From REST to gRPC:

An API Evolution Story

Joe Runde
IBM
@joerunde

Michael Keeling
IBM
@michaelkeeling

What is gRPC?

Open source **R**emote **P**rocedure **C**all framework

What is gRPC?

Open source **R**emote **P**rocedure **C**all framework

"… a modern, bandwidth and CPU efficient, low latency way to create massively distributed systems that span data centers"

Why should we use gRPC?

Why is gRPC awesome?

Performance

Performance Benefits

HTTP / REST

Performance Benefits

HTTP / REST

gRPC

How does gRPC improve performance?

- HTTP/2 transport protocol
- Binary encodings via protocol buffers
- No more parsing text!
- Compression
- Streaming

Why is gRPC awesome?

Performance

Why is gRPC awesome?

Performance

Remote Procedure Calls

REST setup is tedious

```
headers = {}
headers.put(“X-FooBar-User”,
            user.getFooBarID())
headers.put(“FooBarTransaction”,
            app.getNextTxnIDAtomic())
headers.put(“FooBarAlgoType”,
            ApproximateFoo.toHeaderString())
```

REST setup is tedious

```
Public class Foo {

    @JsonProperty(name=”foo_id”)
    String id;

    @JsonProperty(name=”bar”)
    int bar;
    ...
}
```

REST setup is tedious

```
Public class Foo {
    ...
    public
    Foo(@JsonProperty(“foo_id”,required=true)
        String id,
        @JsonProperty(“bar”,required=true)
        int bar)
    { ... }
}
```

RPC setup is easy

```
request = FooService.RequestBuilder()
        .setId(foo.getId())
        .setBar(foo.getBar())...
        .build();

response = fooClient.DoFooBar(request);
```

Why is gRPC Awesome?

Performance

Remote Procedure Calls

Why is gRPC Awesome?

Performance
Remote Procedure Calls
Strategic Direction of our Platform

<insert slick demo here>

Just one problem…

Our current microservices all use REST.


gRPC
REST

Business Constraints

The Transition Plan

- Phase 1: Design gRPC API
- Phase 2: Run REST and gRPC services
- Phase 3: Transition functional tests
- Phase 4: Remove REST functionality

DESIGN THE API

Phase I

Designing a gRPC API

REST:

```
POST
/api/foo

GET
/api/foo/{foo_id}
```

gRPC:

```
rpc AddFoo(Foo)
    returns FooID;

rpc GetFoo(FooID)
    returns Foo;
```

Designing a gRPC API

REST:

```
POST
/api/foo

GET
/api/foo/{foo_id}
```

gRPC:

```
message Foo {
    FooID id = 1;
    repeated Bar bars = 2;
}

message FooID {
    string val = 1;
}
```

Designing a gRPC API

REST:

```
POST
/api/foo/{foo_id}/bar

GET
/api/foo/{foo_id}/bar/
{bar_id}
```

gRPC:

```
rpc AddBar(Bar)
    returns BarID;

rpc GetFoo(BarID)
    returns Bar;
```

Designing a gRPC API

REST:

```
POST
/api/foo/{foo_id}/bar

GET
/api/foo/{foo_id}/bar/
{bar_id}
```

gRPC:

```
rpc AddBar(Bar)
    returns BarID;

rpc GetFoo(BarID)
    returns Bar;
```

Designing a gRPC API

REST:

```
POST
/api/foo/{foo_id}/bar

GET
/api/foo/{foo_id}/bar/
{bar_id}
```

gRPC:

```
message Bar{
    BarID id = 1;
    int baz = 2;
}

message BarID {
    FooID foo_id = 1;
    string bar_id = 2;
}
```

Designing a gRPC API

Bar service:

```
/api/foo/{foo_id}/bar/{bar_id}
```

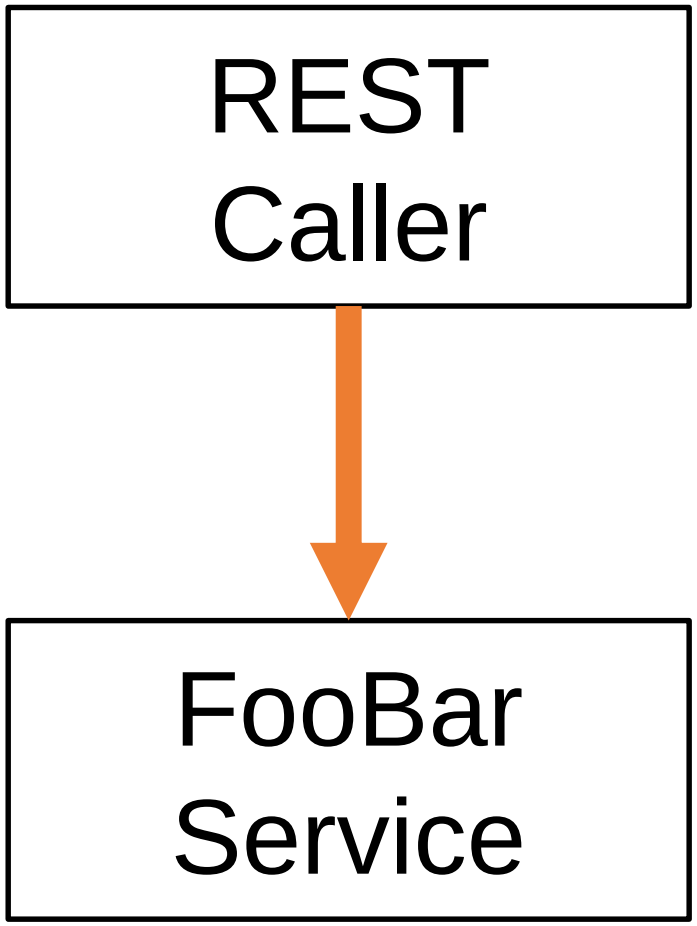
Buzz service:

```
/api/foo/{foo_id}/buzz/{buzz_id}
```

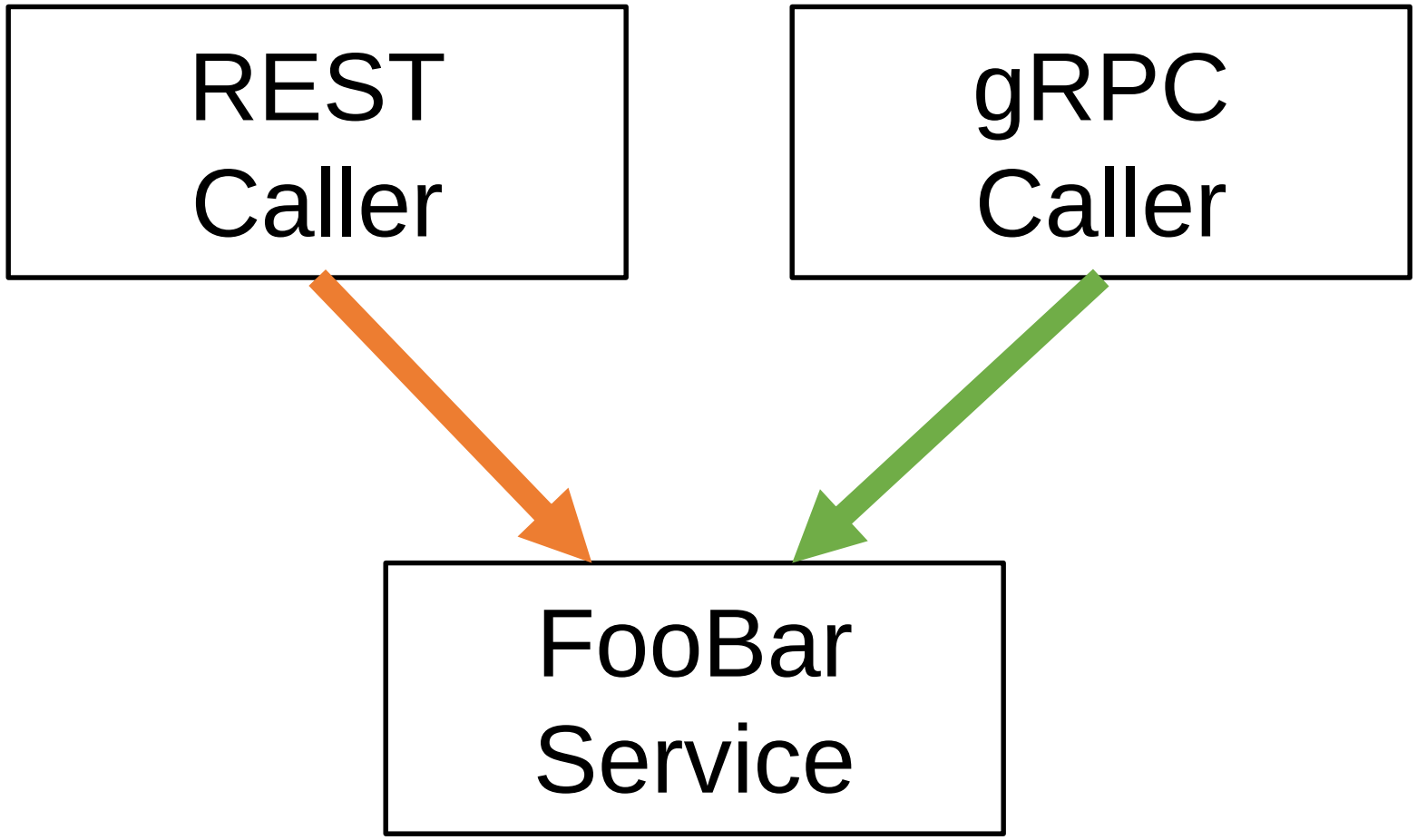
SERVE REST AND GRPC

Phase II

Current: REST Only

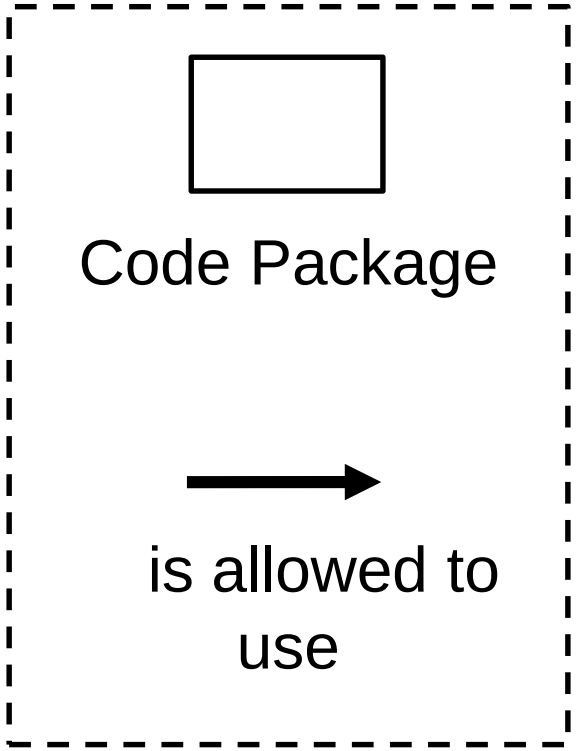


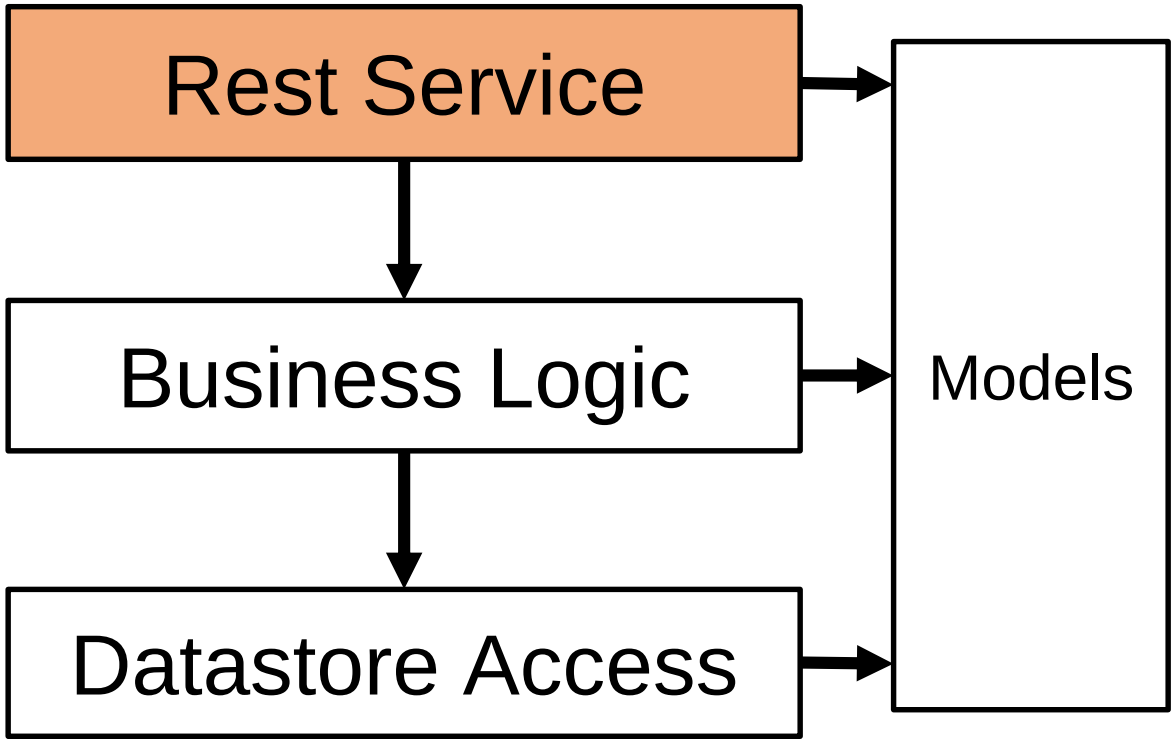
Future: REST and gRPC



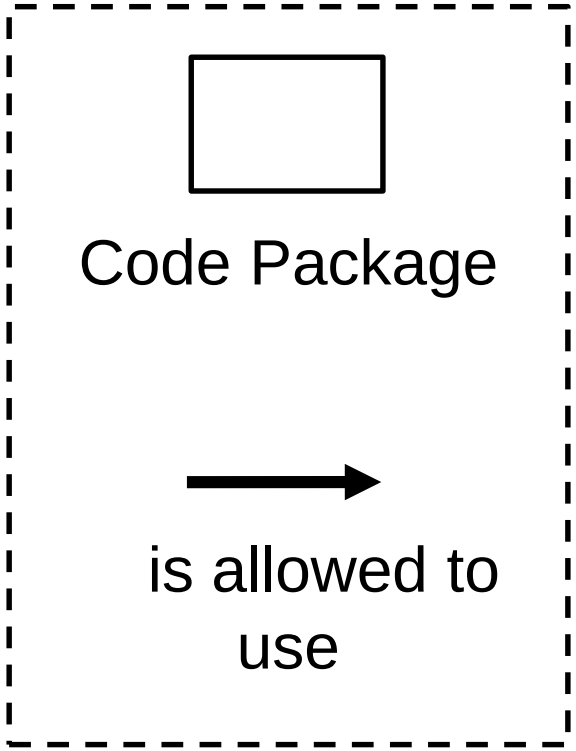
We need to evolve our API without damaging basic functionality.

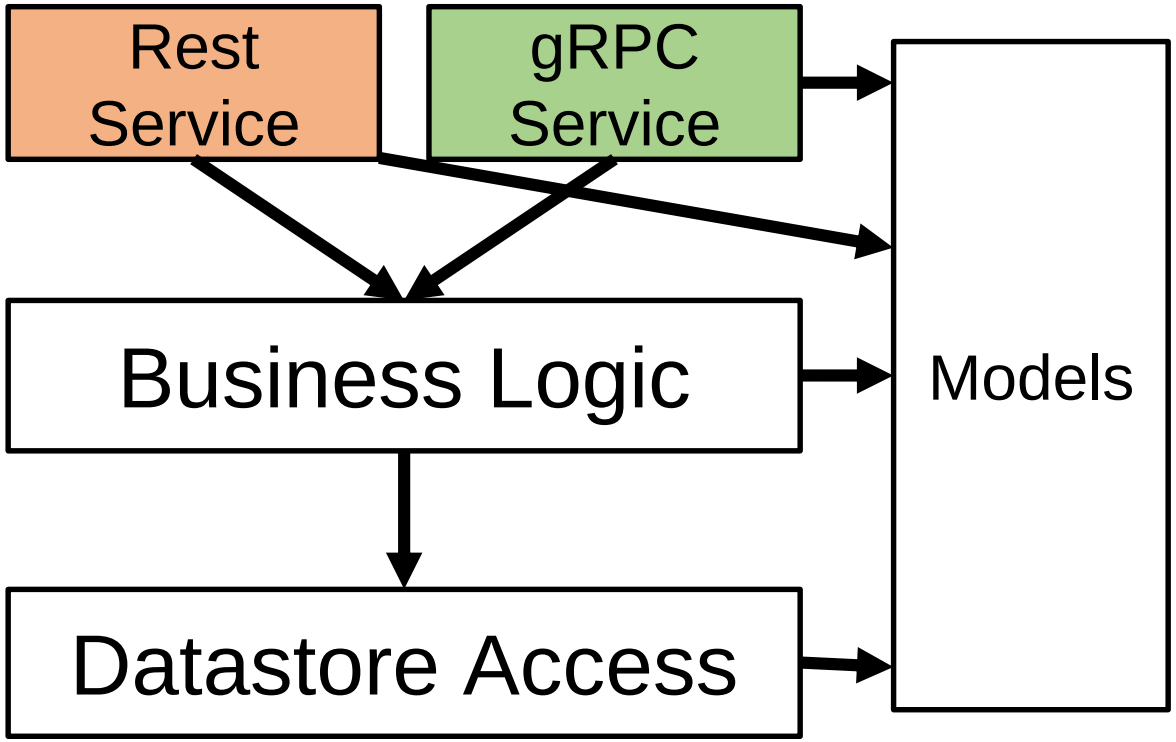
Current: Layers View





Future: Layers View





Layer Pattern Rocks!

A code review comment:

"What's with the service layer? This just passes its inputs to our business logic functions, it's redundant cruft!"

Evolving the code went really well…

Things that suddenly became a problem

- Health Checks
- API Discovery
- No curl
- Headers?
- SSL?
- Simple community examples

TRANSITION FUNCTIONAL TESTS

Phase III

Specification by Example

Specification by Example- REST

```
Scenario: Queries can be deleted
  Given the request body is
    """
    { "natural_language_query": "No Documents Query to be deleted" }
    """
  And a POST request is sent to the default endpoint template
  And the value of 'query_id' in the response is saved in a key called
  When a DELETE request is sent to the endpoint template named 'query'
  Then the response code is 204
  And a GET request is sent to the default endpoint template
  And the 'queries' field in the response is empty
```

Ruby metaprogramming

Choice of programming language *really* paid off

```
request = Object::const_get(
    ”FooBar::#{message_name}").decode_json(json)

response = client.method(method_name.to_sym)
    .call(request)
```

Specification by Example- gRPC

```
Scenario: Queries can be deleted
  Given a 'Query' that looks like
    """
      { "natural_language_query": "No Documents Query to be deleted" }
    """
  And I call the 'add_query' method in the TrainingCrudService
  And the value of 'query_id' in the response object is saved in a key ca
  When I call the 'delete_query' method in the TrainingCrudService with t
  Then I call the 'get_query' method in the TrainingCrudService
  And the response gives the error code '5'
```

200+ tests transitioned in 1 week

Why did this go so well?

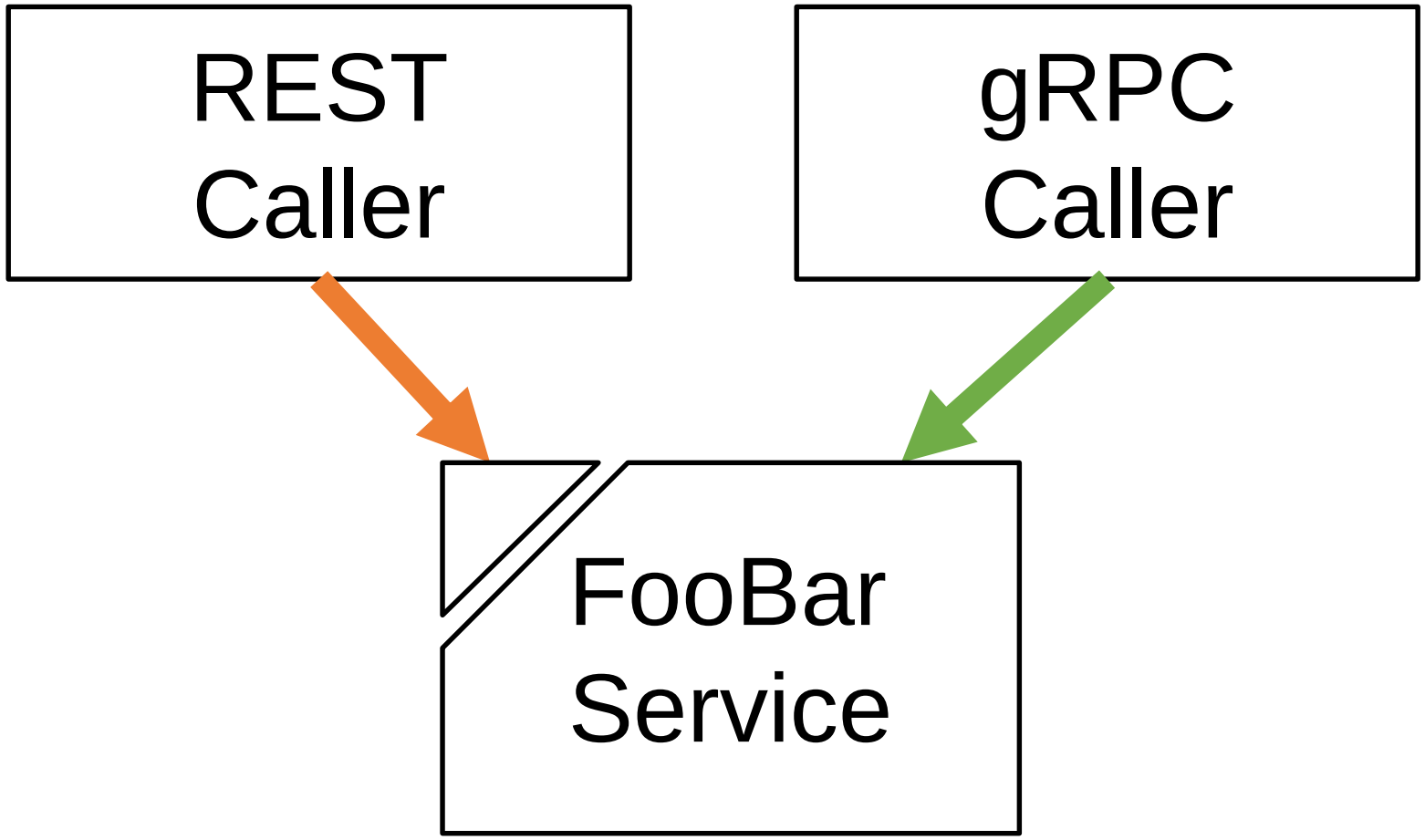
- Expressiveness of spec by example
- Flexibility of Ruby
- gRPC can decode JSON

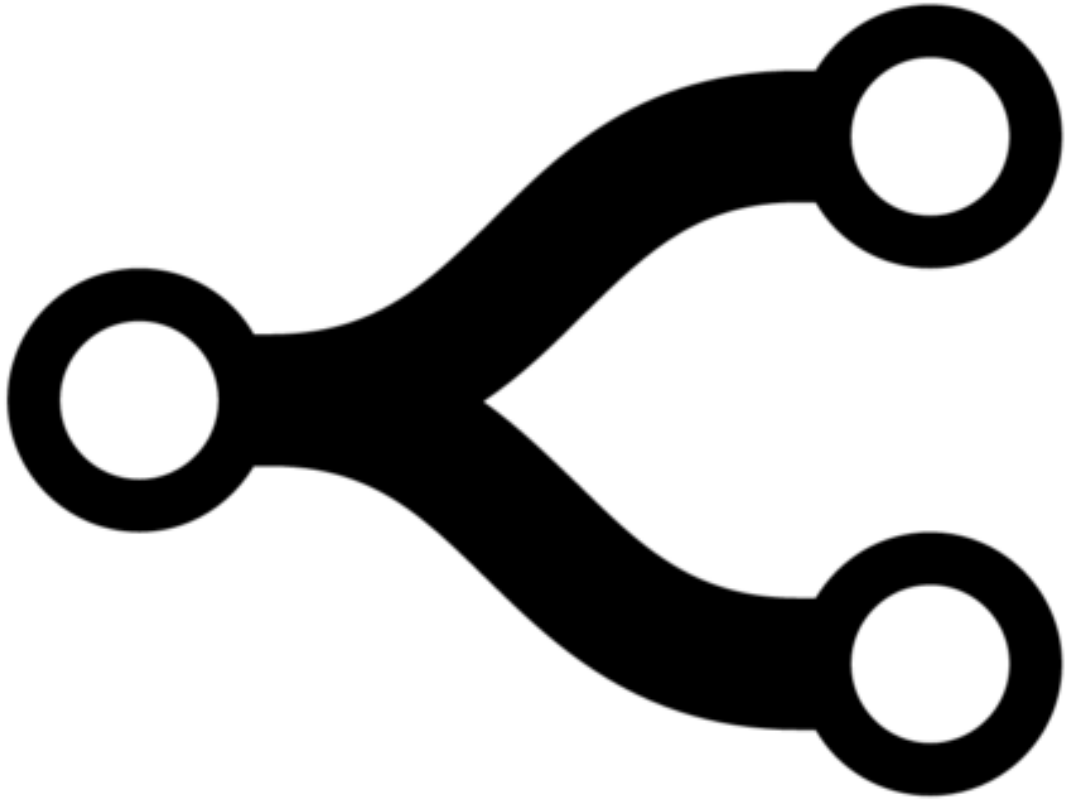
REMOVE REST FUNCTIONALITY

Phase IV

```
rm -rf src/*rest*
```

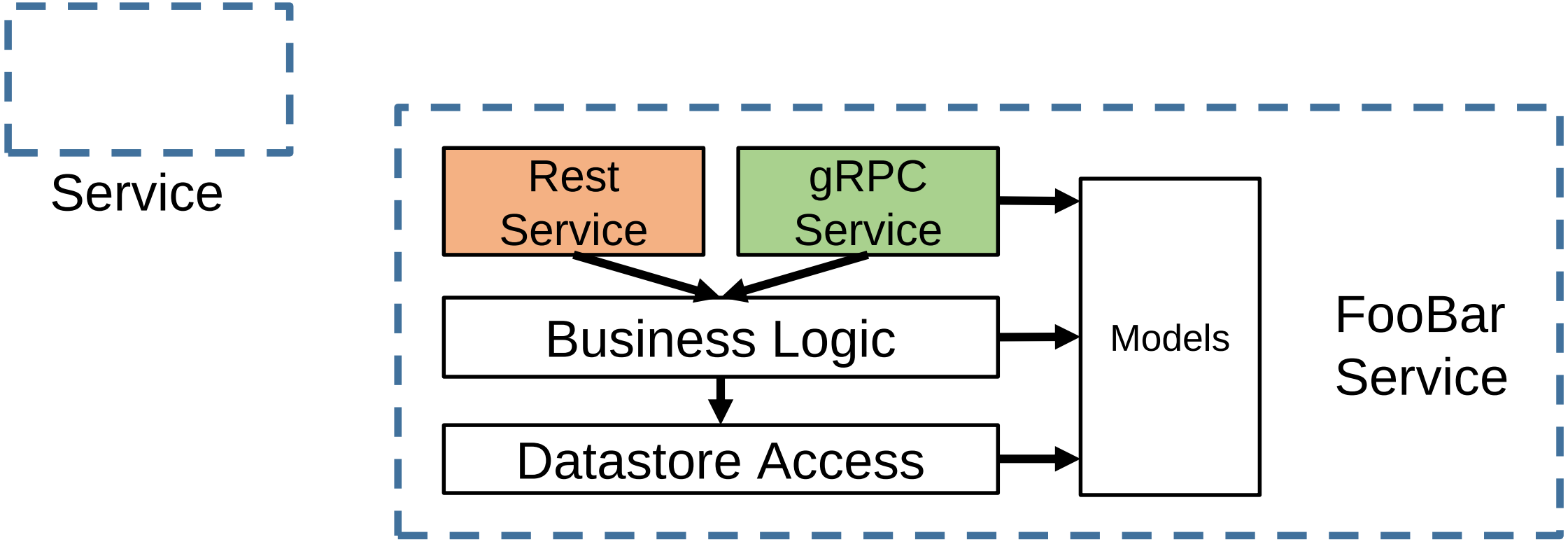
We need to retain some REST endpoints



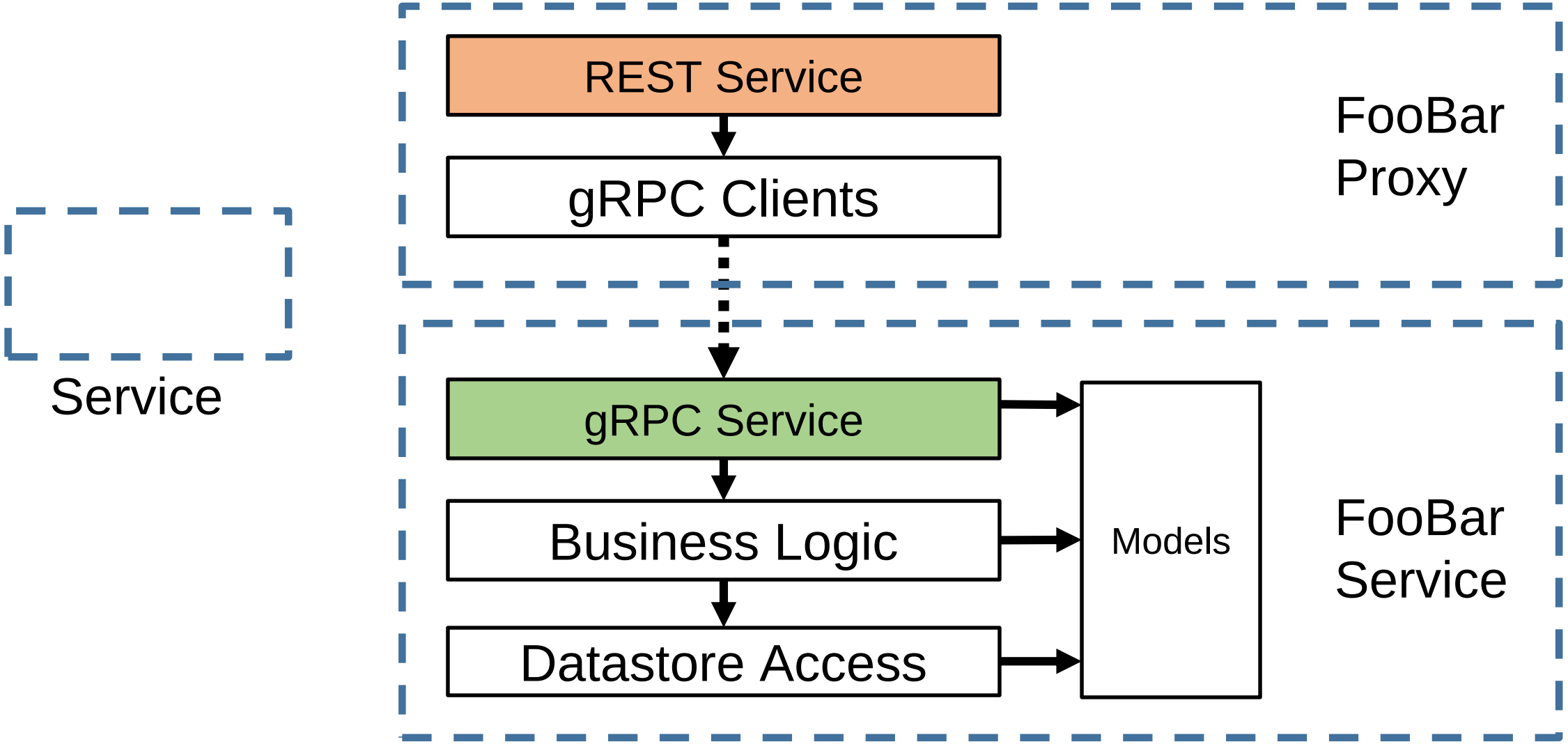


FORKED

Current Layered Architecture



Layers Forked – Two Services



Evil wizards strike again!

NOW WE’RE READY FOR RELEASE…?

New problems we created

- Health Checks
- API Discovery
- No curl
- Headers?
- SSL?

New problems we created

- Health Checks ✓
- API Discovery
- No curl
- Headers? ✓
- SSL? ✓

API Discovery

- REST – Ask the service!
- gRPC – Find the (correct) proto file?

API Discovery

- REST – Ask the service!
- gRPC – Find the (correct) proto file?
  - Standard InfoService serves github url + version
  - Snapshot proto files with releases
  - Client vendors the proto files they use

Tools support

- Basically Nonexistent
- Our solutions:
  - Hand roll mocks for testing
  - Write new functional tests each time we wanted to use curl


Adios,
REST!

INTERESTING THINGS WE LEARNED

- The right kinds of abstractions promote extensibility
- Focus on the domain model
- Create a specification by example
- Take care when choosing frameworks
- Deal with risks of technology adoption

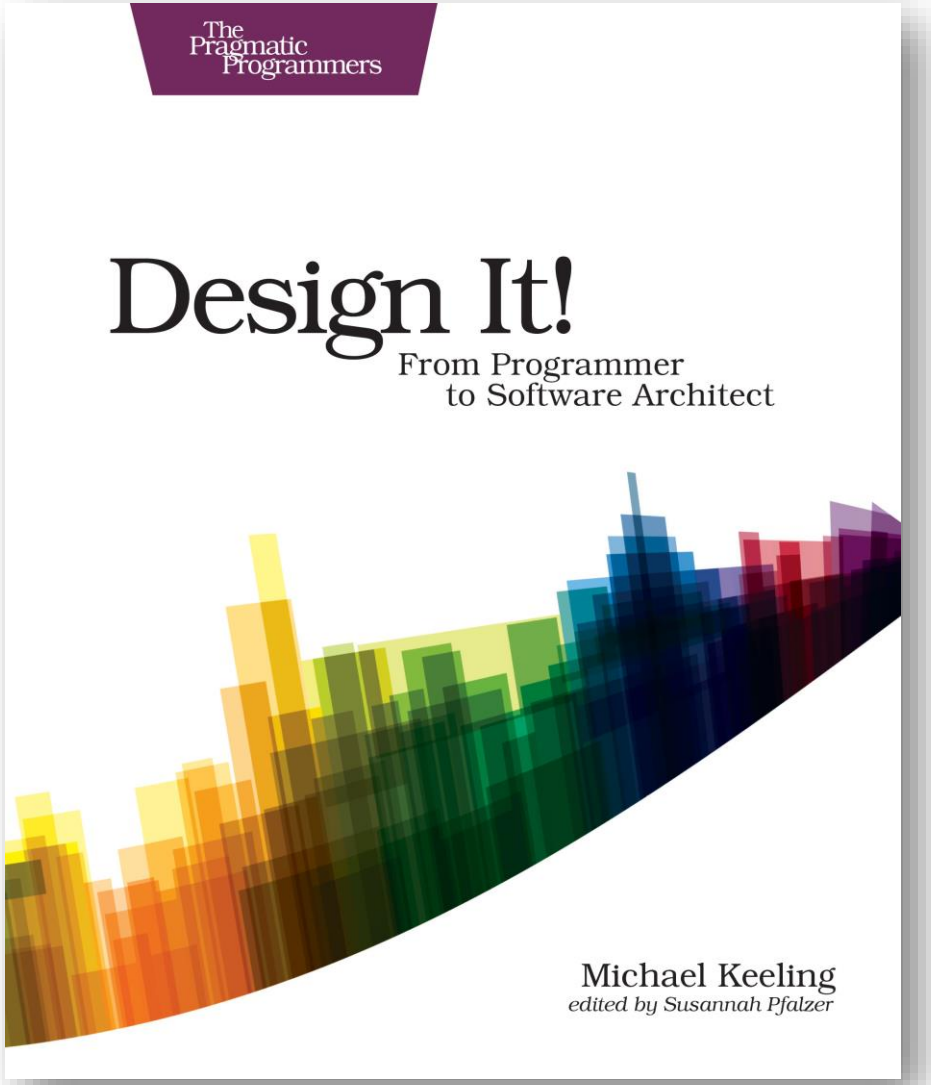
Would we do it again?

Would we do it again?

Yes.

- Super easy to integrate with a service
- Promotes small polyglot services
- Difficult to do bad things
- Performance is 🔥🔥🔥

Thank you!



Michael Keeling
@michaelkeeling
neverletdown.net

Joe Runde
@joerunde

Buy Design It! now at
http://bit.ly/2pJOrly

BACKUP

```
syntax = "proto3";

package math_service;

service Math {
  rpc Divide (Operands) returns (Result) {}
}

// A message with numeric operands for a math operation
message Operands {
  int32 dividend = 1;
  int32 divisor = 2;
}

// A message  with numerical results from a math operation
message Result {
  float quotient = 1;
}
```

```go
package main

import ...

func main() {

        listener, err := net.Listen("tcp", ":50051")
        if err != nil {
                os.Exit(-1)
        }

        server := grpc.NewServer()
        math_service.RegisterMathServer(server, mathServer{})
        server.Serve(listener)
}
```

```go
type mathServer struct {
}

func (m mathServer) Divide(ctx context.Context,
        in *math_service.Operands) (*math_service.Result, error) {

        if in.Divisor == 0 {
                return nil, grpc.Errorf(codes.InvalidArgument, "Divisor may not be zero")
        }

        res := math_service.Result{Quotient: float32(in.Dividend) / float32(in.Divisor)}
        return &res, nil
}
```

```go
package main

import ...

func main() {

        conn, err := grpc.Dial("localhost:50051", grpc.WithInsecure())
        if err != nil {
                println(err.Error())
                os.Exit(-1)
        }

        client := math_service.NewMathClient(conn)

        result, err := client.Divide(context.Background(),
                &math_service.Operands{Dividend: 10, Divisor: 4})
        println(result.Quotient)

        _, err = client.Divide(context.Background(),
                &math_service.Operands{Dividend: 10, Divisor: 0})
        fmt.Println(err)
}
```