

Frontend **Masters**



LET'S MAKE THE WEB FASTER!

# ADVANCED WEB PERFORMANCE

MAXIMILIANO FIRTMAN



# MAXIMILIANO FIRTMAN

MOBILE+WEB DEVELOPER

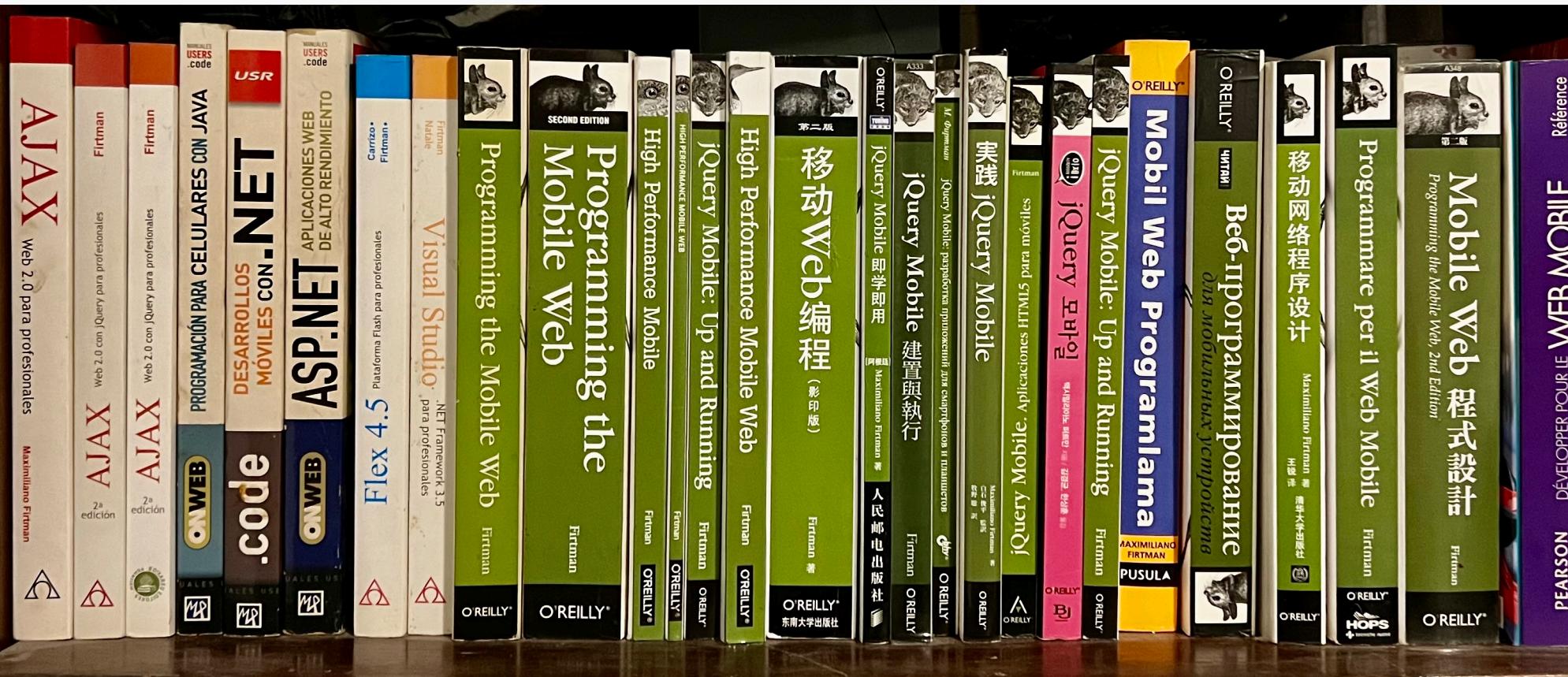
HTML since 1996

JavaScript since 1998

AUTHOR

Authored 13 books and +70 courses

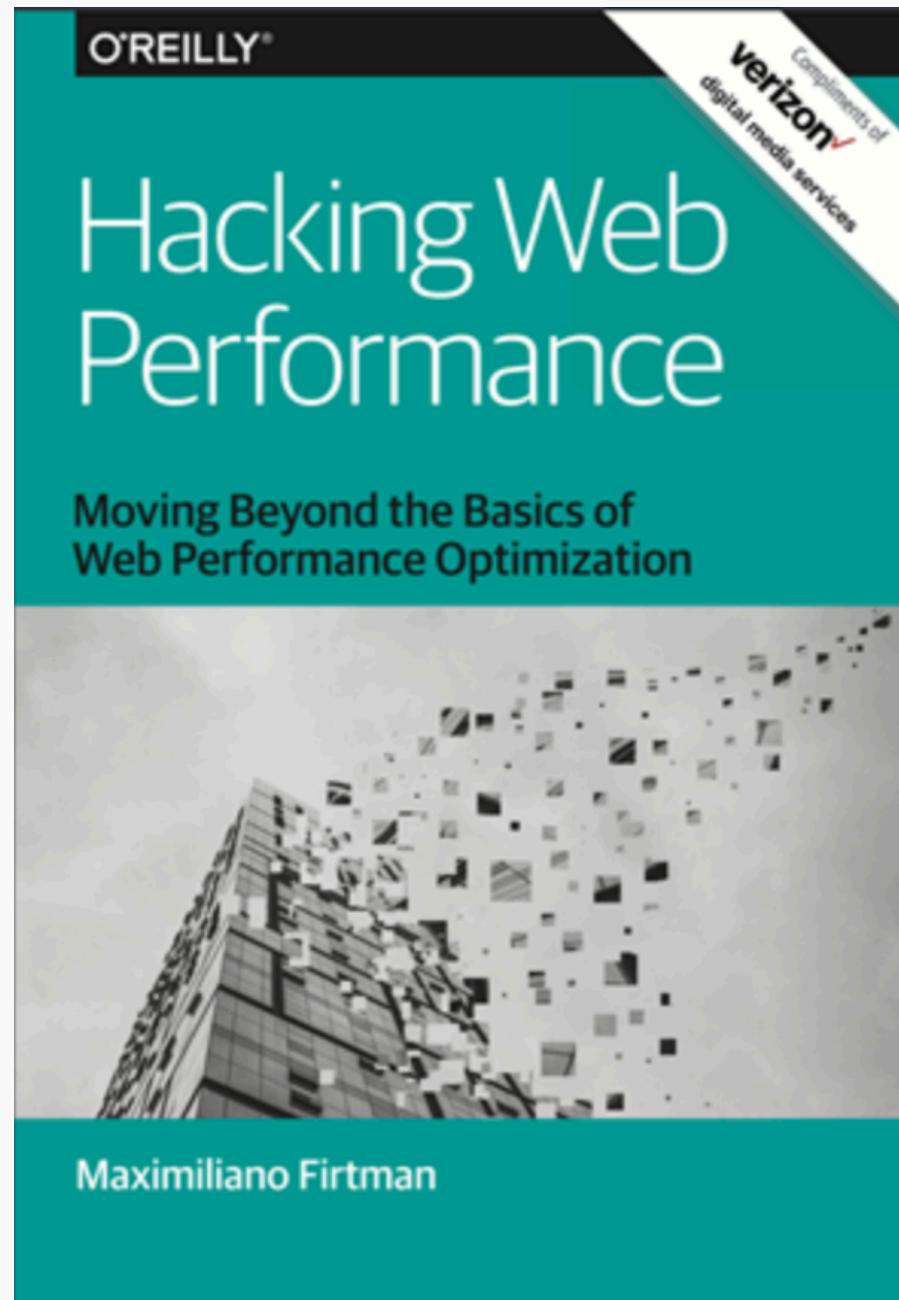
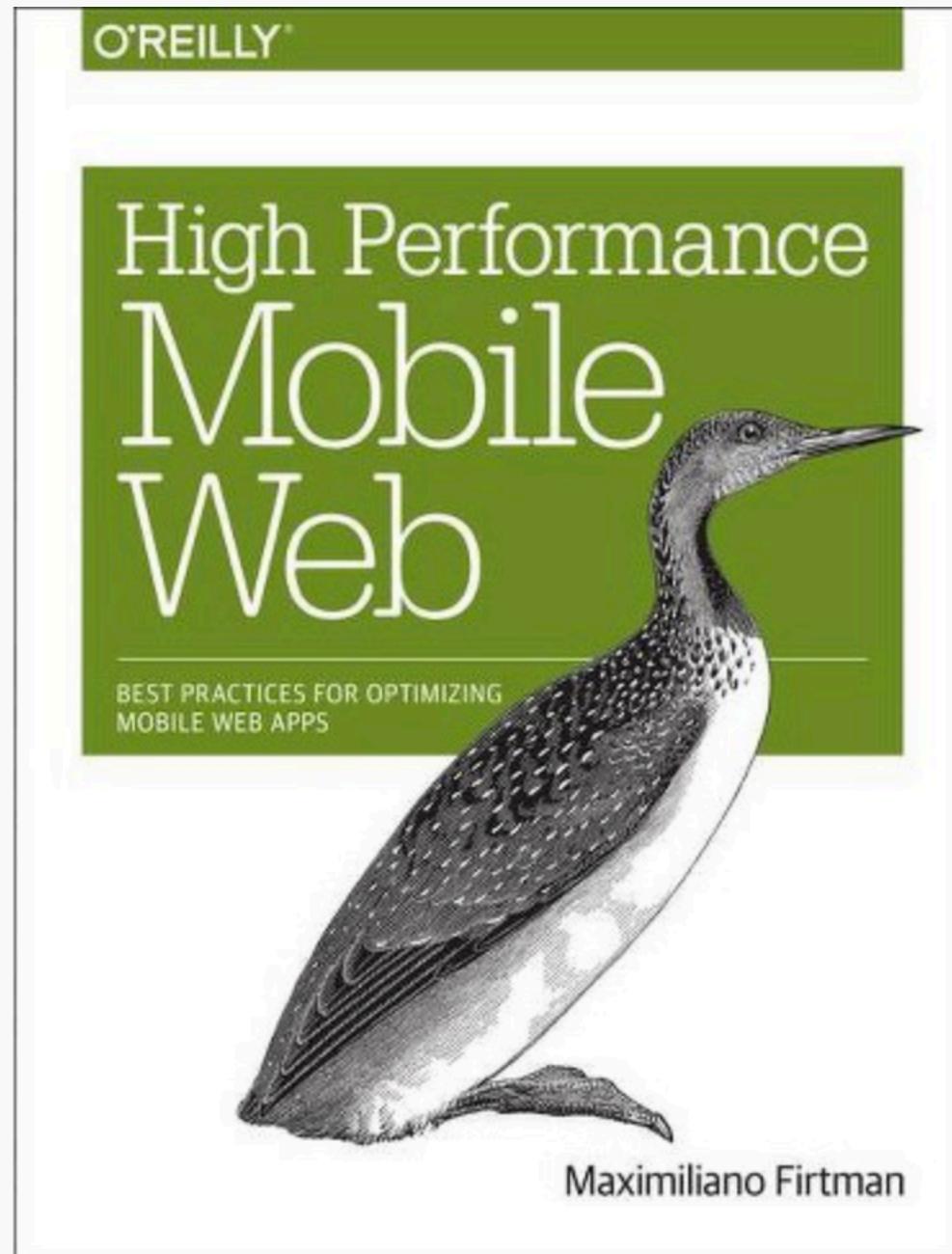
Published +150 webapps



@FIRT · FIRT.DEV



# MAXIMILIANO FIRTMAN



@FIRT · FIRT.DEV

I have 2 goals...

Show you new tricks



**Make you feel bad**



# What we'll cover

The Problem

Metrics

Tools

Charts and Diagrams

Understanding Browsers

Basic Optimizations

Hacking Performance

Performance APIs



# The Problem



You already know that...



<http://www.flickr.com/photos/steenslag/22689920/>  
borrowed from Steve Souders



Picture from Simon Howden [freedigitalphotos.net](http://freedigitalphotos.net)



Picture from Simon Howden freedigitalphotos.n

It's mostly frontend  
responsibility!

I know  
you are already...

You are already...

Optimizing the network  
transfer

You are already...

Working with TLS and  
HTTP/2

You are already...

**CSS as Appetizer**

You are already...

# JavaScript as Dessert

You are already...

# Optimizing Images

You are already...

# Defining a Policy for HTTP Cache

You are already...

# Using Service Workers

.. right?



I'm watching you

**What's the problem  
then?**

What's the problem then?

## Time to Interactive



HTTP Archive with Lighthouse

What's the problem then?



1 2 3 4 5 6  
7 8 9 10 11  
12 12.5

You are already...

Average time to load a mobile  
landing page

**22 seconds**

Research by [thinkwithgoogle.com](http://thinkwithgoogle.com)

You are already...

as page load time went from one second to 10 seconds, the probability of a mobile visitor

**Increased 123%**

Research by thinkwithgoogle.com



# Underestimating Mobile

iOS and Android

# Safari and Chrome?

# browsers with market share

Let's see in action





**ScientiaMobile\_CTO**  
@Scientia\_CTO

...

Replying to @firt @auchenberg and @scientiamobile

Hey Max, you are right. We have some of that data.  
According to our data, 7.6% of global smartphone traffic  
comes from the Facebook App. This and more in our  
MOVR report:



[Mobile Overview Report \(MOVR\) | ScientiaMobile](#)

ScientiaMobile's Overview Report (MOVR) provides timely device data on smartphones, tablets, and desktop devices ...

[scientiamobile.com](http://scientiamobile.com)

Using Cellular  
Networks!

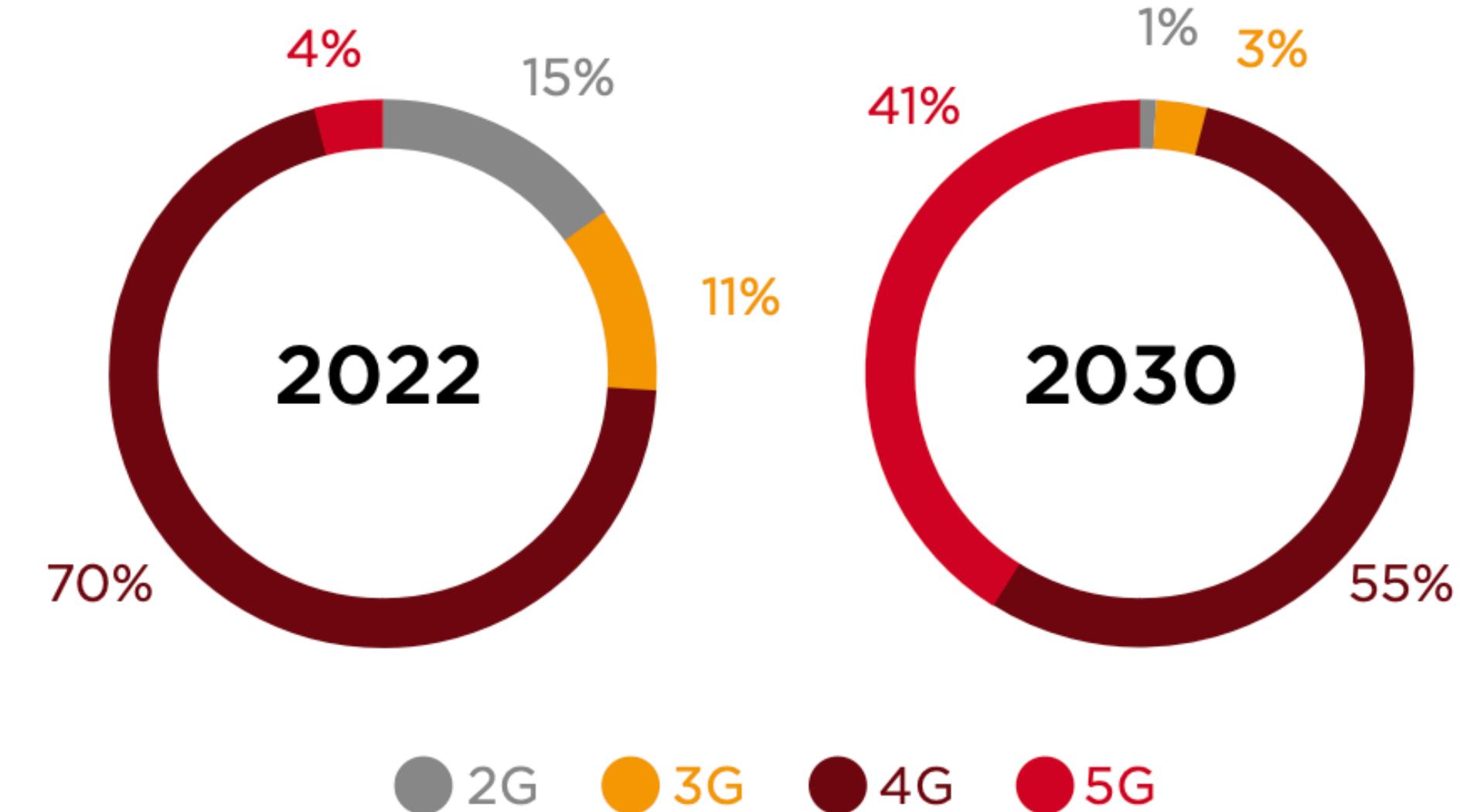
*We have 5G! We don't need to  
worry about performance...*



## Asia Pacific



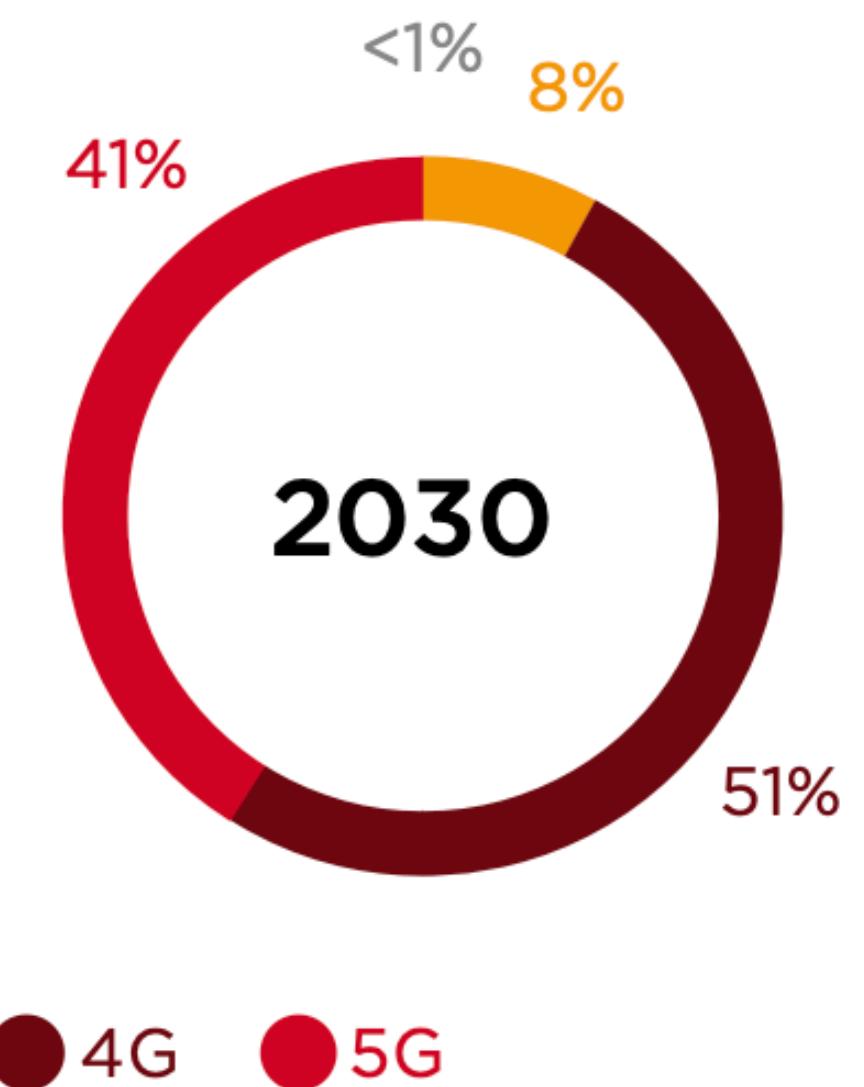
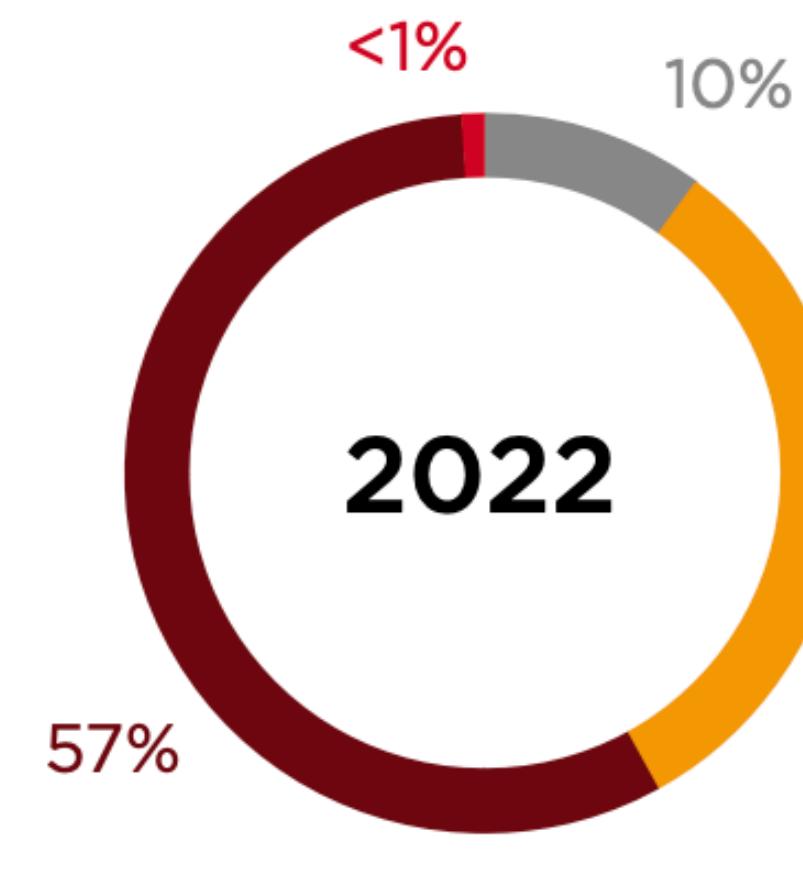
## Technology mix



**CIS**



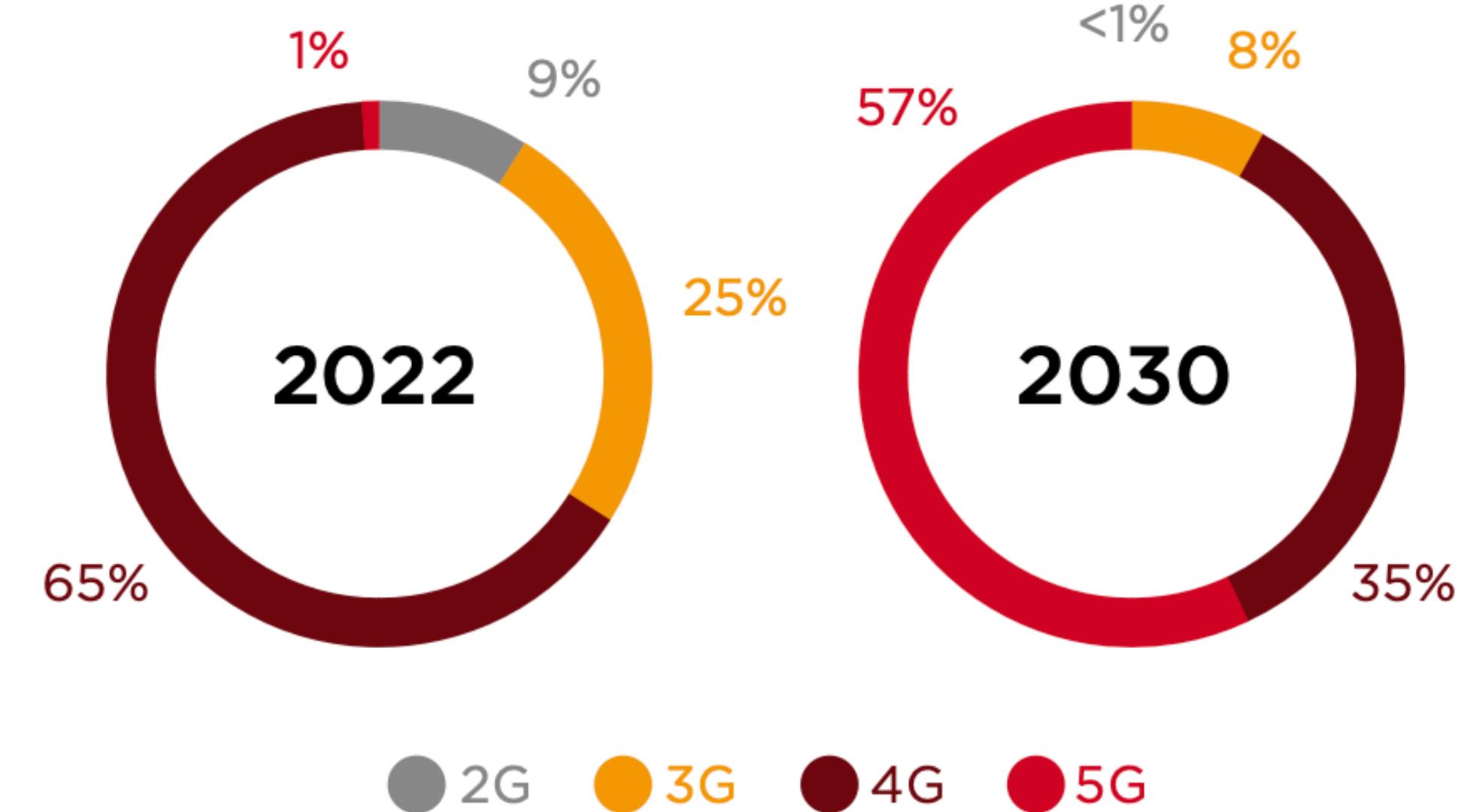
### Technology mix



## Latin America



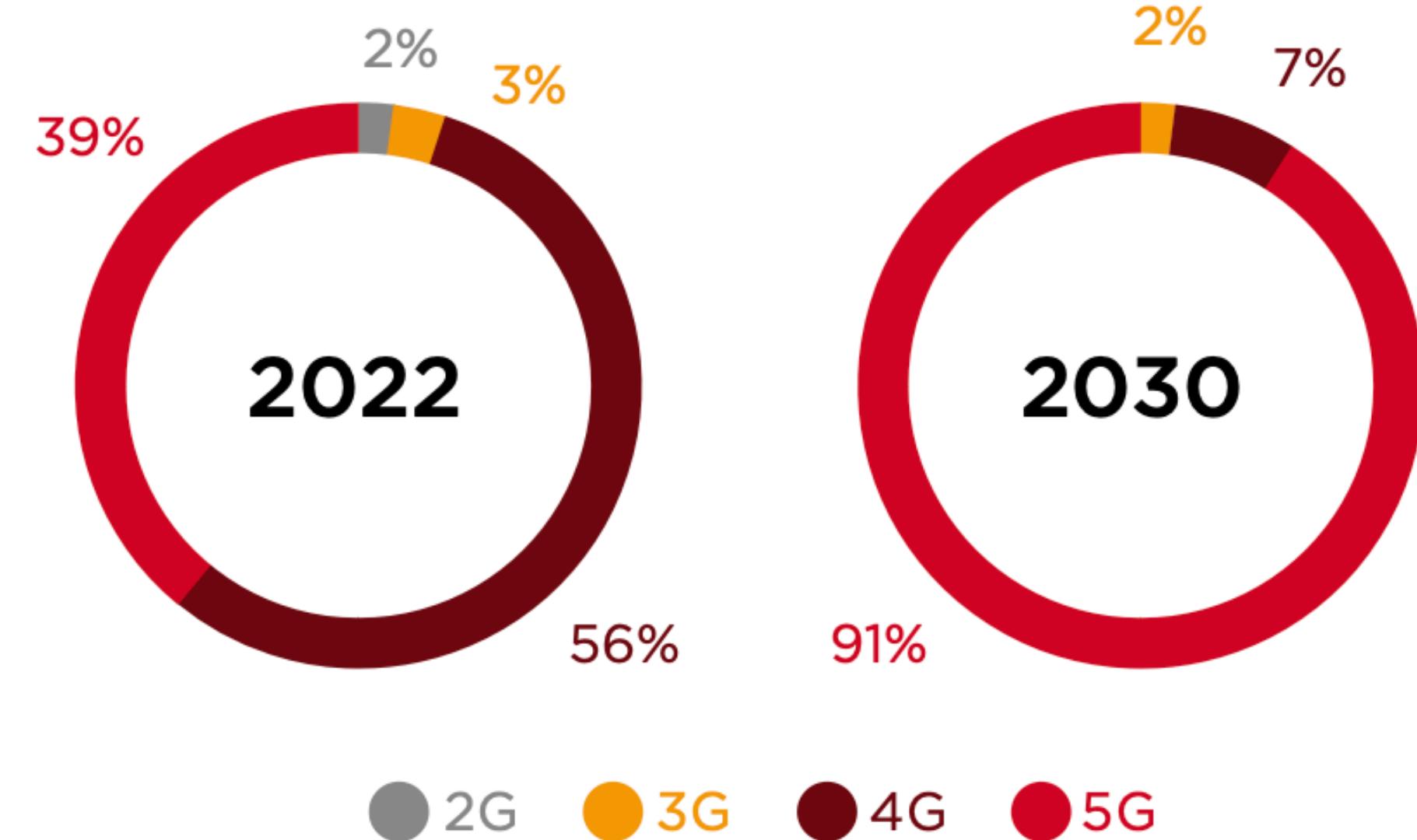
### Technology mix



## North America

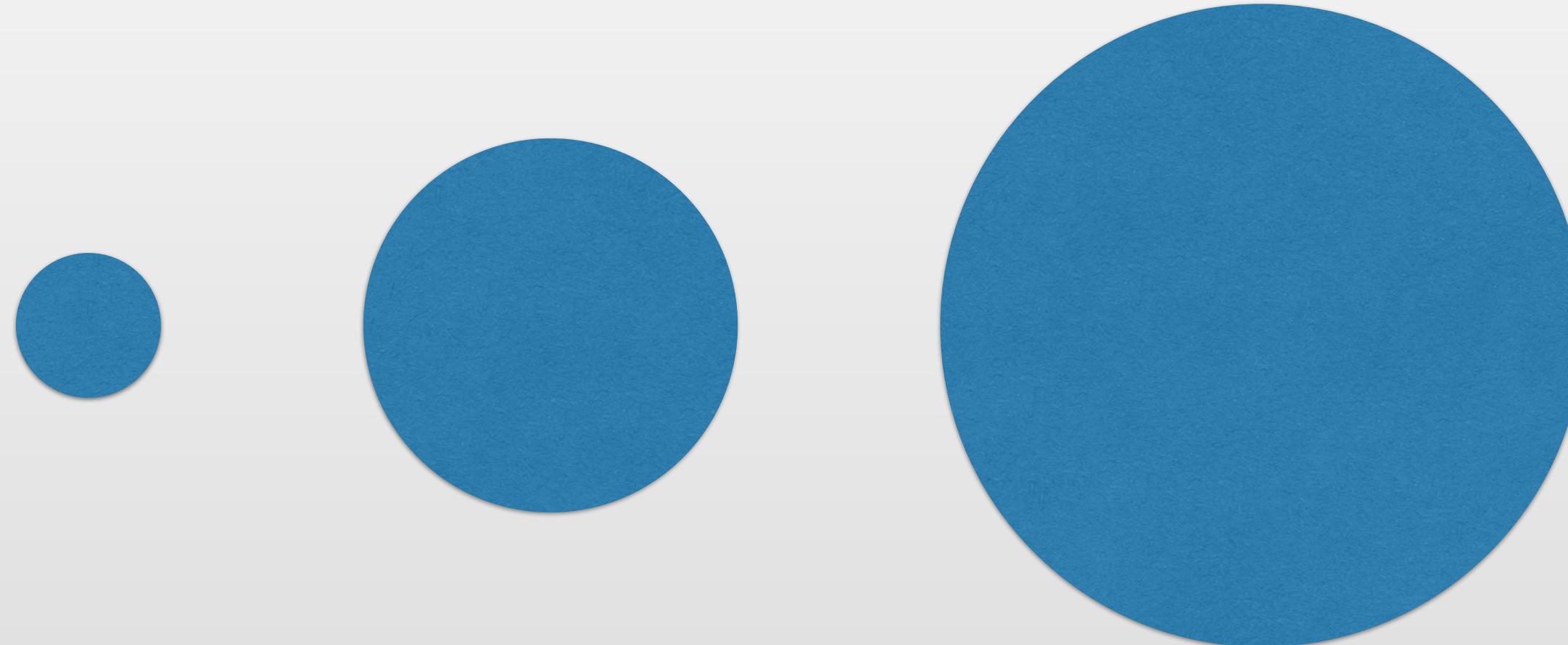


## Technology mix



cellular networks

# Bandwidth





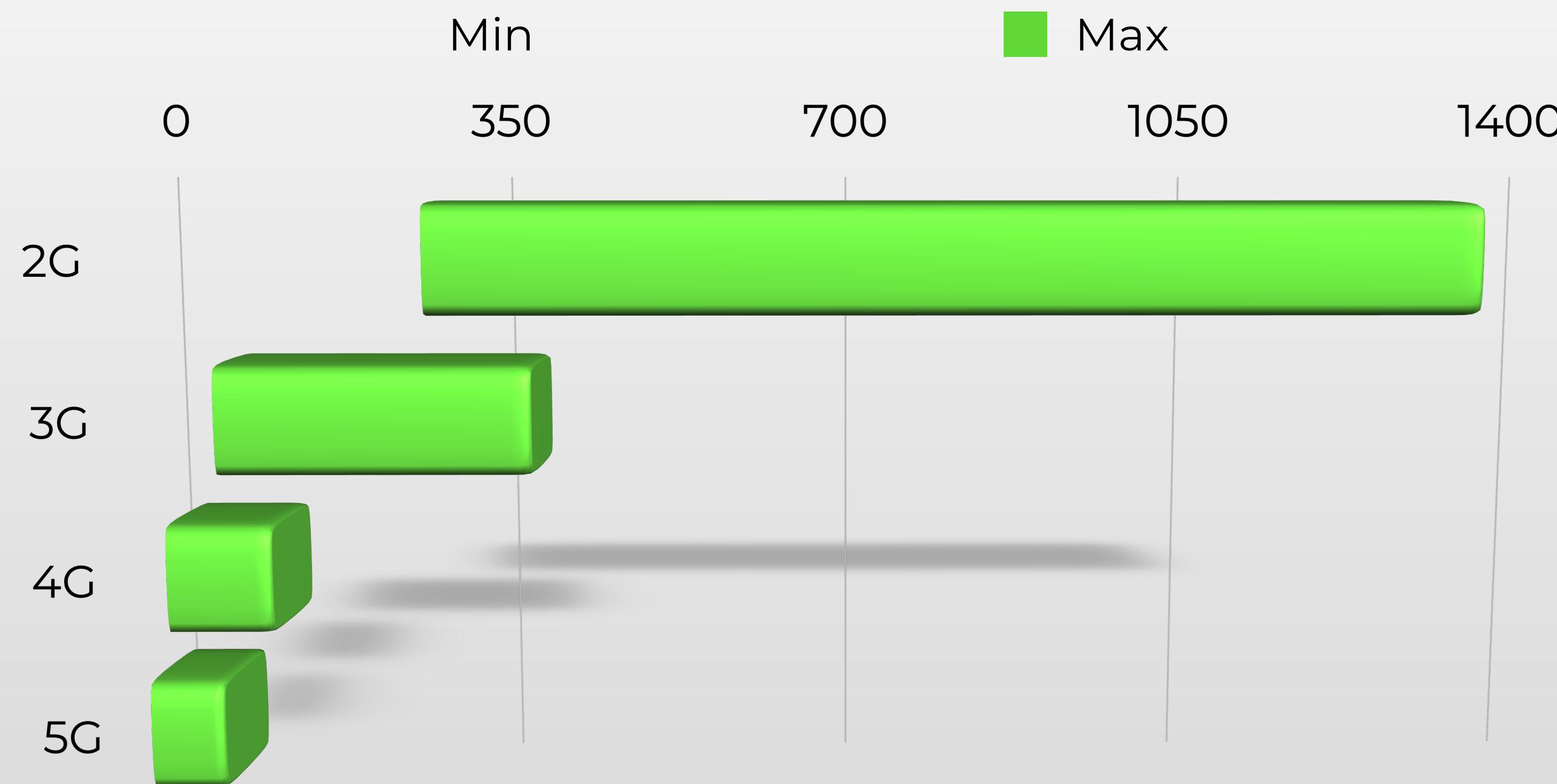
## DEFINITION

# Latency

Time delay between when a data packet is sent and when it is received, expressed in milliseconds.

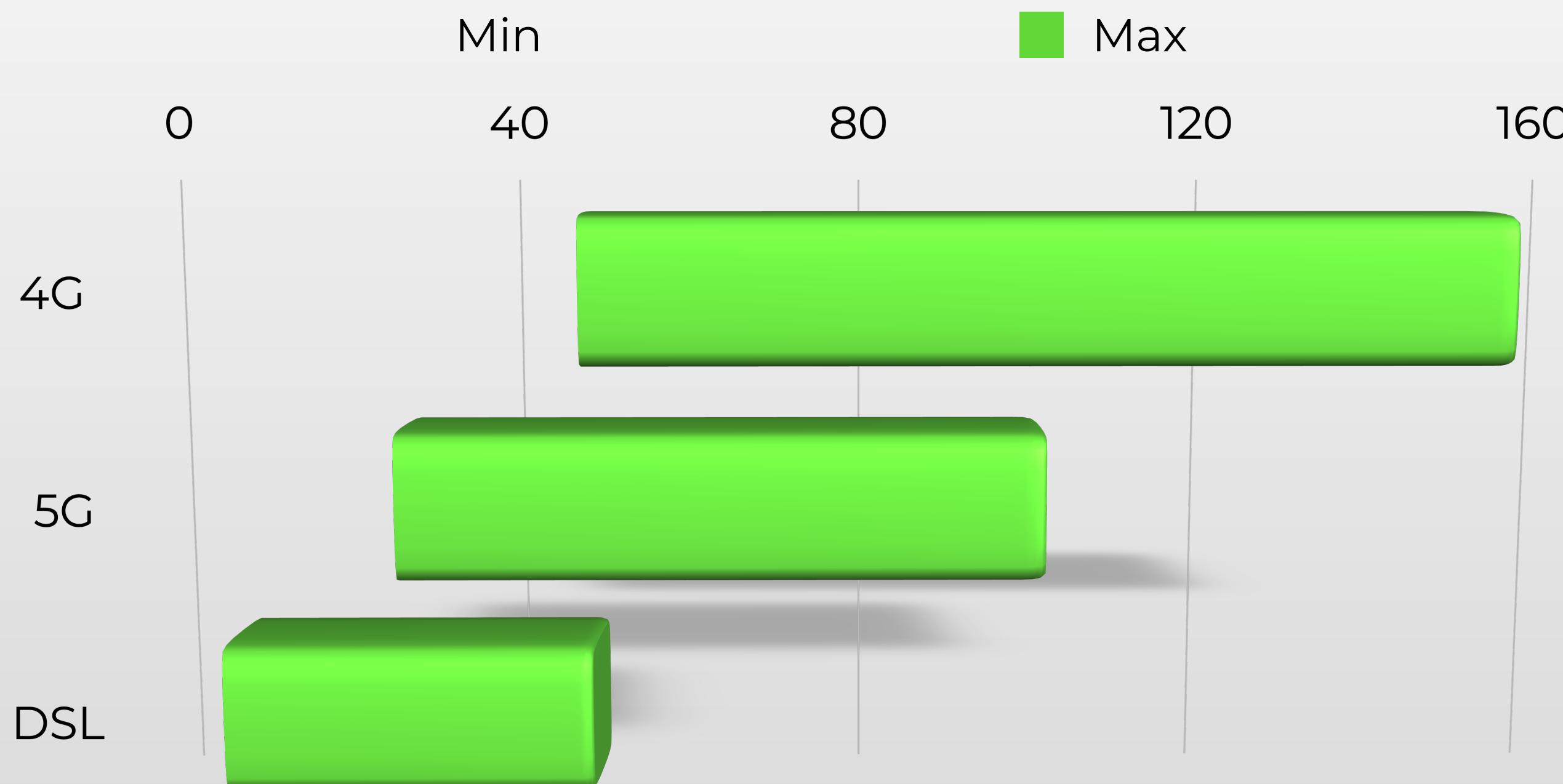
# Cellular Networks

## RTT - latency



# Cellular Networks

## RTT - latency





## IMPORTANT

"Latency hardly impacts smartphone user experience in advanced 5G and 4G networks"

Verizon



## WARNING

Not every country has  
"advanced" 4G/5G and 28%  
of users is still on 3G or  
worse



**WARNING**

We still have a  
performance issue on the  
Mobile Web

# Cellular Devices

**CPU and GPU 8x slower**



We still have a problem

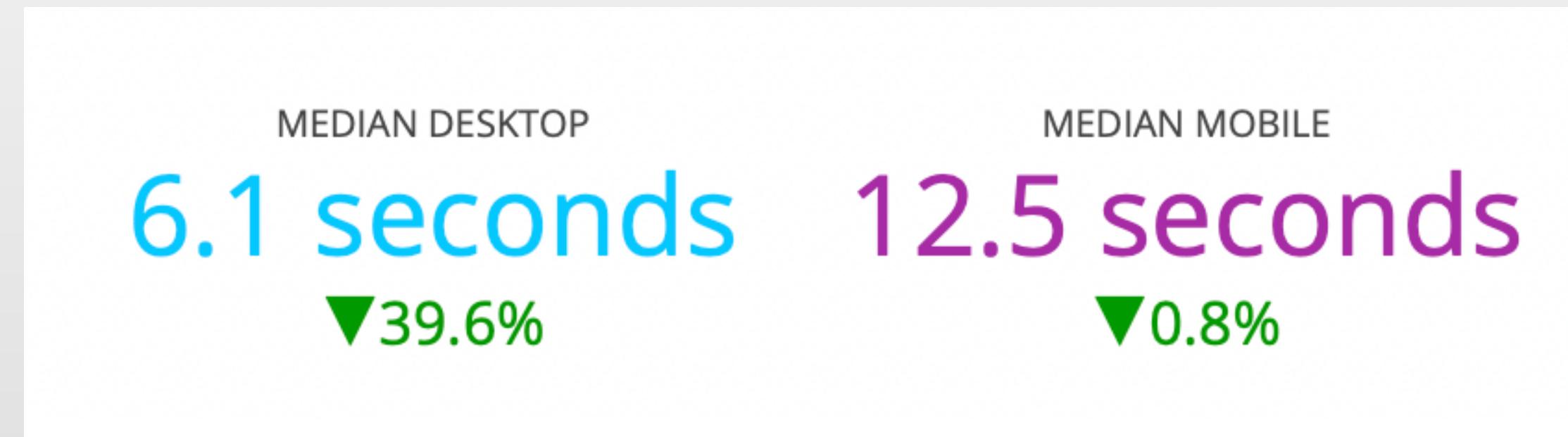
## Time to Interactive



HTTP Archive with Lighthouse

We still have a problem

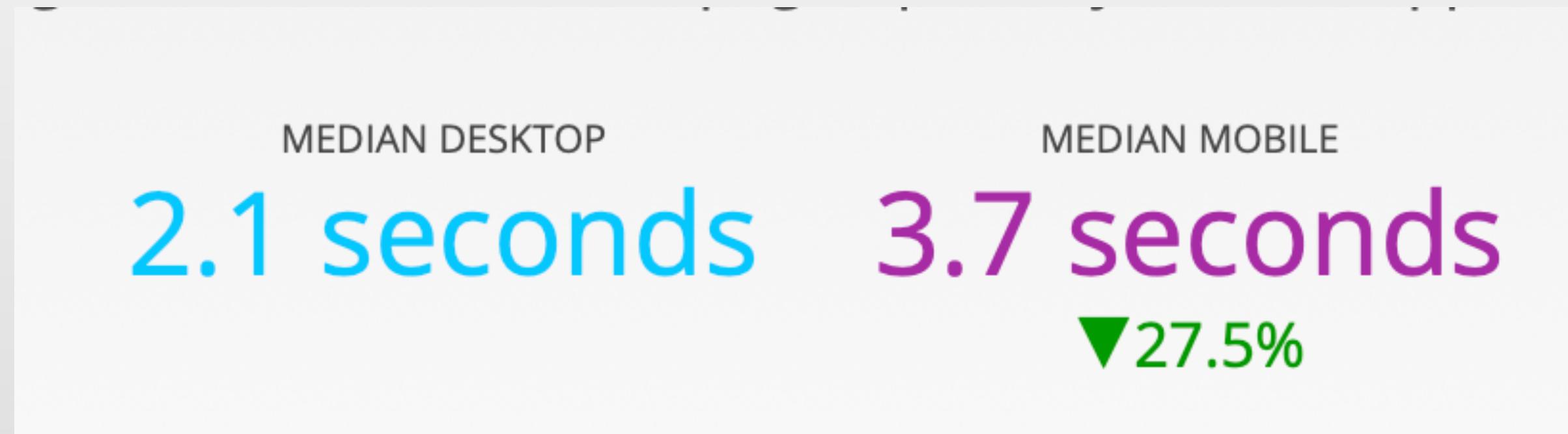
## Time to Interactive



HTTP Archive with Lighthouse

We still have a problem

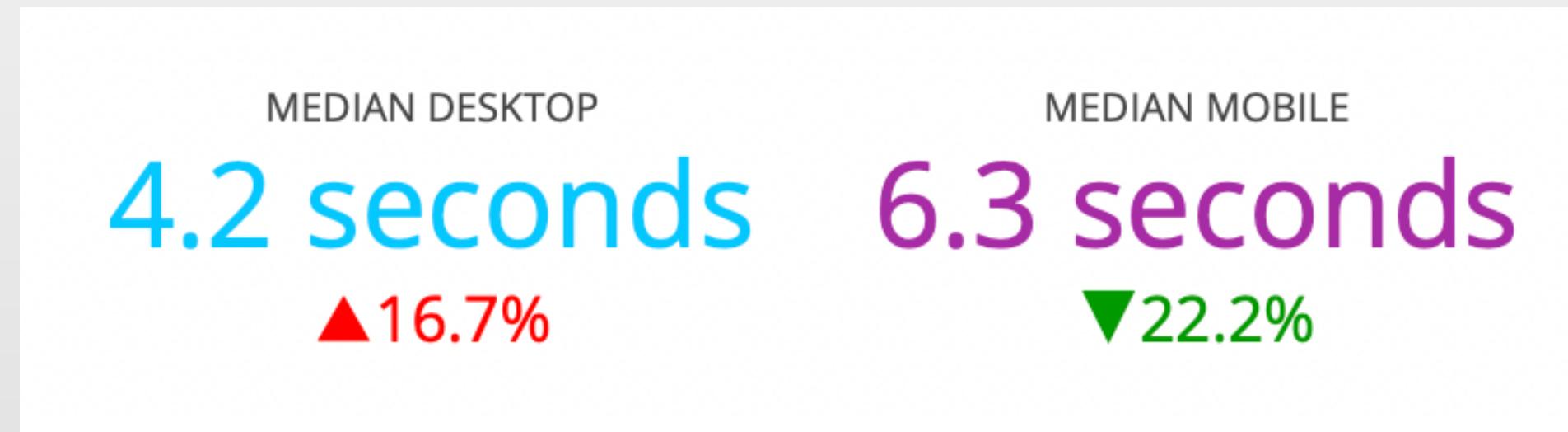
## First Contentful Paint



HTTP Archive with Lighthouse

We still have a problem

## Speed Index



HTTP Archive with Lighthouse

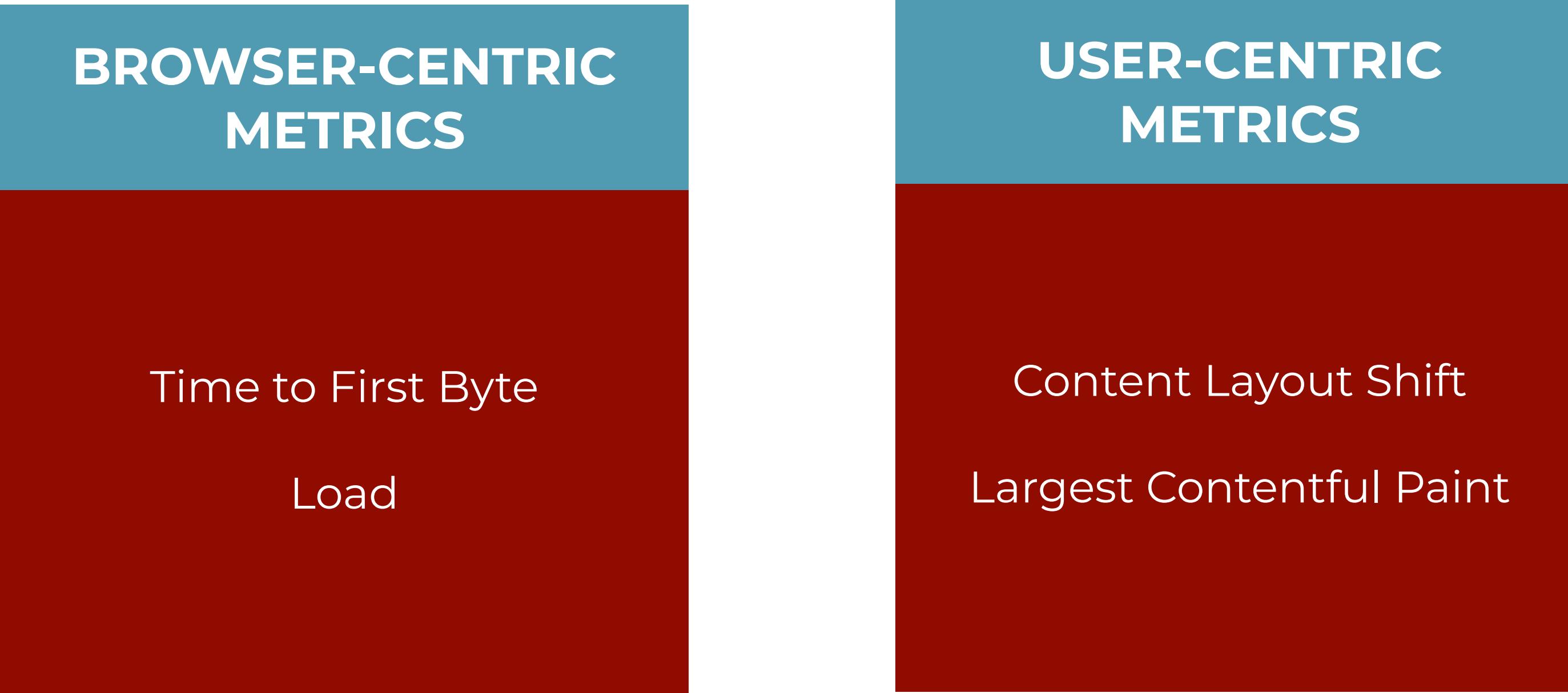


# Metrics and Tools



# Metrics to the Rescue

# We can't improve what we can't measure



# Browser-centric metrics

## Time to First Byte

(TTFB)

Browser-centric metrics

**First Paint**

Browser-centric metrics

# Page Load

 **WARNING**

Browser-centric metrics  
are not so important for  
Web Performance  
Optimization

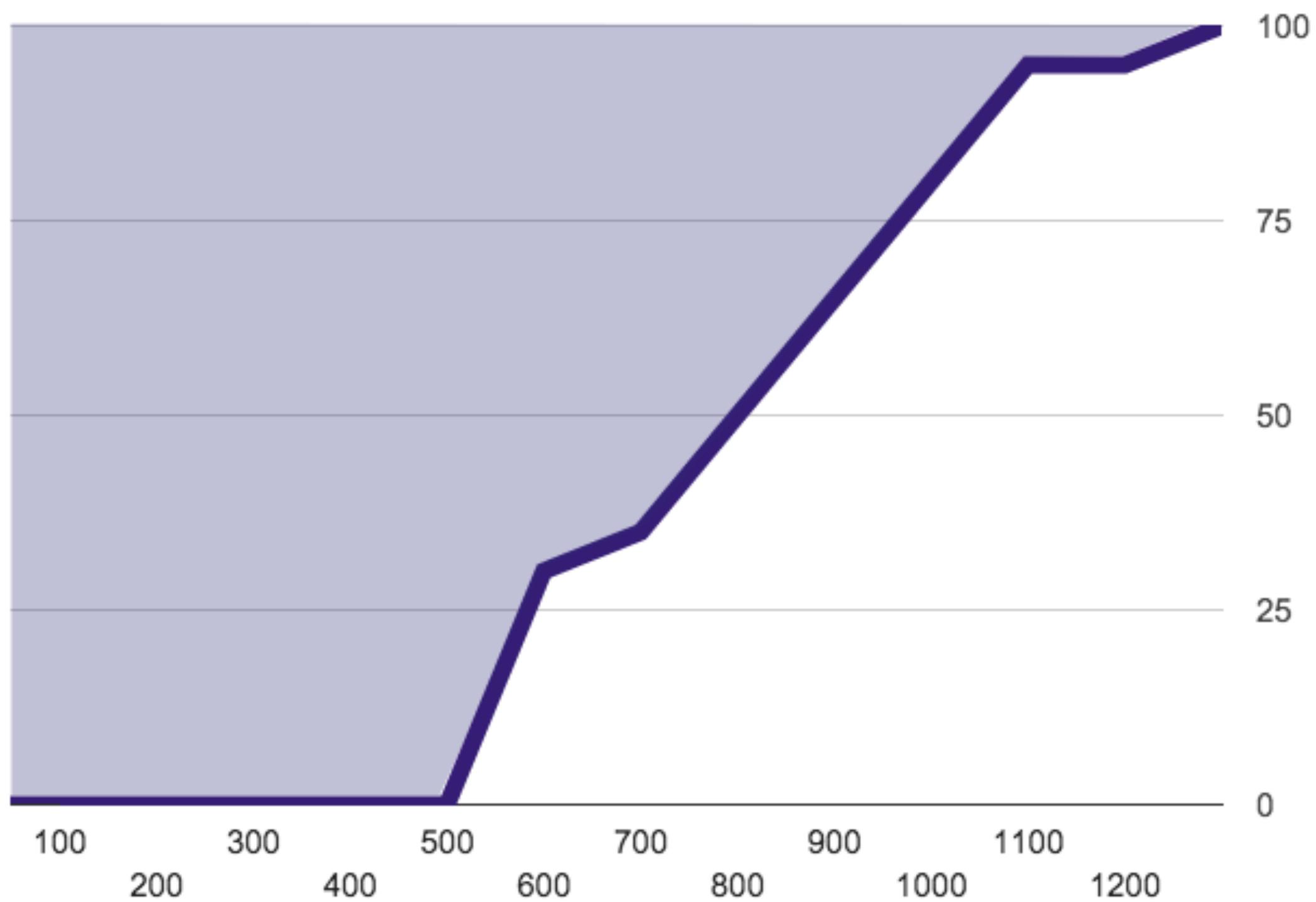
# User-centric metrics

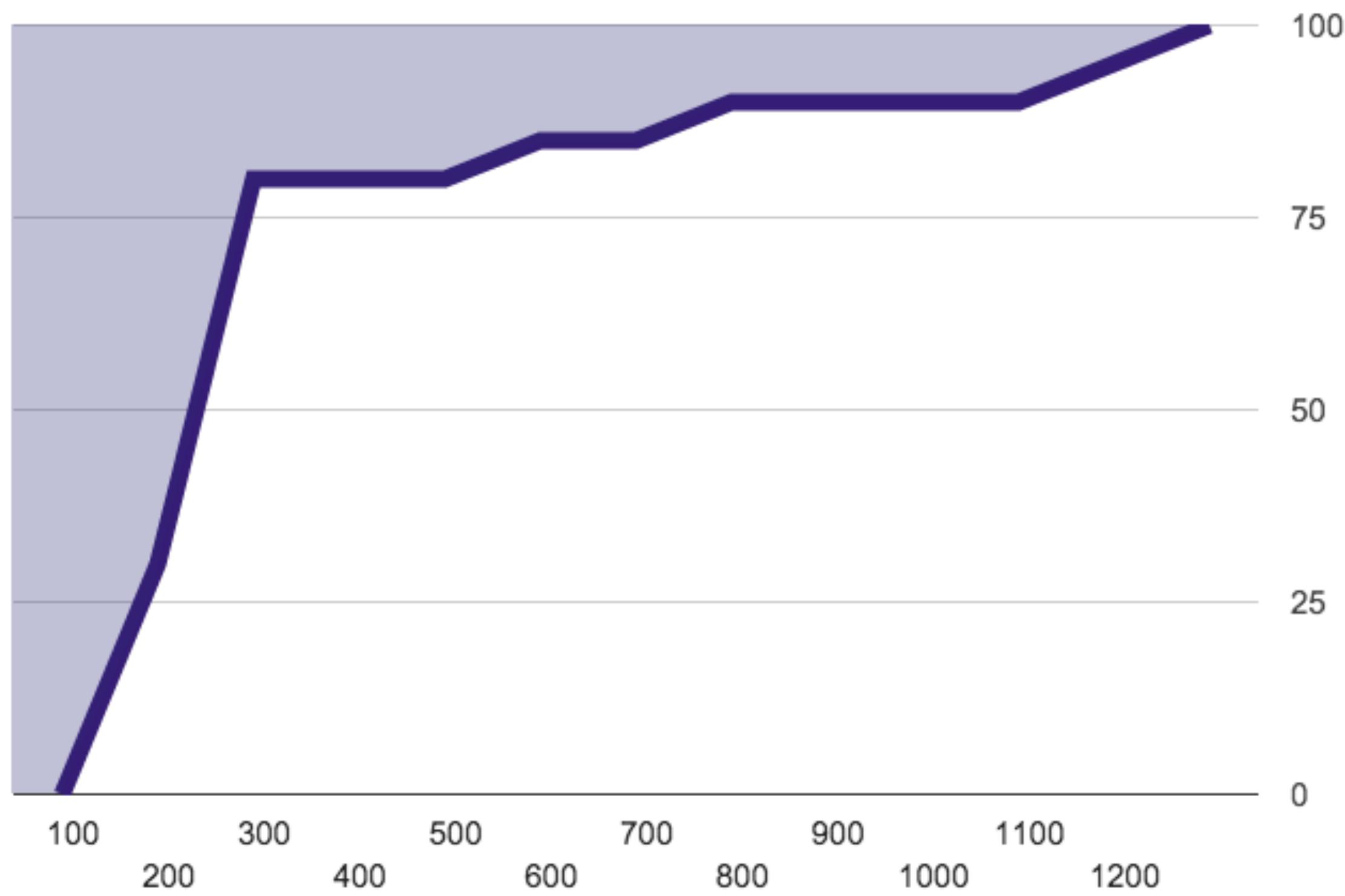
User-centric metrics

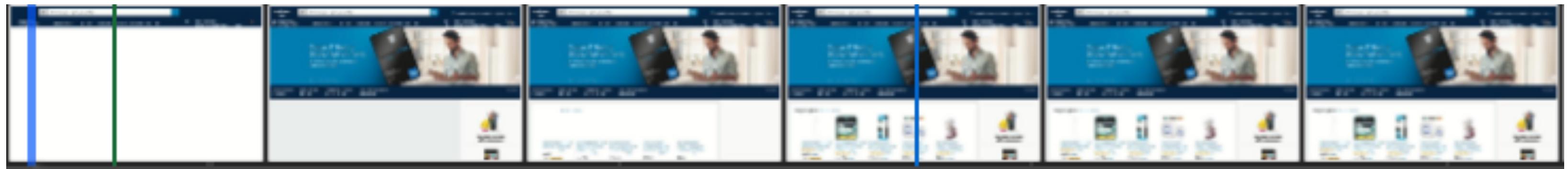
**First Interactive**

# User-centric metrics

## **Speed index**







User-centric metrics

**Time To Interactive**

User-centric metrics

**Largest Contentful**

**Paint (LCP)**

# User-centric metrics

## **Custom Metric**

# Core Web Vitals

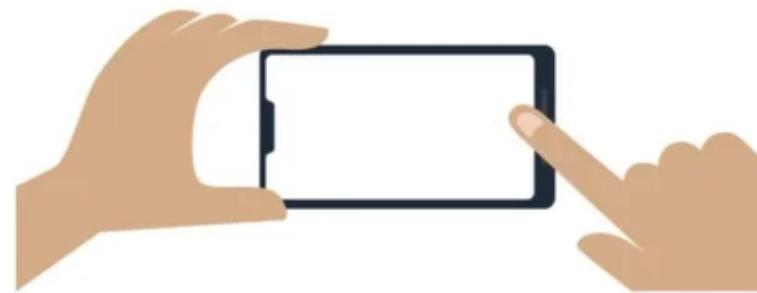


Calculated over the 75th percentile  
over mobile and web

# Core Web Vitals



(Loading)



(Interactivity)



(Visual Stability)

## LCP

Largest Contentful Paint



## FID

First Input Delay



## CLS

Cumulative Layout Shift



User-centric metric

**Largest Contentful Paint**

**LCP** - initial load

User-centric metric

**First Input Delay**

**FID - interactivity**

User-centric metric

**Cumulative Layout Shift**

**CLS - visual stability**

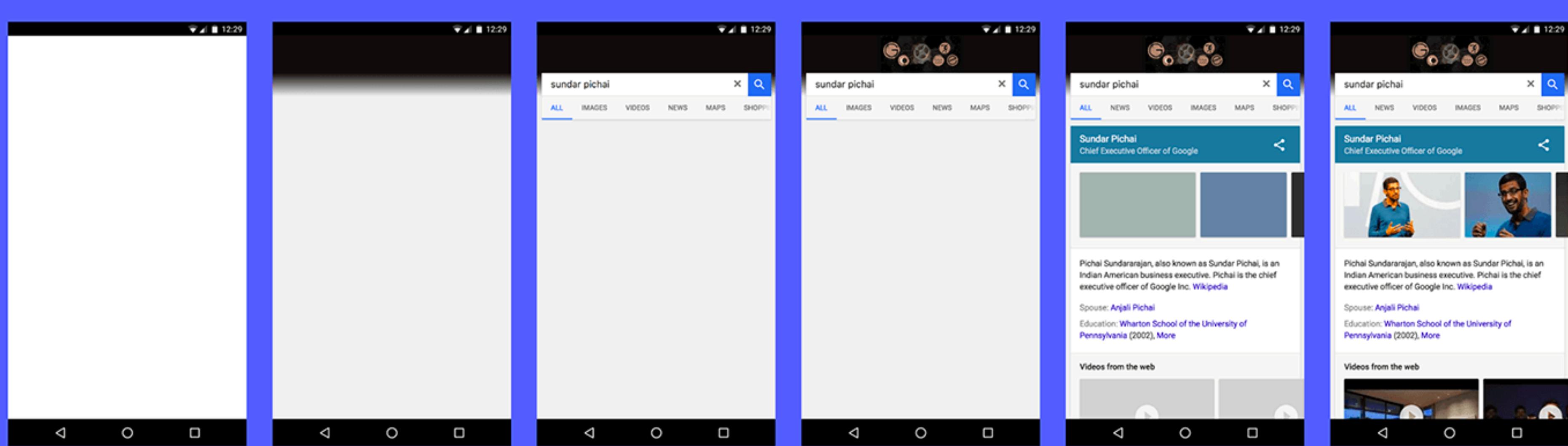
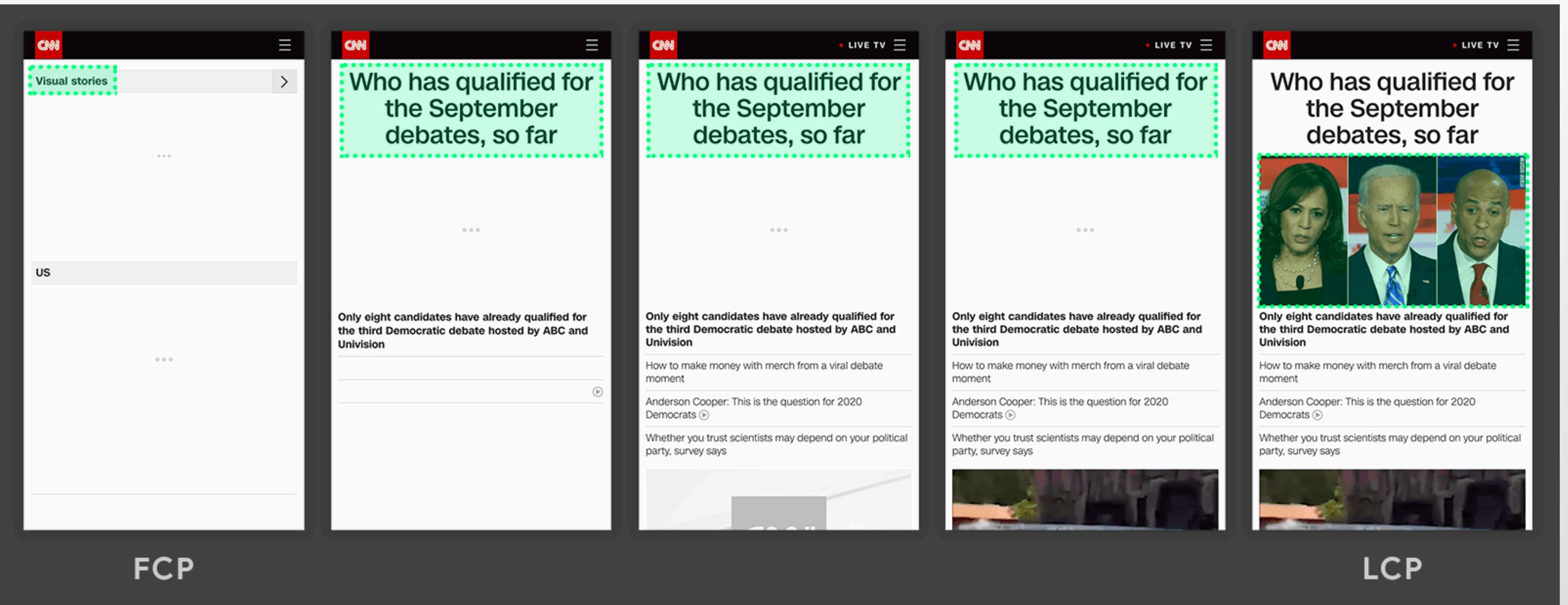


Image from [developers.google.com](https://developers.google.com)



... there are more metrics in

Web Vitals

User-centric metric

**Interaction to Next Paint**

**responsiveness**

*But hey, my website is*

*fast*



**IMPORTANT**

How fast is fast?

We have to define our  
performance budgets

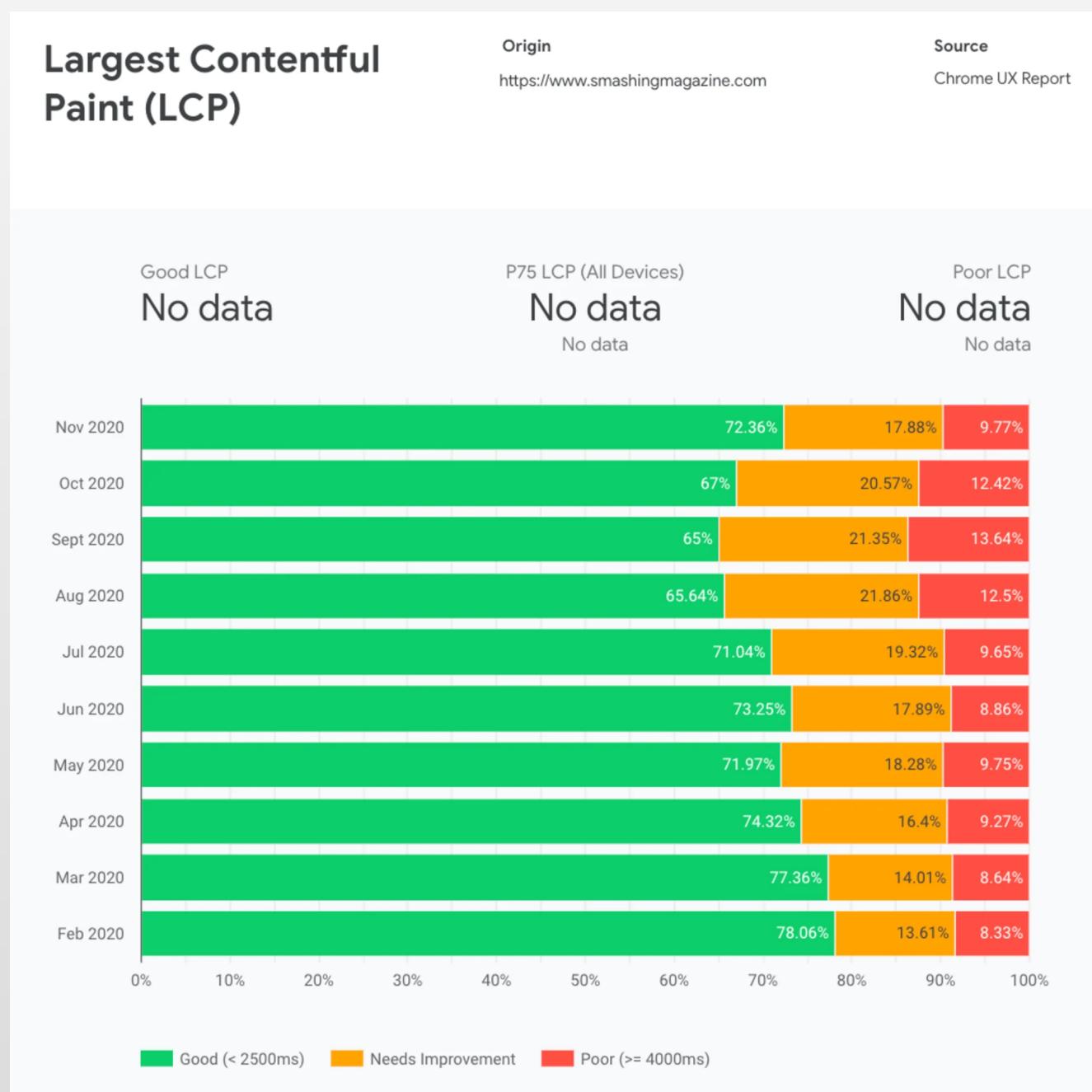
# Goals

LCP: 2.5s - 4.0s

FID: 100ms - 300ms

CLS: 0.1 - 0.25

# Did you checked Chrome UX?



**Origin Summary** — Over the previous 28-day collection period, the aggregate experience of all pages served from this origin **passes** the [Core Web Vitals](#) assessment. To view suggestions tailored to each page, analyze individual page URLs.

● First Contentful Paint (FCP) **1.7 s**



● First Input Delay (FID) [16 ms](#)



● Largest Contentful Paint (LCP) [2.1 s](#)



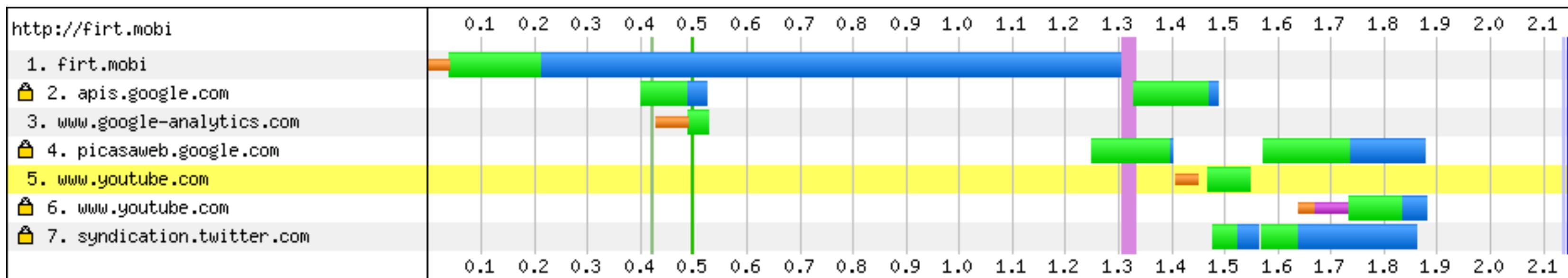
● Cumulative Layout Shift (CLS) [0.01](#)



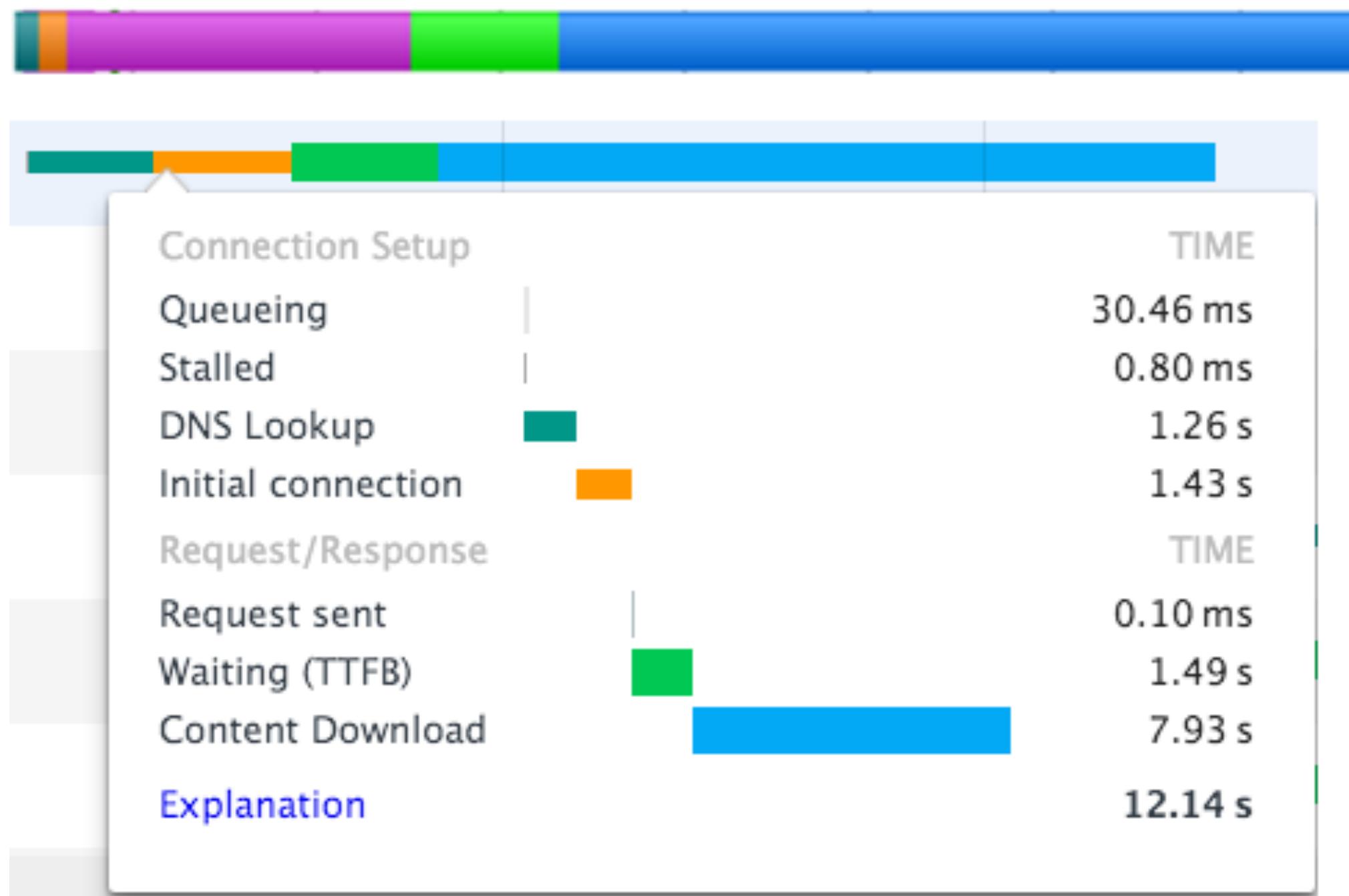


# Tools and Charts

# Waterfall Chart

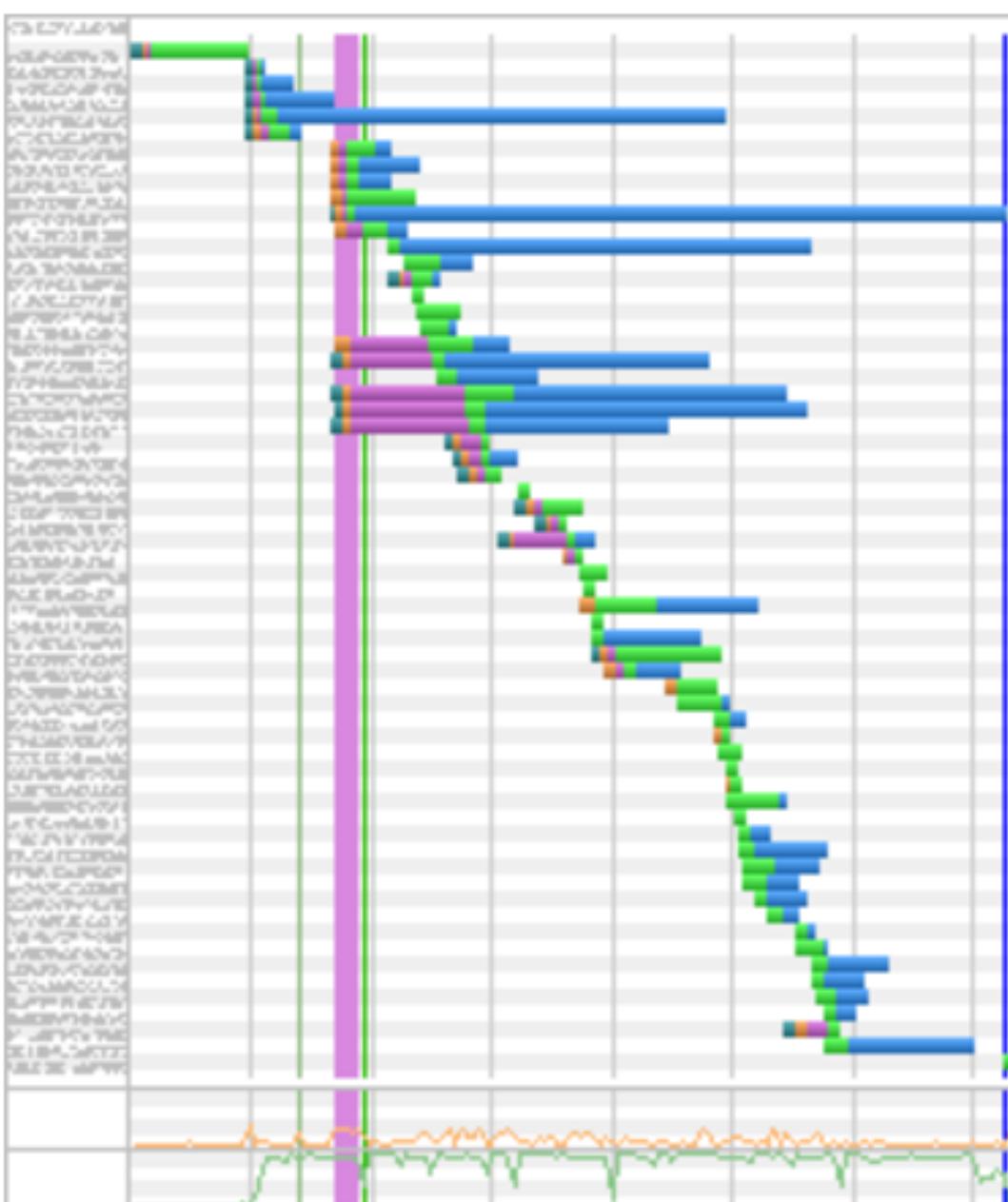


# Waterfall Chart Color bars

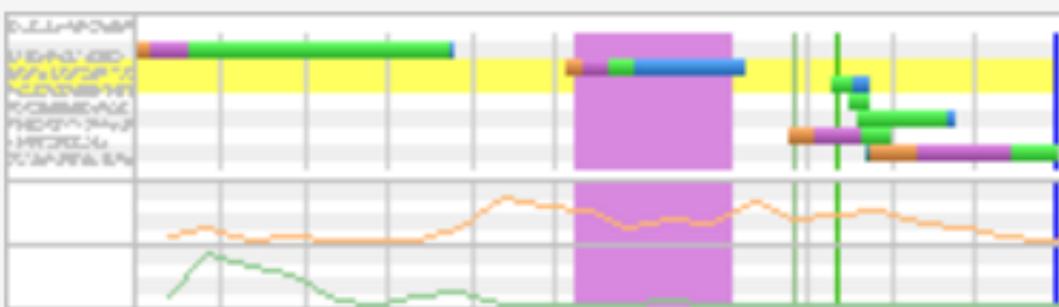


## Waterfall

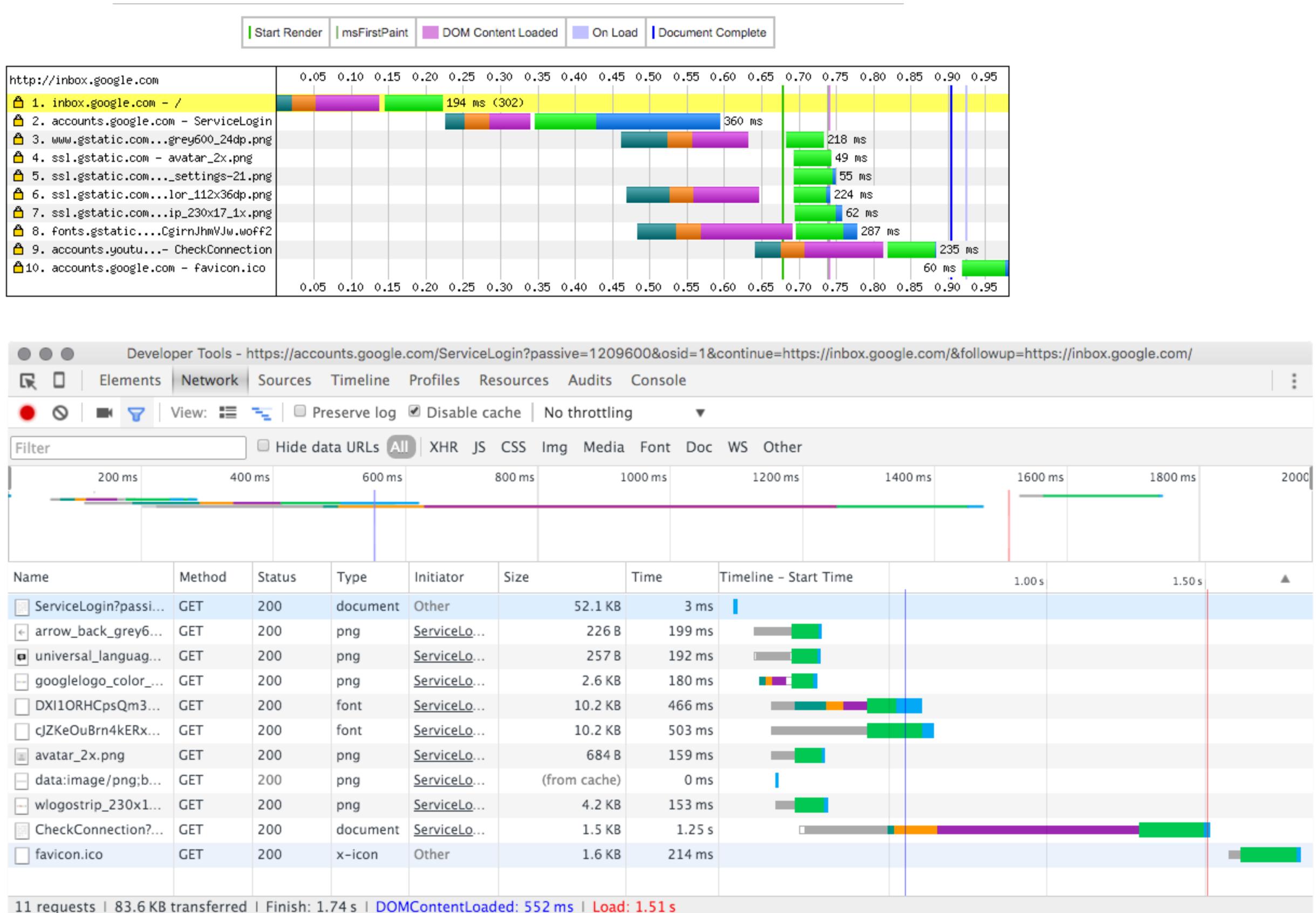
First View  
(14.521s)



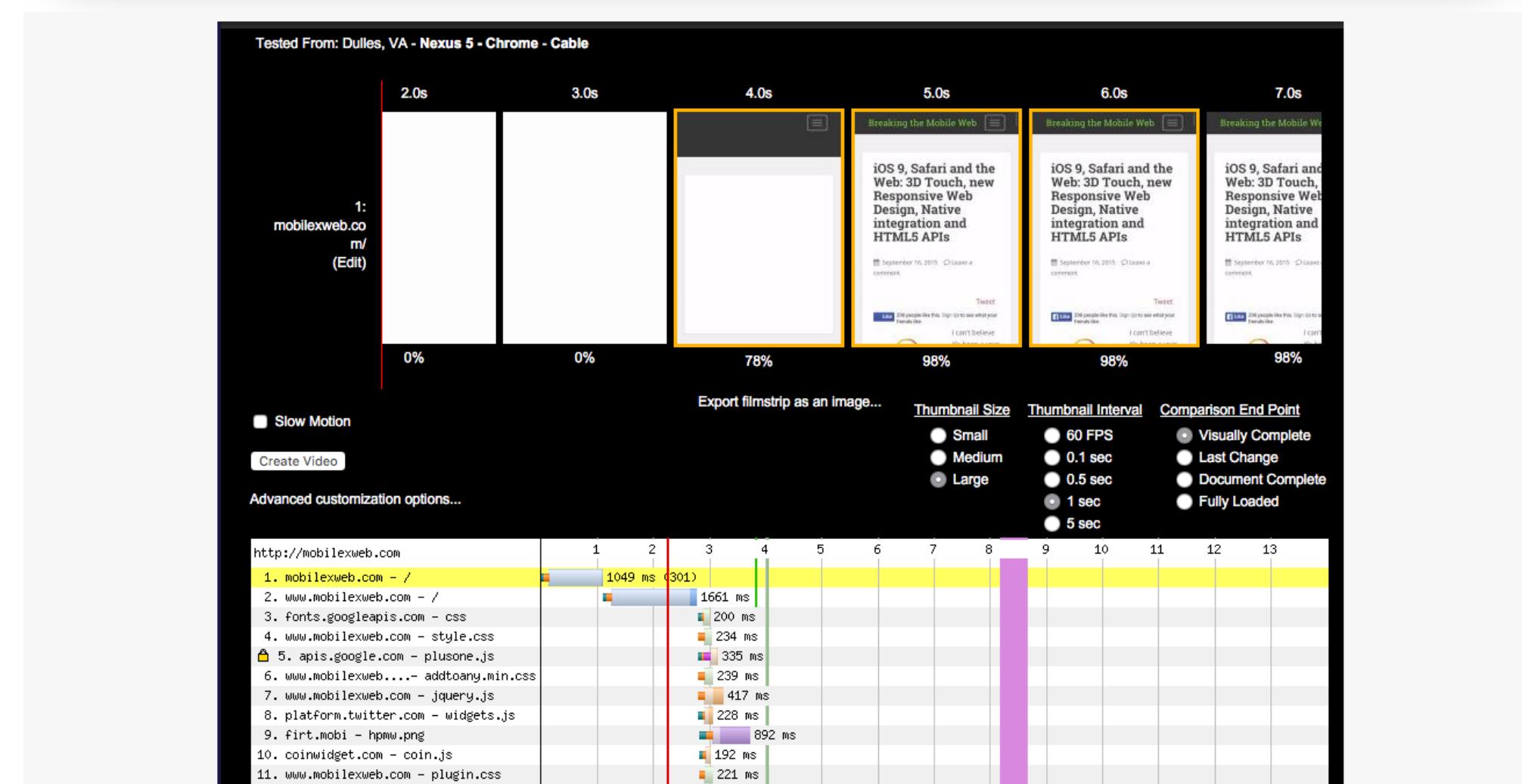
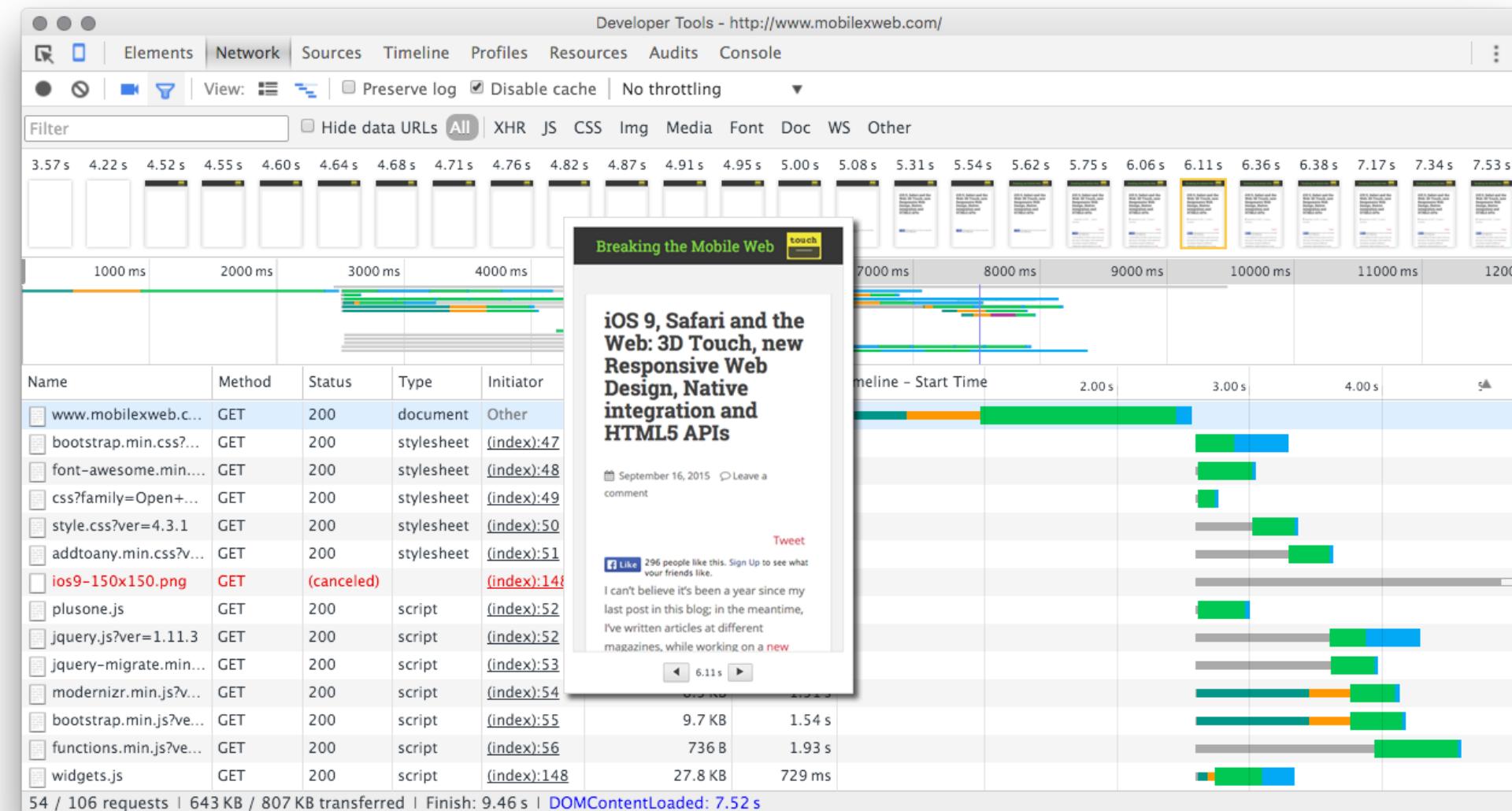
Repeat View  
(2.187s)



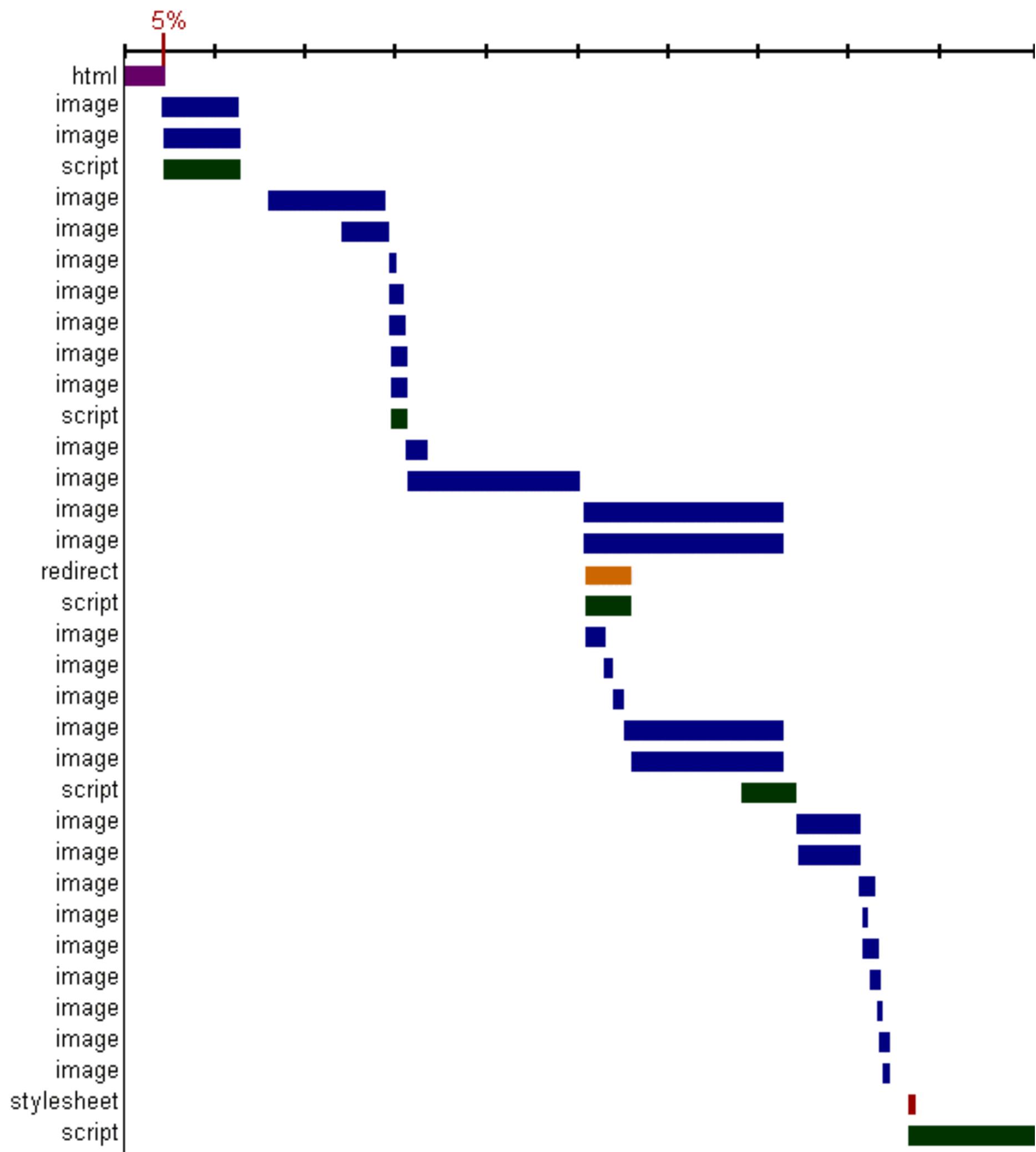
# Waterfall Chart Vertical Bars



# Filmstrip view

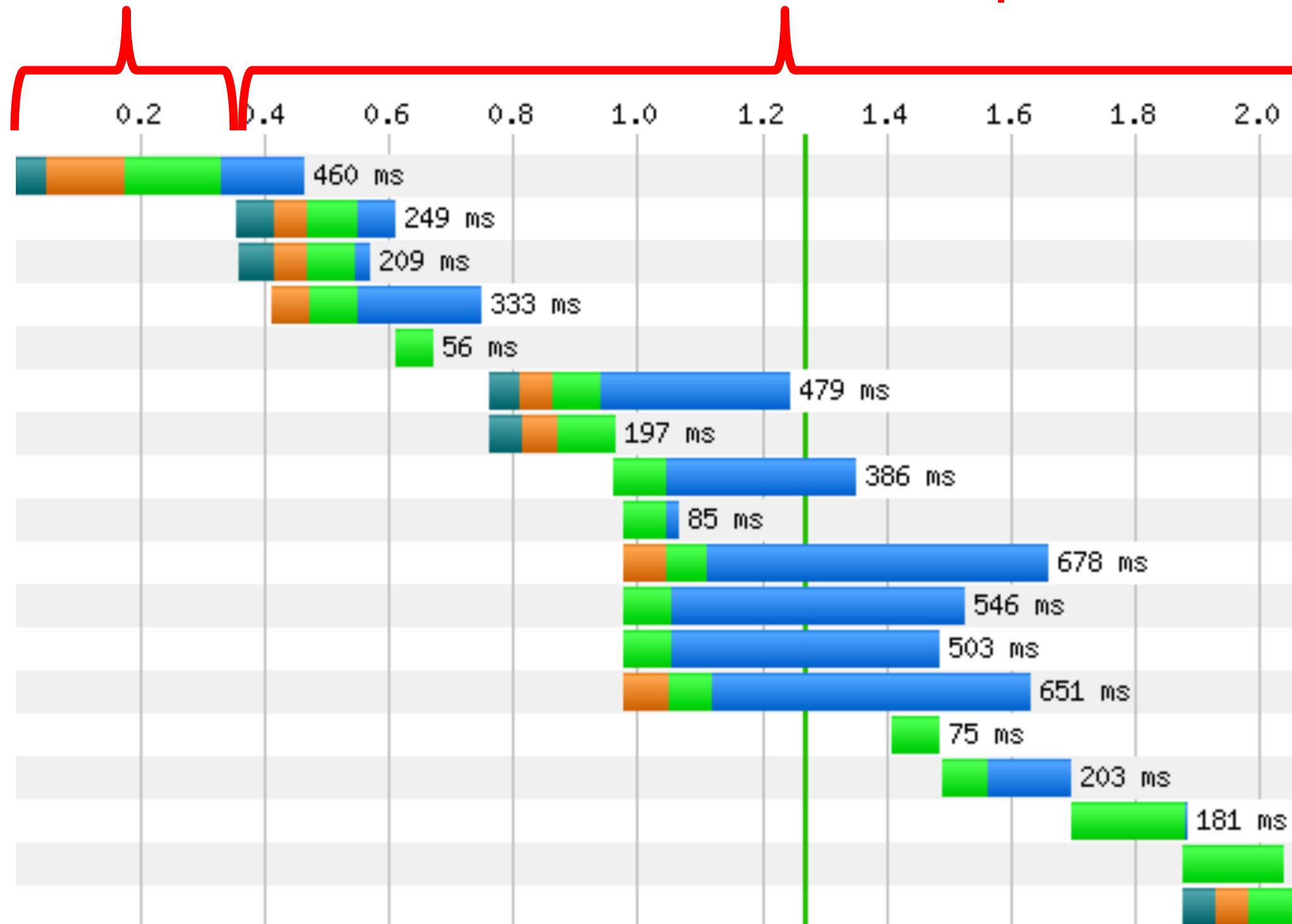


where to  
optimize?

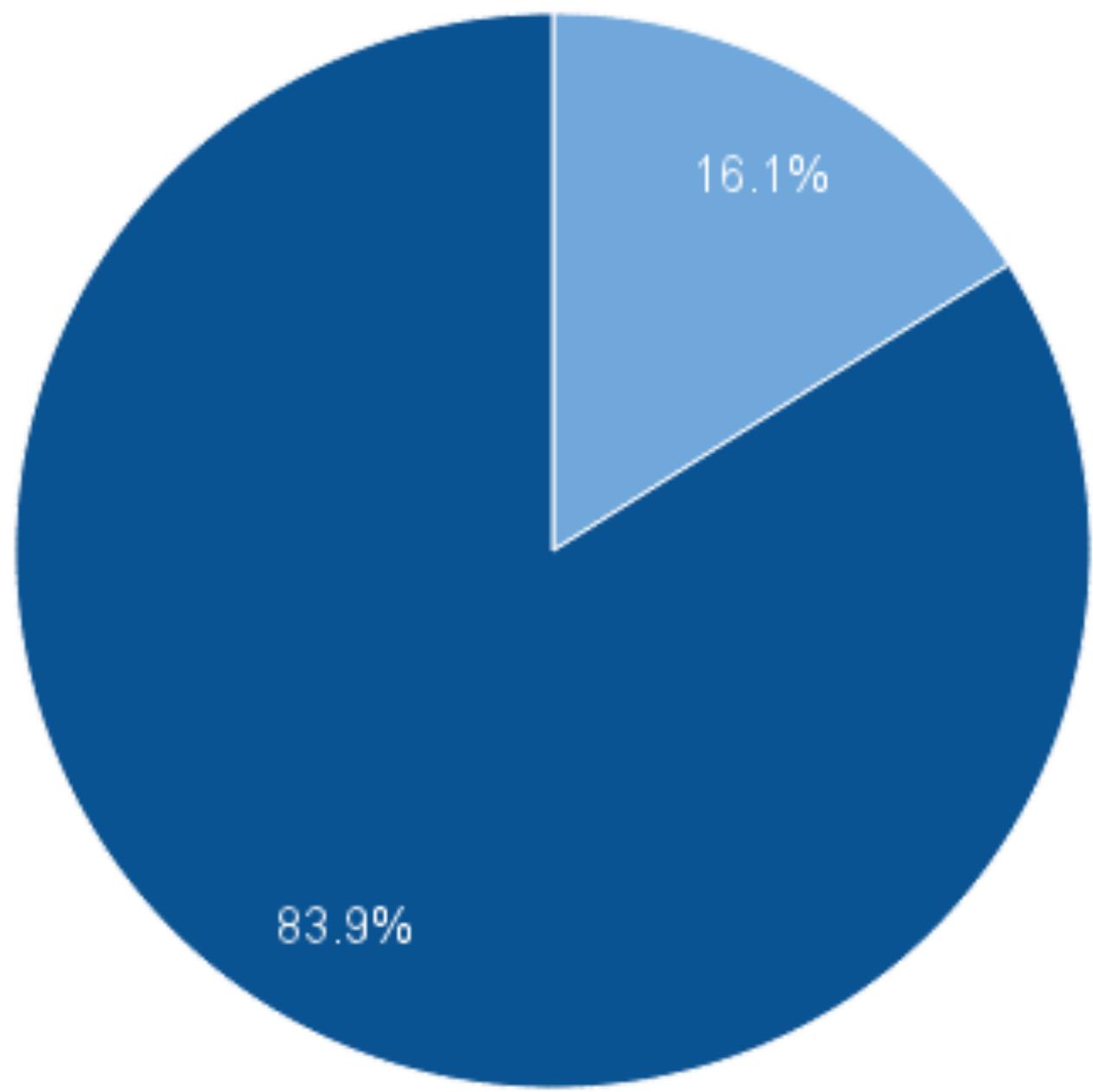


server

frontend development



# Top 100,000 sites



■ backend ■ frontend



## IMPORTANT

Optimizing the backend  
has less impact and it's not  
a simple or cheap task.



## IMPORTANT

Optimizing the frontend  
has high impact, it's  
relatively cheap to optimize  
and results are immediate.



# Understanding the Browser



**What happens when we  
browse the web?**

# Browsing the Web

- Network layer
- Parsing
- Resource Discovery Phase
- Resource Prioritization
- Layout
- Paint





# HTTP

# Basic Performance

## HTTP/1.1

- One request per TCP connection
- Limited number of parallel requests to the server
- GZIP Encoding accepted

# Basic Performance

## HTTP/2

- Performance from scratch
- Header Compression
- TCP connection reuse
- ~~Push to Cache~~

# Basic Performance

## HTTP/2

- TLS-based only
- Good compatibility
- Upgrade your servers or use a CDN
- Upgrade connections

# Basic Performance

## HTTP/3

- Transport protocol over UDP
- Reduces latency and connection messages
- HTTP/2 Interface with TLS



# The Browser's Cache

# The Cache



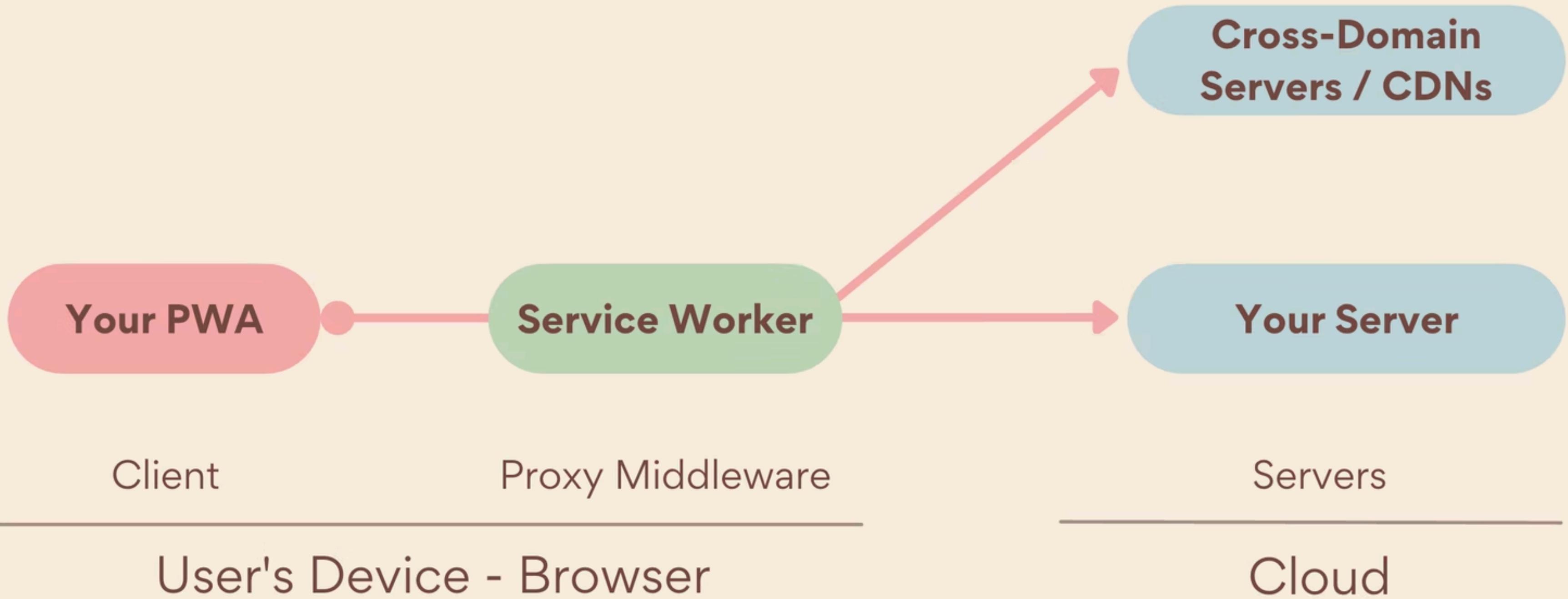
## Back/Forward Cache (bfcache)

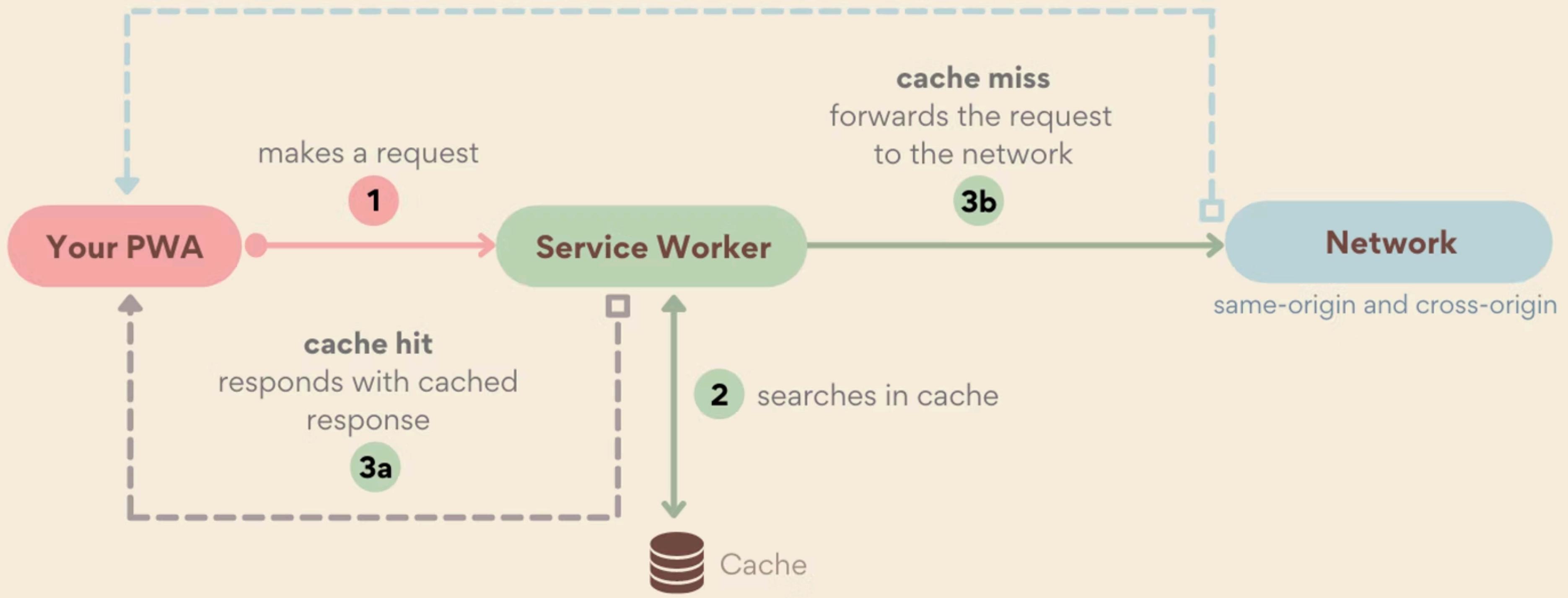
- It keeps your page navigation in memory if the user navigates away
- It's automatic
- You shouldn't use unload events,  
Cache-Control: no-store

## Back/Forward Cache (bfcache)

- Use Page Navigation API to
- Open/Restore connections
- Abort pending transactions

# Service Workers and Cache Storage



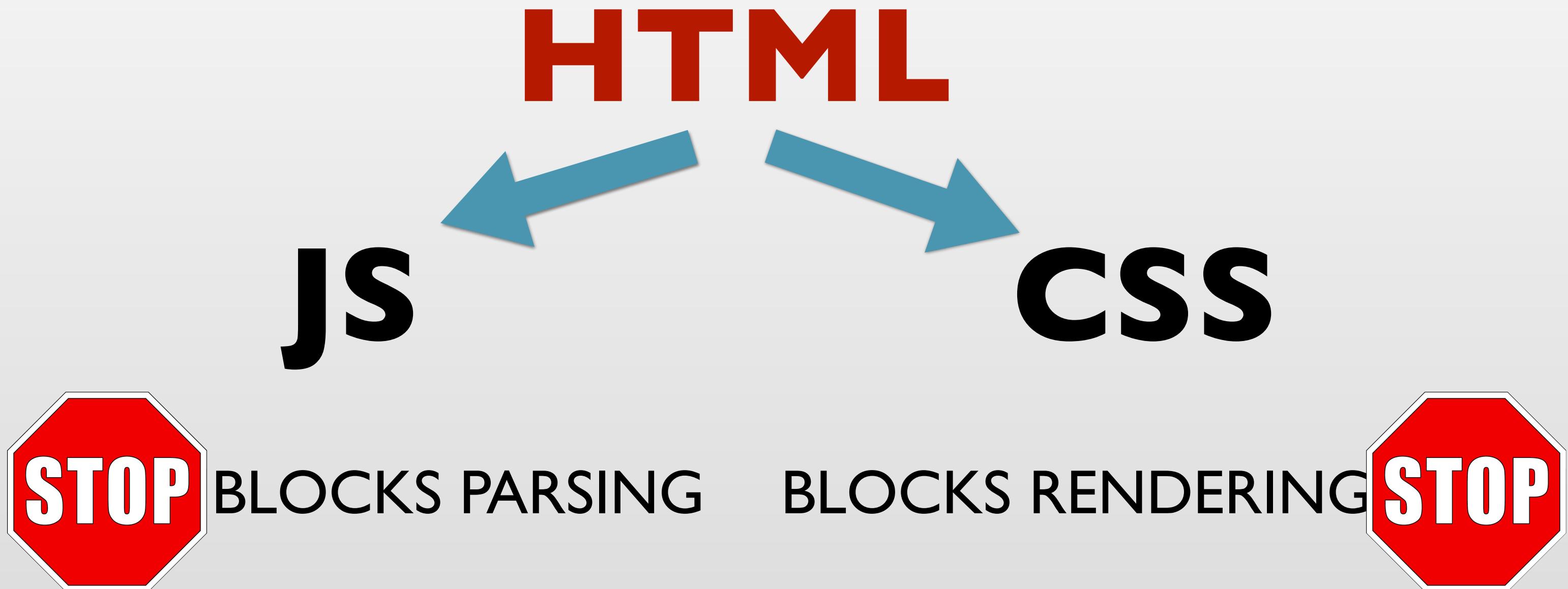


User's Device - Browser



# Resource Loading Impact

# Basic Performance





# Frames and Interactivity

**Frame**

**Input Events**

**Parse HTML,  
Styles, Layout**

**Paint and  
Composite**

**Idle**

**Timers**

**Frame**



# Basic Performance Optimizations

## Basic Performance

Enable GZIP on text-based files

# Enable GZIP on text-based files

- HTML (static & dynamic)
- JavaScript, CSS, JSON, SVG

## Basic Performance

Make static content  
expire  
late in future

## Basic Performance

Use a CDN for static  
content

## Basic Performance

Consider implementing

HTTP/2

(and HTTPS)

## Basic Performance

Use cookie-less domain

# Reduce Cookie Size



## Reduce Cookie Size

Cookie Size	Delta
0 bytes	0 ms
500 bytes	+1 ms
1000 bytes	+16 ms
1500 bytes	+31 ms
2000 bytes	+47 ms
2500 bytes	+63 ms
3000 bytes	+78 ms

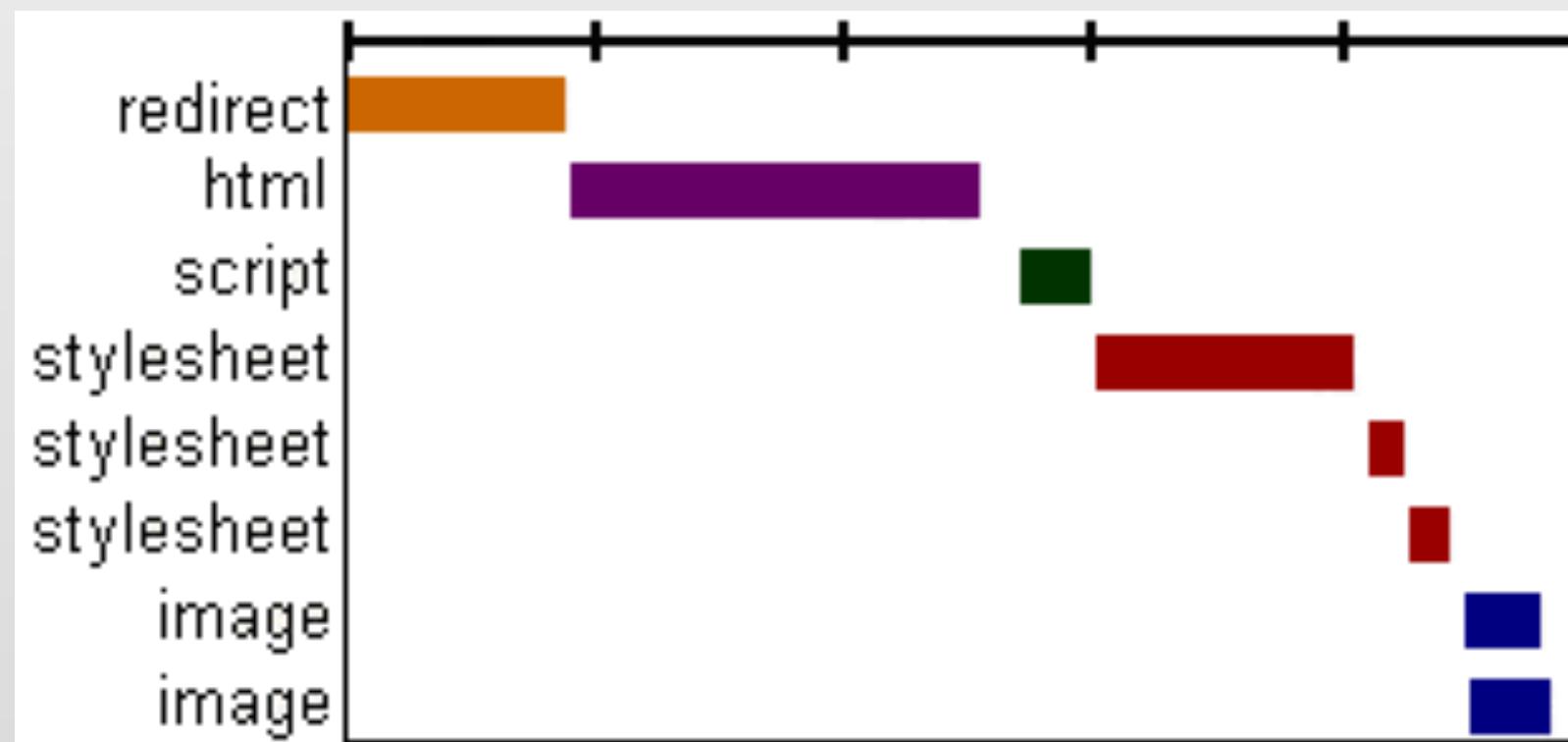
# Basic Performance

Reduce redirects



# Basic Performance

## Reduce redirects



# Reduce redirects

- Consume 100ms to 1s each
- Social networks will use one

# Basic Performance

CSS as appetizer

# CSS as appetizer

- Add a <link> as top as possible
- Don't use @import
- Remove or defer unused styles

# JavaScript as dessert

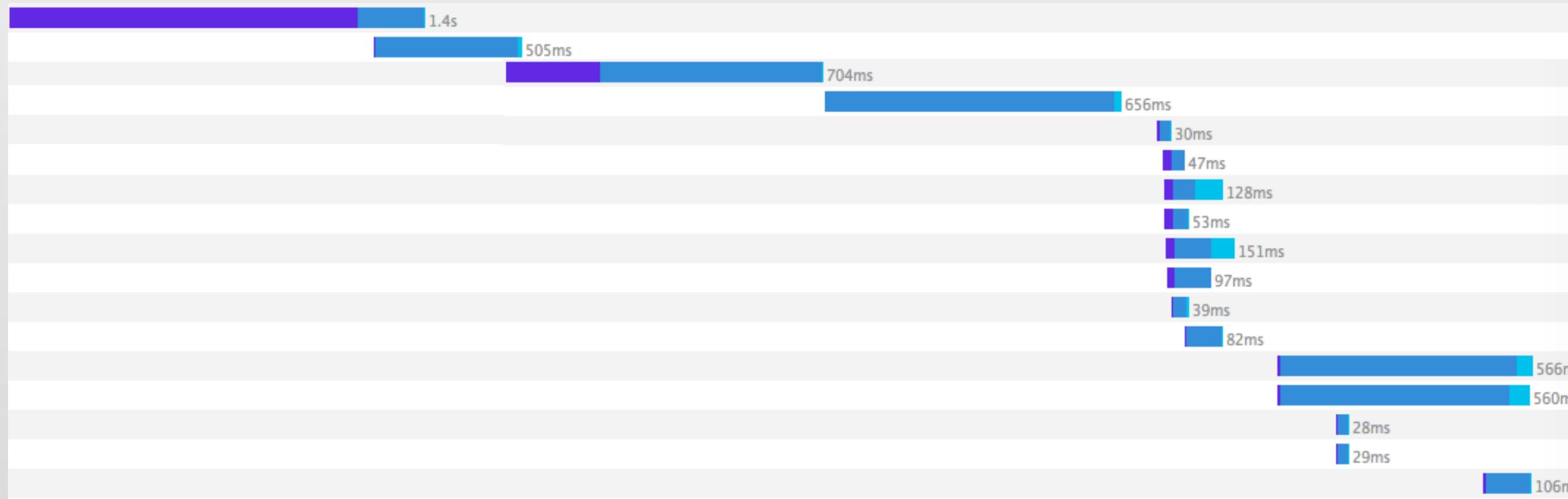
# JavaScript as dessert

- Defer it as much as possible
- It uses the main thread
- Remove or defer unused code

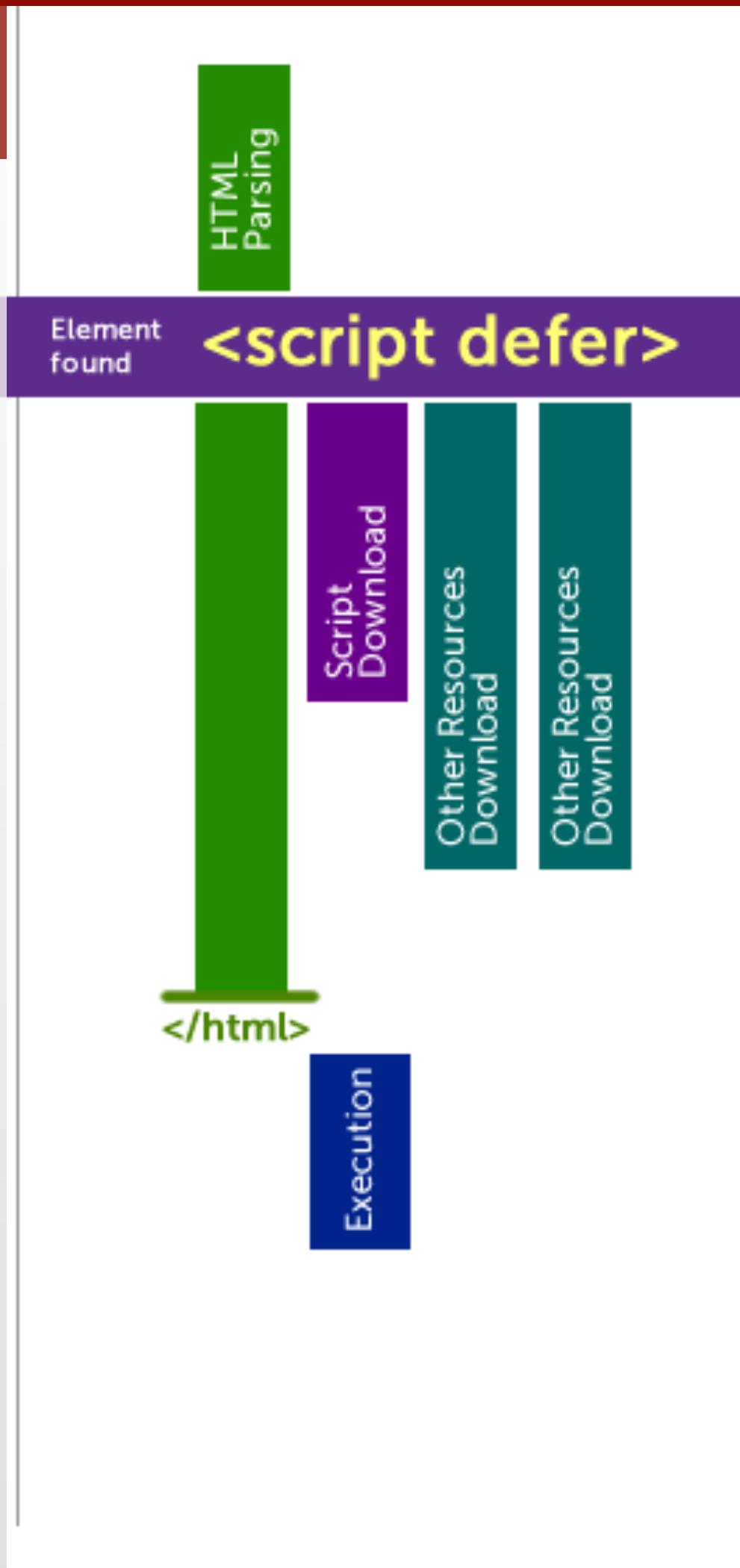
# Non-blocking scripts

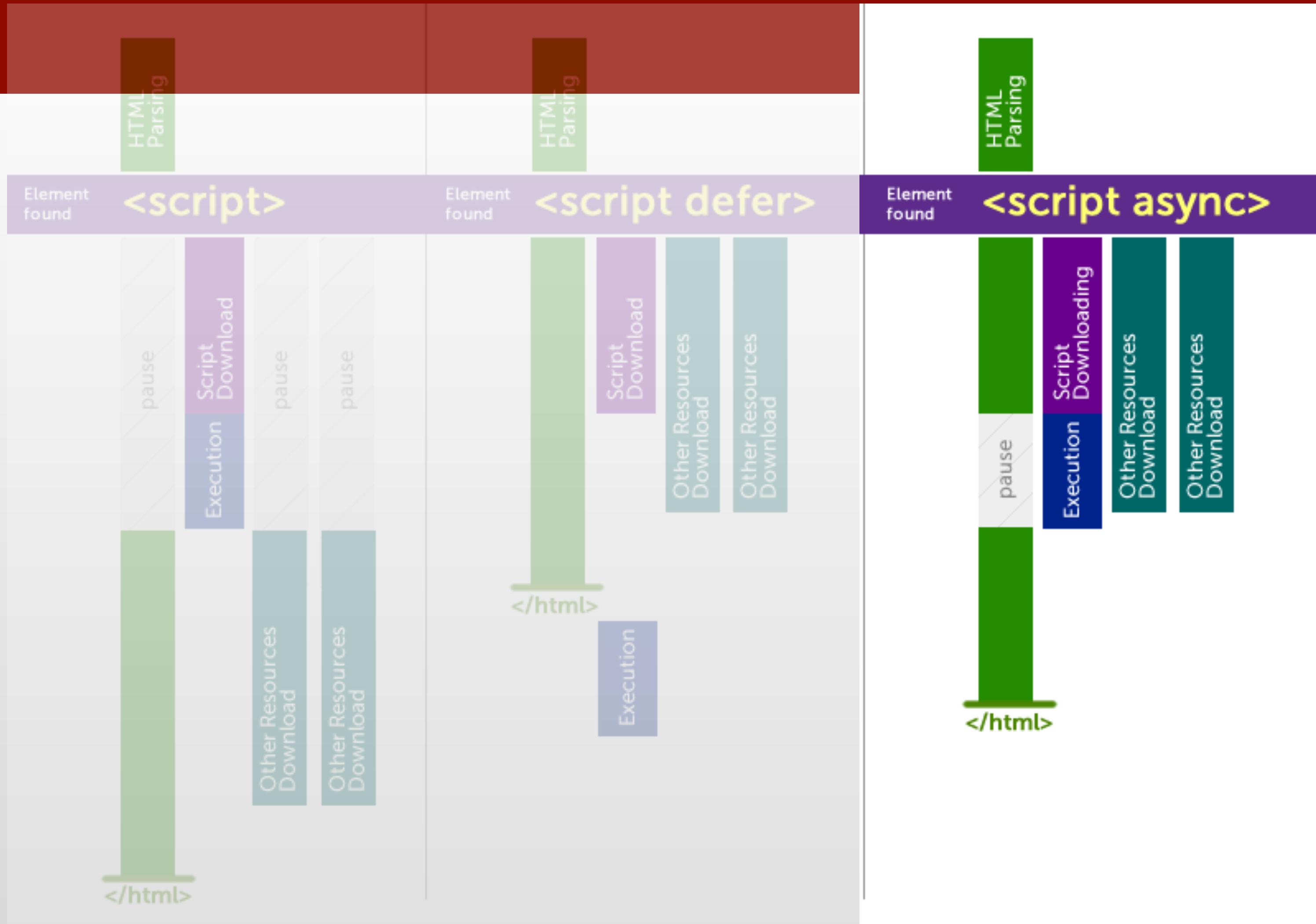
# Basic Performance

# Non-blocking scripts









Basic Performance

Compress/Obfuscate  
JavaScript and CSS

Basic Performance

Bundle Scripts and CSS  
files

## Basic Performance

Release the main thread

ASAP

Basic Performance

Embrace Responsive  
Images

Basic Performance

Embrace SVG

## Basic Performance

**Compress Images**

**Define placeholder size**

Careful with Web Fonts



# Hacking Performance



# Hacking First Load

## Avoid more than one roundtrip

- TCP Slow Start
- Initial congestion window (`initcwnd`)
- Linux: 14.6 KiB

KiB?



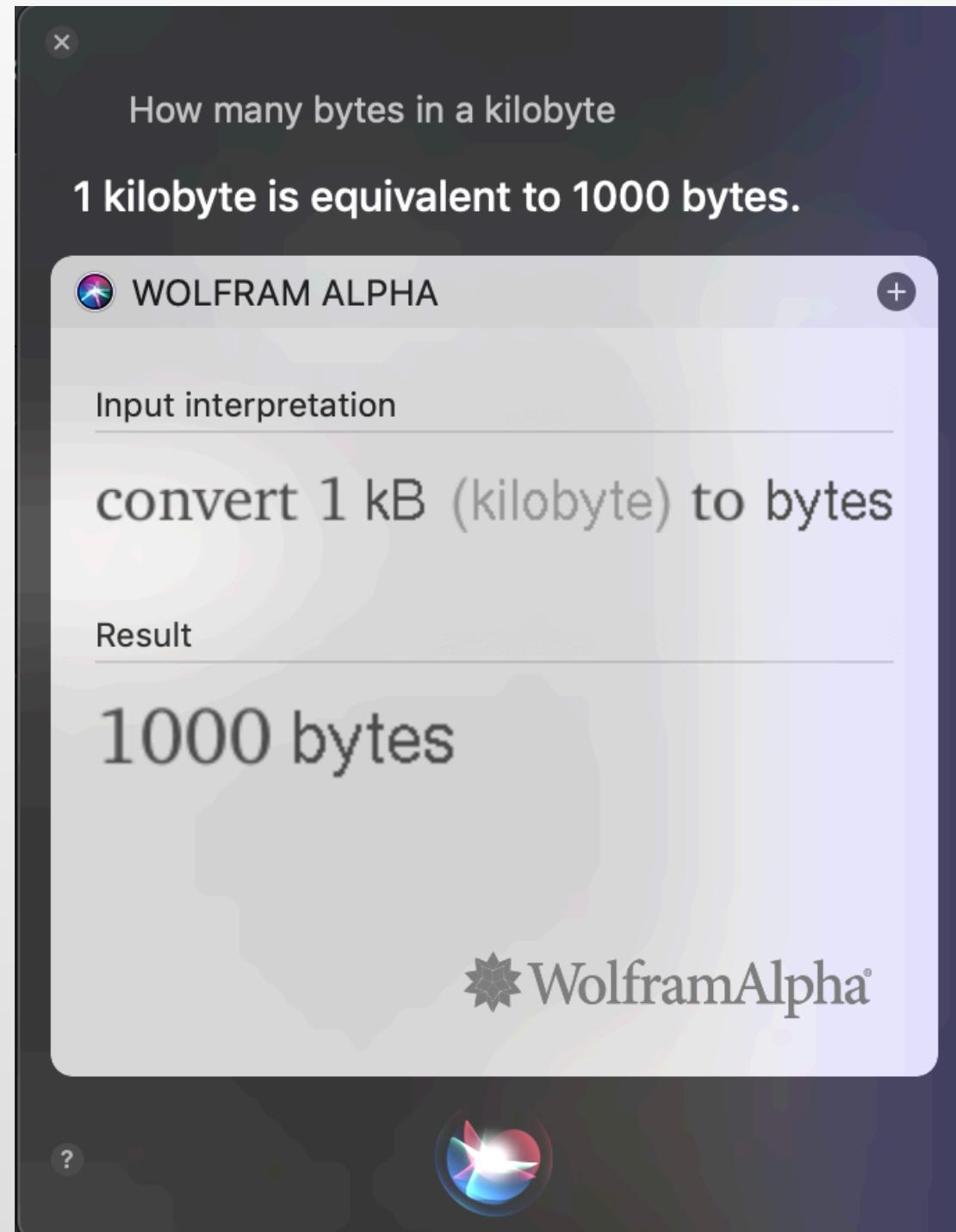
How many bytes in a  
kilobyte?

1 KB = 1024 bytes

~~1 KB = 10<sup>24</sup> bytes~~

1 KB = 1000 bytes

# You are already...



# You are already...

How many bytes in a kilobyte

1 kilobyte is equivalent to 1000 bytes.

 WOLFRAM ALPHA 

Input interpretation

convert 1 kB (kilobyte) to bytes

Result

1000 bytes





Google how many bytes in a kilobyte

All Images Maps Videos News More

About 5,960,000 results (0.42 seconds)

Digital Storage

1 = 1000

Kilobyte Byte

Formula multiply the digital storage value by 1000

# You are already...

The image shows two side-by-side search results for the query "how many bytes in a kilobyte".

**Wolfram Alpha Result:**

- Input: How many bytes in a kilobyte
- Result: 1 kilobyte is equivalent to 1000 bytes.
- Source: WOLFRAM ALPHA
- Input interpretation: convert 1 kB (kilobyte) to
- Result: 1000 bytes

**Google Search Result:**

- Search bar: how many bytes in a kilobyte
- Filter: All
- Results: About 5,960,000 results (0.42 seconds)
- Section: Kilobyte
- Text: From Wikipedia, the free encyclopedia
- Text: The **kilobyte** is a multiple of the unit **byte** for digital information.
- Text: The International System of Units (SI) defines the prefix **kilo** as  $1000 (10^3)$ ; per this definition, one kilobyte is 1000 bytes.<sup>[1]</sup> The internationally recommended unit symbol for the kilobyte is **kB**.<sup>[1]</sup>

You are already...

# Kibibyte

---

From Wikipedia, the free encyclopedia

The **kibibyte** is a multiple of the unit **byte** for quantities of digital **information**. The **binary prefix *kibi*** means  $2^{10}$ , or 1024; therefore, 1 kibibyte is 1024 bytes. The unit symbol for the kibibyte is **KiB**.<sup>[1]</sup>

1 KiB = 1024 bytes

## Avoid more than one roundtrip

- TCP Slow Start
- Initial congestion window (initcwnd)
- Linux: 14.6 KiB



# Above the Fold (ATF)

# Below the Fold (BTF)

# Deliver ATF in 14.6 KiB

- Embed all CSS and JavaScript needed
- If space, embed logo and/or low-res images

# Avoid http to https redirect

- Use HSTS (HTTP Strict Transport Security)
- Header
- Opt-in at [hstspreload.org](https://hstspreload.org)



# Hacking LCP

# Preloading

Help the browser discover resources  
that are obscure in the document

# Preloading

LCP

HTML

```
<link rel="preload" href="styles.css" as="style">
```

# Preloading

HTTP

```
Link: <https://otherhost.com/font.woff2>;  
rel=preload; as=font; crossorigin
```

# Fetch Priority

HTML

```
<link rel="preload" as="image"  
      href="hero.jpg" fetchpriority="high">  
  

```



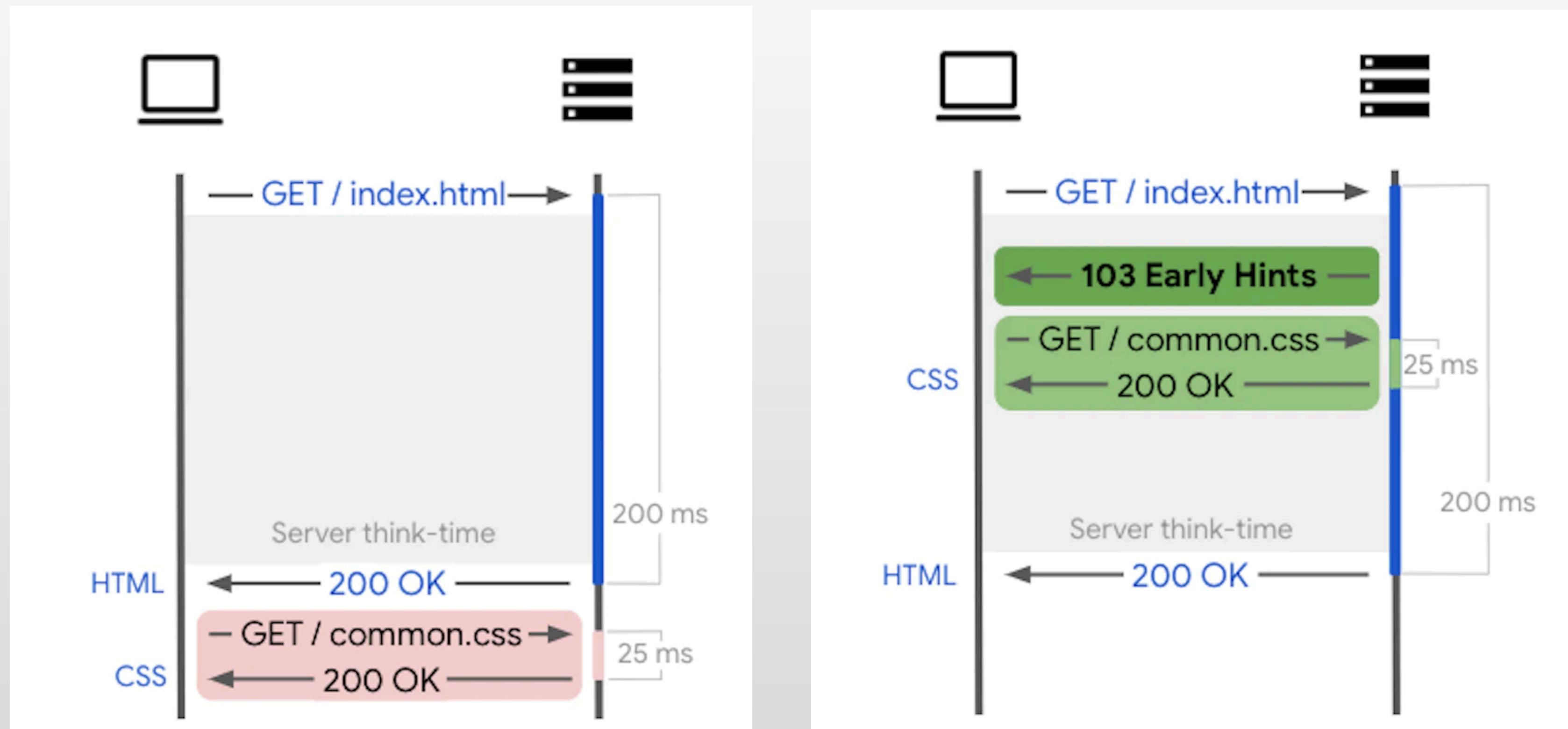
## WARNING

Fetch priority was called Priority hints before standardization, and the **fetchpriority** attribute was called **importance**.

# HTTP Early Hints

- HTTP Code 103
- We sent it before processing server-side
- Then we send HTTP Code 200

# LCP





## IMPORTANT

If a browser doesn't yet support Early Hints, nothing will happen; the browser will simply ignore it the 103 HTTP code and wait for 200



# Hacking Data Transfer

## HTTP/3

- Transport protocol over UDP
- Reduces latency and connection messages
- HTTP/2 Interface with TLS

# Data Transfer

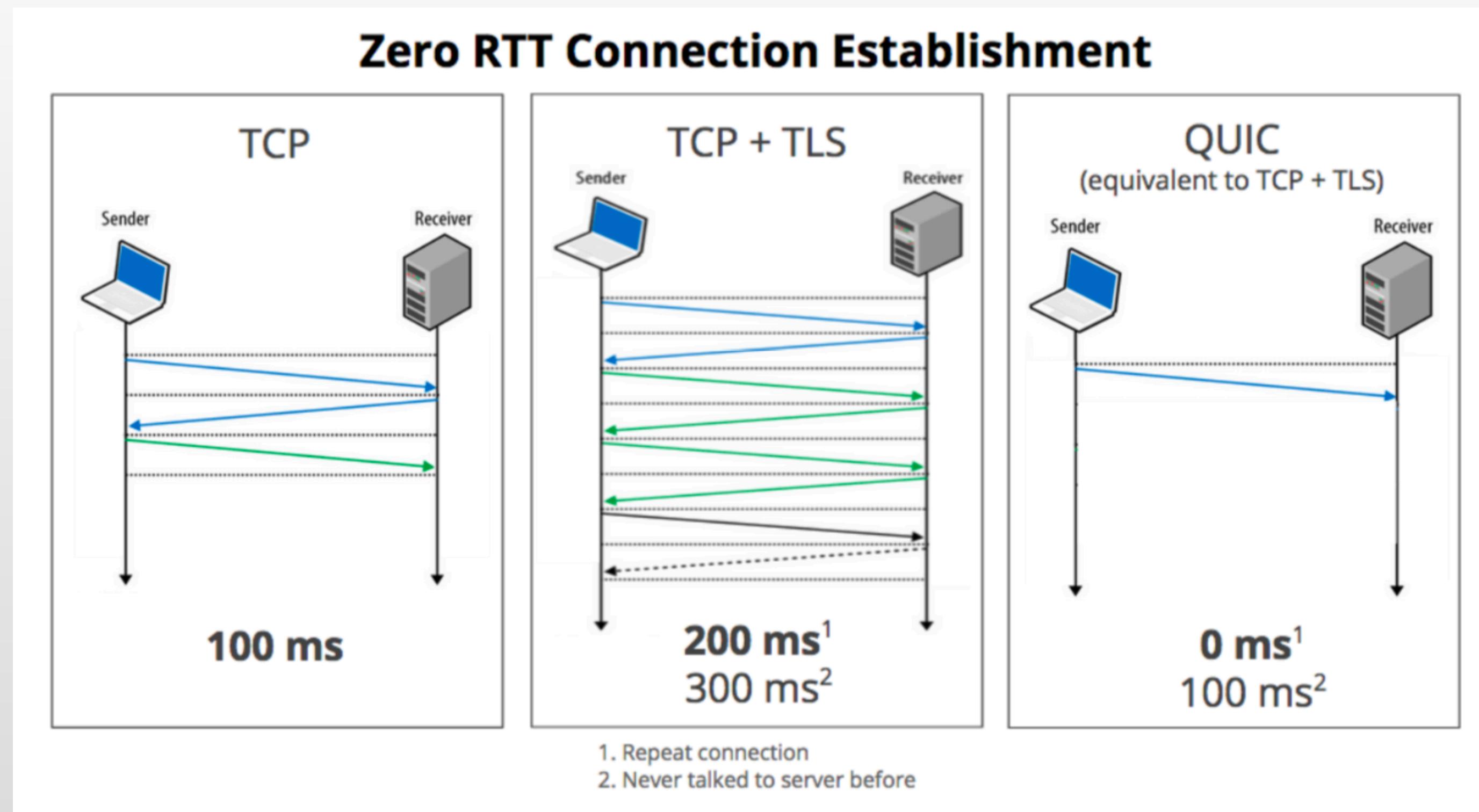


Image from Chromium Blog

## QUIC over HTTP (now HTTP/3)

- Google: +3%
- YouTube: -30% buffering
- Facebook similar protocol: +2%
- 75% of requests can be optimized

You are already...

# Use Zopfli

- Save 3-8% data transfer with GZIP
- It's ~80x slower

You are already...

## Use Brotli

- Save ~25% data transfer compared with GZIP
- Check Encoding Header

You are already...

# Use Brotli

- LinkedIn: 4% savings in load time
- Facebook: 17% savings on CSS, 20% on JavaScript



# Hacking Resource Loading

## Modern Image Formats

- SVG
- WebP, AVIF, Guetzli JPEG, Zopfli PNG
- Muted videos instead of GIFs

## Modern Cache Control

- Hash in filenames is common for versioning
- Browsers make conditional requests

Resource Loading

# Modern Cache Control

Cache-Control: immutable

## Modern Cache Control

It's a common new pattern to

- 1) serve from the cache
- 2) update it in the background for later

## Modern Cache Control

Cache-Control: stale-while-revalidate=99

## Warming up engines

- Help the browser to start ASAP
- DNS Queries: ~100ms
- TCP and TLS connection: ~100ms

## Announce DNS queries

HTML

```
<link rel="dns-prefetch" href="https://newdomain.com">
```

## Announce TLS connections

HTML

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

## Announce on HTTP Response

HTTP

Link: <<https://my-analytics.com>>; **rel=preconnect**; crossorigin

You are already...

# Lazy Load for Images

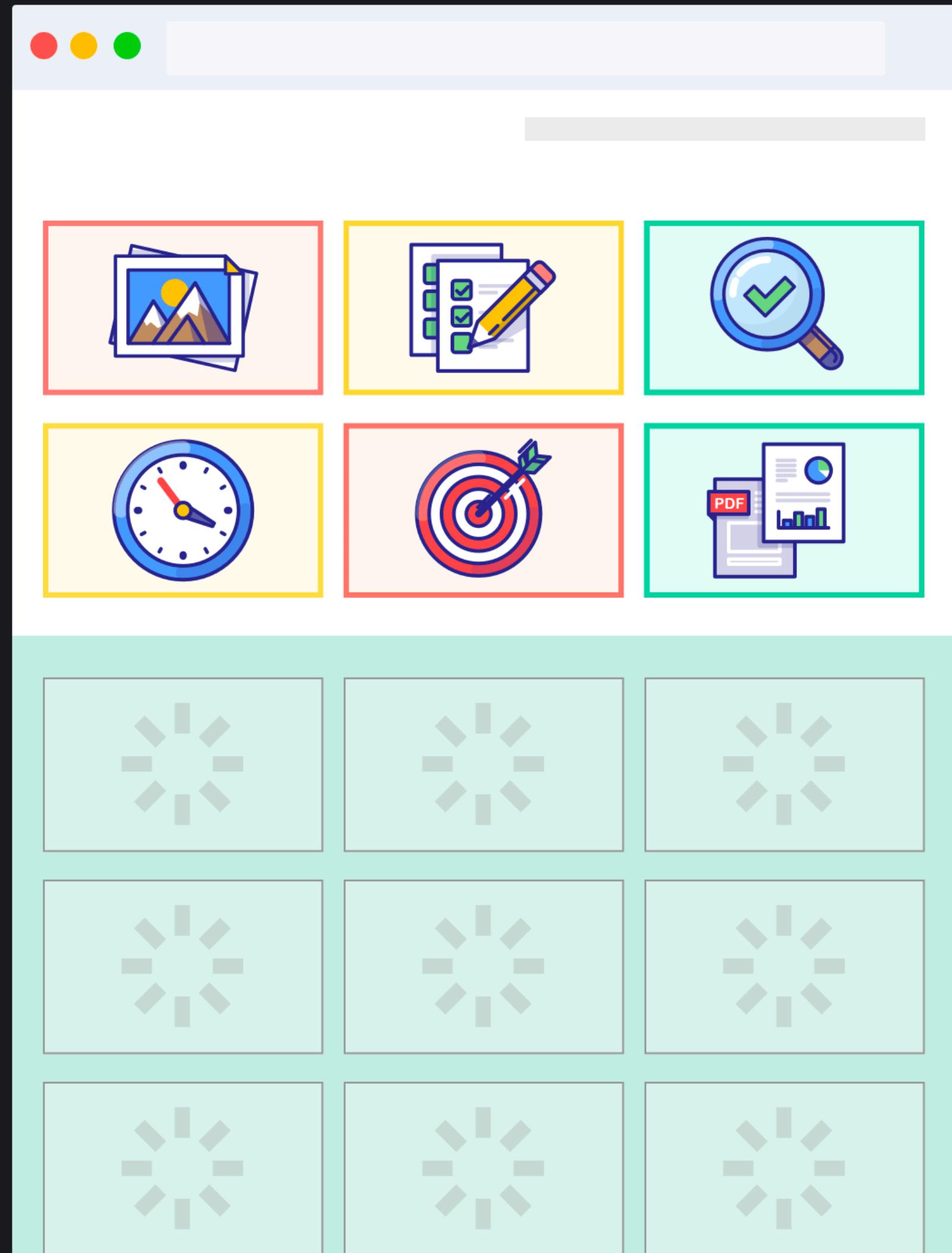
HTML

```

```

# THE `LOADING` ATTRIBUTE

```
<img loading=lazy>  
<iframe loading=lazy>
```



You are already...

# Async image decoding

HTML

```

```

## Web Fonts

- Avoid FOUT (Flash of Unstyled Text)
- **font-display: optional or swap**

## HTTP Client Hints

- Browser will expose data to the server

HTML

```
<meta http-equiv="Accept-CH"  
content="DPR, Viewport-Width, Device-Memory">
```

# Resource Loading

## HTTP Client Hints

- Today we can ask for
  - RTT
  - Downlink
  - ECT (2g, 3g, 4g, slow-2g)
  - Save-Data
  - DPR and Viewport-Width
  - Device-Memory

# Resource Loading

## 2.1. Computing Device Memory Value

### actual device memory in MiB

The value is calculated by using the actual device memory in MiB then rounding it to the nearest number where only the most significant bit can be set and the rest are zeros (nearest power of two). Then dividng that number by 1024.0 to get the value in GiB.

An upper bound and a lower bound should be set on the list of values.

**NOTE:** While implementations may choose different values, the recommended upper bound is 8GiB and the recommended lower bound is 0.25GiB (or 256MiB).



# Hacking Interaction Experience

Interaction Experience

# Client-side rendering



# Interaction Experience

# Move Heavy tasks to WebAssembly



**Stop serving legacy  
code**

Interaction Experience

# **Reactive Web Performance**

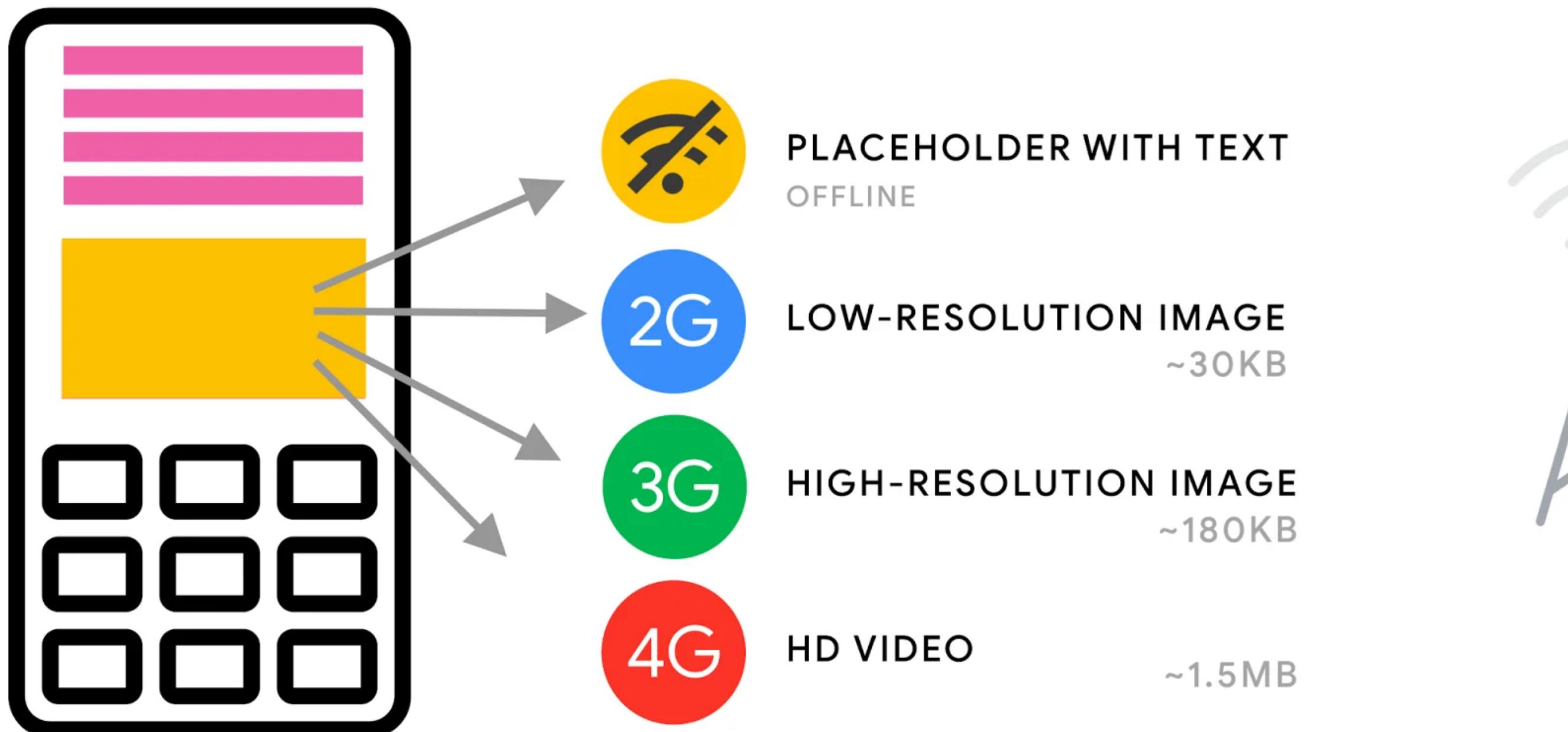
## Know about context with APIs

- Network Information
- Performance Observers
- Save-Data Client Hint
- Device Memory Client Hint

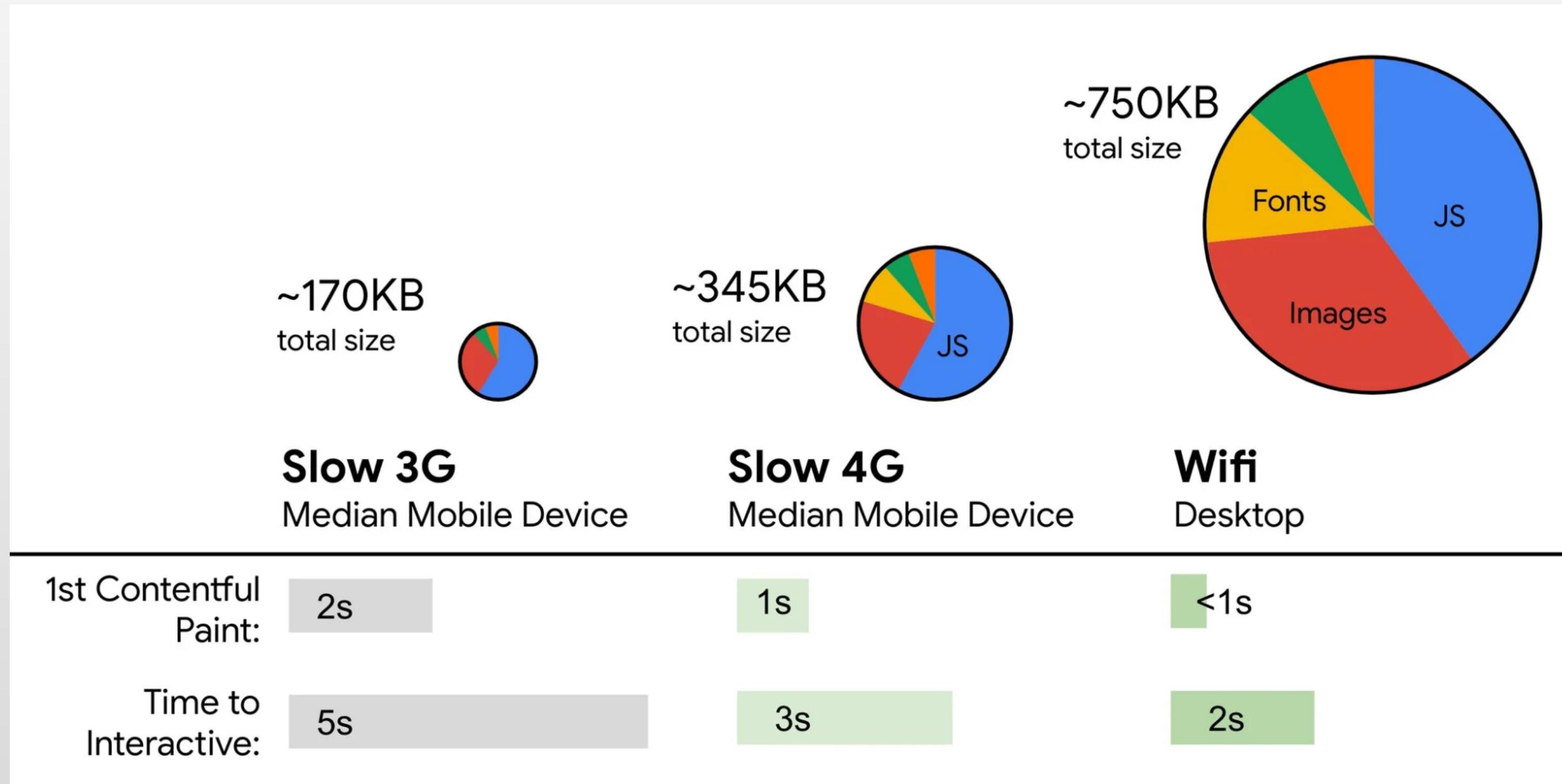
## Provide a Consistent Experience

- Web Fonts
- Change SW's cache policy
- SSR vs CSR
- Reduce amount of loaded data
- 1x image not matter DPR

# Interaction Experience



# Interaction Experience

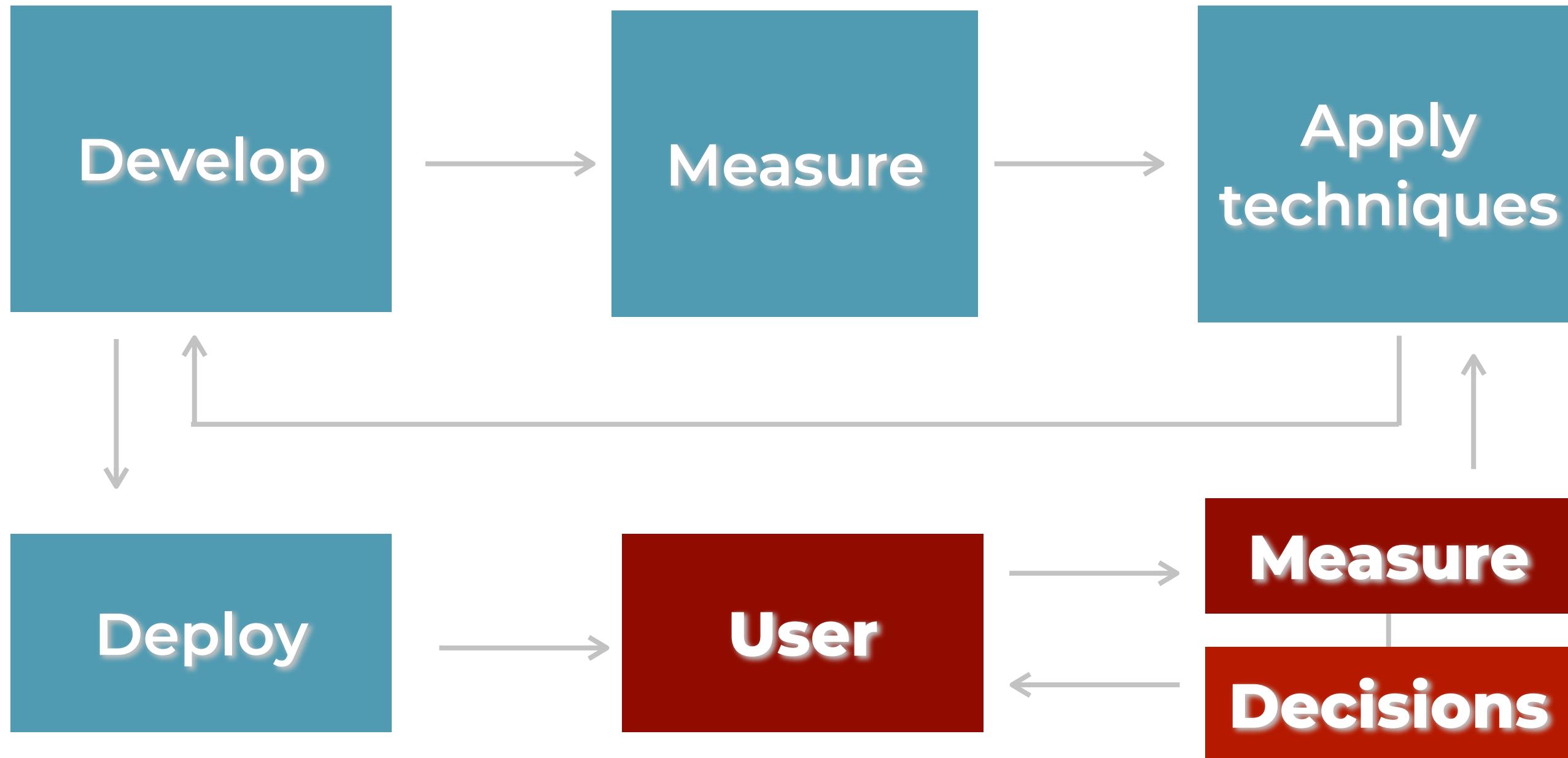


## Get Reports from the Browser

- Reporting API
- Report-To header with a server endpoint
- Browser will send you reports about problems



# Using Performance APIs



**We need to measure and change**  
Keep a consistent experience making on-the-fly  
changes

# APIs for Web Performance

- They work client-side
- Can be shared to the server, automatically or manually
- Not every API is available on every browser
- Fallback to poly fills, other solutions or previous stats

# We can measure

- Timings for DNS, TCP, HTTP Request and Response
- Type of Connection (2G, 3G, 4G, other)
- Bandwidth and Latency (RTT)
- Device's Memory
- Custom metrics for own goals

# We can measure

- Paint Timings, such as FCP or CLS
- CPU-intense operations

# Real User Monitoring (RUM)

- When working statically, we don't know how our users are accessing the website, so our measurements are not real
- We can make analytics on every user to improve their experience and also create stats to improve our local testings

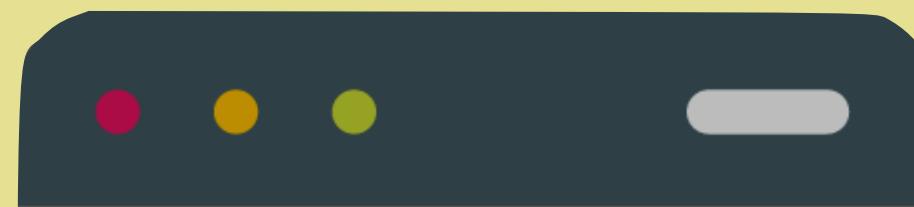
# Navigation Timing API

- It's available on all browsers;
- It's on the window global object
- Also on Web Workers
- It's a global **performance** object

# Navigation Timing API

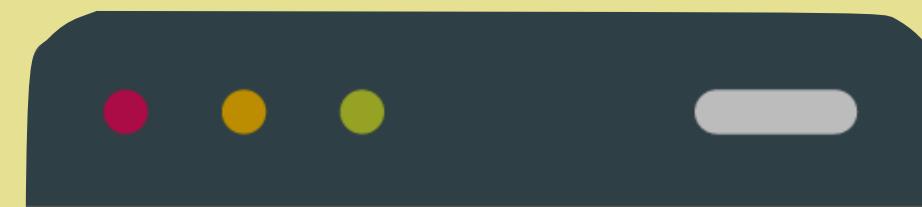
- It's the base API for all the rest of measurement specs available
- Three functionalities:
  - 1) `performance.now()`
  - 2) navigation type
  - 3) get timings for current navigation

# Get Current Timestamp



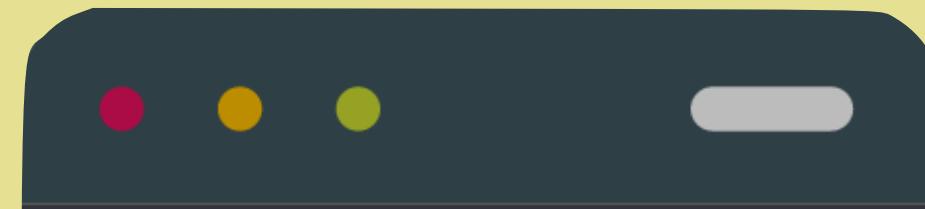
```
const now = performance.now();
const since = performance.timeOrigin;
```

# Get Navigation Data



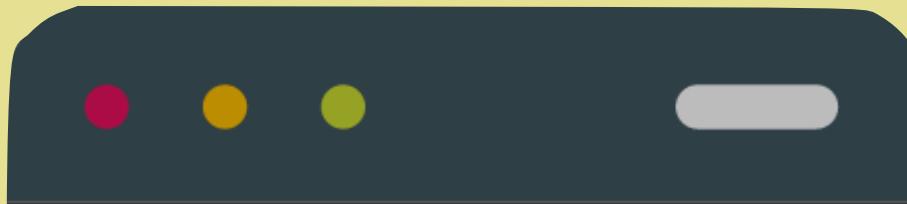
```
// deprecated on Level 2  
const redirects =  
  performance.navigation.redirectCount
```

# Get Navigation Data



```
switch (performance.navigation.type) {  
  case: performance.navigation  
    .TYPE_NAVIGATE:  
  case: performance.navigation  
    .TYPE_RELOAD:  
  case: performance.navigation  
    .TYPE_BACK_FORWARD:  
}
```

# Get Navigation Timings



```
// deprecated on Level 2  
const timings = performance.timing;
```

The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. At the top, there are icons for back, forward, and refresh, followed by tabs for 'Elements', 'Console' (which is highlighted in blue), 'Security', 'Application', 'Audits', and more. Below the tabs is a toolbar with a play button, a stop button, and a dropdown set to 'top'. To the right of the toolbar are 'Filter', 'Custom levels', and a checked checkbox for 'Group similar'. The main area displays the contents of the 'performance.timing' object. It starts with a greater than sign ('>') and 'performance.timing'. A less than sign ('<') and a downward arrow indicate that the following object is a child of 'performance.timing'. The object itself is labeled 'PerformanceTiming {navigationStart: 1536603272416, unloadEventStart: 0, unloadEventEnd: 0, redirectStart: 0, redirectEnd: 0, ...}' with an information icon ('i'). Below this, various properties of the PerformanceTiming object are listed, each preceded by a purple dot and followed by its value in blue. These properties include: connectEnd: 1536603272422, connectStart: 1536603272422, domComplete: 1536603277248, domContentLoadedEventEnd: 1536603273790, domContentLoadedEventStart: 1536603273665, domInteractive: 1536603273664, domLoading: 1536603272837, domainLookupEnd: 1536603272422, domainLookupStart: 1536603272422, fetchStart: 1536603272422, loadEventEnd: 1536603277292, loadEventStart: 1536603277248, navigationStart: 1536603272416, redirectEnd: 0, redirectStart: 0, requestStart: 1536603272428, responseEnd: 1536603272820, responseStart: 1536603272806, secureConnectionStart: 0, unloadEventEnd: 0, and unloadEventStart: 0.

```
> performance.timing
< PerformanceTiming {navigationStart: 1536603272416, unloadEventStart: 0, unloadEventEnd: 0, redirectStart: 0, redirectEnd: 0, ...} i
  connectEnd: 1536603272422
  connectStart: 1536603272422
  domComplete: 1536603277248
  domContentLoadedEventEnd: 1536603273790
  domContentLoadedEventStart: 1536603273665
  domInteractive: 1536603273664
  domLoading: 1536603272837
  domainLookupEnd: 1536603272422
  domainLookupStart: 1536603272422
  fetchStart: 1536603272422
  loadEventEnd: 1536603277292
  loadEventStart: 1536603277248
  navigationStart: 1536603272416
  redirectEnd: 0
  redirectStart: 0
  requestStart: 1536603272428
  responseEnd: 1536603272820
  responseStart: 1536603272806
  secureConnectionStart: 0
  unloadEventEnd: 0
  unloadEventStart: 0
```

**Prepare**



**navigationStart**

**Browsers starts a navigation  
Some optional preparation steps are done**

**Prepare**

↓  
**navigationStart**

**unloadEventStart**  
**unloadEventEnd**  
**redirectStart**  
**redirectEnd**

## **Browser's Preparation**

**Unloading previous page (only for same origin) or  
process a redirect. Deprecated AppCache process**

## Prepare

↓  
**navigationStart**

↓  
**fetchStart**

↓  
**unloadEventStart**

↓  
**unloadEventEnd**

↓  
**redirectStart**

↓  
**redirectEnd**

## Browser's Preparation

**Unloading previous page (only for same origin) or  
process a redirect. Deprecated AppCache process**

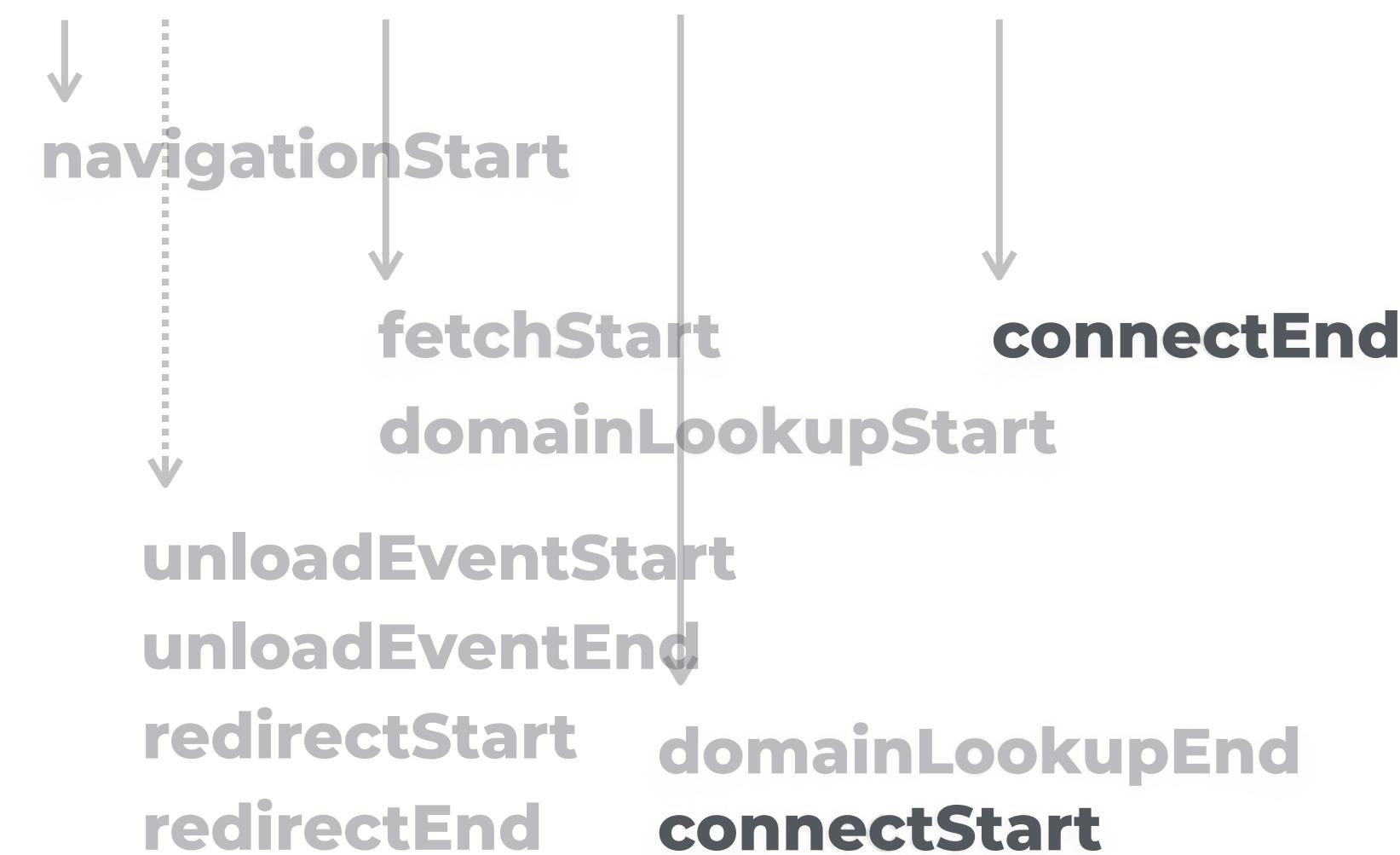


↓  
**navigationStart**  
↓  
**fetchStart**  
**domainLookupStart**  
↓  
**unloadEventStart**  
**unloadEventEnd**  
↓  
**redirectStart**    **domainLookupEnd**  
**redirectEnd**

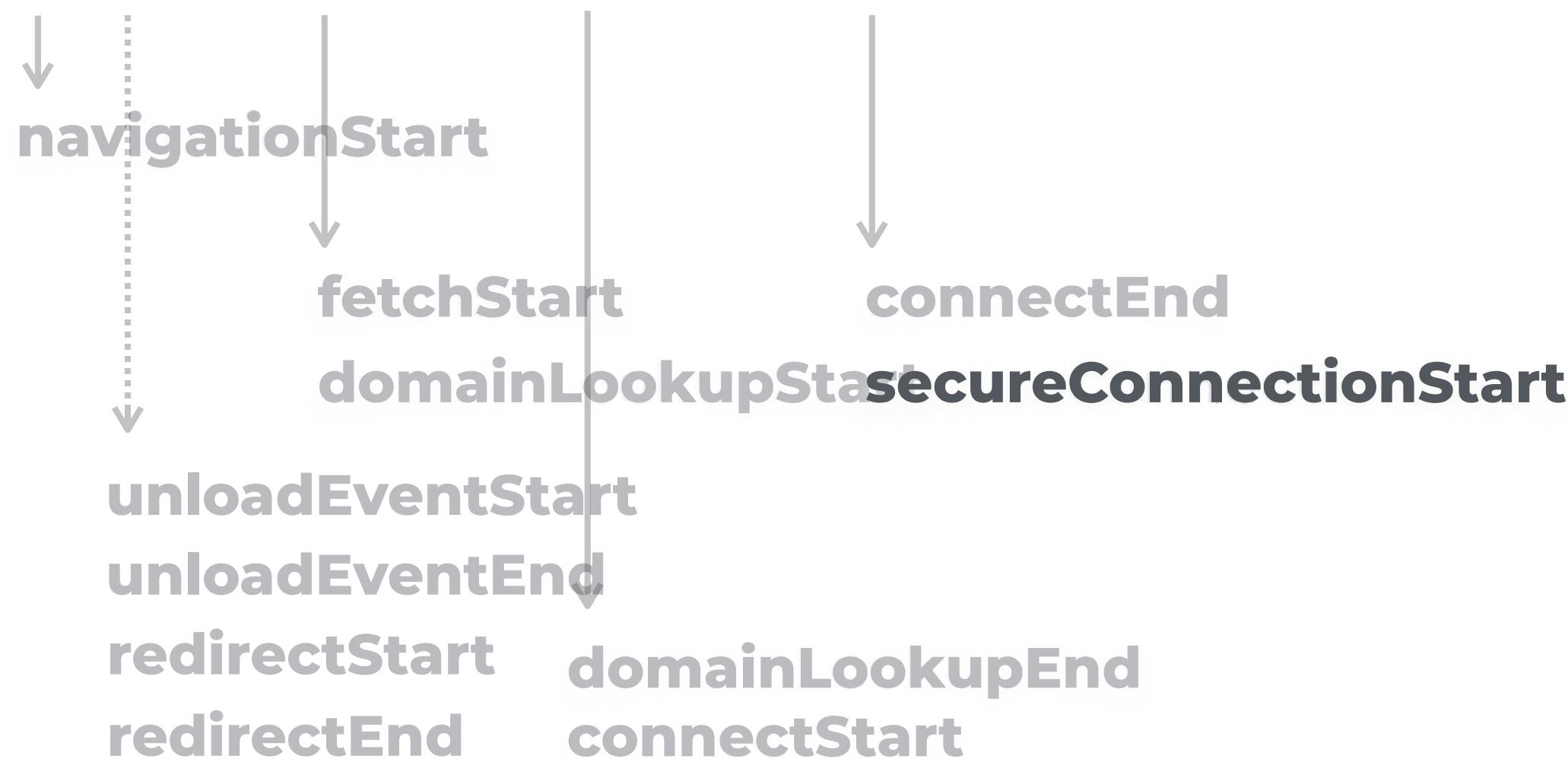
**DNS Query will start**  
**To convert a host into IP address**



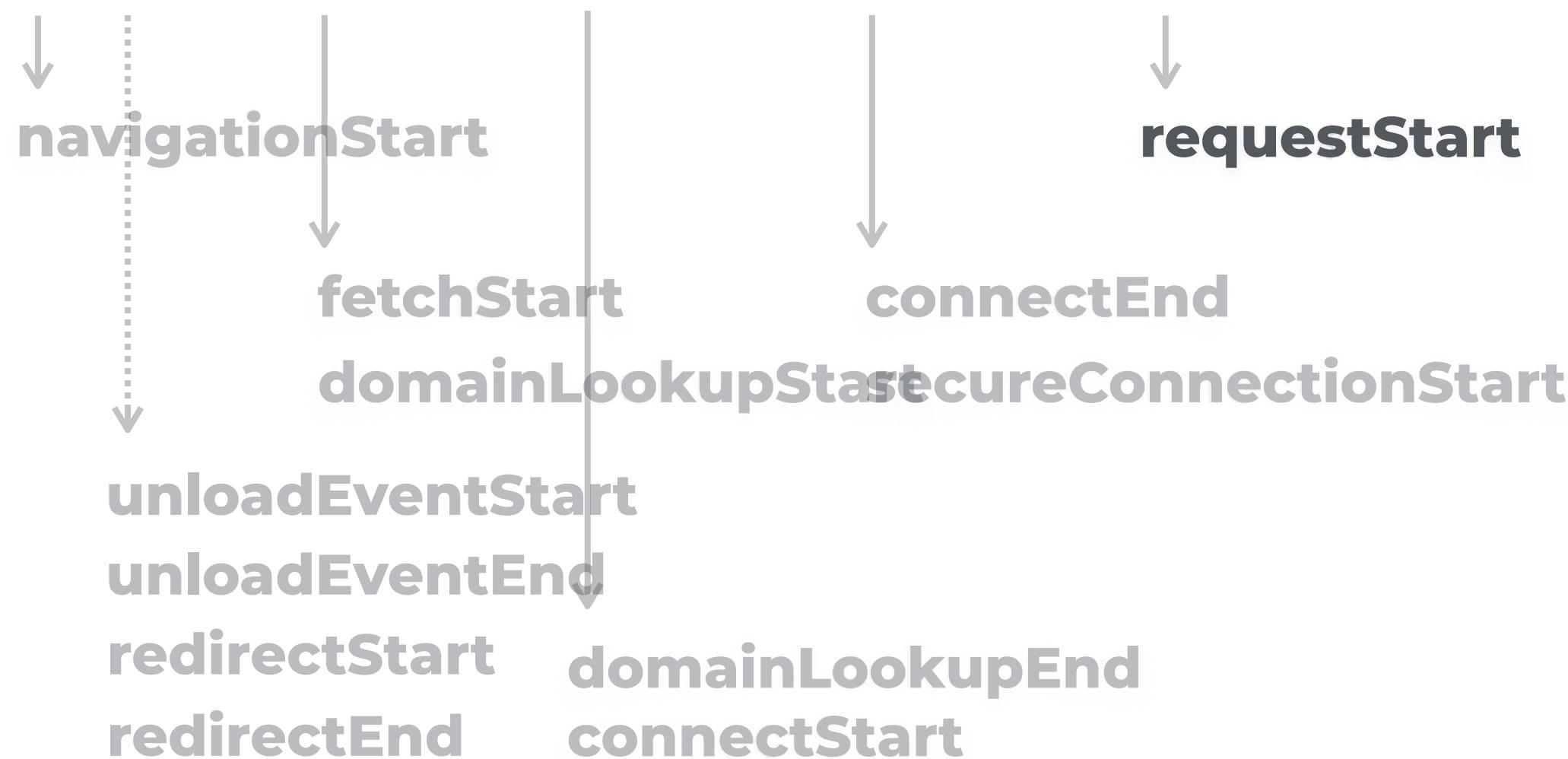
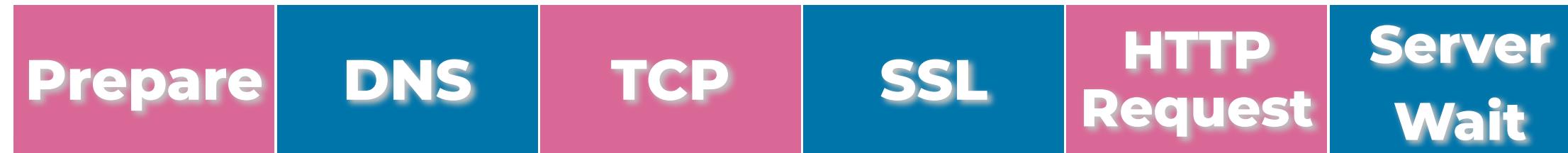
**TCP connection follows**



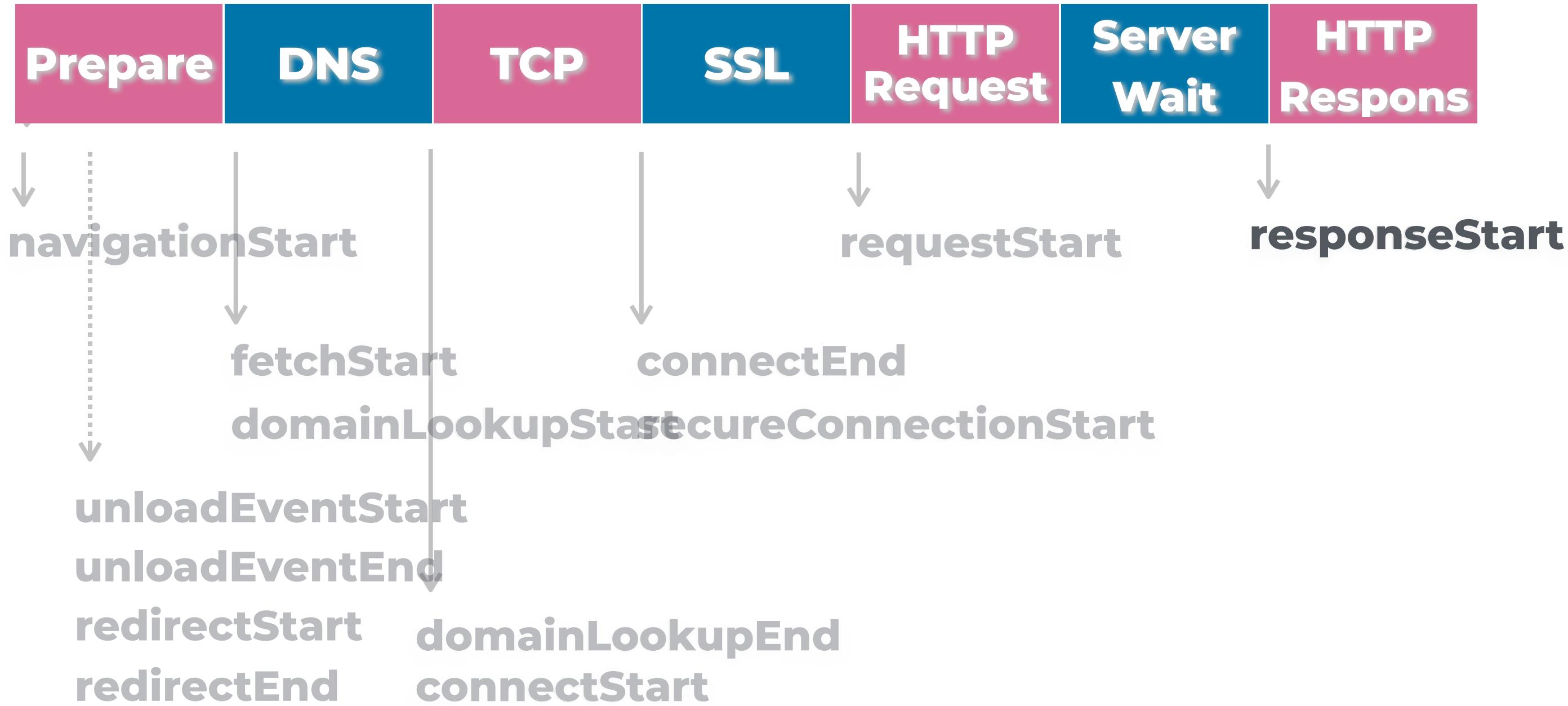
# TCP Opens a connection



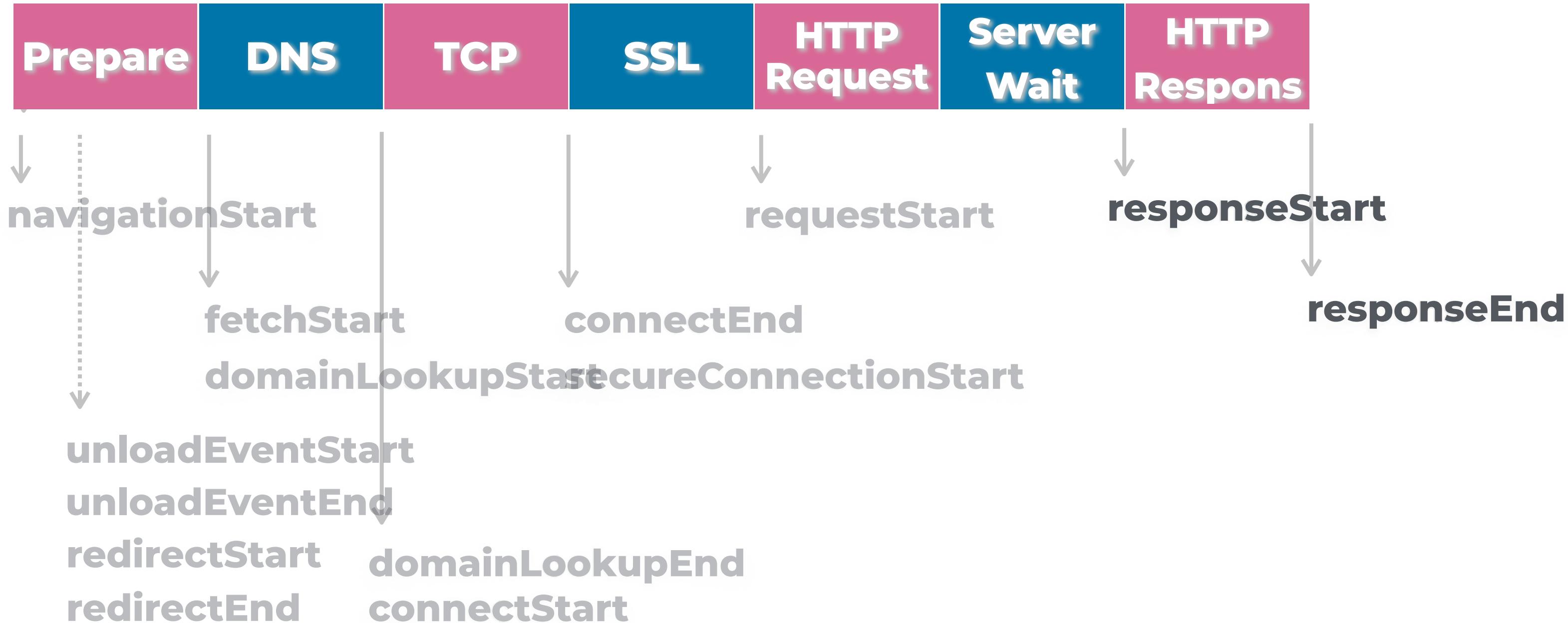
If on HTTPS, a SSL negotiation starts



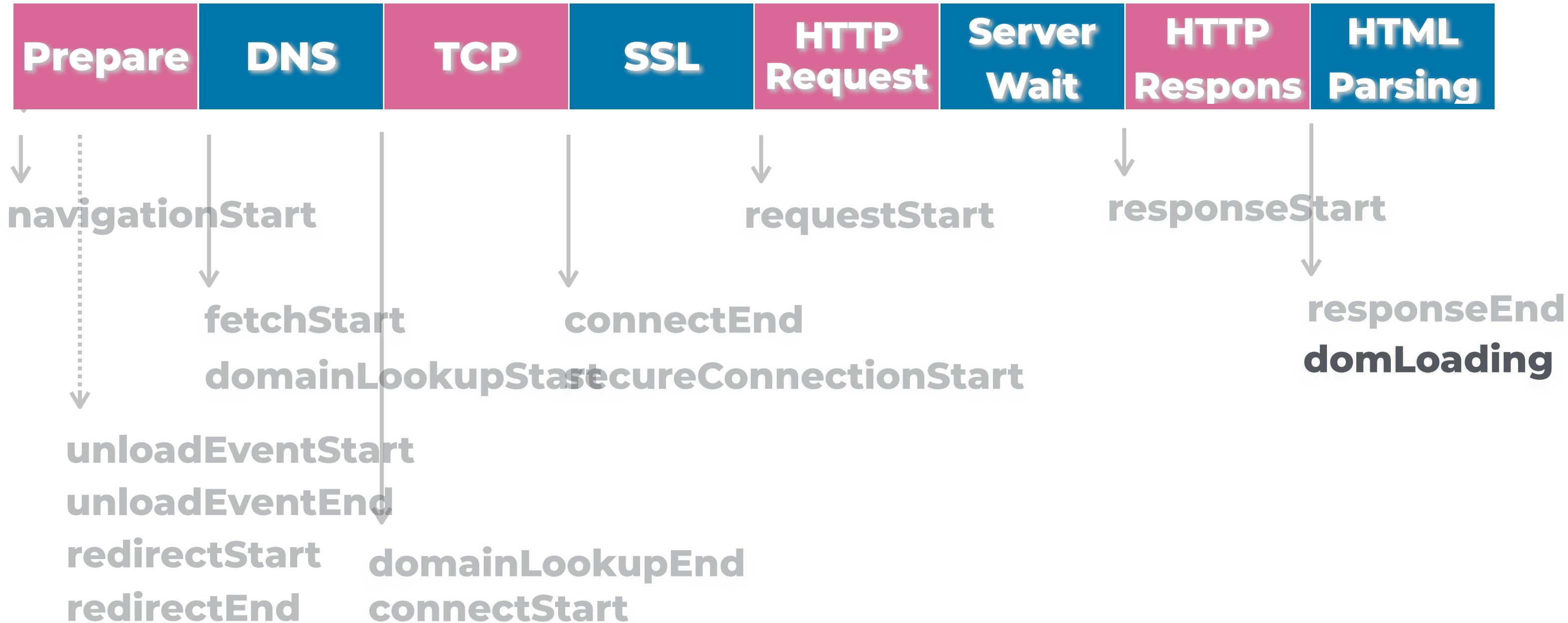
**The HTTP Request is sent to the server**  
**The server takes its time to get it and find the response**



**The response appears in the browser**

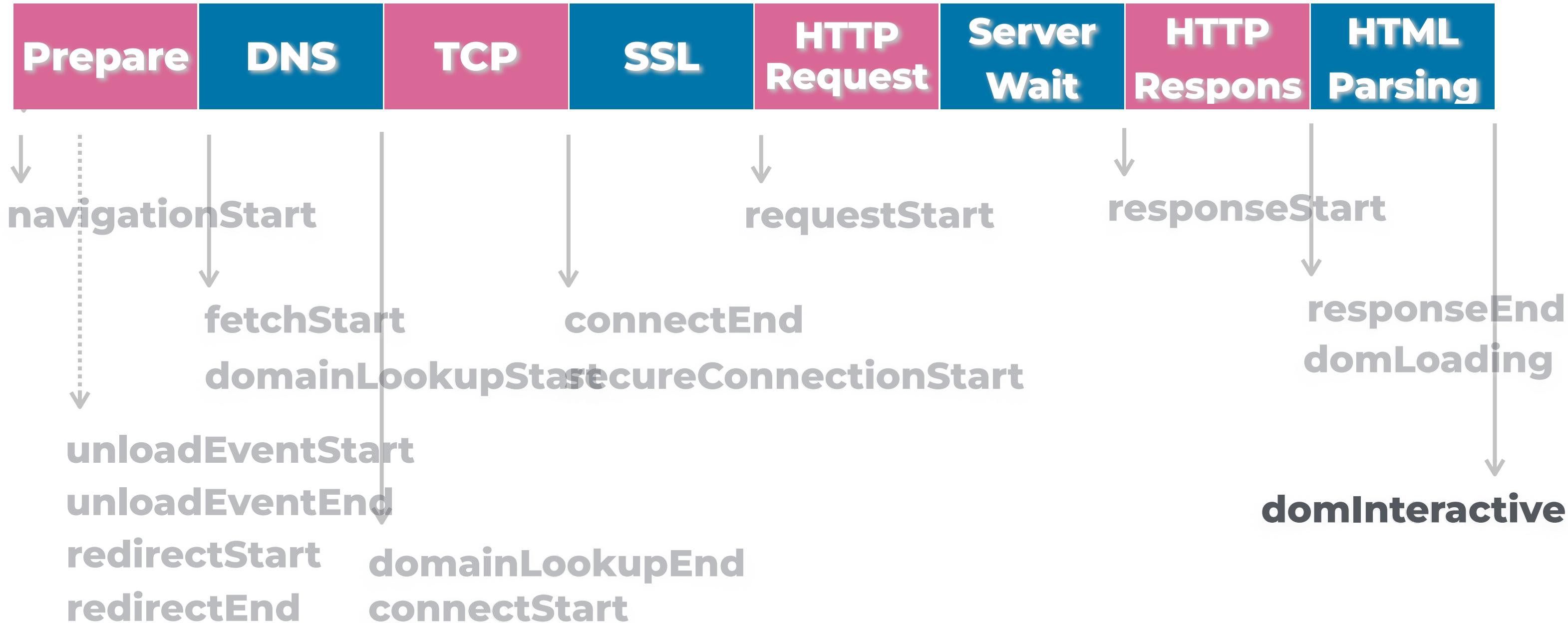


**At one point, it's downloaded**



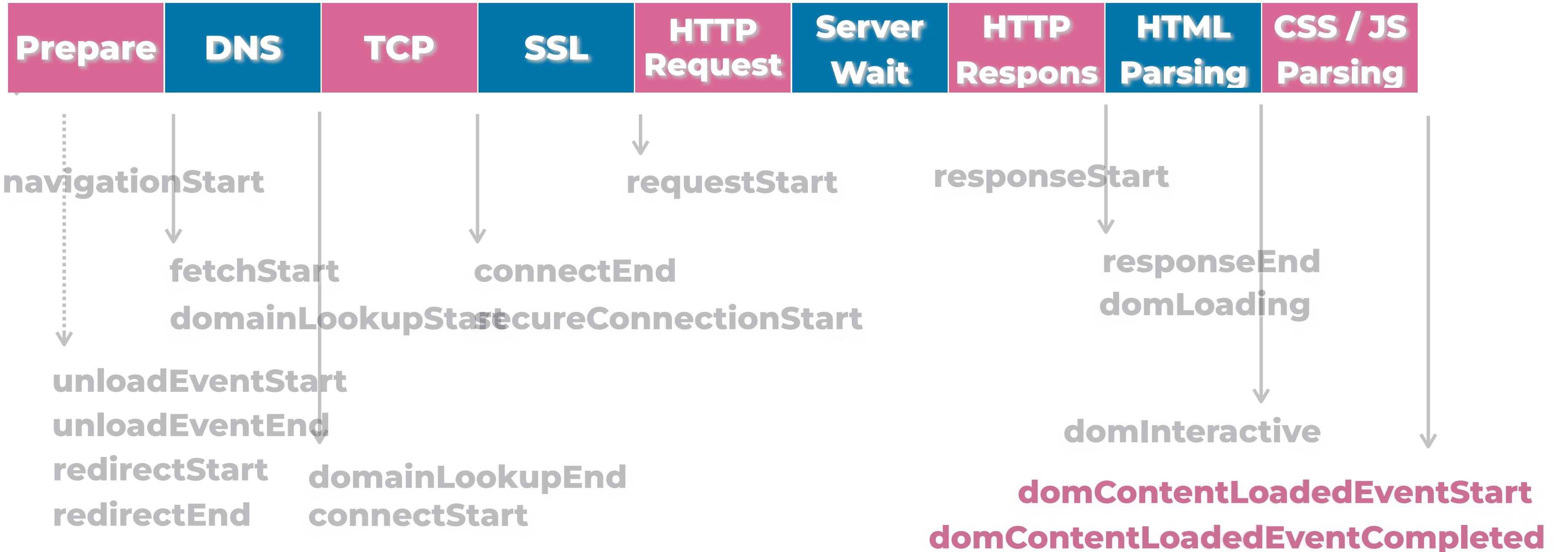
## The HTML Parsing starts

This process also can execute or download and  
execute blocking scripts



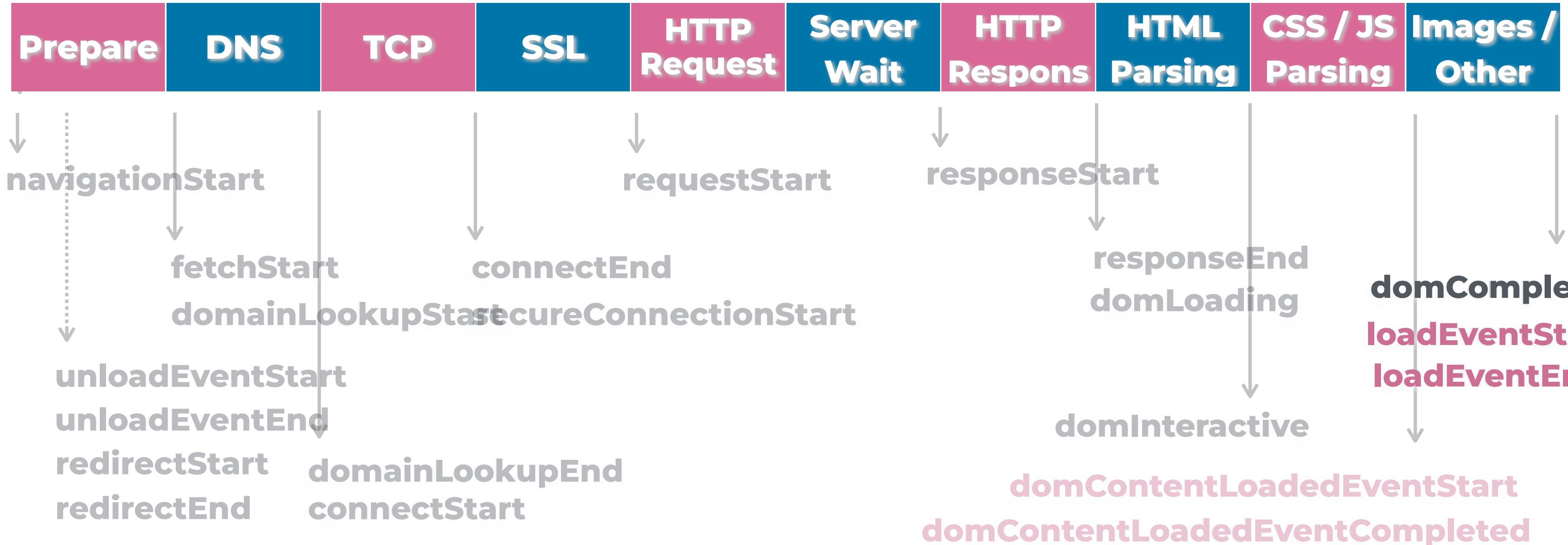
**The HTML parsing has finished**

**Next step is to download and parse necessary CSS files and/or some blocking scripts**



# DOM is ready

Means that all the CSS has been parsed, and  
blocking JavaScript scripts have been executed

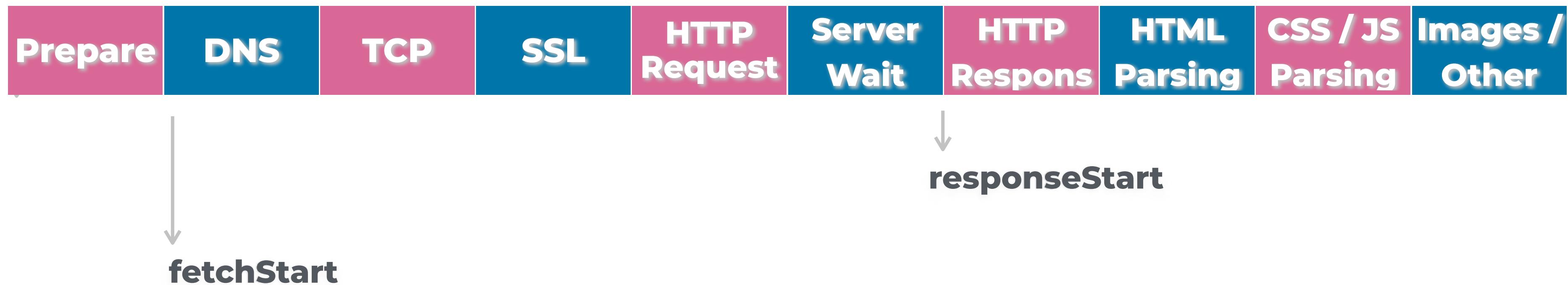


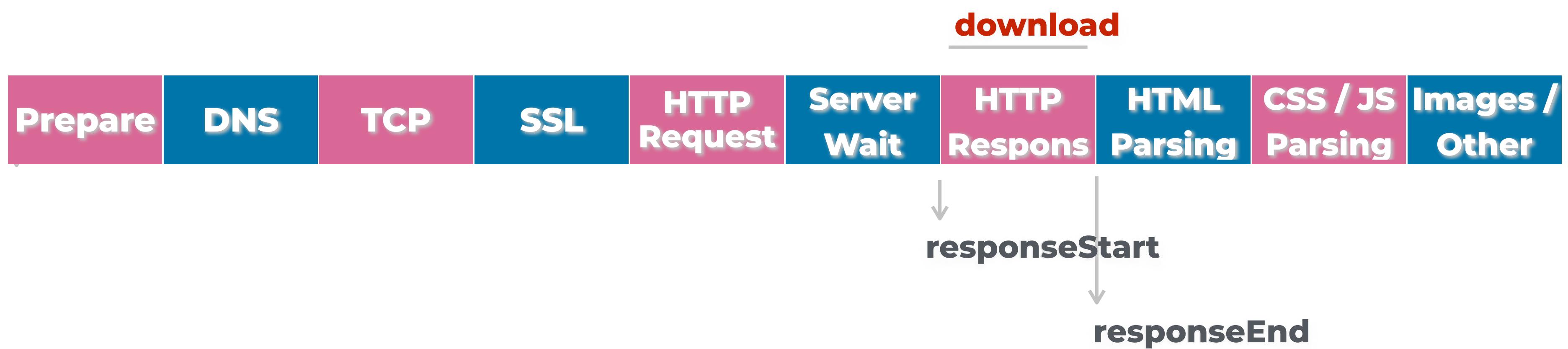
**The DOM is complete**

All the resources were downloaded and parsed.

The onload event handler executes.

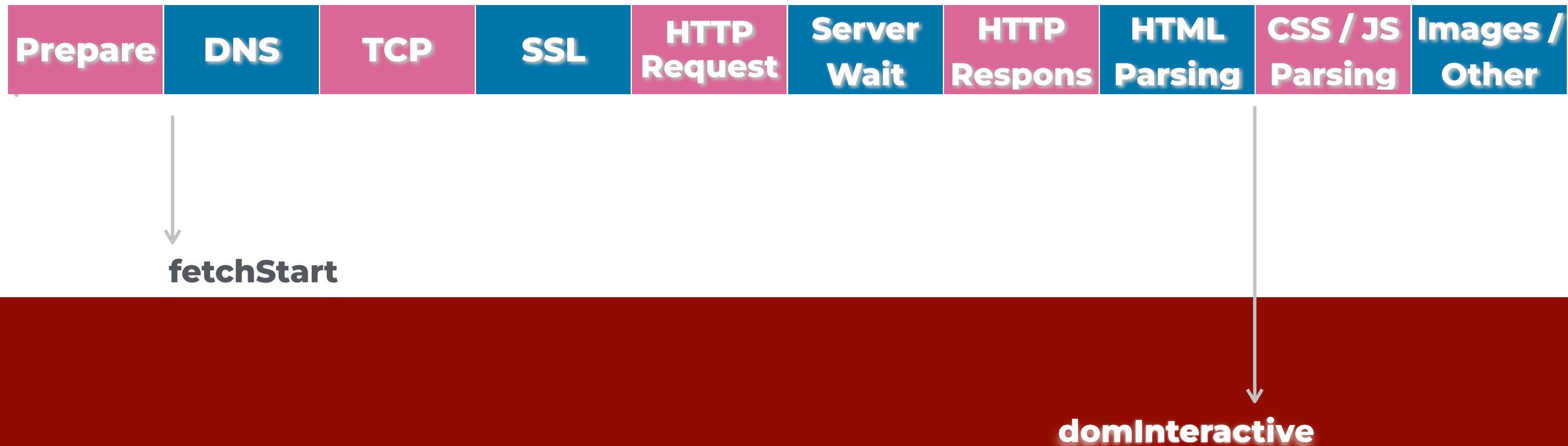
**time to first byte**



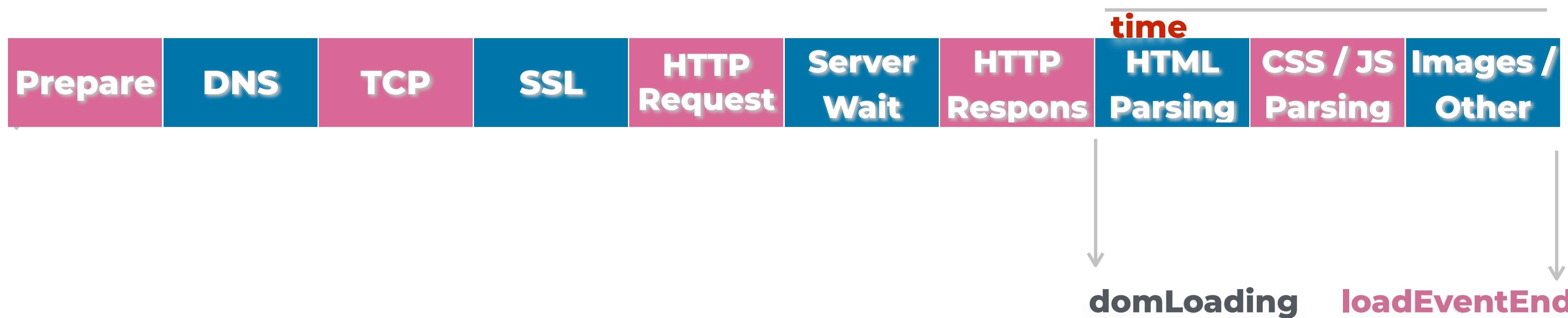


## DOM load time

---

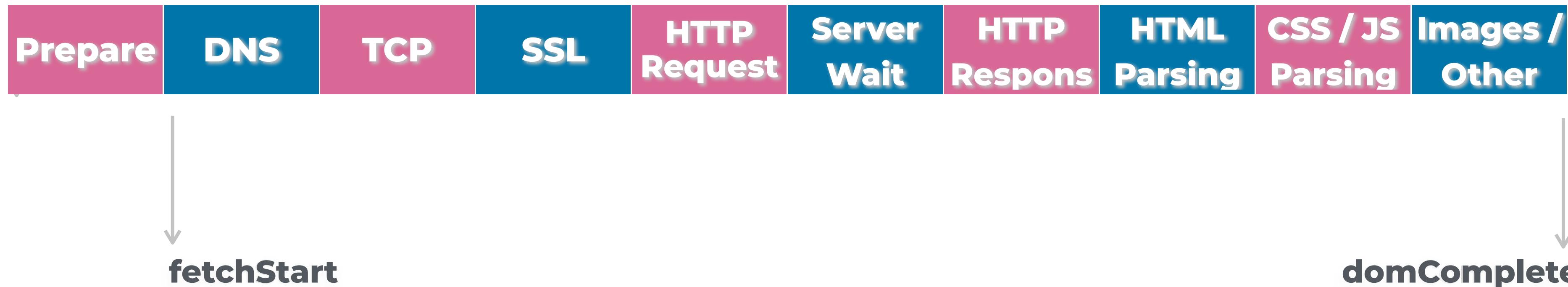


## parsing and rendering



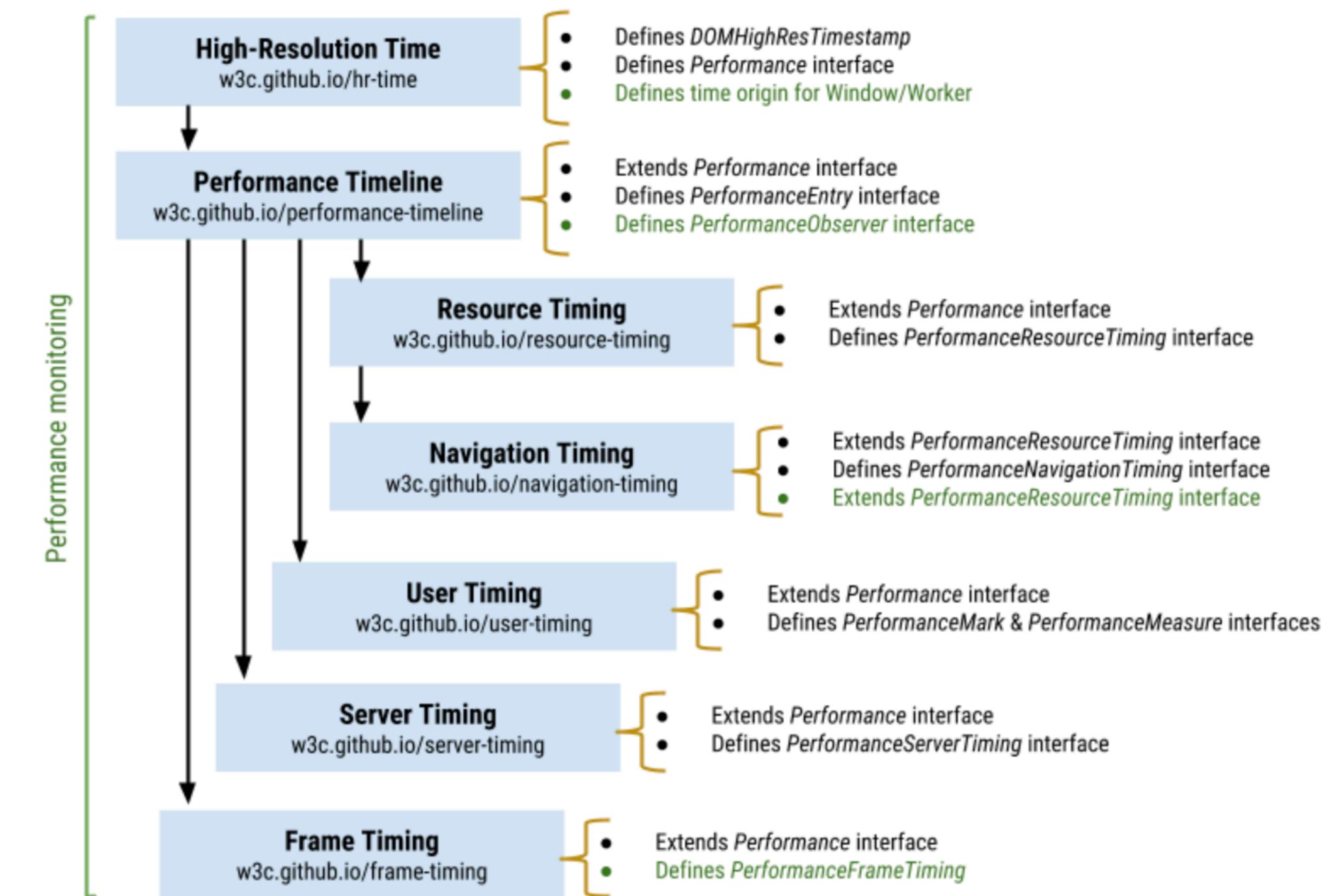
## Page Load Time

---



# Web Performance Working

- Maintain different specs for Web Performance
- Most of them inherit from Performance Timeline Interface



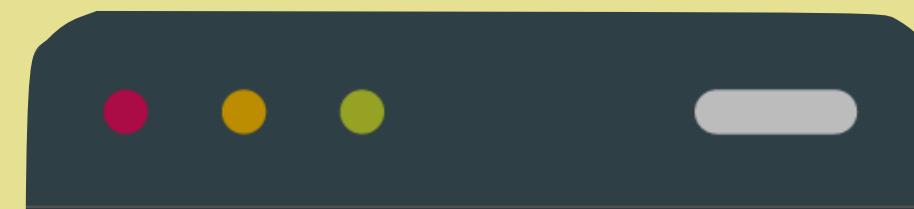
# Performance standard

- "performance" property in window and Worker's contexts
- It works with collections of metrics, known as "entries"
- We can query entries using filters with getEntries(filter)
  - ... by name with getEntriesByName(filter, type)
  - ... by type with getEntriesByType(type)

# Entries' Types

- "**navigation**" and "**frame**": for main navigation timings  
**(Navigation Timing 2)**
- "**resource(Resource Timing)**
- "**mark(User Timing)**
- "**measure**(User Timing)****

# Getting Entries (deprecated)



```
const entries = performance  
    .getEntriesByType("navigation");
```

# PerformanceEntry Properties

- **name**
- **entryType**
- **startTime**
- **duration**

# Performance Observers

# Performance Observers

- **getEntries methods should be called after the events actually happened**
- **A new version of the Performance Timeline (L2) let us create observers**
- **They will be executed every time there is new data in one particular entry type**

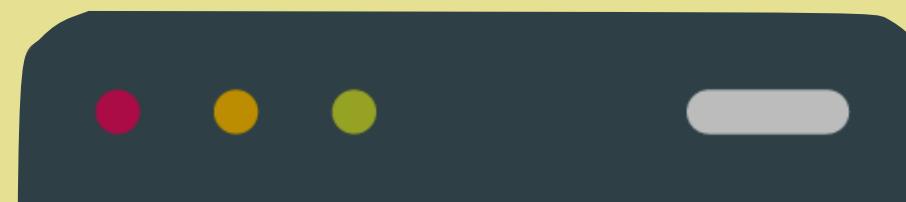
# Performance Observers

```
> PerformanceObserver.supportedEntryTypes
< ▶ (11) ["element", "event", "first-input", "largest-contentful-paint", "layout-shift", "longt
  ↴ ask", "mark", "measure", "navigation", "paint", "resource"] ⓘ
-----  

> PerformanceObserver.supportedEntryTypes
< ["mark", "measure", "resource"] (3) = $2
-----  

>> PerformanceObserver.supportedEntryTypes
← ▶ Array(5) [ "mark", "measure", "navigation", "paint", "resource" ]
```

# Performance Observers



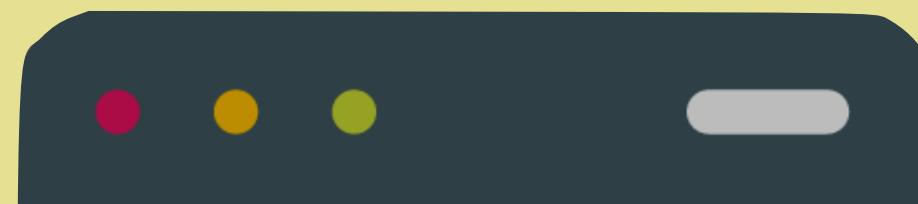
Add text here

```
const obs = new PerformanceObserver(list => {  
  const entries = list.getEntries();  
  
});  
  
obs.observe({entryTypes: ["resource"], buffered: true});
```

# Network Information API

- Available on some browsers
- Different versions of the spec are out there
- It's available in the navigator, in window and worker's space
- Decimal units ~ 1KB = 1000 bytes

# Get connection information



Add text here

`navigator.connection.type`

`navigator.connection.effectiveType`

`navigator.connection.rtt`

`navigator.connection.downlink`

`navigator.connection.downlinkMax`

`navigator.connection.saveData`

**rtt**

Round Trip Time or Latency, in milliseconds

<50ms: fast wired connection

50-200ms: Wi-Fi or 4G

200-450ms: 3G or 4G

# downlink

Effective Bandwidth estimate in Mb/s  
rounded to 25Mb/s

# **type\***

Calculated based on OS report

Possible String Values: **bluetooth, cellular, ethernet, none, mixed, other, unknown, wifi, wimax**

**\*not yet available on all browsers**

# effectiveType

Calculated based on OS report,  
bandwidth and/or RTT

Possible String Values: **slow-2g, 2g, 3g,**  
**4g**

# effectiveType

slow-2g: only text

2g: small images

3g: hi-res images, sd video

4g: hd-video

# downlinkMax

Maximum Effective Bandwidth in Mb/s,  
based on the underlying technology

# **type\***

Calculated based on OS report

Possible String Values: **2g, 3g, 4g, ethernet, wifi**

\*Deprecated from the spec

# **metered\***

\*Deprecated from the spec

# Detecting changes



Add text here

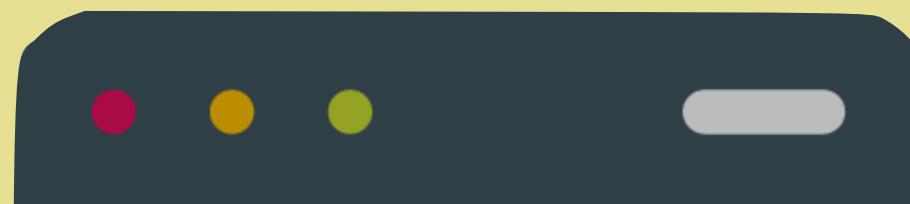
```
navigator.connection
  .addEventListener("change", () => {
});
```

```
// Deprecated but still needed
navigator.connection
  .addEventListener("typechange", () => {
});
```

# Device Memory API

- Available on some browsers
- It gives us a number expressed in GiB (gibibytes)
- 1 Gib = 1,024 MiB

# Get Total Memory



Add text here

```
const gb = navigator.deviceMemory;
```

# deviceMemory

0.25 GiB: minimum value

< 1 GiB: low end device

1-2 GiB: mid end device

2-4 GiB: high end device

4-8 GiB: probable desktop

8 GiB: maximum value

# Save Data Flag

- Available on some browsers
- It's a boolean Flag
- It sits on top of Network Information API

# Get Save Data Flag

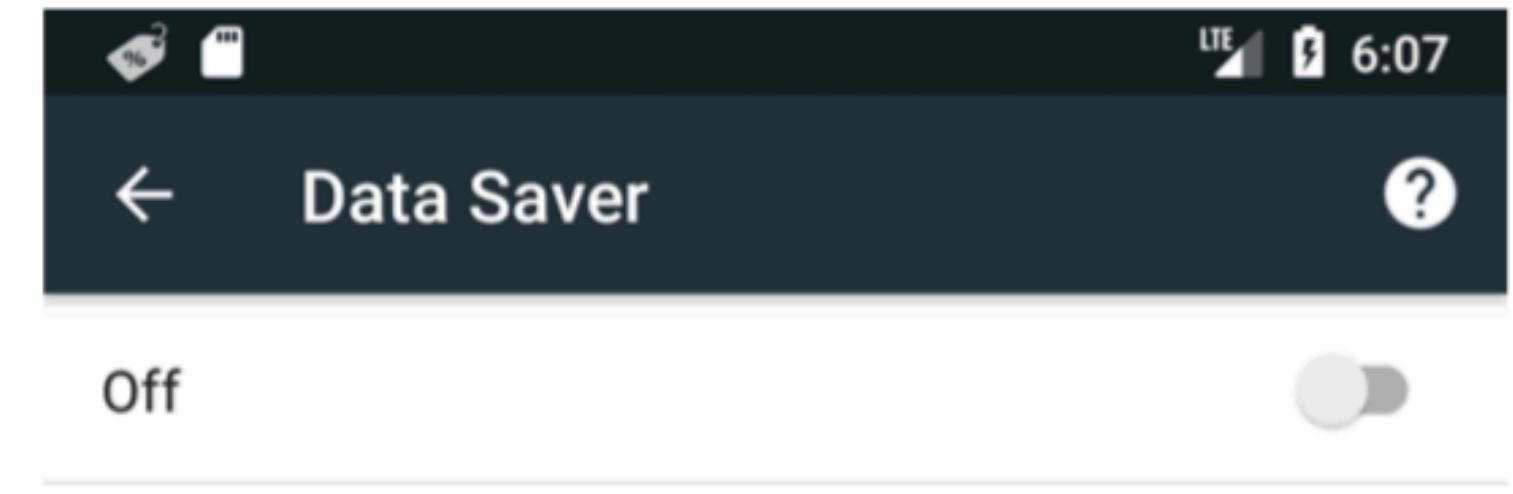


Add text here

```
if (navigator.connection.saveData) {  
}  
}
```

# saveData

If true, we should  
reduce the data we  
consume from our  
webapp



When this feature is turned on, Chrome will use Google servers to compress pages you visit before downloading them. Pages accessed using private connections (HTTPS) or in Incognito tabs will not be optimized or seen by Google.

Chrome's Safe Browsing system will also be used to detect malicious pages and protect you from phishing, malware, and harmful downloads.

This feature may interfere with access to premium data services provided by your carrier.

[Learn more](#)

# Client-side vs server-side

- NetInfo, Device Memory, Responsive Queries and Save-Data are client-side accessible only
- Sometimes we want to access the data in the server
- Apply Reactive Web Performance techniques
- Serve better Responsive Images

# Meet Client-Hints

- HTTP spec to opt in for hints from the client to the server
- Through a meta tag or a Response header
- Let us make decisions on the server
- It can be polyfilled with Service Workers or Cookies

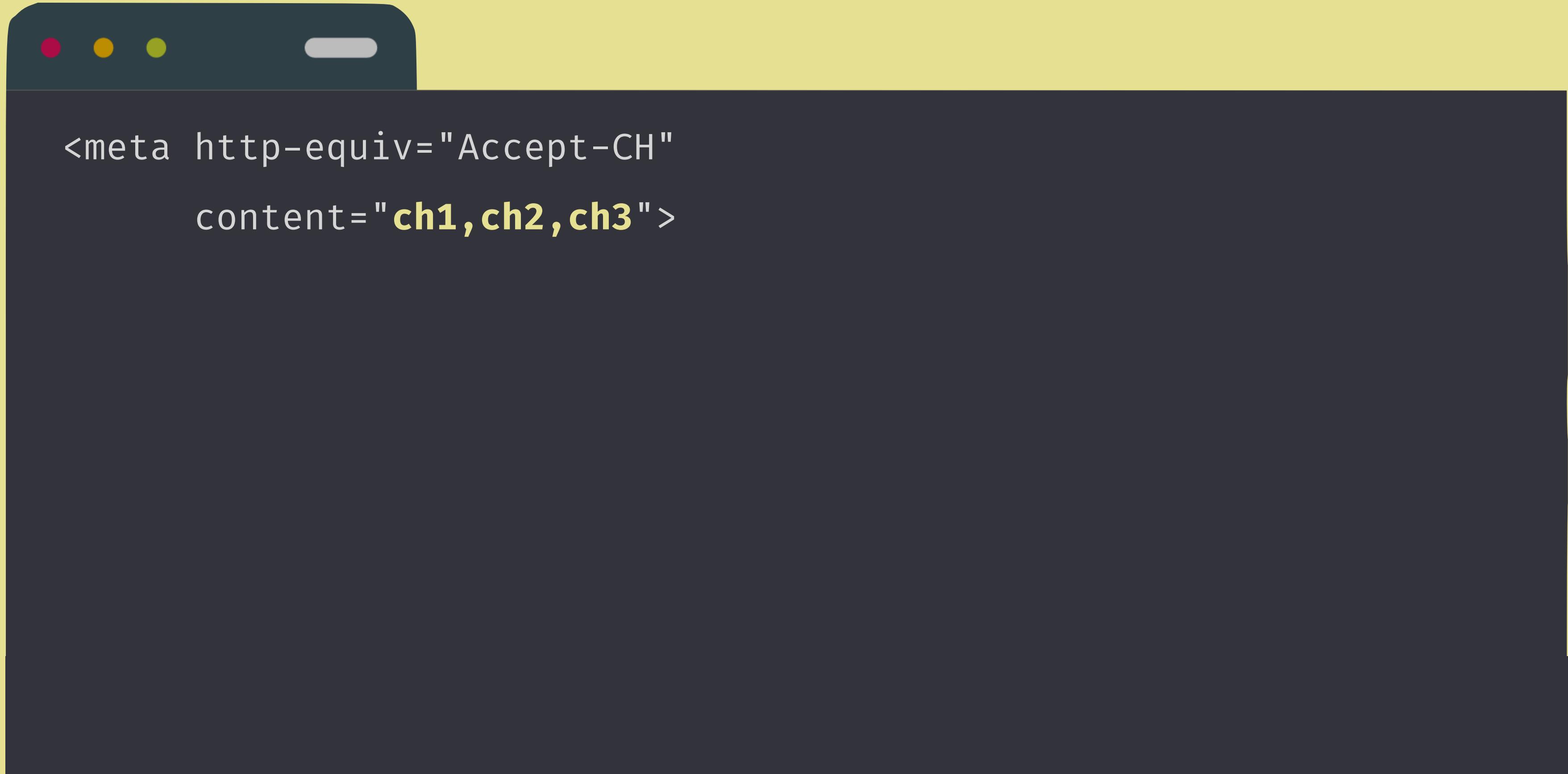


**The Server requests Client Hints**  
**It's an HTTP response header meta tag in the HTML**



**Future resource requests will have CH  
As HTTP request headers**

# Opt-in for Client Hints



# Opt-in for Client Hints



# Possible Client-Hints available

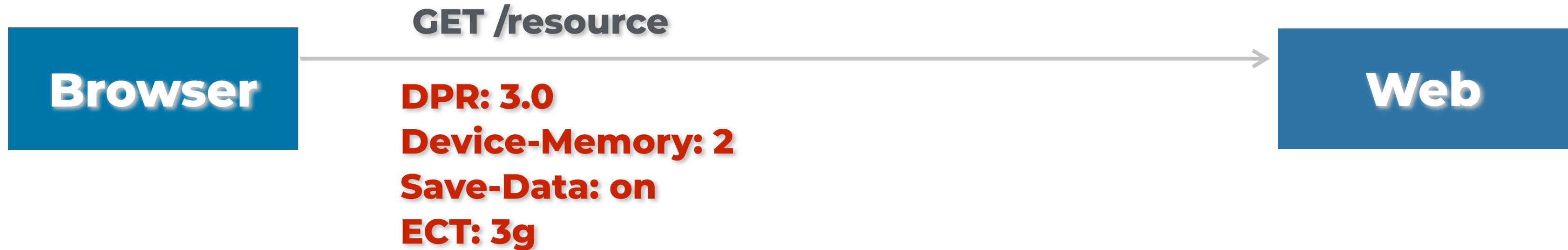
- Width: expected resource width in physical pixels
- Viewport-width: current layout width in CSS pixels
- DPR: device's pixel ratio (resolution)

# Client-Hints from NetInfo and

- RTT: round trip time
- Downlink: estimated Bandwidth
- ECT: effective connection type, such as slow-2g, 2g, 3g, 4g
- Save-Data
- Device-Memory: GiB of memory based on Device Memory

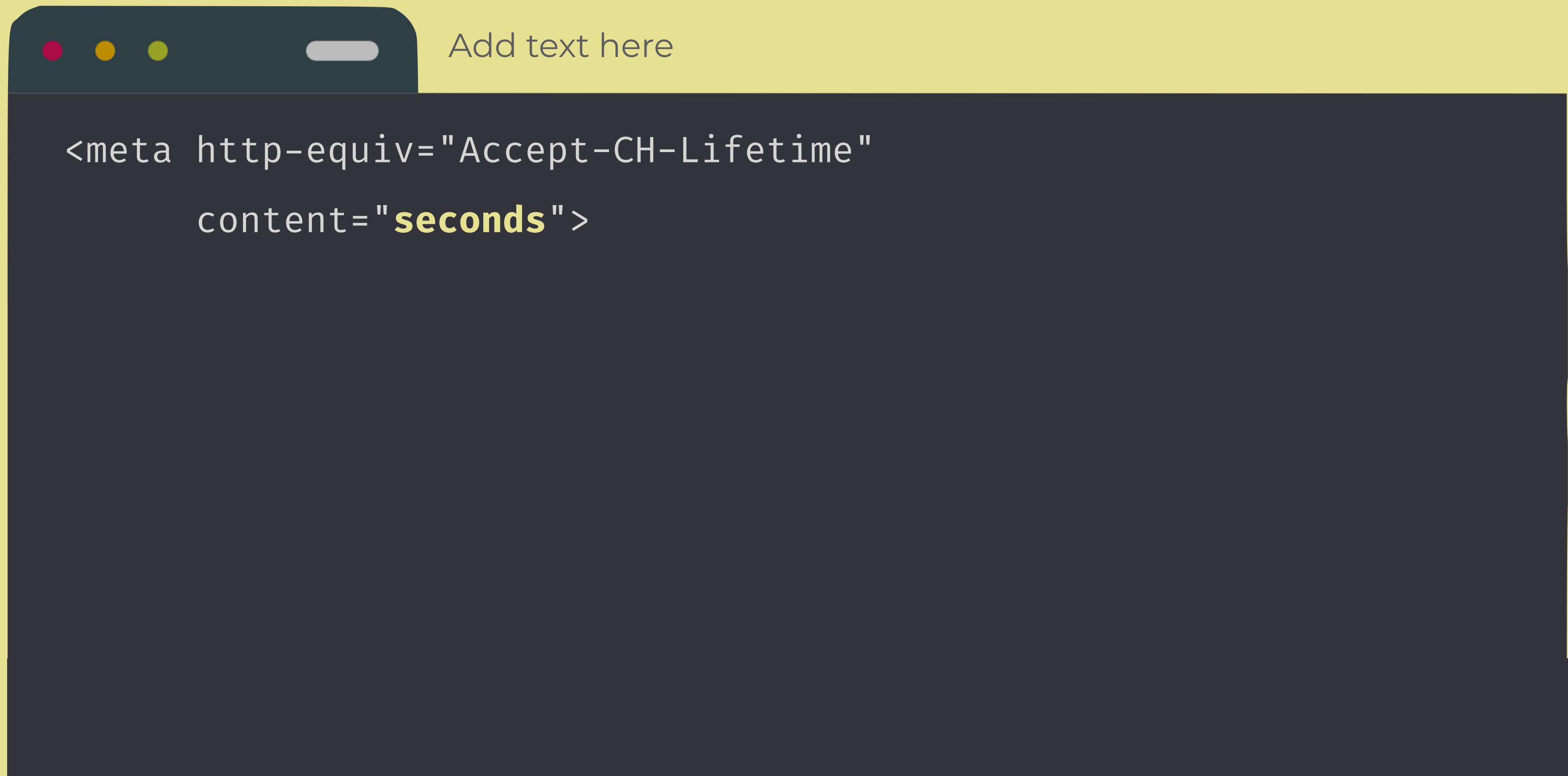


**The Server requests Client Hints  
It's an HTTP response header meta tag in the HTML**



**Future resource requests will have CH  
As HTTP request headers**

# Specify CH cache



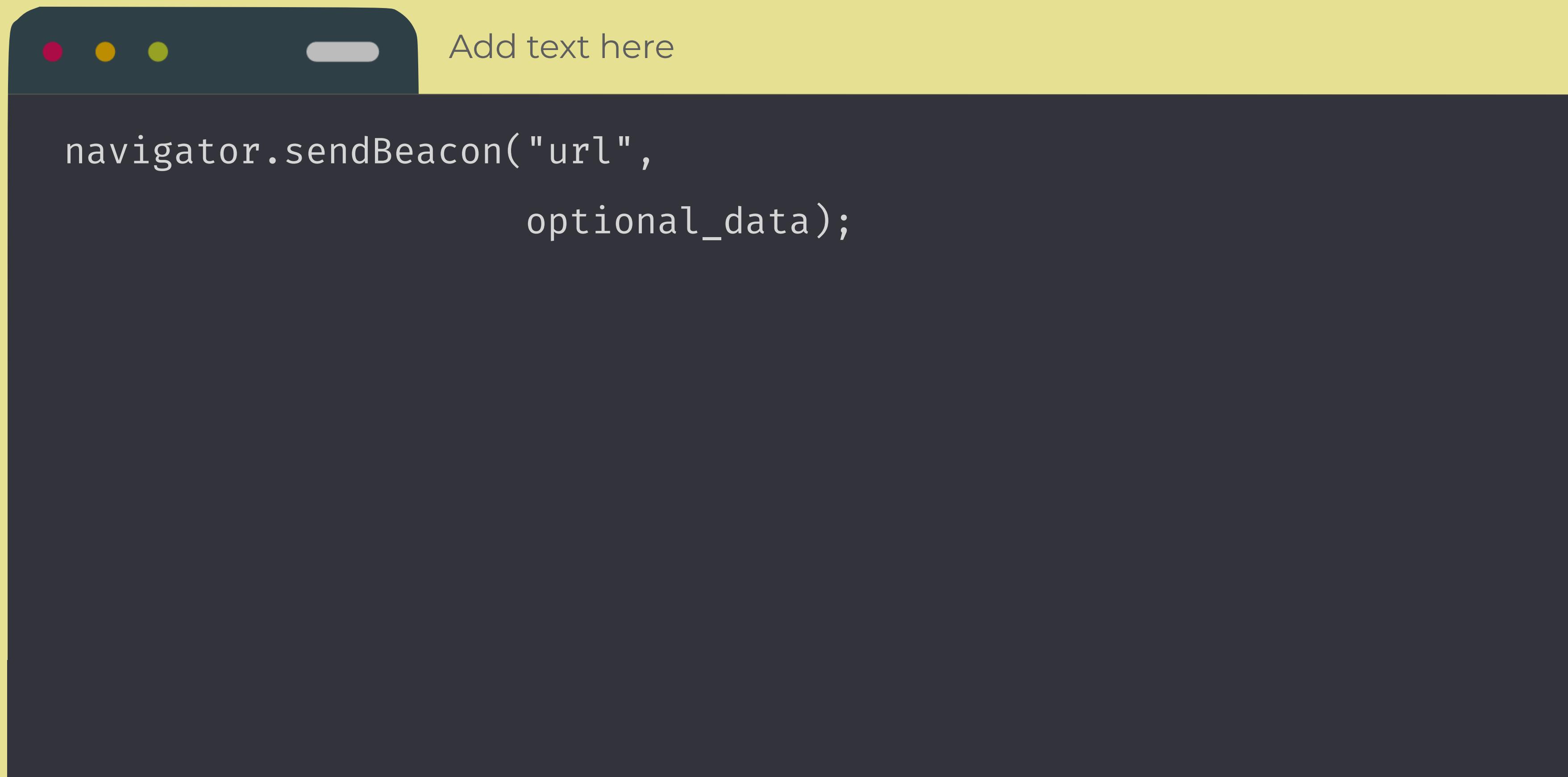
# Lifetime

- Useful for future visits
- Client Hints don't work on initial HTML request

# Beacon API

- Useful for Analytics, Trackers, State Updates
- Waiting for a response is not important
- They should not be high priority
- The requests should be sent even if the navigation context is lost

# Beacon API



# Beacon API

- The browser will send the beacon, even if the user is on a different page
- Or even if the tab is closed
- It will be low priority

# Understand frames

- The browser renders frames
- We want to achieve 60 fps for better performance
- To achieve that we need a new approach

**Frame**

**Input Events**

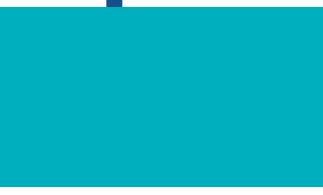
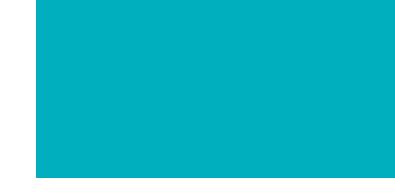
**Parse HTML,  
Styles, Layout**

**Paint and  
Composit**

**Idle**

**Timers**

**Frame**





**Long Script**

# Problems

- Frame rate drop
- Some frames won't execute your code
- Low-priority code should wait for idle time, but when is it happening?

**Frame**

**Input  
Events**

**Animation  
on Frame**

**Parse HTML,  
Styles,**

**Paint and  
Composite**

**Frame**

**Idle  
Callback**

**Guaranteed**

**using idle  
available time**

**New APIs are available**

**Frame**

**Frame**

**Input  
Events**

**Animation  
Frame**

**Parse HTML,  
Styles,**

**Paint and  
Composite**

**Idle  
Callback**

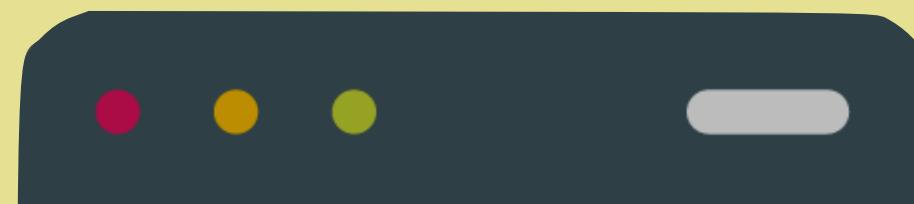
**Guaranteed**

**New APIs are available**

# Animation Frames

- Part of the latest HTML5 spec
- Let us execute code on next frame before painting it
- It's guarantee to be executed on next frame
- Better than timers
- We should keep code small (<10ms)

# Request Animation Frame



Add text here

```
requestAnimationFrame( () => {  
    // code for next frame  
});
```



# Request Background Task

**Frame**



**Frame**

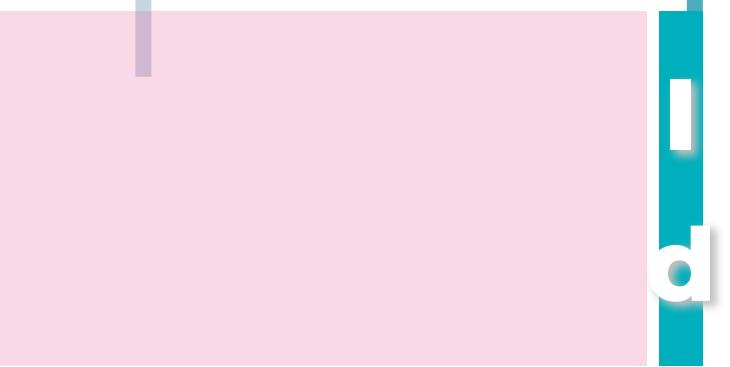
**Idle Callback**

**using idle  
available time**

**Frame**



**Frame**

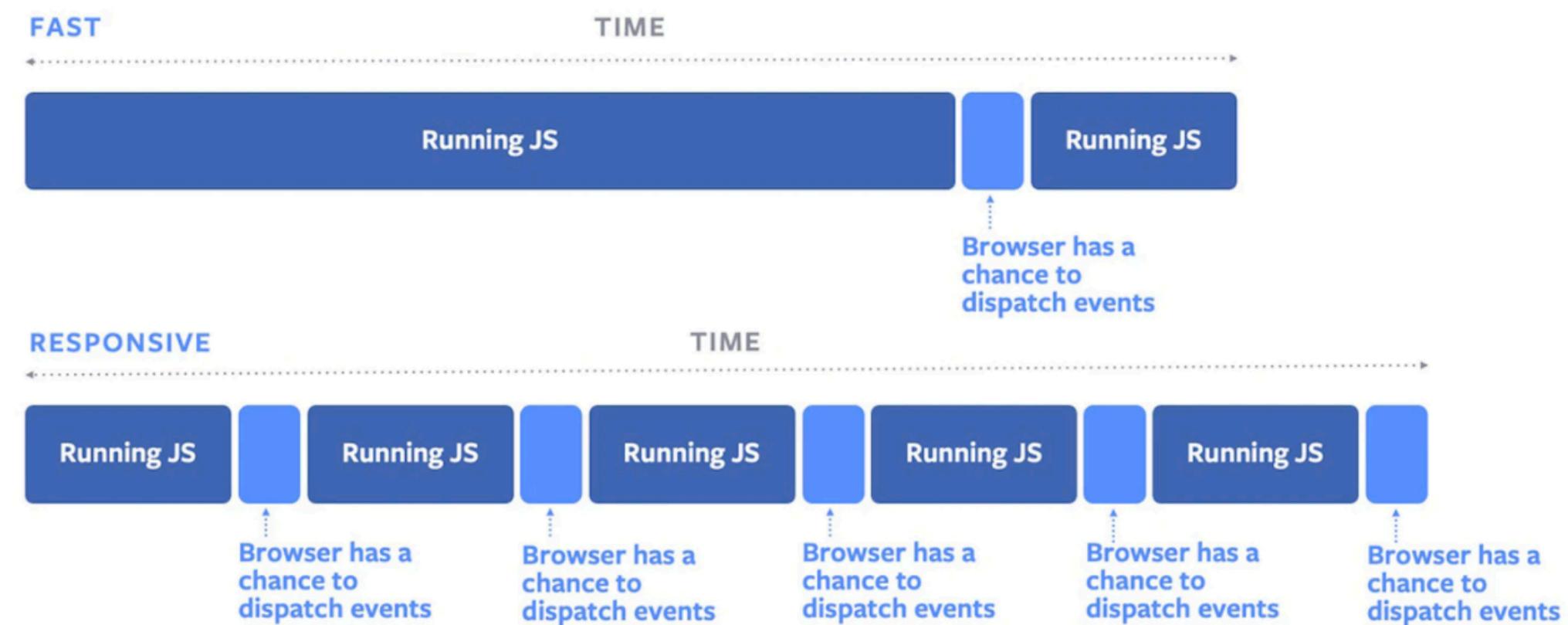


**using idle  
available time**

# Is Input Pending

- Also known as `isInputPending`

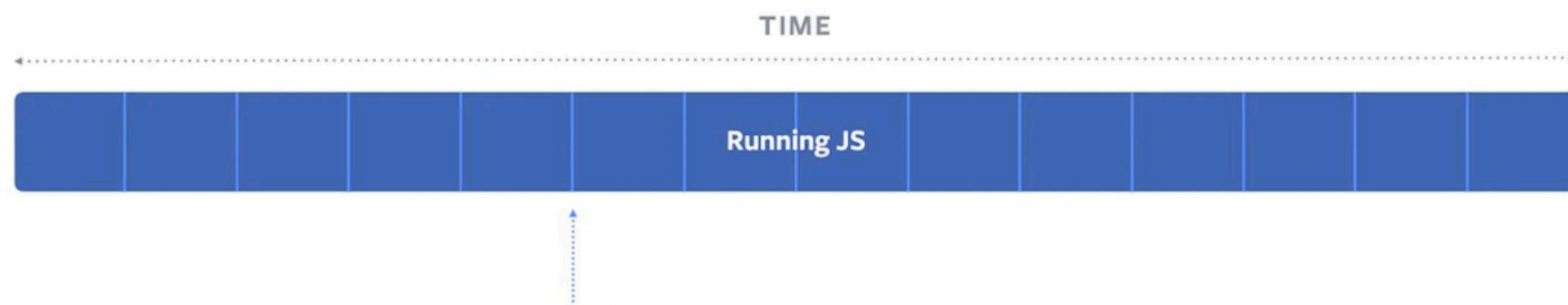
**Load fast or respond fast?**



# Is Input Pending

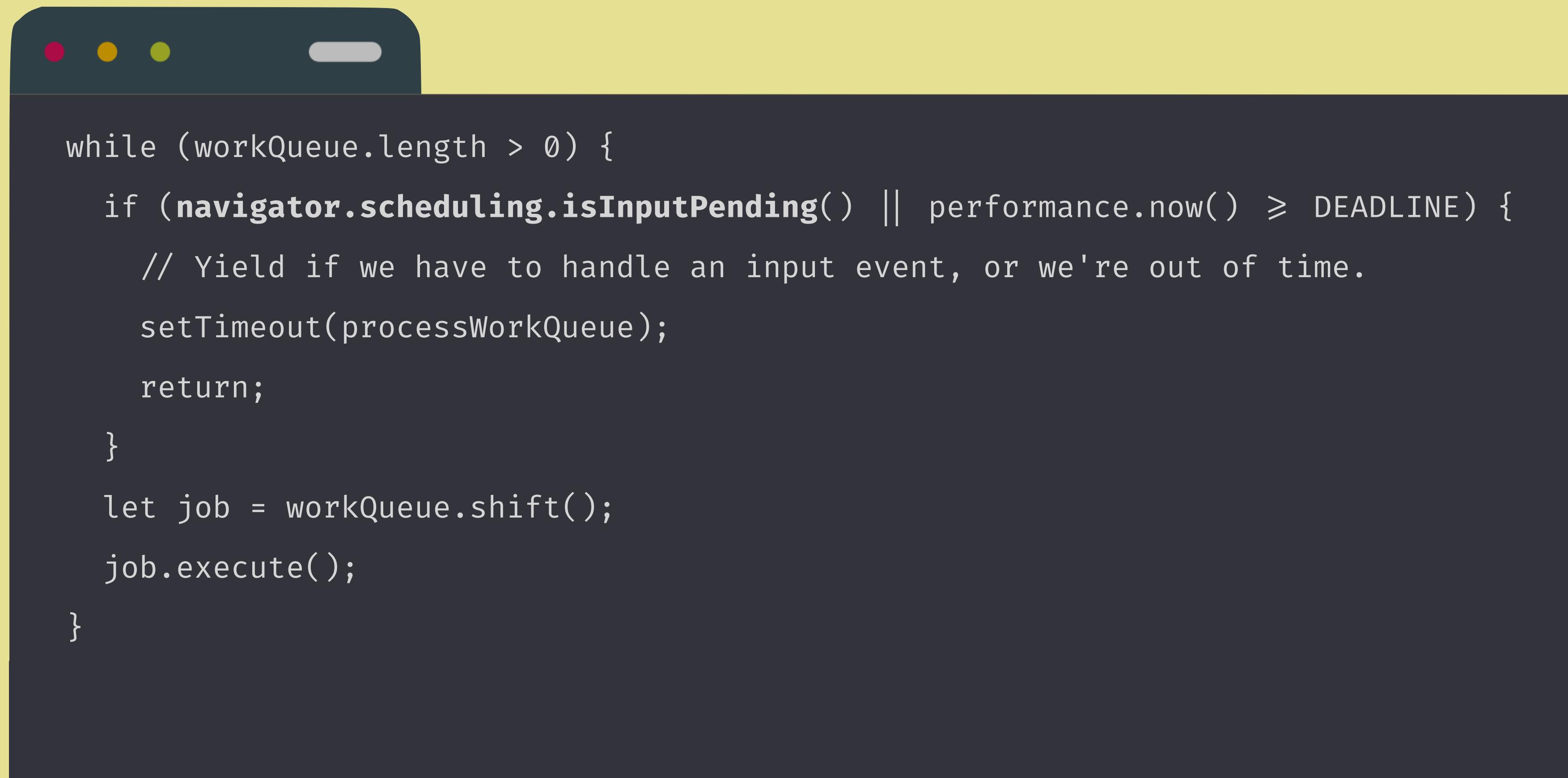
- Also known as `isInputPending`

## `isInputPending`



What each line represents:  
We proactively check whether  
there's user input, without  
incurring the overhead of yielding  
execution to the browser and back.

# Is Input Pending?

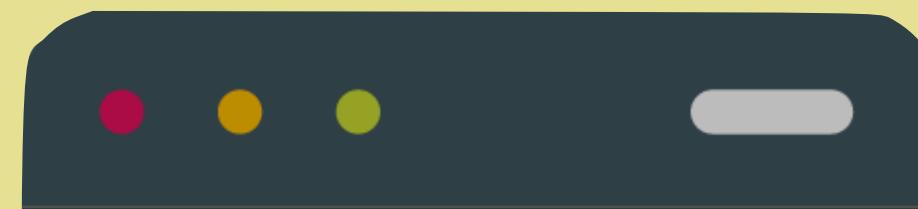


```
while (workQueue.length > 0) {  
  if (navigator.scheduling.isInputPending() || performance.now() ≥ DEADLINE) {  
    // Yield if we have to handle an input event, or we're out of time.  
    setTimeout(processWorkQueue);  
    return;  
  }  
  let job = workQueue.shift();  
  job.execute();  
}
```

# Request Idle Callback

- W3C Cooperative Scheduling of Background Tasks
- They execute low-priority code before each frame, if there is enough idle time
- It sends an argument where we can query how much time is left before idle time is gone and next frame should start
- We should stop and schedule a new Idle Callback if time is zero

# Request Idle Callback

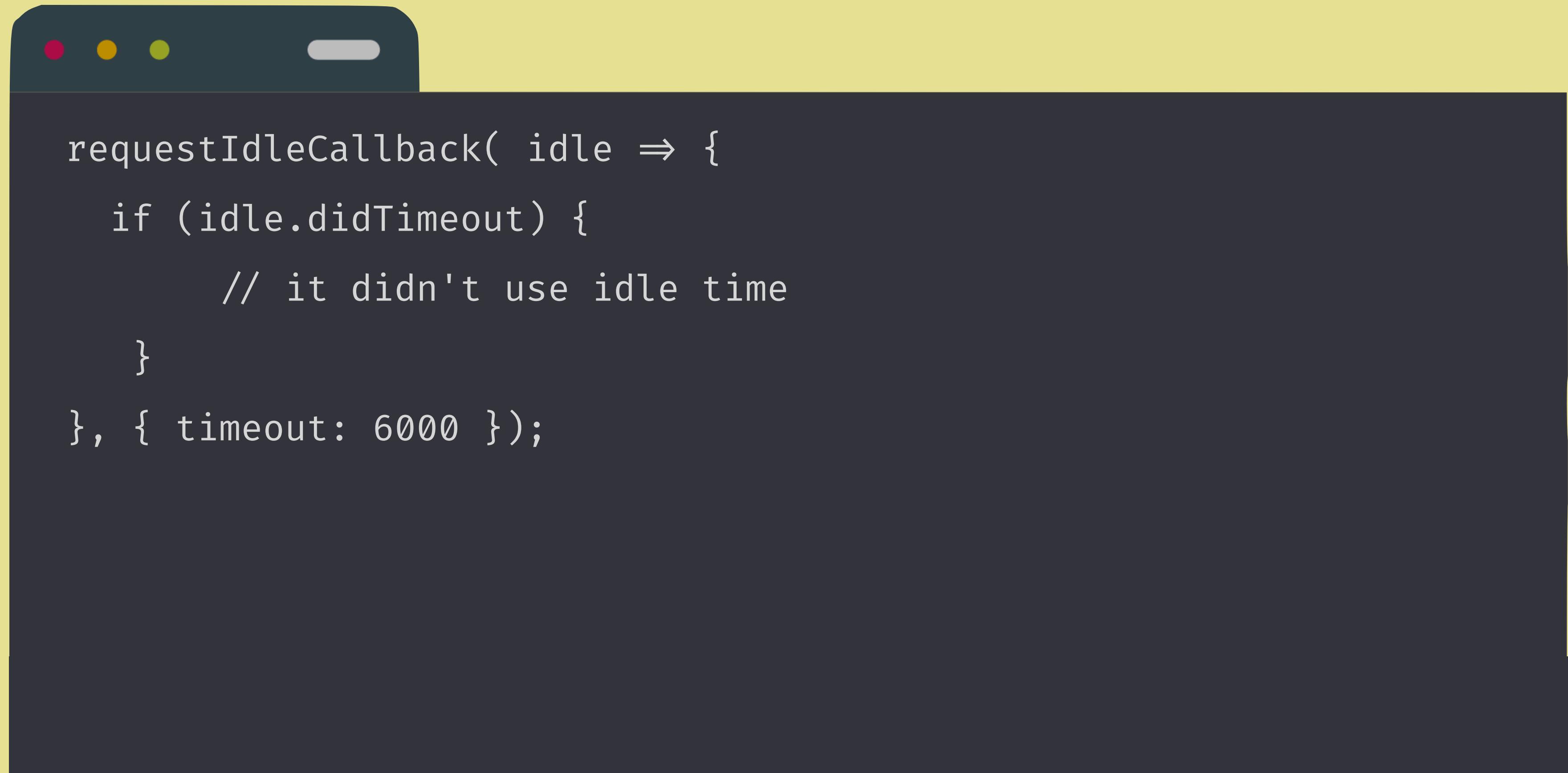


```
requestIdleCallback( idle => {  
    // low priority code  
    if (idle.timeRemaining()=0) {  
        // timeout :(  
    }  
});
```

# What if there is no Idle time

- The browser will retry on each frame
- Unless we specify a timeout
- After the timeout passes, the browser will execute your code on next frame, even if there is on idle time on it

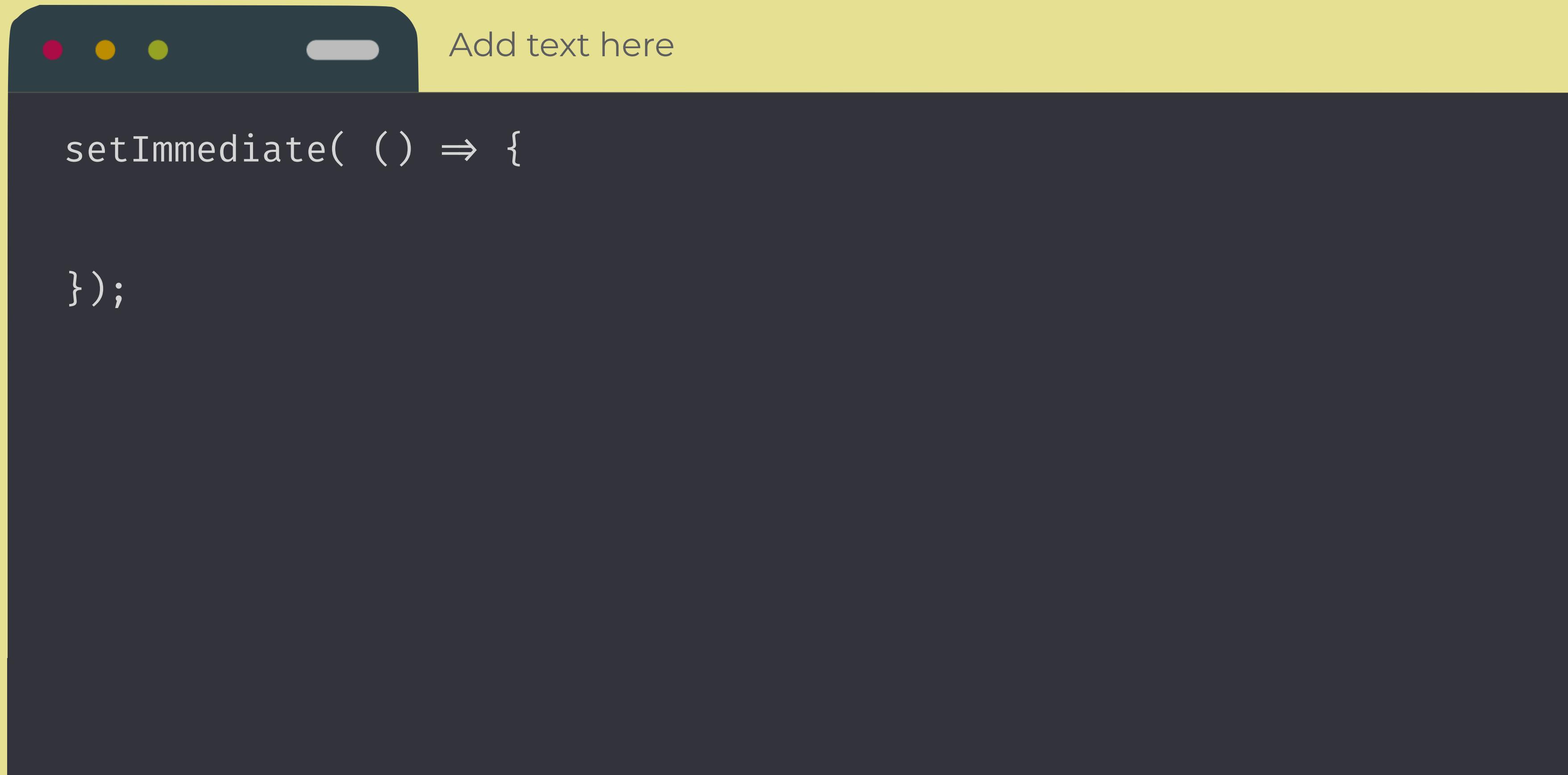
# Request Idle Callback



# Efficient Script Yielding

- A way to yield control to the browser before running our code
- Better than `setTimeout(code, 0)`

# Efficient Script Yielding



# What we've covered

The Problem

Metrics

Tools

Charts and Diagrams

Understanding Browsers

Basic Optimizations

Hacking Performance

Performance APIs



**IMPORTANT**

Performance is TOP PRIORITY

There are more tricks

Push the limits

It's a worthwhile effort

Remember what's the GOAL

**Feeling bad enough?**



Frontend **Masters**



# THANKS! 😊 ADVANCED WEB PERFORMANCE

MAXIMILIANO FIRTMAN

