THE WAY TO

0.0.0.0/ME

CÎRLIG ȘTEFAN

Go developer @ Ellation

@steevehook

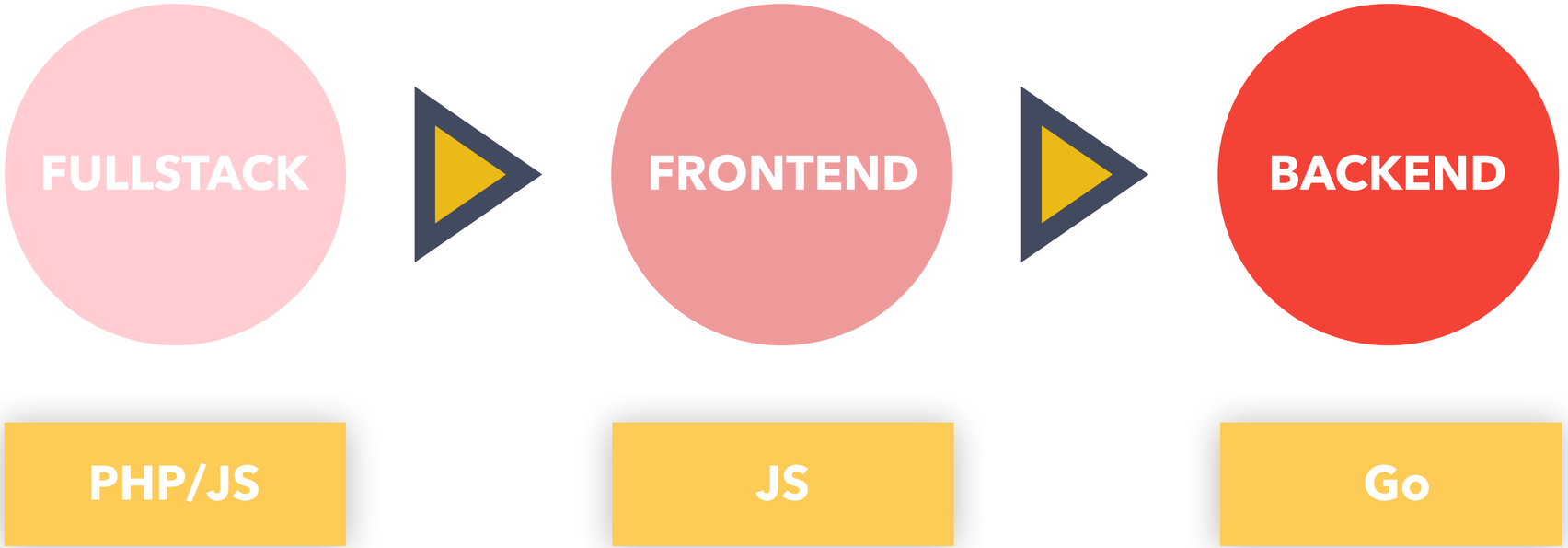
steevehookmd

@steevehook

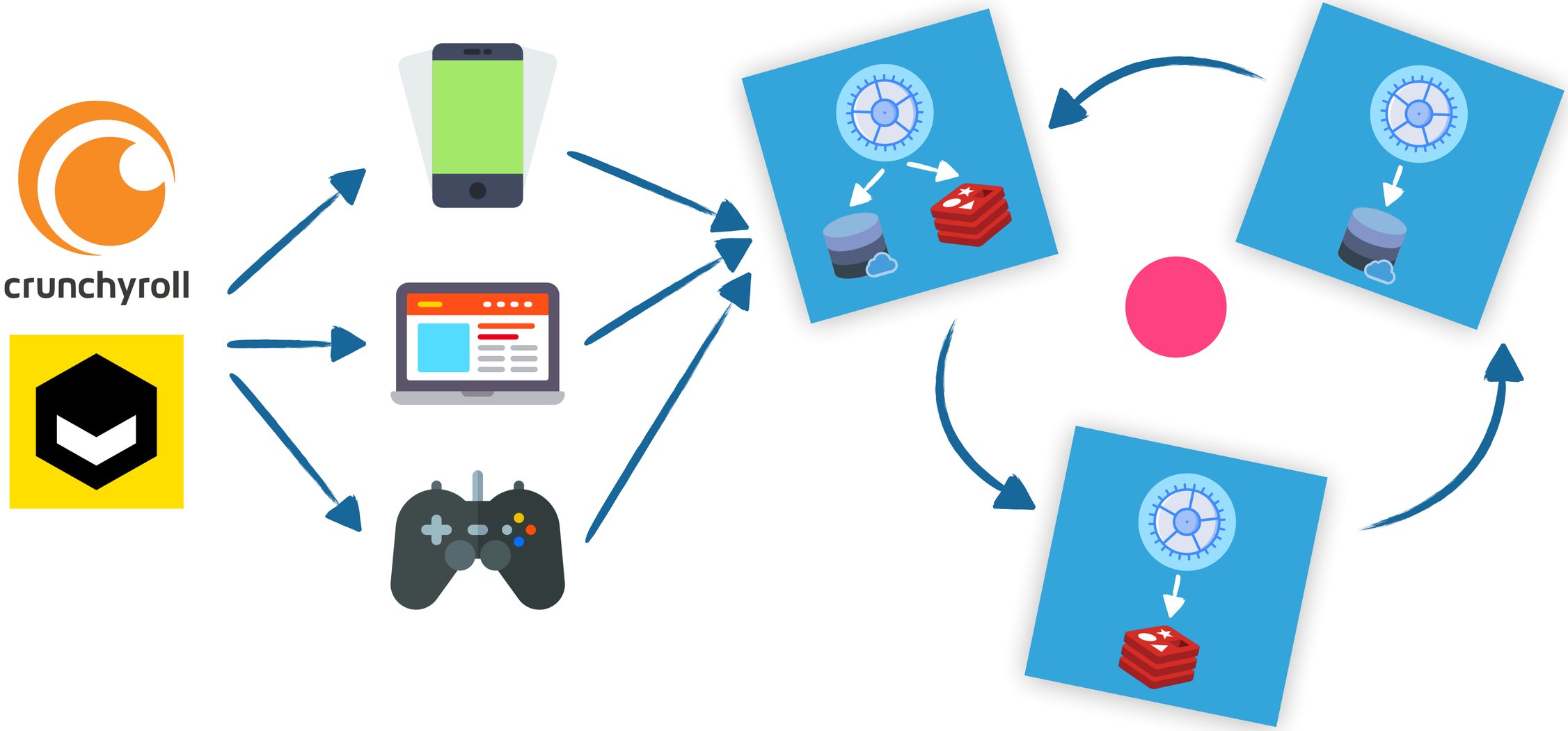
GopherTuts

steevehook/mdc19

MY JOURNEY



WHAT DO I DO @ ELLATION



MY EXPERIENCE SO FAR

9 months

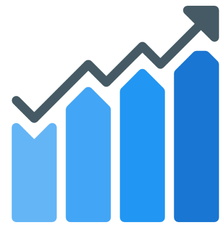
Perfect System?

"**Software engineering** is what happens to programming when you add **time** and **other programmers**"

Russ Cox

WILL YOUR COMPANY INVEST IN GO?

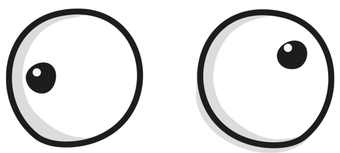
WHY THE F GO?



simplicity

readability

productivity



LET'S **GO**

HELLO WORLD

hello-world.go

```go
package main

import "fmt"

const (
    hello       = "你好"
    exclamation = "!"
)

var world string

func init() {
    world = "世界"
}

func main() {
    fmt.Println(hello + world + exclamation)
}
```

BASIC CONCEPTS

Types

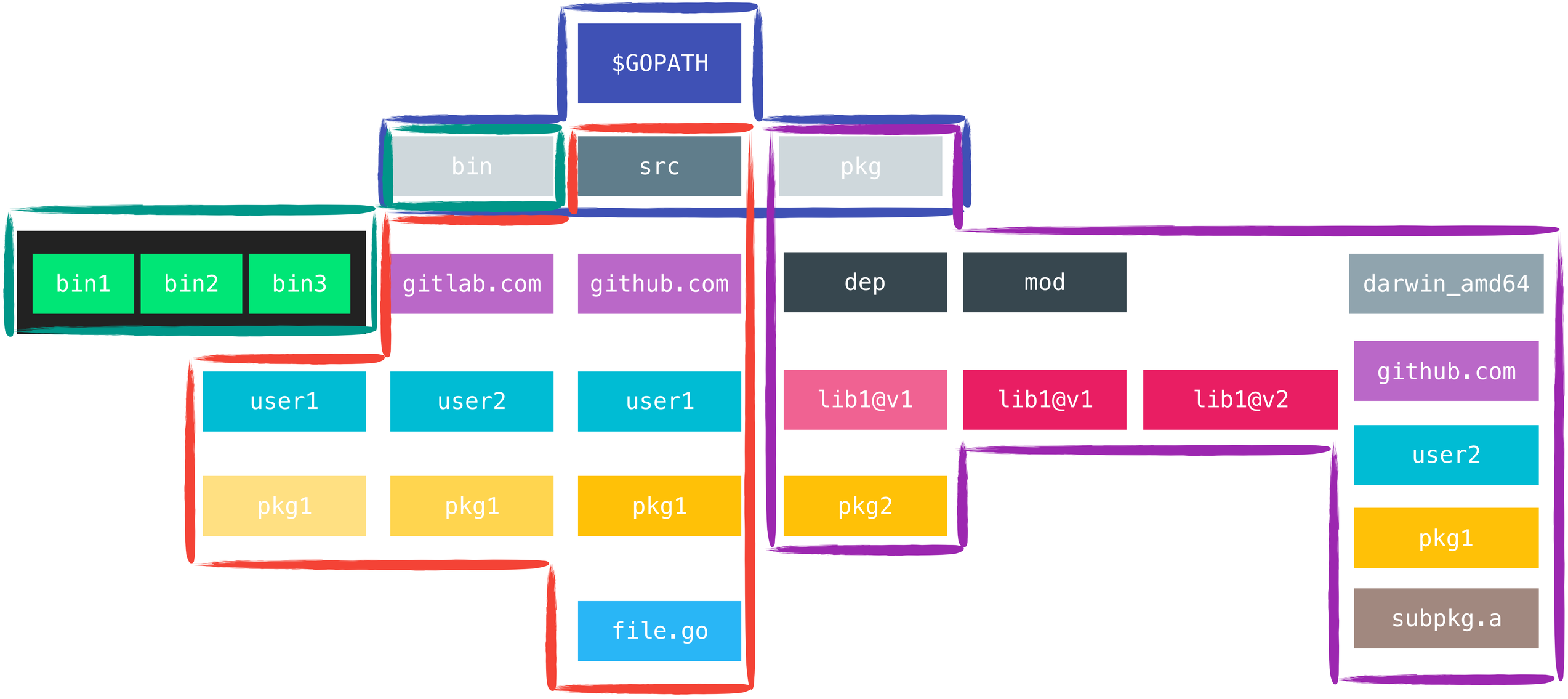
Functions

Pointers

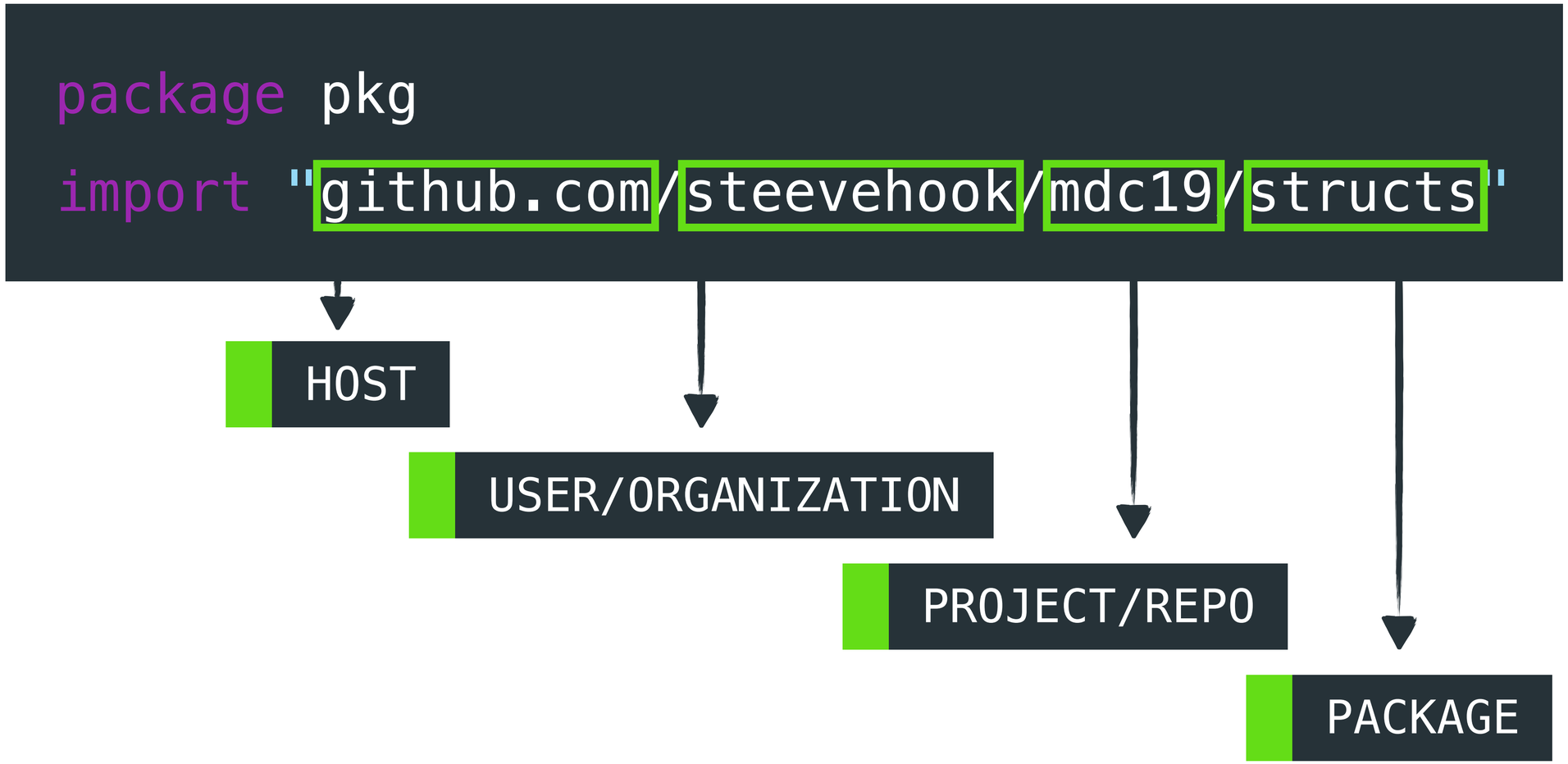
Interfaces

Concurrency

WHAT IS $GOPATH



PACKAGE IMPORTS



PACKAGES

main.go

```go
package main

import (
    "github.com/steevehook/mdc19/examples/packages/pkg1"
    // bad practice, avoid naming the package differently than the dir
    pkg_another "github.com/steevehook/mdc19/examples/packages/pkg2"
)

func main() {
    pkg1.F1()
    pkg_another.F2()
}
```

pkg1/pkg1.go

```go
package pkg1

import "fmt"

func F1() {
    fmt.Println(a...:"f1")
}
```

pkg1/pkg_another.go

```go
package pkg_another

import "fmt"

func F2() {
    fmt.Println(a...:"f2")
}
```

PACKAGES BEST PRACTICES

- Name them **concise** and **meaningful**
- Avoid naming like **utils** and **helpers**
- **Package** name = **directory** name
- Forget about **naming collision**
- No **cross reference** allowed
- Create a **file** exactly as **package name** for **common** things
- Create a **dedicated package** for **test utils**

PACKAGE MANAGEMENT

- Go mod
- Dep
- Glide
- $GOPATH

GO MOD

```
go mod help
```

```
go mod init github.com/steevehook/mdc19
```

```
go get -u package@v1.0.0
```

```
go get ./...
```

```
go mod tidy
```

```
go mod vendor
```

Does **not** have to be inside **$GOPATH**

Migrates well from **dep** or **glide**

GO MOD EXAMPLE

main.go

```go
package main

import "github.com/sirupsen/logrus"

func main() {
    logger := logrus.WithField( key: "some_key",  value: "some_value")
    logger.Println( args...: "printing some args")
}
```

go.mod

```
module github.com/steevehook/mdc19

go 1.12

require github.com/sirupsen/logrus v1.4.2
```

go.sum

```
github.com/davecgh/go-spew v1.1.1/go.mod h1:J7Y8YcW2NihsgmVo/mv3lAwl/skON4iLHjSsI+c5H38=
github.com/konsorten/go-windows-terminal-sequences v1.0.1 h1:mweAR1A6xJ3oS2pRaGiHgQ4OO8tzTaLawm8vnODuwDk=
github.com/konsorten/go-windows-terminal-sequences v1.0.1/go.mod h1:T0+1ngSBFLxvgU3pZ+m/2kptfBszLMUkC4ZK/EgS/cQ=
github.com/pmezard/go-difflib v1.0.0/go.mod h1:iKH77koFhYxTK1pcRnkKkqfTogsbg7gZNVY4sRDYZ/4=
github.com/sirupsen/logrus v1.4.2 h1:SPIRibHv4MatM3XXNO2BJeFLZwZ2LvZgfQ5+UNI2im4=
github.com/sirupsen/logrus v1.4.2/go.mod h1:tLMulIdttU9McNUspp0xgXVQah82FyeX6MwdIuYE2rE=
github.com/stretchr/objx v0.1.1/go.mod h1:HFkY916IF+rwdDfMAkV7OtwuqBVzrE8GR6GFx+wExME=
github.com/stretchr/testify v1.2.2/go.mod h1:a8OnRcib4nhh0OaRAV+Yts87kKdq0PP7pXfy6kDkUVs=
golang.org/x/sys v0.0.0-20190422165155-953cdadca894 h1:Cz4ceDQGXuKRnVBDTS23GTn/pU5OE2C0WrNTOYK1Uuc=
golang.org/x/sys v0.0.0-20190422165155-953cdadca894/go.mod h1:h1NjWce9XRLGQEsW7wpKNCjG9DtNlClVuFLEZdDNbEs=
```

BASIC TYPES

bool	uint	float32
int	uint8	float64
int8	uint16	complex64
int16	uint32	complex128
int32/rune	uint64	string
int64	uintptr	

CONST/VAR DECLARATIONS

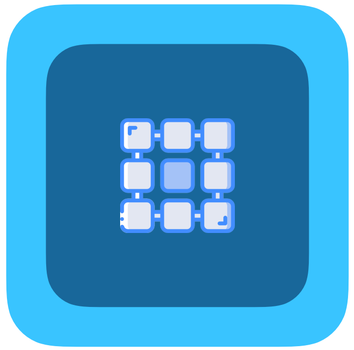
constants.go

```go
package main

import "fmt"

const c1 = 10
const s string = "Hello"

const (
    g1       = 10
    g2 int8 = 32
)

func main() {
    fmt.Println(c1, s, g1, g2)
}
```

variables.go

```go
package main

import "fmt"

var s1 string = "s1"
var s2 = "s2"
var s3 string

// group
var (
    g1 = 1
    g2 = 2
)

func main() {
    s3 = "s3"
    s4 := "s4"
    fmt.Printf( format: "%s\n%s\n%s\n%s\n%d\n%d", s1, s2, s3, s4, g1, g2)
}
```

AGGREGATE TYPES



Arrays/Slices

Maps

Structs

Channels

ARRAYS

arrays.go

```go
package main

import "fmt"

func main() {
    arrInt, arrString := [3]int{1, 2, 3}, [2]string{"Hello", "World"}
    //arrIntErr := [2]int{10, 20, 30} // index out of bounds: 2
    arrVar := [...]string{
        "Some", "very", "long", "string", "array", "Heck", "knows", "the", "end", "of", "it",
    }
    arrVarCap := cap(arrVar)
    fmt.Printf( format: "%d\n%s\n%s\n", arrInt, arrString, arrVar)
    fmt.Printf( format: "The type of arrVar: %T", arrVar) // [11]string
    fmt.Printf( format: "The capacity of arrVar: %d", arrVarCap)
}
```

SLICES

slices.go

```go
package main

import "fmt"

var sComplex []complex128

func main() {
    sInt, sString := []int{1,2,3}, []string{"Hello", "World"}
    s1, s2, s3 := make([]int, 100), make([]int, 0, 100), make([]int, 10, 100)
    sComplex = append(sComplex, elems...: 2 + 3i, 4+2i, 5, 6)
    newIntSlice := append([]int{10, 20}, sInt...)
    fmt.Printf( format: "%d\n%s\n%v\n%d\n", sInt, sString, sComplex, newIntSlice)
    fmt.Printf( format: "s1|len:%d<=>cap:%d\ns2|len:%d<=>cap:%d\ns3|len:%d<=>cap:%d\n",
        len(s1), cap(s1), len(s2), cap(s2), len(s3), cap(s3),
    )
}
```

LEN/CAP/APPEND/MAKE

```
len( array/slice )
cap( array/slice )
append( slice , elem , elem , ...)
make( slice/map/chan ,[ len , cap ])
```

ARRAY/SLICE ADDRESSING

arrays-slices-addressing.go

```go
package main


import "fmt"


func main() {
    s := make([]int, 10, 12)
    fmt.Println( a...: "s:", s) // [0 0 0 0 0 0 0 0 0 0]
    s[0], s[5] = 10, 15
    fmt.Printf( format: "s[0]:%d\ns[5]:%d\ns[9]:%d\n", s[0], s[5], s[9])
    fmt.Println( a...: "cap(s):", cap(s)) // 12
    s = append(s, elems...: 1, 2, 3)
    fmt.Println( a...: "cap(s):", cap(s)) // 24
    //fmt.Println(s[11]) // index out of range
}
```

ARRAY/SLICE LOOPING

array-slice-looping.go

```go
package main

import "fmt"

func main() {
    names := []string{"John", "Mike", "Jane"}
    for i := 0; i < len(names); i++ {
        fmt.Println(names[i])
    }
    for index, name := range names {
        fmt.Println(index, name)
    }
    for _, name := range names {
        fmt.Println(name)
    }
}
```

MAPS

maps.go

```go
package main

import "fmt"

var dictionary map[string]string // nil map (either init it later or avoid this)

func main() {
    //dictionary["some_key"] = "some_value" // panic: assignment to entry in nil map
    wiktionary := make(map[string]string)
    wiktionary["some_key"] = "some_value"
    grades := map[string]int{
        "Math":    10,
        "English": 10,
    }
    fmt.Printf( format: "dictionary: %+v\nwiktionary: %+v\ngrades: %+v\n", dictionary, wiktionary, grades)
}
```

MAPS LOOPING

maps-looping.go

```go
package main

import "fmt"

func main() {
    grades := map[string]int{
        "Math": 8,
        "English": 10,
        "Physics": 9,
    }
    // the order may always be different (because it's hashmap)
    for subject, grade := range grades {
        fmt.Printf( format: "%s ==> %d\n", subject, grade)
    }
}
```

MAPS COMMA, OK IDIOM

maps-comma-ok.go

```go
package main

import "fmt"

func main() {
    dictionary := map[string]string{"some_key": "some_value"}
    val, ok := dictionary["another_key"]
    fmt.Printf( format: "value: %v <=> ok: %v\n", val, ok)
    val, ok = dictionary["some_key"]
    fmt.Printf( format: "value: %v <=> ok: %v\n", val, ok)
}
```

NIL MAP

nil-map.go

```go
package main

import "fmt"

var robot map[string]func() // nil map

func init() {
    robot = map[string]func(){
        "start": func() {
            fmt.Println( a...: "robot started")
        },
    }
}

func main() {
    robot["start"]()
    // panic: runtime error: invalid memory address or nil pointer dereference
    robot["stop"]()
}
```

MAPS DELETE

maps-delete.go

```go
package main

import "fmt"

func main() {
    dictionary := map[string]string{
        "human": "being that has 1 life",
        "programmer": "being that has no life",
    }
    fmt.Printf(format: "%+v\n", dictionary)
    delete(dictionary, key: "programmer")
    delete(dictionary, key: "non-existent") // no error
    fmt.Printf(format: "%+v\n", dictionary)
}
```

ZERO VALUE

bool	false
int/uint	0
float	0
complex	0 + 0i
string	""
pointer	nil
error	nil
map	nil
chan	nil
func	nil
interface{}	nil
struct	empty struct with zero value fields

STRUCTS

structs.go

```go
package main

import "fmt"

type person struct {
    Name string
    Age  uint
}

func main() {
    people := []struct{
        name string
        age uint
    } {
        {name: "John", age: 21},
        { name: "Jane",  age: 19}, // don't use this
    }
    fmt.Printf( format: "%+v\n", person{Name: "Steve", Age: 24})
    fmt.Printf( format: "%+v", people)
}
```

STRUCTS – EXPORTED/UNEXPORTED FIELDS

person/person.go

```go
package person

// Person represents the person type
type Person struct {
    name  string
    age   uint
    Hobby string
}
func NewPerson(name string, age uint, hobby string) Person {
    return Person{name: name, age: age, Hobby: hobby}
}
func (p Person) Name() string {
    return p.name
}
func (p Person) Age() uint {
    return p.age
}
```

main.go

```go
package main

import (
    "fmt"
    "github.com/steevehook/mdc19/examples/struct-exported-unexported/person"
)

func main() {
    p1 := struct {
        name string
        age  uint
    }{name: "Steve", age: 24}
    p2 := person.NewPerson( name: "John",  age: 19,  hobby: "developer")
    p3 := person.Person{Hobby: "human"}
    fmt.Printf( format: "p1:\nname: %s\nage: %d\n\n", p1.name, p1.age)
    fmt.Printf( format: "p2:\nname: %s\nage: %d\nHobby: %s\n\n", p2.Name(), p2.Age(), p2.Hobby)
    fmt.Printf( format: "p3:\nname: %s\nage: %d\nHobby: %s\n", p3.Name(), p3.Age(), p3.Hobby)
}
```

STRUCTS – FIELD PROMOTION

struct-field-promotion.go

```go
package main

import "fmt"

type person struct {
    name string
    age  uint
}
type developer struct {
    person
    //age uint // this will override the previous field
    language string
}

func main() {
    d := developer{
        language: "Go",
        person:   person{name: "Steve", age: 24},
    }
    fmt.Printf( format: "developer info\n---\nname: %s\nage: %d\nlanguage: %s", d.name, d.person.age, d.language)
}
```

STRUCTS - JSON

structs-json.go

```go
package main

import (
    "encoding/json"
    "fmt"
)

type person struct {
    Name string `json:"name"`
    Age  uint   `json:"age,omitempty"`
    //sex   bool `json:"sex"` // not gonna work
    sex   bool
    Hobby string `json:"-"`
}

func main() {
    // always handle errors :D
    bs1, _ := json.Marshal(person{Name: "Steve", Age: 24, sex: true, Hobby: "developer"})
    bs2, _ := json.Marshal(person{Name: "Jane", sex: false, Hobby: "human"})
    fmt.Println(string(bs1)) // {"name":"Steve","age":24}
    fmt.Println(string(bs2)) // {"name":"Jane"}
}
```

ARRAYS/SLICES – JSON

arrays-slices–json.go

```go
package main

import (
    "encoding/json"
    "fmt"
)

func main() {
    grades := []int{10, 20, 30, 40}
    people := []struct {
        Name string `json:"name"`
    }{{Name: "Steve"}, {Name: "John"}, {Name: "Jane"}, {Name: "Mary"}}
    // handle errors you bustard
    gradesBS, _ := json.Marshal(grades)
    peopleBS, _ := json.Marshal(people)
    //[10,20,30,40]
    //[{"name":"Steve"},{"name":"John"},{"name":"Jane"},{"name":"Mary"}]
    fmt.Printf(format: "grades:\n%s\n\npeople:\n%s", string(gradesBS), string(peopleBS))
}
```

MAPS - JSON

maps–json.go

```go
package main

import (
    "encoding/json"
    "fmt"
)

type person struct {
    Name string `json:"name"`
    Age  uint   `json:"age,omitempty"`
}

func main() {
    people := map[string]person{
        "fa1b4f6f-beab-5e55-acdf-6da3a03aabec": {
            Name: "Steve", Age: 24,
        },
        "fa1b4f6f-beab-5e55-acdf-6da3a03aabed": {
            Name: "John",
        },
    }
    // handle errors, PLEASE :D
    bs, _ := json.Marshal(people)
    // {
    //  "fa1b4f6f-beab-5e55-acdf-6da3a03aabec":{"name":"Steve","age":24},
    //  "fa1b4f6f-beab-5e55-acdf-6da3a03aabed":{"name":"John"}
    // }
    fmt.Println(string(bs))
}
```

JSON UNMARSHALLING

json-unmarshal.go

```go
package main

import (
    "encoding/json"
    "fmt"
)

type person struct {
    Name string `json:"name"`
    Age  uint   `json:"age,omitempty"`
    //sex   bool `json:"sex"` // not gonna work
    sex   bool
    Hobby string `json:"-"`
}

func main() {
    j1 := `{"name": "Mary", "age": 21, "sex": true, "hobby": "human"}`
    j2 := `{"name": "Steve", "age": 24, "anything_else": "not_gonna_unmarshal"}`
    var p1 person
    _ = json.Unmarshal([]byte(j1), &p1)
    var p2 person
    _ = json.Unmarshal([]byte(j2), &p2)
    fmt.Printf( format: "p1:\n%+v\n\np2:\n%+v", p1, p2)
}
```

FUNCTIONS

main.go

```go
package main

import (
    "fmt"
    "log"
)

func main() {
    nums := []int{10, 15, -2, 4}
    for _, n := range nums {
        i, err := identity(n)
        if err != nil {
            log.Fatal(err)
        }
        fmt.Println(i)
    }
}
```

identity.go

```go
package main

import "errors"

func identity(num int) (int, error) {
    if num < 0 {
        return 0, errors.New( text: "number is lower than 0")
    }
    return num, nil
}
```

CUSTOM TYPES

custom-types.go

```go
package main

import "fmt"

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

func main() {
    var p person
    p1 := struct {
        name string
        age  uint
    }{name: "Jane", age: 19}
    p = p1
    var a1 int
    var a2 age
    a1 = int(a2)
    //a1 = a2 // won't work
    fmt.Printf( format: "person: %+v\nage: %d\n\n", person{name: "Steve", age: 24}, age(1))
    fmt.Printf( format: "p: %+v\na1:%d", p, a1)
}
```

METHODS / FUNCTION RECEIVERS

type-methods.go

```go
package main

import "fmt"

type age uint

func (a *age) BirthDay() *age {
    *a += 1
    return a
}
func (a age) IsDead() bool {
    return a > 150
}

func main() {
    a := age(148)
    a.BirthDay().BirthDay()
    fmt.Printf( format: "age: %d\nis dead: %t\n\n", a, a.IsDead())
    a.BirthDay()
    fmt.Printf( format: "age: %d\nis dead: %t", a, a.IsDead())
}
```

POINTERS

main.go

```go
package main

import "fmt"

func main() {
    var a, b *int
    iPtr := intPtr( i: 10)
    a = iPtr
    b = iPtr
    *a = 10
    fmt.Printf( format: "a: %d\nb: %d\niPtr: %d", *a, *b, *iPtr)
}
```

int-pointer.go

```go
package main

func intPtr(i int) *int {
    var n int
    n = i
    return &n
}
```

POINTER VS VALUE

pointer-vs-value.go

```go
package main

import "fmt"

type person struct {
    name string
}
func (p person) changeNameValue(name string) {
    p.name = name
}
func (p *person) changeNamePointer(name string) {
    p.name = name
}

func main() {
    p1, p2 := person{name: "Steve"}, &person{name: "George"}
    p1.changeNameValue( name: "John")
    fmt.Println(p1.name)
    p1.changeNamePointer( name: "John")
    fmt.Println(p1.name)
    p2.changeNameValue( name: "Unknown")
    fmt.Println(p2.name)
}
```

INTERFACES

types.go

```go
package main

import "fmt"

type person struct {
    name string
}
func (p person) Run() {
    fmt.Println(p.name, "is running")
}

type runner interface {
    Run()
}
```

main.go

```go
package main

import "fmt"

func marathon(runners ...runner) {
    for _, r := range runners {
        r.Run()
    }
}

func main() {
    steve := person{name: "Steve"}
    john := struct {
        name string
    }{
        name: "John",
    }
    marathon(steve)
    //marathon(steve, john) // incompatible types
    fmt.Println(john.name, "is running")
}
```

INTERFACE{} TYPE

- Represents **any type** in Go
- Any type **implements** it
- Does not equal to **nil**
- Supports **type assertion**
- Has **type** and **value** fields
- The **value** has to be a **concrete type**

INTERFACE{} EXAMPLE

interface{}.go

```go
package main

import "fmt"

func main() {
    anything := map[string]interface{}{
        "animals": []string{"lion", "bear"},
        "grades": []int{10, 8, 9},
        "people": []struct {
            name string
        }{
            {name: "Steve"},
            {name: "John"},
        },
    }
    fmt.Printf( format: "Anything: %+v", anything)
}
```

THE POWER OF INTERFACES

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

type Writer {
    Write(p []byte) (n int, err error)
}
```

IMPLEMENT ERROR INTERFACE

main.go

```go
package main

import "log"

func handle(err error) {
    log.Fatal(err)
}

func main() {
    handle(CustomErr{Message: "some error"})
}
```

custom-error.go

```go
package main

type CustomErr struct {
    Message string
}

func (e CustomErr) Error() string {
    return e.Message
}
```

TYPE ASSERTION

main.go

```go
package main

import (
    "errors"
    "log"
    "net/http"
)

func main() {
    http.HandleFunc( pattern: "/http", func(w http.ResponseWriter, req *http.Request) {
        SendError(w, http.StatusInternalServerError, HTTPError{Message: "something bad happen"})
    })
    http.HandleFunc( pattern: "/data", func(w http.ResponseWriter, req *http.Request) {
        SendError(w, http.StatusBadRequest, DataValidationError{
            Message: "something bad happen",
            Constraints: map[string]interface{}{
                "field1": "some error on field1",
            },
        })
    })
    http.HandleFunc( pattern: "/error", func(w http.ResponseWriter, req *http.Request) {
        SendError(w, http.StatusInternalServerError, errors.New( text: "test"))
    })
    log.Fatal(http.ListenAndServe( addr: ":8080",  handler: nil))
}
```

encoder.go

```go
package main

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

func toJSON(w http.ResponseWriter, code int, data interface{}) {
    w.WriteHeader(code)
    // handle errors, bustard
    _ = json.NewEncoder(w).Encode(data)
}

func SendError(w http.ResponseWriter, code int, e error) {
    switch err := e.(type) {
    case DataValidationError:
        err.Type = DataValidationType
        toJSON(w, code, err)
    case HTTPError:
        err.Type = HTTPErrorType
        toJSON(w, code, err)
    default:
        toJSON(w, code, HTTPError{
            Type:    HTTPErrorType,
            Message: "some http error",
        })
    }
}
```

errors.go

```go
package main

const (
    HTTPErrorType      = "http_error"
    DataValidationType = "data_validation"
)

type DataValidationError struct {
    Type        string                 `json:"type"`
    Message     string                 `json:"message"`
    Constraints map[string]interface{} `json:"constraints"`
}

func (e DataValidationError) Error() string {
    return e.Message
}

type HTTPError struct {
    Type    string `json:"type"`
    Message string `json:"message"`
}

func (e HTTPError) Error() string {
    return e.Message
}
```

ERRORS

- **Errors** are just **values**
- There are **no exceptions** and no catching
- The **error** usually comes as the **2nd return**
- **error** is just another **interface** (Error() string)
- Error messages don’t start with **Uppercase** or have **weird spacing**

ERRORS EXAMPLE

main.go

```go
package main

import (
    "fmt"
    "log"
)

func main() {
    nums := []int{10, -15}
    for _, n := range nums {
        val, err := identity(n)
        if err != nil {
            log.Fatal(err)
            return
        }
        fmt.Println( a...: "identity:", val)
    }
}
```

identity.go

```go
package main

import "errors"

func identity(n int) (int, error) {
    if n < 0 {
        return 0, errors.New( text: "the number is less than zero")
    }
    return n, nil
}
```

DEFER

defer.go

```go
package main

import "fmt"

type operation struct{}

func (op operation) SideEffects() {
    fmt.Println( a...: "doing some side effects")
}
func (op operation) CleanUp() {
    fmt.Println( a...: "cleaning up after the side effects")
}

func main() {
    o := operation{}
    o.SideEffects()
    defer o.CleanUp()
    fmt.Println( a...: "other operations")
    panic( v: "some serious error")
}
```

defer-loop.go

```go
package main

import "fmt"

func clean(i int) {
    fmt.Printf( format: "cleanup for operation: %d\n", i)
}

func main() {
    for i := 0; i < 10; i++ {
        defer clean(i)
    }
}
```

PANIC & RECOVER

panic-recover.go

```go
package main

import "fmt"

func yikes() {
    panic( v: "can't do anything")
}

func main() {
    defer func() {
        err := recover()
        fmt.Println(err)
    }()
    yikes()
}
```

- Don’t **panic**! Avoid panicking
- You can **recover** from any **panic**
- A **panic** shuts down the entire app with an **exit code**
- Have a **panic recovery middleware**, for graceful handling
- You can’t **recover** on the same **function level**
- **panic** can contain **any value**
- Can’t **recover** without **defer**

CHANNELS

main.go

```go
package main

import "time"

type message struct {
    id   string
    text string
}

func main() {
    messages := make(chan message)
    go listen(messages)
    messages <- message{id: "1", text: "first message"}
    messages <- message{id: "2", text: "second message"}
    time.Sleep(1 * time.Second)
}
```

listen.go

```go
package main

import "fmt"

func listen(messages chan message) {
    for {
        select {
        case msg := <-messages:
            fmt.Printf( format: "%+v", msg)
        default:
            return
        }
    }
}
```

CHANNELS FACTS

- They **block** when the **buffer** is **full**
- Can cause **deadlocks** if no routine reads it
- Should be **closed** at the **end**
- Can’t be **written to** if **closed**
- If **main go routine** does not **block** it **exits**
- Ranging over **unclosed chan** causes **deadlock**

BUFFERED CHANNELS

buffered-channels.go

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    buffered := make(chan int, 2)
    buffered <- 1 // will not block - buffer = 1
    buffered <- 2 // will not block - buffer = 2
    //buffered <- 3 // WILL BLOCK - buffer > 2
    go func() {
        for i := range buffered {
            fmt.Println(i)
        }
    }()
    buffered <- 3
    time.Sleep(1 * time.Second)
}
```

DEADLOCK

deadlock.go

```go
package main


func main() {
    ch := make(chan int)
    ch <- 10
}
```

deadlock-fix.go

```go
package main

import "fmt"

func main() {
    ch := make(chan int)
    go func() {
        fmt.Println(<-ch)
    }()
    ch <- 10
}
```

"**Concurrency** is not **parallelism**"

CONCURRENCY (GO ROUTINES)

concurrency.go

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    go func() {
        fmt.Println( a...: "routine 1")
    }()
    go func() {
        fmt.Println( a...: "routine 2")
    }()
    go func() {
        fmt.Println( a...: "routine 3")
    }()
    time.Sleep(1 * time.Second)
}
```

- Start a go routing using **go** in front of **func call**
- They are **not threads**
- They **don't start** in the **order** they **appear**
- They can cause **data races**
- **Go Scheduler** creates **1 thread** per **CPU core**
- **main** also runs in a **go routine** (Main go routine)
- If **main exits** it **terminates** all **go routines**

RACE CONDITIONS

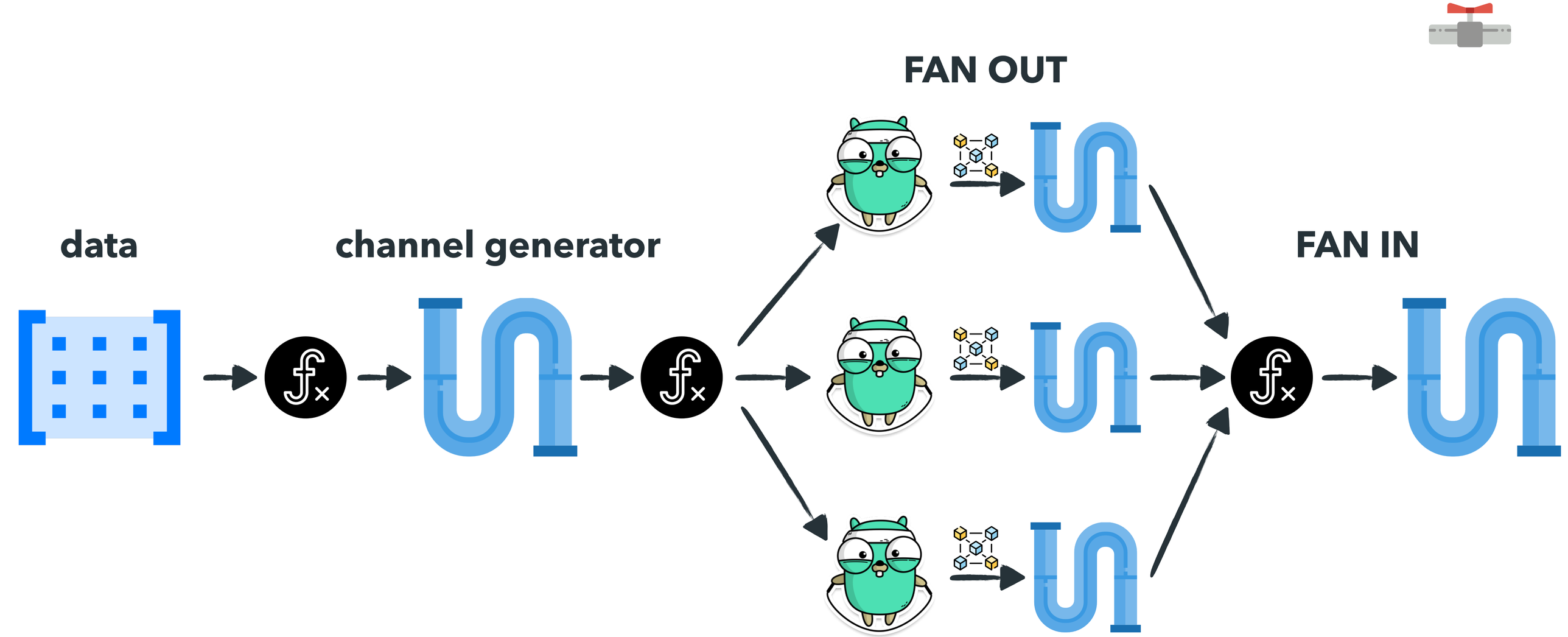
race-conditions.go

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    n := 10
    go func() {
        n = 15
    }()
    go func() {
        n = 12
    }()
    go func() {
        fmt.Println(n)
    }()
    time.Sleep(1 * time.Second)
}
```

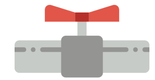
```
go run -race *.go
```

```
go test -race ./...
```

CHANNELS FAN OUT / FAN IN PATTERN



CHANNELS FAN IN/FAN OUT



main.go

```go
package main

import (
    "fmt"
    "time"
)

func generate(numbers ...int) chan int {
    out := make(chan int)
    go func() {
        for _, num := range numbers {
            out <- num
        }
        close(out)
    }()
    return out
}

func main() {
    numbers := generate( numbers...: 3, 6, 9)
    c1 := fanOut(numbers)
    c2 := fanOut(numbers)
    result := fanIn(c1, c2)
    for i := range result {
        fmt.Println(i)
    }
    time.Sleep(1 * time.Second)
}
```

fan-out.go

```go
package main

func fanOut(in chan int) chan int {
    out := make(chan int)
    go func() {
        for i := range in {
            out <- square(i)
        }
        close(out)
    }()
    return out
}

func square(num int) int {
    return num * num
}
```

fan-in.go

```go
package main

func fanIn(channels ...chan int) chan int {
    out := make(chan int)
    done := make(chan struct{})
    for _, c := range channels {
        go func(in chan int) {
            for num := range in {
                out <- num
            }
            done <- struct{}{}
        }(c)
    }
    go func() {
        for i := 0; i < len(channels); i++ {
            <-done
        }
        close(out)
    }()
    return out
}
```

HTTP

main.go

```go
package main

import (
    "log"
    "net/http"
)

func main() {
    mux := http.NewServeMux()
    mux.HandleFunc( pattern: "/", home())
    mux.HandleFunc( pattern: "/about", about())
    log.Fatal(http.ListenAndServe( addr: ":8080", mux))
}
```

about.go

```go
func about() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        // handle errors, son of the bitch
        _, _ = fmt.Fprint(w, a...: "About")
    }
}
```

home.go

```go
func home() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        // handle errors, son of the bitch
        _, _ = fmt.Fprint(w, a...: "Welcome Home!")
    }
}
```

TCP (MEM CACHE)

main.go

```go
package main

import (
    "fmt"
    "log"
    "net"
)

func main() {
    cache := cache{}
    listener, err := net.Listen(
        network: "tcp",
        address: ":8080",
    )
    defer listener.Close()
    if err != nil {
        log.Fatal(err)
    }
    for {
        // handle errors, you bustard
        conn, _ := listener.Accept()
        fmt.Fprintf(
            conn,
            format: "Welcome to CACHE 1.0\n-> ",
        )
        go listen(conn, cache)
    }
}
```

listen.go

```go
package main

import (
    "bufio"
    "fmt"
    "net"
    "strings"
)

func listen(c net.Conn, mem cache) {
    scanner := bufio.NewScanner(c)
    for scanner.Scan() {
        l := strings.ToLower(strings.TrimSpace(scanner.Text()))
        values := strings.Split(l, sep: " ")
        switch {
        case len(values) == 3 && values[0] == "set":
            mem.Set(values[1], values[2])
            fmt.Fprintf(c, format: "OK\n-> ")
        case len(values) == 2 && values[0] == "get":
            fmt.Fprintf(c, format: "%s\n-> ",
                mem.Get(values[1]))
        case len(values) == 1 && values[0] == "exit":
            c.Close()
        default:
            fmt.Fprintf(c, format: "UNKNOWN: %s\n-> ", l)
        }
    }
}
```

cache.go

```go
package main

import "sync"

var mu sync.RWMutex

type cache map[string]string

func (c cache) Set(key, value string) {
    mu.Lock()
    defer mu.Unlock()
    c[key] = value
}
func (c cache) Get(key string) string {
    mu.RLock()
    defer mu.RUnlock()
    return c[key]
}
```

USEFUL GO COMMANDS

```
go run *.go
```

```
go build -o exec
```

```
go install
```

```
go get ./...
```

```
go test ./...
```

```
go fmt ./...
```

```
go vet ./...
```

HOW WE DEVELOP?

- We use a **Makefile**
- Write the **functionality**
- **Unit test** each component
- **Integration test** the whole service
- Write **swagger** docs

HOW WE TEST?

```
go test ./...
```

Use **testify** for test **suites**, **assertions** & **mocks**

controllers/comments.go

```go
type commentsService interface {
    GetComments() ([]string, error)
}

func GetComments(service commentsService) http.HandlerFunc {
    return func(w http.ResponseWriter, req *http.Request) {
        switch req.Method {
        case http.MethodGet:
            comments, err := service.GetComments()
            if err != nil {
                w.WriteHeader(http.StatusServiceUnavailable)
                return
            }
            err = json.NewEncoder(w).Encode(comments)
            if err != nil {
                w.WriteHeader(http.StatusBadRequest)
            }
        default:
            w.WriteHeader(http.StatusMethodNotAllowed)
        }
    }
}
```

main.go

```go
func main() {
    commentsService := services.NewComments()
    http.HandleFunc(
        pattern: "/comments",
        controllers.GetComments(commentsService),
    )
    log.Fatal(http.ListenAndServe(
        addr: ":8080",
        handler: nil,
    ))
}
```

services/comments.go

```go
type CommentsService struct{}

func NewComments() CommentsService {
    return CommentsService{}
}

func (c CommentsService) GetComments() ([]string, error) {
    return []string{
        "comment1",
        "comment2",
    }, nil
}
```

services/comments_test.go

```go
type commentsSuite struct {
    suite.Suite
    service CommentsService
}

func (s *commentsSuite) SetupSuite() {
    s.service = NewComments()
}

func (s *commentsSuite) Test_GetComments_Success() {
    expected := []string{
        "comment1",
        "comment2",
    }

    comments, err := s.service.GetComments()

    s.Require().NoError(err)
    s.Equal(expected, comments)
}

func TestComments(t *testing.T) {
    suite.Run(t, new(commentsSuite))
}
```

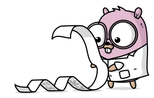
controllers/comments_test.go

```go
type commentsSuite struct {
    suite.Suite
    service *commentsServiceMock
}
func (s *commentsSuite) SetupSuite() {
    s.service = new(commentsServiceMock)
}
func (s *commentsSuite) TearDownSuite() {
    s.service.AssertExpectations(s.T())
}
func (s *commentsSuite) Test_GetComments_Success() {
    body := `["c1", "c2"]`
    req := httptest.NewRequest(
        http.MethodGet,
        target: "/comments",
        body: nil,
    )
    w := httptest.NewRecorder()
    handler := http.HandlerFunc(GetComments(s.service))
    s.service.
        On( methodName: "GetComments").
        Return([]string{"c1", "c2"}, nil).Once()
    handler.ServeHTTP(w, req)
    s.Equal( expected: 200, w.Code)
    s.JSONEq(body, w.Body.String())
}
func TestComments(t *testing.T) {
    suite.Run(t, new(commentsSuite))
}
```

controllers/mocks_test.go

```go
type commentsServiceMock struct {
    mock.Mock
}

func (m *commentsServiceMock) GetComments(
    ) ([]string, error) {
    args := m.Called()
    return args.Get( index: 0).([]string),
    args.Error( index: 1)
}
```

HOW WE DOCUMENT?



- We use **// comments**
- We document any **exported symbols**, except implementations
- We optionally document **un exported**(private) symbols

WHAT CODE STYLE WE USE?

```
go fmt ./...
```

```
go vet ./...
```

HOW WE DEPLOY?

- Compile the code & generate the **binary**
- Create a **daemon** for the binary
- Proxy it using i. e. **Nginx**
- We use **AWS**

WHICH ORM WE USE?

We Don’t

WHAT FRAMEWORKS WE USE?

We Don’t

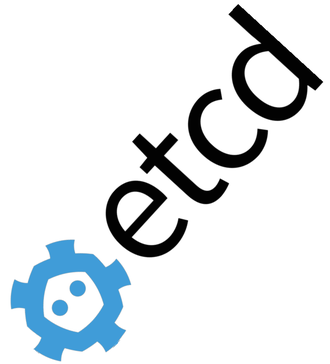
LIBRARIES?

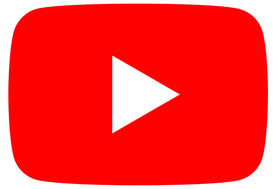
Yes, PLEASE!

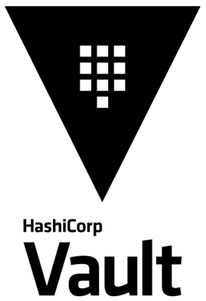
HOW WE HANDLE ERRORS?

```
if err != nil {
    // do something
}
```

APPS WRITTEN IN GO







THINGS I WISH GO HAD

- Generics
- Better error handling
- Ternary operator
- Filter,Map,Reduce
- Struct spread
- Go in gaming
- Go in UI
- Go in AI

RESOURCES

A tour of Go

GopherTuts

steevehook
/mdc19

"We don't stop **playing** because we get old, We get old because we stop playing"

George Bernard Shaw

DEMO