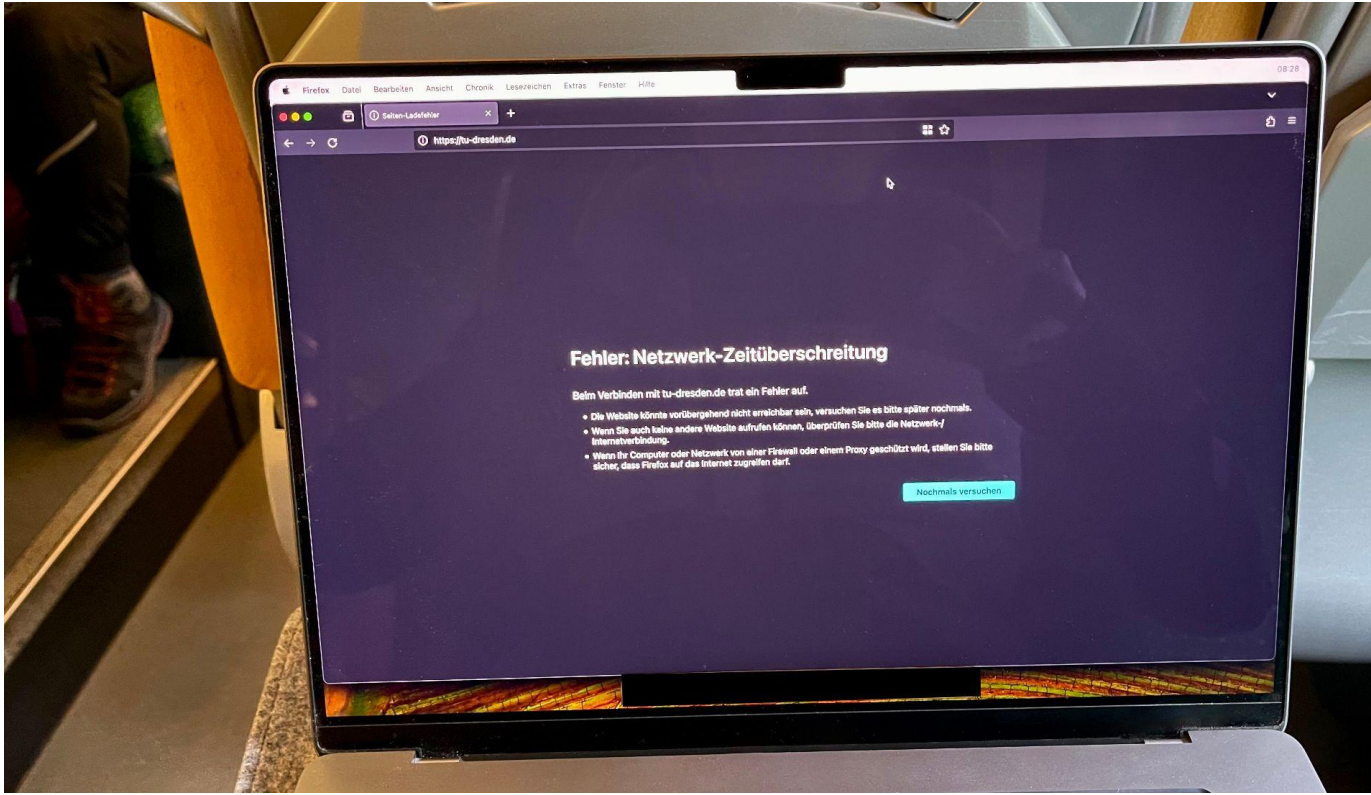


Image Source: Lucas Vogel, 2024

Extreme example: Internet at the south pole

- Story about IT Engineer (*brr*) of McMurdo South Pole Station
- Internet connectivity via satellites ⇒

- RTT of **750ms**
- Network speed:
  “a couple kbps (yes, you read that right), up to 2 mbps on a really good day” ***- brr, May 2024***

- Limited network connectivity, congestion, …
- Websites and apps load minutes or fail to load **due to bad design**

Story Source: https://brr.fyi/posts/engineering-for-slow-internet , 31.05.2024

South Pole's radomes, Credit: brr.fyi
Image source: https://brr.fyi/posts/engineering-for-slow-internet , 31.05.2024

> **It's frustrating when things don't work out, and it's frustrating when we know that certain services work particularly badly in our low-bandwidth, congested environment.”**
>
> —Brr, 2024

… Not just the south pole:

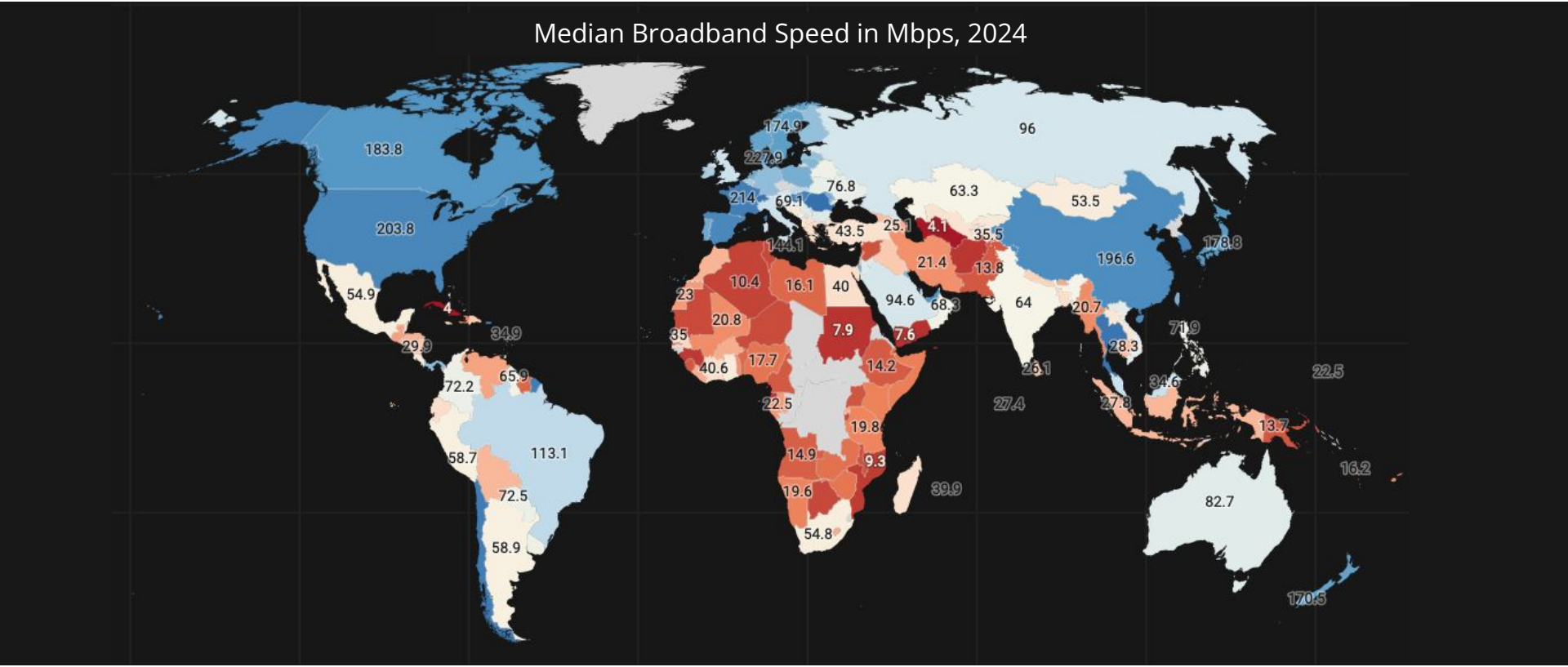


Image Source: https://www.datapandas.org/ranking/internet-speeds-by-country , 31.05.2024
Data Source: https://www.speedtest.net/global-index 31.05.2024

But why does web page speed matter?

User Satisfaction

- Tolerable waiting time: ~**2 seconds** [1]
- Otherwise: People will leave page

  ⇒ Users expect fast pages

Money

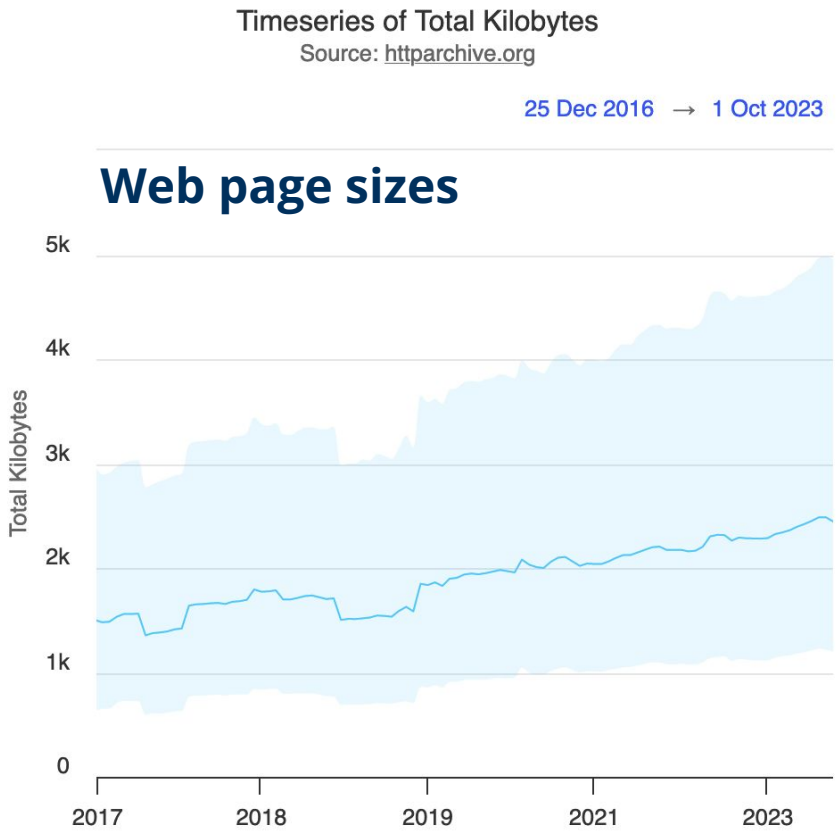
- **Amazon: 1% sales loss** for an additional 100ms delay [2]
- **Google: 20% revenue drop** for 500 ms additional delay [3]

[1] Nah, Fiona Fui-Hoon. "A study on tolerable waiting time: how long are web users willing to wait?." Behaviour & Information Technology 23.3 (2004): 153-163.

[2] Linden G (2006b) Make data useful. [Online] Dec 2006. http://web.archive.org/web/20081117195303/http://home.blarg.net/~glinden/StanfordDataMining.2006-11-29.ppt , archived via The Internet Archive

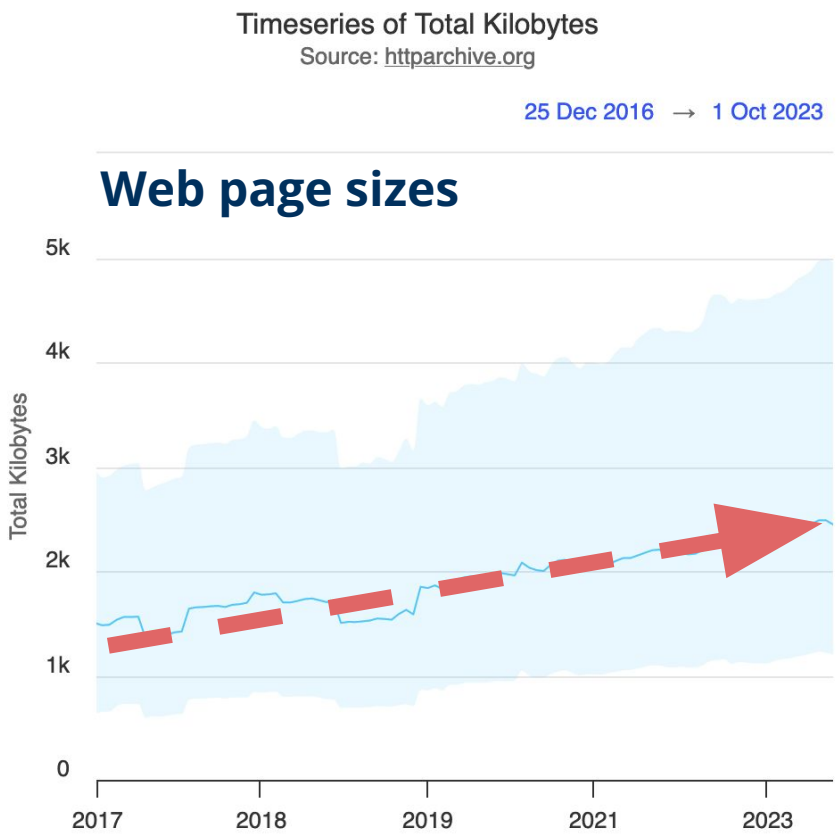
[3] https://ai.stanford.edu/~ronnyk/2009controlledExperimentsOnTheWebSurvey.pdf , online, 31.05.2024

Many changes try to cope with increasing web site complexity

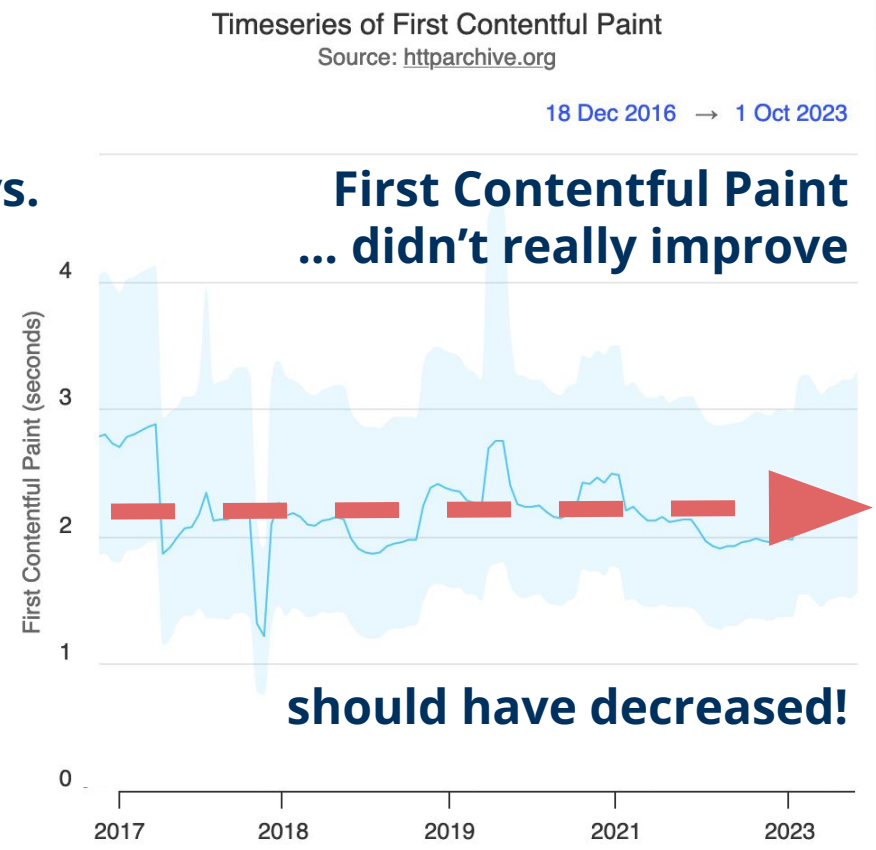


… including all layers of the Web ecosystem, e.g., HTTP, transport (QUIC), delivery infrastructure (CDN), and content bundling.

Many changes try to cope with increasing web site complexity … but



vs.



Website basics

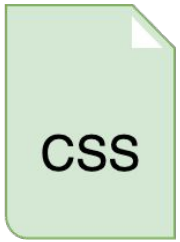
Main components of a web page

- “Hypertext Markup Language”
- Provides structure
- **Important foundation of web pages, allows for linking CSS and JavaScript files**
- based on tags, e.g.: <a>, ,<h1>,…

Source: https://html.spec.whatwg.org , online, 31.05.2024

Main components of a web page

- “Hypertext Markup Language”
- Provides structure
- **Important foundation of web pages, allows for linking CSS and JavaScript files**
- based on tags, e.g.: <a>, ,<h1>,...



- “Cascading Style Sheets”
- **allows for styling HTML elements and creating responsive designs**

3 ways to include CSS:

1. Inline (`style="..."`)
2. In-File (`<style>...</style>`)
3. external file linked via `<link rel="stylesheet" href="file.css">`

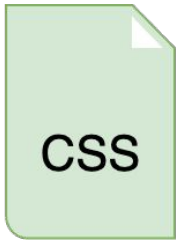
Source: https://html.spec.whatwg.org , online, 31.05.2024

Source: https://www.w3.org/Style/CSS/ online, 31.05.2024

Main components of a web page

- “Hypertext Markup Language”
- Provides structure
- **Important foundation of web pages, allows for linking CSS and JavaScript files**
- based on tags, e.g.: <a>, ,<h1>,...

Source: https://html.spec.whatwg.org , online, 31.05.2024



- “Cascading Style Sheets”
- **allows for styling HTML elements and creating responsive designs**

3 ways to include CSS:

1. Inline (`style="..."`)
2. In-File (`<style>...</style>`)
3. external file linked via `<link rel="stylesheet" href="file.css">`

Source: https://www.w3.org/Style/CSS/ online, 31.05.2024

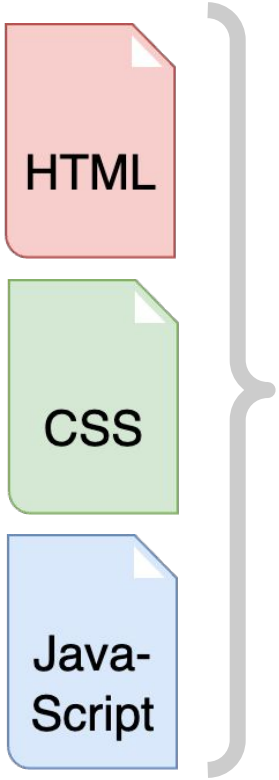
- Scripting language, “JS”
- **allows to create dynamic web applications**

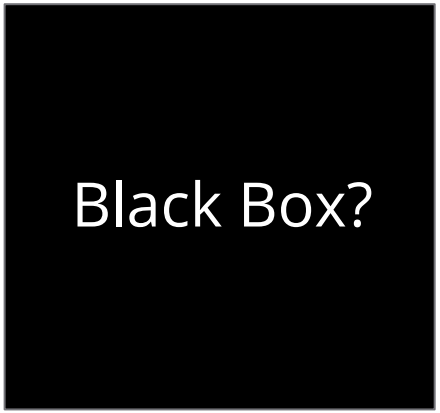
3 ways to include JS:

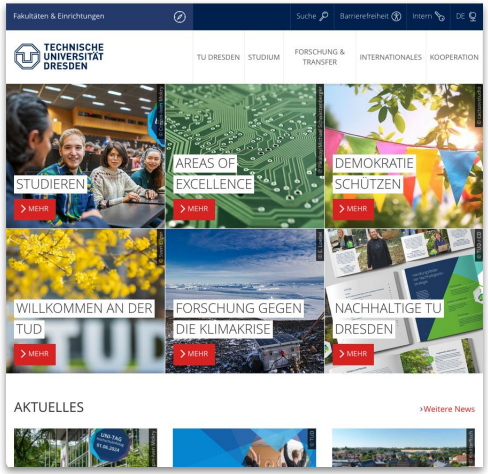
1. Inline (`onload="..."`)
2. In-File (`<script>...</script>`)
3. external file linked via `<script src="file.js">`

JavaScript™ is a trademark of Oracle.
Source: https://ecma-international.org online, 31.05.2024

How are complex websites created?

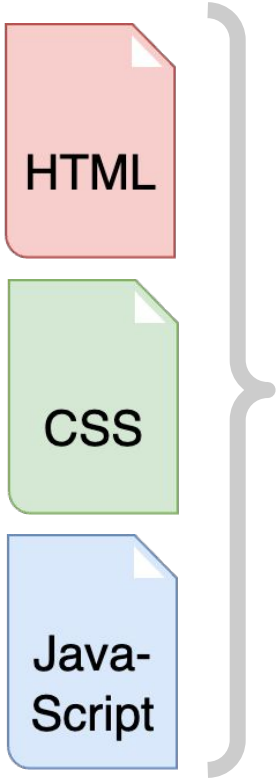




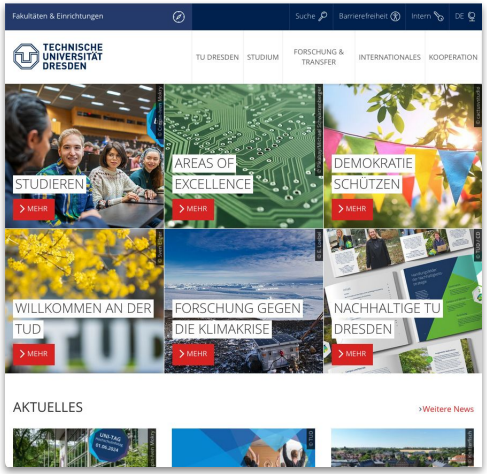


TU Dresden Website
Image source: https://tu-dresden.de/ , 31.05.2024

How are complex websites created?



Mostly frame-works!

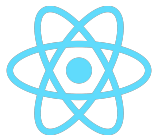


TU Dresden Website
Image source: https://tu-dresden.de/ , 31.05.2024

Frameworks

- **Building large and scalable web applications is difficult** [4]
- Solution: Using frameworks and libraries to mitigate shortcomings

- Popular Web frameworks include: **React, Angular, Vue.js, Svelte,...** [5]
- Popular JavaScript libraries include: **jQuery, Lodash, HTMX,...** [6]
- Popular CSS frameworks include: **Bootstrap, TailwindCSS, Bulma,...** [7]


React


Angular


Vue.js


Svelte

jQuery

Lodash

Bootstrap

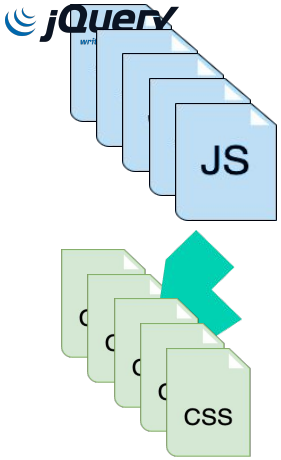

TailwindCSS


Bulma

Sources:
[4] https://8thlight.com/insights/a-history-of-javascript-modules-and-bundling-for-the-post-es6-developer online, 01.06.2024
[5] https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190 online, 01.06.2024
[6] https://2022.stateofjs.com , online, 01.06.2024
[7] https://ossinsight.io/collections/css-framework/ , online, 01.06.2024

Logo Sources:
https://react.dev/ online, 01.06.2024
https://angular.io/ online, 01.06.2024
https://vuejs.org/ online, 01.06.2024
https://svelte.dev/ online, 01.06.2024
https://jquery.com/ online, 01.06.2024
https://lodash.com online, 01.06.2024
https://getbootstrap.com/ online, 01.06.2024
https://tailwindcss.com/brand online, 01.06.2024
https://bulma.io/brand/ online, 01.06.2024

Result of large and complex web apps with frameworks until ≈2015



Problems:

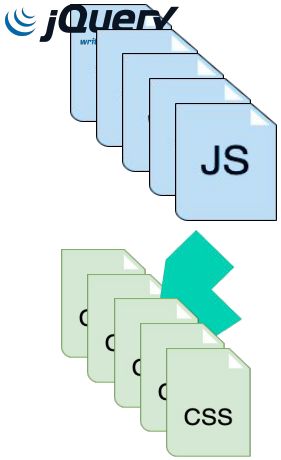
1. Large amounts of code and libraries!
2. Could NOT use imports in JS! *
3. Request time penalty due to handshakes before HTTP/2

What solution was used?

* JavaScript did not have a native module system until 2015 [4]

Sources:
https://jquery.com/ online, 01.06.2024
https://bulma.io/brand/ online, 01.06.2024
[4] https://8thlight.com/insights/a-history-of-javascript-modules-and-bundling-for-the-post-es6-developer online, 01.06.2024

Result of large and complex web apps with frameworks until ≈2015



Problems:

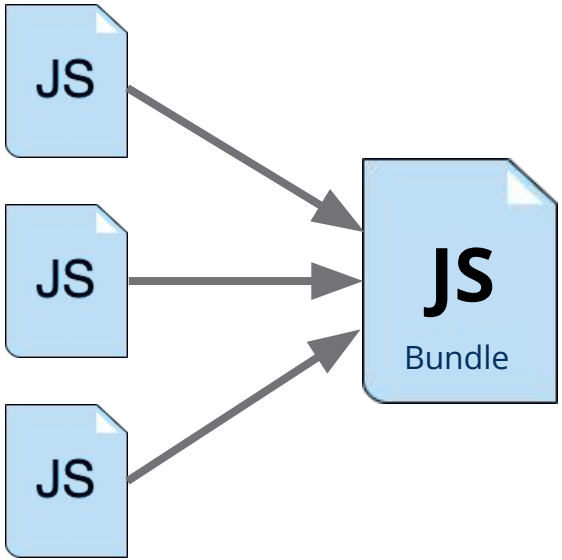
1. Large amounts of code and libraries!
2. Could NOT use imports in JS! *
3. Request time penalty due to handshakes before HTTP/2

Resolving imports and bundle!

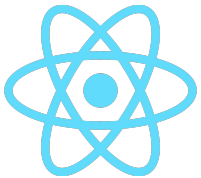
* JavaScript did not have a native module system until 2015 [4]

Sources:
https://jquery.com/ online, 01.06.2024
https://bulma.io/brand/ online, 01.06.2024
[4] https://8thlight.com/insights/a-history-of-javascript-modules-and-bundling-for-the-post-es6-developer online, 01.06.2024

What are bundles?



- Bundles: combine multiple files into one
- Started with **Browserify**, ca. 2013
- **Due to an increase in code, a need for resolving imports and performance issues of a large number of files**
- Today: most popular bundler: **Webpack**
- **By default: produces render-blocking bundles**
- **Webpack** is used by:



React

Angular

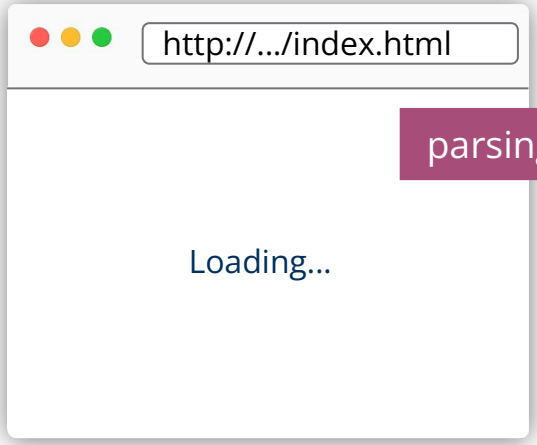
Vue.js

.. and many more

Logo Sources:
https://react.dev/ online, 01.06.2024
https://angular.io/ online, 01.06.2024
https://vuejs.org/ online, 01.06.2024

What is render-blocking? (simplified)

Browser

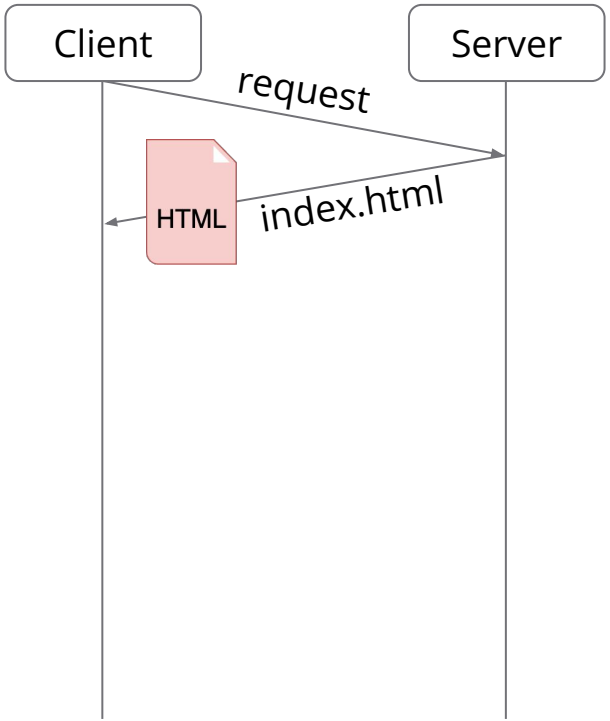


parsing

Code

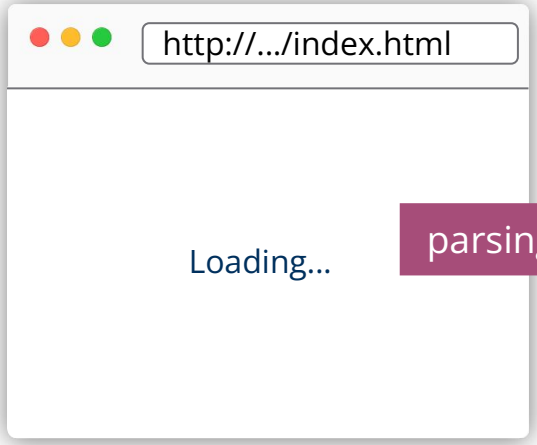
```
index.html

<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet"
    href="styles.css">
</head>
<body>
    <script src="script.js">
    </script>
</body>
</html>
```



What is render-blocking? (simplified)

Browser

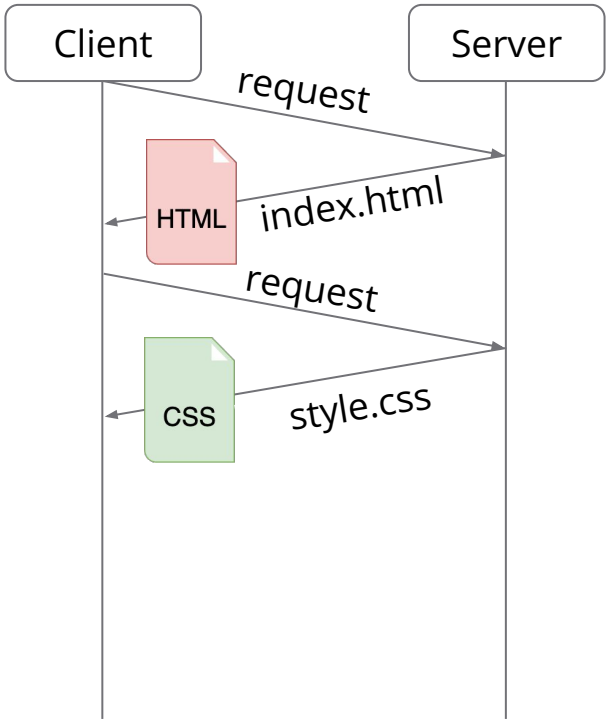


parsing

Code

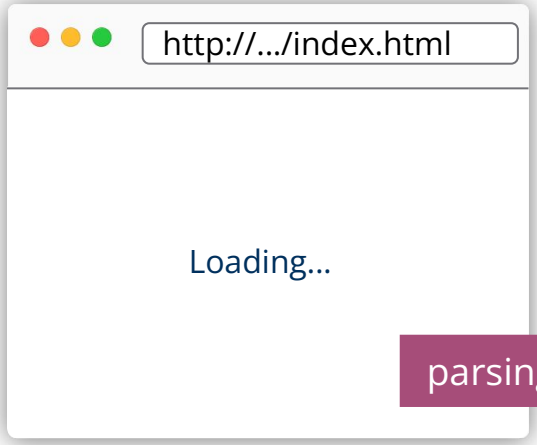
```
index.html

<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet"
    href="styles.css">
</head>
<body>
    <script src="script.js">
    </script>
</body>
</html>
```



What is render-blocking? (simplified)

Browser

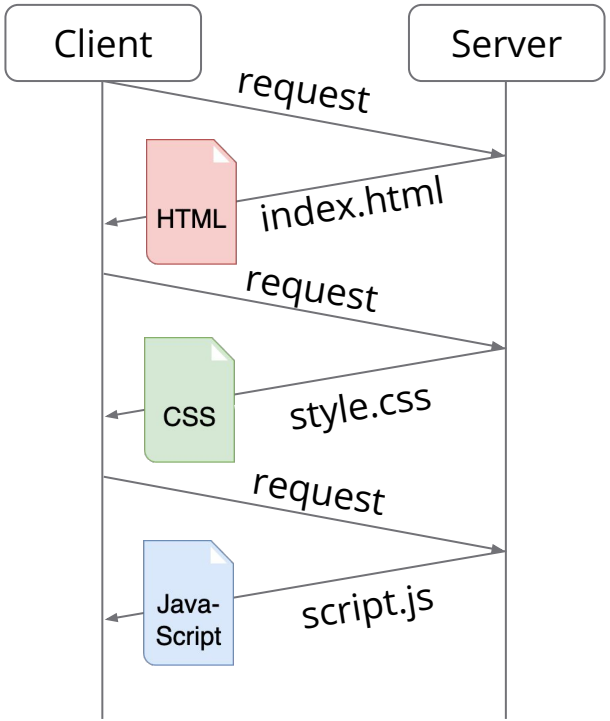


parsing

Code

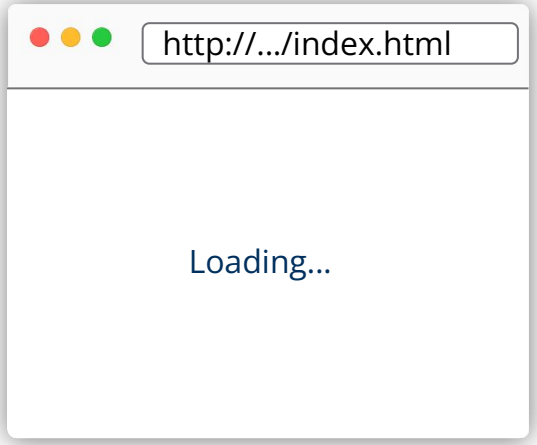
```
index.html

<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet"
    href="styles.css">
</head>
<body>
    <script src="script.js">
    </script>
</body>
</html>
```



What is render-blocking? (simplified)

Browser

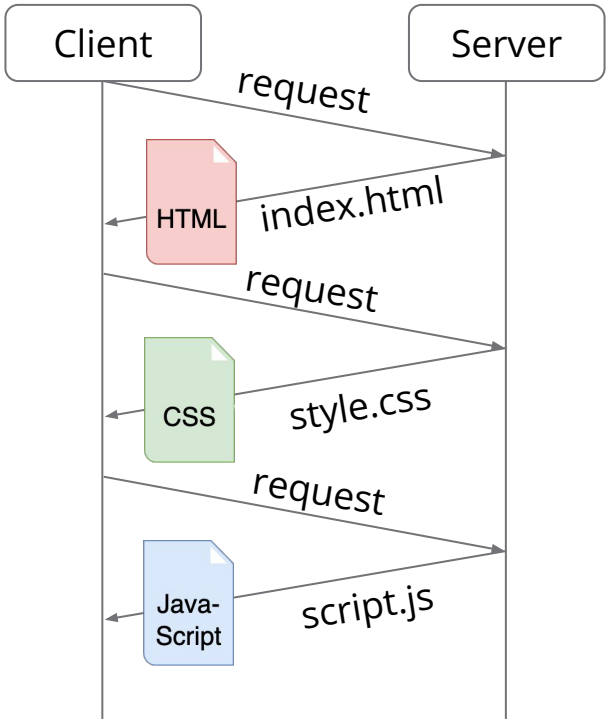


Code

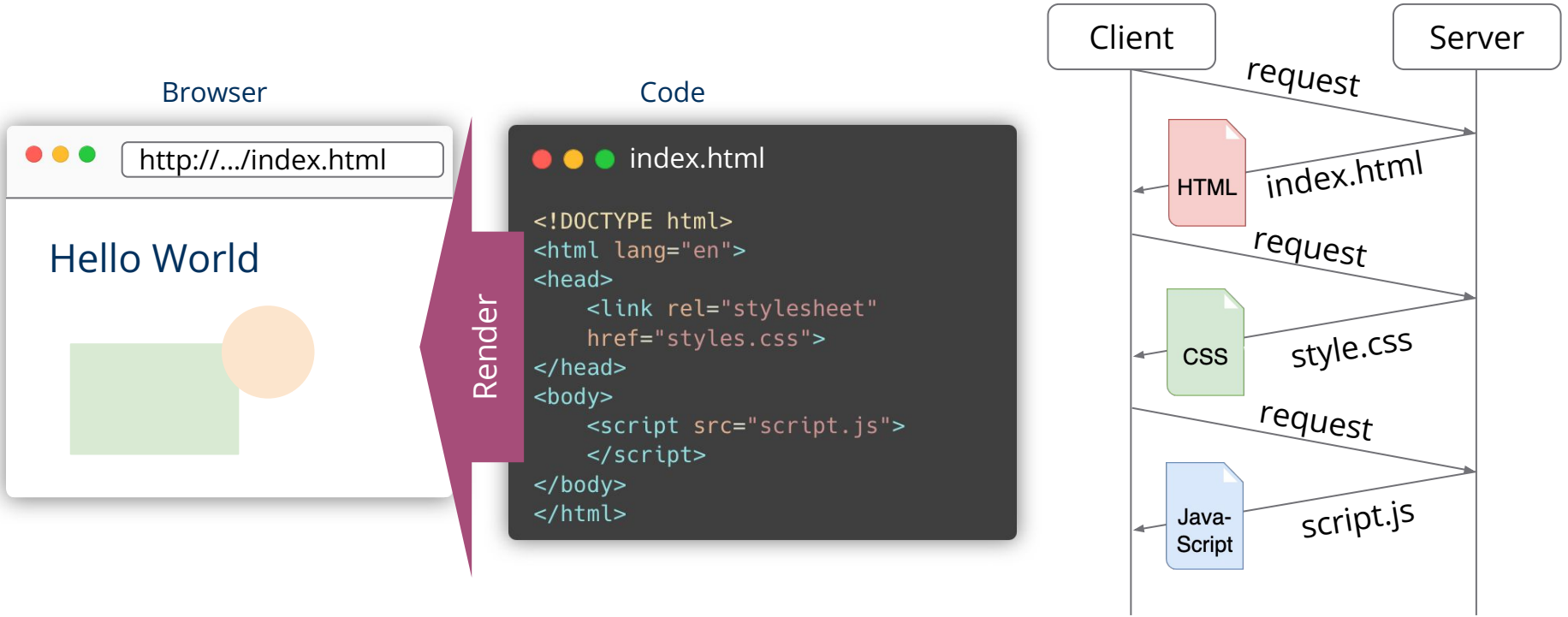
index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet"
    href="styles.css">
</head>
<body>
    <script src="script.js">
    </script>
</body>
</html>
```

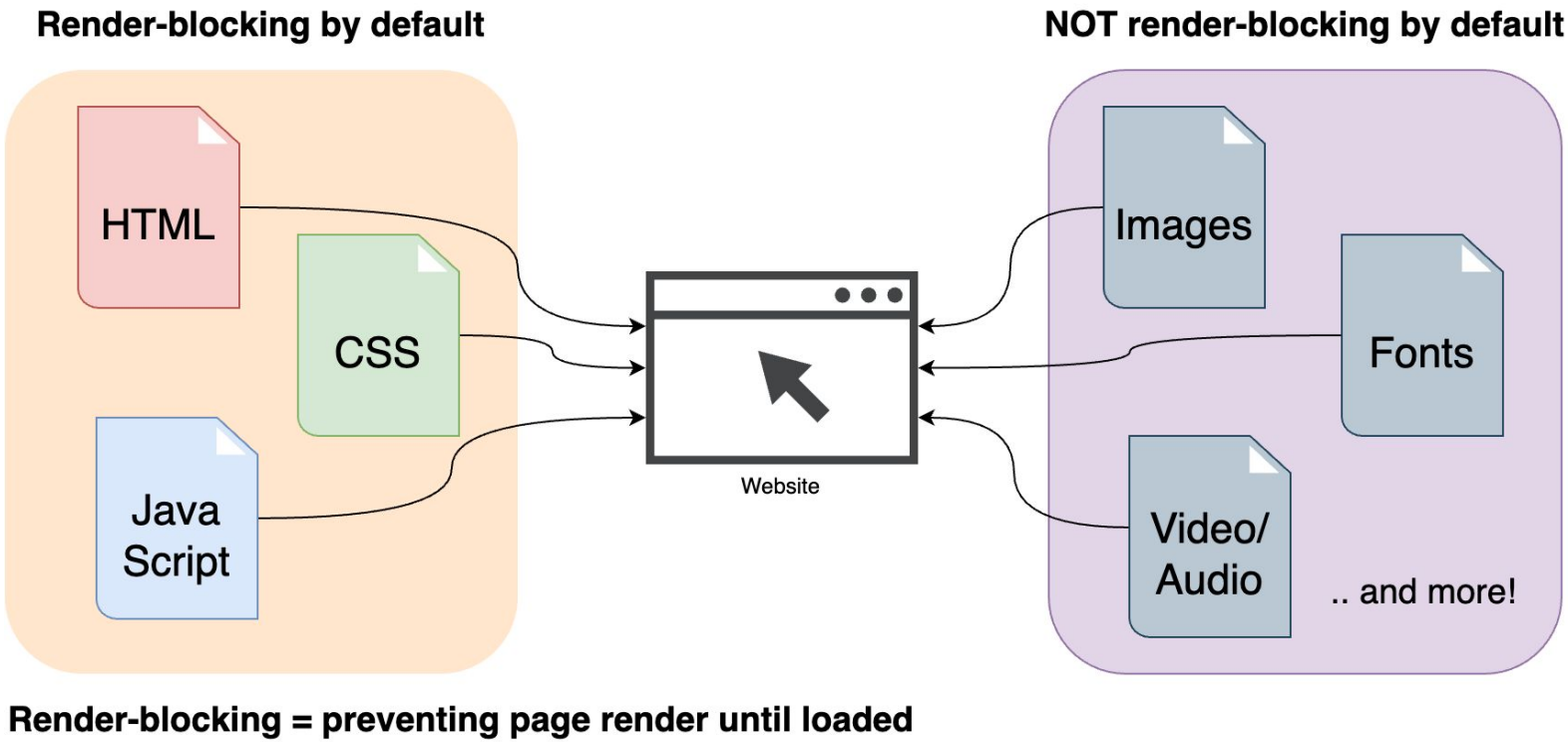
parsing finished



What is render-blocking? (simplified)



What is render-blocking?



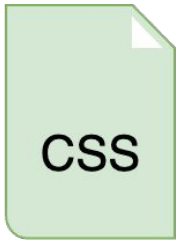
How to make it NOT render-blocking?

- Always render-blocking by definition
- **However**: Not the entire page needs to be loaded in order for parsing and rendering to start in most modern browsers if no render-blocking elements are linked!

Source: https://html.spec.whatwg.org , online, 31.05.2024

How to make it NOT render-blocking?

- Always render-blocking by definition
- **However**: Not the entire page needs to be loaded in order for parsing and rendering to start in most modern browsers if no render-blocking elements are linked!



- Internal CSS is always render-blocking
- Only native way of delaying external CSS: **media**-attributes

```
<link rel="stylesheet"
href="file.css"
media=”print”>
```

- Common hack: using JavaScript to delay CSS

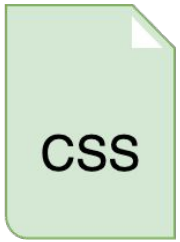
Source: https://html.spec.whatwg.org , online, 31.05.2024

Source: https://www.w3.org/Style/CSS/ online, 31.05.2024

How to make it NOT render-blocking?

- Always render-blocking by definition
- **However**: Not the entire page needs to be loaded in order for parsing and rendering to start in most modern browsers if no render-blocking elements are linked!

Source: https://html.spec.whatwg.org , online, 31.05.2024



- Internal CSS is always render-blocking
- Only native way of delaying external CSS: **media**-attributes

```
<link rel="stylesheet"
href="file.css"
media="print">
```

- Common hack: using JavaScript to delay CSS

Source: https://www.w3.org/Style/CSS/ online, 31.05.2024

Two methods are available:

1. **async:** <script src="..." async> script loads in background, executes when loaded

2. **defer:** <script src="..." defer> script loads in background, executes when HTML is fully parsed

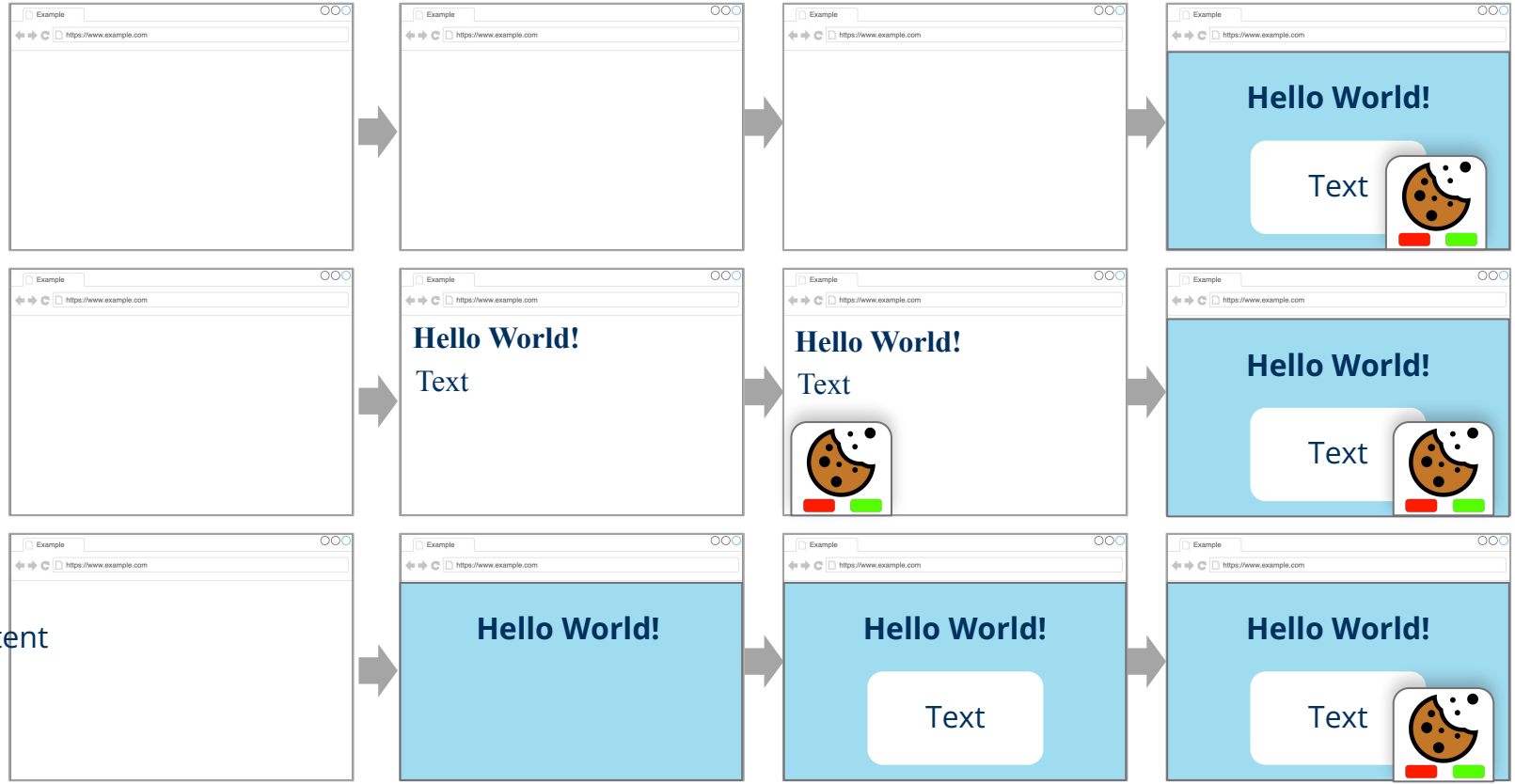
JavaScript™ is a trademark of Oracle.
Source: https://ecma-international.org online, 31.05.2024

The issue with simply making code non-render-blocking

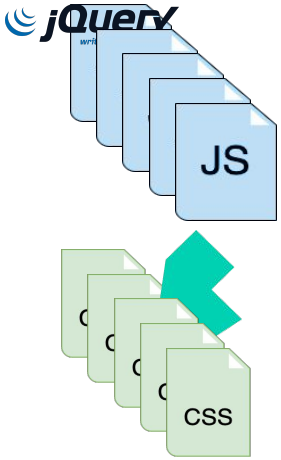
Status quo
All or nothing

Naive approach
Just don't block the render? (async/defer)

What we want
Progressive content



Revisiting the issues of a developer after 2015



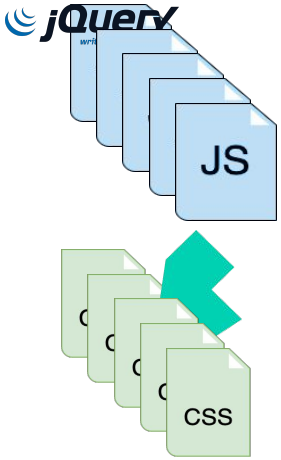
Problems:

1. Large amounts of code and libraries!
2. Could NOT use imports in JS!
3. Request time penalty due to handshakes before HTTP/2

Still an issue today.

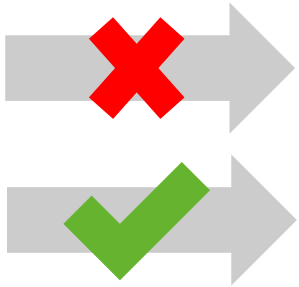
Sources:
https://jquery.com/ online, 01.06.2024
https://bulma.io/brand/ online, 01.06.2024
https://8thlight.com/insights/a-history-of-javascript-modules-and-bundling-for-the-post-es6-developer online, 01.06.2024

Revisiting the issues of a developer after 2015



Problems:

1. Large amounts of code and libraries!
2. Could NOT use imports in JS!
3. Request time penalty due to handshakes before HTTP/2

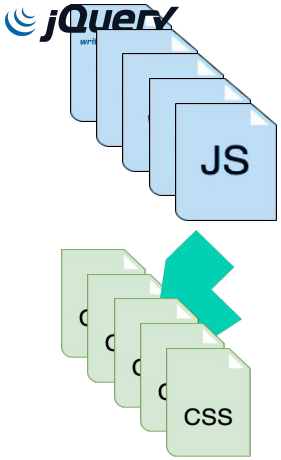


Still an issue today.

JS modules are available

Sources:
https://jquery.com/ online, 01.06.2024
https://bulma.io/brand/ online, 01.06.2024
https://8thlight.com/insights/a-history-of-javascript-modules-and-bundling-for-the-post-es6-developer online, 01.06.2024

Revisiting the issues of a developer after 2015



Problems:

1. Large amounts of code and libraries!
2. Could NOT use imports in JS!
3. Request time penalty due to handshakes before HTTP/2

Still an issue today.

JS modules are available

HTTP/2 and HTTP/3 mostly solve request overhead
(streams, no TCP-HOL-blocking, 0-RTT,..)

Sources:
https://jquery.com/ online, 01.06.2024
https://bulma.io/brand/ online, 01.06.2024
https://8thlight.com/insights/a-history-of-javascript-modules-and-bundling-for-the-post-es6-developer online, 01.06.2024

Revisiting the issues of a developer after 2015

The Problem:
Core issues are resolved, BUT:

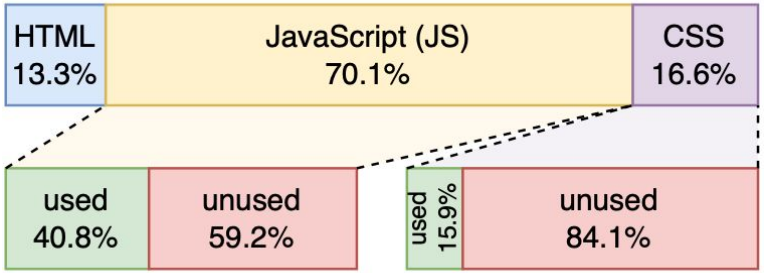
- Developers still use bundlers
- Still large amounts of code

What could be the reason?

Sources:
https://jquery.com/ online, 01.06.2024
https://bulma.io/brand/ online, 01.06.2024
https://8thlight.com/insights/a-history-of-javascript-modules-and-bundling-for-the-post-es6-developer online, 01.06.2024

Bundles and libraries: a deeper look

- **Code efficiency:**
  - Both bundles and libraries contain code that is usually not needed for rendering, even after optimizations such as tree shaking
  - **Example**: CSS framework provides styling for forms ⇒ website does not use forms ⇒ styles are loaded without use
- **How much excess code is loaded on average?**
  - Study reveals: Majority of JavaScript and CSS code is unused until render



Share of HTML, JavaScript and CSS and the average use of JS and CSS until the page is rendered in percent

Source: Vogel, Lucas, and Thomas Springer. "An in-depth analysis of web page structure and efficiency with focus on optimization potential for initial page load." International Conference on Web Engineering. Cham: Springer International Publishing, 2022.

Bundling is an anti-pattern in HTTP/2

> **In HTTP/2, [bundling] will end up impacting the download-time of other resources as well, because of the way HTTP/2 works.”**
>
> —Erwin Hofman, 2022

[https://www.erwinhofman.com/blog/two-main-performance-debts-of-http1/]

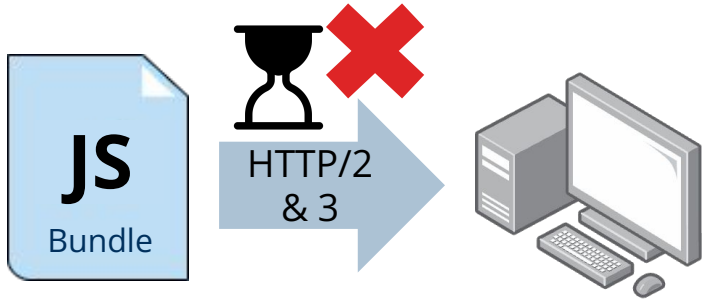
Summary so far:

- Websites are based on **HTML**, **JavaScript,** and **CSS**
- Developers use **frameworks, libraries,** and **bundlers** to create complex web applications

- **Bundlers** combine multiple resources
  - Bundlers solved performance and import issue before approx. 2015
  - However, core issues are resolved now

- **Bundlers are now considered an anti-pattern**

- **Can hurt loading performance due to lack of code use until render**



Measurement basics

Measurement basics - important metrics

First Contentful Paint (FCP)

- Time until the first piece of content is shown to the user

Measurement basics - important metrics

First Contentful Paint (FCP)

- Time until the first piece of content is shown to the user

Largest Contentful Paint (LCP)

- Time until the largest piece of content is shown to the user (e.g. largest image, headline,...)

Measurement basics - important metrics

First Contentful Paint (FCP)

- Time until the first piece of content is shown to the user

Largest Contentful Paint (LCP)

- Time until the largest piece of content is shown to the user (e.g. largest image, headline,...)

Time to First Byte (TTFB)

- Time between request and first byte of response

Measurement basics - important metrics

First Contentful Paint (FCP)

- Time until the first piece of content is shown to the user

Largest Contentful Paint (LCP)

- Time until the largest piece of content is shown to the user (e.g. largest image, headline,...)

Time to First Byte (TTFB)

- Time between request and first byte of response

Even more metrics available:

- **Cumulative Layout Shift (CLS)**
  ⇒ Shift of elements while loading
- **Interaction to Next Paint (INP)**
  ⇒ Input delay (responsiveness)
- and many more ....

Core Web Vitals

- Subset of Web Vitals, by Google
- Three main Components: **LCP, INP, CLS** (currently)
- measures user-centric loading- and interaction speed

Information and image source: https://web.dev/articles/vitals , online, 03.06.2024

Measurement basics

How do you know if your web page is “fast” or “slow”?

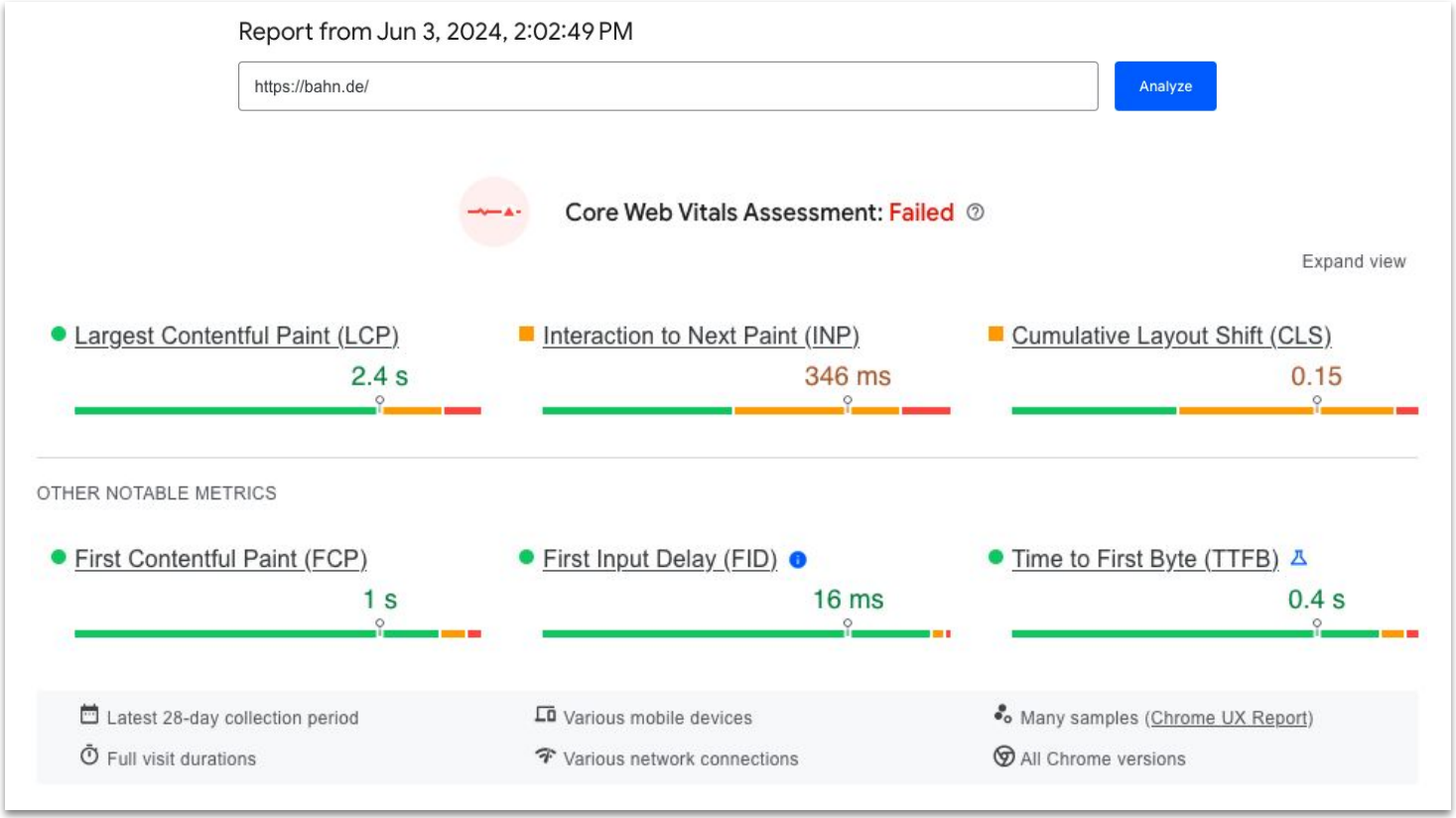
How to actually measure a web page?

Popular tools:

- PageSpeed Insights by Google (pagespeed.web.dev)
- Google Lighthouse
- WebPageTest (webpagetest.org), GTmetrix (gtmetrix.com),....

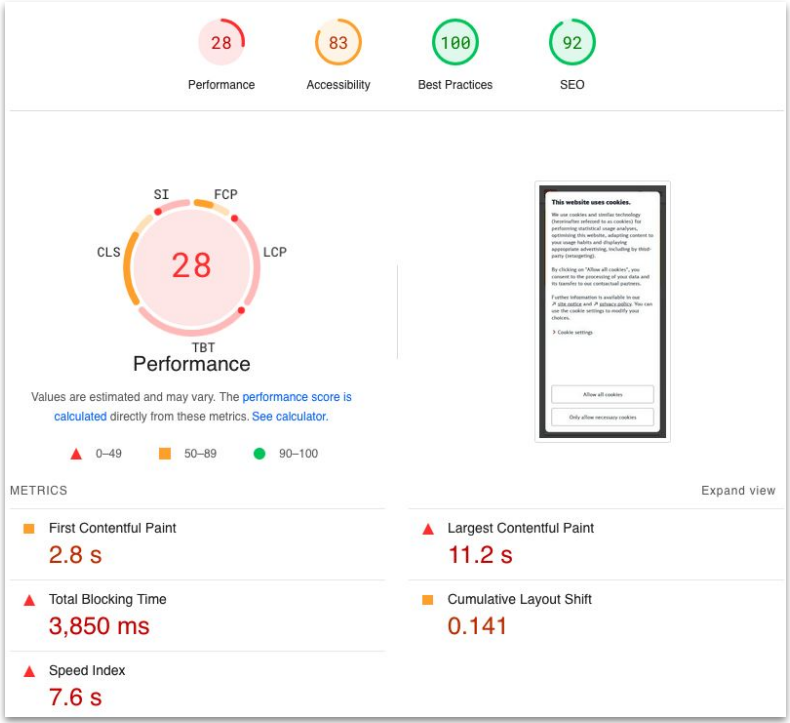
Timeline of tu-dresden.de loading, performance test via GTmetrix
Image source: https://gtmetrix.com/reports/tu-dresden.de/ , 31.05.2024

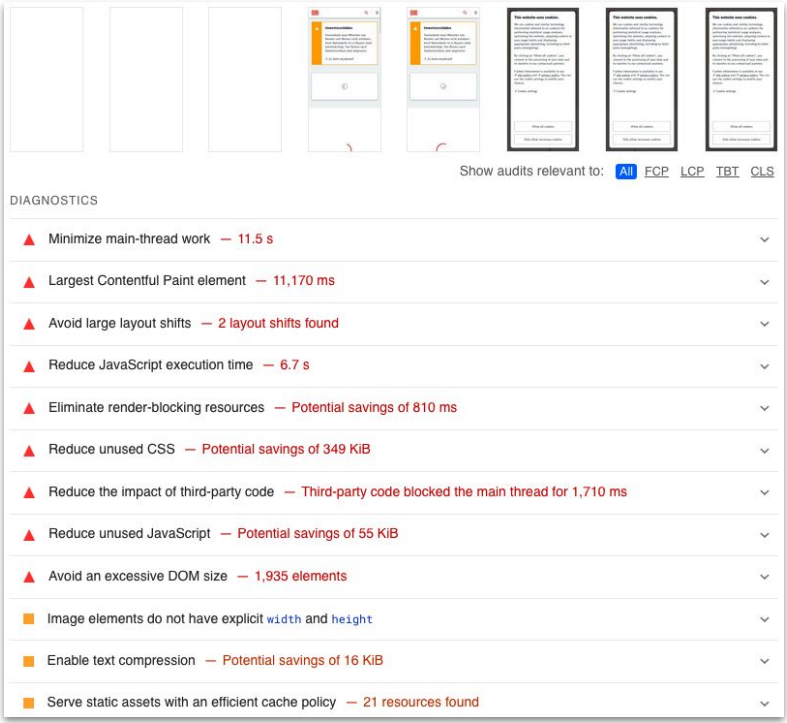
Example: PageSpeed Insights of bahn.de - how to read it



source: https://pagespeed.web.dev/analysis/https-bahn-de/ovls47w5lg?form_factor=mobile , 03.06.2024

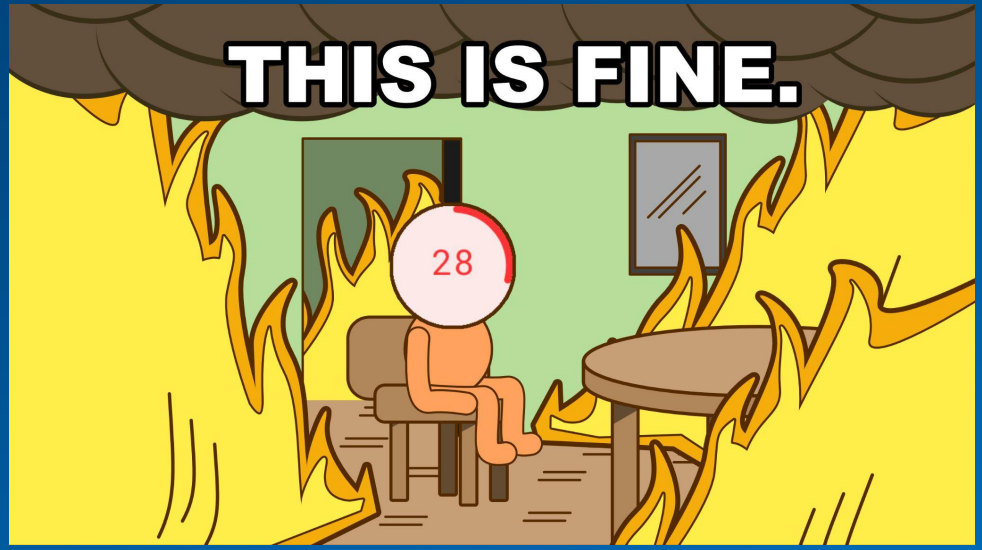
Example: PageSpeed Insights of bahn.de - how to read it





source: https://pagespeed.web.dev/analysis/https-bahn-de/ovls47w5lg?form_factor=mobile , 03.06.2024

Optimization Basics



Optimize JavaScript

Strategy 1: **Delete Code** | Strategy 2: **Delay Code** | Strategy 3: **Split Code**

- **Core idea:**
  automatically delete all unused code

- Ongoing research in this area (see examples ⇒)
- **However**: No existing method has a 100% accurate detection rate
- Can break a web page due to incorrect detection

Moumena Chaqfeh et al.: *To Block or Not to Block: Accelerating Mobile Web Pages On-The-Fly Through JavaScript Classification*

Tofunmi Kupoluyi et al.: *Muzeel: A Dynamic JavaScript Analyzer for Dead Code Elimination in Today's Web*

Gao Qiong and Wenmin Li: *An Optimization Method of Javascript Redundant Code Elimination based On Hybrid Analysis Technique*

Optimize JavaScript

Strategy 1: **Delete Code** | Strategy 2: **Delay Code** | Strategy 3: **Split Code**

- **Core idea:**
  automatically delay code so it is not render-blocking
- Can be done natively ⇒
- Framework also available: **Partytown**
  - Pro: Easy lazy-loading
  - Con: Uses deprecated browser feature, might get deactivated
  - partytown.builder.io

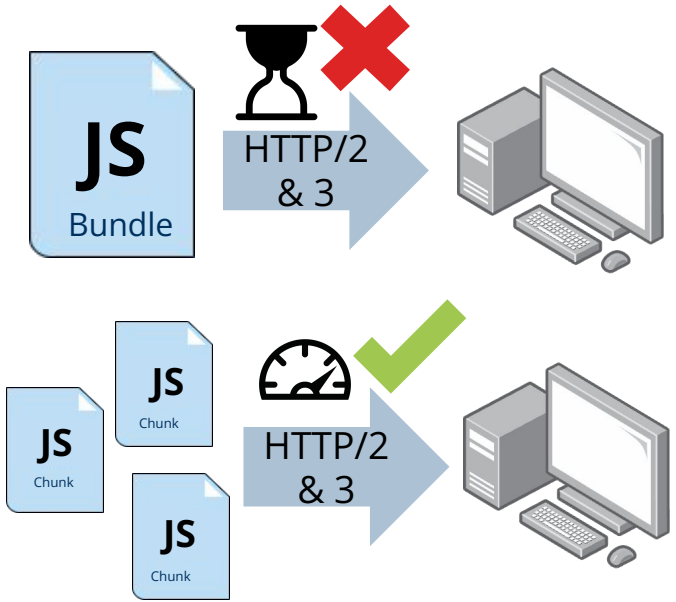
Two methods are available:

1. **async:** <script src="…" async> script loads in background, executes when loaded
2. **defer:** <script src="…" defer> script loads in background, executes when HTML is fully parsed

Optimize JavaScript

Strategy 1: **Delete Code** | Strategy 2: **Delay Code** | Strategy 3: **Split Code**

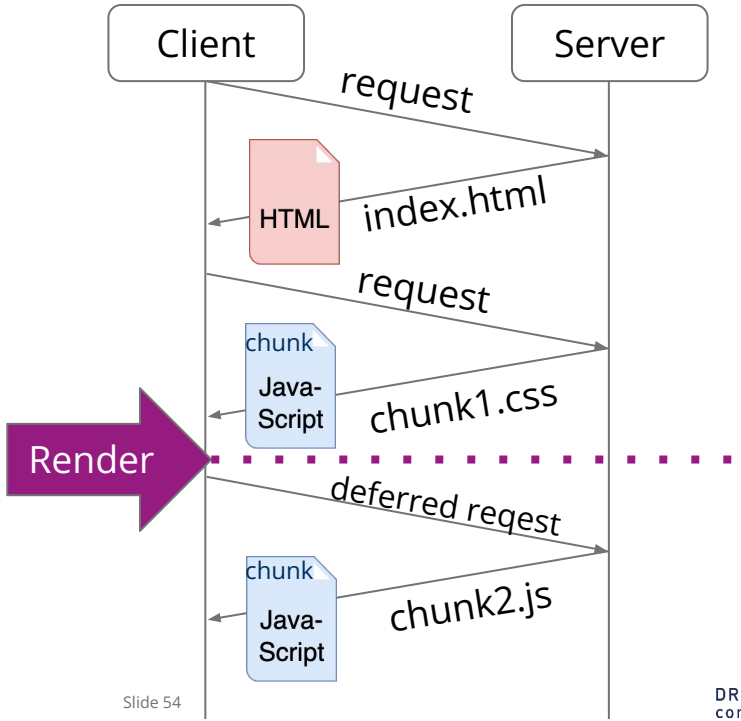
- **Core idea:**
  instead of bundling, use splitting
- Depending on implementation: can be faster, as chunks can be executed earlier, not waiting for the entire bundle to download
- Framework available: **Qwik** (qwik.dev)
  - Pro: fast page load, fast execution
  - Con: requires rewrite of code



Optimize JavaScript

Strategy 1: **Delete Code** | Strategy 2: **Delay Code** | Strategy 3: **Split Code**

- **Why splitting can be faster:**
  - By splitting up content into small chunks, rendering can begin faster by only sending data in the first chunk that is required for initial load
- **Less critical code can be delayed via async/defer**



Optimize JavaScript

Strategy 1: **Delete Code** | Strategy 2: **Delay Code** | Strategy 3: **Split Code**

- **How to not break your code:**
  - Delaying code (e.g. libraries) can lead to calling functions that are not available (yet) ⇒ breaks code
  - Fix: **Waiter**
    - Self-developed, open source library
    - Allows calling delayed functions without breaking code

Waiter logo
Image source: © Dr. Lucas Vogel
https://github.com/waiter-and-autratac/WaiterAndAUTRATAC , 31.05.2024

Optimize CSS

Strategy 1: **Delay CSS**

Strategy 2: **Calculate Necessary CSS**

- **Core idea:**
  Load less important CSS later
- Only native way: **media**-attributes ⇒
  - Pro: native, handled by the browser
  - Con: are only active if media attribute is true, limited use, likely leads to layout shift

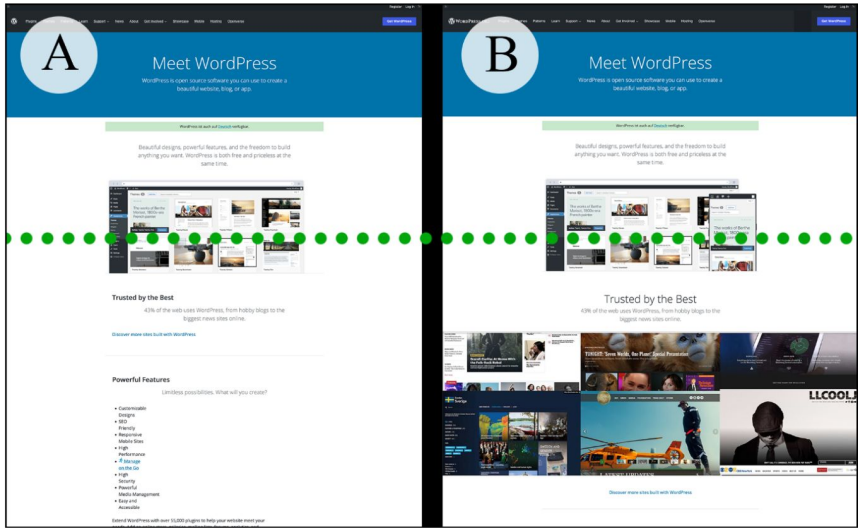
```
<link rel="stylesheet"
href="file.css" media=”print”>
```

Optimize CSS

Strategy 1: **Delay CSS**

Strategy 2: **Calculate Necessary CSS**

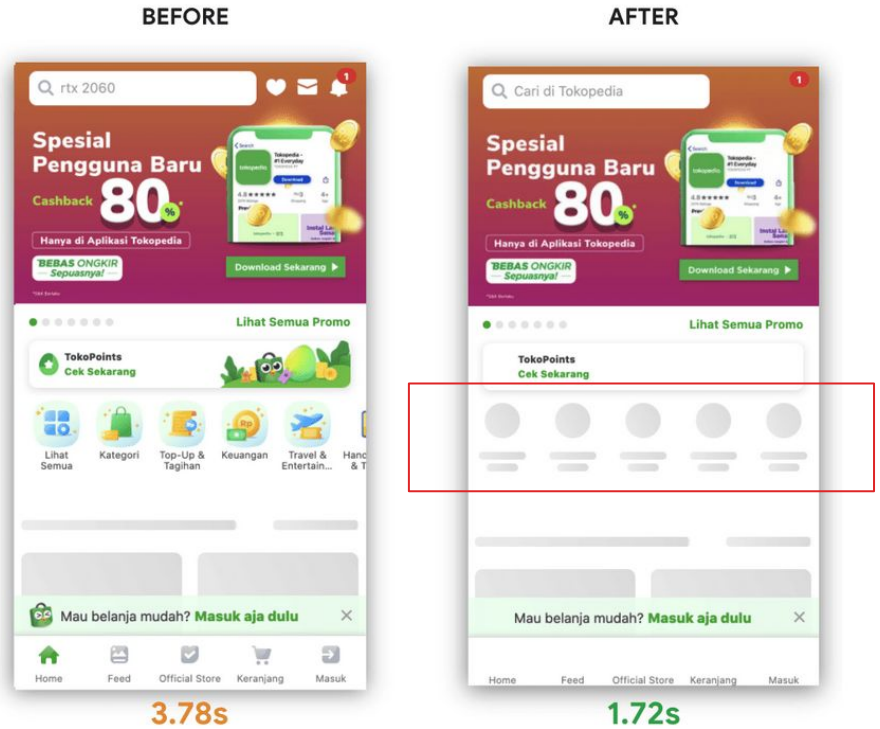
- **Own work:**
  - Focussing on improving render performance and viewport
  - Rendering entire page
    - Overall improved performance of code efficiency, visual similarity, and overall loading times compared to existing methods



CSS calculation (rendering) with (A) Critical and (B) own method
Source: Vogel, Lucas, and Thomas Springer. "Speed Up the Web with Universal CSS Rendering." International Conference on Web Engineering. Cham: Springer Nature Switzerland, 2023.

General improvements

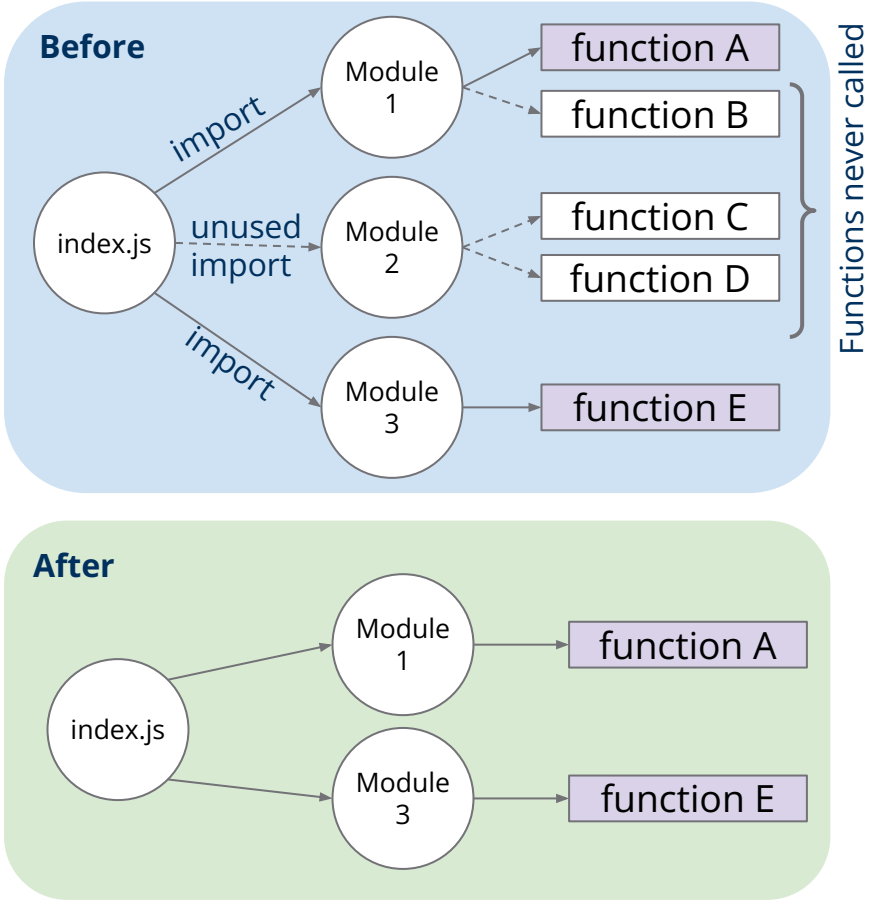
- **Loading is a process over time**
  - Delaying content can be integrated into design
- **Use placeholders:**
  - See example ⇒
  - Users are willing to wait longer if progress is visible
  - also: Important UI elements (such as search) is already available
  - Placeholders reduce/remove layout shift



Delaying UI content and use of placeholders, Tokopedia
Source: https://web.dev/case-studies/vitals-business-impact online, 04.06.2024

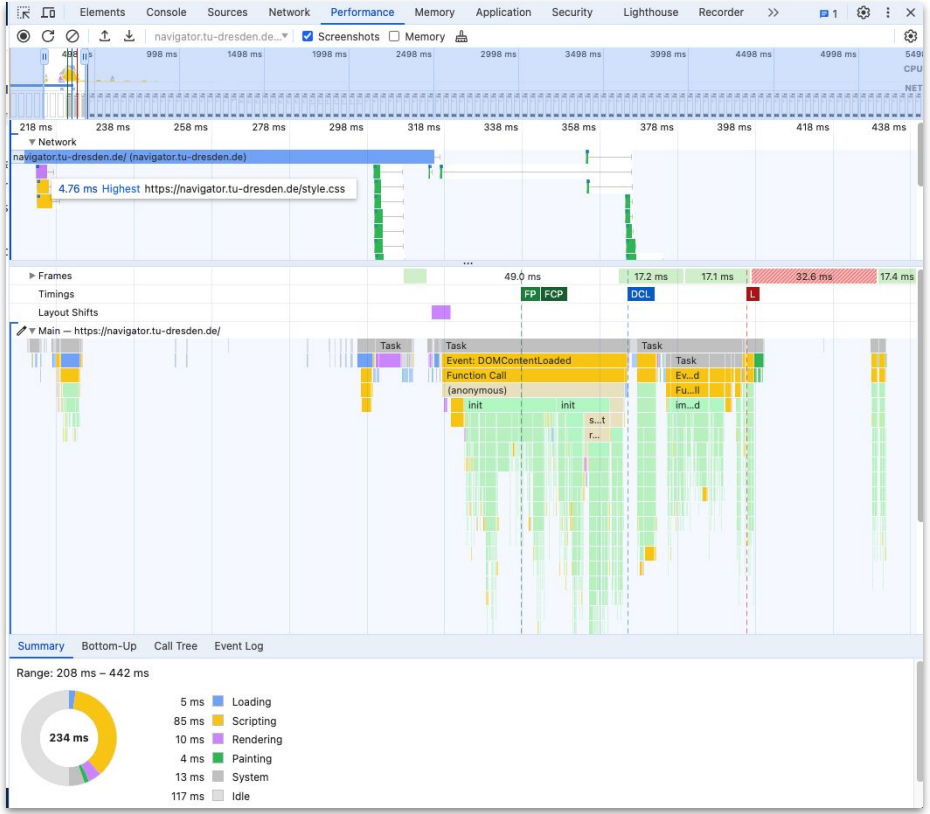
General improvements

- **Tree Shaking**
  - Method of dead code elimination
  - **Resolving unused imports**
  - Often works on file boundaries
  - Technique is commonly referred to as JavaScript optimization, but could also be used with CSS (less common)
  - Caveats: mostly applied to files, requires original, non-bundled source-code and use of imports/exports



Further tools: Developer Console

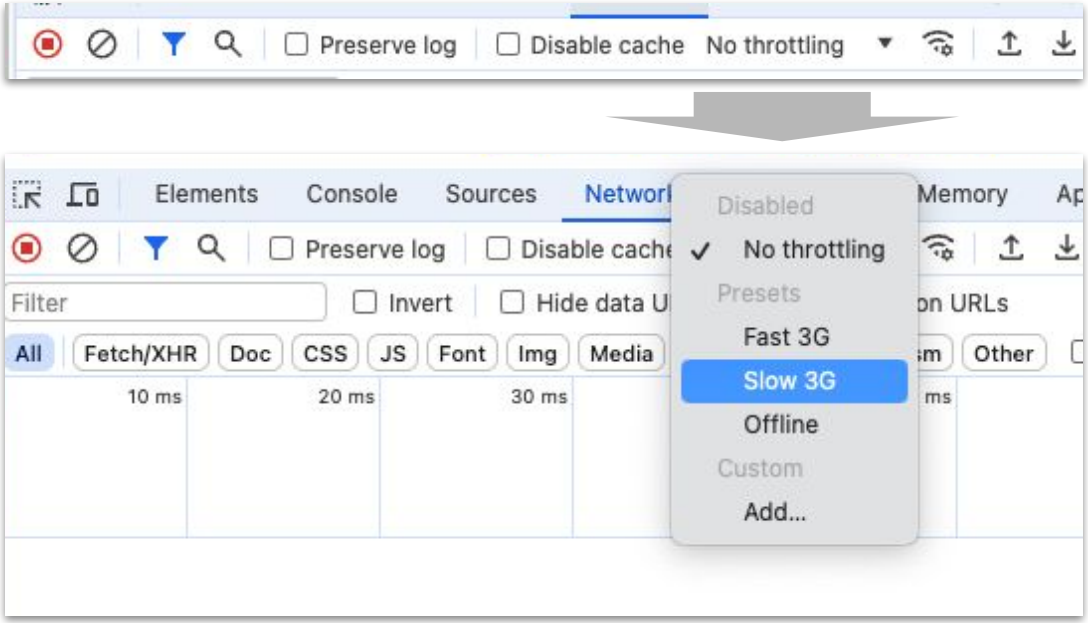
- **All** major browsers provide developer console
- In Chrome: also **Lighthouse**
- All have some form of network and performance information tab



source: Chrome via https://navigator.tu-dresden.de/ , 03.06.2024

Further tools: Developer Console

- Most browsers allow for simulating slow networks
- use developer console ⇒
- **Recommendation:**
  - Start testing slow network speeds early in development
  - Later fixes are often more difficult



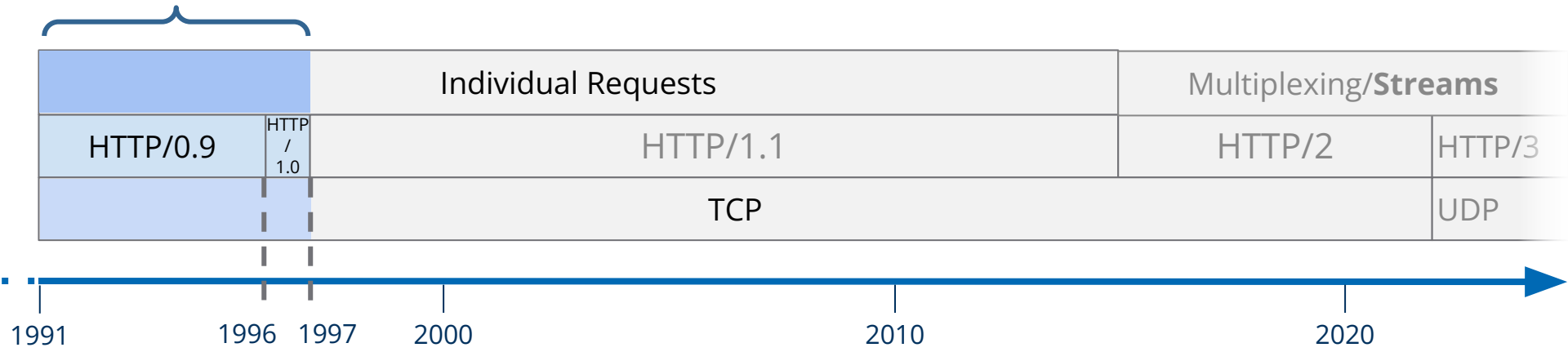
source: Chrome developer console, 03.06.2024

The next generation: streaming

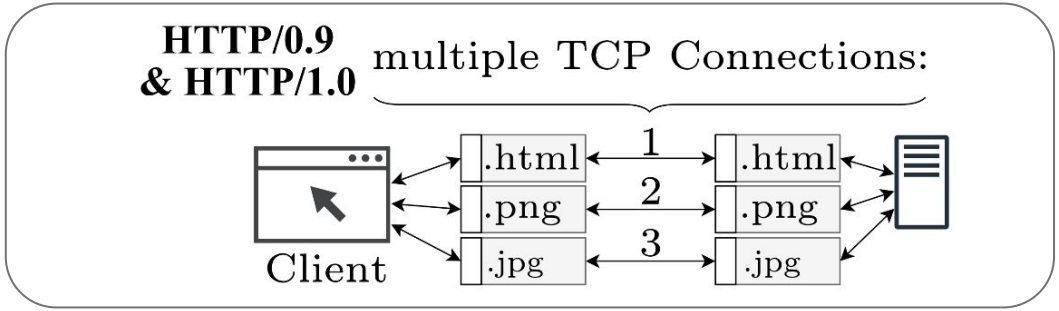
HTTP/0.9 & 1.0

Penalty for every request: needed full TCP handshake

Less files
= faster loading time

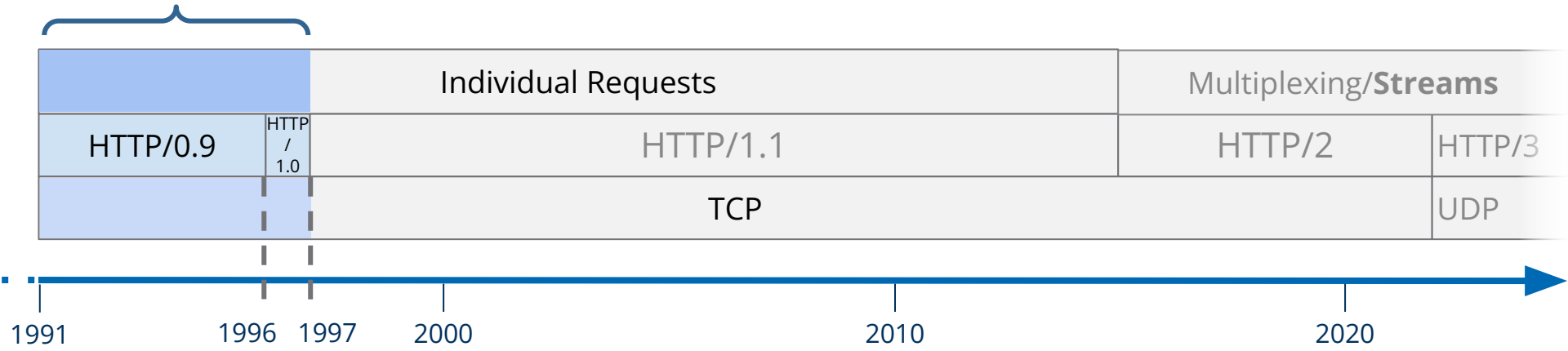


HTTP/0.9 & 1.0



Penalty for every request: needed full TCP handshake

Less files
= faster loading time

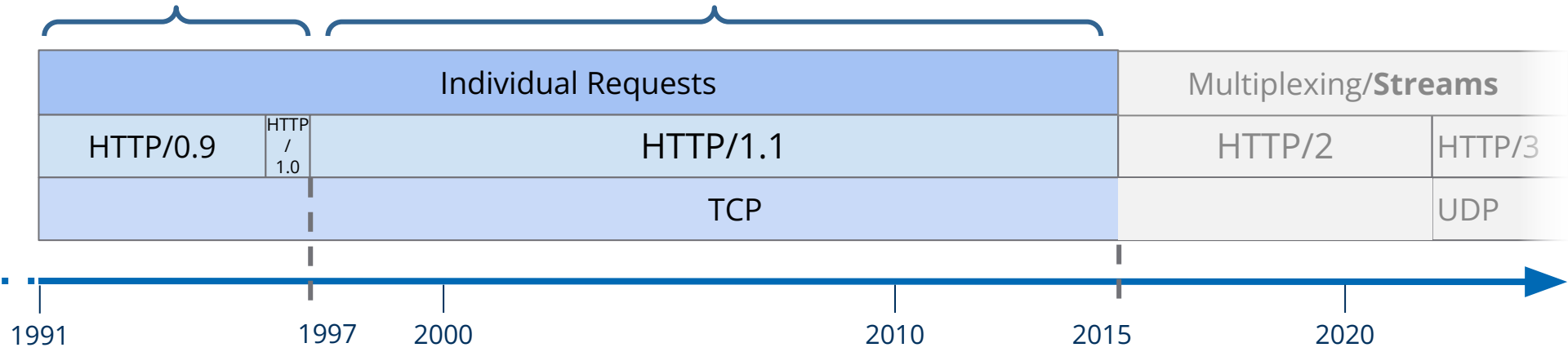


HTTP/1.1

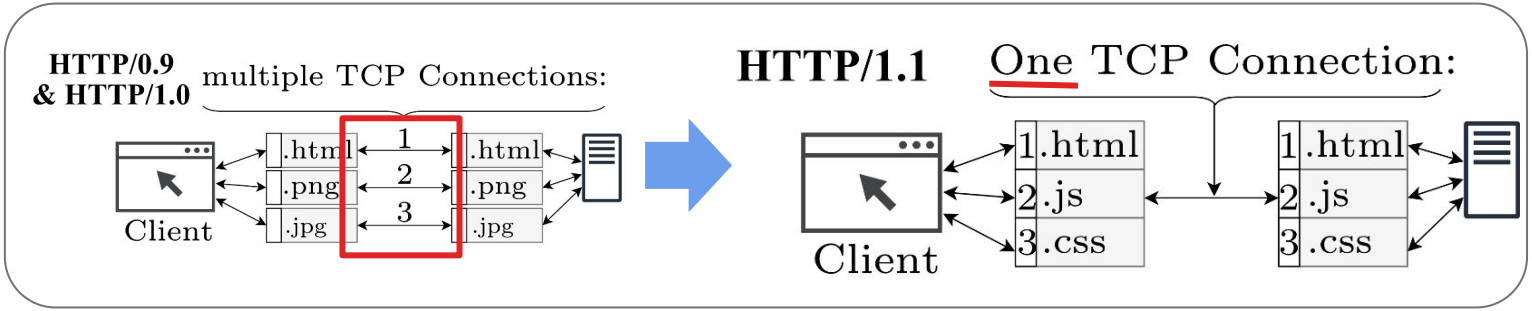
Penalty for every request: needed full TCP handshake

Less penalty for requests due to pipelining and keep-alive.
Still issues, due to buggy proxies and fixed request order

Less files
= faster loading time



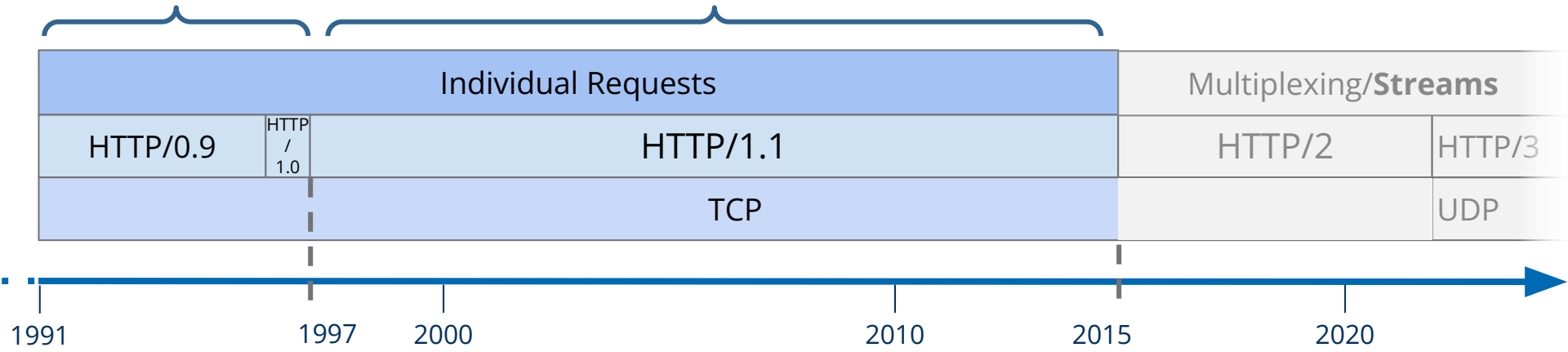
HTTP/1.1



Penalty for every request: needed full TCP handshake

Less penalty for requests due to pipelining and keep-alive.
Still issues, due to buggy proxies and fixed request order

Less files
= faster loading time

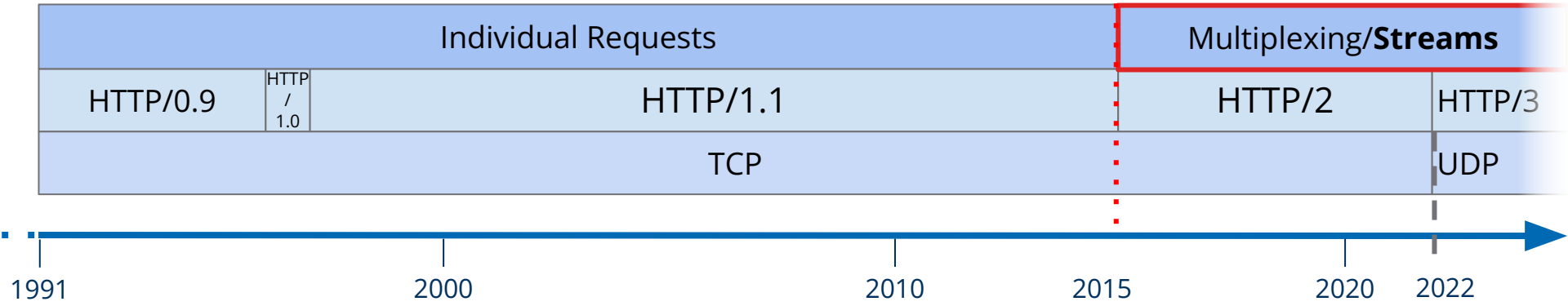


HTTP/2 & 3

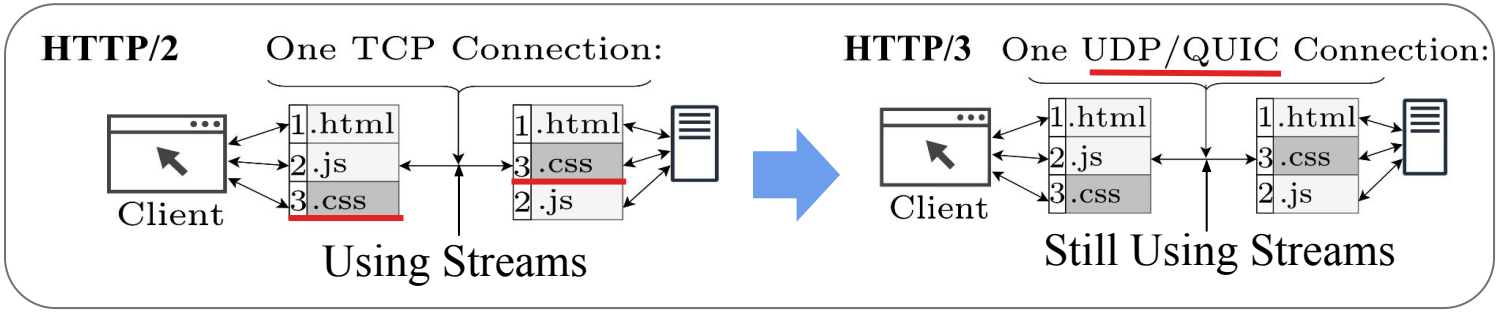
Penalty for every request: needed full TCP handshake

Less penalty for requests due to pipelining and keep-alive.
Still issues, due to buggy proxies and fixed request order

(almost) no penalty for requests due to 0-RTT, multiplexing and streams



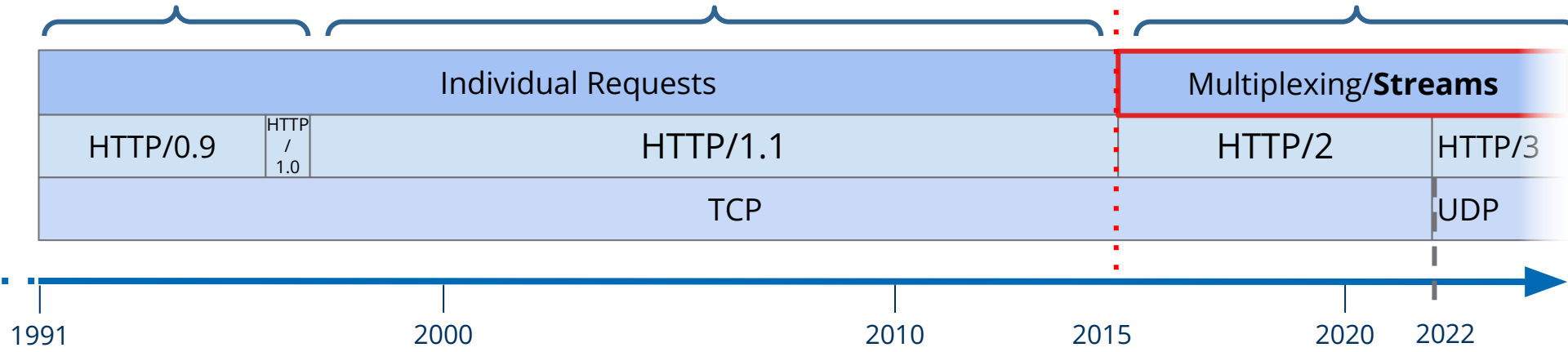
HTTP/2 & 3



Penalty for every request: needed full TCP handshake

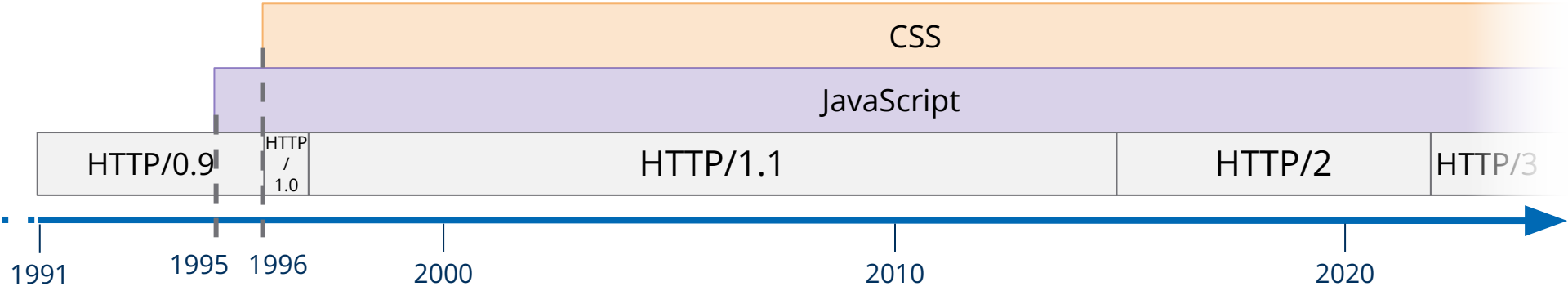
Less penalty for requests due to pipelining and keep-alive.
Still issues, due to buggy proxies and fixed request order

(almost) no penalty for requests due to 0-RTT, multiplexing and streams

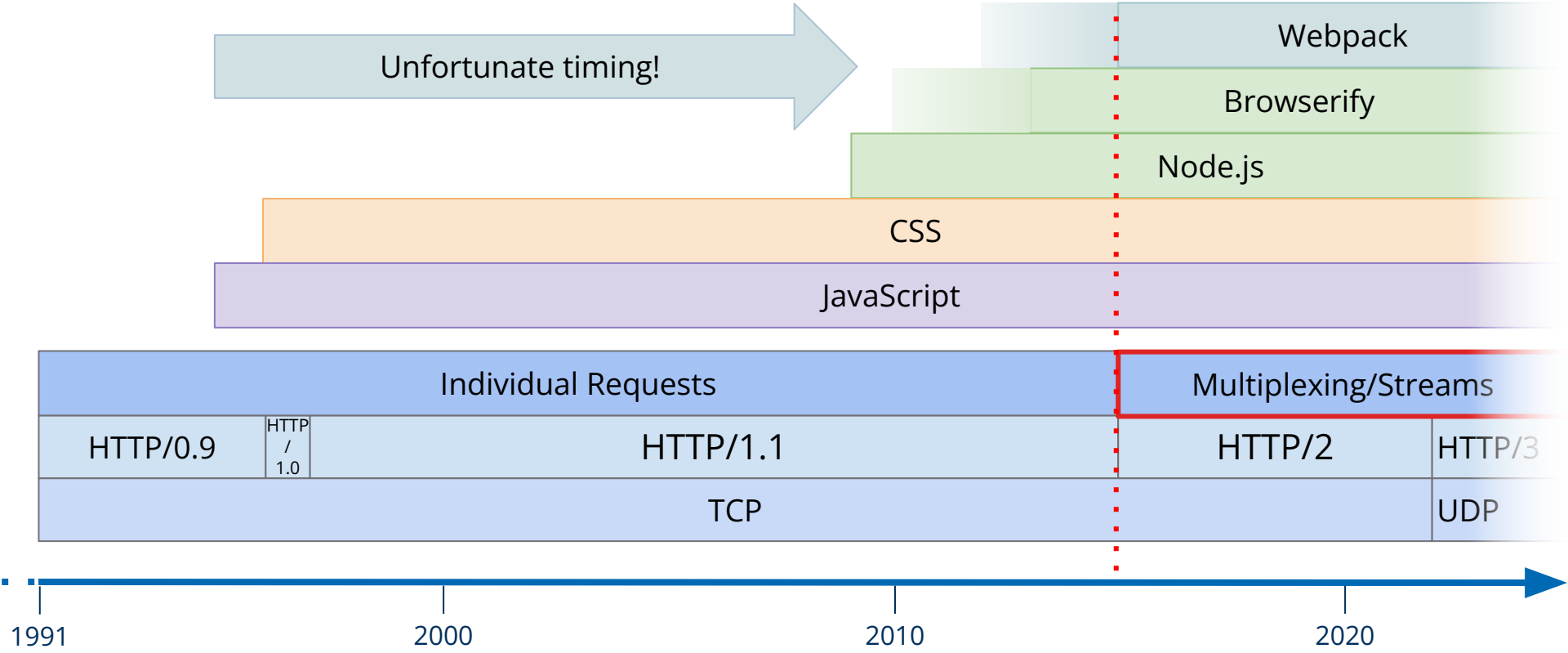


Content rendering

- **Web Content Creation History**:
  - HTTP limitations influenced the creation of web content
- **Beginning of Scripts and Styles**:
  - CSS was proposed by Håkon Wium Lie in 1994, published as CSS1 in **1996**
  - JavaScript emerged in **1995** with Netscape's addition of scripting to their browser
  - DOM specification followed in 1998



The problem: protocol vs. content



What if...

... we split everything?

What if...

... we split everything?
... and then stream the chunks?

Then we could…

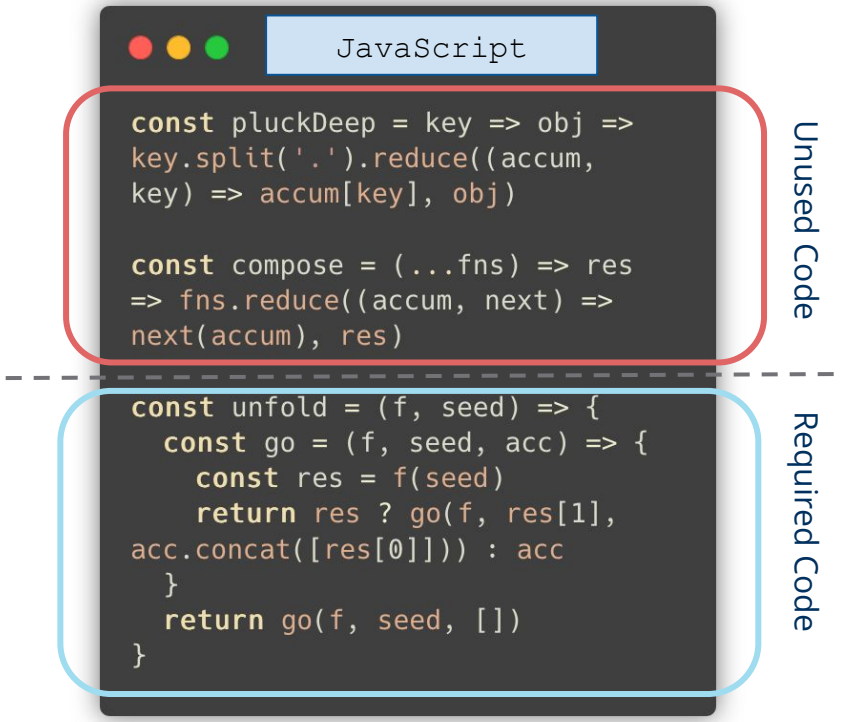
… improve loading speed

Then we could…

… improve loading speed
… use modern HTTP features

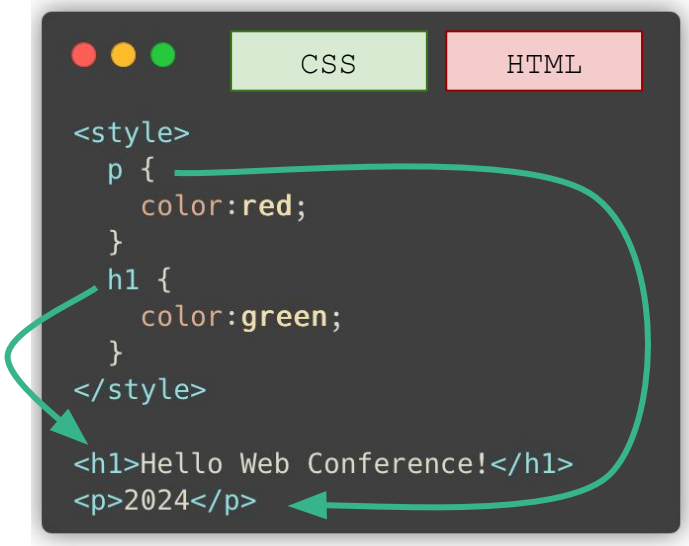
Challenge 1: Content usage detection

- Automatic detection of code usage and where to split the code is essential
- Techniques like Critical for CSS and tree shaking for JavaScript are available but have limitations

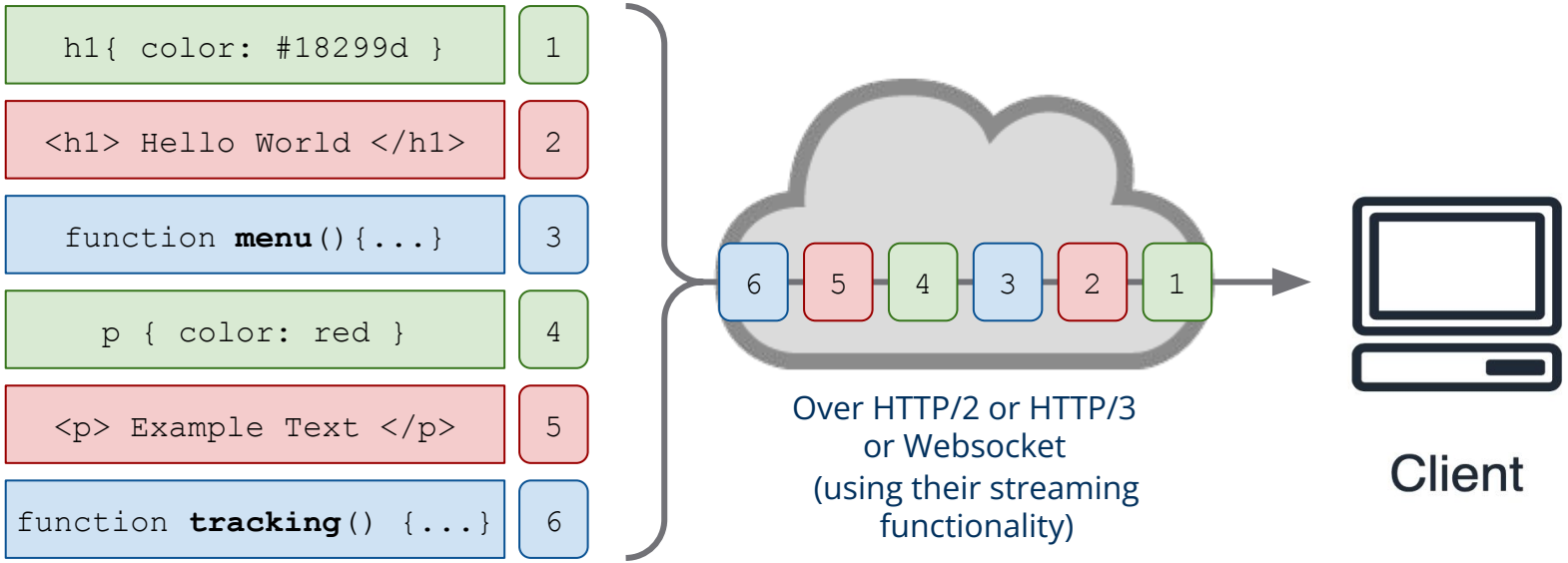


Challenge 2: Content usage location and order

- After detecting necessary code:
  - ordering and interleaving to minimize render-blocking
- **For CSS**: matching selectors with the DOM
- **For JavaScript**: open challenge
- Finally: Splitting after every code-chunk



Challenge 3: Streaming a web page



Proposed streaming in practice
An example, www.solarenergie.de

Example how a streamed web page would load

Image source: https://solarenergie.de , 31.05.2024

Summary

Summary

- **The Web is built on HTML, JavaScript and CSS**
  - All three components are render-blocking
- **Developers use libraries and frameworks bundled into one file**
  - Bundles are inefficient and slow down loading times
- **Performance can be measured by various tools, such as PageSpeed Insights**
- **Different improvements exist, but require developer effort**