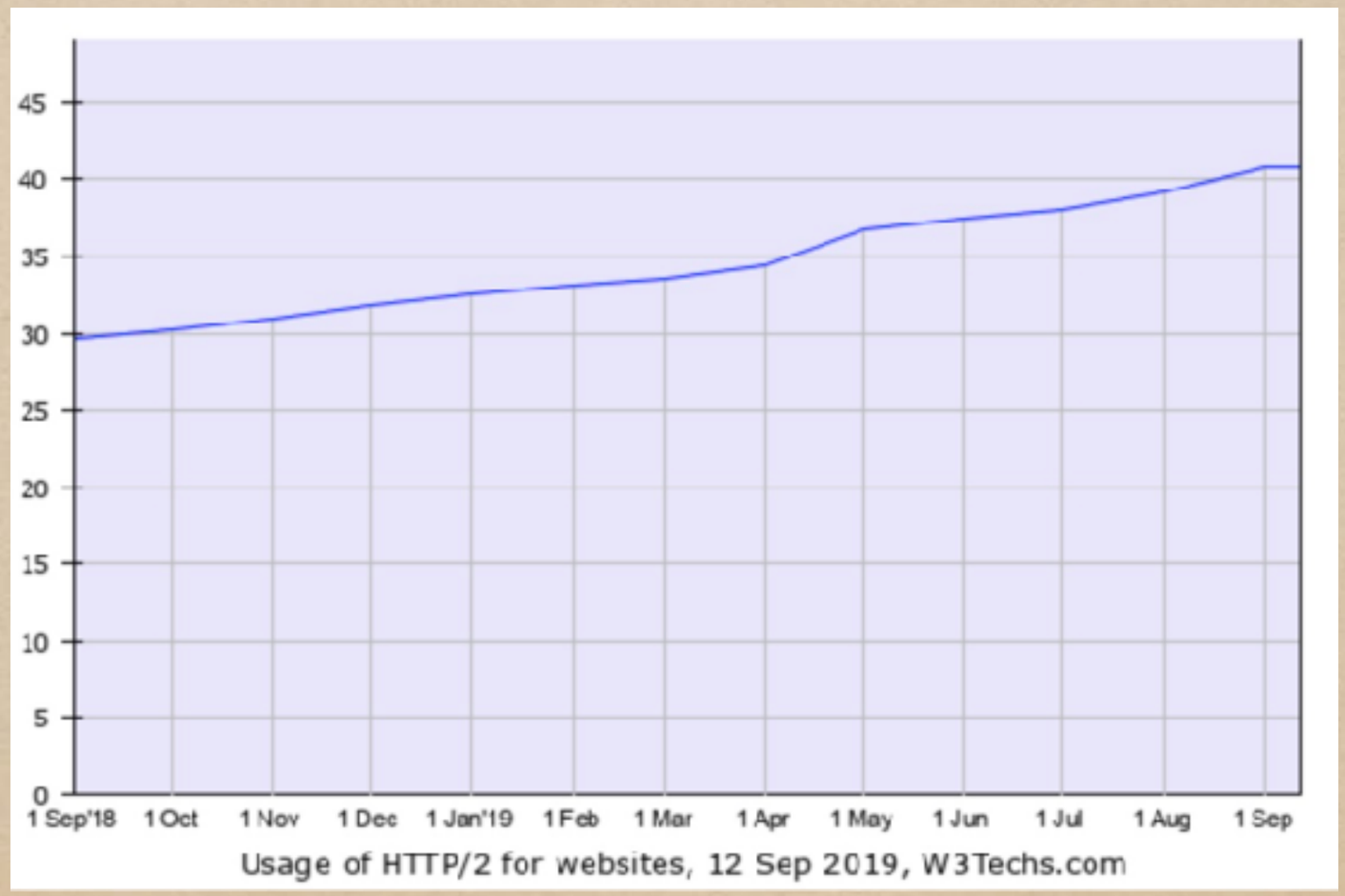
从HTTP1到QUIC

大纲

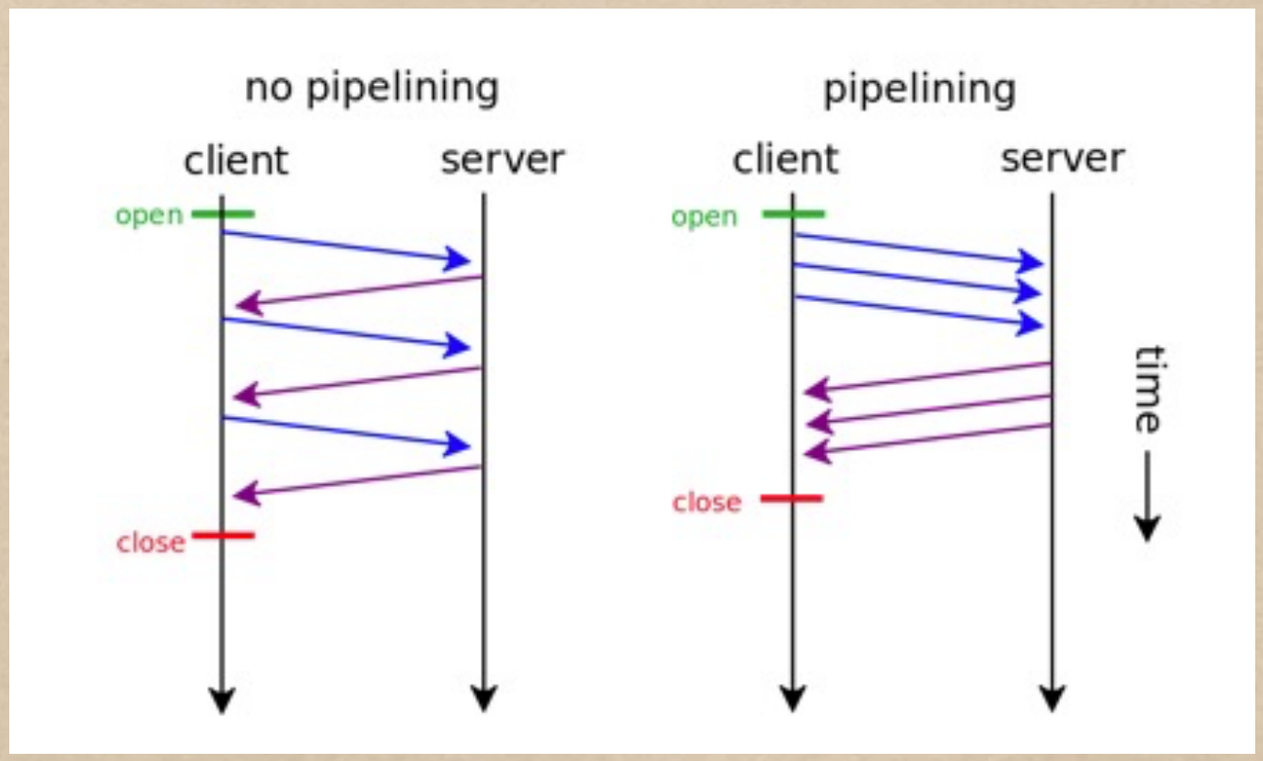
- 从HTTP1到HTTP2
- 从HTTP2到QUIC (HTTP3)

HTTP2 的使用情况

数据来源：w3techs.com



HTTP1-Head-of-Line Blocking



HTTP1-Head-of-Line Blocking

- 整个连接还是先进先出的
- 未完全解决head of line blocking
- 多数http proxy不支持
- 多数浏览器默认关闭 pipeline

HTTP1-header data redundant

数据来源：http2.github.io

> If you assume that a page has about 80 assets (which is conservative in today's Web), and each request has 1400 bytes of headers (again, not uncommon, thanks to Cookies, Referer, etc.)

HTTP1-header data redundant

curl -v http://hao.qq.com/ | head -n 10

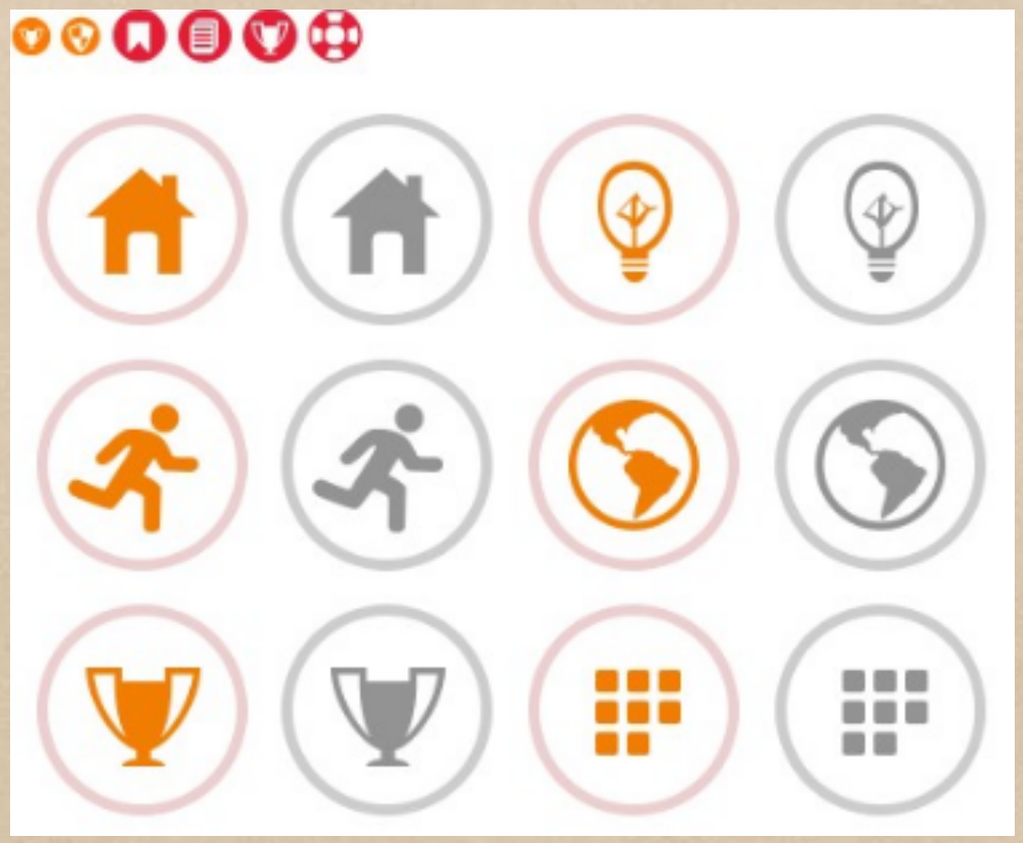
```
> GET / HTTP/1.1
> Host: hao.qq.com
> User-Agent: curl/7.46.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 12 Sep 2019 08:15:20 GMT
< Content-Type: text/html; charset=UTF-8
< Transfer-Encoding: chunked
< Connection: keep-alive
< Server: nginx
< Set-Cookie: IPLOC=CN1100; path=/
< P3P: CP="CURa ADMa DEVa PSAo PSDo OUR BUS UNI PUR INT DEM STA PRE COM NAV OTC NOI DSP COR"
< Cache-Control: no-cache
< Expires: Thu, 01 Jan 1970 00:00:00 GMT
< P3P: CP="CURa ADMa DEVa PSAo PSDo OUR BUS UNI PUR INT DEM STA PRE COM NAV OTC NOI DSP COR"
< Content-Encoding: gzip
<
{ [890 bytes data]
```

HTTP1优化

- 延时
- 吞吐

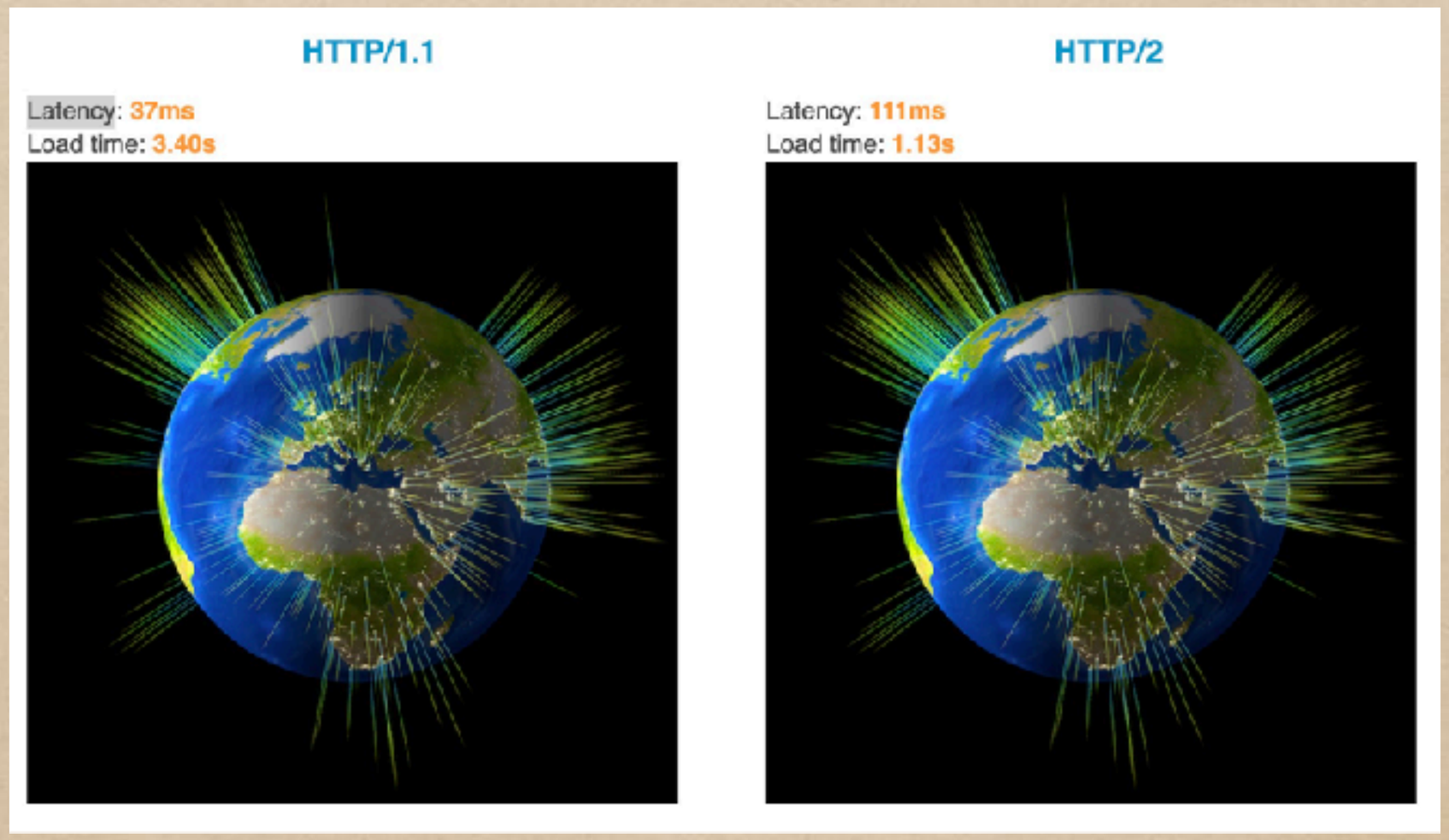
方法	目的
资源合并	减少请求数
多域名拆分	增大连接数
cookie free domain	减少header大小
精灵图	减少请求数
压缩数据	减少无用的字符

HTTP1优化-精灵图示例



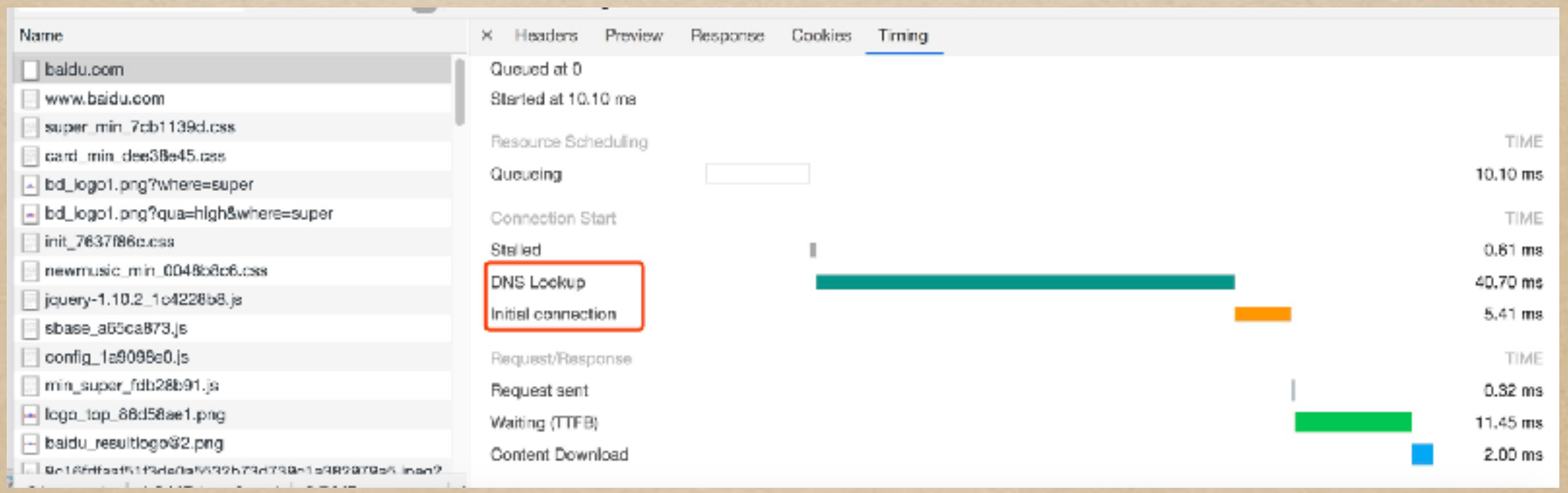
Why HTTP2

https://http2.akamai.com/demo. **378 requests**



Create Connection Cost

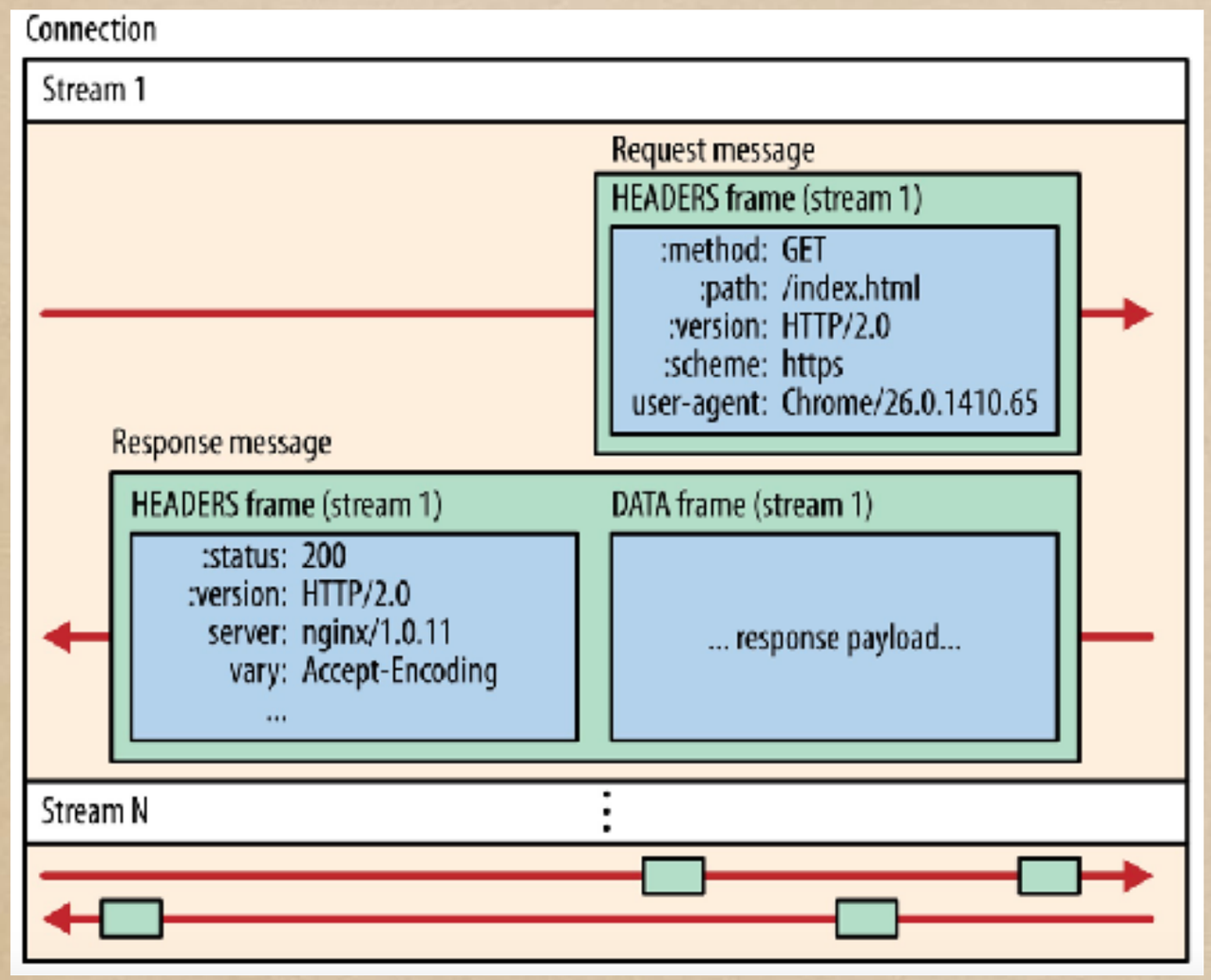
HTTP2能够较好的应对突发流量



HTTP2-Feature

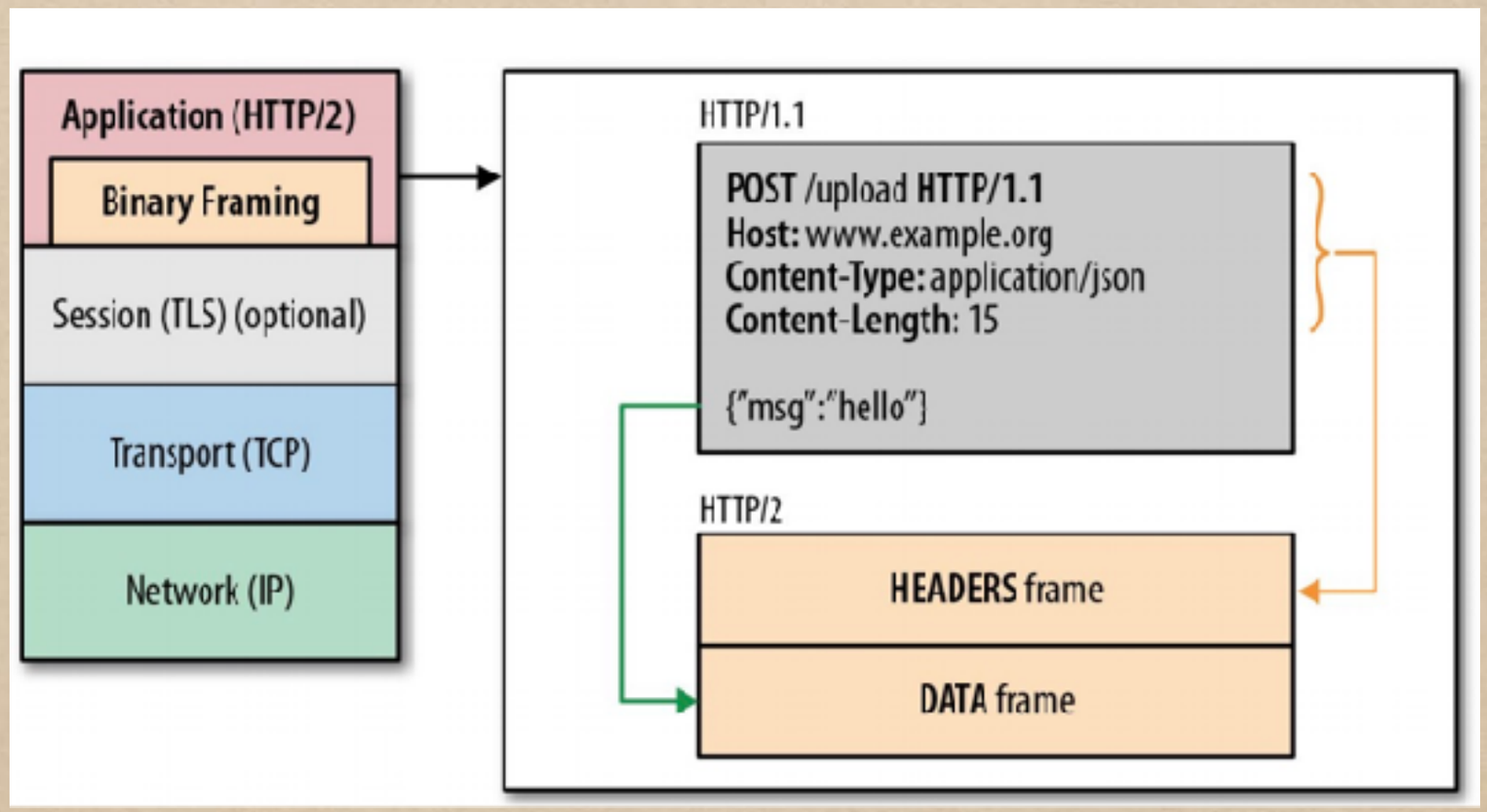
- 多路复用
- Header压缩(HPACK)
- 流控(stream)
- 优先级(stream)
- 服务端推送

HTTP2-Multiplexing



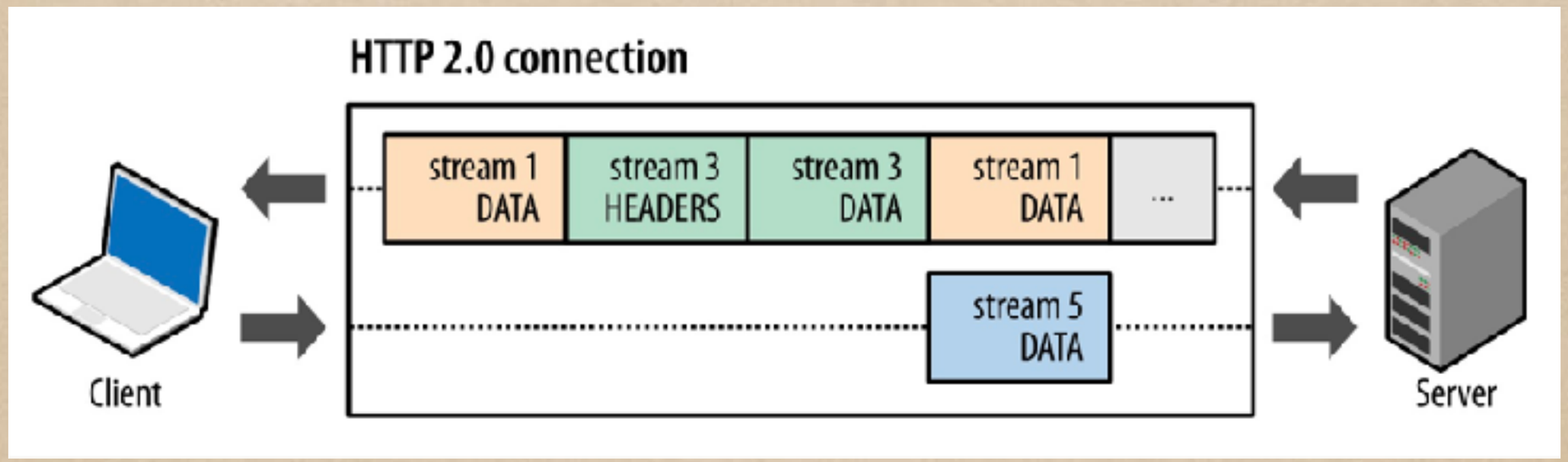
- connection
- stream
- message
- frame

HTTP2-Multiplexing

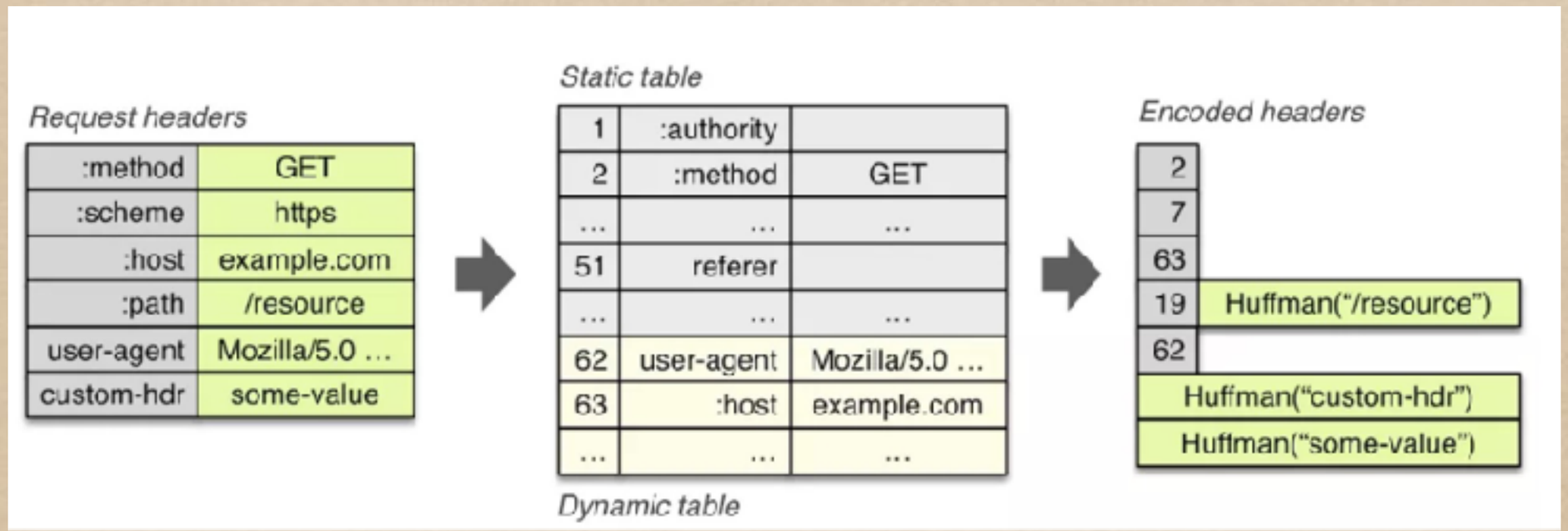


h2/h2c

HTTP2-Multiplexing

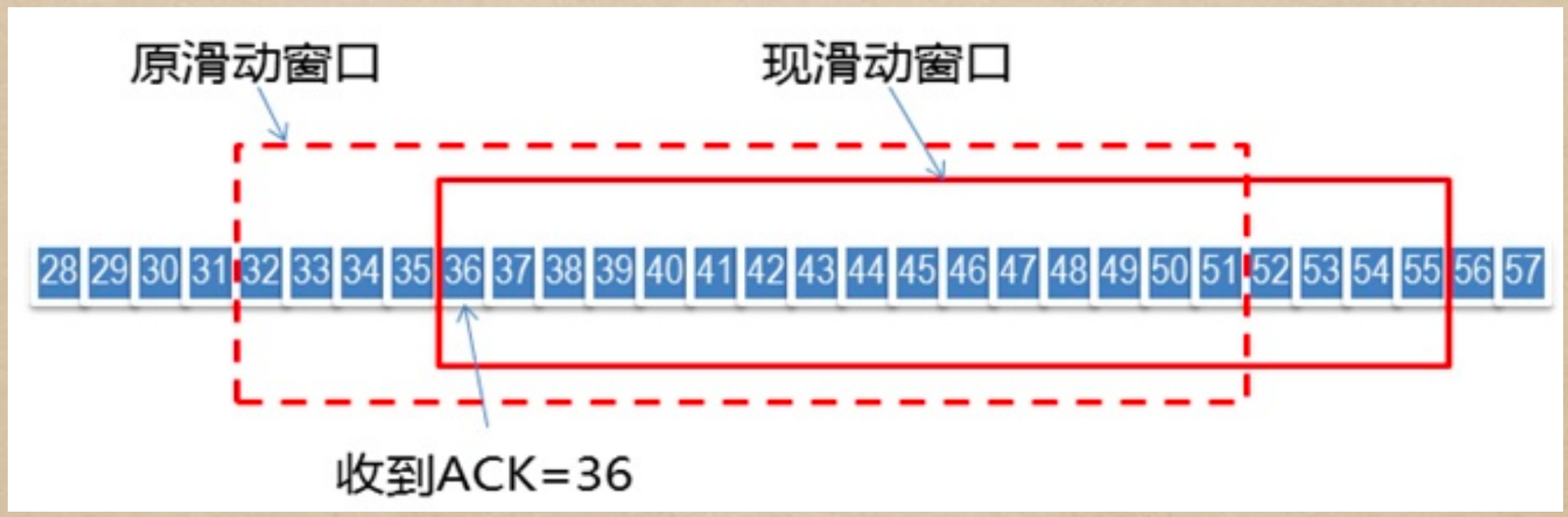


HTTP2-header compress

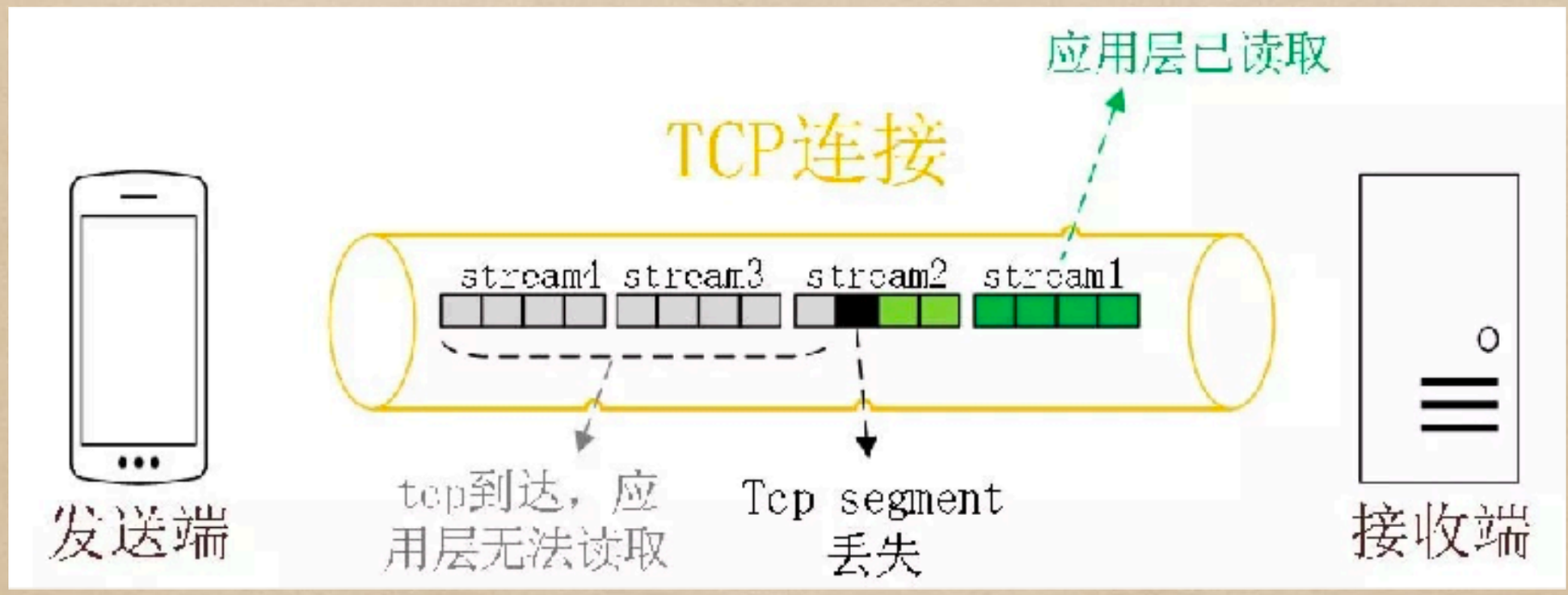


- cookie拆分多个键值对
- 通过传递索引号节省空间(通常只占1个字节)
- 使用Huffman进行编码压缩
- 同一个连接的所有steam共享动态表

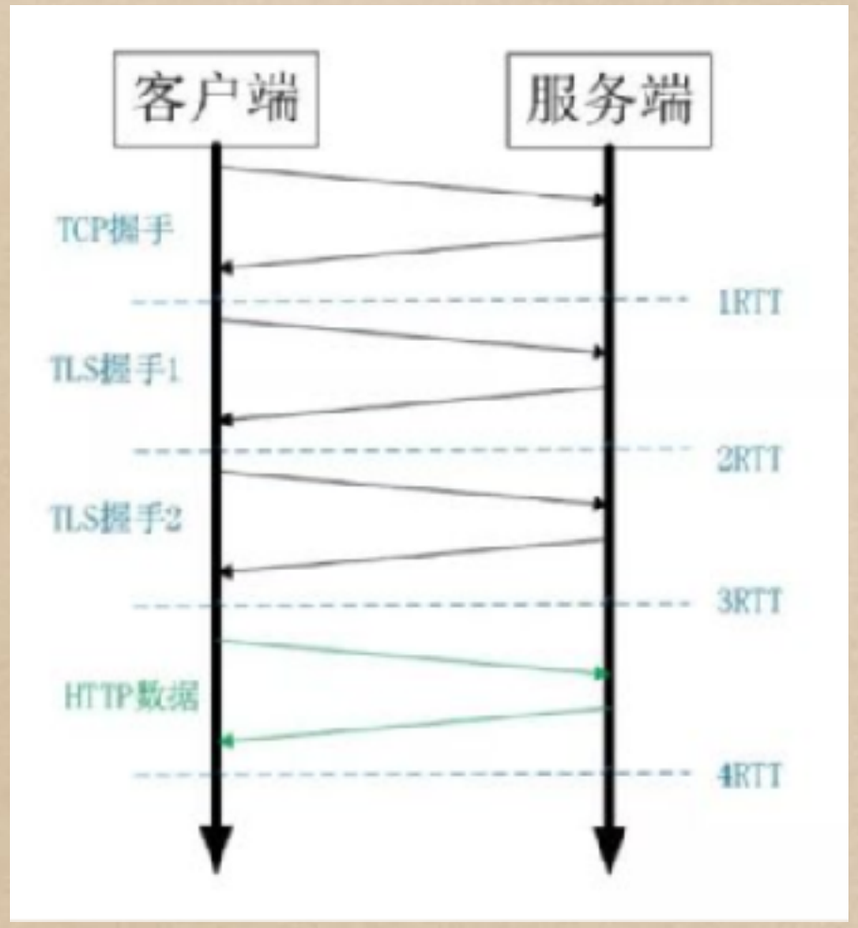
TCP Head-of-Line Blocking



HTTP2 Head-of-Line Blocking



HTTP2-建立连接的握手延迟大

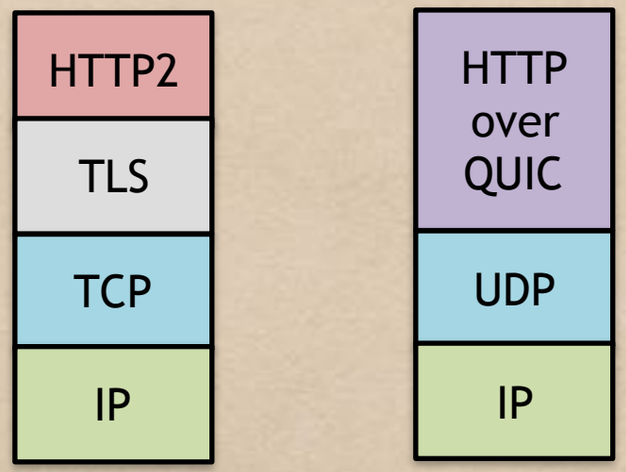


- TCP握手 1RTT
- TLS 握手 2RTT

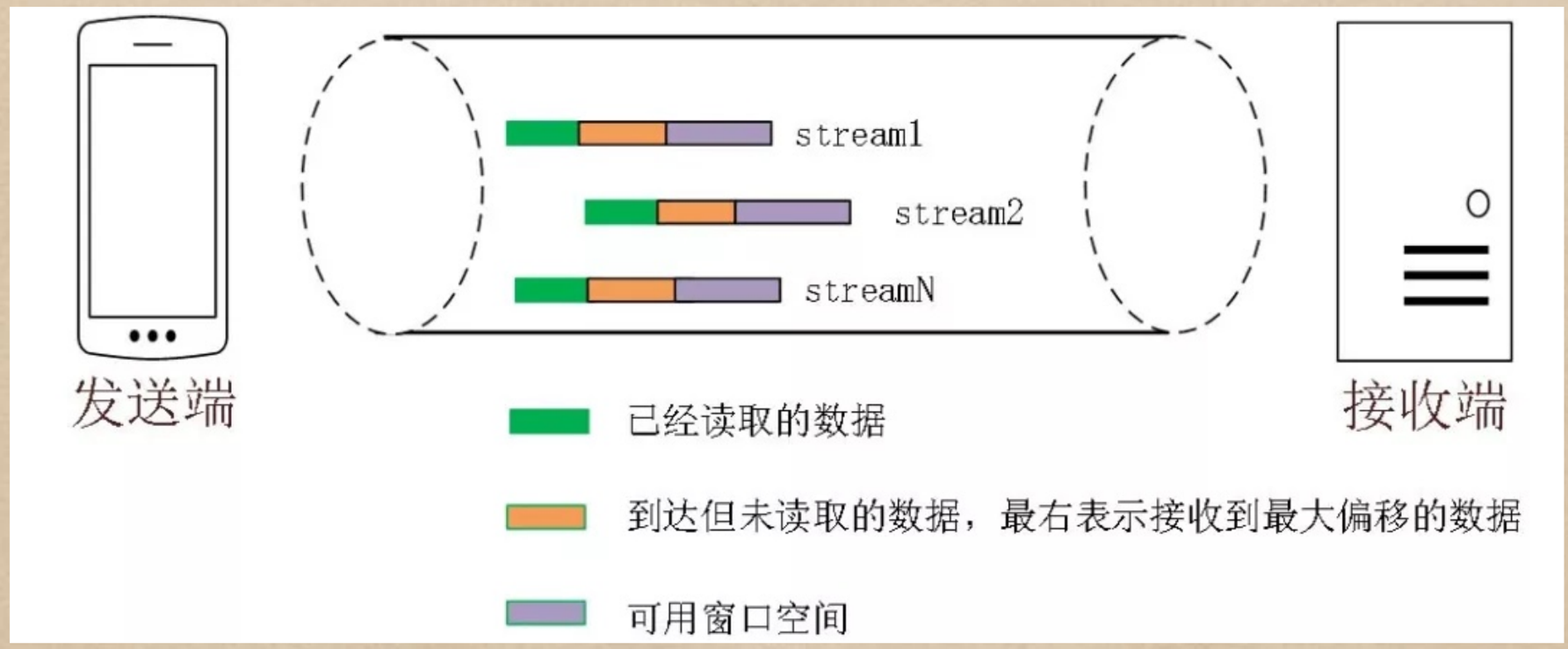
How to do?

- HTTP2多连接
- QUIC

QUIC



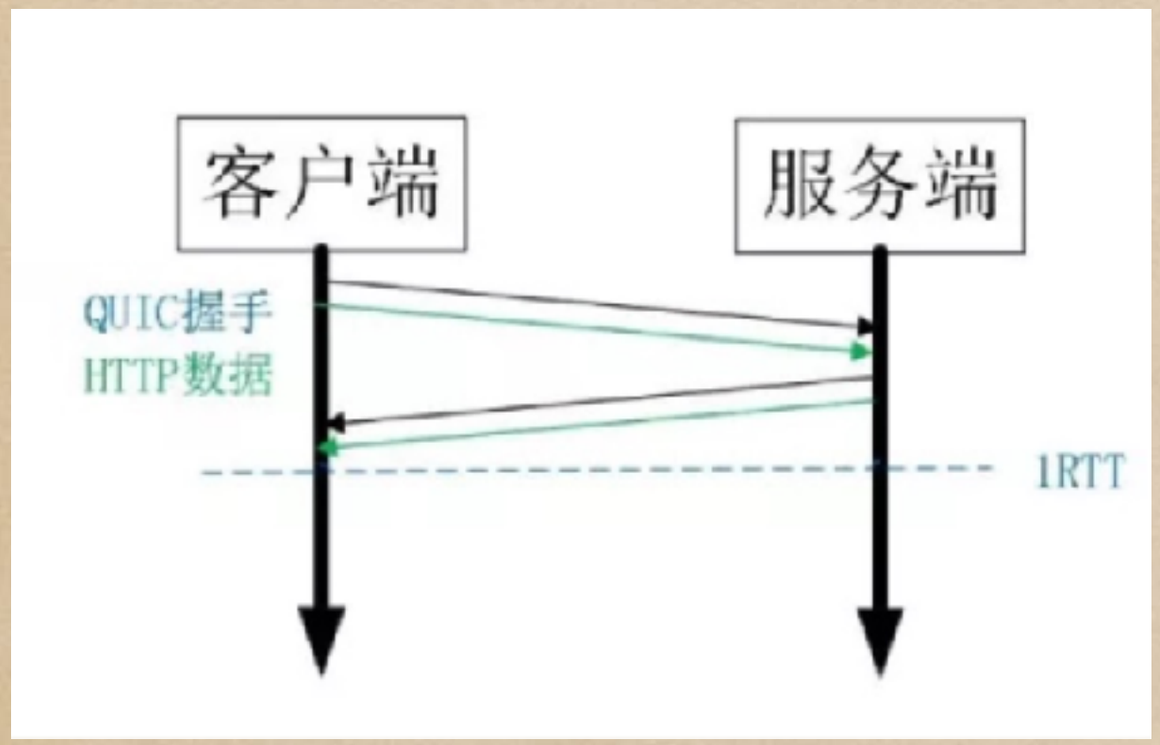
QUIC-解决HOL问题



针对 Stream

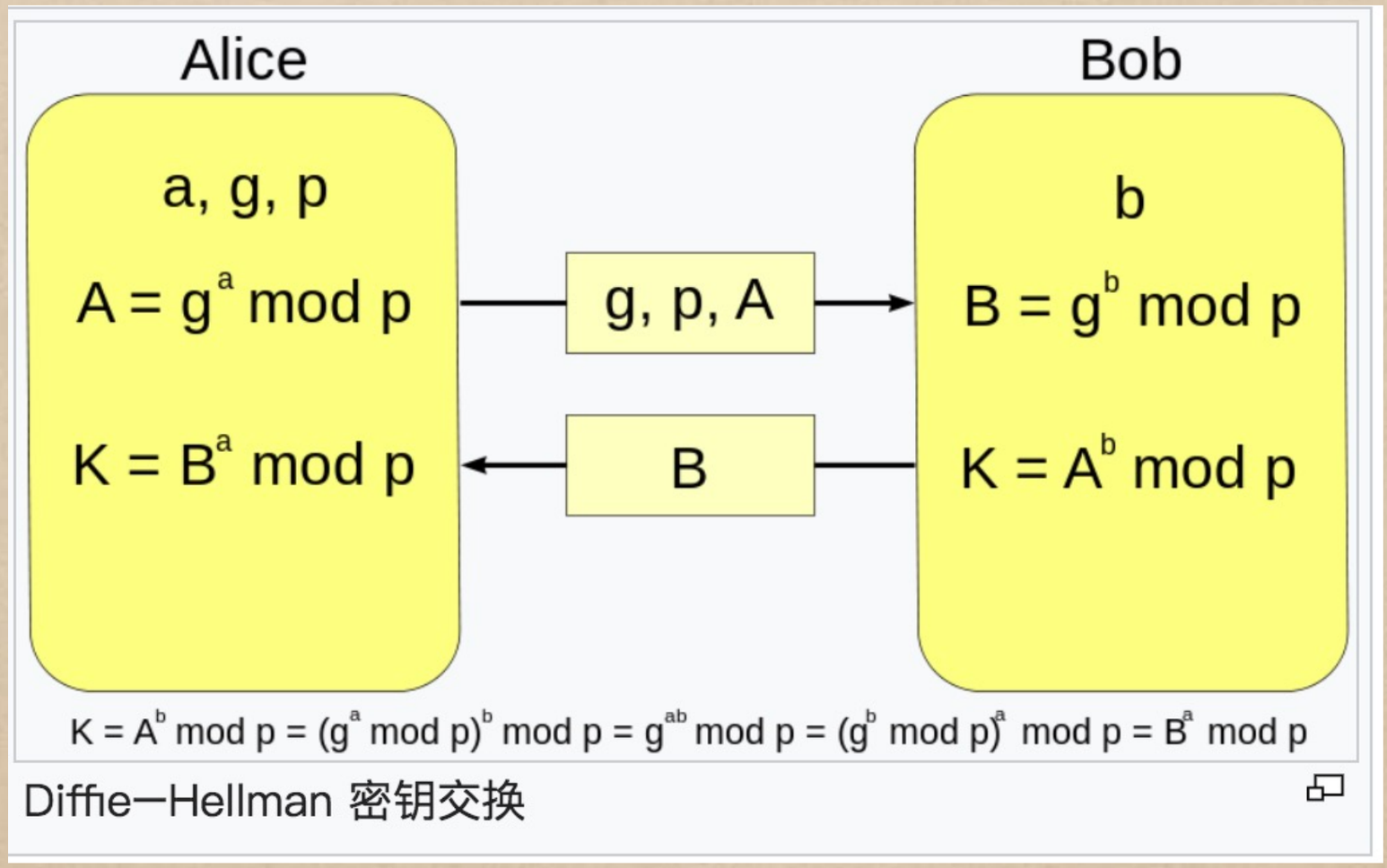
可用窗口 = 最大窗口数 – 接收到的最大偏移数

QUIC-0RTT



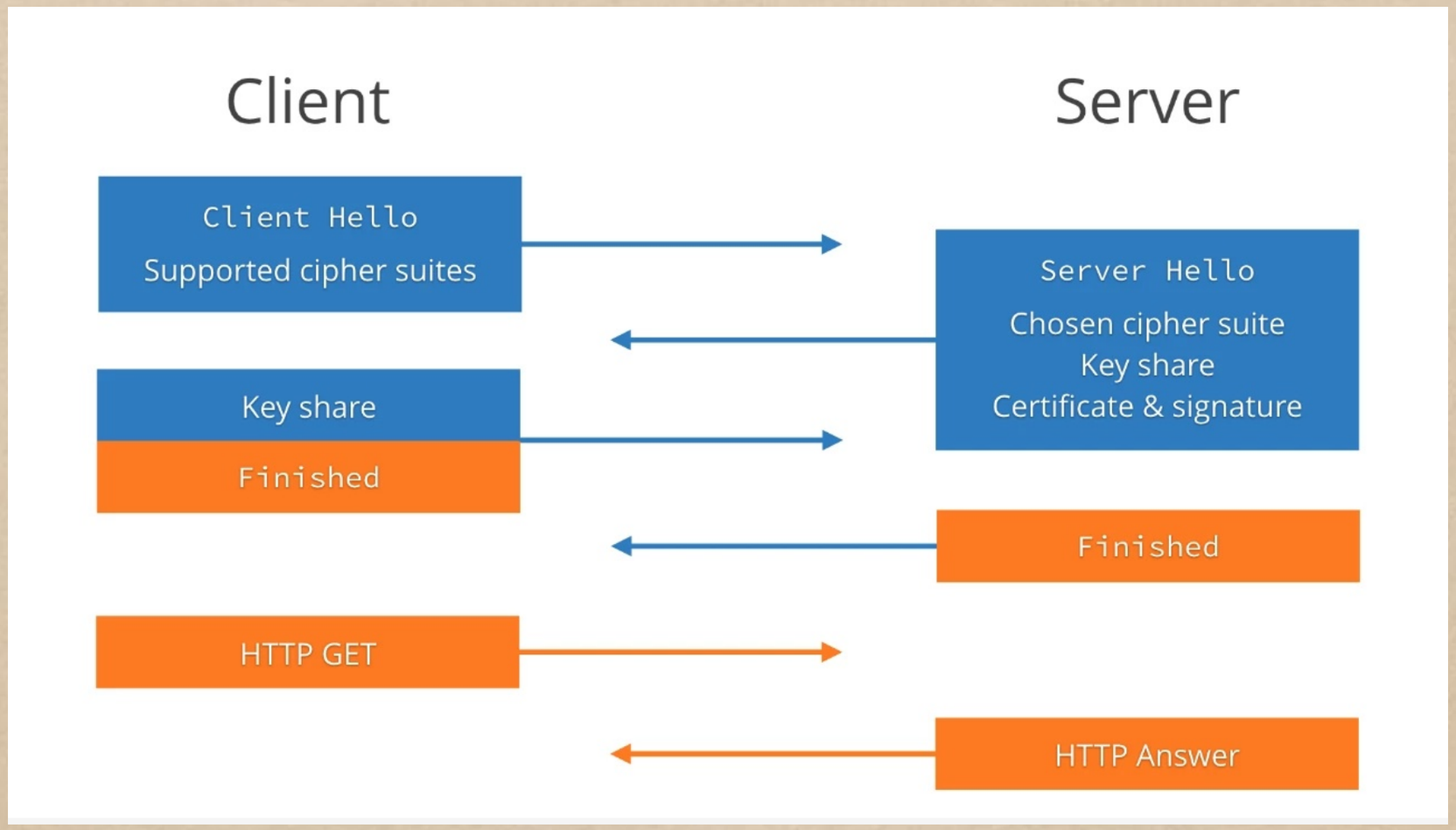
- 首次连接 1RTT
- 重新连接 0RTT
- 连接迁移 0RTT

QUIC-0RTT

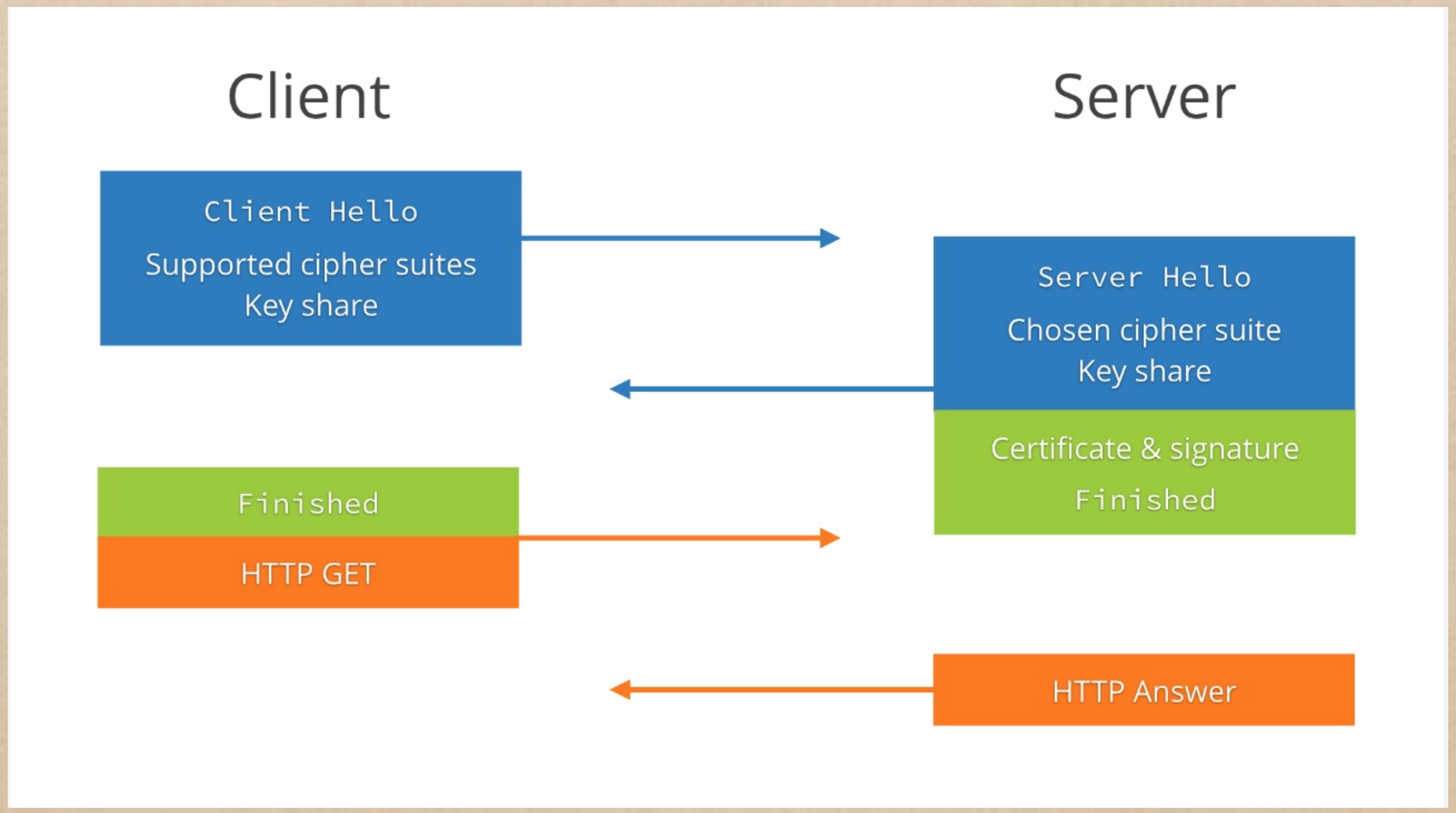


Diffie—Hellman 密钥交换

QUIC-0RTT TLS1.2



QUIC-0RTT TLS1.3



TCP VS UDP VS QUIC

特性	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

QUIC性能参数

成功耗时对比（单位：毫秒）

丢包率 / 协议	0%	2%	4%	6%	8%	10%	20%	30%	40%	50%
QUIC	98.4	60.39	93.04	84.75	73.75	111.32	108.16	288.48	2232.93	5443.79
HTTPS	69.06	126.7	227.18	241.34	303.56	543.68	1293.65	4759.7	5266.67	15873.08

丢包率上升的情况下

文件传输场景耗时对比（单位毫秒）

延迟 / 协议	100ms	500ms
QUIC	170.64	620.63
HTTPS	232.14	1092

健康网络情况下，二者区分并不明显

弱网环境下，QUIC 要优于 HTTPS

HTTP1 -> HTTP2 -> QUIC

Q&A