



Go workshop 2025





Somkiat Puisungnoen

Search

Somkiat | Home

Update Info 1 View Activity Log 10+ ...

Timeline About Friends 3,138 Photos More

When did you work at Opendream? X

... 22 Pending Items

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Intro

Software Craftsmanship

Software Practitioner at สยามชานาญกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Somkiat Puisungnoen 15 mins · Bangkok · ...

Java and Bigdata

When did you work at Opendream? X

... 22 Pending Items

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Intro

Software Craftsmanship

Software Practitioner at สยามชานาญกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Somkiat Puisungnoen 15 mins · Bangkok · ...

Java and Bigdata



Facebook somkiat.cc

Somkiat Home | ? ▾

Page Messages Notifications 3 Insights Publishing Tools Settings Help ▾

somkiat.cc
@somkiat.cc

Home Posts Videos Photos

Liked Following Share ... + Add a Button

Help people take action on this Page. ×



**[https://github.com/up1/
course-go-2025](https://github.com/up1/course-go-2025)**



Topics (1)

Develop Project with Go

 Essential features

 Testing with Go

Testable project with Go

Develop REST API with gin

 Working with Docker

Working with Database (SQL, NoSQL)



Topics (2)

Benchmark and Profiling
Performance Testing
CI/CD for Go project



Topics (3)

Authentication and Authorization
API gateway
Messaging (Asynchronous)



Let's



[Why Go ▾](#)[Learn](#)[Docs ▾](#)[Packages](#)[Community ▾](#)

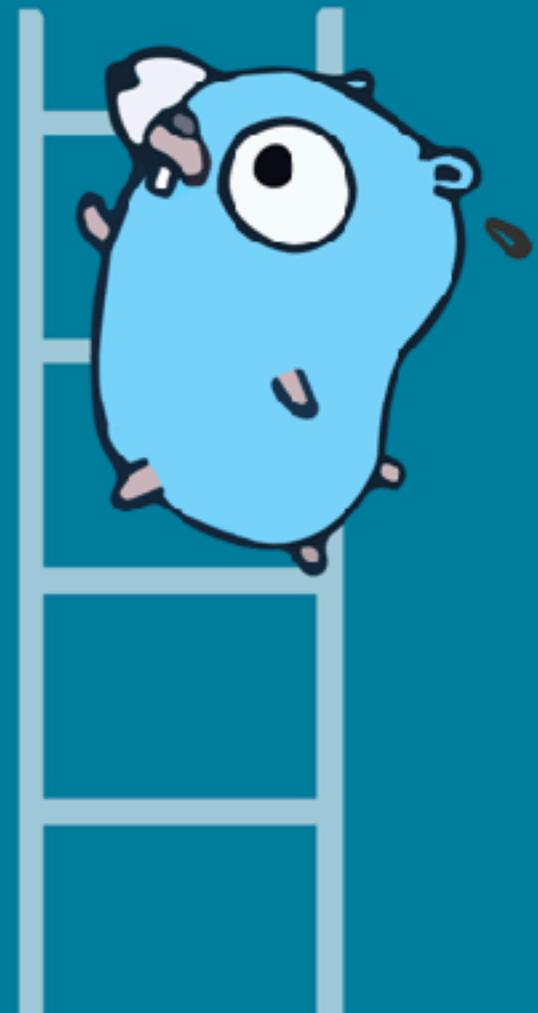
Build simple, secure, scalable systems with Go

- ✓ 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.](#)



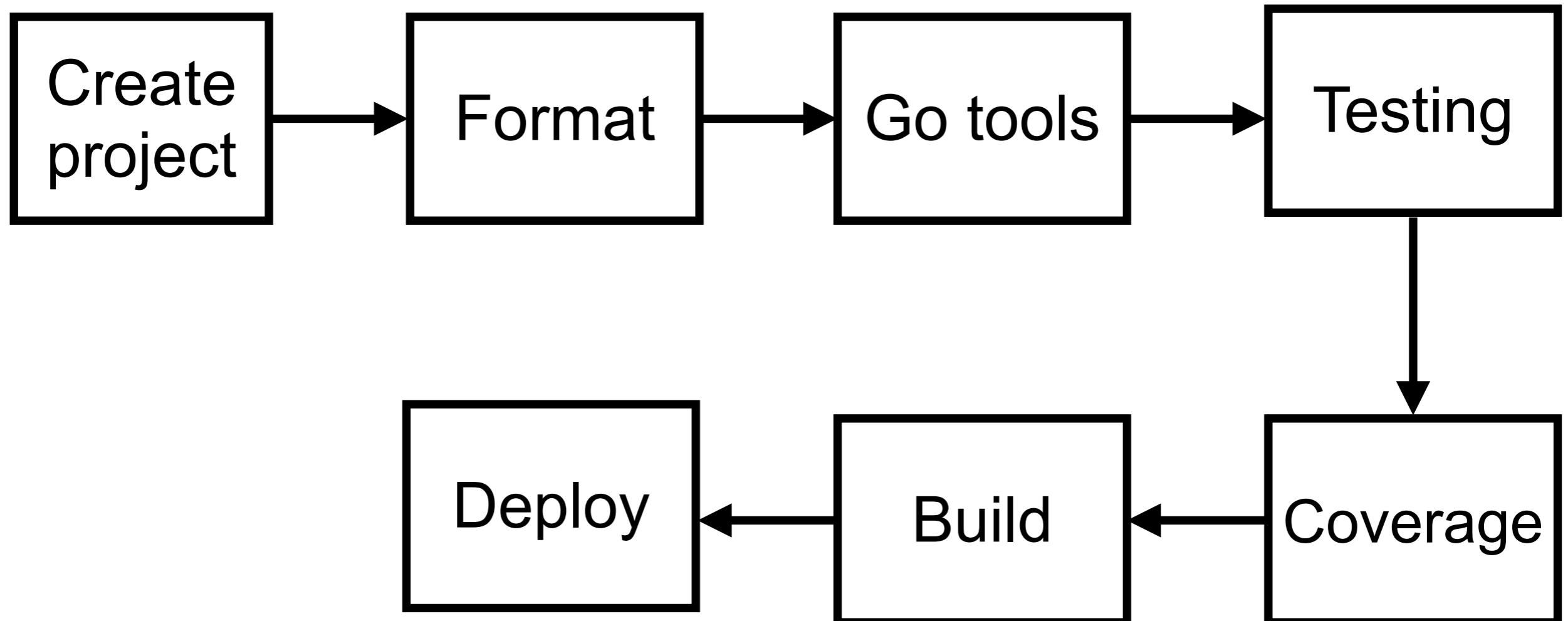
<https://go.dev/>



```
go env -w GOTOOOLCHAIN=go1.25.4  
go version
```



Development workflow



Hello Go !!

```
package main

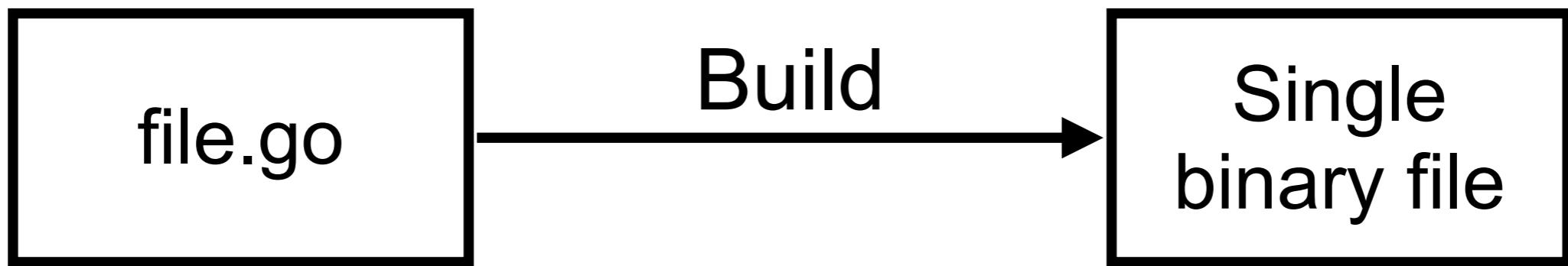
import "fmt"

func main() {
    fmt.Println("Hello, world 2024")
    print("Hello, world 2024")
}
```

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



Basic Go



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



Building for architecture

\$GOOS=?
\$GOARCH=?

GOOS	GOARCH
darwin	amd64 arm64
windows	amd64 arm64 arm
linux	amd64 arm64 arm

<https://go.dev/doc/install/source#environment>



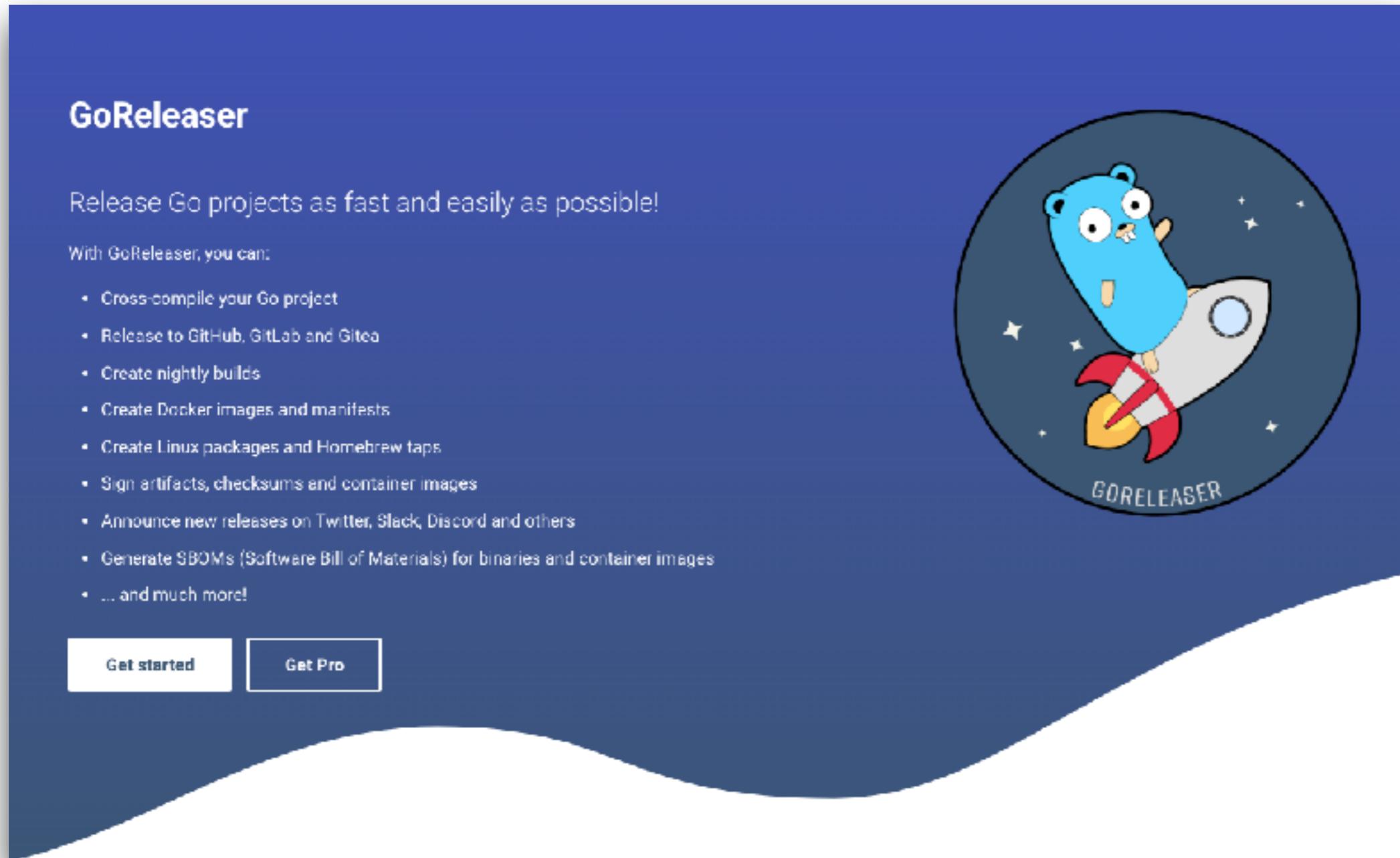
Building for architecture

```
$GOOS=darwin GOARCH=amd64 go build  
$GOOS=darwin GOARCH=arm64 go build  
$GOOS=windows GOARCH=amd64 go build  
$GOOS=windows GOARCH=arm64 go build  
$GOOS=linux GOARCH=amd64 go build  
$GOOS=linux GOARCH=arm64 go build
```

<https://go.dev/doc/install/source#environment>



GoReleaser



The screenshot shows the GoReleaser landing page. At the top left is the "GoReleaser" logo. Below it is the tagline "Release Go projects as fast and easily as possible!". A list of features follows, including cross-compilation, GitHub release, Docker images, and Slack announcements. Two buttons at the bottom are "Get started" and "Get Pro". To the right is a circular illustration of a blue Go gopher riding a white rocket ship against a dark blue background with stars, with the word "GORELEASER" written below it.

<https://goreleaser.com/>



Create go project

```
$go mod init demo
```

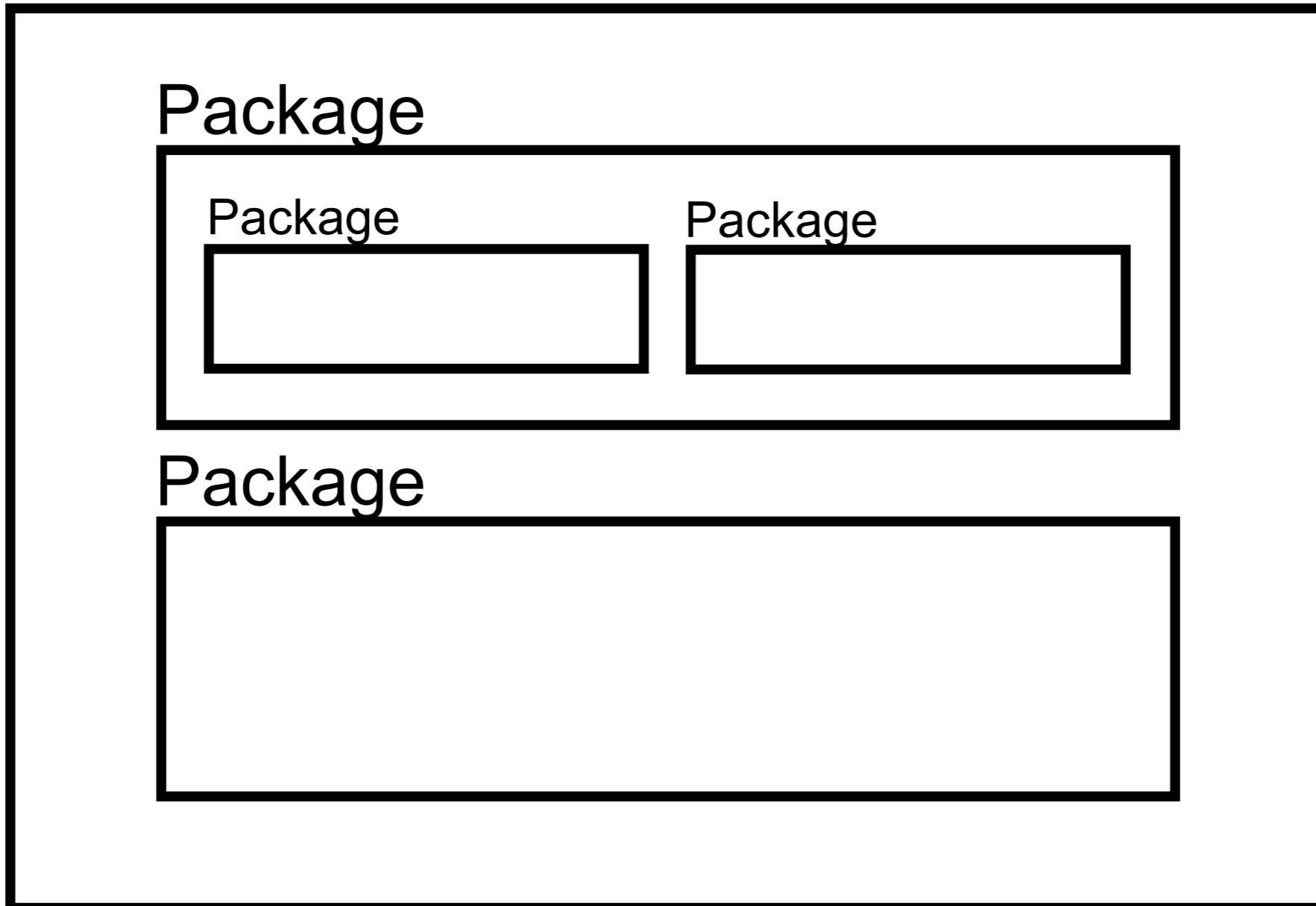
```
$go mod tidy
```

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



Create go project

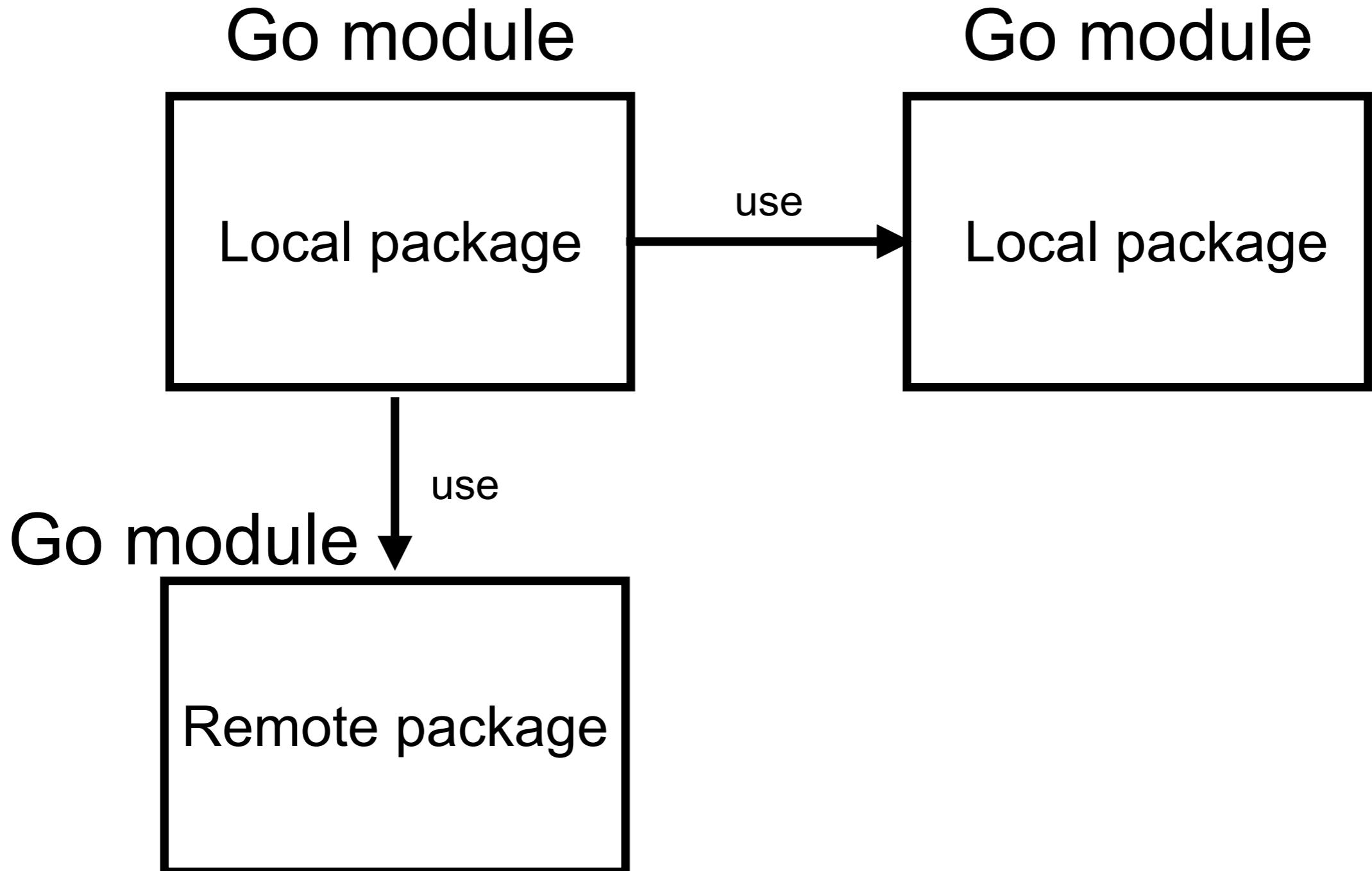
Go module



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



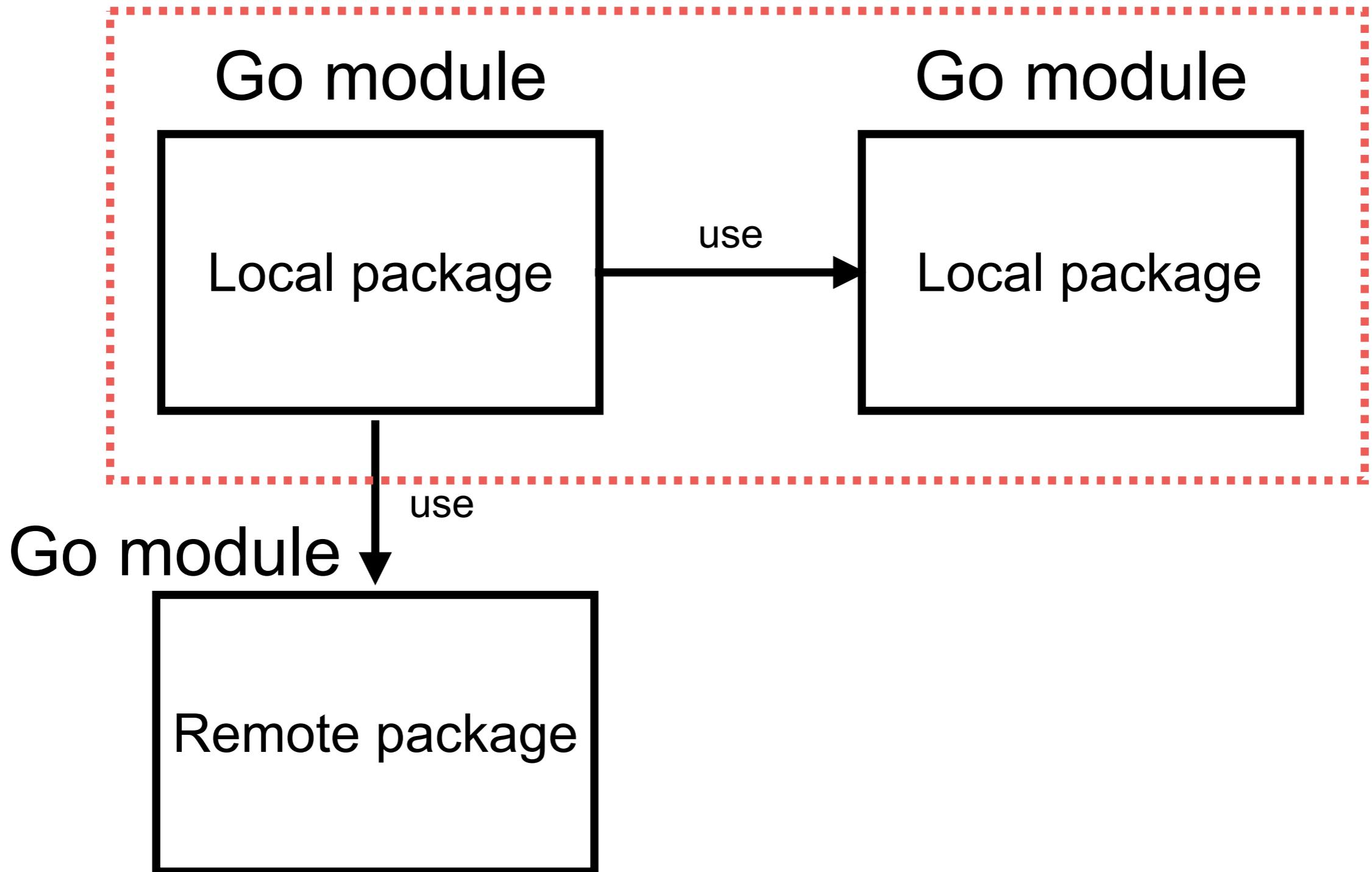
Go Workspace



<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>



Formatting

\$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

The screenshot shows a web browser window displaying the Go Documentation Server at `localhost:6060/pkg/`. The title bar says "Go Documentation Server". A search bar is at the top right. On the left, there's a sidebar with links: "Standard library", "Other packages", "Sub-repositories", and "Community". On the right, there's a cartoon character of a penguin wearing a hard hat. The main content area has a blue header "Standard library ▾". Below it is a table:

Name	Synopsis
archive	
tar	Package tar implements access to tar archives.
zip	Package zip provides support for reading and writing ZIP archives.
arena	The arena package provides the ability to allocate memory for a collection of Go values and free that space manually all at once, safely.
bufio	Package bufio implements buffered I/O. It wraps an io.Reader or io.Writer object, creating another object (Reader or Writer) that also implements the interface but provides buffering and some help for textual I/O.
builtin	Package builtin provides documentation for Go's predeclared identifiers.
bytes	Package bytes implements functions for the manipulation of byte slices.
compress	

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



Linting and Static Analysis

GolangCI Lint



<https://github.com/golangci/golangci-lint>



Security Scanning

GitLeaks

SAST tool for detect and prevent code !!
Secrets, API keys and tokens

<https://github.com/gitleaks/gitleaks>



Testing with Go



Learning by test !!



RED GREEN REFACTOR

<https://quii.gitbook.io/learn-go-with-tests/>



Testing with Go

testing package

Test

Benchmark

Example

Fuzzing

<https://pkg.go.dev/testing>



Testing with Go

hello.go

```
package main

import "fmt"

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

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

hello_test.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
```



Testing

\$go test ./...

\$go test ./... -count=1



https://pkg.go.dev/cmd/go#hdr-Testing_flags



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

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
}
```



Benchmark

```
$go test --bench .
```

```
$go test --bench . -count 5
```

<https://pkg.go.dev/testing#hdr-Benchmarks>



Fuzzing

\$go test -fuzz .

<https://go.dev/doc/tutorial/fuzz>



Essential features



Essential Features

Struct and Pointer

Interface

Error

Defer, panic, recover

Function as a parameter

Go routine and channel



No private/public in Go !!



Data = public or exported
data = private or unexported



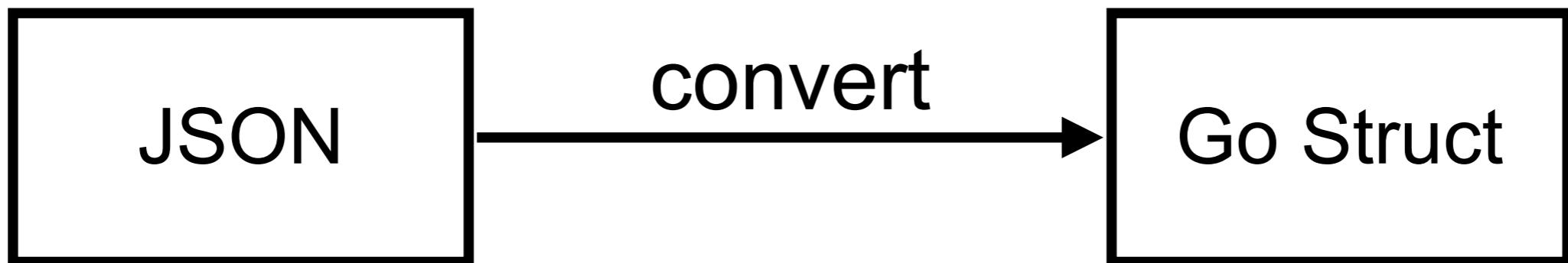
Struct and Pointer

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



Paste JSON as Code v12.0.46

quicktype | ⚡ 1,684,428 | ★★★★★ (44)

Copy JSON, paste as Go, TypeScript, C#, C++ and more.

[Disable](#) | ▾

[Uninstall](#) | ▾



This extension is enabled globally.

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



Go workshop

© 2022 - 2025 Siam Chamnankit Company Limited. All rights reserved.

Convert JSON to Go Struct

The screenshot shows the quicktype web application interface. On the left, there's a sidebar with a 'Name' input field containing 'Welcome' and a 'Source type' dropdown set to 'JSON'. Below these are some sample JSON code snippets. The main area displays the generated Go code:

```
// This file was generated from JSON Schema using quicktype, do not modify it directly.  
// To parse and unparse this JSON data, add this code to your project:  
//  
// welcome, err := UnmarshalWelcome(bytes)  
// bytes, err = welcome.Marshal()  
  
package main  
  
import "encoding/json"  
  
func UnmarshalWelcome(data []byte) (Welcome, error) {  
    var r Welcome  
    err := json.Unmarshal(data, &r)  
    return r, err  
}  
  
func (r *Welcome) Marshal() ([]byte, error) {  
    return json.Marshal(r)  
}  
  
type Welcome struct {  
    Greeting string `json:"greeting"  
    Instructions []string `json:"instructions"  
}
```

On the right, there are settings for the generated package: 'Language' is set to 'Go', 'Generated package name' is 'main', and several optional checkboxes are available: 'Plain types only', 'Renders each top-level object in its own Go file', 'Plain types with package only', and 'Make all properties optional'. A 'Copy Code' button is also present.

<https://app.quicktype.io/>



Sequence of fields in struct

```
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

```
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

Banking

+ balance

Balance():int

Deposit(int)

Withdraw(int)



Workshop with Struct

```
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

```
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

```
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

```
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

```
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

```
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

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

```
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
```



Improve your test



Improve your test

Use sub-test
Refactor assertion



Improve your test

```
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

```
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>



Go workshop

© 2022 - 2025 Siam Chamnkit Company Limited. All rights reserved.

Add new feature with test ?



Withdraw :: not enough money

```
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

```
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

```
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

```
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

```
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>



Error handling with Go

<https://go.dev/blog/error-handling-and-go>



Build-in error type

```
func doSth() (result string, err error) {  
    return "", fmt.Errorf("Error")  
}
```

```
func main() {  
    result, err := doSth()  
    if err != nil {  
        fmt.Println(err)  
    }  
    fmt.Println("Result=", result)  
}
```



Error handling

```
func doSth() (result string, err error) {  
    return "", fmt.Errorf("Error")  
}
```

```
func main() {  
    result, err := doSth()  
    if err != nil {  
        fmt.Println(err)  
    }  
    fmt.Println("Result=", result)  
}
```



Error interface

```
type error interface {
    Error() string
}
```

```
type errorString struct {
    s string
}

func (e *errorString) Error() string {
    return e.s
}
```



Custom error

```
type error interface {
    Error() string
}
```

```
type errorString struct {
    s string
}

func (e *errorString) Error() string {
    return e.s
}
```



Errors package

Provide function to manipulate errors
Use New() to create error

```
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://pkg.go.dev/errors>



Best practices for Error handling

Use `errors.New()` to create error

Wrap error with `fmt.Errorf()`

Return `nil` if everything is ok

Create **custom type** for error

Don't use `panic()` and `recover()`

Check error immediately after call function

Log error before return

<https://climbtheladder.com/10-golang-error-handling-best-practices/>



Example with error

```
func main() {
    f, err := os.Open("filename.ext")
    if err != nil {
        log.Fatal(err)
    }
    defer f.Close()

    // do something with the open file

}
```

<https://climbtheladder.com/10-golang-error-handling-best-practices/>



Working with multiple errors

```
func step1() error {
    return errors.New("step1 error")
}

func step2() error {
    return errors.New("step2 error")
}
```

```
func main() {

    err1 := step1()
    err2 := step2()
    err := errors.Join(err1, err2)

    if err != nil {
        fmt.Printf("some error: \n%v", err)
    }
}
```

<https://go.dev/play/p/fKVMnn9NFEK>

<https://github.com/hashicorp/go-multierror>



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>



Defer, Panic and Recover

try/catch/finally

<https://go.dev/blog/defer-panic-and-recover>



Defer, Panic and Recover

Defer => put function call into a stack

Panic => Stop the normal execution flow

Recover => Handle panic, use with defer



```
func A() {
    defer fmt.Println("Called A")
    B()
}

func B() {
    defer fmt.Println("Called B")
    C()
}

func C() {
    defer fmt.Println("Called C")
    Break()
}
```

```
func Break() {
    defer fmt.Println("Called Break")
    panic(errors.New("My Error with panic"))

}

func main() {
    A()
}
```



Working with Recover()

```
func A() {
    defer fmt.Println("Called A")

    defer func() {
        if x := recover(); x != nil {
            fmt.Printf("Panic: %+v\n", x)
        }
    }()
}

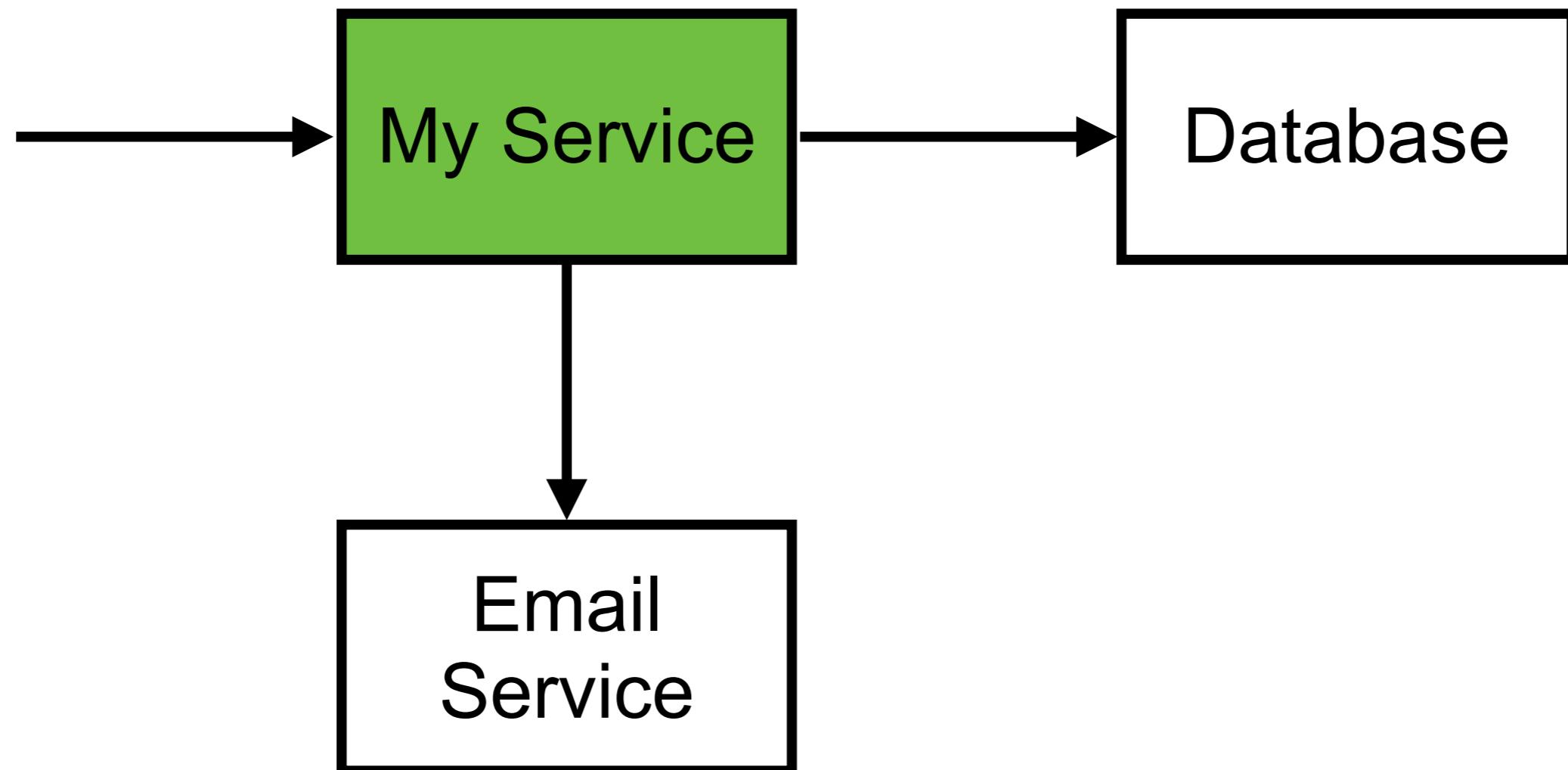
B()
}
```



Project structure with Testable !!



Structure of project



Let's Start

```
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

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

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

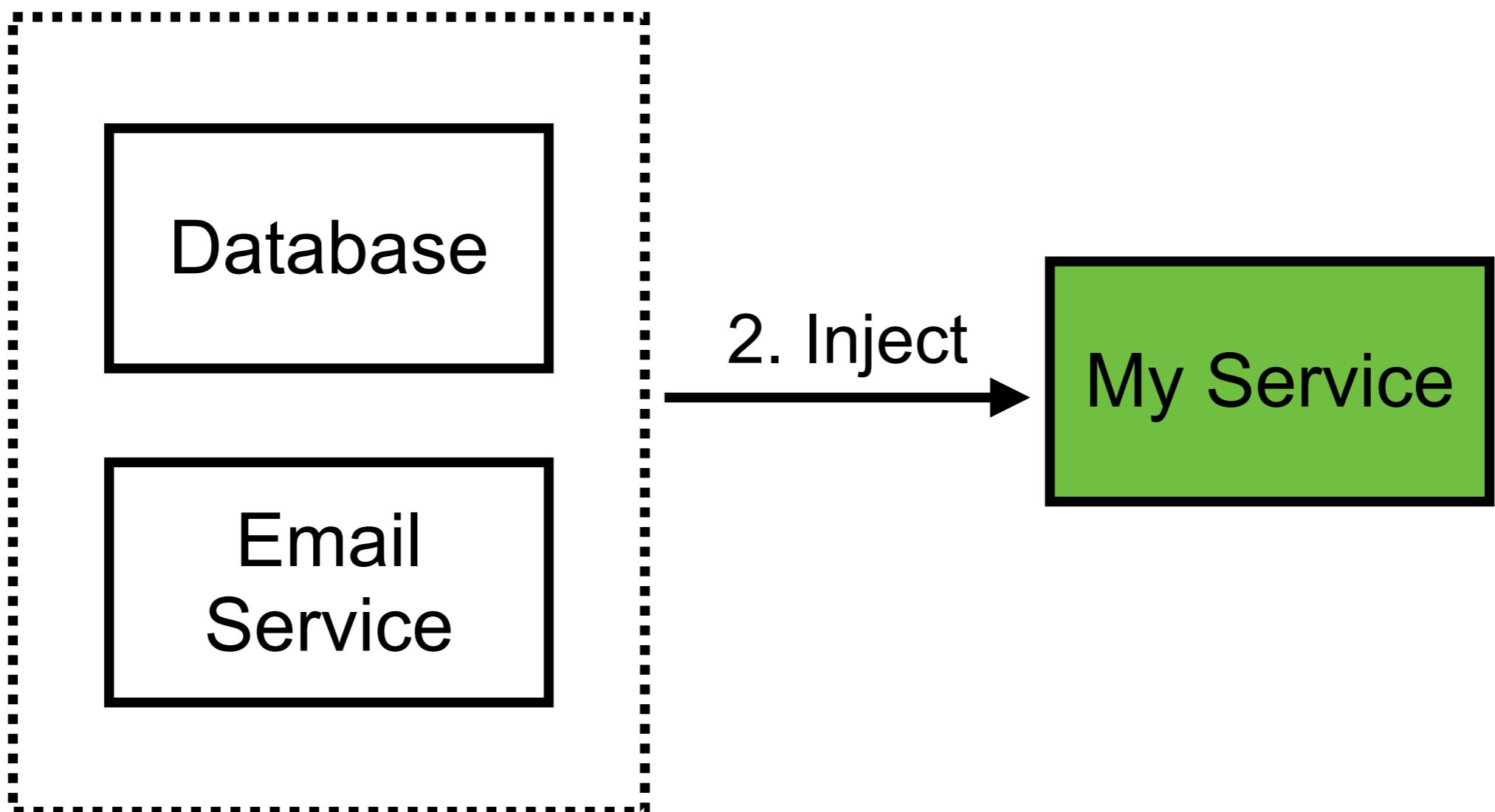


How to create database and email service ?



Use dependency injection

1. Create object of dependencies



Use dependency injection

```
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 MyService with dependencies

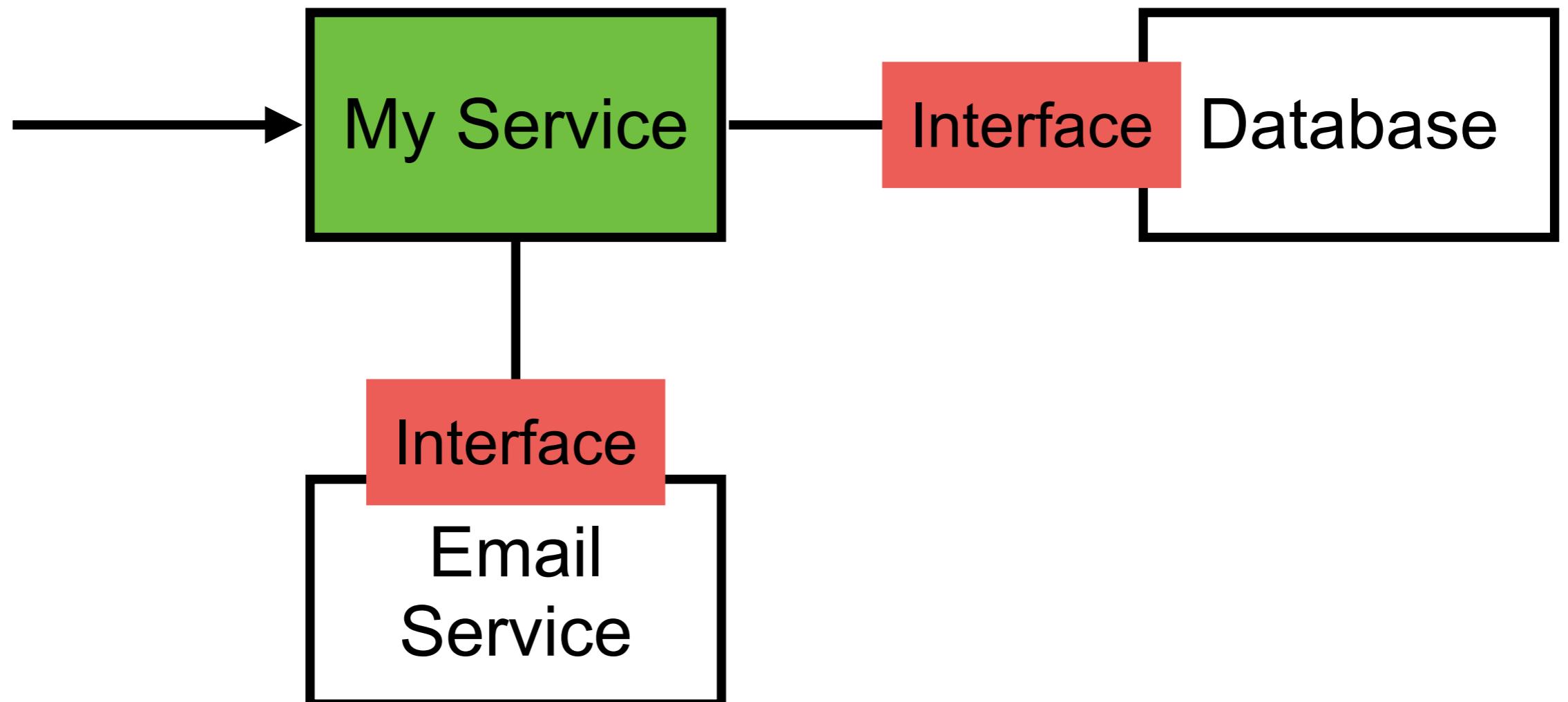
```
func NewMyService(db *Database, email *EmailService) *MyService
{
    return &MyService{
        Db:    db,
        Email: email,
    }
}
```



**How to test ?
Easy to test ?
Testable code ?**



Working with interface ?



Working with interface ?

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

```
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

```
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

```
func TestDI(t *testing.T) {  
  
    db := &MockDatabase{}  
    email := &MockEmailService{}  
    service := NewMyService(db, email)  
    service.Process()  
  
}
```

```
$go test -v
```



Common mistakes with Interface

Create too many interfaces (pollution)

Don't design with interfaces, discover them

Create too many methods

Don't write behavior-driven interface

Create interface on the producer side

Create interface for testing



Go Proverbs

Simple, Poetic, Pithy

Don't communicate by sharing memory, share memory by communicating.

Concurrency is not parallelism.

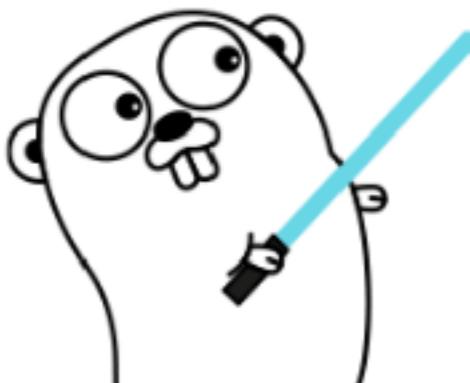
Channels orchestrate; mutexes serialize.

The bigger the interface, the weaker the abstraction.

Make the zero value useful.

interface{} says nothing.

Gofmt's style is no one's favorite, yet gofmt is everyone's favorite.



A little copying is better than a little dependency.

Syscall must always be guarded with build tags.

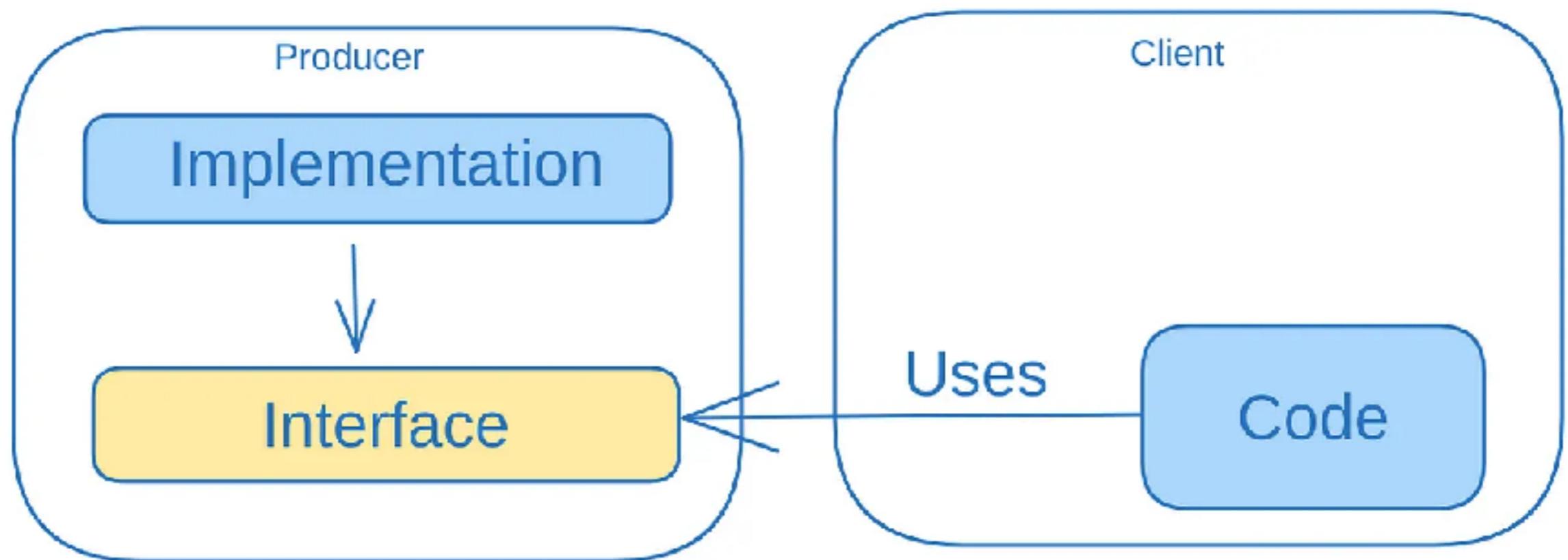
Cgo must always be guarded with build tags.

Cgo is not Go.

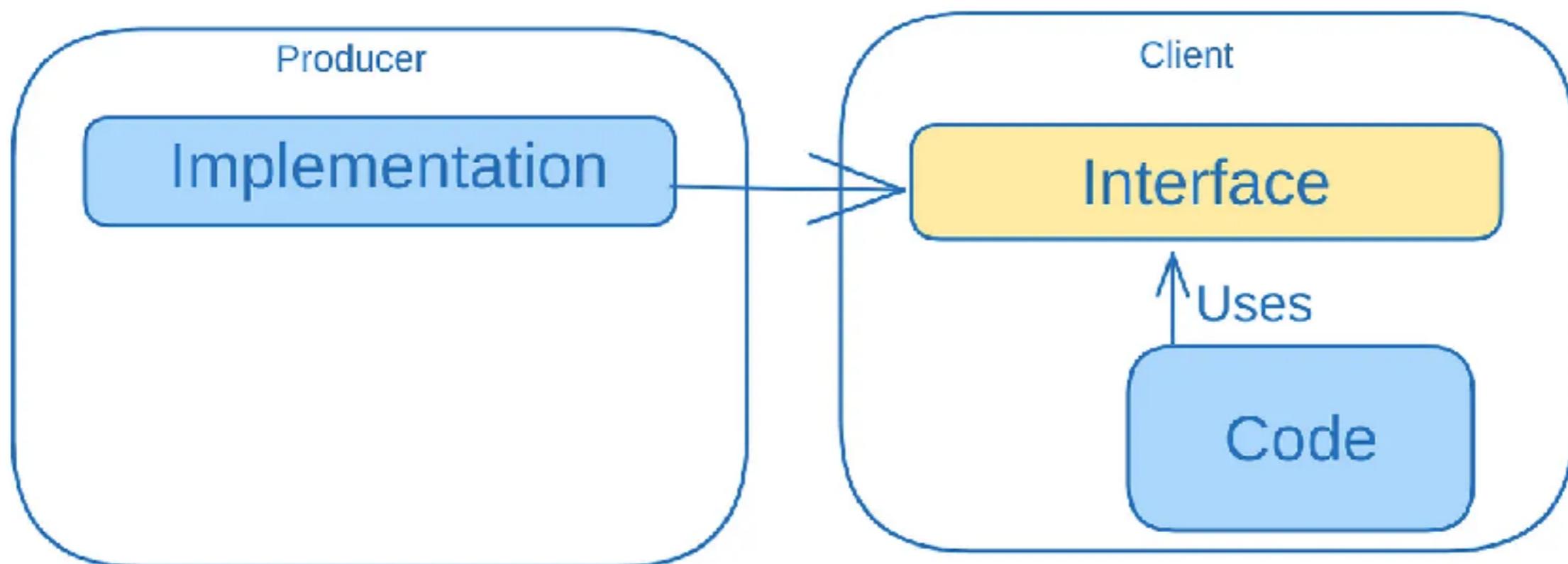
<https://dave.cheney.net/2016/04/27/dont-just-check-errors-handle-them-gracefully>



Create interface on producer side



Create interface on consumer side



Better interface !!

```
type UsersRepository interface {  
    GetAllUsers()  
    GetUser(id string)  
}  
  
type UserRepository struct {  
}  
  
func (UserRepository) GetAllUsers() {}  
func (UserRepository) GetUser(id string) {}
```

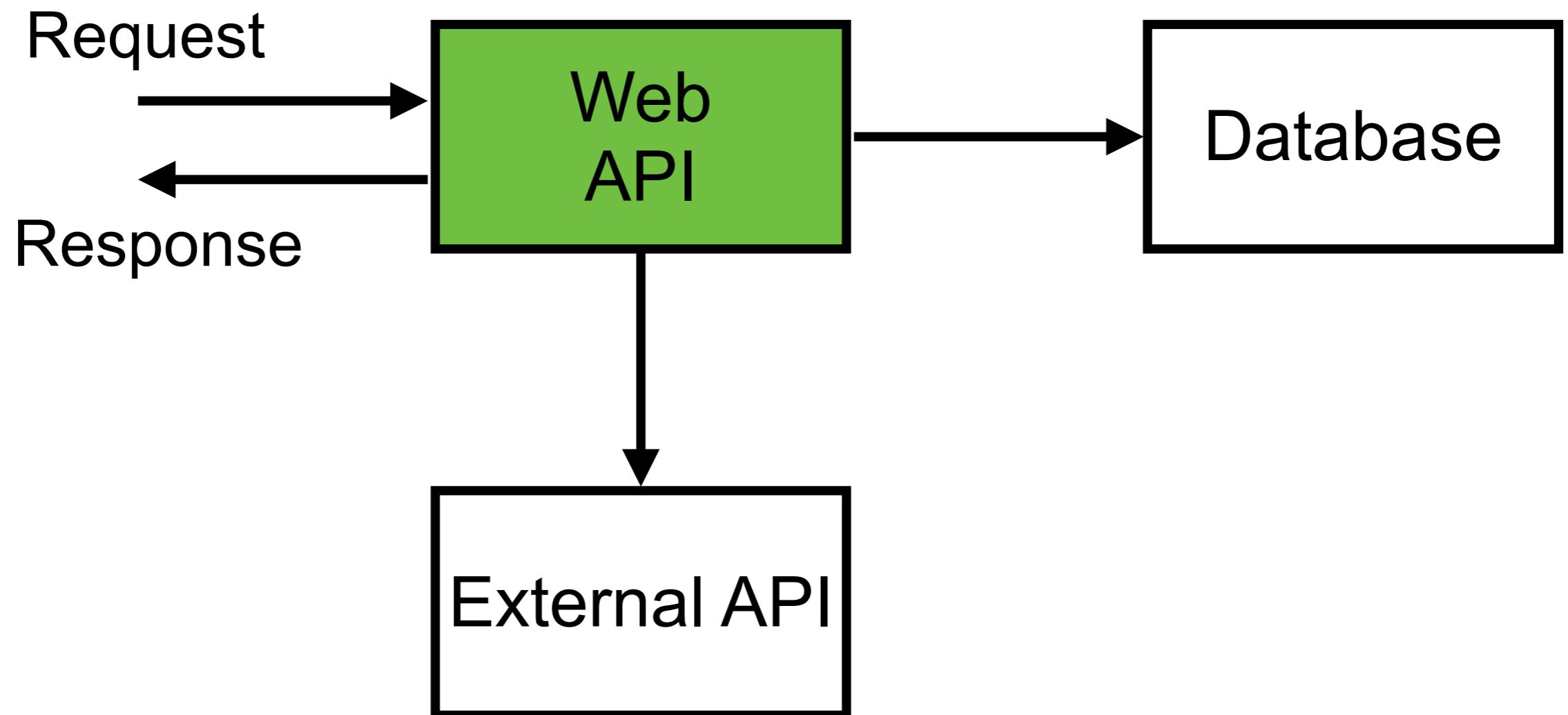
```
type UserGetter interface {  
    GetUser(id string)  
}
```



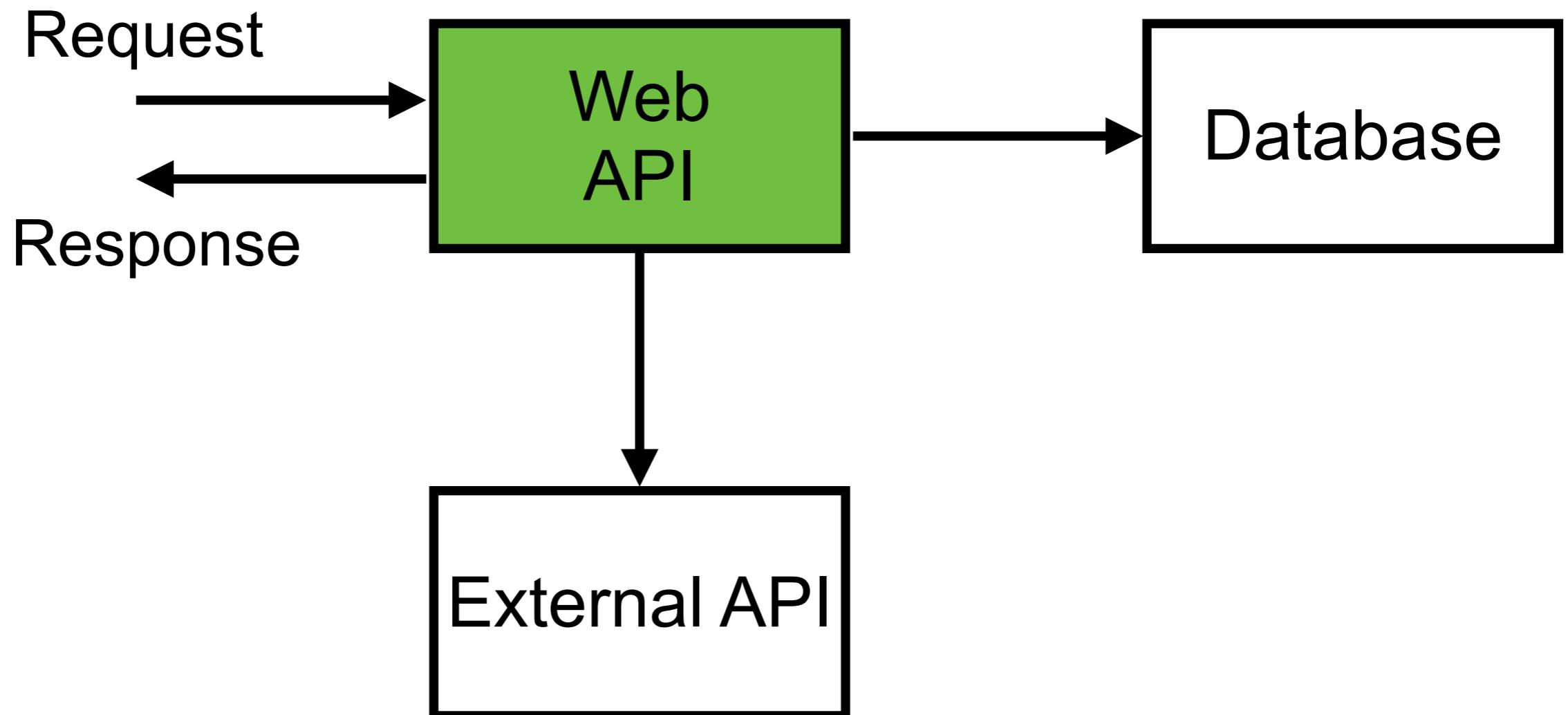
Test Strategies



Architecture



What to Test ?



Component and Contract Testing

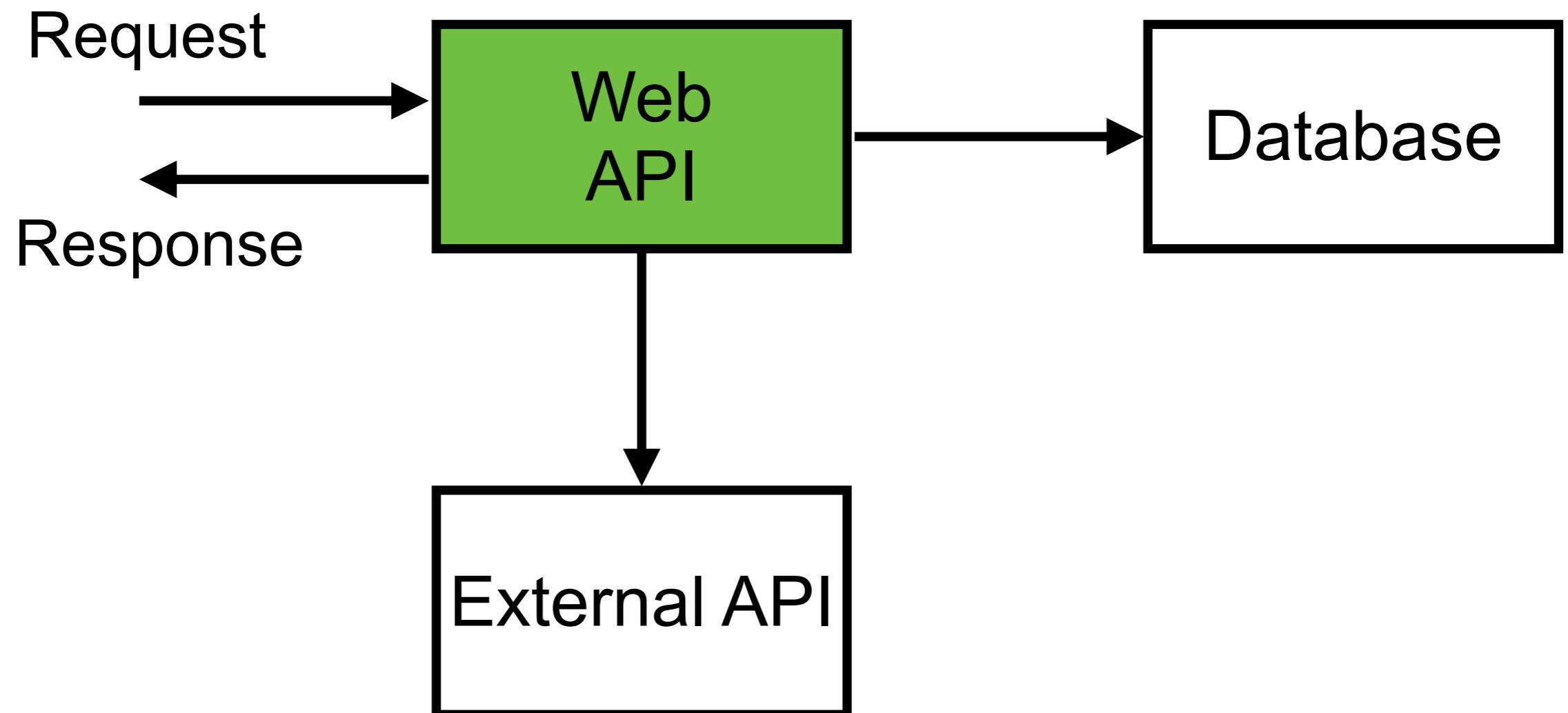
<https://github.com/up1/course-contract-testing>



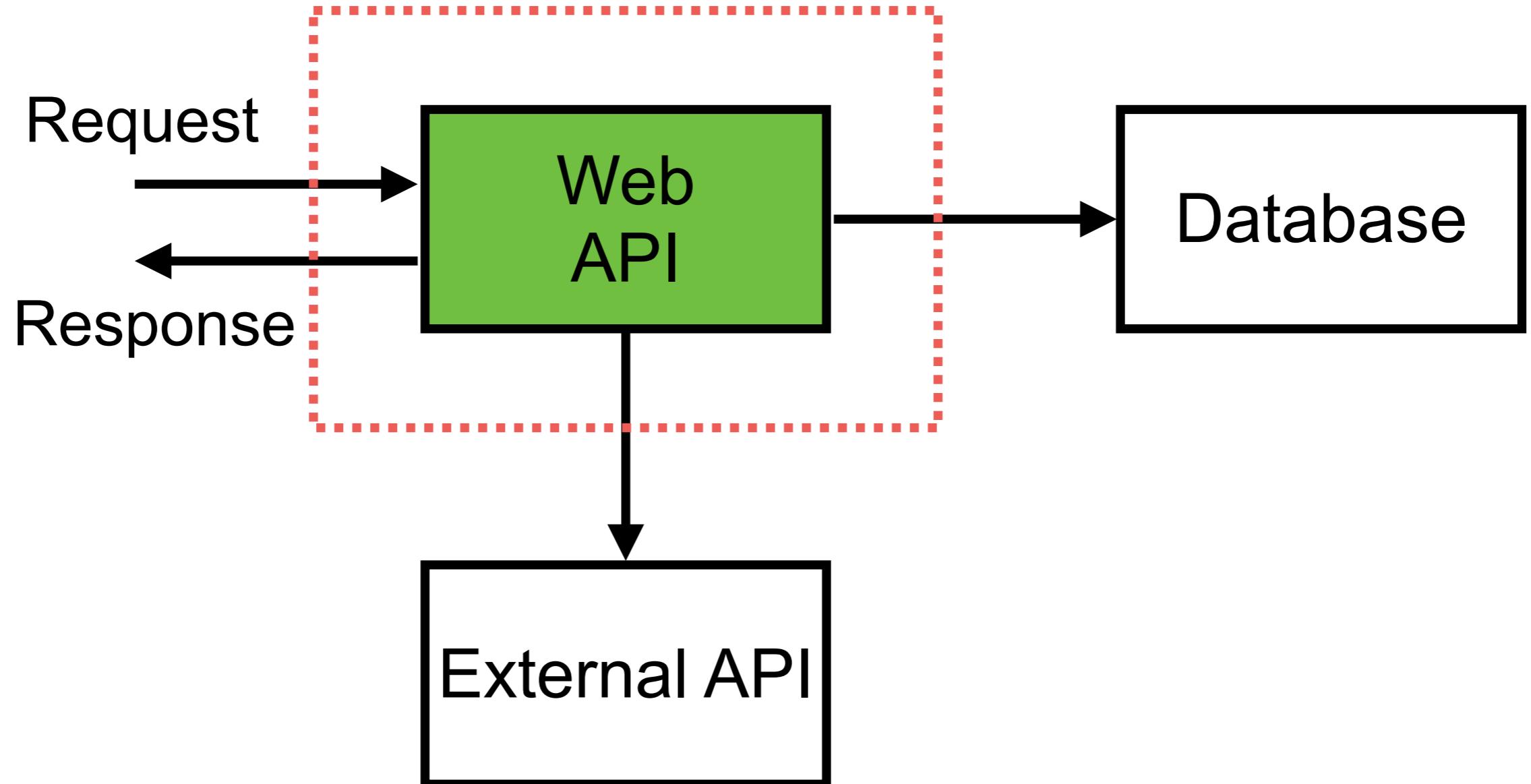
Component Testing



Component Testing



Component Testing

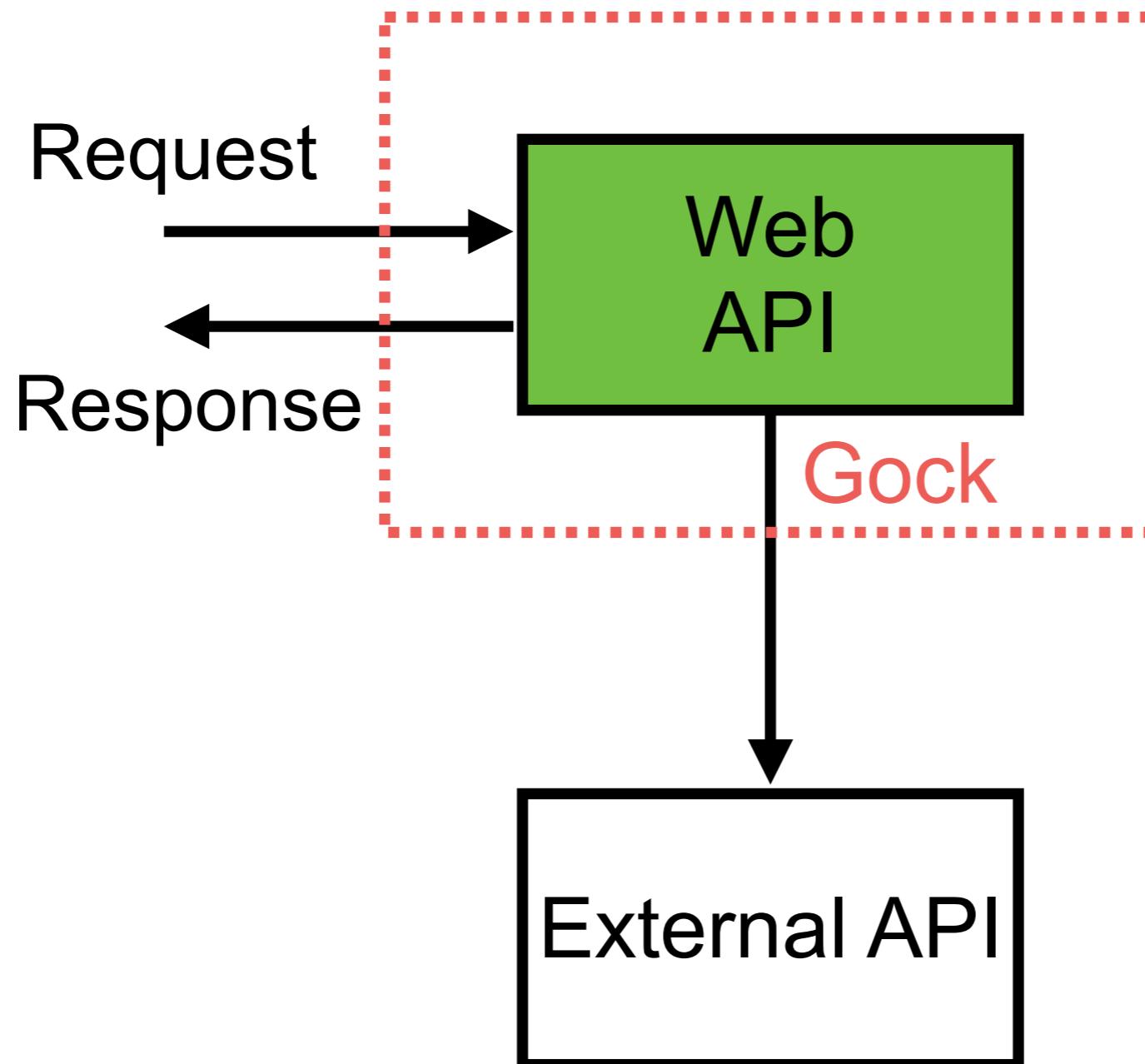


Component testing

In-process
Separate process



In-process with Gock



<https://github.com/h2non/gock>



Try to test with Gock

```
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
```

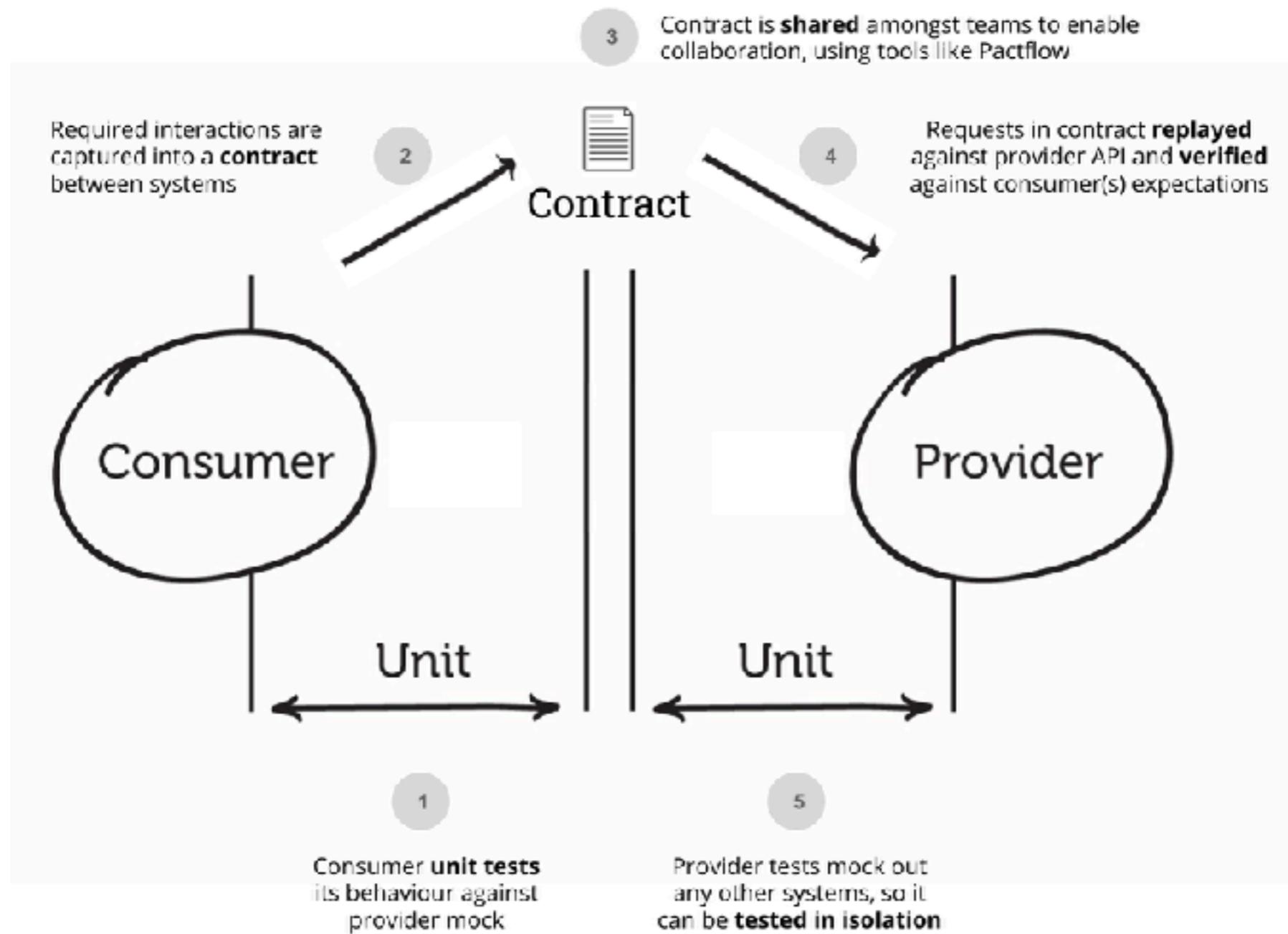


Contract Testing

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



Contract Testing



<https://docs.pact.io/>



Go workshop

© 2022 - 2025 Siam Chamnkit Company Limited. All rights reserved.

Pact Broker

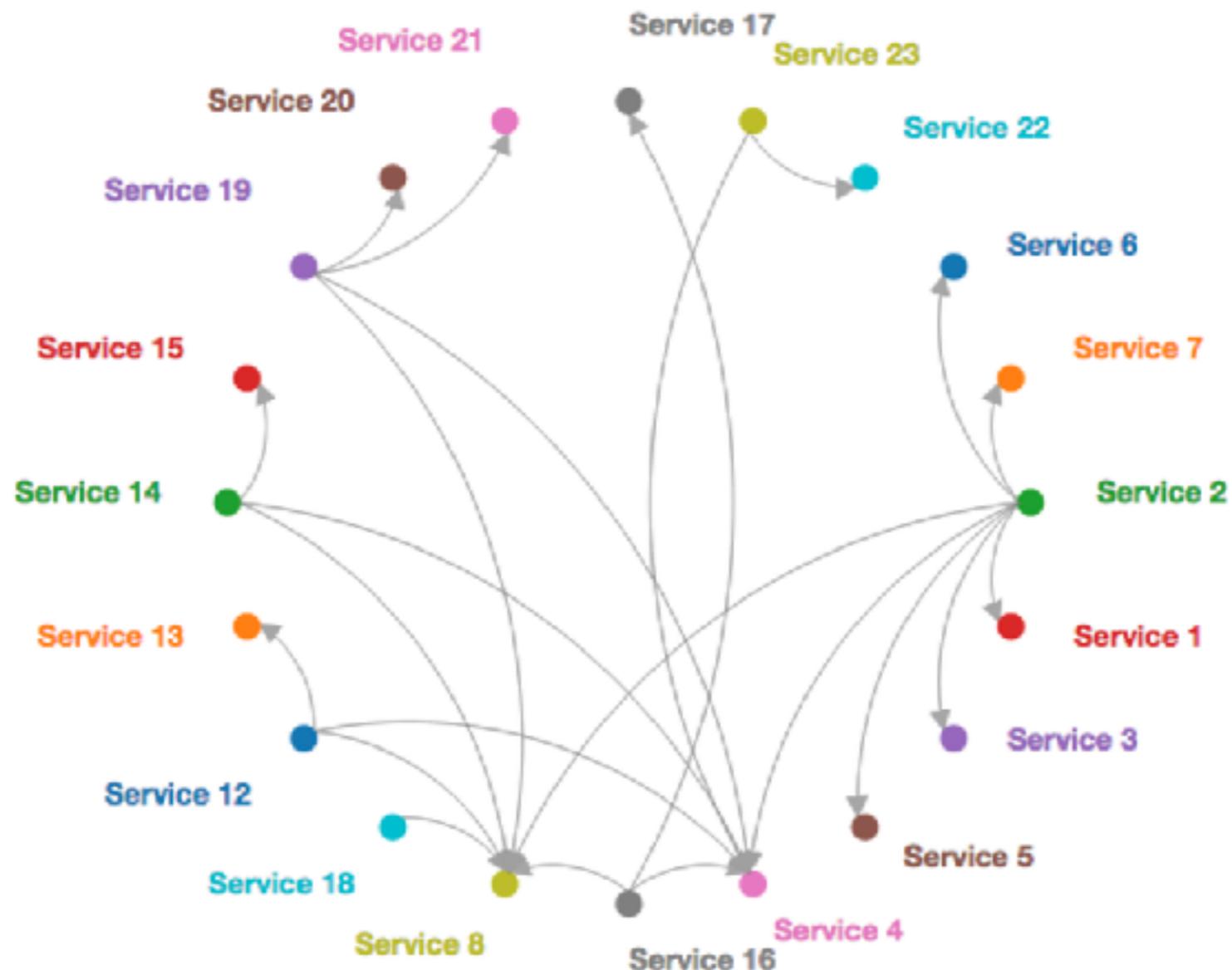
Pacts

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/>



Go workshop

© 2022 - 2025 Siam Chamnkit Company Limited. All rights reserved.

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

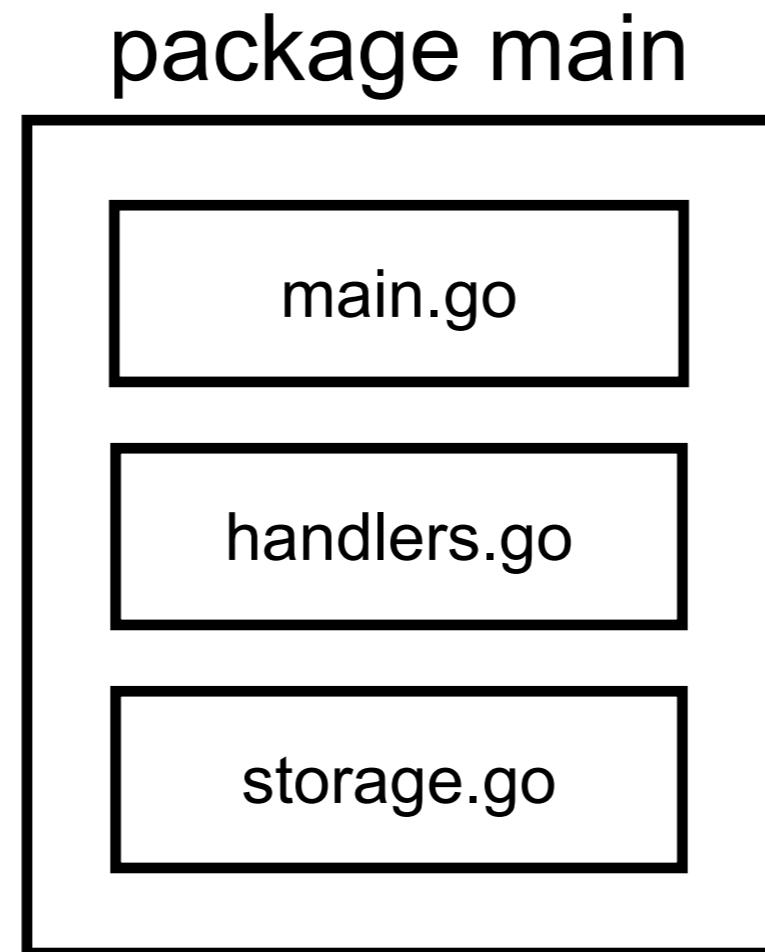
Clean architecture

Hexagonal architecture



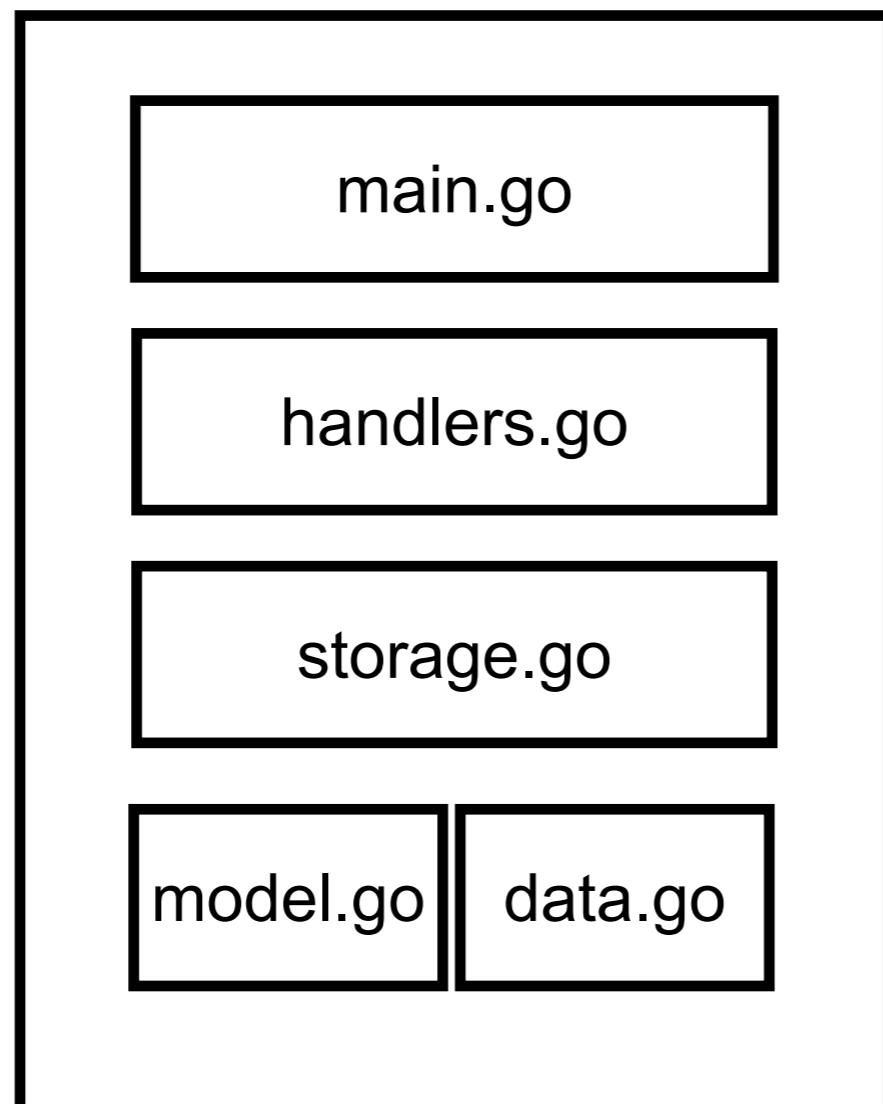
Flat structure

Easy to start
Only main package



Beer !!

package main

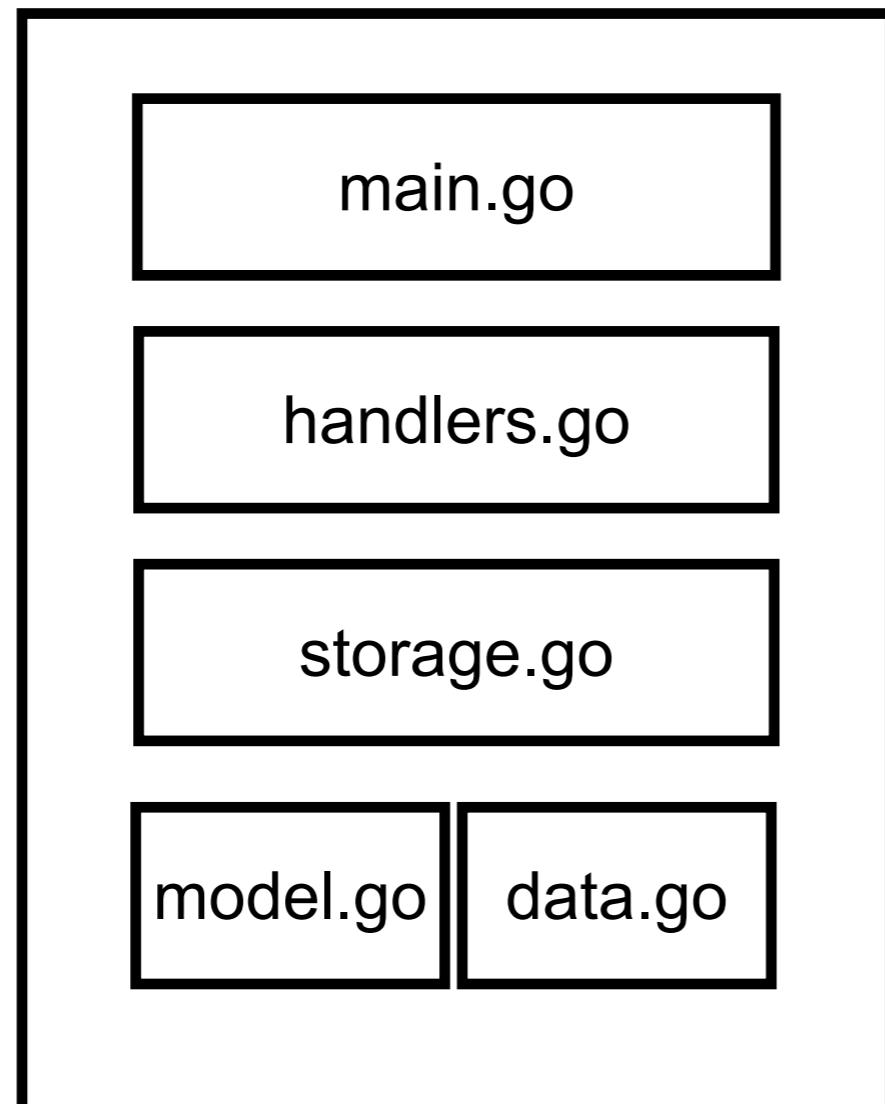


Create Storage as global variable
Start Web server



Beer !!

package main



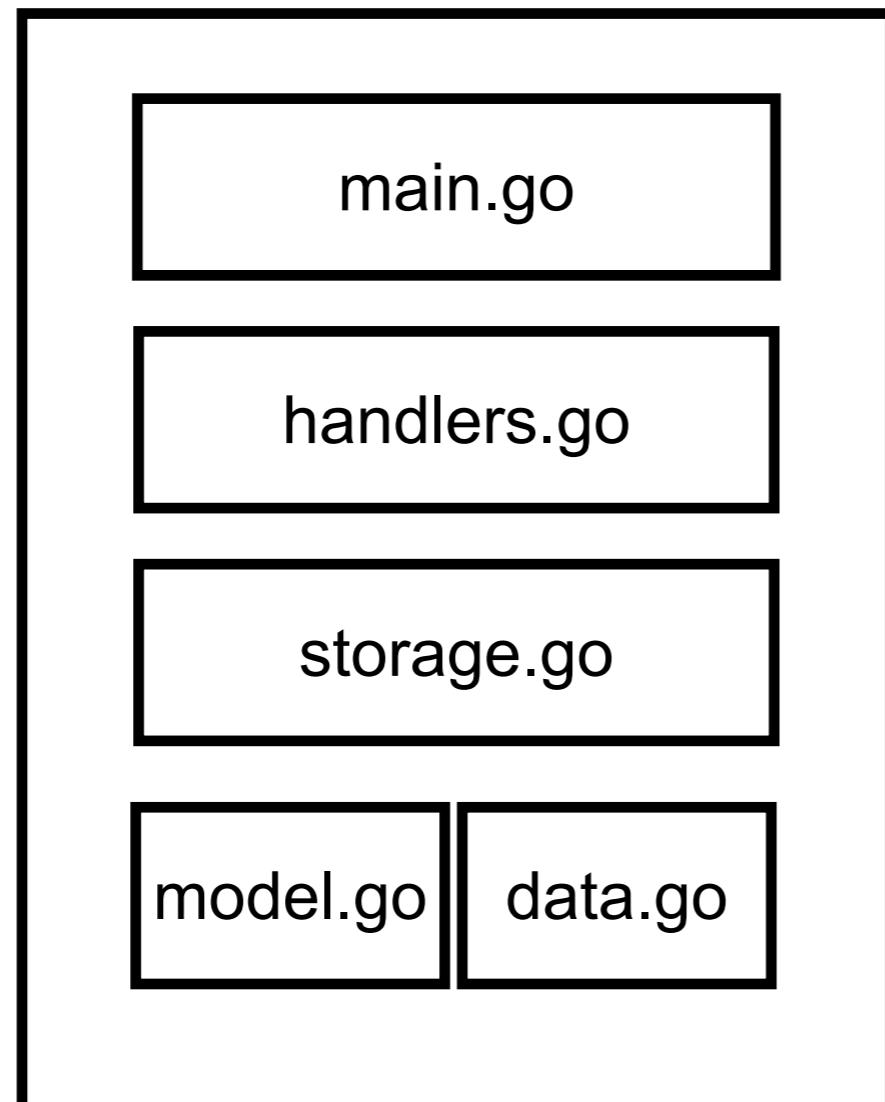
Create Storage as global variable
Start Web server

GET /beers
POST /beers



Beer !!

package main



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)



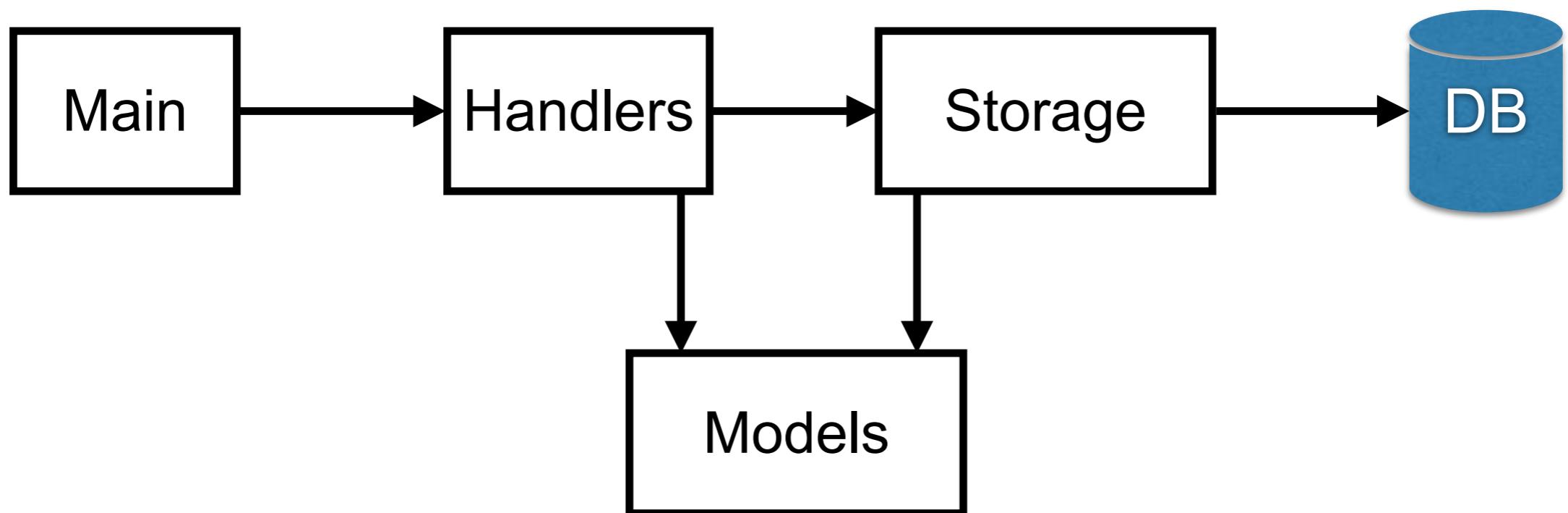
Layer

Grouping by function



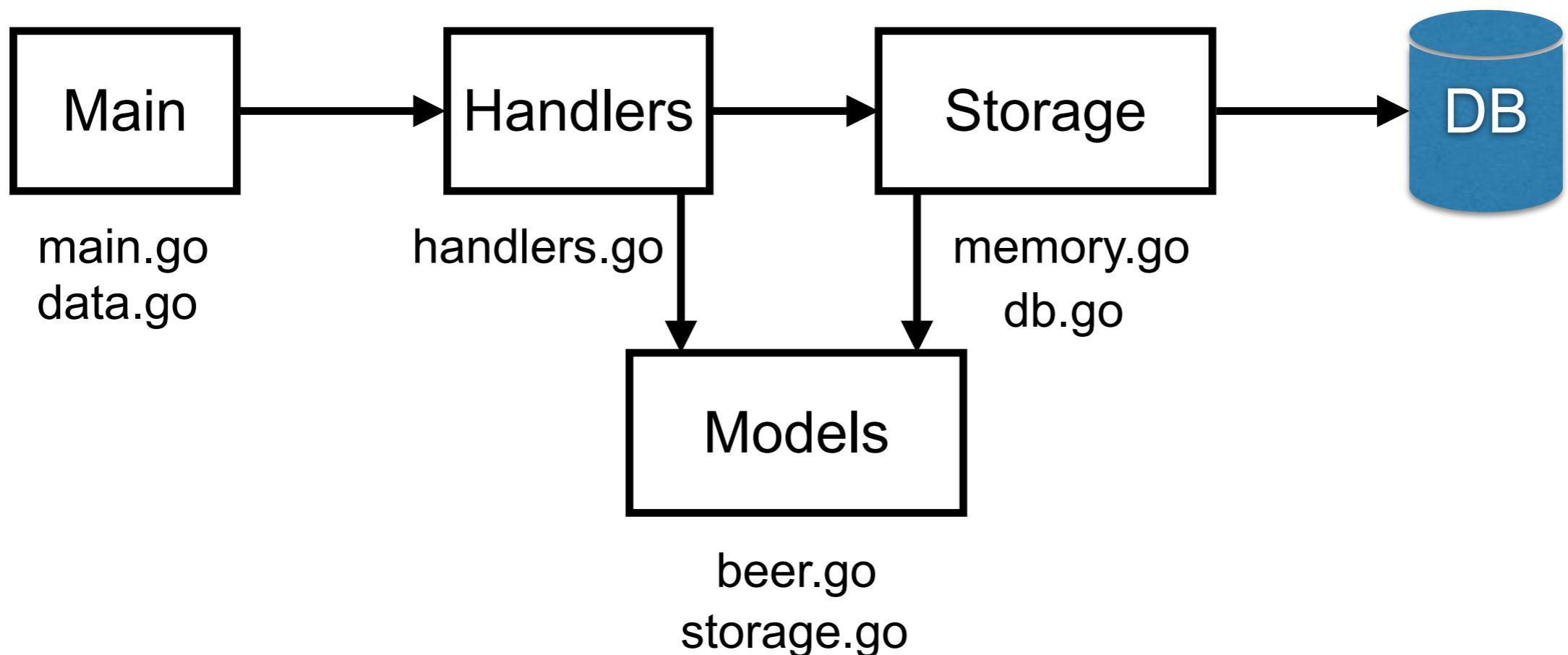
Layer

Grouping by function



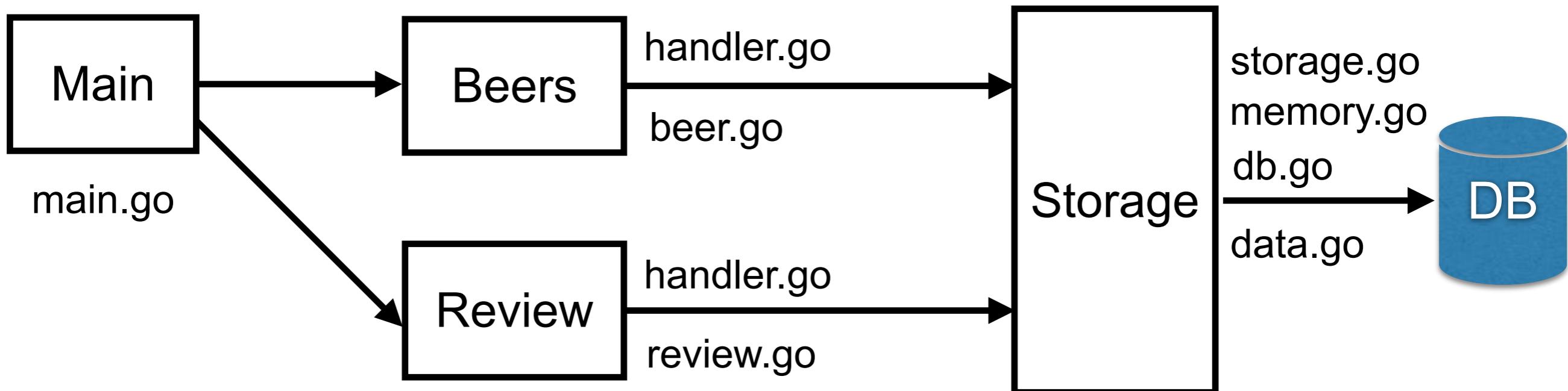
Layer

Grouping by function



Modular

Grouping by module



Manage Dependency



Manage Dependency ?



Manage Dependency ?

Global variable



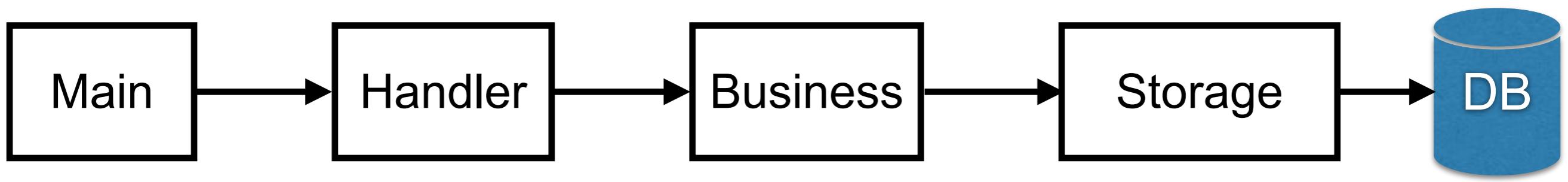
```
// Global variable
var db Storage

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



Manage Dependency ?

Dependency with Struct



```
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

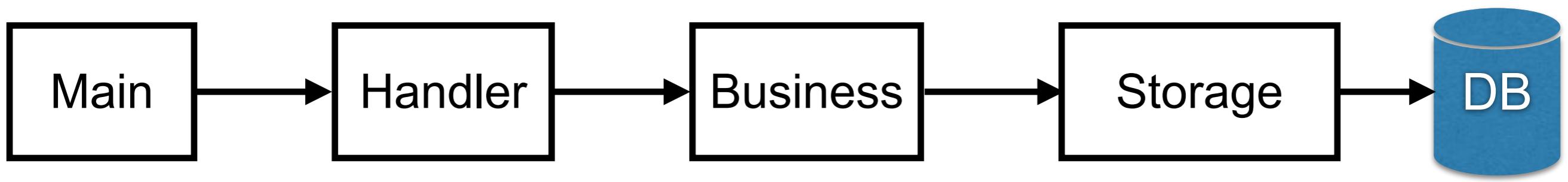
```
func (config *MyConfig) GetBeers(c *gin.Context) {
    beers, err := config.Db.FindBeers()
    if err != nil {
        c.AbortWithStatus(http.StatusBadRequest)
        return
    }
    c.JSON(http.StatusOK, beers)
}
```

2. Add receiver



Manage Dependency ?

Dependency with closure



```

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

```

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)
    }
}

```

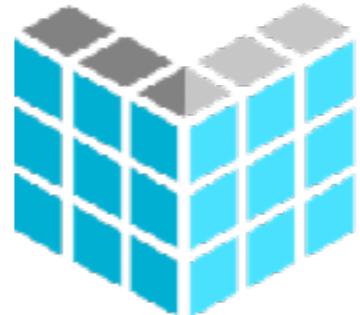
2. Use closure



Let's Go



RESTful API with Go

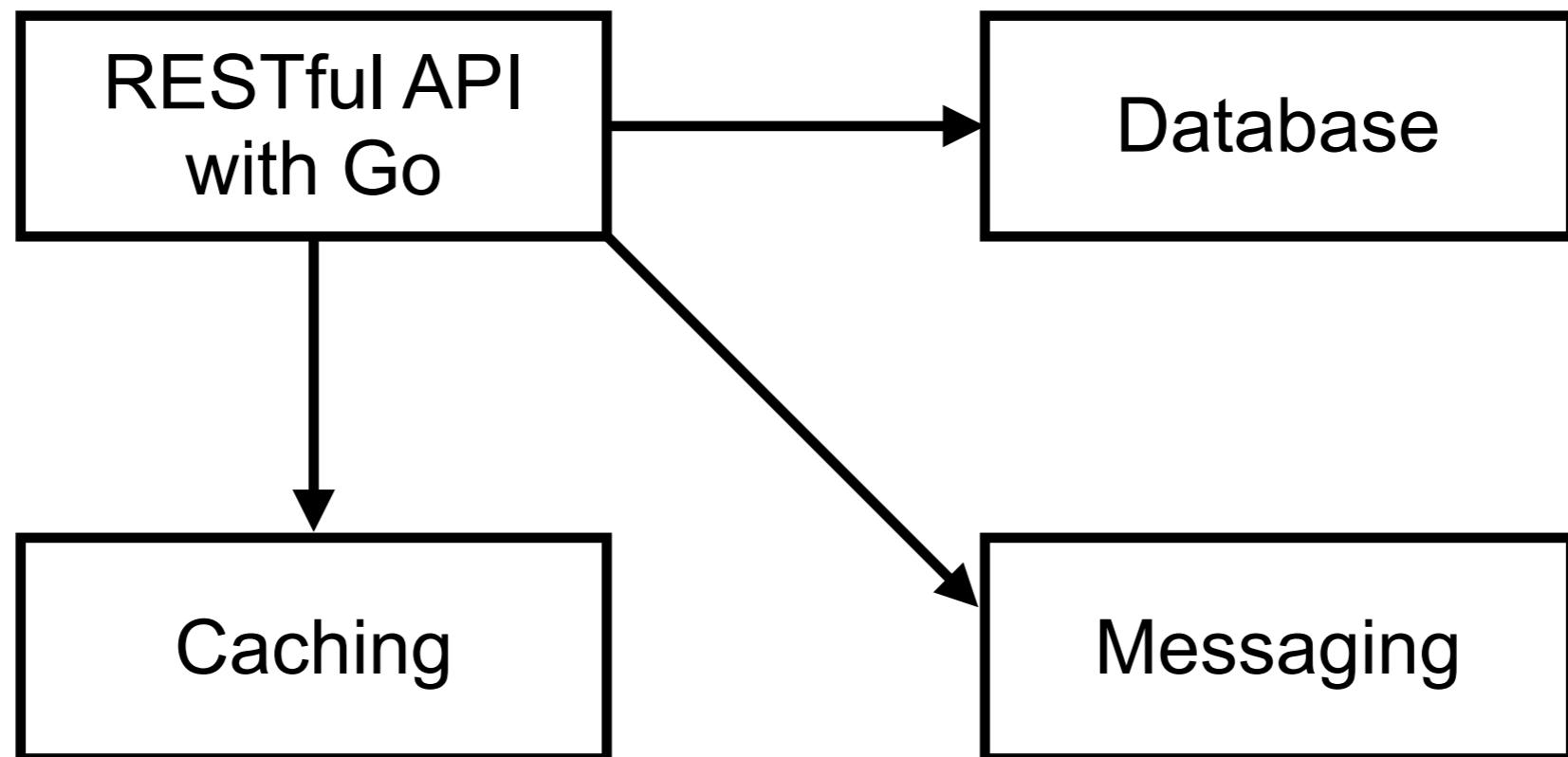


Workshop

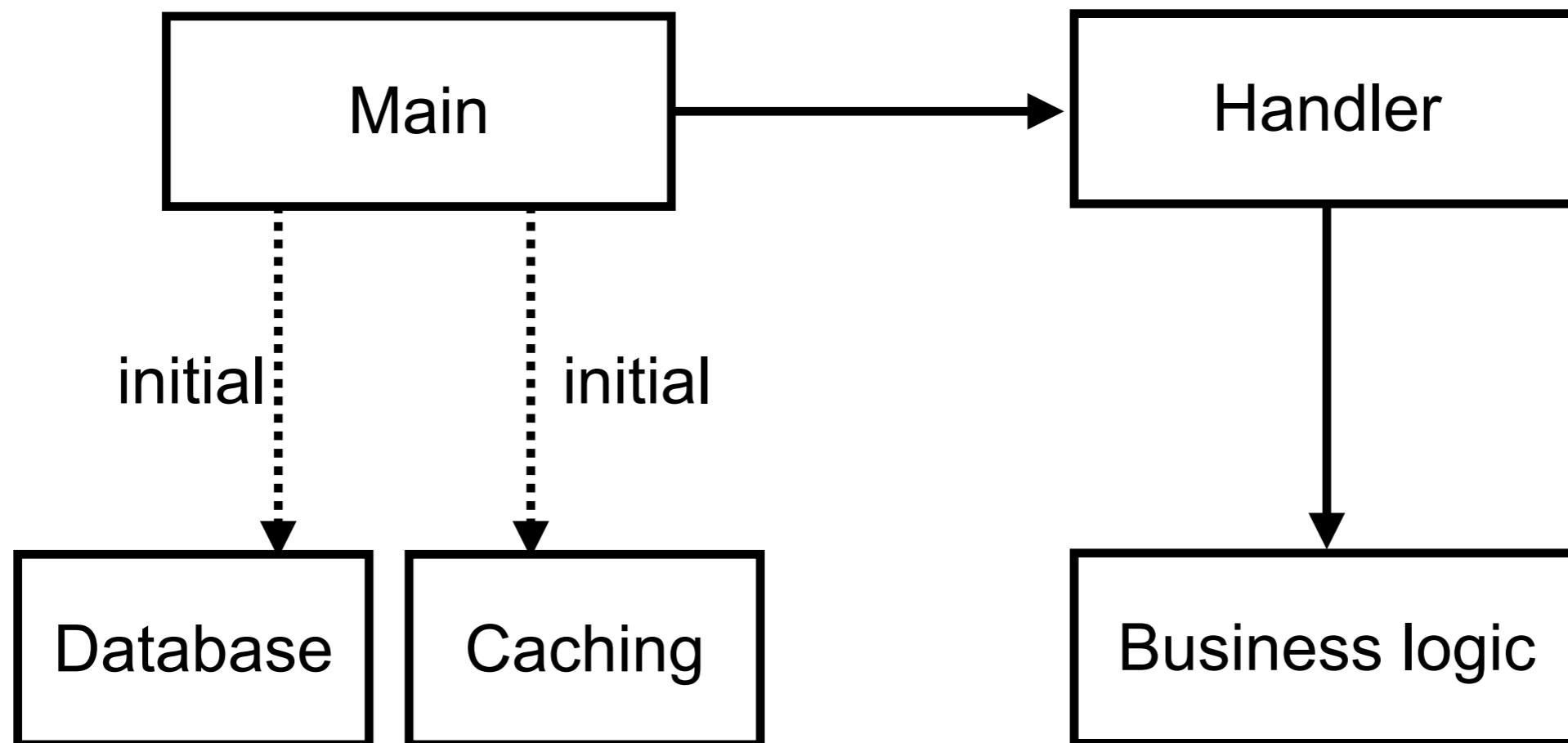
<https://github.com/up1/course-go-2024/wiki/REST-API>



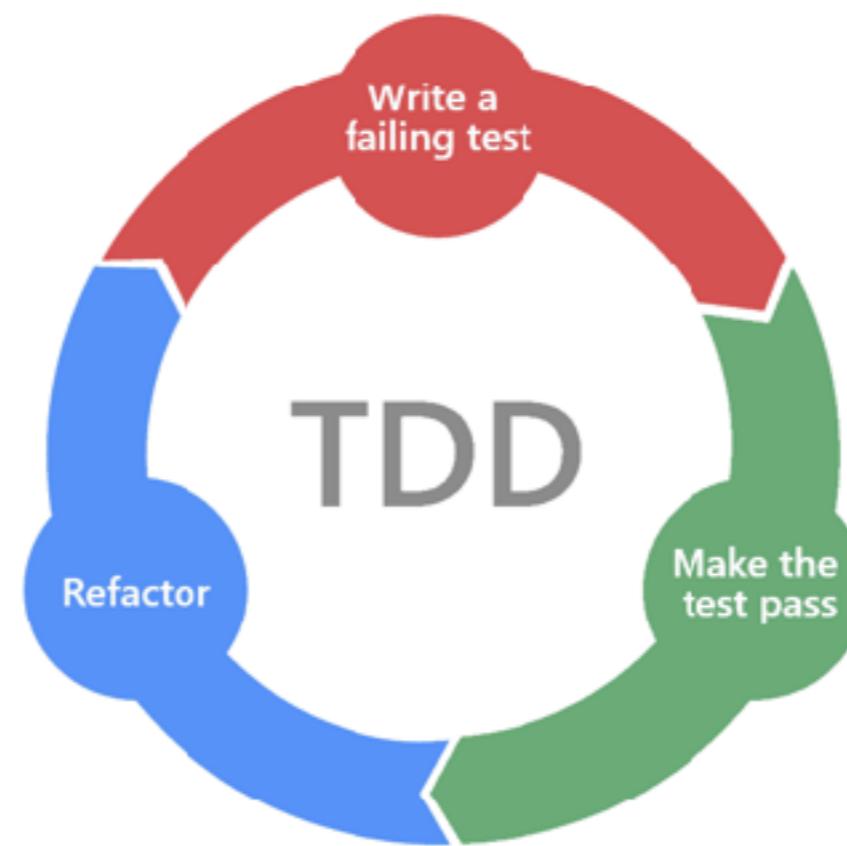
Architecture



Structure for Go project ?



Try to refactor with safety

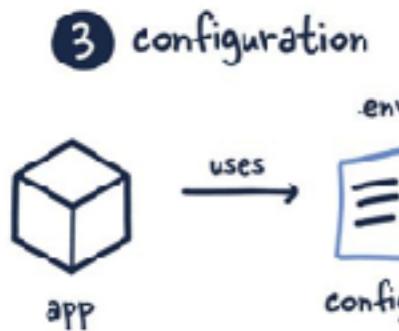
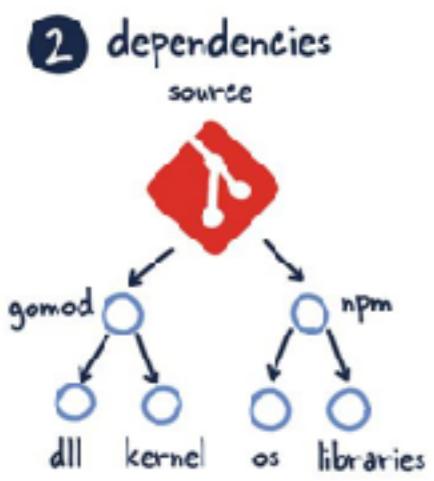
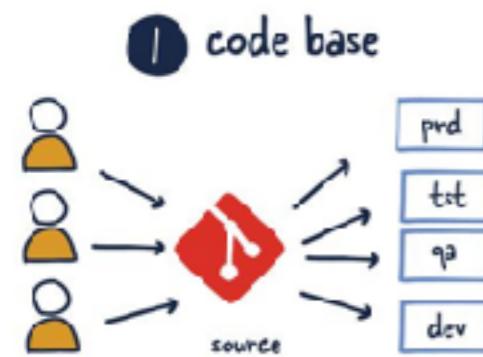


Twelve factors

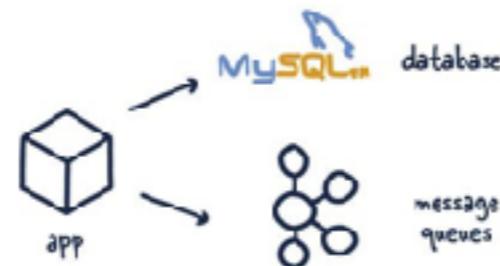
<https://12factor.net/>



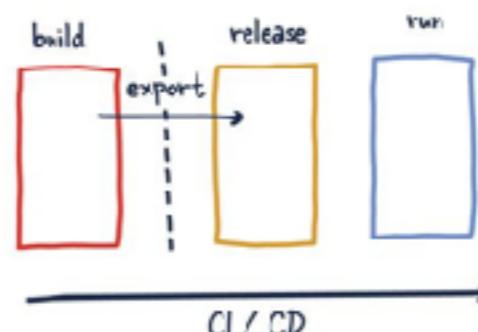
12 FACTOR APP REVISITED



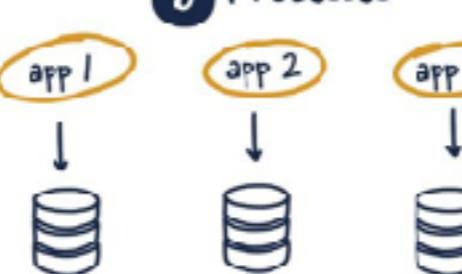
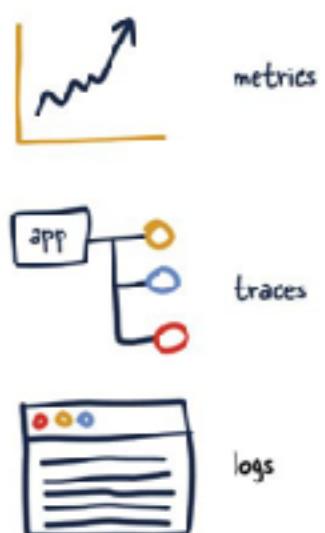
4 backing services



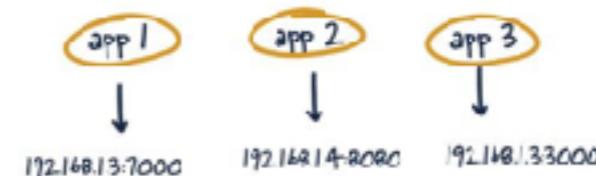
5 build, release, run



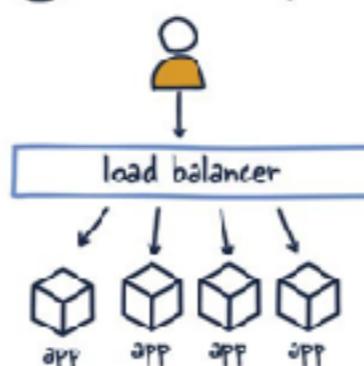
11 logs



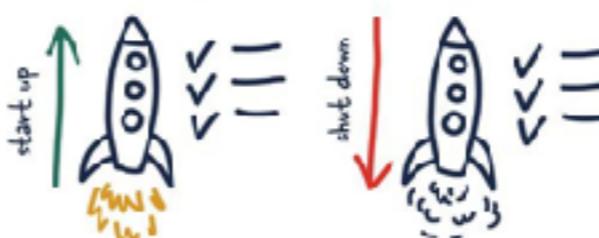
7 port binding



12 admin processes



9 disposability



<https://architecturenotes.co/p/12-factor-app-revisited>



Go workshop

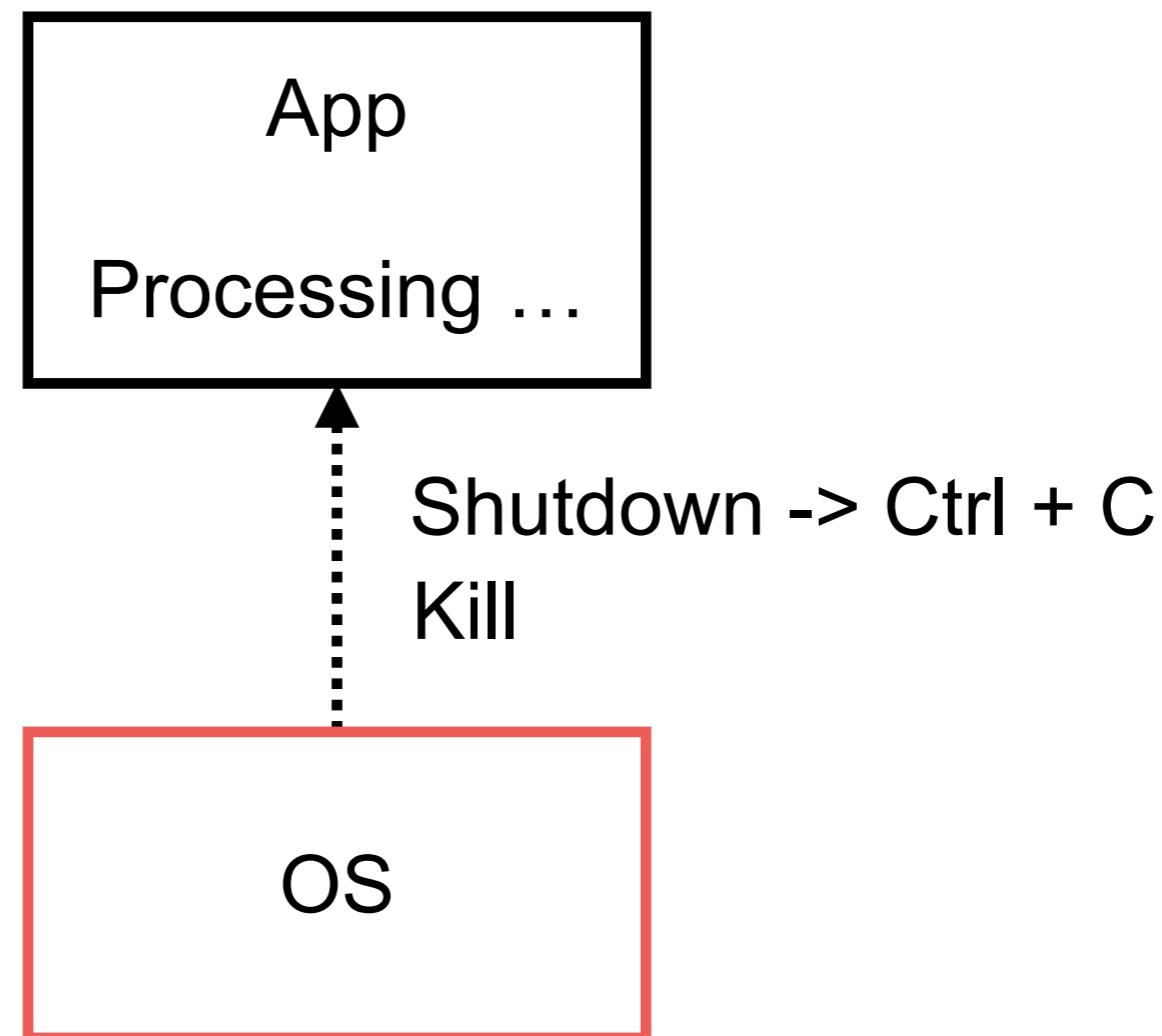
© 2022 - 2025 Siam Chamnkit Company Limited. All rights reserved.

Graceful Shutdown



Graceful Shutdown

How to make sure, existing requests are successfully responded before shutdown ?



Signals

Signal	Action	
SIGTERM	Kill command	kill <process id>
SIGINT	CTRL + C	Terminate foreground process
SIGKILL SIGSTOP		Can't handle from code



Structured Logging

<https://go.dev/blog/slog>



Log Format ?

Structured

- Usually JSON or LogFmt.
- Least human-readable.
- Easy to parse, search and filter.
- Consistent formatting.
- Faster troubleshooting due to added context.
- Automated reporting and analysis.



RECOMMENDED

Semi-structured

- Mix of structured and unstructured data.
- Human-readable.
- Formatting variability leads to inconsistency.
- Limited automation possibilities.
- Requires a more complex parsing logic.

Unstructured

- Free-form plain text messages.
- Human-readable.
- Challenging to parse.
- Requires manual interpretation.
- Suitable only for quick ad-hoc logging in development environments.



NOT RECOMMENDED

<https://betterstack.com/community/guides/logging/structured-logging/>



Go workshop

© 2022 - 2025 Siam Chamnankit Company Limited. All rights reserved.

Log Format ?

```
2024-06-17T08:55:00Z [INFO] User 'user123' logged in from 192.168.0.1
2024-06-17T08:55:01Z [ERROR] Database connection failed: Connection refused
2024-06-17T08:55:00Z [INFO] User 'user123' logged in from 192.168.0.1
2024-06-17T08:55:01Z [ERROR] Database connection failed: Connection refused
```

```
{
  "timestamp": "2024-06-17T08:55:00Z",
  "level": "INFO",
  "message": "User 'user123' logged in",
  "user": "user123",
  "ip_address": "192.168.0.1"
},
{
  "timestamp": "2024-06-17T08:55:01Z",
  "level": "ERROR",
  "message": "Database connection failed",
  "error": {
    "code": "DB_CONNECT_ERROR",
    "message": "Connection refused"
  }
}
```



Structured Log

```
{  
  "timestamp": "2024-06-17T08:55:00Z",  
  "level": "INFO",  
  "message": "User 'user123' logged in",  
  "user": "user123",  
  "ip_address": "192.168.0.1"  
},  
{  
  "timestamp": "2024-06-17T08:55:01Z",  
  "level": "ERROR",  
  "message": "Database connection failed",  
  "error": {  
    "code": "DB_CONNECT_ERROR",  
    "message": "Connection refused"  
  }  
}
```

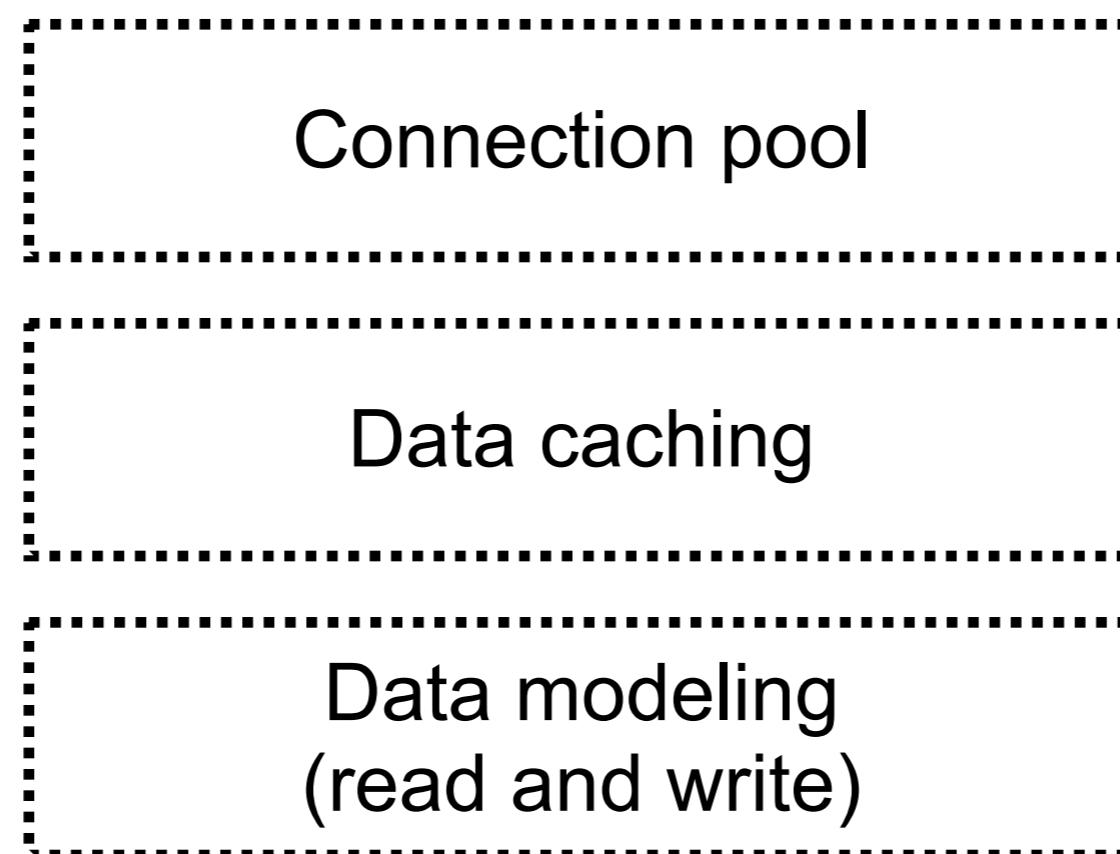
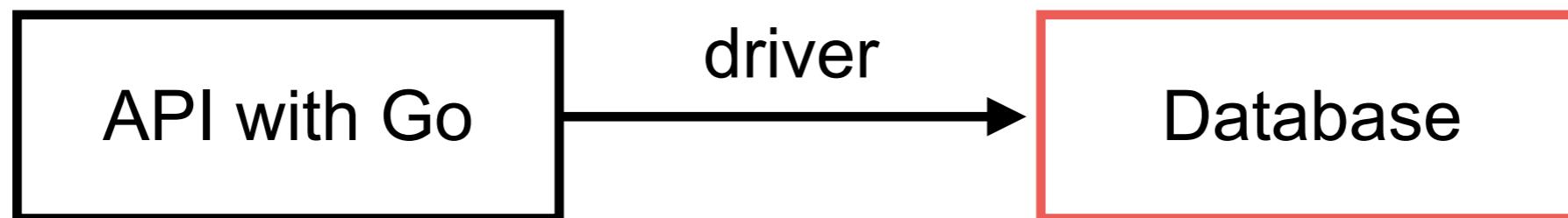
Machine readable format
Key-value pairs
Contextual information
Hierarchical structure



Database



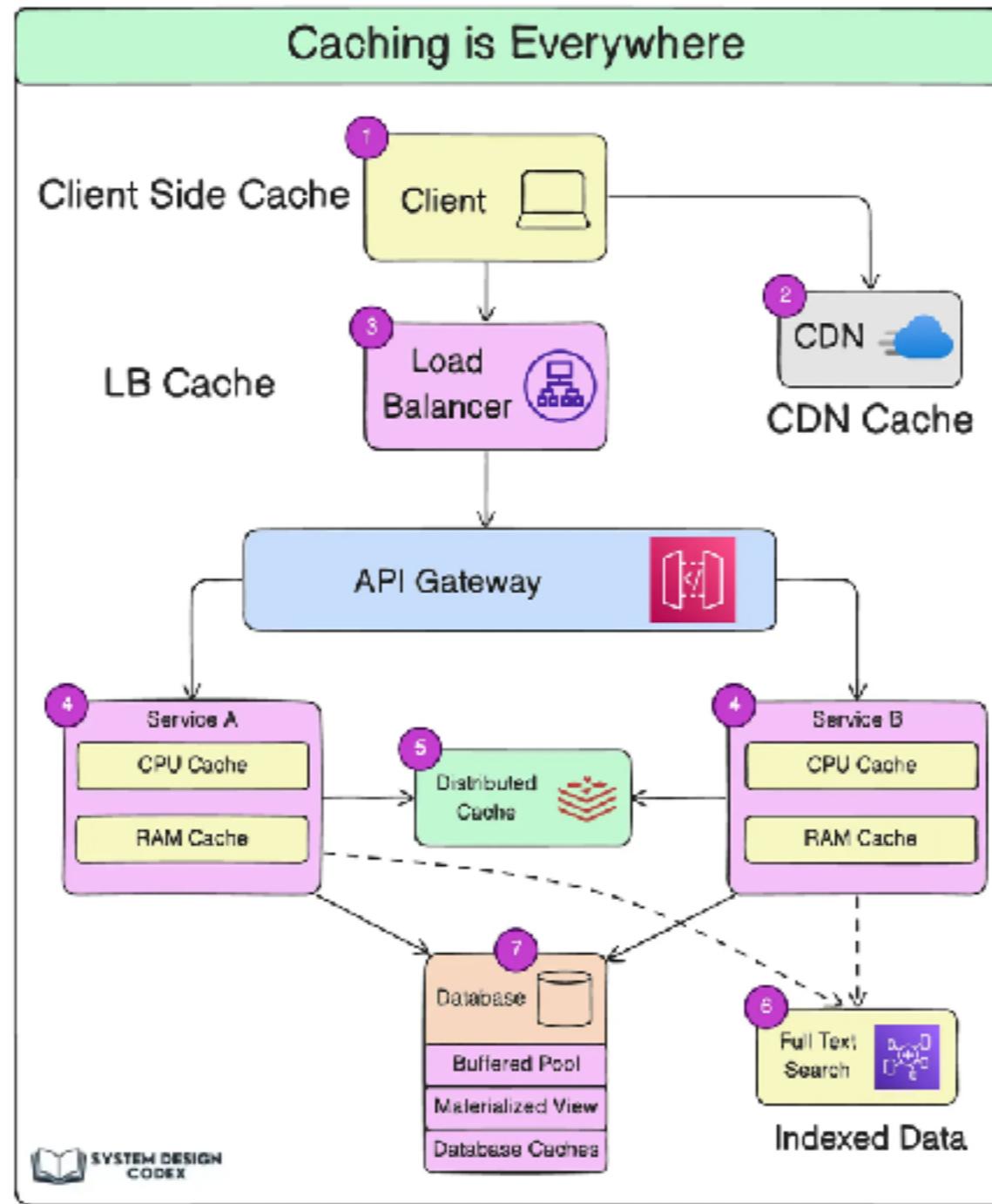
Project Structure



Data Caching



All about caching



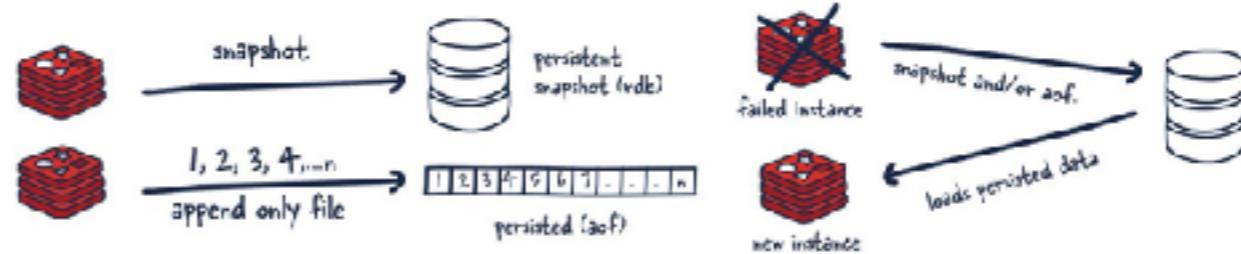
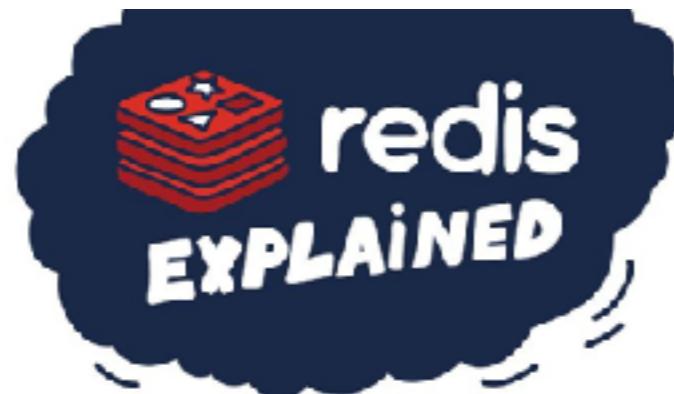
<https://newsletter.systemdesigncodex.com/p/caching-at-multiple-levels>



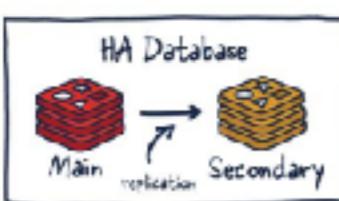
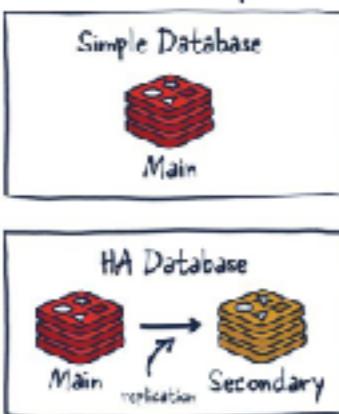
Go workshop

© 2022 - 2025 Siam Chamnkit Company Limited. All rights reserved.

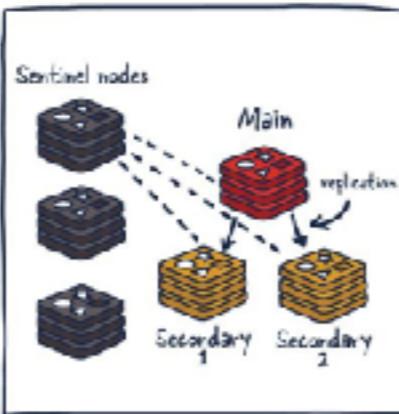
Redis



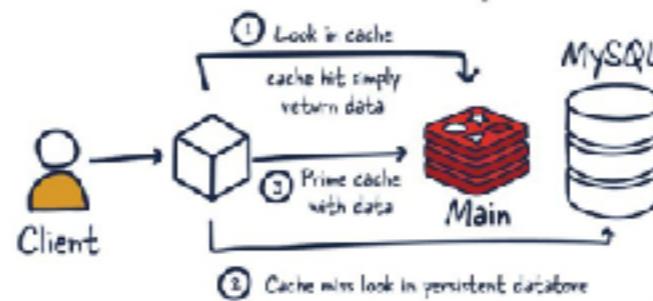
Redis setup



Redis sentinel

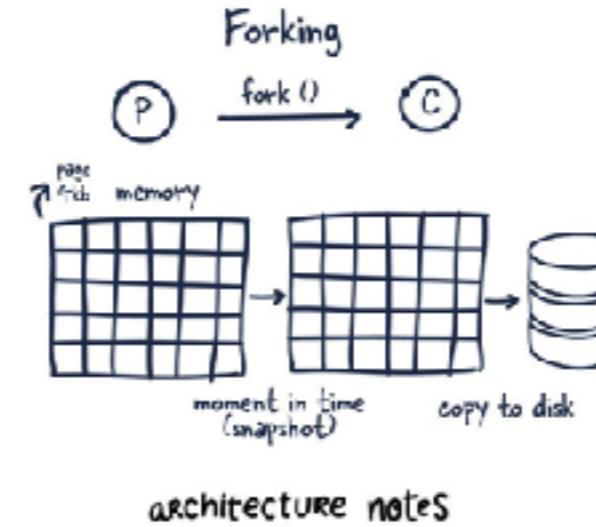
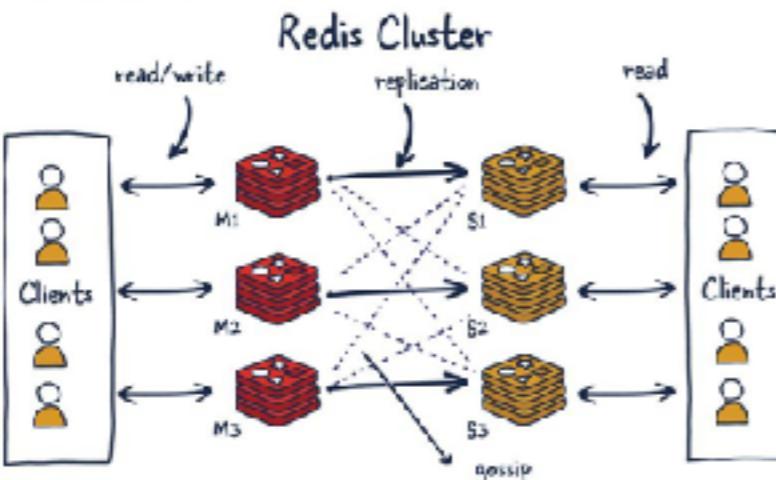


How is redis traditionally used



KEY
foo bar
VALUE

hello world	String
010101010101010101	Bitmap
{23334}{6634728}{916}	Bitfield
{a: "Hello", b: "World"}	Hash
[A>B>C>C]	List
{A<B<C}	Set
[A1, B2, C3]	Sorted set
{A: (50.1, 0.5)}	Geospatial
010101010101010101	Hyperlog
[id=blue, self = "foo", a: "bar"]	Stream



architecture notes

<https://architecturenotes.co/p/redis>



Go workshop

© 2022 - 2025 Siam Chamnkit Company Limited. All rights reserved.

Memcached vs Redis

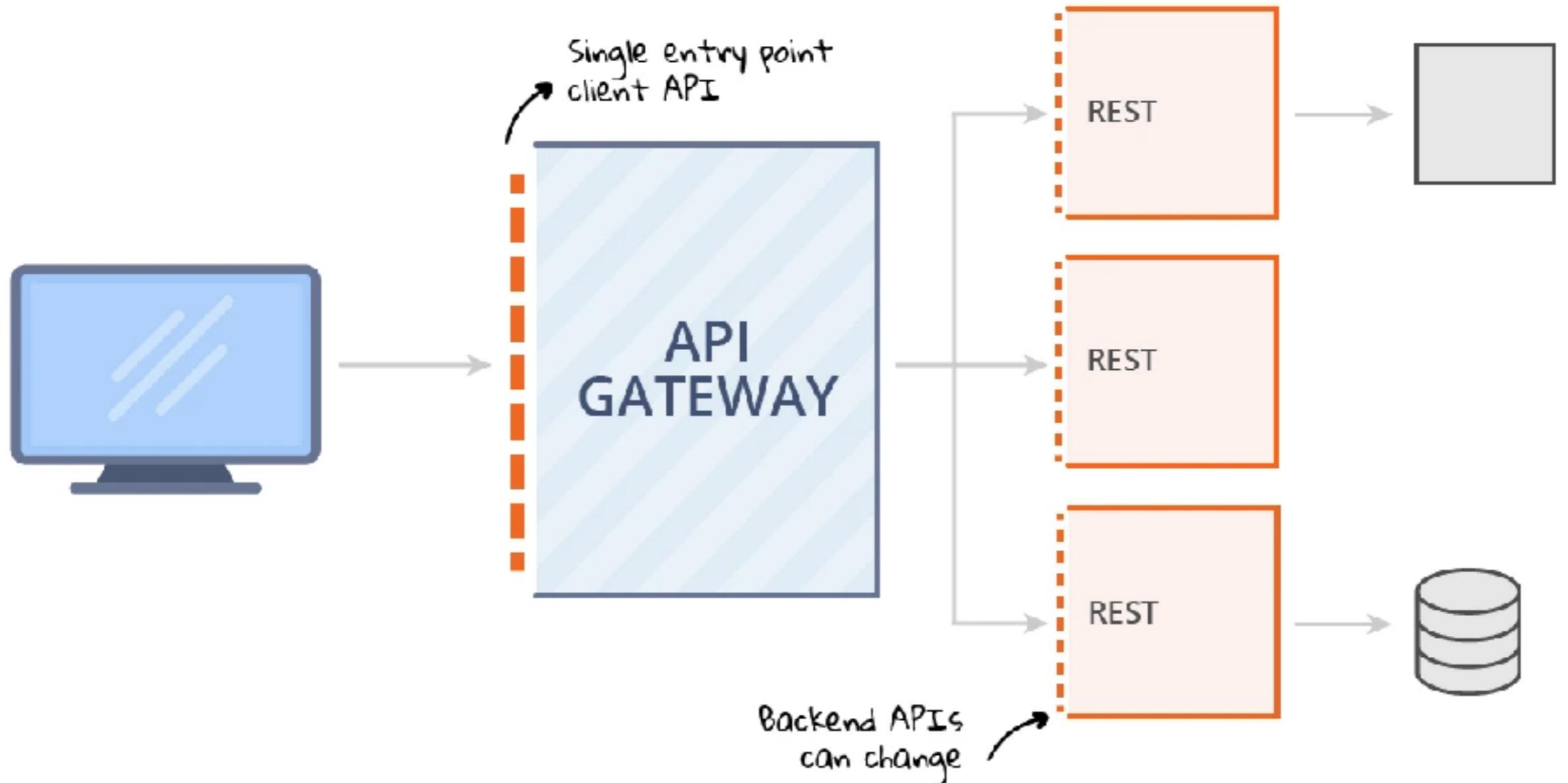
Features	Memcached	Redis
Store key-value In memory	Yes	Yes
Opensource	Yes	Yes
Support difference data types	No (String)	Yes (String, list, set)
Persistence	No	Yes
Replication	No	Yes
Cluster	No	Yes
Support multi-thread	Yes	Yes
Eviction policy	Yes (LRU only)	Yes



API Gateway



API Gateway

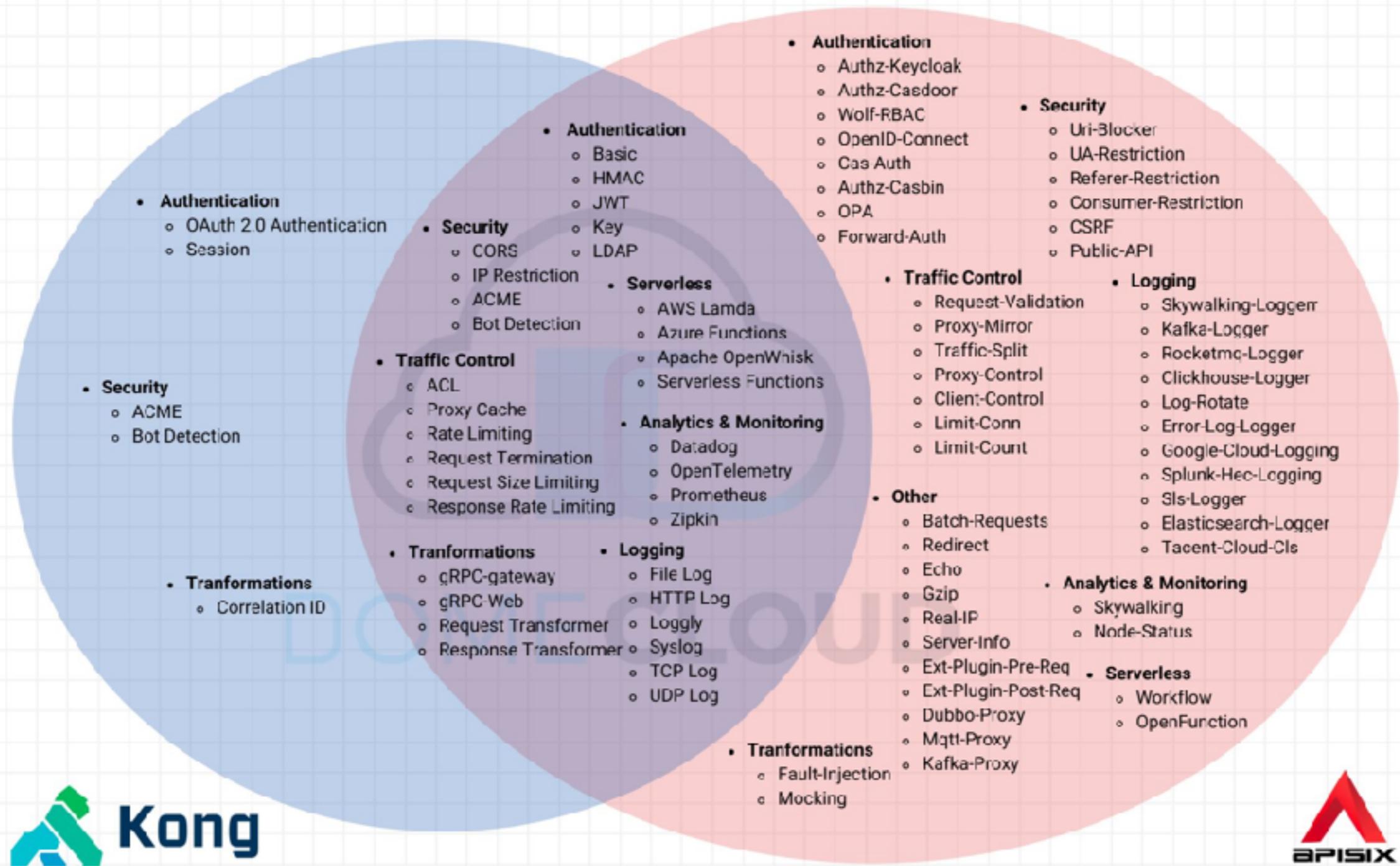


API Gateway

Authentication and Authorization
Rate limit
Circuit breaker
Metric, Tracing, Logging
Refactor assertion g



Kong vs. APISIX Plugins comparison



Go workshop

© 2022 - 2025 Siam Chamnankit Company Limited. All rights reserved.

API Gateway



<https://landscape.cncf.io/guide#orchestration-management--api-gateway>



Go workshop

© 2022 - 2025 Siam Chamnkit Company Limited. All rights reserved.

Security APIs

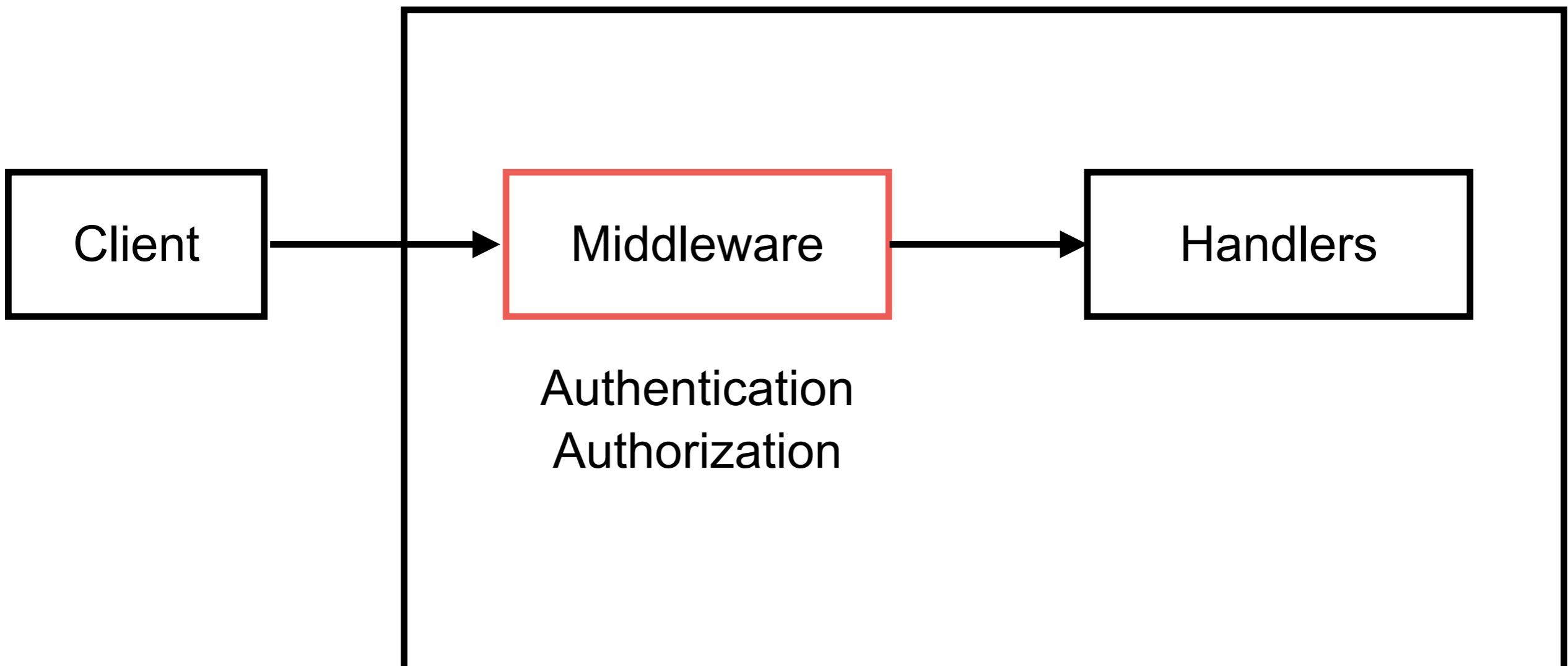
Authentication

Authorization

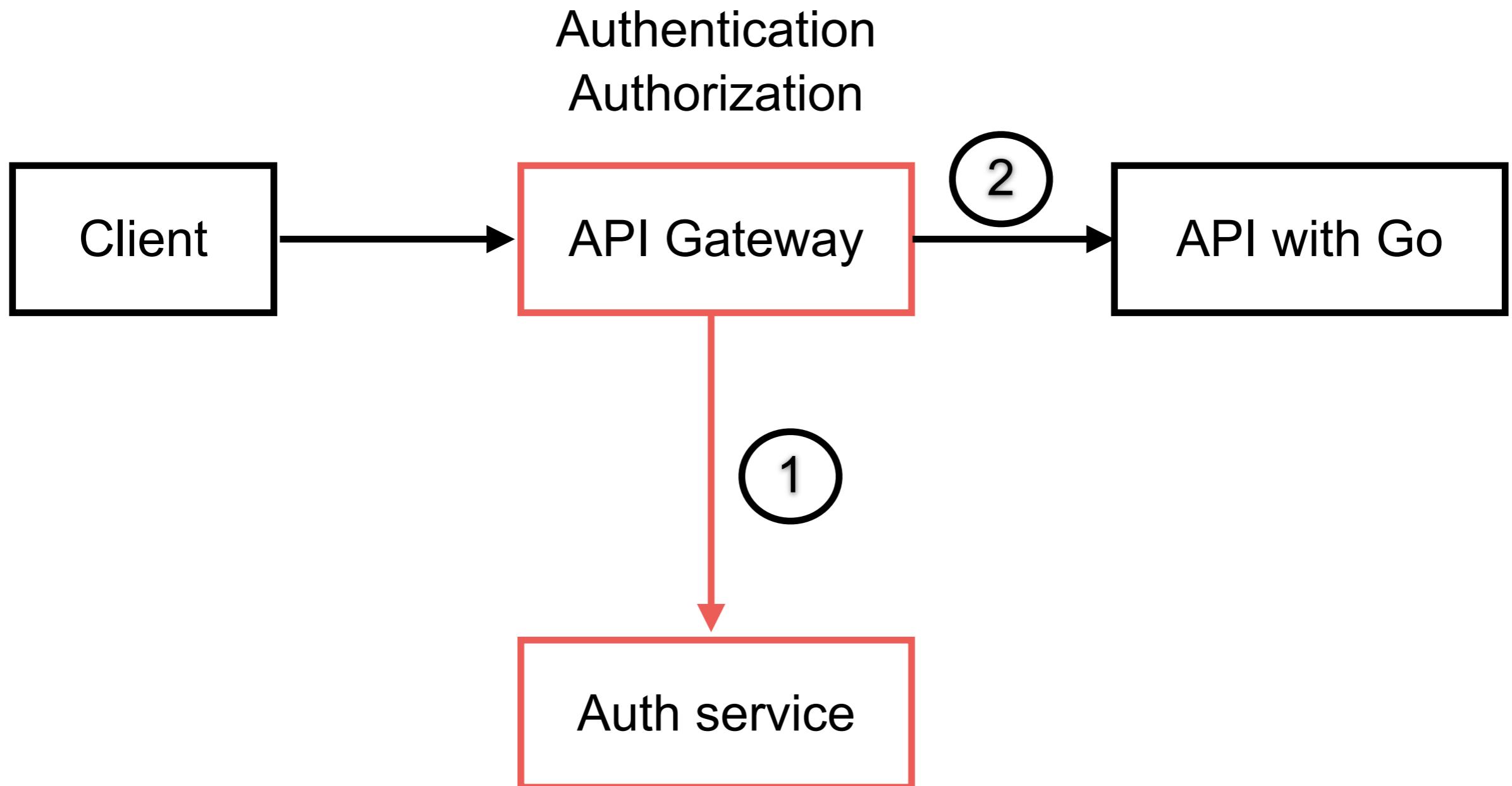


Project Structure

RESTFul API with Go



Project Structure



Observable Service



Observable Service

Application
metric

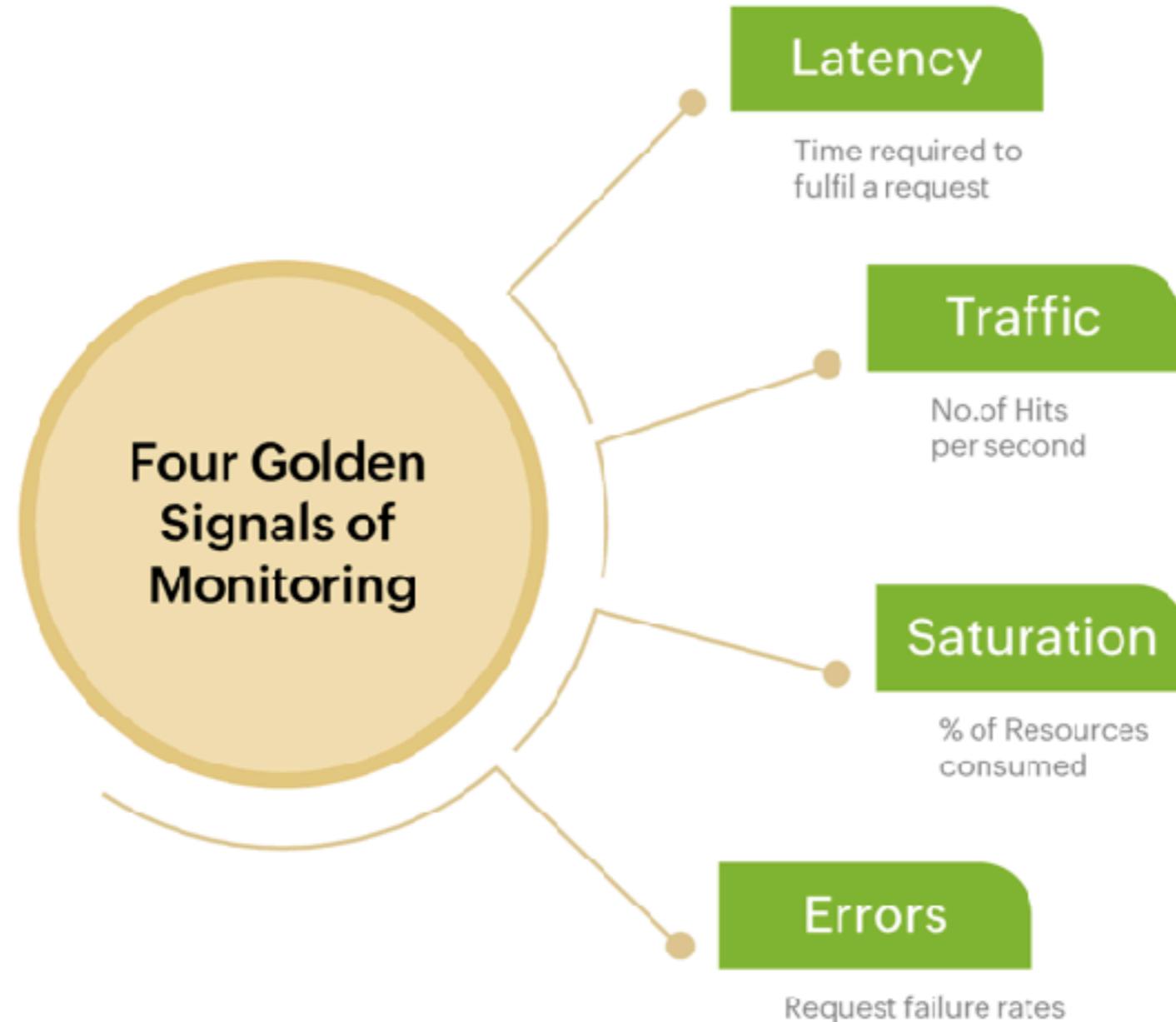
Distributed
tracing

Log aggregation

Exception
tracking



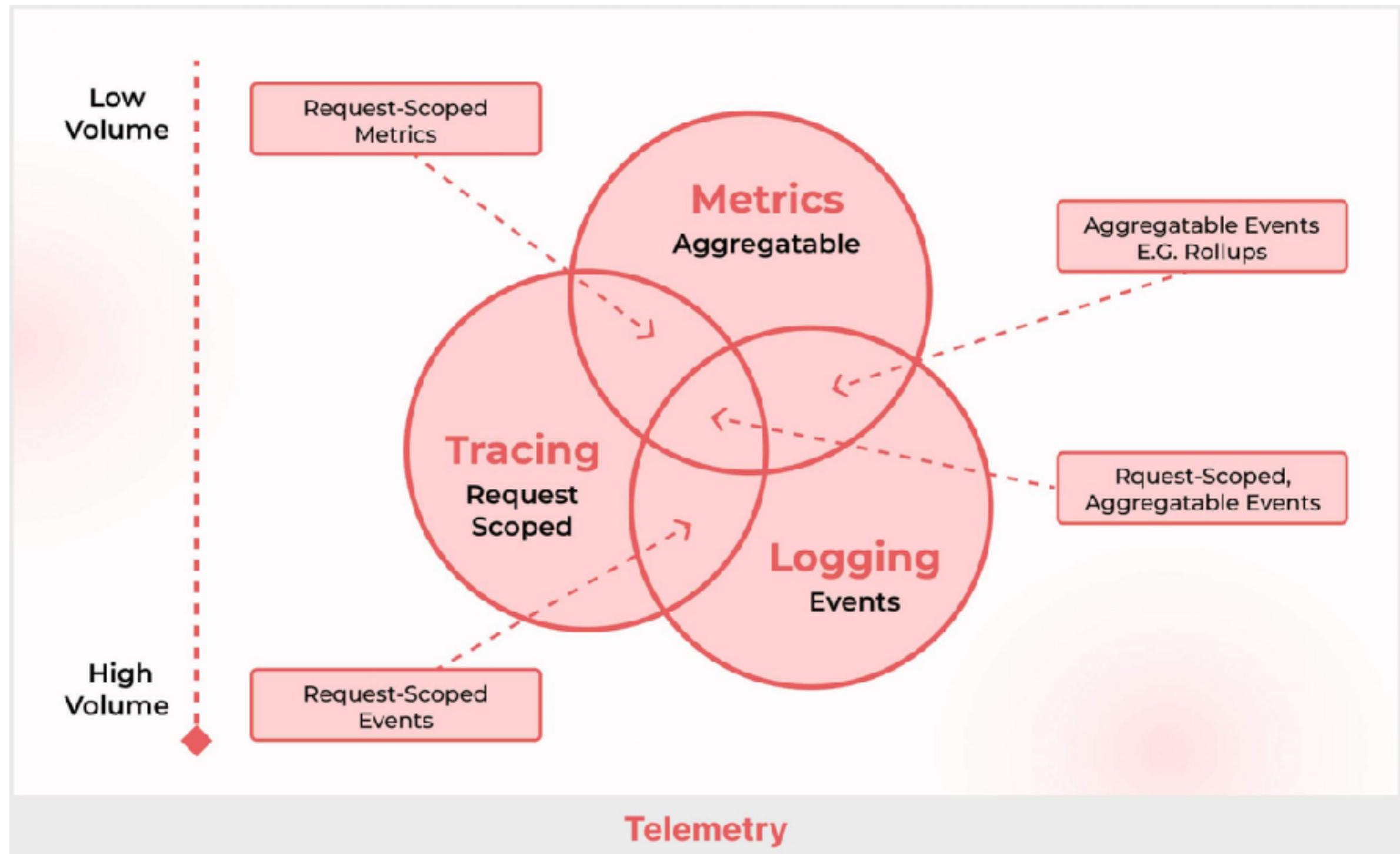
The 4 Golden Signals



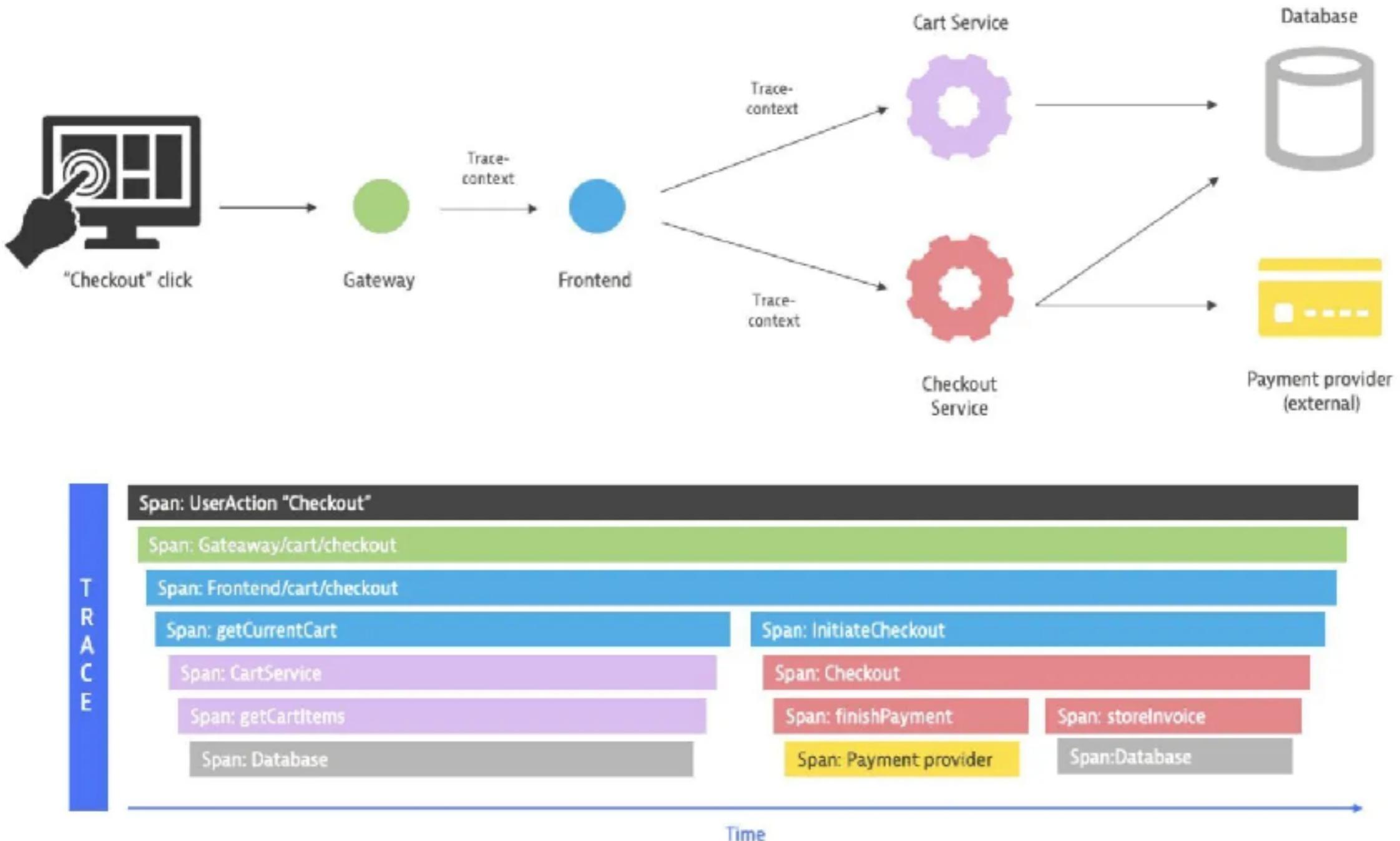
<https://sre.google/sre-book/monitoring-distributed-systems/>



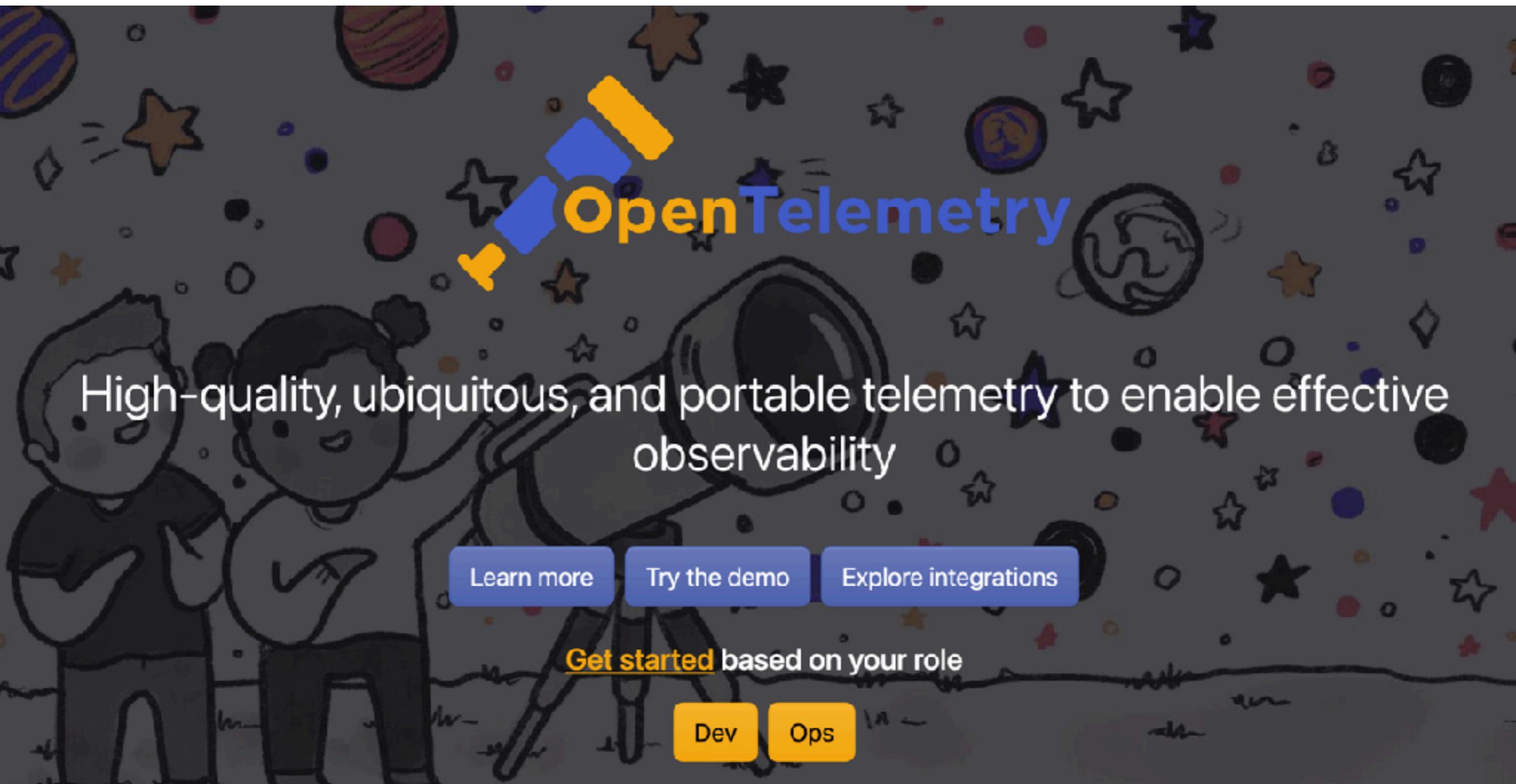
3 Pillars of Observability



Tracing



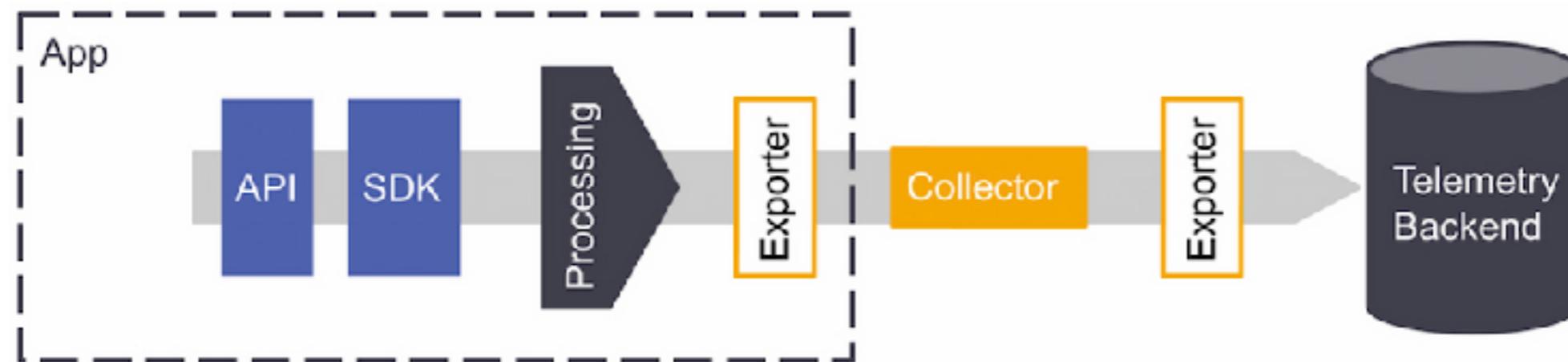
OpenTelemetry



<https://opentelemetry.io/>



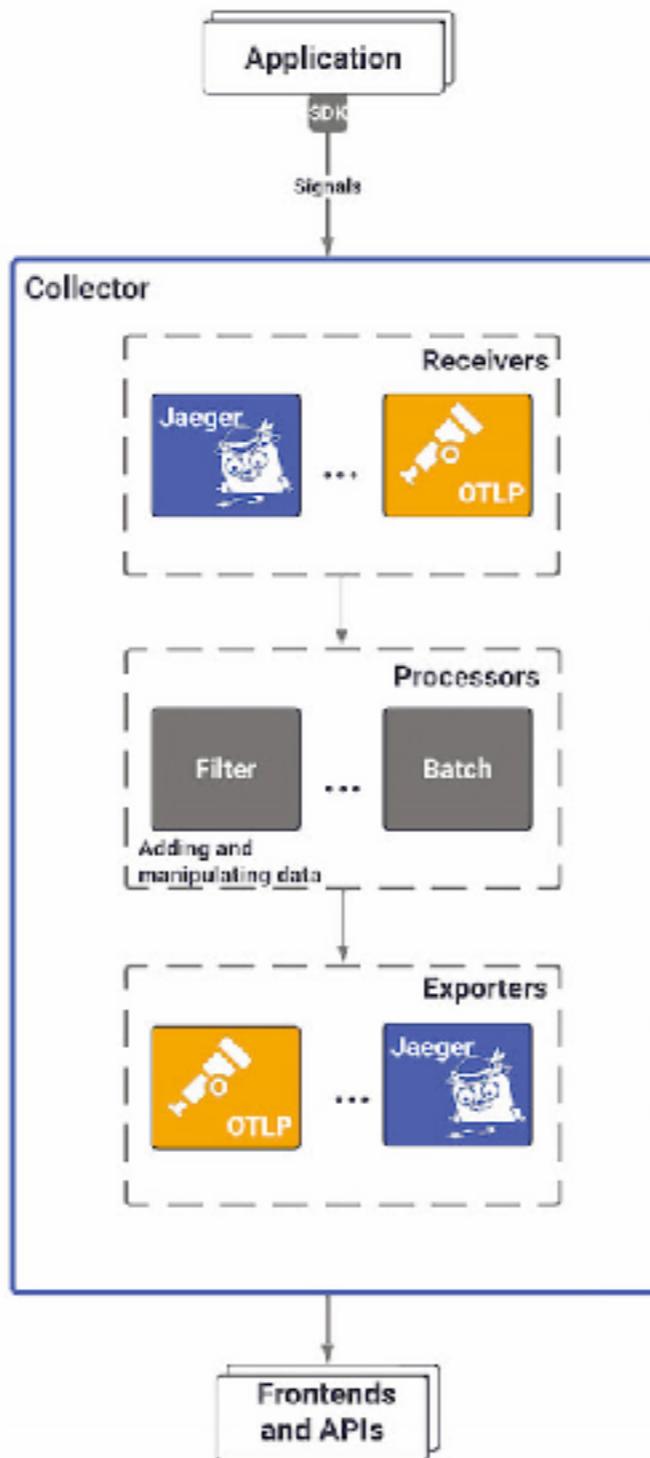
OpenTelemetry Structure



<https://dzone.com/refcardz/getting-started-with-opentelemetry>



OpenTelemetry Structure



<https://dzone.com/refcardz/getting-started-with-opentelemetry>



Go workshop

© 2022 - 2025 Siam Chamnankit Company Limited. All rights reserved.

Messaging

RabbitMQ

Apache Kafka

