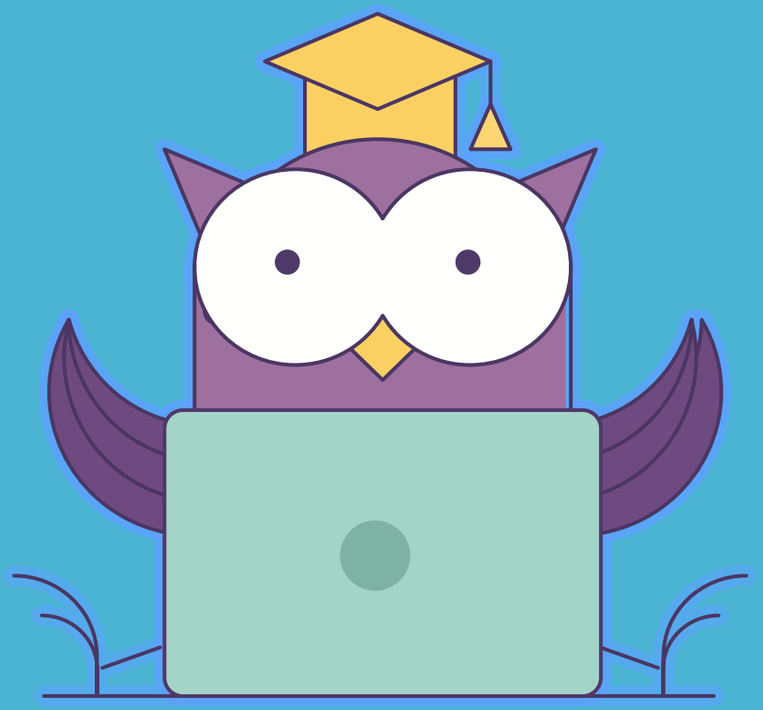
ОНЛАЙН-ОБРАЗОВАНИЕ

gRPC

Alexander Davydov

Как меня слышно и видно?



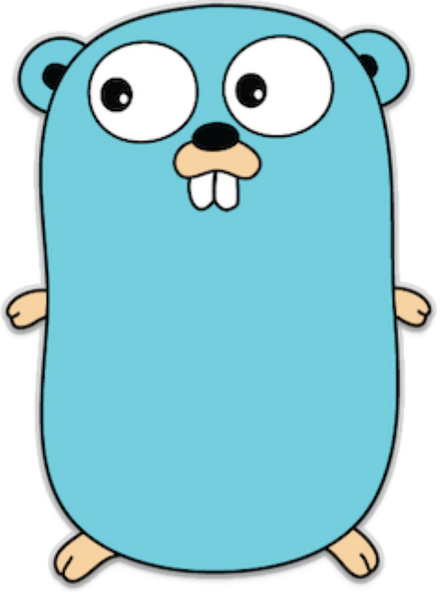
> Напишите в чат

+ если все хорошо

– если есть проблемы со звуком или с видео

Цель занятия

- Научиться писать обратно совместимые схемы в Protobuf
- Научиться писать gRPC сервисы
- Получить представление о Clean Architecture



План занятия

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

Что такое gRPC

RPC: (CORBA, Sun RPC, DCOM etc.)

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

```
try {
   XmlRpcClient client = new XmlRpcClient("http://localhost/RPC2");
   Vector params = new Vector();

   params.addElement(new Integer(17));
   params.addElement(new Integer(13));

   Object result = server.execute("sample.sum", params);

   int sum = ((Integer) result).intValue();

} catch (Exception exception) {
   System.err.println("JavaClient: " + exception);
}
```

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

Что такое gRPC

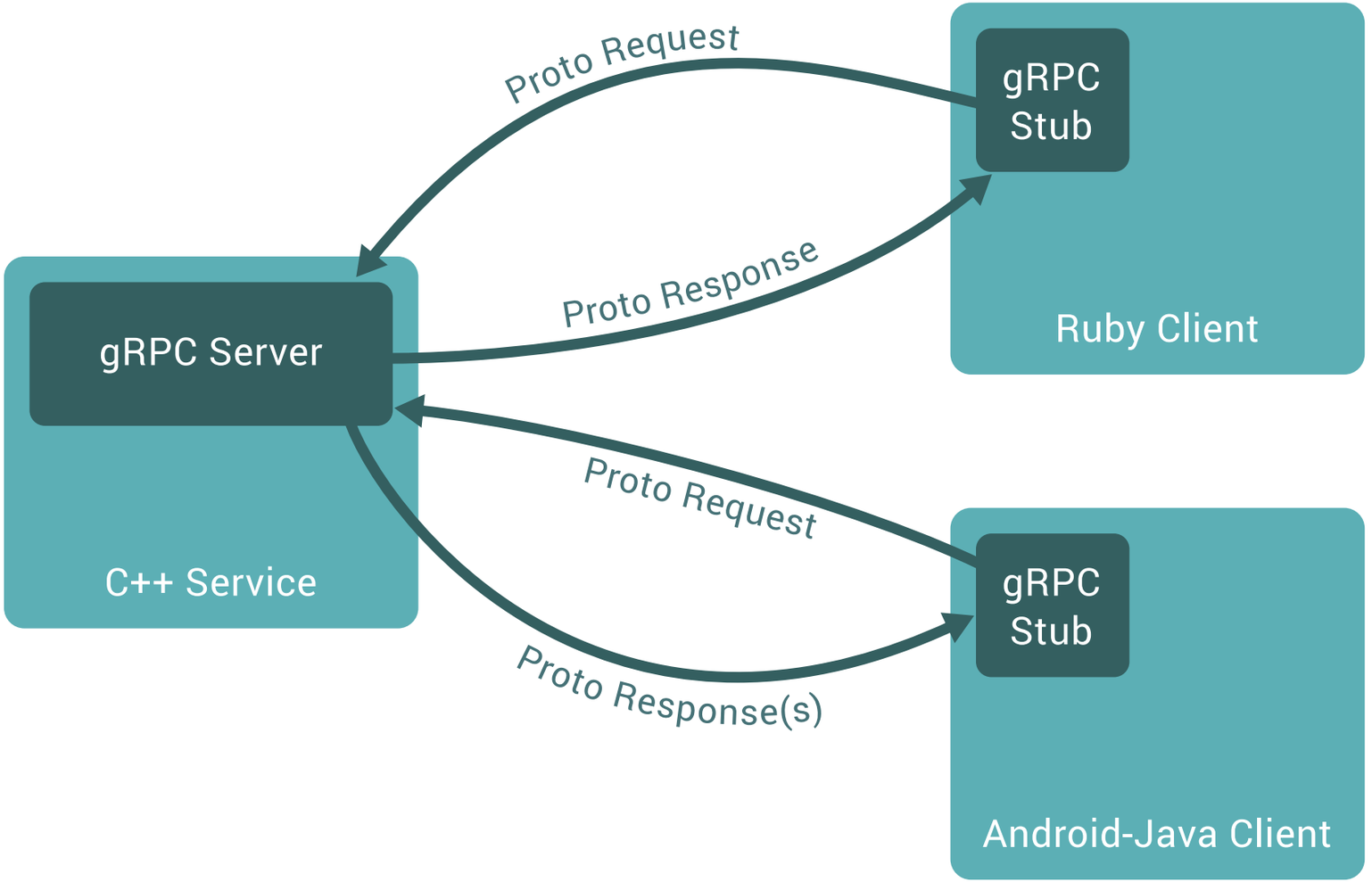
```
syntax = "proto3";

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;
}
```

Что такое gRPC

```
protoc ./search.proto --go_out=plugins=grpc:.
```

```
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"`
}
```

Что такое 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

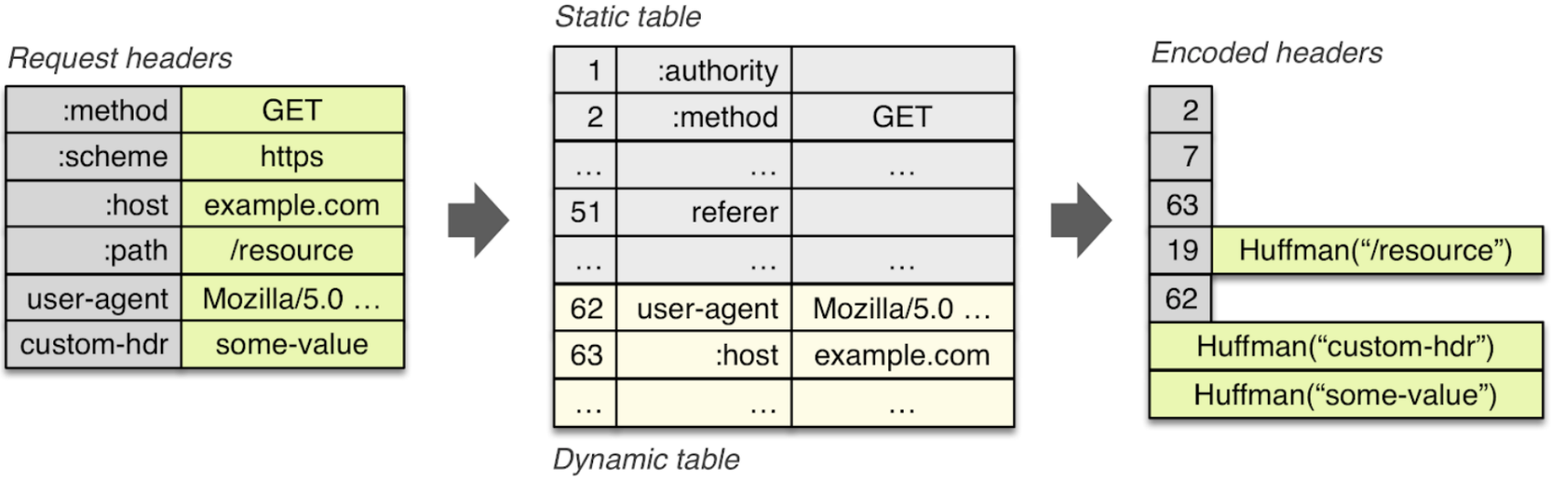
gRPC vs REST

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

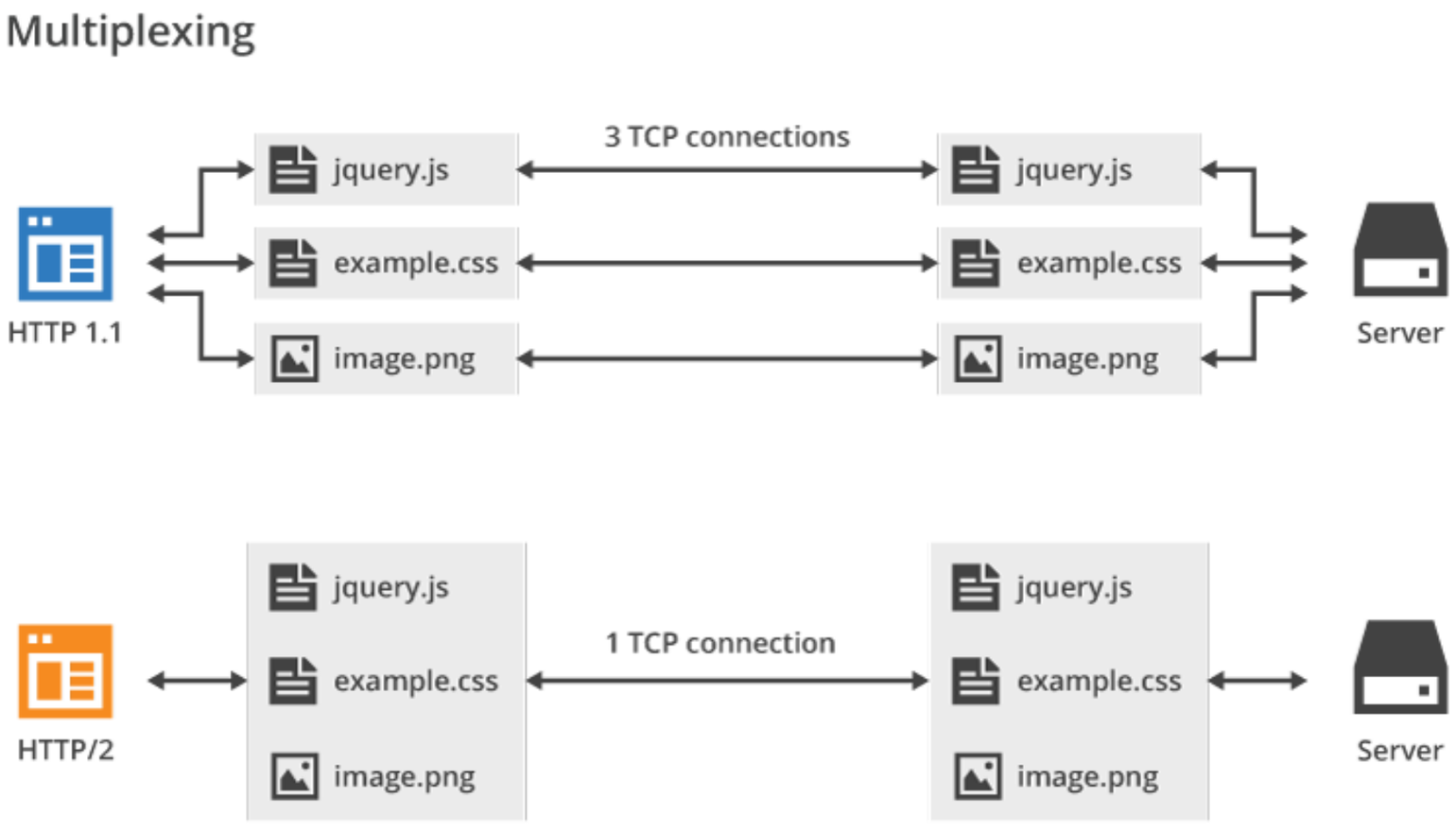
HTTP/2 vs HTTP

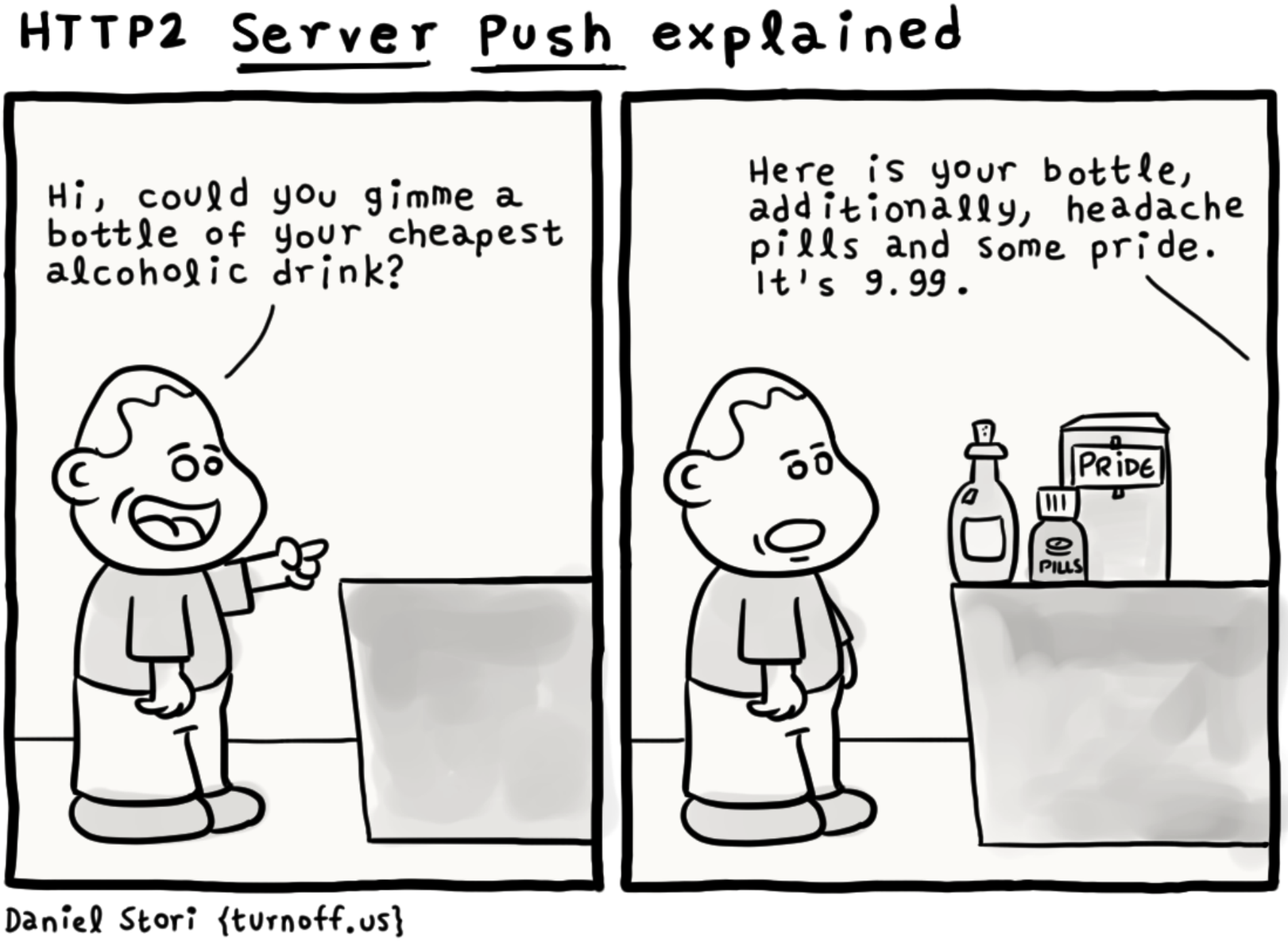
https://imagekit.io/demo/http2-vs-http1 https://developers.google.com/web/fundamentals/performance/http2/

HPACK header compression



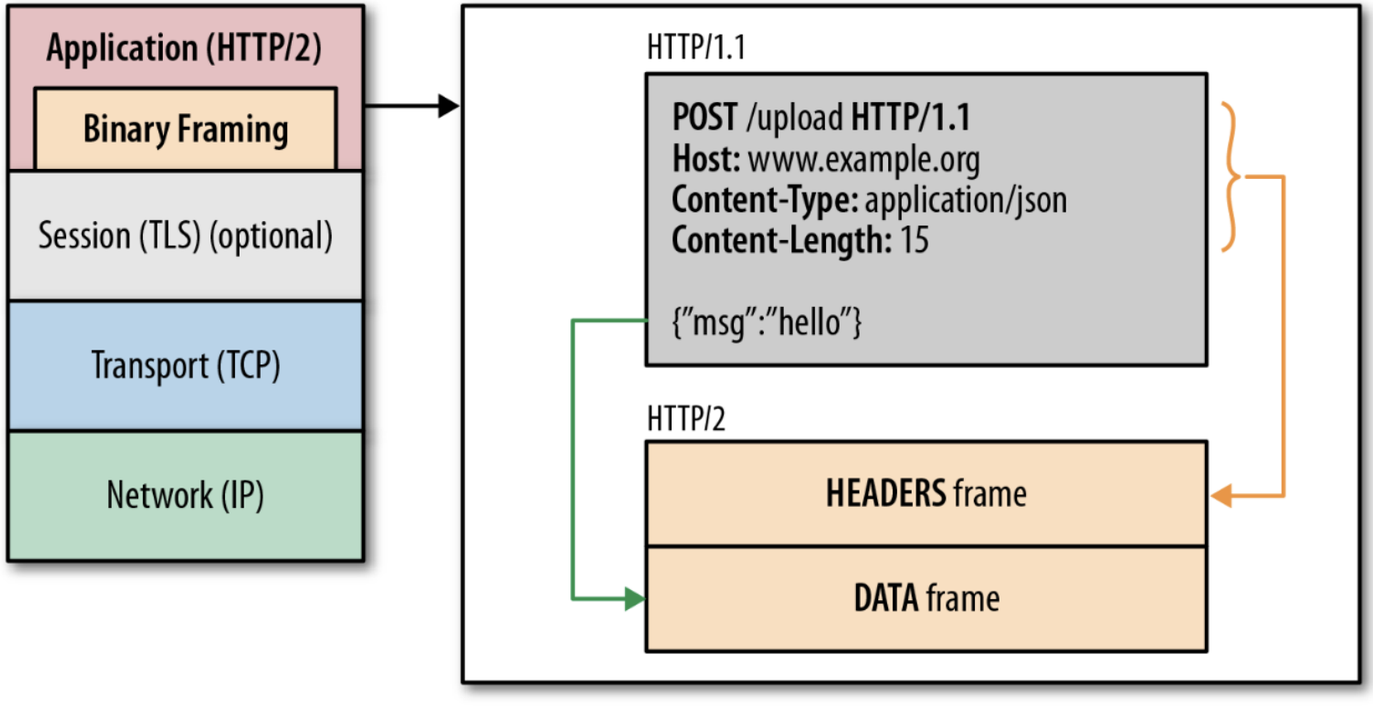
HTTP/2 vs HTTP: multiplexing




HTTP2 Server Push explained
Hi, could you gimme a bottle of your cheapest alcoholic drink?
Here is your bottle, additionally, headache pills and some pride. It's 9.99.
PRIDE
PILLS
Daniel Stori {turnoff.us}

HTTP/2 in one slide…

1. **One TCP connection**
2. **Request → Stream**
   - Streams are multiplexed
   - Streams are prioritized
3. **Binary framing layer**
   - Prioritization
   - Flow control
   - Server push
4. **Header compression (HPACK)**

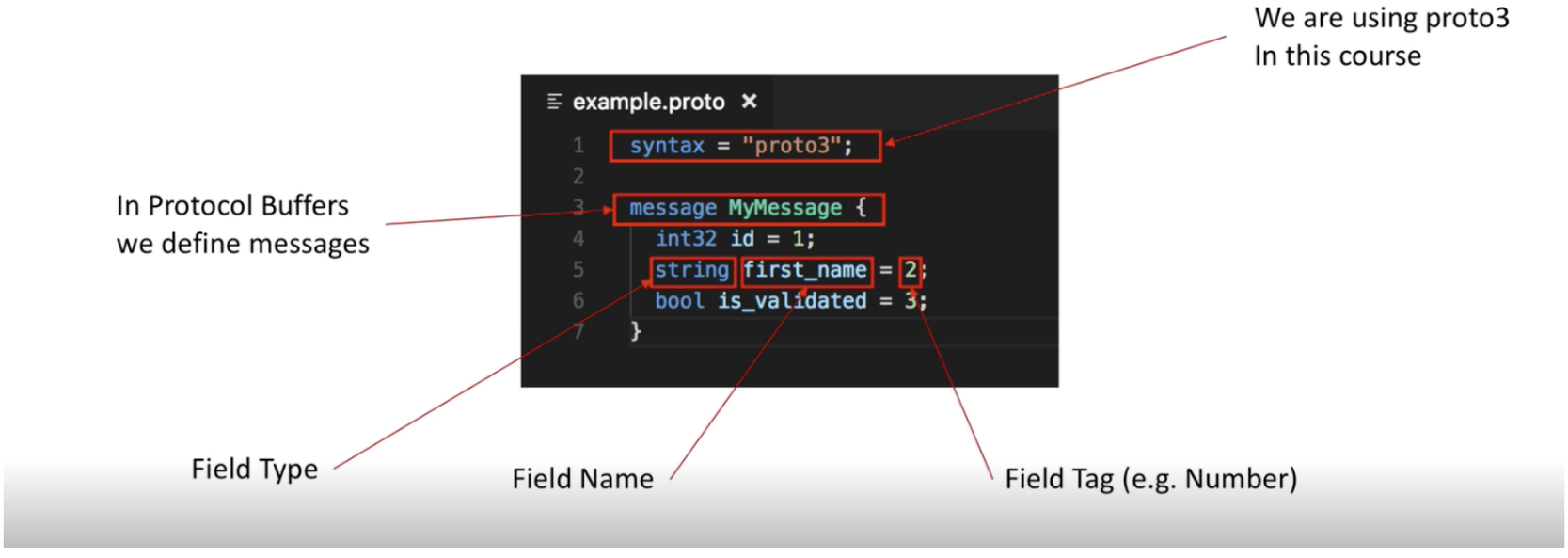


HTTP/2 vs HTTP

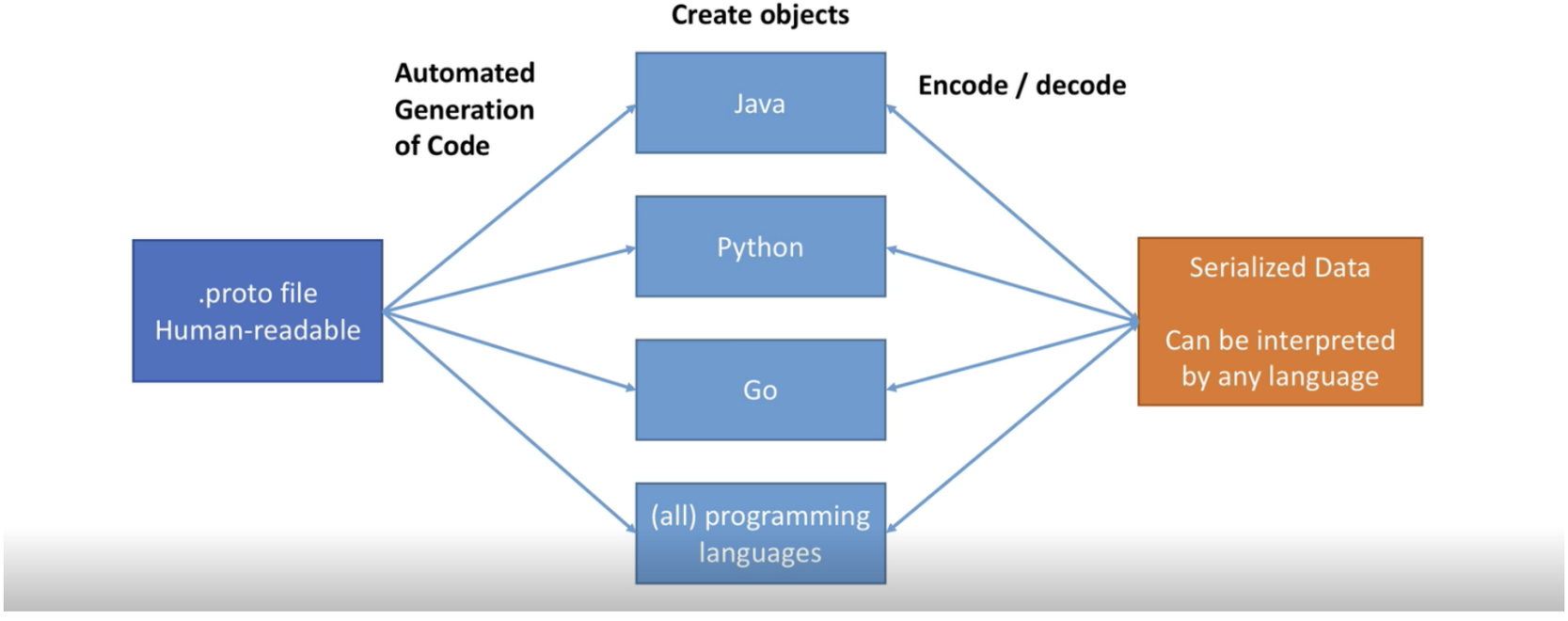
- бинарный вместо текстового
- мультиплексирование — передача нескольких асинхронных HTTP-запросов по одному TCP-соединению
- сжатие заголовков методом HPACK
- Server Push — несколько ответов на один запрос
- приоритизация запросов (https://habr.com/ru/post/452020/)

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

Protocol buffers: краткое содержание предыдущих серий



Protocol buffers: краткое содержание предыдущих серий



Protocol buffers: типы данных

скаляры:

- double (float64)
- float (float32)
- bool (bool)
- string (string) UTF-8 / 7-bit ASCII
- bytes ([]byte)
- 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: wire types

Type	Meaning	Used For
0	Varint	int32, int64, uint32, uint64, sint32, sint64, bool, enum
1	64-bit	fixed64, sfixed64, double
2	Length-delimited	string, bytes, embedded messages, packed repeated fields
3	Start group	groups (deprecated)
4	End group	groups (deprecated)
5	32-bit	fixed32, sfixed32, float

Protocol buffers: тэги

- 1 - 2^29 (536,870,911)
- 19000 - 19999 зарезервированы для имплементации Protocol Buffers
- 1-15 занимают 1 байт, используем для часто используемых полей

Protocol buffers: repeated fields

массив 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: комментарии

```
/* Подробное описание
* результата поиска */
message Result {
  string url = 1;
  // название страницы
  string title = 2;
  repeated string snippets = 3; // фрагменты страницы
}
```

Protocol buffers: дефолтные значения

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

Protocol buffers: Enums

```
enum EyeColor {
    UNKNOWN_EYE_COLOR = 0;
    EYE_GREEN = 1;
    EYE_BLUE = 2;
}
message Person {
    string name = 1;
    repeated string phone_numbers = 2;
    EyeColor eye_color = 3;
}
```

```
type EyeColor int32

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

Protocol buffers: несколько сообщений в одном файле

```
message Person {
    string name = 1;
    Date birthday = 2;
}

message Date {
    int32 year = 1;
    int32 month = 2;
    int32 day = 3;
}
```

```
message Person {
    string name = 1;
    Date birthday = 2;

    message Address {
        string street = 1;
        string city = 2;
        string country = 3;
    }

    Address address = 3;
}
```

Protocol buffers: импорты

date.proto:

```
message BirthDate {
    int32 year = 1;
    int32 month = 2;
    int32 day = 3;
}
```

person.proto:

```
import "date.proto";

message Person {
    string name = 1;
    BirthDate birthday = 2;

    message Address {
        string street = 1;
        string city = 2;
        string country = 3;
    }

    Address address = 3;
}
```

Protocol buffers: пакеты

mydate.proto:

```
syntax = "proto3";

package my.date;

message Date {
    int32 year = 1;
    int32 month = 2;
    int32 day = 3;
}
```

person.proto:

```
syntax = "proto3";

import "date.proto";
import "mydate.proto";

message Person {
    string name = 1;
    BirthDate birthday = 2;
    my.date.Date last_seen = 4;
}
```

Protocol buffers: упражнение

Написать person.proto: имя, фамилия, адрес, рост, вес, возраст

Protocol buffers: go_package

simplepb - более эксплицитно

```
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: 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: - acc. 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: 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: запись на диск, JSON

```
course := &myotus.Course{
    Title:   "Golang",
    Teacher: []*myotus.Teacher{{Name: "Dmitry Smal", Id: 1}, {Name: "Alexander Davydov", Id: 2}},
}
out, err := proto.Marshal(course)
```

```
import "github.com/gogo/protobuf/jsonpb"

marshaler := jsonpb.Marshaler{}
res, err := marshaler.MarshalToString(course)
print(res)
```

```
{"title":"Golang","teacher":[{"name":"Dmitry Smal","id":1},
                             {"name":"Alexander Davydov","id":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 (TODO!)
- поля можно удалять, но не переиспользовать тег / добавить префик OBSOLETE_ / сделать поле reserved

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

Protocol buffers: прямая/обратная совместимость

Добавление полей:

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

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

Переименоваение полей:

```
message MyMessage {
    int32 id = 1;
    - fist_name = 2;
    + person_first_name = 2;
}
```

- бинарное представление не меняется, тк 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 ""
}
```

enum'ы тоже можно добавлять, удалять и резервировать:

```
enum DayOfWeek {
    DAY_OF_WEEK_UNCPECIFIED = 0;
    MONDAY = 1;
    TUESDAY = 2;
    WEDNESDAY = 3;
    ...
}
```

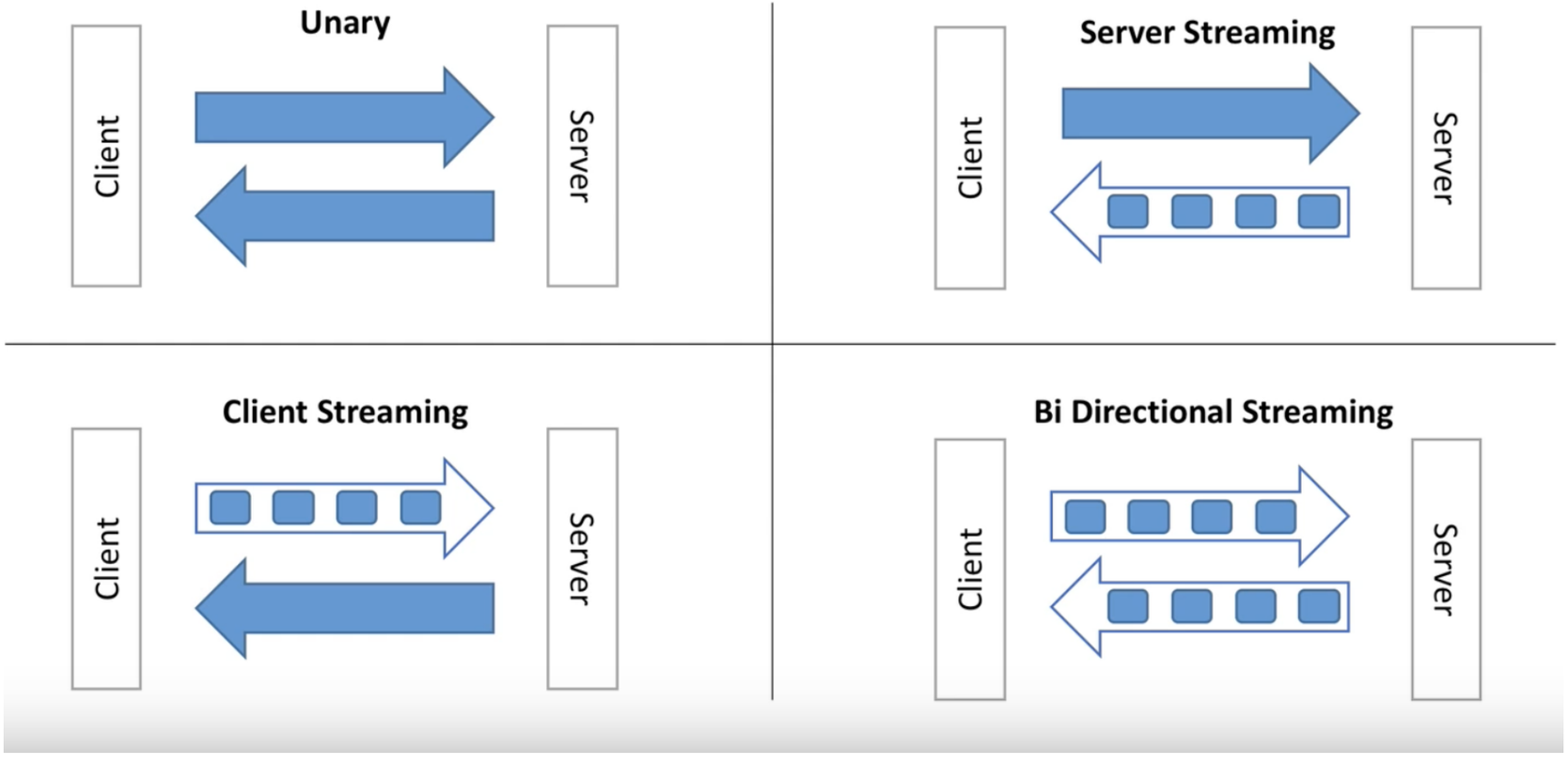
Protocol buffers: style guide

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;
}
```

Типы gRPC API



Unary boilerplate

```
syntax = "proto3";

package homework;

option go_package = "homeworkpb";


service HomeworkChecker {
    rpc CheckHomework (CheckHomeworkRequest) returns (CheckHomeworkResponse) {};
}

message CheckHomeworkRequest {
    int32 hw = 1;
    string code = 2;
}

message CheckHomeworkResponse {
    int32 grade = 1;
}
```

```
go get -u github.com/golang/protobuf/protoc-gen-go
protoc proto/homework.proto --go_out=plugins=grpc:.
protoc  --java_out=java --python_out=python *.proto
```

Boilerplate unary server (импорты убрал для краткости)

```go
package main

import (
    "otus-examples/otusrpc/homeworkpb"

    "google.golang.org/grpc"
)

type otusServer struct {
}

func (s *otusServer) CheckHomework(ctx context.Context, req *homeworkpb.CheckHomeworkRequest) (*homewor
    return nil, nil
}

func main() {
    lis, err := net.Listen("tcp", "0.0.0.0:50051")
    if err != nil {
        log.Fatalf("failed to listen %v", err)
    }

    grpcServer := grpc.NewServer()

    homeworkpb.RegisterHomeworkCheckerServer(grpcServer, &otusServer{})
    grpcServer.Serve(lis)
}
```

Boilerplate unary client (импорты убрал для краткости)

```
package main

import (
    "context"
    "log"
    "otus-examples/otusrpc/homeworkpb"

    "google.golang.org/grpc"
)

func main() {

    cc, err := grpc.Dial("localhost:50051", grpc.WithInsecure())
    if err != nil {
        log.Fatalf("could not connect: %v", err)
    }
    defer cc.Close()

    c := homeworkpb.NewHomeworkCheckerClient(cc)
    grade, err := c.CheckHomework(context.Background(), &homeworkpb.CheckHomeworkRequest{Hw: 10, Code:
    if err != nil {
        log.Fatalf("err getting grade: %v", err)
    }
    println(grade.Grade)
}
```

Boilerplate server streaming (server)

```
func (s *otusServer) CheckAllHomeworks(req *homeworkpb.CheckAllHomeworksRequest, stream homeworkpb.Home
    for _, hw := range req.Hw {
        res := &homeworkpb.CheckHomeworkResponse{Hw: hw, Grade: 67}
        stream.Send(res)
        time.Sleep(time.Second)
    }

    return nil
}
```

Boilerplate server streaming (client)

```
stream, err := c.CheckAllHomeworks(context.Background(), &homeworkpb.CheckAllHomeworksRequest{Hw: [
if err != nil {
    log.Fatalf("CheckAllHomeworks err %v", err)
}
for {
    msg, err := stream.Recv()
    if err == io.EOF {
        break
    }
    if err != nil {
        log.Fatalf("error reading stream: %v", err)
    }
    print(msg.Grade)
}
```

Boilerplate client streaming (server)

```
func (s *otusServer) SubmitAllHomeworks(stream homeworkpb.HomeworkChecker_SubmitAllHomeworksServer) err
    for {
        req, err := stream.Recv()
        if err == io.EOF {
            return stream.SendAndClose(&homeworkpb.SubmitAllHomeworksResponse{Accepted: true})
        }
        if err != nil {
            return err
        }
        _ = req
    }
}
```

Boilerplate client streaming (client)

```
    requests := []*homeworkpb.SubmitAllHomeworksRequest{
        &homeworkpb.SubmitAllHomeworksRequest{Hw: 1, Code: "first"},
        &homeworkpb.SubmitAllHomeworksRequest{Hw: 2, Code: "second"},
    }
    cstream, err := c.SubmitAllHomeworks(context.Background())
    if err != nil {
        log.Fatalf("err streaming: %v", err)
    }
    for _, req := range requests {
        cstream.Send(req)
    }

    res, err := cstream.CloseAndRecv()
    if err != nil {
        log.Fatalf("err getting resp: %v", err)
    }
    println(res.GetAccepted())
```

Boilerplate bi-directional streaming server

```
func (s *otusServer) RealtimeFeedback(stream homeworkpb.HomeworkChecker_RealtimeFeedbackServer) error {
    for {
        req, err := stream.Recv()
        if err == io.EOF {
            return nil
        }
        if err != nil {
            log.Fatalf("error reading client stream: %v", err)
            return err
        }
        _ = req
        sendErr := stream.Send(&homeworkpb.CheckHomeworkResponse{Hw: 1, Grade: 5})
        if sendErr != nil {
```

gRPC: Errors

https://grpc.io/docs/guides/error/ 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/

```
func (*server) SquareRoot(ctx context.Context, req *calculatorpb.SquareRootRequest) (*calculatorpb.Squa
    fmt.Println("Received SquareRoot RPC")
    number := req.GetNumber()
    if number < 0 {
        return nil, status.Errorf(
            codes.InvalidArgument,
            fmt.Sprintf("Received a negative number: %v", number),
        )
    }
    return &calculatorpb.SquareRootResponse{
        NumberRoot: math.Sqrt(float64(number)),
    }, nil
}
```

gRPC: Errors

```
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: Deadlines

```
clientDeadline := time.Now().Add(time.Duration(*deadlineMs) * time.Millisecond)
ctx, cancel := context.WithDeadline(ctx, clientDeadline)
```

```
if ctx.Err() == context.Canceled {
    return status.New(codes.Canceled, "Client cancelled, abandoning.")
}
```

gRPC: Reflection + Evans CLI

```go
import "google.golang.org/grpc/reflection"

s := grpc.NewServer()
pb.RegisterYourOwnServer(s, &server{})

// Register reflection service on gRPC server.
reflection.Register(s)

s.Serve(lis)
```

https://github.com/ktr0731/evans

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 + REST: clay

https://github.com/utrack/clay


Чистая архитектура
Устройства
Web
Контроллеры
Сценарии
Сущности
Шлюзы
Представления
БД
UI
Внешние интерфейсы
Бизнес-правила предприятия
Бизнес-правила приложения
Интерфейс-Адаптеры
Фреймворки и Драйверы
Представление
Сценарий Выходной порт
Сценарий Исполнитель
Контроллер
Сценарий Входной порт

Clean Architecture

https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

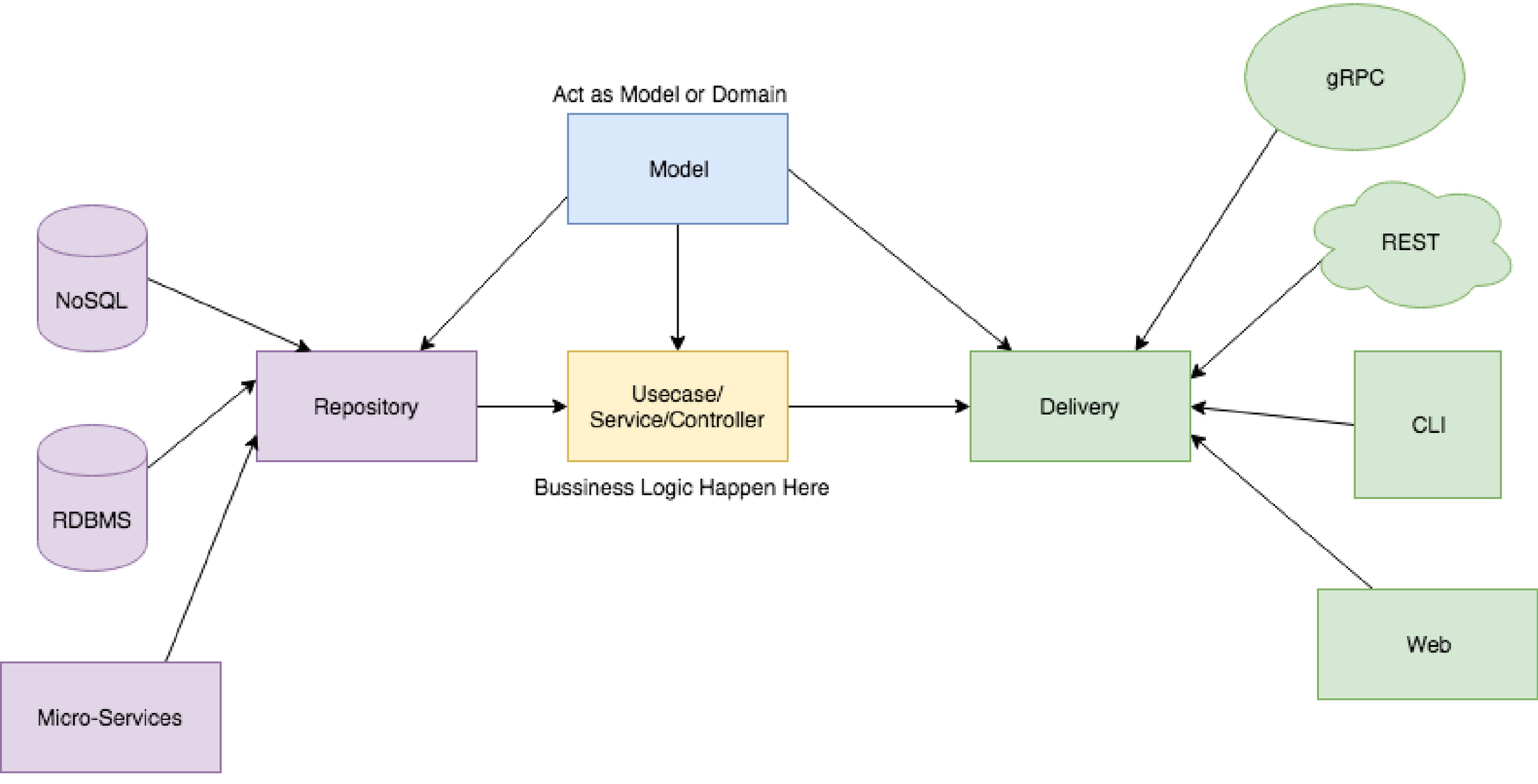
- независимость от фреймворка
- тестируемость
- независимость от UI
- независимость от базы данных
- независимость от какого-либо внешнего сервиса

Правило Зависимостей. Зависимости в исходном коде могут указывать только во внутрь. Ничто из внутреннего круга не может знать что-либо о внешнем круге, ничто из внутреннего круга не может указывать на внешний круг.

Clean Architecture

- Entities (models, модели)
- Use Cases (controllers, сценарии)
- Interface Adapters
- Frameworks and Drivers (инфраструктура)

Clean Architecture



https://github.com/bxcodec/go-clean-arch

-

Тест

https://forms.gle/SiDmYTPUU5La3rA88

На занятии

- Научились писать gRPC сервисы
- Научились писать Protobuf схемы
- Изучили принципы Clean Architecture

Вопросы?

Опрос

Не заполните заполнить опрос. Ссылка на опрос будет в слаке.

Спасибо за внимание!