INP – Yield Patterns to keep the UI smooth

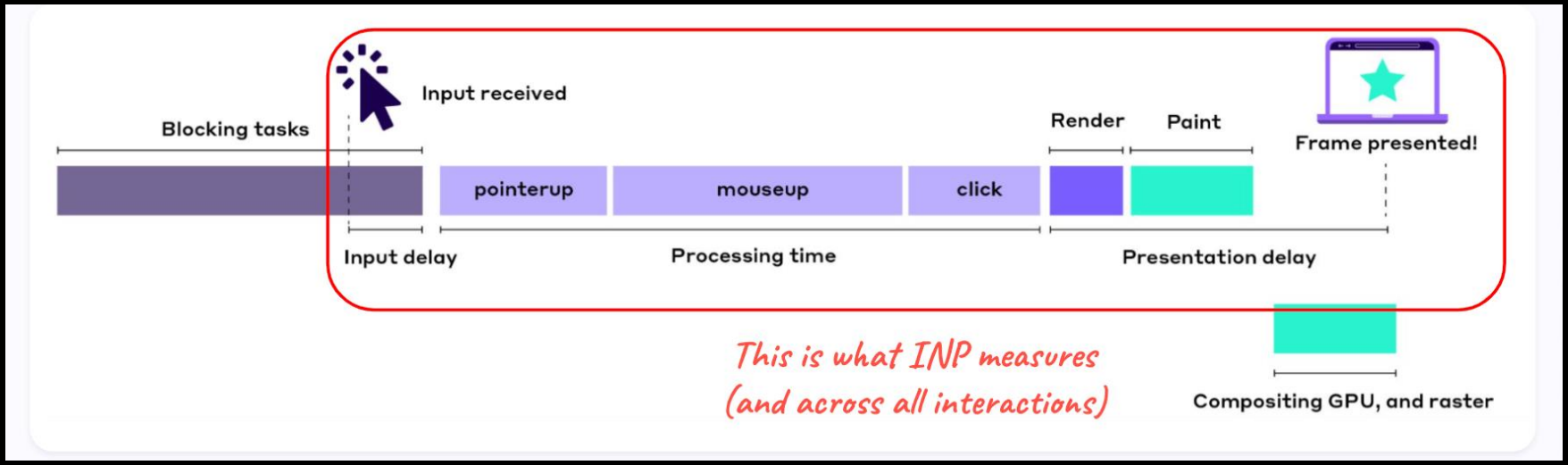
Jacob Groß | 11th September 2024 | kurtextrem.de

Who am I?

- Senior Performance Engineer @ Framer (you might know us from “Framer Motion”)
- I work on Core Web Vitals & other performance related topics
- Participant in W3C WebPerfWG calls, if you have webperf topics, come chat with me

- I like making things **fast & accessible** for all users of the internet

What's Interaction-to-Next-Paint (INP)?



Input delay: Main-thread activity blocking event handler processing

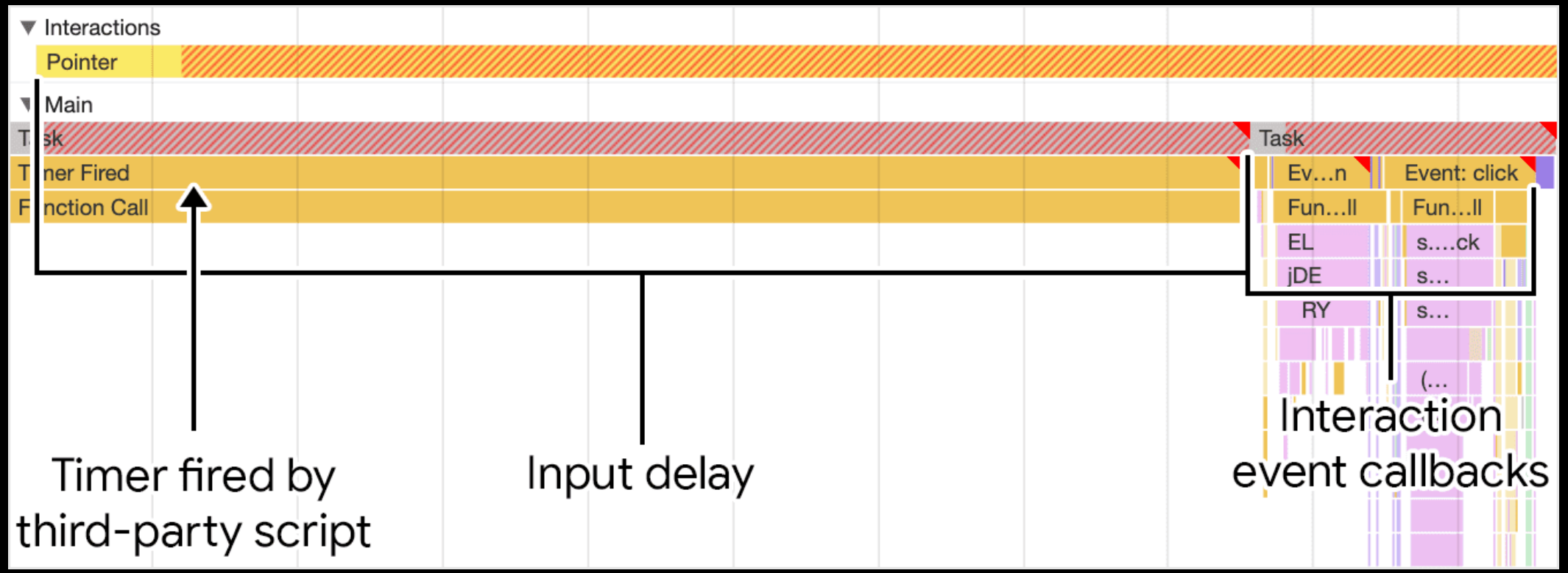
Processing time: Your (or 3rd-party) event handlers

Presentation delay: Browser rendering & maybe prev. unfinished user interactions (keydown -> keyup)

It measures **clicks, taps, key presses**

https://nitropack.io/blog/post/optimize-interaction-to-next-paint-webinar

What’s Interaction-to-Next-Paint (INP)?



https://web.dev/articles/optimize-input-delay

What's Interaction-to-Next-Paint (INP)?

- A browser tries to render new frames continuously (if needed)
- 'fast' is achieved by **keeping the main thread free**

Key here:

- INP is about giving the browser the **opportunity** to paint a frame – but doesn't have to for great INP*
- Keeping the main thread free makes the UI feel 'smooth'(er) **=> improves UX**

INP is how fast the browser *could* paint the next frame after user input

* A browser might skip "Paint" due to optimizations – those count for INP, too

https://web.dev/articles/inp

If your summary now is:

“if you put less work on the main thread,
you improve UX & INP?”

Absolutely.

That’s why INP is a good approximation of felt UX.

Smooth webpages leave a great impression.

An INP eye-opener:

Doing nothing in response to user input allows the browser to paint fast => great INP…

…great INP in that case does not *guarantee* great UX!

Sometimes no feedback can be valid feedback.

Giving (no/any) feedback **fast** is what creates great UX.

What feedback fits your UI? Ask your UI/UX team.

To summarize INP:

Time between User Interaction to Next Paint (Opportunity)

Slower than 500ms -> Bad UX!

Slower than 200ms -> Could be better UX!

Faster than 200ms -> Good INP! **Good UX?**

In reality, we‘re targeting **100ms**:

- **100ms** is the threshold where users are **not able to perceive the delay** ⬆️ ⬆️ **we want this**
- 200ms was picked because of the broad landscape of (mobile) devices – reaching 100ms is hard, but it’s the ✨ **WOW** ✨ goal.

https://www.speedcurve.com/blog/psychology-site-speed/

https://www.nngroup.com/articles/response-times-3-important-limits/

https://web.dev/articles/rail

https://github.com/w3c/event-timing/issues/118#issuecomment-1108700093

<convincing arguments for your boss.slide>

What aspects of page experience are used in ranking?

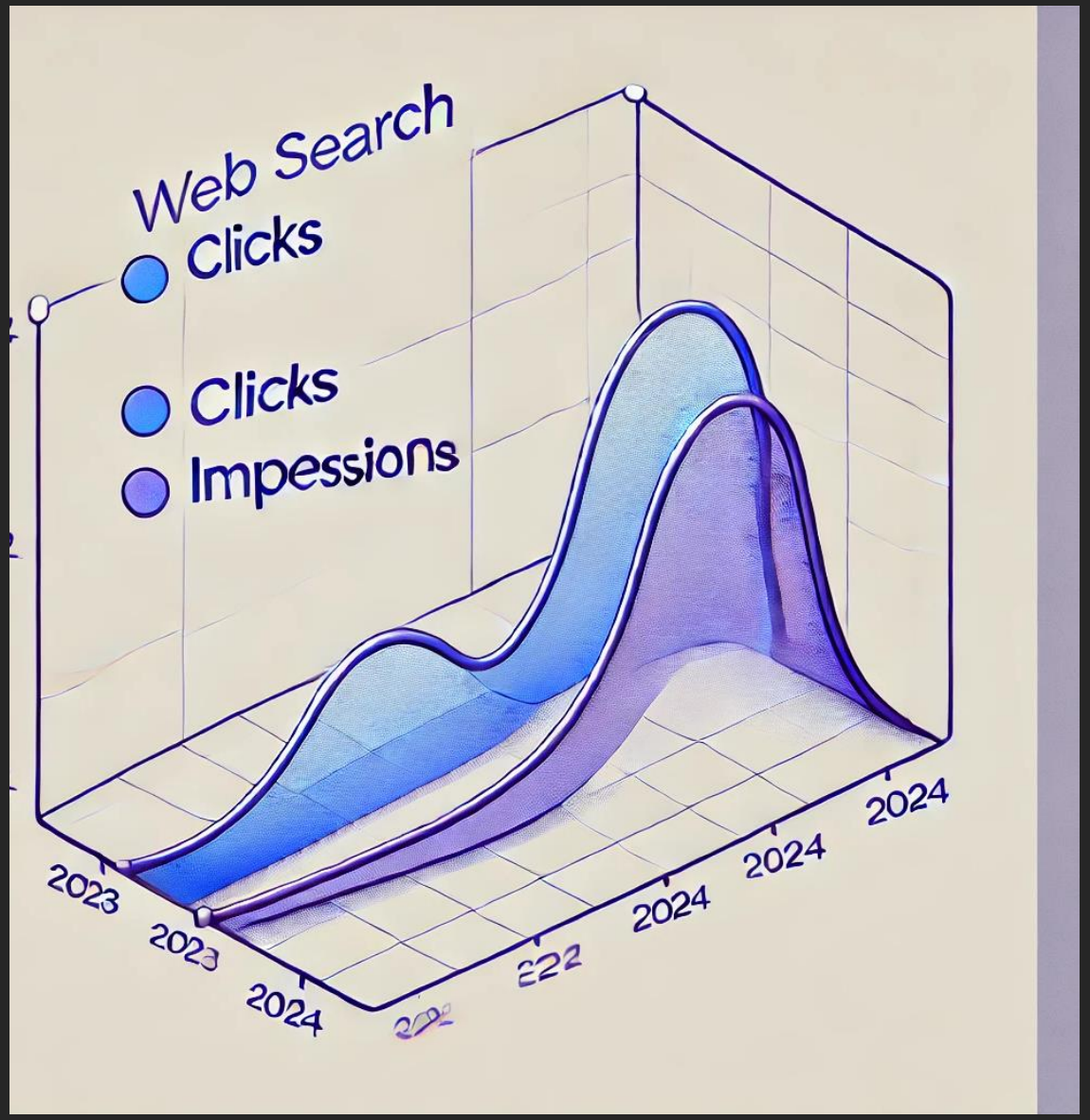
Core Web Vitals are used by our ranking systems. We recommend site owners achieve good Core Web Vitals for success with Search and to ensure a great user experience generally. Keep in mind that getting good results in reports like Search Console's Core Web Vitals report or third-party tools doesn't guarantee that your pages will rank at the top of Google Search results; there's more to great page experience than Core Web Vitals scores alone. These scores are meant to help you to improve your site for your users overall, and trying to get a perfect score just for SEO reasons may not be the best use of your time.

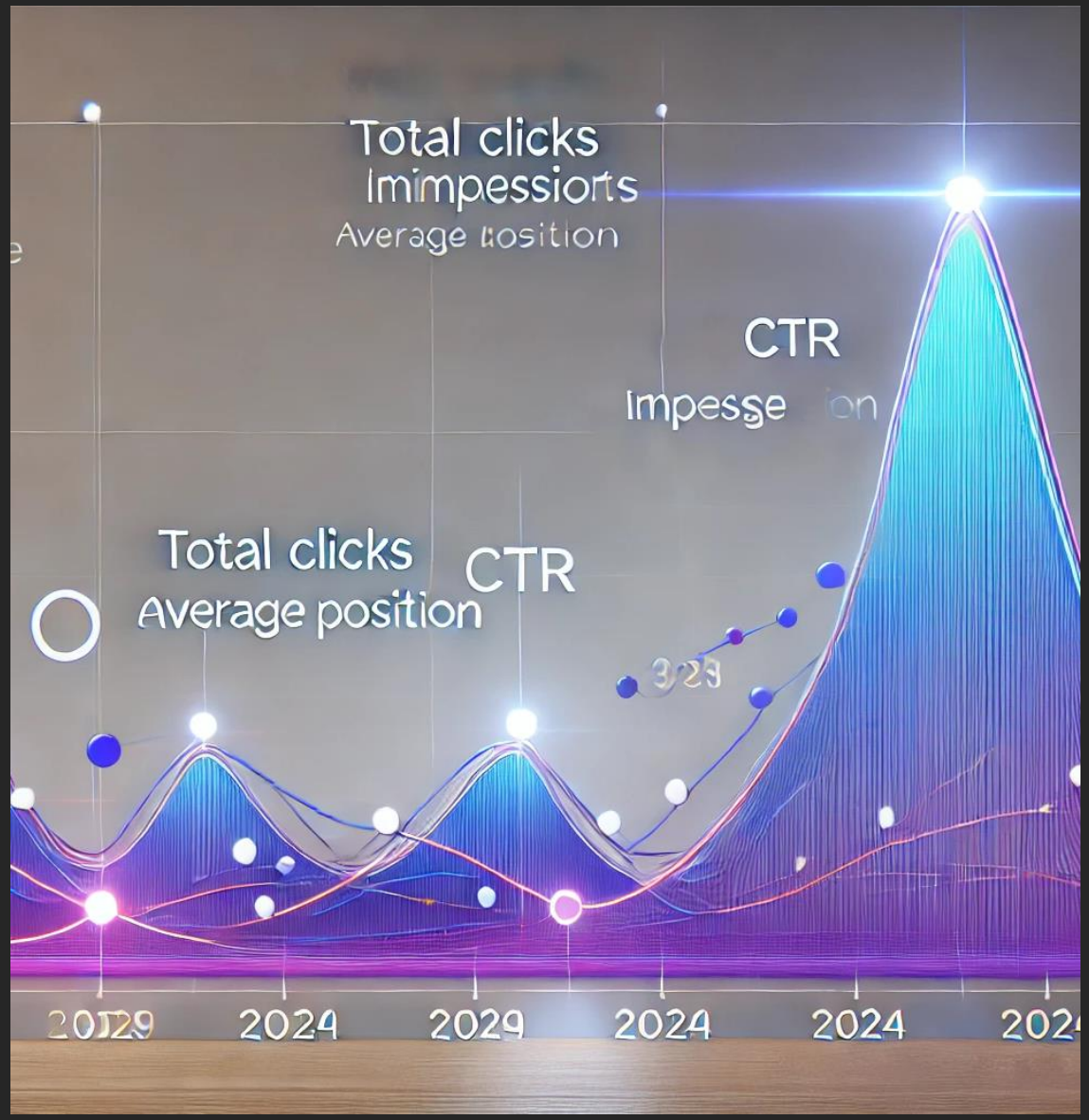
- Google

https://developers.google.com/search/docs/appearance/page-experience

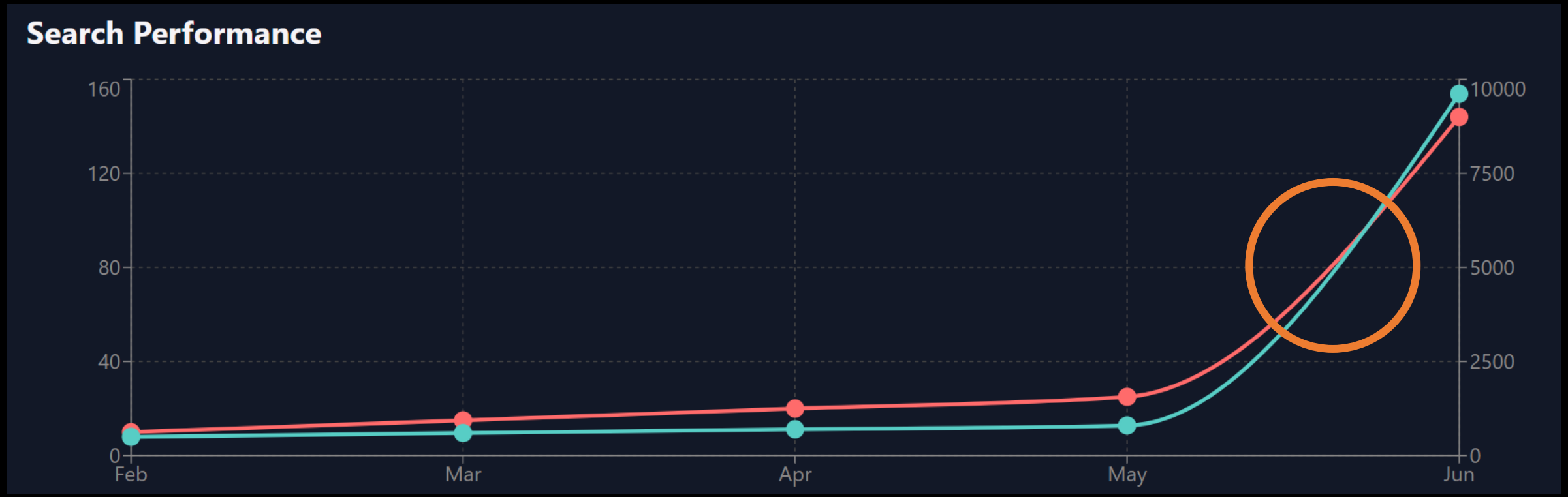
Case Studies

Here’s how AI ‘reimagined’ a real graph I put into ChatGPT:


Web Search
Clicks
Clicks
Impressions
2023
2024
2024
2024
2024


Total clicks
Average position
CTR
Total clicks
Average position
CTR
2024
2024
2024
2024
2024

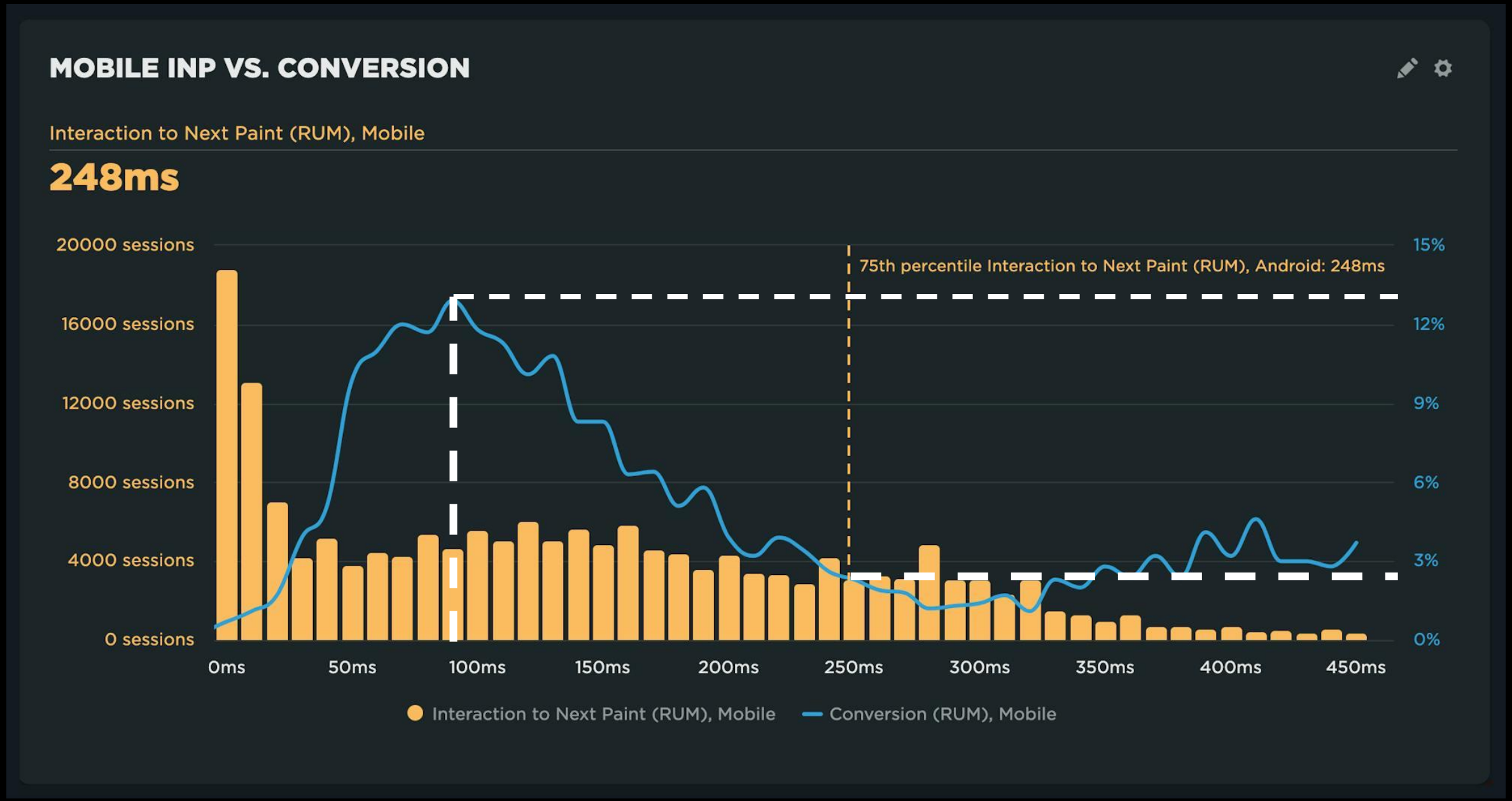
We shipped big INP improvements in…?



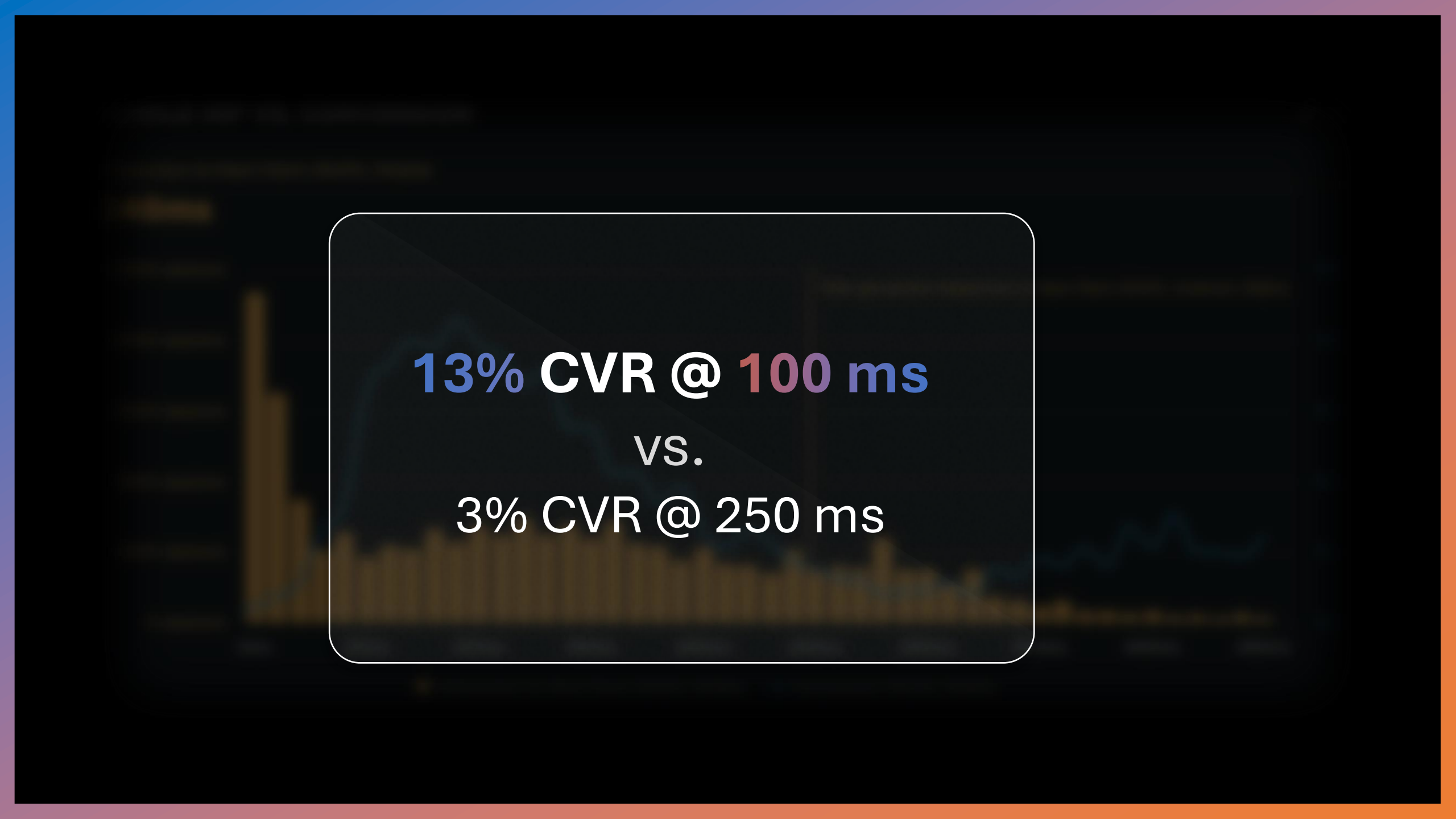
Customer reported 85% faster INP
> +100% search impression
> +100% search clicks

Graph matches the scale of the impact we've seen.

Thanks to friends from the RUMvision team, who helped w/ initial triage.



https://www.speedcurve.com/blog/core-web-vitals-inp-mobile/


13% CVR @ 100 ms
vs.
3% CVR @ 250 ms

Ok Jacob. Sold.
What next?

a) Understand how to find your culprits

Guides: web.dev, DebugBear, RUMvision, Speedcurve, Speedkit, NitroPack, …

Field data: Calibre, RUMvision, Treo, WebPerformance Report, RequestMetrics, …

b) Apply fixes

1. **Prioritize** visible UI work and defer invisible tasks (such as analytics)
2. **Yield** to the main thread before, or frequently during expensive functions
3. **Finish** execution faster – improve runtime efficiency, abort previous tasks and run less JS overall

Ship less JS: https://www.youtube.com/watch?v=f5felHJiACE

If it doesn't provide feedback,

it is *not urgent* to the user.

We run it after *yield*ing.

kurtextrem.de/INP.pdf

Yield Patterns to keep the UI smooth

Yield Patterns to keep the UI smooth

- Yielding is a way of saying ‘continue this later’
- Simplest way: **setTimeout(fn)**

Example:

```
<button onClick={() => {
  updateUI()
  setTimeout(sendAnalytics)
}}>
```

```
setTimeout(() => alert('talk done')
```

As simple as that.

prompt(“Questions?“)

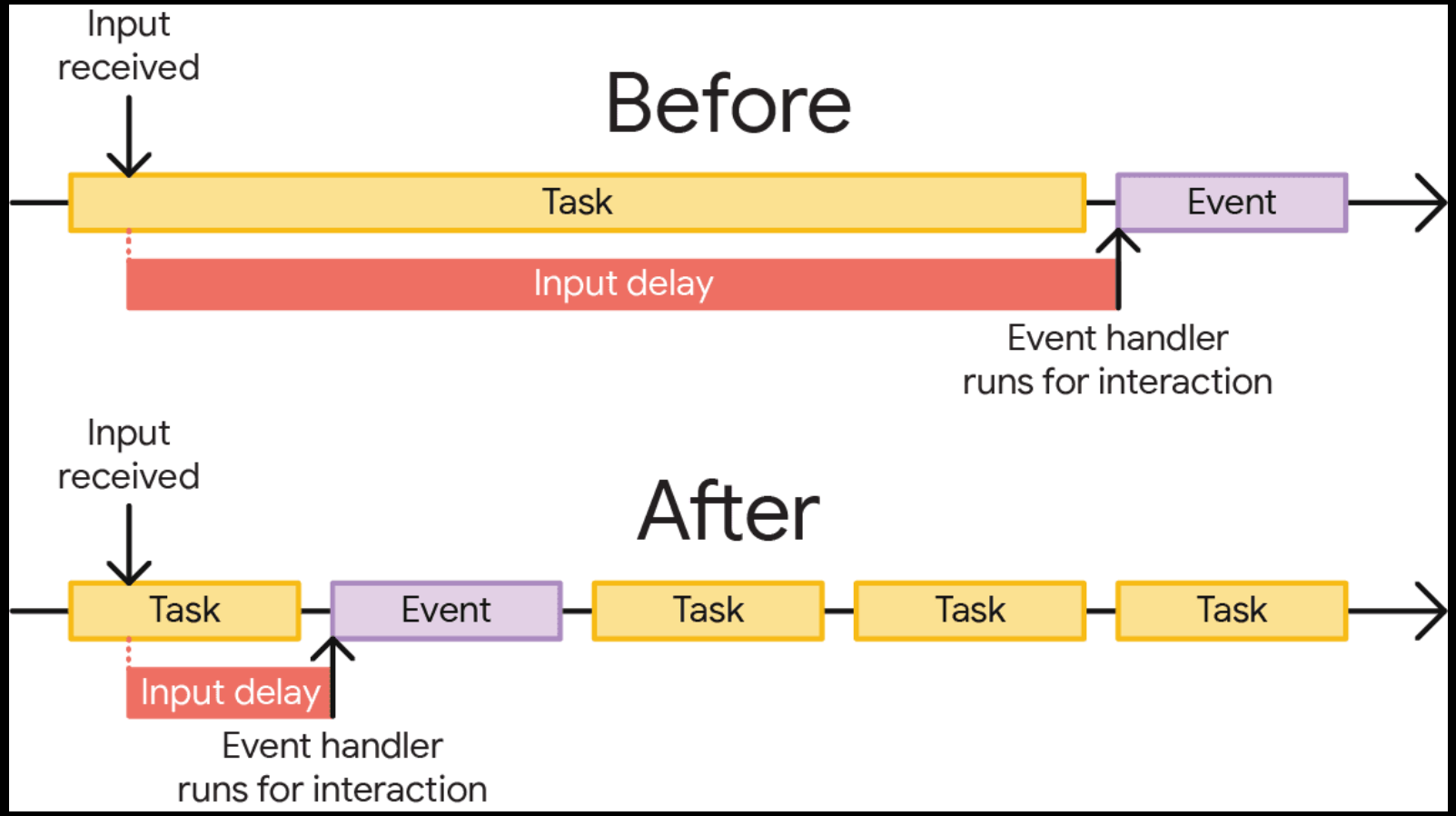
The end. Follow me on X, @kurtextrem, I (re)tweet webperf stuff

Just kidding. setTimeout is *one* way of yielding

(by the way, if you trigger an alert/prompt, it doesn't count as blocking for INP.
Do with that info whatever you feel like doing...)

https://www.debugbear.com/docs/metrics/interaction-to-next-paint#do-alert-and-similar-dialogs-contribute-to-inp

...and a way of breaking up long tasks



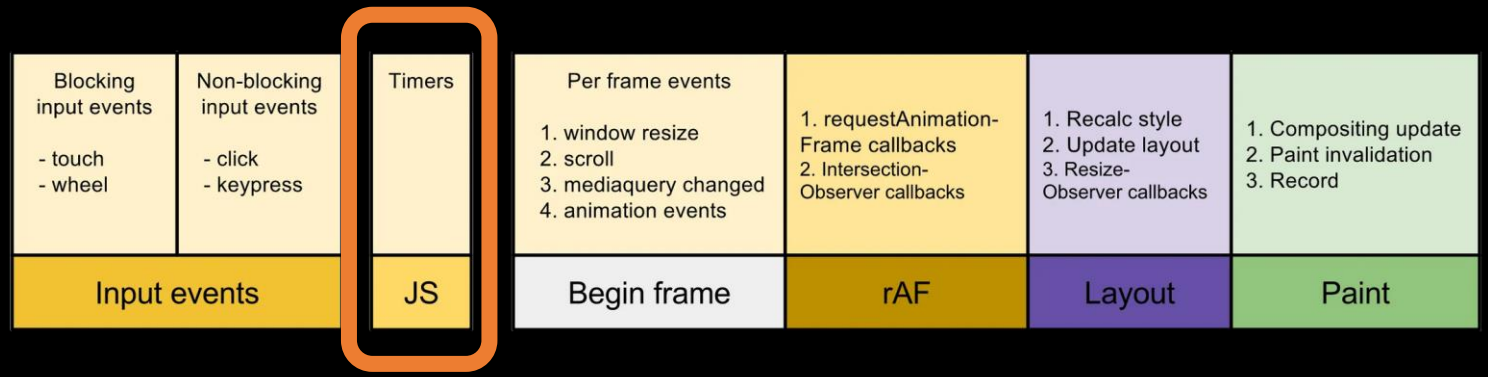
https://web.dev/articles/optimize-input-delay

Used since 2007.

https://x.com/jsmarr/status/1801000265811730807

Life of a (browser) frame

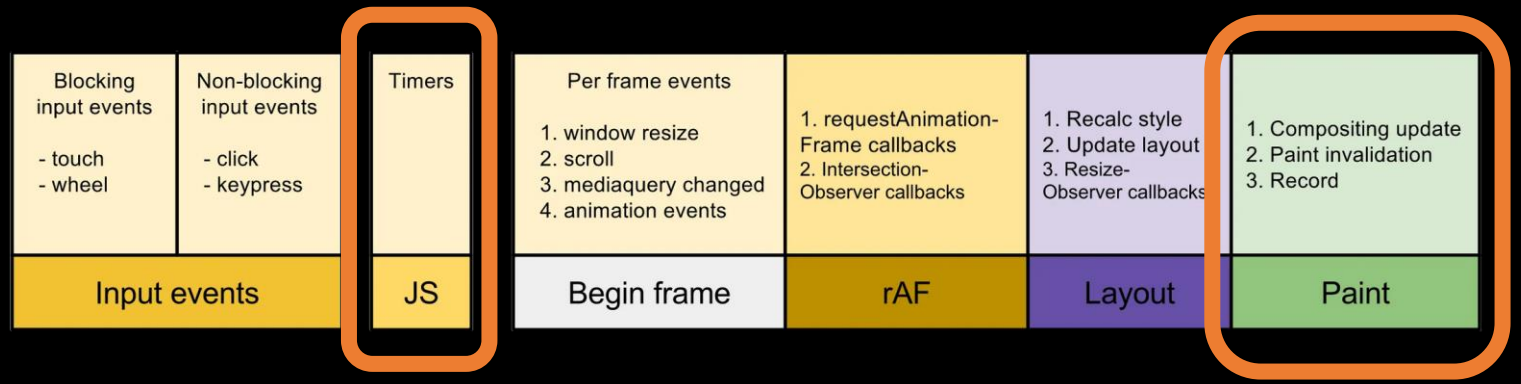
Life of a (browser) frame



This is when setTimeout runs

Based on https://medium.com/@paul_irish/requestanimationframe-scheduling-for-nerds-9c57f7438ef4

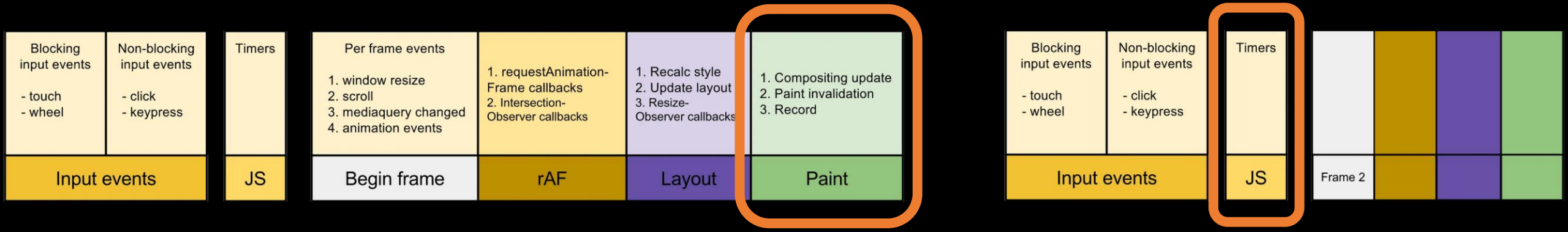
Life of a (browser) frame



It *might* run before a paint

https://github.com/w3c/long-animation-frames/issues/13#issuecomment-2142366987

Life of a (browser) frame



It *might* run after a paint

https://github.com/w3c/long-animation-frames/issues/13#issuecomment-2142366987

How do we ensure it runs after paint?

- INP was about: making sure a browser has the opportunity to paint
- setTimeout alone does not *guarantee* it

npm package introduced by Vercel's CTO Malte Ubl:

```
function interactionResponse() {
  return new Promise((resolve) => {
    setTimeout(resolve, 100) // Fallback for the case where the animation frame never fires.
    requestAnimationFrame(() => {
      setTimeout(resolve)
    })
  })
}
```

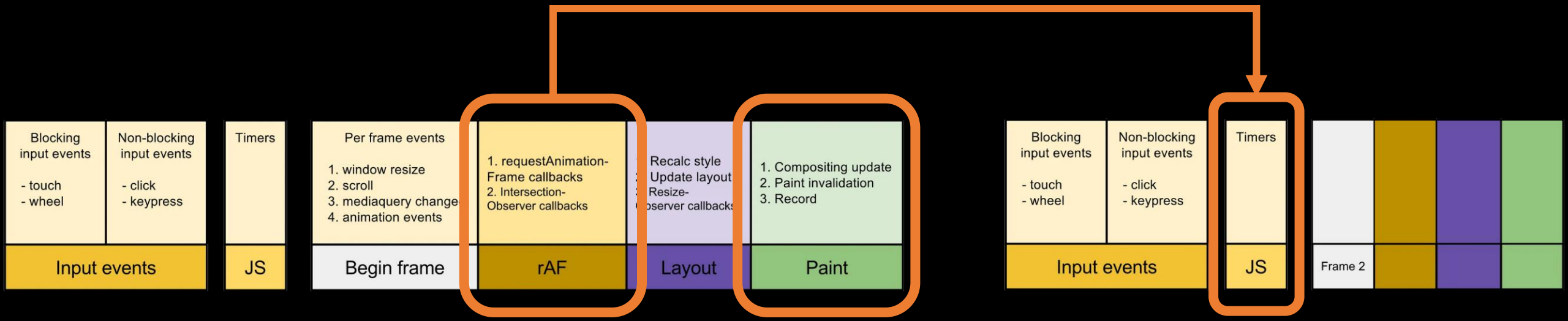
How do we ensure it runs after paint?

npm package introduced by Vercel's CTO Malte Ubl:

```
function interactionResponse() {
  return new Promise((resolve) => {
    setTimeout(resolve, 100) // Fallback for the case where the animation frame never fires.
    requestAnimationFrame(() => {
      setTimeout(resolve)
    })
  })
}
```

```
someButton.addEventListener('click', async (event) => {
  doSomeImportantUIWork(event) // ⬅️ 1. give user meaningful feedback
  await interactionResponse()  // ⬅️ 2. yield by awaiting the paint
  performance.mark('paint-done')
})
```

Run after paint: await-interaction-response



It runs after paint

Run after paint: await-interaction-response

Pros:

- Guarantees better INP processing duration as it runs after paint
- Could be used to batch DOM writes (in the rAF) and DOM reads (after the setTimeout) as runtime optimization

Cons:

- Might not run if a user is about to leave the page
- rAF could be throttled (if tab goes to background)
- setTimeout suffers from ‘queue jumping’ (= might run *somewhen*)

https://kurtextrem.de/posts/improve-inp

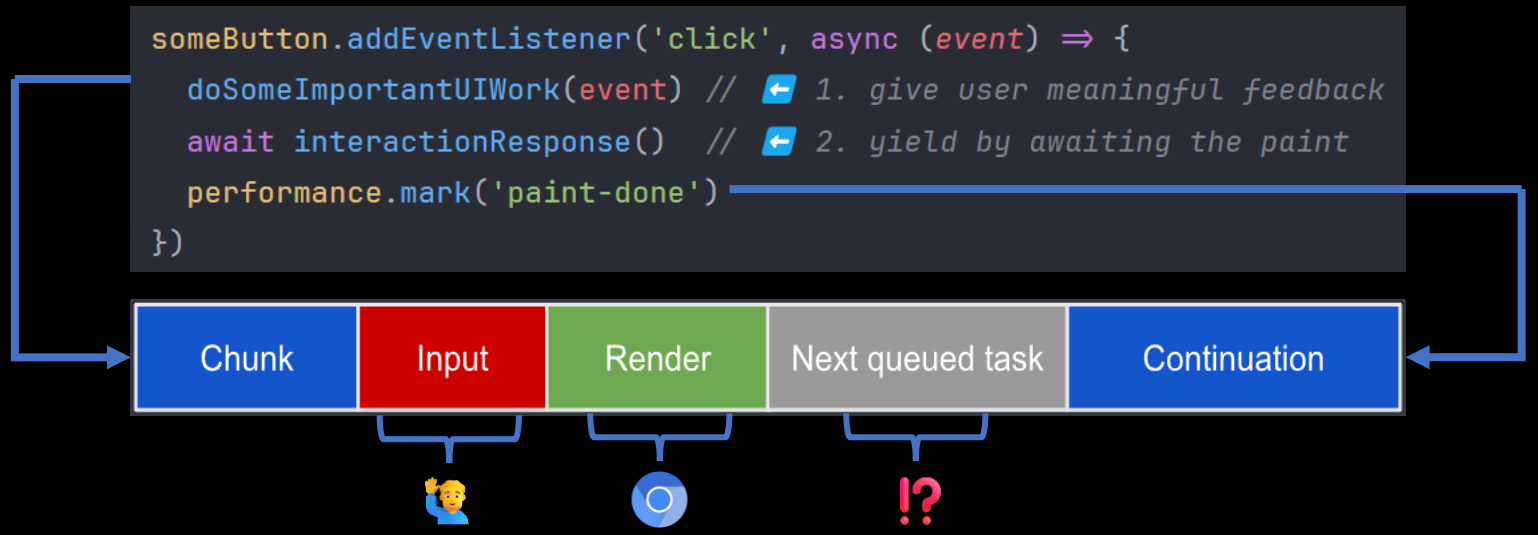
Queue Jumping?

Queue Jumping

- Almost anything running in the browser is a ‘task’ running in a queue
- Tasks can have different priorities
- setTimeout basically puts the callback to the **end of the queue**

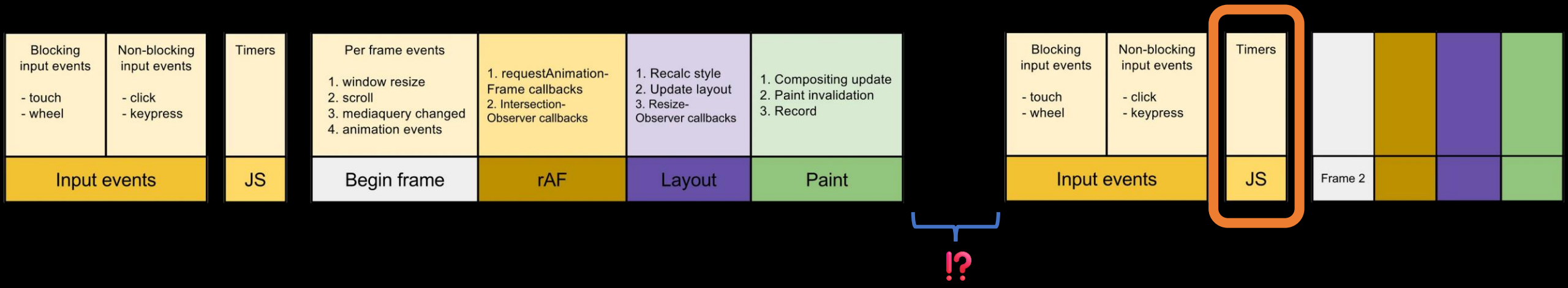
- What if some 3rd-party dependency or analytics scripts you don’t control have scheduled 100 setTimeout’s before?
- **Your setTimeout will run last**, so after 100 other tasks. Yikes.

Queue Jumping



“paint-done” *might* not be accurate

Run after paint: await-interaction-response



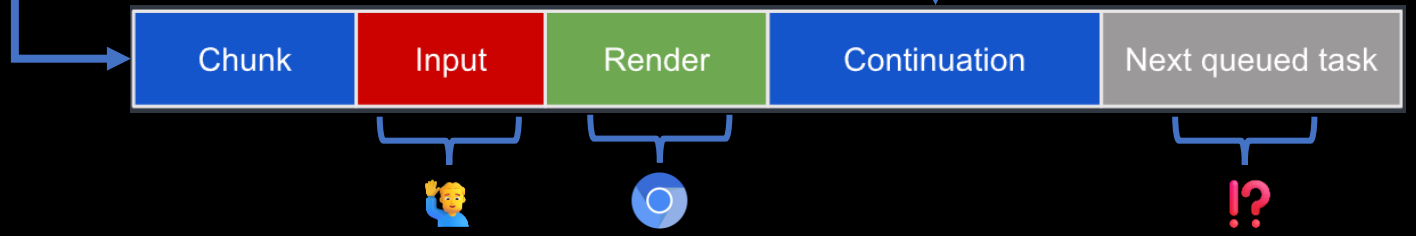
It runs after paint
(somewhen)

Jacob, are you telling me nothing to fix this has been released for over 17 years?

scheduler.yield

scheduler.yield

```
someButton.addEventListener('click', async (event) => {
  doSomeImportantUIWork(event) // ↩ 1. give user meaningful feedback
  await scheduler.yield()  // ↩ 2. yield
  performance.mark('after-yield')
})
```



General Purpose: yieldToMain (scheduler.yield)

```
function yieldToMain(options) {
    if ("scheduler" in window) {
      if ("yield" in scheduler) return scheduler.yield(options)
      if ("postTask" in scheduler) return scheduler.postTask(() => {}, options)
    }

    // `setTimeout` could suffer from being delayed for longer - so for browsers not supporting yield,
    // we guarantee execution for high priority actions, but it doesn't yield as trade-off.
    if (options?.priority === 'user-blocking') return Promise.resolve()
    return new Promise((resolve) => setTimeout(resolve))
}
```

- Try **scheduler.yield** first – Chromium 129+ only
- Fallback to next modern API: **scheduler.postTask** – Chromium 94+
- **setTimeout** or at the end of the microtasks queue for high priority

General Purpose: yieldToMain (scheduler.yield)

- Yields to the main thread without awaiting paints
- *Can* improve INP, but does not guarantee

```
function yieldToMain(options) {
    if ("scheduler" in window) {
      if ("yield" in scheduler) return scheduler.yield(options)
      if ("postTask" in scheduler) return scheduler.postTask(() => {}, options)
    }

    // `setTimeout` could suffer from being delayed for longer - so for browsers not supporting yield,
    // we guarantee execution for high priority actions, but it doesn't yield as trade-off.
    if (options?.priority === 'user-blocking') return Promise.resolve()
    return new Promise((resolve) => setTimeout(resolve))
}
```

General Purpose: yieldToMain (scheduler.yield)

Pros:

- Great for slicing longer tasks into smaller ones as it doesn’t wait for paints all the time
- ‘continuation’

Cons:

- we need to know how long / expensive our tasks are to make sure it improves INP
- Limited browser support
- ‘continuation’ is slightly more confusing than ‘await paint & run code’

Calling hobby cooks, we now have 2 recipes. What do we do?

await-interaction-response

yieldToMain *in* interactionResponse

```
// A little helper which yields before running the function
async function yieldBefore(fn, options) {
  await yieldToMain(options)
  return fn()
}

function interactionResponse() {
  return new Promise((resolve) => {
    setTimeout(resolve, 100) // Fallback for the case where the animation frame never fires.
    requestAnimationFrame(() => {
      yieldBefore(resolve)
    })
  })
}
```

This is how Framer runs interactionResponse on prod (btw don't remove the setTimeout fallback)

This us?

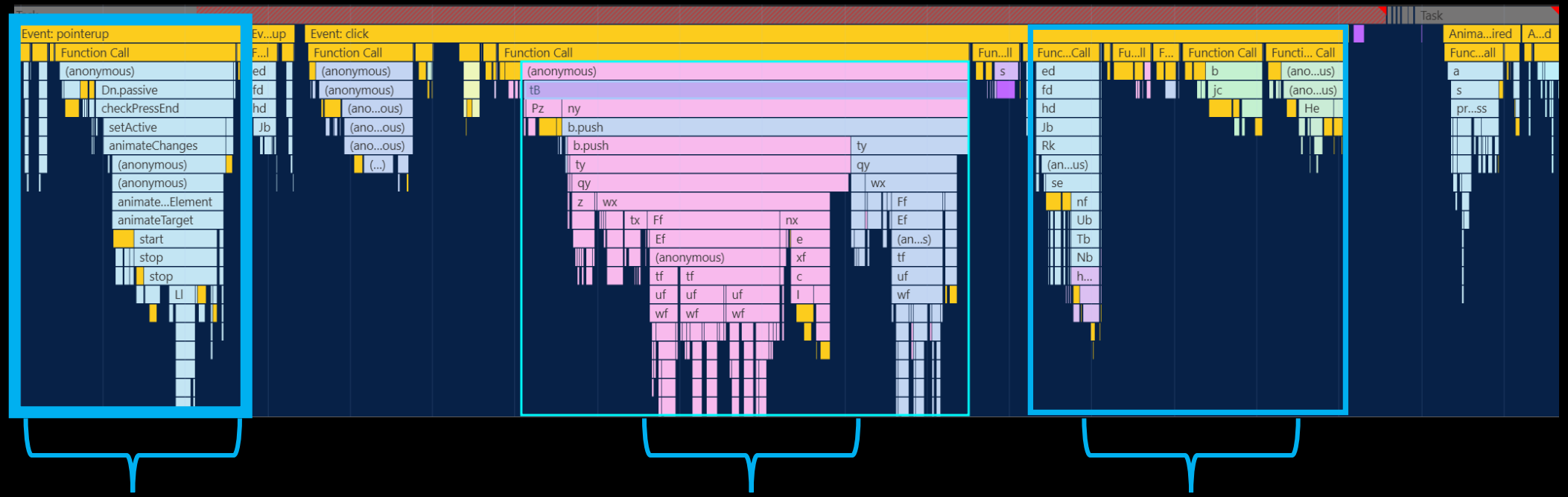
Before we continue with more
browser scheduling stuff,
let’s take a look at a **practical example**.

Cookie Banners

We use cookies to enhance your experience, analyze site traffic and deliver personalized content. Read our Cookie Policy.

Reject Accept

"Accept" (or "Reject") usually triggers lots of 3rd-party code



UI updates

Stuff the user never sees

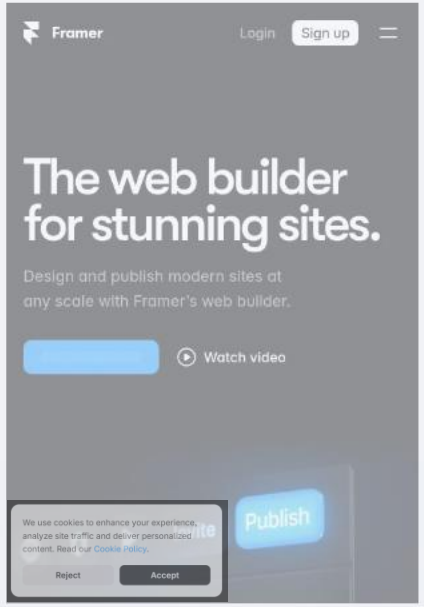
Stuff the user never sees

Before yielding on “Accept”

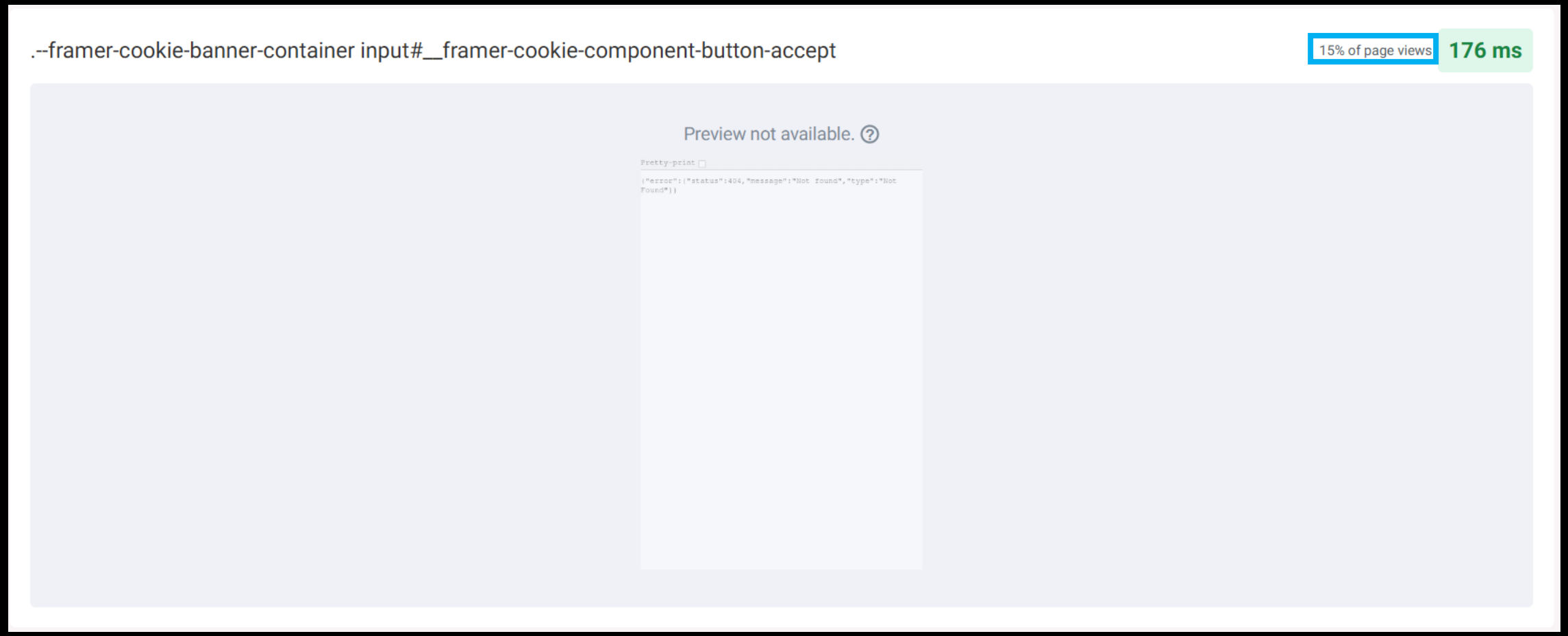
.--framer-cookie-banner-container input#__framer-cookie-component-button-accept

12% of page views 272 ms

Preview not available.



After yielding: **100ms** faster on p75!



Copy Contents

```
        if (onConsentChange) {
            yieldBeforeCb(() =>
                onConsentChange({ isInEU, consent: consent.modes })
            )
        }
    }, [consent.modes])

    function handleDismiss() {

        consent.dismiss()
        setIsOpen(false)

        // Fire callback
        if (onDismiss) {
            yieldBeforeCb(() => onDismiss({ isInEU }))
        }
    }

    function handleAcceptAll() {

        consent.acceptAll()
        setIsOpen(false)

        // Fire callback
        if (onAccept) {
            yieldBeforeCb(() => onAccept({ isInEU }))
        }
    }

    function handleRejectAll() {

        consent.rejectAll()
        setIsOpen(false)

        // Fire callback
        if (onReject) {
            yieldBeforeCb(() => onReject({ isInEU }))
        }
    }

    function handleAcceptCurrent() {

        consent.acceptCurrent()
        setIsOpen(false)
```

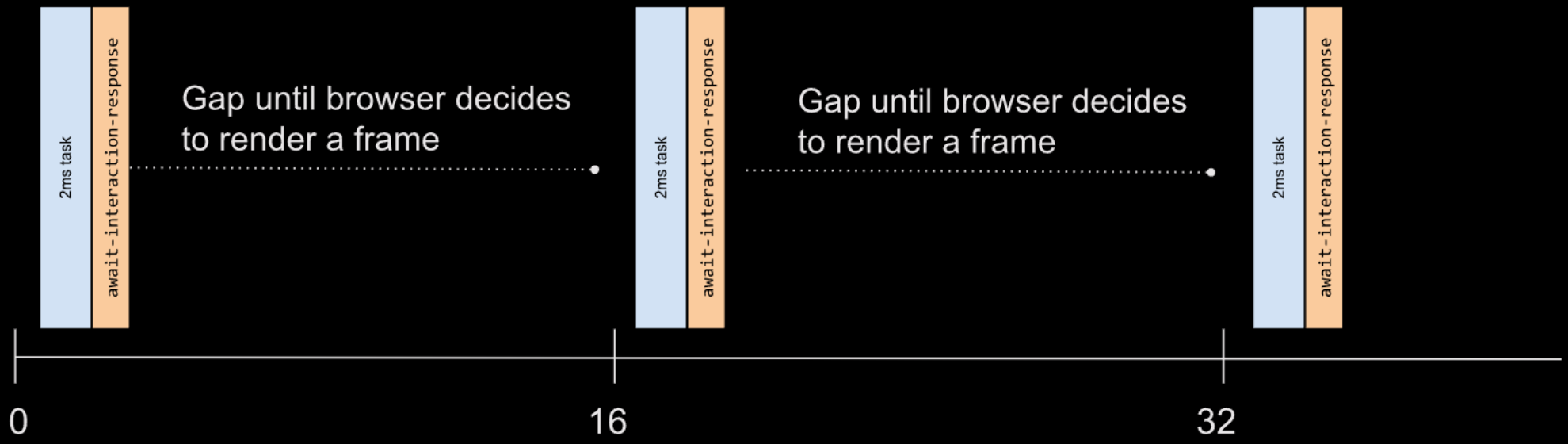
```
        if (onConsentChange) {
            yieldBeforeCb(() =>
                onConsentChange({ isInEU, consent: consent.modes })
            )
        }
    }, [consent.modes])

    async function handleDismiss() {
        await interactionResponse()

        consent.dismiss()
        setIsOpen(false)

        // Fire callback
        if (onDismiss) {
            yieldBeforeCb(() => onDismiss({ isInEU }))
        }

    tion handleAcceptAll() {
        interactionResponse()

    t.acceptAll()
    pen(false)

    e callback
    Accept) {
    eldBeforeCb(() => onAccept({ isInEU }))
    }

    async function handleRejectAll() {
        await interactionResponse()

        consent.rejectAll()
        setIsOpen(false)

        // Fire callback
        if (onReject) {
            yieldBeforeCb(() => onReject({ isInEU }))
        }
    }

    async function handleAcceptCurrent() {
        await interactionResponse()

        consent.acceptCurrent()
        setIsOpen(false)
```

Cookie Banners - Fix

- Yield (await paint) before running any CMP accept or reject callbacks
- Also give your CMP vendor a friendly reminder (because they also might run events you cannot change)
- (Basically) a one liner to copy & paste

When to use yieldToMain vs. await-interaction-response?

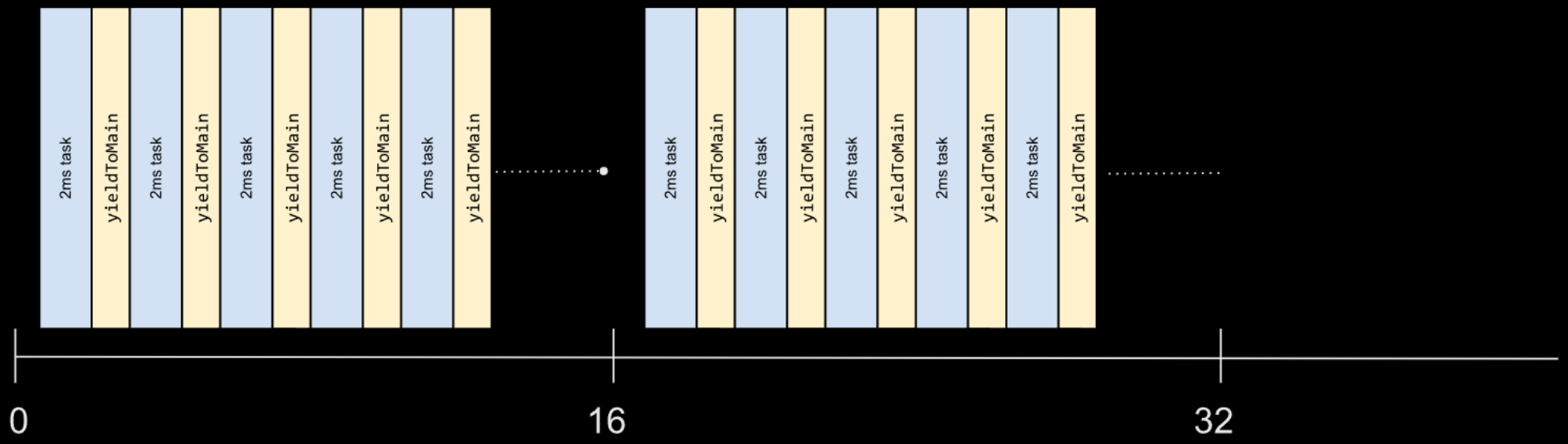
await-interaction-response



While waiting for paints, we do literally nothing.
Great for INP, inefficient for processing (e.g. data).

Images adopted from Barry Pollard

yieldToMain



Keeps the UI responsive & is fast for processing,
but might fail to improve INP if we don’t know the workload.
If we’re not careful, we might do much before yielding back to the main thread.

“I’d advocate for **splitting the tasks liberally** using something like **yieldToMain** at good yield points and then **letting the browser worry about what to schedule when**. In most cases that should strike a good balance between optimizing INP and also getting the work done, **without having to think too much about it**.”

- Barry Pollard, Google WebPerf Dev Advocate

await-interaction-response

(again?)

yieldToMain + interactionResponse

- Important takeaway: Both alone have their pros and cons…
- But in combination, we can achieve great things:

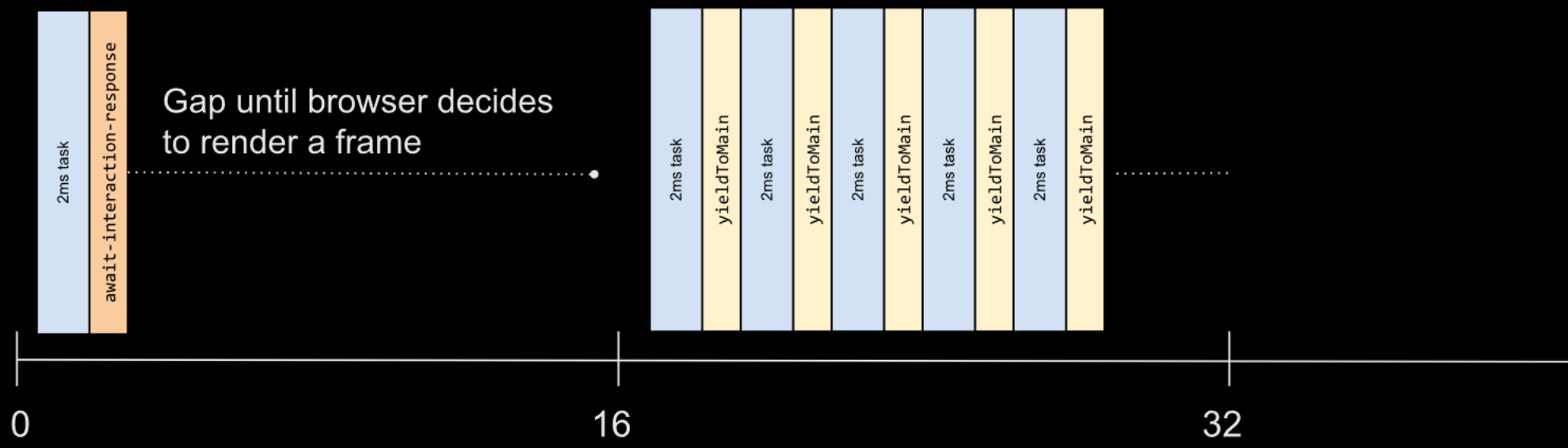
a) **Defer** non-critical UI updates until the next paint via **interactionResponse**

b) **Split** long tasks into smaller ones via **yieldToMain**

Goal: Keep tasks below 50ms.

The **trade-off to make** is either **faster results** or a more **responsive UI** (+ possibly better INP).

yieldToMain + interactionResponse



A DIY scheduler

```
// ⬇️ We introduce a 'limit' for 3., so the loop stays below ~50ms
const QUANTUM = 48; // ms, 3*16ms (3 frames on 60 hz)

async function searchFilterHandler(searchTerm: string, items: any[]) {
  // ⬇️ 1. Update UI by e.g. changing opacity of previous results
  updateUI()
  // ⬇️ 2. Await the next paint
  await interactionResponse()

  // ⬇️ Store timestamp+limit, so we know when to stop
  let deadline = performance.now() + QUANTUM

  const results = []
  for (const item of items) {
    // ⬇️ 3. Yield when we've crossed the deadline
    if (performance.now() ≥ deadline) {
      await yieldToMain()
      deadline = performance.now() + QUANTUM // update limit
    }

    // check if the item matches the search term
    if (itemMatchesSearch(item, searchTerm)) {
      results.push(result)
    }
  }

  return results
}
```

One more pattern:
yieldUnlessUrgent

Introduced by Google Engineer Philip Walton

```ts
/** A set to keep track of all unresolved yield promises */
const pendingResolvers = new Set()

/** Resolves all unresolved yield promises and clears the set. */
function resolvePendingPromises() {
  document.removeEventListener('visibilitychange', resolvePendingPromises)
  for (const resolve of pendingResolvers) resolve()
  pendingResolvers.clear()
}

/**
 * Returns a promise that, if the document is visible, will resolve in a new
 * task in the next frame. If the document is not visible (or changes to
 * hidden prior to the promise resolving), the promise is resolved immediately.
 * @return {Promise<void>}
 */
export function yieldUnlessUrgent() {
  return new Promise((resolve) => {
    pendingResolvers.add(resolve)
    if (document.visibilityState === 'visible') {
      document.addEventListener('visibilitychange', resolvePendingPromises)
      // 🔽 you could use `yieldBefore` too, depending on the task
      return requestAnimationFrame(() => setTimeout(() => {
        pendingResolvers.delete(resolve)
        resolve()
      }))
    }
    // Still here? Resolve immediately.
    resolvePendingPromises()
  })
}
```

Exit Event Handlers: yieldUnlessUrgent

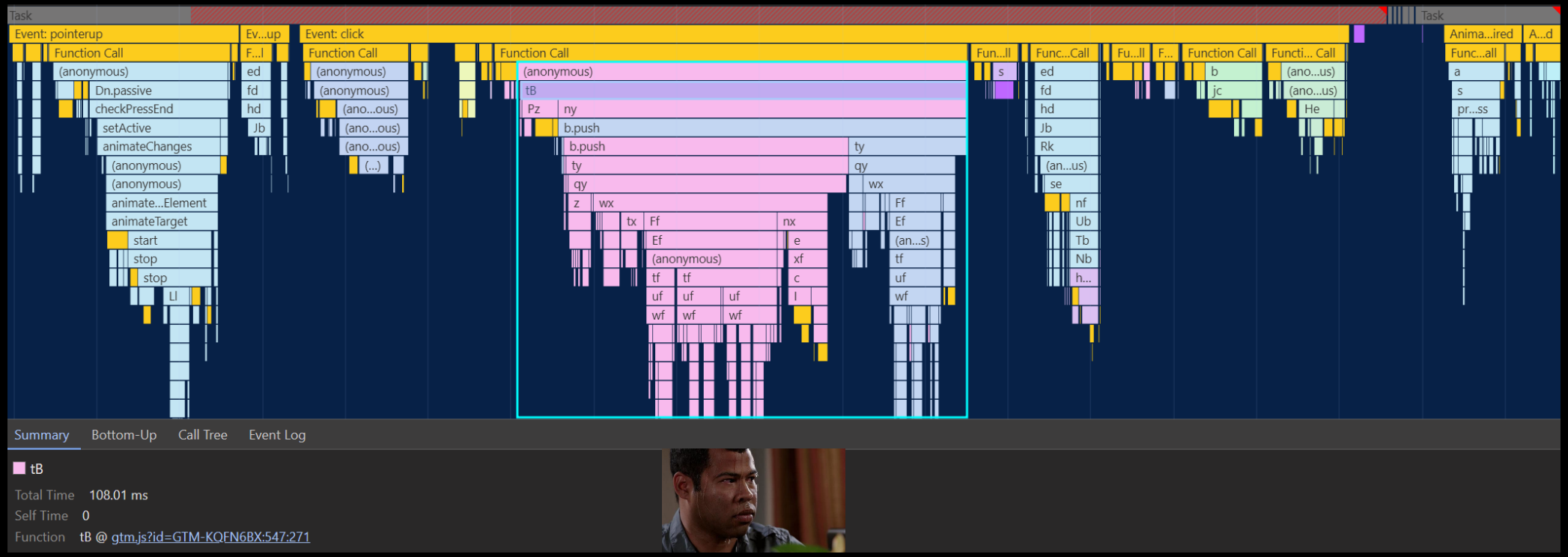
- With just yielding, there is a chance the browser unloads the page before executing the callback after the yield point

```
someButton.addEventListener('click', (event) ⇒ {
  doSomeImportantUIWork(event) //     give user meaningful feedback
  await yieldUnlessUrgent()    // 🔙 yield before running any analytics code, unless...
  sendAnalytics()              //    ... the user is about to leave the page.
})
```

- Guarantees a callback runs before a user leaves the page
- Useful for business-critical analytics, or saving forms
- Can be paired with both **yieldToMain** & **interactionResponse**

Only 5 slides to go

Google Tag Manager (GTM)



Analytics

- Usually causes tons of INP issues
  - *dataLayer.push / gtag() is more like “push INP up”*

- “What if we remove GTM” -> realistically this will never happen

Experimental 🧪 – auto yield GTM/GA

```
// Google Tag Manager
const originalDataLayerPush = dataLayer.push
dataLayer.push = (...args) => { paintBefore(() => { originalDataLayerPush(...args) }) }

// Google Analytics
const originalGtagPush = gtag
gtag = (...args) => { paintBefore(() => { originalGtagPush(...args) }) }
```

Analytics

Experimental 🧪 – auto yield GTM/GA

```
// Google Tag Manager
const originalDataLayerPush = dataLayer.push
dataLayer.push = (...args) ⇒ { paintBefore(() ⇒ { originalDataLayerPush(...args) }) }

// Google Analytics
const originalGtagPush = gtag
gtag = (...args) ⇒ { paintBefore(() ⇒ { originalGtagPush(...args) }) }
```

I’ve shipped it to prod just-like-that™, results:

P99 – 50ms

P75 – 15ms

Make sure to read the ‘Cons’: https://kurtextrem.de/posts/improve-inp#event-handlers--google-analytics--google-tag-manager-

Analytics

Experimental 🧪 – auto yield GTM/GA

```
// Google Tag Manager
const originalDataLayerPush = dataLayer.push
dataLayer.push = (...args) => { paintBefore(() => { originalDataLayerPush(...args) }) }

// Google Analytics
const originalGtagPush = gtag
gtag = (...args) => { paintBefore(() => { originalGtagPush(...args) }) }
```

Tips:

- Run it at the (window) `load` event, so that your push function overrides the GTM function
- Pair it with the yieldUnlessUrgent pattern (rAF + setTimeout) so you‘re not the one to blame for less visitors ;)

"I don't know if this helps anyone,
but one thing I've been stressing
to our dev teams at Crate and Barrel
is this: We're not trying to speed up the
website by 500ms. We're trying to speed up
the website by 100ms, five times.
Or 50ms, ten times."

Dan Gayle // Crate & Barrel

https://www.youtube.com/watch?v=L6gZp3-7w8c

Thank you. Questions?

Contact me on X: @kurtextrem, I (re)tweet webperf stuff

React folks: Check out my previous talk specifically about React & INP, it has a few more practical examples & useful tips - kurtextrem.de/INP.pdf