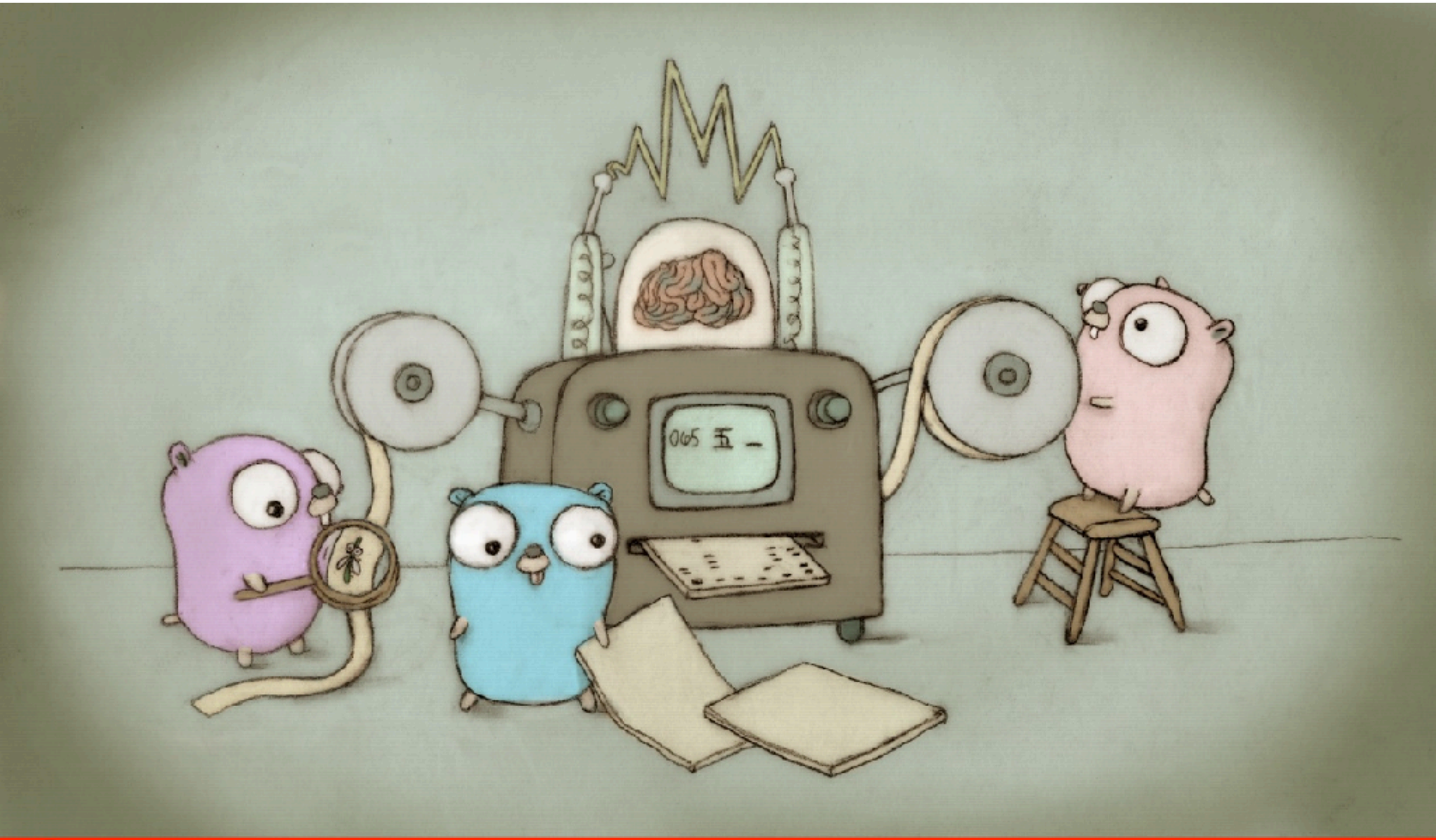
TDD with Golang



https://www.facebook.com/somkiatspns

Somkiat Puisungnoen

Somkiat | Home

Somkiat Puisungnoen

Update Info 2 | View Activity Log 10+ | •••

Timeline | About | Friends 2,604 | Photos | More

สยามชำนาญกิจ

SIAM CHAMNAN KIT

https://www.facebook.com/somkiat.cc/

somkiat.cc

Somkiat | Home

Page | Messages | Notifications 1 | Insights | Publishing Tools | Settings

somkiat.cc

@somkiat.cc

Home

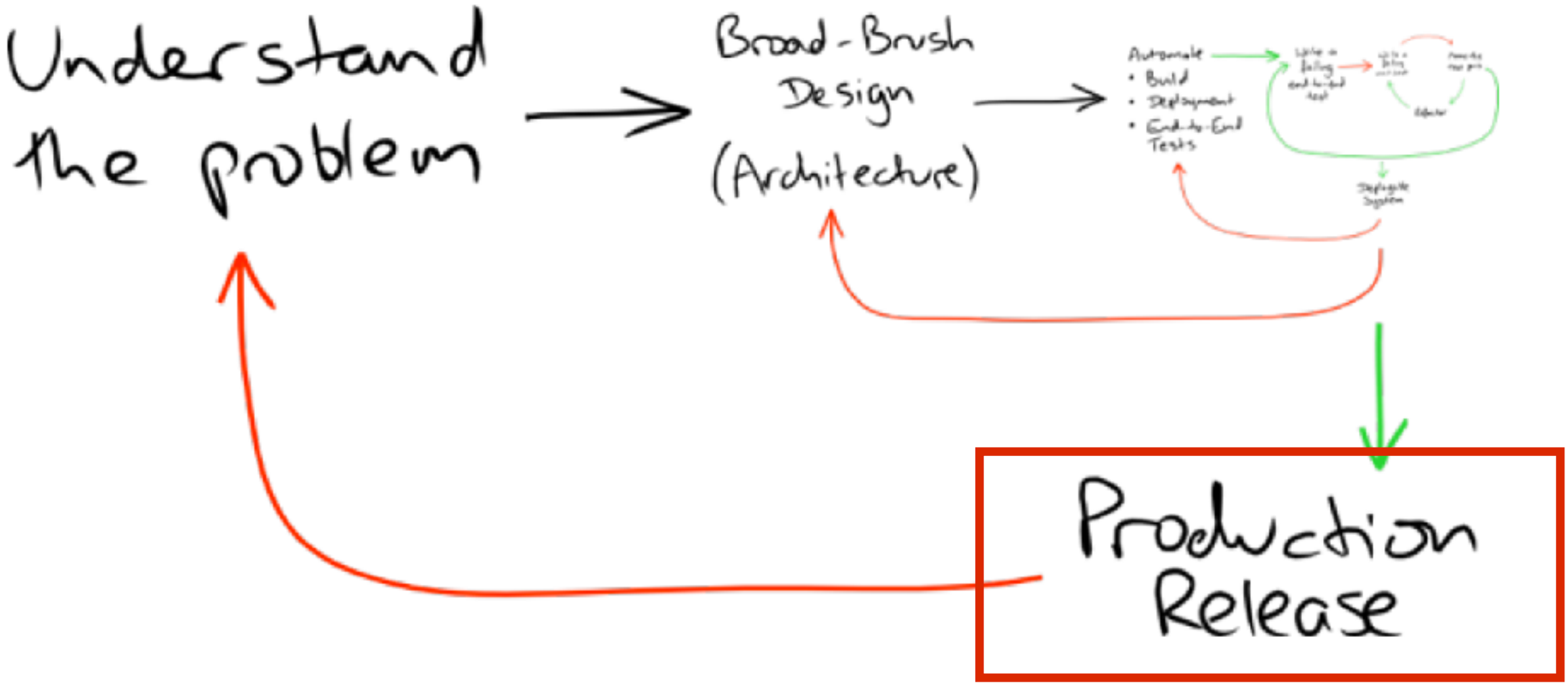
About

Agenda

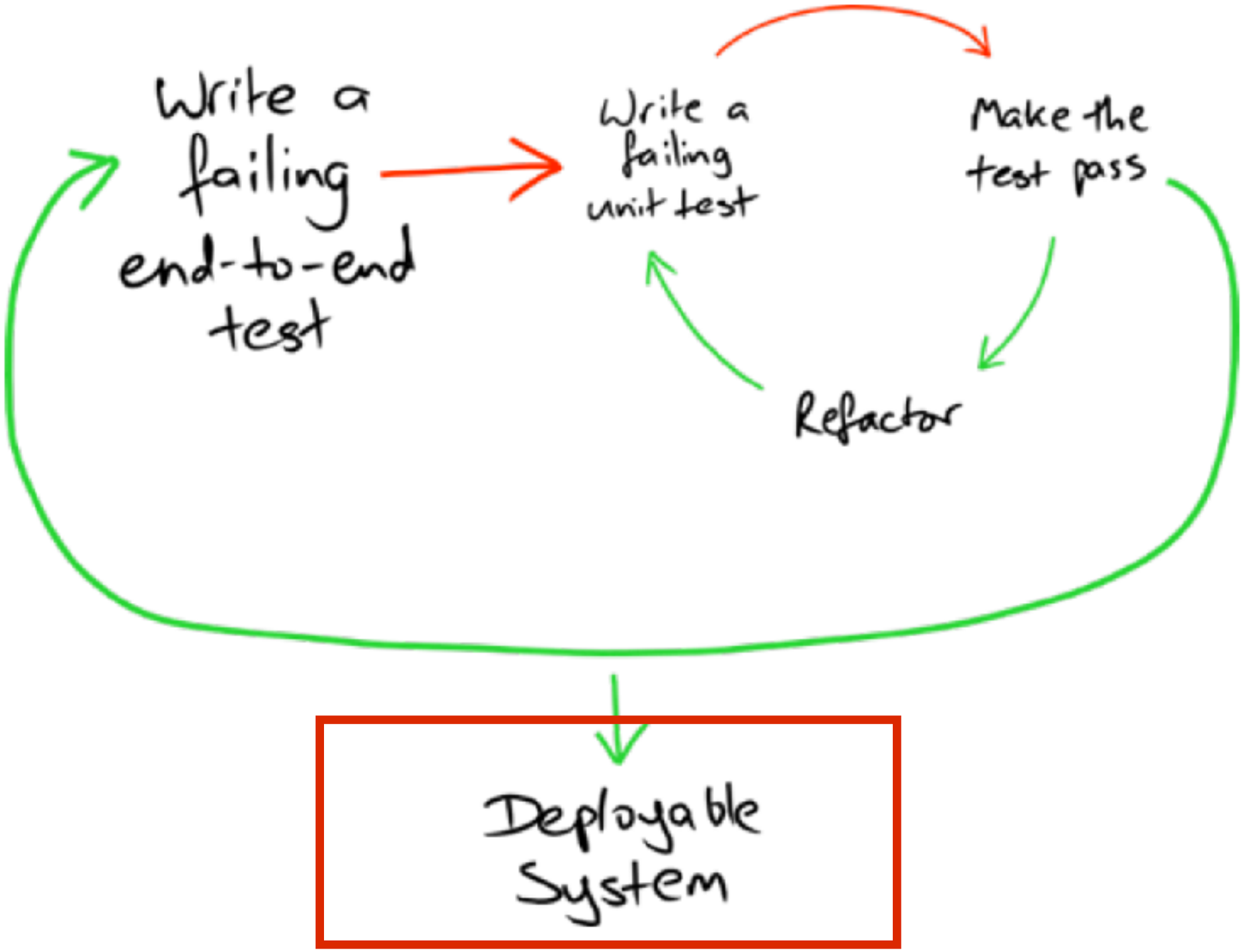
Basic of TDD
Basic of Go
Using interface
Building REST APIs
Testing REST APIs

TDD

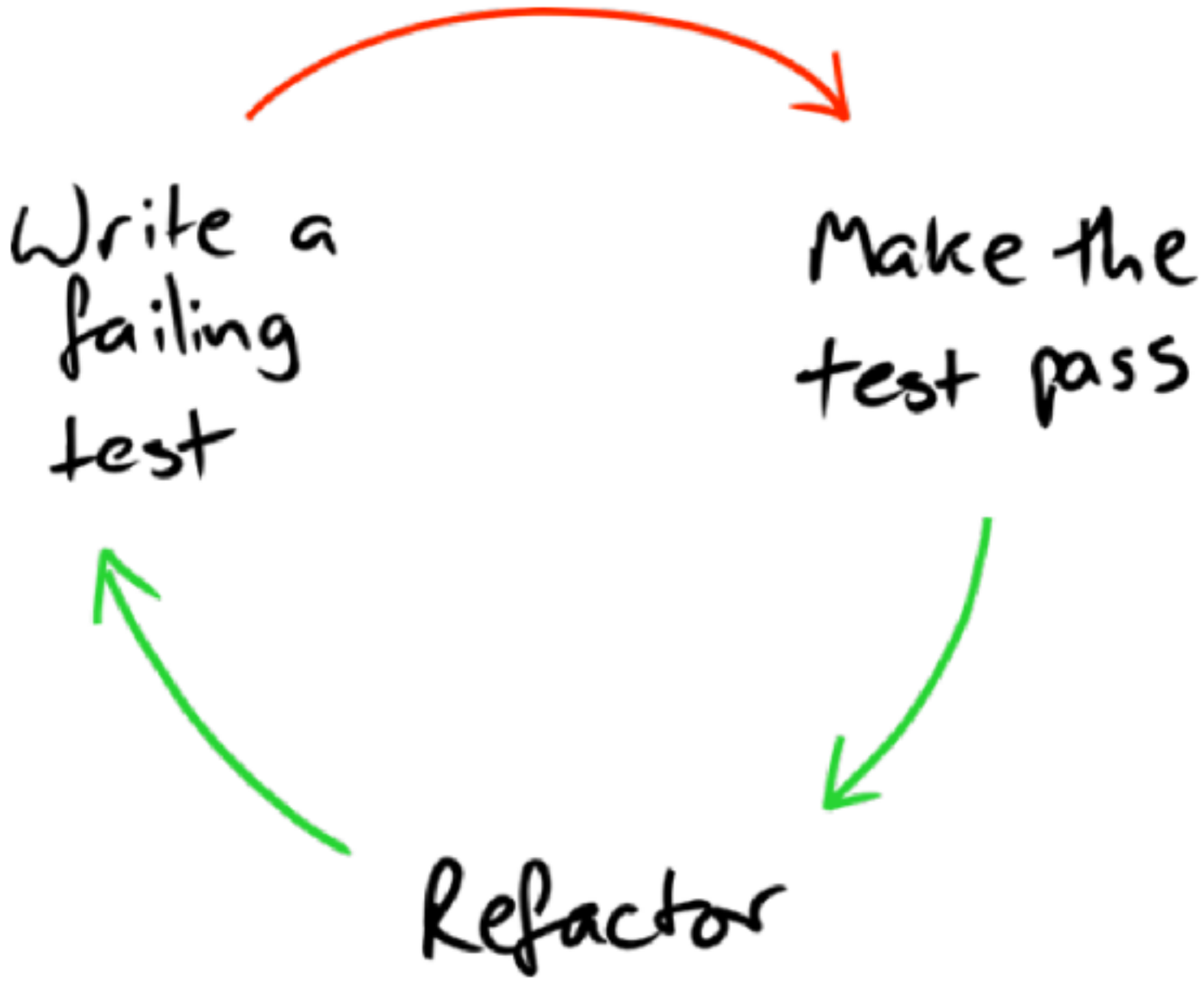
LARGER FEEDBACK LOOP



ACCEPTANCE TEST DRIVEN DEVELOPMENT

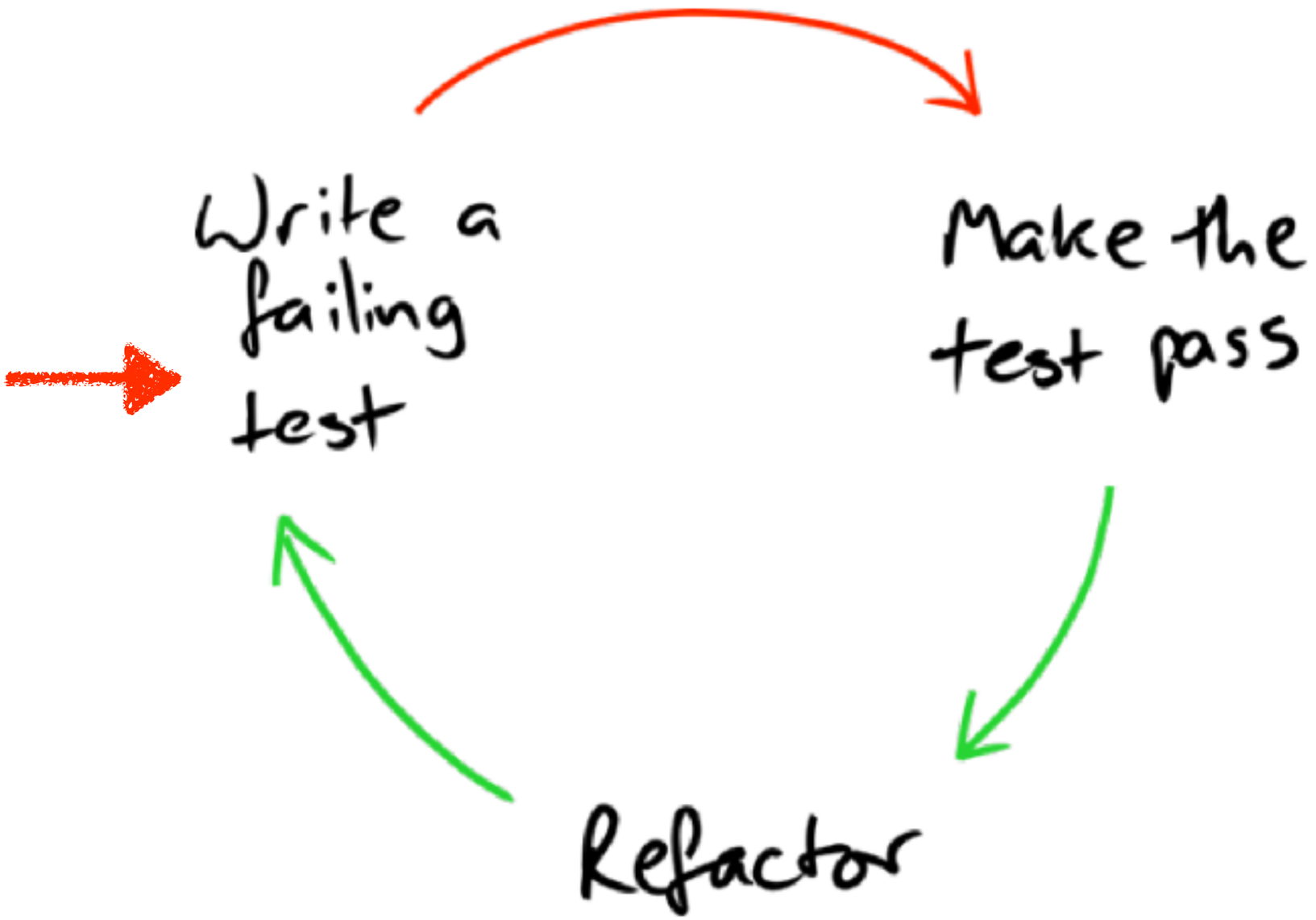


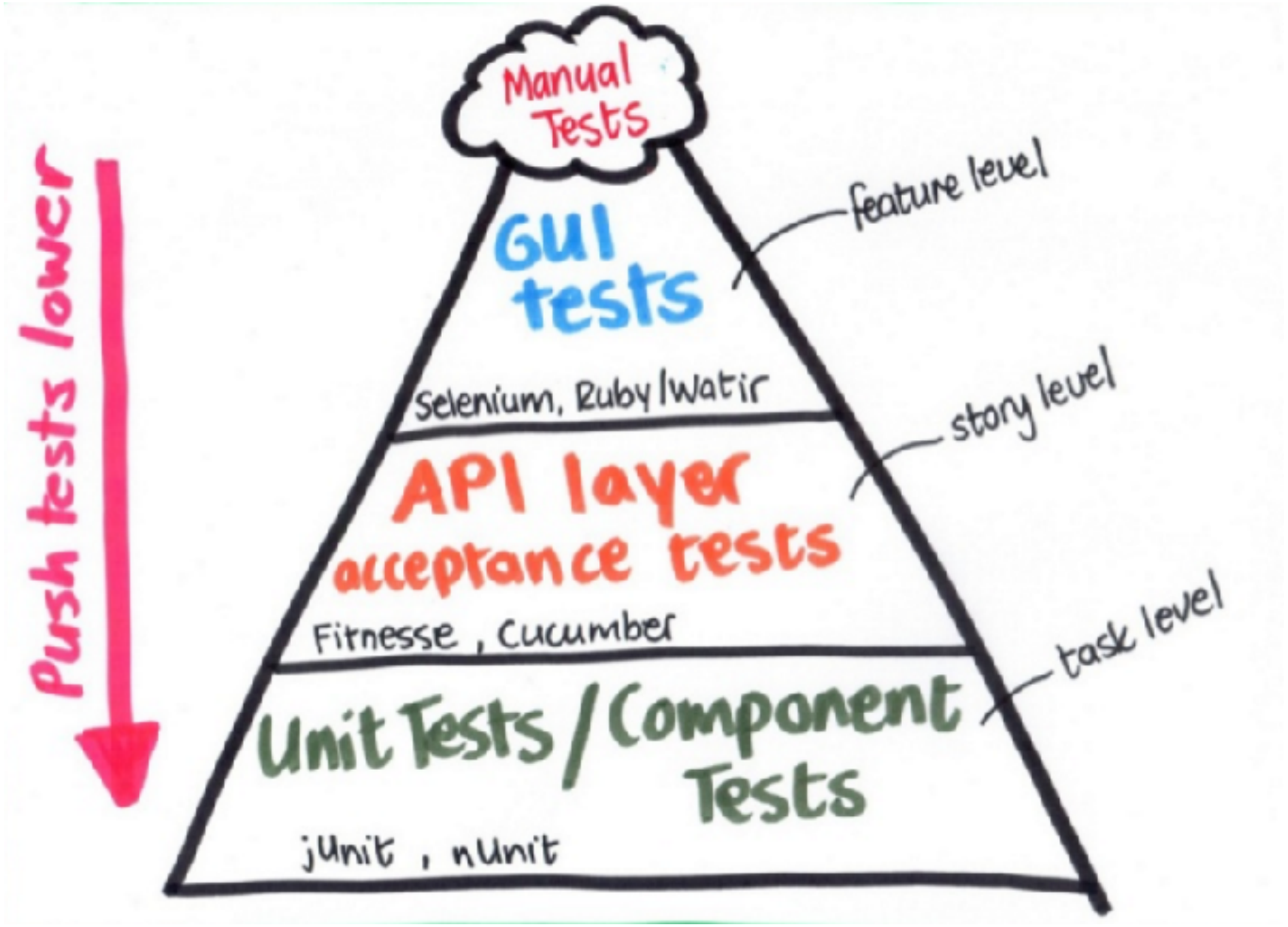
TEST DRIVEN DEVELOPMENT



TEST DRIVEN DEVELOPMENT

Start with THINK





http://www.slideshare.net/growingagile/expoqa-tutorial-agile-testing-techniques-for-the-whole-team

Disadvantage of TDD

Increase development time
Increase complexity
Design changes

Basic of Go

Summary of Go

Modern, **fast**
Powerful of standard library
Concurrency build-in
Use **interface** as the building block of code

Hello Go

```go
package main

import "fmt"

func main() {
  message := "Hello world"
  fmt.Printf("%s", message)
}
```

Running program

$go run hello.go

$go build hello.go

$hello

Go document

$**godoc** fmt Printf

```
use 'godoc cmd/fmt' for documentation on the fmt command

func Printf(format string, a ...interface{}) (n int, err error)
    Printf formats according to a format specifier and writes to standard
    output. It returns the number of bytes written and any write error
    encountered.
```

Types

Number (int, float, complex)

String

Boolean (true, false)

Types

```
func main() {
   i := 1
   f := 1.0
   s := "Hello"
   b := true
   fmt.Printf("Type %T has value %d\n", i, i)
   fmt.Printf("Type %T has value %.2f\n", f, f)
   fmt.Printf("Type %T has value %s\n", s, s)
   fmt.Printf("Type %T has value %v\n", b, b)
}
```

Variables

Number (int, float, complex)

String

Boolean (true, false)

Declaration variable

Using **var** keyword

```
func main() {
   var a string = "first"
   var b string
   b = "second"

   var c = "third"

   d := "forth"
}
```

Constant

Using **const** keyword

```
package main

import "fmt"

func main() {
   const name string = "Somkiat"
   fmt.Println(name)
}
```

Multiple variables

```
func main() {
  var (
    a = 1
    b = 2
    c = 3
  )

  fmt.Println(a, b, c)
}
```

Multiple variables

```
func main() {
  var a, b, c = 1, 2, 3

  fmt.Println(a, b, c)
}
```

Multiple variables

```
func main() {
  a, b, c := 1, 2, 3

  fmt.Println(a, b, c)
}
```

Control flow statement

For

If

Switch

For

```
func main() {

   for i := 0; i<10; i++ {
      fmt.Println(i)
   }
```

```
   var i = 0
   for i < 10 {
      fmt.Println(i)
      i++
   }
}
```

Switch

```
func main() {
   i := 1

   switch i {
   case 0: fmt.Println("case 0")
   case 1: fmt.Println("case 1")
   case 2: fmt.Println("case 2")
   case 3: fmt.Println("case 2")
   default: fmt.Println("Unknow number")
   }
}
```

Data structure

Array

Slice

Map

Array

```
func main() {
  var data [10]string
  for i := 0; i<len(data); i++ {
    data[i] = strconv.Itoa(i+1)
  }

  for i, value := range data {
    fmt.Println(i, value)
  }

}
```

Array

```
func main() {
  var data [10]string
  for i := 0; i<len(data); i++ {
    data[i] = strconv.Itoa(i+1)
  }

  for _, value := range data {
    fmt.Println(value)
  }

}
```

Array trick !!

```
func main() {

   x := [4]int {
      1,
      2,
      3,
      //4,
      5,
   }

   fmt.Println(len(x))

}
```

Slice

Segment of an array

Look like an array BUT

Length of slice can be change !!

Create Slice

Using **make** function

```
func main() {
   var x = []int  // len = 0
   x := make([]int, 5)
   fmt.Println(len(x), cap(x))

   y := make([]int, 5, 10)
   fmt.Println(len(y), cap(y))
}
```

Create Slice

```
x := make([]int, 5, 10)
```

Create Slice from array

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

  fmt.Printf("%v\n", arr[0:5])
  fmt.Printf("%v\n", arr[0:len(arr)])
  fmt.Printf("%v\n", arr[0:])
  fmt.Printf("%v\n", arr[:5])
  fmt.Printf("%v\n", arr[:])

  fmt.Printf("%v\n", arr[1:5])
  fmt.Printf("%v\n", arr[1:4])
}
```

Slice functions

2 Build-in function => **append**, copy

```
func main() {
  slice1 := []int {1, 2, 3}
  slice2 := append(slice1, 4, 5)

  fmt.Printf("%v\n", slice1)
  fmt.Printf("%v\n", slice2)
}
```

Slice functions

2 Build-in function => append, **copy**

```
func main() {
   slice1 := []int{1, 2, 3}
   slice2 := make([]int, 2)

   copy(slice2, slice1)

   fmt.Println(slice1, slice2)
}
```

Map

An **unordered** collection of **key-value** pair

key	value
firstname	Somkiat
lastname	Puisungnoen
gender	Male

Create Map

Using **map** and **make**

```
func main() {
   x := make(map[string]string)
   x["firstname"] = "Somkiat"
   x["lastname"] = "Puisungoen"
   x["gender"] = "Male"

   fmt.Println(x["firstname"])
   fmt.Println(x["lastname"])
   fmt.Println(x["gender"])
}
```

Iterate data in map

```
func main() {
   x := make(map[string]string)
   x["firstname"] = "Somkiat"
   x["lastname"] = "Puisungoen"
   x["gender"] = "Male"

   for key, value := range x {
      fmt.Println(key, value)
   }
}
```

Check data in map

```
func main() {
   x := make(map[string]string)
   x["firstname"] = "Somkiat"
   x["lastname"] = "Puisungoen"
   x["gender"] = "Male"

   if x["firstname"] == "" {
      fmt.Println(x["firstname"])
   }

   if data, ok := x["firstname"]; ok {
      fmt.Println(data, ok)
   }
}
```

Delete data in map

delete(map, key)

Function

Declaration Function

```
func average(data []int) float64 {
  panic(“Not implement”)
}

func main() {
  data := []int{10, 2, 3, 4, 5}
  fmt.Println(average(data))
}
```

Return

```
func average(data []int) float64 {
  total := 0.0
  for _, v := range data {
    total += float64(v)
  }
  return total/float64(len(data))
}

func main() {
  data := []int{10, 2, 3, 4, 5}
  fmt.Println(average(data))
}
```

Name of return type

```
func average(data []int) (result float64) {
    total := 0.0
    for _, v := range data {
        total += float64(v)
    }
    result = total/float64(len(data))
    return
}
```

Return multiple values

```
func call() (int, int) {
  return 1, 2
}

func main() {
  x, y := call()
  fmt.Print(x, y)
}
```

Variadic function

```go
func add(ops ...int) int {
    total := 0
    for _, v := range ops {
        total += v
    }
    return total
}

func main() {
    fmt.Println(add())
    fmt.Println(add(1))
    fmt.Println(add(1, 2))
    fmt.Println(add(1, 2, 3))
}
```

Defer, Panic, Recover

Defer

Schedules a function call
to be run after the function completes

Defer

```
func first() {
  fmt.Println("Call first")
}

func second() {
  fmt.Println("Call second")
}

func main() {
  defer second()
  first()
}
```

Defer

Often used when resources need to be free

```
func main() {
   f, err := os.Open(“some_file.txt”)
   defer f.Close()

   if err != nil {
      fmt.Println("Start...")
   }
}
```

Advantage of Defer

Easy to understand

Keep **Close** near **Open**

Deferred functions are run even if error

Panic and Recover

Panic to cause a runtime error

Handle a runtime panic with **recover**

Panic and Recover

```
func main() {
  panic("PANIC")
  str := recover()
  fmt.Println(str)
}
```

Panic, Recover and Defer

```
func main() {
   defer func(){
      str := recover()
      fmt.Println(str)
   }()
   panic("PANIC")
}
```

More example

```
func f() {
  defer func(){
    str := recover()
    fmt.Println(str)
  }()
  x := []int{1, 2, 3}
  fmt.Println(x[10])
}

func main() {
  f()
  fmt.Println("TODO NEXT")
}
```

Workshop TDD with Go

Testing with Go

Testing with go

Build-in testing framework
Using **testing** package
Command **go test**

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

Hello Testing

```go
package main

import(
   "testing"
)
```

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

Hello Testing

```go
package main

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

Running test

Run tests for specified package
It defaults to package in **current directory**

```
$go test
$go test -v
```

Running test

Run tests for all my project

$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 driven test

Working with data driven testing

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

https://dave.cheney.net/2013/06/09/writing-table-driven-tests-in-go

Table driven test

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

}
```

Table driven test

```
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 coverage

Go tool can report test coverage statistic

$go test -cover

Workshop

Struct

Data value which contains name fields

```
type Point struct {
  x float64
  y float64
}

type Point struct {
  x, y float64
}
```

Initial Struct

```
type Point struct {
  x, y float64
}

func main() {
  p1 := Point{x: 1.0, y: 2.0}
  p2 := Point{2.0, 4.0}

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

Add function to struct

```
type Point struct {
  x, y float64
}
```

```
func (p *Point) someFunc() float64 {
  return math.Sqrt(p.x)
}
```

```
func main() {
  p := Point{x: 1.0, y: 2.0}
  fmt.Println(p.someFunc())
}
```

Interface

Types that just declare behavior

Problem ?

```
type Circle struct {
  x, y, r float64
}

func (c *Circle) area() float64 {
  return math.Pi * c.r * c.r
}

type Rectangle struct {
  w, h float64
}
func (r *Rectangle) area() float64 {
  return r.w * r.h
}
```

Solution ?

```
type Shape interface {
  area() float64
}
```

Solution ?

Implement function from interface

```
type Circle struct {
  x, y, r float64
}
```

```
func (c Circle) area() float64 {
  return math.Pi * c.r * c.r
}
```

Solution ?

Implement function from interface

```
type Rectangle struct {
  w, h float64
}
```

```
func (r Rectangle) area() float64 {
  return r.w * r.h
}
```

Solution ?

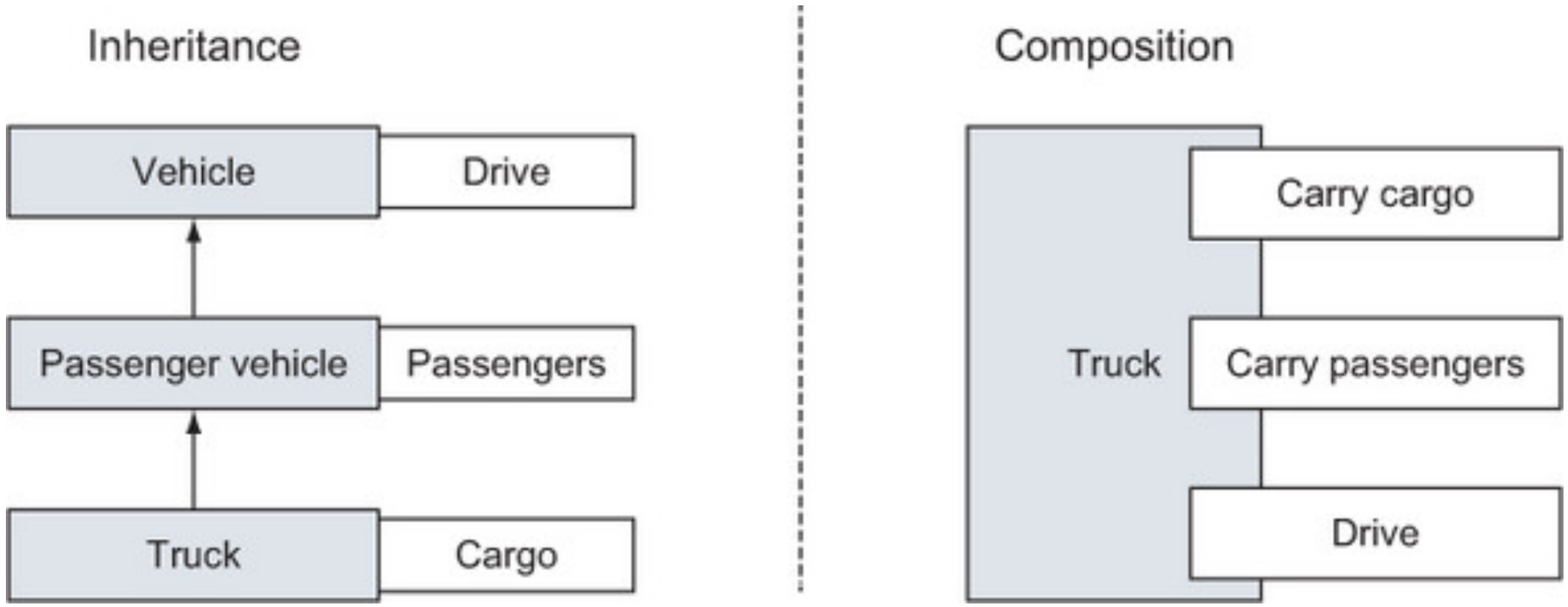
```
func main() {
    c := Circle{1, 1, 10}
    r := Rectangle{5, 10}
    shapes := []Shape{c, r}
    for _, s := range shapes {
        fmt.Println(s.area())
    }
}
```

Go’s Type System

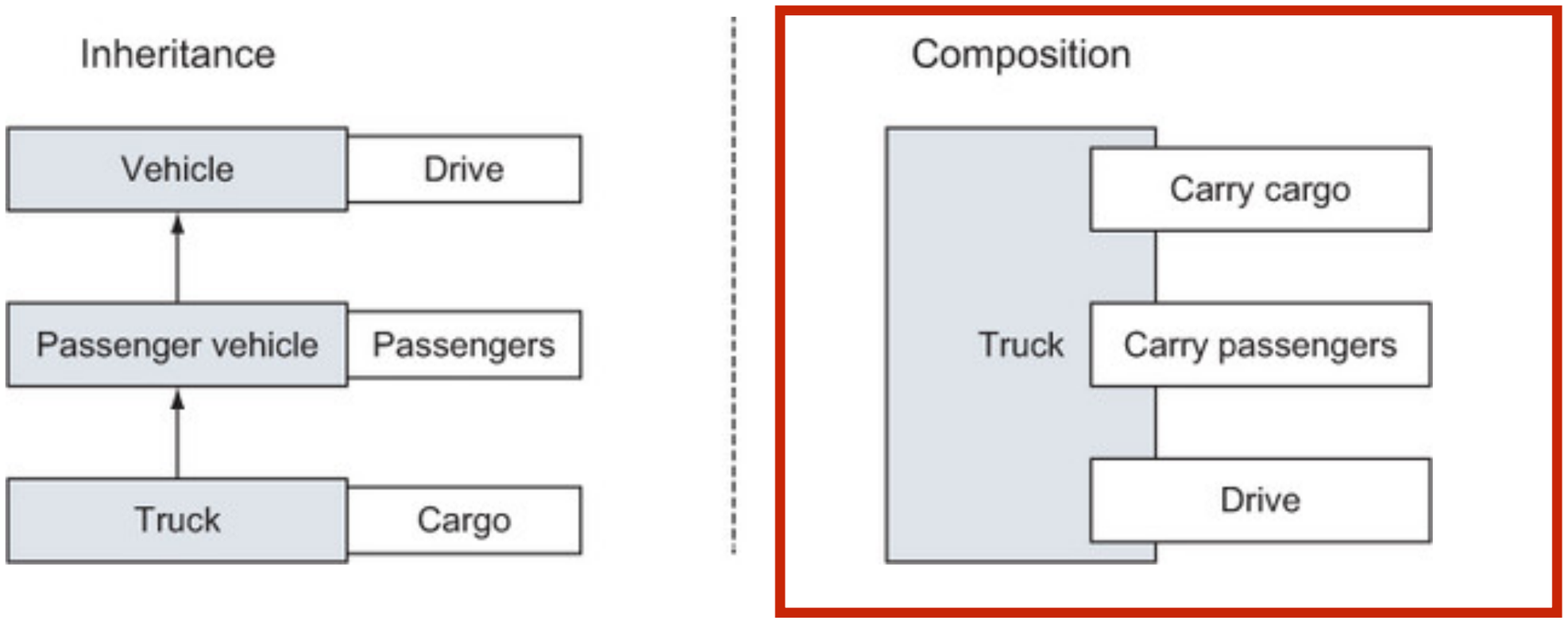
Type System

Types are **simple**
Types are **composed** of smaller types

Inheritance vs Composition



Inheritance vs Composition



Go Interface

Interface allow you to express
the **behavior** of a type

```
type Reader interface {
        Read(p []byte) (n int, err error)
}
```

https://golang.org/pkg/io/#Reader

Go Interface

Just **single** action/behavior

```
type Reader interface {
        Read(p []byte) (n int, err error)
}
```

https://golang.org/pkg/io/#Reader

Go Interface

Just single action/behavior

Smaller

Enable **reuse** and **composability**

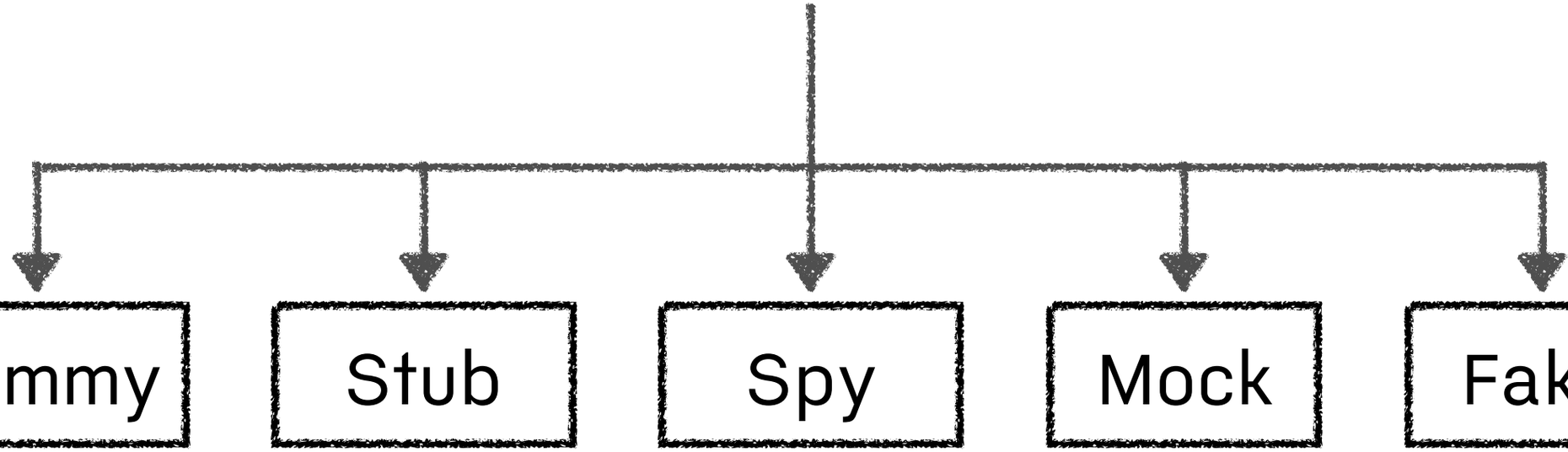
Example in go

Go Interface !!

Using a single interface
allow you to operate
on **data efficiently**

Test Double with Go

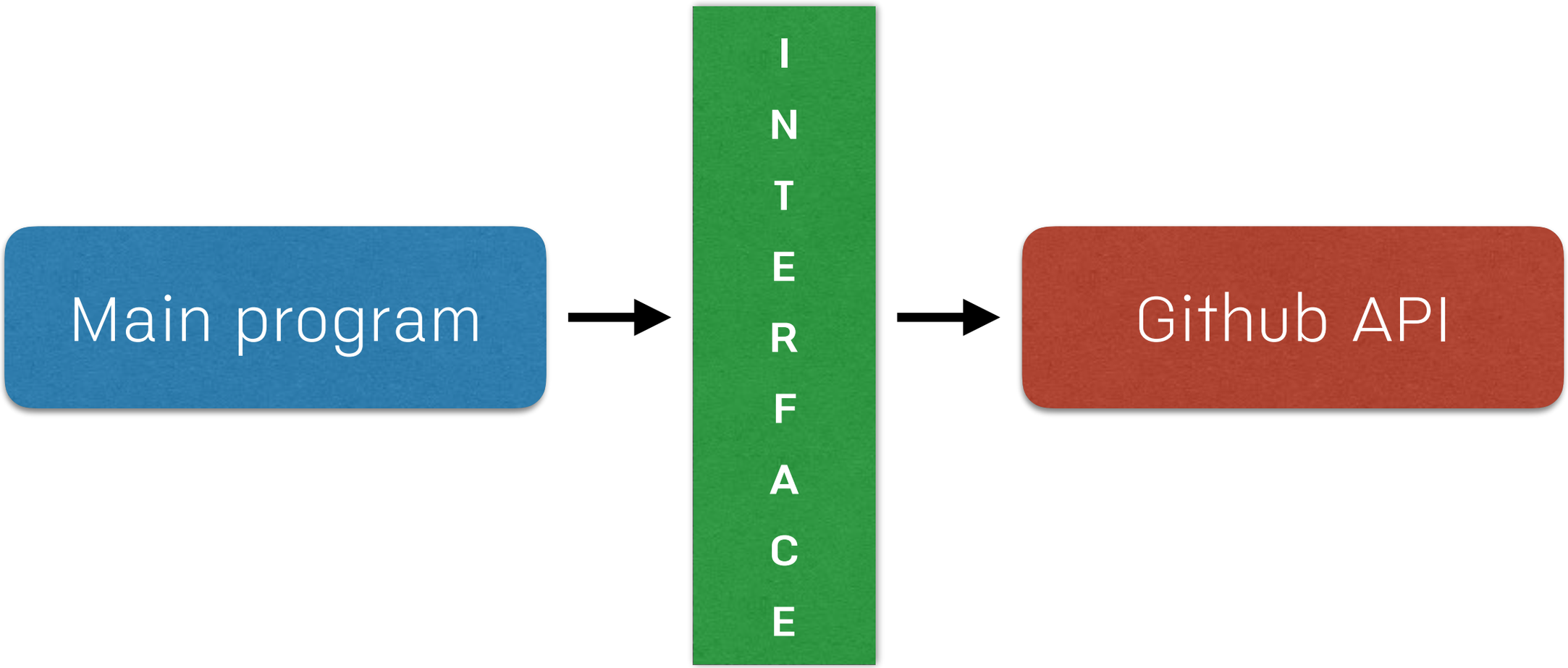
Test double



Problem

How to test ?

Solution



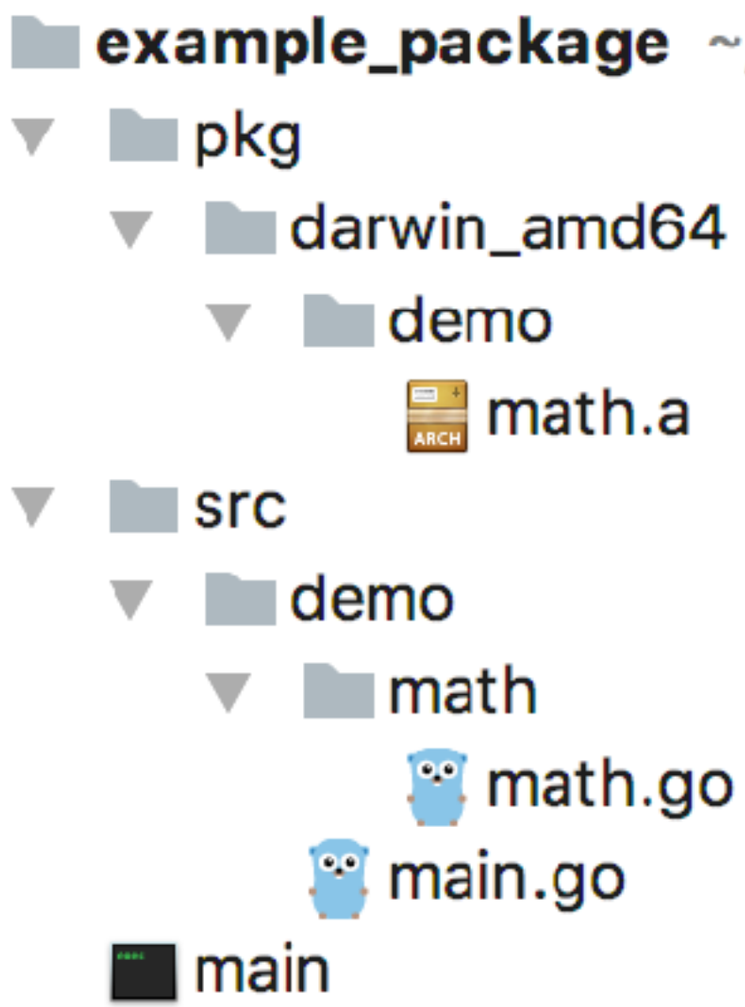
How to code ?

Package

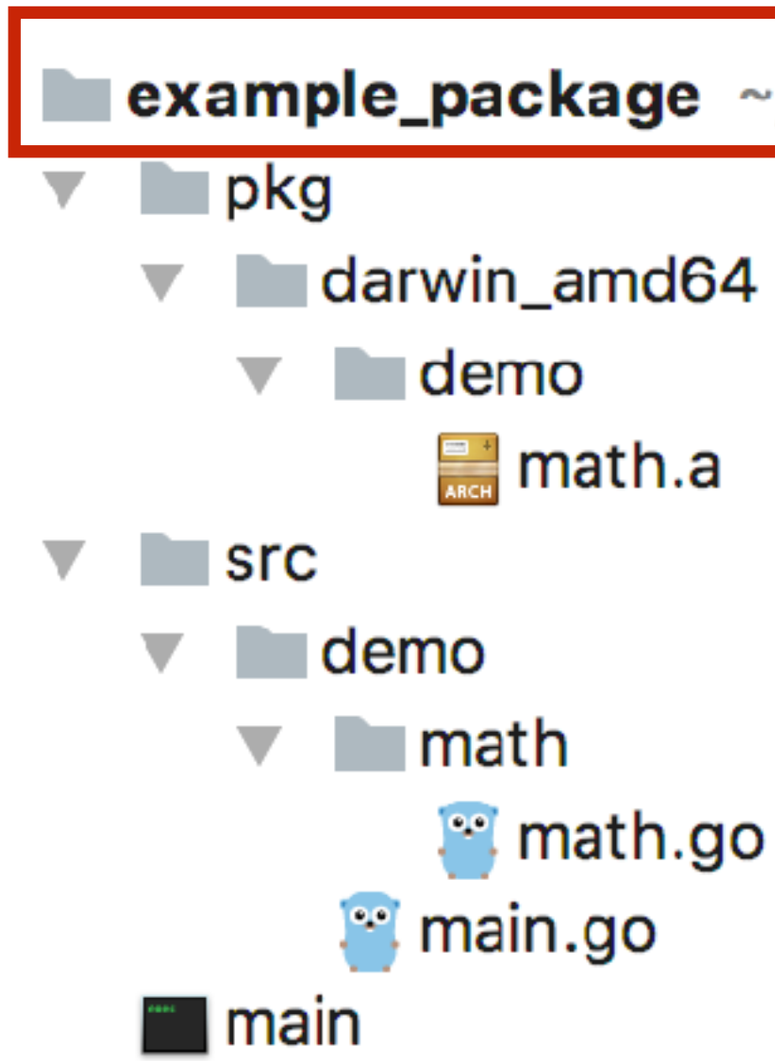
Package

Designed with **good** engineering practices
High quality software is **reuse**
Don’t Repeat Yourself (DRY)

Demo Package



Demo Package



Call GOPATH

Where is your GOPATH ?

$go env

Create my package

src/demo/main.go
src/demo/math/math.go

demo/math/Math.go

public function with **First capital letter**

```go
package math

func Average(datas []float64) float64 {
  total := float64(0)
  for _, v := range datas {
    total += v
  }
  return total / float64(len(datas))
}
```

demo/Main.go

```go
package main

import (
    "demo/math"
    "fmt"
)

func main() {
    datas := []float64{1, 2, 3 ,4 ,5}
    average := math.Average(datas)
    fmt.Println(average)
}
```

Install my package

$go install

Run my program

$go run main.go

Build my program

$go build main.go

Godoc my package

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

Workshop

REST APIs with Go

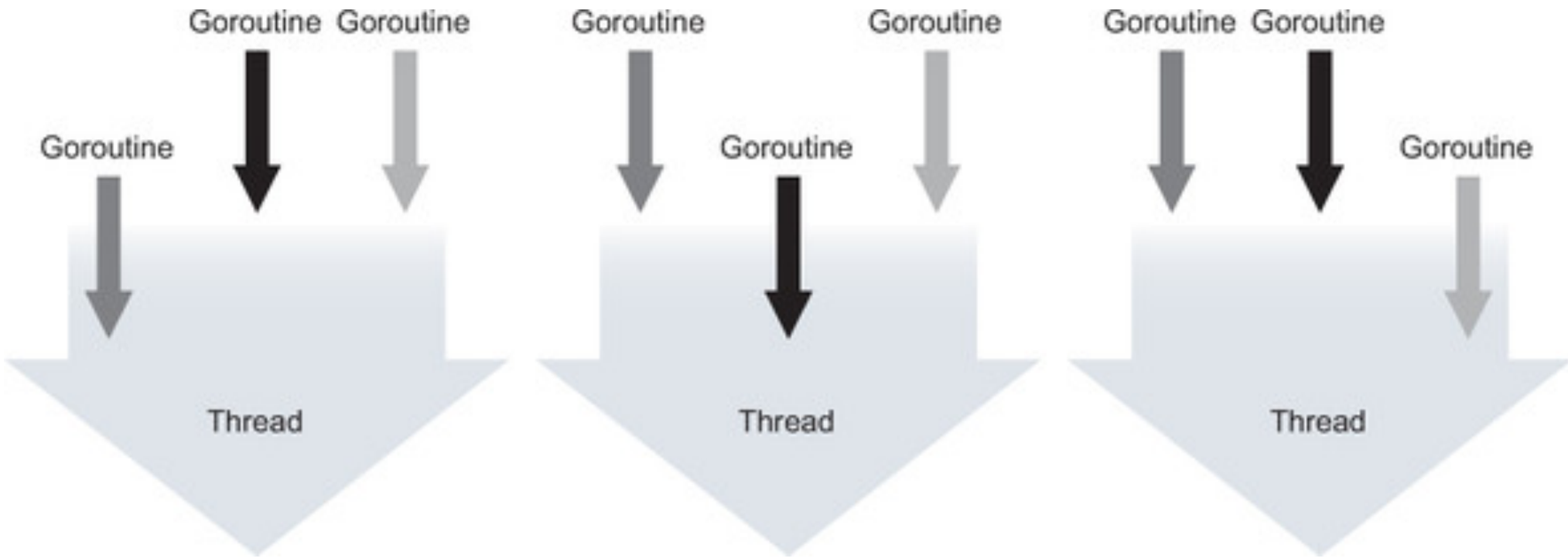
Testing REST APIs

Testing REST APIs

With Go
With silk
With Robotframework
With Postman

And more ...

Go routine


Goroutine
Goroutine
Goroutine
Thread
Goroutine
Goroutine
Goroutine
Thread
Goroutine
Goroutine
Goroutine
Thread

Go routine

```
package main

import "fmt"

func log(msg string) {
    //TODO
}


func main() {
   go log("Some message")
}
```

Go channel