**Q1. Difference b/w HTTP1.1 vs HTTP2**

HTTP2 was derived from the earlier experimental SPDY protocol.

Few major features of HTTP2 are:

**Request multiplexing over a single TCP connection.**

HTTP/2 can send multiple requests for data in parallel over a single TCP connection which can be considered the most advanced feature of the HTTP/2 protocol because it lets you to download web files asynchronously from one server. Most modern browsers limit TCP connections to one server which in turn reduces additional round trip time (RTT).

# Header compression

HTTP/2 compress many redundant header frames. It uses the HPACK specification as a simple and secure approach to header compression. Both client and server maintain a list of headers used in previous client-server requests.

HPACK then compresses the individual value of each header before it is transferred to the server, which then looks up to the encoded information in a list of previously transferred header values to reconstruct the full header information.

# C) Binary protocol

The latest HTTP version has evolved significantly in terms of capabilities and attributes such as transforming from a text protocol to a binary protocol. HTTP1.x used to process text commands to complete request-response cycles. HTTP/2 will use binary commands (in 1s and 0s) to execute the same tasks. This attribute eases complications with framing and simplifies implementation of commands that were confusingly intermixed due to commands containing text and optional spaces.

Browsers using HTTP/2 implementation will convert the same text commands into binary before transmitting it over the network which helps in

* Less prone to errors.
* Lighter network footprint.
* Effective network resource utilization.
* Reduced network latency and improved throughput.

# D) HTTP/2 Server Push

This allows the server to send additional cacheable information to the client that isn’t requested but is anticipated in future requests.

Benefits are:

* The client saves pushed resources in the cache.
* The client can reuse these cached resources across different pages.
* The server can multiplex pushed resources along with originally requested information within the same TCP connection.

**Q 2. Http version history**

# HTTP 0.9: The One-Line Protocol

The original HTTP proposal by Tim Berners-Lee was designed help with the adoption of his World Wide Web. In 1991, Berners-Lee outlined the motivation for the new protocol and listed several high-level design goals: file transfer functionality, ability to request an index search of a hypertext archive, format negotiation, and an ability to refer the client to another server. To prove the theory in action, a simple prototype was built, which implemented a small subset of the proposed functionality:

* + Client request is a single ASCII character string.
  + Client request is terminated by a carriage return (CRLF).
  + Server response is an ASCII character stream.
  + Connection is terminated after the document transfer is complete
  + Server response is a hypertext markup language (HTML).

# HTTP/1.0:

The period from 1991 to 1995 is one of rapid coevolution of the HTML specification, a new breed of software known as a web browser.

we needed a protocol that could serve more than just hypertext documents, provide richer metadata about the request and the response, enable content negotiation, and more.

Both the request and response headers were kept as ASCII encoded, but the response object itself could be of any type: an HTML file, a plain text file, an image, or any other content type.

In addition to media type negotiation, the RFC also documented several other commonly implemented capabilities: content encoding, character set support, multi-part types, authorization, caching, proxy behaviors, date formats, and more.

# HTTP/1.1: Internet Standard

The first official HTTP/1.1 standard is defined in RFC 2068, which was officially released in January 1997, roughly six months after the publication of HTTP/1.0. Then, two and a half years later, in June of 1999, several improvements and updates were incorporated into the standard and were released as RFC 2616.The HTTP/1.1 standard resolved a lot of the protocol ambiguities found in earlier versions and introduced a number of critical performance optimizations: keepalive connections, chunked encoding transfers, byte-range requests, additional caching mechanisms, transfer encodings, and request pipelining.

With these capabilities in place, we can now inspect a typical HTTP/1.1 session as performed by any modern HTTP browser and client:

# HTTP/2: Improving Transport Performance

To meet these new challenges, HTTP must continue to evolve, and hence the HTTPb is working group announced a new initiative for HTTP/2 in early 2012:

*.* There is emerging implementation experience and interest in a protocol that retains the semantics of HTTP without the legacy of HTTP/1.x message framing and syntax, which have been identified as hampering performance and encouraging misuse of the underlying transport.

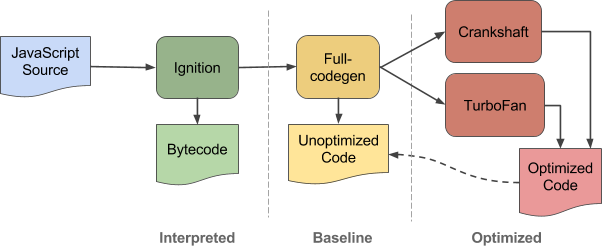
The primary focus of HTTP/2 is on improving transport performance and enabling both lower latency and higher throughput. The major version increment sounds like a big step, which it is and will be as far as performance is concerned, but it is important to note that none of the high-level protocol semantics are affected: all HTTP headers, values, and use cases are the same.

Any existing website or application can and will be delivered over HTTP/2 without modification: you do not need to modify your application markup to take advantage of HTTP/2. The HTTP servers will have to speak HTTP/2, but that should be a transparent upgrade for most users. The only difference if the working group meets its goal, should be that our applications are delivered with lower latency and better utilization of the network link!

**Q 3. List 5 difference between Browser JS (console) vs Nodejs**

* In the browser, most of the time what you are doing is interacting with the DOM, or other Web Platform APIs like Cookies. Those do not exist in Node.js. You do not have the document, window and all the other objects that are provided by the browser.
* 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 you control the environment. Unless you are building an open source application that anyone can deploy anywhere, you know which version of Node.js you will run the application on. Compared to the browser environment, where you do not get the luxury to choose what browser your visitors will use, this is very convenient.
* 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 with using 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.js, you will not need that.
* Another difference is that Node.js uses the CommonJS module system, while in the browser we are starting to see the ES Modules standard being implemented.

**Q 4. 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 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.

**Q5. What happens when you type a URL in the address bar in the browser?**

DNS (Domain Name System) is a database that maintains the name of the website (URL) and the IP address it links to. Every single URL on the internet has a unique IP address assigned to it. The IP address belongs to the computer which hosts the server of the website we are requesting to access. For example, [www.google.com](http://www.google.com/) has an IP address of 209.85.227.104. So if you’d like, you can reach [www.google.com](http://www.google.com/) by typing [http://209.85.227.104](http://209.85.227.104/) on your browser. DNS is a list of URLs, and their IP addresses, like how a phone book is a list of names and their corresponding phone numbers.

You can easily access a website by typing the correct IP address for it on your browser. Therefore, it is easier to remember the name of the website using a URL and let DNS do the work for us by mapping it to the correct IP.

* First, it checks the browser cache. The browser maintains a repository of DNS records for a fixed duration for websites you have previously visited. So, it is the first place to run a DNS query.
* Second, the browser checks the OS cache. If it is not in the browser cache, the browser will make a system to your underlying computer OS to fetch the record
* Third, it checks the router cache. If it is not on your computer, the browser will communicate with the router that maintains its’ own cache of DNS records.
* Fourth, it checks the ISP cache. If all steps fail, the browser will move on to the ISP. Your ISP maintains its’ own DNS server, which includes a cache of DNS records, which the browser would check with the last hope of finding your requested URL.