



KubeCon



CloudNativeCon

North America 2023

A Practical Guide to Debugging Browser Performance with OpenTelemetry

Purvi Kanal

“ Web performance is all about making websites fast, including making slow processes *seem* fast.

- MDN

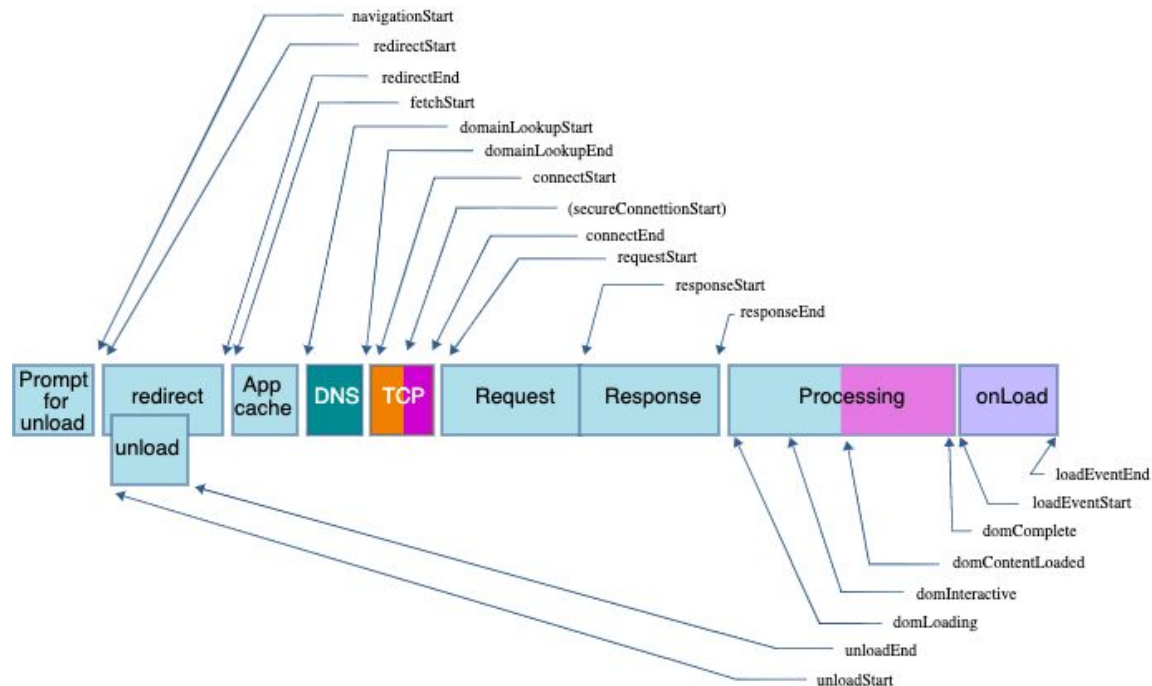
What are our web performance goals?

- Reducing overall **load time**
- Making the site **usable** as soon as possible
- Making the site **reliable and pleasant to use**
- **Responding and reassuring** the user when they take actions



Common web performance metrics

Page load time



Sites might load fast but not *feel* fast

Core Web Vitals

- Initiative by Google to provide unified guidance for measuring what a “good web experience” is.
- Play a factor in web search rankings



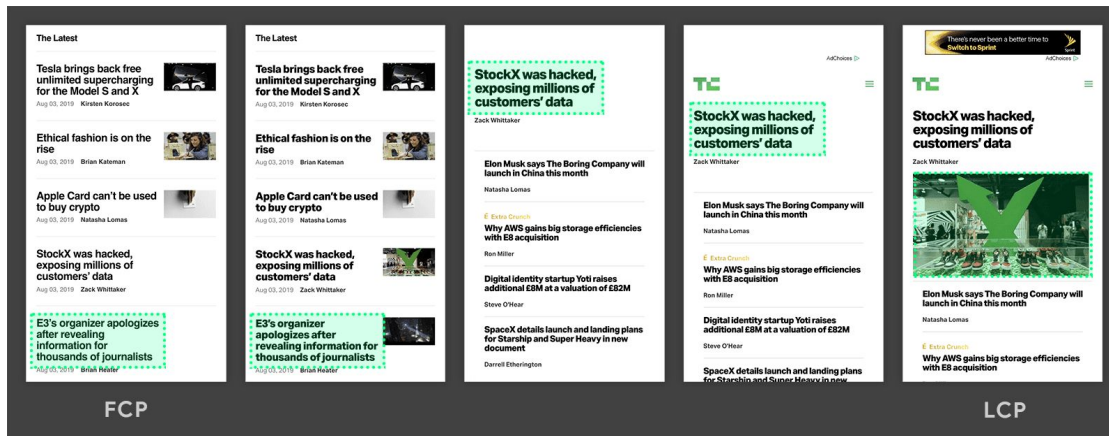
Largest Contentful Paint (LCP)

What is LCP?

The **render time** of the **largest image or text block visible within the viewport**, relative to when the page first started loading.

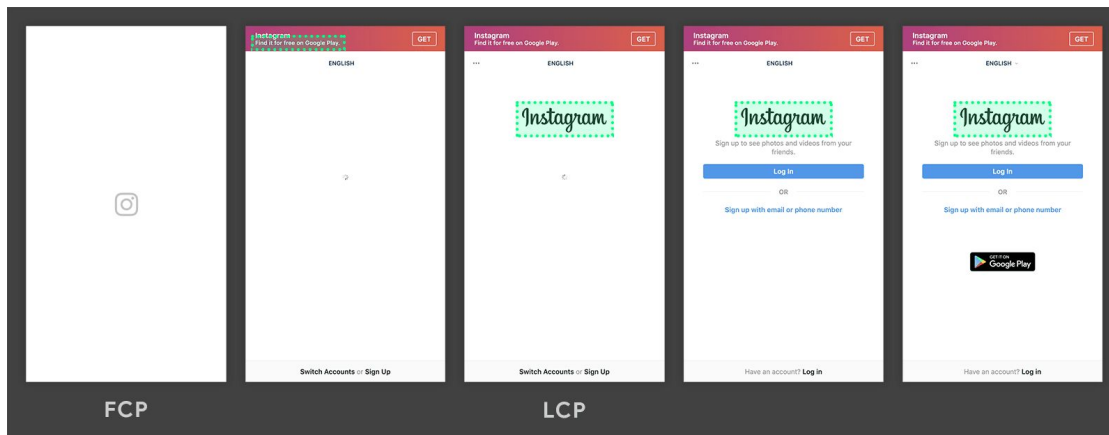
Web performance goal: How long does it take to render the “main content” of the page.





TL;DR: The **biggest** thing on the page is the “**main content**”.

LCP aims to measure the load time to the “main content” of the page.



Some issues with LCP

- **< 2.5 seconds for 75% of page loads**
- Based on studies from the early 2000s
- Sample sizes were 100ish people
- The “main content” of a page is usually more than any single element

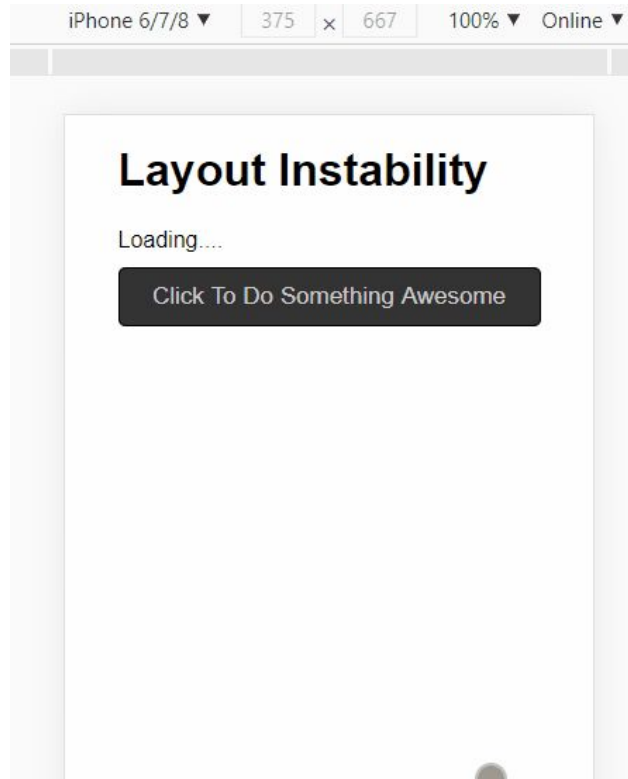


Cumulative Layout Shift (CLS)

What is CLS?

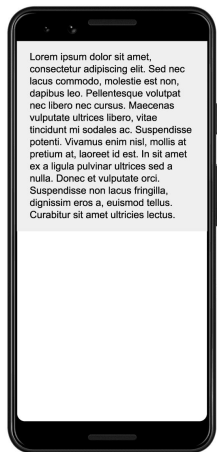
Measure of the **largest burst of layout shift scores** for every **unexpected layout shift** that occurs during the entire lifespan of a page.

Web performance goal: Making the site **reliable and pleasant to use**

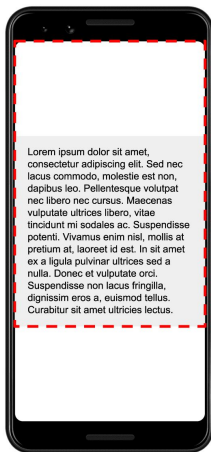


How is CLS calculated?

$$\text{CLS Score} = \text{impact fraction} * \text{distance fraction}$$

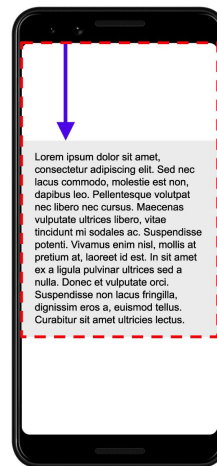
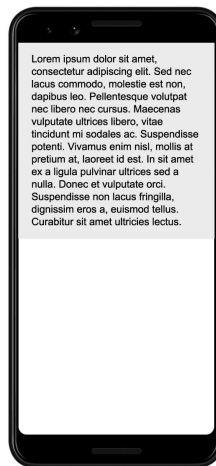


50% of viewport



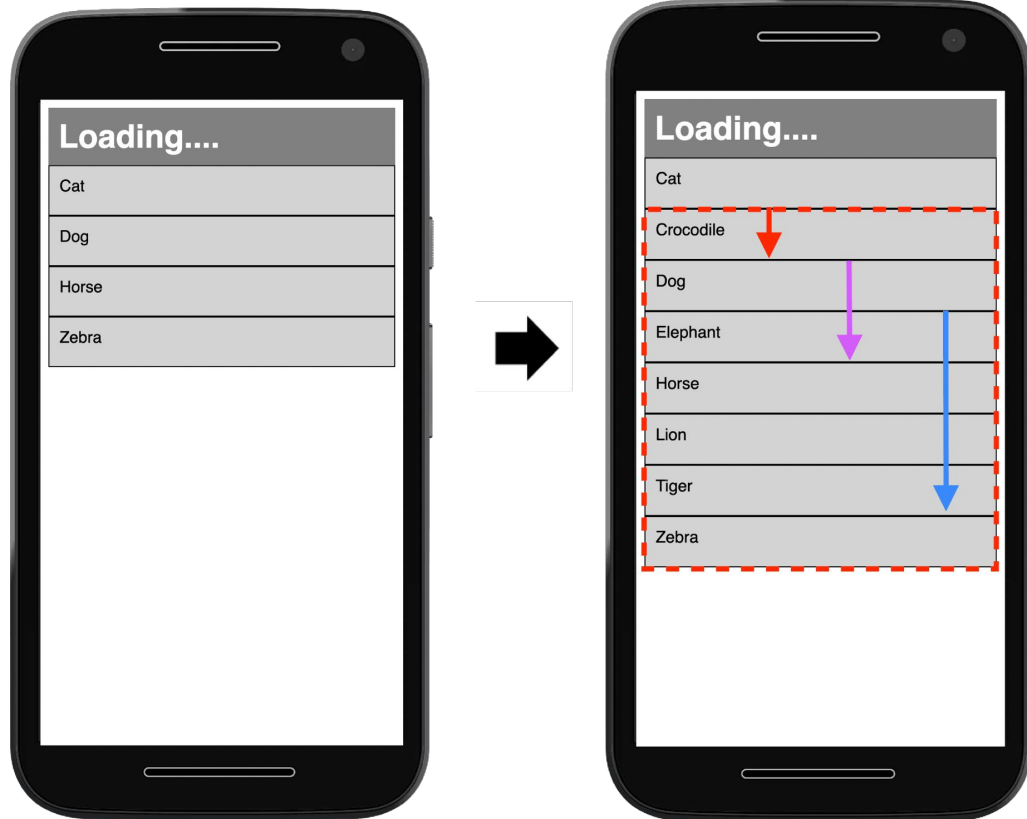
75% of viewport

Impact Fraction = 0.75



Distance Fraction = 0.25

$$\text{CLS Score} = 0.75 * 0.25 = 0.1875$$



First Input Delay (FID)

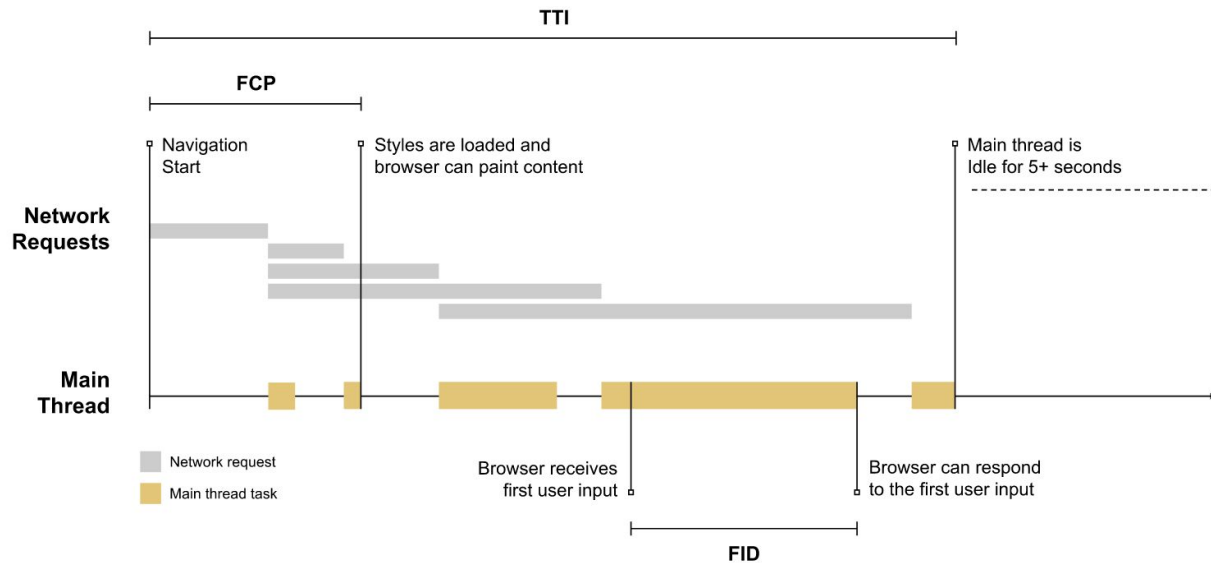
What is FID?

Measures the time from when a **user first interacts with a page** to the time when the **browser is actually able to begin processing** event handlers in **response to that interaction**.

Web performance goal: Responding and reassuring the user when they take actions



How is FID measured?



How to interpret FID values

- This metric will have a **large distribution of values**
- Some users will have no FID value
- Some users will have low FID values
- Some users will have high FID values
- It is important to focus on the **higher percentiles (P95 - P99)**



What do these measures tell us?

- What a user experiences when they **first load a site**
- That we're running **way too much JavaScript** 🤖 (who isn't though?)
- Whether our **assets are optimized**
- How **annoying** the site is to use

Where we usually end up: playing node_modules whack-a-mole



What do these measures leave out?

- Anything about JavaScript performance **after** initial page load
- Exactly how network requests are affecting page performance
- Anything **more specific** about the specific app / website in question

Most importantly: the recommended values for these metrics are **absolute** and spending time trying to optimize them might **not be the best way** to improve performance for **your specific user base**



So... what should we measure?

Spoiler alert: ✨ IT DEPENDS ✨

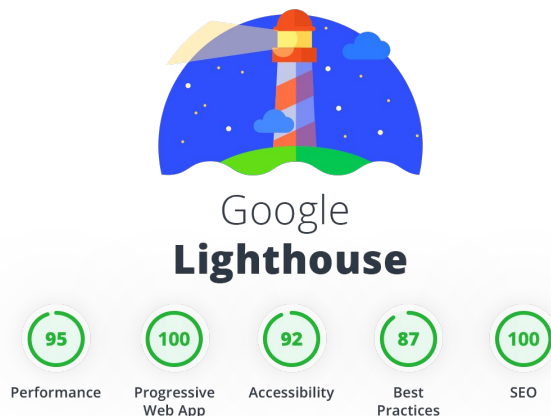
Context is everything

- Do we care about **search rankings**?
- Are the users of my app “**familiar**” or “**unfamiliar**” users?
- Is my app used heavily by **mobile web** users?
- What are the **critical actions** users take on my app?
- What **type of service** does my app deliver? How much **reassurance** do users need about the actions they are taking?






How to measure

- **Synthetic testing:** running Lighthouse manually, tests by synthetic monitoring
- **User testing:** collect data from **real** users as they use your application



Recommended: Data from real users

-  Statistical significance
-  Ability to collect **high-dimensional** data
-  A more accurate picture of what your users are experiencing



OpenTelemetry for the Browser

Open standard for collecting and sending **telemetry** data

What you will need to get started with OpenTelemetry for the browser:

- A browser site or app
- OpenTelemetry web APIs
- A backend that accepts OTel data
- Optional: OTel collector



Getting started

```
$ npm install --save \
  @opentelemetry/api \
  @opentelemetry/sdk-trace-web \
  @opentelemetry/exporter-trace-otlp-http \
  @opentelemetry/context-zone \
  @opentelemetry/instrumentation \
  @opentelemetry/auto-instrumentations-web
```

```
// tracing.js
import { OTLPTraceExporter } from '@opentelemetry/exporter-trace-otlp-http';
import { WebTracerProvider, BatchSpanProcessor } from '@opentelemetry/sdk-trace-web';
import { ZoneContextManager } from '@opentelemetry/context-zone';
import { Resource } from '@opentelemetry/resources';
import { SemanticResourceAttributes } from '@opentelemetry/semantic-conventions';
import { registerInstrumentations } from '@opentelemetry/instrumentation';
import { getWebAutoInstrumentations } from '@opentelemetry/auto-instrumentations-web';

// The exporter is responsible for sending traces from the browser to your collector
const exporter = new OTLPTraceExporter({
  url: 'https://<your collector endpoint>:443/v1/traces'
});

// The TracerProvider is the core library for creating traces
const provider = new WebTracerProvider({
  resource: new Resource({
    [SemanticResourceAttributes.SERVICE_NAME]: 'browser',
  }),
});

// The processor sorts through data as it comes in, before it is sent to the exporter
provider.addSpanProcessor(new BatchSpanProcessor(exporter));
```



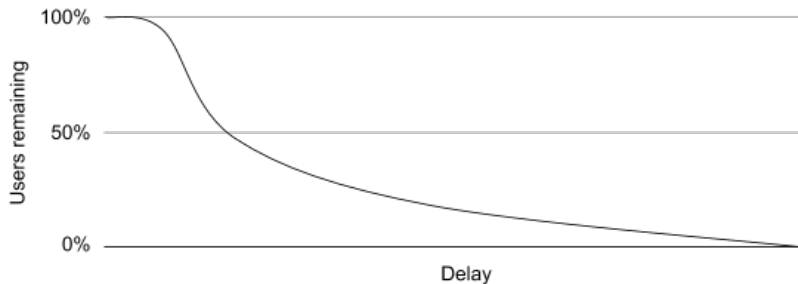
Add context

```
// The TracerProvider is the core library for creating traces
const provider = new WebTracerProvider({
  resource: new Resource({
    [SemanticResourceAttributes.SERVICE_NAME]: 'docs',
    'user_agent.original': window.navigator.userAgent,
    'browser.language': window.navigator.language,
    'browser.mobile': window.navigator.mobile,
    'browser.width': window.screen.width,
    'browser.height': window.screen.height,
    'browser.geolocation': window.navigator.geolocation,
  }),
});
```



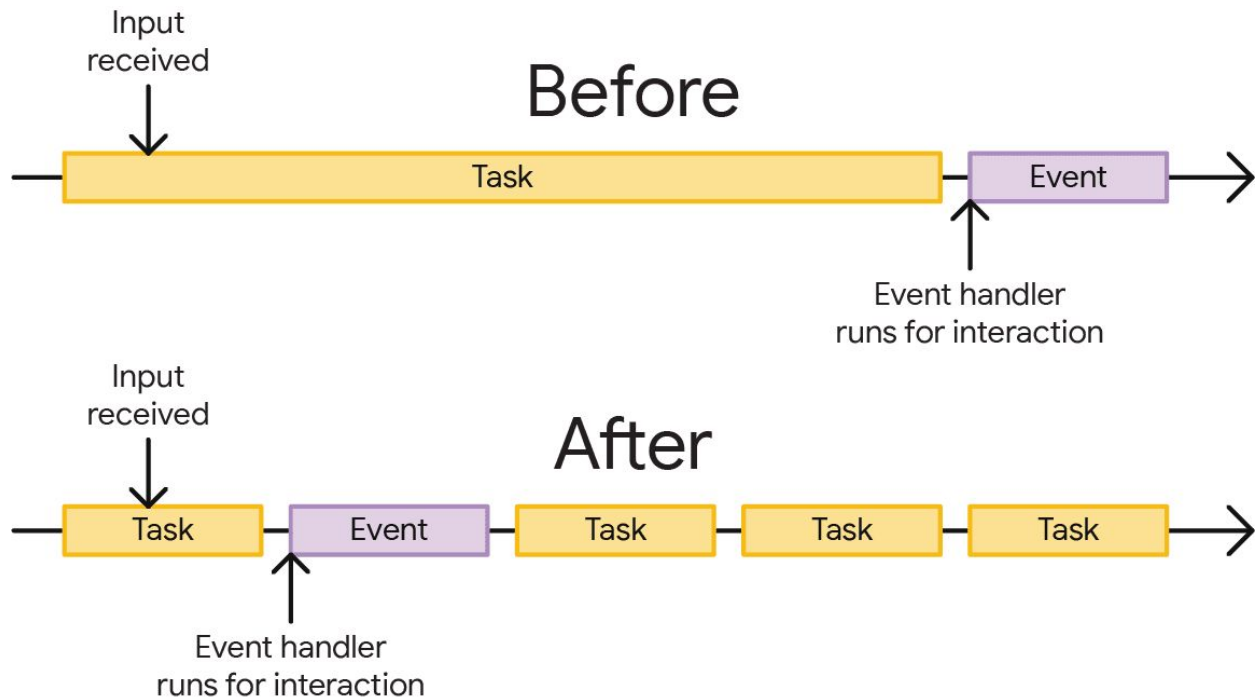
Beacon API

- OTel web can send telemetry through browser beacon API
- Allows telemetry to be sent even after user navigates away
- Can be used to collect data about **phantom bounce rate**
- Use the OTel collector without any headers set to send telemetry with Beacon API





Long task instrumentation



Web vitals instrumentation

(Loading)

LCP

Largest Contentful Paint



(Interactivity)

FID

First Input Delay



(Visual Stability)

CLS

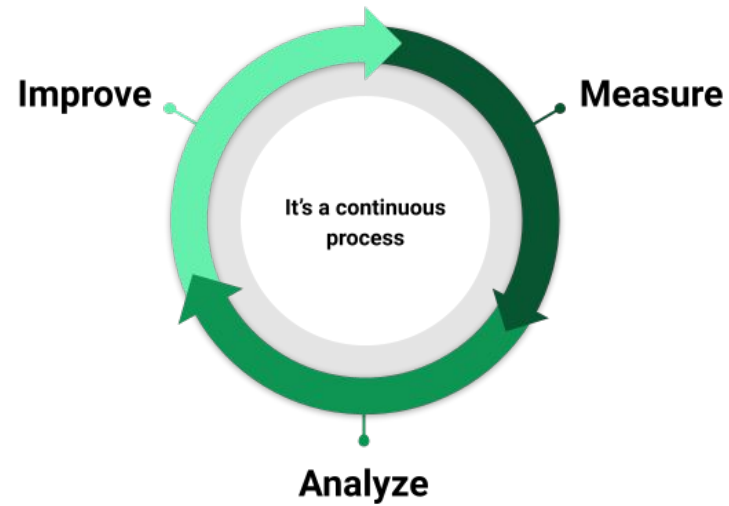
Cumulative Layout Shift



Manual Instrumentation

- Define what your **“main-content”** is to define your own **page load** event
- Identify functions that are running **long tasks** and add additional instrumentation
- Critical paths in your app
- Spinners
- **Add manual instrumentation to further investigate performance issues**





Recap

- **How** things are measured is relevant to the **insight** we get from the measurement
- Your context dictates what measurements are important and what baseline values should be set
- Measure with **real users**
- Add extra information to ensure **high-dimensionality** data
- **Manually instrument** things to investigate them further
- **Measuring performance is a continuous process**





Questions?
