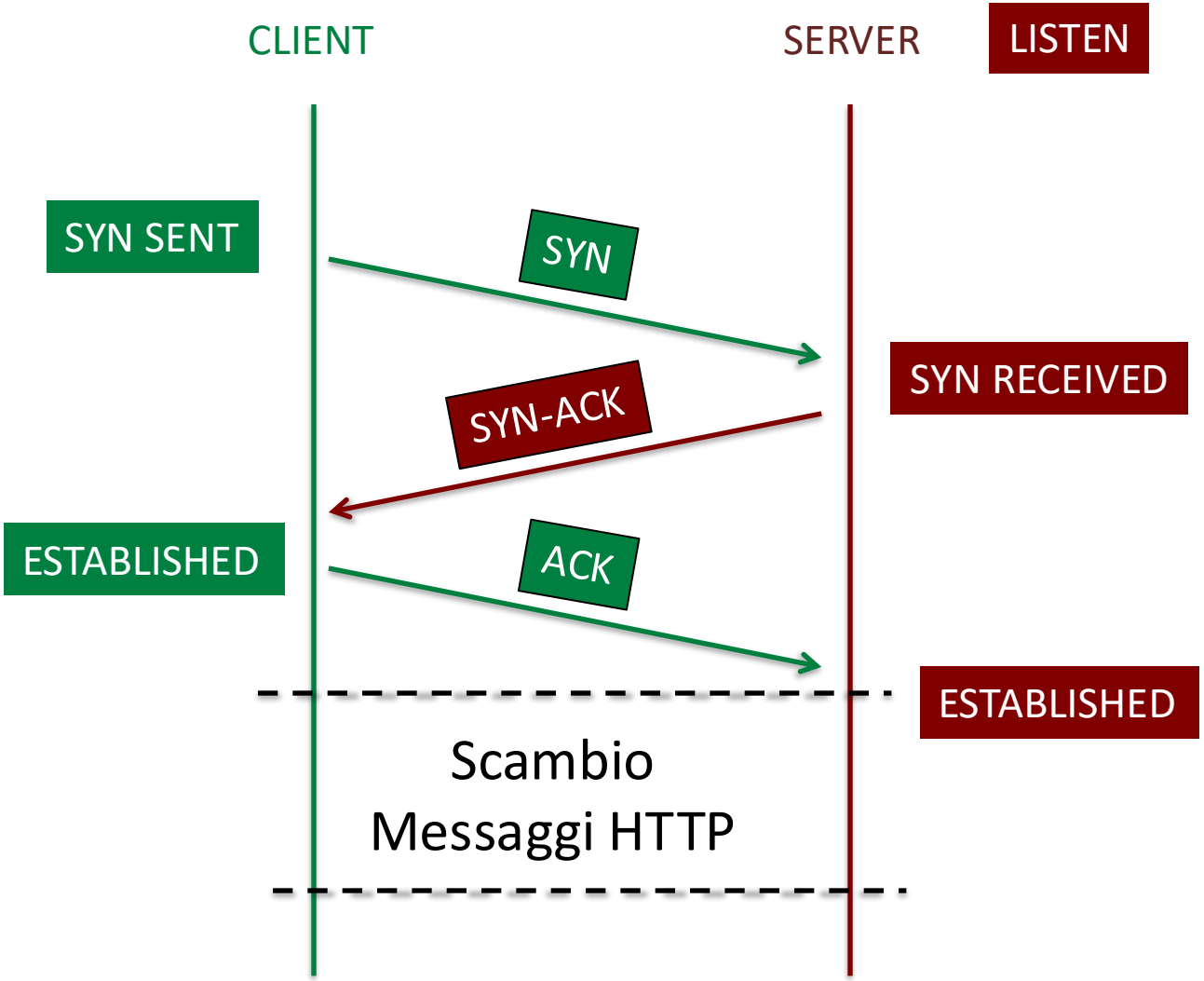
EVOLUZIONE DI HTTP

HTTP 1.1 - RFC 2616

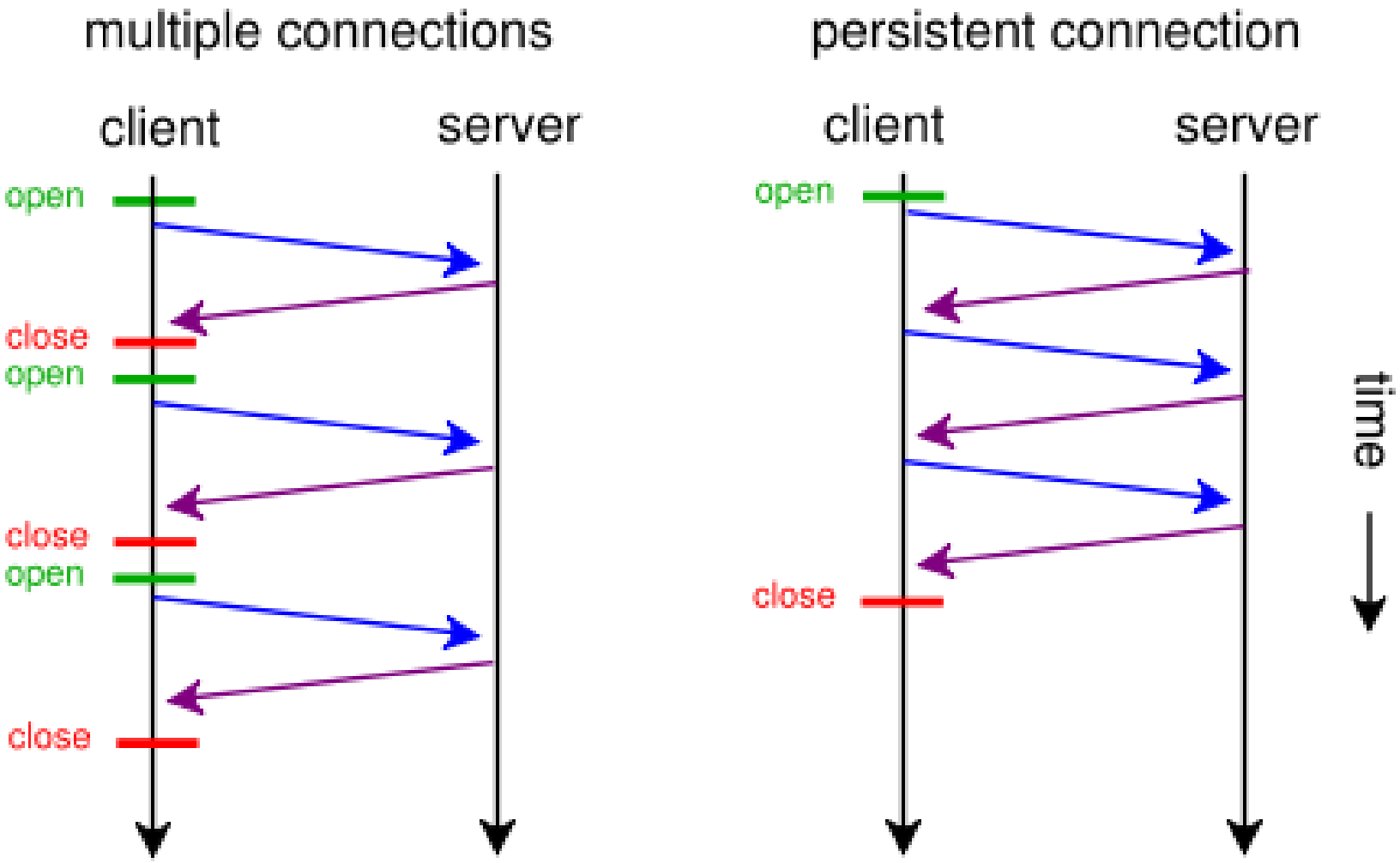
- Connessioni persistenti
  - (è il default) e pipelining
  - maggior velocità (più transazioni su stesso canale)
- Gestione trasmissione del body
  - Negoziazione formato dati e lingua del body
  - frammentazione (chunked encoding) per pagine dinamiche
  - trasmissione parziale
- Gestione della cache
  - Modalità specificate dal protocollo
- Gestione dell'autenticazione
  - autenticazione con digest

CONNESSIONE PERSISTENTE

Set up connessione TCP



Connessione Persistente



VIRTUAL HOST

Virtual Host

Problema: far ospitare più siti ad un computer con un solo IP!!

- HTTP/1.0:
  - un indirizzo IP per ogni server web ospitato su un nodo (multihomed)
- HTTP/1.1:
  - più server logici associati allo stesso indirizzo IP
  - ottenibili tramite alias nel DNS (record CNAME)
- Il client deve indicare il virtual host desiderato
  - **Host:** *Fully Qualified Domain Name* [: port]"

Esempio

Configurazione DNS			
host.provider.it	IN	A	10.1.1.1
www.musica.it	IN	CNAME	host.provider.it
www.libri.it	IN	CNAME	host.provider.it

collegamento a host.provider.it, ossia IP 10.1.1.1
GET /index.html HTTP/1.1
Host: www.libri.it

collegamento a host.provider.it, ossia IP 10.1.1.1
GET /index.html HTTP/1.1
Host: www.musica.it

Negoziazione del contenuto

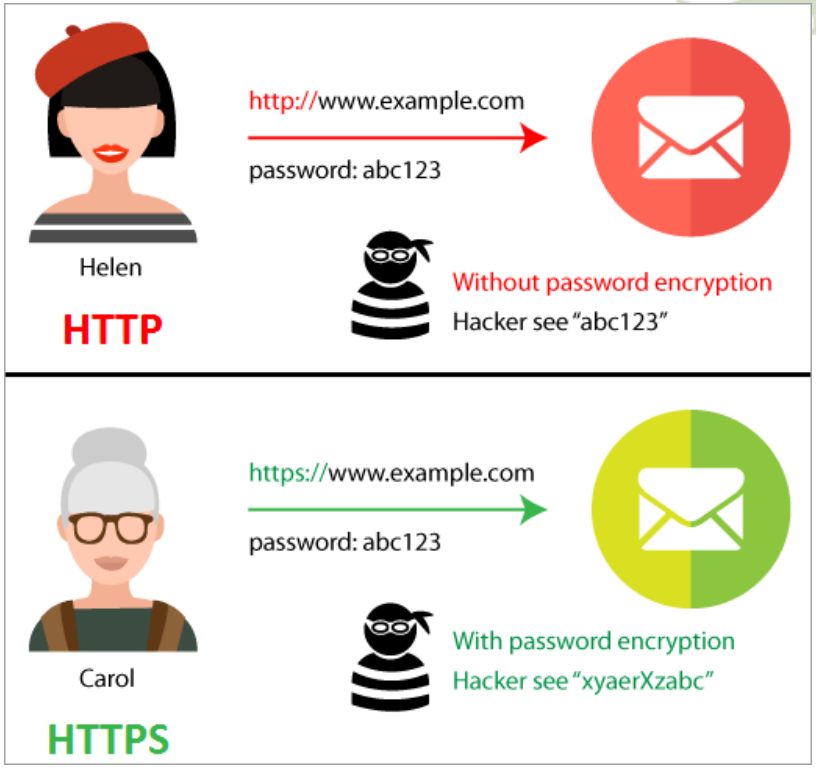
- Il server può adattare il contenuto in base alle informazioni inviate nella richiesta

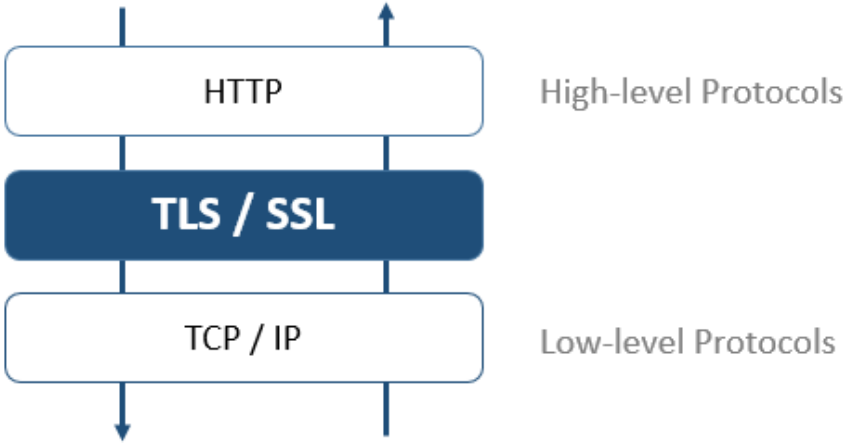
- Quality factors
  - Numeri reali fra 0 e 1
    - Default 1
- Esempio
  - Accept-Language: fr, en-gb; q=0.8, en; q=0.7

HTTPS

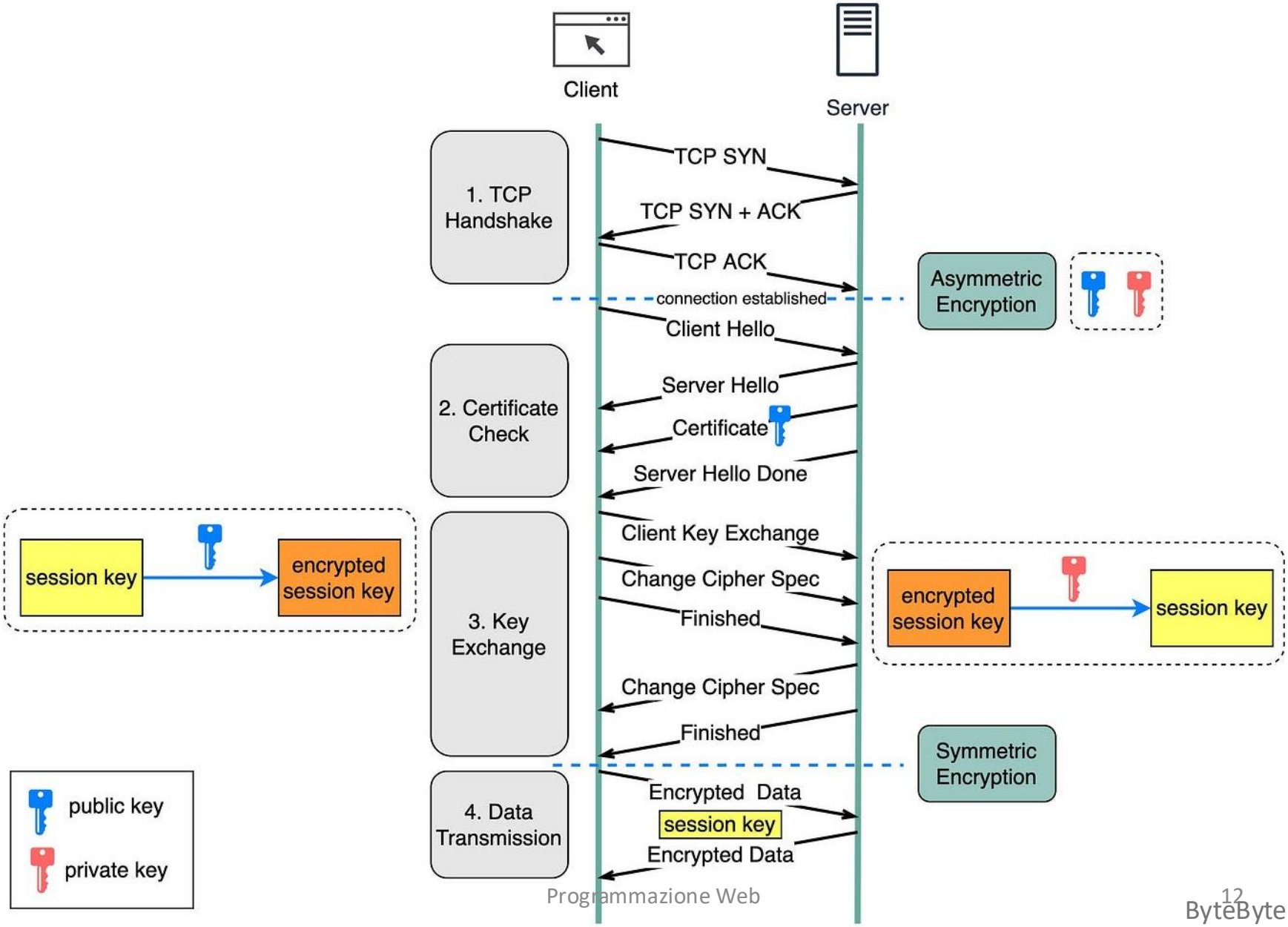
HTTPS = HTTP + Security

- I messaggi HTTP vengono cifrati
- Si aggiunge il protocollo TLS/SSL
  - tra HTTP e TCP
- le url cambiano in https://
- I server usano la porta 443





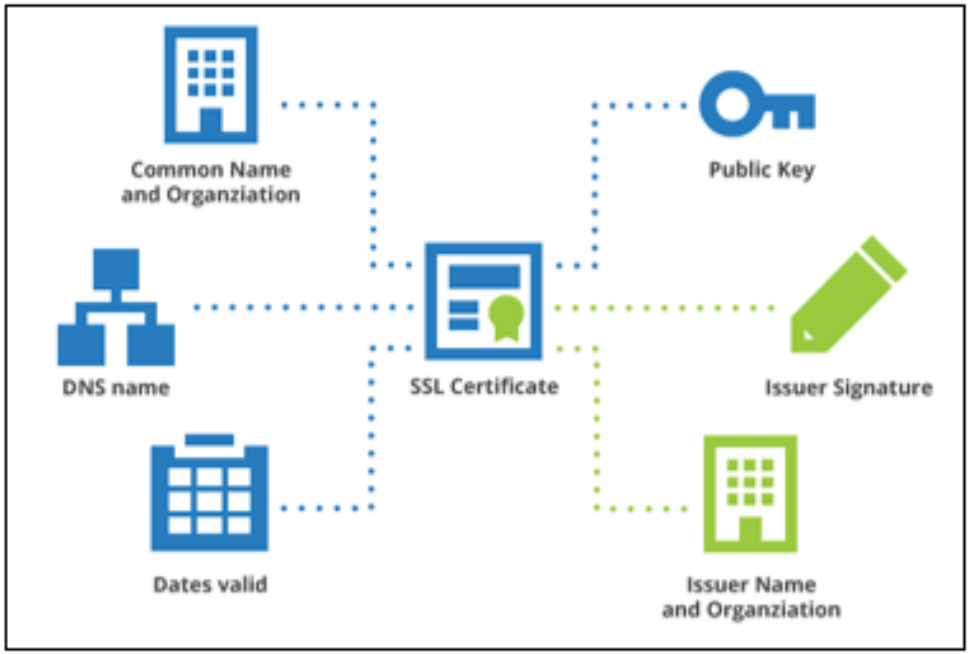
TLS handshake



ByteByteGo

Certificati e chiavi

- I certificati si installano nel server
  - includono una chiave pubblica
  - formato X.509

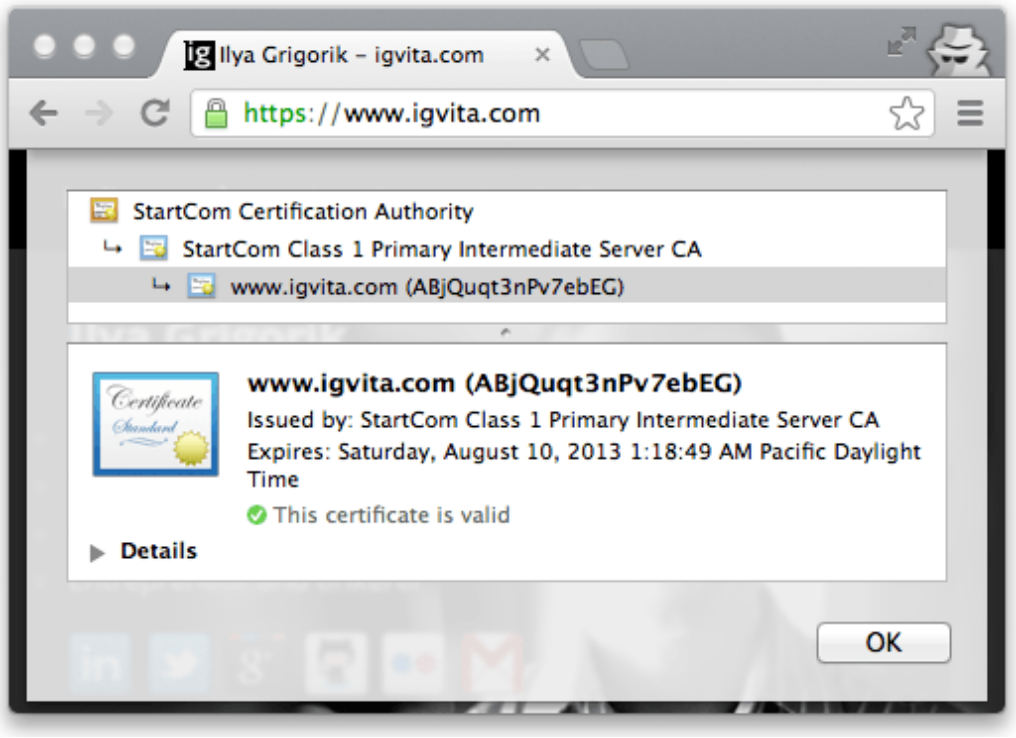


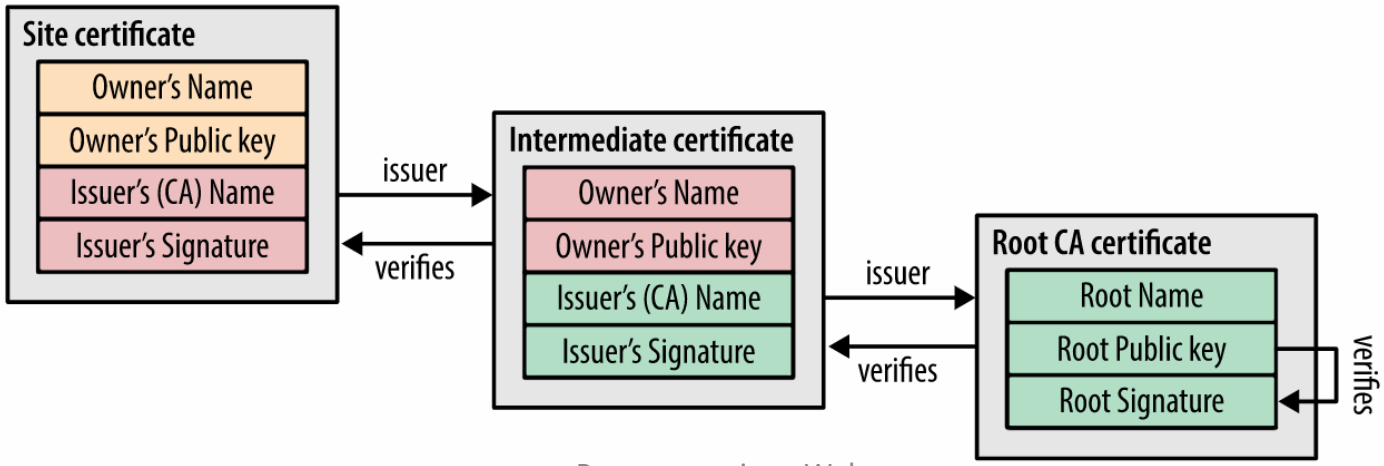

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAqwitjHs18lz19cGdvgne1ck8IYQUPyKAPtkPU3HZ+FwY3aTw
1IizXTH1QHXAMrpYF8MACk/lzC4aVlaUW6DxcXWyQnDkddJ8DKYgKxqE0m3Adtyy
XknNH1xWG7EDWXL5aysJ+kubtDqqhOHuMCPNbCNJ+zcnPsetccdNwL2prISFLUO5
LrLZ/NqHxXKPep5F6GrxRZj1DTMeQ7TUG+MawJG0vxN2GHdyU+CBx7nvQ2xI0nCL
mkVSFBsW1vrP1LPEVi7Dwh8oD5nNqslxitDBI+nTZbmx9iCFFjxfeGh6VryVRaGK
/3lFH21rNMVh6Kk2HfK3j5MJZ5dft3gbPznZiQIDAQABAoIBABtnn49D6sfz10FS
jbj2kesFfIoORMQ8/fmUBB2lv3xJHX7eP+x+HudO6vIyK+K08wqTpzz/w11uAbBV
3R21cxpv9mko96JdYJ0SPVNKw3+IMG+bO0VaY/CjdfSn71Z00rw2QDoHxWIiiFIU
dXrnMbIjEWacVsB7dDIn8jBw82/95N4n+9rhloNccH9jeCoRIgsLVbpu8tcHsSFN
Uk5SwNRfPYUoq7Q6jD6HNbeEtabyvJfumDbdiB8DusUW83RohKpNaP8+z1ZAMbUp
IySayM4ZNn79ZwGBFZXpe2Ib4MII8IYtxH0vd3zzMRfXf8nsWjuXCFueOVY93Xgn
+ejgYhUCgYEA0n0GvU7LxgIdlc6zi5BULV0ryBsWrQ2NRC+1HG4MlwoJpZs/VFTY
3XQN8qNaz9VvLdo6GVP0hIfFUvs6JA+MdqLMC6+i0cOfdMNs1APOCtQHXrSI0s+h
XrNFXHd0OyqG373Q9JNfcWDLHkaSGWpaQqX3AgJTNy7Dr8VJ8Olym1MCgYEA0APD
VZEf3uJP1HIN16FqGjKGDzVdcviW52OViKMPp39vH1eOMWSu1J9g8uG1wqDn+vGy
8cm/PQrv1+CvFGmhdooP34Z1pKXyTagUIV1TqeG90xsLnw1vZi9//PGEG7pxLejJ
L0DH+L3DoAMUECsgBLSt942gC3OfjGbFwAYBGTMCgYEAxnhTOX8pzwvBn7bmPuuZ
I6kdljJMoGIN2MRvL15qCqTRUdDf1wGl9rHuUOJMx3mFuqNxct5hyfFHUyZFvSjY
FikM4tv6V3XwTaXTYNox1q40QM1T0gsrKmM5PKX596vyvtd5RpQLQ5m0DFkYe1R3
xS4+iStuEbNLmtoZDUrh37cCgYBiyna5q6vcOdU6C1ThpOp0OKyMioawNLnOePAq
rW6g2RfYZcfFU5DosCzmDSY6ohqqhqG6zVsCza/s8euUoTqqxxUUzpIOuTVjhkDR
OdBPLUPu/7g8NsnuWP7LDZT1ak2YRSdPRs/yJYXD99JOf3y8X2cD3AI3jhknICQo
t3FMGwKBgDR7+ya7Tn8KZd19oANZjuqIuD9DF1iTeIQc7fRcVlUDJxQaT7joWvSM
EexltK3KQc27rK1j77ED3maG9Ll9RP5SuFQVsK78pBosMNZxNfDp4wCtsjWicYcG
OG1Zr1QgVgoWerDjPYa9gLomCum+PNKNA7fTxqVqb8Z6957kTgFn
-----END RSA PRIVATE KEY-----
```


chiave privata

Chain of Trust

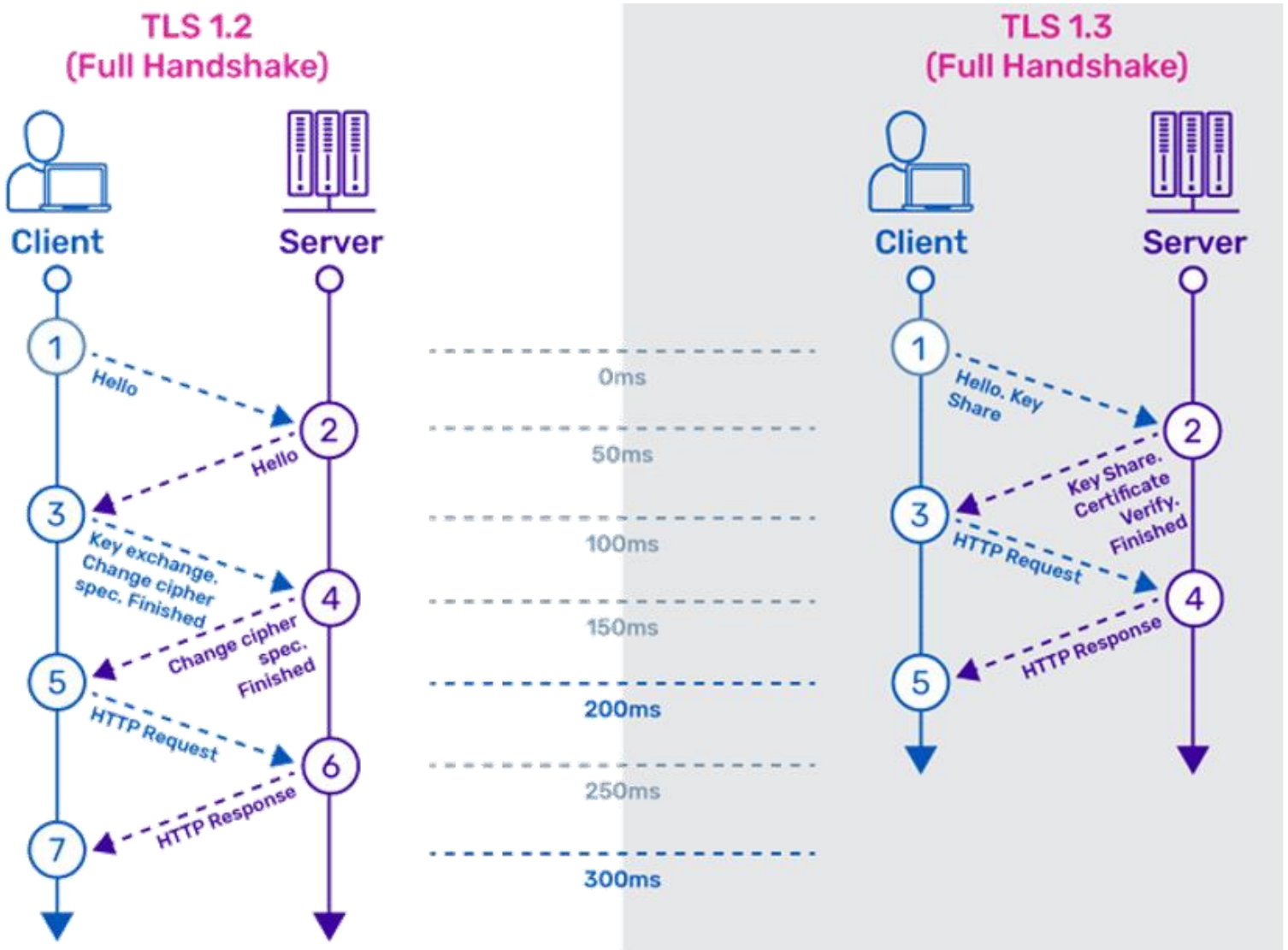
Come possiamo verificare l'identità del server?





https://hpbn.co/transport-layer-security-tls/

TLS 1.3 with Diffie–Hellman



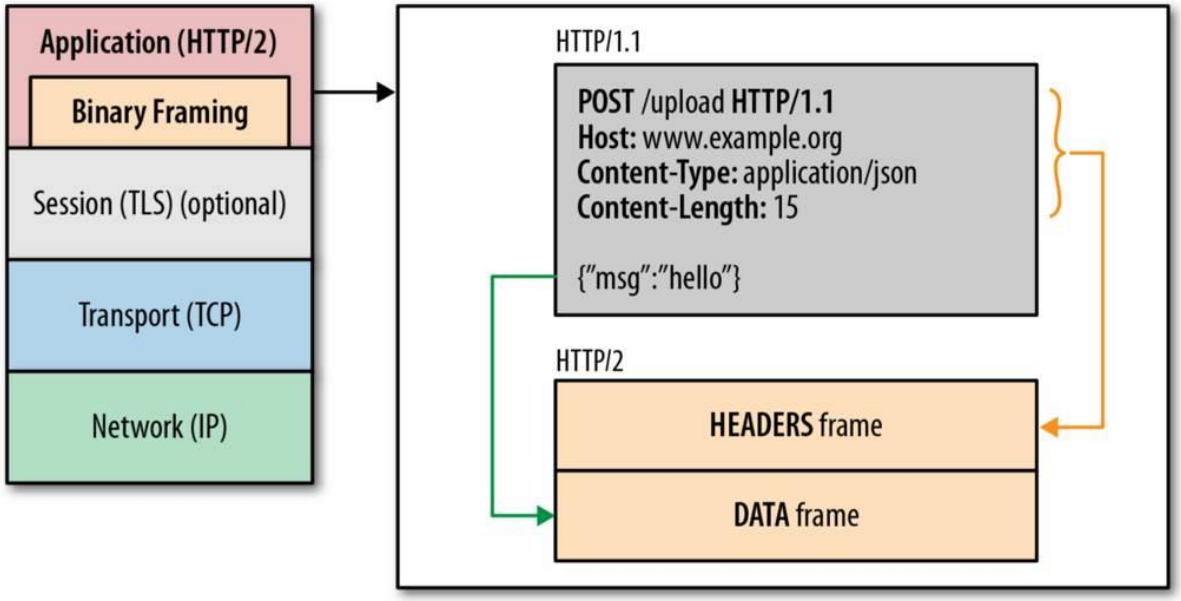
HTTP2

https://developers.google.com/web/fundamentals/performance/http2

Overview

HTTP/2

1. **One TCP connection**
2. **Request → Stream**
   - Streams are multiplexed
   - Streams are prioritized
3. **Binary framing layer**
   - Prioritization
   - Flow control
   - Server push
4. **Header compression (HPACK)**



HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP1.1: introduced multiple, pipelined GETs over single TCP connection

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (head-of-line (HOL) blocking) behind large object(s)
- loss recovery (retransmitting lost TCP segments) stalls object transmission

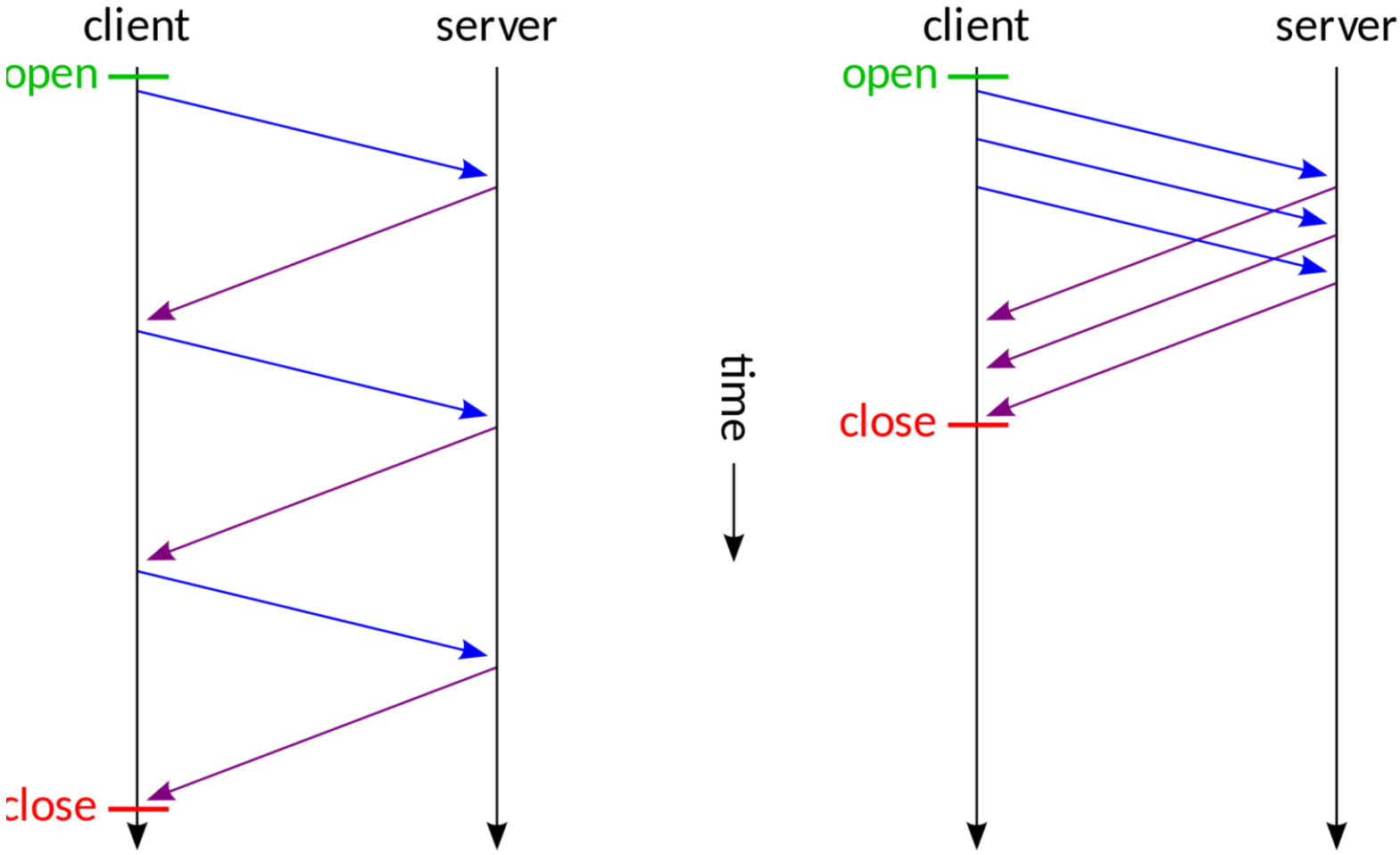
HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:

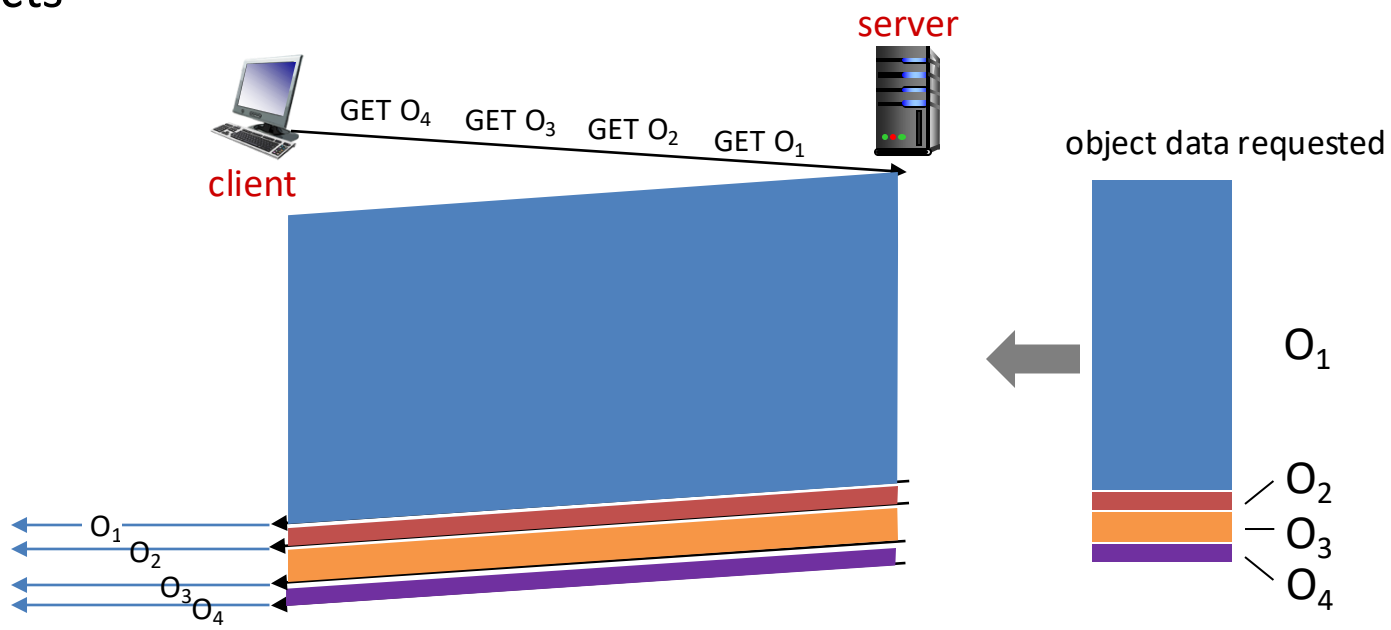
- methods, status codes, most header fields unchanged from HTTP 1.1
- transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
- *push* unrequested objects to client
- divide objects into frames, schedule frames to mitigate HOL blocking

Single TCP Connection



HTTP/2: mitigating HOL blocking

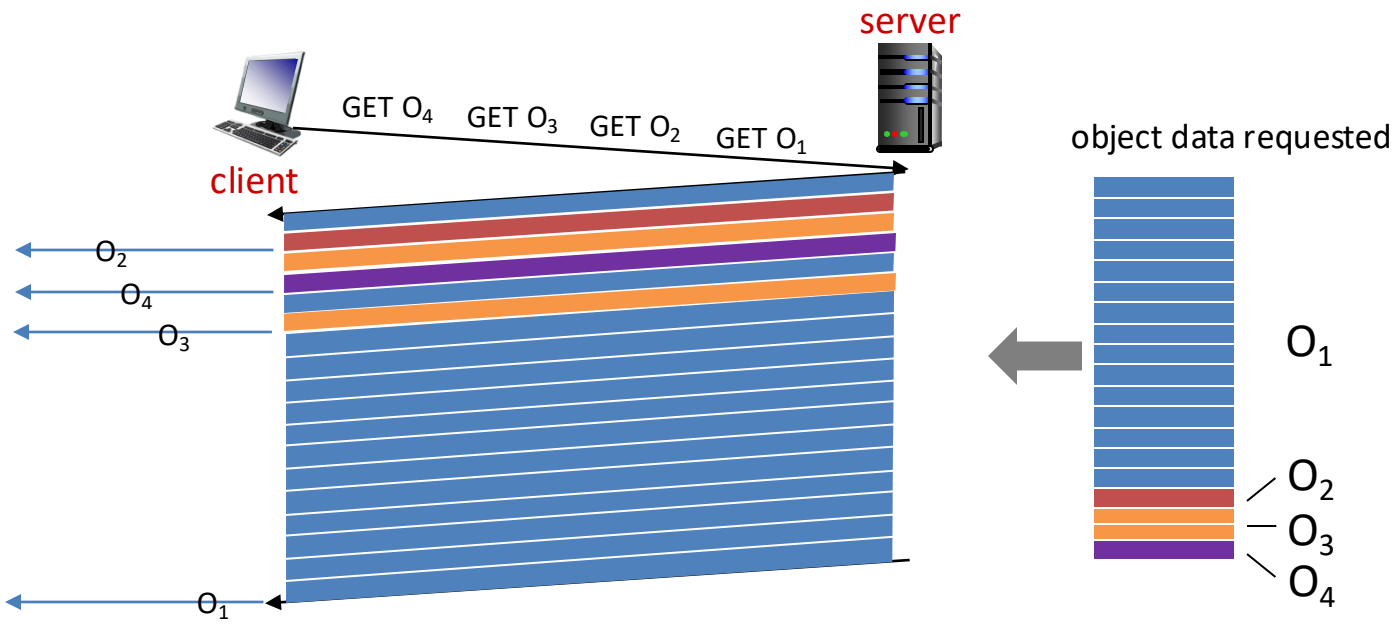
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

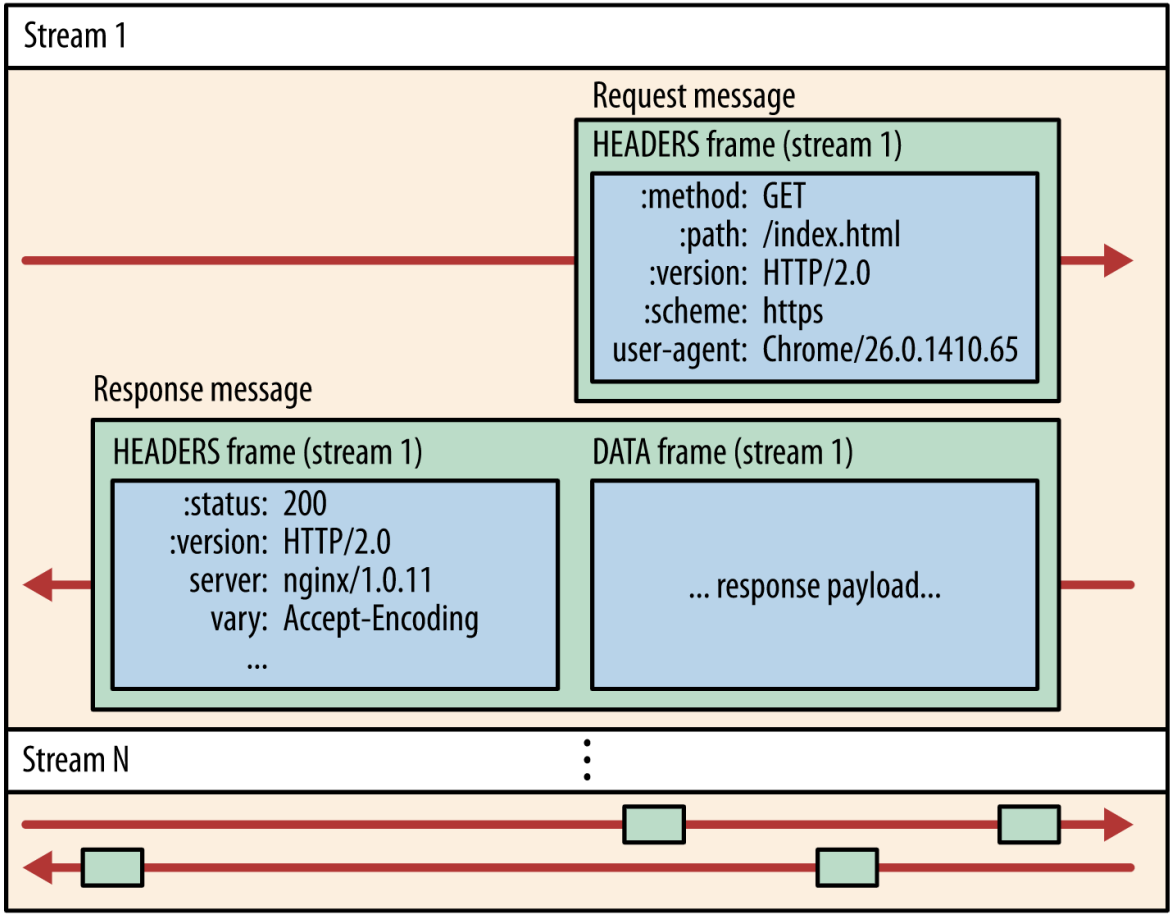
HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved

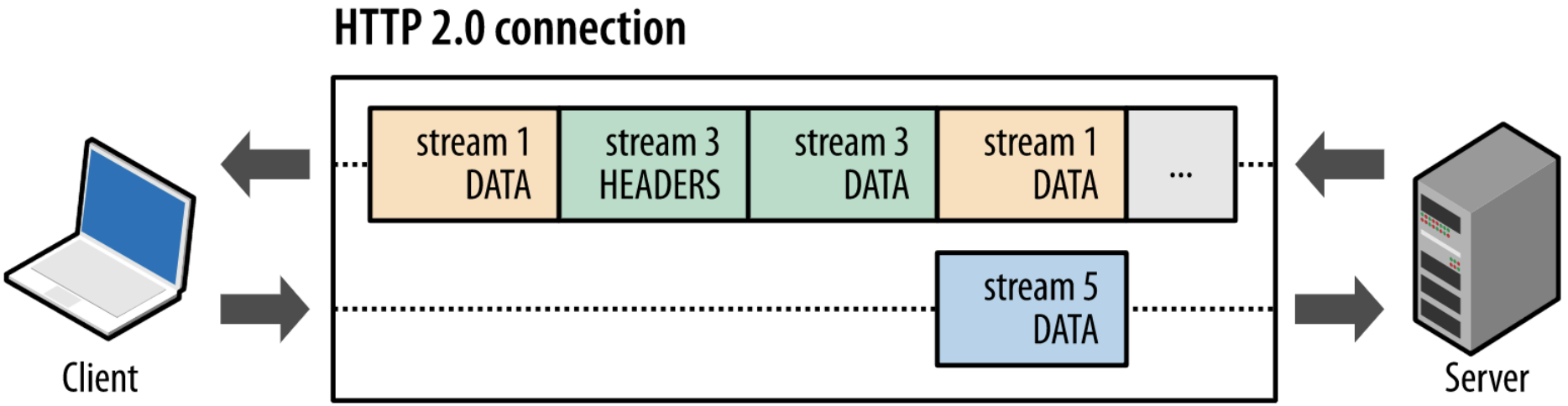


O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

Streams

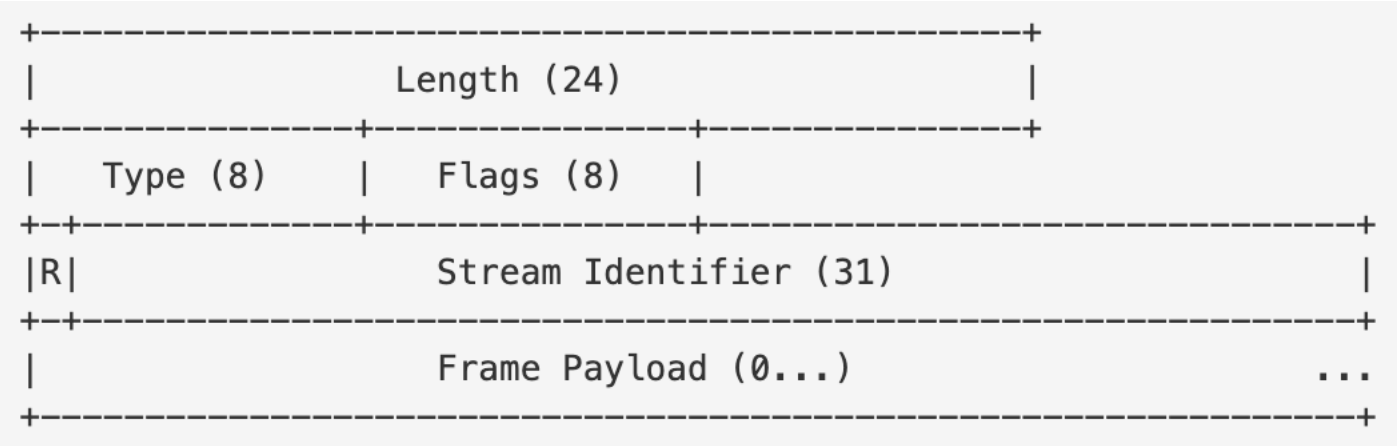

Stream 1
Request message
HEADERS frame (stream 1)
:method: GET
:path: /index.html
:version: HTTP/2.0
:scheme: https
user-agent: Chrome/26.0.1410.65
Response message
HEADERS frame (stream 1)
:status: 200
:version: HTTP/2.0
server: nginx/1.0.11
vary: Accept-Encoding
...
DATA frame (stream 1)
... response payload...
Stream N

Streams and frames



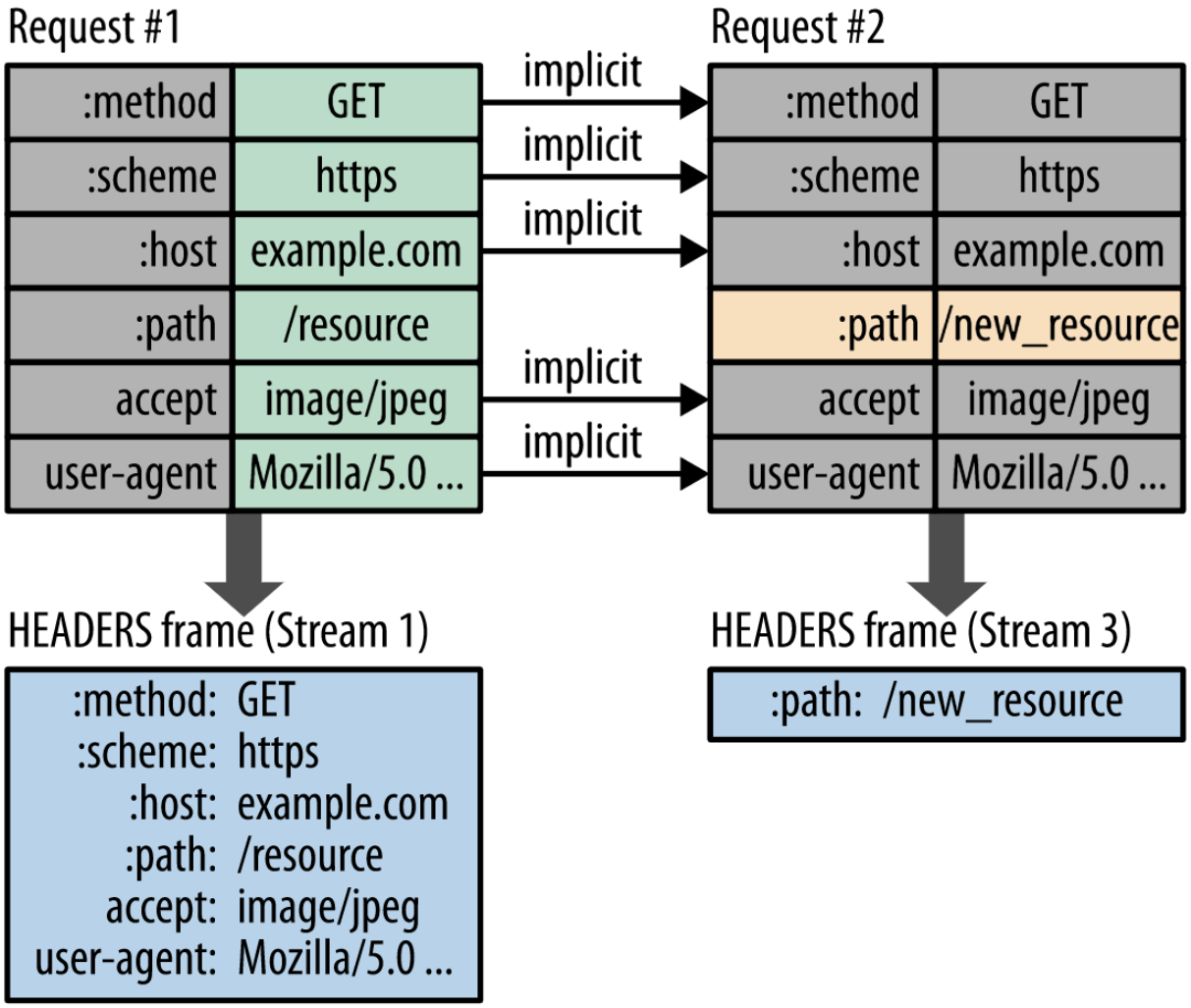
stream1: index.html
stream2: style.css
stream3: script.js

Binary Framing

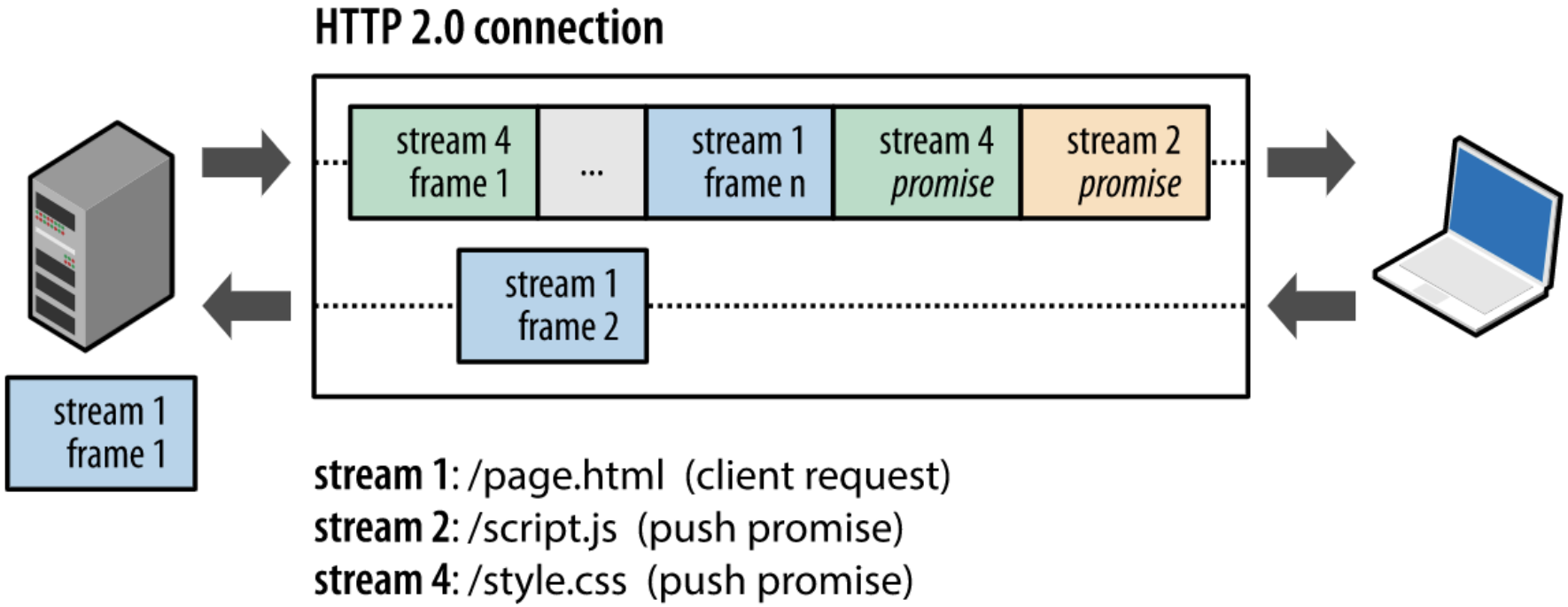


- **Length**: 24 bits storing the length of the frame payload
- **Type**: 8 bits storing the type of this frame. (**SETTINGS, HEADER, DATA)**
- **Flags**: 8 bits storing any boolean modifiers that apply to this frame type.
- **R**: 1 bit reserved for future use.
- **Stream Identifier**: 31 bits that uniquely identify each stream of this connection.
- **Frame Payload**: A variable length field containing the actual payload for this frame. The structure and content of the payload is dependent entirely on the frame type.

Header Compression

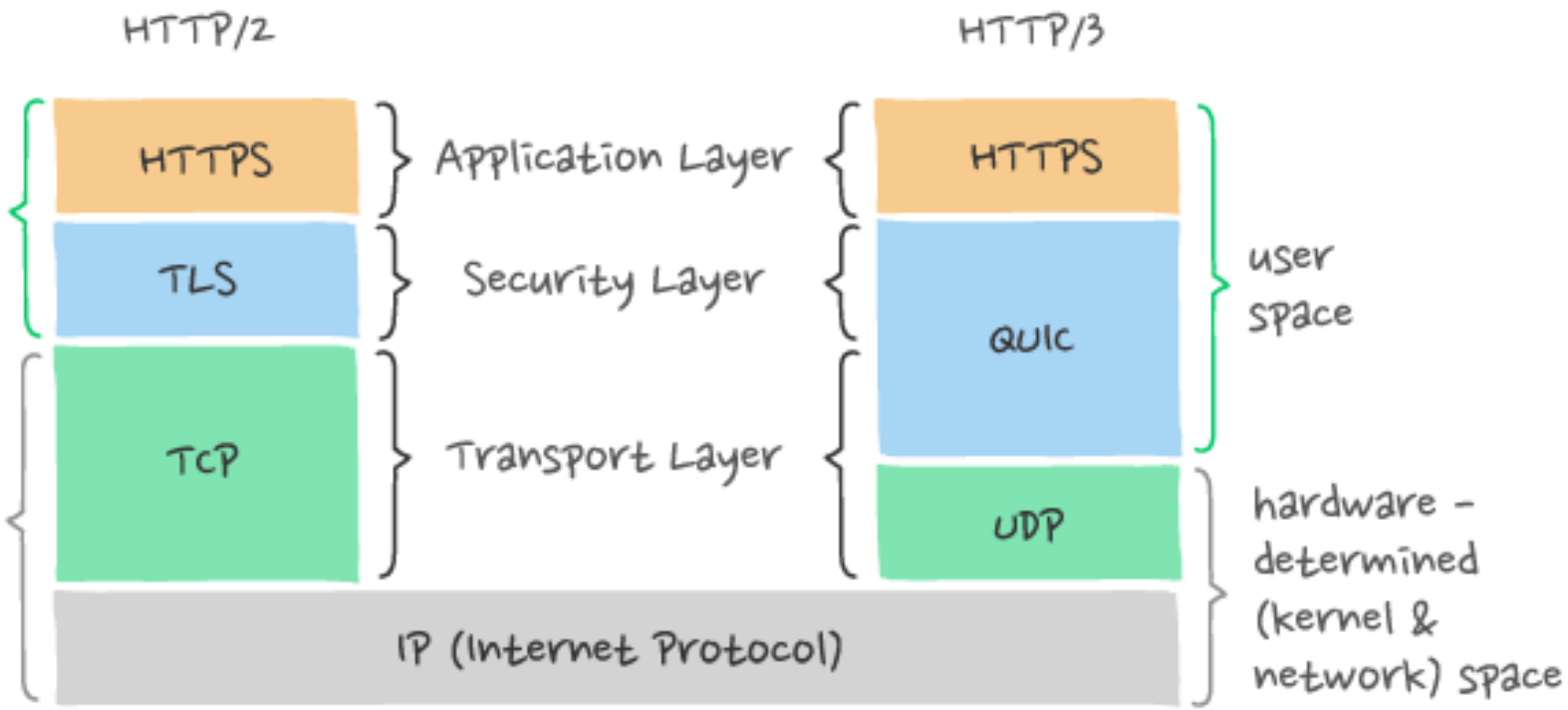


Server Push

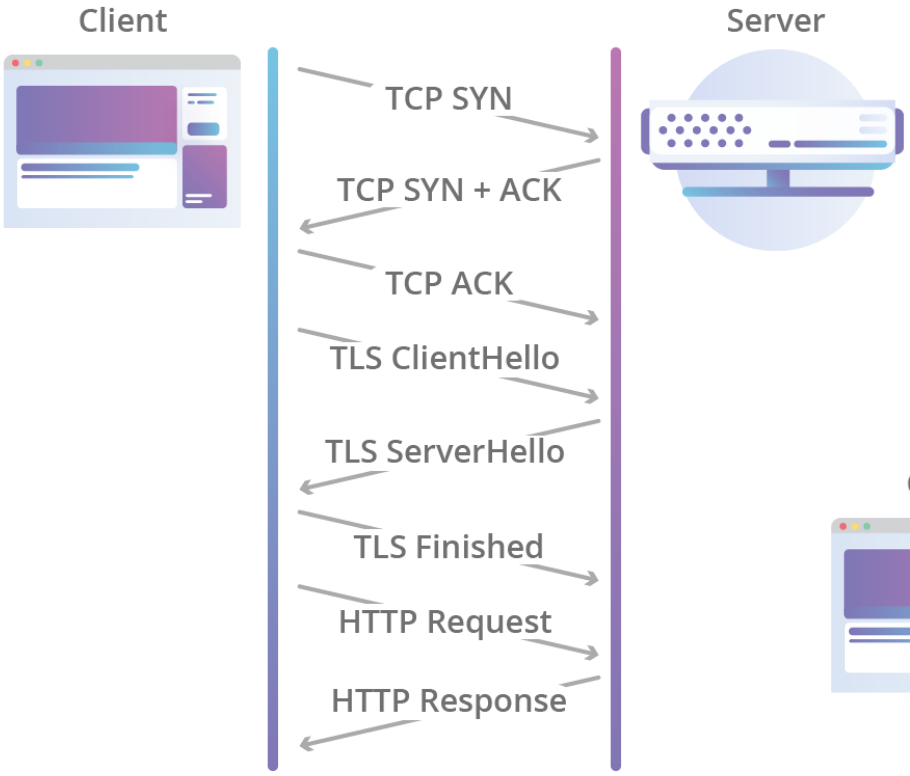

HTTP 2.0 connection
stream 4 frame 1
...
stream 1 frame n
stream 4 promise
stream 2 promise
stream 1 frame 2
stream 1 frame 1
stream 1: /page.html (client request)
stream 2: /script.js (push promise)
stream 4: /style.css (push promise)

HTTP 3

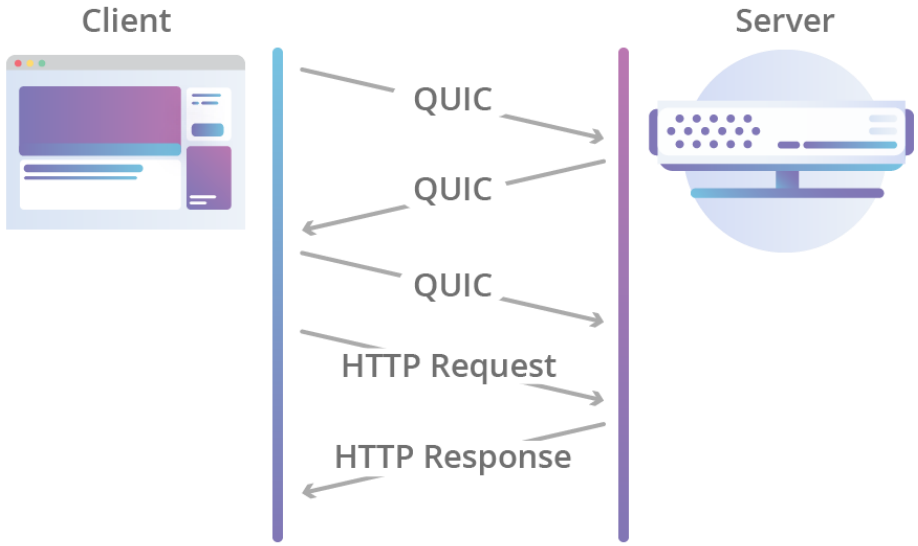
HTTP 3 Protocol Stack



HTTP Request Over TCP + TLS



HTTP Request Over QUIC



Network Switch