Interaction to Next Paint (INP) becomes a Core Web Vital on March 12. Start making your websites more responsive to user input today. Learn how. (https://web.dev/blog/inp-cwv-march-12?utm_source=web.dev&utm_medium=banner&utm_campaign=inp-cwv)

Largest Contentful Paint (LCP)

Philip Walton
(https://twitter.com/philwalton)
(https://github.com/philipwalton)
(https://philipwalton.com/)

Barry Pollard
(https://twitter.com/tunetheweb)
(https://github.com/tunetheweb)
(https://webperf.social/@tunetheweb)
(https://www.tunetheweb.com)

Browser Support				
	77	79	122	x

Source (https://developer.mozilla.org/docs/Web/API/LargestContentfulPaint)

Largest Contentful Paint (LCP) is a stable (/articles/vitals#lifecycle) Core Web Vital metric for measuring perceived load speed (/articles/user-centric-performance-metrics#types_of_metrics%22). It marks the point in the page load timeline when the page's main content has likely loaded. A fast LCP helps reassure the user that the page is useful (/articles/user-centric-performance-metrics#defining_metrics).

Historically, it's been a challenge for web developers to measure how quickly the main content of a web page loads and is visible to users. Older metrics like load (https://developer.mozilla.org/docs/Web/Events/load) or DOMContentLoaded (https://developer.mozilla.org/docs/Web/Events/DOMContentLoaded) don't work well because they don't necessarily correspond to what the user sees on their screen. And newer, user-centric performance metrics like First Contentful Paint (FCP) (/articles/fcp) only capture the very beginning

(https://developer.chrome.com/docs/lighthouse/performance/speed-index/) (both available in Lighthouse) to help capture more of the loading experience after the initial paint, but these metrics are complex, hard to explain, and often wrong, meaning they still don't identify when the main content of the page has loaded.

Based on discussions in the W3C Web Performance Working Group (https://www.w3.org/webperf/) and research done at Google, we've found that a more accurate way to measure when the main content of a page is loaded is to look at when the largest element is rendered.

What is LCP?

LCP reports the render time of the largest image or text block (#what-elements-are-considered) visible in the viewport, relative to when the user first navigated to the page.

Key Point: LCP includes any unload time from the previous page, connection setup time, redirect time, and Time To First Byte (TTFB) (/articles/ttfb), which can all be significant when measured in the field and can lead to differences between field and lab measurements.

What is a good LCP score?

To provide a good user experience, sites should strive to have LCP of **2.5 seconds** or less. To ensure you're hitting this target for most of your users, a good threshold to measure is the **75th percentile** of page loads, segmented across mobile and desktop devices.

A good LCP value is 2.5 seconds or less.

Note: To learn more about the research and methodology behind this recommendation, see Defining the Core

types of elements considered for Largest Contentful Paint are:

- `<img>` elements (the first frame presentation time (https://chromium.googlesource.com/chromium/src/+/refs/heads/main/docs/speed/metrics_changelog/2023_08_lcp.md) is used for animated content such as GIFs or animated PNGs)
- `<image>` elements inside an `<svg>` element
- `<video>` elements (the poster image load time or first frame presentation time (https://chromium.googlesource.com/chromium/src/+/refs/heads/main/docs/speed/metrics_changelog/2023_08_lcp.md) for videos is used—whichever is earlier)
- An element with a background image loaded using the `url()` (https://developer.mozilla.org/docs/Web/CSS/url()) function, (as opposed to a CSS gradient (https://developer.mozilla.org/docs/Web/CSS/CSS_Images/Using_CSS_gradients))
- Block-level (https://developer.mozilla.org/docs/Web/HTML/Block-level_elements) elements containing text nodes or other inline-level text element children.

Restricting the elements to this limited set was intentional to reduce complexity. Additional elements (like the full `<svg>` support) may be added in the future as more research is conducted.

As well as only considering some elements, LCP measurements use heuristics to exclude certain elements that users are likely to see as "non-contentful". For Chromium-based browsers, these include:

- Elements with an opacity of 0, making them invisible to the user.
- Elements that cover the full viewport, that are likely to be background elements.
- Placeholder images or other images with a low entropy, that likely don't reflect the true content of the page.

Browsers are likely to continue to improve these heuristics to ensure we match user expectations of what the largest *contentful* element is.

These "contentful" heuristics differ from those used by FCP (/articles/fcp), which might consider some of these elements, such as placeholder images or full viewport images, even if they're ineligible to be LCP candidates. Despite both using "contentful" in their name, the aim of these metrics is different. FCP measures when *any content* is painted to screen, whereas LCP measures when the *main content* is painted.

non-visible overflow (https://developer.mozilla.org/docs/Web/CSS/overflow), those portions don't count toward the element's size.

For image elements that have been resized from their intrinsic size (https://developer.mozilla.org/docs/Glossary/Intrinsic_Size), the size that gets reported is either the visible size or the intrinsic size, whichever is smaller.

For text elements, LCP considers only the smallest rectangle that can contain all text nodes.

For all elements, LCP doesn't consider margins, paddings, or borders applied using CSS.

Note: Determining which text nodes belong to which elements can be tricky, especially for elements whose children include inline elements and plain text nodes but also block-level elements. For LCP, every text node belongs to only its closest block-level ancestor element. In terms of specifications (https://wicg.github.io/element-timing/#set-of-owned-text-nodes): each text node belongs to the element that generates its containing block (https://developer.mozilla.org/docs/Web/CSS/Containing_block)

When is LCP reported?

Web pages often load in stages, and as a result, the largest element on the page might change during loading.

To handle this potential for change, the browser dispatches a `PerformanceEntry` (https://developer.mozilla.org/docs/Web/API/PerformanceEntry) of type `largest-contentful-paint` identifying the largest contentful element as soon as the browser has painted the first frame. After rendering subsequent frames, it dispatches another `PerformanceEntry` (https://developer.mozilla.org/docs/Web/API/PerformanceEntry) any time the largest contentful element changes.

For example, on a page with text and a hero image, the browser might initially render only the text, and the browser would dispatch a `largest-contentful-paint` entry whose `element` property references a `<p>` or `<h1>`. After the hero image finishes loading, a second `largest-contentful-paint` entry is dispatched, with an `element` property referencing the `<img>`.

An element can only be considered the largest contentful element after it has rendered and is visible to the user. Images that haven't yet loaded aren't considered "rendered". Neither are text nodes using web fonts during the font block period (https://developer.mozilla.org/docs/Web/CSS/@font-face/font-display#The_font_display_timeline). In such cases, a smaller element might be reported as the largest contentful element, but as soon as the

If the largest contentful element is removed from the viewport, or even from the DOM, it remains the largest contentful element unless a larger element is rendered.

Note: Before Chrome 88, removed elements weren't considered largest contentful elements, and removing the current candidate would create a new **`largest-contentful-paint`** entry. However, because of popular UI patterns like image carousels often removing DOM elements, the metric was updated to more accurately reflect what users experience. See the changelog (https://chromium.googlesource.com/chromium/src/+/main/docs/speed/metrics_changelog/2020_11_lcp.md) for more details.

The browser stops reporting new entries as soon as the user interacts with the page (through a tap, scroll, or keypress), because user interaction often changes what's visible to the user (especially when scrolling).

For analysis purposes, report only the most recently dispatched `PerformanceEntry` to your analytics service.

Caution: Because users can open pages in a background tab, it's possible that **`largest-contentful-paint`** entries won't be dispatched until the user focuses the tab, which can be much later than when they first loaded it. Google tools that measure LCP don't report this metric if the page was loaded in the background, because it doesn't reflect the user-perceived load time.

Load time versus render time

For security reasons, the render timestamp of images is not exposed for cross-origin images that lack the `Timing-Allow-Origin` (https://developer.mozilla.org/docs/Web/HTTP/Headers/Timing-Allow-Origin) header. Instead, only their load time, which other APIs already expose, is available.

This can lead to the seemingly impossible situation where LCP is reported by web APIs as earlier than FCP. This is only because of this security restriction, and it doesn't represent what's really happening.

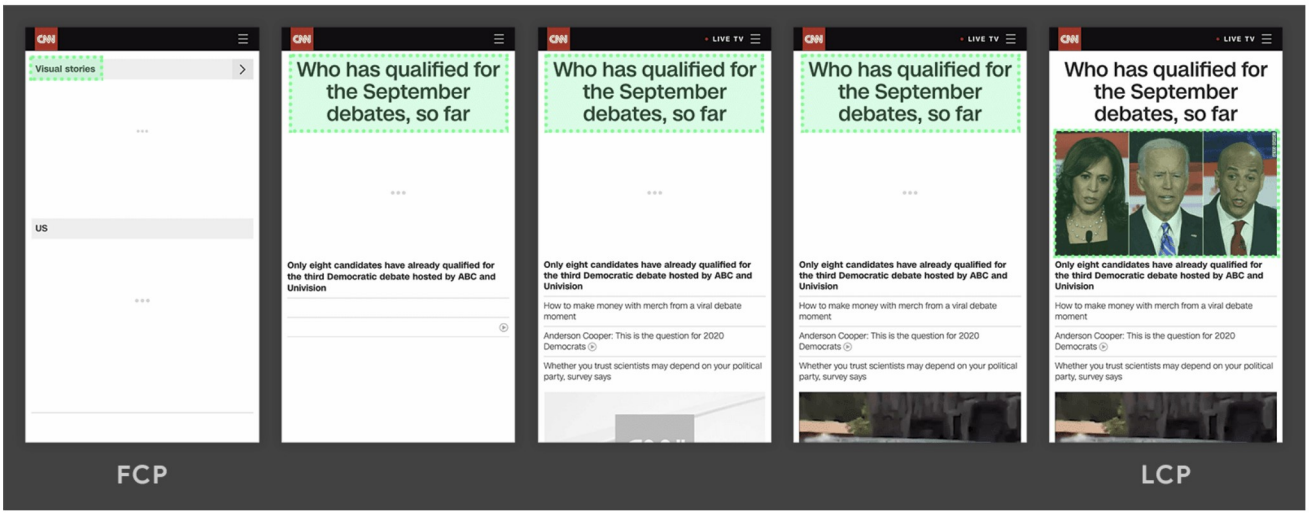
When possible, we always recommend setting the `Timing-Allow-Origin` (https://developer.mozilla.org/docs/Web/HTTP/Headers/Timing-Allow-Origin) header to make your metrics more accurate.

initial size and position in the viewport is considered.

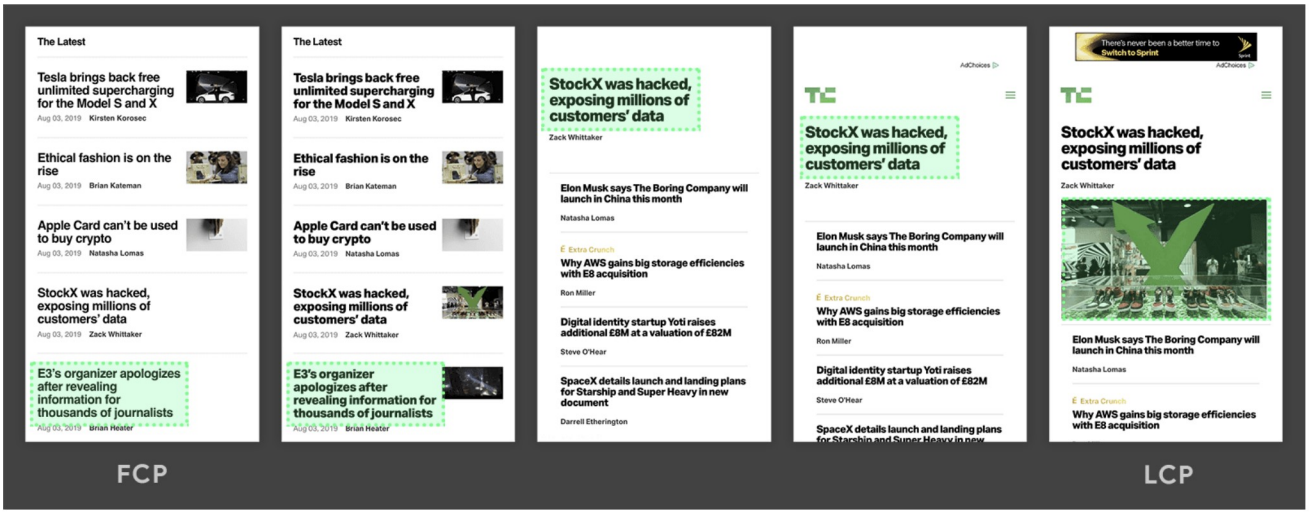
This means images that are initially rendered off-screen and then transition on-screen might not be reported. It also means elements initially rendered in the viewport that then get pushed out of view still report their initial in-viewport size.

Examples

Here are some examples of when the Largest Contentful Paint occurs on a few popular websites:



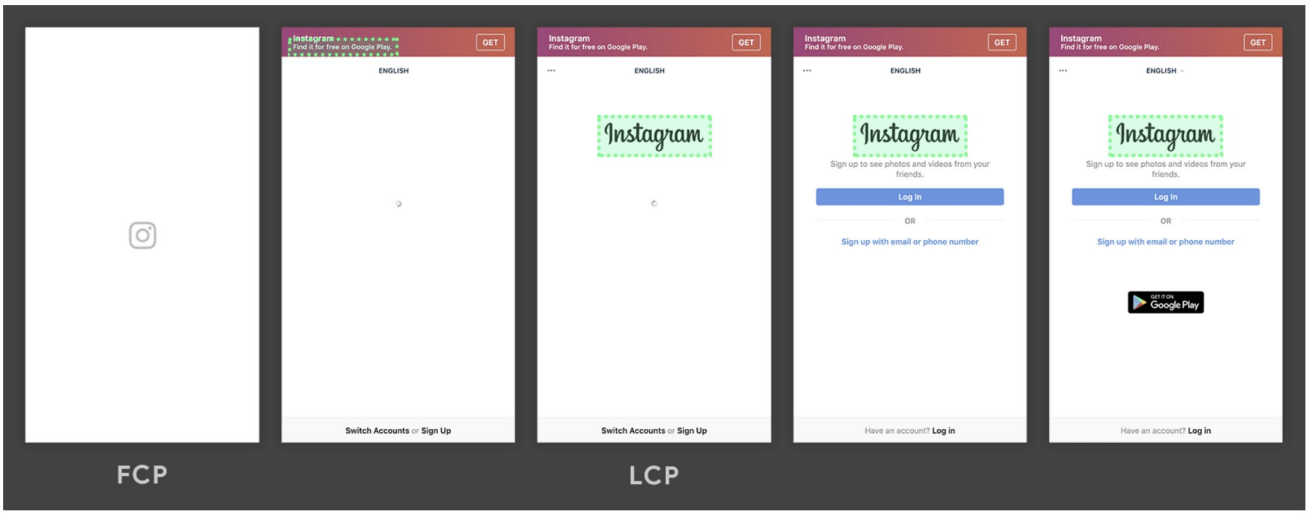
An LCP timeline from cnn.com.



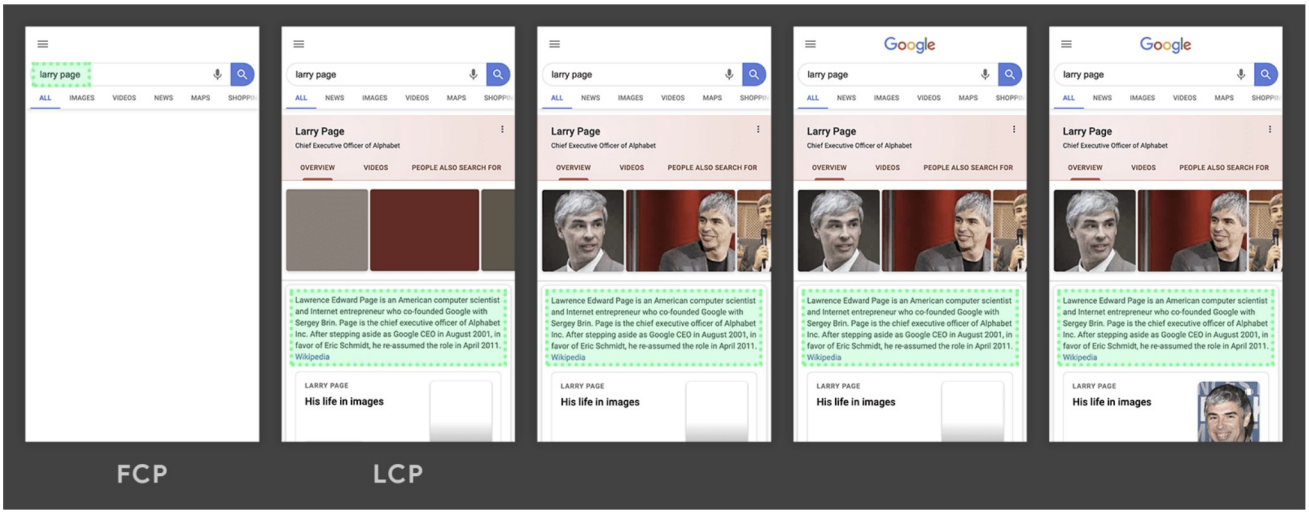
An LCP timeline from techcrunch.com.

In both of these timelines, the largest element (highlighted in green) changes as content loads. In

the case. The next two examples show the LCP happening before the page fully loads.



An LCP timeline from instagram.com.



An LCP timeline from google.com.

In the first example, the Instagram logo loads relatively early and remains the largest element even as other content is added. In the Google Search results page example, the largest element is a paragraph of text that displays before any of the images or the logo finish loading. Because each individual image is smaller than this paragraph, it remains the largest element throughout the load process.

Note: In the first frame of the Instagram timeline, the camera logo doesn't have a green box around it. That's because it's an **<svg>** element, which isn't considered as an LCP candidate. The first LCP candidate is the text in

LCP can be measured in the lab (/articles/user-centric-performance-metrics#in_the_lab) or in the field (/articles/user-centric-performance-metrics#in_the_field), and it's available in the following tools:

Field tools

- Chrome User Experience Report (https://developer.chrome.com/docs/crux/)
- PageSpeed Insights (https://pagespeed.web.dev/)
- Search Console (Core Web Vitals report) (https://support.google.com/webmasters/answer/9205520)
- `web-vitals` JavaScript library (https://github.com/GoogleChrome/web-vitals)

Lab tools

- Chrome DevTools (https://developer.chrome.com/docs/devtools/)
- Lighthouse (https://developer.chrome.com/docs/lighthouse/overview/)
- PageSpeed Insights (https://pagespeed.web.dev/)
- WebPageTest (https://webpagetest.org/)

Measure LCP in JavaScript

To measure LCP in JavaScript, use the Largest Contentful Paint API (https://wicg.github.io/largest-contentful-paint/). The following example shows how to create a `PerformanceObserver` (https://developer.mozilla.org/docs/Web/API/PerformanceObserver) that listens for `largest-contentful-paint` entries and logs them to the console.

```
new PerformanceObserver((entryList) => {
  for (const entry of entryList.getEntries()) {
    console.log('LCP candidate:', entry.startTime, entry);
  }
}).observe({type: 'largest-contentful-paint', buffered: true});
```

Warning: This code shows how to log `largest-contentful-paint` entries to the console, but measuring LCP in JavaScript is more complicated. For details, see Differences between the Metric and the API

However, not all `largest-contentful-paint`entries are valid for measuring LCP.

The following section lists the differences between what the API reports and how the metric is calculated.

Differences between the metric and the API

- The API dispatches `largest-contentful-paint` entries for pages loaded in a background tab, but those pages should be ignored when calculating LCP.
- The API continues to dispatch `largest-contentful-paint` entries after a page has been backgrounded, but those entries should be ignored when calculating LCP. Elements can only be considered if the page was in the foreground the entire time.
- The API doesn't report `largest-contentful-paint` entries when the page is restored from the back/forward cache (/articles/bfcache#impact_on_core_web_vitals), but LCP should be measured in these cases because users experience them as distinct page visits.
- The API doesn't consider elements within iframes, but the metric does because they're part of the user experience of the page. In pages with an LCP within an iframe—for example a poster image on an embedded video—this will show as a difference between CrUX and RUM (/articles/crux-and-rum-differences#iframes). To measure LCP properly, you must include iframes. Sub-frames can use the API to report their `largest-contentful-paint` entries to the parent frame for aggregation.
- The API measures LCP from navigation start. For prerendered pages (https://developer.chrome.com/docs/web-platform/prerender-pages), measure LCP from `activationStart` (https://developer.mozilla.org/docs/Web/API/PerformanceNavigationTiming/activationStart) instead, because that corresponds to the LCP time as experienced by the user.

Instead of memorizing all these subtle differences, we recommend that developers use the `web-vitals` JavaScript library (https://github.com/GoogleChrome/web-vitals) to measure LCP, which handles most of these differences for you. (It doesn't cover the iframe issue.)

```
import {onLCP} from 'web-vitals';

// Measure and log LCP as soon as it's available.
onLCP(console.log);
```

limitations (https://github.com/GoogleChrome/web-vitals#limitations) section of the **`web-vitals`** library for details.

What if the largest element isn't the most important?

In some cases the most important element (or elements) on the page isn't the same as the largest element, and developers might be more interested in measuring the render times of these other elements instead. This is possible using the Element Timing API (https://wicg.github.io/element-timing/), as described in the article on custom metrics (/articles/custom-metrics#element_timing_api).

How to improve LCP

A full guide on optimizing LCP (/articles/optimize-lcp) is available to guide you through the process of identifying LCP timings in the field and using lab data to drill down and optimize them.

Additional resources

- Lessons learned from performance monitoring in Chrome (https://youtu.be/ctavZT87syI) by Annie Sullivan (https://anniesullie.com/) at performance.now() (https://perfnow.nl/) (2019)

Changelog

Occasionally, bugs are discovered in the APIs used to measure metrics, and sometimes in the definitions of the metrics themselves. As a result, changes must sometimes be made, and these changes can show up as improvements or regressions in your internal reports and dashboards.

To help you manage this, all changes to either the implementation or definition of these metrics is surfaced in this changelog (http://bit.ly/chrome-speed-metrics-changelog).

If you have feedback for these metrics, provide it in the web-vitals-feedback Google group (https://groups.google.com/g/web-vitals-feedback).