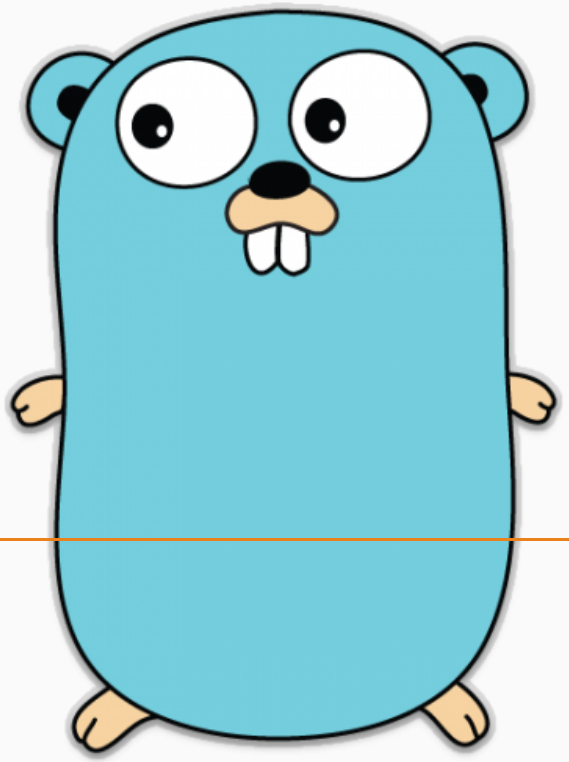
Golang

Anas El Halouani



Introduction

Before

In 2009, most software were written in C++ or Java.

It was clear that multiprocessors were becoming universal but most languages offered little help to program them efficiently and safely.

Efficient compilation, efficient execution, or ease of programming ?

Go is born!

Attempt to combine the ease of programming of an interpreted, dynamically typed language with the efficiency and safety of a statically typed, compiled language.

It also aimed to be modern, with support for networked and multicore computing.

It was born in 2009 and was made by **Google** and many contributors from the open source community.

How does it work?

Go Design

Garbage collected

Compiled

No virtual Machine

Object-oriented (yes and no)

Go Design

No generic types

No exceptions

No assertions

Go syntax

Variables

```
var msg string = "Hello"
msg2 := "Hello"
var a, b int
// a and b are 0
```

Array

```
fixed_size := [5]int{5, 4, 3, 2, 1}
sliced := []int{5, 4, 3, 2, 1}
sliced = append(sliced, 13)
```

If statement

```
if day == "sunday" || day == "saturday" {
    println("rest")
} else if day == "monday" {
    println("Work and groan")
} else {
    println("Work")
}
```

For loop

```go
for i := 0; i < 5; i++ {
    // Do things
}

i := 0
for i < 5 {
    // Do things
    i++;
}

arr := []{"a", "b", "c"}
for index, value := range arr {
    // Do things
}
```

Maps

```
m := make(map[string]int)
m["one"] = 1
m["two"] = 2

for key, value := range m {
    println("key:", key, " value:", value)
}
```

Struct

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

p := person{name: "Anas", age: 22}
println(p)
// Output: {Anas, 22}
```

Many Others...

Interface

Function

Duck Typing

Errors

Pointers

Channel

...

...And goroutine !

One of main feature of **Golang**

A **goroutine** is a lightweight thread managed by the Go runtime.

```
f(x, y, z) // In the main thread

go f(x, y, z) // New goroutine running
```

Channel

Channels are a typed conduit through which you can send and receive values with the channel operator, <-.

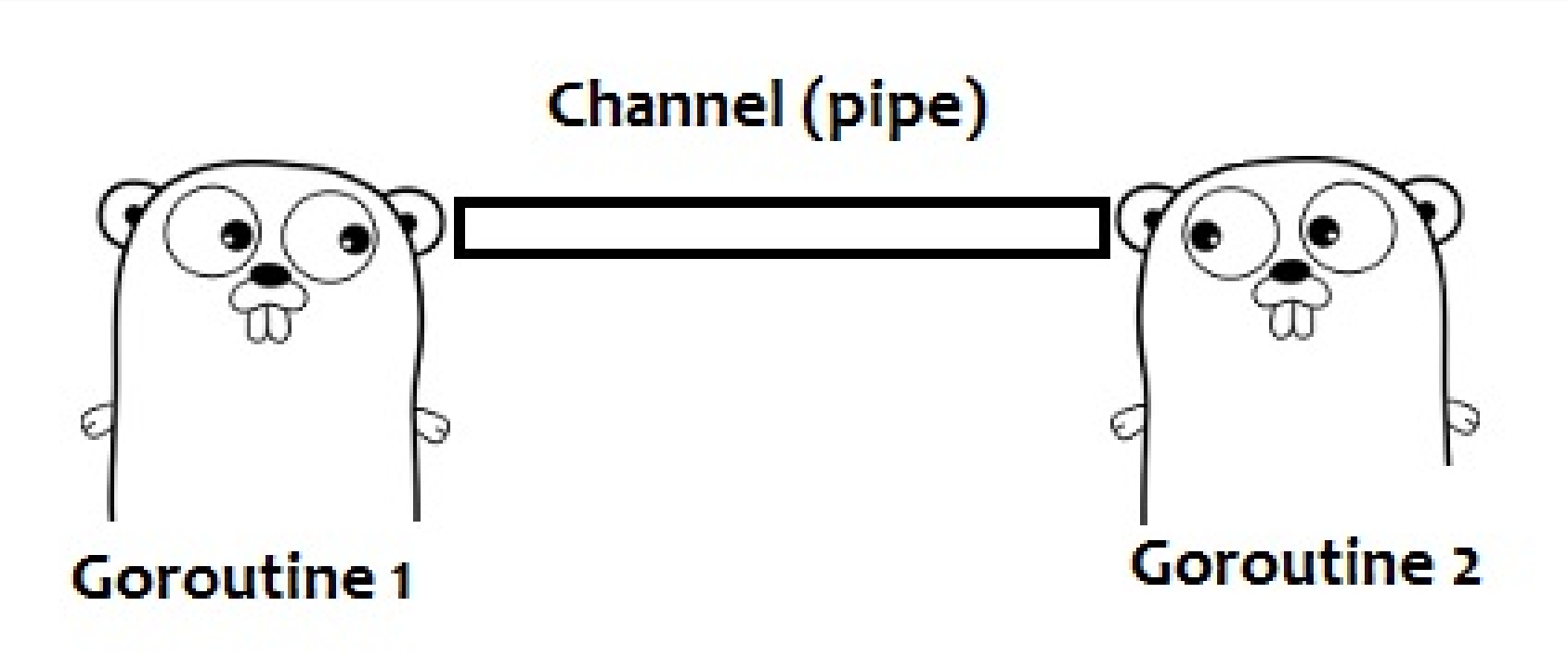


Figure 1: Go Channel

Channel

```
func sum(s []int, c chan int) {
    res := 0
    ... // Compute the sum
    c <- res
}
func main() {
    s := []int{7, 2, 8, -9, 4, 0}
    c := make(chan int)
    go sum(s[:len(s)/2], c)
    go sum(s[len(s)/2:], c)
    x := <-c // receive from c
    y := <-c // receive from c
    println(x, y, x+y) // Output: -5 17 12
}
```

Deadlock

```
func main() {
    ch := make(chan int)
    ch <- 1 // I send ressource
    fmt.Println(<-ch) // I wait ressource
}
```

Buffered

```
func main() {
    ch := make(chan int, 2)
    ch <- 1
    ch <- 2
    fmt.Println(<-ch)
    fmt.Println(<-ch)
}
```

Let’s GO

Demo time

Conclusion

Question ?