[HTTP in depth]

↳ Application layer protocol

↳ Have a defined header bytes

HTTP → hypertext transfer protocol

```
const char* httpRequest = "GET / HTTP/1.1\r\n"
                          "Host: www.example.com\r\n"
                          "Connection: close\r\n\r\n";
```

↳ Request header data

```
std::string response =
    "HTTP/1.1 200 OK\r\n"
    "Content-Type: text/plain\r\n"
    "Access-Control-Allow-Origin: *\r\n" // Allow any origin
    "Access-Control-Allow-Methods: POST, GET, OPTIONS\r\n"
    "Access-Control-Allow-Headers: Content-Type\r\n"
    "\r\n"
    "File uploaded successfully!";
```

header

"\r\n" → separator for header and body

↳ Response header

Body

[HTTP 1.0] (1996)

↳ For each Request one TCP connection was open

↳ So, TCP handshake overhead for all subsequent request to same server

[HTTP 1.1] (1997)

↳ Single TCP Connection for Multiple request

↳ persistent connection

↳ How? "Keep-alive" Request header

L, For how much time this connection will be open?

- HTTP/1.1 assumes persistent connections unless explicitly stated otherwise using the `Connection: close` header.
- The `Connection: Keep-Alive` header (optional) can also include additional parameters to specify the duration or request count:

```makefile
Connection: Keep-Alive
Keep-Alive: timeout=5, max=100
```

  - `timeout`: Specifies the idle time (in seconds) before the connection is closed.
  - `max`: Limits the number of requests allowed on a connection before it is closed.

Idle Timeout:

- Servers typically impose an **idle timeout** to close connections that remain inactive for too long. Common timeout values range from **5 to 30 seconds**, though this depends on server configurations.
- For example:
  - **NGINX default**: 75 seconds
  - **Apache default**: 5 seconds

* HTTP 1.1 is persistent by default

[HTTP 2.0] (2015)

- ① Header Compression (HPACK)
- ② Binary Protocol → uses binary format for communication
- ③ Multiplexing → what does this mean?
- ④ Server Push

Understanding HTTP 1.1 problems

① GET /index.html
② GET /main.js
③ GET /img.jpg
④ GET /main.css

In HTTP 1.1

↳ Until or unless 1'st request is not complete, second request can't be fired.

So, if GET /img.jpg takes 10 seconds

For 10 seconds, this TCP Connection can't be used for another request

⇓

Obviously, we can open 4 TCP Connection → Hack

* Chrome creates Max 6 TCP per origin server

⇓

Need some solution

⇓

Google developed HTTP 2.0

↳ Based on SPDY

* Multiplexing

↳ Some TCP but no head of line Blocking

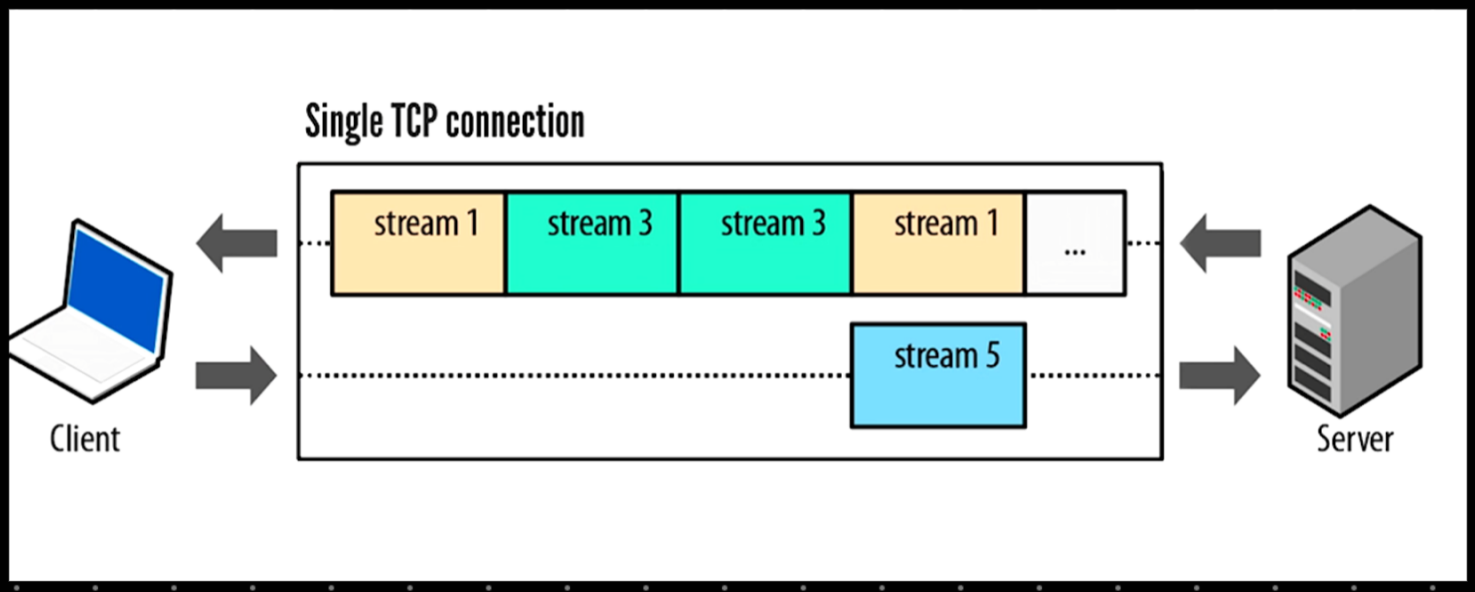
↳ Multiple concurrent request and response on Some TCP connection

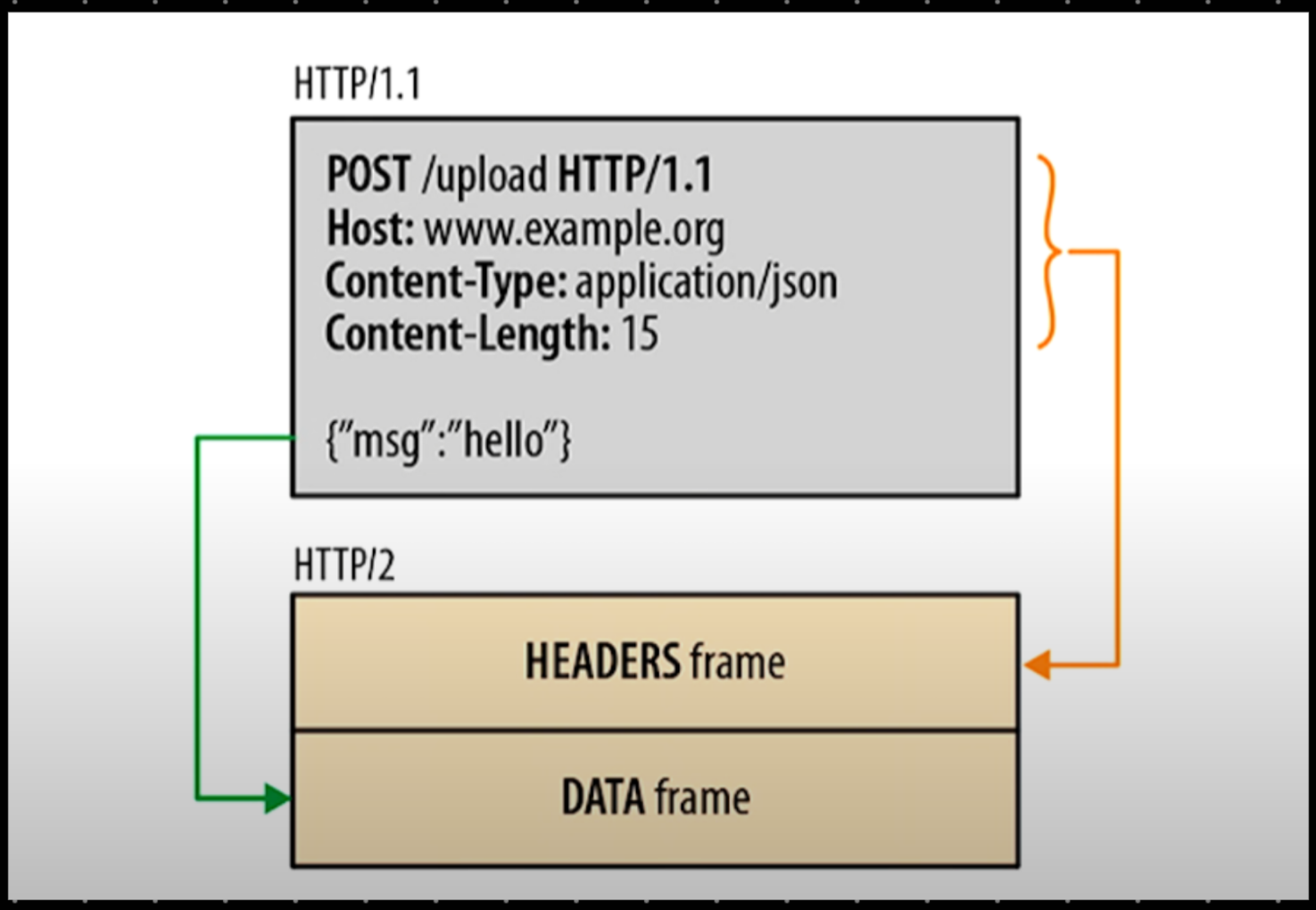
||

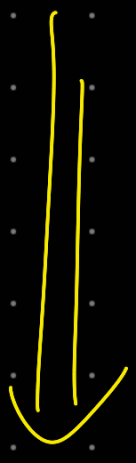
How it is done?

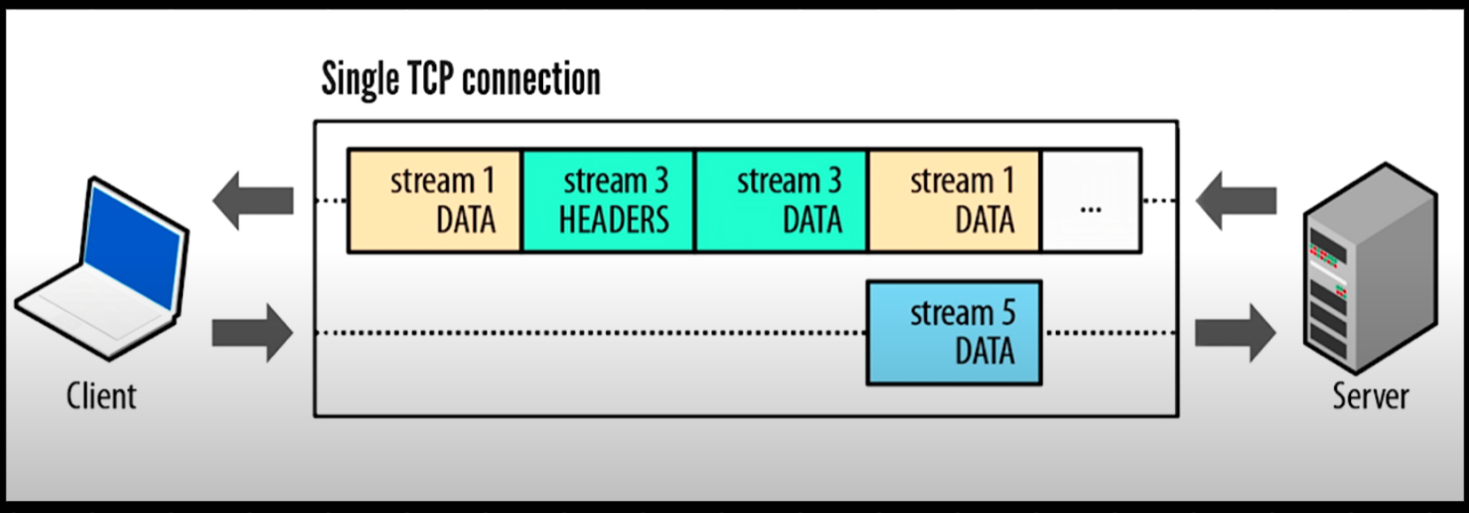
* Each request has to tagged to let server and client understand which request/response recieved.

↳ Also known as stream ID


Single TCP connection
stream 1
stream 3
stream 3
stream 1
...
stream 5
Client
Server


HTTP/1.1
POST /upload HTTP/1.1
Host: www.example.org
Content-Type: application/json
Content-Length: 15
{"msg":"hello"}
HTTP/2
HEADERS frame
DATA frame





* What is Server Push?
* Client → request /html → server responds

Server can send other related files
↳ This has to be configured at server properly

* There are lot of headers which are common in every HTTP packet, so we can compress. (HMACK)

* HTTP 2.0 would be useful for assets, so better enable HTTP 2.0 on CDN servers

[HTTP 3.0]

↳ Uses UDP at transport layer

↳ Also known as QUIC

↳ Useful for video streaming, assets

↳ Used by Youtube to stream video