HTTP/2

Kevin Scrivnor

COMP 429

Fall 2023

Review

- HTTP is a request/response model for delivering web applications
- HTTP is an ASCII based protocol
- Statistics (Spring 2023)
  - Total kilobytes: median 2340
  - Total request: median 71
  - https request: 95%
  - TCP conn. Per page: median 12
  - HTTP/2 requests: 67.5%
  - (new stat) HTTP/3 support: 18.7%
  - https://httparchive.org/reports/state-of-the-web

Statistics (Spring 2021)

Total kilobytes: median 2038
Total request: median 73
https request: 89.3%
TCP conn. Per page: median 13
HTTP/2 requests: 66.9%

Statistics (Fall 2021)

Total kilobytes: median 2174
Total request: median 74
https request: 92.3%
TCP conn. Per page: median 14
HTTP/2 requests: 66.8%

Heart of the Internet

- TCP/IP was first proposed together in 1974 as the "Internet Protocol Suite"
  - *A solution for transmitting data across an unreliable channel.*
  - 1981, we have our two RFCs published
    - IPv4, RFC 791
    - TCP, RFC 793
- TCP
  - Complexity of dealing with the unreliable channel is abstracted away
    - Retransmission of lost data
    - In-order delivery (to the application)
    - Congestion control and avoidance
    - Data Integrity
    - Etc.
  - For now, we just need to know general concepts
    - *TCP does a thing*, not necessarily *TCP does a thing and this is how it works.*

TCP doesn’t necessarily help modern day web browsing

- HTTP standard does not state to use TCP
- In practice, all HTTP traffic is delivered over TCP
  - Though potentially that will change with QUIC which is still reliable

- TCP makes guarantees:
  - In order delivery of data from the server to the client
  - Mechanisms to avoid flooding the channel with data
  - An established connection

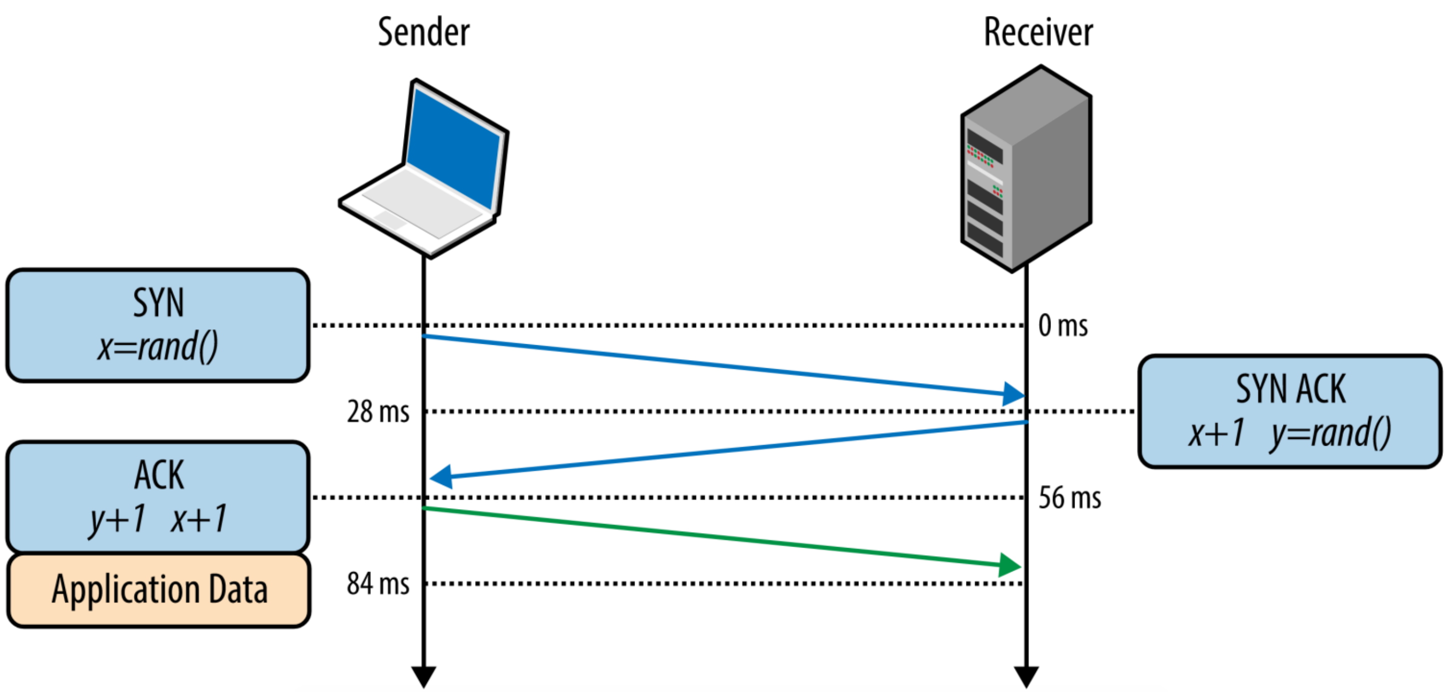
- But all of this comes at a cost!

TCP Handshake

- New York to London connection establishment
- `SYN, SYNACK, ACK`
- Before the data is sent

Route	Distance	Time, light in vacuum	Time, light in fiber	Round-trip time (RTT) in fiber
New York to San Francisco	4,148 km	14 ms	**21 ms**	42 ms
New York to London	5,585 km	19 ms	**28 ms**	56 ms
New York to Sydney	15,993 km	53 ms	**80 ms**	160 ms
Equatorial circumference	40,075 km	133.7 ms	**200 ms**	200 ms

- Connection establishment is expensive
- Critical for HTTP to *reuse connections* that are already established



- *Note the delay before the data is sent*

Congestion Avoidance and Control

- TCP has three mechanisms for preventing an application from overflowing the network.
  - Flow Control
  - Congestion Avoidance
  - Congestion Control

- Governs the rate at which data is placed onto the network.

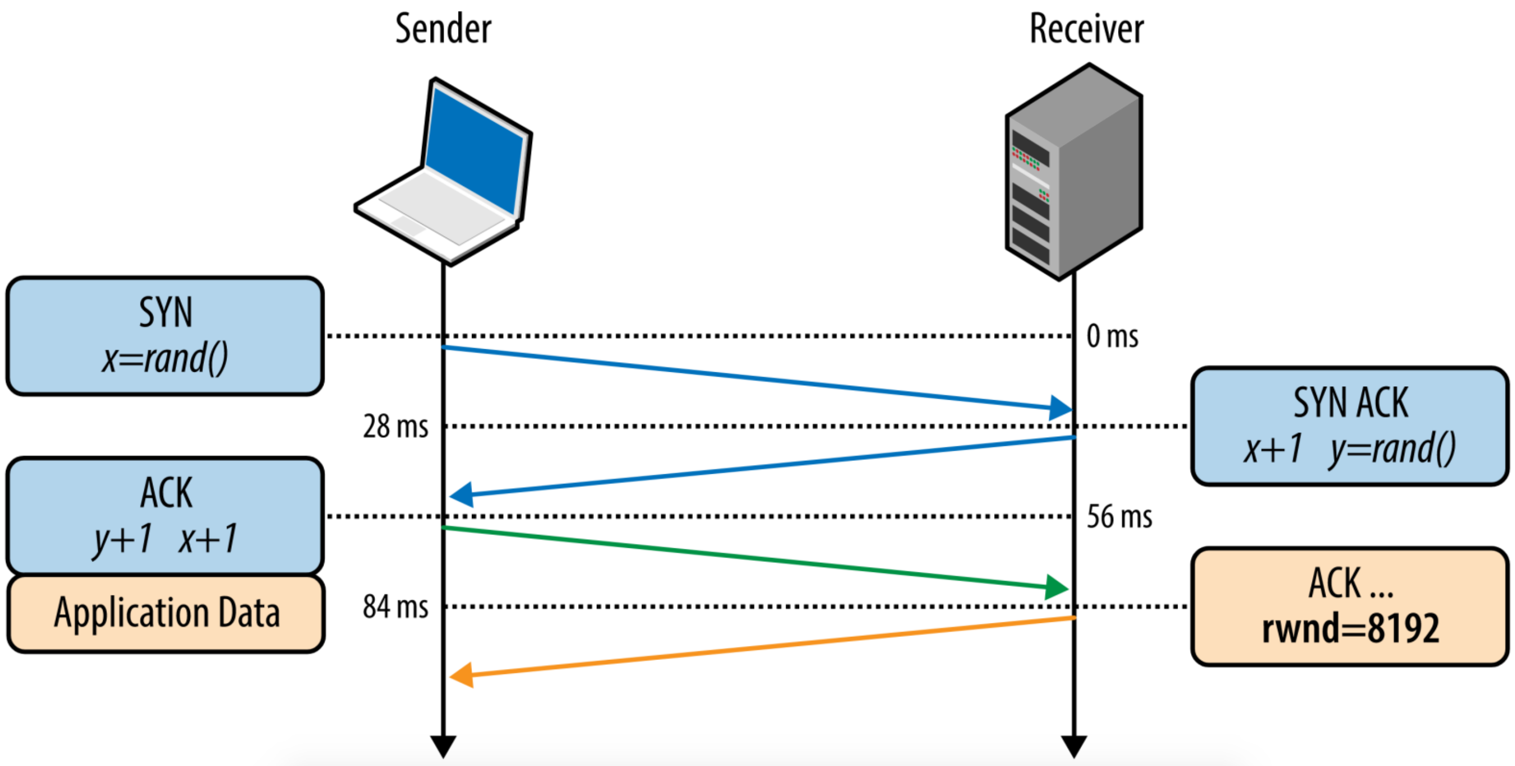
- Which means, TCP governs the rate at which an application sends data.

Flow Control

- *Prevent the sender from overwhelming the receiver with data it cannot process.*

- Done through advertising a window of availability

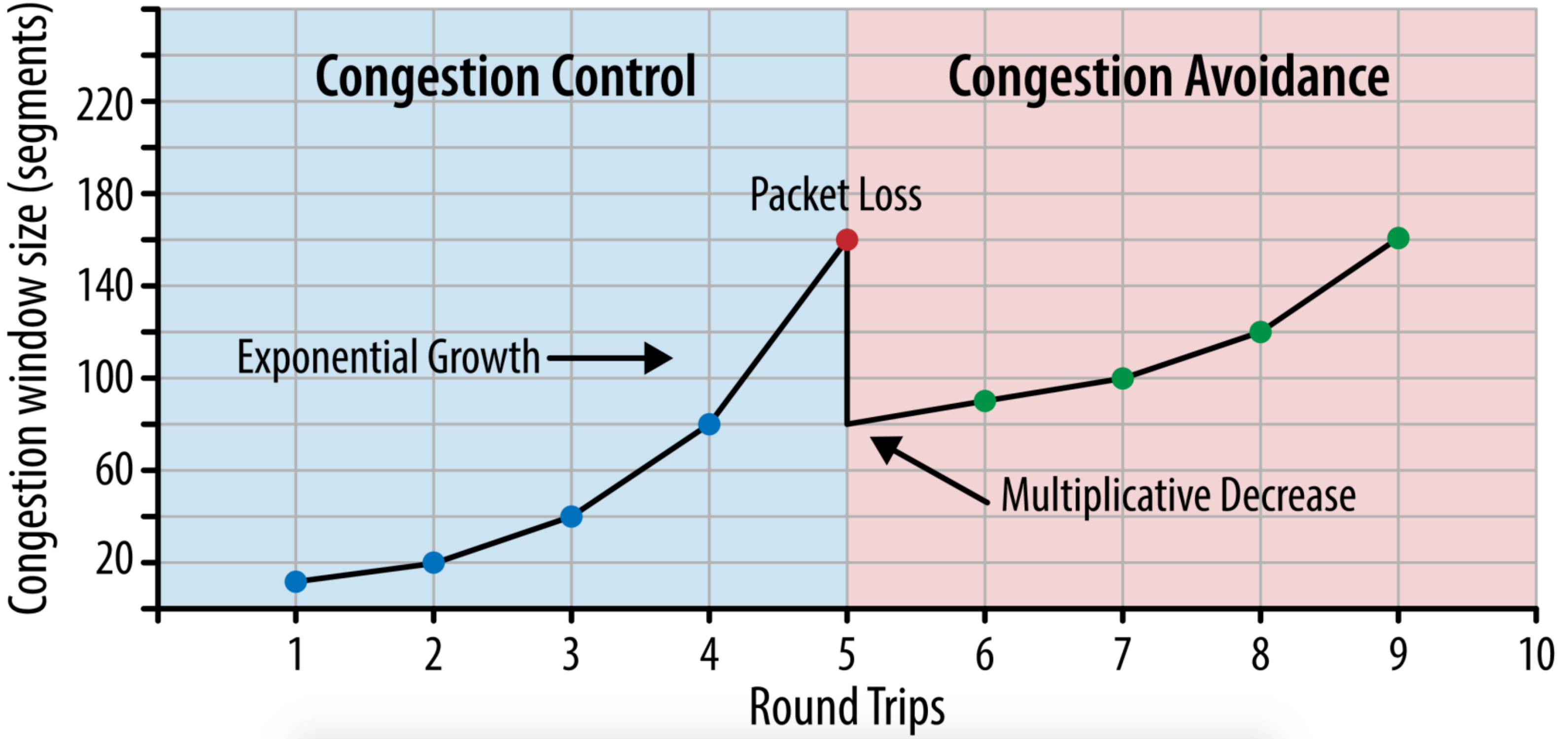
- How many bytes are available to be recv'd
- Both Sender/Receiver will advertise these windows



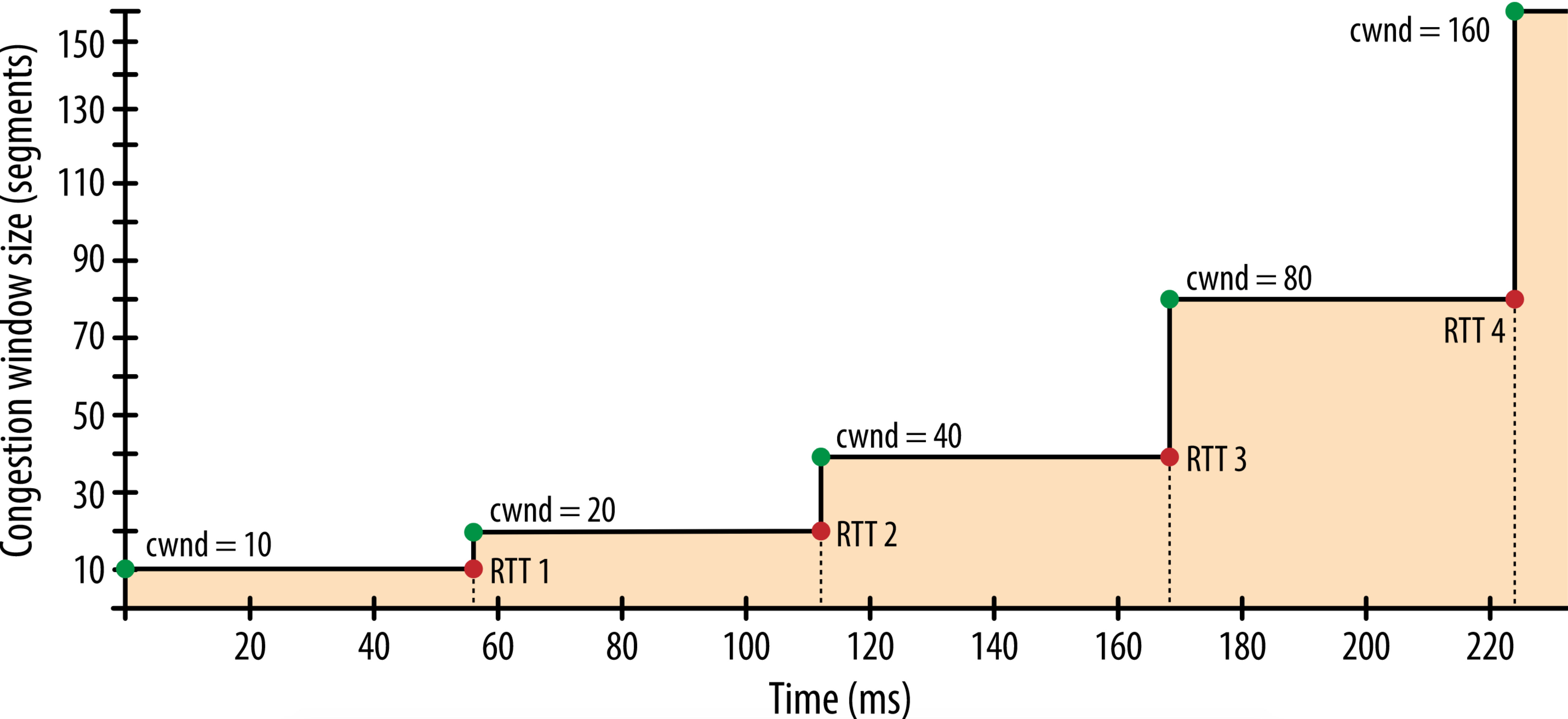
(sort of) Slow Start

- Start slow, find out where the network starts to break down (quickly), hang out around that area window size
- Turns out, maybe not quick enough.
- HTTP tends to be “*bursty*”: many small-ish files needed to complete a web application.

- Every new connection needs to start this process anew.



Congestion window size doubling every RTT

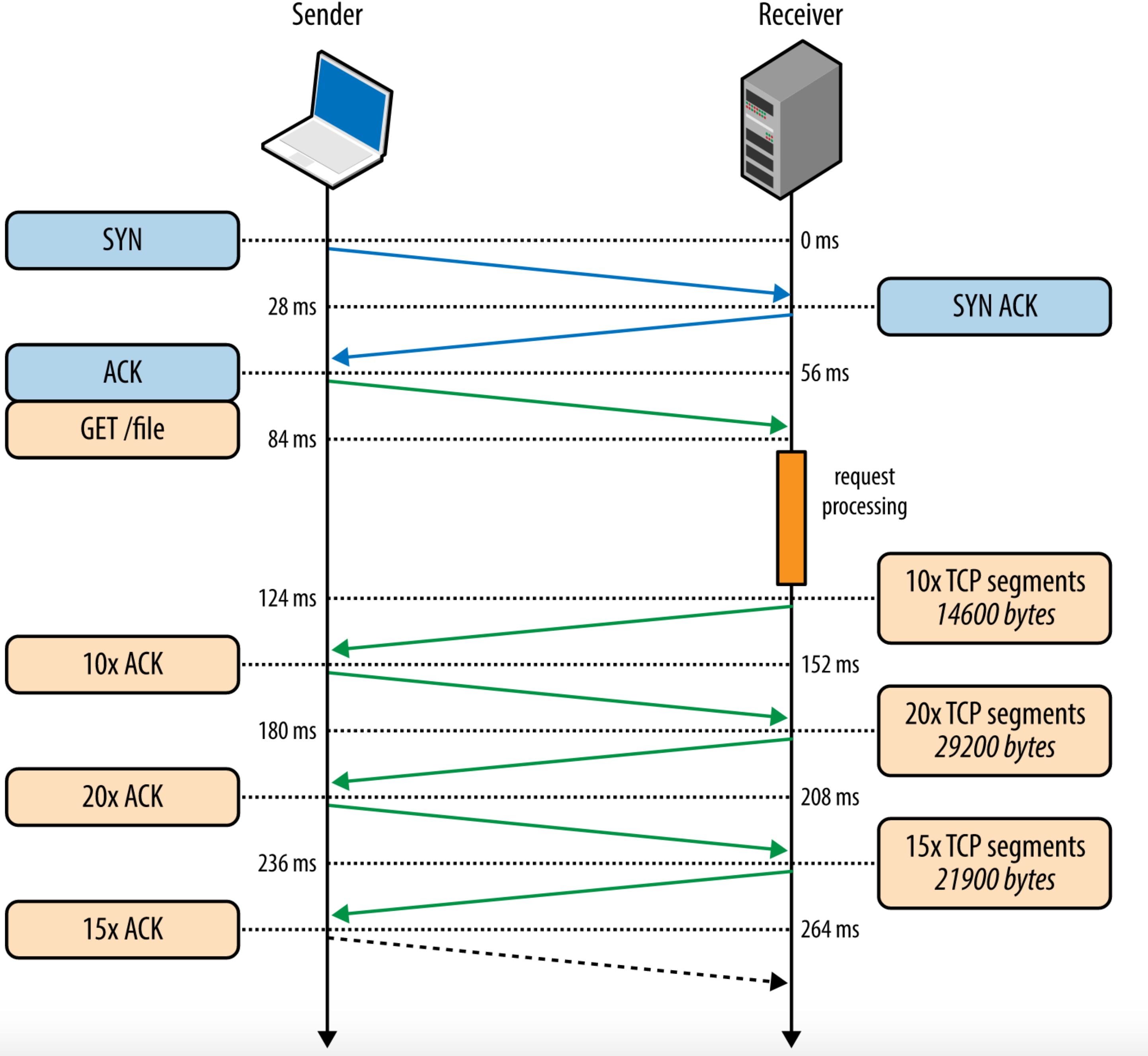


In a world where….

- RTT ~= 56 ms
- Client/Server Bandwidth: 5mbps
- Client/Server recv window: 64k bytes
- Initial Congestion Window: 10 segments (1460 bytes per segment)
  - 14k bytes per 10 segments
- Server processing time ~= 40ms
- *Assume:*
  - *No packet loss*
  - *ACK per packet*
  - *GET request fits in single segment*

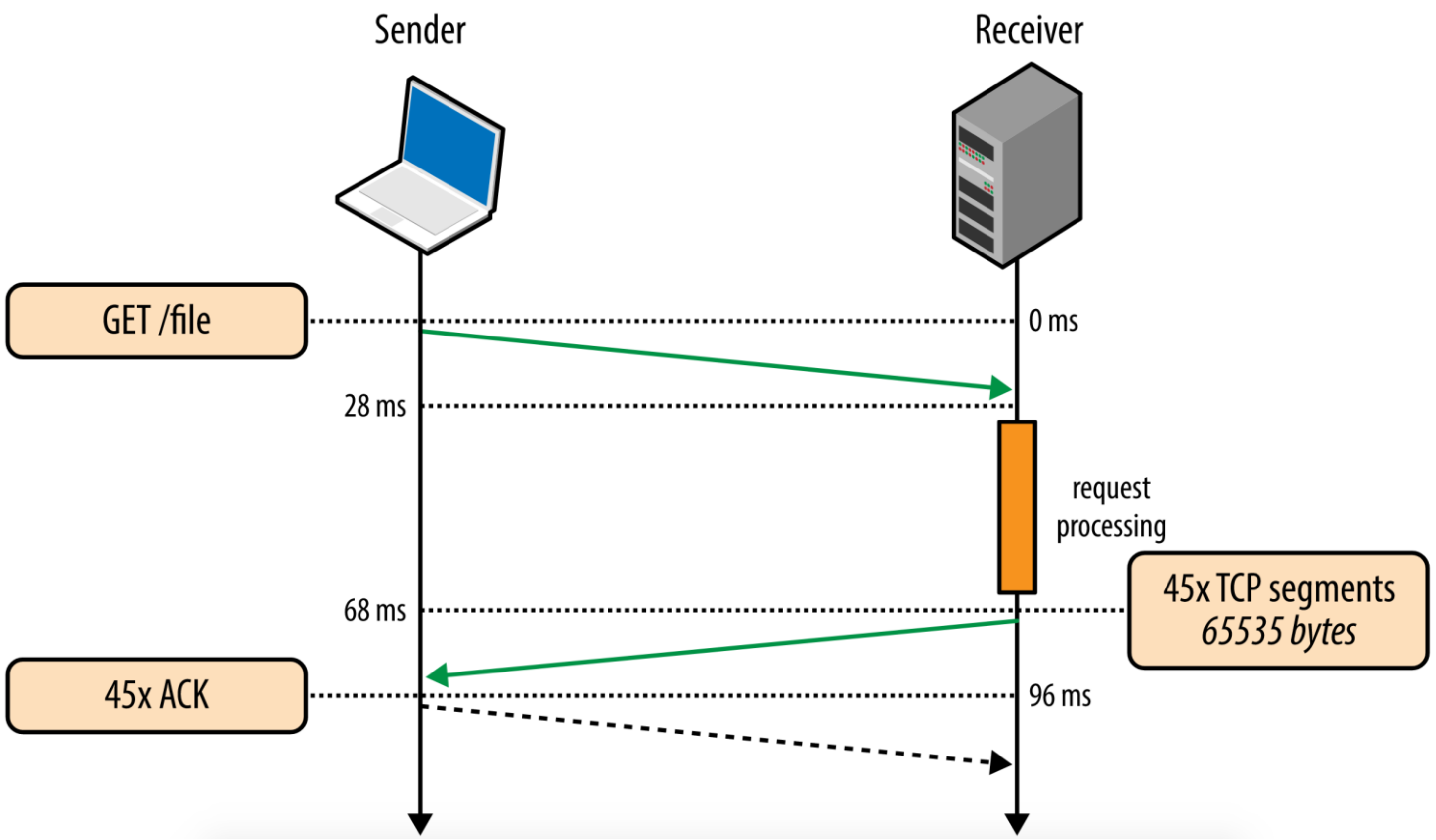
- *Up next: Server wants to respond with 64k bytes of data. So what happens?*

- 0 ms Client begins the TCP handshake with the SYN packet.
- 28 ms Server replies with SYN-ACK and specifies its rwnd size.
- 56 ms Client ACKs the SYN-ACK, specifies its rwnd size, and immediately sends the HTTP GET request.
- 84 ms Server receives the HTTP request.
- 124 ms Server completes generating the 64 KB response and sends 10 TCP segments before pausing for an ACK (initial cwnd size is 10).
- 152 ms Client receives 10 TCP segments and ACKs each one.
- 180 ms Server increments its cwnd for each ACK and sends 20 TCP segments.
- 208 ms Client receives 20 TCP segments and ACKs each one.
- 236 ms Server increments its cwnd for each ACK and sends remaining 15 TCP segments.
- 264 ms Client receives 15 TCP segments, ACKs each one.

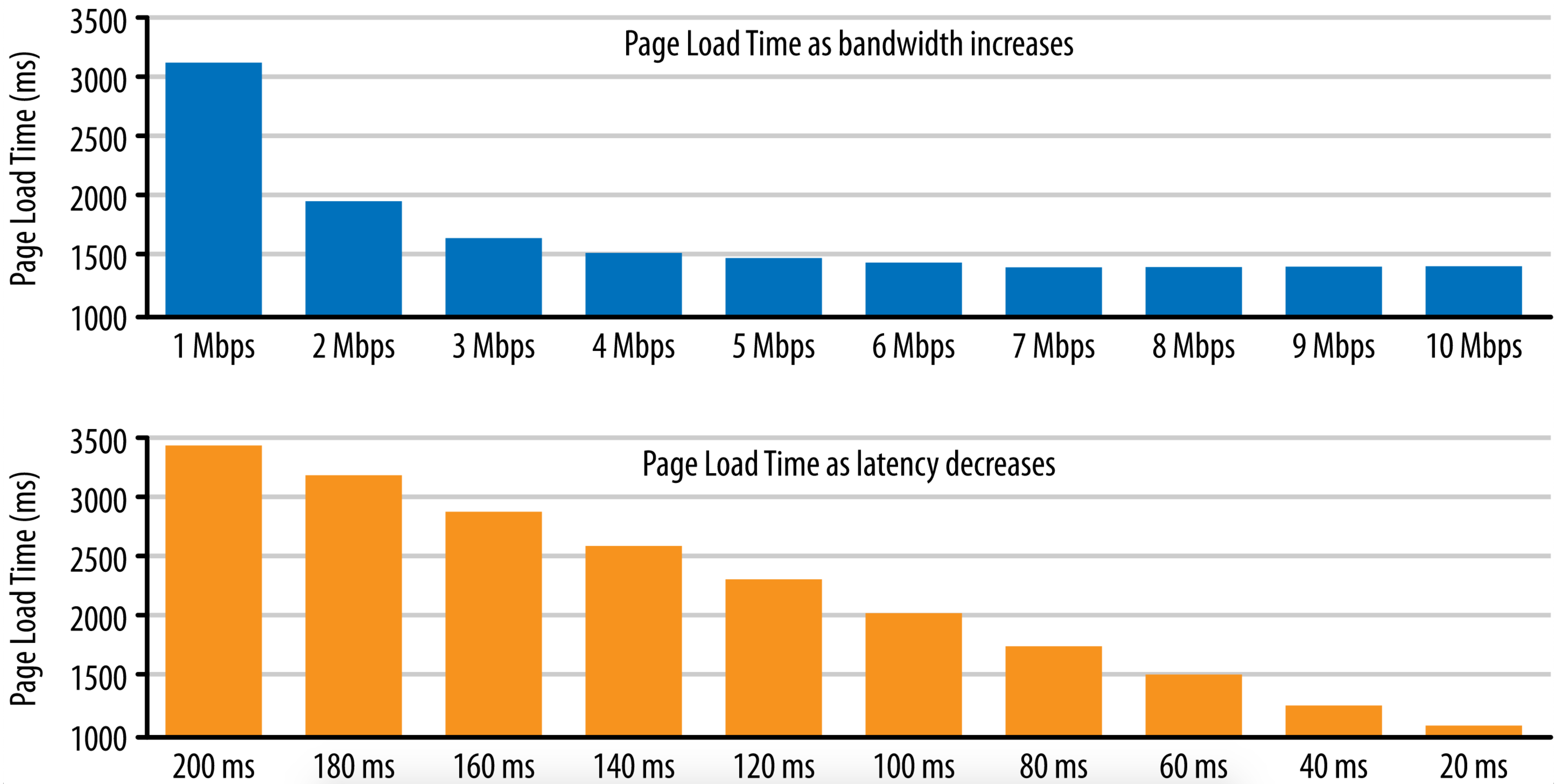


and if we reused that connection for another 64k file

- 0 ms Client sends the HTTP request.
- 28 ms Server receives the HTTP request.
- 68 ms Server completes generating the 64 KB response, but the cwnd value is already greater than the 45 segments required to send the file; hence it dispatches all the segments in one burst.
- 96 ms Client receives all 45 segments, ACKs each one.



Bandwidth vs. Latency



Lesson

- Bandwidth was irrelevant here (especially since we live in the future)
  - Latency is royalty (updated from Latency is king, then again…down with the monarch!)
  - Latency is the thing that matters the most!
  - The time it takes for a packet to get from one place to another
- Same request
  - 275% improvement in performance
- Lesson:
  - *Reusing connections has huge payoffs*
  - *Why use multiple connections for sending streams of data?*

Side note: linux kernel

- You can view various TCP settings the kernel has with the command:
- $ sysctl –a | egrep net.ipv4

- sysctl net.ipv4.tcp_slow_start_after_idle
  - TCP will reset slow start if a connection has been idle
  - This setting can be disabled
- sysctl net.ipv4.tcp_window_scaling
  - Check if window scaling is enabled.

- Do not blindly modify kernel settings… ask an expert
- Keep your kernels updated.

Congestion Avoidance

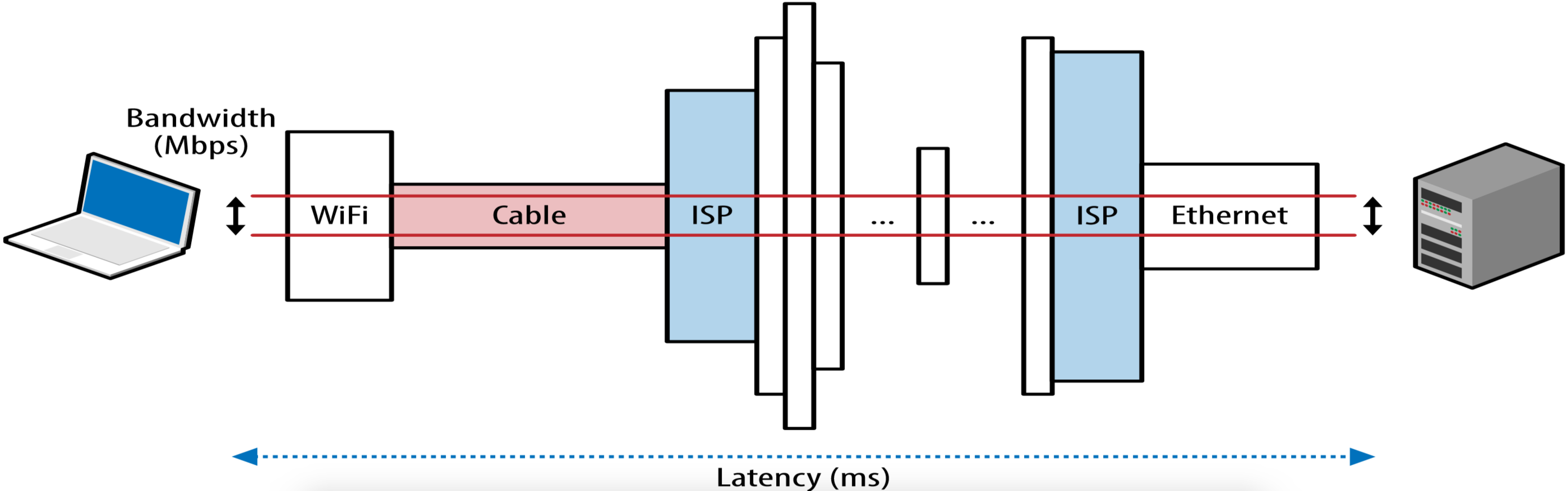
- *Packet loss will happen*
  - Somewhere along the way we encounter a router, or some other node that is overwhelmed.
  - The connection must the adjust the window to avoid further packet loss

- Congestion Avoidances kicks in after slow start to grow the window but avoid further loss.
  - After exponential, small increases to the window size

- Still an active area of research
  - Kernel changes
  - Tweaks available
  - Money maker

onto bandwidth and latency

- Speed costs money
  - 2015, Hibernia spent 300+ million dollars to create a new route between New York at London
  - ~5 ms faster than another other transatlantic link
  - **$60,000,000 per millisecond saved ($60,000,000 for 1ms smaller ping)**
  - Financial markets happy to pay this to have an advantage over competitors



Why we care about delay

General rule of thumb is a web page should be displayed within 250ms otherwise the user may feel something is off.

Delay	User perception
0–100 ms	Instant
100–300 ms	Small perceptible delay
300–1000 ms	Machine is working
1,000+ ms	Likely mental context switch
10,000+ ms	Task is abandoned

HTTP/2

- Goals (incomplete list)
  - **Reduce latency** with full multiplexing
  - **Minimize overhead** with efficient compression of HTTP headers
  - Add support for prioritization and server push
- HTTP semantics are not modified
  - Same HTTP methods, status codes, URIs, headers, etc.
  - Smarter Data Formatting instead
  - Result: any application can be upgraded from 1.1 -> 2 without modification
- Performance in mind
  - We can still improve performance

Timeline

- March 2012: Call for proposals for HTTP/2
- November 2012: First draft of HTTP/2 (based on SPDY)
- August 2014: HTTP/2 draft-17 and HPACK draft-12 are published
- August 2014: Working Group last call for HTTP/2
- February 2015: IESG approved HTTP/2 and HPACK drafts
- May 2015: RFC 7540 (HTTP/2) and RFC 7541 (HPACK) are published

SPDY developed at Google

Drama

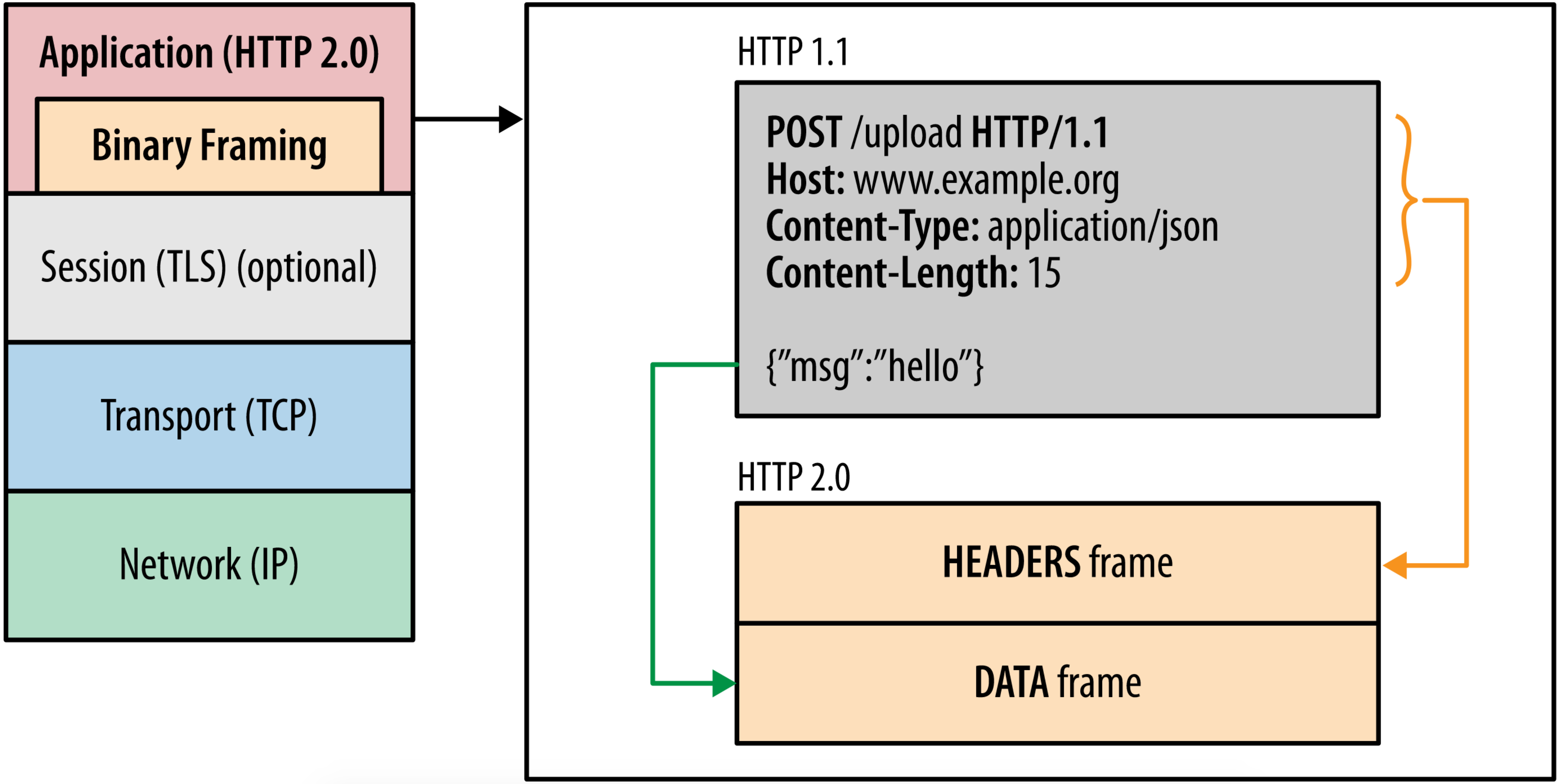
- Relatively short timeline, did others have a chance to implement an alternative?
  - Microsoft alternative:
  - https://en.wikipedia.org/wiki/HTTP_Speed%2BMobility
  - Facebook's response: https://lists.w3.org/Archives/Public/ietf-http-wg/2012JulSep/0251.html
  - Network-Friendly HTTP Upgrade (alternative): https://tools.ietf.org/html/draft-tarreau-httpbis-network-friendly-00
    - FB's response contains logical and sound reasoning for not considering this alternative
- Encryption as a Requirement?
  - CPU requirements
  - Which encryption protocol?
  - Working group decided against it, though all clients who implement HTTP/2 have made it mandatory in the end.
  - Makes our HTTP/2 captures a little more painful since we have to break encryption

Drama

- BINARY PROTOCOL?!?!?
  - HTTP/2 is no longer an ASCII Protocol
  - Meaning, you need a tool to inspect HTTP/2 packets (like Wireshark)
  - This bothers some people I guess

- HTTP/3 is coming
  - HTTP/2 + TLS + QUIC
  - Switching from TCP -> UDP (sort of)
  - Discussed later in the course

Biggest Change: Binary Framing Layer



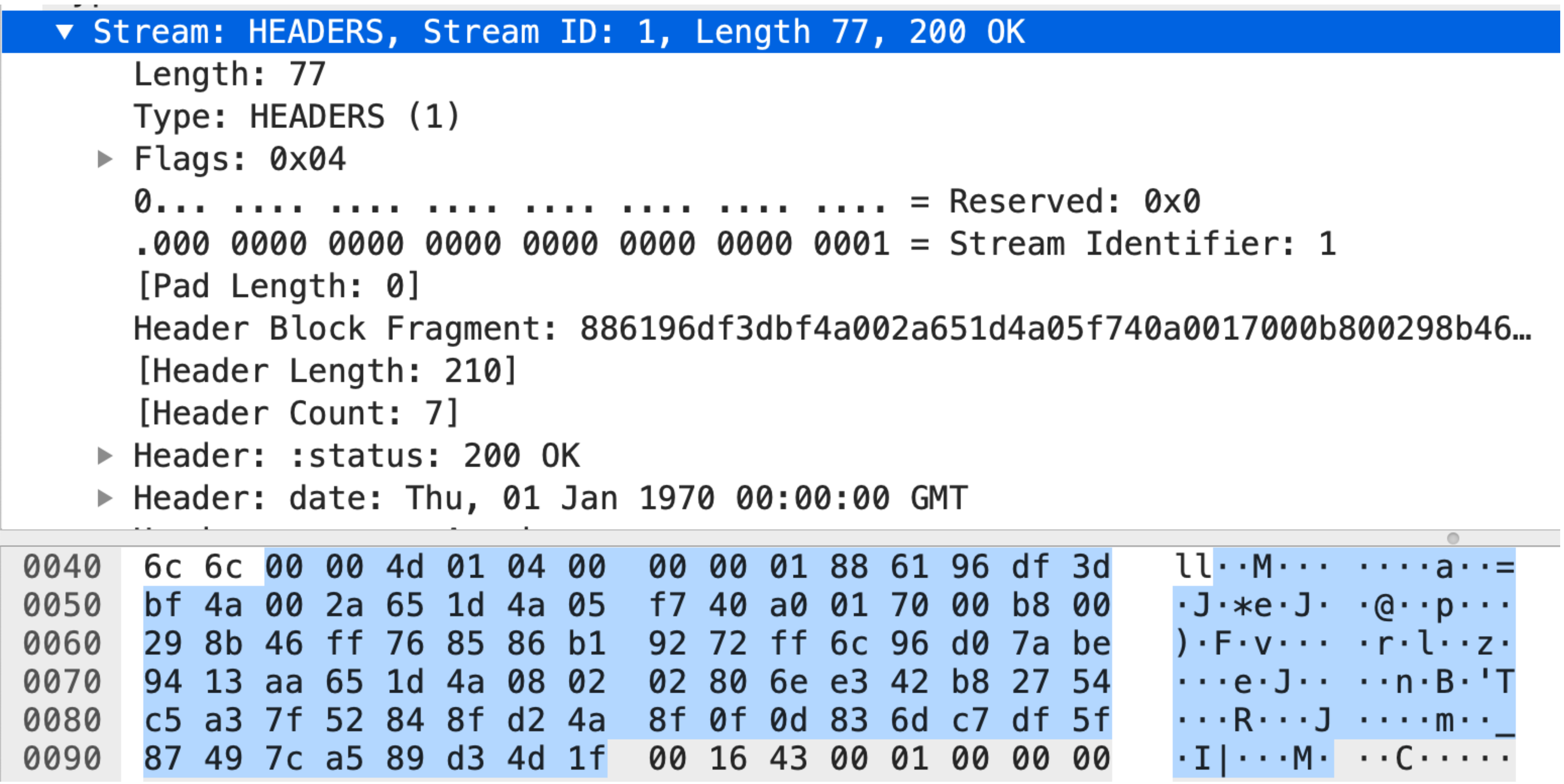
ASCII Protocol (HTTP/1.1)

```
▼ Hypertext Transfer Protocol
  ► GET / HTTP/1.1\r\n
    Host: scrivnor.cikeys.com\r\n
    User-Agent: curl/7.64.1\r\n
    Accept: */*\r\n
    Connection: Upgrade, HTTP2-Settings\r\n
    Upgrade: h2c\r\n
    HTTP2-Settings: AAMAAABkAARAAAAAAAIAAAAA\r\n
    \r\n
    [Full request URI: http://scrivnor.cikeys.com/]
    [HTTP request 1/1]
```

```
0030  08 0a fc ed 00 00 01 01  08 0a 51 c1 6c 6c 02 56   ········ ··Q·ll·V
0040  82 67 47 45 54 20 2f 20  48 54 54 50 2f 31 2e 31   ·gGET /  HTTP/1.1
0050  0d 0a 48 6f 73 74 3a 20  73 63 72 69 76 6e 6f 72   ··Host:  scrivnor
0060  2e 63 69 6b 65 79 73 2e  63 6f 6d 0d 0a 55 73 65   .cikeys. com··Use
0070  72 2d 41 67 65 6e 74 3a  20 63 75 72 6c 2f 37 2e   r-Agent:  curl/7.
0080  36 34 2e 31 0d 0a 41 63  63 65 70 74 3a 20 2a 2f   64.1··Ac cept: */
0090  2a 0d 0a 43 6f 6e 6e 65  63 74 69 6f 6e 3a 20 55   *··Conne ction: U
00a0  70 67 72 61 64 65 2c 20  48 54 54 50 32 2d 53 65   pgrade,  HTTP2-Se
00b0  74 74 69 6e 67 73 0d 0a  55 70 67 72 61 64 65 3a   ttings·· Upgrade:
00c0  20 68 32 63 0d 0a 48 54  54 50 32 2d 53 65 74 74    h2c··HT TP2-Sett
00d0  69 6e 67 73 3a 20 41 41  4d 41 41 41 42 6b 41 41   ings: AA MAAABkAA
00e0  52 41 41 41 41 41 41 41  49 41 41 41 41 41 0d 0a   RAAAAAAA IAAAAA··
```

Binary Protocol (HTTP/2)

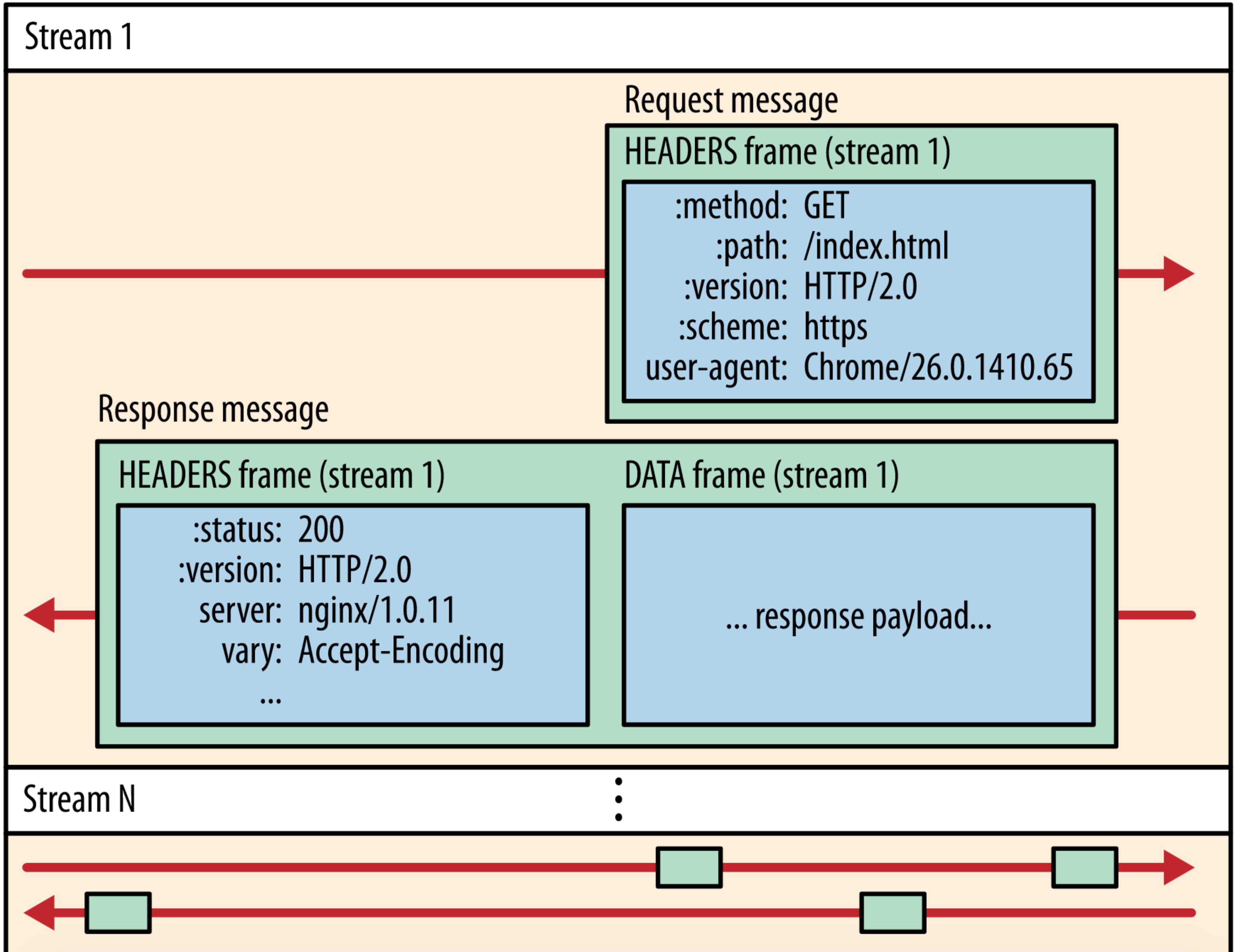
- HTTP Headers and Data are encoded in a binary format (not ASCII)



Key Terms

- **Stream**
  - Bi-directional flow of data, may carry one or more messages
- **Message**
  - A sequence of frames that map to a logical request/response.
- **Frame**
  - Smallest unit of communication.
  - Carries a specific type of data, headers, message payload, etc.

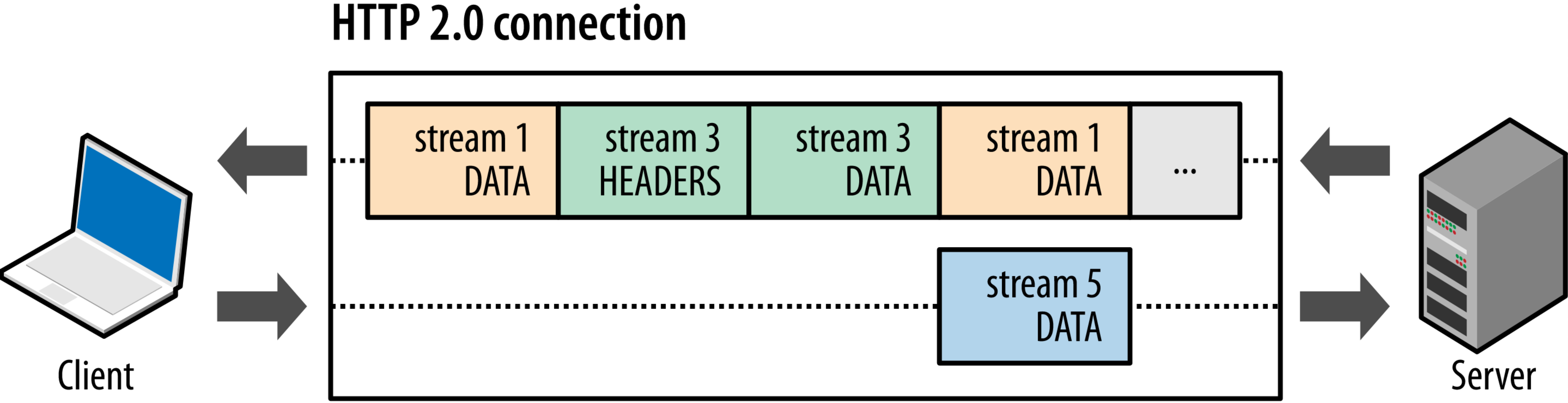
- A stream is a bi-directional flow of frames. Those frames are received and put back together to form the message.



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

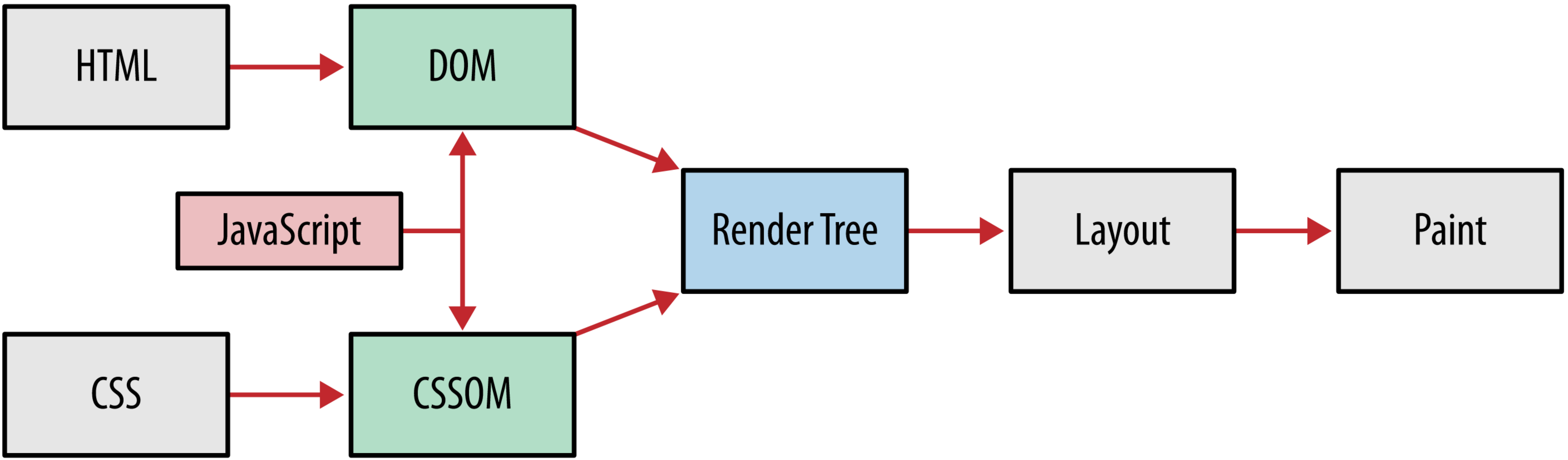
Simple change, many benefits

- By adding the ability to split messages up into frames that can be interleaved over a stream, we get
  - 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
  - Remove unnecessary HTTP/1.x workarounds
  - Deliver lower page load times by eliminating unnecessary latency and improving utilization



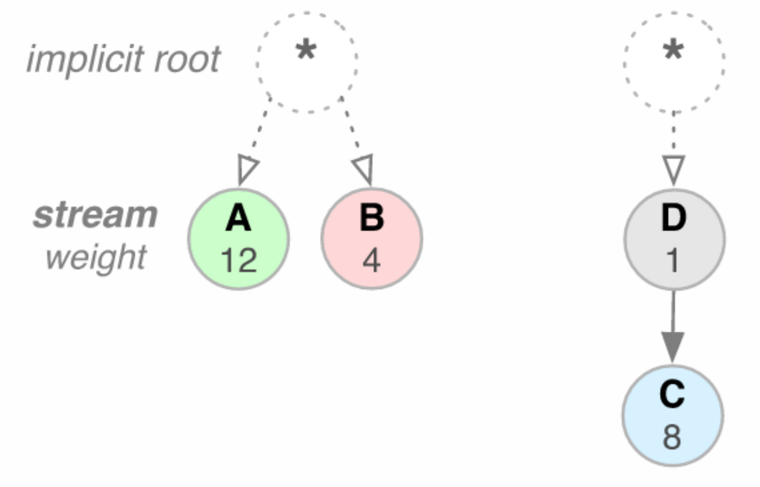
Stream Prioritization

- Recall, web applications have dependencies and order matters
- Stream dependencies (one stream depends on another)
- Stream priorities (one message has a higher priority over another)



Example: Prioritization Tree

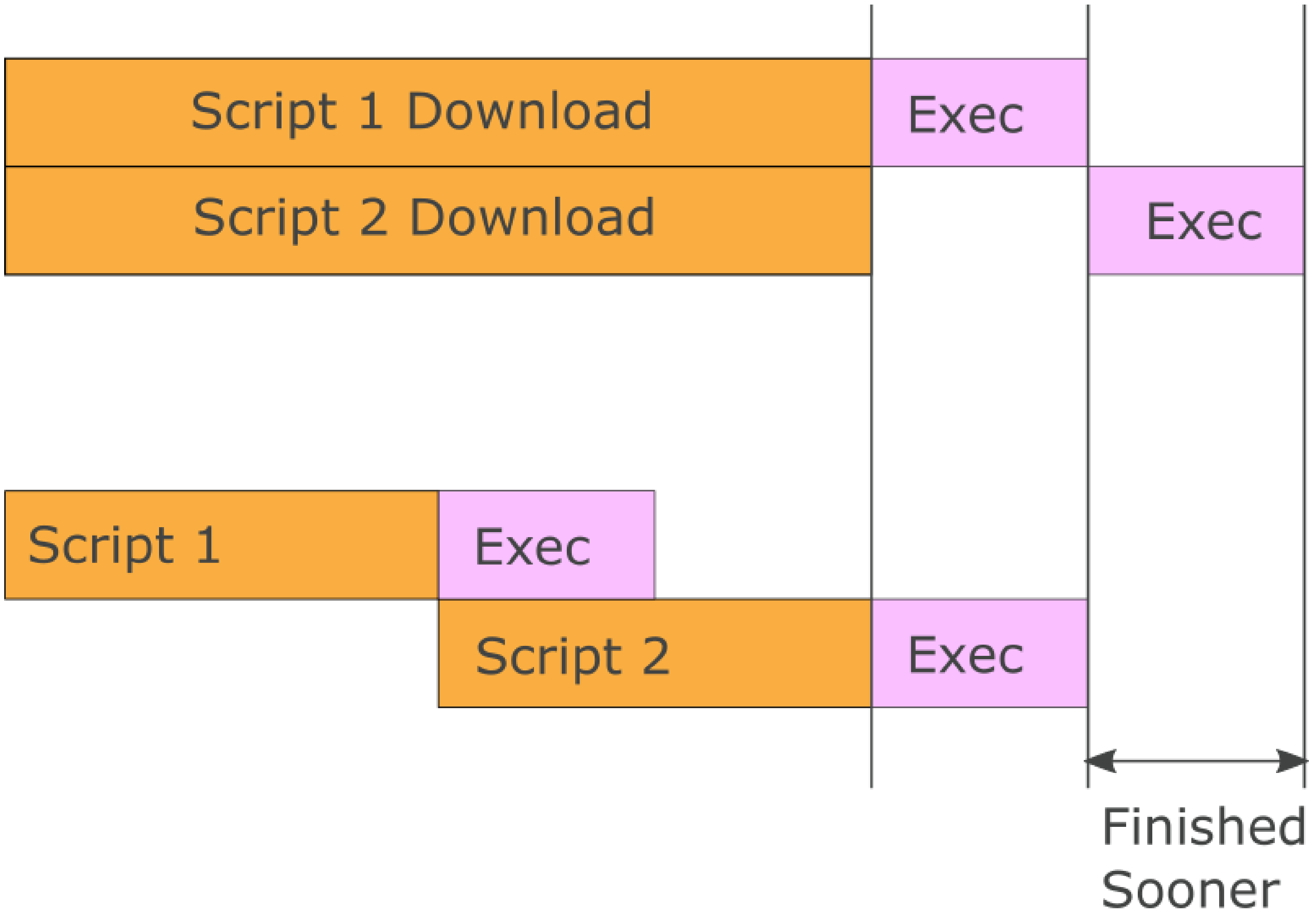
- A+B
  - Same dependency level in the tree
  - Total weight: W(A) + W(B) = 12 + 4 = 16
  - A then gets 12/16 of the bandwidth
  - B then gets 4/16 of the bandwidth
- D+C
  - C depends on D, therefore D gets full bandwidth until finished.



- Weights are simply integers 1-255
- Note: a client may update its priority *preference*, but it can't tell the server what to do.

- https://blog.cloudflare.com/better-http-2-prioritization-for-a-faster-web/

How does it help?



Just One TCP Connection (per origin)

- HTTP/2 only requires a single TCP connection
- Benefits
  - TCP designed for long-lived connections, HTTP is short/bursty
  - Fewer TLS handshakes
  - Overall reduction of resource demand on servers/clients
- TCP's head of line blocking not solved
  - During packet loss, TCP will hold onto data while trying to recover the missing
  - HTTP head of line blocking is solved by HTTP/2
- Packet loss reduces maximum throughput for entire connection now, not just one of twelve, say
- A well configured TCP on a server can lead to huge gainz
- TCP not the only choice, the future may be QUIC

HTTP/2 Flow Control

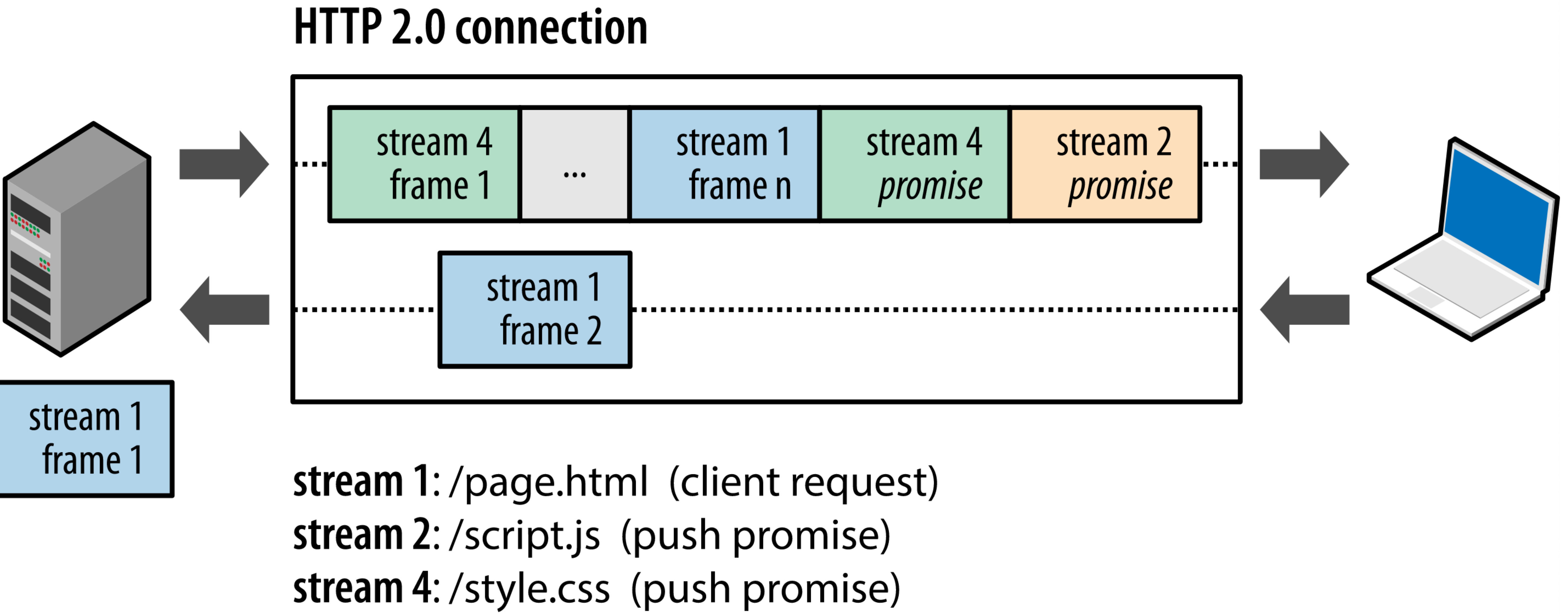
- Slightly controversial because the transport layer has flow control already.
  - Reimplementing features at a higher layer?

- The concept is roughly the same as TCP, so we will study this concept later in the course.

- For now, just know:
  - Directional
  - Credit based
  - Can be updated

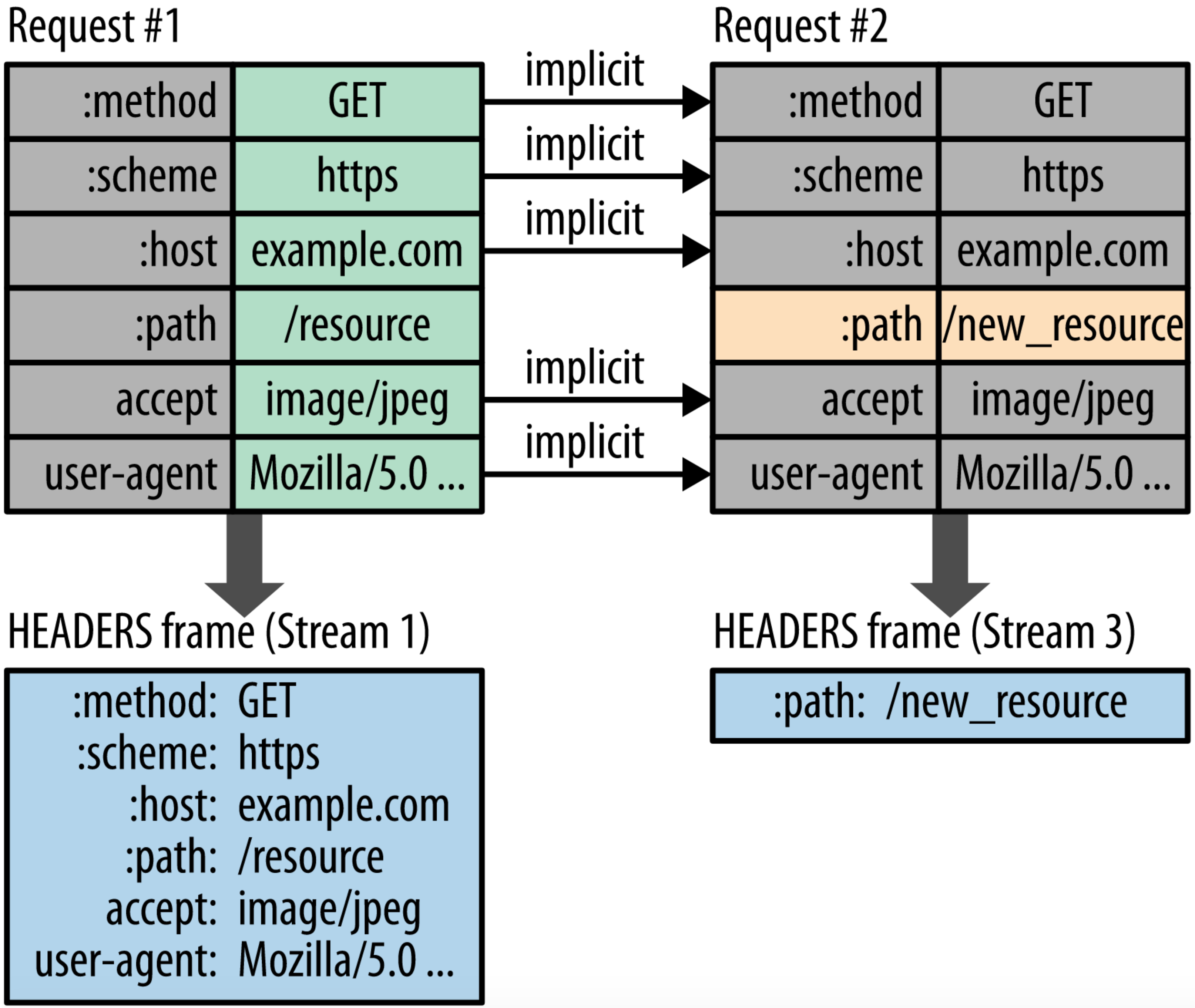
Server Push

- Client receives a promise, can accept or deny it



Optimizing headers

- Huffman encoding
- Implicit headers have no need to be sent, redundant
- Headers can add 500-800 bytes of data

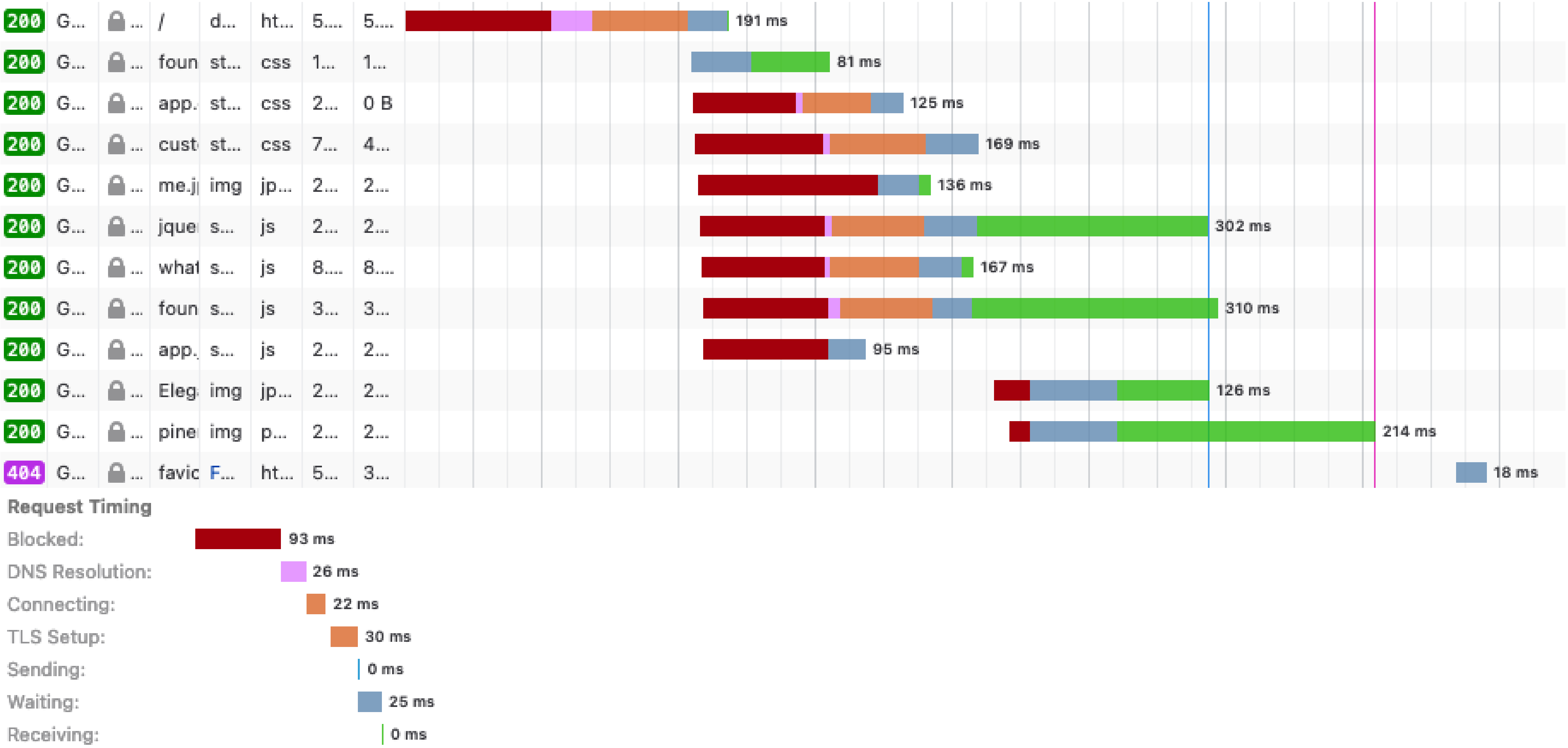


▶ HyperText Transfer Protocol 2

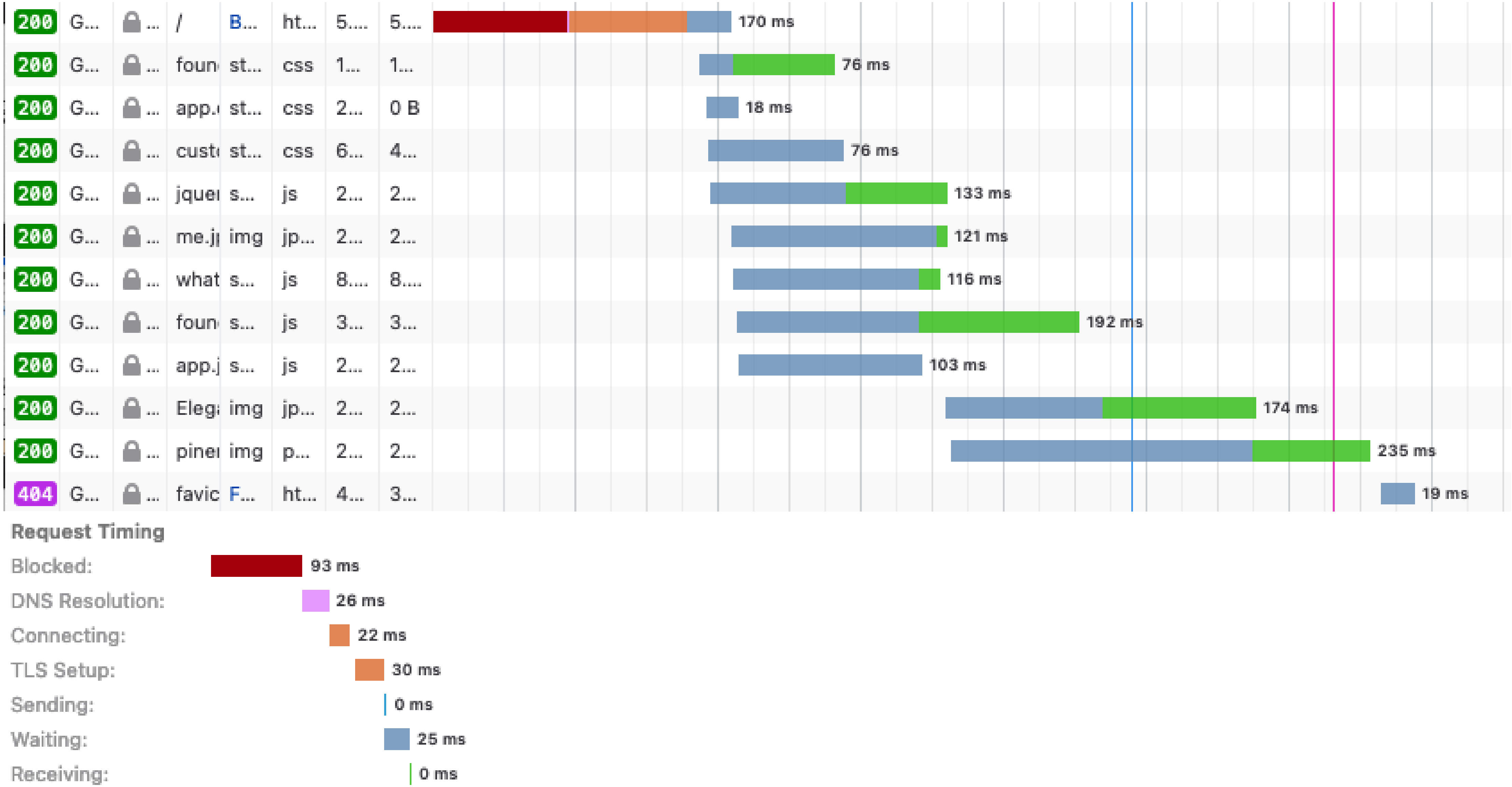
```
0000  8c 85 90 31 3c 75 fa 5b  3b 20 40 6d 08 00 45 00   ···1<u·[ ;  @m··E·
0010  00 45 6d 55 40 00 33 06  cd 52 8a c5 c0 4e c0 a8   ·EmU@·3·  ·R···N··
0020  01 4f 00 50 ca 2d 39 9b  ff 85 83 37 af 2a 80 18   ·O·P·-9·  ···7·*··
0030  00 eb ad fd 00 00 01 01  08 0a 02 be 85 78 52 28   ········  ·····xR(
0040  a1 49 00 00 08 07 00 00  00 00 00 00 00 00 01 00   ·I······  ········
0050  00 00 00                                           ···
```

Bit	+0..7	+8..15	+16..23	+24..31
0	Length			Type
32	Flags			
40	R	Stream Identifier		
...	Frame Payload			

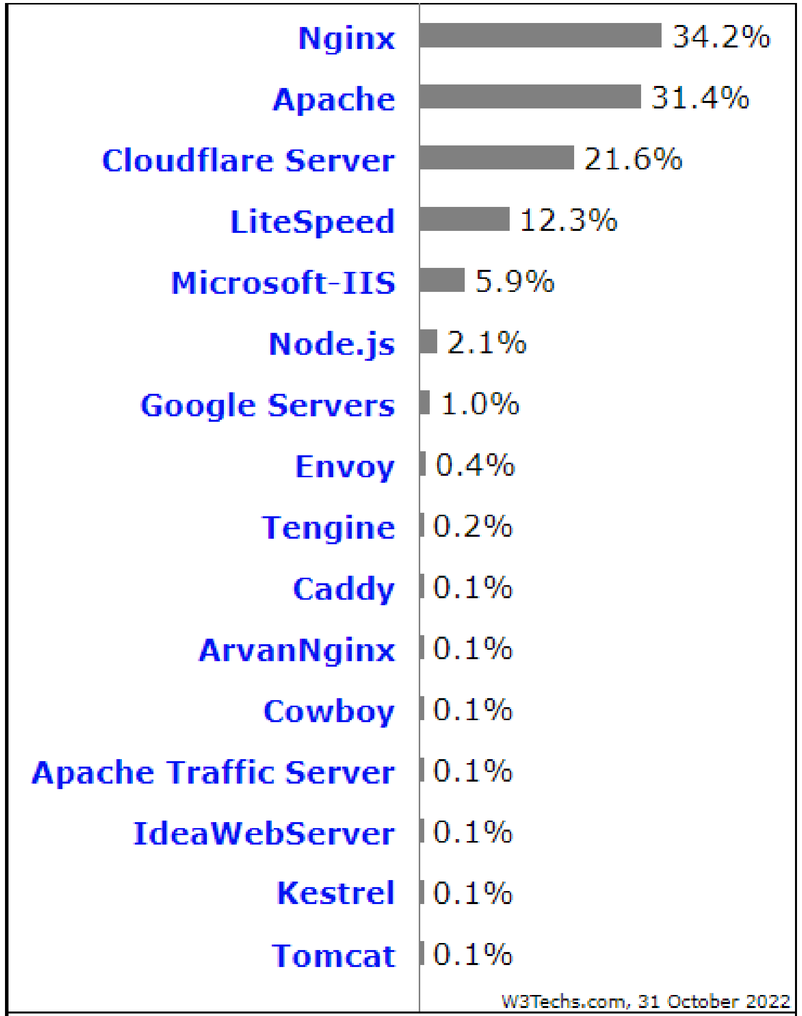
Looking at HTTP/1.1 Traffic



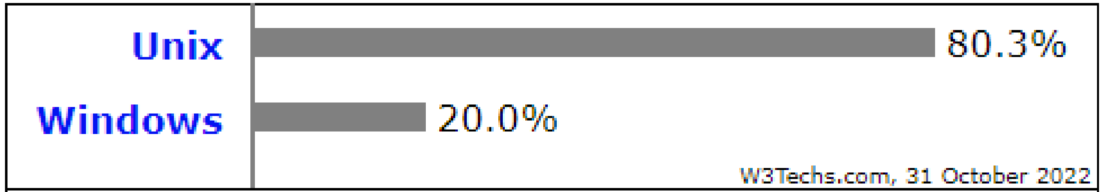
Compared to HTTP/2.0



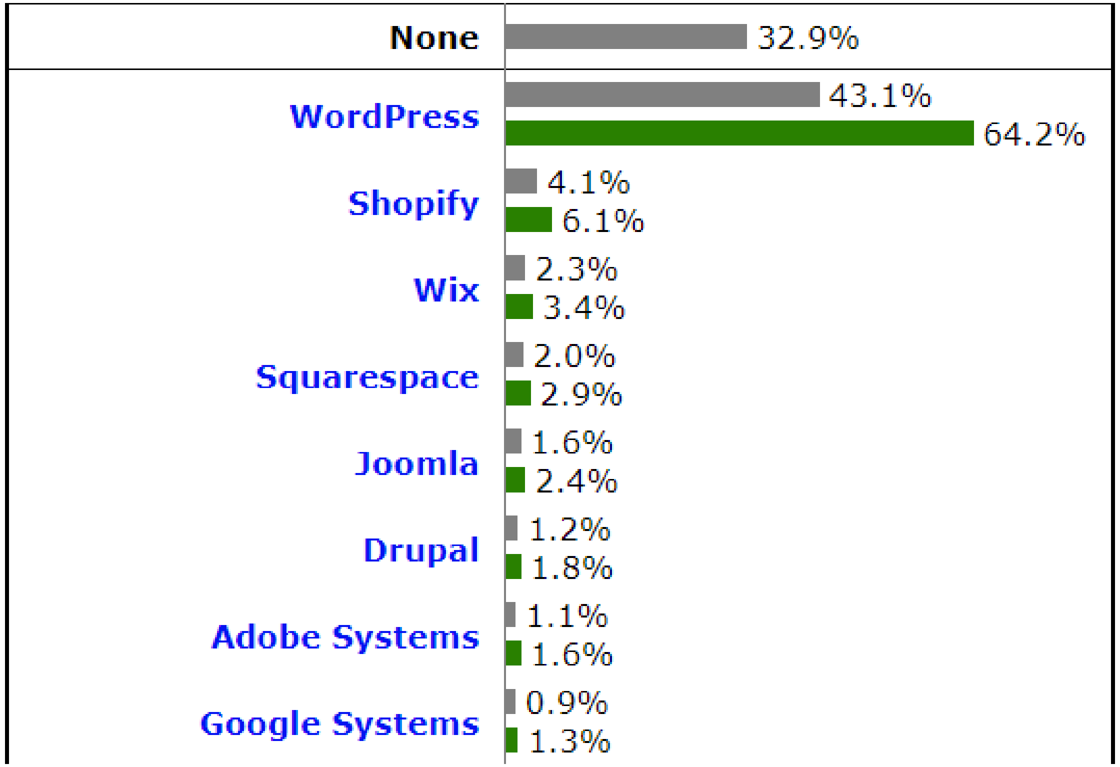
Web Statistics: web servers



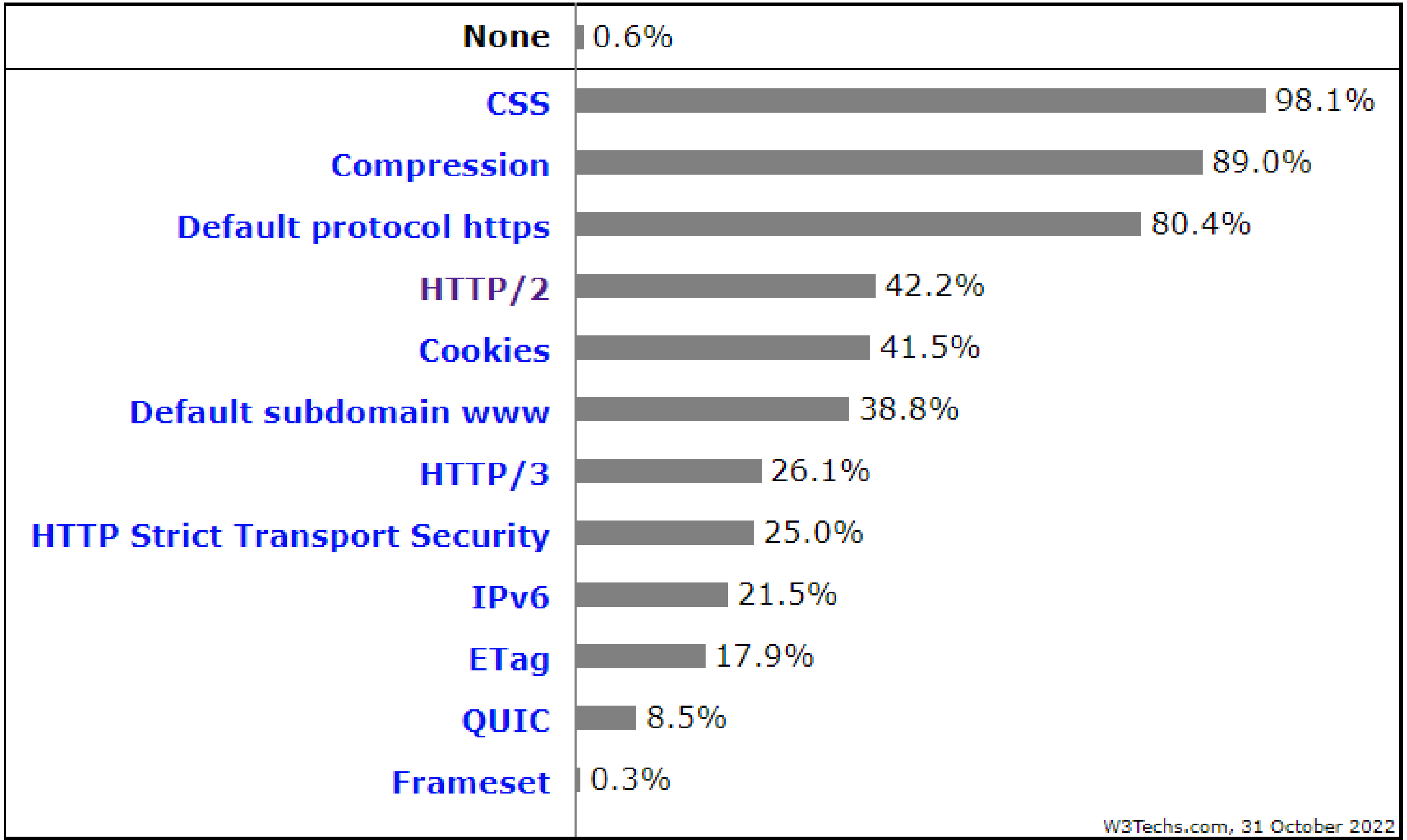
Percentages of websites using various web servers
Note: a website may use more than one web server



Percentages of websites using various operating systems
Note: a website may use more than one operating system

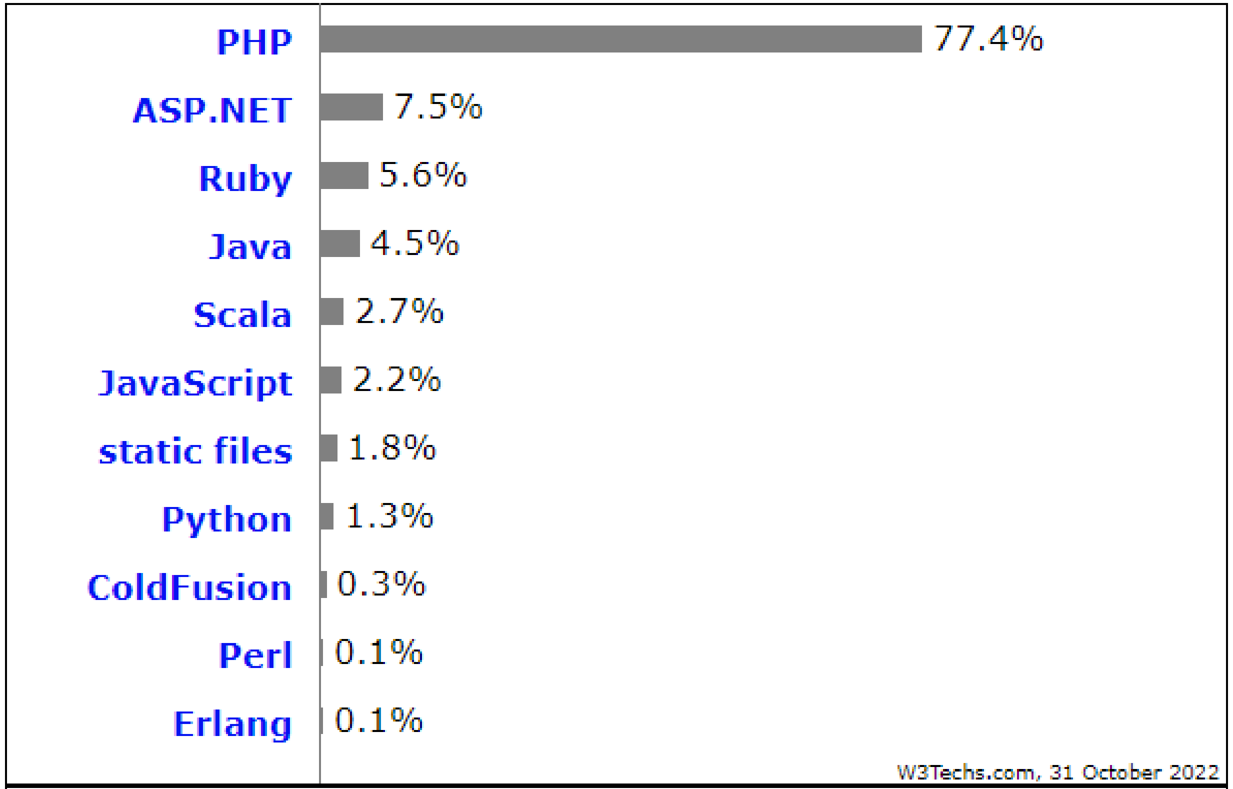


Web Statistics: elements

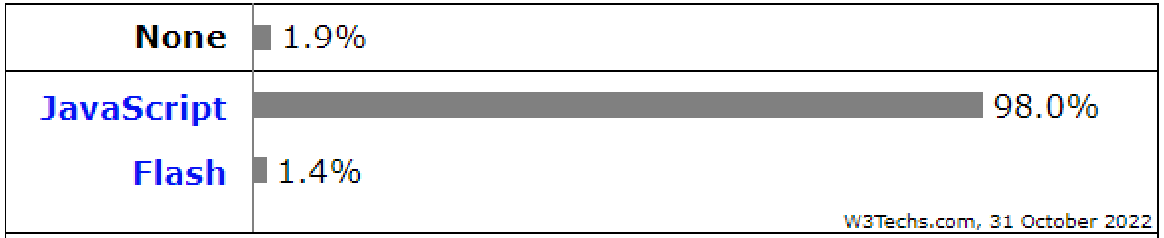


Percentages of websites using various site elements
Note: a website may use more than one site element

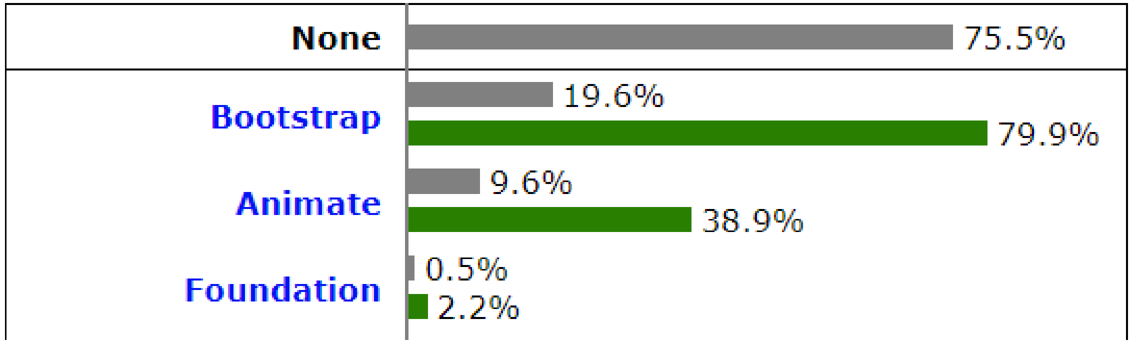
Web Statistics: powered by…



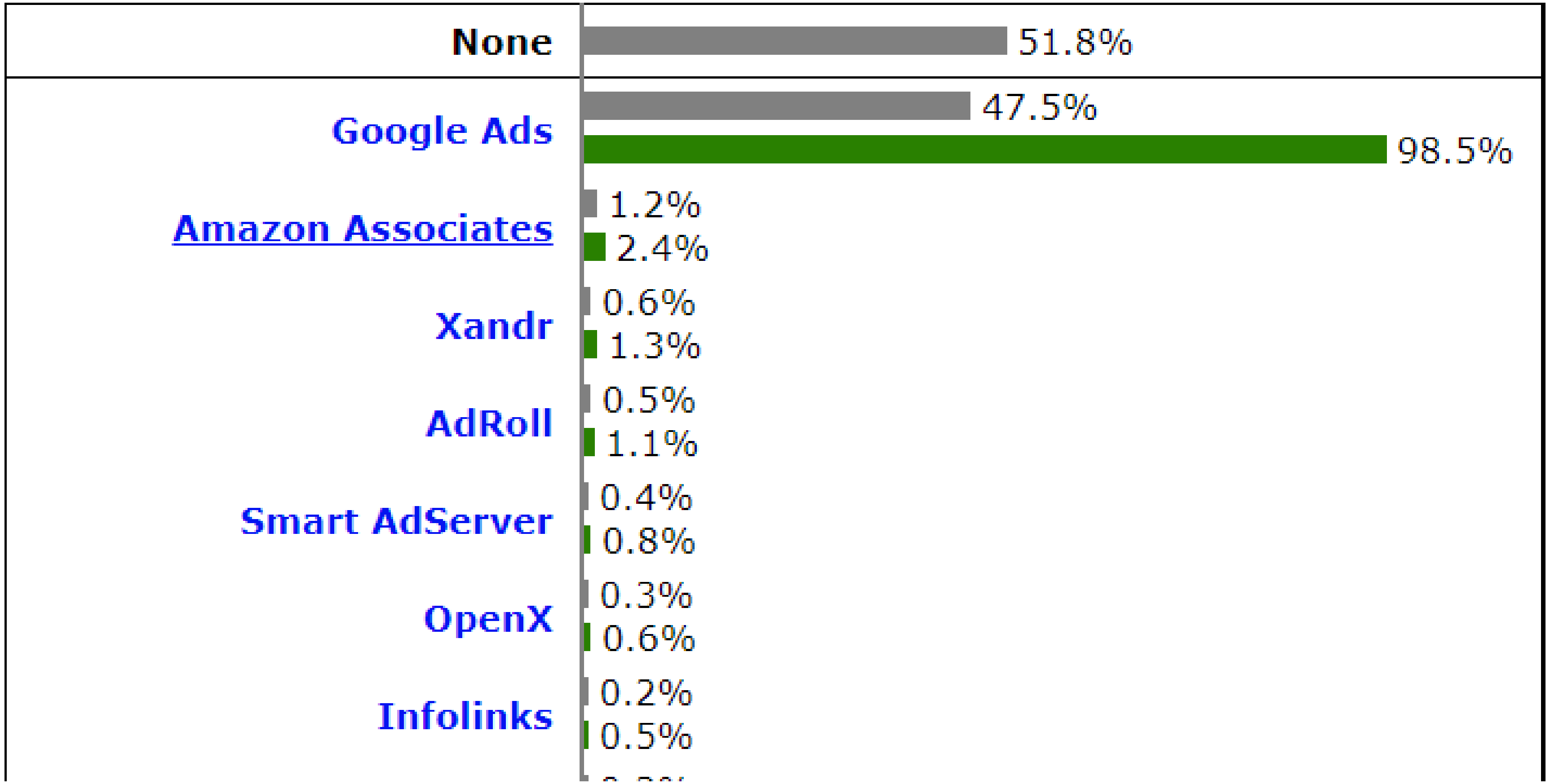
Percentages of websites using various server-side programming languages
Note: a website may use more than one server-side programming language



Percentages of websites using various client-side programming languages
Note: a website may use more than one client-side programming language



Web Statistics: advertising



HTTP/3 is ~~coming soon~~ here!

- https://datatracker.ietf.org/doc/draft-ietf-quic-http/
- Version 30 (Fall 2020)
- Version 34 (Spring 2021)
- RFC 9114 (Summer 22)

- HTTP/2 + QUIC + TLS
- Implemented in most major browsers at this point.
- Major changes:
    - Use UDP instead of TCP
    - Encrypted traffic required

JSON Review

- **J**ava**S**cript **O**bject **N**otation
- Extension: `.json`
- Language independent format derived from JavaScript.

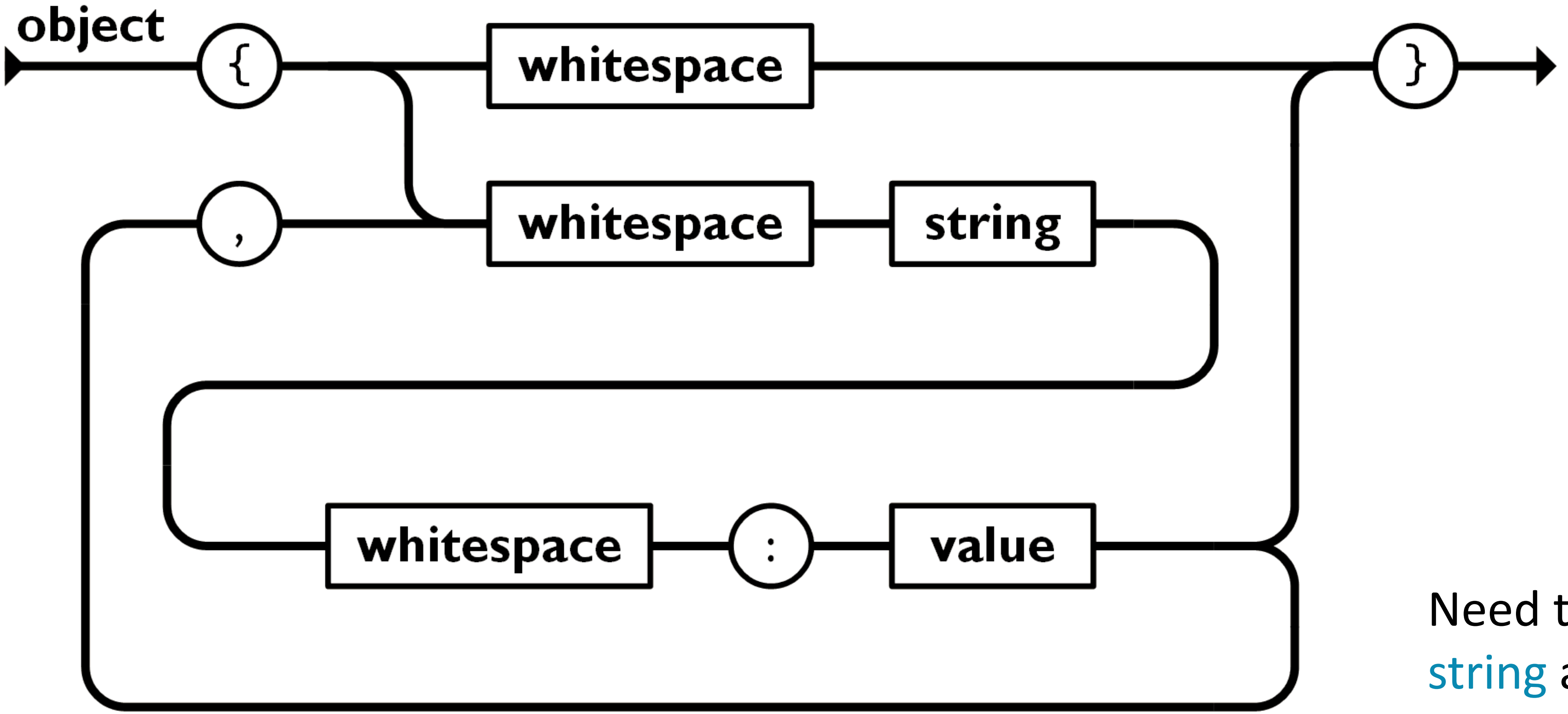
- Pronounced: jay-sawn
  - Creator says it is pronounced “Jason” but also, “doesn’t care”

- Early 2000’s – present

JSON Example: course schedule

```
{
        "course": "COMP 221",
        "semester": "Fall 2021",
        "title": "UNIX Systems Programming I",
        "schedule": {
                "headers": [
                        "Week", "Topics", "Reading", "Assignments"
                ],
                "weeks": [
                        { "type": "week", "data": {
                                "topic": "Getting Started",
                                "reading": "Ch 2, Ch 3",
                                "assignments": "L01"
                                }
                        }
                ]
        }
}
```

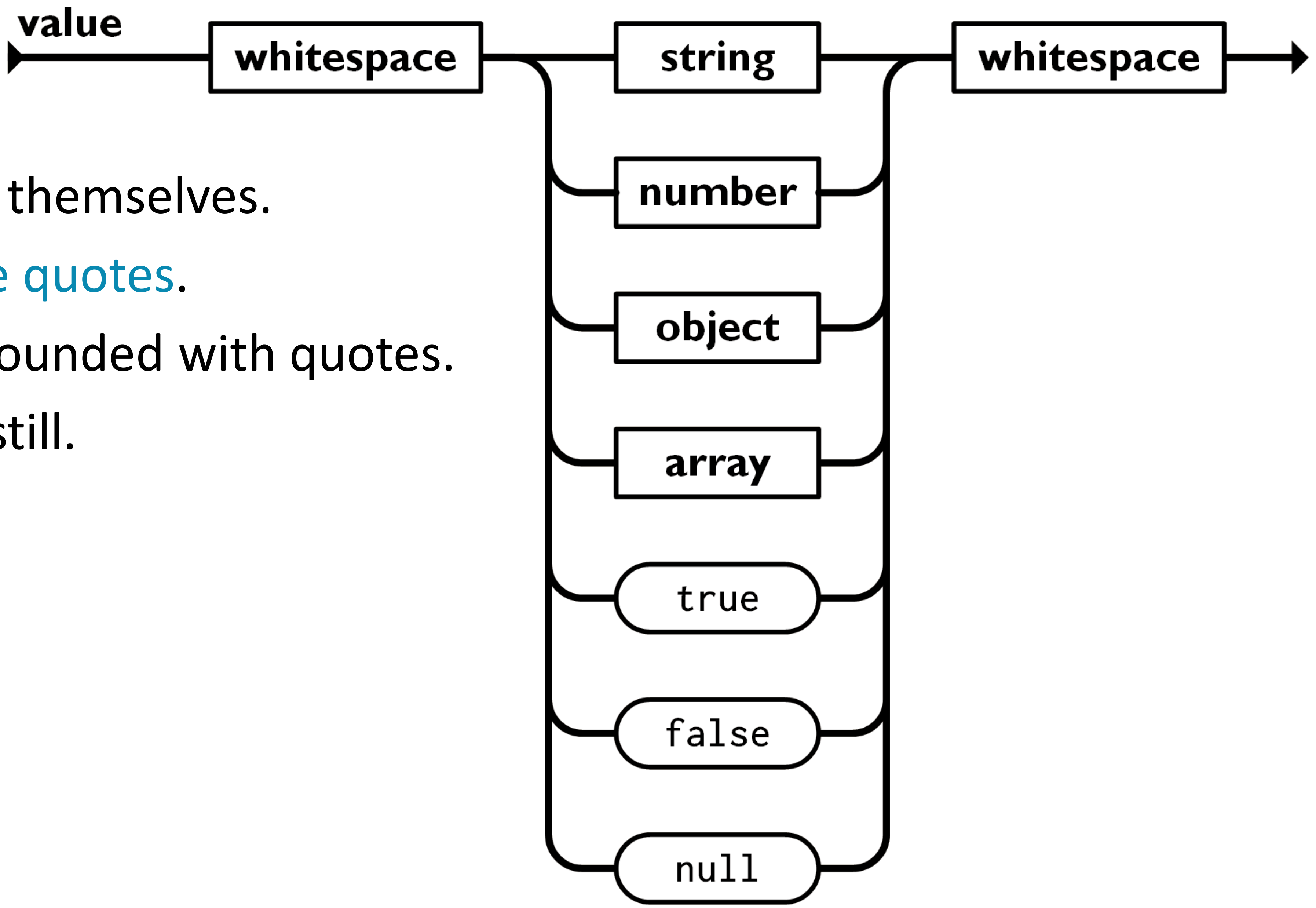
JSON: Object specification



Need to define
string and value still

{ "course": "COMP 221", "semester": "Fall 2021” }
{ }
{ "course": "COMP 221” }

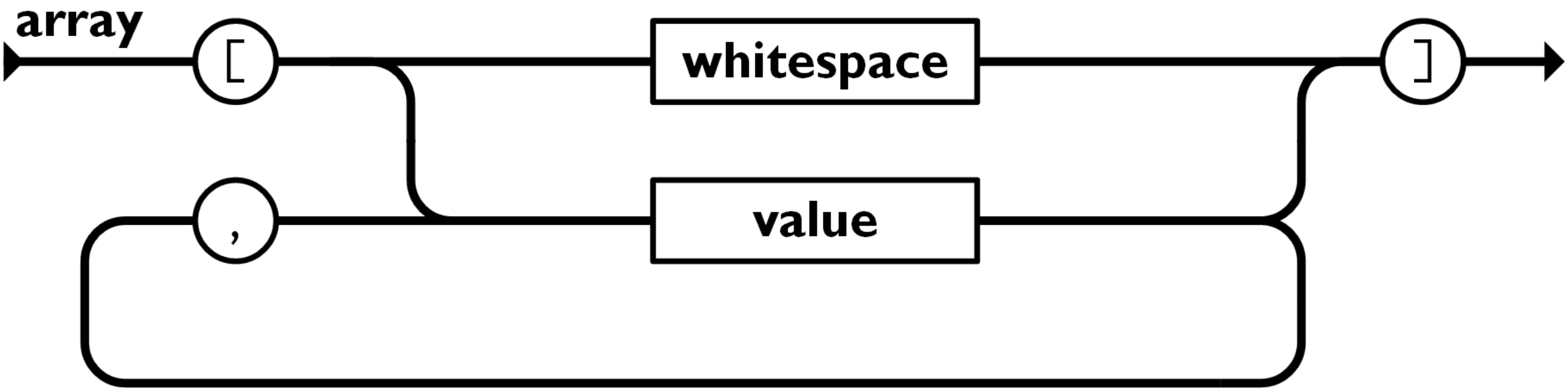
JSON: Value Specification



- Values can be objects themselves.
- Strings require double quotes.
- Numbers are not surrounded with quotes.
- Need to define array still.

JSON: Array specification

- Comma separated values surrounded with [ ].



HAR (Http ARchive) files are JSON

- Example from documentation
- https://w3c.github.io/web-performance/specs/HAR/Overview.html

request

This object contains detailed info about performed request.

```
"request": {
    "method": "GET",
    "url": "http://www.example.com/path/?param=value",
    "httpVersion": "HTTP/1.1",
    "cookies": [],
    "headers": [],
    "queryString" : [],
    "postData" : {},
    "headersSize" : 150,
    "bodySize" : 0,
    "comment" : ""
}
```