Fork, clone and follow setup instructions:

github.com/liveloveapp/fast-apps-with-devtools

build blazing-fast apps using chrome devtools

LiveLoveApp

Brian Love

Principal Architect, LiveLoveApp

@brian_love

Mike Ryan

Principal Architect, LiveLoveApp

@MikeRyanDev

Housekeeping

- [x] Code of Conduct
- [x] 10 am to 4 pm MST
- [x] Two 20-minute breaks
- [x] 1 hour break for lunch

The Value of Speed

Pinterest reduced perceived wait times by 40% resulting in 15%+ sign-ups

— https://web.dev/why-speed-matters/

A 1s reduction in response time could cost Amazon $1.6b per year.

— Fast Company

“Load is not a single moment in time — it’s an experience that no one metric can fully capture”

— W3C Paint Timing Spec

The RAIL Model

0–16 ms

smooth animations at 60 fps

0–100 ms

responsive to user actions

100–1000 ms

waiting time for loading or page transition

1000+ ms

focus is lost

10000+ ms

The RAIL Model

Response

Animation

Idle

Load

Response

The RAIL Model

Response

- < 100 ms response to user input
- < 50 ms process user events
- Provide feedback for tasks that take more than 50 ms

Animation

The RAIL Model

Animation

- < 10 ms produce a frame
- 60 fps
- Aim for visual smoothness
- This includes scrolling & dragging

Idle

The RAIL Model

Idle

- Maximize idle time
- Use idle time for deferred work
- < 50 ms of work during idle
- User interaction is higher priority than idle work

Load

The RAIL Model

Load

- < 5 s load time on 3G
- < 2 s for second load
- Eliminate render blocking resources
- Use lazy-loading and code-splitting techniques

Metrics

What are metrics?

- ✅ Precise
- ✅ Quantitatively measurable

How are they measured?

- [x] Lab
- [x] Field

Metric types

- [x] Perceived load time
- [x] Responsiveness
- [x] Runtime
- [x] Visual stability
- [x] Smoothness

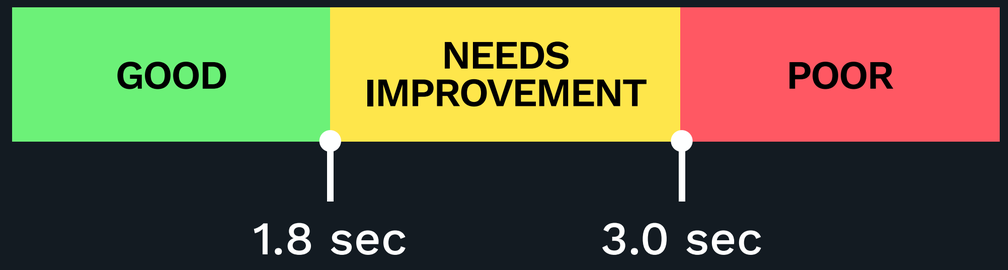
First Contentful Paint (FCP)

First Contentful Paint (FCP)

- Measures perceived load speed
- Reassures something is *happening*
- Measure in lab and field

FCP

First Contentful Paint



Largest Contentful Paint (LCP)

Page Load

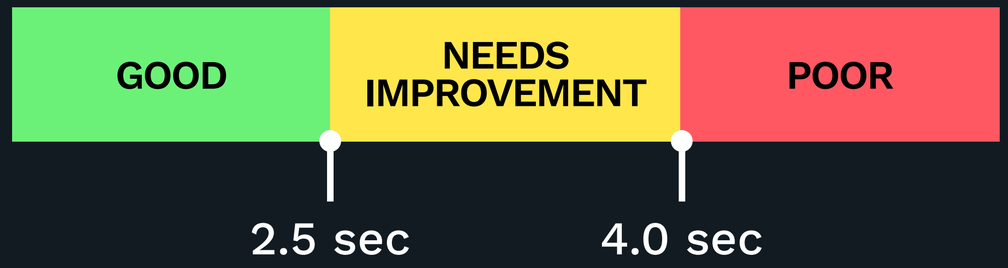
Largest text/ image block rendered

Largest Contentful Paint (LCP)

- Measures perceived load speed
- Reassures the page is *helpful*
- Measure in lab and field

LCP

Largest Contentful Paint



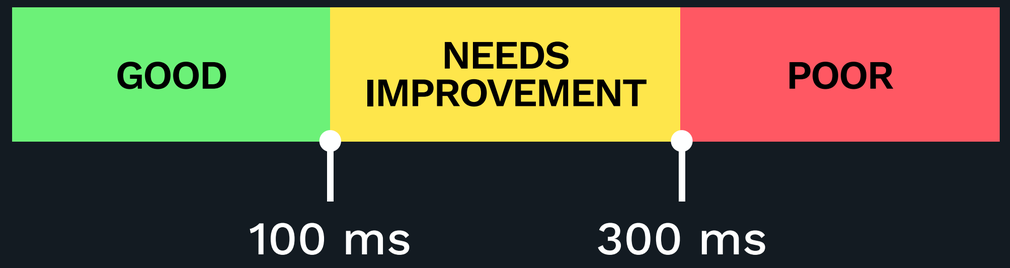
First Input Delay (FID)

First Input Delay (FID)

- Measures load responsiveness
- Ensures page is *usable*
- Measure in field

FID

First Input Delay



Time to Interactivity (TTI)

Page Load

Reliable response to interactive input

Time to Interactivity (TTI)

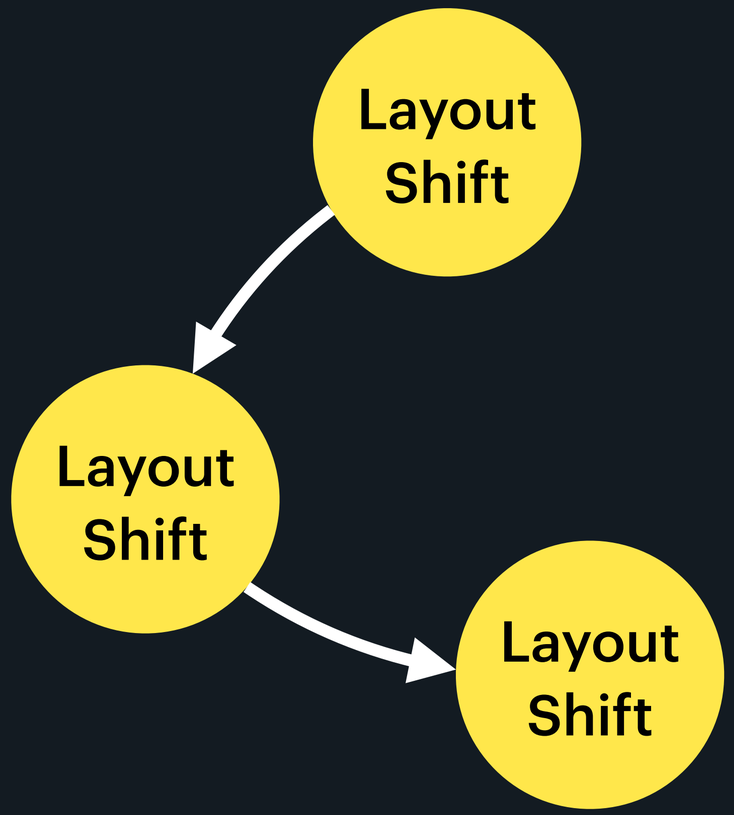
- Measures load responsiveness
- Ensures the page is *usable*
- Measure in lab

Total Blocking Time (TBT)

Total Blocking Time (TBT)

- Measures load responsiveness
- Ensures the page is *usable*
- Measure in lab

Cumulative Layout Shift (CLS)



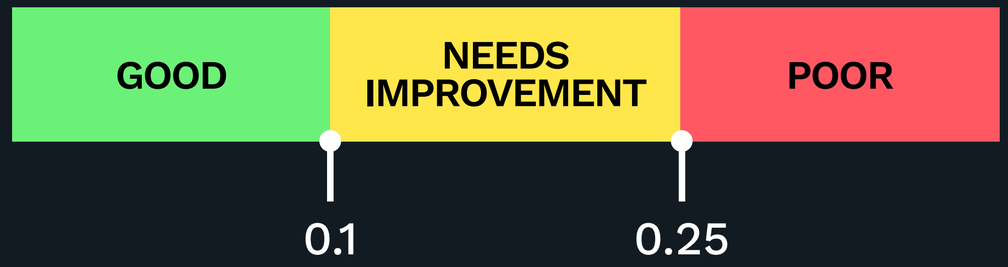
- Largest burst of layout shift scores for every unexpected layout shift that occurs during the entire lifespan of a page.
- A layout shift is only bad if the user isn't expecting it.

Cumulative Layout Shift (CLS)

- Measures visual stability
- Ensures the page is *delightful*
- Measure in lab and field

CLS

Cumulative Layout Shift



Core Web Vitals

Field metrics that measure important aspects of real-world user experience

Core Web Vitals

Core Web Vitals

Loading: LCP

Interactivity:
FCP

Visual Stability:
CLS

Performance Budget

Performance Budget

Resource Quantity

Metrics

Lighthouse

Resource Quantity

Performance Budget

Resource Quantity

- The size/number of images
- The number of web fonts
- Weight (KB) of JavaScript
- Number of HTTP requests

Metrics

Performance Budget

Metrics

- Single point in time
- Focus on Core Web Vitals
- Browser APIs
- Lighthouse

Lighthouse Score

Performance Budget

Lighthouse Score

- Performance score
- Best practices score
- SEO score

Steps

Performance Budget

- [x] Establish a baseline
- [x] Add budgets to build process
- [x] Track and monitor

Establish a baseline

Performance Budget

- ✅ First load < 5s TTI
- ✅ Second load < 2s TTI
- ✅ < 170 KB critical-path resources (HTML, CSS, JS, and data)

https://infrequently.org/2017/10/can-you-afford-it-real-world-web-performance-budgets/

Build Process Tools

Performance Budget

- [x] Webpack Performance
- [x] Lighthouse CI
- [x] bundlesize

What if I exceed the budget?

— Developer

Exceeding Budget

Performance Budget

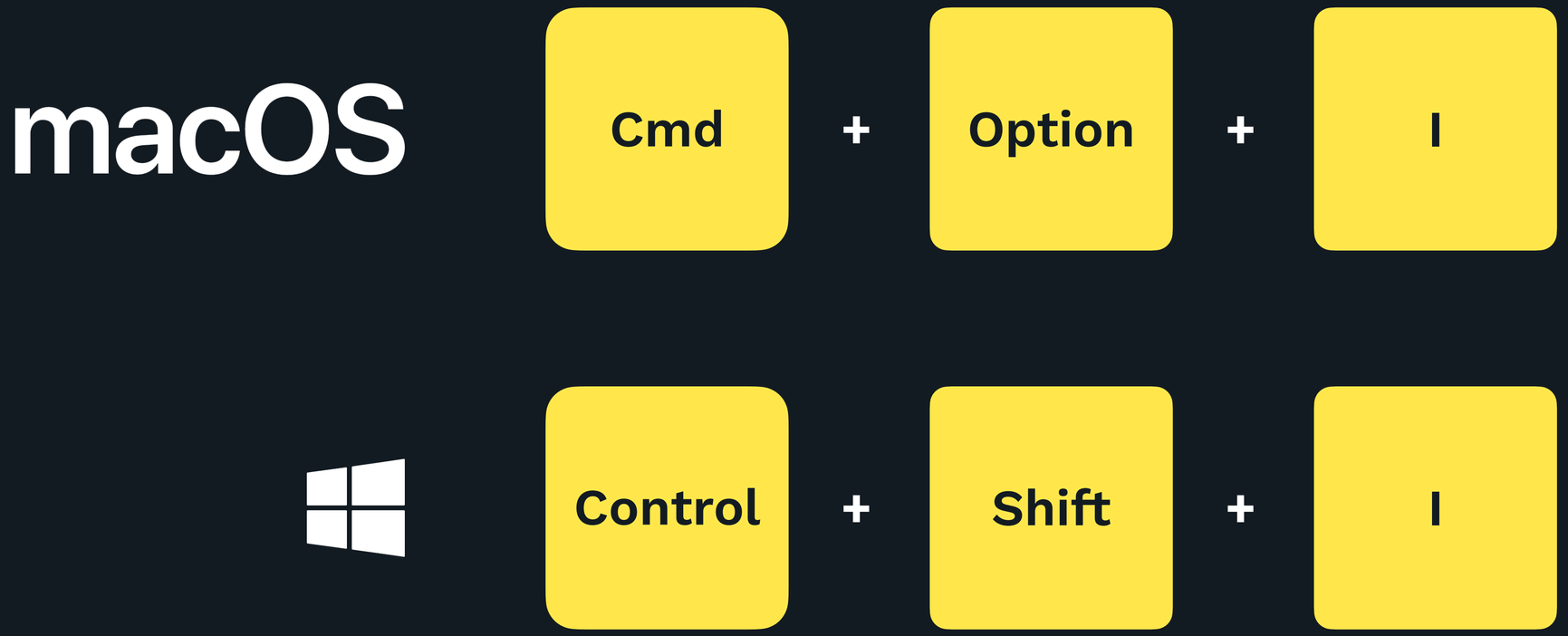
- [x] Optimize existing features or assets
- [x] Remove existing features or assets
- [x] Don’t add a new feature or asset

Exceeding Budget

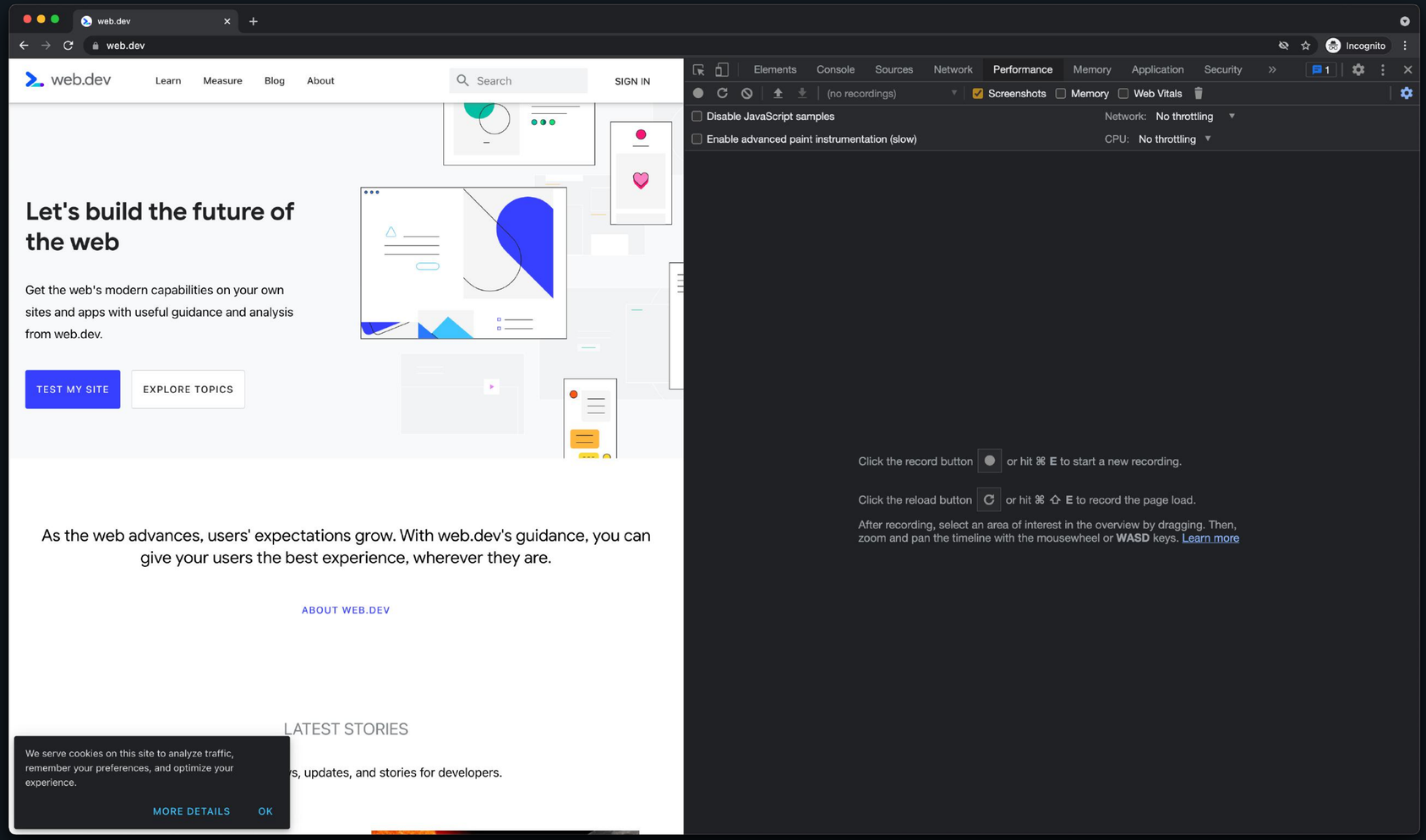
Performance Budget

- [x] Code-splitting
- [x] Service worker caching
- [x] HTTP/2 Push

Chrome DevTools


macOS
Cmd
+
Option
+
I
Control
+
Shift
+
I

DevTools


web.dev
web.dev
Incognito
web.dev
Learn
Measure
Blog
About
Search
SIGN IN
Let's build the future of the web
Get the web's modern capabilities on your own sites and apps with useful guidance and analysis from web.dev.
TEST MY SITE
EXPLORE TOPICS
As the web advances, users' expectations grow. With web.dev's guidance, you can give your users the best experience, wherever they are.
ABOUT WEB.DEV
LATEST STORIES
s, updates, and stories for developers.
We serve cookies on this site to analyze traffic, remember your preferences, and optimize your experience.
MORE DETAILS
OK
Elements
Console
Sources
Network
Performance
Memory
Application
Security
(no recordings)
Screenshots
Memory
Web Vitals
Disable JavaScript samples
Network: No throttling
Enable advanced paint instrumentation (slow)
CPU: No throttling
Click the record button or hit ⌘ E to start a new recording.
Click the reload button or hit ⌘ ⇧ E to record the page load.
After recording, select an area of interest in the overview by dragging. Then, zoom and pan the timeline with the mousewheel or WASD keys. Learn more

DevTools Fun Facts

- [x] Built with TypeScript
- [x] 150,000 lines of code
- [x] Over ten years old
- [x] Console loggers welcome 🤗

Panels we are NOT going to use today

- [x] Device mode
- [x] Element panel
- [x] Console panel
- [x] And more

Panels focused on performance

- Lighthouse panel
- Rendering panel
- Performance Monitor panel
- Performance panel
- Memory panel
- Coverage panel
- Layers panel

DevTools: Lighthouse

Lighthouse

Open-source

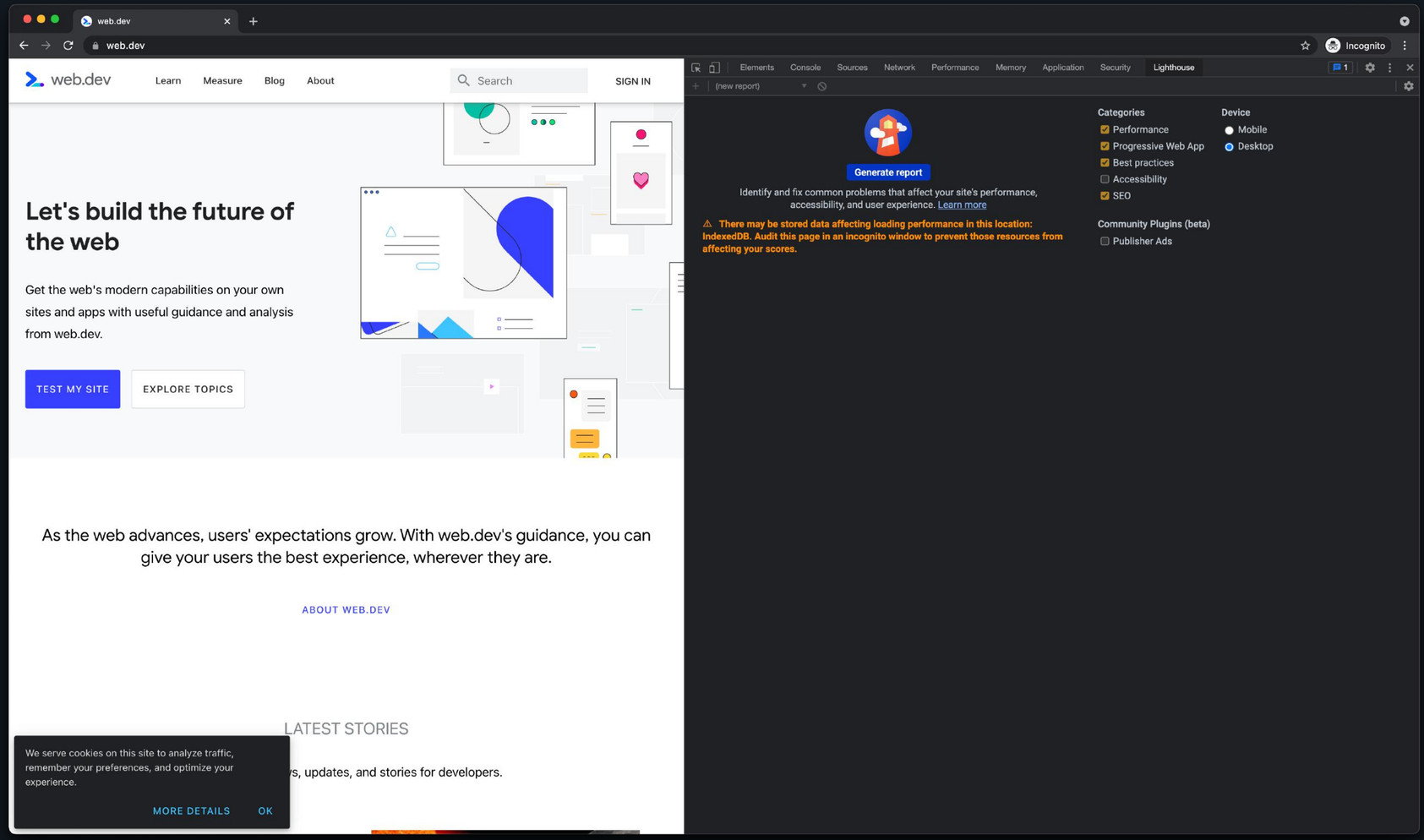
Automated tool

Improve quality

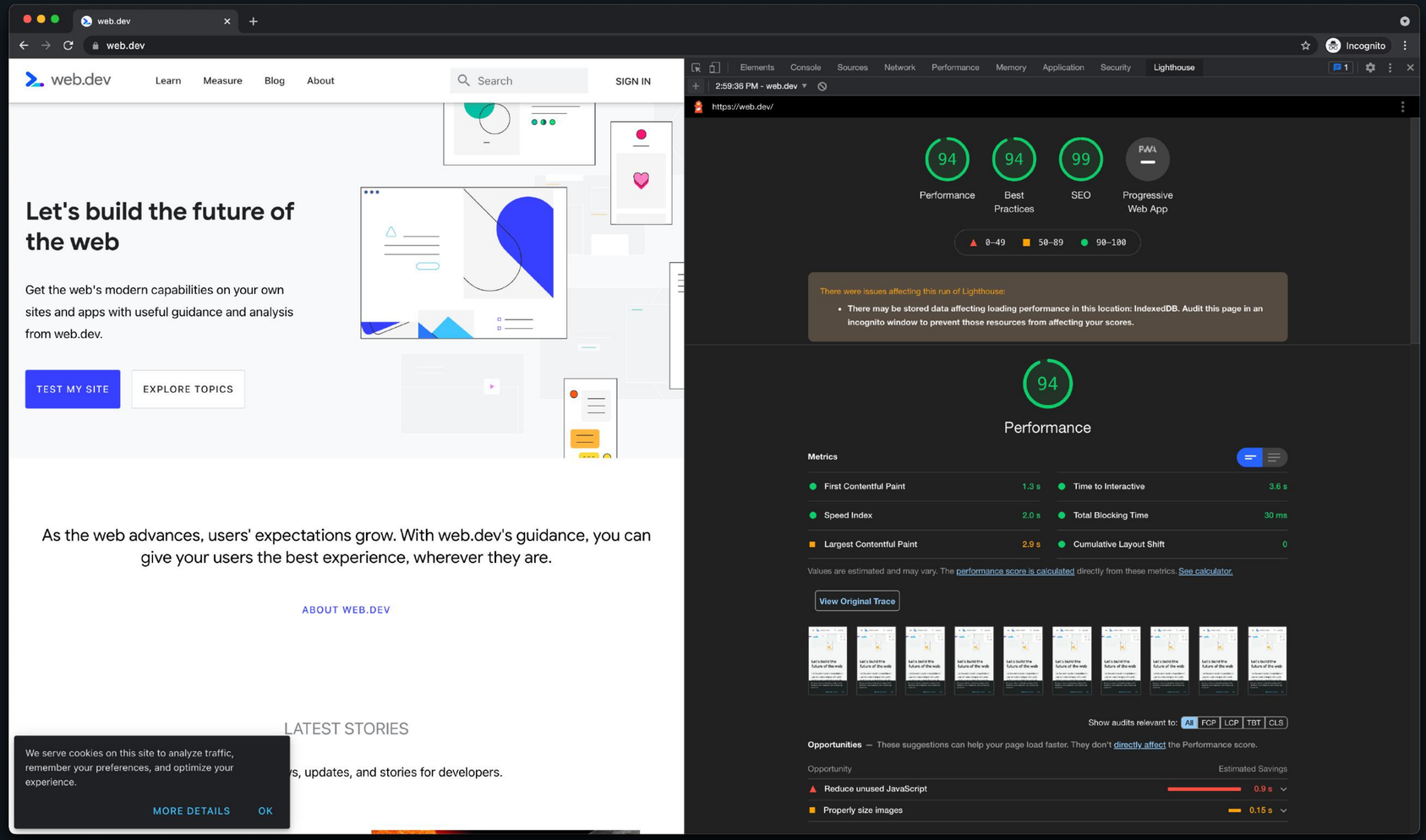
Lighthouse

- [x] Chrome DevTools
- [x] Node
- [x] Command line
- [x] PageSpeed Insights

Lighthouse


web.dev
web.dev
Incognito
web.dev
Learn
Measure
Blog
About
Search
SIGN IN
Let's build the future of the web
Get the web's modern capabilities on your own sites and apps with useful guidance and analysis from web.dev.
TEST MY SITE
EXPLORE TOPICS
As the web advances, users' expectations grow. With web.dev's guidance, you can give your users the best experience, wherever they are.
ABOUT WEB.DEV
LATEST STORIES
s, updates, and stories for developers.
We serve cookies on this site to analyze traffic, remember your preferences, and optimize your experience.
MORE DETAILS
OK
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
(new report)
Generate report
Identify and fix common problems that affect your site's performance, accessibility, and user experience. Learn more
There may be stored data affecting loading performance in this location: IndexedDB. Audit this page in an incognito window to prevent those resources from affecting your scores.
Categories
Performance
Progressive Web App
Best practices
Accessibility
SEO
Community Plugins (beta)
Publisher Ads
Device
Mobile
Desktop

Lighthouse


web.dev
web.dev
Incognito
web.dev
Learn
Measure
Blog
About
Search
SIGN IN
Let's build the future of the web
Get the web's modern capabilities on your own sites and apps with useful guidance and analysis from web.dev.
TEST MY SITE
EXPLORE TOPICS
As the web advances, users' expectations grow. With web.dev's guidance, you can give your users the best experience, wherever they are.
ABOUT WEB.DEV
LATEST STORIES
s, updates, and stories for developers.
We serve cookies on this site to analyze traffic, remember your preferences, and optimize your experience.
MORE DETAILS
OK
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
2:59:36 PM - web.dev
https://web.dev/
94
Performance
94
Best Practices
99
SEO
PWA
Progressive Web App
0–49
50–89
90–100
There were issues affecting this run of Lighthouse:
There may be stored data affecting loading performance in this location: IndexedDB. Audit this page in an incognito window to prevent those resources from affecting your scores.
94
Performance
Metrics
First Contentful Paint
1.3 s
Time to Interactive
3.6 s
Speed Index
2.0 s
Total Blocking Time
30 ms
Largest Contentful Paint
2.9 s
Cumulative Layout Shift
0
Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.
View Original Trace
Show audits relevant to:
All
FCP
LCP
TBT
CLS
Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.
Opportunity
Estimated Savings
Reduce unused JavaScript
0.9 s
Properly size images
0.15 s

Lighthouse

- [x] Lab measurements
- [x] Metrics
- [x] Opportunities
- [x] Diagnostics

Lighthouse: Performance Budget

```
→ npm install -g lighthouse
```

Lighthouse: Performance Budget

```
→ touch budget.json
```

Lighthouse: Performance Budget

```
[
  {
    "path": "/*",
    "resourceCounts": [
      {
        "resourceType": "third-party",
        "budget": 10
      }
    ]
  }
]
```

Lighthouse: Performance Budget

```
[
  {
    "path": "/*",
    "timings": [
      {
        "metric": "interactive",
        "budget": 3000
      },
      {
        "metric": "first-meaningful-paint",
        "budget": 1000
      }
    ],
  }
]
```

Lighthouse: Performance Budget

```
[
  {
    "path": "/*",
    "resourceSizes": [
      {
        "resourceType": "script",
        "budget": 125
      },
      {
        "resourceType": "total",
        "budget": 300
      }
    ]
  }
]
```

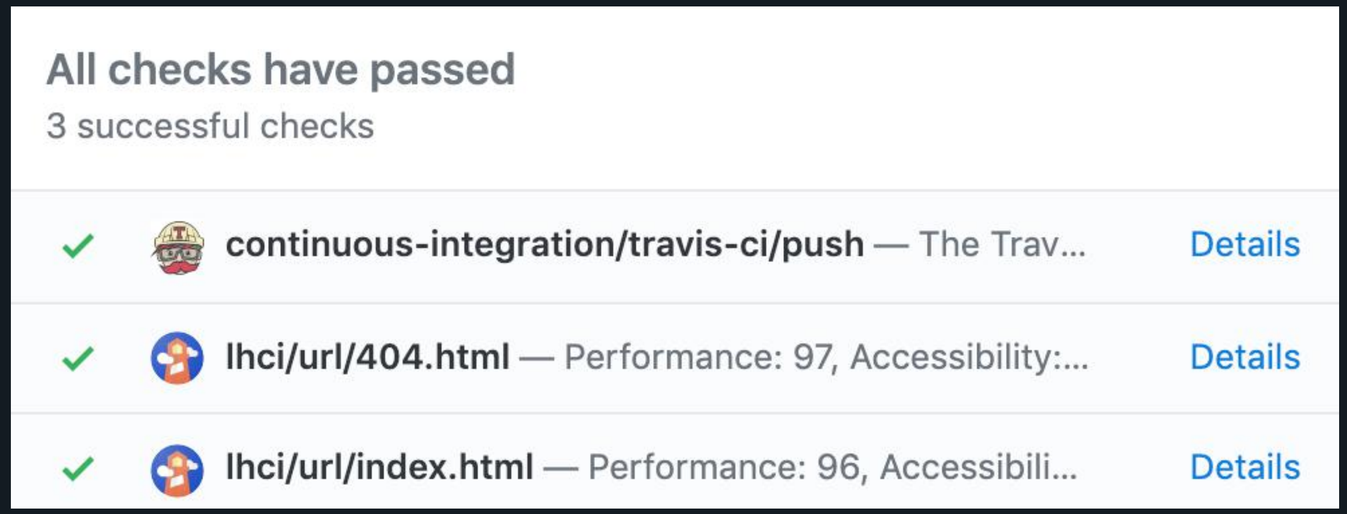
Lighthouse: Performance Budget

```
➜ lighthouse https://localhost:8080 -budget-path=./budget.json
```

Lighthouse Performance Budget

1. Open the **01 Lighthouse** challenge in Chrome.
2. Use the Lighthouse panel in Chrome DevTools to generate a report, and note the current FCP and TTI scores.
3. Use the generated report to set the **`first-contentful-paint`** and **`interactive`** performance metric budgets in the timings array to the current values.
4. Execute the **`npm run lh`** command to run lighthouse CLI using the budget file.
5. Note the “Budgets” section and verify that the current budgets pass.
6. Update the budget.json file with lower values, run lighthouse again, and note that the budgets fail.

Lighthouse CI



https://github.com/GoogleChrome/lighthouse/blob/master/docs/performance-budgets.md

Lighthouse CI

```
→ npm i @lhci/cli
```

Lighthouse CI

```
→ lhci autorun
```

Lighthouse CI

```
→ touch lighthouserc.js
```

Lighthouse CI

```
module.exports = {
  ci: {
    collect: {
      staticDistDir: './',
    },
    upload: {
      target: 'temporary-public-storage',
    },
  },
};
```

Lighthouse CI

```
module.exports = {
  ci: {
    collect: {
      staticDistDir: './',
    },
    upload: {
      target: 'temporary-public-storage',
    },
    assert: {
      preset: 'lighthouse:recommended',
    },
  },
};
```

DevTools: Rendering

DevTools: Rendering

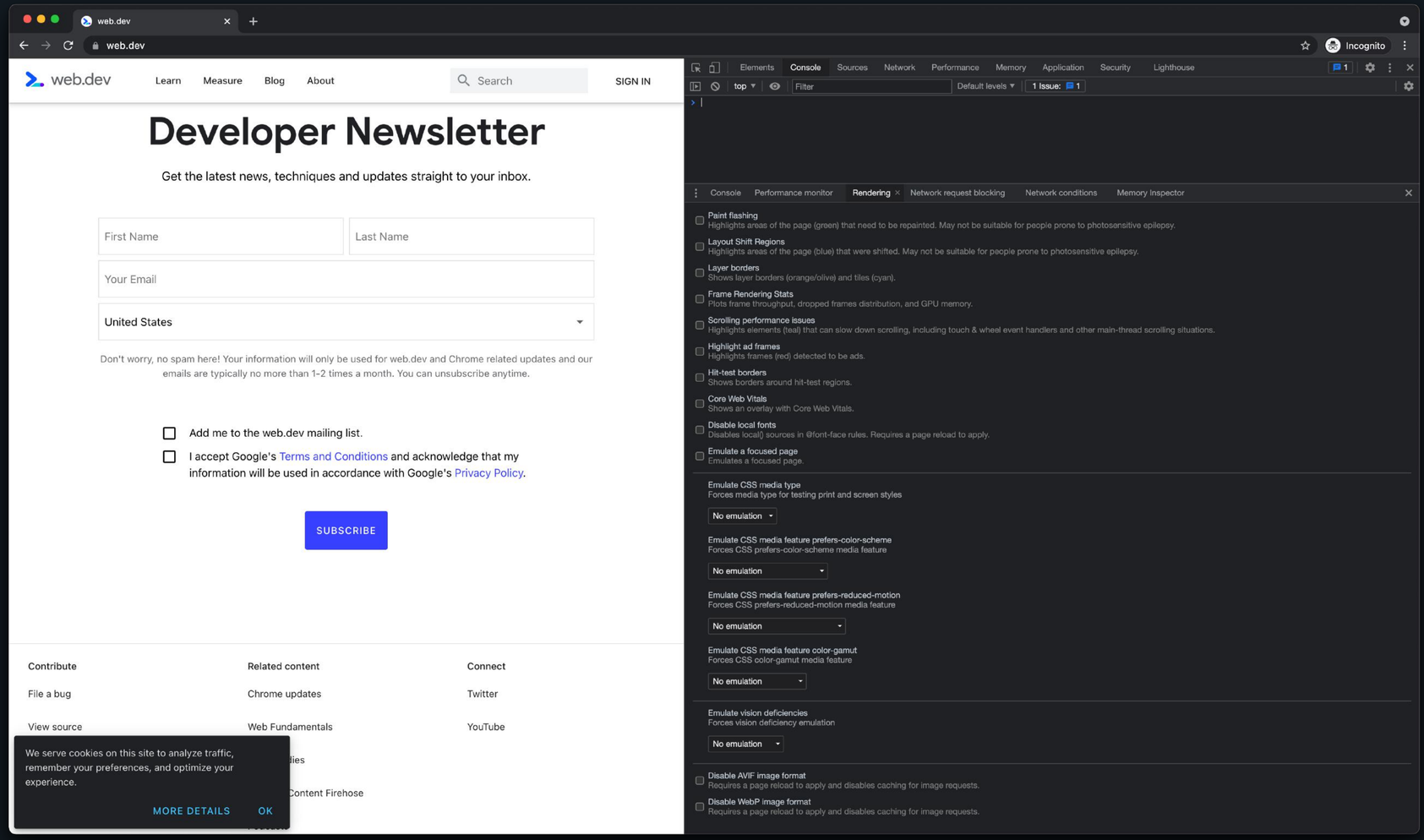
Layout Shift Regions

Layer Borders

Frame Rendering Stats

Scrolling Performance Issues

DevTools: Rendering



How does the browser render?

- Download HTML, parse it, and generate **DOM tree**
- Process styling and generate **render tree**
- Generate **layers** and paint each layer into bitmaps
- Upload bitmaps to GPU as textures
- Composite layers together

This all happens before frame 1

So, what are these layers?

DOM Tree

- Tree of **`Node`** objects
- Every element (and it’s inner text) is a **`Node`** object
- Top level is the **`Document`** node

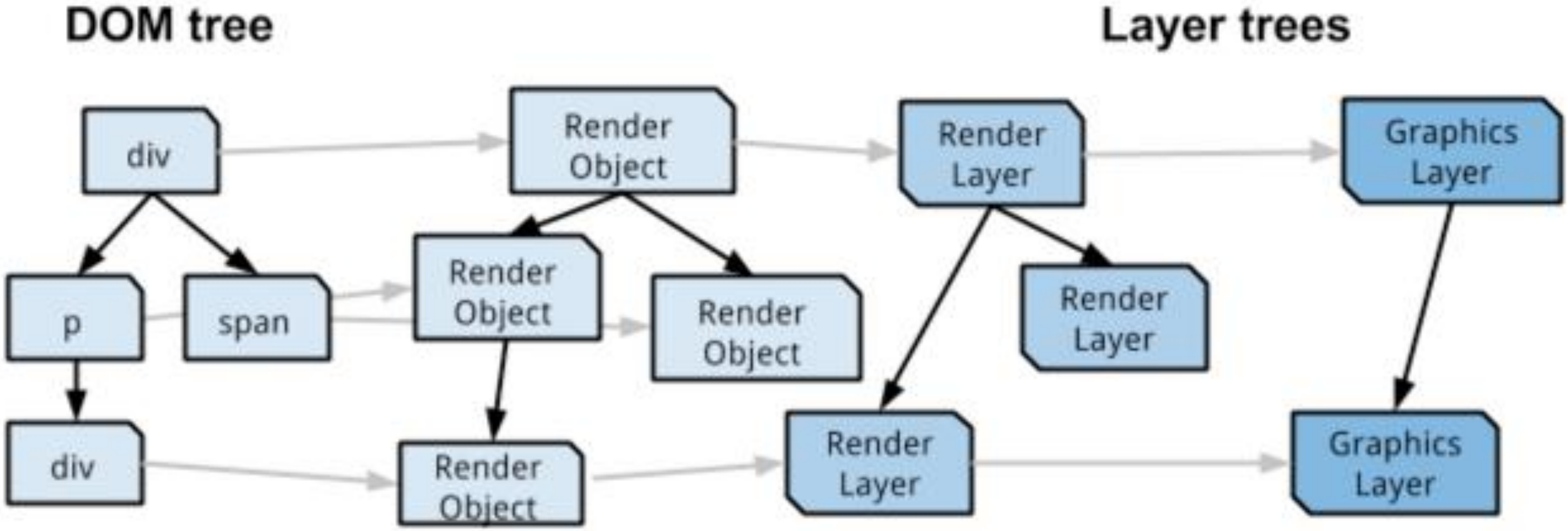
RenderObject

- Each **Node** has a corresponding **RenderObject**
- Composed into the render tree
- Contains paint instructions

RenderLayer and GraphicsLayer

- Each **RenderGroup** is associated with a **RenderLayer**
- **RenderLayer** renders subtrees of the DOM
- **GraphicsLayer** compositing layers that leverage the GPU for parallelism

Compositing of Layers



When is a new `GraphicsLayer` created?

- [x] Layer has 3D or perspective transform CSS properties
- [x] Layer is used by <video> element using accelerated video decoding
- [x] Layer is used by a <canvas> element with a 3D context or accelerated 2D context
- [x] Layer uses a CSS animation for its opacity or uses an animated webkit transform

When is a new `GraphicsLayer` created?

- [x] Layer uses accelerated CSS filters
- [x] Layer has a descendant that is a compositing layer
- [x] Layer has a sibling with a lower z-index which has a compositing layer

A `GraphicsLayer` Cost

- ✅ Memory
- ✅ CPU calculations

Compositor

- Composes layers
- Paints bitmaps to the screen

Compositing is efficient *without* repaints or layout shifts

Repaint

Updating the rendered results of a recalculated render tree, caused by:

- ✅ Changes in geometric properties of a node
- ✅ Changes in style (e.g. background color)

Layout Shift

Entire or partial recalculation of render tree, caused by:

- DOM element position change
- DOM element dimension change
- Inserting or removing DOM elements
- Animations that trigger layout

Layer compositing has implications for rendering performance of your application

Layout Shift Regions

Layout Shift Regions

- Layout Shifts reduce *delightfulness*
- Identify and reduce layout shifts
- **Enable Layout Shifts** Region
- Highlights layout shifts in blue

Layout Shift Regions

1. Open the **02 Rendering** challenge in Chrome.
2. Press **Esc** (or navigate to the ellipsis menu and choose Show console drawer)
3. In the DevTools console drawer, click on the **Rendering** tab, and check the **Layout Shift Regions** checkbox.
4. Reload the application and note the layout shift regions in blue.
5. Click the **Start** button and note the additional layout shift regions.

How do we fix layout shifts?

Element Position Change

Identify Layout Shifts

- Stylesheets are loaded late
- Stylesheets overwrite previously declared styles
- Animations
- Transitions

Element Dimension Change

Identify Layout Shifts

- Stylesheets are loaded late
- Stylesheets overwrite previously declared styles
- Images and frames without height and width that are loaded after slot is rendered
- Text blocks without height and width where font swaps after text is rendered

Inserting or Removing DOM Elements

Identify Layout Shifts

- Dynamic images or ads
- Inserting modals, banners, etc.
- Infinite scroll
- Loading content above existing content

Fix Layout Shifts

1. Execute **npm run build** and **npm run serve** for a production build.
2. Open the **02 Rendering** challenge in Chrome.
3. Throttle the network and enable the **Layout Shift Regions** in DevTools.
4. Create a new audit using Lighthouse and note the **CLS value**.
5. Open common.css and inline the web fonts.
6. Open index.html and remove the **`hidden`** attribute from the **`#warning`** element. Refactor the **`<img>`** elements to **`<div>`** specifying a background-image and fixed height.
7. Run another audit using Lighthouse and note the reduction of the **CLS value**.

Layer Borders

Layer Borders

- Overlay of layer borders and tiles
- Easily identify layers
- Layers impact rendering performance

Layer Border Colors

- Composite layer borders are orange.
- Image layers are olive.
- High-res tile borders are cyan.

Composite Layers

- A **`GraphicLayer`** instance
- Each layer uses memory
- Each layer’s texture is uploaded to the GPU
- Mostly, important to rendering performance

Tile Borders

- Tiles are like sub-units of a layer.
- Tiles represent parts of a large layer that are uploaded to the GPU.
- Mostly, not important. ¯_(ツ)_/¯

Layer Borders

1. Open the **02 Rendering** challenge in Chrome.
2. Press **Esc** (or navigate to the ellipsis menu and choose Show console drawer).
3. In the DevTools console drawer, click on the **Rendering** tab, and check the **Layout borders** checkbox.
4. Note layer borders:
    1. CSS **`transition`** when hovering over a location card.
    2. CSS **`position: fixed`** banner at the bottom.
    3. CSS 3D **`transform`** applied to the text within the banner.

Frame Rendering Stats

Frame Rendering Stats

- FPS meter overlay
- Updates in real-time
- Optionally displays GPU memory usage

Frame Rendering Stats

1. Open the **02 Rendering** challenge in Chrome.
2. Press **Esc** (or navigate to the ellipsis menu and choose Show console drawer).
3. Click on the **Rendering** tab.
4. Check the **Frame Rendering Stats** checkbox.
5. Note the overlay and the frames per second.

Scrolling Performance Issues

Scrolling Performance Issues

- Highlight elements that may cause scrolling issues
- **`TouchEvent`**
- **`WheelEvent`**
- Non-fast scrolling rectangles

TouchEvent

- Main thread is blocked in case preventDefault() is invoked
- Avoid adding touch event unless necessary (e.g. a map)

TouchEvent

```
this.images = document.querySelectorAll('img');
this.images.forEach((img) => {
  img.addEventListener('touchmove', this.boundOnTouchMove);
});
```

TouchEvent

```
private onTouchMove(event: TouchEvent): void {
  // event.preventDefault();
  this.fibonacci(35);
  console.log('touchmove');
}
```

WheelEvent

- [x] Main thread is blocked in case preventDefault() is invoked
- [x] Capture mouse wheel event unnecessarily

WheelEvent

```
el.addEventListener('wheel', this.boundOnZoom);
```

WheelEvent

```
private onZoom(event: WheelEvent): void {
  event.preventDefault();
  this.scale += event.deltaY * -0.01;
  this.scale = Math.min(Math.max(0.125, this.scale), 4);
  this.el.style.transform = `scale(${this.scale})`;
}
```

Scrolling Performance Issues

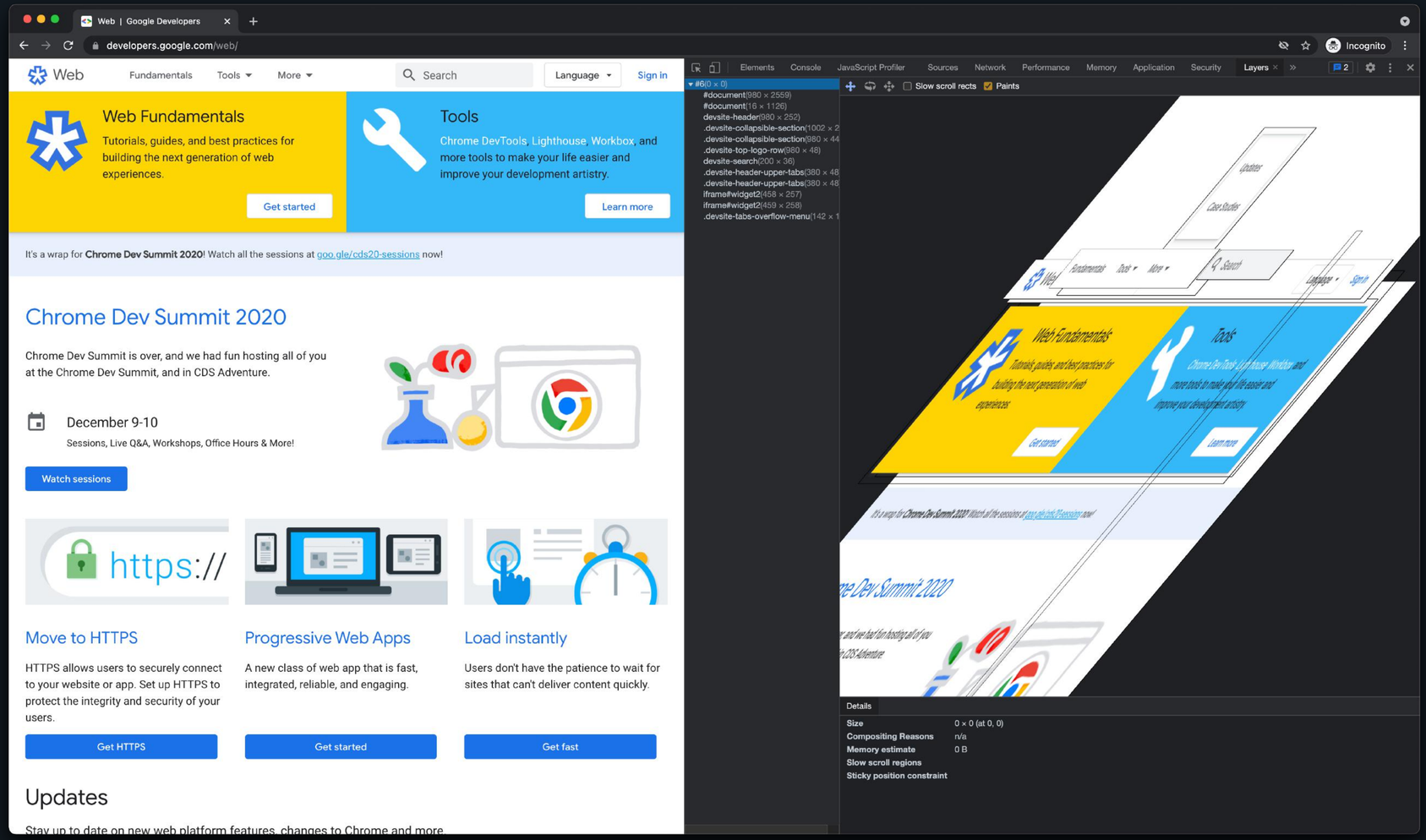
1. Open the **02 Rendering** challenge in Chrome.
2. Press **Esc** (or navigate to the ellipsis menu and choose Show console drawer).
3. In the DevTools console drawer, click on the **Rendering** tab, and check the **Scrolling performance issues** checkbox.
4. Add **`touchmove`** event handlers to all of the images in the document and invoke the **`preventDefault()`** method on the **`TouchEvent`** object.
5. Toggle to a mobile device in Devools, and note the highlighted scrolling performance issues.
6. BONUS: remove **`preventDefault()`** and instead calculate a length of 30 of the Fibonacci sequence (or another long-running task) and note lag.

Resources

- [x] GPU Accelerating Compositing in Chrome
- [x] Accelerated Rendering in Chrome
- [x] Debug Layout Shifts

DevTools: Layers

DevTools: Layers



Layer compositing has implications for rendering performance of your application

Compositing and Layers

- Use **`opacity`** and **`transform`** CSS for animations (not JS)
- Limit and reduce the number of layers
- Manually promote elements to a layer via **`will-change`**

will-change

```
.animate {
  will-change: transform;
}
```

DevTools: Layers

1. Open the **03 Rendering** challenge in Chrome.
2. Open the **Layers** panel in Chrome DevTools and check the **Paints** checkbox.
3. Click the “Let’s Go!” button.
4. Select the **`span#refresh`** layer and note the compositing reasons.
5. Open main.css and add CSS to promote the **#refresh** element to a new layer using the **`will-change`** property.

DevTools: Layers

1. Open the **03 Rendering** challenge in Chrome.
2. In the DevTools console drawer, click on the Search tab.
3. Search the source code for **`z-index`**.
4. How many layers do you expect the **`Dialog`** component to create?

DevTools: Layers

1. Open the **03 Rendering** challenge in Chrome.
2. In the DevTools console drawer, click on the Search tab.
3. Search the source code for **`z-index`**.
4. How many layers do you expect the **`Dialog`** component to create?
5. Open the **Layers** panel and inspect the layers.
6. Why is a new layer created for the **`<video>`** element?

Layout Instability API

Layout Instability API

- Measure and report layout shifts
- Includes layout shift **`value`**
- Includes layout shift **`startTime`**
- Includes up to 5 sources that substantially contributed to the layout shift

Layout Instability API

- DevTools is built upon API
- Uses **`PerformanceObserver`** to observe events
- The observe method accepts a **`PerformanceObserverInit`** dictionary
- Prevents blocking the main thread

PerformanceObserver

```
new(callback: PerformanceObserverCallback): PerformanceObserver;
```

PerformanceObserver

```
interface PerformanceObserver {
    disconnect(): void;
    observe(options?: PerformanceObserverInit): void;
    takeRecords(): PerformanceEntryList;
}
```

PerformanceObserverCallback

```
interface PerformanceObserverCallback {
 (entries: PerformanceObserverEntryList, observer: PerformanceObserver): void;
}
```

PerformanceObserverInit

```
interface PerformanceObserverInit {
    buffered?: boolean;
    entryTypes?: string[];
    type?: string;
}
```

PerformanceObserverInit

- **buffered** checks entry buffer for entries created before initialization of PerformanceObserver
- **type** is a string of the entry type to observe

Cumulative Layout Shift

```
let cls = 0;
new PerformanceObserver((entryList) => {})
  .observe({ type: 'layout-shift', buffered: true });
```

Cumulative Layout Shift

```
let cls = 0;
new PerformanceObserver((entryList) => {
  for (const entry of entryList.getEntries()) {
    // todo: ignore layout shifts within 500 ms of user input
  }
}).observe({ type: 'layout-shift', buffered: true });
```

Cumulative Layout Shift

```
let cls = 0;
new PerformanceObserver((entryList) => {
  for (const entry of entryList.getEntries()) {
    if (!entry.hadRecentInput) {
      cls += entry.value;
      console.log('Current CLS value:', cls, entry);
    }
  }
}).observe({ type: 'layout-shift', buffered: true });
```

LayoutShift

Warning: you may need to define the LayoutShift interface with TS

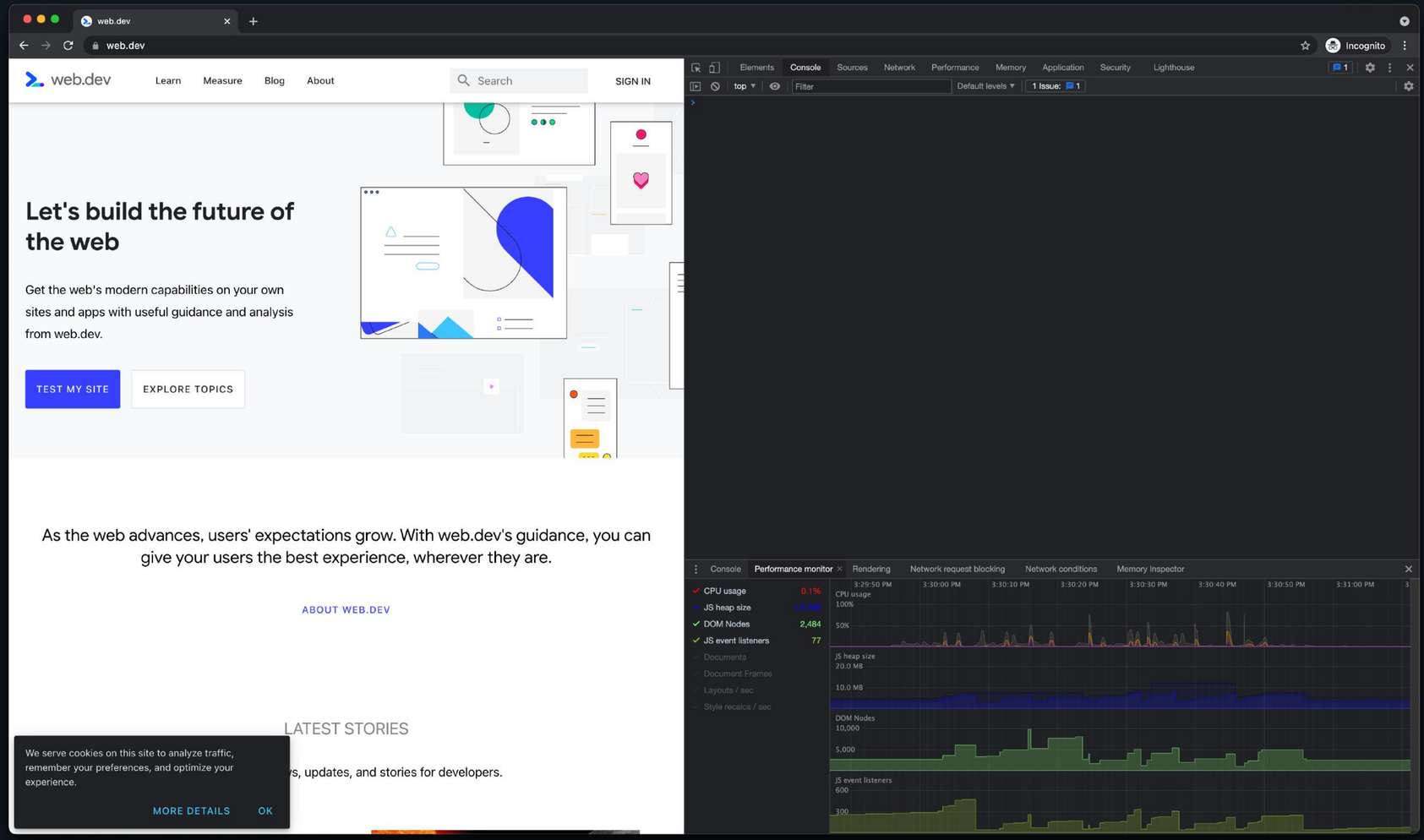
```
interface LayoutShift extends PerformanceEntry {
  hadRecentInput: boolean;
  value: number;
}
```

Instability API: CLS

1. Open the **challenges/04-instability-api/main.ts** file.
2. Instantiate a **`new PerformanceObserver()`** and invoke the **`observer()`** method, specifying the options object: **`{ type: 'layout-shift', buffered: true }`**
3. Within the callback function, iterate over the **`entryList.getEntries()`** collection. *Note: you may need to cast the* ***`entry`*** *to* ***`any`*** *or define the* ***`LayoutShift`*** *interface.*
4. Check if the entry **`hadRecentValue`** is false.
5. Sum the cumulative layout shift using the entry's **`value`**.
6. Log out the cumulate layout shift.

DevTools: Performance Monitor

DevTools: Performance Monitor



Performance Monitor

- Real-time visualization of performance metrics
- Useful for identifying bottlenecks while using the app
- Metrics can be toggle on and off

Performance Monitor

- [x] CPU usage
- [x] JS heap size
- [x] DOM Nodes
- [x] JS Event Listeners

Performance Monitor

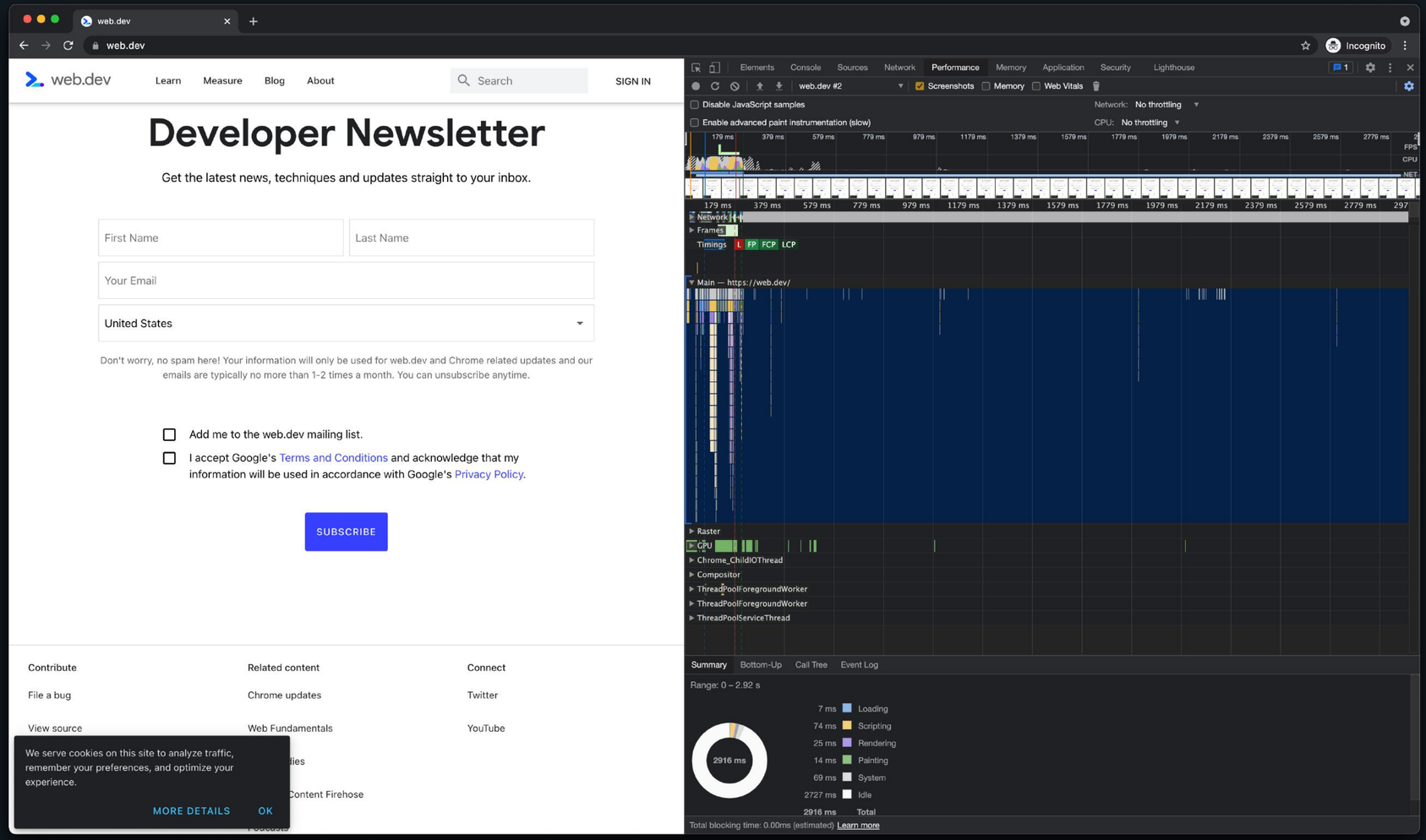
- [x] Documents
- [x] Document Frames (iframe)
- [x] Layouts per second
- [x] Style recalculations per second

Performance Monitor

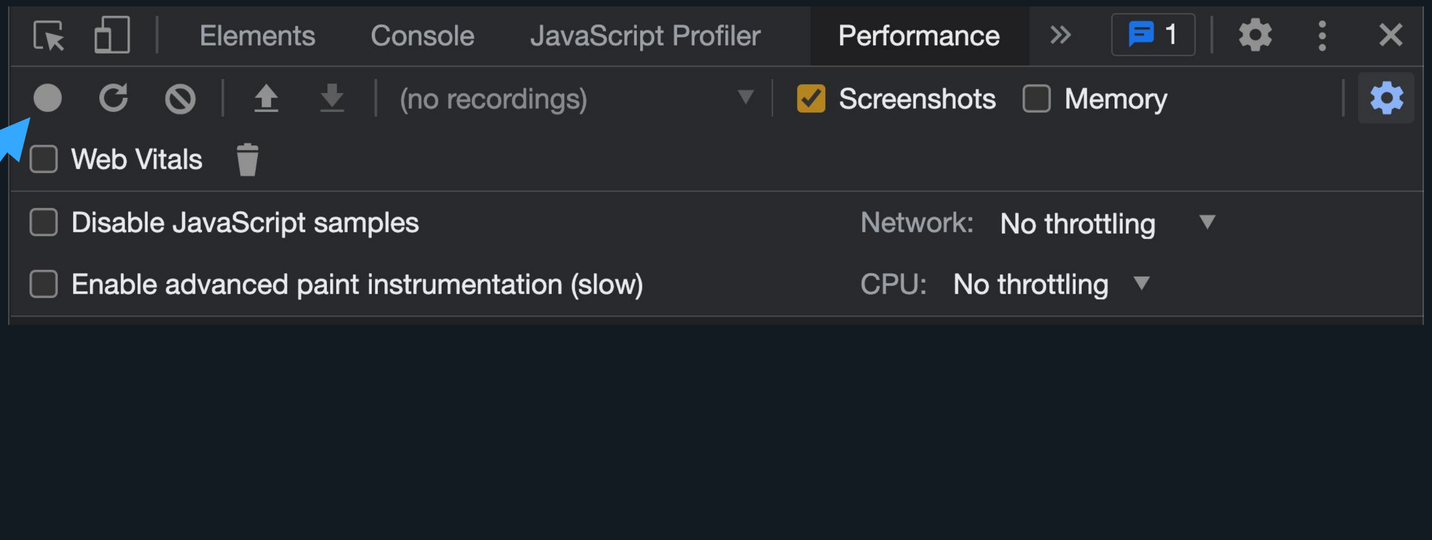
1. Open **05 Performance Monitor** challenge in Chrome.
2. Press **Esc** (or navigate to the ellipsis menu and choose Show console drawer)
3. In the DevTools console drawer, click on the **Performance monitor** tab.
4. Toggle on the **Layouts / sec** and **Style recalcs / sec** metrics.
5. Search for a resort by name or state and note:
    1. CPU Usage
    2. DOM Nodes
    3. JS Event Listeners
    4. Layouts / sec and Style recalcs / sec

DevTools: Performance

DevTools: Performance

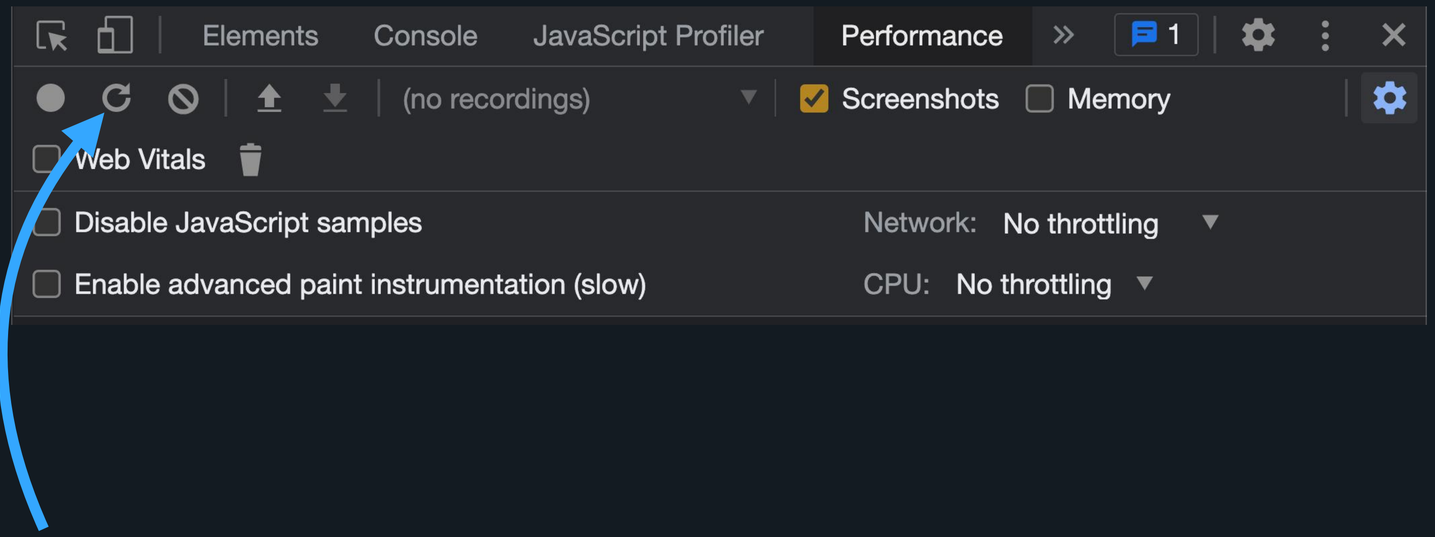


DevTools: Performance



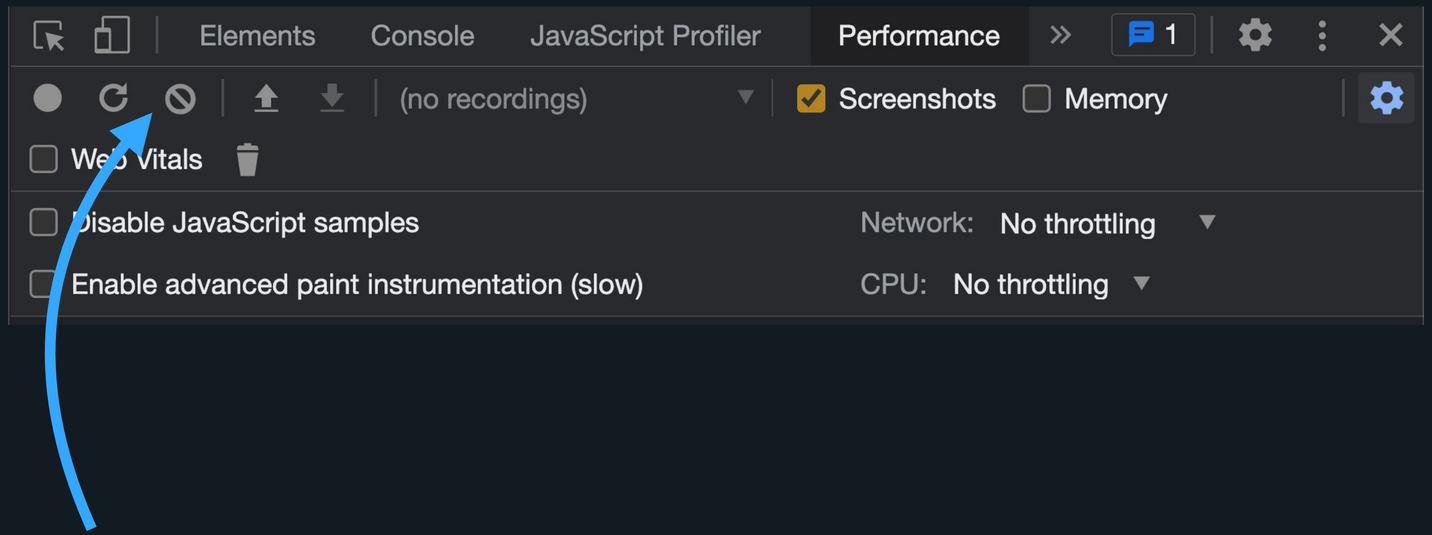
Start a new recording

DevTools: Performance



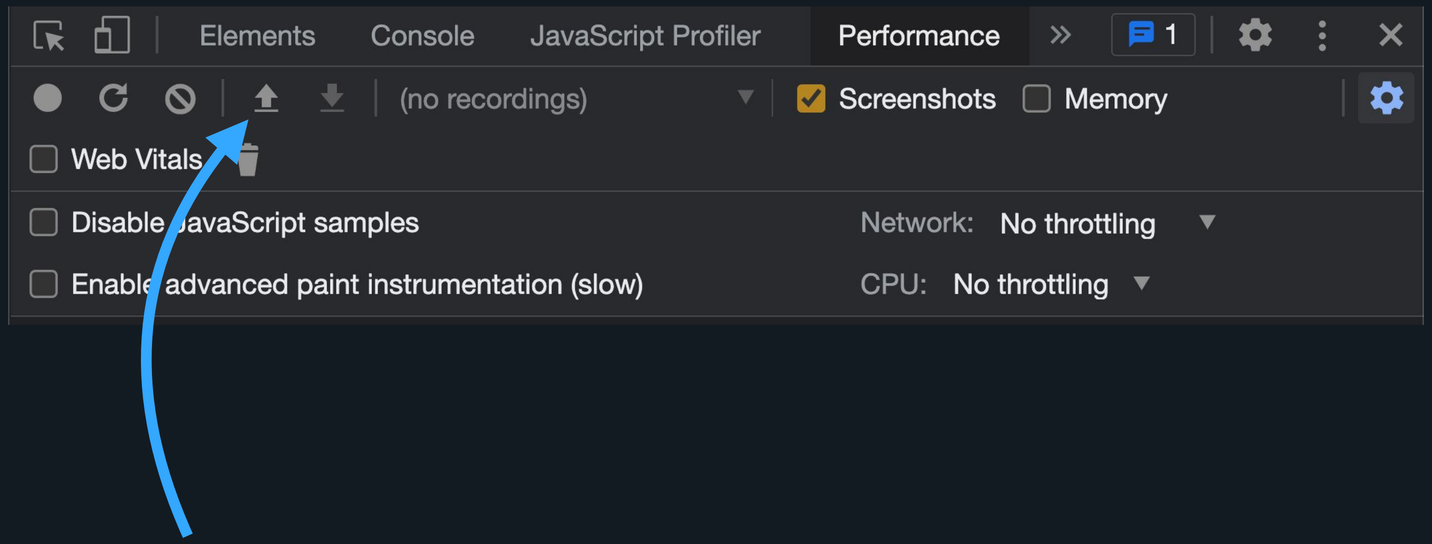
Reload page and start recording

DevTools: Performance



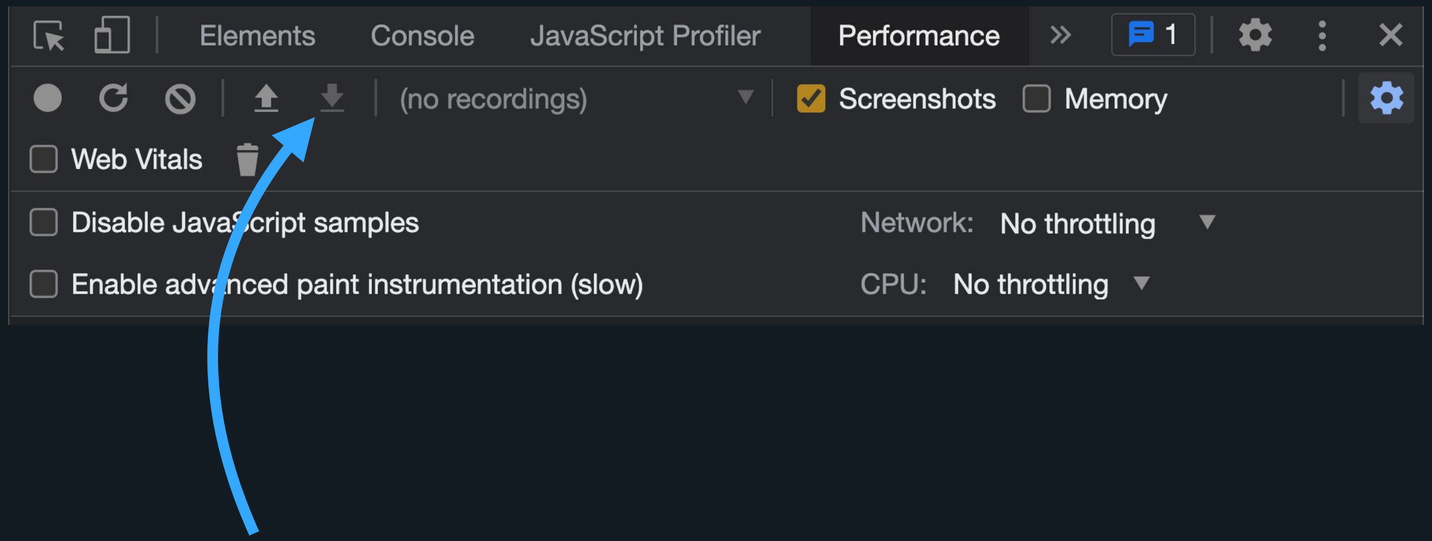
Clear (delete) the current recording

DevTools: Performance



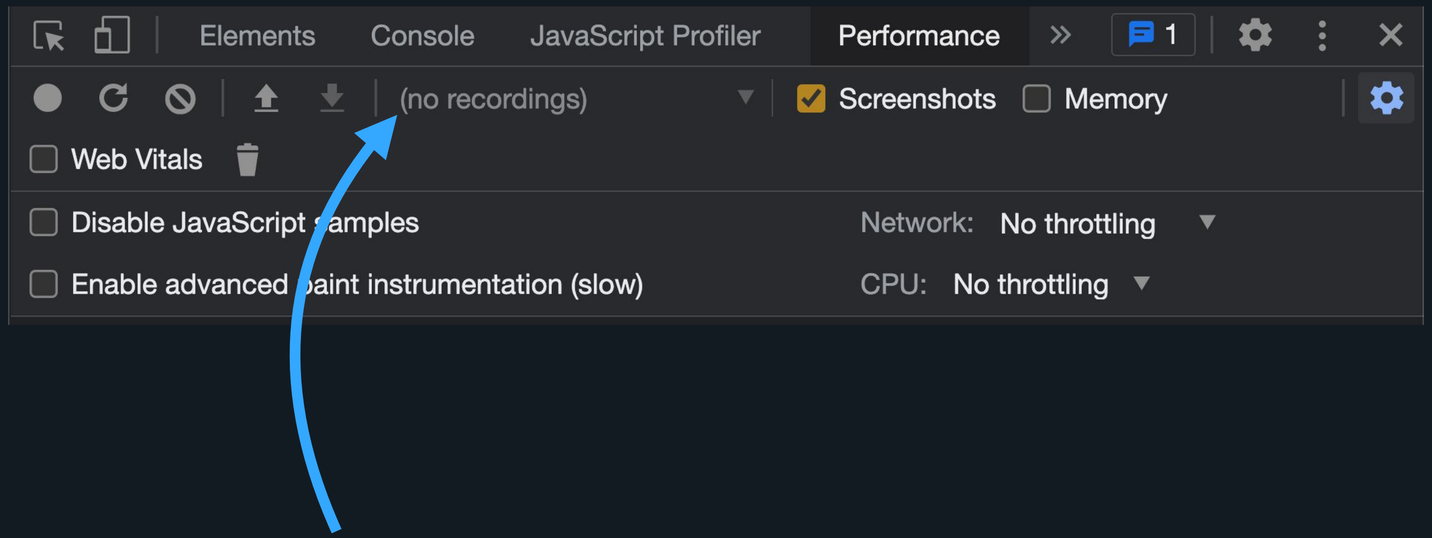
Upload a previous recording

DevTools: Performance



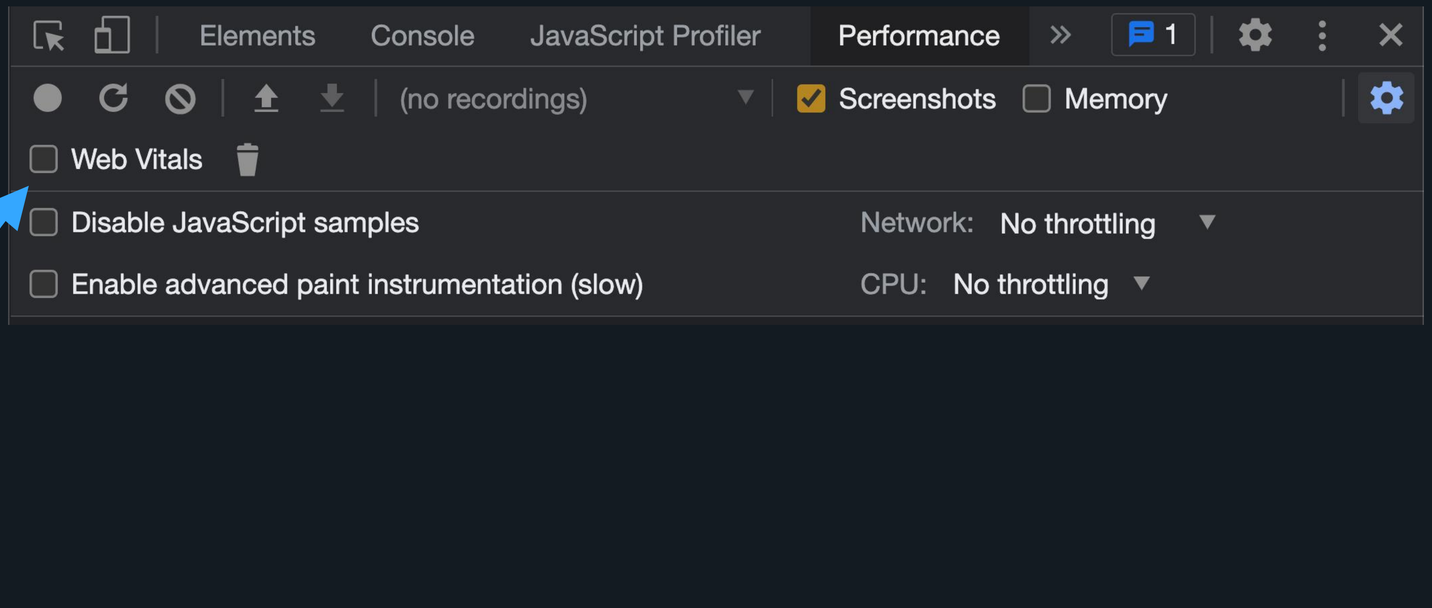
Download the current recording

DevTools: Performance



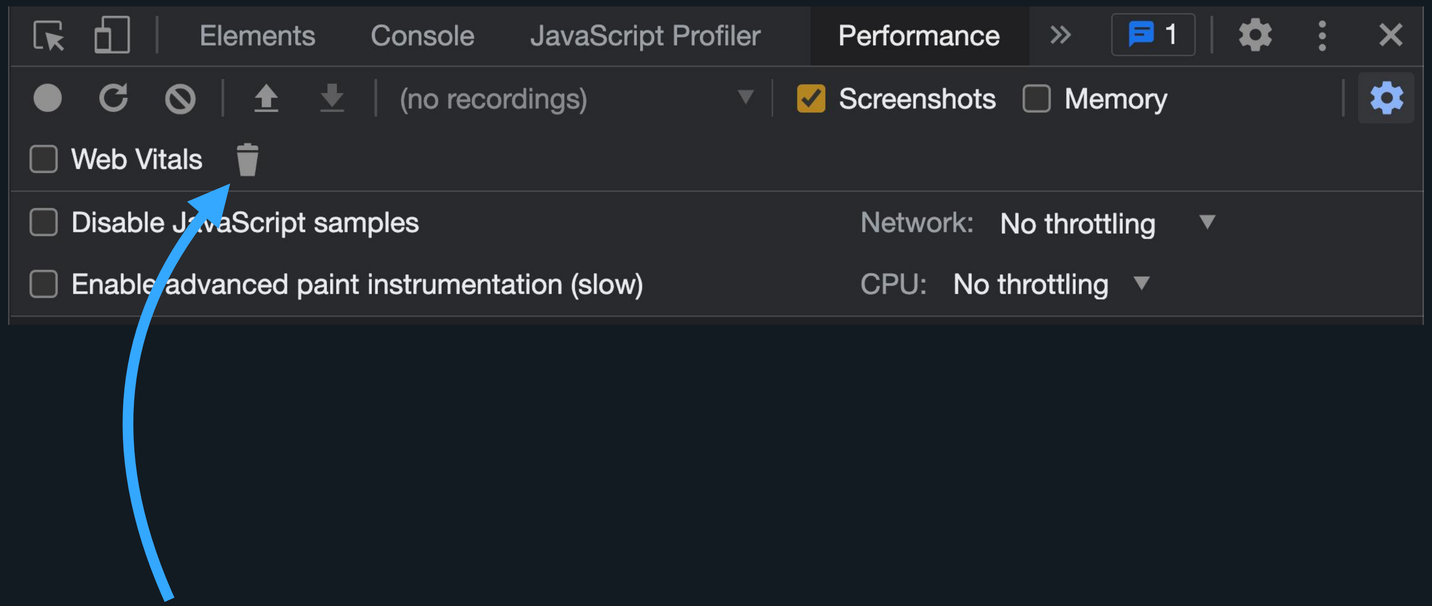
Select a previous recording

DevTools: Performance



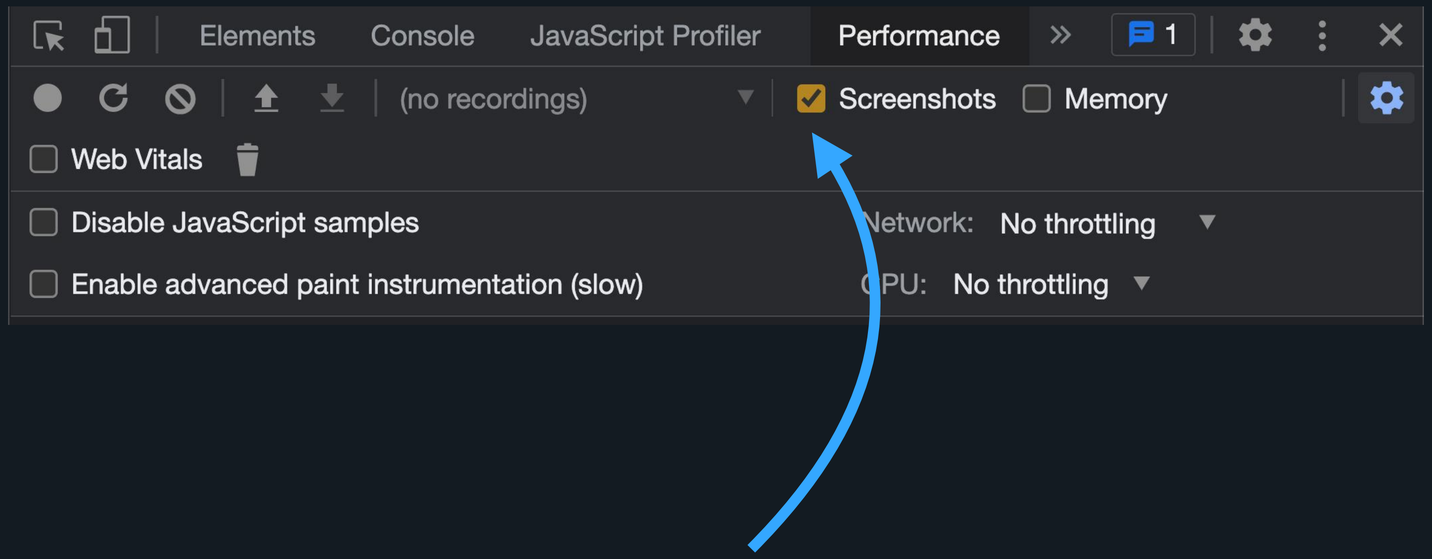
Display Web Vitals metrics

DevTools: Performance



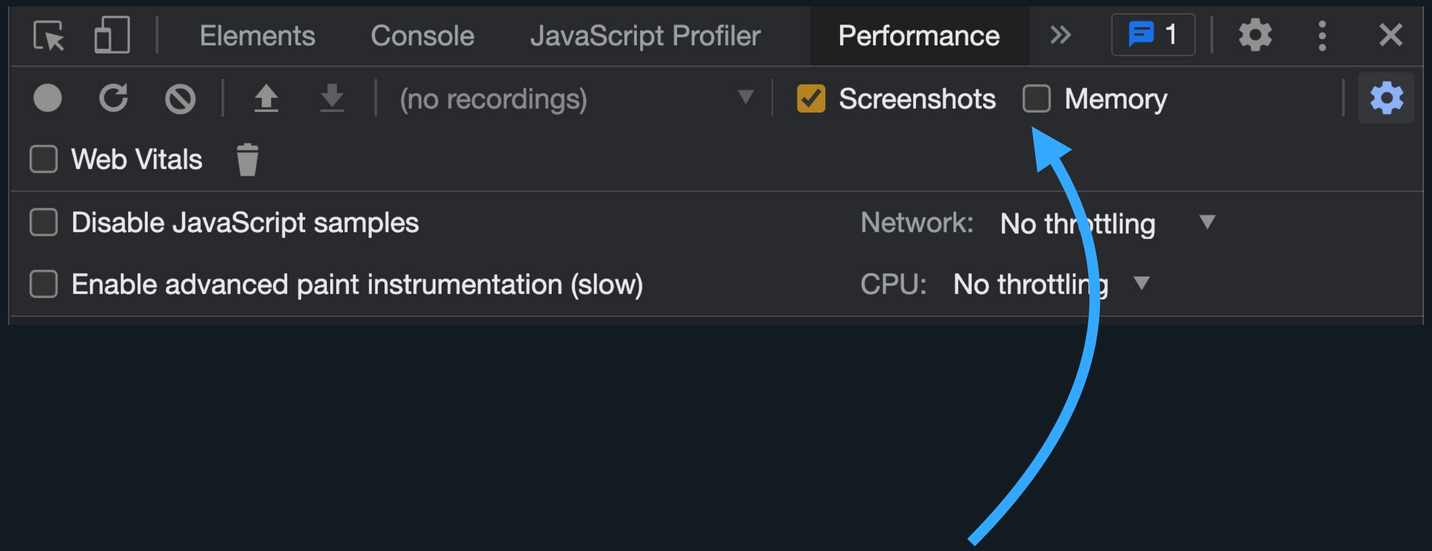
Force garbage collection

DevTools: Performance



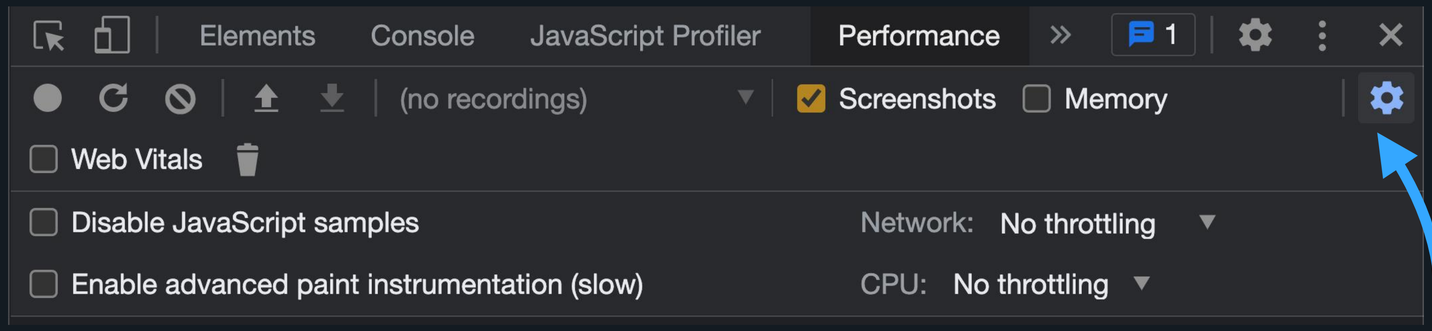
Take screenshots during the recording

DevTools: Performance



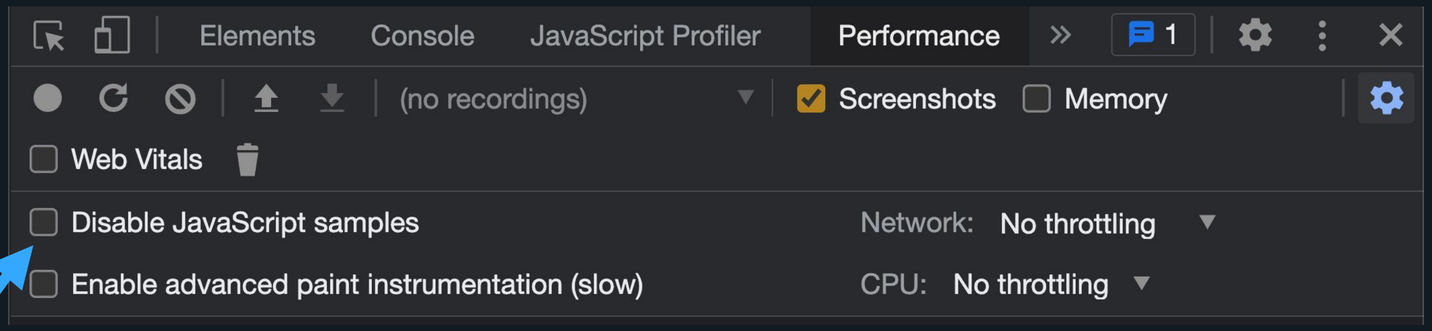
Record a timeline of the JS heap size

DevTools: Performance



Show more configuration options

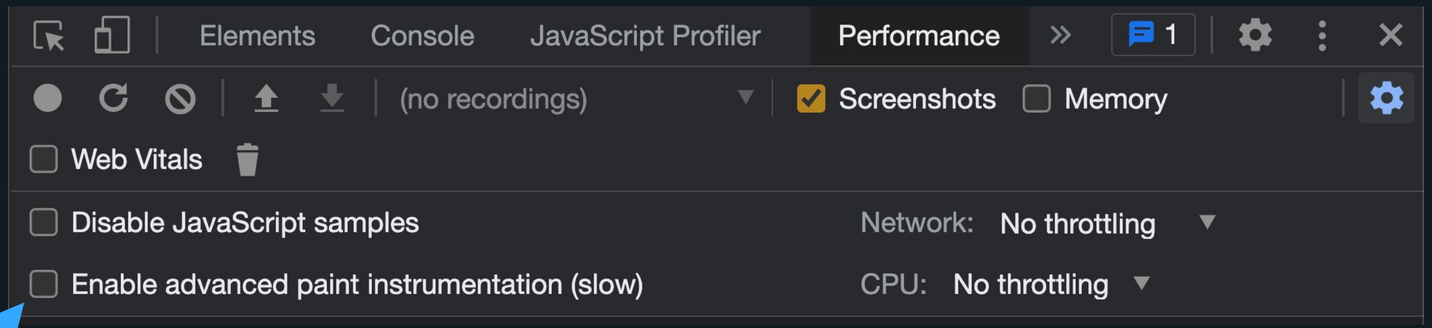
DevTools: Performance



Do not include JS code in

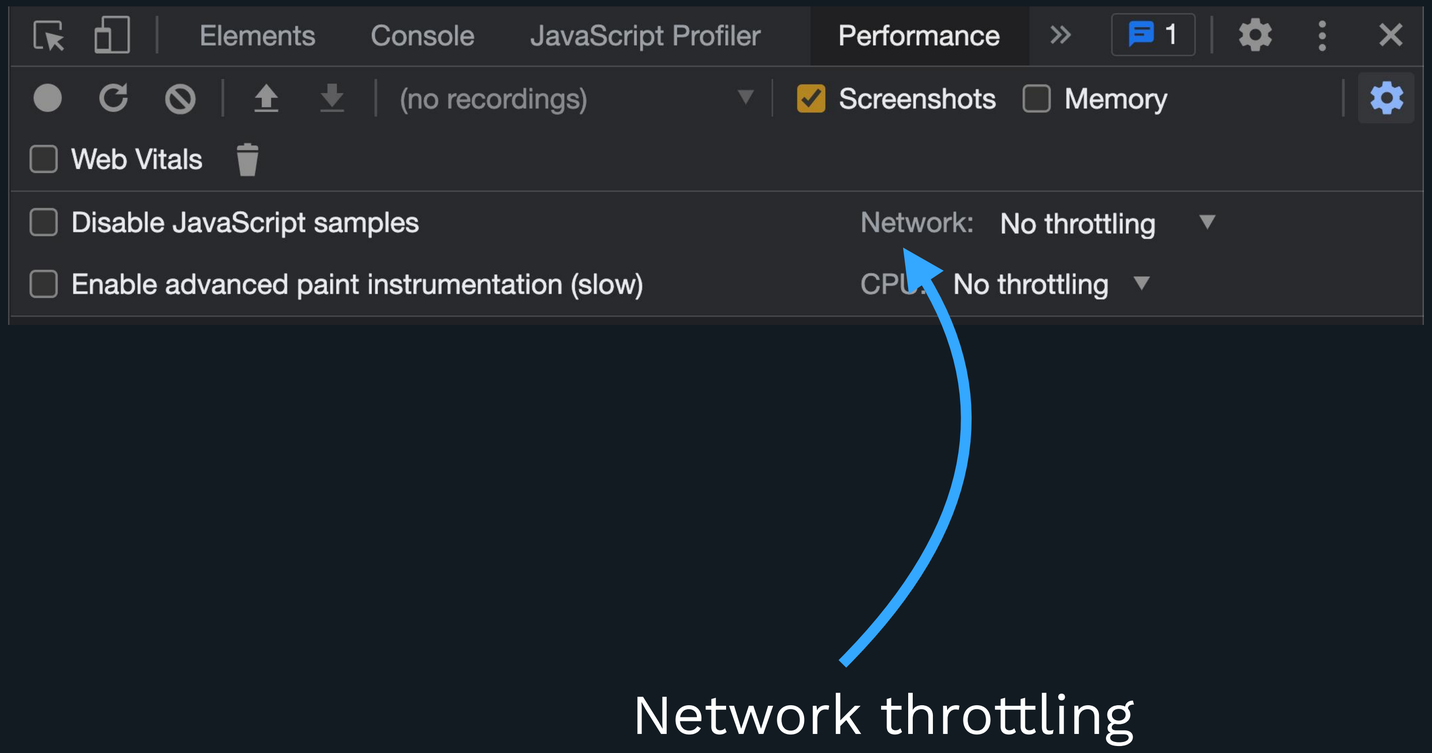
the flame graphs of the main thread

DevTools: Performance

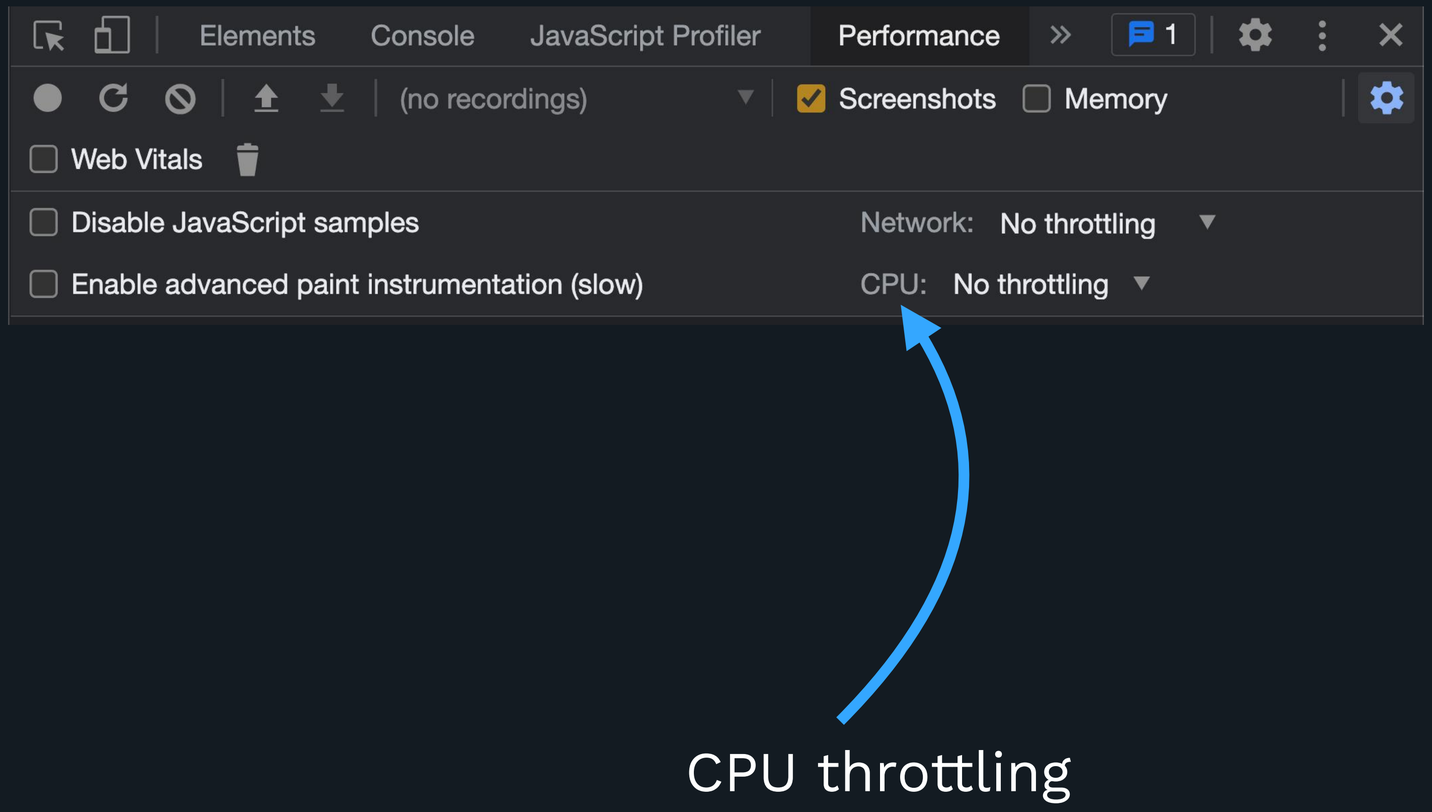


Includes layer information (similar to Layers panel) for frames, and paint profiling information during paints.

DevTools: Performance



DevTools: Performance



Performance

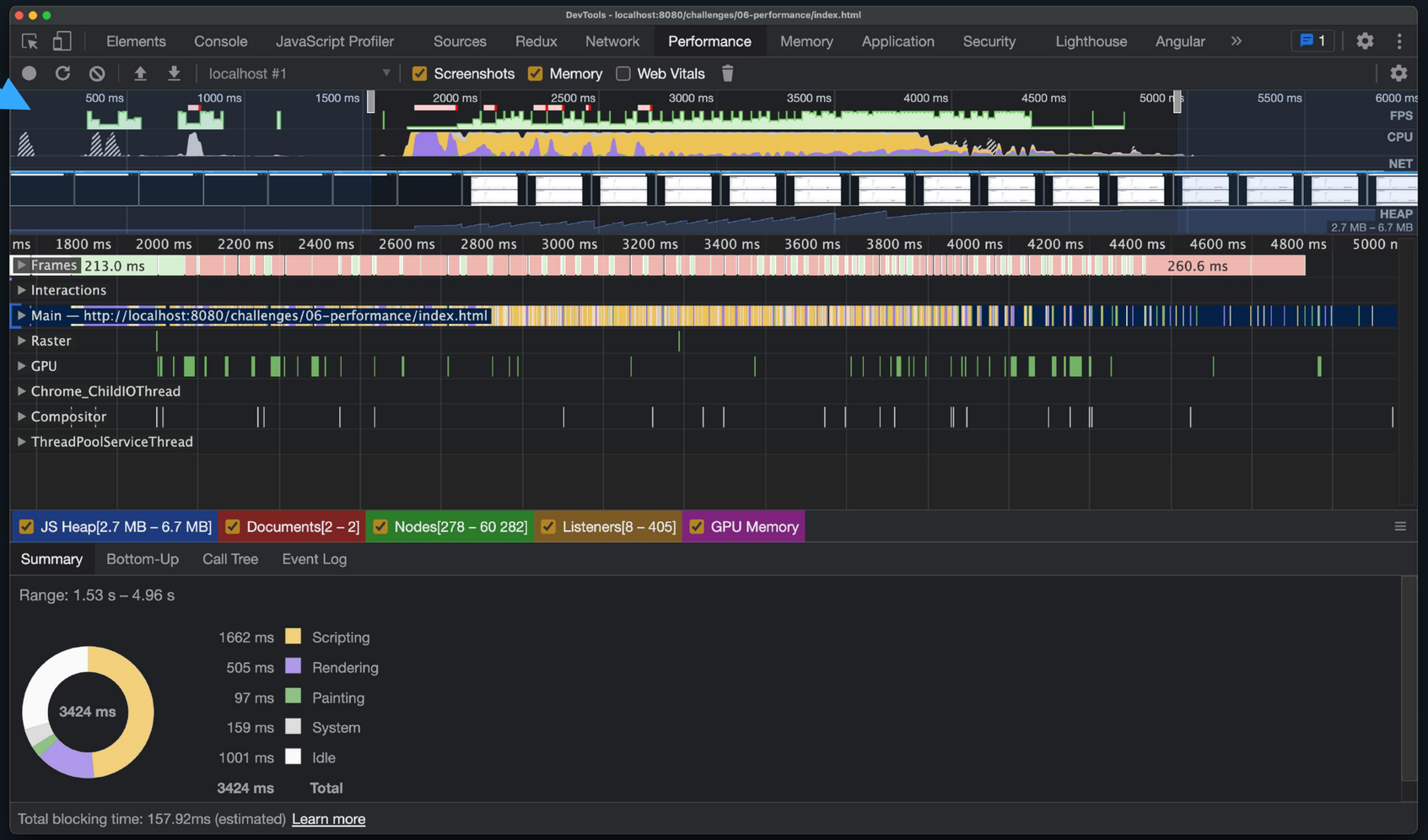
1. Open the **06 Performance** challenge in Chrome.
2. Open the **Performance** panel.
3. Click the **Screenshots** and **Memory** checkboxes to enable both features.
4. Enter the state “Colorado” into the search field.
5. Clock the **Stop** button to stop for recording.

Performance

Frames per second

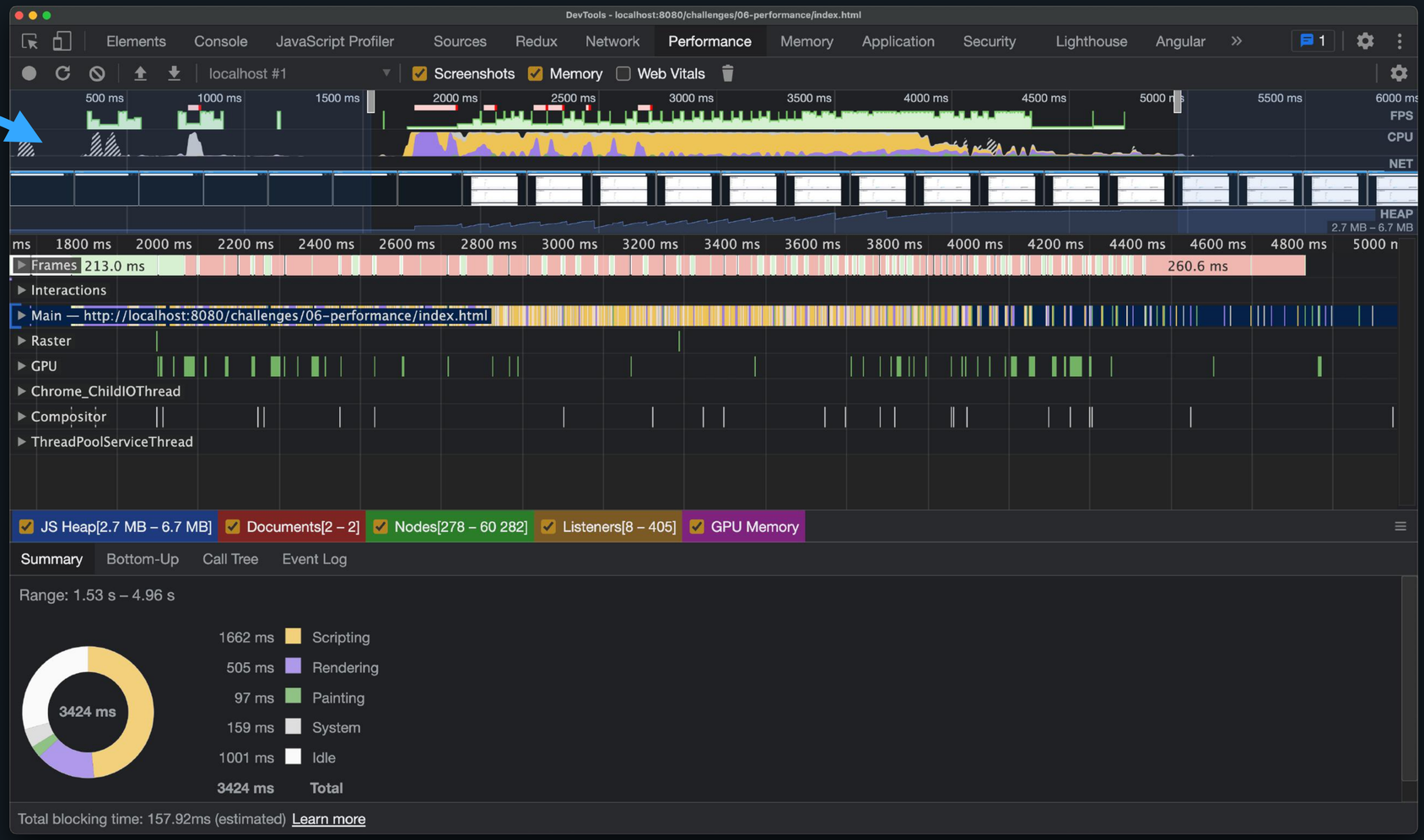
Green 👍

Red 👎



Performance

CPU summary



CPU categories

- Scripting (JS)
- Rendering
- Painting
- System
- Idle

Scripting

- [x] Animation frame fired/ cancel/requested
- [x] GC event
- [x] DOMContentLoaded
- [x] Evaluate script

Scripting

- [x] Event (e.g. mouseup)
- [x] Function call
- [x] Install timer
- [x] Remove timer

Scripting

- [x] XHR ready state change
- [x] XHR load

Rendering

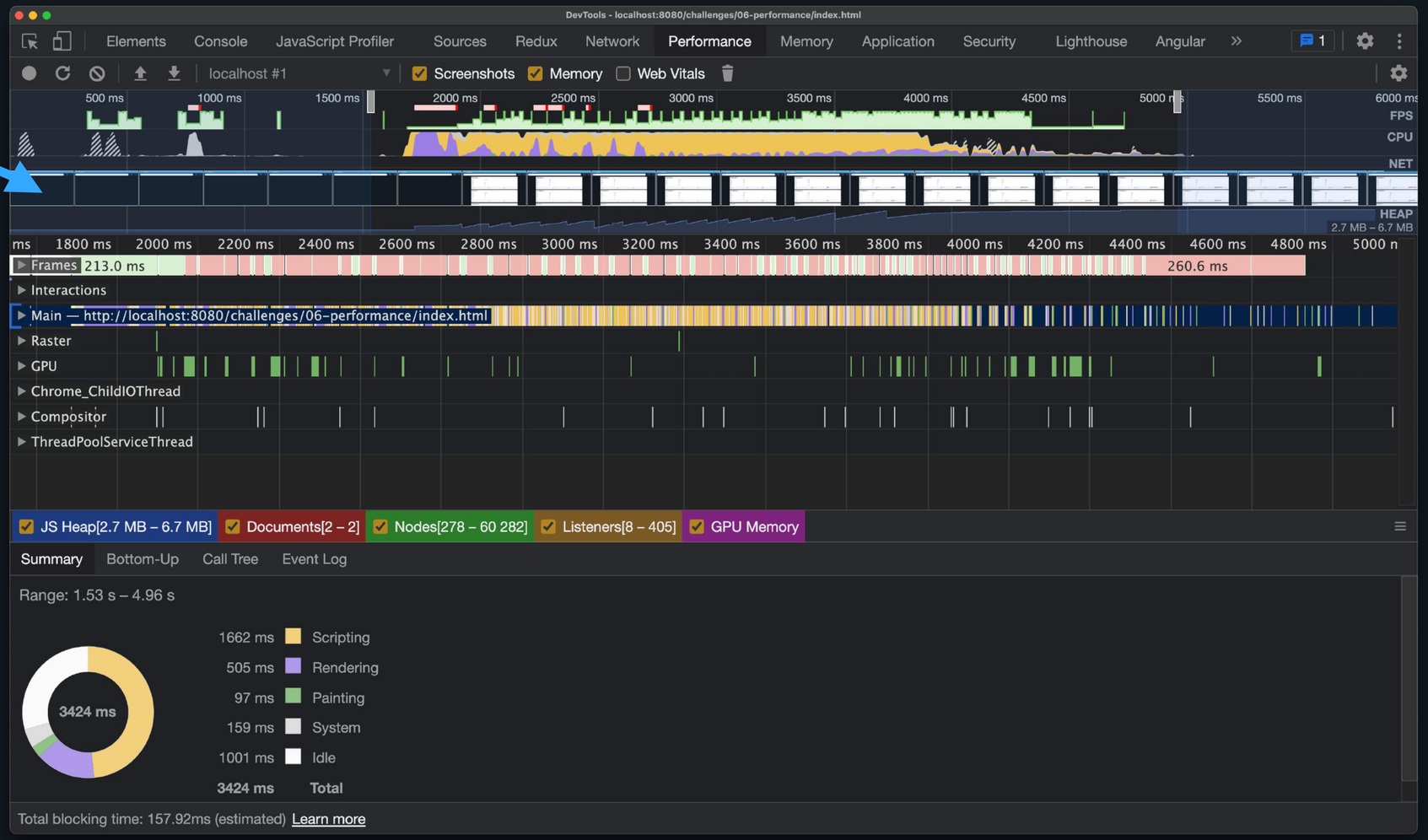
- [x] Invalidate layout
- [x] Layout
- [x] Recalculate style
- [x] Scroll

Painting

- [x] Composite layers
- [x] Image decode
- [x] Image resize
- [x] Paint

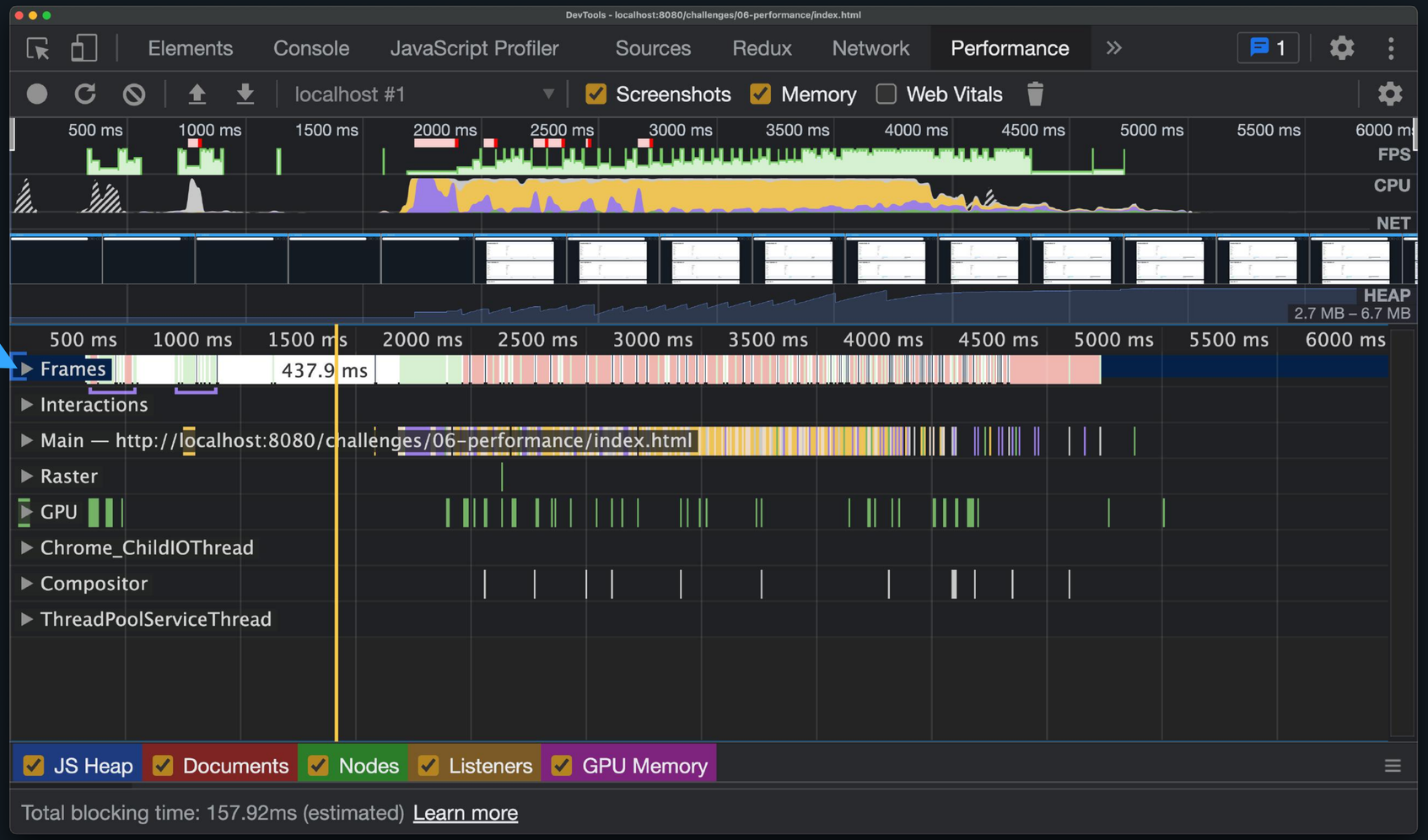
Performance

Screenshots

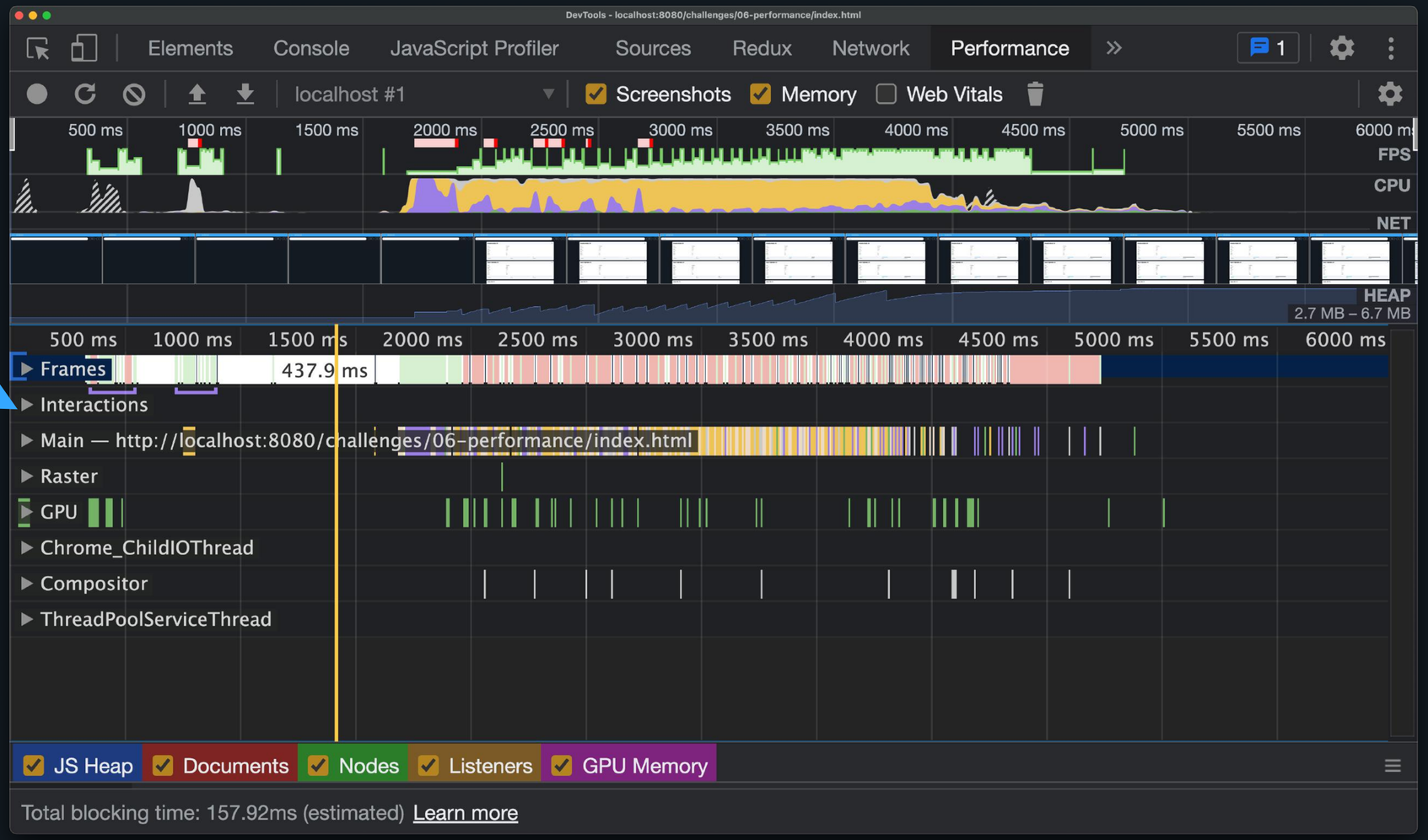


Performance

Frame rate per second activity



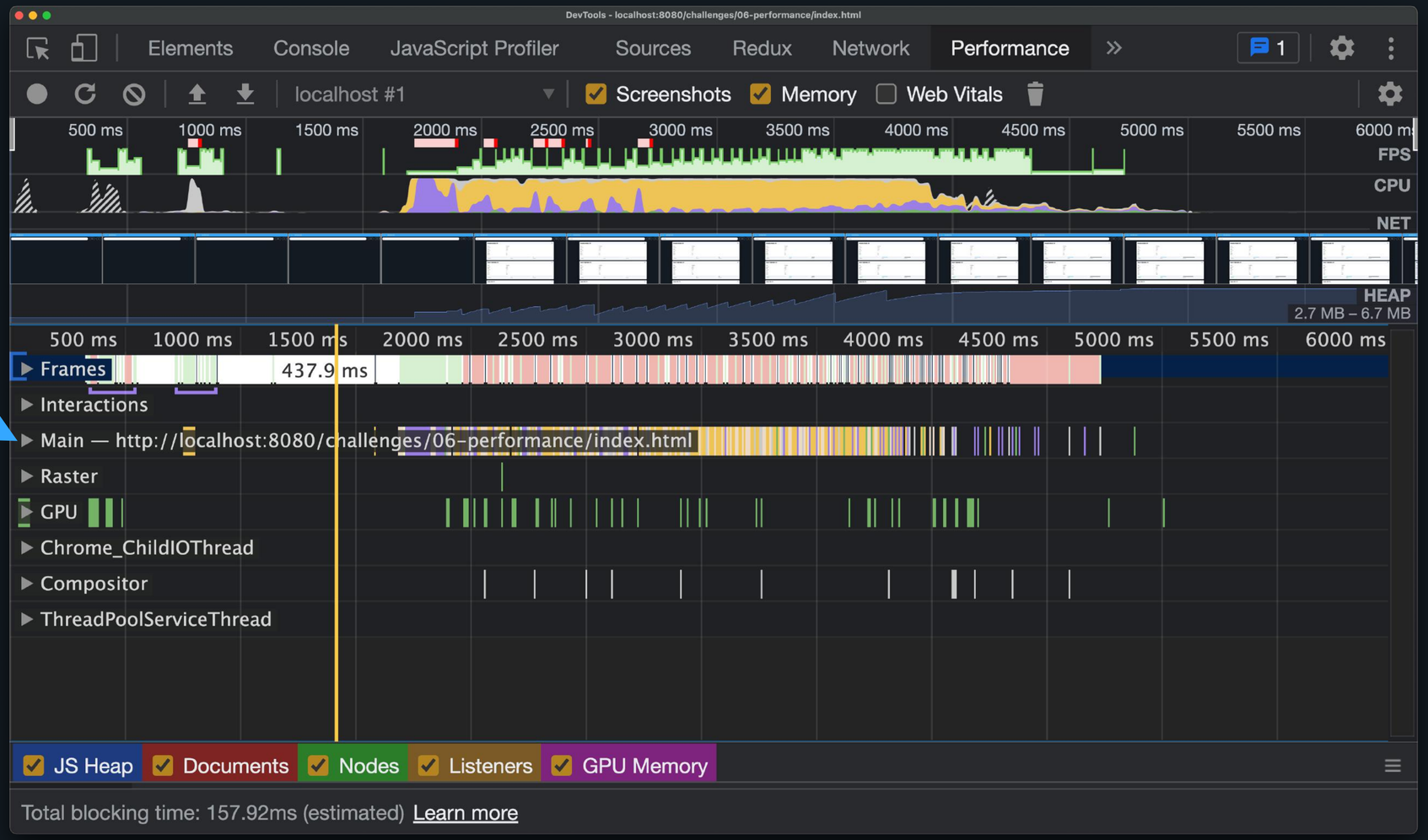
Performance



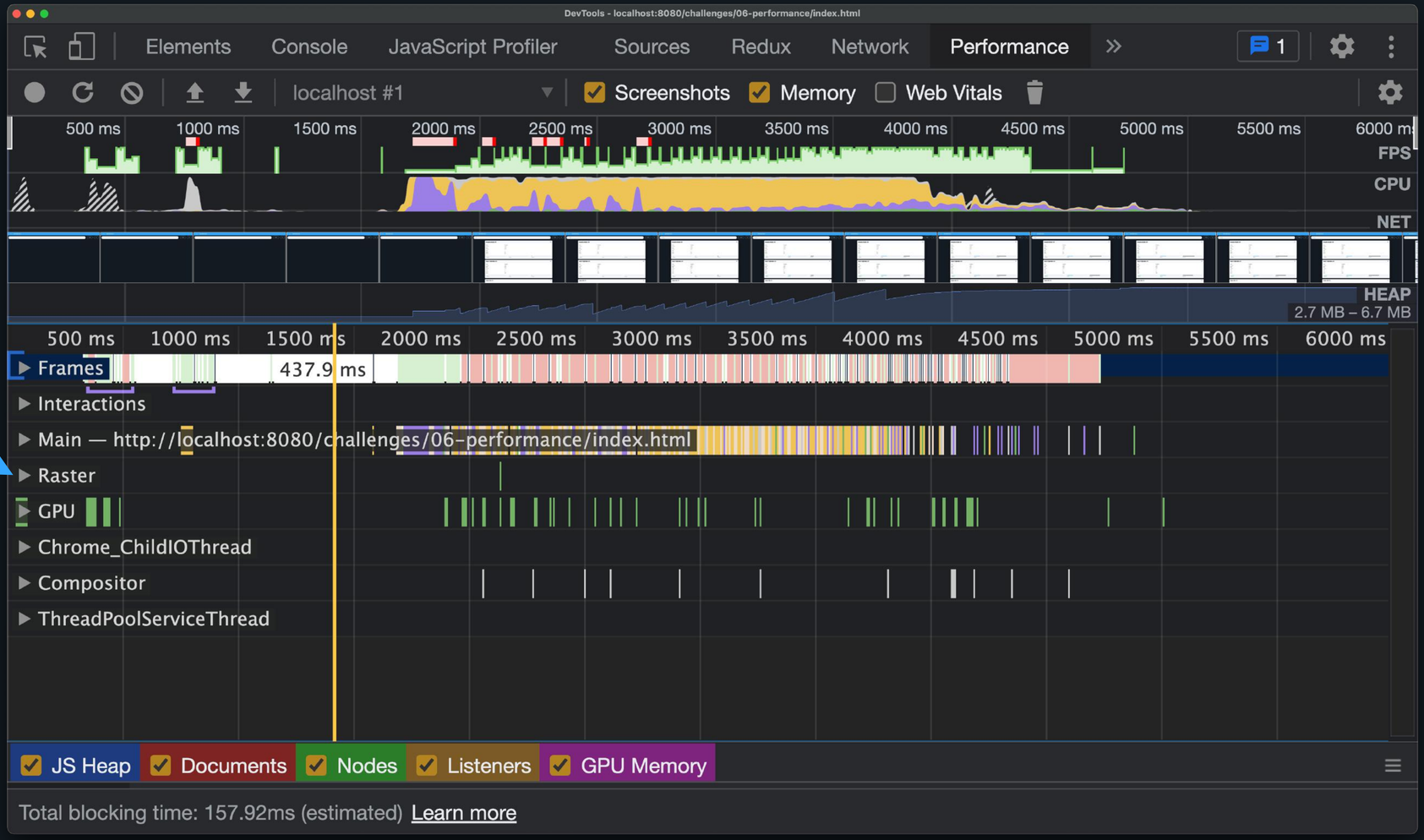
Interactions (by the user) activity

Performance

Main thread activity including call stacks (aka flame charts)

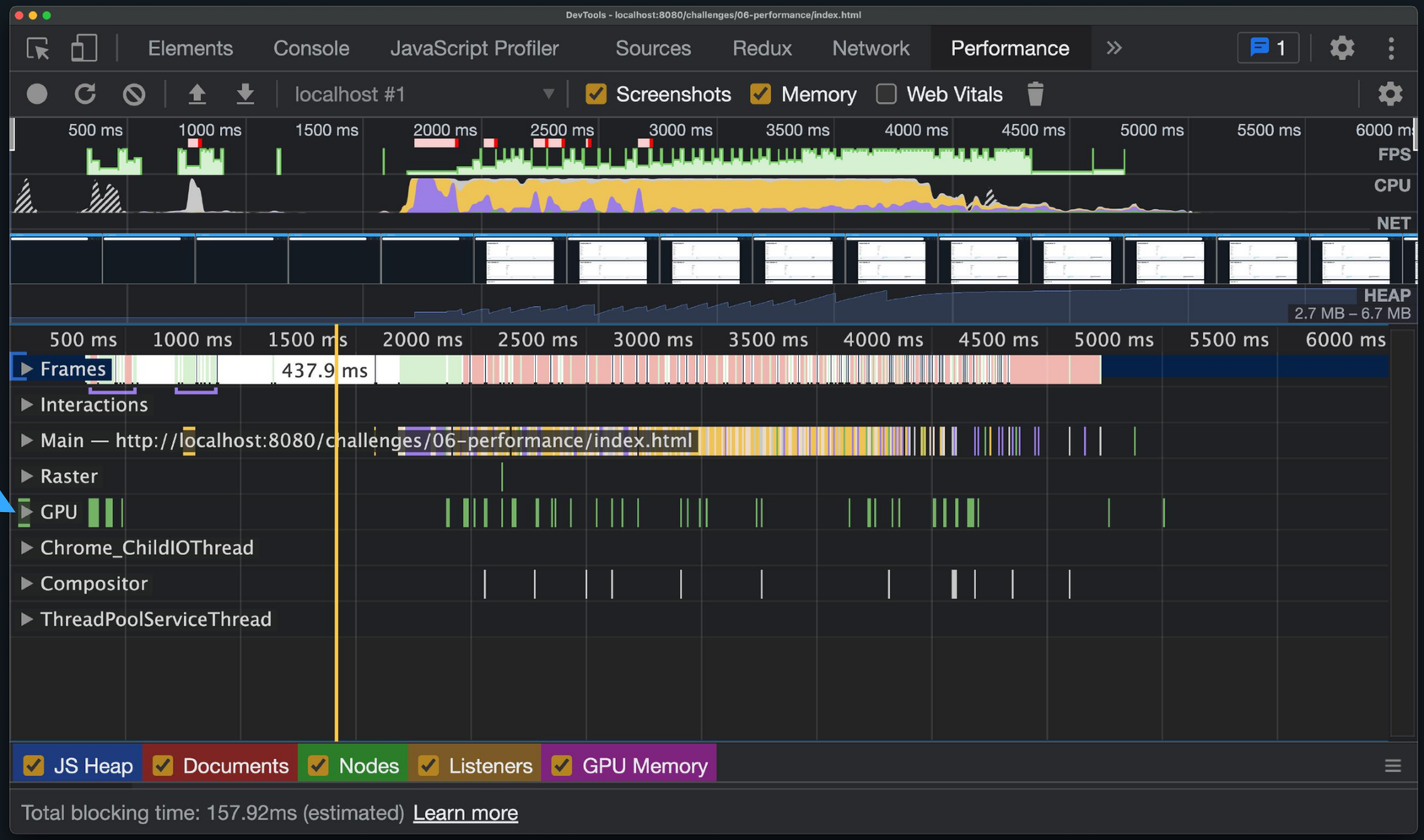


Performance



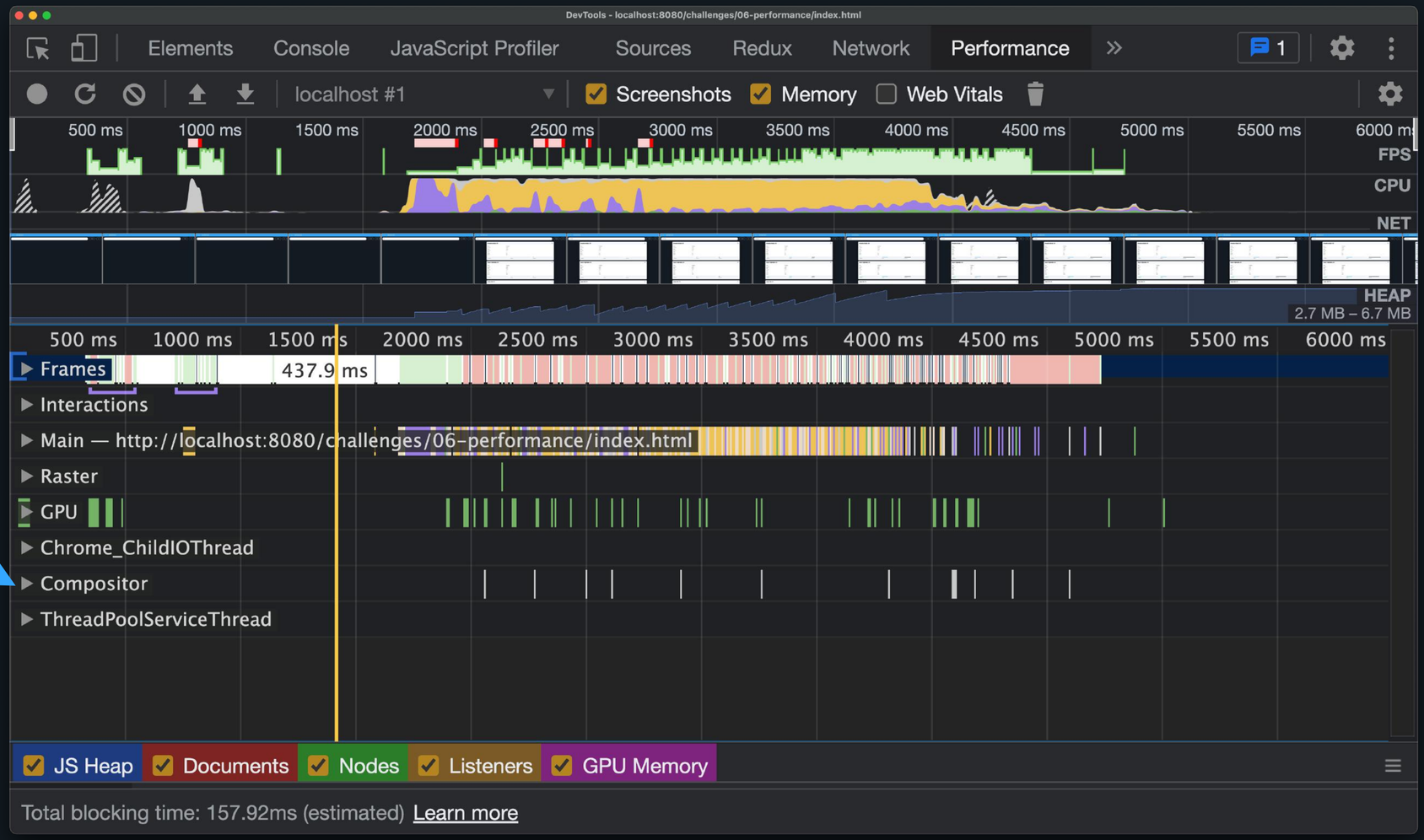
Rasterizer thread activity

Performance



GPU activity

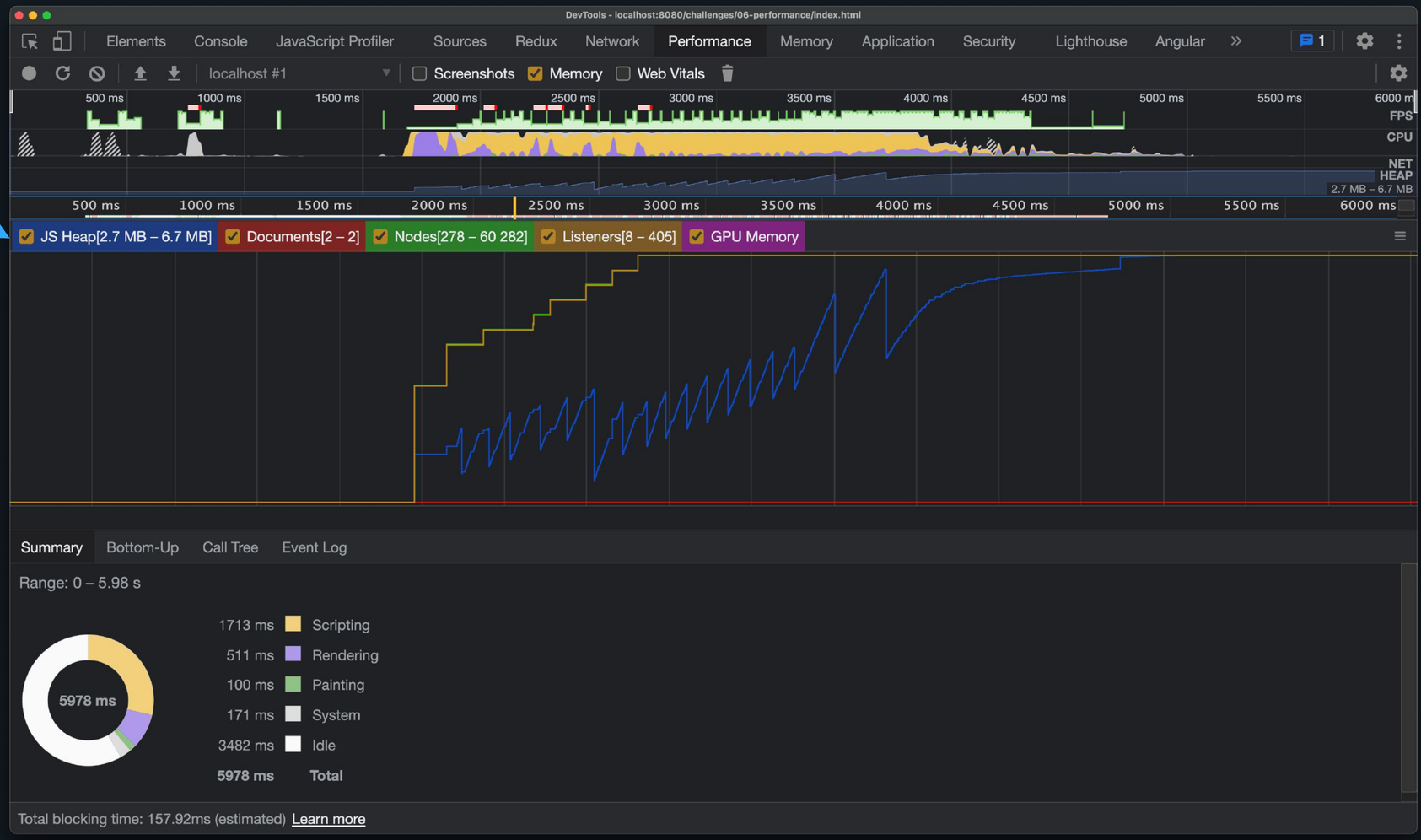
Performance



Compositor activity

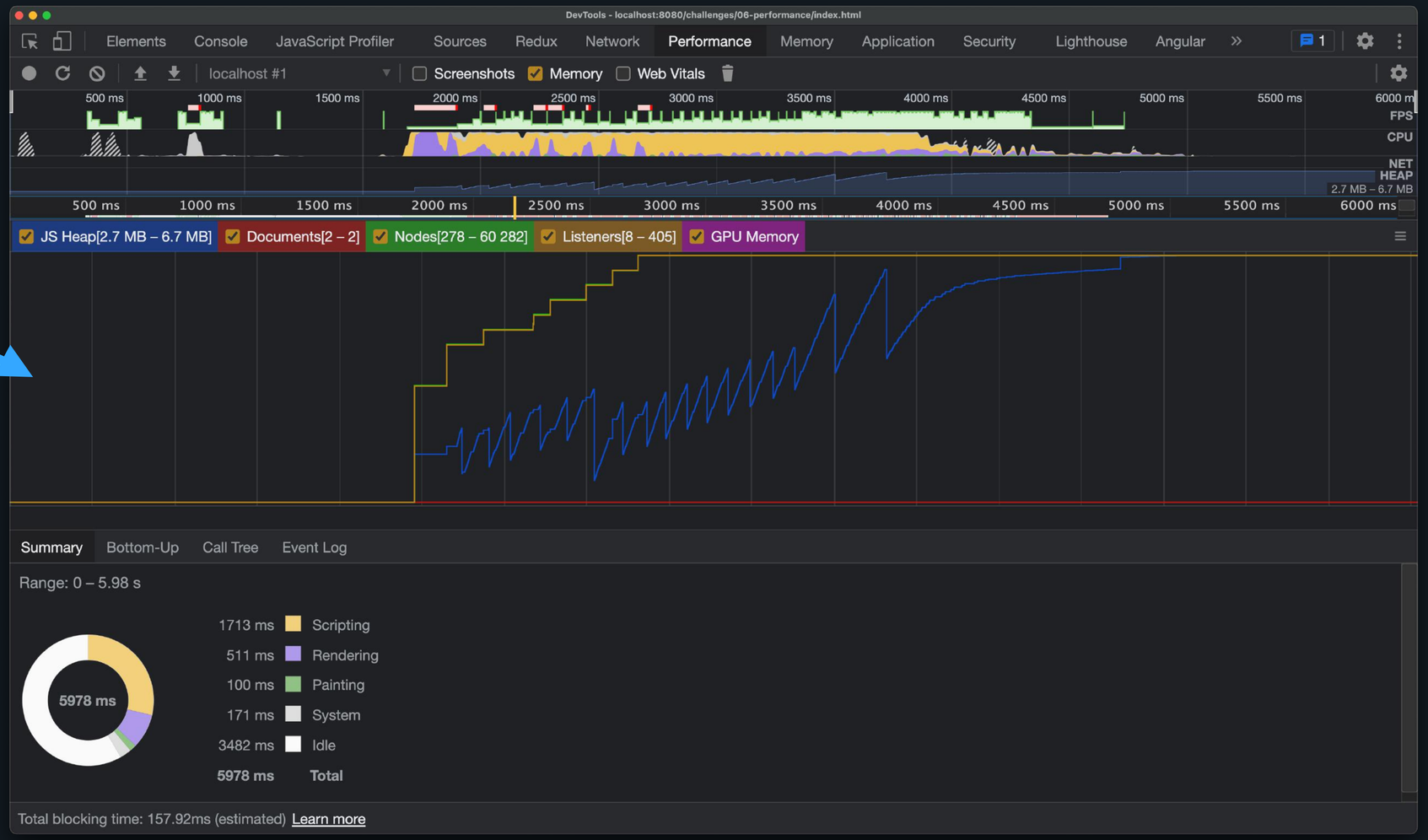
Performance

Toggle performance metrics

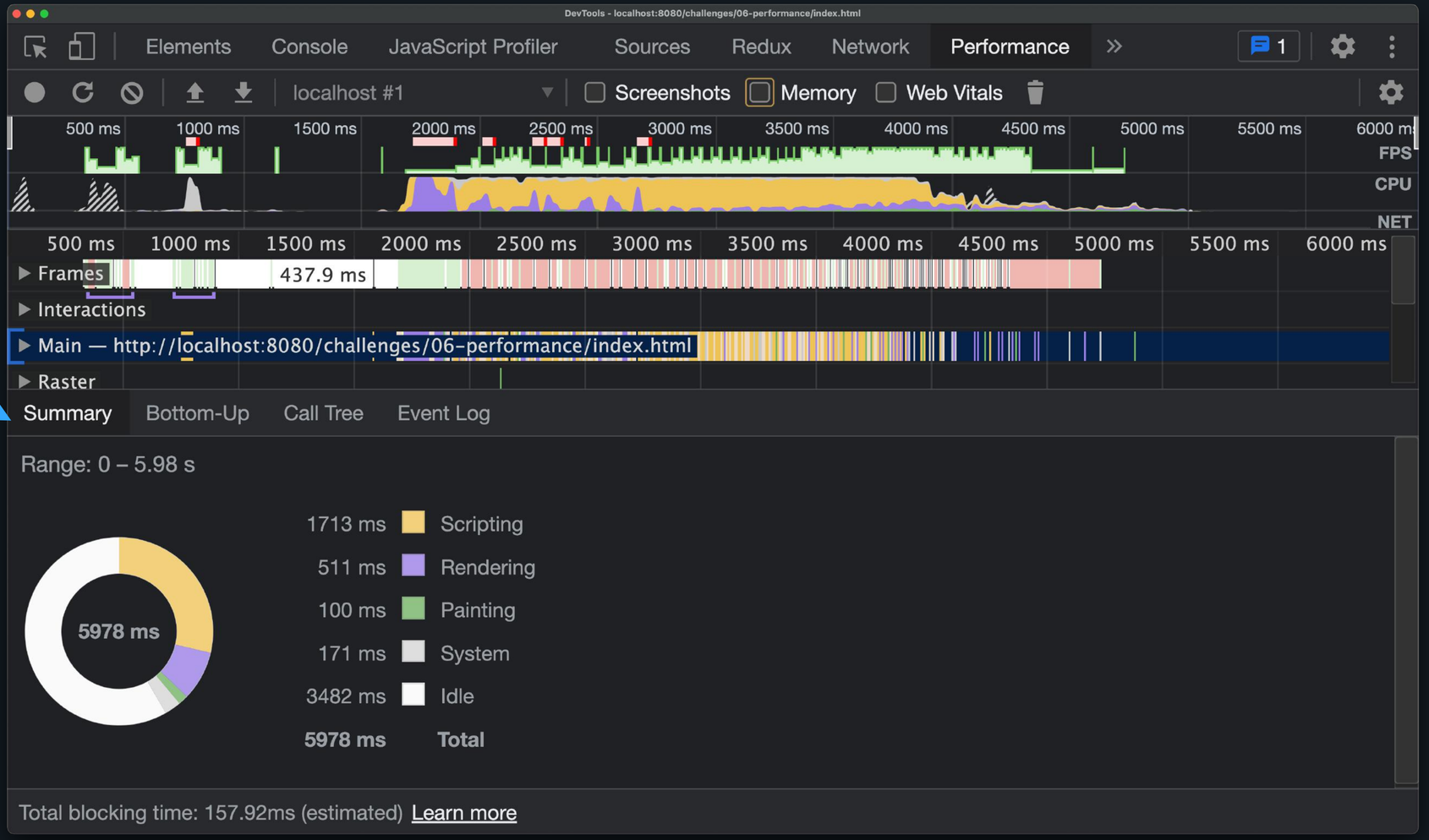


Performance

Graph of performance metrics

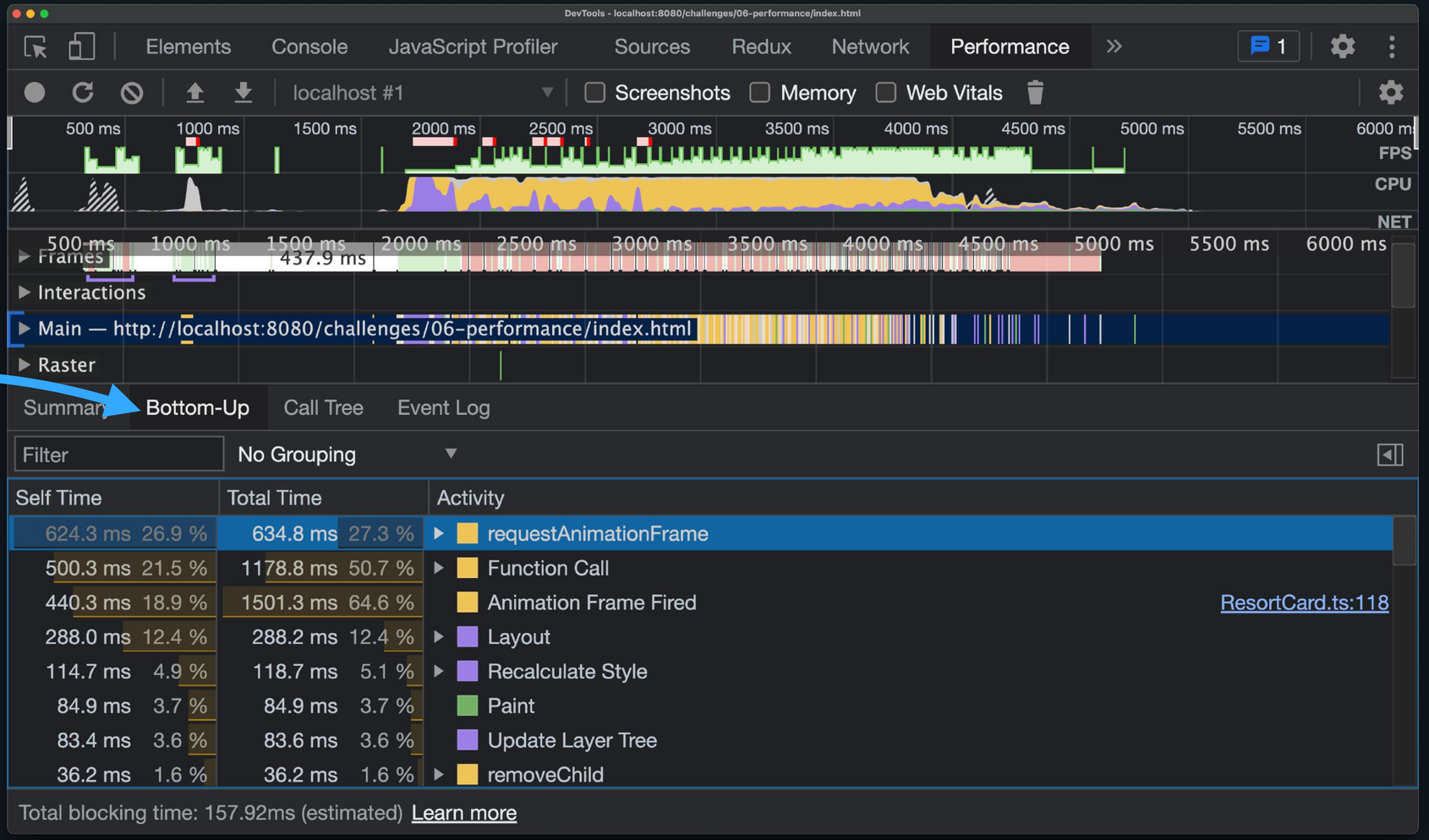


Performance



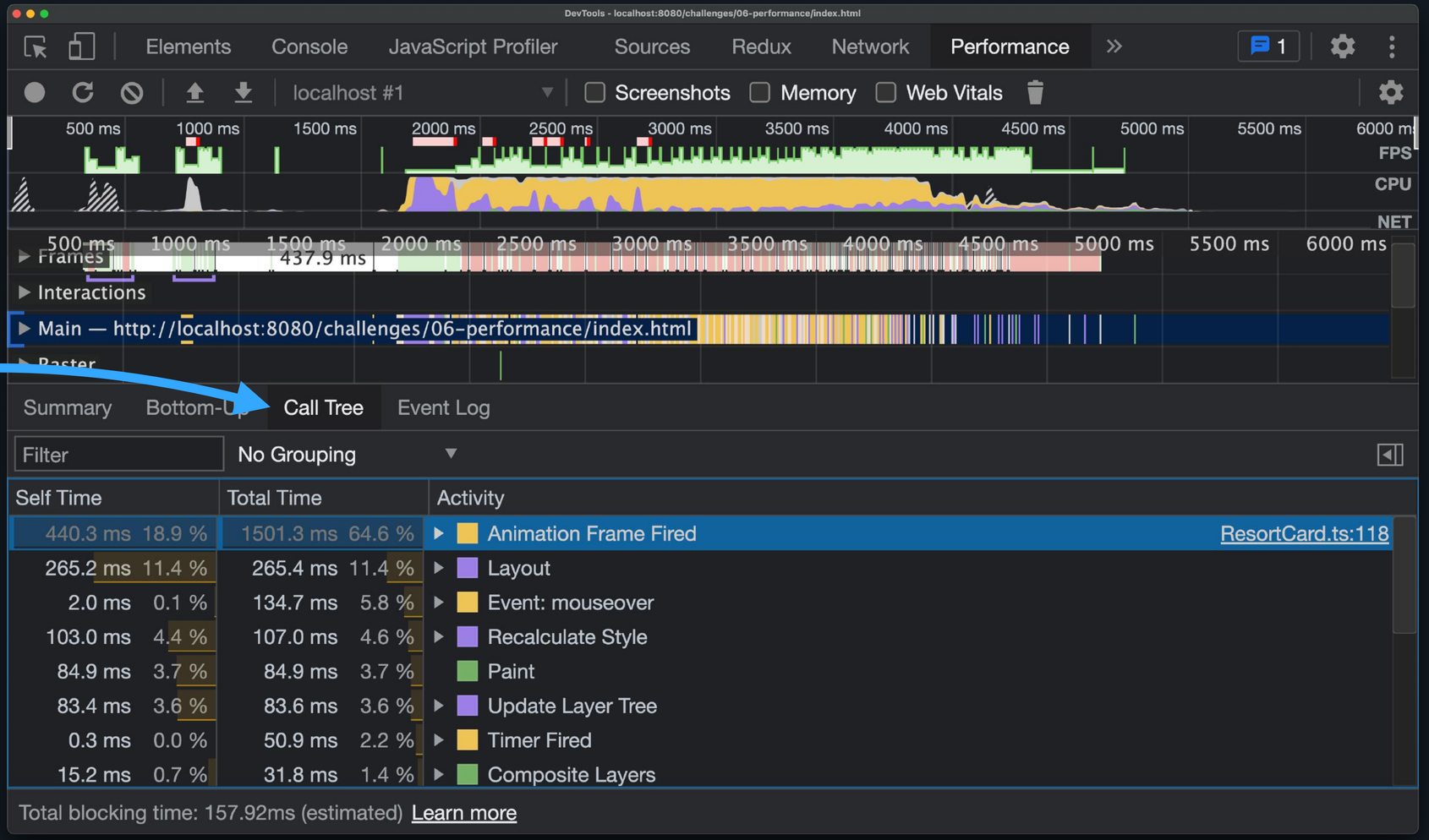
Performance summary during the selected timeframe or event

Performance



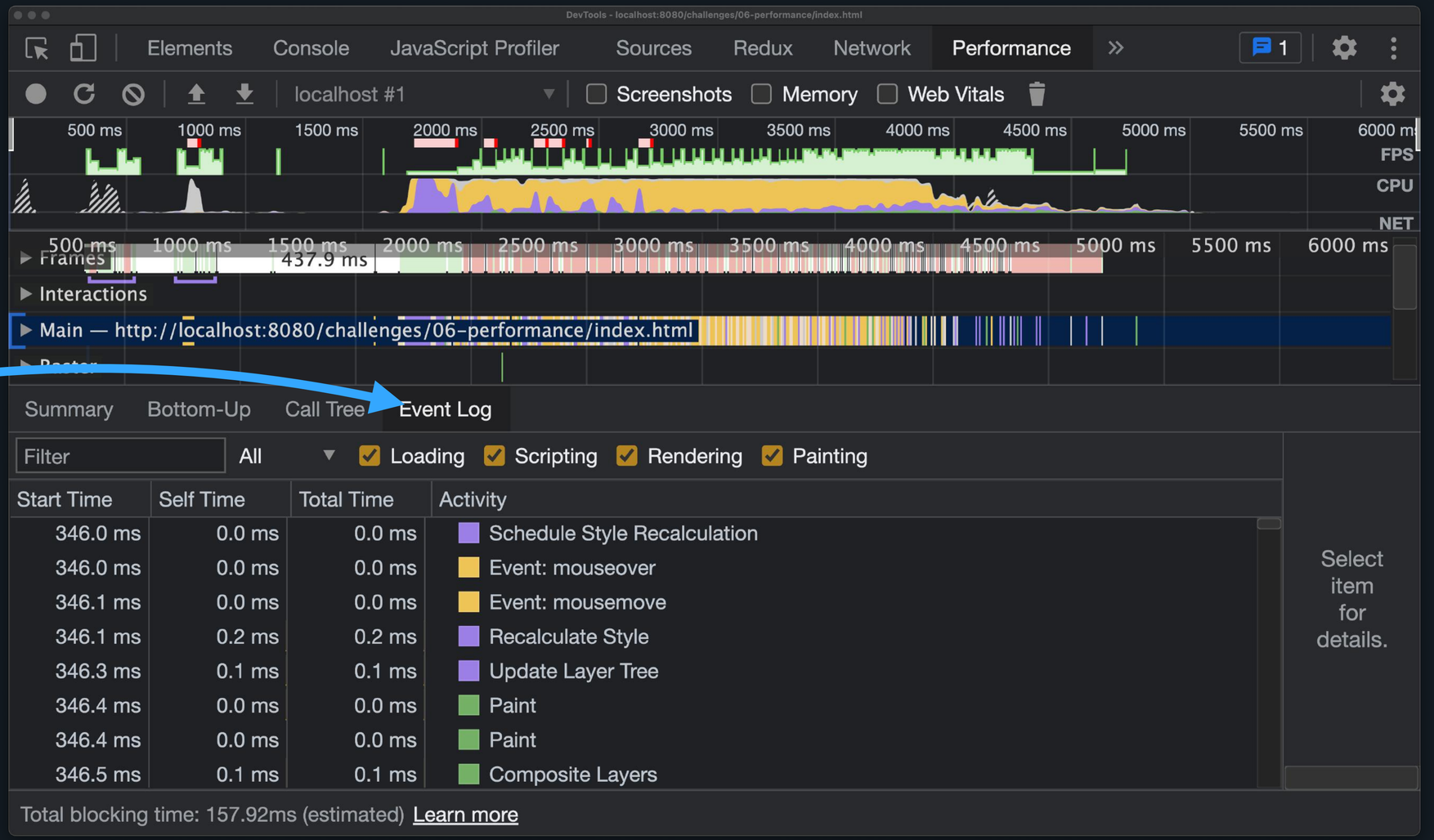
Events during the selected timeframe or event ordered by the self *time spent*.

Performance



Events during the selected timeframe or event ordered by the *total time spent*.

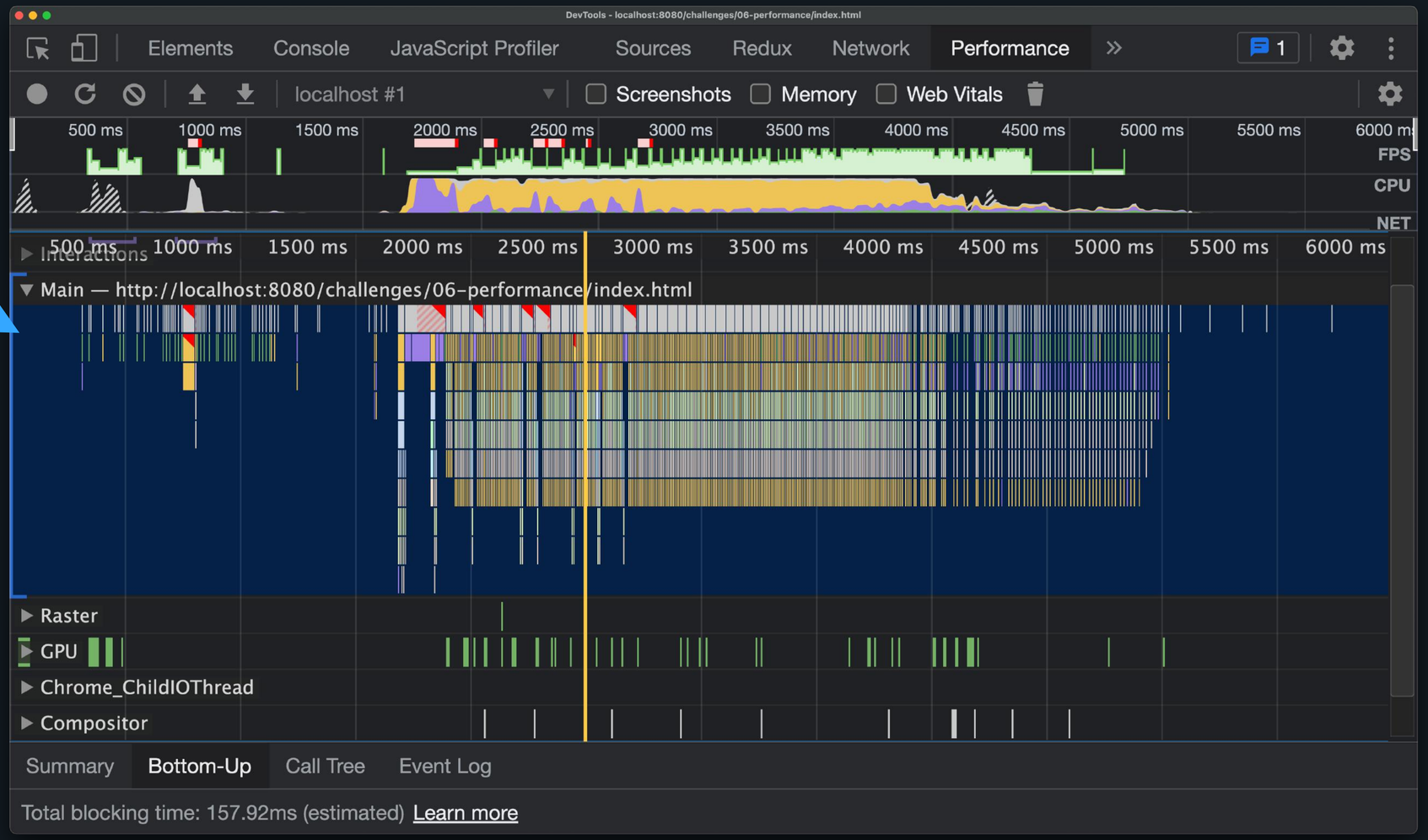
Performance



Events log in the order in which they occurred.

Performance

The main thread call stacks are vital for analyzing and solving runtime performance issues.



Performance

- [x] Color coded (matching summary)
- [x] X-axis represents recording over time
- [x] The wider the bar, the longer the event took
- [x] Y-axis represents the call stack

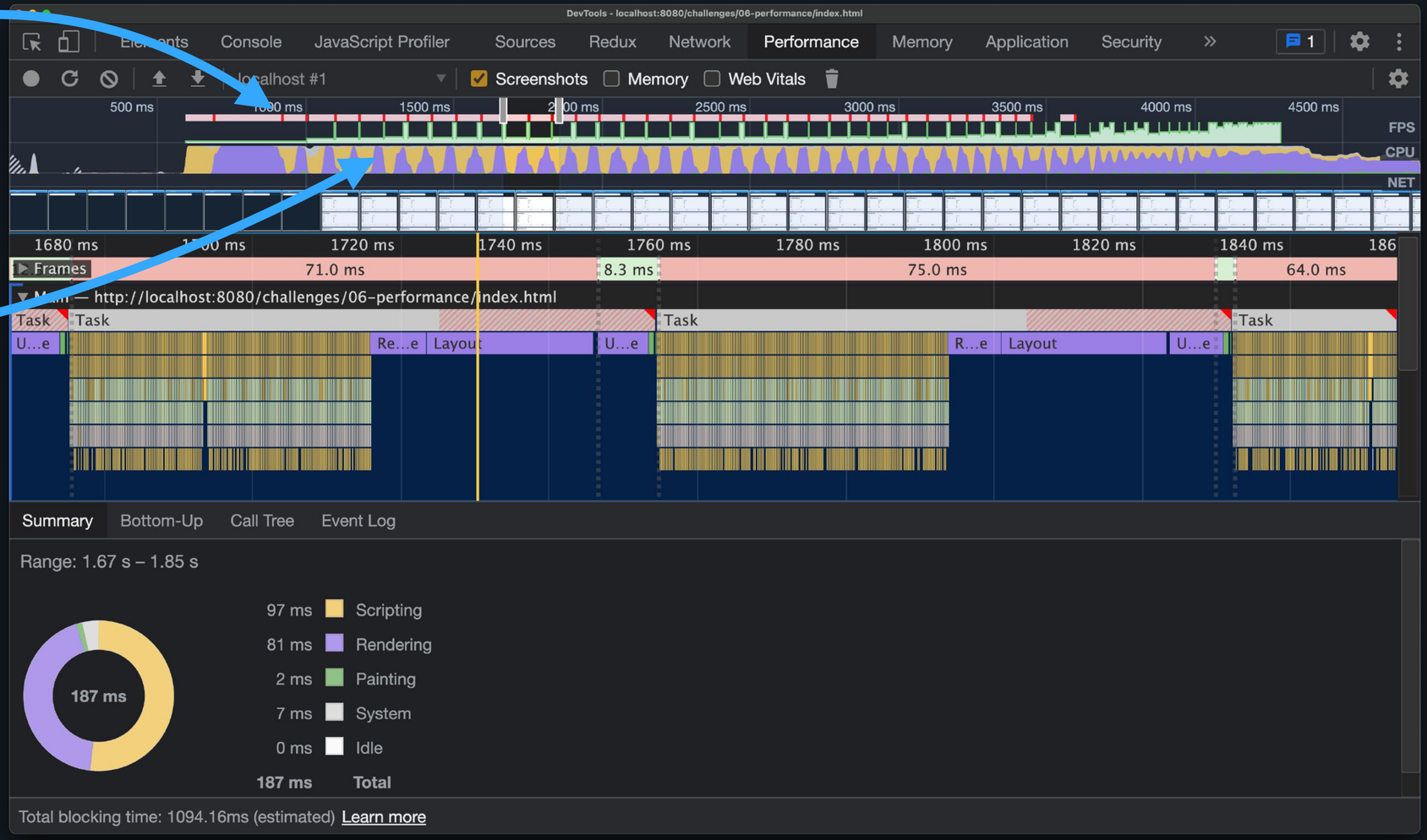
Performance Analysis 1

1. Open the **06 Performance** challenge in Chrome.
2. Open the **Performance** panel
3. Start a new recording and then click the **All** button to show all of the resorts.
4. Analyze the frame rates. Where are they dropping?
5. What amount of work is the compositor doing?
6. Use the **Call Tree** tab to determine the root activity that caused the *most work*.
7. Use the **Bottom-up** tab to determine the root activity where the most time is *directly being spent*.

Performance Analysis 1

Woah, lots of dropped frames here.

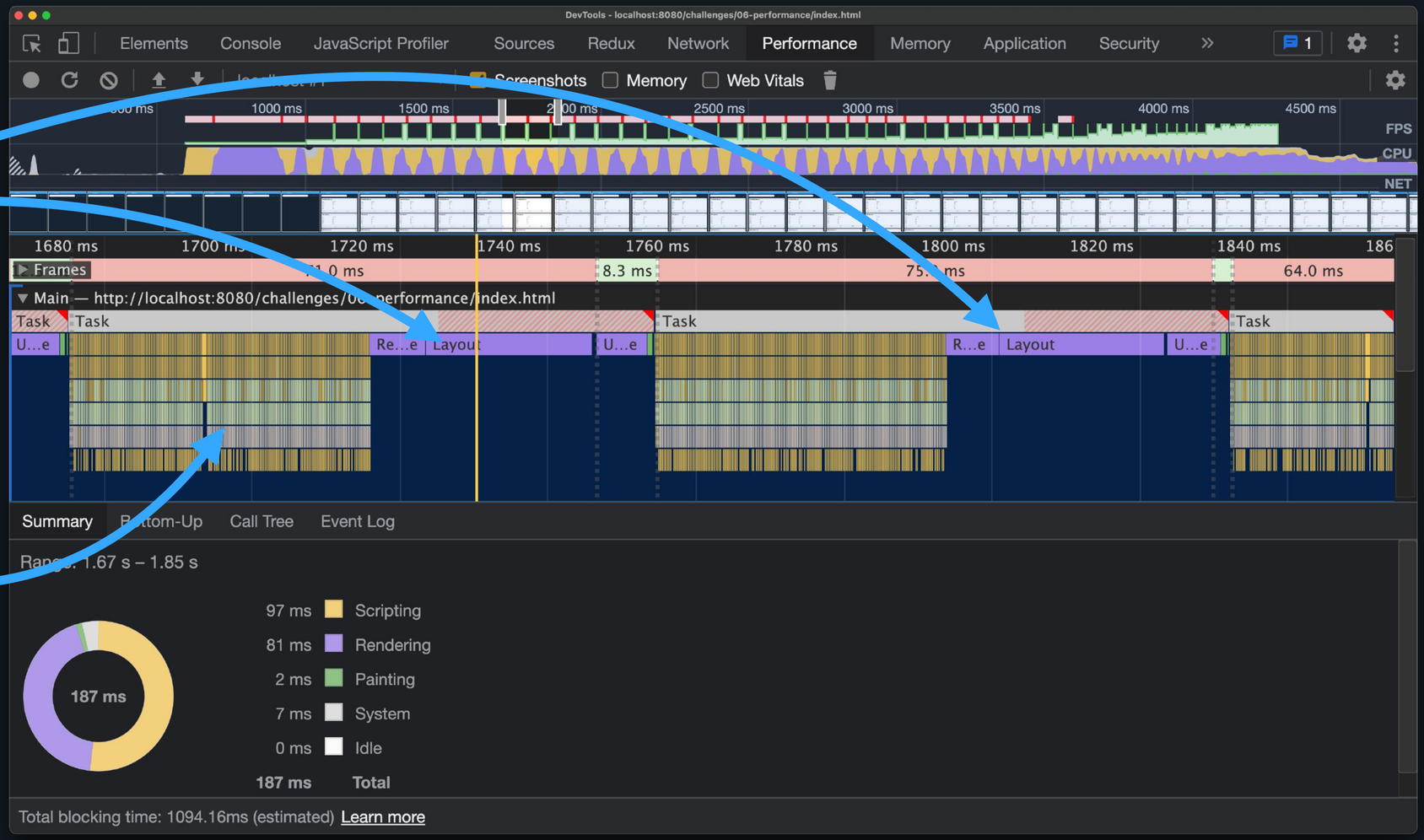
Look at all of that rendering work, huh...



Performance Analysis 1

Look at that series of: recalculation of styles, layout and updating the layer tree.

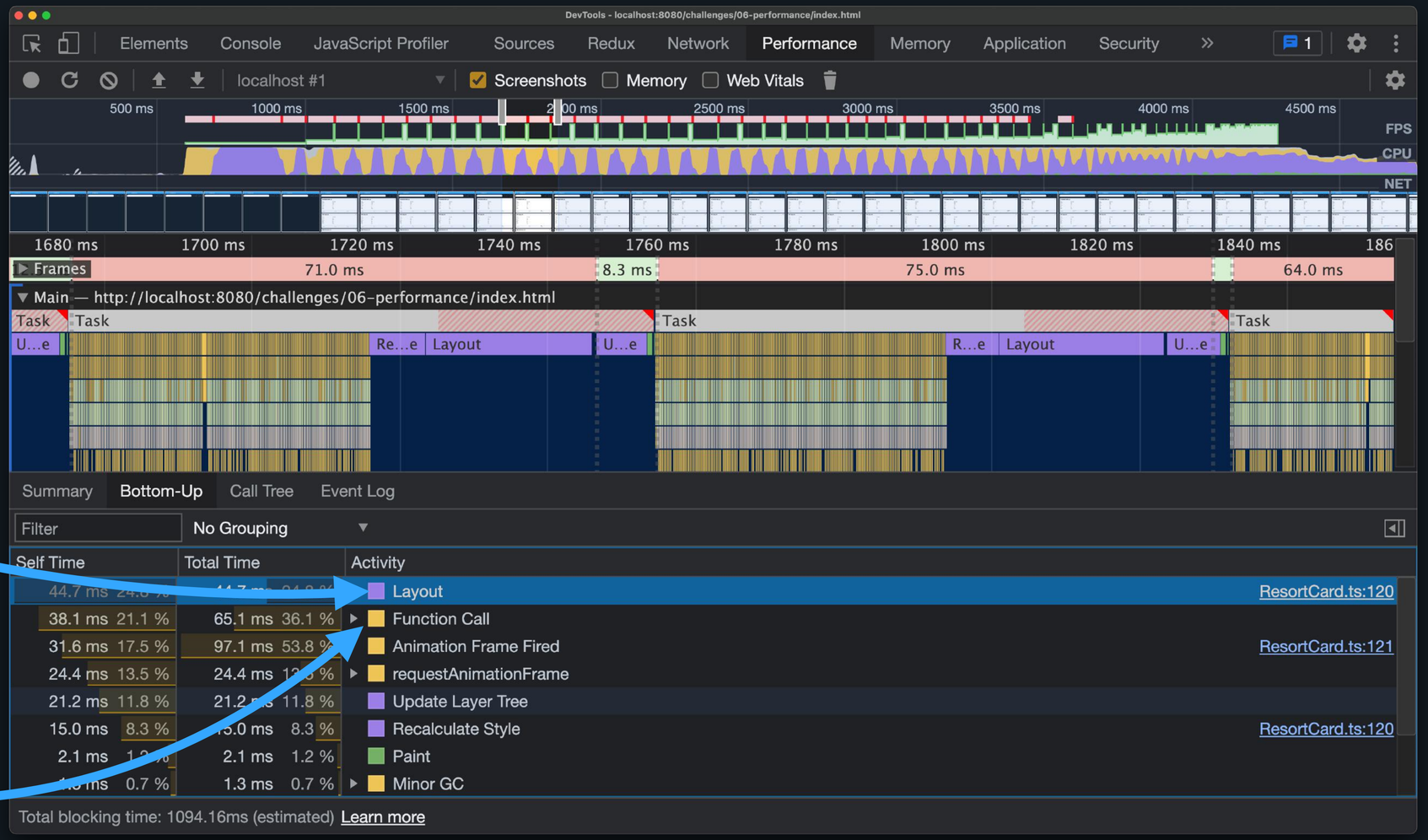
That’s a lot of function calls, what is that?



Performance Analysis 1

Layout is our number one issue based on the bottom-up analysis

Function call is also directly consuming CPU time



Performance Analysis 1

- [x] Recalculation of style, layout, and update layer tree is occurring frequently and expensive.
- [x] Function calls are frequent
- [x] Lots of frame drops
- [x] We need to fix this!

Refactor 1

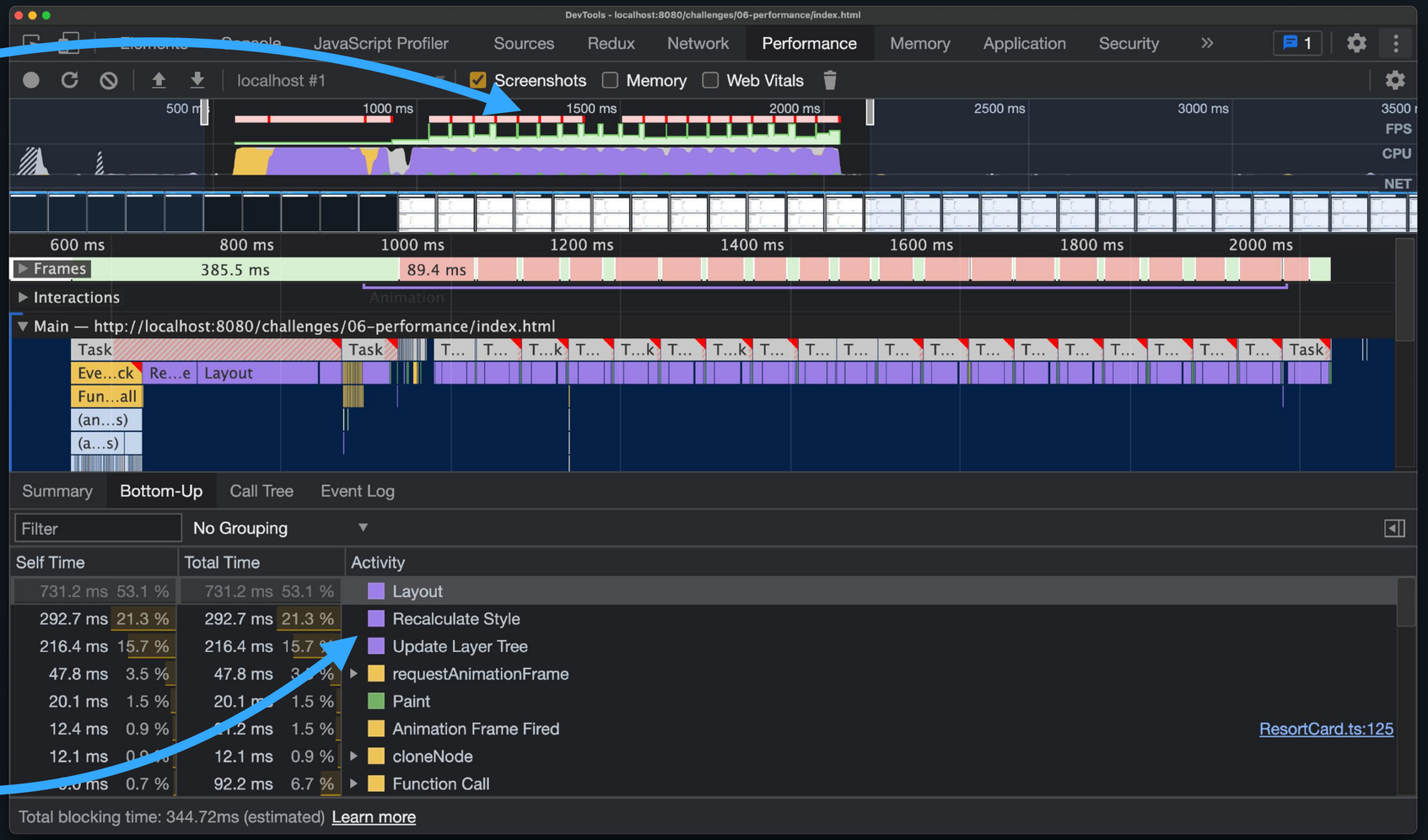
1. Based on the performance analysis we are observing LOTS of recalculation of styles, layout and updating the layer tree. The compositor is busy! 🥵
2. Open components/ResortCard.ts and determine how the animation is being performed. Yikes! 😱
3. Remove the **`animatePercent()`** method and update the **`setAcresPercent()`** method to set the **`width`** style property to a percent.
4. Open challenges/06-performance/index.html and add a CSS **`transition`** property for the width to animate the width of the **`.percent`** element. You'll need to wrap this style change in the callback function supplied to **`requestAnimationFrame()`**.

Performance Analysis 2

1. Open the **06 Performance** challenge in Chrome.
2. Open the **Performance** panel
3. Start a new recording and then click the **All** button to show all of the resorts.
4. Analyze the frame rates. Where are they dropping?
5. What amount of work is the compositor doing?
6. Use the **Call Tree** tab to determine the root activity that caused the *most work*.
7. Use the **Bottom-up** tab to determine the root activity where the most time is *directly being spent*.

Performance Analysis 2

Ok, that’s better. We are getting few dropped frames. Still not great.



Lots of layout work here still. But all those Function calls are gone!

Performance Analysis 2

- [x] Remaining issues: layout and fps
- [x] Compositor is less busy
- [x] Expensive function as been removed
- [x] But, do we need to set the width for *all elements* at once?

IntersectionObserver

```
const intersectionObserver = new IntersectionObserver();
intersectionObserver.observe(percentEl);
```

IntersectionObserver

```
const intersectionObserver = new IntersectionObserver(
  (entries) => {
    // todo
  },
  {
    root: null,
    threshold: 1.0,
  }
);
intersectionObserver.observe(percentEl);
```

IntersectionObserver

```
const intersectionObserver = new IntersectionObserver(
  (entries) => {
    if (entries.length === 0) {
      return;
    }
    const entry = entries[0];
    if (entry.isIntersecting) {
      // todo: set width
    }
  });
```

Refactor 2

1. Open the components/ResortCard.ts file.
2. Instantiate a **`new IntersectionObserver()`** and observe the **`percentEl`**.
3. In the callback function, first check the length of the **`entries`** and then get the first **`entry`**.
4. Check the entry **`isIntersecting`** property to determine if the element is in view.
5. When the entry is intersecting, update the element's **`width`** in the first frame of the next available animation frame.
6. Otherwise, check if the element is initially in the viewport, and if so, set the element's **`width`**.

Performance Analysis 3

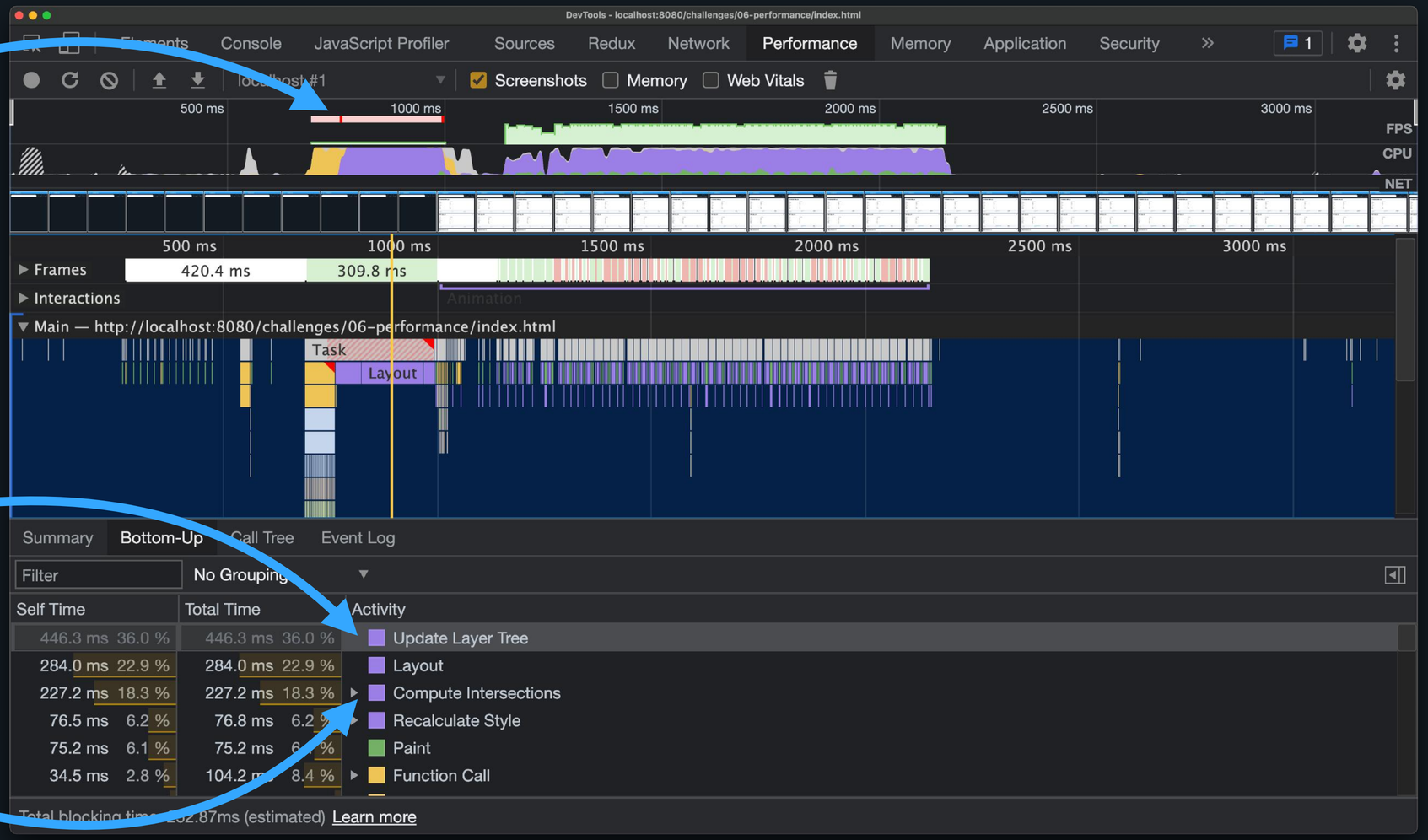
1. Open the **06 Performance** challenge in Chrome.
2. Open the **Performance** panel
3. Start a new recording and then click the **All** button to show all of the resorts.
4. Analyze the frame rates. Where are they dropping?
5. What amount of work is the compositor doing?
6. Use the **Call Tree** tab to determine the root activity that caused the *most work*.
7. Use the **Bottom-up** tab to determine the root activity where the most time is *directly being spent*.

Performance Analysis 3

Big improvement in fps!

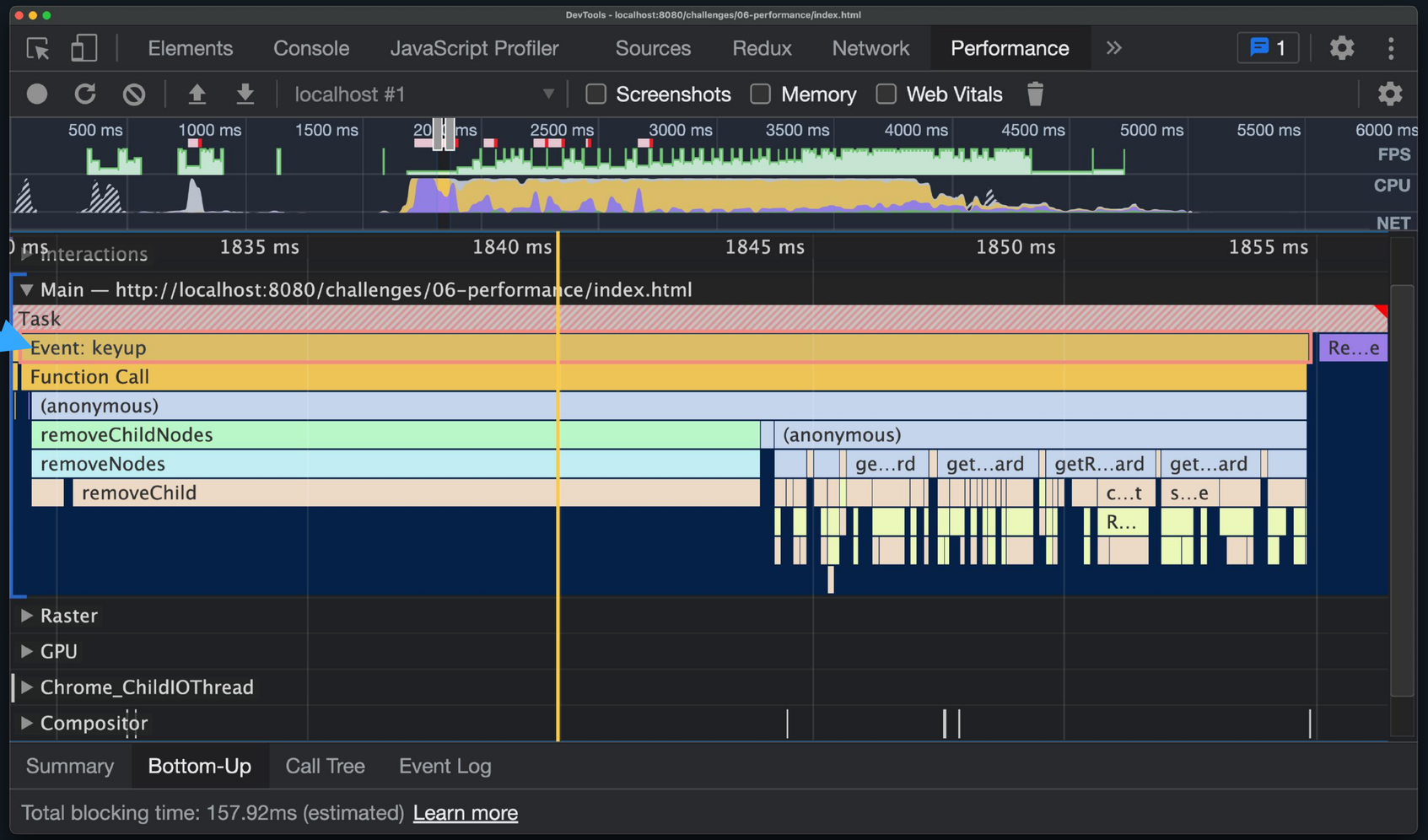
Updating the layer tree and layout is still there.

Computing intersections is a new issue.



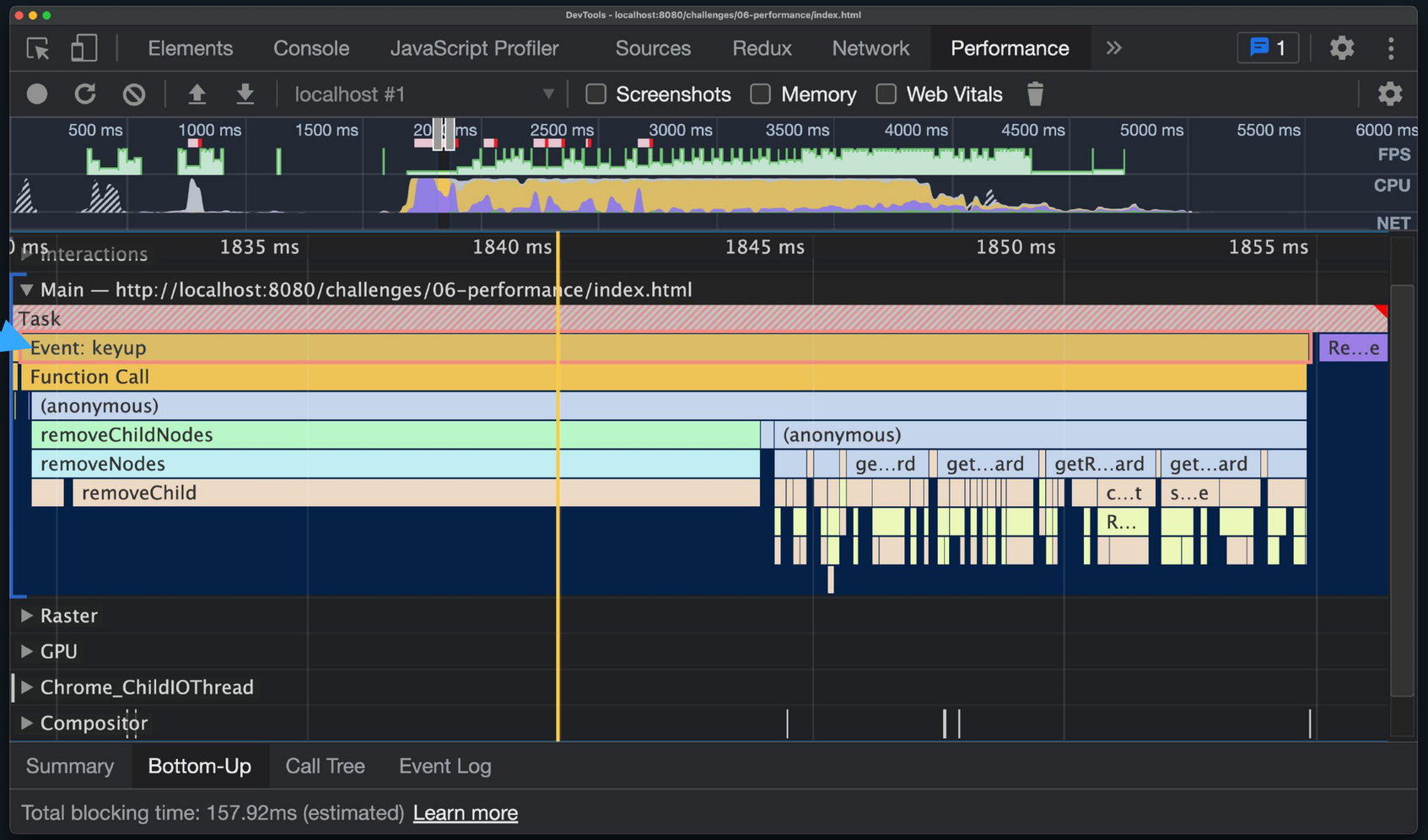
Analyzing Single Event

Let’s focus on a keyup event



Analyzing Single Event

Let’s focus on a keyup event



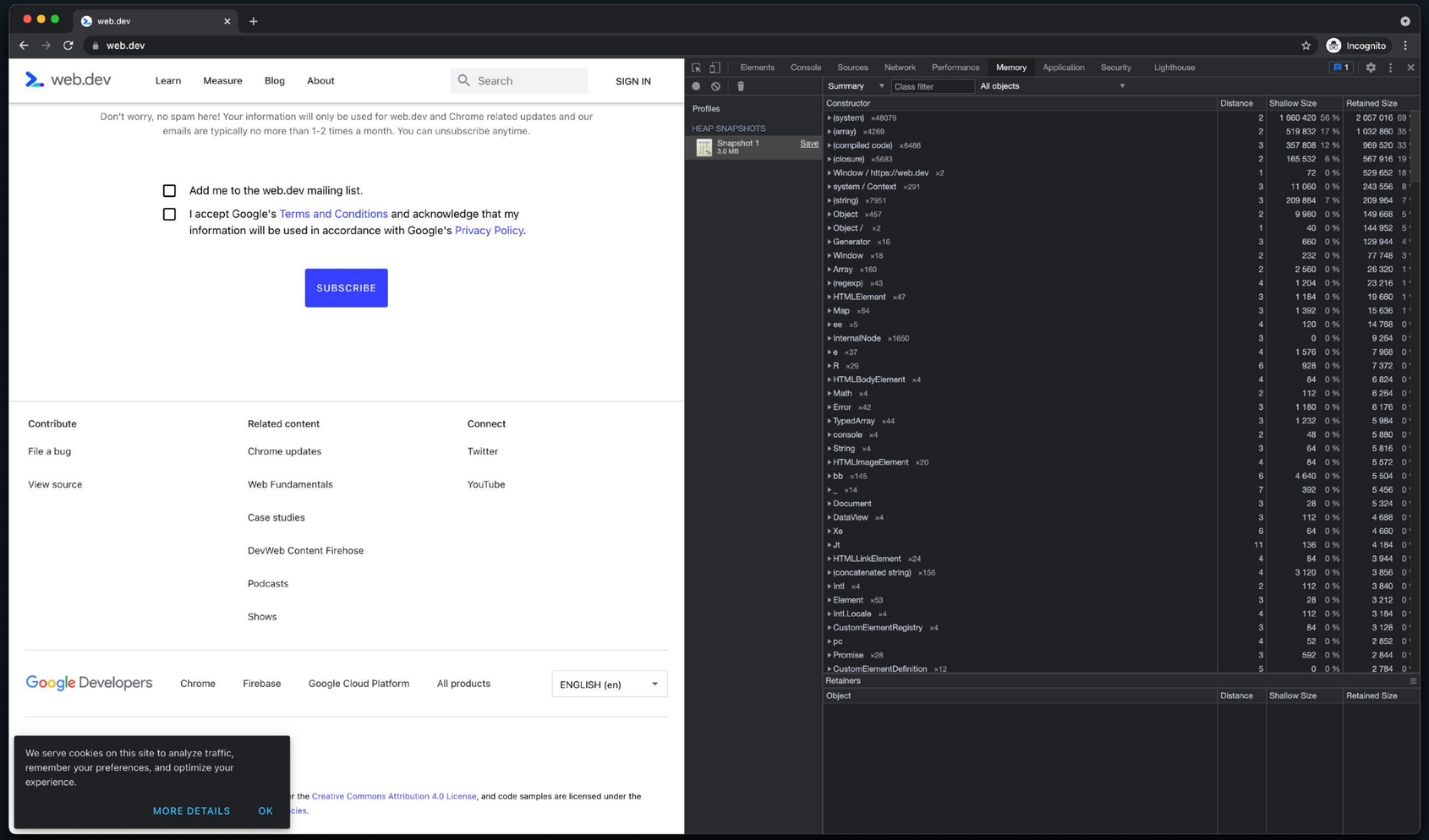
Flame graph analysis

1. Zoom in and select a single keyup event in the main thread timeline.
2. Use the **Call Tree** tab to determine the root activity that caused the *most work*.
3. Use the **Bottom-up** tab to determine the root activity where the most time is *directly being spent*.
4. Click the source link to show the associated code in the sources tab.

The Performance Panel is critical to analyzing and solving runtime performance of your app.

DevTools: Memory

DevTools: Memory



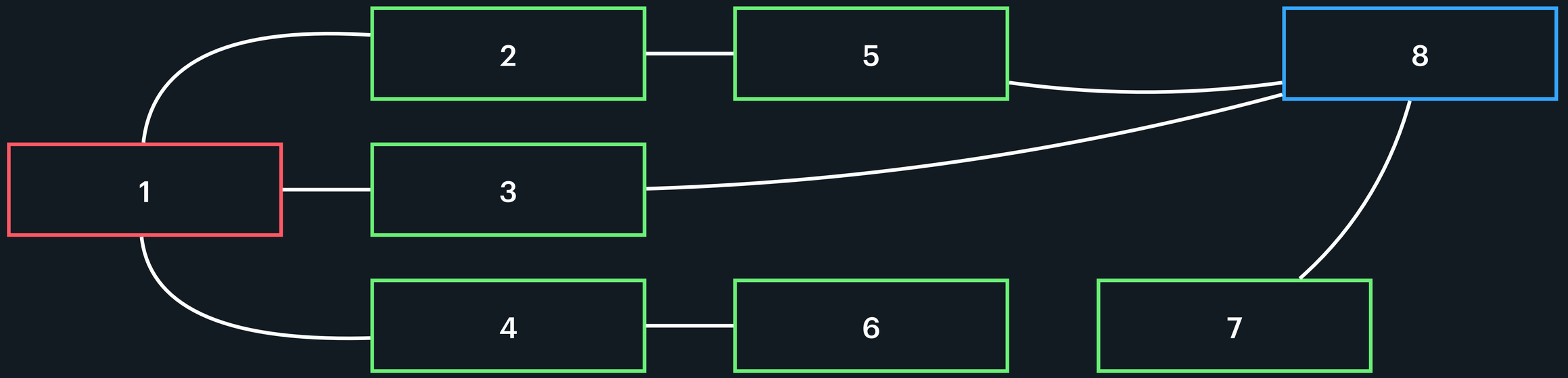
Object Size

- [x] Primitive types: numbers, booleans, and strings
- [x] Objects

Object Size

- [x] Directly contains primitives and/or objects
- [x] Implicitly via references

Memory



Root (1) Objects

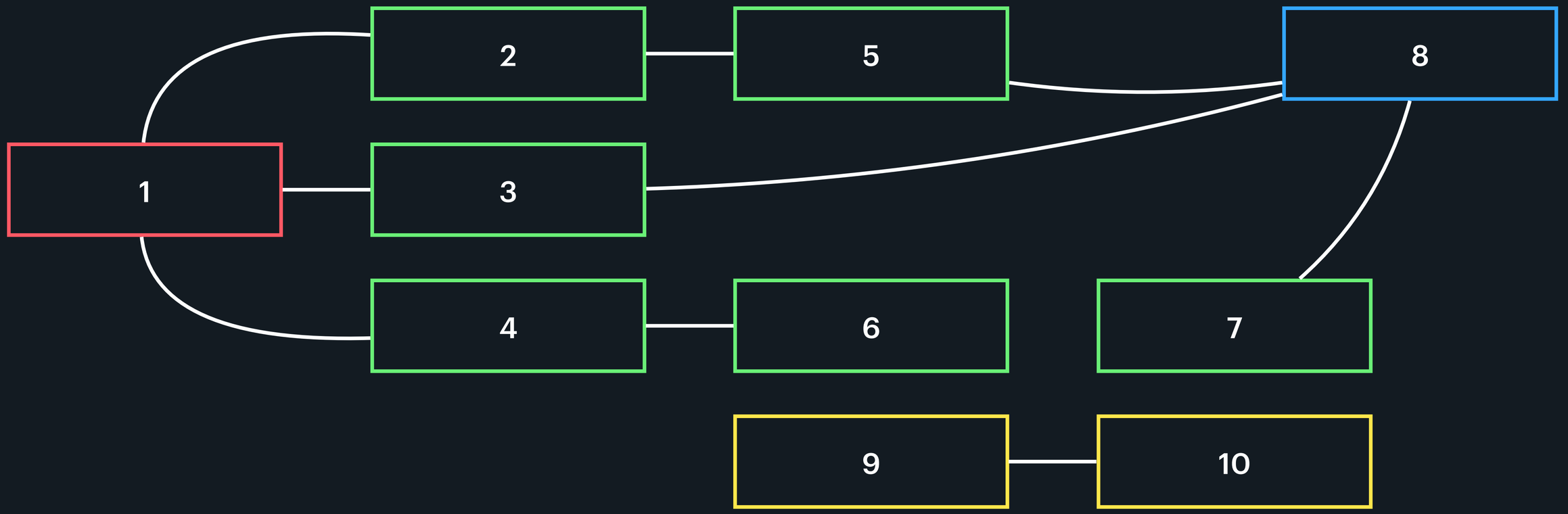
- `window` or `globalThis`
- Document DOM
- Retained by debugger (breakpoints)

Objects with references cannot be automatically disposed by the garbage collector (GC)

Object Sizes

- Shallow: memory that is held by itself
- Retained: memory that is freed up when it’s deleted

Memory



GC in V8

- Generational garbage collector
- Performed during idle

GC in V8

Most Objects die young

- Young generation put into semi-space
- GC in smaller young generations are called scavenges
- No tracing of objects in the old generation
- Fast and frequent

GC in V8

Most Objects die young

- Once a semi-space is filled, these young generation objects are moved to a new semi-space
- Once an object has been moved twice, it is moved to the old generation
- Old generation objects are considered to be long-lived
- Scavenges often take < 1 ms

GC in V8

- Major GC is performed when the old generation grows beyond a limit
- Uses a mark-and-sweep collection
- Goal is to keep marking < 5 ms
- Memory is freed after marking and sweeping the entire old generation

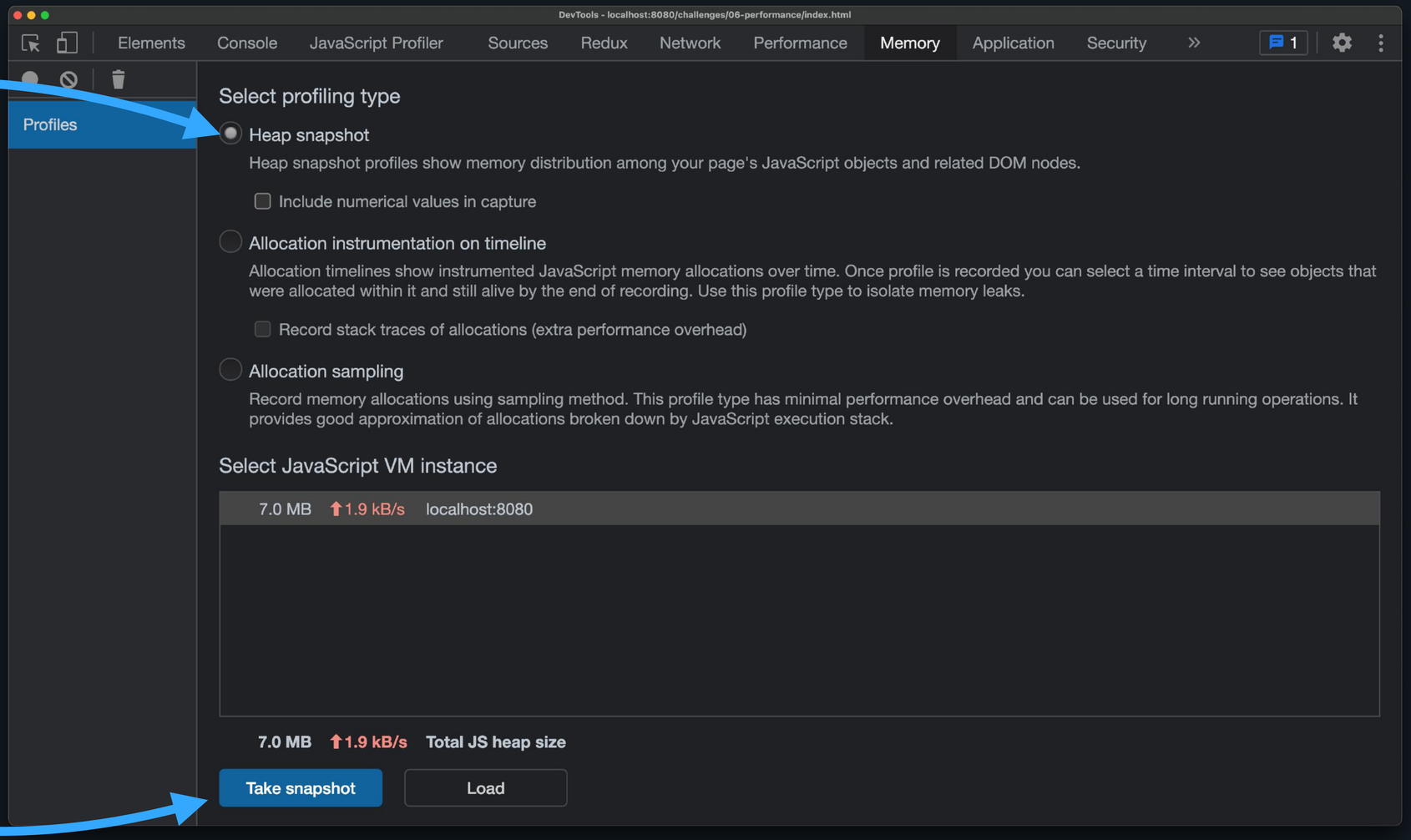
mark-and-sweep

- First, mark reachable objects
- Then, sweep over memory and recycle objects that are unmarked
- Mark phase: follow reference chains to mark all reachable objects
- Sweep page: sequential pass over memory to recycle all unmarked objects

Memory Panel

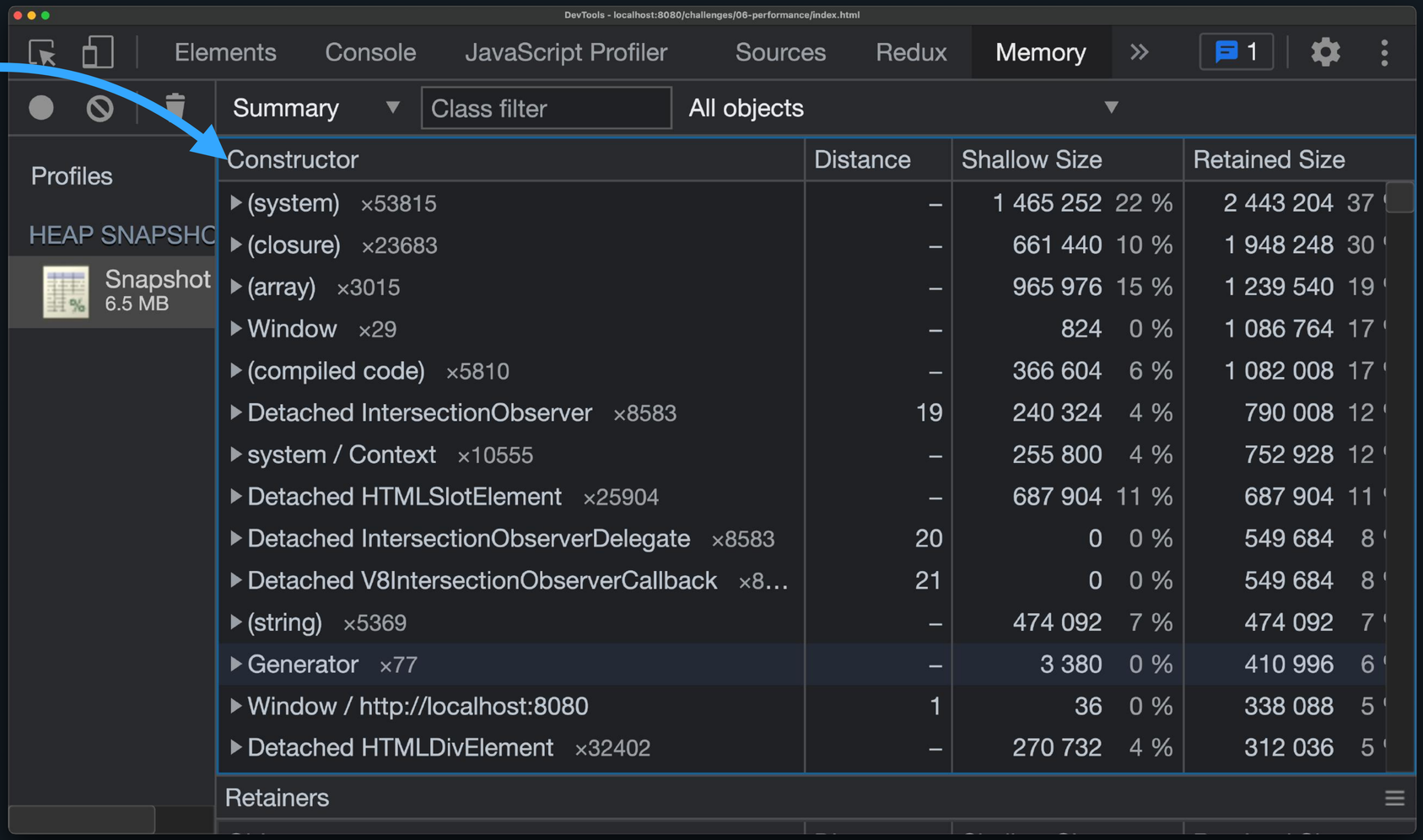
Select the “Heap snapshot” profile type

Click “Take snapshot”



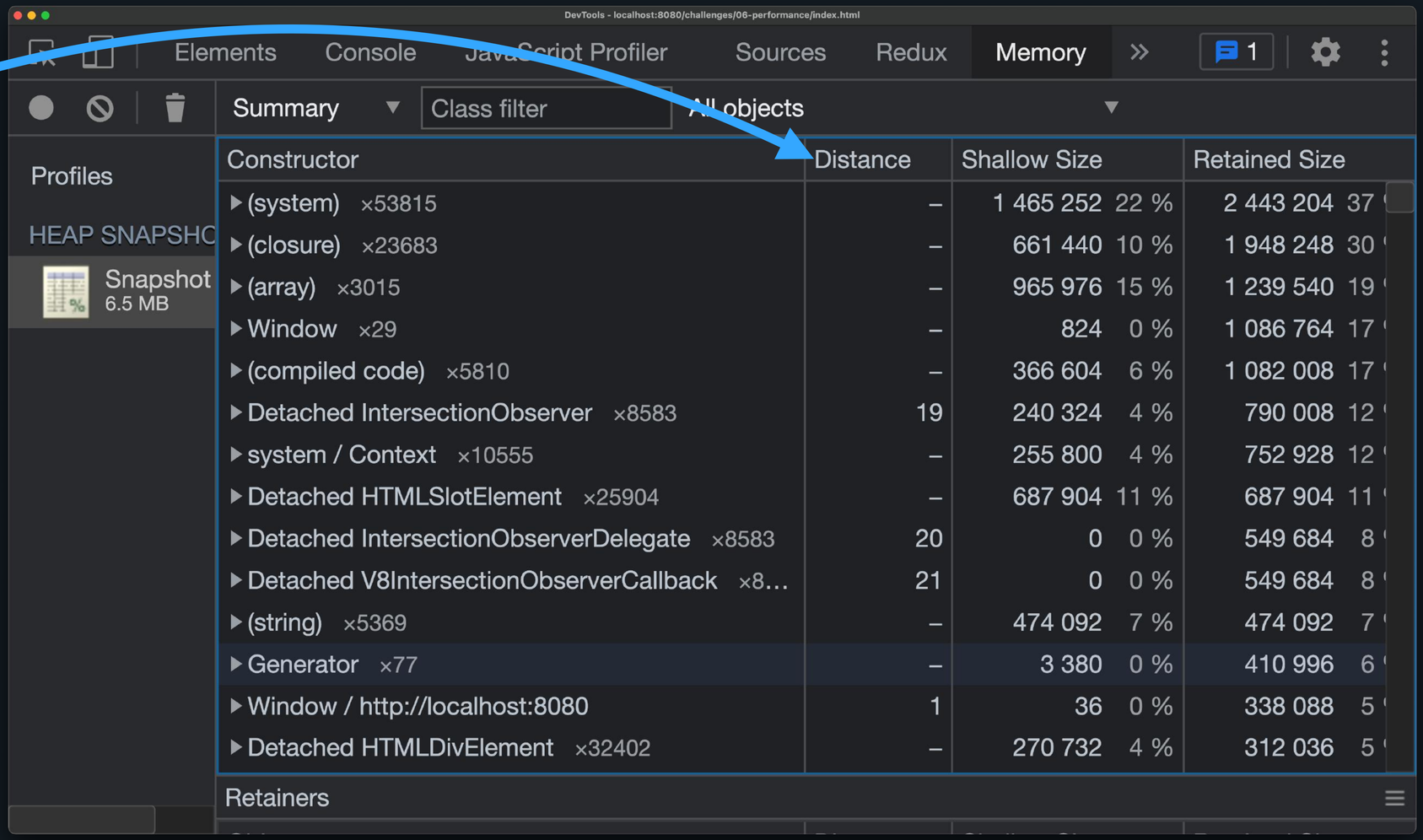
Memory Panel

All objects created using the constructor



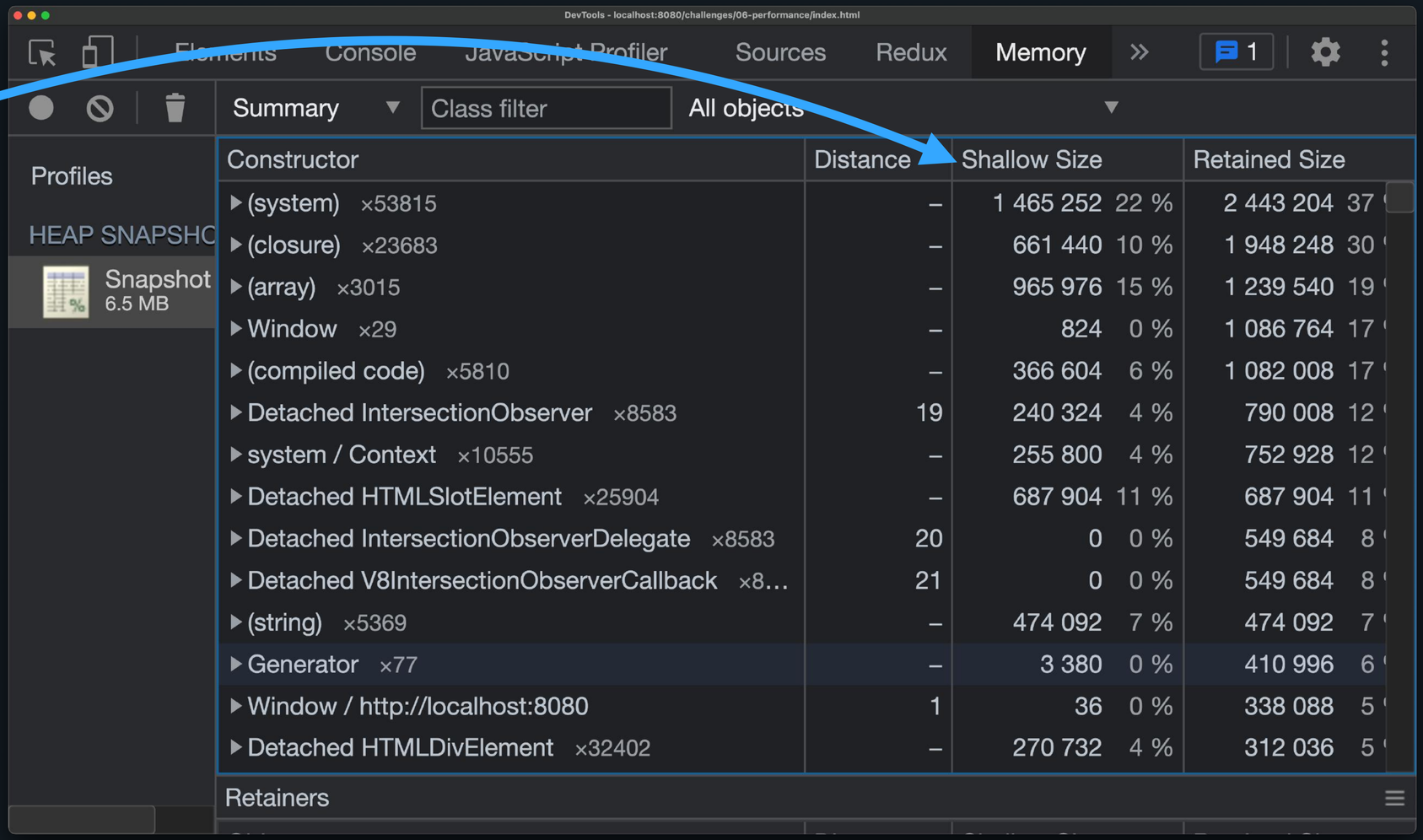
Memory Panel

The distance from the root object using the shortest path of nodes



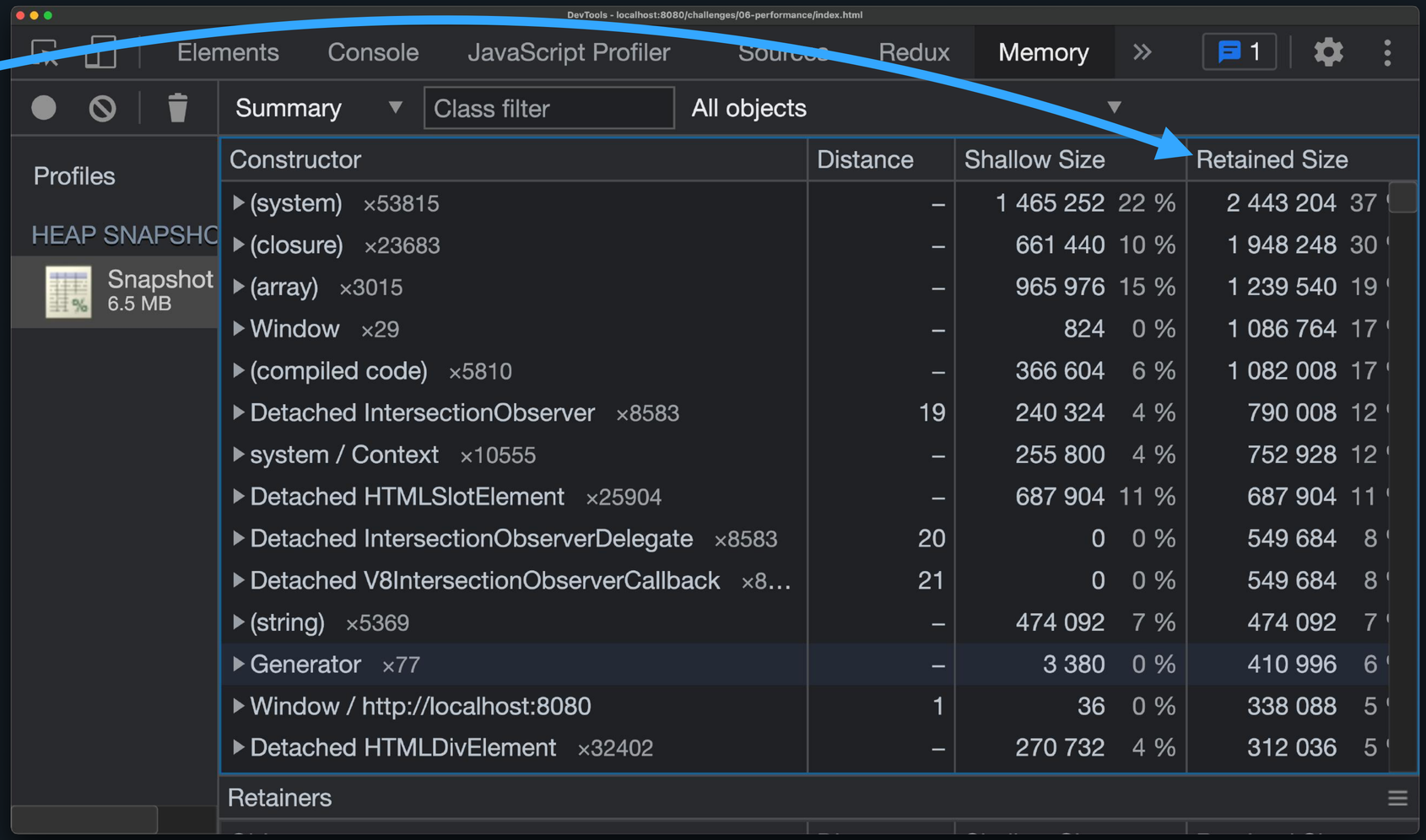
Memory Panel

Sum of shallow sizes of Objects



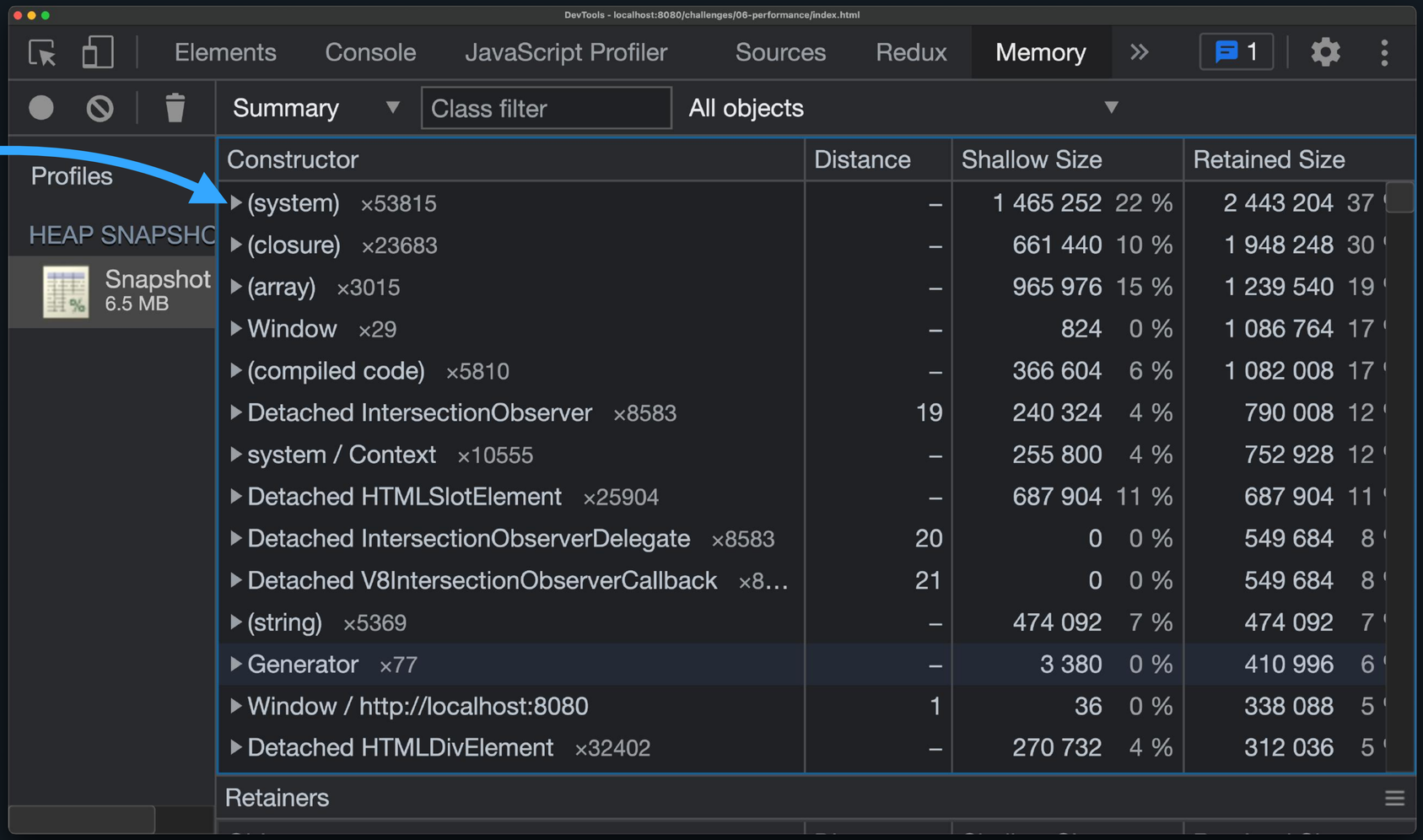
Memory Panel

Maximum retained size of Objects



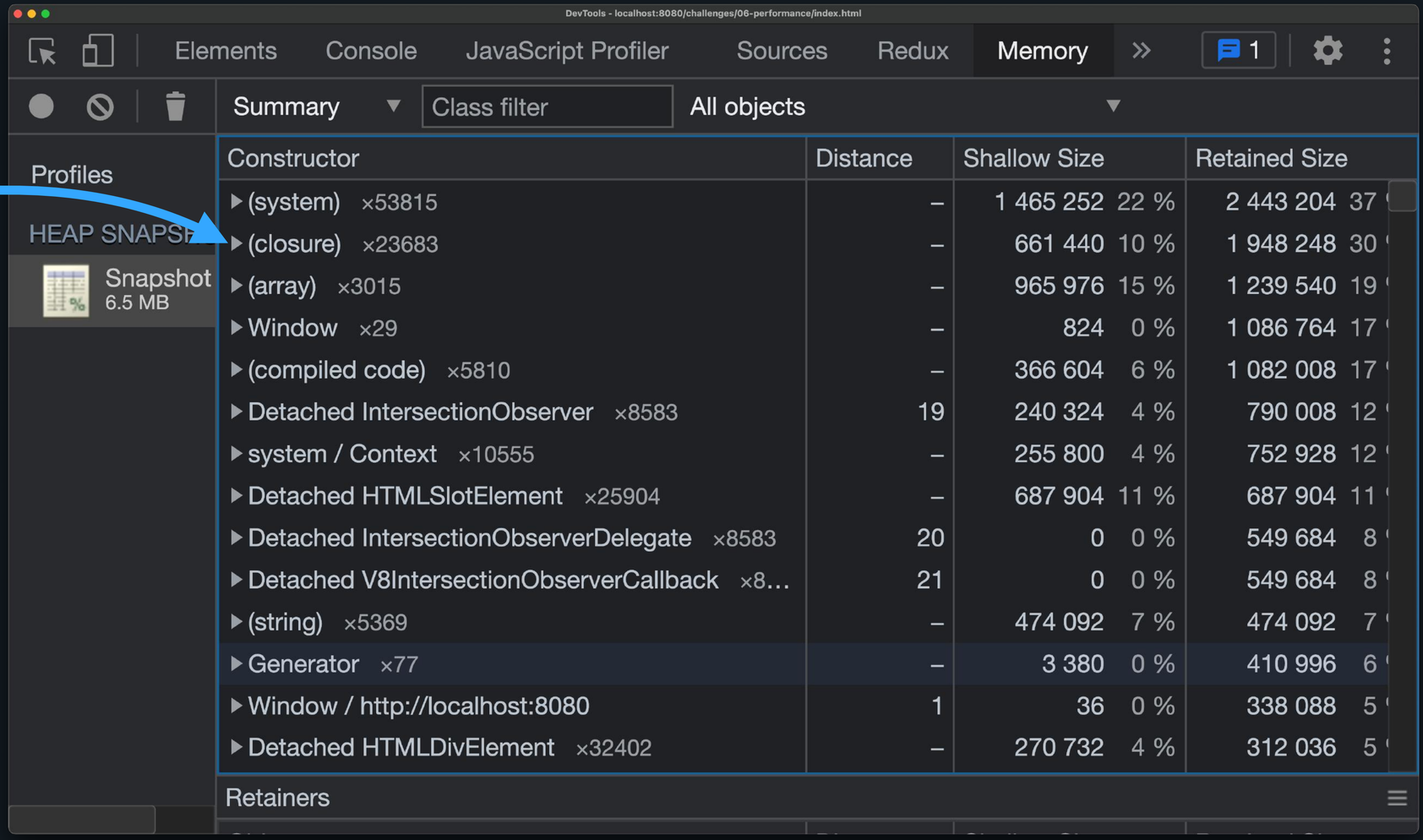
Memory Panel

Intermediate objects between root (window or globalThis) and an object referenced by it.



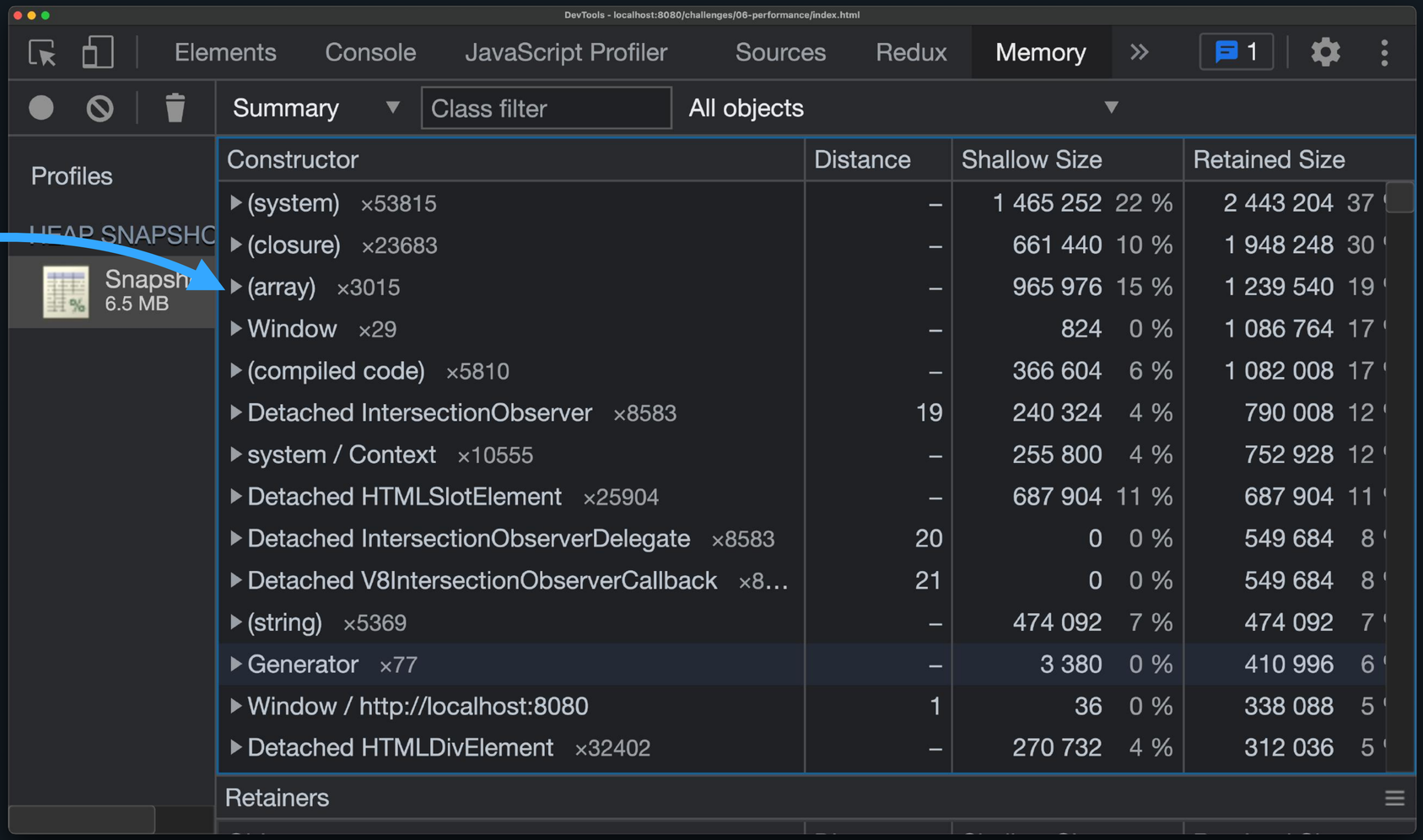
Memory Panel

References to a objects through function closures



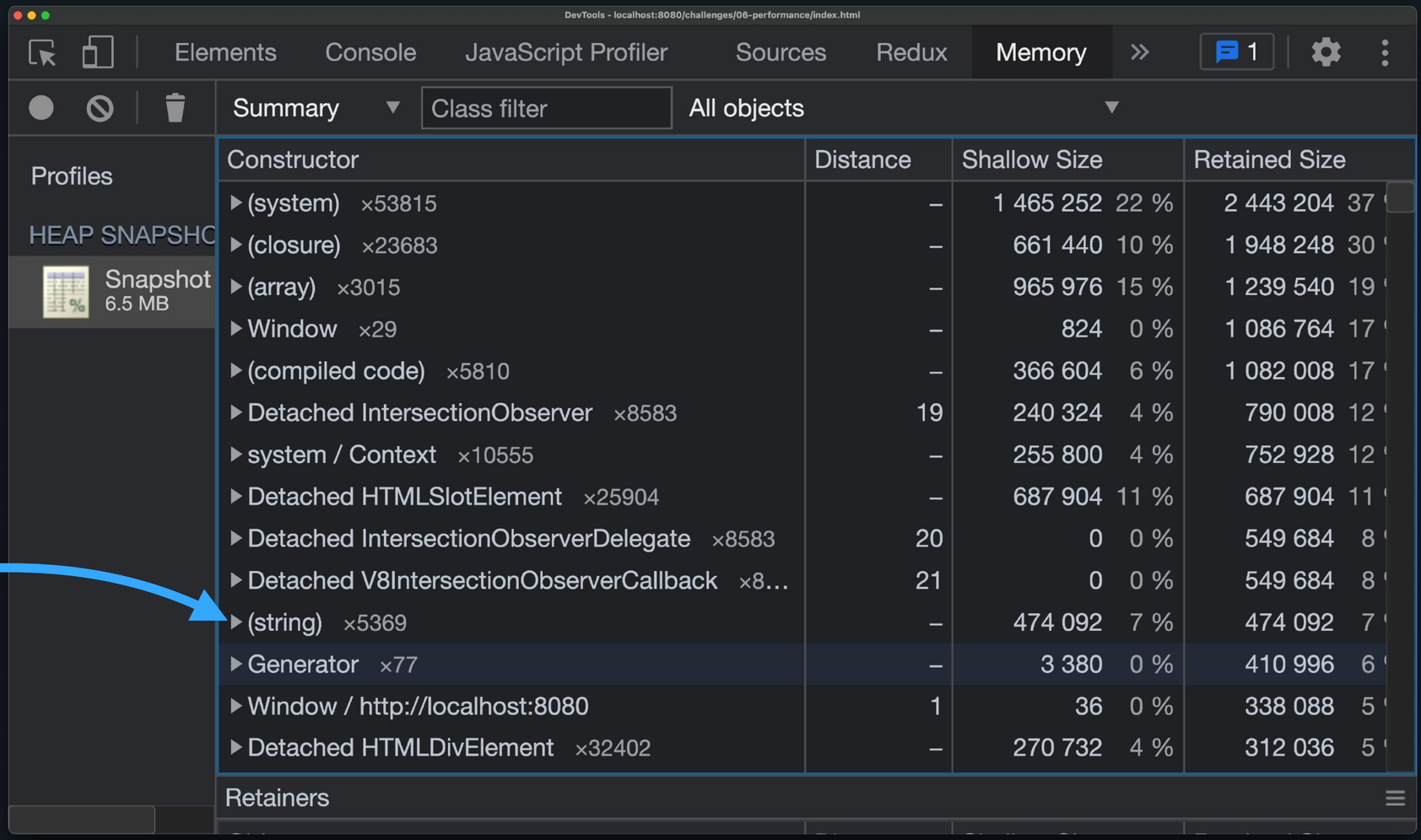
Memory Panel

Objects with properties referencing an Array



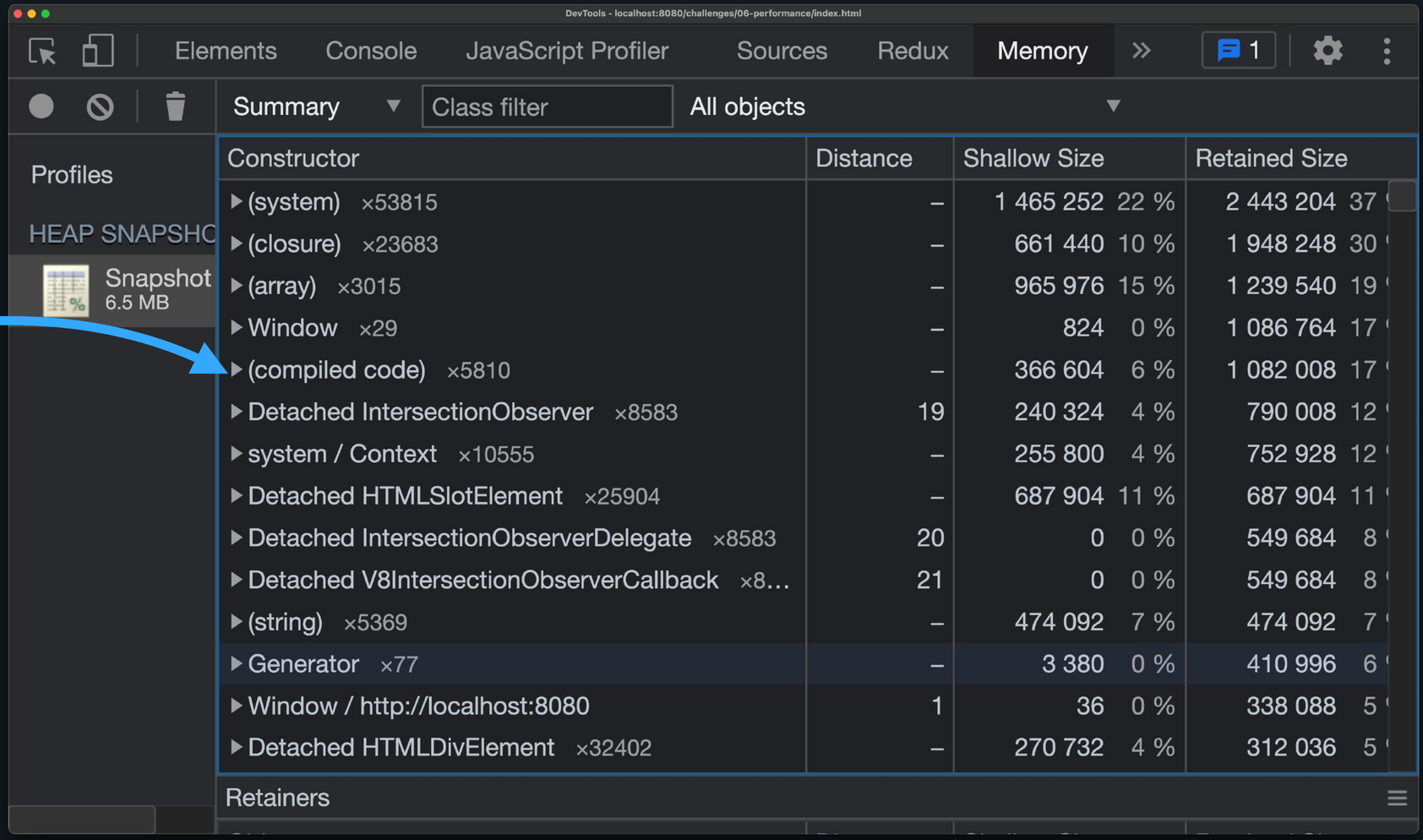
Memory Panel

Objects with properties referencing an String



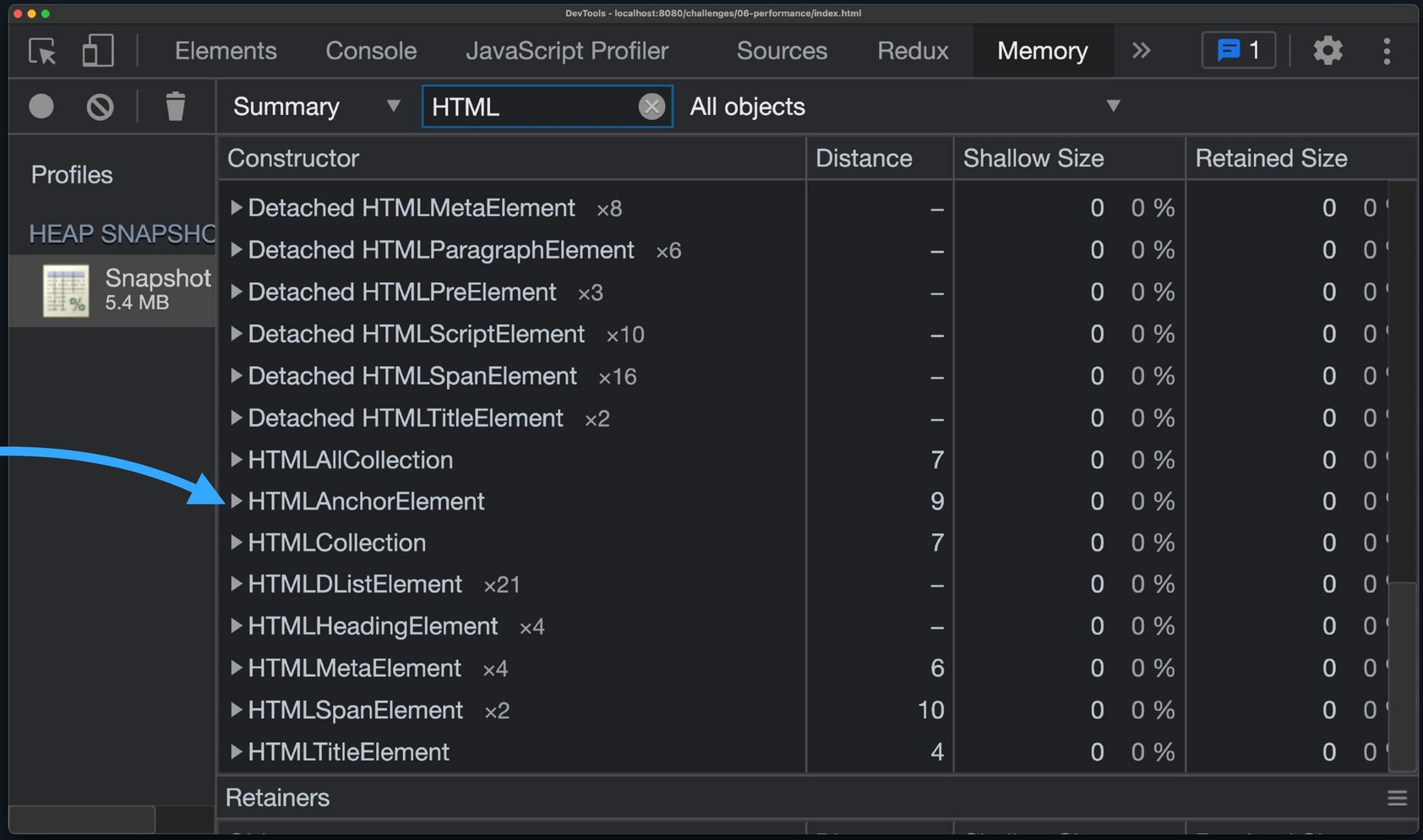
Memory Panel

Everything related to compiled code (your app)



Memory Panel

References to elements or document objects referenced by the code of our app



Memory Heap Snapshot

1. Run **`npm start`** and open **`08 Memory`** in Chrome.
2. Open the **Memory** panel.
3. Ensure the console is clear and there are no breakpoints.
4. Click the **Collect Garbage** (trash icon) button.
5. Click the **Take Snapshot** button.
6. Note the **Distance**, **Shallow Size** and **Retained Size** for the (compiled code).

General Tips

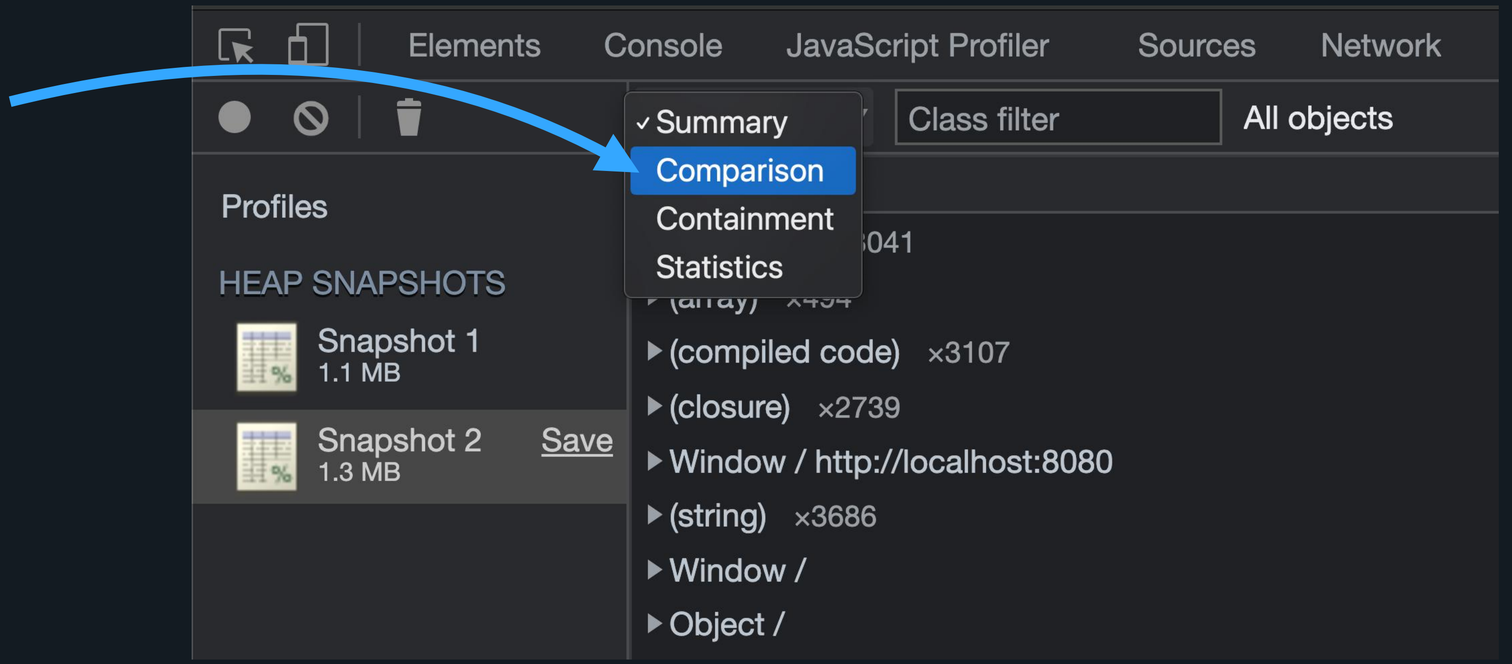
- [x] Clear console
- [x] Remove all breakpoints
- [x] Manually perform GC

Finding Memory Leaks

- [x] Take a snapshot before performing an action
- [x] Interact with page that may be causing a memory leak
- [x] Undo that operation (go back to original state)
- [x] Repeat the interaction a few times
- [x] Take a second snapshot

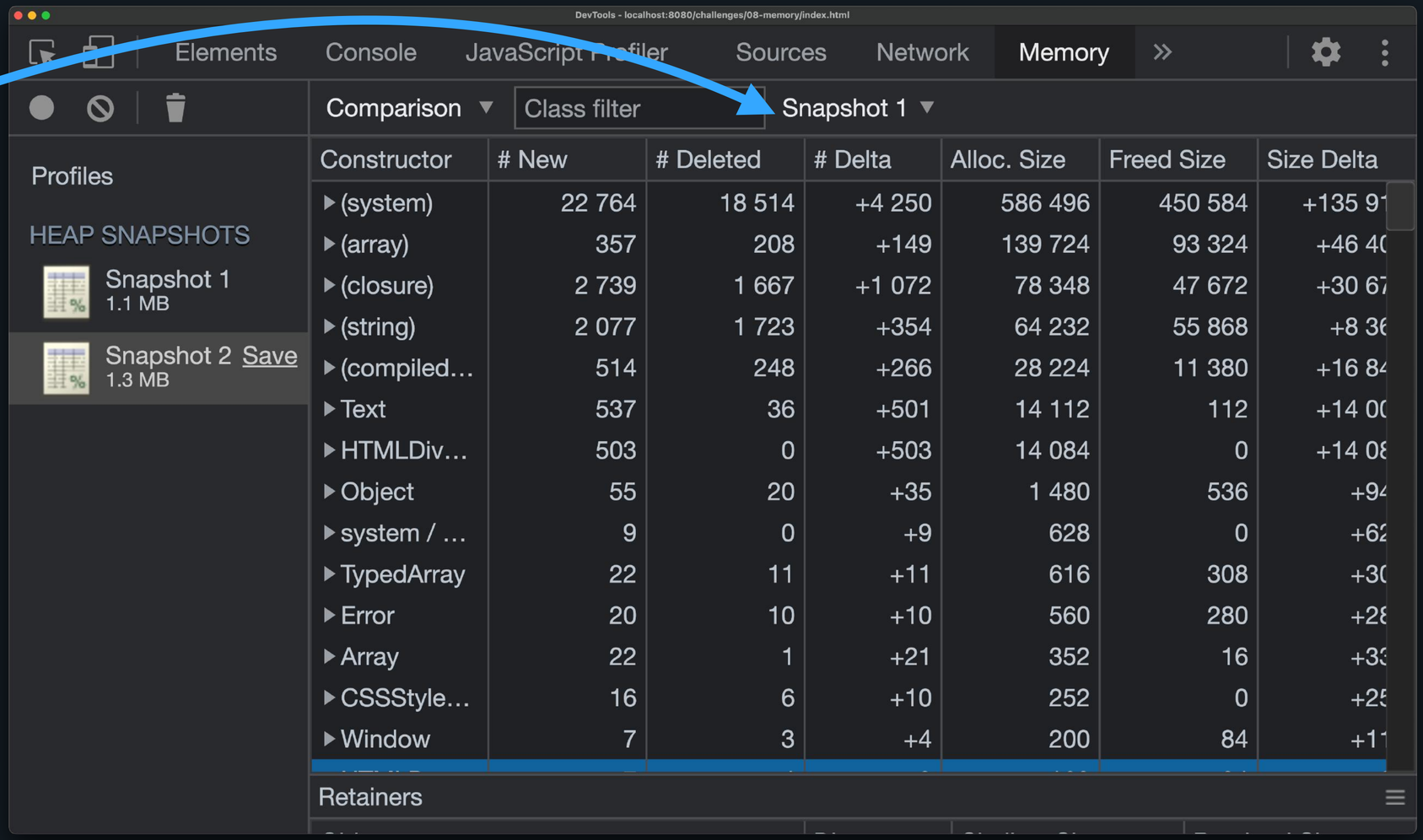
Snapshot Comparison

Switch to the comparison view



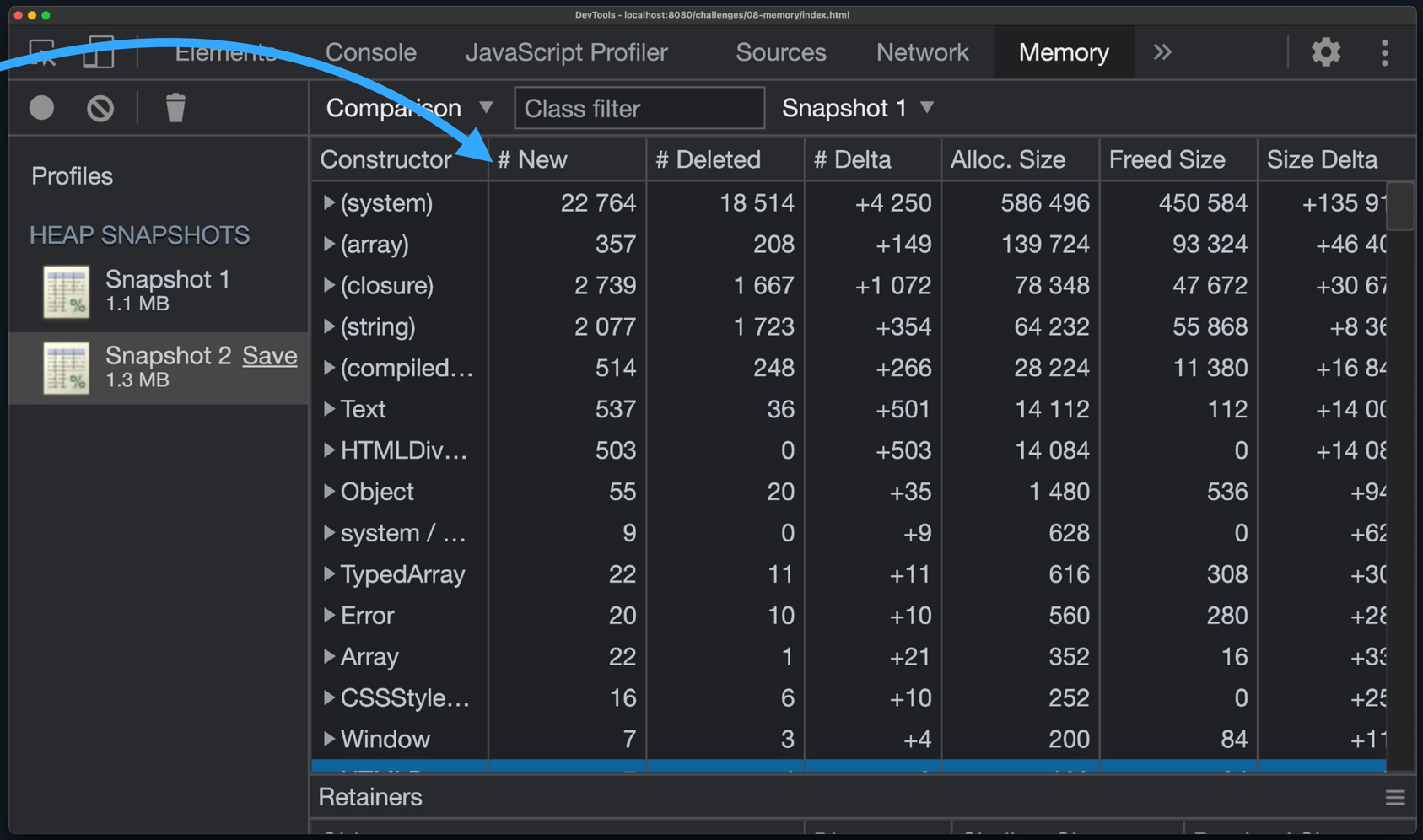
Snapshot Comparison

Note that we are comparing **Snapshot 2** to **Snapshot 1**



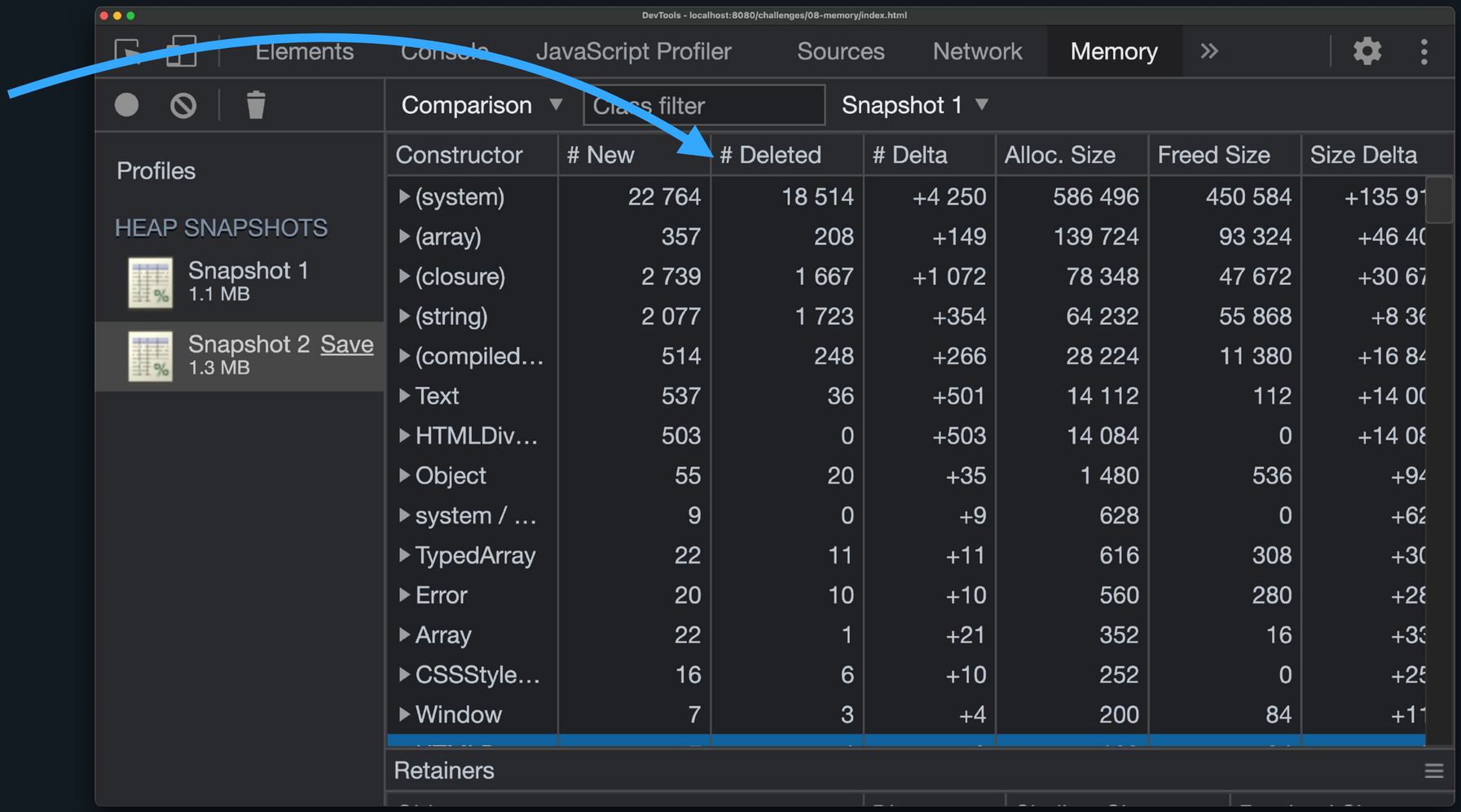
Snapshot Comparison

The number of new objects



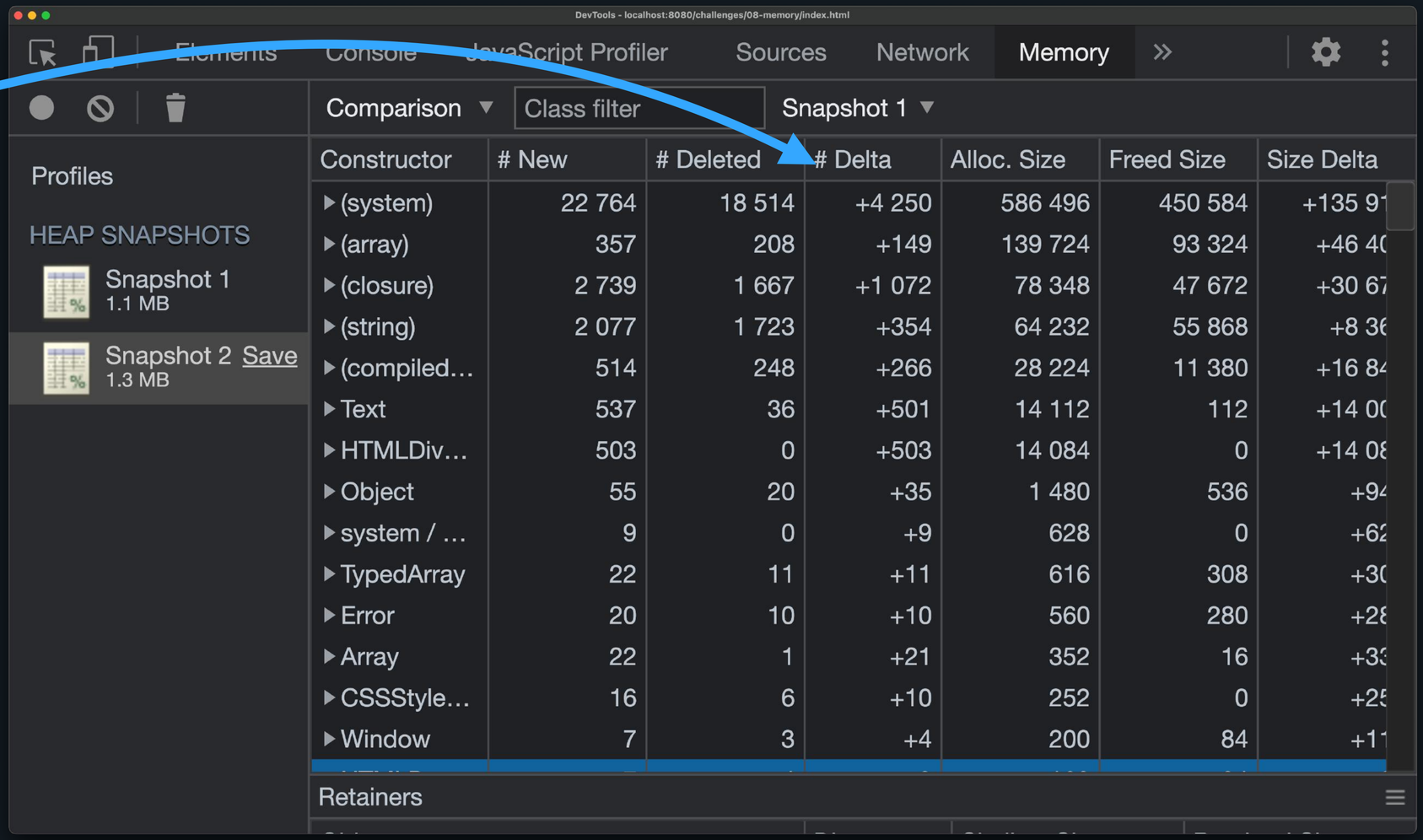
Snapshot Comparison

The number of deleted objects



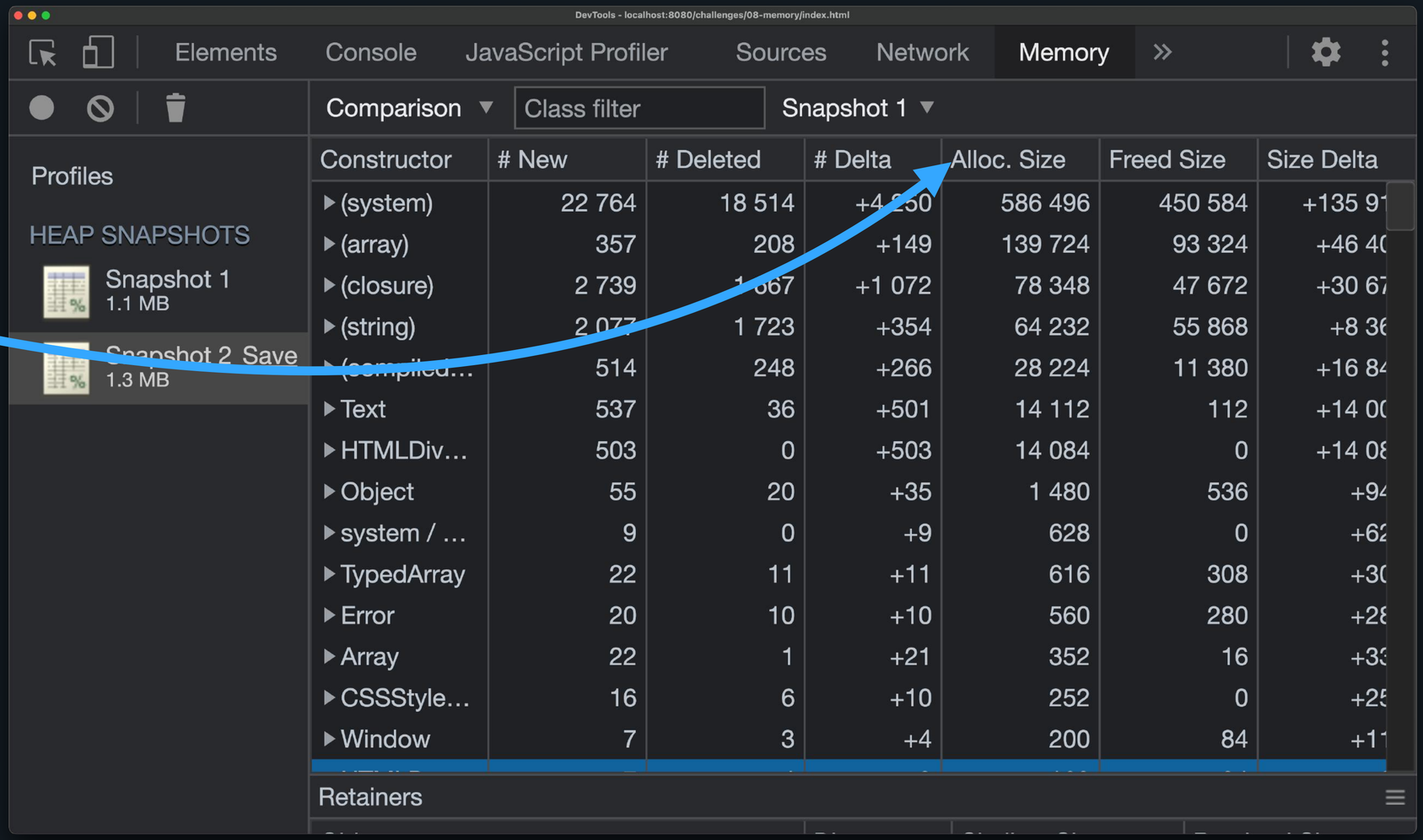
Snapshot Comparison

The difference in the number of objects between snapshots



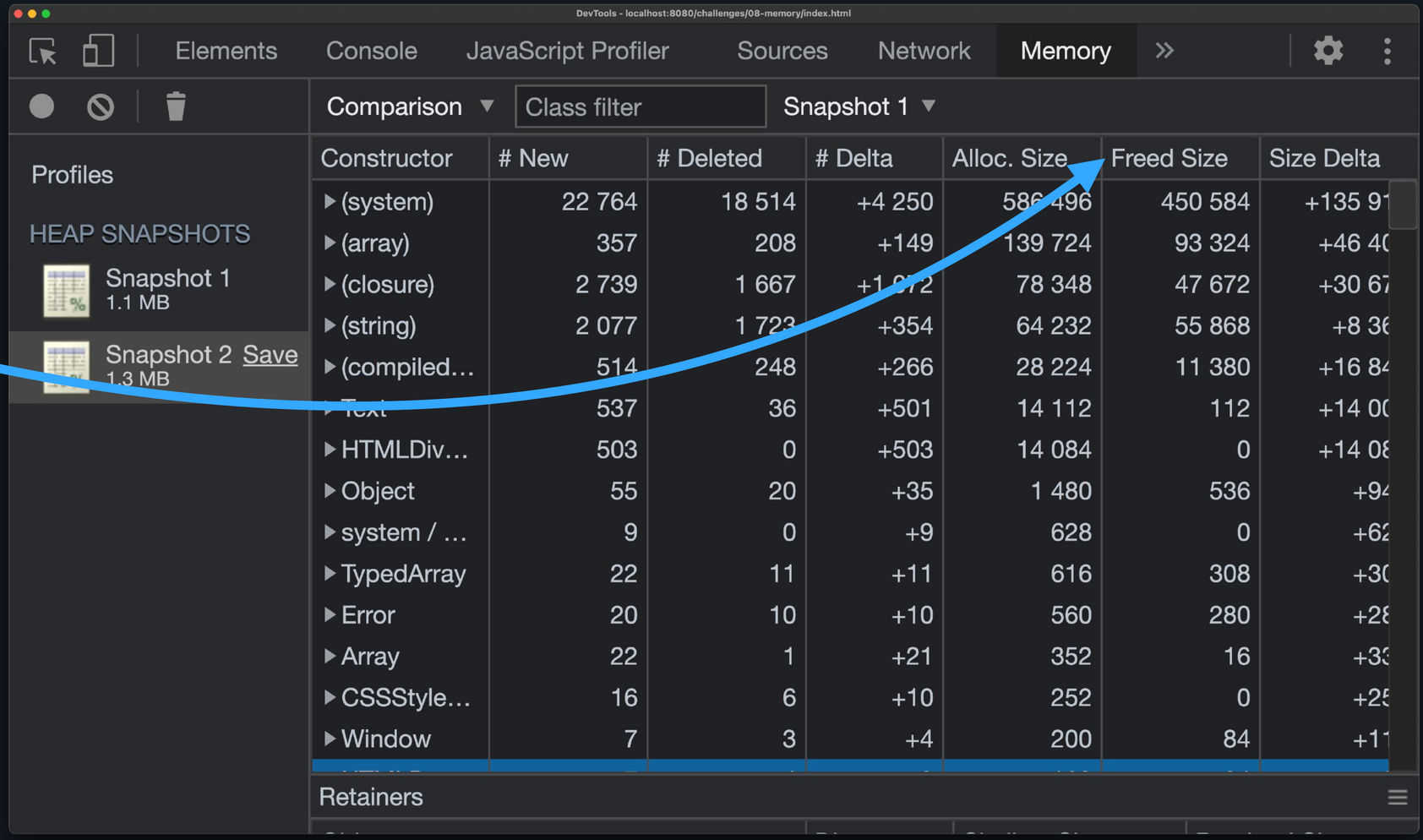
Snapshot Comparison

Additional memory allocated between snapshots



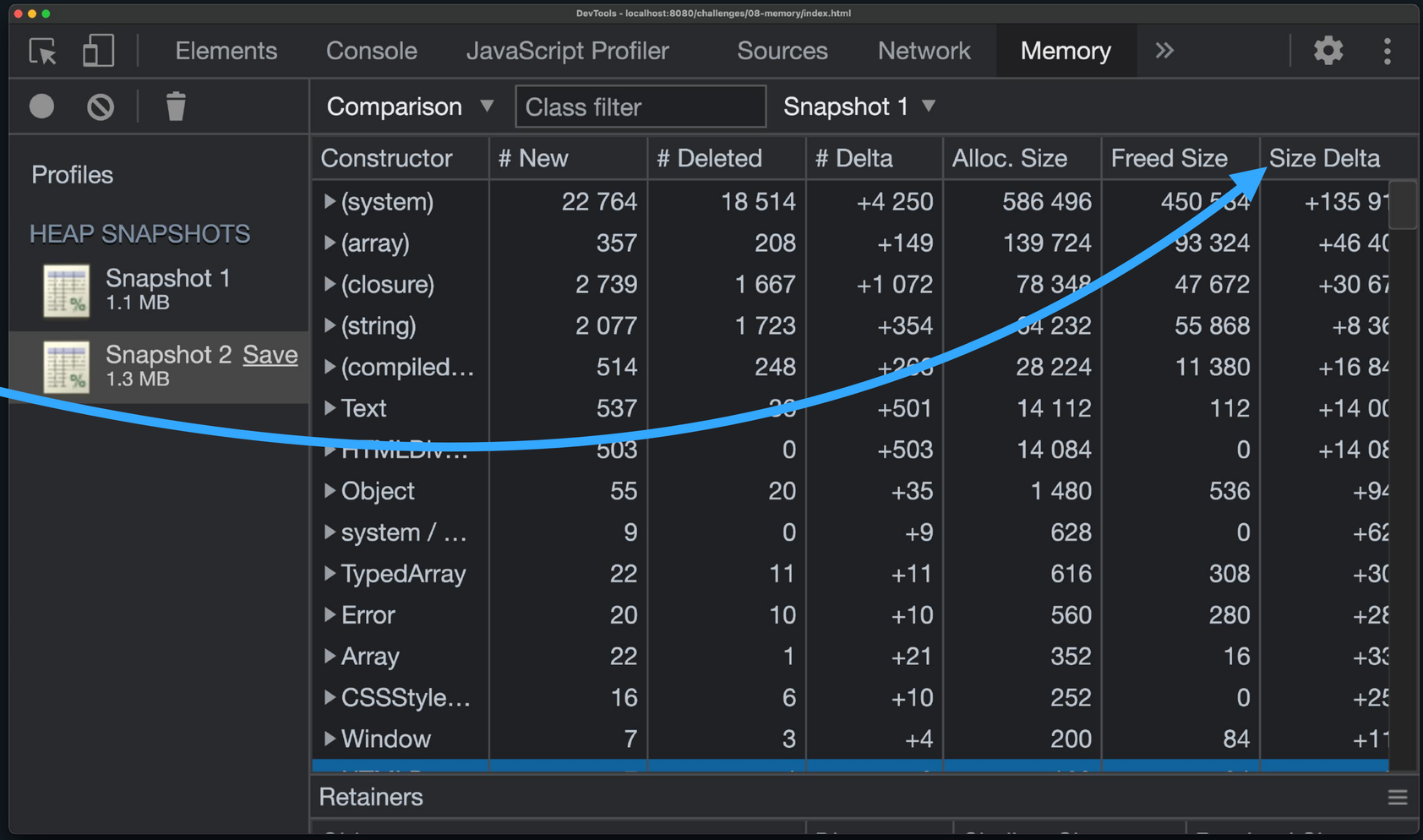
Snapshot Comparison

Memory freed up between snapshots



Snapshot Comparison

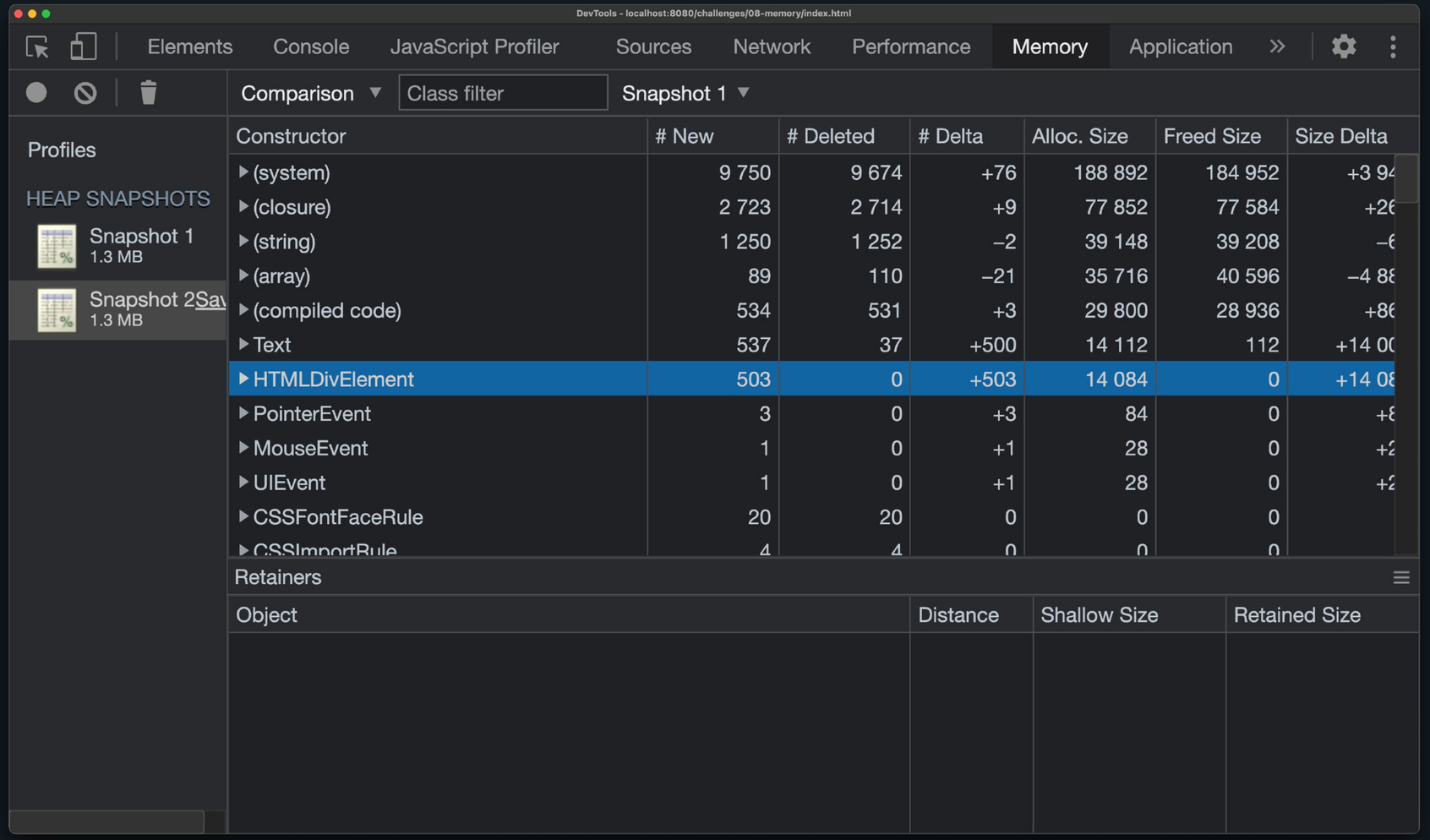
The size difference between snapshots



Snapshot Comparison

1. Run **`npm start`** and open **`08 Memory`** in Chrome.
2. Open the **Memory** panel.
3. Click the **Take Snapshot** button to take a new snapshot.
4. Click the **Grow** button, and then take another snapshot (using the record icon).
5. Change to the **Comparison** view.
6. What objects in memory increased the most?

Snapshot Comparison



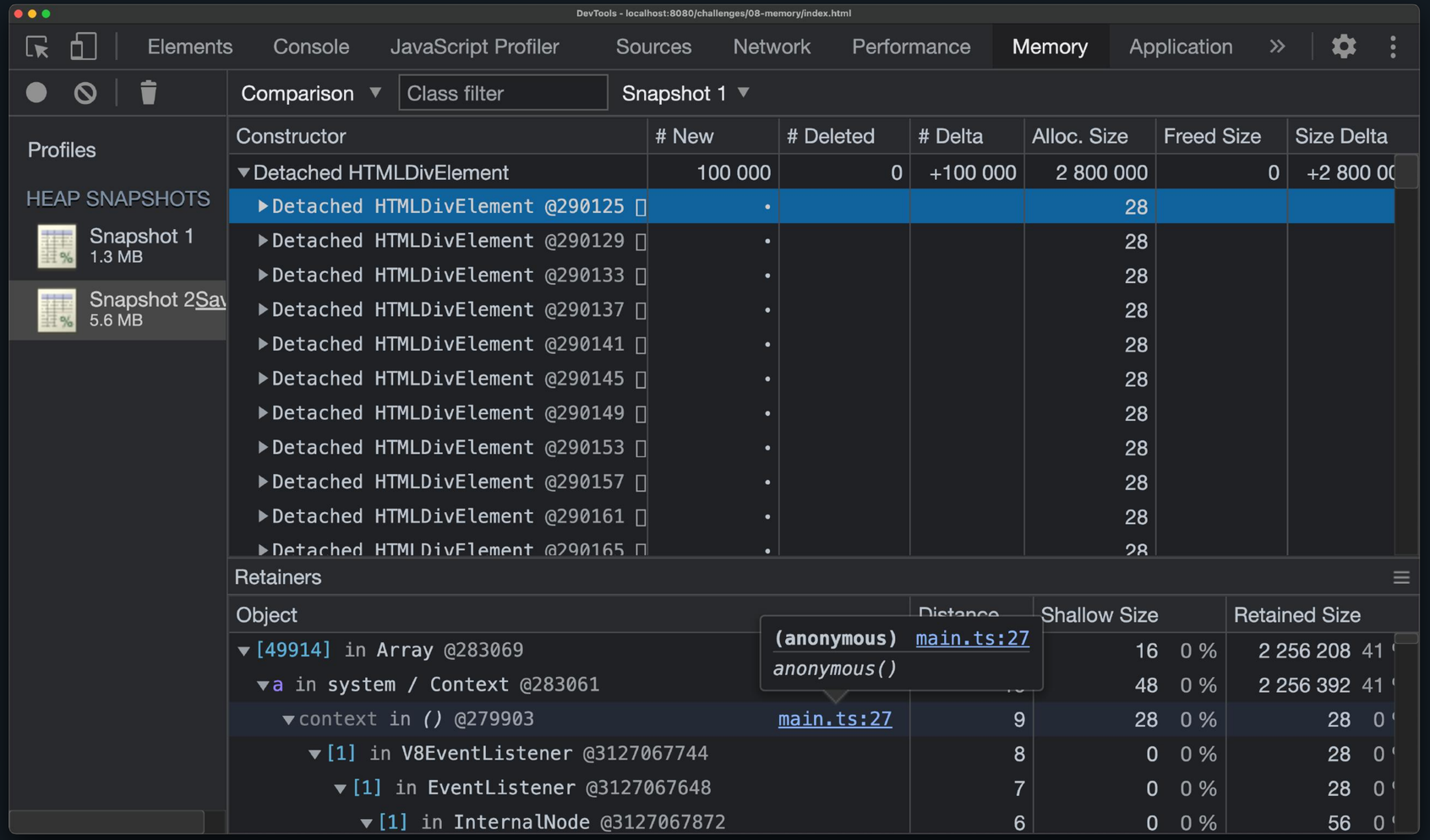
Detached DOM Nodes

- DOM Nodes that have been removed from the DOM
- JS reference to DOM still exists
- GC cannot recycle objects due to hanging references

Detached DOM Nodes

1. Run **`npm build`** followed by **`npm serve`**, then open **`08 Memory`** in Chrome.
2. Open the **Memory** panel.
3. Click the **Take Snapshot** button to take a new snapshot.
4. Click the **Detach** button a few times, and then take another snapshot.
5. Change to the Comparison view.
6. Identify the **Detached HTMLDivElement** objects.
7. Open the challenges/08-memory/main.ts file and examine the **`detach()`** function. Why are the DOM nodes detached?

Detached DOM Nodes

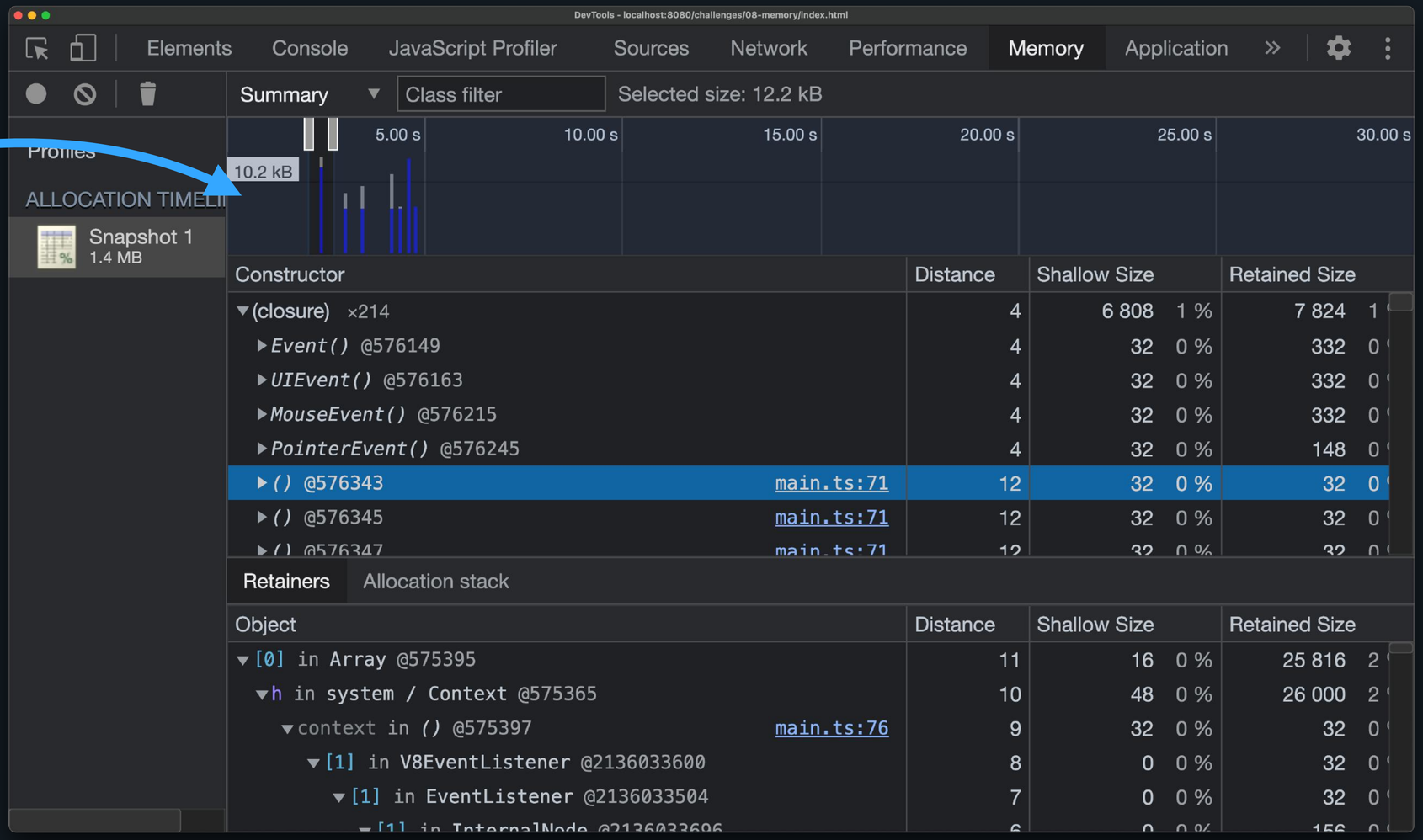


Allocation Instrumentation

- Take a heap snapshot every 50 ms
- Identify object allocation and retention over time
- Valuable for identify memory leaks!
- Four

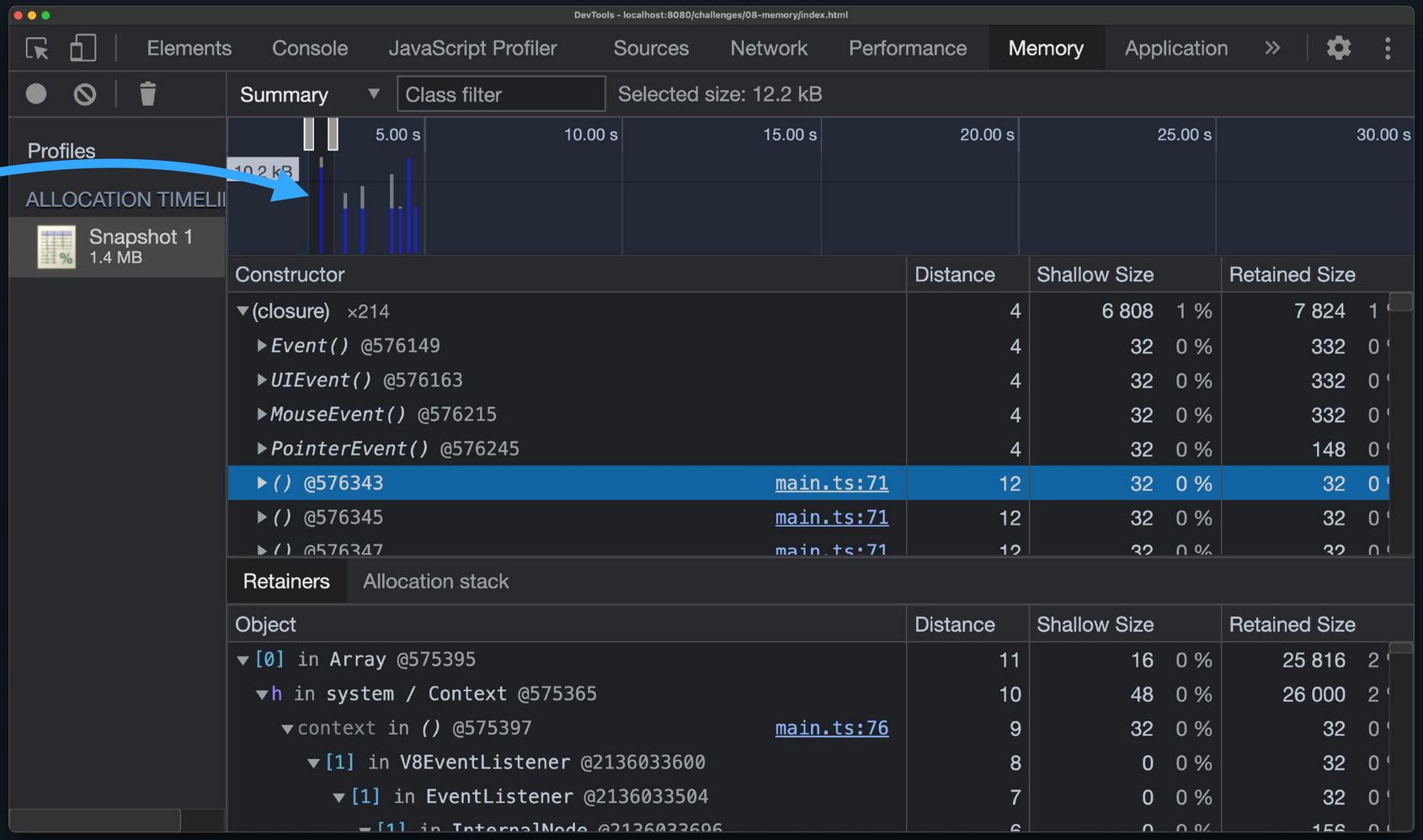
Allocation Instrumentation

Identify in timeline where objects are created and retained



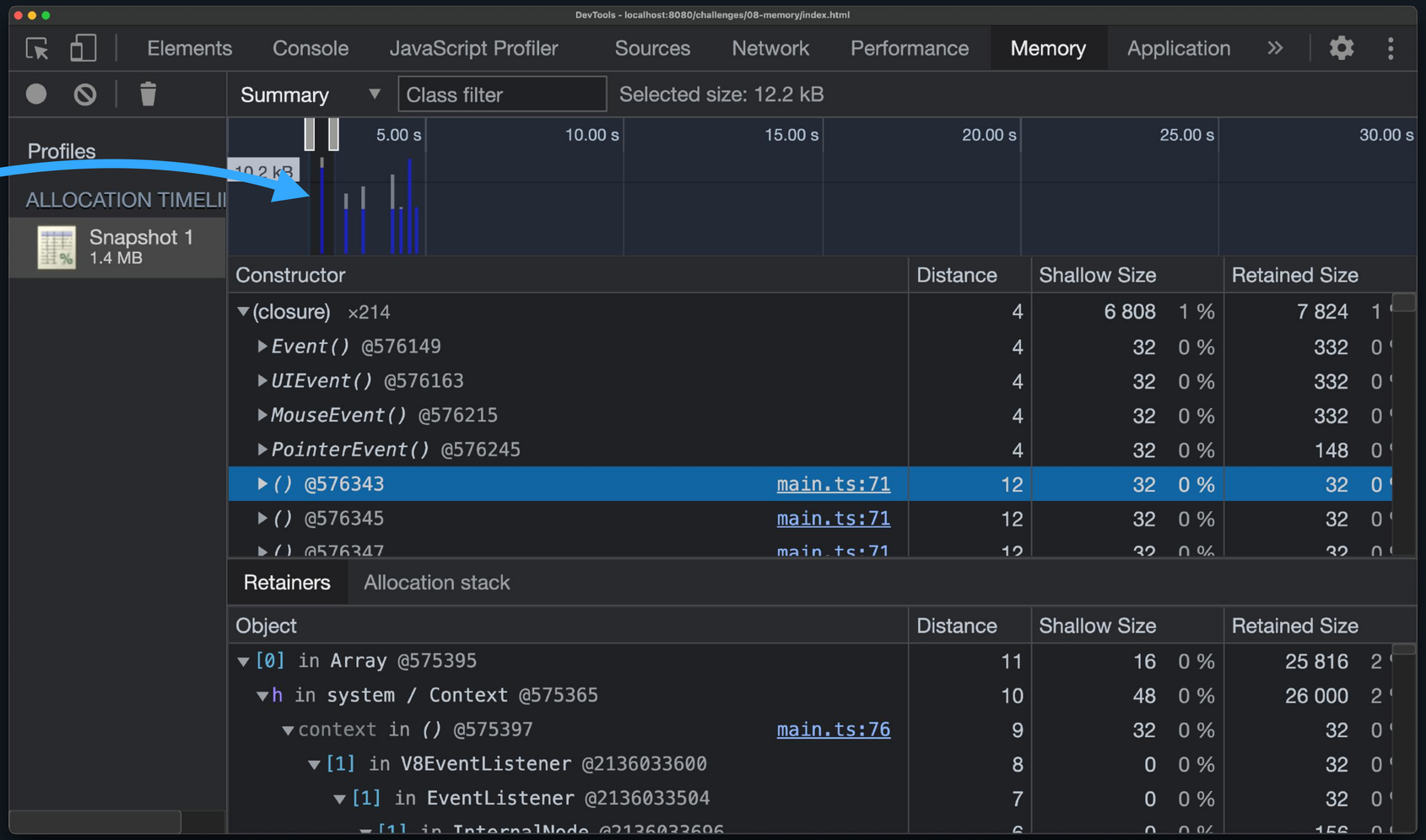
Allocation Instrumentation

Height of each bar represents the size of the memory allocation



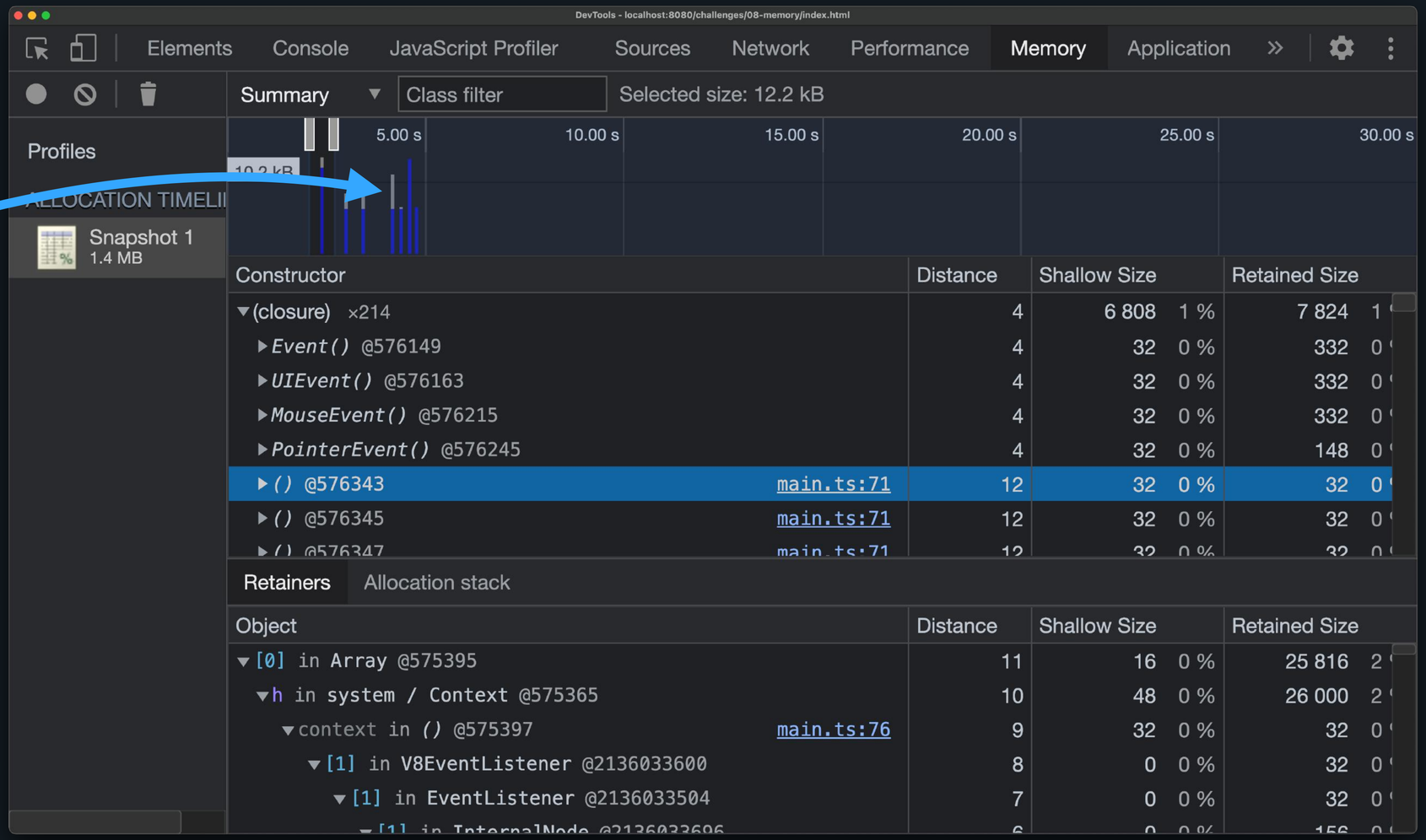
Allocation Instrumentation

Blue bar represents objects that are still retained in the final heap snapshot



Allocation Instrumentation

Gray bar represents objects that have been allocated and garbage collected

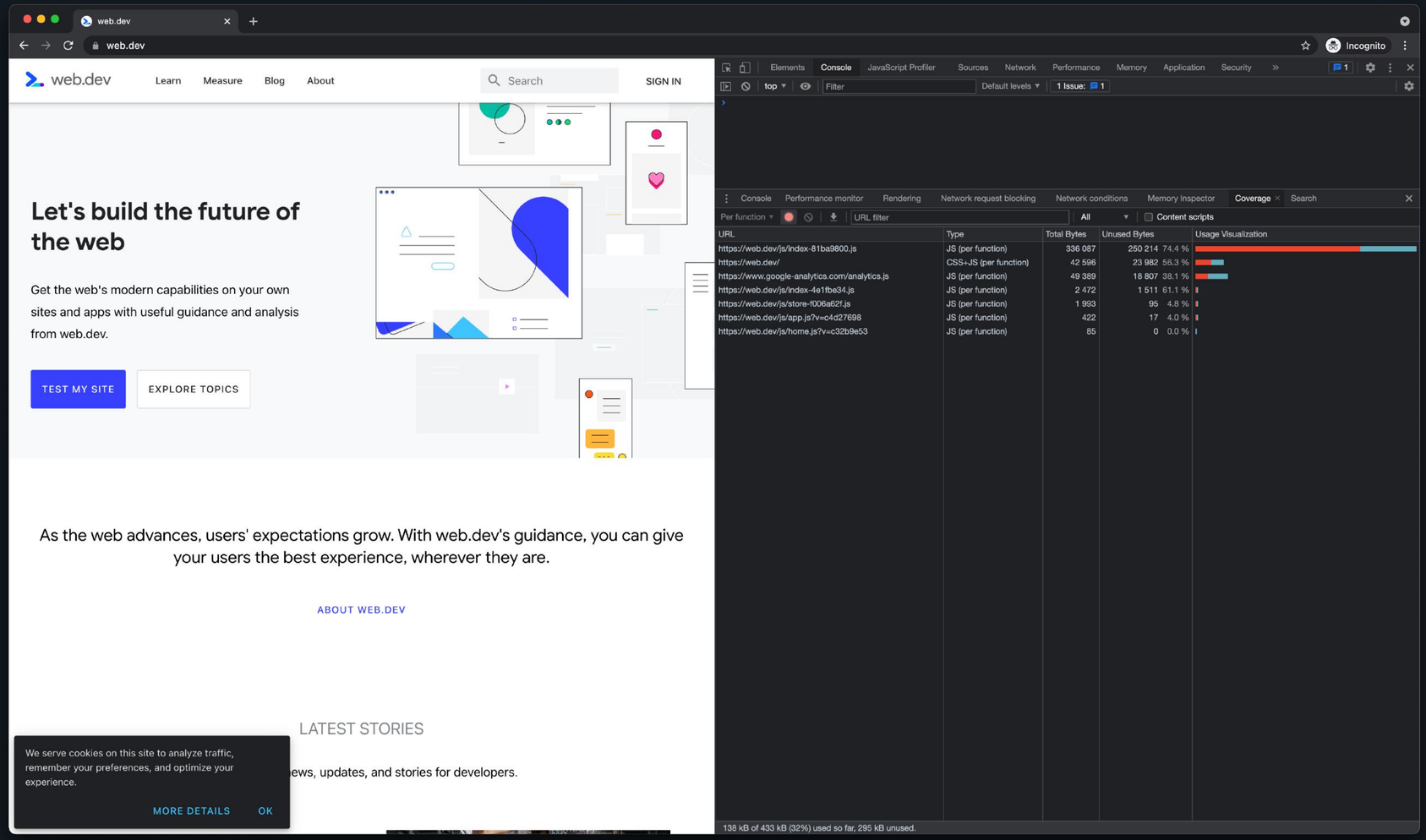


Allocation Instrumentation

1. Run **`npm build`** followed by **`npm serve`**, then open **`08 Memory`** in Chrome.
2. Open the **Memory** panel.
3. Choose the **Allocation instrumentation on timeline** option and click **Start**.
4. Click the **Closure** button a few times.
5. Stop the recording.
6. Choose the **(closure)** Constructor global object and expand the child nodes.
7. Click the source link to identify where the memory leak is coming from.
8. Open the challeges/08-memory/main.ts file and examine the **`createClosure()`** function. What is causing this memory leak?

DevTools: Coverage

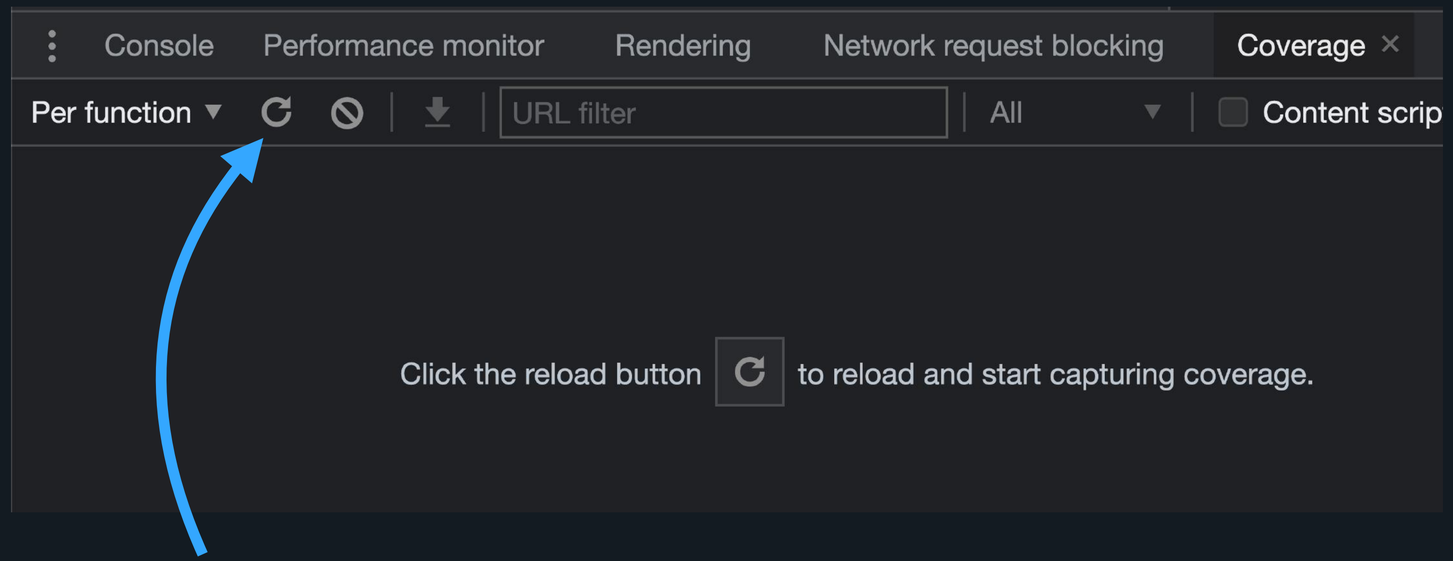
DevTools: Coverage



DevTools: Coverage

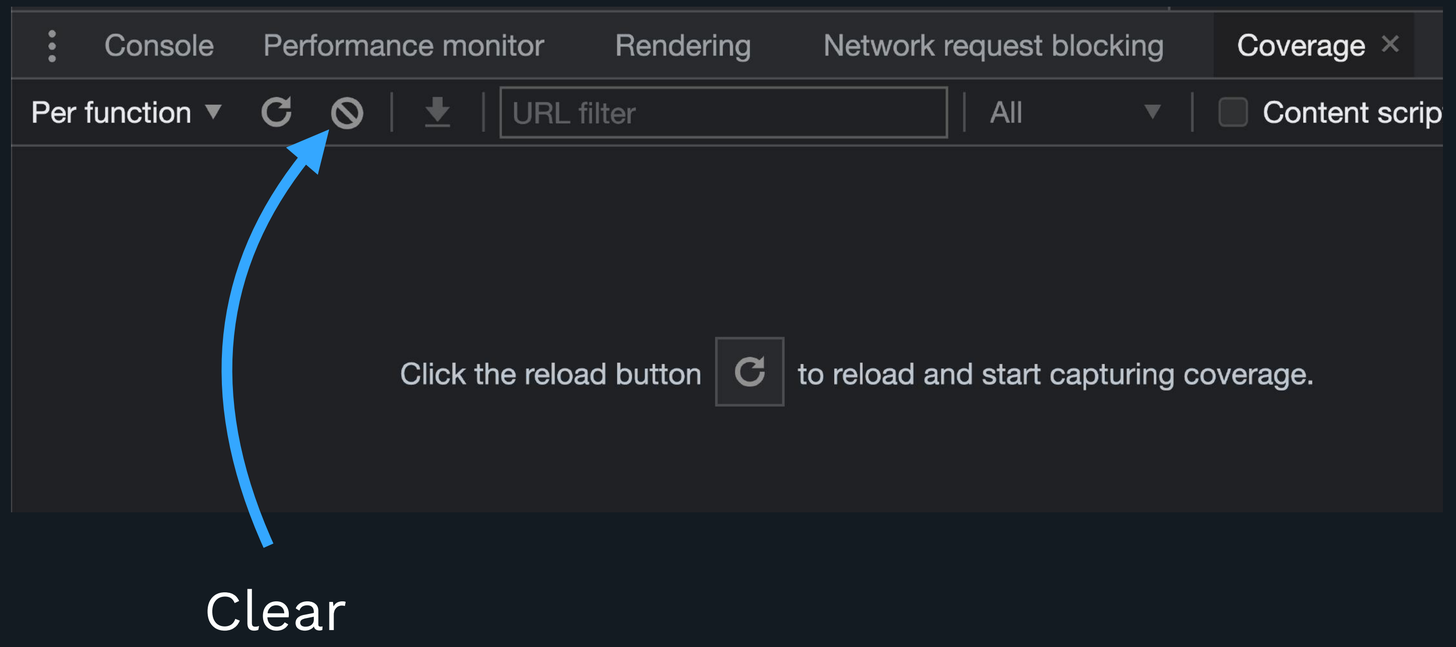
- Resources containing unused code
- Identify unused JS
- Identify unused CSS

DevTools: Coverage

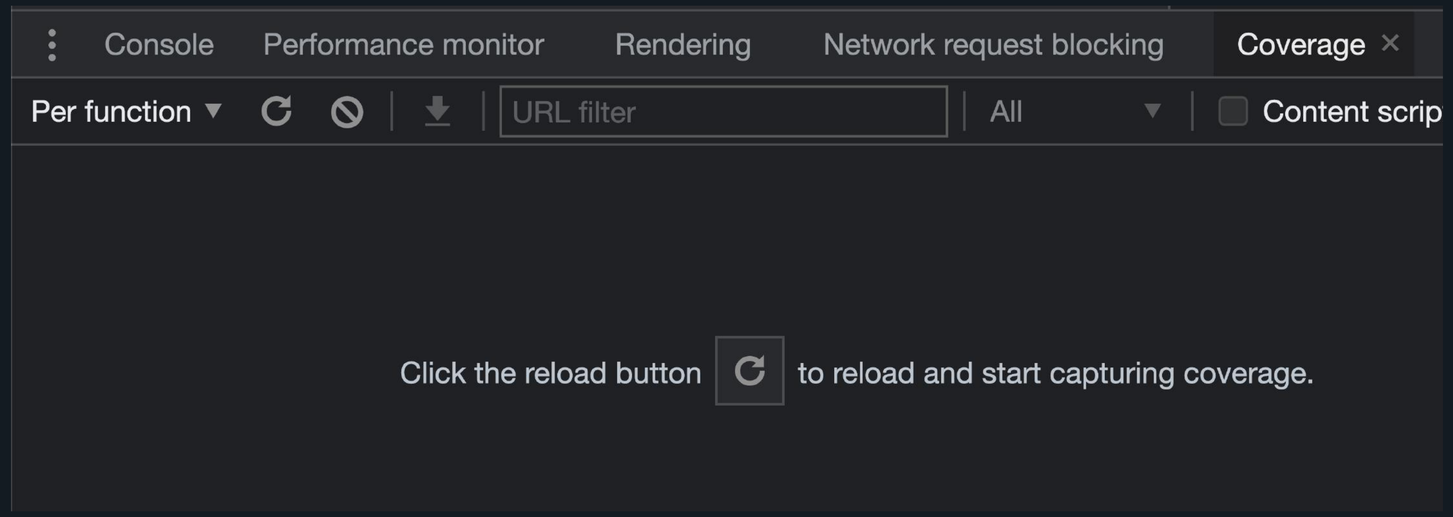


Reload and start capturing coverage

DevTools: Coverage

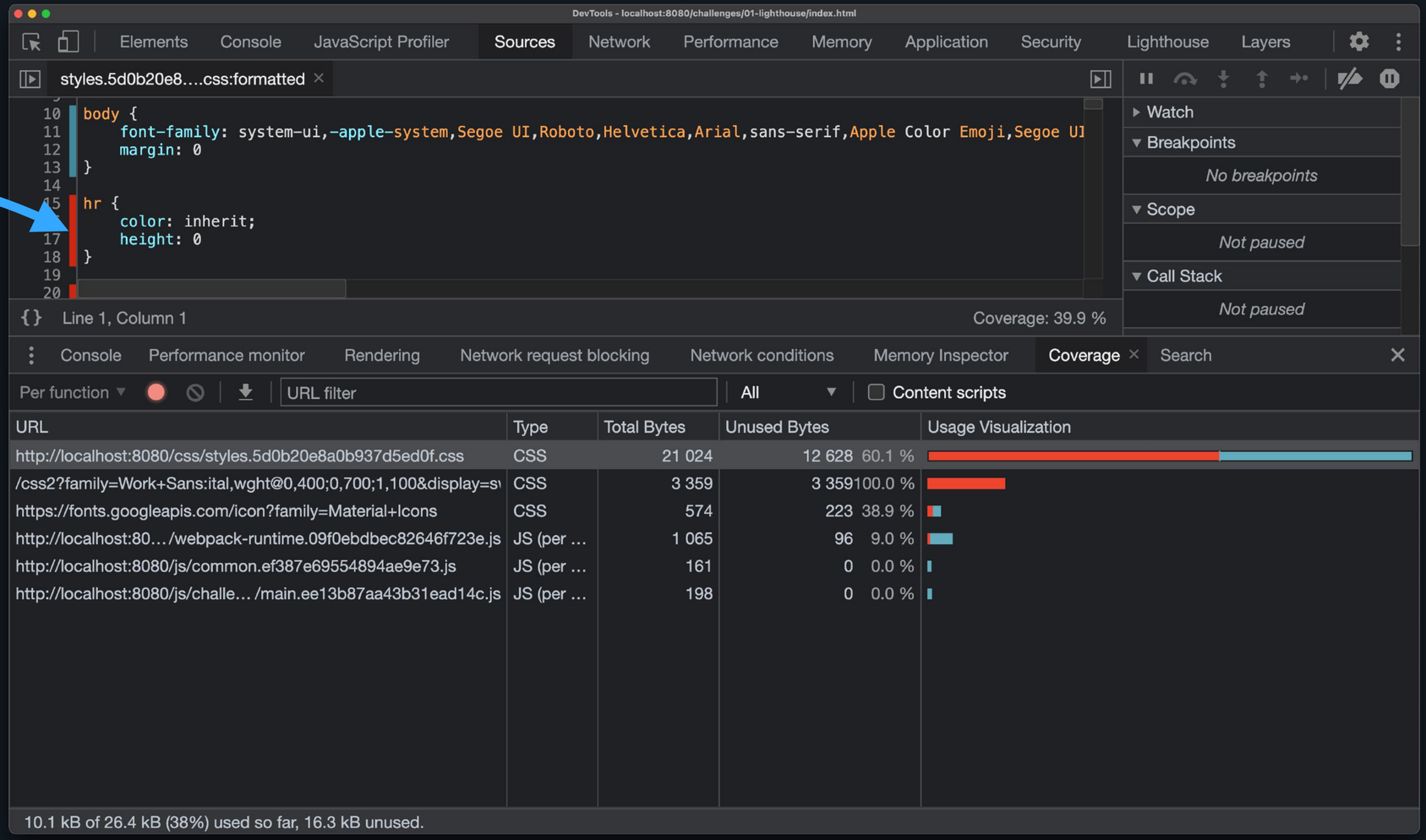


DevTools: Coverage



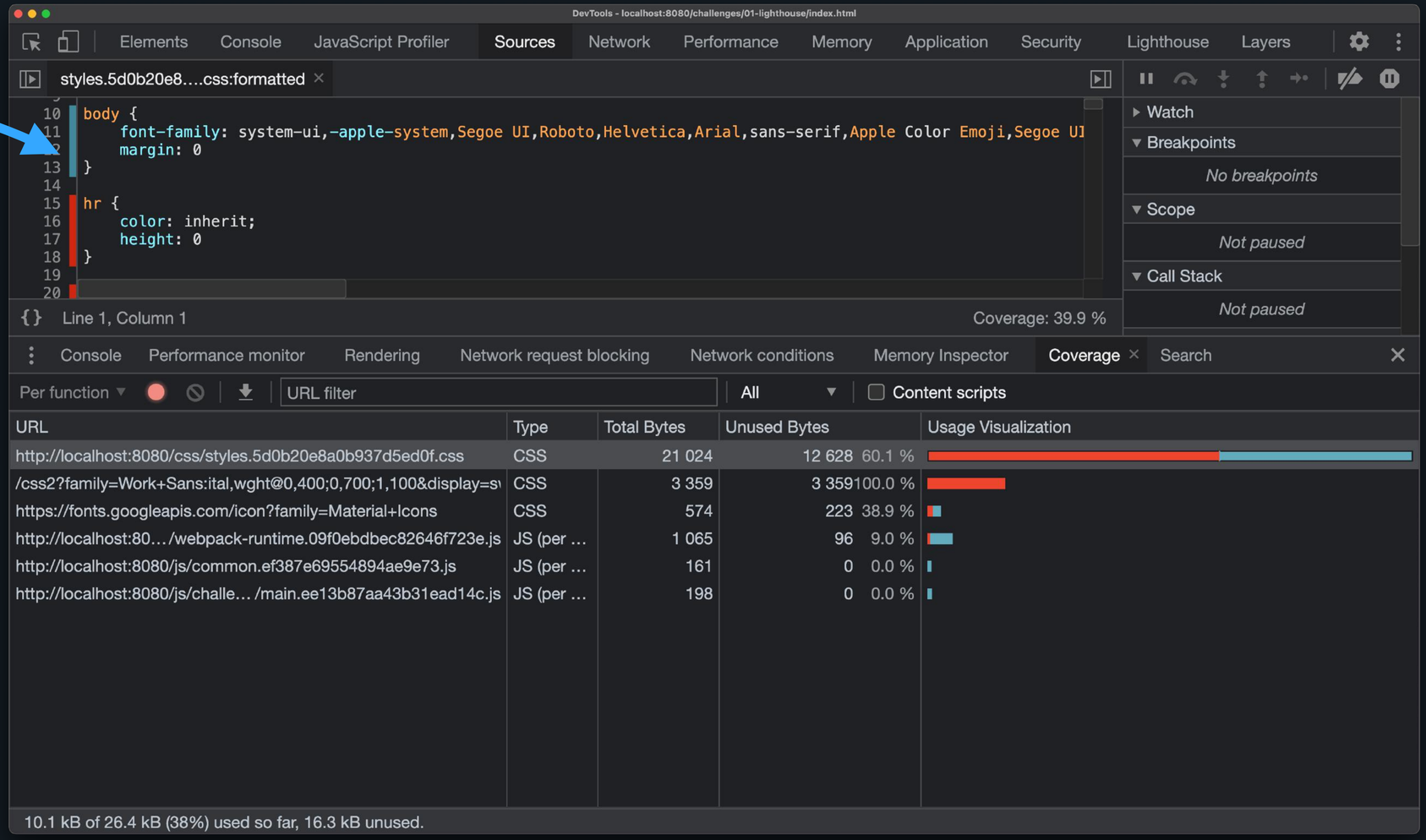
DevTools: Coverage

Unused code



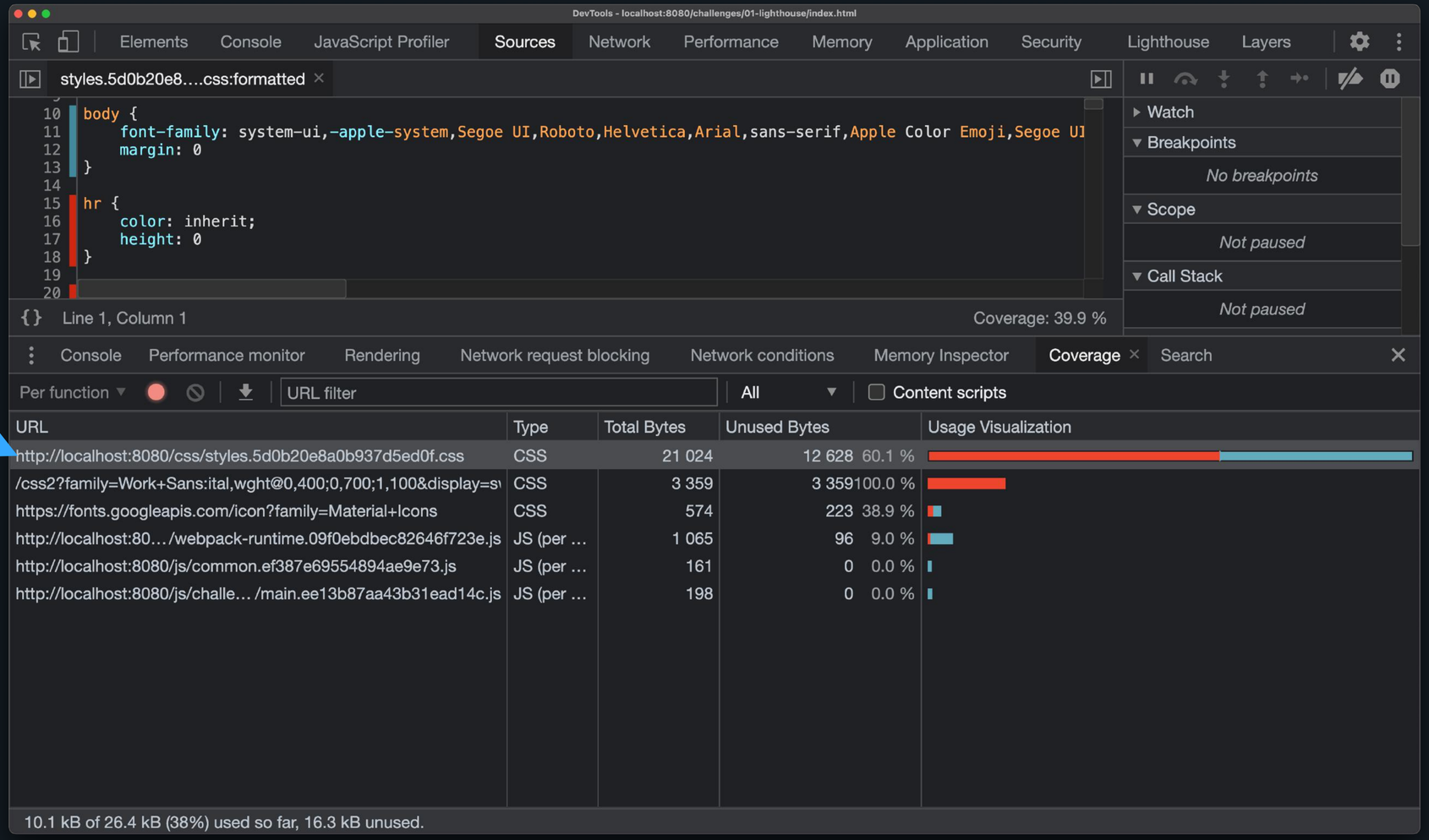
DevTools: Coverage

Used code



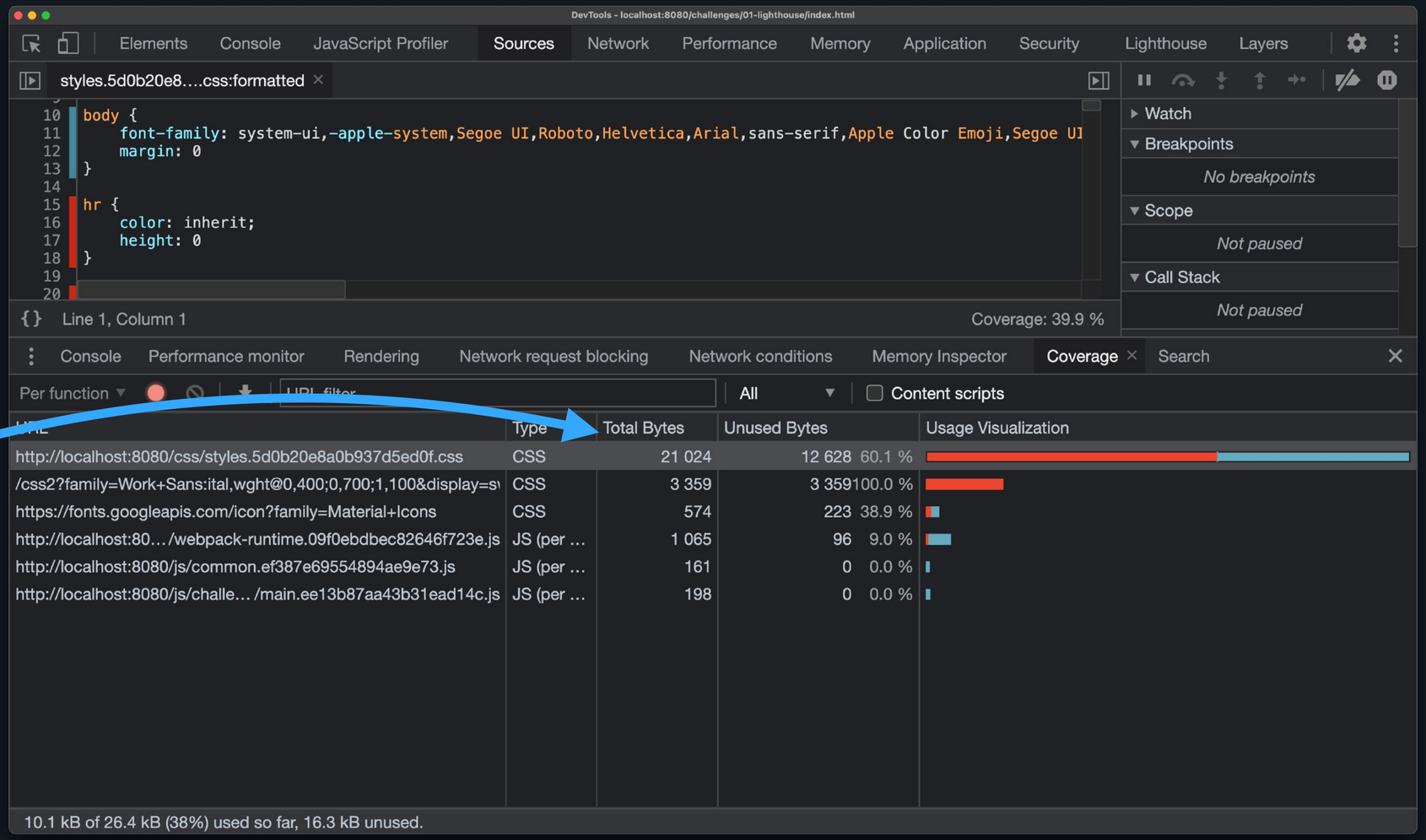
DevTools: Coverage

Resource URL



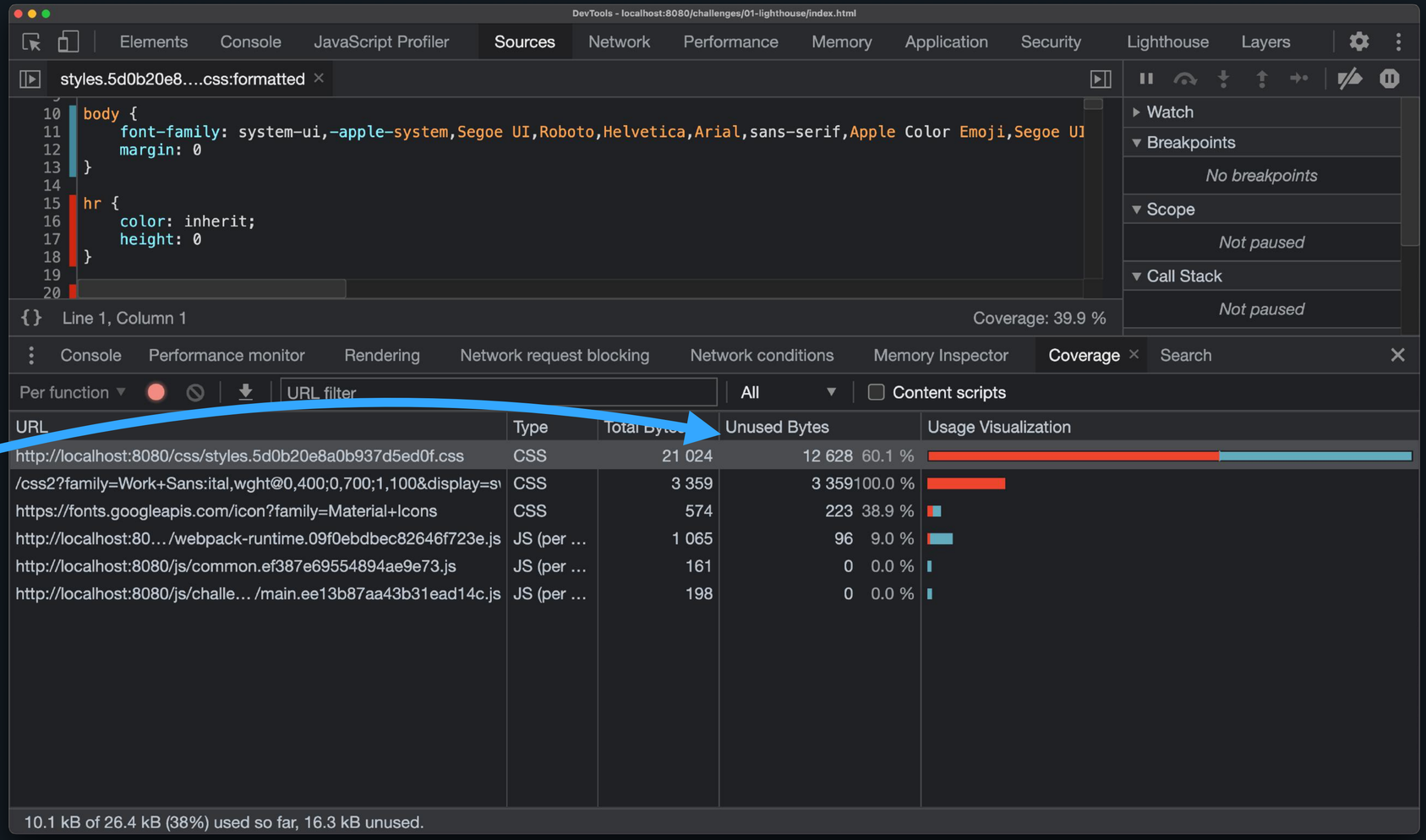
DevTools: Coverage

Resource size in bytes



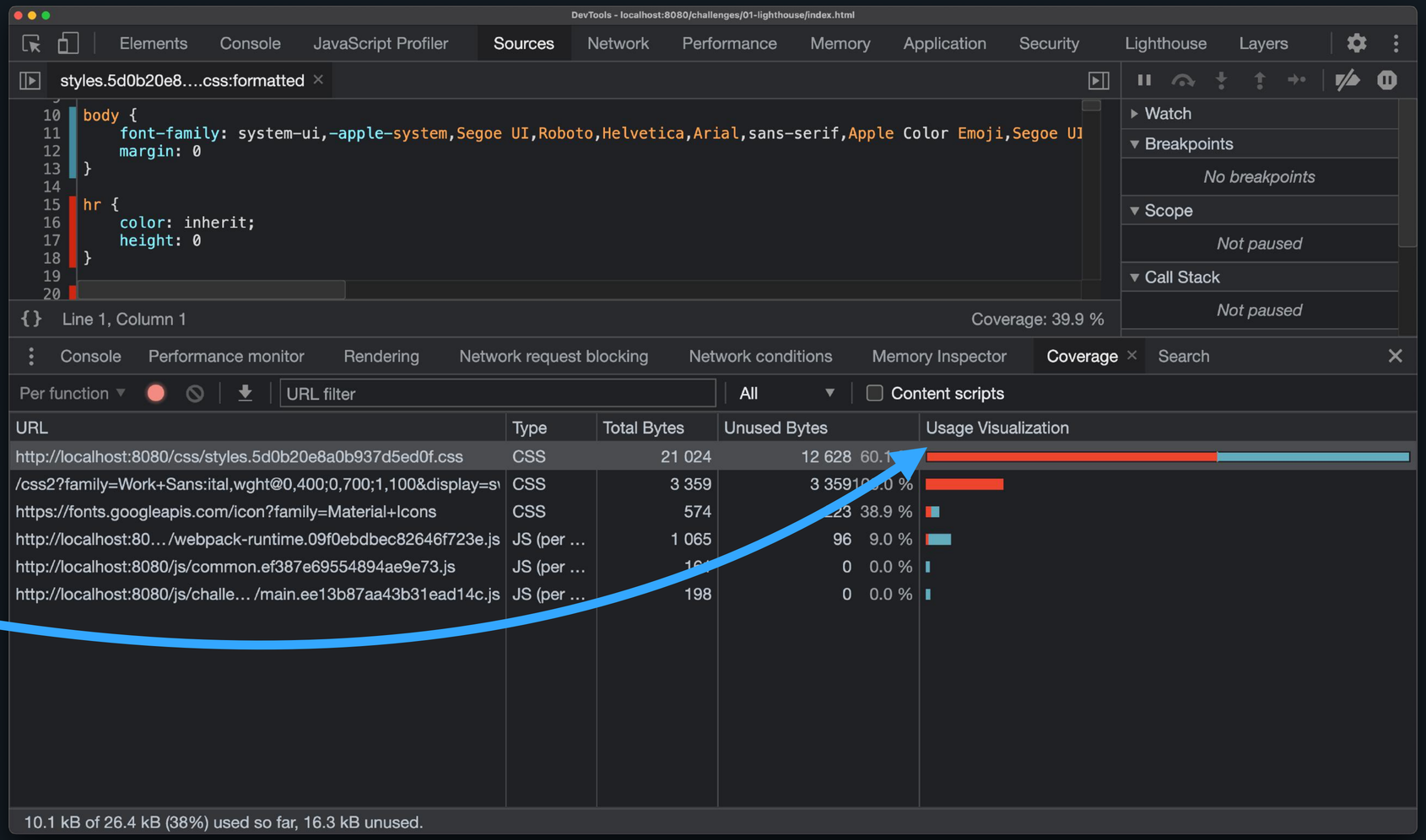
DevTools: Coverage

Unused bytes



DevTools: Coverage

Used and unused percentage graph



1. Open the **09 Coverage** challenge in Chrome.
2. Press **Esc** (or navigate to the ellipsis menu and choose Show console drawer)
3. In the DevTools console drawer, click on the **Coverage** tab.
4. Click the **Reload** button to start instrumenting coverage and reload the page.
5. Note the unused CSS and JS.

Resources

Resources

- https://web.dev
- https://developer.chrome.com/docs/devtools
- https://dev.chromium.org/developers/design-documents/gpu-accelerated-compositing-in-chrome
- https://developers.google.com/web/fundamentals/performance/rendering

Thank You!

Build blazing-fast apps with Chrome DevTools

@brian_love — @MikeRyanDev

Performance Audit

- Free 1/2-day Performance Audit
- brian@liveloveapp.com
- liveloveapp.com