**Cumulative Layout Shift (CLS)**

Jun 11, 2019 • Updated Jun 1, 2021

Available in: [Español](https://web.dev/i18n/es/cls/), [日本語](https://web.dev/i18n/ja/cls/), [한국어](https://web.dev/i18n/ko/cls/), [Português](https://web.dev/i18n/pt/cls/), [Русский](https://web.dev/i18n/ru/cls/), [中文](https://web.dev/i18n/zh/cls/), [English](https://web.dev/i18n/en/cls/)

Appears in: [Web Vitals](https://web.dev/learn-web-vitals)|[Metrics](https://web.dev/metrics)

[](https://web.dev/authors/philipwalton/)

[*Philip Walton*](https://web.dev/authors/philipwalton/)

* [Twitter](https://twitter.com/philwalton)
* [GitHub](https://github.com/philipwalton)
* [Blog](https://philipwalton.com)

[](https://web.dev/authors/mihajlija/)

[*Milica Mihajlija*](https://web.dev/authors/mihajlija/)

* [Twitter](https://twitter.com/bibydigital)
* [GitHub](https://github.com/mihajlija)
* [Blog](https://mihajlija.github.io/)

**Jun 1, 2021:** The implementation of CLS has changed. To learn more about the reasons behind the change, check out [Evolving the CLS metric](https://web.dev/evolving-cls).

**Key Term**: Cumulative Layout Shift (CLS) is an important, user-centric metric for measuring [visual stability](https://web.dev/user-centric-performance-metrics/#types-of-metrics) because it helps quantify how often users experience unexpected layout shifts—a low CLS helps ensure that the page is [delightful](https://web.dev/user-centric-performance-metrics/#questions).

Have you ever been reading an article online when something suddenly changes on the page? Without warning, the text moves, and you've lost your place. Or even worse: you're about to tap a link or a button, but in the instant before your finger lands—BOOM—the link moves, and you end up clicking something else!

Most of the time these kinds of experiences are just annoying, but in some cases, they can cause real damage.

A screencast illustrating how layout instability can negatively affect users.

Unexpected movement of page content usually happens because resources are loaded asynchronously or DOM elements get dynamically added to the page above existing content. The culprit might be an image or video with unknown dimensions, a font that renders larger or smaller than its fallback, or a third-party ad or widget that dynamically resizes itself.

What makes this issue even more problematic is that how a site functions in development is often quite different from how users experience it. Personalized or third-party content often doesn't behave the same in development as it does in production, test images are often already in the developer's browser cache, and API calls that run locally are often so fast that the delay isn't noticeable.

The Cumulative Layout Shift (CLS) metric helps you address this problem by measuring how often it's occurring for real users.

**What is CLS?** [**#**](https://web.dev/cls/#what-is-cls)

CLS is a measure of the largest burst of *layout shift scores* for every [unexpected](https://web.dev/cls/#expected-vs.-unexpected-layout-shifts) layout shift that occurs during the entire lifespan of a page.

A *layout shift* occurs any time a visible element changes its position from one rendered frame to the next. (See below for details on how individual [layout shift scores](https://web.dev/cls/#layout-shift-score) are calculated.)

A burst of layout shifts, known as a [*session window*](https://web.dev/evolving-cls/#why-a-session-window), is when one or more individual layout shifts occur in rapid succession with less than 1-second in between each shift and a maximum of 5 seconds for the total window duration.

The largest burst is the session window with the maximum cumulative score of all layout shifts within that window.

Example of session windows. Blue bars represent the scores of each individual layout shift.

**Caution**: Previously CLS measured the sum total of *all individual layout shift scores* that occurred during the entire lifespan of the page. To see which tools still provide the ability to benchmark against the original implementation, check out [Evolving Cumulative Layout Shift in web tooling](https://web.dev/cls-web-tooling).

**What is a good CLS score?** [**#**](https://web.dev/cls/#what-is-a-good-cls-score)

To provide a good user experience, sites should strive to have a CLS score of **0.1** 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.

To learn more about the research and methodology behind this recommendation, see: [Defining the Core Web Vitals metrics thresholds](https://web.dev/defining-core-web-vitals-thresholds/)

**Layout shifts in detail** [**#**](https://web.dev/cls/#layout-shifts-in-detail)

Layout shifts are defined by the [Layout Instability API](https://github.com/WICG/layout-instability), which reports layout-shift entries any time an element that is visible within the viewport changes its start position (for example, its top and left position in the default [writing mode](https://developer.mozilla.org/docs/Web/CSS/writing-mode)) between two frames. Such elements are considered *unstable elements*.

Note that layout shifts only occur when existing elements change their start position. If a new element is added to the DOM or an existing element changes size, it doesn't count as a layout shift—as long as the change doesn't cause other visible elements to change their start position.

**Layout shift score** [**#**](https://web.dev/cls/#layout-shift-score)

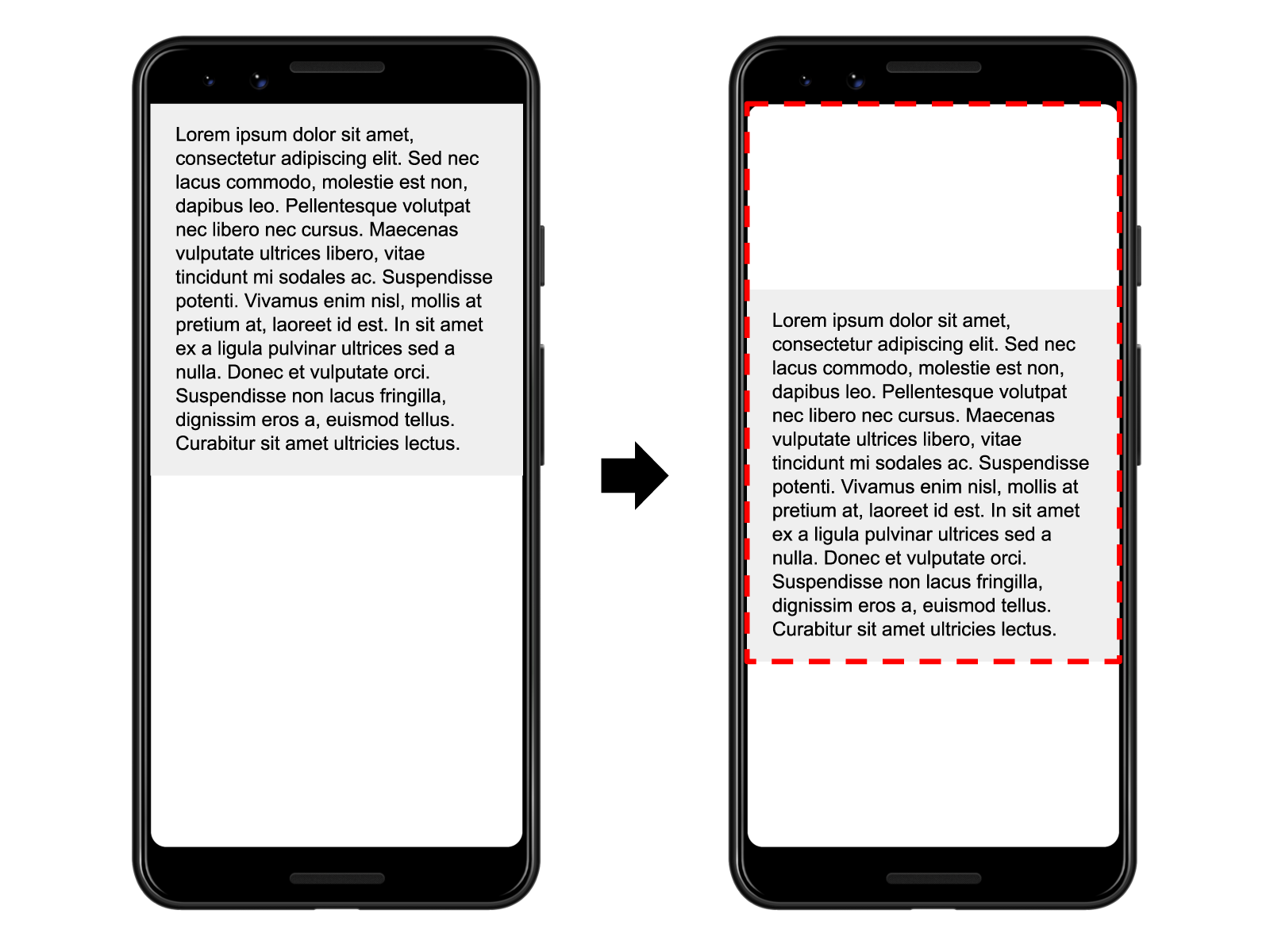
To calculate the *layout shift score*, the browser looks at the viewport size and the movement of *unstable elements* in the viewport between two rendered frames. The layout shift score is a product of two measures of that movement: the *impact fraction* and the *distance fraction* (both defined below).

layout shift score = impact fraction \* distance fraction

**Impact fraction** [**#**](https://web.dev/cls/#impact-fraction)

The [impact fraction](https://github.com/WICG/layout-instability#Impact-Fraction) measures how *unstable elements* impact the viewport area between two frames.

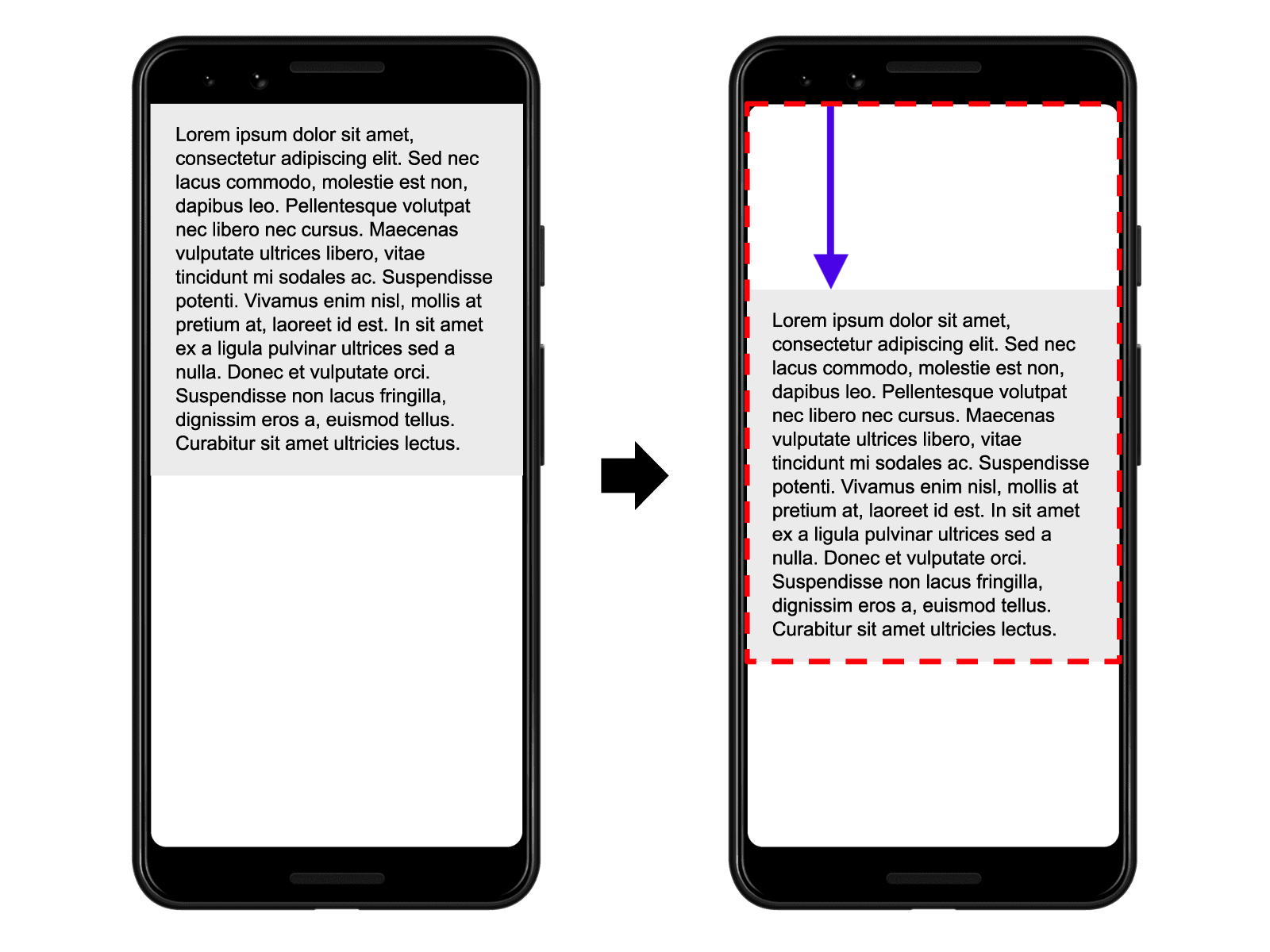
The union of the visible areas of all *unstable elements* for the previous frame *and* the current frame—as a fraction of the total area of the viewport—is the *impact fraction* for the current frame.

[](https://web-dev.imgix.net/image/tcFciHGuF3MxnTr1y5ue01OGLBn2/BbpE9rFQbF8aU6iXN1U6.png?auto=format)

In the image above there's an element that takes up half of the viewport in one frame. Then, in the next frame, the element shifts down by 25% of the viewport height. The red, dotted rectangle indicates the union of the element's visible area in both frames, which, in this case, is 75% of the total viewport, so its *impact fraction* is 0.75.

**Distance fraction** [**#**](https://web.dev/cls/#distance-fraction)

The other part of the layout shift score equation measures the distance that unstable elements have moved, relative to the viewport. The *distance fraction* is the greatest distance any *unstable element* has moved in the frame (either horizontally or vertically) divided by the viewport's largest dimension (width or height, whichever is greater).

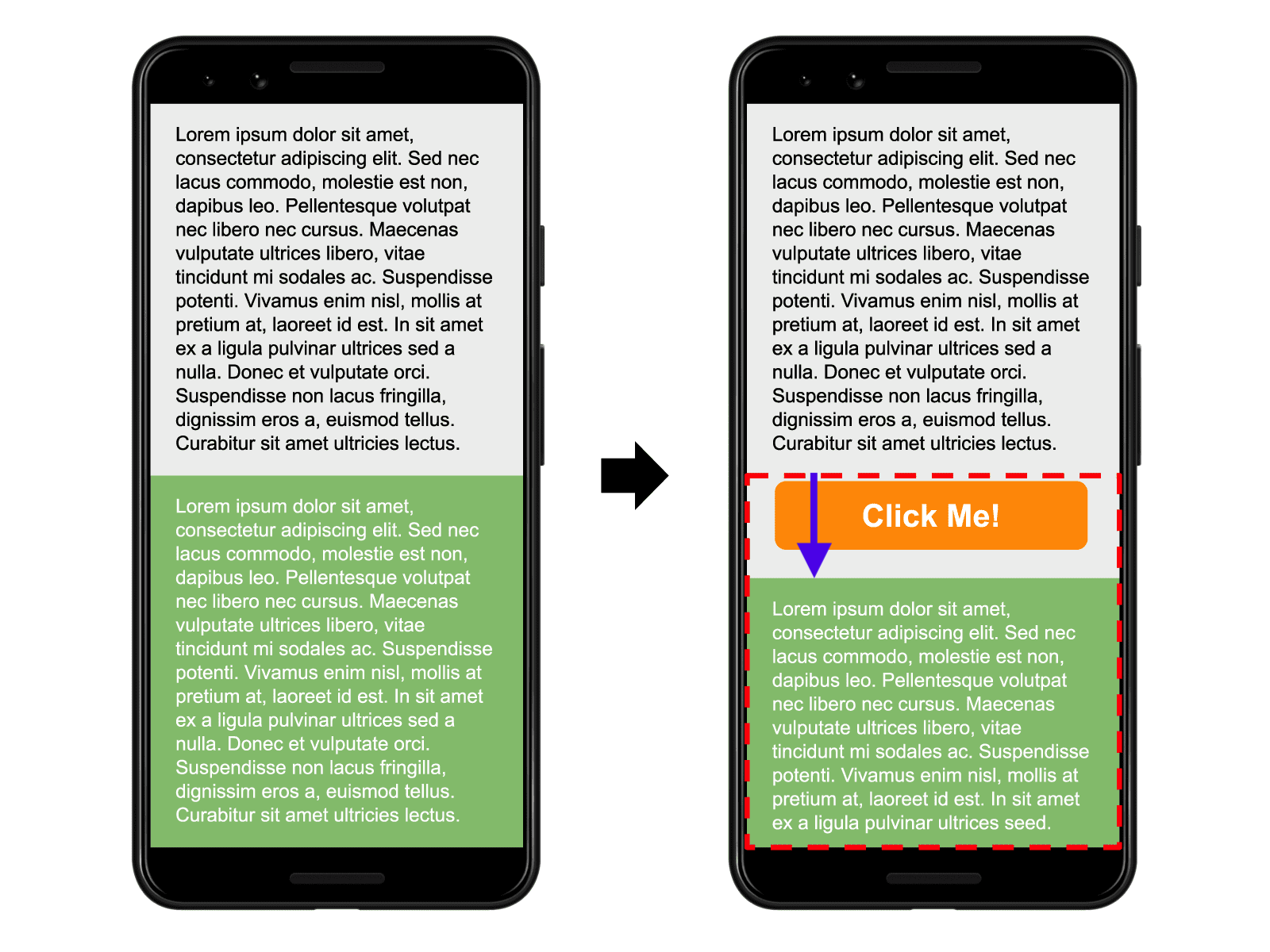
[](https://web-dev.imgix.net/image/tcFciHGuF3MxnTr1y5ue01OGLBn2/ASnfpVs2n9winu6mmzdk.png?auto=format)

In the example above, the largest viewport dimension is the height, and the unstable element has moved by 25% of the viewport height, which makes the *distance fraction* 0.25.

So, in this example the *impact fraction* is 0.75 and the *distance fraction* is 0.25, so the *layout shift score* is 0.75 \* 0.25 = 0.1875.

Initially, the layout shift score was calculated based only on *impact fraction*. The *distance fraction* was introduced to avoid overly penalizing cases where large elements shift by a small amount.

The next example illustrates how adding content to an existing element affects the layout shift score:

[](https://web-dev.imgix.net/image/tcFciHGuF3MxnTr1y5ue01OGLBn2/xhN81DazXCs8ZawoCj0T.png?auto=format)

The "Click Me!" button is appended to the bottom of the gray box with black text, which pushes the green box with white text down (and partially out of the viewport).

In this example, the gray box changes size, but its start position does not change so it's not an *unstable element*.

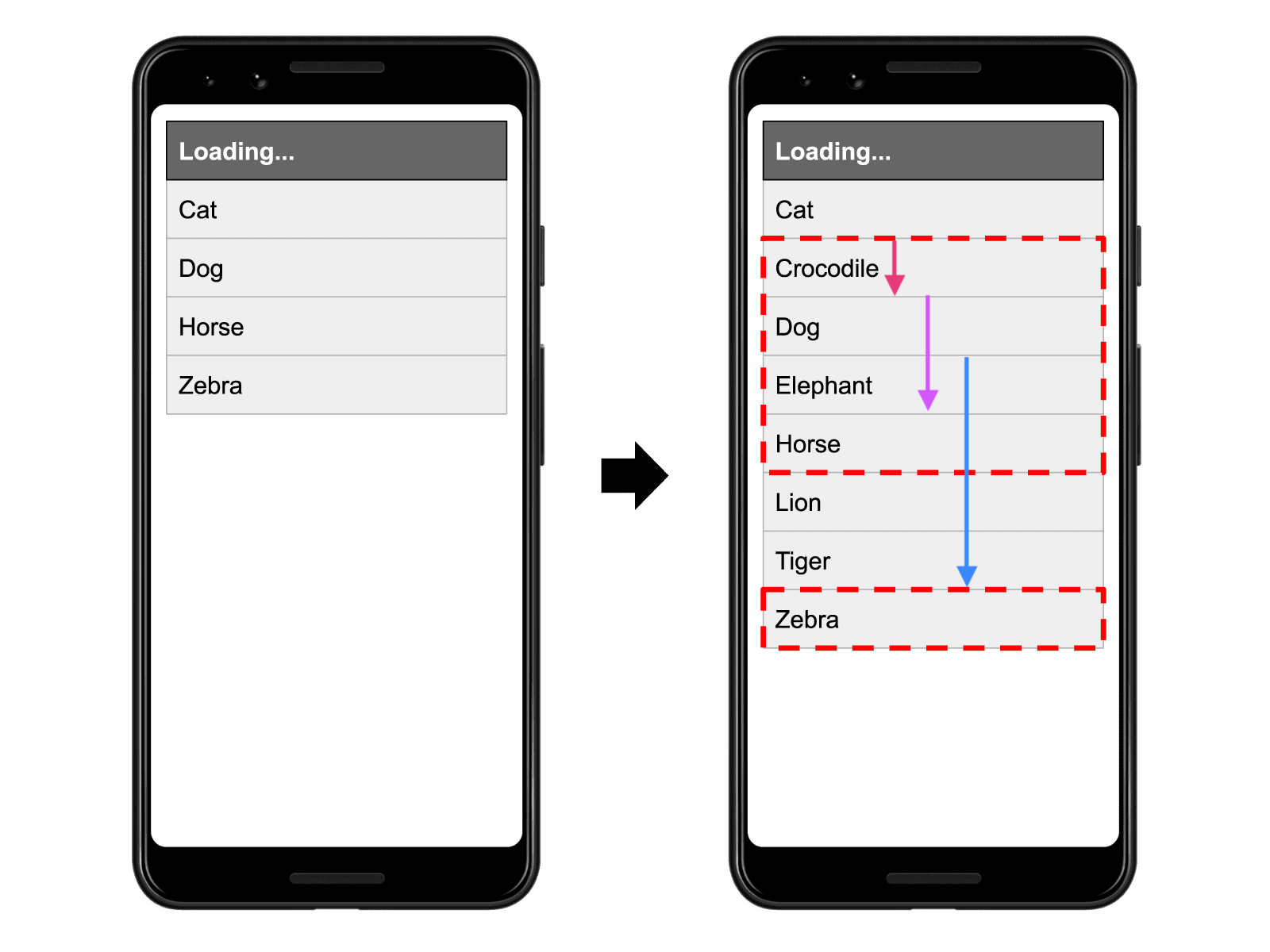
The "Click Me!" button was not previously in the DOM, so its start position doesn't change either.

The start position of the green box, however, does change, but since it's been moved partially out of the viewport, the invisible area is not considered when calculating the *impact fraction*. The union of the visible areas for the green box in both frames (illustrated by the red, dotted rectangle) is the same as the area of the green box in the first frame—50% of the viewport. The *impact fraction* is 0.5.

The *distance fraction* is illustrated with the purple arrow. The green box has moved down by about 14% of the viewport so the *distance fraction* is 0.14.

The layout shift score is 0.5 x 0.14 = 0.07.

This last example illustrates multiple *unstable elements*:

[](https://web-dev.imgix.net/image/tcFciHGuF3MxnTr1y5ue01OGLBn2/FdCETo2dLwGmzw0V5lNT.png?auto=format)

In the first frame above there are four results of an API request for animals, sorted in alphabetical order. In the second frame, more results are added to the sorted list.

The first item in the list ("Cat") does not change its start position between frames, so it's stable. Similarly, the new items added to the list were not previously in the DOM, so their start positions don't change either. But the items labelled "Dog", "Horse", and "Zebra" all shift their start positions, making them *unstable elements*.

Again, the red, dotted rectangles represent the union of these three *unstable elements*' before and after areas, which in this case is around 38% of the viewport's area (*impact fraction* of 0.38).

The arrows represent the distances that *unstable elements* have moved from their starting positions. The "Zebra" element, represented by the blue arrow, has moved the most, by about 30% of the viewport height. That makes the *distance fraction* in this example 0.3.

The layout shift score is 0.38 x 0.3 = 0.1172.

**Expected vs. unexpected layout shifts** [**#**](https://web.dev/cls/#expected-vs.-unexpected-layout-shifts)

Not all layout shifts are bad. In fact, many dynamic web applications frequently change the start position of elements on the page.

**User-initiated layout shifts** [**#**](https://web.dev/cls/#user-initiated-layout-shifts)

A layout shift is only bad if the user isn't expecting it. On the other hand, layout shifts that occur in response to user interactions (clicking a link, pressing a button, typing in a search box and similar) are generally fine, as long as the shift occurs close enough to the interaction that the relationship is clear to the user.

For example, if a user interaction triggers a network request that may take a while to complete, it's best to create some space right away and show a loading indicator to avoid an unpleasant layout shift when the request completes. If the user doesn't realize something is loading, or doesn't have a sense of when the resource will be ready, they may try to click something else while waiting—something that could move out from under them.

Layout shifts that occur within 500 milliseconds of user input will have the [hadRecentInput](https://wicg.github.io/layout-instability/#dom-layoutshift-hadrecentinput) flag set, so they can be excluded from calculations.

**Caution**: The hadRecentInput flag will only be true for discrete input events like tap, click, or keypress. Continuous interactions such as scrolls, drags, or pinch and zoom gestures are not considered "recent input". See the [Layout Instability Spec](https://github.com/WICG/layout-instability#recent-input-exclusion) for more details.

**Animations and transitions** [**#**](https://web.dev/cls/#animations-and-transitions)

Animations and transitions, when done well, are a great way to update content on the page without surprising the user. Content that shifts abruptly and unexpectedly on the page almost always creates a bad user experience. But content that moves gradually and naturally from one position to the next can often help the user better understand what's going on, and guide them between state changes.

Be sure to respect [prefers-reduced-motion](https://web.dev/prefers-reduced-motion/) browser settings, as some site visitors can experience ill effects or attention issues from animation.

CSS [transform](https://developer.mozilla.org/docs/Web/CSS/transform) property allows you to animate elements without triggering layout shifts:

* Instead of changing the height and width properties, use transform: scale().
* To move elements around, avoid changing the top, right, bottom, or left properties and use transform: translate() instead.

**How to measure CLS** [**#**](https://web.dev/cls/#how-to-measure-cls)

CLS can be measured [in the lab](https://web.dev/user-centric-performance-metrics/#in-the-lab) or [in the field](https://web.dev/user-centric-performance-metrics/#in-the-field), and it's available in the following tools:

**Caution**: Lab tools typically load pages in a synthetic environment and are thus only able to measure layout shifts that occur during page load. As a result, CLS values reported by lab tools for a given page may be less than what real users experience in the field.

**Field tools** [**#**](https://web.dev/cls/#field-tools)

* [Chrome User Experience Report](https://developers.google.com/web/tools/chrome-user-experience-report)
* [PageSpeed Insights](https://developers.google.com/speed/pagespeed/insights/)
* [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** [**#**](https://web.dev/cls/#lab-tools)

* [Chrome DevTools](https://developers.google.com/web/tools/chrome-devtools/)
* [Lighthouse](https://developers.google.com/web/tools/lighthouse/)
* [WebPageTest](https://webpagetest.org/)

**Measure CLS in JavaScript** [**#**](https://web.dev/cls/#measure-cls-in-javascript)

To measure CLS in JavaScript, you can use the [Layout Instability API](https://github.com/WICG/layout-instability). The following example shows how to create a [PerformanceObserver](https://developer.mozilla.org/docs/Web/API/PerformanceObserver) that listens for unexpected layout-shift entries, groups them into sessions, and logs the maximum session value any time it changes.

let clsValue = 0;  
let clsEntries = [];  
  
let sessionValue = 0;  
let sessionEntries = [];  
  
new PerformanceObserver((entryList) => {  
 for (const entry of entryList.getEntries()) {  
 // Only count layout shifts without recent user input.  
 if (!entry.hadRecentInput) {  
 const firstSessionEntry = sessionEntries[0];  
 const lastSessionEntry = sessionEntries[sessionEntries.length - 1];  
  
 // If the entry occurred less than 1 second after the previous entry and  
 // less than 5 seconds after the first entry in the session, include the  
 // entry in the current session. Otherwise, start a new session.  
 if (sessionValue &&  
 entry.startTime - lastSessionEntry.startTime < 1000 &&  
 entry.startTime - firstSessionEntry.startTime < 5000) {  
 sessionValue += entry.value;  
 sessionEntries.push(entry);  
 } else {  
 sessionValue = entry.value;  
 sessionEntries = [entry];  
 }  
  
 // If the current session value is larger than the current CLS value,  
 // update CLS and the entries contributing to it.  
 if (sessionValue > clsValue) {  
 clsValue = sessionValue;  
 clsEntries = sessionEntries;  
  
 // Log the updated value (and its entries) to the console.  
 console.log('CLS:', clsValue, clsEntries)  
 }  
 }  
 }  
}).observe({type: 'layout-shift', buffered: true});

**Warning**:

This code shows the basic way to calculate and log CLS. However, accurately measuring CLS in a way that matches what is measured in the [Chrome User Experience Report](https://developers.google.com/web/tools/chrome-user-experience-report) (CrUX) is more complicated. See below for details:

In most cases, the current CLS value at the time the page is being unloaded is the final CLS value for that page, but there are a few important exceptions:

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

**How to improve CLS** [**#**](https://web.dev/cls/#how-to-improve-cls)

**New:** Check out [Web Vitals Patterns](https://web.dev/patterns/web-vitals-patterns) for implementations of common UX patterns optimized for Core Web Vitals. This collection includes patterns that are often tricky to implement without layout shifts.

For most websites, you can avoid all unexpected layout shifts by sticking to a few guiding principles:

* **Always include size attributes on your images and video elements, or otherwise reserve the required space with something like** [**CSS aspect ratio boxes**](https://css-tricks.com/aspect-ratio-boxes/)**.** This approach ensures that the browser can allocate the correct amount of space in the document while the image is loading. Note that you can also use the [unsized-media feature policy](https://github.com/w3c/webappsec-feature-policy/blob/master/policies/unsized-media.md) to force this behavior in browsers that support feature policies.
* **Never insert content above existing content, except in response to a user interaction.** This ensures any layout shifts that occur are expected.
* **Prefer transform animations to animations of properties that trigger layout changes.** Animate transitions in a way that provides context and continuity from state to state.

For a deep dive on how to improve CLS, see [Optimize CLS](https://web.dev/optimize-cls/) and [Debug layout shifts](https://web.dev/debug-layout-shifts) .