Full Stack Application Development

BITS Pilani

innovate achieve lead

BITS Pilani

Pilani Campus

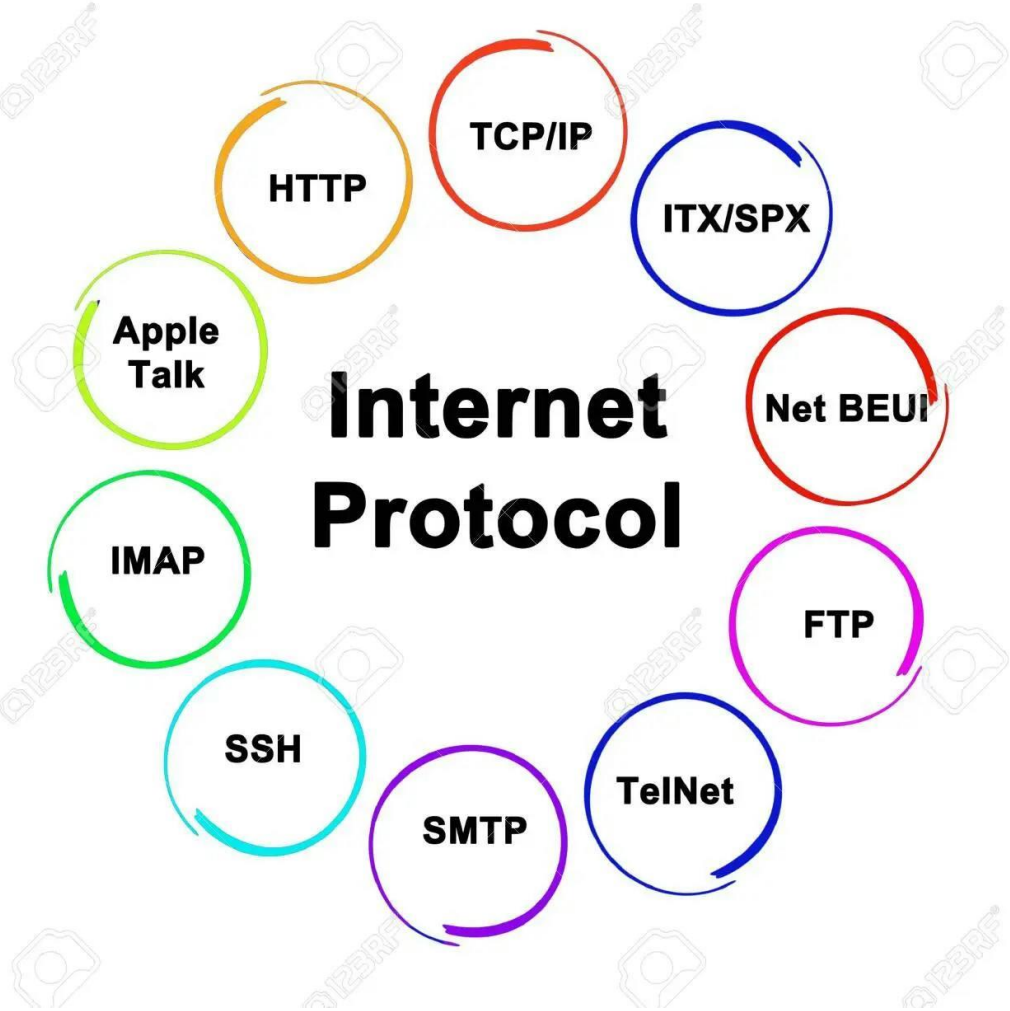
Module 3: Web Protocols

Agenda

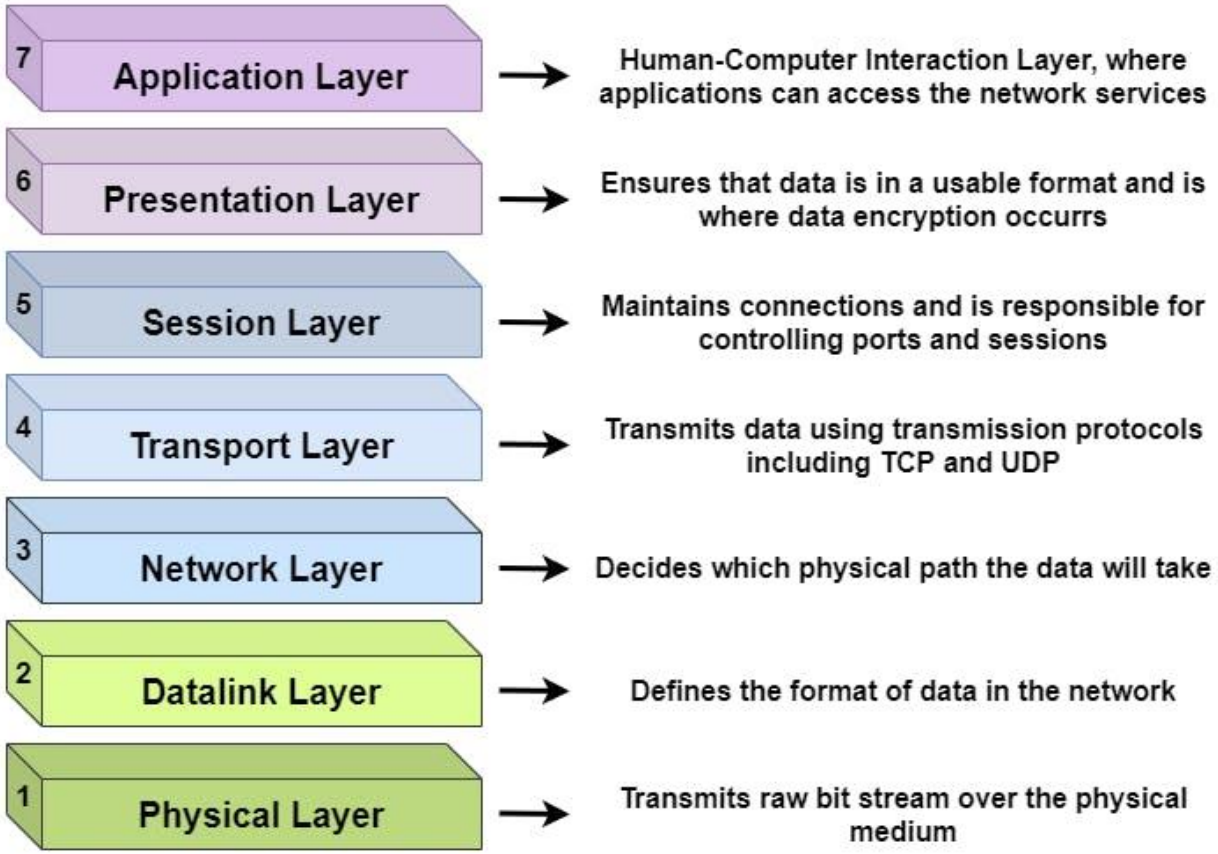
innovate achieve lead

- HTTP Request- Response and its structure
- HTTP Methods
- HTTP Headers
- Connection management - HTTP/1.1 and HTTP/2
- AJAX, Fetch API
- HTTP Polling
- Webhooks
- Server Sent Events
- Websockets

Web Protocols



OSI Model

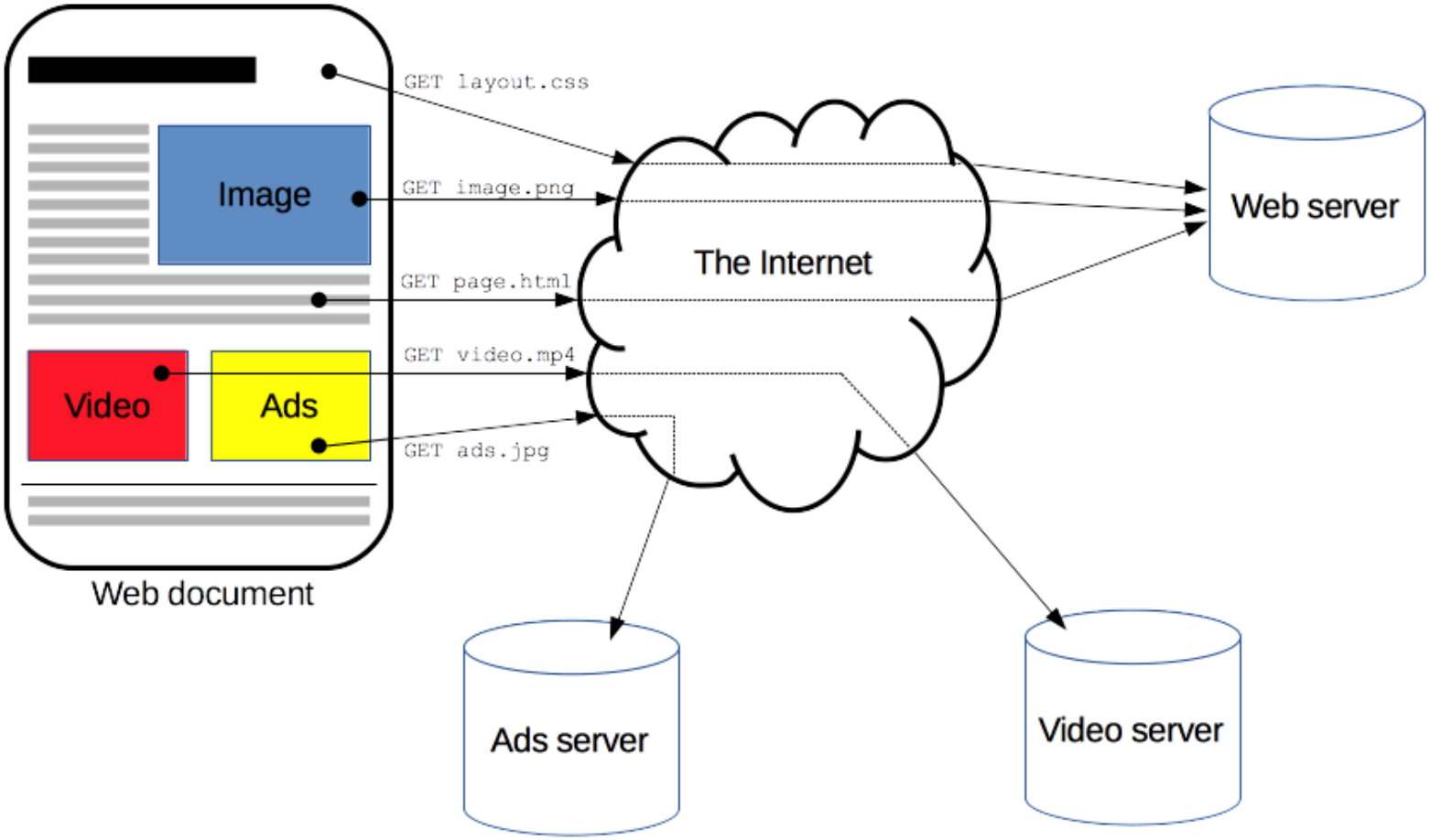

7
Application Layer
Human-Computer Interaction Layer, where applications can access the network services
6
Presentation Layer
Ensures that data is in a usable format and is where data encryption occurrs
5
Session Layer
Maintains connections and is responsible for controlling ports and sessions
4
Transport Layer
Transmits data using transmission protocols including TCP and UDP
3
Network Layer
Decides which physical path the data will take
2
Datalink Layer
Defines the format of data in the network
1
Physical Layer
Transmits raw bit stream over the physical medium

8 Popular Network Protocols

blog.bytebytego.com

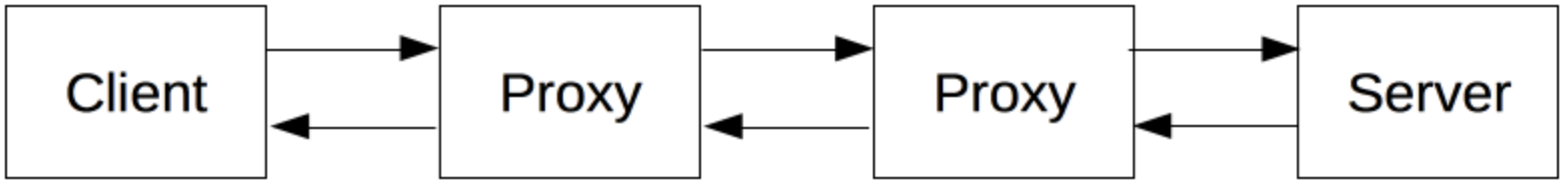
Protocol	How does It Work?	Use Cases
HTTP	TCP Connection HTTP REQ HTTP RESP	Web Browsing
HTTP/3 (QUIC)	UDP Connection 1 2 3 4 5	IoT Virtual Reality
HTTPS	TCP Connection public key session key encrypted data	Web Browsing
WebSocket	HTTP Upgrade Full Duplex	Live Chat Real-Time Data Transmission
TCP	SYN SYN + ACK ACK	Web Browsing Email Protocols
UDP	REQUEST RESPONSE	Video Conferencing
SMTP	sender SMTP Server receiver	Sending/Receiving Emails
FTP	Control Channel Data Channel	Upload/Download Files

HTTP


GET layout.css
Image
GET image.png
GET page.html
GET video.mp4
Video
Ads
GET ads.jpg
Web document
The Internet
Web server
Ads server
Video server

Question

Why do we need Proxies?

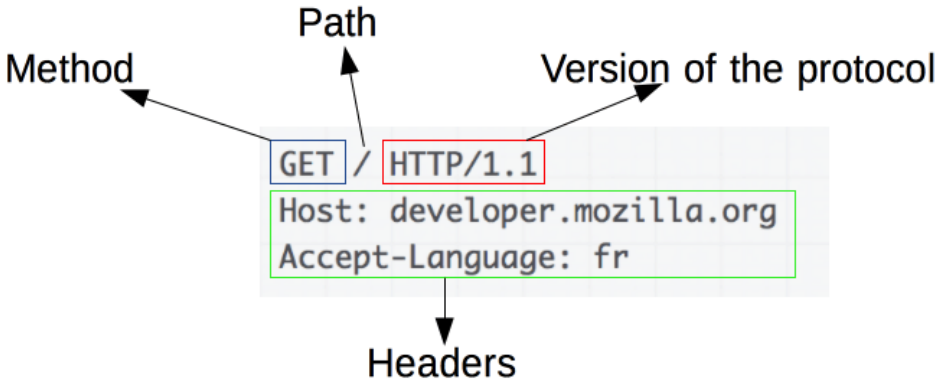


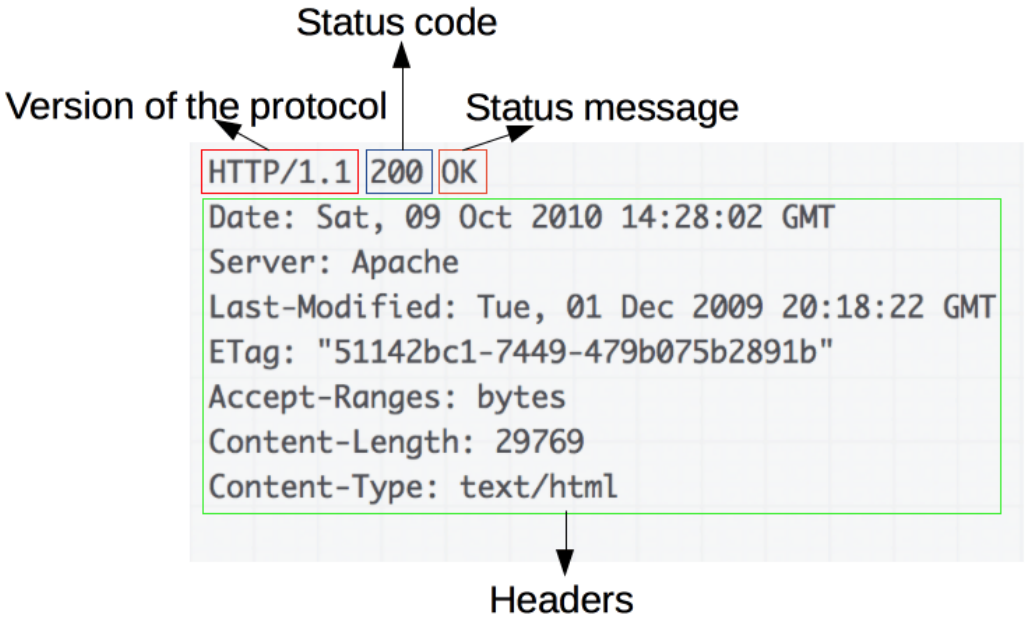
Proxies

Between the Web browser and the server, numerous computers and machines relay the HTTP messages. Due to the layered structure of the Web stack, most of these operate at the transport, network or physical levels, becoming transparent at the HTTP layer and potentially having a significant impact on performance. Those operating at the application layers are generally called **proxies**. These can be transparent, forwarding on the requests they receive without altering them in any way, or non-transparent, in which case they will change the request in some way before passing it along to the server. Proxies may perform numerous functions:

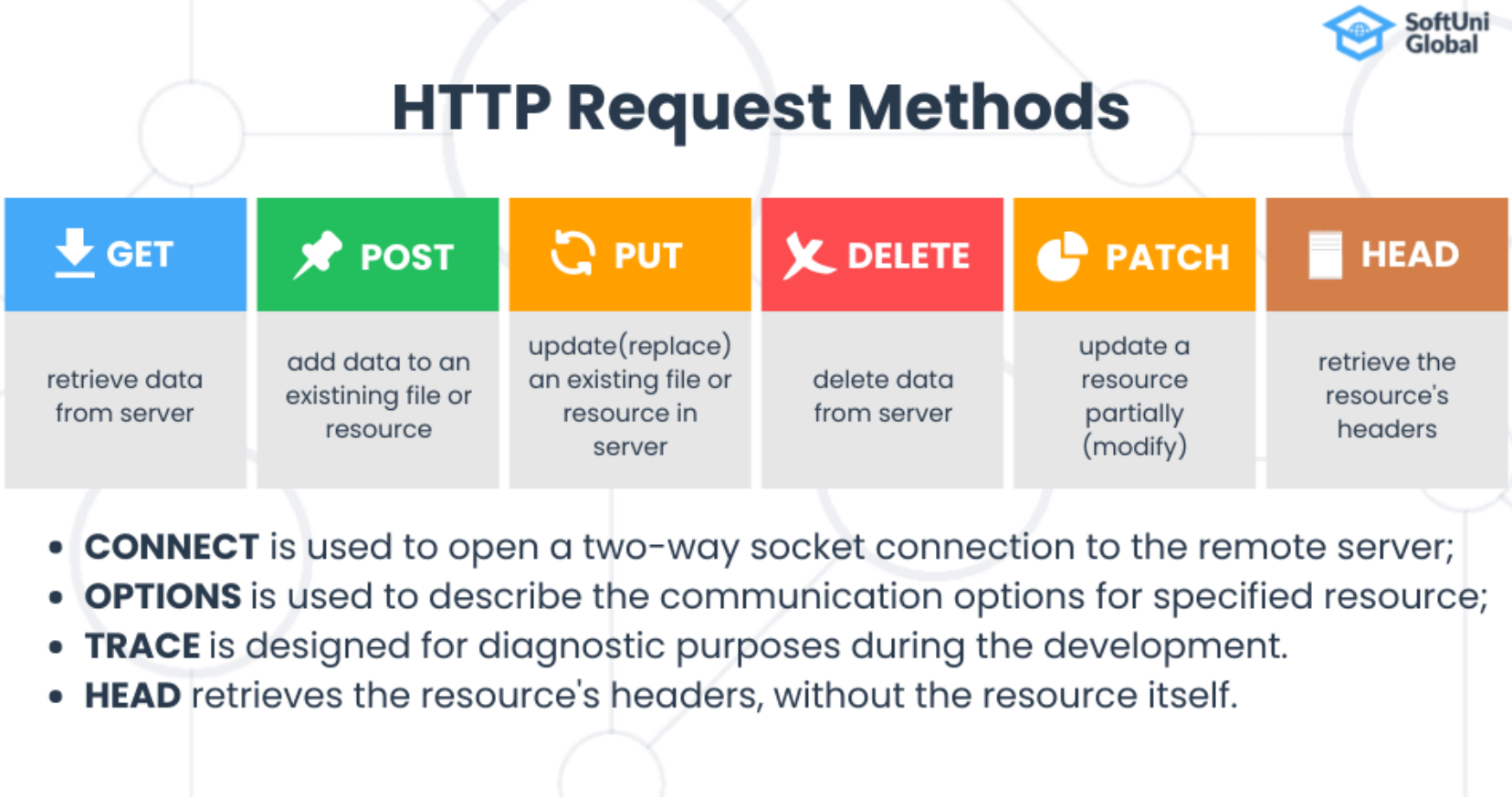
- **caching** (the cache can be public or private, like the browser cache)
- **filtering** (like an antivirus scan or parental controls)
- **load balancing** (to allow multiple servers to serve different requests)
- **authentication** (to control access to different resources)
- **logging** (allowing the storage of historical information)

HTTP Request and Response





HTTP Request Methods



HTTP Patch

innovate achieve lead

PATCH /users/123 HTTP/1.1

Host: example.com

Content-Type: application/json { "op": "**replace**", "path": "/email",
"value": "new_email@example.com" }

- This request uses the PATCH method to update the resource at /users/123.
- It specifies the content type as application/json, indicating the data format.
- The body contains a JSON object with three properties:
  - op: The operation to perform, which is "replace" in this case.
  - path: The path to the specific field to be modified, which is "/email".
  - value: The new value for the specified field, which is "new_email@example.com".

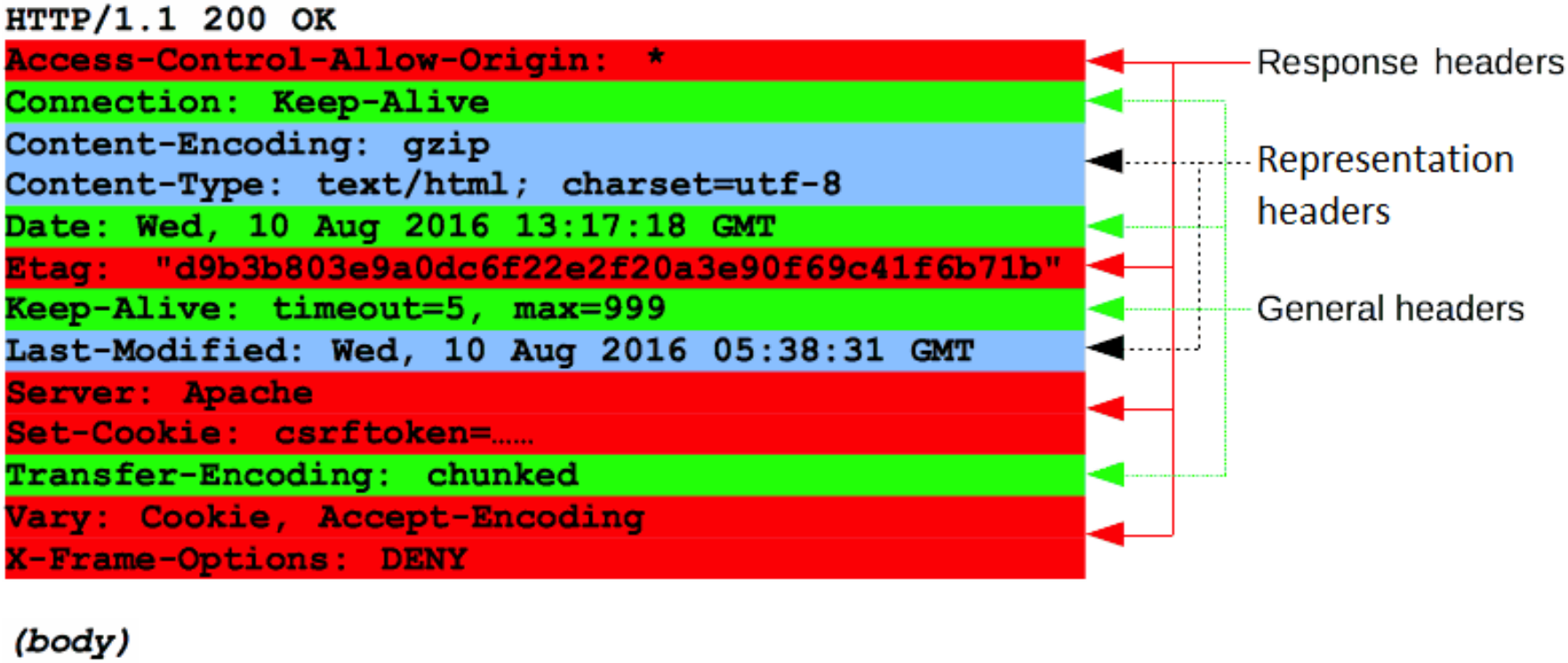
HTTP Request Methods

innovate achieve lead

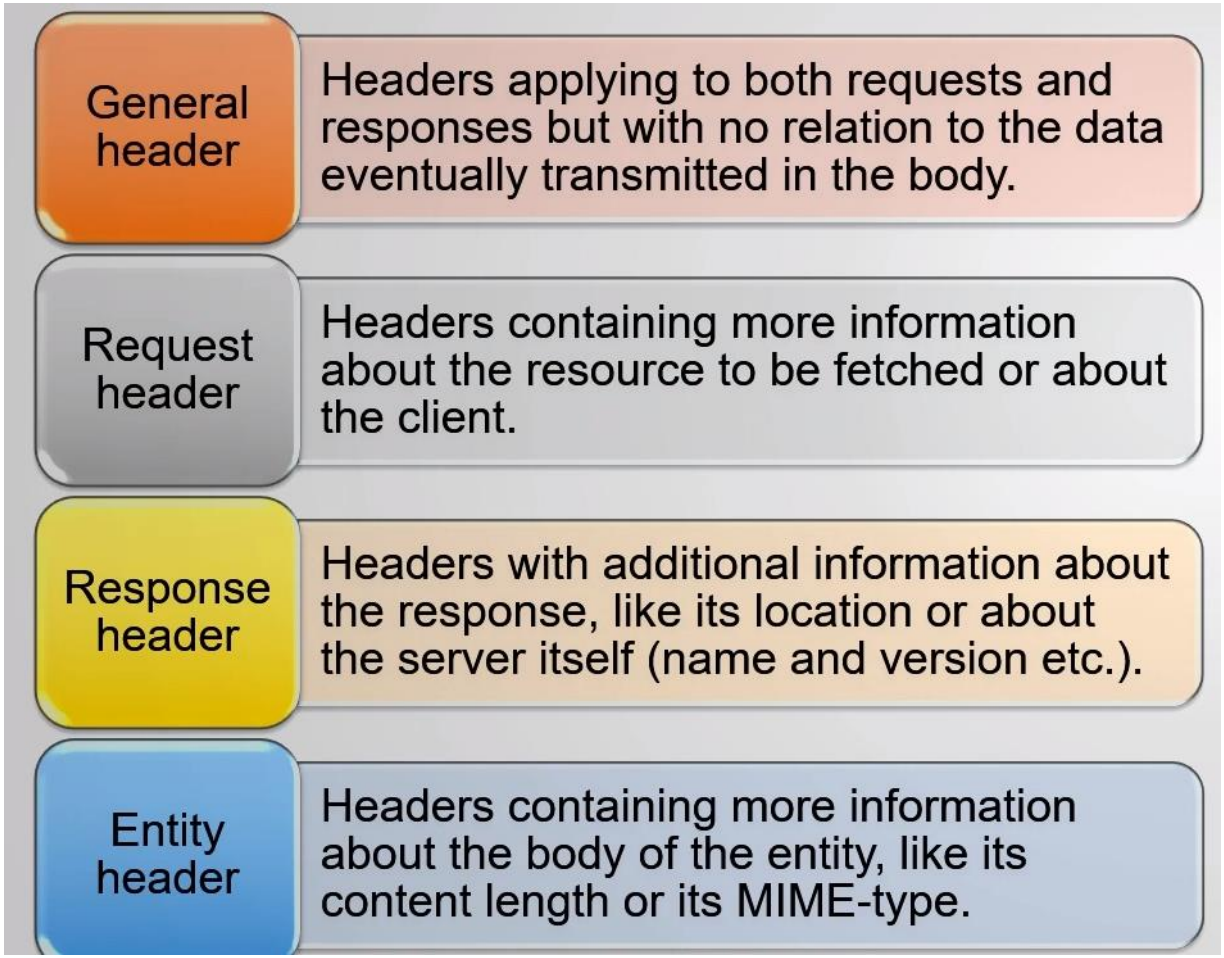
	SAFE	IDEMPOTENT
GET	Yes	Yes
POST	No	No
PUT	No	Yes
PATCH	No	No
DELETE	No	Yes
TRACE	Yes	Yes
HEAD	Yes	Yes
OPTIONS	Yes	Yes
CONNECT	No	No

An HTTP method is considered **idempotent** if **repeated identical requests** have the same **outcome** on the server, regardless of how many times they are sent. In simpler terms, it means **executing the request multiple times doesn't cause unintended side effects**.

HTTP Headers



HTTP Headers



HTTP Headers (Content)

- The Accept header lists the **MIME** types of media resources that the agent is willing to process.
- A media type also known as **MIME** type is a two-part identifier for file formats and format contents transmitted on the Internet.
- Each combined with a quality factor, a parameter indicating the relative degree of preference between the different **MIME** types.

Accept	image/gif, image/jpeg, /
Form	type/subtype
Examples	`text/plain`, `text/html`, `image/gif`, `image/jpeg`

HTTP Headers-content

The Accept-Charset header

- It indicates to the server what kinds of character encodings are understood by the user-agent.

```
Accept-Charset:utf-8
```

The Accept-Encoding header

- The Accept-Encoding header defines the acceptable content-encoding (supported compressions).
- The value is a q-factor list (e.g.: br, gzip;q=0.8) that indicates the priority of the encoding values.
- Compressing **HTTP** messages is one of the most important ways to improve the performance of a Web site.

```
Accept-Encoding:gzip, deflate
```

HTTP Headers-Content

innovate achieve lead

The Accept-Language header

- It is used to indicate the language preference of the user.

```
Accept-Language: en-us
```

The User-Agent header

- It identifies the browser sending the request.
- These headers helps us to understand more information about the Message body.

HTTP Headers - Content

innovate achieve lead

Content-Type

- Indicates the media type of the resource.

Content-Encoding

- Used to specify the compression algorithm.

Content-Language

- Describes the language(s) intended for the audience, so that it allows a user to differentiate according to the users' own preferred language.

Content-Length

- indicates the size of the entity-body, in decimal number of octets, sent to the recipient.

HTTP Headers-Caching

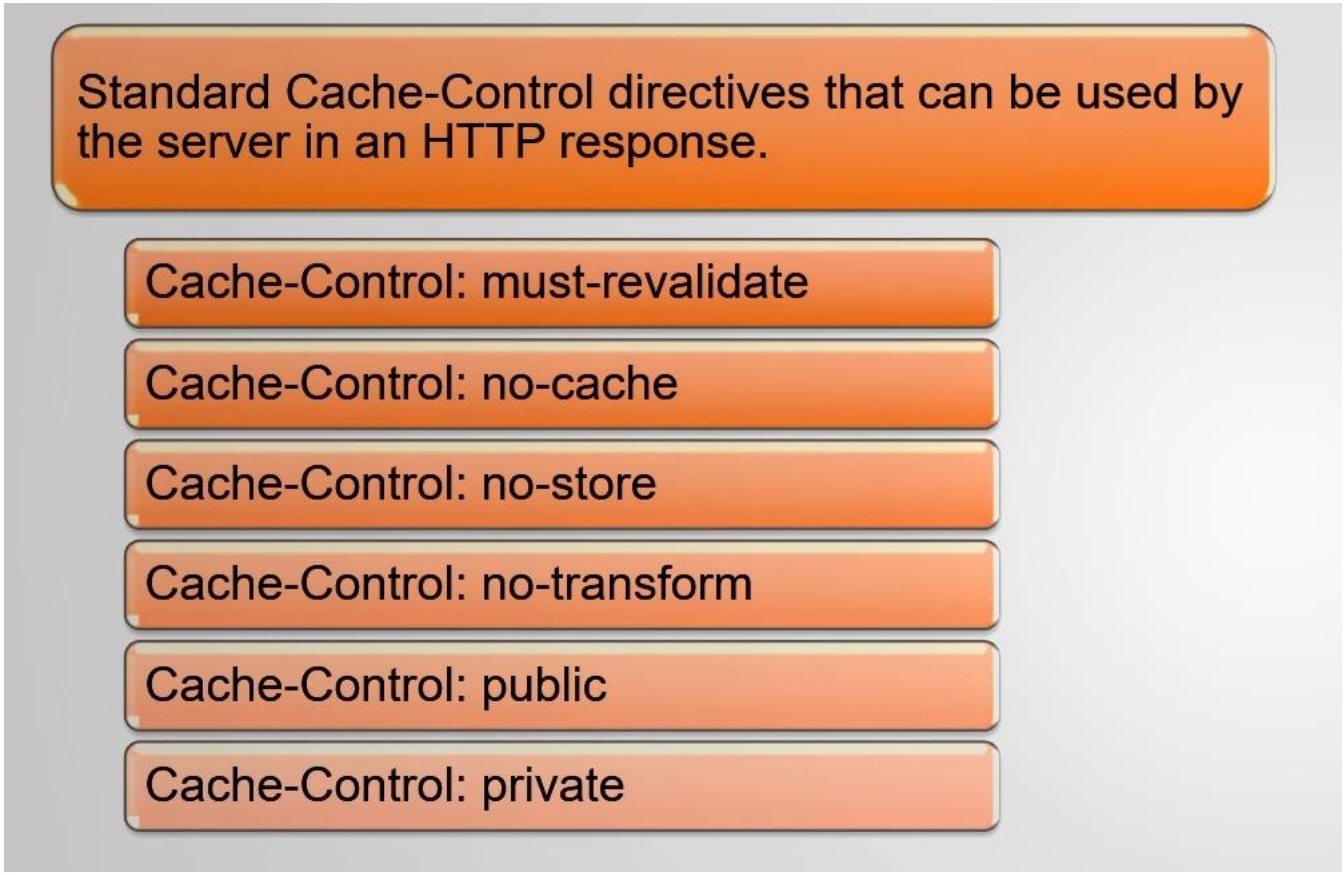
The Cache-Control general-header field is used to specify directives for caching mechanisms in both requests and responses.

Caching directives are unidirectional, i.e., directive in a request is not implying that the same directive is to be given in the response.

Standard Cache-Control directives that can be used by the client in an HTTP request.

- Cache-Control: max-age=<seconds>
- Cache-Control: no-cache
- Cache-Control: no-store
- Cache-Control: no-transform

HTTP Headers-Caching



HTTP Headers-Caching

innovate achieve lead

Cache Request Directives

```
Cache-Control:
   max-age=<seconds>
   max-stale[=<seconds>]
   min-fresh=<seconds>
   no-cache
   no-store
   no-transform
   only-if-cached
```

Cache Response Directives

```
Cache-Control:
   must-revalidate
   no-cache
   no-store
   no-transform
   public
   private
   proxy-revalidate
   max-age=<seconds>
   s-maxage=<seconds>
```

Example

```
Cache-Control: max-age: 31536000
```

Store this file for one year and use it instead of requesting to download a fresh copy during that time.

```
Cache-Control: no-cache, max-age: 31536000
```

Store this file for one year, but before you use it, check to see if a new version is available.

Example

innovate achieve lead

```
Cache-Control: no-store
```

Do not cache this file.

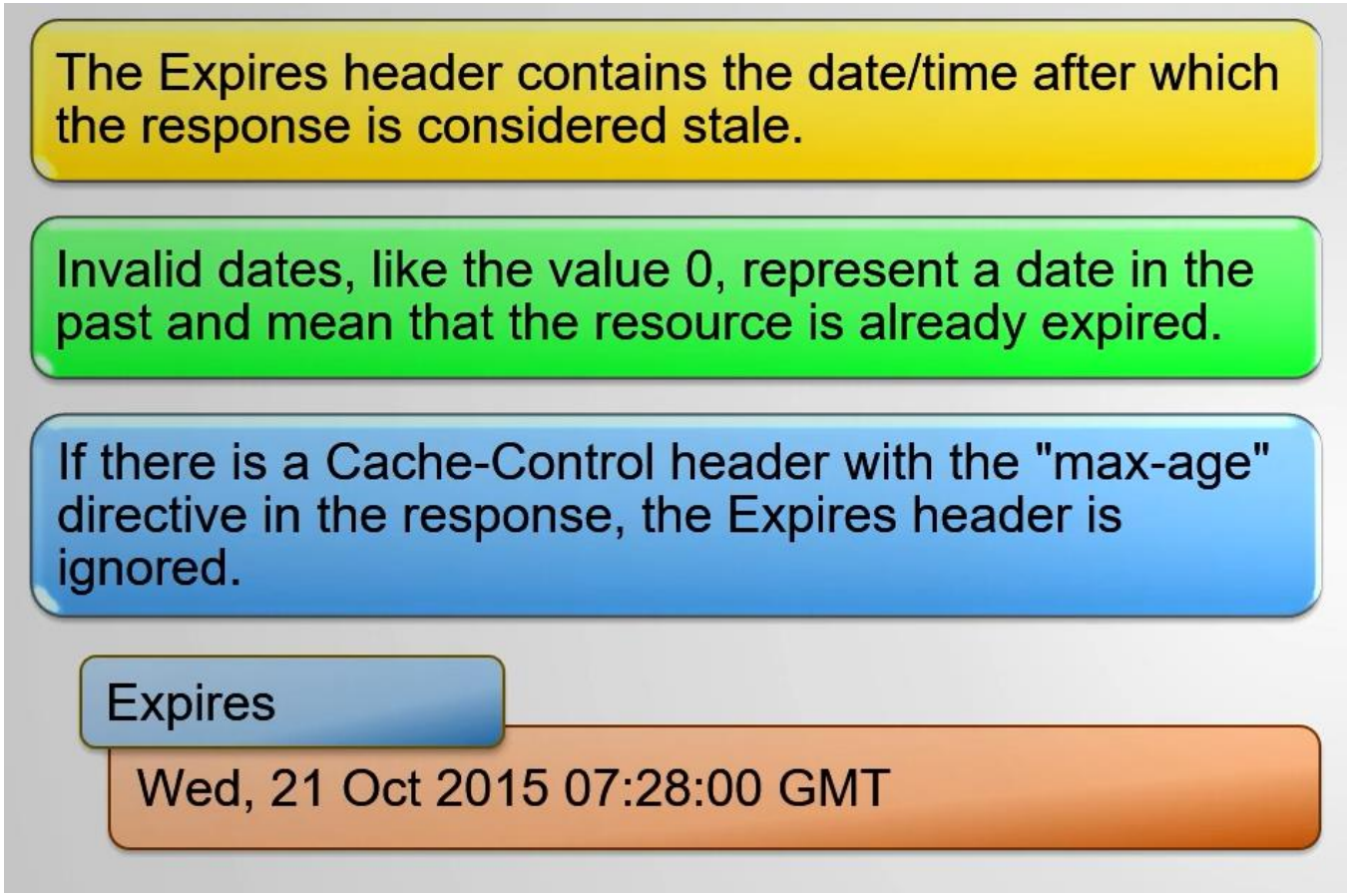
```
Cache-Control: max-age: 86400
```

Store this file for one day and use it instead of requesting to download a fresh copy during that time.

```
Cache-Control: max-age: 2592000
```

Store this file for one month and use it instead of requesting to download a fresh copy during that time.

HTTP Headers-caching



HTTP Headers-caching

The ETag HTTP response header is an identifier for a specific version of a resource.

If the resource at a given URL changes, a new Etag value must be generated.

ETag

"33a64df551425fcc55e4d42a148795d9f25f89d4"

The client will send the Etag value of its cached resource along in an If-None-Match header field:

If-None-Match

"33a64df551425fcc55e4d42a148795d9f25f89d4"

HTTP Headers-caching

The server compares the client's ETag (sent with If-None-Match) with the ETag for its current version of the resource

If both values match, the server send back a 304 Not Modified status, without any body

Tells the client that the cached version of the response is still good.

The Last-Modified response HTTP header contains the date and time at which the origin server believes the resource was last modified.

It is used as a validator to determine if a resource received or stored is the same.

Less accurate than an ETag header

HTTP Headers-Cookies

- An **HTTP** cookie is a small piece of data that a server sends to the user's web browser.
- The browser may store it and send it back with the next request to the same server.
- The Set-Cookie **HTTP** response header sends cookies from the server to the user agent.

```
Set-Cookie: <cookie-name>=<cookie-value>
```

- The Cookie **HTTP** request header contains stored **HTTP** cookies previously sent by the server with the Set-Cookie header.

```
Cookie: name=value; name2=value2; name3=value3
```

HTTP Headers-Connection Management

- The Connection general header controls whether or not the network connection stays open after the current transaction finishes.
- If the value sent is keep-alive, the connection is persistent and not closed, allowing for subsequent requests to the same server to be done.

```
Connection: keep-alive
```

```
Connection: close
```

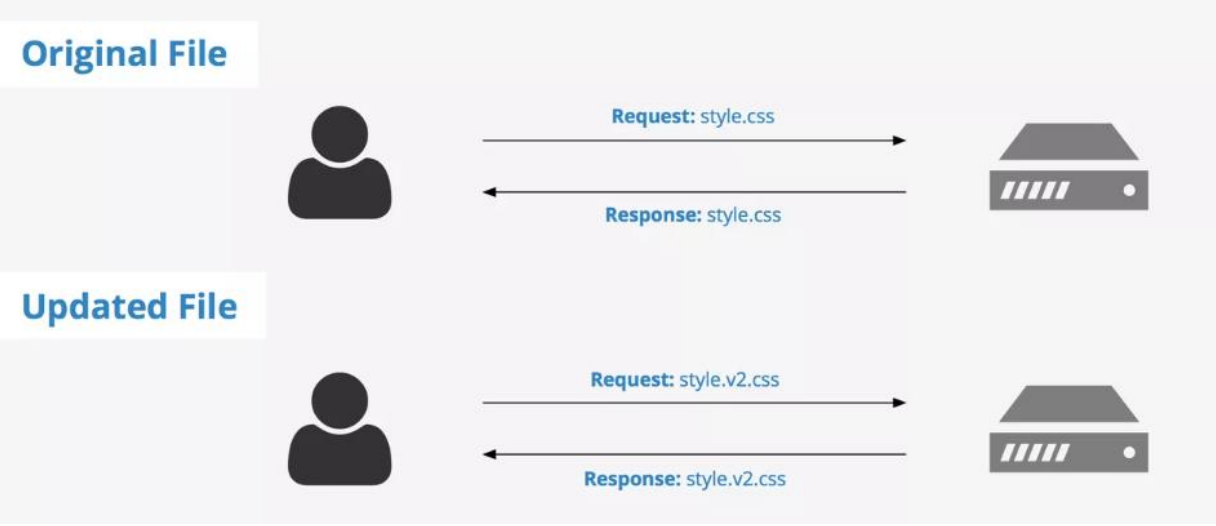
Keep-alive Header

- The Keep-Alive general header allows the sender to hint about how the connection may be used to set a timeout and a maximum amount of requests.

```
Keep-Alive: timeout=5, max=1000
```

Cache-busting

Cache busting is a technique web developers employ to ensure that users receive the most recent version of a website or application. It "busts" the browser's cache, forcing it to download new files instead of using outdated ones. This is particularly important when updates or changes are made to a site's content, design, or functionality, as it guarantees that users will experience the latest version without any hiccups.



https://community.sap.com/t5/technology-blogs-by-members/cache-buster-indexing-ui5-tooling-task/ba-p/13503701

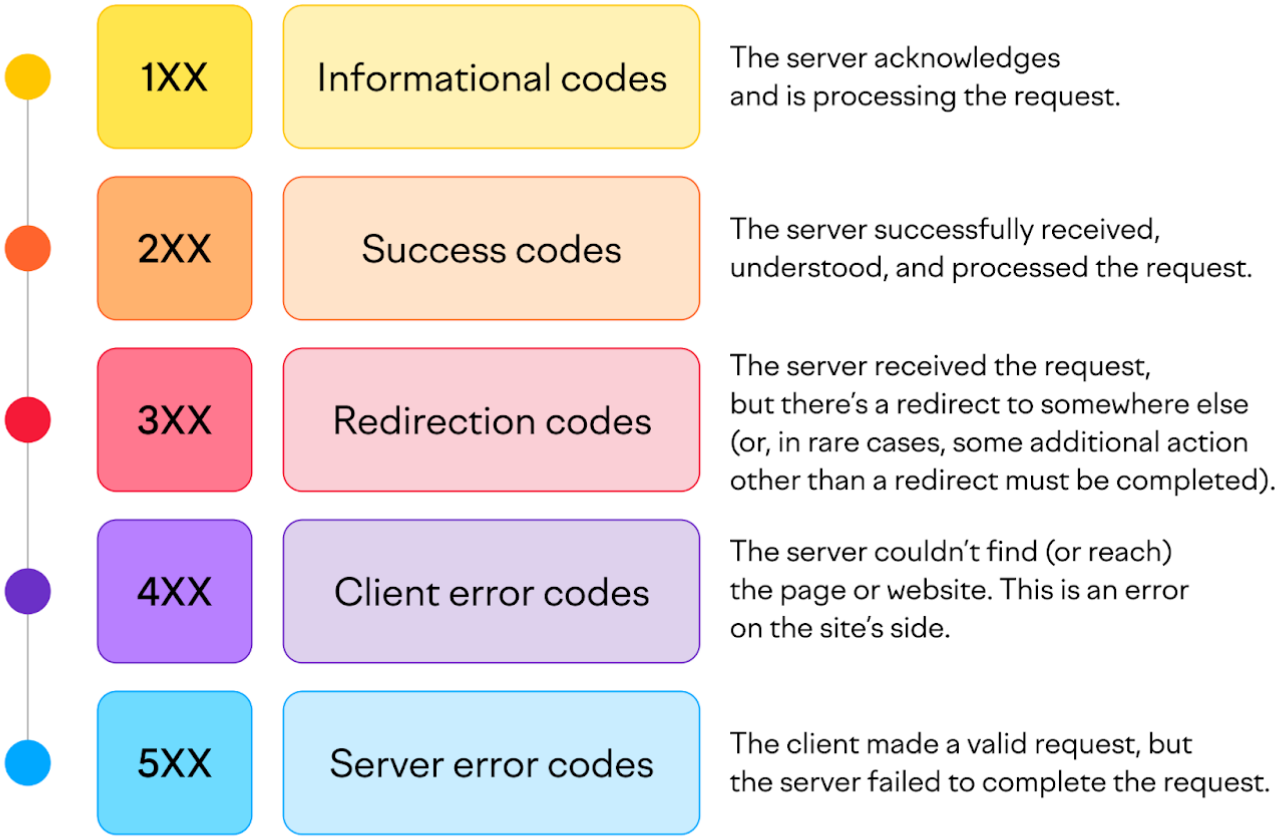
Advantages - HTTP

- Memory usage and CPU usage are low because of fewer simultaneous connections.
- Since there are few TCP connections hence network congestion is less.
- Since handshaking is done at the initial connection stage, then latency is reduced because there is no further need for handshaking for subsequent requests.
- The error can be reported without closing the connection.
- HTTP allows HTTP pipe-lining of requests or responses.

Disadvantages - HTTP

- HTTP requires high power to establish communication and transfer data.
- HTTP is less secure because it does not use any encryption method like HTTPS and use TLS to encrypt regular HTTP requests and response.
- HTTP is not optimized for cellular phones and it is too gabby.
- HTTP does not offer a genuine exchange of data because it is less secure.
- The client does not close the connection until it receives complete data from the server; hence, the server needs to wait for data completion and cannot be available for other clients during this time.

HTTP Response Status Codes



Question

innovate achieve lead

Is it possible to launch DDoS assaults over HTTP?

Remember that because HTTP is a “stateless” protocol, every command executed over it operates independently of every other operation. Each HTTP request opened and terminated a TCP connection according to the original specification. Multiple HTTP requests can now flow over a persistent TCP connection in HTTP 1.1 and later versions of the protocol, which improves resource use. Large-scale HTTP requests are regarded as application layer or layer 7 attacks in the context of DoS or DDoS attacks, and they can be used to mount an attack on a target device.

Case Study

innovate achieve lead

Optimizing Image Delivery on an E-commerce Website using HTTP Techniques

Scenario:

An e-commerce website is experiencing slow loading times, particularly for product pages with large images. This is impacting user experience and conversion rates.

Problem:

The current approach involves sending all product images at their original high resolution, regardless of the user's device or screen size. This leads to large data transfer sizes, especially for mobile users on slower connections.

Solution

innovate achieve lead

Step-1 : Content Negotiation (using Accept header)

a) The server can send a list of supported image formats (e.g., JPEG, WebP) in the response header.
b) The client (browser) can then send an Accept header in the request, indicating its preferred format based on device capabilities and network conditions.
c) The server can select and deliver the most suitable image format based on the negotiation.

Solution (cont)

Step2 : Responsive Images (using srcset and sizes attributes)

a) The website can use the srcset attribute in the image tag to specify different image versions at various resolutions.

b) The sizes attribute can then be used to define the appropriate image based on the viewport size of the user's device.

c) The browser can then automatically choose the most appropriate image based on the screen size, reducing unnecessary data download.

Solution(cont.)

innovate achieve lead

Step 3: Caching (using Cache-Control header)

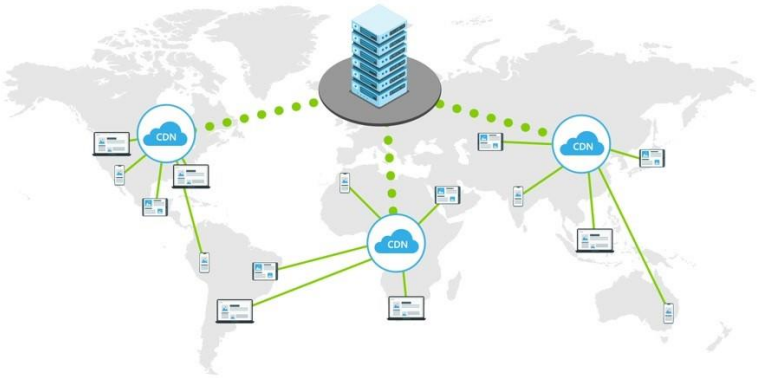
- The server can set caching headers like Cache-Control to instruct browsers and intermediate servers to cache frequently accessed images.
- This reduces the need to download the same image repeatedly, improving subsequent page load times.

Solution (cont.)

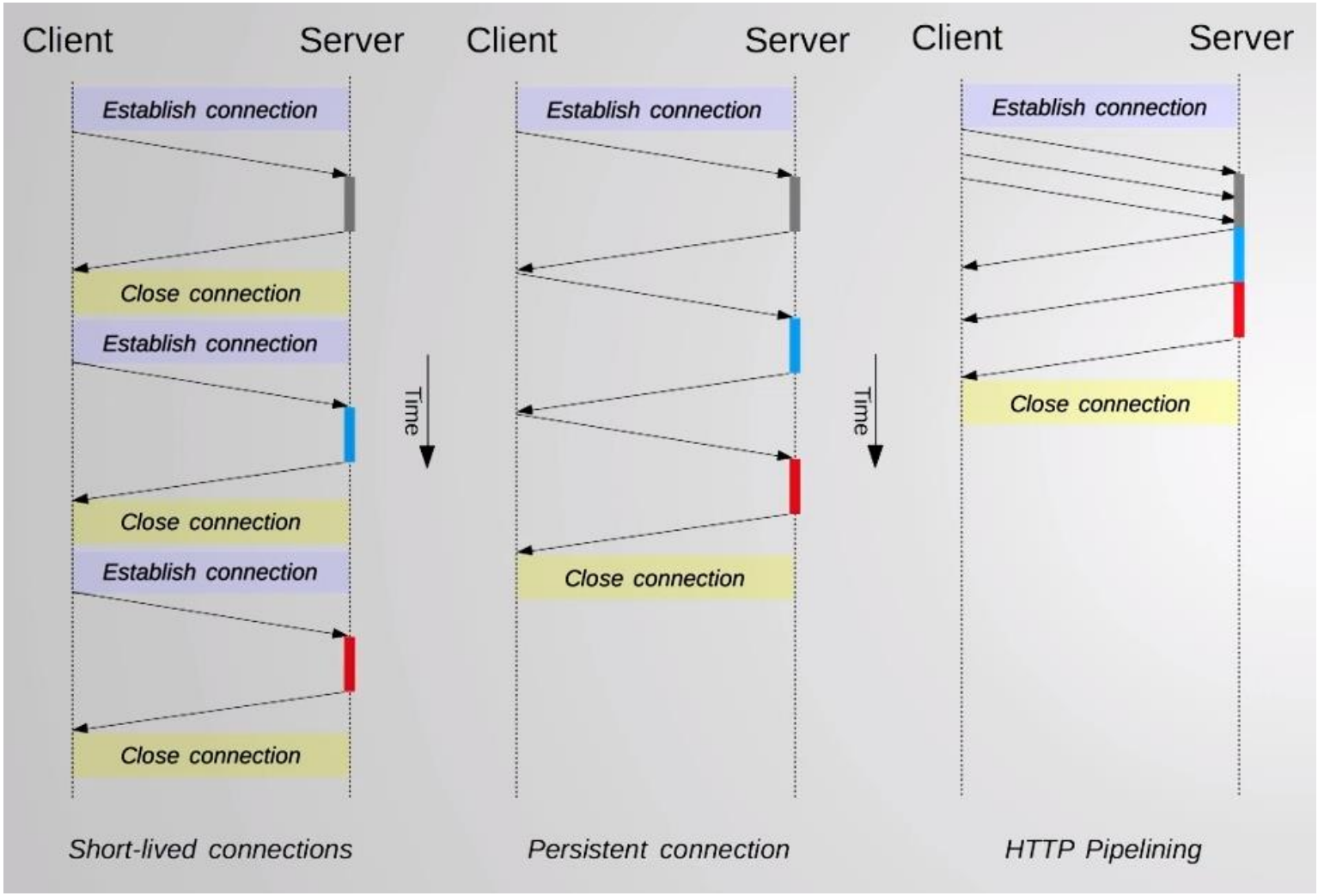
innovate achieve lead

Content Delivery Networks (CDNs)

- The website can utilize a CDN to store and deliver static content, including images, from geographically distributed servers.
- This reduces the distance the data needs to travel, leading to faster loading times for users in different locations.



HTTP/1.1 vs HTTP/2



HTTP/2 protocol

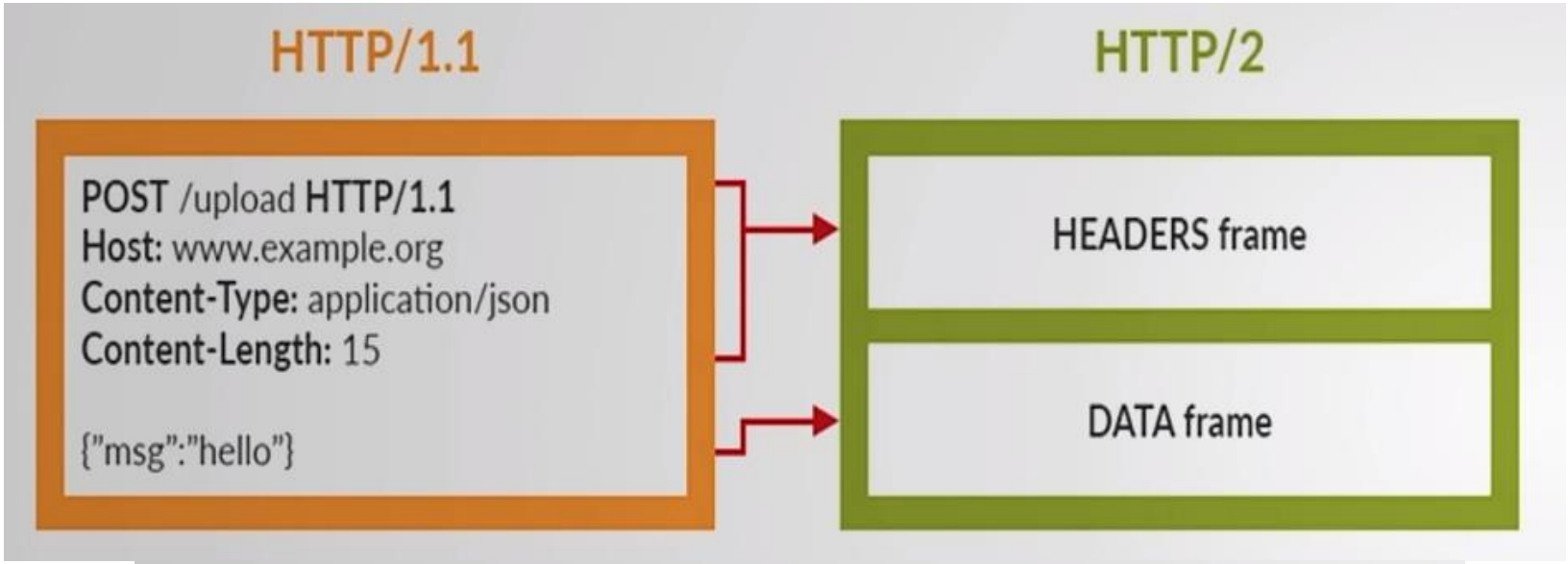
Multiplexing

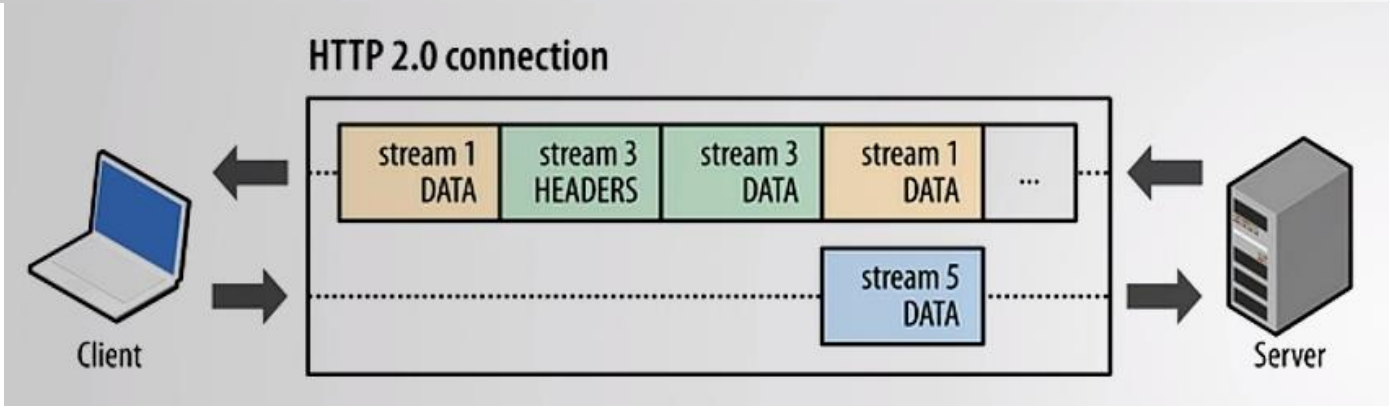
Header Compression

Effective Resource Prioritization

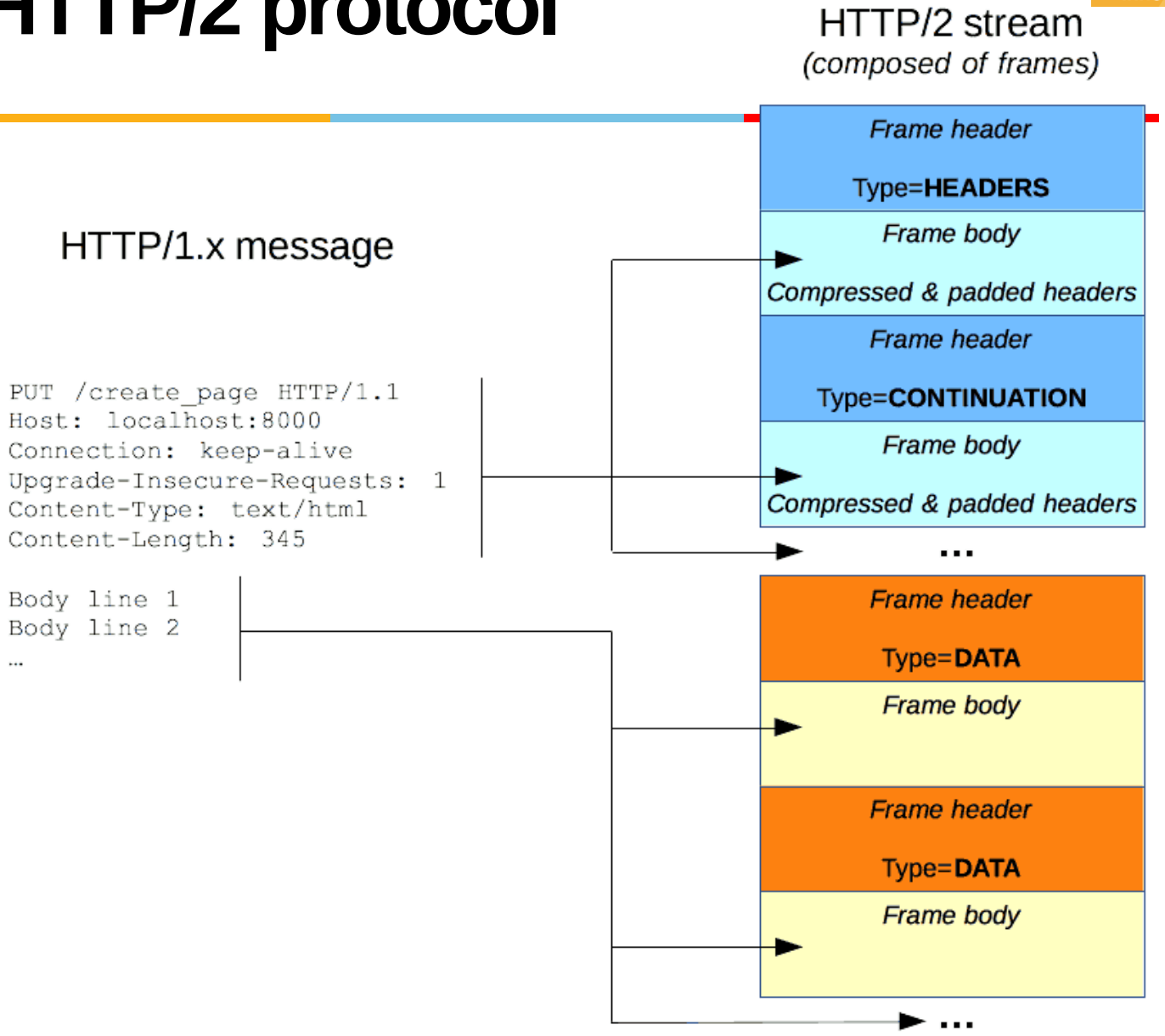
- It's a **binary protocol** rather than a text protocol. It can't be read and created manually. Despite this hurdle, it allows for the implementation of improved optimization techniques.
- It's a **multiplexed protocol**. Parallel requests can be made over the same connection, removing the constraints of the HTTP/1.x protocol.
- It compresses headers. As these are often similar among a set of requests, this removes the duplication and overhead of data transmitted.
- It allows a server to populate data in a client cache through a mechanism called the server push.

HTTP/2 protocol

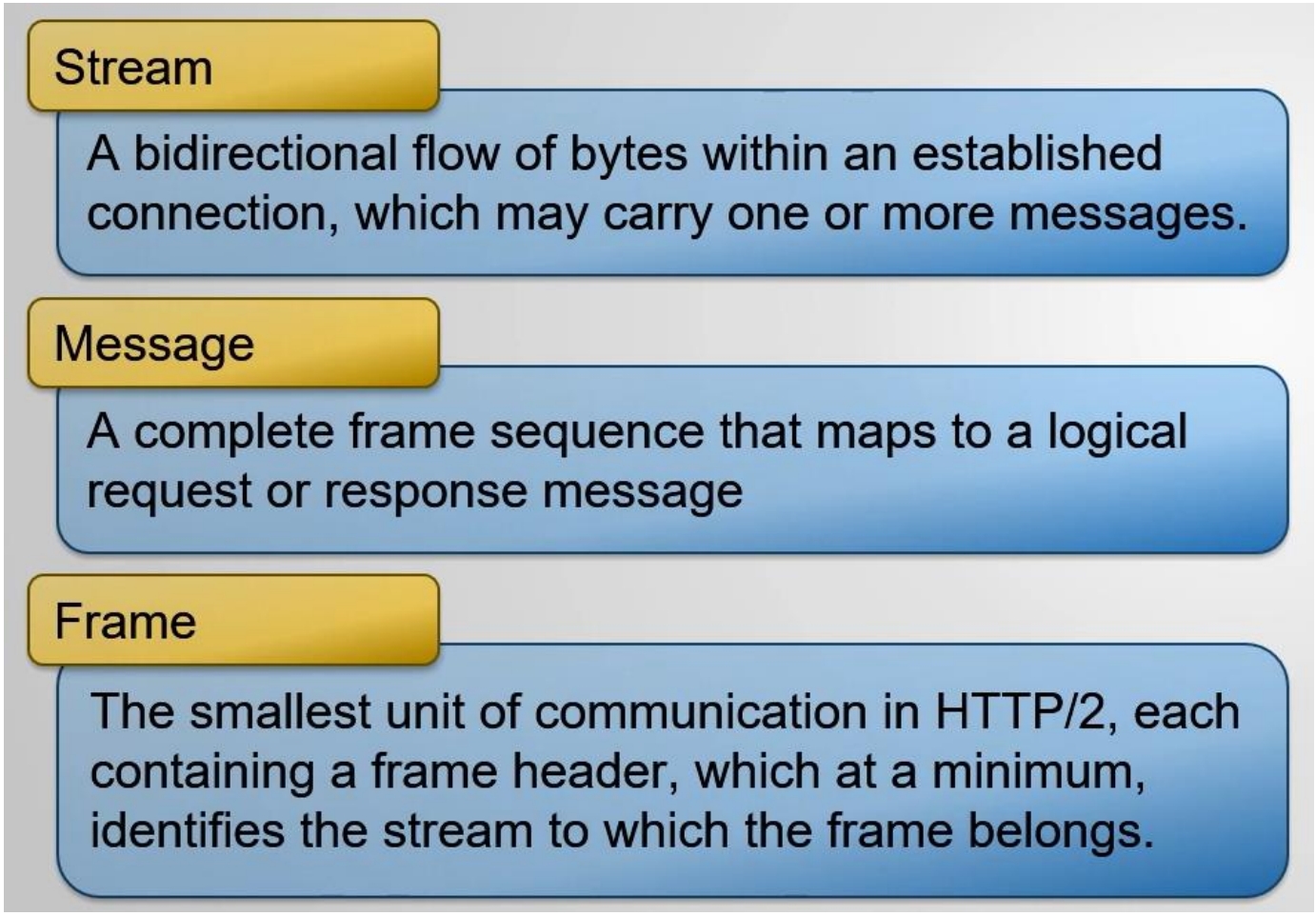




HTTP/2 protocol



HTTP/2 protocol



HTTP/2 protocol

It's a **binary** protocol

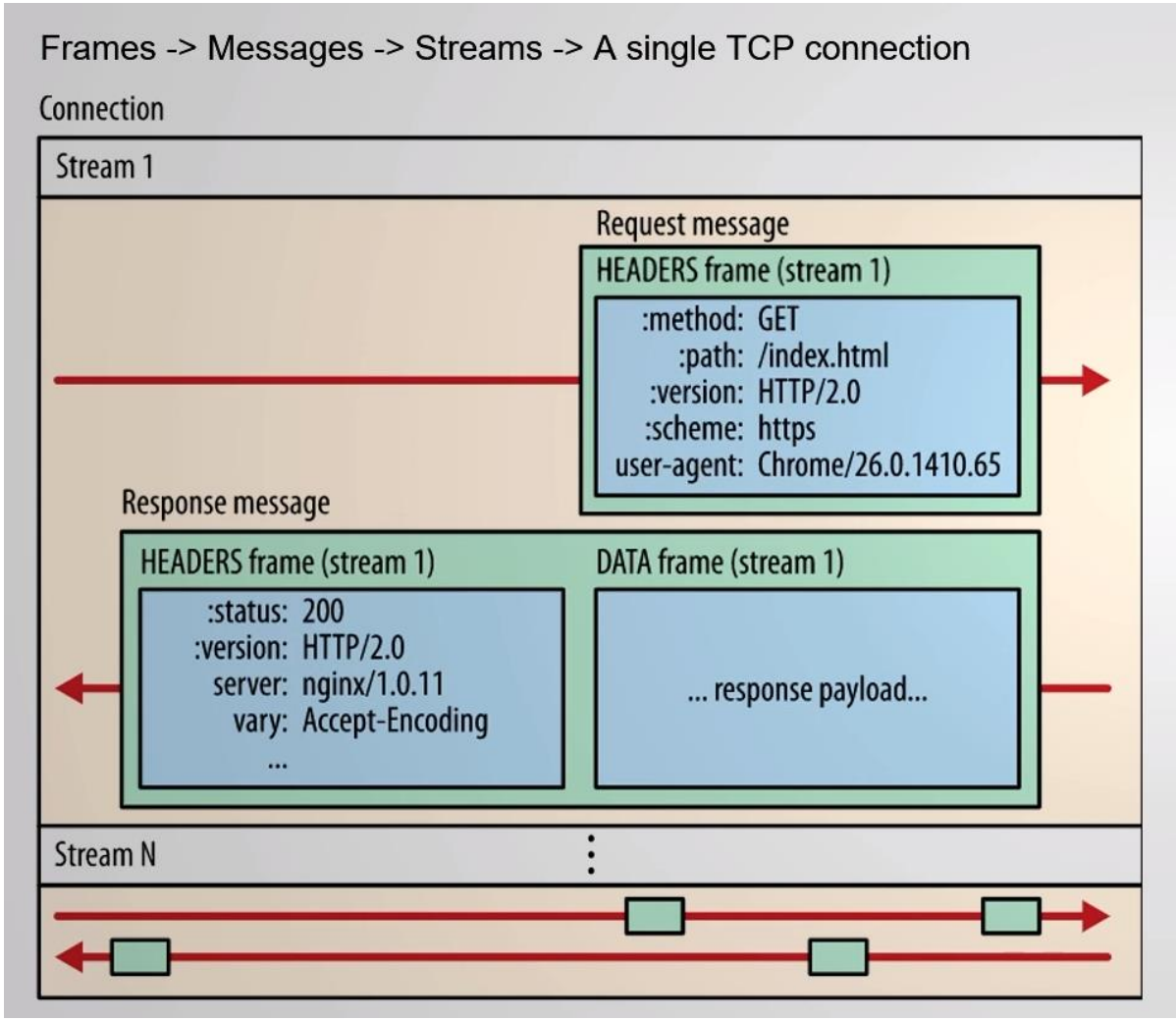
It used header **compression** HPACK to reduce the overhead size

HTTP/2 supports response **prioritization**.

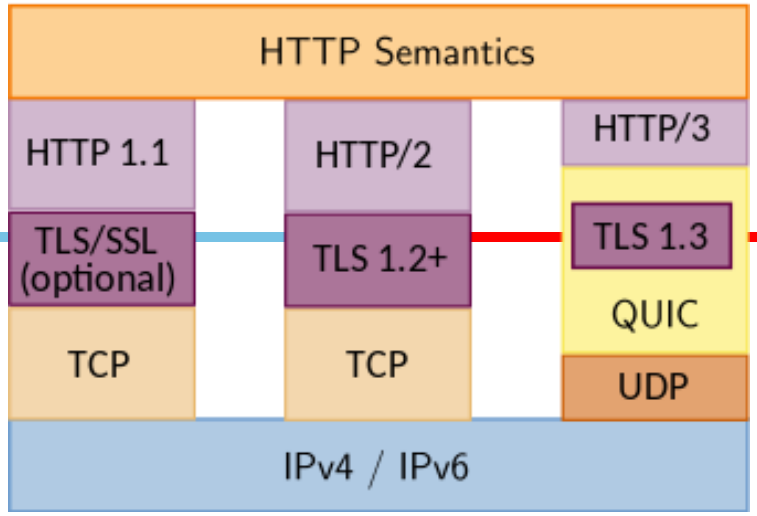
It allows servers to "**push**" responses proactively into client caches instead of waiting for a new request for each resource

It reduces additional **round trip times (RTT),** making your website load faster without any optimization.

HTTP/2 protocol



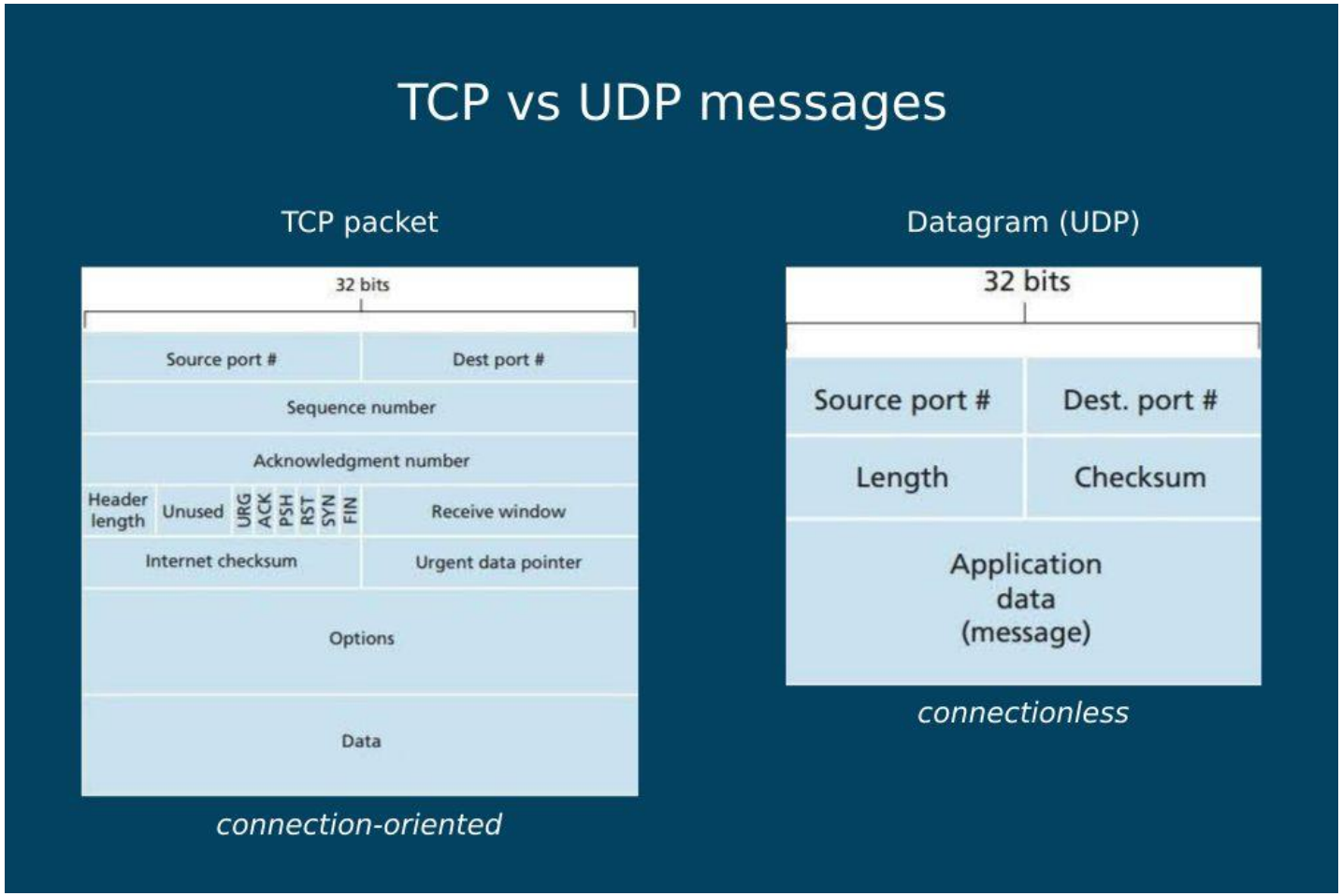
HTTP/3



An important difference in HTTP/3 is that it runs on QUIC, a new transport protocol. QUIC is designed for mobile-heavy Internet usage in which people carry smartphones that constantly switch from one network to another as they move about their day. This was not the case when the first Internet protocols were developed: devices were less portable and did not switch networks very often.

The use of QUIC means that HTTP/3 relies on the User Datagram Protocol (UDP), not the Transmission Control Protocol (TCP). Switching to UDP will enable faster connections and faster user experience when browsing online.

TCP vs UDP


TCP vs UDP messages
TCP packet
32 bits
Source port #
Dest port #
Sequence number
Acknowledgment number
Header length
Unused
URG
ACK
PSH
RST
SYN
FIN
Receive window
Internet checksum
Urgent data pointer
Options
Data
connection-oriented
Datagram (UDP)
32 bits
Source port #
Dest. port #
Length
Checksum
Application data (message)
connectionless

Why do we need QUIC?

QUIC was created to replace TCP with a more flexible transport protocol with fewer performance issues, built-in security, and a faster adoption rate. It needs UDP as a lower-level transport protocol primarily because most devices only support TCP and UDP port numbers.

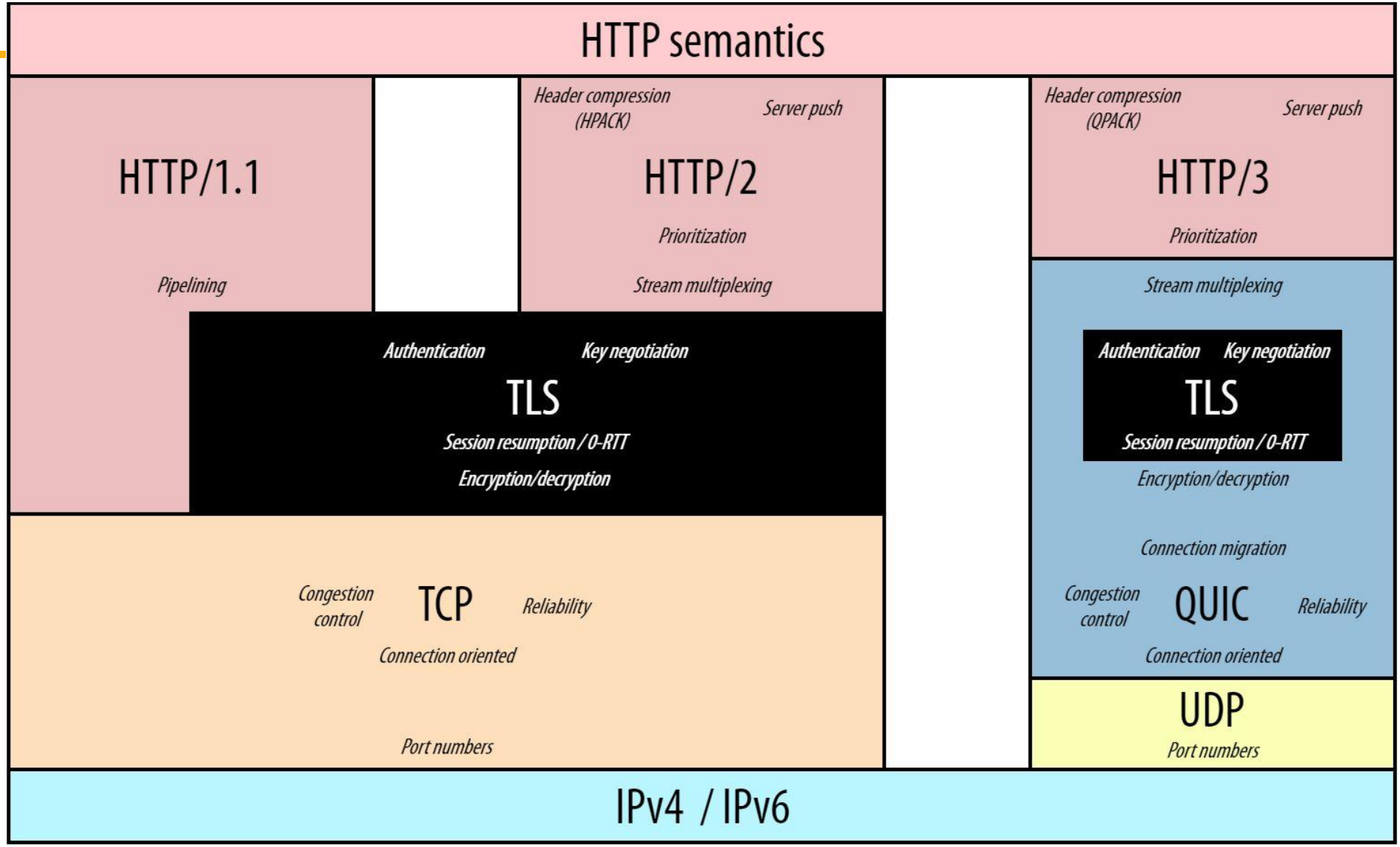
In addition, QUIC leverages UDP's:

- connectionless nature that makes it possible to move multiplexing down to the transport layer and removes TCP's head-of-line blocking issue
- simplicity that allows QUIC to re-implement TCP's reliability and bandwidth management features in its own way

QUIC transport is a unique solution. While it's connectionless at the lower level thanks to the underlying UDP layer, it's connection-oriented at the higher level thanks to its re-implementation of TCP's connection establishment and loss detection features that guarantee delivery. In other words, QUIC merges the advantages of both types of network transport.

Protocol Differences

	TCP	UDP	QUIC
Layer in the TCP/IP model	transport	transport	transport
Place in the TCP/IP model	on top of IPv4 or IPv6	on top of IPv4 or IPv6	on top of UDP
Connection type	connection-oriented	connectionless	connection-oriented
Order of delivery	in-order delivery	out-of-order delivery	out-of-order delivery between streams, in-order delivery within streams
Guarantee of delivery	guaranteed (lost packets are retransmitted)	no guarantee of delivery	guaranteed (lost packets are retransmitted)
Handshake mechanism	non-cryptographic handshake	no handshake	cryptographic handshake
Security	unencrypted	unencrypted	encrypted


HTTP semantics
HTTP/1.1
Pipelining
Header compression
(HPACK)
Server push
HTTP/2
Prioritization
Stream multiplexing
Header compression
(QPACK)
Server push
HTTP/3
Prioritization
Stream multiplexing
Authentication
Key negotiation
TLS
Session resumption / 0-RTT
Encryption/decryption
Authentication
Key negotiation
TLS
Session resumption / 0-RTT
Encryption/decryption
Congestion control
TCP
Reliability
Connection oriented
Port numbers
Connection migration
Congestion control
QUIC
Reliability
Connection oriented
UDP
Port numbers
IPv4 / IPv6

Fetch API

The Fetch API is a modern JavaScript interface for making network requests, specifically designed to be a more streamlined and powerful alternative to the older XMLHttpRequest (XHR) method. Here's a breakdown of what it offers:

Functionality:

- **Fetching resources:** It allows you to fetch various resources across the network, including data from APIs, HTML documents, images, and more.
- **Asynchronous operations:** Like XHR, Fetch utilizes asynchronous operations, meaning your code execution doesn't pause while waiting for the response from the server.

Benefits:

- **Simpler syntax:** Compared to XHR, the Fetch API offers a cleaner and more concise syntax, making it easier to write and understand code.
- **Promise-based:** It uses Promises instead of callbacks to handle the asynchronous nature of network requests, leading to cleaner and more readable code structure.
- **Flexibility:** It provides more flexibility in handling requests and responses, allowing you to specify options like request methods (GET, POST, PUT, etc.), headers, and body content.
- **Integration with advanced features:** The Fetch API seamlessly integrates with advanced features like CORS (Cross-Origin Resource Sharing) and other HTTP concepts.

Fetch API

The Fetch API provides an interface for fetching resources

Provides a generic definition of Request and Response objects

The fetch() method takes one mandatory argument, the path to the resource you want to fetch.

It returns a promise that resolves to the response of that request (successful or not).

You can optionally pass an init options object as a second argument (used to configure req headers for other types of HTTP requests such as PUT, POST, DELETE)

XMLHttpRequest(XHR) vs Fetch

Feature	Fetch API	XHR
Syntax	Cleaner, more concise	More verbose, requires multiple callbacks
Asynchronous Handling	Uses Promises for cleaner asynchronous handling	Uses callbacks, which can lead to nested code and potential race conditions
Error Handling	Uses .catch for error handling	Requires additional logic for error handling
Flexibility	Offers more flexibility in configuring requests	Offers less flexibility compared to Fetch API
Integration	Integrates seamlessly with advanced features like CORS	Requires additional configuration for CORS
Community Adoption	More widely adopted and supported in modern browsers	Less commonly used in modern development

HTTP Polling

HTTP Polling is a technique used in web applications to simulate real-time communication between a client (like your web browser) and a server. It works by having the client repeatedly send requests to the server at regular intervals, asking for updates.

Here's a breakdown of the process:

1. **Client makes a request:** The client sends an HTTP request to the server, typically a GET request, asking for any new data.
2. **Server holds the request:** Instead of sending an immediate response, the server holds onto the request for a specific period (timeout). During this time, the server checks if any new data is available for the client.
3. **Server sends response:** If new data is available, the server sends a response containing the updated information to the client. Otherwise, the server keeps holding the request until the timeout expires.
4. **Client receives response (or timeout):** The client receives the server's response (containing data or indicating no changes) or encounters a timeout, whichever happens first.
5. **Client repeats:** Regardless of the response, the client typically initiates another request after receiving a response or experiencing the timeout, repeating the cycle.

Long Running Transaction

API that performs tasks that could run longer than the request timeout limit.

The server typically replies with a 202 Accepted Response indicating that the request is accepted and is in progress.

So client has to periodically check the server , if the work has been completd.

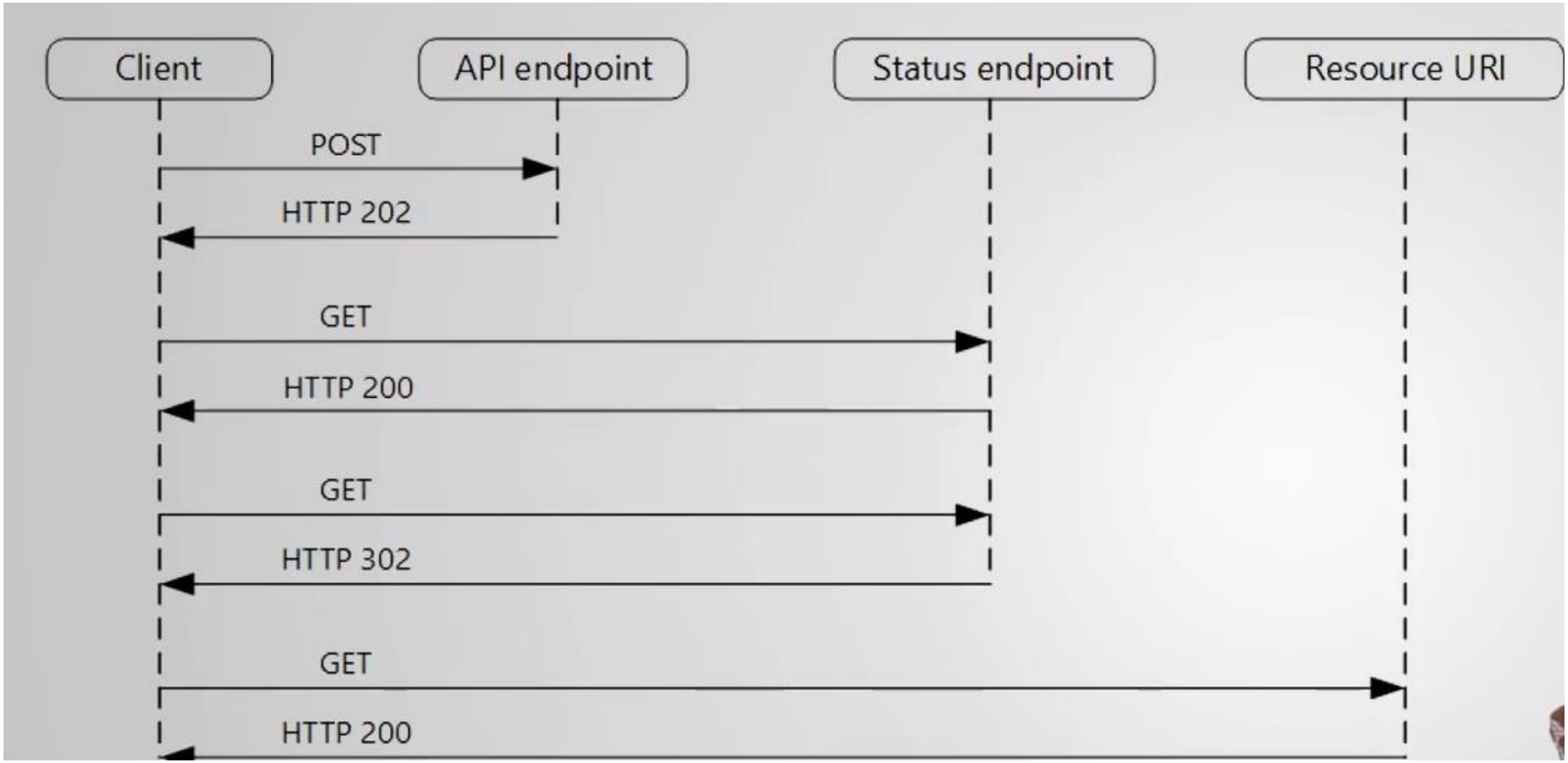
HTTP is a **unidirectional** protocol, which means the client always initiates the communication.

HTTP Polling is a mechanism where the client requests the resource regularly at intervals.

If the resource is available, the server sends the resource as part of the response.

If the resource is not available, the server returns an empty response to the client.

HTTP Polling



HTTP Polling

An **HTTP 202** response should indicate the location and frequency that the client should poll for the response.

- It should have the following additional headers:

Location	A URL the client should poll for a response status.
Retry-After	This header is designed to prevent polling clients from overwhelming the back-end with retries.

Use Cases – HTTP Polling

1. **Social Media Feeds**: Social media platforms use polling to update users' feeds with new posts, comments, or notifications. When a user scrolls down their feed, the client may trigger a new HTTP request to fetch additional posts from the server. Similarly, the client may periodically poll the server for new notifications or updates.
2. **Stock Market Updates**: Financial applications often use HTTP polling to retrieve real-time stock market data. Clients may send periodic requests to the server to fetch the latest stock prices, market trends, or news updates. This allows traders and investors to stay informed about market changes in near-real-time.
3. **Weather Updates**: Weather applications may employ HTTP polling to fetch the latest weather forecasts or updates from a remote server. Clients can periodically poll the server to retrieve current weather conditions, forecasts for the upcoming days, or weather alerts for specific locations.
4. **Online Gaming**: In multiplayer online games, HTTP polling can be used to synchronize game state between players and the game server. Clients may send regular requests to the server to fetch updates on game events, player movements, or changes in the game environment.

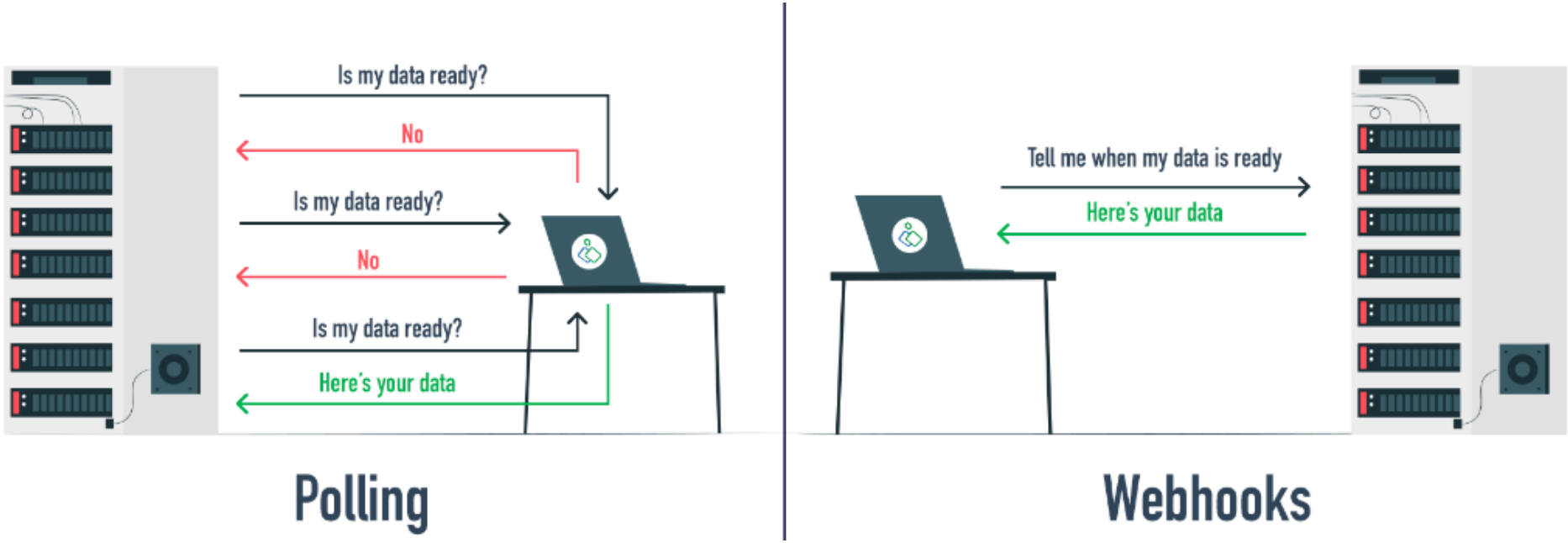
Webhooks

What is a webhook?

A webhook is also known as a “reverse API.” It is a tool that enables one system or application to communicate and deliver real-time notifications about a specific event to another system or application.

Webhooks are a communication method between applications that utilizes **HTTP callbacks** triggered by specific **events**. They allow one application (the **source**) to notify another application (the **destination**) when something significant happens, along with relevant information about the event.

Webhooks vs Polling



Use Cases - Webhooks

1. **Notification Services**: Consider a notification service where users subscribe to receive alerts for certain events, such as new email arrivals, social media mentions, or server status updates. When these events occur, the server triggers a webhook, sending relevant data to the subscribed users' endpoints. This allows users to receive immediate notifications without having to continuously poll the server for updates.
2. **E-commerce Platforms**: In e-commerce, webhooks can be used to automate various processes, such as order fulfillment, inventory management, and customer communication. For example, when a new order is placed or an order status changes, the e-commerce platform can trigger a webhook to notify the warehouse management system to prepare and ship the order.
3. **Integration with Third-Party Services**: Webhooks are commonly used for integrating different software systems and services. For instance, a marketing automation platform may utilize webhooks to notify a CRM system when a new lead is captured or when a lead's status changes. This enables seamless data synchronization and workflow automation between the two systems.

Use Cases - Webhooks

1. **Authentication and Authorization**: Webhooks can also be used for authentication and authorization purposes. For instance, a service provider may use webhooks to notify clients of successful or failed login attempts, allowing clients to take appropriate security measures, such as locking user accounts or triggering multi-factor authentication challenges.
2. **Payment Gateways**: Payment gateways often utilize webhooks to notify merchants of payment-related events, such as successful payments, failed transactions, or chargebacks. Merchants can then automatically update order statuses, send confirmation emails to customers, or initiate refunds based on these webhook notifications.

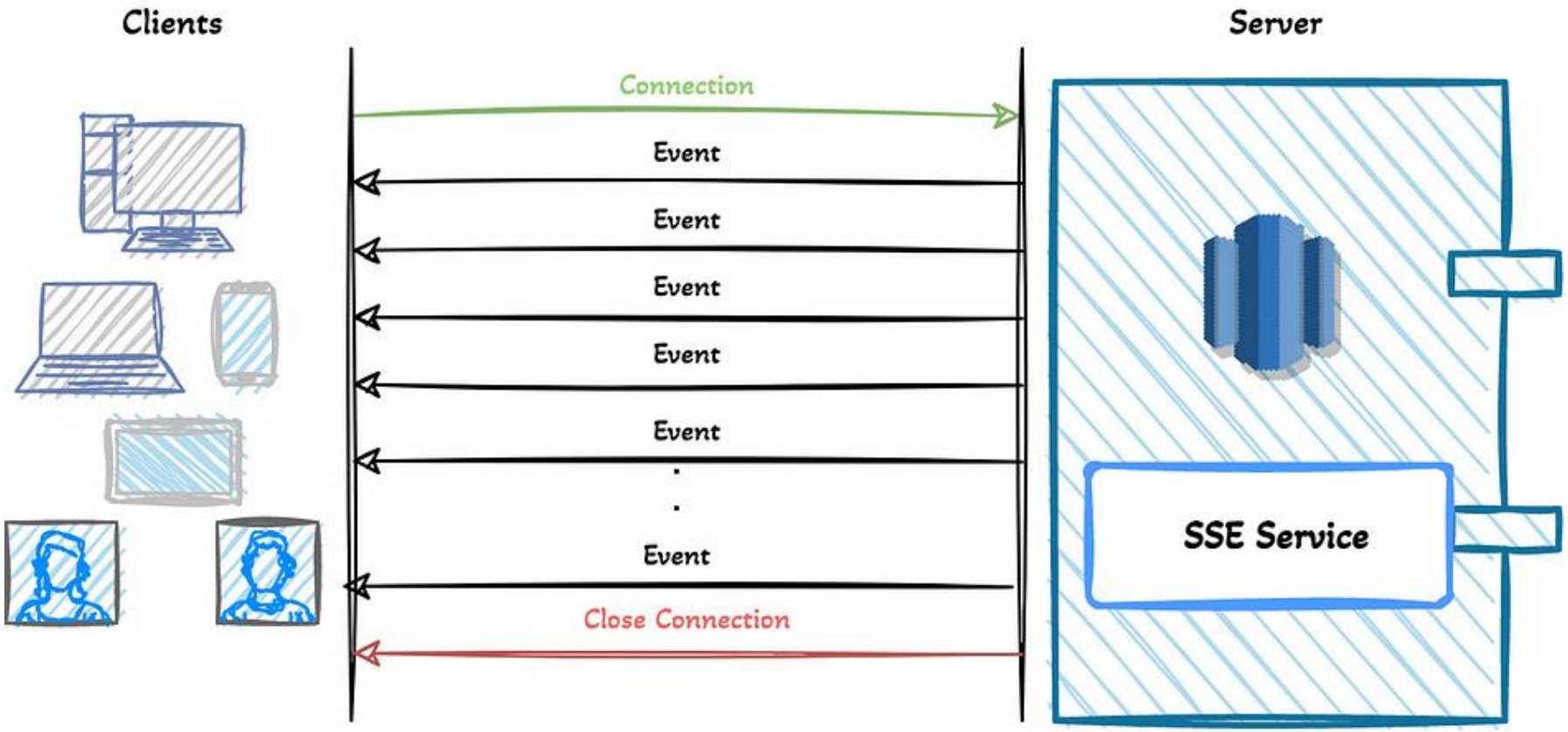
Server-Sent Events

A server-sent event is when a web page automatically gets updates from a server.

This was also possible before, but the web page would have to ask if any updates were available. With server-sent events, the updates come automatically.

Examples: Facebook/Twitter updates, stock price updates, news feeds, sport results, etc.

Server-Sent Events



SSE – How it works?

innovate achieve lead

1. **Client Requests Data**: The client (usually a web browser) initiates a request to the server, typically via an HTTP GET request.
2. **Server Responds**: The server responds to the client's request, but instead of closing the connection immediately, it leaves the connection open.
3. **Server Sends Updates**: As new data becomes available on the server, it sends updates to the client over the same connection. Each update is typically sent as a separate event.
4. **Client Handles Updates**: The client receives these updates and processes them as needed. This could involve updating the UI, displaying new content, or executing JavaScript code in response to the received events.
5. **Connection Persistence**: The connection remains open until either the client or the server decides to close it. This allows for continuous real-time communication between the server and the client.

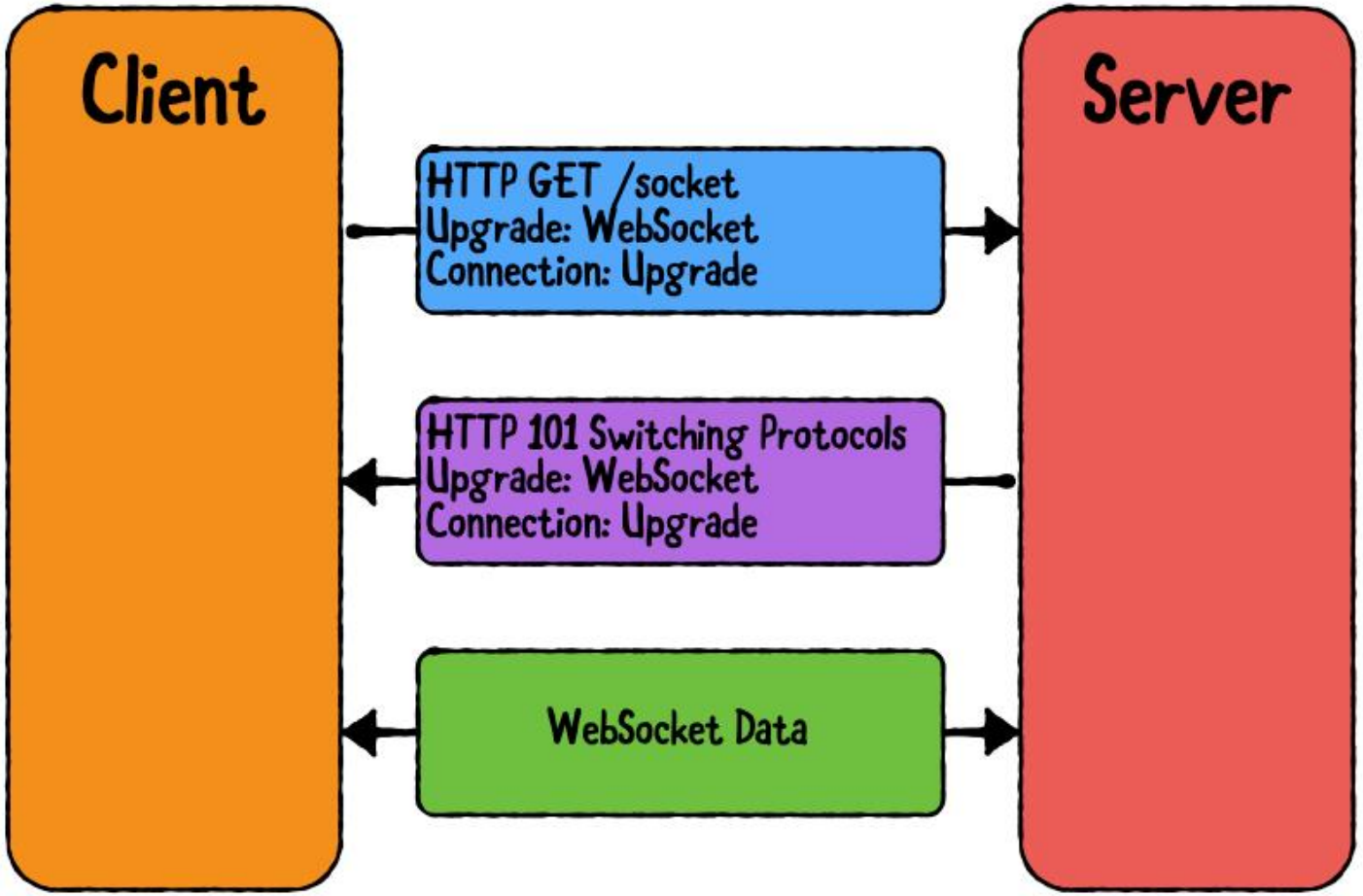
Server-Sent Events are often used for applications that require **real-time updates**, such as chat applications, live sports scores, live dashboards, stock tickers, or any other scenario where immediate data updates are necessary.

Web Sockets

WebSockets are a communication protocol that enables **real-time, two-way communication** between a client (like a web browser) and a server. This means that data can be exchanged in both directions simultaneously, unlike traditional HTTP which follows a request-response model.

- **Full-duplex communication:** Both the client and server can send and receive data at the same time.
- **Persistent connection:** The connection remains open until it's closed by either party, allowing for continuous data exchange without needing to re-establish a connection for each message.
- **Lower overhead:** Compared to other methods like polling, WebSockets have lower overhead due to the persistent connection and efficient data transfer format.
- **Use cases:** WebSockets are commonly used in applications requiring real-time updates, such as:
  - **Chat applications:** Messages are sent and received instantly between users.
  - **Multiplayer games:** Players' actions are synchronized in real-time.
  - **Live streaming:** Data like stock prices or sports scores can be updated continuously.

WebSockets


Client
Server
HTTP GET /socket
Upgrade: WebSocket
Connection: Upgrade
HTTP 101 Switching Protocols
Upgrade: WebSocket
Connection: Upgrade
WebSocket Data

Review Questions

1. Explain the fundamental differences between HTTP/1.1 and HTTP/2, focusing on connection management, performance optimization, and how these differences impact HTTP requests and responses.
2. Compare and contrast the usage of HTTP Polling, Server Sent Events (SSE), and Websockets for real-time communication in web applications, considering their trade-offs.
3. Discuss the significance of HTTP methods beyond GET and POST, including PUT, DELETE, PATCH, and OPTIONS, explaining their functionalities and providing practical scenarios for their use.
4. Explain the role of common HTTP headers like Content-Type, Accept, User-Agent, and Cache-Control in the request-response cycle, providing examples of their impact on client and server behavior.
5. Describe the structure of an HTTP request and response, including request line, headers, and body, and explain how these components contribute to client-server communication.
6. Explain the concept of AJAX and how it facilitates asynchronous communication in web applications. Discuss strategies for optimizing the performance of AJAX requests, considering factors like caching, compression, and reducing round-trip times.