Terraform plugin programming is easy

SDKv2 Operations | Passing data clumps

Sign In Register

TIBCO Cloud Mashery API

DOCUMENTATION INTERACTIVE API SEARCH

In the Docs

- API Documentation Portal
- IO Docs Definition ▶
- Mashery API Documentation ▼
  - Overview
  - Mashery 3.0 API Guide ▼
    - Overview
    - Authentication
    - Errors
    - Resource Hierarchy
    - Pagination, Sorting, Filtering
    - Transform API
    - Example Use Cases
    - Resources ▼
      - Overview
      - Members ▶
      - Organizations ▶
      - Domains ▶
      - Services ▶
      - Applications ▶
      - ScheduledMaintenanceEvents ▶

Plan Filters

Resource URI

① ② ③ ④ ⑤

/packages/{packageId}/plans/{planId}/services/{serviceId}/endpoints/{endpointId}/methods/{methodId}/responseFilter

Items in curly braces represent variables.

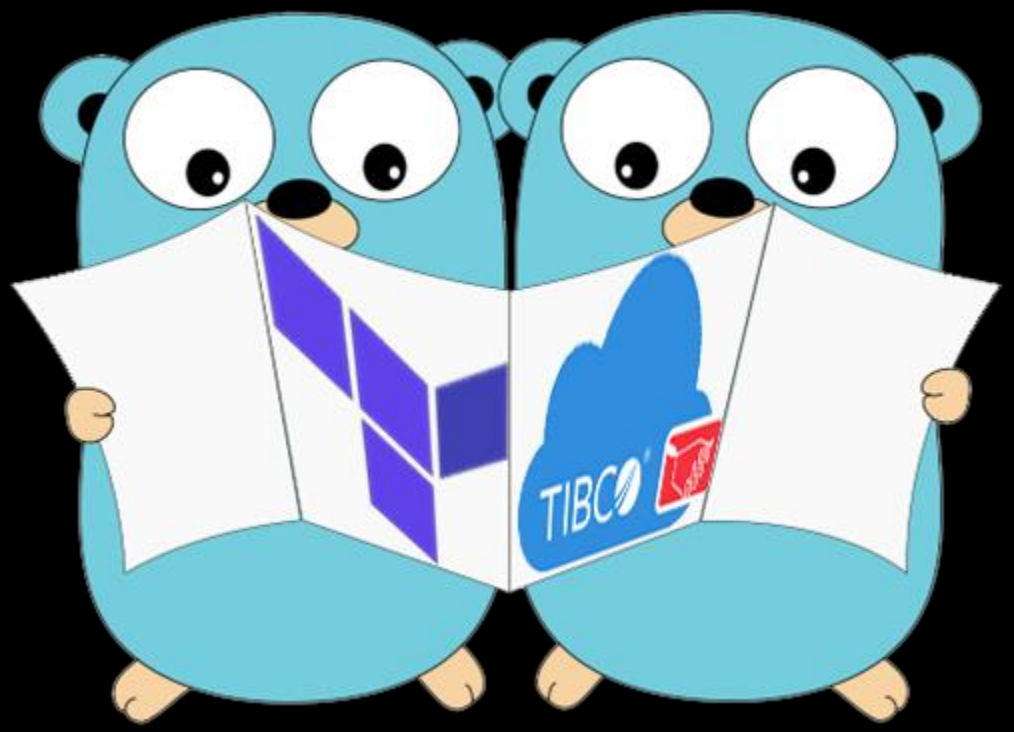
Description

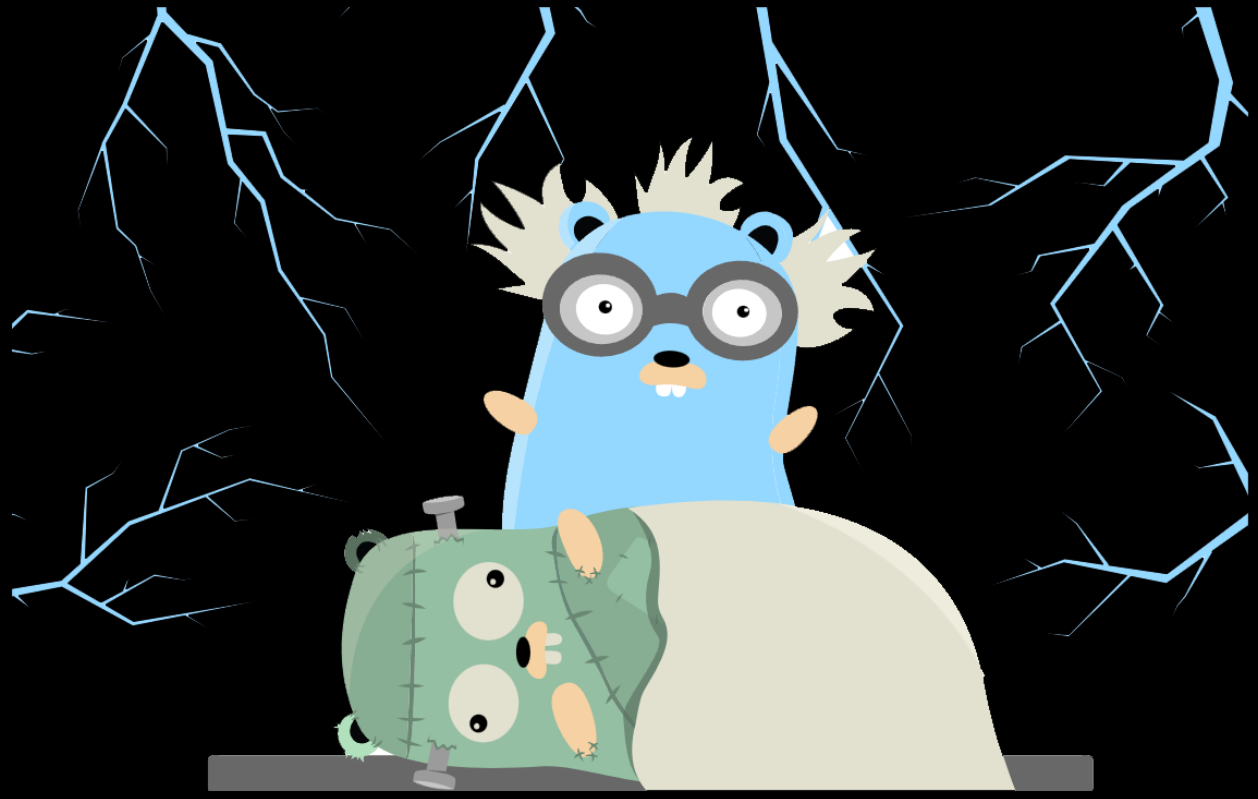
A plan filter represents the relationship between a plan and a response filter. It is created when a response filter is attached to a plan method.

Resource Schema

Property	Characteristics	
id	Type	string
	Sub-type	
	Create Rule	Ignored
	Update Rule	Required (if not in context)
	Fetch Rule	Implicit

My practical problems from the field



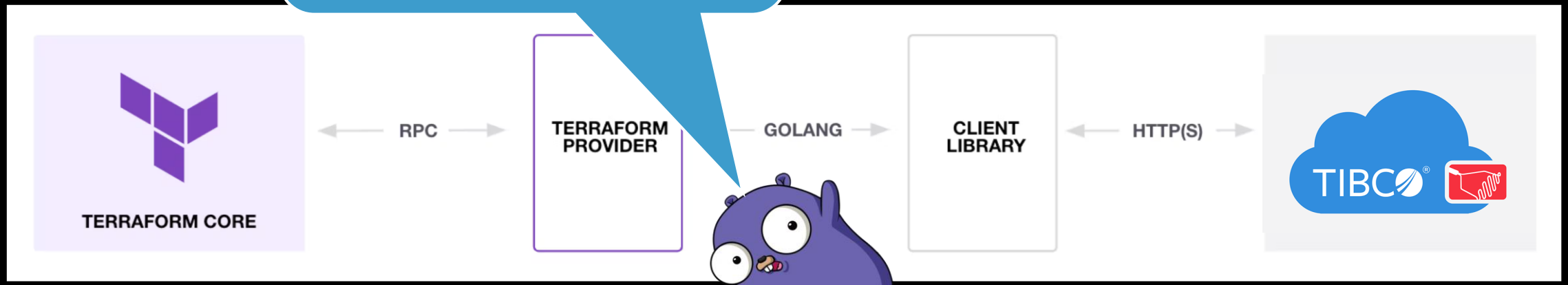


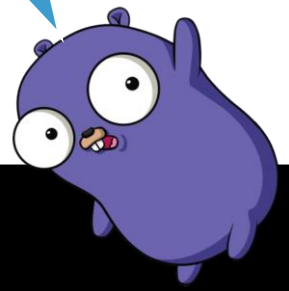
My practical problems from the field

```
// ResourceMapper Base interface for converting HCL constructs into the Mashery V3 object upsertable calls.
type ResourceMapper interface {  4 usages   14 implementations   lspwd2

    // TerraformSchema get the Terraform schema for this resource
    TerraformSchema() TFResourceSchema  14 implementations

    V3ObjectName() string  14 implementations

    // V3Identity Creates the v3Identity for this object. Will return non-nil diagnostics
    // if the object is not positively identified
    V3Identity(d *schema.ResourceData) (V3ObjectIdentifier, diag.Diagnostics)  14 implementations

    // Upsertable creates an update-insertable object that can be passed onto the client. Where applicable,
    // the {@link V3ObjectIdentifier} will contain the identifier of the context
    Upsertable(d *schema.ResourceData) (Upsertable, V3ObjectIdentifier, diag.Diagnostics)  14 impl

    // SetState Set the state the V3 object in the terraform schema
    SetState(rv Upsertable, d *schema.ResourceData) diag.Diagnostics  14 implementations

    TestResourceData() *schema.ResourceData  14 implementations

    TestResourceDataWith(init map[string]interface{}) (*schema.ResourceData, diag.Diagnostics)  14 implemen
}
```

Terraform plugin programming is easy

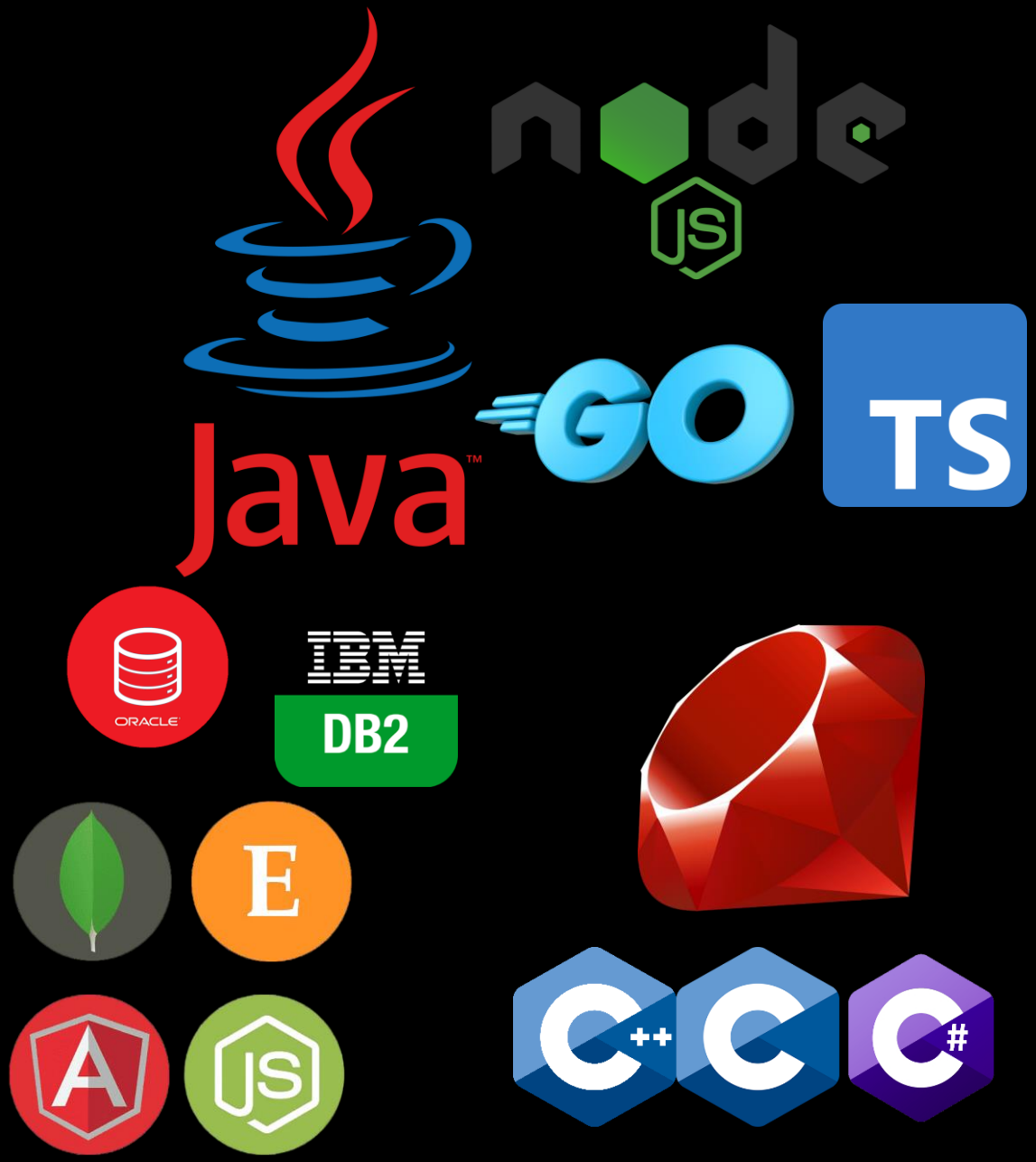
[T]
to the rescue!





Excited me

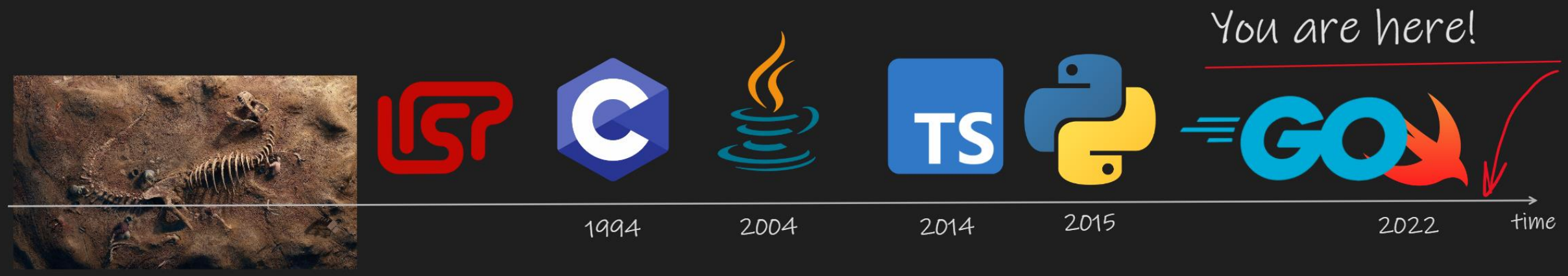
Awesome to have you here!



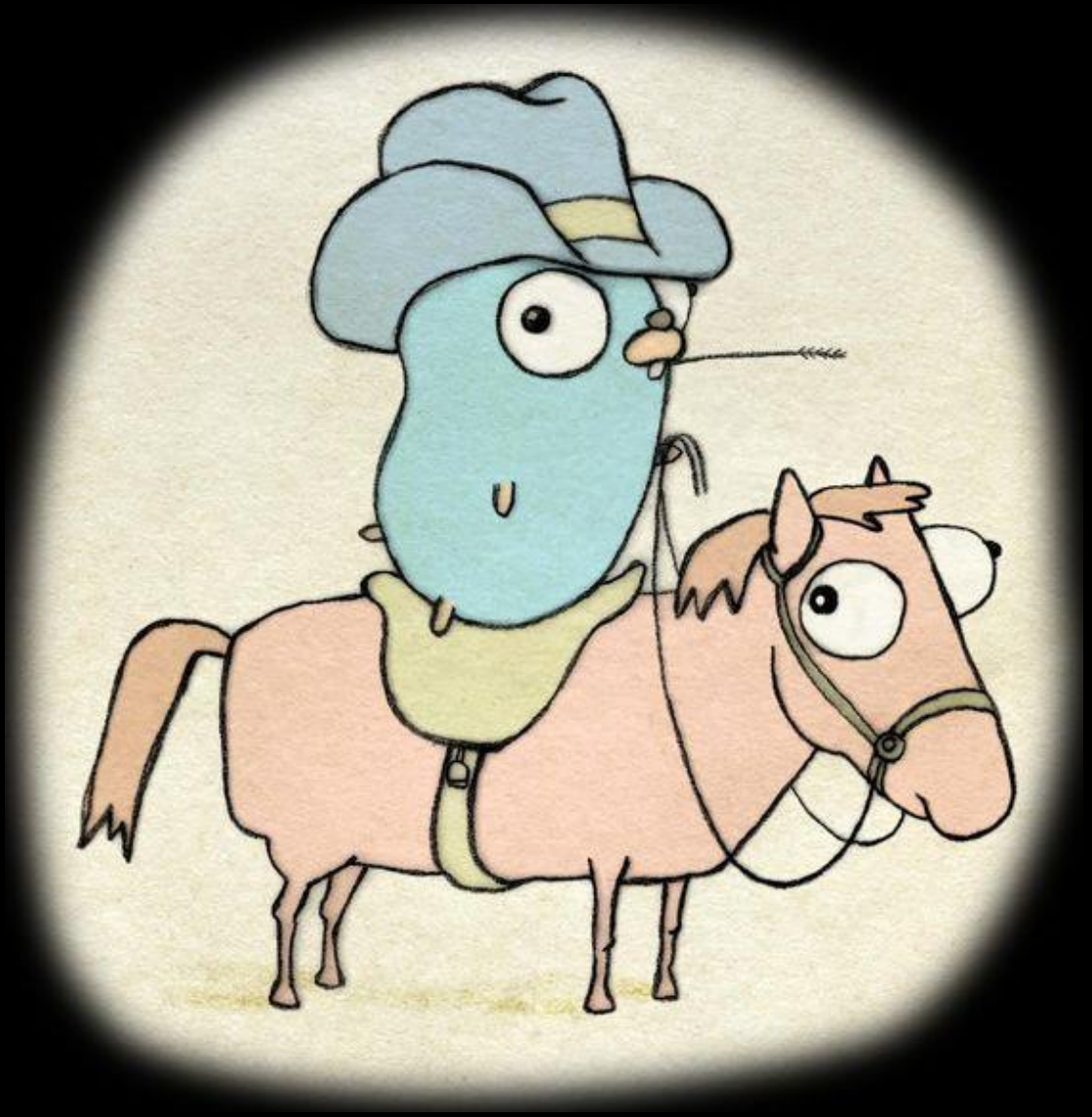
GO Generics

Let GO Generics

New to generics?



There’s <u>nothing</u> fundamentally wrong with `interface{}`



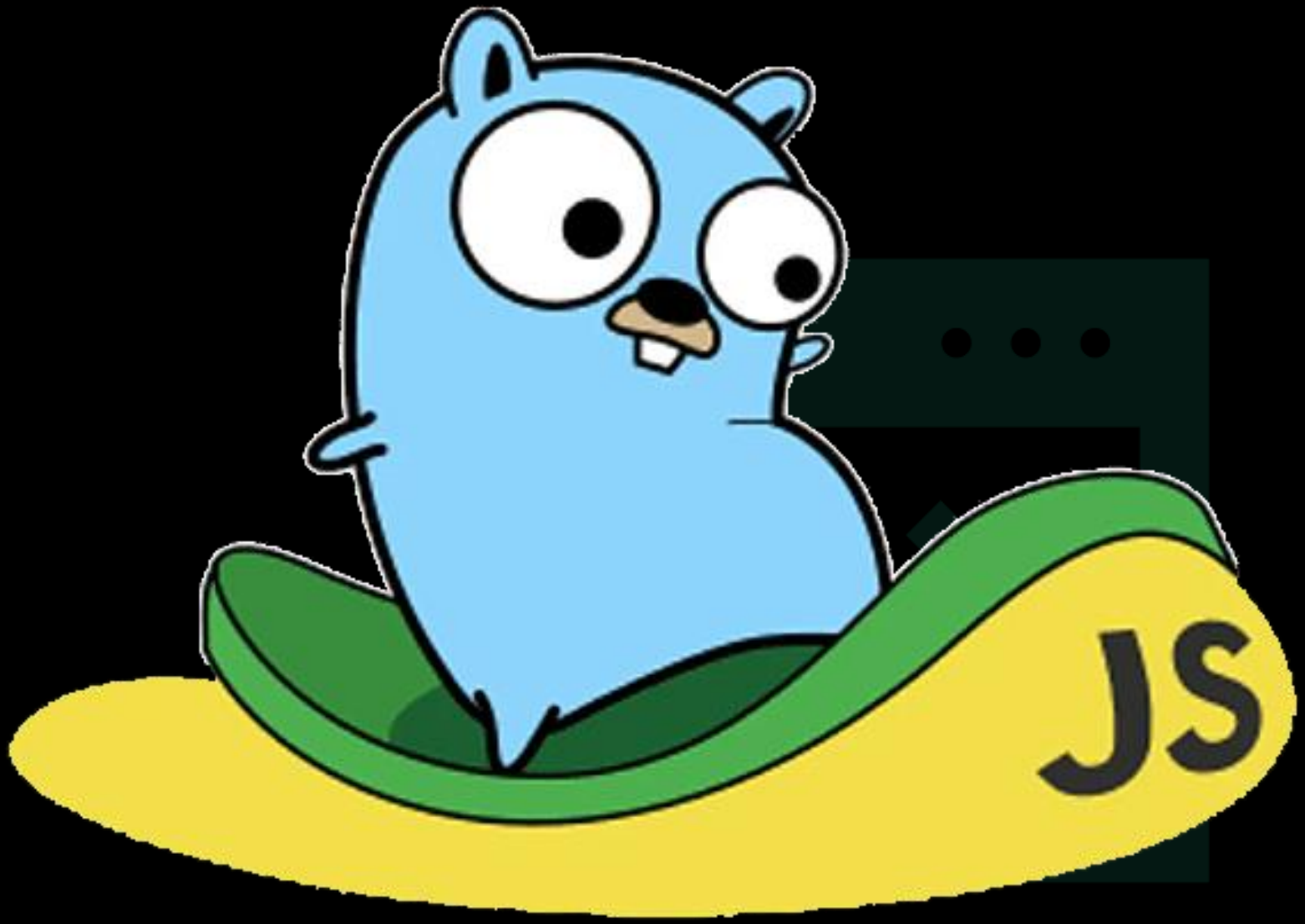
Which very popular language doesn’t use generics and is totally fine?

```
> console.log("11"+1)
111
```

```
> console.log("11"-1)
10
```

http://etc.ch/kQS6

There're lots to improve on `interface{}`



dev meets GO generics

The Generification Workflow

- Model concrete types
- Identify common capabilities
- Build an interface
- Write generic code / refactor into generified code

```java
package nl.yanchuk.golanggenerics;

import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.IOException;
import java.io.Reader;

1 usage
public class JacksonUnmarshal {
    1 usage
    static final ObjectMapper om = new ObjectMapper();
    2 usages
    public static  <T> T unmarshal(Reader r, Class<T> clazz) throws StupidJsonException {
        try {
            return om.readValue(r, clazz);
        } catch (IOException ex) {
            throw new StupidJsonException("promise json, send xml?", ex);
        }
    }
}
```

```java
package nl.yanchuk.golanggenerics;

import lombok.Data;
import lombok.NoArgsConstructor;
import org.junit.jupiter.api.Test;

import java.io.StringReader;

import static nl.yanchuk.golanggenerics.JacksonUnmarshal.unmarshal;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class JacksonUnmarshalTest {
    2 usages
    @Data
    @NoArgsConstructor
    public static class SampleData {
        String a;
        Integer b;
    }

    @Test
    void showHowAwesomeGenericsAndJacksonAre() throws StupidJsonException {
        SampleData t = unmarshal(new StringReader( s: "{\"a\": \"Kramp Hub\", \"b\": 20231010}"), SampleData.class);
        assertEquals( expected: "Kramp Hub", t.getA());
        assertEquals( expected: 20231010, t.getB());
    }
}
```

XDoclet would have been nice

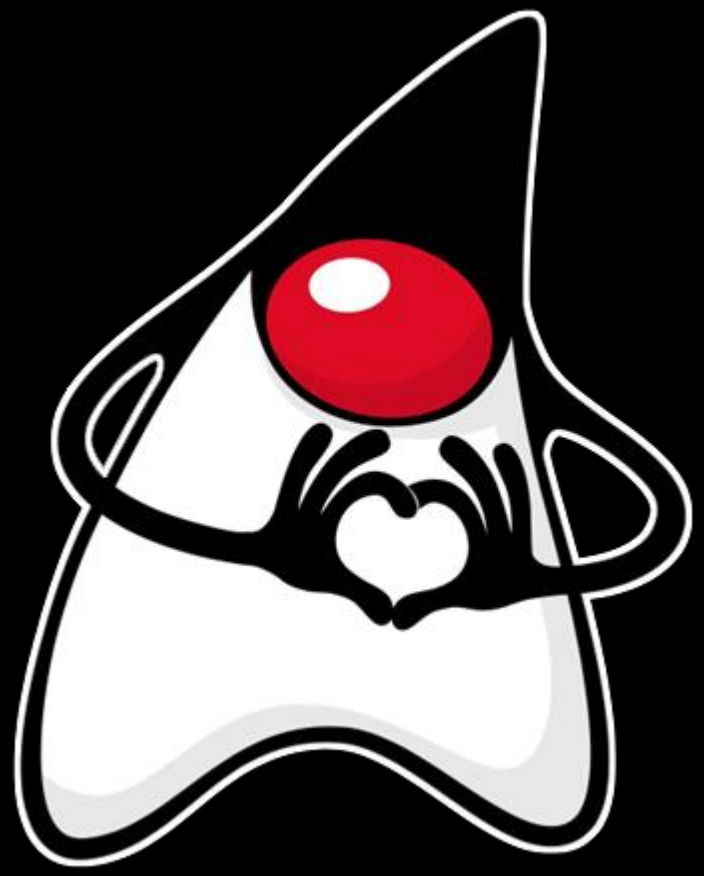
```
te) (interface{}, int, error) {

&rv)

return rv, 1, err
func ParseServiceCache(dat [] (interface{}, int, error) {
    var rv ServiceCache
func ParseMasheryServiceArray(dat []byte) (inter
    var rv []Service
    err := json.Unmarshal(dat, &rv)
    return rv, len(rv), err
}
```

```go
// CreateObject Create a new object in target service.
func (c *V3Transport) CreateObject(ctx context.Context, objIn interface{}, opCtx FetchSpec) (interface{}, error) {  lspwd2
    if resp, err := c.Post(ctx, opCtx.DestResource(), objIn); err == nil {
        if dat, err := resp.Body(); err != nil {
            return nil, &errwrap.WrappedError{
                Context: fmt.Sprintf("create #{opCtx.AppContext}->read server response"),
                Cause:   err,
            }
        } else {
            if resp.StatusCode == 200 {
                // Ignore page size when retrieving an object
                if rv, _, jsonErr := opCtx.ResponseParser(dat); jsonErr != nil {
                    return nil, &errwrap.WrappedError{
                        Context: fmt.Sprintf("create #{opCtx.AppContext}->unmarshal response (#{dat})"),
                        Cause:   jsonErr,
                    }
                } else : rv, nil
            } else {
                return nil, v3ErrorFromResponse(fmt.Sprintf("create #{opCtx.AppContext}"), resp.StatusCode, resp.Header, dat)
            }
        }
    } else {
        return nil, &errwrap.WrappedError{
            Context: fmt.Sprintf("create #{opCtx.AppContext}->connect"),
            Cause:   err,
        }
    }
}
```

```go
// CreateMember Create a new member using the data supplied
func CreateMember(ctx context.Context, member masherytypes.Member, c *transport.V3Transport) (*masherytypes.Member, error) {
    rawResp, err := c.CreateObject(ctx, member, transport.FetchSpec{
        Resource:       "/members",
        AppContext:     "members",
        ResponseParser: masherytypes.ParseMasheryMember,
    })

    if err == nil {
        rv, _ := rawResp.(masherytypes.Member)
        return &rv, nil
    } else : nil, err ⤴
}
```

Polymorphism

- Method overloads achieve ad-hoc polymorphism
- Subtypes achieve subtype polymorphism
- Generics achieve parameter polymorphism



List<T>

No generics in code =
you work for the compiler

Generics in code =
compiler works for you

Generics: 3rd polymorphism type

So this is how these would creep into your code

```go
package genericDemo

import (
    "fmt"
    "math"
)

type Audi struct {  2 usages  new *
    fuel      int
    fuelPerKM float64
}
type Tesla struct {  2 usages  new *
    charge      int
    chargePerKm float64
}

func (a *Audi) ride(distance int) {  1 usage  new *
    a.fuel = a.fuel - int(math.Round(float64(distance)*a.fuelPerKM))
    fmt.Printf( format: "You rode %d km in Audi and have now %d fuel remaining\n", distance, a.fuel)
}

func (t *Tesla) ride(distance int) {  1 usage  new *
    t.charge = t.charge - int(math.Round(float64(distance)*t.chargePerKm))
    fmt.Printf( format: "You rode %d km in Audi and have now %d fuel remaining\n", distance, t.charge)
}
```

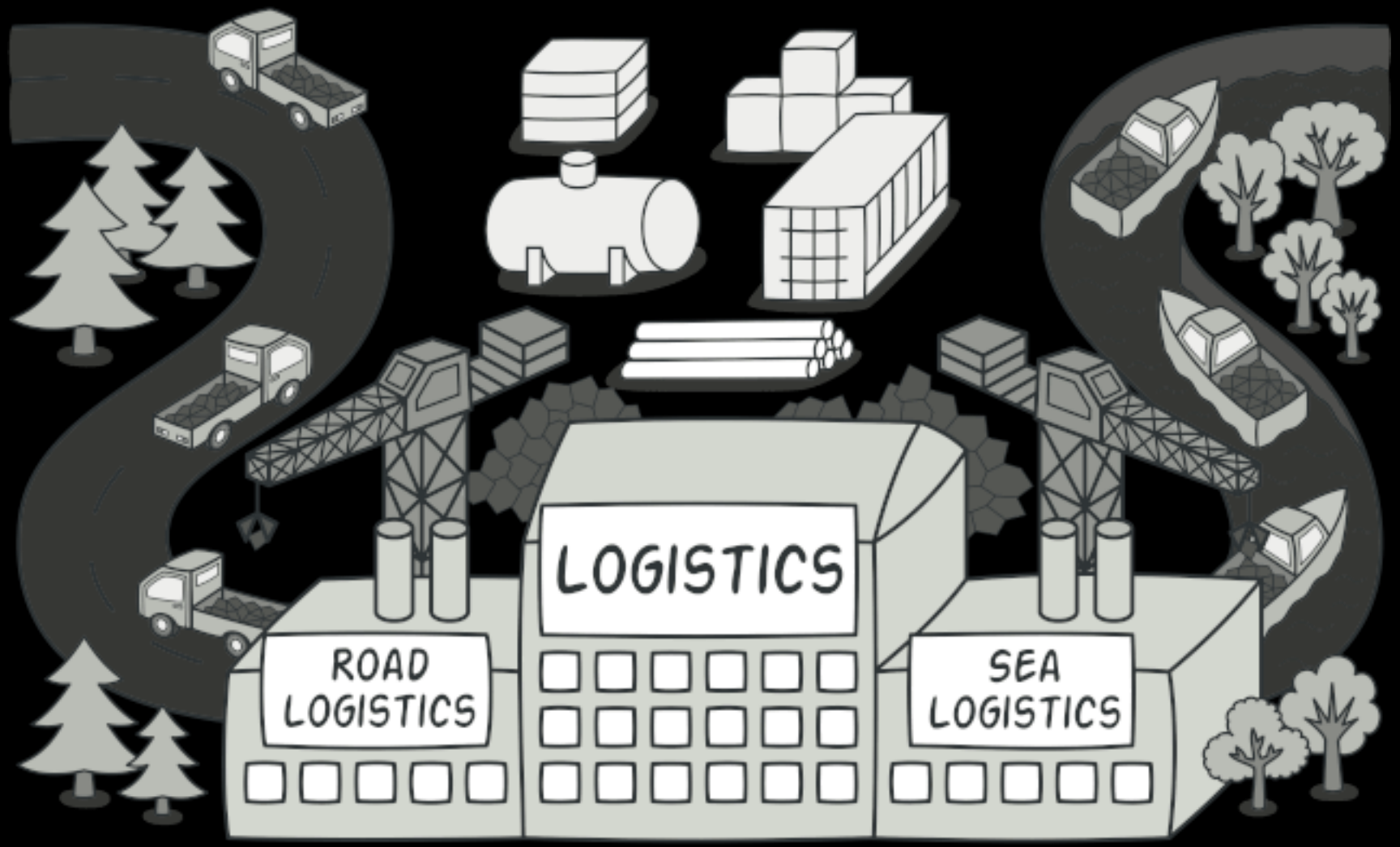
```go
package genericDemo

import "testing"

func fuelEconomyOf(unit, distance int) float64 {  2 usages  new *
    return float64(unit) / float64(distance)
}

func TestGenerics(t *testing.T) {  new *
    audi := Audi{ fuel: 50,  fuelPerKM: fuelEconomyOf( unit: 6,  distance: 100)}
    tesla := Tesla{ charge: 70,  chargePerKm: fuelEconomyOf( unit: 3,  distance: 100)}

    audi.ride( distance: 40)
    tesla.ride( distance: 40)
}
```

✓ **Tests passed: 1** of 1 test – 174 ms

```
<3 go setup calls>
/usr/local/go/bin/go tool test2json -t /Users/aliakseiyanchuk/Library/Caches/JetBrains/GoLand2
=== RUN   TestGenerics
You started off with 50 fueld, then rode 40 km in Audi and have now 47 fuel remaining
You started off with 70 charge, then rode 40 km in Tesla and have now 68 fuel remaining
--- PASS: TestGenerics (0.00s)
PASS

Process finished with the exit code 0
```

```go
package genericDemo

import "testing"

// As a user of the car dealership, I would like to get myself a vehicle that
// can ride me a certain amount of kilometers
func TestWheelerDealer(t *testing.T) {  new *
    wheelerDealer := WheelerDealer{}
    v := wheelerDealer.GetVehicle()
    v.ride(40)
}
```

```go
type Vehicle interface {  1 usage  2 implementations  new *
    ride(distance int)  2 implementations
}

func (wd WheelerDealer) GetVehicle() Vehicle {  1 usage  new *
    //return &Audi{fuel: 70, fuelPerKM: fuelEconomyOf(5, 100)}
    return &Tesla{charge: 70, chargePerKm: fuelEconomyOf( unit: 3,  distance: 100)}
}
```



Sourcel https://refactoring.guru/design-patterns/factory-method

```go
package genericDemo

import "testing"

// As a user of the car dealership, I would like to get myself a vehicle that
// can ride me a certain amount of kilometers
func TestWheelerDealerWithRefueling(t *testing.T) {  new *
    wheelerDealer := WheelerDealer{}
    v := wheelerDealer.GetVehicle()
    v.ride( distance: 40)

    v.refuel(fuel)
}
```

```go
type Vehicle interface {  1 usage  new *
    ride(distance int)
    refuel(qty int) // Is this a good design?
}
```

```go
func (a *Audi) refuel(qty int) {  no usages  new *
    a.fuel += qty
}

func (a *Tesla) refuel(qty int) {  no usages  new *
    a.charge += qty
}
```

```go
type DieselFuel struct {  no usages  new *
    qty int
}

type Benzine95Fuel struct {  no usages  new *
    qty int
}

type Benzine95E10Fuel struct {  no usages  new *
    qty int
}

type Electricity struct {  no usages  new *
    voltage  int
    amperes  int
    duration time.Duration
}
```

```go
func (a *Audi) refuel(fuel interface{}) {  no usages  new *
    if b, ok := fuel.(Benzine95Fuel); ok {
        a.fuel += b.qty
    } else if b, ok := fuel.(Benzine95E10Fuel); ok {
        a.fuel += b.qty
    } else {
        panic( v: "wrong fuel!")
    }
}

func (a *Tesla) refuel(charge interface{}) {  no usages  new *
    if b, ok := charge.(Electricity); ok {
        a.charge += trickyConversionFormula(b.voltage, b.amperes, b.duration)
    }
}
```

```go
type Vehicle interface {  1 usage  2 implementations  new *
    ride(distance int)  2 implementations
    refuel(fuel interface{}) // Is this a good design?  2 implementations
}
```

```go
func (wd *WheelerDealer) GetVehicle() Vehicle {  2 usages  new *
    //return &Audi{fuel: 70, fuelPerKM: fuelEconomyOf(5, 100)}
    return &Tesla{charge: 70, chargePerKm: fuelEconomyOf( unit: 3,  distance: 100)}
    //return nil
}
```

```go
// As a user of the car dealership, I would like to get myself a vehicle that
// can ride me a certain amount of kilometers
func TestWheelerDealerWithRefueling(t *testing.T) {  new *
    wheelerDealer := WheelerDealer{}
    v := wheelerDealer.GetVehicle()
    v.ride( distance: 40)

    b := Benzine95Fuel{qty: 5}
    v.refuel(b)
}
```

Generics!!!!
Or No Go!

```
Proces                                  de 1 ⚠

                                  on -t /Users/aliakseiyanchuk/Library/Caches/JetBrains/GoLand2
                                  efueling
                                  then rode 40 km in Tesla and have now 68 charge remaining
---                               ithRefueling (0.00s)
panic: trying putting fo          fuel into an electric car is terminally intelligent [recovered]
    panic: trying putting fossil fuel into an electric car is terminally intelligent
```

```
type Vehicle[TFuel Benzine95 | Benzine95E10 | Diesel | Electricity] interface {  1 usage
    ride(distance int)  2 implementations
    refuel(fuel TFuel) // That's better!  2 implementations
}
```

```
type AllFuels interface {  1 usage  new *
    Benzine95 | Benzine95E10 | Diesel | Electricity
}

type Vehicle[TFuel AllFuels] interface {  1 usage  2 implementations  new *
    ride(distance int)  2 implementations
    refuel(fuel TFuel) // That's better!  2 implementations
}
```

```go
// As a user of the car dealership, I would like to get myself a vehicle that
// can ride me a certain amount of kilometers
func TestWheelerDealerWithRefueling(t *testing.T) {  new *
    wheelerDealer := WheelerDealer{}
    v := wheelerDealer.GetVehicle()
    v.ride( distance: 40)

    b := Benzine95{qty: 5}
    v.refuel(b)
}
```

Writing crashing code has come (a bit) harder

```go
// As a user of the car dealership, I would like to get myself a vehicle that
// can ride me a certain amount of kilometers
func TestWheelerDealerWithRefueling(t *testing.T) {  new *
    wheelerDealer := WheelerDealer{}
    v := wheelerDealer.GetVehicle()
    v.ride( distance: 40)

    b := Electricity{
        voltage:  350,
        amperes:  500,
        duration: time.Minute * time.Duration(30),
    }
    v.refuel(b)
}
```

```go
    return &Tesla{charge: 70, chargePerKm: fuelEconomyOf( unit: 3,  distance: 100)}
    //return nil
}

func (wd *WheelerDealer) GetAnyVehicle() Vehicle[Diesel] {  no usages  new *
    //return &Audi{fuel: 70, fuelPerKM: fuelEconomyOf(5, 100)}
    return &Tesla{charge: 70, chargePerKm: fuelEconomyOf( unit: 3,  distance: 100)}
    //return nil
}
```

Cannot use '&Tesla{charge: 70, chargePerKm: fuelEconomyOf(3, 100)}' (type *Tesla) as the type Vehicle[Diesel]
Type does not implement 'Vehicle[Diesel]'
need the method: refuel(fuel Diesel)
have the method: refuel(charge Electricity)

field chargePerKm float64 **of** Tesla type

This composite literal allocates a new struct instance with the given values.

You may either specify the field values in order or use their names.

```go
func TestRideAnything(t *testing.T) {  new *
    wheelerDealer := WheelerDealer{}

    RideAnything( distance: 30, &wheelerDealer, func(dealer *WheelerDealer) func() Vehicle[Benzine95E10] {
        return dealer.GetSportsVehicle
    })
    RideAnything( distance: 30, &wheelerDealer, func(dealer *WheelerDealer) func() Vehicle[Electricity] {
        return dealer.GetVehicle
    })
}
```

```go
func RideAnything[T AllFuels](distance int, wd *WheelerDealer,   2 usages  new *
    locator func(dealer *WheelerDealer) func() Vehicle[T]) {
    factoryFunc := locator(wd)
    v := factoryFunc()
    v.ride(distance)
}
```

```java
package nl.yanchuk.golanggenerics.wheelers;

import org.junit.jupiter.api.Test;

class WheelerDemoTest {
    @Test
    public void testShowWildcardGenerics() {
        WheelerDealer wd = new WheelerDealer();
        Vehicle<?> v = wd.getVehicle();
        v.ride( distance: 49);
    }
}
```

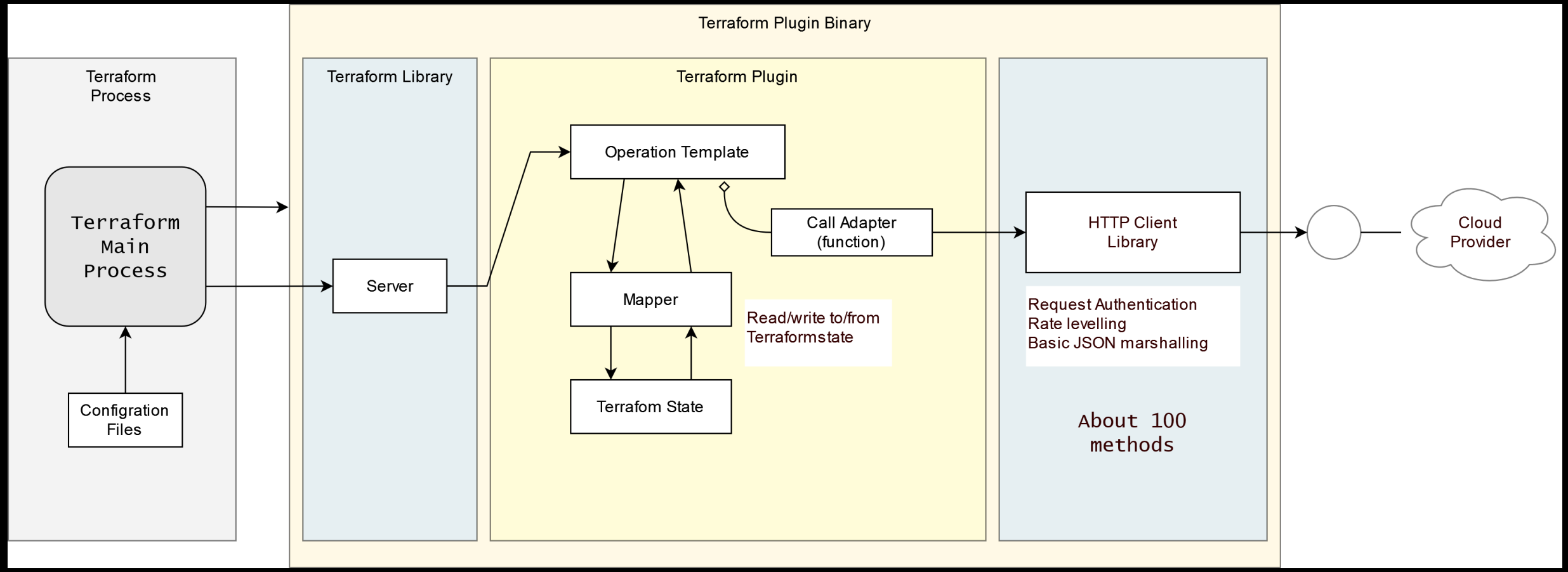
This design cannot, probably, be generified

```go
type Propelled interface {  1 usage  3 implementations  new *
    ride(distance int)  2 implementations
}

type Vehicle[TFuel AllFuels] interface {  5 usages  2 implementations  new *
    Propelled
    refuel(fuel TFuel) // That's better!  2 implementations
}
```

Associative generic types

Introducing Generics into existing codebase

Go 1.15 architecture



```go
type ResourceTemplate[ParentIdent any, Ident any, MType any] struct {
    Schema map[string]*schema.Schema
    Mapper *tfmapper.Mapper[ParentIdent, Ident, MType]

    UpsertableFunc func() MType

    // Validate function: if defined, checks the consistency of the parameters with each other.
    ValidateFunc func(parent ParentIdent, upsertable MType) string

    DoRead   ReaderFunc[Ident, MType]
    DoCreate CreatorFunc[ParentIdent, Ident, MType]
    DoUpdate UpdaterFunc[Ident, MType]
    DoDelete DeleterFunc[Ident]

    DoCountOffending OffendersCounterFunc[Ident]
}
```

Aliaksei Yanchuk

```go
func (rt *ResourceTemplate[ParentIdent, Ident, MType]) Read(ctx context.Context,  Aliaksei Yanchuk *
    state *schema.ResourceData,
    m interface{}) diag.Diagnostics {

    ident, err := rt.Mapper.Identity(state)
    if err != nil {
        return diag.Diagnostics{diag.Diagnostic{
            Severity: diag.Error,
            Summary:  fmt.Sprintf( format: "invalid resource identifier"),
            Detail:   fmt.Sprintf("attempt to parse identified returned this error: #{err.Error()}"),
        }}
    }

    v3Client := m.(v3client.Client)
    curState, err := rt.DoRead(ctx, v3Client, ident)
```

```go
    if err != nil {
        return diag.Diagnostics{diag.Diagnostic{
            Severity: diag.Error,
            Summary:  fmt.Sprintf(format: "unexpected error returned from Mashery V3 api"),
            Detail:   fmt.Sprintf("read operation on Mashery V3 api has returned the following error: #{err.Error()}")
        }}
    }

    rv := diag.Diagnostics{}

    if curState != nil {
        if setErrors := rt.Mapper.RemoteToSchema(curState, state); len(setErrors) > 0 {
            rv = append(rv, setErrors...)
        }
    } else {
        rt.Mapper.ResetIdentity(state)
    }

    return rv
}
```

It almost didn’t change
It’s simple
It’s just a bit verbose
Vvvvvverbose

```
var PackagePlanServiceResource *ResourceTemplate[masherytypes.PackagePlanIdentifier,    9 usages
    masherytypes.PackagePlanServiceIdentifier,
    mashschemag.PackagePlanServiceParam]
```

Bloat example

```go
func performGenericObjectCRUDWithResponse[T any](ctx context.Context,  5 usages   Aliaksei Yanchuk *
    opCtx ObjectFetchSpec[T],
    verb string,
    f CallFunc) (T, *WrappedResponse, error) {
    for i := 0; i < 10; i++ {
        if resp, err := f(ctx); err == nil {
            if dat, err := resp.Body(); err != nil {
                return opCtx.ValueFactory(), resp, &errwrap.WrappedError{
                    Context: fmt.Sprintf("#{verb} #{opCtx.AppContext}->read server response"),
                    Cause:   err,
                }
            } else {
                if resp.StatusCode == 200 {
                    // In specific situations (e.g. with the DELETE operation) the body of the response
                    // will be ignored, as the server is not supplying any useful information in the body.
                    // The calling code would specify a primitive (e.g. int) value that will be ignored by
                    // the calling code.
                    rv := opCtx.ValueFactory()
```

1

Bloat example

```
        if !opCtx.IgnoreResponse {
            if jsonErr := json.Unmarshal(dat, &rv); jsonErr != nil {
                return rv, resp, &errwrap.WrappedError{
                    Context: fmt.Sprintf("#{verb} #{opCtx.AppContext}->unmarshal response"),
                    Cause:   jsonErr,
                }
            }
        }

        return rv, resp, nil
    } else if resp.StatusCode == 403 {
        if str := resp.Header.Get(key: "X-Mashery-Error-Code"); str == "ERR_403_DEVELOPER_OVER_QPS" {
            d := time.Duration(1+i) * time.Second
            time.Sleep(d)
            continue
        } else {
            return opCtx.ValueFactory(), resp, &errwrap.WrappedError{
                Context: fmt.Sprintf("#{verb} #{opCtx.AppContext}->not authorized (http code 403)"),
                Cause:   errors.New(text: "call denied by server"),
            }
        }
```

2

Bloat example

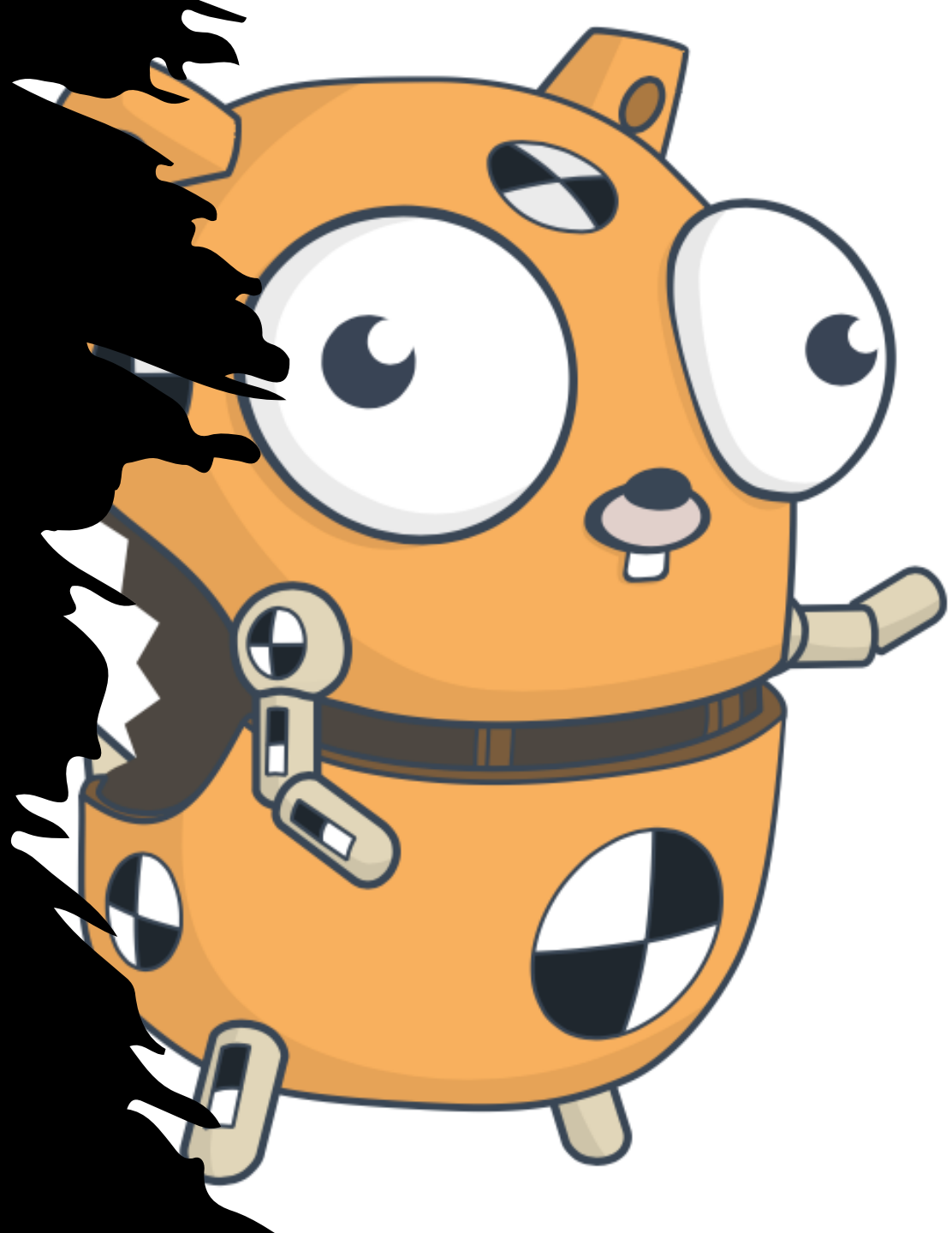
3

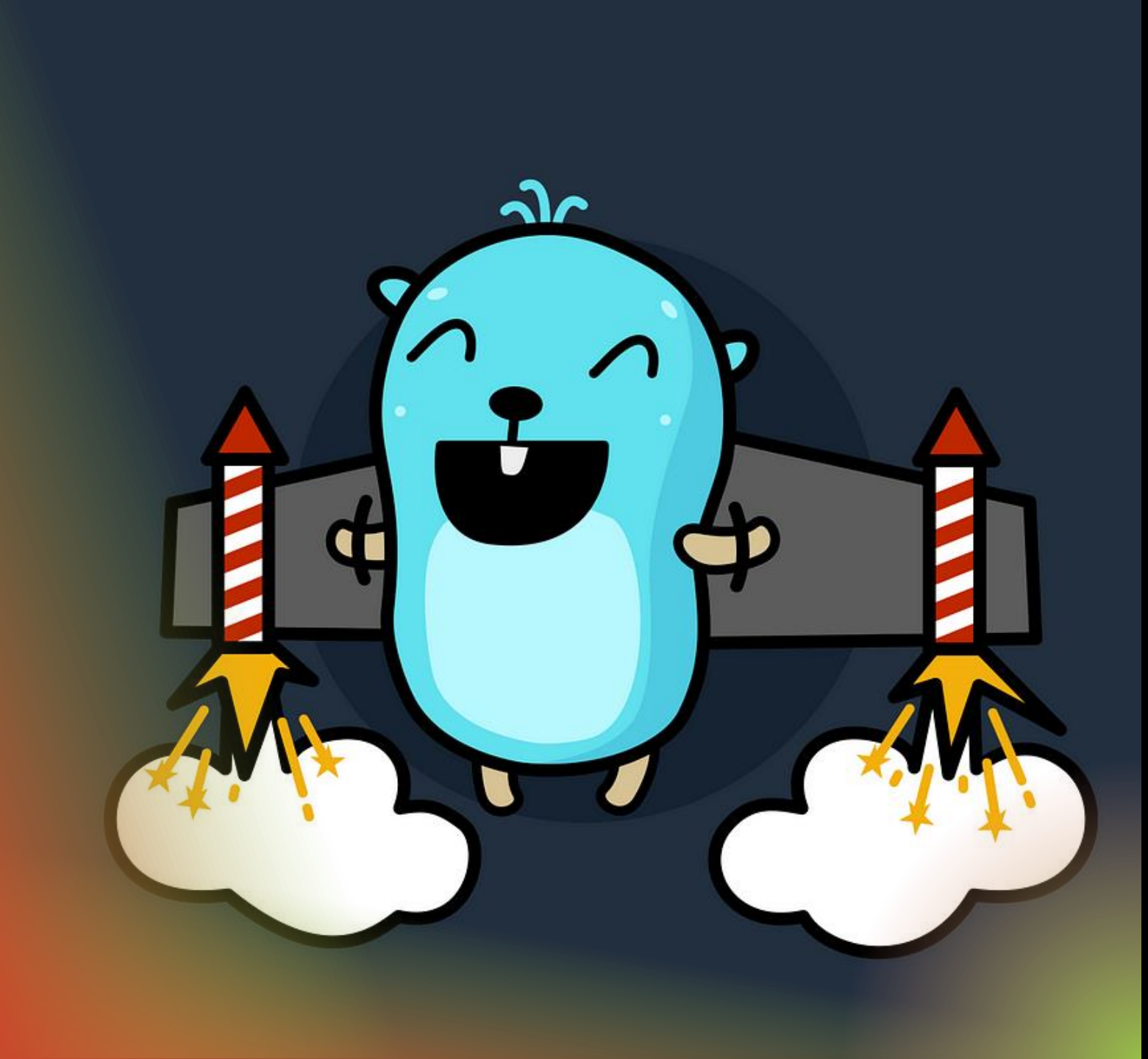
```
                } else if resp.StatusCode == 404 {
                    if opCtx.Return404AsNil : opCtx.ValueFactory(), resp, nil ↱ else {
                        return opCtx.ValueFactory(), resp, &errwrap.WrappedError{
                            Context: fmt.Sprintf("#{verb} #{opCtx.AppContext}->no such resource"),
                            Cause:   errors.New( text: "error code 404 is not an expected response to this request"),
                        }
                    }
                } else {
                    return opCtx.ValueFactory(), resp, v3ErrorFromResponse(fmt.Sprintf("#{verb} #{opCtx.AppContext}"),
                        resp.StatusCode, resp.Header, dat)
                }
            }
        } else : opCtx.ValueFactory(), nil, err ↱
    }

    return opCtx.ValueFactory(), nil, &errwrap.WrappedError{
        Context: fmt.Sprintf("#{verb} #{opCtx.AppContext}->unsatisfiable operation"),
        Cause:   errors.New( text: "operation unsuccessful after all available retries"),
    }
}
```

Take-out

It takes much, much deeper refactoring than I thought!...

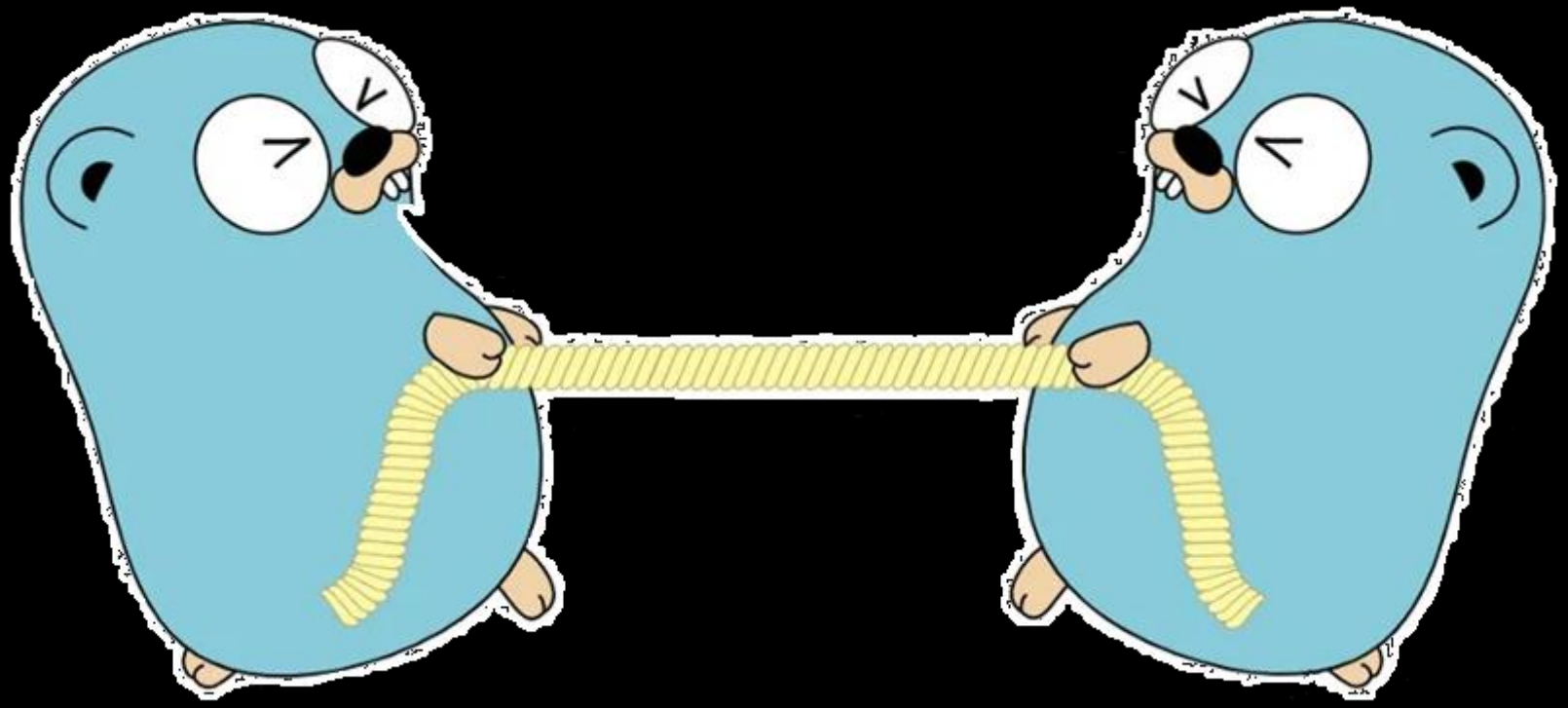
7 Reasons to Let go Generics





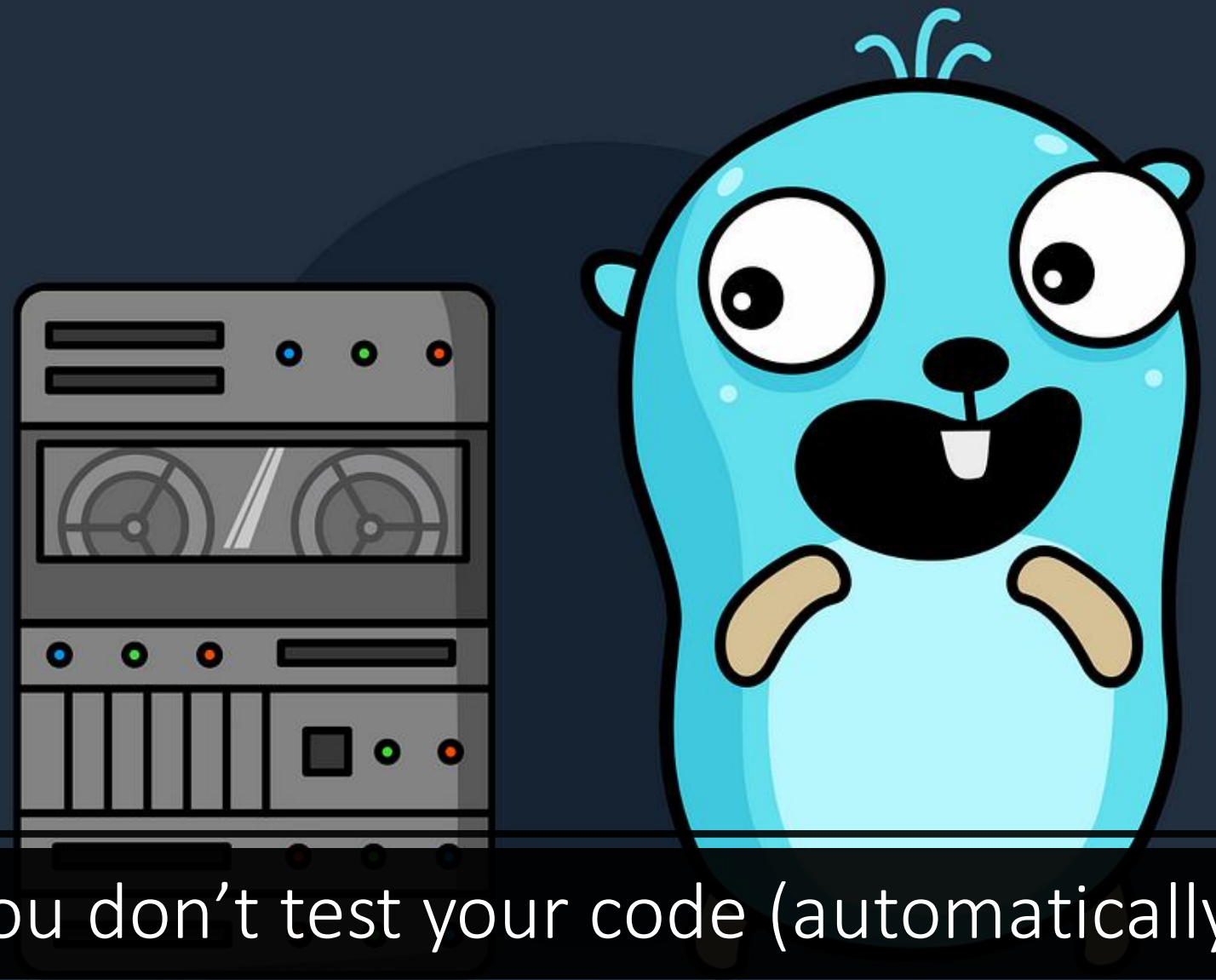
You like type-unsafe code

Writing is rewriting. Coding is re-coding



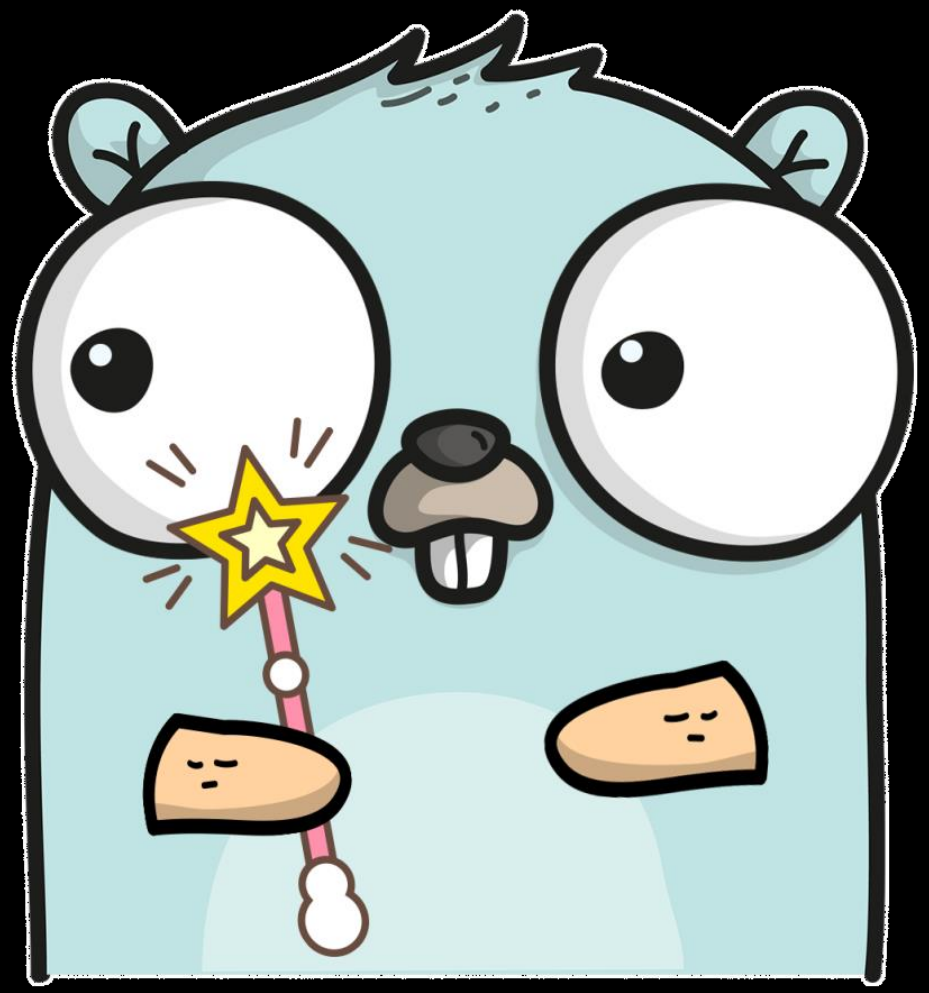
Your tech lead has told you to clean your code

You are convinced that functional programming is a fad

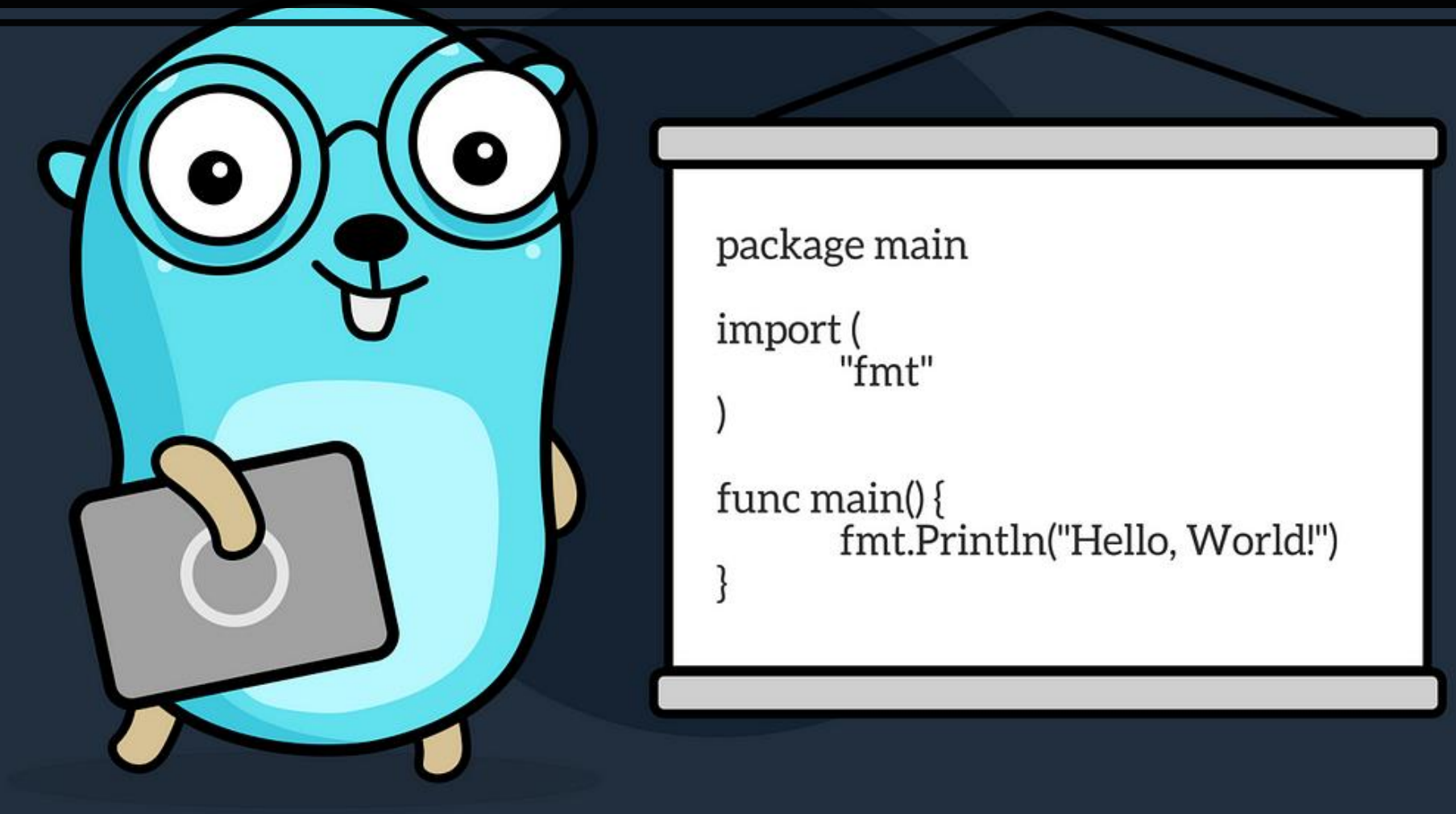


You don’t test your code (automatically)

The worse the code, the higher the job security



There’s always something generics can’t do

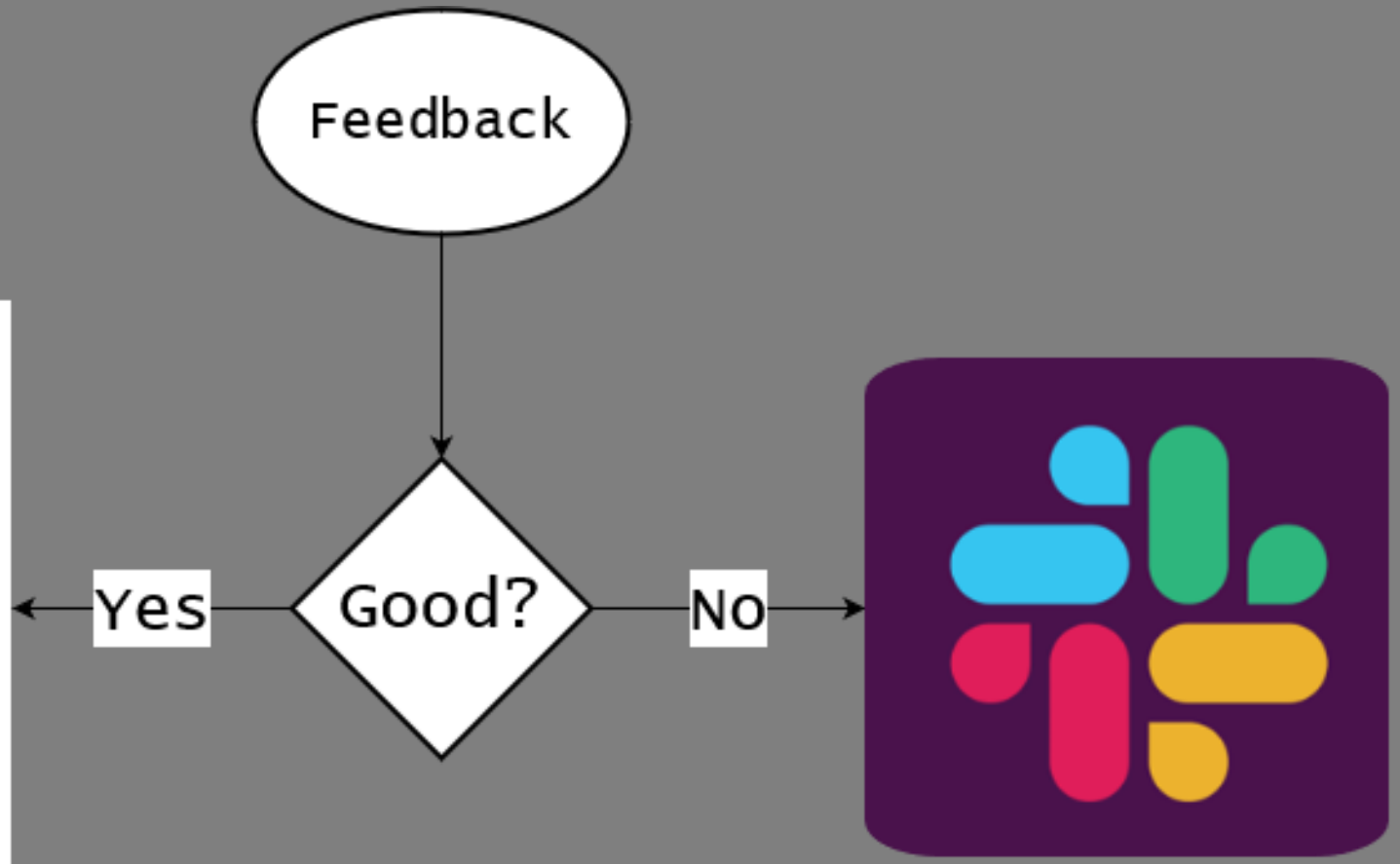


Nr. 8

We don’t do Java here!

List<T>
Brains!..

Happy Generifying!



Aleks Yanchuk