TYPES ALL THE WAY DOWN

PROTOCOL BUFFERS, GRPC & GO

HI!

Chris Roche

Client/Server Networking

Formerly VSCO

@rodaine

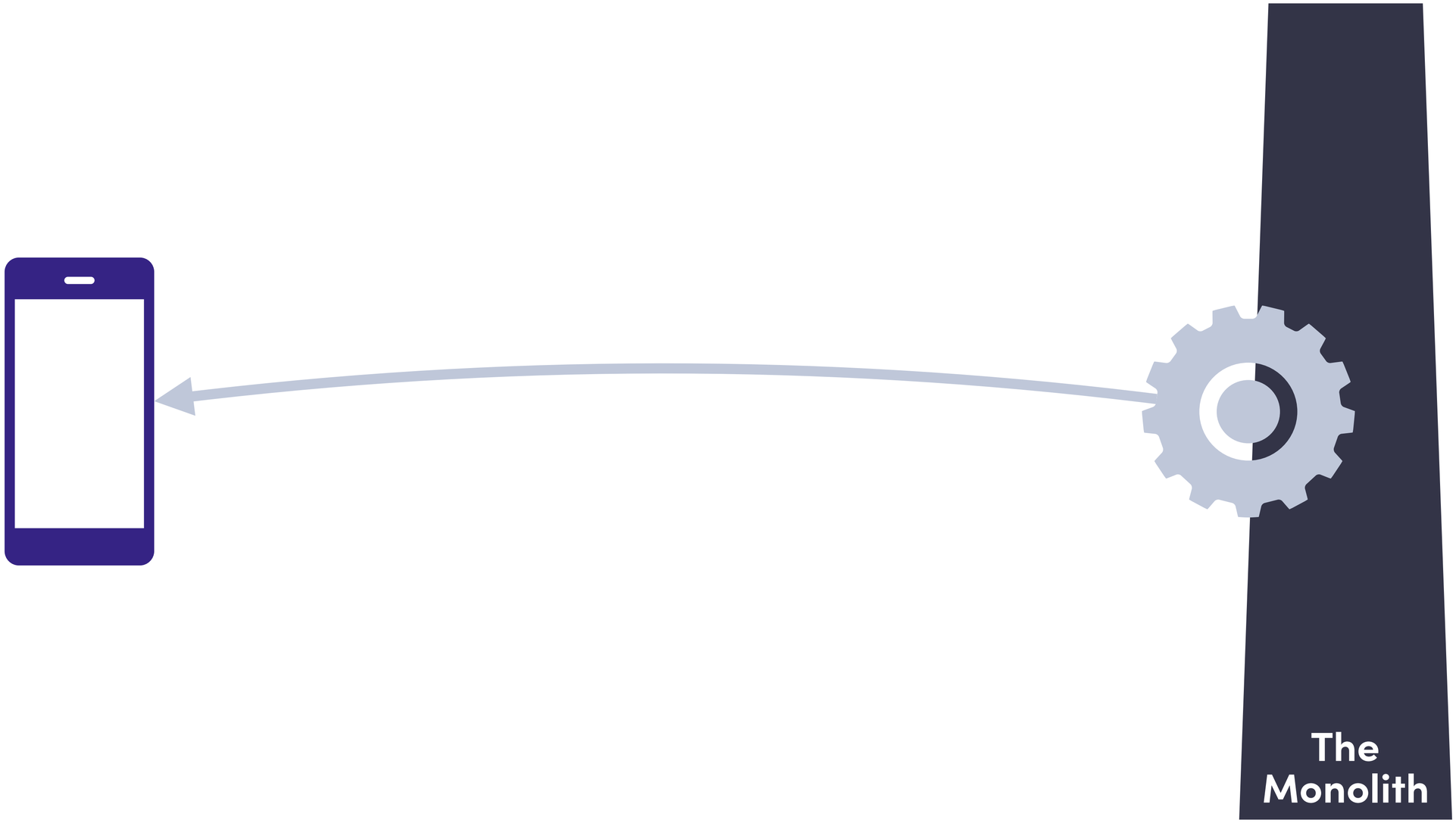
A LITTLE HISTORY

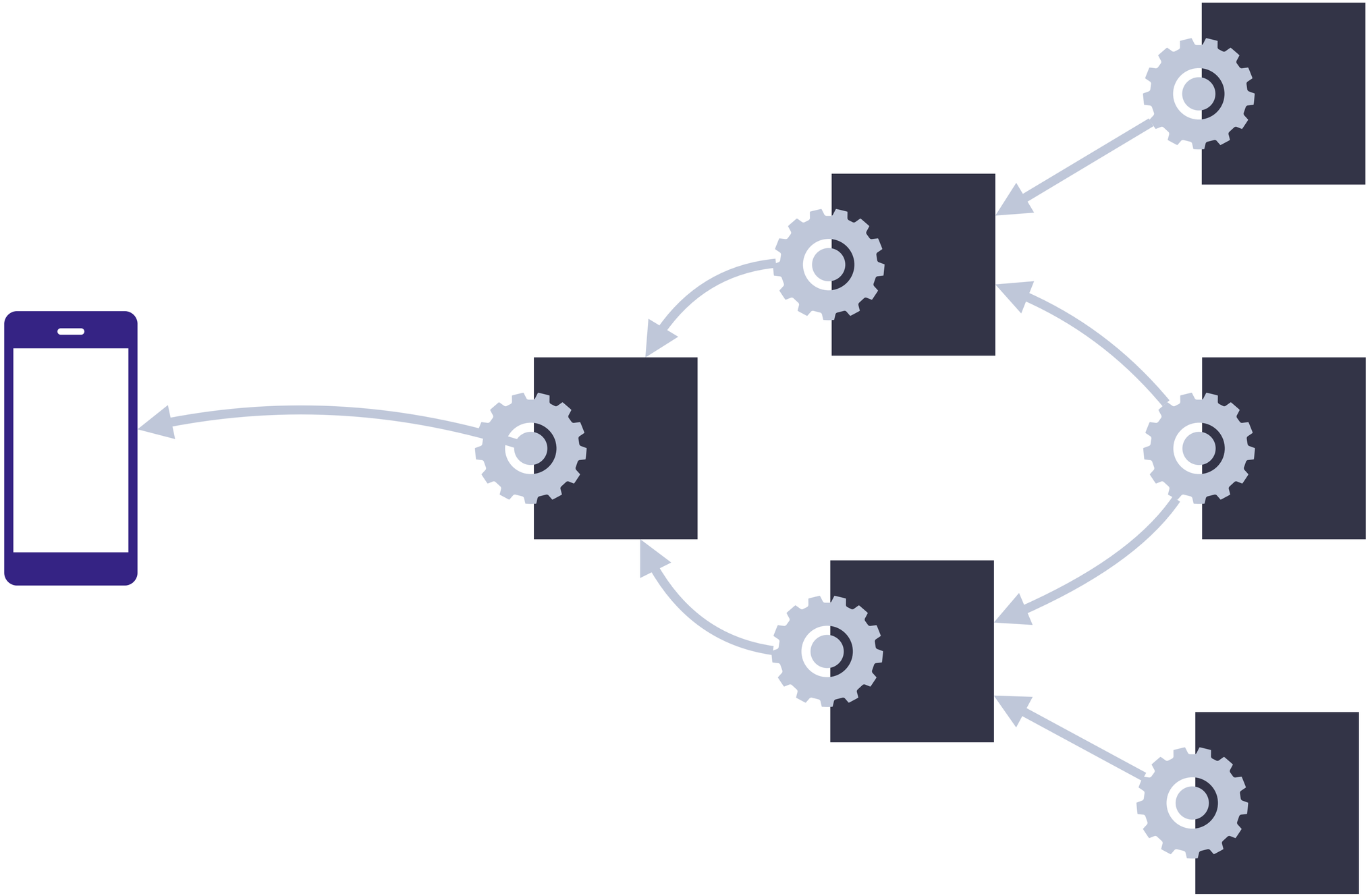
Like any good story we begin with a PHP monolith...

- Active decomp efforts
- 100s of Python Microservices
  - Flask HTTP/REST

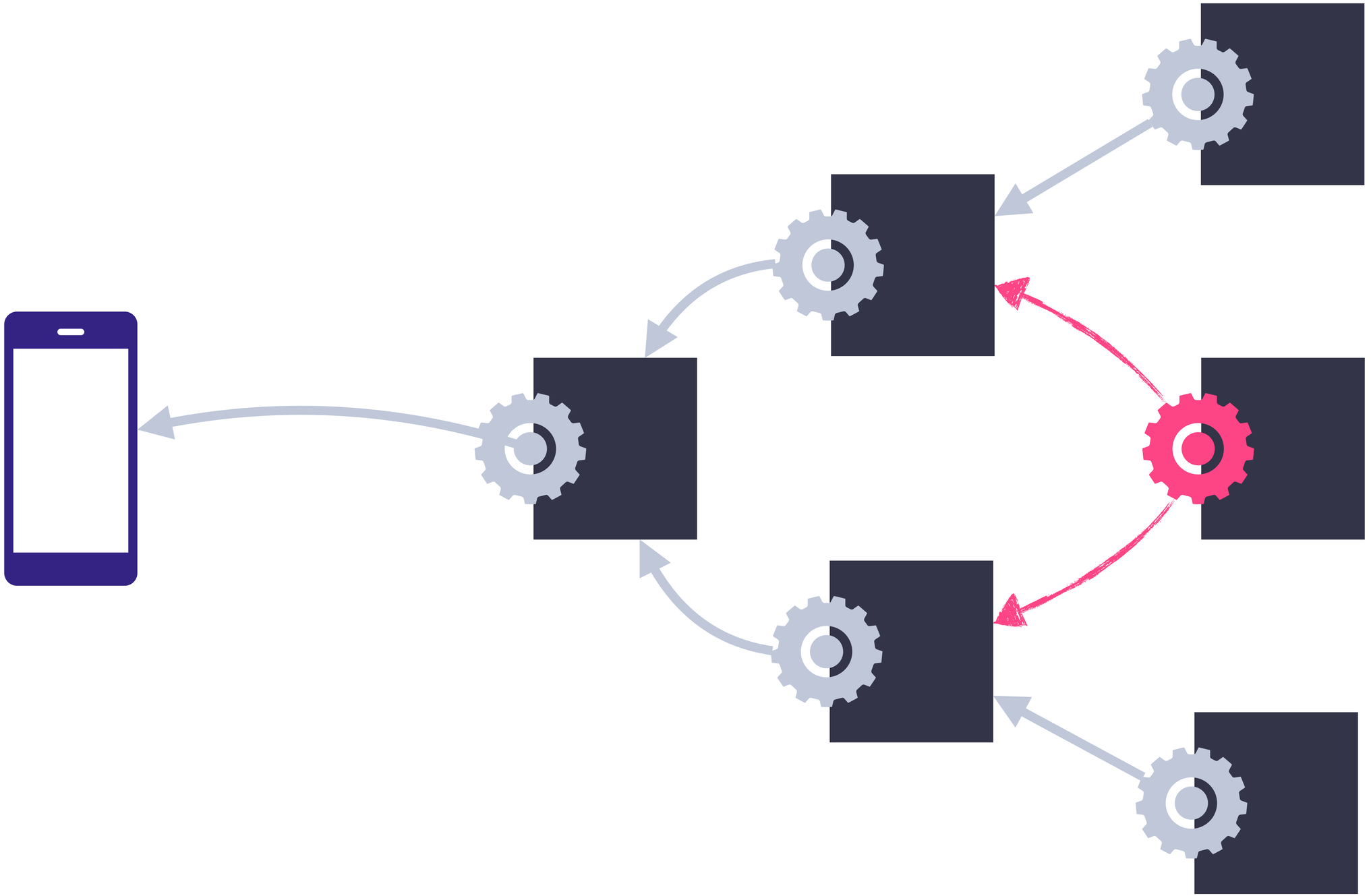
And to keep things interesting...

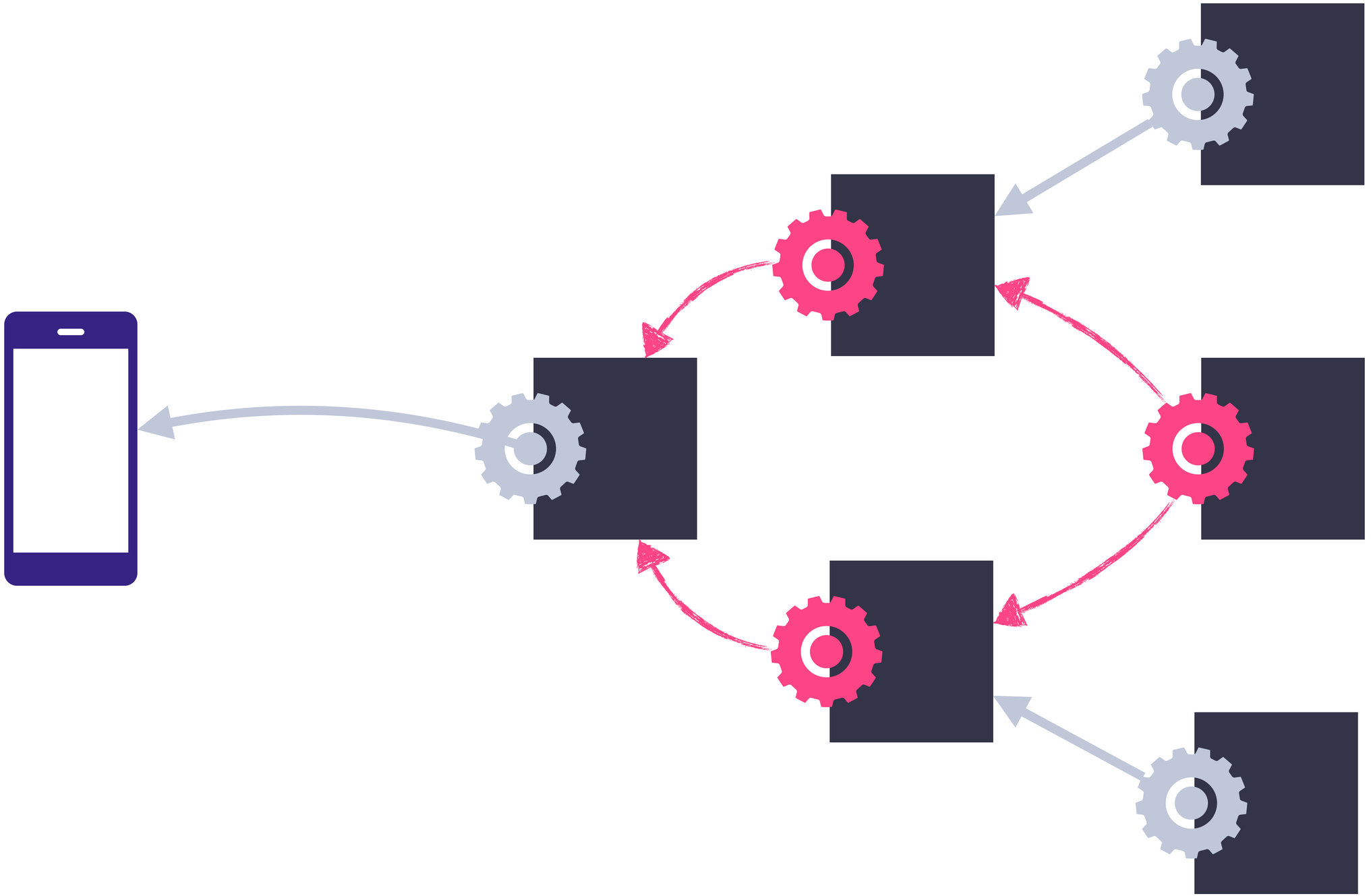
- Go Core and Compositional Services

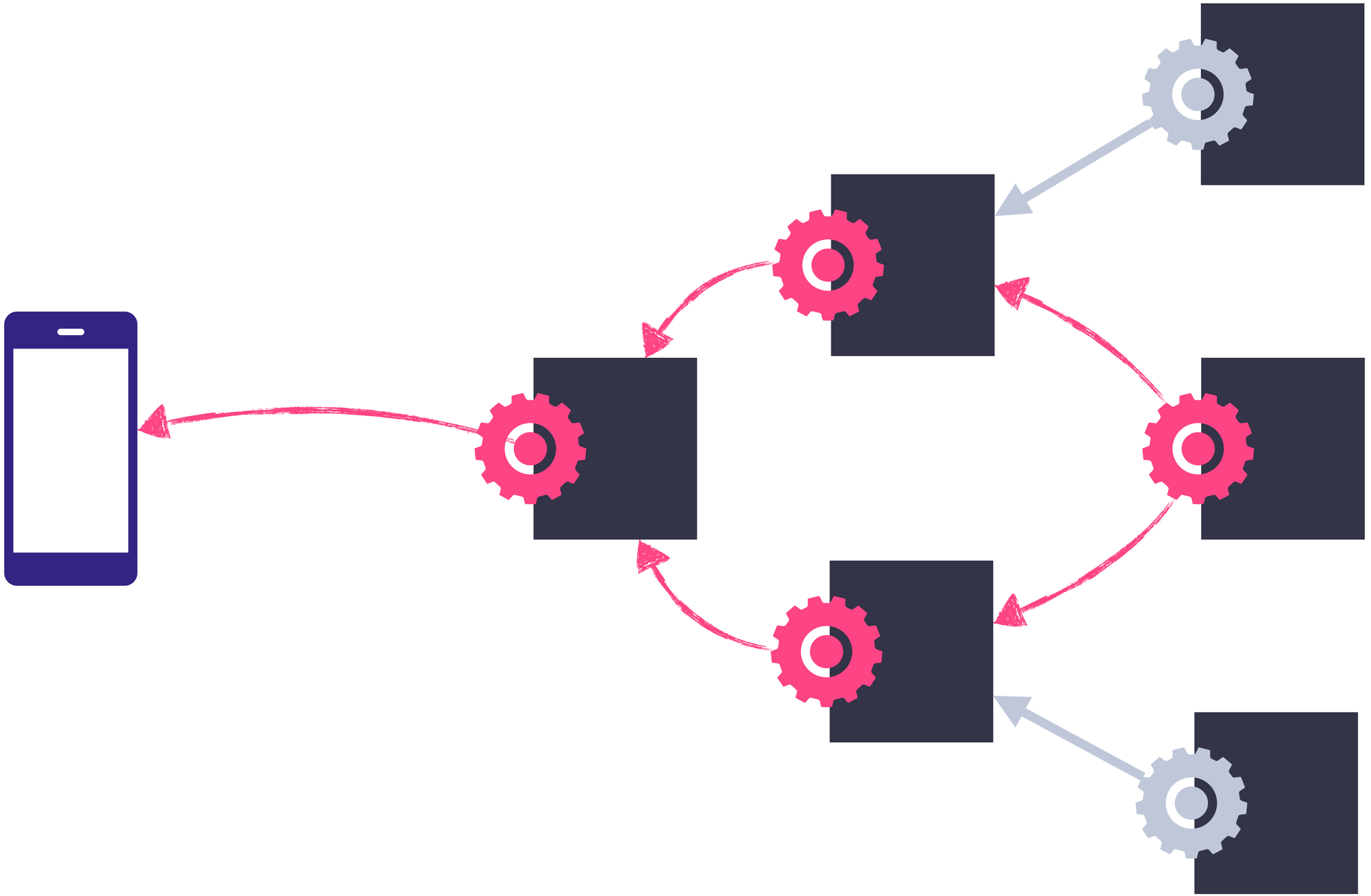

The
Monolith



TypeError: unsupported operand type(s) for -: 'string' and 'float'








pd

CONTRACT BETWEEN CLIENTS & SERVERS

PROTOCOL BUFFERS

- Fully-Typed Interface Definition Language (IDL)
- Binary Wire Format
- Extensible
- Language-Agnostic
- Backwards/Forwards Compatible

PROTOCOL BUFFERS

```
package users;

message User {
  Name   name   = 1;
  Status status = 2;

  repeated uint64 vehicle_ids = 3;

  oneof contact {
    string phone_number = 4;
    string email_addr   = 5;
  }
}
```

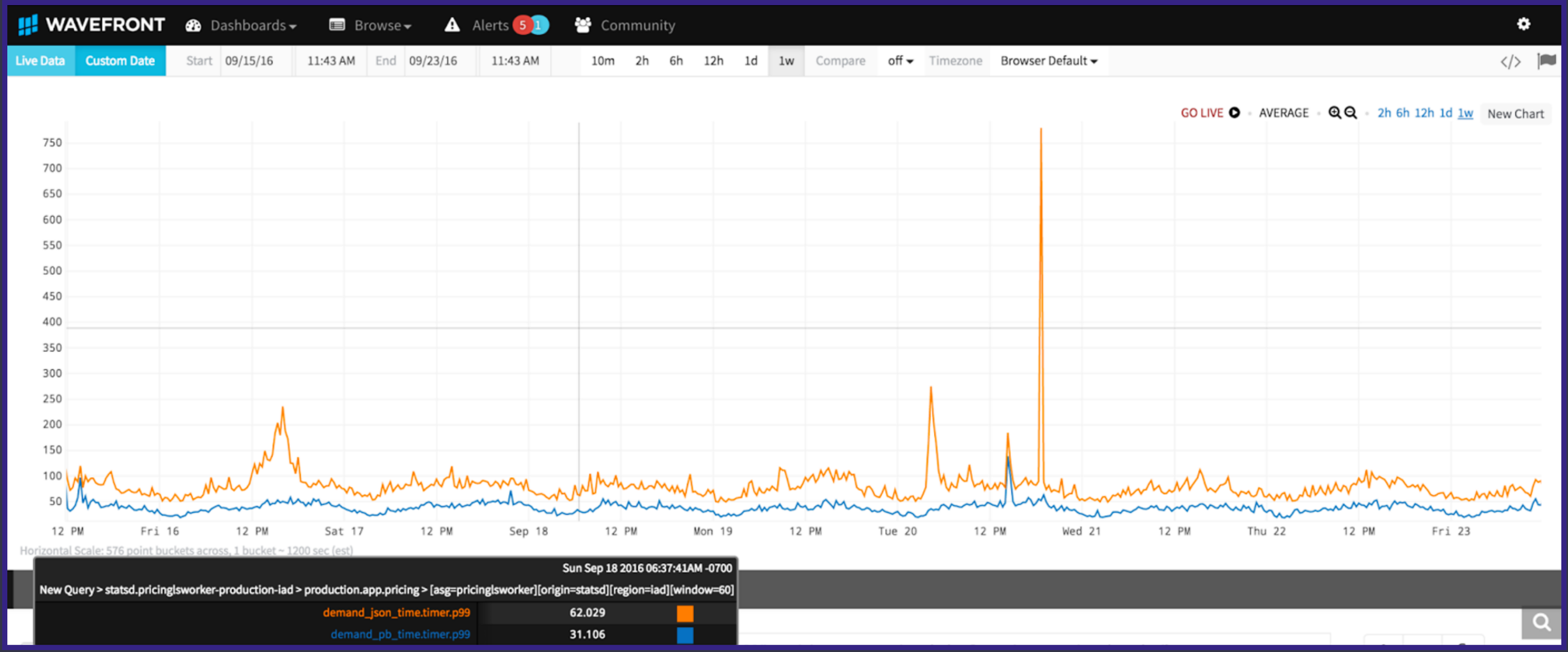
```
message Name {
  string first  = 1;
  string middle = 2;
  string last   = 3;
}

enum Status {
  ACTIVE    = 0;
  INACTIVE  = 1;
  SUSPENDED = 2;
}
```

GENERATED CODE

```go
type User struct {
 Name        *Name
 Status      Status
 VehicleIds []uint64
 Contact     isUser_Contact
}

type Name struct {
 First  string
 Middle string
 Last   string
}
```

```go
type Status int32

const (
 ACTIVE    Status = 0
 INACTIVE  Status = 1
 SUSPENDED Status = 2
)

type isUser_Contact interface {
 isUser_Contact()
}
```

TRANSFER COST



Types on the wire eliminate an entire class of errors

Can we do the
same for the API itself?

EVERY TEN YEARS...

A furious bout of language and protocol design takes place and a new distributed computing paradigm is announced that is compliant with the latest programming model.

– *A Note On Distributed Computing, Waldo 1994*

LOOK FAMILIAR?

- CORBA
- Thrift
- SOAP
- WDDX
- JSON-RPC
- XML-RPC
- Avro
- HyperMedia
- REST
- MessagePack

GRPC!

JSON:PB :: REST:GRPC

BUT FIRST...
LET'S TALK ABOUT REST

RESTful

RESTish

REST/JSON: S2S COMMUNICATION

```
POST /api/updateUser HTTP/1.0
Content-Type: application/json

{
  "id": 18446744073709551615,
  "username": “chris"
}
```

ALRIGHT, LET’S PAINT THAT SHED...

```
PUT /api/users HTTP/1.0
Content-Type: application/json

{
  "id": 18446744073709551615,
  "username": “chris"
}
```

PUTTING ON ANOTHER COAT...

```
PUT /api/users/18446744073709551615 HTTP/1.0
Content-Type: application/json

{
  "username": “chris"
}
```

FINISHING TOUCHES...

```
PUT /api/v1/users/18446744073709551615 HTTP/1.0
Content-Type: application/json

{
  "username": “chris"
}
```

gRPC takes the argument away

IDL SERVICE DEFINITION

```
package lyft.service.users.v1

service Users {
  rpc Update(UpdateRequest) returns (UpdateResponse);
}

message UpdateRequest {
  uint64 id   = 1;
  string name = 2;
}
```

GENERATED CODE

```
type UsersServer interface {
 Update(context.Context, *UpdateRequest)↵
    (*UpdateResponse, error)
}

func RegisterUsersServer(s *grpc.Server, srv UsersServer)

type UsersClient interface {
 Update(context.Context, *UpdateRequest)↵
    (*UpdateResponse, error)
}

func NewUsersClient(c *grpc.ClientConn) UsersClient
```

HTTP/2.0

HTTP/2.0

- Full Duplex Streaming
- Binary Transport
- Push
- Header Compression

IDLS ARE PRETTY GREAT

- Single Source of Truth
  - Primitive definitions
- Code Generation
  - APIs, Clients, Servers, Data Models, Docs, Observability
- Extensibility
  - Plugins for everything else

WHAT'S NOT SO GREAT?

Introducing a new protocol or language can be highly traumatic for teams.

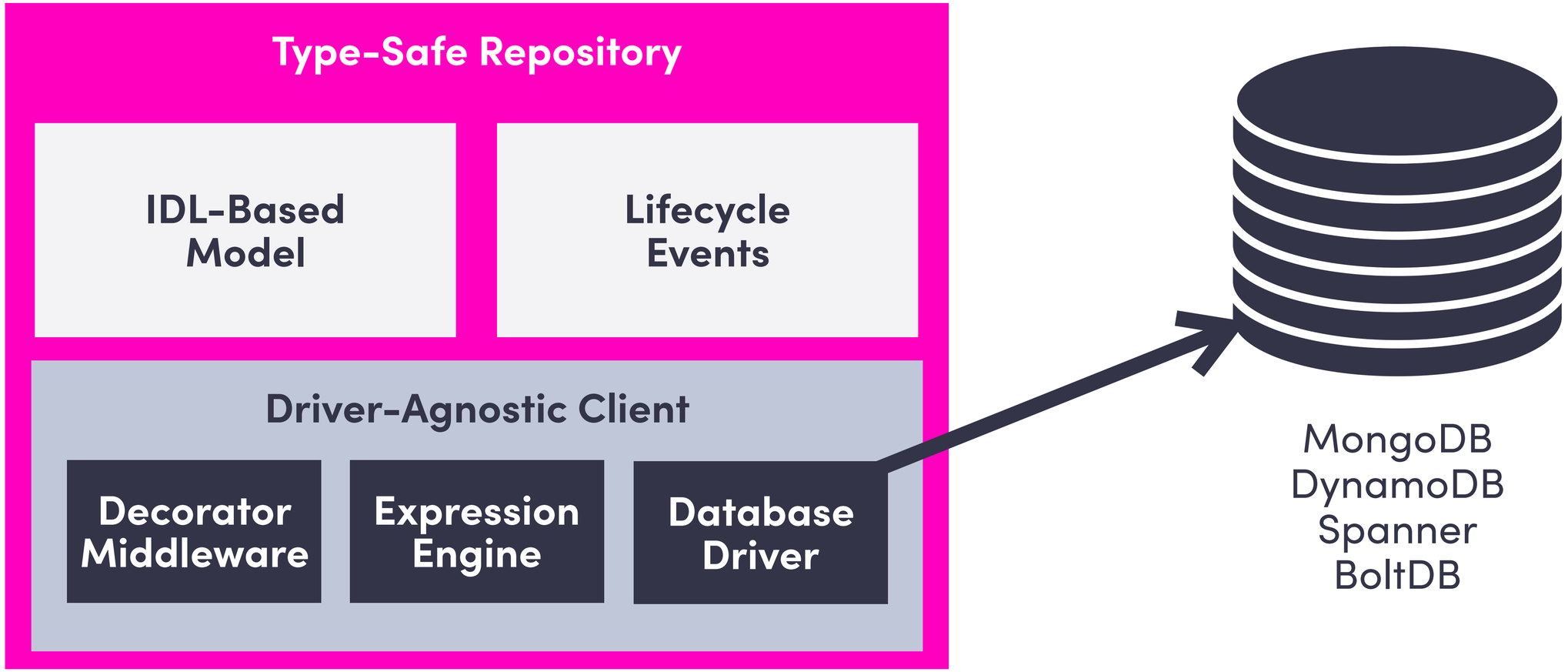
“How do I cURL this?”

WHAT CAN MAKE THIS BETTER?

- Incremental Adoption
  - Allow teams to opt-in to the new shiny things
- Familiarity
  - Tooling that feels welcoming
  - Standardized framework patterns
- Roll Forward
  - Wire format first, then the protocol and frameworks

How can we leverage IDLs beyond the API?

ODIE: IDLs MEET THE DATASTORE



ODIE: MODELS AS PROTOCOL BUFFERS

```
message User {
  option (odie.mongo).enabled = true;

  string id    = 1 [(odie.mongo).primary   = true,
                    (odie.type).object_id = true];

  string name = 2 [(odie.mongo).name          = "username"];
  int64  date = 3 [(odie.type).datetime       = true];
  uint32 vers = 4 [(odie.locking).revision = true];
}
```

ODIE: MODELS AS PROTOCOL BUFFERS

```
type UserModel struct {
  Id    bson.ObjectId `bson:"_id"`
  Name  string        `bson:"username"`
  Date  time.Time
  Vers  uint32
}

func (pb *User) ToModel() *UserModel
func (m *UserModel) ToProto() *User
```

ODIE: TYPE-SAFE REPOSITORIES

```
type UserRepo interface {
 Events() *Events

 Get(ctx context.Context, id bson.ObjectId) *GetBuilder
 Put(ctx context.Context, m *UserModel) *PutBuilder
 Delete(ctx context.Context) *DeleteBuilder
 Update(ctx context.Context) *UpdateBuilder
 Query(ctx context.Context) *QueryBuilder
}
```

FURTHER CODE GENERATION

- JSON Schema-esque validation
- PB over HTTP clients/server
- Observability interceptors
- Ergonomics helpers
- Response caching
- CLI

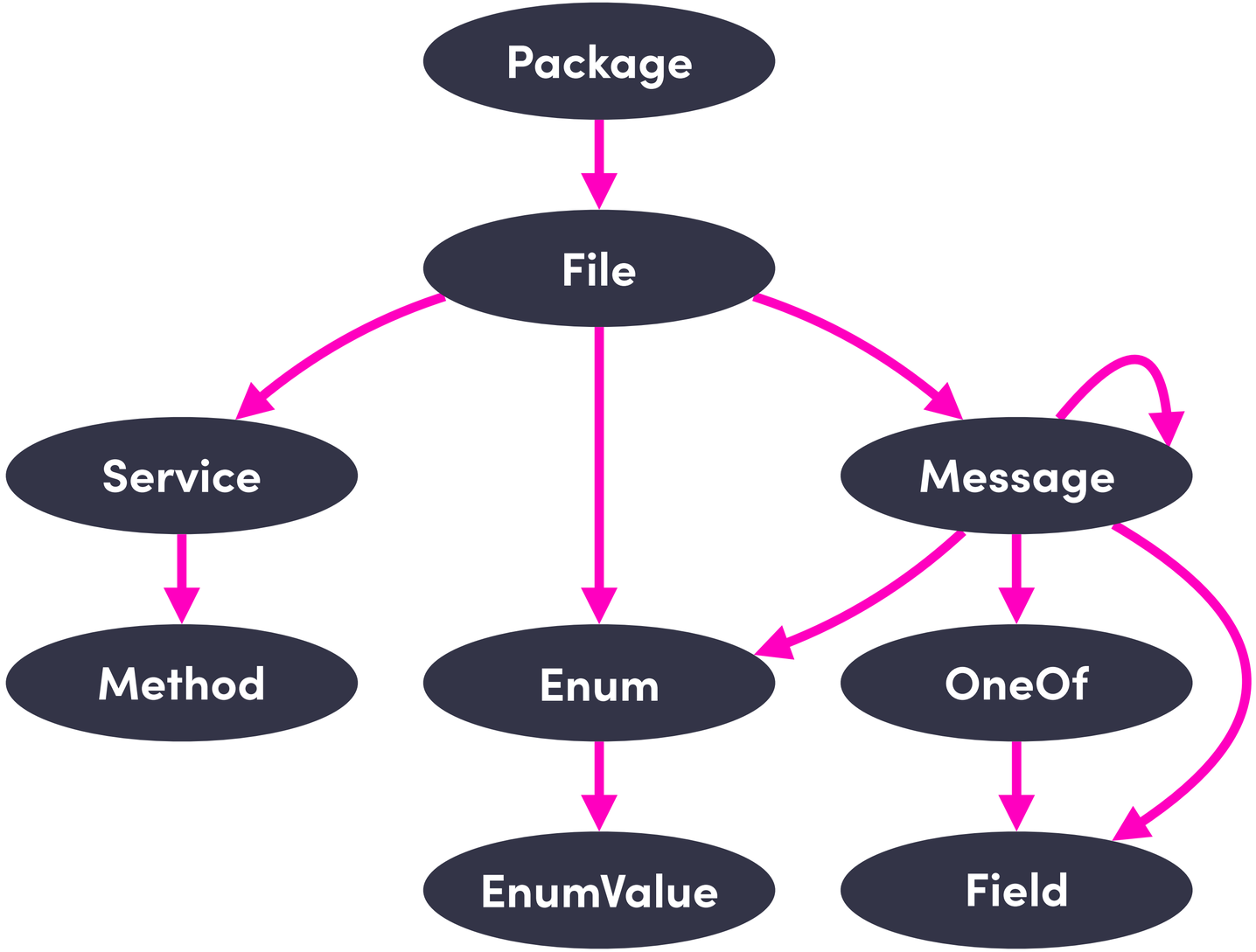
github.com/lyft/protoc-gen-validate

That's an awful lot of codegen...

PROTOC-GEN-STAR (PG*)

Code generation framework

- AST of primitives
- Simplifies code generation
- Highly testable



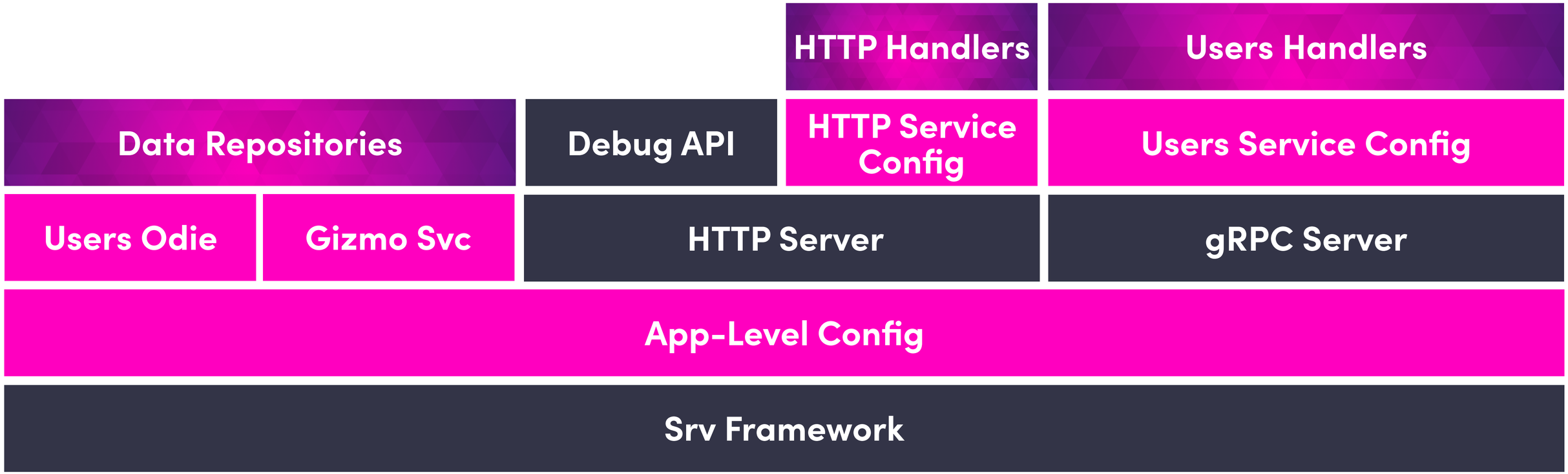
github.com/lyft/protoc-gen-star

PG*: WALK THE AST

```
type Visitor interface {
 VisitPackage(Package) (v Visitor, err error)
 VisitFile(File) (v Visitor, err error)
 VisitMessage(Message) (v Visitor, err error)
 VisitEnum(Enum) (v Visitor, err error)
 VisitEnumValue(EnumValue) (v Visitor, err error)
 VisitField(Field) (v Visitor, err error)
 VisitOneOf(OneOf) (v Visitor, err error)
 VisitService(Service) (v Visitor, err error)
 VisitMethod(Method) (v Visitor, err error)
}
```

How far can we take this?

SERVICE GENERATION



FUTURE TOOLS

Linting & Static Analysis

- Enforce best practices
- Protect production code
- Networking ≠ IDL Police

Mocks & Test Fixtures

- Scenarios of valid state
- Reduce reliance on integration tests
- Developer confidence

gRPC on Mobile

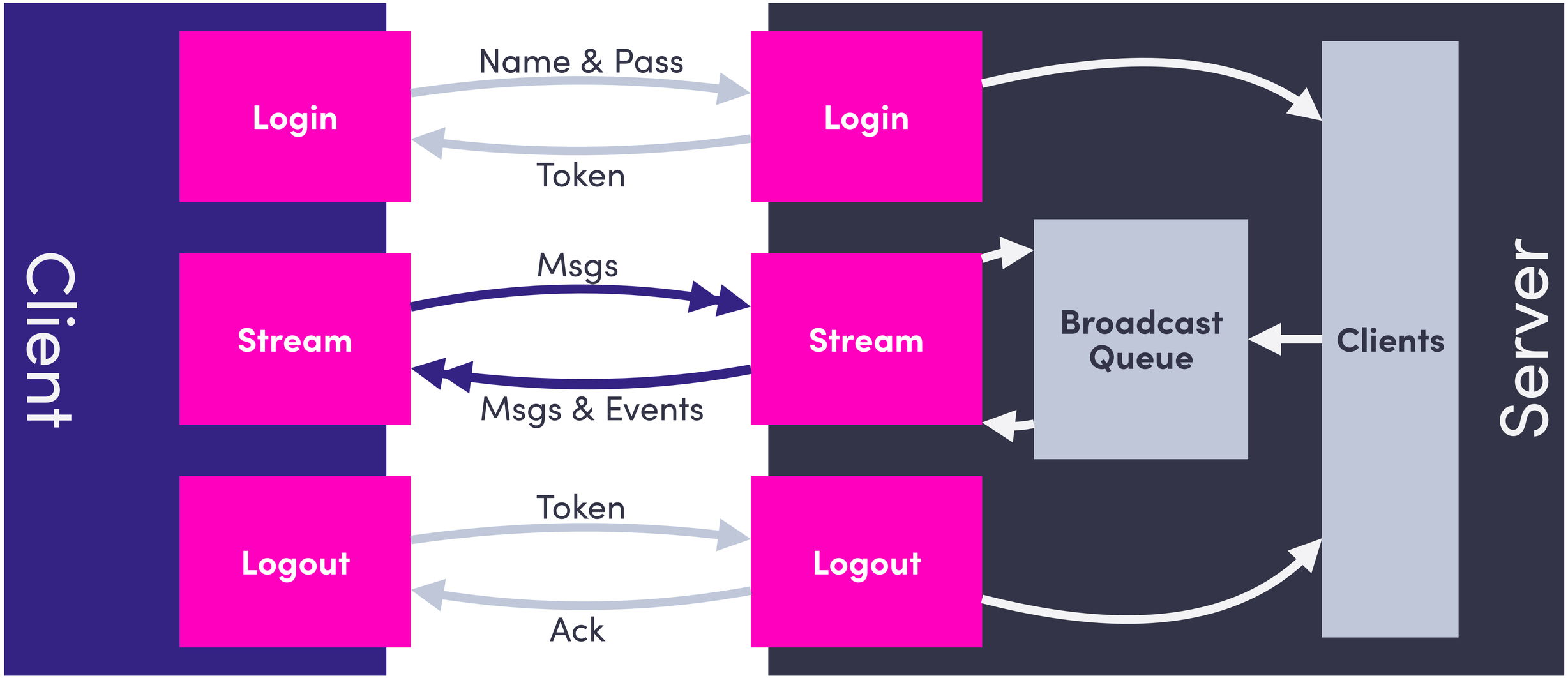
- Reduced payload size
- Leverage streaming APIs
- Global consistency

Live & Interactive (optionally) Demo

gRPC Chat

github.com/rodaine/grpc-chat

ARCHITECTURE



SERVICE DEFINITION

```
syntax = "proto3";

package chat;

import "google/protobuf/timestamp.proto";

service Chat {
 rpc Login(LoginRequest) returns (LoginResponse) {}
 rpc Logout(LogoutRequest) returns (LogoutResponse) {}
 rpc Stream(stream StreamRequest) returns (stream StreamResponse) {}
}
```

LOGIN

```
message LoginRequest {
    string password = 1;
    string name     = 2;
}

message LoginResponse {
    string token = 1;
}
```

```go
func (s *server) Login(ctx context.Context, req
*chat.LoginRequest) (*chat.LoginResponse, error) {

  // validate username & password

  tkn := s.genToken()
  s.setName(tkn, req.Name)

  s.Broadcast <- s.loginEvent(req.Name)

  return &chat.LoginResponse{Token: tkn}, nil
}
```

LOGOUT

```
message LogoutRequest {
    string token = 1;
}

message LogoutResponse {}
```

```go
func (s *server) Logout(ctx context.Context, req
*chat.LogoutRequest) (*chat.LogoutResponse, error) {

name, ok := s.delName(req.Token)
 if !ok {
   return nil, status.Error(codes.NotFound, "token not found")
 }

 s.Broadcast <- s.logoutEvent(name),

 return new(chat.LogoutResponse), nil
}
```

STREAM RESPONSE

```
message StreamResponse {
 google.protobuf.Timestamp timestamp = 1;

 oneof event {
  Login     client_login    = 2;
  Logout    client_logout   = 3;
  Message   client_message  = 4;
  Shutdown  server_shutdown = 5;
 }

 message Login {
  string name = 1;
 }

 message Logout {
  string name = 1;
 }

 message Message {
  string name    = 1;
  string message = 2;
 }

 message Shutdown {}
}
```

STREAM REQUEST

```
message StreamRequest {
 string message = 2;
}
```

Token?

CLIENT

```
func (c *client) stream(ctx context.Context) error {

  md := metadata.New(map[string]string{tokenHeader: c.Token})
  ctx = metadata.NewOutgoingContext(ctx, md)

  client, err := c.ChatClient.Stream(ctx)
  if err != nil {
    return err
  }
  defer client.CloseSend()

  go c.send(client)
  return c.receive(client)
}
```

SERVER - EXTRACT TOKEN

```
func (s *server) extractToken(ctx context.Context) (tkn
string, ok bool) {

 md, ok := metadata.FromIncomingContext(ctx)

 if !ok || len(md[tokenHeader]) == 0 {
  return "", false
 }

 return md[tokenHeader][0], true
}
```

SERVER

```
func (s *server) Stream(srv chat.Chat_StreamServer) error {

 tkn, ok := s.extractToken(srv.Context())
 // check for tkn
 name, ok := s.getName(tkn)
 // check that tkn is known

 go s.sendBroadcasts(srv, tkn)

 // [ receive loop ]

 return srv.Context().Err()
}
```

SERVER - RECEIVE LOOP

```
for {
 req, err := srv.Recv()

 if err == io.EOF {
  break // client is done streaming
 } else if err != nil {
  return err // unexpected error
 }

 s.Broadcast <- s.messageEvent(name, req.Message),
}
```

```
func (s *server) sendBroadcasts(srv chat.Chat_StreamServer, tkn string) {
 stream := s.subscribe(tkn)
 defer s.unsubscribe(tkn)

 for {
  select {
  case <-srv.Context().Done():
   // client is done streaming
   return
  case res := <-stream:
   if s, ok := status.FromError(srv.Send(&res)); ok {
   switch s.Code() {
    case codes.OK:
     // noop
    default:
     // couldn't send broadcast
     return
 }}}}
}
```

Live & Interactive (optionally) Demo

```
grpc-chat -h chat.rodaine.com:6262 -p "oc-gophers" -n <NAME>
```