สยามชำนาญกิจ
SIAM CHAMNAN KIT

Go workshop

Somkiat Puisungnoen

Somkiat | Home

Somkiat Puisungnoen

Update Info 1 | View Activity Log 10+ | ...

Timeline | About | Friends 3,138 | Photos | More

When did you work at Opendream?

22 Pending Items

Post | Photo/Video | Live Video | Life Event

What's on your mind?

Public | Post

Intro

Software Craftsmanship

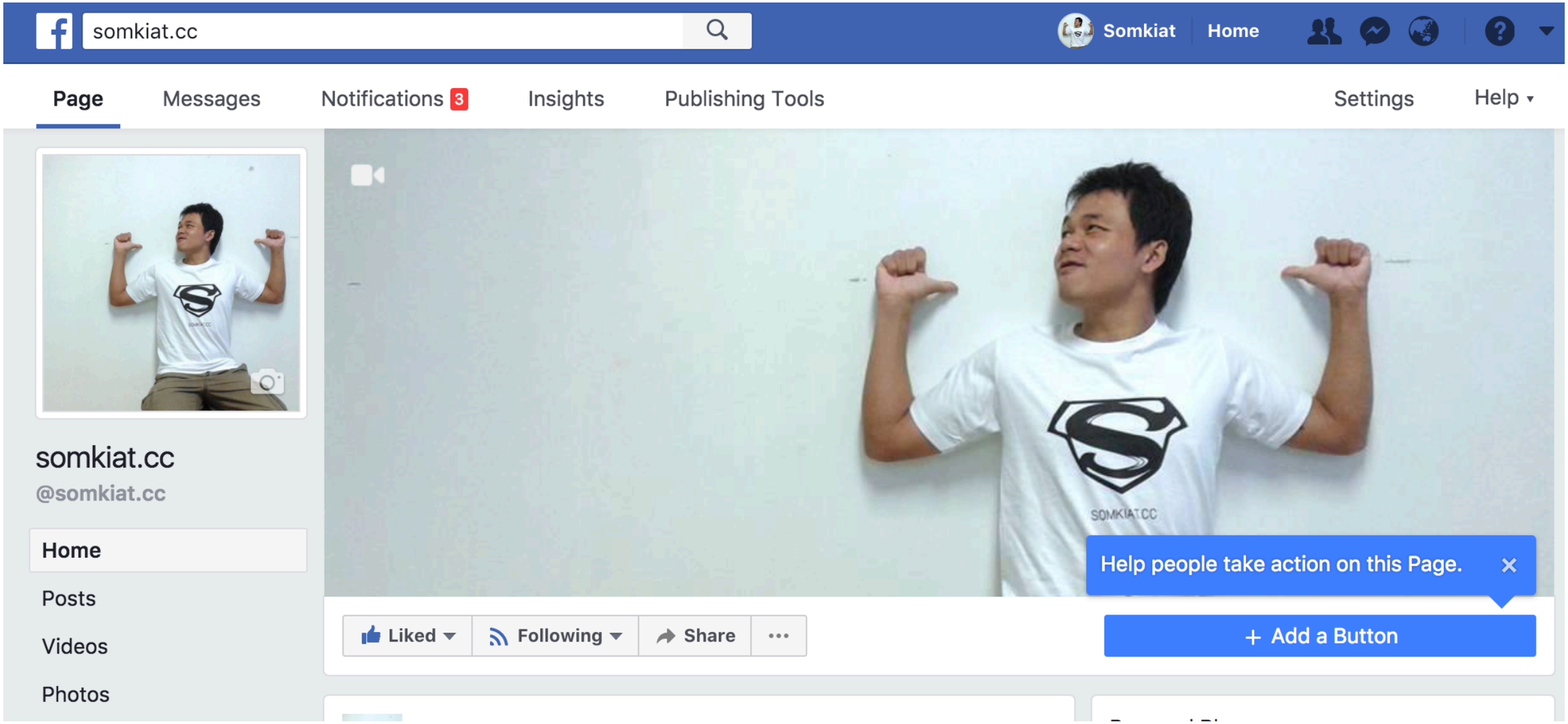
Software Practitioner at สยามชำนาญกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

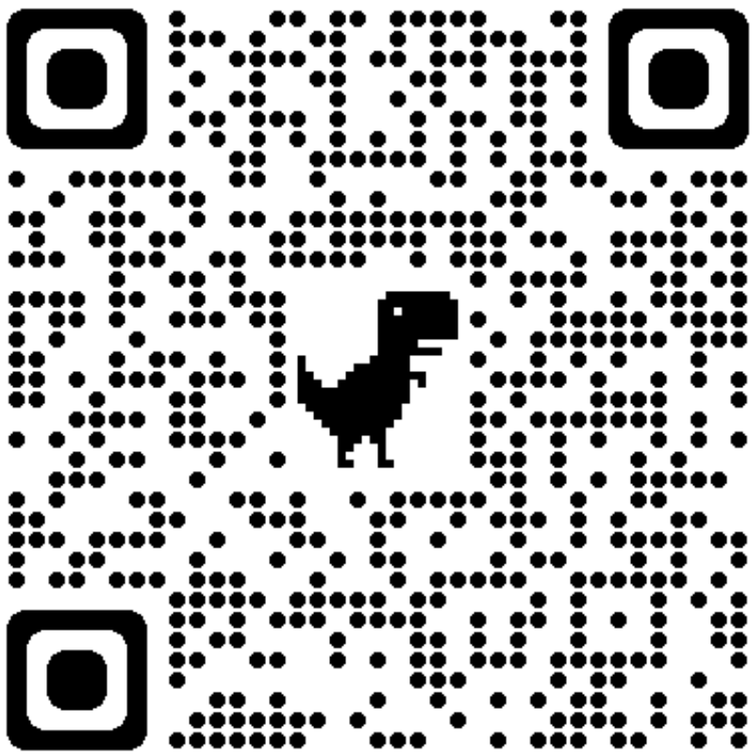
Somkiat Puisungnoen

15 mins · Bangkok ·

Java and Bigdata



https://github.com/up1/course-go-2023



Agenda

Topics (1)

Project with Go
Essential features
Testing with Go
Testable project with Go
Develop REST API with gin
Testable project with Go

Topics (2)

Benchmark and Profiling
Working Docker
Performance Testing
CI/CD for Go project

Let’s

Why Go ▾ Learn Docs ▾ Packages Community ▾

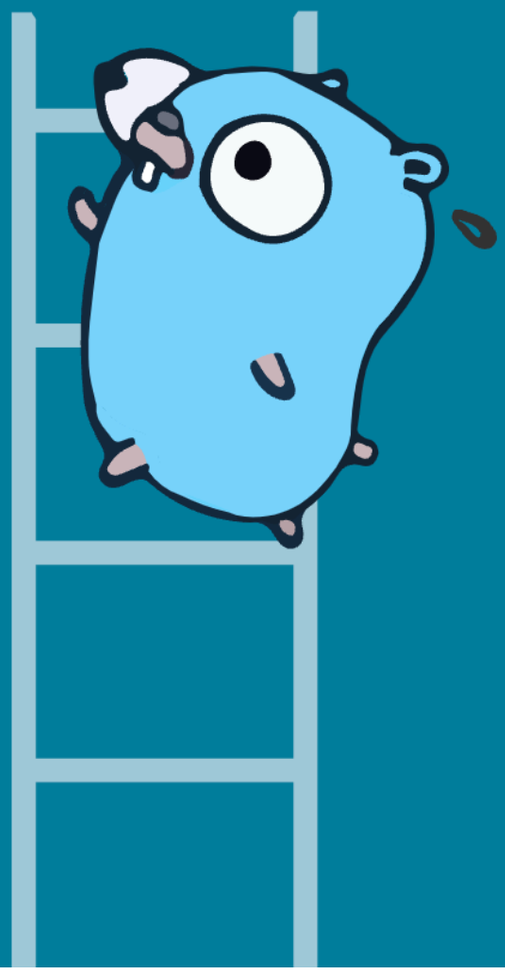
Build simple, secure, scalable systems with Go

- ✓ An open-source programming language supported by Google
- ✓ Easy to learn and great for teams
- ✓ Built-in concurrency and a robust standard library
- ✓ Large ecosystem of partners, communities, and tools

Get Started | Download

Download packages for Windows 64-bit, macOS, Linux, and more

The go command by default downloads and authenticates modules using the Go module mirror and Go checksum database run by Google. Learn more.



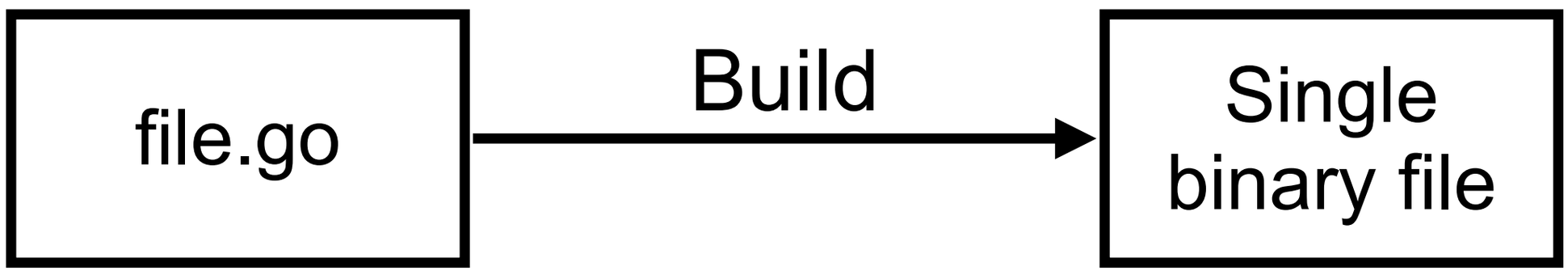
https://go.dev/

$go version

Technical Excellence

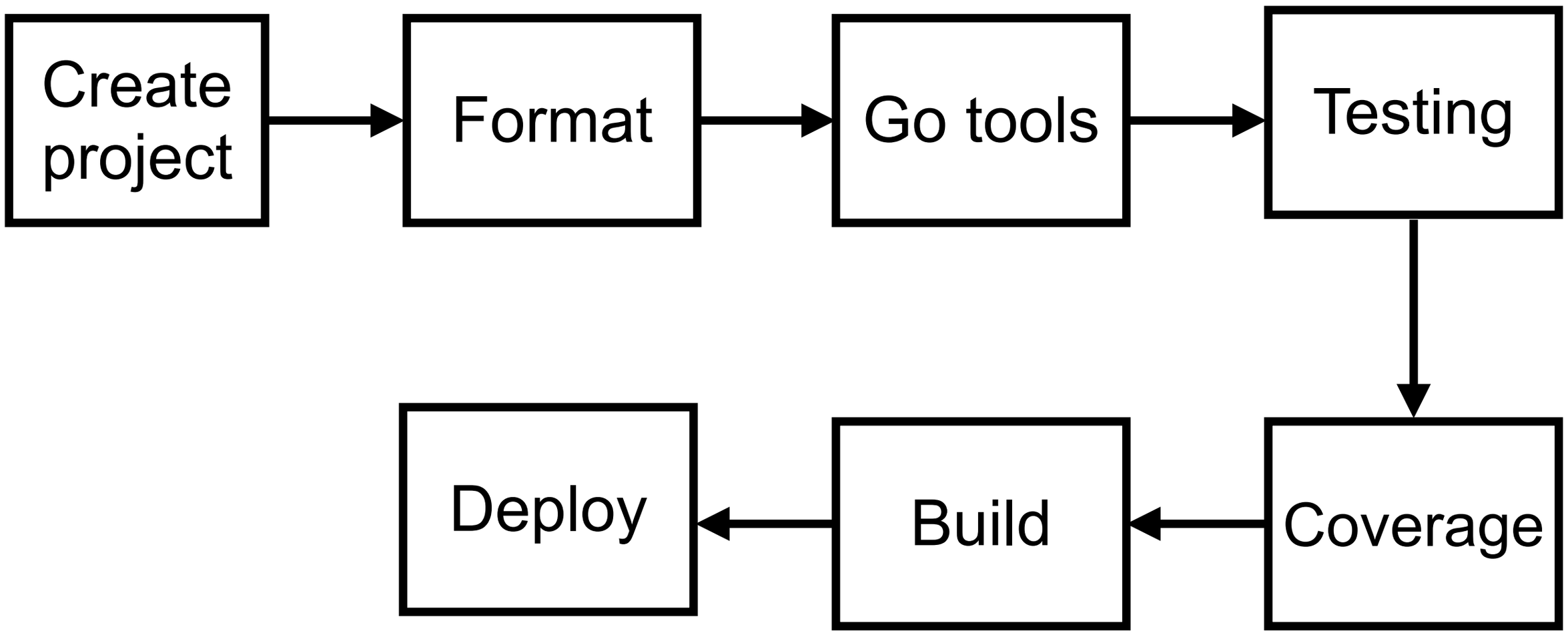
https://less.works/less/technical-excellence

Basic Go




```
$go run file.go
$go build -o <output>
```

Development workflow



Create go project

$go mod init **demo**
$go mod tidy

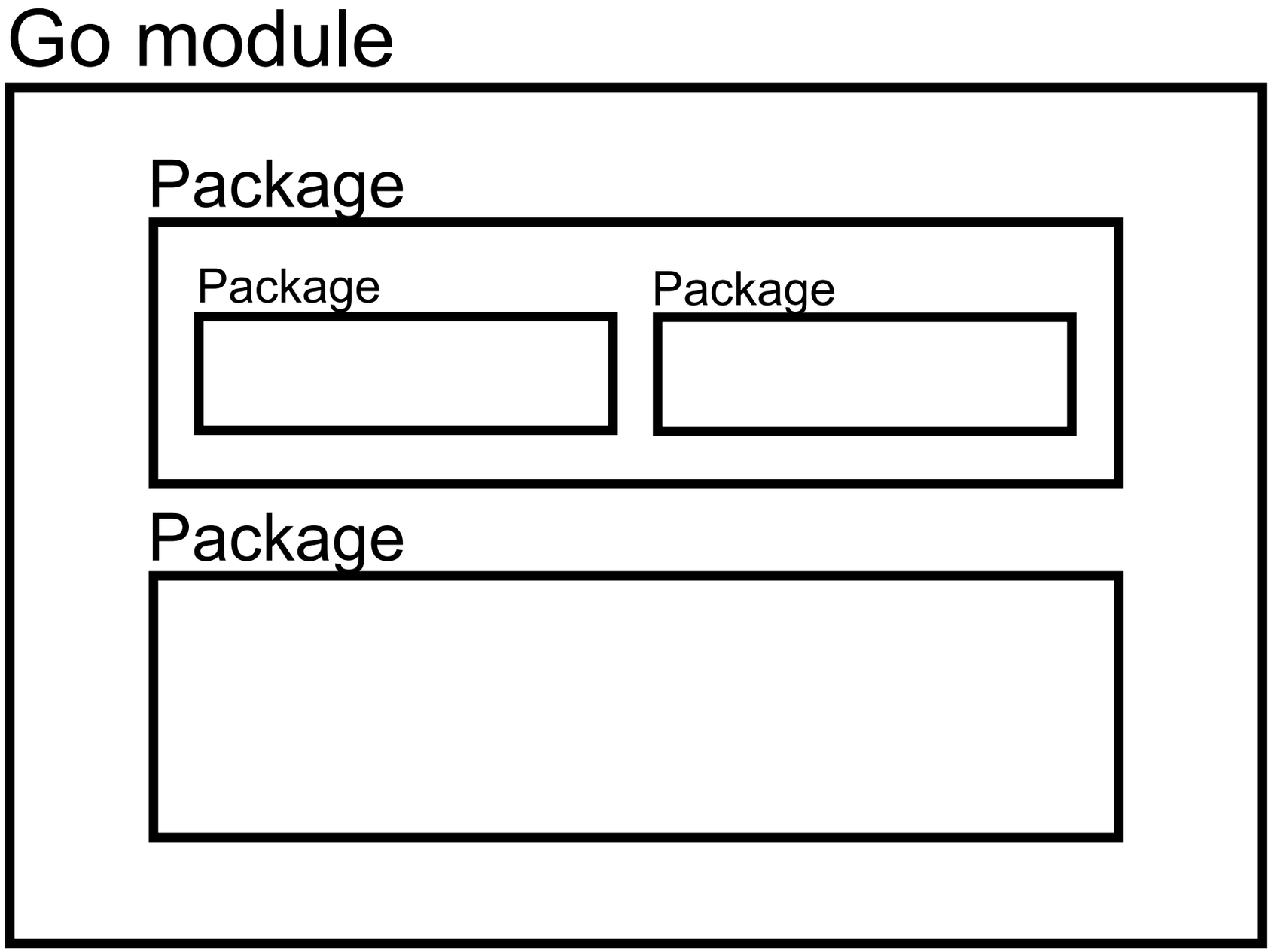
https://go.dev/blog/using-go-modules

Hello Go !!

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, world 2023")
    print("Hello, world 2023")
}
```

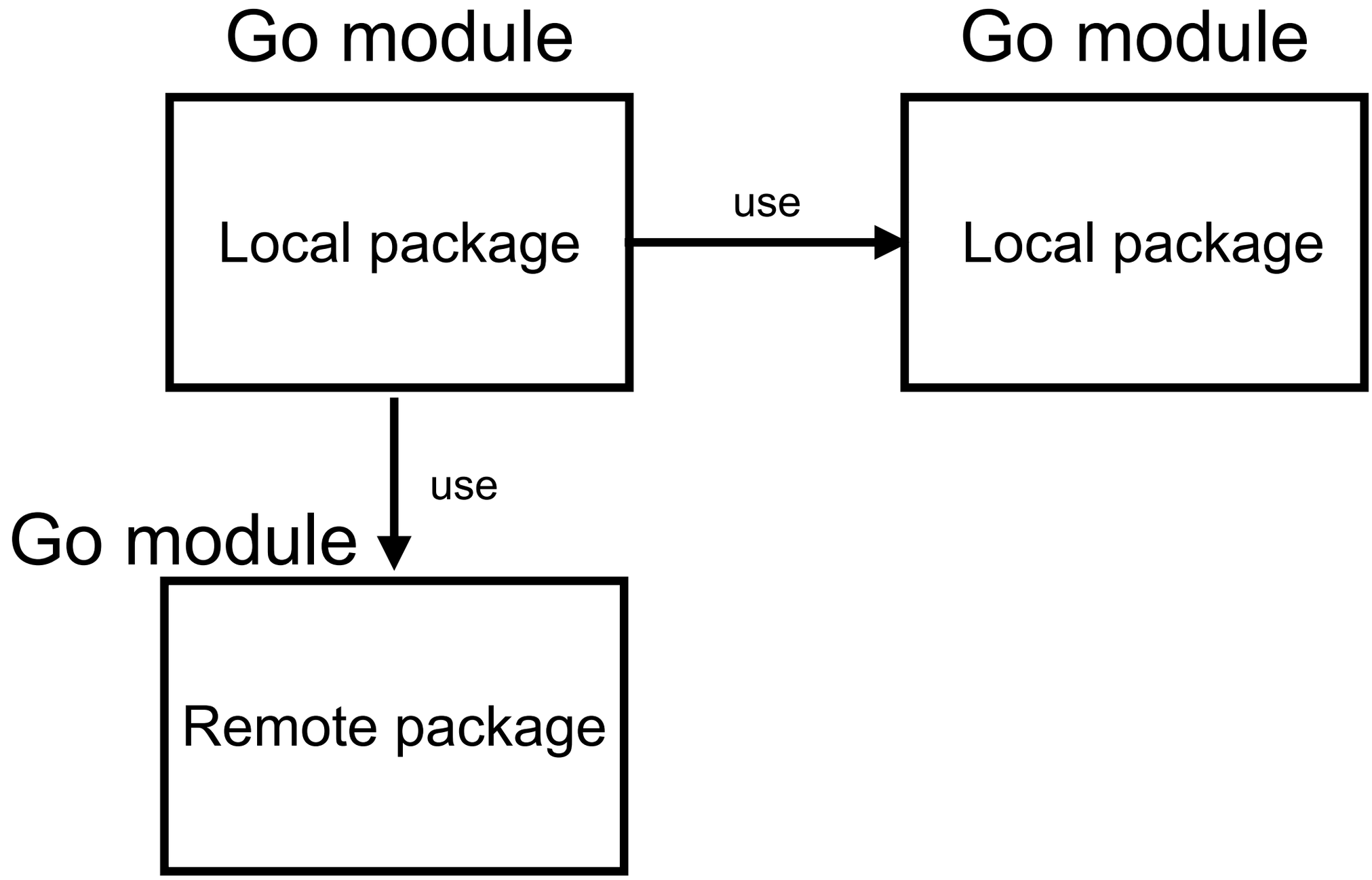
```
$go run hello.go
```

Create go project



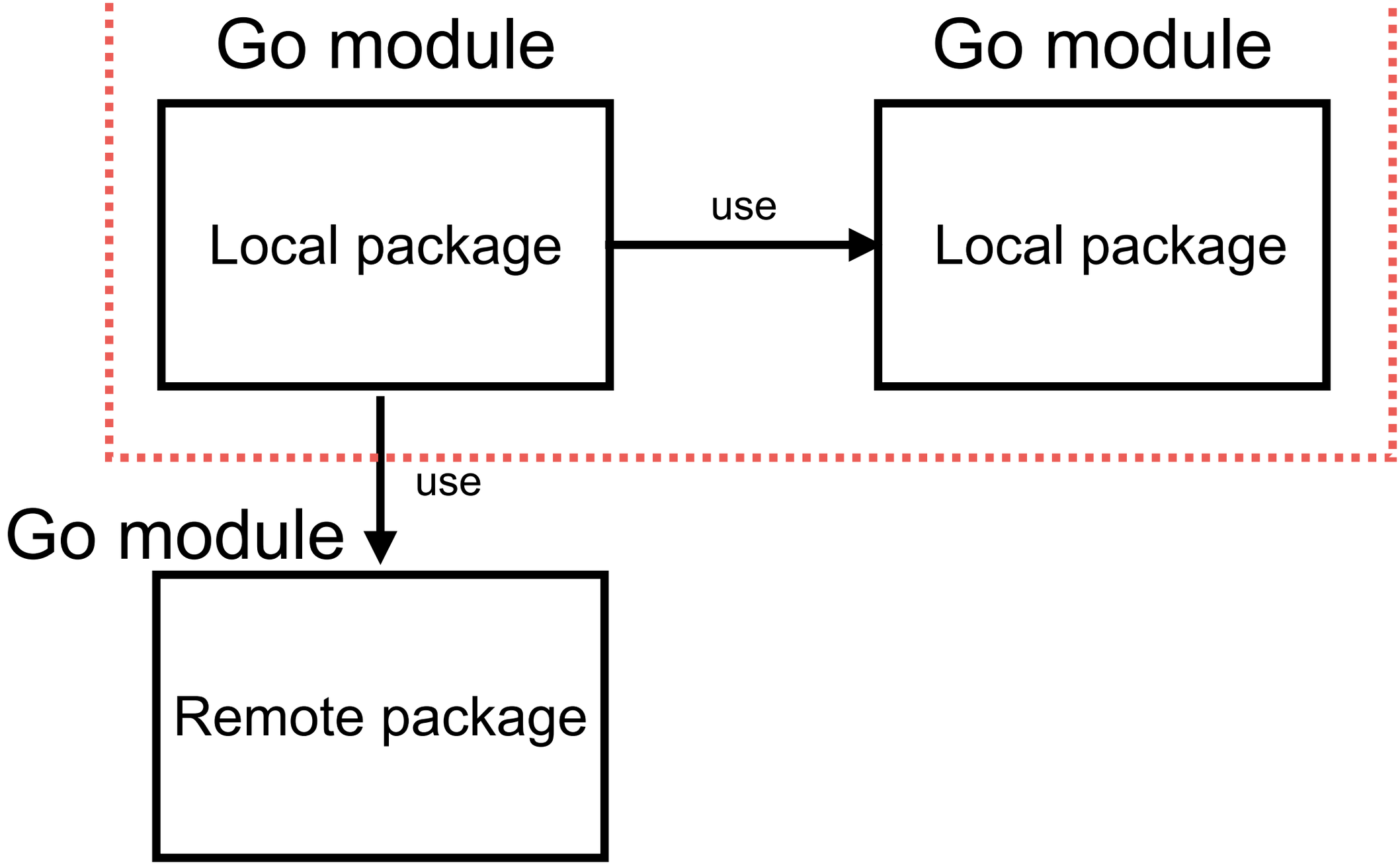
https://go.dev/blog/using-go-modules

Go Workspace



https://go.dev/doc/tutorial/workspaces

Go Workspace



https://go.dev/doc/tutorial/workspaces

Go workspace

$go work init
$go work use ./m1
$go work use ./m2

https://go.dev/blog/using-go-modules

Formating

$go fmt

Go tools

$go tool

$go tool cover
$go tool doc
$go tool vet

$go tool pprof

https://pkg.go.dev/golang.org/x/tools

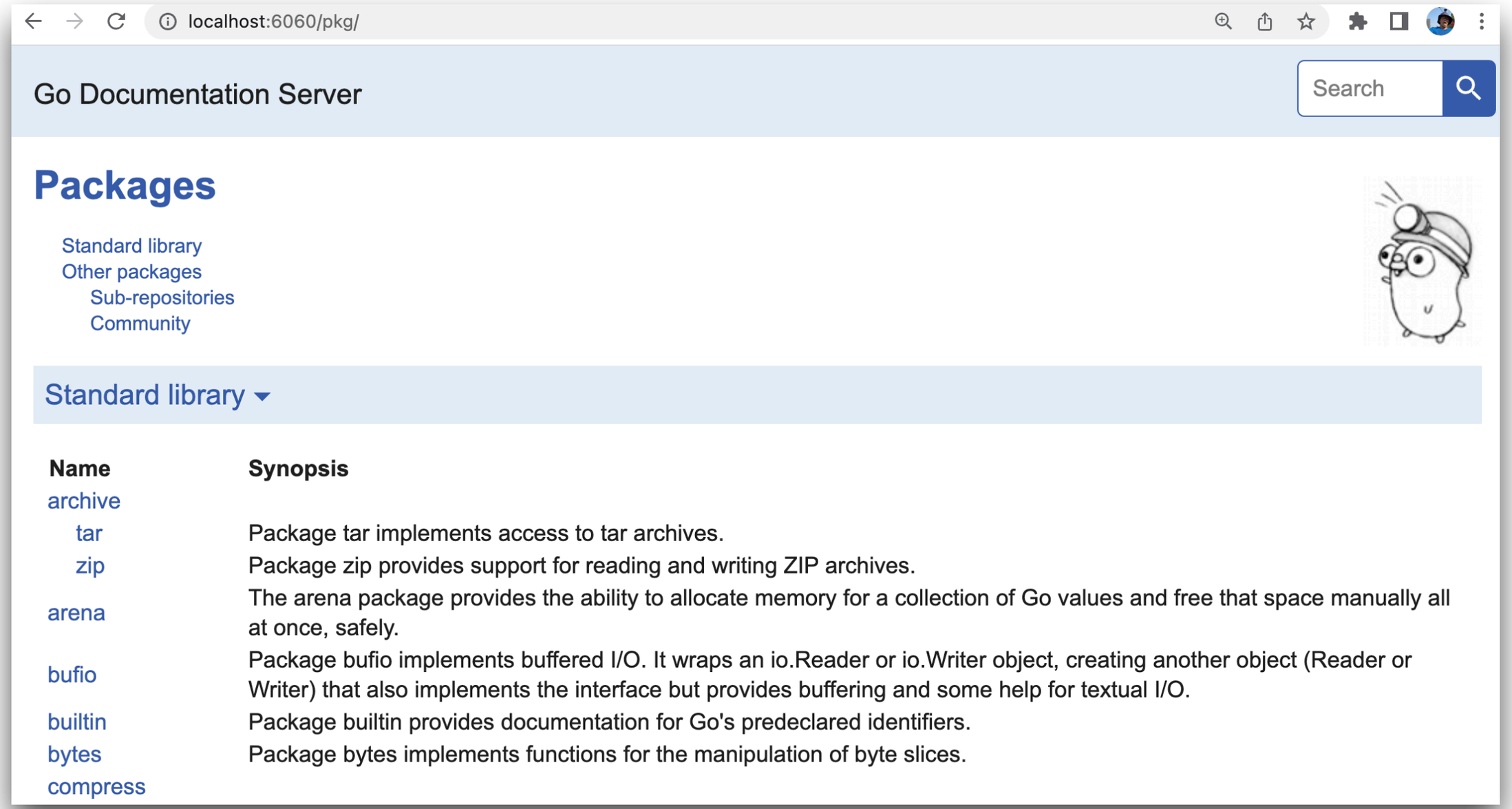
Go Doc

$go install -v golang.org/x/tools/cmd/godoc@latest

$godoc -http=:6060

https://pkg.go.dev/golang.org/x/tools

Go Doc



https://pkg.go.dev/golang.org/x/tools

Testing with Go

Learning by test !!

Testing with Go

testing package

Test	Benchmark	Example	Fuzzing

https://pkg.go.dev/testing

Testing with Go

hello.go

```go
package main

import "fmt"

func Hello() string {
    return "Hello, world 2023"
}

func main() {
    fmt.Println(Hello())
}
```

hello_test.go

```go
package main

import "testing"

func TestHello(t *testing.T) {
    got := Hello()
    want := "Hello, world 2023"

    if got != want {
        t.Errorf("got %q want %q", got, want)
    }
}
```

```
$go test -v
```

No private/public in Go !!

Data = public or exported

data = private or unexported

Essential features

Essential Features

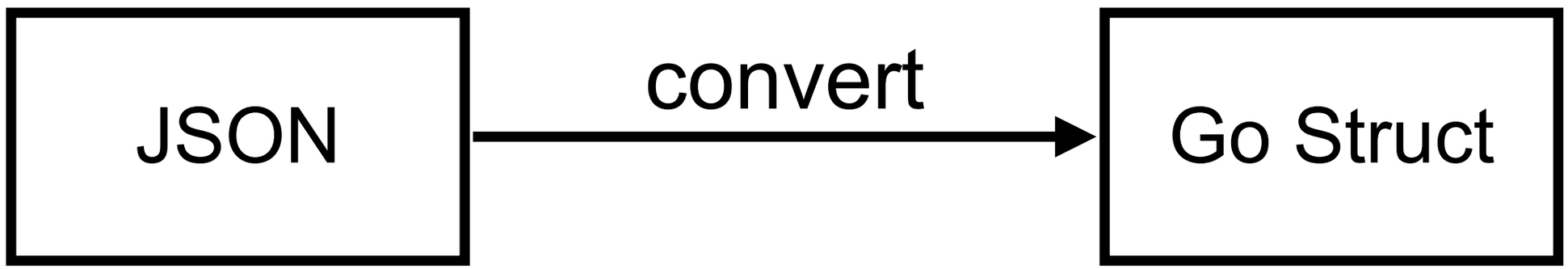
Struct and Pointer
Interface
Error
Defer
Function as a parameter
Go routine and channel

Struct and Pointer

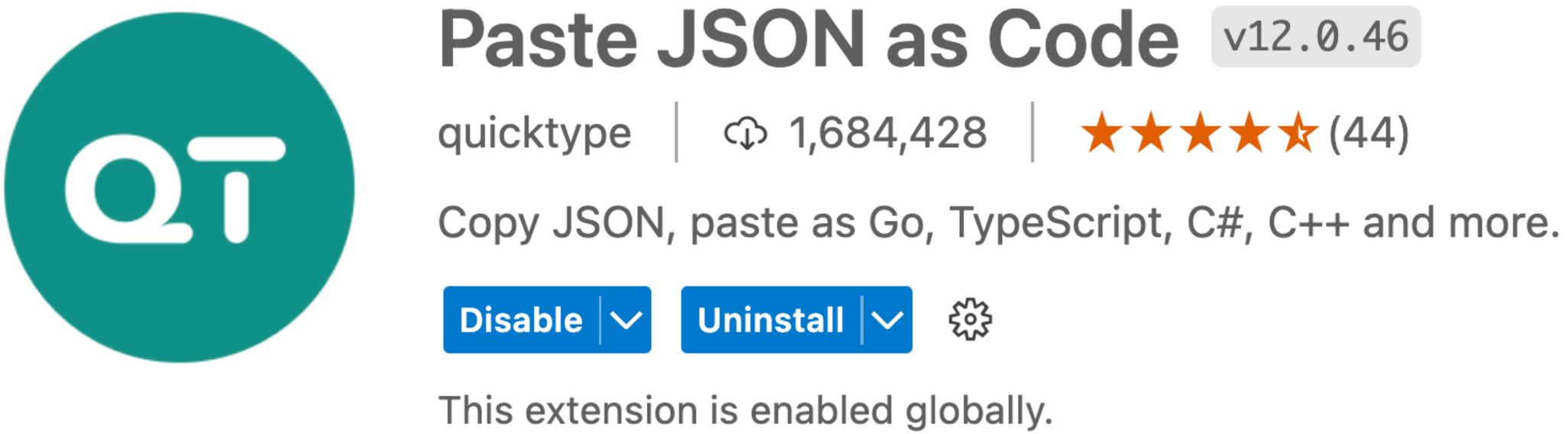
Named collection of fields
Use to store data

```
type User struct {
    ID       int    `json:"id"`
    Name     string `json:"name"`
    Username string `json:"username"`
    Email    string `json:"email"`
    Address  struct {
        Street  string `json:"street"`
        Suite   string `json:"suite"`
        City    string `json:"city"`
        Zipcode string `json:"zipcode"`
        Geo     struct {
            Lat string `json:"lat"`
            Lng string `json:"lng"`
        } `json:"geo"`
    } `json:"address"`
}
```

Convert JSON to Go Struct

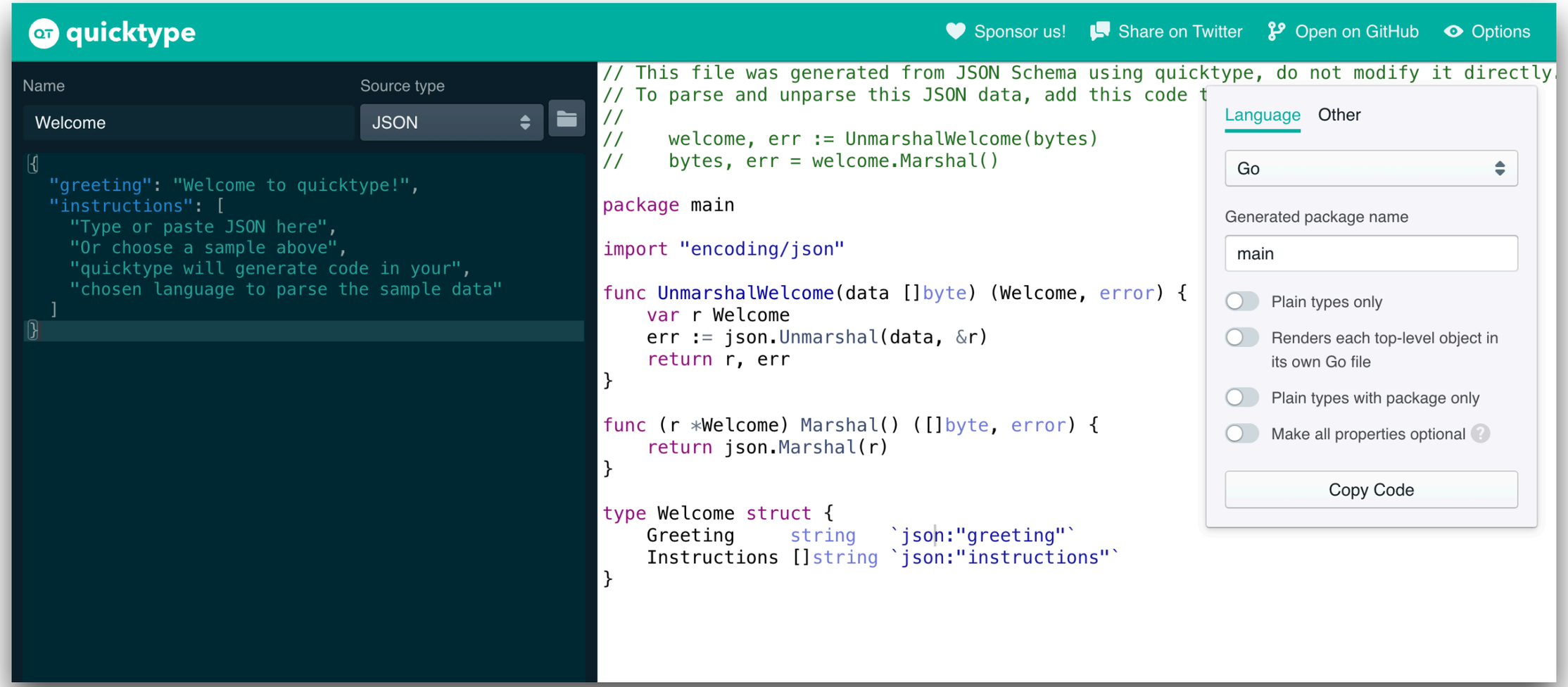


Convert JSON to Go Struct



https://marketplace.visualstudio.com/items?itemName=quicktype.quicktype

Convert JSON to Go Struct

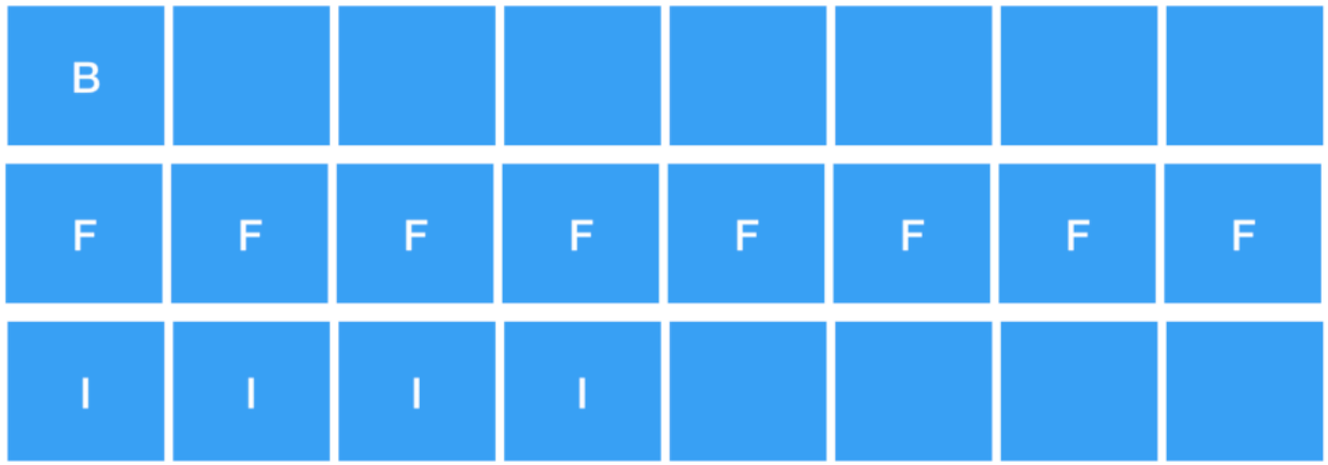


https://app.quicktype.io/

Sequence of fields in struct

```go
func main() {
    type first struct {
        b bool    // 1 byte
        f float64 // 8 bytes
        i int32   // 4 bytes
    }
    a := first{}
    fmt.Println(unsafe.Sizeof(a)) // 24 bytes
}
```

Memory padding



https://www.somkiat.cc/golang-memory-of-struct/

Sequence of fields in struct

```go
func main() {
    type first struct {
        f float64 // 8 bytes
        b bool    // 1 byte
        i int32   // 4 bytes
    }
    a := first{}
    fmt.Println(unsafe.Sizeof(a)) // 16 bytes
}
```

Memory padding

https://www.somkiat.cc/golang-memory-of-struct/

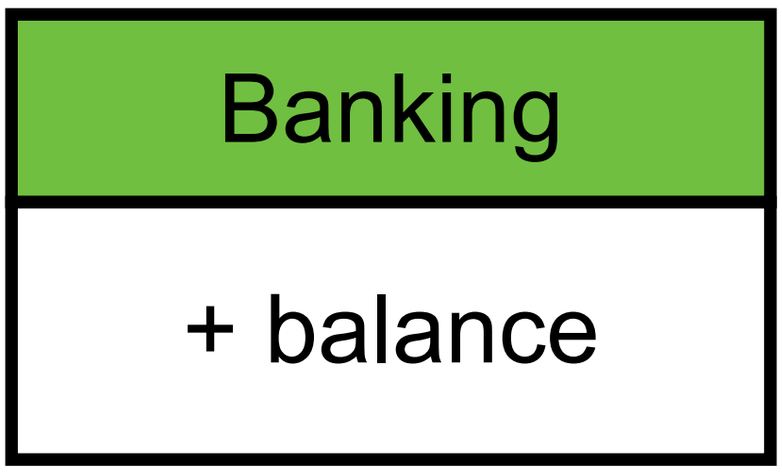
Tool :: Field alignment

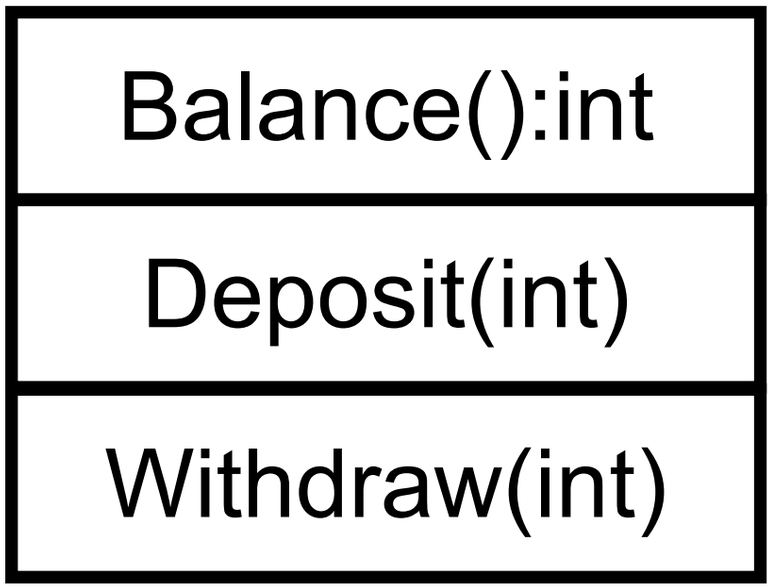
$go install golang.org/x/tools/go/analysis/passes/fieldalignment/cmd/fieldalignment@latest

$fieldalignment -fix demo.go

https://www.somkiat.cc/go-struct-field-alignment/

Workshop with Struct





Workshop with Struct

```go
package main

import "fmt"

type THB int

func (thb THB) String() string {
    return fmt.Sprintf("%d THB", thb)
}

type Banking struct {
    balance THB
}

func (b *Banking) Balance() THB {
    return b.balance
}

func (b *Banking) Deposit(amount THB) {
    b.balance += amount
}

func (b *Banking) Withdraw(amount THB) {
    b.balance -= amount
}
```

Workshop with Struct

Create custom type from integer

```go
package main

import "fmt"

type THB int

func (thb THB) String() string {
    return fmt.Sprintf("%d THB", thb)
}
```

Workshop with Struct

Implement from Stringer interface

```go
package main

import "fmt"

type THB int

func (thb THB) String() string {
    return fmt.Sprintf("%d THB", thb)
}
```

https://pkg.go.dev/fmt#Stringer

Workshop with Struct

Create struct and receiver

```go
type Banking struct {
    balance THB
}

func (b *Banking) Balance() THB {
    return b.balance
}

func (b *Banking) Deposit(amount THB) {
    b.balance += amount
}

func (b *Banking) Withdraw(amount THB) {
    b.balance -= amount
}
```

How to testing ?

Testing with Deposit

```go
func TestBankingDeposit(t *testing.T) {
    // Arrange
    banking := Banking{}
    // Act
    banking.Deposit(THB(1000))
    got := banking.Balance()
    // Assert
    want := THB(1000)
    if got != want {
        t.Errorf("got %s want %s", got, want)
    }
}
```

```
$go test -v
```

Testing with Withdraw

```go
func TestBankingWithdraw(t *testing.T) {
    // Arrange
    banking := Banking{
        balance: 5000,
    }
    // Act
    banking.Withdraw(THB(1000))
    got := banking.Balance()
    // Assert
    want := THB(4000)
    if got != want {
        t.Errorf("got %s want %s", got, want)
    }
}
```

```
$go test -v
```

Try by yourself

Remove pointer (*)

```
type Banking struct {
    balance THB
}

func (b Banking) Balance() THB {
    return b.balance
}

func (b Banking) Deposit(amount THB) {
    b.balance += amount
}

func (b Banking) Withdraw(amount THB) {
    b.balance -= amount
}
```

```
$go test -v
```

Result !!

```
--- FAIL: TestBankingDeposit (0.00s)
    banking_test.go:16: got 0 THB want 1000 THB
--- FAIL: TestBankingWithdraw (0.00s)
    banking_test.go:31: got 5000 THB want 4000 THB
FAIL
exit status 1
FAIL    demo    0.460s
```

Why ?

Print memory address

```go
func (b Banking) Deposit(amount THB) {
    fmt.Printf("Address of balance in Deposit is %p \n", &b.balance)
    b.balance += amount
}
```

```go
func TestBankingDeposit(t *testing.T) {
    // Arrange
    banking := Banking{}
    // Act
    …

    fmt.Printf("Address of balance in Deposit is %p \n", &banking.balance)


    …
}
```

Result

```
Address of balance in Deposit is 0x14000016128
Address of balance in Deposit is 0x14000016120
--- FAIL: TestBankingDeposit (0.00s)
    banking_test.go:18: got 0 THB want 1000 THB
--- FAIL: TestBankingWithdraw (0.00s)
    banking_test.go:33: got 5000 THB want 4000 THB
FAIL
exit status 1
FAIL    demo    0.457s
```

Test Coverage with Go

Test Coverage

$go test -v -cover

```
=== RUN   TestBankingDeposit
--- PASS: TestBankingDeposit (0.00s)
=== RUN   TestBankingWithdraw
--- PASS: TestBankingWithdraw (0.00s)
PASS
        demo    coverage: 75.0% of statements
ok      demo    0.093s
```

Test Coverage

Generate coverage report (HTML)

```
$go test -coverprofile=coverage.out
$go tool cover -html=coverage.out
```

Test Coverage

Generate coverage report (HTML)

demo/banking.go (75.0%) not tracked not covered covered

```go
package main

import "fmt"

type THB int

func (thb THB) String() string {
        return fmt.Sprintf("%d THB", thb)
}

type Banking struct {
        balance THB
}

func (b *Banking) Balance() THB {
        return b.balance
}

func (b *Banking) Deposit(amount THB) {
        b.balance += amount
}

func (b *Banking) Withdraw(amount THB) {
        b.balance -= amount
}
```

Improve your test

Improve your test

Use sub-test
Refactor assertion

Improve your test

```go
func TestBanking(t *testing.T) {

    assertBalance := func(t testing.TB, b Banking, want THB) {
        t.Helper()
        got := b.Balance()
        if got != want {
            t.Errorf("got %s want %s", got, want)
        }
    }

}
```

Create custom assertion :: easy to read

Improve your test

```go
func TestBanking(t *testing.T) {
    assertBalance := func(t testing.TB, b Banking, want THB) {
        t.Helper()
        got := b.Balance()
        if got != want {
            t.Errorf("got %s want %s", got, want)
        }
    }

    t.Run("deposit", func(t *testing.T) {
        b := Banking{}
        b.Deposit(THB(1000))
        assertBalance(t, b, THB(1000))
    })

    t.Run("withdraw", func(t *testing.T) {
        b := Banking{balance: 5000}
        b.Withdraw(THB(1000))
        assertBalance(t, b, THB(4000))
    })
}
```

Sub-test

https://go.dev/blog/subtests

Add new feature with test ?

Withdraw :: not enough money

```go
func TestBanking(t *testing.T) {

    t.Run("withdraw not enough money", func(t *testing.T) {

        startingBalance := THB(1000)
        b := Banking{startingBalance}

        err := b.Withdraw(THB(2000))

        assertBalance(t, b, startingBalance)

        if err == nil {
            t.Error("wanted an error but didn't get one")
        }
    })

}
```

Return error !!

Withdraw()

Use errors package

```go
func (b *Banking) Withdraw(amount THB) error {

    if amount > b.balance {
        return errors.New("Money not enough")
    }

    b.balance -= amount
    return nil
}
```

Return error

https://pkg.go.dev/errors

Run test :: passed

Refactor code

Production code

Define error into variable

```go
var ErrNotEnoughMoney = errors.New("Cannot withdraw, money not enough")

func (b *Banking) Withdraw(amount THB) error {
    if amount > b.balance {
        return ErrNotEnoughMoney
    }

    b.balance -= amount
    return nil
}
```

https://github.com/uber-go/guide/blob/master/style.md#errors

Test code

```go
func TestBanking(t *testing.T) {

    t.Run("withdraw not enough money", func(t *testing.T) {

        startingBalance := THB(1000)
        b := Banking{startingBalance}

        err := b.Withdraw(THB(2000))

        assertBalance(t, b, startingBalance)

        if err == nil {
            t.Error("wanted an error but didn't get one")
        }
    })

}
```

Assert with Error

```go
func assertBalance(t testing.TB, b Banking, want THB) {
    t.Helper()
    got := b.Balance()

    if got != want {
        t.Errorf("got %q want %q", got, want)
    }
}

func assertError(t testing.TB, got, want error) {
    t.Helper()
    if got == nil {
        t.Fatal("didn't get an error but wanted one")
    }

    if got != want {
        t.Errorf("got %q, want %q", got, want)
    }
}
```

Test code

```
t.Run("withdraw", func(t *testing.T) {
    b := Banking{balance: 5000}
    err := b.Withdraw(THB(1000))

    assertBalance(t, b, THB(4000))
    assertNoError(t, err)
})

t.Run("withdraw not enough money", func(t *testing.T) {
    startingBalance := THB(1000)
    b := Banking{startingBalance}
    err := b.Withdraw(THB(2000))

    assertBalance(t, b, startingBalance)
    assertError(t, err, ErrNotEnoughMoney)
})
```

More testing library

https://github.com/stretchr/testify

How to checking for unchecked errors in Go code ?

Use errcheck

```
$go install github.com/kisielk/errcheck@latest
$errcheck .
```

https://github.com/kisielk/errcheck

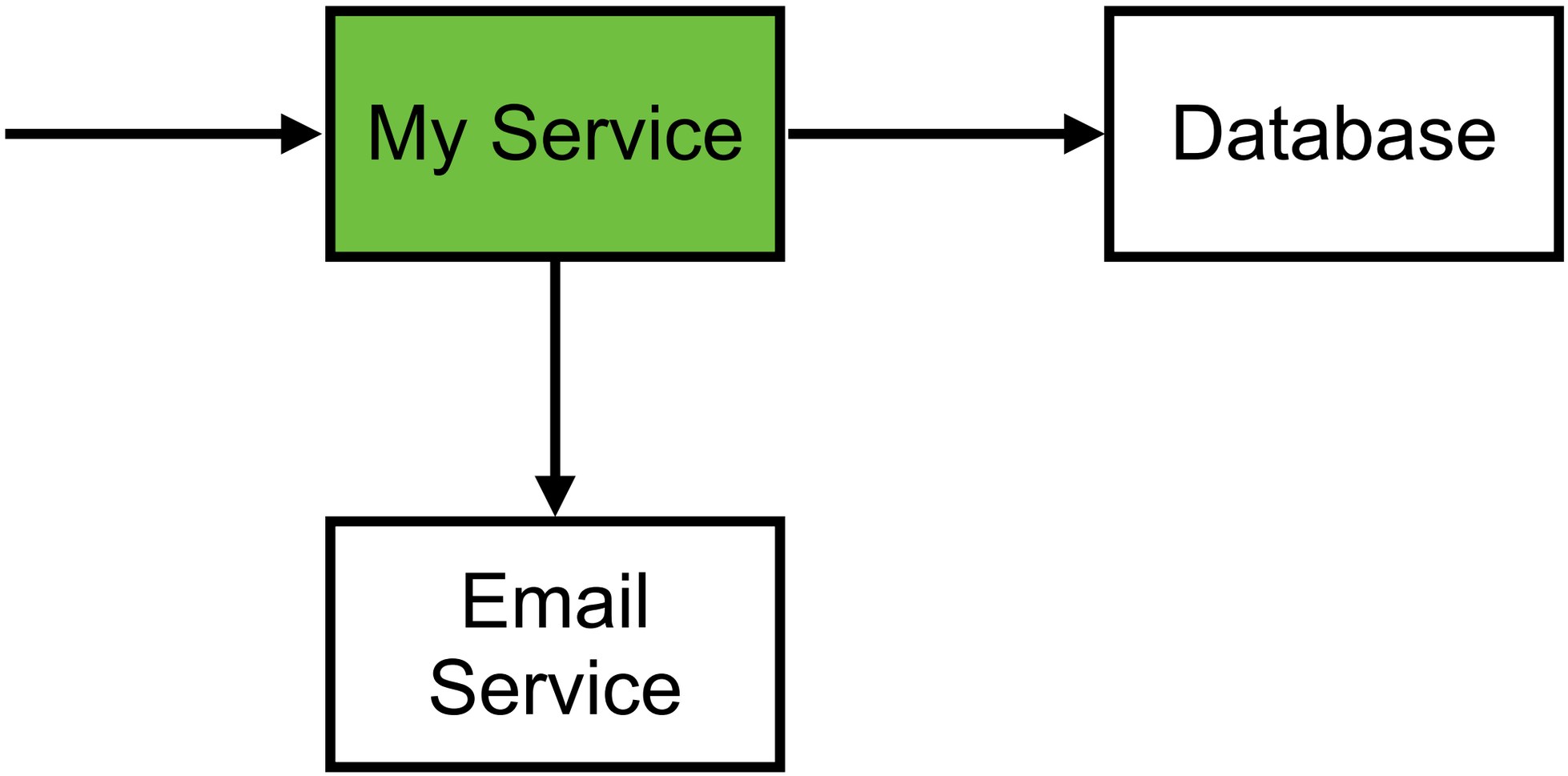
Don't just check errors, handle them gracefully 🌸

– Go Proverb

https://dave.cheney.net/2016/04/27/dont-just-check-errors-handle-them-gracefully

Project structure with Testable !!

Structure of project



Let’s Start

```go
type MyService struct {

}

func (s *MyService) Process() error {
    // 1. Create database connection
    // 2. Insert data into database
    // 3. Create email service
    // 4. Send email
}
```

Design with struct

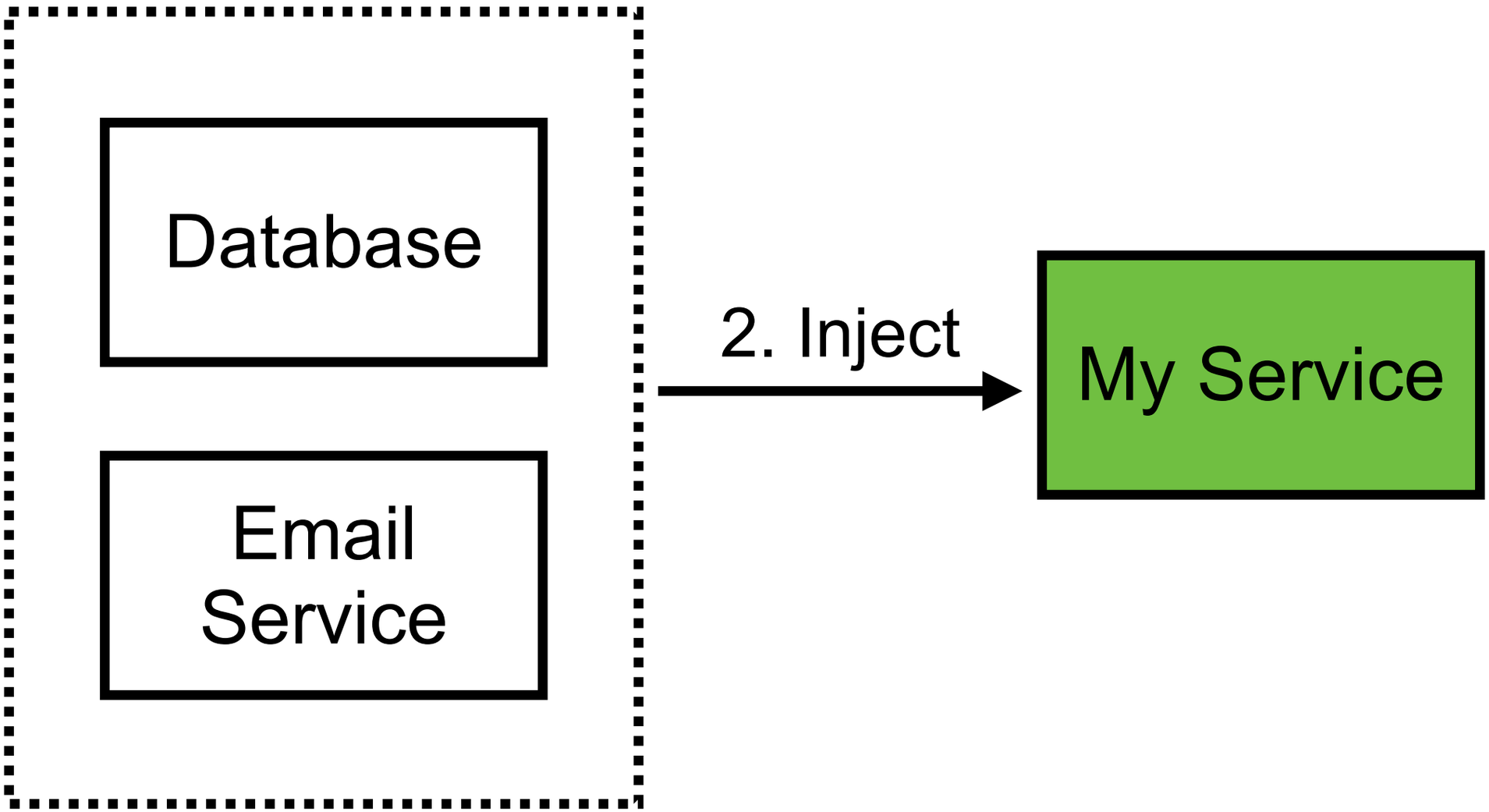
```go
type Database struct {
}

func (d *Database) connect() error {
    return nil
}
func (d *Database) insertData() error {
    return nil
}
```

```go
type EmailService struct {
}

func (e *EmailService) connect() error {
    return nil
}

func (e *EmailService) send() error {
    return nil
}
```

How to create database and email service ?

Use dependency injection

1. Create object of dependencies



Use dependency injection

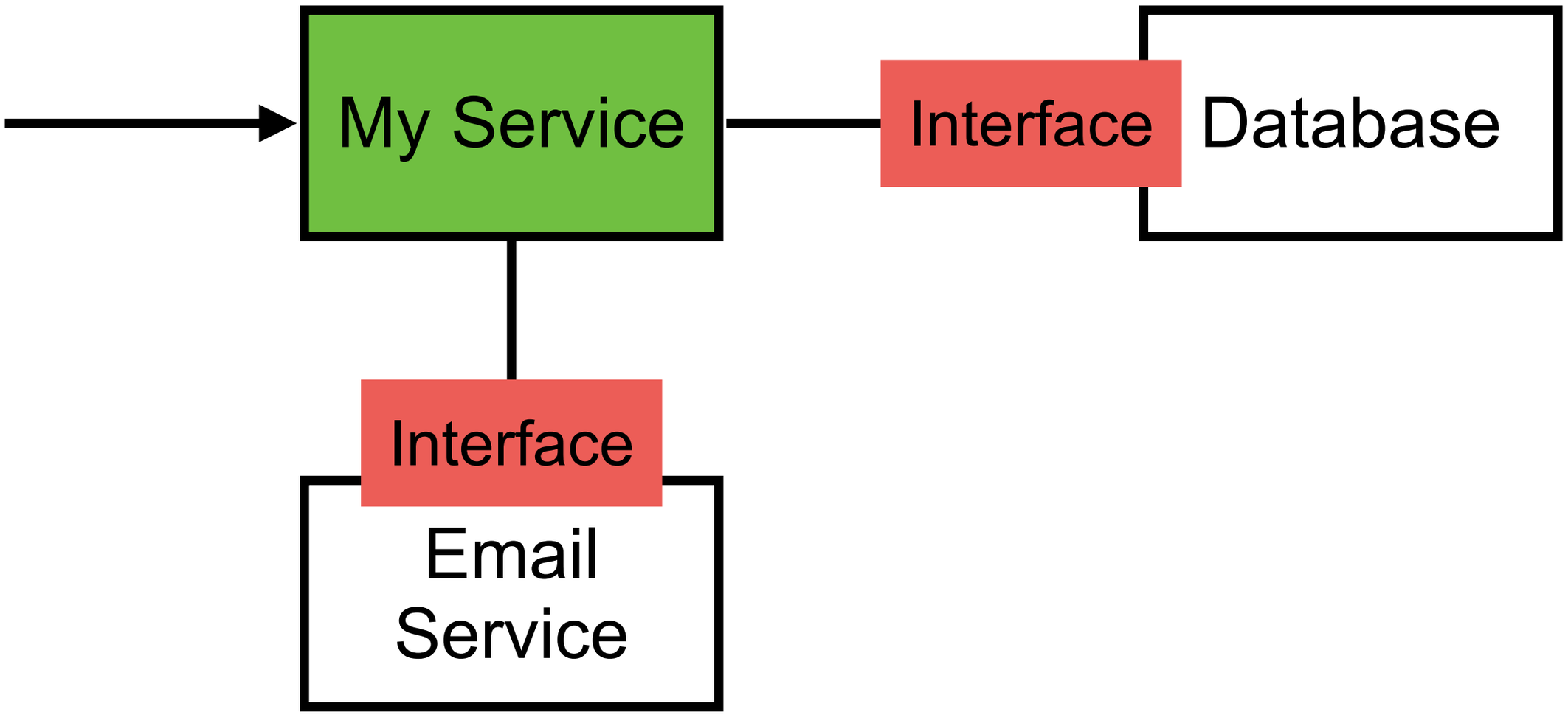
```go
type MyService struct {
    Db    *Database
    Email *EmailService
}

func (s *MyService) Process() error {
    // 1. Create database connection
    s.Db.connect()
    // 2. Insert data into database
    s.Db.insertData()
    // 3. Create email service
    s.Email.connect()
    // 4. Send email
    s.Email.send()
}
```

Create MySevice with dependencies

```
func NewMyService(db *Database, email *EmailService) *MyService
{
    return &MyService{
        Db:    db,
        Email: email,
    }
}
```

How to test ?
Easy to test ?
Testable code ?

Working with interface ?



Working with interface ?

```go
type IDatabase interface {
    connect() error
    insertData() error
}

type IEmailService interface {
    connect() error
    send() error
}
```

```go
type MyService struct {
    Db    IDatabase
    Email IEmailService
}

func NewMyService(db IDatabase, email IEmailService) *MyService {
    return &MyService{
        Db:    db,
        Email: email,
    }
}
```

Try to test with Unit test

```go
type MockDatabase struct{}

func (d *MockDatabase) connect() error {
    return nil
}
func (d *MockDatabase) insertData() error {
    return nil
}



type MockEmailService struct{}

func (e *MockEmailService) connect() error {
    return nil
}

func (e *MockEmailService) send() error {
    return nil
}
```

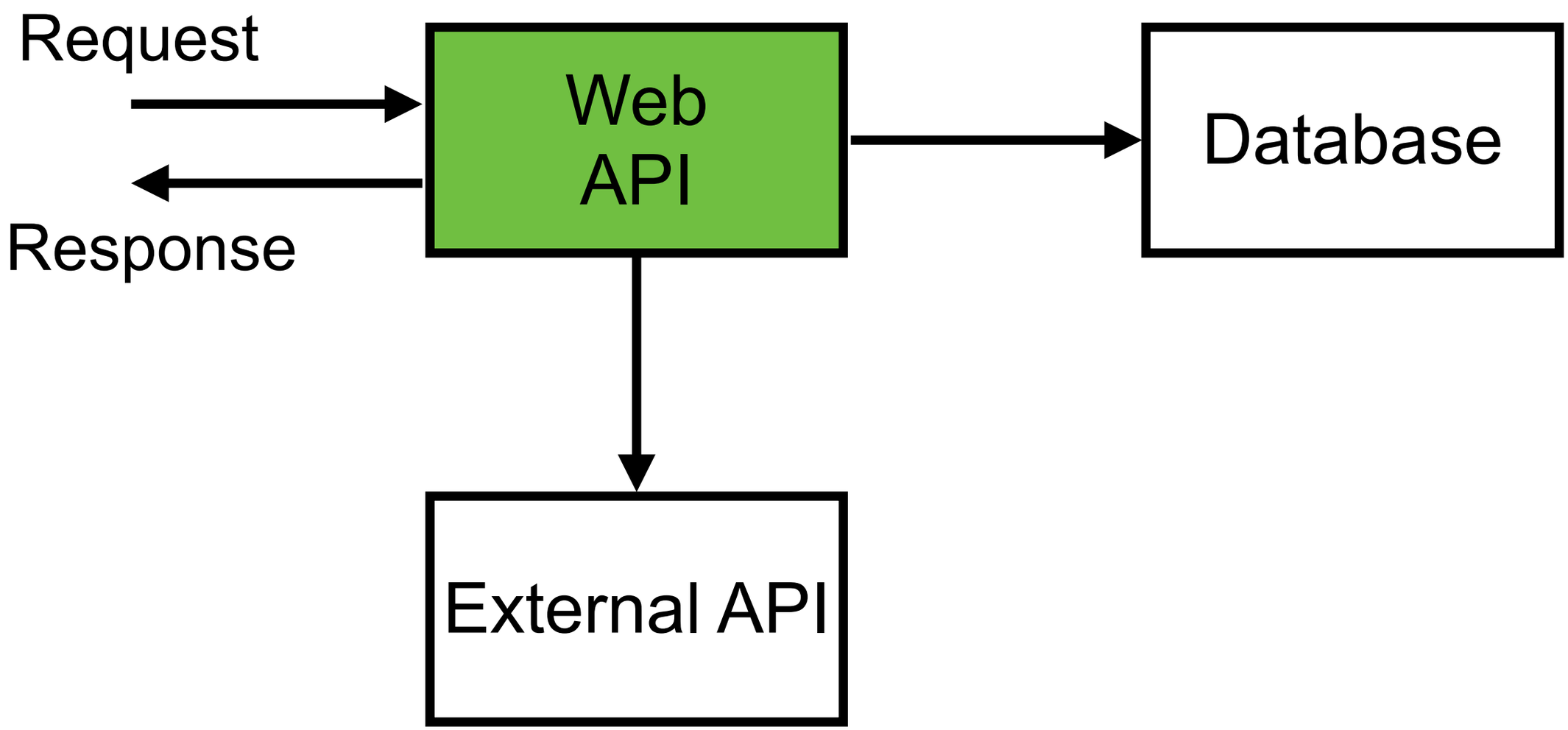
Try to test with Unit test

```go
func TestDI(t *testing.T) {

    db := &MockDatabase{}
    email := &MockEmailService{}
    service := NewMyService(db, email)
    service.Process()

}
```

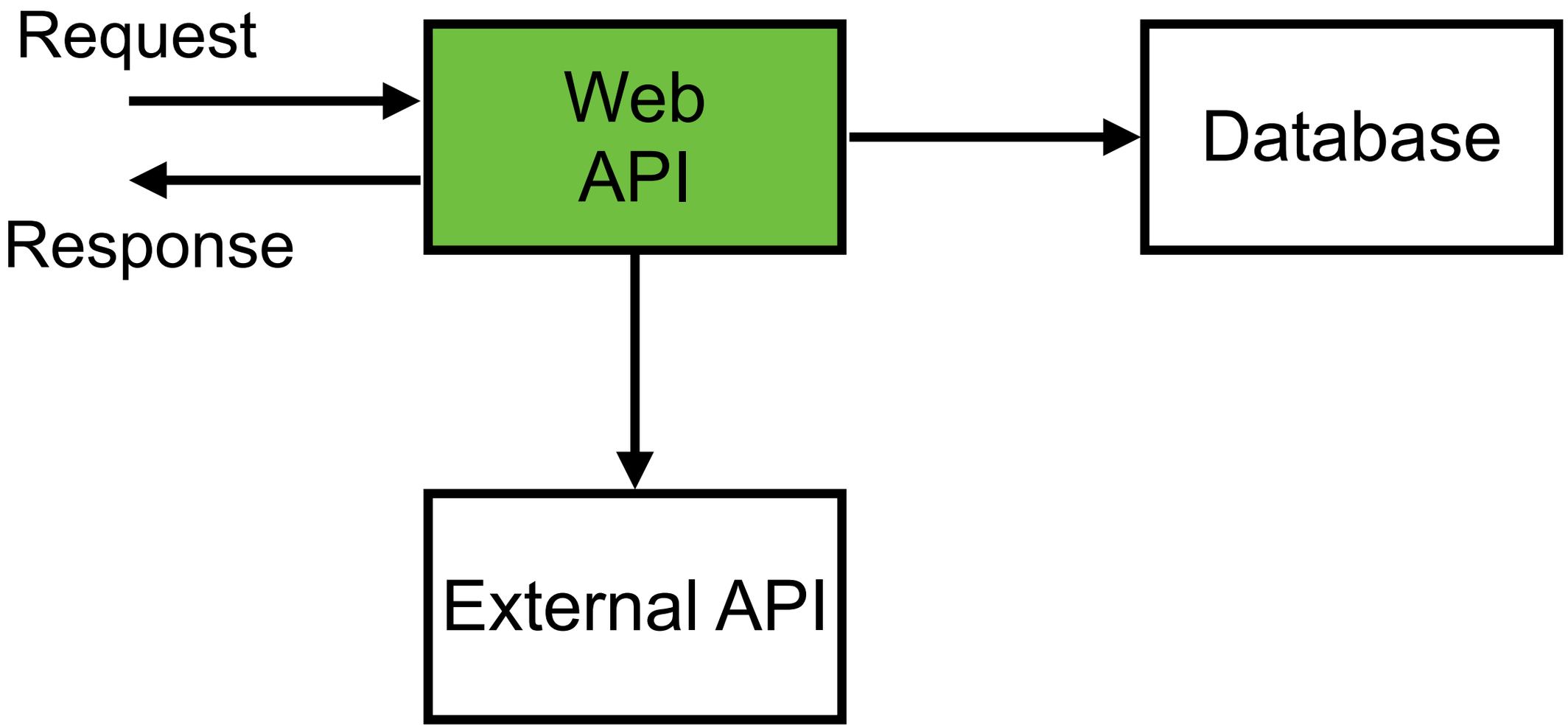
```
$go test -v
```

Test Strategies

Architecture



What to Test ?

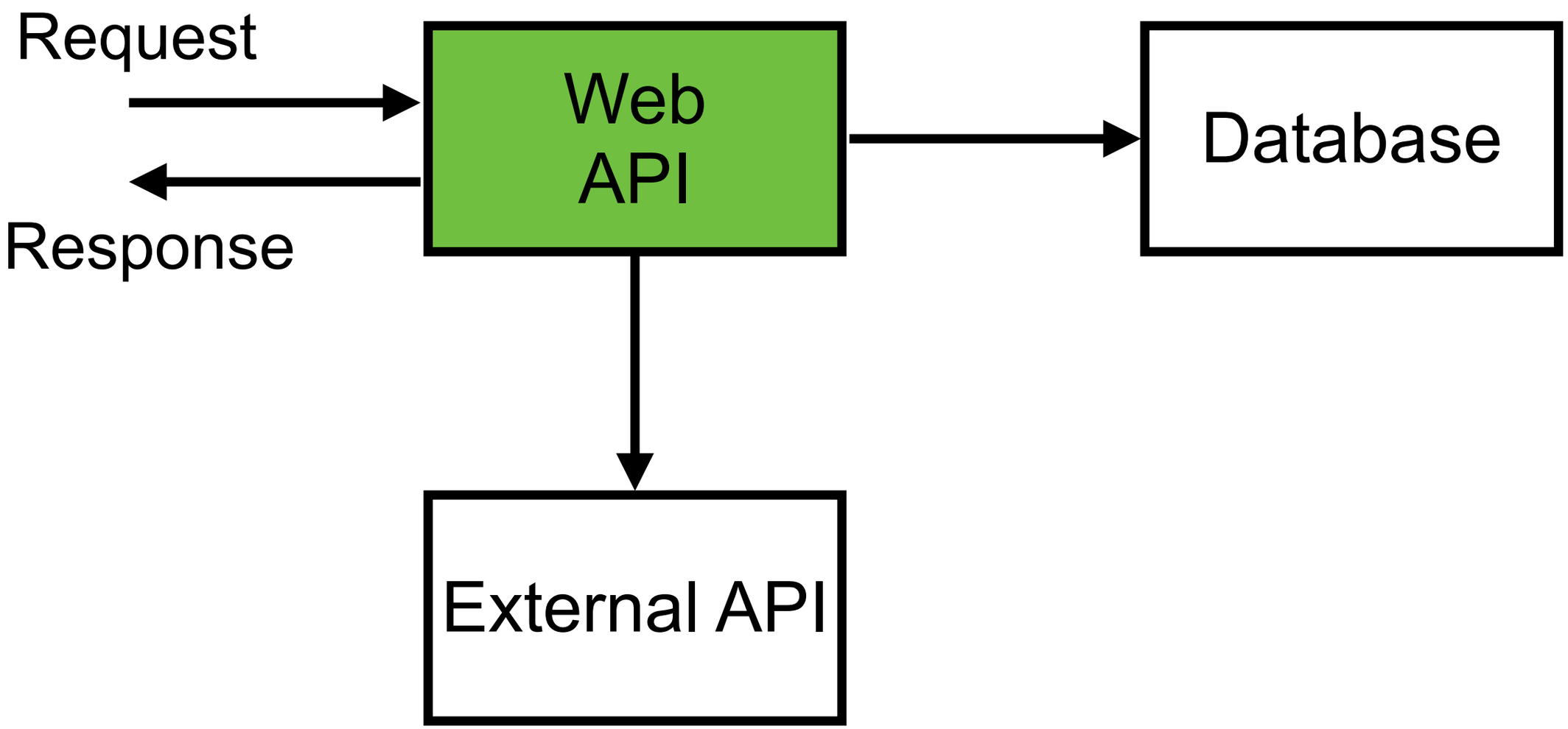


Component and Contract Testing

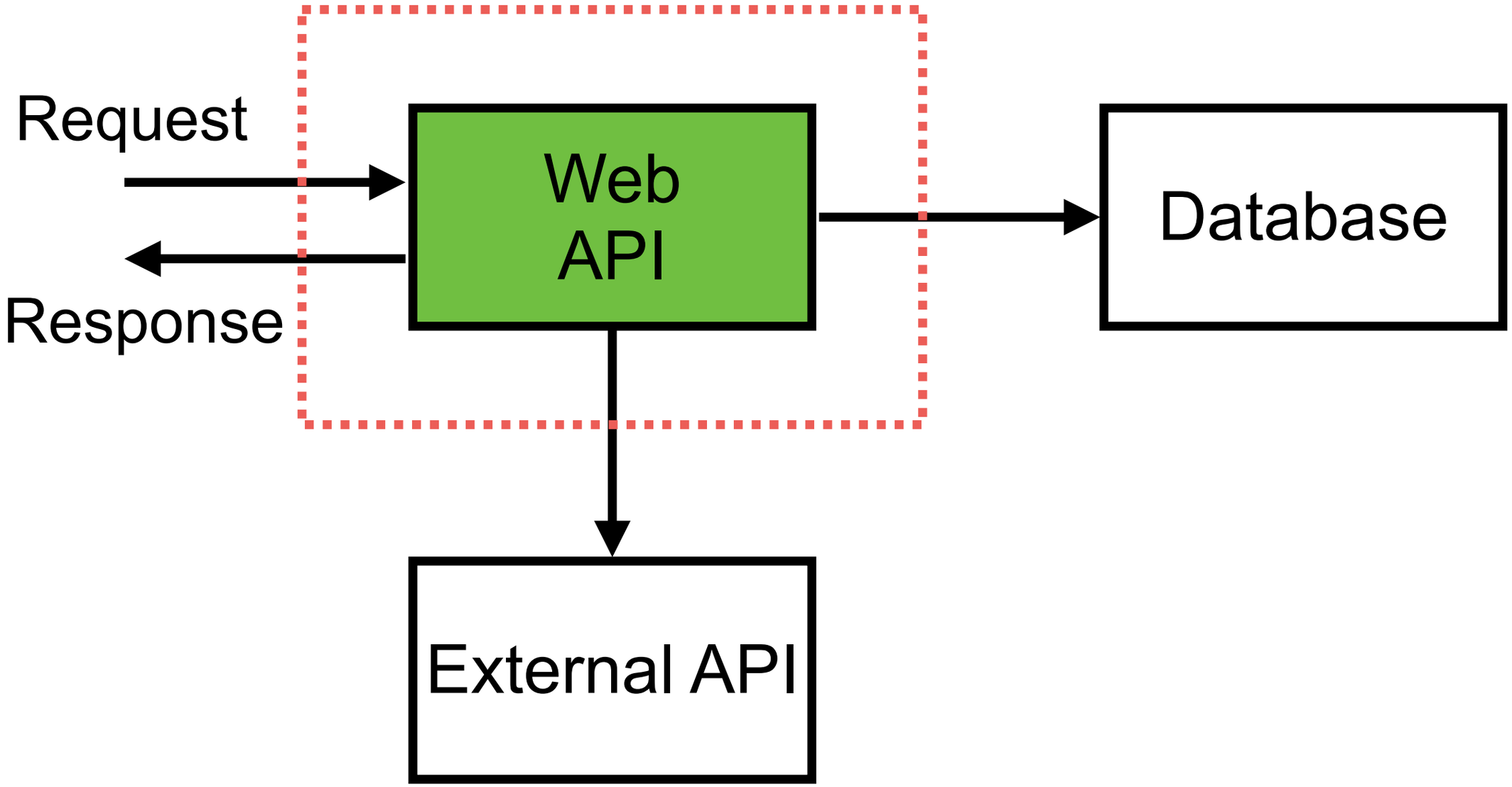
https://github.com/up1/course-contract-testing

Component Testing

Component Testing



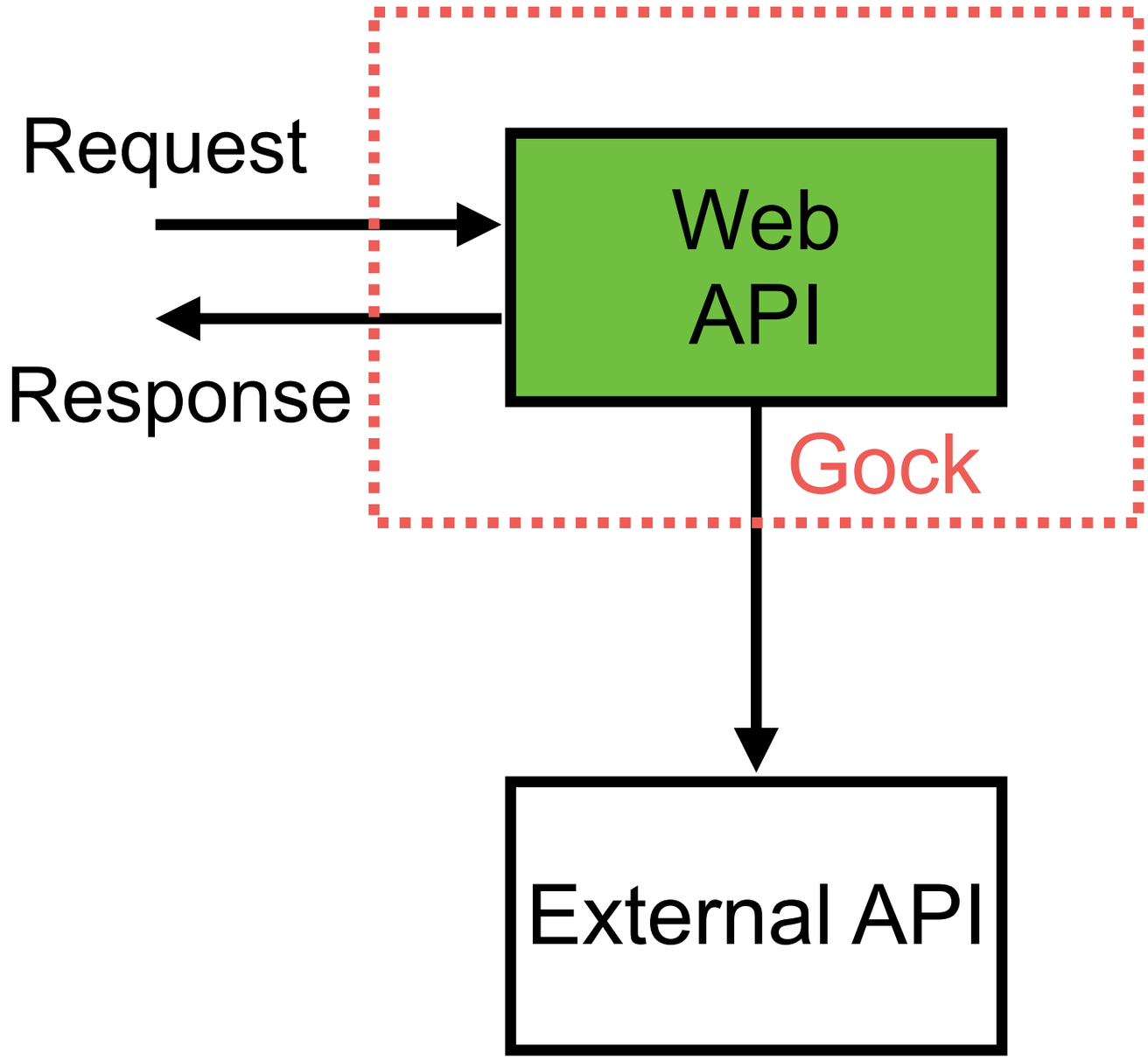
Component Testing



Component testing

In-process
Separate process

In-process with Gock



https://github.com/h2non/gock

Try to test with Gock

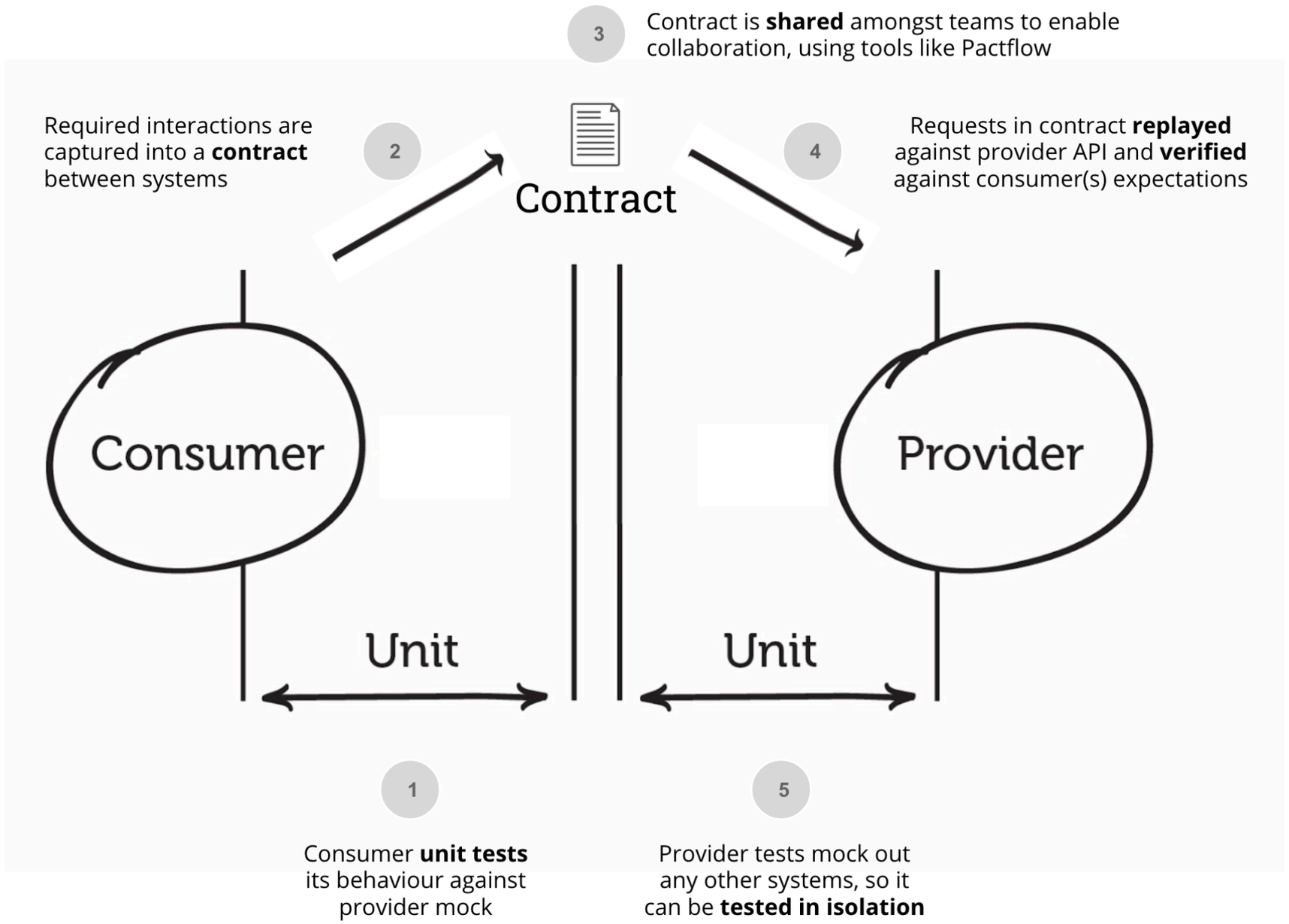
```go
func TestSimple(t *testing.T) {
    defer gock.Off()

    // Arrange
    gock.New("http://facebook.com").
        Get("/bar").
        Reply(200).
        JSON(map[string]string{"foo": "bar"})

    // Act
    res, err := http.Get("http://facebook.com/bar")

    // Assert
    st.Expect(t, err, nil)
    st.Expect(t, res.StatusCode, 200)

    body, _ := ioutil.ReadAll(res.Body)
    st.Expect(t, string(body)[:13], `{"foo":"bar"}`)

    // Verify that we don't have pending mocks
    st.Expect(t, gock.IsDone(), true)
}
```

```
$go test -v
```

Contract Testing

https://github.com/up1/course-contract-testing

Contract Testing



https://docs.pact.io/

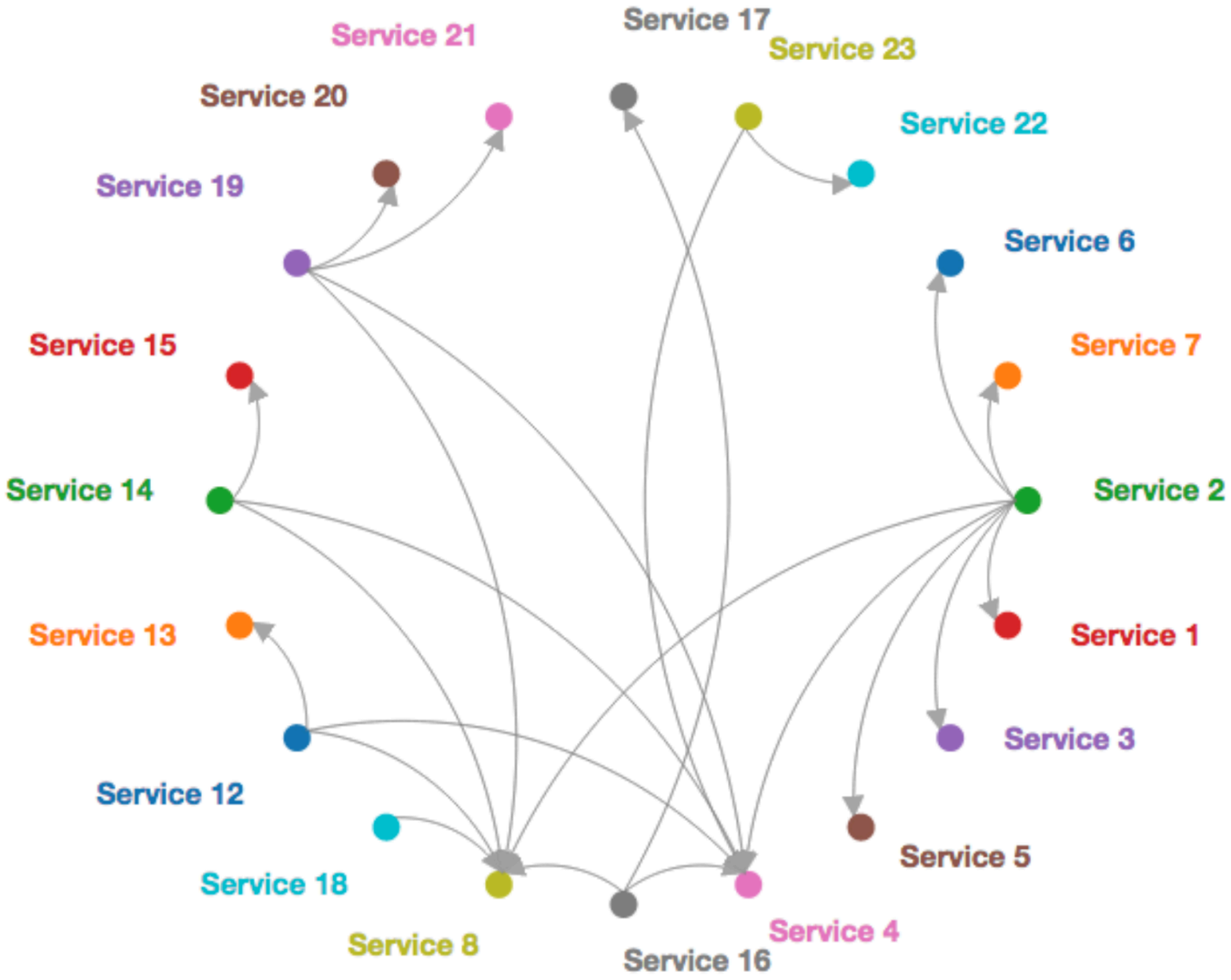
Pact Broker

Pacts

Consumer		Provider	Latest pact published	Last verified
Foo		Animals	2 minutes ago	2 days ago
Foo		Bar	7 days ago	15 days ago ⚠
Foo		Hello World App	1 day ago	
Foo		Wiffles	less than a minute ago	7 days ago
Some other app		A service	26 days ago	less than a minute ago
The Android App		The back end	less than a minute ago	

https://docs.pact.io/

Pact Broker



https://docs.pact.io/

Project Structure

Good structure ?

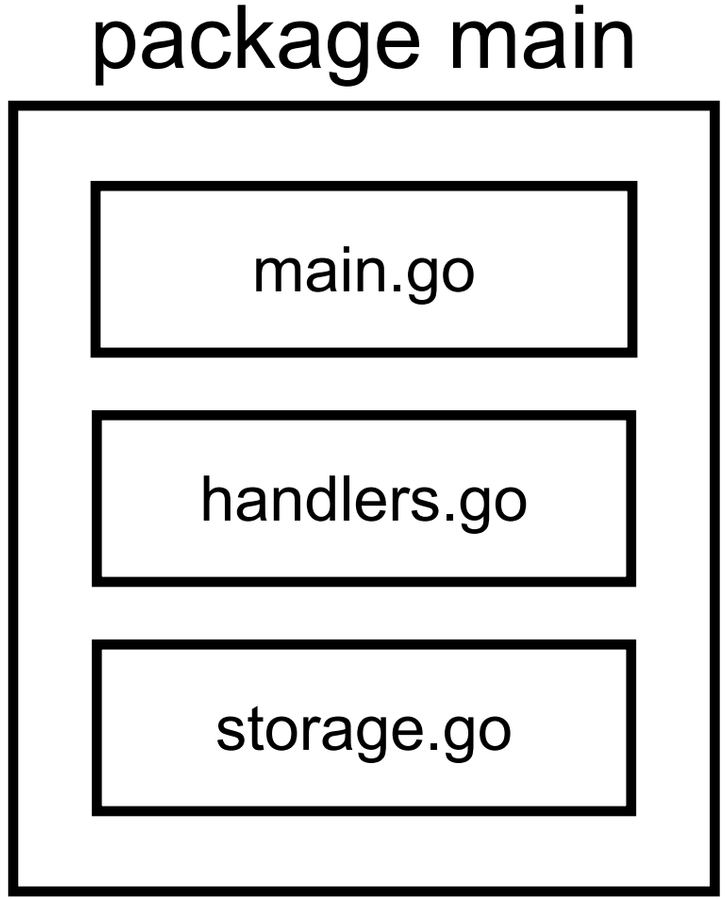
Consistent
Easy to understand, navigate and reason
Easy to change, loosely coupled
Easy to test
Simple as possible
Structure reflects the design !!

Project Structure

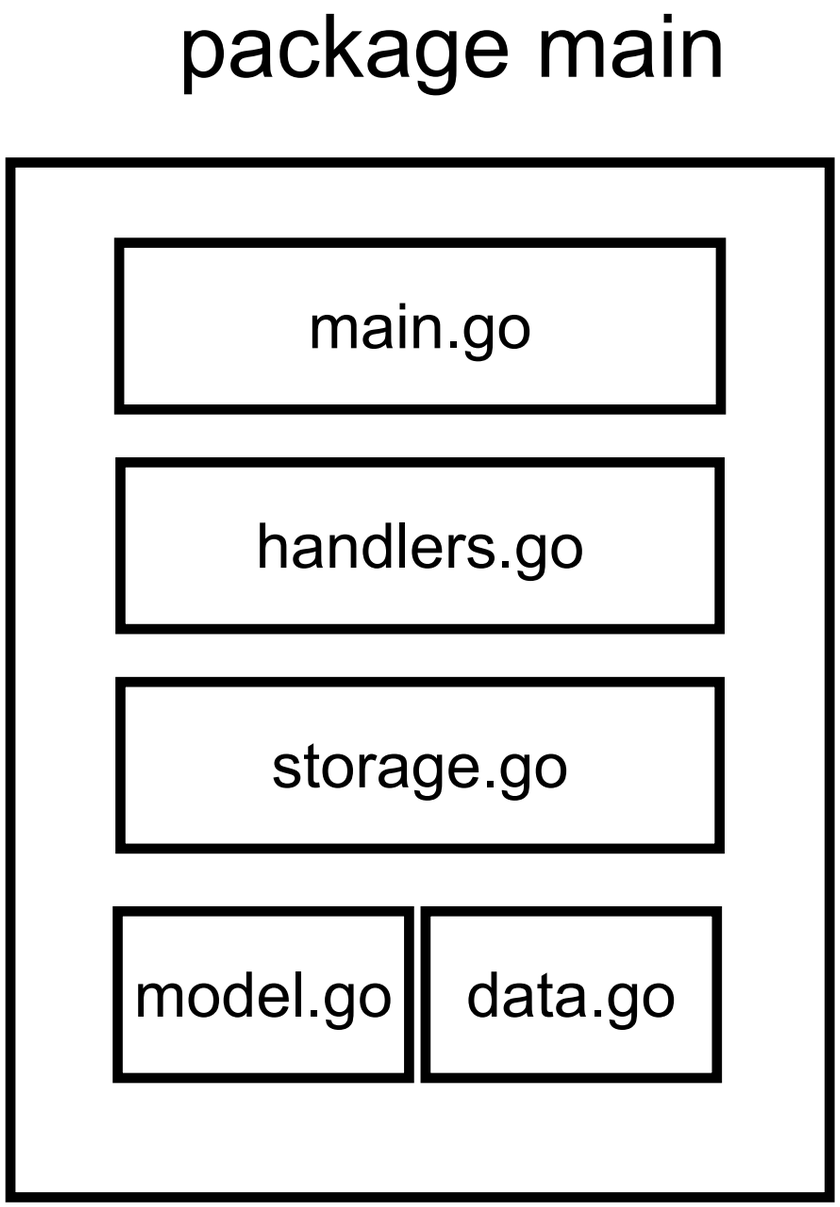
Flat structure
Layer (group of function)
Group by module
Group by context
Clean architecture
Hexagonal architecture

Flat structure

Easy to start
Only main package



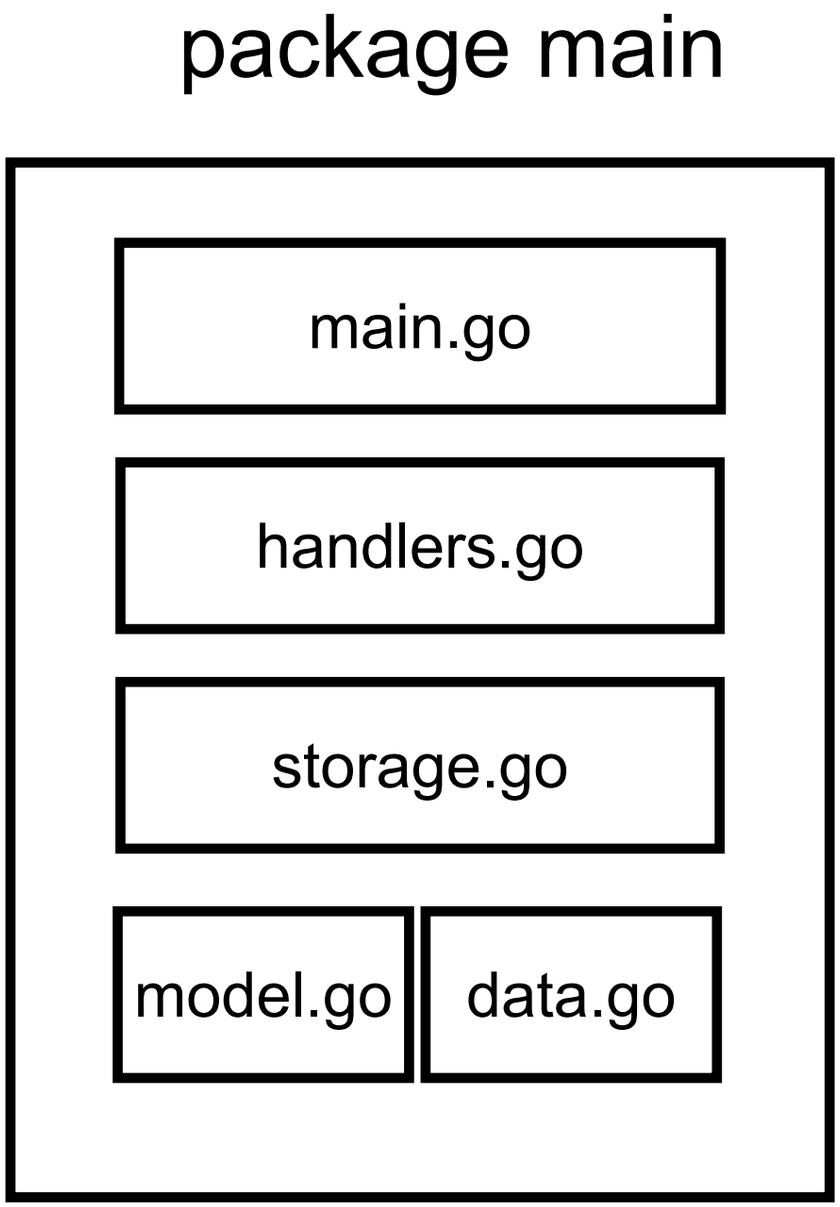
Beer !!



Create Storage as global variable

Start Web server

Beer !!



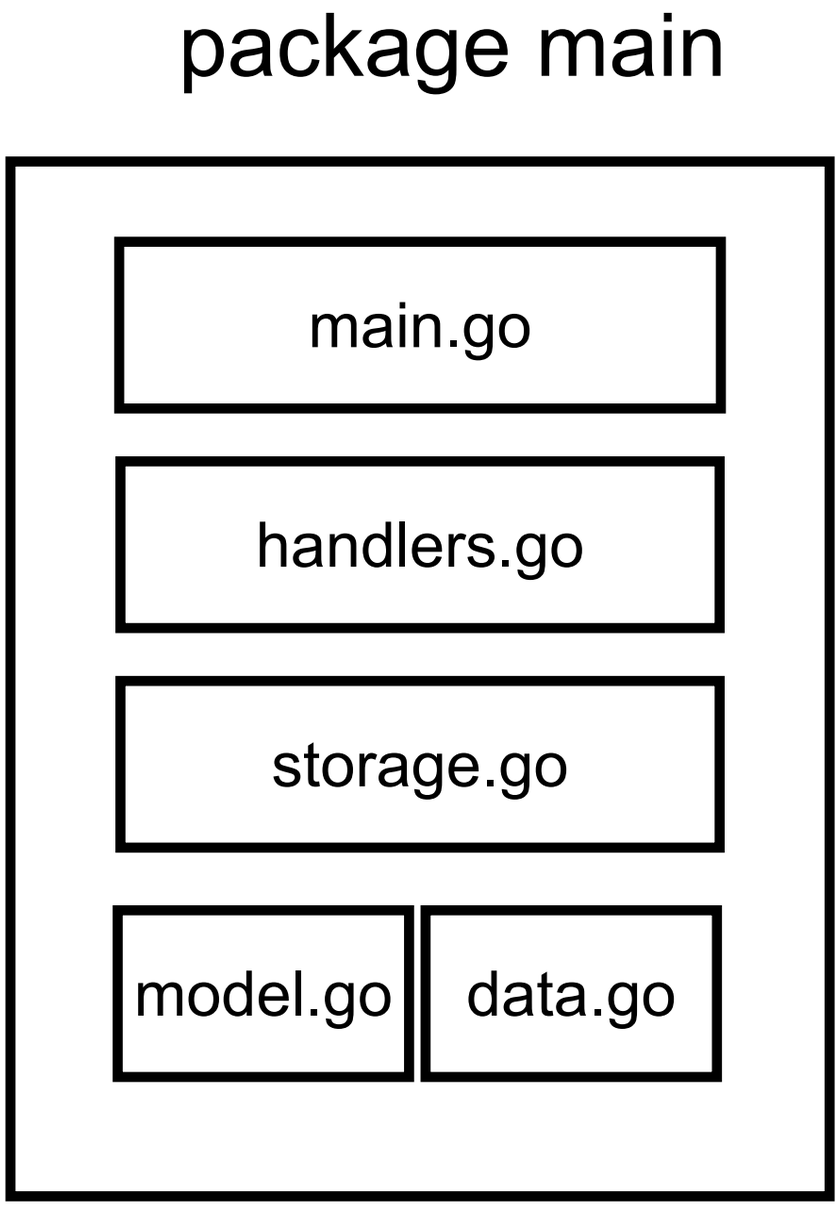
Create Storage as global variable

Start Web server

GET /beers

POST /beers

Beer !!



Create Storage as global variable

Start Web server

GET /beers

POST /beers

GetBeers() from memory

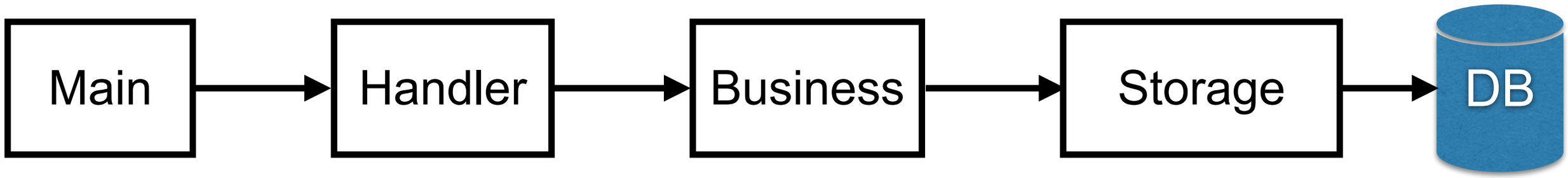
SaveBeer(beer) into memory

How to test ?

Storage
Handler (integration)
Handler (mock)

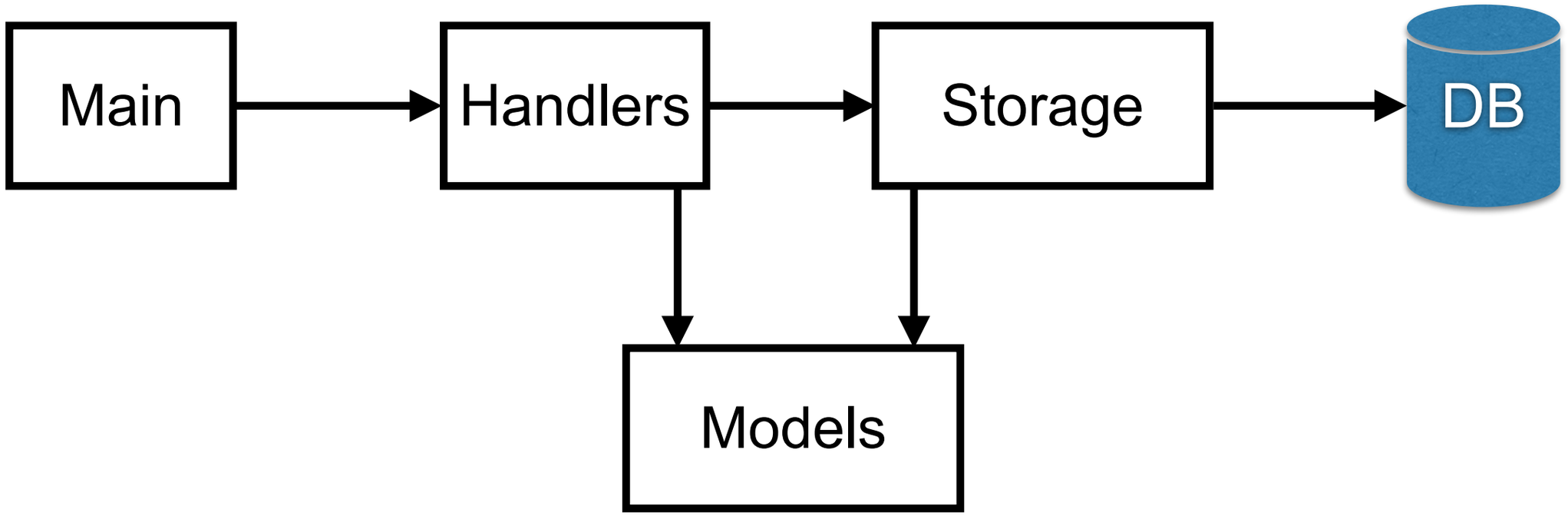
Layer

Grouping by function



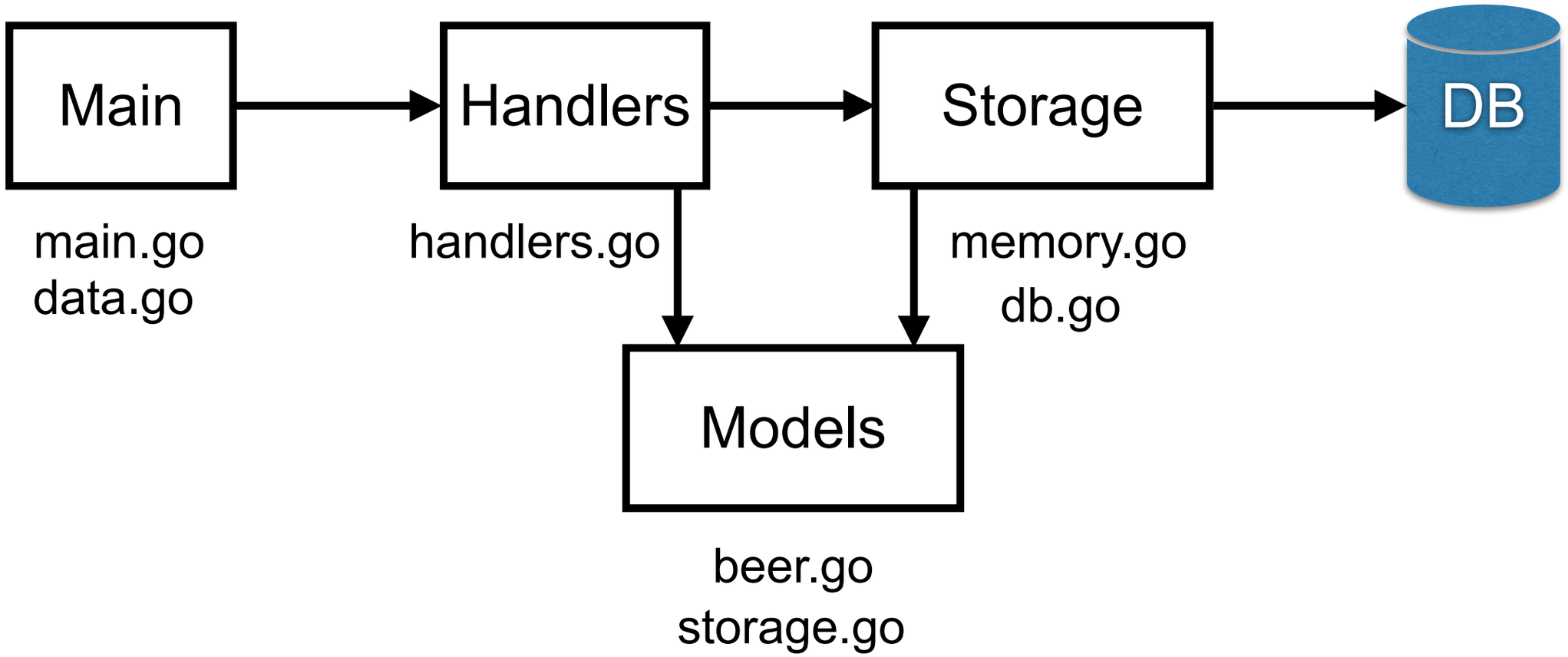
Layer

Grouping by function



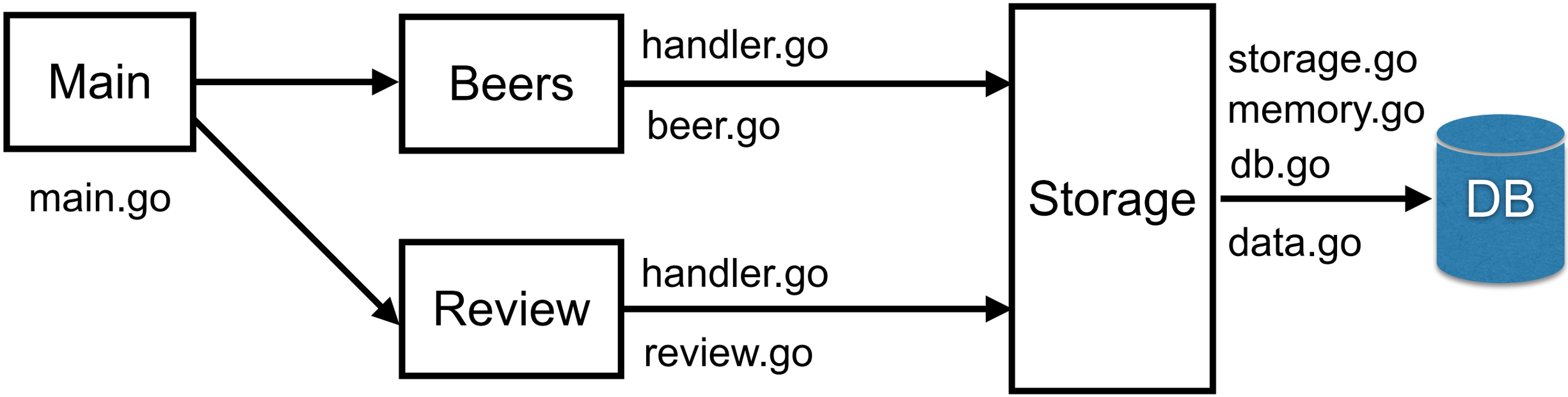
Layer

Grouping by function



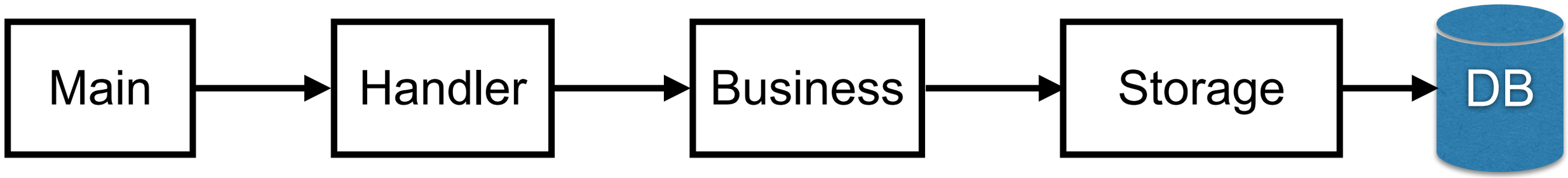
Modular

Grouping by module



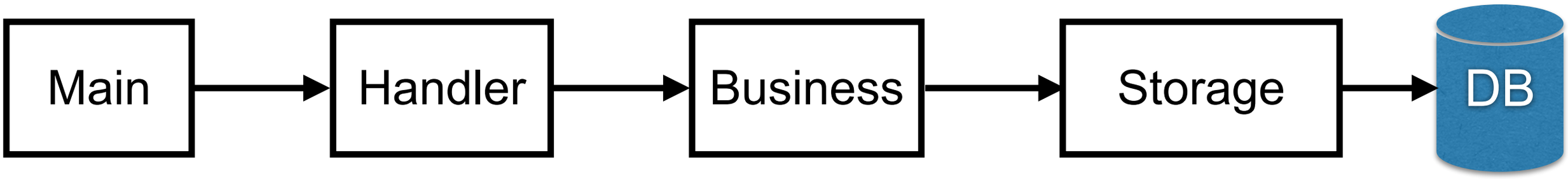
Manage Dependency

Manage Dependency ?



Manage Dependency ?

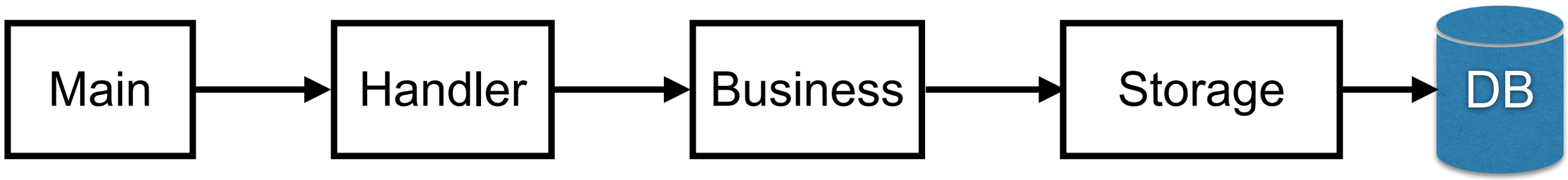
Global variable




```go
// Global variable
var db Storage

func setupRouter() *gin.Engine {
    var err error
    db, err = NewStorage()
    if err != nil {
        log.Fatal(err)
    }
}
```

Manage Dependency ?

Dependency with Struct



```go
type MyConfig struct {
    Db Storage
}

func setupRouter() *gin.Engine {
    db, _ := NewStorage()
    config := &MyConfig{Db: db}

    PopulateBeers(config)

    r := gin.New()
    r.GET("/beers", config.GetBeers)
    r.POST("/beers", config.AddBeer)
    return r
}
```

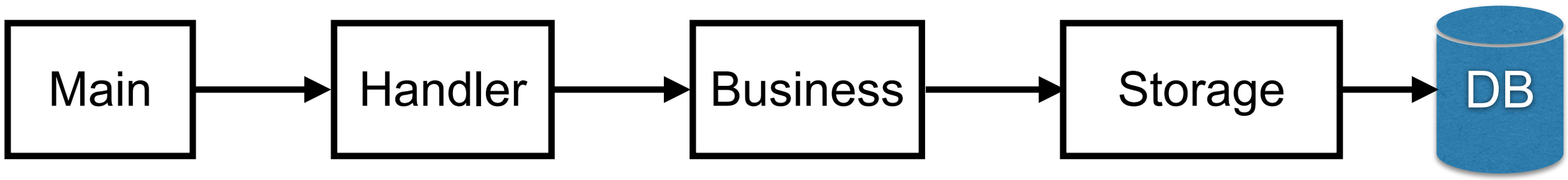
1. Create and initial

2. Add receiver

```go
func (config *MyConfig) GetBeers(c *gin.Context) {
    beers, err := config.Db.FindBeers()
    if err != nil {
        c.AbortWithStatus(http.StatusBadRequest)
        return
    }
    c.JSON(http.StatusOK, beers)
}
```

Manage Dependency ?

Dependency with closure



```go
type MyConfig struct {
    Db Storage
}

func setupRouter() *gin.Engine {
    db, _ := NewStorage()
    config := &MyConfig{Db: db}

    PopulateBeers(config)

    r := gin.New()
    r.GET("/beers", GetBeers(config))
    r.POST("/beers", AddBeer(config))
    return r
}
```

1. Create and initial

2. Use closure

```go
func GetBeers(config *MyConfig) gin.HandlerFunc {
    return func(c *gin.Context) {
        beers, err := config.Db.FindBeers()
        if err != nil {
            c.AbortWithStatus(http.StatusBadRequest)
            return
        }
        c.JSON(http.StatusOK, beers)
    }
}
```

Let’s Go

My journey in Software Development