Native image lazy-loading for the web!

**APRIL 6, 2019**

In this post, we'll look at the new [loading](https://github.com/scott-little/lazyload) attribute which brings native <img> and <iframe> lazy-loading to the web!. For the curious, here's a sneak preview of it in action:

<img src="celebration.jpg" loading="lazy" alt="..." />

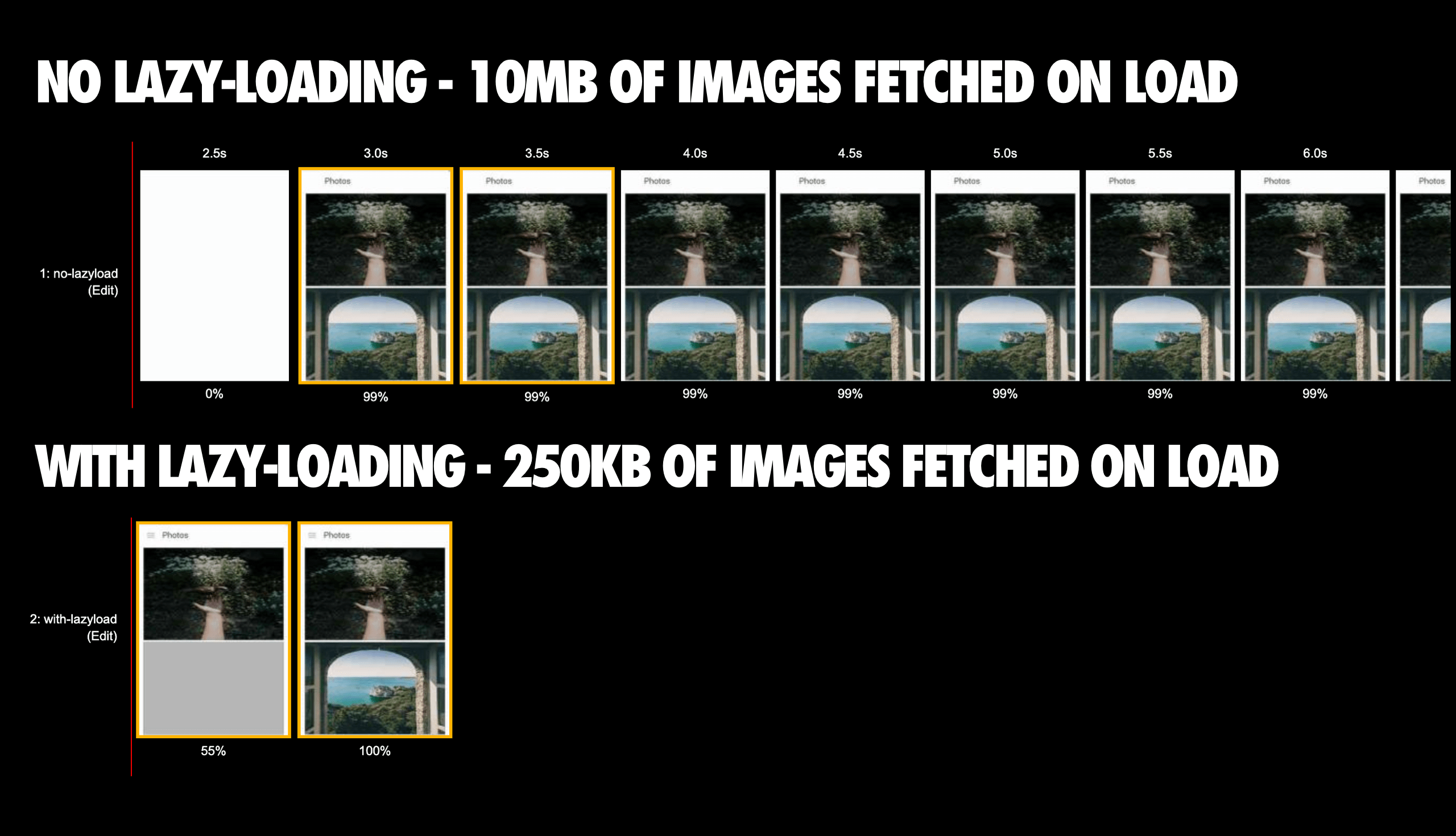
<iframe src="video-player.html" loading="lazy"></iframe>

We are hoping to ship support for loading in ~[Chrome 75](https://chromestatus.com/feature/5645767347798016) and are working on a deep-dive of the feature we'll publish soon. Until then, let's dive into how loading works.

Introduction

Web pages often contain a large number of images, which contribute to data-usage, [page-bloat](https://httparchive.org/reports/state-of-images) and how fast a page can load. Many of these images are offscreen, requiring a user to scroll in order to view them.

Historically, to limit the impact offscreen images have on page load times, developers have needed to use a JavaScript library (like [LazySizes](https://github.com/aFarkas/lazysizes)) in order to defer fetching these images until a user scrolls near them.

A page loading 211 images. The version without lazy-loading fetches 10MB of image data. The lazy-loading version (using LazySizes) loads only 250KB upfront - other images are fetched as the user scrolls through the experience. See [WPT](https://webpagetest.org/video/compare.php?tests=190406_2K_30b9b9cd6b48735a41bce2daee27b7f5,190406_6R_4ce0ac65b7e11d2e132e4ea8d887edd9).

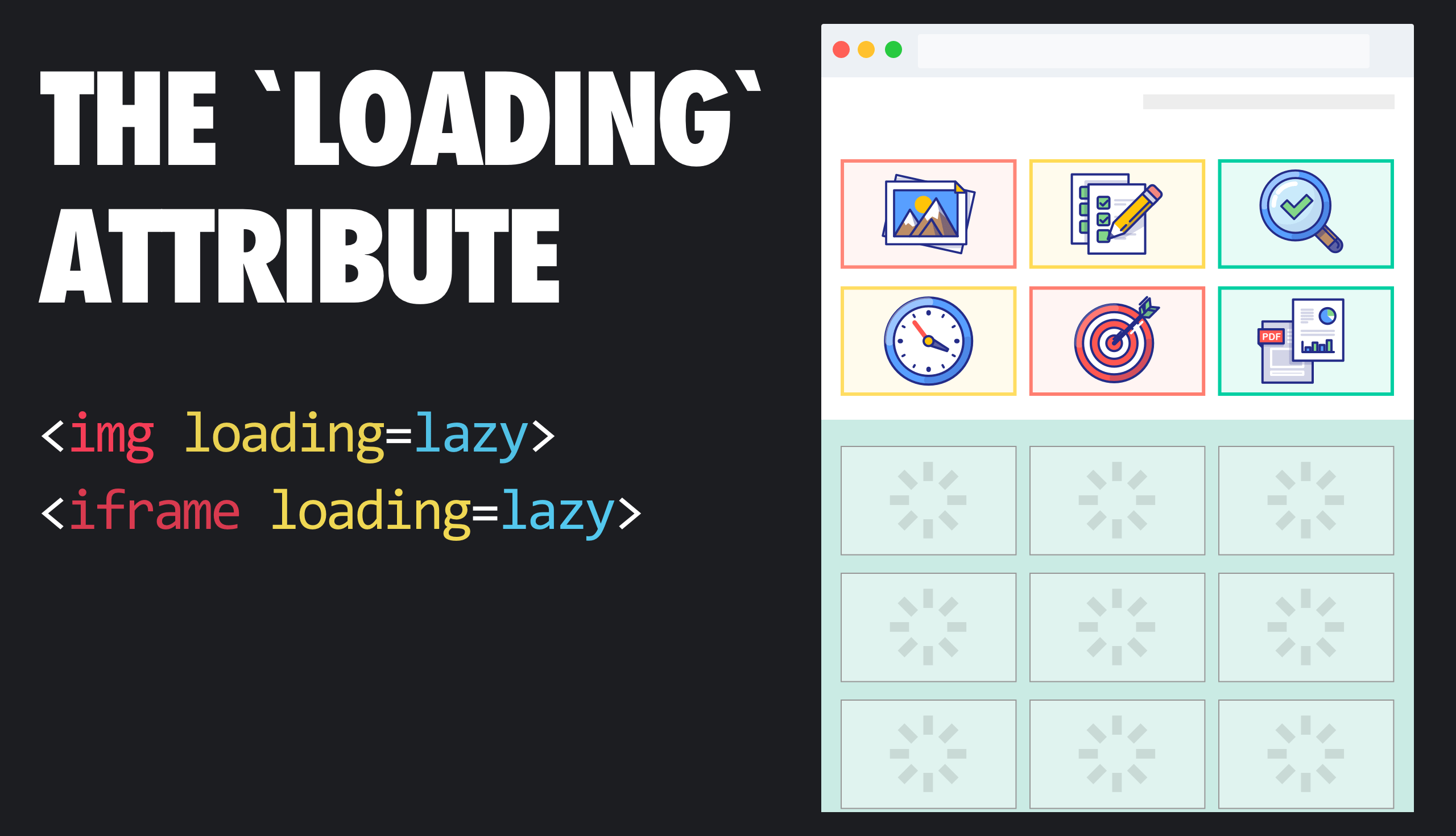
What if the browser could avoid loading these offscreen images for you? This would help content in the view-port load quicker, reduce overall network data usage and on lower-end devices, reduce memory usage. Well, I'm happy to share that this will soon be possible with the new loading attribute for images and iframes.

The loading attribute

The loading attribute allows a browser to defer loading offscreen images and iframes until users scroll near them. loading supports three values:

* lazy: is a good candidate for lazy loading.
* eager: is not a good candidate for lazy loading. Load right away.
* auto: browser will determine whether or not to lazily load.

Not specifying the attribute at all will have the same impact as setting loading=auto.



The loading attribute for <img> and <iframe> is being worked on as part of the [HTML standard](https://github.com/whatwg/html/pull/3752).

Examples

The loading attribute works on <img> (including with srcset and inside <picture>) as well as on <iframe>:

<!-- Lazy-load an offscreen image when the user scrolls near it -->

<img src="unicorn.jpg" loading="lazy" alt=".."/>

<!-- Load an image right away instead of lazy-loading -->

<img src="unicorn.jpg" loading="eager" alt=".."/>

<!-- Browser decides whether or not to lazy-load the image -->

<img src="unicorn.jpg" loading="auto" alt=".."/>

<!-- Lazy-load images in <picture>. <img> is the one driving image

loading so <picture> and srcset fall off of that -->

<picture>

<source media="(min-width: 40em)" srcset="big.jpg 1x, big-hd.jpg 2x">

<source srcset="small.jpg 1x, small-hd.jpg 2x">

<img src="fallback.jpg" loading="lazy">

</picture>

<!-- Lazy-load an image that has srcset specified -->

<img src="small.jpg"

srcset="large.jpg 1024w, medium.jpg 640w, small.jpg 320w"

sizes="(min-width: 36em) 33.3vw, 100vw"

alt="A rad wolf" loading="lazy">

<!-- Lazy-load an offscreen iframe when the user scrolls near it -->

<iframe src="video-player.html" loading="lazy"></iframe>

The exact heuristics for "when the user scrolls near" is left up to the browser. In general, our hope is that browsers will start fetching deferred images and iframe content a little before it comes into the viewport. This will increase the change the image or iframe is done loading by the time the user has scrolled to them.

Note: I [suggested](https://github.com/whatwg/html/pull/3752#issuecomment-478200976) we name this the loading attribute as its naming aligns closer with the [decoding](https://developer.mozilla.org/en-US/docs/Web/API/HTMLImageElement/decoding) attribute. Previous proposals, such as the lazyload attribute didn't make it as we needed to support multiple values (lazy, eager and auto).

Feature detection

We've kept in mind the importance of being able to fetch and apply a JavaScript library for lazy-loading (for the cross-browser support). Support for loading can be feature-detected as follows:

<script>

if ('loading' in HTMLImageElement.prototype) {

// Browser supports `loading`..

} else {

// Fetch and apply a polyfill/JavaScript library

// for lazy-loading instead.

}

</script>

Note: You can also use loading as a progressive enhancement. Browsers that support the attribute can get the new lazy-loading behavior with loading=lazy and those that don't will still have images load.

Cross-browser image lazy-loading

If cross-browser support for lazy-loading images is important, it's not enough to just feature-detect and lazy-load a library if you're using <img src=unicorn.jpg loading=lazy /> in the markup.

The markup needs to use something like <img data-src=unicorn.jpg /> (rather than src, srcset or <source>) to avoid triggering an eager load in browsers that don't support the new attribute. JavaScript can be used to change those attributes to the proper ones if loading is supported and load a library if not. You could think of this as hybrid lazy-loading.

Below is an example of what this could look like.

* In-viewport / above-the-fold images are regular <img> tags. A data-src would defeat the preload scanner so we want to avoid it for everything likely to be in the viewport.
* We use data-src on images to avoid an eager load in unsupported browsers. If loading is supported, we swap data-src for src.
* If loading is not supported, we load a fallback (LazySizes) and initiate it. Here, we use class=lazyload as a way to indicate to LazySizes images we want to be lazily-loaded.

<!-- Let's load this in-viewport image normally -->

<img src="hero.jpg" alt=".."/>

<!-- Let's lazy-load the rest of these images -->

<img data-src="unicorn.jpg" loading="lazy" alt=".." class="lazyload"/>

<img data-src="cats.jpg" loading="lazy" alt=".." class="lazyload"/>

<img data-src="dogs.jpg" loading="lazy" alt=".." class="lazyload"/>

<script>

if ('loading' in HTMLImageElement.prototype) {

const images = document.querySelectorAll("img.lazyload");

images.forEach(img => {

img.src = img.dataset.src;

});

} else {

// Dynamically import the LazySizes library

let script = document.createElement("script");

script.async = true;

script.src =

"https://cdnjs.cloudflare.com/ajax/libs/lazysizes/4.1.8/lazysizes.min.js";

document.body.appendChild(script);

}

</script>

Here's an alternative to the above that relies on [dynamic import](https://developers.google.com/web/updates/2017/11/dynamic-import) to perform the same fallback library fetching:

<!-- Let's load this in-viewport image normally -->

<img src="hero.jpg" alt=".."/>

<!-- Let's lazy-load the rest of these images -->

<img data-src="unicorn.jpg" loading="lazy" alt=".." class="lazyload"/>

<img data-src="cats.jpg" loading="lazy" alt=".." class="lazyload"/>

<img data-src="dogs.jpg" loading="lazy" alt=".." class="lazyload"/>

<script>

(async () => {

if ('loading' in HTMLImageElement.prototype) {

const images = document.querySelectorAll("img.lazyload");

images.forEach(img => {

img.src = img.dataset.src;

});

} else {

// Dynamically import the LazySizes library

const lazySizesLib = await import('/lazysizes.min.js');

// Initiate LazySizes (reads data-src & class=lazyload)

lazySizes.init(); // lazySizes works off a global.

}

})();

</script>

Andrea Verlicchi has a good article that also looks at [hybrid lazy-loading](https://www.smashingmagazine.com/2019/05/hybrid-lazy-loading-progressive-migration-native/) worth reading.

Demo

[A loading=lazy demo featuring exactly 100 kitten pics](https://mathiasbynens.be/demo/img-loading-lazy) is available. Check it out!

I've also recorded a video of the feature in action you can check out above.

Chrome implementation details

We strongly recommend waiting until the loading attribute is in a stable release before using it in production. Early testers may find the below notes helpful.

Try today

Go to chrome://flags and turn on both the "Enable lazy frame loading" and "Enable lazy image loading" flags, then restart Chrome.

Configuration

Chrome’s lazy-loading implementation is based not just on how near the current scroll position is, but also the connection speed. The lazy frame and image loading distance-from-viewport thresholds for different connection speeds are [hardcoded](https://cs.chromium.org/chromium/src/third_party/blink/renderer/core/frame/settings.json5?l=937-1003&rcl=e8f3cf0bbe085fee0d1b468e84395aad3ebb2cad), but can be overriden from the command-line. Here’s an example that overrides the lazy-loading settings for images:

canary --user-data-dir="$(mktemp -d)" --enable-features=LazyImageLoading --blink-settings=lazyImageLoadingDistanceThresholdPxUnknown=5000,lazyImageLoadingDistanceThresholdPxOffline=8000,lazyImageLoadingDistanceThresholdPxSlow2G=8000,lazyImageLoadingDistanceThresholdPx2G=6000,lazyImageLoadingDistanceThresholdPx3G=4000,lazyImageLoadingDistanceThresholdPx4G=3000 'https://mathiasbynens.be/demo/img-loading-lazy'

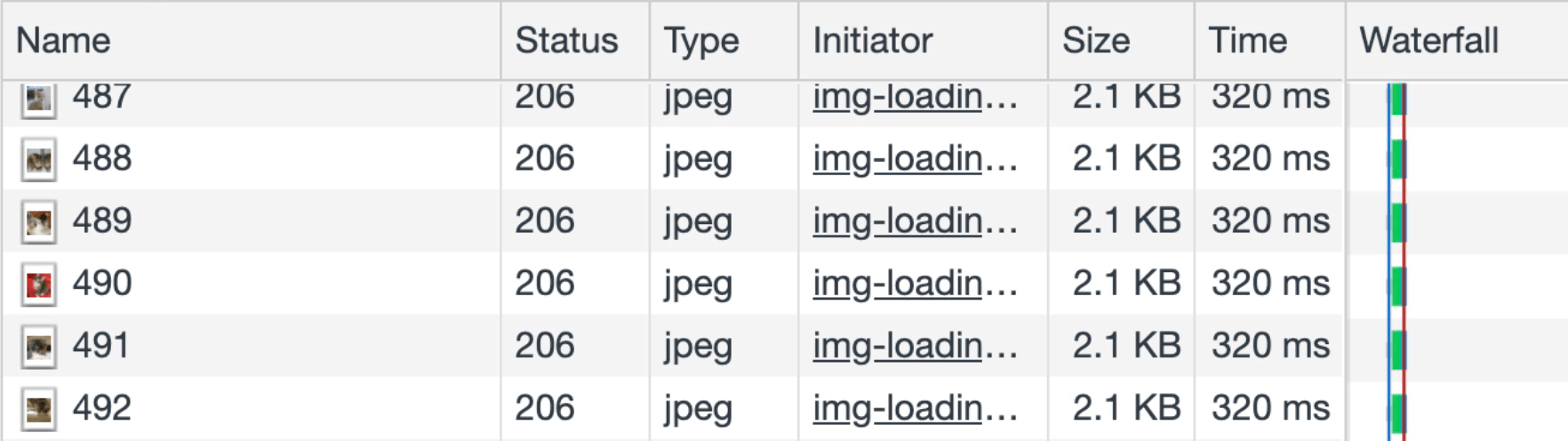
The above command corresponds to the (current) default configuration. Change all the values to 400 to start lazy-loading only when the scroll position is within 400 pixels of the image. Below we can also see a 1 pixel variation (which the video earlier in the article uses):

canary --user-data-dir="$(mktemp -d)" --enable-features=LazyImageLoading --blink-settings=lazyImageLoadingDistanceThresholdPxUnknown=1,lazyImageLoadingDistanceThresholdPxOffline=1,lazyImageLoadingDistanceThresholdPxSlow2G=1,lazyImageLoadingDistanceThresholdPx2G=1,lazyImageLoadingDistanceThresholdPx3G=1,lazyImageLoadingDistanceThresholdPx4G=1 'https://mathiasbynens.be/demo/img-loading-lazy'

It’s very likely our default configuration will change as the implementation stabilizes over the coming weeks.

DevTools

An implementation detail of loading in Chrome is that it fetches the first 2KB of images on page-load. If the server supports range requests, the first 2KB likely contains image dimensions. This enables us to generate/display a placeholder with the same dimensions. First 2KB also likely includes the whole image for assets like icons.



Chrome fetches the rest of the image bytes when the user is about to see them. A caveat for Chrome DevTools is that this can result in (1) double fetches to 'appear' in the DevTools Network panel and (2) Resource Timing to have 2 requests for each image.

Determine loading support on the server

In a perfect world, you wouldn't need to rely on JavaScript feature detection on the client to decide whether or not a fallback library needs to be loaded - you would handle this before serving HTML that includes a JavaScript lazy-loading library. A Client Hint could enable such a check.

A hint for conveying loading preferences is being [considered](https://bugs.chromium.org/p/chromium/issues/detail?id=949365) but is currently in the early discussion phase.

Wrapping up

Give <img loading> a spin and let us know what you think. I'm particularly interested in how folks find the cross-browser story and whether there are any edge-cases we've missed. We hope to ship the loading attribute this summer around Chrome 76.

References

* [Intent to ship this feature in Blink](https://groups.google.com/a/chromium.org/forum/#!msg/blink-dev/jxiJvQc-gVg/wurng4zZBQAJ)
* [Specification PR](https://github.com/whatwg/html/pull/3752)
* [Explainer](https://github.com/scott-little/lazyload)
* [Demo](https://mathiasbynens.be/demo/img-loading-lazy)

*With thanks to Simon Pieters, Yoav Weiss and Mathias Bynens for their feedback. A large thanks to Ben Greenstein, Scott Little, Raj T and Houssein Djirdeh for their work on LazyLoad.*

Last updated May 7th, 2019