สยามชำนาญกิจ
SIAM CHAMNAN KIT

Basic of Go programming

Hello Go

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

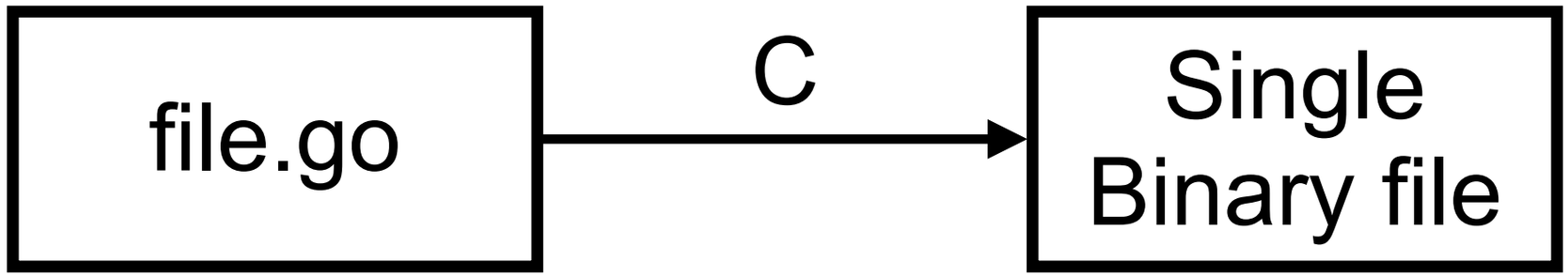
BCPL => the semicolon rule

Smalltalk => method

Newsqueak => <-, :=

APL => iota

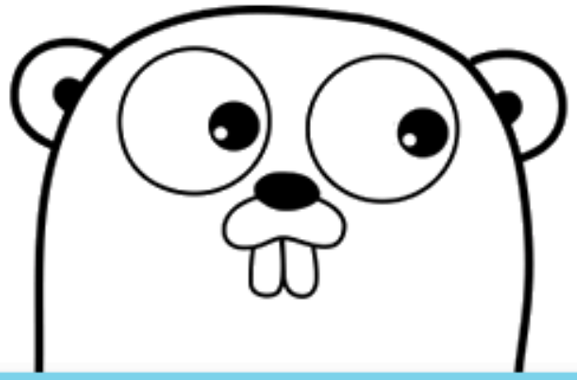
About Go



Installation

Documents Packages The Project Help Blog Play Search

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")
}
```

Hello, World! Run Share Tour

https://golang.org/

Hello Go

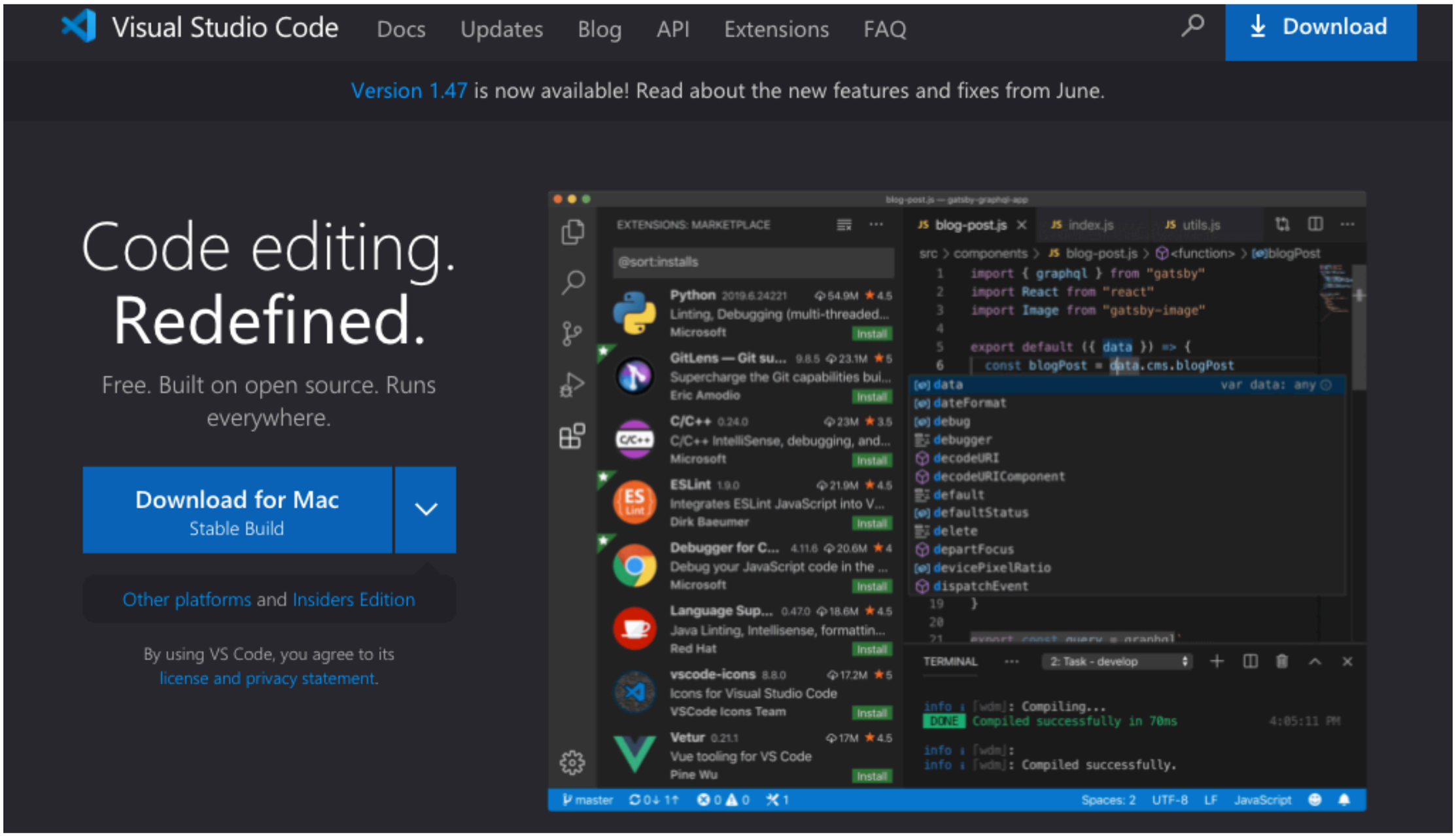
$go version

Hello Go environment

$go env

Development tools

Visual Studio Code



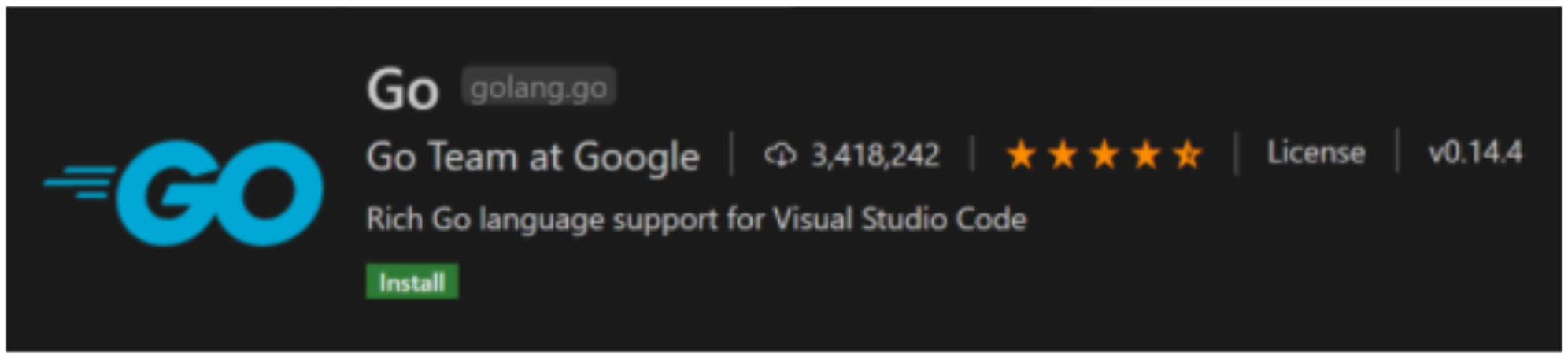
https://code.visualstudio.com/

Extension for Go

Go in Visual Studio Code

Edit

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 gofmt tool. You can switch syntax highlighting on and off by clicking on the

< 1/5 >

hello.go | Imports off | Syntax off

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

Reset | Format | Run

https://tour.golang.org

Effective Go

Effective Go

https://golang.org/doc/effective_go.html

Learn Go

Learn

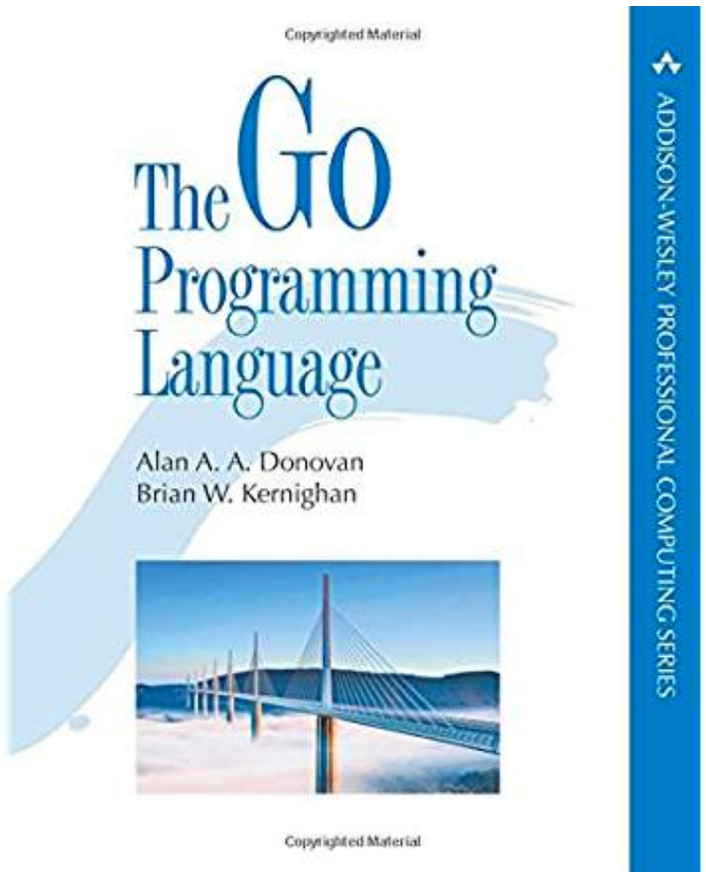
Saurabh Hooda edited this page on Jul 1 · 33 revisions

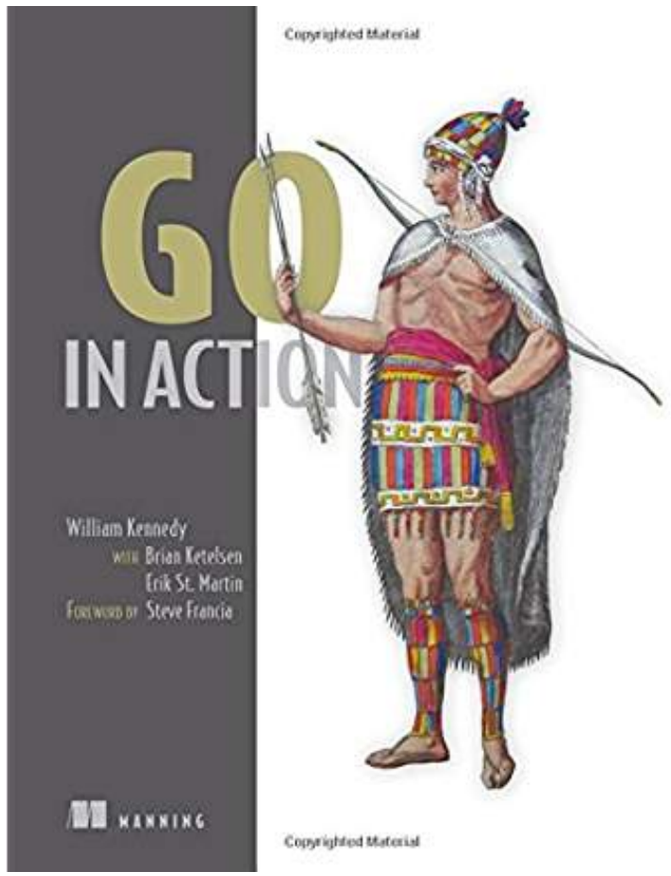
In addition to the resources available at golang.org there are a range of community-driven initiatives:

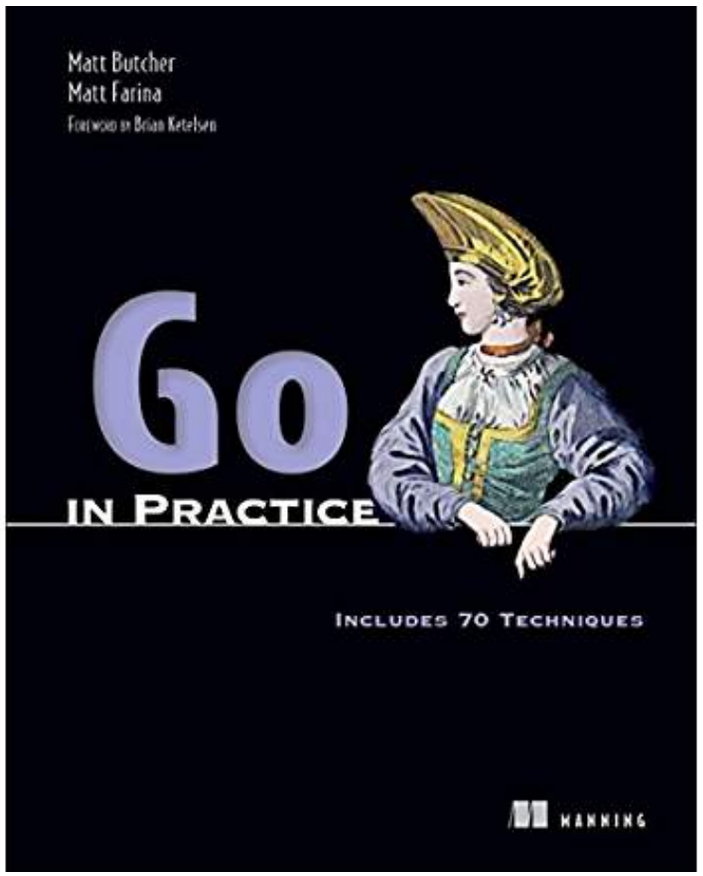
- The Little Go Book
- Exercism.io - Go - Online code exercises for Go for practice and mentorship.
- Learn Go in an Hour - Video *2015-02-15*
- Learning to Program in Go, a multi-part video training class.
- Pluralsight Classes for Go - A growing collection of (paid) online classes.
- Ardan Labs Training - 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







Basic of Go

Features of Go is no feature

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

Hello Go

```go
// hello.go
package main

import "fmt"

func main() {
    fmt.Println("สวสัดี Go")
}
```

Run and Build

$go run hello.go

$go build hello.go
$go build -o xxx hello.go

Code formatting

$go fmt
$gofmt

Godoc my package

$godoc demo
$godoc --http=":6060"

https://github.com/golang/tools

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"
```

Grouping

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

Compiler feature

$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

String

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

Working with String

```go
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

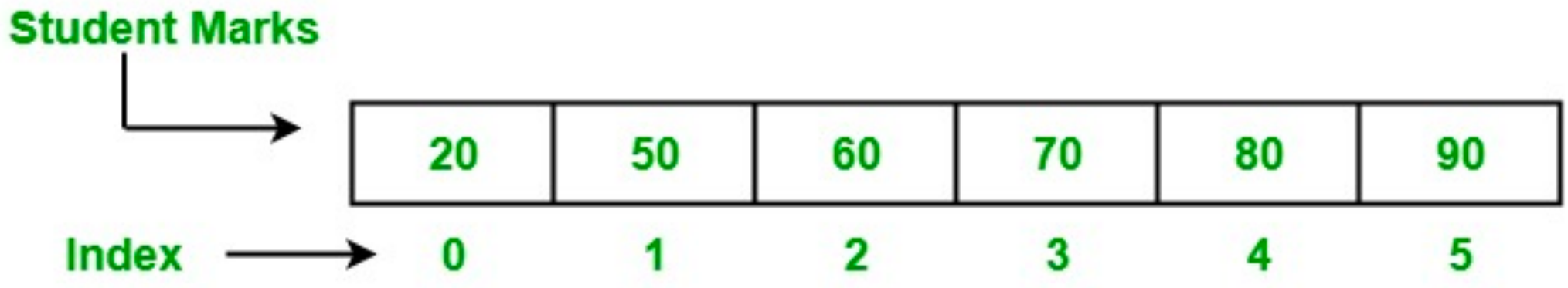
Go has **error type** to dealing with errors
Use from package errors

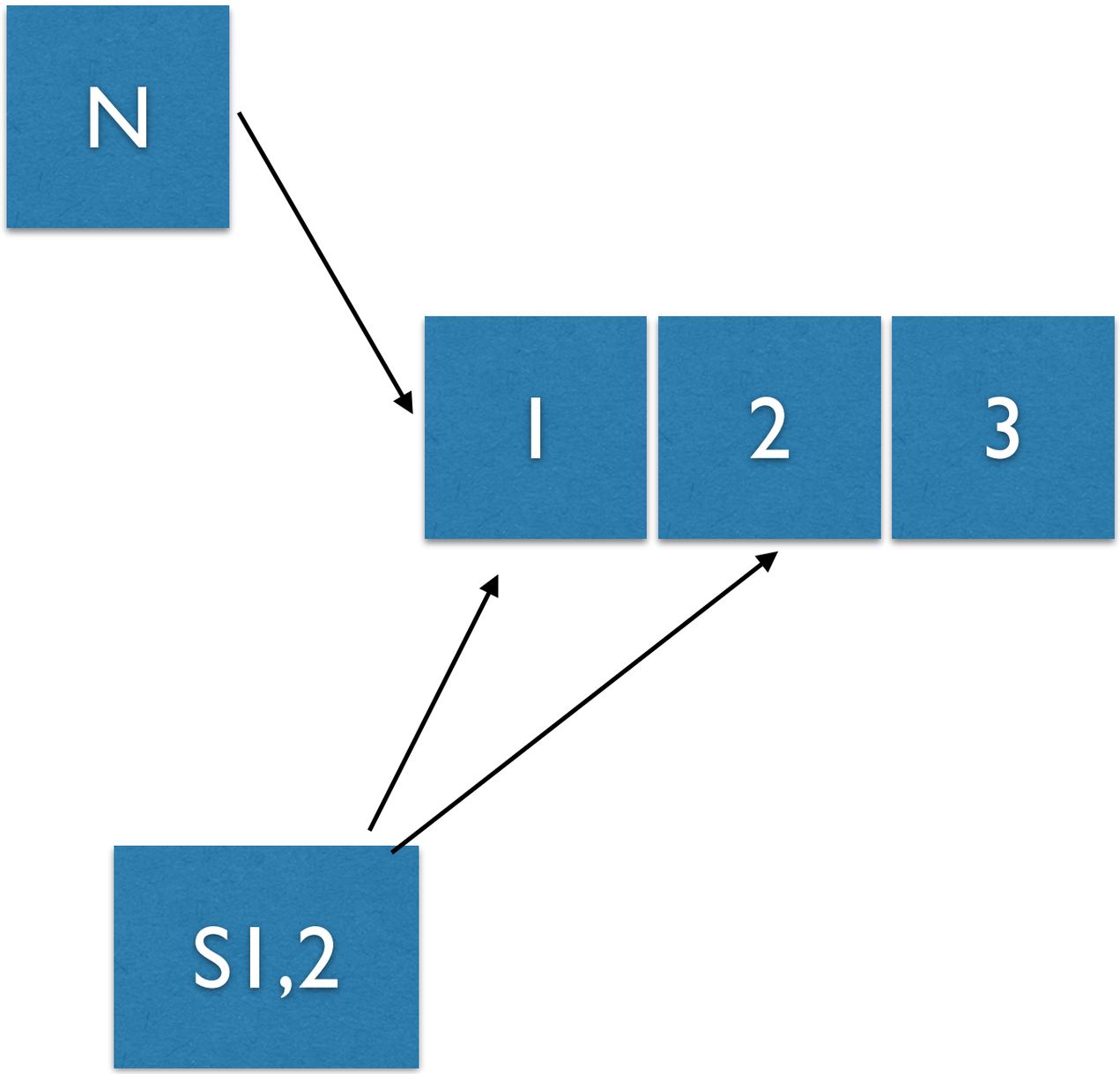
https://golang.org/pkg/errors/

Error types

```
package main

import (
    "errors"
    "fmt"
)

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

Arrays





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

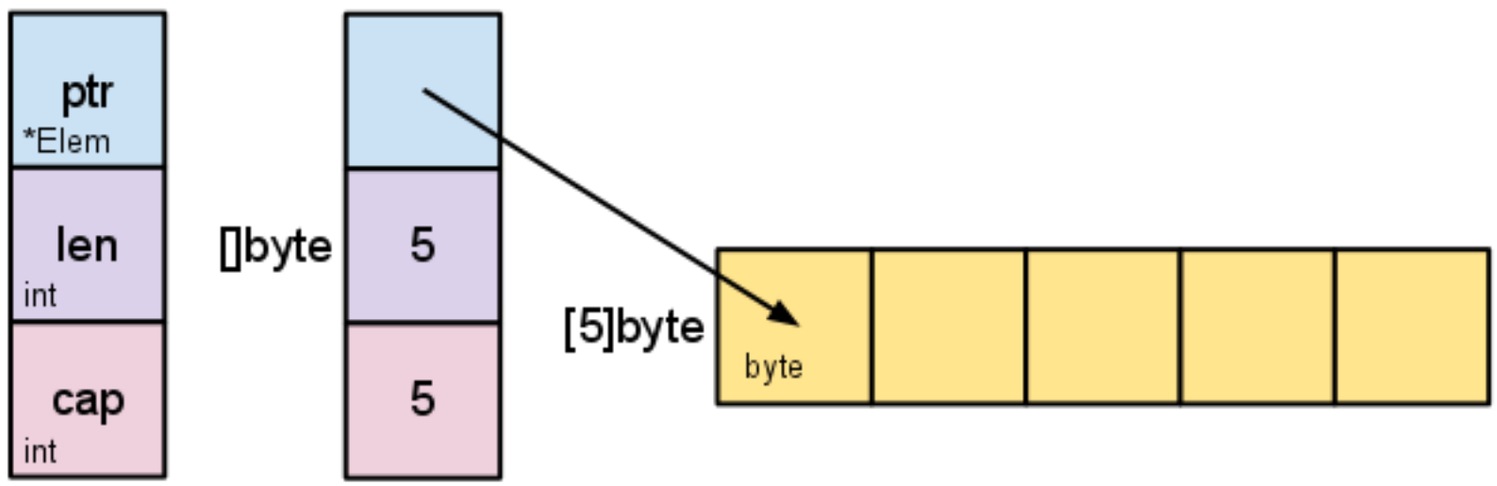
```
func main() {

    var colors = [...]string{"Red", “Blue"}

    for i := 0; i < len(colors); i++ {
        fmt.Println(colors[i])
    }
}
```

Array is of value type !!

```go
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)
}
```

Slice

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



Working with slice

```go
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 = append(names, "n3")
    fmt.Println(names)
}
```

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

Slice with array

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

    var s []int = numbers[1:3]
    fmt.Println(s)
}
```

Slice is reference to array !!

```go
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)
}
```

Sorting with Slice

```
import (
    "fmt"
    "sort"
)

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

https://golang.org/pkg/sort/

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

```go
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 statements

Control statements

If-else
Goto
For
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")
    }
}
```

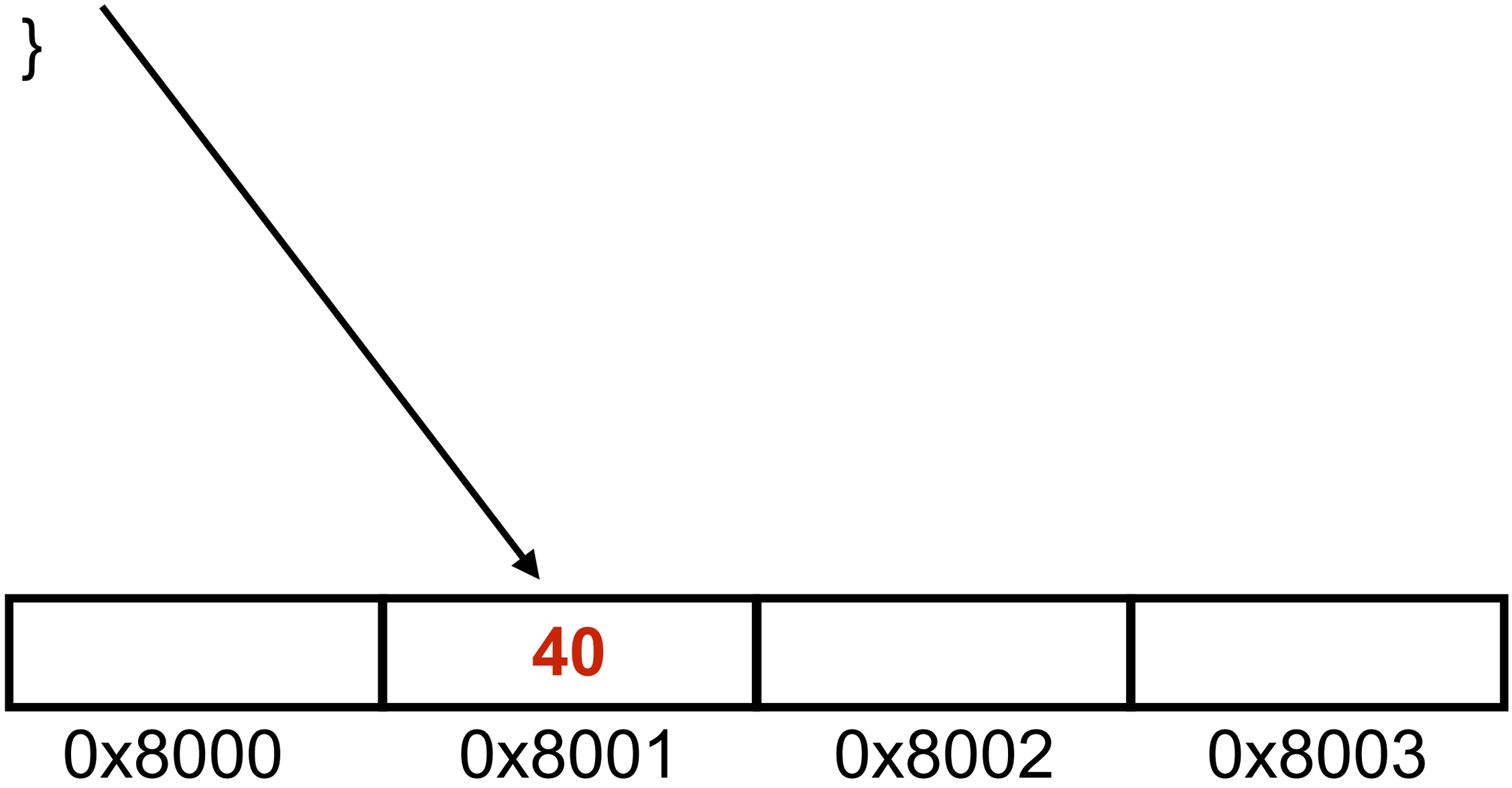
Switch-case with no condition

Readable more than if-else

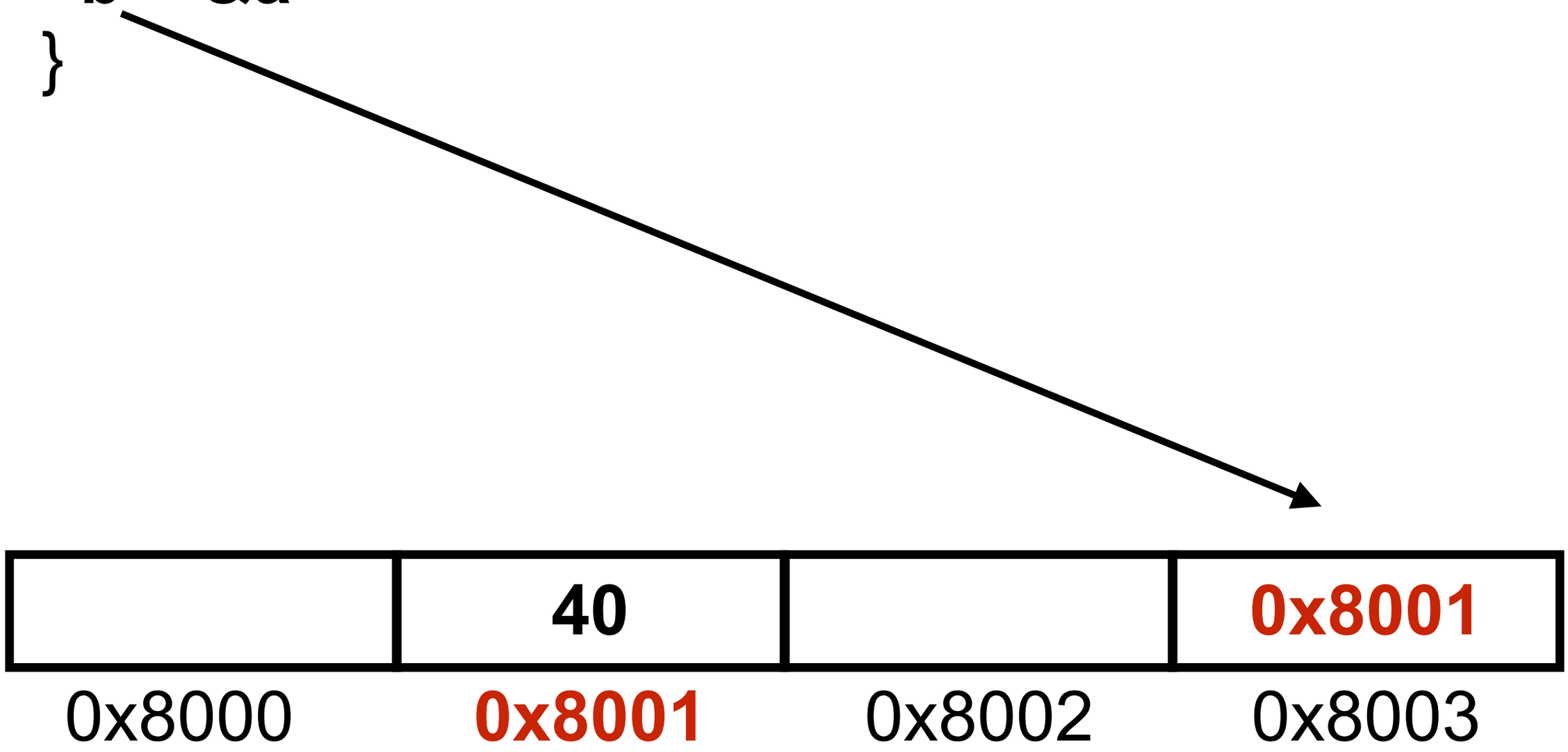
```
func main() {
	score := 65
	switch {
	case score > 80:
		fmt.Println("Grade A")
	case score > 70:
		fmt.Println("Grade B")
	case score > 60:
		fmt.Println("Grade C")
	default:
		fmt.Println("Grade D")
	}
}
```

Pointers

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



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



Functions

Functions

Use keyword **func**

```
func funcName(input1 type1, input2 type2) (output1 type1, output2 type2) {

    // function body
    // multi-value return
    return value1, value2
}
```

Functions

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

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

Multiple return values

```go
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)
    }
}
```

Defer functions

Execute when end function

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

Read file

```go
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)
    }
}
```

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.Print(p1, p2, p3, p4)
}
```

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.Print(p1)
}
```

Object-Oriented

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")
}
```

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")
}
```

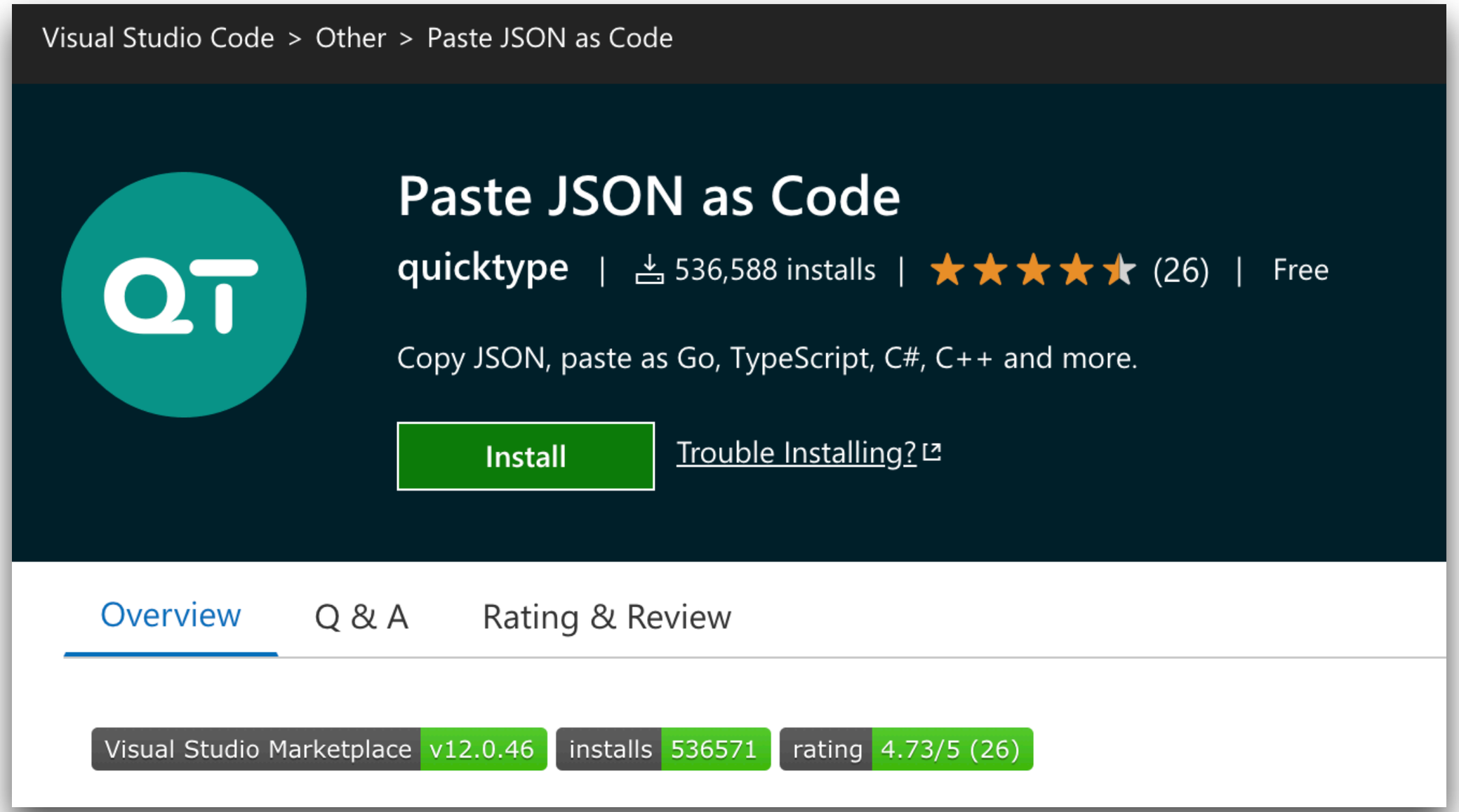
Method overriding

```
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")
}
```

Working with JSON

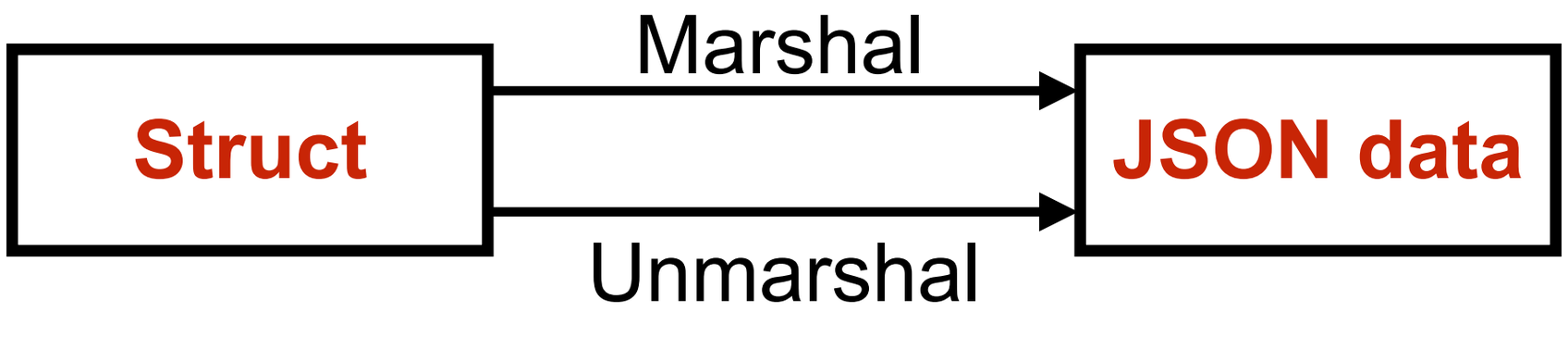
https://golang.org/pkg/encoding/json/

Generate Struct from JSON



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

Working with JSON



https://blog.golang.org/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

```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

```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.Errorf
t.Fatalf
t.Logf

*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>

Interface

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

Go Modules

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

Modules

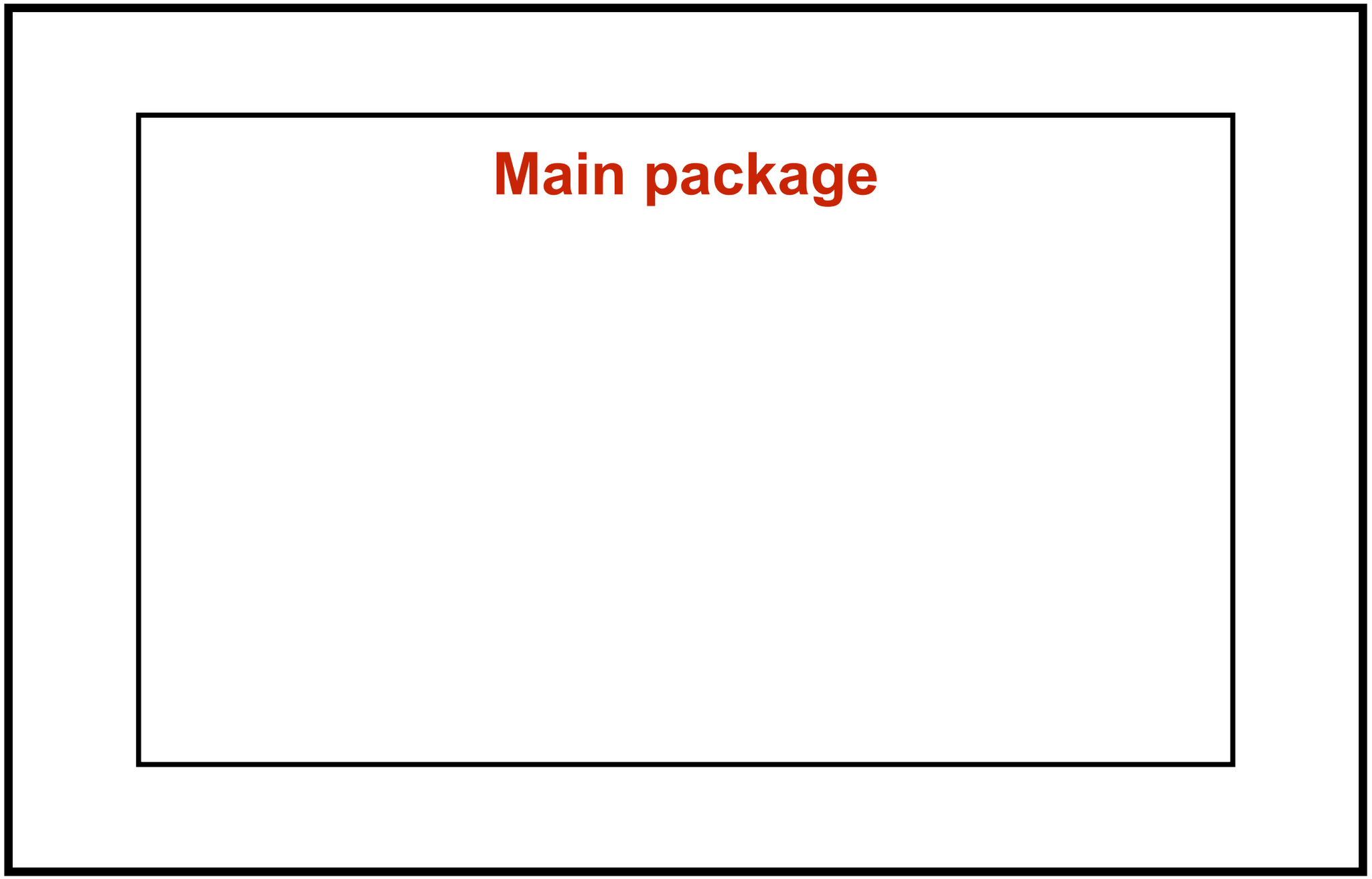
Basis of dependency management
Group of packages into single unit

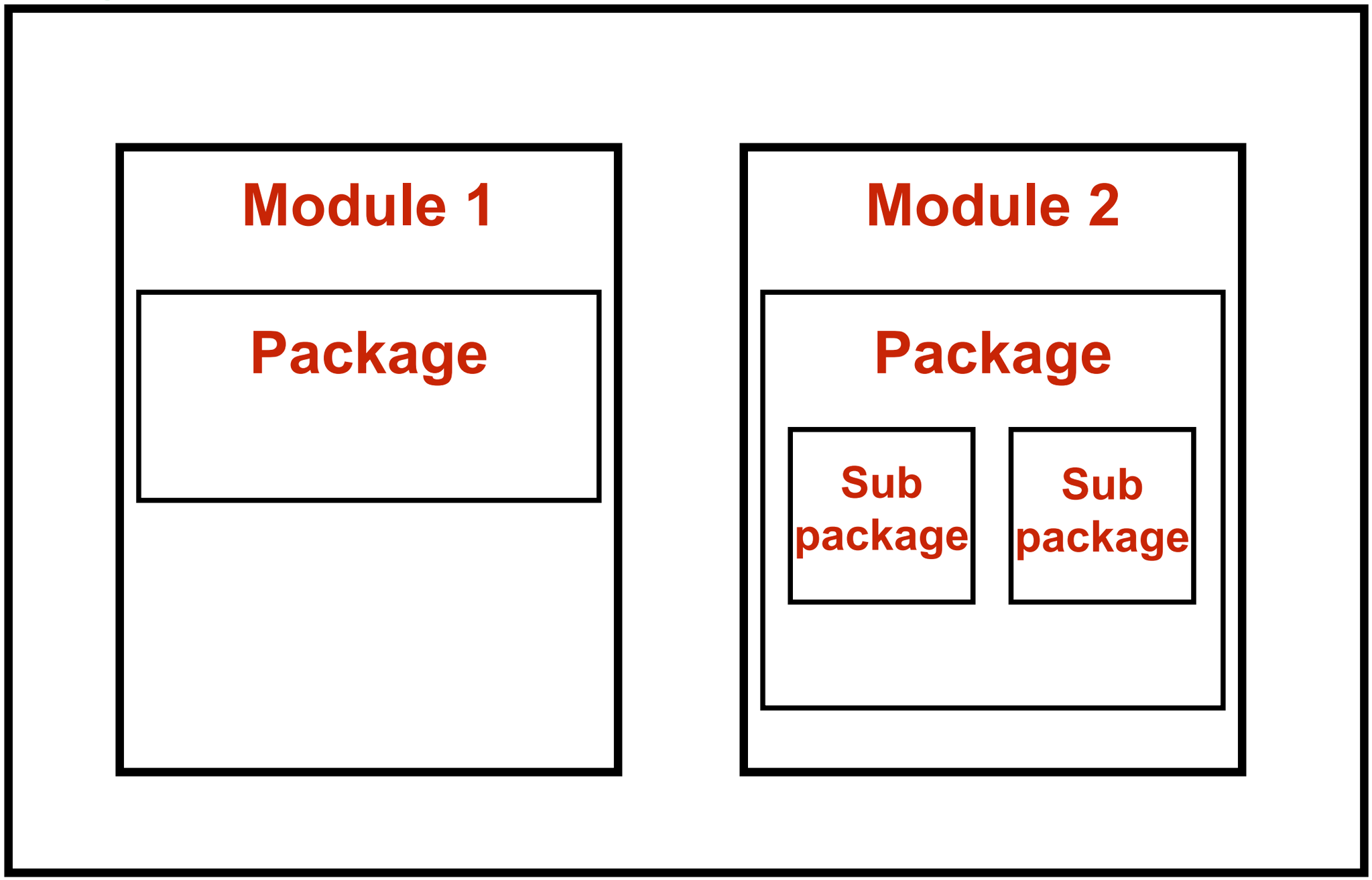
Set of dependencies and versioning

Semantic versioning

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

https://semver.org/





Using a single module per repository

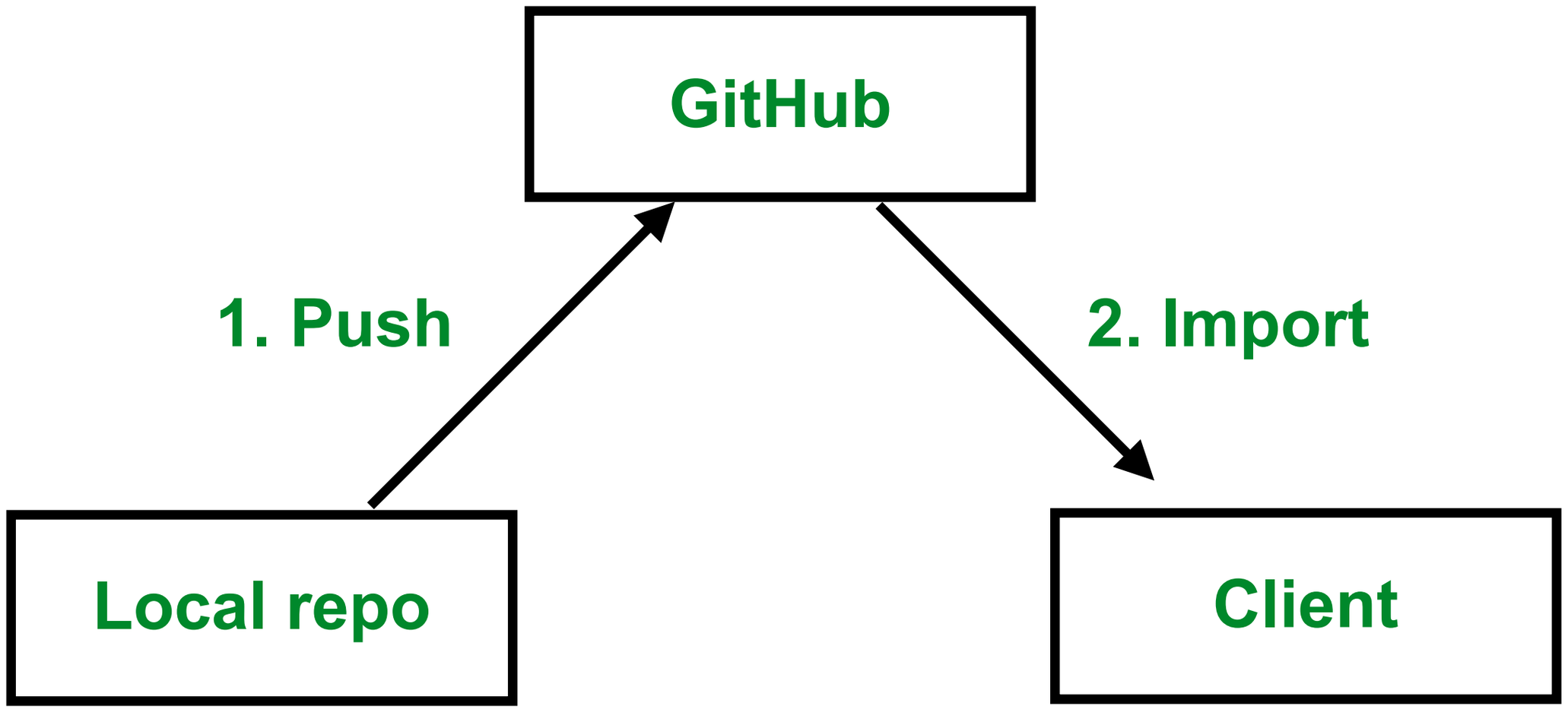
Create a module

$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

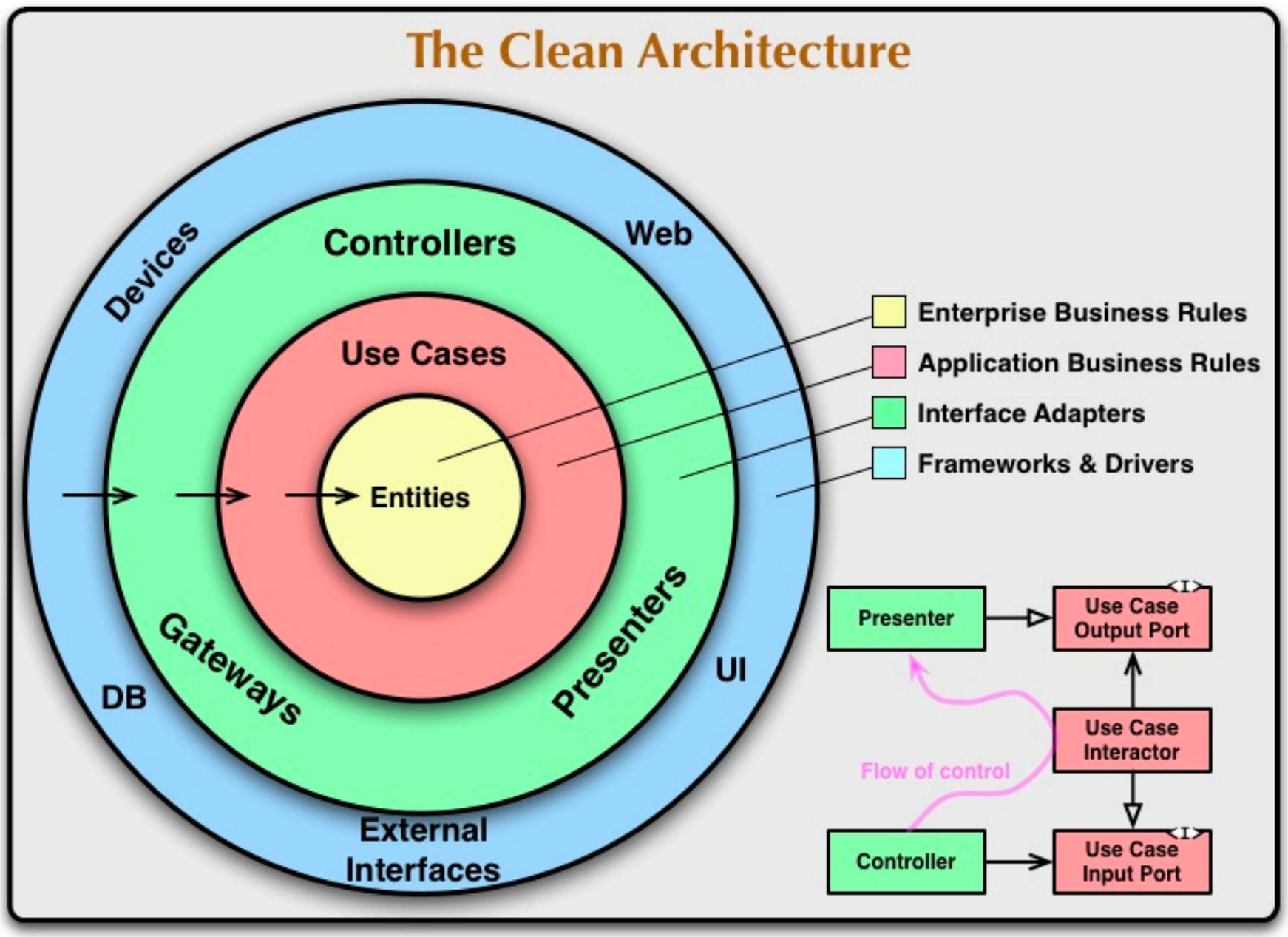
Layering

Grouping by function

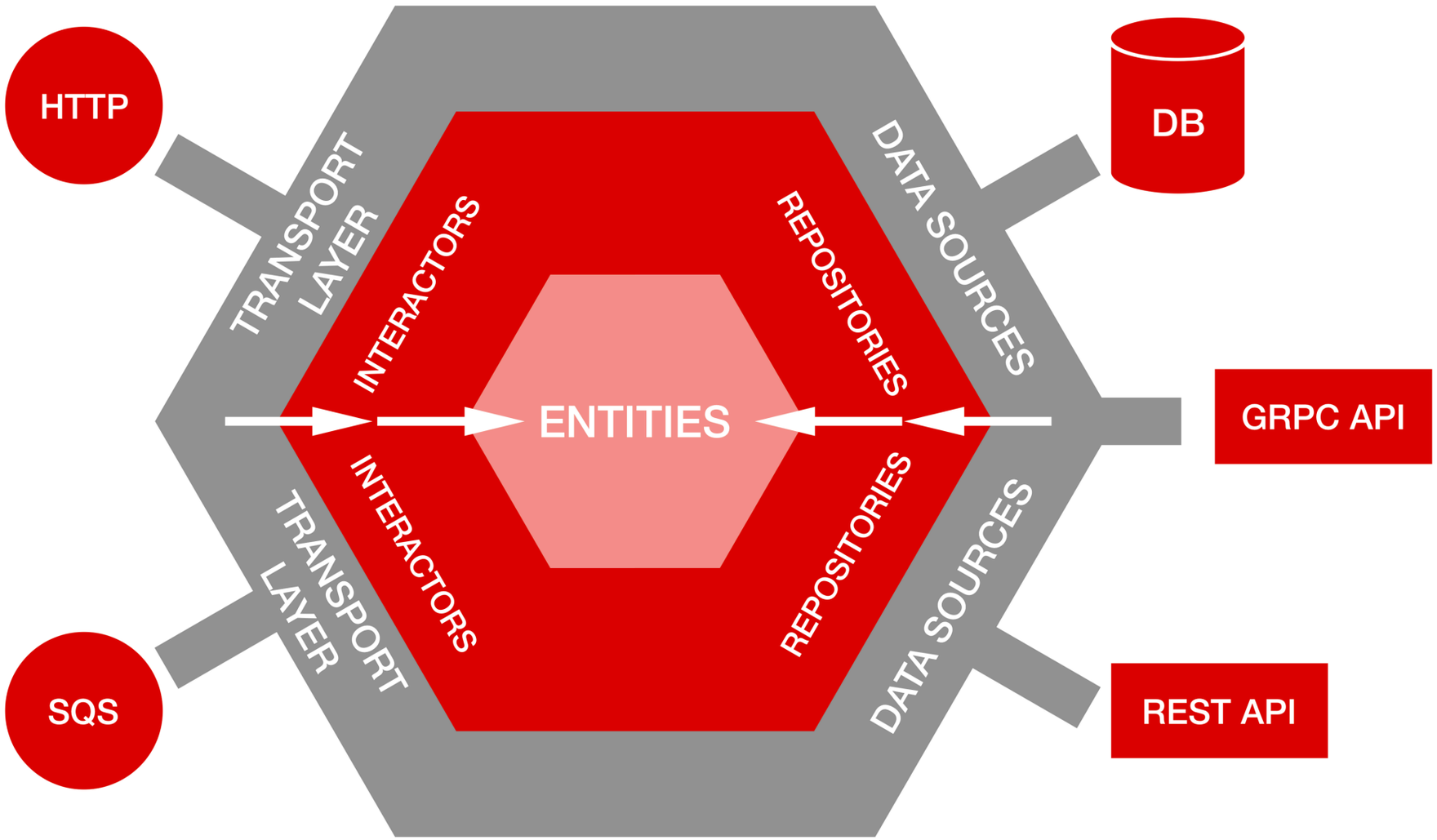
Modular

Grouping by logical/business

Clean architecture

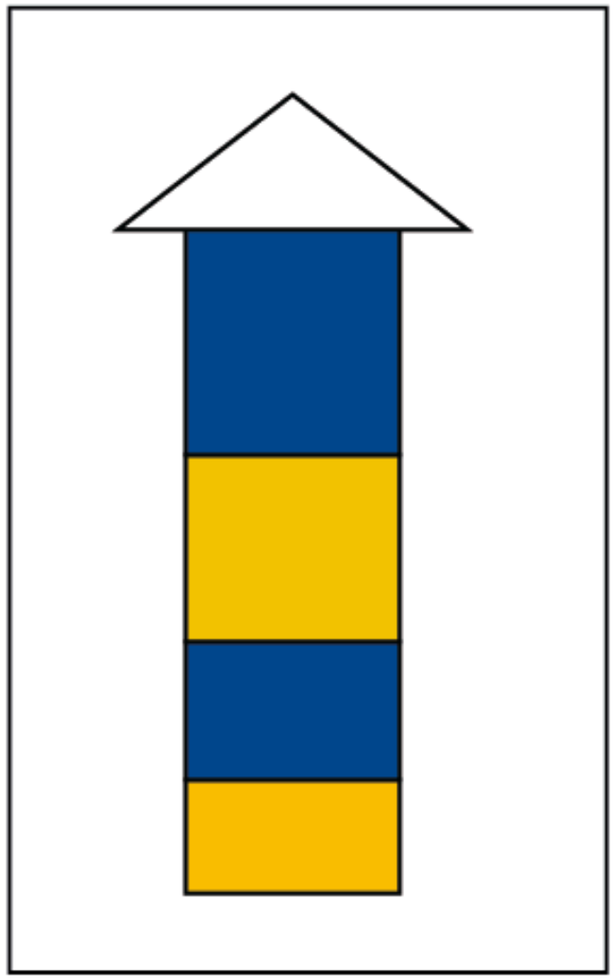


Hexagonal architecture



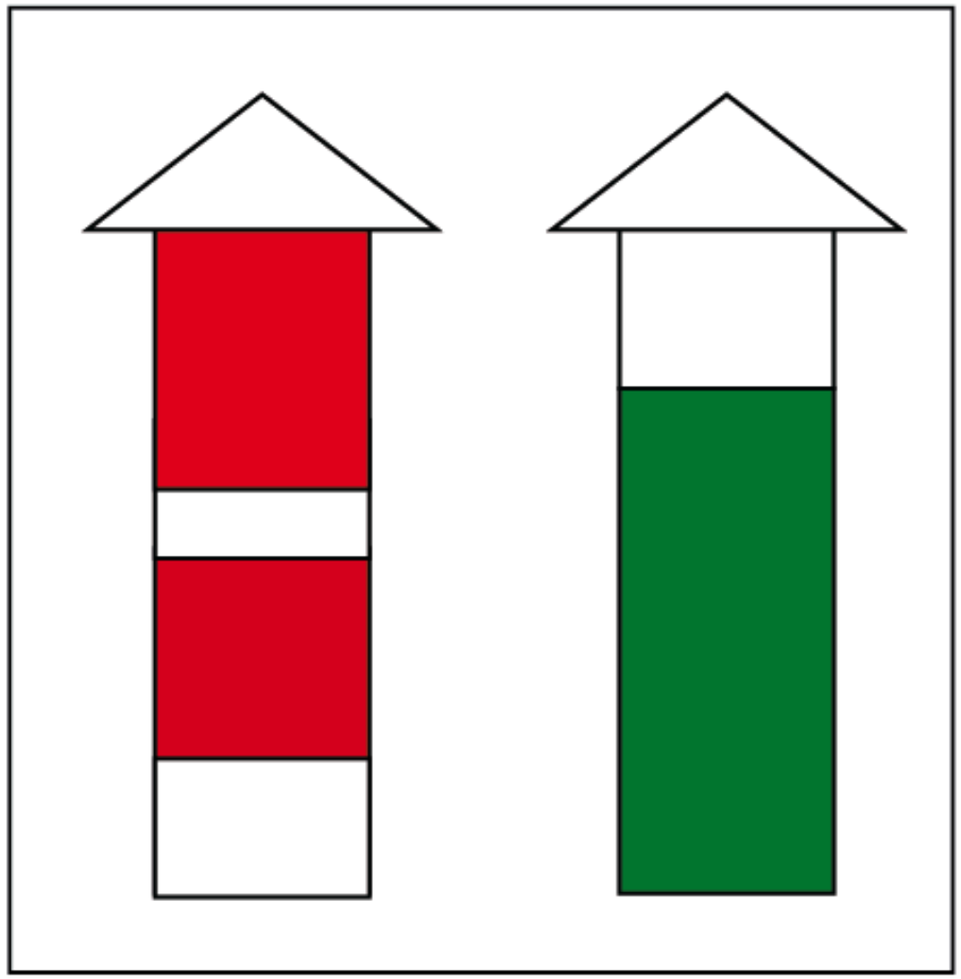
Go routine and channel

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

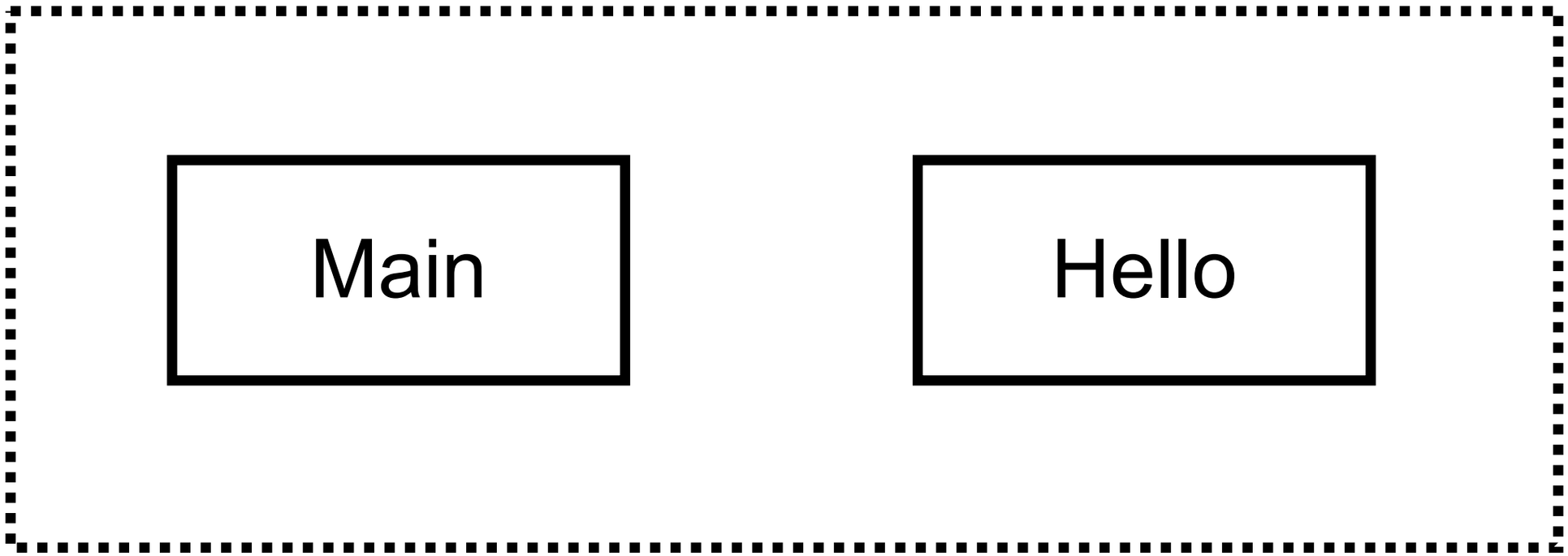
Very cheap

Not a thread

May be only a thread with 1,000 of goroutines

Example

Independently executing function



Example

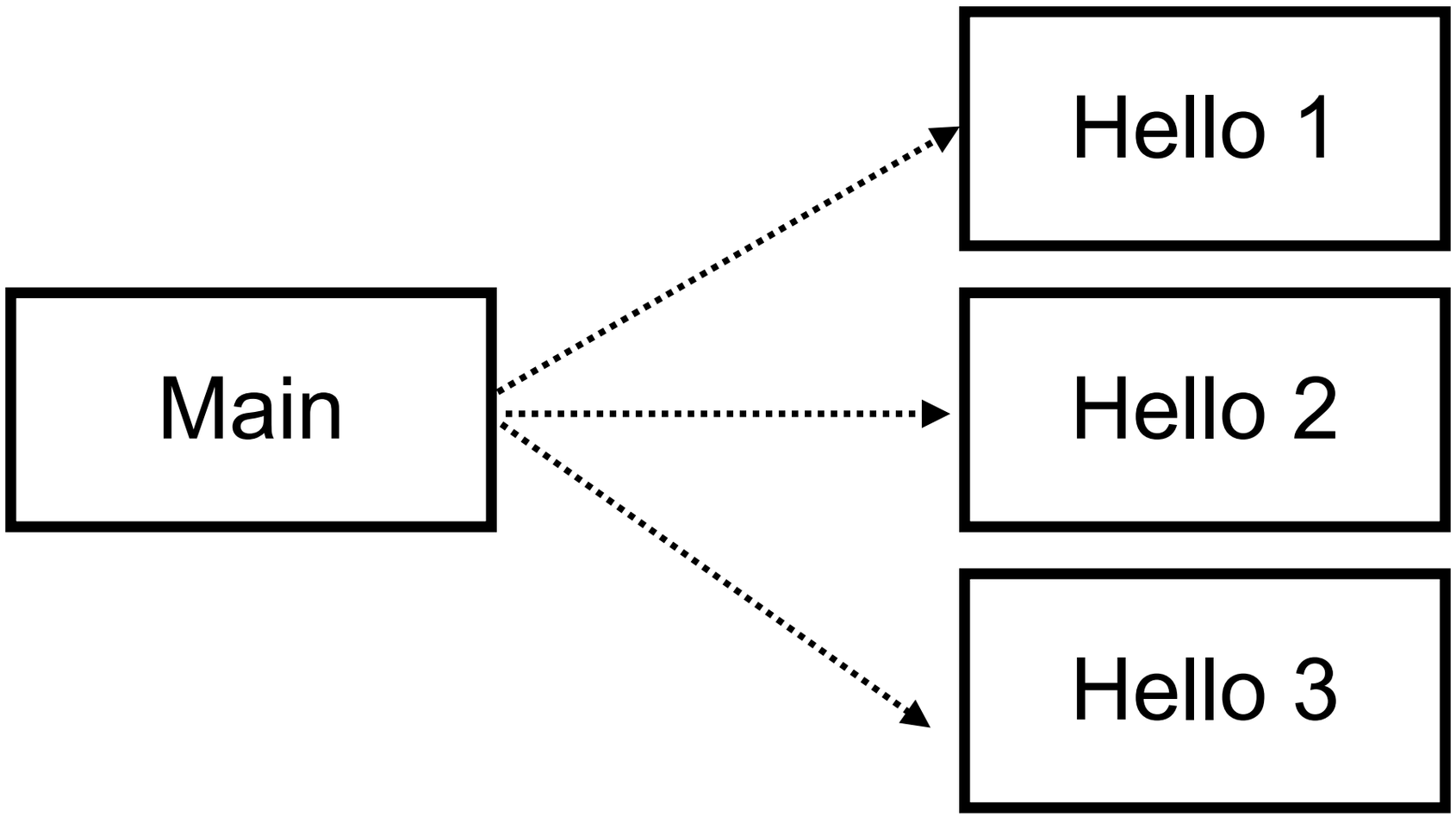
```go
func hello(name string) {
    for i := 0; ; i++ {
        fmt.Println(name, i)
        time.Sleep(time.Duration(rand.Intn(1e3)) * time.Millisecond)
    }

}
```

```go
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

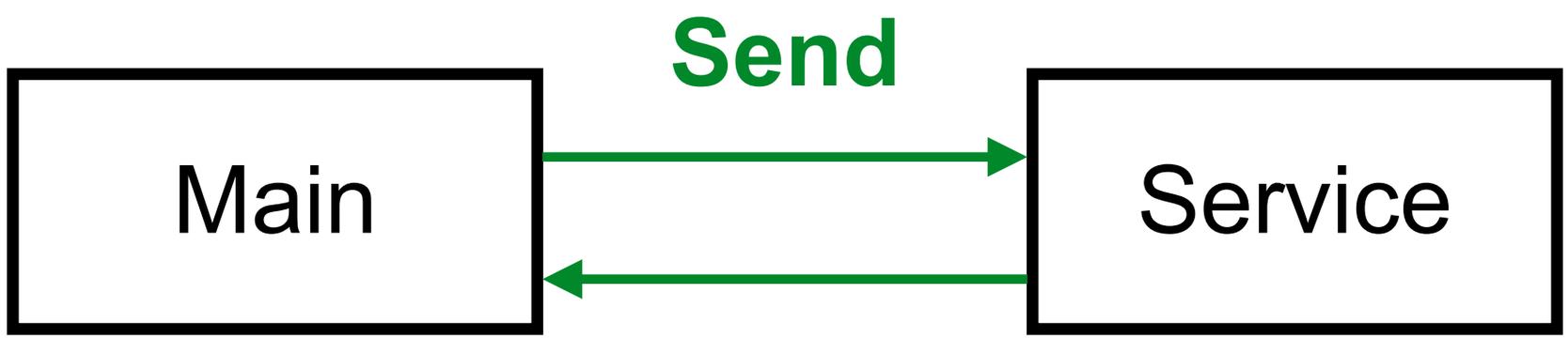
Provides a connection between 2 go routines
Allow them to communicate

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



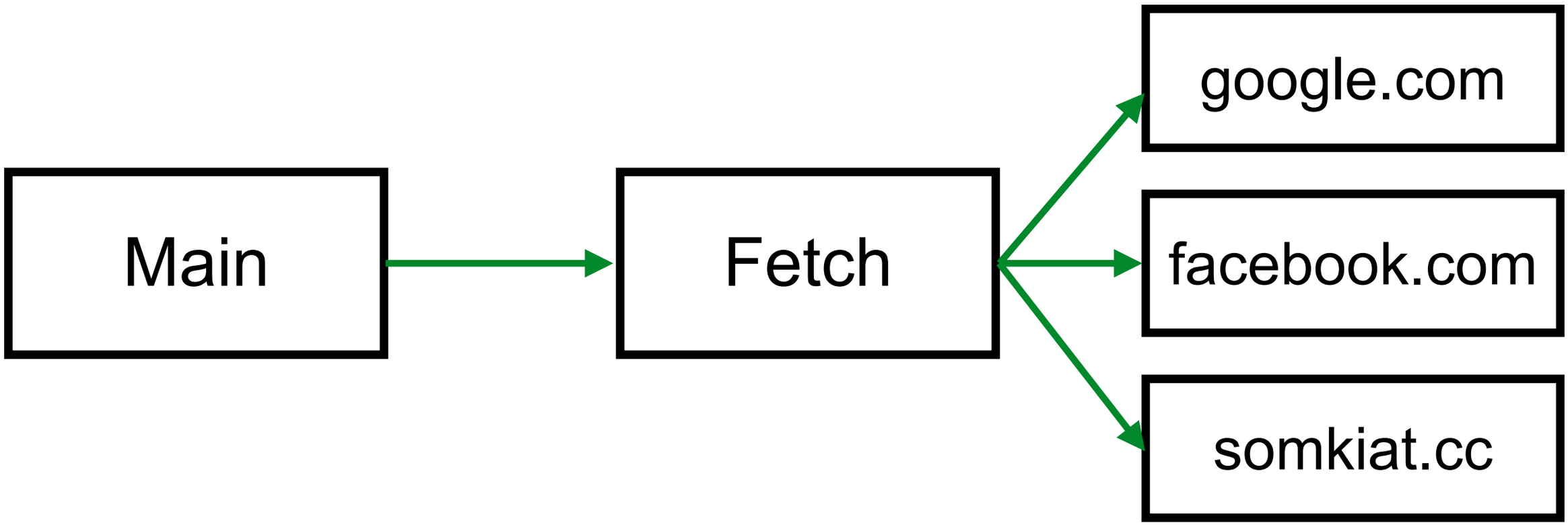
Example

```
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



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

RESTful API

Building REST APIs

HTTP + JSON

Go provide **net/http** package

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

REST

REpresentational **S**tate **T**ransfer

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

Hello API Server

```go
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>

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)
}
```

Web framework

Testing RESTful APIs

Testing

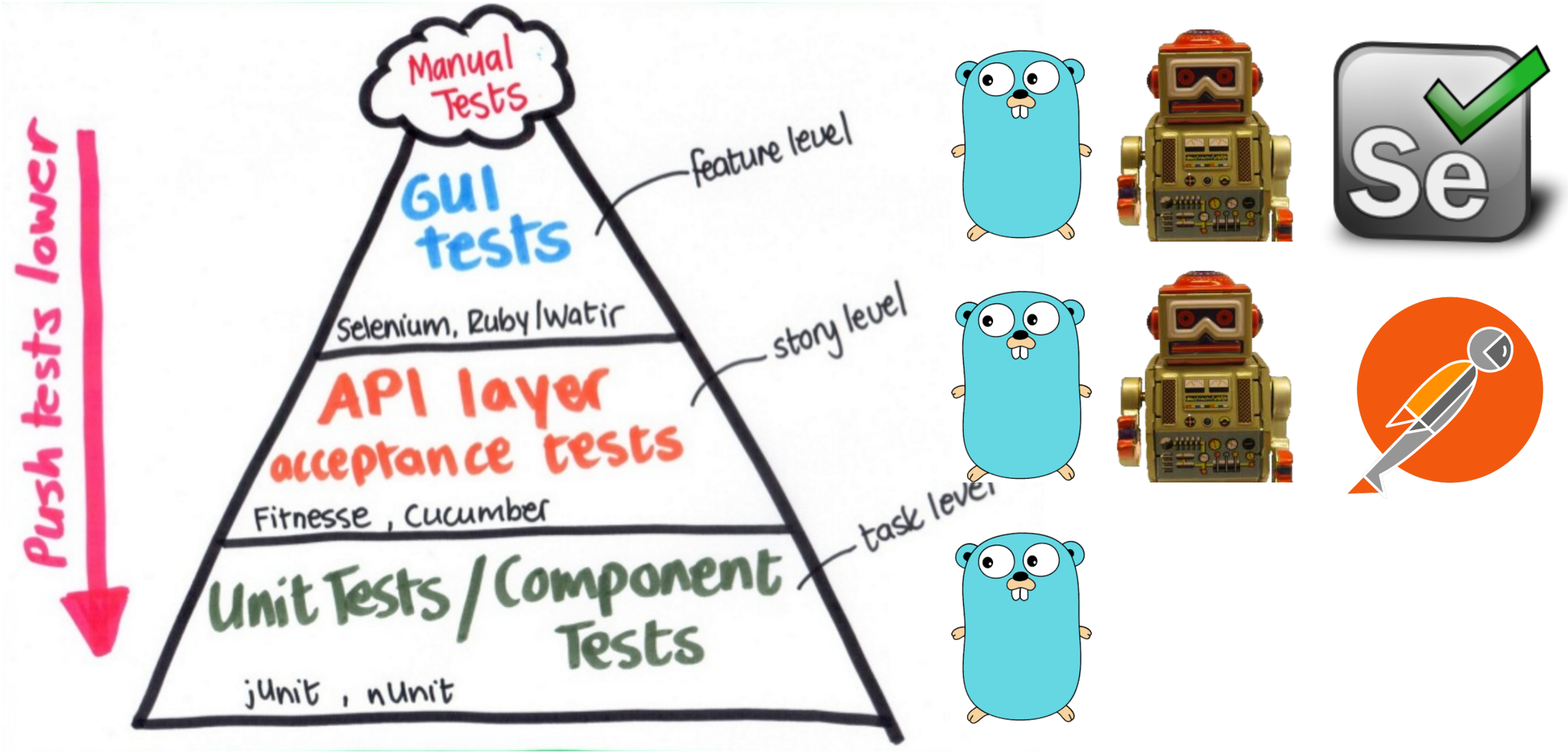
net/httptest

Postman

Robotframework

Cotton

Testing



Using net/httptest

```go
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())
    }
}
```

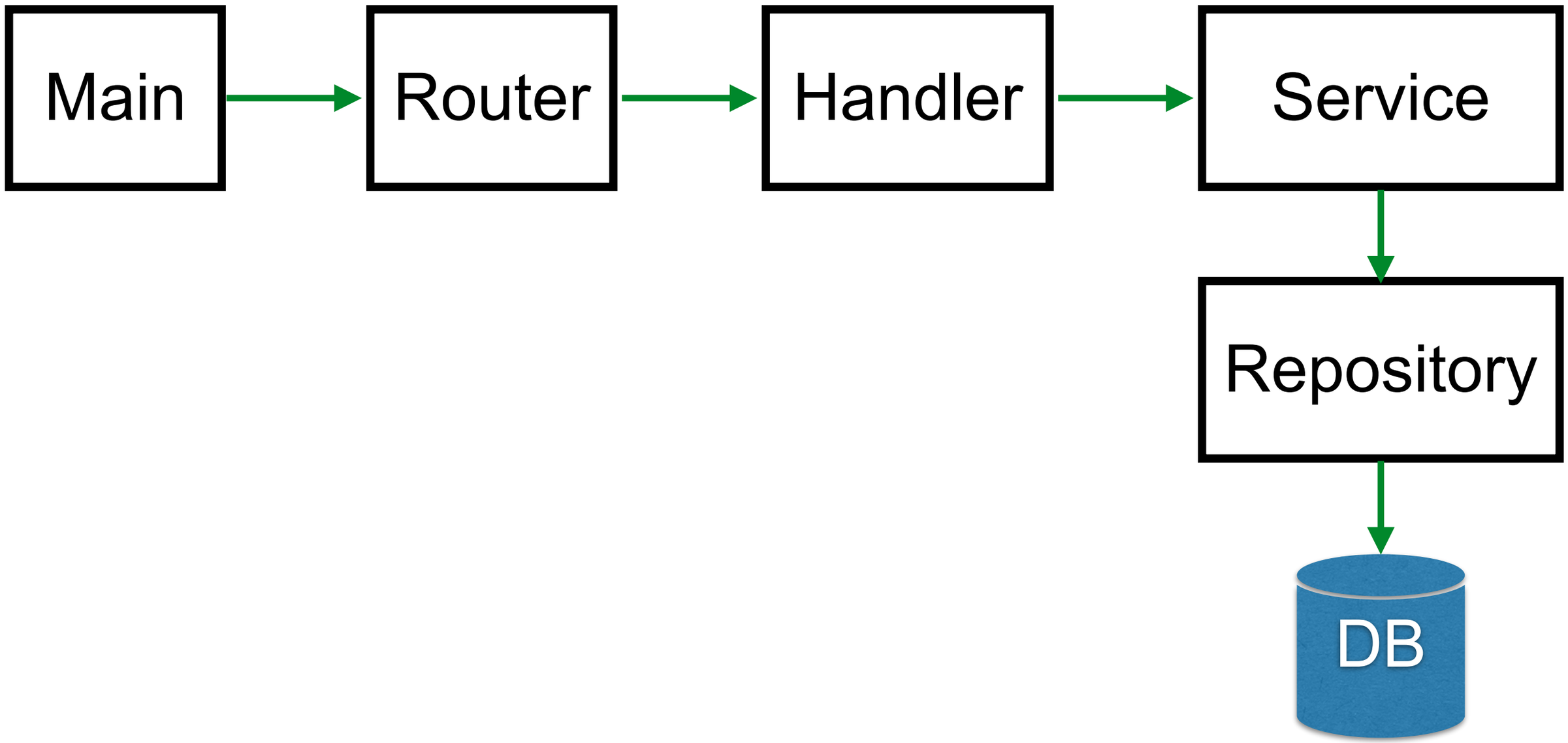
Testing JSON response

```go
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))
    }
}
```

Workshop with RESTful API

Better structure

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

POST http://localhost:8080/todo/ Params Send

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

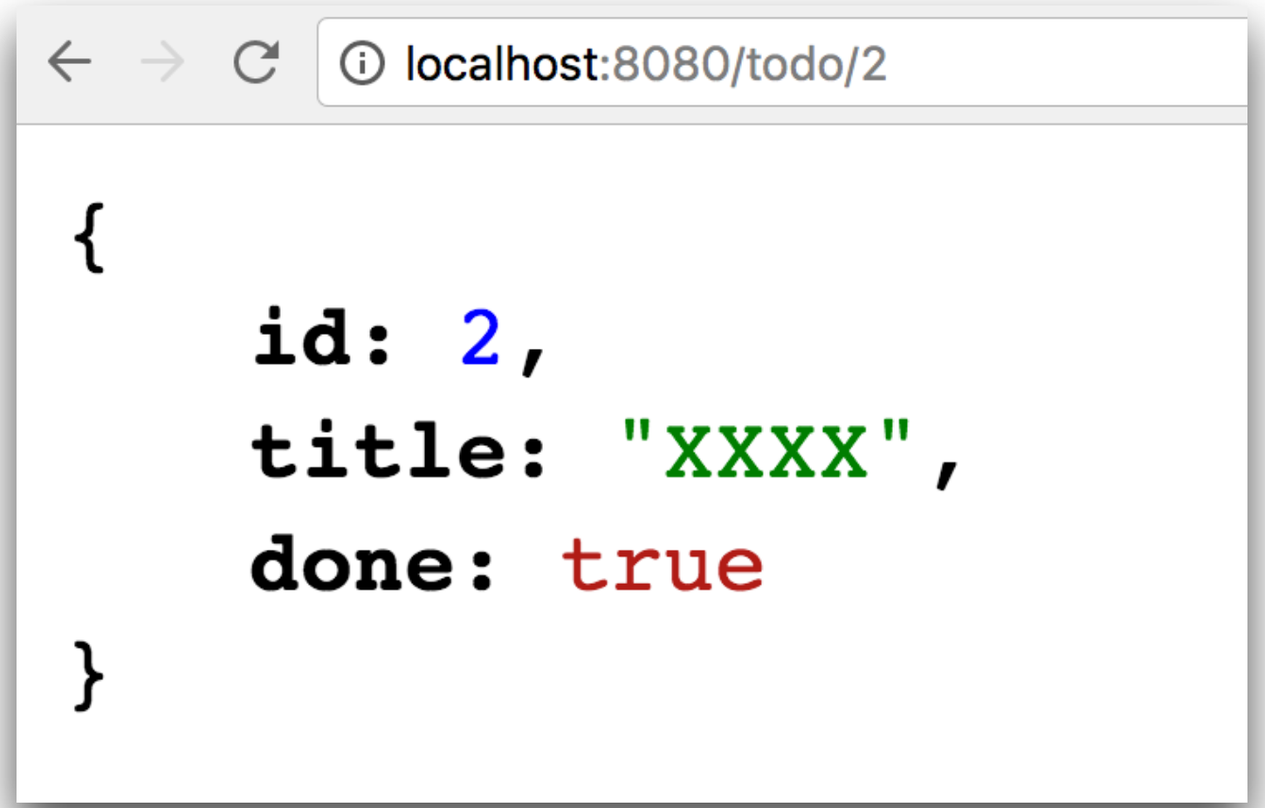
```
{
    "title": "test",
    "done": true
}
```

Body Cookies (25) Headers (3) Tests Status: 200 OK

Pretty Raw Preview JSON

```
{
  "id": 0,
  "title": "test",
  "done": true
}
```

Get TODO by ID



Let’s Go !!