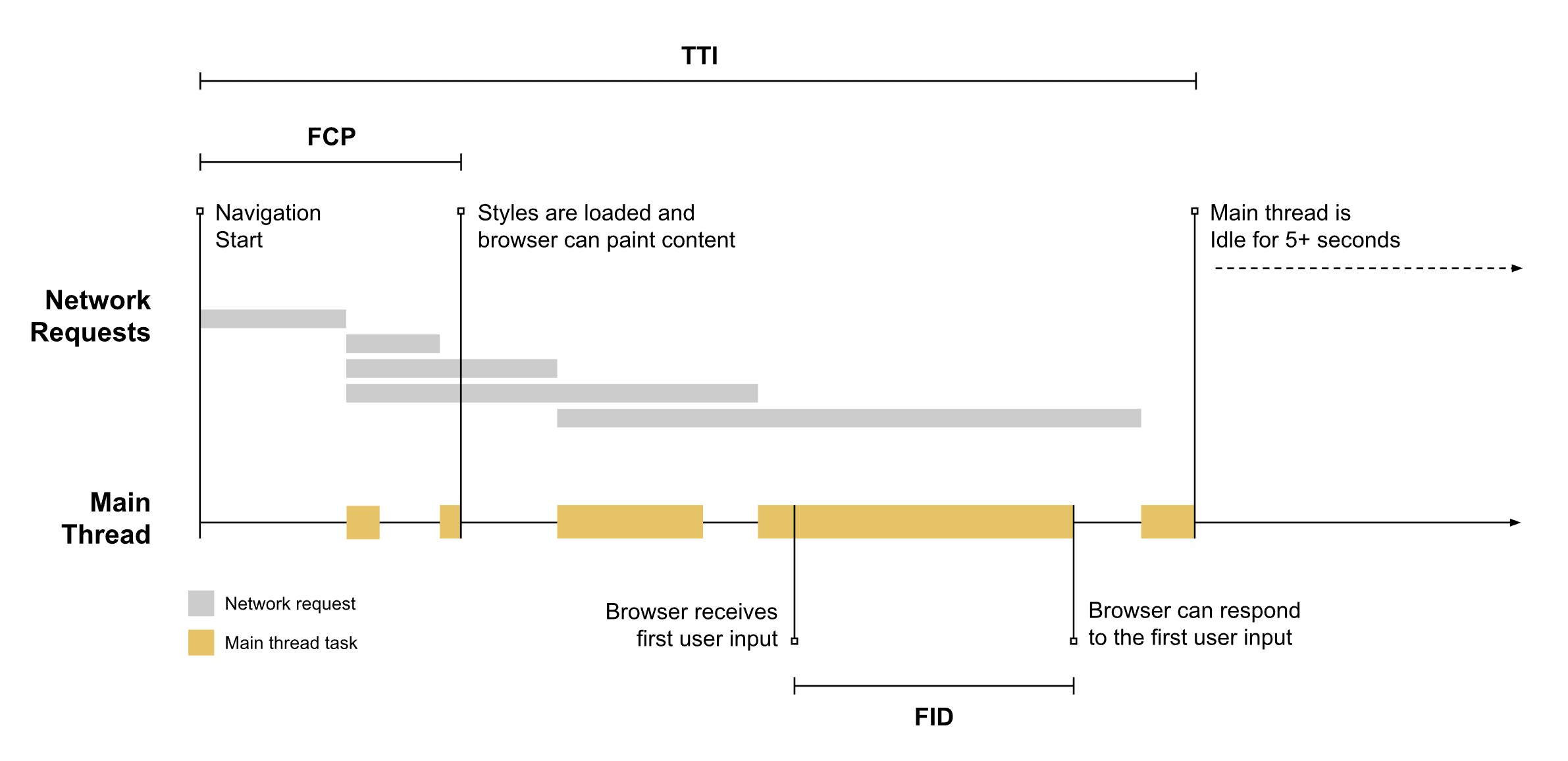
FID:

FID measures the time from when a user first interacts with a page (that is, when they click a link, tap on a button, or use a custom, JavaScript-powered control) to the time when the browser is actually able to begin processing event handlers in response to that interaction.

In general, input delay (a.k.a. input latency) happens because the browser's main thread is busy doing something else, so it can't (yet) respond to the user. One common reason this might happen is the browser is busy parsing and executing a large JavaScript file loaded by your app. While it's doing that, it can't run any event listeners because the JavaScript it's loading might tell it to do something else.



Because the input occurs while the browser is in the middle of running a task, it has to wait until the task completes before it can respond to the input. The time it must wait is the FID value for this user on this page.next

### What if an interaction doesn't have an event listener? [#](https://web.dev/fid/" \l "what-if-an-interaction-doesnt-have-an-event-listener)

FID measures the delta between when an input event is received and when the main thread is next idle. This means FID is measured **even in cases where an event listener has not been registered.** The reason is because many user interactions do not require an event listener but do require the main thread to be idle in order to run.

For example, all of the following HTML elements need to wait for in-progress tasks on the main thread to complete prior to responding to user interactions:

* Text fields, checkboxes, and radio buttons (<input>, <textarea>)
* Select dropdowns (<select>)
* links (<a>)

LCP:

Original link: https://web.dev/lcp/

The Largest Contentful Paint (LCP) metric reports the render time of the largest [image or text block](https://web.dev/lcp/" \l "what-elements-are-considered) visible within the viewport, relative to when the page [first started loading](https://w3c.github.io/hr-time/" \l "timeorigin-attribute).

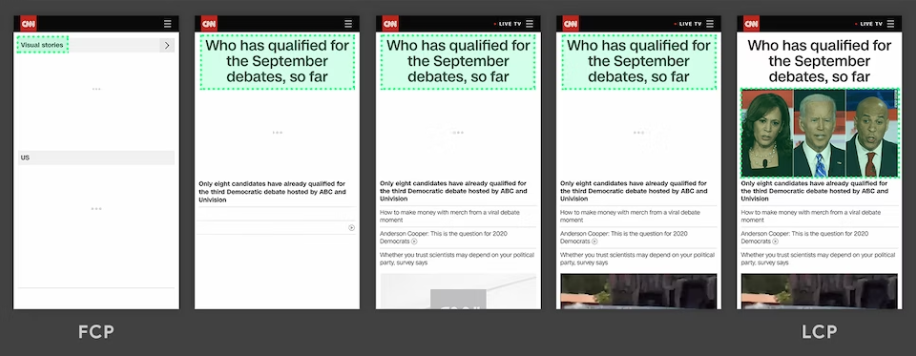
Largest Contentful Paint (LCP) is an important, user-centric metric for measuring [perceived load speed](https://web.dev/user-centric-performance-metrics/" \l "types-of-metrics) because it marks the point in the page load timeline when the page's main content has likely loaded—a fast LCP helps reassure the user that the page is [useful](https://web.dev/user-centric-performance-metrics/" \l "questions).

Older metrics like [load](https://developer.mozilla.org/docs/Web/Events/load) or [DOMContentLoaded](https://developer.mozilla.org/docs/Web/Events/DOMContentLoaded) are not good 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)](https://web.dev/fcp/) only capture the very beginning of the loading experience. If a page shows a splash screen or displays a loading indicator, this moment is not very relevant to the user.

### What elements are considered? [#](https://web.dev/lcp/" \l "what-elements-are-considered)

As currently specified in the [Largest Contentful Paint API](https://wicg.github.io/largest-contentful-paint/), the types of elements considered for Largest Contentful Paint are:

* <img> elements
* <image> elements inside an <svg> element
* <video> elements (the poster image is used)
* An element with a background image loaded via 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 elements children.



## How to improve LCP [#](https://web.dev/lcp/" \l "how-to-improve-lcp)

LCP is primarily affected by four factors:

* Slow server response times
* Render-blocking JavaScript and CSS
* Resource load times
* Client-side rendering

For a deep dive on how to improve LCP, see [Optimize LCP](https://web.dev/optimize-lcp/). For additional guidance on individual performance techniques that can also improve LCP, see:

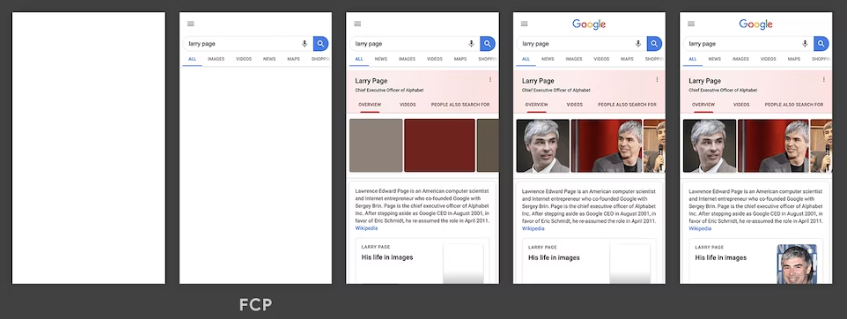
* [Apply instant loading with the PRPL pattern](https://web.dev/apply-instant-loading-with-prpl)
* [Optimizing the Critical Rendering Path](https://web.dev/critical-rendering-path/)
* [Optimize your CSS](https://web.dev/fast" \l "optimize-your-css)
* [Optimize your Images](https://web.dev/fast" \l "optimize-your-images)
* [Optimize web Fonts](https://web.dev/fast" \l "optimize-web-fonts)
* [Optimize your JavaScript](https://web.dev/fast" \l "optimize-your-javascript) (for client-rendered sites)

FCP:

First Contentful Paint (FCP) is an important, user-centric metric for measuring [perceived load speed](https://web.dev/user-centric-performance-metrics/" \l "types-of-metrics) because it marks the first point in the page load timeline where the user can see anything on the screen—a fast FCP helps reassure the user that something is [happening](https://web.dev/user-centric-performance-metrics/" \l "questions).

## What is FCP? [#](https://web.dev/fcp/" \l "what-is-fcp)

The First Contentful Paint (FCP) metric measures the time from when the page starts loading to when any part of the page's content is rendered on the screen. For this metric, "content" refers to text, images (including background images), <svg> elements, or non-white <canvas> elements.



In the above load timeline, FCP happens in the second frame, as that's when the first text and image elements are rendered to the screen.

You'll notice that though some of the content has rendered, not all of it has rendered. This is an important distinction to make between First Contentful Paint (FCP) and [Largest Contentful Paint (LCP)](https://web.dev/lcp/) —which aims to measure when the page's main contents have finished loading.

## How to improve FCP [#](https://web.dev/fcp/" \l "how-to-improve-fcp)

To learn how to improve FCP for a specific site, you can run a Lighthouse performance audit and pay attention to any specific [opportunities](https://developer.chrome.com/docs/lighthouse/performance/" \l "opportunities) or [diagnostics](https://developer.chrome.com/docs/lighthouse/performance/" \l "diagnostics) the audit suggests.

To learn how to improve FCP in general (for any site), refer to the following performance guides:

* [Eliminate render-blocking resources](https://developer.chrome.com/docs/lighthouse/performance/render-blocking-resources/)
* [Minify CSS](https://developer.chrome.com/docs/lighthouse/performance/unminified-css/)
* [Remove unused CSS](https://developer.chrome.com/docs/lighthouse/performance/unused-css-rules/)
* [Remove unused JavaScript](https://developer.chrome.com/docs/lighthouse/performance/unused-javascript/)
* [Preconnect to required origins](https://developer.chrome.com/docs/lighthouse/performance/uses-rel-preconnect/)
* [Reduce server response times (TTFB)](https://web.dev/ttfb/)
* [Avoid multiple page redirects](https://developer.chrome.com/docs/lighthouse/performance/redirects/)
* [Preload key requests](https://developer.chrome.com/docs/lighthouse/performance/uses-rel-preload/)
* [Avoid enormous network payloads](https://developer.chrome.com/docs/lighthouse/performance/total-byte-weight/)
* [Serve static assets with an efficient cache policy](https://developer.chrome.com/docs/lighthouse/performance/uses-long-cache-ttl/)
* [Avoid an excessive DOM size](https://developer.chrome.com/docs/lighthouse/performance/dom-size/)
* [Minimize critical request depth](https://developer.chrome.com/docs/lighthouse/performance/critical-request-chains/)
* [Ensure text remains visible during webfont load](https://developer.chrome.com/docs/lighthouse/performance/font-display/)
* [Keep request counts low and transfer sizes small](https://developer.chrome.com/docs/lighthouse/performance/resource-summary/)

Load:

Historically, web performance has been measured with the [load](https://developer.mozilla.org/docs/Web/API/Window/load_event) event. However, even though load is a well-defined moment in a page's lifecycle, that moment doesn't necessarily correspond with anything the user cares about.

For example, a server could respond with a minimal page that "loads" immediately but then defers fetching content and displaying anything on the page until several seconds after the load event fires. While such a page might technically have a fast load time, that time would not correspond to how a user actually experiences the page loading.

Over the past few years, members of the Chrome team—in collaboration with the [W3C Web Performance Working Group](https://www.w3.org/webperf/)—have been working to standardize a set of new APIs and metrics that more accurately measure how users experience the performance of a web page.