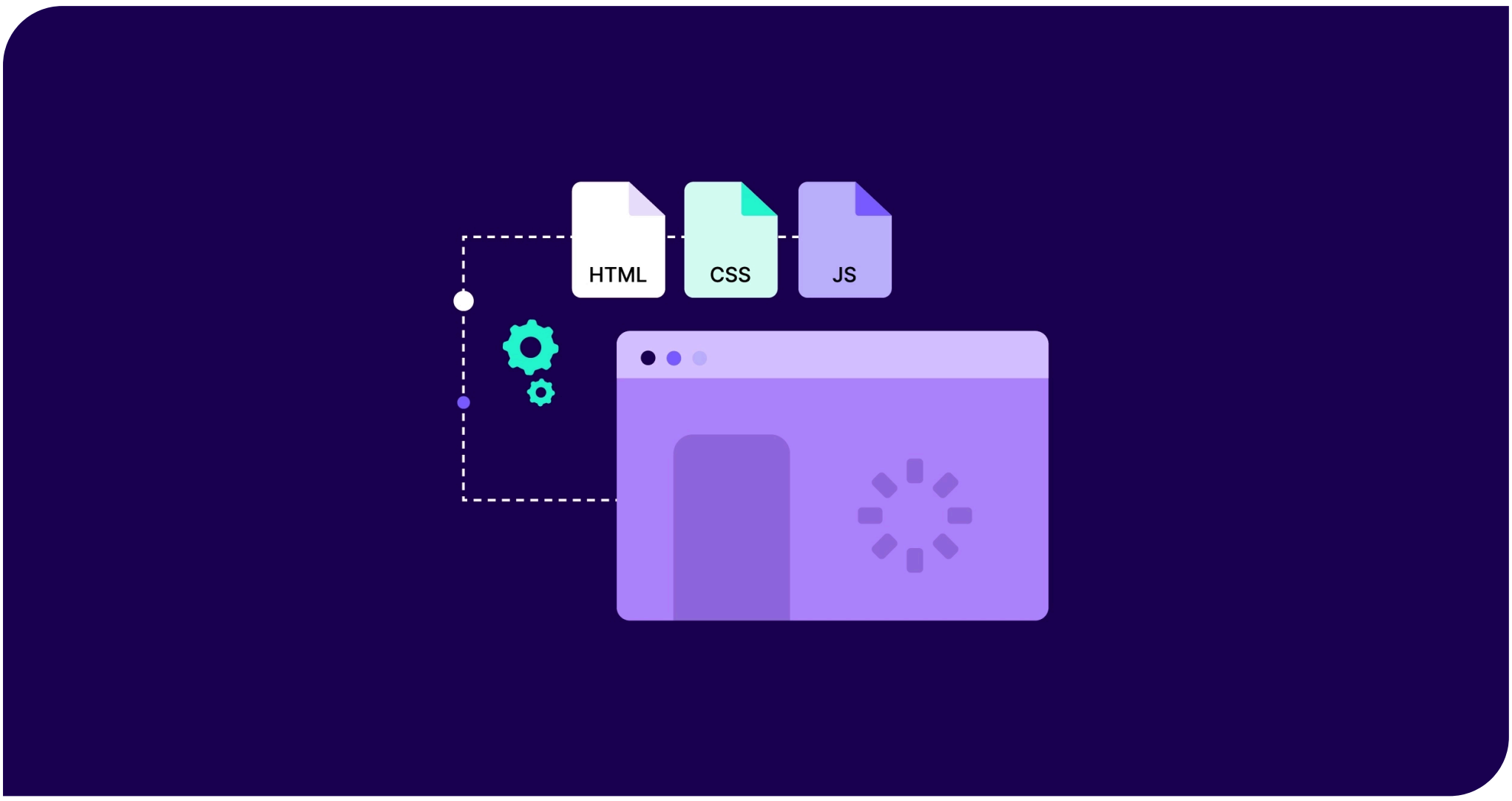
Critical Rendering Path: What It Is and How to Optimize It

Last updated on Feb 14th, 2024 | 7 min



TL;DR: The Critical Rendering Path is how browsers turn code into a visible webpage. To make pages load faster, optimize this path. Reduce file sizes of CSS and JavaScript, load important content first, and use asynchronous loading for less important scripts. Optimize images, use browser caching, and utilize CDNs for efficient resource distribution.

When we talk about providing users with an ultra-fast web experience, we often focus solely on what we, as website owners and web developers, should do.

But the truth is:

Delivering a fast web experience also requires a lot of work by the browser.

It receives our HTML, CSS, and JavaScript files and takes specific steps to convert them into pixels on the screen.

The secret to speeding up your performance lies in understanding what happens between receiving the resources and their processing to turn them into rendered pixels.

This process is also known as the **critical rendering path (CRP)**.

And in this article, you'll learn everything you need to know about CRP, and how to optimize it for faster rendering.

- What is Critical Rendering Path?
- The Critical Rendering Path sequence explained
- How to optimize your site's Critical Rendering Path

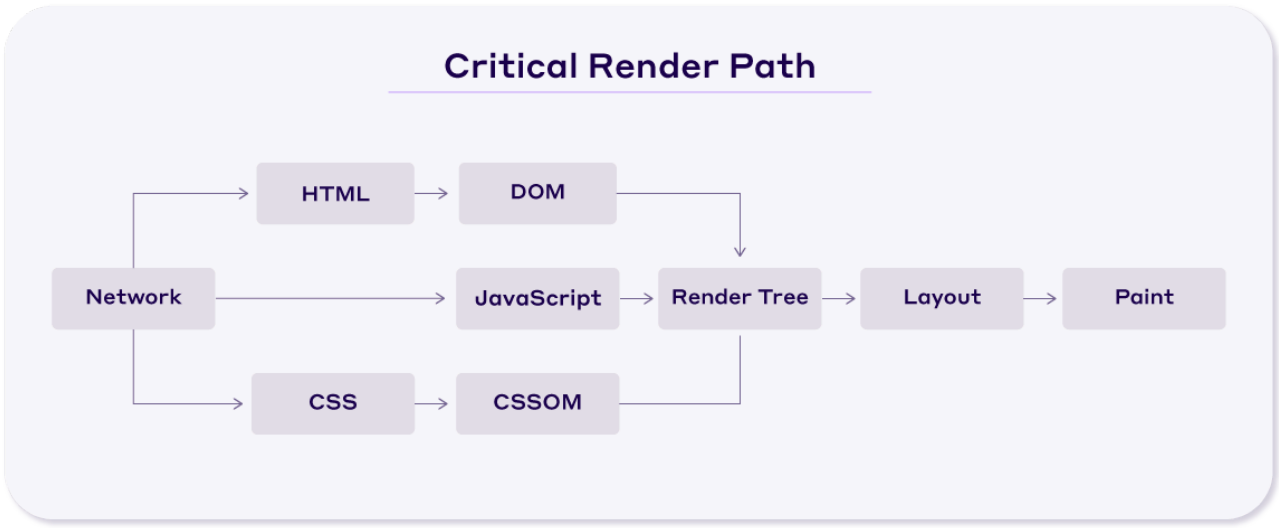
- 3 WordPress plugins to optimize your CRP
- CRP optimization checklist

Let’s begin!

What is Critical Rendering Path?

The Critical Rendering Path refers to the sequence of steps that a web browser takes to convert HTML, CSS, and JavaScript code into a visual representation on a user's screen.

It involves a series of processes, such as constructing the Document Object Model (DOM), generating the CSS Object Model (CSSOM), and combining both to create the Render Tree. The Render Tree is then used to calculate the layout and paint the pixels on the user's screen.



Critical Rendering Path optimization, on the other hand, refers to reducing the time spent by the web browser to execute each step of the sequence while prioritizing the content relevant to the user's current action.

To ensure your optimization efforts hit the nail on the head, you need to have an in-depth understanding of each step of the sequence. So the next couple of paragraphs are essential, and we strongly recommend reading them before taking action.

The Critical Rendering Path Sequence Explained

Here’s a quick overview of the steps performed by the browser when rendering a page:

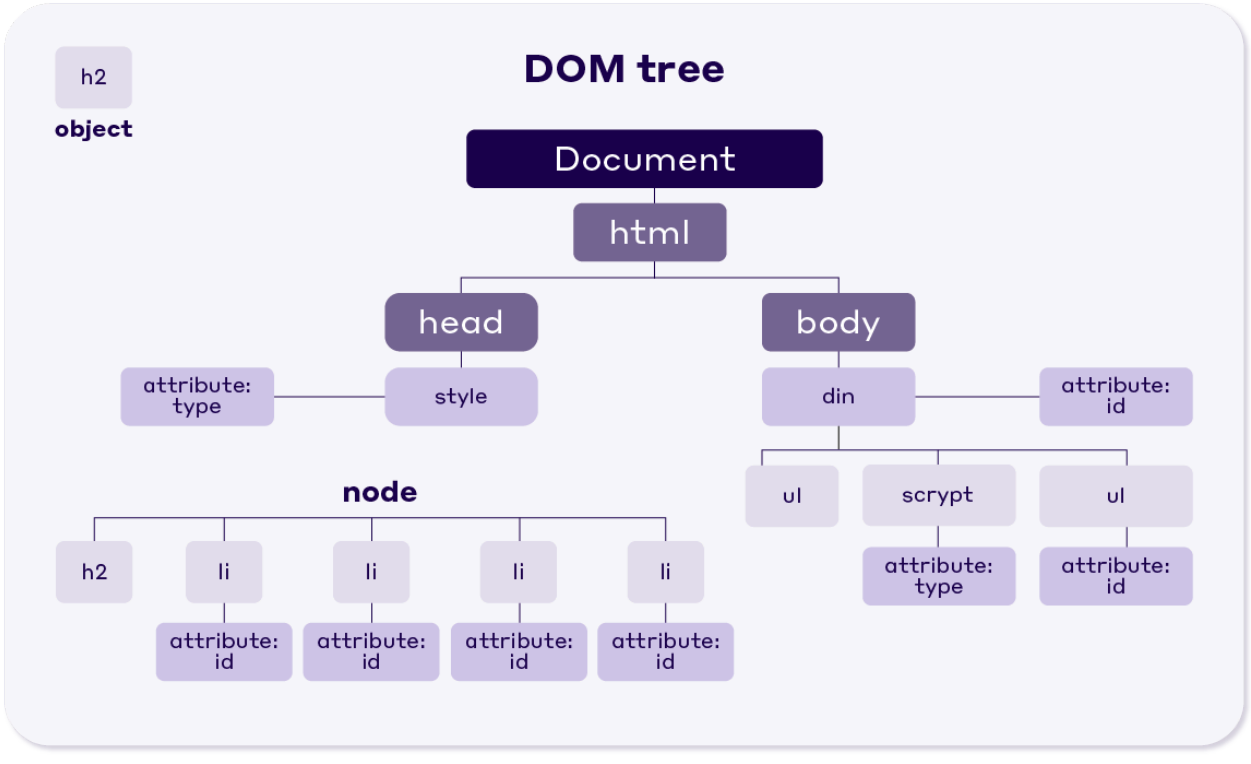
1. The browser downloads and parses the HTML markup and creates the DOM.
2. Next, it downloads and processes the CSS markup and constructs the CSS Object Model (CSSOM).
3. Then, it combines the necessary nodes from the DOM and CSSOM to create the Render Tree, a tree structure of all visible nodes required to render the page.
4. It calculates the dimensions and position of every element on the page through the Layout process.
5. Finally, the browser paints the pixels on the screen.

Now let’s zone in on each step.

The DOM

The Document Object Model (DOM) is the browser's internal representation of the HTML document.

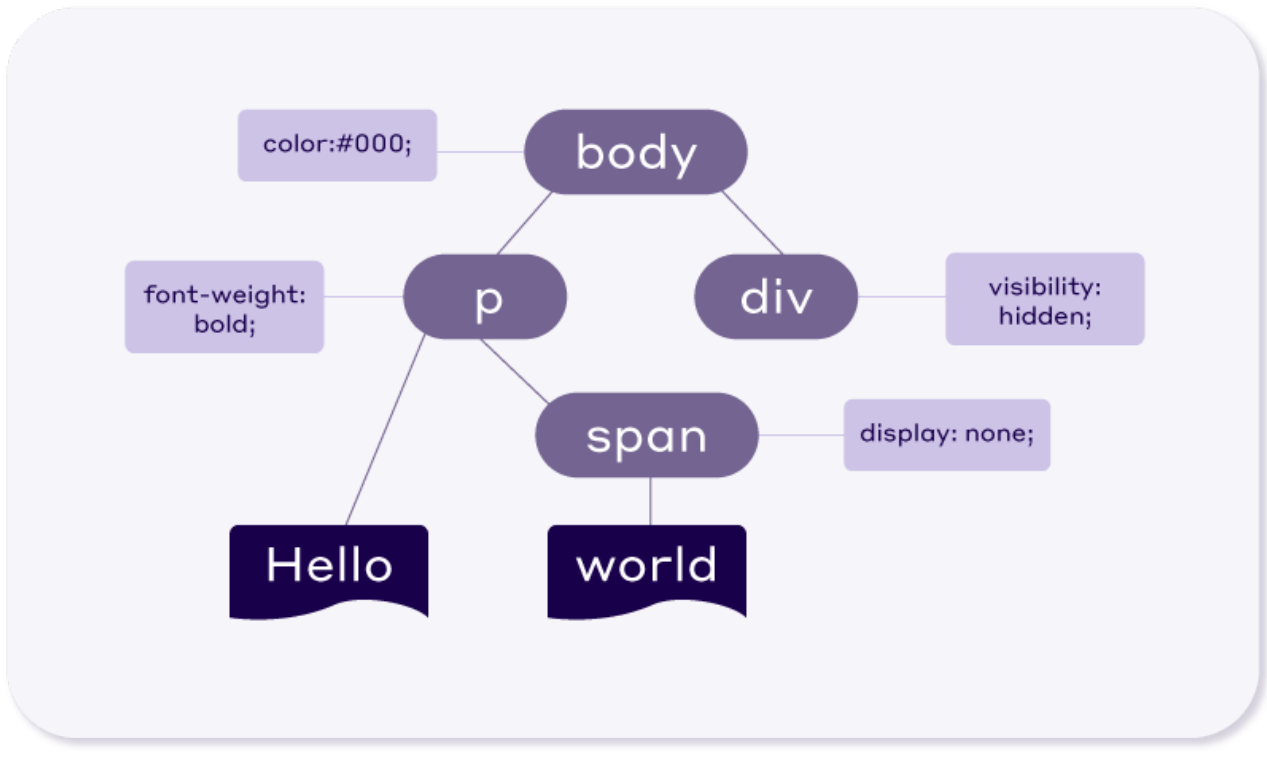
When a web page is loaded, the browser parses the HTML and creates a tree-like structure of nodes that represent the elements in the document. Each node corresponds to an HTML element and has properties that describe its attributes, content, and position in the tree.



Important: The browser builds the DOM gradually, allowing us to optimize the rendering of the page by constructing an efficient structure and avoiding excessive DOM sizes.

The CSSOM

While the DOM contains all the content of the page, the CSSOM includes all the information on how to style the DOM.



Another difference between DOM and CSSOM is that:

DOM construction is gradual, while CSSOM is not.

When a website is loaded, the browser has to process the CSS to apply the styles. Unlike HTML, which can be processed bit by bit, CSS needs to be processed all at once. This is because some styles might be overwritten by others later in the CSS file, so the browser needs to wait until it has read the whole CSS file before deciding which styles to apply.

This is done to avoid showing styles that will later be overwritten and wasting resources.

Simply put:

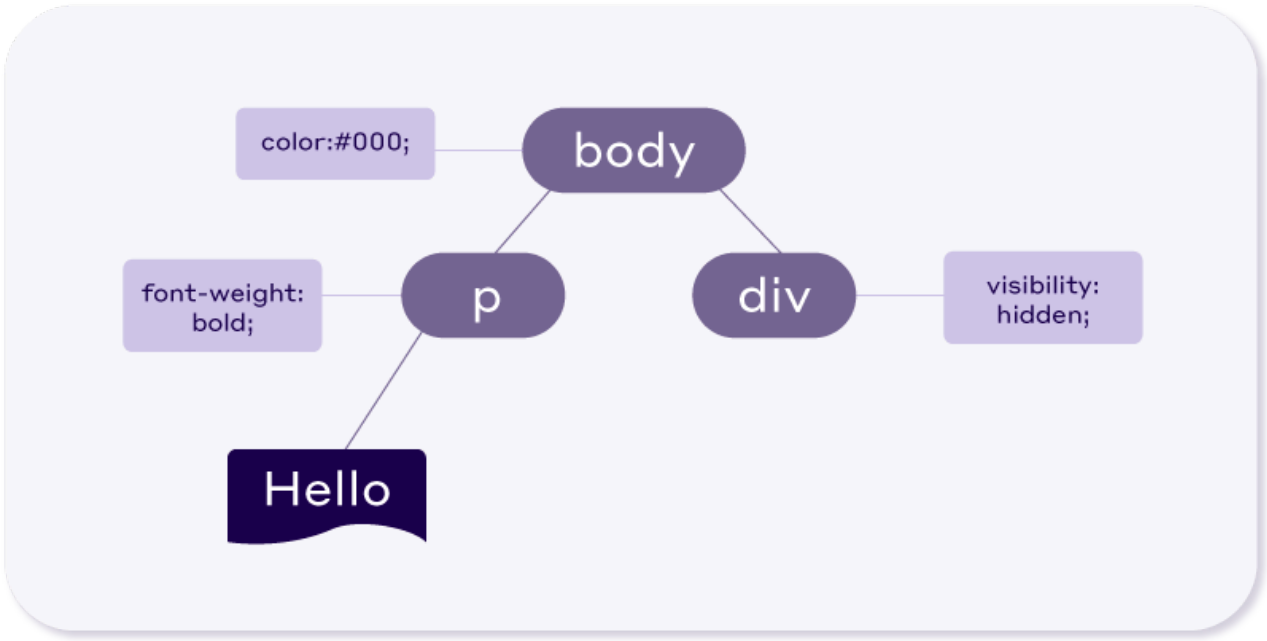
The browser blocks the rendering process until it receives and parses all the CSS.

That’s why CSS is considered a render-blocking resource.

The Render Tree

The Render Tree is the combination of the DOM and CSSOM that the browser uses to create the visual representation of the web page.

The browser uses the Render Tree to calculate node dimensions and position as input for the painting process.



Important: Only visible content is captured in the render tree. Typically, the head section contains no visible information and is therefore excluded. Furthermore, If an element has a *display: none* property, neither the element nor its descendants are included in the render tree.

Layout

After the render tree is constructed, the next step is the layout. The layout establishes the placement and orientation of each element on the page by defining its dimensions, position, and interrelationships.

But here's the thing:

The layout performance is impacted by the DOM.

In other words:

The greater the number of DOM nodes, the longer the layout process is.

Paint

The final stage is painting the pixels onto the screen, which follows the creation of the render tree and the layout.

Initially, the entire screen is painted during the load process. Subsequently, only the affected parts of the screen are repainted, as browsers are designed to repaint only the necessary area.

Keep in mind that the duration of the paint stage depends on the nature of the updates being implemented on the render tree.

Now let's see what optimizations you can apply to help the browser and speed up some of the processes.

How to Optimize Your Site's Critical Rendering Path

The time required for the browser to run through the entire process can vary. There are a lot of moving parts that contribute to the critical path length:

- Document size
- Number of requests
- User device
- Applied styles

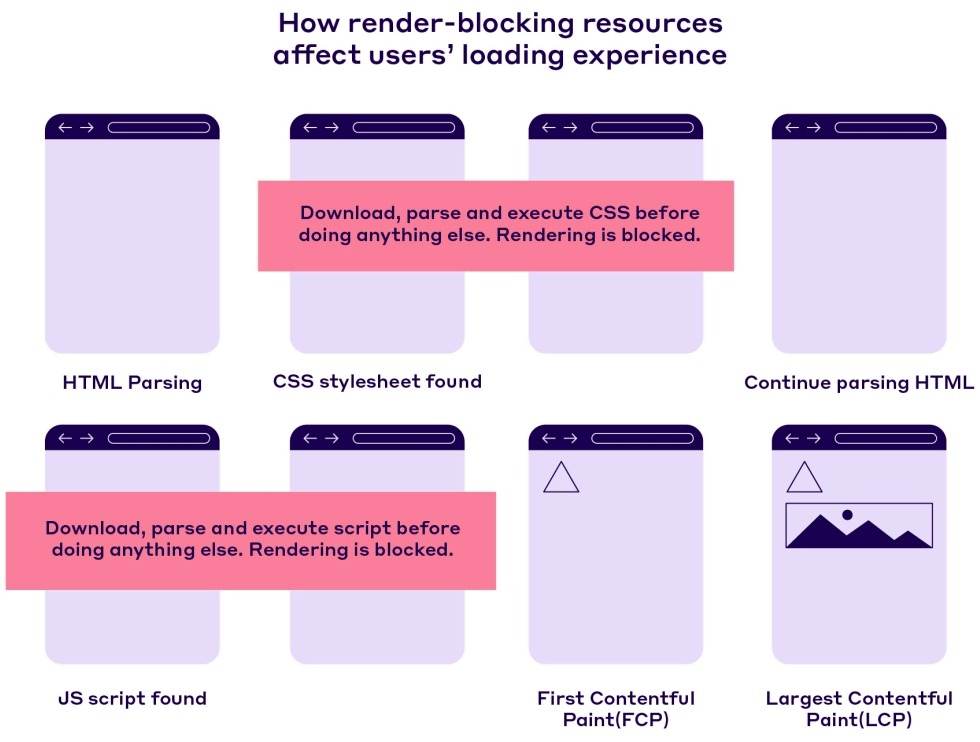
Nevertheless, there are three techniques that are considered to be the go-to options when it comes to CRP optimization:

1. Minimize the number of critical resources by deferring non-critical ones or eliminating them altogether
2. Optimize the number of requests required along with the file size of each request
3. Prioritize the downloading of critical assets, thereby shortening the critical path length

Let’s dive a little deeper into how to implement each of the recommended optimization strategies:

Optimize render-blocking CSS and JS resources

You already know that when the browser encounters render-blocking CSS and JS resources, it must download, parse, and execute them before doing anything else, including rendering.



When it comes to optimizing CSS, you can implement the following techniques:

- Critical CSS. It identifies the minimal set of CSS rules needed to render the visible portion of a webpage and delivers them to the browser as inline CSS rather than loading a full stylesheet. By loading only the necessary CSS for above-the-fold content, **the browser can render the page more quickly and improve the user experience**. This is because the browser doesn't have to wait for the entire stylesheet to be loaded before rendering the page.

- **Combine CSS files.** CSS concatenation is the process of combining multiple CSS files into a single file. This technique **improves performance by reducing the number of HTTP requests required to load a webpage**, as the browser only needs to download and parse a single CSS file instead of multiple ones.

In terms of your JavaScript files, here’s what you can do:

- **Delay JS loading.** Defer JS loading is a technique that **will speed up your site by delaying loading JavaScript files until after the HTML document has been loaded and parsed**. You can use *defer* attribute on the *script* tag that references the JS file. It's important to note that the defer attribute should only be used for JS files that do not need to be executed immediately upon loading (e.g., files that are only used on specific pages), as the order of execution may be unpredictable if multiple deferred scripts are used.

- **Load JS asynchronously.** Some JS files may require the use of the async attribute, which allows the file to be loaded and executed asynchronously with the parsing of the HTML document.

There are a few optimizations you can apply to both CSS and JavaScript:

- **Minification.** Minification involves removing unnecessary characters, such as white space, comments, and line breaks, from CSS and JavaScript files. This process **can significantly reduce the size of the files without affecting their functionality or appearance**.

- **Remove unused CSS and JS.** Unused CSS and JS refer to specific rules that aren't used on a particular page but are still loaded. **Removing these parts of your files will directly affect how fast the browser builds the render tree**.

Reduce the size of your files

To reduce the amount of data the browser needs to download, we can employ techniques such as minification, compression, and caching of HTML, CSS, and JavaScript resources.

You already know what minification means, so let's focus on the other two:

- **Compression.** Compression is a technique that applies algorithms to rewrite the files' binary code using fewer bits than the original. As a result, your files are of much smaller size which reduces page load times and bandwidth usage.

- **Caching.** Caching takes advantage of the HTTP cache implemented in every browser. To ensure effective caching, we must ensure that each server response provides the correct HTTP headers, instructing the browser when and how long it should cache the requested resources.

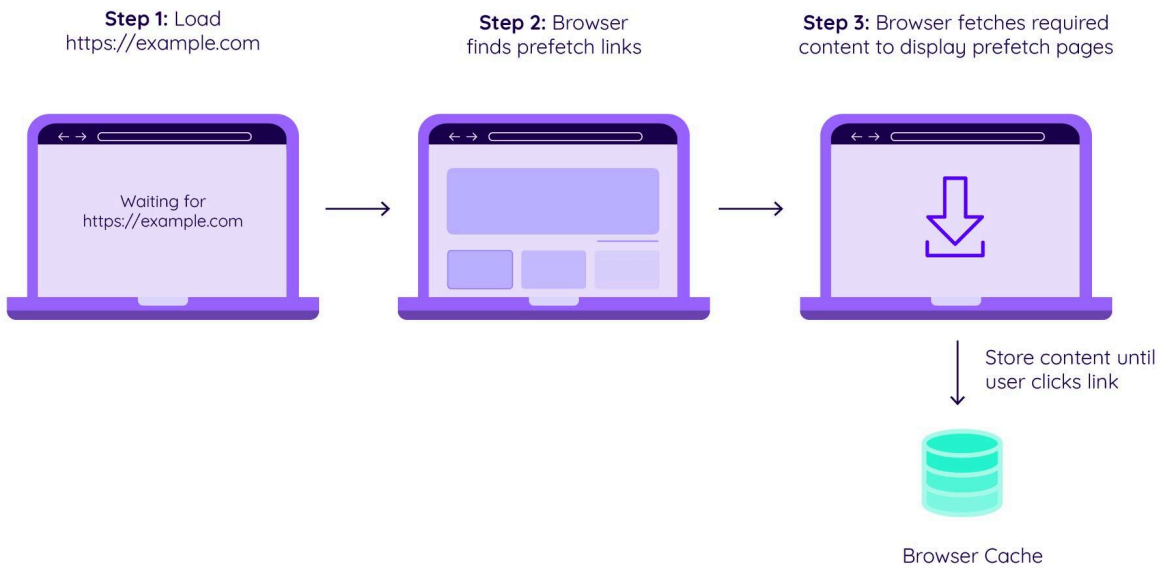
Prioritize the downloading of critical assets

In general, browsers are pretty good at prioritizing the most important resources and fetching them first. However, in some cases, you could help them load your site even faster by manually prioritizing the most crucial resources.

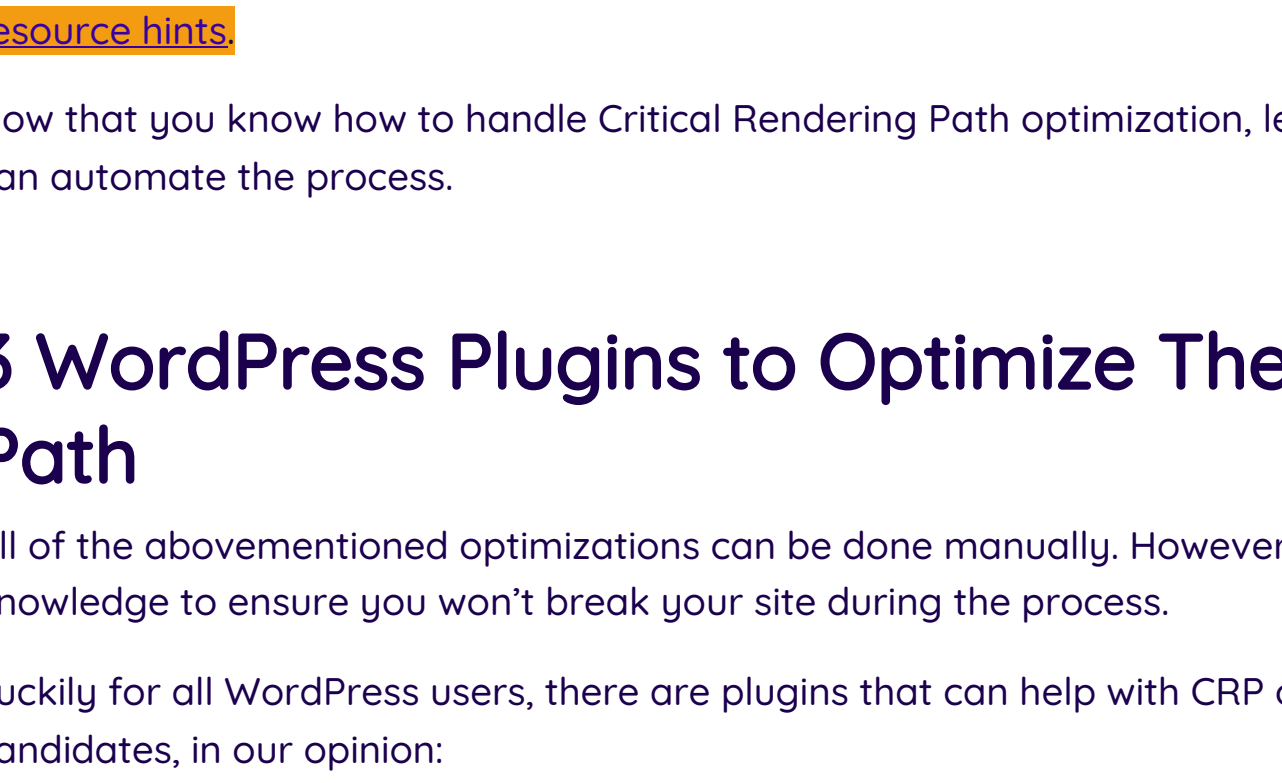
You can use *resource hints* to tell the browser how to handle specific resources or web pages.

Here are the three main ones:

- **Link rel=prefetch.** Prefetch is a low-priority resource hint that allows the browser to fetch resources that might be needed later and store them in the browser's cache.



- **Link rel=preconnect.** The preconnect directive helps the browser establish early connections before sending an initial request to the server.



- **Link rel=preload.** Preload is used to force the browser to download a resource sooner than the browser would discover it because it is crucial for the page.

Important: Prefetch and preconnect are resource hints, and they are executed as the browser sees fit. The preload directive is a command which is mandatory for the browsers. Learn more about how to implement resource hints.

Now that you know how to handle Critical Rendering Path optimization, let's look at some WordPress plugins that can automate the process.

3 WordPress Plugins to Optimize The Critical Rendering Path

All of the abovementioned optimizations can be done manually. However, some of them require technical knowledge to ensure you won't break your site during the process.

Luckily for all WordPress users, there are plugins that can help with CRP optimization. Let's check the top 3 candidates, in our opinion: