

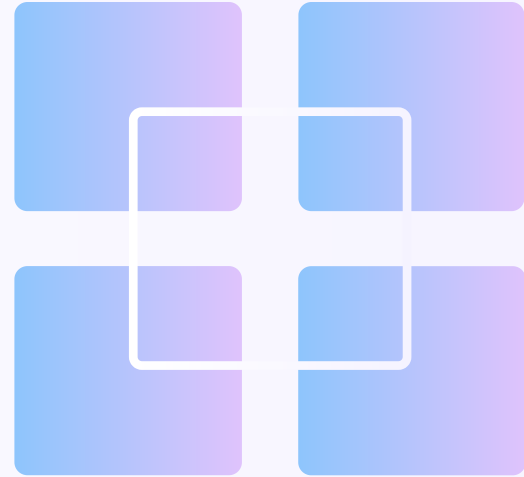


Website optimization

How to improve the performance of a website

Table of contents

1. Why speed matters?
2. What factors affect website load time?
3. How the browser renders a webpage?



Why speed matters?

- ➡ The longer a webpage takes to load, the more its 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

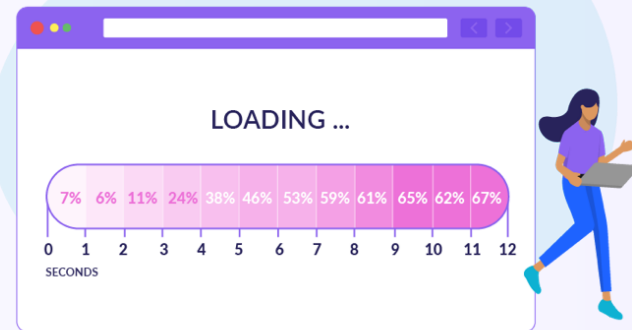
64% OF SHOPPERS

who are dissatisfied with their site visit will shop somewhere else next time

1 SECOND DELAY

reduces customer satisfaction by 16%

BOUNCE RATE



What factor affect website load time?

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

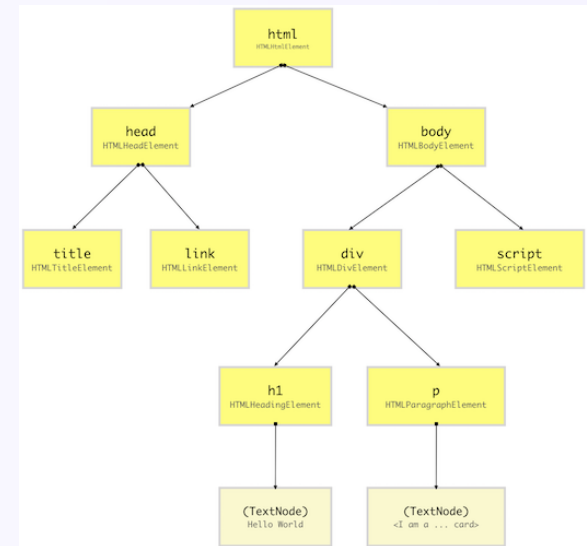


How the browser render a webpage

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

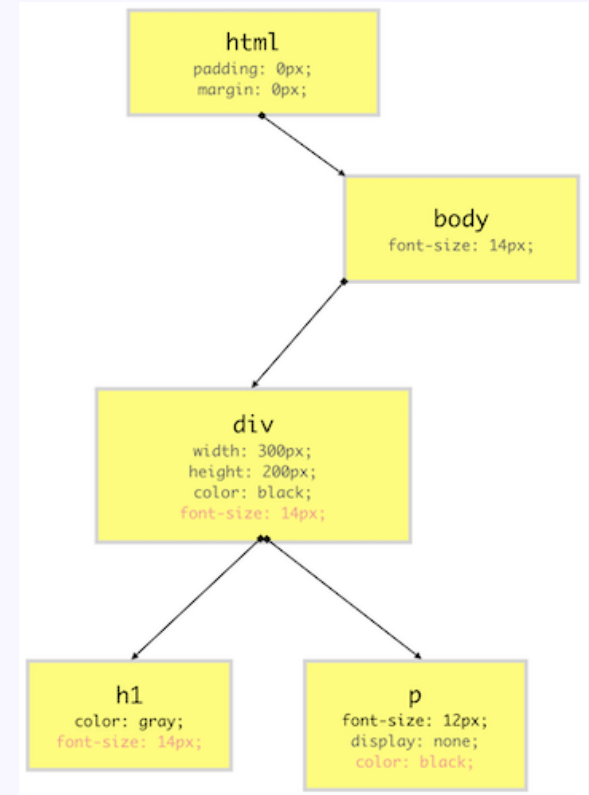
DOM

- 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.



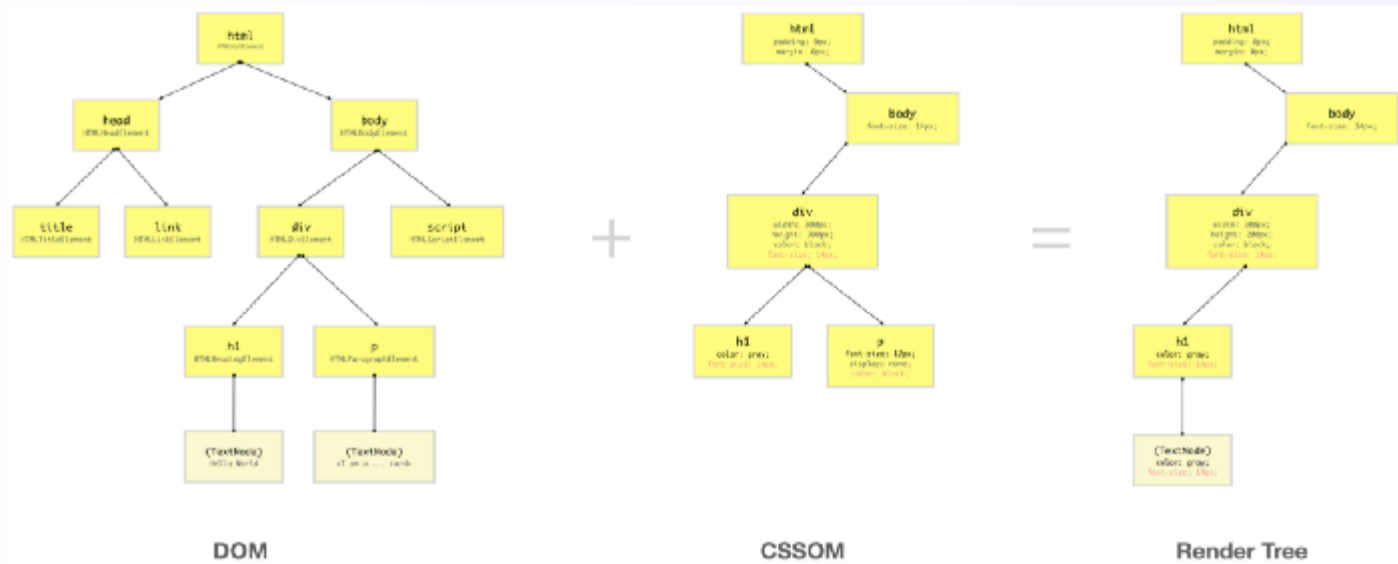
CSSOM

- 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
  arguments required, but only 1 present.
  at <anonymous>:1:11
  VM33665:1
> 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 resources, 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.

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

🔗 A script (JavaScript)

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.