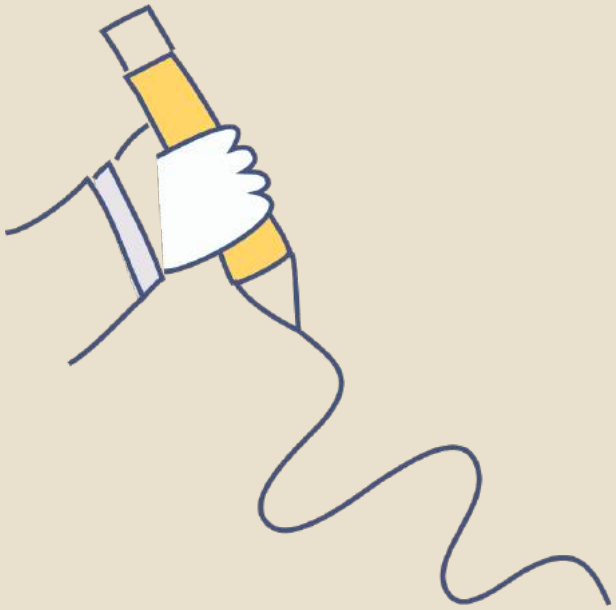




# grpc 的那些事儿

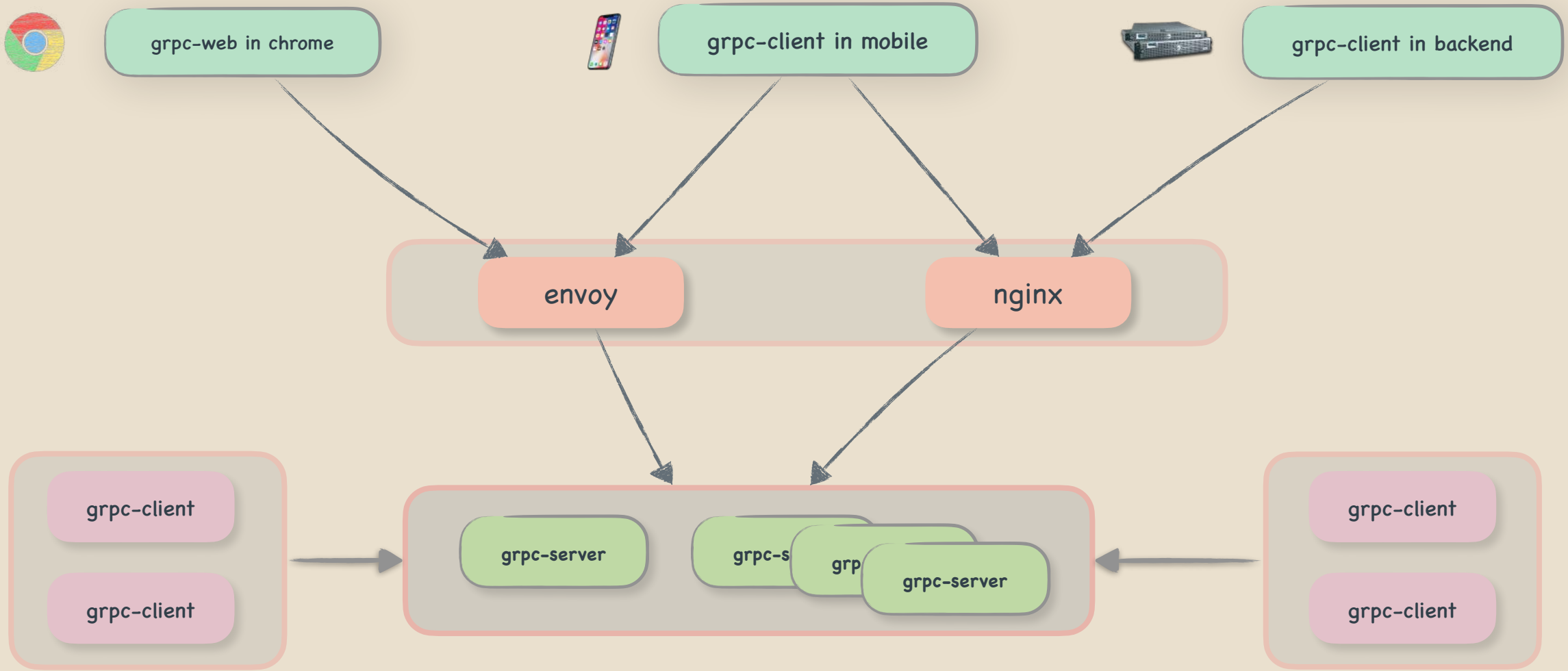
- xiaorui.cc

- github.com/rfyiamcool

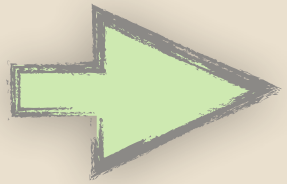



1. http2 design
2. protocol buffer
3. grpc design
4. grpc optimize

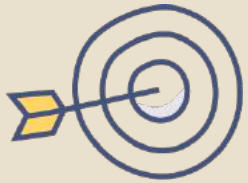
# what is grpc ?


grpc-web in chrome
grpc-client in mobile
grpc-client in backend
envoy
nginx
grpc-client
grpc-client
grpc-server
grpc-server
grpc-client
grpc-client

# what is grpc ?

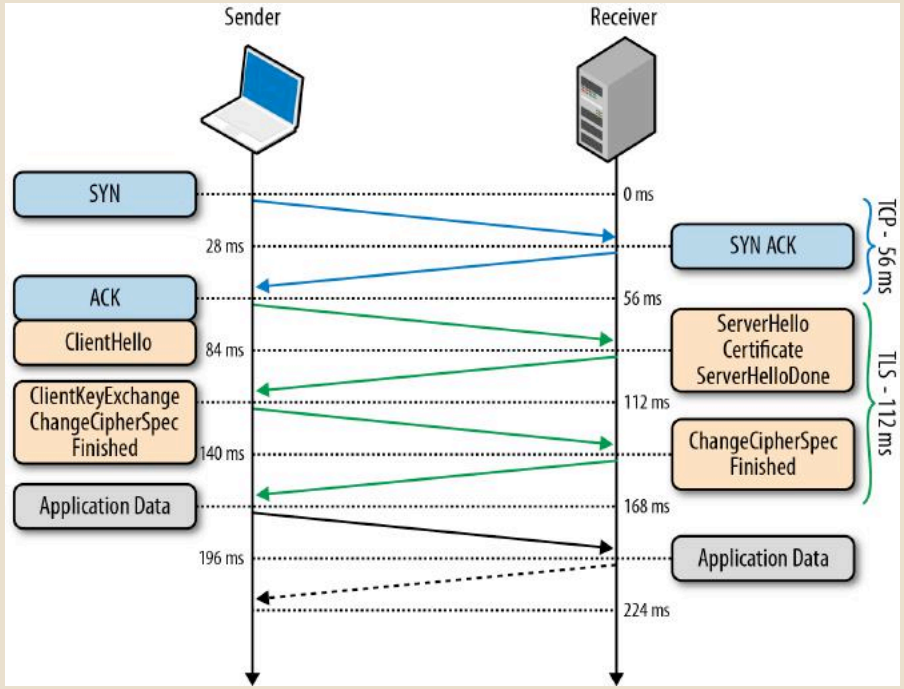



- feature
  - High compatibility
  - high performance
  - verty simple
- rpc framework
  - http2
  - protobuf
  - code generated



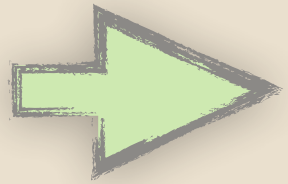

# http 2.0


# what is RTT ?

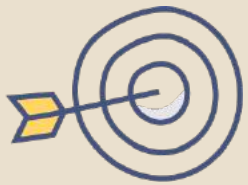



- RTT (Round Trip Time)
  - Round trip time for a packet
  - rtt = recvtime - sendtime
- http request cost N rtt
  - dns
  - http tcp handshake
  - http redirect https
  - https tcp handshake
  - tls handshake
  - ocsp (dns query + tcp handshake + req/resp )

# select http2 ?

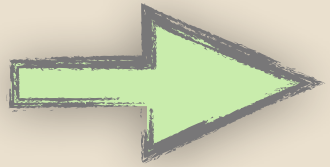



- 最高兼容性 （官方说法）
- 多路复用
- TLS
- 较好的性能 😅
- more

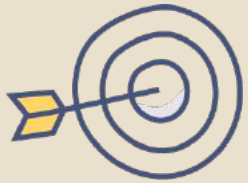



# http1.0 vs http1.1

* http 1.0
  * 古董 ...

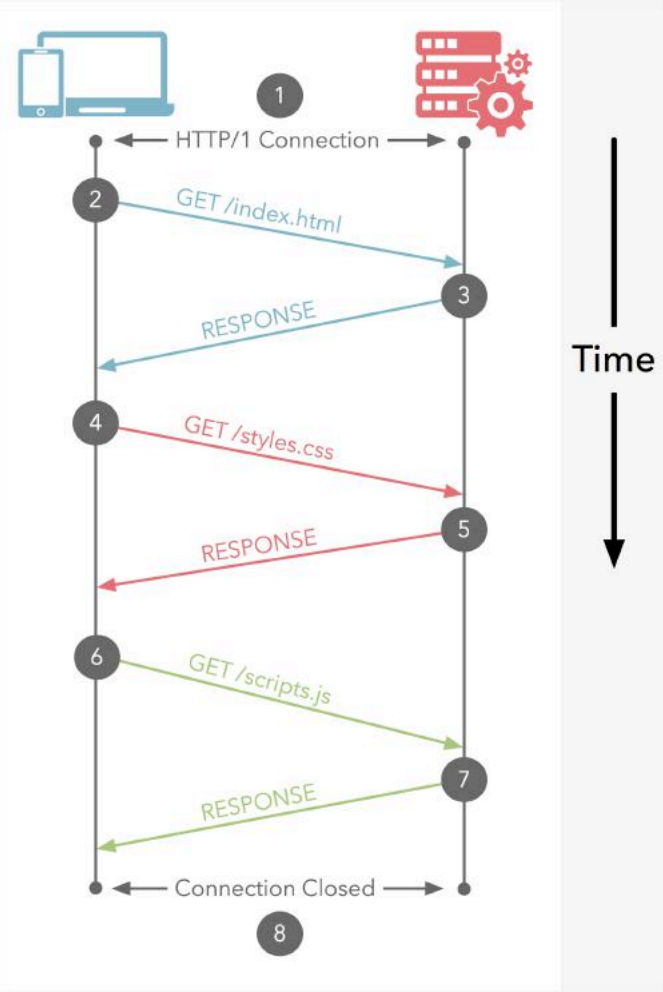


* http 1.1
  * 持久化连接
  * 缓存
  * accept-range
  * more header



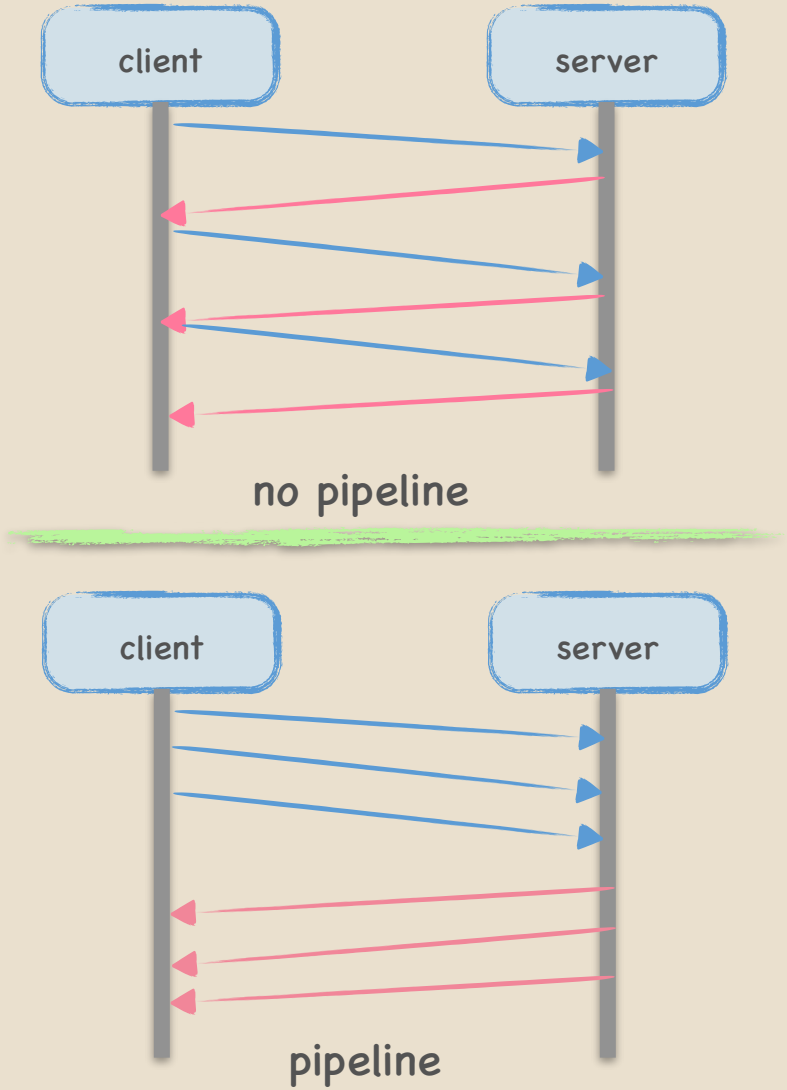
HTTP/2

# 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 pipeline ?

# http 1.1 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

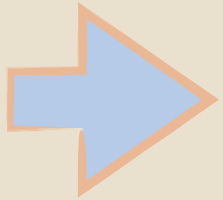
Google Report

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


# 优化时延

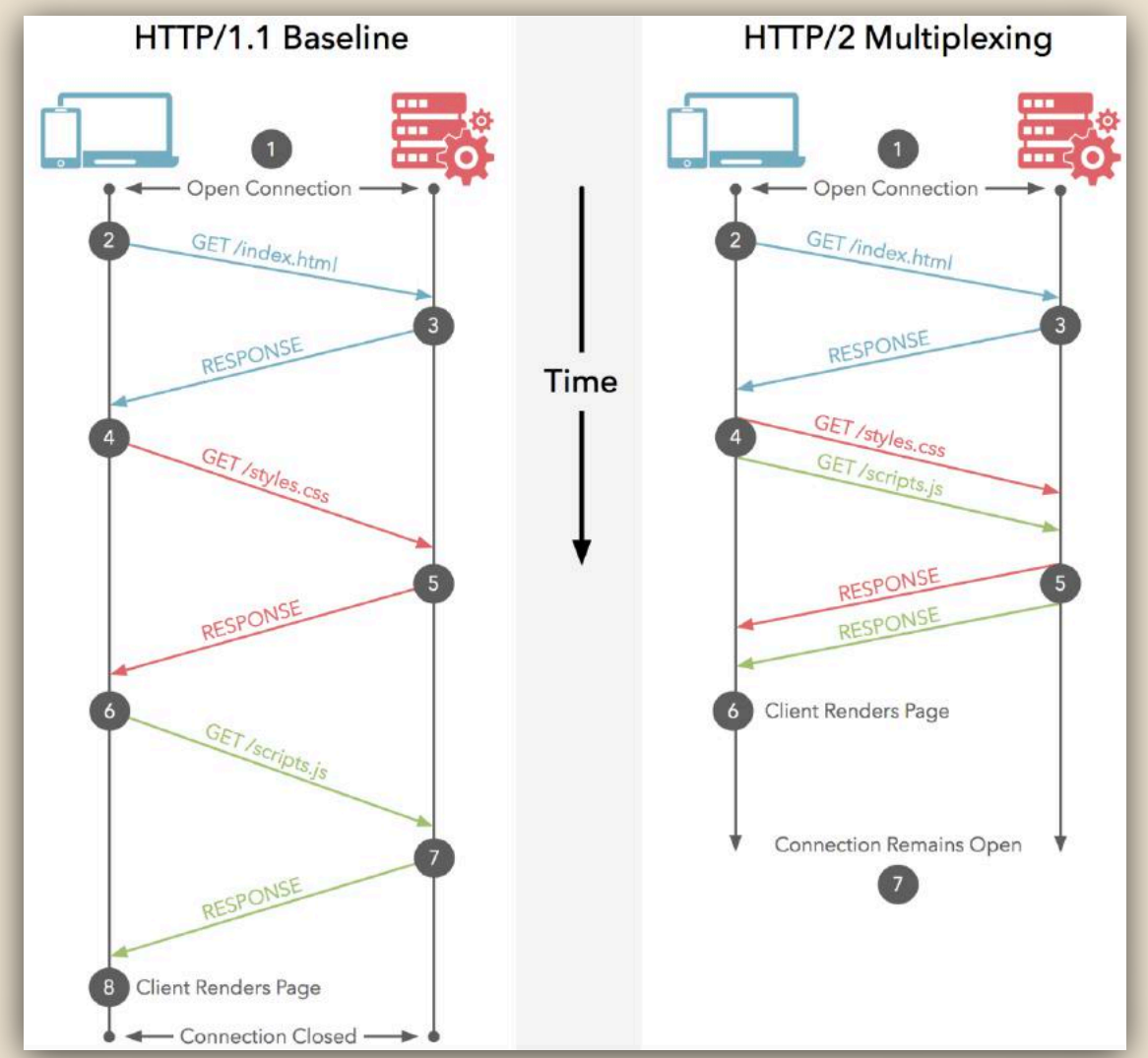
在浏览器多开连接，提高吞吐？ 同域名下chrome的连接数限制在6个

* 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
* ...

# http1.1 vs http2.0

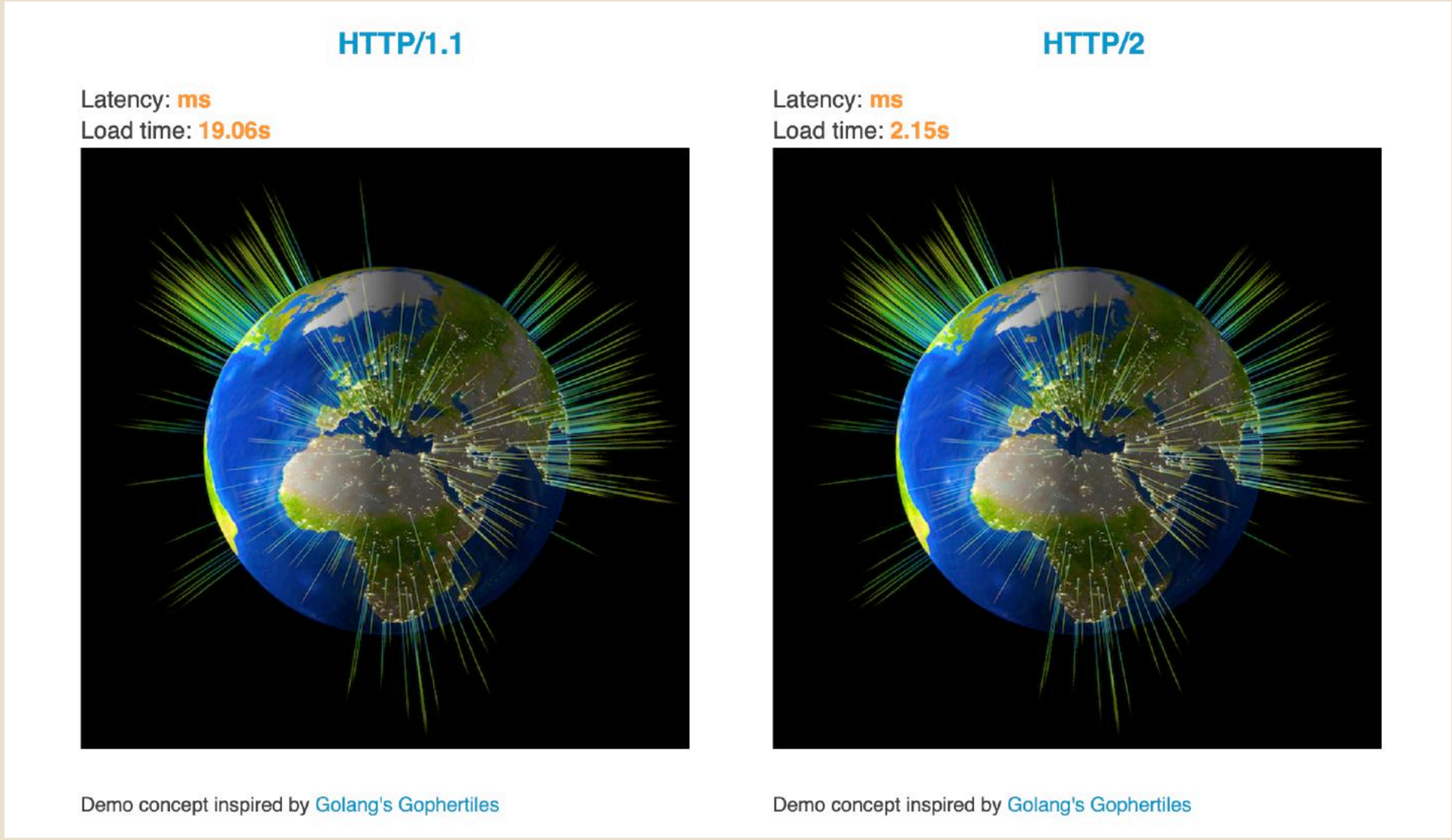



## 时延！

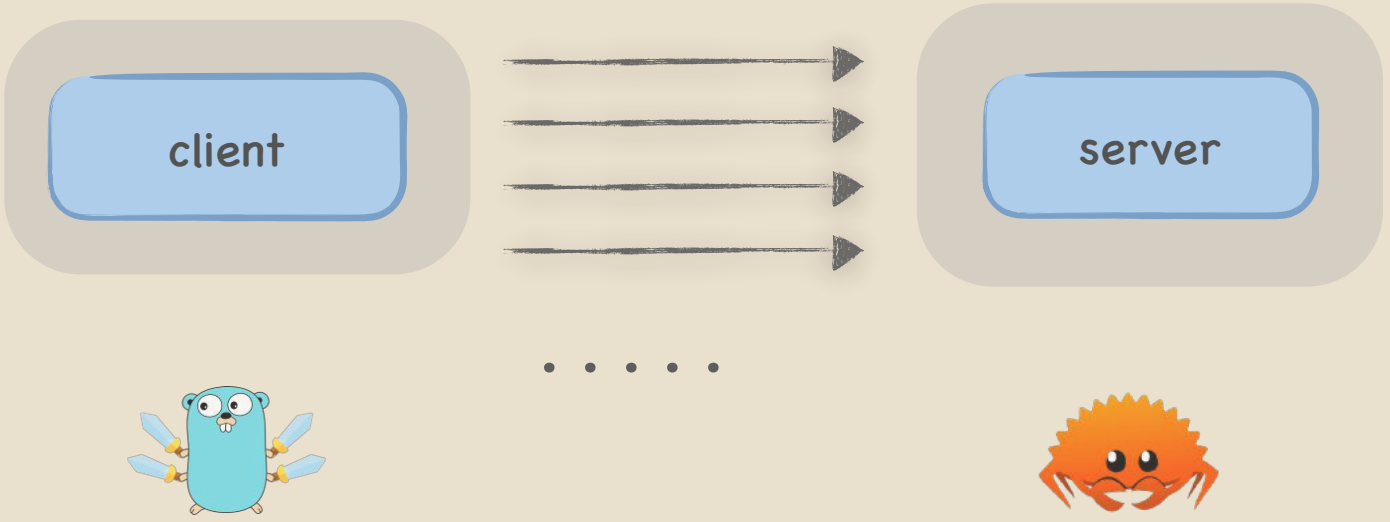
等一个是等, 多个一起等也是等 ！

* http 1.1
  * N个排队请求 ≈ N x latency
* http 2.0
  * N个并发请求 >≈ 1 x latency

https://http2.akamai.com/demo




# if don't limit conn ?




- It's still a problem
  - tcp handshake
  - tls handshake
  - slow start
  - tcp/ip protocol stack

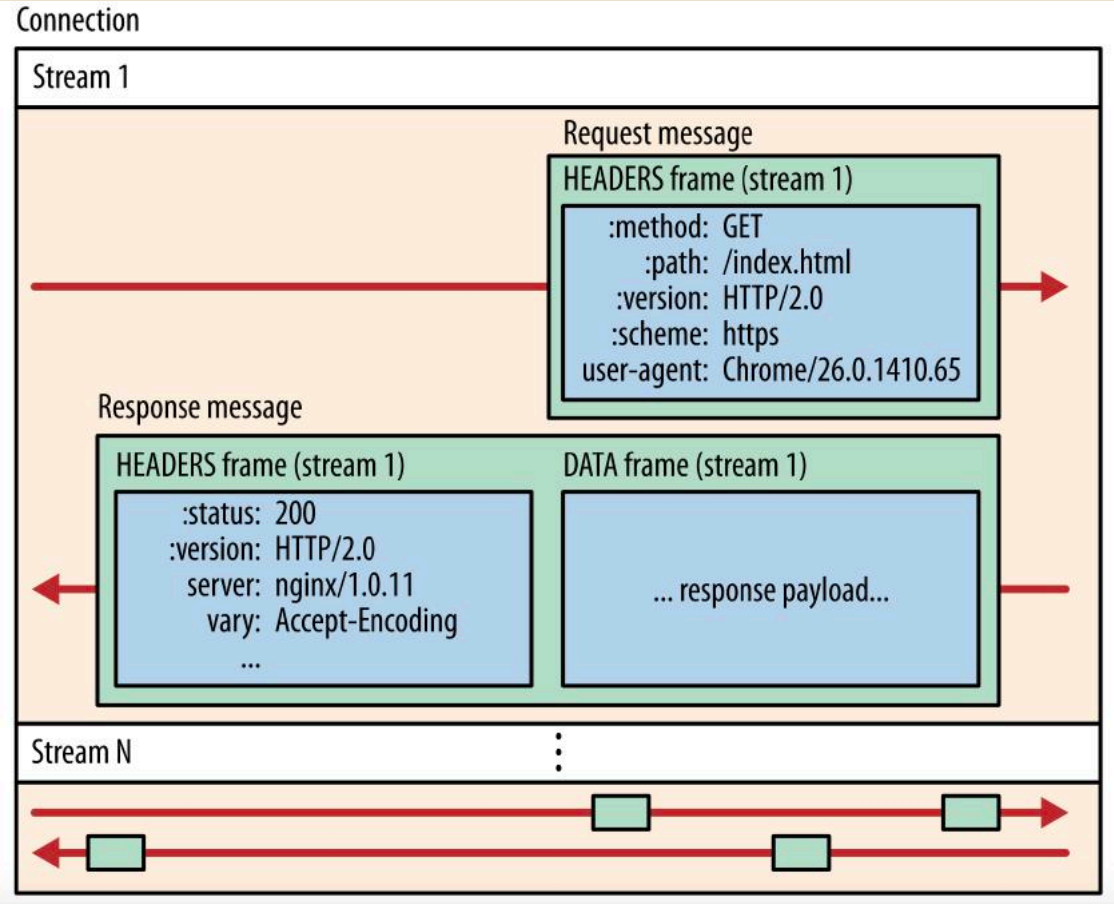

# http 2.0 优点

* 多路复用
* header压缩
* 流控
* 优先级
* 服务端推送
* ...

HTTP/2

HTTP/2

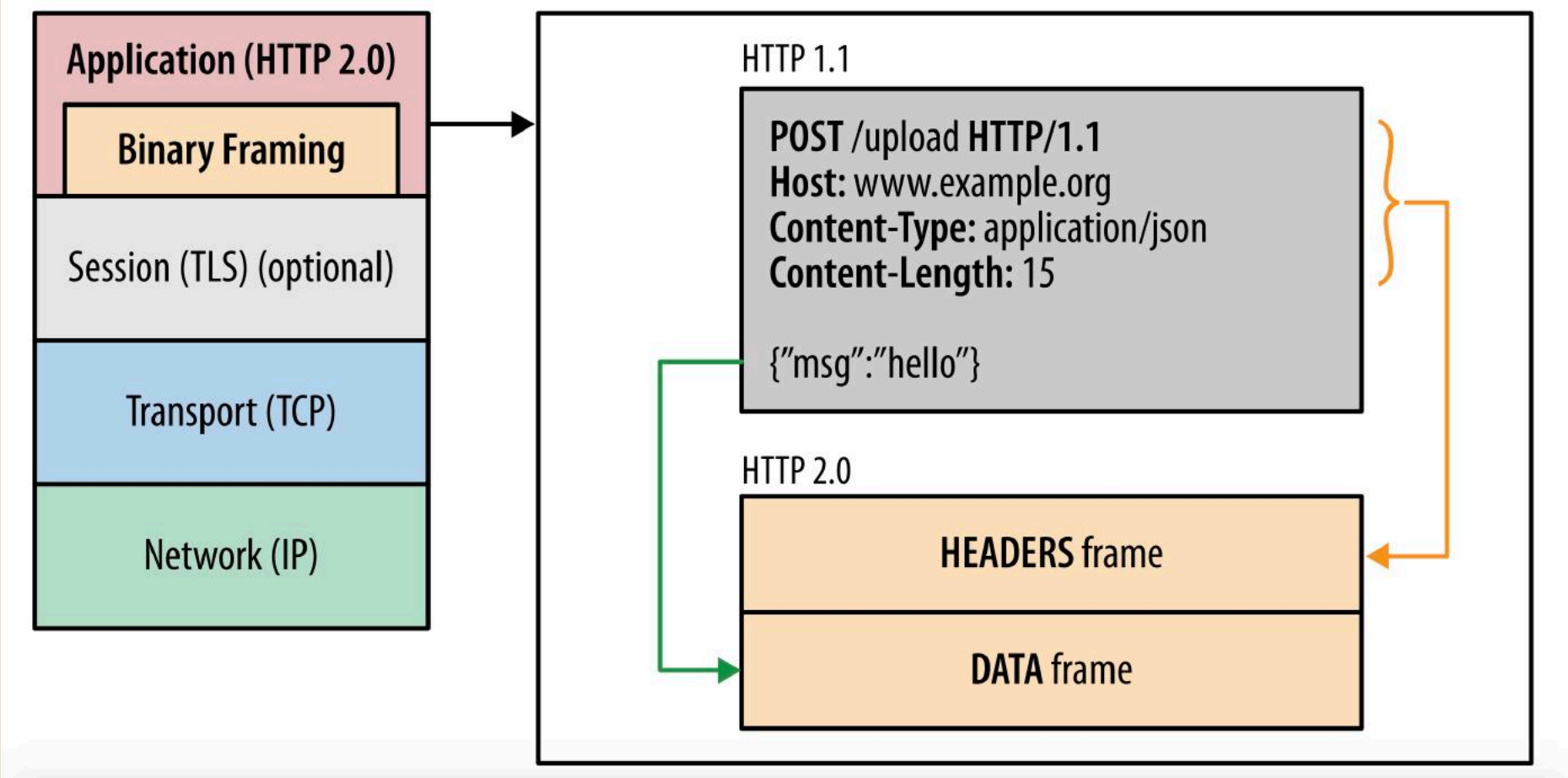
# definition




* connection
* stream
* message
* frame

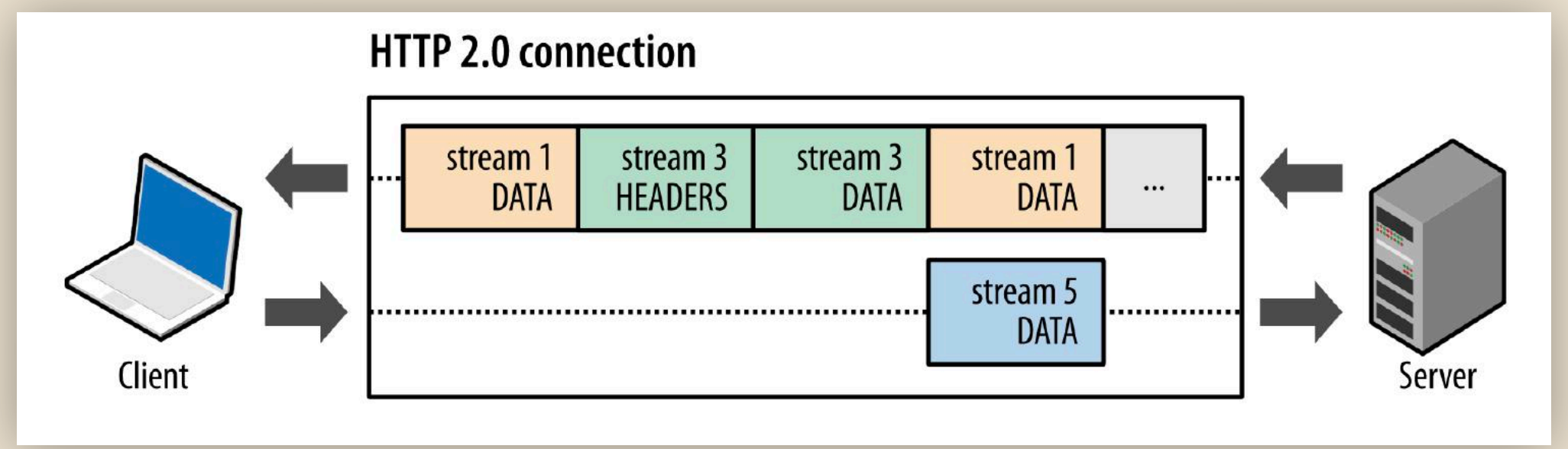
HTTP/2

# 二进制分帧层




# 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。

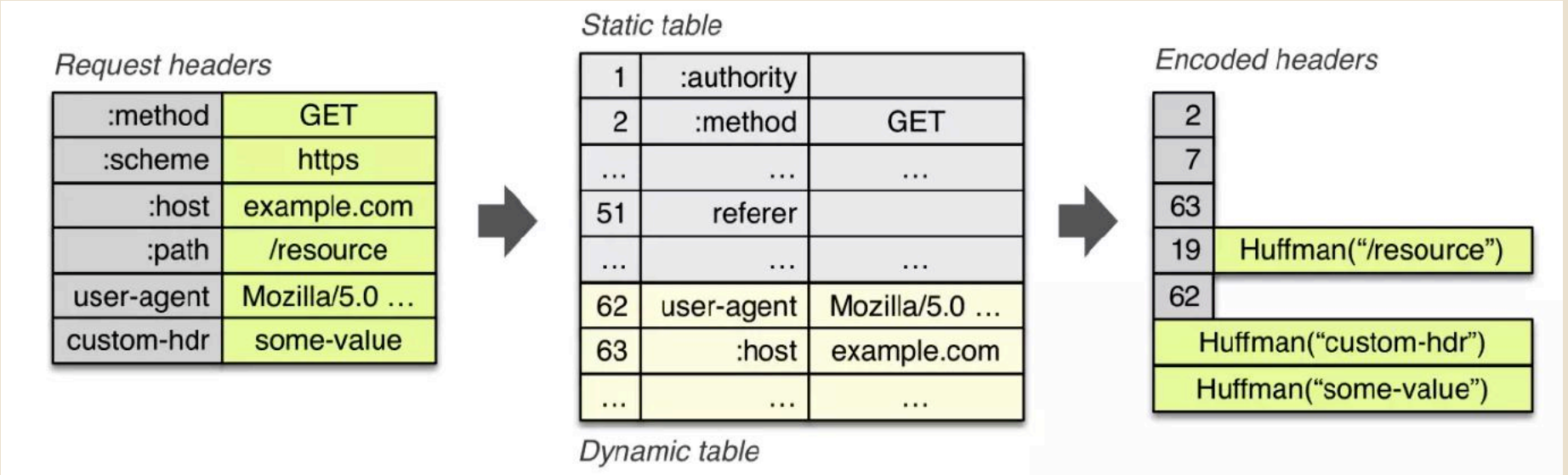



# 多路复用

```
▶ 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
```

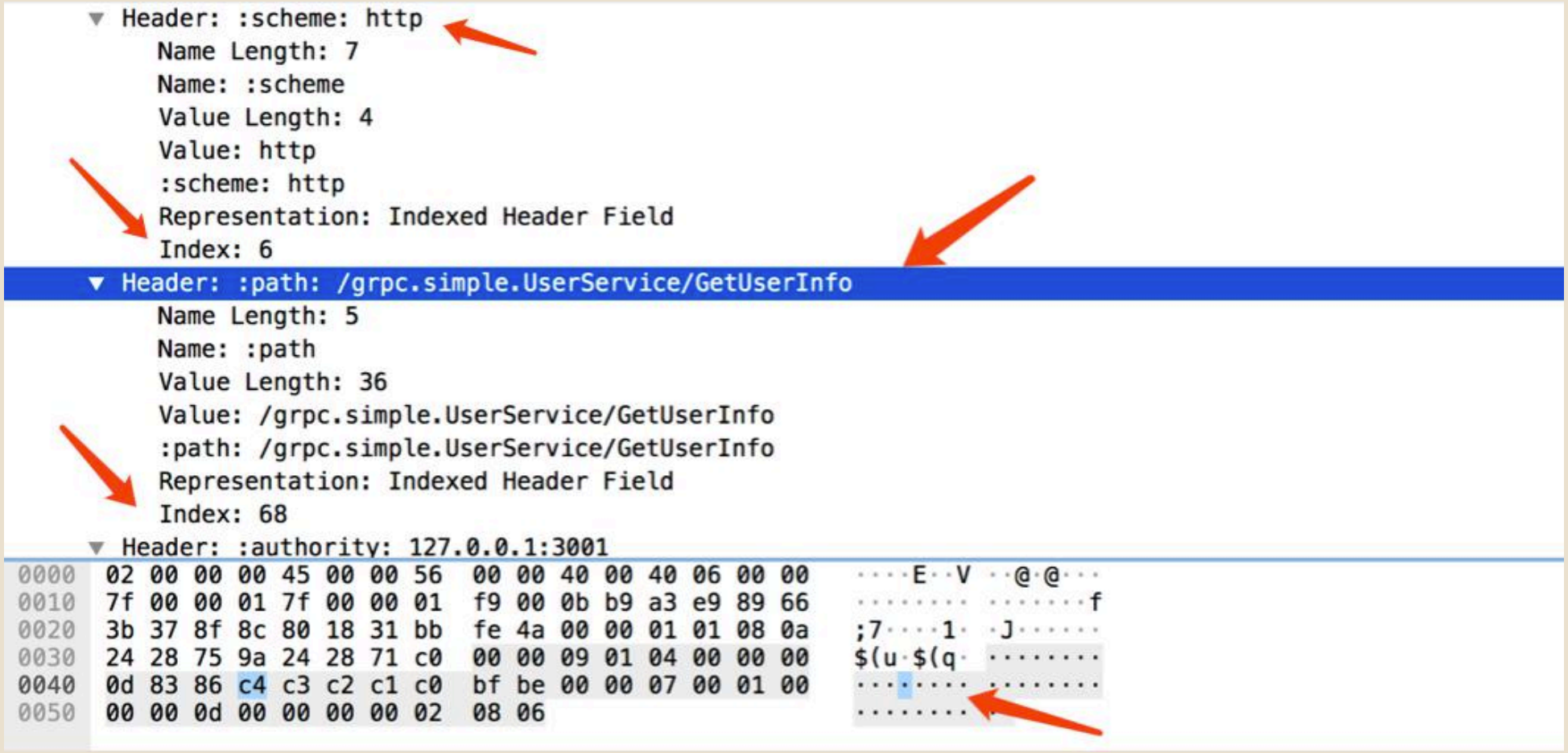
# hpack

- 硬编码静态表 （1-61）
- 两端协商扩充动态表（62-n）
- 通过传递索引号节省Header头部空间
- 使用Huffman对字符串压缩编码



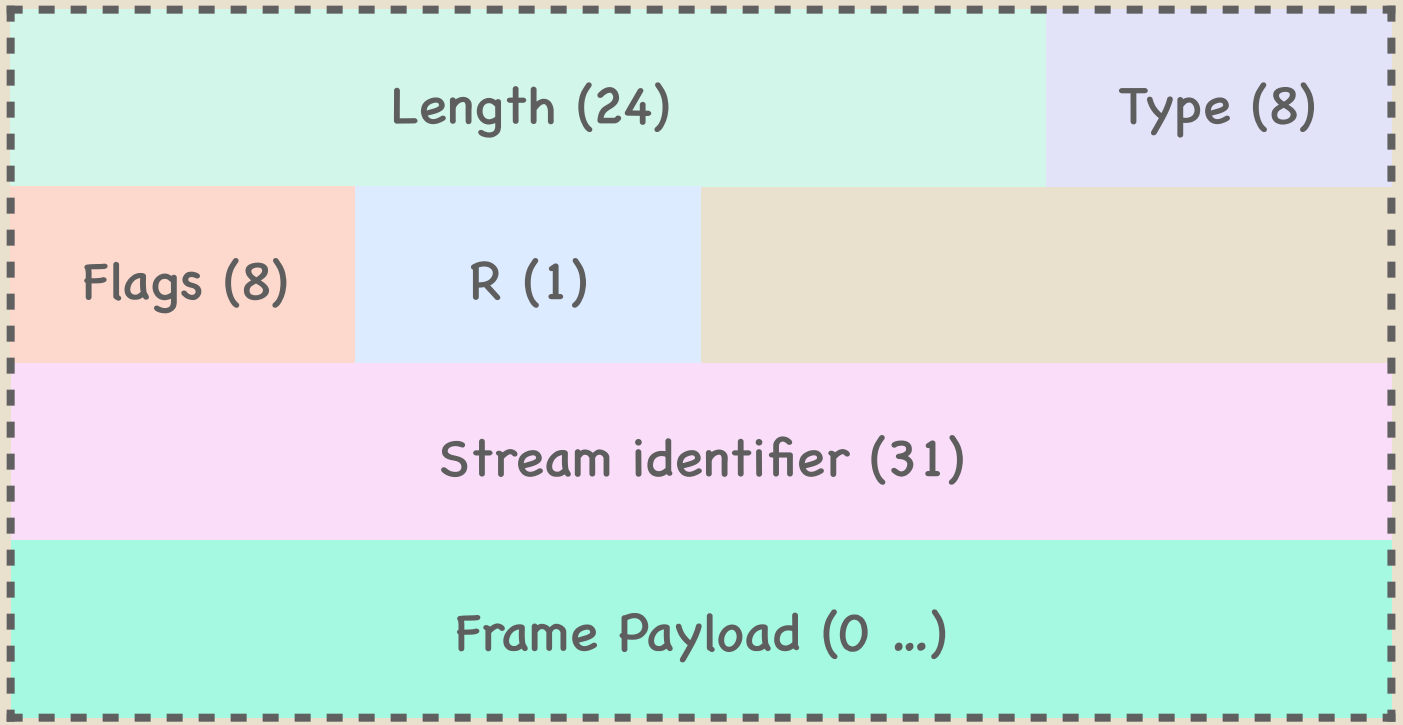

http://http2.github.io/http2-spec/compression.html#static.table.definition

# wirshark hpack



HTTP/2

# frame



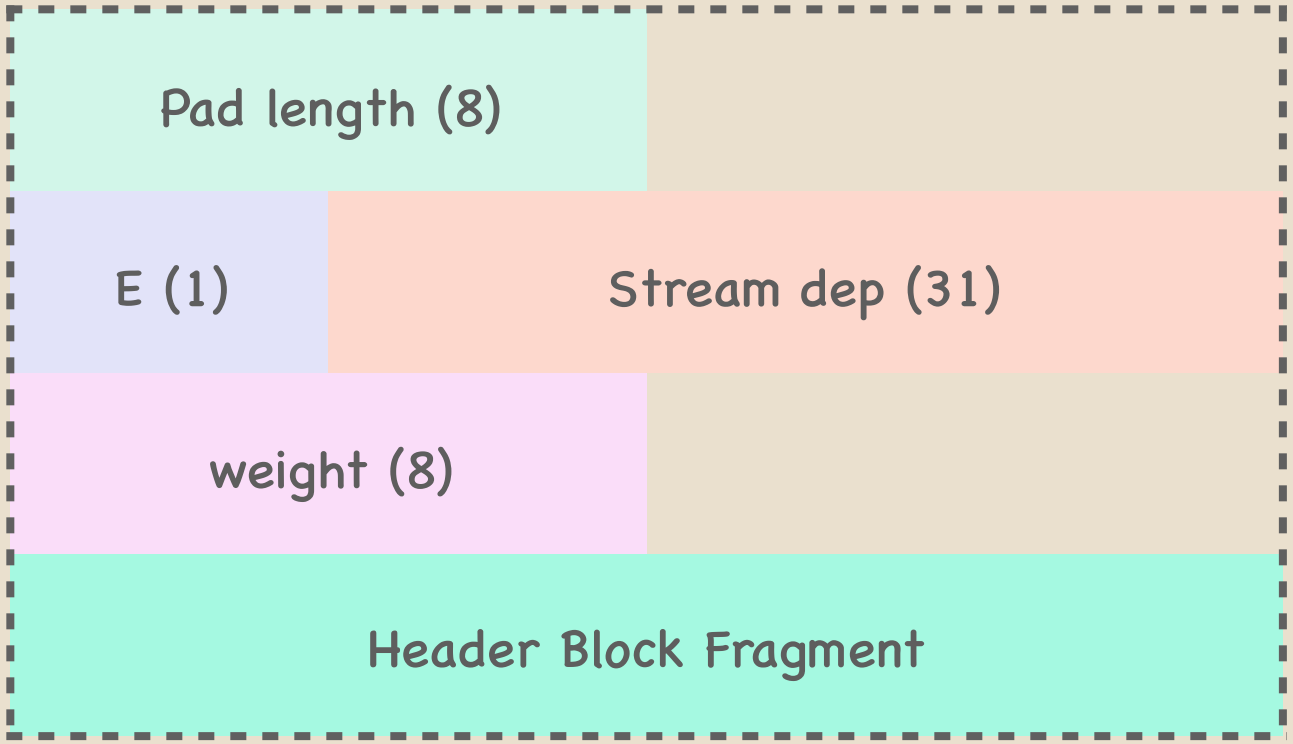

* Length ( payload size )
* Type (类型)
* Flags (状态)
* R (保留)
* Stream Identifier
* Frame Payload


# 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

HTTP/2

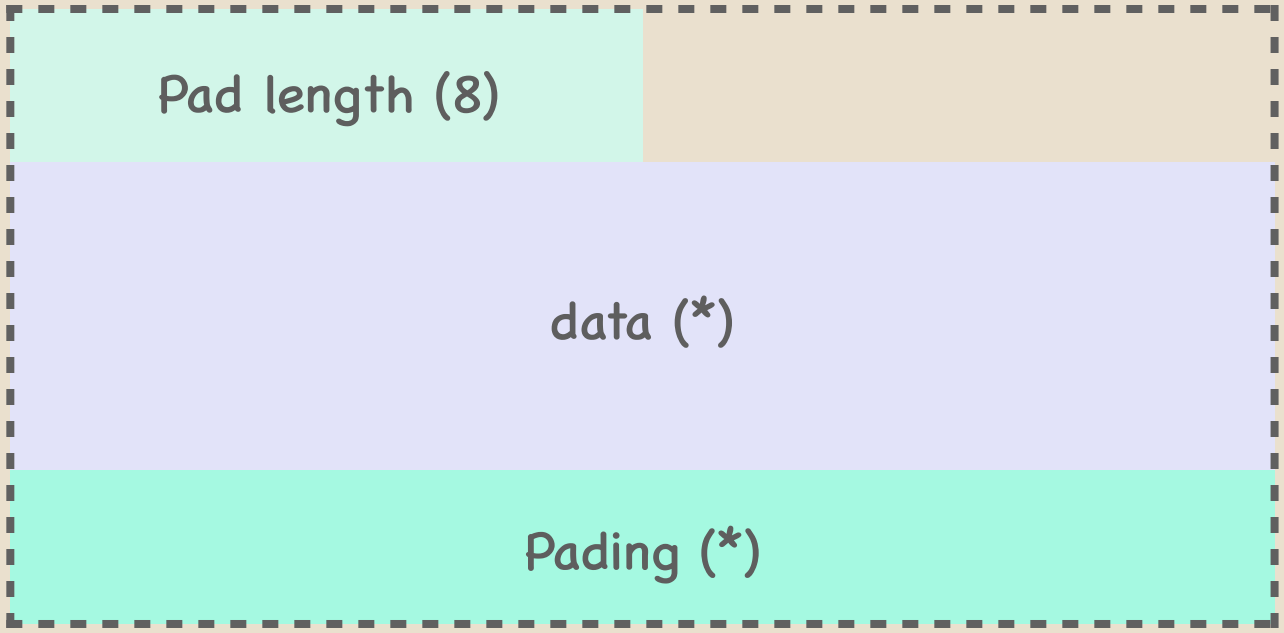
# header frame




* 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

HTTP/2

# data frame




* pad length
* data
* padding


# select protocol

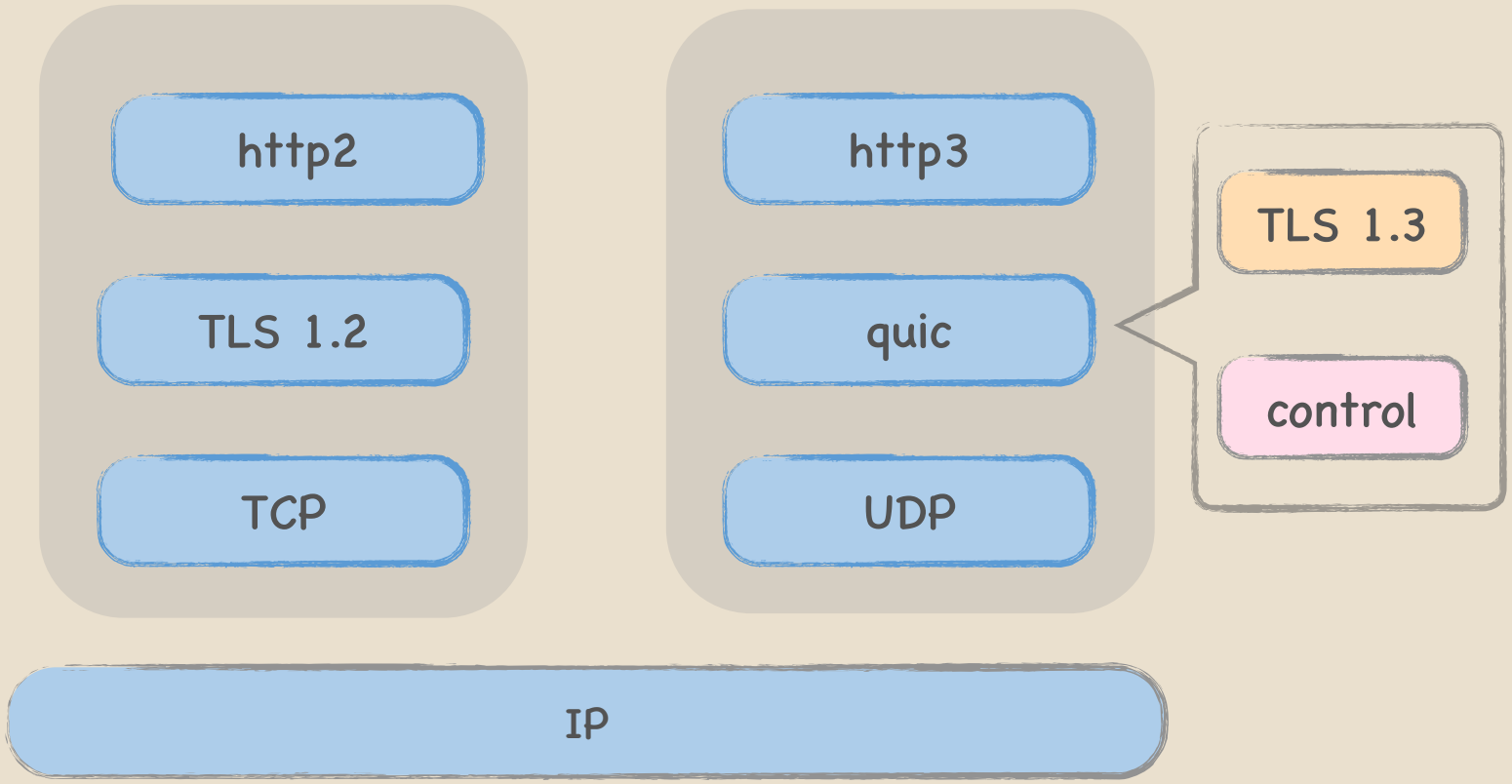
```
▶ Extension: next_protocol_negotiation
▶ Extension: signed_certificate_timestamp
▼ Extension: Application Layer Protocol Negotiation
    Type: Application Layer Protocol Negotiation (0x0010)
    Length: 23
    ALPN Extension Length: 21
  ▼ ALPN Protocol
      ALPN string length: 2
      ALPN Next Protocol: h2
      ALPN string length: 8
      ALPN Next Protocol: spdy/3.1
      ALPN string length: 8
      ALPN Next Protocol: http/1.1
▶ Extension: channel_id
```

```
 93 14.940515  10.18.60.26   114.215.116.…  TLSv1…   254 Client Hello
 94 14.942239  123.151.10…   10.18.60.26    SSL      349 Continuation Data
 97 14.970500  114.215.11…   10.18.60.26    TLSv1…  1514 Server Hello
 99 14.970505  114.215.11…   10.18.60.26    TLSv1…  1347 Certificate
102 14.971300  10.18.60.26   114.215.116.…  TLSv1…   180 Client Key Exchange, Change Cipher Spe…
103 14.987435  114.215.11…   10.18.60.26    TLSv1…   312 New Session Ticket, Change Cipher Spec…
104 14.987437  114.215.11…   10.18.60.26    TLSv1…   123 Application Data
111 15.135321  10.18.60.26   114.215.116.…  TLSv1…   107 Application Data
```

```
▶ Extension: renegotiation_info
▶ Extension: ec_point_formats
▶ Extension: SessionTicket TLS
▶ Extension: status_request
▶ Extension: signed_certificate_timestamp
▼ Extension: Application Layer Protocol Negotiation
    Type: Application Layer Protocol Negotiation (0x0010)
    Length: 5
    ALPN Extension Length: 3
  ▼ ALPN Protocol
      ALPN string length: 2
      ALPN Next Protocol: h2
```

- http2 without tls (h2c)
  - use 101 upgrade
- http2 over tls
  - client hello with APLN
  - server hello with ALPN
- when client and server support h2; use h2

# HTTP 3




* http2
  * handshark cost
    * 1 rtt from tcp handshake
    * 2 rtt from tls
  * TCP HOL Blocking
* http3
  * use udp protocol
  * only need 1 rtt
  * 1 rtt from tls 1.3


# protobuf


# what is protobuf ?

```proto
syntax = "proto3";

option go_package = "google.golang.org/grpc/examples/helloworld/helloworld";

package helloworld;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message
message HelloRequest {
  string   name    = 1;
  int32    age     = 2;
  int32    phone   = 3;
  bool     sex     = 4;
  bytes    data    = 5;
  repeated string hobbies = 6;
}

// The response message
message HelloReply {
  string message = 1;
}
```

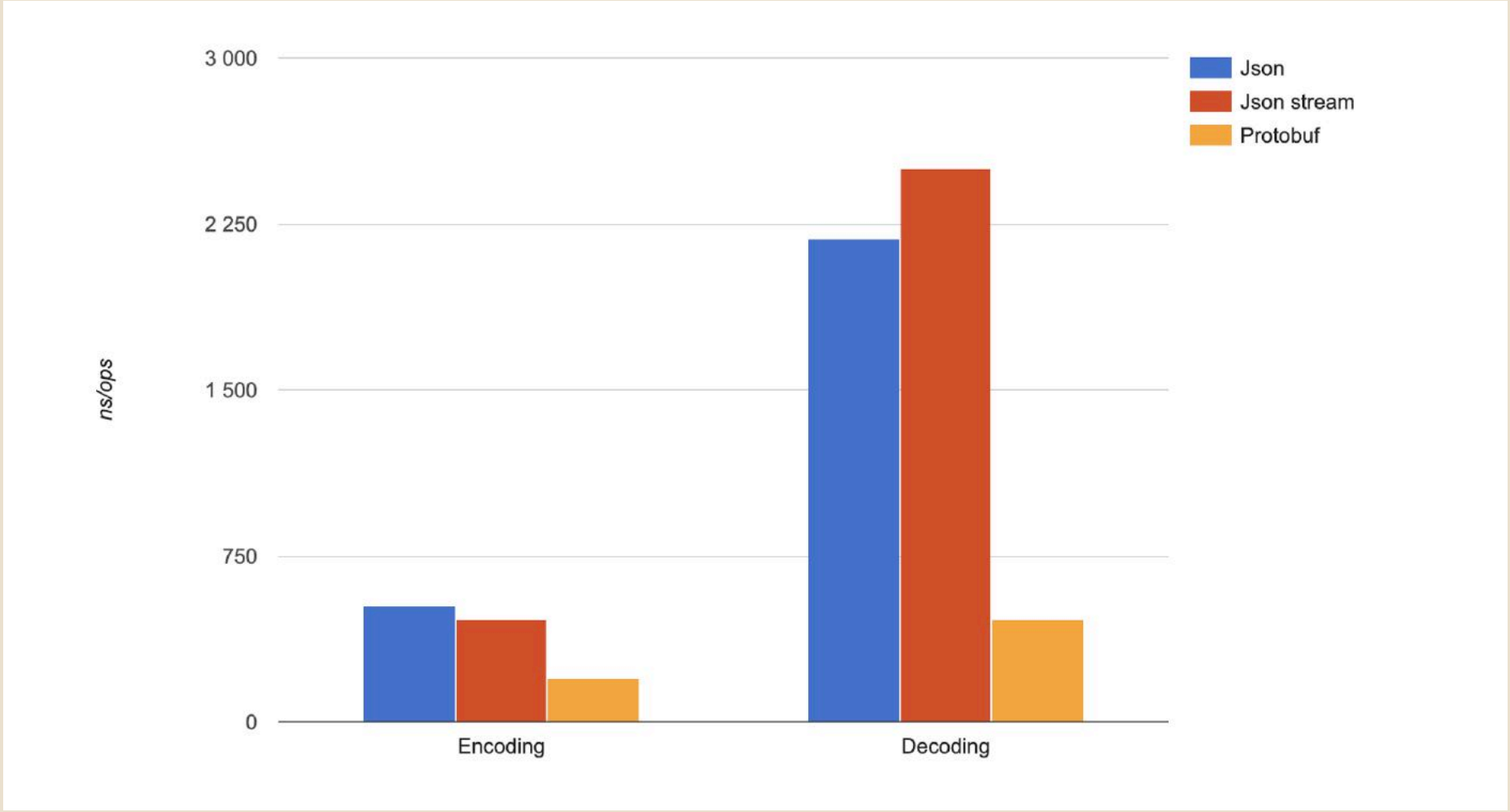
- protobuf
  - idl define
  - generate code
  - high performance
  - compress
  - binary
  - more

protobuf
Protocol Buffers

# select protobuf ?

- select json ?
  - waste space
  - lost type
  - not safe
  - low performance
- select thrift ?
  - thrift is good, but ...
    - grpc from google
    - protobuf from google
    - thrift from facebook

# pb vs json


3 000
2 250
1 500
750
0
ns/ops
Encoding
Decoding
Json
Json stream
Protobuf

# json vs pb (optimize size)

```
{
    "id": 12345678,
    "name": "rfyiamcool",
    "address": "beijing",
    "wages": 6666,
    "fans": [11, 22, 333, 444],
    "target": "hello\'\"\'\" world"
}
```

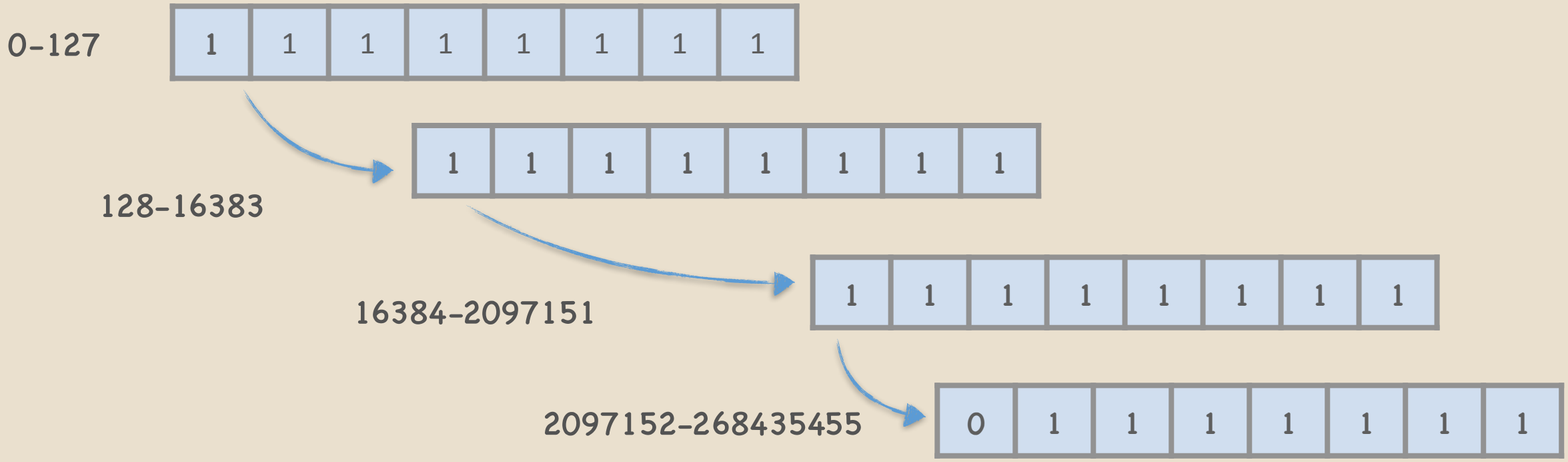
- json
  - with keyname
  - use {}, "" :, [], space ... as delimiter
  - 12345678 is text
  - ...

- pb
  - tags replace keyname
  - tlv replace delimiter
  - 12345678 is varint
  - ...

# protobuf optimize

- 空间优化
  - TLV存储
    - 节省keyname
    - 减少了分隔符
    - 数据格式更紧凑, 空间利用率高
  - 对数字进行varint, zigzag编码
- 高性能
  - TLV编码
  - 不需要遍历解析边界
  - 生成解析代码
  - 初始化各tab对应的解码器
  - 减少反射
  - more ...

# varint实现




https://golang.org/src/encoding/binary/varint.go

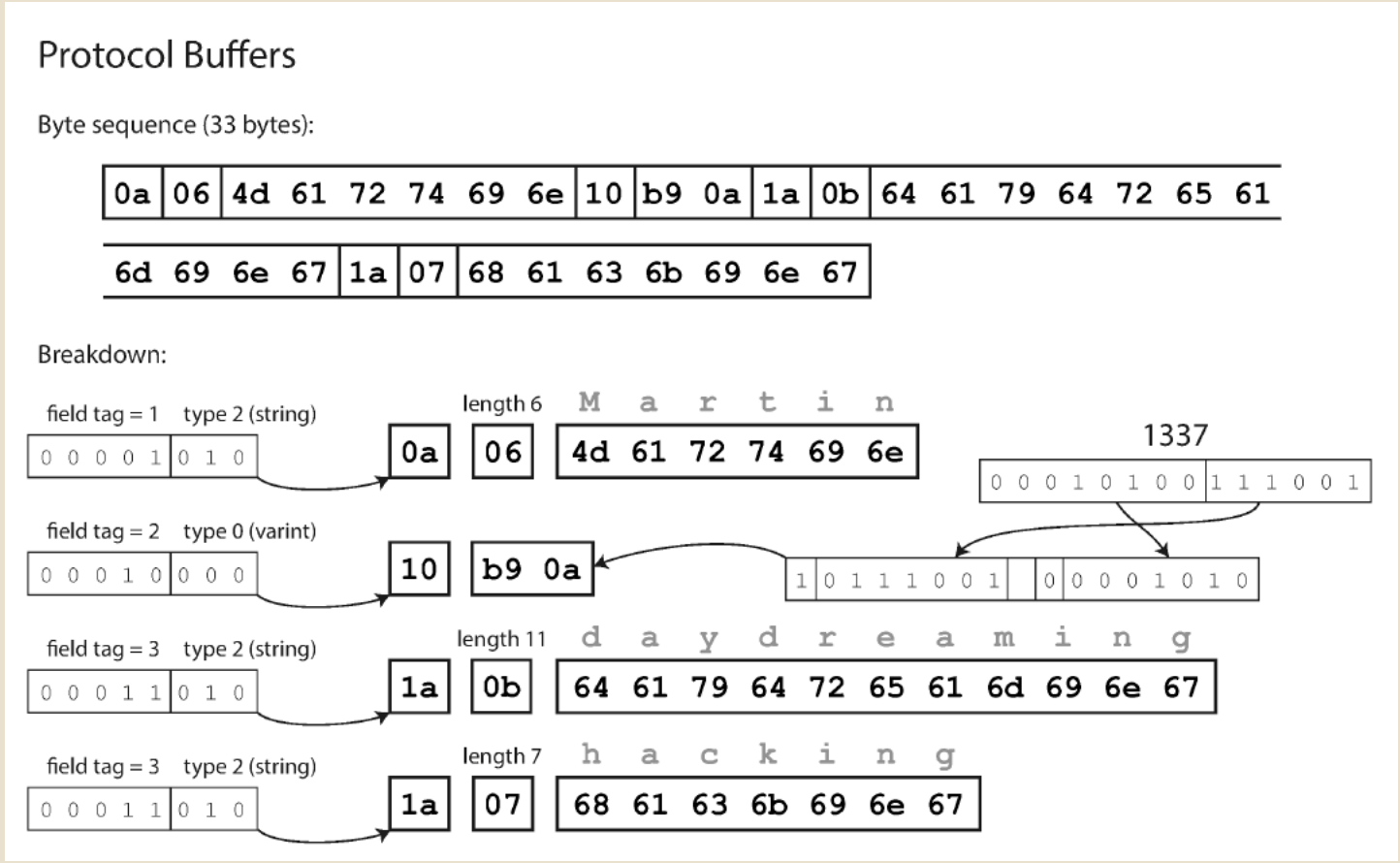
# wire type

| wire type | encode | encode length | 存储方式 | Used For |
|---|---|---|---|---|
| 0 | Varint | 变长 (1-10个字节) | T-V | int32, int64, uint32, uint64, sint32, sint64, bool, enum |
| 1 | 64-bit | 固定8个字节 | T-V | fixed64, sfixed64, double |
| 2 | Length-delimited | 变长 | T-L-V | string, bytes, embedded messages, packed repeated fields |
| 3 | Start group | 已经弃用 | 弃用 | groups (deprecated) |
| 4 | End group | | 弃用 | groups (deprecated) |
| 5 | 32-bit | 固定4个字节 | T-V | fixed32, sfixed32, float |

* 变长
  * TLV
* 定长
  * TV

# TLV

| tag | length | value | tag | length (option) | value |
|---|---|---|---|---|---|
| field = 1<br>type = 2 | 10 | xiaorui.cc | field = 2<br>type = 0 | 0 | 111 |




* tag = (field_number << 3) | wire_type
  * 0 bit
    * extend byte
  * 1-5 bit
    * field_number
  * 6-8 bit
    * wire_type
* length encode is varint too !!!
* value ...

# protobuf types

| | | |
|---|---|---|
| enum | string | `"FOO_BAR"` |
| map<K,V> | object | `{"k": v, ...}` |
| repeated V | array | `[v, ...]` |
| bool | true, false | `true, false` |
| string | string | `"Hello World!"` |
| bytes | base64 string | `"YWJjMTIzIT8kKiYoKSctPUB+"` |
| int32, fixed32, uint32 | number | `1, -10, 0` |
| int64, fixed64, uint64 | string | `"1", "-10"` |
| float, double | number | `1.1, -10.0, 0, "NaN", "Infinity"` |

| | | |
|---|---|---|
| Any | `object` | `{"@type": "url", "f": v, ... }` |
| Timestamp | string | `"1972-01-01T10:00:20.021Z"` |
| Duration | string | `"1.000340012s", "1s"` |
| Struct | `object` | `{ ... }` |
| Wrapper types | various types | `2, "2", "foo", true, "true", null, 0, ...` |
| FieldMask | string | `"f.fooBar,h"` |
| ListValue | array | `[foo, bar, ...]` |
| Value | value | |
| NullValue | null | |
| Empty | object | {} |

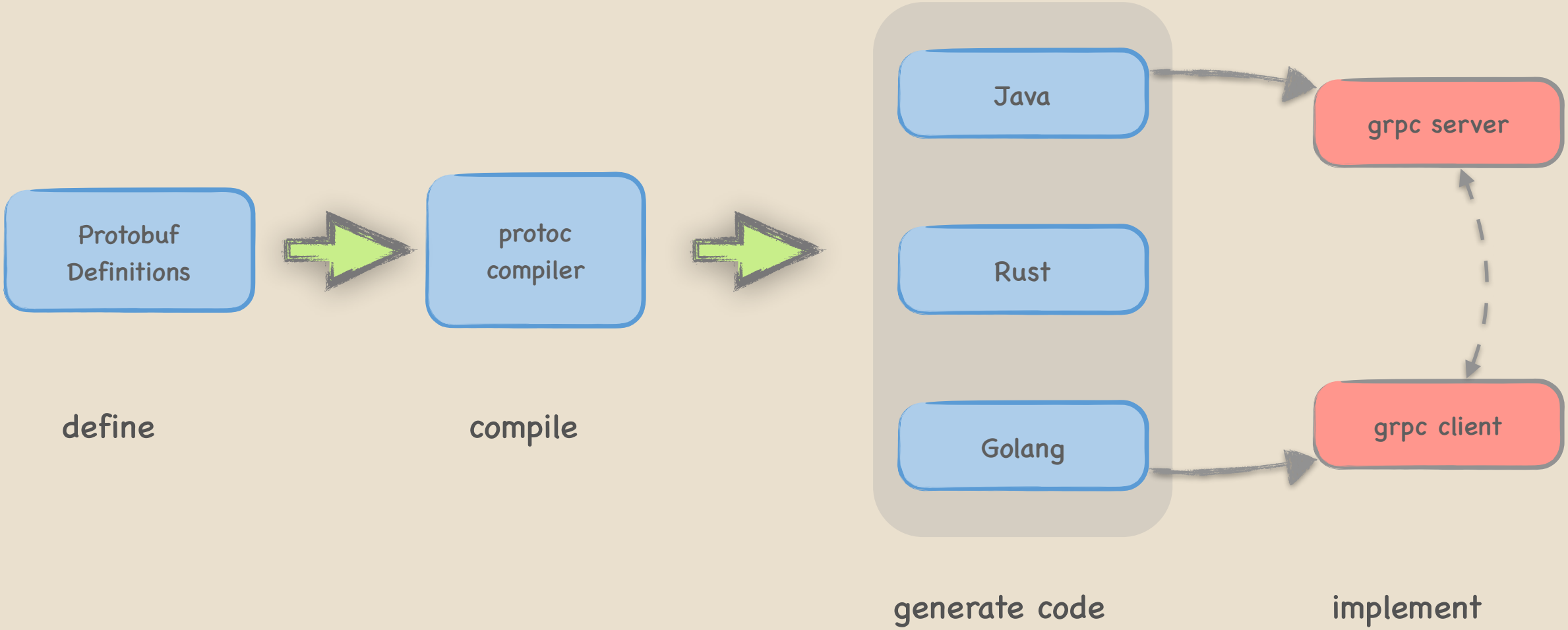
# string not compress !

protobuf
Protocol Buffers

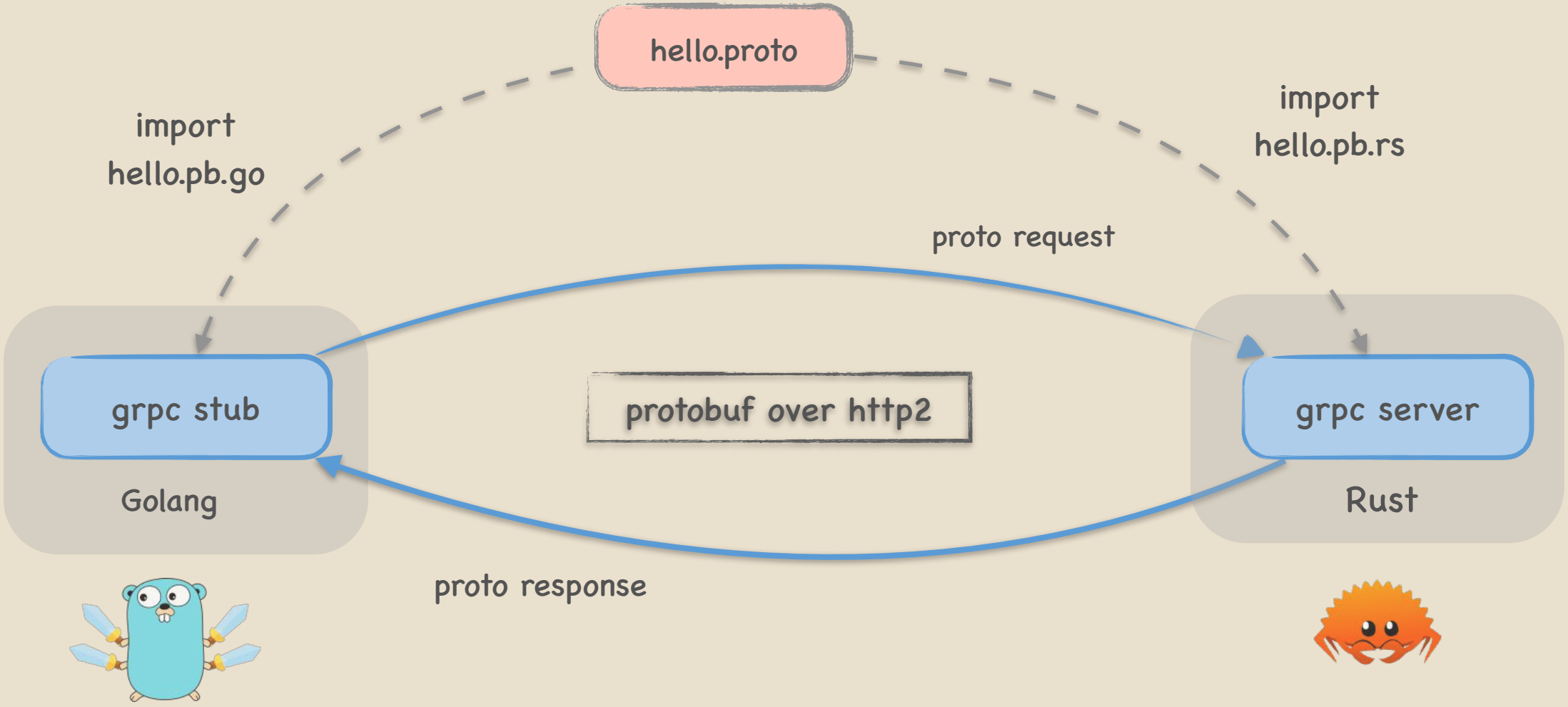
* zstd
* lz4
* snappy
* ...


# grpc

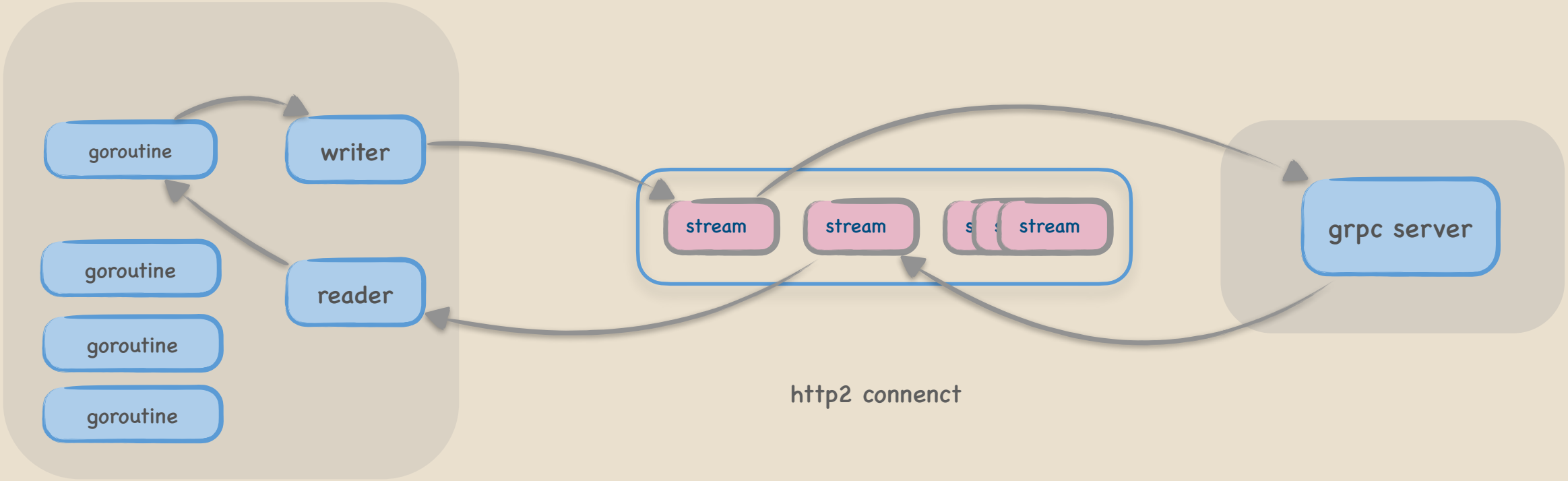

# grpc workflow



# how to reqeust ?

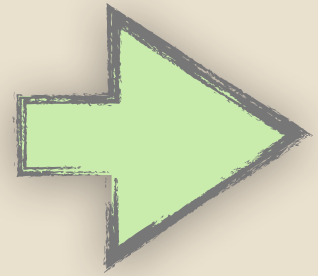


# deep request

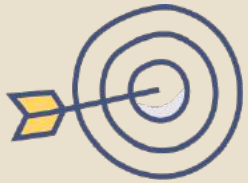



single worer ensure thread safe

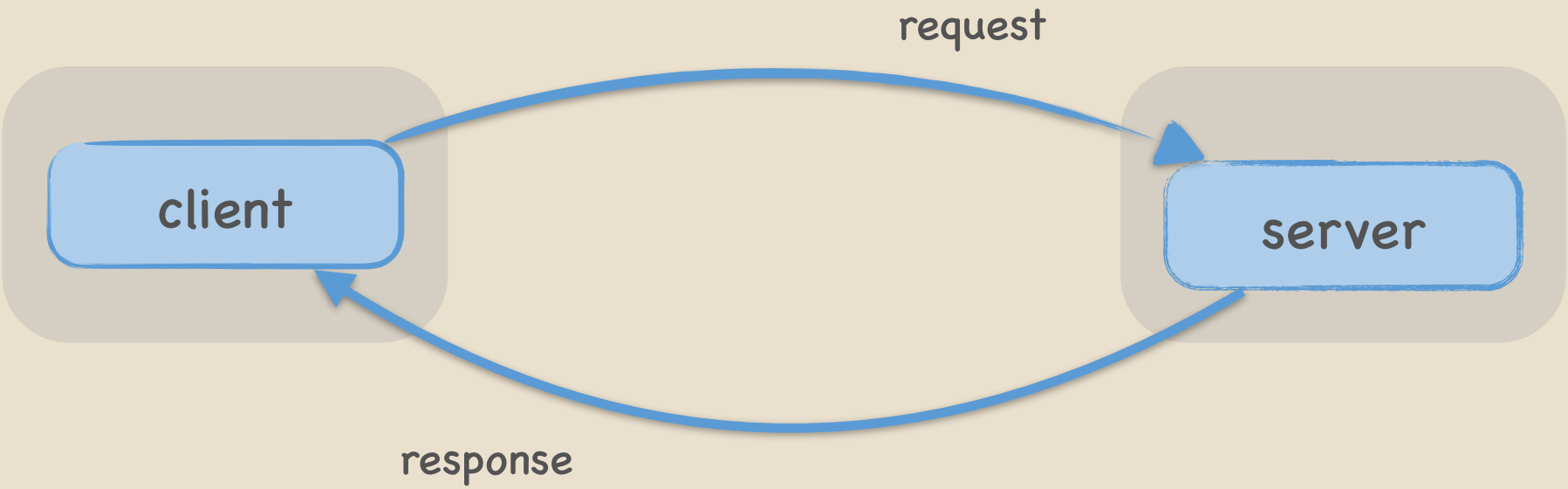
# 4 type of grpc

- unary
- client streaming
- server streaming
- bidi streaming



# unary request

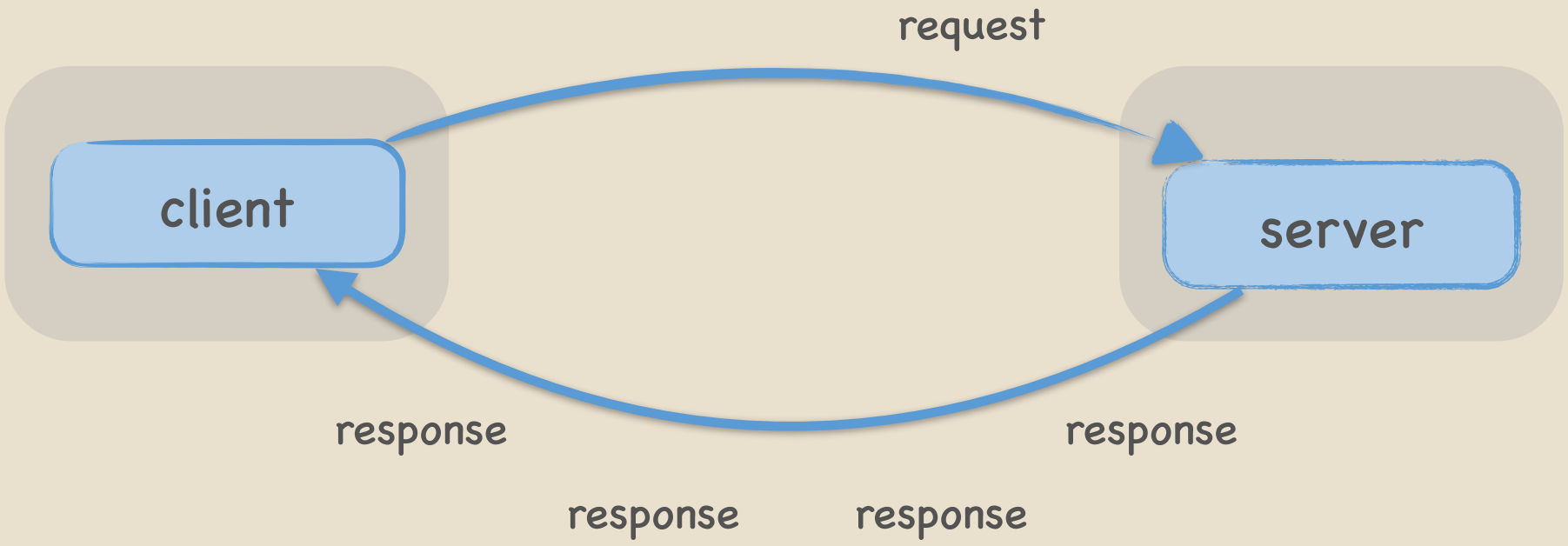




一去一回

gRPC

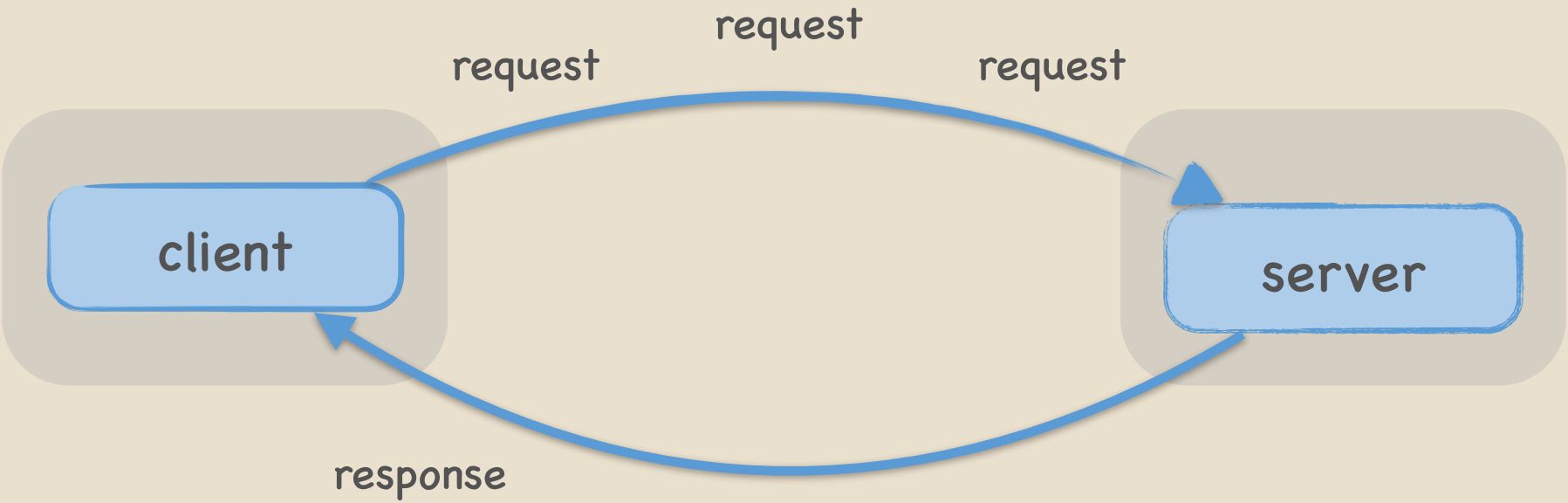
# server streaming





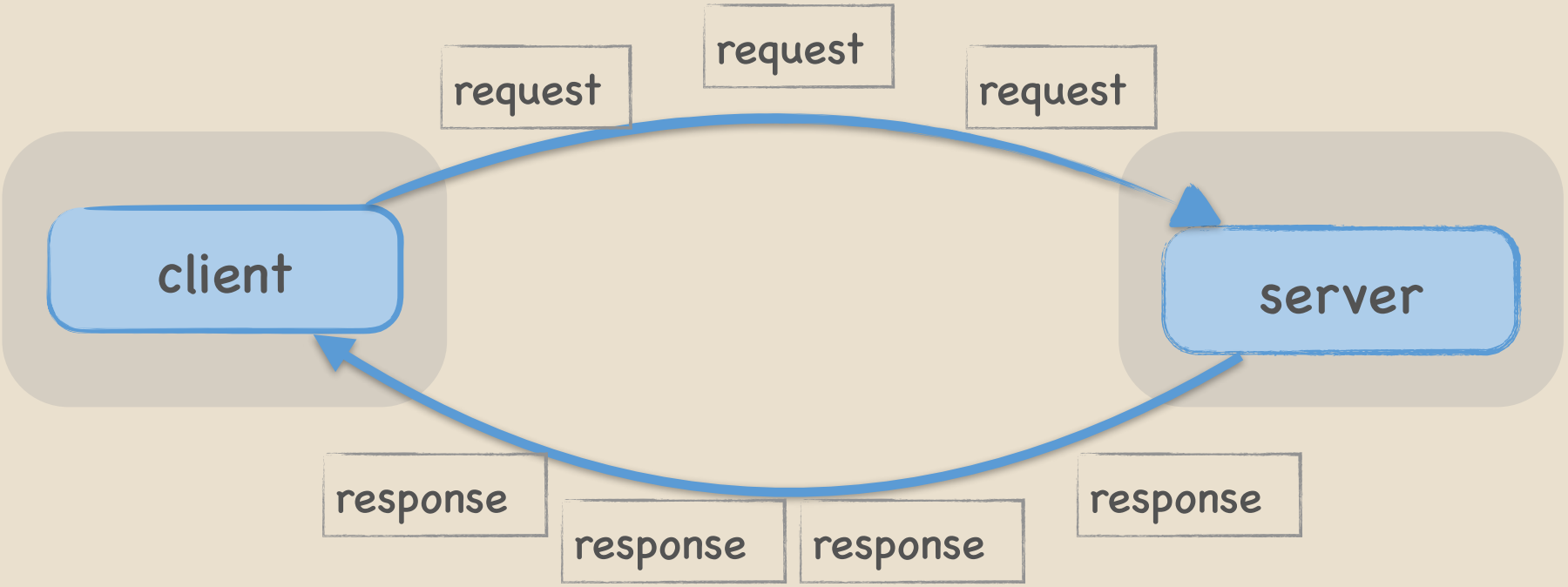
server单边推送数据

# client streaming




client单边推送数据

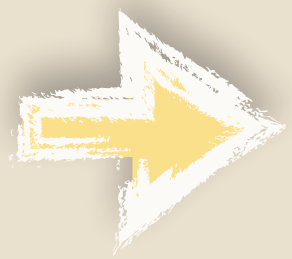
# bidi streaming




双向推送流数据

# Interceptor

- middleware (third party)
  - unary interceptor
  - stream interceptor



- try catch recovery
- rate limiter
- log wrapper
- circuit breaker
- auth wrapper
- … …

# other feature

- metadata
- keepalive
- encryption
- reflection
- ...

- grpc client
  - WithBalancer
  - WithBlock
  - WithInsecure
  - WaitForReady
  - WithTransportCredentials
  - ...

- grpc server
  - MaxConcurrentStreams
  - InitialWindowSize
  - InitialConnWindowSize
  - UnknownServiceHandler
  - CustomCodec
  - ...

# hello world

```
package helloworld;

service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

message HelloRequest {
  string name = 1;
}

message HelloReply {
  string message = 1;
```

protoc --go_out=plugins=grpc:. route_guide.proto

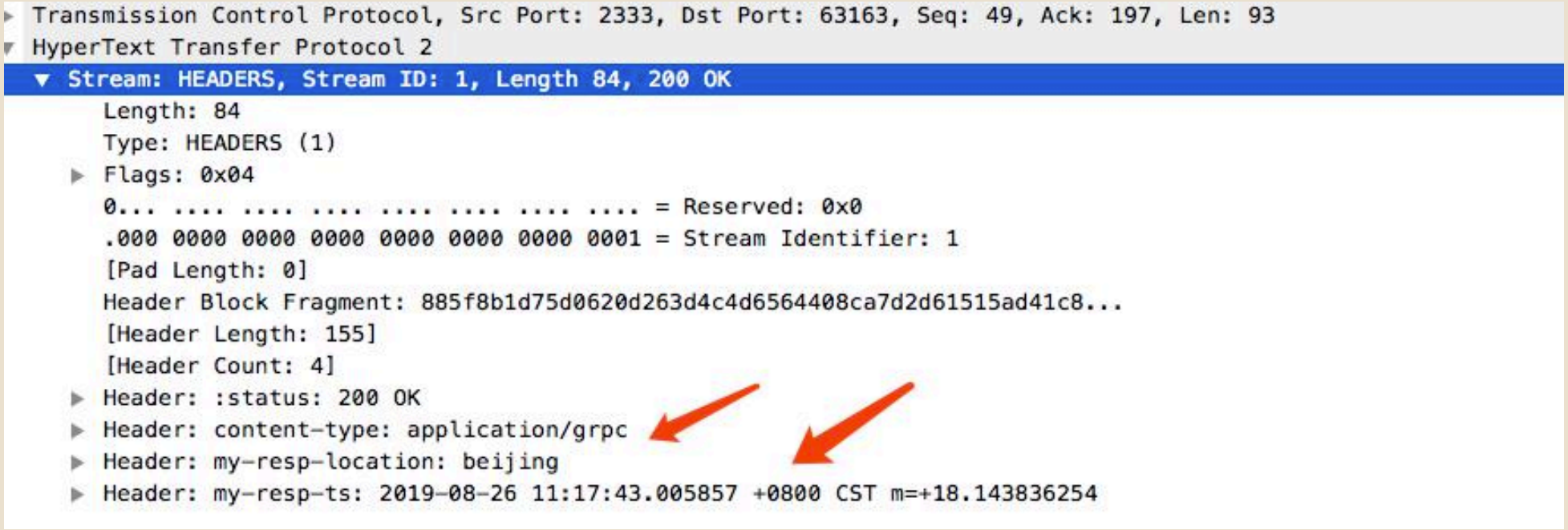
```go
package main

import (
    "context"
    "log"
    "net"

    "google.golang.org/grpc"
    pb "google.golang.org/grpc/examples/helloworld/helloworld"
)

type server struct {
    pb.UnimplementedGreeterServer
}

// SayHello implements helloworld.GreeterServer
func (s *server) SayHello(ctx context.Context, in *pb.HelloRequest) (*pb.HelloReply, error) {
    log.Printf("Received: %v", in.GetName())
    return &pb.HelloReply{Message: "Hello " + in.GetName()}, nil
}

func main() {
    lis, err := net.Listen("tcp", ":50051")
    if err != nil {
        log.Fatalf("failed to listen: %v", err)
    }
    s := grpc.NewServer()
    pb.RegisterGreeterServer(s, &server{})
    if err := s.Serve(lis); err != nil {
        log.Fatalf("failed to serve: %v", err)
    }
}
```

```go
package main

import (
    "context"
    "log"
    "time"

    "google.golang.org/grpc"
    pb "google.golang.org/grpc/examples/helloworld/helloworld"
)

const (
    address     = "localhost:50051"
    defaultName = "xiaorui.cc"
)

func main() {
    conn, err := grpc.Dial(address, grpc.WithInsecure(), grpc.WithBlock())
    if err != nil {
        log.Fatalf("did not connect: %v", err)
    }
    defer conn.Close()
    c := pb.NewGreeterClient(conn)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second)
    defer cancel()
    r, err := c.SayHello(ctx, &pb.HelloRequest{Name: defaultName})
    if err != nil {
        log.Fatalf("could not greet: %v", err)
    }
    log.Printf("Greeting: %s", r.GetMessage())
}
```

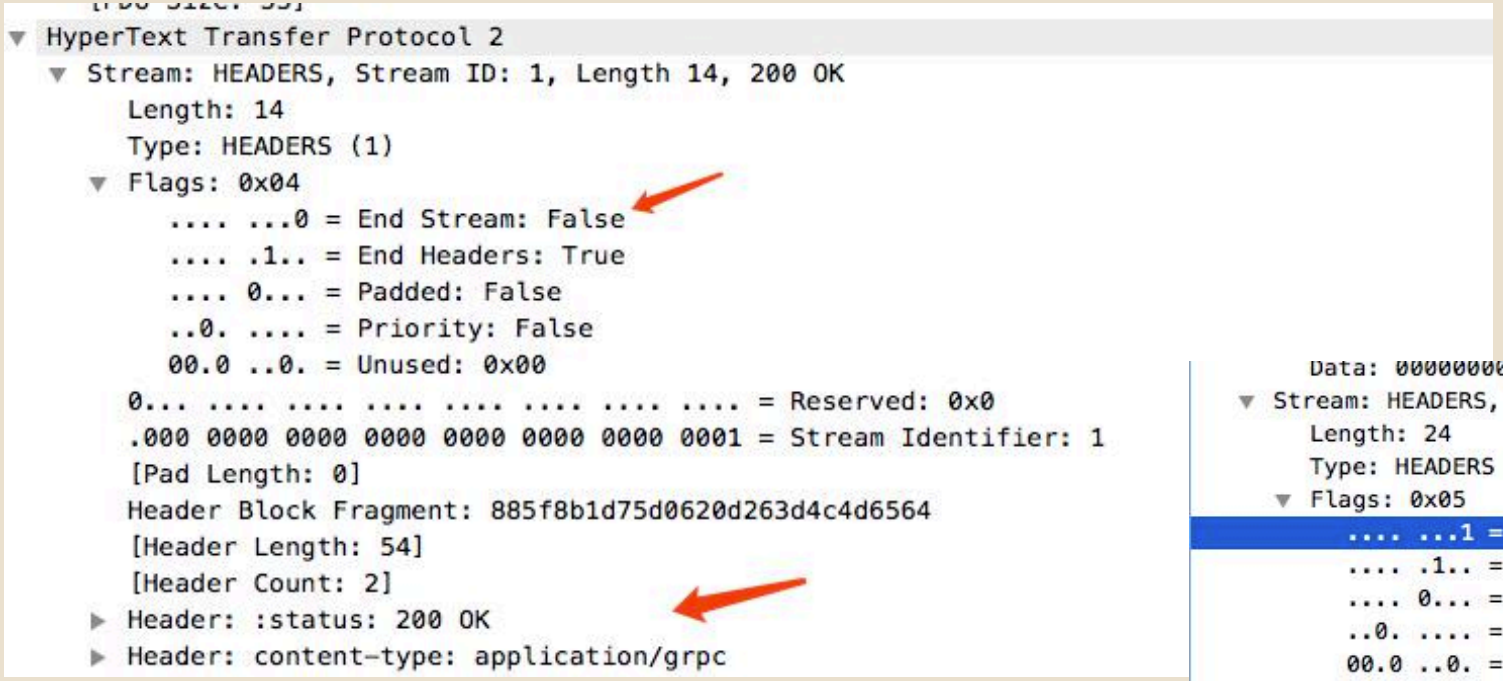
# grpc request

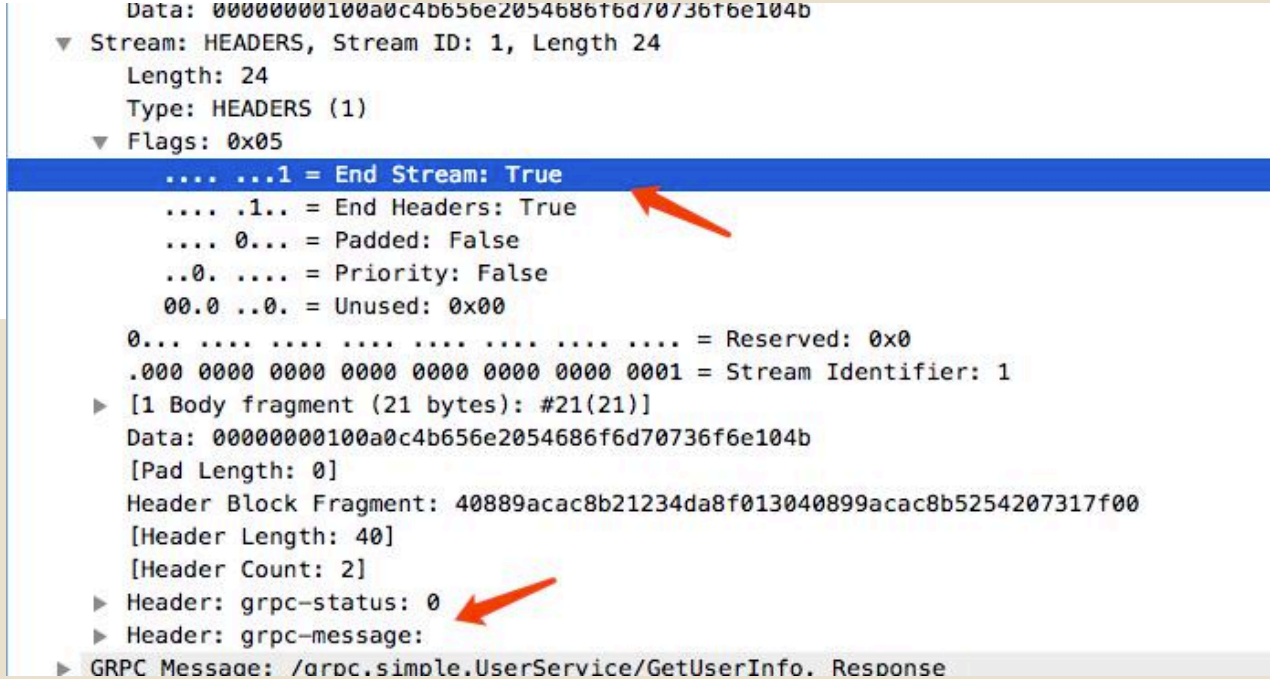

- header frame
  - metadata in header
  - service/method in header
- data frame
  - protobuf

# grpc response



# end flags


HyperText Transfer Protocol 2
Stream: HEADERS, Stream ID: 1, Length 14, 200 OK
Length: 14
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... .... .... .... .... .... .... .... = Reserved: 0x0
.000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
[Pad Length: 0]
Header Block Fragment: 885f8b1d75d0620d263d4c4d6564
[Header Length: 54]
[Header Count: 2]
Header: :status: 200 OK
Header: content-type: application/grpc

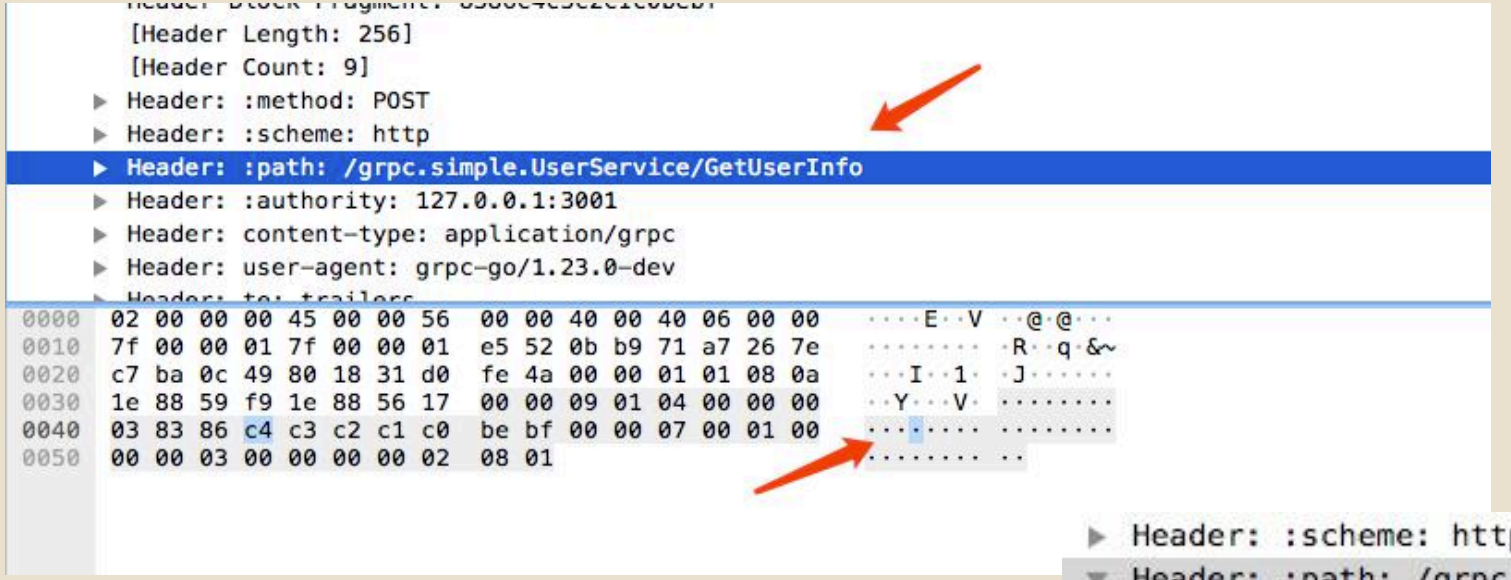

Stream: HEADERS, Stream ID: 1, Length 24
Length: 24
Type: HEADERS (1)
Flags: 0x05
.... ...1 = End Stream: True
.... .1.. = End Headers: True
.... 0... = Padded: False
..0. .... = Priority: False
00.0 ..0. = Unused: 0x00
0... .... .... .... .... .... .... .... = Reserved: 0x0
.000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
[1 Body fragment (21 bytes): #21(21)]
Data: 00000000100a0c4b656e2054686f6d70736f6e104b
[Pad Length: 0]
Header Block Fragment: 40889acac8b21234da8f013040899acac8b5254207317f00
[Header Length: 40]
[Header Count: 2]
Header: grpc-status: 0
Header: grpc-message:
GRPC Message: /grpc.simple.UserService/GetUserInfo, Response

# grpc resp error

```
var customStatusCode = 999
grpc.Errorf(customStatusCode, "not found user id")
```

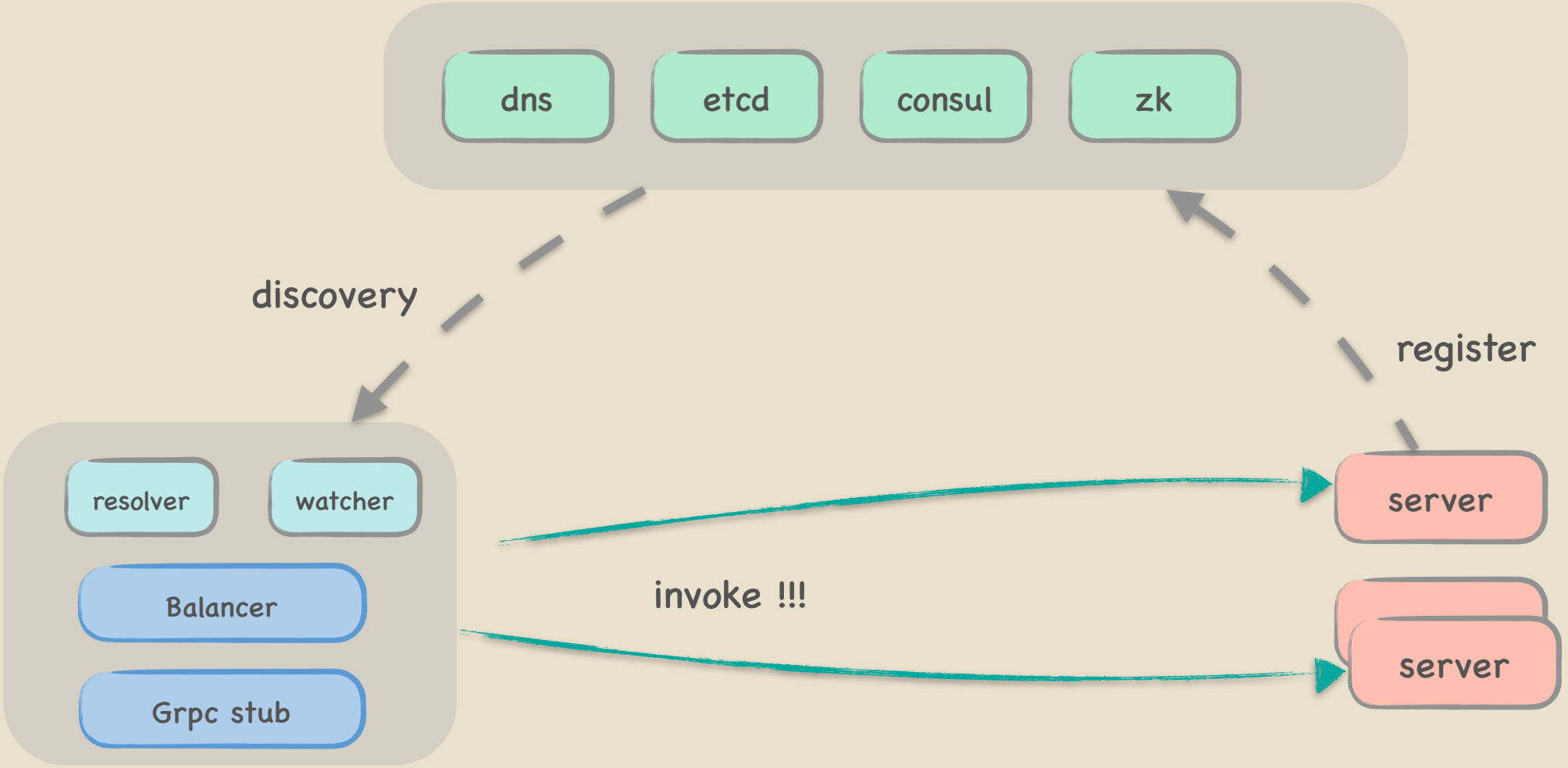
```
    [Pad Length: 0]
    Header Block Fragment: 88c77f05033939397f058ca8e95253db548a5a82d8a1a4
    [Header Length: 113]
    [Header Count: 4]
▶ Header: :status: 200 OK
▶ Header: content-type: application/grpc
▶ Header: grpc-status: 999
▶ Header: grpc-message: not found user id
```

# grpc hpack

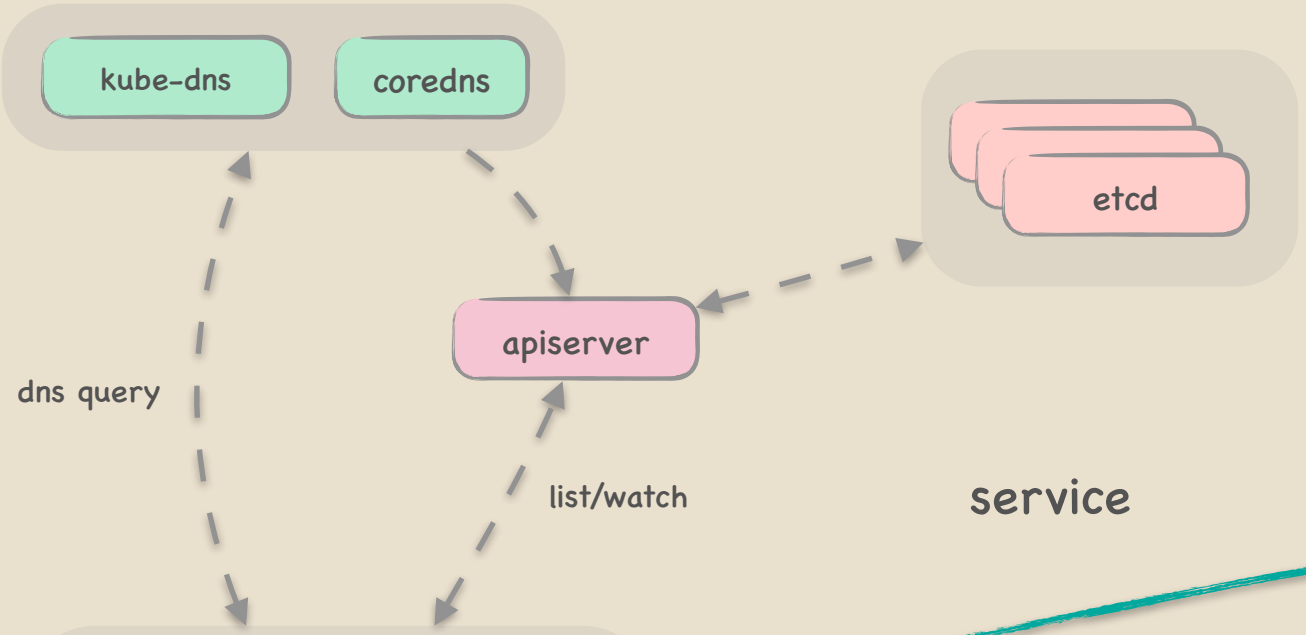

[Header Length: 256]
[Header Count: 9]
Header: :method: POST
Header: :scheme: http
Header: :path: /grpc.simple.UserService/GetUserInfo
Header: :authority: 127.0.0.1:3001
Header: content-type: application/grpc
Header: user-agent: grpc-go/1.23.0-dev
0000 02 00 00 00 45 00 00 56 00 00 40 00 40 06 00 00
0010 7f 00 00 01 7f 00 00 01 e5 52 0b b9 71 a7 26 7e
0020 c7 ba 0c 49 80 18 31 d0 fe 4a 00 00 01 01 08 0a
0030 1e 88 59 f9 1e 88 56 17 00 00 09 01 04 00 00 00
0040 03 83 86 c4 c3 c2 c1 c0 be bf 00 00 07 00 01 00
0050 00 00 03 00 00 00 00 02 08 01


Header: :scheme: http
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

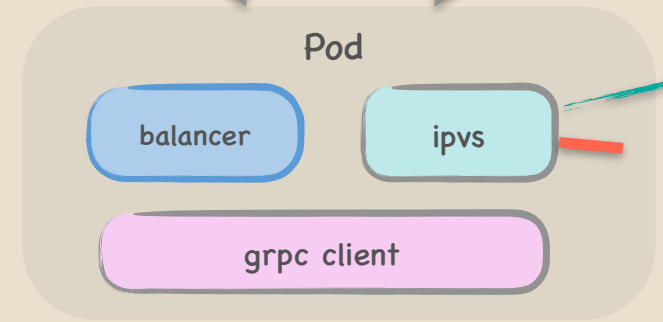
# grpc microservice

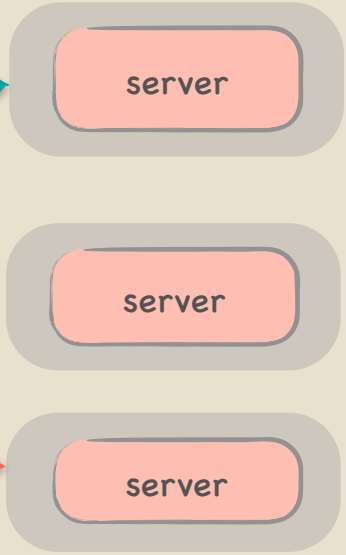


# grpc in k8s



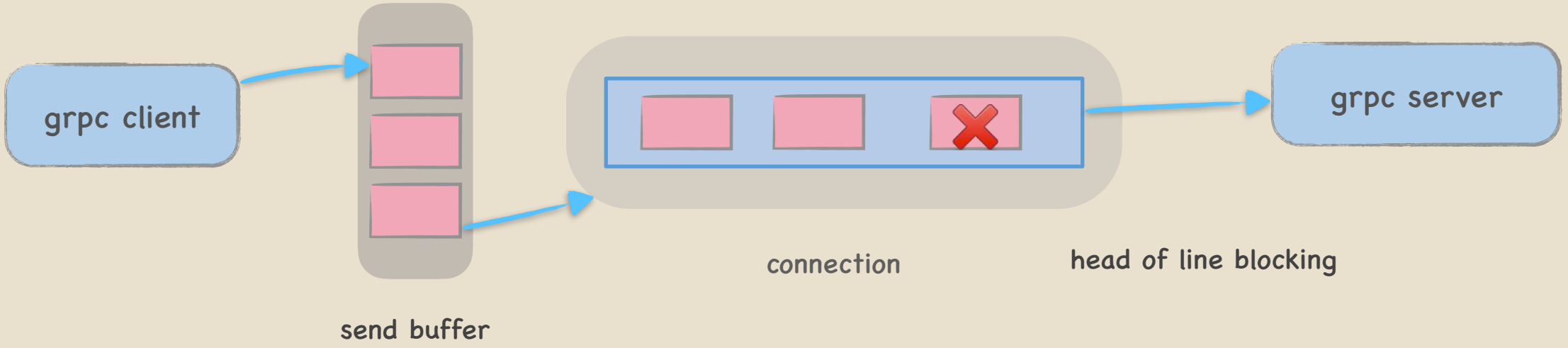

service




- internal balancer lib
  - use service headless
- list/watch apiserver
- if use service proxy
  - grpc client pool

# Q 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, 后进入慢启动。

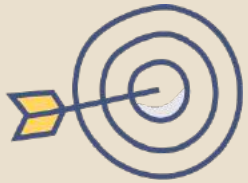
# Q futex syscall

pprof VIEW SAMPLE REFINE Search regexp

| Flat | Flat% | Sum% | Cum | Cum% | Name |
|---|---|---|---|---|---|
| 10.99s | 14.22% | 14.22% | 11.36s | 14.69% | syscall.Syscall |
| 9.06s | 11.72% | 25.93% | 9.06s | 11.72% | runtime.futex |
| 6.20s | 8.02% | 33.95% | 6.20s | 8.02% | runtime.epollwait |
| 2.12s | 2.74% | 36.70% | 8.06s | 10.43% | runtime.lock |
| 1.88s | 2.43% | 39.13% | 4.95s | 6.40% | runtime.scanobject |

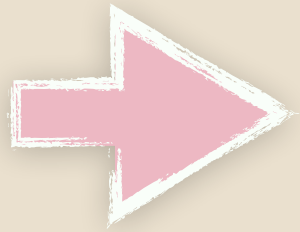
xiaorui.cc

```
strace: Process 89952 attached
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 94.46  965.933911         567   1704001    209908 futex
  1.94   19.849214          35    572598           nanosleep
  1.82   18.589932          77    242824           epoll_pwait
  0.90    9.162674     3054225         3         2 restart_syscall
  0.64    6.534568          12    546626           sched_yield
  0.20    2.032028          18    112756           write
  0.05    0.533621          19     28080       751 read
  0.00    0.000762          54        14           sigaltstack
  0.00    0.000644          31        21           rt_sigprocmask
```

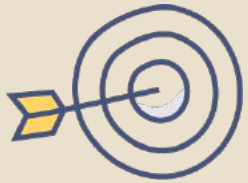


# optimize

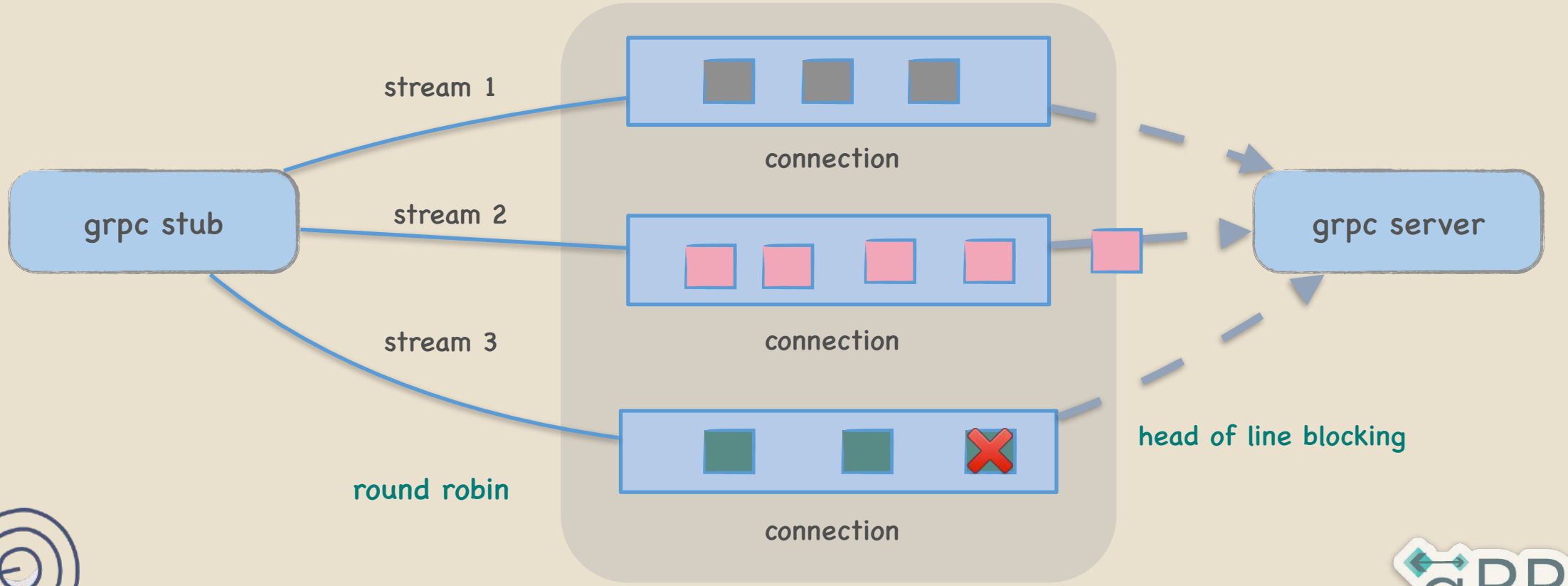
- tcp hol blocking
- single LoopyWriter
- lock race

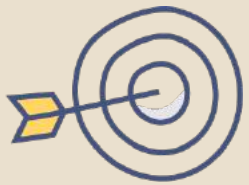


- open multi connction
  - grpc client pool !
  - grpc client pool !!
  - grpc client pool !!!
- bbr vs cubic
  - use bbr abroad

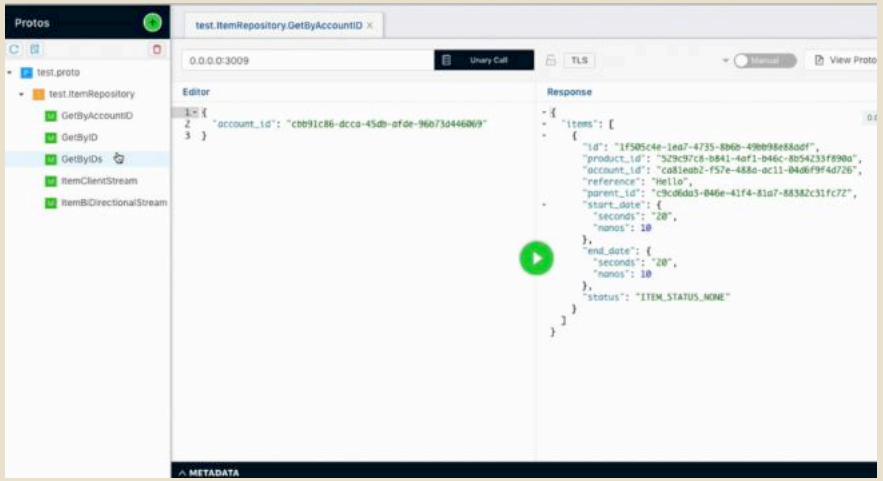


# grpc client pool

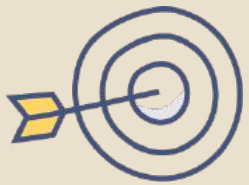


# easy test ?

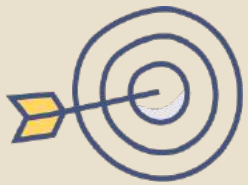



- benchmark
  - ghz
- like postman, curl
  - fullstorydev/grpcurl
  - uw-labs/bloomrpc
  - ktr0731/evans
- more



# rest to grpc

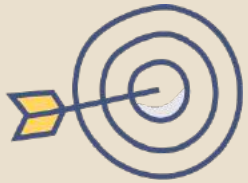




* grpc-gateway
* envoy
* custom code
* grpcall (only lib)

# grpc proxy

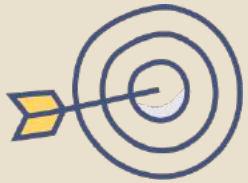


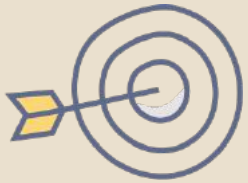




* nginx, kong, apisix
* envoy
* traefik
* tyk
* ...

# refer

- https://github.com/grpc/grpc-go
- https://tools.ietf.org/html/rfc7540
- https://github.com/rfyiamcool/grpcall
- https://http2.akamai.com/demo
- http://vearne.cc/archives/39117
- http://xiaorui.cc/archives/6117





# Q & A

- xiaorui.cc

- github.com/rfyiamcool