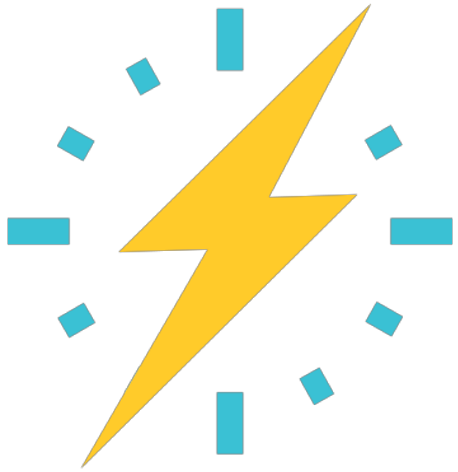


Making HTTP realtime with HTTP 2.0

aka, dropping the hacks, reclaiming performance!

Ilya Grigorik

igrigorik@google.com

Google

```
$> telnet igvita.com 80
Connected to 173.230.151.99

GET /archive

Hypertext delivery with HTTP 0.9! - eom.
(connection closed)
```

HTTP 0.9 is the ultimate MVP *- one line, plain-text "protocol" to test drive the "www idea".*

```
$> telnet ietf.org 80
Connected to 74.125.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
Last-Modified: Wed, 1 May 1996 12:45:26 GMT
Server: Apache 0.84

4 years of rapid iteration later... eom.
(connection closed)
```

HTTP 1.0 is an informational RFC *- documents "common usage" of HTTP found in the wild.*

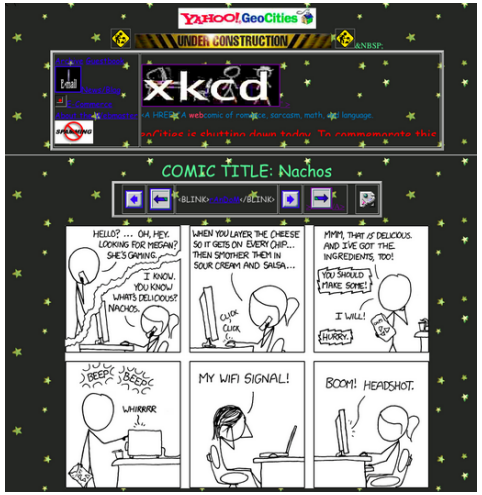
```
$> telnet google.com 80
Connected to 74.125.xxx.xxx

GET /index.html HTTP/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
Accept-Language: en-US,en;q=0.8
Cookie: __qca=P0-800083390... (snip)

HTTP/1.1 200 OK
Connection: keep-alive
Transfer-Encoding: chunked
Server: nginx/1.0.11
Content-Type: text/html; charset=utf-8
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

100
<!doctype html>
(snip)
```

HTTP 1.1 ships as RFC standard in 1999 *- hyper*
{~~text~~}media all the things!

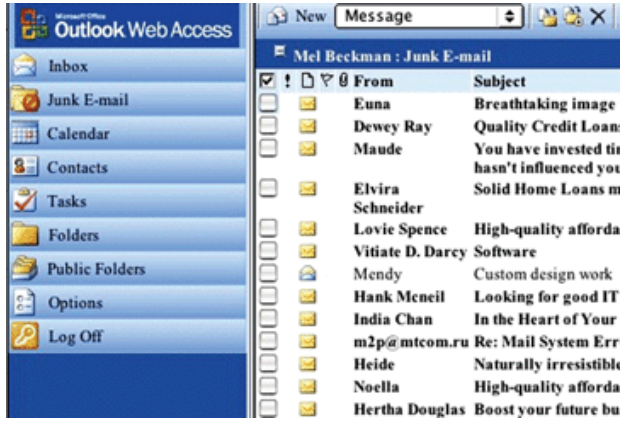
15 years later!

HTTPbis is getting dangerously close to closing all the outstanding 1.1 bugs. :)

In the meantime a few things have happened since...



Geocities ftw!
(circa 1997-2000)



Then AJAX happened.
(circa 2002-2005)

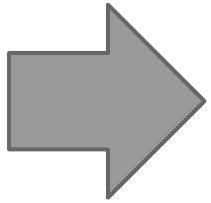
WebGL
WebRTC
Web Audio
WebSockets
...
Mobile web

(today)

State of the HTTP nation…

- **11** distinct hosts per page
- **76** distinct requests per page
- **987** transferred KBytes per page

- **Concurrency: 1**
- **Parallelism: 6**

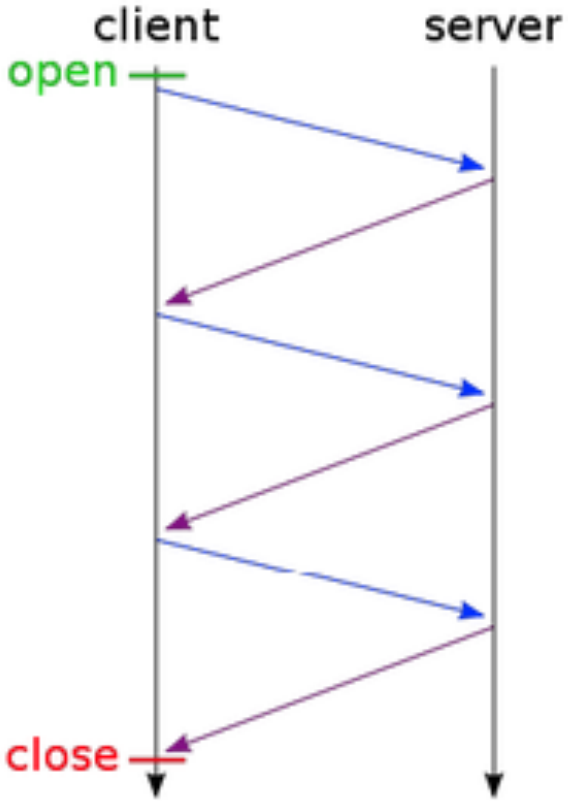


One transfer per connection.
At most six parallel connections.

*Resulting in typical load times of **3.8-10.5 seconds**.*

* All numbers are medians, based on HTTP Archive crawl data.

For all its awesome, HTTP 1.X has a lot of perf problems...



Top Desktop name	score	PerfTiming	Connections per Hostname
Chrome 20 →	12/16	yes	6
Firefox 14 →	13/16	yes	6
IE 8 →	7/16	no	6
IE 9 →	12/16	yes	6
Opera 12 →	10/16	no	6
RockMelt 0.9 →	13/16	yes	6
Safari 5.1 →	12/16	no	6

- Limited parallelism (~6)
- Client-side request queuing
- Browser queueing heuristics
- High protocol overhead
  - *~800 bytes + cookies*

- Competing TCP flows
- Spurious retransmissions
- Extra memory / FD resources
- Handshake overhead
- Slow-start overhead
- ...

Where there's a will, there's a way...

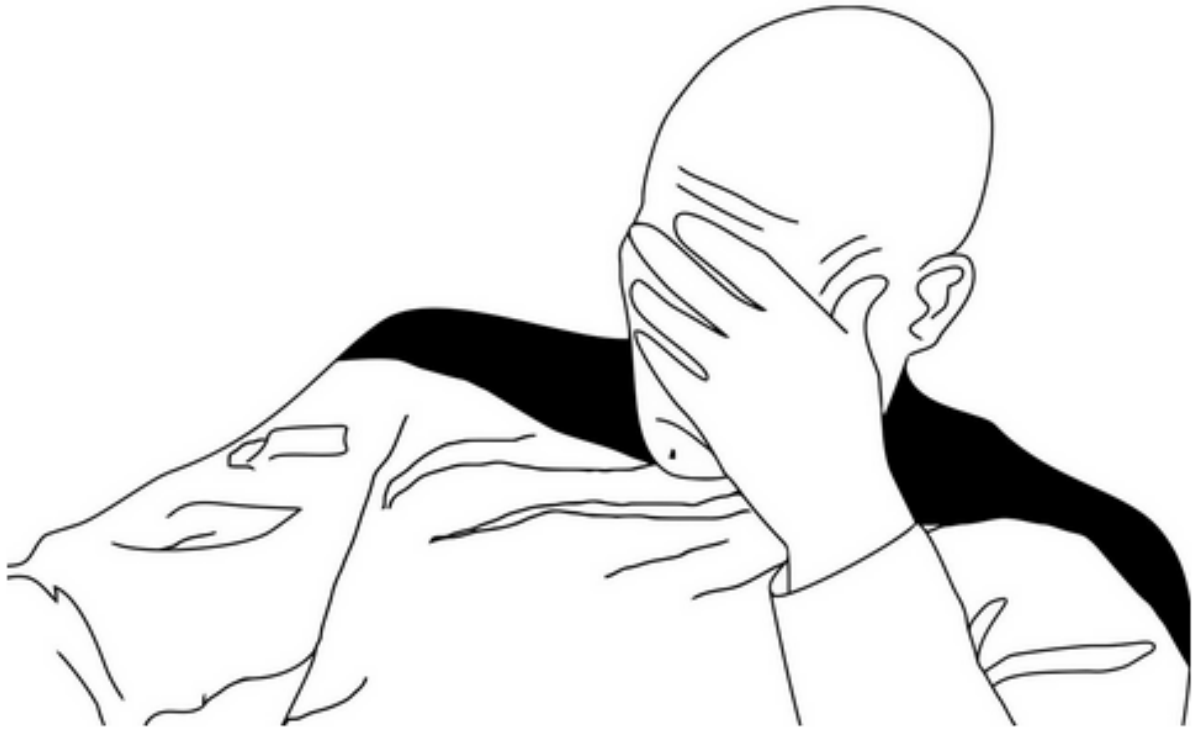
we're an inventive bunch, so we came up with some "optimizations" (read, "hacks")

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

- **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

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



Ok, that sucked. *Let's* ***fix HTTP*** *instead...*

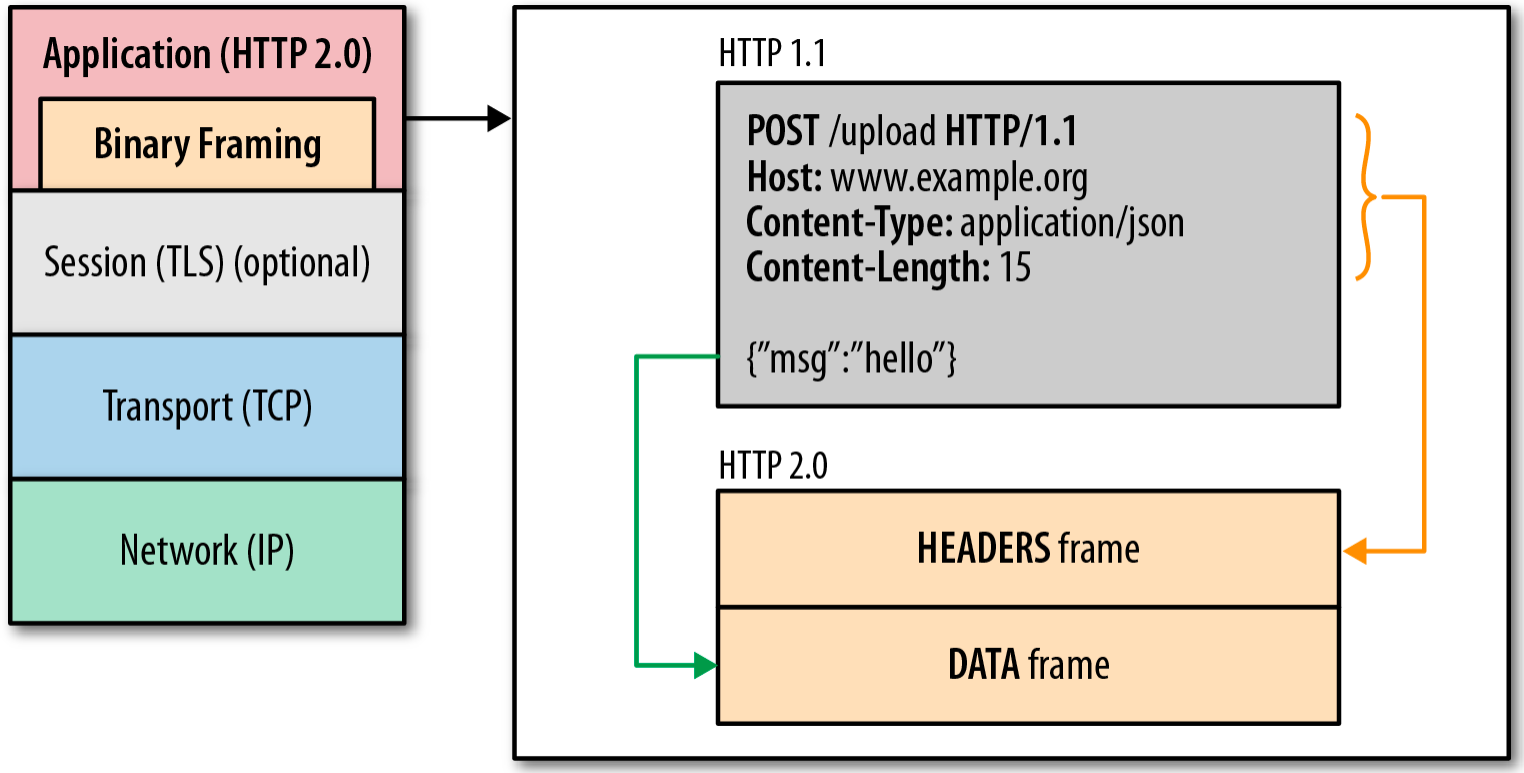
*"**HTTP 2.0** is a protocol designed for **low-latency transport of content** over the World Wide Web"*

- Improve end-user perceived latency
- Address the "head of line blocking"
- Not require multiple connections
- Retain the semantics of HTTP/1.1

HTTP 2.0 work kicked off in Jan 2012.

HTTP 2.0 in one slide...

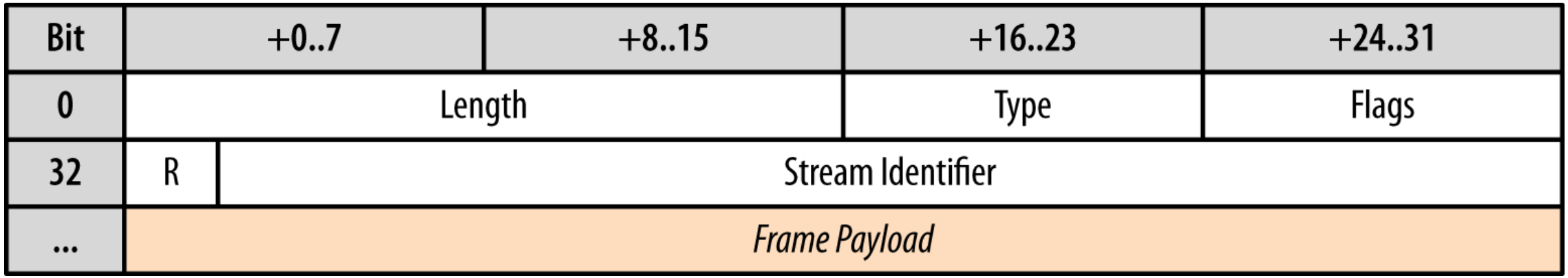
- **One TCP connection**
- **Request = Stream**
  - Streams are multiplexed
  - Streams are prioritized
- **Binary framing layer**
  - Prioritization
  - Flow control
  - Server push
- **Header compression**



*"... **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 redefining 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 (HTTPbis chair)

Binary framing crash course in one slide...

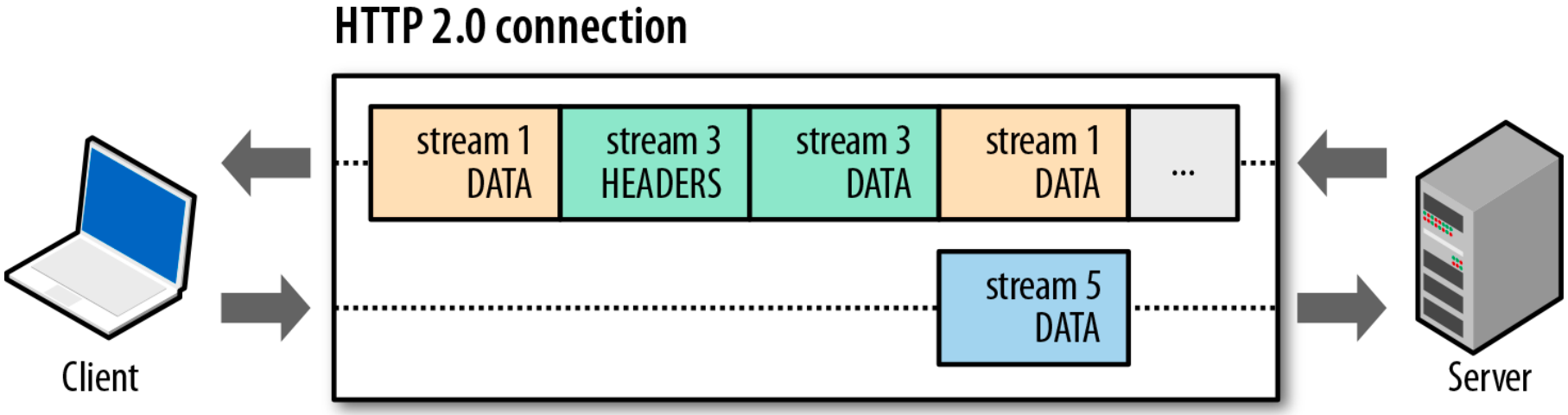


- Length-prefixed frames
- All frames have same 8-byte header

- **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 = buffer.read(8)
if frame_i_care_about
  do_something_smart
else
  buffer.skip(frame.length)
end
```

Basic data flow in HTTP 2.0...

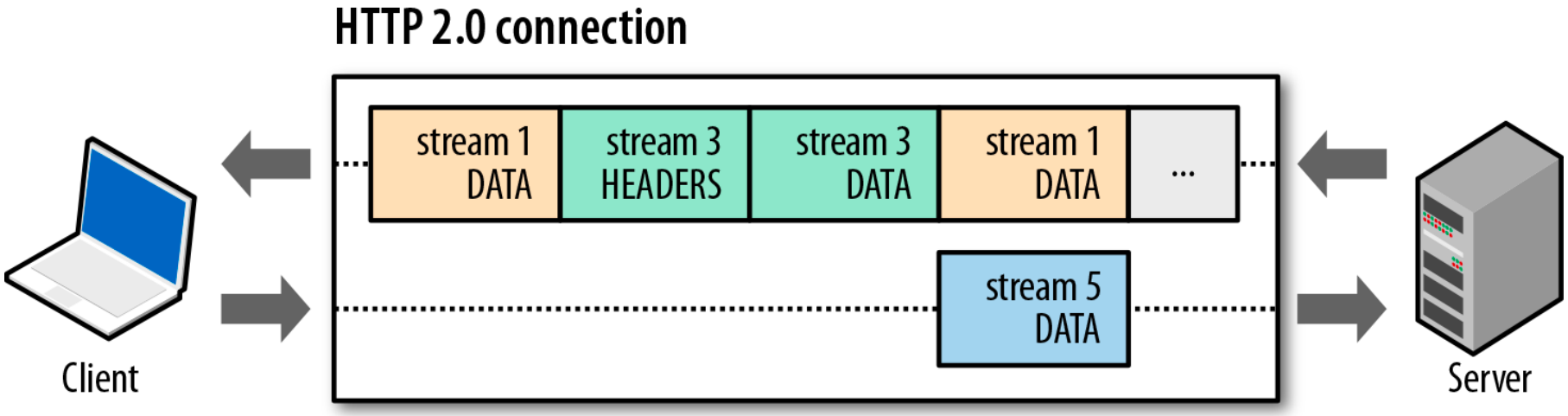


- **Streams are multiplexed** by splitting communication into frames
  - e.g. HEADERS, DATA, etc.

- **Frames are interleaved**
  - Frames can be prioritized
  - Frames can be flow controlled

*E.g. Please send **critical.css** with **priority 1**, please! **kittens.jpg** with **priority 10**.*

Connection + stream flow-control!

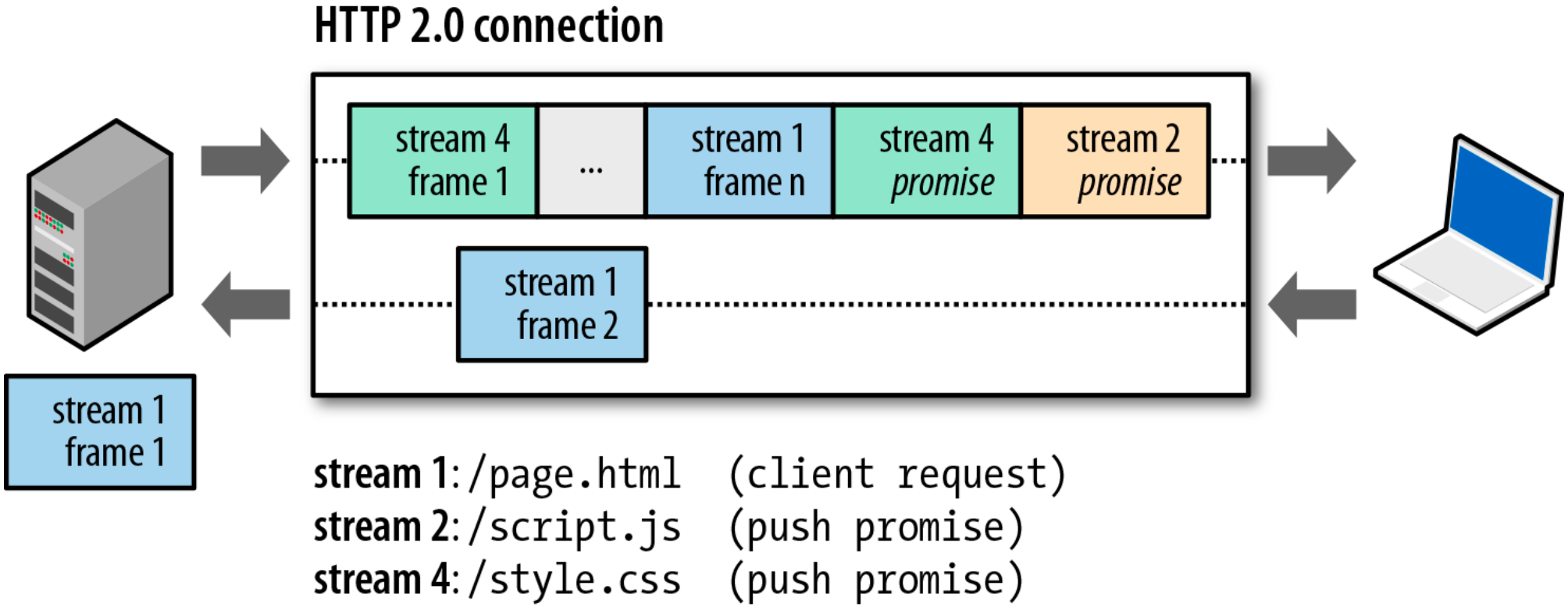


Stream flow-control enables **fine-grained resource control** between streams. E.g...

- **T(0):** I am willing to receive **64 KB** of kittens.jpg.
- **T(0):** I am willing to receive **500KB** of critical.js
- ...
- **T(n):** Ok, now send the **remainder** of kittens.jpg.

Client controls how and when the stream and connection window is incremented!

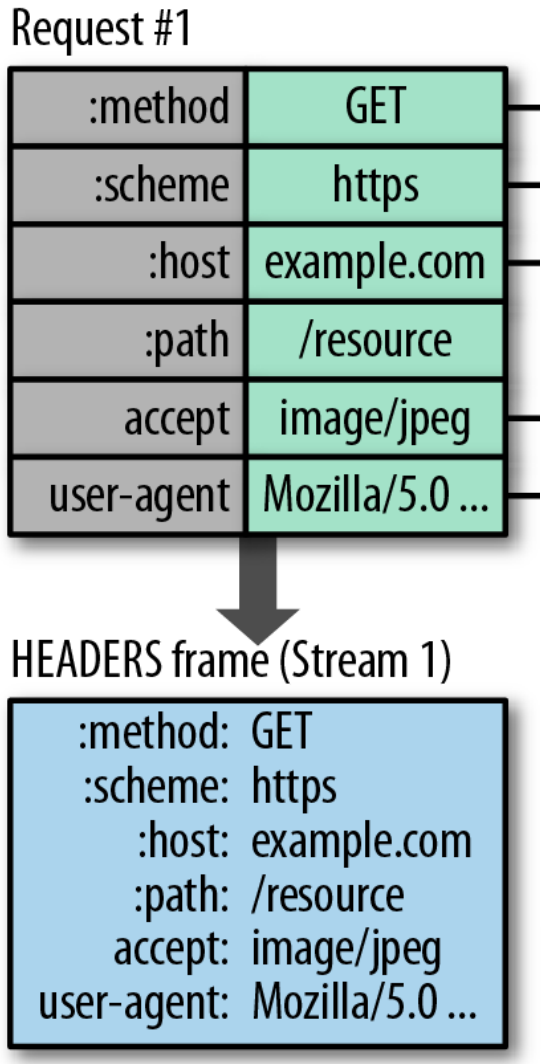
Server push... is replacing inlining.



*Inlining **is** server push. Except, **HTTP 2.0 server push is cacheable!***

- One request, multiple responses.
- Push redirects, cache invalidations, ... lots of new and exciting possibilities!

HTTP 2.0 provides header compression!



- Both sides maintain “header tables”
- New requests “toggle” or “insert” new values into the table

- New header set is a “diff” of the previous set of headers
- Repeat request (polling) with exact same headers incurs **no overhead**

Byte cost of new stream: ***8 bytes! ****

** as low as 8 bytes for an identical request.*

tl;dr: framing, mux, prioritization, flow control

same HTTP protocol semantics, complete performance reboot!

Benefits

- `min(request overhead)` = 8 bytes
- `max(parallelism)` = 100~1000 streams
- `max(client queueing latency)` = 0 ms

- Stream multiplexing
- Stream prioritization
- Stream flow control
- Server push

- **Plus all the usual HTTP goodies!**
  - Browser caching, authentication, content negotiation, content-encoding, transfer encoding, encryption, ...

*Available in your browser circa ~2014! **

** or today, via SPDY.*

Opportunities

- **Unshard, un-concat, un-sprite, ...**
  - Modular code, modular assets
  - Improved cache performance
  - Improved page load times

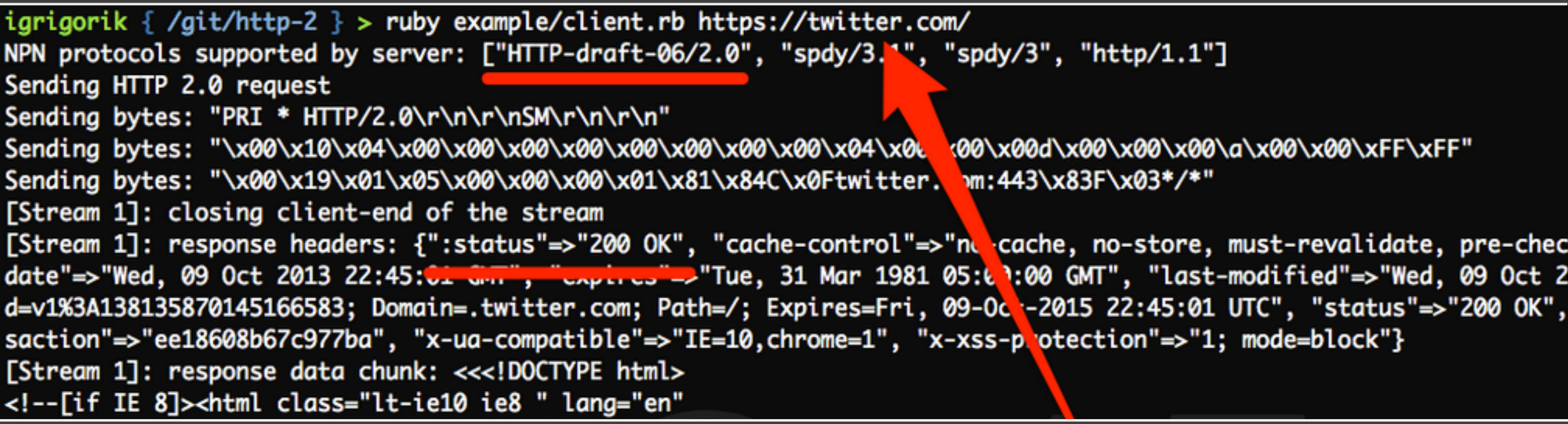
- **Layer other browser transports**
  - XHR / SSE: work out of the gate
  - (TODO) WebSockets over HTTP 2.0: free mux, flow control, prioritization

- **We need smarter servers!**
  - Support for prioritization, flow control, and resource push!

- **One RPC stack to rule them all**
  - Standardized, high-performance RPC protocol stack (internal, external)

*<crazy idea> **HTTP 2.0 over UDP? QUIC!** </crazy idea> **

** we'll leave that for RealTimeConf 2014 ;-)*



github.com/igrigorik/http-2

HTTP 2.0 draft 6 implementations in Firefox, Chrome, IE11 has SPDY support (stepping stone). Plus client/server implementations in C, JavaScript (node.js), Ruby, Perl, etc.

HTTP 2.0 is coming to a client / server near you in 2014.

HTTP 2.0: hpbn.co/http2

Ilya Grigorik
igrigorik@google.com
@igrigorik