**How Browsers Work**

**1. Điều hướng (Navigation)**

Điều hướng là bước đầu tiên trong quá trình tải một trang web. Nó xảy ra bất cứ khi nào người dùng yêu cầu một trang bằng cách nhập URL vào thanh địa chỉ, nhấp vào liên kết, gửi biểu mẫu cũng như các hành động khác.

Một trong những mục tiêu của hiệu suất web là giảm thiểu lượng thời gian hoàn thành điều hướng. Trong điều kiện lý tưởng, điều này thường không mất quá nhiều thời gian, nhưng độ trễ và băng thông là những kẻ thù có thể gây ra sự chậm trễ.

**2. Tra cứu DNS**

Bước đầu tiên để điều hướng đến một trang web là tìm vị trí của nội dung cho trang đó. Nếu bạn điều hướng đến https://example.com, trang HTML nằm trên máy chủ có địa chỉ IP là 93.184.216.34. Nếu bạn chưa bao giờ truy cập trang web này, bạn phải tiến hành tra cứu DNS.

Trình duyệt của bạn yêu cầu tra cứu DNS, tra cứu này cuối cùng sẽ được thực hiện bởi một máy chủ định danh, máy chủ này sẽ phản hồi bằng một địa chỉ IP. Sau yêu cầu ban đầu này, IP có thể sẽ được lưu vào bộ đệm trong một thời gian, giúp tăng tốc các yêu cầu tiếp theo bằng cách truy xuất địa chỉ IP từ bộ đệm thay vì liên hệ lại với máy chủ định danh.

Tra cứu DNS thường chỉ cần được thực hiện một lần cho mỗi tên máy chủ để tải trang. Tuy nhiên, việc tra cứu DNS phải được thực hiện cho từng tên máy chủ duy nhất mà trang được yêu cầu tham chiếu. Nếu phông chữ, hình ảnh, tập lệnh, quảng cáo và số liệu của bạn đều có tên máy chủ lưu trữ khác nhau, thì bạn sẽ phải thực hiện tra cứu DNS cho từng tên máy chủ.



Điều này có thể gây ra vấn đề về hiệu suất, đặc biệt là trên các mạng di động. Khi người dùng sử dụng mạng di động, mỗi lần tra cứu DNS phải đi từ điện thoại đến tháp di động để đến máy chủ DNS có thẩm quyền. Khoảng cách giữa điện thoại, tháp di động và máy chủ định danh có thể tăng thêm độ trễ đáng kể.

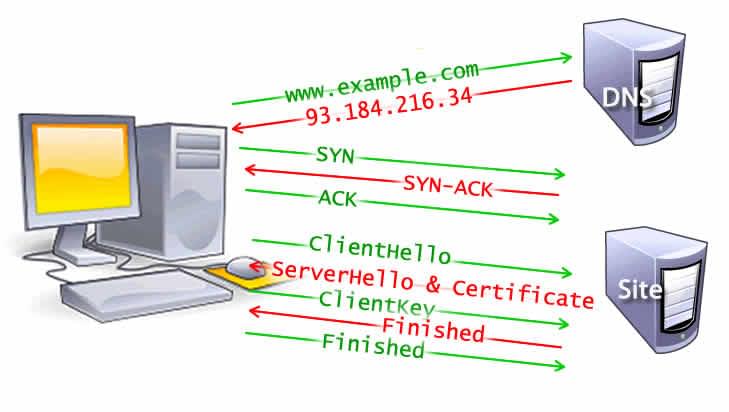
**3. TCP Handshake**

Sau khi biết địa chỉ IP, trình duyệt sẽ thiết lập kết nối với máy chủ thông qua [bắt tay ba bước TCP](https://developer.mozilla.org/en-US/docs/Glossary/TCP_handshake) . Cơ chế này được thiết kế sao cho hai thực thể cố gắng giao tiếp—trong trường hợp này là trình duyệt và máy chủ web—có thể thương lượng các tham số của kết nối ổ cắm TCP mạng trước khi truyền dữ liệu, thường là qua [HTTPS](https://developer.mozilla.org/en-US/docs/Glossary/HTTPS) .

Kỹ thuật bắt tay ba bước của TCP thường được gọi là "SYN-SYN-ACK"—hay chính xác hơn là SYN, SYN-ACK, ACK—vì có ba thông báo được truyền bởi TCP để đàm phán và bắt đầu một phiên TCP giữa hai máy tính. Có, điều này có nghĩa là có thêm ba tin nhắn qua lại giữa mỗi máy chủ và yêu cầu vẫn chưa được thực hiện.

**4. Đàm phán TLS**

Đối với các kết nối an toàn được thiết lập qua HTTPS, cần có một "bắt tay" khác. Quá trình bắt tay này, hay đúng hơn là đàm phán [TLS](https://developer.mozilla.org/en-US/docs/Glossary/TLS) , xác định mật mã nào sẽ được sử dụng để mã hóa thông tin liên lạc, xác minh máy chủ và thiết lập kết nối an toàn trước khi bắt đầu truyền dữ liệu thực tế. Điều này yêu cầu thêm ba chuyến đi khứ hồi tới máy chủ trước khi yêu cầu nội dung thực sự được gửi đi.



Mặc dù làm cho kết nối an toàn làm tăng thêm thời gian tải trang, nhưng kết nối an toàn đáng giá với chi phí độ trễ vì dữ liệu được truyền giữa trình duyệt và máy chủ web không thể được giải mã bởi bên thứ ba.

Sau 8 vòng lặp, trình duyệt cuối cùng cũng có thể thực hiện yêu cầu.

**5. Phản ứng**

Sau khi chúng tôi có kết nối được thiết lập với máy chủ web, trình duyệt sẽ gửi [yêu cầu HTTPGET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods) ban đầu thay mặt cho người dùng, yêu cầu này đối với các trang web thường là tệp HTML nhất. Khi máy chủ nhận được yêu cầu, nó sẽ trả lời với các tiêu đề phản hồi có liên quan và nội dung của HTML.

*<!DOCTYPE html>*

*<html lang="en-US">*

*<head>*

*<meta charset="UTF-8" />*

*<title>My simple page</title>*

*<link rel="stylesheet" href="styles.css" />*

*<script src="myscript.js"></script>*

*</head>*

*<body>*

*<h1 class="heading">My Page</h1>*

*<p>A paragraph with a <a href="https://example.com/about">link</a></p>*

*<div>*

*<img src="myimage.jpg" alt="image description" />*

*</div>*

*<script src="anotherscript.js"></script>*

*</body>*

*</html>*

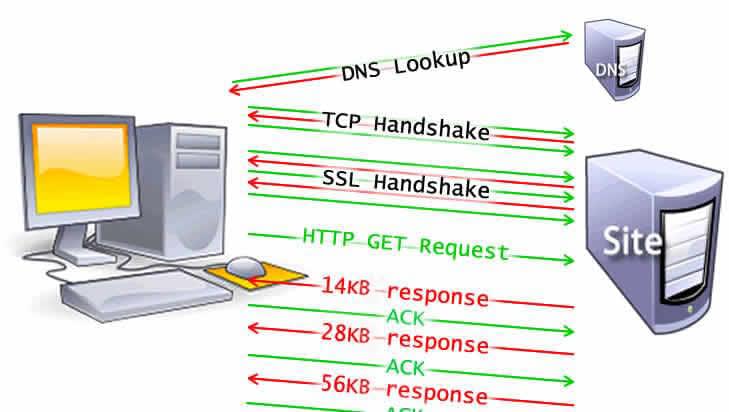
Phản hồi cho yêu cầu ban đầu này chứa byte dữ liệu đầu tiên nhận được. [Time to First Byte](https://developer.mozilla.org/en-US/docs/Glossary/Time_to_first_byte) (TTFB) là khoảng thời gian từ khi người dùng thực hiện yêu cầu—giả sử bằng cách nhấp vào một liên kết—đến khi nhận được gói HTML đầu tiên này. Đoạn nội dung đầu tiên thường là 14KB dữ liệu.

Trong ví dụ của chúng tôi ở trên, yêu cầu chắc chắn nhỏ hơn 14KB, nhưng các tài nguyên được liên kết không được yêu cầu cho đến khi trình duyệt gặp các liên kết trong quá trình phân tích cú pháp, được mô tả bên dưới.

**6. Bắt đầu chậm TCP/ quy tắc 14KB**

Gói phản hồi đầu tiên sẽ là 14KB. Đây là một phần của [khởi động chậm TCP](https://developer.mozilla.org/en-US/docs/Glossary/TCP_slow_start) , một thuật toán cân bằng tốc độ kết nối mạng. Khởi động chậm tăng dần lượng dữ liệu được truyền cho đến khi có thể xác định được băng thông tối đa của mạng.

Trong [khởi động chậm TCP](https://developer.mozilla.org/en-US/docs/Glossary/TCP_slow_start) , sau khi nhận được gói ban đầu, máy chủ sẽ tăng gấp đôi kích thước của gói tiếp theo lên khoảng 28KB. Các gói tiếp theo tăng kích thước cho đến khi đạt đến ngưỡng được xác định trước hoặc xảy ra tắc nghẽn.



Nếu bạn đã từng nghe nói về quy tắc 14KB cho lần tải trang đầu tiên, thì tốc độ khởi động chậm của TCP là lý do tại sao phản hồi ban đầu là 14KB và tại sao tối ưu hóa hiệu suất web lại yêu cầu tập trung tối ưu hóa với phản hồi 14KB ban đầu này. TCP khởi động chậm dần dần xây dựng tốc độ truyền phù hợp với khả năng của mạng để tránh tắc nghẽn.

### [**Điều khiển tắc nghẽn**](https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work#congestion_control)

Khi máy chủ gửi dữ liệu trong các gói TCP, máy khách của người dùng sẽ xác nhận việc gửi bằng cách trả lại xác nhận hoặc ACK. Kết nối có dung lượng hạn chế tùy thuộc vào điều kiện mạng và phần cứng. Nếu máy chủ gửi quá nhiều gói quá nhanh, chúng sẽ bị loại bỏ. Có nghĩa là, sẽ không có sự thừa nhận. Máy chủ đăng ký đây là ACK bị thiếu. Thuật toán kiểm soát tắc nghẽn sử dụng luồng gói tin đã gửi và ACK này để xác định tốc độ gửi.

**7. Phân tích cú pháp**

Khi trình duyệt nhận được đoạn dữ liệu đầu tiên, nó có thể bắt đầu phân tích cú pháp thông tin nhận được. [Phân tích cú pháp](https://developer.mozilla.org/en-US/docs/Glossary/Parse) là bước mà trình duyệt thực hiện để biến dữ liệu mà nó nhận được qua mạng thành [DOM](https://developer.mozilla.org/en-US/docs/Glossary/DOM) và [CSSOM](https://developer.mozilla.org/en-US/docs/Glossary/CSSOM) , được trình kết xuất sử dụng để tô một trang lên màn hình.

DOM là đại diện bên trong của đánh dấu cho trình duyệt. DOM cũng được hiển thị và có thể được thao tác thông qua các API khác nhau trong JavaScript.

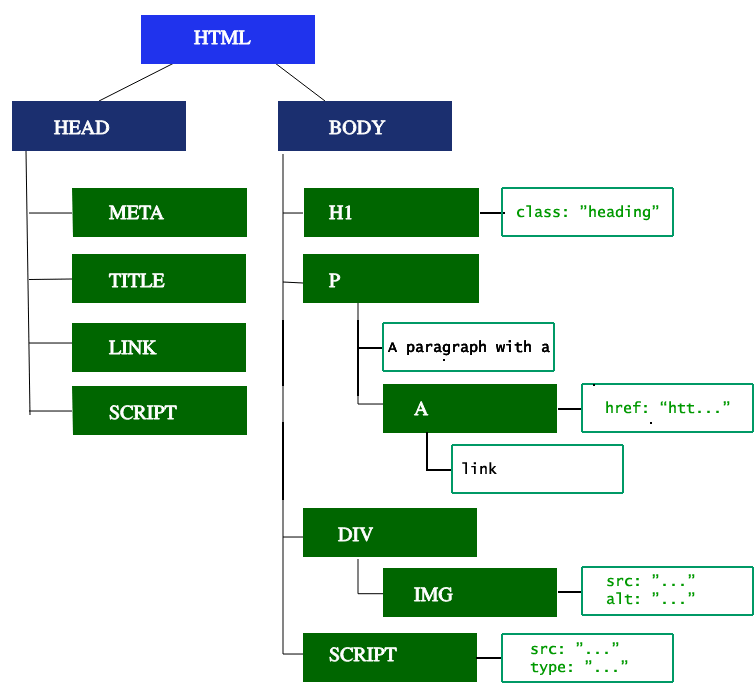
Ngay cả khi HTML của trang được yêu cầu lớn hơn gói 14KB ban đầu, trình duyệt sẽ bắt đầu phân tích cú pháp và cố gắng hiển thị trải nghiệm dựa trên dữ liệu mà nó có. Đây là lý do tại sao điều quan trọng đối với tối ưu hóa hiệu suất web là bao gồm mọi thứ mà trình duyệt cần để bắt đầu hiển thị một trang hoặc ít nhất là một mẫu của trang - CSS và HTML cần thiết cho lần hiển thị đầu tiên - trong 14 kilobyte đầu tiên. Nhưng trước khi mọi thứ được hiển thị trên màn hình, HTML, CSS và JavaScript phải được phân tích cú pháp.

**a. Xây dựng cây DOM**

Chúng tôi mô tả năm bước trong [đường dẫn hiển thị quan trọng](https://developer.mozilla.org/en-US/docs/Web/Performance/Critical_rendering_path) .

Bước đầu tiên là xử lý đánh dấu HTML và xây dựng cây DOM. Phân tích cú pháp HTML liên quan đến [mã thông báo](https://developer.mozilla.org/en-US/docs/Web/API/DOMTokenList) và xây dựng cây. Mã thông báo HTML bao gồm thẻ bắt đầu và thẻ kết thúc, cũng như tên và giá trị thuộc tính. Nếu tài liệu được định dạng tốt, việc phân tích cú pháp sẽ đơn giản và nhanh hơn. Trình phân tích cú pháp phân tích cú pháp đầu vào được mã hóa vào tài liệu, xây dựng cây tài liệu.

Cây DOM mô tả nội dung của tài liệu. Phần tử [<html>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/html) là thẻ đầu tiên và nút gốc của cây tài liệu. Cây phản ánh mối quan hệ và thứ bậc giữa các thẻ khác nhau. Các thẻ được lồng trong các thẻ khác là các nút con. Số nút DOM càng nhiều thì thời gian xây dựng cây DOM càng lâu.



Khi trình phân tích cú pháp tìm thấy các tài nguyên không chặn, chẳng hạn như hình ảnh, trình duyệt sẽ yêu cầu các tài nguyên đó và tiếp tục phân tích cú pháp. Quá trình phân tích cú pháp có thể tiếp tục khi gặp tệp CSS, nhưng các thẻ <script>—đặc biệt là những thẻ không có thuộc tính [async](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function) hoặc defer—sẽ chặn hiển thị và tạm dừng quá trình phân tích cú pháp HTML. Mặc dù trình quét tải trước của trình duyệt đẩy nhanh quá trình này, nhưng các tập lệnh quá mức vẫn có thể là một nút thắt cổ chai đáng kể.

**b. Tải trước máy quét**

Trong khi trình duyệt xây dựng cây DOM, quá trình này chiếm chủ đề chính. Khi điều này xảy ra, trình quét tải trước sẽ phân tích nội dung có sẵn và yêu cầu các tài nguyên có mức độ ưu tiên cao như CSS, JavaScript và phông chữ web. Nhờ trình quét tải trước, chúng tôi không phải đợi cho đến khi trình phân tích cú pháp tìm thấy tham chiếu đến tài nguyên bên ngoài để yêu cầu nó. Nó sẽ truy xuất tài nguyên ở chế độ nền để vào thời điểm trình phân tích cú pháp HTML chính tiếp cận nội dung được yêu cầu, chúng có thể đã sẵn sàng hoặc đã được tải xuống. Các tối ưu hóa mà máy quét tải trước cung cấp giúp giảm tắc nghẽn.

*<link rel="stylesheet" href="styles.css" />*

*<script src="myscript.js" async></script>*

*<img src="myimage.jpg" alt="image description" />*

*<script src="anotherscript.js" async></script>*

Trong ví dụ này, trong khi luồng chính đang phân tích cú pháp HTML và CSS, trình quét tải trước sẽ tìm các tập lệnh và hình ảnh, đồng thời cũng bắt đầu tải chúng xuống. Để đảm bảo tập lệnh không chặn quá trình, hãy thêm thuộc tính async hoặc thuộc tính defer nếu thứ tự thực thi và phân tích cú pháp JavaScript là quan trọng.

Việc chờ đợi để lấy CSS không chặn quá trình phân tích cú pháp hoặc tải xuống HTML, nhưng nó chặn JavaScript, vì JavaScript thường được sử dụng để truy vấn tác động của các thuộc tính CSS đối với các phần tử.

**c. Xây dựng CSSOM**

Bước thứ hai trong lộ trình kết xuất quan trọng là xử lý CSS và xây dựng cây CSSOM. Mô hình đối tượng CSS tương tự như DOM. DOM và CSSOM đều là cây. Chúng là những cấu trúc dữ liệu độc lập. Trình duyệt chuyển đổi các quy tắc CSS thành bản đồ các kiểu mà trình duyệt có thể hiểu và làm việc cùng. Trình duyệt duyệt qua từng bộ quy tắc trong CSS, tạo ra một cây gồm các nút có mối quan hệ cha, con và anh chị em dựa trên bộ chọn CSS.

Cũng như HTML, trình duyệt cần chuyển đổi các quy tắc CSS nhận được thành thứ mà nó có thể hoạt động được. Do đó, nó lặp lại quy trình chuyển HTML thành đối tượng, nhưng đối với CSS.

Cây CSSOM bao gồm các kiểu từ biểu định kiểu tác nhân người dùng. Trình duyệt bắt đầu với quy tắc chung nhất áp dụng cho một nút và tinh chỉnh đệ quy các kiểu được tính toán bằng cách áp dụng các quy tắc cụ thể hơn. Nói cách khác, nó xếp tầng các giá trị thuộc tính.

Xây dựng CSSOM rất, rất nhanh và không được hiển thị bằng một màu duy nhất trong các công cụ dành cho nhà phát triển hiện tại. Thay vào đó, "Kiểu tính toán lại" trong các công cụ dành cho nhà phát triển hiển thị tổng thời gian cần thiết để phân tích cú pháp CSS, xây dựng cây CSSOM và tính toán đệ quy các kiểu được tính toán. Về mặt tối ưu hóa hiệu suất web, có ít rủi ro hơn vì tổng thời gian để tạo CSSOM thường ít hơn thời gian cần thiết cho một lần tra cứu DNS.

**8. Quy trình khác**

**a. Biên dịch JavaScript**

Trong khi CSS đang được phân tích cú pháp và CSSOM được tạo, các nội dung khác, bao gồm cả tệp JavaScript, đang tải xuống (nhờ trình quét tải trước). JavaScript được giải thích, biên dịch, phân tích cú pháp và thực thi. Các tập lệnh được phân tích thành các cây cú pháp trừu tượng. Một số công cụ trình duyệt lấy [Cây cú pháp trừu tượng](https://en.wikipedia.org/wiki/Abstract_Syntax_Tree) và chuyển nó vào trình thông dịch, xuất ra mã byte được thực thi trên luồng chính. Điều này được gọi là biên dịch JavaScript.

**b. Xây dựng cây trợ năng**

Trình duyệt cũng xây dựng cây [khả năng truy cập](https://developer.mozilla.org/en-US/docs/Learn/Accessibility) mà các thiết bị trợ giúp sử dụng để phân tích cú pháp và diễn giải nội dung. Mô hình đối tượng trợ năng (AOM) giống như một phiên bản ngữ nghĩa của DOM. Trình duyệt cập nhật cây khả năng truy cập khi DOM được cập nhật. Cây trợ năng không thể sửa đổi được bằng chính các công nghệ hỗ trợ.

[Trình đọc màn hình](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Screen_Reader_Implementors_Guide) không thể truy cập nội dung cho đến khi AOM được tạo .

**9. Render**

Các bước Render bao gồm kiểu dáng, bố cục, màu vẽ và trong một số trường hợp là tổng hợp. Cây CSSOM và DOM được tạo trong bước phân tích cú pháp được kết hợp thành cây kết xuất, cây này sau đó được sử dụng để tính toán bố cục của mọi thành phần hiển thị, sau đó được vẽ lên màn hình. Trong một số trường hợp, nội dung có thể được thăng cấp lên các lớp riêng của chúng và được tổng hợp, cải thiện hiệu suất bằng cách vẽ các phần của màn hình trên GPU thay vì CPU, giải phóng luồng chính.

**a. Style**

Bước thứ ba trong lộ trình kết xuất quan trọng là kết hợp DOM và CSSOM thành một cây kết xuất. Cây kiểu được tính toán hoặc cây kết xuất, quá trình xây dựng bắt đầu bằng gốc của cây DOM, duyệt qua từng nút hiển thị.

Các thẻ sẽ không được hiển thị, chẳng hạn như nút [<head>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/head)và nút con của nó cũng như bất kỳ nút nào có display: none, chẳng hạn như nút script { display: none; }bạn sẽ tìm thấy trong biểu định kiểu tác nhân người dùng, sẽ không được đưa vào cây kết xuất vì chúng sẽ không xuất hiện trong kết quả được kết xuất. Các nút được visibility: hiddenáp dụng được bao gồm trong cây kết xuất vì chúng chiếm dung lượng. Vì chúng tôi chưa đưa ra bất kỳ chỉ thị nào để ghi đè mặc định tác nhân người dùng, nên nút scripttrong ví dụ mã của chúng tôi ở trên sẽ không được đưa vào cây kết xuất.

Mỗi nút hiển thị có các quy tắc CSSOM được áp dụng cho nó. Cây kết xuất chứa tất cả các nút hiển thị có nội dung và kiểu được tính toán — đối sánh tất cả các kiểu có liên quan với mọi nút hiển thị trong cây DOM và xác định, dựa trên tầng CSS , kiểu được tính toán cho mỗi [nút](https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade) .

**b. Layout**

Bước thứ tư trong lộ trình kết xuất quan trọng là chạy bố cục trên cây kết xuất để tính toán hình dạng của từng nút. Bố cục là quá trình xác định chiều rộng, chiều cao và vị trí của tất cả các nút trong cây kết xuất, cộng với việc xác định kích thước và vị trí của từng đối tượng trên trang. Chỉnh lại dòng là bất kỳ xác định kích thước và vị trí tiếp theo của bất kỳ phần nào của trang hoặc toàn bộ tài liệu.

Khi cây kết xuất được tạo, bố cục bắt đầu. Cây kết xuất đã xác định các nút nào được hiển thị (ngay cả khi ẩn) cùng với các kiểu được tính toán của chúng, chứ không phải kích thước hoặc vị trí của từng nút. Để xác định kích thước và vị trí chính xác của từng đối tượng, trình duyệt bắt đầu từ gốc của cây kết xuất và duyệt qua nó.

Trên trang web, hầu hết mọi thứ đều là một hộp. Các thiết bị khác nhau và các tùy chọn máy tính để bàn khác nhau có nghĩa là không giới hạn các kích thước khung nhìn khác nhau. Trong giai đoạn này, khi xem xét kích thước khung nhìn, trình duyệt sẽ xác định kích thước của tất cả các hộp khác nhau sẽ có trên màn hình. Lấy kích thước của khung nhìn làm cơ sở, bố cục thường bắt đầu với phần thân, sắp xếp kích thước của tất cả các phần tử con của phần thân, với các thuộc tính mô hình hộp của mỗi phần tử, cung cấp không gian giữ chỗ cho các phần tử được thay thế mà nó không biết kích thước của nó, chẳng hạn như hình ảnh của chúng tôi.

Lần đầu tiên kích thước và vị trí của các nút được xác định được gọi là bố cục . Các tính toán lại sau đó về kích thước nút và vị trí được gọi là chỉnh lại dòng . Trong ví dụ của chúng tôi, giả sử bố cục ban đầu xảy ra trước khi hình ảnh được trả về. Vì chúng tôi đã không khai báo kích thước hình ảnh của mình nên sẽ có một dòng chỉnh lại khi kích thước hình ảnh được biết.

**c. Paint**

Bước cuối cùng trong đường dẫn hiển thị quan trọng là tô các nút riêng lẻ lên màn hình, lần xuất hiện đầu tiên của nó được gọi là lần [sơn có ý nghĩa đầu tiên](https://developer.mozilla.org/en-US/docs/Glossary/First_meaningful_paint) . Trong giai đoạn vẽ hoặc rasterization, trình duyệt chuyển đổi từng hộp được tính toán trong giai đoạn bố cục thành các pixel thực tế trên màn hình. Vẽ tranh liên quan đến việc vẽ mọi phần trực quan của một phần tử lên màn hình, bao gồm văn bản, màu sắc, đường viền, bóng đổ và các phần tử được thay thế như nút và hình ảnh. Trình duyệt cần thực hiện việc này siêu nhanh.

Để đảm bảo hoạt ảnh và cuộn mượt mà, mọi thứ chiếm luồng chính, bao gồm các kiểu tính toán, cùng với chỉnh lại dòng và vẽ, phải mất ít hơn 16,67 mili giây để trình duyệt hoàn thành. Ở 2048 X 1536, iPad có hơn 3.145.000 điểm ảnh được tô lên màn hình. Đó là rất nhiều pixel phải được vẽ rất nhanh. Để đảm bảo việc sơn lại có thể được thực hiện nhanh hơn so với lần sơn ban đầu, bản vẽ trên màn hình thường được chia thành nhiều lớp. Nếu điều này xảy ra, thì việc tổng hợp là cần thiết.

Tranh có thể chia các phần tử trong cây bố cục thành các lớp. Thúc đẩy nội dung thành các lớp trên GPU (thay vì luồng chính trên CPU) giúp cải thiện hiệu suất sơn và sơn lại. Có các thuộc tính và phần tử cụ thể khởi tạo một lớp, bao gồm [<video>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video)và [<canvas>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas)và bất kỳ phần tử nào có thuộc tính CSS của [opacity](https://developer.mozilla.org/en-US/docs/Web/CSS/opacity), 3D [transform](https://developer.mozilla.org/en-US/docs/Web/CSS/transform), [will-change](https://developer.mozilla.org/en-US/docs/Web/CSS/will-change)và một số phần tử khác. Các nút này sẽ được vẽ trên lớp riêng của chúng, cùng với các nút con của chúng, trừ khi một nút con cần phải có lớp riêng của nó vì một (hoặc nhiều) lý do trên.

Các lớp cải thiện hiệu suất, nhưng tốn kém khi nói đến quản lý bộ nhớ, vì vậy không nên lạm dụng như một phần của chiến lược tối ưu hóa hiệu suất web.

**d. Tổng hợp**

Khi các phần của tài liệu được vẽ trong các lớp khác nhau, chồng chéo lên nhau, việc tổng hợp là cần thiết để đảm bảo chúng được vẽ trên màn hình theo đúng thứ tự và nội dung được hiển thị chính xác.

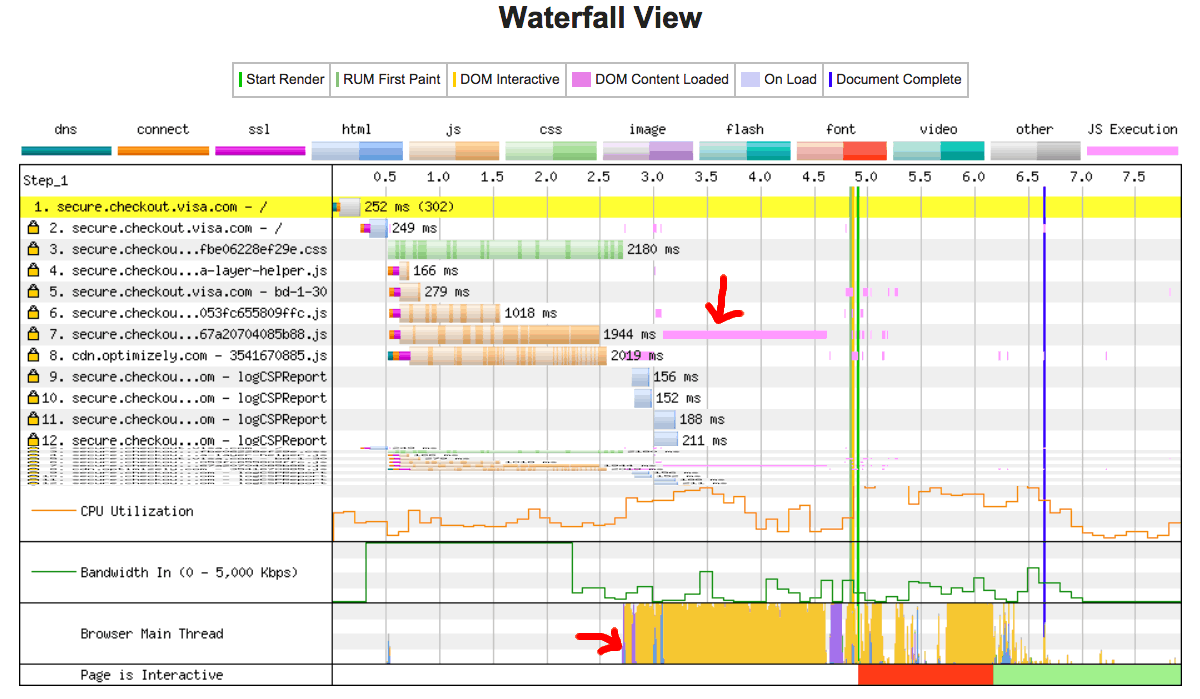
Khi trang tiếp tục tải nội dung, có thể xảy ra hiện tượng chỉnh lại dòng (hãy nhớ lại hình ảnh ví dụ của chúng tôi đã đến muộn). Một dòng chảy lại gây ra sơn lại và tổng hợp lại. Nếu chúng tôi đã xác định kích thước của hình ảnh của mình, thì sẽ không cần chỉnh lại dòng và chỉ lớp cần sơn lại sẽ được sơn lại và được tổng hợp nếu cần. Nhưng chúng tôi không bao gồm kích thước hình ảnh! Khi hình ảnh được lấy từ máy chủ, quá trình hiển thị sẽ quay lại các bước bố cục và bắt đầu lại từ đó.

**10. Tương tác**

Sau khi chủ đề chính hoàn thành việc vẽ trang, bạn sẽ nghĩ rằng chúng tôi sẽ "hoàn tất". Đó không nhất thiết phải là trường hợp. Nếu tải bao gồm JavaScript, được trì hoãn chính xác và chỉ được thực thi sau khi sự [onload](https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event)kiện kích hoạt, thì luồng chính có thể đang bận và không khả dụng để cuộn, chạm và các tương tác khác.

[Thời gian để tương tác](https://developer.mozilla.org/en-US/docs/Glossary/Time_to_interactive) (TTI) là phép đo khoảng thời gian từ yêu cầu đầu tiên dẫn đến tra cứu DNS và kết nối SSL đến khi trang tương tác — tương tác là thời điểm sau Sơn nội dung đầu tiên khi trang [phản](https://developer.mozilla.org/en-US/docs/Glossary/First_contentful_paint) hồi tương tác của người dùng trong vòng 50 mili giây. Nếu luồng chính đang bận phân tích cú pháp, biên dịch và thực thi JavaScript, luồng đó sẽ không khả dụng và do đó không thể phản hồi các tương tác của người dùng một cách kịp thời (dưới 50 mili giây).

Trong ví dụ của chúng tôi, có thể hình ảnh được tải nhanh nhưng có lẽ tệp anotherscript.jscó dung lượng 2 MB và kết nối mạng của người dùng của chúng tôi chậm. Trong trường hợp này, người dùng sẽ thấy trang siêu nhanh, nhưng sẽ không thể cuộn mà không giật cho đến khi tập lệnh được tải xuống, phân tích cú pháp và thực thi. Đó không phải là một trải nghiệm người dùng tốt. Tránh chiếm chủ đề chính, như được minh họa trong ví dụ WebPageTest này:



Trong ví dụ này, quá trình thực thi JavaScript mất hơn 1,5 giây và chuỗi chính đã được sử dụng hoàn toàn trong toàn bộ thời gian đó, không phản hồi với các sự kiện nhấp chuột hoặc chạm vào màn hình.

**Tài liệu tham khảo:**

<https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work>