



What React 18 does and how it impacts your INP

Jacob Groß | [@kurtextrem](#) | 7th Feb 2024

Who am I?

- Performance Engineer @ Framer *(you might know us from “Framer Motion”)*
- prev. Principal Engineer @ Jochen Schweizer mydays
- I participate in W3C WebPerfWG calls *(quite fresh)*
- I like making things fast & accessible for all users of the internet
- Always open for discussions about all things perf
- github.com/kurtextrem/awesome-performance-patches

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

Time between User Interaction -> UI update (paint)

- Slower than 200ms -> Bad!
- **Faster than 200ms -> Good!**





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

Time between User Interaction -> UI update (paint)

- Slower than 200ms -> Bad!
- Faster than 200ms -> Good! **Good?**

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

RESPONSE

0.1
SECONDS

ANIMATION

16
MILLISECONDS

IDLE

50
MILLISECONDS

LOAD

1
SECOND

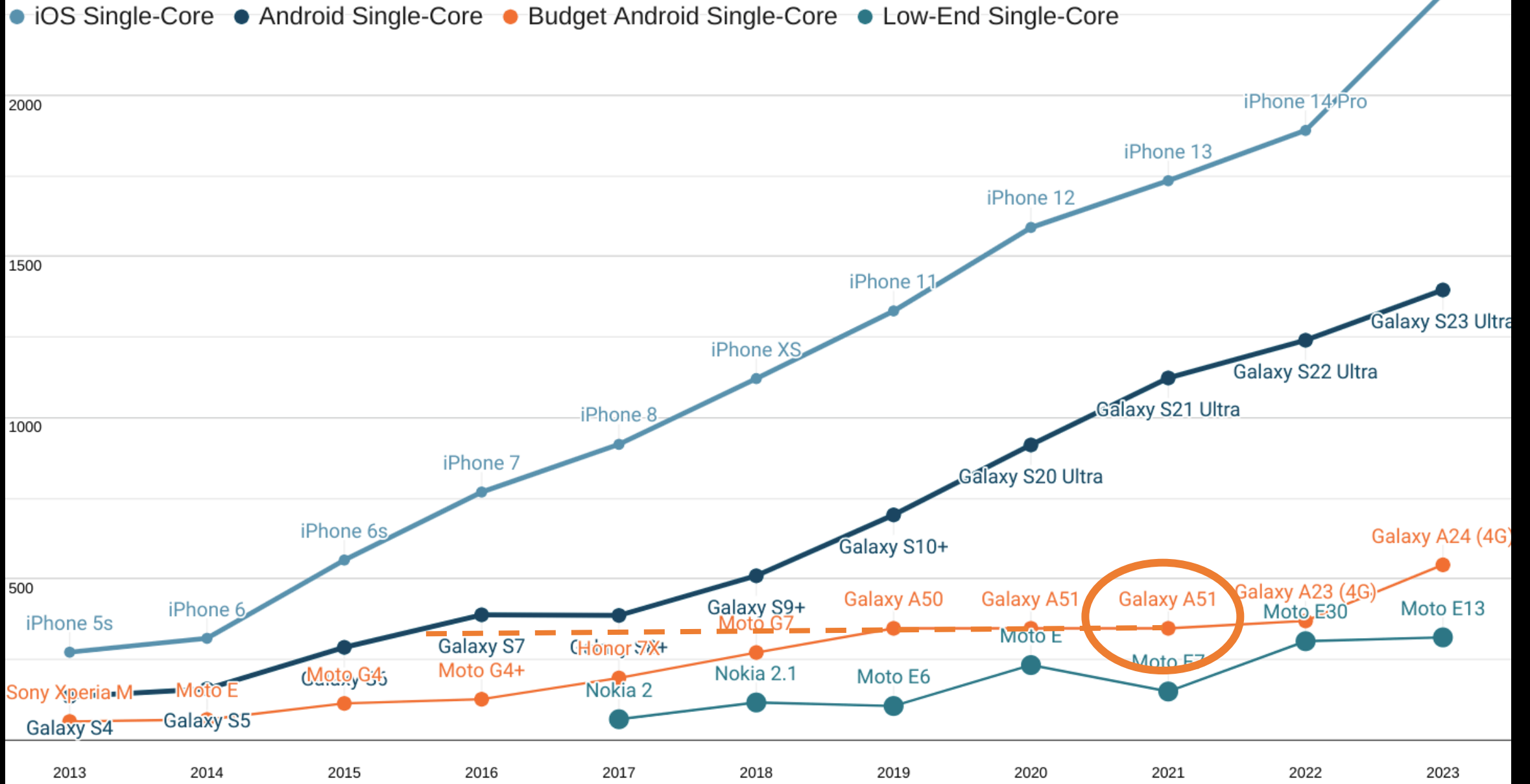
Why should I care?

- Do you like clicking buttons 2, 3, 4 times until something happens?
 - Do you like clicking buttons 2, 3, 4 times and the action happens 2, 3, 4 times with a delay each time?
 - Do you like clicking buttons 2, 3, 4 times and the action happens 2, 3, 4 times with a delay each time?
 - Do you like clicking buttons 2, 3, 4 times and the action happens 2, 3, 4 times with a delay each time?
-
- The usual response is **no**.
 - **Rage Clicks => Bad UX** 🤡

Why must I care?

- Amplified on mobile, because devices are **sloooooow** 🐌
- Your 🍷 device != users' device
- Avg. 🇩🇪 Android device != avg. developing country device
- Galaxy A51 for 240€ from 2021 is the avg. device
 - Sounds recent?
 - Specs match devices from 2017 or earlier!

Geekbench 5 Single-Core Scores



Why should my boss care?



Why should my boss care?

- Failing INP = you fail Core Web Vitals on 12th March
=> **could be bad for SEO!**
- INP is **not** measured on the very powerful iOS devices
-> bad luck if you have 99% Safari customers; you will need to optimize for Chrome anyway
- Improve UX for ppl. with slow devices => improves UX for all users
- INP has impact on KPIs like Click-Through-Rate (CTR), **Conversion Rate (CVR)**, Bounce Rate

Speeding up websites is important—not just to site owners, but to all Internet users. **Faster sites create happy users** and we've seen in our **internal studies** that **when a site responds slowly, visitors spend less time there.** But faster sites don't just improve user experience; recent data shows that improving site speed also **reduces operating costs**. Like us, our users place a lot of value in speed—that's why we've decided to take site speed into account in our search rankings. We use a variety of sources to determine the speed of a site relative to other sites.



Why should my boss care?

- Failing INP = you fail Core Web Vitals on 12th March
=> **could be bad for SEO!**
- INP is **not** measured on the very powerful iOS devices
-> bad luck if you have 99% Safari customers; you will need to optimize for Chrome anyway
- Improve UX for ppl. with slow devices => improves UX for all users
- INP has impact on KPIs like Click-Through-Rate (CTR), **Conversion Rate (CVR)**, Bounce Rate

LEADERSHIP • FORBESWOMEN

**Customers Who Have Excellent
Experiences With Brands Spend
140% More**

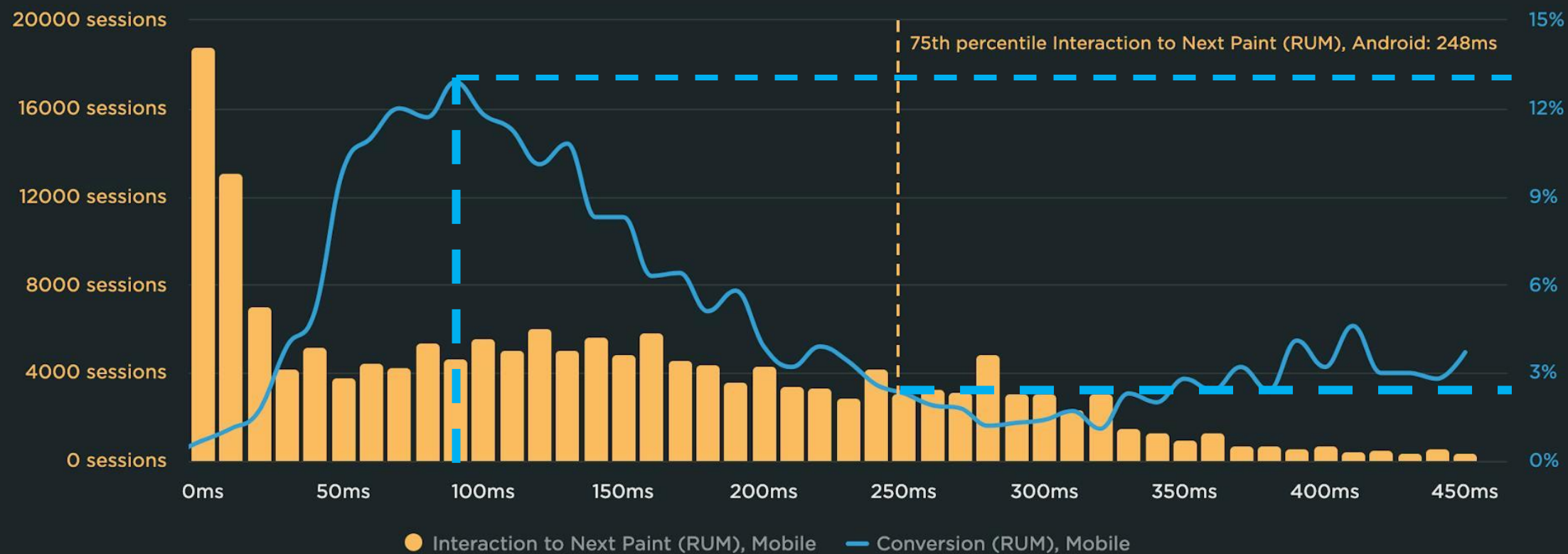
Forbes

MOBILE INP VS. CONVERSION



Interaction to Next Paint (RUM), Mobile

248ms



The background is a dark, blurred image of a data visualization. On the left, there is a histogram with yellow bars. To the right of the histogram, there is a line graph with a blue line that rises and then fluctuates. The entire scene is framed by a white border with rounded corners.

13% CVR @ 100 ms

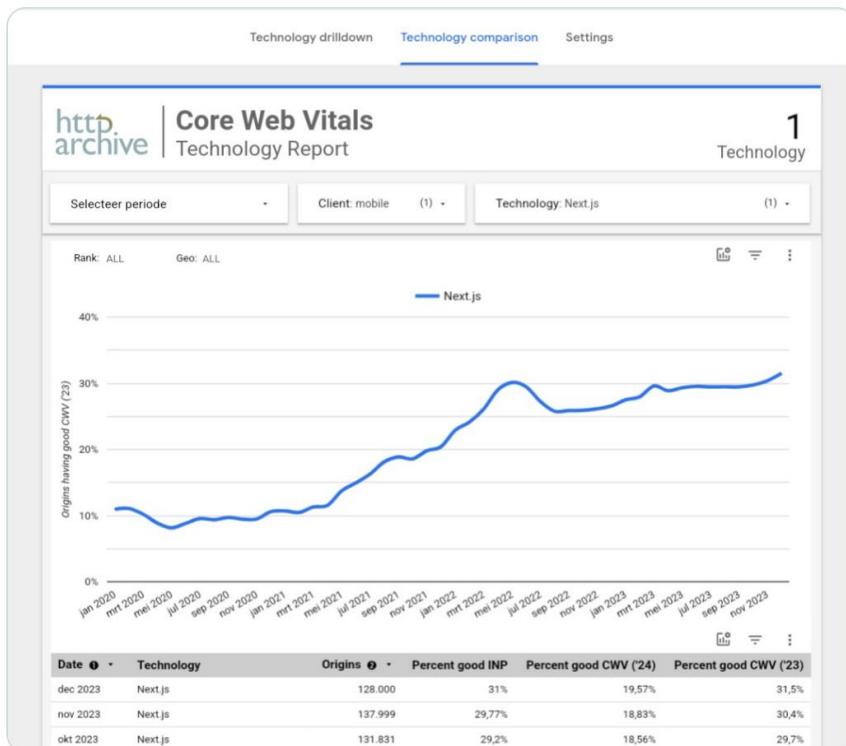
VS.

3% CVR @ 250 ms

This is not **trivial**.

 **Jordy**
@JordyScholing · Follow

How will #Nextjs (devs) fix INP and CWV issues in 2024?



9:25 AM · Jan 22, 2024

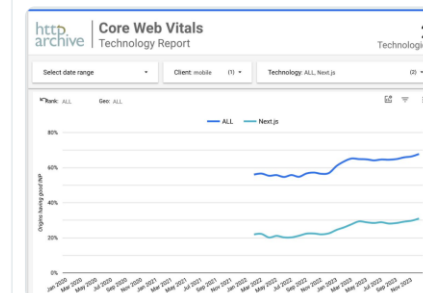
3 Reply Share

Read 4 replies

<https://twitter.com/JordyScholing>

 **Dan Shappir** host on @JSjabber podcast
@DanShappir · Follow

How much of this improvement are NextJS devs? How much is NextJS itself? How much is the underlying platform? Looks to be mostly (only?) the latter



10:12 AM · Jan 22, 2024

1 Reply Share

Read 1 reply

 **Jordy** @JordyScholing · 22. Jan.

The percentage of origins passing the Core Web Vitals will drop from a low 30% to 20%.

48

<https://twitter.com/DanShappir>

The NextJS team has awesome devs.
They know how to fix INP.

This is not trivial.

Should my face now look like this 🤪 ?

- No.
- That's why you're here, right? 😊
- Don't underestimate the efforts and start "today"
 - *This includes convincing your boss!*
 - *Grab a drink w/ your SEO team and start making "alliances"*
 - *CEO's usually listen to their SEO team more than to a dev saying "boss, INP is important"*
- Remember: **200ms is your first target**, currently there are no plans to lower this threshold *(however, at some point, Google may)*


I'm sold, what do I do?

1) Find your culprits!

a) DIY: Use the Web Vitals chrome extension and click through your page

Enable logging + use the “Performance” tab for **6x CPU slowdown**

b) Free INP/CrUX tools

Lab:  DebugBear INP debugging tool

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

c) Convince your boss to invest in web perf tool(ing)

=> **best data, as you get field data** (esp. once LoAF lands)

See above, or also SpeedCurve, Akamai mPulse, Catchpoint, SpeedVitals, Sentry, ...

2) Fix it

Make INP **green** with this one simple trick:



Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

20% complete



For more information about this issue and possible fixes, visit <https://www.windows.com/stopcode>

If you call a support person, give them this info:

Stop code: CRITICAL_PROCESS_DIED

Hah, bad news. There is no “magic fix”.

* But we get to see some magic fixes later

React 17 vs. React 18

... finally coming to the part you're probably here for?

React 17

- Quick recap about how React event handlers worked for ages:
set state after async -> update **synchronously**

```
onClick={()=>{ fetchSomething().then(() => {  
  setState(a); // causes a re-render  
  setState(b); // causes a re-render  
})}}
```

=> user will never see “a”, so it was unnecessary

- We usually use throttling/debouncing + memo to avoid double work

React 18

🪄 Magic fix #1: Event handlers batched updates

```
onClick={() => { fetchSomething().then(() => {  
  setState(a);  
  setState(b);  
})}}}
```

=> 1x re-render with state “b”

=> Less re-rendering work on main thread => better INP

React 18

Enter “concurrent mode”

- We now have “urgent” and “non-urgent” updates
- via scheduler that gives control back to the browser every 5ms

```
let startTime = performance.now(); // updated on every render

function shouldYieldToHost() {
  return (performance.now() - startTime) > 5 /* ms */;
}
```

- Non-urgent is non-blocking => can be interrupted by urgent updates

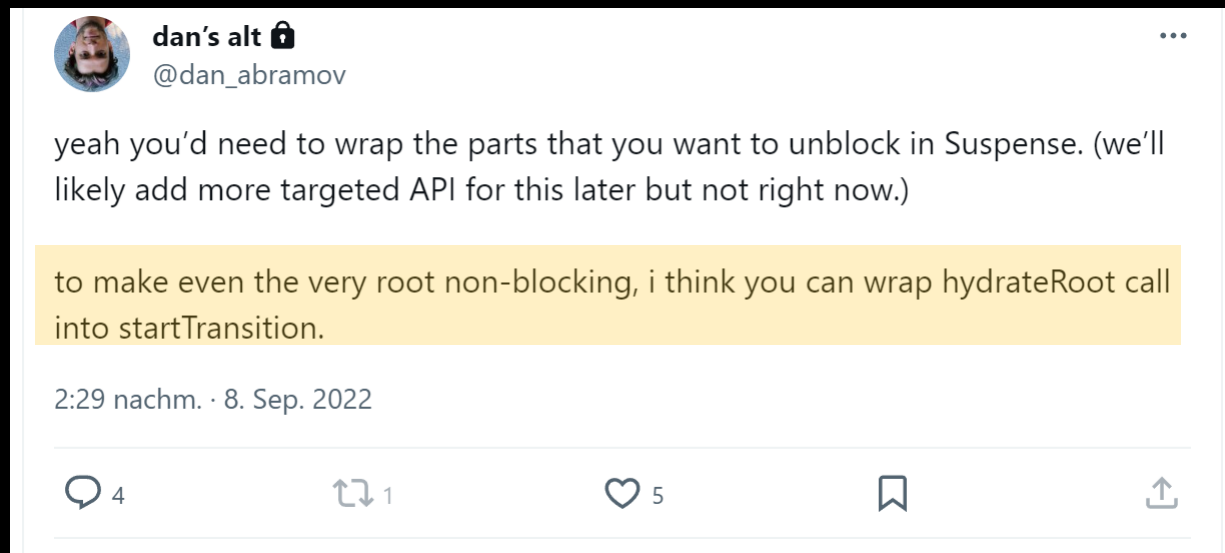
! This is not the default. It is only in play when using transition utilities. **!**

React 18

- ✨ Magic fix #2: “selective hydration” with `<Suspense>`
 - New API: `hydrateRoot()`
 - All children are marked as “non-urgent”
=> hydrated after all other “urgent” components
 - User input (like click) makes `<Suspense>` trees urgent
 - Expensive hydration = slow INP => fixed 😎

React 18

`startTransition(() => hydrateRoot())`: enable non-blocking hydration

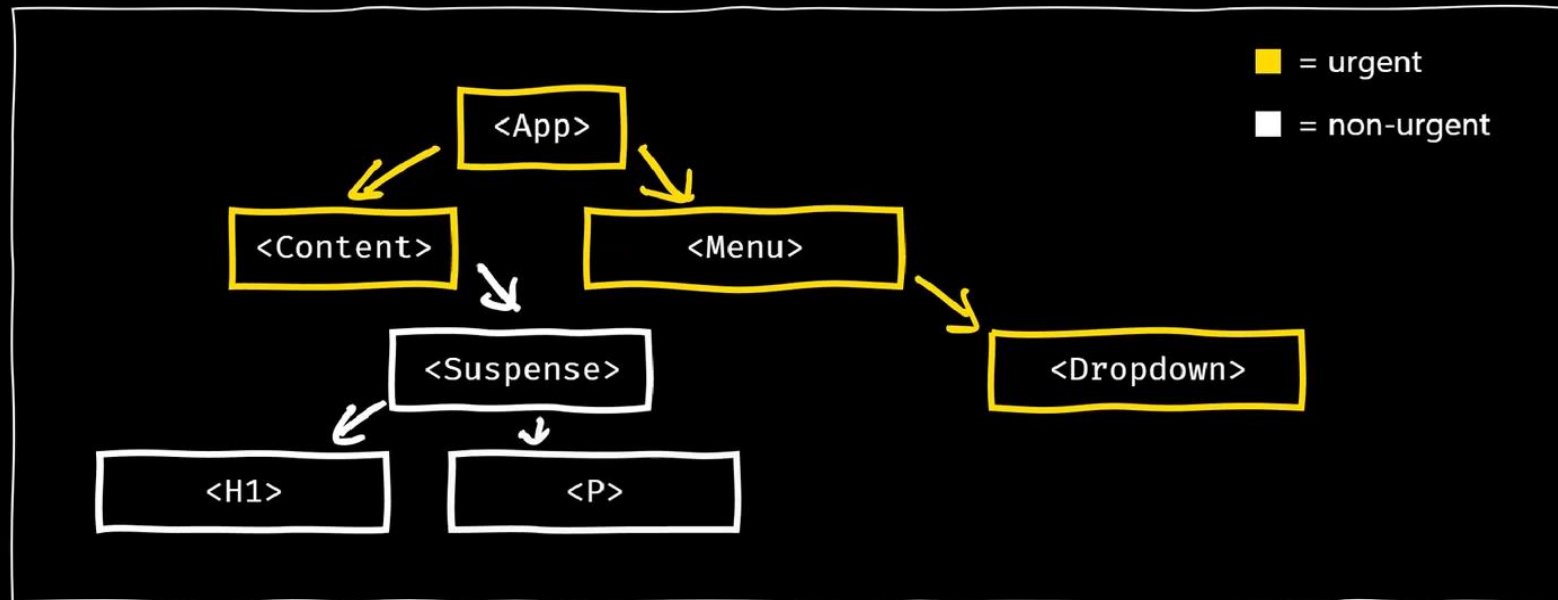


Undocumented btw. A real ✨ magic ✨ tip!

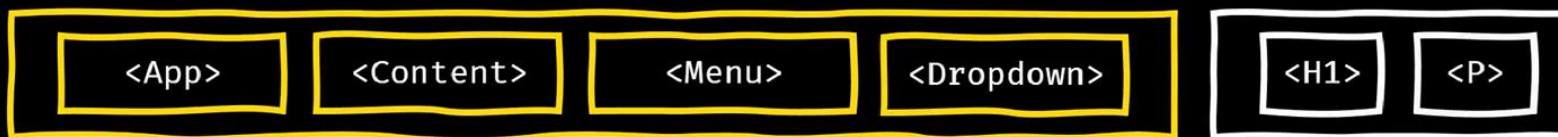
NextJS does that already for you (and maybe others).

1 `hydrateRoot();`

2



3



678 ms 1778 ms 1878 ms 1978 ms 2078 ms 2178 ms 2278 ms 2378 ms 2478 ms 2578 ms 2678 ms 2778 ms 2878 ms 2978 ms 3078 ms

► Network
► Frames
► Animation
Timings

React hydrates the
non-`<Suspense>`
part of the app

Then React starts hydrating
components inside `<Suspense>`,
~5ms at a time

However, as soon as the user clicks
something inside `<Suspense>`, React
switches back to urgent hydration

Main — http://localhost:8080/

| Task |
|---------------|
| Function Call |
| R |
| J |
| tl |
| vl |
| HI |
| II |
| Jl |

| Task |
|---------------|
| Function Call |
| R |
| J |
| tl |
| vl |
| HI |
| II |
| Jl |



| Task |
|--------------------|
| Event: pointerdown |
| Function Call |
| sd |
| td |
| Sc |
| Fl |
| xg |
| rl |
| vl |
| HI |
| II |
| Jl |
| li |

Summary Bottom-Up Call Tree Event Log

Range: 1.64 s - 3.13 s

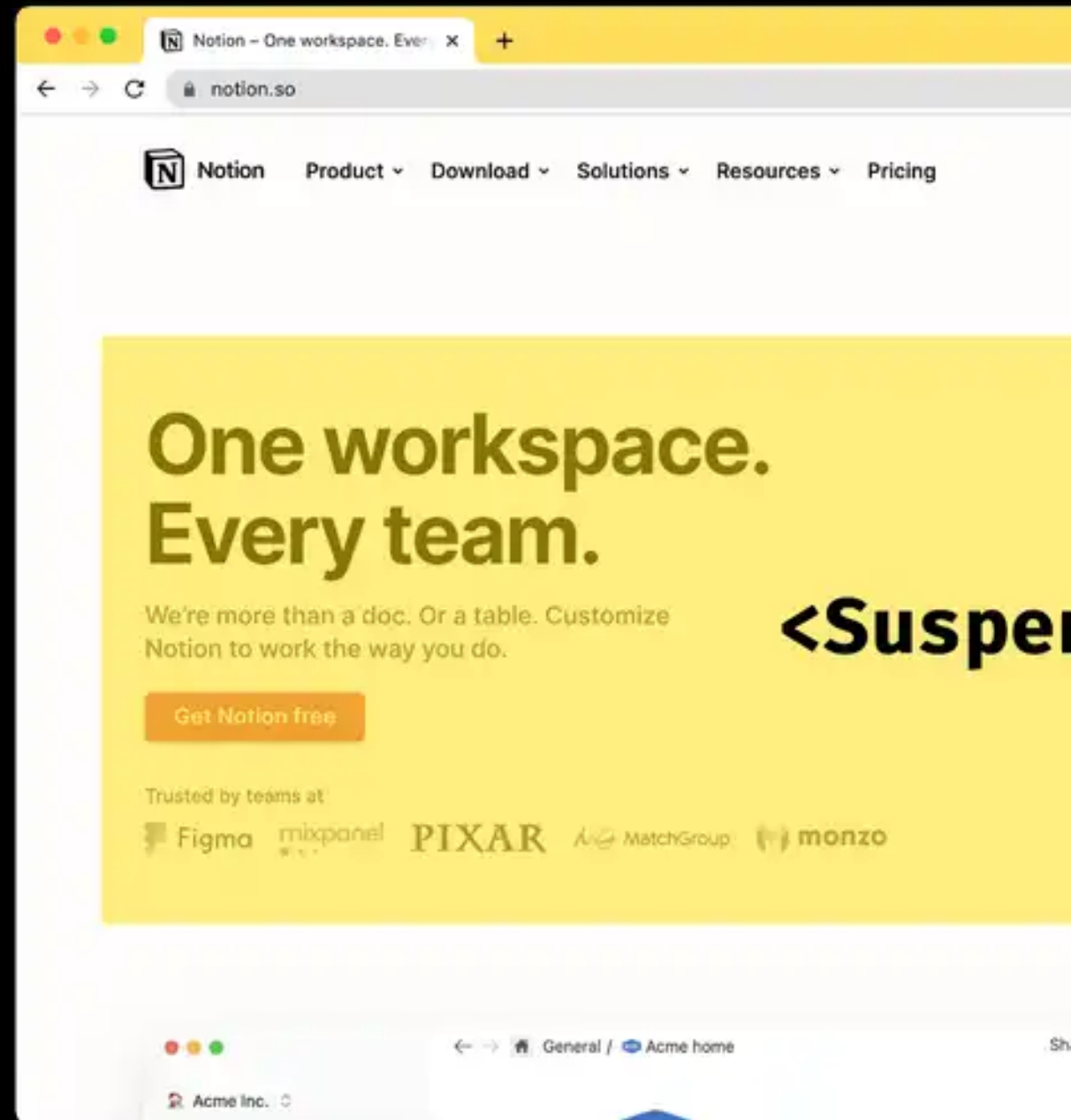
1230 ms Scripting

26 ms Rendering

Total blocking time: 1383.29ms (estimated) [Learn more](#)

```
import { Suspense } from 'react';

<Suspense>
  <H1>One workspace. Every team.</H1>
  <P>We're more than a doc.</P>
  <Link href={...}>
    Get Notion free
  </Link>
  ...
</Suspense>
```



React 18

🪄 Magic fix #1: Batched updates

🌟 Magic fix #2: Selective hydration

Result – Zalando:

-5.69% INP

-2.43% LCP

-0.24% Bounce Rate

Result – Vercel:

TBT: 80ms (from **480**)

INP: 48ms

Transition Utilities – React 18

useDeferredValue(): re-render w/ old value, schedule bg re-render

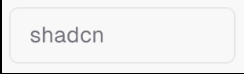
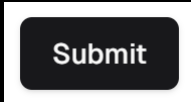
“if the user is typing into an input faster than an expensive component using its deferred value can re-render, it will only re-render after the user stops typing.”

⬆ Better **debounce** mechanisms – no artificial delay ⬇

useTransition()/startTransition(): mark state updates as “non-blocking”

*“if you update an expensive component inside a transition, but then start typing into an input while it is in the middle of a re-render, React will restart the rendering work on the expensive component **after** handling the input update.”*

What is urgent?

- Anything that a user expects immediate **feedback** from is urgent
- An **input** is urgent 
 - If you have a controlled component, updating the **value** is urgent
 - A 'word counter' maybe not so much (a tiny bit of delay is fine)
- **Click** on a button/link/element 
- “Avoid rage 🤔” as a mental model

Optimistic UI & Pending UI



Optimistic UI:

Act like the action was successful, before e.g., a network request finishes (run in parallel)



Immediate Feedback:

Show user something is happening by updating UI right on the user interaction



Pending UI:

Uncertain what's happening next (e.g., checkout success or failure)?
Use a “busy indicator” (spinner, skeletons)

Immediate *feedback*
creates better *UX*.

Be *optimistic*.

Quick fix: Analytics

“Debugging INP” @ <https://youtu.be/nQByr5Yyclw?t=1625>

Common slugs



Long Input Delay

Longa mora inputus



Event Callback

Reversus eventus



Presentation Delay

Mora presentationis



Image source: Oregon State University

○ ○ ○

```
1 window.yieldToMainThread = async () => {  
2   if ('scheduler' in window && typeof window.scheduler === 'object' && window.scheduler !== null) {  
3     if ('yield' in window.scheduler && typeof window.scheduler.yield === 'function')  
4       return window.scheduler.yield()  
5  
6     if ('postTask' in window.scheduler && typeof window.scheduler.postTask === 'function')  
7       // We choose user-visible since user can interact with the page during the yielding interlude.  
8       return window.scheduler.postTask(() => {}, { priority: 'user-visible' });  
9   }  
10  
11   return new Promise(resolve => { setTimeout(resolve, 0) });  
12 };
```

Quick fix – Button/Link

○ ○ ○

```
1 const CheckoutBtn = () => (  
2   <Button  
3     onClick={async(e) => {  
4       try {  
5         await doSomeAJAXStuff();  
6         pushToDataLayer();  
7         redirectToNextPage();  
8       } catch (e) {  
9         // aww, checkout failed  
10      }  
11    }  
12  />  
13 )
```

Change color, disable it, ...

```
1 const CheckoutBtn = () => (  
2   <Button  
3     onClick={async(e) => {  
4       const promise = doSomeAJAXStuff().catch(/*handleError*/); // ⬅ no `await`  
5       // ⬇ update UI & yield (= allow browser to paint)  
6       e.target.setAttribute('aria-disabled', 'true');  
7       window.yieldToMainThread();  
8     }  
9     try {  
10      await promise;  
11      pushToDataLayer();  
12      redirectToNextPage();  
13    } catch (e) {  
14      // aww, checkout failed  
15      e.target.setAttribute('aria-disabled', 'false');  
16    }  
17  }  
18 />  
19 )
```

Disable immediately + trigger visual
update with CSS

e.g. [aria-disabled] { opacity: 0.7; pointer-events: none }

Undo on error



Wes Bos

@wesbos · Follow

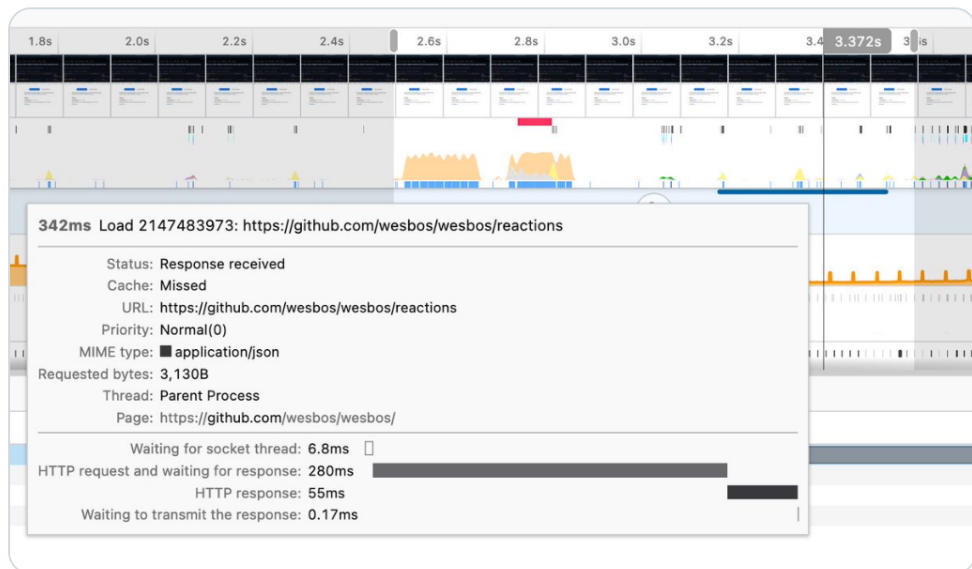


Perf nerds - help us learn !

How would you find out why clicking a github emoji reaction takes ~2.5 seconds from clicking it to it updating on the page.

Network seems to take up ~350ms, but there is almost a second before that request even fires.

Is there some sort of event... [Show more](#)



6:59 PM · Jan 29, 2024



54



Reply



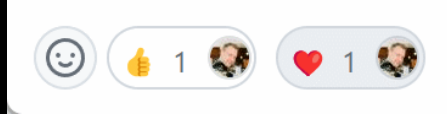
Share

[Read 10 replies](#)

<https://twitter.com/wesbos/status/1752028584590319830>

Optimistic UI: GitHub reactions

- Rage x 3000



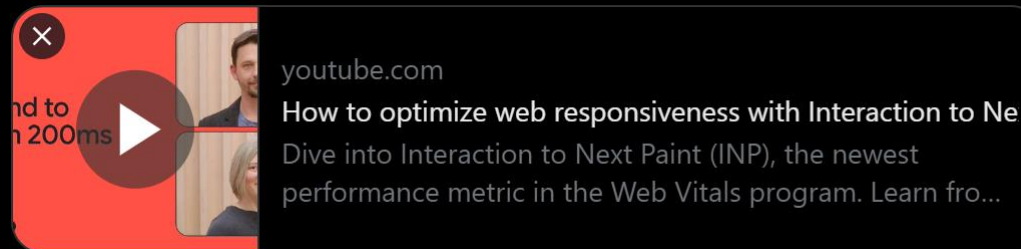
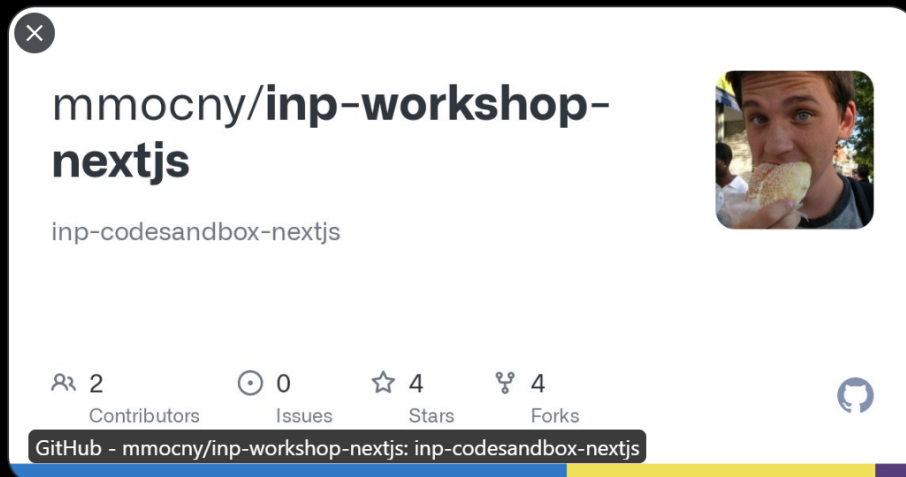
- Long story short: They have a 1000ms debounce in place (no idea why)
- Does anyone use Instagram here?
Clicking “Like” on any post runs the animation immediately
=> **feels fast** on any device & network 🚀

💡 Update count & run network; update UI again if needed

Fixing an expensive search

<https://codesandbox.io/p/github/mmocny/inp-codesandbox-nextjs/csb-cswxqy/draft/mystifying-bell?file=%2Fsrc%2Fapp%2Fpage.tsx%3A37%2C10>

- Prefer transition utilities over debouncing
- Break up long tasks into smaller ones
 - **Yield** your JS code *(like you yield work for coffee ☕)*
 - **Abort** running work & network if there is more recent work



If it doesn't provide feedback,
it is *not urgent* to the user.

We run it after *yielding*.

Recap: How do I fix – Cheatsheet (1/2)

☐ Use the “**optimistic UI**” & “**pending UI**” pattern:

- Always run network reqs in parallel, never blocking to UI updates (prefetch where applicable)
- Any interactive element -> change appearance (color, size, ...)
- Analytics **always** has lowest priority (= run after UI update)

☐ React: useAbortSignallingTransition() & transition utilities

☐ Wrap less important components in <Suspense>

☐ or better, eliminate need for hydration (lazy hydration / responsive hydration)

☐ JS: yieldToMainThread()

☐ Animations: Avoid causing reflow or animating expensive CSS props

Recap: How do I fix – Cheatsheet (2/2)

- ❑ Use Field Data if available, else use DebugBear & similar for Lab Data w/ 6x CPU slowdown
- ❑ Use the “**optimistic fix**” pattern:
 - Fix now in a maybe not-so-clean way -> make INP green now
 - Fix “clean” afterwards
 - **Favor** fixes that improve UX
- ❑ Handle edge-cases like error scenarios, network failures, long loading states (> 1s), ... in optimistic UI
- ❑ Contribute solutions to github.com/kurtextrem/awesome-performance-patches
- ❑ Mega thread: twitter.com/rick_viscomi/status/1754536134690898053

"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

Outlook

- React 18 Canary improves INP with the intro of RSC => avoids hydration of static components; e.g., available in NextJS ([Vercel's Blog](#))
- `useOptimistic` ([react.dev](#)) is perfect for the optimistic UI pattern ([demo](#))
- [Remix](#) has great docs for optimistic / pending UI + soon has RSC support
- React Forget Compiler ([react-forgetti](#) / [Million](#))
- Don't only defer your 3rd party, run it in WebWorkers ([PartyTown](#) 🎉), server-side GTM / [Cloudflare Zaraz](#)



Thank you. Questions?

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