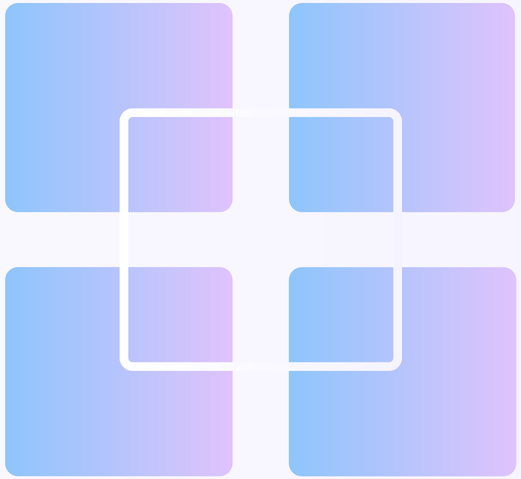
Website optimization

How to improve the performance of a website

https://rikkeisoft.com

@tult

Table of contents



Why speed matters?

Website load time affects the number of visitors.

Why speed matters?

👉 The longger a webpage takes to load, the more it bounce rate will skyrocket 🚀

👉 The high bounce rate tells search engines that this page is useless, so its ranking will slip ↘

DID YOU KNOW?

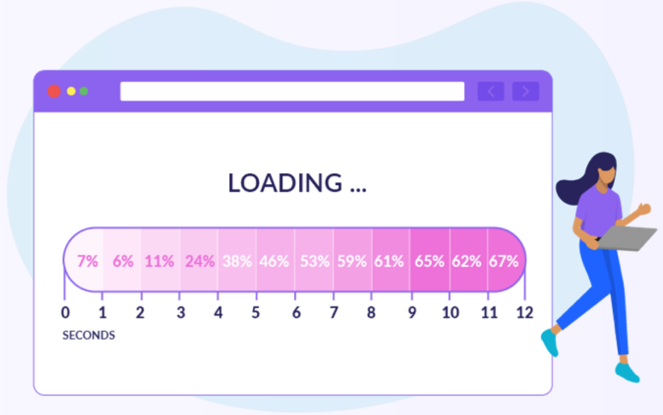
1 IN 4 VISITORS
would abandon a website that takes more than 4 seconds to load

46% OF USERS
don't revisit poorly performing websites

64% OF SHOPPERS
who are dissatisfied with their site visit will shop somewhere else next time

1 SECOND DELAY
reduces customer satisfaction by 16%

BOUNCE RATE



https://rikkeisoft.com

@tult

What factor affect website load time?

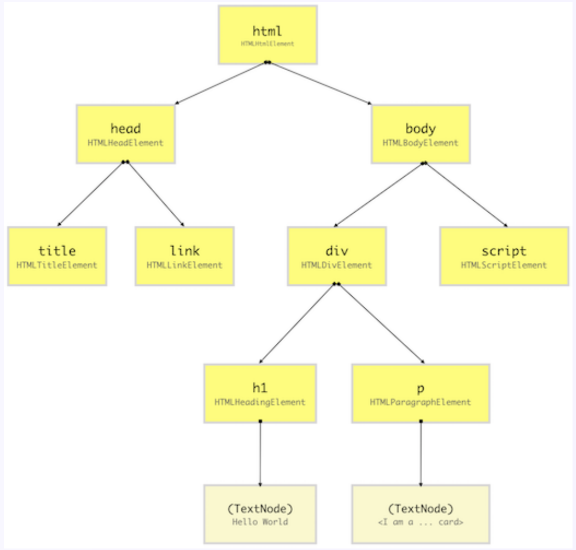
- User’s internet connection
- Web hoisting and user’s computer
- The size of the resources that needed

How the browser render a webpage

Deep dive into the rendering process and figure out where can be optimized.

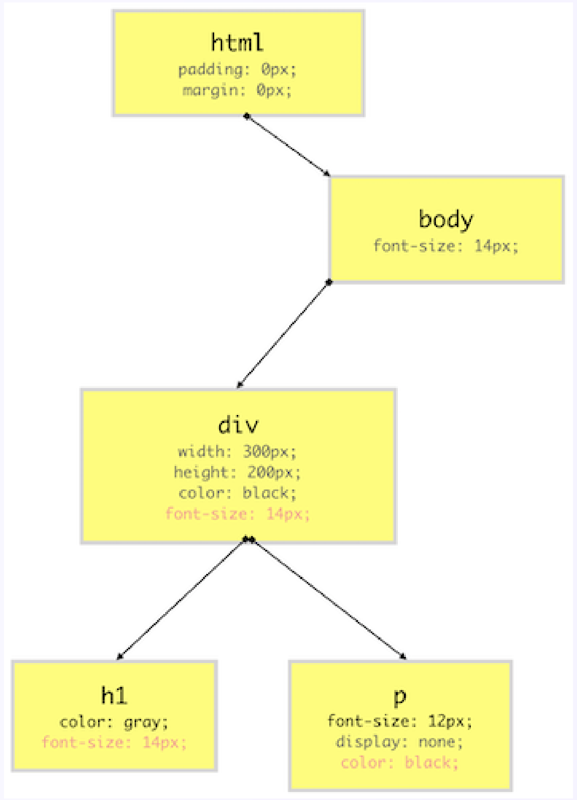
DOM

- DOM stands for document object model
- When browser encounters a HTML element, it creates a JavaScript object called a node.
- After create a node, the browser has to create a **tree-like structure** of created nodes.



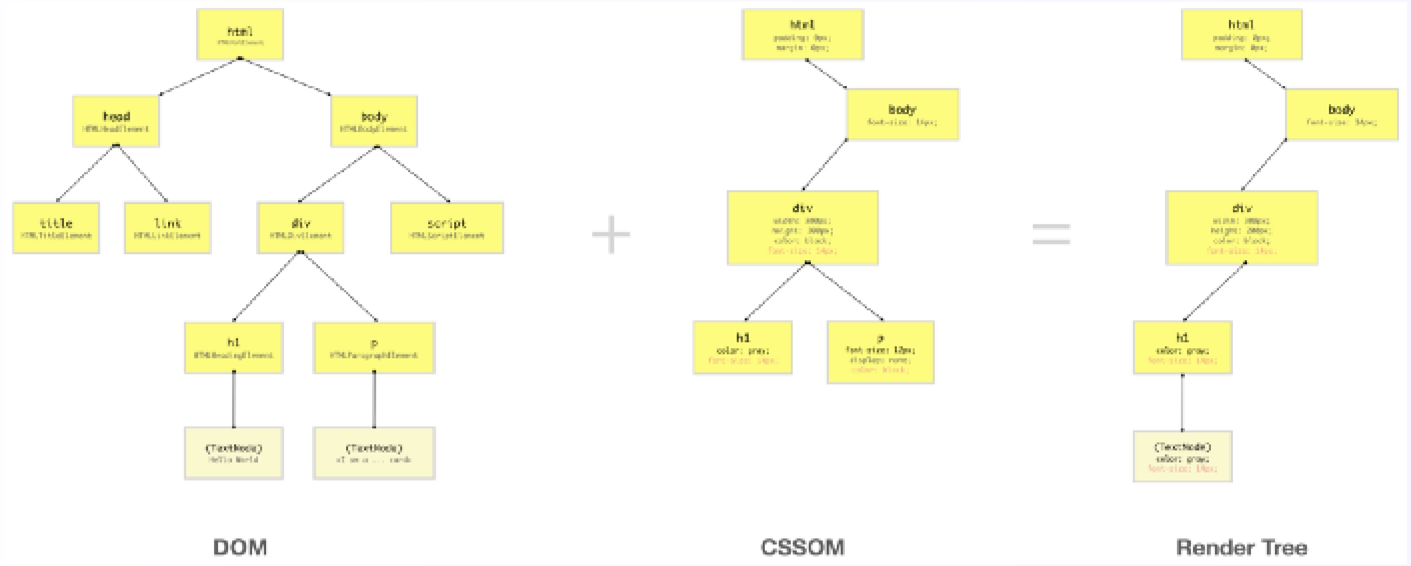
CSSOM

- CSSOM stands for CSS object model
- After the browser has done constructing the DOM, it'll read CSS from all the sources (external, embedded, inline, user-agent, etc.) to construct CSSOM.
- Each node in CSSOM tree contains the style information that will be applied to DOM elements that it target.



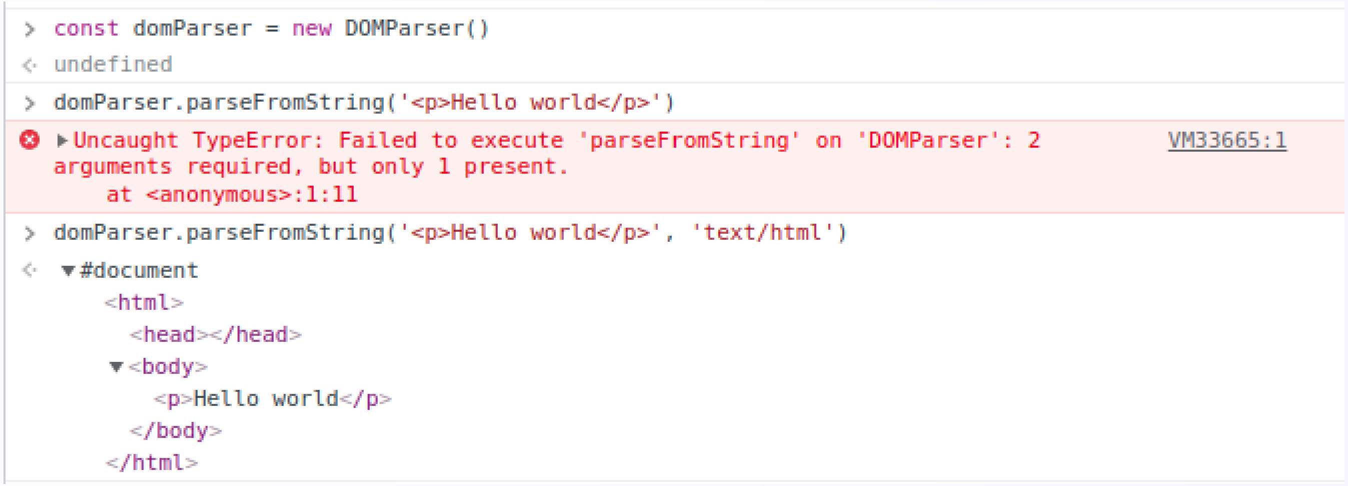
Render tree

- This is **tree-like structure** constructed by combining DOM and CSSOM trees together.
- The browser calculate the layout for each **visible elements** and paint them on the screen.



Parsing

- Parsing the the process of reading HTML and constructing the DOM tree from it.
- The browser starts the parsing process as soon as it recevices few bytes of HTML document.
- Because of that, the browser can build the DOM tree incrementally.



```
> const domParser = new DOMParser()
<· undefined
> domParser.parseFromString('<p>Hello world</p>')
✖ ▶Uncaught TypeError: Failed to execute 'parseFromString' on 'DOMParser': 2          VM33665:1
  arguments required, but only 1 present.
      at <anonymous>:1:11
> domParser.parseFromString('<p>Hello world</p>', 'text/html')
<·  ▼#document
      <html>
        <head></head>
       ▼<body>
          <p>Hello world</p>
        </body>
      </html>
```

Parse raw HTML codes into a DOM tree

External resources & parser-blocking script

- Whenever the browser encounters a external resouces, it'll start download that file in background **except** for script files. Hence script files are called **parser-blocking**.
- DOM parsing is executed on the main thread and will not progress if that thread is busy.

A script (JavaScript)

Embedded scripts → Executing the embedded codes on the main thread.

External script file → Halt the execution of the main thread until that file is downloaded and executed

❓ Halting the DOM parsing while the script file is being downloaded is unnecessary (in most cases). What is the solution ?

Async & defer attributes

- HTML5 provides us `async` and `defer` attribute for `script` tag.
- With `async`, the parsing process won't be blocked while the file is being downloaded. And will be block right after the script file is ready to be executed.
- With `defer`, the script doesn't execute even when the file is fully downloaded. All `defer` scripts are executed once the DOM is fully constructed.

Render-Blocking CSS

- Render tree is getting built incrementally as the DOM tree is getting constructed.
- The browser constructs the CSSOM tree from the stylesheet content
- The CSSOM tree construction is **not incremental**

Script-Blocking CSS

❓ Scenario where the browser start downloading the stylesheet file, then it encounter an external script file and start downloading it. The script file is downloaded before the stylesheet file ? In this case, should the browser start executing the script?

⚓ In conclusion, the browser may fully downloaded the script but will not execute it unless all the stylesheets before it are parsed. Those stylesheets are called script-blocking.

General rules

1. Injecting stylesheet or required script files in the `` `<head>` `` tag of the HTML document.
2. Use `` `rel="preload"` `` to instruct the browser to download key resources as soon as possible.
3. The best place to inject script files is the end of `` `<body>` `` tag.