

Gold Sponsors



Community Supporter



Optimizing Interaction to Next Paint

Jeremy Wagner — jlwagner.net — @ma1chata — PrDC Regina

What is responsiveness?

gShoe product Q&A:

What is gShoe?

What technology does gShoe use?

How much does gShoe cost?

Poor responsiveness

gShoe product Q&A:

What is gShoe?

What technology does gShoe use?

How much does gShoe cost?

Good responsiveness

What causes poor responsiveness?

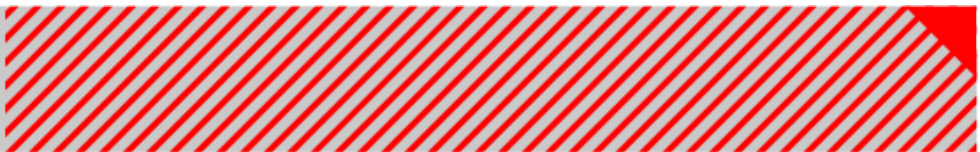
Main thread

Task

Evaluate script

Compile script

Compile code



Task

Event: keyup

Function call

fetchWrapper

sendFetch



Task

Evaluate script

Compile script

Compile code

77.6 ms (total)

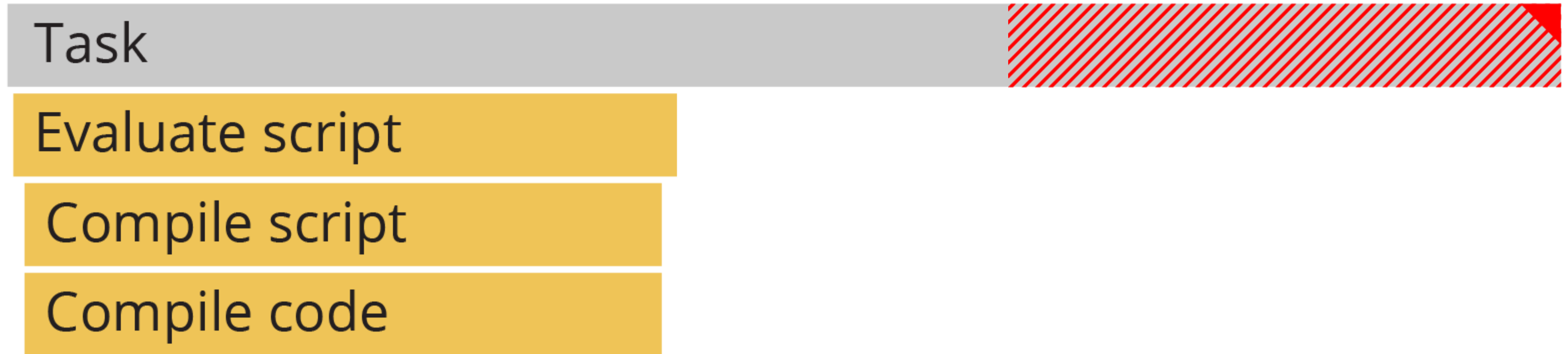
27.6 ms (blocking)

Task

Evaluate script

Compile script

Compile code



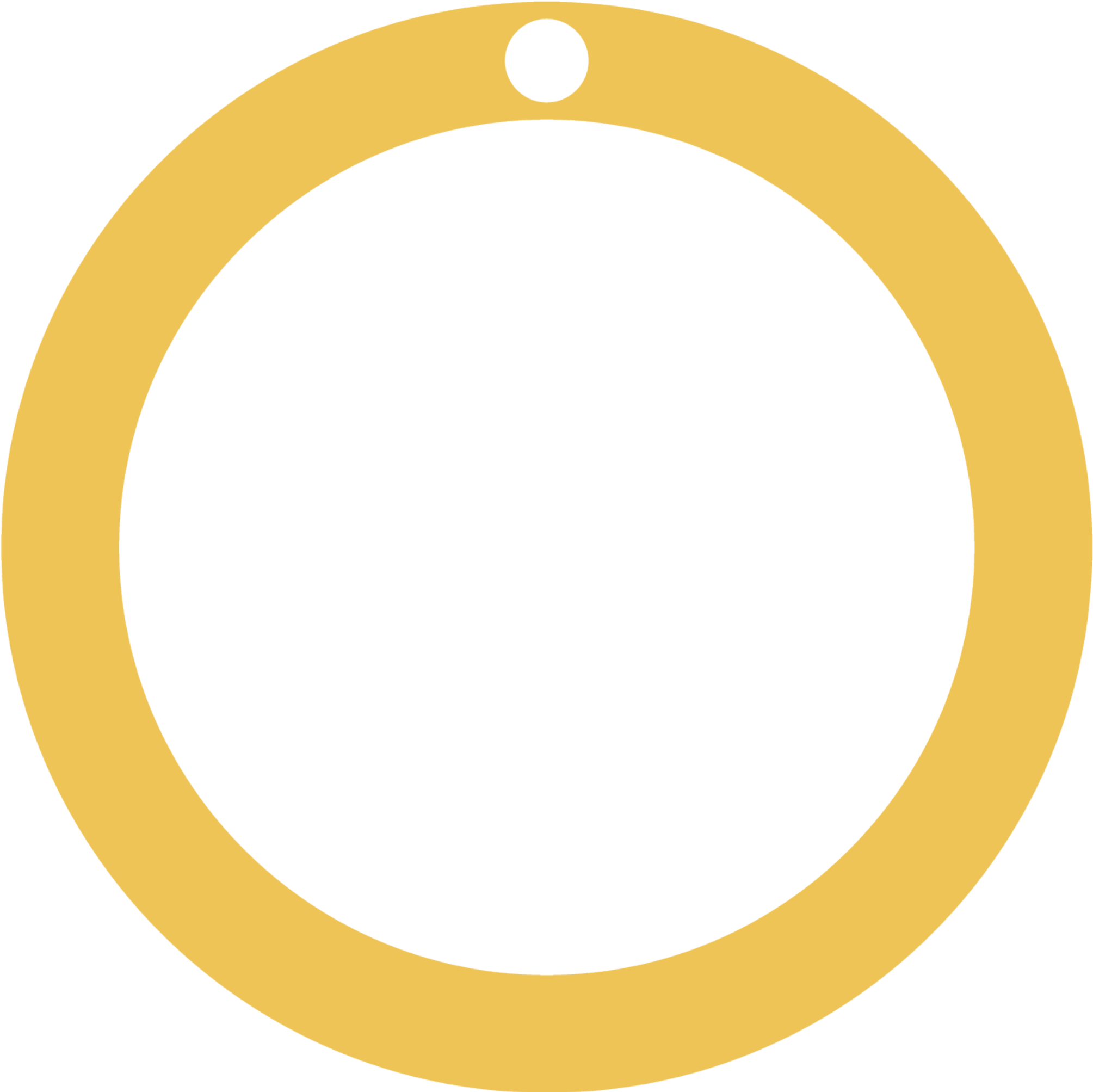
Task

Event: keydown

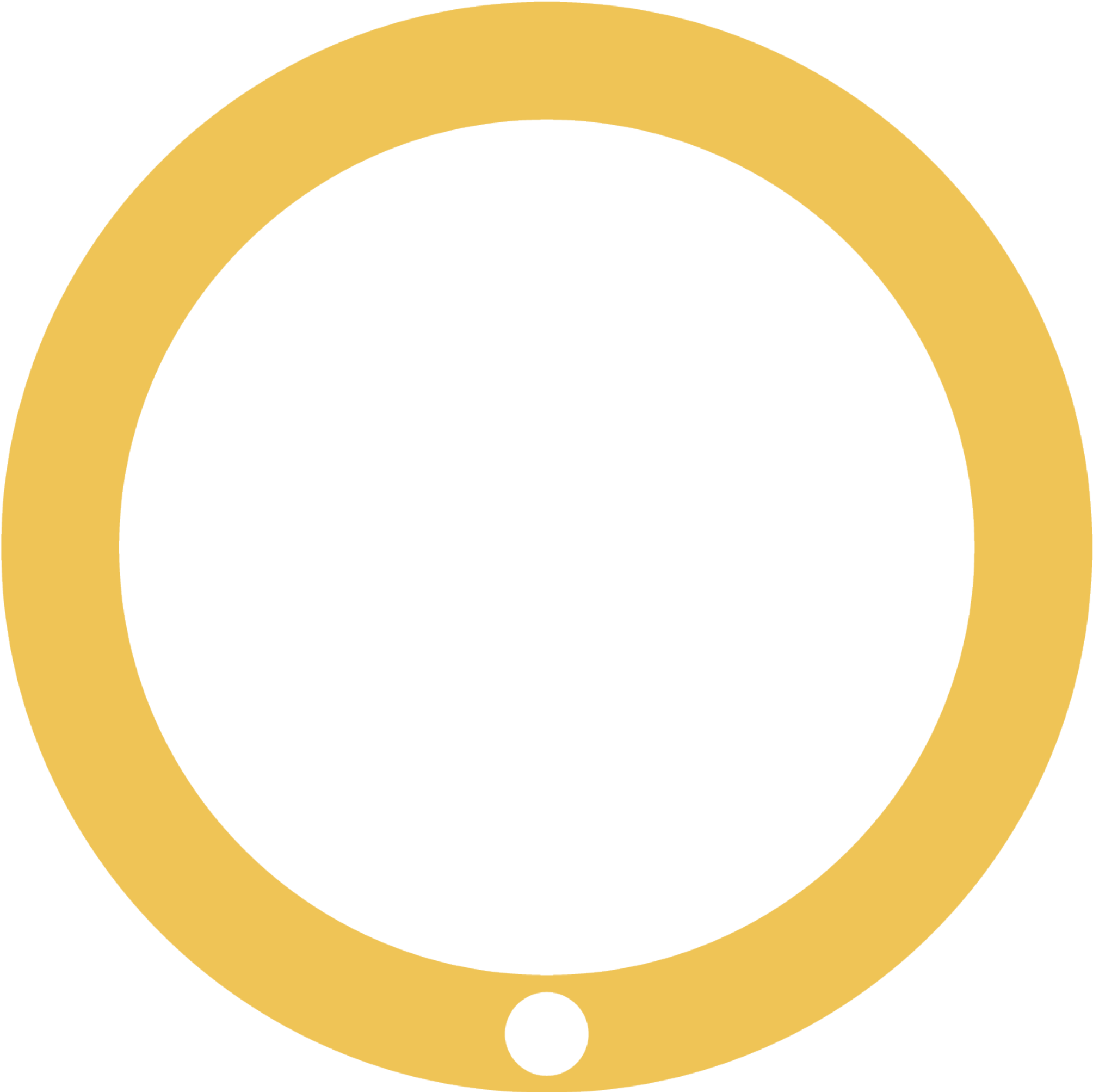
Function call

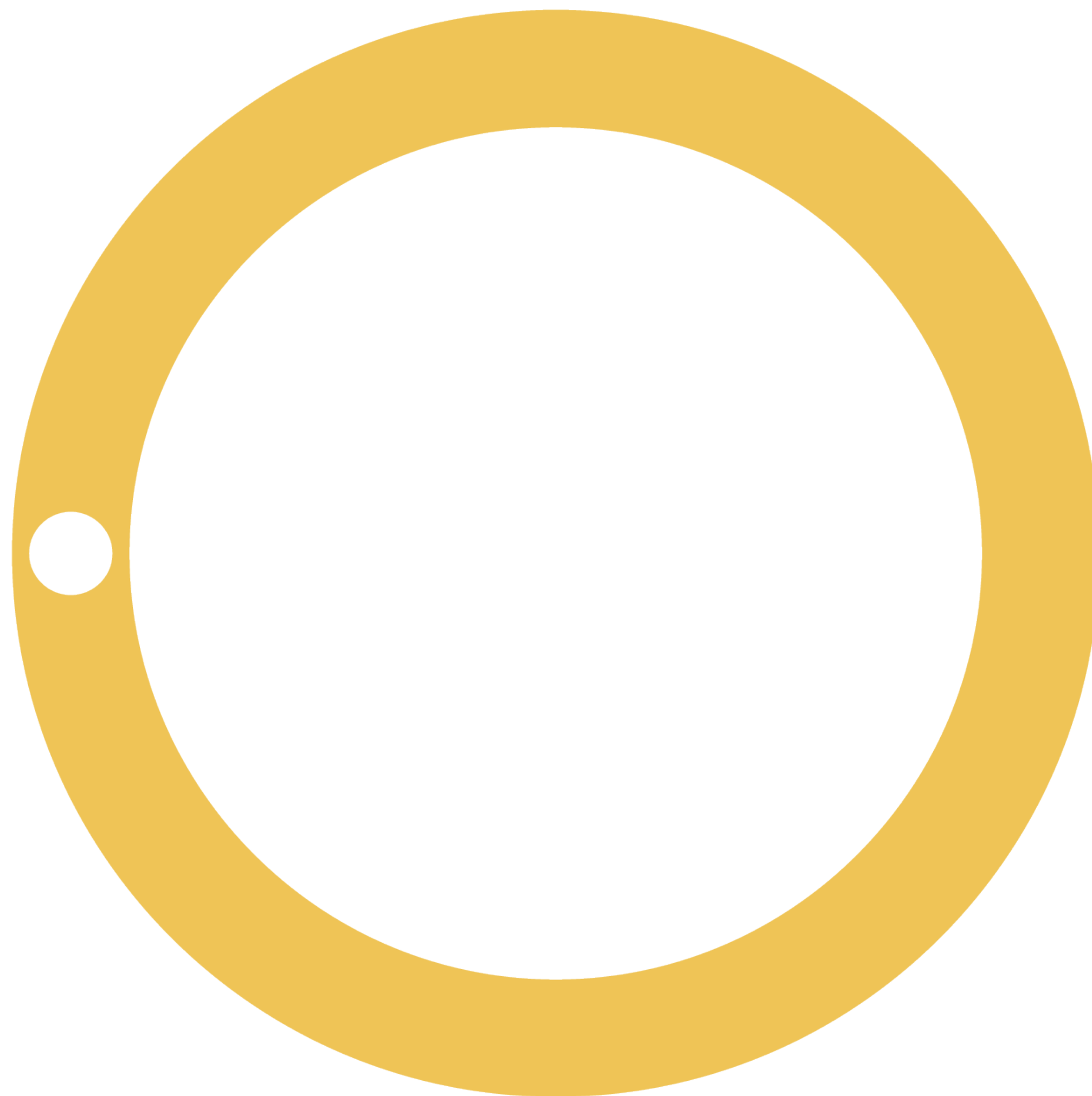
showSpinner

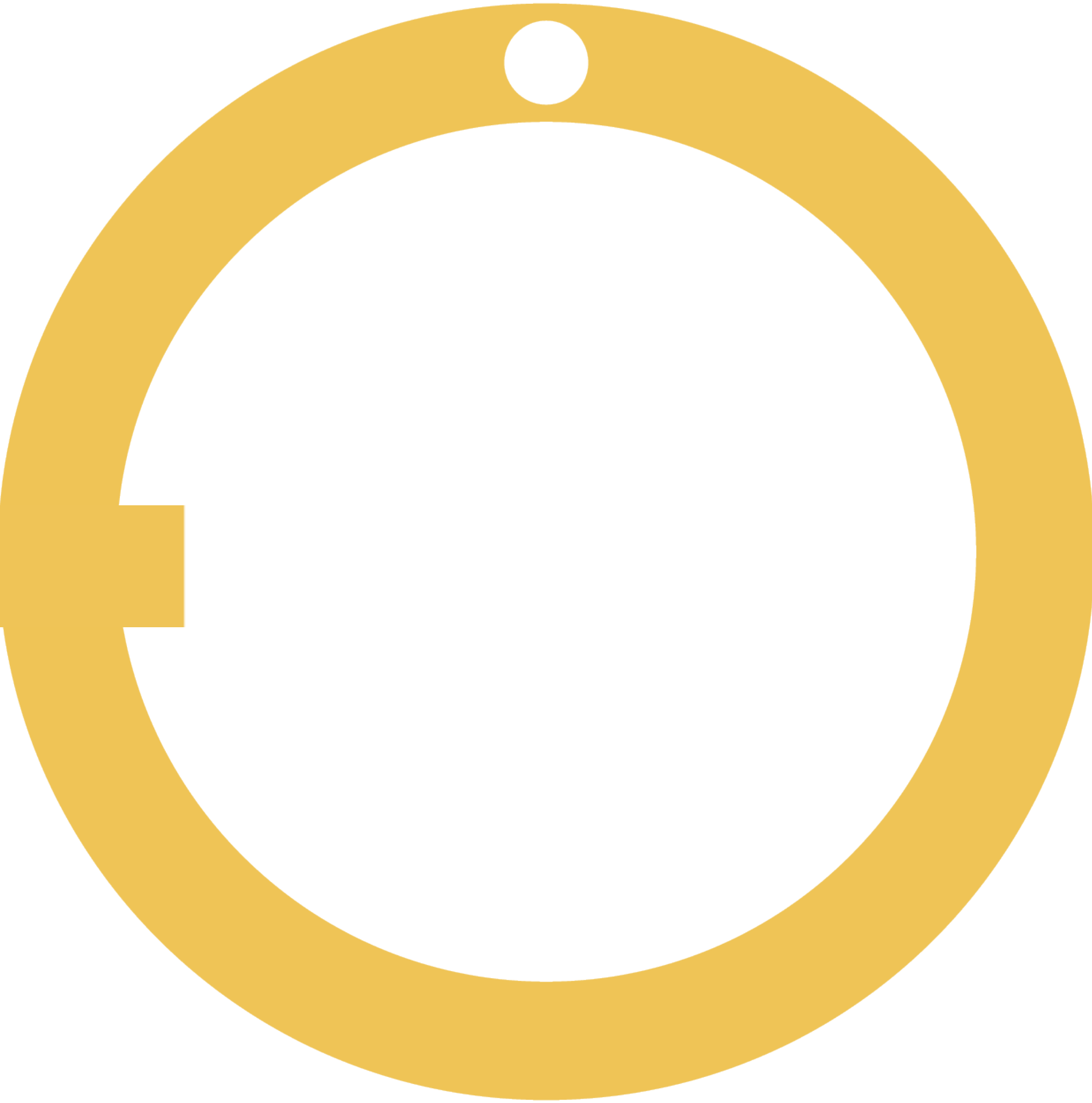
Why so much focus on tasks?









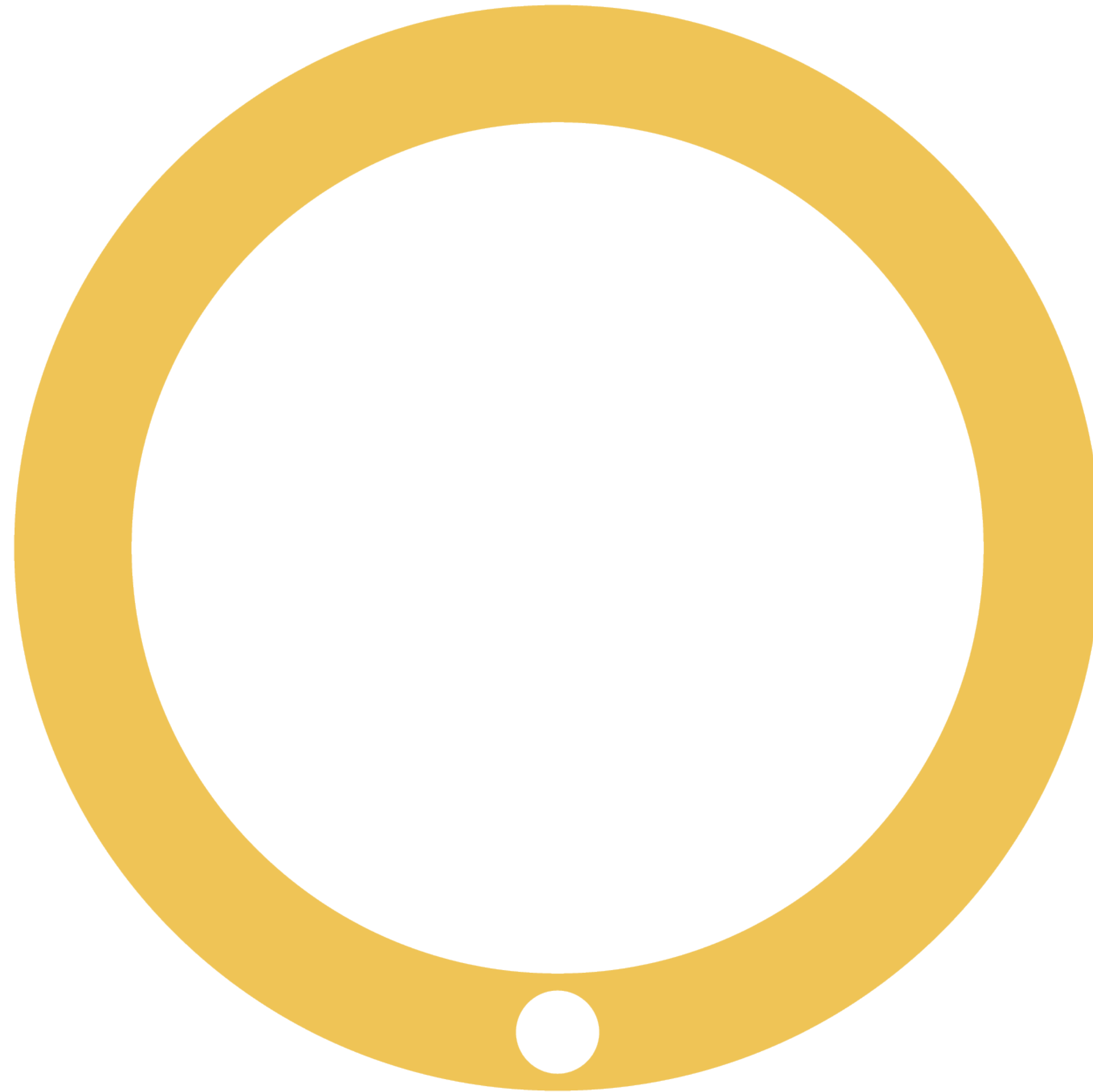


Event: keyup

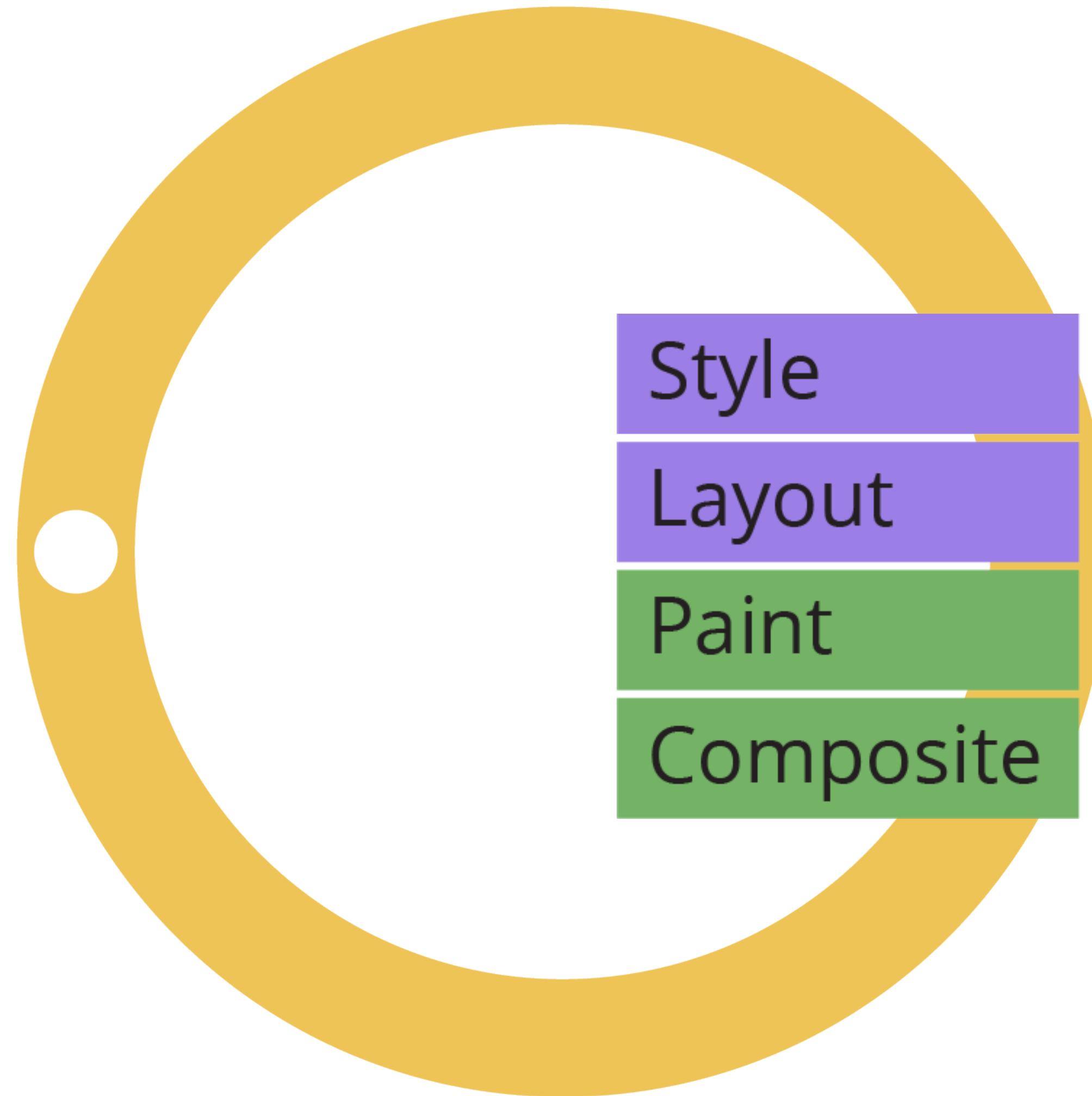
Event: keyup

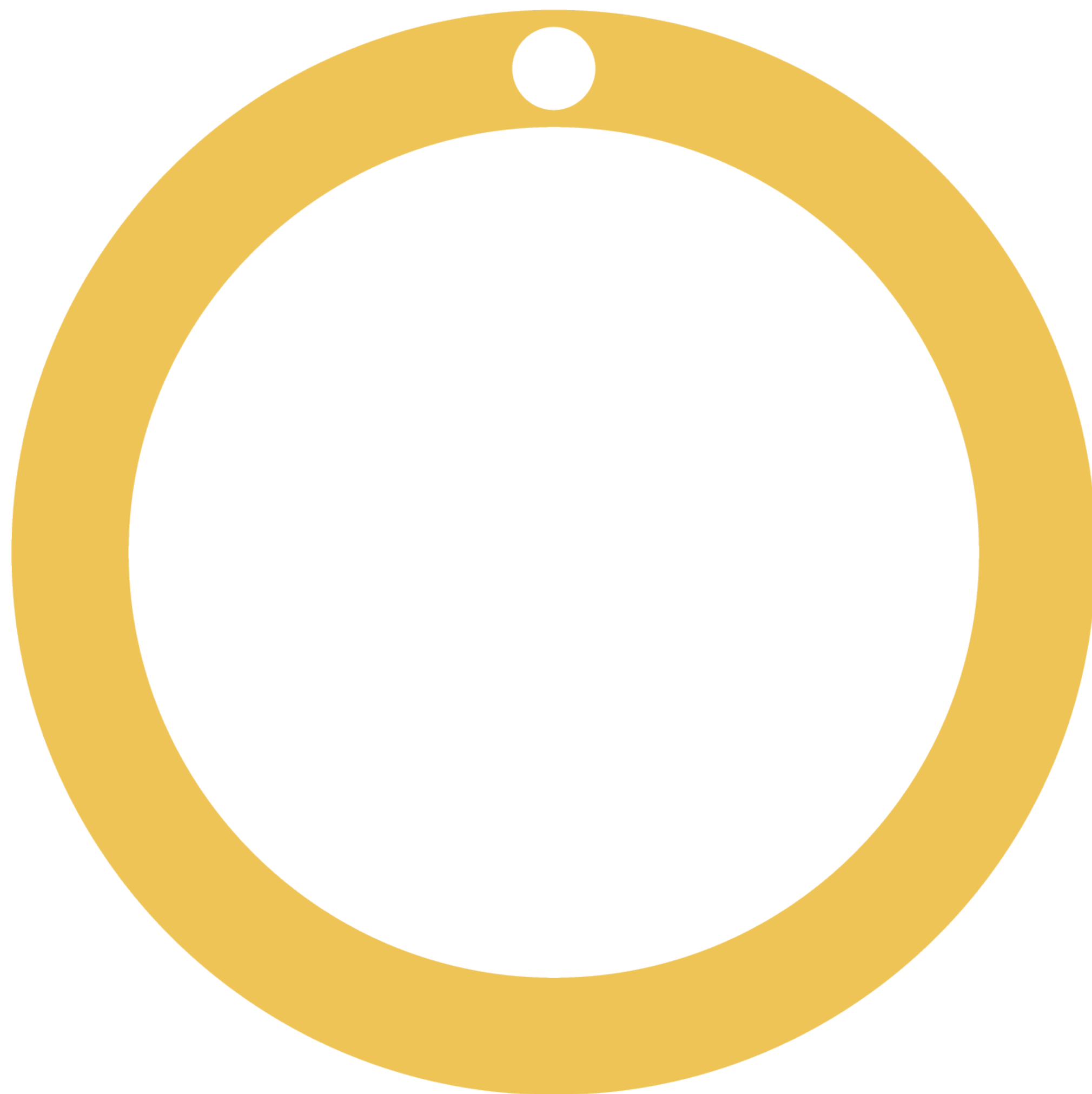


Event: keyup



Event: keyup





Style

Layout

Paint

Composite



Style

Layout

Paint

Composite

Page Unresponsive

You can wait for it to become responsive or exit the page.



Welcome — jlwagner.net

Exit Page

Wait

The PWA community is coming together for #PWASummit22. Have a great story about developing a web app? [Submit your talk today](#)

Interaction to Next Paint (INP)

May 6, 2022 — Updated Jul 18, 2022

Appears in: [Metrics](#)



Jeremy Wagner

[Twitter](#)

[GitHub](#)

[Homepage](#)

On this page



 SHARE

 SUBSCRIBE

Interaction to Next Paint (INP) is an experimental metric that assesses

- Observes certain interactions using the Event Timing API.
- Unlike First Input Delay (FID), measures more than just the input delay of the first interaction.
- Chooses the single worst* interaction latency (in milliseconds), which is representative of the page's overall responsiveness.

INP is all about the next frame.

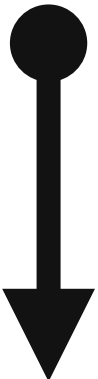
- Keyboard interactions (including physical and onscreen).
- Mouse clicks.
- Taps on touchscreens.

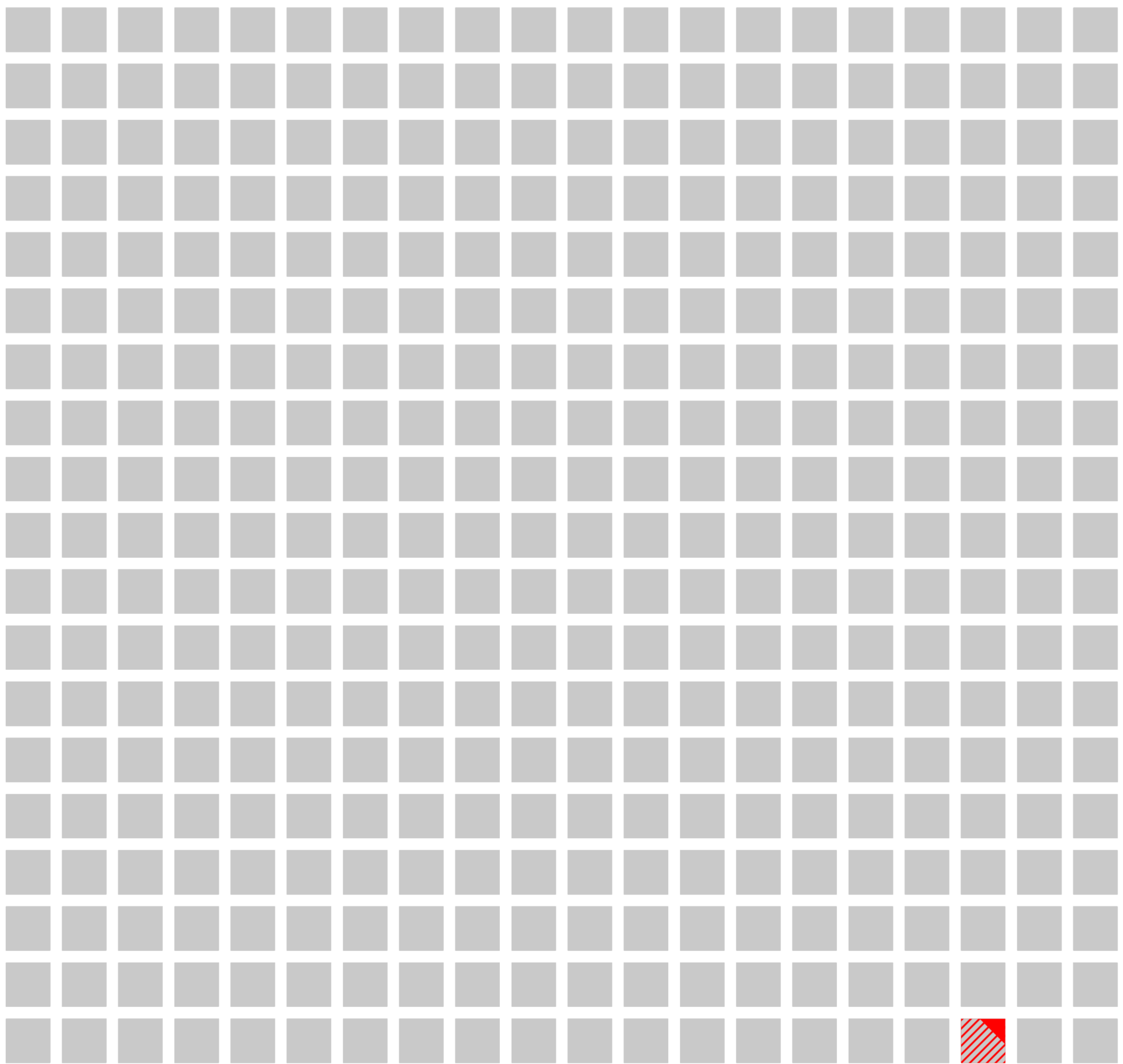
- Hovering.
- Network activity.
- Scrolling*.

One interaction



INP

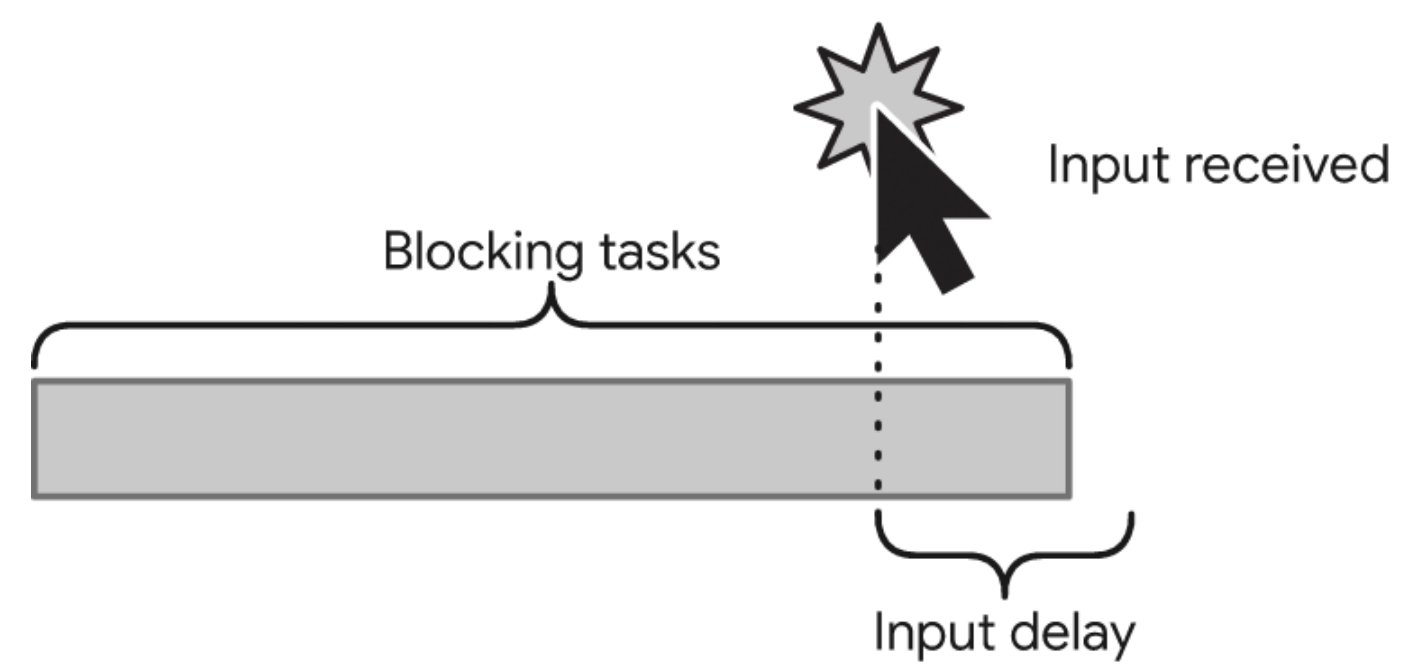


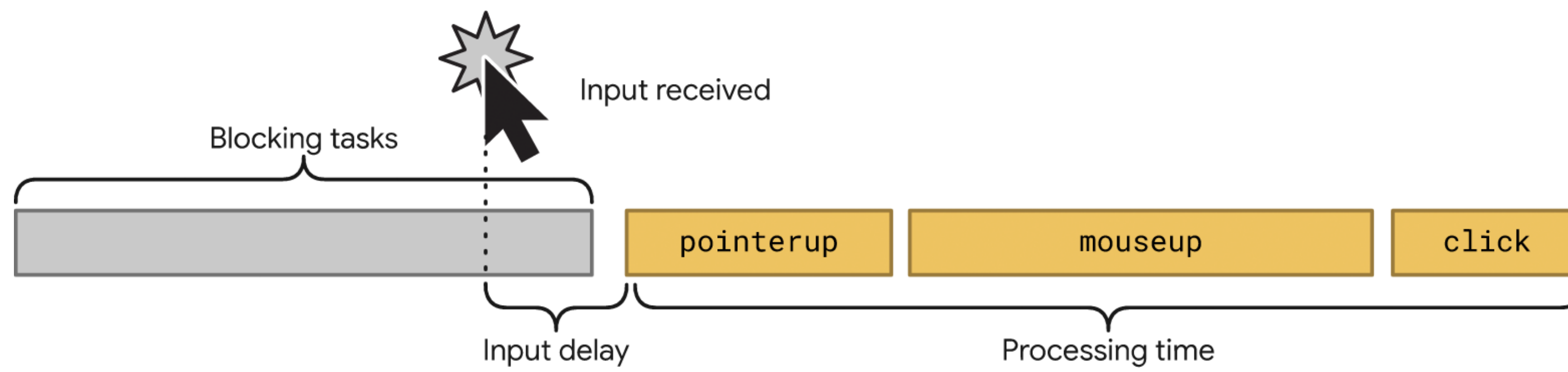


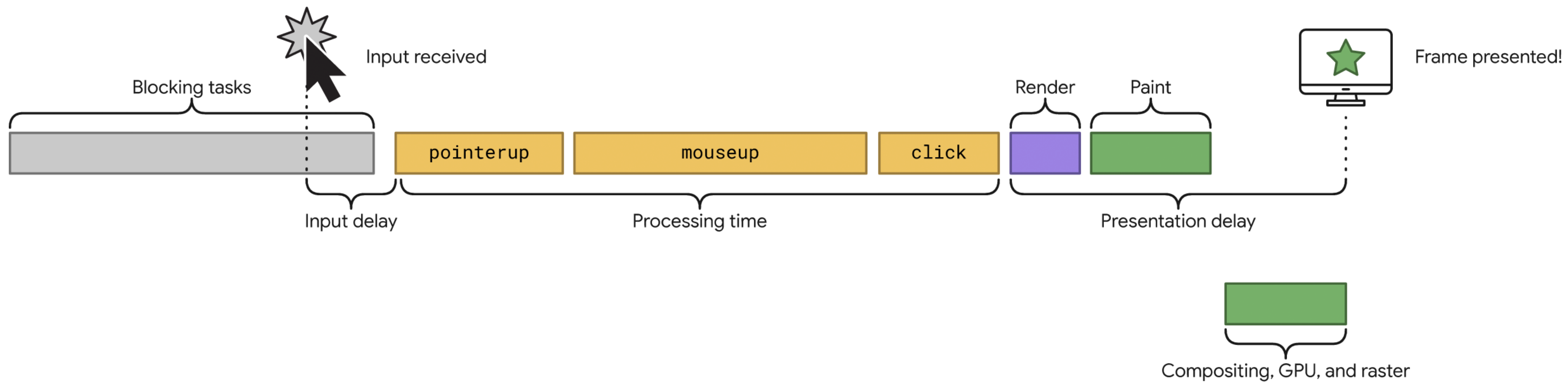
INP

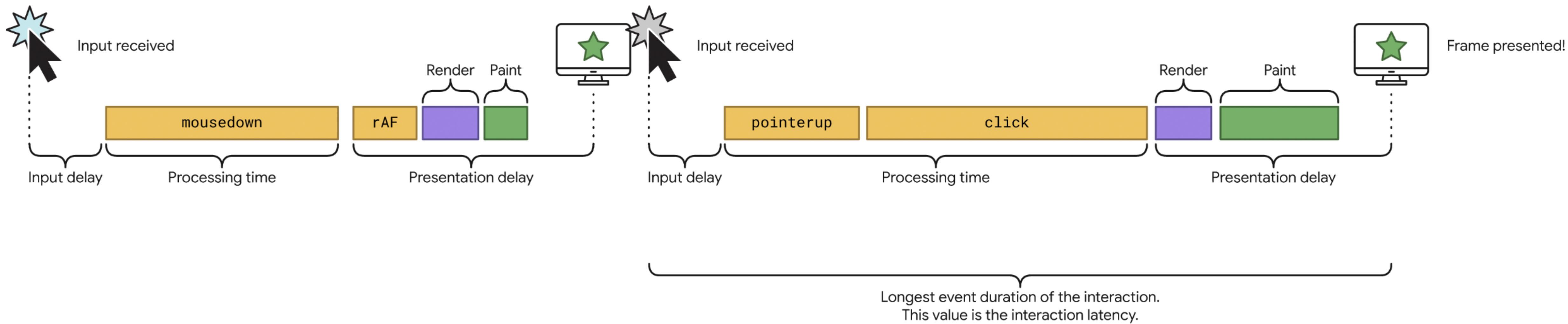
Interaction to Next Paint











Optimizing INP

```
// Vendors
import { onINP } from "web-vitals";

// https://gist.github.com/malchata/56e92306acb0c5b9fd3541ed16ce2d23
import { getSelector } from "./utils.js";

function sendMetrics({ metric, value, entries }) {
  fetch("https://mycoolwebsite.net/collect", {
    body: JSON.stringify({
      metric: value,
      element: getSelector(entries[0].target)
    }),
    method: "POST",
    keepalive: true,
    headers: {
      "Content-Type": "application/json",
    }
  });
}

onINP(sendMetrics);
```

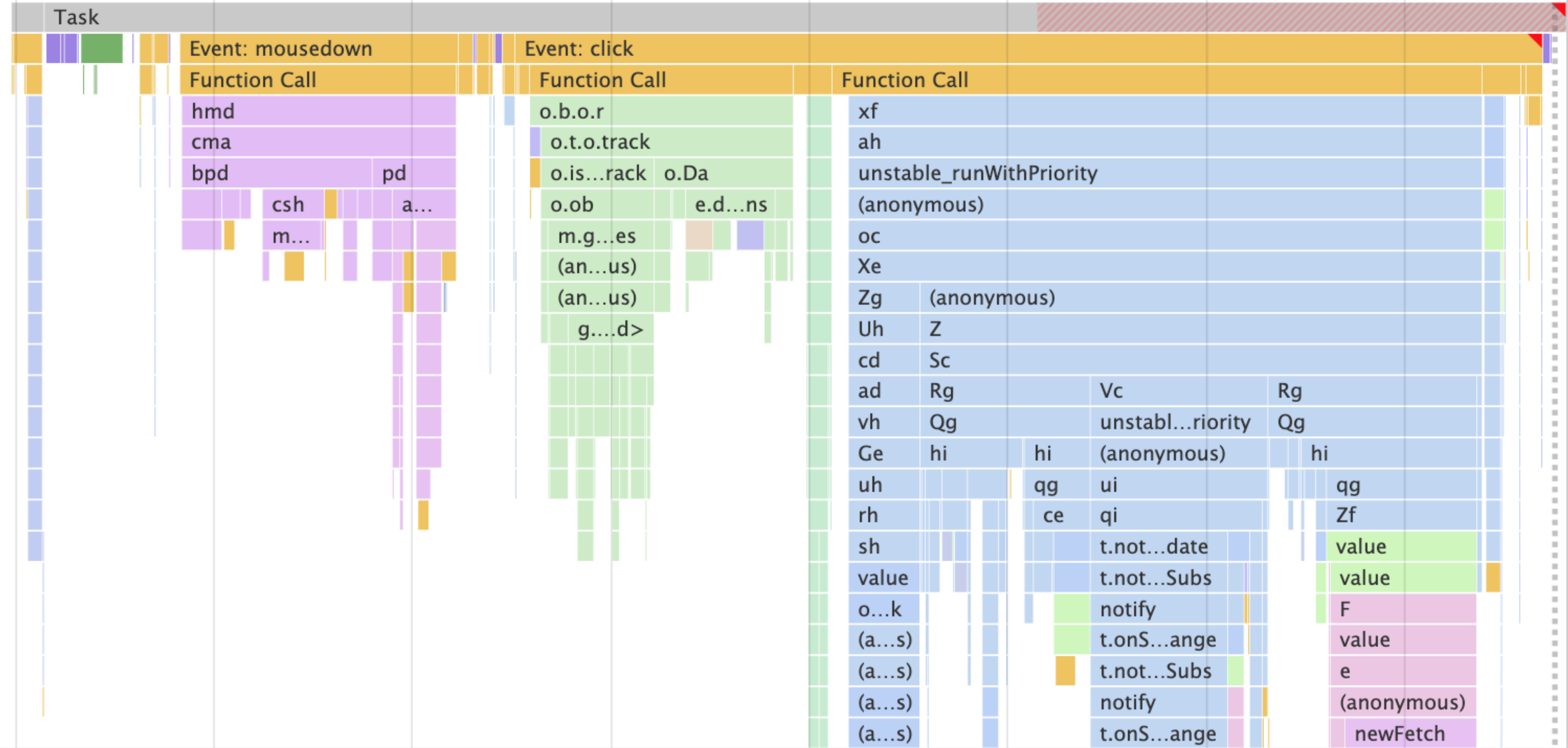
github.com/GoogleChrome/web-vitals

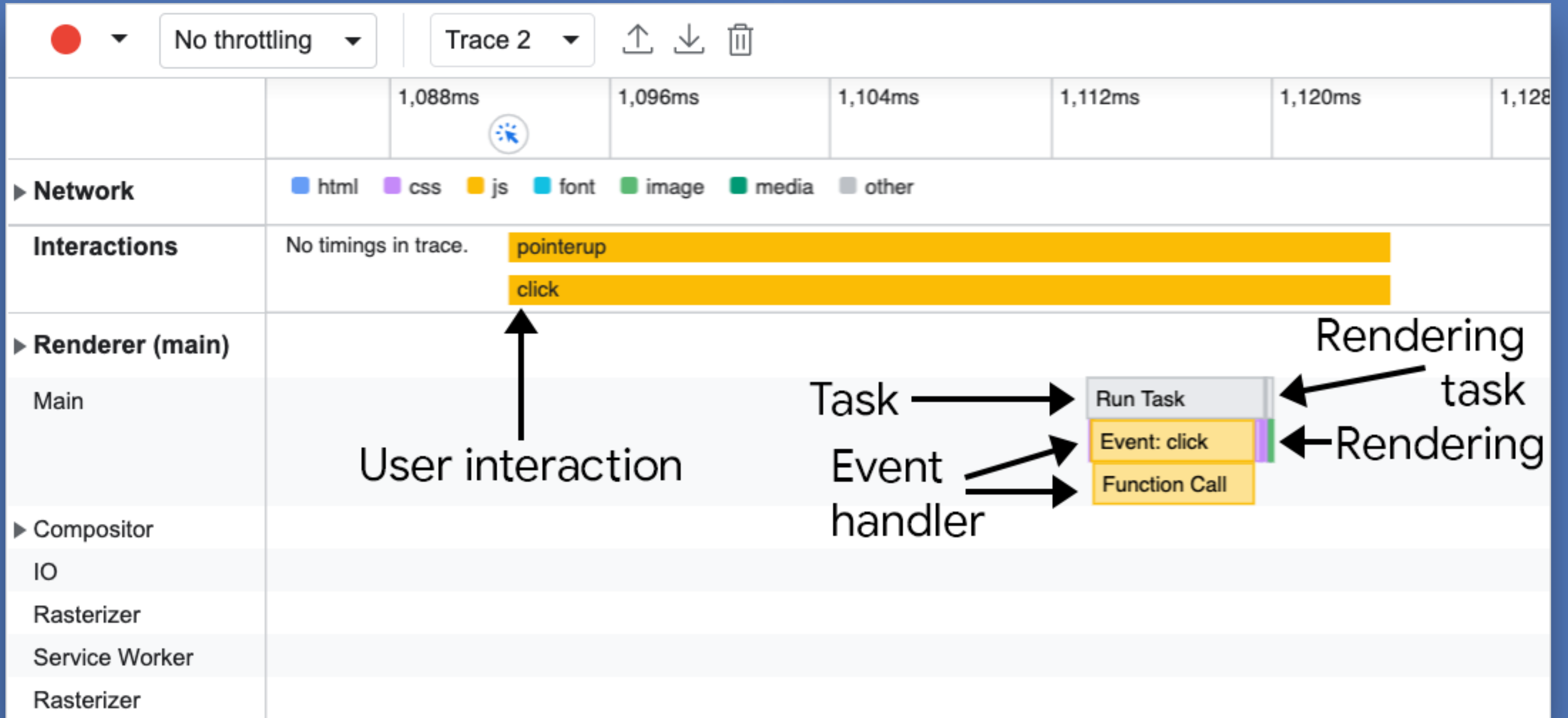
Lab tools

Interactions

Interaction type:click id:8047

▼ Main —





Optimize!



[Home](#) > [All articles](#)

Optimize long tasks

You've been told: "don't block the main thread" and "break up your long tasks", but what does it mean to do those things? Find out in this article.

Sep 30, 2022



Jeremy Wagner

[Twitter](#)

[GitHub](#)

[Homepage](#)

On this page

What is a task?

What is the main thread?

Task management strategies

Manually defer code execution

Use `async/await` to create yield points

Yield only when necessary

Gaps in current APIs

A dedicated scheduler API

Built-in yield with continuation

Conclusion

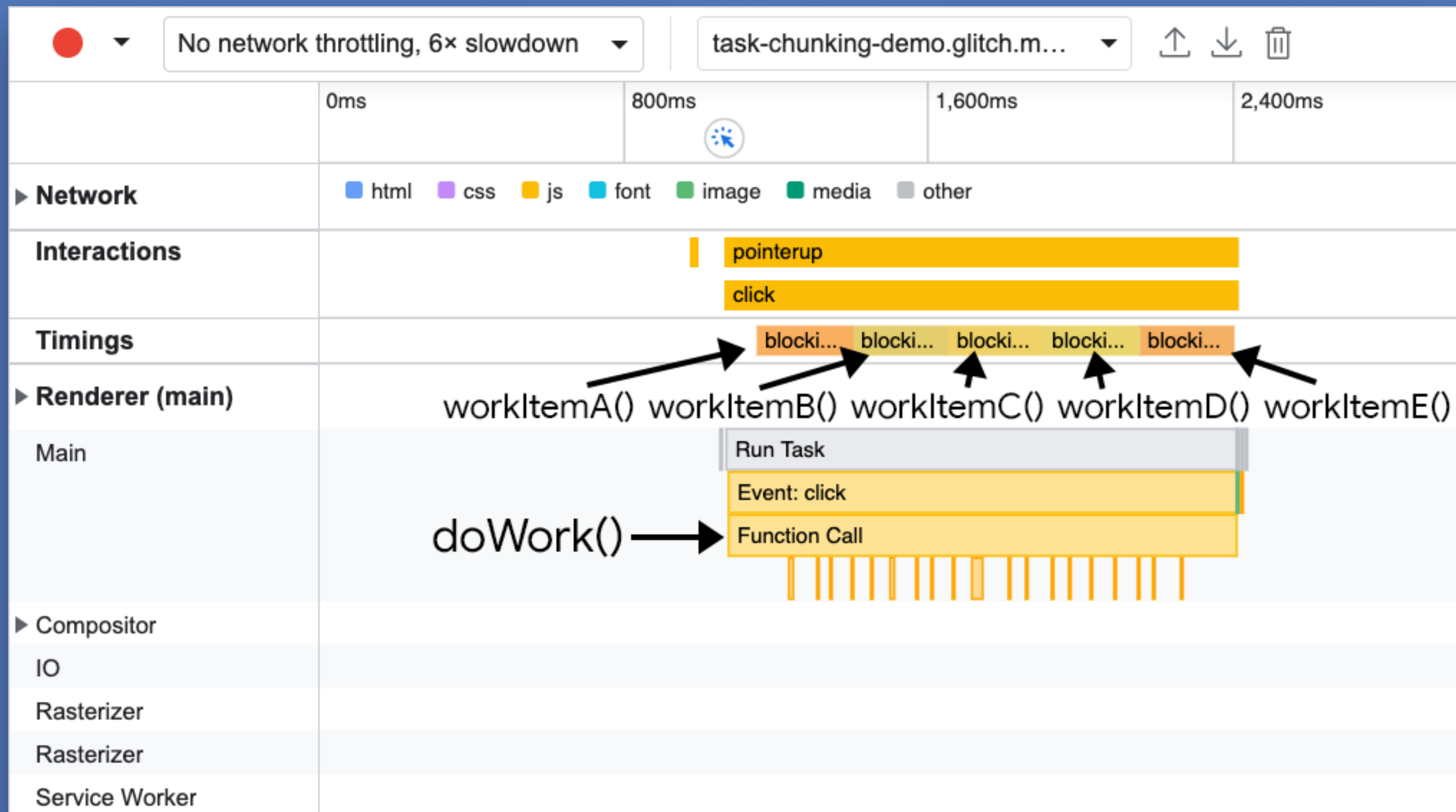
🔗 SHARE

✉ SUBSCRIBE

web.dev/optimize-long-tasks

If you read lots of stuff about web performance, the advice for keeping your JavaScript

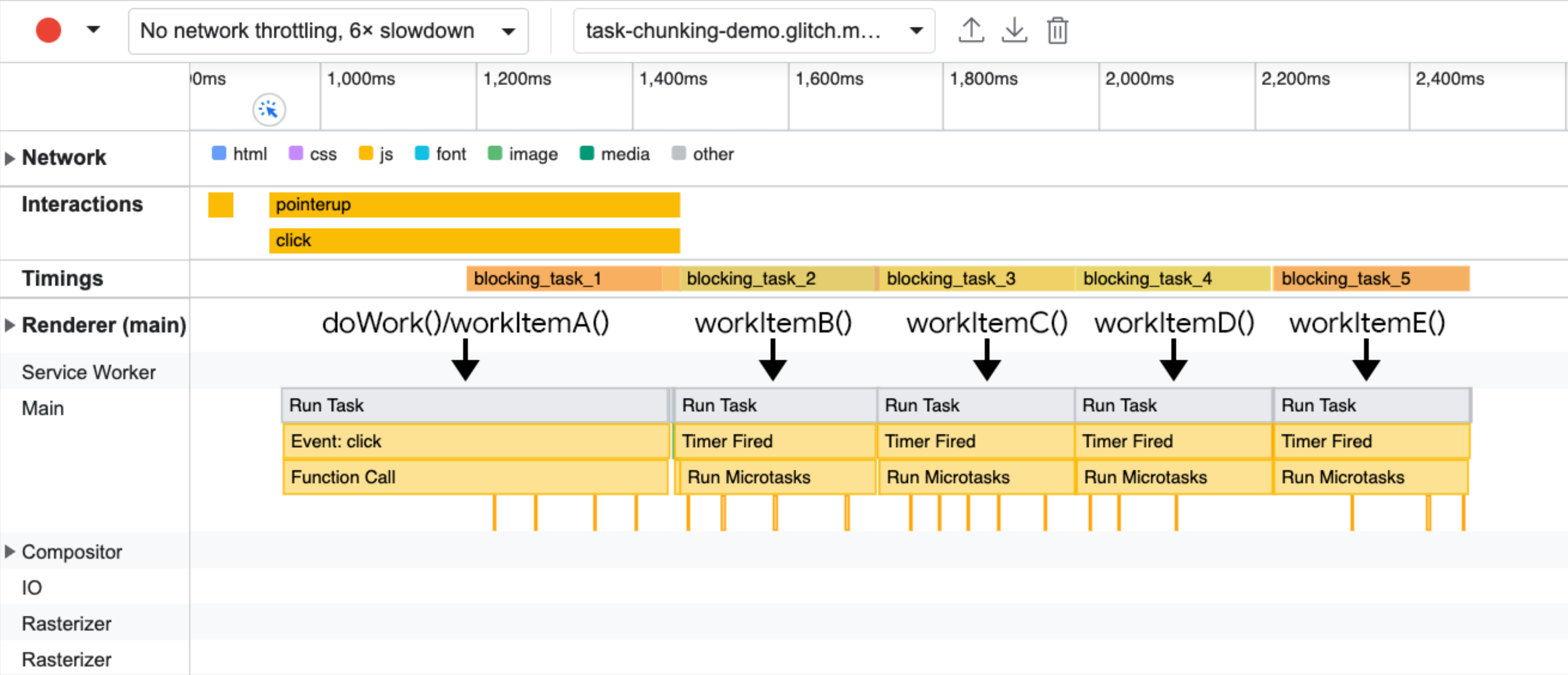

```
function doWork () {  
    workItemA();  
    workItemB();  
    workItemC();  
    workItemD();  
    workItemE();  
}
```



```
function yield () {  
  return new Promise(resolve => {  
    resolve(setTimeout);  
  });  
}
```



```
async function doWork () {  
  workItemA();  
  await yield();  
  workItemB();  
  await yield();  
  workItemC();  
  await yield();  
  workItemD();  
  await yield();  
  workItemE();  
}
```

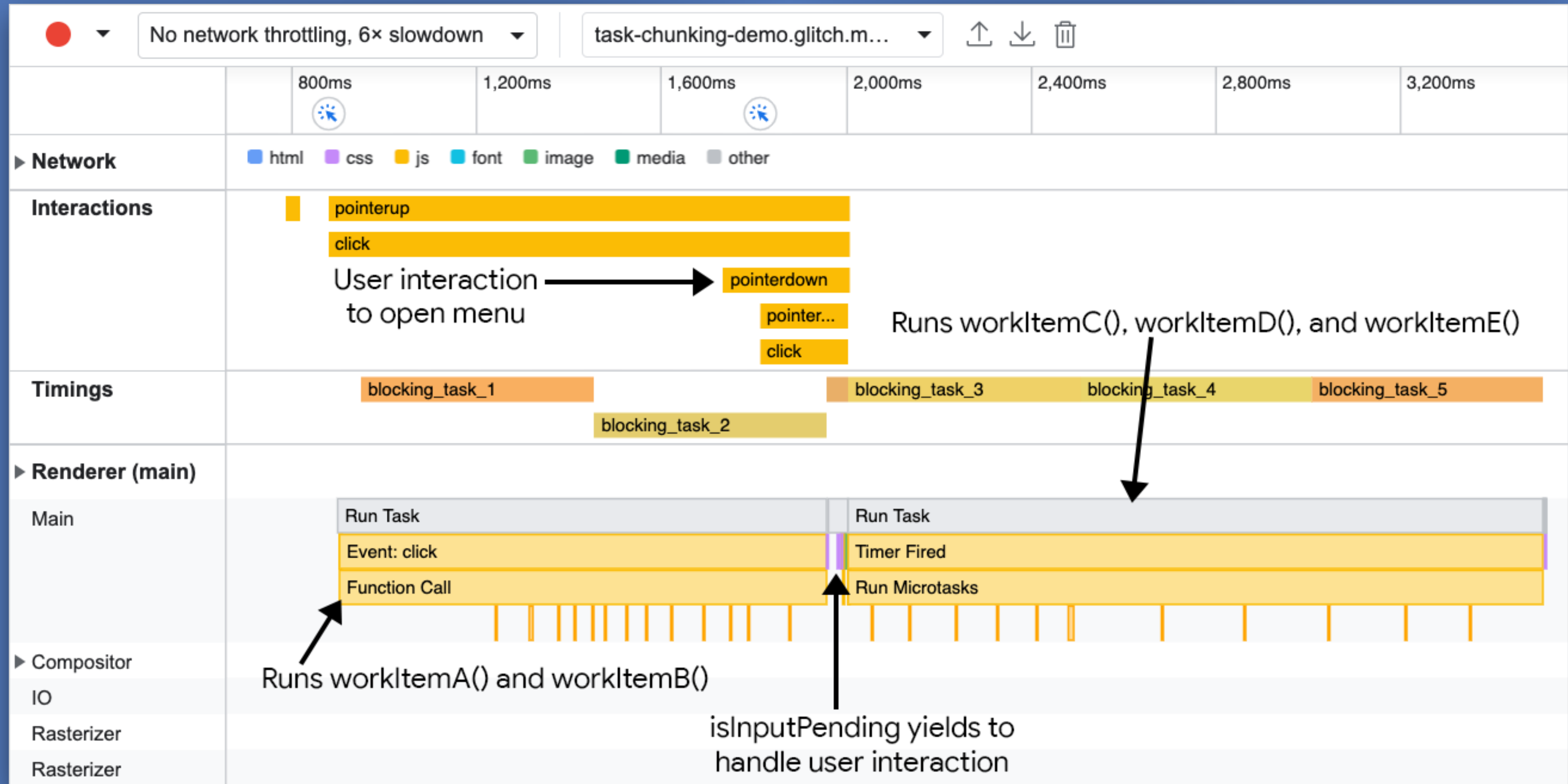


```
async function doWork () {  
  workItemA();  
  await scheduler.yield();  
  workItemB();  
  await scheduler.yield();  
  workItemC();  
  await scheduler.yield();  
  workItemD();  
  await scheduler.yield();  
  workItemE();  
}
```

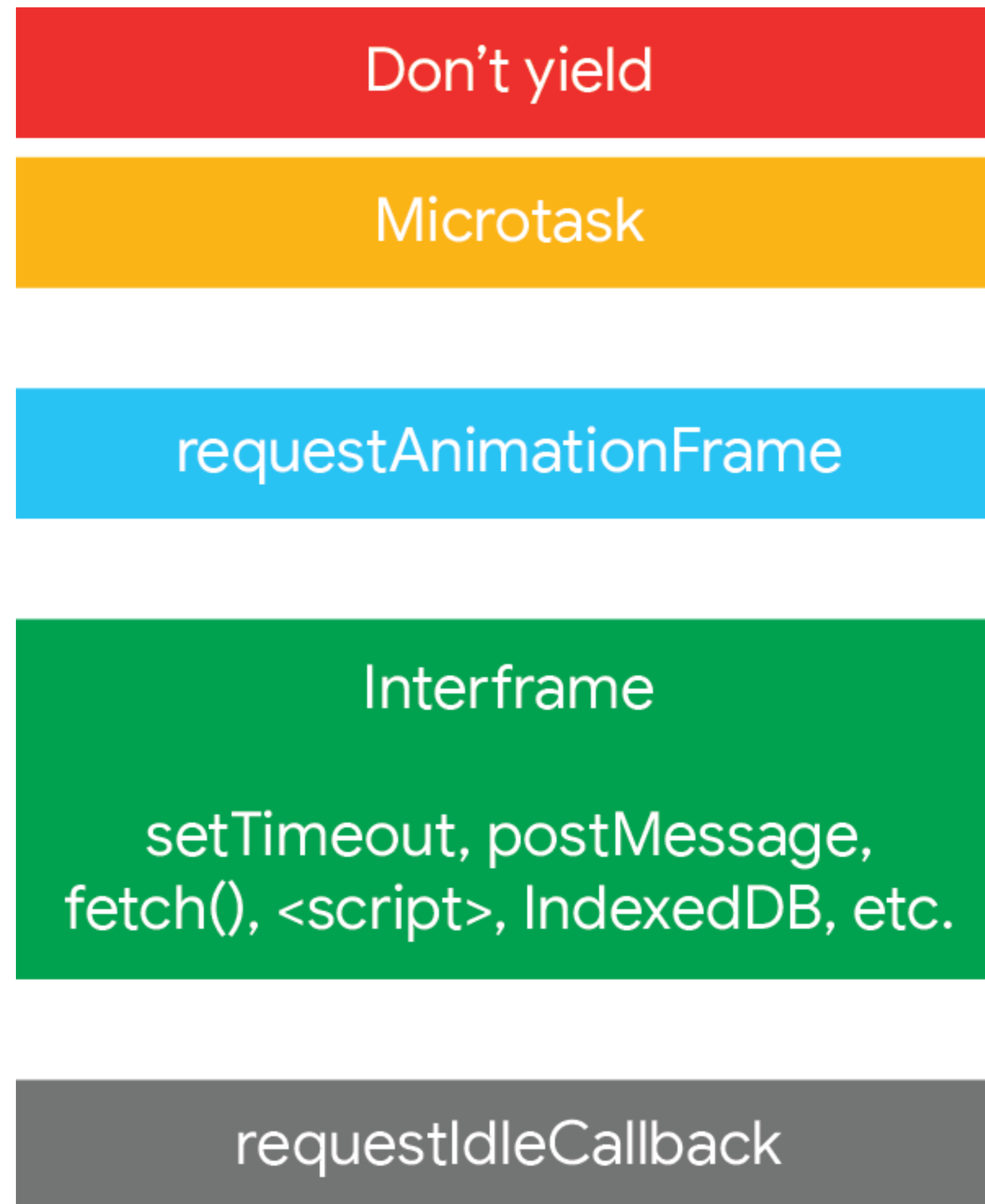

You still need to do as
little work as possible.

Yield to user input.

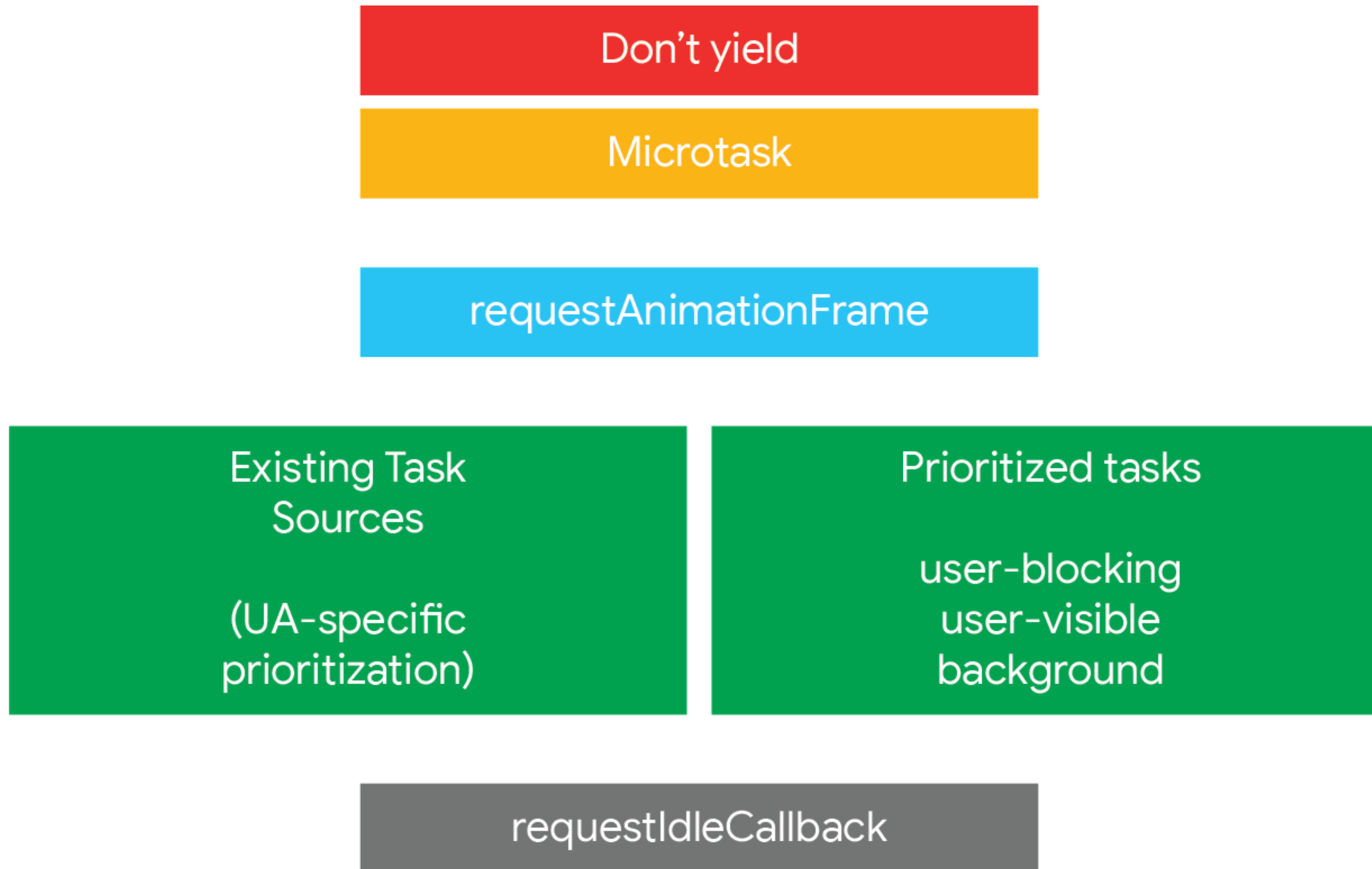
```
async function doWork () {  
  // A task queue of functions  
  const tasks = [  
    workItemA,  
    workItemB,  
    workItemC,  
    workItemD,  
    workItemE  
  ];  
  
  while (tasks.length > 0) {  
    // Optional chaining operator used here helps to avoid  
    // errors in browsers that don't support `isInputPending`:  
    if (navigator?.scheduling?.isInputPending()) {  
      // There's a pending user input. Yield here:  
      await yield();  
    } else {  
      // Shift the the task out of the queue:  
      const task = tasks.shift();  
  
      // Run the task:  
      task();  
    }  
  }  
}
```



Prioritize work.



← Browser-specific
prioritization




```
// Create a new task scheduler to schedule tasks to run at low priority:
const {signal} = new TaskController({
  priority: "background" // "background" === low priority
});

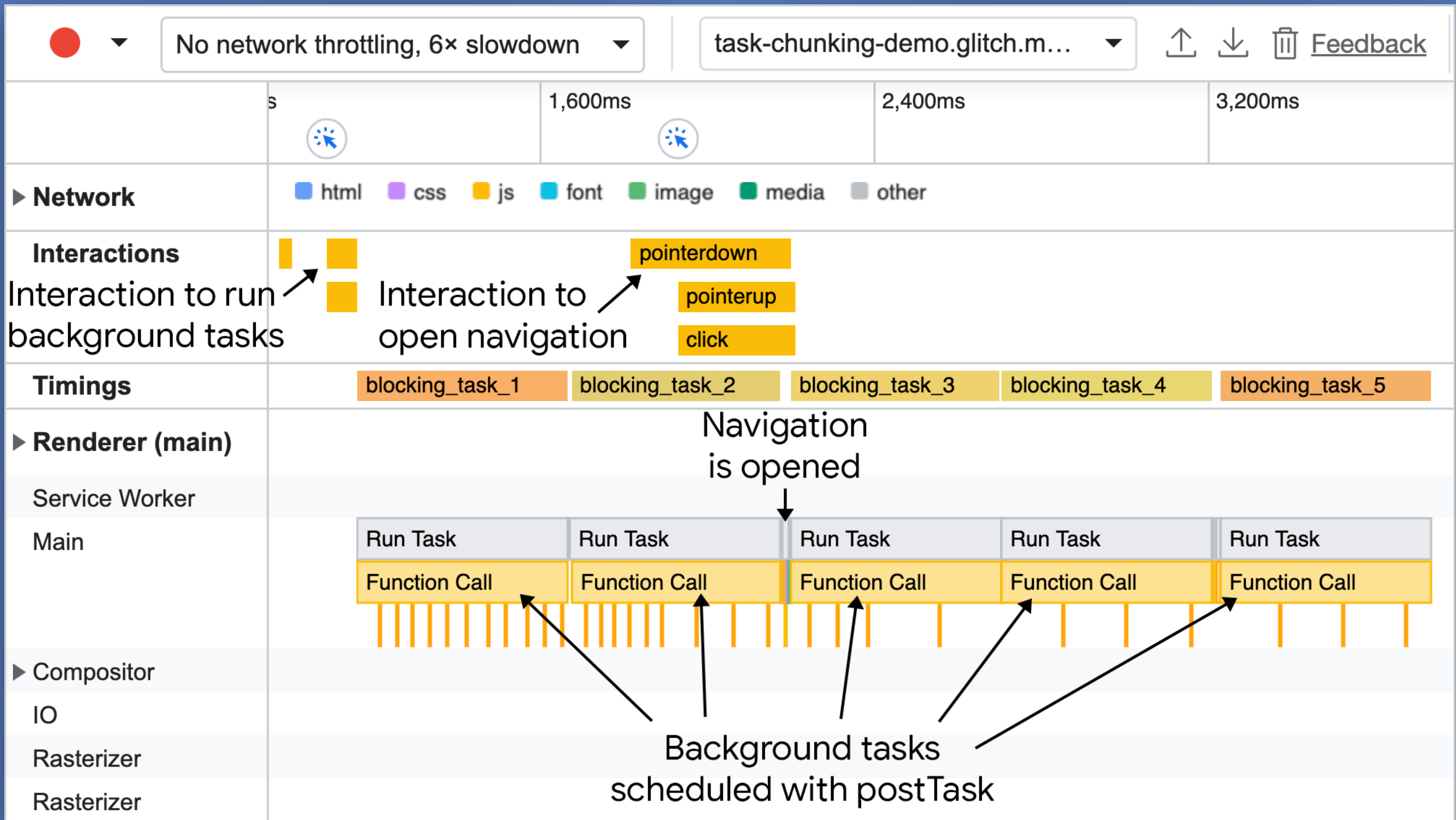
async function doWork () {
  // Schedule all tasks to run at low priority:
  await scheduler.postTask(workItemA, {
    signal
  });

  await scheduler.postTask(workItemB, {
    signal
  });

  await scheduler.postTask(workItemC, {
    signal
  });

  await scheduler.postTask(workItemD, {
    signal
  });

  await scheduler.postTask(workItemE, {
    signal
  });
};
```

INP is evolving.



Thank you for your time.

Jeremy Wagner – j1wagner.net – @ma1chata – PrDC Regina