

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

-  Code of Conduct
-  10 am to 4 pm MST
-  Two 20-minute breaks
-  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

100+ ms

focus is lost

1000+ ms



The RAIL Model



Response



Animation



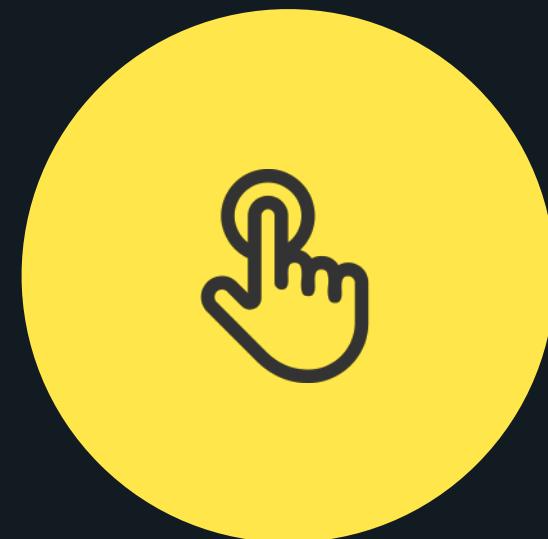
Idle



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?

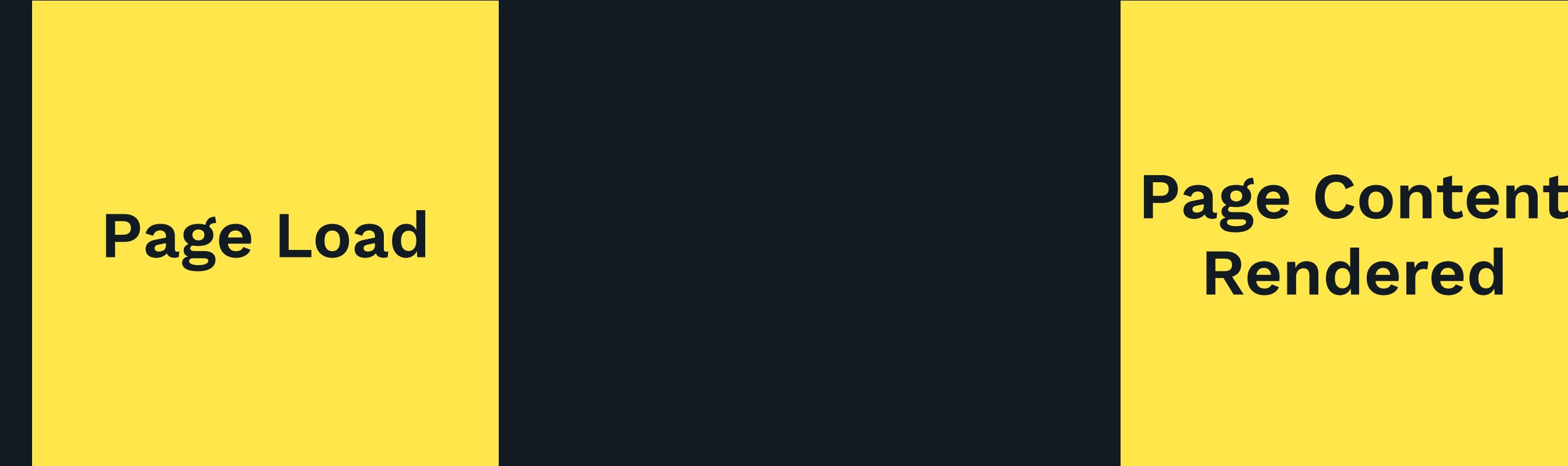
 Lab

 Field

Metric types

-  Perceived load time
-  Responsiveness
-  Runtime
-  Visual stability
-  Smoothness

First Contentful Paint (FCP)



Page Load

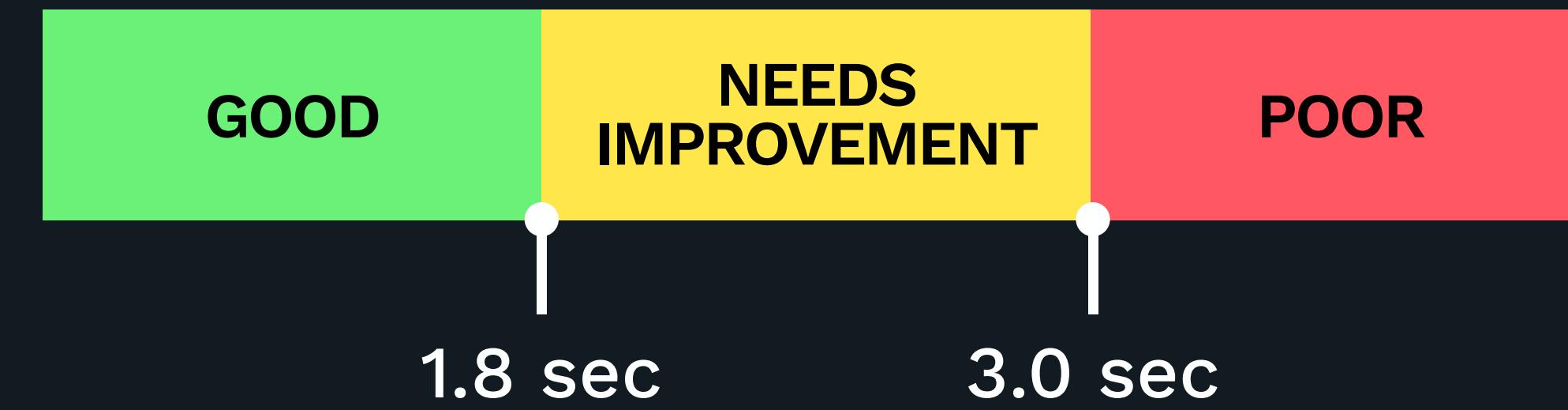
Page Content
Rendered

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

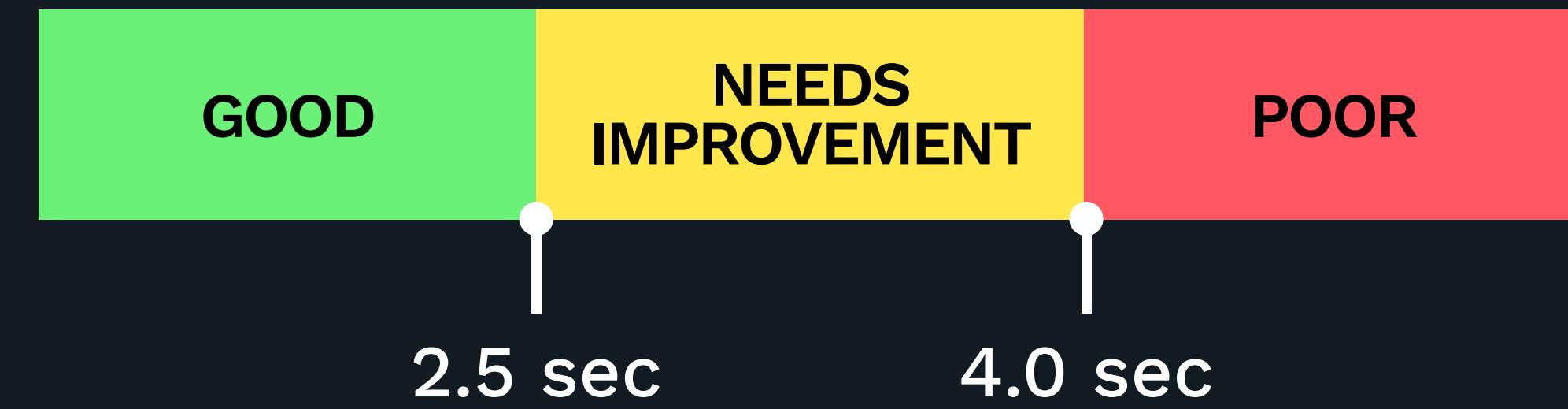
Largest text/
image block
rendered

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
Interaction**

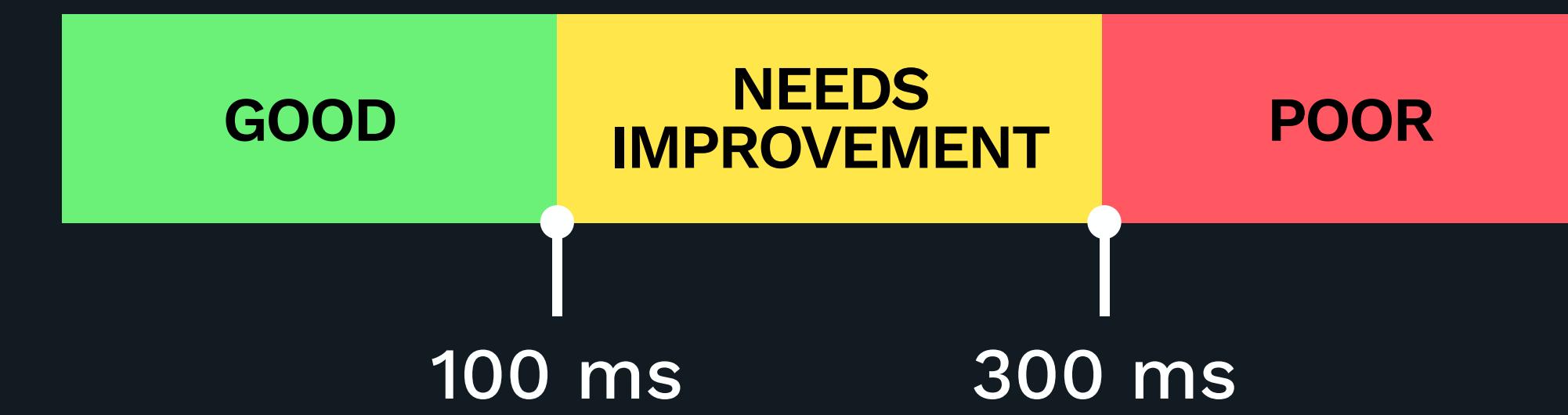
**Response to
interaction**

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)

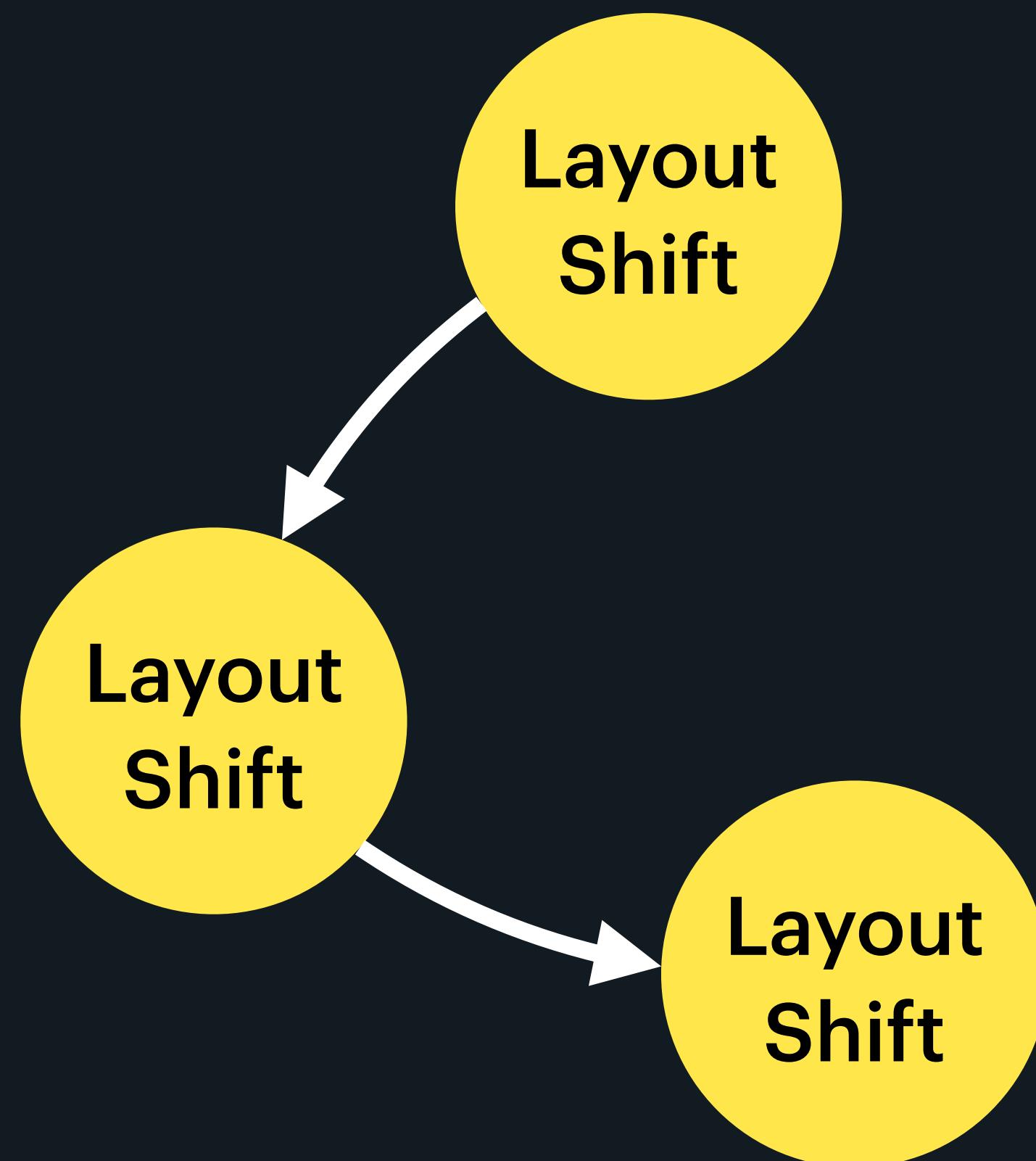
FCP

TTI

Total Blocking Time (TBT)

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

Cumulative Layout Shift (CLS)



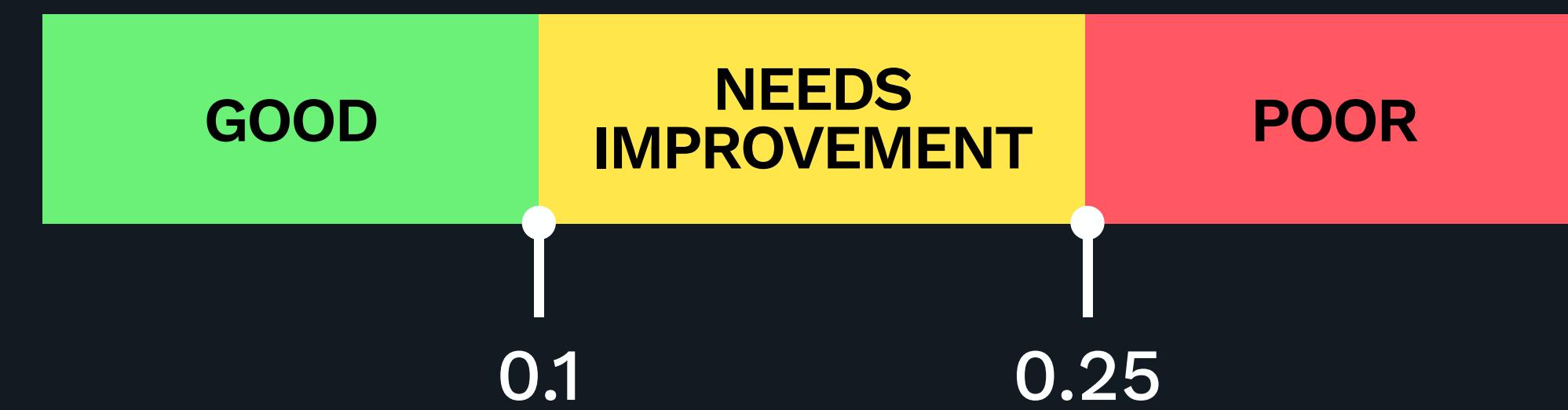
- Largest burst of layout shift scores for every unexpected layout shift that occurs during the entire lifespan of a page.
- A layout shift is only bad if the user isn't expecting it.

Cumulative Layout Shift (CLS)

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

CLS

Cumulative Layout Shift



Core Web Vitals

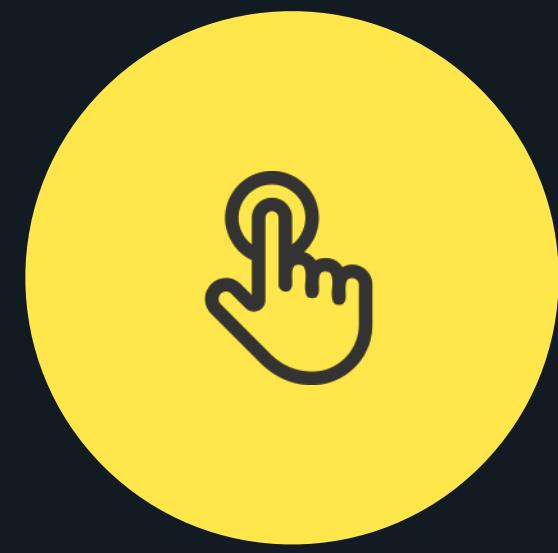
**Field metrics that measure important
aspects of real-world user experience**

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

-  Establish a baseline
-  Add budgets to build process
-  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)

Build Process Tools

Performance Budget



Webpack Performance



Lighthouse CI



bundlesize

What if I exceed the budget?

— Developer

Exceeding Budget

Performance Budget

-  Optimize existing features or assets
-  Remove existing features or assets
-  Don't add a new feature or asset

Exceeding Budget

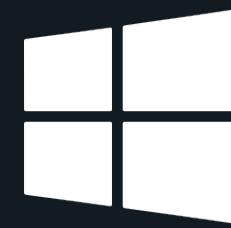
Performance Budget

-  Code-splitting
-  Service worker caching
-  HTTP/2 Push

Chrome DevTools

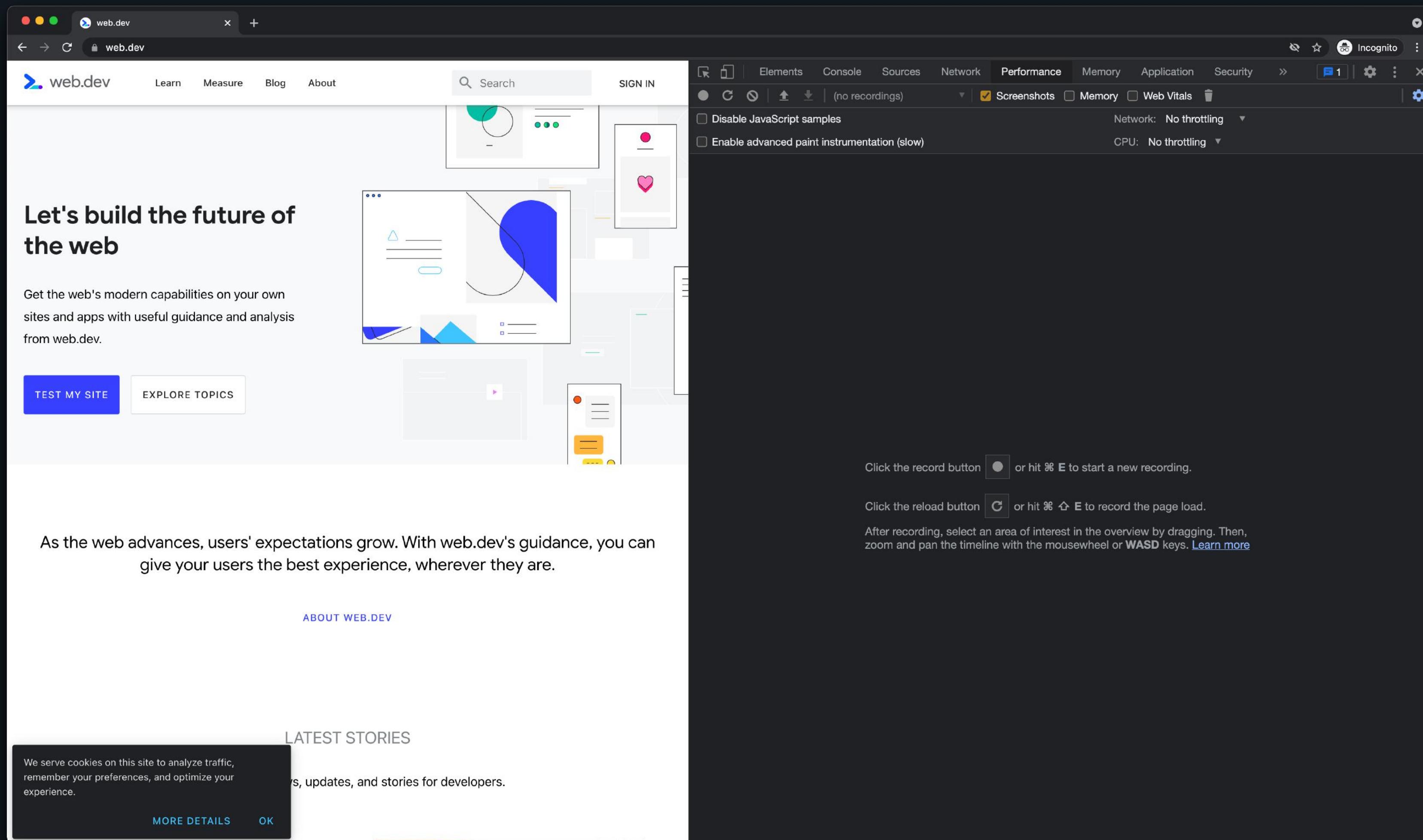
macOS

Cmd + Option + I



Control + Shift + I

DevTools



DevTools Fun Facts

-  Built with TypeScript
-  150,000 lines of code
-  Over ten years old
-  Console loggers welcome 😊

Panels we are NOT going to use today

-  Device mode
-  Element panel
-  Console panel
-  And more

Panels focused on performance

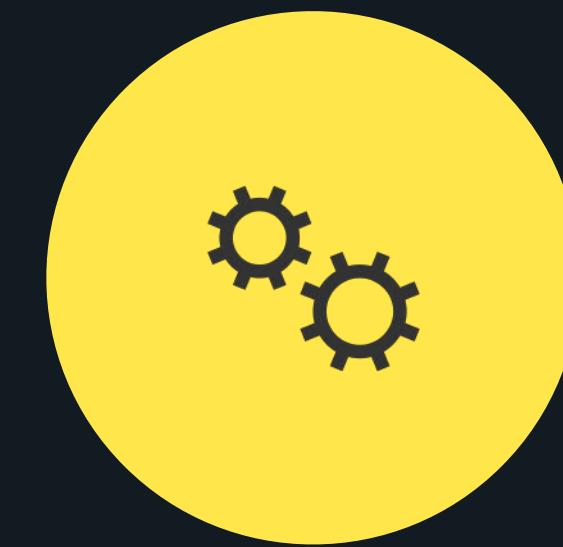
-  Lighthouse panel
-  Rendering panel
-  Performance Monitor panel
-  Performance panel
-  Memory panel
-  Coverage panel
-  Layers panel

DevTools: Lighthouse

Lighthouse



Open-source



Automated
tool



Improve
quality

Lighthouse

- Chrome DevTools
- Node
- Command line
- PageSpeed Insights

Lighthouse

The image shows a web browser window with two main panes. The left pane displays the web.dev homepage, which features a dark background with white and light gray text. It includes sections for "Let's build the future of the web", "TEST MY SITE", "EXPLORE TOPICS", and "LATEST STORIES". A cookie consent banner at the bottom states: "We serve cookies on this site to analyze traffic, remember your preferences, and optimize your experience. This helps us provide you with updates, and stories for developers." with "MORE DETAILS" and "OK" buttons. The right pane shows the Lighthouse audit interface in the Chrome DevTools. The "Lighthouse" tab is selected, showing a report with a lighthouse icon and a "Generate report" button. The report area contains text about identifying and fixing common problems related to performance, accessibility, and user experience. A warning message points out potential issues with IndexedDB. On the right, there are sections for "Categories" (Performance, Progressive Web App, Best practices, Accessibility, SEO), "Device" (Mobile, Desktop), and "Community Plugins (beta)".

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

We serve cookies on this site to analyze traffic, remember your preferences, and optimize your experience. This helps us provide you with updates, and stories for developers.

MORE DETAILS OK

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

Device

- Mobile
- Desktop

Community Plugins (beta)

- Publisher Ads

Lighthouse

The screenshot displays the Lighthouse performance audit results for the web.dev homepage. The overall score is 94. The audit results are categorized into four main sections: Performance, Best Practices, SEO, and Progressive Web App.

Performance Metrics:

| Metric | Value |
|--------------------------|-------|
| First Contentful Paint | 1.3 s |
| Speed Index | 2.0 s |
| Largest Contentful Paint | 2.9 s |
| Time to Interactive | 3.6 s |
| Total Blocking Time | 30 ms |
| Cumulative Layout Shift | 0 |

Opportunities:

- Reduce unused JavaScript (Estimated Savings: 0.9 s)
- Properly size images (Estimated Savings: 0.15 s)

Best Practices Score: 94

SEO Score: 99

PWA Score: 94

Issues: 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.

Metrics:

- TEST MY SITE
- EXPLORE TOPICS

ABOUT WEB.DEV

LATEST STORIES

We serve cookies on this site to analyze traffic, remember your preferences, and optimize your experience.

MORE DETAILS OK

Lighthouse

-  Lab measurements
-  Metrics
-  Opportunities
-  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

All checks have passed
3 successful checks

| | | | |
|--|--|--|-------------------------|
| | | continuous-integration/travis-ci/push — The Trav... | Details |
| | | lhci/url/404.html — Performance: 97, Accessibility:... | Details |
| | | lhci/url/index.html — Performance: 96, Accessibili... | Details |

Lighthouse CI

```
→ npm i @lhci/cli
```

Lighthouse CI

→ **lhci autorun**

Lighthouse CI

→ touch `lighthousec.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

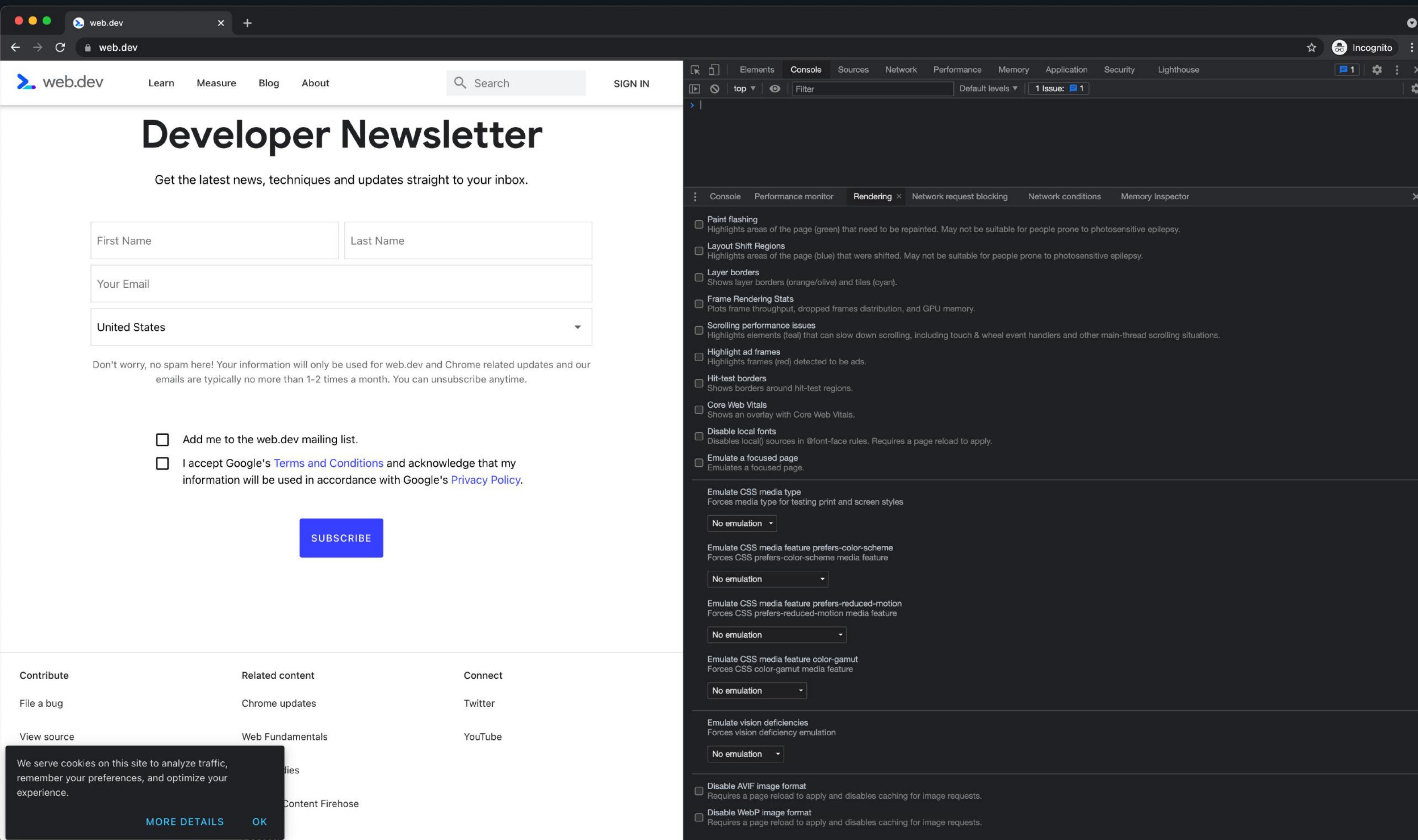


Frame
Rendering
Stats



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

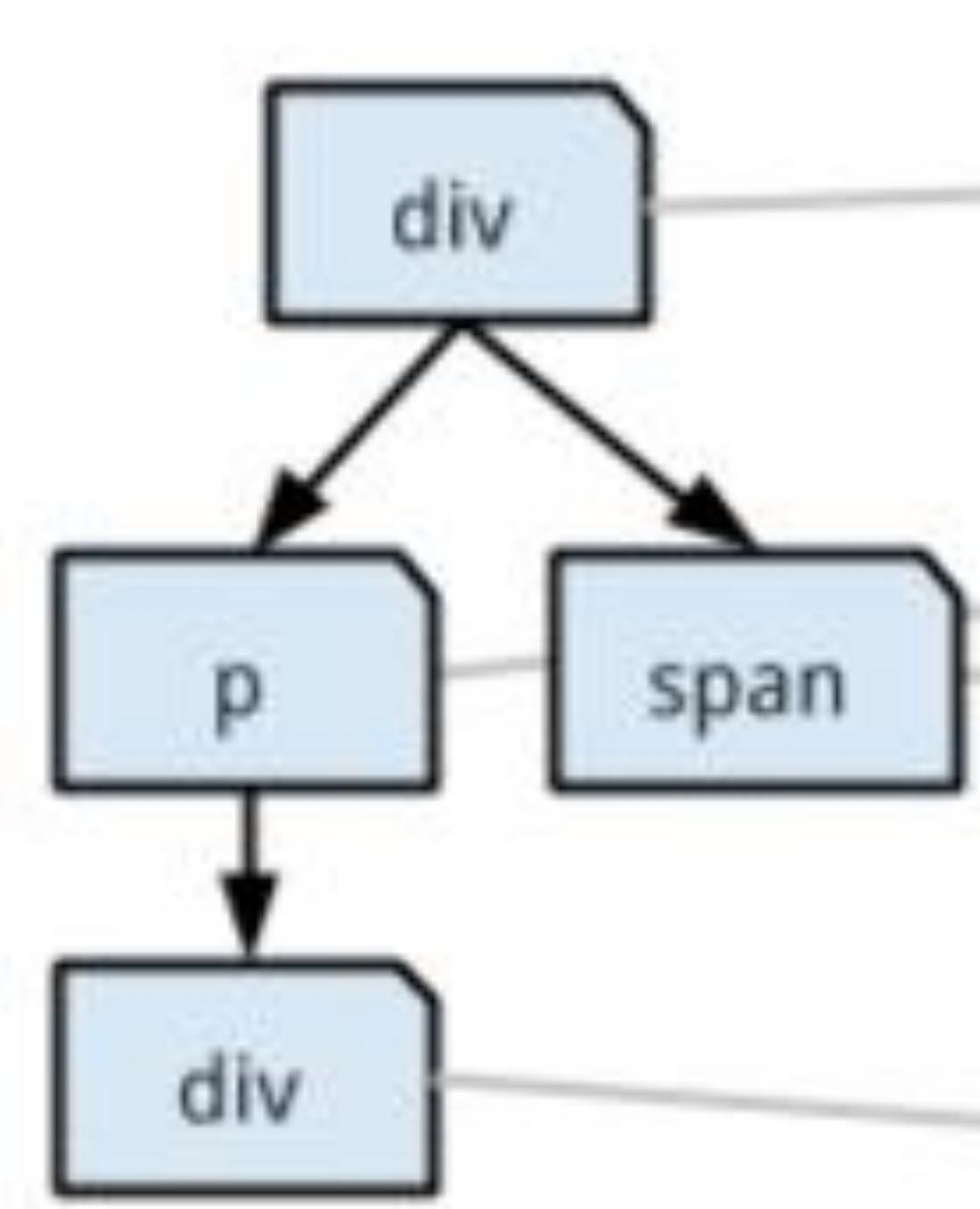
-  Each **Node** has a corresponding **RenderObject**
-  Composed into the render tree
-  Contains paint instructions

RenderLayer and GraphicsLayer

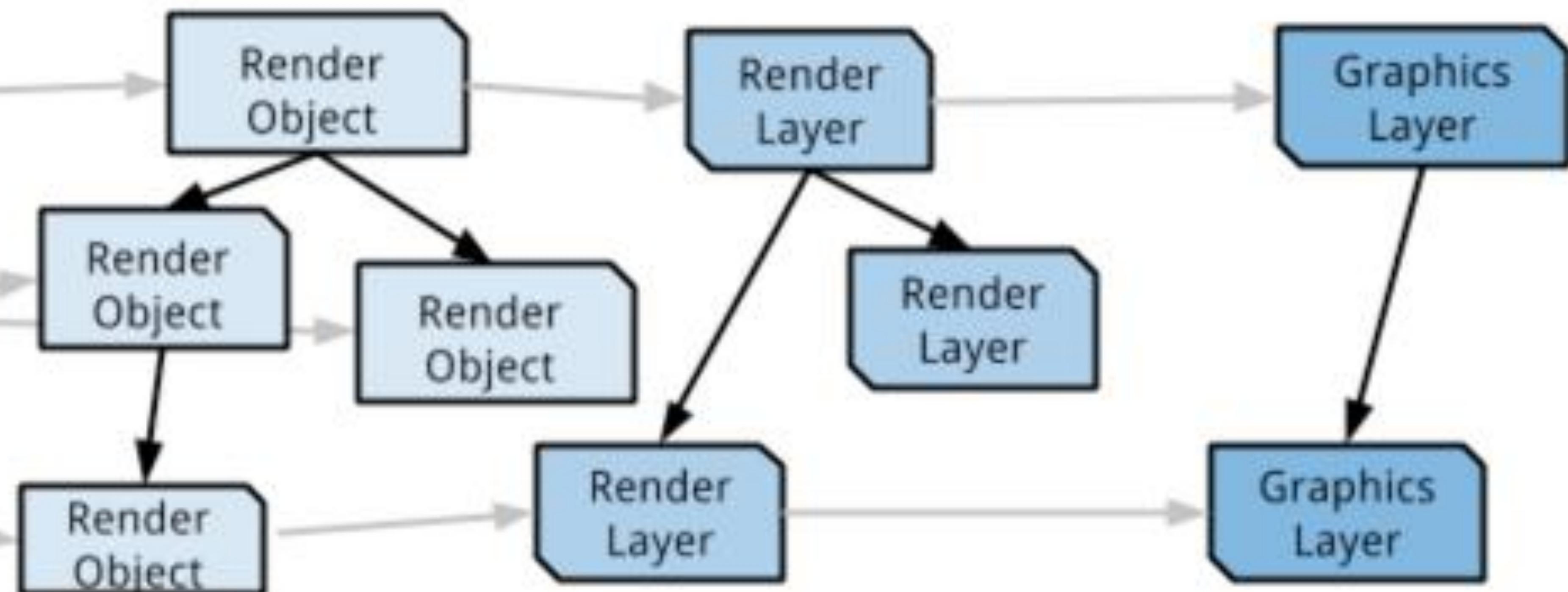
- ✓ Each **RenderGroup** is associated with a **RenderLayer**
- ✓ **RenderLayer** renders subtrees of the DOM
- ✓ **GraphicsLayer** compositing layers that leverage the GPU for parallelism

Compositing of Layers

DOM tree



Layer trees



When is a new GraphicsLayer created?

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

When is a new GraphicsLayer created?

-  Layer uses accelerated CSS filters
-  Layer has a descendant that is a compositing layer
-  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)

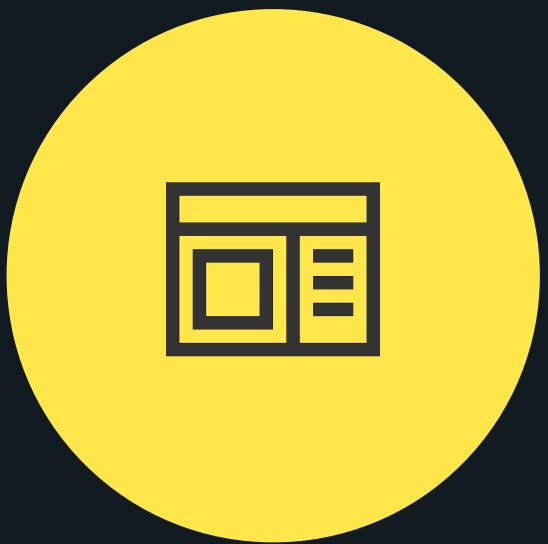
Layout Shift

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 **** 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

-  Main thread is blocked in case preventDefault() is invoked
-  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

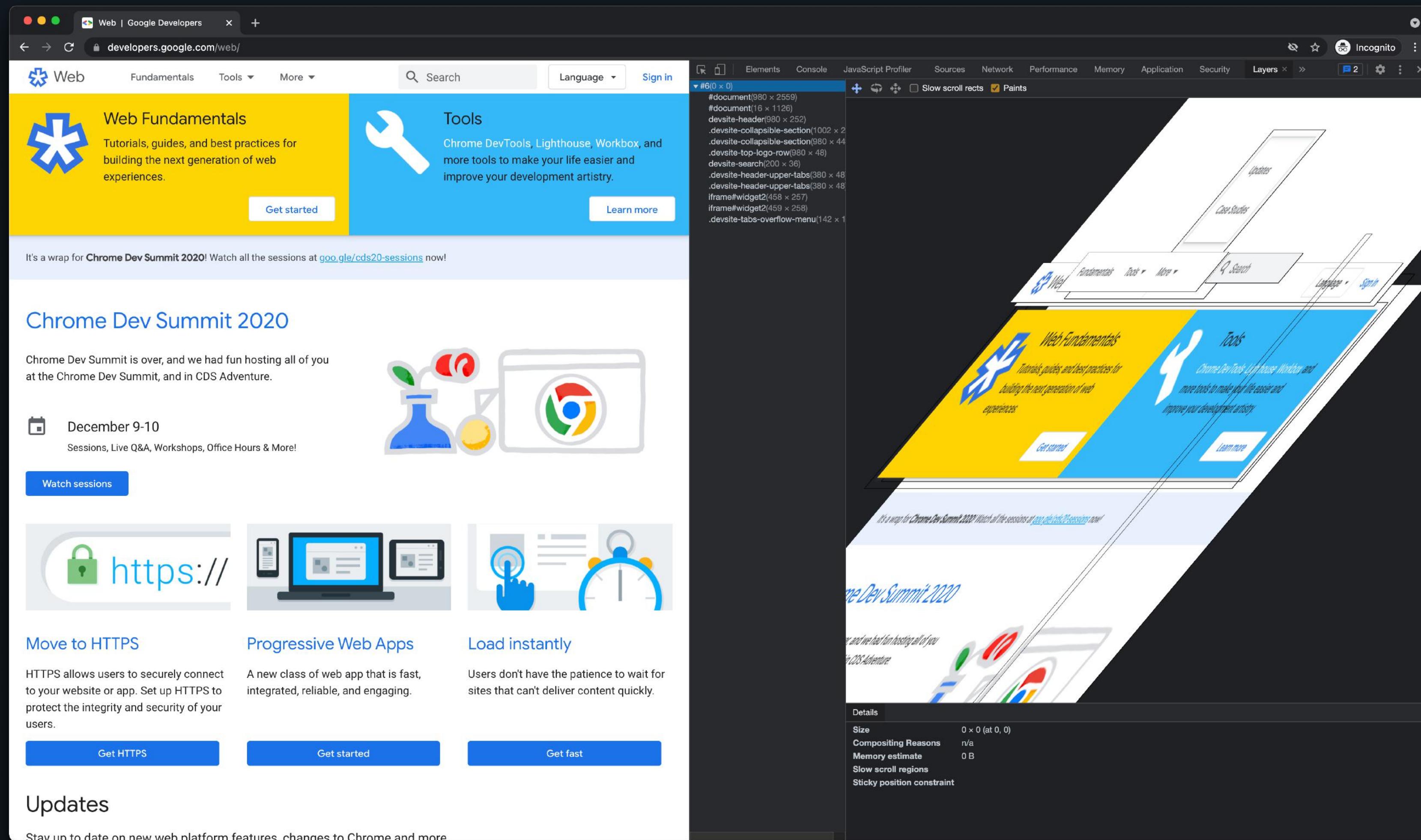
1. Open the **02 Rendering** challenge in Chrome.
2. Press **Esc** (or navigate to the ellipsis menu and choose Show console drawer).
3. In the DevTools console drawer, click on the **Rendering** tab, and check the **Scrolling performance issues** checkbox.
4. Add **touchmove** event handlers to all 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

-  [GPU Accelerating Compositing in Chrome](#)
-  [Accelerated Rendering in Chrome](#)
-  [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

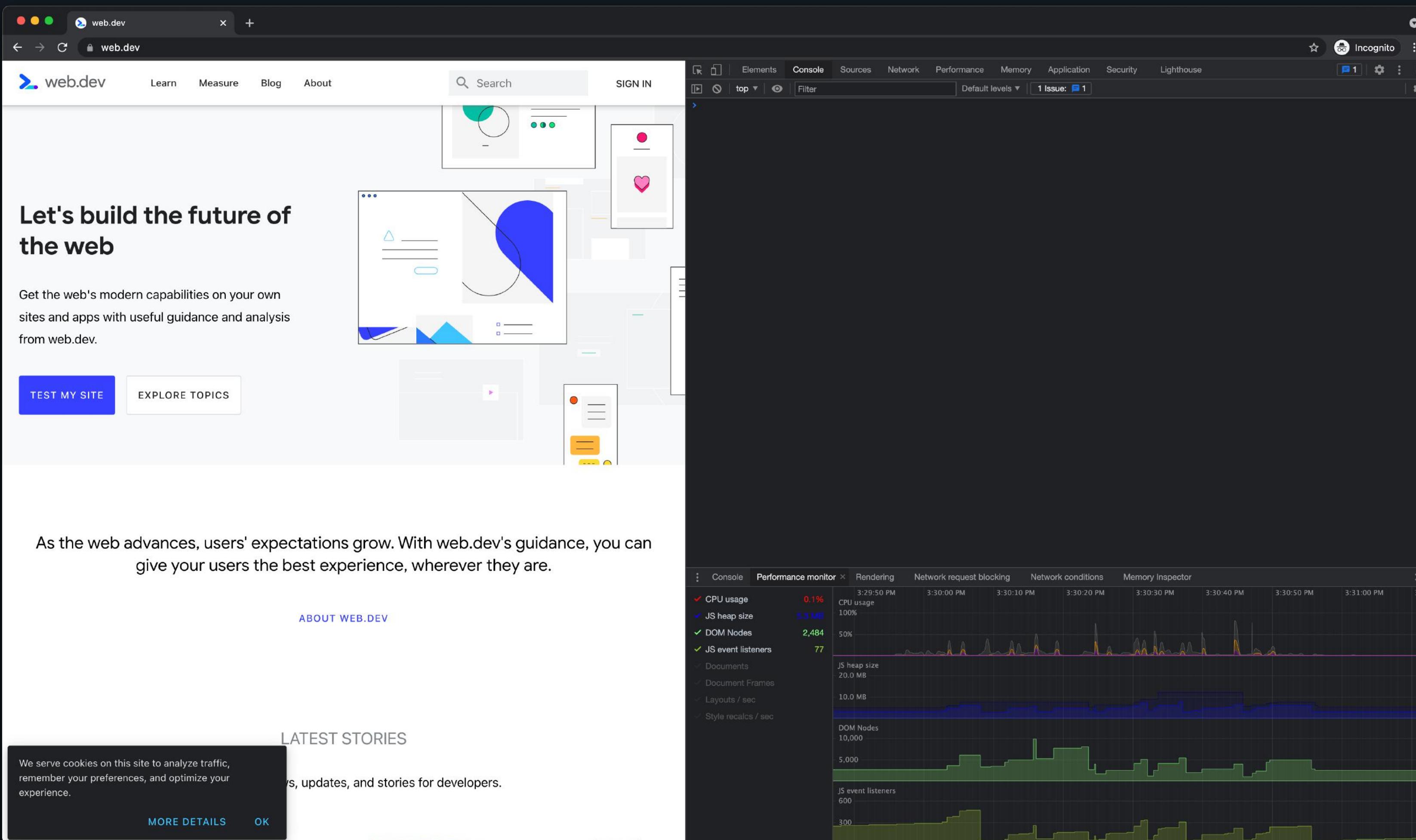
```
interface LayoutShift extends PerformanceEntry {  
  hadRecentInput: boolean;  
  value: number;  
}
```

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

-  CPU usage
-  JS heap size
-  DOM Nodes
-  JS Event Listeners

Performance Monitor

- Documents
- Document Frames (iframe)
- Layouts per second
- 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

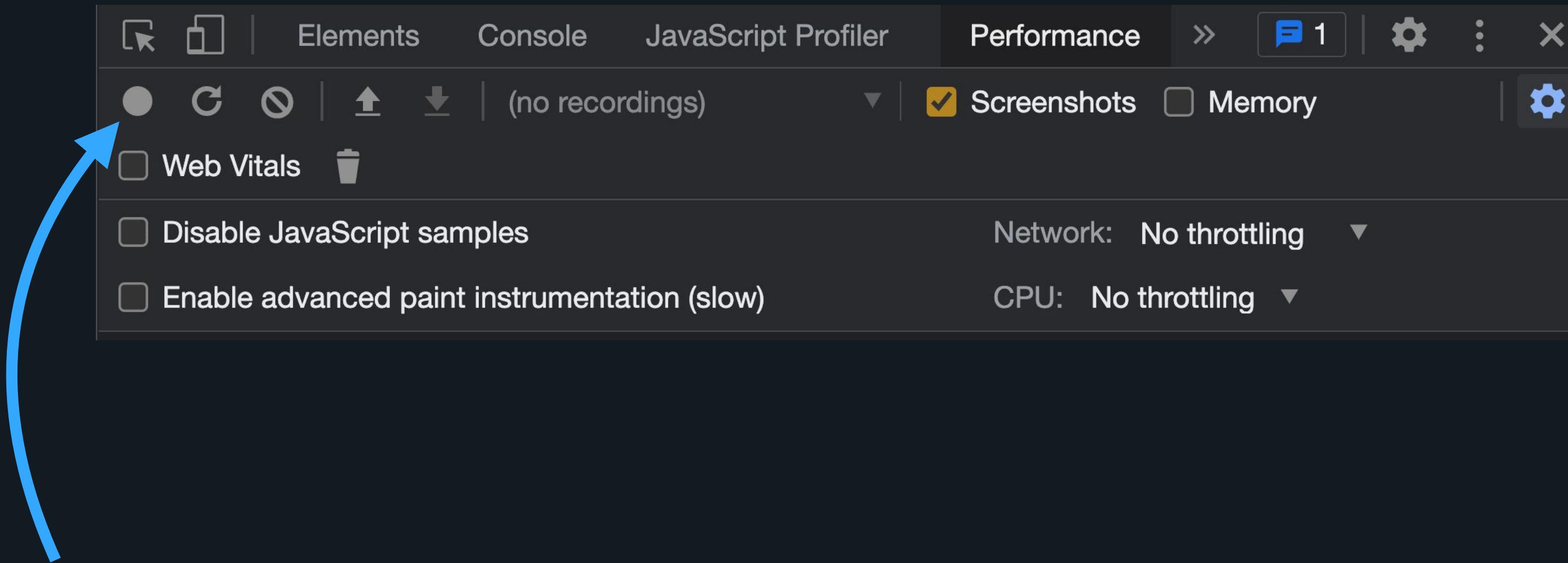
DevTools: Performance

DevTools: Performance

The image shows a split-screen view of a web browser. On the left, the main content area displays the web.dev website, specifically its "Developer Newsletter" section. This page features a form for signing up to receive news, including fields for First Name, Last Name, Your Email, and a dropdown for United States. It also includes a note about spam prevention and two checkboxes for mailing list and terms/conditions acceptance. A prominent blue "SUBSCRIBE" button is centered below the form. At the bottom, there are links for Contribute, Related content, and Connect, along with a cookie consent message and a "MORE DETAILS" link.

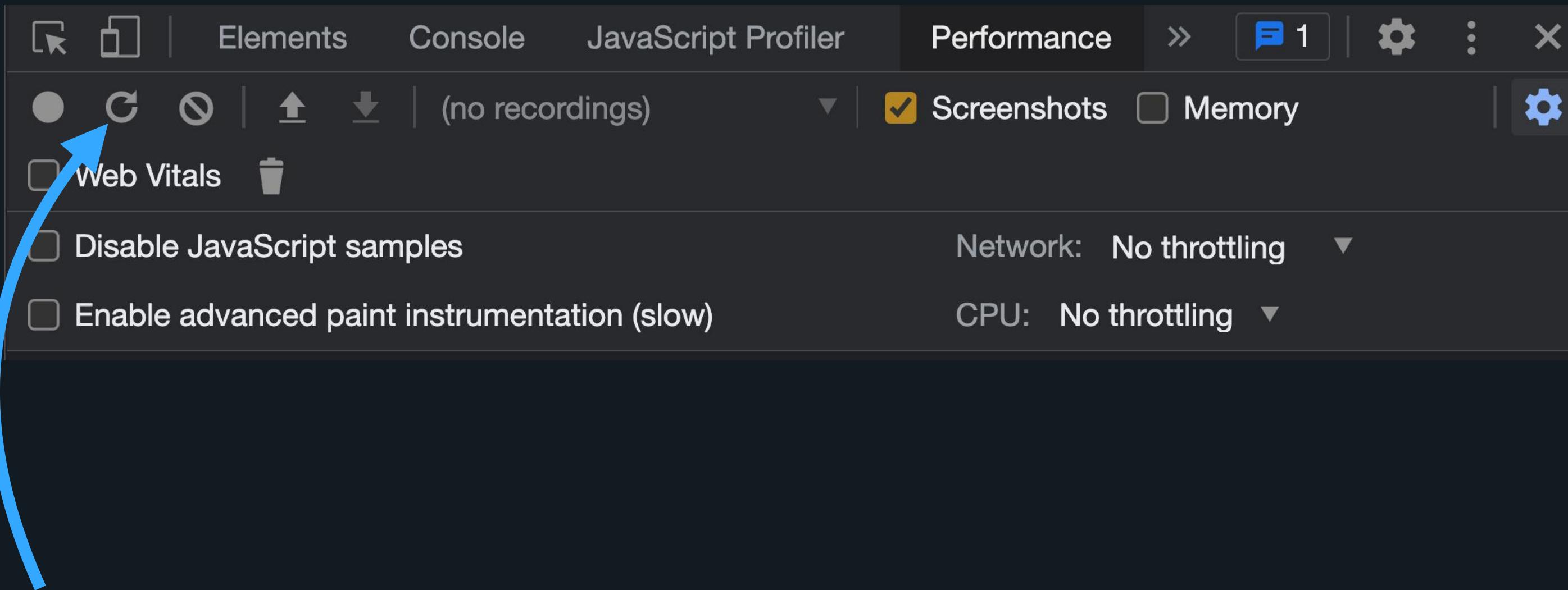
On the right, the developer tools' Performance panel is open, showing a timeline of network requests and CPU activity for the [Main](https://web.dev) thread. The timeline is color-coded by task type: Loading (blue), Scripting (yellow), Rendering (purple), Painting (green), System (grey), and Idle (white). A summary at the bottom indicates a total duration of 2916 ms, with the vast majority being Idle time. The CPU section shows various threads like CPU, Chrome_ChildIOThread, and Compositor.

DevTools: Performance



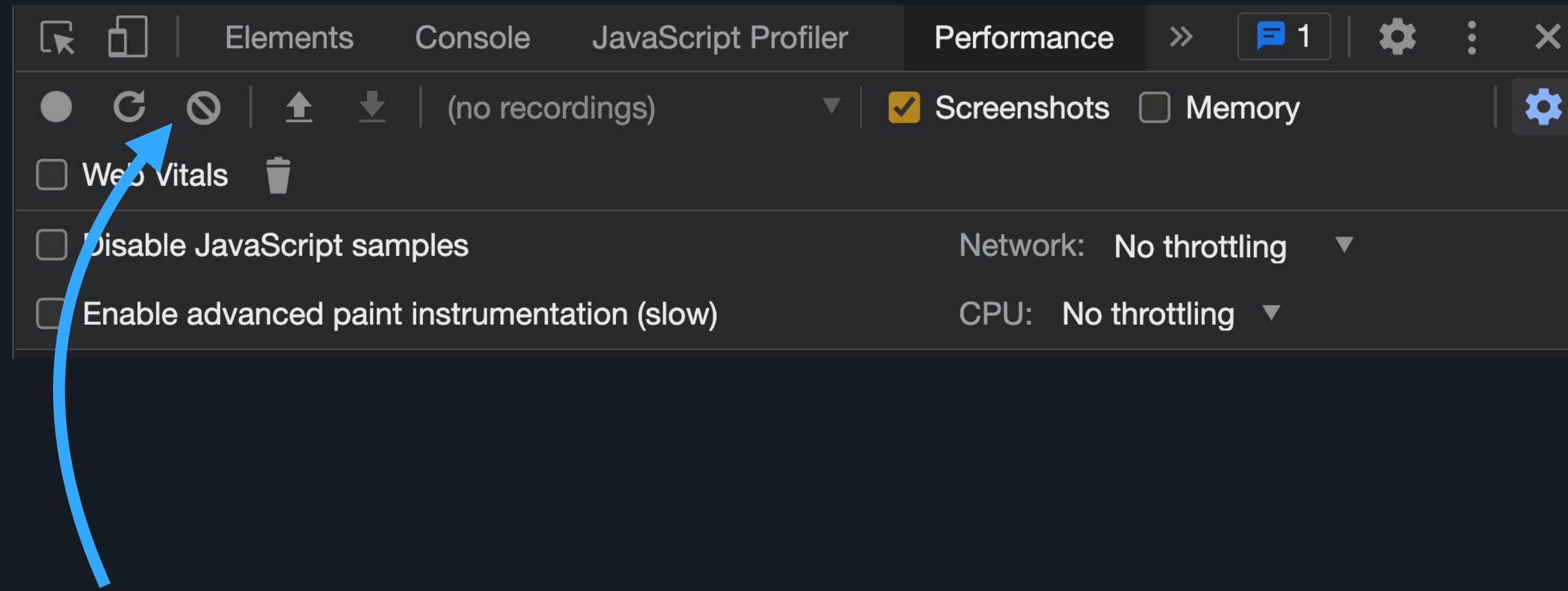
Start a new recording

DevTools: Performance



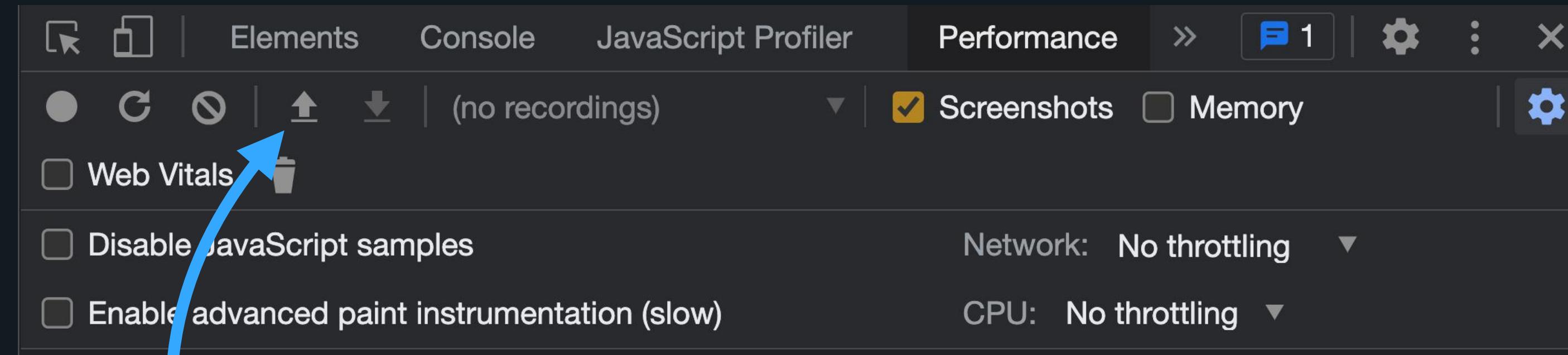
Reload page and start recording

DevTools: Performance



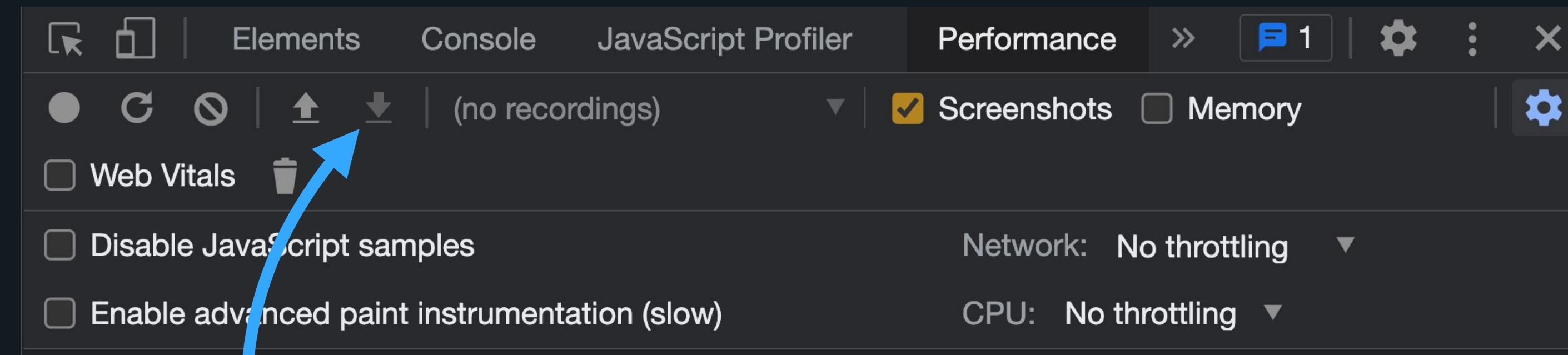
Clear (delete) the current recording

DevTools: Performance



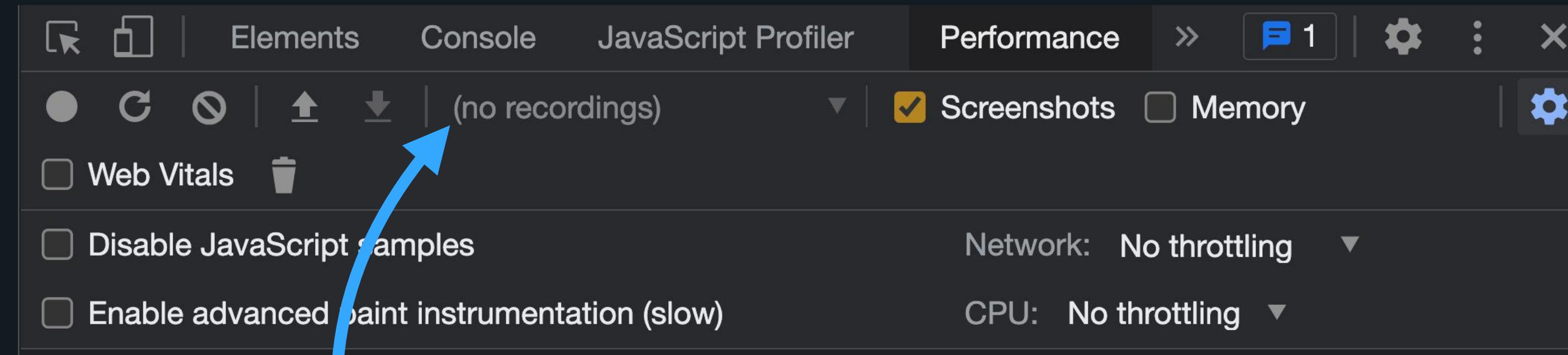
Upload a previous recording

DevTools: Performance



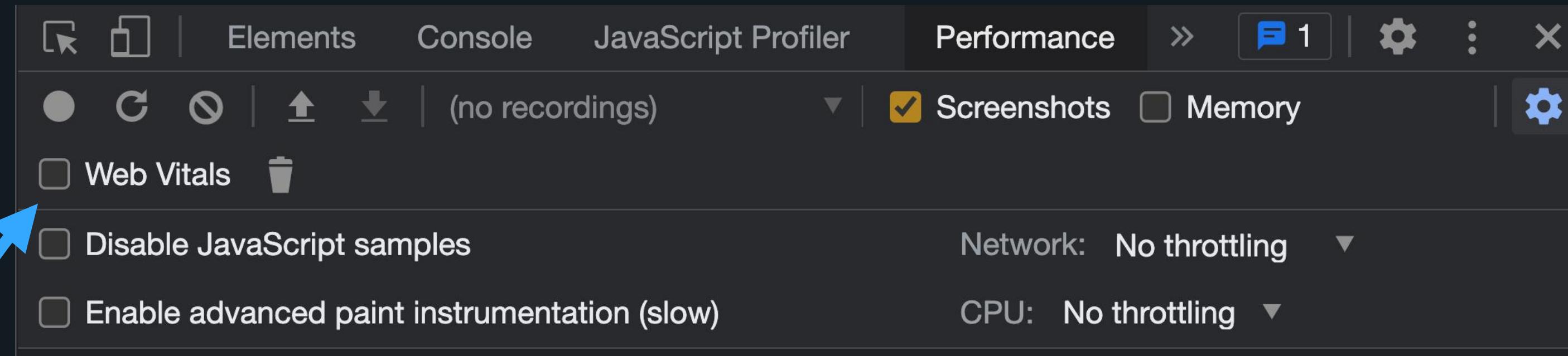
Download the current recording

DevTools: Performance



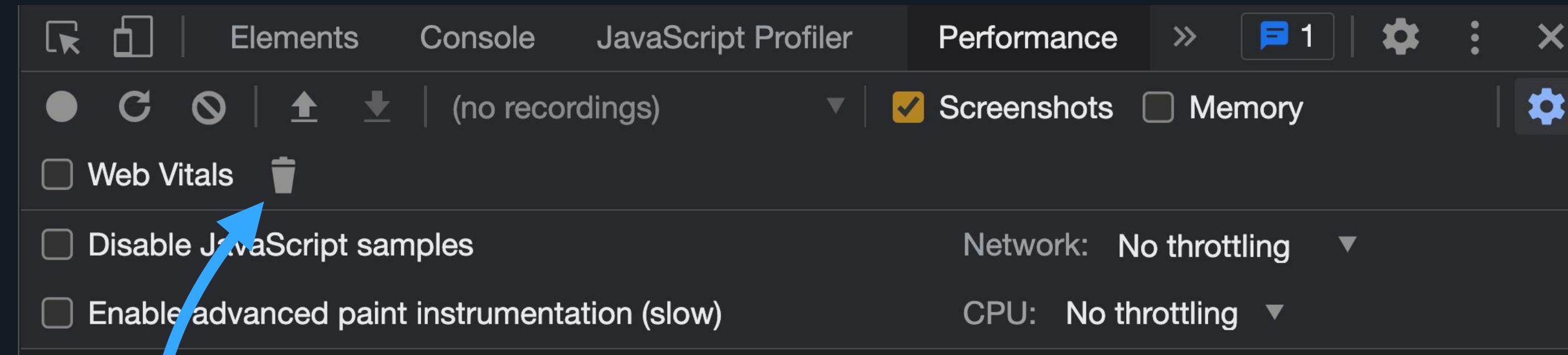
Select a previous recording

DevTools: Performance



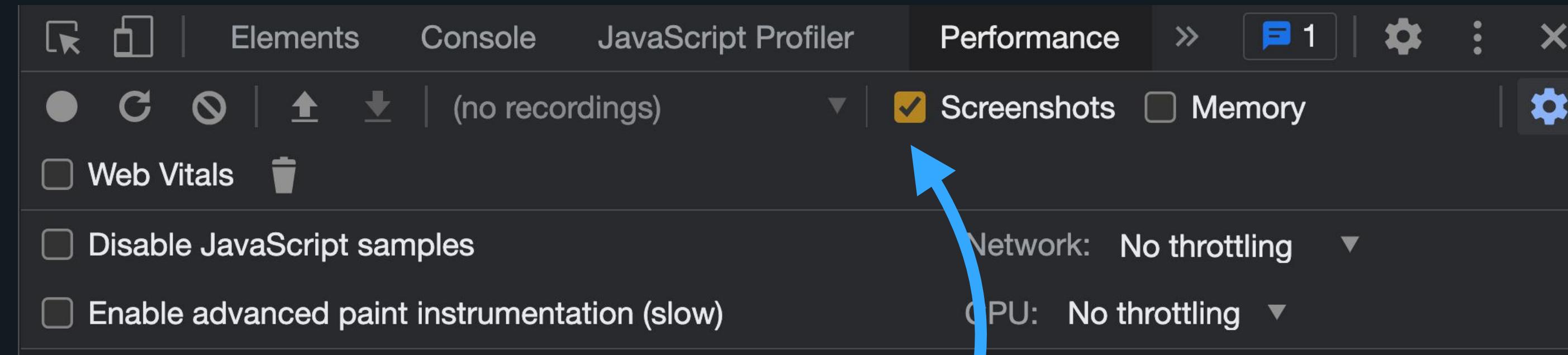
Display Web Vitals metrics

DevTools: Performance



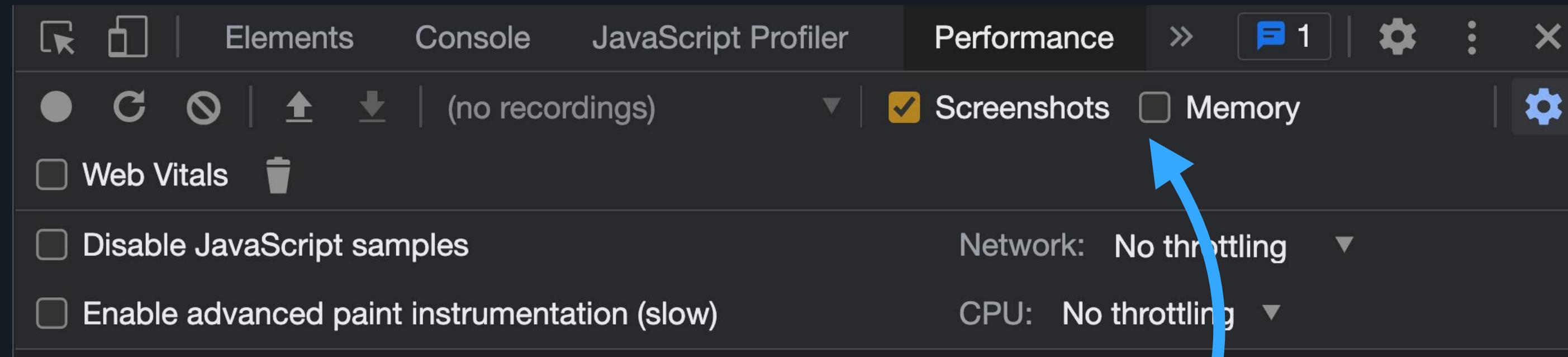
Force garbage collection

DevTools: Performance



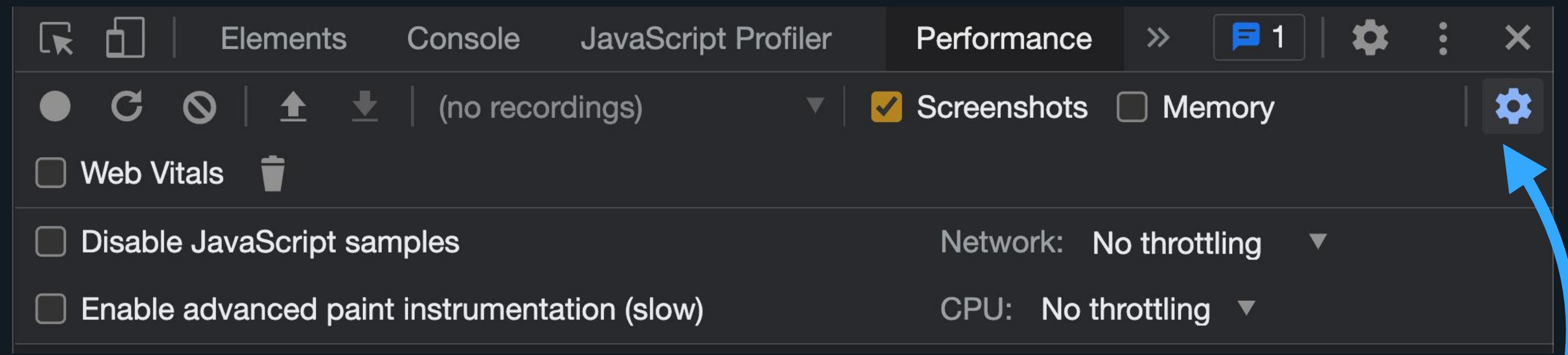
Take screenshots during the recording

DevTools: Performance



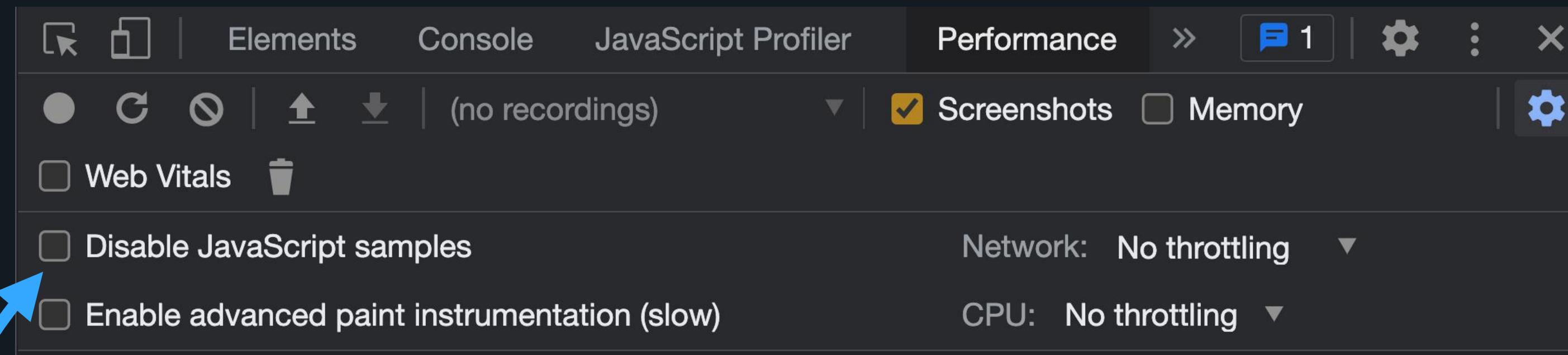
Record a timeline of the JS heap size

DevTools: Performance



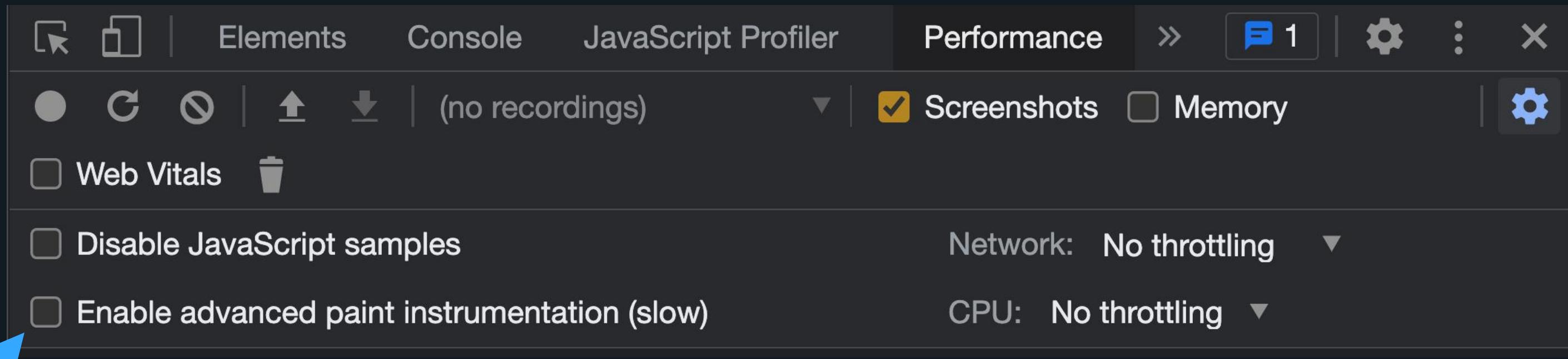
Show more configuration options

DevTools: Performance



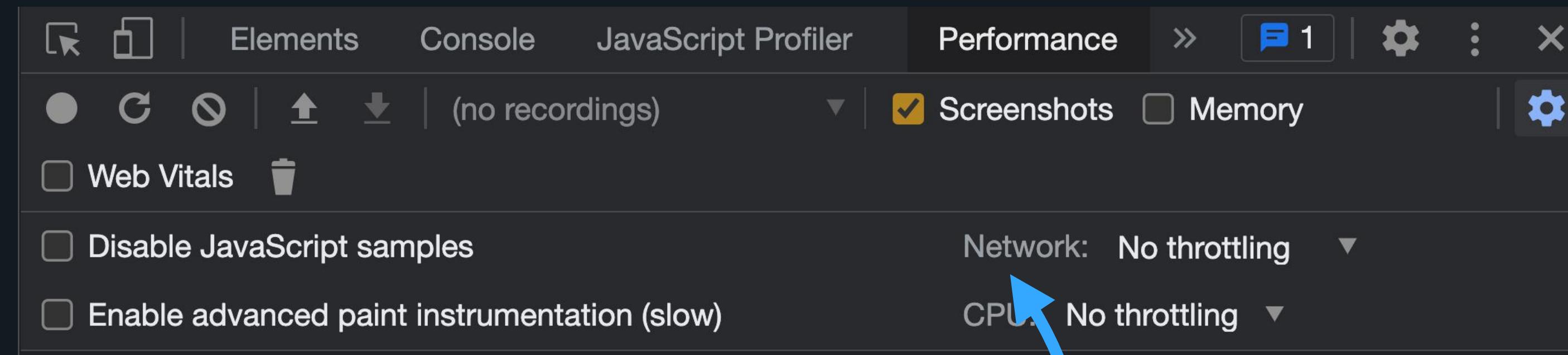
Do not include JS code in
the flame graphs of the main thread

DevTools: Performance



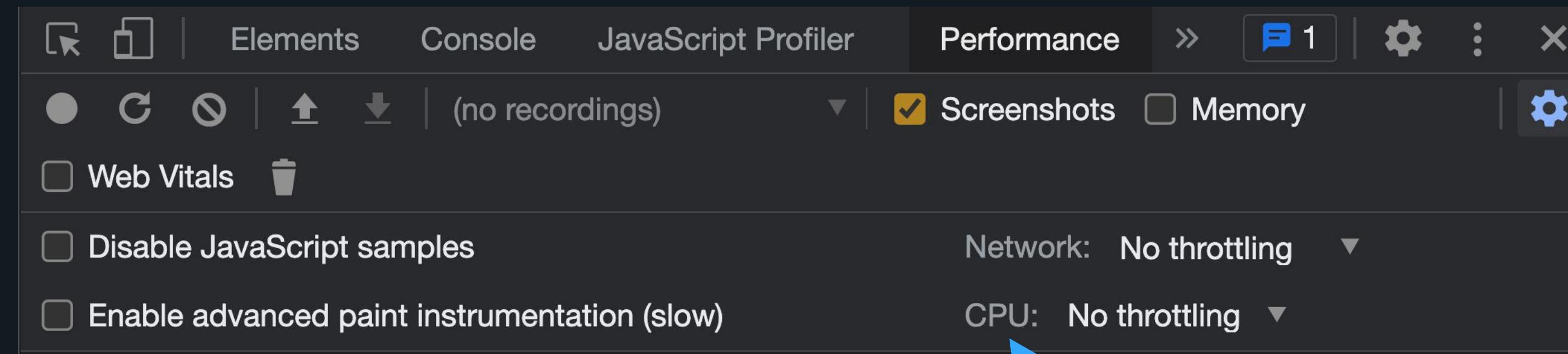
Includes layer information (similar to Layers panel) for frames, and paint profiling information during paints.

DevTools: Performance



Network throttling

DevTools: Performance



CPU throttling

Performance

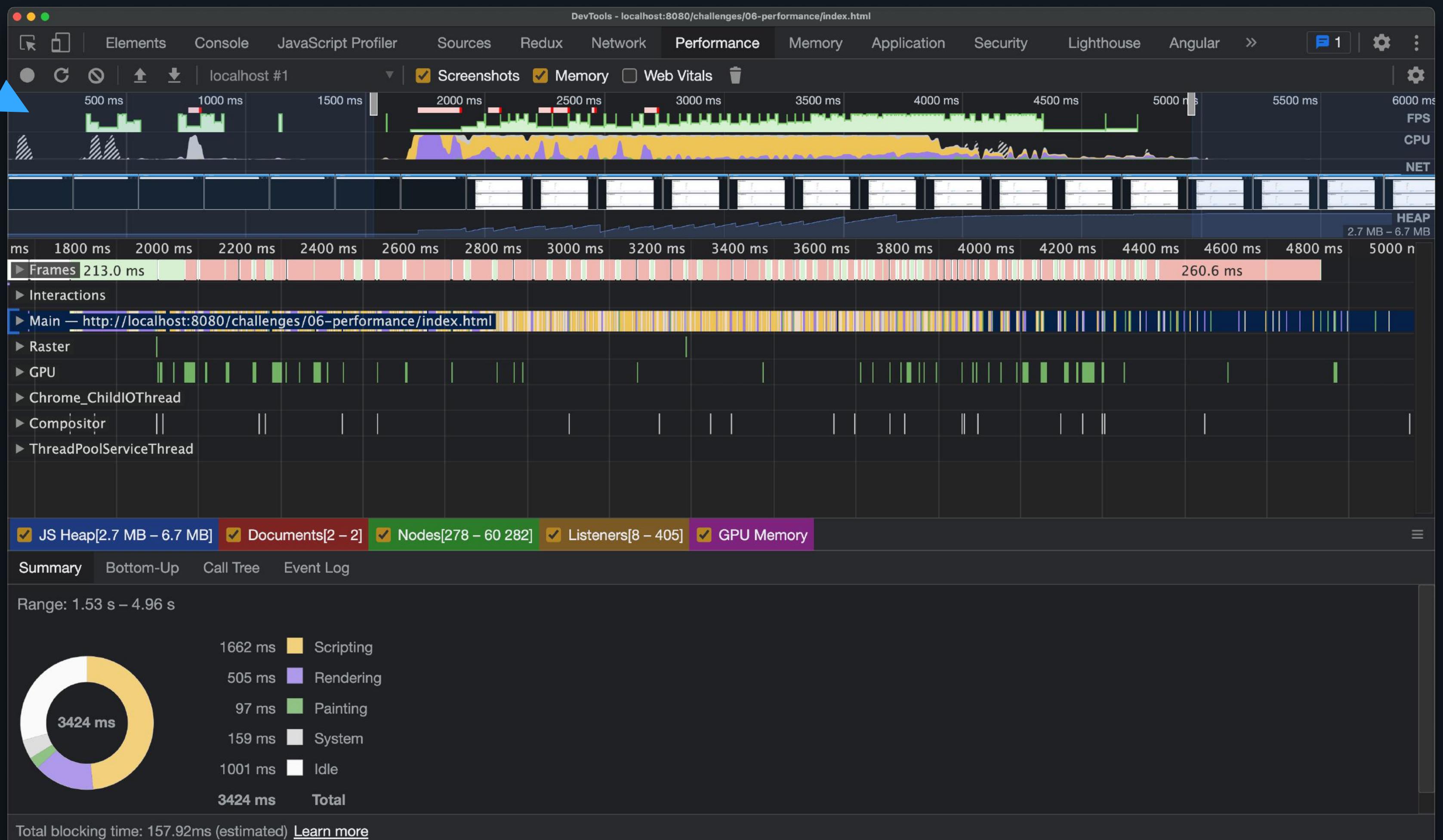
1. Open the **06 Performance** challenge in Chrome.
2. Open the **Performance** panel.
3. Click the **Screenshots** and **Memory** checkboxes to enable both features.
4. Enter the state “Colorado” into the search field.
5. Clock the **Stop** button to stop for recording.

Performance

Frames per second

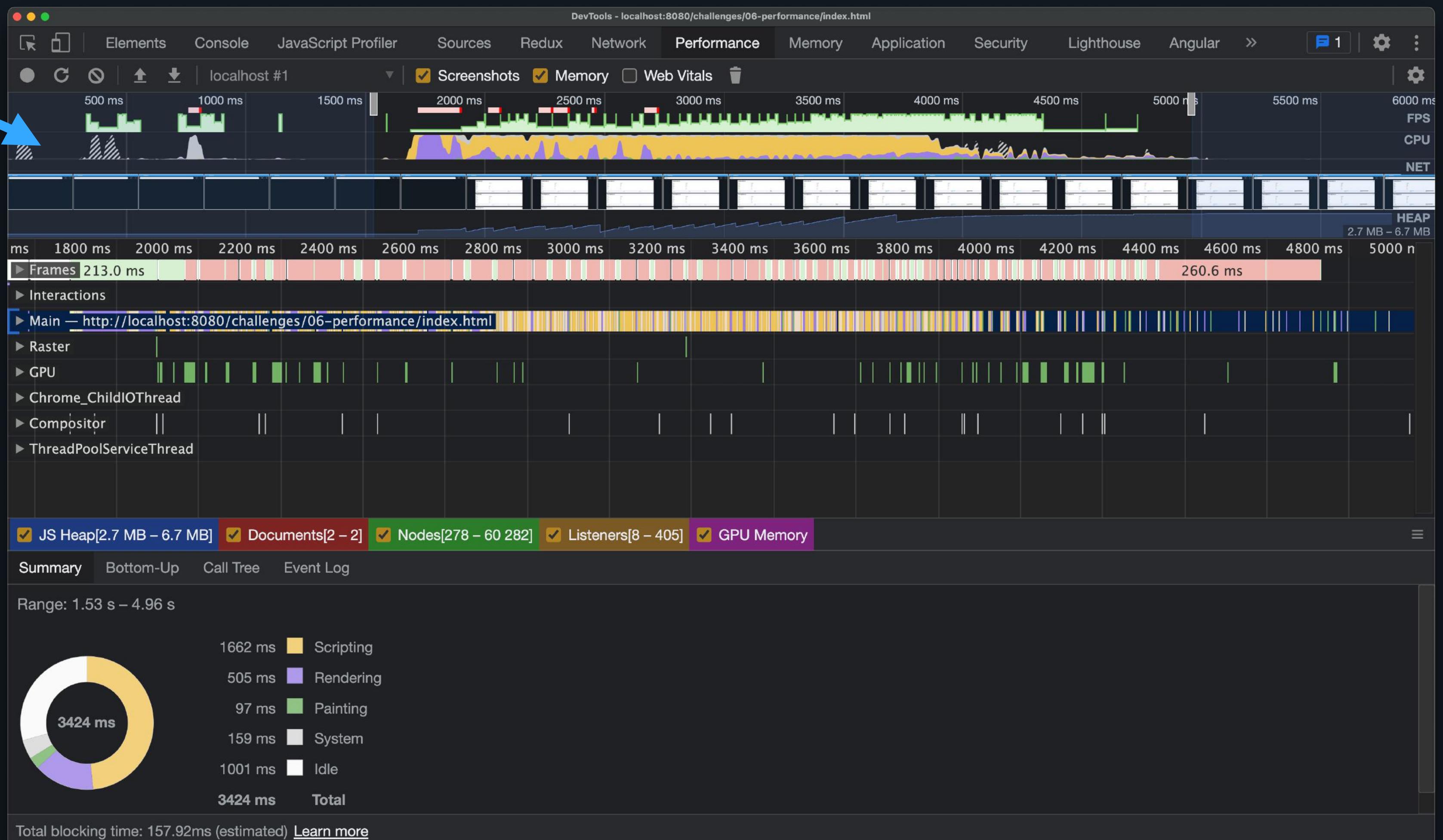
Green 

Red 



Performance

CPU summary



CPU categories

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

Scripting

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

Scripting

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

Scripting

-  XHR ready state change
-  XHR load

Rendering

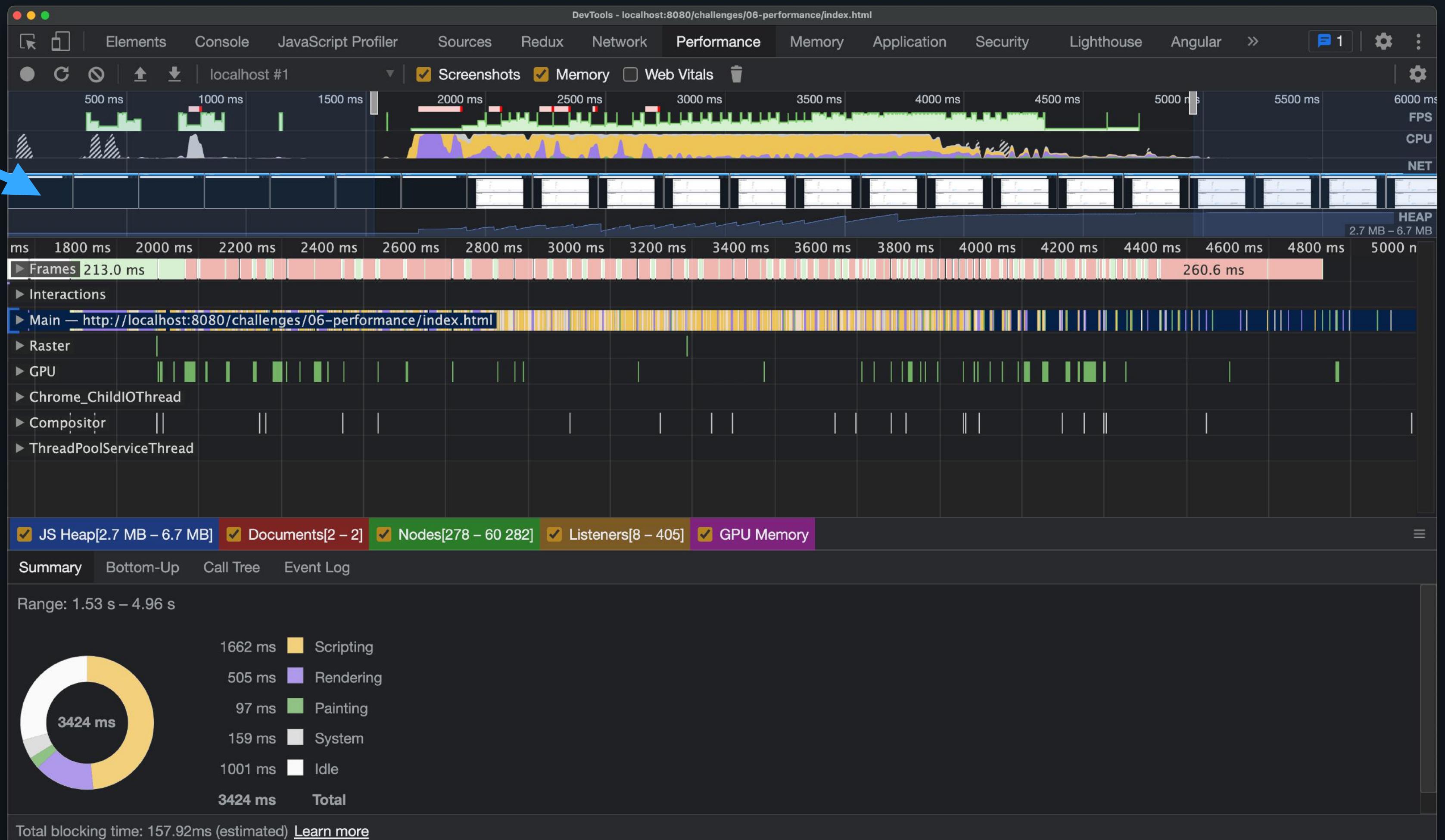
- Invalidate layout
- Layout
- Recalculate style
- Scroll

Painting

- Composite layers
- Image decode
- Image resize
- Paint

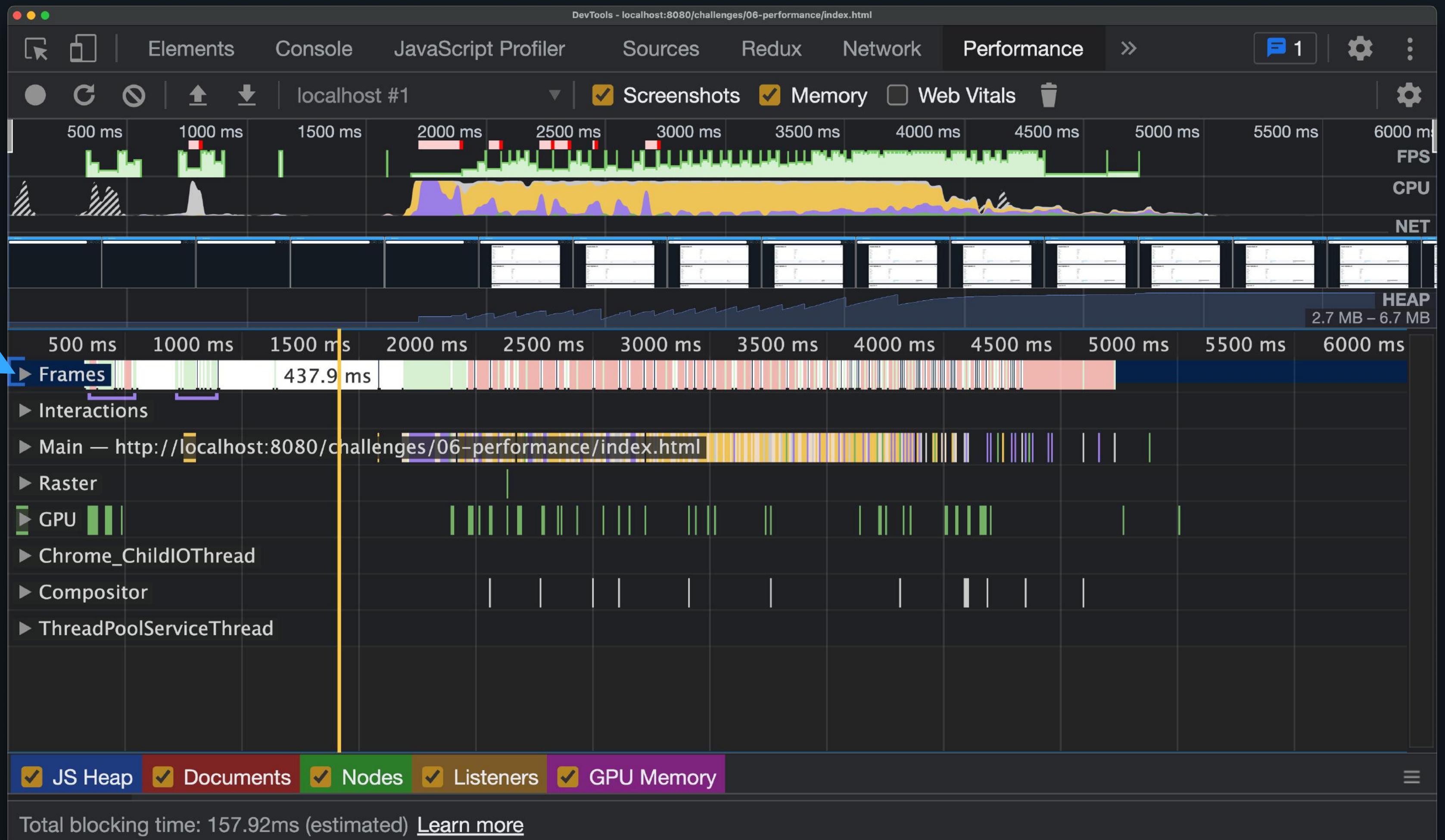
Performance

Screenshots



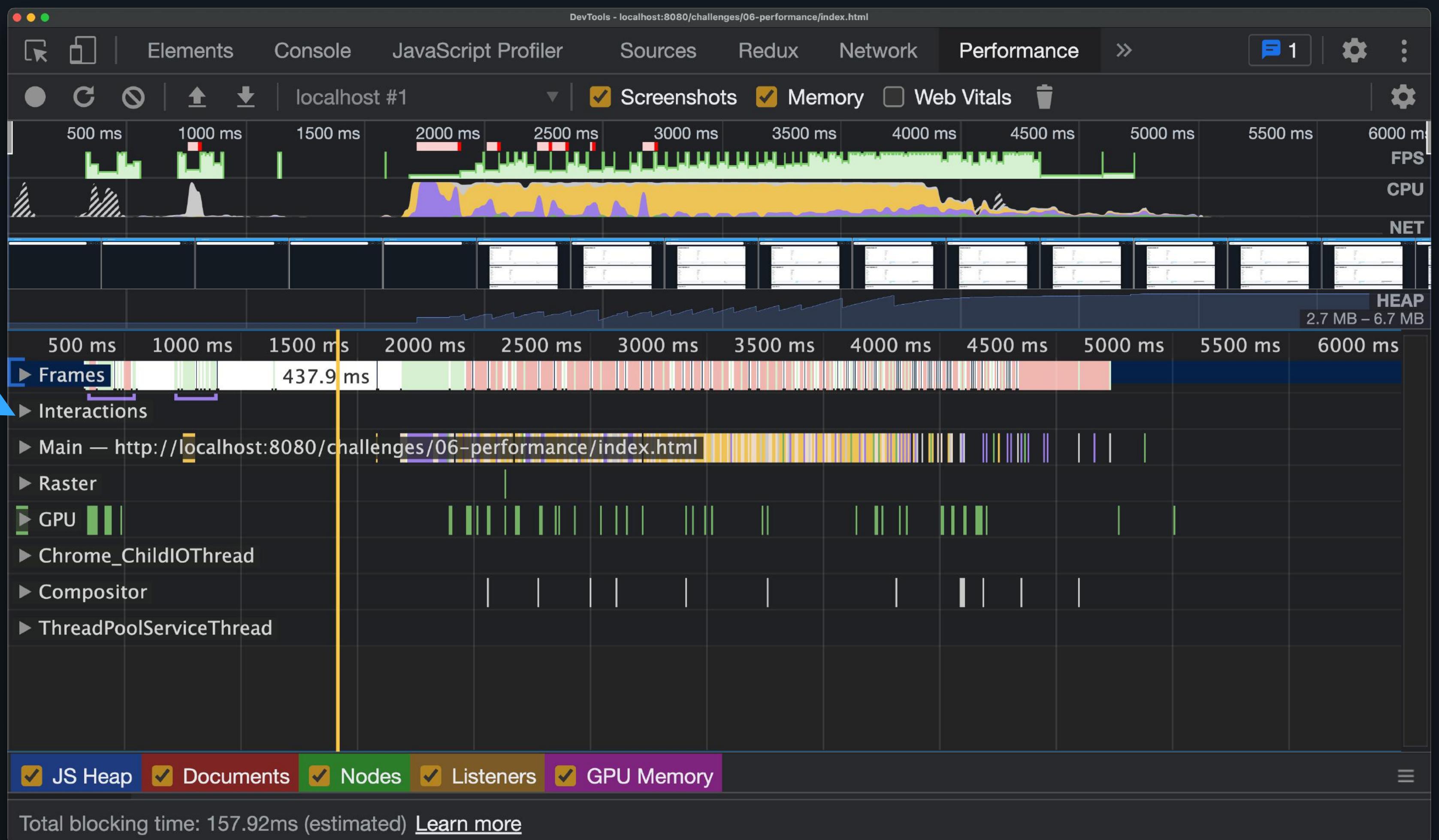
Performance

Frame rate per second activity



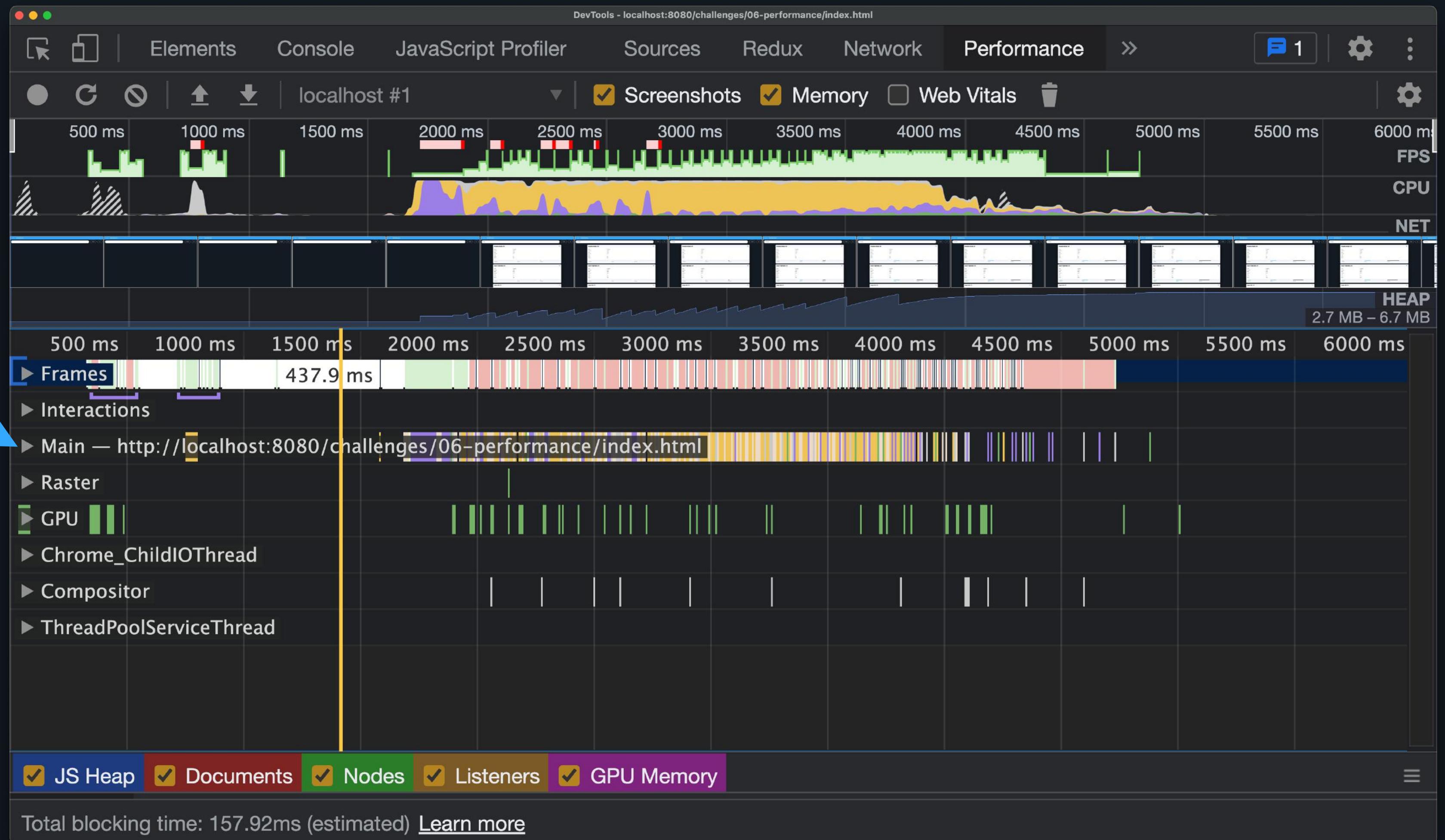
Performance

Interactions (by the user) activity



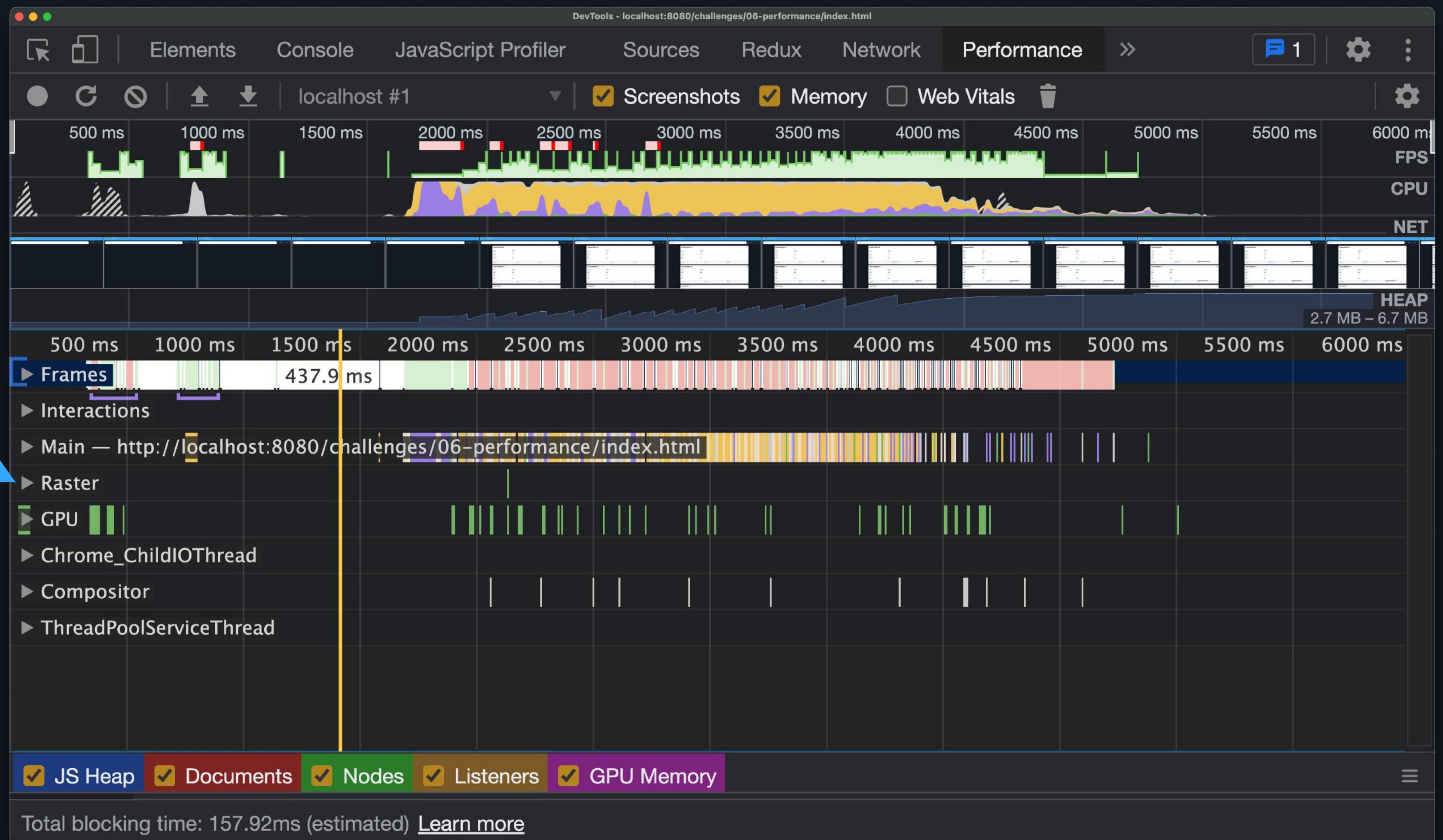
Performance

Main thread activity
including call stacks
(aka flame charts)

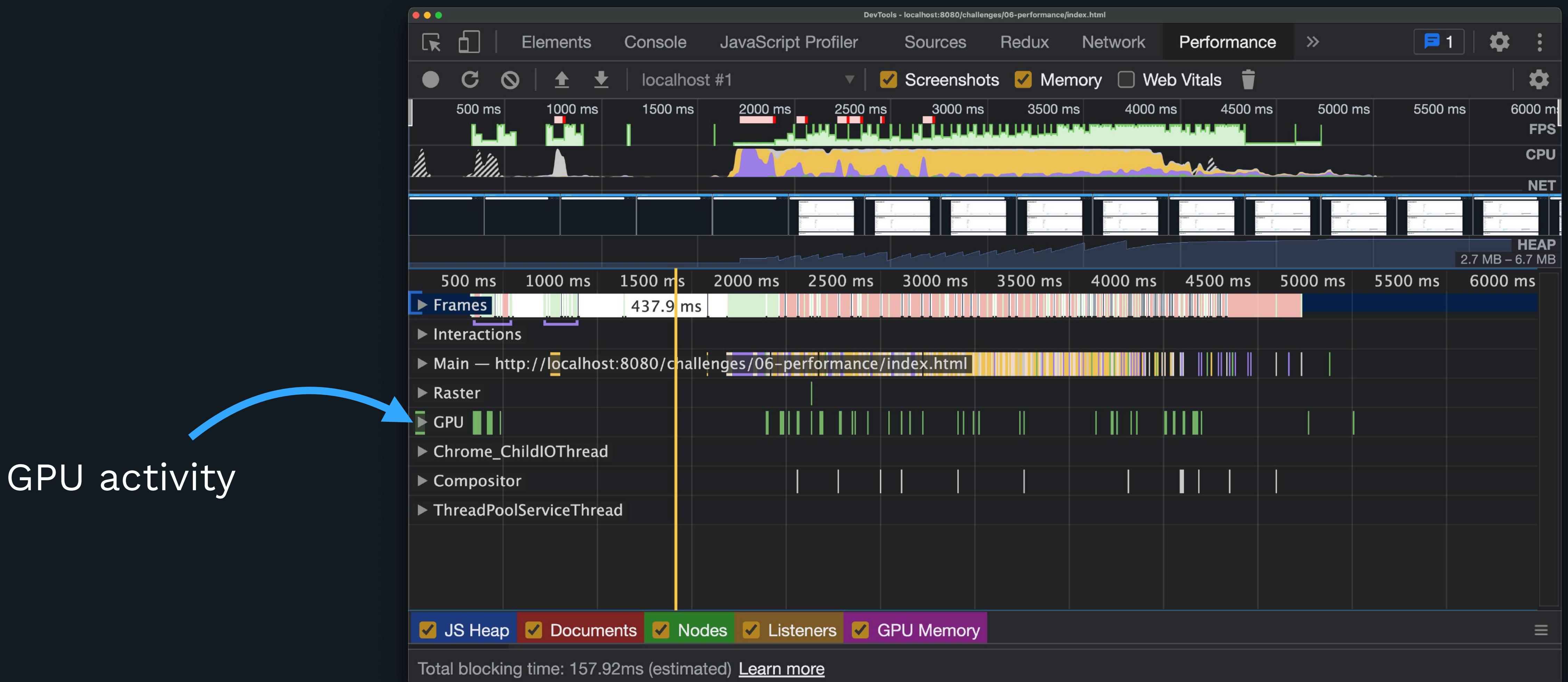


Performance

Rasterizer thread activity

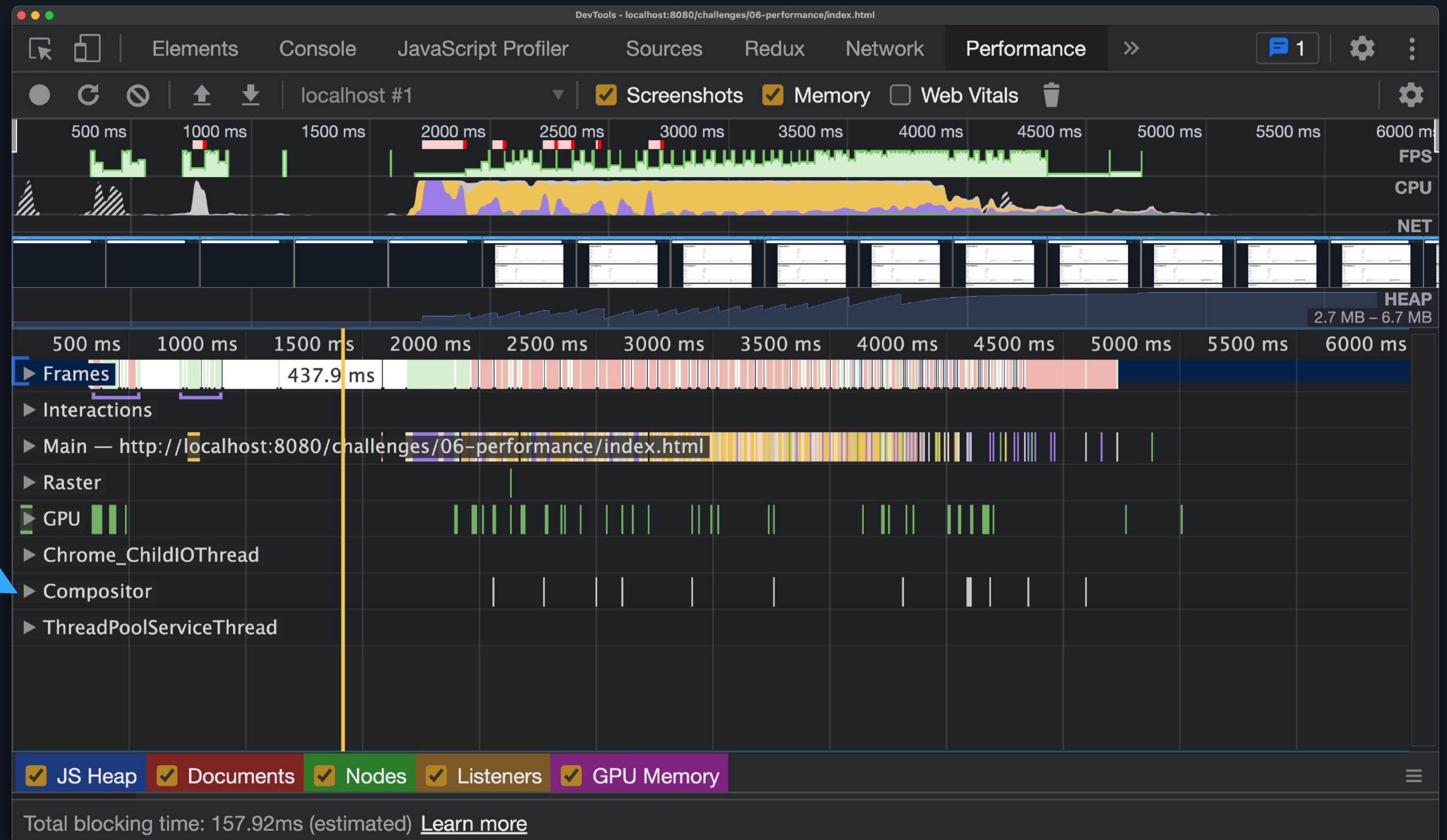


Performance



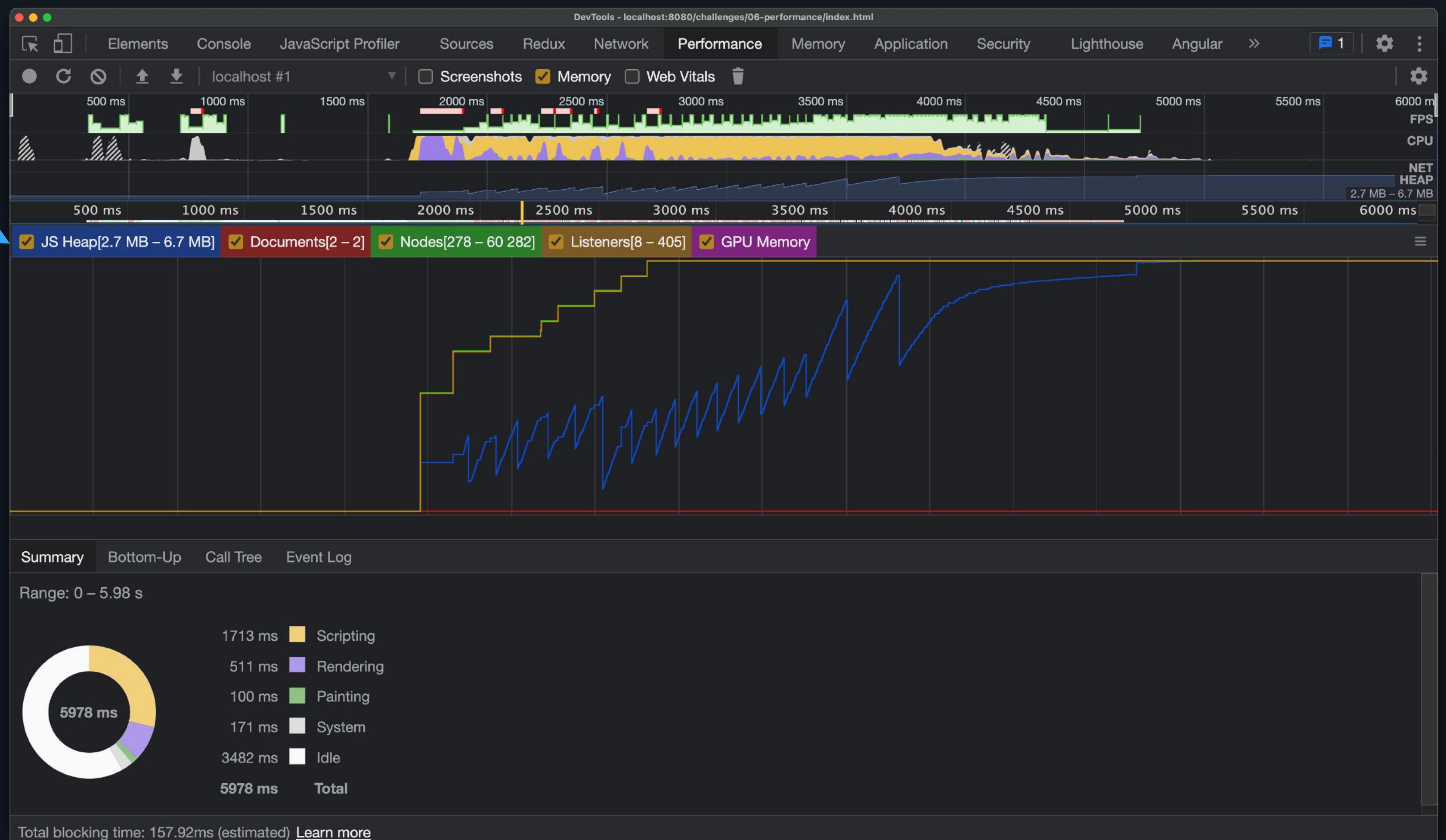
Performance

Compositor activity



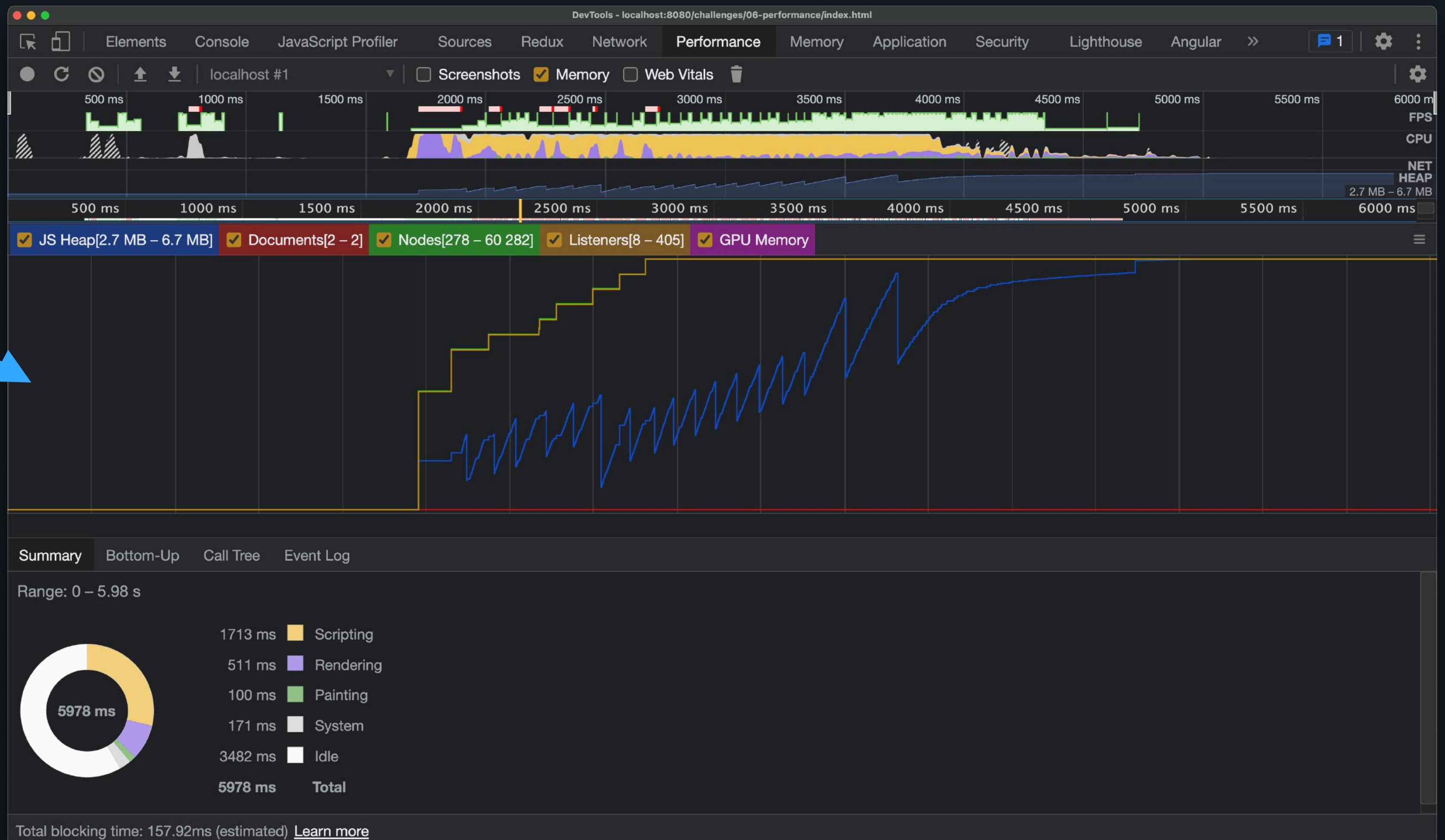
Performance

Toggle performance metrics



Performance

Graph of performance metrics

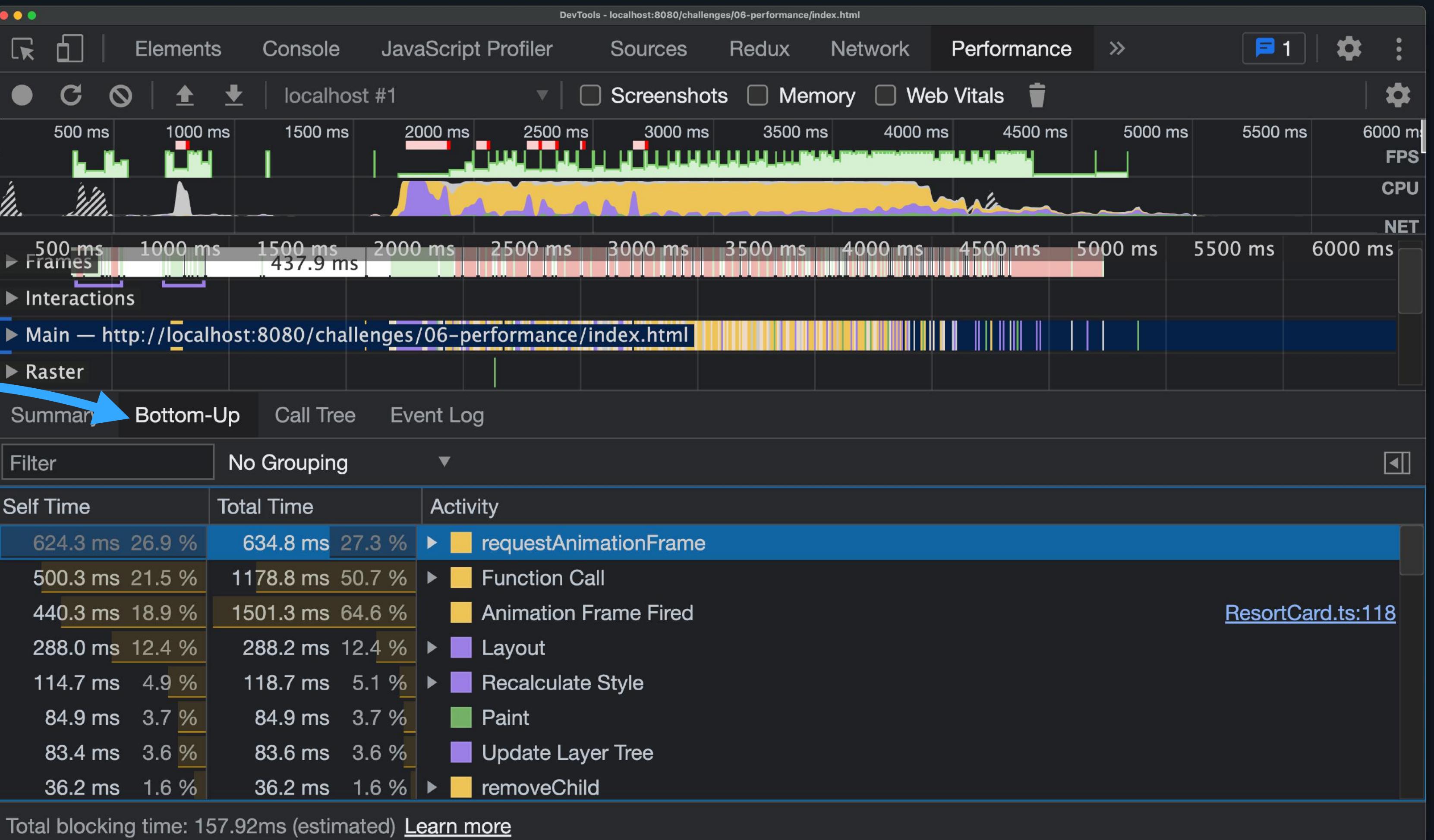


Performance

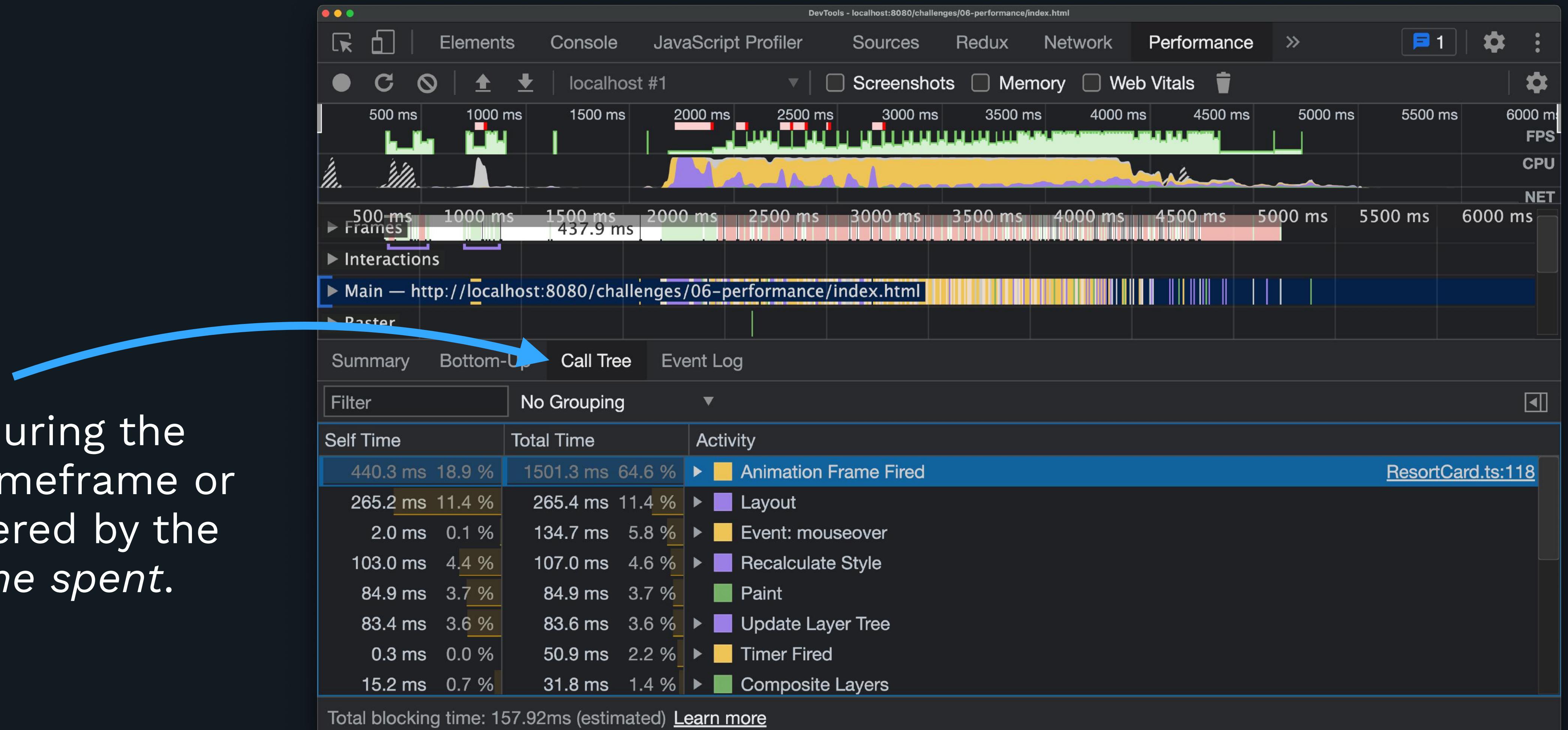
Performance summary
during the selected
timeframe or event



Performance

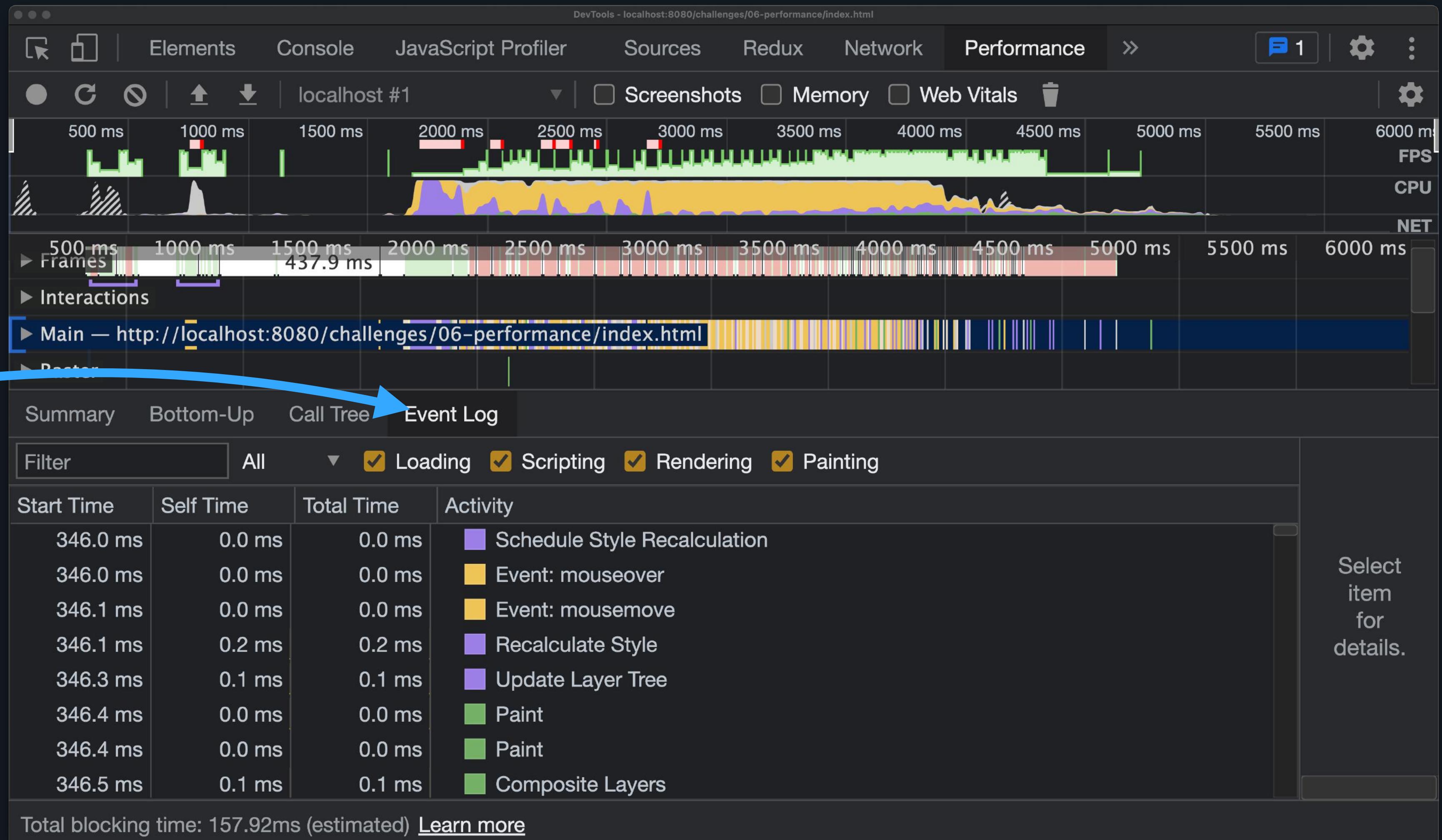


Performance



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

Performance

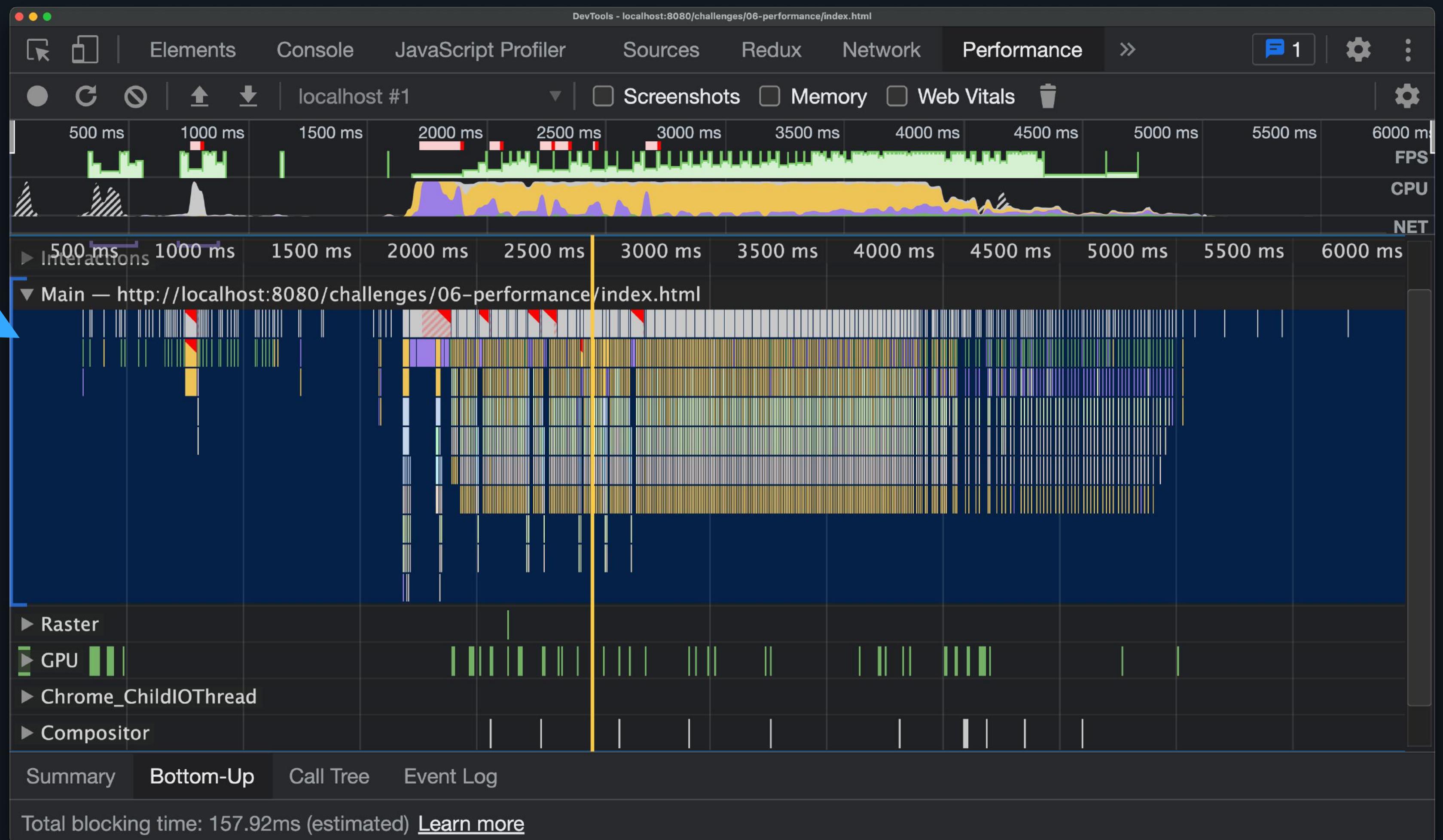


Events log in the order
in which they occurred.

Select
item
for
details.

Performance

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



Performance

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

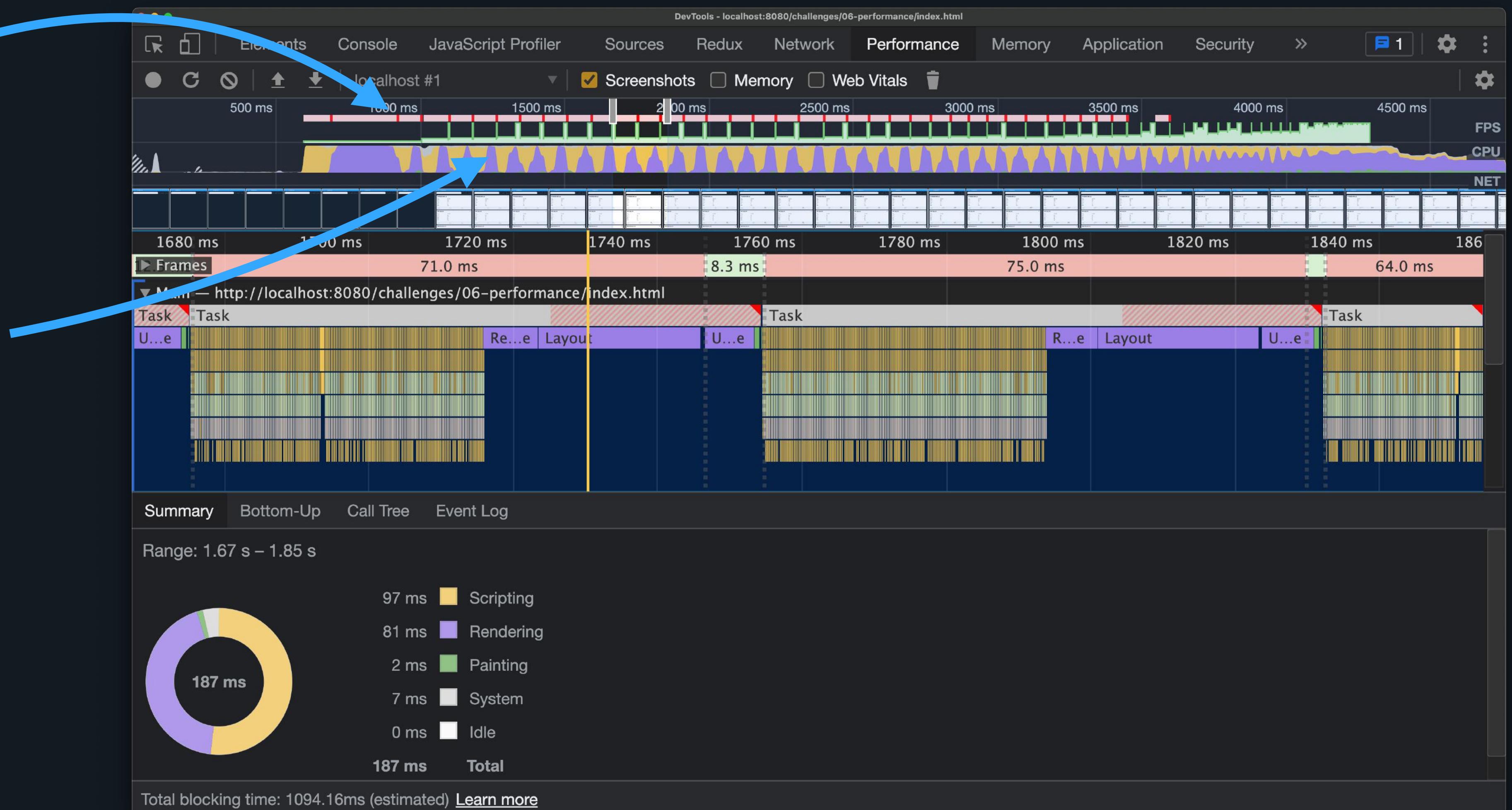
Performance Analysis 1

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

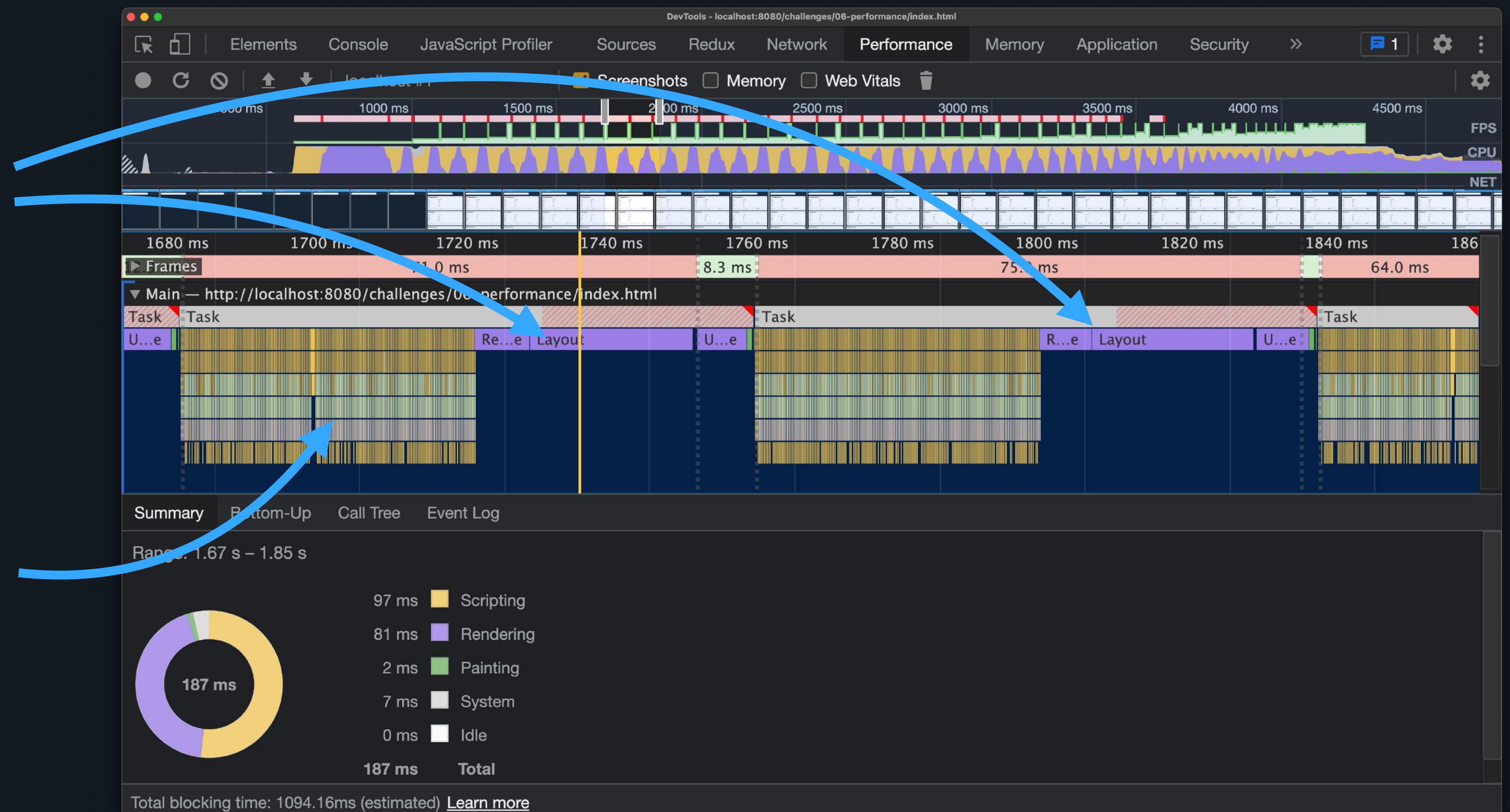
Woah, lots of dropped frames here.

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



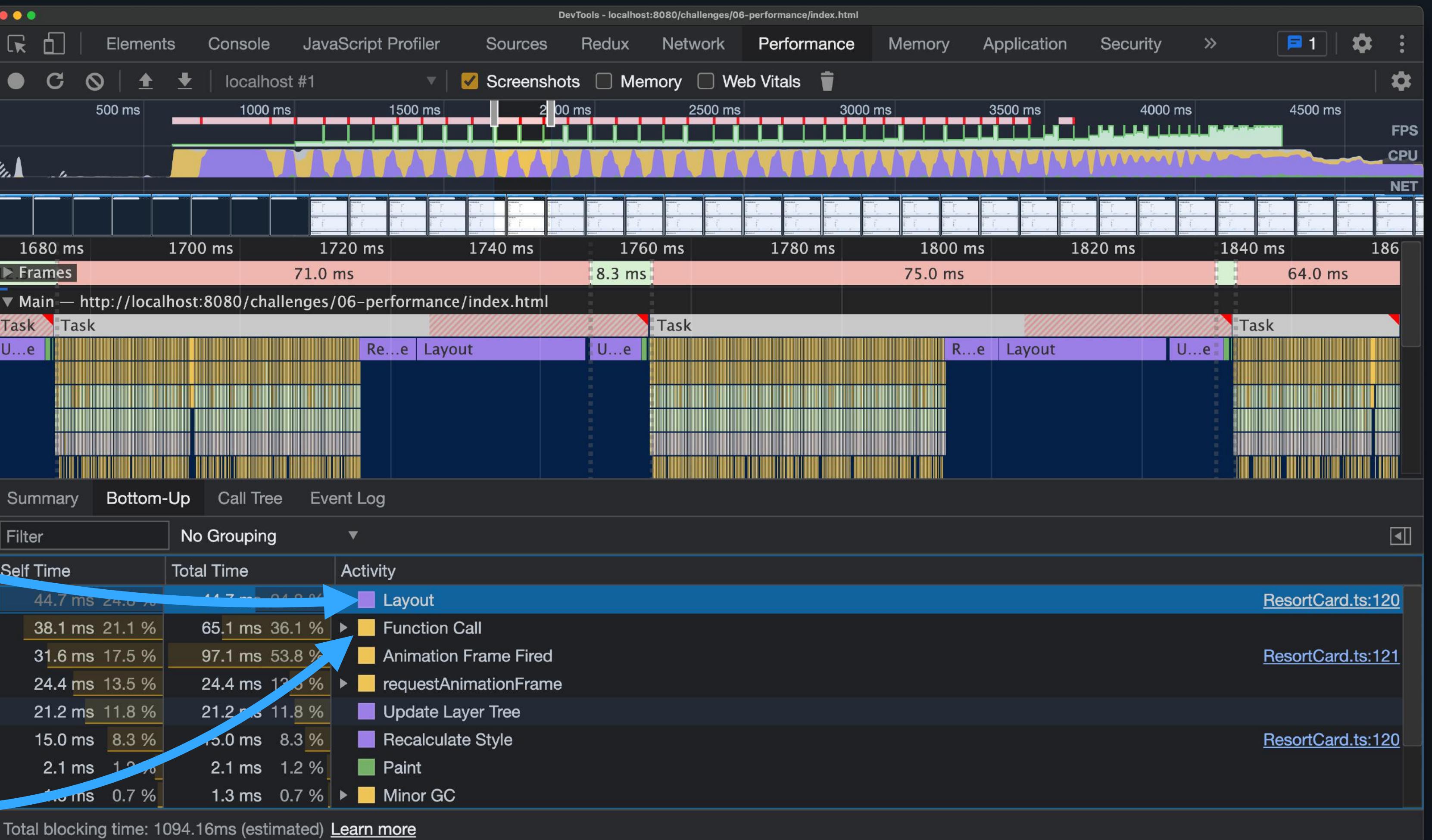
Performance Analysis 1

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

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

Refactor 1

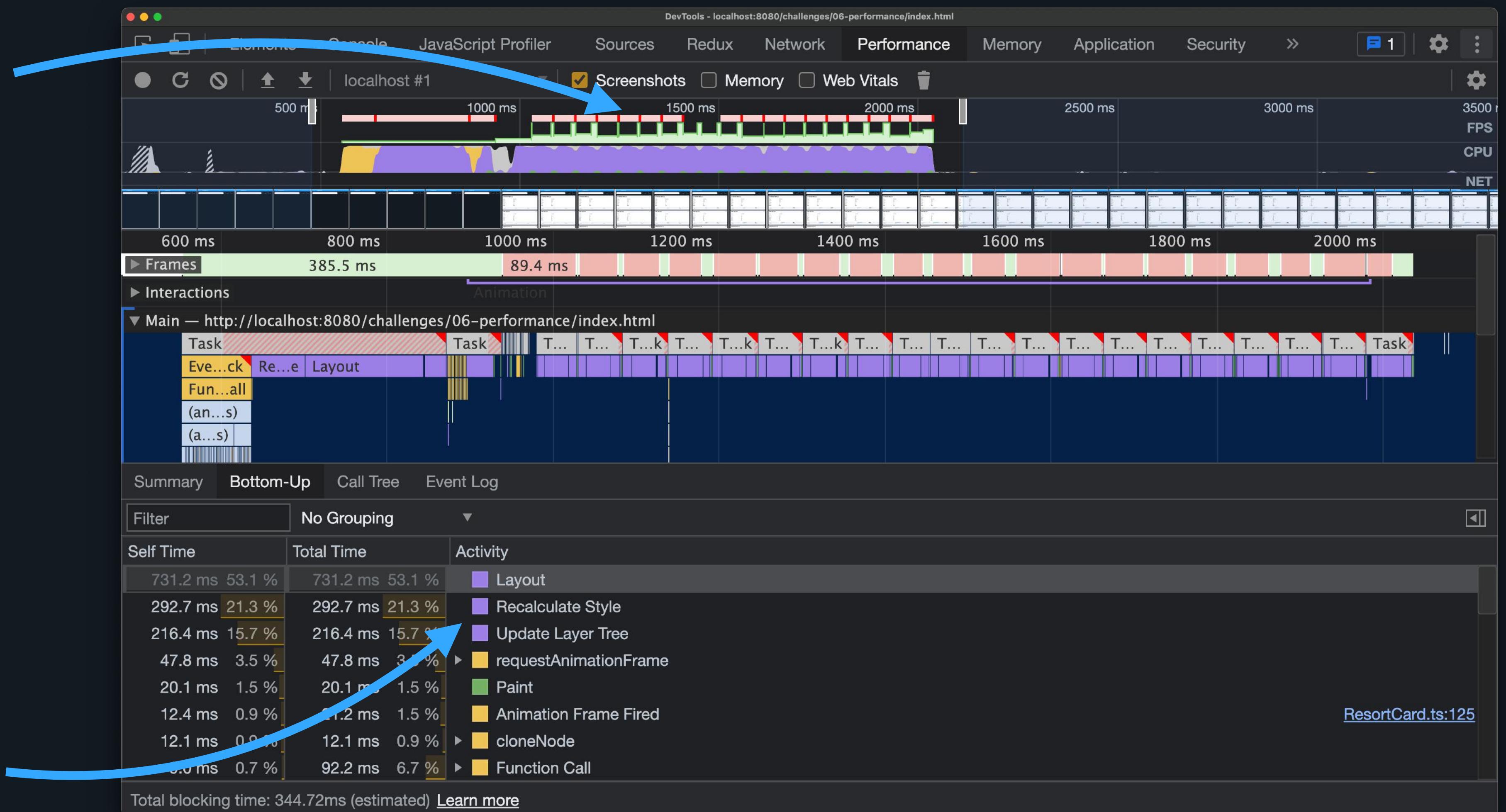
1. Based on the performance analysis we are observing LOTS of recalculation of styles, layout and updating the layer tree. The compositor is busy! 
2. Open components/ResortCard.ts and determine how the animation is being performed. Yikes! 
3. Remove the **animatePercent()** method and update the **setAcresPercent()** 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

- ✓ Remaining issues: layout and fps
- ✓ Compositor is less busy
- ✓ Expensive function as been removed
- ✓ 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

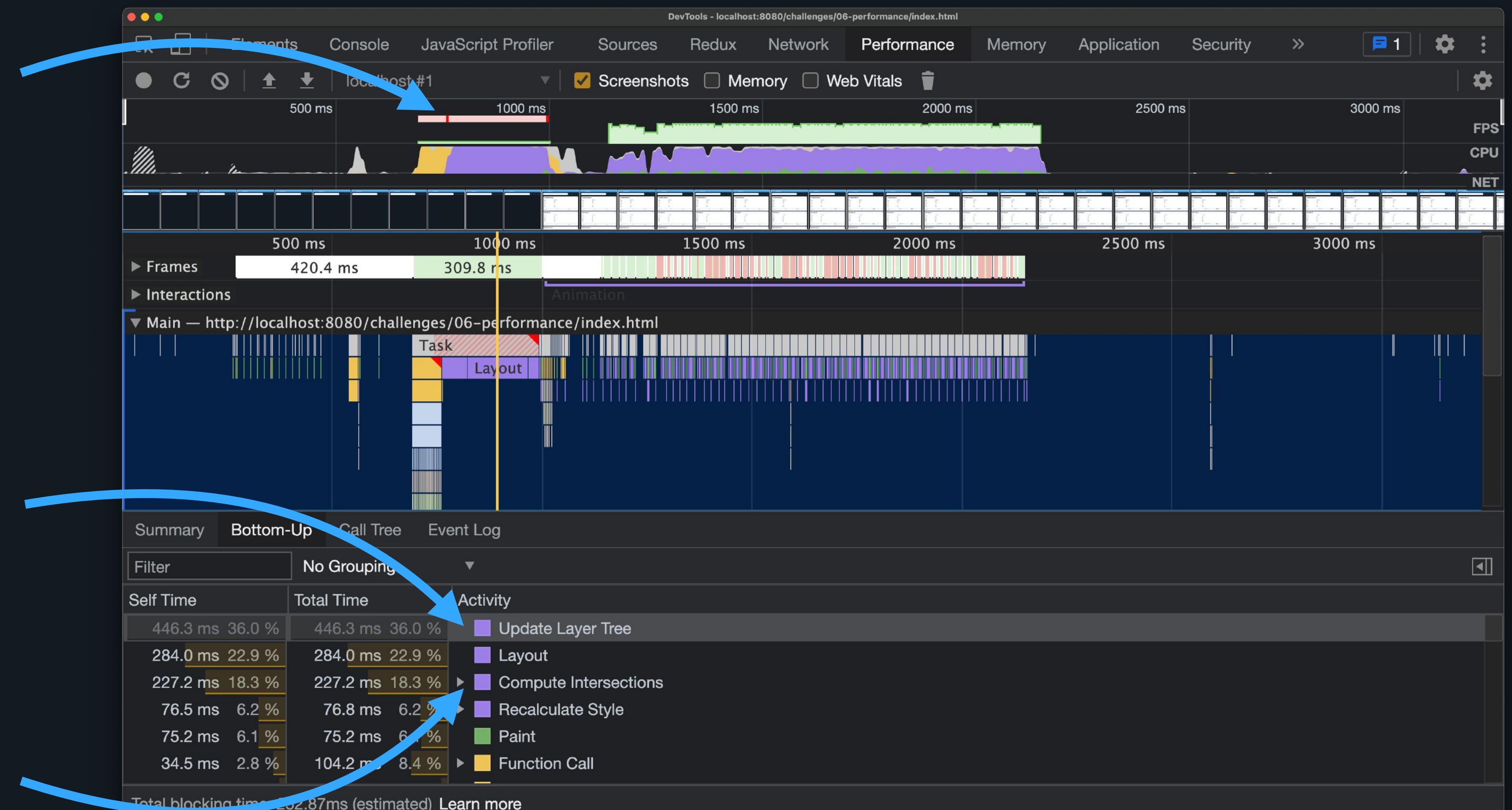
1. Open the components/ResortCard.ts file.
2. Instantiate a **new IntersectionObserver()** and observe the **percentEl**.
3. In the callback function, first check the length of the **entries** and then get the first **entry**.
4. Check the entry **isIntersecting** property to determine if the element 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

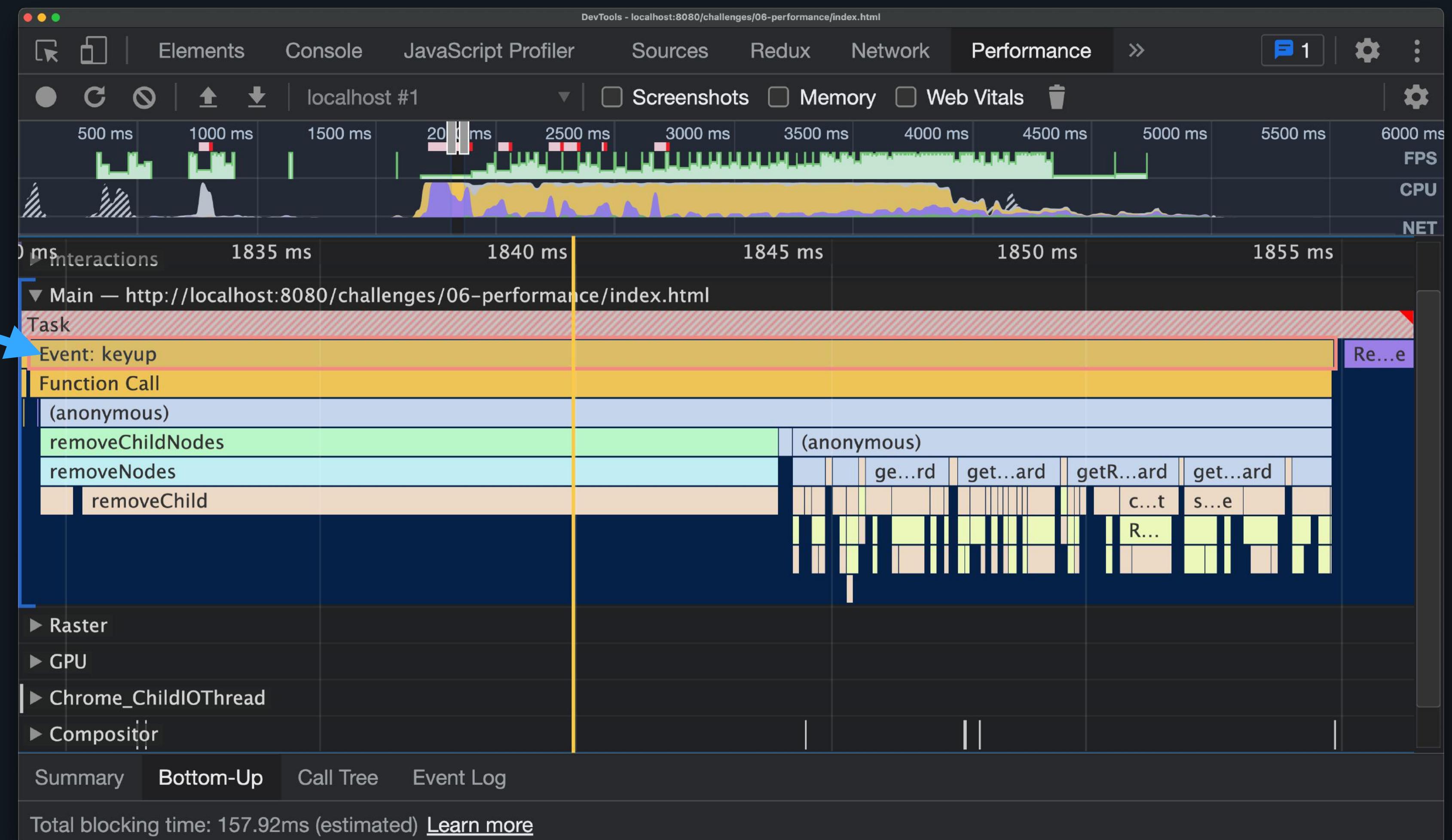
Big improvement in fps!



Computing
intersections is a new
issue.

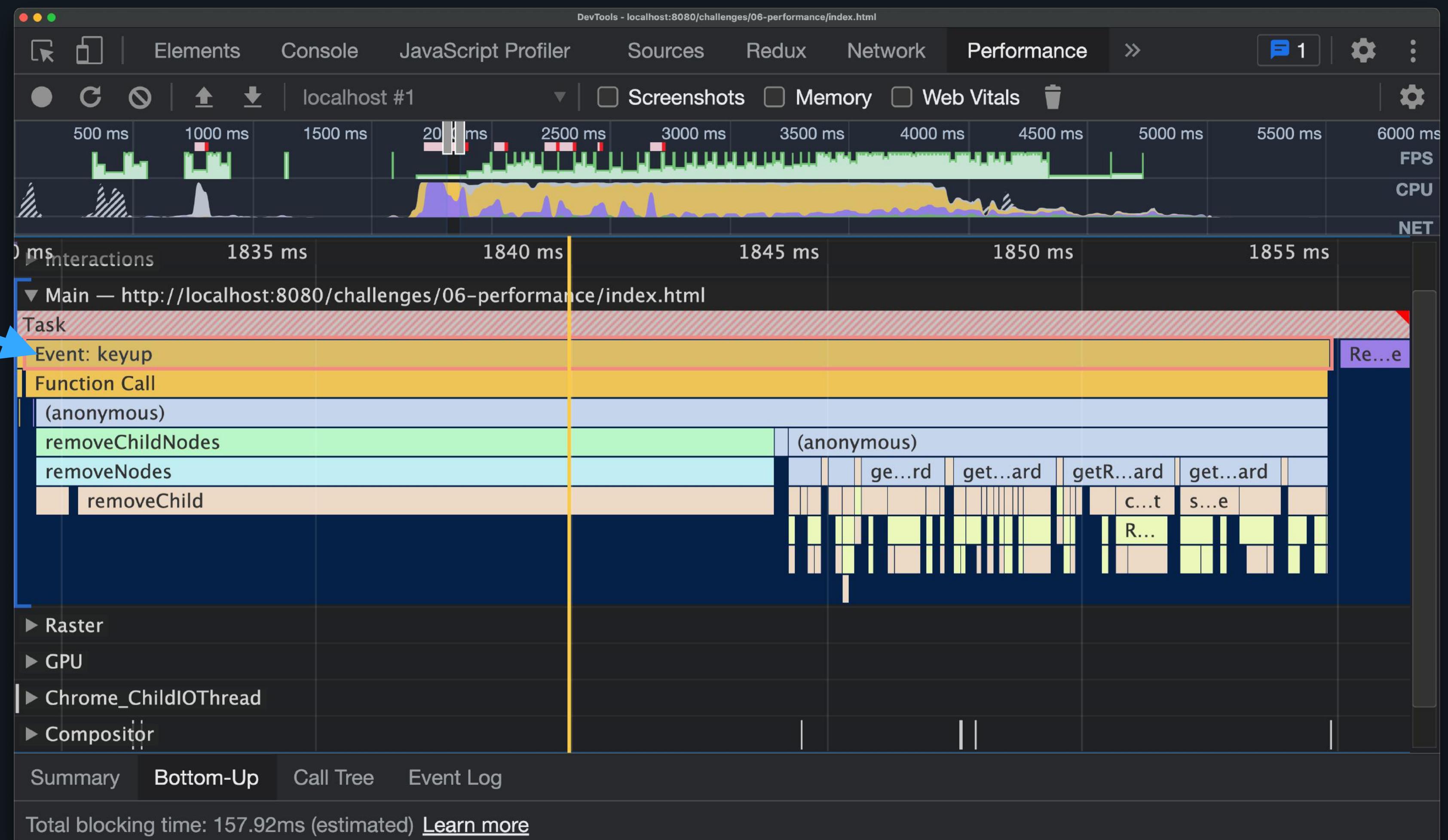
Analyzing Single Event

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

The screenshot shows the Chrome DevTools interface with the "Memory" tab selected. A "HEAP SNAPSHTOS" panel on the left displays "Snapshot 1" (3.0 MB) with a "Save" button. The main area is a table titled "Summary" showing memory usage by object type. The table has columns for Distance, Shallow Size, and Retained Size. The "Constructor" row is at the top, followed by various array, compiled code, closure, and window context objects.

| | Distance | Shallow Size | Retained Size |
|----------------------------|----------|--------------|---------------|
| Constructor | 2 | 1 860 420 | 56 % |
| ▸ (system) | 2 | 519 832 | 17 % |
| ▸ (array) | 2 | 357 808 | 12 % |
| ▸ (compiled code) | 3 | 165 532 | 6 % |
| ▸ (closure) | 2 | 72 | 0 % |
| ▸ Window / https://web.dev | 1 | 529 652 | 18 % |
| ▸ system / Context | 3 | 11 060 | 0 % |
| ▸ (string) | 3 | 209 884 | 7 % |
| ▸ Object | 2 | 9 980 | 0 % |
| ▸ Object / | 1 | 40 | 0 % |
| ▸ Generator | 3 | 660 | 0 % |
| ▸ Window | 2 | 232 | 0 % |
| ▸ Array | 2 | 2 560 | 0 % |
| ▸ (regexp) | 4 | 1 204 | 0 % |
| ▸ HTMLElement | 3 | 1 184 | 0 % |
| ▸ Map | 3 | 1 392 | 0 % |
| ▸ ee | 4 | 120 | 0 % |
| ▸ InternalNode | 3 | 0 | 0 % |
| ▸ e | 4 | 1 576 | 0 % |
| ▸ R | 6 | 928 | 0 % |
| ▸ HTMLBodyElement | 4 | 84 | 0 % |
| ▸ Math | 2 | 112 | 0 % |
| ▸ Error | 3 | 1 180 | 0 % |
| ▸ TypedArray | 3 | 1 232 | 0 % |
| ▸ console | 2 | 48 | 0 % |
| ▸ String | 3 | 64 | 0 % |
| ▸ HTMLImageElement | 4 | 84 | 0 % |
| ▸ bb | 6 | 4 640 | 0 % |
| ▸ _ | 7 | 392 | 0 % |
| ▸ Document | 3 | 28 | 0 % |
| ▸ DataView | 3 | 112 | 0 % |
| ▸ Xe | 6 | 64 | 0 % |
| ▸ Jt | 11 | 136 | 0 % |
| ▸ HTMLLinkElement | 4 | 84 | 0 % |
| ▸ (concatenated string) | 4 | 3 120 | 0 % |
| ▸ Intl | 2 | 112 | 0 % |
| ▸ Intl.Locale | 3 | 28 | 0 % |
| ▸ CustomElementRegistry | 4 | 112 | 0 % |
| ▸ pc | 3 | 84 | 0 % |
| ▸ Promise | 4 | 52 | 0 % |
| ▸ CustomElementDefinition | 3 | 592 | 0 % |
| | 5 | 0 | 0 % |
| Retainers | | | |
| Object | | | |

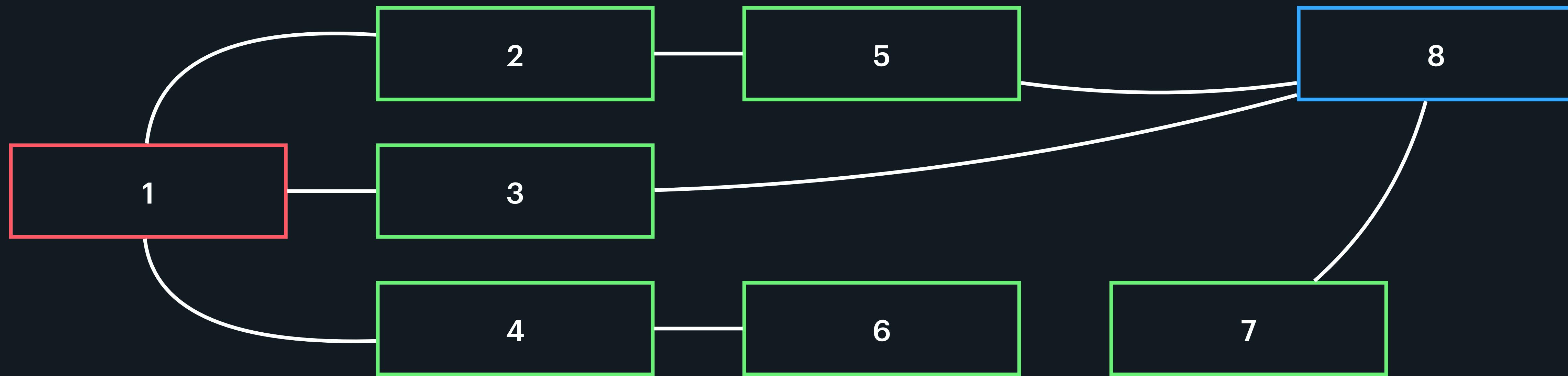
Object Size

-  Primitive types: numbers, booleans, and strings
-  Objects

Object Size

- Directly contains primitives and/or objects
- Implicitly via references

Memory



Root (1) Objects

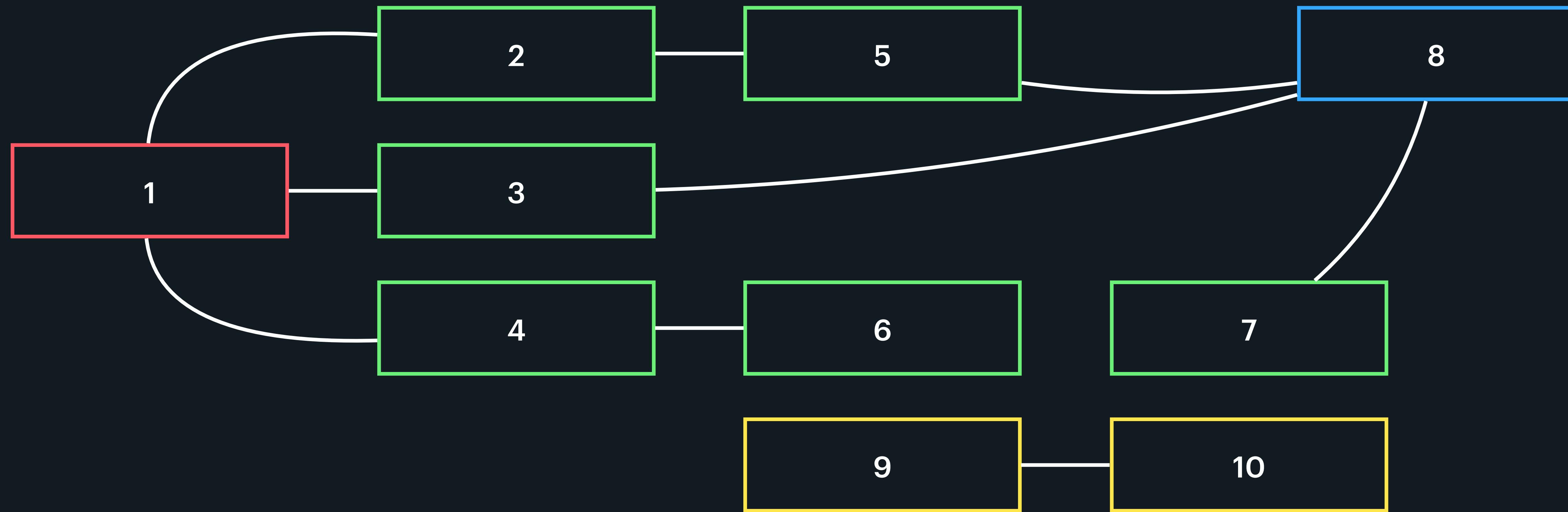
-  **window or globalThis**
-  Document DOM
-  Retained by debugger (breakpoints)

**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

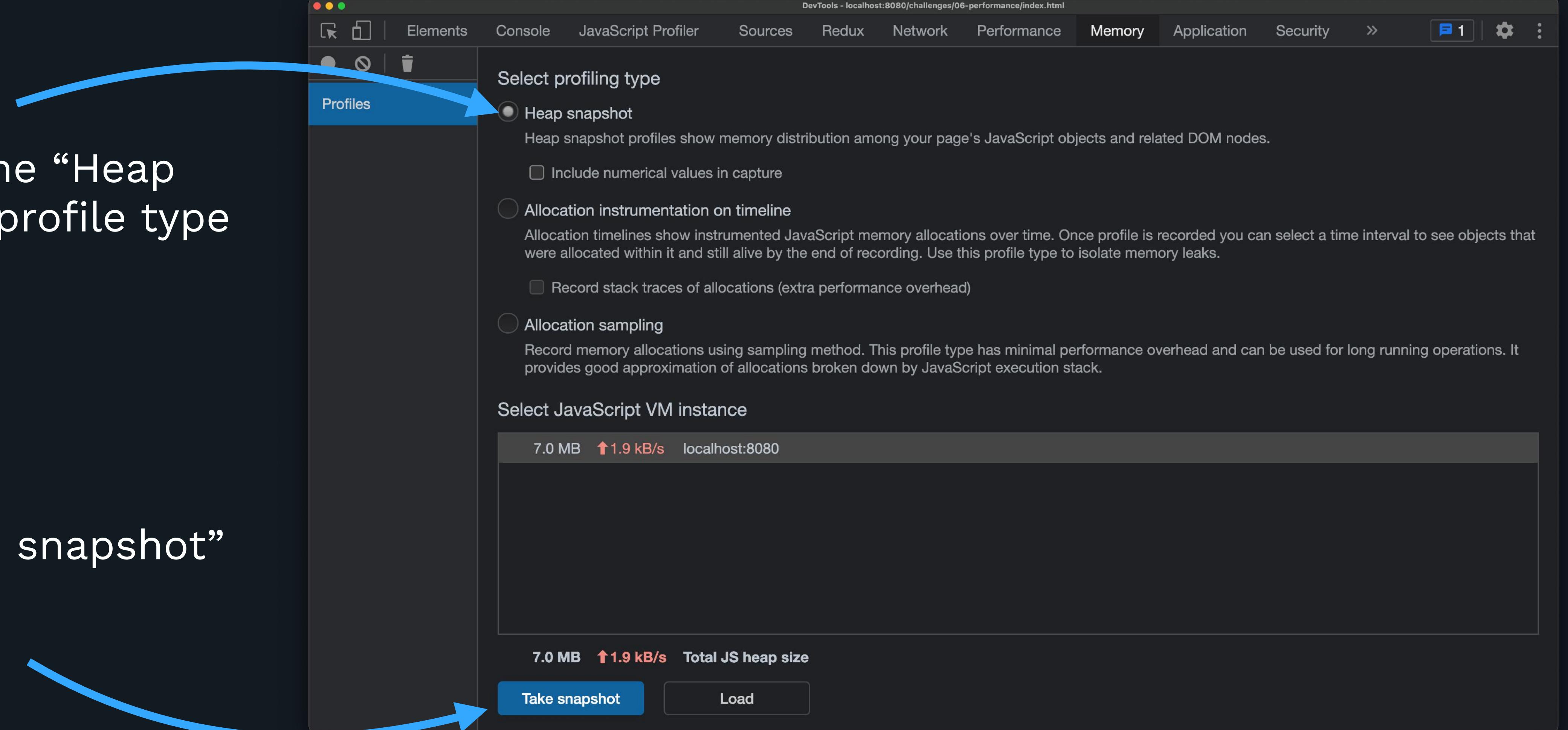
-  Major GC is performed when the old generation grows beyond a limit
-  Uses a mark-and-sweep collection
-  Goal is to keep marking < 5 ms
-  Memory is freed after marking and sweeping the entire old generation

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 phase: sequential pass over memory to recycle all unmarked objects

Memory Panel

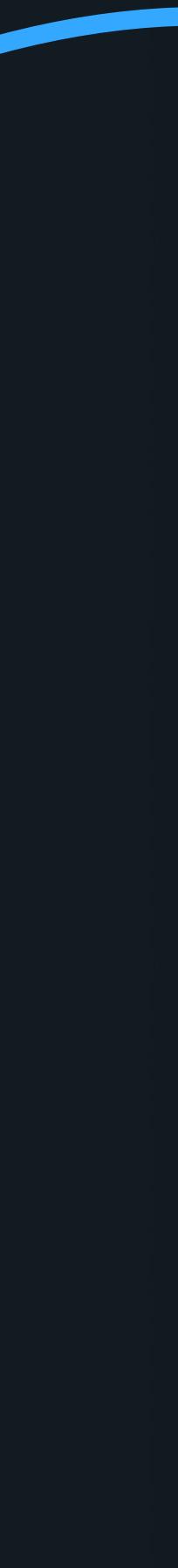
Select the “Heap snapshot” profile type



Click “Take snapshot”

Memory Panel

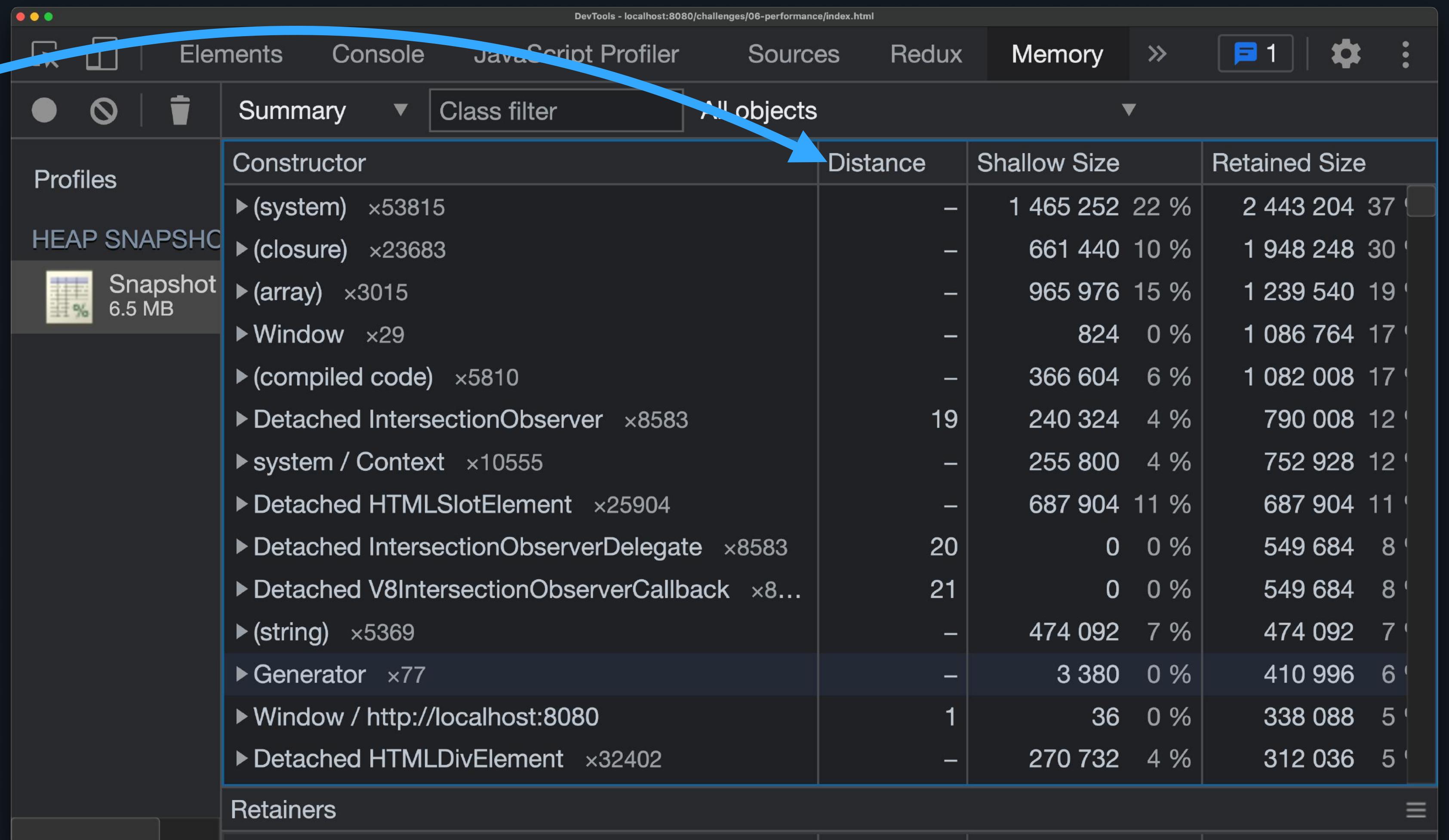
All objects created using the constructor



| Profiles | Summary | Class filter | All objects | Distance | Shallow Size | Retained Size |
|---------------|---|--------------|-------------|----------|----------------|----------------|
| HEAP SNAPSHOT | Constructor | | | - | 1 465 252 22 % | 2 443 204 37 % |
| | ▶ (system) ×53815 | | | - | 661 440 10 % | 1 948 248 30 % |
| | ▶ (closure) ×23683 | | | - | 965 976 15 % | 1 239 540 19 % |
| | ▶ (array) ×3015 | | | - | 824 0 % | 1 086 764 17 % |
| | ▶ Window ×29 | | | - | 366 604 6 % | 1 082 008 17 % |
| | ▶ (compiled code) ×5810 | | | - | 240 324 4 % | 790 008 12 % |
| | ▶ Detached IntersectionObserver ×8583 | | | 19 | 255 800 4 % | 752 928 12 % |
| | ▶ system / Context ×10555 | | | - | 687 904 11 % | 687 904 11 % |
| | ▶ Detached HTMLSlotElement ×25904 | | | - | 0 0 % | 549 684 8 % |
| | ▶ Detached IntersectionObserverDelegate ×8583 | | | 20 | 0 0 % | 549 684 8 % |
| | ▶ Detached V8IntersectionObserverCallback ×8... | | | 21 | 0 0 % | 549 684 8 % |
| | ▶ (string) ×5369 | | | - | 474 092 7 % | 474 092 7 % |
| | ▶ Generator ×77 | | | - | 3 380 0 % | 410 996 6 % |
| | ▶ Window / http://localhost:8080 | | | 1 | 36 0 % | 338 088 5 % |
| | ▶ Detached HTMLDivElement ×32402 | | | - | 270 732 4 % | 312 036 5 % |
| | Retainers | | | | | |

Memory Panel

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

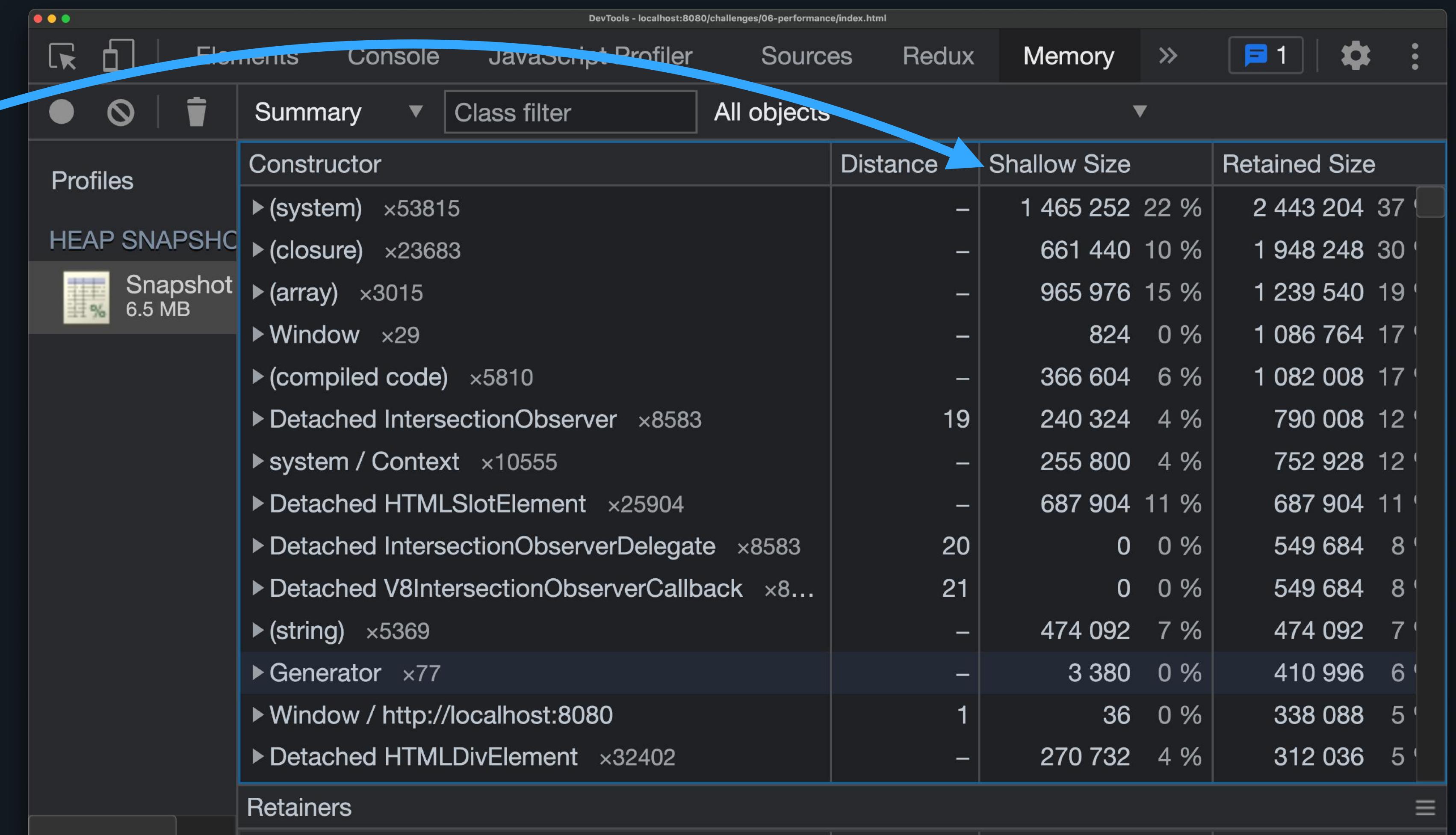


A screenshot of the Chrome DevTools Memory Panel. The title bar shows 'DevTools - localhost:8080/challenges/06-performance/index.html'. The tabs at the top are Elements, Console, JavaScript Profiler, Sources, Redux, Memory, and more. A blue arrow points from the text above to the 'Distance' column header in the table below. The table has columns: Profiles, Class filter, Summary, All objects, Distance, Shallow Size, and Retained Size. The 'Profiles' column shows 'HEAP SNAPSHOT' and 'Snapshot 6.5 MB'. The 'All objects' section lists various objects with their counts and memory usage. The 'Distance' column indicates the shortest path from the root object for each item.

| Profiles | Class filter | Summary | All objects | Distance | Shallow Size | Retained Size |
|---------------|--------------|--|-------------|----------|----------------|----------------|
| HEAP SNAPSHOT | | Constructor | | - | 1 465 252 22 % | 2 443 204 37 % |
| | | ▶(system) ×53815 | | - | 661 440 10 % | 1 948 248 30 % |
| | | ▶(closure) ×23683 | | - | 965 976 15 % | 1 239 540 19 % |
| | | ▶(array) ×3015 | | - | 824 0 % | 1 086 764 17 % |
| | | ▶Window ×29 | | - | 366 604 6 % | 1 082 008 17 % |
| | | ▶(compiled code) ×5810 | | - | 240 324 4 % | 790 008 12 % |
| | | ▶Detached IntersectionObserver ×8583 | | 19 | 255 800 4 % | 752 928 12 % |
| | | ▶system / Context ×10555 | | - | 687 904 11 % | 687 904 11 % |
| | | ▶Detached HTMLSlotElement ×25904 | | - | 0 0 % | 549 684 8 % |
| | | ▶Detached IntersectionObserverDelegate ×8583 | | 20 | 0 0 % | 549 684 8 % |
| | | ▶Detached V8IntersectionObserverCallback ×8... | | 21 | 0 0 % | 549 684 8 % |
| | | ▶(string) ×5369 | | - | 474 092 7 % | 474 092 7 % |
| | | ▶Generator ×77 | | - | 3 380 0 % | 410 996 6 % |
| | | ▶Window / http://localhost:8080 | | 1 | 36 0 % | 338 088 5 % |
| | | ▶Detached HTMLDivElement ×32402 | | - | 270 732 4 % | 312 036 5 % |

Memory Panel

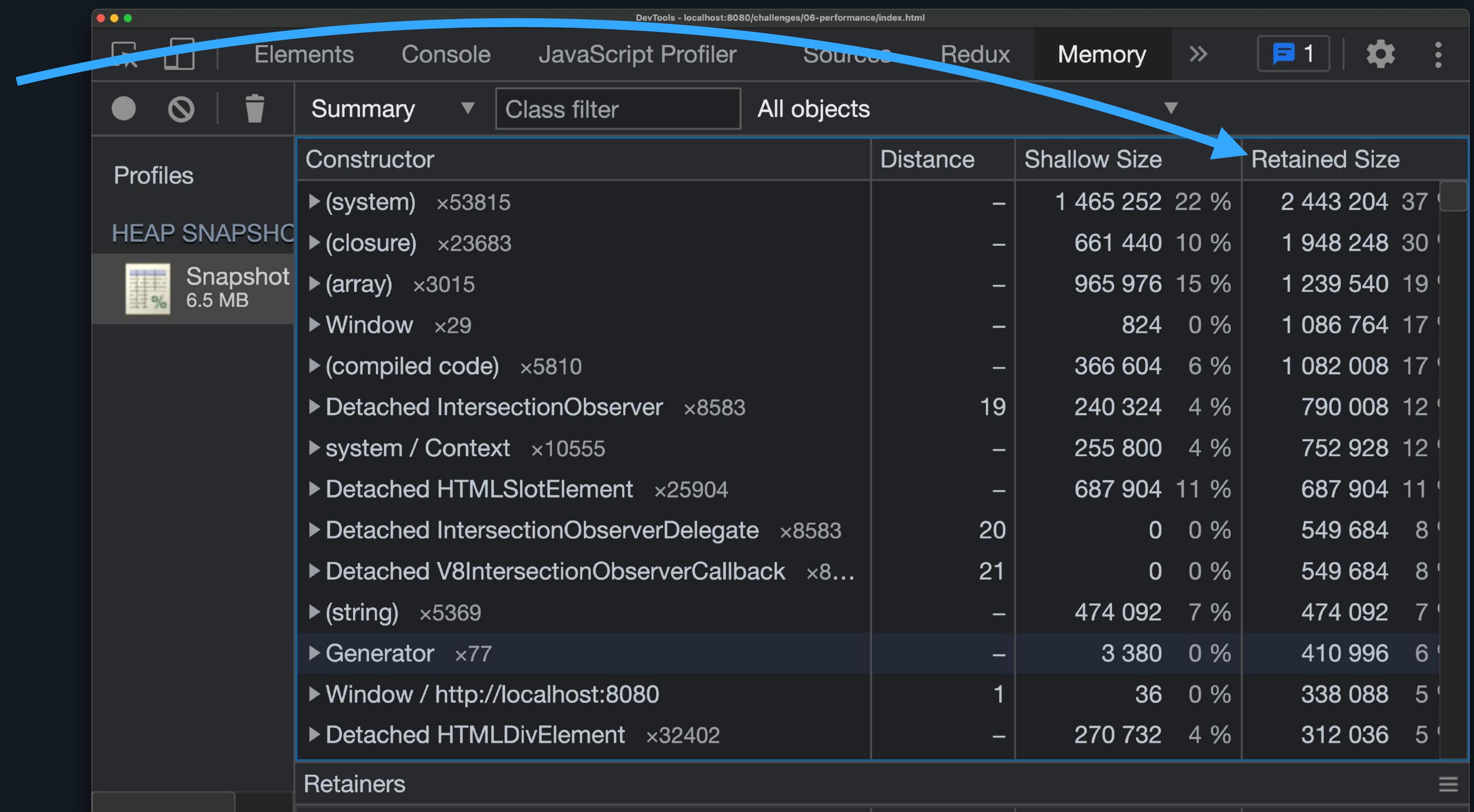
Sum of shallow sizes of Objects



| Profiles | Constructor | Distance | Shallow Size | Retained Size |
|-----------------|--|----------|----------------|----------------|
| HEAP SNAPSHOT | ▶(system) ×53815 | - | 1 465 252 22 % | 2 443 204 37 % |
| Snapshot 6.5 MB | ▶(closure) ×23683 | - | 661 440 10 % | 1 948 248 30 % |
| | ▶(array) ×3015 | - | 965 976 15 % | 1 239 540 19 % |
| | ▶Window ×29 | - | 824 0 % | 1 086 764 17 % |
| | ▶(compiled code) ×5810 | - | 366 604 6 % | 1 082 008 17 % |
| | ▶Detached IntersectionObserver ×8583 | 19 | 240 324 4 % | 790 008 12 % |
| | ▶system / Context ×10555 | - | 255 800 4 % | 752 928 12 % |
| | ▶Detached HTMLSlotElement ×25904 | - | 687 904 11 % | 687 904 11 % |
| | ▶Detached IntersectionObserverDelegate ×8583 | 20 | 0 0 % | 549 684 8 % |
| | ▶Detached V8IntersectionObserverCallback ×8... | 21 | 0 0 % | 549 684 8 % |
| | ▶(string) ×5369 | - | 474 092 7 % | 474 092 7 % |
| | ▶Generator ×77 | - | 3 380 0 % | 410 996 6 % |
| | ▶Window / http://localhost:8080 | 1 | 36 0 % | 338 088 5 % |
| | ▶Detached HTMLDivElement ×32402 | - | 270 732 4 % | 312 036 5 % |

Memory Panel

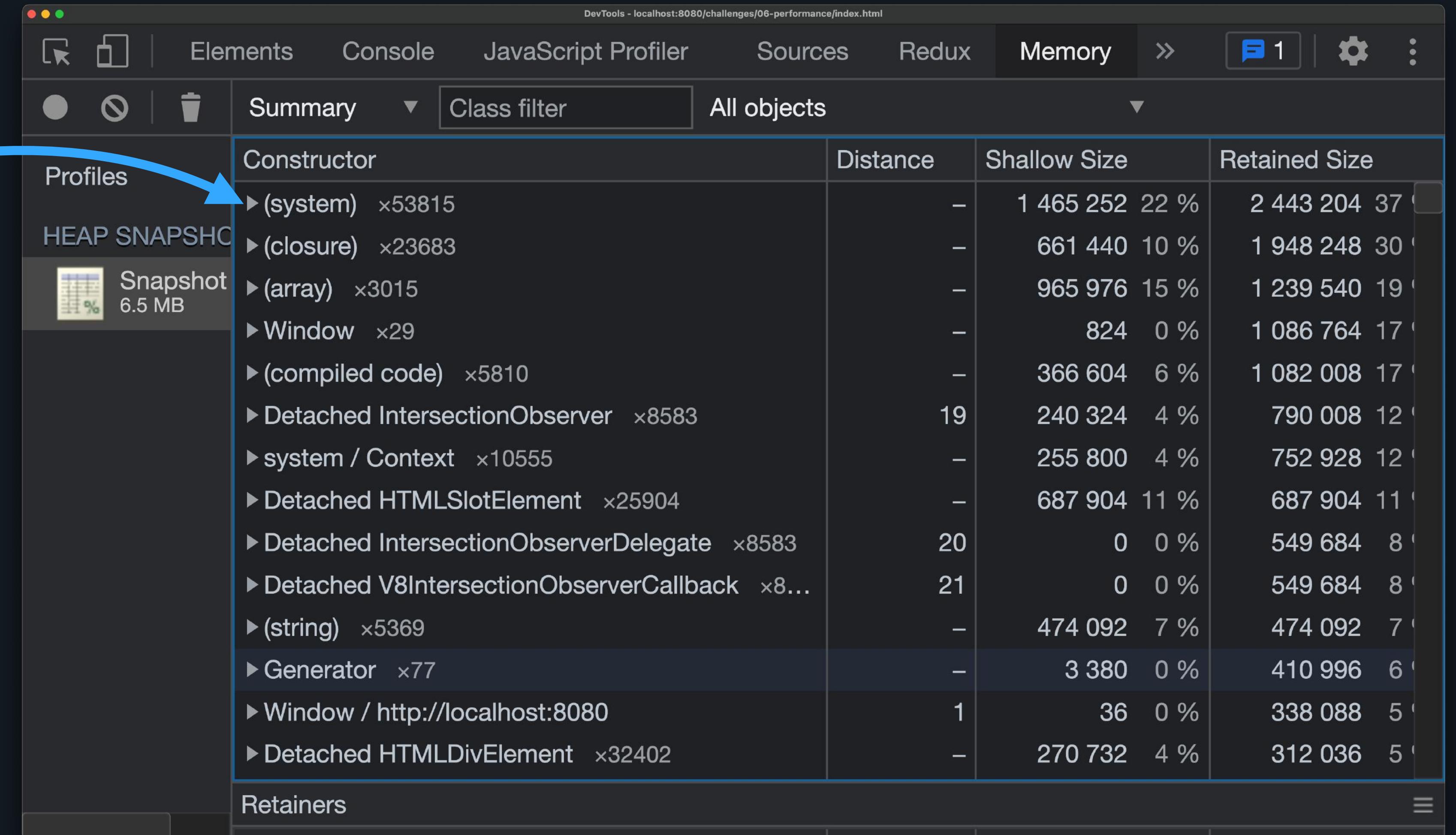
Maximum retained size
of Objects



| | Constructor | Distance | Shallow Size | Retained Size |
|-----------------|--|----------|----------------|----------------|
| Profiles | ▶(system) ×53815 | - | 1 465 252 22 % | 2 443 204 37 % |
| HEAP SNAPSHOT | ▶(closure) ×23683 | - | 661 440 10 % | 1 948 248 30 % |
| Snapshot 6.5 MB | ▶(array) ×3015 | - | 965 976 15 % | 1 239 540 19 % |
| | ▶Window ×29 | - | 824 0 % | 1 086 764 17 % |
| | ▶(compiled code) ×5810 | - | 366 604 6 % | 1 082 008 17 % |
| | ▶Detached IntersectionObserver ×8583 | 19 | 240 324 4 % | 790 008 12 % |
| | ▶system / Context ×10555 | - | 255 800 4 % | 752 928 12 % |
| | ▶Detached HTMLSlotElement ×25904 | - | 687 904 11 % | 687 904 11 % |
| | ▶Detached IntersectionObserverDelegate ×8583 | 20 | 0 0 % | 549 684 8 % |
| | ▶Detached V8IntersectionObserverCallback ×8... | 21 | 0 0 % | 549 684 8 % |
| | ▶(string) ×5369 | - | 474 092 7 % | 474 092 7 % |
| | ▶Generator ×77 | - | 3 380 0 % | 410 996 6 % |
| | ▶Window / http://localhost:8080 | 1 | 36 0 % | 338 088 5 % |
| | ▶Detached HTMLDivElement ×32402 | - | 270 732 4 % | 312 036 5 % |

Memory Panel

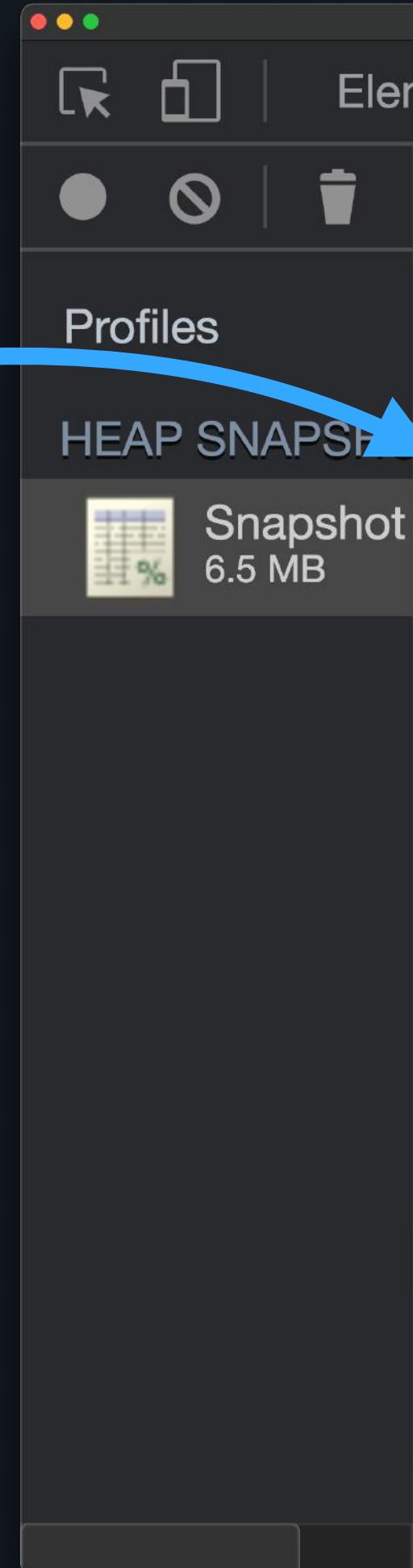
Intermediate objects between root (window or globalThis) and an object referenced by it.



| Constructor | Distance | Shallow Size | Retained Size |
|--|----------|----------------|----------------|
| ▶(system) ×53815 | - | 1 465 252 22 % | 2 443 204 37 % |
| ▶(closure) ×23683 | - | 661 440 10 % | 1 948 248 30 % |
| ▶(array) ×3015 | - | 965 976 15 % | 1 239 540 19 % |
| ▶Window ×29 | - | 824 0 % | 1 086 764 17 % |
| ▶(compiled code) ×5810 | - | 366 604 6 % | 1 082 008 17 % |
| ▶Detached IntersectionObserver ×8583 | 19 | 240 324 4 % | 790 008 12 % |
| ▶system / Context ×10555 | - | 255 800 4 % | 752 928 12 % |
| ▶Detached HTMLSlotElement ×25904 | - | 687 904 11 % | 687 904 11 % |
| ▶Detached IntersectionObserverDelegate ×8583 | 20 | 0 0 % | 549 684 8 % |
| ▶Detached V8IntersectionObserverCallback ×8... | 21 | 0 0 % | 549 684 8 % |
| ▶(string) ×5369 | - | 474 092 7 % | 474 092 7 % |
| ▶Generator ×77 | - | 3 380 0 % | 410 996 6 % |
| ▶Window / http://localhost:8080 | 1 | 36 0 % | 338 088 5 % |
| ▶Detached HTMLDivElement ×32402 | - | 270 732 4 % | 312 036 5 % |

Memory Panel

References to objects through function closures

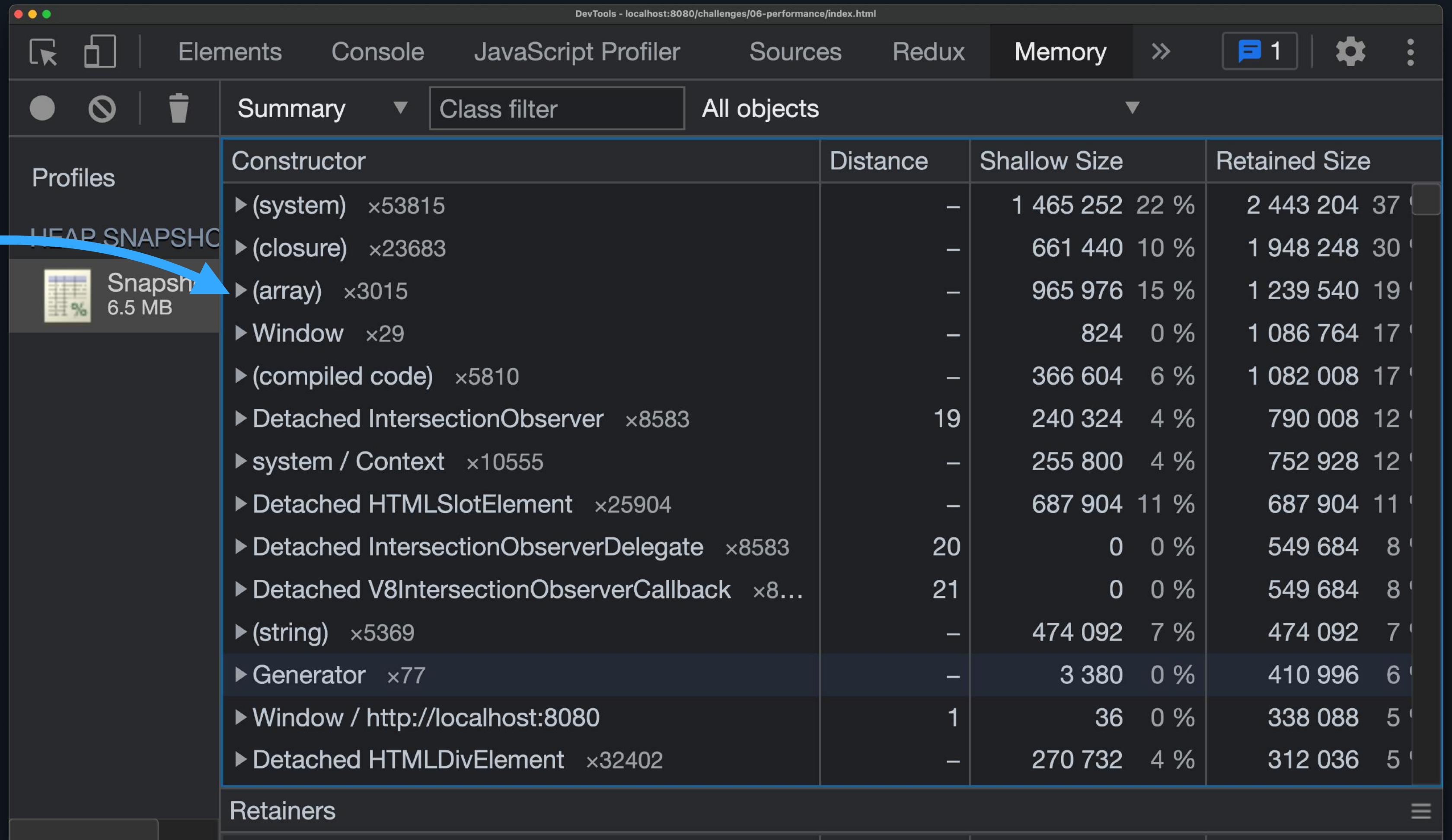


The screenshot shows the Chrome DevTools Memory Panel. The title bar reads "DevTools - localhost:8080/challenges/06-performance/index.html". The tabs at the top are Elements, Console, JavaScript Profiler, Sources, Redux, Memory, and more. The Memory tab is selected. In the sidebar, there are three tabs: Profiles, HEAP SNAPSHOT (which is selected), and Snapshot (6.5 MB). The main area displays a table titled "All objects" with columns: Constructor, Distance, Shallow Size, and Retained Size. The table lists various objects and their memory usage. A blue arrow points from the text "References to objects through function closures" to the "HEAP SNAPSHOT" tab in the sidebar.

| Constructor | Distance | Shallow Size | Retained Size |
|--|----------|----------------|----------------|
| ▶(system) ×53815 | - | 1 465 252 22 % | 2 443 204 37 % |
| ▶(closure) ×23683 | - | 661 440 10 % | 1 948 248 30 % |
| ▶(array) ×3015 | - | 965 976 15 % | 1 239 540 19 % |
| ▶Window ×29 | - | 824 0 % | 1 086 764 17 % |
| ▶(compiled code) ×5810 | - | 366 604 6 % | 1 082 008 17 % |
| ▶Detached IntersectionObserver ×8583 | 19 | 240 324 4 % | 790 008 12 % |
| ▶system / Context ×10555 | - | 255 800 4 % | 752 928 12 % |
| ▶Detached HTMLSlotElement ×25904 | - | 687 904 11 % | 687 904 11 % |
| ▶Detached IntersectionObserverDelegate ×8583 | 20 | 0 0 % | 549 684 8 % |
| ▶Detached V8IntersectionObserverCallback ×8... | 21 | 0 0 % | 549 684 8 % |
| ▶(string) ×5369 | - | 474 092 7 % | 474 092 7 % |
| ▶Generator ×77 | - | 3 380 0 % | 410 996 6 % |
| ▶Window / http://localhost:8080 | 1 | 36 0 % | 338 088 5 % |
| ▶Detached HTMLDivElement ×32402 | - | 270 732 4 % | 312 036 5 % |

Memory Panel

Objects with properties
referencing an Array

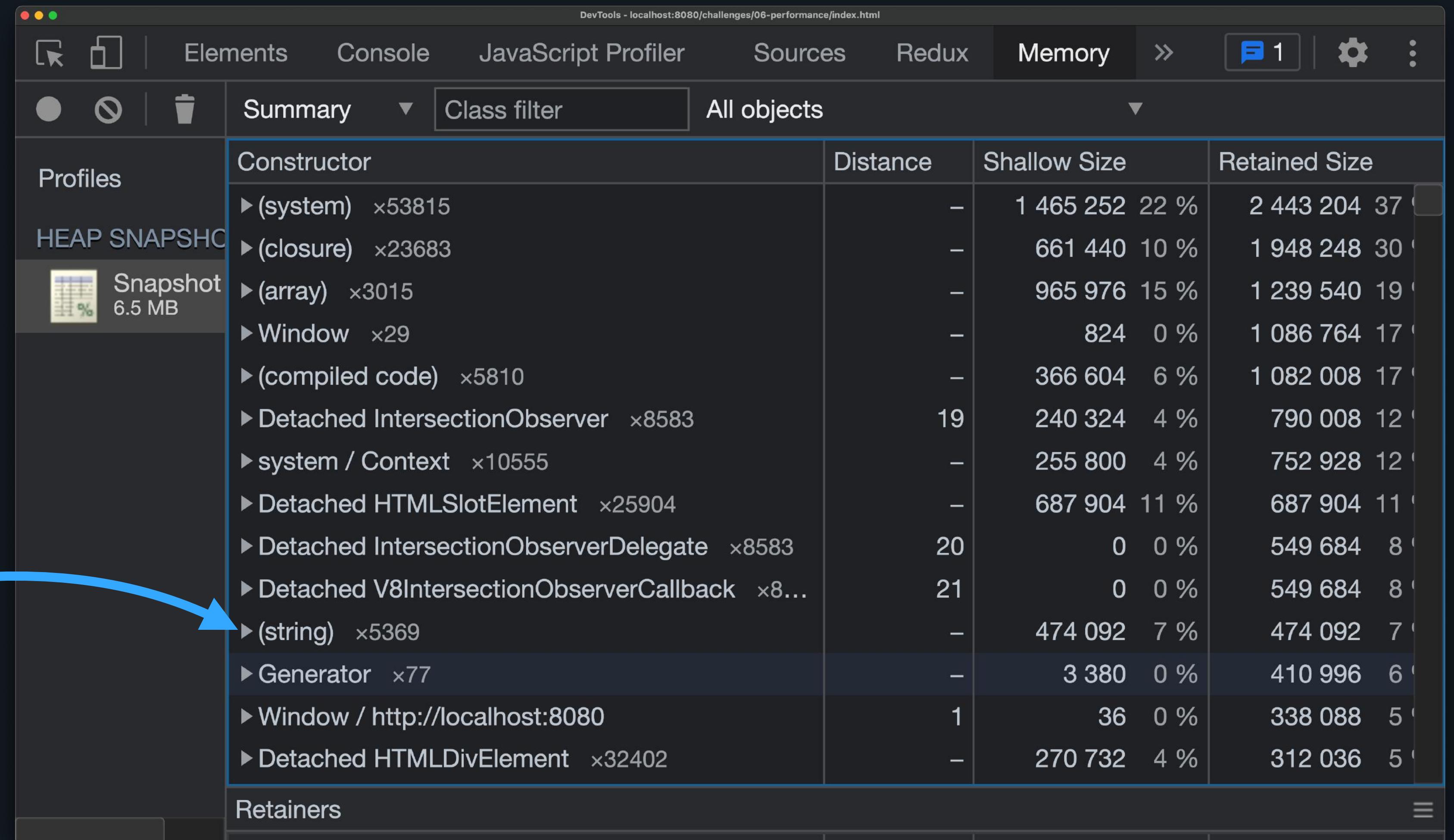


The screenshot shows the Chrome DevTools Memory Panel. The title bar reads "DevTools - localhost:8080/challenges/06-performance/index.html". The tabs are Elements, Console, JavaScript Profiler, Sources, Redux, Memory, and more. The Memory tab is active. In the left sidebar, there are buttons for Profiles, HEAP SNAPSHOT (which is selected), and Snapshot (6.5 MB). The main area is titled "All objects" and shows a table with columns: Constructor, Distance, Shallow Size, and Retained Size. The table lists various objects and their memory usage. A blue arrow points from the text "Objects with properties referencing an Array" to the "Snapshot" button in the sidebar.

| Constructor | Distance | Shallow Size | Retained Size |
|---|----------|----------------|----------------|
| ▶ (system) ×53815 | - | 1 465 252 22 % | 2 443 204 37 % |
| ▶ (closure) ×23683 | - | 661 440 10 % | 1 948 248 30 % |
| ▶ (array) ×3015 | - | 965 976 15 % | 1 239 540 19 % |
| ▶ Window ×29 | - | 824 0 % | 1 086 764 17 % |
| ▶ (compiled code) ×5810 | - | 366 604 6 % | 1 082 008 17 % |
| ▶ Detached IntersectionObserver ×8583 | 19 | 240 324 4 % | 790 008 12 % |
| ▶ system / Context ×10555 | - | 255 800 4 % | 752 928 12 % |
| ▶ Detached HTMLSlotElement ×25904 | - | 687 904 11 % | 687 904 11 % |
| ▶ Detached IntersectionObserverDelegate ×8583 | 20 | 0 0 % | 549 684 8 % |
| ▶ Detached V8IntersectionObserverCallback ×8... | 21 | 0 0 % | 549 684 8 % |
| ▶ (string) ×5369 | - | 474 092 7 % | 474 092 7 % |
| ▶ Generator ×77 | - | 3 380 0 % | 410 996 6 % |
| ▶ Window / http://localhost:8080 | 1 | 36 0 % | 338 088 5 % |
| ▶ Detached HTMLDivElement ×32402 | - | 270 732 4 % | 312 036 5 % |

Memory Panel

Objects with properties referencing an String



The screenshot shows the Chrome DevTools Memory Panel with a heap snapshot named 'Snapshot 6.5 MB'. The table lists objects by their constructor, shallow size, and retained size. A blue arrow points from the text 'Objects with properties referencing an String' to the '(string)' entry in the list.

| Profiles | Constructor | Distance | Shallow Size | Retained Size |
|-----------------|--|----------|----------------|----------------|
| HEAP SNAPSHOT | ▶(system) ×53815 | - | 1 465 252 22 % | 2 443 204 37 % |
| Snapshot 6.5 MB | ▶(closure) ×23683 | - | 661 440 10 % | 1 948 248 30 % |
| | ▶(array) ×3015 | - | 965 976 15 % | 1 239 540 19 % |
| | ▶Window ×29 | - | 824 0 % | 1 086 764 17 % |
| | ▶(compiled code) ×5810 | - | 366 604 6 % | 1 082 008 17 % |
| | ▶Detached IntersectionObserver ×8583 | 19 | 240 324 4 % | 790 008 12 % |
| | ▶system / Context ×10555 | - | 255 800 4 % | 752 928 12 % |
| | ▶Detached HTMLSlotElement ×25904 | - | 687 904 11 % | 687 904 11 % |
| | ▶Detached IntersectionObserverDelegate ×8583 | 20 | 0 0 % | 549 684 8 % |
| | ▶Detached V8IntersectionObserverCallback ×8... | 21 | 0 0 % | 549 684 8 % |
| | ▶(string) ×5369 | - | 474 092 7 % | 474 092 7 % |
| | ▶Generator ×77 | - | 3 380 0 % | 410 996 6 % |
| | ▶Window / http://localhost:8080 | 1 | 36 0 % | 338 088 5 % |
| | ▶Detached HTMLDivElement ×32402 | - | 270 732 4 % | 312 036 5 % |

Memory Panel

Everything related to compiled code (your app)

The screenshot shows the Chrome DevTools Memory Panel. The title bar reads "DevTools - localhost:8080/challenges/06-performance/index.html". The tabs at the top are Elements, Console, JavaScript Profiler, Sources, Redux, Memory, and more. The Memory tab is selected. Below the tabs, there are buttons for Profiles, HEAP SNAPSHOT, and Snapshot (6.5 MB). A blue arrow points from the text "Everything related to compiled code (your app)" to the Snapshot button. The main area is a table titled "All objects" with columns: Constructor, Distance, Shallow Size, and Retained Size. The table lists various objects and their memory usage. Notable entries include "(system)" with 53815 instances and "(closure)" with 23683 instances.

| Constructor | Distance | Shallow Size | Retained Size |
|--|----------|----------------|----------------|
| ▶(system) ×53815 | - | 1 465 252 22 % | 2 443 204 37 % |
| ▶(closure) ×23683 | - | 661 440 10 % | 1 948 248 30 % |
| ▶(array) ×3015 | - | 965 976 15 % | 1 239 540 19 % |
| ▶Window ×29 | - | 824 0 % | 1 086 764 17 % |
| ▶(compiled code) ×5810 | - | 366 604 6 % | 1 082 008 17 % |
| ▶Detached IntersectionObserver ×8583 | 19 | 240 324 4 % | 790 008 12 % |
| ▶system / Context ×10555 | - | 255 800 4 % | 752 928 12 % |
| ▶Detached HTMLSlotElement ×25904 | - | 687 904 11 % | 687 904 11 % |
| ▶Detached IntersectionObserverDelegate ×8583 | 20 | 0 0 % | 549 684 8 % |
| ▶Detached V8IntersectionObserverCallback ×8... | 21 | 0 0 % | 549 684 8 % |
| ▶(string) ×5369 | - | 474 092 7 % | 474 092 7 % |
| ▶Generator ×77 | - | 3 380 0 % | 410 996 6 % |
| ▶Window / http://localhost:8080 | 1 | 36 0 % | 338 088 5 % |
| ▶Detached HTMLDivElement ×32402 | - | 270 732 4 % | 312 036 5 % |

Memory Panel

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



DevTools - localhost:8080/challenges/06-performance/index.html

Elements Console JavaScript Profiler Sources Redux Memory » ⚑ 1 ⚒ ⚓ :

| Profiles | Constructor | Distance | Shallow Size | Retained Size |
|----------------------------------|------------------------------------|----------|--------------|---------------|
| HEAP SNAPSHOT Snapshot 5.4 MB | ▶ Detached HTMLMetaElement ×8 | - | 0 0 % | 0 0 % |
| | ▶ Detached HTMLParagraphElement ×6 | - | 0 0 % | 0 0 % |
| | ▶ Detached HTMLPreElement ×3 | - | 0 0 % | 0 0 % |
| | ▶ Detached HTMLScriptElement ×10 | - | 0 0 % | 0 0 % |
| | ▶ Detached HTMLSpanElement ×16 | - | 0 0 % | 0 0 % |
| | ▶ Detached HTMLTitleElement ×2 | - | 0 0 % | 0 0 % |
| | ▶ HTMLAllCollection | 7 | 0 0 % | 0 0 % |
| | ▶ HTMLAnchorElement | 9 | 0 0 % | 0 0 % |
| | ▶ HTMLCollection | 7 | 0 0 % | 0 0 % |
| | ▶ HTMLDLListElement ×21 | - | 0 0 % | 0 0 % |
| | ▶ HTMLHeadingElement ×4 | - | 0 0 % | 0 0 % |
| | ▶ HTMLMetaElement ×4 | 6 | 0 0 % | 0 0 % |
| ▶ HTMLSpanElement ×2 | 10 | 0 0 % | 0 0 % | |
| ▶ HTMLTitleElement | 4 | 0 0 % | 0 0 % | |
| Retainers | | | | |

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

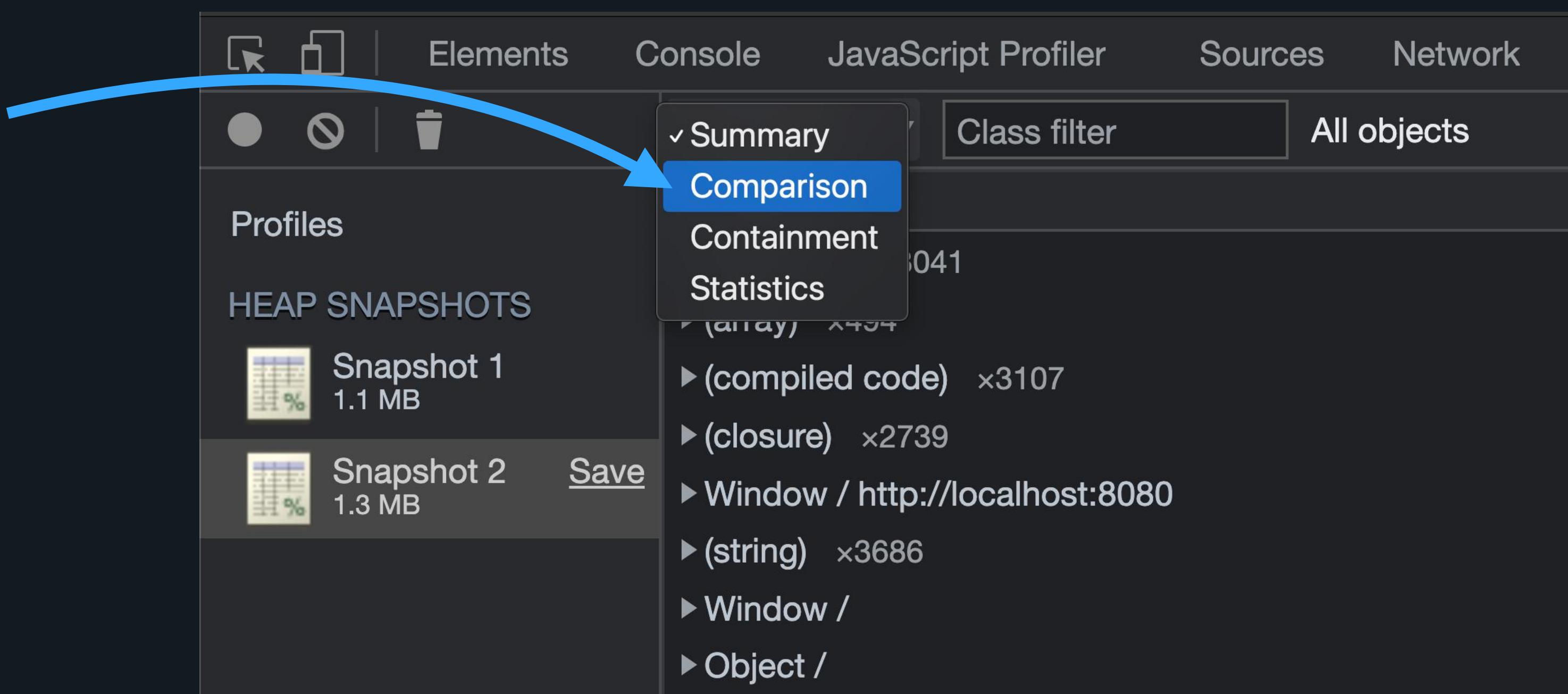
-  Clear console
-  Remove all breakpoints
-  Manually perform GC

Finding Memory Leaks

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

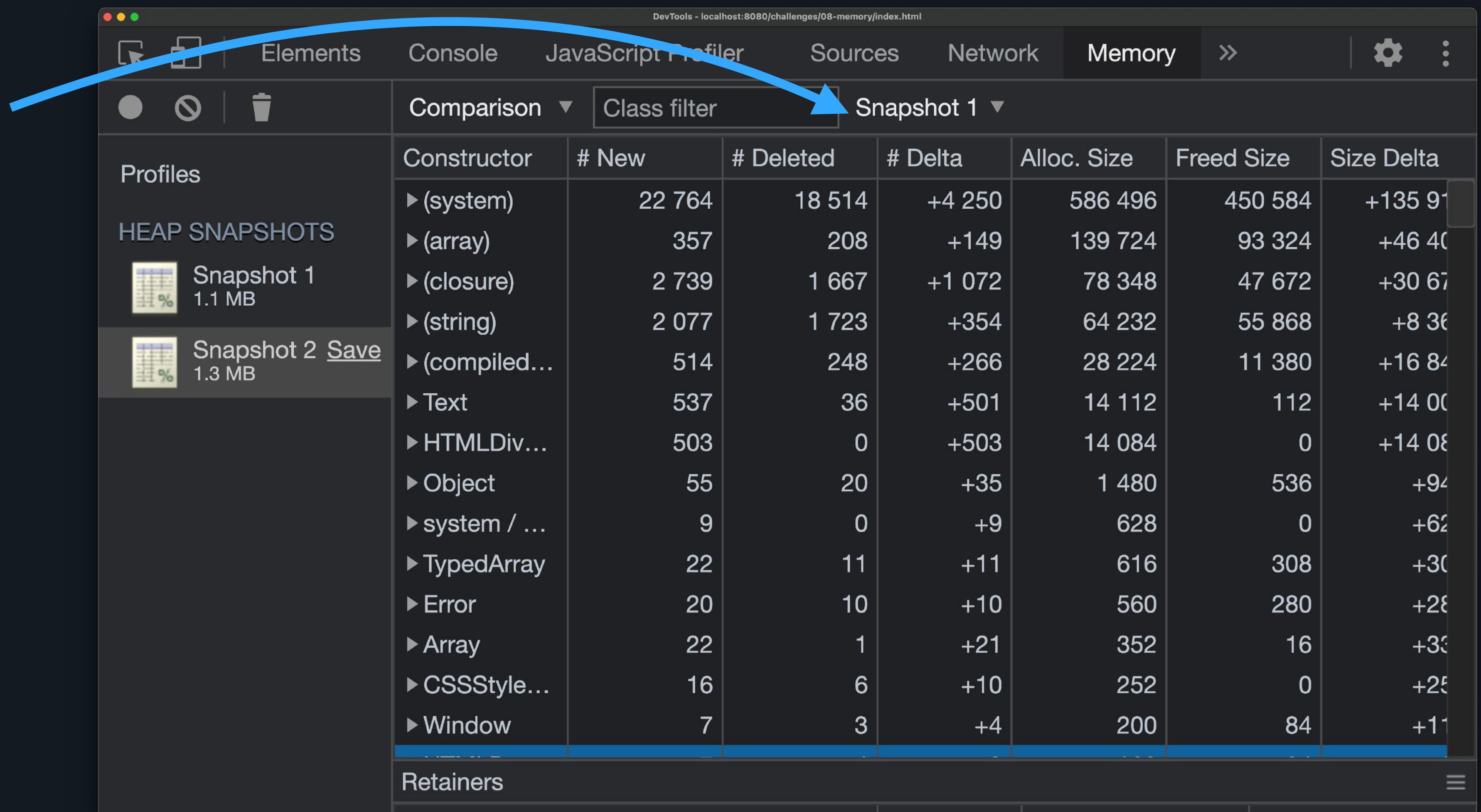
Snapshot Comparison

Switch to the comparison view



Snapshot Comparison

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



Snapshot Comparison

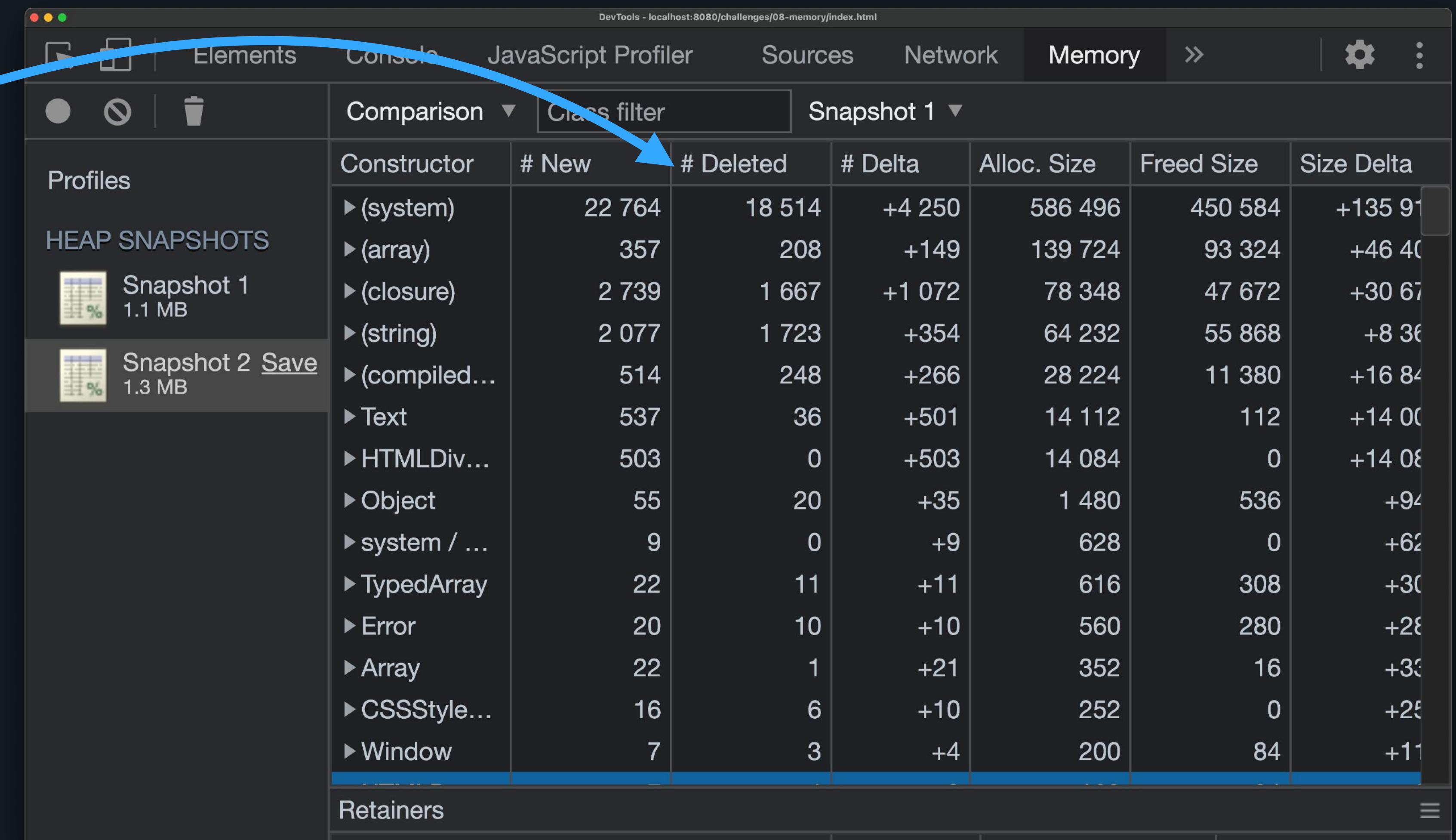
The number of new objects

The screenshot shows the Chrome DevTools Memory tab with a comparison between two heap snapshots. The left sidebar lists 'HEAP SNAPSHOTS' with 'Snapshot 1' (1.1 MB) and 'Snapshot 2' (1.3 MB, highlighted). The main table compares the differences between these two snapshots. A blue arrow points to the 'Constructor' column header.

| Profiles | Constructor | # New | # Deleted | # Delta | Alloc. Size | Freed Size | Size Delta |
|----------------|-----------------|--------|-----------|---------|-------------|------------|------------|
| HEAP SNAPSHOTS | ► (system) | 22 764 | 18 514 | +4 250 | 586 496 | 450 584 | +135 91 |
| | ► (array) | 357 | 208 | +149 | 139 724 | 93 324 | +46 40 |
| | ► (closure) | 2 739 | 1 667 | +1 072 | 78 348 | 47 672 | +30 67 |
| | ► (string) | 2 077 | 1 723 | +354 | 64 232 | 55 868 | +8 36 |
| | ► (compiled...) | 514 | 248 | +266 | 28 224 | 11 380 | +16 84 |
| | ► Text | 537 | 36 | +501 | 14 112 | 112 | +14 00 |
| | ► HTMLDiv... | 503 | 0 | +503 | 14 084 | 0 | +14 08 |
| | ► Object | 55 | 20 | +35 | 1 480 | 536 | +94 |
| | ► system / ... | 9 | 0 | +9 | 628 | 0 | +62 |
| | ► TypedArray | 22 | 11 | +11 | 616 | 308 | +30 |
| | ► Error | 20 | 10 | +10 | 560 | 280 | +28 |
| | ► Array | 22 | 1 | +21 | 352 | 16 | +33 |
| | ► CSSStyle... | 16 | 6 | +10 | 252 | 0 | +25 |
| | ► Window | 7 | 3 | +4 | 200 | 84 | +11 |

Snapshot Comparison

The number of deleted objects



The screenshot shows the Chrome DevTools Memory tab with a comparison between two heap snapshots. The left sidebar lists 'Profiles' and 'HEAP SNAPSHOTS'. Under 'HEAP SNAPSHOTS', 'Snapshot 1' (1.1 MB) is listed above 'Snapshot 2' (1.3 MB, which is currently selected). The main table compares the differences between these two snapshots. A blue arrow points to the '# Deleted' column header.

| Constructor | # New | # Deleted | # Delta | Alloc. Size | Freed Size | Size Delta |
|-----------------|--------|-----------|---------|-------------|------------|------------|
| ► (system) | 22 764 | 18 514 | +4 250 | 586 496 | 450 584 | +135 91 |
| ► (array) | 357 | 208 | +149 | 139 724 | 93 324 | +46 40 |
| ► (closure) | 2 739 | 1 667 | +1 072 | 78 348 | 47 672 | +30 67 |
| ► (string) | 2 077 | 1 723 | +354 | 64 232 | 55 868 | +8 36 |
| ► (compiled...) | 514 | 248 | +266 | 28 224 | 11 380 | +16 84 |
| ► Text | 537 | 36 | +501 | 14 112 | 112 | +14 00 |
| ► HTMLDiv... | 503 | 0 | +503 | 14 084 | 0 | +14 08 |
| ► Object | 55 | 20 | +35 | 1 480 | 536 | +94 |
| ► system / ... | 9 | 0 | +9 | 628 | 0 | +62 |
| ► TypedArray | 22 | 11 | +11 | 616 | 308 | +30 |
| ► Error | 20 | 10 | +10 | 560 | 280 | +28 |
| ► Array | 22 | 1 | +21 | 352 | 16 | +33 |
| ► CSSStyle... | 16 | 6 | +10 | 252 | 0 | +25 |
| ► Window | 7 | 3 | +4 | 200 | 84 | +11 |

Snapshot Comparison

The difference in the
number of objects
between snapshots

The screenshot shows the Chrome DevTools JavaScript Profiler tab with a comparison between two heap snapshots. The left sidebar lists 'Profiles' and 'HEAP SNAPSHOTS'. Under 'HEAP SNAPSHOTS', 'Snapshot 1' (1.1 MB) is selected, and 'Snapshot 2' (1.3 MB) is shown below it with a 'Save' button. The main area displays a table with columns: Constructor, # New, # Deleted, # Delta, Alloc. Size, Freed Size, and Size Delta. A blue arrow points to the '# Delta' column header. The table data is as follows:

| | Constructor | # New | # Deleted | # Delta | Alloc. Size | Freed Size | Size Delta |
|----------------|----------------|--------|-----------|---------|-------------|------------|------------|
| HEAP SNAPSHOTS | ►(system) | 22 764 | 18 514 | +4 250 | 586 496 | 450 584 | +135 91 |
| | ►(array) | 357 | 208 | +149 | 139 724 | 93 324 | +46 40 |
| | ►(closure) | 2 739 | 1 667 | +1 072 | 78 348 | 47 672 | +30 67 |
| | ►(string) | 2 077 | 1 723 | +354 | 64 232 | 55 868 | +8 36 |
| | ►(compiled...) | 514 | 248 | +266 | 28 224 | 11 380 | +16 84 |
| | ►Text | 537 | 36 | +501 | 14 112 | 112 | +14 00 |
| | ►HTMLDiv... | 503 | 0 | +503 | 14 084 | 0 | +14 08 |
| | ►Object | 55 | 20 | +35 | 1 480 | 536 | +94 |
| | ►system / ... | 9 | 0 | +9 | 628 | 0 | +62 |
| | ►TypedArray | 22 | 11 | +11 | 616 | 308 | +30 |
| | ►Error | 20 | 10 | +10 | 560 | 280 | +28 |
| | ►Array | 22 | 1 | +21 | 352 | 16 | +33 |
| ►CSSStyle... | 16 | 6 | +10 | 252 | 0 | +25 | |
| ►Window | 7 | 3 | +4 | 200 | 84 | +11 | |

At the bottom, there is a 'Retainers' section.

Snapshot Comparison

Additional memory allocated between snapshots

DevTools - localhost:8080/challenges/08-memory/index.html

Elements Console JavaScript Profiler Sources Network Memory > ⚙️ ⋮

Comparison ▾ Class filter Snapshot 1 ▾

| Profiles | Constructor | # New | # Deleted | # Delta | Alloc. Size | Freed Size | Size Delta |
|---------------------------|------------------|--------|-----------|------------|-------------|------------|------------|
| HEAP SNAPSHOTS | ▶ (system) | 22 764 | 18 514 | +4 250 | 586 496 | 450 584 | +135 91 |
| | ▶ (array) | 357 | 208 | +149 | 139 724 | 93 324 | +46 40 |
| | ▶ (closure) | 2 739 | 1 867 | +1 072 | 78 348 | 47 672 | +30 67 |
| | ▶ (string) | 2 077 | 1 723 | +354 | 64 232 | 55 868 | +8 36 |
| | ▶ (compiled...) | 514 | 248 | +266 | 28 224 | 11 380 | +16 84 |
| | ▶ Text | 537 | 36 | +501 | 14 112 | 112 | +14 00 |
| | ▶ HTMLDivElement | 503 | 0 | +503 | 14 084 | 0 | +14 08 |
| | ▶ Object | 55 | 20 | +35 | 1 480 | 536 | +94 |
| | ▶ system / ... | 9 | 0 | +9 | 628 | 0 | +62 |
| | ▶ TypedArray | 22 | 11 | +11 | 616 | 308 | +30 |
| | ▶ Error | 20 | 10 | +10 | 560 | 280 | +28 |
| | ▶ Array | 22 | 1 | +21 | 352 | 16 | +33 |
| ▶ CSSStyle... ▶ Window | 16 7 | 6 3 | +10 +4 | 252 200 | 0 84 | +25 +11 | |

Retainers

Snapshot Comparison

Memory freed up
between snapshots

DevTools - localhost:8080/challenges/08-memory/index.html

Elements Console JavaScript Profiler Sources Network Memory >  

Comparison ▾ Class filter Snapshot 1 ▾

| Profiles | Constructor | # New | # Deleted | # Delta | Alloc. Size | Freed Size | Size Delta | |
|----------------|----------------------------------|-----------------|-----------|---------|-------------|------------|------------|--------|
| HEAP SNAPSHOTS | ▶ (system) | 22 764 | 18 514 | +4 250 | 586 496 | 450 584 | +135 91 | |
| | ▶ (array) | 357 | 208 | +149 | 139 724 | 93 324 | +46 40 | |
| | Snapshot 1 1.1 MB | ▶ (closure) | 2 739 | 1 667 | +1 072 | 78 348 | 47 672 | +30 67 |
| | Snapshot 2 <u>Save</u> 1.3 MB | ▶ (string) | 2 077 | 1 723 | +354 | 64 232 | 55 868 | +8 36 |
| | | ▶ (compiled...) | 514 | 248 | +266 | 28 224 | 11 380 | +16 84 |
| | | ▶ Text | 537 | 36 | +501 | 14 112 | 112 | +14 00 |
| | | ▶ HTMLDiv... | 503 | 0 | +503 | 14 084 | 0 | +14 08 |
| | | ▶ Object | 55 | 20 | +35 | 1 480 | 536 | +94 |
| | | ▶ system / ... | 9 | 0 | +9 | 628 | 0 | +62 |
| | | ▶ TypedArray | 22 | 11 | +11 | 616 | 308 | +30 |
| | | ▶ Error | 20 | 10 | +10 | 560 | 280 | +28 |
| | | ▶ Array | 22 | 1 | +21 | 352 | 16 | +33 |
| | ▶ CSSStyle... | 16 | 6 | +10 | 252 | 0 | +25 | |
| | ▶ Window | 7 | 3 | +4 | 200 | 84 | +11 | |
| | Retainers | | | | | | | |

Snapshot Comparison

The size difference between snapshots

DevTools - localhost:8080/challenges/08-memory/index.html

Elements Console JavaScript Profiler Sources Network Memory > ⚙️ ⋮

Comparison ▾ Class filter Snapshot 1 ▾

| Profiles | Constructor | # New | # Deleted | # Delta | Alloc. Size | Freed Size | Size Delta | |
|----------------|----------------------------------|-------------|-----------|---------|-------------|------------|------------|--------|
| HEAP SNAPSHOTS | ▶ (system) | 22 764 | 18 514 | +4 250 | 586 496 | 450 534 | +135 91 | |
| | ▶ (array) | 357 | 208 | +149 | 139 724 | 93 324 | +46 40 | |
| | Snapshot 1 1.1 MB | ▶ (closure) | 2 739 | 1 667 | +1 072 | 78 348 | 47 672 | +30 67 |
| | ▶ (string) | 2 077 | 1 723 | +354 | 54 232 | 55 868 | +8 36 | |
| | ▶ (compiled...) | 514 | 248 | +266 | 28 224 | 11 380 | +16 84 | |
| | Snapshot 2 <u>Save</u> 1.3 MB | ▶ Text | 537 | 0 | +501 | 14 112 | 112 | +14 00 |
| | ▶ HTMLDiv... | 503 | 0 | +503 | 14 084 | 0 | +14 08 | |
| | ▶ Object | 55 | 20 | +35 | 1 480 | 536 | +94 | |
| | ▶ system / ... | 9 | 0 | +9 | 628 | 0 | +62 | |
| | ▶ TypedArray | 22 | 11 | +11 | 616 | 308 | +30 | |
| ▶ Error | 20 | 10 | +10 | 560 | 280 | +28 | | |
| ▶ Array | 22 | 1 | +21 | 352 | 16 | +33 | | |
| ▶ CSSStyle... | 16 | 6 | +10 | 252 | 0 | +25 | | |
| ▶ Window | 7 | 3 | +4 | 200 | 84 | +11 | | |
| Retainers | | | | | | | | |

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

The screenshot shows the Chrome DevTools Memory tab interface. At the top, there are tabs for Elements, Console, JavaScript Profiler, Sources, Network, Performance, Memory (which is selected), Application, and more. Below the tabs, there's a toolbar with icons for selection, copy, and delete, followed by 'Comparison' and 'Class filter' dropdowns, and a 'Snapshot 1' dropdown.

The main area is divided into sections: 'Profiles' (listing 'Constructor'), 'HEAP SNAPSHOTS' (listing 'Snapshot 1' and 'Snapshot 2'), and 'Retainers' (listing 'Object').

The 'Profiles' section contains a table with columns: Constructor, # New, # Deleted, # Delta, Alloc. Size, Freed Size, and Size Delta. The table lists various constructor types with their respective statistics. The row for 'HTMLDivElement' is highlighted with a blue background.

| Constructor | # New | # Deleted | # Delta | Alloc. Size | Freed Size | Size Delta |
|-------------------|-------|-----------|---------|-------------|------------|------------|
| ▶ (system) | 9 750 | 9 674 | +76 | 188 892 | 184 952 | +3 940 |
| ▶ (closure) | 2 723 | 2 714 | +9 | 77 852 | 77 584 | +268 |
| ▶ (string) | 1 250 | 1 252 | -2 | 39 148 | 39 208 | -60 |
| ▶ (array) | 89 | 110 | -21 | 35 716 | 40 596 | -4 880 |
| ▶ (compiled code) | 534 | 531 | +3 | 29 800 | 28 936 | +864 |
| ▶ Text | 537 | 37 | +500 | 14 112 | 112 | +14 000 |
| ▶ HTMLDivElement | 503 | 0 | +503 | 14 084 | 0 | +14 084 |
| ▶ PointerEvent | 3 | 0 | +3 | 84 | 0 | +84 |
| ▶ MouseEvent | 1 | 0 | +1 | 28 | 0 | +28 |
| ▶ UIEvent | 1 | 0 | +1 | 28 | 0 | +28 |
| ▶ CSSFontFaceRule | 20 | 20 | 0 | 0 | 0 | 0 |
| ▶ CSSImportRule | 4 | 4 | 0 | 0 | 0 | 0 |

The 'Retainers' section has a table with columns: Object, Distance, Shallow Size, and Retained Size. It currently displays a single row for 'Object'.

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

The screenshot shows the Chrome DevTools Memory tab interface. On the left, there's a sidebar with 'Profiles' and 'HEAP SNAPSHOTS'. Under 'HEAP SNAPSHOTS', 'Snapshot 1' (1.3 MB) is selected, while 'Snapshot 2' (5.6 MB) is shown as a thumbnail.

The main area displays two tables: 'Constructor' and 'Retainers'.

Constructor Table:

| Constructor | # New | # Deleted | # Delta | Alloc. Size | Freed Size | Size Delta |
|-----------------------------------|---------|-----------|----------|-------------|------------|------------|
| Detached HTMLDivElement | 100 000 | 0 | +100 000 | 2 800 000 | 0 | +2 800 000 |
| ▶ Detached HTMLDivElement @290125 | . | . | . | 28 | . | . |
| ▶ Detached HTMLDivElement @290129 | . | . | . | 28 | . | . |
| ▶ Detached HTMLDivElement @290133 | . | . | . | 28 | . | . |
| ▶ Detached HTMLDivElement @290137 | . | . | . | 28 | . | . |
| ▶ Detached HTMLDivElement @290141 | . | . | . | 28 | . | . |
| ▶ Detached HTMLDivElement @290145 | . | . | . | 28 | . | . |
| ▶ Detached HTMLDivElement @290149 | . | . | . | 28 | . | . |
| ▶ Detached HTMLDivElement @290153 | . | . | . | 28 | . | . |
| ▶ Detached HTMLDivElement @290157 | . | . | . | 28 | . | . |
| ▶ Detached HTMLDivElement @290161 | . | . | . | 28 | . | . |
| ▶ Detached HTMLDivElement @290165 | . | . | . | 28 | . | . |

Retainers Table:

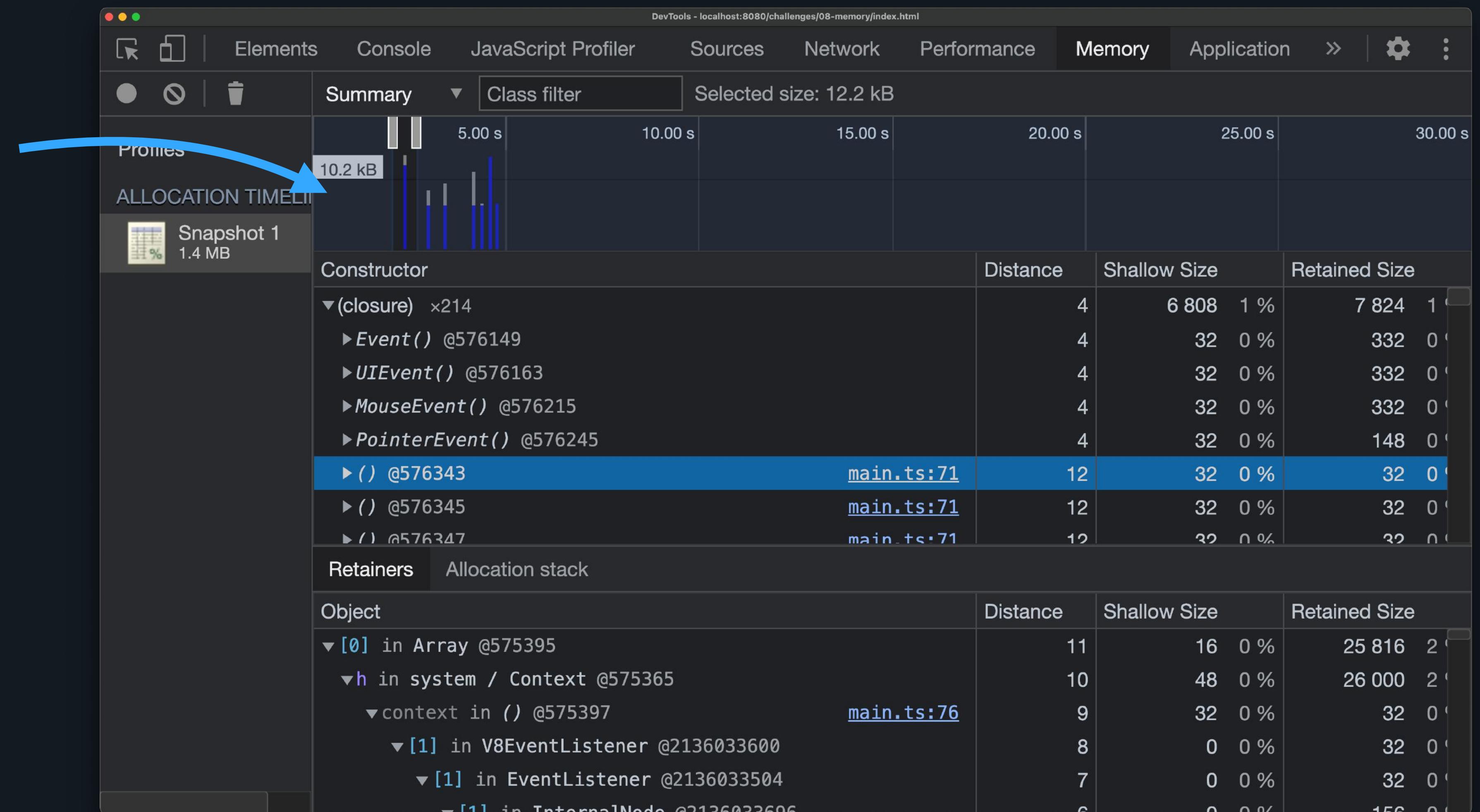
| Object | Distance | Shallow Size | Retained Size |
|--------------------------------------|------------------------|--------------|---------------|
| ▶ [49914] in Array @283069 | (anonymous) main.ts:27 | 16 0 % | 2 256 208 41 |
| ▶ a in system / Context @283061 | anonymous() | 48 0 % | 2 256 392 41 |
| ▶ context in () @279903 | main.ts:27 | 9 0 % | 28 0 % |
| ▶ [1] in V8EventListener @3127067744 | | 8 0 % | 28 0 % |
| ▶ [1] in EventListener @3127067648 | | 7 0 % | 28 0 % |
| ▶ [1] in InternalNode @3127067872 | | 6 0 % | 56 0 % |

Allocation Instrumentation

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

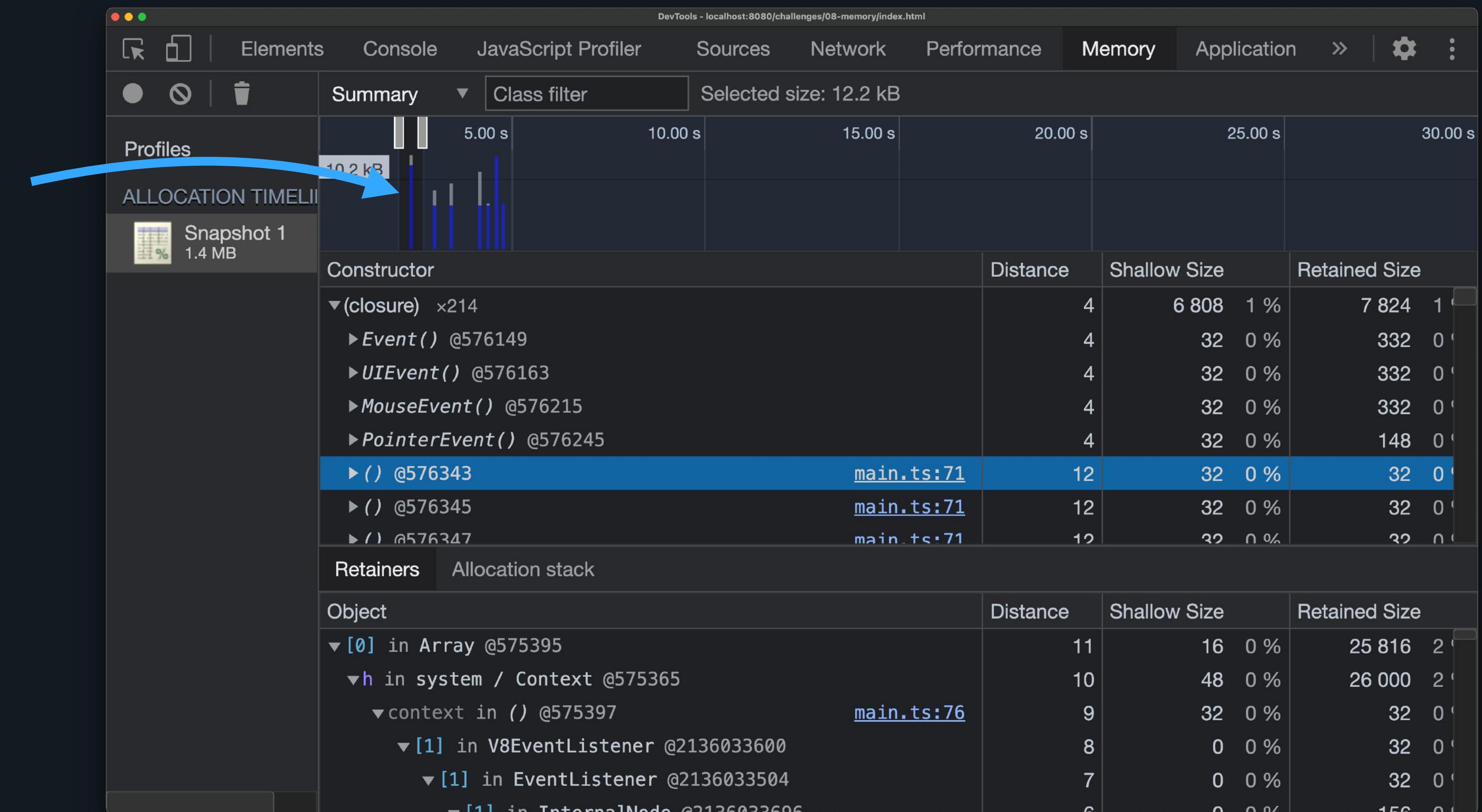
Allocation Instrumentation

Identify in timeline where objects are created and retained



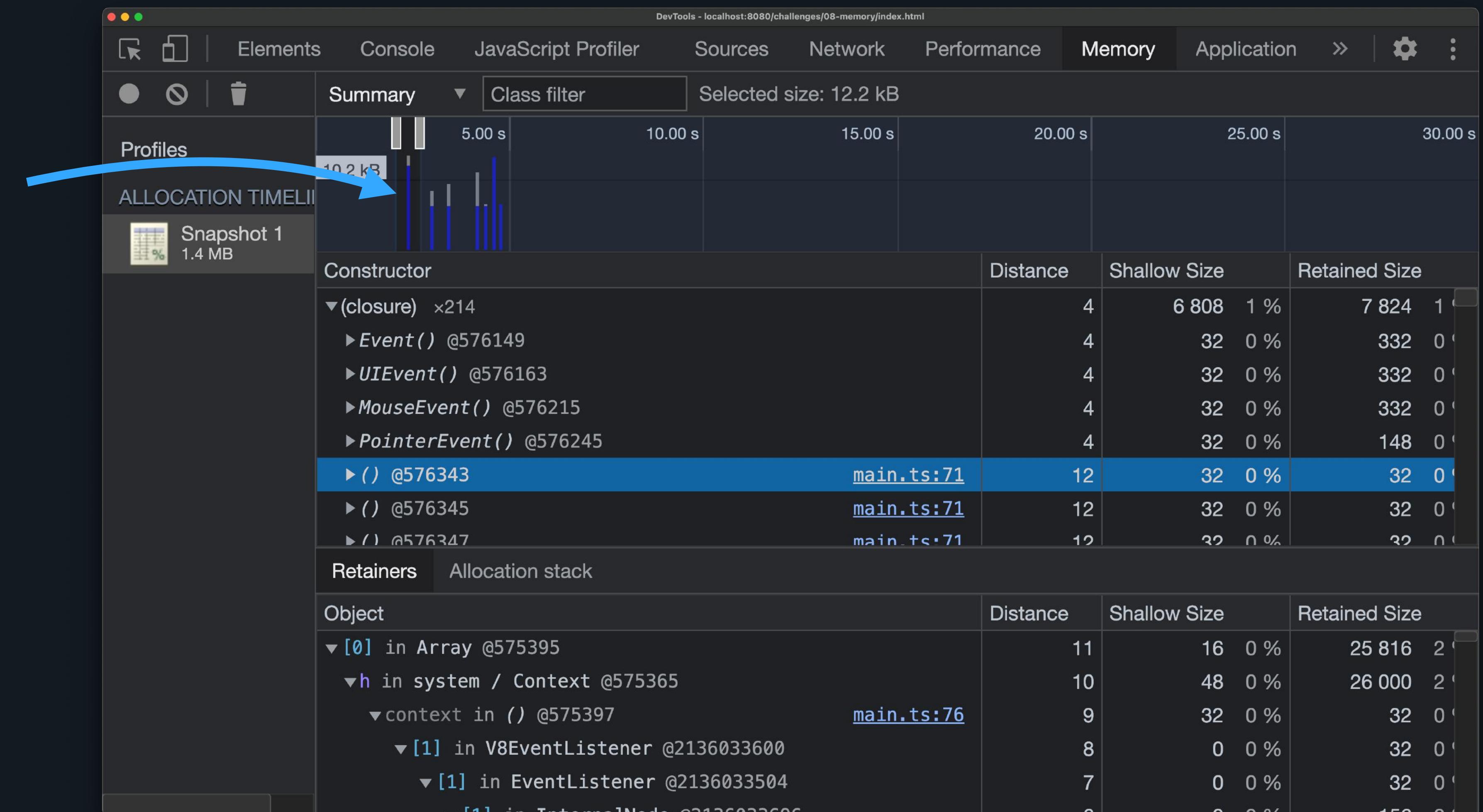
Allocation Instrumentation

Height of each bar represents the size of the memory allocation



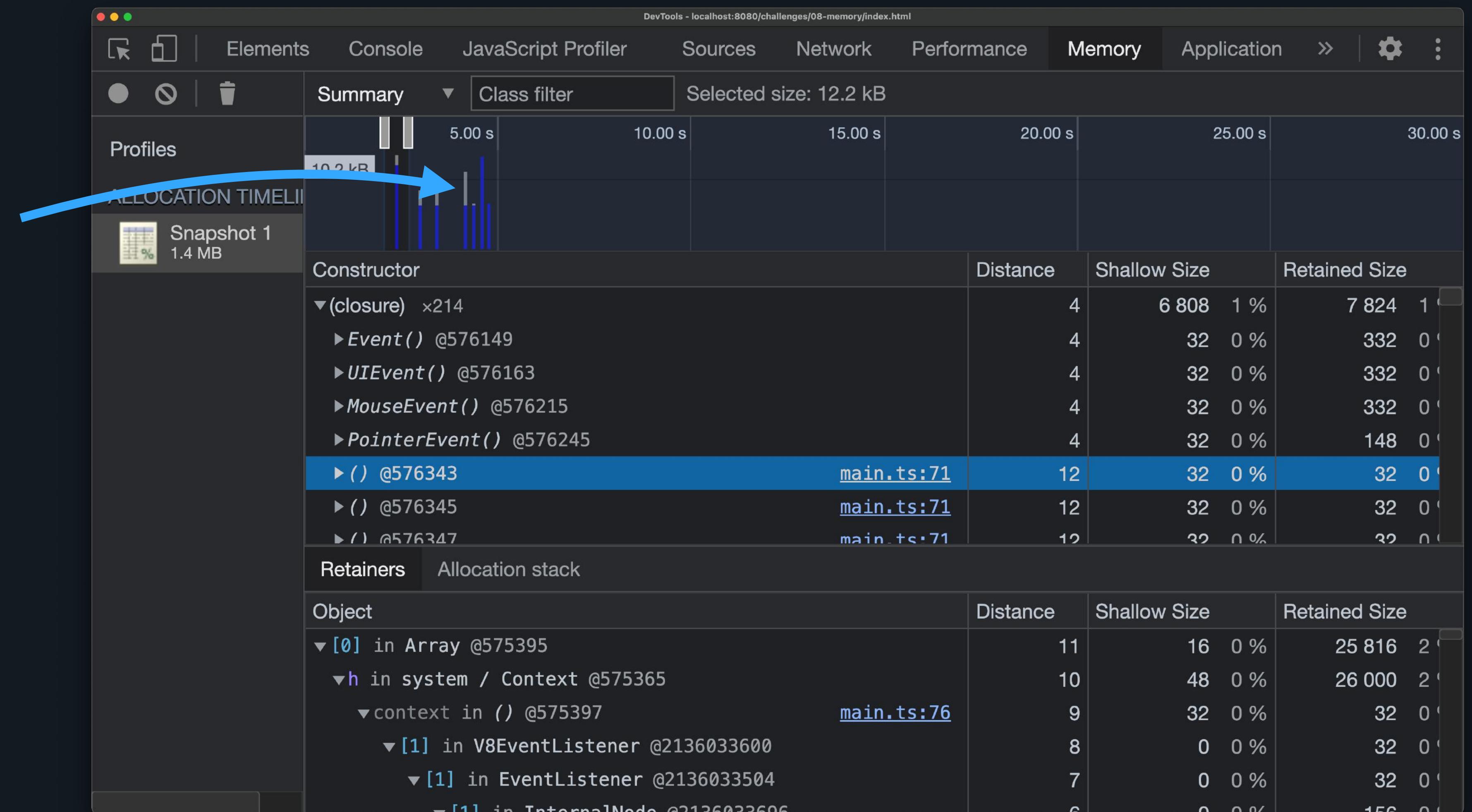
Allocation Instrumentation

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

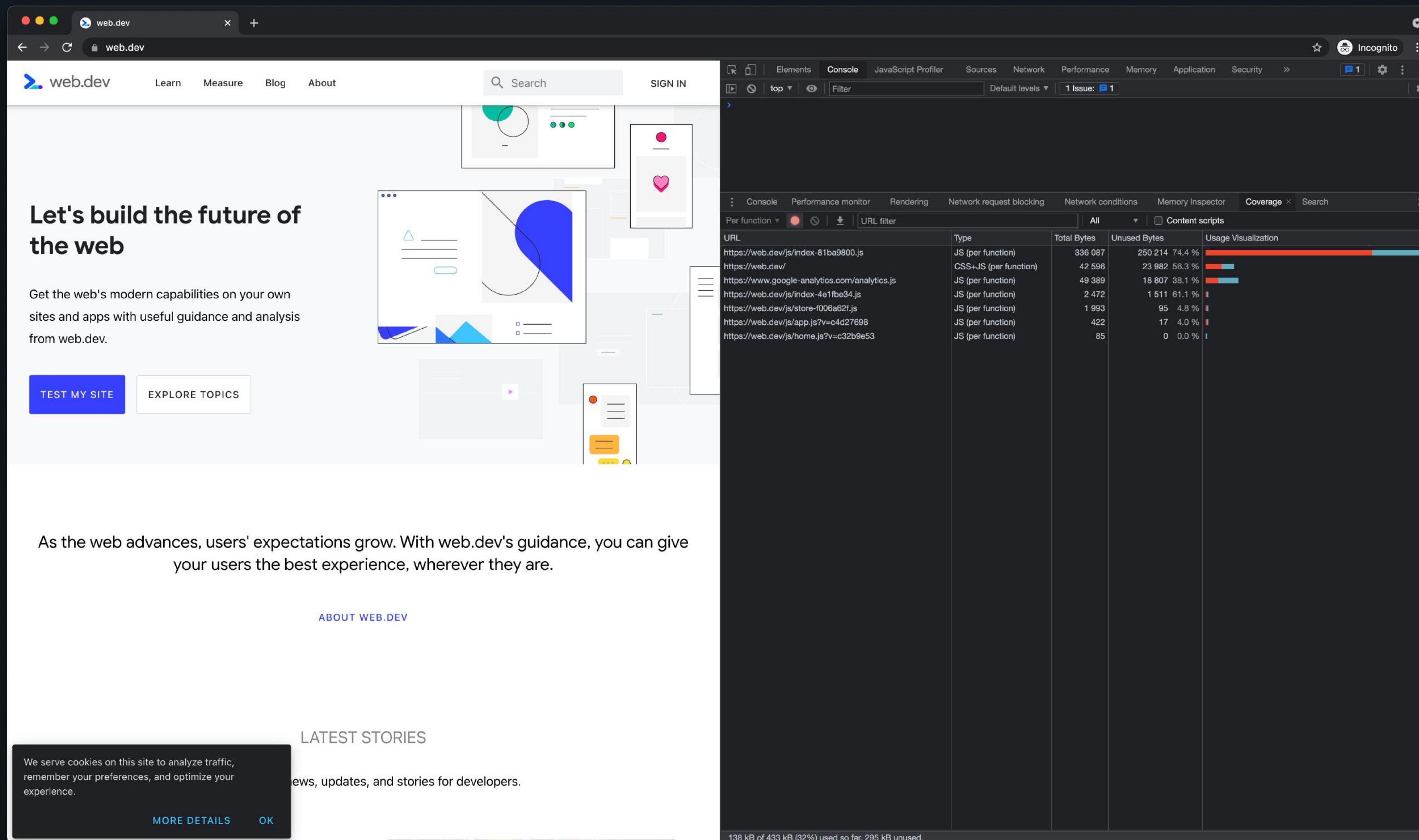


Allocation Instrumentation

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 challenges/08-memory/main.ts file and examine the **createClosure()** function. What is causing this memory leak?

DevTools: Coverage

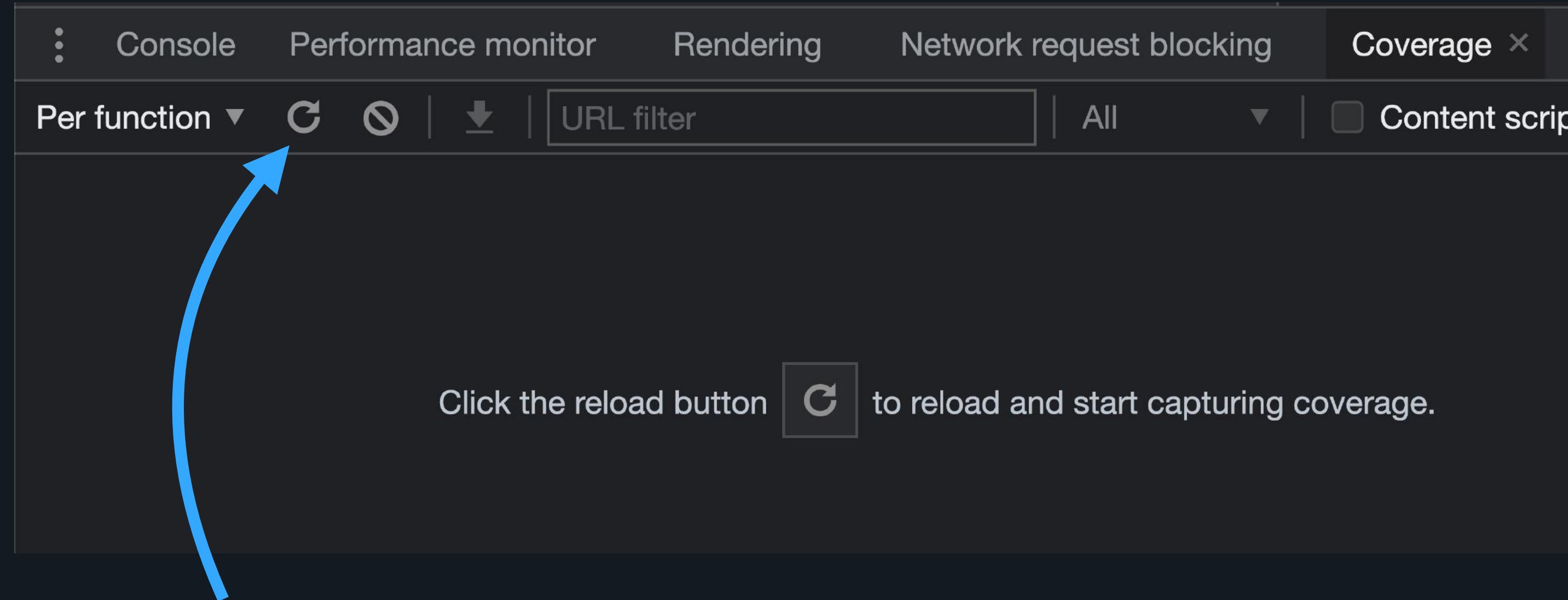
DevTools: Coverage



DevTools: Coverage

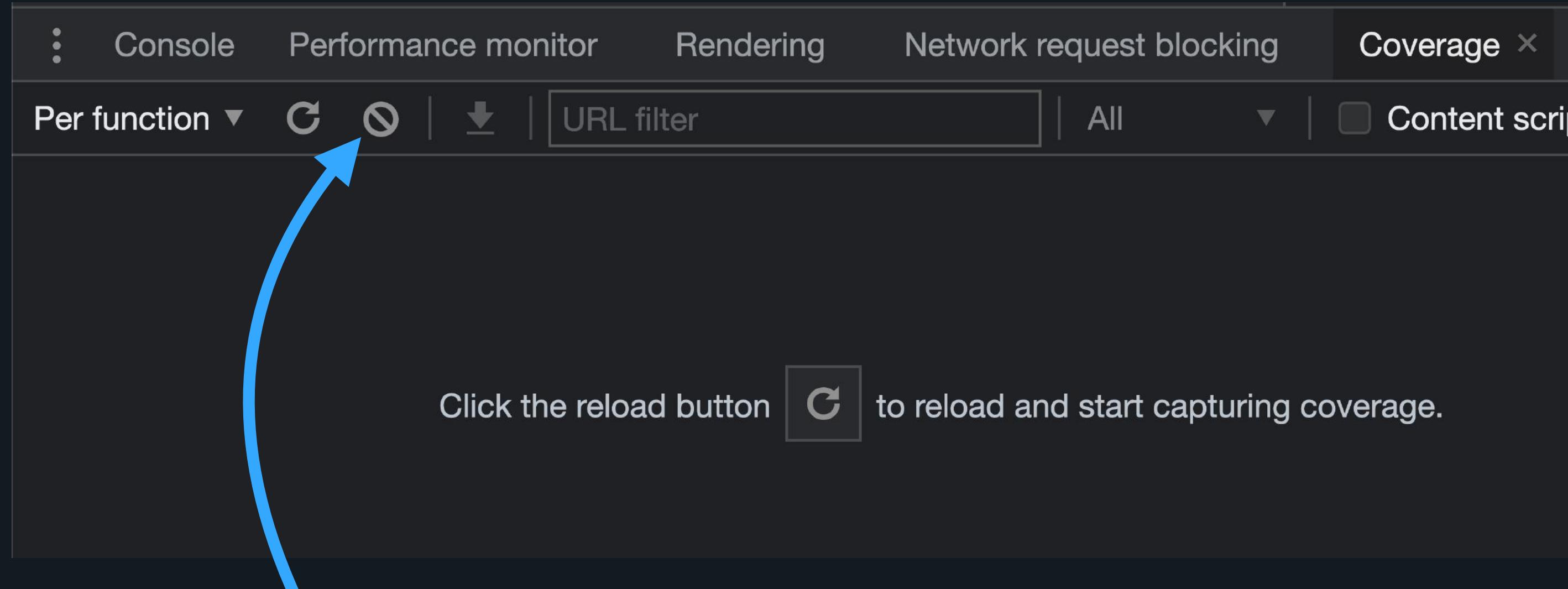
-  Resources containing unused code
-  Identify unused JS
-  Identify unused CSS

DevTools: Coverage



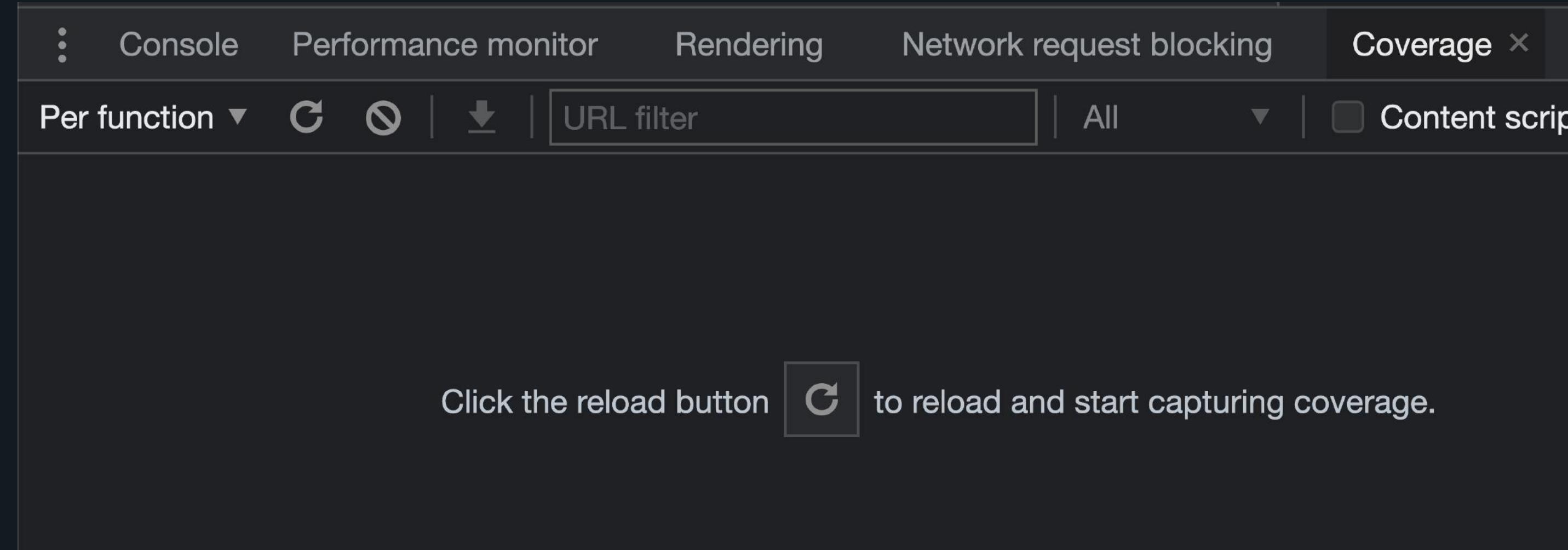
Reload and start capturing
coverage

DevTools: Coverage



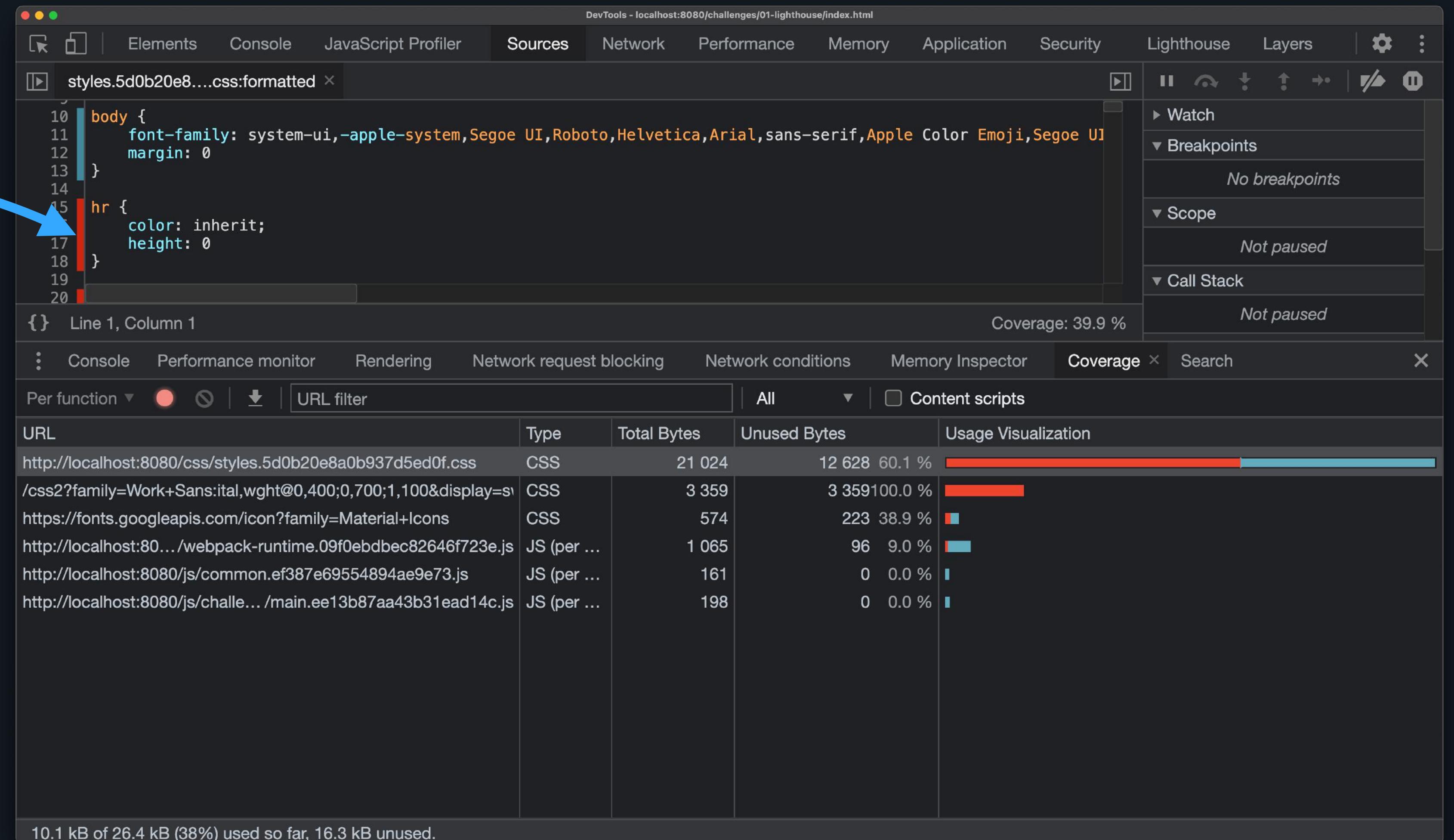
Clear

DevTools: Coverage



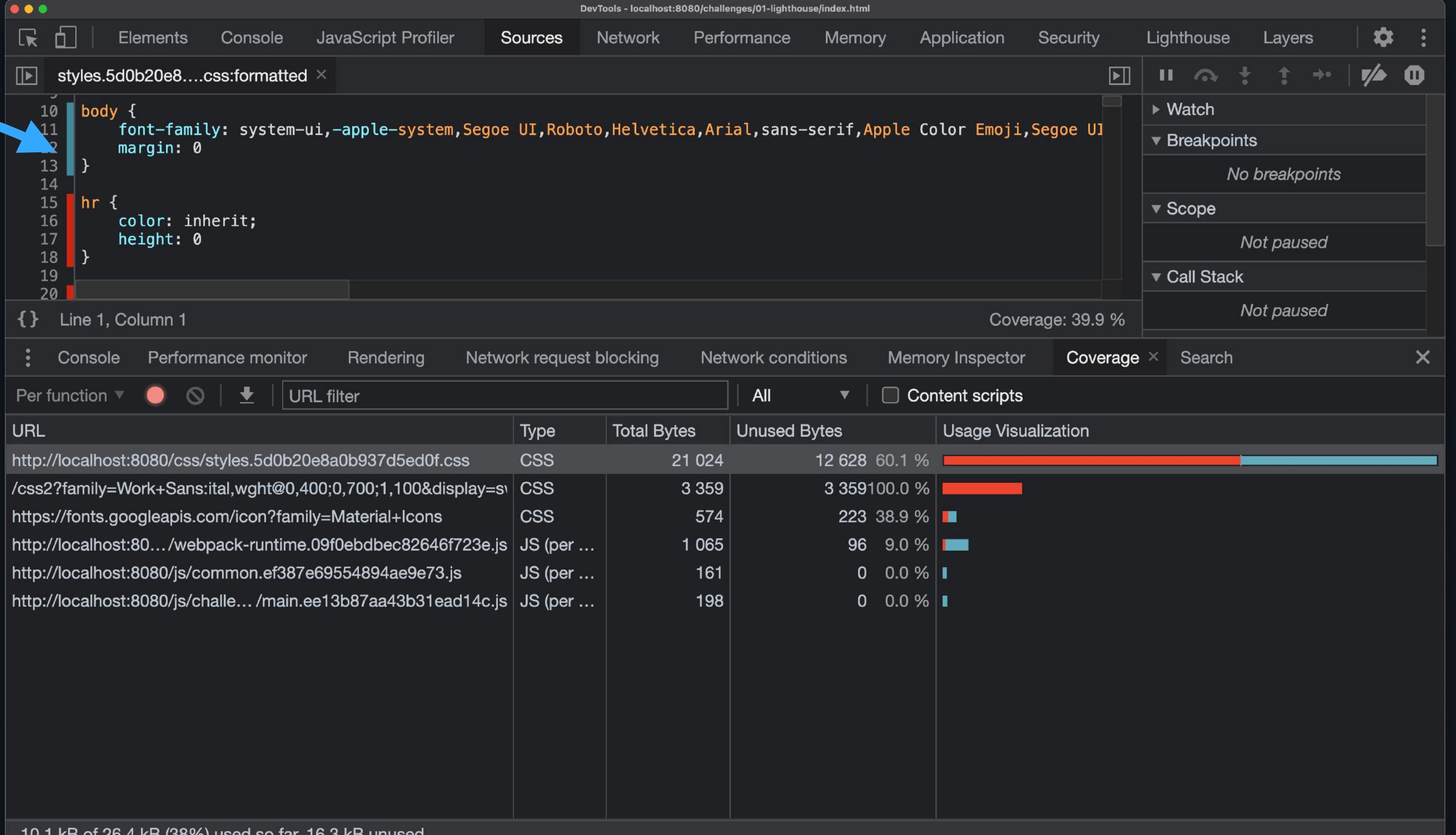
DevTools: Coverage

Unused code



DevTools: Coverage

Used code



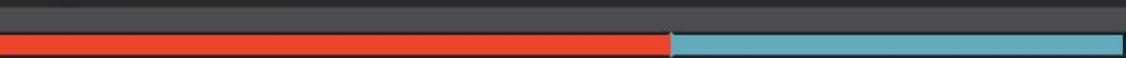
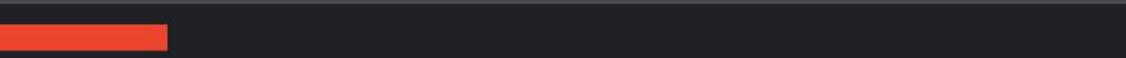
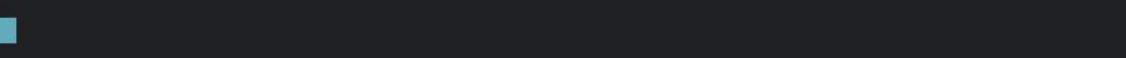
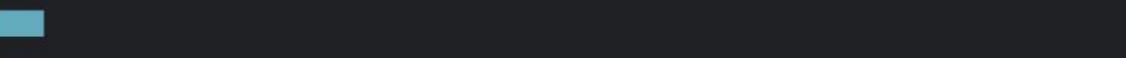
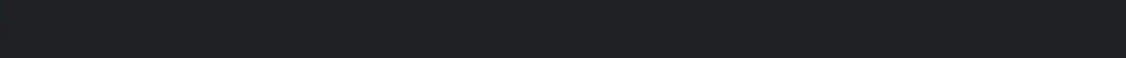
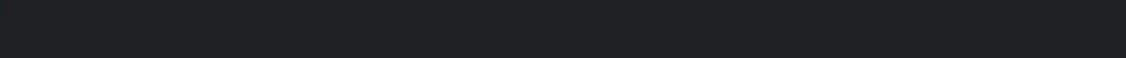
The screenshot shows the Google Chrome DevTools Coverage tab. A blue arrow points from the text "Used code" to the highlighted line of CSS in the Sources panel.

Sources Panel:

```
body {
  font-family: system-ui,-apple-system,Segoe UI,Roboto,Helvetica,Arial,sans-serif,Apple Color Emoji,Segoe UI
  margin: 0
}
hr {
  color: inherit;
  height: 0
}
```

Coverage Statistics: Coverage: 39.9 %

Network Table:

| URL | Type | Total Bytes | Unused Bytes | Usage Visualization |
|--|--------------|-------------|--------------|---|
| http://localhost:8080/css/styles.5d0b20e8a0b937d5ed0f.css | CSS | 21 024 | 12 628 | 60.1 %  |
| /css2?family=Work+Sans:ital,wght@0,400;0,700;1,100&display=s | CSS | 3 359 | 3 359 | 100.0 %  |
| https://fonts.googleapis.com/icon?family=Material+Icons | CSS | 574 | 223 | 38.9 %  |
| http://localhost:80.../webpack-runtime.09f0ebdbec82646f723e.js | JS (per ...) | 1 065 | 96 | 9.0 %  |
| http://localhost:8080/js/common.ef387e69554894ae9e73.js | JS (per ...) | 161 | 0 | 0.0 %  |
| http://localhost:8080/js/challe... /main.ee13b87aa43b31ead14c.js | JS (per ...) | 198 | 0 | 0.0 %  |

Summary: 10.1 kB of 26.4 kB (38%) used so far, 16.3 kB unused.

DevTools: Coverage

Resource URL



DevTools - localhost:8080/challenges/01-lighthouse/index.html

Sources Network Performance Memory Application Security Lighthouse Layers

Watch Breakpoints No breakpoints Scope Not paused Call Stack Not paused

Coverage: 39.9 %

Line 1, Column 1

Console Performance monitor Rendering Network request blocking Network conditions Memory Inspector Coverage Search

Per function URL filter All Content scripts

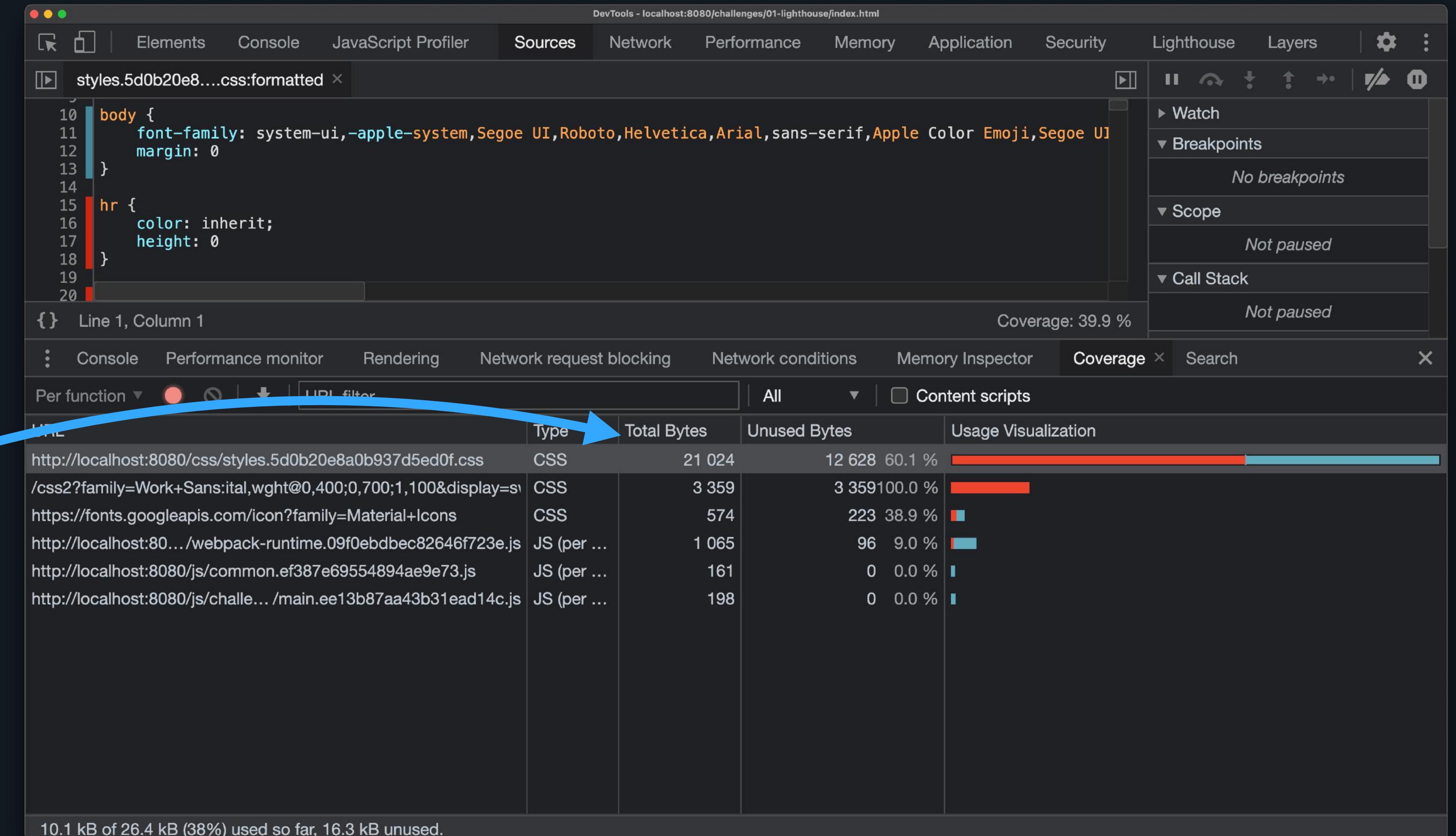
| URL | Type | Total Bytes | Unused Bytes | Usage Visualization |
|---|--------------|-------------|--------------|---------------------|
| http://localhost:8080/css/styles.5d0b20e8a0b937d5ed0f.css | CSS | 21 024 | 12 628 | 60.1 % |
| /css2?family=Work+Sans:ital,wght@0,400;0,700;1,100&display=s | CSS | 3 359 | 3 359 | 100.0 % |
| https://fonts.googleapis.com/icon?family=Material+Icons | CSS | 574 | 223 | 38.9 % |
| http://localhost:80.../webpack-runtime.09f0ebdbec82646f723e.js | JS (per ...) | 1 065 | 96 | 9.0 % |
| http://localhost:8080/js/common.ef387e69554894ae9e73.js | JS (per ...) | 161 | 0 | 0.0 % |
| http://localhost:8080/js/challe.../main.ee13b87aa43b31ead14c.js | JS (per ...) | 198 | 0 | 0.0 % |

10.1 kB of 26.4 kB (38%) used so far, 16.3 kB unused.

| URL | Type | Total Bytes | Unused Bytes | Usage Visualization |
|---|--------------|-------------|--------------|---------------------|
| http://localhost:8080/css/styles.5d0b20e8a0b937d5ed0f.css | CSS | 21 024 | 12 628 | 60.1 % |
| /css2?family=Work+Sans:ital,wght@0,400;0,700;1,100&display=s | CSS | 3 359 | 3 359 | 100.0 % |
| https://fonts.googleapis.com/icon?family=Material+Icons | CSS | 574 | 223 | 38.9 % |
| http://localhost:80.../webpack-runtime.09f0ebdbec82646f723e.js | JS (per ...) | 1 065 | 96 | 9.0 % |
| http://localhost:8080/js/common.ef387e69554894ae9e73.js | JS (per ...) | 161 | 0 | 0.0 % |
| http://localhost:8080/js/challe.../main.ee13b87aa43b31ead14c.js | JS (per ...) | 198 | 0 | 0.0 % |

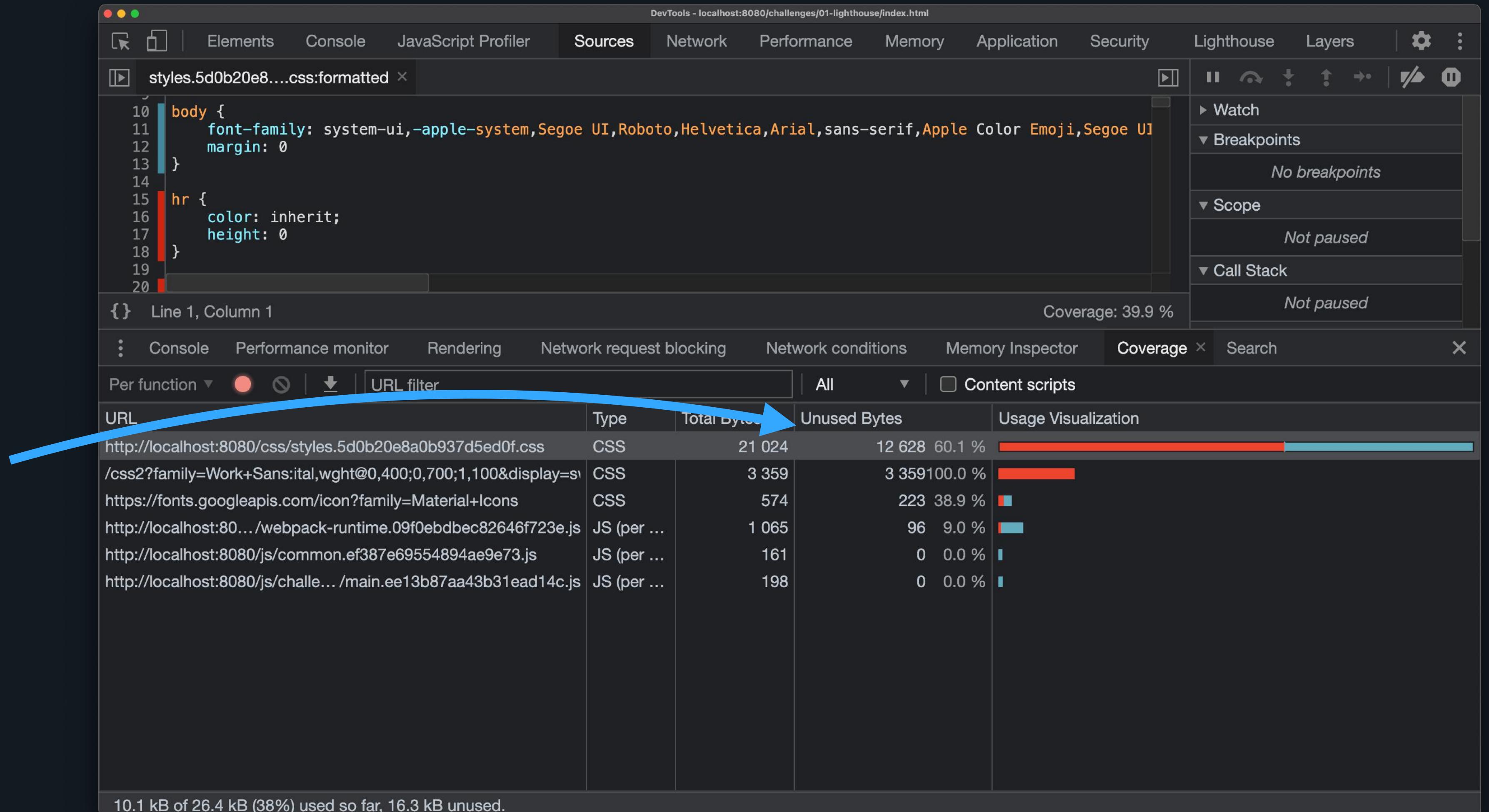
DevTools: Coverage

Resource size in bytes



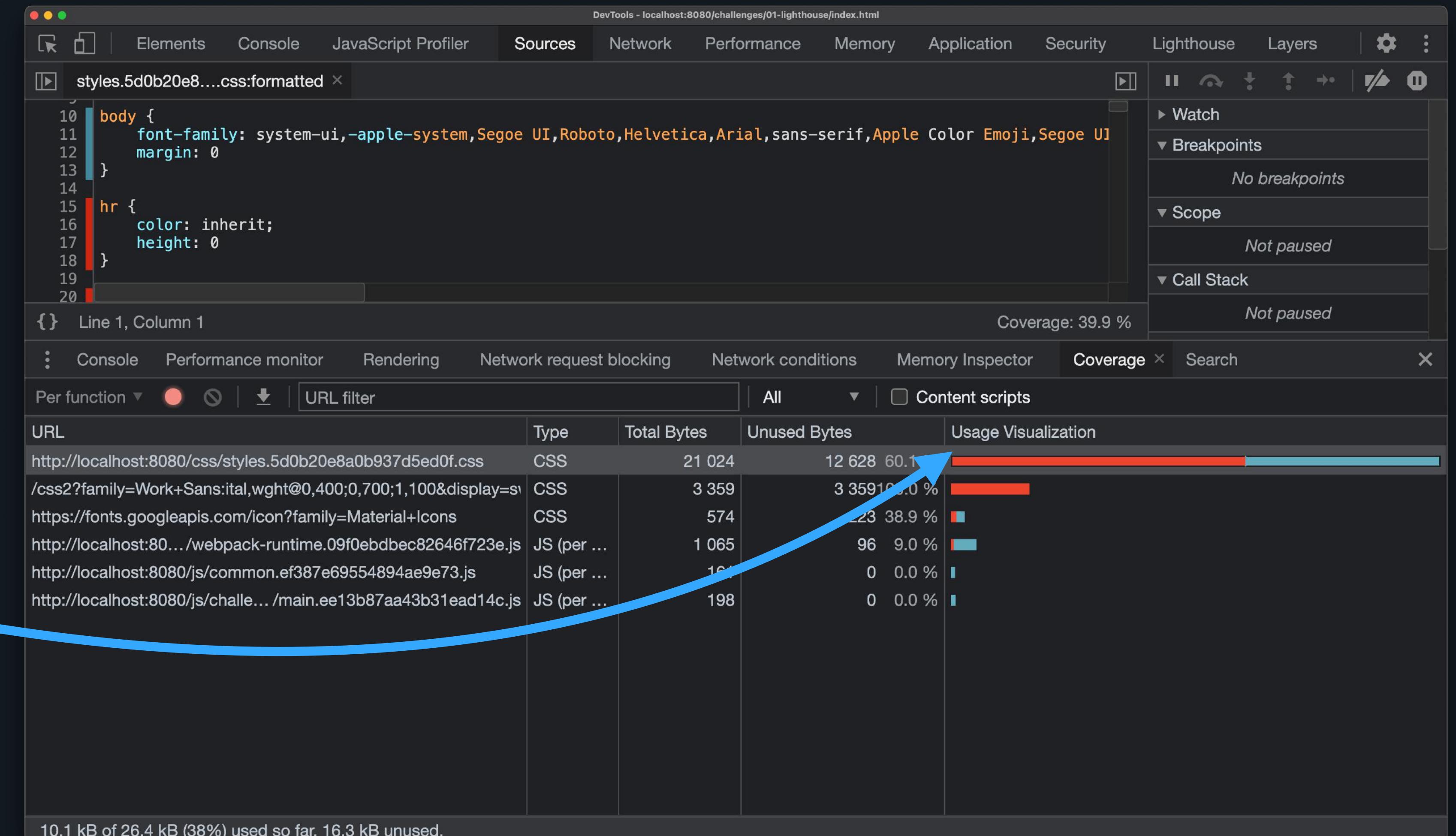
DevTools: Coverage

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