Introduction to HTTP

Topics

HTTP

HTTP/1.x

HTTP/2

HTTP/3

HTTP Request/Response Messages

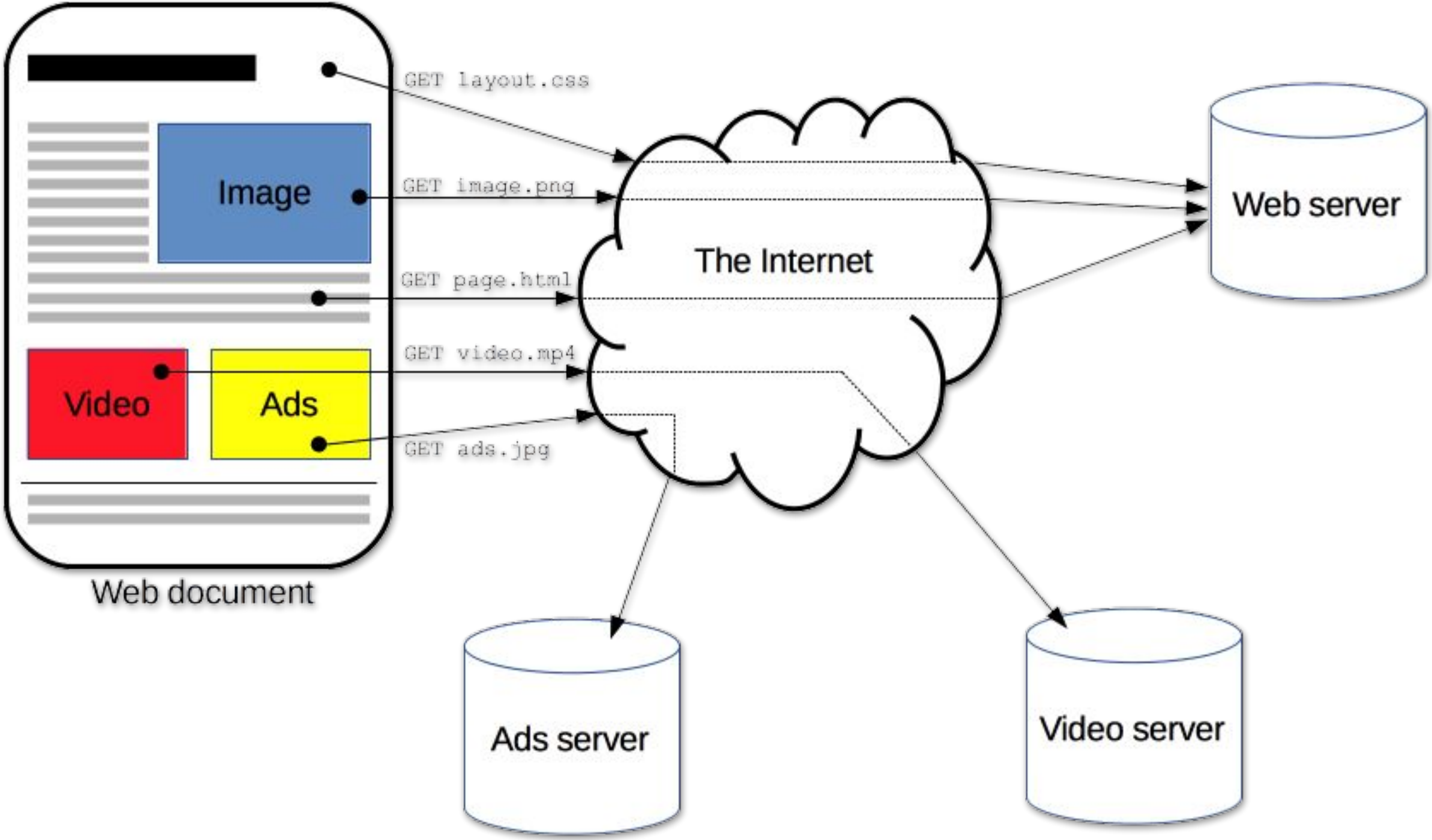
HTTP

A **protocol** for fetching resources such as HTML documents

Foundation of any data exchange on the Web

Client-Server protocol, which means requests are initiated by the recipient, usually the Web browser

A complete document is reconstructed from the different sub-documents fetched, for instance, **text**, **layout description**, **images**, **videos**, **scripts**, and more

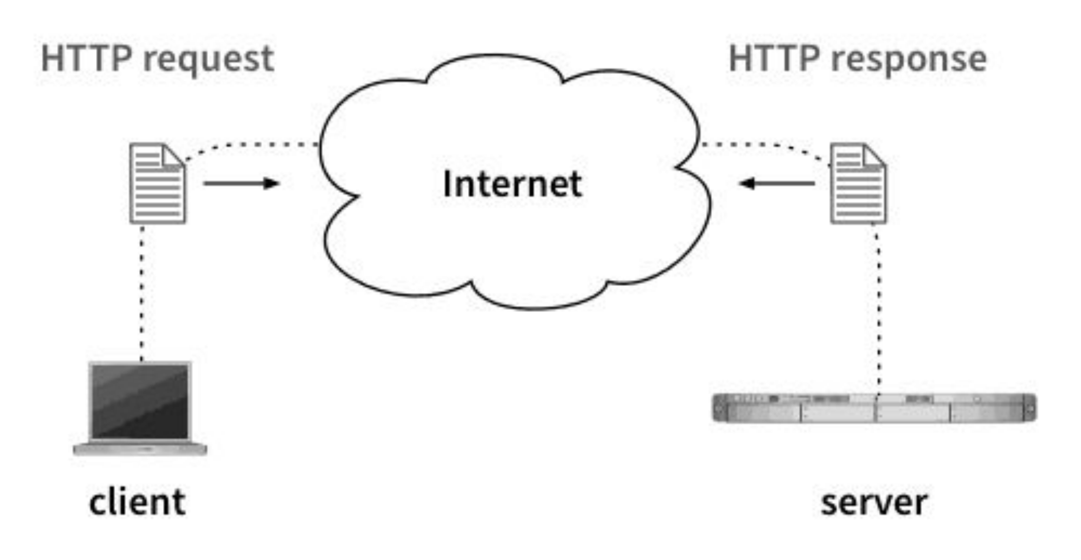

GET layout.css
Image
GET image.png
GET page.html
The Internet
Web server
GET video.mp4
Video
Ads
GET ads.jpg
Web document
Ads server
Video server

HTTP Request/Response

Clients and servers communicate by **exchanging individual messages** (as opposed to a stream of data)

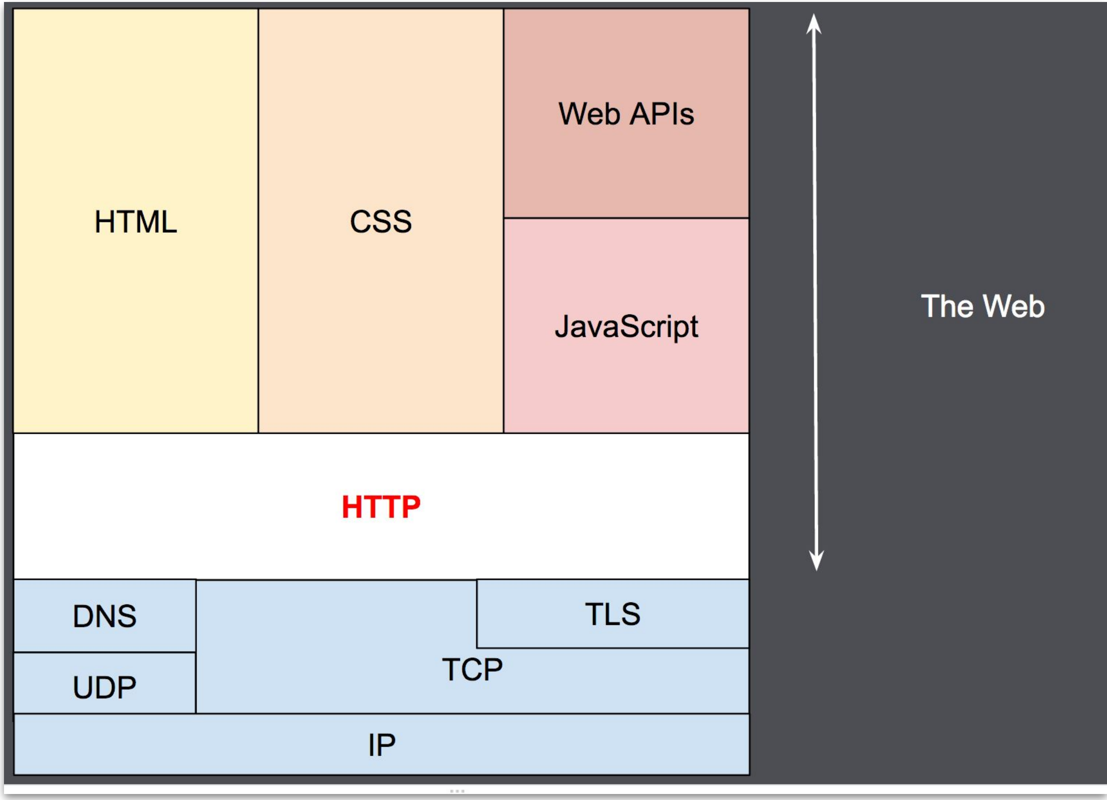
The **messages sent by the client**, usually a Web browser, are called **requests**

The **messages sent by the server** as an answer are called **responses**



HTTP Protocol

HTTP is an application layer protocol that is sent over TCP, or over a TLS-encrypted TCP connection



HTML

CSS

Web APIs

JavaScript

The Web

HTTP

DNS

UDP

TLS

TCP

IP

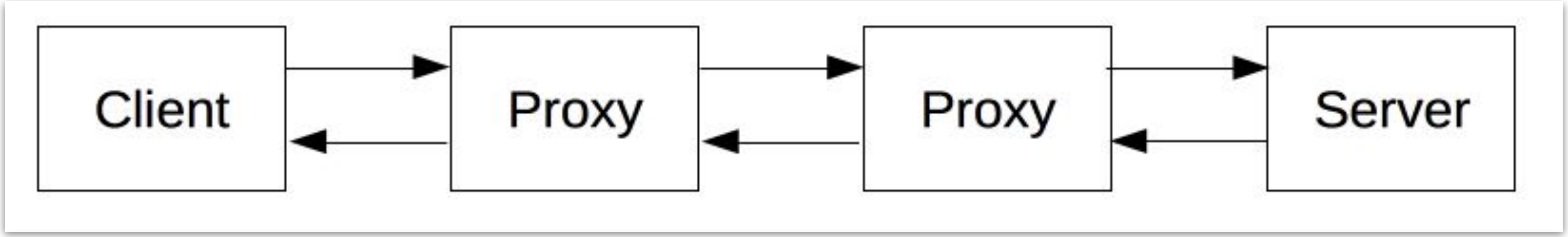
Components of HTTP-based Systems

Client/User-Agent

any tool that acts on behalf of the user

sends HTTP requests

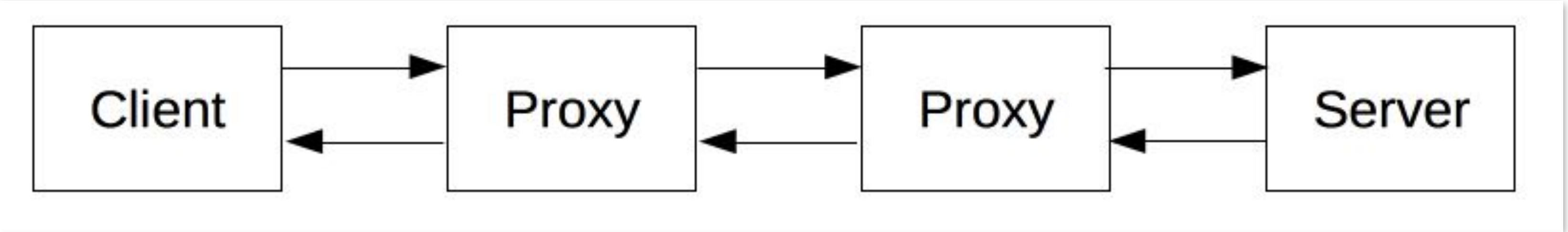
most of the time the user-agent is a Web browser



Components of HTTP-based Systems

Web Server

handles client requests and provides an answer called the response

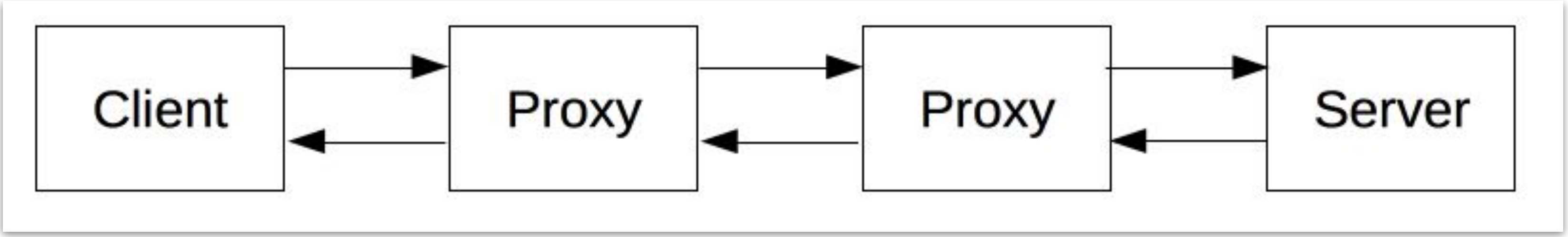


Components of HTTP-based Systems

Proxies

sit between the client and the server,

perform functions such as **caching**, **filtering, load balancing**, **authentication**, and **logging**



HTTP Statelessness

HTTP is Stateless

there is no link between two requests being successively carried out

However, you can create **sessions** using HTTP Cookies which allows **sharing the same context, or the same state** between requests

HTTP Request/Response Flow

When a client wants to communicate with a server it performs the following steps

Step-1: Open a TCP Connection

The TCP connection is used to send a request(s) and receive a response

The client may open a

new connection,

reuse an existing connection, or

open multiple TCP connections to the servers

HTTP Request/Response Flow

Step-2: Send an HTTP Message

```
GET / HTTP/1.1

Host: developer.mozilla.org

Accept-Language: fr
```

HTTP Request/Response Flow

Step-3: Read the Response sent by the Server

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```

```
<!DOCTYPE html... (here come the 29769
bytes of the requested web page)
```

HTTP Request/Response Flow

Step-4: Close or reuse the connection for further requests

Evolution of HTTP

HTTP has gone through many changes

- `HTTP/0.9`
- `HTTP/1.0`
- `HTTP/1.1`
- `HTTP/2`
- `HTTP/3`

HTTP/0.9 – The one-line protocol

Requests consisted of a single line

```
GET /mypage.html
```

The response only consisted of the file itself

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

Characteristics of `HTTP/0.9`

Client request is a single **`ASCII`** character string

Client request is terminated by a carriage return (**`CRLF`**)

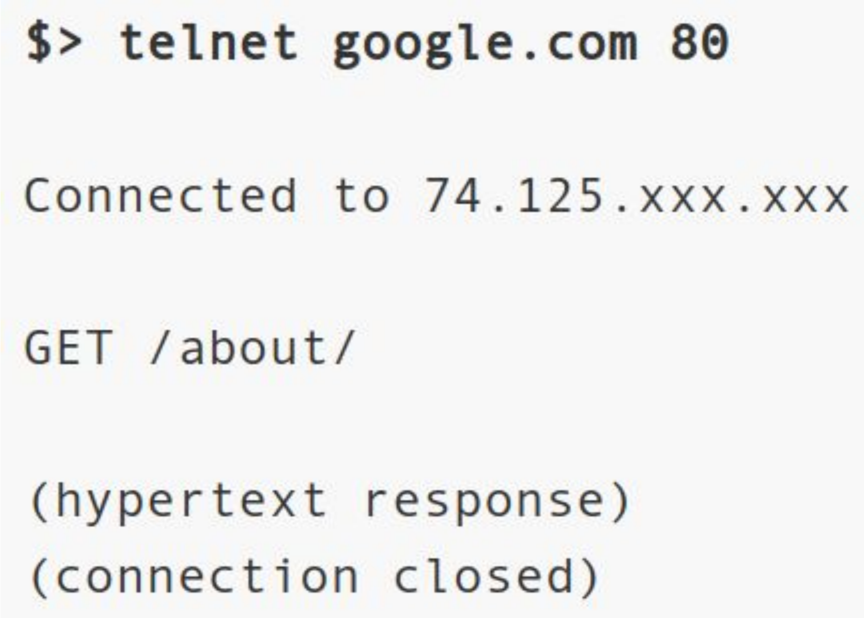
Server response is an **`ASCII`** character stream

Server response is a hypertext markup language (**`HTML`**)

Connection is terminated after the document transfer is complete

HTTP/0.9

Example



```
$> telnet google.com 80

Connected to 74.125.xxx.xxx

GET /about/

(hypertext response)
(connection closed)
```

Question

Can you identify some of the limitations of the `HTTP/0.9` protocol ?

`HTTP/0.9`: Limitations

Could not serve other documents than hypertext documents

It has **`GET`** request method only

Unable to provide metadata about the request and the response

Unable to negotiate content

`HTTP/1.0:` Building extensibility

HTTP Working Group (HTTP-WG) published **RFC 1945**, which **documented** the "**common usage**" of the many **`HTTP/1.0`** implementations found in the internet

`HTTP/1.0`: Features

Versioning information is now sent within each request

- **`HTTP/1.0`** is appended to the **`GET`** line

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

HTTP/1.0: Features

Request and response **headers** were **ASCII encoded**

With the help of the new **HTTP headers**, the **ability to transmit other documents than plain HTML files** has been added

using the **Content-Type** header

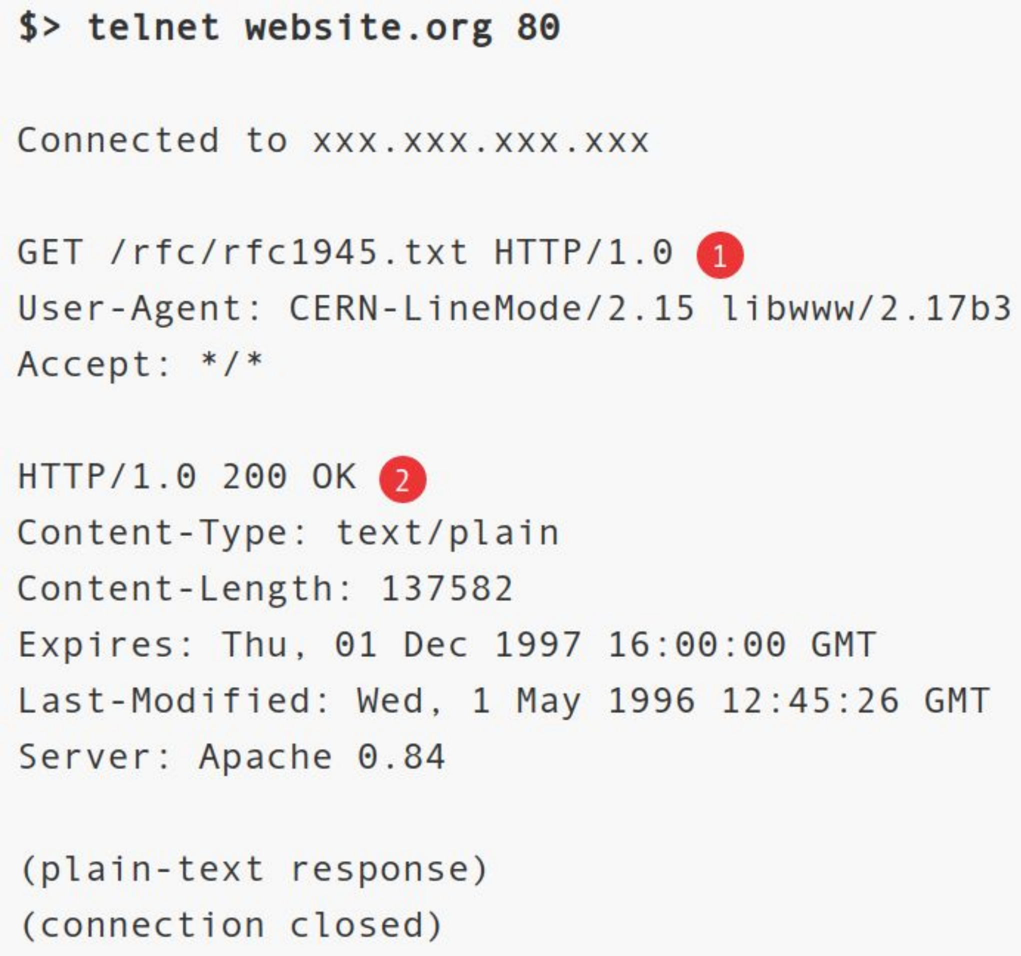
In addition to media type negotiation it included capabilities such as content encoding, character set support, multi-part types, authorization, caching, proxy behaviors, date formats, and more

The connection between server and client is closed after every request

HTTP/1.0 Example

1. **Request line** with HTTP version number, followed by **request headers**

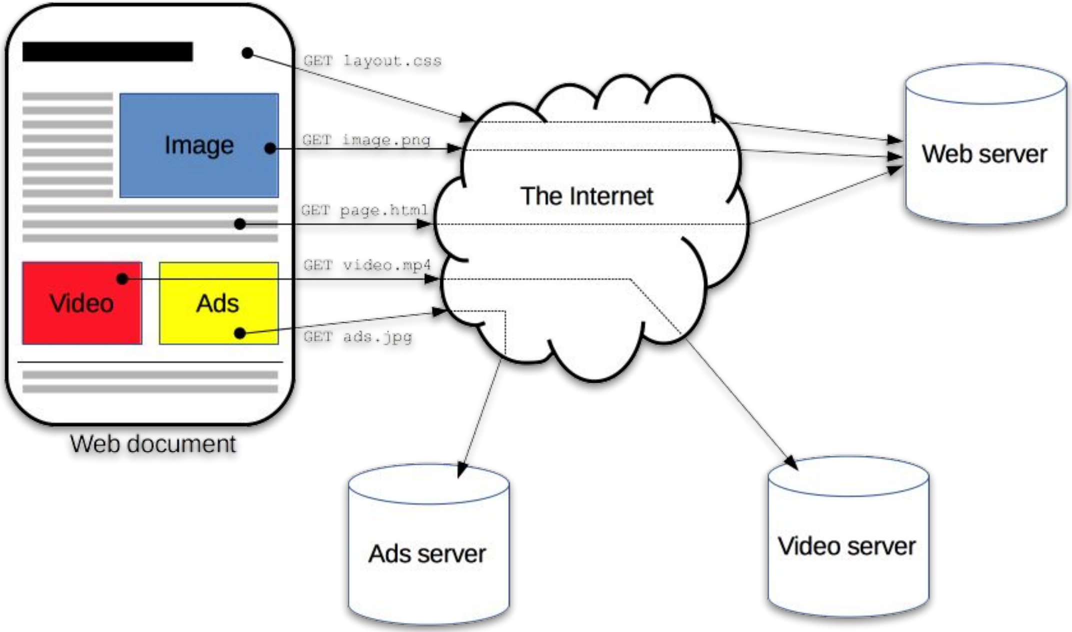
2. **Response status**, followed by **response headers**



```
$> telnet website.org 80

Connected to xxx.xxx.xxx.xxx

GET /rfc/rfc1945.txt HTTP/1.0
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
Accept: */*

HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 137582
Expires: Thu, 01 Dec 1997 16:00:00 GMT
Last-Modified: Wed, 1 May 1996 12:45:26 GMT
Server: Apache 0.84

(plain-text response)
(connection closed)
```

Question

What do you think are **the limitations of the `HTTP/1.0` protocol ?**



HTTP/1.0: Limitations

Requiring a new TCP connection per request

imposes a significant performance penalty

HTTP/1.1: Standardized protocol

The official **HTTP/1.1** standard is defined in **RFC 2616**

The **HTTP/1.1** standard resolved a lot of the protocol ambiguities found in earlier versions

HTTP/1.1 by default leaves the connection open

HTTP/1.1

It introduced a number of critical performance optimizations:

Keepalive Connections

A connection can be reused, saving the time to reopen it numerous times to display the resources embedded into the single original document retrieved

Chunked Encoding Transfers

Chunked responses are now also supported

HTTP/1.1

It introduced a number of critical performance optimizations:

Byte-range Requests

Request 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

HTTP/1.1

It introduced a number of critical performance optimizations:

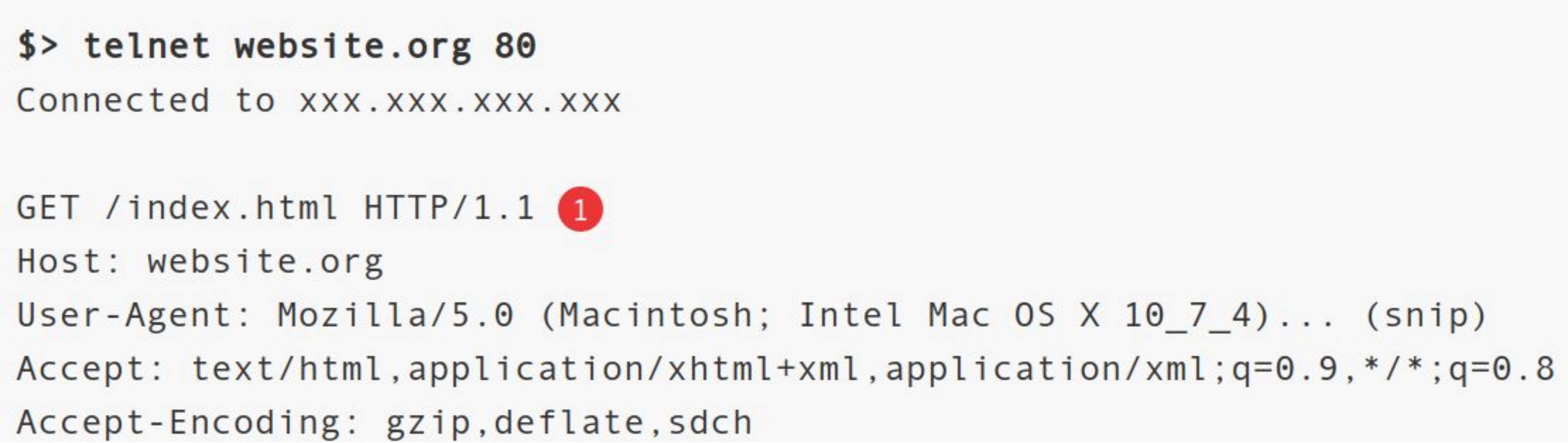
Additional cache control mechanisms have been introduced

Content negotiation, including language, encoding, or type, has been introduced

The addition of the **Host header** allowed to host different domains at the same IP address (allowing server colocation)

HTTP/1.1: Example

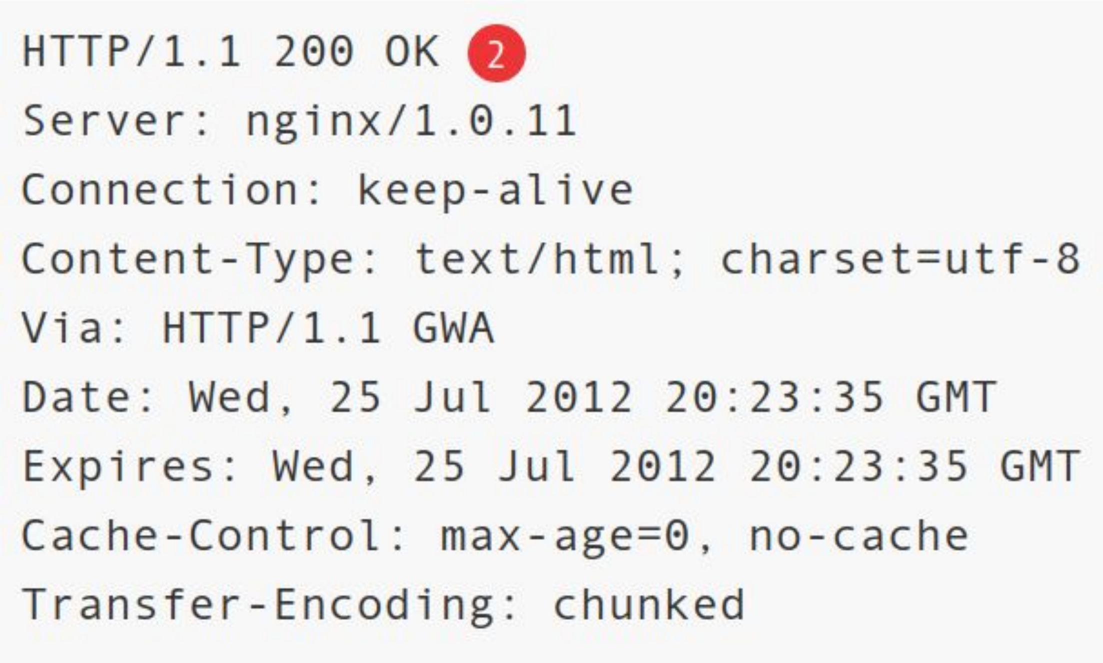
1 Request for HTML file, with encoding metadata



```
$> telnet website.org 80
Connected to xxx.xxx.xxx.xxx

GET /index.html HTTP/1.1 1
Host: website.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4)... (snip)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,sdch
```

HTTP/1.1: Example

2 Chunked response for original HTML request

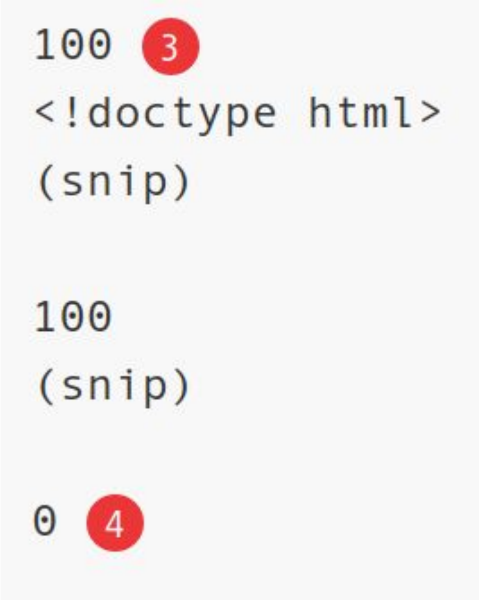


```
HTTP/1.1 200 OK
Server: nginx/1.0.11
Connection: keep-alive
Content-Type: text/html; charset=utf-8
Via: HTTP/1.1 GWA
Date: Wed, 25 Jul 2012 20:23:35 GMT
Expires: Wed, 25 Jul 2012 20:23:35 GMT
Cache-Control: max-age=0, no-cache
Transfer-Encoding: chunked
```

HTTP/1.1: Example

③ Number of octets in the chunk expressed as an ASCII hexadecimal number

④ End of chunked stream response

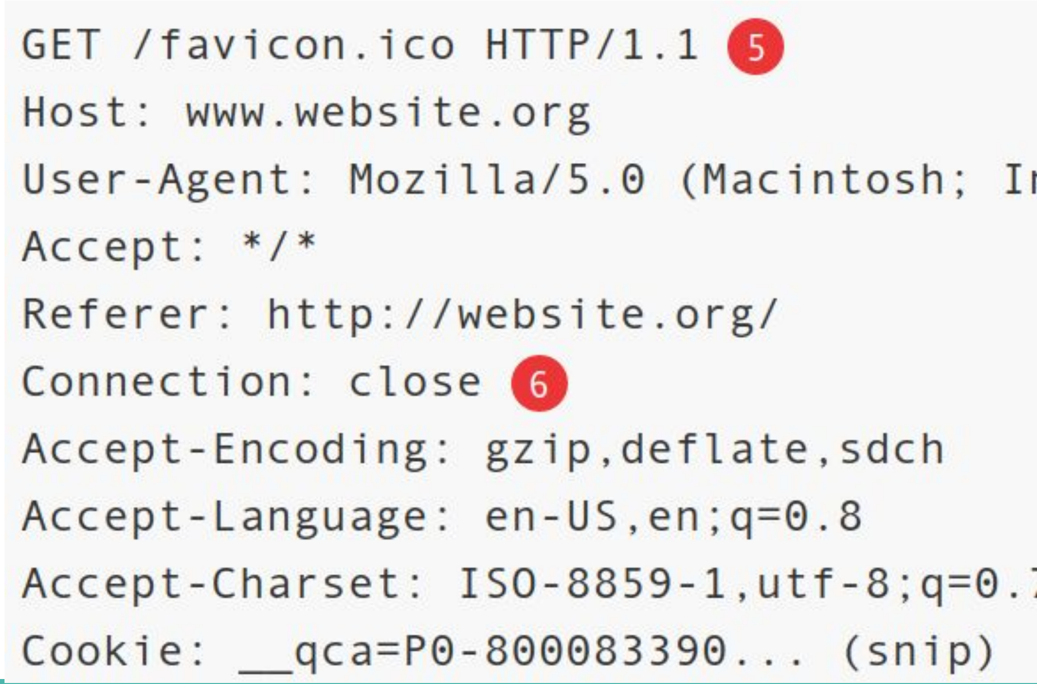


```
100 ③
<!doctype html>
(snip)

100
(snip)

0 ④
```

HTTP/1.1: Example

5 Request for an icon file made on same TCP connection

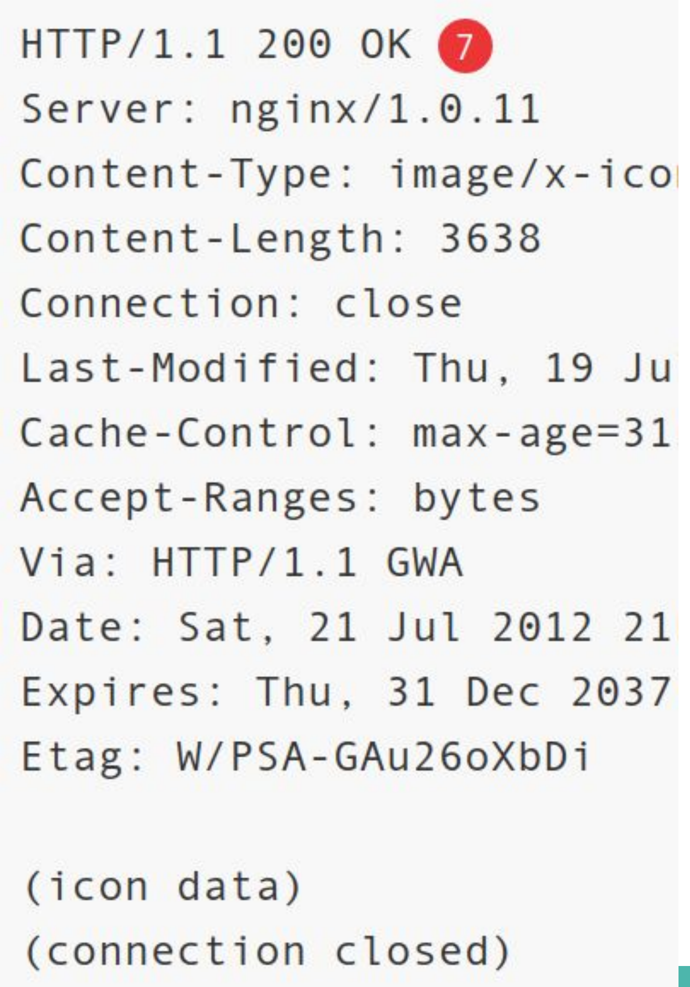
6 Inform server that the connection will not be reused



```
GET /favicon.ico HTTP/1.1 5
Host: www.website.org
User-Agent: Mozilla/5.0 (Macintosh; I
Accept: */*
Referer: http://website.org/
Connection: close 6
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.
Cookie: __qca=P0-800083390... (snip)
```

HTTP/1.1: Example

7 Icon response, followed by connection close



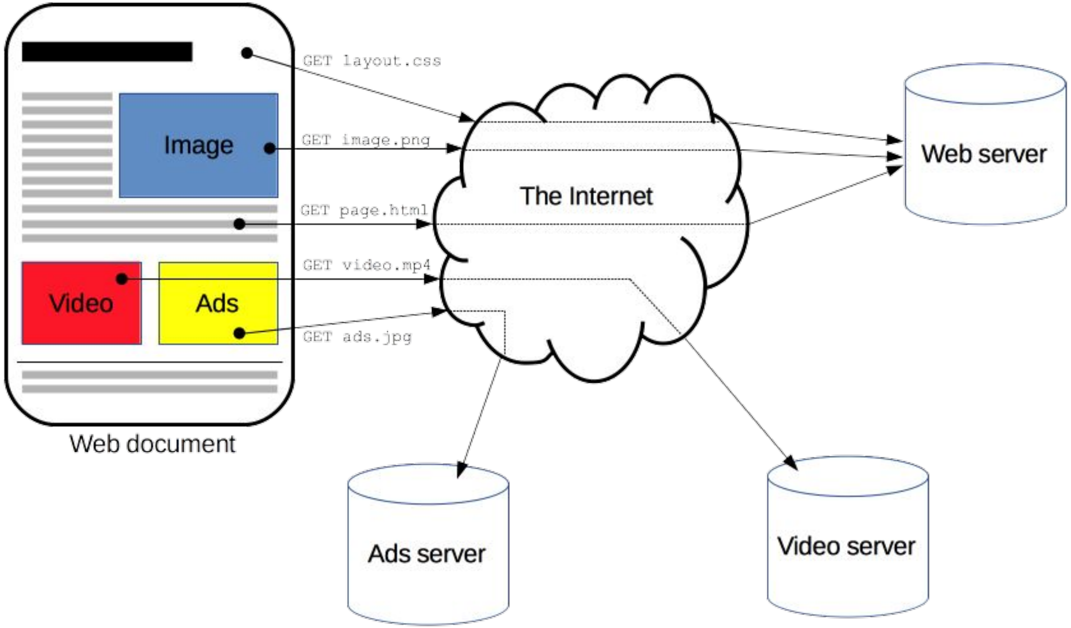
```
HTTP/1.1 200 OK 7
Server: nginx/1.0.11
Content-Type: image/x-ico
Content-Length: 3638
Connection: close
Last-Modified: Thu, 19 Ju
Cache-Control: max-age=31
Accept-Ranges: bytes
Via: HTTP/1.1 GWA
Date: Sat, 21 Jul 2012 21
Expires: Thu, 31 Dec 2037
Etag: W/PSA-GAu26oXbDi

(icon data)
(connection closed)
```

Question

What is the name of the header part that is added on **`HTTP/1.1`** version to improve the limitation of **`HTTP/1.0`** (requiring a new TCP connection for each request)

Question

What do you think are **the limitations of the `HTTP/1.X` protocol ?**



Limitations of `HTTP/1.x`

Clients need to use **multiple connections to achieve concurrency** and **reduce latency**

Does not compress request and response **headers**, causing unnecessary network traffic

Does not allow effective **resource prioritization**, resulting in poor use of the underlying TCP connection

SPDY

SPDY was an experimental protocol, developed at Google

Its primary goal was to try to **reduce the load latency of web pages** by addressing some of the well-known performance limitations of **`HTTP/1.1`**

SPDY

The **specific** project **goals** were the following

- Target a **50% reduction in page load time (PLT)**
- Avoid the need for any changes to content by website authors
- Minimize deployment complexity, avoid changes in network infrastructure
- Develop this new protocol in partnership with the open-source community
- Gather real performance data to (in)validate the experimental protocol

SPDY

SPDY in lab condition has shown **55% reduction in page load time**

As a result SPDY was supported in Chrome, Firefox, and Opera, and a rapidly growing number of sites, both large (e.g., Google, Twitter, Facebook) and small

In effect, SPDY was on track to become a de facto standard through growing industry adoption

SPDY and HTTP/2

Observing the trend, the **HTTP Working Group (HTTP-WG)** launched a new effort

to take the lessons learned from SPDY,

to build and improve on them, and

to deliver an official "**HTTP/2**" standard

HTTP/2

HTTP/2 is a protocol designed for **low-latency transport of content** over the World Wide Web

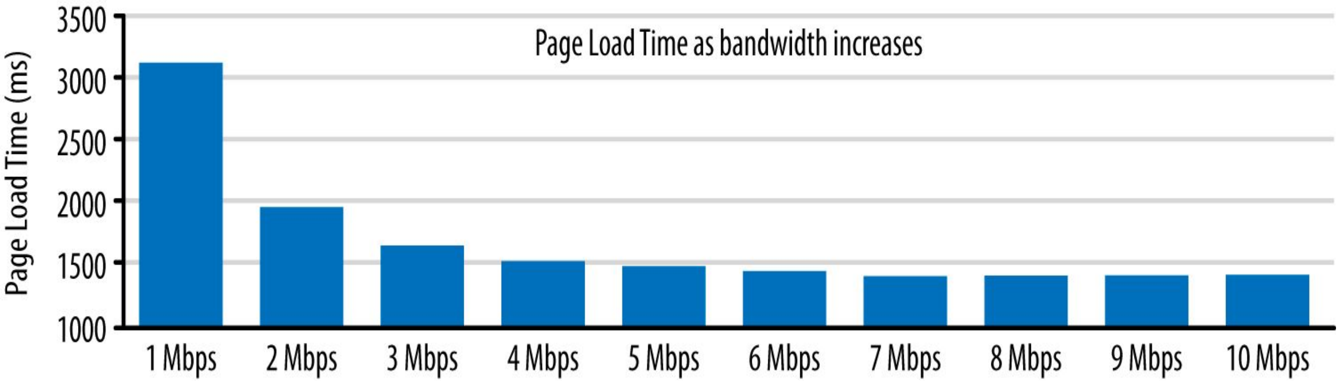
Improve end-user perceived latency

Address the "head of line blocking"

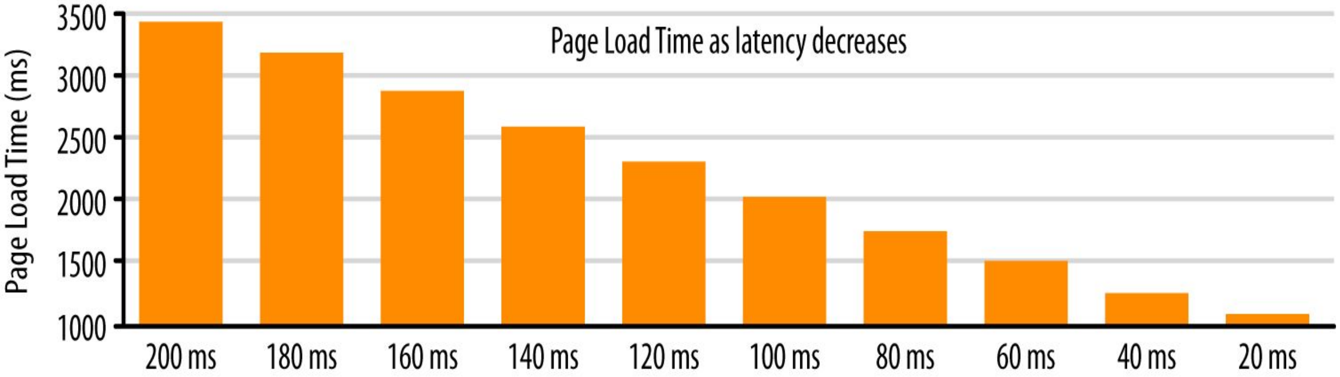
Not require multiple connections

Retain the semantics of HTTP/1.1

Latency vs Bandwidth impact on Page Load Time


Page Load Time as bandwidth increases

Single digit % perf improvement after 5 Mbps


Page Load Time as latency decreases

Linear improvement in page load time!

Latency vs Bandwidth impact on Page Load Time

Decreasing latency has more impact than increasing bandwidth

For Example

> Decreasing RTTs from 150 ms to 100 ms have a larger effect on the speed of the internet than increasing a user’s bandwidth from 3.9 Mbps to 10 Mbps or even 1 Gbps

HTTP/2 : Streams, Messages, and Frames

The introduction of the **new binary framing mechanism** changes how the data is exchanged between the client and server

Stream

A bidirectional flow of bytes within an established connection, which may carry one or more messages

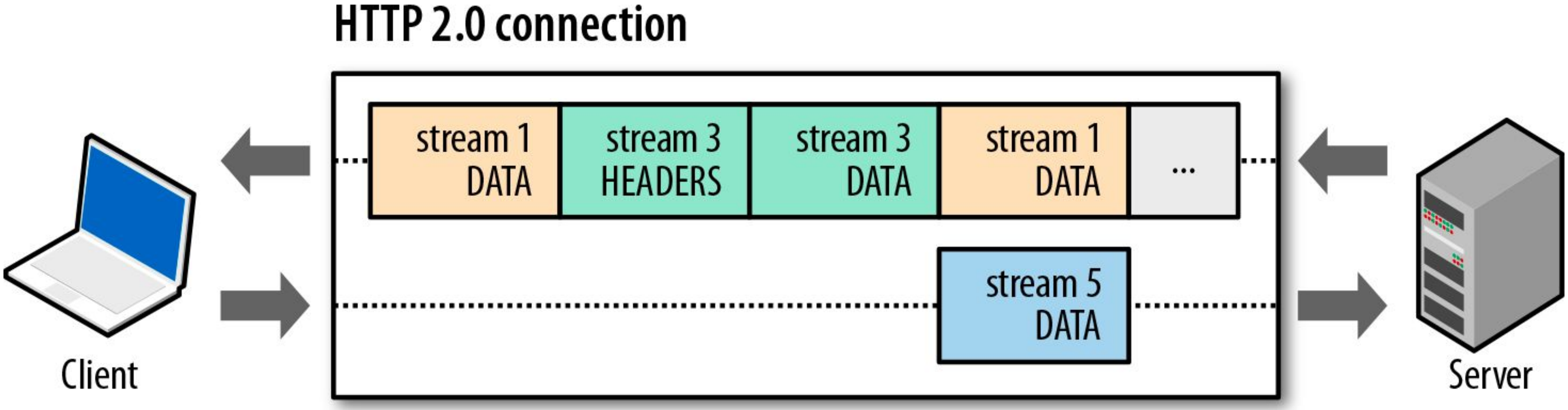
Message

A complete sequence of frames that map to a logical request or response message

`HTTP/2`: Streams, Messages, and Frames

Frame

The smallest unit of communication in **`HTTP/2`**, each containing a frame header, which at a minimum identifies the stream to which the frame belongs



`HTTP/2`: Streams, Messages, and Frames

The **frame is the smallest unit of communication** that carries a specific type of data—e.g., **HTTP headers**, **message payload**, and so on

Frames from different streams may be **interleaved** and then **reassembled** via the embedded **stream identifier** in the header of each frame

`HTTP/2` breaks down the HTTP protocol communication into an exchange of **binary-encoded frames**, which are then mapped to **messages** that belong to a particular **stream**, and all of which are **multiplexed** within a **single TCP connection**

HTTP/2: Main Characteristics

One TCP connection

Stream

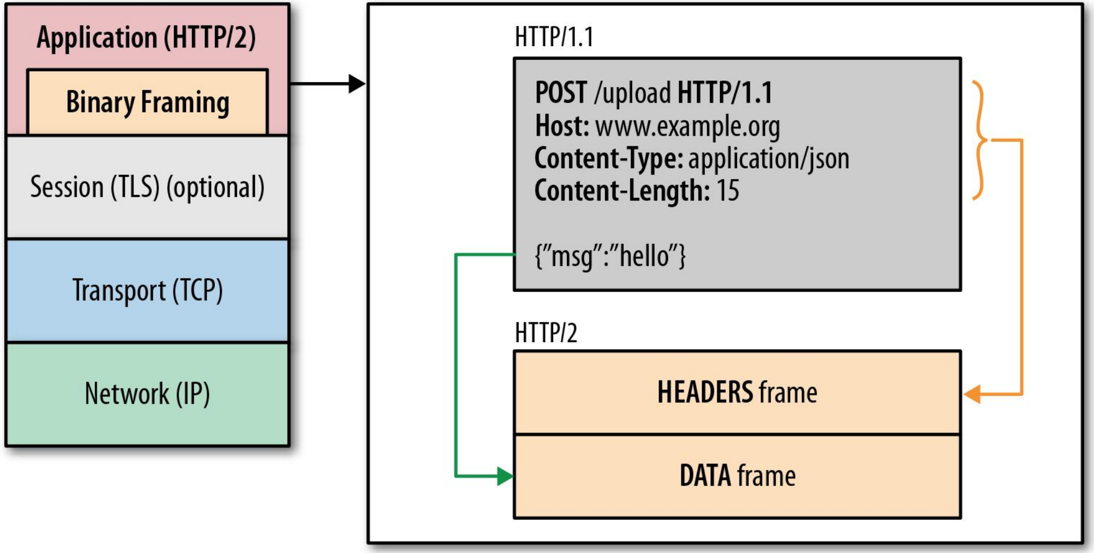
- Streams are multiplexed
- Streams are prioritized

HTTP/2 : Main Characteristics

Binary framing layer

- Prioritization
- Flow control
- Server push

Header compression (HPACK)

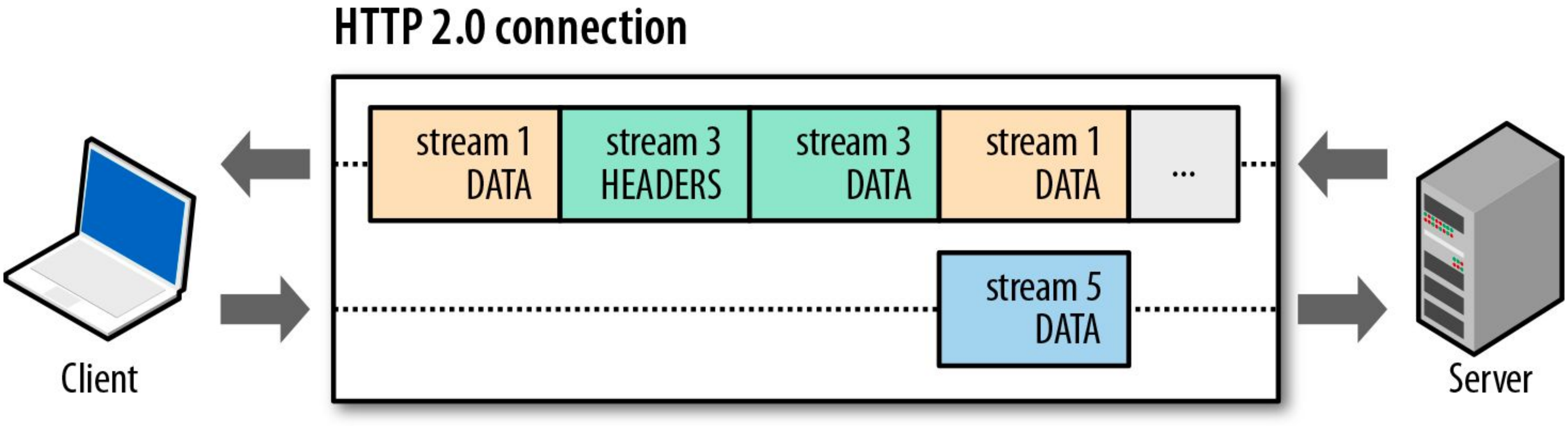


```
POST /upload HTTP/1.1
Host: www.example.org
Content-Type: application/json
Content-Length: 15

{"msg":"hello"}
```

HTTP/2: Basic data flow

How many streams are there in the diagram?

How many frames?



`HTTP/2:` Stream Multiplexing

Advantages

- **Interleave multiple requests in parallel** without blocking on any one
- **Interleave multiple responses in parallel** without blocking on any one
- Use a **single connection to deliver multiple requests** and responses in parallel

`HTTP/2:` Stream Multiplexing

Advantages

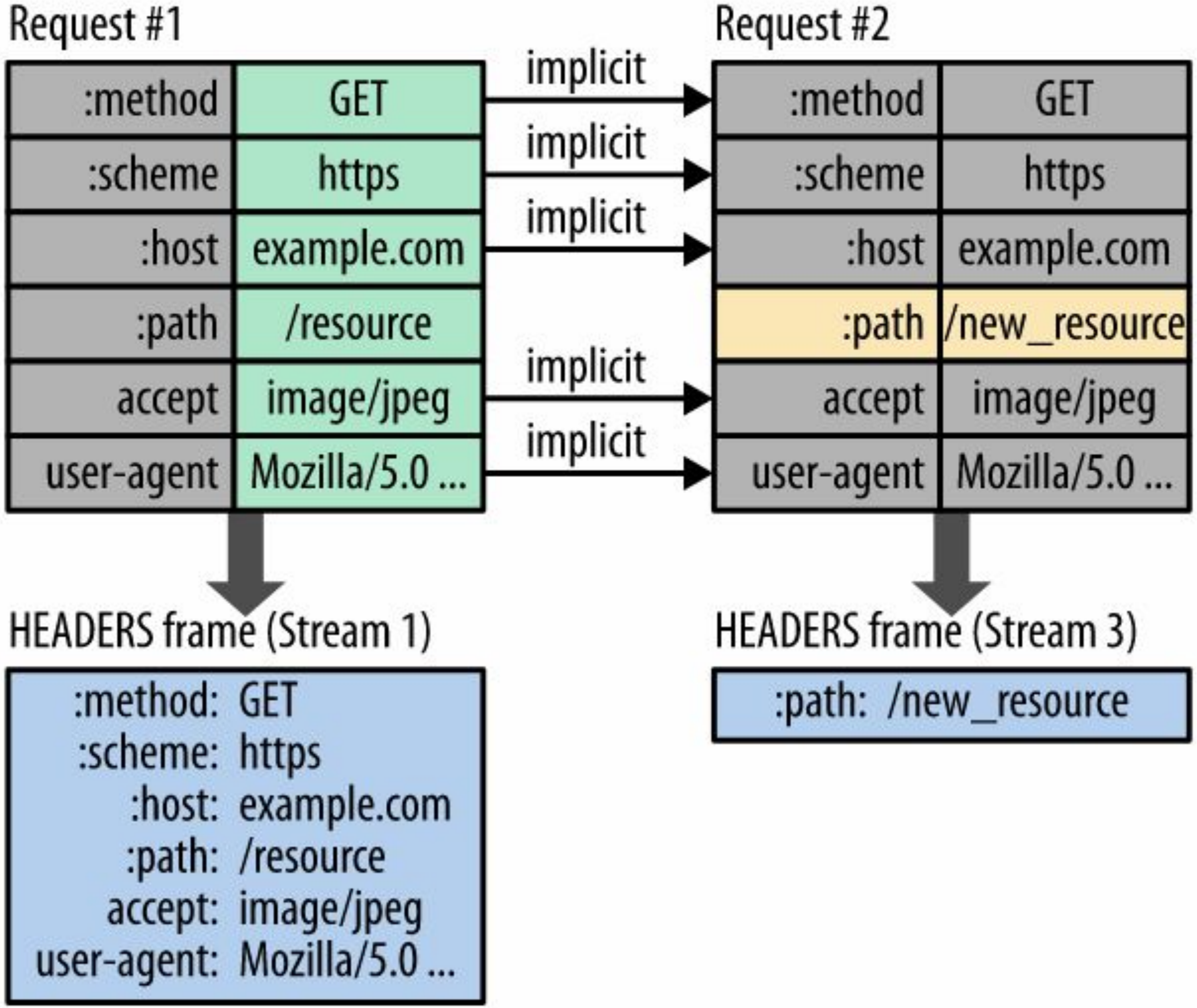
Remove unnecessary HTTP/1.x workarounds for optimization, such as concatenated files, image sprites, and domain sharding

Deliver **lower page load times** by eliminating unnecessary latency and improving utilization of available network capacity

HTTP/2

Header Compression

Uses **HPACK** algorithm



Request #1

:method	GET
:scheme	https
:host	example.com
:path	/resource
accept	image/jpeg
user-agent	Mozilla/5.0 ...

Request #2

:method	GET
:scheme	https
:host	example.com
:path	/new_resource
accept	image/jpeg
user-agent	Mozilla/5.0 ...

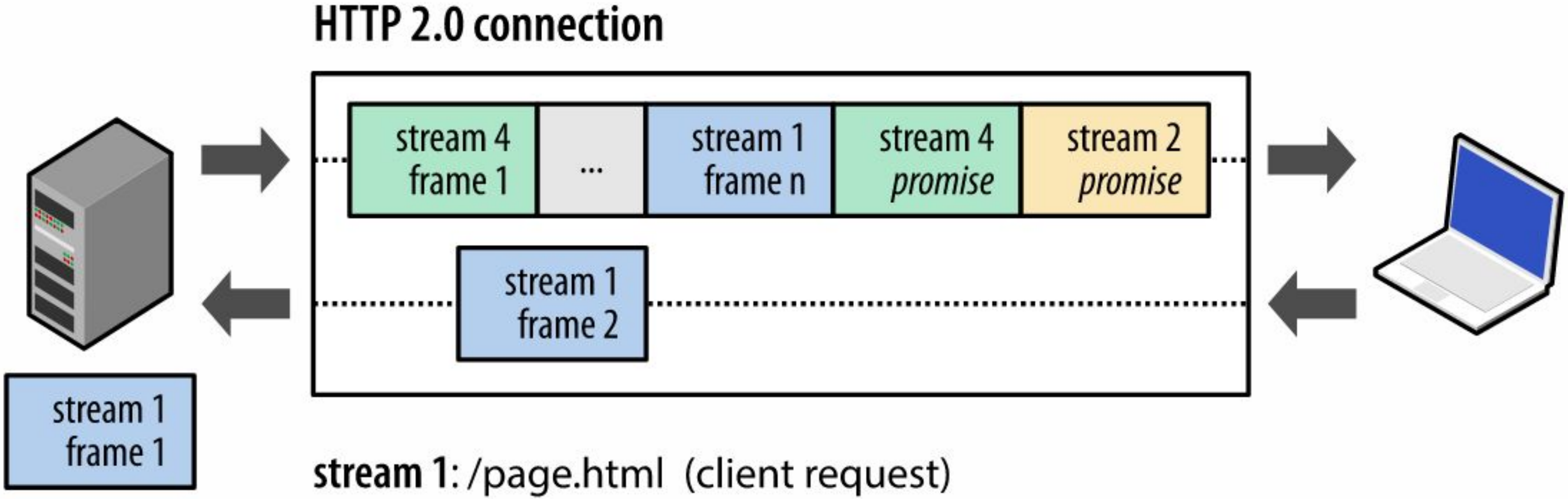
HEADERS frame (Stream 1)

:method:	GET
:scheme:	https
:host:	example.com
:path:	/resource
accept:	image/jpeg
user-agent:	Mozilla/5.0 ...

HEADERS frame (Stream 3)

:path: /new_resource

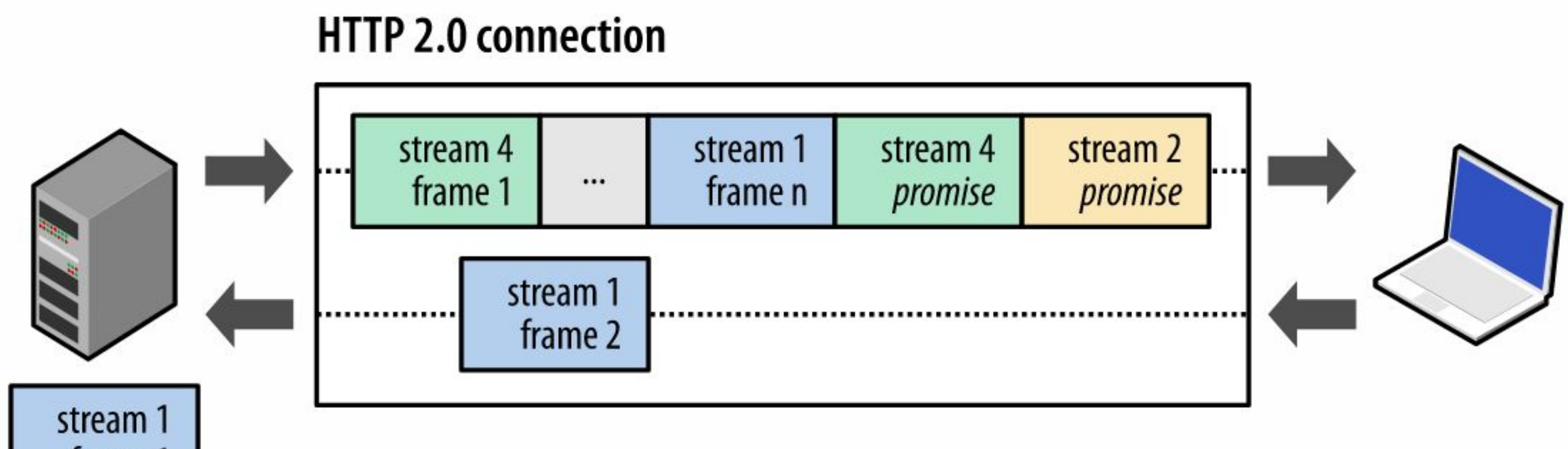
HTTP/2： Server Push



stream 1: /page.html (client request)
stream 2: /script.js (push promise)
stream 4: /style.css (push promise)

HTTP/2: Server Push

What are the advantages of server push?



stream 1: /page.html (client request)
stream 2: /script.js (push promise)
stream 4: /style.css (push promise)

HTTP/2: Advantages of Server Push

Pushed resources can be cached by the client

Pushed resources can be reused across different pages

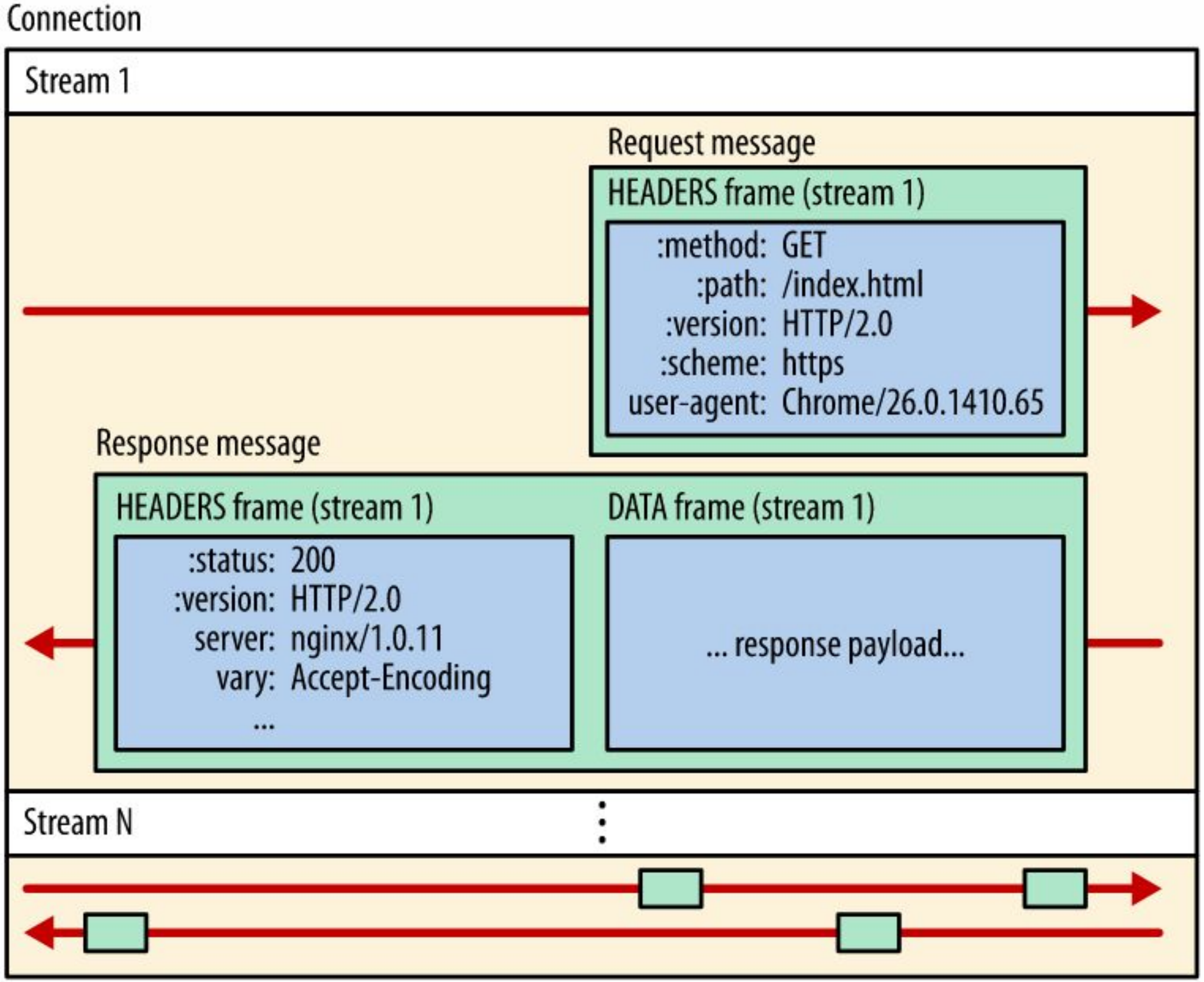
Pushed resources can be multiplexed alongside other resources

Pushed resources can be prioritized by the server

HTTP/2

Single TCP

Multiple Stream



Connection

Stream 1

Request message

HEADERS frame (stream 1)

```
    :method:  GET
      :path:  /index.html
   :version:  HTTP/2.0
    :scheme:  https
 user-agent:  Chrome/26.0.1410.65
```

Response message

HEADERS frame (stream 1)

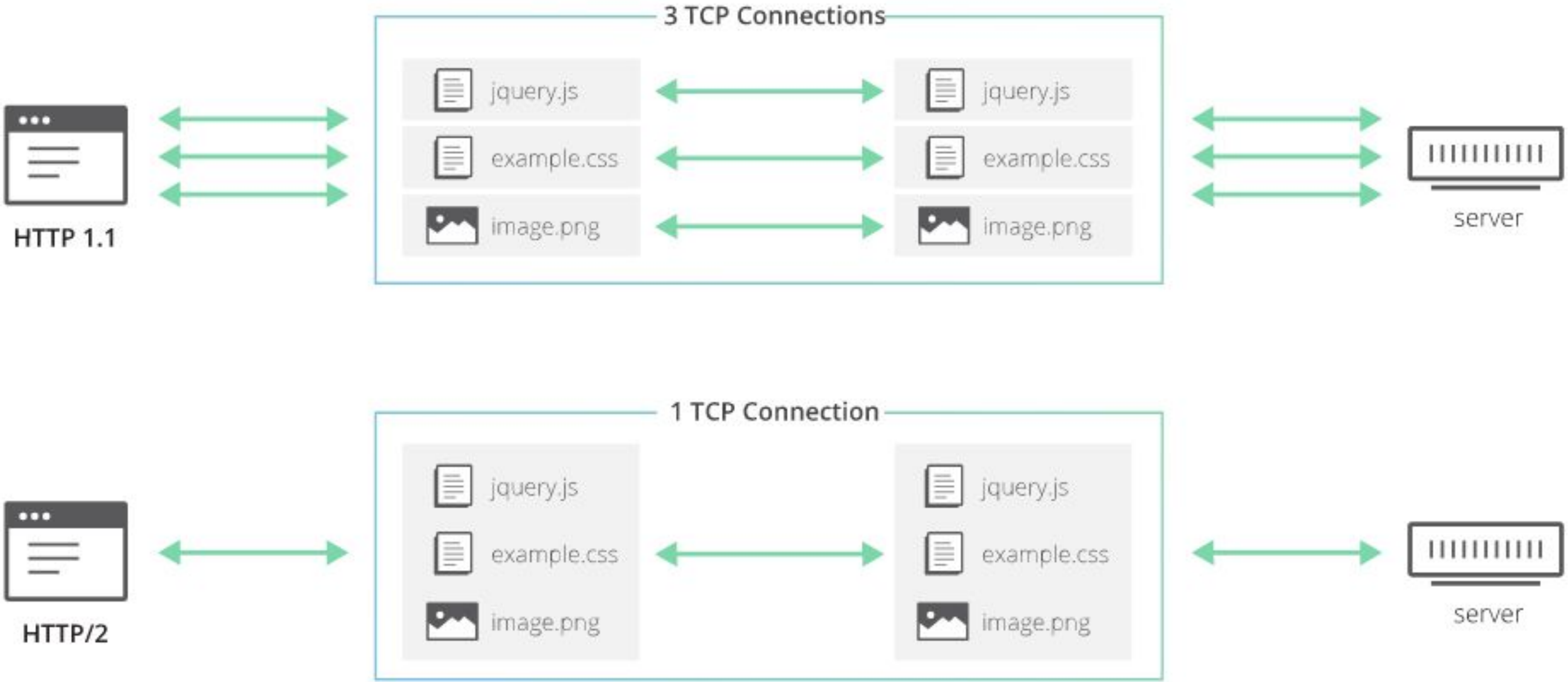
```
   :status:  200
  :version:  HTTP/2.0
    server:  nginx/1.0.11
      vary:  Accept-Encoding
       ...
```

DATA frame (stream 1)

... response payload...

Stream N

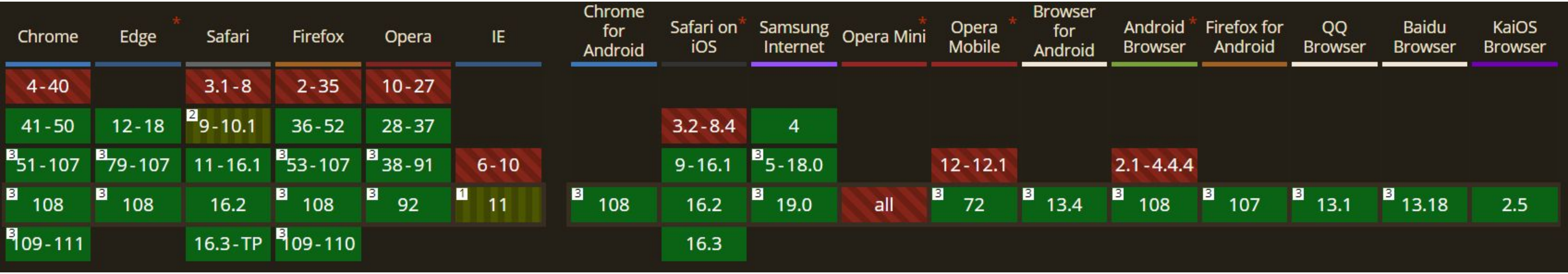
HTTP/1.X vs HTTP/2 TCP Connection



Browsers Supporting HTTP/2

CanIUse

Global Usage: 97%



Chrome	Edge *	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
4-40		3.1-8	2-35	10-27												
41-50	12-18	[2] 9-10.1	36-52	28-37			3.2-8.4	4								
[3] 51-107	[3] 79-107	11-16.1	[3] 53-107	[3] 38-91	6-10		9-16.1	[3] 5-18.0		12-12.1		2.1-4.4.4				
[3] 108	[3] 108	16.2	[3] 108	[3] 92	[1] 11	[3] 108	16.2	[3] 19.0	all	[3] 72	[3] 13.4	[3] 108	[3] 107	[3] 13.1	[3] 13.18	2.5
[3] 109-111		16.3-TP	[3] 109-110				16.3									

Limitations of `HTTP/2`

Can you think of any limitation of **HTTP/2** protocol?

QUIC

QUIC (Quick UDP Internet Connections) is a new transport protocol for the internet, developed by Google

QUIC solves a number of transport-layer and application-layer problems experienced by modern web applications, while requiring little or no change from application writers

QUIC is very similar to **TCP+TLS+HTTP2**, but **implemented on top of UDP**

QUIC

Key **advantages** of QUIC over **`TCP+TLS+HTTP2`** include:

Low connection establishment latency

Improved congestion control

Multiplexing without head-of-line blocking

Forward error correction

Connection migration

HTTP/3: HTTP over QUIC

Instead of using TCP as the transport layer for the session, it uses QUIC

QUIC introduces streams as first-class citizens at the transport layer

QUIC streams share the same QUIC connection, so no additional handshakes and slow starts are required to create new ones

QUIC streams are delivered independently such that in most cases packet loss affecting one stream doesn't affect others.

This is possible because QUIC packets are encapsulated on top of UDP datagrams

`HTTP/3`: HTTP over QUIC

Using **`UDP`** allows **much more flexibility** compared to **`TCP`**, and enables **`QUIC`** implementations to live fully in user-space — updates to the protocol implementations are not tied to operating systems updates as is the case with TCP

QUIC also combines the typical 3-way TCP handshake with **`TLS 1.3`**'s handshake

Encryption and **authentication** are provided by default, and also enables **faster connection establishment**

HTTP/3: HTTP over QUIC

Check the browsers supporting **QUIC** or **HTTP/3**

[CanIUse]

HTTP Messages

HTTP messages, as defined in HTTP/1.1 and earlier, are human-readable

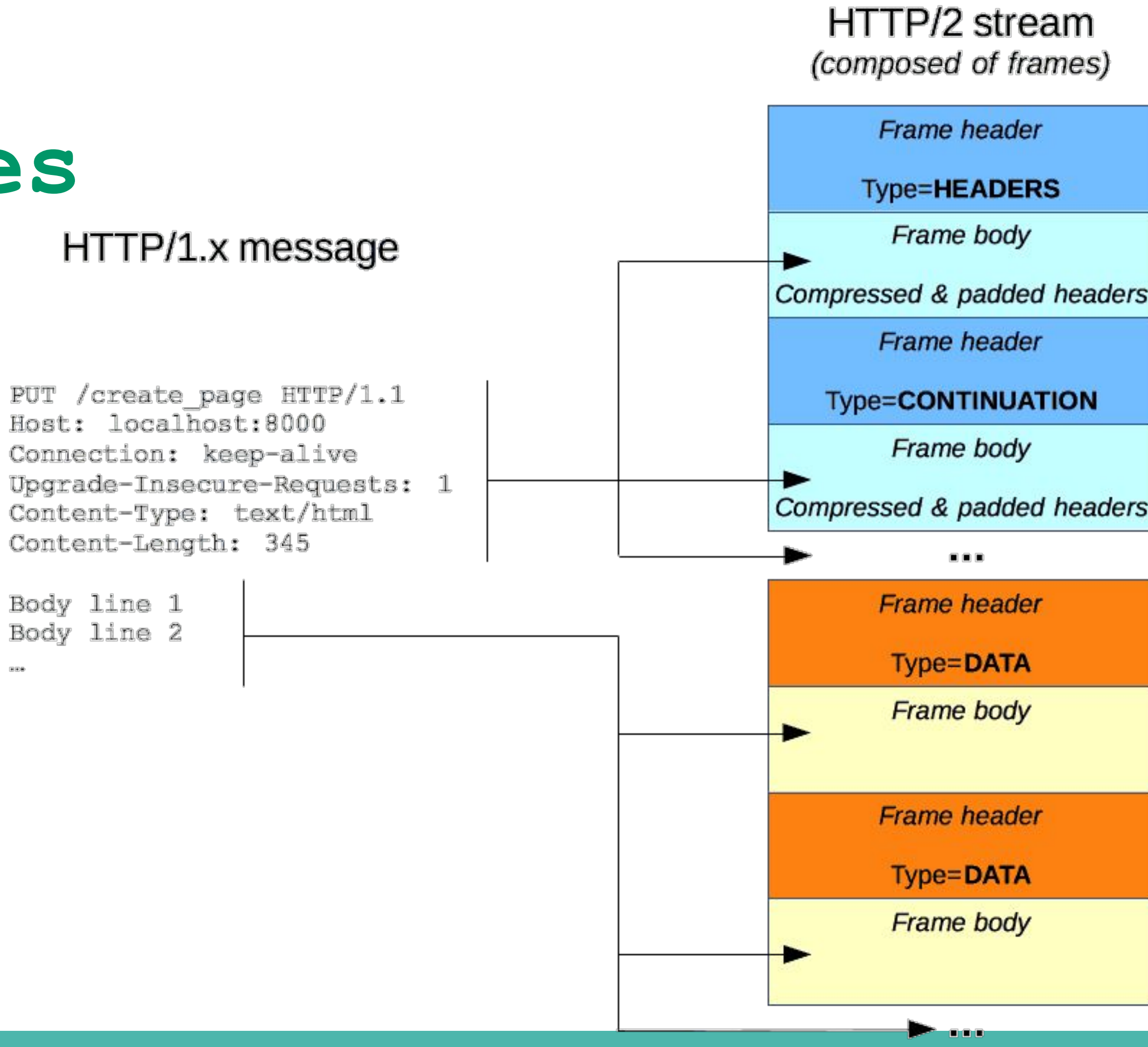
In **HTTP/2**, these messages are embedded into a binary structure, a frame, allowing optimizations like compression of headers and multiplexing

Even if only part of the original HTTP message is sent in **HTTP/2**, **the semantics of each message is unchanged** and the client reconstitutes the original HTTP/1.1 request

HTTP messages typically contain, **request/response line**, **request/response headers**, and/or **request/response body**

HTTP Messages

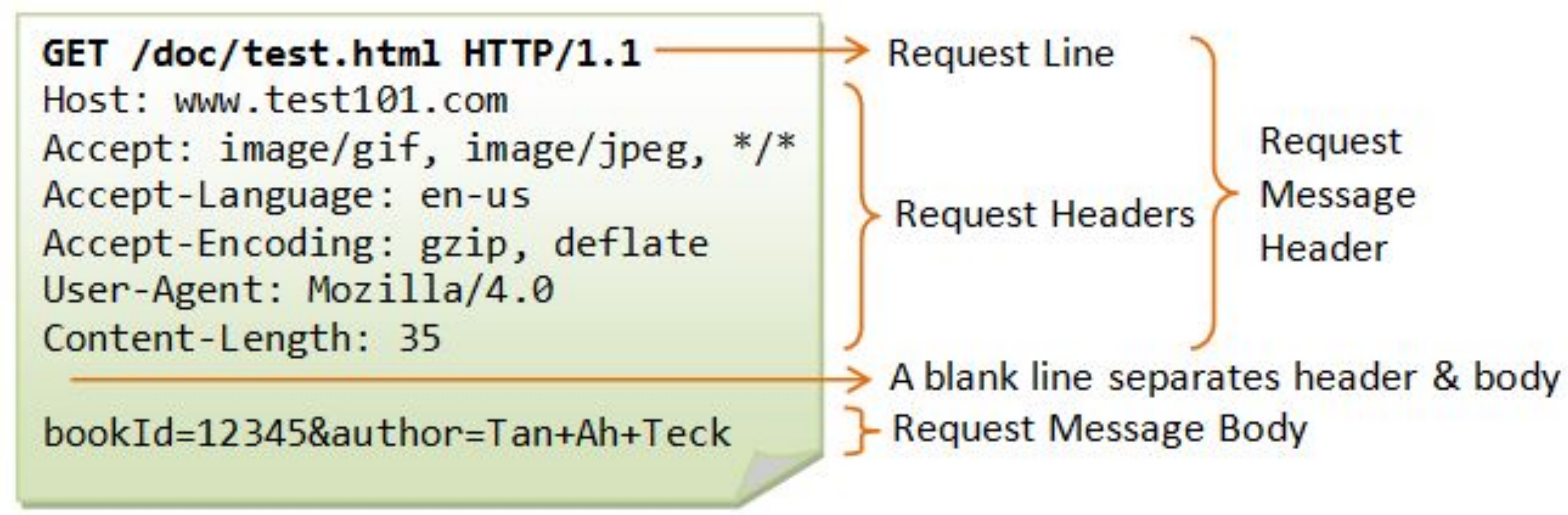
HTTP/1.X vs HTTP/2



HTTP/1.x message

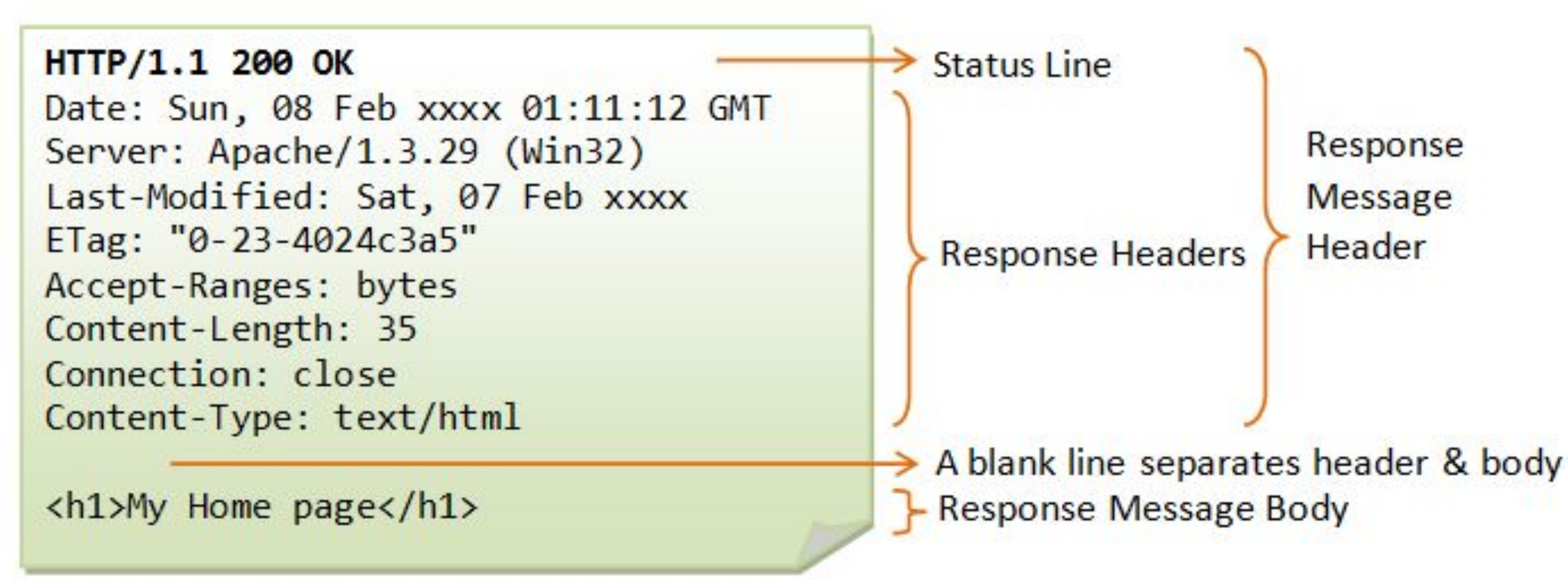
```
PUT /create_page HTTP/1.1
Host: localhost:8000
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: text/html
Content-Length: 345

Body line 1
Body line 2
…
```

HTTP Request Message



```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

HTTP Response Message



```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

Reference

High Performance Browser Networking

HTTP | MDN

https://www.rfc-editor.org/rfc/rfc9114.html

https://www.rfc-editor.org/rfc/rfc9000.html

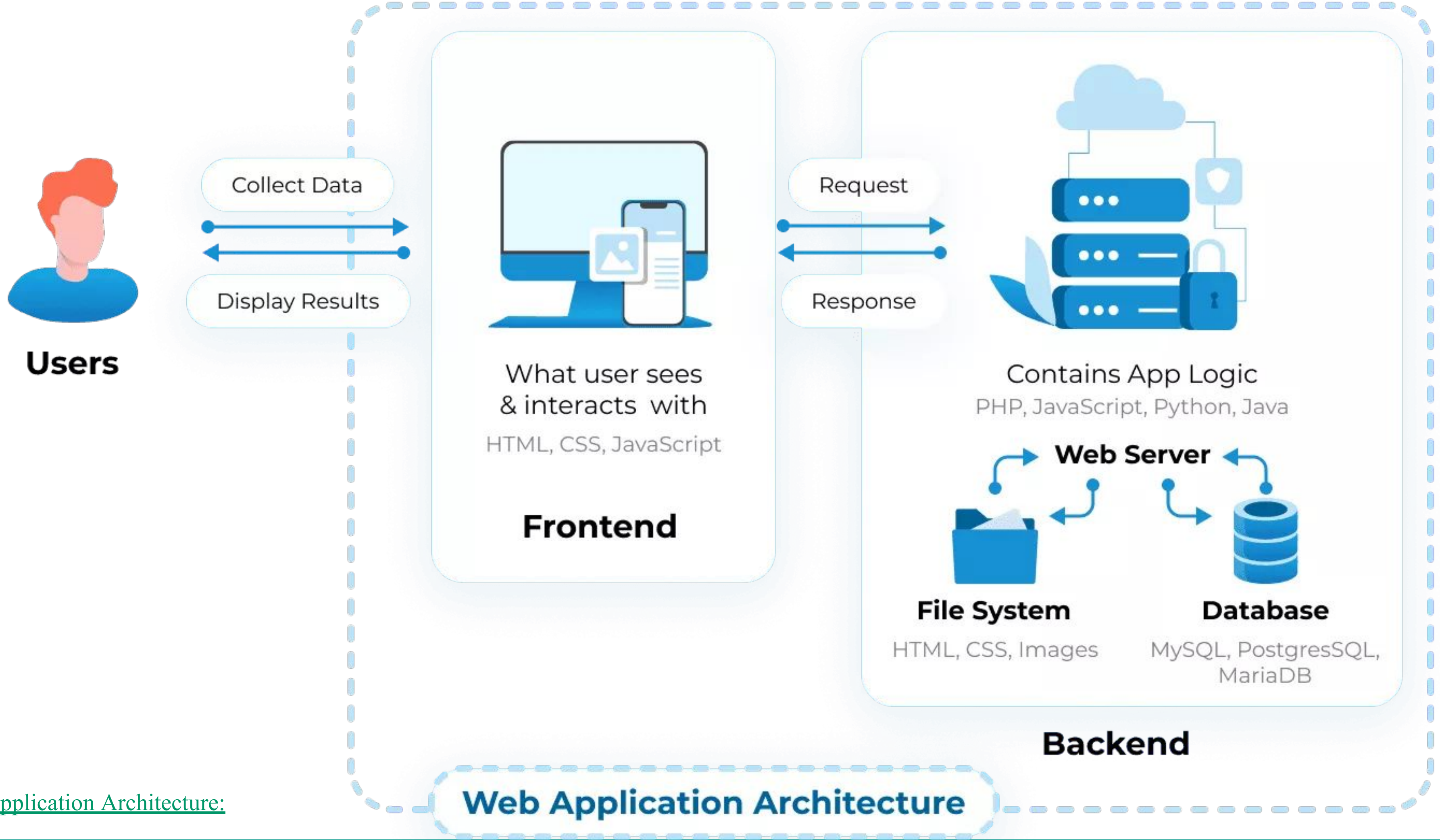
Introduction to
Server Side Programming

Topics

Server-Side/Backend Programming

Static/Dynamic Websites

Server-Side Web Frameworks



Web Application Architecture:

Server Side Programming

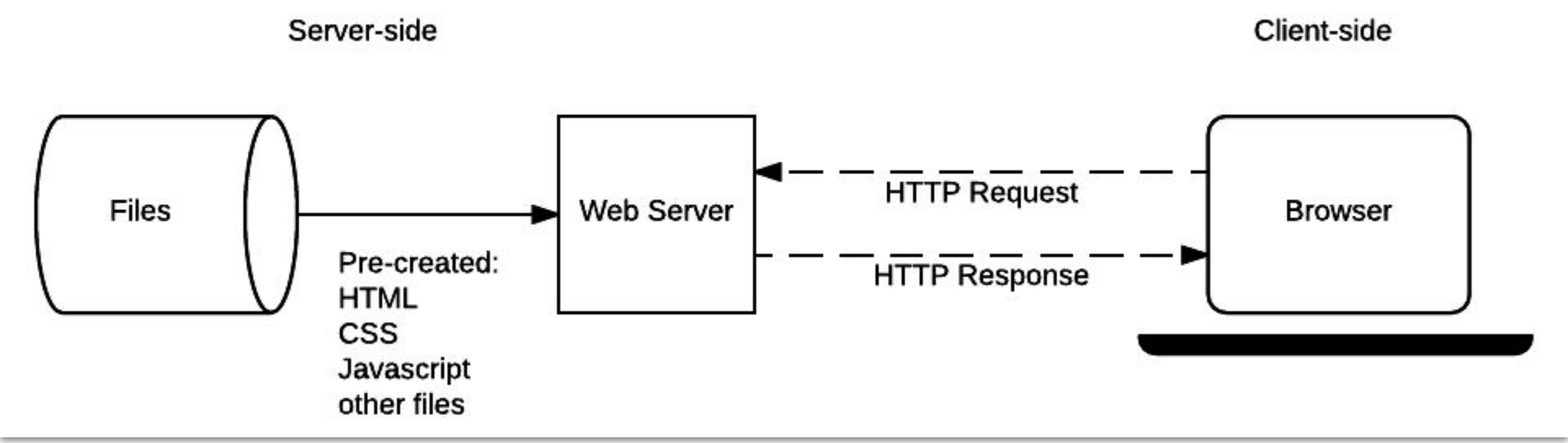
Dynamically display different data when needed, generally pulled out of a database stored on a server and sent to the client to be displayed via some code (e.g. HTML and JavaScript)

Web Servers

Web servers wait for client request messages, process them when they arrive, and reply to the web browser with an HTTP response message

Static Sites

Returns the same hard-coded content from the server whenever a particular resource is requested



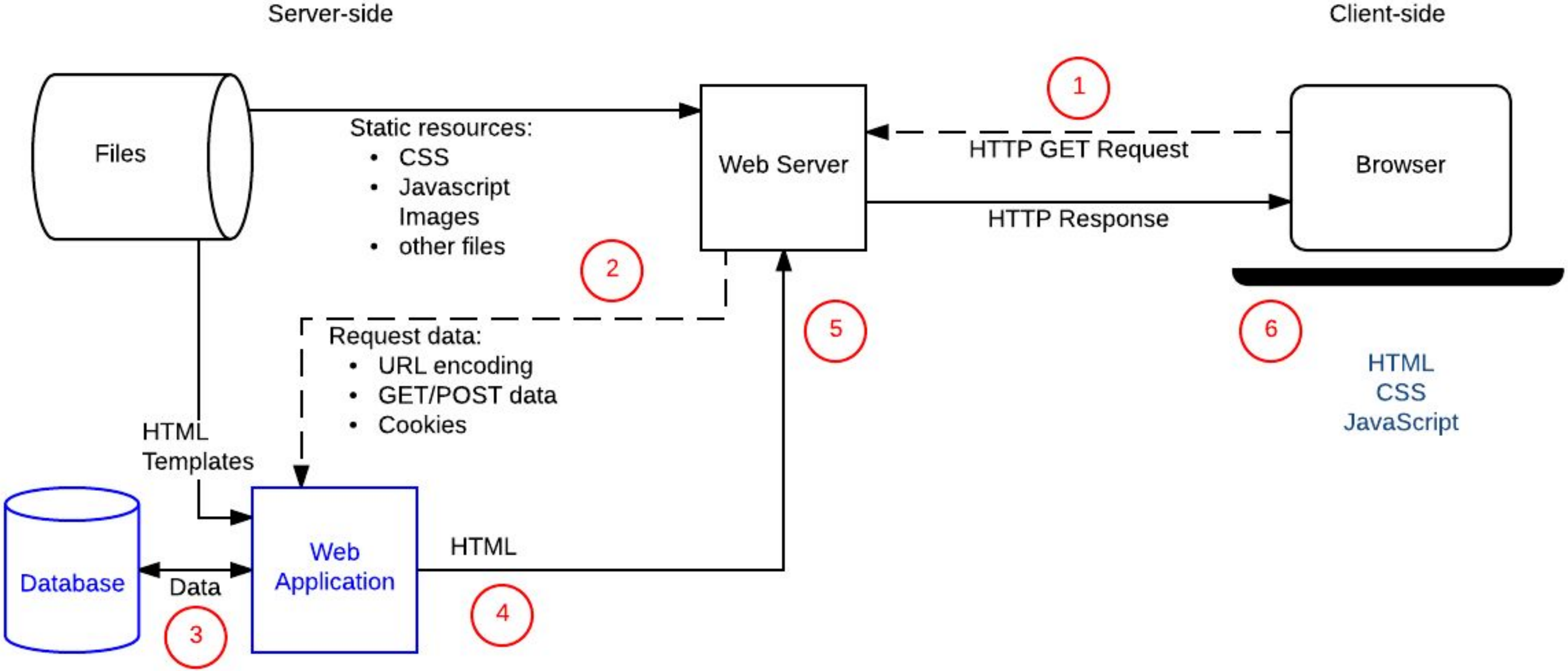
Dynamic Sites

Some of the **response** content is **generated dynamically**, only when needed

HTML pages are normally created by inserting data from a database into placeholders in **HTML templates**

Can return different data for a URL based on information provided by the user or stored preferences and can perform other operations as part of returning a response

Dynamic Sites



Server-side

Client-side

Static resources:

- CSS
- Javascript
  Images
- other files

Request data:

- URL encoding
- GET/POST data
- Cookies

What can you do on the server-side?

Efficient Storage and Delivery of Information

Customised User Experience

servers can store and use information about clients to provide a convenient and tailored user experience

Controlled Access to Content

restrict access to authorized users and serve only the information that a user is permitted to see

What can you do on the server-side?

Store Session/State Information

server-side programming allows developers to make use of sessions

session is a mechanism that allows a server to store information on the current user of a site and send different responses based on that information

What can you do on the server-side?

Notifications and Communication

servers can send general or user-specific notifications through the website itself or via email, SMS, instant messaging, video conversations, or other communications services

What can you do on the server-side?

Data Analysis

A website may collect a lot of data about users: what they search for, what they buy, what they recommend, how long they stay on each page

Server-side programming can be used to refine responses based on analysis of this data

Server-Side Web Frameworks

Also known as web application frameworks

They make it easier to write, maintain and scale web applications

Server-Side Web Frameworks

Provide tools and libraries that simplify common web development tasks, such as

- **route requests** to the appropriate handler
- make it easy to **access data in the request**
- abstract and simplify **database access**
- **formatting** output (e.g. HTML, JSON, XML)
- improving **security** against web attacks

Some Web Frameworks

Framework	Language and Environment
Django	Python
Flask	Python
Express.js	NodeJs, Javascript
Nest.js	NodeJs, Express.js/Fastify, Javascript
Deno	JavaScript, TypeScript, Chrome v8, Rust
Ruby on Rails	Ruby

Some Web Frameworks

Framework	Language and Environment
ASP.NET	.NET, C#
Micronaut	Java
Quarkus	Java
Spring Boot	Java, Kotlin
Laravel	PHP

Reference

https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction