



JAN  
2017

# GLOBAL INTERNET USE AND PENETRATION

INTERNET AND MOBILE INTERNET USER NUMBERS COMPARED TO POPULATION

TOTAL NUMBER  
OF ACTIVE  
INTERNET USERS



**3.773**  
BILLION

INTERNET USERS AS A  
PERCENTAGE OF THE  
TOTAL POPULATION



**50%**

TOTAL NUMBER  
OF ACTIVE MOBILE  
INTERNET USERS



**3.448**  
BILLION

MOBILE INTERNET USERS  
AS A PERCENTAGE OF  
THE TOTAL POPULATION



**46%**



we  
are  
social



JAN  
2017

# SHARE OF WEB TRAFFIC BY DEVICE

BASED ON EACH DEVICE'S SHARE OF ALL WEB PAGES SERVED TO WEB BROWSERS

LAPTOPS &  
DESKTOPS



**45%**

YEAR-ON-YEAR CHANGE:

**-20%**

MOBILE  
PHONES



**50%**

YEAR-ON-YEAR CHANGE:

**+30%**

TABLET  
DEVICES



**5%**

YEAR-ON-YEAR CHANGE:

**-5%**

OTHER  
DEVICES



**0.12%**

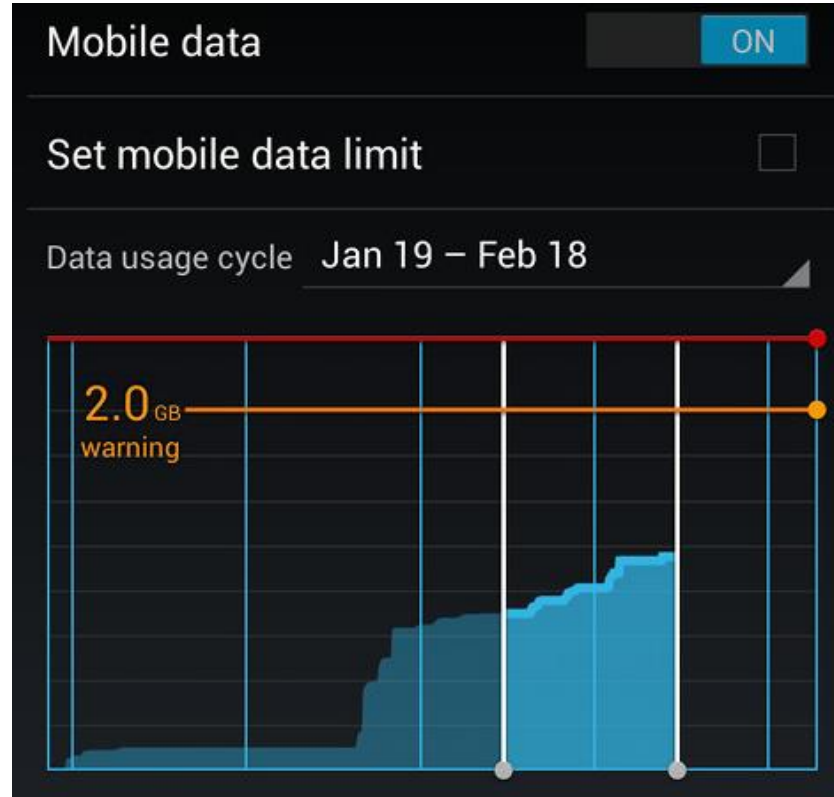
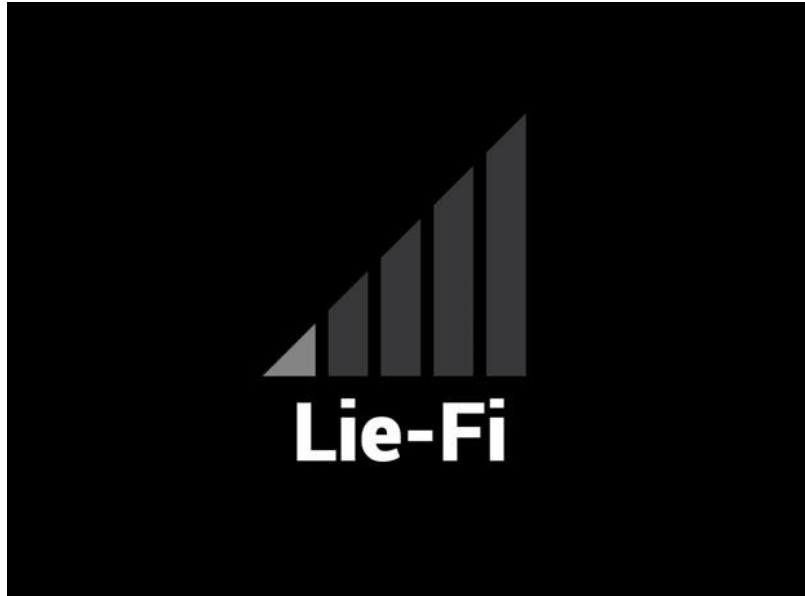
YEAR-ON-YEAR CHANGE:

**+33%**



we  
are  
social

StatCounter



**53%** of mobile site visits are abandoned if pages take longer than 3 seconds to load

<https://www.doubleclickbygoogle.com/articles/mobile-speed-matters/>

**70%** of mobile sites take longer than **10** seconds to load on 3G networks

<https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/>

**JS**



# Jakub Sowiński



# Practical tips for optimising JavaScript



1. Why?

**2. How to measure?**

3. How to improve?

a) Code manipulations

b) Code splitting

c) Caching

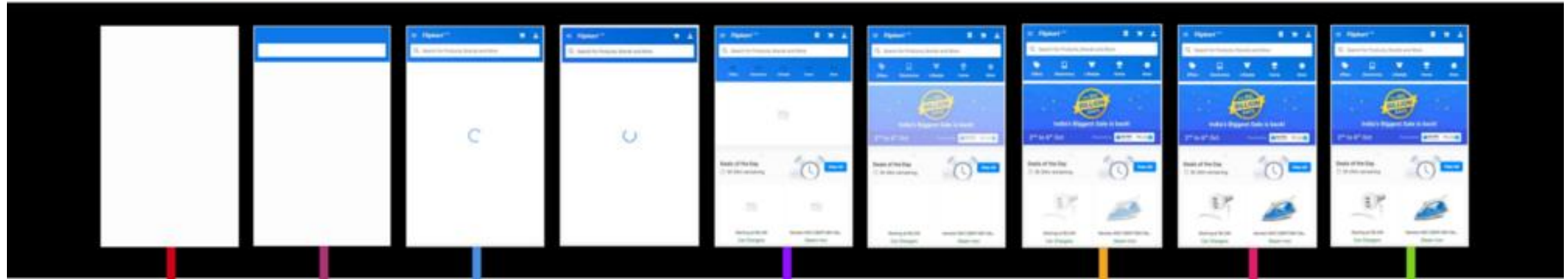


How to measure JS  
performance?

is it happening?

is it useful?

is it usable?



Navigation begins

First Contentful Paint

First Meaningful Paint

Visually ready

Fully Loaded

Time to first byte

Navigation has successfully started

Page's primary content is visible

Page looks nearly done

End of load lifecycle

First Paint

Time to Interactive

The first non-blank paint on screen

Visually usable and engaging

0.0s

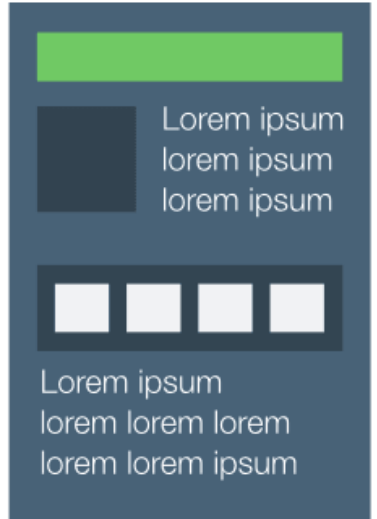
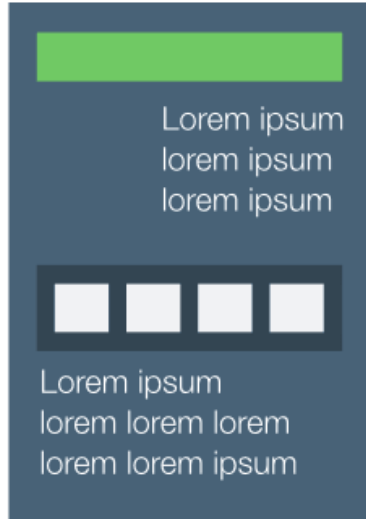
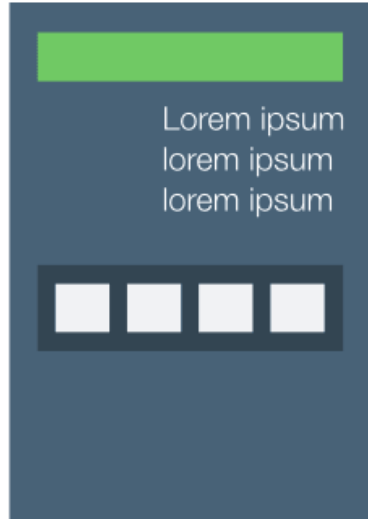
0.3s

0.8s

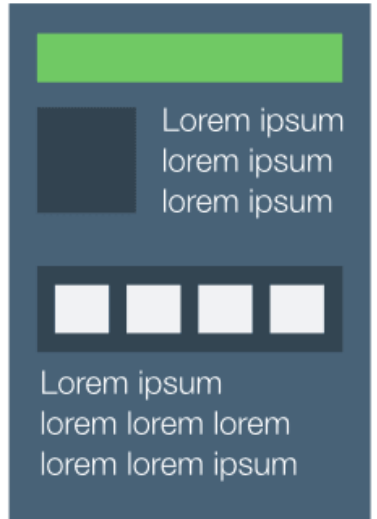
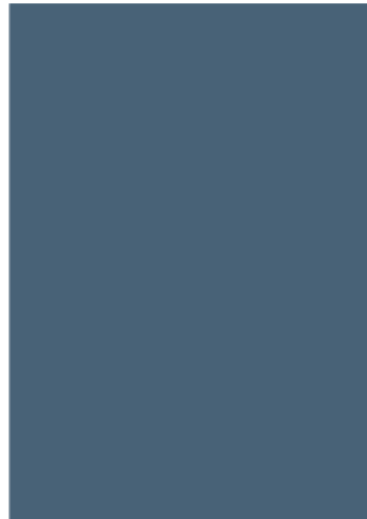
1.2s

1.5s

Optimized  
(progressive)  
rendering

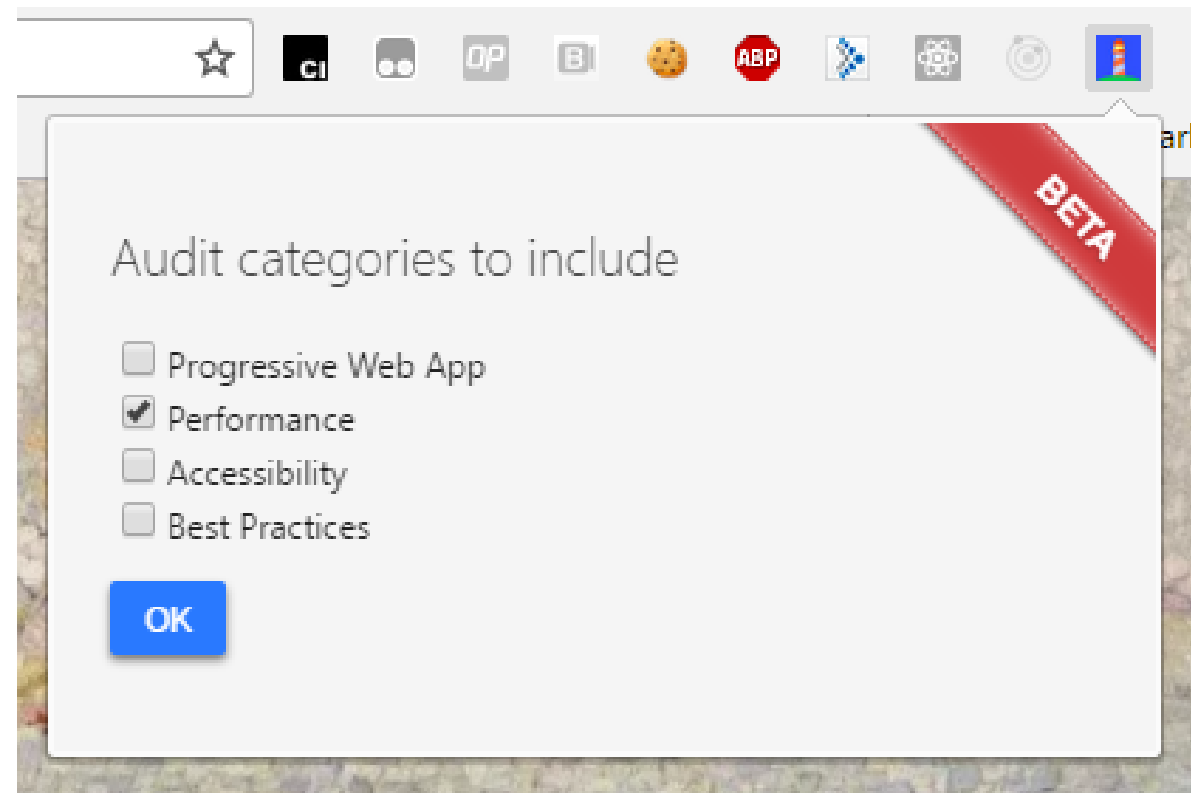
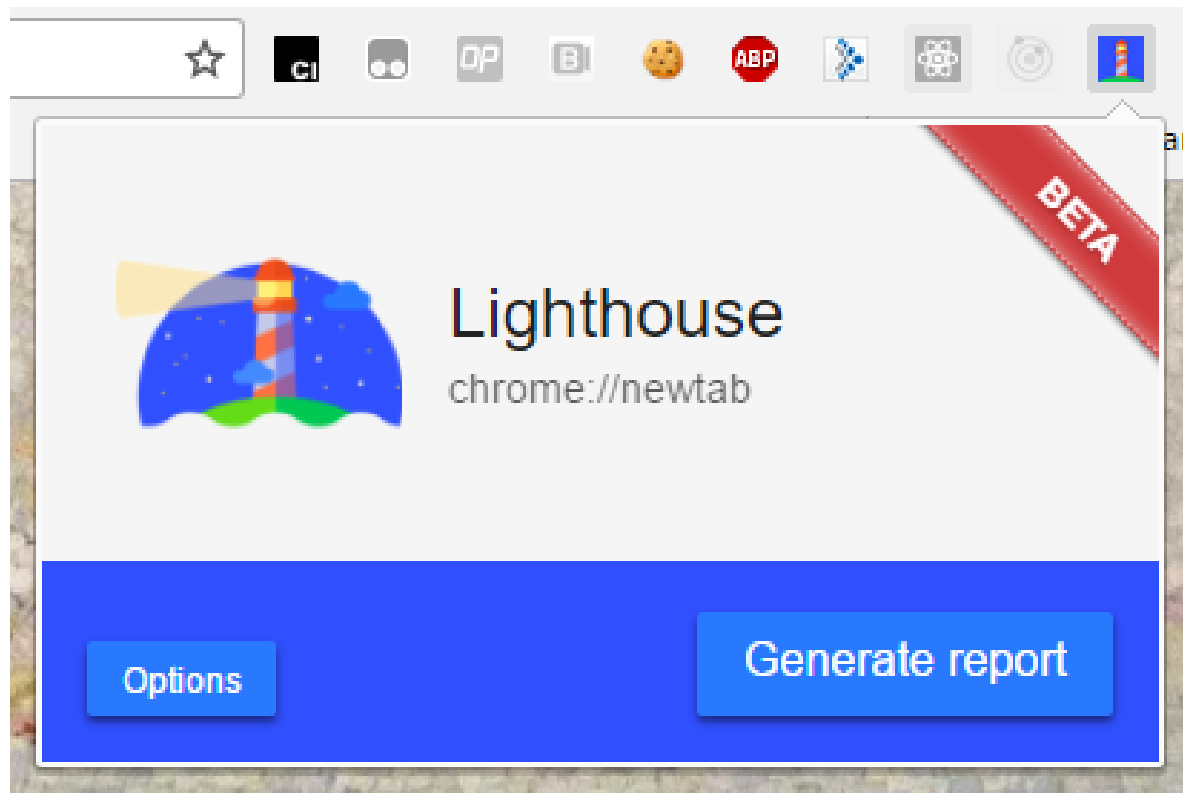


Unoptimized  
rendering



Tools





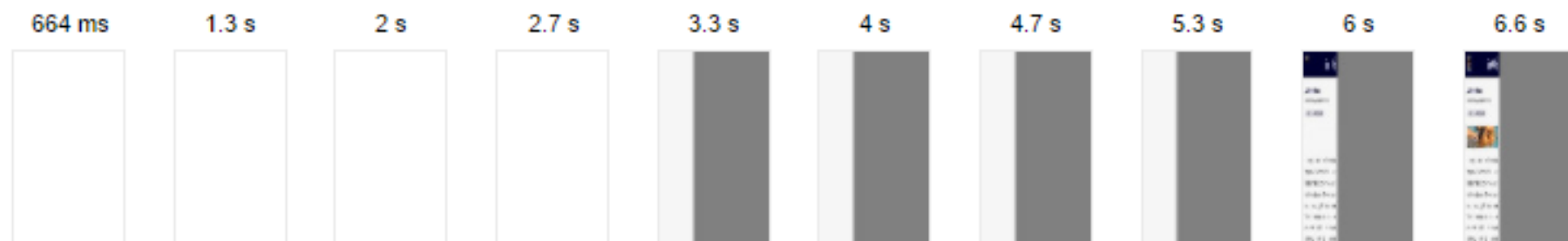


## Performance

These encapsulate your app's performance.

### Metrics

These metrics encapsulate your app's performance across a number of dimensions.



First meaningful paint: **5828.5ms**

First meaningful paint measures when the primary content of a page is visible. [Learn more.](#)

First Interactive (beta): **5,830ms**

The first point at which necessary scripts of the page have loaded and the CPU is idle enough to handle most user input.

Consistently Interactive (beta): **5,830ms**





The point at which most network resources have finished loading and the CPU is idle for a prolonged period.

**61** Perceptual Speed Index: 4540 (target: < 1,250)

**100** Estimated Input Latency: 16ms (target: < 50 ms)


## Opportunities


These are opportunities to speed up your application by optimizing the following resources.

Enable text compression		1,280 ms 231 KB	∨
Reduce render-blocking stylesheets		690 ms	∨
Properly size images		300 ms 54 KB	∨
Serve images as WebP		60 ms 11 KB	∨

## Diagnostics

More information about the performance of your application.

 Critical Request Chains: 5

 View 6 passed items



# Lighthouse

Version: 2.0.0

Results for: <http://soofka.pl/>

May 21, 2017, 10:29 PM GMT+2 • Runtime settings ▾



Progressive Web App



Performance



Accessibility



Best Practices

Progressive Web App 27

Performance 63

Accessibility 94

Best Practices 67



## Progressive Web App

These audits validate the aspects of a Progressive Web App, as specified by the baseline [PWA Checklist](#).

✘ Registers a Service Worker



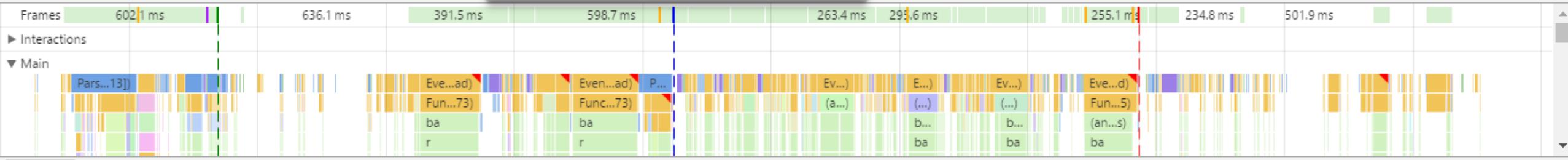
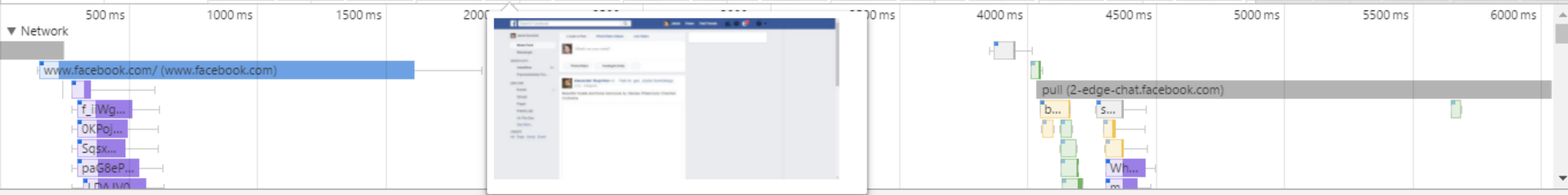
✘ Responds with a 200 when offline



Disable JavaScript Samples

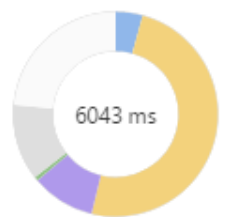
Enable advanced paint instrumentation (slow)

Network: Regular 4G (20ms, 4.0M)  
CPU: No throttling



Summary Bottom-Up Call Tree Event Log

Range: 0 - 6.04 s



- 257.9 ms Loading
- 2993.3 ms Scripting
- 606.3 ms Rendering
- 36.9 ms Painting
- 727.2 ms Other
- 1421.4 ms Idle

Network: Regular 4G (20ms, 4.0M)▼

CPU: No

**Disabled**  
No throttling

**Presets**

- Offline (0ms, 0kb/s, 0kb/s)
- GPRS (500ms, 50kb/s, 20kb/s)
- Regular 2G (300ms, 250kb/s, 50kb/s)
- Good 2G (150ms, 450kb/s, 150kb/s)
- Regular 3G (100ms, 750kb/s, 250kb/s)
- Good 3G (40ms, 1.5Mb/s, 750kb/s)
- Regular 4G (20ms, 4.0Mb/s, 3.0Mb/s)**
- DSL (5ms, 2.0Mb/s, 1.0Mb/s)
- WiFi (2ms, 30Mb/s, 15Mb/s)

**Custom**  
Add...

## Network Throttling Profiles

Add custom profile...

Profile Name	Download	Upload	Latency
Potato	10 optional	2 optional	1000 optional
Offline	0 kb/s	0 kb/s	0ms
GPRS	50 kb/s	20 kb/s	500ms
Regular 2G	250 kb/s	50 kb/s	300ms
Good 2G	450 kb/s	150 kb/s	150ms
Regular 3G	750 kb/s	250 kb/s	100ms
Good 3G	1.5 Mb/s	750 kb/s	40ms
Regular 4G	4.0 Mb/s	3.0 Mb/s	20ms
DSL	2.0 Mb/s	1.0 Mb/s	5ms
WiFi	30 Mb/s	15 Mb/s	2ms

# Other tools

- Google PageSpeed Insights
- Google TestMySite

Here are the scores for  
**facebook.com**



1K	May	21	20:30	index.html
437K	May	21	20:30	main.css
43K	May	21	20:30	main.css.gz
761K	May	21	20:30	main.js
176K	May	21	20:30	main.js.gz
11596K	May	21	20:49	stats.html
5023K	May	21	20:30	stats.json
209K	May	21	20:30	vendor.js
67K	May	21	20:30	vendor.js.gz

1. Why?

2. How to measure?

**3. How to improve?**

**a) Code manipulations**

b) Code splitting

c) Caching

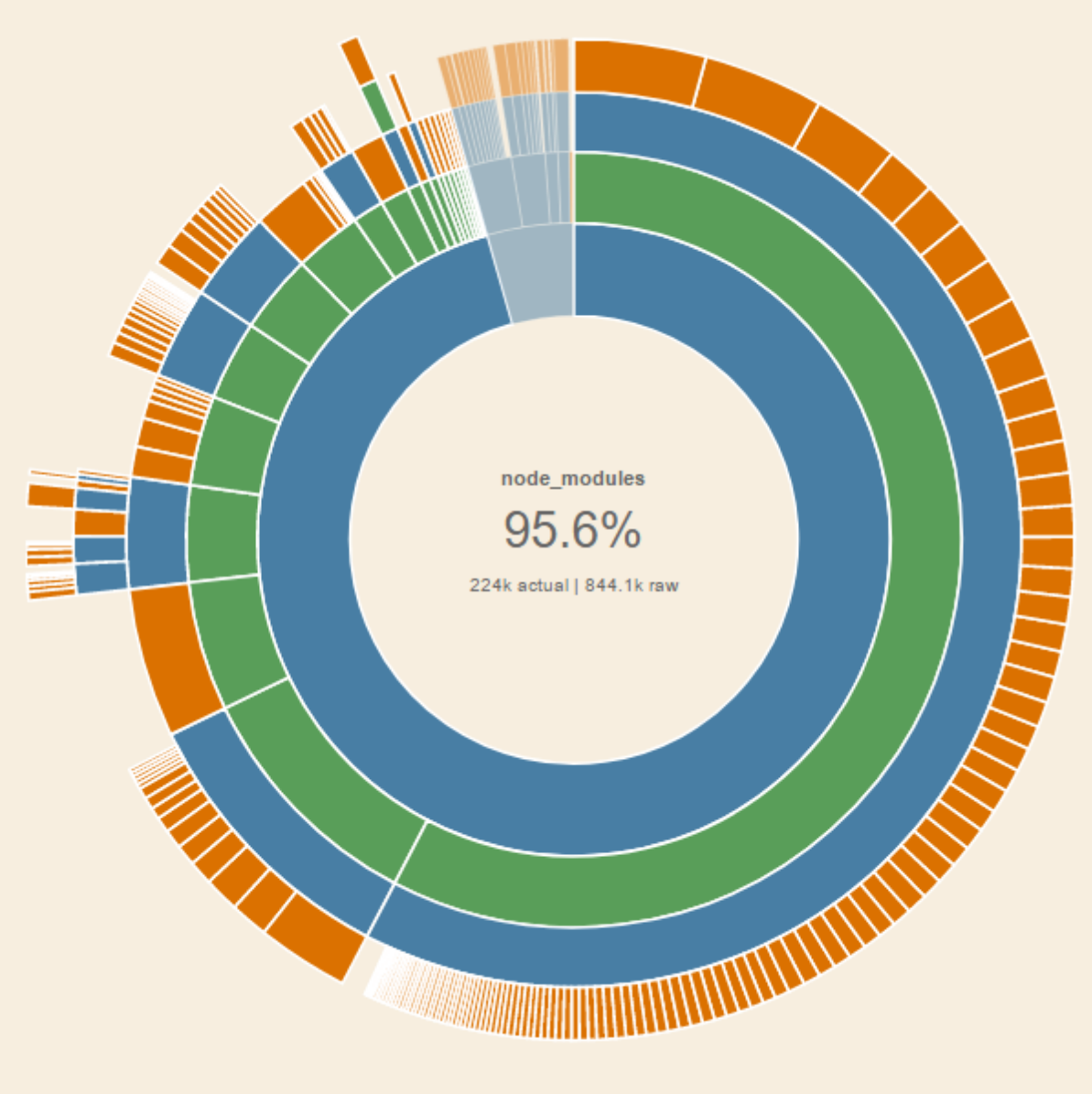
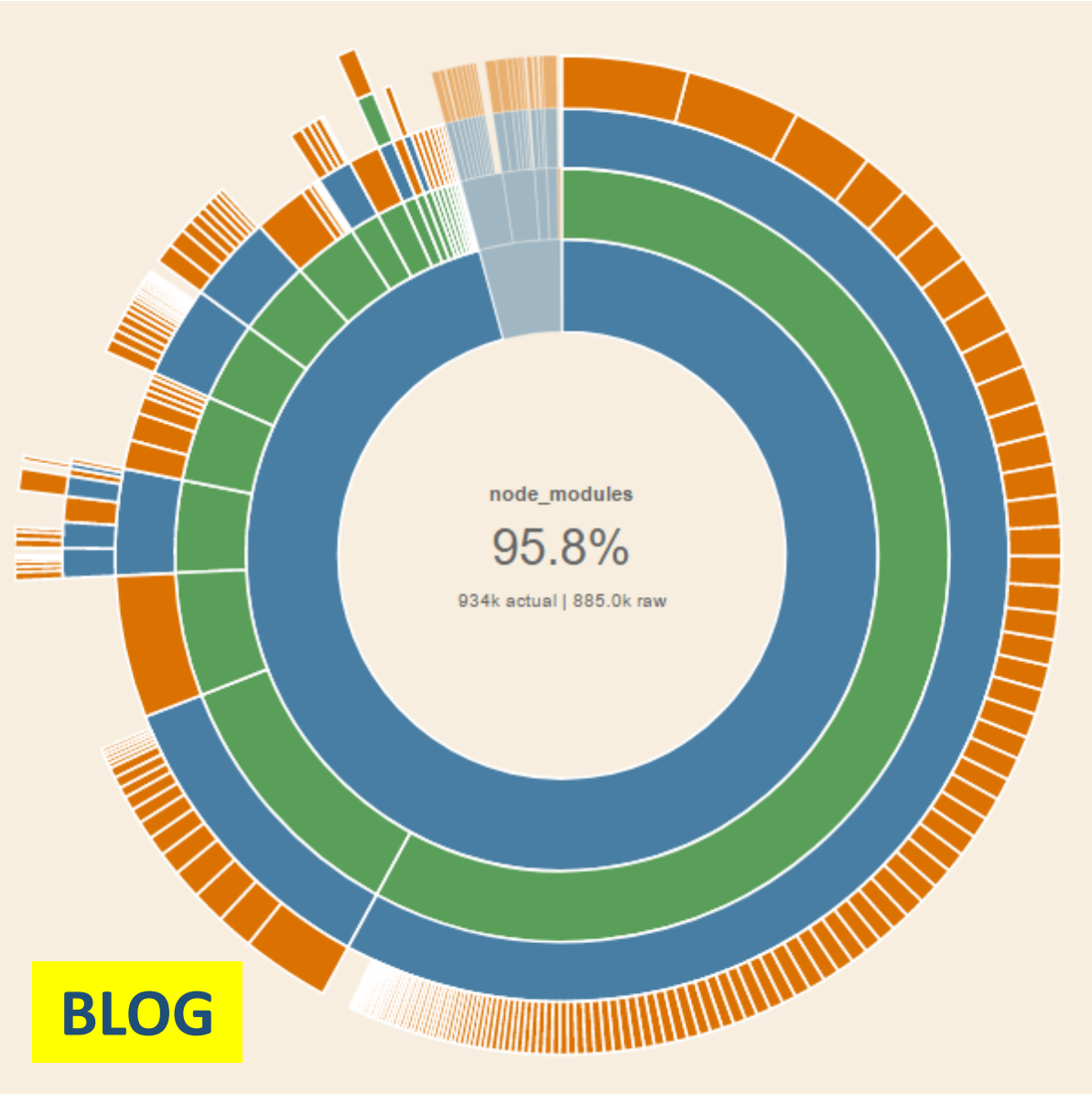


How to improve JS  
performance?

Code manipulations

Original: **975 KB**

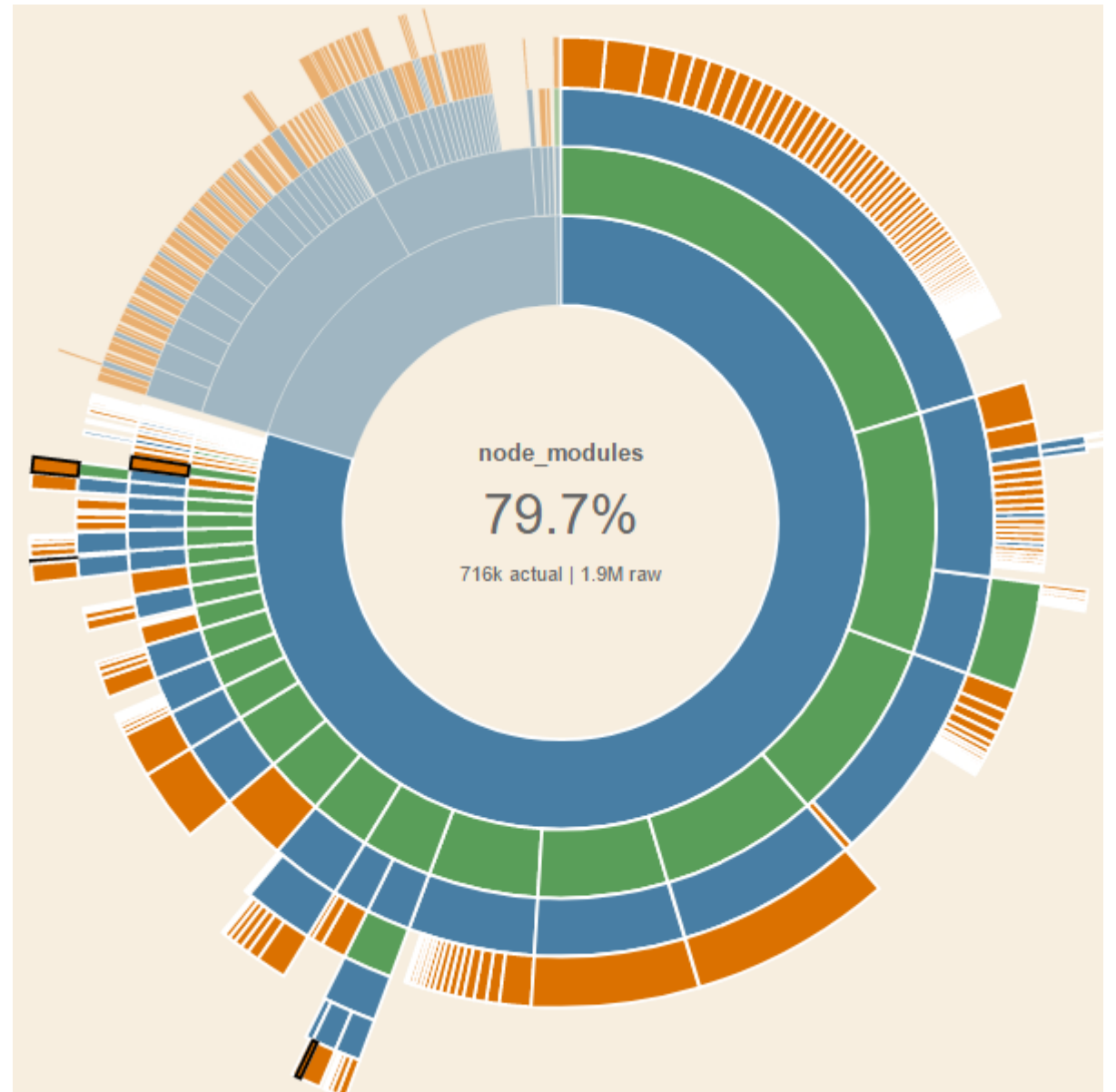
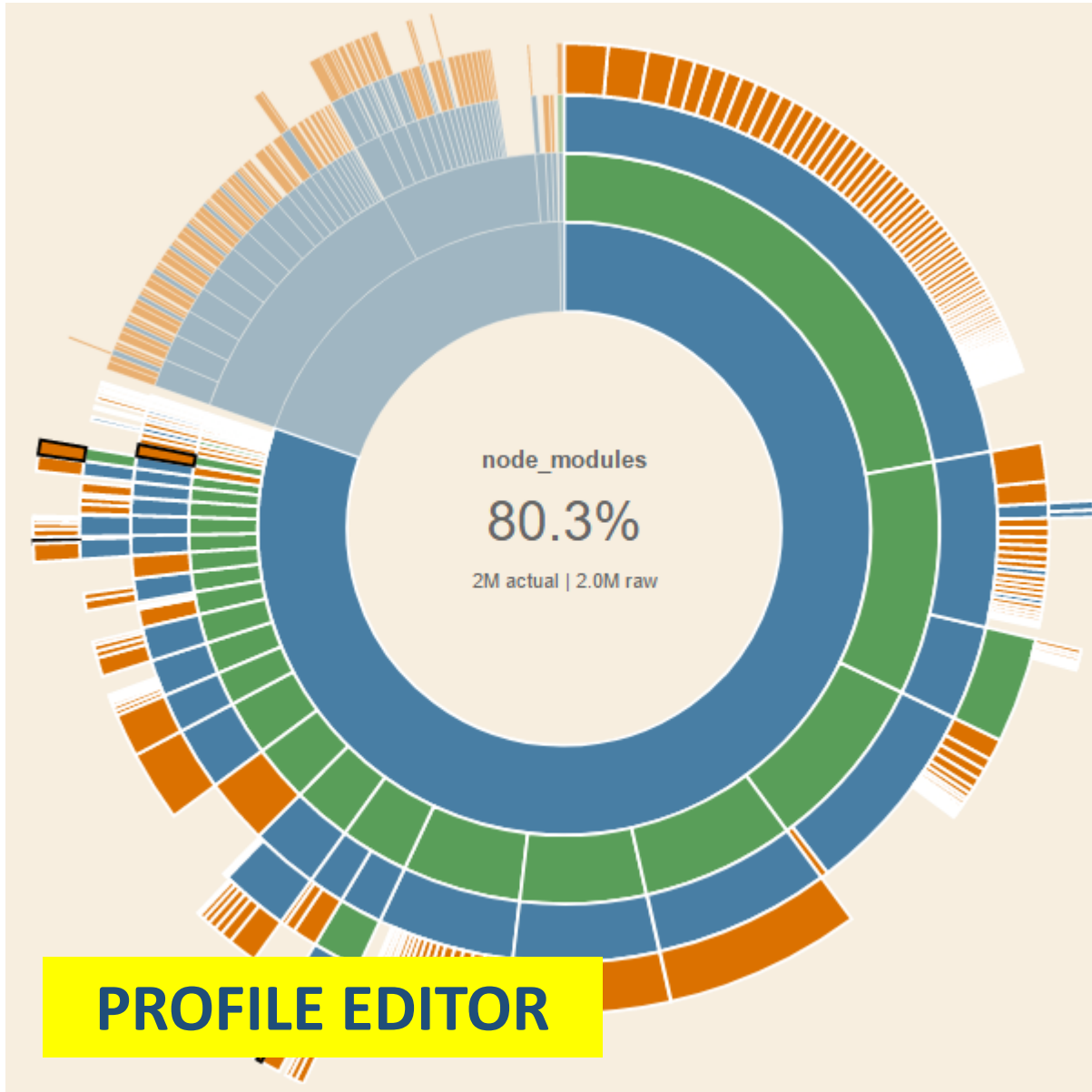
Minified: **235 KB -76%**





Original: **2649 KB**

Minified: **900 KB** **-66%**



# Precompression

```
module.exports = {
  plugins: [
    new CompressionPlugin({
      asset: "[path].gz[query]",
      algorithm: "gzip",
      test: /\. (js|html)$/,
      threshold: 10240,
      minRatio: 0.8
    })
  ]
}
```

```
rollup({
  entry: 'src/index.js',
  plugins: [
    zopfli()
  ]
}).then(/* ... */)
```

# Measurements

- Blog:

size: **235 KB** -> **68 KB** (-71%)

first meaningful paint: **4036 ms** -> **3846 ms** (-190 ms)

- Profile editor:

**900 KB** -> **221 KB** (-75%)

first meaningful paint: **5519 ms** -> **5404 ms** (-115 ms)

# Prepack



## Input

```
(function () {  
  function hello() { return 'hello'; }  
  function world() { return 'world'; }  
  global.s = hello() + ' ' + world();  
})();
```

## Output

```
(function () {  
  s = "hello world";  
})();
```

## Elimination of abstraction tax

Input

```
(function () {  
  var self = this;  
  ['A', 'B', 42].forEach(function(x) {  
    var name = '_' + x.toString()[0].toLowerCase();  
    var y = parseInt(x);  
    self[name] = y ? y : x;  
  });  
})();
```

Output

```
(function () {  
  _a = "A";  
  _b = "B";  
  _4 = 42;  
})();
```

## Environment Interactions and Branching

Input

```
(function(){  
  function fib(x) { return x <= 1 ? x : fib(x - 1) + fib(x - 2); }  
  let x = Date.now();  
  if (x === 0) x = fib(10);  
  global.result = x;  
})();
```

Output

```
(function () {  
  var _0 = Date.now();  
  if (typeof _0 !== "number") {  
    throw new Error("Prepack model invariant violation");  
  }  
  result = _0 === 0 ? 55 : _0;  
})();
```

# Prepack

**A tool for making JavaScript code run faster.**

\*Prepack is still in an early development stage and not ready for production use just yet. Please try it out, give feedback, and help fix bugs.



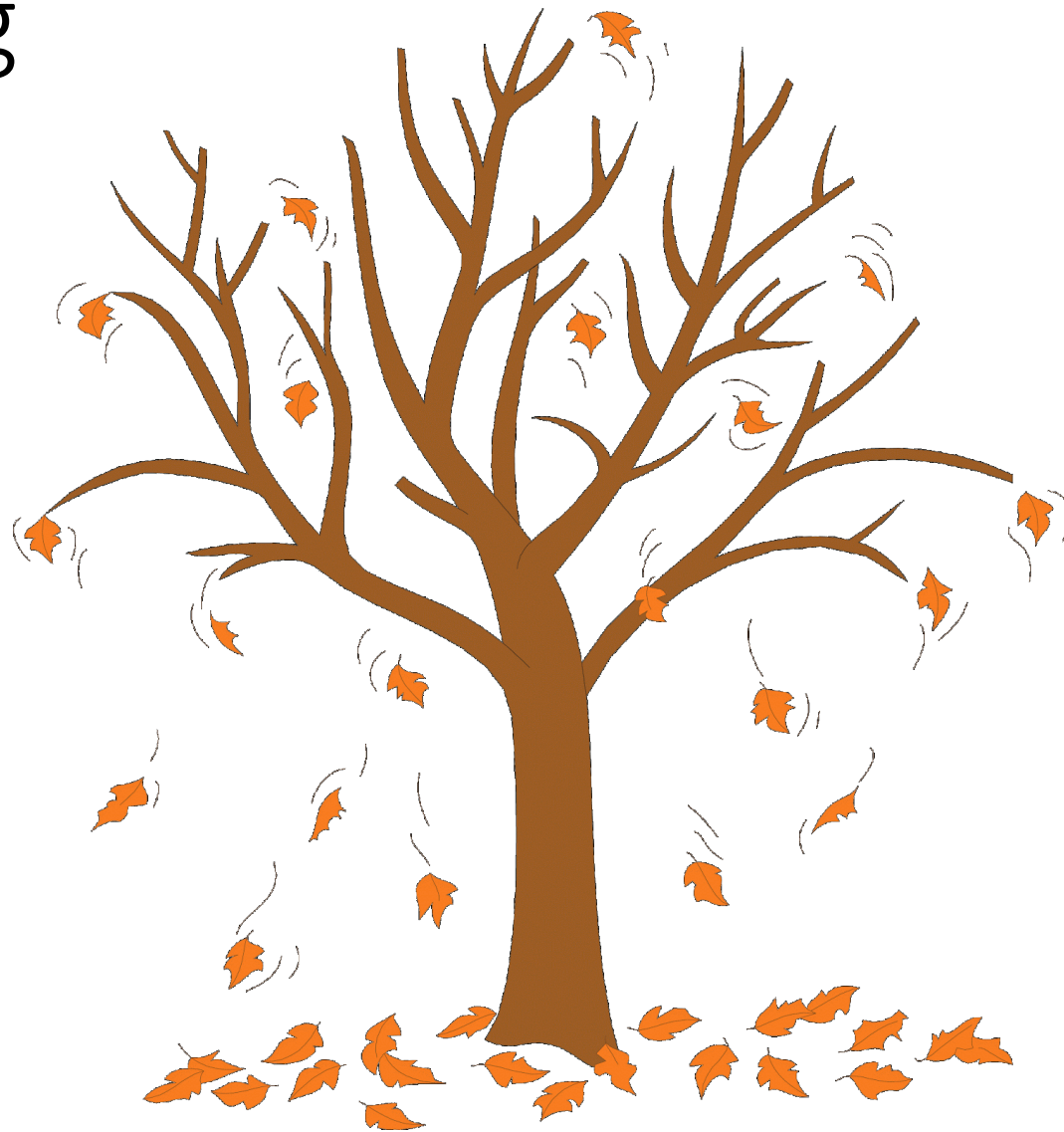
**cblappert** published 2 weeks ago

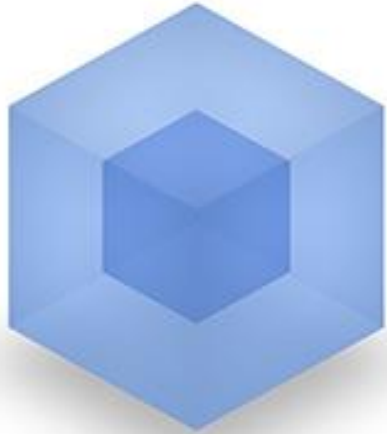
---

**0.2.2** is the latest of 6 releases

---

# Treeshaking





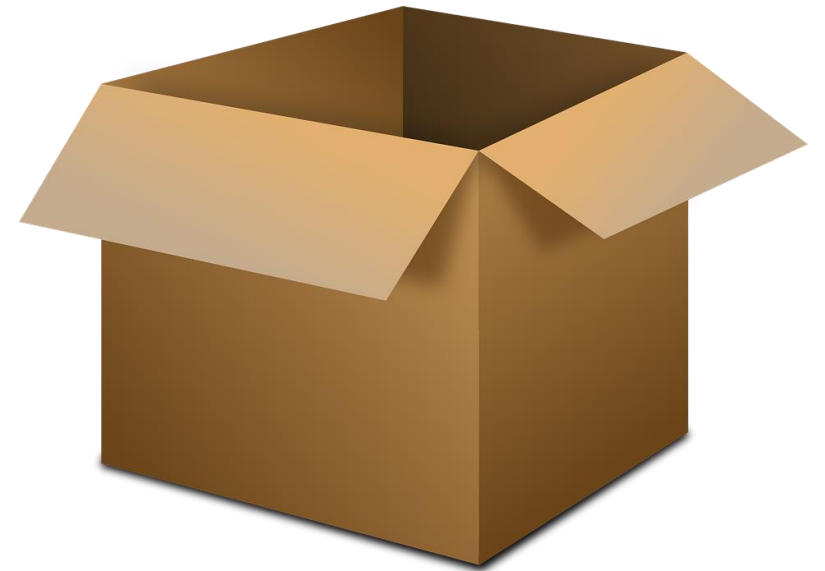
webpack



rollup.js

.babelrc

```
{  
  "presets": [ ["es2015", { "modules": false }], "react" ]  
}
```





**-28%**

**-7 KB**

# Code manipulations summary

- Minification
- Uglification
- Precompression
- Prepacking
- Treeshaking



## BLOG

size: **975 KB** -> **228 KB** (-77%)

first meaningful paint: **4661 ms** -> **3846 ms** (-18%)

time to interactive: **5560 ms** -> **4880 ms** (-12%)

Lighthouse score: **67** -> **74**

## PROFILE EDITOR

size: **2649 KB** -> **900 KB** (-66%)

first meaningful paint: **12129 ms** -> **5519 ms** (-55%)

time to interactive: **13760 ms** -> **7310 ms** (-49%)

Lighthouse score: **29** -> **57**

1. Why?

2. How to measure?

**3. How to improve?**

a) Code manipulations

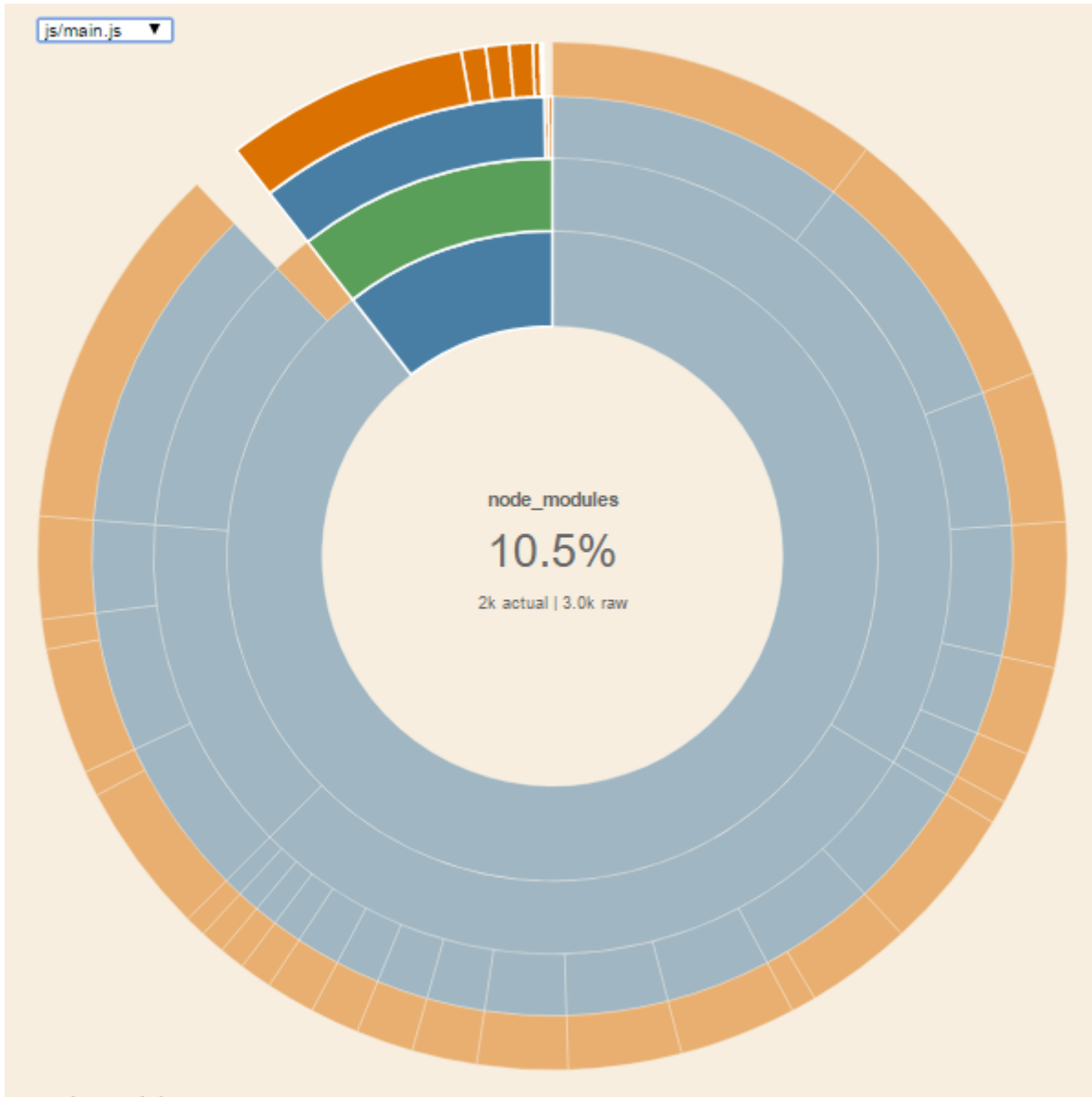
**b) Code splitting**

c) Caching

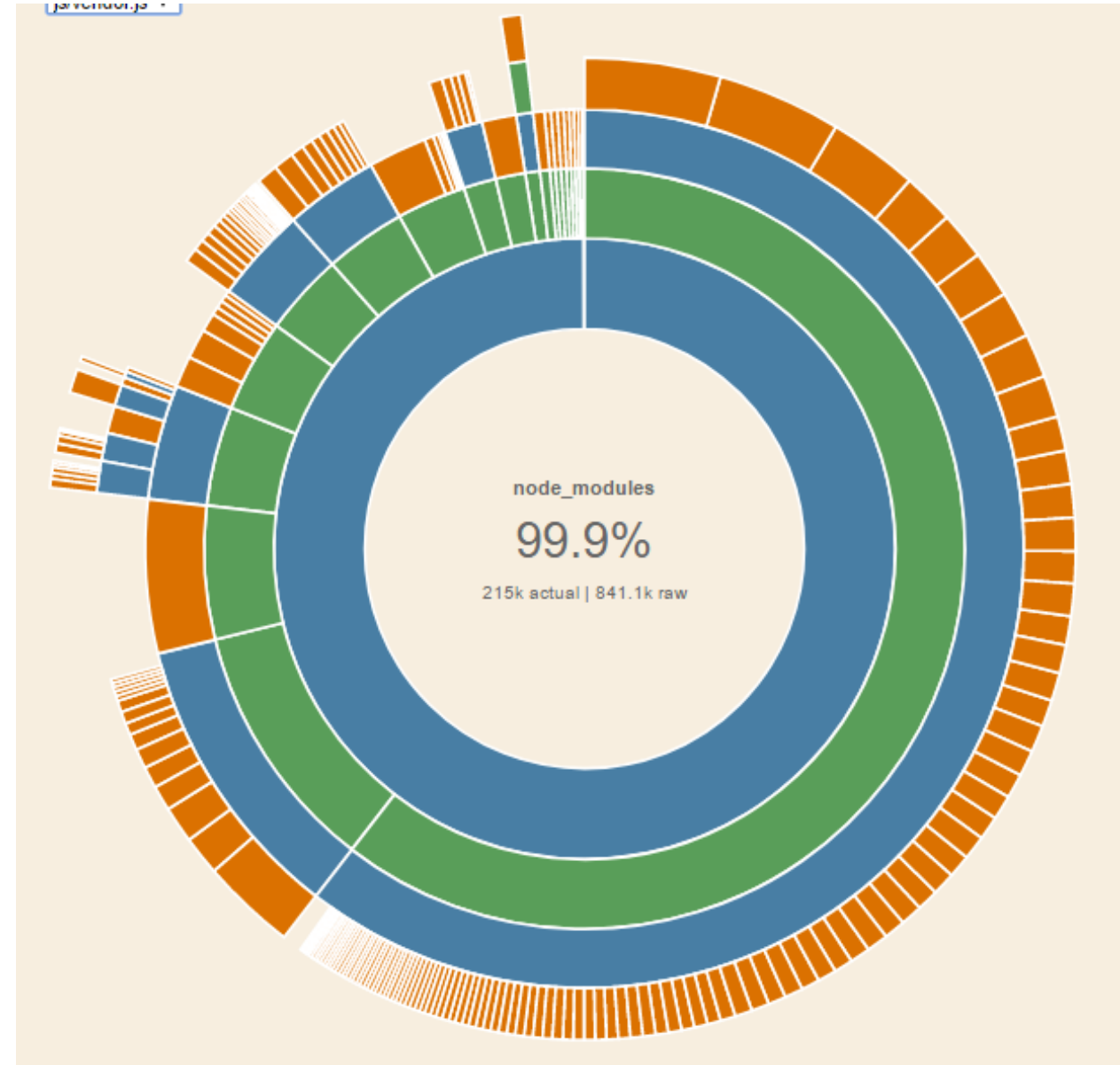


Code splitting

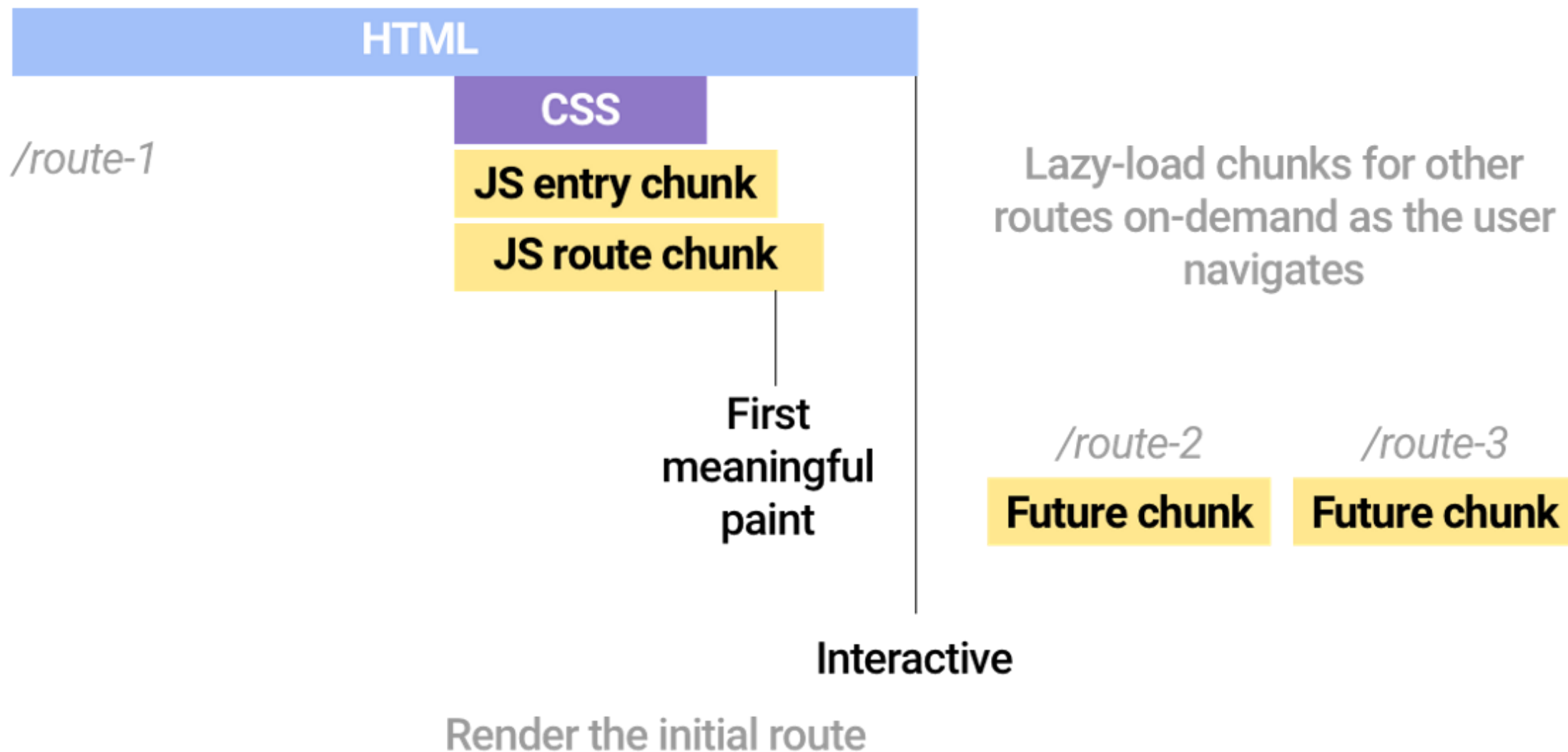
# main.js (16 KB)



# vendor.js (215 KB)



Push the minimal functional code for a route





- `require.ensure(...)`
- `System.import(...)`

```
import App from '../containers/App';

function errorLoading(err) {
  console.error('Lazy-loading failed', err);
}

function loadRoute(cb) {
  return (module) => cb(null, module.default);
}

export default {
  component: App,
  childRoutes: [
    // ...
    {
      path: 'booktour',
      getComponent(location, cb) {
        System.import('../pages/BookTour')
          .then(loadRoute(cb))
          .catch(errorLoading);
      }
    }
  ]
};
```

Asset	Size	Chunks	Chunk Names
0.Item.js.map	47.4 kB	0	[emitted] Item
0.Item.js	8.96 kB	0	[emitted] Item
2.PermalinkedComment.js	7.53 kB	2	[emitted] PermalinkedComment
3.UserProfile.js	2.28 kB	3	[emitted] UserProfile
4.NotFound.js	346 bytes	4	[emitted] NotFound
app.js	39.6 kB	5	[emitted] app
vendor.js	350 kB	1	[emitted] vendor
vendor.js.map	2.61 MB	1	[emitted] vendor
2.PermalinkedComment.js.map	42.7 kB	2	[emitted] PermalinkedComment
3.UserProfile.js.map	10.8 kB	3	[emitted] UserProfile
4.NotFound.js.map	1.56 kB	4	[emitted] NotFound
app.js.map	225 kB	5	[emitted] app

Preload chunks for other routes so they're available before the user navigates to them

*/route-1*



**chunk-2**

*/route-2*

**chunk-3**

*/route-3*

**chunk-4**

*/route-4*

```
<head>
```

```
<link rel="preload" as="script" href="chunk-2.js">
```

```
<link rel="preload" as="script" href="chunk-3.js">
```

```
<link rel="preload" as="script" href="chunk-4.js">
```

```
</head>
```

# Code splitting summary

- Main and vendor
- Route based chunking
- Lazy loading



## BLOG

size: **228 KB** -> **234 KB** (+3%)

first meaningful paint: **3846 ms** -> **3424 ms** (-11%)

time to interactive: **4880 ms** -> **4410 ms** (-10%)

Lighthouse score: **74** -> **79**

## PROFILE EDITOR

size: **900 KB** -> **923 KB** (+3%)

first meaningful paint: **5519 ms** -> **5170 ms** (-6%)

time to interactive: **7310 ms** -> **7100 ms** (-3%)

Lighthouse score: **57** -> **59**

1. Why?

2. How to measure?

**3. How to improve?**

a) Code manipulations

b) Code splitting

**c) Caching**

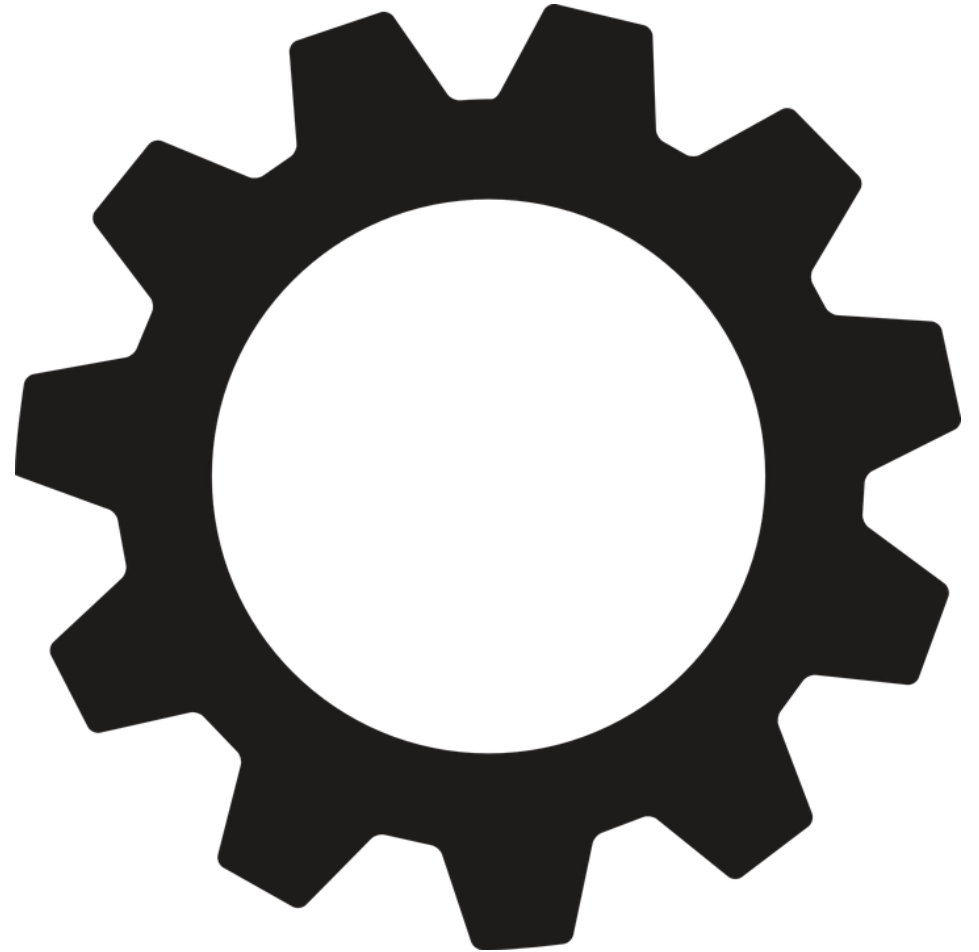


Caching



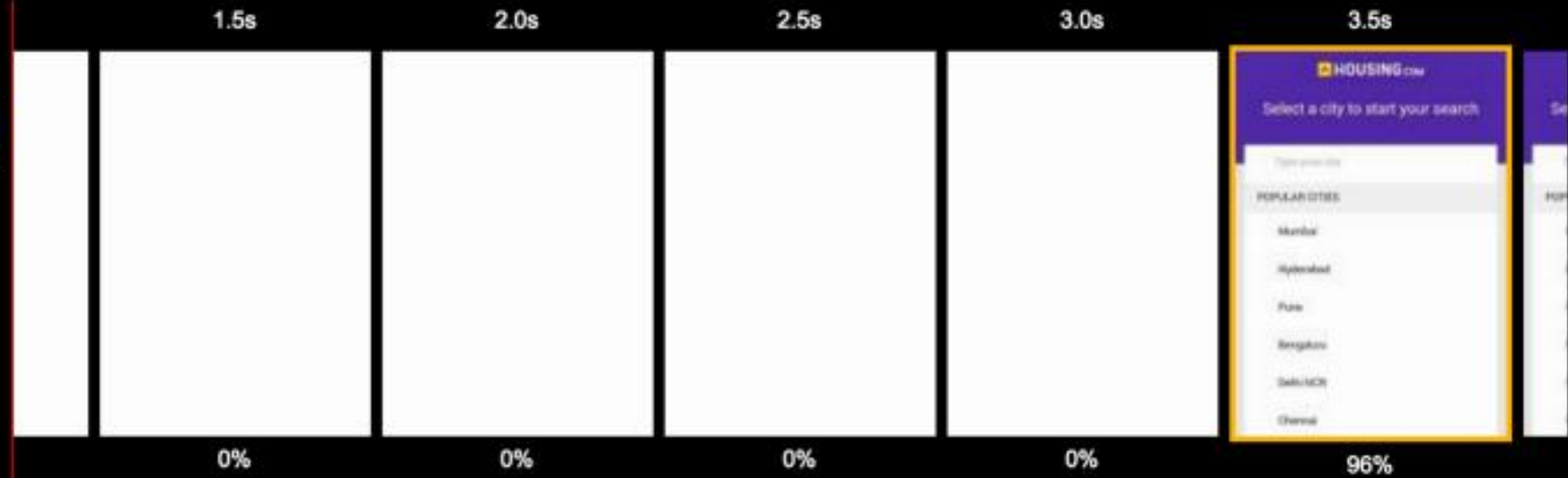
# Service worker

- JS background worker
- Programmable proxy
- Control request-by-request
- Make app work offline



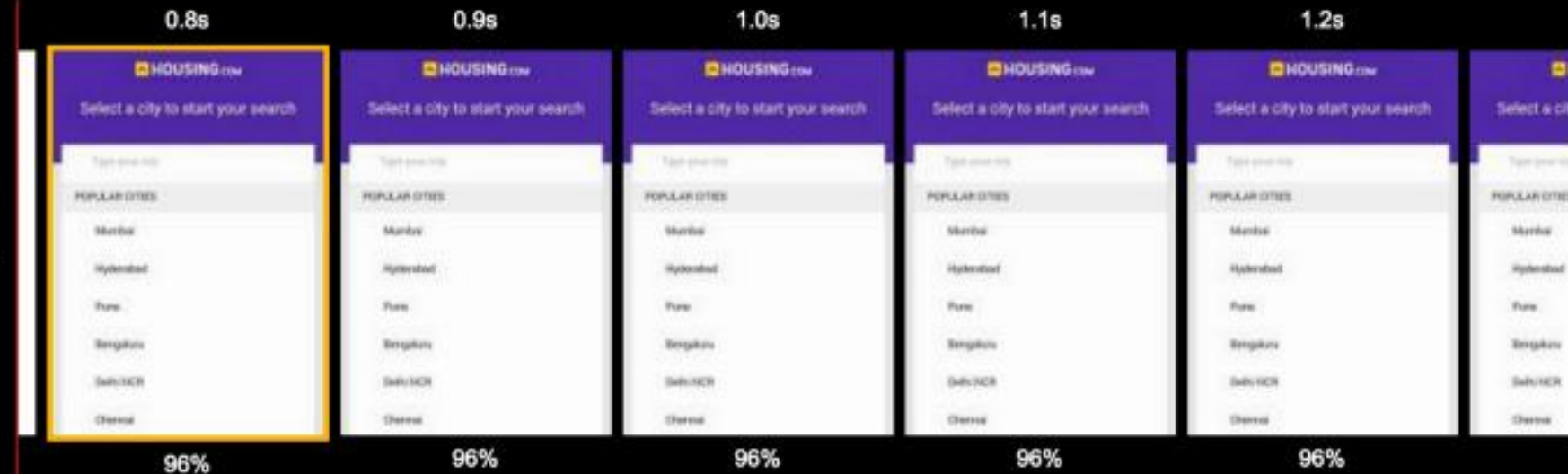
## First visit

Moto G / 3G



## Repeat visit

Near instant loading thanks to offline caching with Service Worker



# is SERVICEWORKER ready?

## ServiceWorker enthusiasm

The first thing any implementation needs.



**Chrome:** Shipped.

**Firefox:** Shipped.

**Samsung Internet:** Shipped. Based on Chromium 44.2403 with some [additions and changes](#). (See "Service Worker" section.)

**Safari:** [Under consideration](#), Brief positive signals in [five year plan](#).

**Edge:** [In development](#).

Support does not include iOS versions of third-party browsers on that platform (see [Safari support](#)).

## `navigator.serviceWorker`

Namespace for page-side ServiceWorker API. [Spec](#). [Test](#).



# Registration

```
// Check for browser support of service worker
if ('serviceWorker' in navigator) {

  navigator.serviceWorker.register('service-worker.js')
  .then(function(registration) {
    // Successful registration
    console.log('Hooray. Registration successful, scope is:',
registration.scope);
  }).catch(function(err) {
    // Failed registration, service worker won't be installed
    console.log('Whoops. Service worker registration failed, error:',
error);
  });
}
```

```
navigator.serviceWorker.register('service-worker.js', {
  scope: '/app/'
});
```

# Installation and activation

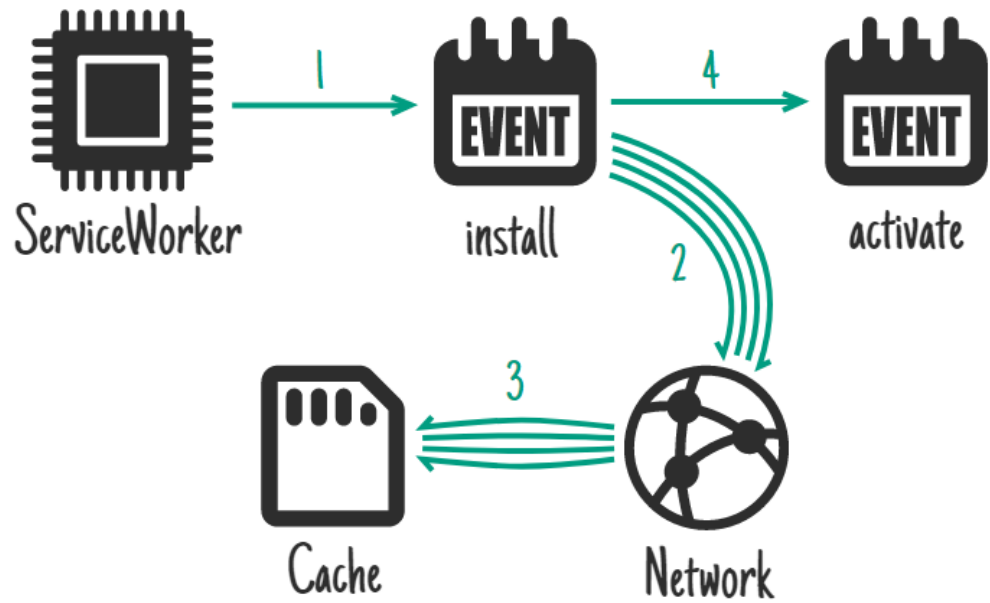
```
var CACHE_NAME = 'my-pwa-cache-v1';
var urlsToCache = [
  '/',
  '/styles/styles.css',
  '/script/webpack-bundle.js'
];

self.addEventListener('install', function(event) {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(function(cache) {
        // Open a cache and cache our files
        return cache.addAll(urlsToCache);
      })
  );
});
```

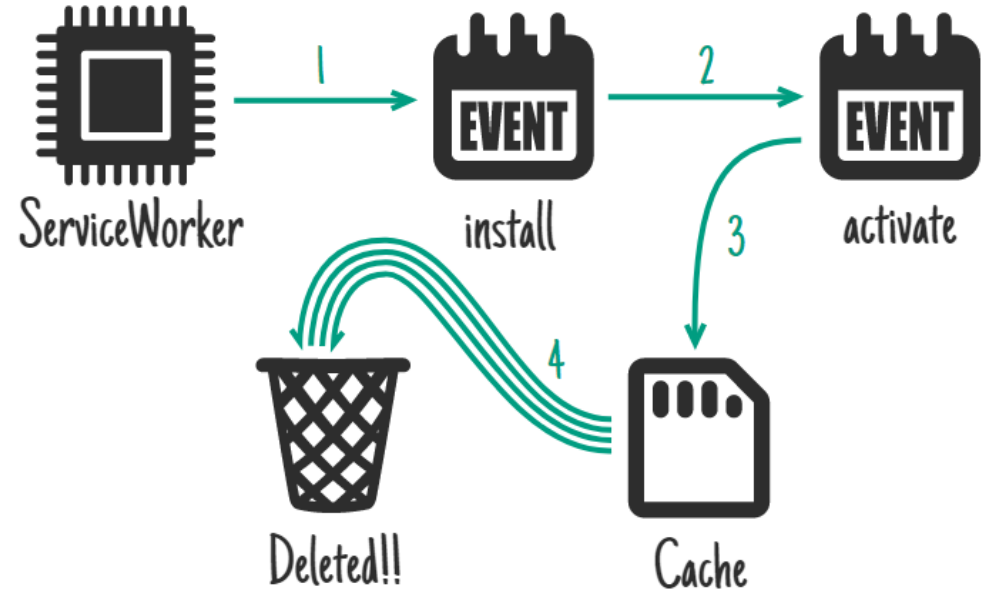
# Intercepting and caching requests

```
self.addEventListener('fetch', function(event) {  
  console.log(event.request.url);  
  event.respondWith(  
    caches.match(event.request).then(function(response) {  
      return response || fetch(event.request);  
    })  
  );  
});
```

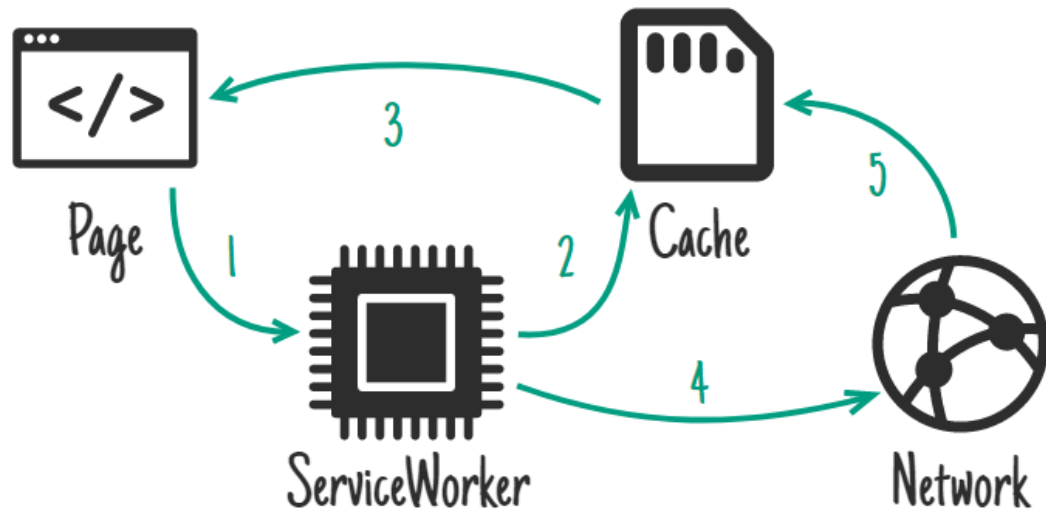
On install - as a dependency



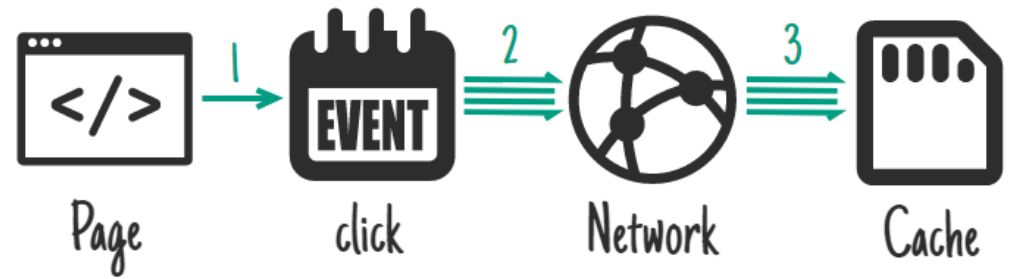
On activate



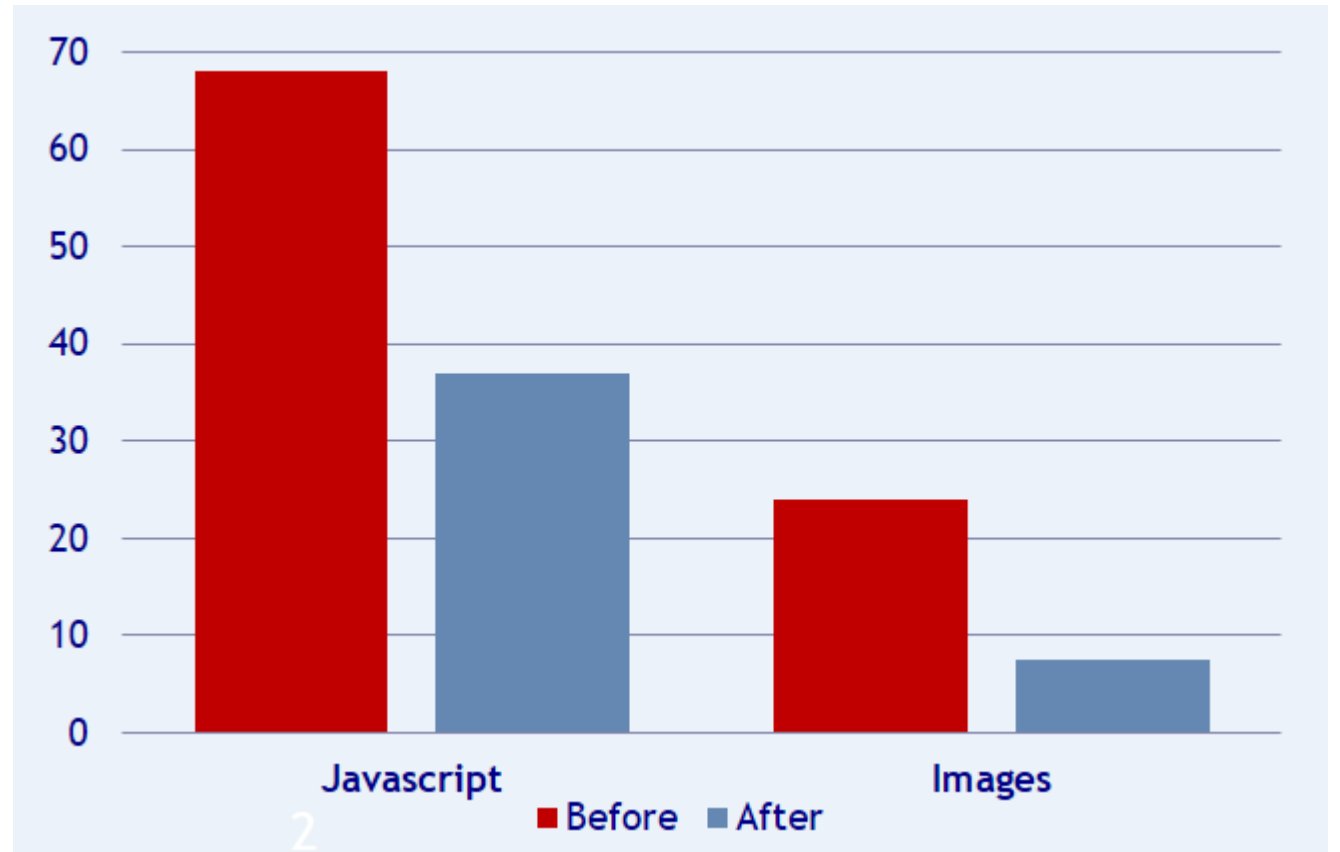
Stale-while-revalidate



On user interaction



**-30%**





# Tools

- Google Chrome Developer Tools
- Lighthouse

The screenshot displays the Google Chrome Developer Tools interface, specifically the Service Workers panel. The top navigation bar includes tabs for Elements, Console, Sources, Network, Performance, Memory, Application, Security, Audits, EditThisCookie, and Redux. The left sidebar shows the Application panel with sub-sections: Manifest, Service Workers (selected), and Clear storage. Below this are Storage (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies), Cache (Cache Storage, Application Cache), and Frames (top).

The main content area is titled "Service Workers" and contains two entries:

- https://jakearchibald.com/**
  - Source: [sw.js](#)
  - Received 5/22/2017, 9:52:29 PM
  - Status: ● #179 activated and is running [stop](#)
  - Clients: <https://jakearchibald.com/2014/offline-cookbook/> [focus](#)
- https://www.youtube.com/**
  - Source: [sw.js](#)
  - Received 5/9/2017, 7:47:15 PM
  - Status: ● #43 activated and is stopped [start](#)  
○ #182 installing  
1/1/1970, 1:00:00 AM
  - Clients: <https://www.youtube.com/watch?v=-fGoW73hLLg> [focus](#)  
<https://jakearchibald.com/2014/offline-cookbook/> [focus](#)



# Other features


- Push API

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			1 2 49						
			2 56			9.3		4.4	
	14	2 52	2 57	3 10		10.2		4.4.4	
11	15	2 53	2 58	3 10.1	2 44	10.3	all	56	57

- Background Sync

**Background sync**

Deferring tasks until the user has connectivity. [Spec.](#) [Test.](#)



**Firefox:** [Bug 1217544](#)

**Edge:** In development

# PRPL architecture

- **Push** critical resources for the initial URL route.
- **Render** initial route.
- **Pre-cache** remaining routes.
- **Lazy-load** and create remaining routes on demand.

# Caching summary

- Service worker fundamentals
- Service worker lifetime
- Is service worker ready?
- Background sync and Push API
- PRPL architecture



