
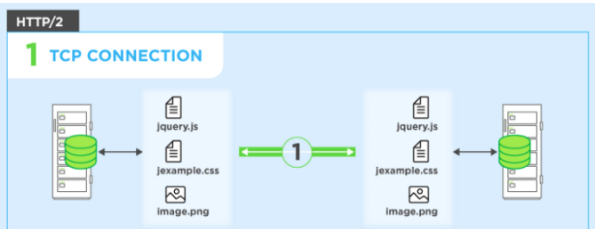


# JS-FW TASK-1

## 1) HTTP/1.1 vs HTTP/2

HTTP/1.1	HTTP/2
<p>Developed by Timothy Berners-Lee in 1989 as a communication standard for the World Wide Web, HTTP is a top-level application protocol that exchanges information between a client computer and a local or remote web server. In this process, a client sends a text-based request to a server by calling a <i>method</i> like GET or POST. In response, the server sends a resource like an HTML page back to the client.</p>	<p>HTTP/2 began as the SPDY protocol, developed primarily at Google with the intention of reducing web page load latency by using techniques such as compression, multiplexing, and prioritization. This protocol served as a template for HTTP/2 when the Hypertext Transfer Protocol working group httpbis of the IETF (Internet Engineering Task Force) put the standard together, culminating in the publication of HTTP/2 in May 2015.</p>
 <p>HTTP/1.1</p> <p>3 TCP CONNECTIONS</p>	 <p>HTTP/2</p> <p>1 TCP CONNECTION</p>
<p>HTTP/1.1 share semantics, ensuring that the requests and responses traveling between the server and client in both protocols reach their destinations as traditionally formatted messages with headers and bodies, using familiar methods like GET or POST</p>	<p>HTTP/2 uses the same methods like GET or POST for requests and responses traveling between the server and client. But while HTTP/1.1 transfers these in plain-text messages, HTTP/2 encodes these into binary, allowing for significantly different delivery model possibilities.</p>
<p>HTTP/1.1 can send only one request at a time over a single TCP connection.</p>	<p>HTTP/2 can send multiple requests for data in parallel over a single TCP connection. This is the most advanced feature of the HTTP/2 protocol because it allows you to download web files asynchronously from one server.</p>
<p>HTTP/1.1 loads resources one after the</p>	<p>In contrast, HTTP/2 is able to use a single</p>

other, so if one resource cannot be loaded, it blocks all the other resources behind it.	TCP connection to send multiple streams of data at once so that no one resource blocks any other resource. HTTP/2 does this by splitting data into binary-code messages and numbering these messages so that the client knows which stream each binary message belongs to.
Typically, a server only serves content to a client device if the client asks for it. However, this approach is not always practical for modern webpages, which often involve several dozen separate resources that the client must request.	HTTP/2 solves this problem by allowing a server to "push" content to a client before the client asks for it. The server also sends a message letting the client know what pushed content to expect – like if Bob had sent Alice a Table of Contents of his novel before sending the whole thing.
Small files load more quickly than large ones. To speed up web performance, both HTTP/1.1 compress HTTP messages to make them smaller.	HTTP/2 does the same thing but with a more advanced compression method called HPACK that eliminates redundant information in HTTP header packets.

## 2) HTTP Version History

### a) HTTP/0.9 - The One Line Protocol

- i) The initial version of HTTP had no version number; it has been later called 0.9 to differentiate it from the later versions. HTTP/0.9 is extremely simple: requests consist of a single line and start with the only possible method GET followed by the path to the resource.

### b) HTTP/1.0 - Building Extensibility

- i) Versioning information is now sent within each request (HTTP /1.0 is appended to the GET line)
- ii) A status code line is also sent at the beginning of the response, allowing the browser itself to understand the success or failure of the request and to adapt its behavior in consequence.

- iii) The notion of HTTP headers has been introduced, both for the requests and the responses, allowing metadata to be transmitted and making the protocol extremely flexible and extensible.
- iv) With the help of the new HTTP headers, the ability to transmit other documents than plain HTML files has been added.

#### c) HTTP/1.1 - The Standardized protocol

- i) A connection can be reused, saving time to reopen it numerous times to display the resources embedded into the single original document retrieved.
- ii) Pipelining has been added, allowing to send a second request before the answer for the first one is fully transmitted, lowering the latency of the communication.
- iii) Chunked responses are now also supported.
- iv) Additional cache control mechanisms have been introduced.
- v) Content negotiation, including language, encoding, or type, has been introduced and allows a client and a server to agree on the most adequate content to exchange.
- vi) Thanks to the Host header, the ability to host different domains at the same IP address now allows server colocation.

#### d) HTTP/2 - A Protocol for Greater Performance

- i) Over the years, Web pages have become much more complex, even becoming applications in their own right.
- ii) The amount of visual media displayed, the volume and size of scripts adding interactivity, has also increased: much more data is transmitted over significantly more HTTP requests. HTTP/1.1 connections need requests sent in the correct order.
- iii) It is a binary protocol rather than a text. It can no longer be read and created manually. Despite this hurdle, improved optimization techniques can now be implemented.

- iv) It is a multiplexed protocol. Parallel requests can be handled over the same connection, removing the order and blocking constraints of the HTTP/1.x protocol.
- v) It compresses headers. As these are often similar among a set of requests, this removes duplication and overhead of data transmitted.
- vi) It allows a server to populate data in a client cache, in advance of it being required, through a mechanism called the server push.
- vii) By healing flaws, yet retaining the flexibility and extensibility which made HTTP such a success, the adoption of HTTP/2 hints at a bright future for the protocol.

### **3) 5 difference between Browser JS(console) & Nodejs History**

In the browser, most of the time what we are doing is interacting with the DOM or other Web Platform APIs like Cookies. Those do not exist in Node, of course. You don't have the document, window, and all the other objects that are provided by the browser.

And in the browser, we don't have all the nice APIs that Node.js provides through its modules, like the filesystem access functionality.

Another big difference is that in Node.js we control the environment. Unless you are building an open-source application that anyone can deploy anywhere, you know which version of Node you will run the application on. Compared to the browser environment, where you don't get the luxury to choose what browser your visitors will use, this is very convenient.

This means that you can write all the modern ES6-7-8-9 JavaScript that your Node version supports.

Since JavaScript moves so fast, but browsers can be a bit slow and users a bit slow to upgrade, sometimes on the web, you are stuck to use older JavaScript / ECMAScript releases.

You can use Babel to transform your code to be ES5-compatible before shipping it to the browser, but in Node, you won't need that.

Another difference is that Node uses the CommonJS module system, while in the browser we are starting to see the ES Modules standard being implemented.

In practice, this means that for the time being you use `require()` in Node and `import` in the browser.

#### 4) Abstract Working of JS engine(V8)

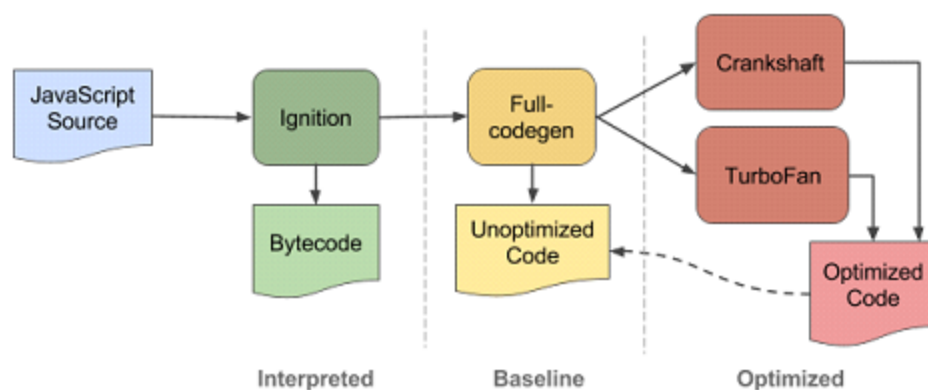


Fig. - Abstract working of js engine (v8)

V8 gets its speed from just-in-time (JIT) compilation of JavaScript to native machine code, just before executing it. First, the code is compiled by a baseline compiler, which quickly generates a non-optimized machine code. On runtime, the compiled code is

analyzed and can be re-compiled for optimal performance. Ignition provides the first while Turbofans & Crankshaft the second.

JIT compilation result machine code can take a large amount of memory, while it might be executed once. This is solved by the Ignition, which is executing code with less memory overhead.

The Turbofan project started in 2013 to improve the weakness of Crankshaft which isn't optimized for some part of the JavaScript functionality e.g. error handling. It was designed for optimizing both existing and future planned features at the time.

## **5) What happens when you type a URL in the address bar in the browser?**

- a) The browser checks the cache for DNS entry to find the corresponding IP Address of website. It looks for the following cache. If not found in one, then continues checking to the next until found.
  - i) Browser Cache
  - ii) Operating Systems Cache
  - iii) Router Cache
  - iv) ISP Cache
- b) If not found in the cache, ISP's (Internet Service Provider) DNS server initiates a DNS query to find the IP address of the server that hosts the domain name.

The requests are sent using small data packets that contain information content of the request and the IP address it is destined for.
- c) The browser initiates a TCP (Transfer Control Protocol) connection with the server using synchronize(SYN) and acknowledge(ACK) messages.
- d) The browser sends an HTTP request to the webserver. GET or POST request. The server on the host computer handles that request and sends back a response. It assembles a response in some formats like JSON, XML, and HTML.

- e) The server sends out an HTTP response along with the status of the response.
- f) The browser displays HTML content on the user window.