From HTTP 1.1 tp HTTP 2.0

Bandwidth and Latency

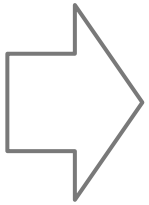
Bandwidth

In computing, bandwidth is the bit-rate of available or consumed information capacity expressed typically in metric multiples of bits per second. Variously, bandwidth may be characterized as network bandwidth, data bandwidth, or digital bandwidth.

Latency

Latency is a time interval between the stimulation and response, or, from a more general point of view, as a time delay between the cause and the effect of some physical change in the system being observed.

Delay	User reaction
0 - 100 ms	Instant
100 - 300 ms	Slight perceptible delay
300 - 1000 ms	Task focus, perceptible delay
1 s+	Mental context switch
10 s+	I'll come back later...



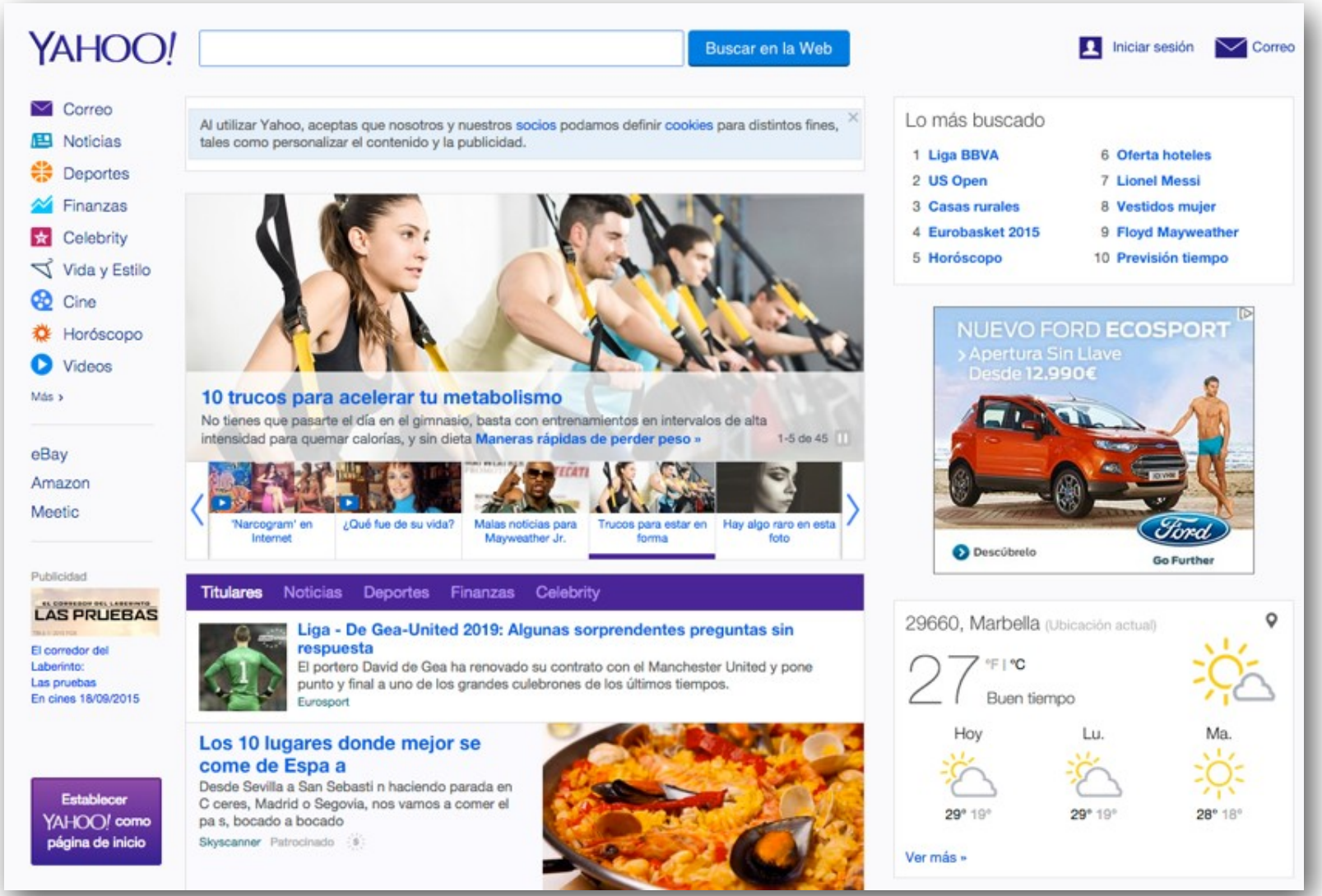
"1000 ms time to glass challenge"

- *Simple user-input must be acknowledged within ~100 milliseconds.*
- *To keep the user engaged, the task must complete within 1000 milliseconds.*

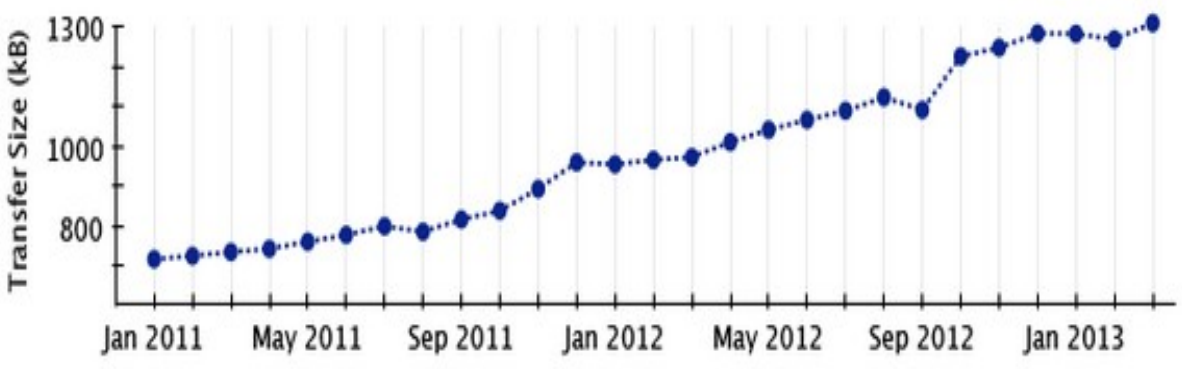
Ergo, our pages should render within 1000 milliseconds.

The World Wide Web in 1996

The World Wide Web Today



Our applications are complex, and growing…

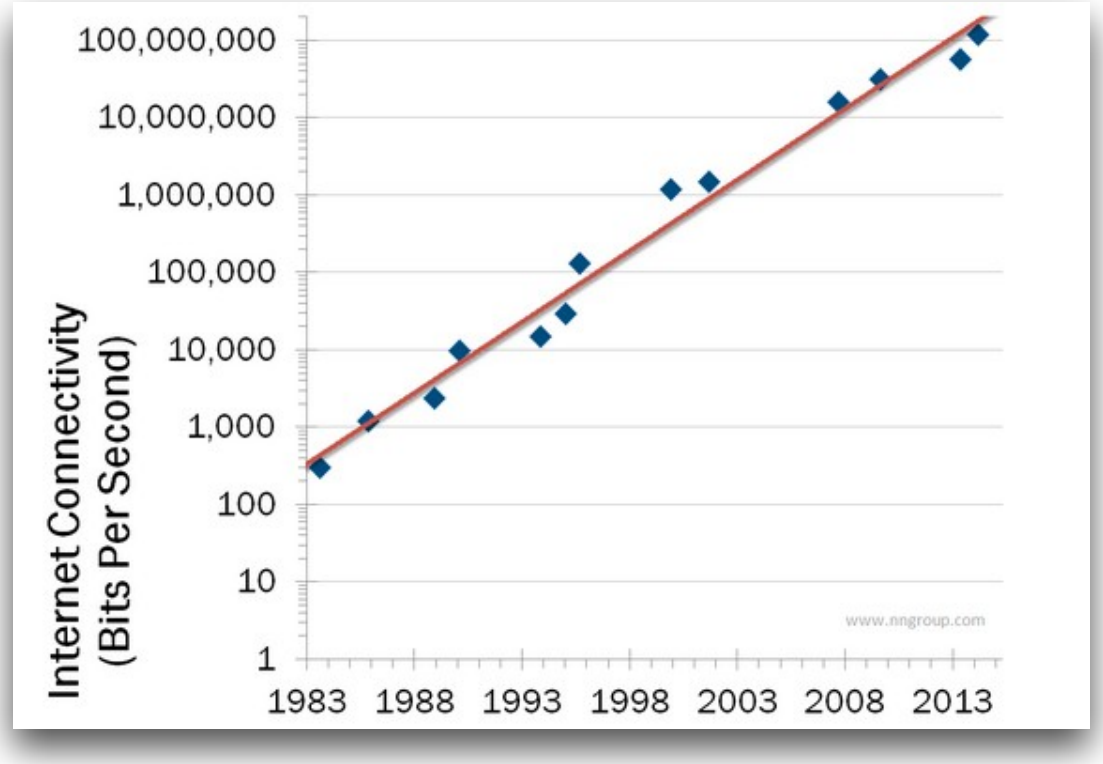


Content Type	Desktop		Mobile	
	Avg # of requests	Avg size	Avg # of requests	Avg size
HTML	10	56 KB	6	40 KB
Images	56	856 KB	38	498 KB
Javascript	15	221 KB	10	146 KB
CSS	5	36 KB	3	27 KB
Total	86+	1169+ KB	57+	711+ KB

Ouch!

HTTP Archive

Nielsen's Law of Bandwidth



1984 - 300 bps

50% Growth per Year

Source http://www.nngroup.com/articles/law-of-bandwidth/

Bandwidth by Country

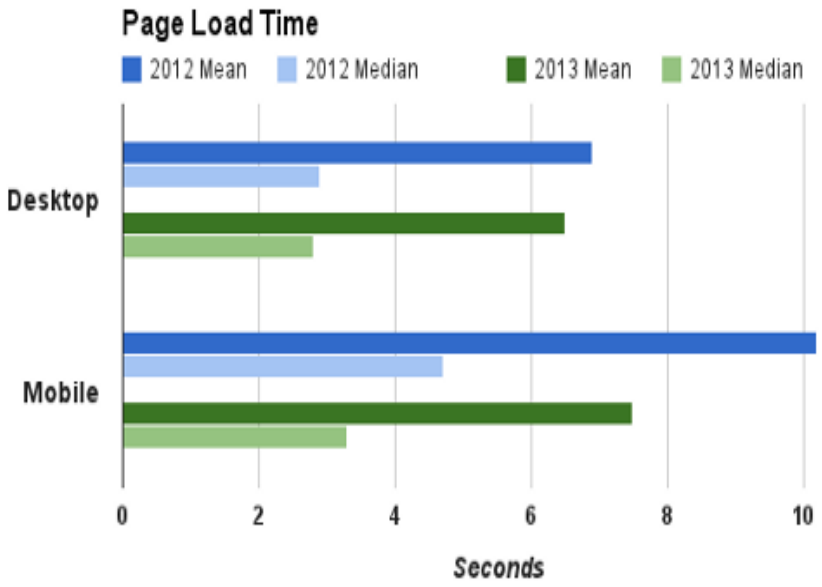
Position	Country	Speed (Mbps)
1	South Korea	25.3
2	Hong Kong	16.3
3	Japan	15
4	Switzerland	14.5
…	…	…
12	United States	11.5
13	Belgium	11.4
…	…	…
24	Germany	8.7
…	…	…
28	Spain	7.8
…	…	…
30	Australia	6.9
31	France	6.9
…	…	…
55	Bolivia	1.1

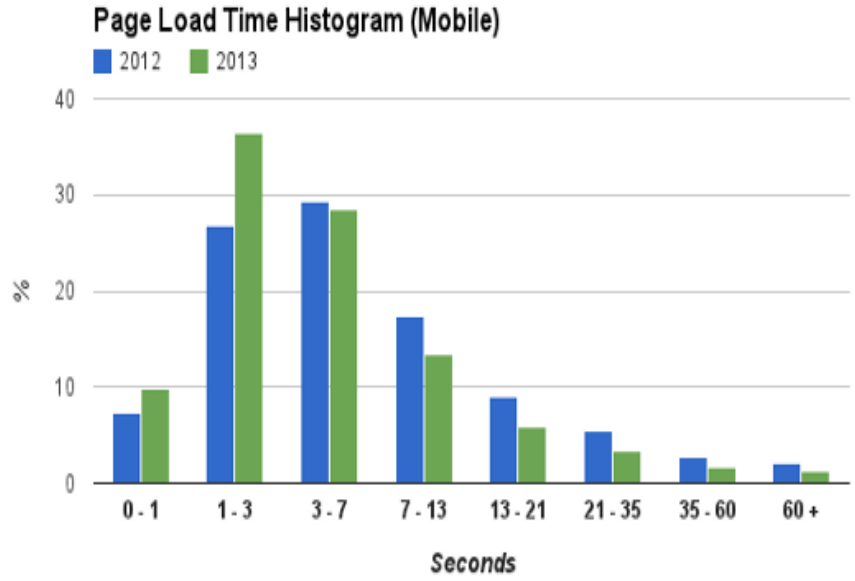
Source Akamai 2014 Rankings - Wikipedia

Mobile Networks

Region	Average Speed (Mbps)
Europe	20.4
North America	9.6
Asia Pacific	8.8
South America	7.0
Africa	4.8

Source Akamai State of the Internet Q1 2015



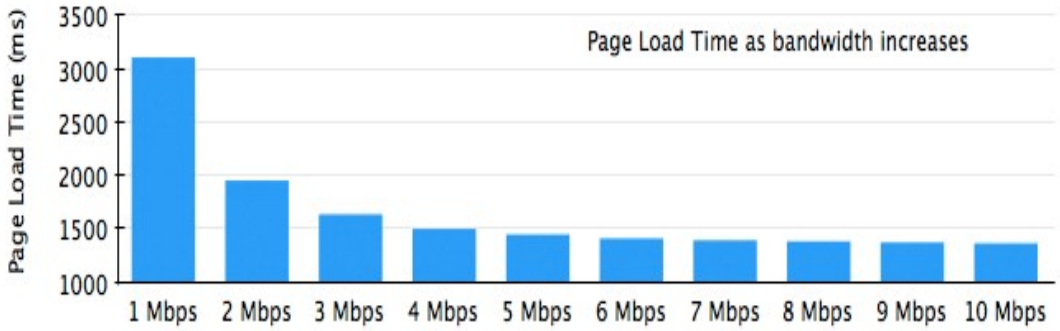


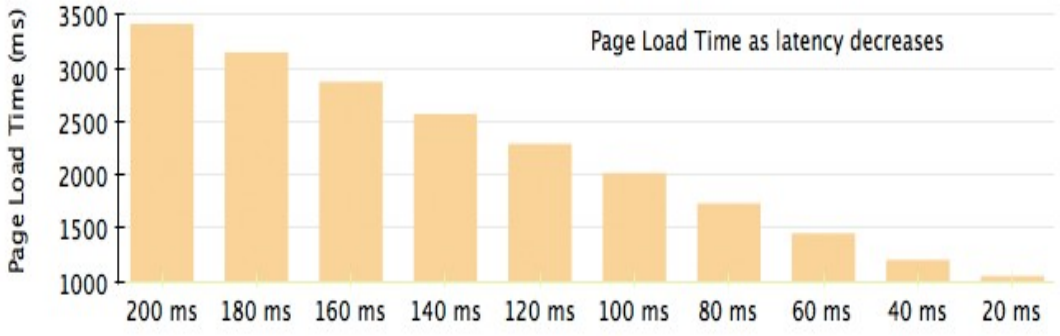
Desktop: ~3.1 s

Mobile ~3.5 s

[Is the web getting faster?](#) - Google Analytics Blog

Latency *vs.* Bandwidth impact on Page Load Time





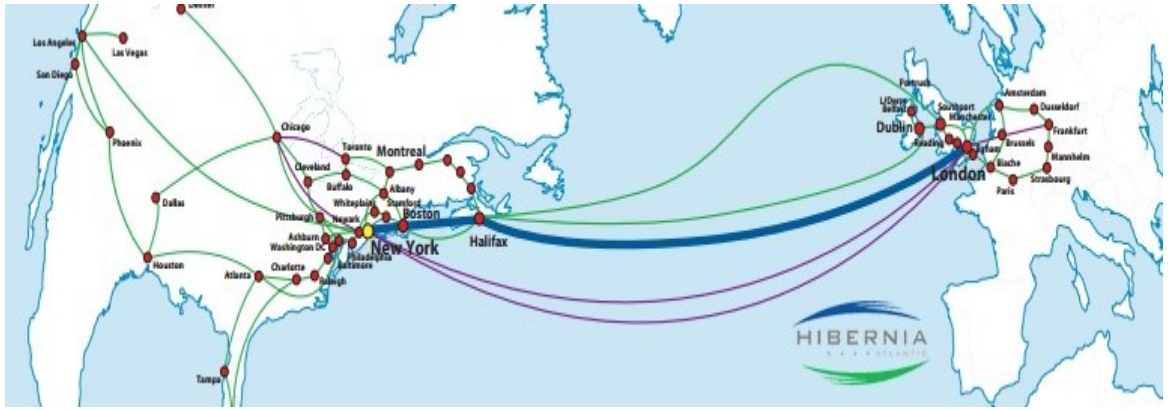
Single digit perf improvement after 5 Mbps

Average household in is running on a **5 Mbps+** connection. Ergo, **average consumer would not see an improvement in page loading time by upgrading their connection.**

(doh!)

[Bandwidth doesn't matter (much)](#) - Google

- Improving bandwidth is"easy"...
  - 60% of new capacity through upgrades in past decade + unlit fiber
  - *"Just lay more fiber..."*
- **Improving latency is expensive... impossible?**
  - Bounded by the speed of light - oops!
  - We're already within a small constant factor of the maximum
  - *"Shorter cables?"*



$80M / ms

[Latency is the new Performance Bottleneck]

And latency is per connection

Typical Web Page

Name	Method	Status	Type	Initiator	Size	Time
comunes.css	GET	200	stylesheet	(index):37	(from cache)	42 ms
pxlctl2.gif?r=2&m=1&s=25c10673e...	GET	302	gif	http://pxlctl.elpais.com/pxlc...	508 B	58 ms
advert.gif?648618153	GET	200	gif	(index):164	392 B	59 ms
caja_tlife.css	GET	200	stylesheet	(index):1619	793 B	70 ms
pxlctl2.gif?m=1&r=2&w=2123515	GET	302	gif	http://elpais.com/pxlctl.gif?...	347 B	80 ms
?script=0&random=4019638651&ip...	GET	200	gif	http://www.google.com/ads...	343 B	94 ms
gpt.js	GET	304	script	pbs.slots.js:1	363 B	95 ms
rta.js?netId=4045&cookieName=crt...	GET	200	script	pbs.slots.js:1	457 B	118 ms
pxlctl.gif?m=1&r=2&w=2123515	GET	302	text/html	(index):164	241 B	121 ms
elpais.com	GET	200	document	http://www.elpais.es/	61.9 KB	123 ms
?script=0&random=4019638651	GET	302	text/html	http://googleads.g.doublecli...	380 B	138 ms
rep.gif?ver=1&typ=pgv&rnd=iemvn...	GET	200	gif	(index):165	494 B	147 ms
bid?src=3226&u=http%3A%2F%2Felp...	GET	200	script	amzn_ads.js:1	217 B	191 ms
s64909375414717?AQB=1&pccr=tr...	GET	302	text/plain	http://prisacom.112.2o7.net...	811 B	213 ms
?value=0&guid=ON&script=0	GET	302	gif	(index):164	651 B	226 ms
?idsite=255&url=http%3A%2F%2Felp...	GET	200	xhr	vrs.20150907.js:6	337 B	355 ms
?transport=jsonp&idsite=255&url=h...	GET	200	xhr	vrs.20150907.js:6	1.1 KB	369 ms
s64909375414717?AQB=1&ndh=1&...	GET	302	text/plain	(index):165	1.8 KB	432 ms
ads?gdfp_req=1&correlator=231131...	GET	200	script	pubads_impl_72.js:189	29.4 KB	528 ms

380B

Optimization Possibilities with HTTP 1.1

Optimizations…

Reduce DNS lookups

Every hostname resolution requires a network roundtrip, imposing latency on the request and blocking the request while the lookup is in progress.

Make fewer HTTP requests

No request is faster than a request not made: eliminate unnecessary resources on your pages.

Use a Content Delivery Network

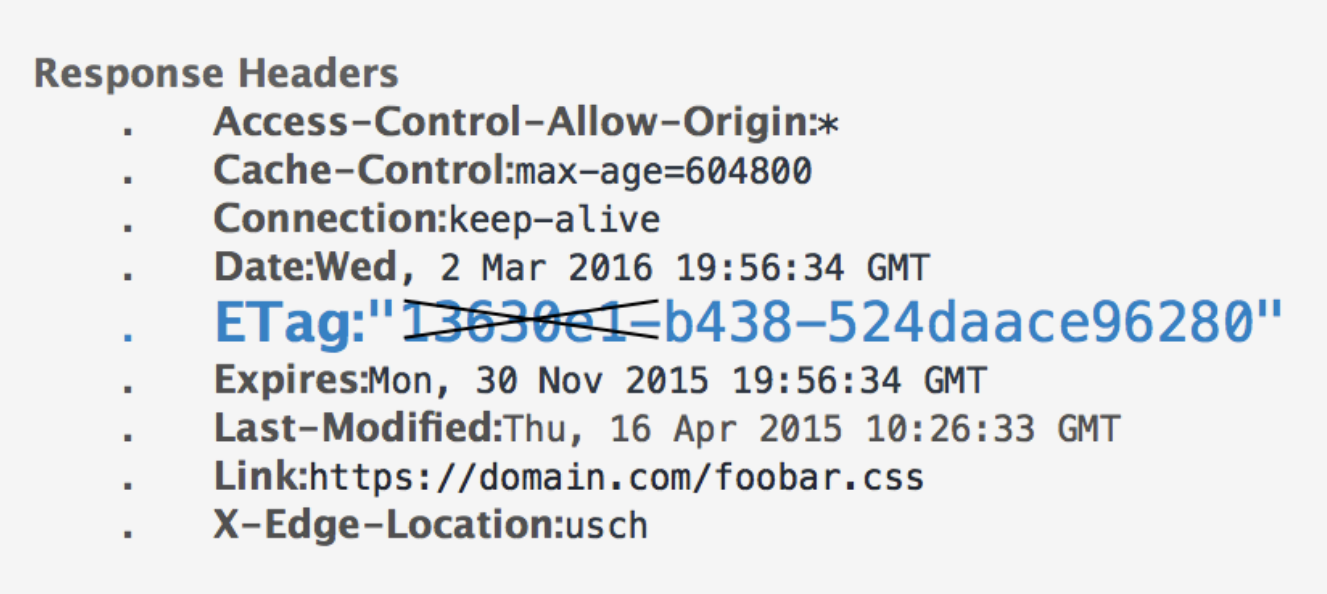
Locating the data geographically closer to the client can significantly reduce the network latency of every TCP connection and improve throughput.

Optimizations…

Add an Expires header and configure Etags

An Expires header can be used to specify the cache lifetime of the object, allowing it to be retrieved directly from the user's cache and eliminating the HTTP request entirely.

ETags and Last-Modified headers provide an efficient cache revalidation mechanism—effectively a fingerprint or a timestamp of the last update.



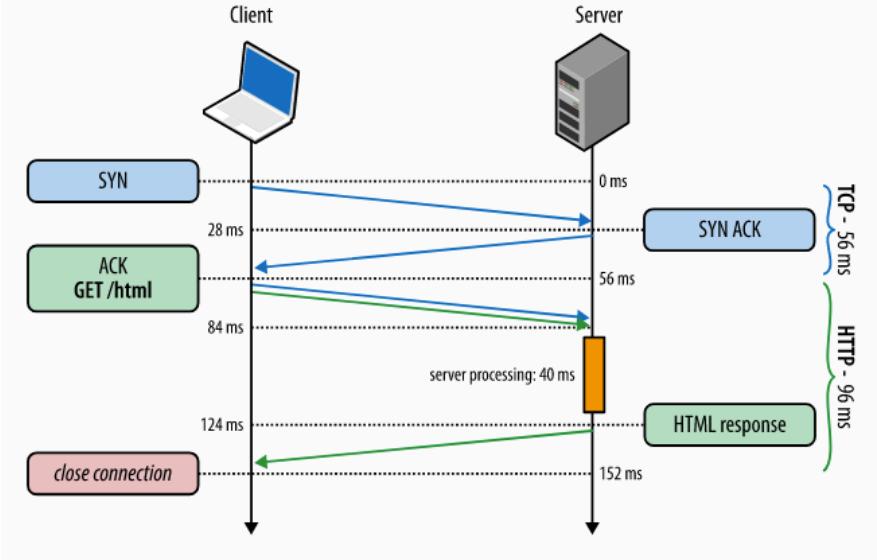
Optimizations…

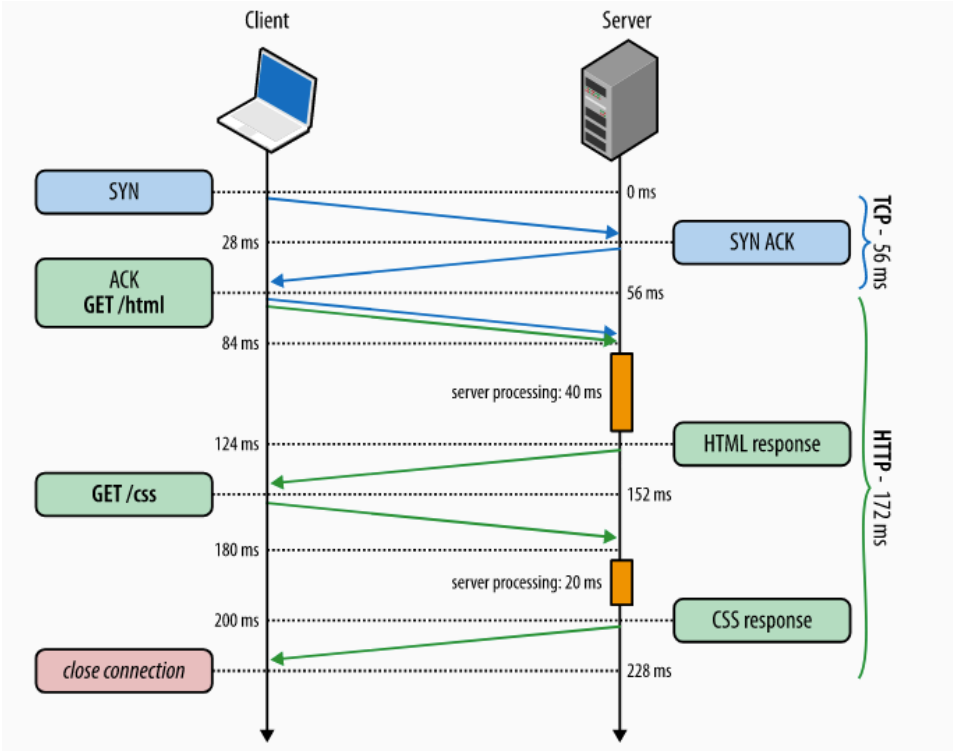
Gzip assets

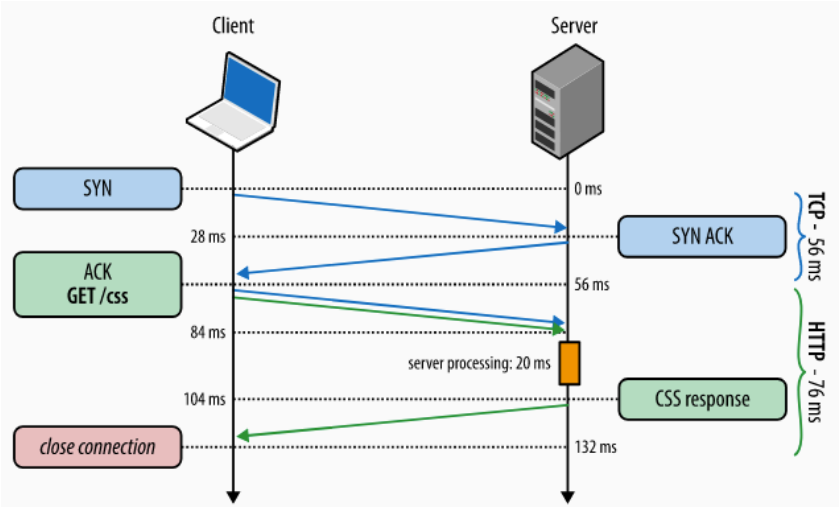
All text-based assets should be compressed with Gzip when transferred between the client and the server.
On average, Gzip will reduce the file size by 60–80%, which makes it one of the simpler (configuration flag on the server) and high-benefit optimizations you can do.

Avoid HTTP redirects

HTTP redirects can be extremely costly, especially when they redirect the client to a different hostname, which results in additional DNS lookup, TCP connection latency, etc.







Using Keep-Alive

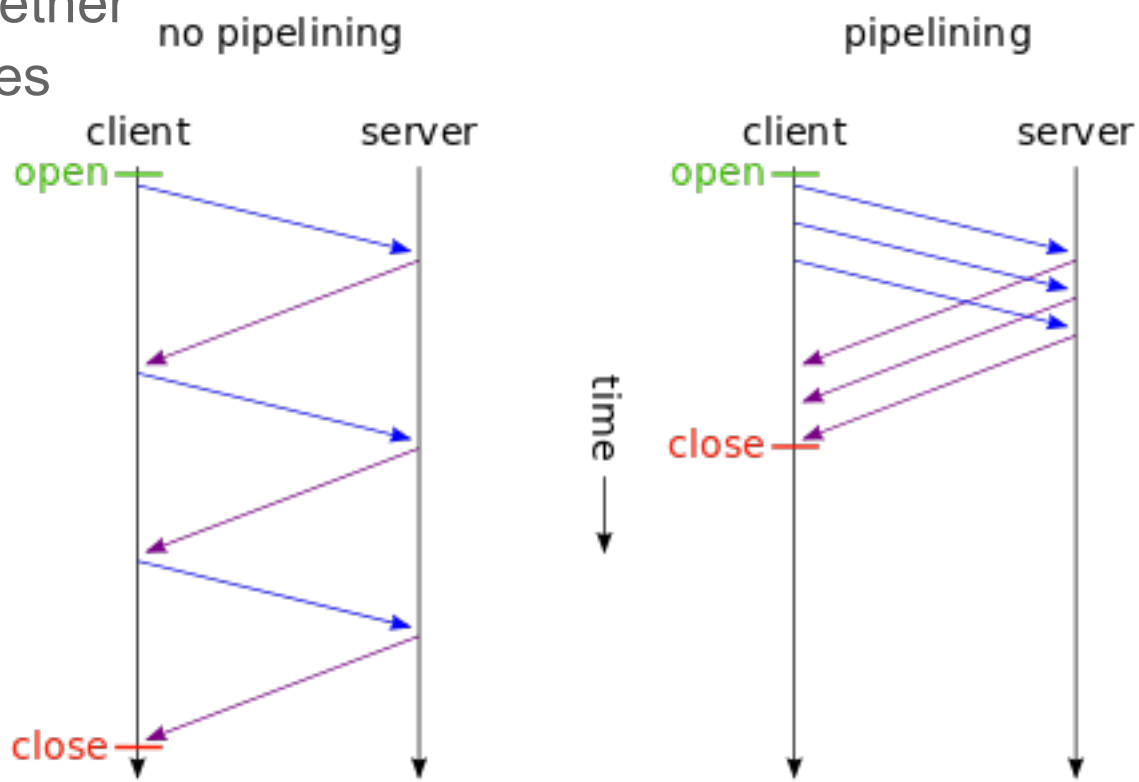
Keep Alive-Pipelining

Using a single connection to send multiple successive requests

Pipelining Requests

Send several requests together

Head of line blocking issues



Keep-alive + pipelining

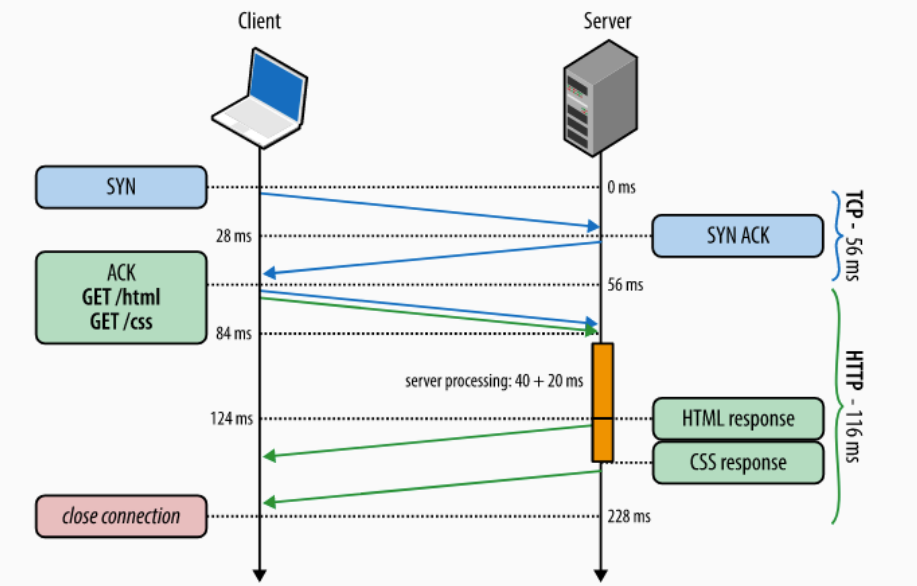


Figure 11-3. Pipelined HTTP requests with server-side FIFO queue

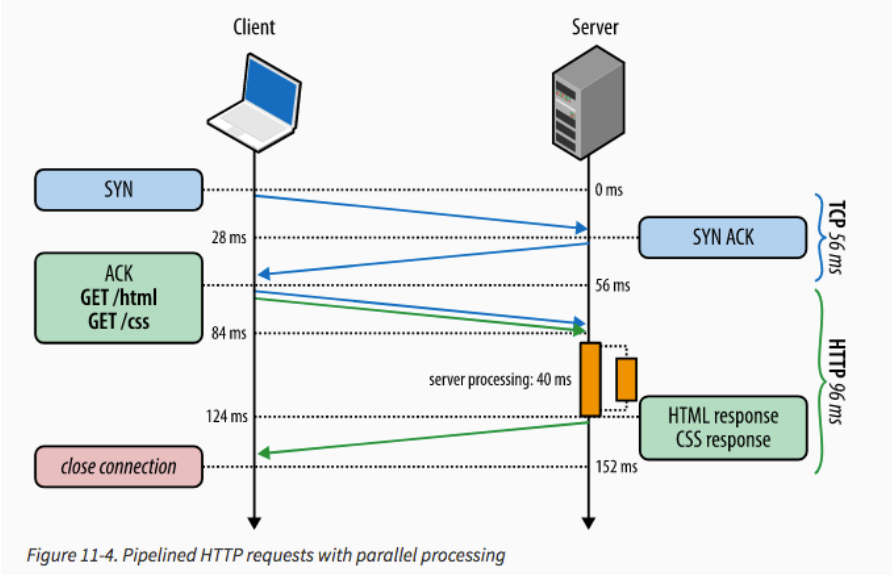


Figure 11-4. Pipelined HTTP requests with parallel processing

Keep Alive-Pipelining

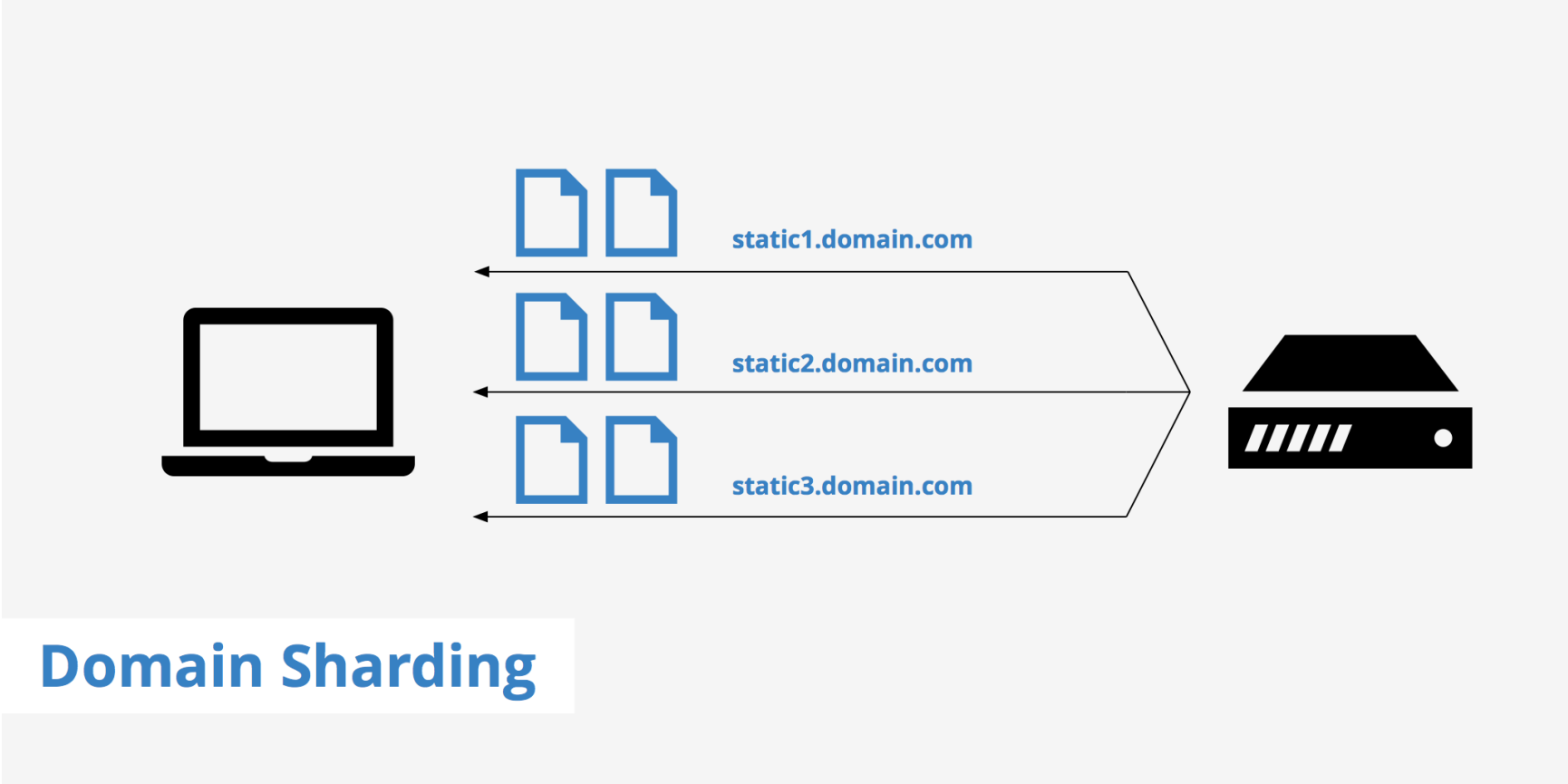
The pipelining of requests results in **a dramatic improvement] in the loading times of HTML pages**, especially over high latency connections such as satellite Internet connections.

The speedup is less apparent on broadband connections, as the limitation of HTTP 1.1 still applies**: the server must send its responses in the same order that the requests were received — so the entire connection remains first-in-first-out and Head Of Line blocking can occur.**

Domain sharding

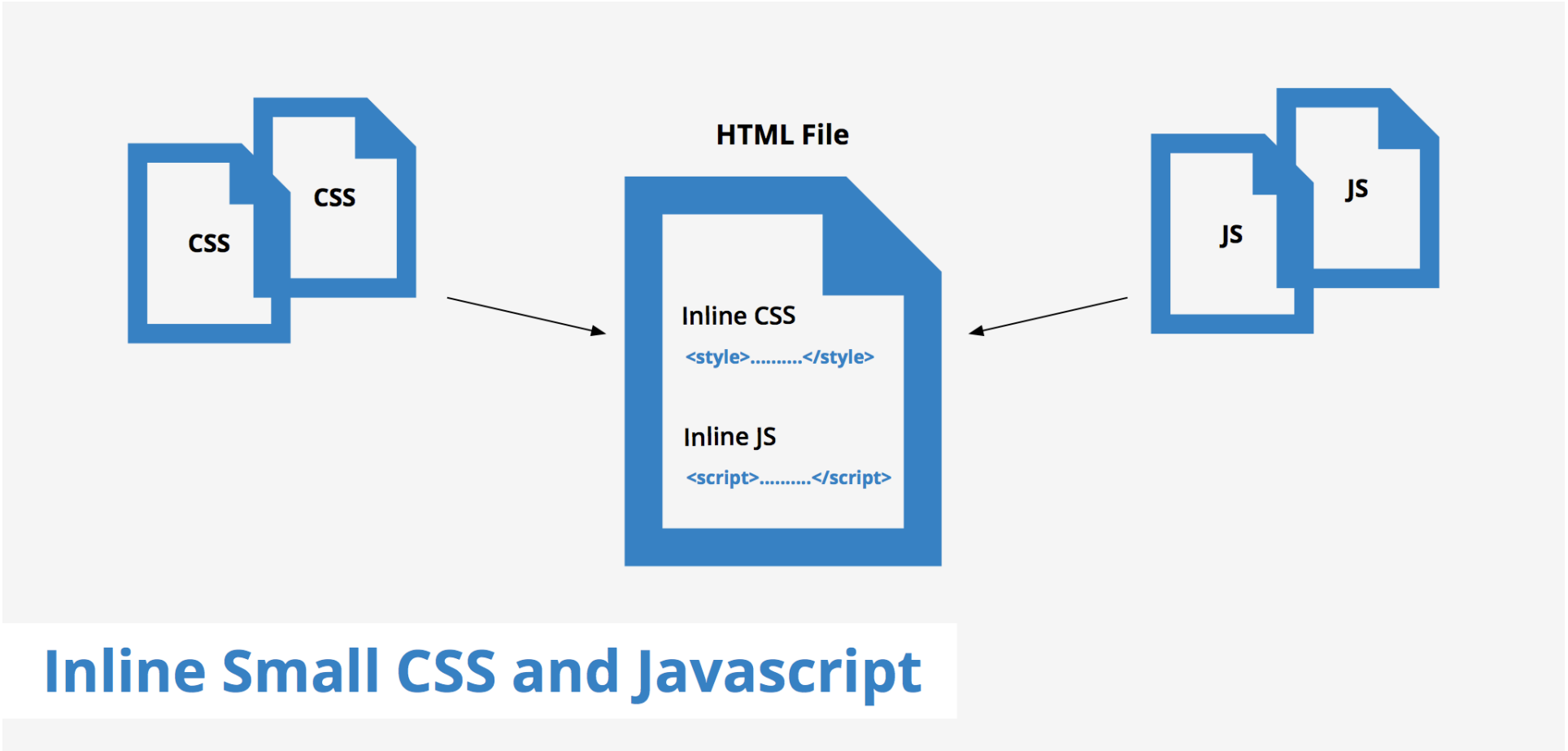
- Web browsers traditionally place limits on the amount of concurrent downloads allowed for each domain (between 2-16). This limit was put in place by the Internet Engineering Task Force and is mentioned in the HTTP/1.1 specification. It was recommended in order to reduce Internet congestion and web server overloading.

Domain sharding



Inlining

- Inline Small CSS and Javascript



Spriting

- Multiple images are combined into a larger, composite image.

Concatenating files (JavaScript, CSS)

- Reduces number of downloads and latency overhead
- Less modular code and expensive cache invalidations (e.g. app.js)
- Slower execution (must wait for entire file to arrive)

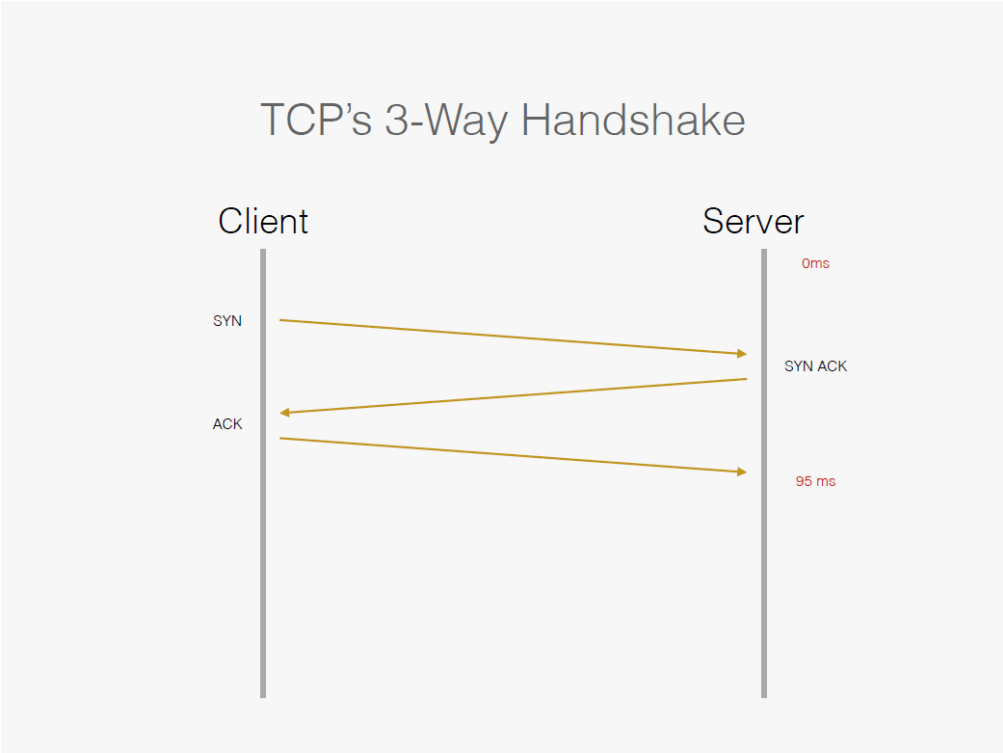
- **Spriting images**
  - Reduces number of downloads and latency overhead
  - Painful and annoying preprocessing and expensive cache invalidations
  - Have to decode entire sprite bitmap - CPU time and memory

- **Domain sharding**
  - TCP Slow Start? Browser limits, Nah... 15+ parallel requests -- *Yeehaw!!!*
  - Causes congestion and unnecessary latency and retransmissions

- **Resource inlining**
  - Eliminates the request for small resources
  - Resource can't be cached, inflates parent document
  - 30% overhead on base64 encoding

The culprit is HTTP on TCP

Http 1.1 chatty - TCP is not made for chatty protocols
TCP has slow start and head of line blocking



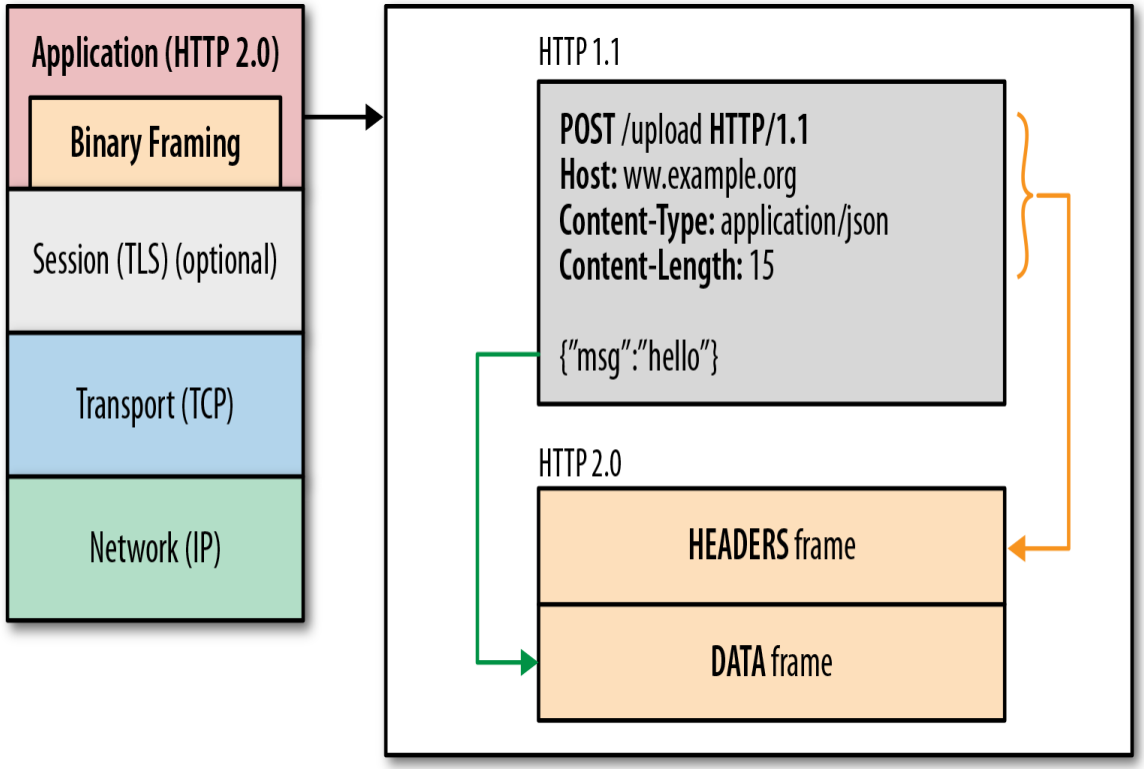
Most HTTP transfers are short and bursty, whereas TCP is optimized for long-lived, bulk data transfers

HTTP 2.0 in a nutshell

- One TCP connection
- Request = Stream
  - Streams are multiplexed
  - Streams are prioritized

- (New) binary framing layer
  - Prioritization
  - Flow control
  - Server push

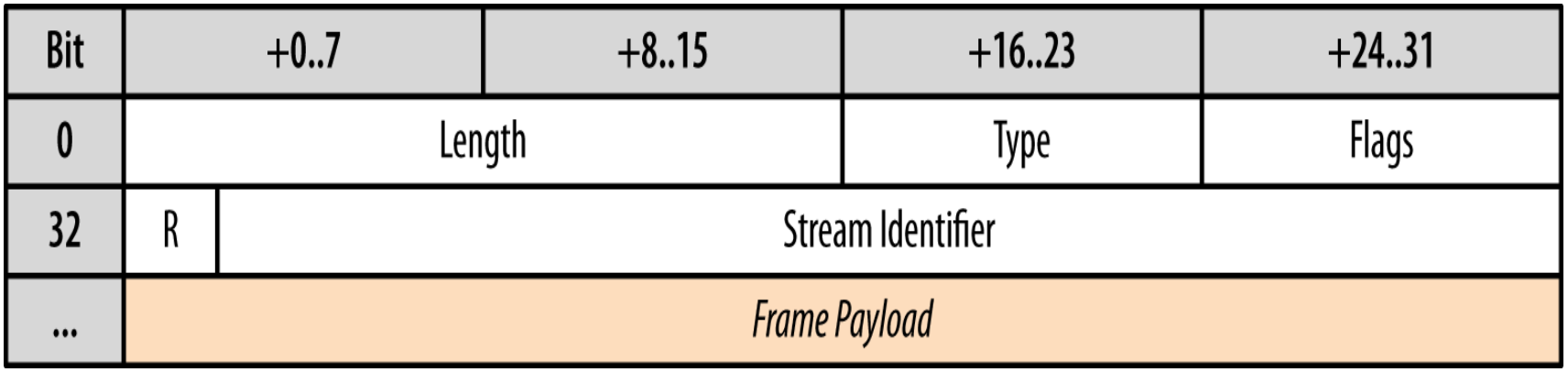
- Header compression



*"... **we're not replacing all of HTTP** — the methods, status codes, and most of the headers you use today will be the same. Instead, **we're re-defining how it gets used "on the wire" so it's more efficient**, and so that it is more gentle to the Internet itself ...."*

- Mark Nottingham
(chair)

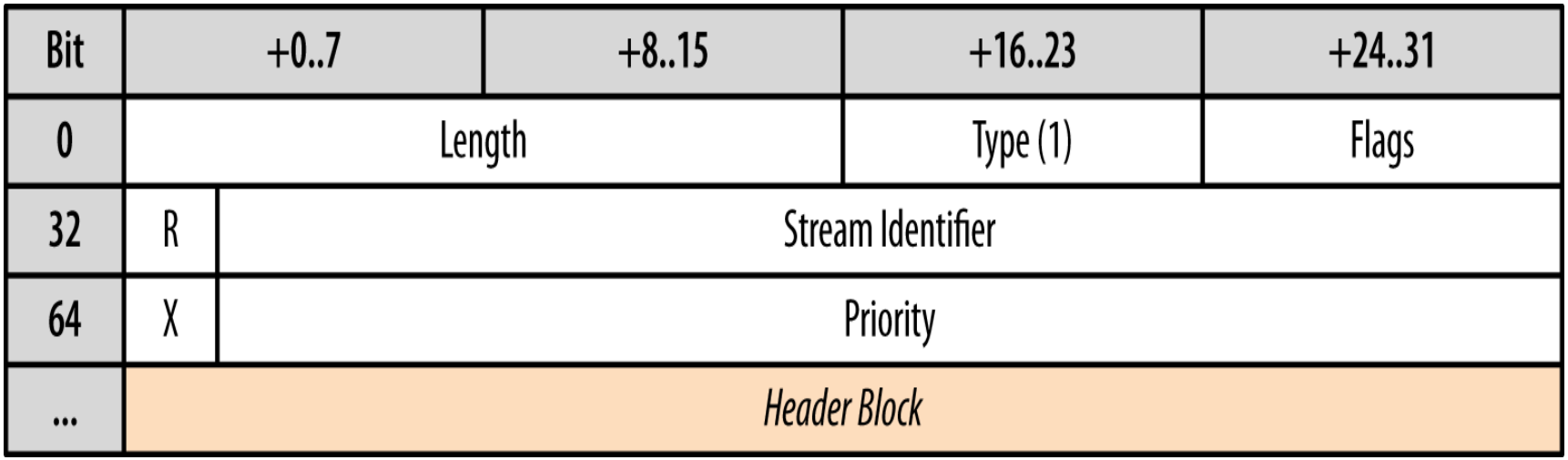
All frames have a common 8-byte header



- Length-prefixed frames
- **Type** indicates …type of frame
  - *DATA, HEADERS, PRIORITY, PUSH_PROMISE, …*
- Each frame may have custom **flags**
  - e.g. *END_STREAM*
- Each frame carries a **31-bit stream identifier**
  - After that, it's frame specific payload...

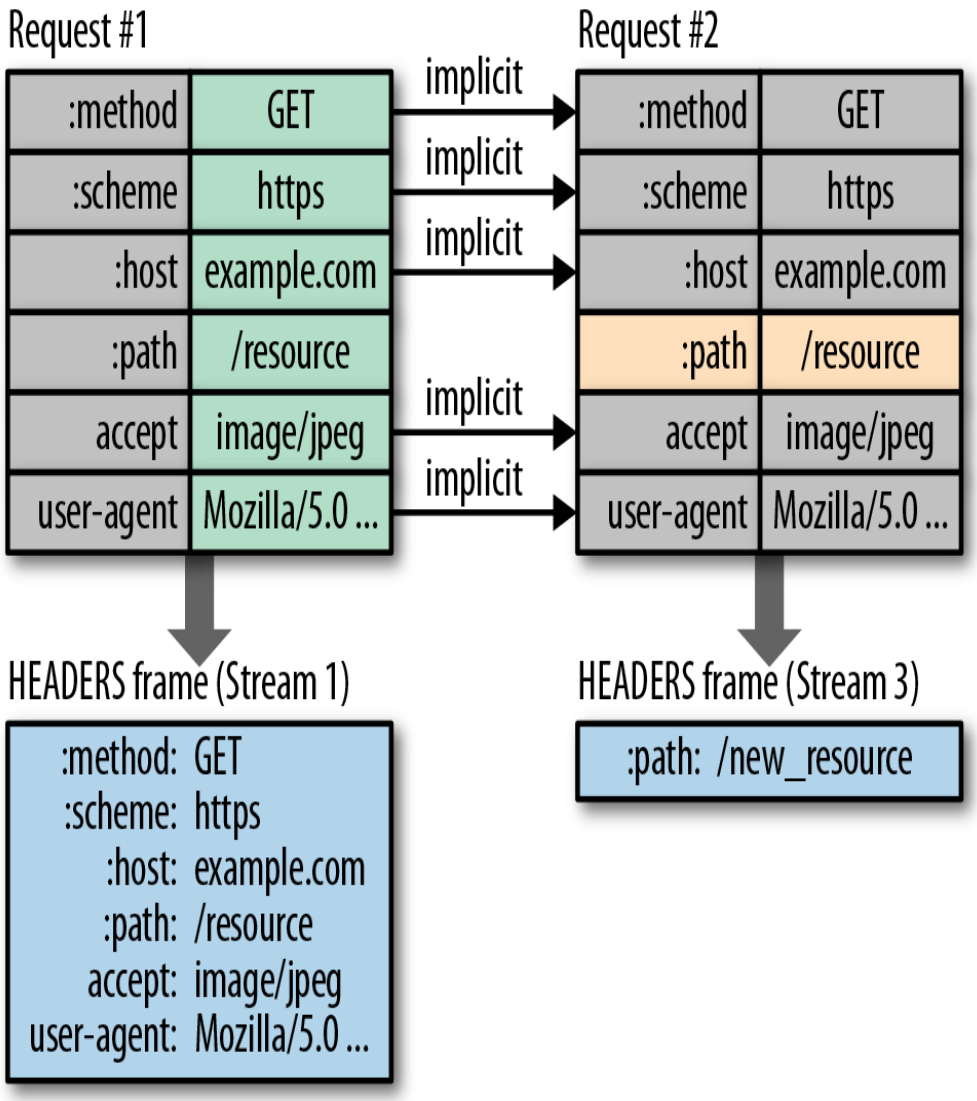
```
frame =
buf.read(8)
if
frame_i_care
_about
  do_someth
ing_smart
else
  buf.skip(fram
```

Opening a new stream with HTTP 2.0 (HEADERS)



- Common 8-byte header
- Client / server allocate new stream ID
  - *client: odd, server: even*
- Optional 31-bit stream priority field
  - *Flags indicates if priority is present*
  - *2^31 is lowest priority*
- HTTP header payload
  - *see h eader-compression-01*

HTTP 2.0 header compression (in a nutshell)

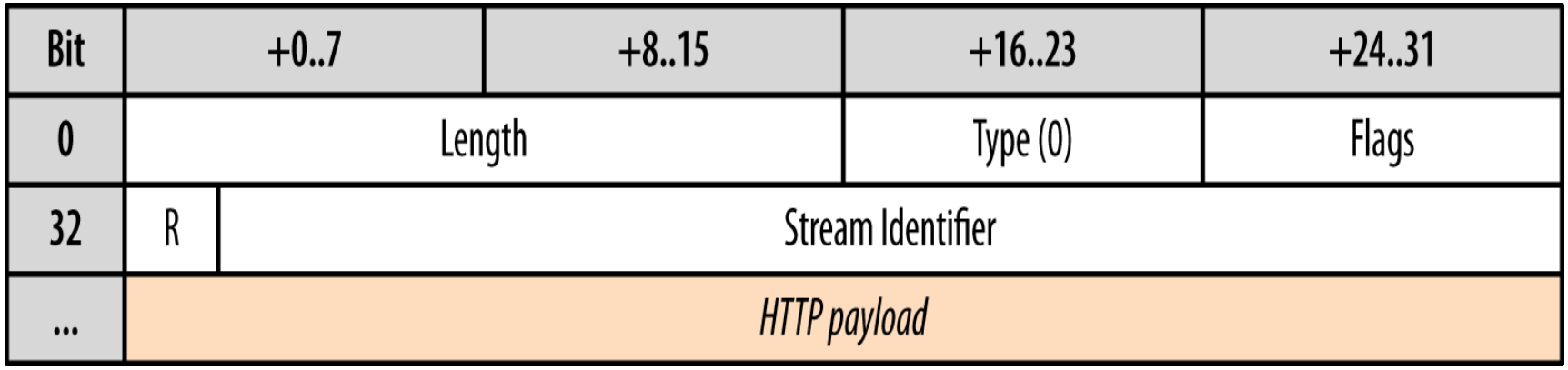


- Each side maintains "header tables"
- Header tables are initialized with common header key-value pairs

- New requests "toggle" or "insert" new values into the table
- New header set is a "diff" of the previous set of headers

- *E.g. Repeat request (polling) with exact same headers incurs no overhead (sans frame header)*

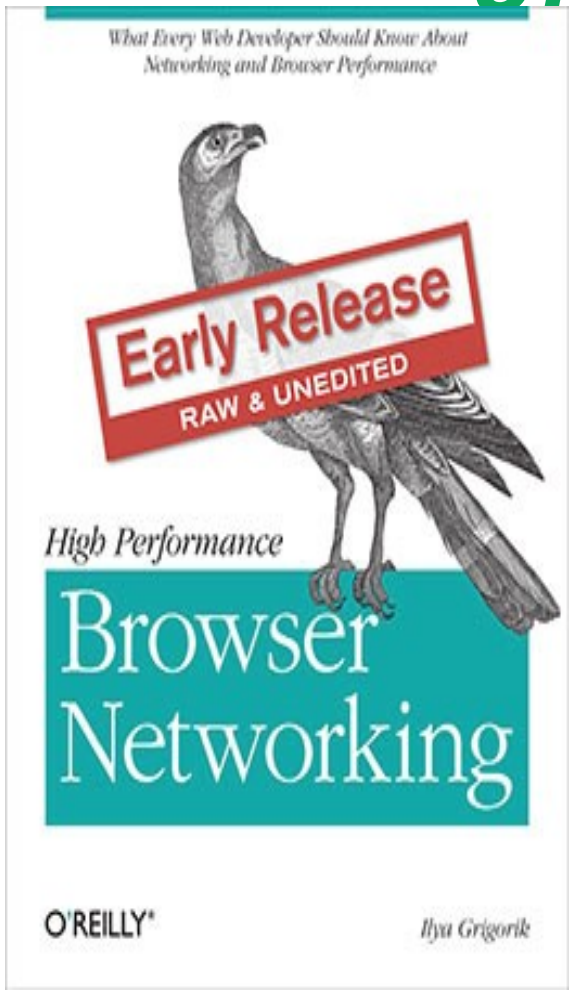
Sending application data with …DATA frames.



- Common 8-byte header
- Followed by application data…

- In theory, max-length =2^16-1
- To reduce head-of-line blocking: **max frame size is 2^14-1** (~16KB)
  - Larger payloads are split into multiple DATA frames, last frame carries “END_STREAM” flag

For an in-depth discussion on all of the above...



http://hpbn.co

- Optimizing **TCP** server stacks
- Optimizing **TLS** deployments
- Optimizing for **mobile** networks
- HTTP 2.0 features, framing, deployment...
- XHR, SSE, WebSocket, WebRTC, ...