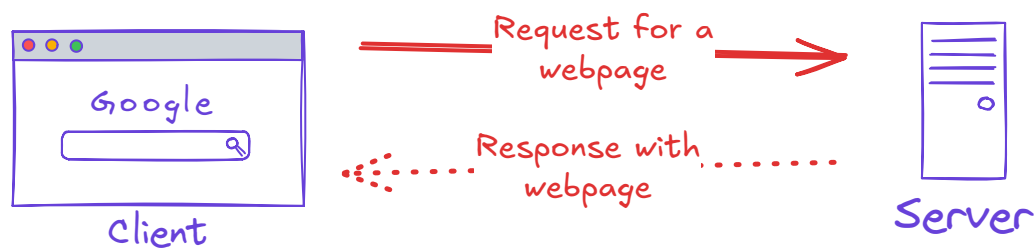


Client-Server Architecture

To send data over the internet we require an architecture where data is owned by one end and data is requested by another end. Client-Server architecture involves two main parts: the client and the server. The client is responsible for showing data to the user, like how a web browser displays a webpage. It sends requests to the **server** to get the data it needs. The server, on the other hand, holds and manages the data, and sends it back to the client when asked. The server acts as the central authority, making sure the data is stored and delivered correctly to the client. In simple terms, the client asks for data, and the server provides it.



The data between client and server is exchanged using the request response model. Client sends the request to server and after analyzing the request server sends back the response to client

Unlike Client-Server, there is Peer to Peer mechanism also in which there are no clients and there are no servers, there are only peers, or we can say that every peer can act as both client and server.

Peer-Peer Architecture

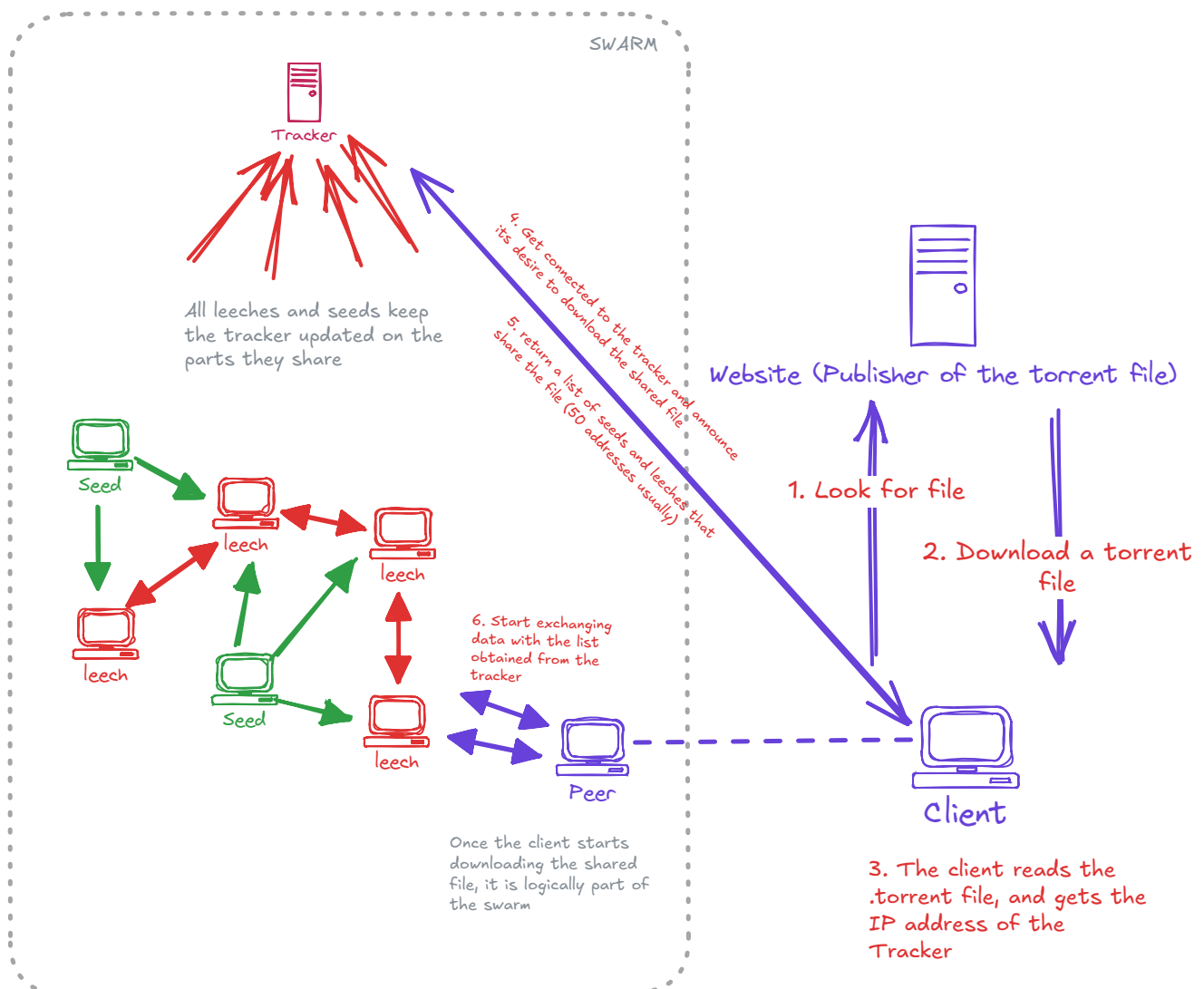
Some common examples of Peer-Peer is Blockchain, Torrent and earlier Skype used to be Peer-Peer. To understand Peer-Peer architecture, let's take the example of Bit-Torrent.

How Bit-Torrent Works?

Suppose you have to download the latest movie from Bit-Torrent. First you will visit a website which have a torrent file. This torrent file is not the actual movie but a few KBs file which have information of Torrent's Tracker Server.

Tracker Servers

Tracker servers are actual servers but they do not contain the movie but they have list of peers (addresses) which have the movie downloaded in their computers.



Tracker servers are actual servers but they do not contain the movie but they have list of peers (addresses) which have the movie downloaded in their computers.

After getting the list of peer addresses, our computer will try to connect to different available peers which are near to our network to download different parts of the movie from different available peers.

A single movie can be divided into multiple parts and these multiple parts based on the availability and distance can be downloaded from different peers.

A peer who contributes to the Torrent system, that is allow to upload movie for other peer is called Seeder.

A peer who only downloads from the network and do not contribute to the network is called Leacher.

Usually when we connect to Bit-Torrent, we are both Seeder and Leacher. Which means we download movie from other peers as well as we uploads for other peers making us both Leacher & Seeder.

That's why when you use Bit-Torrent you can see there is a download speed as well as upload speed when you are only downloading something.

HTTP (Hypertext Transfer Protocol)

HTTP was invented because prior to HTTP there was no standard way to send web pages over the internet between browsers and servers.

HTTP stands for Hypertext Transfer Protocol. It is a set of rules used to transfer hypertext (like clickable links) between a client and server over the internet. Hypertext is more than just regular text because it includes elements like hyperlinks that let you navigate to other pages or sections.

HTTP is an Application layer protocol, and it uses TCP (Transmission Control Protocol) to transfer data. However, with HTTP/3, the transport protocol changed to QUIC, which is based on UDP (User Datagram Protocol).

Method Path Protocol Version

GET / HTTP/1.1

Host: codersgyan.com
Accept-Language: en

Headers

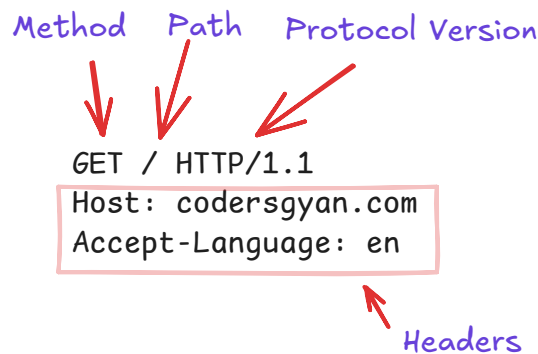


Figure. Client Request

Protocol Version Status Code Status Message

HTTP/1.1 200 OK

Date: Sat, 09 Jan 2025 14:28:02 GMT
Server: Nginx
Last-Modified: Tue, 01 Jan 2025 13:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

Headers

<html>
<head>
<title> Coder's Gyan </title>
</head>
<body>
<h1> Welcome to Coder's Gyan </h1>
</body>
</html>

Body

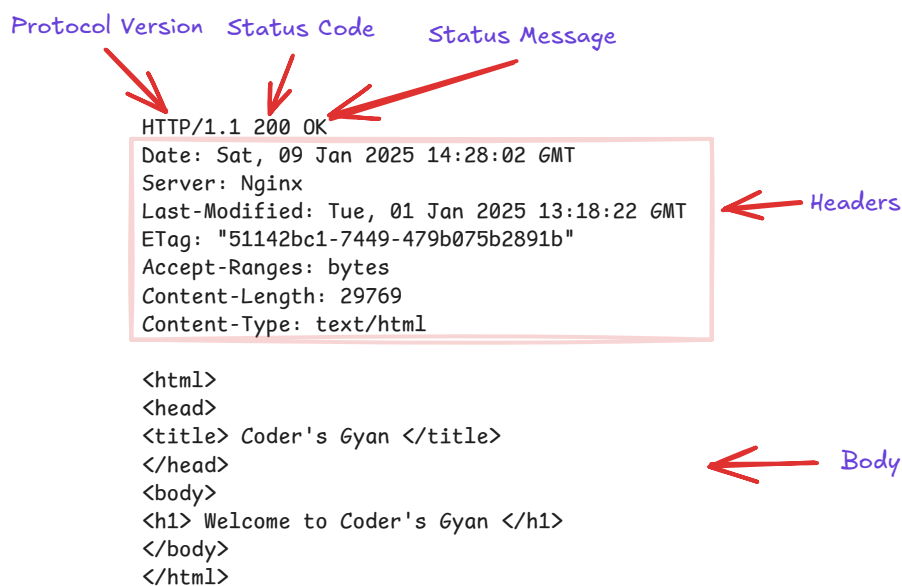


Figure. Client Response

It is also known as Request-Response protocol, where client who needs data send a request to server and server who owns the data, processes that request and sends the response which has the requested data.

HTTP is a stateless protocol. It means it does not store any information regarding the previous requests. To make our web applications stateful either client or server or both needs to save some information and needs to send it with each request or response. e.g. Token Authentication management.

The reason why HTTP is made stateless so it can be scaled easily.

HTTP/0.9

HTTP/0.9 was the first version of the Hypertext Transfer Protocol (HTTP), introduced in 1991 by Tim Berners-Lee. It was a very simple protocol designed for retrieving HTML documents.

Simple request format: Only supported GET requests (no headers, no status codes).

Response: Only returned raw HTML content (no metadata, images, or other media types).

Connection handling: A single request per connection; after sending the response, the connection was closed.

No HTTP headers: No support for additional data like content type, caching, or cookies.



HTTP/1.0

HTTP/1.0 was introduced in 1996 (defined in RFC 1945) and brought significant improvements over HTTP/0.9. It added headers, status codes, and support for different content types, making it much more flexible and useful for the growing web.

Introduced status codes like 200, 404 & 500

It has added two new HTTP methods POST & HEAD apart from GET which was already in HTTP 0.9.

Enabled serving images, videos, CSS, JavaScript, and other files.

Status Codes

Info 1XX

Success 2XX

Redirection 3XX

Client Error 4XX

Server Error 5XX

HTTP Requests included
POST
HEAD

Connections closed after each request, causing inefficiency and increased latency.

No persistent connections: Each request required a new TCP connection, slowing down performance.

Limited virtual hosting support: Required separate IPs for each domain before Host headers became widely used.

Inefficient for modern web applications: No support for caching, compression, or advanced connection handling.



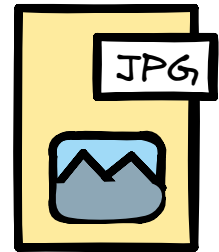
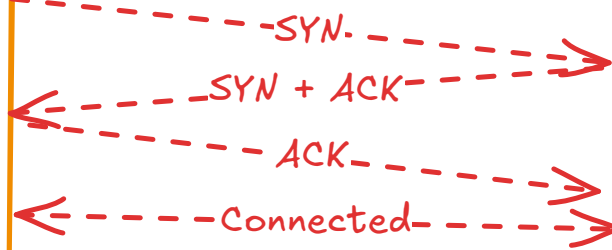
HTTP 1.0



Server

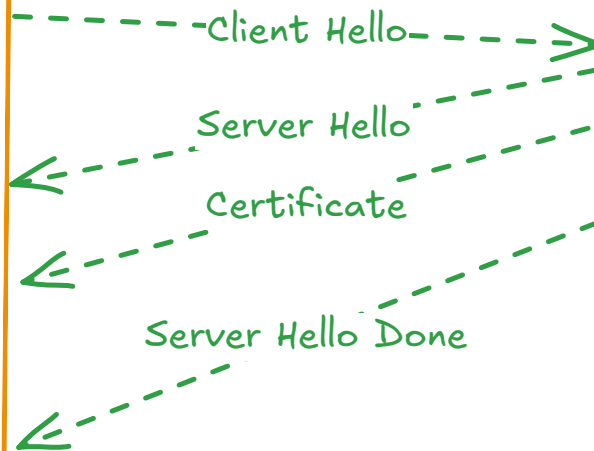
Client

1. TCP Handshake



JPG

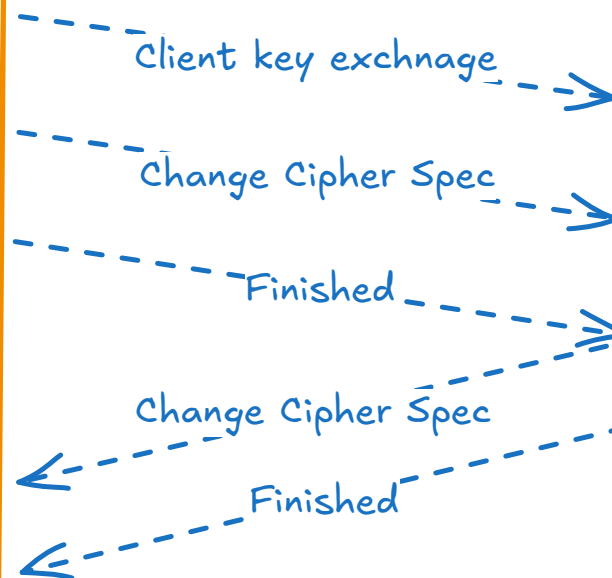
2. Certificate Check



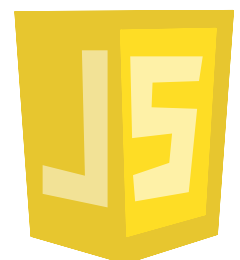
CSS



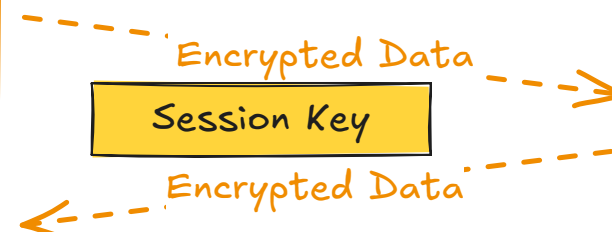
3. Key Exchange



JS



4. Data Transmission



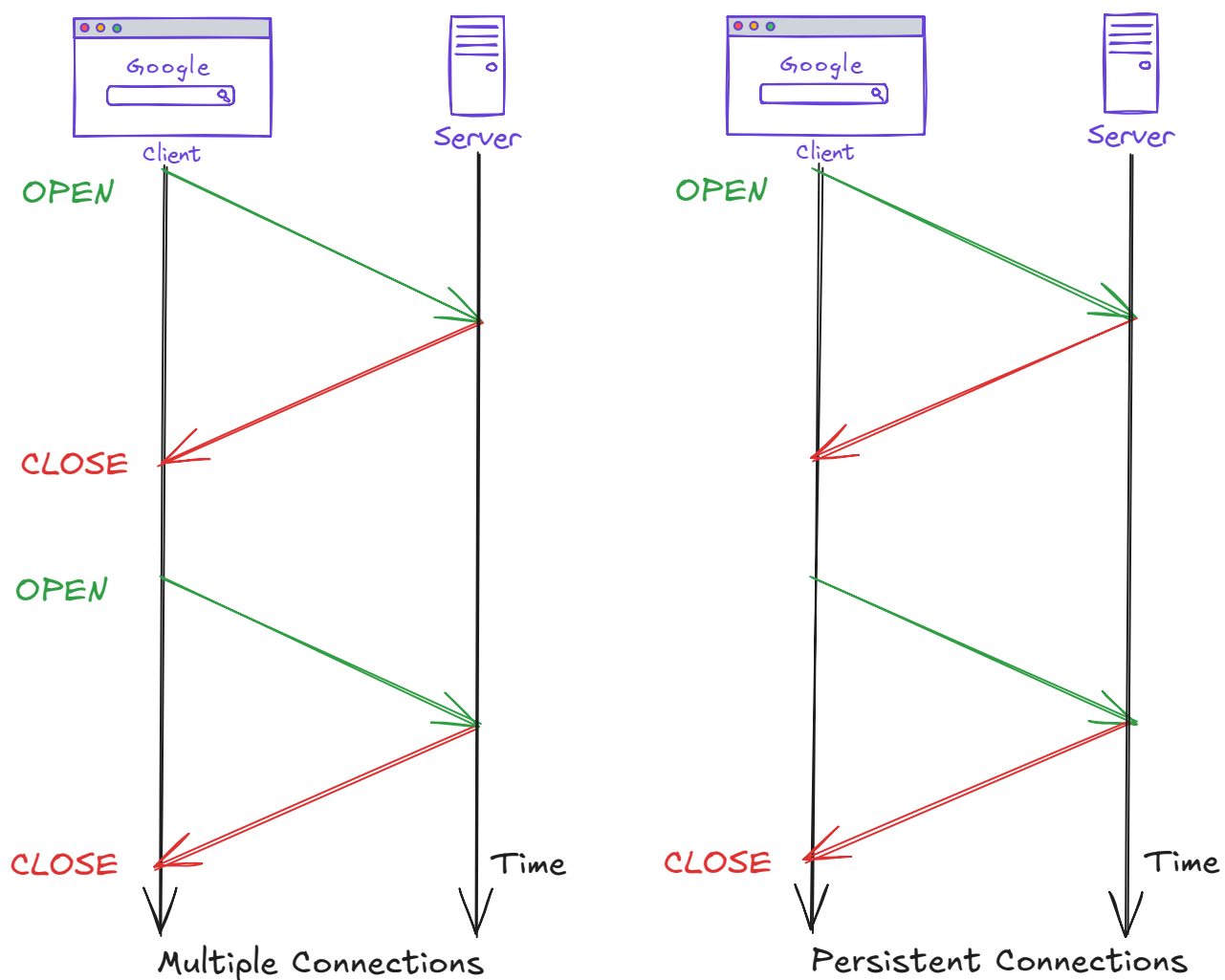
HTTP/1.1

HTTP/1.1, introduced in 1997 (defined in RFC 2068 and later updated in RFC 2616), brought major performance improvements and new features over HTTP/1.0. It became the most widely used HTTP version for decades.

Persistent Connections (Keep-Alive by Default)

Unlike HTTP/1.0, where each request required a new TCP connection, HTTP/1.1 keeps connections open for multiple requests.

Allows servers to send responses in smaller chunks instead of a single large response



Allows sending multiple requests without waiting for previous responses.

Unlike HTTP/1.0, where a separate IP was needed for each website, HTTP/1.1 introduced the mandatory Host header, allowing multiple domains to share a single IP address.

Cache-Control: Fine-grained control over caching (e.g., max-age, no-cache).

ETag: Helps prevent unnecessary re-downloads by validating if a resource has changed.

New Methods Introduced

OPTIONS → Lists allowed methods for a resource.

PUT → Uploads/replaces a resource.

DELETE → Removes a resource.

TRACE → Debugging tool to see how a request is processed by intermediate servers.

CONNECT → Used for establishing a tunnel (e.g., for HTTPS proxies)

No Multiplexing = But have Persistent Connection (How many parallel requests allowed & does it use same TCP Connection)

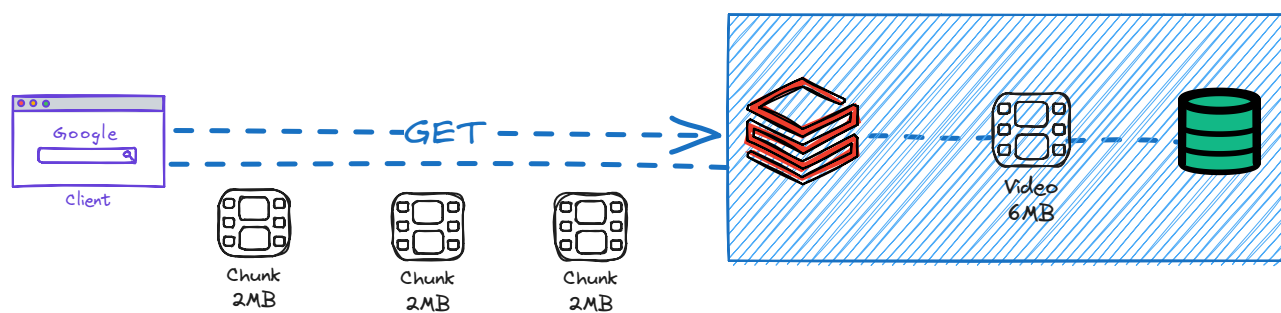
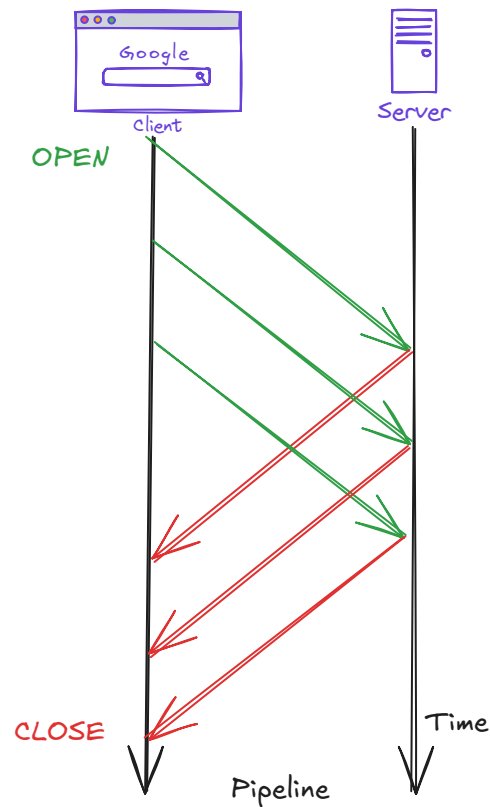
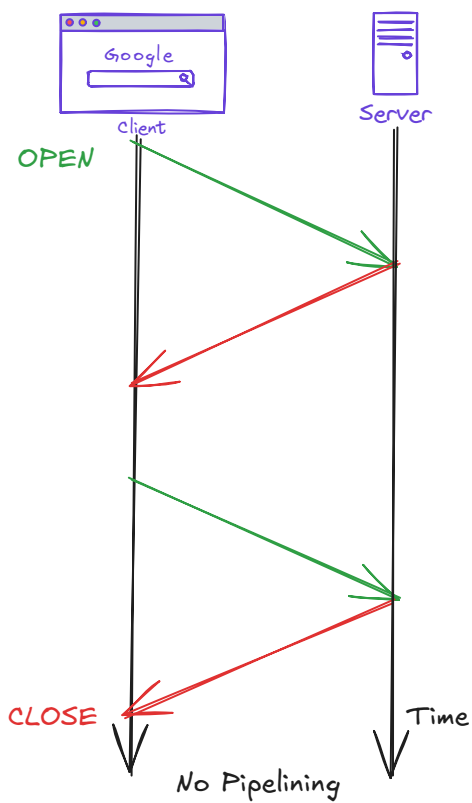
Request 1-6 → Wait for response → Request 6-12 → Wait for response
→ Request 12-18 → etc.

Optional Encryption Via TLS

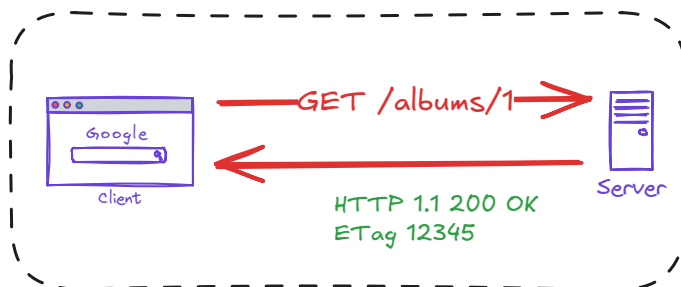
Head of Line Blocking = If one request or packet is delayed, all subsequent requests must wait (even if unrelated).

No header compression

Slower due to No Multiplexing and Head of Line Blocking and no header compression



```
<filesMatch "\.(ico|pdf|flv|jpeg|jpg|png|gif|js|css|swf)$">
Header set Cache-Control "max-age=84600, public"
</filesMatch>
```



HTTP/2

HTTP/2, introduced in 2015 (standardized in RFC 7540), was designed to improve the performance limitations of HTTP/1.1. It introduced multiplexing, header compression, and server push, making web communication faster and more efficient

1. Multiplexing - parallel http requests not sequential over same TCP connection
2. Optional Encryption via TLS.
3. Head of Blocking in TCP level = Since same TCP connection is used if any segment is lost then TCP stop all other resources to send and first retry that packet until receives.
4. Header Compression
5. Comparatively faster than HTTP/1.1 due to multiplexing but it gets limited by TCP.

Pipelining vs. Multiplexing

1. TCP Connections

HTTP/1.1 Pipelining → Uses a single TCP connection.

HTTP/2 Multiplexing → Uses a single TCP connection but more efficiently.

2. Request Handling

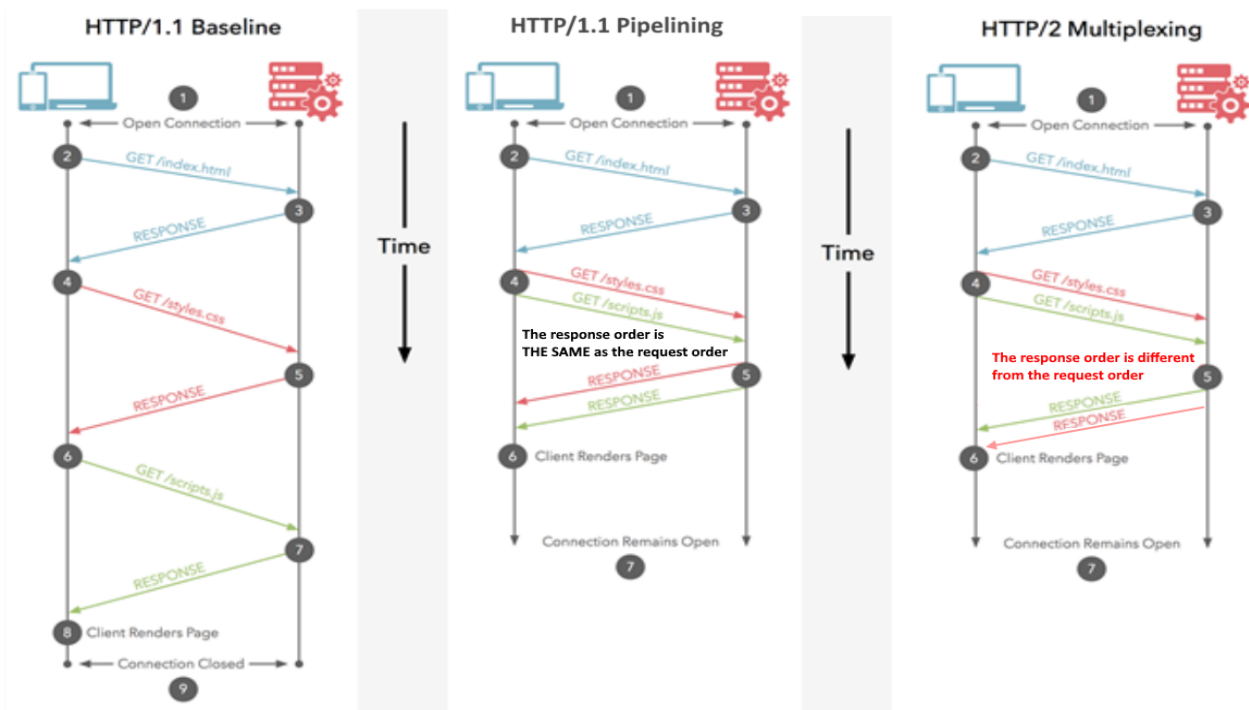
HTTP/1.1 Pipelining → Requests are sent sequentially without waiting for responses.

HTTP/2 Multiplexing → Requests are sent in parallel (simultaneously).

3. Response Order

HTTP/1.1 Pipelining → Responses must be returned in order.

HTTP/2 Multiplexing → Responses can be returned out of order.



4. Head-of-Line (HOL) Blocking

HTTP/1.1 Pipelining → Yes, if one request is slow, others must wait.

HTTP/2 Multiplexing → No, independent streams prevent blocking. However, because it still relies on TCP, packet loss in one stream blocks all other streams in that connection.

5. Efficiency & Performance

HTTP/1.1 Pipelining → Inefficient, as slow requests cause delays.

HTTP/2 Multiplexing → Highly efficient, allowing multiple requests & responses at the same time.

HTTP/3

HTTP/3, introduced in 2018 and standardized in RFC 9114, is the latest version of HTTP. It replaces TCP with QUIC (Quick UDP Internet Connections) to improve speed, reliability, and security.

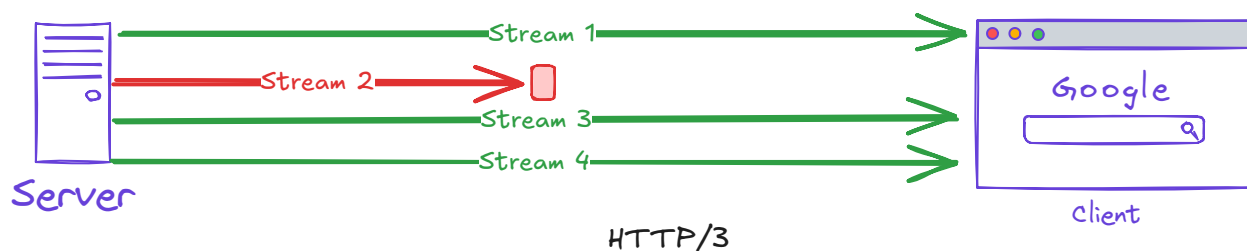
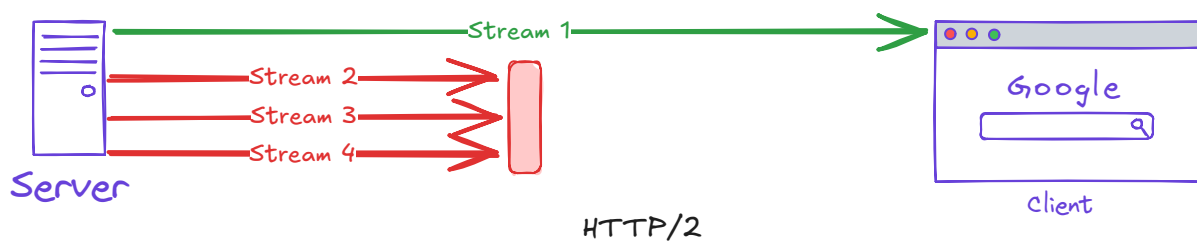
1. Supports Multiplexing

2. Built-in Encryption via TLS.

3. No Head of line Blocking = QUIC (UDP-based) handles lost packets independently in each stream, so one delayed packet doesn't block other streams.

4. Header Compression

5. Faster than both HTTP/1.1 & HTTP/2.



6. QUIC = Quick UDP Internet Connection, Unlike UDP it has the retry mechanism for lost packets
7. Requires support for QUIC (UDP-based), which may not be fully supported by older networks or devices.
8. May have compatibility issues with certain middleboxes (like firewalls or proxies) that are not QUIC-aware.
9. Requires more resources due to the additional complexity of QUIC, impacting performance on resource-constrained devices