Turbocharging Walmart.com: Speed without compromise

Vasanth Krishnamoorthy

vasanth_k | vasanthk

Walmart Labs

Focus Areas

Approaching Performance

7 Key Optimizations

Guardrails for Performance

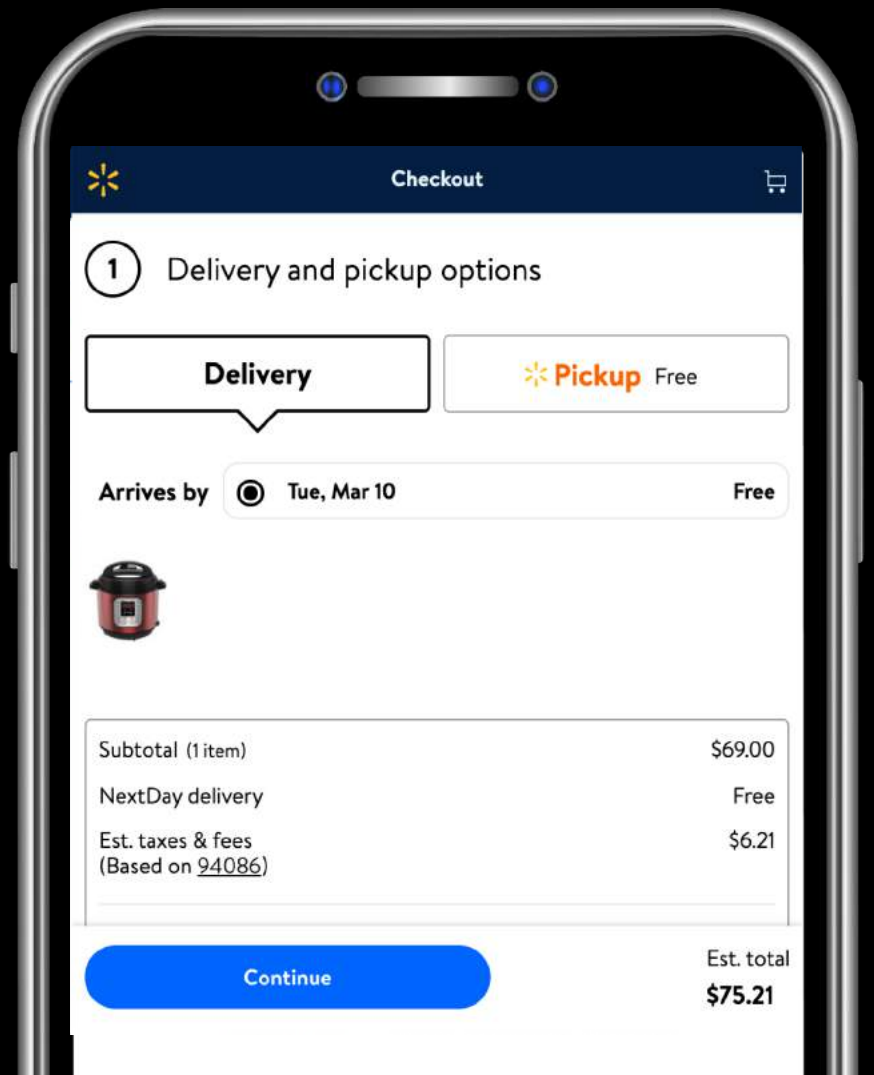
100 million+
customers per month

Over 30%
growth YoY

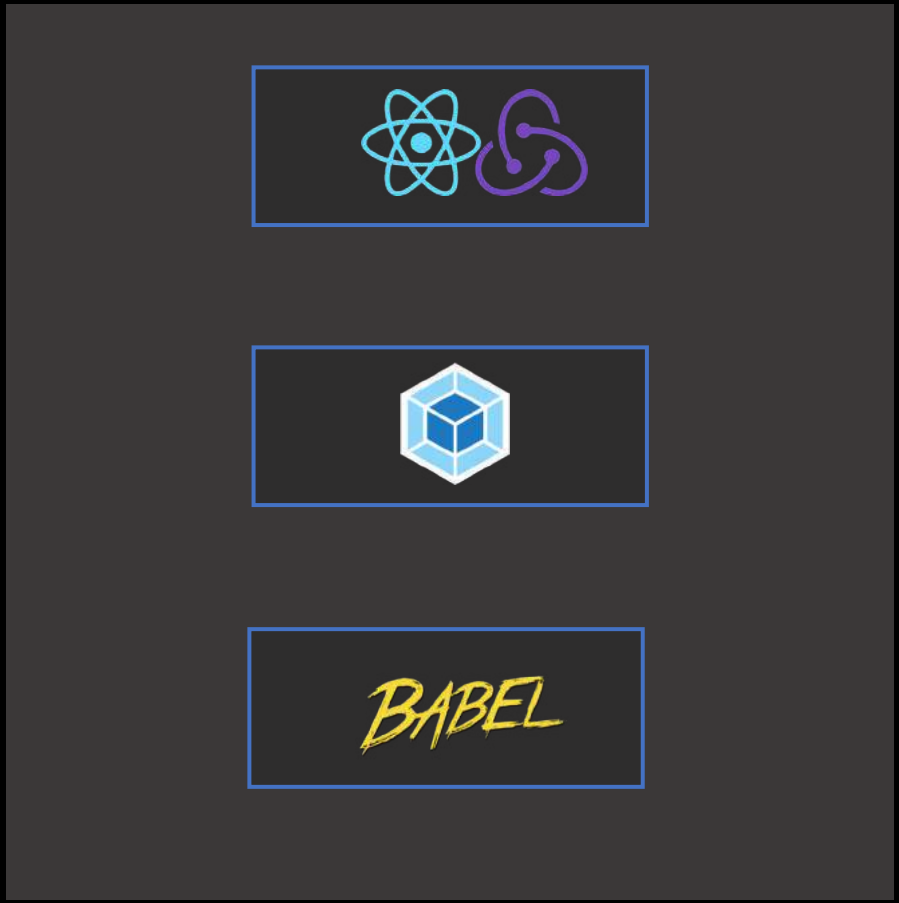
Performance as a Measure Of User Happiness

Typical Customer Journey

Checkout



Tech Stack



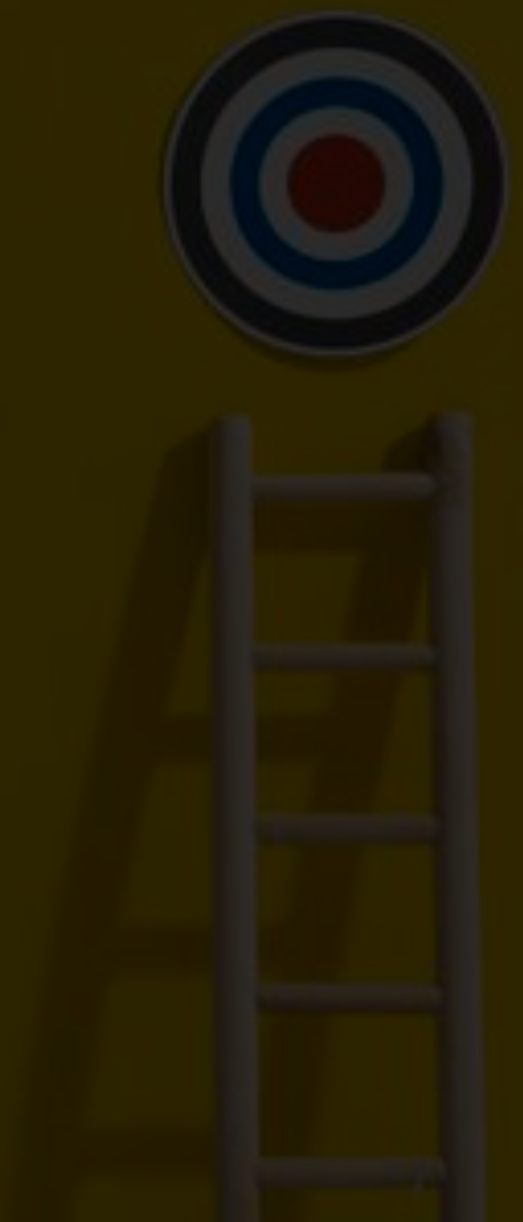
Customer Journey

42% Improvement

TTI - 95th %ile mobile web

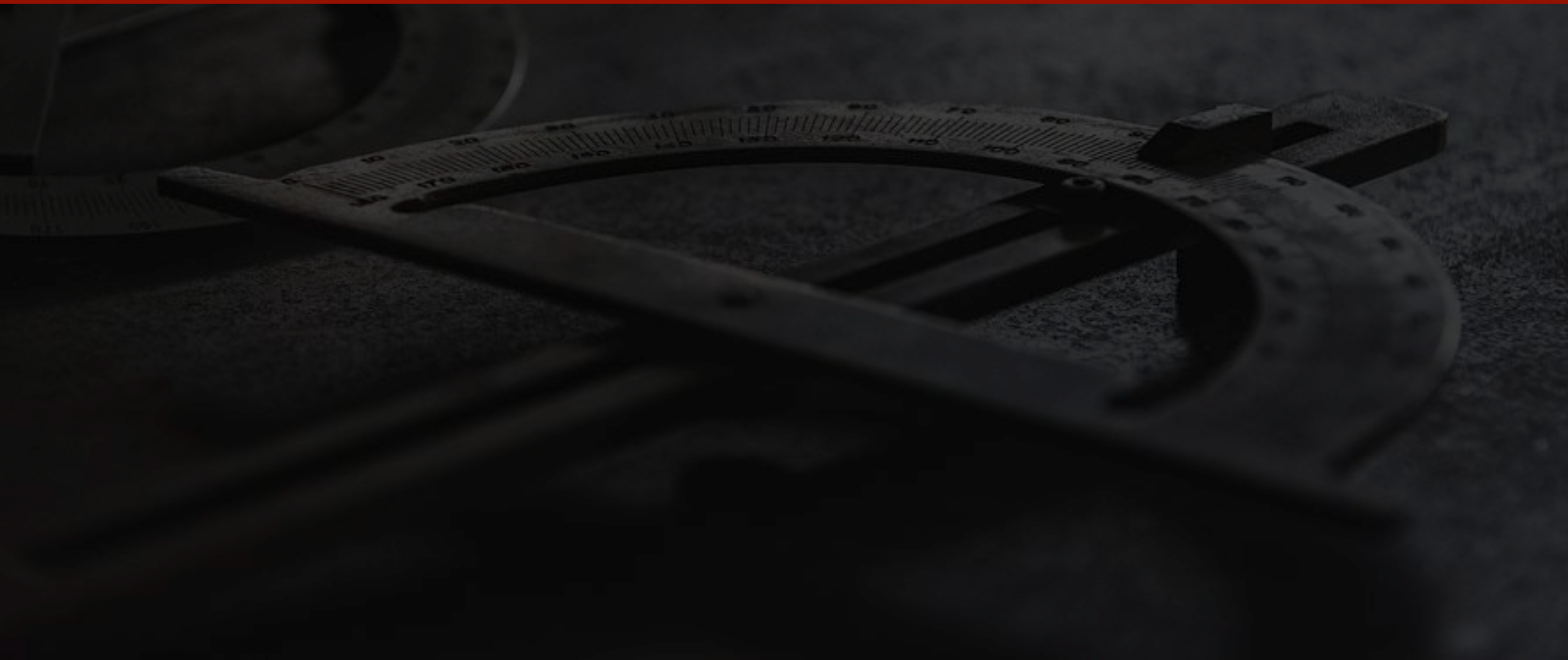
PART 1

Approaching Performance



Where do you start?

“You can’t improve what you don’t measure”

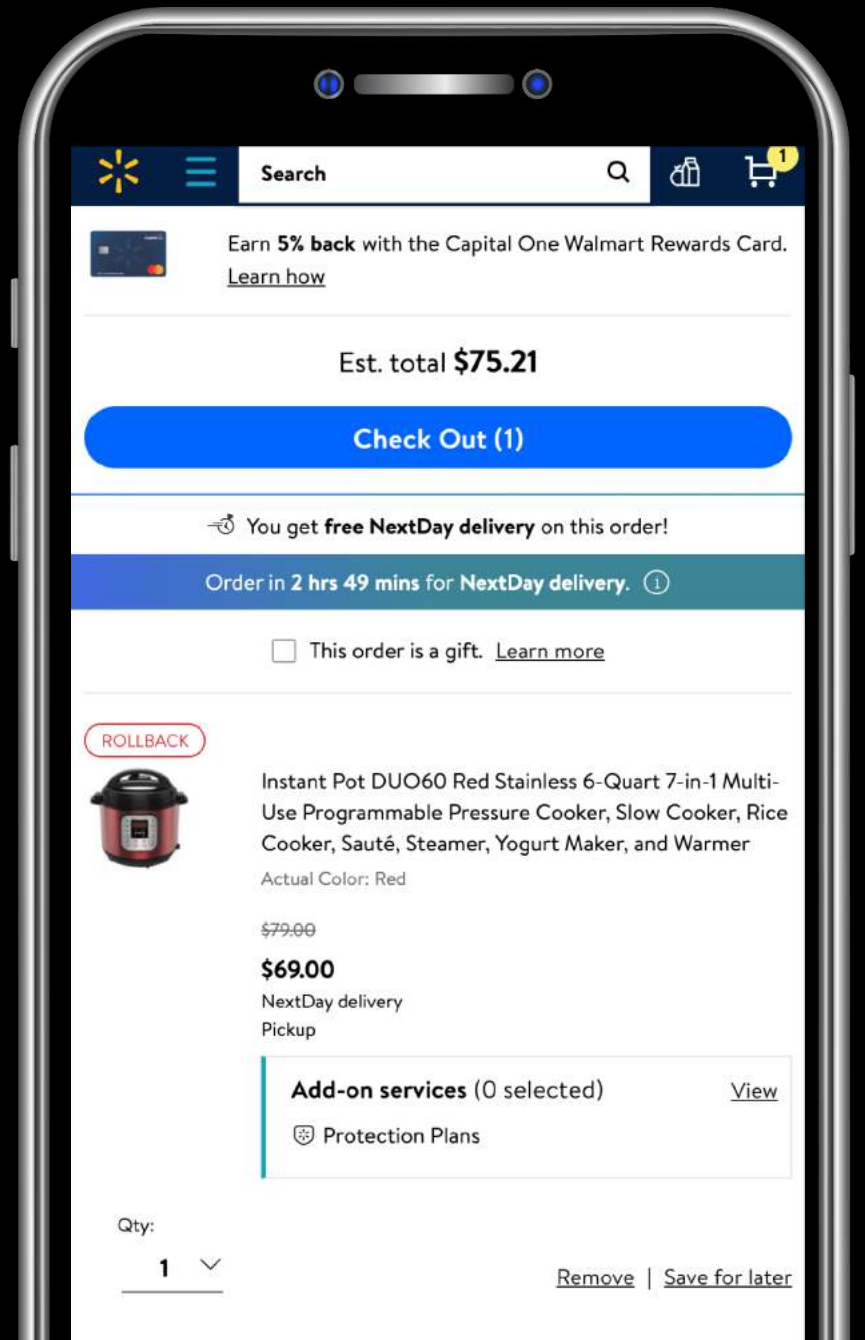


Measure What Matters

TTI – Time to interactive (custom)

TTFB – Time to first byte

TLOAD - Page is fully loaded (custom)



Define Your Scope

Define Your Scope & Constraints

01

No compromise to existing product features

02

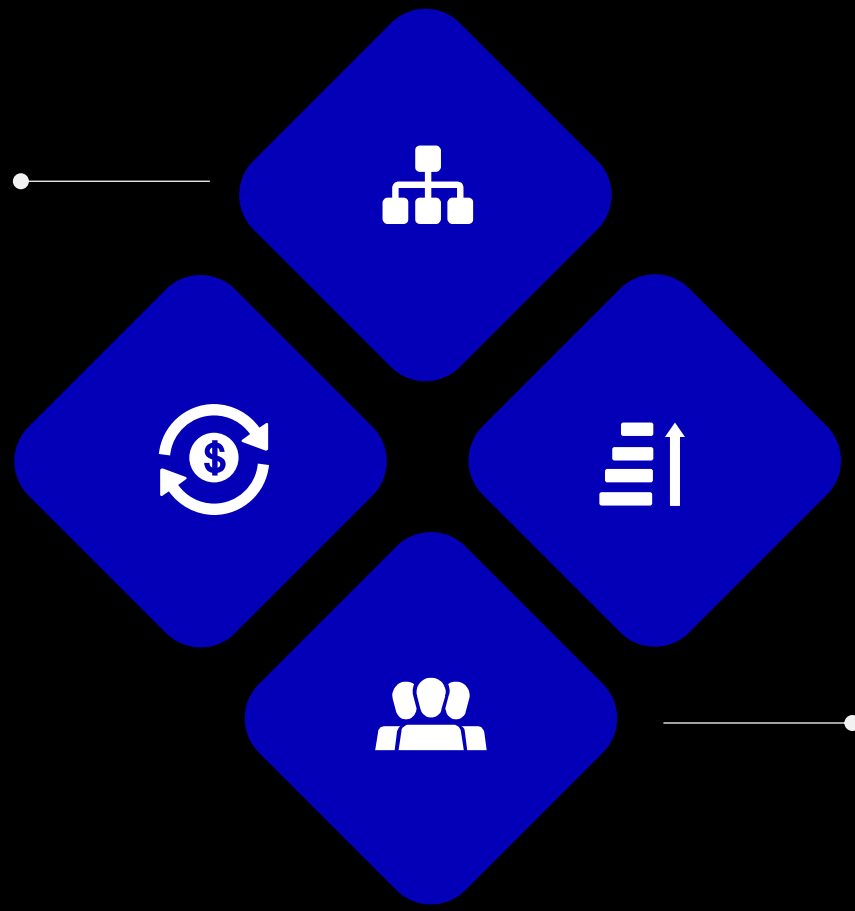
No slowdown on new product features being built

03

No big rewrites or tech stack changes

Identify The Right Opportunities

Empathize With Your Users



Understand common user flows

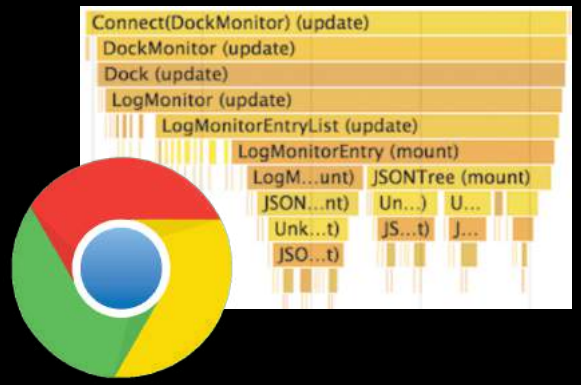
Revenue distribution by browsers & platform

Step progression

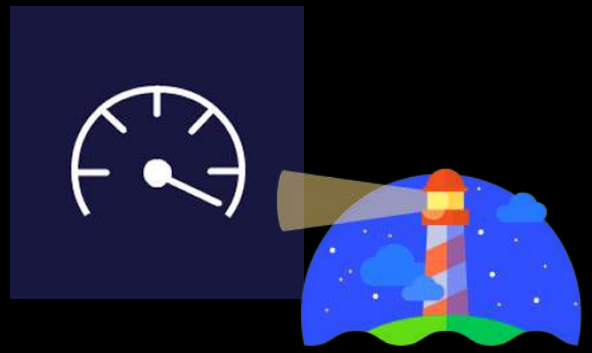
User distribution by browsers/devices/connection

Use data to understand user pain points

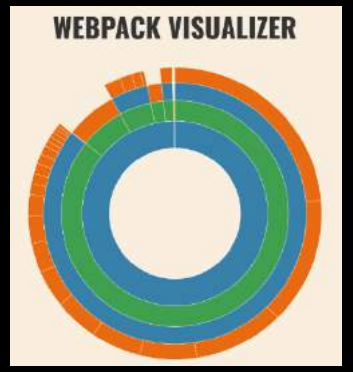
Audit Your Application



Chrome's Dev Tools



WebPageTest & Lighthouse



Webpack Bundle Analyzer

Make **the invisible** visible

Apply First Principles

Principles of Web Performance

1 Do only what is needed

2 Minimize round trip time

3 Optimize Perceived Performance

PART 2

7 Key Optimizations

Do only what is needed

Reduce Bundle Size

1 Code Splitting & Lazy Loading

“Get it when you need it”

- Component centric code splitting via dynamic imports (ES2020)
- Code split user/item specific feature code
- Lazy load components which are below the fold

TTI **18.2%**

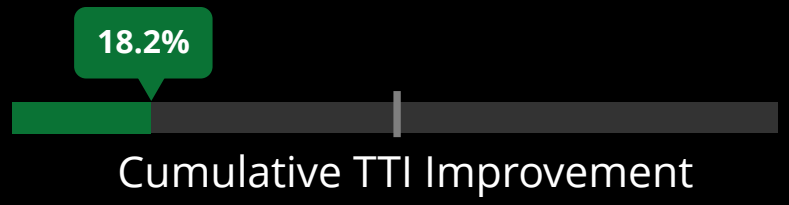
1 Code Splitting - Learnings

Aggressive code splitting increased overall bytes downloaded

- Duplicates in chunks
- Reduced Compression

Our sweet spot

< 15 split bundles



2 Slim Down Libraries

From	Switch To	Gains (compressed)
Moment.js	date-fns	50KB
React v15	React v16	15KB
React-Intl	Custom utility	14KB
Recompose	React API / Hooks	5KB

Upgrade from Webpack v3 to v4 reduced bundle size by 10%

TTI **9.7%**

Cumulative TTI Improvement

3 Differential Serving

Enables us to serve modern JS code to users

Why do we need it?

- Transformed ES5 code is verbose (more KB)
- Cut down on polyfills needed (~35KB for us)

3 Differential Serving

Module No-Module Pattern

```
<!-- Browsers with ES module support load this file. -->

  <script type="module" src="main.mjs"></script>

<!-- Older browsers load this file (and module-supporting --
<!-- browsers know *not* to load this file). -->

  <script nomodule src="main.es5.js"></script>
```

Cumulative TTI Improvement

3 Differential Serving - Learnings

Problem

On a few browser versions both ES5 & ES6 scripts were downloaded/executed leading to degraded experience for those customers

Cumulative TTI Improvement

3 Differential Serving - Learnings

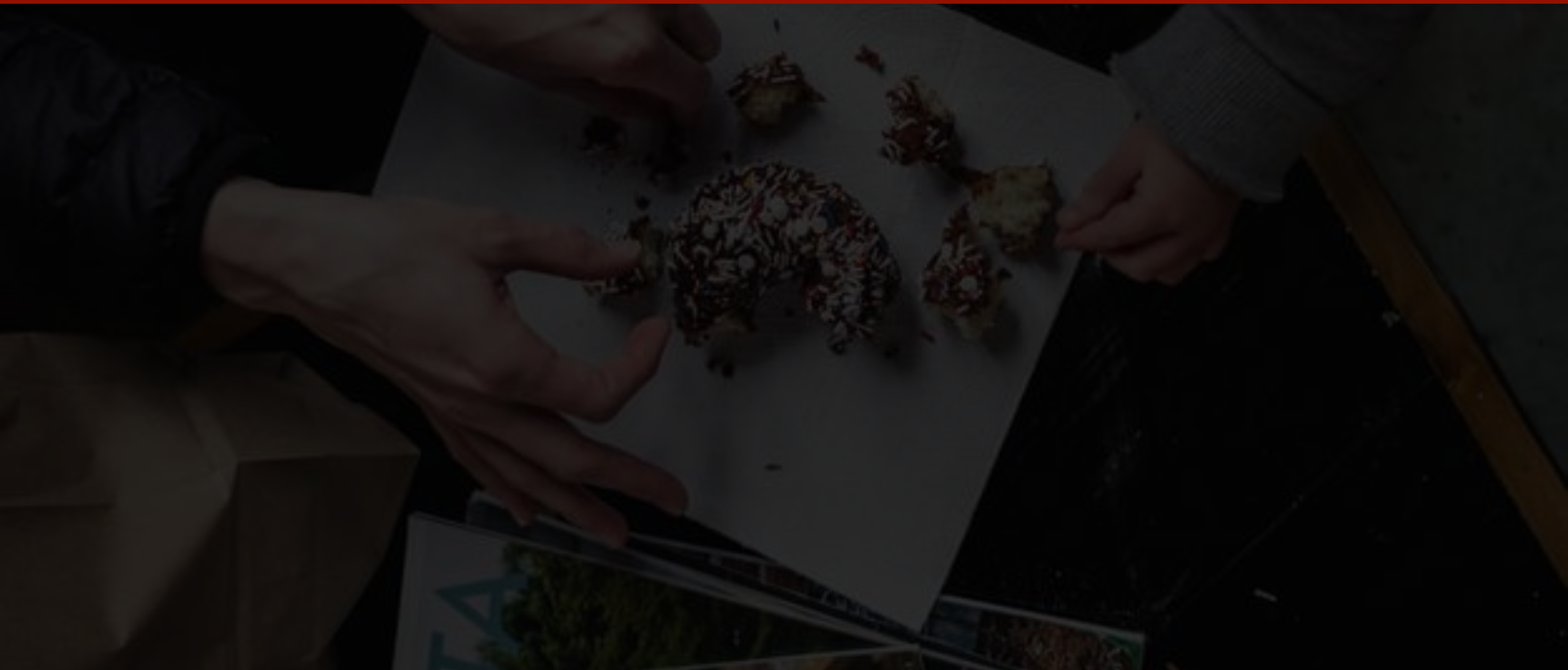
Solution

On Server side, we check for the user agent passed and depending on whether the browser supports modern syntax or not we include the right bundle into the page.

Cumulative TTI Improvement

Minimize round trip time

Sharing is caring for our users



4 Shared Bundles

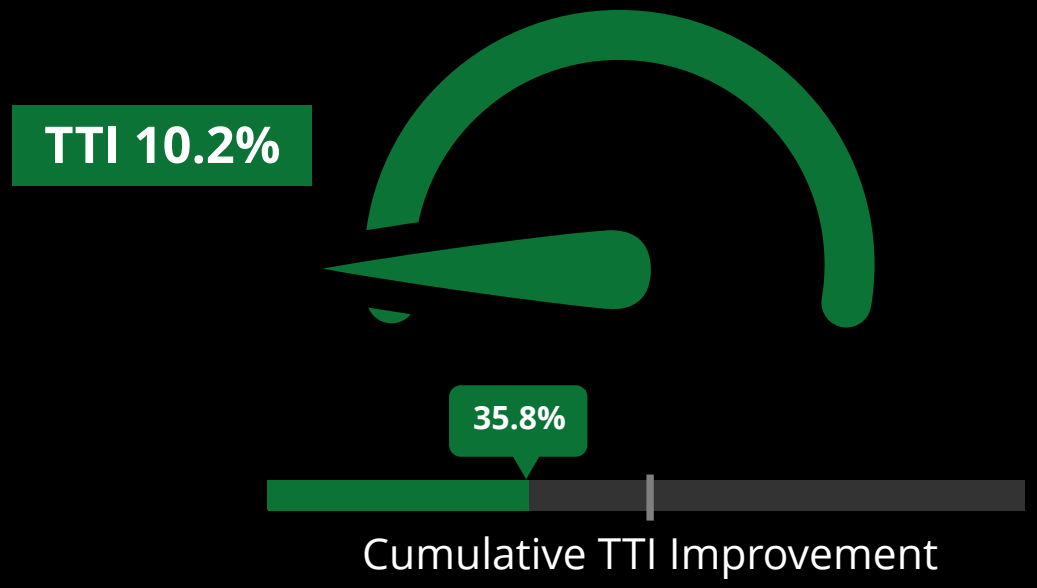
1P/3P bundles

- Shared across whole site

Functional shared bundles

- Shared across Cart & Checkout

Webpack DLL Plugin



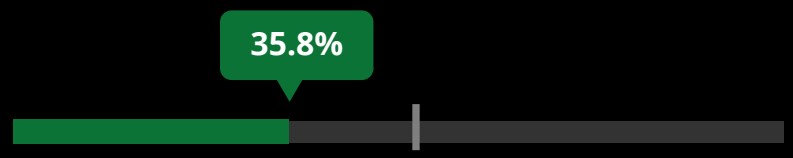
4 Shared Bundles - Learnings

1P/3P shared bundle

- Needs unification of package versions across applications
- Updates to shared bundles require coordination with multiple teams

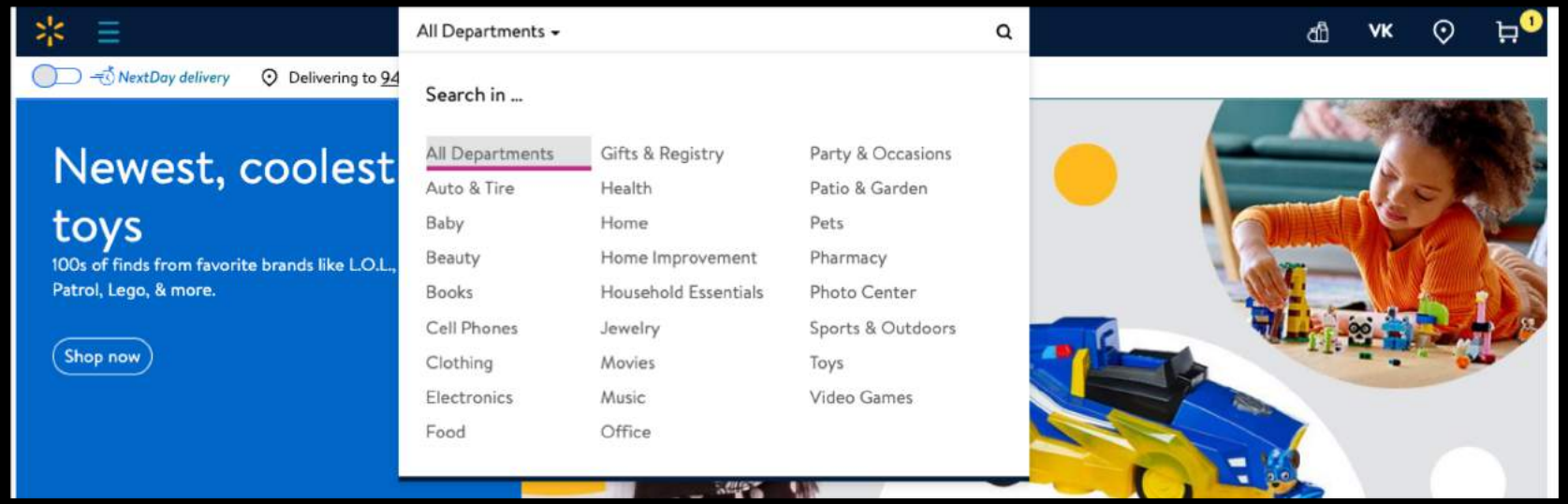
Functional shared bundles

- Updates frequently and requires changes across all shared web-apps
- Testing/validations & Release effort



Cumulative TTI Improvement

5 Shared Header/Footer



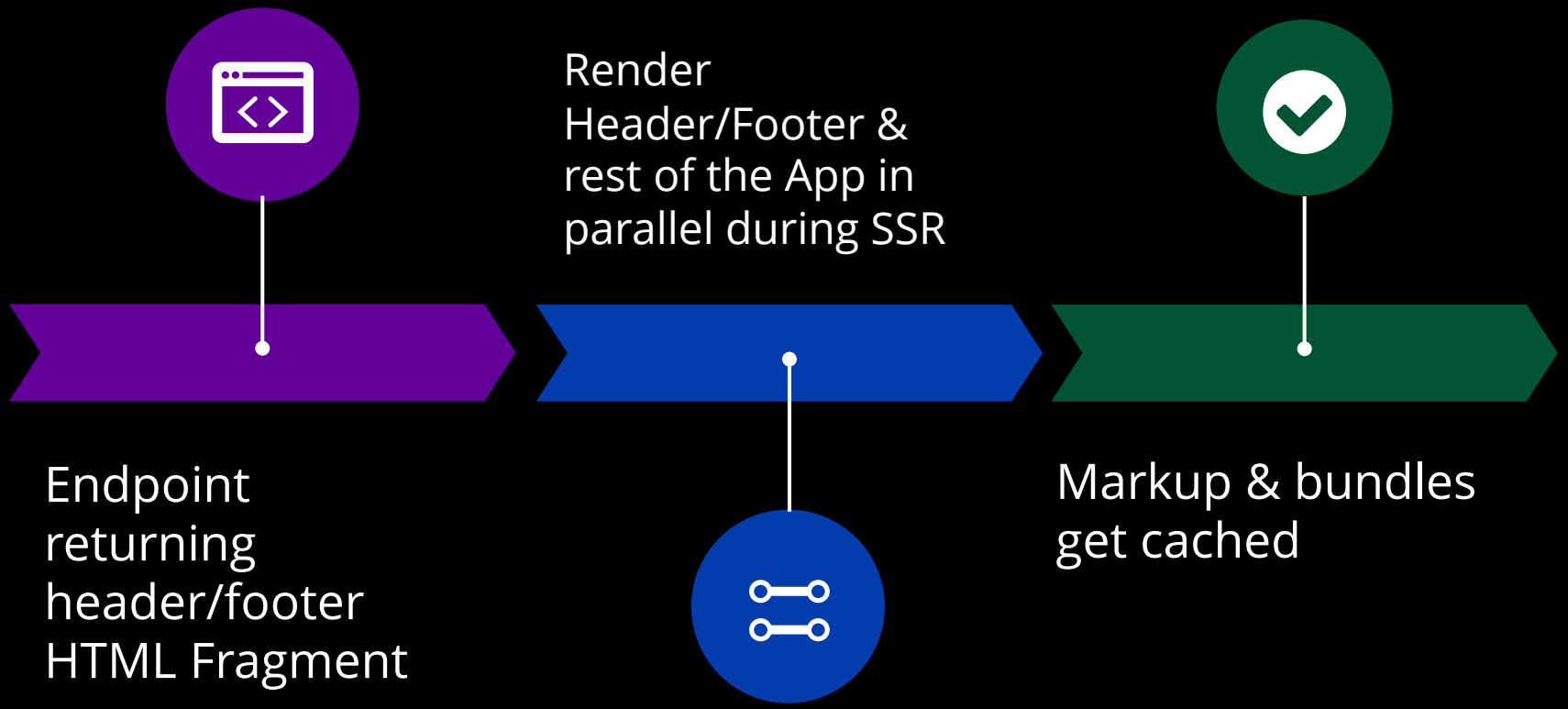
Problem

Header/Footer package was bundled into each app leading to bloat

Any change required testing/validation & deployments across for all teams

5 Shared Header/Footer

Solution



5 Shared Header/Footer

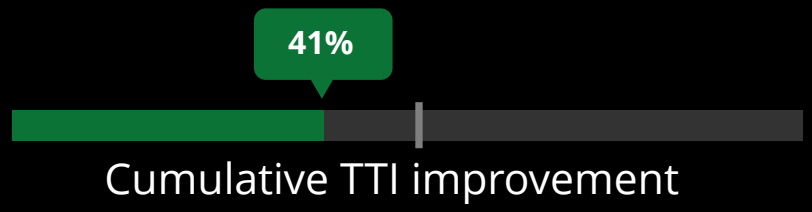
Optimization

Reuse existing code and render with React on Server Side

Use Vanilla JS for handling events on client

70%

reduction in client side JS



6 Brotli Compression

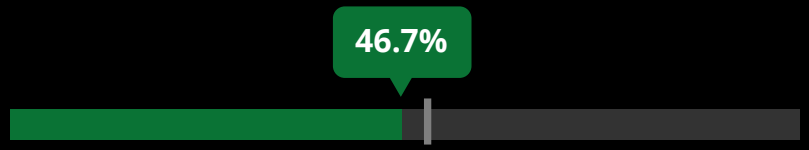
“It’s like GZIP on steroids”

Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *
			4-48			
			1 49		10-35	
	12-14	2-43	2 50	3.1-10.1	1 36-37	3.2-10.3
6-10	15-79	44-71	51-79	3 11-12.1	38-65	11-13.1
11	80	72	80	3 13	66	13.2
		73-74	81-83	3 TP		13.3

12% smaller bundles than GZIP

TTI 9.8%

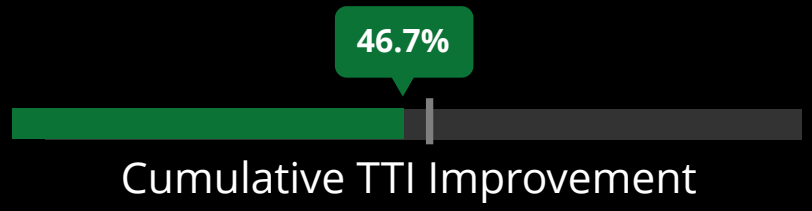


Cumulative TTI Improvement

6 Brotli Compression – Learnings

Dynamic compression can be slow

For Best Perf - Pre-build compressed assets and serve it from a CDN to save the runtime cost.



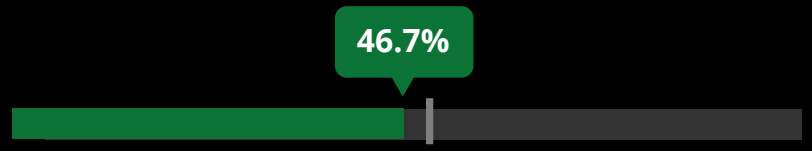
6 Brotli Compression - Learnings

For **Differential Serving**

Brotli compressed ES5 bundles pretty well

ES6 vs ES5 bundle difference dropped from
10% to 4%*

*YMMV



Cumulative TTI Improvement

Optimize Perceived Performance

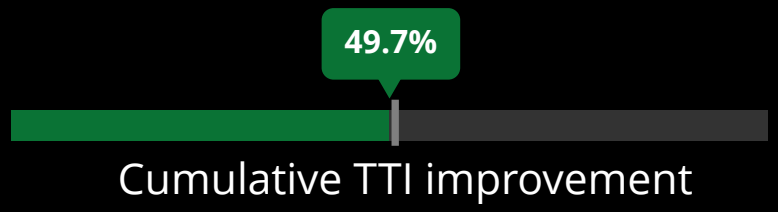
7 Leverage Priorities & Resource Hints

HTML parsing

HTML parsing paused

Script download

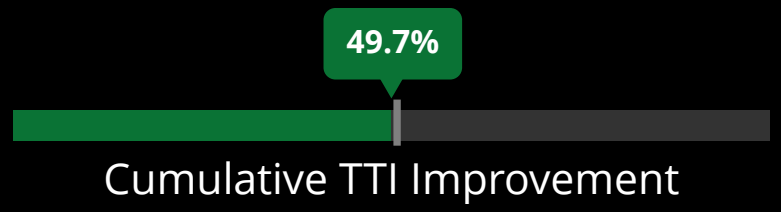
Script execution



7 Leverage Priorities & Resource Hints

```
<link rel="preload" href="main.js" as="script">
```

Tells the browser to download and cache a resource to have them available for execution when it is needed



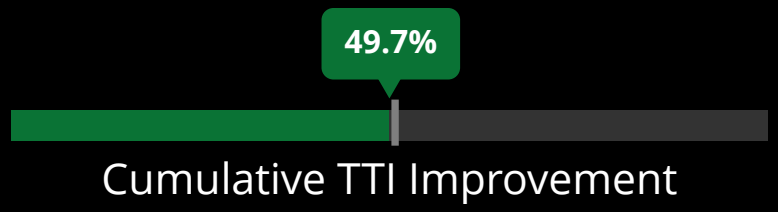
7 Leverage Priorities & Resource Hints

dns-prefetch

DNS lookup

preconnect

DNS lookup, TLS negotiation, and TCP handshake



Can we make it faster?

“The fastest HTTP request is the one not made”

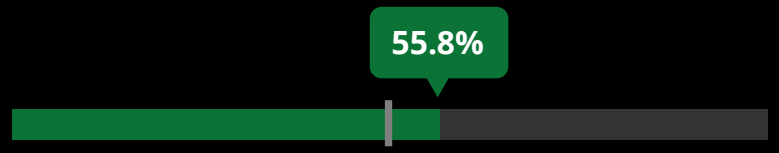
8 Prefetch

```
<link rel="prefetch" href="bundle.js" as="script">
<link rel="prefetch" href="main.css"  as="style">
```

Downloads scripts with lower priority & stores it in prefetch cache

- Cached for at least 5mins
- Does not execute JS

TTI **12.1%**



Cumulative TTI Improvement

8 Prefetch - Learnings

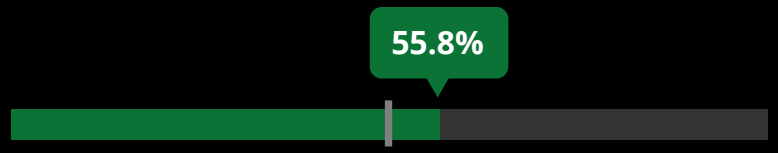
Problem

Impacts current page's load times

Workaround

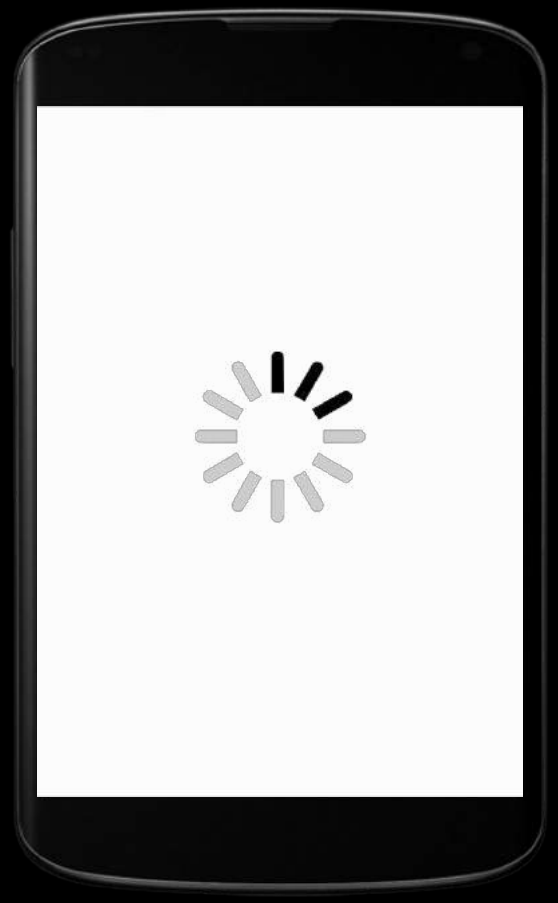
- We include prefetch tags into the page after *onLoad* event is fired
- We do not prefetch if the user has data saver on

```
if('connection' in navigator && !navigator.connection.saveData){.. }
```

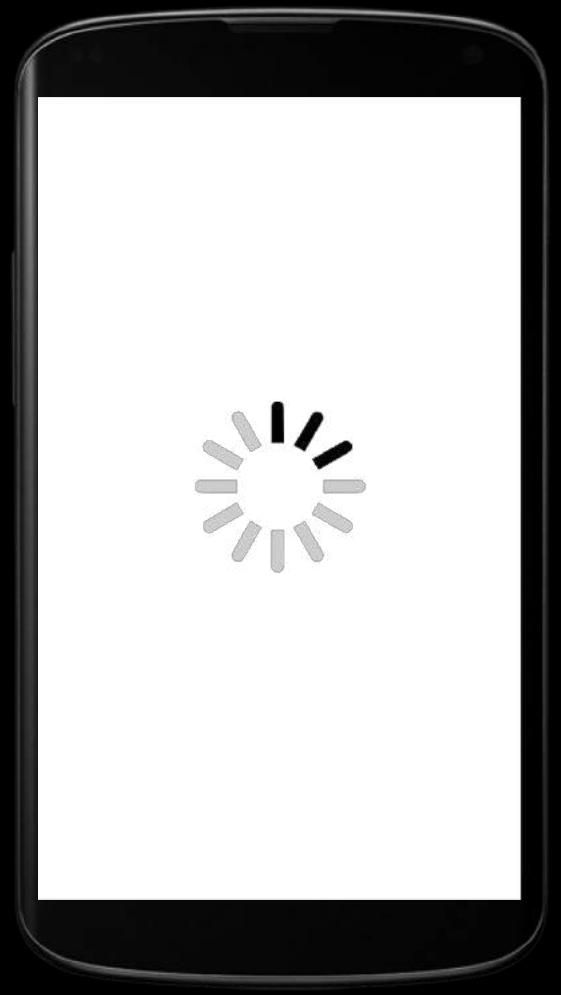


Cumulative TTI Improvement

Video Comparison



Before



After

Key Takeaways

Perf Optimizations – Key Takeaways

Reduce Bundle Size
- Code Splitting & Lazy Loading
- Slim down libraries
- Differential Serving

Shared bundles
- 1P/3P & functional shared bundles
- Header/Footer HTML fragment

Better compression
- Brotli

Priorities & resource hints
- Prefer defer over async
- dns-prefetch & preconnect
- Prefetch

Font Optimization
- Remove unused glyphs & styles
- WOFF & WOFF2 for better compression

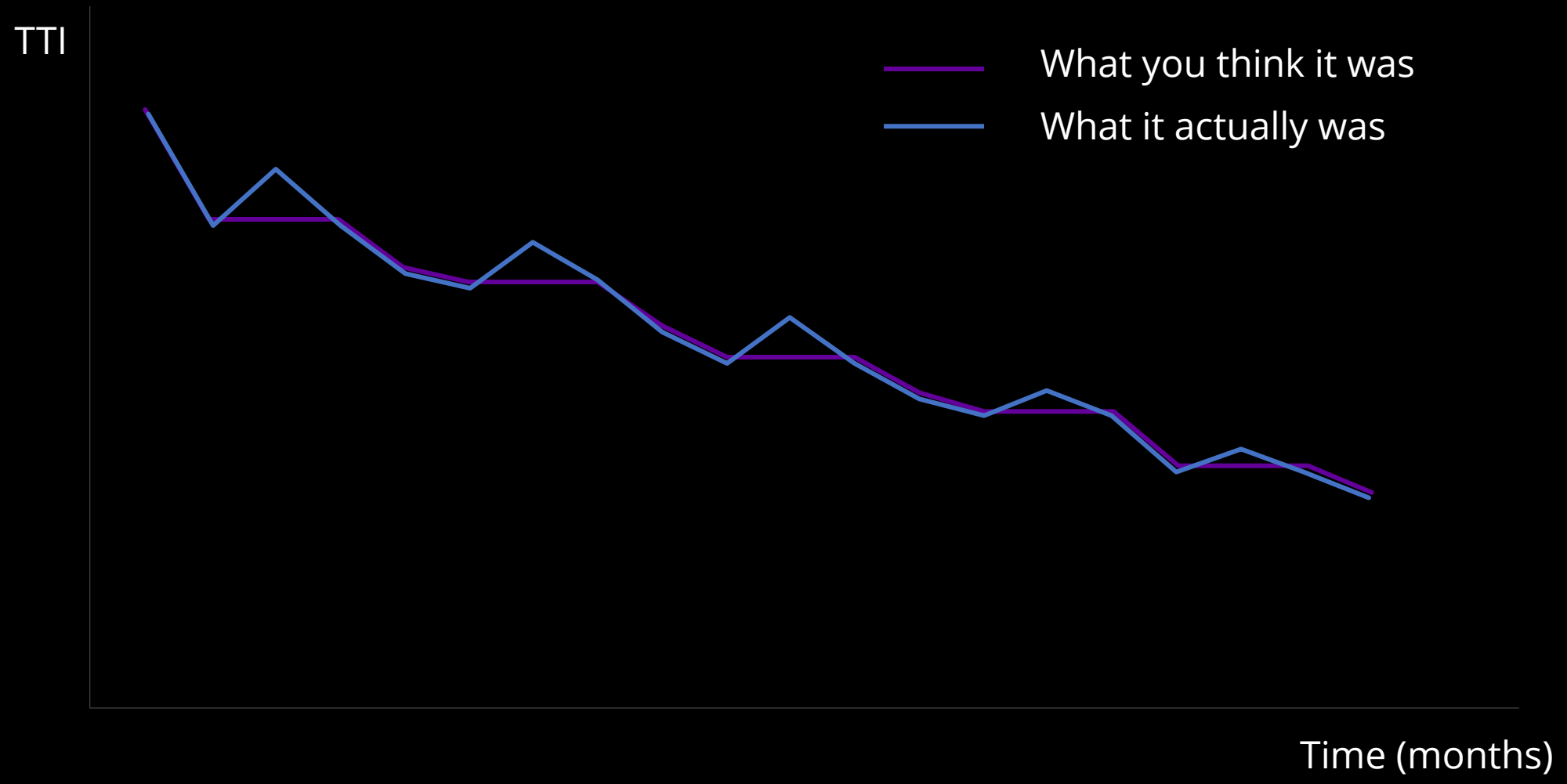
Image
- Lazy load images
- WEBP
- SVG

Redux State transfer optimization
- Send state to client in an inert tag

Other Cleanup
- Code which you have always wanted to delete. You know what they are ☺

Cumulative TTI Improvement

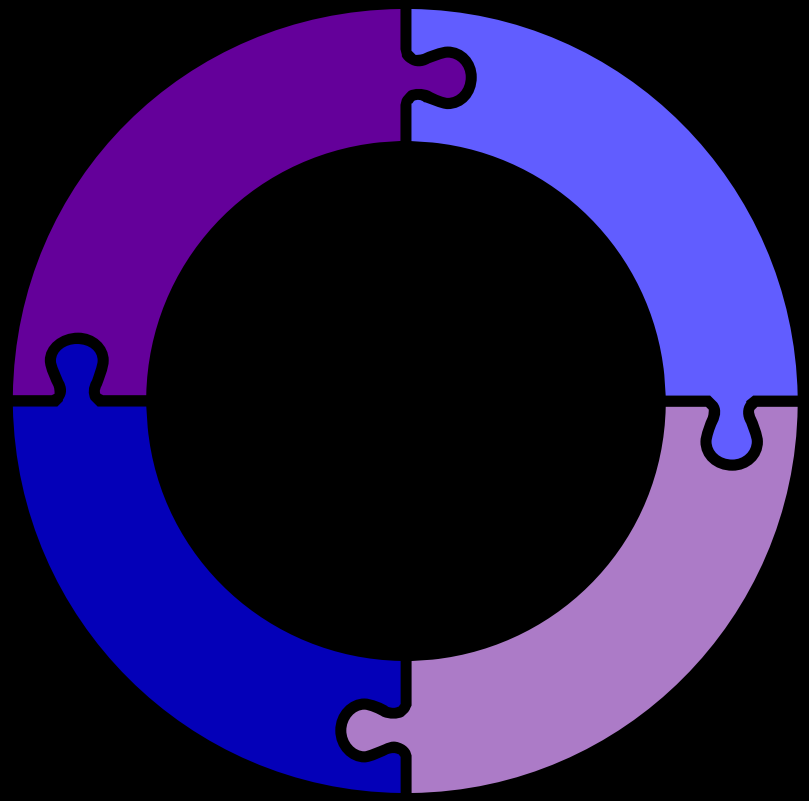
Perf Improvements Over Time



Challenges

Performance Regressions

3rd Party Scripts: Ads, Marketing Tags



Time to complete A/B Tests

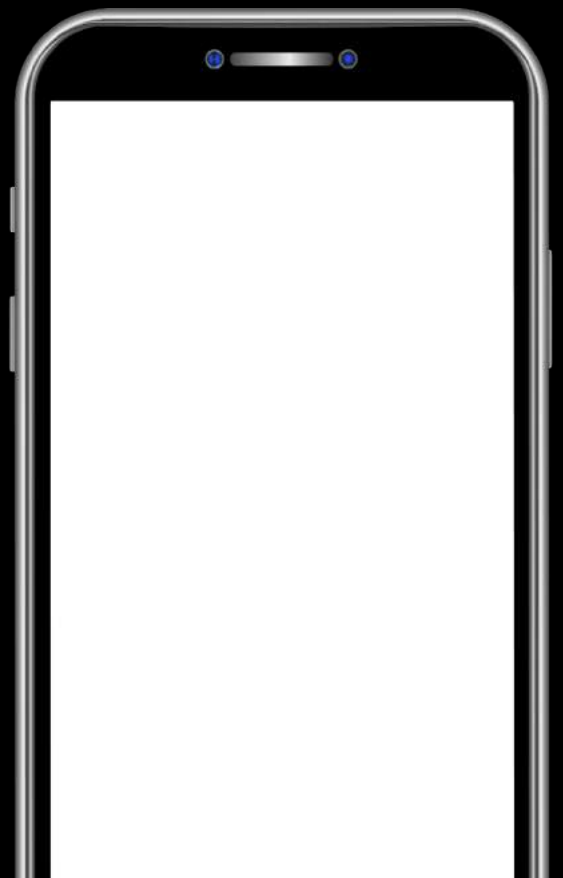
Cross-team initiatives

PART 3

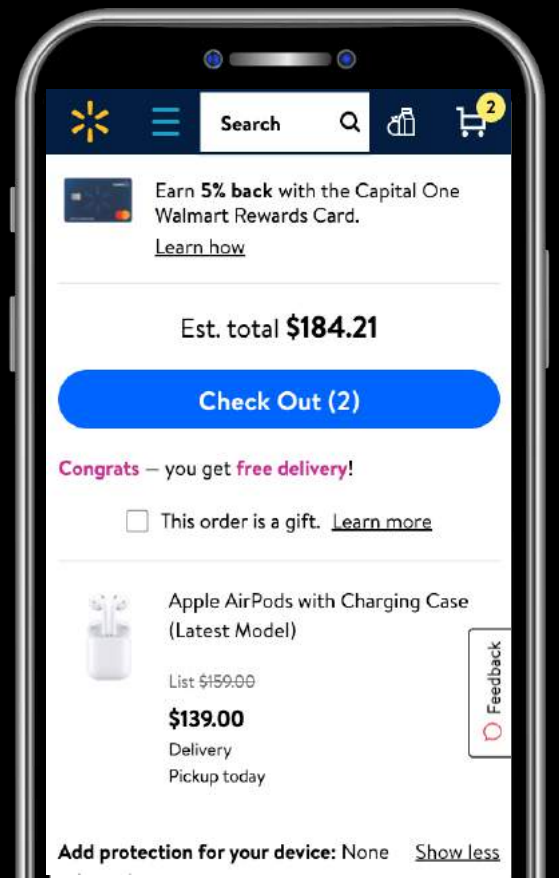
Guardrails For Performance

Performance Budgets

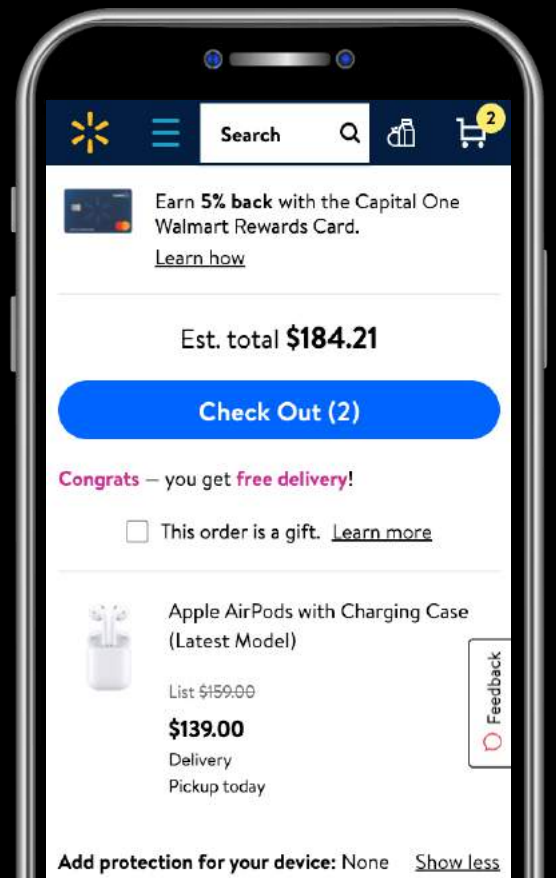
TTFB



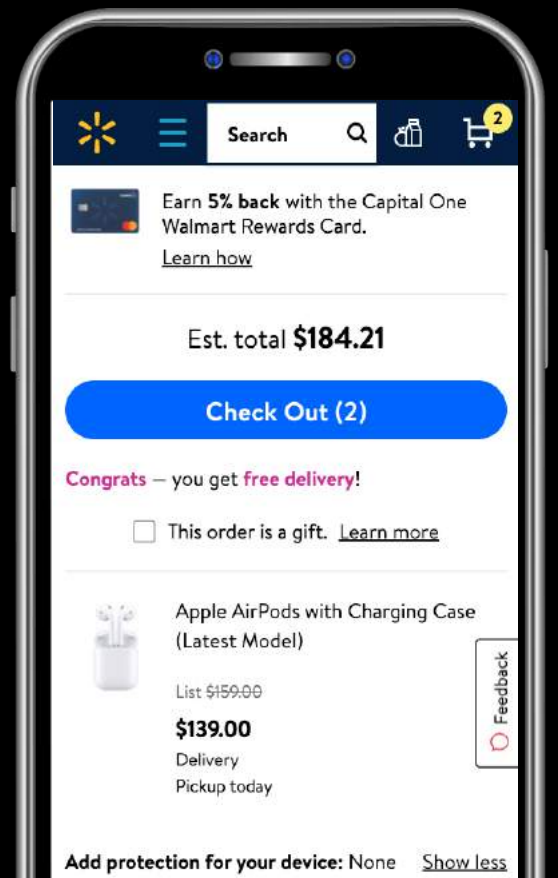
Speed Index



TTI



Page Complete



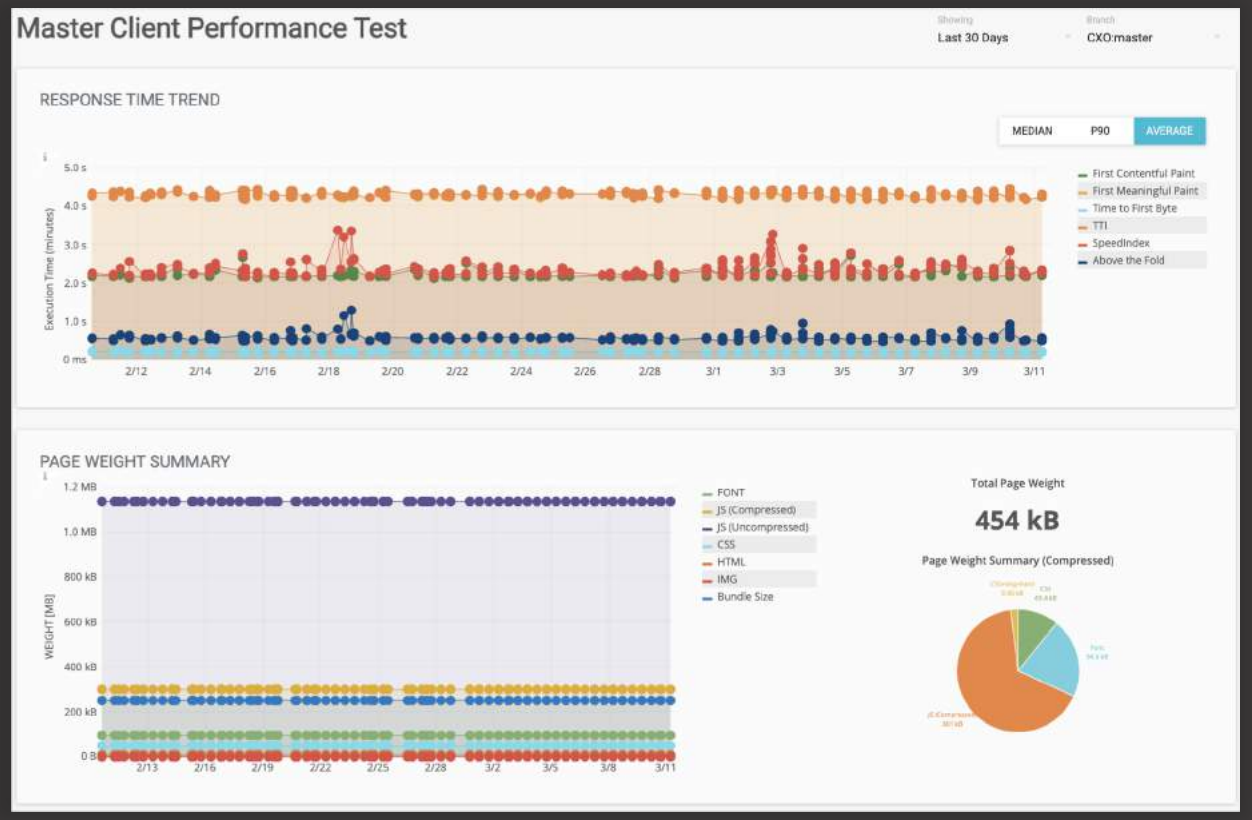
Performance Metrics Per PR

Bundle Size check at PR

✓ [PR] R-Transaction/r-transaction_checkout_master - bundlesize Success Details

```
[
  {
    "path": "./build/vendor.js",
    "maxSize": "30 kB"
  },
  {
    "path": "./build/chunk-*.js",
    "maxSize": "10 kB"
  }
]
```

Performance Metrics Per PR



- Lighthouse metrics for each PR
- View perf metrics over time

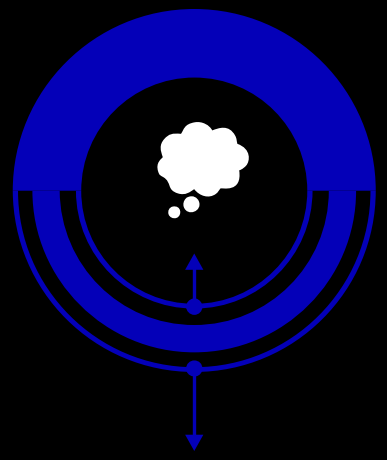
Compare Metrics Across Branches

COMPARISON RESULTS

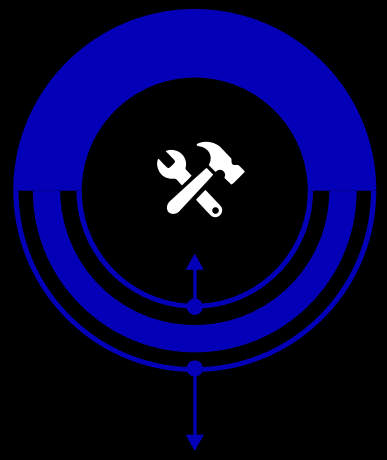
METRIC	1251304 PROD Cart:master 03/11/2020 06:12AM	1248735 Cart:master 03/10/2020 06:15PM	DELTA +/-
cATF	6.431 s	6.527 s	0.096 s (+1.49%)
TTI	10.300 s	10.153 s	0.147 s (-1.42%)
SpeedIndex	8.168 s	8.056 s	0.112 s (-1.37%)
TTFB	2.176 s	2.162 s	0.014 s (-0.64%)
JavaScript (Minified & Uncompressed)	1134.3 kb	1137.1 kb	2.8 kb (0.24%)

- Teams can compare branch performance to production performance
- Click through commits and see what caused the degradation
- Results used to accept or reject release

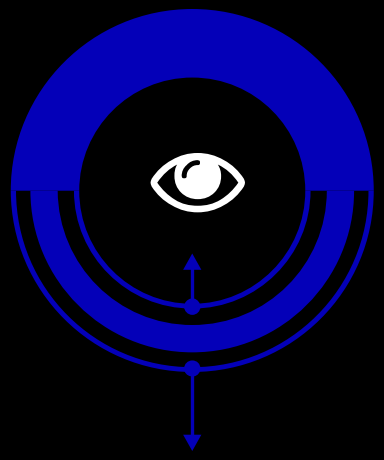
Sustaining A Performance Culture



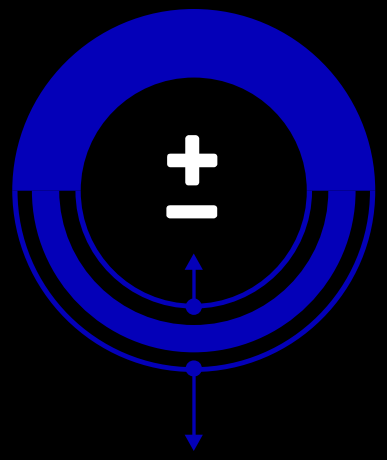
Embed performance thinking early in the product development process



Use tooling and data to help drive decisions on performance tradeoffs



Maintain gains by monitoring key metrics, tooling and having guardrails



Recognize performance is hard and there will be tradeoffs

The Team

Bryan Morgan

Ah Hyun Cho

Hiren Patel

Madhav Deverkonda

Test Armada & Torbit Team

Denys Mikhalenko

Gauri Shankar

Rodrigo Delgado

Jon Campbell

Uma Mahesh

Cory Dang

Meet Parikh

Megha Gupta

Patrick Stapleton

Vijay Muniswamy

Future Plans

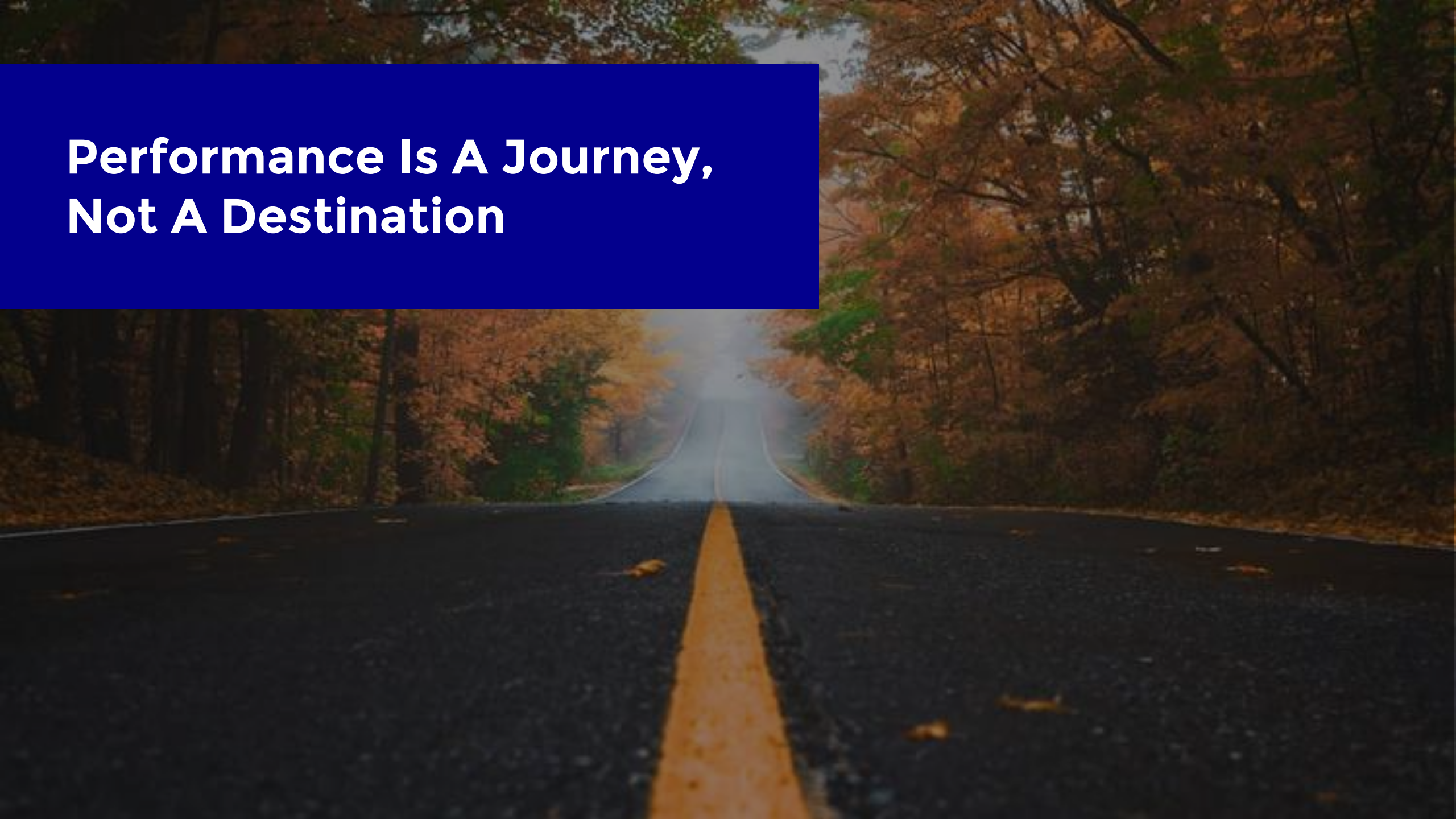
01 Progressive Web App (PWA)

02 Experimenting with alternative UI libraries

03 Streaming SSR

04 Different experiences based on Speed Profiles

Performance Is A Journey, Not A Destination



THANK YOU

vasanth_k | vasanthk

Walmart Labs