

# Basic of Go programming





Somkiat Puisungnoen

Somkiat Puisungnoen

Update Info 1 View Activity Log 10+ ...

Timeline About Friends 3,138 Photos More

When did you work at Opendream? X

... 22 Pending Items

Intro

Software Craftsmanship

Software Practitioner at สยามชัมนาณกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Somkiat Puisungnoen 15 mins · Bangkok · ⚙️

Java and Bigdata





Page

Messages

Notifications 3

Insights

Publishing Tools

Settings

Help ▾



somkiat.cc

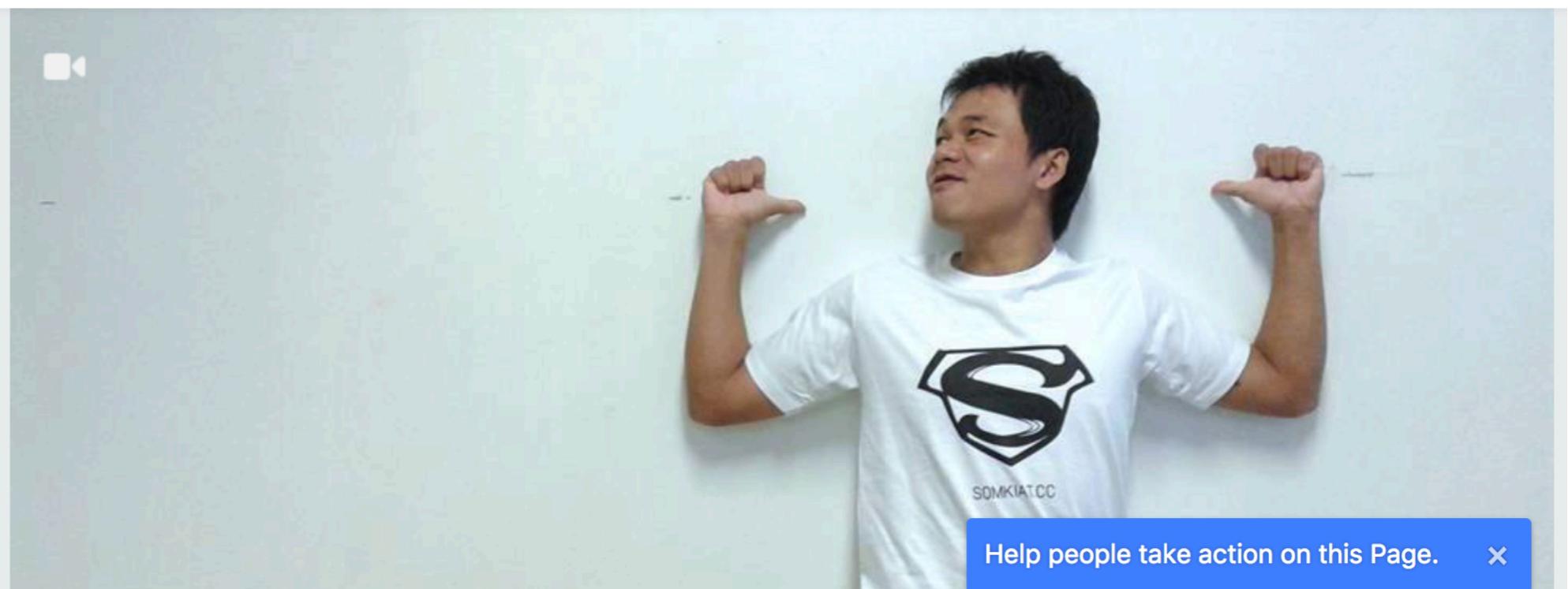
@somkiat.cc

Home

Posts

Videos

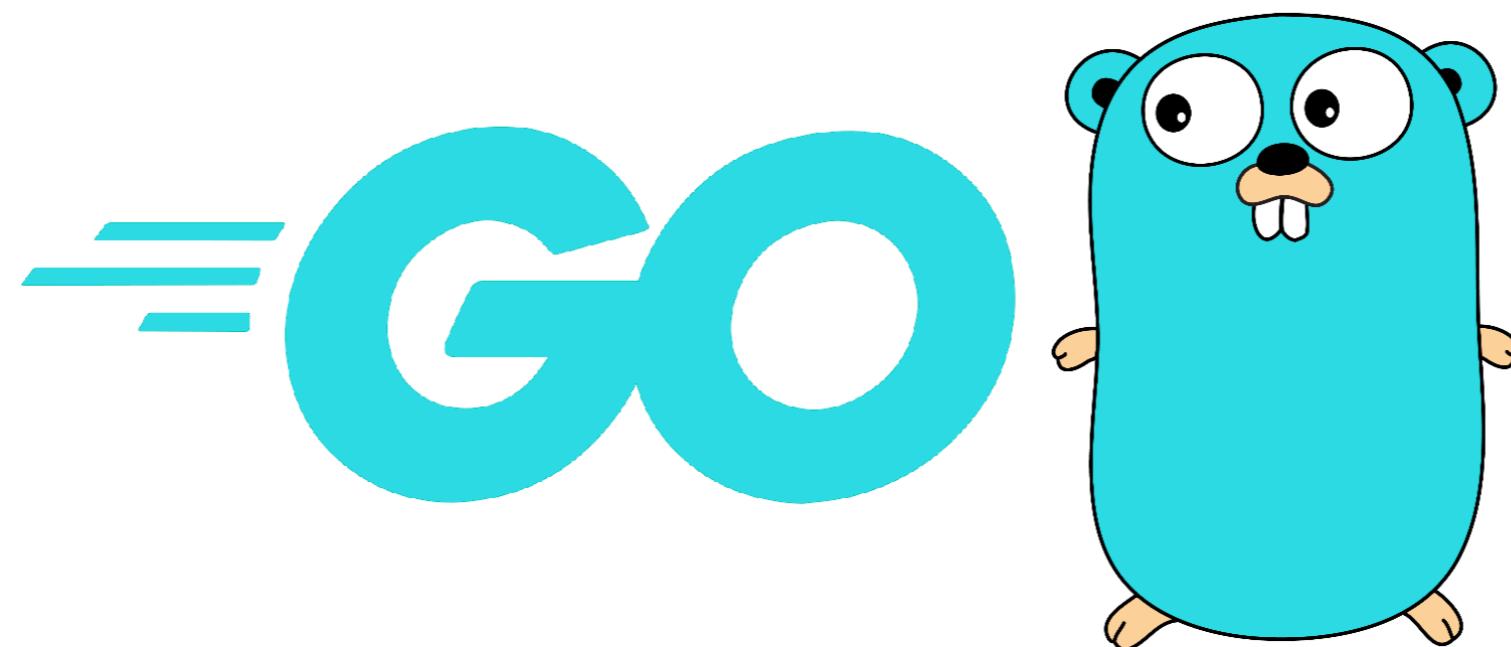
Photos



**[https://github.com/up1/  
course-go-2021](https://github.com/up1/course-go-2021)**



# Let's



# About Go

Compiled language  
Modern and Fast  
Powerful of standard library  
Concurrency build-in  
Static language  
Perform garbage collection  
Designed for multi-core computers



# Go's inspiration

**C => statement and expression syntax**

**Pascal => declaration syntax**

**Modula/Oberon 2 => package**

**CSP/Occam/Limbo => concurrency**

**BCPL => the semicolon rule**

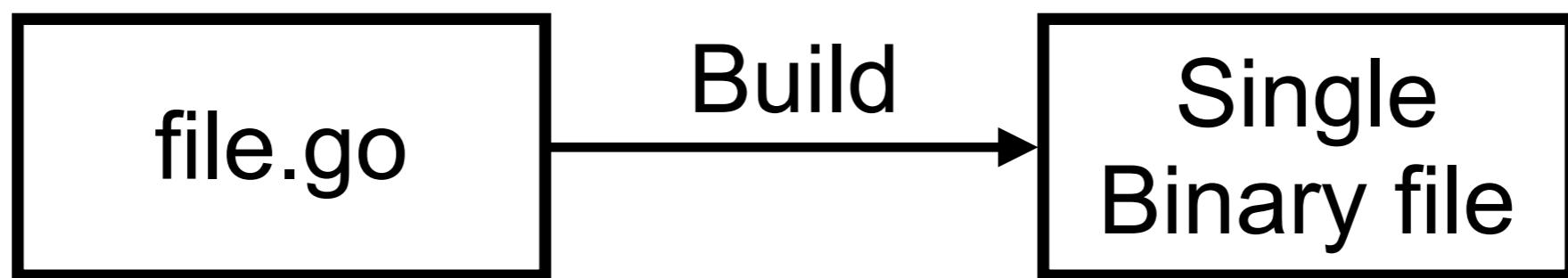
**Smalltalk => method**

**Newsqueak => <-, :=**

**APL => iota**



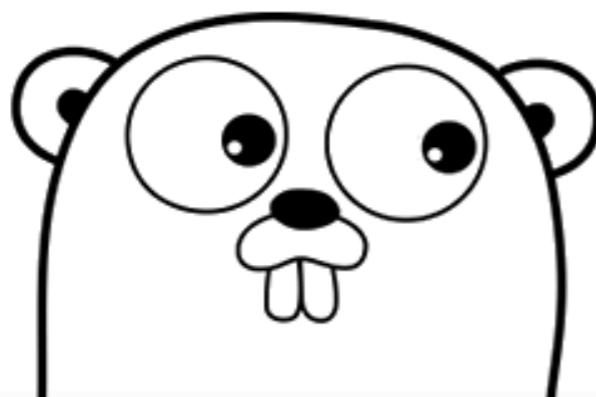
# About Go



# Installation

[Documents](#)[Packages](#)[The Project](#)[Help](#)[Blog](#)[Play](#)

Go is an open source programming language that makes it easy to build **simple, reliable, and efficient** software.

[Download Go](#)

Binary distributions available for Linux, macOS, Windows, and more.

[Try Go](#)[Open in Playground ↗](#)

```
// You can edit this code!
// Click here and start typing.
package main

import "fmt"

func main() {
    fmt.Println("Hello, 世界")
}
```

[Run](#)[Share](#)[Tour](#)

<https://golang.org/>



# Hello Go

## \$go version



# Hello Go environment

\$go env



# Development tools



# Visual Studio Code

The screenshot shows the official website for Visual Studio Code. At the top, there's a navigation bar with links for "Visual Studio Code", "Docs", "Updates", "Blog", "API", "Extensions", and "FAQ". There's also a search icon and a large blue "Download" button. A banner at the top says "Version 1.47 is now available! Read about the new features and fixes from June." Below the banner, the main headline reads "Code editing. Redefined." followed by the subtext "Free. Built on open source. Runs everywhere." A prominent blue button says "Download for Mac Stable Build". To the right, there's a screenshot of the VS Code interface showing the code editor with some JavaScript code, the Extensions Marketplace sidebar listing various extensions like Python, GitLens, C/C++, ESLint, and Vetur, and the terminal window showing build logs.

<https://code.visualstudio.com/>

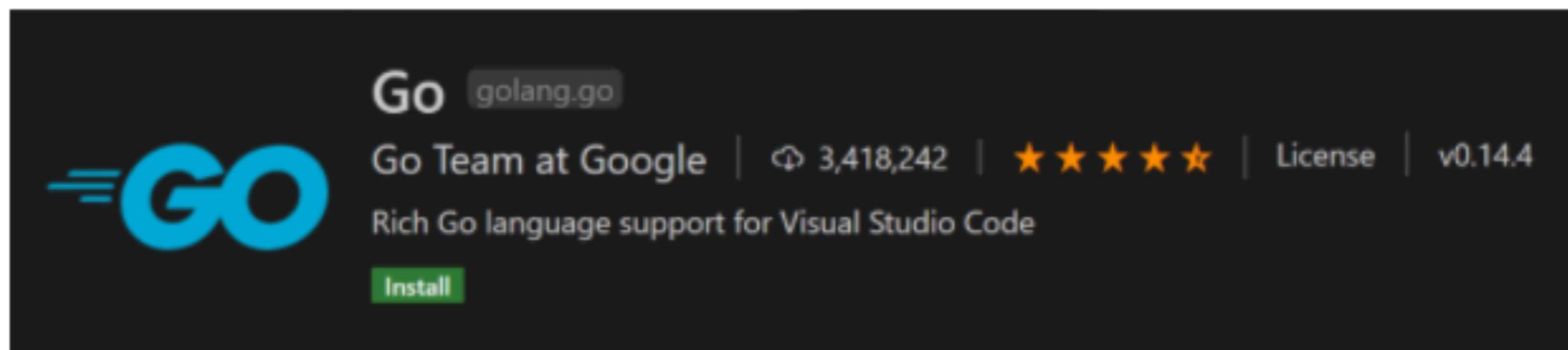


# Extension for Go

## Go in Visual Studio Code



Using the Go extension for Visual Studio Code, you get language features like IntelliSense, code navigation, symbol search, bracket matching, snippets, and many more that will help you in [Golang](#) development.



You can install the Go extension from the [VS Code Marketplace](#).

<https://code.visualstudio.com/docs/languages/go>



# Resources for beginner



# Go tour

## A Tour of Go

### Hello, 世界

Welcome to a tour of the Go programming language.

The tour is divided into a list of modules that you can access by clicking on [A Tour of Go](#) on the top left of the page.

You can also view the table of contents at any time by clicking on the [menu](#) on the top right of the page.

Throughout the tour you will find a series of slides and exercises for you to complete.

You can navigate through them using

- "[previous](#)" or PageUp to go to the previous page,
- "[next](#)" or PageDown to go to the next page.

The tour is interactive. Click the [Run](#) button now (or type shift-enter) to compile and run the program on a remote server. The result is displayed below the code.

These example programs demonstrate different aspects of Go. The programs in the tour are meant to be starting points for your own experimentation.

Edit the program and run it again.

Note that when you click on [Format](#) or ctrl-enter the text in the editor is formatted using the [fmt](#) tool. You can switch syntax highlighting on and off by clicking on the [Imports off](#) and [Syntax off](#) buttons.

< 1/5 >

```
hello.go
```

```
1 |
2 package main
3
4 import ("fmt")
5
6 func main() {
7     fmt.Println("Hello, 世界")
8 }
9
```

Imports off Syntax off

Reset Format Run



<https://tour.golang.org>



# Effective Go

## Effective Go

Introduction  
Examples  
Formatting  
Commentary  
Names  
  Package names  
  Getters  
  Interface names  
  MixedCaps  
Semicolons  
Control structures  
  If  
  Redeclaration and reassignment  
  For  
  Switch  
  Type switch  
Functions  
  Multiple return values  
  Named result parameters  
  Defer

Constants  
Variables  
The init function  
Methods  
  Pointers vs. Values  
Interfaces and other types  
  Interfaces  
  Conversions  
  Interface conversions and type assertions  
  Generality  
  Interfaces and methods  
The blank identifier  
  The blank identifier in multiple assignment  
  Unused imports and variables  
  Import for side effect  
  Interface checks  
Embedding  
Concurrency  
  Share by communicating  
  Goroutines

[https://golang.org/doc/effective\\_go.html](https://golang.org/doc/effective_go.html)



# Learn Go

## Learn

Saurabh Hooda edited this page on Jul 1 · 33 revisions

---

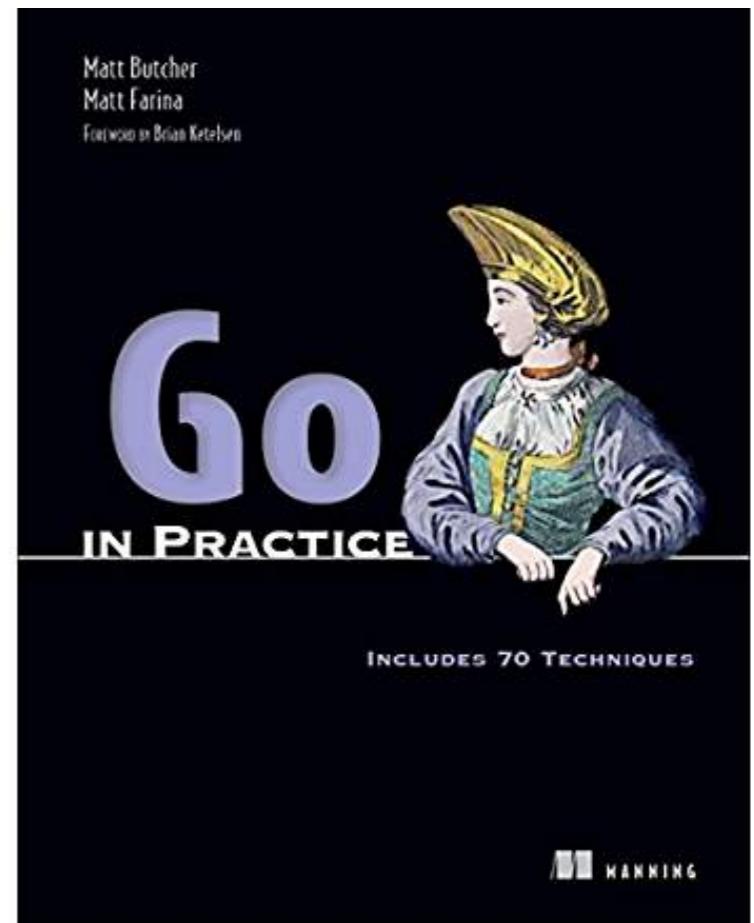
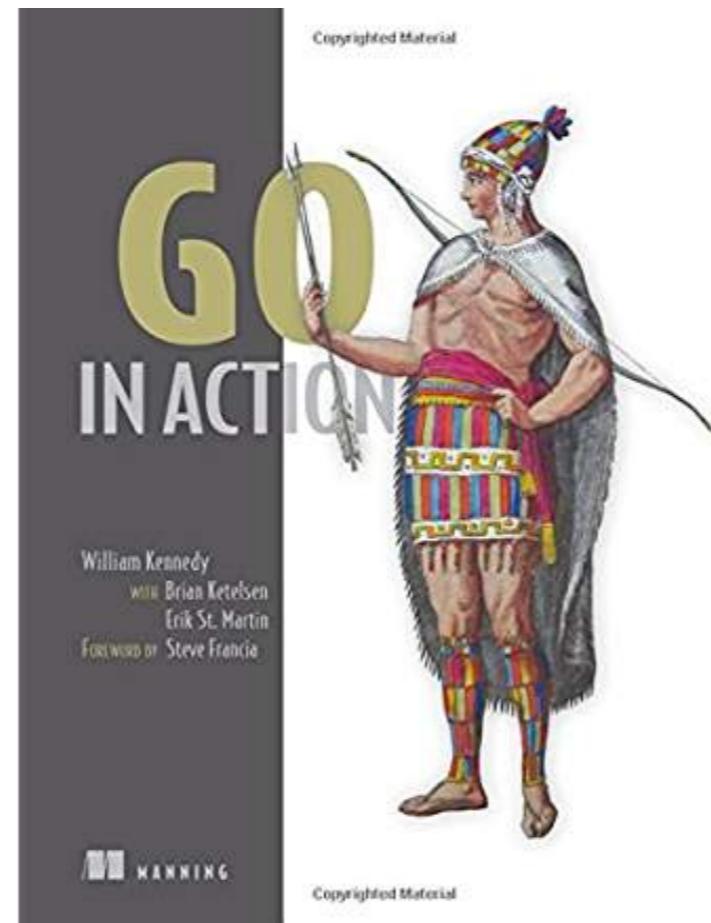
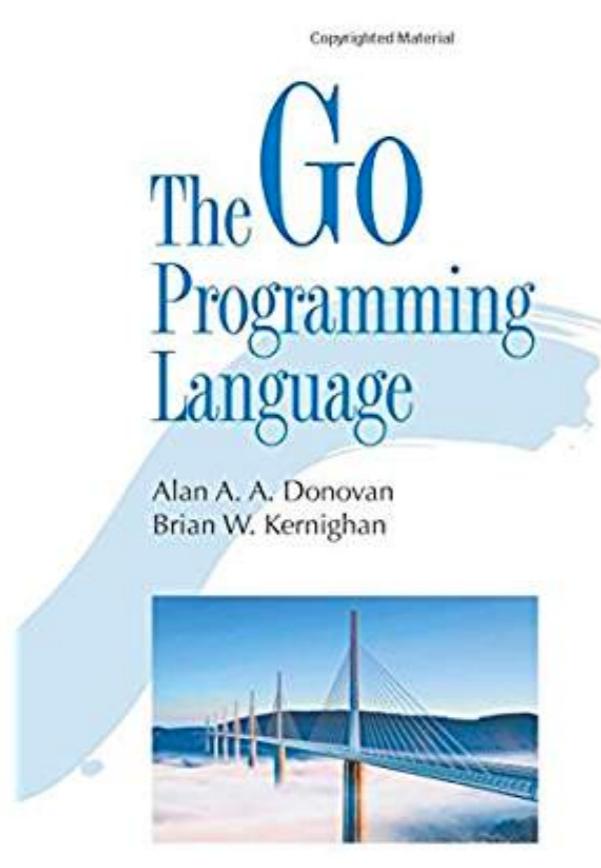
In addition to the resources available at [golang.org](https://golang.org) there are a range of community-driven initiatives:

- [The Little Go Book](#)
- [Exercism.io - Go](#) - Online code exercises for Go for practice and mentorship.
- [Learn Go in an Hour - Video 2015-02-15](#)
- [Learning to Program in Go](#), a multi-part video training class.
- [Pluralsight Classes for Go](#) - A growing collection of (paid) online classes.
- [Ardan Labs Training](#) - Commercial, live instruction for Go programming.
- [O'Reilly Go Fundamentals](#) - Video learning path for Go programming.
- [Go By Example](#) provides a series of annotated code snippets.
- [Learn Go in Y minutes](#) is a top-to-bottom walk-through of the language.
- [Workshop-Go](#) - Startup Slam Go Workshop - examples and slides.
- [Go Fragments](#) - A collection of annotated Go code examples.
- [50 Shades of Go: Traps, Gotchas, Common Mistakes for New Golang Devs](#)

<https://github.com/golang/go/wiki/Learn>



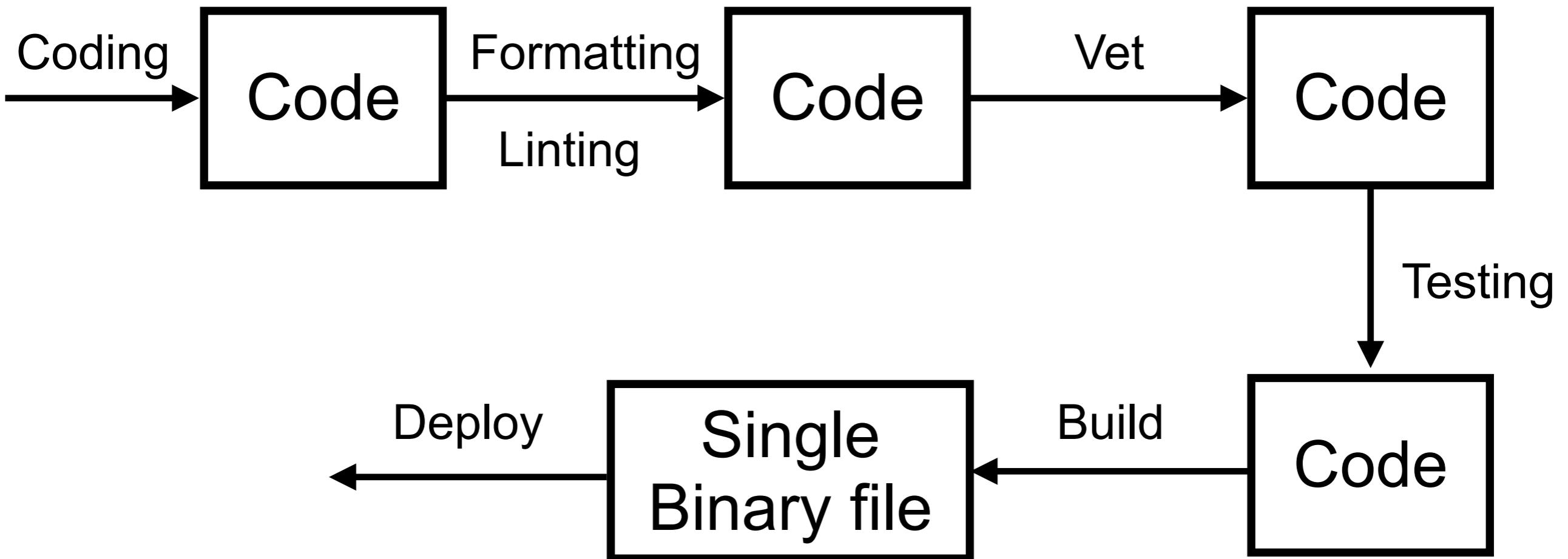
# Books



# Go development workflow



# Development workflow



<https://golang.org/cmd/go/>



# Coding

```
package main

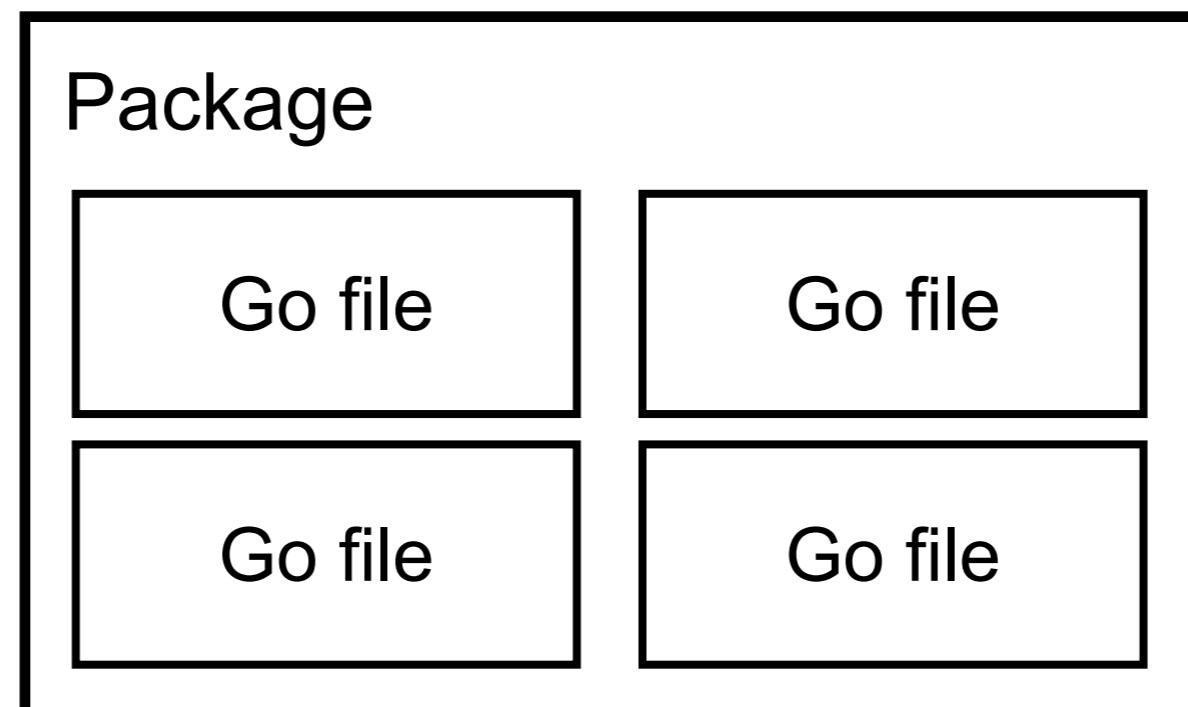
import "fmt"

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



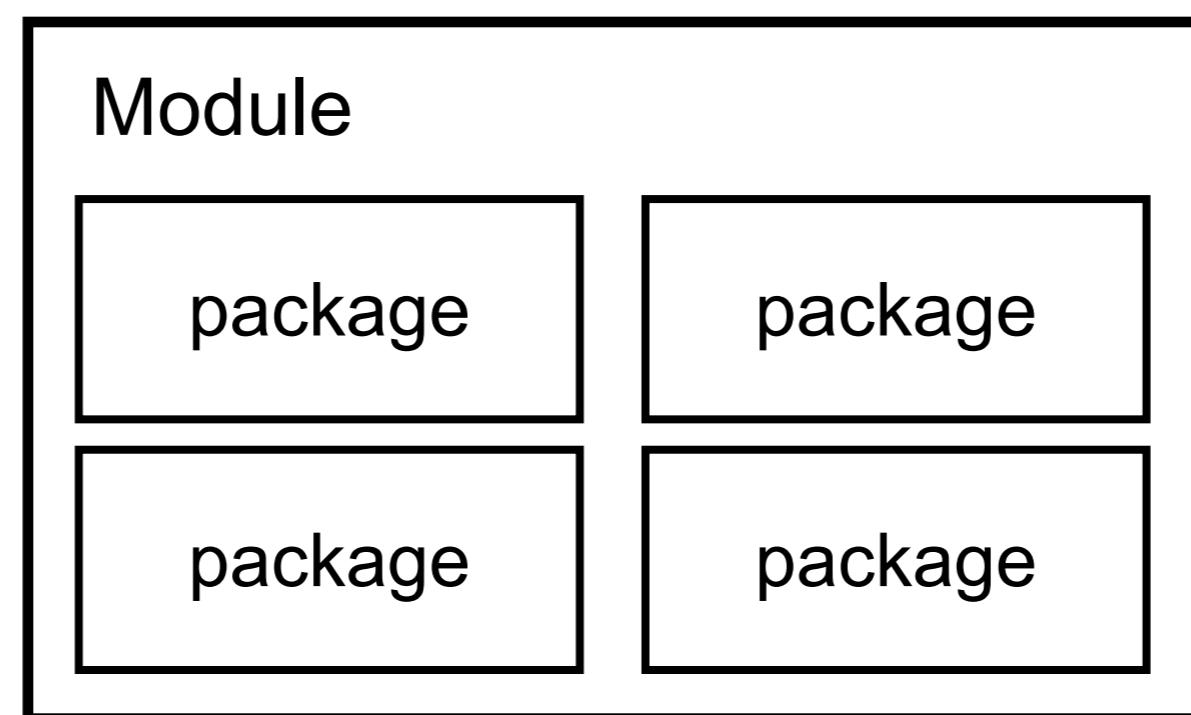
# Package

Collection of source files in the save directory  
Main package = “main”



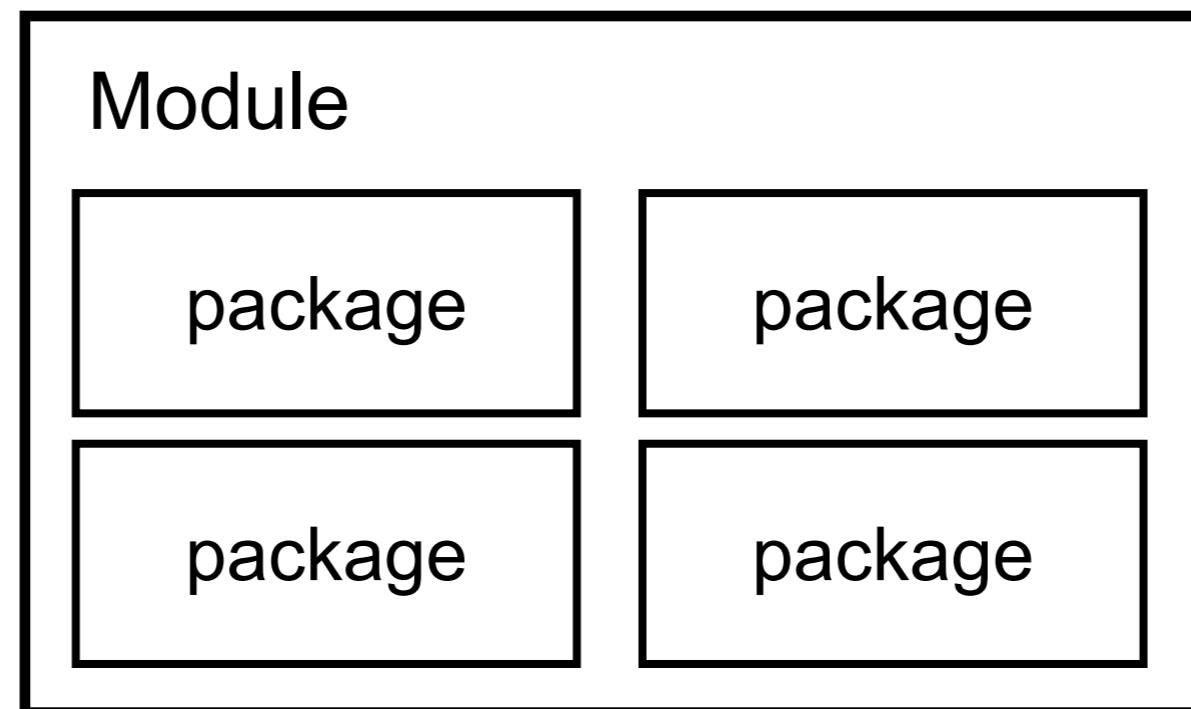
# Module

Collection of related Go packages that are released together



# Module

\$go mod init <name>



*Remote modules keep in \$GOPATH/pkg/mod*



# Formatting

Reformat Go source code

```
$go fmt <filename>  
$go fmt  
$go fmt ./...
```

<https://golang.org/cmd/gofmt/>



# Linting

Check mistake of coding style

```
$go get -u golang.org/x/lint/golint  
$golint
```

<https://github.com/golang/lint>



# Vet

Help you detect any suspicious, abnormal,  
or useless code in your application.

**\$go vet**

**\$go tool vet help**

<https://golang.org/cmd/vet/>



# Build

```
$go install  
$go build -o <output>
```

<https://golang.org/pkg/go/build/>



# Build with OS specified

```
$GOOS=windows go build  
$GOARCH=amd64 go build
```

<https://golang.org/doc/install/source#environment>

<https://github.com/golang/go/blob/master/src/go/build/syslist.go>



# Testing

## Production code

```
package hello

func SayHi(m string) string {
    return "Hello " + m
}
```

## Test code

```
package hello_test

import (
    "hello"
    "testing"
)

func TestHello(t *testing.T) {
    result := hello.SayHi("somkiat")
    if result != "Hello somkiat" {
        t.Errorf("Error %v", result)
    }
}
```

<https://golang.org/pkg/testing/>



# Testing

\$go test

\$go test ./...

\$go help test

<https://golang.org/pkg/cmd/go/internal/test/>



# Testing with coverage

```
$go test -cover
```

```
$go test -coverprofile=coverage.out
```

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

<https://golang.org/pkg/cmd/go/internal/test/>



# Testing with benchmark

\$go test -bench .

```
func BenchmarkHello(b *testing.B) {
    for i := 0; i < b.N; i++ {
        hello.SayHi("somkiat")
    }
}
```

[https://golang.org/cmd/go/#hdr-Testing\\_flags](https://golang.org/cmd/go/#hdr-Testing_flags)



# Basic of Go



# **Features of Go is no new features**



# Keywords

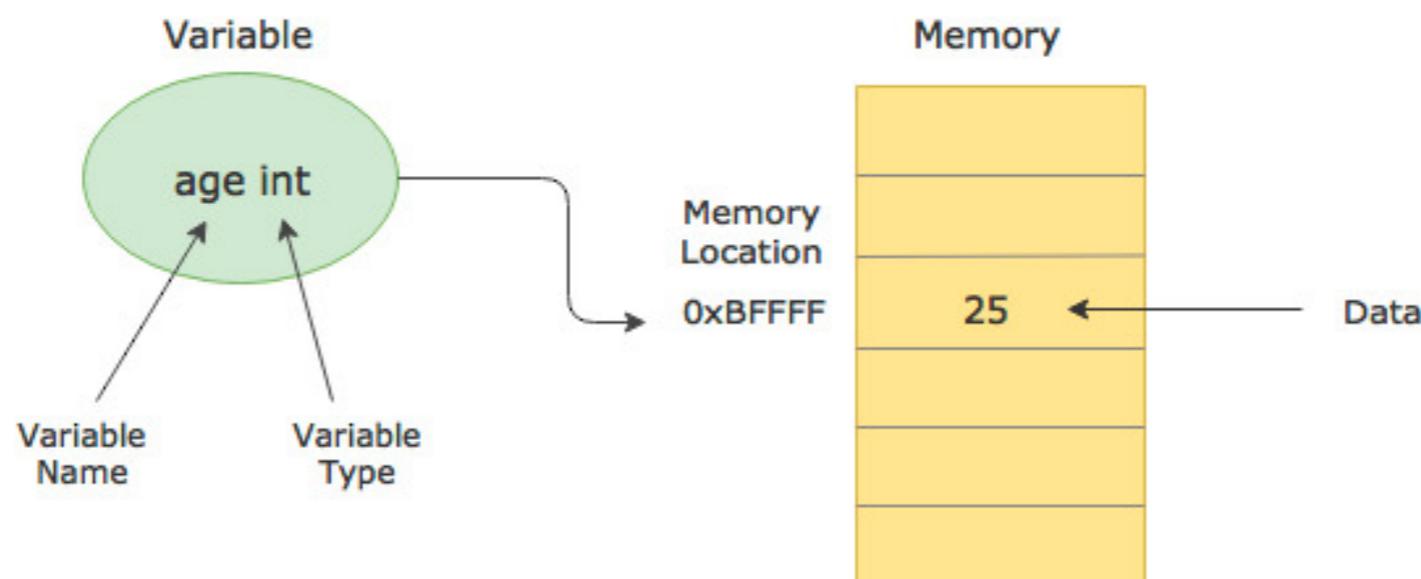
break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

<https://golang.org/ref/spec#Keywords>



# Variables and Data types

A variable is just a convenient name given to a memory location where the data is stored



# Define variables

```
var <variableName> <type>
```

```
var a int  
var i, j, k int  
var b int = 1  
var x, y, z = 1, 2, 3
```

```
// Short assignment  
number := 1  
name := "Hello"
```

```
// _ (blank) is a special variable name  
_, email := 1, "xxx.com"
```



# Define variables

Declare variables that are set to their zero value

```
var a int  
var i, j, k int  
var s string
```



# Zero values

Any variable declared without an initial value will have a **zero-value** depending on the type

Type	Zero value
bool	FALSE
string	""
Int, int8, int16, int32	0
float32, float64	0.0
Pointer, struct, interface	Nil

[https://golang.org/ref/spec#The\\_zero\\_value](https://golang.org/ref/spec#The_zero_value)

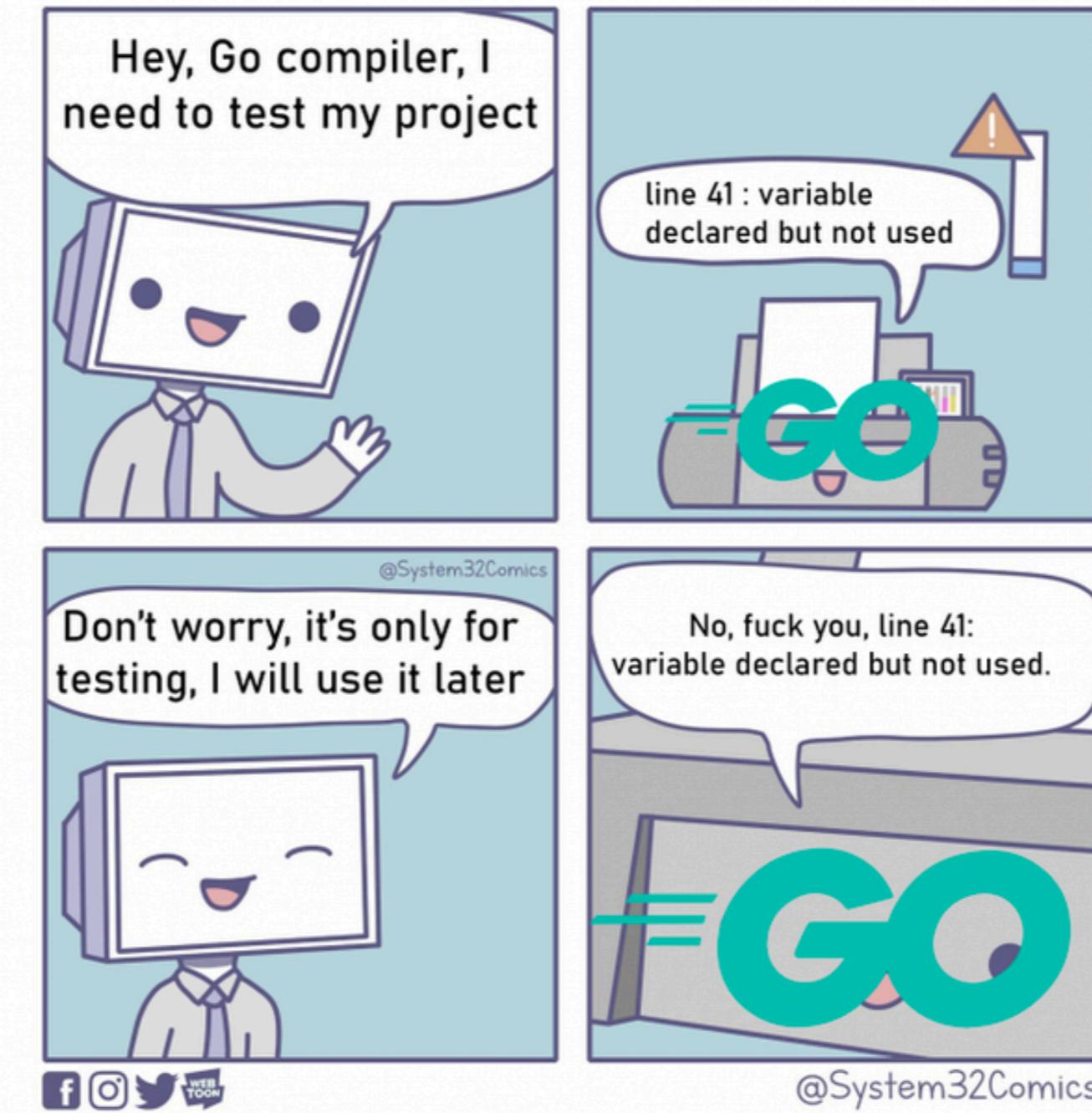


# Grouping

```
var (
    a      int
    i, j, k int
    b      int = 1
    x, y, z      = 1, 2, 3
)
```



# Compiler features



# Compiler features

\$go run variable.go

```
./variable.go:4:6: a declared but not used
./variable.go:5:6: i declared but not used
./variable.go:5:9: j declared but not used
./variable.go:5:12: k declared but not used
./variable.go:6:6: b declared but not used
./variable.go:7:6: x declared but not used
./variable.go:7:9: y declared but not used
./variable.go:7:12: z declared but not used
./variable.go:10:2: number declared but not used
./variable.go:11:2: name declared but not used
./variable.go:11:2: too many errors
```



# Constants

**const <constantName> = <value>**



# Data Types

Boolean (true, false)

Numerical (int, uint)

String (use UTF-8)

Error

Data structures (array, slice, map)



# Numerical

uint8	the set of all unsigned 8-bit integers (0 to 255)
uint16	the set of all unsigned 16-bit integers (0 to 65535)
uint32	the set of all unsigned 32-bit integers (0 to 4294967295)
uint64	the set of all unsigned 64-bit integers (0 to 18446744073709551615)
int8	the set of all signed 8-bit integers (-128 to 127)
int16	the set of all signed 16-bit integers (-32768 to 32767)
int32	the set of all signed 32-bit integers (-2147483648 to 2147483647)
int64	the set of all signed 64-bit integers (-9223372036854775808 to 9223372036854775807)
float32	the set of all IEEE-754 32-bit floating-point numbers
float64	the set of all IEEE-754 64-bit floating-point numbers
complex64	the set of all complex numbers with float32 real and imaginary parts
complex128	the set of all complex numbers with float64 real and imaginary parts
byte	alias for uint8
rune	alias for int32

[https://golang.org/ref/spec#Numeric\\_types](https://golang.org/ref/spec#Numeric_types)



# String

**Using double quotes for single line  
Using backticks for multi-line**



# Working with String

```
package main

import "fmt"

func main() {
    name := "Hello"

    // Convert string to []byte type
    tmp := []byte(name)
    fmt.Println(tmp[0])

    // Convert to string
    s := string(tmp[0])
    fmt.Println(s)
    fmt.Println(s + name[1:])

}
```



# Error types

Go has **error type** to dealing with errors  
Use from package errors

<https://golang.org/pkg/errors/>



# Error types

```
package main

import (
    "errors"
    "fmt"
)

func main() {
    err := errors.New("Normal error")
    if err != nil {
        fmt.Println(err)
    }
}
```



# Data Structures



# Arrays



# Arrays

Fixed length data structure, can't change



# Working with Arrays

```
func main() {
    var numbers [5]int
    numbers[0] = 1
    numbers[1] = 2

    var colors = [2]string{"Red", "Blue"}
    for i:=0; i< len(colors); i++ {
        fmt.Println(colors[i])
    }
}
```



# Using “...” or ellipsis

```
func main() {  
  
    var colors = [...]string{"Red", "Blue"}  
  
    for i := 0; i < len(colors); i++ {  
        fmt.Println(colors[i])  
    }  
}
```



# Array is of value type !!

```
func main() {
    var color1 = [2]string{"Red", "Blue"}
    var color2 = [...]string{"Red", "Blue"}

    color3 := color1
    color3[0] = "New Red"

    fmt.Println(color1)
    fmt.Println(color2)
    fmt.Println(color3)

    fmt.Println(color1 == color2)
    fmt.Println(color1 == color3)
}
```

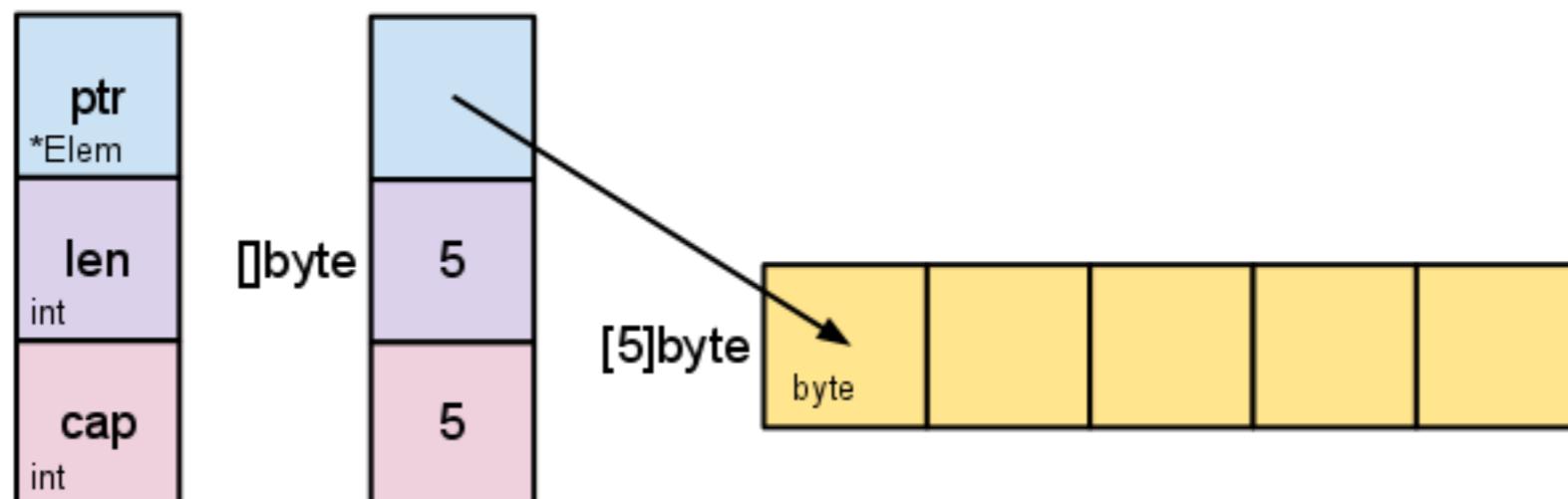


# Slices



# Slice

More powerful, flexible than an array  
Lightweight data structure  
Dynamic size



# Working with slice

```
func main() {
    numbers := []int{1, 2, 3, 4, 5}
    var s []int = numbers[1:3]
    fmt.Println(s)

    names := make([]string, 2)
    names[0] = "n1"
    names[1] = "n2"
    names[2] = "Error" // Runtime error

    names = append(names, "n3")

    fmt.Println(names)
}
```

<https://golang.org/pkg/builtin/#append>



# Slice is reference to array !!

Slices allow for multiple "views" of the same underlying array

```
func main() {
    numbers := [5]int{1, 2, 3, 4, 5}

    var s1 []int = numbers[1:3]
    var s2 []int = numbers[2:4]

    fmt.Println(numbers)
    fmt.Println(s1)
    fmt.Println(s2)

    s2[0] = 333

    fmt.Println(numbers)
    fmt.Println(s1)
    fmt.Println(s2)
}
```



# Maps



# Map

Data structure that allow for the storage and management of key/value pair data



# Map

**map[keyType]valueType**

```
func main() {
    var numbers map[string] int
    numbers = make(map[string] int)

    numbers["one"] = 1
    numbers["two"] = 2
    numbers["three"] = 3

    fmt.Println(numbers)
}
```



# Working with Map

## Insert, Update, Get, Check

```
func main() {
    numbers := make(map[string]int)

    numbers["one"] = 1
    e1 := numbers["one"]
    e2, ok := numbers["two"]

    fmt.Println(numbers)
    fmt.Println(e1, e2, ok)

    delete(numbers, "one")
    numbers["two"] = 2
    fmt.Println(numbers)
}
```



# Control Flows



# Control statements

If-else

Goto

For

**Switch (not required break)**



# Control statements

Control Flow in Go is similar to most C family languages

## Switch-case



# If with initialize value

```
func main() {  
  
    if score := 10; score > 10 {  
        fmt.Println("Case 1")  
    } else {  
        fmt.Println("Case 2")  
    }  
  
}
```



# For loop

Most powerful control logic in Go

```
func main() {  
    sum := 0  
    for i := 0; i < 100; i++ {  
        sum += i  
    }  
  
    sum = 1  
    for sum < 100 {  
        sum += sum  
    }  
}
```



# For loop with range

```
func main() {
    var numbers = []int{100, 200, 300, 400, 500}
    for i, v := range numbers {
        fmt.Printf("%d => %d\n", i, v)
    }
}
```



# Switch-case

Readable more than if-else

```
func main() {
    input := 5
    switch input {
        case 1:
            fmt.Println("Case 1")
        case 2, 3, 5:
            fmt.Println("Case 2")
            fallthrough
        case 4:
            fmt.Println("Case 3")
        default:
            fmt.Println("Default")
    }
}
```



# Switch-case with no condition

Readable more than if-else

```
func main() {
    score := 65
    switch {
        case score > 80:
            fmt.Println("Grade A")
        case score > 70:
            fmt.Println("Grade B")
        case score > 60:
            fmt.Println("Grade C")
        default:
            fmt.Println("Grade D")
    }
}
```



# Struct



# Struct

## Type of containers of properties/fields

```
type person struct {
    name string
    age  int
}

func main() {
    p1 := person{}
    p2 := person{"your name", 20}
    p3 := person{age: 20}
    p4 := &person{"your name", 20}

    fmt.Println(p1, p2, p3, p4)
}
```



# Anonymous struct

Type of containers of properties/fields

```
// Declare a variable of an anonymous type
var e1 struct {
    flag    bool
    counter int16
    pi      float32
}
```

```
// Declare a variable of an anonymous type and init
e2 := struct {
    flag    bool
    counter int16
    pi      float32
} {
    flag:    true,
    counter: 10,
    pi:      3.141592,
}
```



**Anonymous struct  
can be assigned to variables of  
a named struct type  
when they are identical**



# Padding in struct

Reduce memory usage

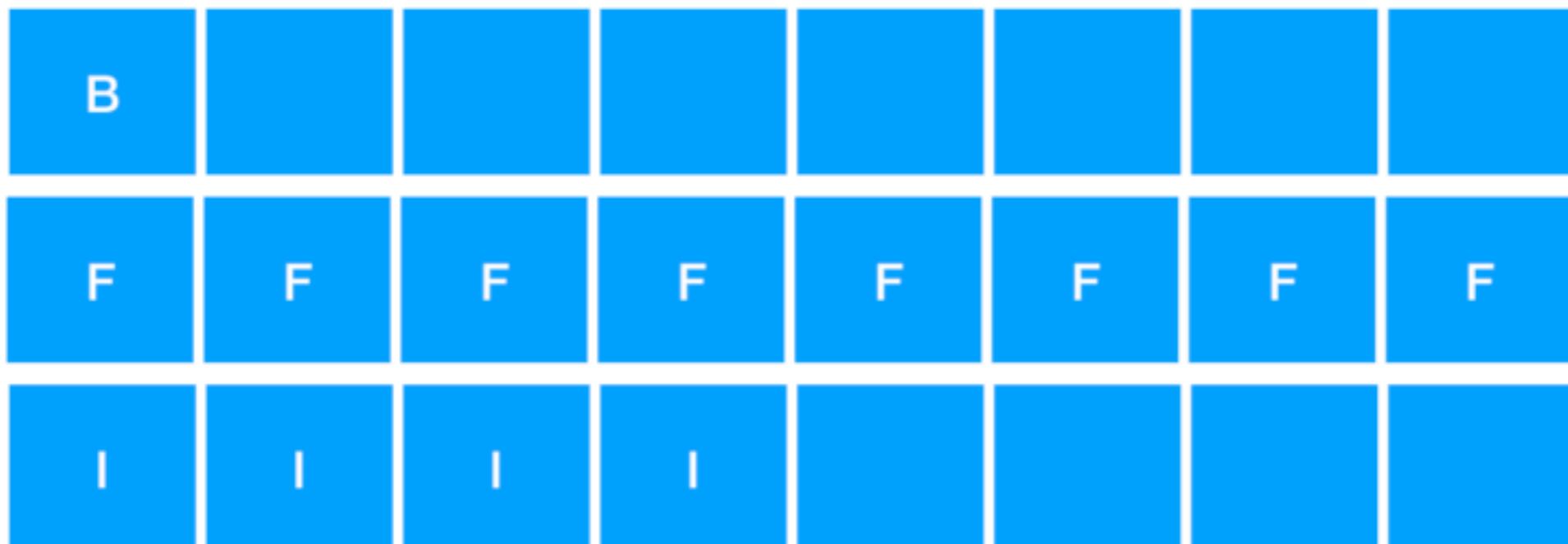
```
type first struct {
    b bool      // 1 byte
    f float64   // 8 bytes
    i int32     // 4 bytes
}
a := first{}

fmt.Println(unsafe.Sizeof(a)) // 24 bytes
```



# Padding in struct

Reduce memory usage



# Padding in struct

Reduce memory usage

```
type first struct {
    f float64 // 8 bytes
    b bool     // 1 byte
    i int32    // 4 bytes
}
a := first{}

fmt.Println(unsafe.Sizeof(a)) // 16 bytes
```



# Padding in struct

Reduce memory usage



# Embedded fields in Struct

```
type person struct {
    name string
    age  int
}

type special struct {
    person
    email string
}

func main() {
    p1 := special{person{}, "xxx.com"}
    fmt.Println(p1)
}
```



# Pointers



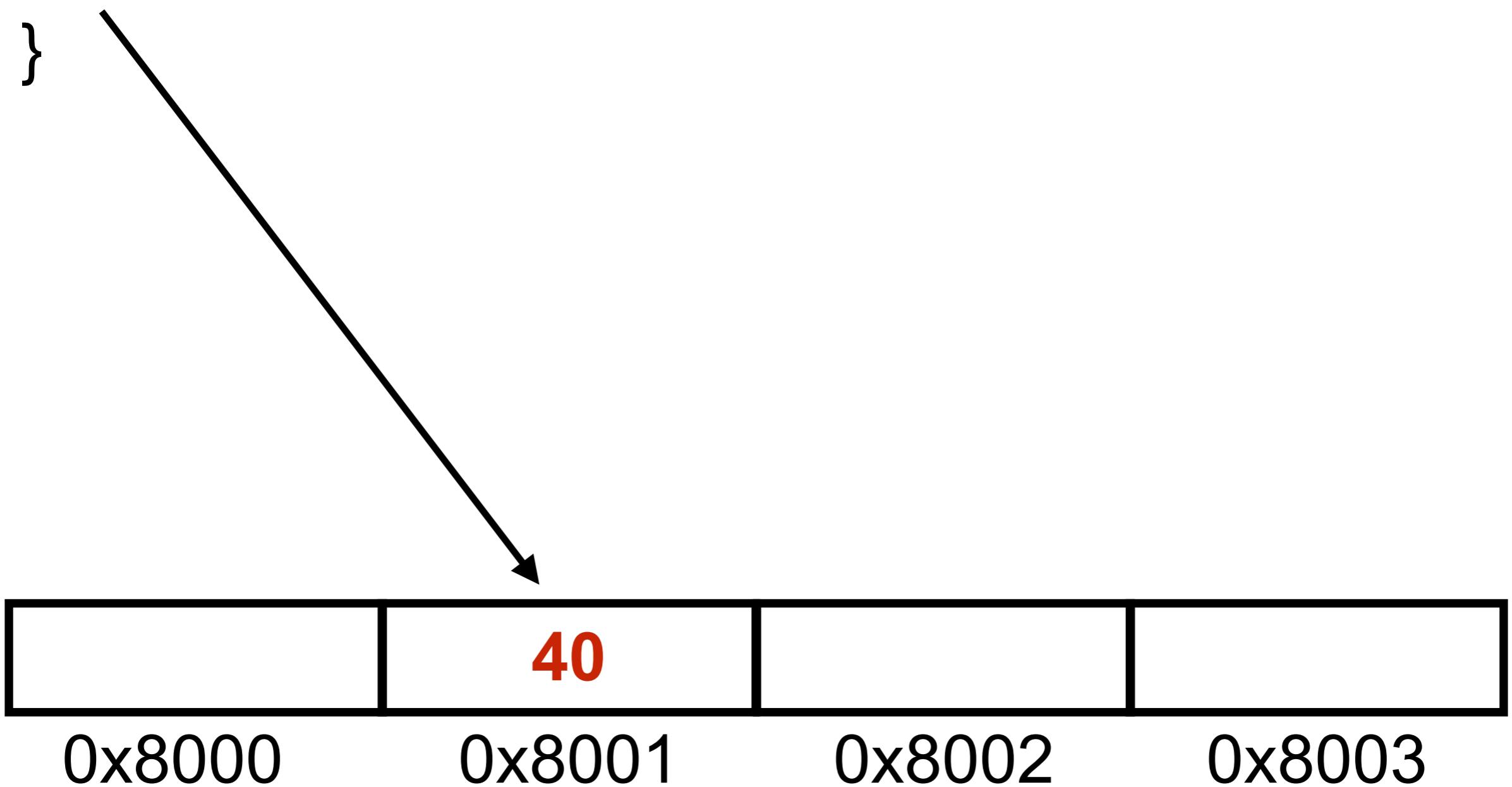
# Pointers

Pointers provide a way to **share data** across program boundaries

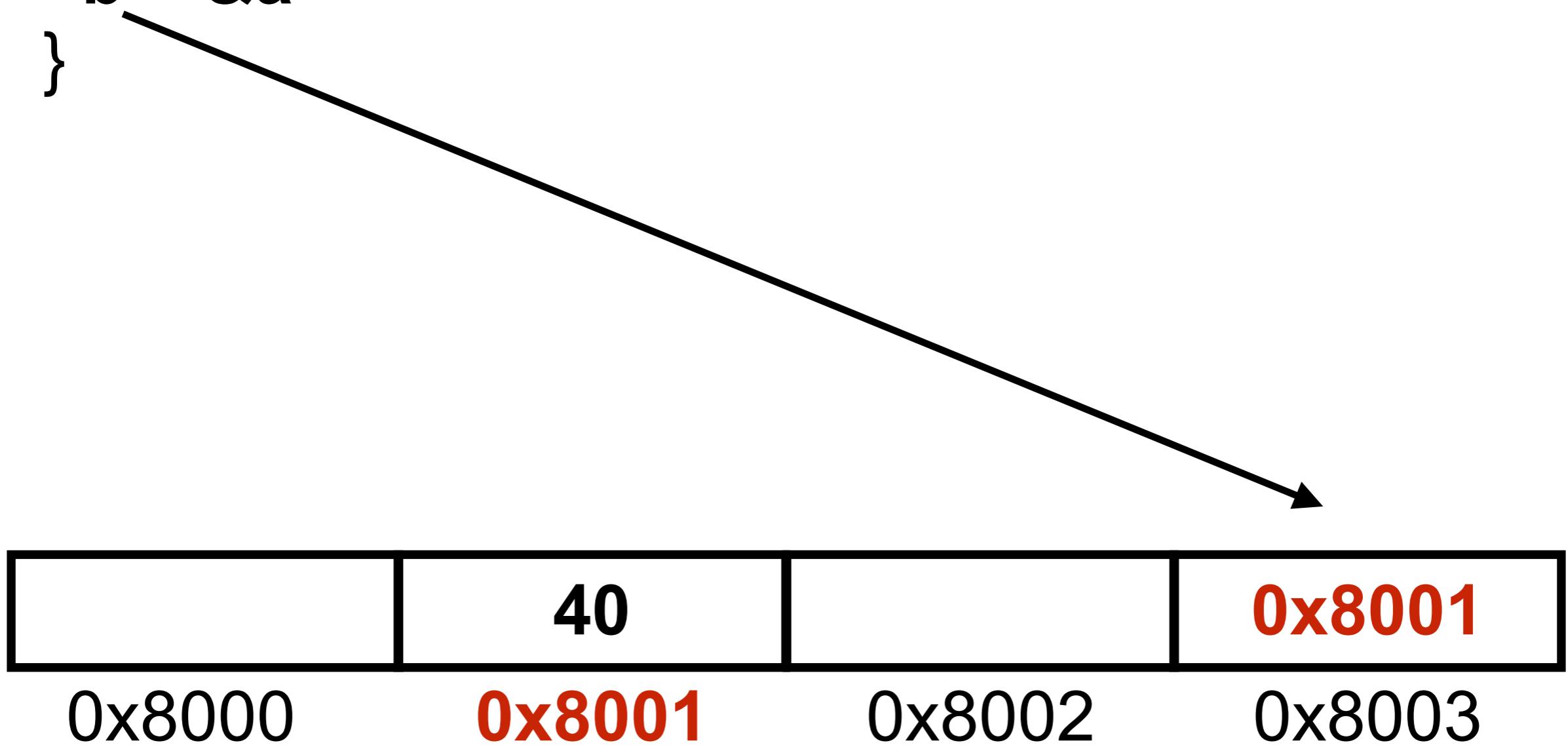
There is **only one copy** of the data and everyone can see it changing



```
func main() {  
    a := 40  
}
```



```
func main() {  
    a := 40  
    var b *int  
    b = &a  
}
```



# Pass by value

Values in Go are always **pass by value**

```
func main() {
    count := 10

    // Pass the "value of" the count.
    increment(count)

    println("count:\tValue Of[", count, "]\tAddr Of[", &count, "]")
}

func increment(inc int) {
    // Increment the "value of" inc.
    inc++
    println("inc:\tValue Of[", inc, "]\tAddr Of[", &inc, "]")
}
```



# Share data

## Using a pointer to share data

```
func main() {
    count := 10
    // Pass the "address of" count.
    increment(&count)
    println("count:\tValue Of[", count, "]\t\tAddr Of[", &count, "]")
}

func increment(inc *int) {
    // Increment the "value of" count that the "pointer points to".
    *inc++
}
```



# Share data

## Using a pointer to share data

```
func main() {
    count := 10

    // Pass the "address of" count.
    increment(&count)

    println("count:\tValue Of[" , count, "] \t\tAddr Of[" , &count, "]")
}
```

```
func increment(inc *int) {
    // Increment the "value of" count that the "pointer points to".
    *inc++
}
```



# Function



# Function

Functions are at the core of the language  
Group and organise our code to separate off functionality



# Function

## Use keyword **func**

```
func funcName(input1 type1, input2 type2) (output1 type1, output2 type2) {  
    // function body  
    // multi-value return  
    return value1, value2  
}
```



# Function

```
func main() {
    result := add(1, 2)
    fmt.Println(result)
}

func add(a int, b int) int {
    return a + b
}
```



# Return multiple values

```
func main() {
    result, err := divide(10, 0)
    if err != nil {
        fmt.Println(err)
    } else {
        fmt.Println(result)
    }
}

func divide(a int, b int) (int, error) {
    if b <= 0 {
        return 0, fmt.Errorf("Invalid input")
    }
    return a / b, nil
}
```



# Variadic functions

Function with a variable number of arguments

```
func main() {
    print("N1", "N2", "N3")
}

func print(args ...string) {
    for _, val := range args {
        fmt.Printf("Data with %s\n", val)
    }
}
```



# Anonymous function/Closure

```
func main() {
    var n int

    // Declare an anonymous function and call it.
    func() {
        fmt.Println("Direct:", n)
    }()
}

// Declare an anonymous function and assign it to a variable.
f := func() {
    fmt.Println("Variable:", n)
}

// Call the anonymous function through the variable.
f()
}
```



# **More function !!**

## **Defer, Panic, Recover**





# Defer functions

Execute when end function

```
func main() {  
    defer fmt.Println("World")  
  
    fmt.Println("Hello")  
}
```



# Read file

```
func main() {  
  
    f, err := os.Open("input.txt")  
    if err != nil {  
        log.Fatal(err)  
    }  
  
    defer f.Close()  
  
    scanner := bufio.NewScanner(f)  
    for scanner.Scan() {  
        fmt.Println(scanner.Text())  
    }  
  
    if err := scanner.Err(); err != nil {  
        log.Fatal(err)  
    }  
}
```



# Read file

```
func main() {  
  
    f, err := os.Open("input.txt")  
    if err != nil {  
        log.Fatal(err)  
    }  
  
    defer f.Close()  
  
    scanner := bufio.NewScanner(f)  
    for scanner.Scan() {  
        fmt.Println(scanner.Text())  
    }  
  
    if err := scanner.Err(); err != nil {  
        log.Fatal(err)  
    }  
}
```

<https://golang.org/src/log/log.go?s=10156:10184#L320>



# os.Exit(1) !!!

```
// Fatal is equivalent to Print() followed by a call to os.Exit(1).
func Fatal(v ...interface{}) {
    std.Output(2, fmt.Sprint(v...))
    os.Exit(1)
}
```

```
// Fatalf is equivalent to Printf() followed by a call to os.Exit(1).
func Fatalf(format string, v ...interface{}) {
    std.Output(2, fmt.Sprintf(format, v...))
    os.Exit(1)
}
```

```
// Fatalln is equivalent to Println() followed by a call to os.Exit(1).
func Fatalln(v ...interface{}) {
    std.Output(2, fmt.Sprintln(v...))
    os.Exit(1)
}
```

<https://golang.org/src/log/log.go?s=10156:10184#L320>



# Panic

Similar to throwing an **exception** in Go  
Continue from **defer** function  
**Runtime error**



# Recover

Recover from a panic

Using build-in function **recover()**

Back to normal situation

Only call recover within a deferred function



# Example with panic

```
func main() {
    // Read data from file
    b, err := ioutil.ReadFile("try_panic.go")
    if err != nil {
        panic(err)
    }
    fmt.Println(string(b))
}
```



# Example recover

```
func panicHandler() {  
    err := recover()  
    if err == "some error" {  
        fmt.Println("Try to recover from panic")  
        debug.PrintStack()  
    }  
  
    func main() {  
        // Defer  
        defer panicHandler()  
  
        // Read data from file  
        b, err := ioutil.ReadFile("try_panic.go")  
        if err != nil {  
            panic("some error")  
        }  
  
        fmt.Println(string(b))  
    }  
}
```



# Object-Oriented in Go !!



# No class in Go

How to add then behaviour to **struct** ?



# No class in Go

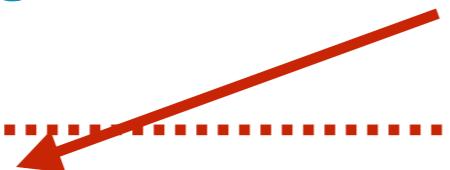
## How to add then behaviour to struct ?

```
type person struct {
    name string
    age  int
}

func (p person) say(message string) {
    fmt.Printf("Hi from %s with %s", p.name, message)
}

func main() {
    p := person{"pui", 20}
    p.say("called")
}
```

**Receiver argument (Value)**



# Working with pointer !!

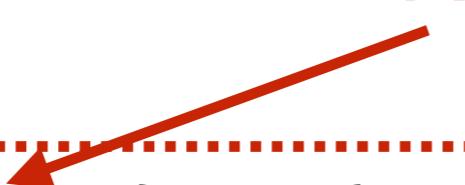
Try to update value

```
type person struct {
    name string
    age  int
}

func (p *person) say(message string) {
    p.age = 200
    fmt.Printf("Hi from %s with %s", p.name, message)
}

func main() {
    p := person{"pui", 20}
    p.say("called")
}
```

Receiver argument (Pointer)



# Value vs Pointer

[https://golang.org/doc/effective\\_go.html#methods](https://golang.org/doc/effective_go.html#methods)



# Example

```
type user struct {  
    name string  
    email string  
}
```

```
// notify implements a method with a value receiver.  
func (u user) notify() {  
    fmt.Printf("Sending User Email To %s<%s>\n",  
        u.name,  
        u.email)  
}
```

```
// changeEmail implements a method with a pointer receiver.  
func (u *user) changeEmail(email string) {  
    u.email = email  
}
```



# Builder pattern



# Method overriding

```
type person struct {
    name string
    age  int
}

type special struct {
    person
    email string
}

func (p person) say(message string) {
    fmt.Printf("Hi from %s with %s\n", p.name, message)
}

func main() {
    p1 := person{}
    p2 := special{person{}, "xxx.com"}

    p1.say("From person")
    p2.say("From special")
}
```



# Named Types Methods



# Named Types Methods

```
type duration int64
```

```
const (
    nanosecond duration = 1
    microsecond = 1000 * nanosecond
    millisecond = 1000 * microsecond
    second       = 1000 * millisecond
    minute      = 60 * second
    hour        = 60 * minute
)
```

```
// setHours sets the specified number of hours.
```

```
func (d *duration) setHours(h float64) {
    *d = duration(h) * hour
}
```

```
// hours returns the duration as a floating point number of hours.
```

```
func (d duration) hours() float64 {
    hour := d / hour
    nsec := d % hour
    return float64(hour) + float64(nsec)*(1e-9/60/60)
}
```



# Working with JSON

<https://golang.org/pkg/encoding/json/>



# Working with JSON



<https://golang.org/pkg/encoding/json/>



# JSON message

```
{  
    "userId": 0,  
    "id": 100,  
    "title": "",  
    "body": ""  
}
```



# Create struct

Go code

```
import (
    "encoding/json"
    "fmt"
)

type Post struct {
    UserID int `json:"userId"`
    ID     int `json:"id"`
    Title  string `json:"title"`
    Body   string `json:"body"`
}
```

JSON

```
{  
    "userId": 0,  
    "id": 100,  
    "title": "",  
    "body": ""  
}
```

<https://golang.org/pkg/encoding/json/>



# Generate Struct from JSON

Visual Studio Code > Other > Paste JSON as Code

The screenshot shows the Visual Studio Marketplace page for the "quicktype" extension. The extension icon is a teal circle containing the letters "QT". The title is "Paste JSON as Code" and the developer name is "quicktype". It has 536,588 installs and a rating of 4.73/5 based on 26 reviews. The description says "Copy JSON, paste as Go, TypeScript, C#, C++ and more." There are "Install" and "Trouble Installing?" buttons. Below the main section, there are tabs for "Overview" (which is selected), "Q & A", and "Rating & Review". At the bottom, it shows "Visual Studio Marketplace v12.0.46", "installs 536571", and "rating 4.73/5 (26)".

Paste JSON as Code

quicktype | 536,588 installs | ★★★★★ (26) | Free

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

Install    [Trouble Installing?](#)

Overview    Q & A    Rating & Review

Visual Studio Marketplace v12.0.46    installs 536571    rating 4.73/5 (26)

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



Go programming

© 2017 - 2018 Siam Chamnankit Company Limited. All rights reserved.

# Struct to JSON

```
p1 := Post{ID: 100}
// b, err := json.Marshal(p1)
b, err := json.MarshalIndent(p1, "", "    ")
if err != nil {
    fmt.Println("Error ", err)
} else {
    fmt.Println(string(b))
}
```

<https://golang.org/pkg/encoding/json/>



# JSON to Struct

```
var out Post
err = json.Unmarshal(b, &out)
if err != nil {
    fmt.Println("Error ", err)
} else {
    fmt.Println(out)
}
```

<https://golang.org/pkg/encoding/json/>



# Testing



# Testing in Go

Build-in testing framework

Using **testing** package

\$go test

<https://golang.org/pkg/testing/>



# Testing package

## Testing Benchmark

<https://golang.org/pkg/testing/>



# Hello testing

## hello\_test.go

```
package main

import(
    "testing"
)

func TestHello(t *testing.T) {
    expectedResult := "Hello my first testing"
    result := hello()
    if result != expectedResult {
        t.Fatalf("Expected %s but got %s", expectedResult, result)
    }
}
```



# System under test

## hello.go

```
package main

func hello() string {
    return "Hello my first testing"
}
```



# Run test

\$go test

\$go test -v

\$go test -v -run <test name>

\$go test ./...



# **\*testing.T ?**

Used for error reporting

**t.Error**

**t.Fatal**

**t.Log**



# **\*testing.T ?**

Enable parallel testing

**t.Parallel()**



# \*testing.T ?

To control a test run

**t.Skip()**



# Table/data driven test

Working with data driven testing

Operand 1	Operand 2	Expected result
1	2	3
5	10	15
10	-5	5



# Table structure

```
func TestAdd(t *testing.T) {  
  
    var dataTests = []struct{  
        op1 int  
        op2 int  
        expectedResult int  
    }{  
        {1, 2, 3},  
        {5, 10, 15},  
        {10, -5, 5},  
    }  
}
```



# Testing

```
func TestAdd(t *testing.T) {  
    ...  
  
    for _, test := range dataTests{  
        result := add(test.op1, test.op2)  
        if result != test.expectedResult {  
            t.Fatalf("Expected %d but got %d",  
                    test.expectedResult, result)  
        }  
    }  
}
```



# Test/code coverage

Go tool can report test coverage statistic

```
$go test -cover
```



# Generate coverage report

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

```
/Users/somkiat/data/slide/golang/go2020/demo/testing/hello.go (100.0%) ▾ not tracked not covered covered  
  
package main  
  
func hello() string {  
    return "Hello my first testing"  
}
```



# Benchmark



# Write first benchmark

\$go test -bench=.

```
package main

import "testing"

func BenchmarkFib(b *testing.B) {
    for n := 0; n < b.N; n++ {
        Fib(n)
    }
}

func Fib(n int) int {
    if n < 2 {
        return n
    }
    return Fib(n-1) + Fib(n-2)
}
```



# Run benchmark

\$go test -bench=.

\$go test -bench=. -run=<test name>



# Benchmark Struct to JSON



# Solution 1

## Use json.Marshal

```
func structToJson01() {
    p1 := Post{ID: 100}
    _, err := json.Marshal(p1)
    if err != nil {
        fmt.Println("Error ", err)
    }
}
```



# Solution 2

## Use json.Encoder

```
func structToJson02() {  
    p1 := Post{ID: 100}  
    var buffer bytes.Buffer  
    json.NewEncoder(&buffer).Encode(&p1)  
}
```



# Create benchmark

```
func BenchmarkSolution01(b *testing.B) {
    for i := 0; i < b.N; i++ {
        structToJson01()
    }
}

func BenchmarkSolution02(b *testing.B) {
    for i := 0; i < b.N; i++ {
        structToJson02()
    }
}
```



# Run benchmark

go test -bench=.

goos: darwin		
goarch: amd64		
BenchmarkSolution01-8	4019841	309 ns/op
BenchmarkSolution02-8	3472522	352 ns/op



# Workshop with JSON

Read data from JSON file

[https://raw.githubusercontent.com/up1/course-go-2021/main/  
demo/json/read\\_from\\_file/data.json](https://raw.githubusercontent.com/up1/course-go-2021/main/demo/json/read_from_file/data.json)



# Interface

[https://golang.org/doc/effective\\_go.html#interfaces\\_and\\_types](https://golang.org/doc/effective_go.html#interfaces_and_types)



# Interface

Collection of method signatures

Way to specify behaviour for a type

Interfaces are implemented implicitly



# Stringer example

Interface from **fmt** package

```
type Stringer interface {
    String() string
}
```

<https://golang.org/pkg/fmt/#Stringer>



# Stringer example

Interface from **fmt** package

```
type Animal struct {
    Name string
    Age  uint
}

func (a Animal) String() string {
    return fmt.Sprintf("%v (%d)", a.Name, a.Age)
}
```

<https://golang.org/pkg/fmt/#Stringer>



# Empty interface

Interface with zero methods

```
func describe(i interface{}) {  
    fmt.Printf("(%v, %T)\n", i, i)  
}
```

<https://golang.org/src/fmt/print.go?s=7925:7974#L263>



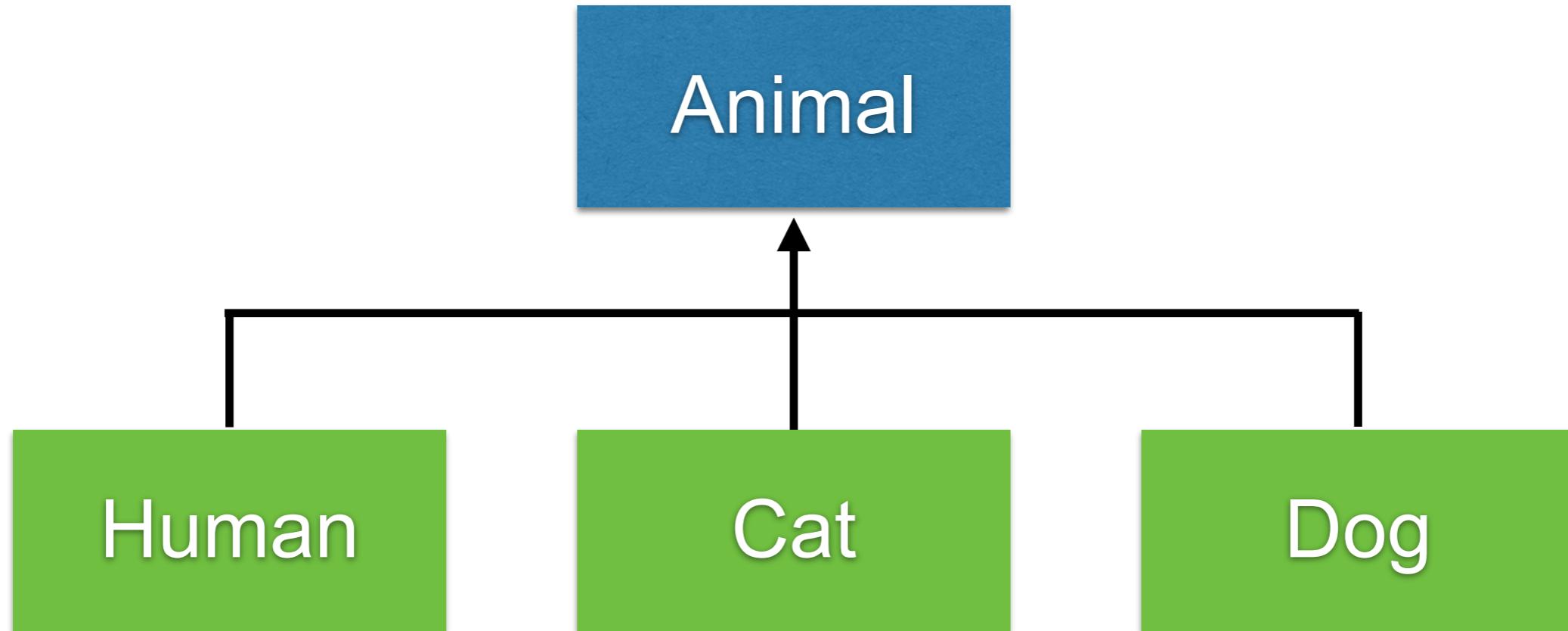
# Interface for polymorphism



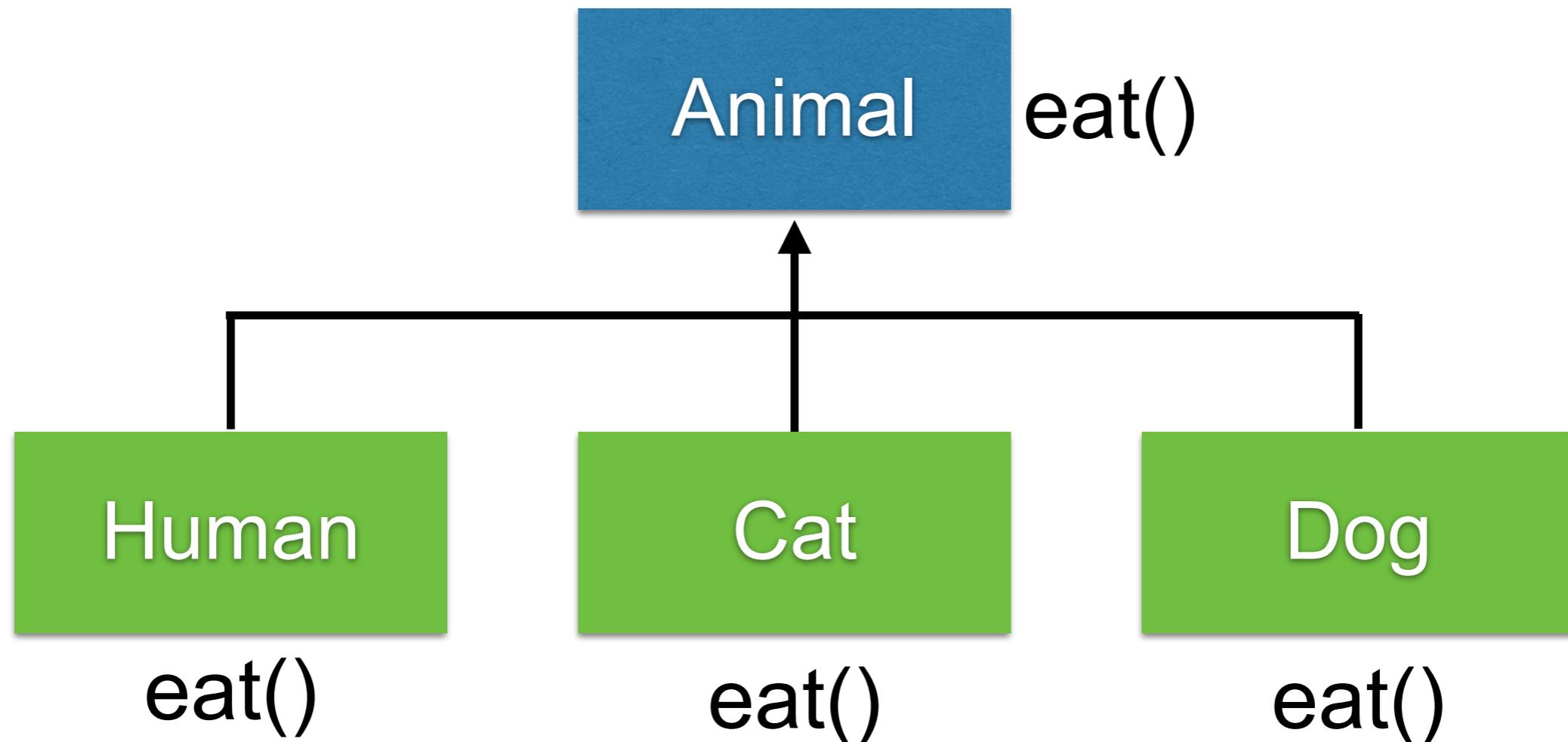
[https://en.wikipedia.org/wiki/Polymorphism\\_%28computer\\_science%29](https://en.wikipedia.org/wiki/Polymorphism_%28computer_science%29)



# Working with interface



# Working with interface



# Interface Animal

```
type animal interface {  
    eat()  
}
```

```
type human int  
type cat int  
type dog int
```



# Implement from interface

```
func (h human) eat() {  
    fmt.Println("Human eat")  
}  
  
func (c cat) eat() {  
    fmt.Println("Cat eat")  
}  
  
func (d dog) eat() {  
    fmt.Println("Dog eat")  
}
```



# Using like Polymorphism

```
func callEat(a animal) {  
    a.eat()  
}  
  
func main() {  
    var h human  
    var c cat  
    callEat(h)  
    callEat(c)  
}
```



# Using interface as parameter

```
func callEat2(i interface{}) {  
    i.(animal).eat()  
}
```

```
func main() {  
    var h human  
    var c cat  
    callEat2(h)  
    callEat2(c)  
}
```

Empty interface



# Using empty interface

```
func myPrintln(a interface{}) {  
  
    switch v := a.(type) {  
    case string:  
        fmt.Printf("Is string : type(%T) : value(%s)\n", v, v)  
    case int:  
        fmt.Printf("Is int      : type(%T) : value(%d)\n", v, v)  
    case float64:  
        fmt.Printf("Is float64 : type(%T) : value(%f)\n", v, v)  
    default:  
        fmt.Printf("Is unknown : type(%T) : value(%v)\n", v, v)  
    }  
  
}  
}
```



# Workshop with interface

[https://github.com/up1/course-go-2021/blob/main/  
demo/interface/main\\_problem.go](https://github.com/up1/course-go-2021/blob/main/demo/interface/main_problem.go)



# Error interface

## **type error**

The error built-in interface type is the conventional interface representing no error.

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

<https://golang.org/pkg/builtin/#error>



# Go Modules

<https://blog.golang.org/using-go-modules>



# Modules

Basis of dependency management  
Group of packages into single unit

*Set of dependencies and versioning*



# Semantic versioning

v<major>.<minor>.<patch>

<https://semver.org/>

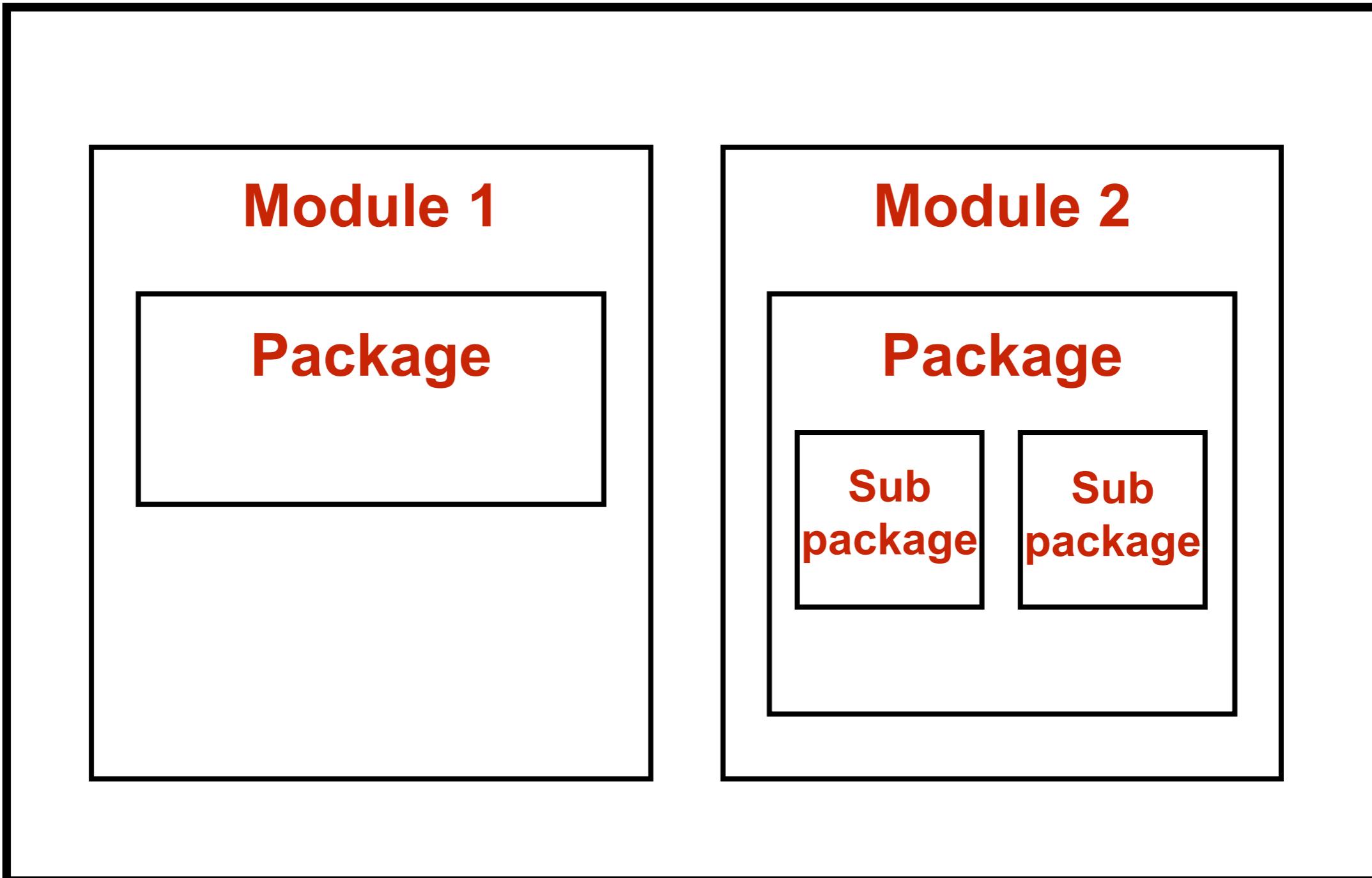


# Project

Main package



# Project



# Using a single module per repository



# Create a module

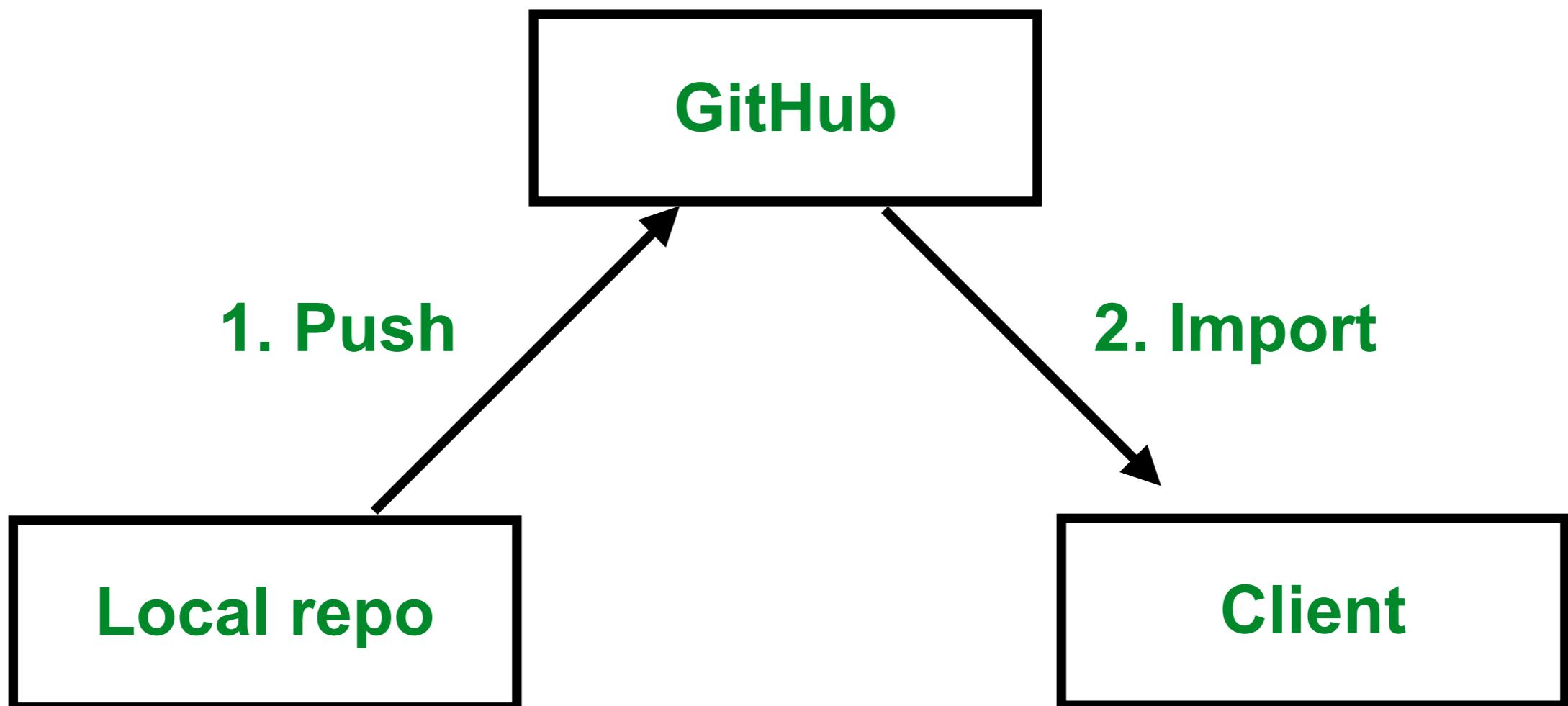
```
$go mod init <module name>
```

*go: creating new go.mod: module demo*



# Publish module to GitHub

```
$go mod init github.com/<user>/<repo>
```



# Go project structure



# Project structure

Flat structure

Layering

Modular, DDD

Clean architecture

Hexagonal architecture



# Flat

## Easy to start

```
/  
|   └── data.go  
|   └── handlers.go  
|   └── main.go  
|   └── model.go  
|   └── storage.go  
|   └── storage_json.go  
└── storage_mem.go
```



# Layering

## Grouping by function



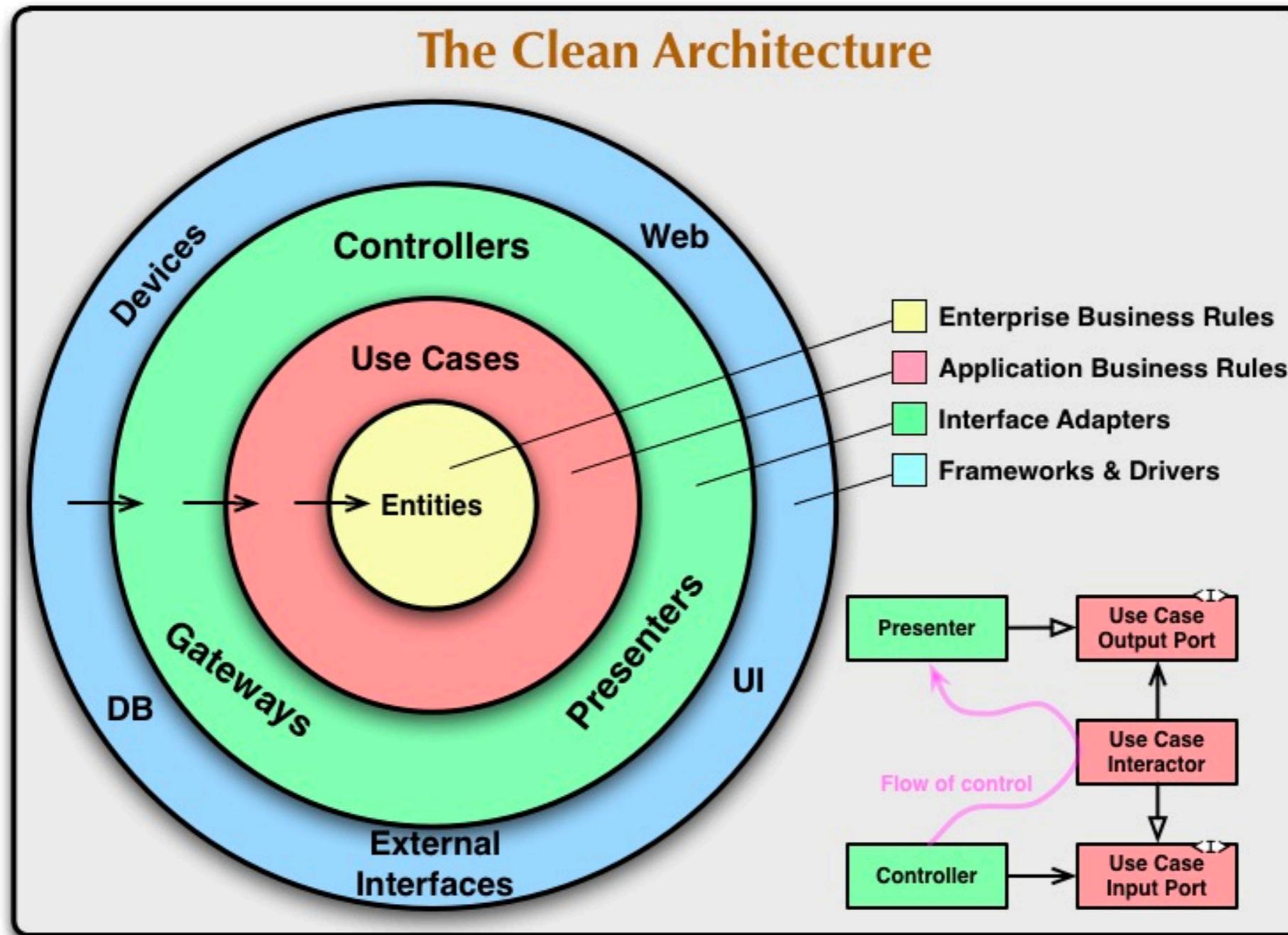
# Modular

## Grouping by logical/business

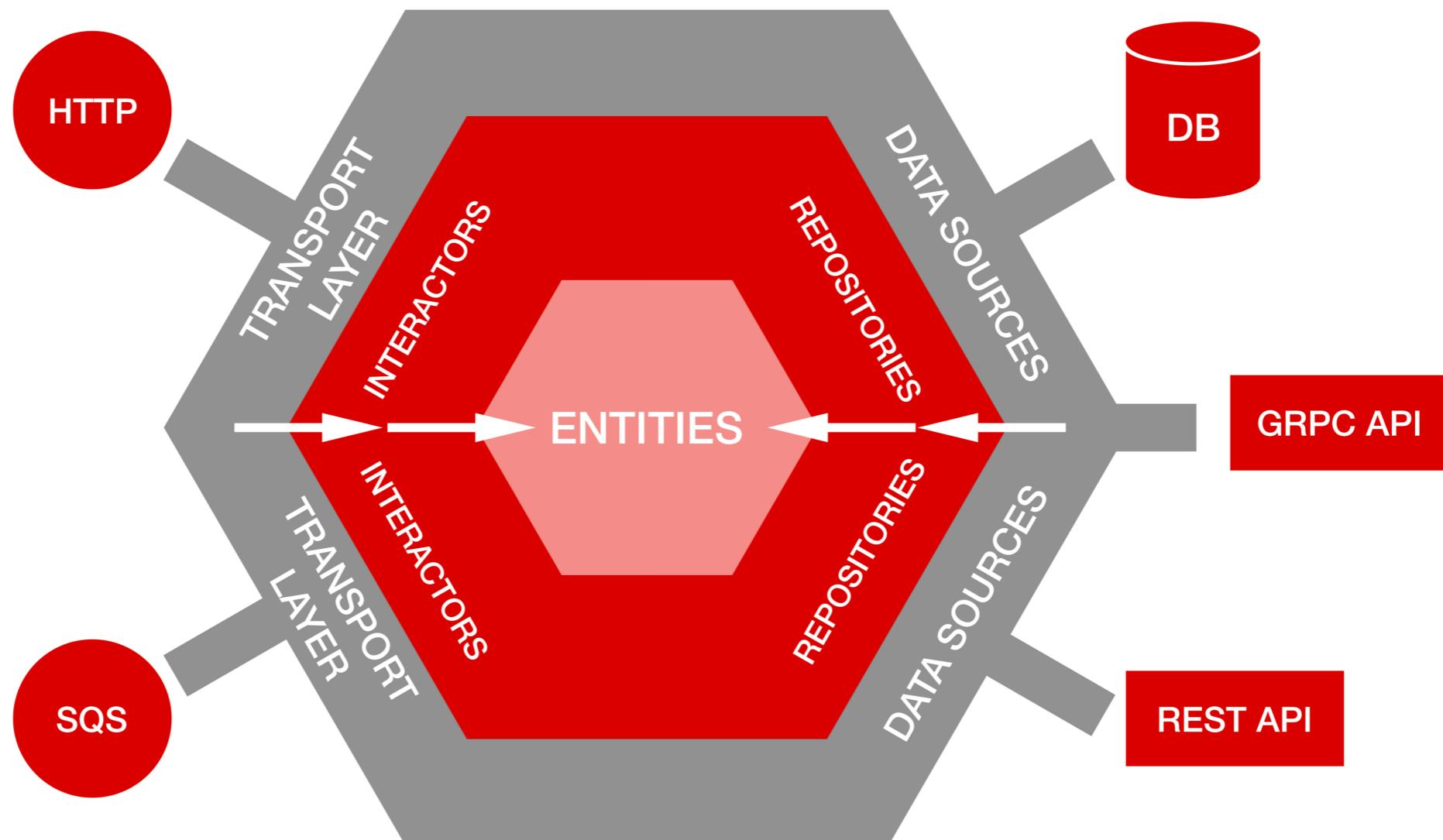
```
.  
  └── beers  
      ├── beer.go  
      └── handler.go  
  └── main.go  
  └── reviews  
      ├── handler.go  
      └── review.go  
  └── storage  
      ├── data.go  
      ├── json.go  
      ├── memory.go  
      └── storage.go
```



# Clean architecture



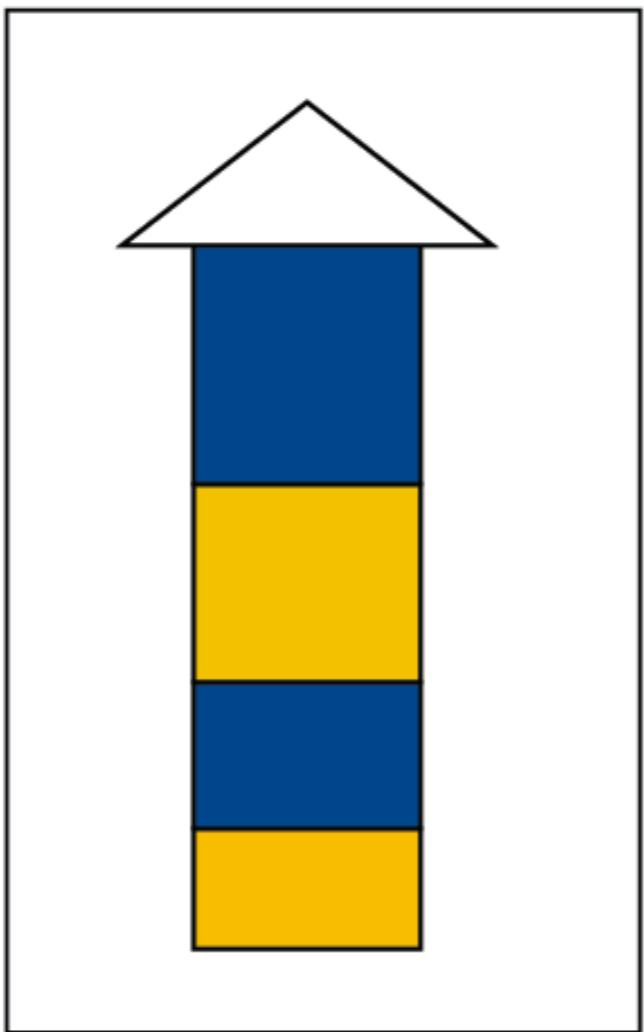
# Hexagonal architecture



# Go routine and channel

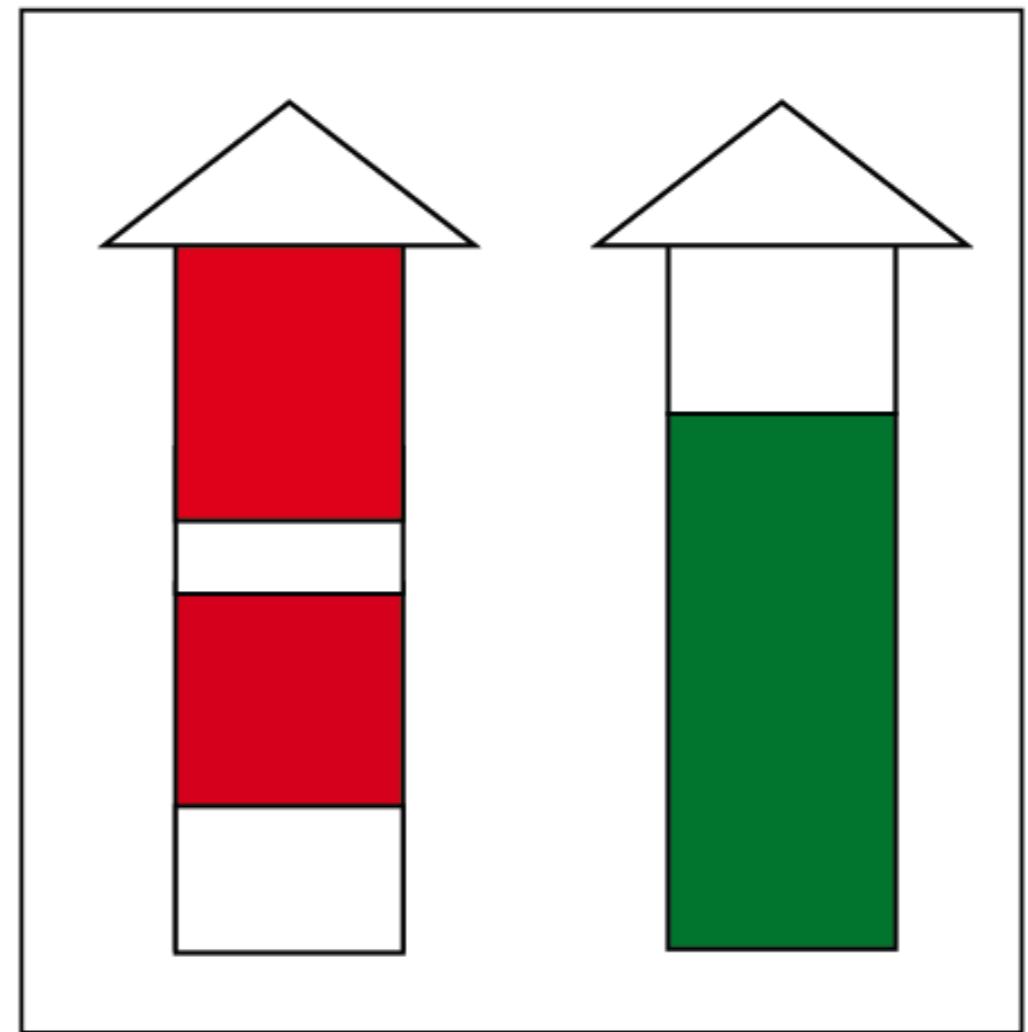


## Concurrency



Concurrency is about *dealing with*  
lots of things at once

## Parallelism



Parallelism is about *doing*  
lots of things at once



# Go routine

Independently executing function

Launch by a **go** statement

Very cheap

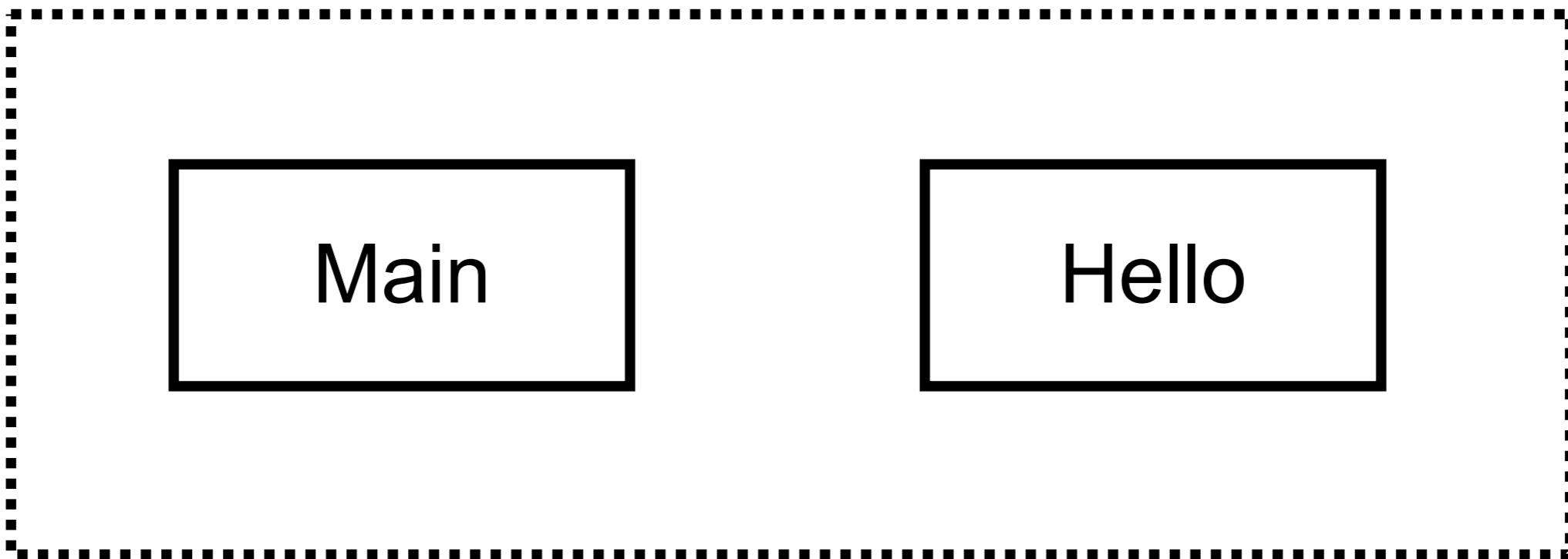
Not a thread

May be only a thread with 1,000 of goroutines



# Example

Independently executing function



# Example

```
func hello(name string) {
    for i := 0; ; i++ {
        fmt.Println(name, i)
        time.Sleep(time.Duration(rand.Intn(1e3)) * time.Millisecond)
    }
}

func main() {
    go hello("somkiat")
    fmt.Println("Start process")
    time.Sleep(2 * time.Second)
    fmt.Println("Finish process")
}
```



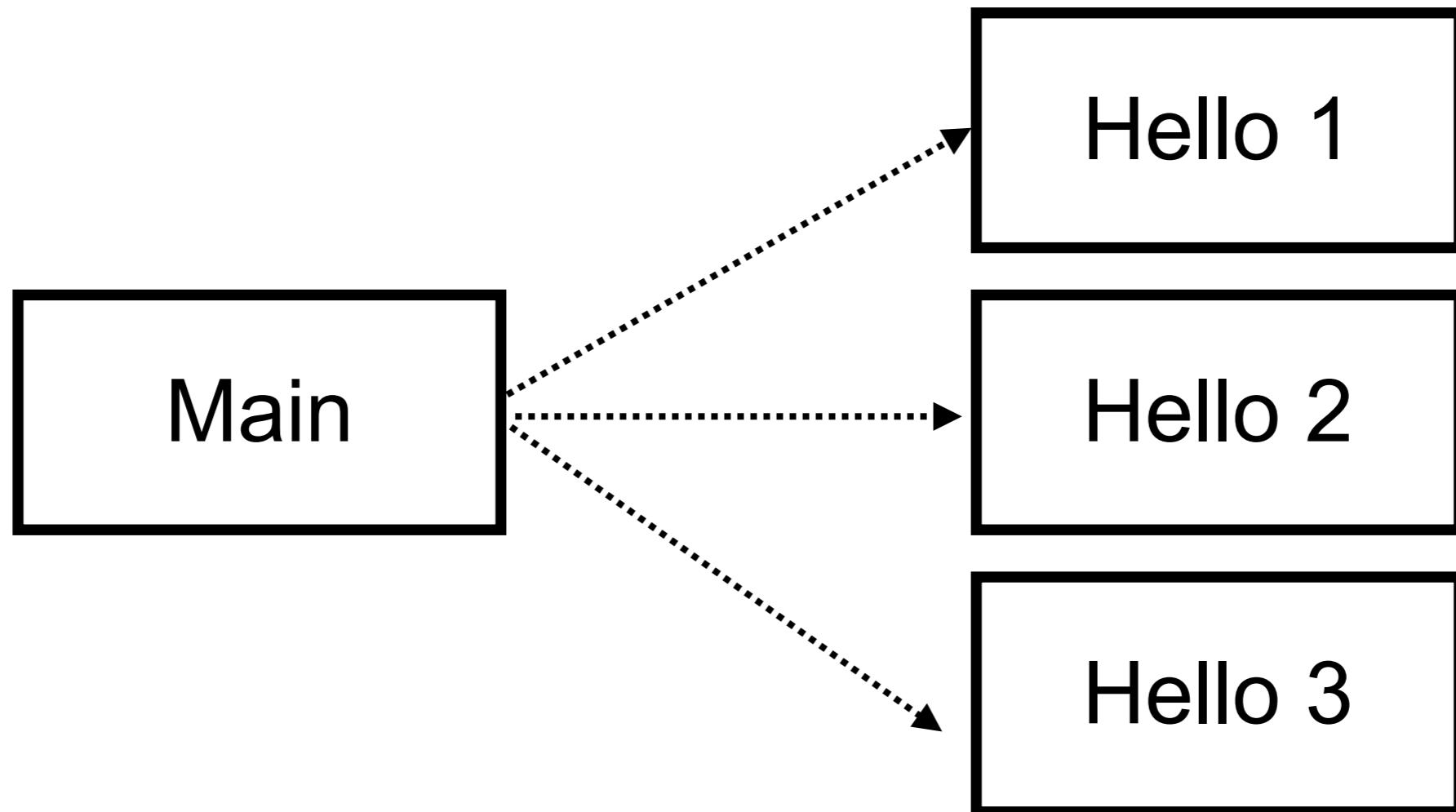
# Main function can't see the output from other go routine !!



# Need waiting !!



# Using sync.WaitGroup



# Using sync.WaitGroup

```
var wg sync.WaitGroup

func hello(name string) {
    defer wg.Done()

    fmt.Printf("Start with = %s\n", name)
    time.Sleep(time.Duration(rand.Intn(1e3)) * time.Millisecond)
    fmt.Printf("Processed with = %s\n", name)
}

func main() {
    fmt.Println("Start process")

    for i := 0; i < 5; i++ {
        wg.Add(1)
        go hello(fmt.Sprintf("task=%d", i))
    }
    wg.Wait()

    fmt.Println("Finish process")
}
```



# Need communication !!



# Channel

Provides a connection between 2 go routines  
Allow them to communicate



# Channel

## Create channel with `make()`

```
func hello(out chan< string) {
    time.Sleep(2 * time.Second)
    out <- "Called hello"
}

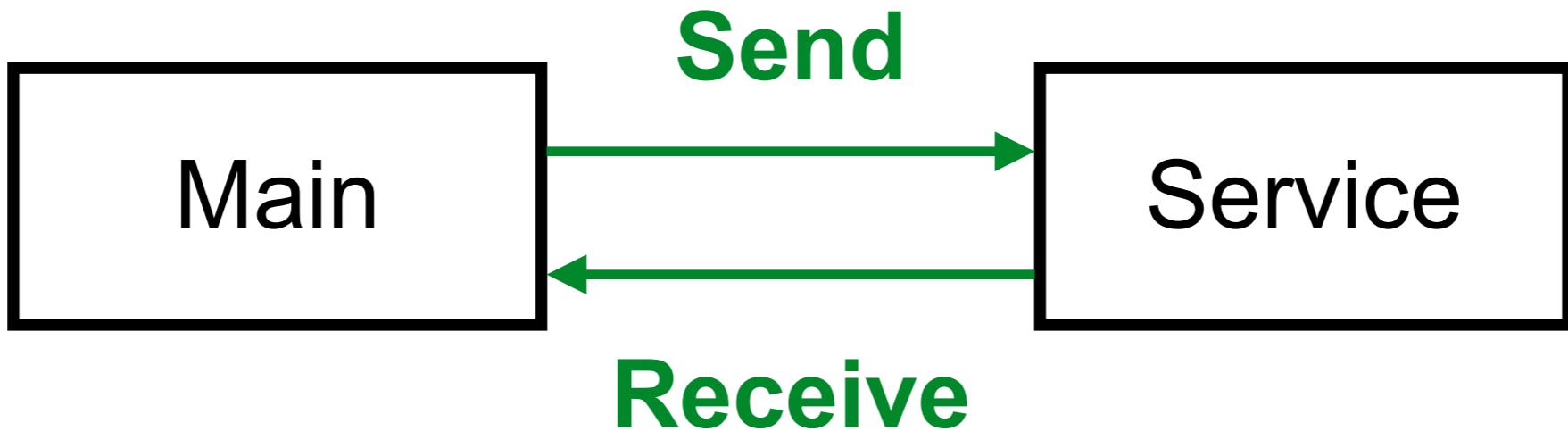
func main() {
    c := make(chan string)

    go hello(c)
    fmt.Println("Main")

    fmt.Println(<-c)
}
```



# Duplex of channel



# Example

```
func service(name string, jobs <chan int, results chan<- string) {
    for j := range jobs {
        fmt.Println("Worker", name, "started job", j)
        time.Sleep(time.Duration(rand.Intn(1e3)) * time.Millisecond)
        results <- fmt.Sprintf("Worker %s finished job %d", name, j)
    }
}
```

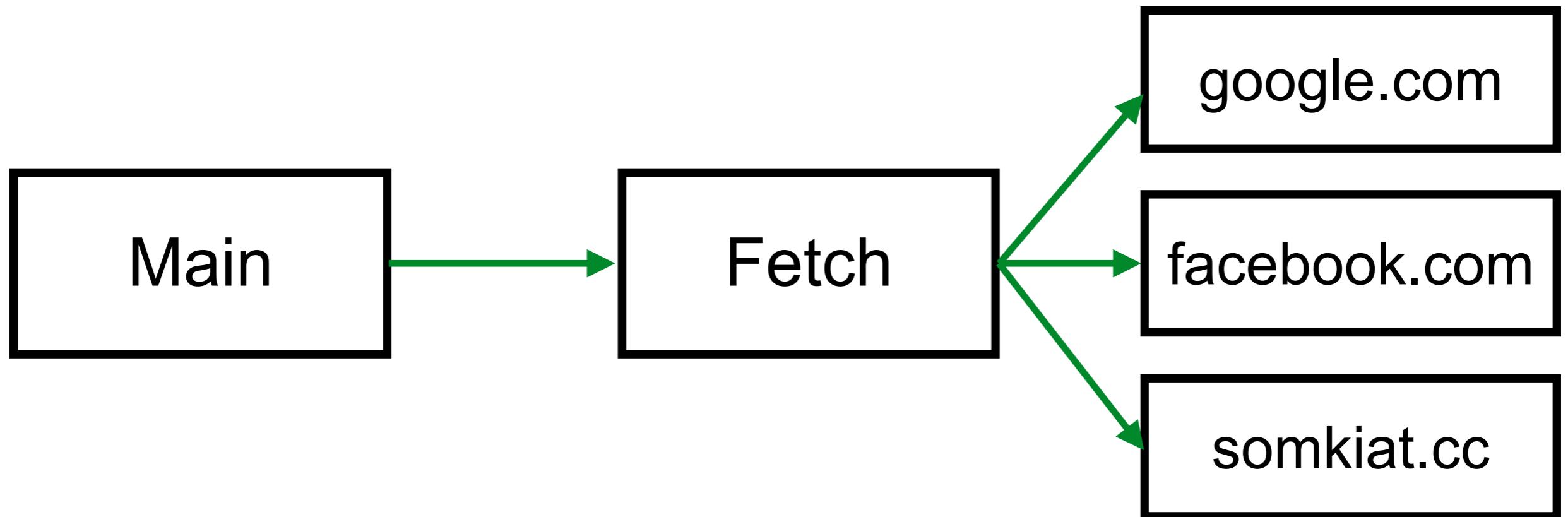


# Example

```
func main() {  
  
    jobs := make(chan int, 3)  
    results := make(chan string, 5)  
  
    for i := 0; i < 3; i++ {  
        name := fmt.Sprintf("name_%d", i)  
        go service(name, jobs, results)  
    }  
  
    for i := 0; i < 5; i++ {  
        jobs <- i  
    }  
    close(jobs)  
  
    for a := 1; a < 5; a++ {  
        fmt.Println(<results)  
    }  
}
```



# Workshop



# Race detector

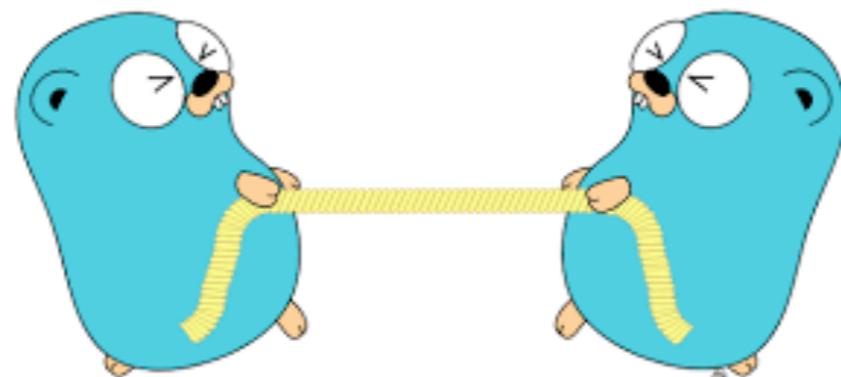


[https://en.wikipedia.org/wiki/Race\\_condition](https://en.wikipedia.org/wiki/Race_condition)



# Race Detector

to prevent simultaneous **read** and **write** access  
to the **same** variable or memory location



# Race Detector

Added to Go 1.1+

`$go run -race <package name>`

[https://golang.org/doc/articles/race\\_detector.html](https://golang.org/doc/articles/race_detector.html)



# Problem ?

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

func getNumber() int {
    var i int
    go func() {
        i = 5
    }()
    return i
}
```



# Run with race detector

go run -race problem.go

```
=====
WARNING: DATA RACE
Write at 0x00c00001c0b0 by goroutine 7:
  main.getNumber.func1()
```



# Problem ?

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

```
func getNumber() int {  
    var i int  
    go func() {  
        i = 5  
    }()  
  
    return i  
}
```

Write and read data !!



# Describe

Main go routine



# Describe

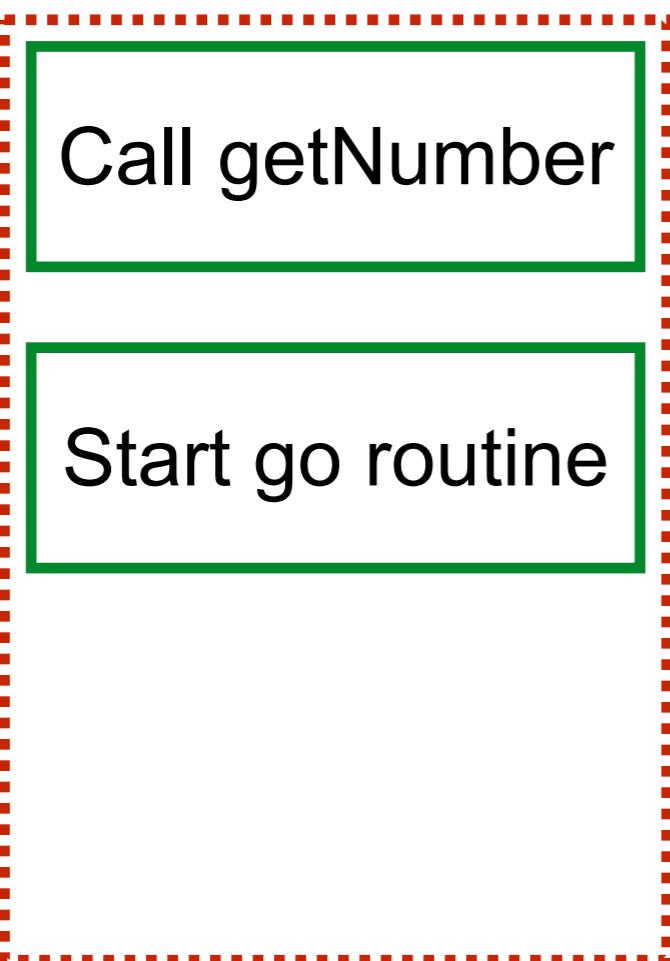
Main go routine

Call getNumber



# Describe

Main go routine

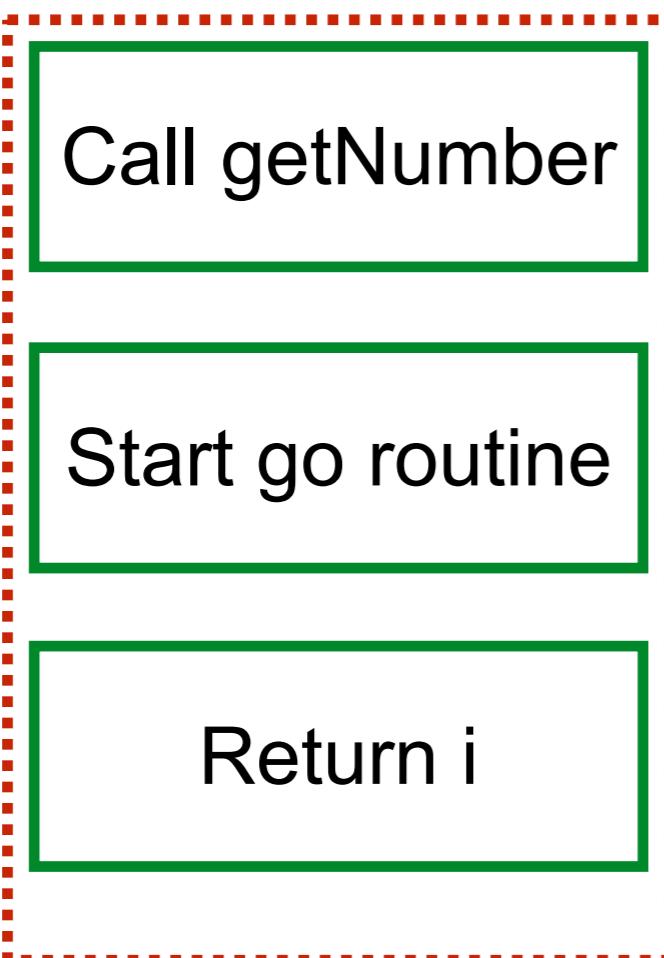


Second go routine

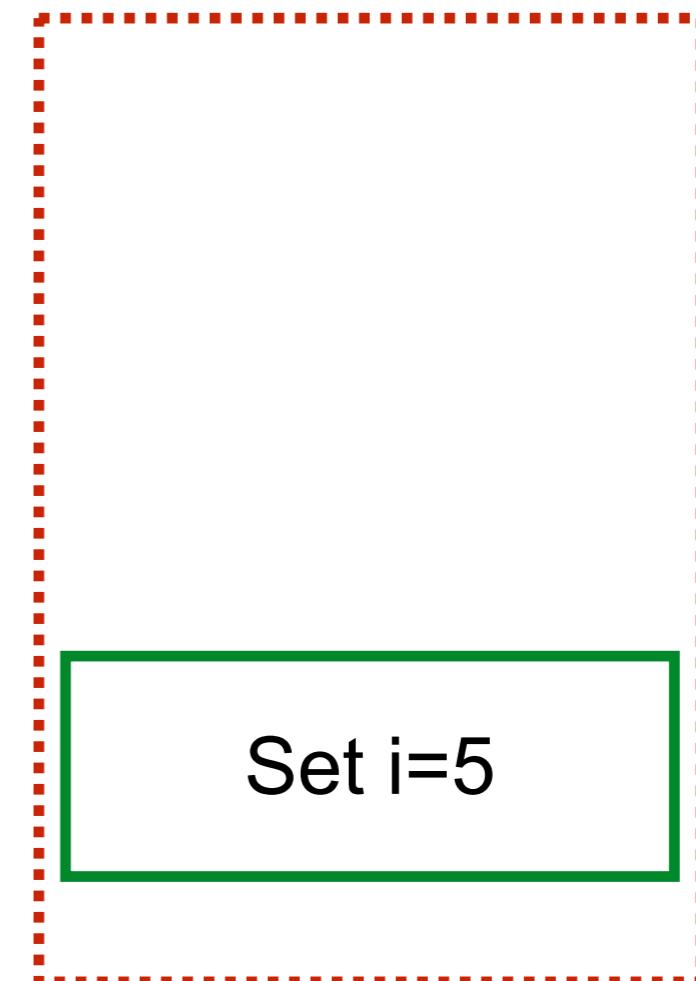


# Describe

## Main go routine

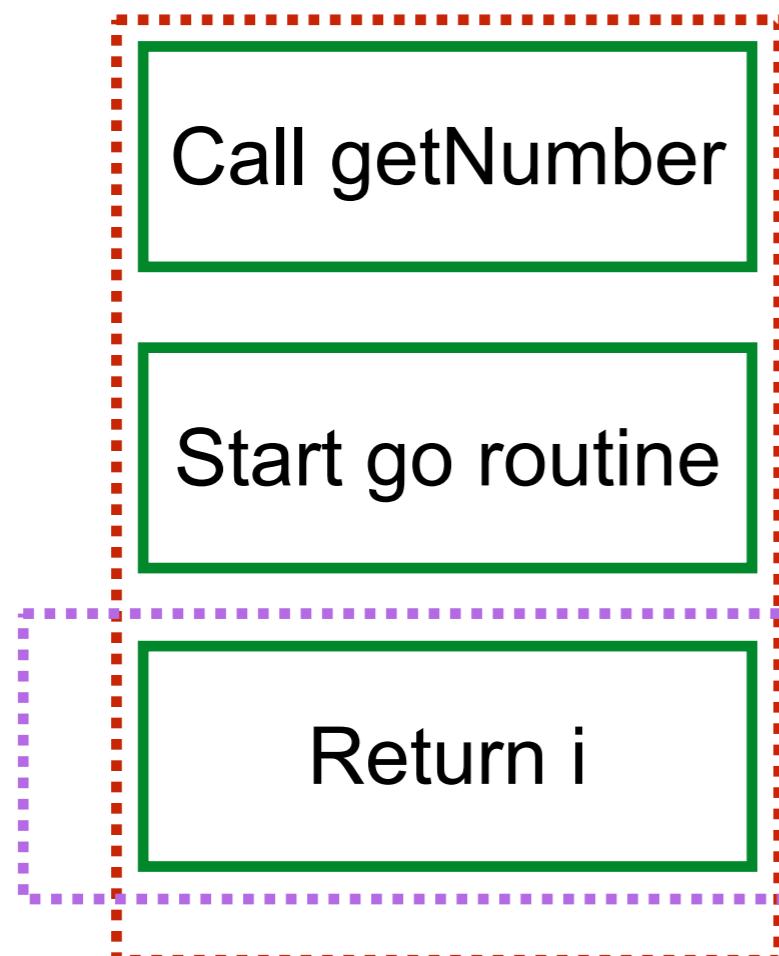


## Second go routine



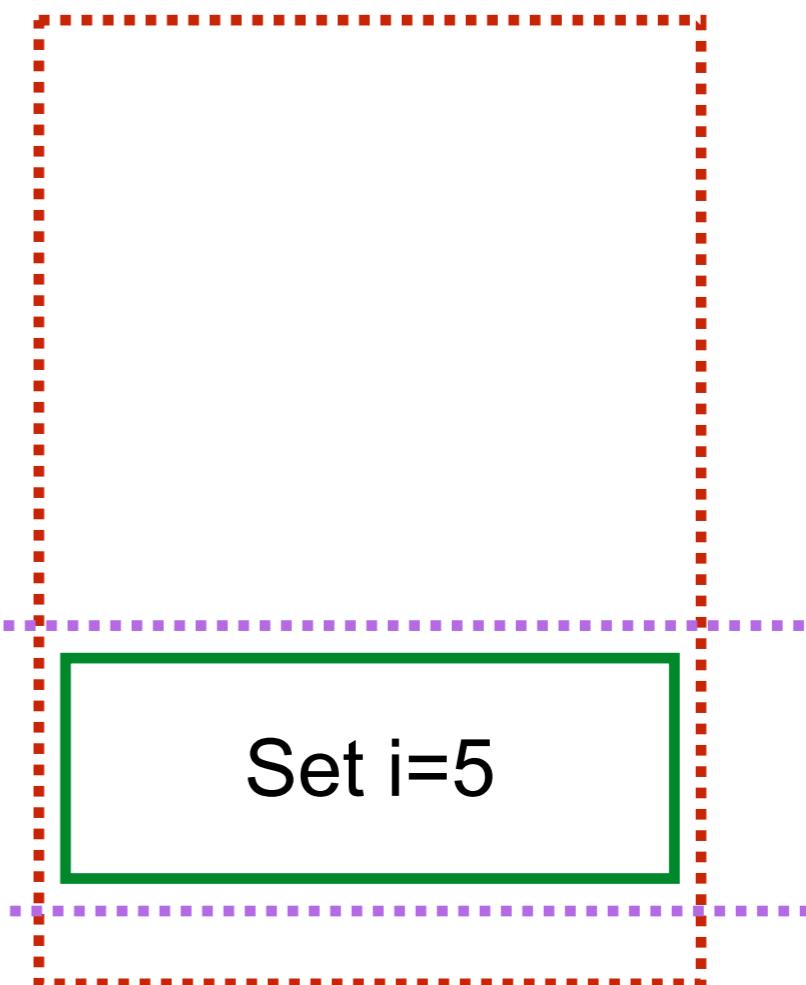
# Describe

Main go routine



Read data !!

Second go routine



Write data !!



# How to solve a problem ?



# Solutions

Blocking with waitGroup

Blocking with channel

Mutex

Atomic



# Wait Group

```
func getNumber() int {
    var i int
    var wg sync.WaitGroup
    wg.Add(1)

    go func() {
        i = 5
        wg.Done()
    }()

    wg.Wait()
    return i
}
```



# Channel

```
func getNumber() int {  
    var i int  
    done := make(chan struct{})  
  
    go func() {  
        i = 5  
        done <- struct{}{}  
    }()  
  
    <done  
    return i  
}
```



# Logging

<https://golang.org/pkg/log/>



# Logging

Logging is an important part of every program  
Working with standard library



# RESTful API with Go

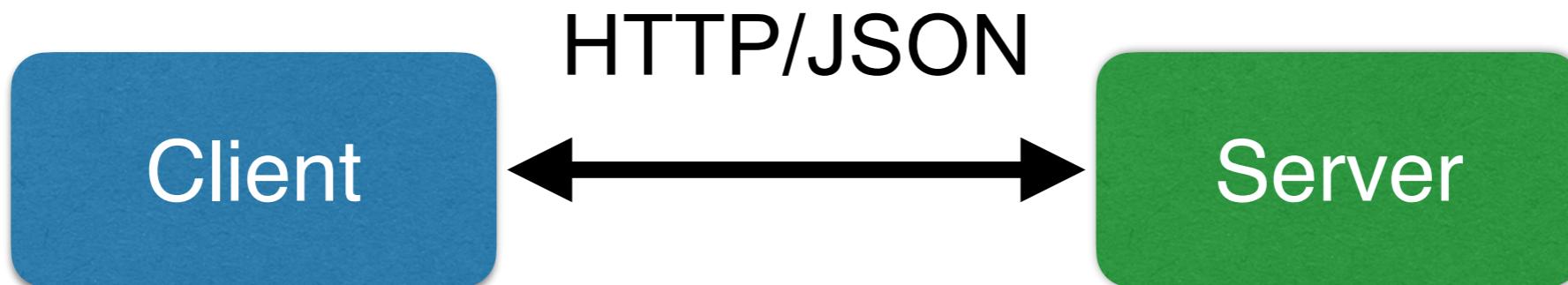
[https://github.com/up1/workshop-go-20201019/blob/main/demo/rest\\_api/README.md](https://github.com/up1/workshop-go-20201019/blob/main/demo/rest_api/README.md)



# Building REST APIs

HTTP + JSON

Go provide **net/http** package



<https://golang.org/pkg/net/http/>



# REST

## REpresentational State Transfer

HTTP Method	Description
GET	Get data
POST	Create data
PUT	Update data
DELETE	Delete data



# Hello API Server

```
package main

import (
    "net/http"
)

func Response(w http.ResponseWriter, r *http.Request) {
    w.Write([]byte("Hello world."))
}

func main() {
    http.HandleFunc("/", Response)
    http.ListenAndServe(":8080", nil)
}
```



# Run

\$go run <filename.go>



# Performance testing

\$go-wrk http://localhost:8080



# Working with JSON

## Data structure with struct

```
import (
    "encoding/json"
    "net/http"
)

type User struct {
    Firstname string `json:"firstname"`
    Lastname  string `json:"lastname"`
    Title     string `json:"title"`
}

type Users []User
```



# Working with JSON

```
func UserHandler(w http.ResponseWriter, r *http.Request) {
    u := Users{
        User{
            Firstname: "f1",
            Lastname: "l1",
            Title:     "Mr.",
        },
        User{
            Firstname: "f2",
            Lastname: "l2",
            Title:     "Miss.",
        },
    }
    w.WriteHeader(http.StatusOK)
    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(u)
}
```



# Better structure



# Manage routes

```
func main() {  
    http.HandleFunc("/", Response)  
    http.HandleFunc("/users", UserHandler)  
    http.ListenAndServe(":8080", nil)  
}
```



# Handlers

```
func UserHandler(w http.ResponseWriter, r *http.Request) {  
    ...  
    w.WriteHeader(http.StatusOK)  
    w.Header().Set("Content-Type", "application/json")  
    json.NewEncoder(w).Encode(u)  
}
```



# Web framework



# Example with Echo

```
import (
    "net/http"
    "github.com/labstack/echo/v4"
)

func main() {
    e := echo.New()
    e.GET("/", func(c echo.Context) error {
        return c.String(http.StatusOK, "Hello, World!")
    })
    e.Logger.Fatal(e.Start(":8080"))
}
```

<https://echo.labstack.com/guide>



# Performance testing

\$go-wrk http://localhost:8080



# Example with Gin

```
import (
    "io/ioutil"
    "github.com/gin-gonic/gin"
)

func main() {
    gin.SetMode(gin.ReleaseMode)
    gin.DefaultWriter = ioutil.Discard
    r := gin.Default()
    r.GET("/", func(c *gin.Context) {
        c.JSON(200, gin.H{
            "message": "pong",
        })
    })
    r.Run()
}
```

<https://github.com/gin-gonic/gin>



# Performance testing

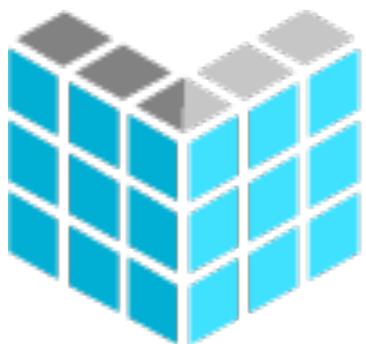
\$go-wrk http://localhost:8080



# Workshop refactoring



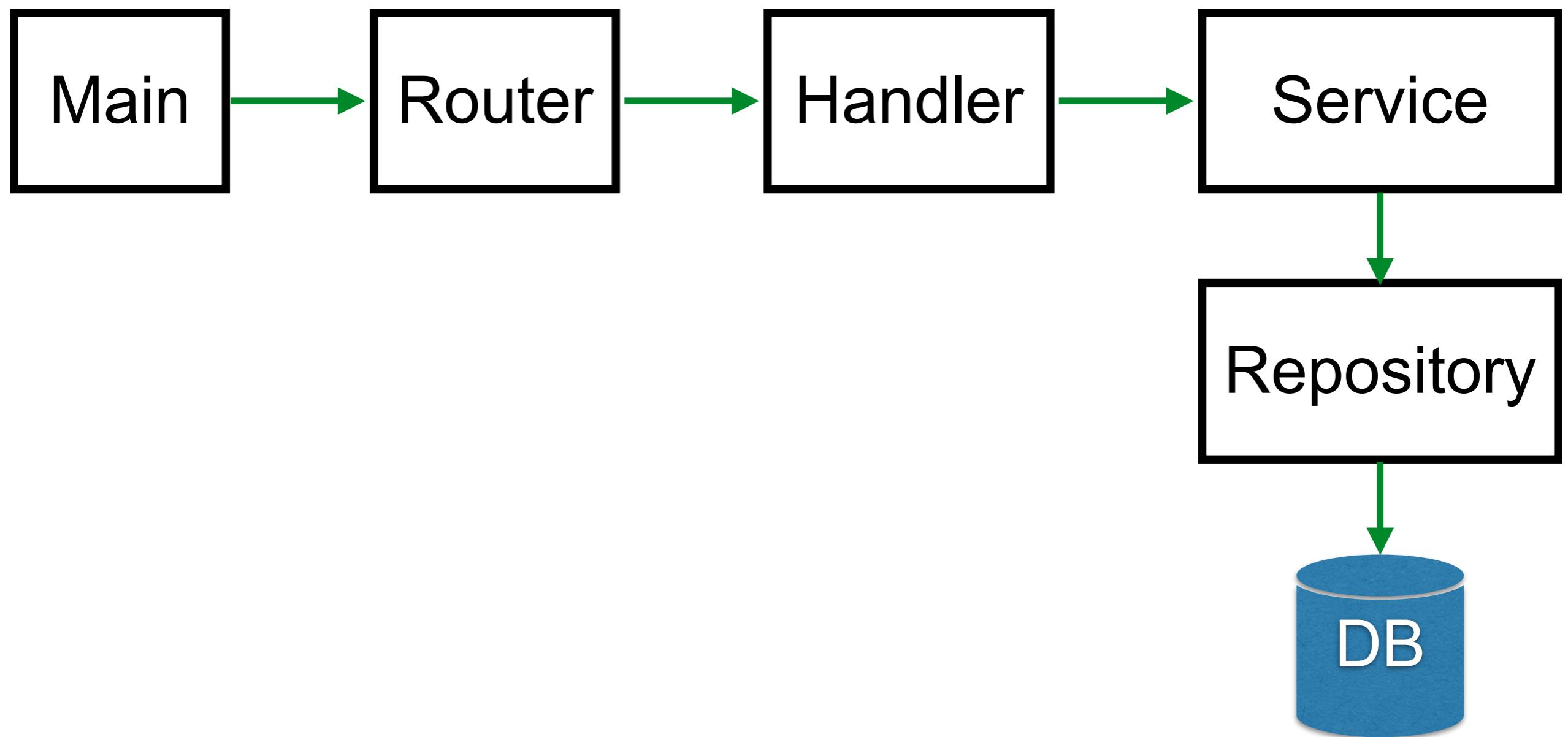
# Easy to change !!



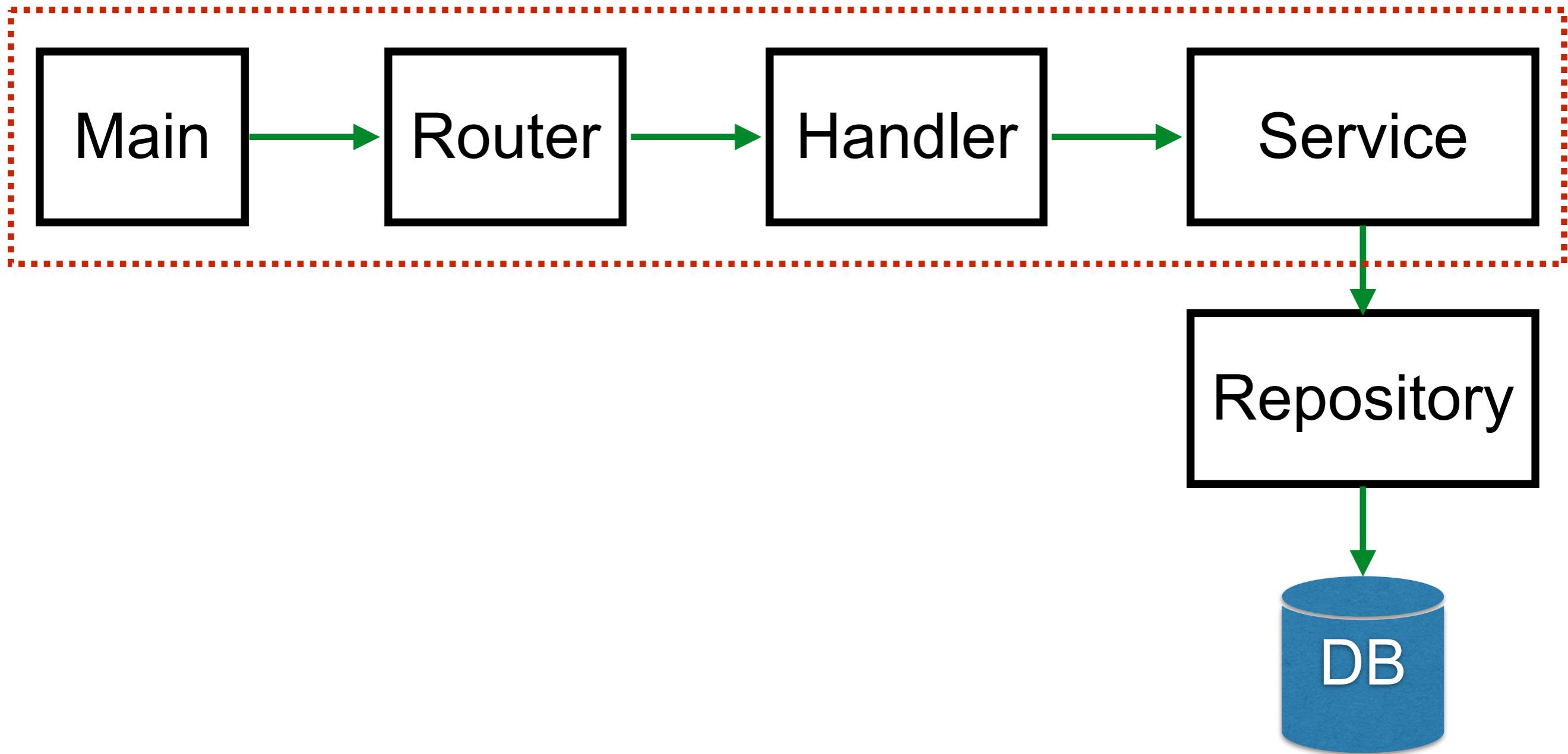
echo



# Better structure



# Better structure



# Service

## Business process/flow

```
type UserService struct{}

func NewUserService() UserService {
    return UserService{}
}

func (us UserService) GetUser(c context.Context) (User, error) {
    user := User{}
    return user, nil
}
```



# Handler/Router

Create routes of RESTful API

```
type UserHandler struct {  
    Service UserServicer  
}
```

```
func NewUserHandler(e *echo.Echo, service UserServicer) {  
    h := UserHandler{Service: service}  
    e.GET("/user", h.GetUser)  
}
```

Inject service to handler and Create routes



# Handler/Router

Handle request and return response

```
func (h *UserHandler) GetUser(c echo.Context) error {
    ctx := c.Request().Context()

    user, err := h.Service.GetUser(ctx) Call service layer
    if err != nil {
        return c.JSON(getStatusCode(err), ResponseError{Message:
err.Error()})
    }
    return c.JSON(http.StatusOK, user)
}
```



# Main

## Initial and start server

```
func main() {
    // Create server with Echo
    e := echo.New()
    Create service and initial handler
    // Initial dependencies
    userService := user.NewUserService()
    user.NewUserHandler(e, userService)
    // Start server
    e.Logger.Fatal(e.Start(":8080"))
}
```



# Testing RESTful APIs



# Testing

## net/http/test

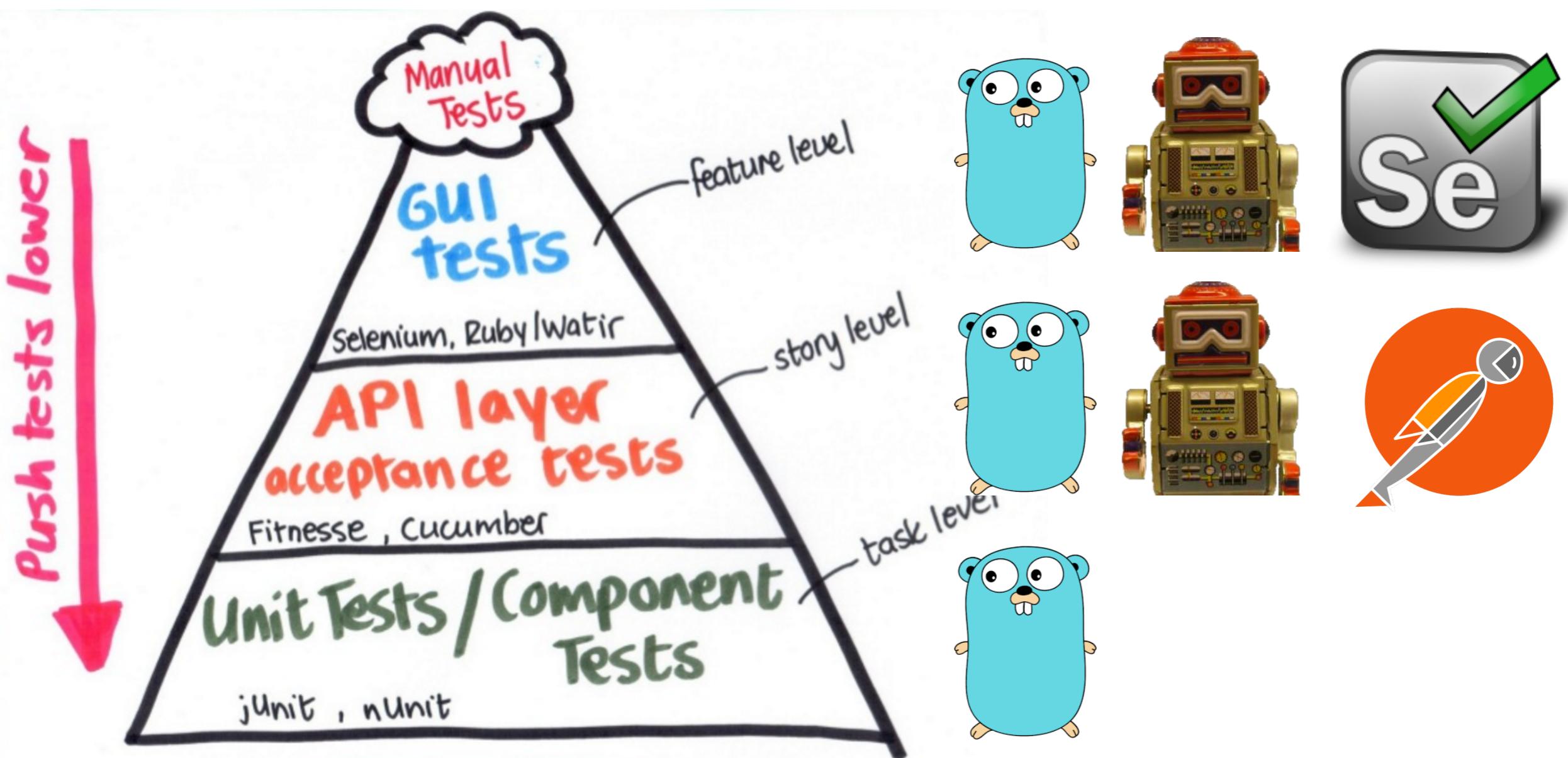
## Postman

## Robotframework

## Cotton



# Testing



# Using net/httpptest

```
func TestHTTPGetHello(t *testing.T) {
    // Arrange
    request, err := http.NewRequest(http.MethodGet, "/", nil)
    if err != nil {
        t.Error(err)
    }
    response := httptest.NewRecorder()
    handler := http.HandlerFunc(Response)
    handler.ServeHTTP(response, request)

    // Assert
    if status := response.Code; status != http.StatusOK {
        t.Errorf("Wrong code: got %v want %v", status, http.StatusOK)
    }
    if response.Body.String() != "Hello world" {
        t.Errorf("errors %s", response.Body.String())
    }
}
```

<https://golang.org/pkg/net/http/httptest/>



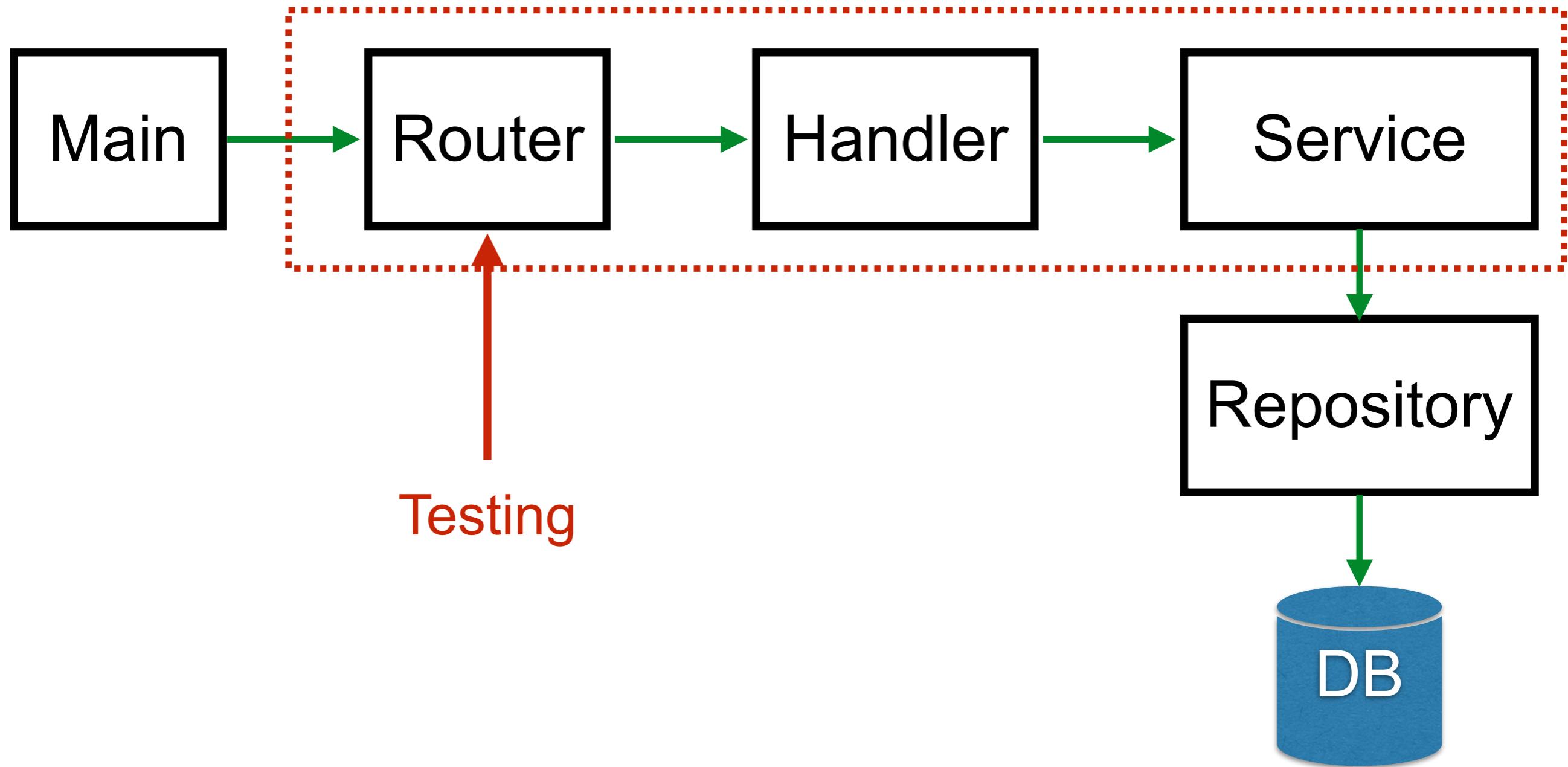
# Testing JSON response

```
func TestHTTPGetUsers(t *testing.T) {
    // Arrange
    request, err := http.NewRequest(http.MethodGet, "/users", nil)
    if err != nil {
        t.Error(err)
    }
    response := httptest.NewRecorder()
    handler := http.HandlerFunc(UserHandler)
    handler.ServeHTTP(response, request)

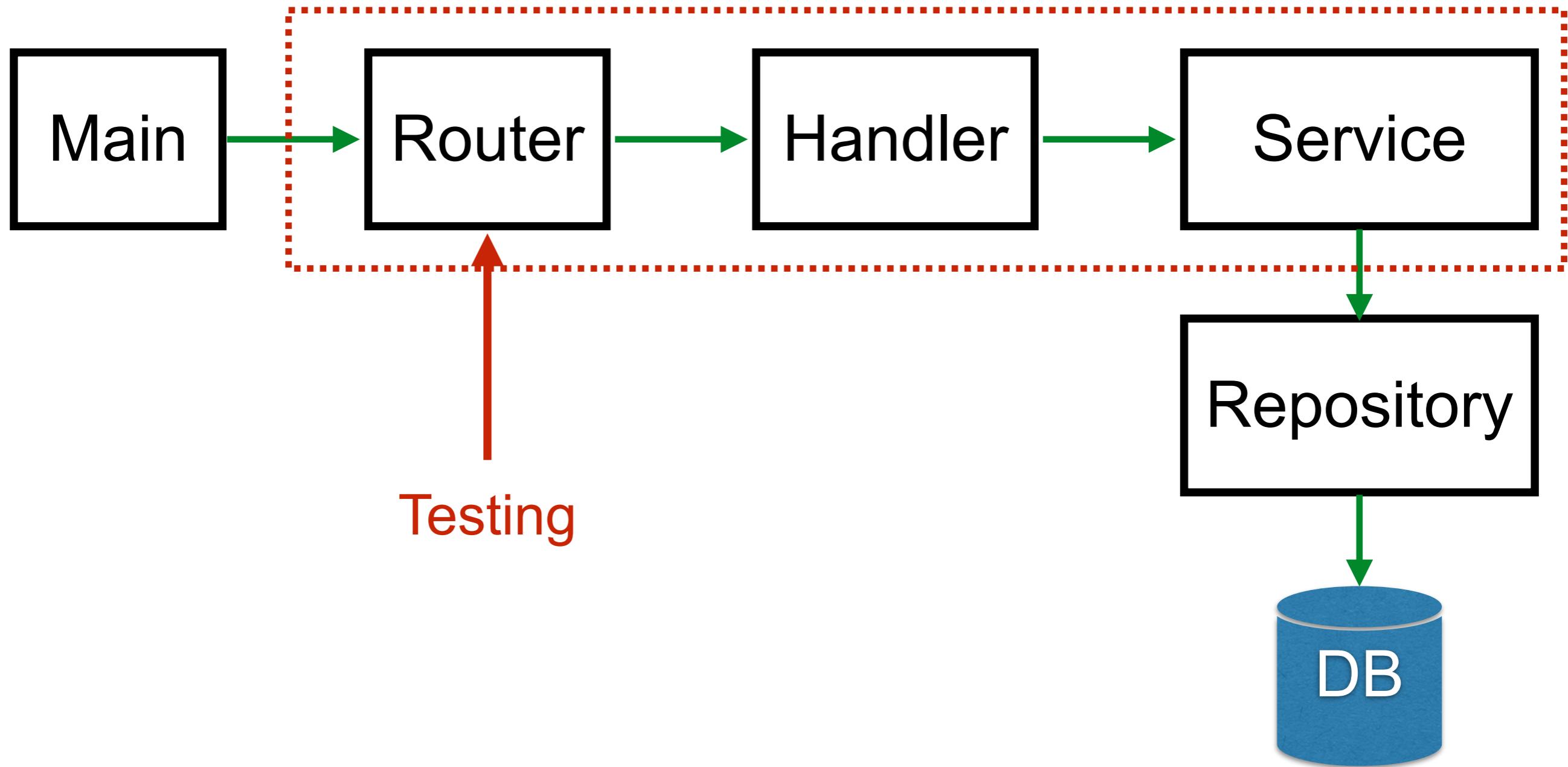
    // Assert
    results := Users{}
    if err := json.NewDecoder(response.Body).Decode(&results); err != nil {
        t.Error(err)
    }
    if len(results) != 2 {
        t.Errorf("Errors with length %v", len(results))
    }
}
```



# Testing !!

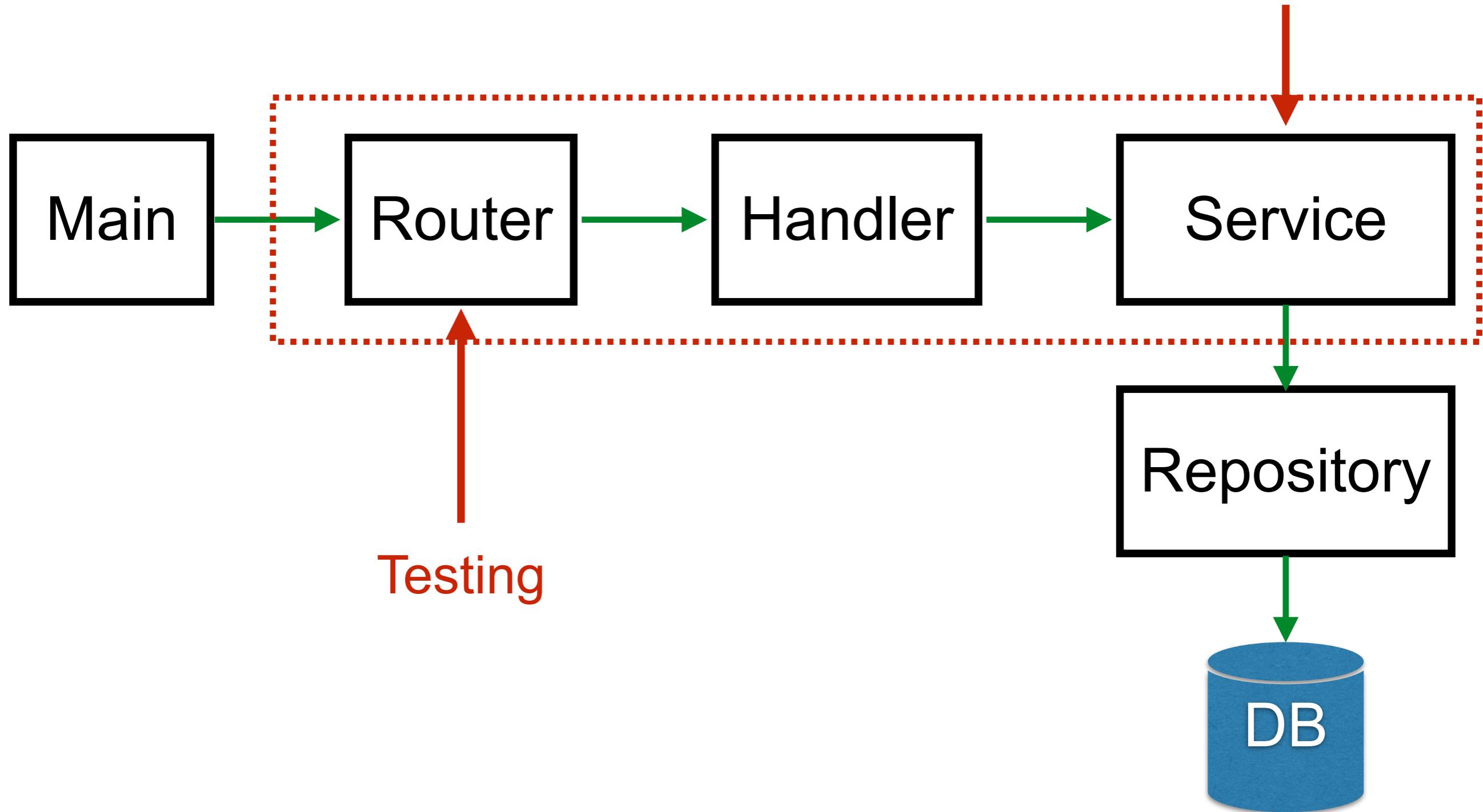


# Problem ?

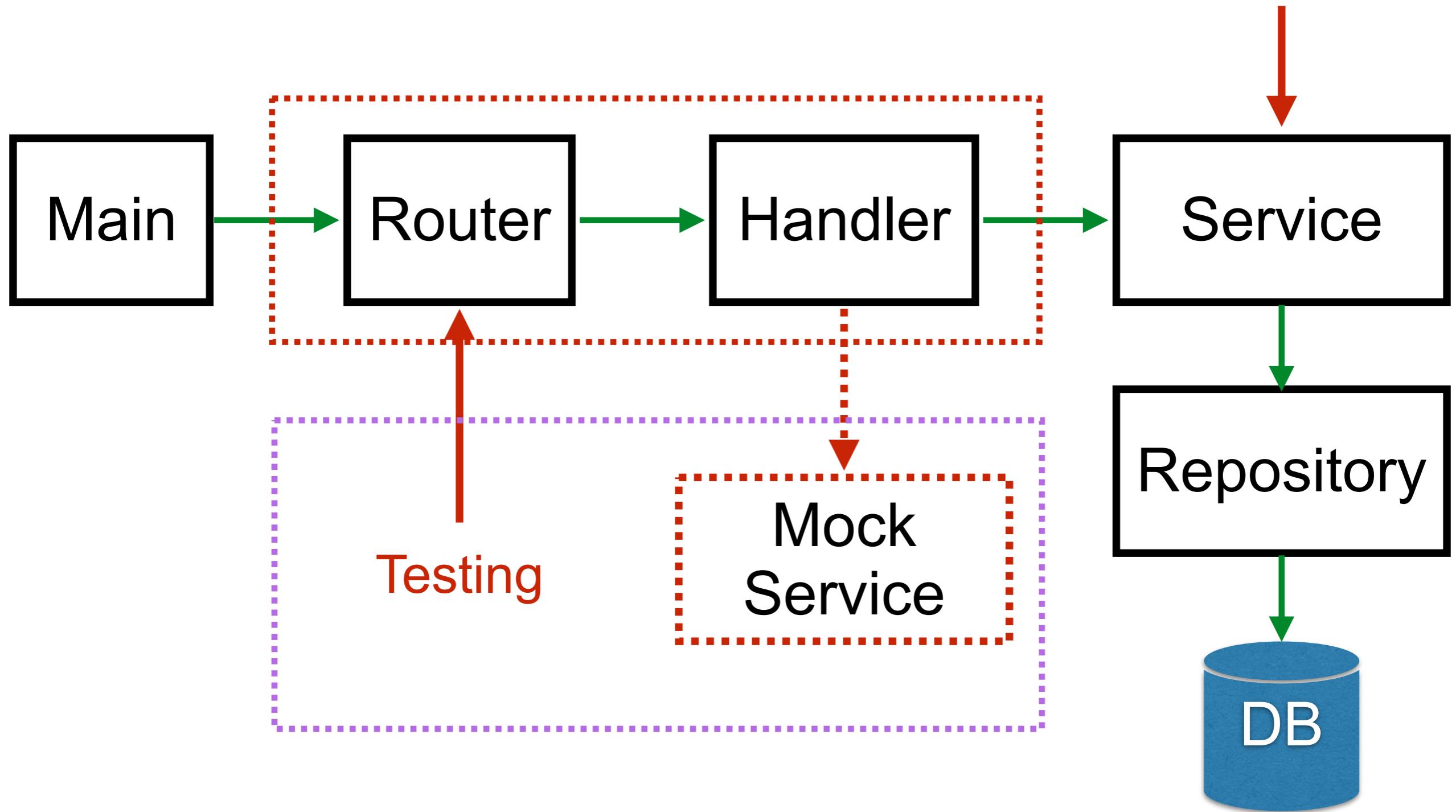


# Problem ?

Problem ...



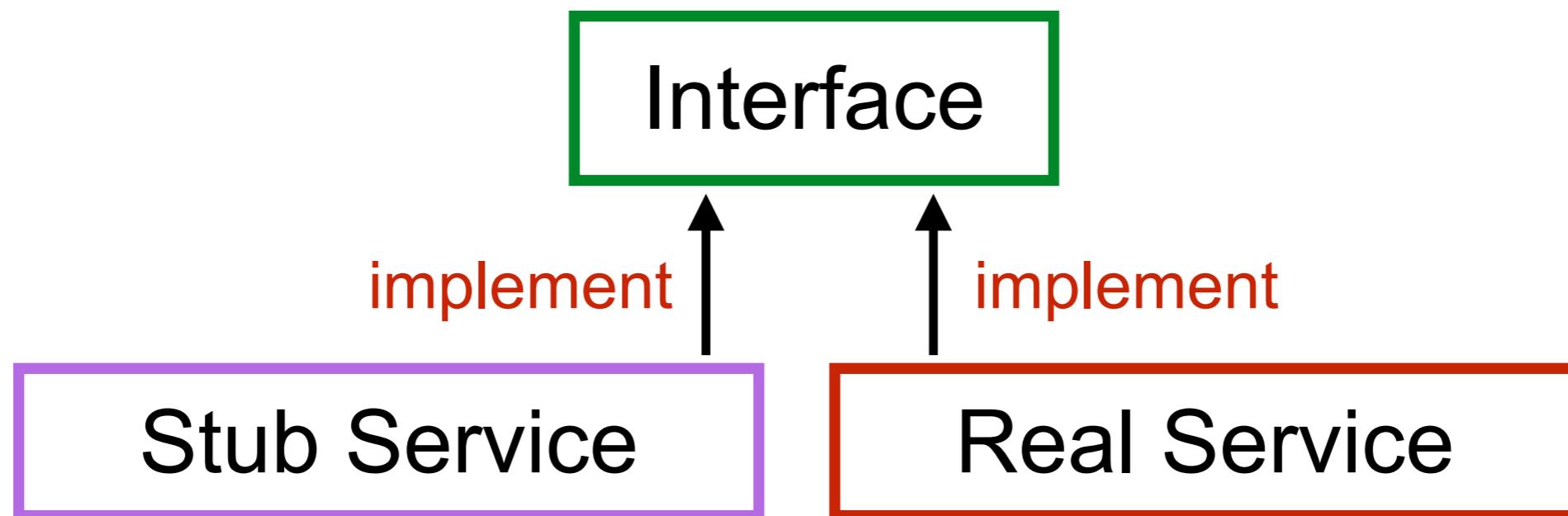
# Problem ?



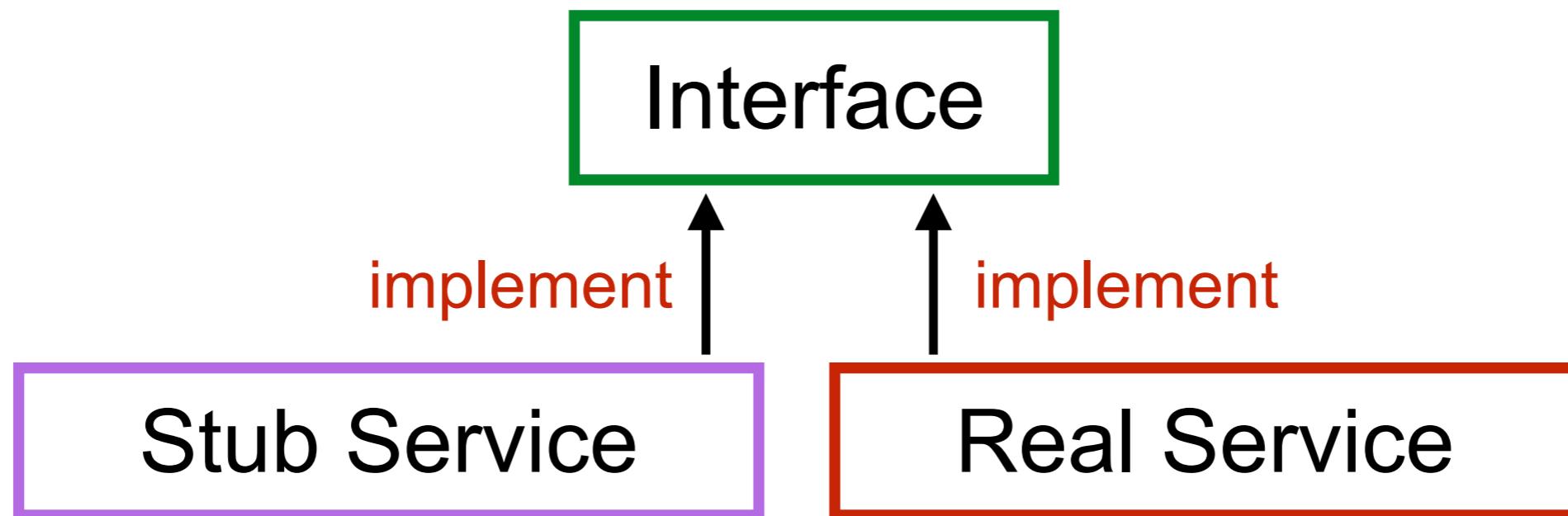
# Testing with Mock service

```
type stubUserService struct {  
    err error  
}
```

```
func (s stubUserService) GetUser(c context.Context) (user.User,  
error) {  
    user := user.User{Firstname: "user test"}  
    return user, s.err  
}
```



# Testing with Mock service



# Create interface of user

```
type UserServicer interface {
    GetUser(c context.Context) (User, error)
}
```



# Testing :: success

```
func TestSuccessWith GetUser(t *testing.T) {
    e := echo.New()
    req, err := http.NewRequest(http.MethodGet, "/user", nil)
    assert.NoError(t, err)

    rec := httptest.NewRecorder()
    c := e.NewContext(req, rec)
    handler := user.UserHandler{
        Service: stubUserService{},
    }
    err = handler.GetUser(c)
    require.NoError(t, err)

    assert.Equal(t, http.StatusOK, rec.Code)
    assert.JSONEq(t, `{"firstname":"user test","lastname":"","title":""}`,
        rec.Body.String())
}
```



# Testing :: failure

```
func TestFailWith GetUser(t *testing.T) {
    e := echo.New()
    req, err := http.NewRequest(http.MethodGet, "/user", nil)
    assert.NoError(t, err)

    rec := httptest.NewRecorder()
    c := e.NewContext(req, rec)
    handler := user.UserHandler{
        Service: stubUserService{err: fmt.Errorf("Error")},
    }
    err = handler.GetUser(c)
    require.NoError(t, err)

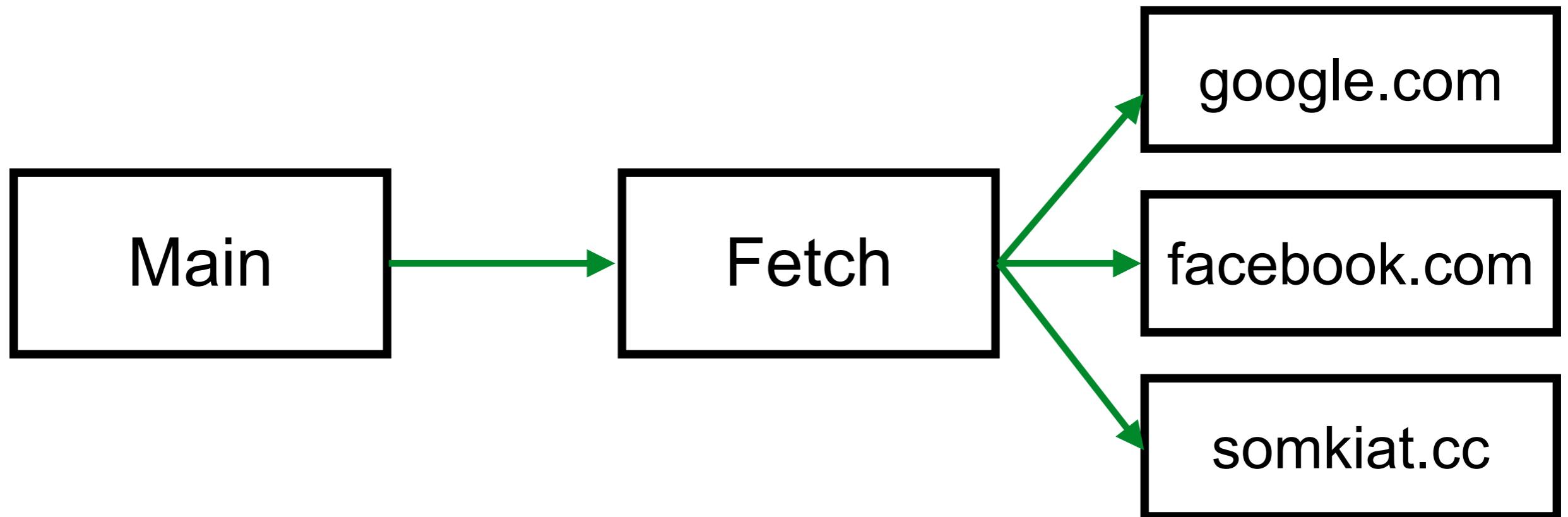
    assert.Equal(t, http.StatusInternalServerError, rec.Code)
}
```



# Graceful Shutdown



# Workshop



# Workshop with RESTful API



# Working with database



# Connect to Database

```
import (
    _ "github.com/lib/pq"
)

func createConnection() *sql.DB {
    // Open the connection
    db, err := sql.Open("postgres", os.Getenv("POSTGRES_URL"))
    if err != nil {
        panic(err)
    }
    db.SetMaxOpenConns(5)

    // check the connection
    err = db.Ping()
    if err != nil {
        panic(err)
    }

    return db
}
```

<https://github.com/lib/pq>



# Query data

```
sqlStatement := `SELECT * FROM users`
rows, err := u.Db.Query(sqlStatement)
if err != nil {
    log.Fatalf("Unable to execute the query. %v", err)
}
defer rows.Close()

// iterate
for rows.Next() {
    var user model.User
    err = rows.Scan(&user.Id, &user.Name, &user.Price)
    if err != nil {
        log.Fatalf("Unable to scan the row. %v", err)
    }
    users = append(users, user)
}
```

<https://github.com/lib/pq>



# Database testing



# Performance testing



# Test Double with Go



# Better structure



# Design RESTful API

Resource	Path	HTTP Verb	Description
todo	/todo/	GET	List of TODO
todo	/todo/	POST	Create new TODO
todo	/todo/1	GET	Get detail of TODO by id
todo	/todo/1	PUT	Update TODO by id
todo	/todo/1	DELETE	Delete TODO by id



# List of TODO

```
← → ⌂ ⓘ localhost:8080/todo/  
[  
  - {  
      id: 1,  
      title: "Todo 1",  
      done: false  
    },  
  - {  
      id: 2,  
      title: "Todo 2",  
      done: false  
    },  
  - {  
      id: 3,  
      title: "Todo 3",  
      done: false  
    }  
]
```



# Create a new TODO

The screenshot shows the Postman application interface. At the top, there is a header bar with 'POST' selected, the URL 'http://localhost:8080/todo/', a 'Params' button, and a 'Send' button. Below the header, tabs for 'Authorization', 'Headers (1)', 'Body' (which is currently selected), 'Pre-request Script', and 'Tests' are visible. Under the 'Body' tab, there are five options: 'form-data', 'x-www-form-urlencoded', 'raw' (which is selected), and 'binary'. The 'raw' section is set to 'JSON (application/json)'. Below these options is a code editor containing the following JSON:

```
1 {  
2   "title": "test",  
3   "done": true  
4 }
```

Below the body editor, there are tabs for 'Body', 'Cookies (25)', 'Headers (3)', and 'Tests'. The 'Body' tab is selected, and the status is shown as 'Status: 200 OK'. Under the 'Body' tab, there are three buttons: 'Pretty', 'Raw', and 'Preview'. The 'Pretty' button is selected. To the right of these buttons is a dropdown menu set to 'JSON'. Below this is another code editor showing the same JSON response:

```
1 {  
2   "id": 0,  
3   "title": "test",  
4   "done": true  
5 }
```



# Get TODO by ID



A screenshot of a web browser window. The address bar shows the URL `localhost:8080/todo/2`. The main content area displays the following JSON object:

```
{  
  id: 2,  
  title: "XXXX",  
  done: true  
}
```



# Go Tools



# Dynamic program analysis

Profiling  
Tracing



# Profiling

Benchmark  
Profiling  
Flame graph



# Pprof

<https://golang.org/pkg/runtime/pprof/>



# Workshop Pprof

<https://github.com/up1/workshop-go-20201019/tree/main/demo/profiling>



# Using graphviz



The screenshot shows the official Graphviz website. At the top, there is a navigation bar with links: About, Download, Gallery, Documentation, Theory and Publications, License, Resources, Credits, FAQ, Contact, and Issues/Bugs. Below the navigation bar is a large teal header section containing the text "Graphviz - Graph Visualization Software" and a small icon of a magnifying glass over a graph. The main content area has a white background and features the heading "Welcome to Graphviz". Below this, a text block invites users to join the forum and provides a note about the URL change.

About Download Gallery Documentation Theory and Publications License  
Resources Credits FAQ Contact Issues/Bugs

## Graphviz - Graph Visualization Software

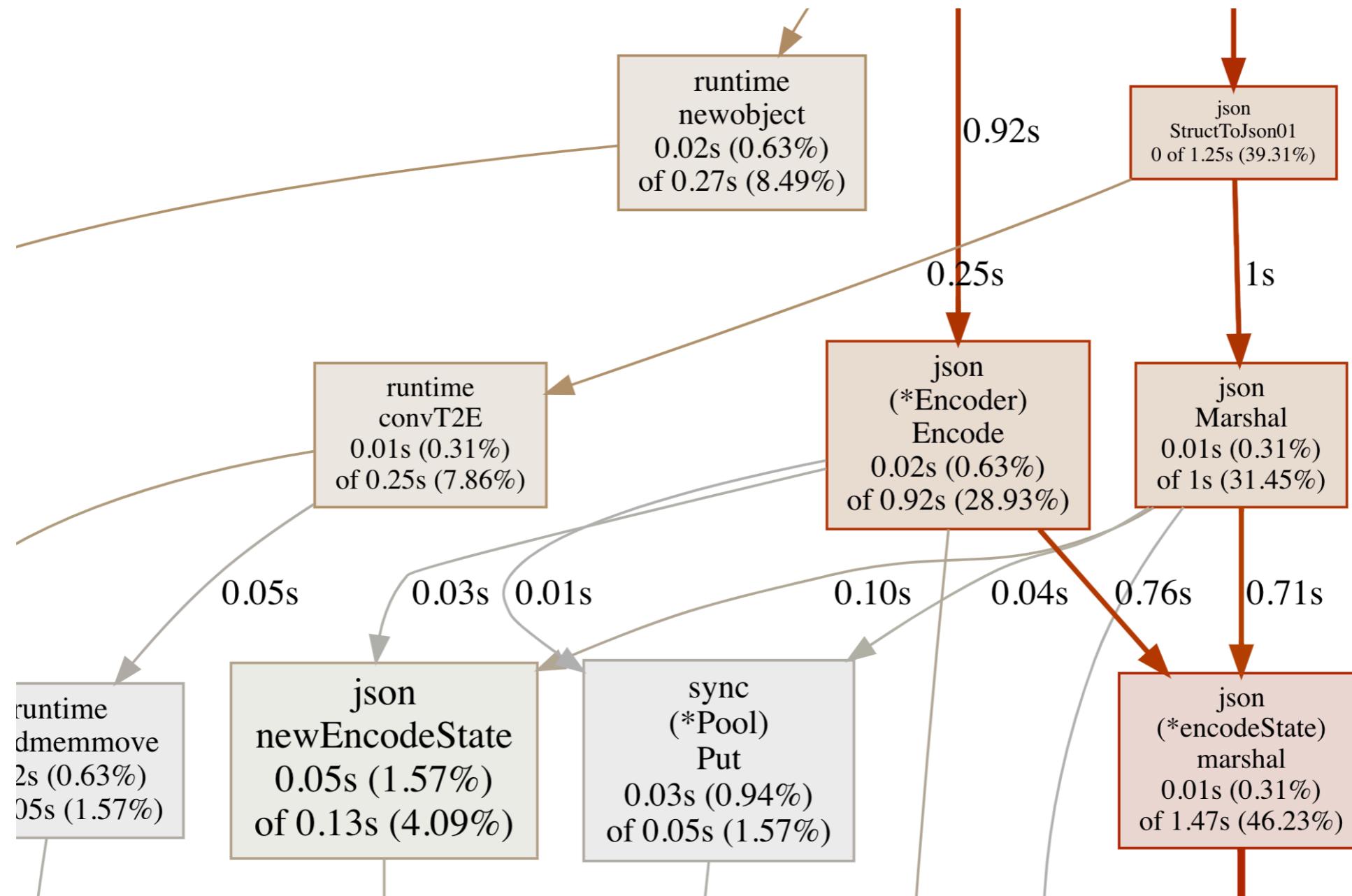
## Welcome to Graphviz

Please join the brand new (March 2020) [Graphviz forum](#) to ask questions and discuss Graphviz. **Note:** The URL is new since May 6 2020. Please update your bookmarks.

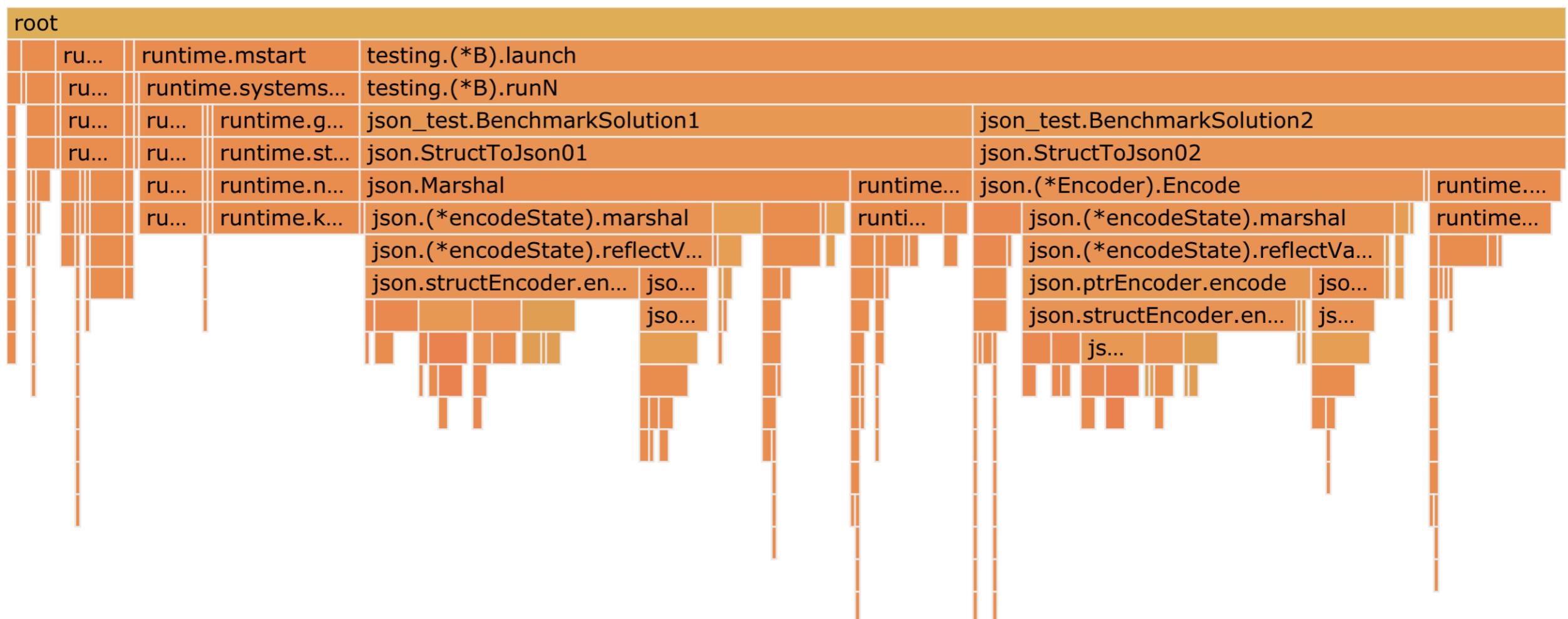
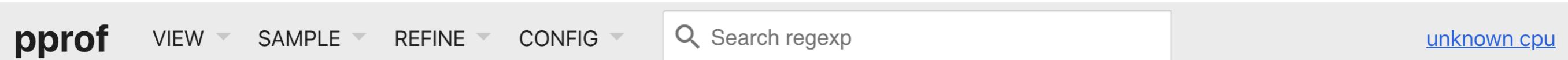
<https://graphviz.org/>



# Result



# Result



# Tracing

Tracing Go programs for deep understanding  
Added to Go 1.5

**\$go tool trace**



# Workshop Tracing

<https://github.com/up1/workshop-go-20201019/tree/main/demo/tracing>



# Let's Go !!

