OTUS

Golang Developer. Professional

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Ставим “+”, если все хорошо
“-”, если есть проблемы

Тема вебинара

gRPC ч.1

Олег Венгер

Руководитель группы Защита профилей в Wildberries

Правила вебинара

Активно
участвуем

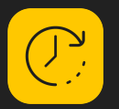
Off-topic обсуждаем
в учебной группе

Задаем вопрос
в чат или голосом

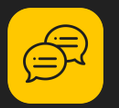
Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально



Время, необходимое
на активность



Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

О чем будем говорить

- Посмотрим на Protobuf
- Что такое gRPC

План занятия

- Знакомимся с Protocol buffers
- Прямая и обратная совместимость в Protocol buffers
- Описание API с помощью Protobuf
- Что такое gRPC и HTTP/2
- Генерация кода для GRPC клиента и сервера
- Реализация API

Protobuf

```
syntax = "proto3";

message SearchRequest {
  string query = 1;
  int32 page_number = 2;
  int32 results_per_page = 3;
}
```

Protobuf: tags

- Номера тегов уникальны в рамках сообщения
- Номера от `1` до `536,870,911`
- Номера от `19,000` до `19,999` заразервированы компилятором
- 1 - 15 занимают 1 байт

Protocol buffers: типы данных

скаляры:

- double (float64)
- float (float32)
- bool (bool)
- string (string) UTF-8 / 7-bit ASCII (max 2^32)
- bytes ([]byte)
- fixed{32,64} (uint32/64 эффективен для больших чисел)
- int{32,64} (отрицательные значения - 10 байт)
- uint{32,64}
- sint{32,64} (ZigZag для отрицательных значений)

https://developers.google.com/protocol-buffers/docs/encoding

Protocol buffers: repeated fields

Слайс реализуется через repeated:

```
message SearchResponse {
  repeated Result results = 1;
}

message Result {
  string url = 1;
  string title = 2;
  repeated string snippets = 3;
}
```

```
...
Snippets             []string `protobuf:"bytes,3,rep,name=snippets,proto3" json:"snippets,omitempty"`
...
Results              []*Result `protobuf:"bytes,1,rep,name=results,proto3" json:"results,omitempty"`
```

Protocol buffers: дефолтные значения

- string: пустая строка
- number (int32/64 etc.): 0
- bytes: пустой слайс
- enum: первое значение
- repeated: пустой слайс
- Message - зависит от языка (https://developers.google.com/protocol-buffers/docs/reference/go-generated#singular-message) в Go - nil

Protocol buffers: Enums

```
enum EyeColor {
  EYE_COLOR_UNSPECIFIED = 0;
  EYE_COLOR_GREEN = 1;
  EYE_COLOR_BLUE = 2;
}

message Person {
  string name = 1;
  EyeColor eye_color = 2;
}
```

```
type EyeColor int32

const (
  EyeColor_UNSPECIFIED EyeColor = 0
  EyeColor_EYE_GREEN   EyeColor = 1
  EyeColor_EYE_BLUE    EyeColor = 2
)
```

Protocol buffers: oneof, map

`oneof` - только одно поле из списка может иметь значение и не может быть repeated.

```
message Message {
  int32 id = 1;
  oneof auth {
    string mobile = 2;
    string email = 3;
    int32 userid = 4;
  }
}
```

`map` - ассоциативный массив; ключи - скаляры (кроме float/double); значения - любые типы, не может быть repeated.

```
message Result {
  string result = 1;
}

message SearchResponse {
  map<string, Result> results = 1;
}
```

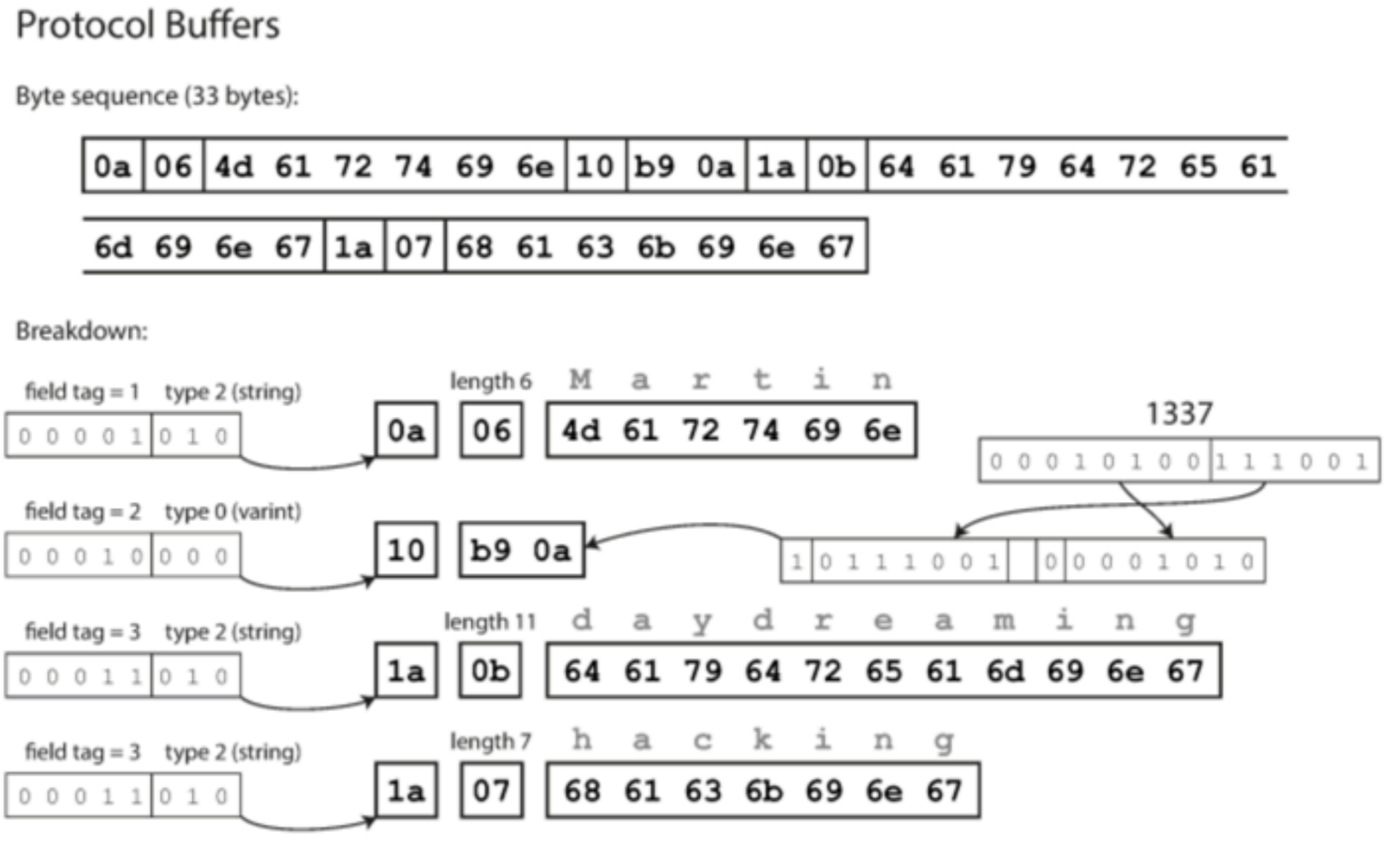
Protocol buffers: wire types

ID	Name	Used For
0	VARINT	int32, int64, uint32, uint64, sint32, sint64, bool, enum
1	I64	fixed64, sfixed64, double
2	LEN	string, bytes, embedded messages, packed repeated fields
3	SGROUP	group start (deprecated)
4	EGROUP	group end (deprecated)
5	I32	fixed32, sfixed32, float

Protocol buffers: encoding format

```
message Person {
  required string user_name = 1;
  optional int64 favorite_number = 2;
  repeated string interests = 3;
}
```

Protocol buffers: encoding format



Где `required` и `optional`?

Protocol buffers: go_package

simplepb - более явно

```
syntax = "proto3";

package example.simple;

option go_package = "./;simplepb";

message SimpleMessage {
  int32 id = 1;
  bool is_simple = 2;
  string name = 3;
  repeated int32 sample_list = 4;
}
```

```
// Code generated by protoc-gen-go. DO NOT EDIT.
// source: simple/simple.proto

package simplepb

import proto "github.com/golang/protobuf/proto"
import fmt "fmt"
import math "math"
```

Protocol buffers: Well Known Types

https://developers.google.com/protocol-buffers/docs/reference/google.protobuf

```
syntax = "proto3";

import "google/protobuf/timestamp.proto";
import "google/protobuf/duration.proto";


message MyMessage {
  google.protobuf.Timestamp last_online = 1;
  google.protobuf.Duration session_length = 2;
}
```

Protocol buffers: прямая/обратная совместимость

- Scenario 1:

Forward compatible change

- Scenario 2:

Backward compatible change

Protocol buffers: прямая/обратная совместимость

- не меняйте теги
- старый код будет игнорировать новые поля
- при неизвестных полях испольуются дефолтные значения
- поля можно удалять, но не переиспользовать тег / сделать поле reserved

https://developers.google.com/protocol-buffers/docs/proto#updating

Protocol buffers: прямая/обратная совместимость

Добавление полей:

```
message MyMessage {
  int32 id = 1;
  string fist_name = 2;  // + добавим
}
```

- старый код будет игнорировать новое поле
- новый код будет использовать значение по умолчанию при чтении "старых" данных

Переименоваение полей:

```
message MyMessage {
  int32 id = 1;
  //string fist_name = 2;  удалим
  string person_first_name = 2;  // добавим
}
```

- бинарное представление не меняется, так как имеет значение только тег

Protocol buffers: прямая/обратная совместимость

reserved:

```
message Foo {
  reserved 2, 15, 9 to 11;
  reserved "foo", "bar";
}
```

- можно резервировать теги и поля
- смешивать нельзя
- резервируем теги чтобы новые поля их не переиспользовали (runtime errors)
- резервируем имена полей, чтобы избежать багов

никогда не удаляйте зарезервированные теги

Protocol buffers: дефолтные значения

- не можем отличить отсутствующее поле от пустого
- убедитесь, что с тз бизнес логики дефолтные значения бессмысленны

```
func (m *Course) GetTitle() string {
  if m != nil {
    return m.Title
  }
  return ""
}
```

Protocol buffers: style guide

https://developers.google.com/protocol-buffers/docs/style

- строка 80, отступ 2
- файлы lower_snake_case.proto
- сообщения CamelCase, поля - underscore_separated_names
- CAPITALS_WITH_UNDERSCORES для enums

```
message SongServerRequest {
  required string song_name = 1;
}

enum Foo {
  FOO_UNSPECIFIED = 0;
  FOO_FIRST_VALUE = 1;
  FOO_SECOND_VALUE = 2;
}
```

Прежде, чем начать

1) Обновляем protoc
https://github.com/protocolbuffers/protobuf/releases

2) Обновляем `protoc-gen-go` и `protoc-gen-go-grpc`

```
go install google.golang.org/protobuf/cmd/protoc-gen-go@latest
go install google.golang.org/grpc/cmd/protoc-gen-go-grpc@latest
```

Что такое gRPC

RPC - Remote Procedure Call

RPC:
- сетевые вызовы абстрагированы от кода
- интерфейсы как сигнатуры функций (Interface Definition Language)
- тулзы для кодогенерации
- кастомные протоколы

g:
https://github.com/grpc/grpc/blob/master/doc/g_stands_for.md

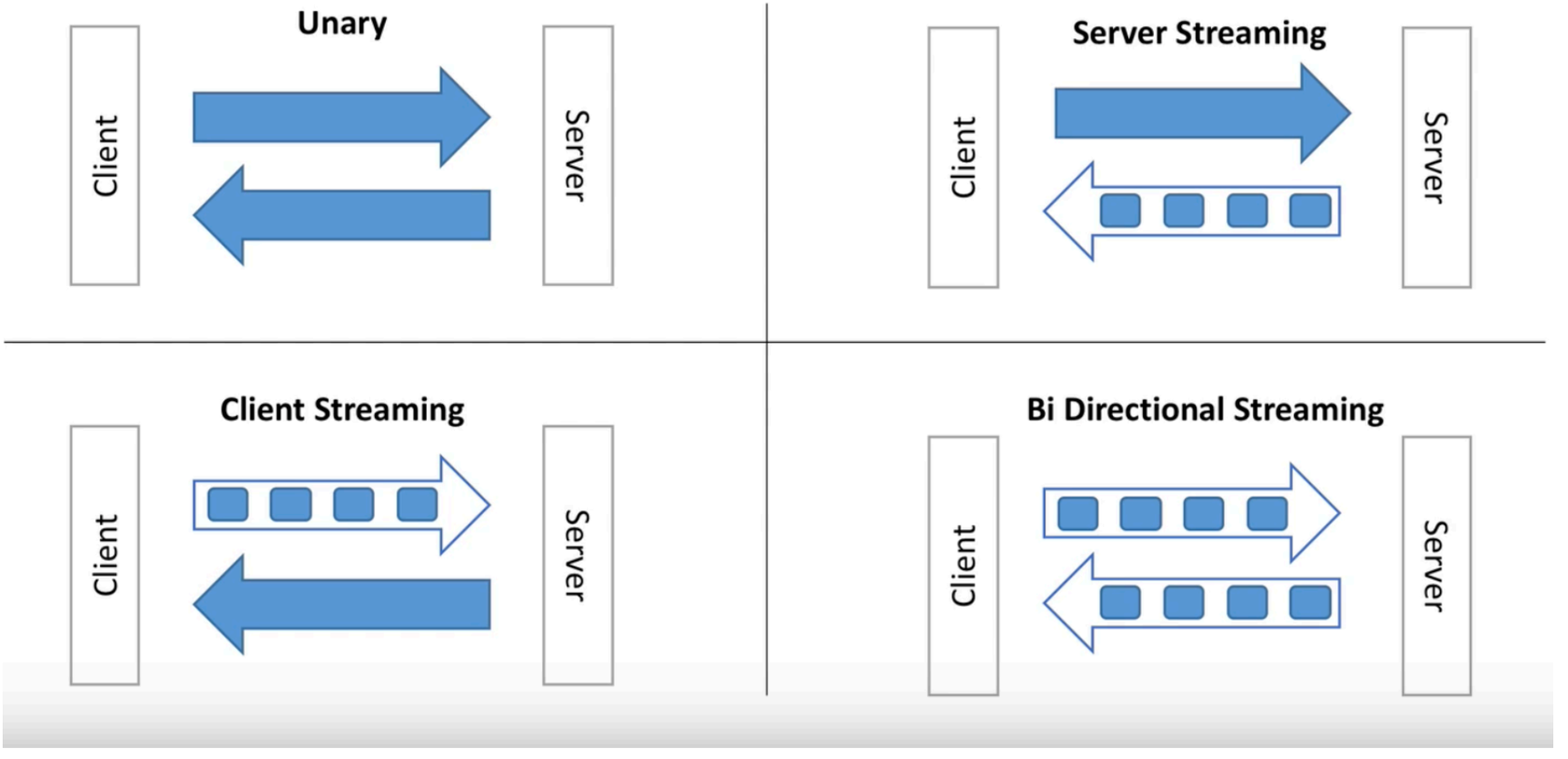
Что такое gRPC

- Фреймворк для разработки масштабируемых, современных и быстрых API
- Для описания интерфейса используется Protocol Buffers, где описывается структура для передачи, кодирования и обмена данных между клиентом и сервером
- В качестве транспорта использует HTTP2

Описание сервиса в gRPC

```
syntax = "proto3";

package search;
option go_package = "./;searchpb";

service Google {
  // Search returns a Google search result for the query.
  rpc Search(Request) returns (Result) {
  }
}

message Request {
  string query = 1;
}

message Result {
  string title = 1;
  string url = 2;
  string snippet = 3;
}
```

Типы gRPC API

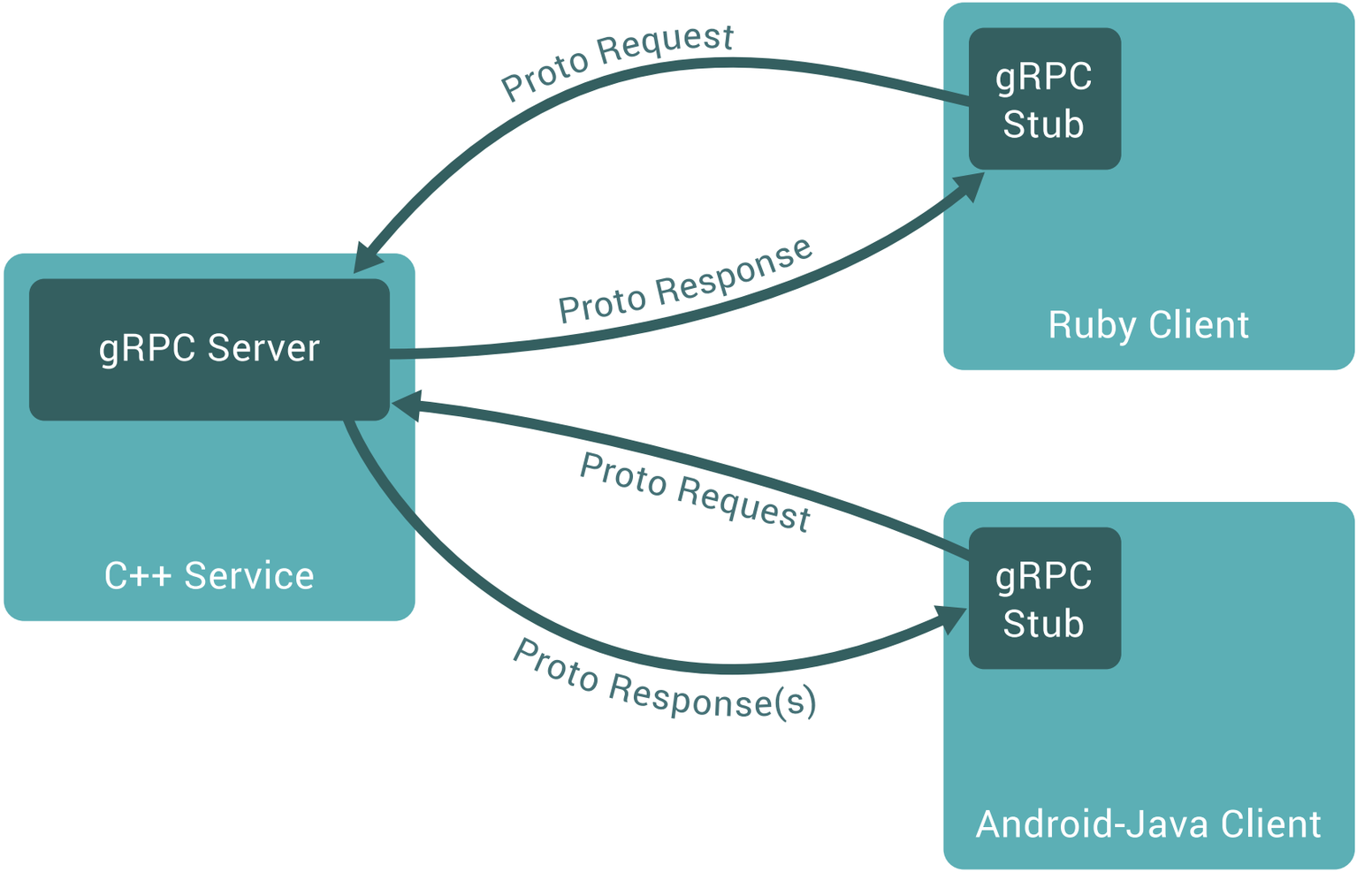


Кодогенерация gRPC

```
protoc search.proto --go_out=. --go-grpc_out=.
```

```
package searchpb;

type GoogleClient interface {
    // Search returns a Google search result for the query.
    Search(ctx context.Context, in *Request, opts ...grpc.CallOption) (*Result, error)
}

type GoogleServer interface {
    // Search returns a Google search result for the query.
    Search(context.Context, *Request) (*Result, error)

}

type Request struct {
    Query string `protobuf:"bytes,1,opt,name=query" json:"query,omitempty"`
}

type Result struct {
    Title   string `protobuf:"bytes,1,opt,name=title" json:"title,omitempty"`
    Url     string `protobuf:"bytes,2,opt,name=url" json:"url,omitempty"`
    Snippet string `protobuf:"bytes,3,opt,name=snippet" json:"snippet,omitempty"`
}
```

Независимая разработка с gRPC



gRPC: где использовать

- микросервисы
- клиент-сервер
- интеграции / API

- Apcera/Kurma: container OS
- Bazil: distributed file system
- CoreOS/Etcd: distributed consistent key-value store
- Google Cloud Bigtable: sparse table storage
- Monetas/Bitmessage: transaction platform
- Pachyderm: containerized data analytics
- YouTube/Vitess: storage platform for scaling MySQL

gRPC vs REST

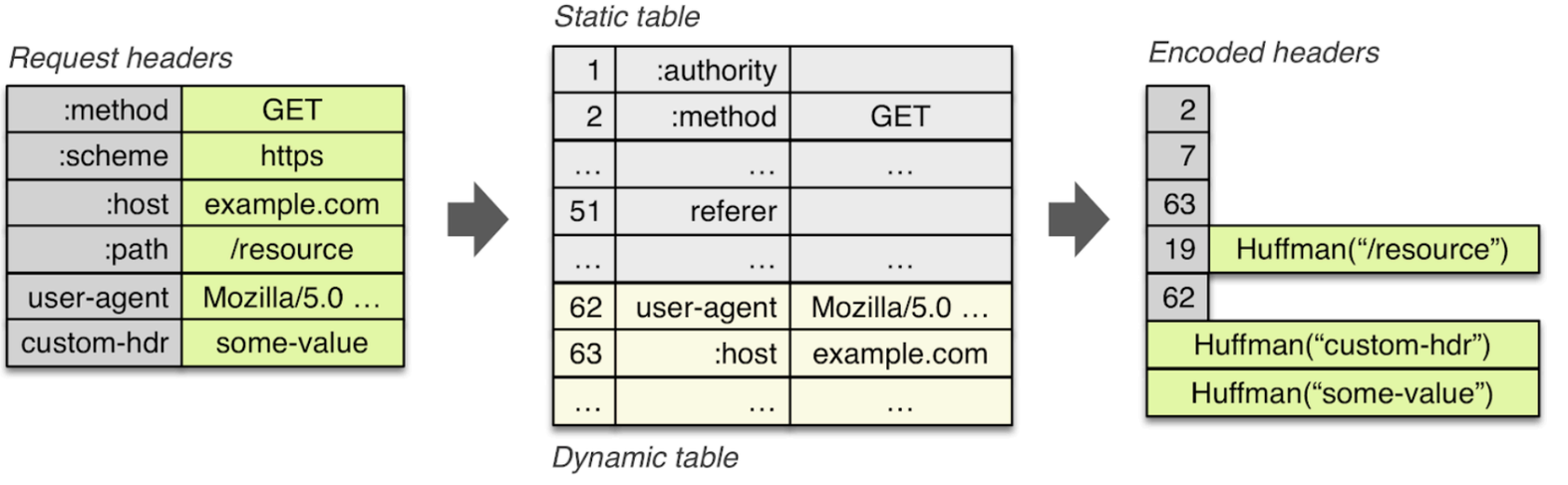
gRPC	REST
Protobuffers - smaller, faster	JSON - test based, slower, larger size
HTTP/2 (lower latency)	HTTP1.1 (higher latency)
Bi-directional & async	Client->Server requests only
Stream support	Request/Response mechanism only
API Oriented, no contraints	CRUD Oriented
Code generation through protobuffers	Code generation to third party tools Swagger/OpenAPI
RPC Based - Call functions on the server- gRPC does the plumbing	HTTP verbs based - have to write plumbing

HTTP/2 vs HTTP

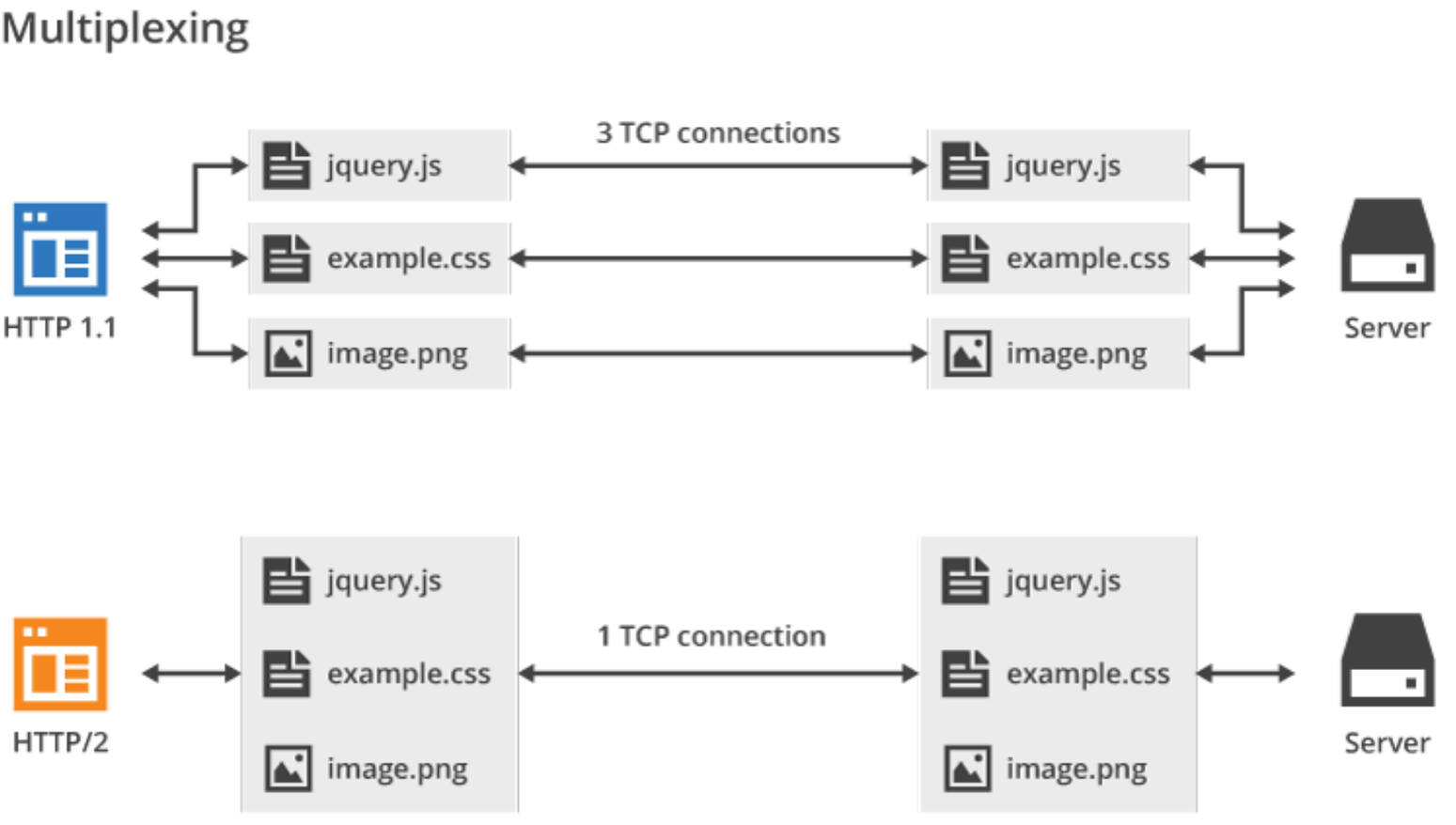
- https://imagekit.io/demo/http2-vs-http1
- https://developers.google.com/web/fundamentals/performance/http2/

HTTP/2 vs HTTP: header compression

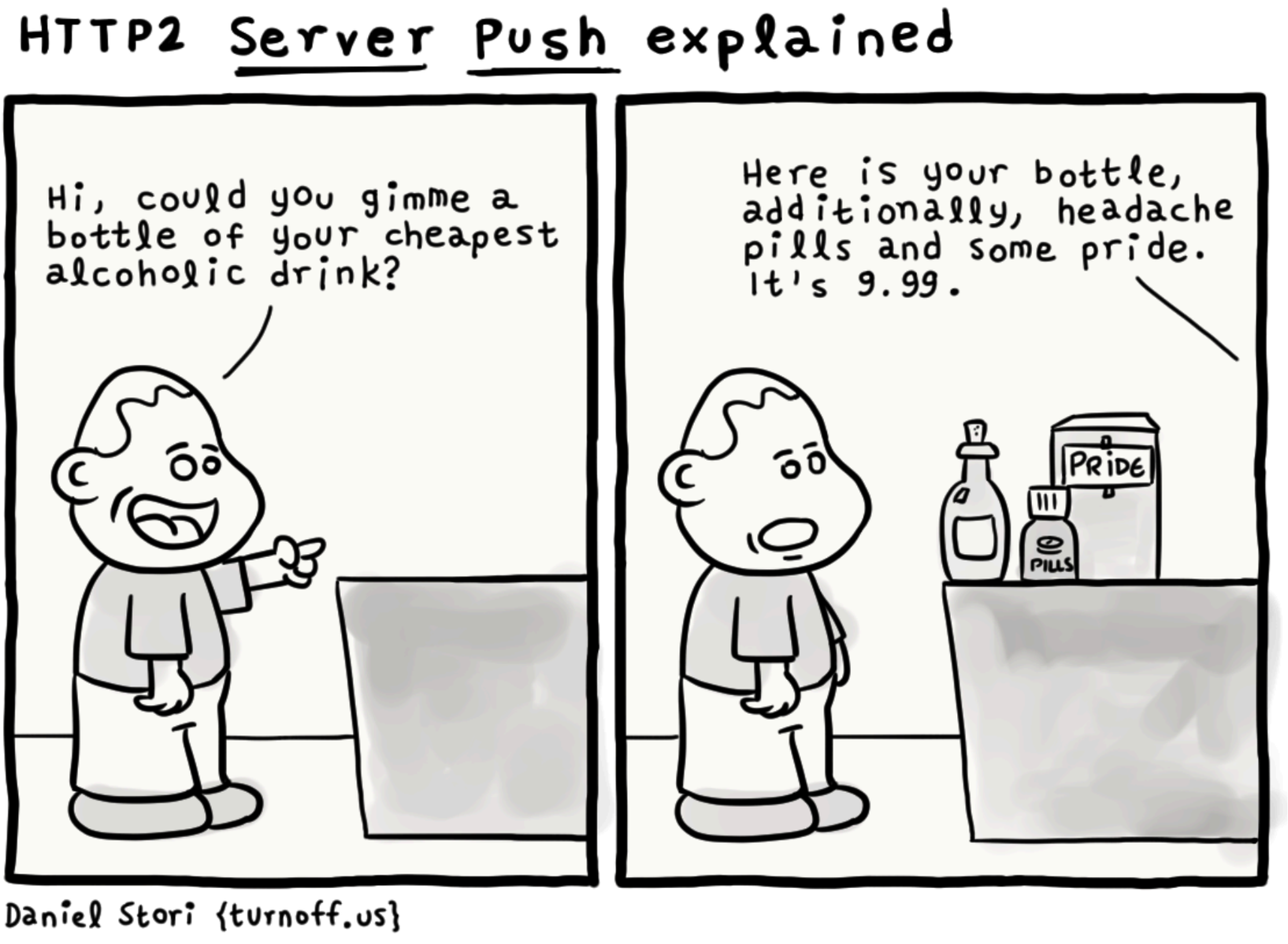
HPACK header compression



HTTP/2 vs HTTP: multiplexing



HTTP/2 vs HTTP: server push



HTTP/2 vs HTTP

- бинарный вместо текстового
- мультиплексирование — передача нескольких асинхронных HTTP-запросов по одному TCP-соединению
- сжатие заголовков методом HPACK
- Server Push — несколько ответов на один запрос
- приоритизация запросов (https://habr.com/ru/post/452020/)

https://medium.com/@factoryhr/http-2-the-difference-between-http-1-1-benefits-and-how-to-use-it-38094fa0e95b

Примеры

https://github.com/OtusGolang/webinars_practical_part/tree/master/27-grpc

gRPC: Errors

- https://grpc.io/docs/guides/error/
- https://grpc.io/docs/guides/error/#protocol-errors
- https://godoc.org/google.golang.org/grpc/codes
- https://godoc.org/google.golang.org/grpc/status
- https://jbrandhorst.com/post/grpc-errors/
- http://avi.im/grpc-errors/

```
// Server
func (*server) SquareRoot(ctx context.Context, req *calculatorpb.SquareRootRequest)
    (*calculatorpb.SquareRootResponse, error) {

    number := req.GetNumber()
    if number < 0 {
        return nil, status.Errorf(codes.InvalidArgument,
          "received a negative number: %v", number)
    }

    return &calculatorpb.SquareRootResponse{
      NumberRoot: math.Sqrt(float64(number)),
    }, nil
}
```

gRPC: Errors

```
// Client
res, err := c.SquareRoot(
    context.Background()
    &calculatorpb.SquareRootRequest{Number: n},
)
if err != nil {
    respErr, ok := status.FromError(err)
    if ok {
        // actual error from gRPC (user error)
        fmt.Printf("Error message from server: %v\n", respErr.Message())
        fmt.Println(respErr.Code())
        if respErr.Code() == codes.InvalidArgument {
            fmt.Println("We probably sent a negative number!")
            return
        }
    } else {
        return
    }
}
```

gRPC: Security (SSL/TLS)

- https://bbengfort.github.io/programmer/2017/03/03/secure-grpc.html
- https://medium.com/@gustavoh/building-microservices-in-go-and-python-using-grpc-and-tls-ssl-authentication-cfcee7c2b052

gRPC: tools

Для разработки:

- https://github.com/bufbuild/buf

Для отладки:

- https://github.com/ktr0731/evans
- https://github.com/fullstorydev/grpcurl
- https://github.com/uw-labs/bloomrpc

gRPC: transcoding

- https://github.com/grpc-ecosystem/grpc-gateway

На занятии

- Узнали, что такое gRPC
- Научились писать Protobuf схемы
- Научились писать gRPC сервисы

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Следующий вебинар

21 мая

GRPC ч.2

Ссылка на вебинар будет в ЛК за 15 минут

Материалы к занятию в ЛК — можно изучать

Обязательный материал обозначен красной лентой

Спасибо за внимание!

Приходите на следующие вебинары

Олег Венгер

Руководитель группы Защита профилей в Wildberries