Style Recalculation Secrets They Don't Want You To Know

Patrick Brosset – CSS Day – June 2023

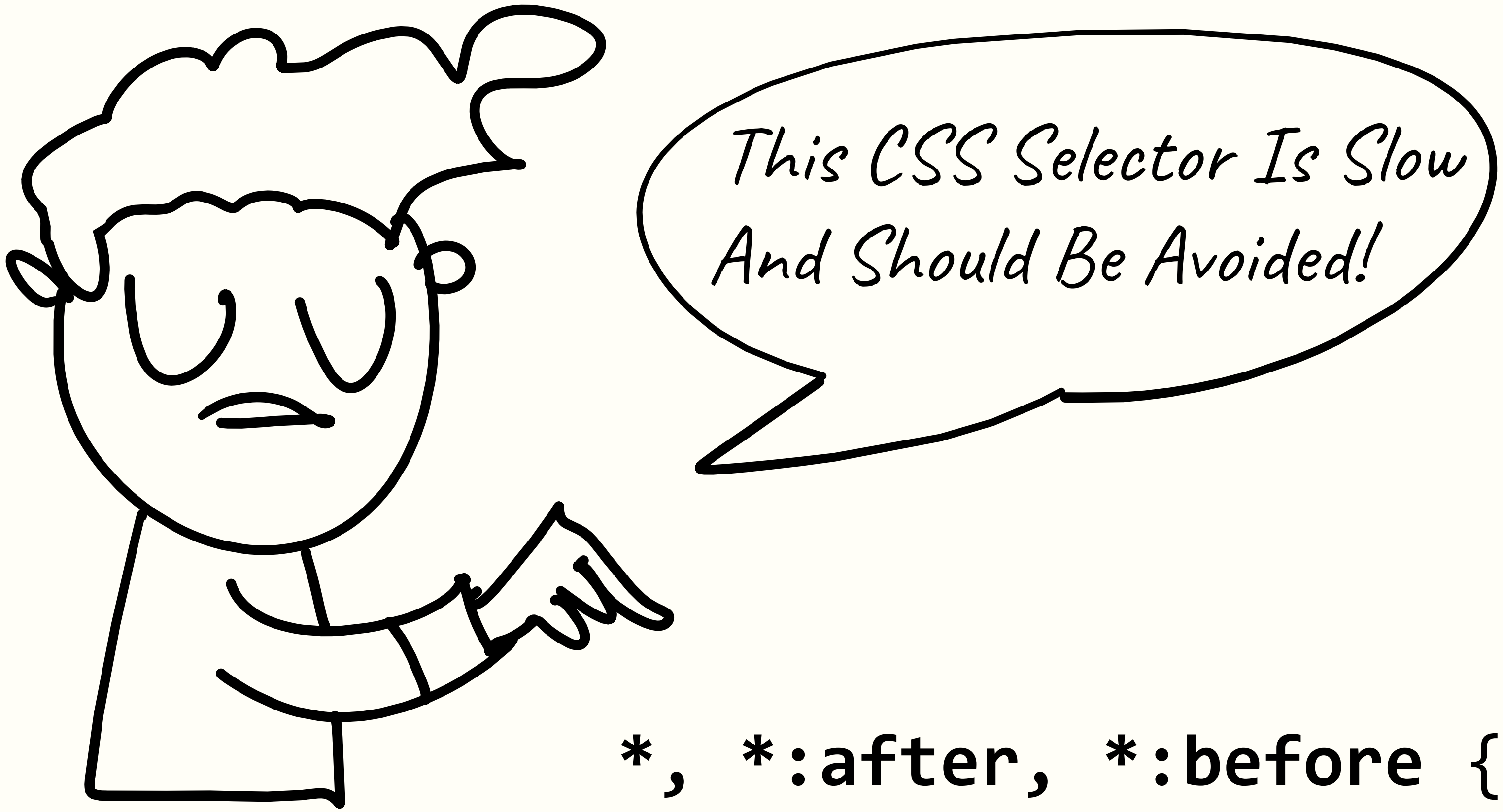
Chapter 1

CSS Selector performance doesn’t matter

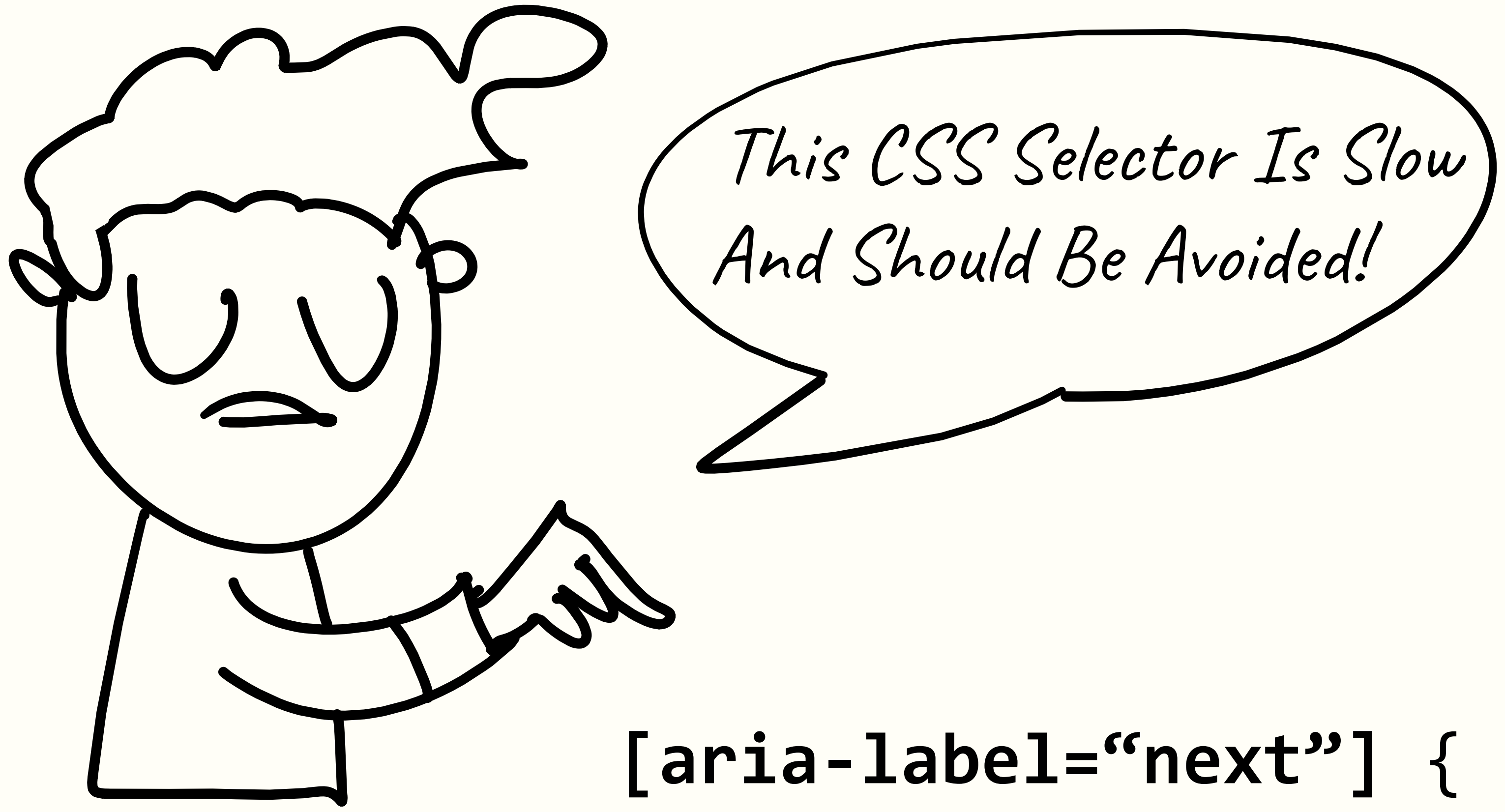
Chapter 1

CSS Selector performance doesn’t matter

… in most cases anyway



```
*, *:after, *:before {
   box-sizing: border-box;
}
```



```
[aria-label=“next”] {
    ...
}
```

Why do those rules exist?

- Historical reasons.
- People go from “don’t use X when Y” to “never use X”.
- We love rules!

Don’t optimize your CSS selectors just because someone said so.

– Me, just now

Selector performance is not something to optimize for […] We micro-manage our work for gains that aren’t noticeable.

– Jens Oliver Meiert, 6 years ago

CSS selectors are FAST. Do not spend time optimizing them.

– Paul Irish, 10 years ago

For most websites, optimizing CSS selectors won’t be worth the cost.

– Steve Souders, 12 years ago

What will we be talking about today?

How browsers **invalidate and recalculate styles** when changes occur on a webpage

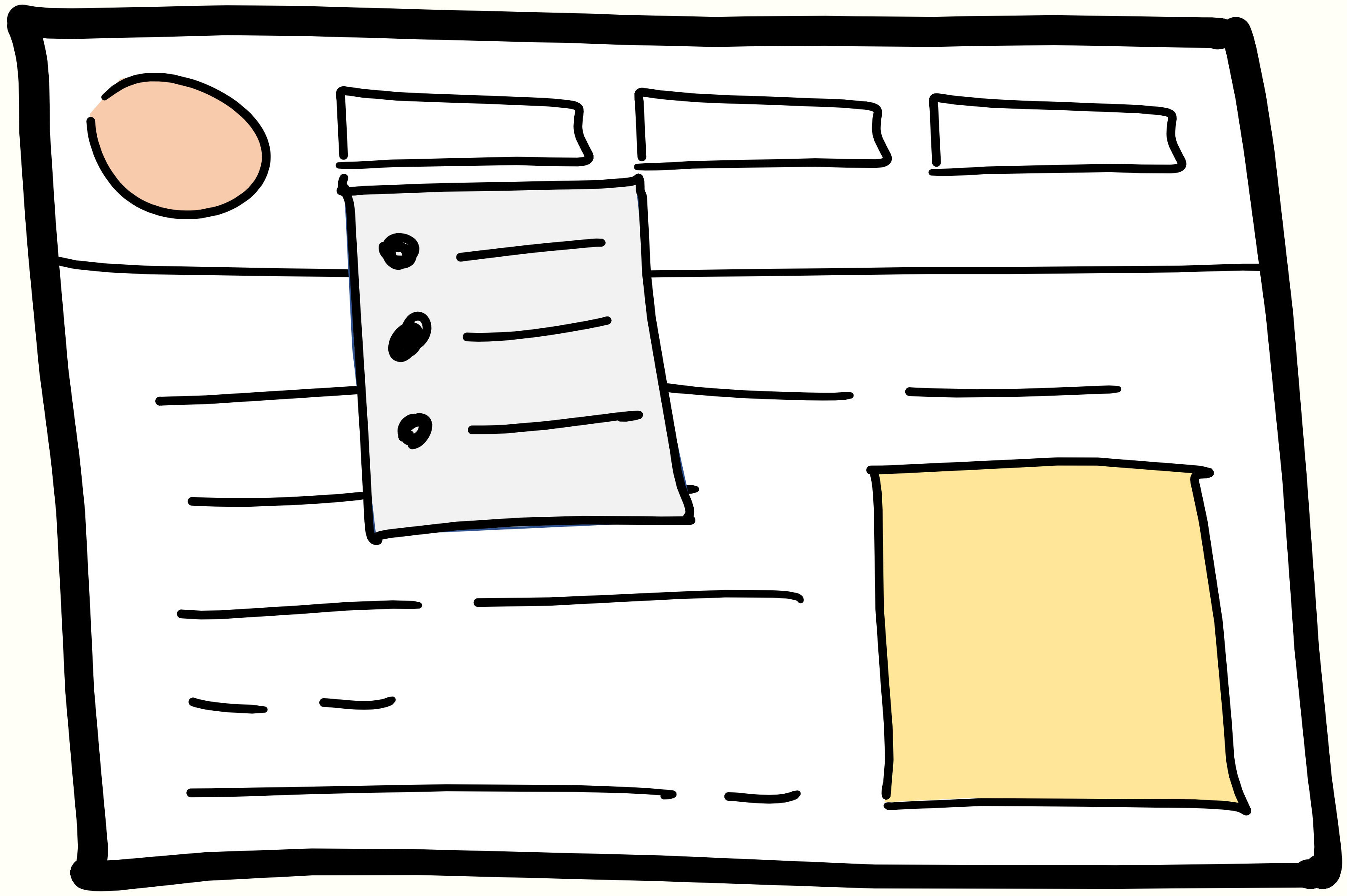
How **browsers** work

About **performance**

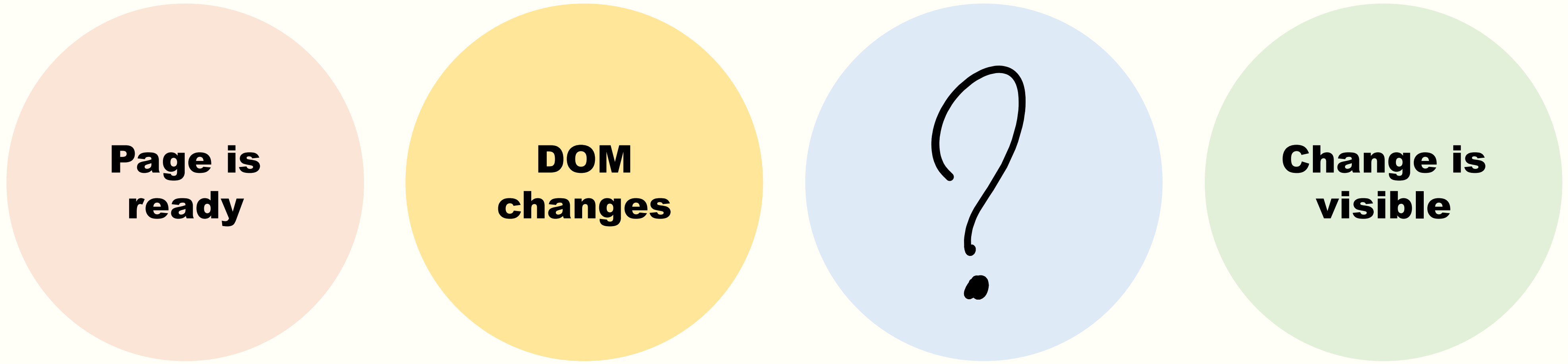
Chapter 2

Information overload

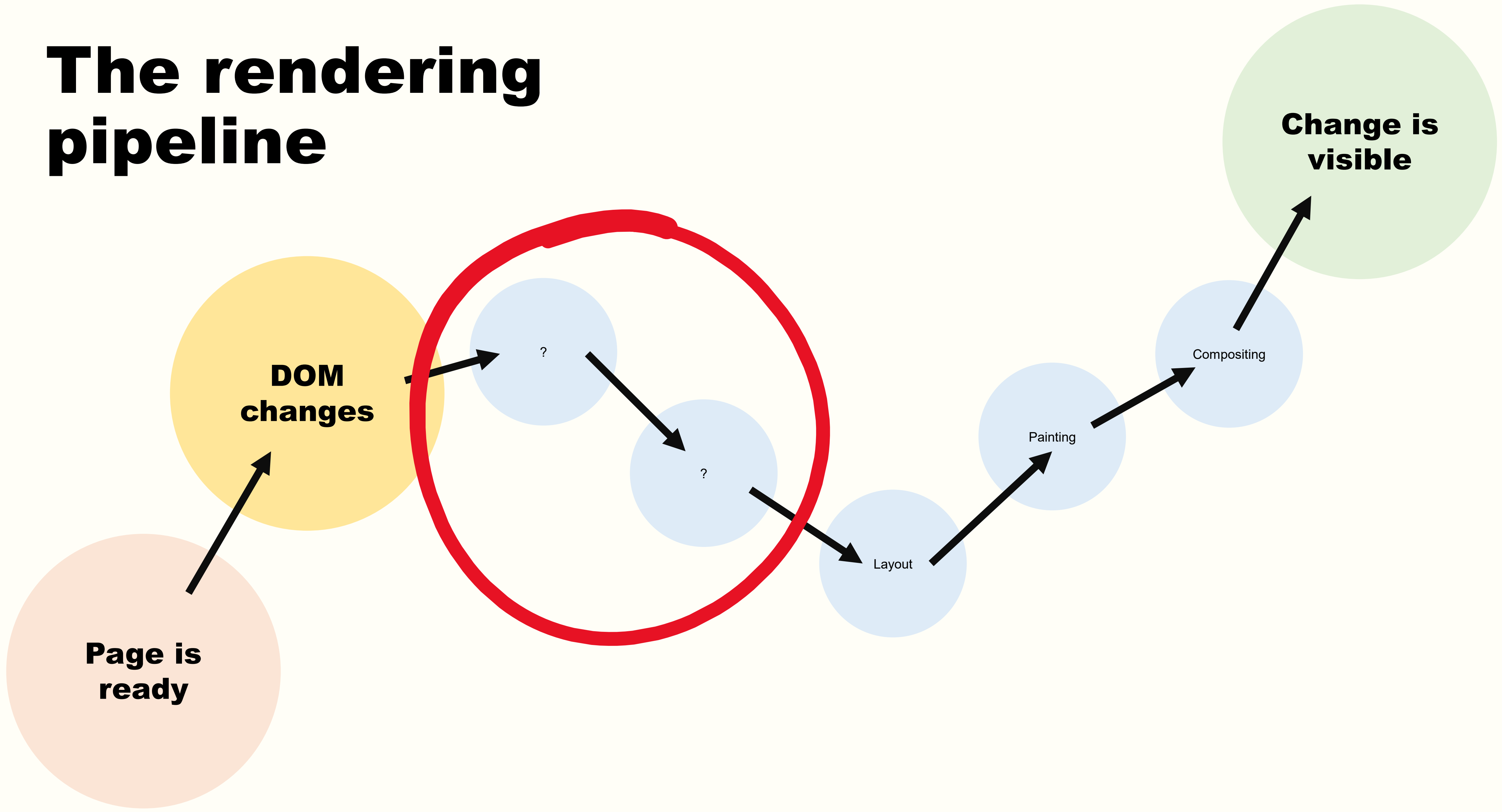
Our scenario



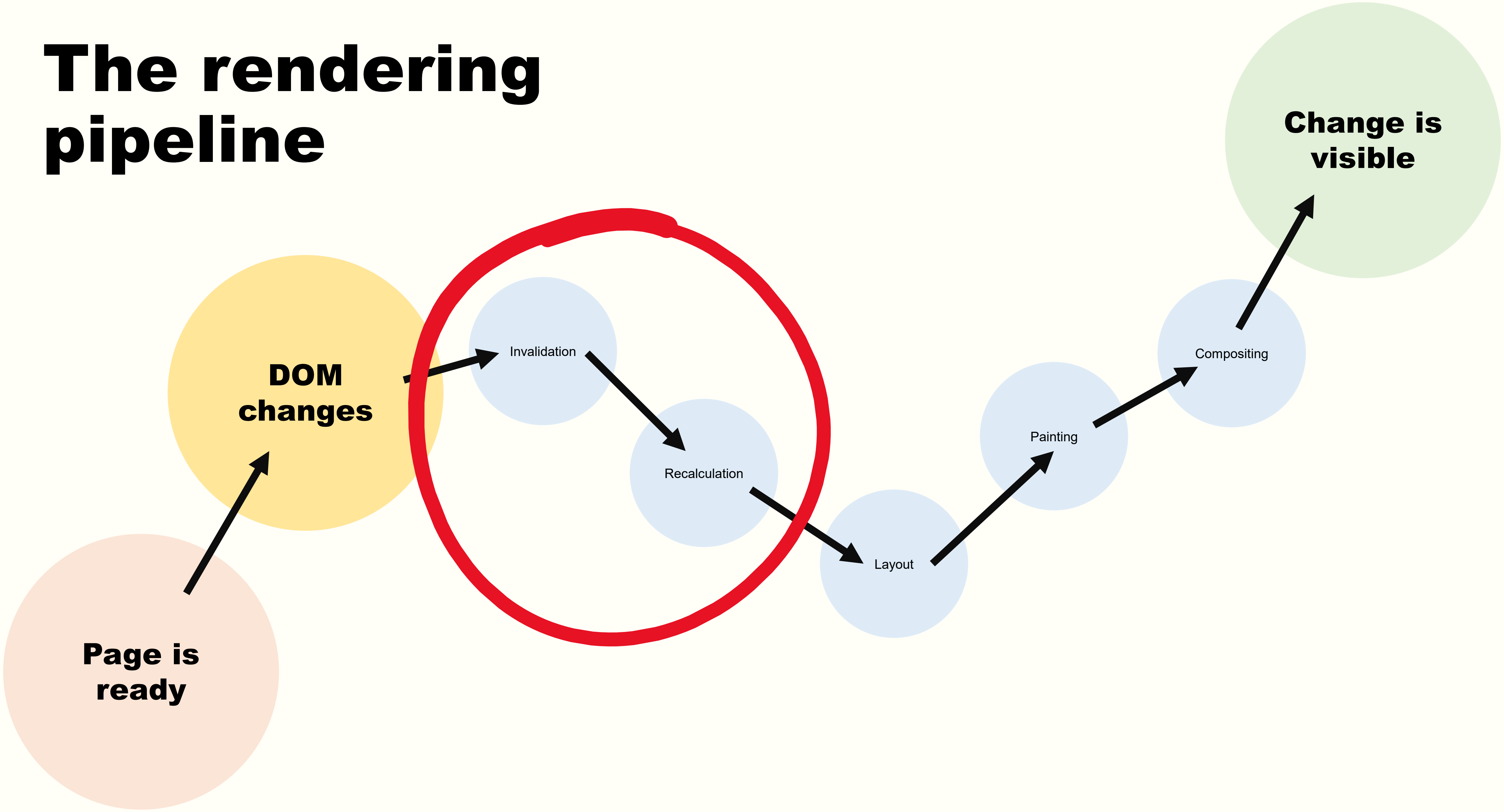
Our scenario



The rendering pipeline



The rendering pipeline



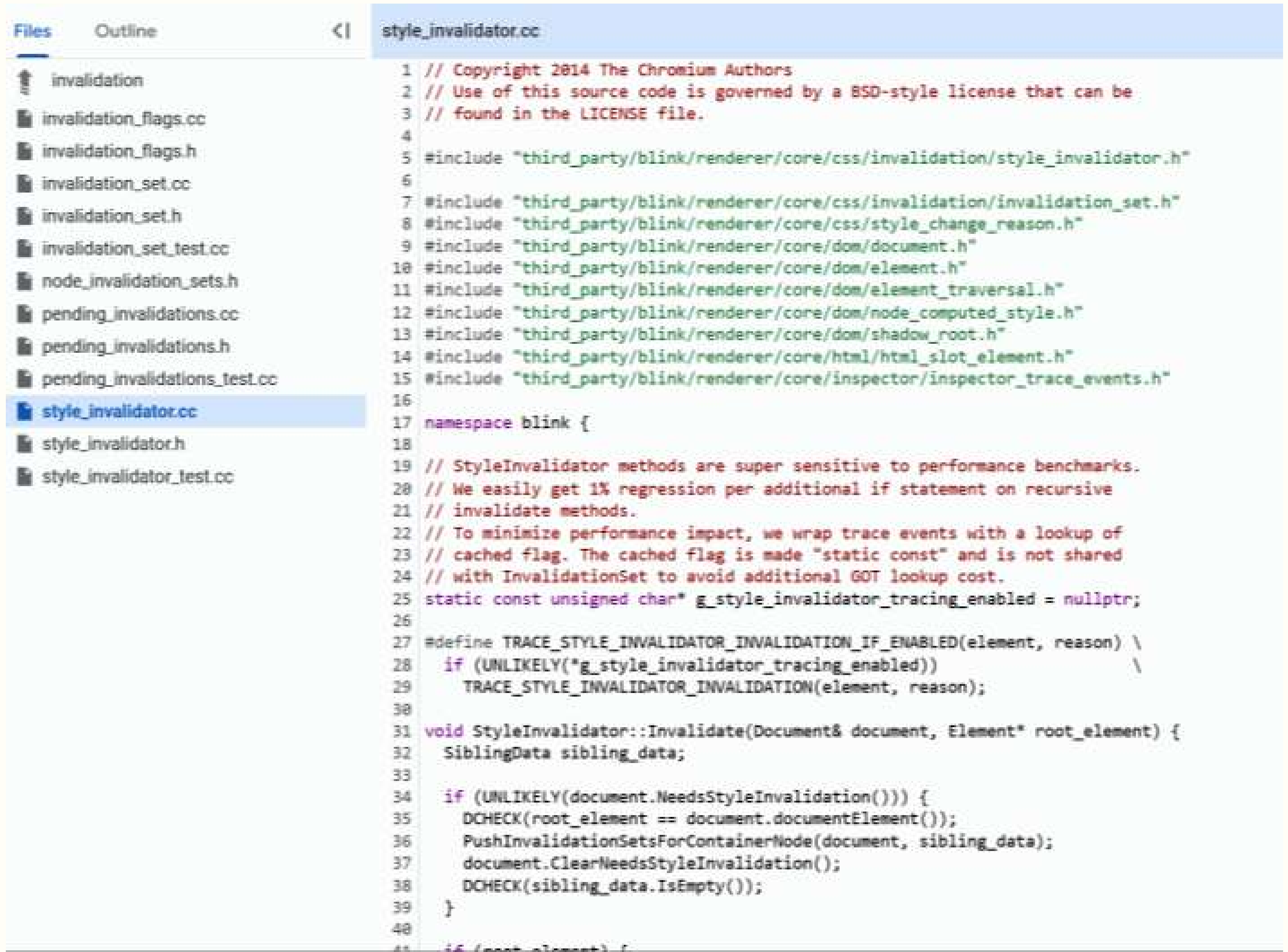
Invalidation?

```
<div class="container selected"> ⚑
  <div id="content" class="snippet-hidden">
    <div class="inner-content clearfix"> ⚑
      <div id="question-header" class="d-flex sm:fd-column">
        <h1 class="fs-headline1"> ⚑
          <a href="…" class="question-hyperlink">DOM?</a> ⚑
        </h1>
      </div>
    </div>
  </div>
</div>
```

Naive implementation

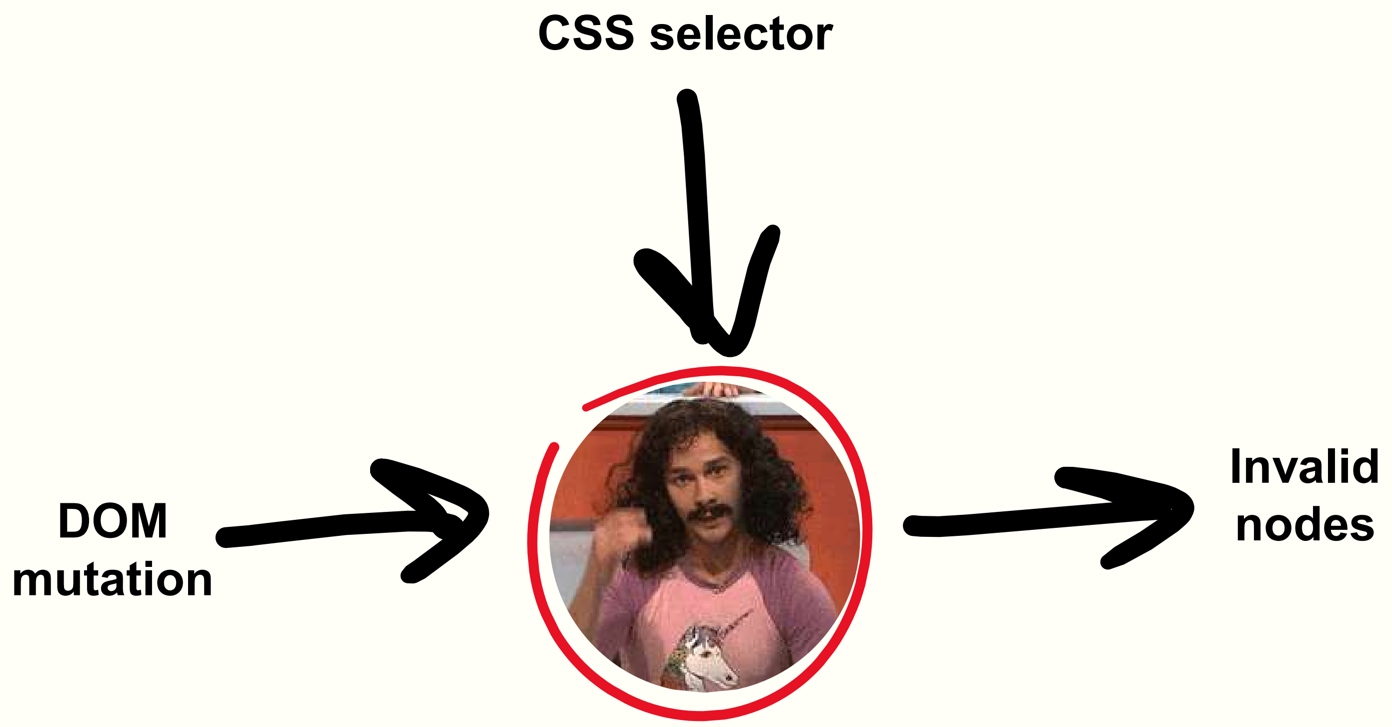
```
function onDomMutation(mutationEvent) {
  // Flag all elements as having their styles invalid!
  const invalidElements = document.querySelectorAll("*");

  // Recalculate styles.
  for (const el of invalidElements) {
    recalculateStyles(el);
  }

  // Re-layout and paint.
  …
}
```

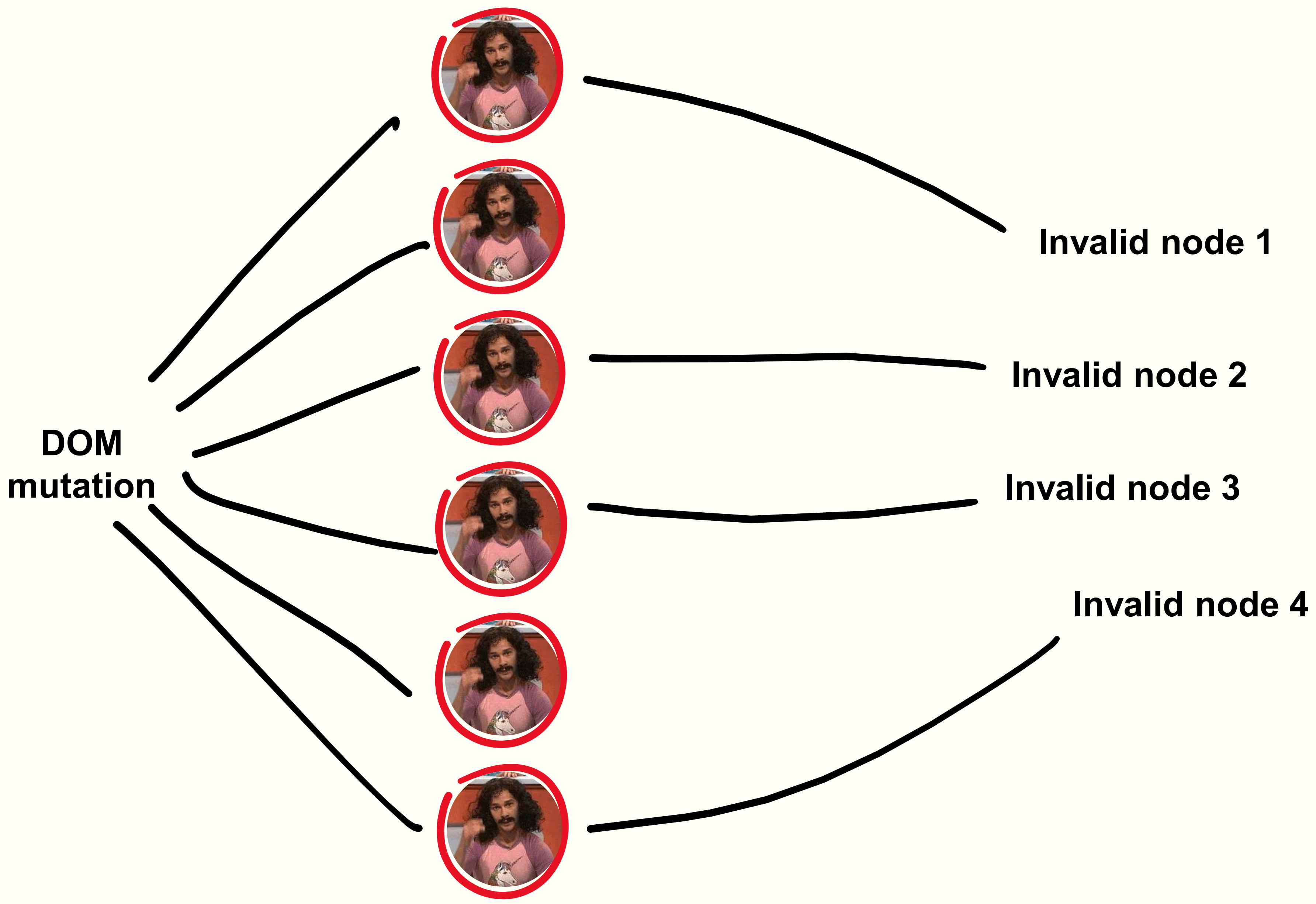
Real implementation

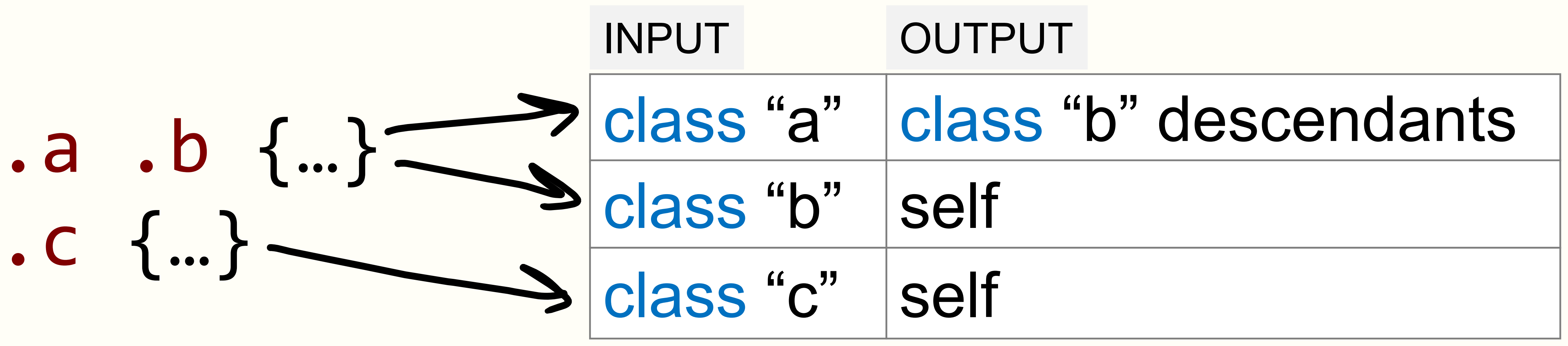


```
// Copyright 2014 The Chromium Authors
// Use of this source code is governed by a BSD-style license that can be
// found in the LICENSE file.

#include "third_party/blink/renderer/core/css/invalidation/style_invalidator.h"

#include "third_party/blink/renderer/core/css/invalidation/invalidation_set.h"
#include "third_party/blink/renderer/core/css/style_change_reason.h"
#include "third_party/blink/renderer/core/dom/document.h"
#include "third_party/blink/renderer/core/dom/element.h"
#include "third_party/blink/renderer/core/dom/element_traversal.h"
#include "third_party/blink/renderer/core/dom/node_computed_style.h"
#include "third_party/blink/renderer/core/dom/shadow_root.h"
#include "third_party/blink/renderer/core/html/html_slot_element.h"
#include "third_party/blink/renderer/core/inspector/inspector_trace_events.h"

namespace blink {

// StyleInvalidator methods are super sensitive to performance benchmarks.
// We easily get 1% regression per additional if statement on recursive
// invalidate methods.
// To minimize performance impact, we wrap trace events with a lookup of
// cached flag. The cached flag is made "static const" and is not shared
// with InvalidationSet to avoid additional GOT lookup cost.
static const unsigned char* g_style_invalidator_tracing_enabled = nullptr;

#define TRACE_STYLE_INVALIDATOR_INVALIDATION_IF_ENABLED(element, reason) \
  if (UNLIKELY(*g_style_invalidator_tracing_enabled))                   \
    TRACE_STYLE_INVALIDATOR_INVALIDATION(element, reason);

void StyleInvalidator::Invalidate(Document& document, Element* root_element) {
  SiblingData sibling_data;

  if (UNLIKELY(document.NeedsStyleInvalidation())) {
    DCHECK(root_element == document.documentElement());
    PushInvalidationSetsForContainerNode(document, sibling_data);
    document.ClearNeedsStyleInvalidation();
    DCHECK(sibling_data.IsEmpty());
  }

  if (root_element) {
```

Invalidation sets are magic




DOM mutation
Invalid node 1
Invalid node 2
Invalid node 3
Invalid node 4



```
.a .b {…}
.c {…}
```

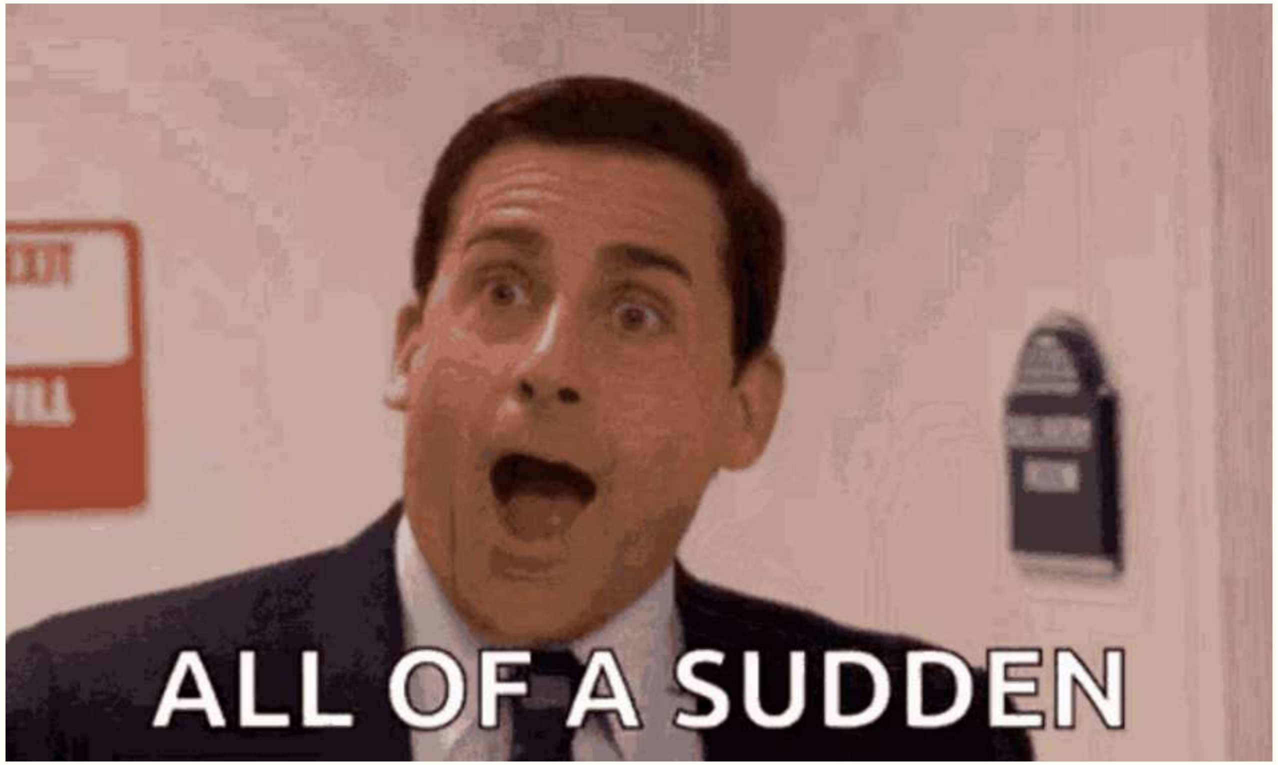
INPUT	OUTPUT
class “a”	class “b” descendants
class “b”	self
class “c”	self

```
.a :not(.b) {…}
#x * {…}
```

INPUT	OUTPUT
class “a”	whole subtree
id “x”	whole subtree

```
.foo > .bar ~ :nth-child(even) { ... }
```

Suddenly, a DOM mutation occurs



Mutation?

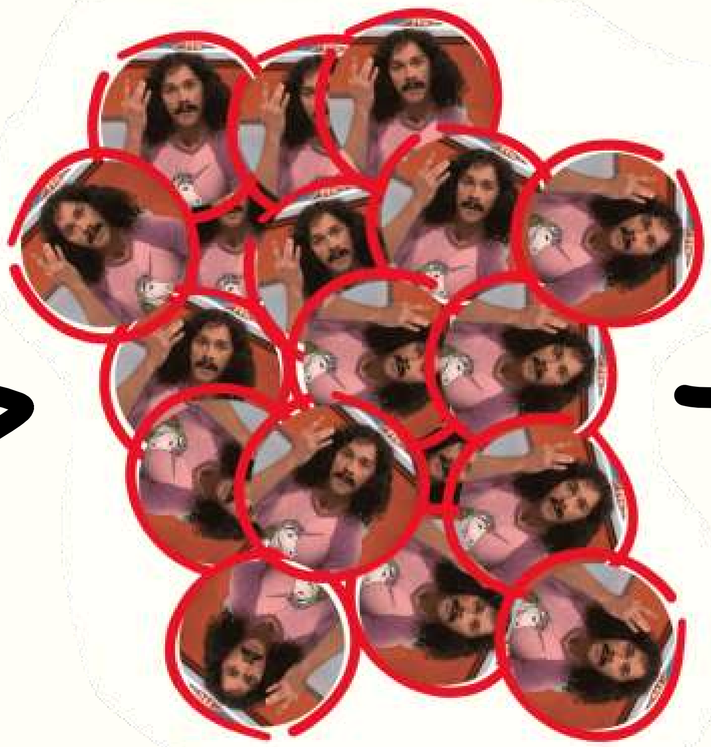
- Toggling a class
- Adding/removing an element
- Hovering an element

```
:hover { ... }
```

Style invalidation starts

add
class
foo

→



→

Immediate
and pending
invalidations

.a .b .c {...}

add class c →

Immediate invalidation

Still need to check **.a** and **.b** parents but done at selector matching during style recalculation

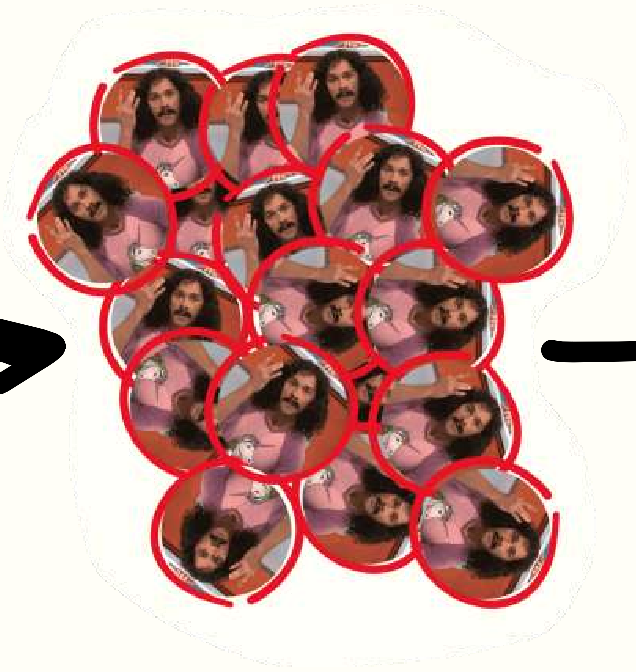
add class a →

Pending invalidation

Don’t know yet if and what to invalidate. Will descend in subtree to find **.b** and **.c**

add
class
foo

→



→

Immediate
and pending
invalidations

→

Invalid
elements

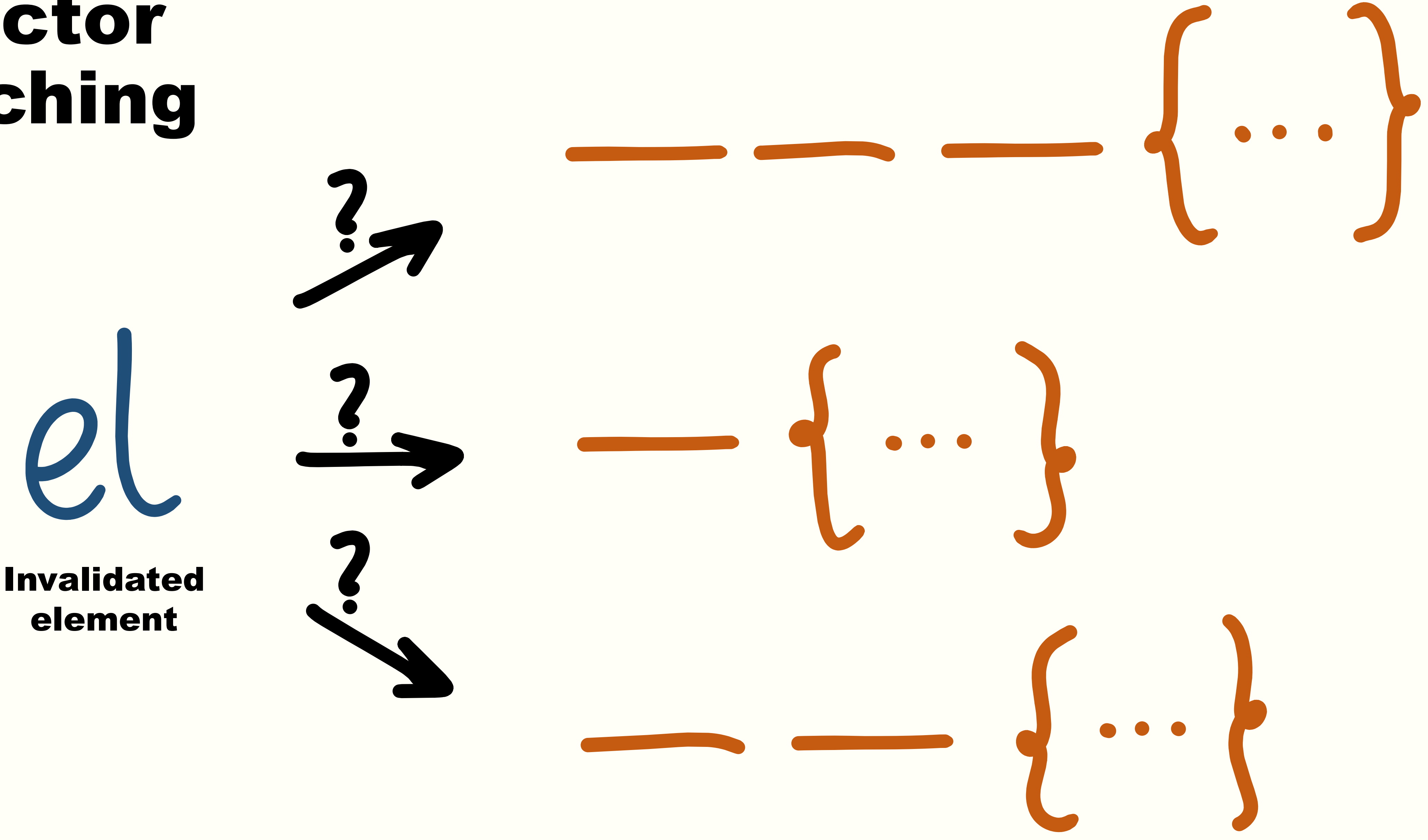
Style recalculation starts when ...

16ms

or

`a = el.offsetWidth`

Selector matching



style → layout → painting → compositing

Can you make this thing slow?

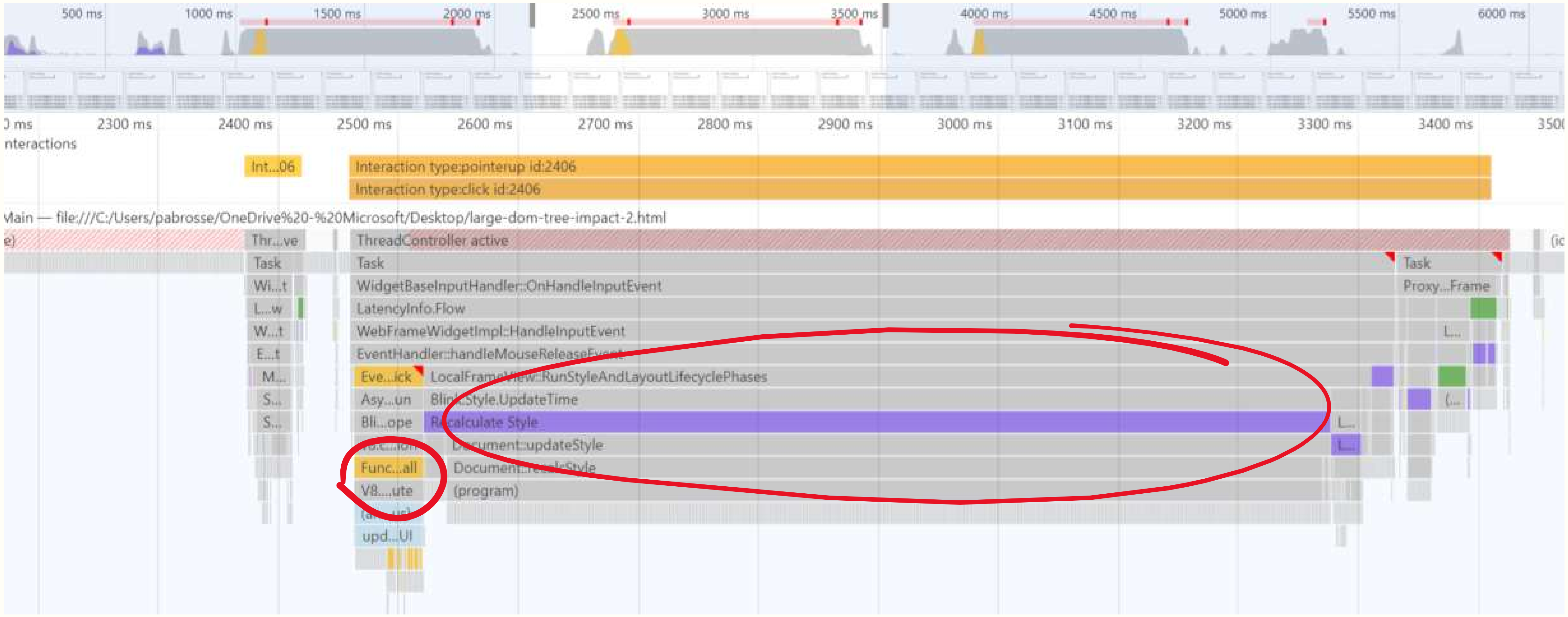
Demo...

https://patrickbrosset.com/lab/css-recalc-talk/tabbar1/

https://patrickbrosset.com/lab/css-recalc-talk/tabbar2/

Can you make this thing slow?

- 😭 Large and deep DOM tree
- 😭 Frequent and large DOM changes
- 😭 Huge CSS stylesheets
- 😭 Less optimized CSS selectors


500 ms
1000 ms
1500 ms
2000 ms
2500 ms
3000 ms
3500 ms
4000 ms
4500 ms
5000 ms
5500 ms
6000 ms
2300 ms
2400 ms
2500 ms
2600 ms
2700 ms
2800 ms
2900 ms
3000 ms
3100 ms
3200 ms
3300 ms
3400 ms
nteractions
Int...06
Interaction type:pointerup id:2406
Interaction type:click id:2406
Main — file:///C:/Users/pabrosse/OneDrive%20-%20Microsoft/Desktop/large-dom-tree-impact-2.html
Thr...ve
ThreadController active
Task
Task
Task
Wi...t
WidgetBaseInputHandler::OnHandleInputEvent
Proxy...Frame
L...w
LatencyInfo.Flow
W...t
WebFrameWidgetImpl::HandleInputEvent
E...t
EventHandler::handleMouseReleaseEvent
M...
Eve...ick
LocalFrameView::RunStyleAndLayoutLifecyclePhases
S...
Asy...un
Blink.Style.UpdateTime
S...
Bli...ope
Recalculate Style
Document::updateStyle
Func...all
Document::recalcStyle
V8....ute
(program)
upd...UI

Chapter 3

Based on a true story

Complex apps?

100+ sheets | 10000+ rules | 2000~6000 nodes

Step 1 – Total confusion

We're seeing 100ms to 1s+ long recalculate styles when we show this thing!

I don't understand, I've set display:none but it doesn't fix the long recalc.

Is there a way I can capture something in the render process to see what is happening during the recalculate styles?

What css rules could possibly be taking so long??

I can't figure out why there's such a big style recalc…

It'd be awesome if there was a way to profile the app and get a listing of how much time is spent dealing with each style rule.

Step 2 – Repro scenario

Repro the bug
Start a trace
The bug no longer repros
The bug no longer repros

Step 3 – Down the rabbit hole

There are a set of bits on each node that determine the scope of style invalidations, e.g., when a child node is added. These bits are currently getting set on the body via a '~' or series of '+' selectors such that child changes to body end up invalidating the style of all children. Currently there is no code the clears these bits, outside of the node being removed, so even if the selector is removed the state is 'sticky'.

Invalidation sets have the overall principle that they are conservative and quick to calculate against recalc candidates. Certainly, there are degenerate cases where this can cause over-recalc.

wat (⨀_⨀)

Invalidation game #1

```html
<div>
  <h1>Title</h1>
  <p>Lorem ipsum dolor sit amet consectetur...</p>
</div>
```

```css
.foo p {
  color: red;
}
```

```js
document.querySelector("div").classList.add("foo");
```

Invalidation game #2

```html
<div>
  <h1>Title</h1>
  <span>
    <span>
      <span>
        <span>
          <span>Lorem ipsum dolor sit amet...</span>
        </span>
      </span>
    </span>
  </span>
</div>
```

```css
.foo span {
  color: red;
}
```

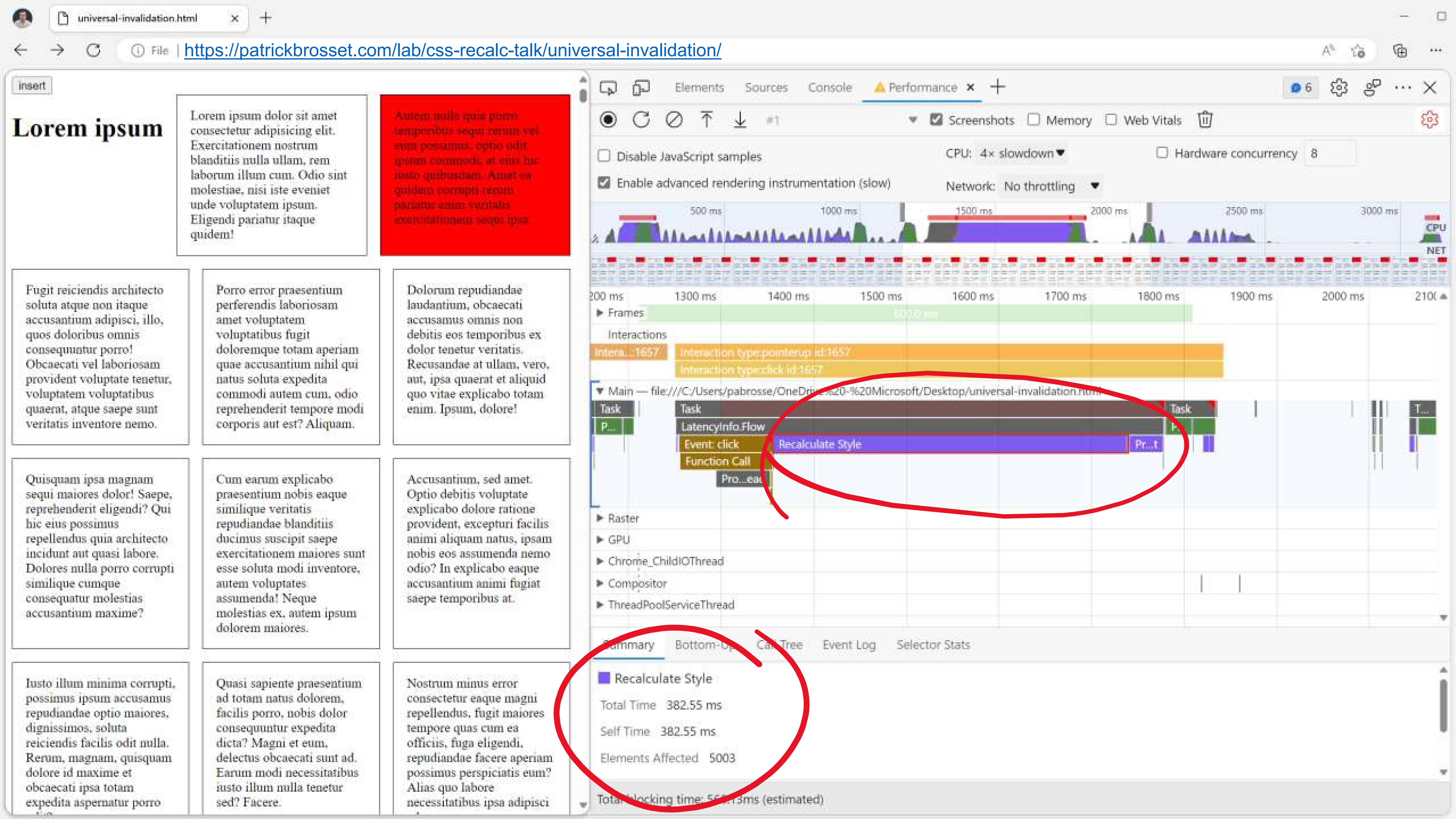
```js
document.querySelector("div").classList.add("foo");
```

Invalidation game #3

```
<div class=“first">
  <h1>Lorem ipsum</h1>
  <p>Lorem ipsum dolor sit...</p>
  <p>Autem nulla quia porro temporibus...</p>
  <p>Fugit reiciendis architect...</p>
  <p>Porro error...</p>
  ... Many more p elements ...
</div>
<div class="last"></div>
```

```
.first h1 + p + p {
  background: red;
}
```

```
document.body.insertBefore(document.createElement('span'), document.querySelector(‘.last'));
```



universal-invalidation.html

https://patrickbrosset.com/lab/css-recalc-talk/universal-invalidation/

insert

Lorem ipsum

Lorem ipsum dolor sit amet consectetur adipisicing elit. Exercitationem nostrum blanditiis nulla ullam, rem laborum illum cum. Odio sint molestiae, nisi iste eveniet unde voluptatem ipsum. Eligendi pariatur itaque quidem!

Autem nulla quia porro temporibus sequi rerum vel enim possimus, optio odit ipsum commodi, at eius hic iusto quibusdam. Amet ea quidem corrupti rerum pariatur enim veritatis exercitationem sequi ipsa

Fugit reiciendis architecto soluta atque non itaque accusantium adipisci, illo, quos doloribus omnis consequuntur porro! Obcaecati vel laboriosam provident voluptate tenetur, voluptatem voluptatibus quaerat, atque saepe sunt veritatis inventore nemo.

Porro error praesentium perferendis laboriosam amet voluptatem voluptatibus fugit doloremque totam aperiam quae accusantium nihil qui natus soluta expedita commodi autem cum, odio reprehenderit tempore modi corporis aut est? Aliquam.

Dolorum repudiandae laudantium, obcaecati accusamus omnis non debitis eos temporibus ex dolor tenetur veritatis. Recusandae at ullam, vero, aut, ipsa quaerat et aliquid quo vitae explicabo totam enim. Ipsum, dolore!

Quisquam ipsa magnam sequi maiores dolor! Saepe, reprehenderit eligendi? Qui hic eius possimus repellendus quia architecto incidunt aut quasi labore. Dolores nulla porro corrupti similique cumque consequatur molestias accusantium maxime?

Cum earum explicabo praesentium nobis eaque similique veritatis repudiandae blanditiis ducimus suscipit saepe exercitationem maiores sunt esse soluta modi inventore, autem voluptates assumenda! Neque molestias ex, autem ipsum dolorem maiores.

Accusantium, sed amet. Optio debitis voluptate explicabo dolore ratione provident, excepturi facilis animi aliquam natus, ipsam nobis eos assumenda nemo odio? In explicabo eaque accusantium animi fugiat saepe temporibus at.

Iusto illum minima corrupti, possimus ipsum accusamus repudiandae optio maiores, dignissimos, soluta reiciendis facilis odit nulla. Rerum, magnam, quisquam dolore id maxime et obcaecati ipsa totam expedita aspernatur porro

Quasi sapiente praesentium ad totam natus dolorem, facilis porro, nobis dolor consequuntur expedita dicta? Magni et eum, delectus obcaecati sunt ad. Earum modi necessitatibus iusto illum nulla tenetur sed? Facere.

Nostrum minus error consectetur eaque magni repellendus, fugit maiores tempore quas cum ea officiis, fuga eligendi, repudiandae facere aperiam possimus perspiciatis eum? Alias quo labore necessitatibus ipsa adipisci

Elements Sources Console Performance

Screenshots ✓ Memory Web Vitals

Disable JavaScript samples
Enable advanced rendering instrumentation (slow) ✓

CPU: 4× slowdown
Network: No throttling
Hardware concurrency 8

Frames
Interactions
Interaction type:pointerup id:1657
Interaction type:click id:1657

Main — file:///C:/Users/pabrosse/OneDrive%20-%20Microsoft/Desktop/universal-invalidation.html

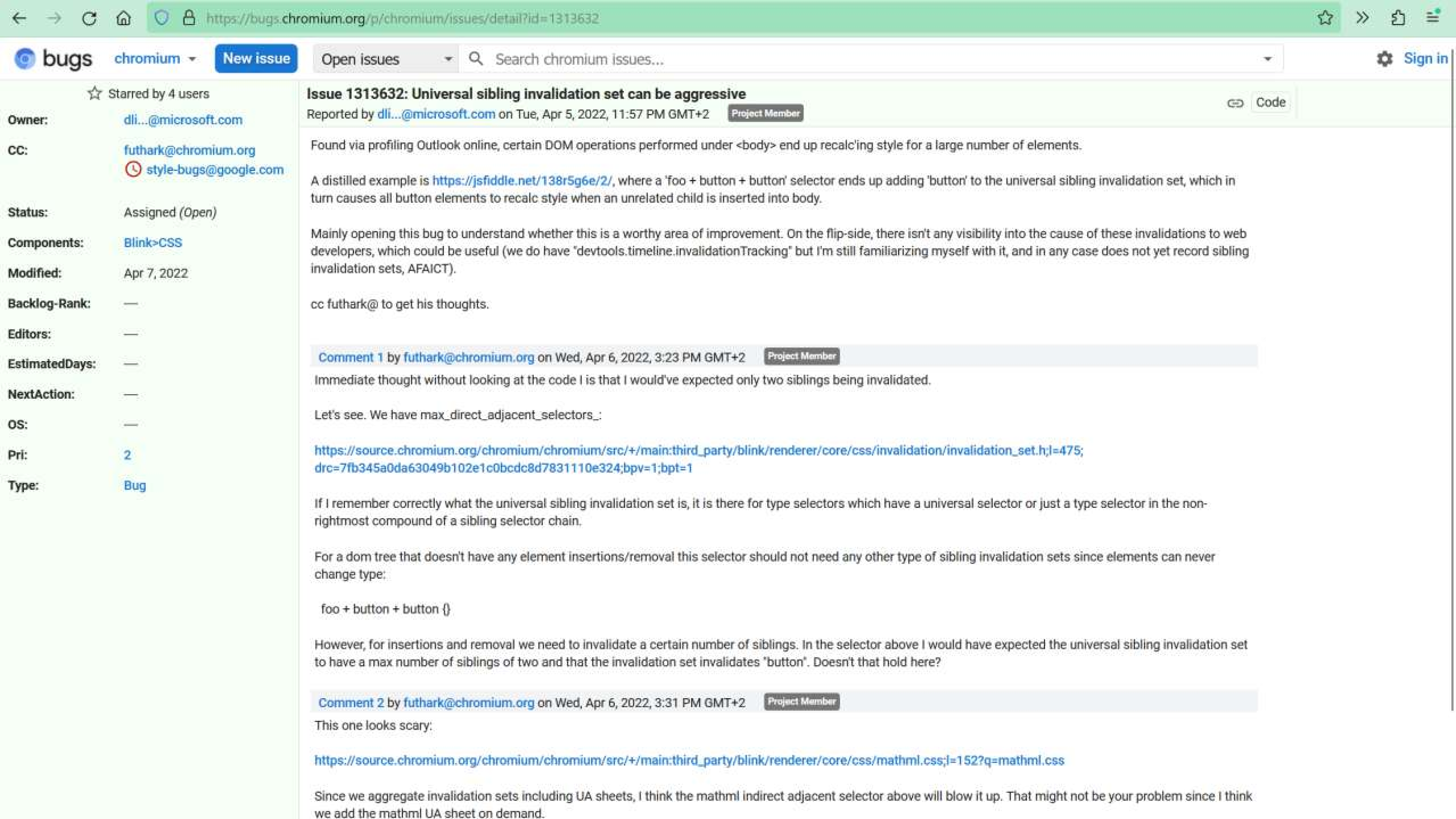
Task
LatencyInfo.Flow
Event: click
Recalculate Style
Function Call

Raster
GPU
Chrome_ChildIOThread
Compositor
ThreadPoolServiceThread

Summary Bottom-Up Call Tree Event Log Selector Stats

Recalculate Style
Total Time 382.55 ms
Self Time 382.55 ms
Elements Affected 5003

Total blocking time: 5??.?3ms (estimated)


https://bugs.chromium.org/p/chromium/issues/detail?id=1313632
bugs
chromium
New issue
Open issues
Search chromium issues...
Sign in
Code

Starred by 4 users

Owner:	dli...@microsoft.com
CC:	futhark@chromium.org style-bugs@google.com
Status:	Assigned *(Open)*
Components:	Blink>CSS
Modified:	Apr 7, 2022
Backlog-Rank:	—
Editors:	—
EstimatedDays:	—
NextAction:	—
OS:	—
Pri:	2
Type:	Bug

Issue 1313632: Universal sibling invalidation set can be aggressive

Reported by dli...@microsoft.com on Tue, Apr 5, 2022, 11:57 PM GMT+2 Project Member

Found via profiling Outlook online, certain DOM operations performed under <body> end up recalc'ing style for a large number of elements.

A distilled example is https://jsfiddle.net/138r5g6e/2/, where a 'foo + button + button' selector ends up adding 'button' to the universal sibling invalidation set, which in turn causes all button elements to recalc style when an unrelated child is inserted into body.

Mainly opening this bug to understand whether this is a worthy area of improvement. On the flip-side, there isn't any visibility into the cause of these invalidations to web developers, which could be useful (we do have "devtools.timeline.invalidationTracking" but I'm still familiarizing myself with it, and in any case does not yet record sibling invalidation sets, AFAICT).

cc futhark@ to get his thoughts.

Comment 1 by futhark@chromium.org on Wed, Apr 6, 2022, 3:23 PM GMT+2 Project Member

Immediate thought without looking at the code I is that I would've expected only two siblings being invalidated.

Let's see. We have max_direct_adjacent_selectors_:

https://source.chromium.org/chromium/chromium/src/+/main:third_party/blink/renderer/core/css/invalidation/invalidation_set.h;l=475;drc=7fb345a0da63049b102e1c0bcdc8d7831110e324;bpv=1;bpt=1

If I remember correctly what the universal sibling invalidation set is, it is there for type selectors which have a universal selector or just a type selector in the non-rightmost compound of a sibling selector chain.

For a dom tree that doesn't have any element insertions/removal this selector should not need any other type of sibling invalidation sets since elements can never change type:

```
foo + button + button {}
```

However, for insertions and removal we need to invalidate a certain number of siblings. In the selector above I would have expected the universal sibling invalidation set to have a max number of siblings of two and that the invalidation set invalidates "button". Doesn't that hold here?

Comment 2 by futhark@chromium.org on Wed, Apr 6, 2022, 3:31 PM GMT+2 Project Member

This one looks scary:

https://source.chromium.org/chromium/chromium/src/+/main:third_party/blink/renderer/core/css/mathml.css;l=152?q=mathml.css

Since we aggregate invalidation sets including UA sheets, I think the mathml indirect adjacent selector above will blow it up. That might not be your problem since I think we add the mathml UA sheet on demand.

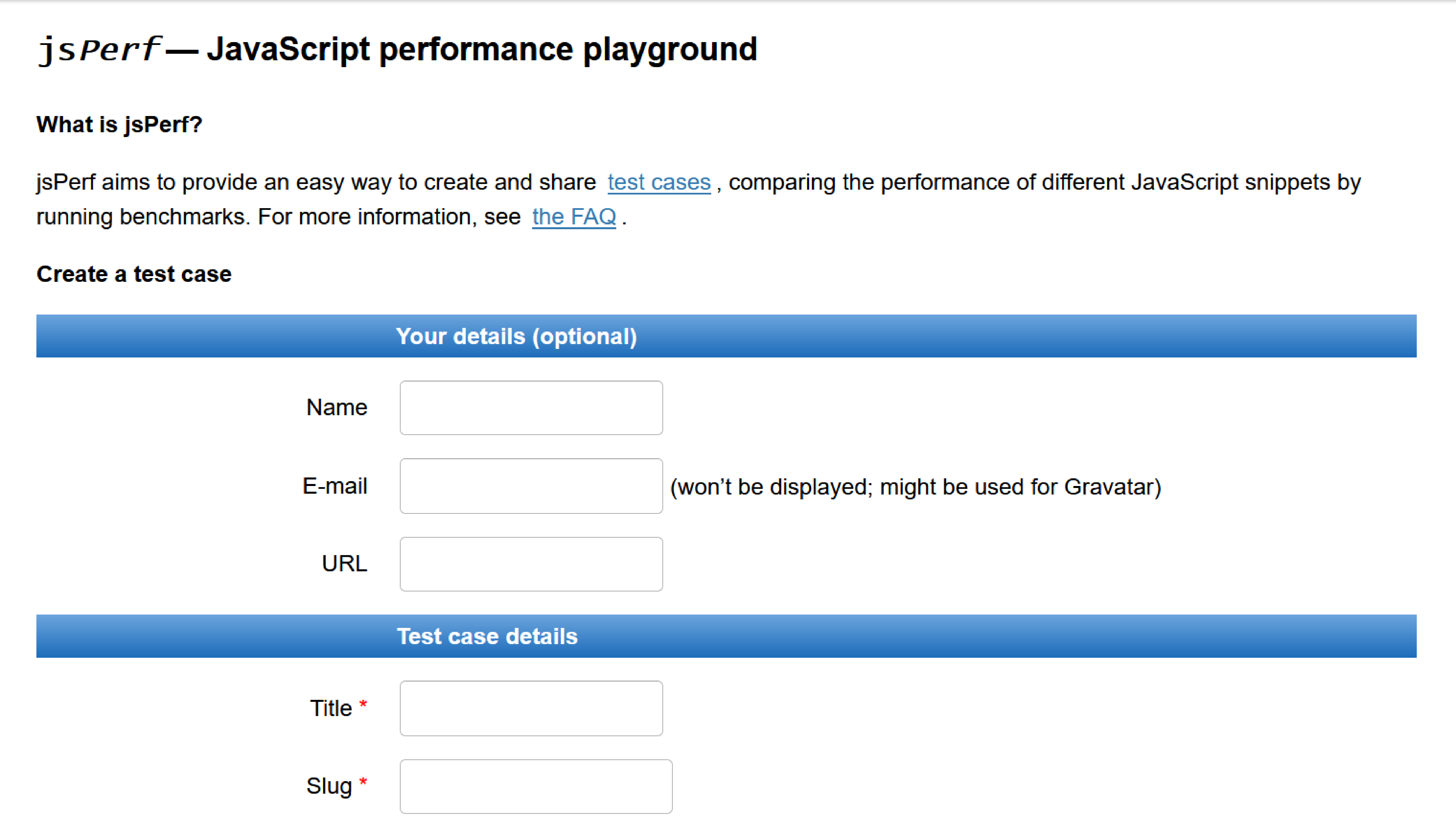
Chapter 4

The ignorant compliant

Solution #1 Ignore the problem

Solution #2
Comply with arbitrary rules

Remember JS micro benchmarks?



jsPerf — JavaScript performance playground

What is jsPerf?

jsPerf aims to provide an easy way to create and share test cases, comparing the performance of different JavaScript snippets by running benchmarks. For more information, see the FAQ.

Create a test case

Your details (optional)

Name		
E-mail		(won’t be displayed; might be used for Gravatar)
URL		

Test case details

Title *	
Slug *	

```
// Case 1

var x = [];
var start = performance.now();
for (var i = 0; i < 1000000; i++) {
  x[i] = "x";
}

var end = performance.now();
console.log(end - start);
```

```
// Case 2

x = [];
start = performance.now();
for (var i = 0; i < 1000000; i++) {
  x[x.length] = "x";
}

end = performance.now();
console.log(end - start);
```

```
// Case 3

x = [];
start = performance.now();
for (var i = 0; i < 1000000; i++) {
  x.push( "x" );
}

end = performance.now();
console.log(end - start);
```

*

4 simple steps

1. Identify slow scenarios
2. Measure
3. Try to improve
4. Go back to step 2

Chapter 5

DevTools

I f’ing love DevTools

- I’ve worked on browser DevTools for 10 years
- I maintain devtoolstips.org which contains hundreds of tips and tricks

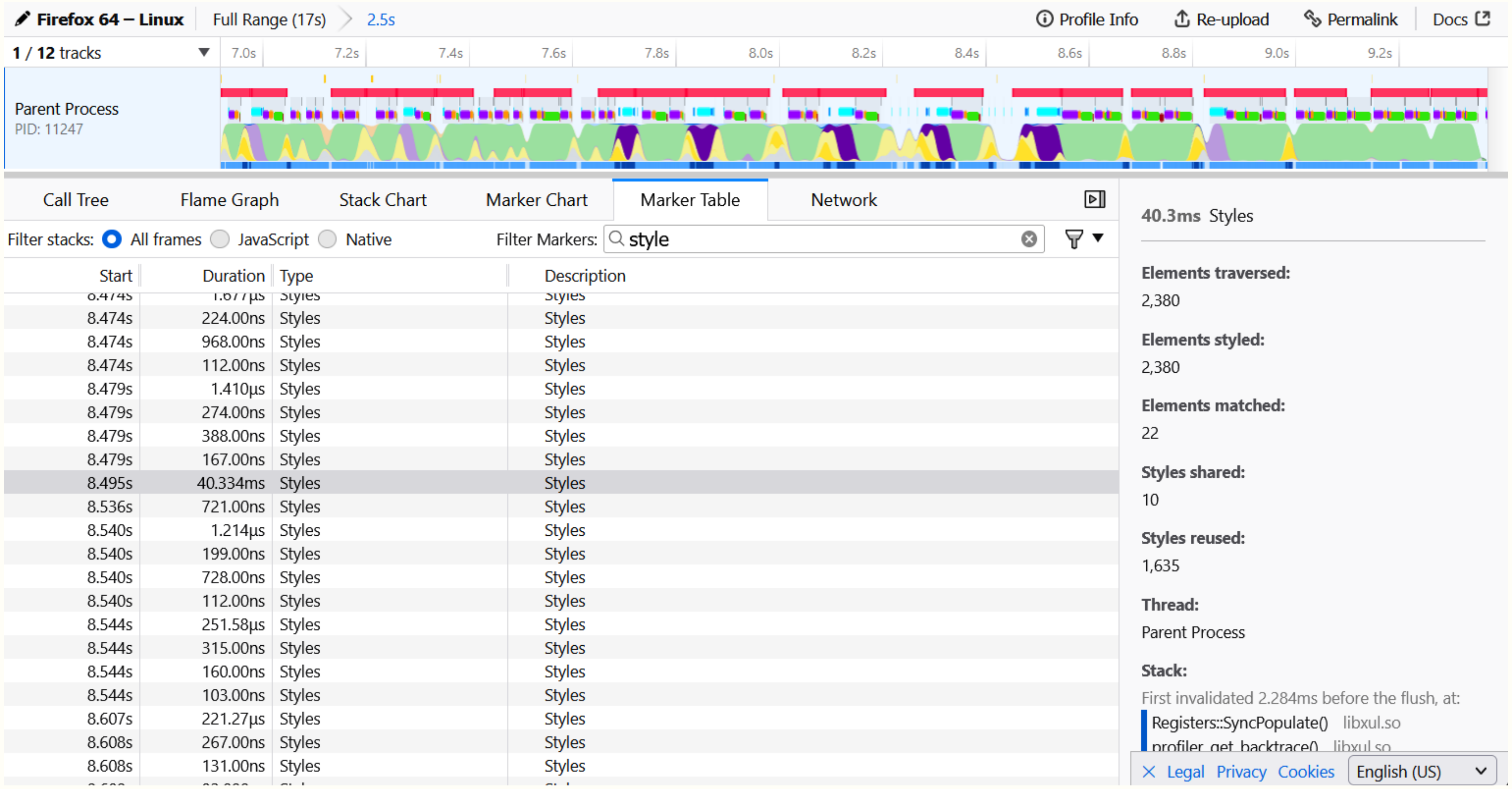
1. Firefox Profiler

2. Chromium Performance tool

   a) Invalidation tracking experiment
   b) Event initiator experiment

3. Edge Selector Stats

Firefox Profiler



Start	Duration	Type	Description
8.474s	1.677µs	Styles	Styles
8.474s	224.00ns	Styles	Styles
8.474s	968.00ns	Styles	Styles
8.474s	112.00ns	Styles	Styles
8.479s	1.410µs	Styles	Styles
8.479s	274.00ns	Styles	Styles
8.479s	388.00ns	Styles	Styles
8.479s	167.00ns	Styles	Styles
8.495s	40.334ms	Styles	Styles
8.536s	721.00ns	Styles	Styles
8.540s	1.214µs	Styles	Styles
8.540s	199.00ns	Styles	Styles
8.540s	728.00ns	Styles	Styles
8.540s	112.00ns	Styles	Styles
8.544s	251.58µs	Styles	Styles
8.544s	315.00ns	Styles	Styles
8.544s	160.00ns	Styles	Styles
8.544s	103.00ns	Styles	Styles
8.607s	221.27µs	Styles	Styles
8.608s	267.00ns	Styles	Styles
8.608s	131.00ns	Styles	Styles

40.3ms Styles

Elements traversed:
2,380

Elements styled:
2,380

Elements matched:
22

Styles shared:
10

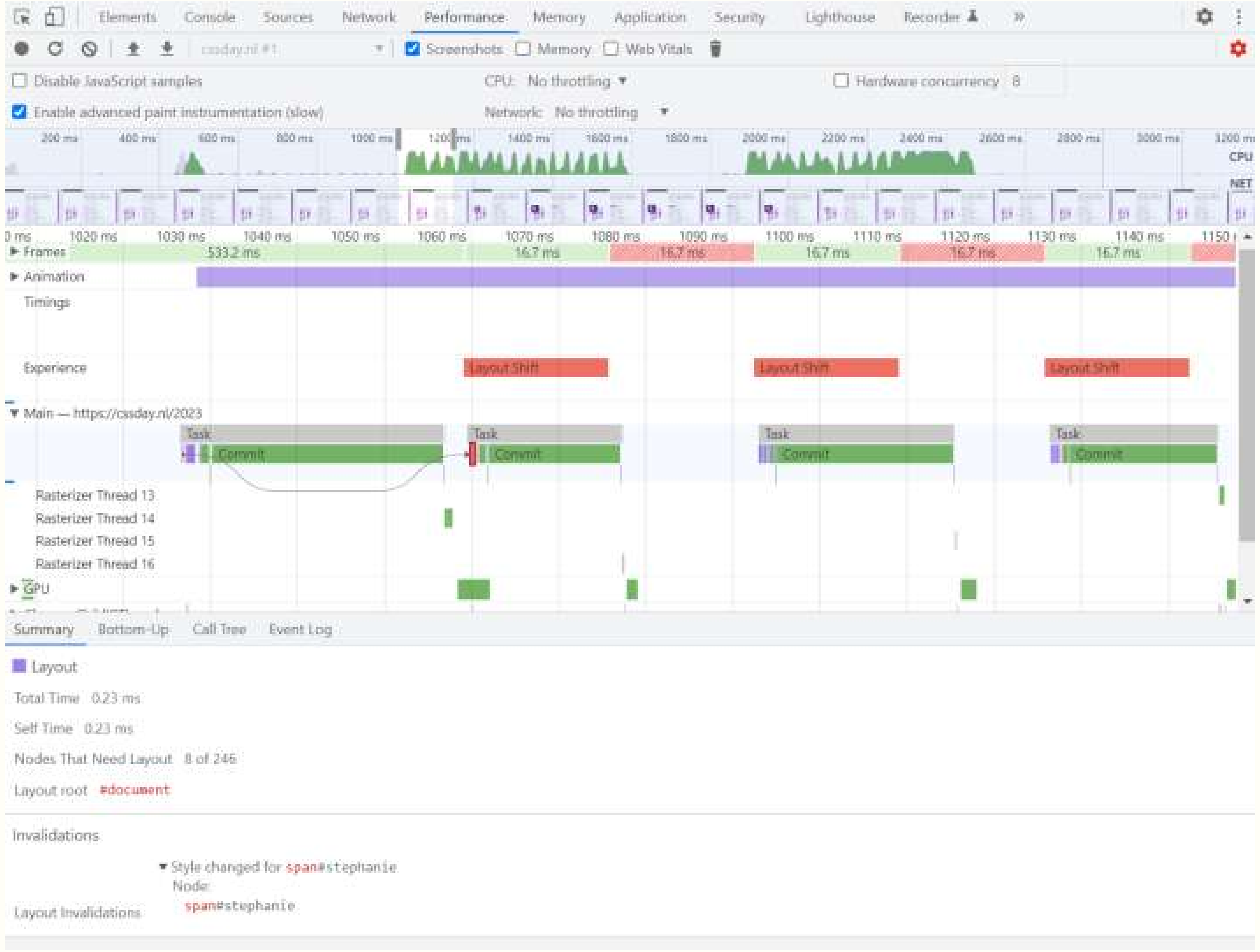
Styles reused:
1,635

Thread:
Parent Process

Stack:
First invalidated 2.284ms before the flush, at:

```
Registers::SyncPopulate()   libxul.so
profiler_get_backtrace()   libxul.so
```

Chromium Performance tool



☑ Screenshots ☐ Memory ☐ Web Vitals

☐ Disable JavaScript samples

☑ Enable advanced paint instrumentation (slow)

CPU: No throttling

Network: No throttling

☐ Hardware concurrency 8

Layout

Total Time 0.23 ms

Self Time 0.23 ms

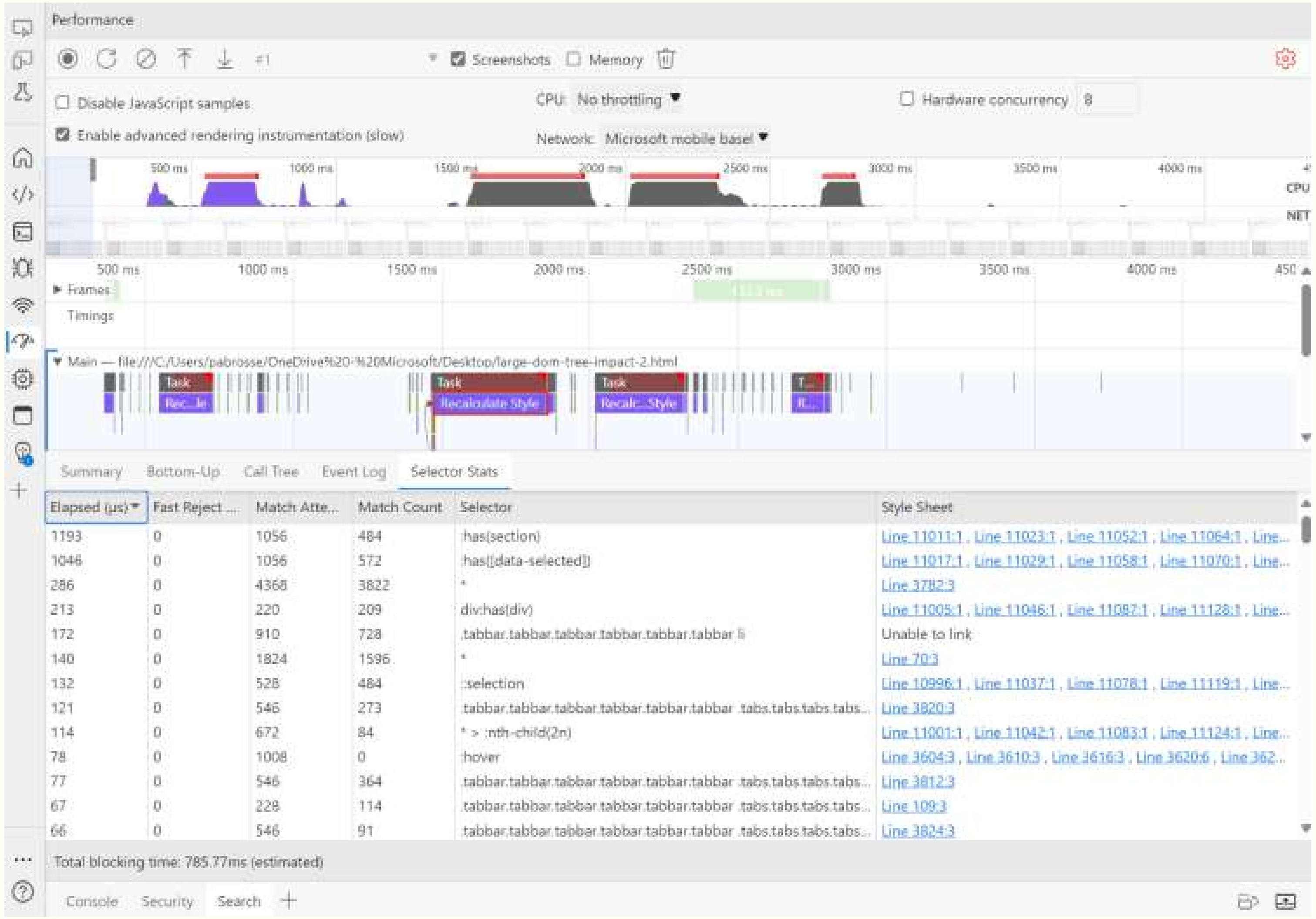
Nodes That Need Layout 8 of 246

Layout root #document

Invalidations

Layout Invalidations
▾ Style changed for span#stephanie
Node:
span#stephanie

Edge
Selector Stats



Performance
Screenshots
Memory
Disable JavaScript samples
Enable advanced rendering instrumentation (slow)
CPU: No throttling
Network: Microsoft mobile basel
Hardware concurrency 8
CPU
NET
Frames
Timings
Main — file:///C:/Users/pabrosse/OneDrive%20-%20Microsoft/Desktop/large-dom-tree-impact-2.html
Task
Recalculate Style
Summary
Bottom-Up
Call Tree
Event Log
Selector Stats
Console
Security
Search

Elapsed (µs)	Fast Reject ...	Match Atte...	Match Count	Selector	Style Sheet
1193	0	1056	484	:has(section)	Line 11011:1 , Line 11023:1 , Line 11052:1 , Line 11064:1 , Line...
1046	0	1056	572	:has([data-selected])	Line 11017:1 , Line 11029:1 , Line 11058:1 , Line 11070:1 , Line...
286	0	4368	3822	*	Line 3782:3
213	0	220	209	div:has(div)	Line 11005:1 , Line 11046:1 , Line 11087:1 , Line 11128:1 , Line...
172	0	910	728	.tabbar.tabbar.tabbar.tabbar.tabbar.tabbar li	Unable to link
140	0	1824	1596	*	Line 70:3
132	0	528	484	::selection	Line 10996:1 , Line 11037:1 , Line 11078:1 , Line 11119:1 , Line...
121	0	546	273	.tabbar.tabbar.tabbar.tabbar.tabbar.tabbar .tabs.tabs.tabs.tabs...	Line 3820:3
114	0	672	84	* > :nth-child(2n)	Line 11001:1 , Line 11042:1 , Line 11083:1 , Line 11124:1 , Line...
78	0	1008	0	:hover	Line 3604:3 , Line 3610:3 , Line 3616:3 , Line 3620:6 , Line 362...
77	0	546	364	.tabbar.tabbar.tabbar.tabbar.tabbar.tabbar .tabs.tabs.tabs.tabs...	Line 3812:3
67	0	228	114	.tabbar.tabbar.tabbar.tabbar.tabbar.tabbar .tabs.tabs.tabs.tabs...	Line 109:3
66	0	546	91	.tabbar.tabbar.tabbar.tabbar.tabbar.tabbar .tabs.tabs.tabs.tabs...	Line 3824:3

Total blocking time: 785.77ms (estimated)

Remember

- Don’t blindly follow rules.
- Measure, improve, measure.
- CSS selector performance mostly doesn’t matter.

Thank you!

- **Nolan Lawson's CSS runtime performance talk at performance.now() in 2022**
  https://www.youtube.com/watch?v=nWcexTnvIKI
- **Kevin Powell's video on selector performance**
  https://www.youtube.com/watch?v=J24xS21FlmY
- **Performance tool docs**
  https://developer.chrome.com/docs/devtools/performance/
- **My blog post about selector performance**
  https://blogs.windows.com/msedgedev/2023/01/17/the-truth-about-css-selector-performance/
- **Edge Selector Stats docs**
  https://learn.microsoft.com/microsoft-edge/devtools-guide-chromium/evaluate-performance/selector-stats