Banked :
A journey into Protocol Buffers.

Contact
chris@banked.com
Chris Reeves — Tech Lead (Kuiper Team)

Team 🔥 Kuiper Tech Lead

Platform Golang Engineering Team

2014 - 2018
SOON_

2018 - 2019
Karhoo

2019
City Sprint (on the dot)

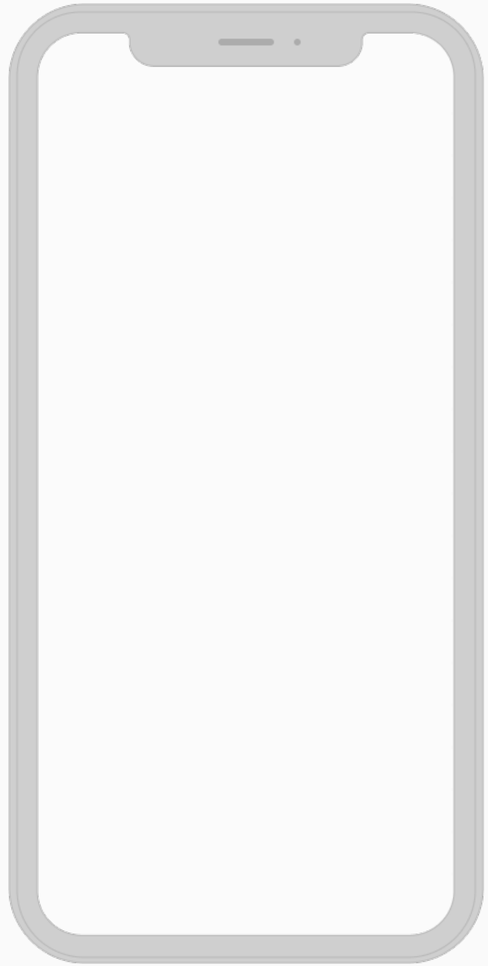
2019 — 2021
Vidsy

- Full time Go Engineer for 6/7 🤷‍♂️ years.
- @krak3n Twitter.
- @krak3n Github.
- linkedin.com/in/krak3n/

What is Banked?

A global payments network built on modern banking rails.

Banked powers real-time payments for consumers, businesses and banks, improving customer experience, payment security, business efficiency and cost effectiveness.



A better way to take and make payments.

banked.com

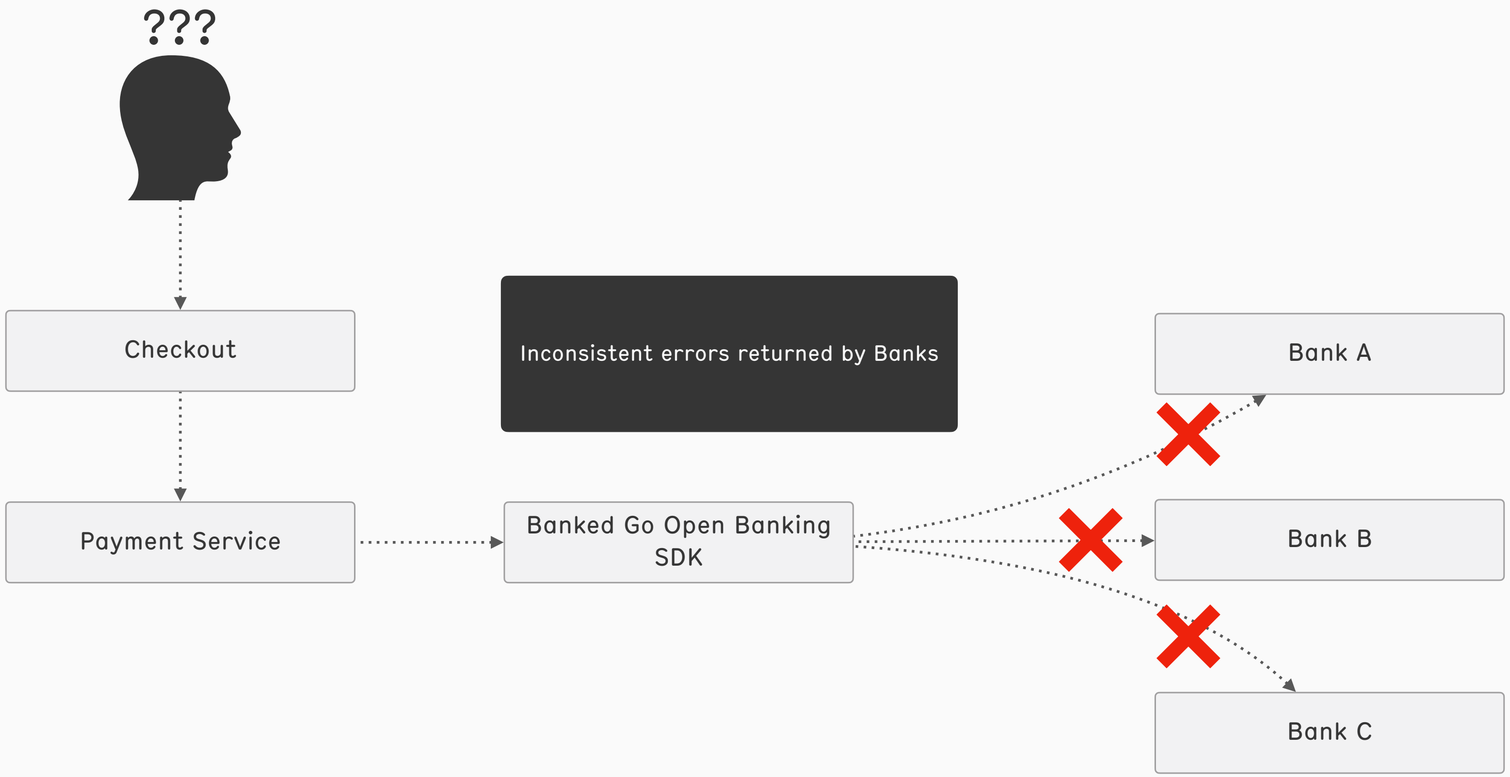
OPEN BANKING

We are hiring.

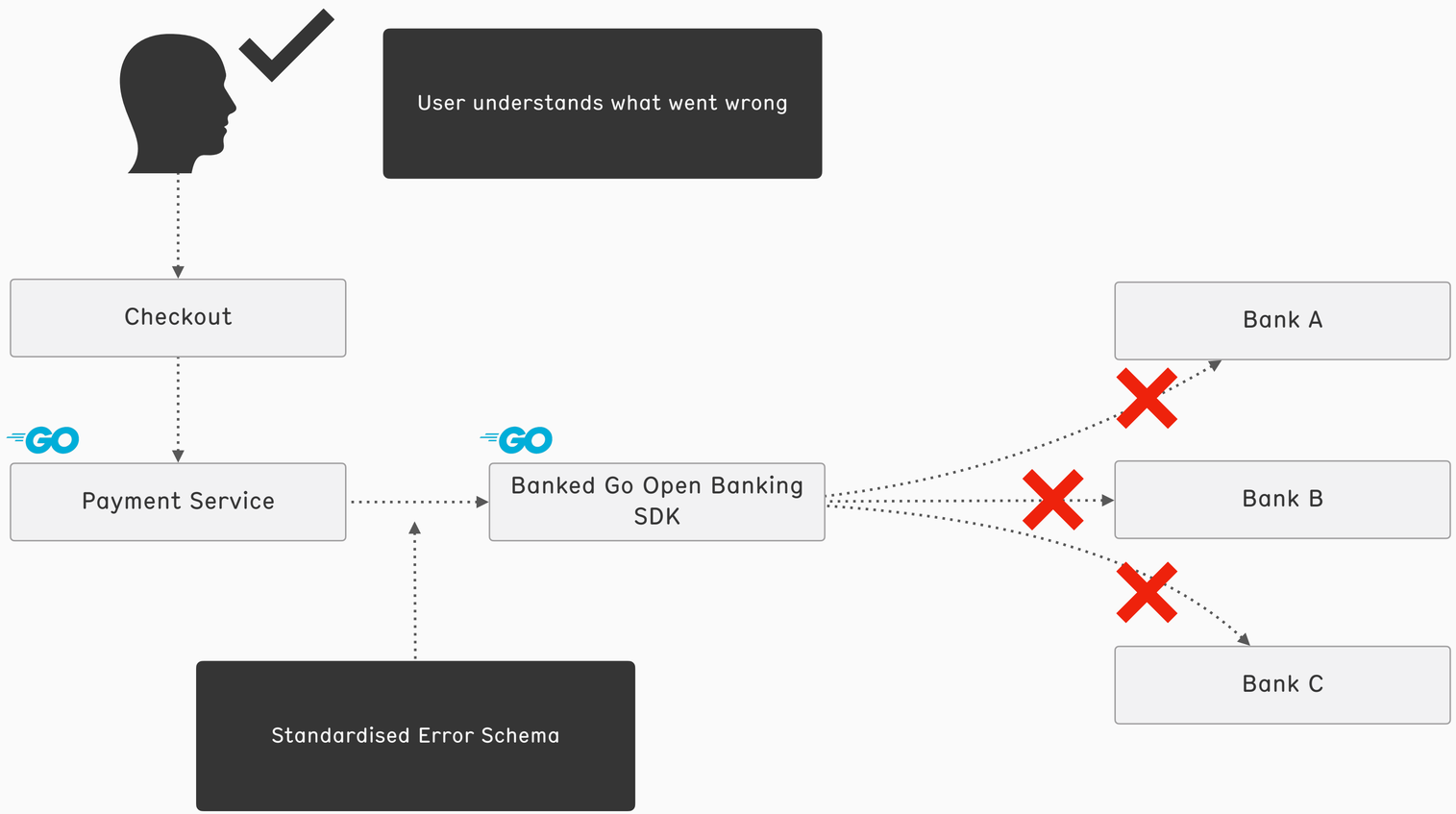
GITHUB.COM/BANKED/JOBS

Chapter I
Darkness looms on the horizon

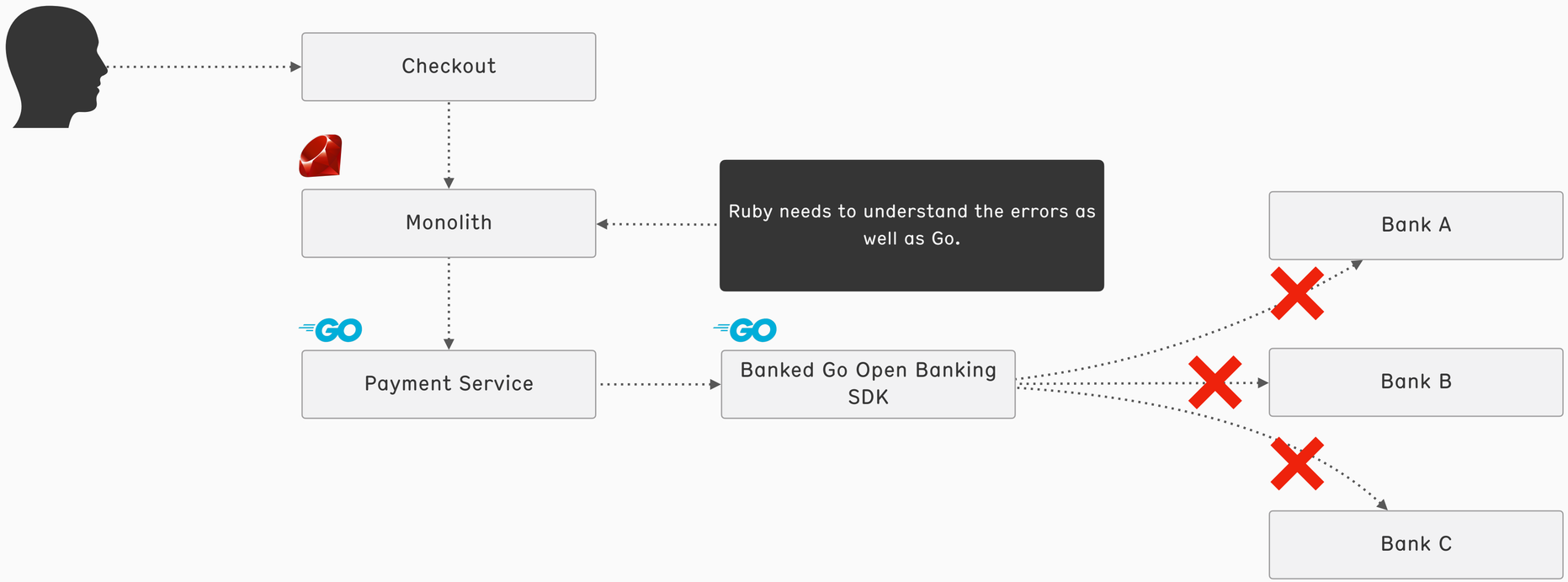
Inconsistent Errors



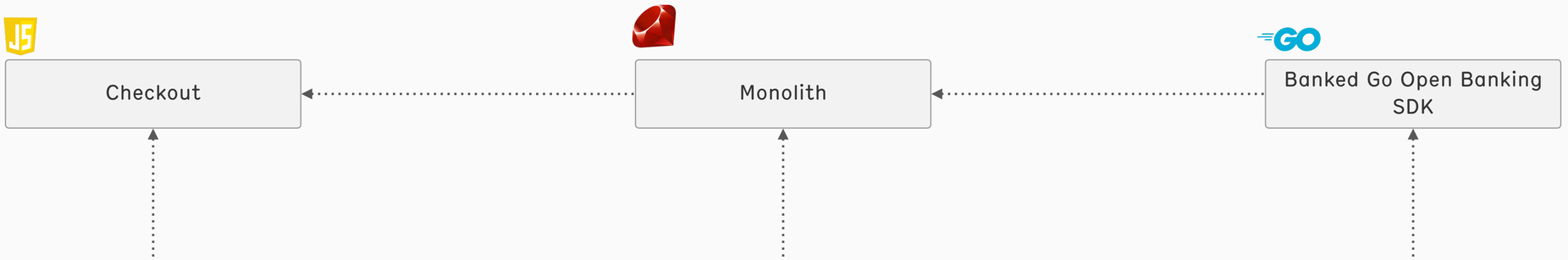
Standardised Error Schema



Our Monolith



3 Places where we need to define the schema



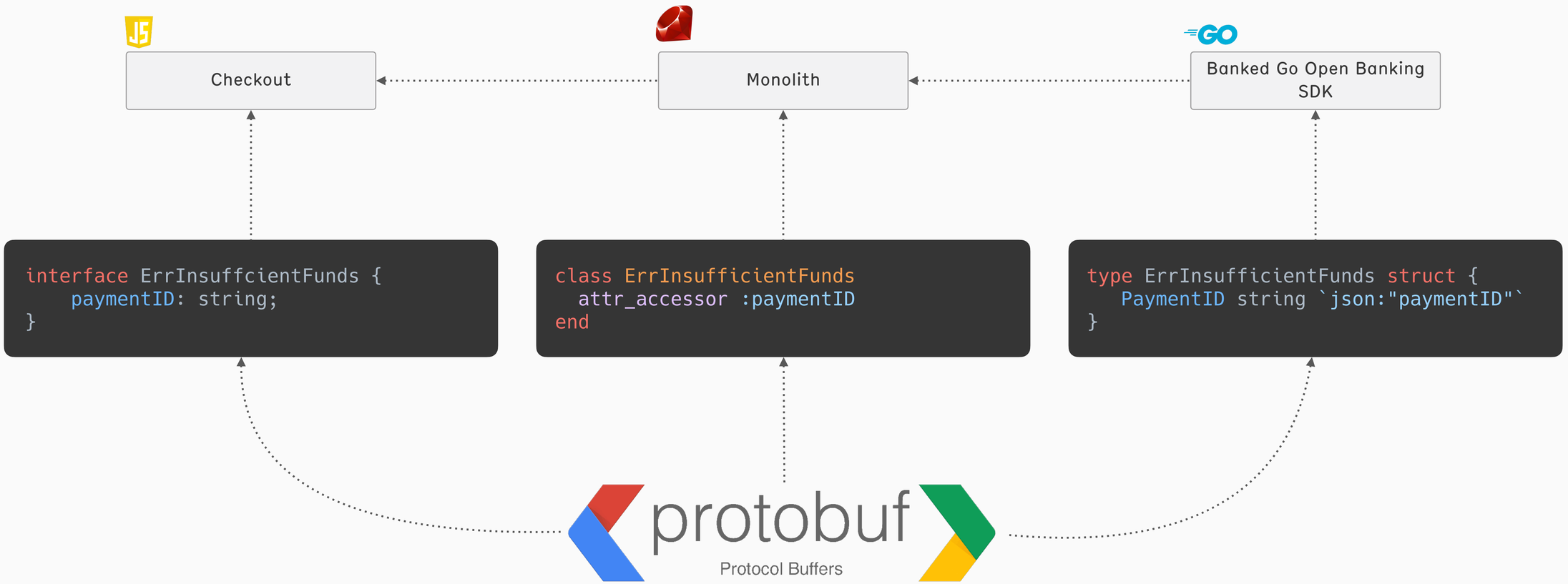

```typescript
interface ErrInsuffcientFunds {
    paymentID: string;
}
```

```ruby
class ErrInsufficientFunds
  attr_accessor :paymentID
end
```

```go
type ErrInsufficientFunds struct {
   PaymentID string `json:"payment_id"`
}
```

Chapter II
And then a hero comes along

Protocol Buffers to the rescue



What are Protocol Buffers

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

- Mechanism for serialising structured data
- Language & Platform Neutral
- Extensible
- Define your structure once and use code generation for the languages you use
- Compact (Binary message format, but you don’t have too)
  - Fast parsing
- Supports C++, C#, Go, Java, Kotlin, Objective-C, PHP, Python, Ruby

```
syntax = "proto3";

package payment;

message ErrInsuffcientFunds {
  string payment_id = 1;
  string amount = 2;
  string currency = 3;
  string provider = 4;
}
```

GO

```
type ErrInsuffcientFunds struct {
	PaymentId string `protobuf:"bytes,1,opt,name=paymentID,proto3" json:"paymentID,omitempty"`
	Amount    string `protobuf:"bytes,2,opt,name=amount,proto3" json:"amount,omitempty"`
	Currency  string `protobuf:"bytes,3,opt,name=currency,proto3" json:"currency,omitempty"`
	Provider  string `protobuf:"bytes,4,opt,name=provider,proto3" json:"provider,omitempty"`
}
```

Anatomy of a .proto file

https://developers.google.com/protocol-buffers/docs/proto3

```
syntax = "proto3";

option go_package = "github.com/banked/gopherconuk2022/demo1";

package payment;

message ErrInsuffcientFunds {

  string payment_id = 1;

  int64 amount = 2;

  string currency = 3;
}
```

Syntax

Defines the protocol buffer `syntax` used in the proto file. Valid options are:

- `proto2`
- `proto3`

Anatomy of a .proto file

https://developers.google.com/protocol-buffers/docs/proto3

```
syntax = "proto3";

option go_package = "github.com/banked/gopherconuk2022/demo1";

package payment;

message ErrInsuffcientFunds {

  string payment_id = 1;

  int64 amount = 2;

  string currency = 3;
}
```

Option

Some language support `option`'s. This options do not alter the overall meaning of the definition but may affect the generated code.

For example the `go_package` configures the package name and therefore the location of the generated go code.

A full list can be found in the Protocol Buffers documentation.

https://developers.google.com/protocol-buffers/docs/proto3#options

Anatomy of a .proto file

https://developers.google.com/protocol-buffers/docs/proto3

```proto
syntax = "proto3";

option go_package = "github.com/banked/gopherconuk2022/demo1";

package payment;

message ErrInsuffcientFunds {

  string payment_id = 1;

  int64 amount = 2;

  string currency = 3;
}
```

Package

Messages can be organised into different packages a lot like go packages.

Package names should always be lowercase.

Use `import` to import messages from other packages.

```proto
syntax = "proto3";

import "google/protobuf/timestamp.proto";

package payment;

message Foo {
  google.protobuf.Timestamp created_at = 1;
}
```

Well Known Types

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

Anatomy of a .proto file

https://developers.google.com/protocol-buffers/docs/proto3

```
syntax = "proto3";

option go_package = "github.com/banked/gopherconuk2022/demo1";

package payment;

message ErrInsuffcientFunds {

  string payment_id = 1;

  int64 amount = 2;

  string currency = 3;
}
```

Message Names

Defines a name for the message. Akin to a `struct` type name for example.

Should use `TitleCase` (`PascalCase`).

Anatomy of a .proto file

https://developers.google.com/protocol-buffers/docs/proto3

```
syntax = "proto3";

option go_package = "github.com/banked/gopherconuk2022/demo1";

package payment;

message ErrInsuffcientFunds {

  string payment_id = 1;

  int64 amount = 2;

  string currency = 3;
}
```

Field Number

Each field in the message must have a unique number. These numbers are used to identify the field in the binary message format.

Field numbers should NOT be changed once defined on a message.

The ordering of the numbers does not matter.

You can have gaps in the field numbers.

Anatomy of a .proto file

https://developers.google.com/protocol-buffers/docs/proto3

```
syntax = "proto3";

option go_package = "github.com/banked/gopherconuk2022/demo1";

package payment;

message ErrInsuffcientFunds {

  string payment_id = 1;

  int64 amount = 2;

  string currency = 3;
}
```

Field Name

Names are not included in the encoded binary message.
Convention is to use `underscore_separated_names`.

Names can technically change and still work between two different versions of the .proto

For example we could rename `amount` to `cents` and code using the old field name would not be affected.

But this is not the case for JSON encoded messages.

```
{
  "paymentId": "123",
  "amount": "1000",
  "currency": "GBP",
  "provider": "Bank A"
}
```

Anatomy of a .proto file

https://developers.google.com/protocol-buffers/docs/proto3

```
syntax = "proto3";

option go_package = "github.com/banked/gopherconuk2022/demo1";

package payment;

message ErrInsuffcientFunds {

  string payment_id = 1;

  int64 amount = 2;

  string currency = 3;
}
```

Field Type

Each field must define its type. These can be scalar values or other message types.

https://developers.google.com/protocol-buffers/docs/proto3#scalar

```
message Foo {
  string id = 1;
}

message Bar {
  Foo foo = 1;
}
```

Package Structure

Use version sub directories.

```
└── proto
    └── foo
        └── bar
            ├── bat
            │   └── bat.proto // package foo.bar.bat
            └── baz
                ├── baz.proto         // package foo.bar.baz
                └── baz_service.proto // package foo.bar.baz
```

```
└── proto
    └── foo
        └── bar
            ├── bat
            │   └── v1
            │       └── bat.proto // package foo.bar.bat.v1
            └── baz
                └── v1
                    ├── baz.proto         // package foo.bar.baz.v1
                    └── baz_service.proto // package foo.bar.baz.v1
```

Avoid Breaking Changes

Package Structure with a V2

```
└── proto
    └── foo
        └── bar
            ├── bat
            │   └── v1
            │       └── bat.proto // package foo.bar.bat.v1
            └── baz
                ├── v1
                │   ├── baz.proto         // package foo.bar.baz.v1
                │   └── baz_service.proto // package foo.bar.baz.v1
                └── v2
                    ├── baz.proto         // package foo.bar.baz.v2
                    └── baz_service.proto // package foo.bar.baz.v2
```

Generating code from a .proto file

`protoc` is the tool which handles processing `.proto` files and generating code.

```
protoc -I=$(pwd) --go_out=$(pwd) $(pwd)/payment.proto
```

Generating code from a .proto file

`protoc` is the tool which handles processing `.proto` files and generating code.

```
protoc -I=$(pwd) --go_out=$(pwd) $(pwd)/payment.proto
```

`-I` tells `protoc` to look for `.proto` files in this directory. Can be provided multiple times.

Generating code from a .proto file

`protoc` is the tool which handles processing `.proto` files and generating code.

Tells `protoc` to generate go code and where to put the generated code. `proto-gen-go` required.

```
protoc -I=$(pwd) --go_out=$(pwd) $(pwd)/payment.proto
```

`-I` tells `protoc` to look for `.proto` files in this directory. Can be provided multiple times.

Generating code from a .proto file

`protoc` is the tool which handles processing `.proto` files and generating code.

Tells `protoc` to generate go code and where to put the generated code. `proto-gen-go` required.

```
protoc -I=$(pwd) --go_out=$(pwd) $(pwd)/payment.proto
```

`-I` tells `protoc` to look for `.proto` files in this directory. Can be provided multiple times.

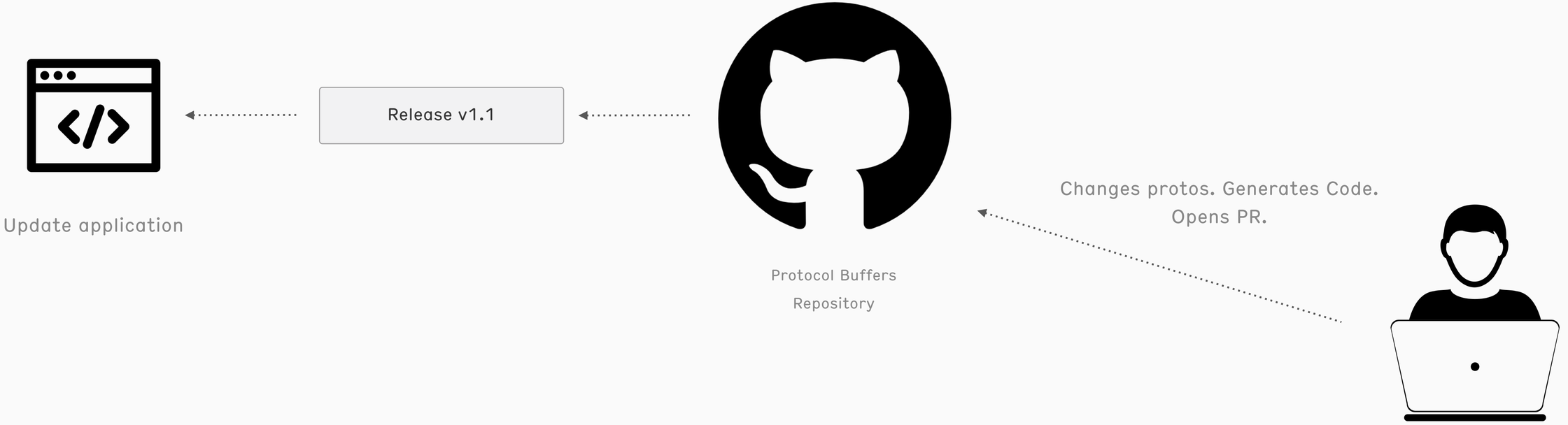
Path to `.proto` file(s) to generate code from.

👷 Demo

Generating code with protoc

Our First Iteration

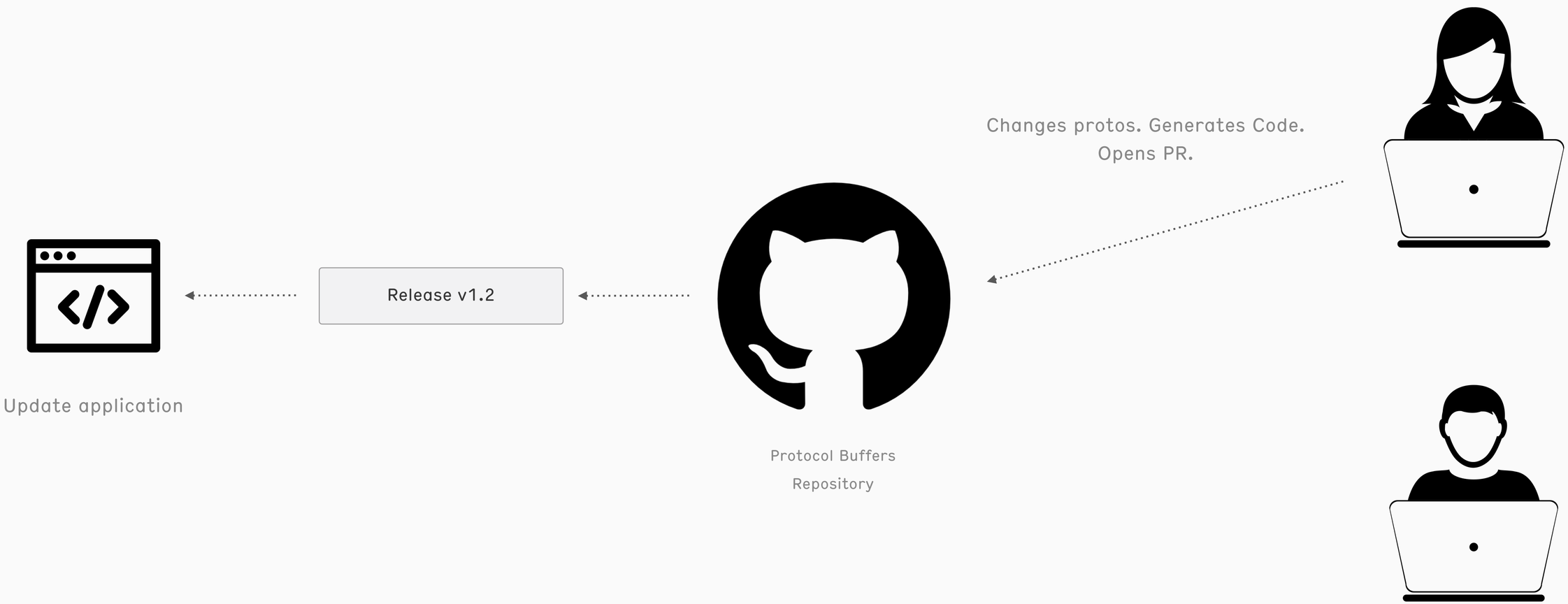
Baby Steps



Images from Lagot Design & TukTuk Design

Our First Iteration

Baby Steps



Images from Lagot Design & TukTuk Design

Chapter III
Our hero needs a partner

Problems we faced scaling up

- `protoc` is a complicated tool
- Complex `Makefile` targets
- Complexity gets harder over time as the company scales
- No Linting
- No backwards / forwards compatibility checks
- No dependency management
- Differences in versions of `protoc` / `proto-gen-*` can result in changes in the generated code.

```
$ protoc \
    -I proto \
    -I vendor/protoc-gen-validate \
    --go_out=. \
    --go_opt=paths=source_relative \
    --go-grpc_out=. \
    --go-grpc_opt=paths=source_relative \
    $(find . -type f -name '*.proto')
```

A linting tool: https://github.com/yoheimuta/protolint

They have thought about Protocol Buffers
So we don’t have too 🎉

A set of tools for Protocol Buffers

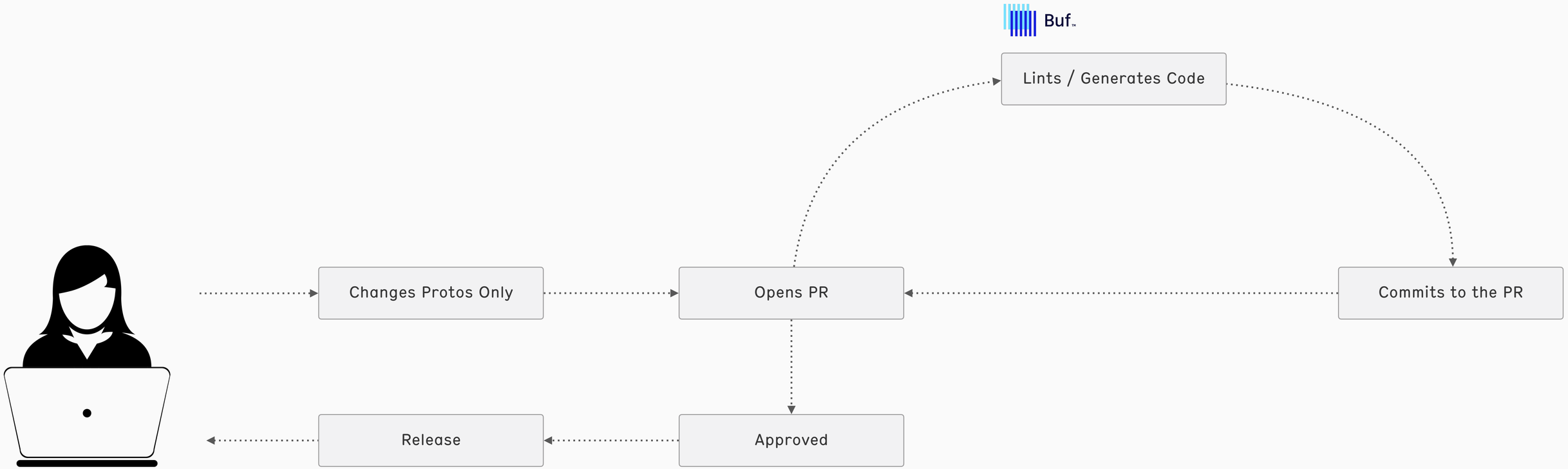
- Linting
- Breaking Change detection
- Code Generation
  - Taking the pain of protoc away 🎉
  - Managed mode
    - Remote code generation
- Easier multi language support
- Schema / Plugin Registry (think Github for Protocol Buffers) ✨
- Dependency Management 🥳

CONSISTENCY

🤦 Demo

Using Buf locally

Level up with CI/CD



👷 Demo

Using Buf™ Github Actions

Chapter IV
The future is bright

Plugins - proto-gen-constants

Contributed by Amelia on the Payments Team

```yaml
version: v1
plugins:
  - remote: buf.build/banked/plugins/goconstants
    out: go
    opt:
      - module=github.com/super/secret/go
      - paths=import
```

```go
package payment

const (
   MessageTypeErrInsufficientFunds   = "banked.payment.v1.ErrInsufficientFunds"
   MessageTypeFoo                    = "banked.payment.v1.Foo"
   MessageTypeBar                    = “banked.payment.v1.Bar"
)
```

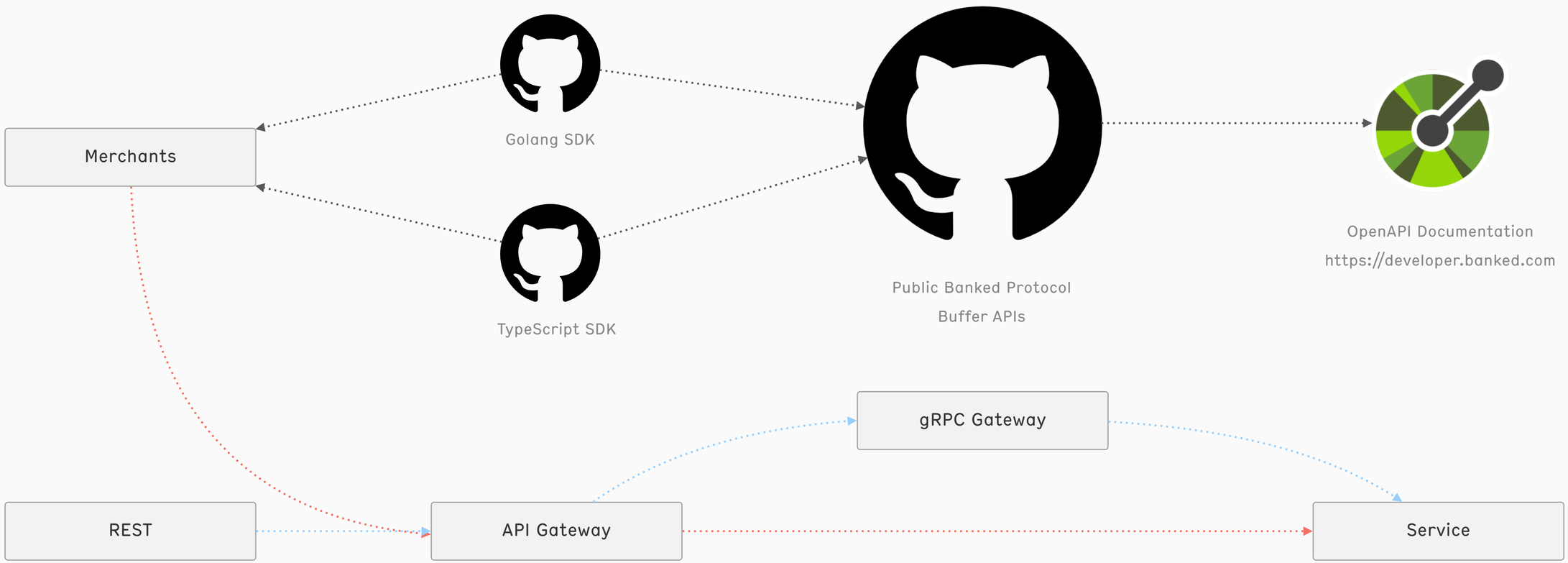
Plugins - grpc-gateway

```yaml
version: v1
plugins:
  - name: grpc-gateway
    out: gen/go
    opt:
      - paths=source_relative
      - generate_unbound_methods=true
```

```
service FooService {
    rpc CreateBar(CreateBarRequest) returns (CreateBarResponse) {
        option (google.api.http) = {
            post: "/v1/bar"
            body: "*"
        };
        option (grpc.gateway.protoc_gen_openapiv2.options.openapiv2_operation) = {
            tags: "Foo"
        };
    }
}
```

Package Organisation & Schema Registry

Using Protocol Buffers for Public APIs



Events and Envelopes

```proto
// Trace holds tracing data for the message envelope.
message Trace {
  // The Trace ID
  string id = 1;
  // The Trace Span ID
  string span_id = 2;
  // Indicates if the span should be sampled
  bool sampled = 3;
}

// Envelope is a common message envelope used to send all messages within the banked platform.
message Envelope {
  // The unique message ID.
  string id = 1;
  // Trace context
  Trace trace = 2;
  // Message creation time
  google.protobuf.Timestamp created_at = 3;
  // Message payload
  google.protobuf.Any payload = 4;
}
```

Wrapping Up

Thank you :)

Contact
chris@banked.com
Chris Reeves — Tech Lead (Kuiper Team)