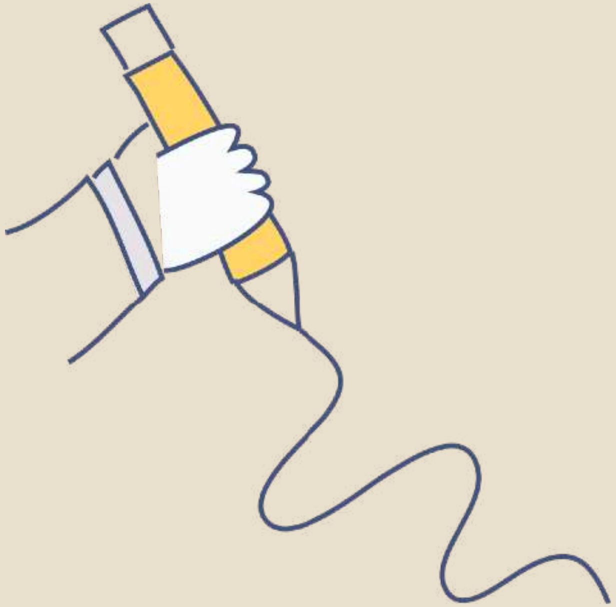
http2和quic那些事儿

xiaorui.cc

menu



01 http 1.0 vs 1.1

02 http 2.0

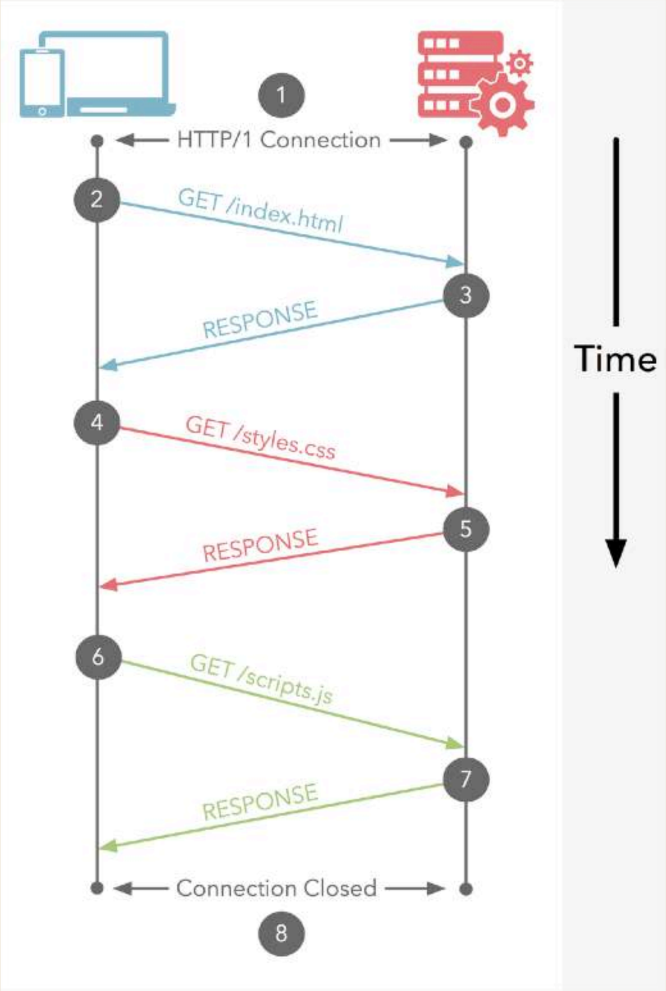
03 quic

http1.0 vs http1.1

- http 1.0
  - 古董 ...

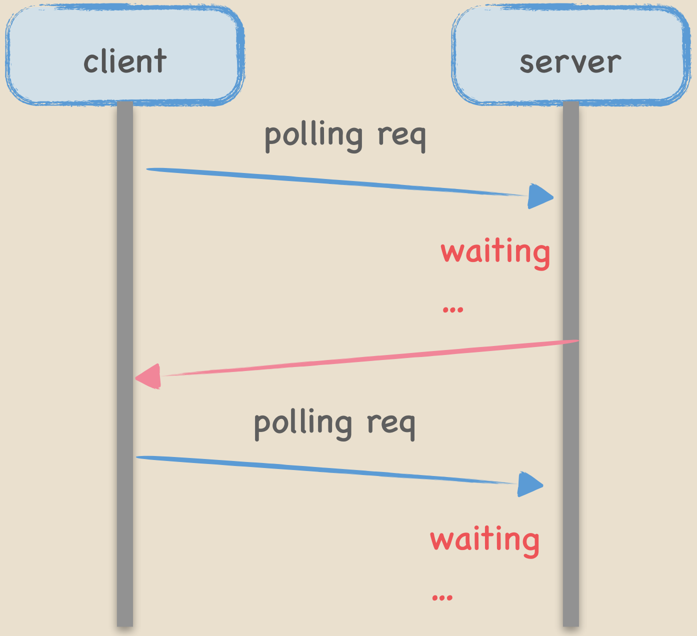
- http 1.1
  - 持久化连接
  - 缓存
  - accept-range
  - 更多header语义

http 1.1 的缺点

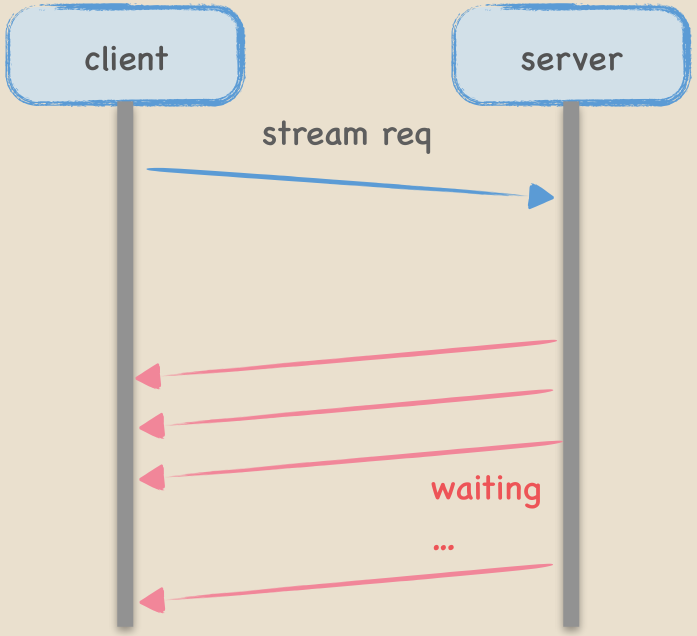


- Head-of-Line Blocking (No Pipelining)
  一个连接同一时间只能处理一个请求
- 如果当前请求阻塞，那么该连接就无法复用
- http 1.1 定义的pipeline, 浏览器和开发者都不友好

http 1.1 流处理

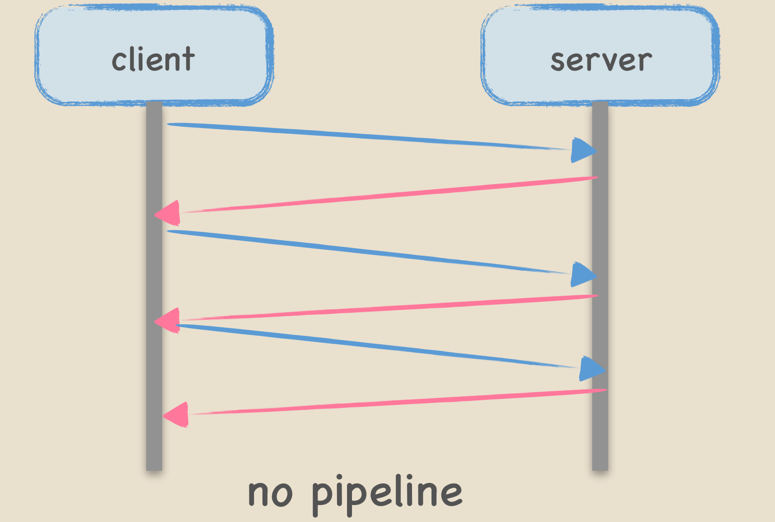


- http long poll
- http long streaming

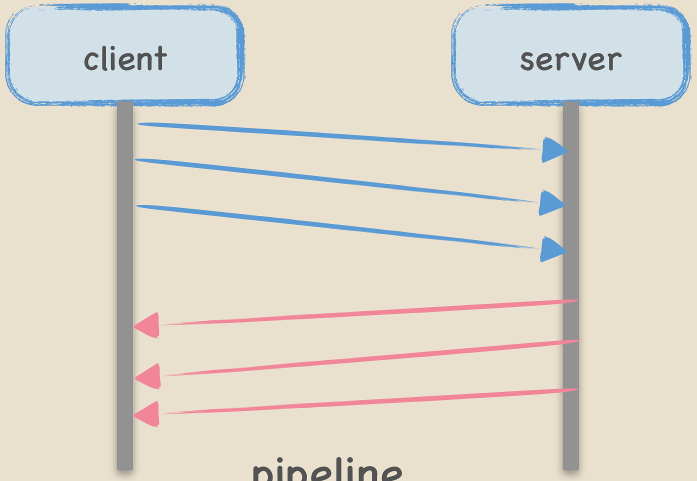


独享连接 !!!

http 1.1 pipeline



no pipeline



pipeline

- 未完全解决head of line blocking
- fifo原则, 需要等待最后的响应
- 多数http proxy不支持
- 多数浏览器默认关闭 h1.1 pipeline

Google Report

https://www.chromium.org/developers/design-documents/network-stack/http-pipelining

优化时延

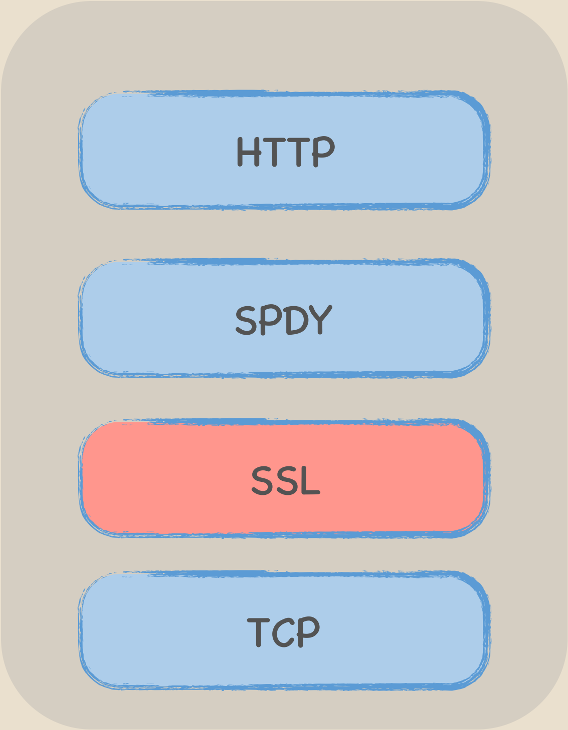
在浏览器多开连接，提高吞吐？ 同域名下chrome的连接数限制在6个

- 时延
- 吞吐
- 流量

- 资源合并 / nginx concat
- base64切图 （ 精灵图 ）
- 多域名拆分
- 压缩数据 (去除无用字符，gzip压缩)
- cookie free
- …

短暂的spdy

解决了http1.1的线头阻塞的问题 !!!



HTTP

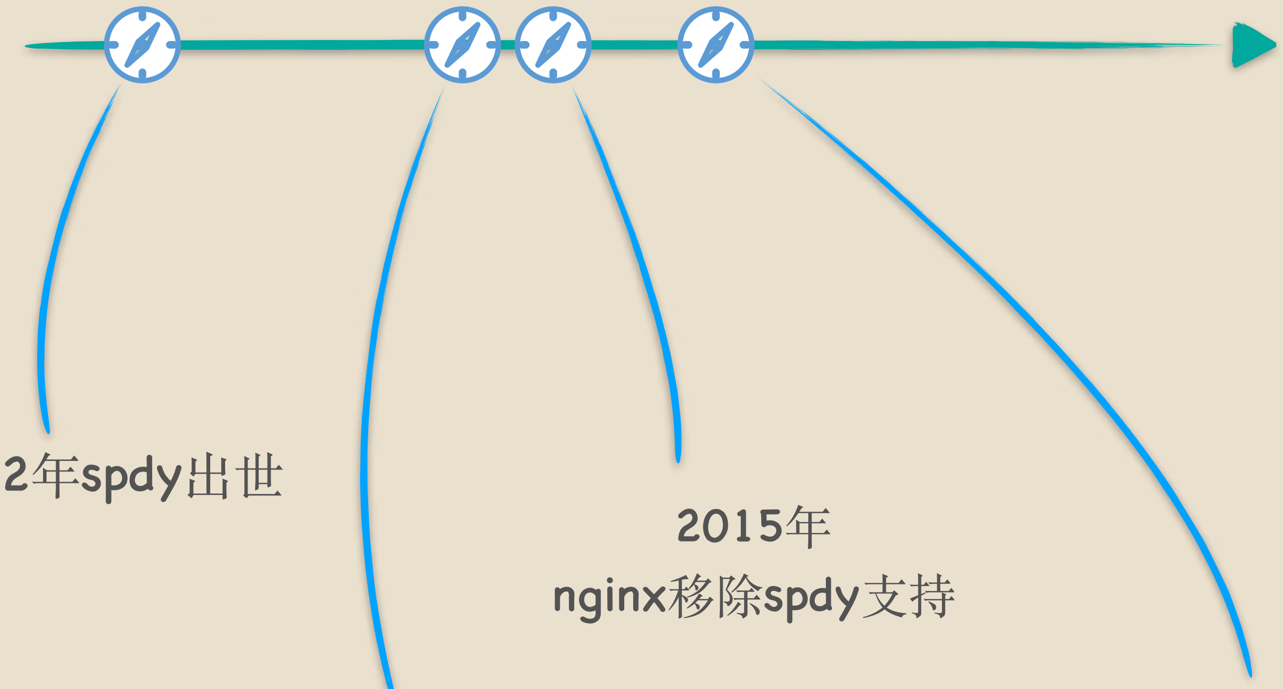
SPDY

SSL

TCP

- 功能
  - 多路复用
  - 优先级
  - header压缩
  - 推送
  - ...

短暂的spdy



2012年spdy出世

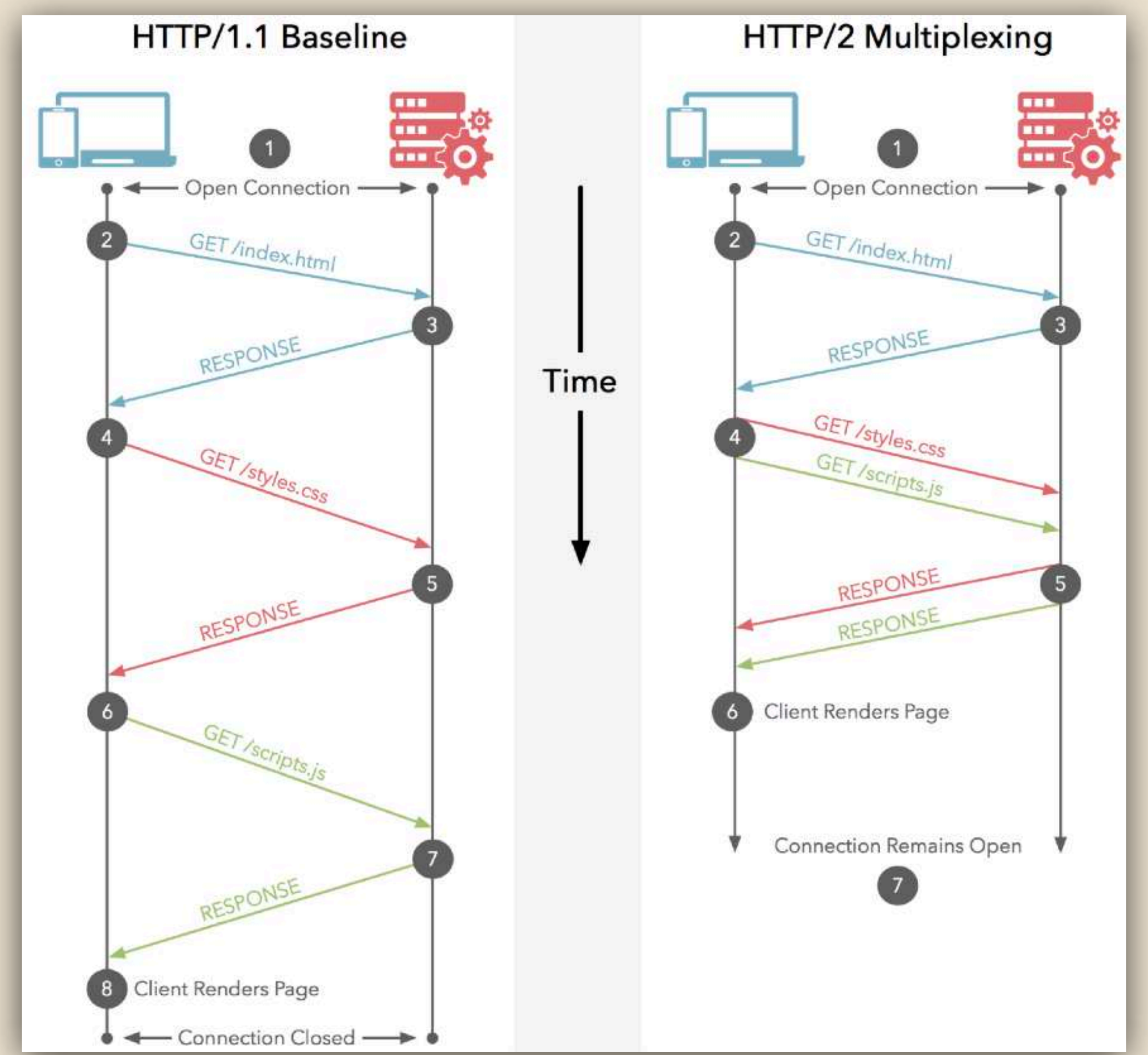
2015年http2出世

2015年
nginx移除spdy支持

2016年
chrome移除spdy支持

- spdy vs http2
  - header压缩算法不同
  - 完善控制协议
  - 支持明文HTTP传输
  - …

http1.1 vs http2.0

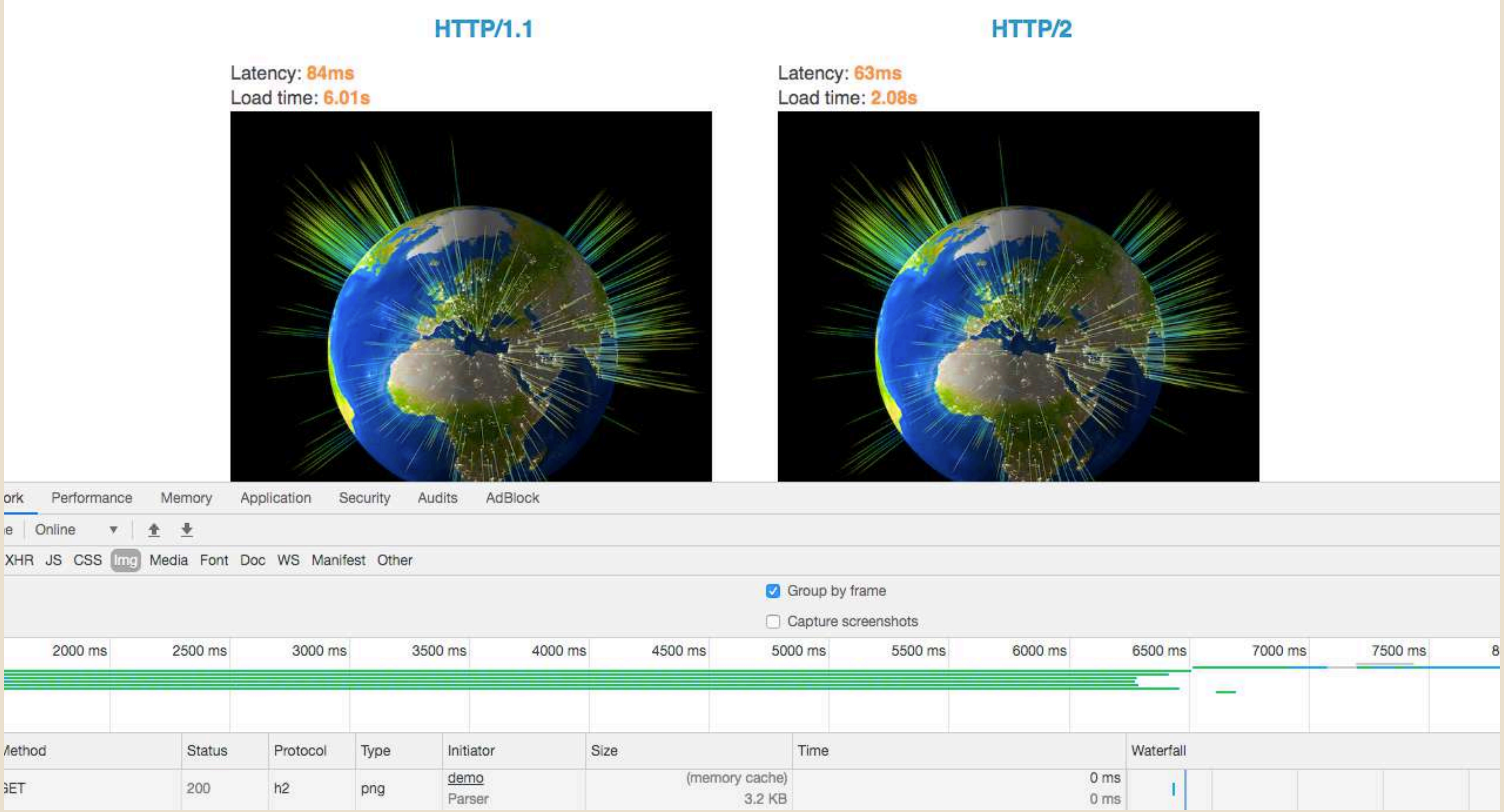


时延！

等一个是等, 多个一起等也是等 ！

- ✱ http 1.1
  - ✱ N个排队请求 >≈ N x latency
- ✱ http 2.0
  - ✱ N个并发请求 >≈ N x latency

http 1.1 vs 2.0



HTTP/1.1

Latency: 84ms
Load time: 6.01s

HTTP/2

Latency: 63ms
Load time: 2.08s

☑ Group by frame

□ Capture screenshots

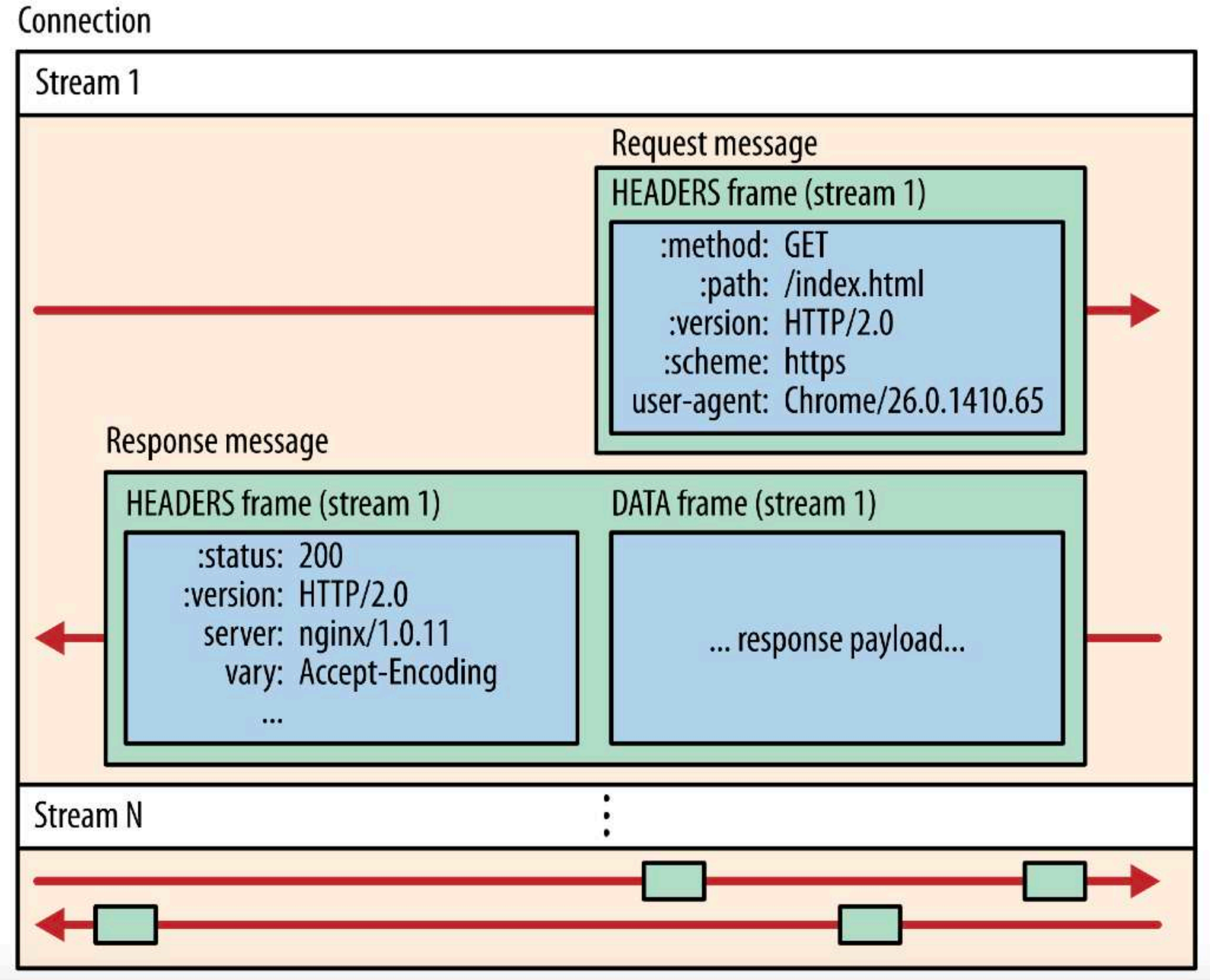
ethod	Status	Protocol	Type	Initiator	Size	Time	Waterfall
ET	200	h2	png	demo Parser	(memory cache) 3.2 KB	0 ms 0 ms	

http 2.0 优点

- 请求/返回多路复用
- header压缩
- 流控
- 优先级
- 服务端推送
- ...

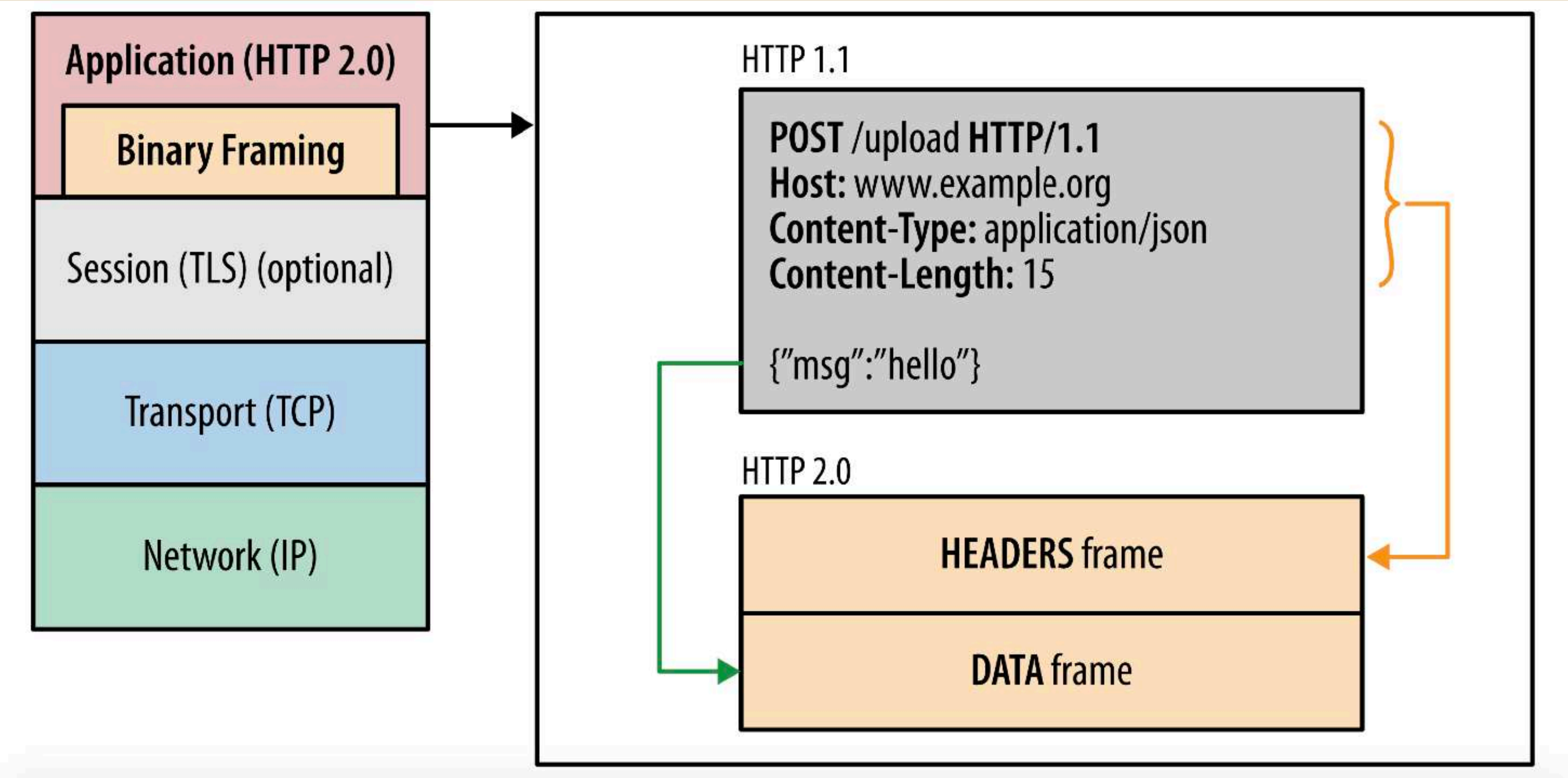
HTTP/2

概念



- stream
  - 一个完整的请求和响应的字节流
- message
  - 一个完整的http请求或响应，由一个或多个帧组成。
- frame
  - 通信的最小单位, 每个帧都包含帧头，可标识出当前帧所属的数据流。

二进制分帧层



Application (HTTP 2.0)

Binary Framing

Session (TLS) (optional)

Transport (TCP)

Network (IP)

HTTP 1.1

```
POST /upload HTTP/1.1
Host: www.example.org
Content-Type: application/json
Content-Length: 15

{"msg":"hello"}
```

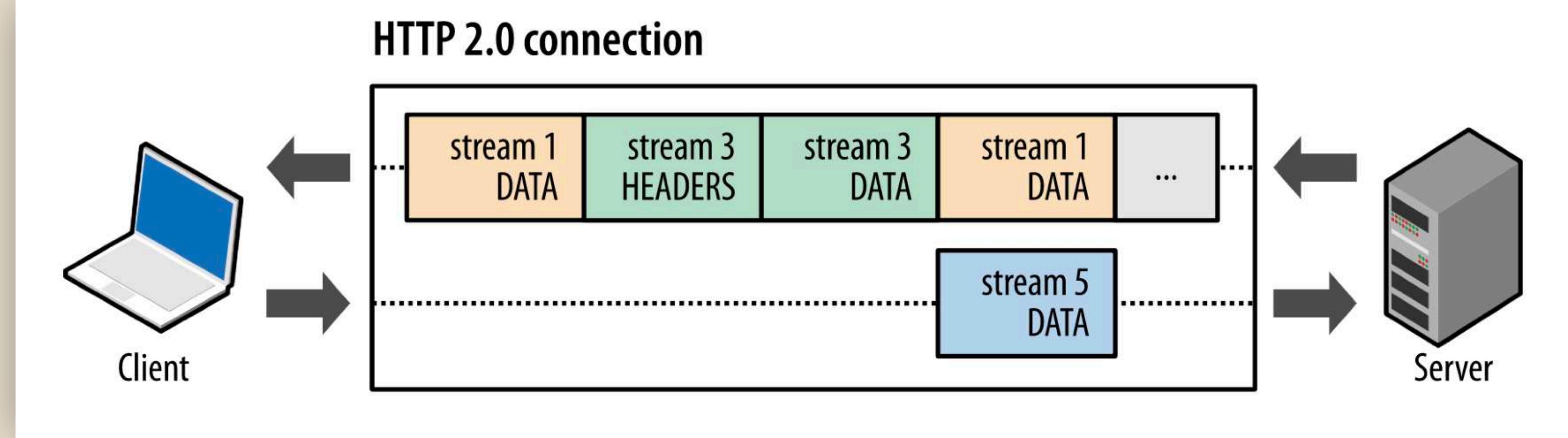
HTTP 2.0

HEADERS frame

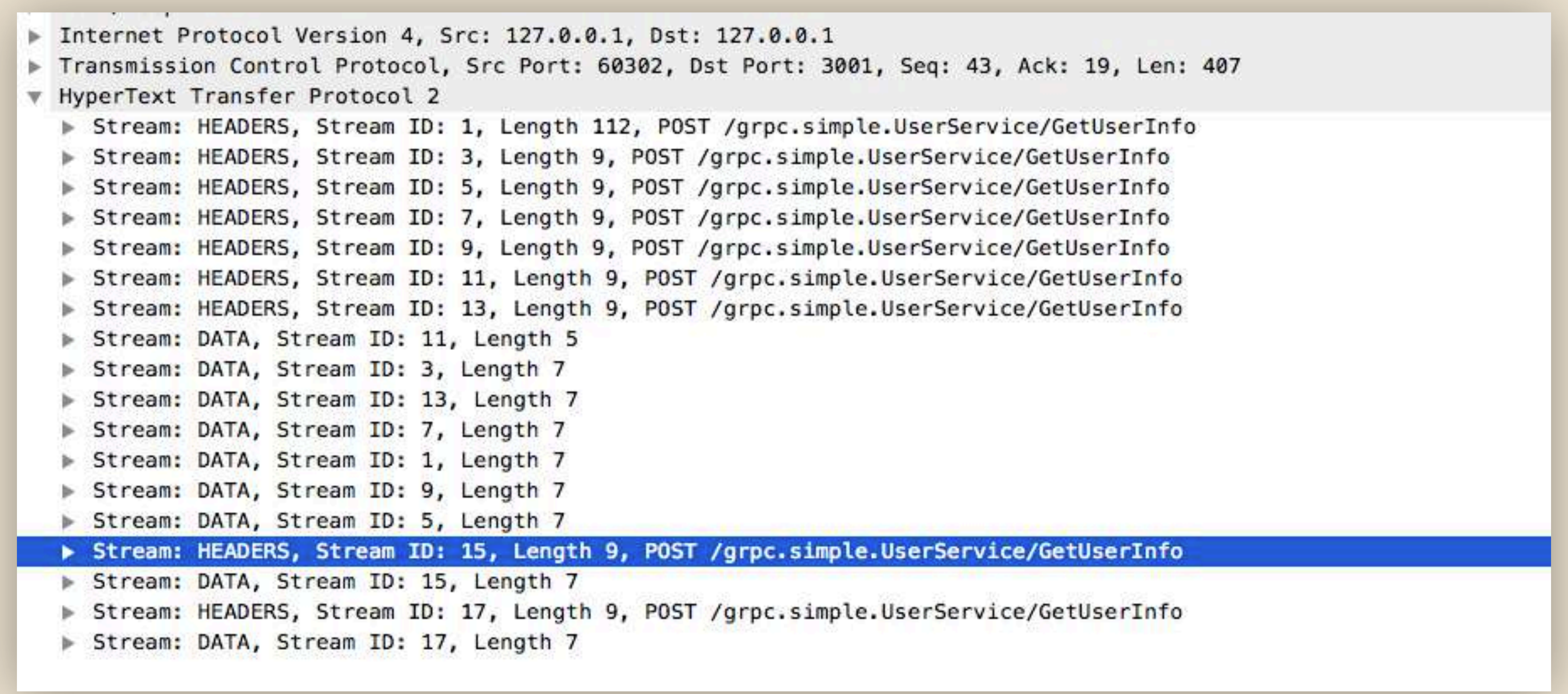
DATA frame

多路复用

- 并行交错地发送多个请求，请求之间互不影响。
- 并行交错地发送多个响应，响应之间互不干扰。
- 使用一个连接并行发送多个请求和响应。



多路复用



```
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 60302, Dst Port: 3001, Seq: 43, Ack: 19, Len: 407
▼ HyperText Transfer Protocol 2
    ▶ Stream: HEADERS, Stream ID: 1, Length 112, POST /grpc.simple.UserService/GetUserInfo
    ▶ Stream: HEADERS, Stream ID: 3, Length 9, POST /grpc.simple.UserService/GetUserInfo
    ▶ Stream: HEADERS, Stream ID: 5, Length 9, POST /grpc.simple.UserService/GetUserInfo
    ▶ Stream: HEADERS, Stream ID: 7, Length 9, POST /grpc.simple.UserService/GetUserInfo
    ▶ Stream: HEADERS, Stream ID: 9, Length 9, POST /grpc.simple.UserService/GetUserInfo
    ▶ Stream: HEADERS, Stream ID: 11, Length 9, POST /grpc.simple.UserService/GetUserInfo
    ▶ Stream: HEADERS, Stream ID: 13, Length 9, POST /grpc.simple.UserService/GetUserInfo
    ▶ Stream: DATA, Stream ID: 11, Length 5
    ▶ Stream: DATA, Stream ID: 3, Length 7
    ▶ Stream: DATA, Stream ID: 13, Length 7
    ▶ Stream: DATA, Stream ID: 7, Length 7
    ▶ Stream: DATA, Stream ID: 1, Length 7
    ▶ Stream: DATA, Stream ID: 9, Length 7
    ▶ Stream: DATA, Stream ID: 5, Length 7
    ▶ Stream: HEADERS, Stream ID: 15, Length 9, POST /grpc.simple.UserService/GetUserInfo
    ▶ Stream: DATA, Stream ID: 15, Length 7
    ▶ Stream: HEADERS, Stream ID: 17, Length 9, POST /grpc.simple.UserService/GetUserInfo
    ▶ Stream: DATA, Stream ID: 17, Length 7
```

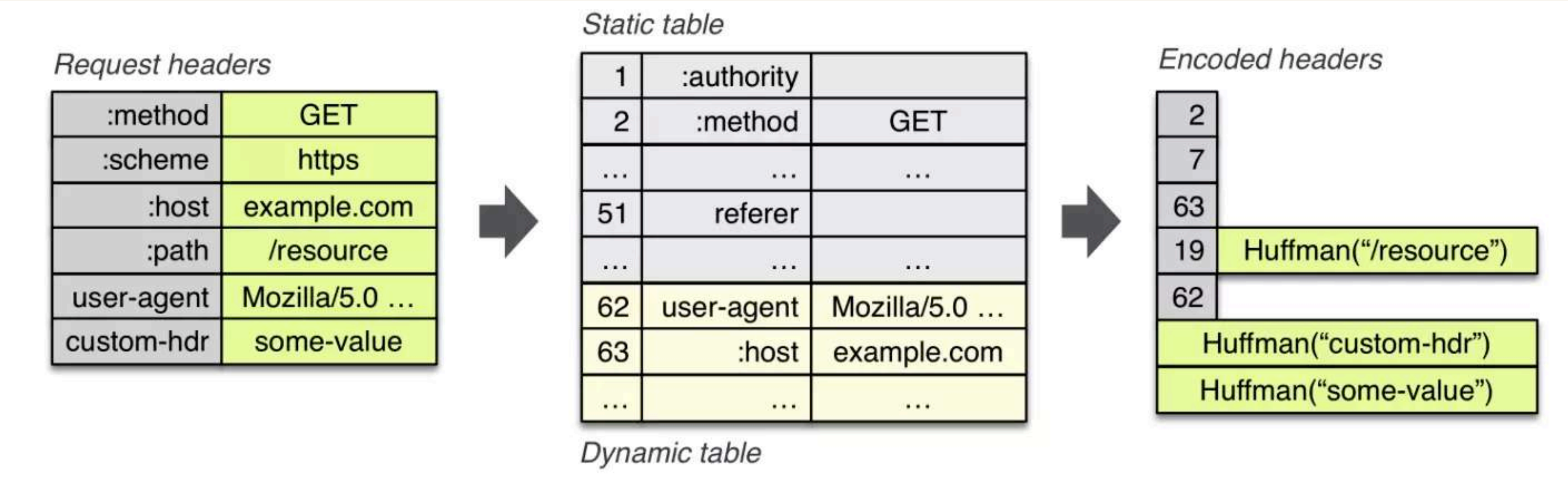
流量控制

- 只能针对data frame帧
- 初始化大小为 65535 字节 (2^16 - 1)
- 可以针对连接限制 (stream id = 0)
- 也可针对流限制 (stream id > 0)
- Data Frame大小为16K字节

均衡分配可用的网络资源，尽可能让所有流拿到调度

头部压缩

- 通过rfc商定静态和动态表的扩展, header通过传递索引号节省空间
- 使用Huffman进行编码压缩



Request headers

:method	GET
:scheme	https
:host	example.com
:path	/resource
user-agent	Mozilla/5.0 …
custom-hdr	some-value

Static table

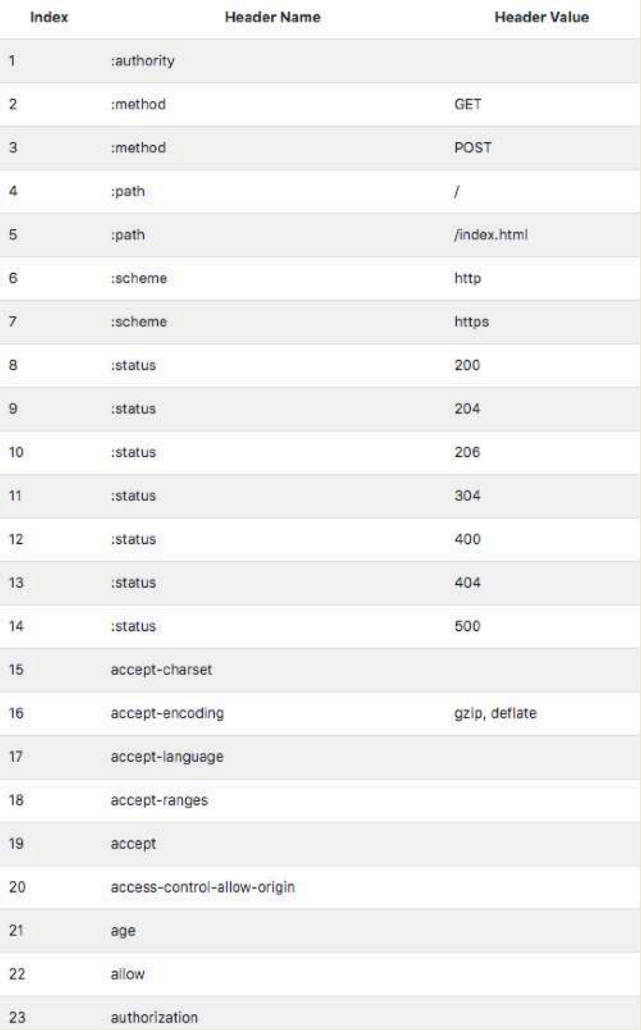
1	:authority	
2	:method	GET
…	…	…
51	referer	
…	…	…
62	user-agent	Mozilla/5.0 …
63	:host	example.com
…	…	…

Dynamic table

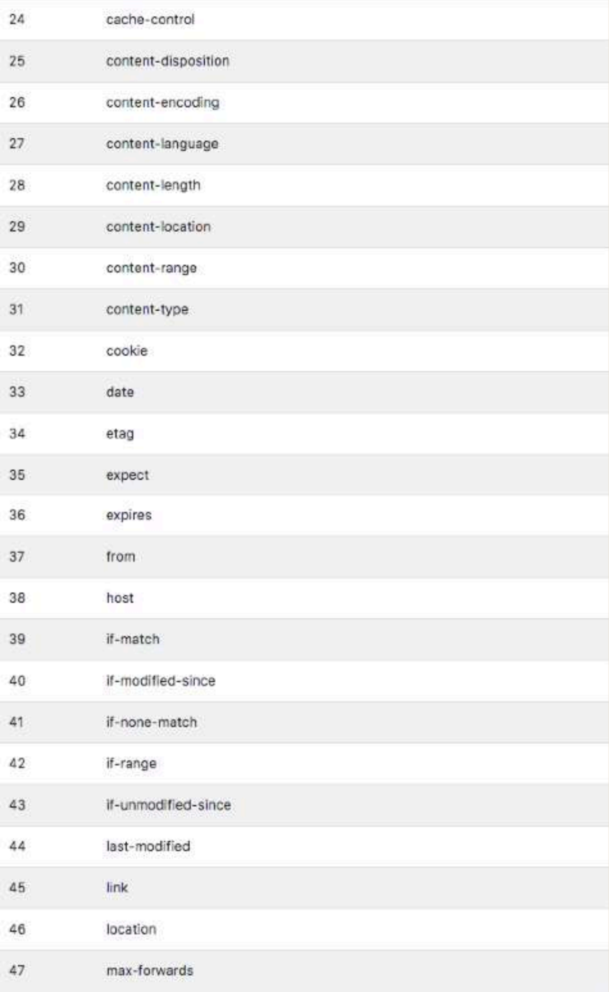
Encoded headers

2	
7	
63	
19	Huffman(“/resource”)
62	
Huffman(“custom-hdr”)	
Huffman(“some-value”)	

Hpack static

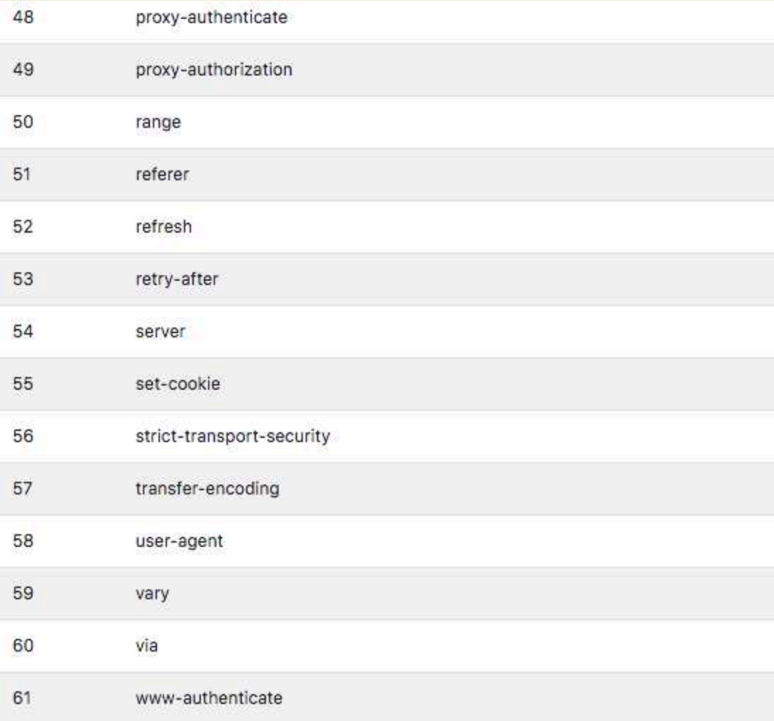


Index	Header Name	Header Value
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/
5	:path	/index.html
6	:scheme	http
7	:scheme	https
8	:status	200
9	:status	204
10	:status	206
11	:status	304
12	:status	400
13	:status	404
14	:status	500
15	accept-charset	
16	accept-encoding	gzip, deflate
17	accept-language	
18	accept-ranges	
19	accept	
20	access-control-allow-origin	
21	age	
22	allow	
23	authorization	



24	cache-control
25	content-disposition
26	content-encoding
27	content-language
28	content-length
29	content-location
30	content-range
31	content-type
32	cookie
33	date
34	etag
35	expect
36	expires
37	from
38	host
39	if-match
40	if-modified-since
41	if-none-match
42	if-range
43	if-unmodified-since
44	last-modified
45	link
46	location
47	max-forwards

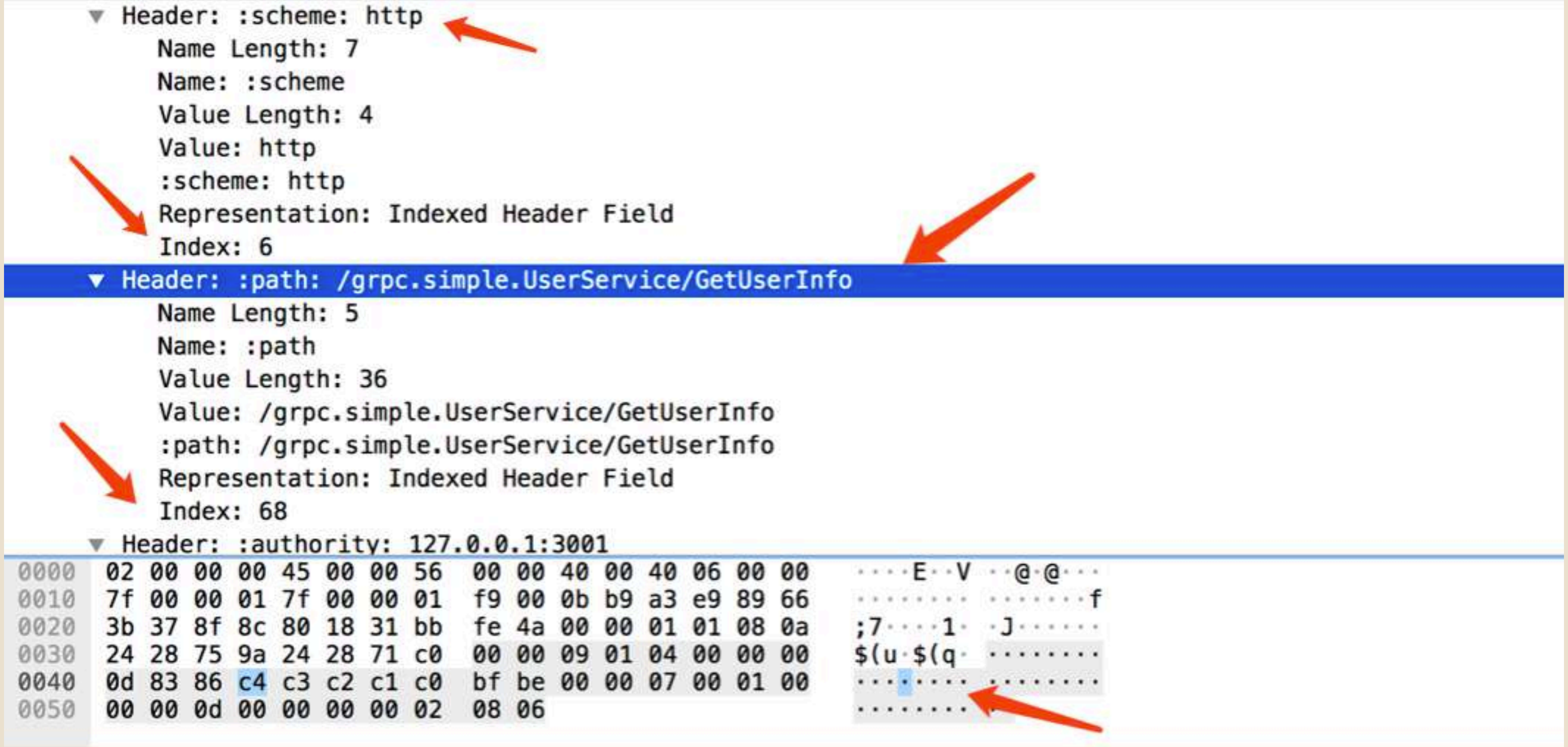
- ✱ 含有61个映射
- ✱ 也可对key单独映射



48	proxy-authenticate
49	proxy-authorization
50	range
51	referer
52	refresh
53	retry-after
54	server
55	set-cookie
56	strict-transport-security
57	transfer-encoding
58	user-agent
59	vary
60	via
61	www-authenticate

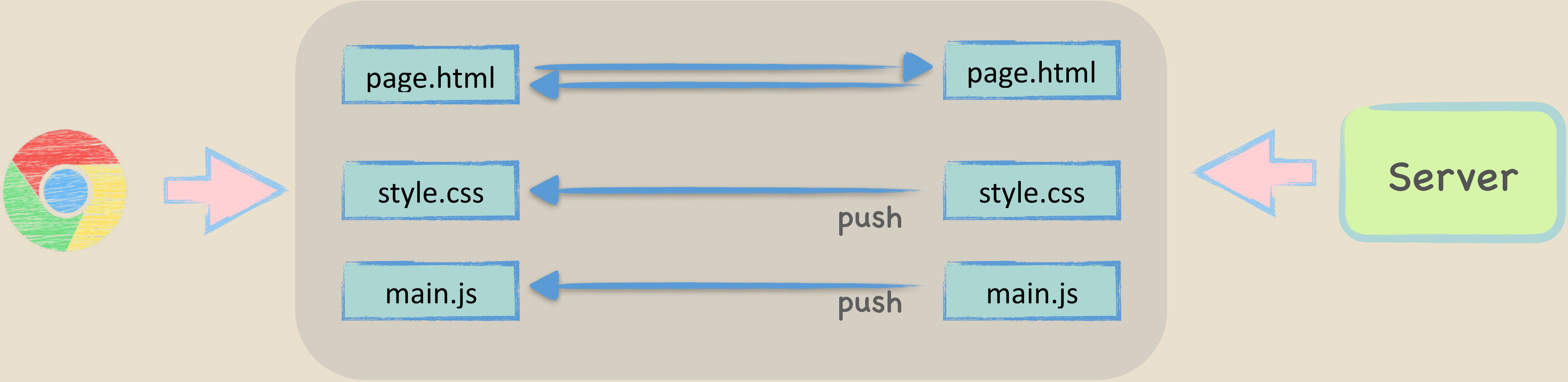
http://http2.github.io/http2-spec/compression.html#static.table.definition

wirshark hpack



```
▼ Header: :scheme: http
    Name Length: 7
    Name: :scheme
    Value Length: 4
    Value: http
    :scheme: http
    Representation: Indexed Header Field
    Index: 6
▼ Header: :path: /grpc.simple.UserService/GetUserInfo
    Name Length: 5
    Name: :path
    Value Length: 36
    Value: /grpc.simple.UserService/GetUserInfo
    :path: /grpc.simple.UserService/GetUserInfo
    Representation: Indexed Header Field
    Index: 68
▼ Header: :authority: 127.0.0.1:3001
0000  02 00 00 00 45 00 00 56  00 00 40 00 40 06 00 00   ····E··V ··@·@···
0010  7f 00 00 01 7f 00 00 01  f9 00 0b b9 a3 e9 89 66   ········ ·······f
0020  3b 37 8f 8c 80 18 31 bb  fe 4a 00 00 01 01 08 0a   ;7····1· ·J······
0030  24 28 75 9a 24 28 71 c0  00 00 09 01 04 00 00 00   $(u·$(q· ········
0040  0d 83 86 c4 c3 c2 c1 c0  bf be 00 00 07 00 01 00   ········ ········
0050  00 00 0d 00 00 00 00 02  08 06                     ········ ··
```

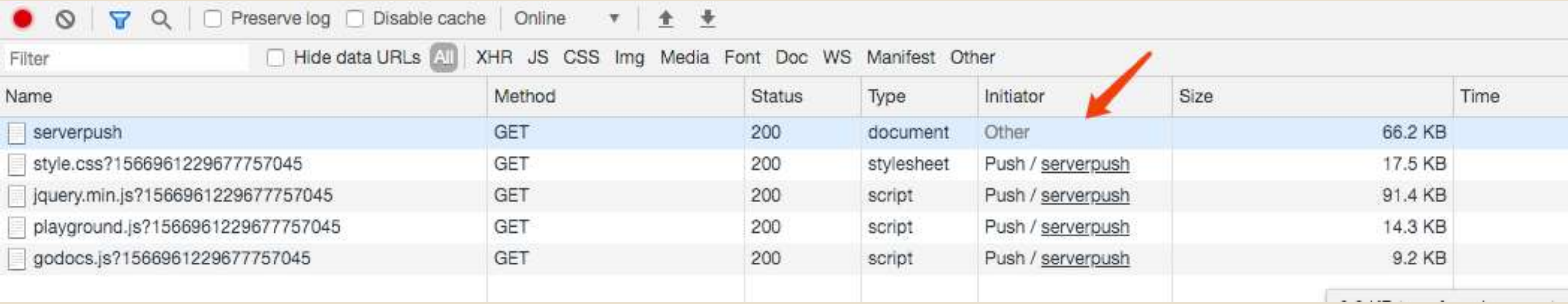
server push



- 当服务端需要主动推送某个资源时，便会发送一个 Frame Type 为 PUSH_PROMISE 的 Frame，里面带了 PUSH 需要新建的 Stream ID。

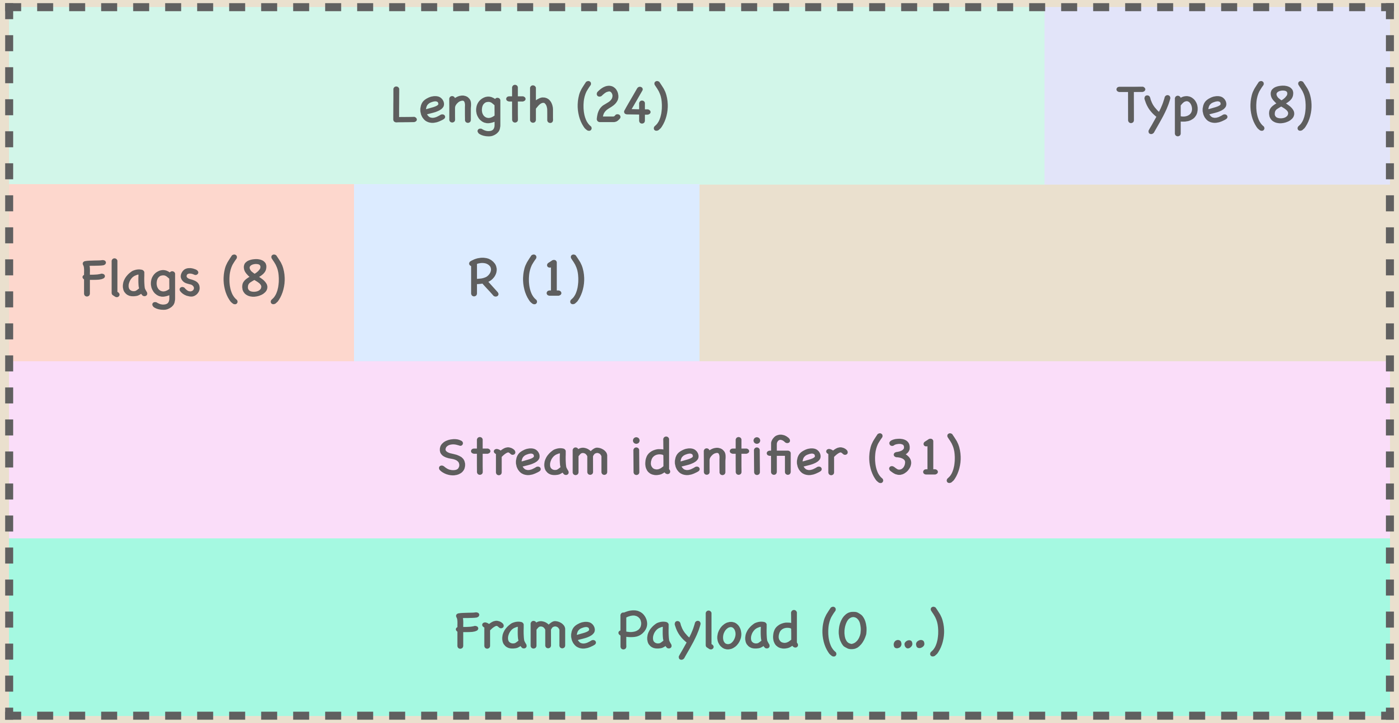
- 客户端接收该 Stream ID的数据就可以了

server push



Name	Method	Status	Type	Initiator	Size	Time
serverpush	GET	200	document	Other	66.2 KB	
style.css?1566961229677757045	GET	200	stylesheet	Push / serverpush	17.5 KB	
jquery.min.js?1566961229677757045	GET	200	script	Push / serverpush	91.4 KB	
playground.js?1566961229677757045	GET	200	script	Push / serverpush	14.3 KB	
godocs.js?1566961229677757045	GET	200	script	Push / serverpush	9.2 KB	

frame



Length (24)

Type (8)

Flags (8)

R (1)

Stream identifier (31)

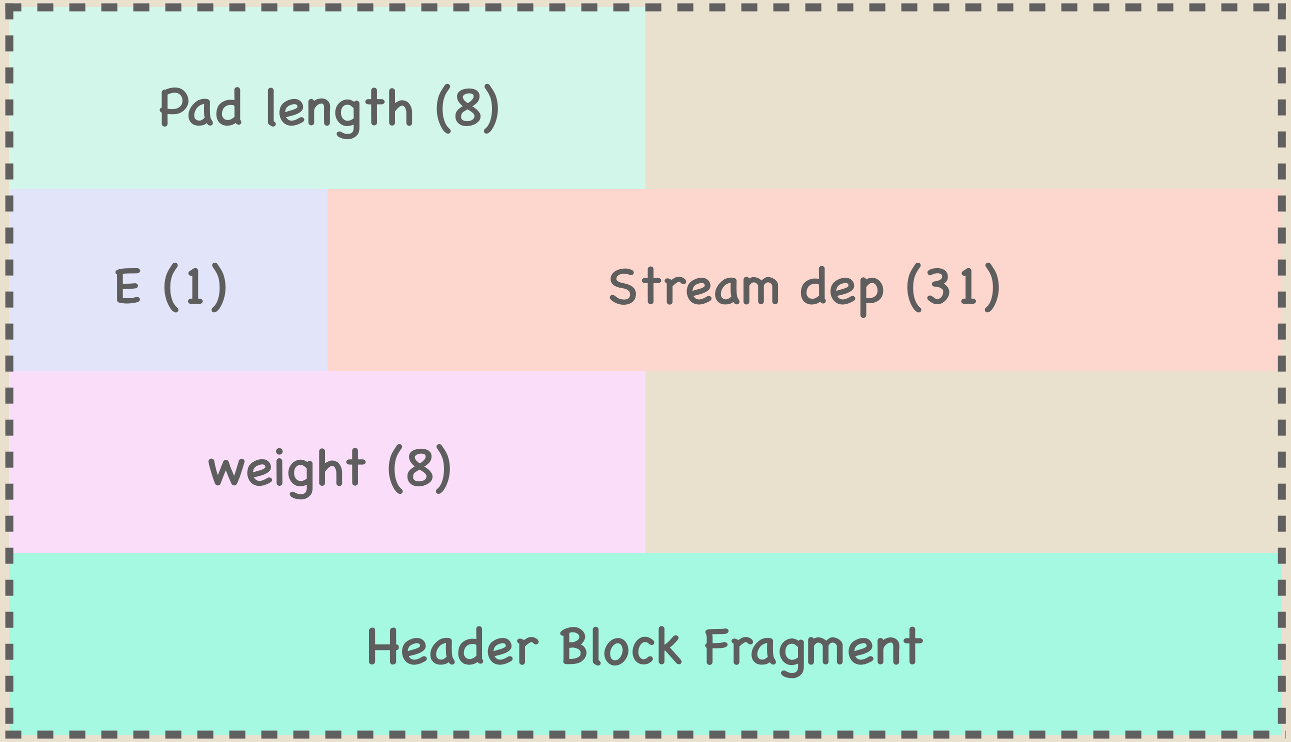
Frame Payload (0 ...)

- Length (payload size)
- Type (类型)
- Flags (状态)
- R (保留)
- Stream Identifier
- Frame Payload

frame types

- Header
- Data
- PRIORITY
  - 优先级
- RST_STREAM
  - 停止 (由于错误)
- SETTINGS
  - 连接级参数
- PUSH_PROMISE
  - 推送
- PING
- GOAWAY
  - 停止
- WINDOW_UPDATE
  - 流量控制
- CONTINUATION
  - 扩展header数据块

header frame



Pad length (8)

E (1)

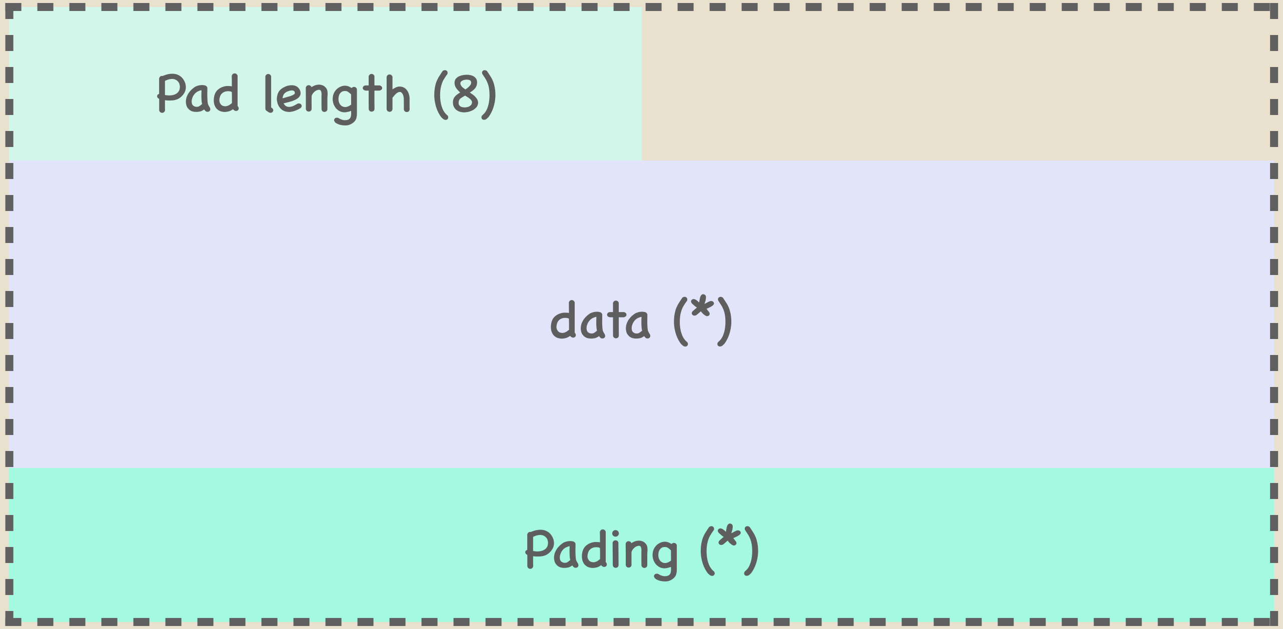
Stream dep (31)

weight (8)

Header Block Fragment

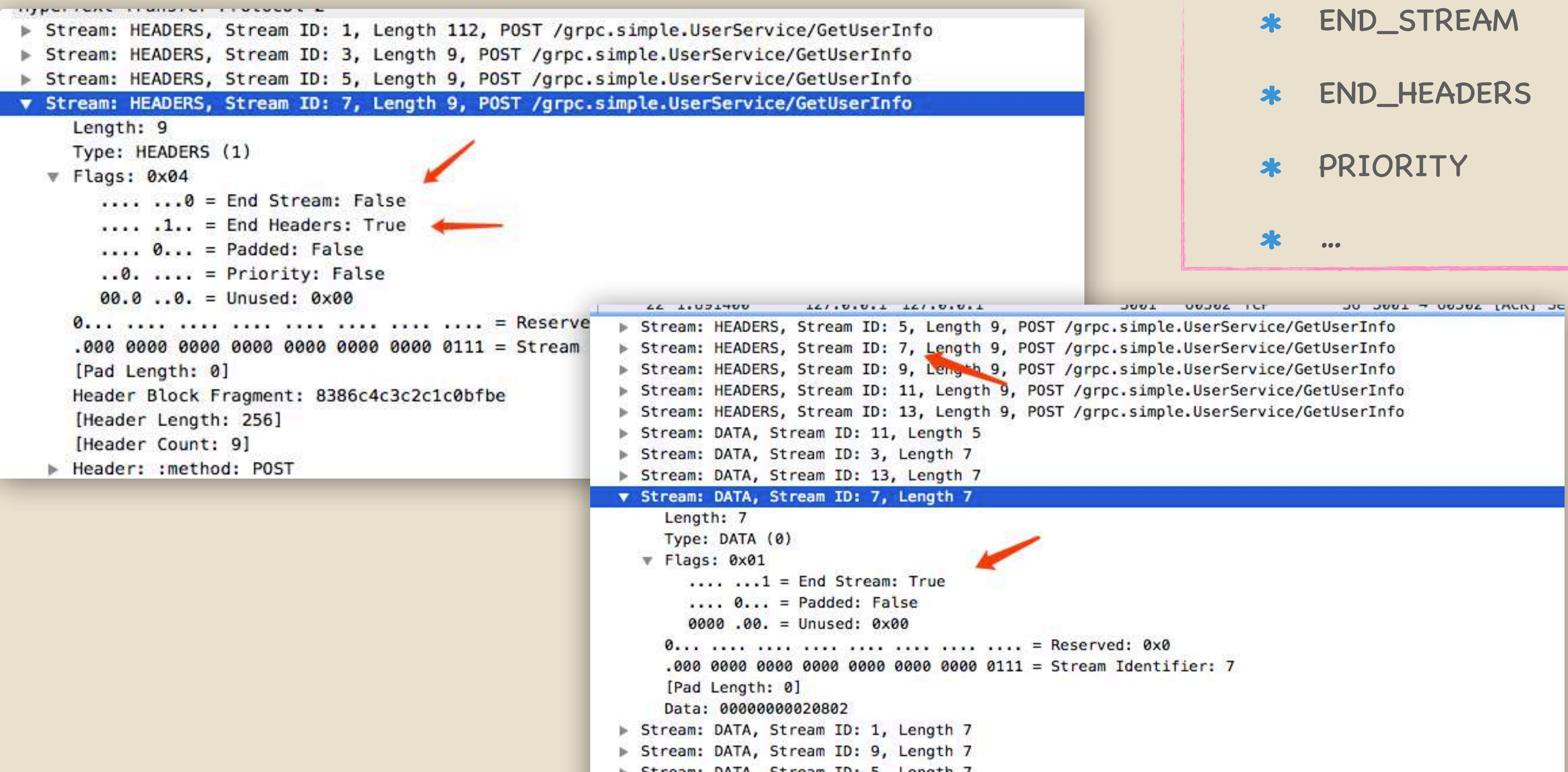
- Pad length
- E 依赖排他
- Stream dep
- weight 优先级 1-256
- Header Block Fragment 数据
- Padding 填充字节

data frame



- pad length
- data
- padding

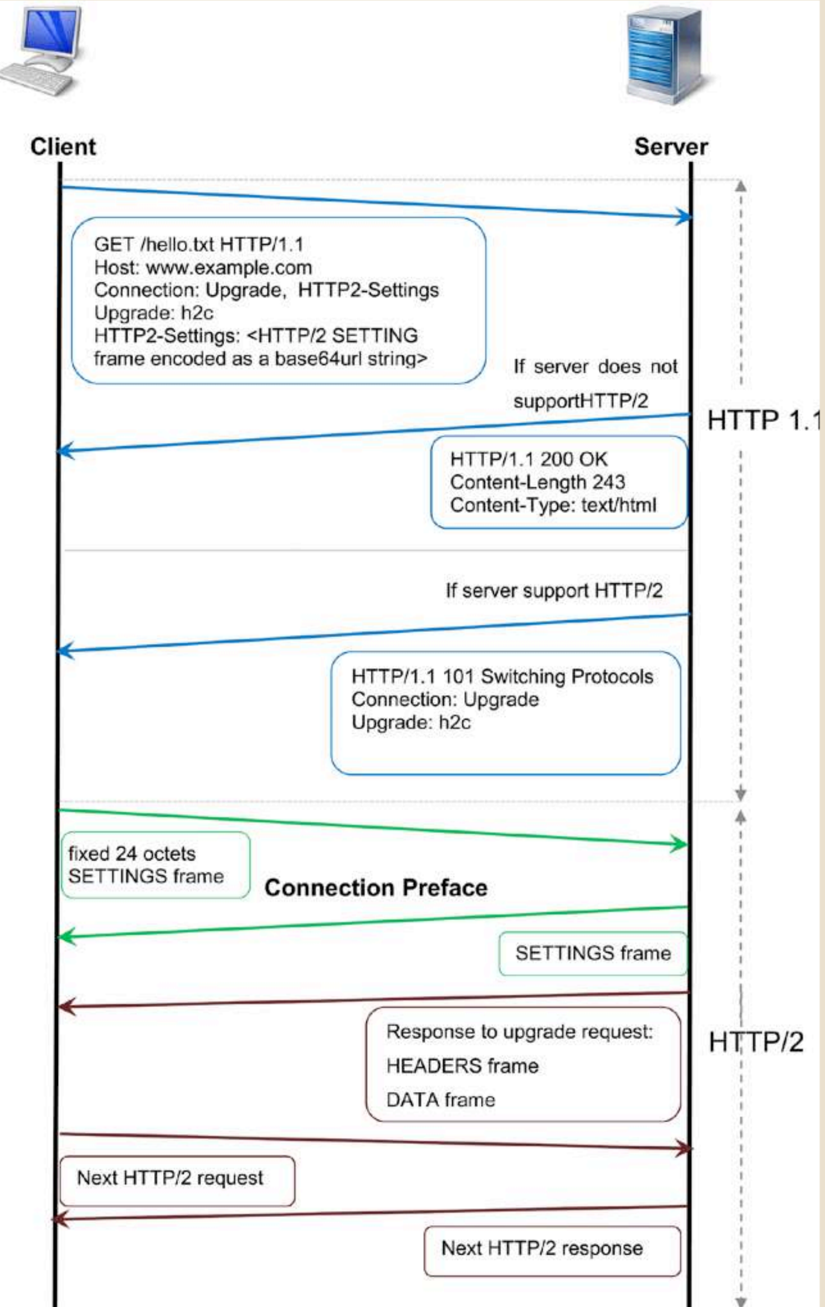
flags



```
Hypertext Transfer Protocol 2
▶ Stream: HEADERS, Stream ID: 1, Length 112, POST /grpc.simple.UserService/GetUserInfo
▶ Stream: HEADERS, Stream ID: 3, Length 9, POST /grpc.simple.UserService/GetUserInfo
▶ Stream: HEADERS, Stream ID: 5, Length 9, POST /grpc.simple.UserService/GetUserInfo
▼ Stream: HEADERS, Stream ID: 7, Length 9, POST /grpc.simple.UserService/GetUserInfo
    Length: 9
    Type: HEADERS (1)
  ▼ Flags: 0x04
      .... ...0 = End Stream: False
      .... .1.. = End Headers: True
      .... 0... = Padded: False
      ..0. .... = Priority: False
      00.0 ..0. = Unused: 0x00
    0... .... .... .... .... .... .... .... = Reserve
    .000 0000 0000 0000 0000 0000 0000 0111 = Stream
    [Pad Length: 0]
    Header Block Fragment: 8386c4c3c2c1c0bfbe
    [Header Length: 256]
    [Header Count: 9]
  ▶ Header: :method: POST
```

```
▶ Stream: HEADERS, Stream ID: 5, Length 9, POST /grpc.simple.UserService/GetUserInfo
▶ Stream: HEADERS, Stream ID: 7, Length 9, POST /grpc.simple.UserService/GetUserInfo
▶ Stream: HEADERS, Stream ID: 9, Length 9, POST /grpc.simple.UserService/GetUserInfo
▶ Stream: HEADERS, Stream ID: 11, Length 9, POST /grpc.simple.UserService/GetUserInfo
▶ Stream: HEADERS, Stream ID: 13, Length 9, POST /grpc.simple.UserService/GetUserInfo
▶ Stream: DATA, Stream ID: 11, Length 5
▶ Stream: DATA, Stream ID: 3, Length 7
▶ Stream: DATA, Stream ID: 13, Length 7
▼ Stream: DATA, Stream ID: 7, Length 7
    Length: 7
    Type: DATA (0)
  ▼ Flags: 0x01
      .... ...1 = End Stream: True
      .... 0... = Padded: False
      0000 .00. = Unused: 0x00
    0... .... .... .... .... .... .... .... = Reserved: 0x0
    .000 0000 0000 0000 0000 0000 0000 0111 = Stream Identifier: 7
    [Pad Length: 0]
    Data: 00000000020802
▶ Stream: DATA, Stream ID: 1, Length 7
▶ Stream: DATA, Stream ID: 9, Length 7
▶ Stream: DATA, Stream ID: 5, Length 7
```

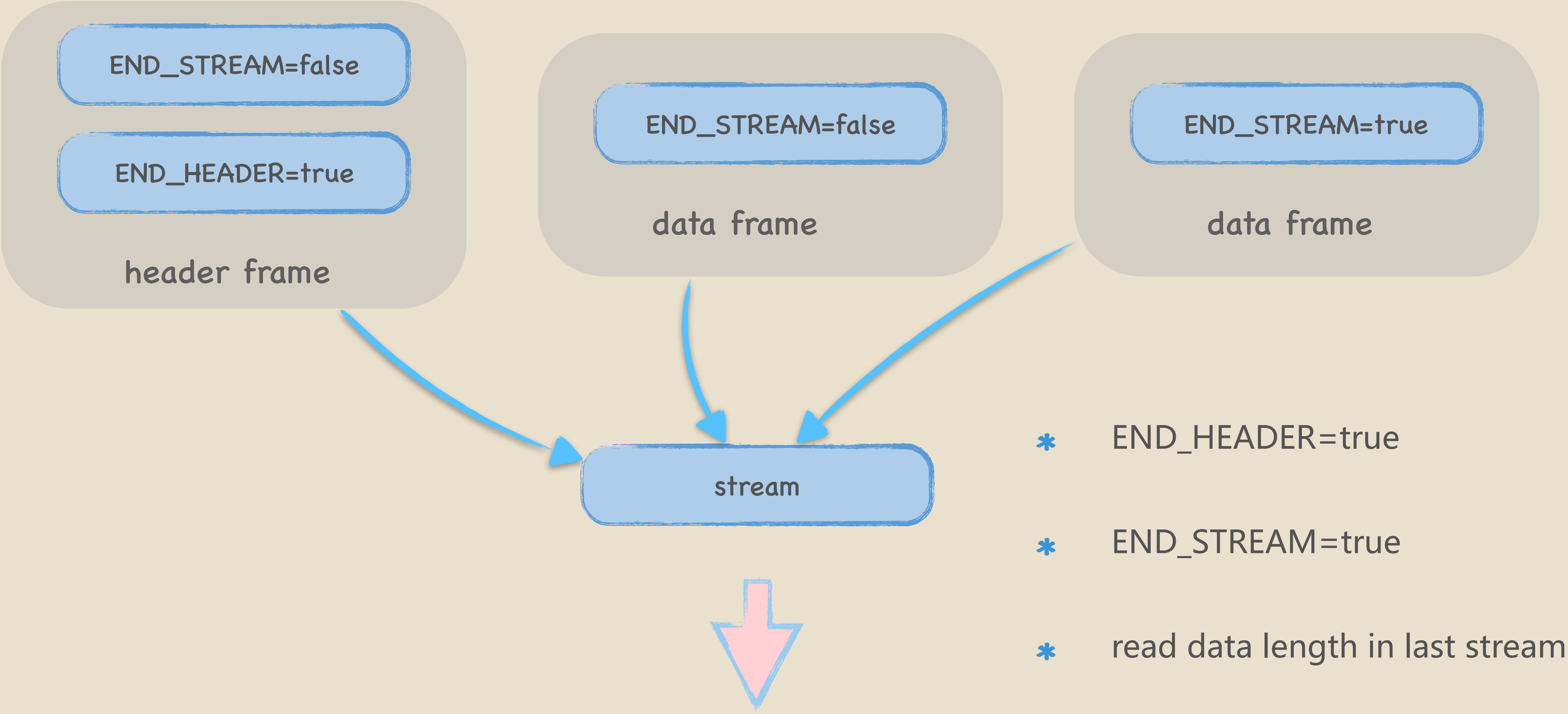
- END_STREAM
- END_HEADERS
- PRIORITY
- …



http 2.0 连接过程

- 客户端用http1.1发起升级协议请求
  - Upgrade: h2c
  - HTTP2-Settings: <base64url encoding of h2 SETTINGS payload>
- 服务端使用http1.1返回 101 Switching Protocols
- 服务端使用http2.0发送SETTINGS frame连接序言(preface)
- 客户端也必须回应一个包含SETTINGS帧的连接序言

协议解析



- END_HEADER=true
- END_STREAM=true
- read data length in last stream

how to test

- wireshark
- nghttp2
  - nghttp -nv https://nghttp2.org
- http2 vs http1.1 效果
  - https://http2.golang.org/gophertiles
  - https://http2.akamai.com/demo

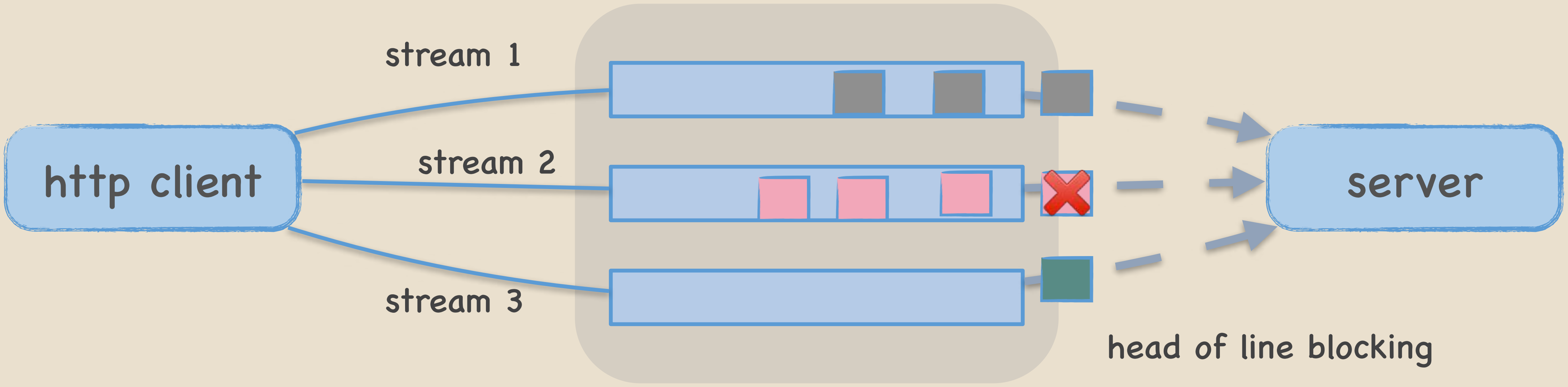
http2的缺点？

- TCP三次握手
  - Tcp fast open ?
    - > linux kernel 3.6
- TLS的交互
  - Tls 1.3 = 0 RTT
- TCP慢启动
- TCP队首阻塞

- TCP 拥塞处理
  - 慢启动
  - 拥塞避免
  - 拥塞发生
    - 快重传
    - 慢启动
  - 快速恢复

这尼玛是TCP的缺点吧 ...

tcp hol Blocking



当某个tcp packet丢包, 触发重传定时器, 继而触发 “拥塞发生”

其拥塞窗口降为1, 对丢包进行重传, 未收到ack之前其他包阻塞！

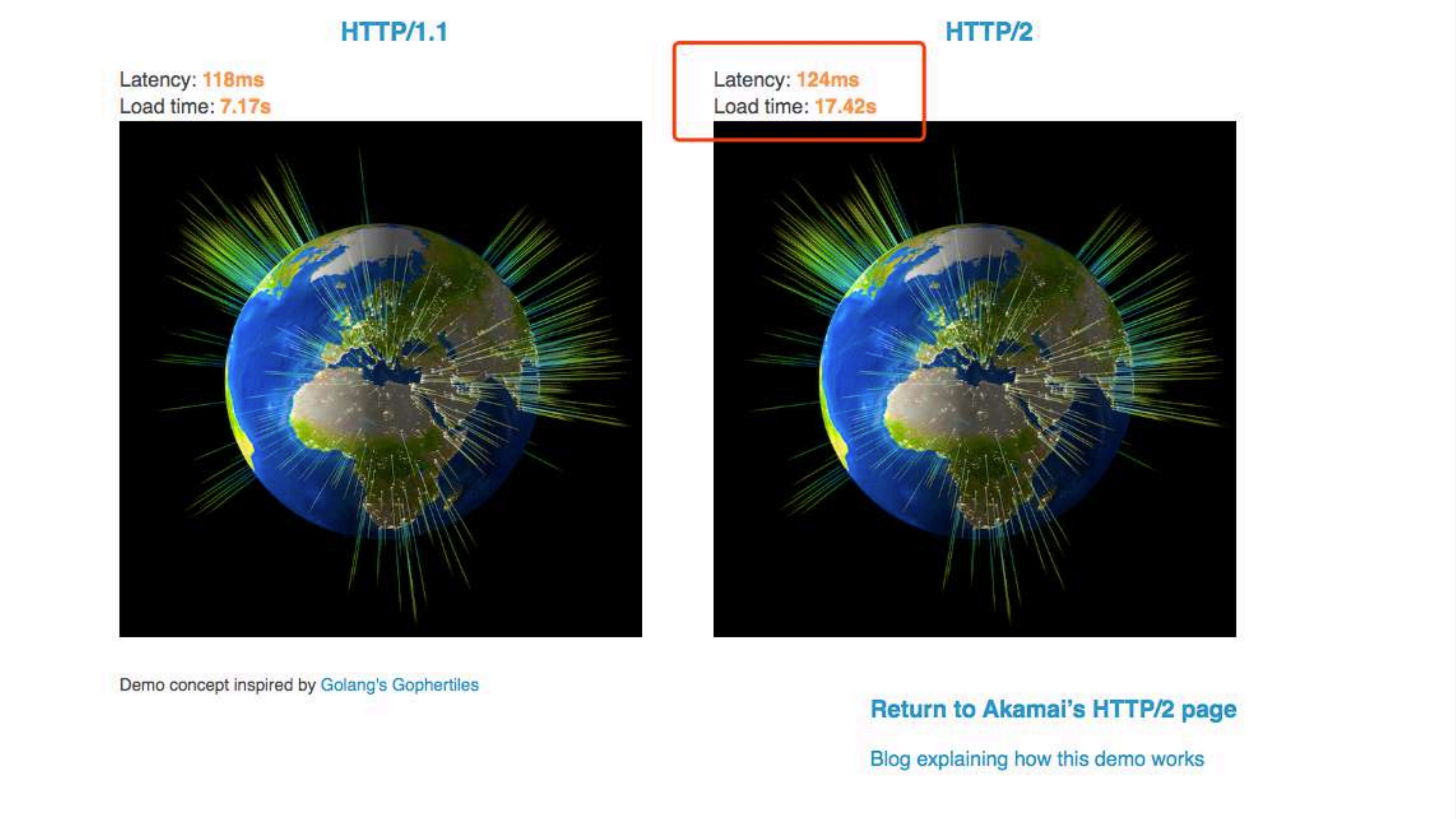
how to do ?

- http2 多连接
- quic

Akamai CDN report

在频繁丢包的网络环境下, http2比http1的多连接更低效.

http2 弱网络测试



HTTP/1.1

Latency: 118ms
Load time: 7.17s

HTTP/2

Latency: 124ms
Load time: 17.42s

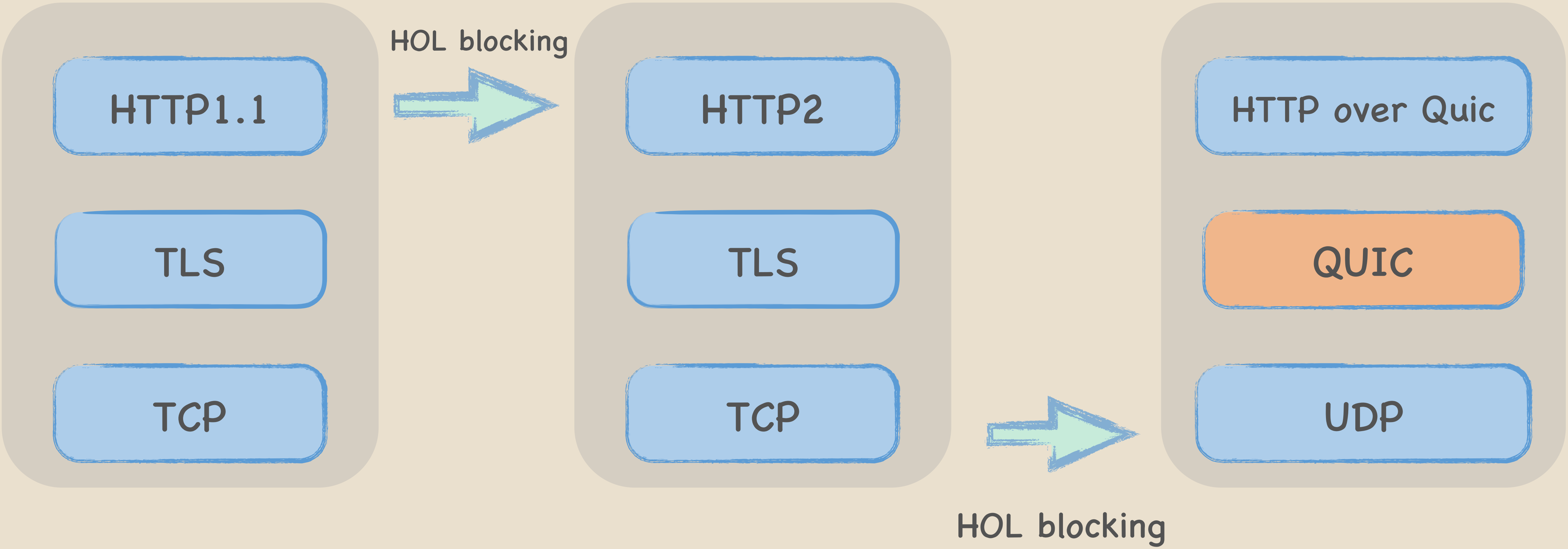
Demo concept inspired by Golang's Gophertiles

Return to Akamai's HTTP/2 page

Blog explaining how this demo works

- Charles
- Fiddler

Quic



HTTP1.1

TLS

TCP

HOL blocking

HTTP2

TLS

TCP

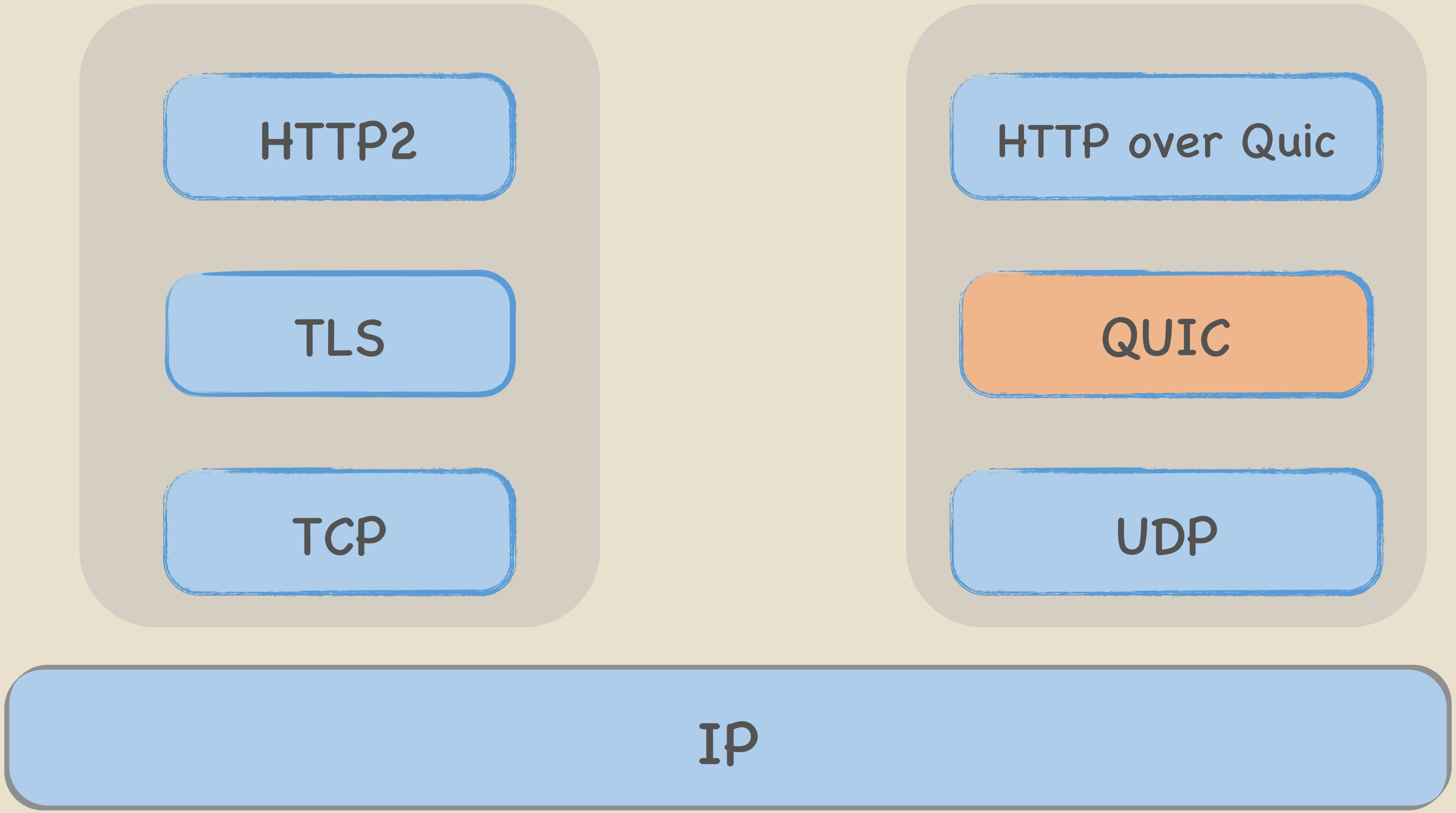
HOL blocking

HTTP over Quic

QUIC

UDP

Quic



quic over udp

测试点	TCP	UDP	Quic
可靠性	可靠	不可靠	可靠
连接性	面向连接	无连接	无连接
流量控制	有	无	有
拥塞控制	有	无	有
效率	低	高	高

optimize hol blocking ?

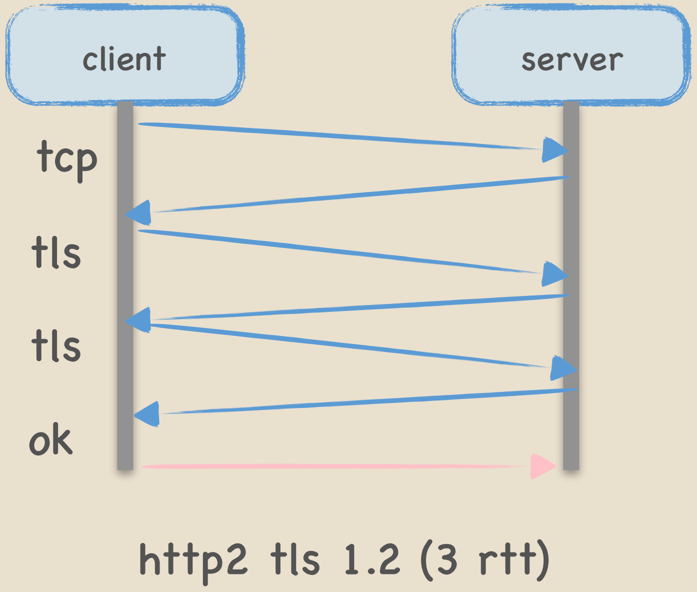
- UDP
- UDP
- UDP

- 面向数据报文
- 数据包之间没有阻塞约束
- 丢包只会影响对应的stream
- 前向纠错减少了重传的可能

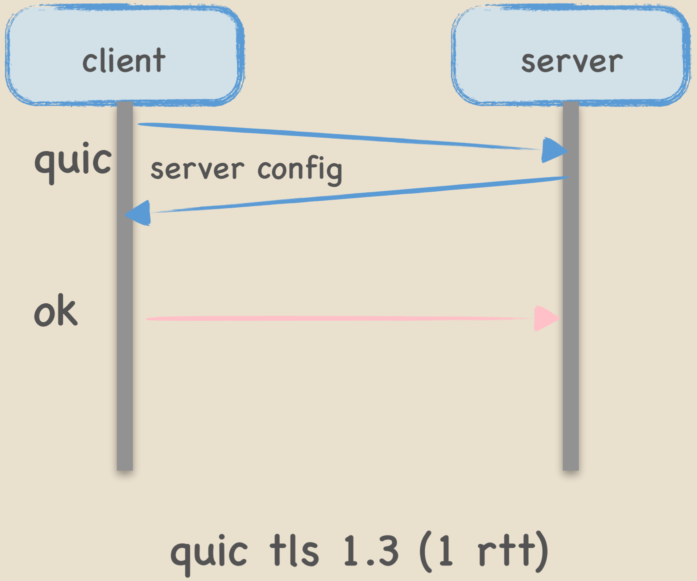
quic 0 Rtt

- 0 RTT
  - 使用UDP规避三次握手
  - 使用DH加密来规避TLS交互
  - 首包就可发送数据

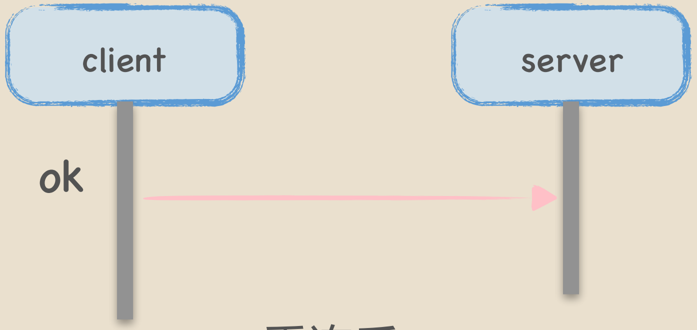
一次Server Config的缓存,
后面只要缓存不失效, 重连无需TLS交互！



http2 tls 1.2 (3 rtt)



quic tls 1.3 (1 rtt)



quic重连后 0 rtt !!!

quic 可靠性

- 手段
  - 单调递增seq
  - 前向冗余纠错 (异或)
  - 失败重传

- 当发生丢包
  - 尝试使用前向纠错来修复
  - 不能fec, 则失败重传

发送数据A和B, 增加发送一个数据C等于A和B的异或。

接收方接到这3个包的任意2个包，异或一下就可以得到第3个包。

quic 拥塞窗口

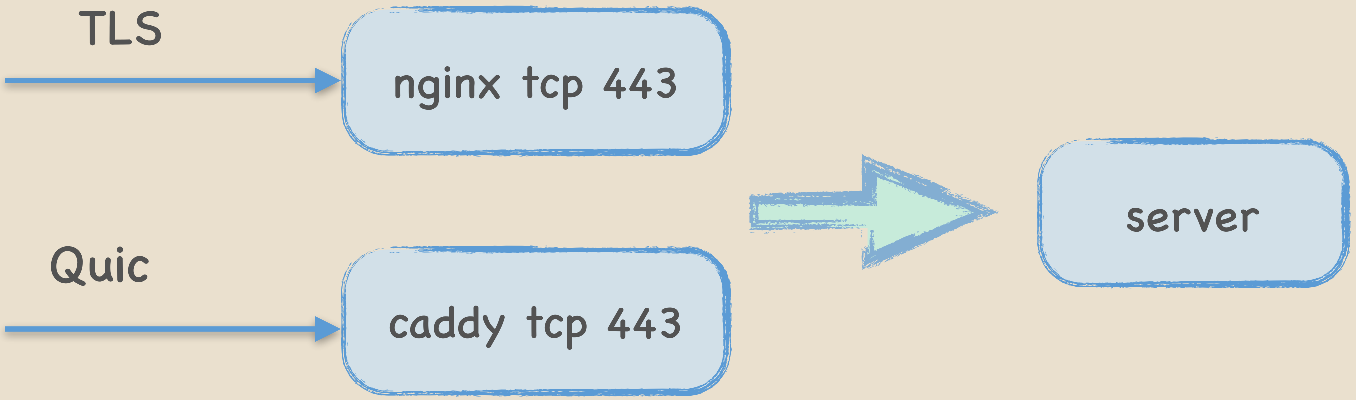
- quic 拥塞窗口
  - 实现了tcp的Cubic和NewReno算法
  - 默认采用Cubic
  - nack机制，由接收端告知哪几个包丢失
  - Tail Loss Probes更及时的重传

为毛还需要拥塞窗口算法呀？

网络质量的探测, 避免流量浪费

quic 支持状况

- 运营商
    - udp 降级
    - udp qos
- chrome支持
- nginx暂不支持
- caddy proxy

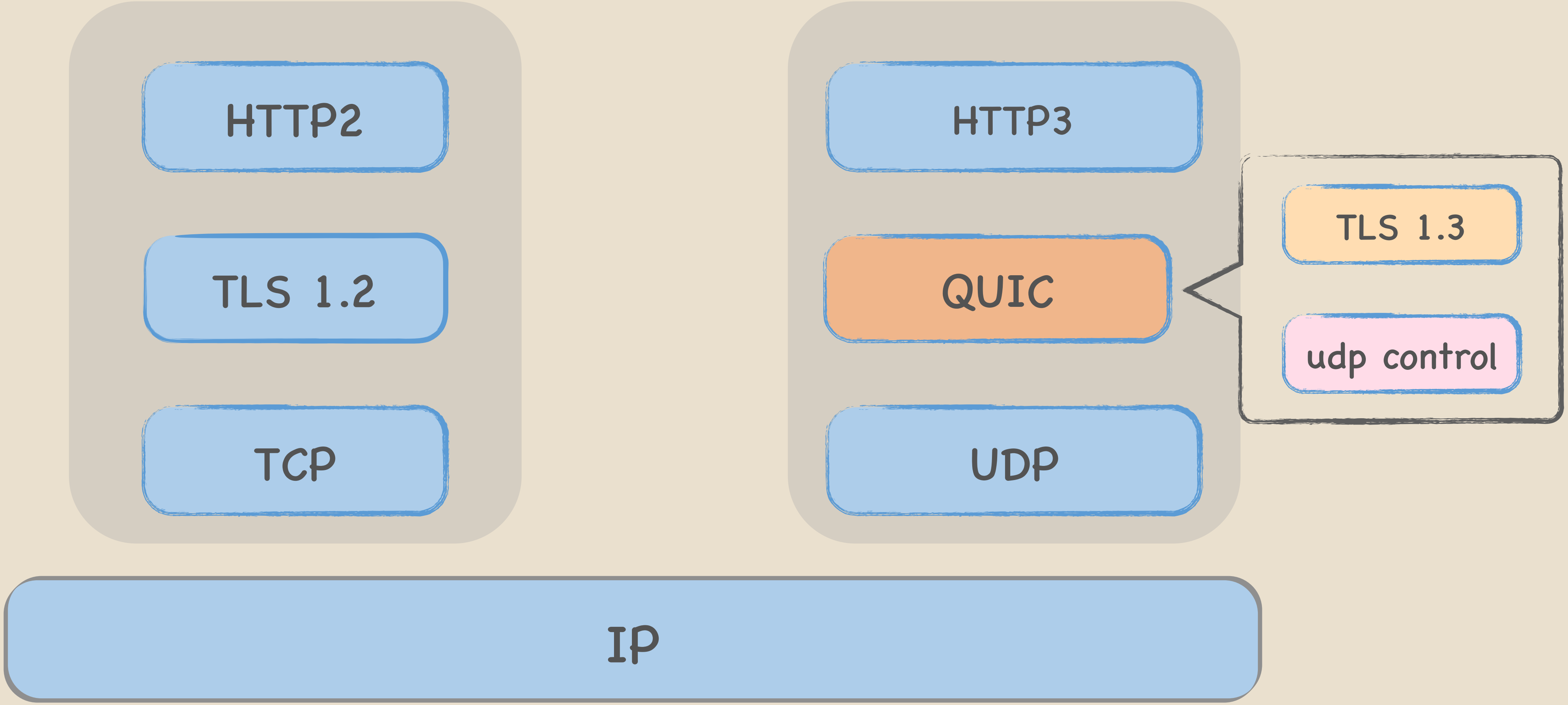


- nginx返回支持quic
    - add_header alt-svc 'quic=":443"; ma=2592000; v= “39"';
- caddy作为quic代理

quic 其他

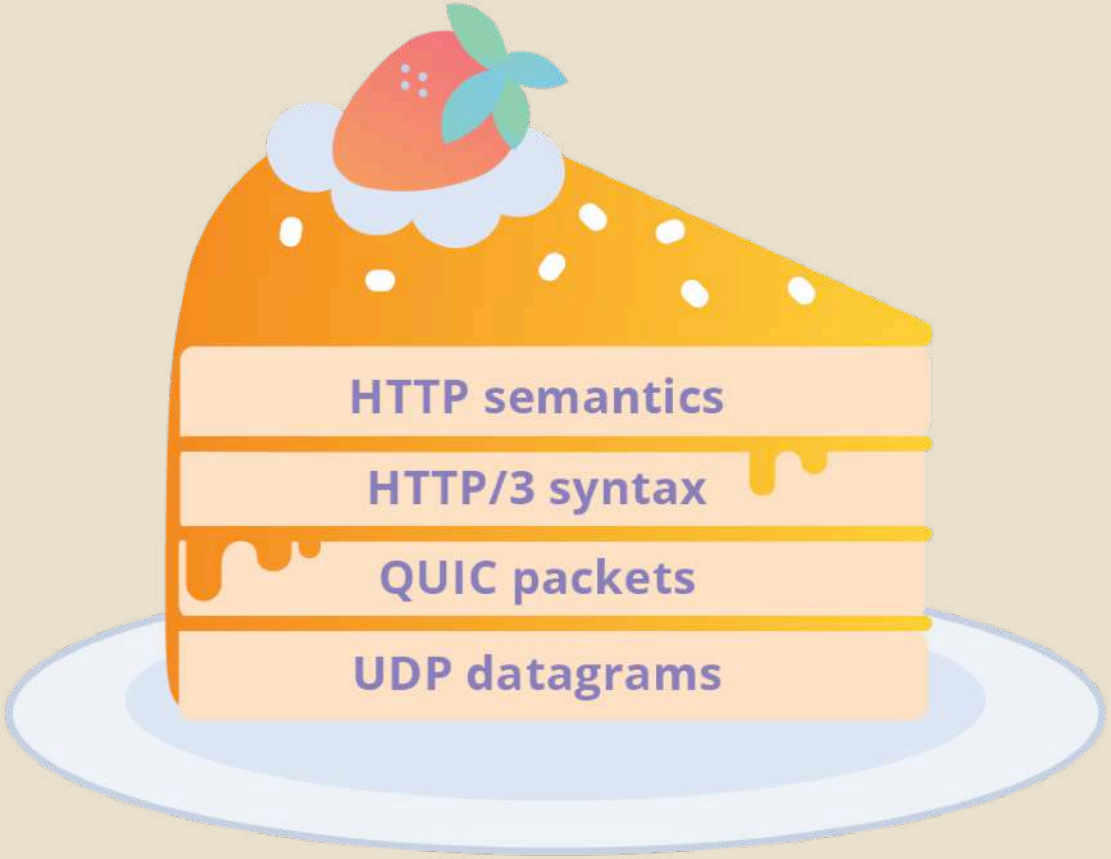
- quic 连接性
  - 使用connection id来识别重新连接的请求
- quic 流量控制
  - 可以针对连接控制，也可以对具体的stream id进行控制

万众期待的http3



Google

- google spdy
  - http2
- google quic
  - http3



Q & A

– xiaorui.cc