Difference between HTTP1.1 vs HTTP2

HTTP1.1	**HTTP2**
HTTP/1.1, keeps all requests and responses in plain text format	HTTP/2 uses the binary framing layer to encapsulate all messages in binary format
HTTP/1.1, which must make use of multiple TCP connections to lessen the effect of HOL blocking	HTTP/2 establishes a single connection object between the two machines. Within this connection there are multiple streams of data. Each stream consists of multiple messages in the familiar request/response format
HTTP/1.1 relies on the transport layer to avoid buffer overflow, each new TCP connection requires a separate flow control mechanism	HTTP/2 allow the client and server to implement their own flow controls, rather than relying on the transport layer
In HTTP/1.1, if the developer knows in advance which additional resources the client machine will need to render the page, they can use a technique called resource inlining	Since HTTP/2 enables multiple concurrent responses to a client's initial GET request, a server can send a resource to a client along with the requested HTML page, providing the resource before the client asks for it. This process is called server push.
Programs like gzip have long been used to compress the data sent in HTTP messages, especially to decrease the size of CSS and JavaScript files.Without compressing the header which can lead to heavier weight after making many request	In order to solve this bottleneck, HTTP/2 uses HPACK compression to shrink the size of headers.This algorithm can encode the header metadata using Huffman coding.

https://http2.golang.org/gophertiles Google demo that compares the protocols for different latenciesGoogle demo that compares the protocols for different latencies

HTTP version history

HTTP (HyperText Transfer Protocol) is the underlying protocol of the World Wide Web. Developed by Tim Berners-Lee and his team between 1989-1991, HTTP has seen many changes, keeping most of the simplicity and further shaping its flexibility. HTTP has evolved from an early protocol to exchange files in a semi-trusted laboratory environment, to the modern maze of the Internet, now carrying images, videos in high resolution and 3D.

1. **HTTP/0.9 – The one-line protocol**

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 (not the URL as both the protocol, server, and port are unnecessary once connected to the server).

Request Example: GET /mypage.html

The response is extremely simple too: it only consisted of the file itself.

Response Example:

```
<HTML>
A very simple HTML page
</HTML>
```

2. **HTTP/1.0 – Building extensibility**

HTTP/0.9 was very limited and both browsers and servers quickly extended it to be more versatile:

Versioning information is now sent within each request (HTTP/1.0 is appended to the GET line)

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 (like in updating or using its local cache in a specific way)

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.

With the help of the new HTTP headers, the ability to transmit other documents than plain HTML files has been added (thanks to the Content-Type header).

At this point, a typical request and response looked like this:

```
GET /mypage.html HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

200 OK
Date: Tue, 15 Nov 1994 08:12:31 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/html
<HTML>
A page with an image
```

```
  <IMG SRC="/myimage.gif">
</HTML>
```

Followed by a second connection and request to fetch the image (followed by a response to that request):

```
GET /myimage.gif HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

200 OK
Date: Tue, 15 Nov 1994 08:12:32 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/gif
(image content)
```

3. **HTTP/1.1 – The standardized protocol**

The first standardized version of HTTP, HTTP/1.1 was published in early 1997, only a few months after HTTP/1.0.

1. HTTP/1.1 clarified ambiguities and introduced numerous improvements:
2. A connection can be reused, saving the time to reopen it numerous times to display the resources embedded into the single original document retrieved.
3. 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.
4. Chunked responses are now also supported.
5. Additional cache control mechanisms have been introduced.
6. 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.
7. Thanks to the `Host` header, the ability to host different domains at the same IP address now allows server colocation.

4. **HTTP/2 – A protocol for greater performance**

In the first half of the 2010s, Google demonstrated an alternative way of exchanging data between client and server, by implementing an experimental protocol SPDY. This amassed interest from developers working on both browsers and servers. Defining an increase in responsiveness, and solving the problem of duplication of data transmitted, SPDY served as the foundations of the HTTP/2 protocol.
The HTTP/2 protocol has several prime differences from the HTTP/1.1 version:

1. It is a binary protocol rather than text. It can no longer be read and created manually. Despite this hurdle, improved optimization techniques can now be implemented.
2. 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.
3. It compresses headers. As these are often similar among a set of requests, this removes duplication and overhead of data transmitted.
4. It allows a server to populate data in a client cache, in advance of it being required, through a mechanism called the server push.

List 5 differences between Browser JS vs Node Js.

Topic	**Node JS**	**Browser JS**
Definition	Node Provides standalone mechanism	Its is a inbuilt JS enviornment present in browser
Usage	Backend	Forntend
Global Object	global	Window
JS engine	V8	Different for Different browsers
Asynchronous activity handle by	Libuv	WEBAPI

what happens when you type a URL in the address bar in the browser?

1. You enter a URL into a web browser
2. The browser looks up the IP address for the domain name via DNS
3. Initiate the TCP connection in between your machine and the server
4. The browser sends a HTTP request to the server
5. The server sends back a HTTP response
6. The browser begins rendering the HTML
7. The browser sends requests for additional objects embedded in HTML (images, css, JavaScript) and repeats steps 3-5.
8. Once the page is loaded, the browser sends further async requests as needed.