Web Performance Calendar

The speed geek's favorite time of year
2025 Edition

The Curious Case of the Shallow Session SPAs

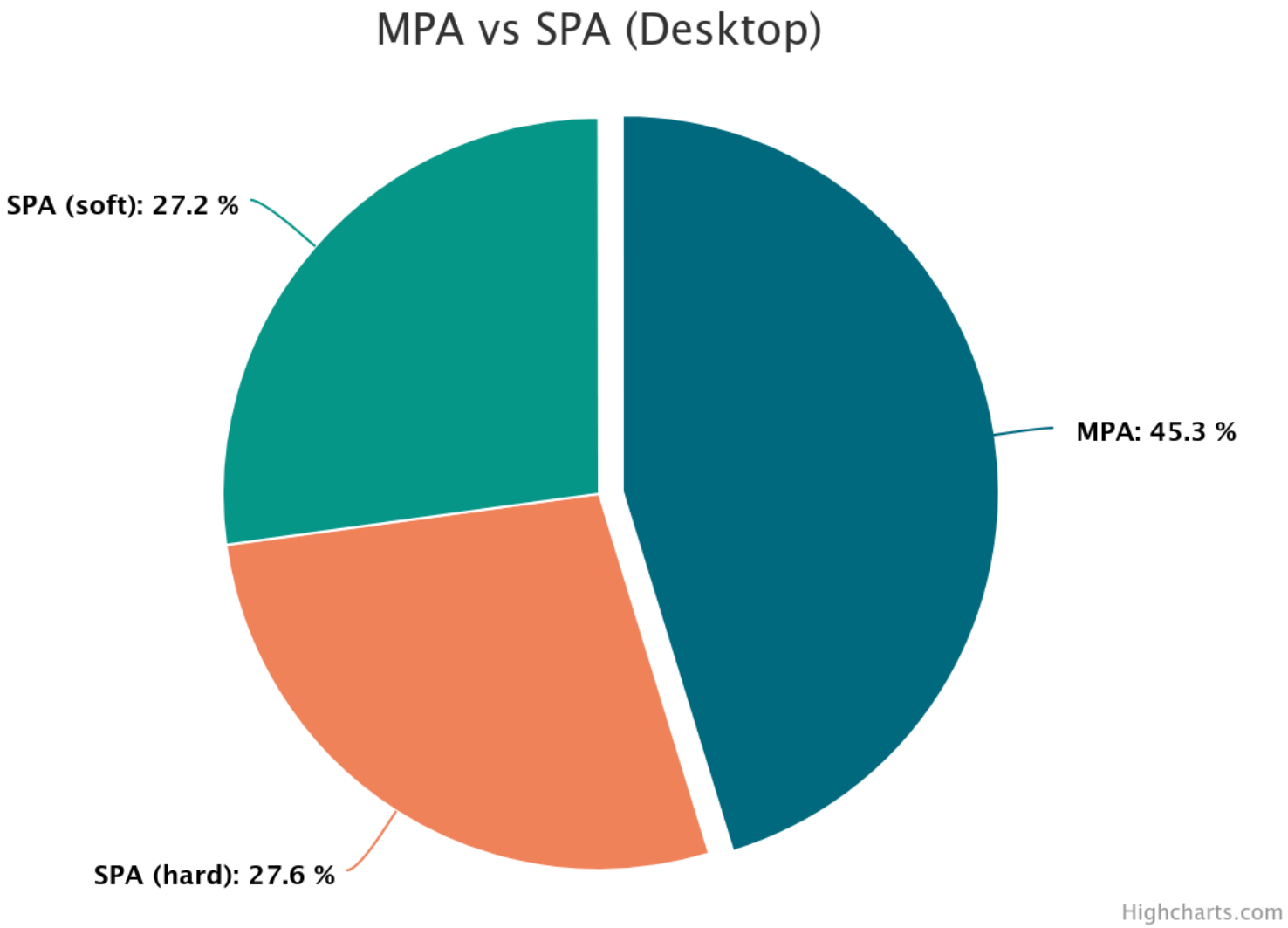
31stDec 2025 by Alex Russell
ABOUT THE AUTHOR

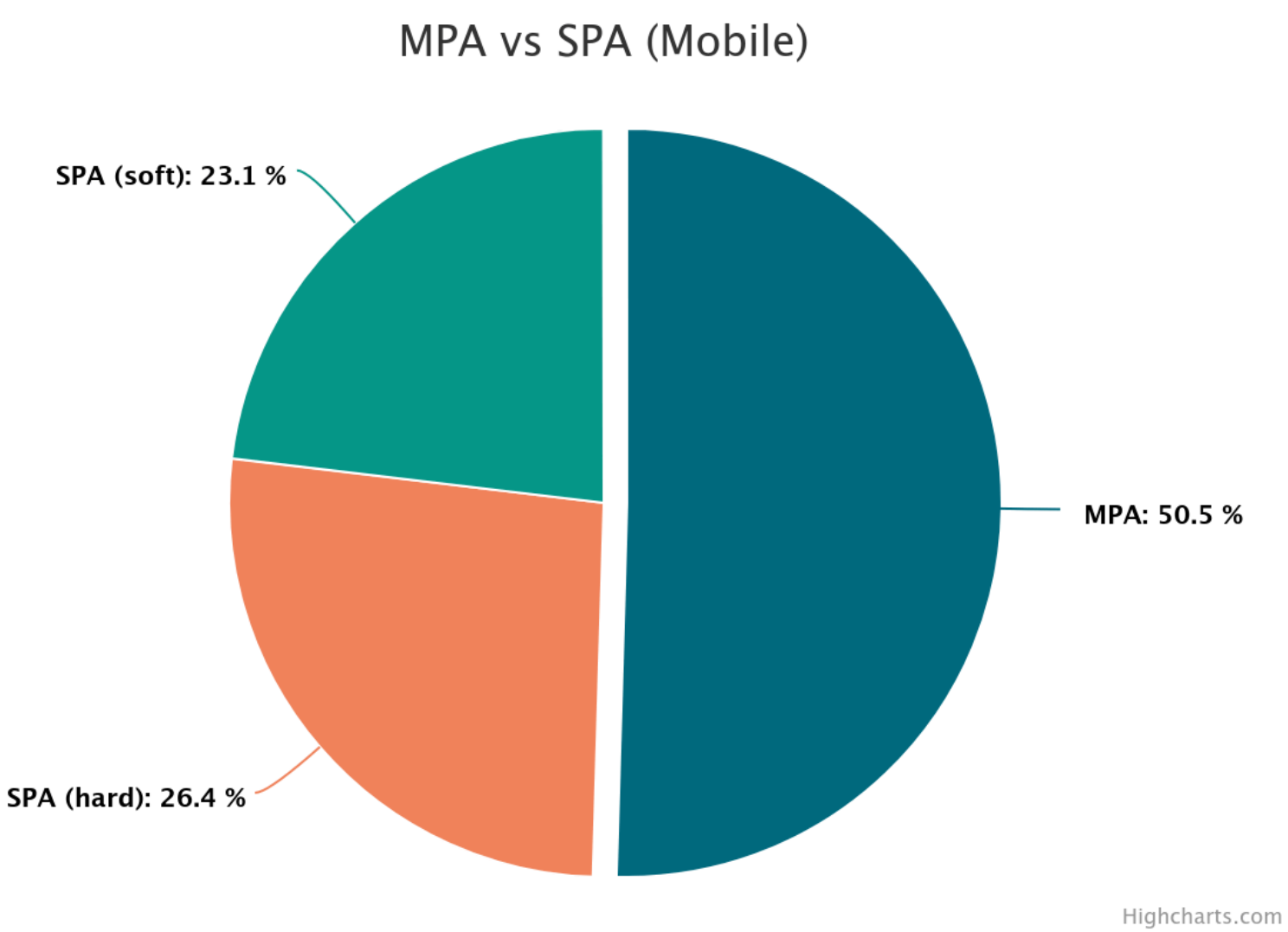
Alex Russell (@slightlylate) is Partner Program Architect on the Microsoft Edge team and Blink API OWNER. Before joining Edge in 2021, he worked on Chrome's Web Platform team for a dozen years where he helped design many new features. He served as overall Tech Lead for Chromium's Project Fugu, lead Chrome's Standards work, and acted as a platform strategist for the web. He also served as a member of ECMA TC39 for more than a decade and was elected to three terms on the W3C's Technical Architecture Group.

His technical projects have included Fugu, Progressive Web Apps, Service Workers, and Web Components, along with ES6 features like Classes and Promises. Previously he helped build Google Chrome Frame and led the Dojo Toolkit project. Alex plays for Team Web.

Buried at the end of this year's installment of my semi-annual series on network and device reality is a mystery: multiple, independent data sets from the Web Performance community indicate sites built as Single-Page Applications ("SPAs") receive, on average, only one (1) "soft navigation" for every "hard" page load.

The RUM Archive and Michal Mocny's early analysis from the Chrome Soft Navigations Origin Trial data agree: users of SPAs don't generate long sessions on average:


MPA vs SPA (Desktop)
SPA (soft): 27.2 %
MPA: 45.3 %
SPA (hard): 27.6 %
Highcharts.com



Taken at face value, this *appears* to indicate an industry-wide failure of technology selection, implying huge accretions of complexity for very little payoff. Is that what's happening? I believe this is the most important mystery for the web performance community to get a handle on in 2026.

In strange congruence, this mystery is partially modeled by the definition of web performance I haltingly offered four years ago in these very pages:

- The mission of web performance is to expand access to information and services.
- **We expand access by reducing latency and variance across all interactions in a user's session to more reliably return the system to an interactive state.**
- We work to improve the tail of the distribution (P75+) because that is how we make systems accessible, reliable, and equitable.

In that piece, I outlined the core loop for handling any input in a computer program:

1. The system is ready to receive input.
2. Input is received and processed.
3. Progress indicators are displayed.
4. Work starts; progress indicators update.
5. Work completes; output is displayed.
6. GOTO 1

The soft-vs-hard navigation ratio question is fundamentally about this loop and our goal to reduce latency and variance. Can we *really* make additional trips through the loop faster via JS? Is there even a credible opportunity to do so?

When we discuss SPAs, MPAs, hybrid approaches, this question lurks in the background, with proponents of each stressing costs and variability in different stages of the loop. Session depth also reads on the possibilities for technologies like Service Workers, Multi-Page View Transitions and Speculation Rules to change the calculus. If sessions are shallow, the denominator we would divide up-front JavaScript costs by could be unsatisfyingly small.

If sessions are truly as shallow as the RUM Archive and Chrome Origin Trial data suggest, there's almost no space at all for SPA technologies in the market. Adopting SPA technology under these conditions looks foolish outside a narrow set of high-confidence use-cases (e.g., GIS, chat, and document editing).

So we have an industrial-scale mystery on our hands: can this data possibly be right?

Mysterious Circumstances

The deal presented by SPAs has always been a trade: for the price of more up-front JavaScript (increased app loading latency), follow-on interactions will feel faster.

We can understand this mathematically in terms of session-length amortized latency. Average interaction latency is the sum of latencies handled by the application (not the browser), divided by the number of total interactions (N):

$$L_{\text{avg}} = \frac{\text{latency(navigation)} + \sum_{i=1}^{I} \text{latency}(i)}{N}$$

This assumes a single "hard" navigation, with all follow-on interactions (I) handled by script, but even in mixed-mode apps, the same logic roughly holds. Navigation latency (LCP), plus the number of JS-handled events, gives a session-weighted understanding of the benefits and costs of the SPA trade. Every interaction could be a full-page reload, or handled by script, giving us a powerful way to analyse the trade SPAs offer from real-world data.

How these architectures perform in practice matters a great deal to the businesses funding site development, which means the value of N is must-know information. And to date, we've been flying blind.

The Stakes

The inherent complexity of SPAs is foundationally justified in terms of delivering better user experiences. Proponents describe *"fluid transitions"* and *"faster updates"* as benefits that are impossible to deliver (practically) when building in other styles. And this sales pitch has worked.

It's no exaggeration to say the default way new projects begin today is a decision over which React-toting toolchain to base the frontend on, laying the keel for everything that follows. These toolchains have historically sent *all* the JavaScript for components a site might ever need down the wire, often in the critical path.

Any team member, working on any sub-interface—no matter how infrequently surfaced—regresses first-load performance for all users by default. Only when teams notice can they adopt the situated strategies that mitigate the impact of these quickly-growing bundles. This is hard work, and it can take months for things to level out, let alone improve. My own consulting experience suggests that the usual moment when this happens is *"way too late."*

The spread of these tools has been correlated with failure at scale:

Once products get into this state, the road back to health is long, complicated, and cuts against the grain of these tools. It was not always thus.

UI logic used to reside primarily on the server, and filigrees of JavaScript were layered on via progressive enhancement. Incremental additions to the codebase were not *de facto* additions to the most contested and expensive critical-path assets. But progressive enhancement doesn't cover every base. Some applications benefit tremendously from the lower incremental update latency due to optimistic updates of local (client-side) data models. These updates rely on enough componentry being available on the client to render the results without waiting on the server, creating a market for component systems and a need to pre-fetch component definitions.

Over time, a growing set of developers assumed they'd need to build this way, and given the shocking complacency of certain browser vendors, this wasn't as barmy as it seems in retrospect. Even today, targeting only the best browsers, some classes of applications are clunky to deliver via Progressive Enhancement; all browsers lack common native components including data-backed scrolling lists, gesture-based UIs, scroll-consuming interactions, maps, date-range pickers, login-state indicators, customisable menus, and much, much more.

Having taken a few steps down this road, SPA developers also rediscovered the components the platform provides "for free" remain stubbornly unstyleable and inextensible. This owes chiefly to Apple's 15-year streak of rejecting HTML element customisation and disengagement from attempts to upgrade the DOM to better serve application needs. Cupertino has even gone so far as to serially veto attempts at integrations with the HTML parser for subclassing form elements, making it impossible to reach any iOS user with modern PE approaches.

But for these obvious and persistent omissions, entire classes of applications have taken a frameworkist path. Developers leapt into the Turing tarpit, only to find themselves moving slower, not faster. But unless they decamped for JavaScript abstractions, they would not be able to deliver popular UI affordances *at all*. Eventually, routing around these yawning Web Platform gaps via JavaScript became *de rigueur*, and realism about network and device limits became unpopular to discuss.

The results of these overlapping failures have been disastrous.

Far from managerial dreams of labour arbitrage, the spread of React SPAs has created new specialisations in remediation. Experts are now needed to beat back the chronic issues these architectures aggravate. Client-side JavaScript has always been the slowest and most expensive way to accomplish anything in a browser, but that reality was wished away over frontend's lost decade. Teams investing in these approaches are finding variable networks and weak CPUs to be limiting factors in a world where *most* of the world's browsers run on cheap Android phones.

Nobody's winning.

Can N Really = ~2?

The RUM Archive and Chrome OT data paint a bleak and cloudy portrait, shrouded in mysteries.

The first mystery is distributional. If N = ~2, then SPA architectures are an industrial-scale mistake. Medians and averages don't tell us much about distributions, and so if we expect that some sessions and application types feature Ns well above 2. The clear implication is that *most* sites built as SPAs shouldn't be.

The second mystery is definitional. Many interactions might not be captured in the soft-navigations mPulse and Chromium track. Conceptually we can consider a soft-nav like a primary view transition, moving from one screen to the next. Non-route updates happen regularly, though, including through mechanisms like infinite scrolling. Given the propensity of SPA tools and frameworks to fail to reconstruct URL state for in-page updates, it seems likely that some fraction of server-synchronised view updates may not be represented via these metrics. How big is this dark matter? We should try to find out.

This quandary leads to a set of sub-investigations. For example:

- How should pure view-layer updates (e.g., expanding/collapsing an accordion or switching a tab) be modelled in our data?
- What criteria should we use for including an interaction in our value of N? We definitively exclude browser-handled interactions like default scrolling, but when scrolling leads to content changes…what then? And how to think about script handlers for input, e.g. for spell-checking?
- Can we produce per-vertical distributions from our aggregate data? I.e., exemplar session depth histograms for e-commerce, productivity, and news?

The final major question is what to do with this information now that we're starting to be able to see the picture more clearly. Data-driven interventions like Google's Page Experience updates built on the data derived from Core Web Vitals to help shape industry behaviour. Given the unfortunate recent levelling-off in CWV pass rates, can a session-depth-based understanding of site construction help us steer teams towards more appropriate architectures *en masse*?

I don't know the answers to these questions, but it seems urgent for the web performance community to find out given that it seems probable that we can build a global understanding of session depths in privacy-preserving ways. It's my hope that this sort of characterisation can steer teams away from FOMO-based narratives that land them in the deep end of overly-complex stacks whose selling points are based on problems those sites may never encounter.

If there are high callings for the Web Performance community, steering teams away from needing our help and, in the process, making life better for users, would seem near the top of the stack. I suspect this is closer to our grasp now than at any time before, and the whole community can contribute to answering these deep questions.

- How to contribute to this calendar
- A project by Stoyan Stefanov
- Powered by WordPress
- Grab the RSS feed
- Part of Planet Performance
- Check out the Podcast
- ... and the Feed
- Bsky updates: @stoyan.me

Archives

- 2025
- 2024
- 2023
- 2022
- 2021
- 2020
- 2019
- 2018
- 2017
- 2016
- 2015
- 2014
- 2013
- 2012
- 2011
- 2010
- 2009