Introduction to go

Basics, tools, resources and patterns

Girish Ramnani

Golang

- Created at google in 2009
- Currently at version 1.8
- Started as a systems language
- Like C but simpler
- Produces compiled code
- Statically typed, garbage collected language

Companies using go

intel

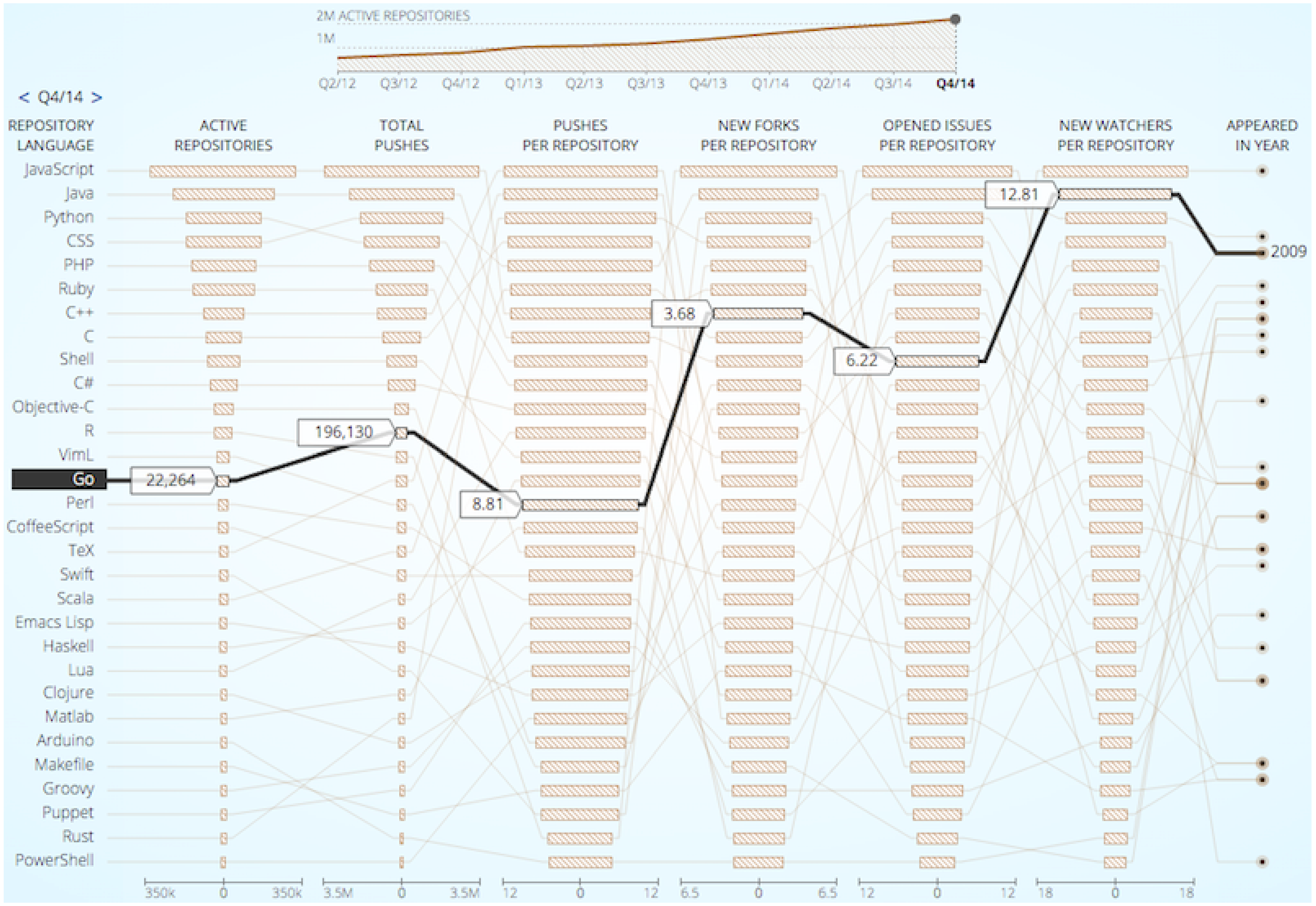
continued ..

influxdata

Famous OSS in go

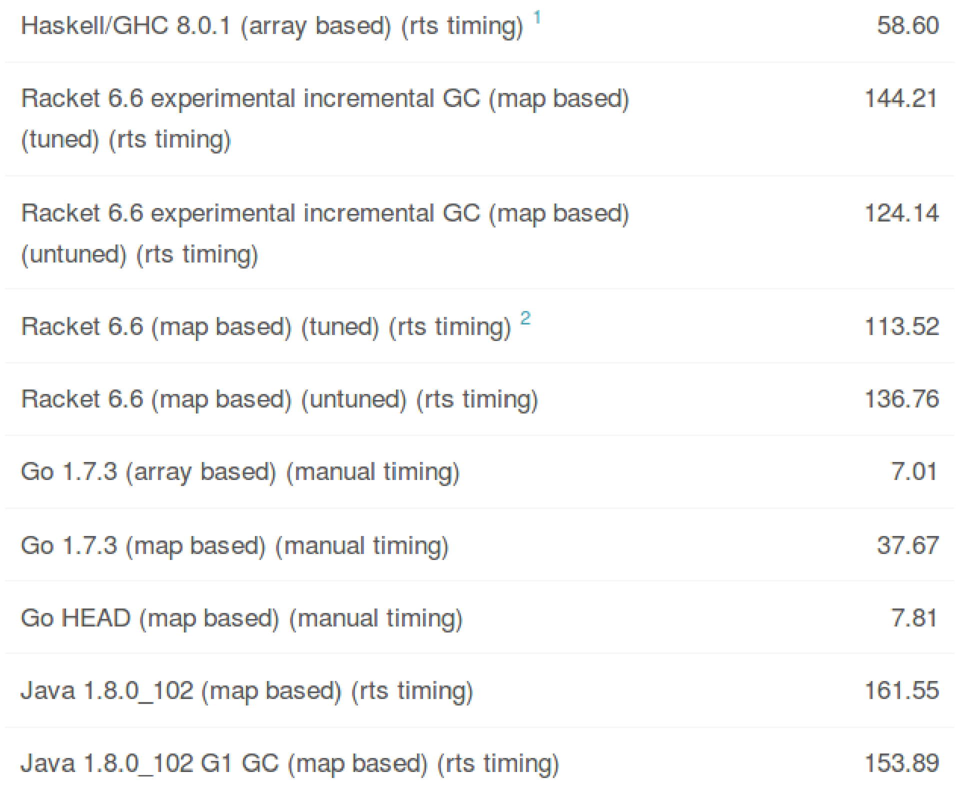
- Kubernetes (https://github.com/kubernetes/kubernetes)
- Weave (https://github.com/weaveworks/weave)
- Influxdb (https://github.com/influxdata/influxdb)
- Consul (https://github.com/hashicorp/consul)
- Pachyderm (https://github.com/pachyderm/pachyderm)
- Minio (https://github.com/minio/minio)
- Docker (https://github.com/docker/docker)

Stats for Golang



Whats makes it so good

+ Compiles to native Code (well so does C, C++)
+ Concurrent Garbage collector
+ Small language (27 Keywords)
+ Really easy to learn



Haskell/GHC 8.0.1 (array based) (rts timing) [1]	58.60
Racket 6.6 experimental incremental GC (map based) (tuned) (rts timing)	144.21
Racket 6.6 experimental incremental GC (map based) (untuned) (rts timing)	124.14
Racket 6.6 (map based) (tuned) (rts timing) [2]	113.52
Racket 6.6 (map based) (untuned) (rts timing)	136.76
Go 1.7.3 (array based) (manual timing)	7.01
Go 1.7.3 (map based) (manual timing)	37.67
Go HEAD (map based) (manual timing)	7.81
Java 1.8.0_102 (map based) (rts timing)	161.55
Java 1.8.0_102 G1 GC (map based) (rts timing)	153.89

Places to learn

Exercism (http://exercism.io/)

Awesome Go (https://github.com/avelino/awesome-go#websites)

Gopher Academy (https://blog.gopheracademy.com/)

Dave Cheney's Blog (https://dave.cheney.net)

William Kennedy's Blog (https://www.goinggo.net/)

Tools

goimports (https://godoc.org/golang.org/x/tools/cmd/goimports)

gofmt (https://golang.org/cmd/gofmt/)

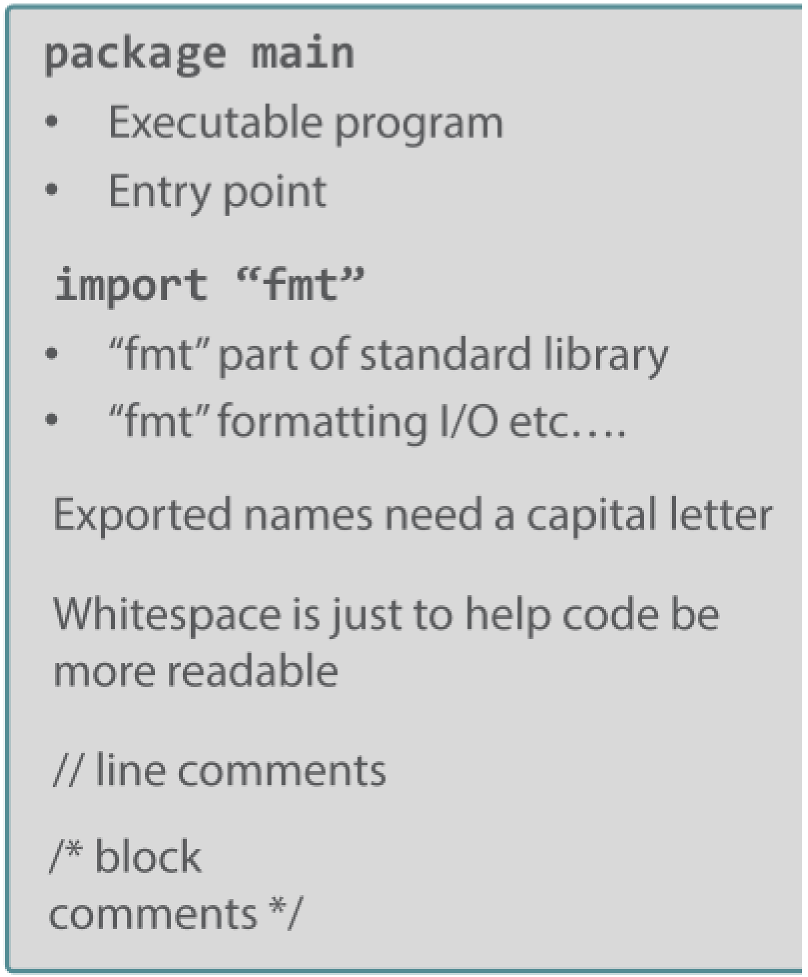
golint (https://github.com/golang/lint)

```
go tool pprof

go tool trace
```

Lets begin

Lets move rookie stuff out of the way



`package main`

- Executable program
- Entry point

`import “fmt”`

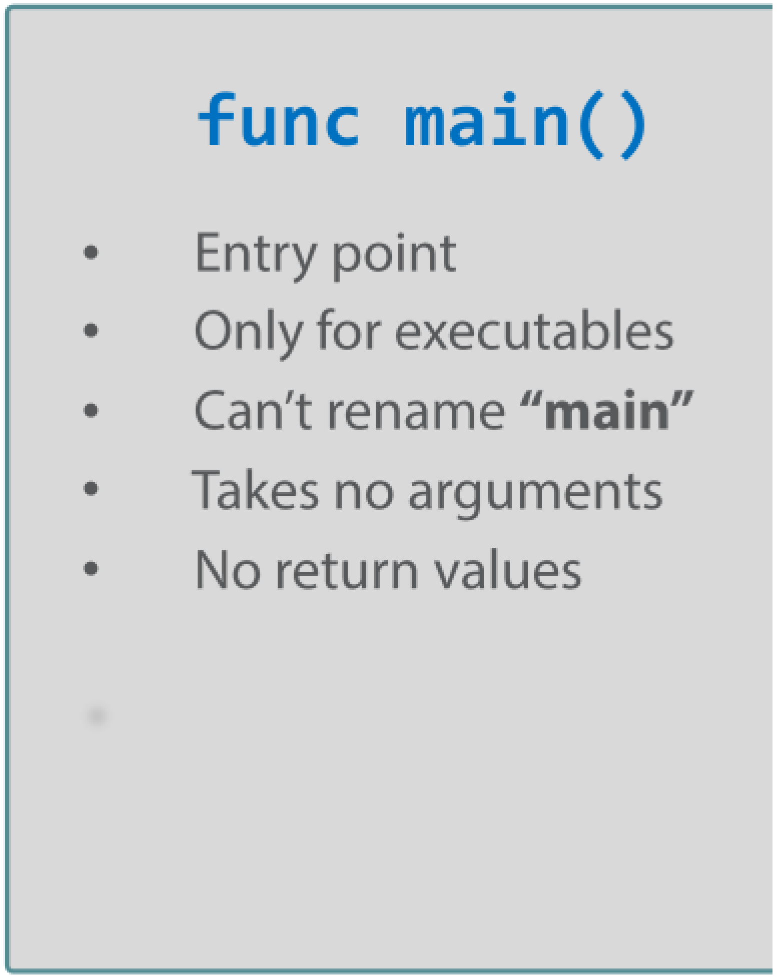
- “fmt” part of standard library
- “fmt” formatting I/O etc….

Exported names need a capital letter

Whitespace is just to help code be more readable

// line comments

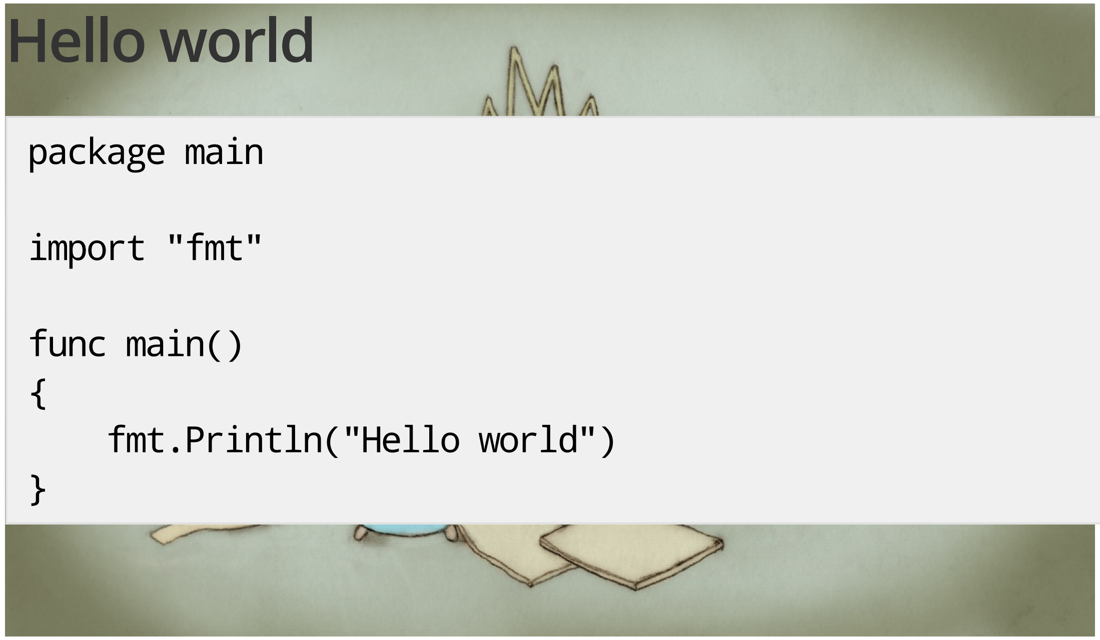
/* block
comments */



`func main()`

- Entry point
- Only for executables
- Can’t rename **“main”**
- Takes no arguments
- No return values

Hello world



```
package main

import "fmt"

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

Run

Variables and types

```go
package main

import (
    "fmt"
    "reflect"
)



func main() {
    var name string = "Hello"
    second := "World"

    var Unum uint8 = 12
    num := 65
    decinum := 56.433
    fmt.Println(name,second,Unum,decinum)
    fmt.Println(reflect.TypeOf(num))
}
```

Run

Functions

Functions are first class citizens.

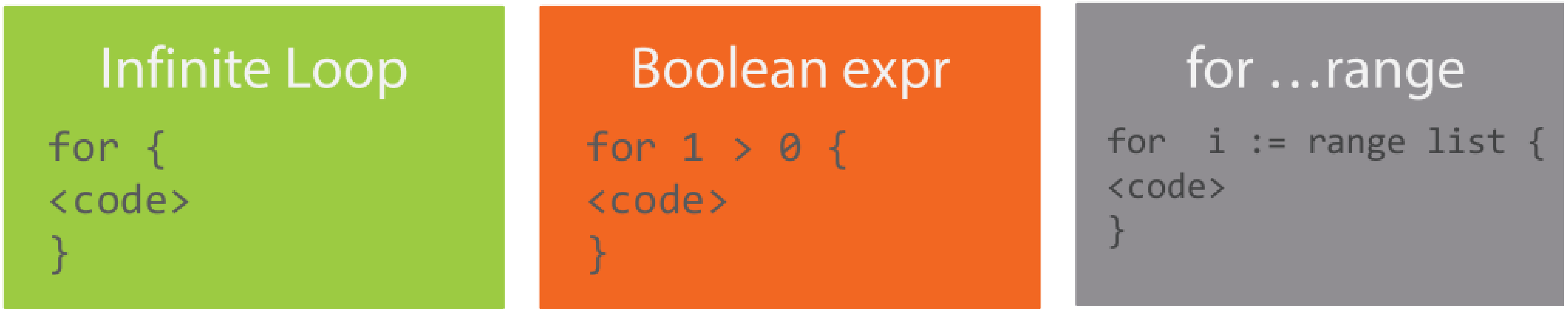
```go
package main

import (
    "fmt"
)

func Map(function func(input int) int , inputs []int  ) []int {
    var output []int
    for _,val  := range inputs {
        output = append(output,function(val))
    }
    return output
}
func main(){
    square := func(i int) int {
        return i*i
    }
    fmt.Println(Map(square,[]int {
        1,2,3,4,5,6,
    }))
}
```

Run

Loops



Infinite Loop

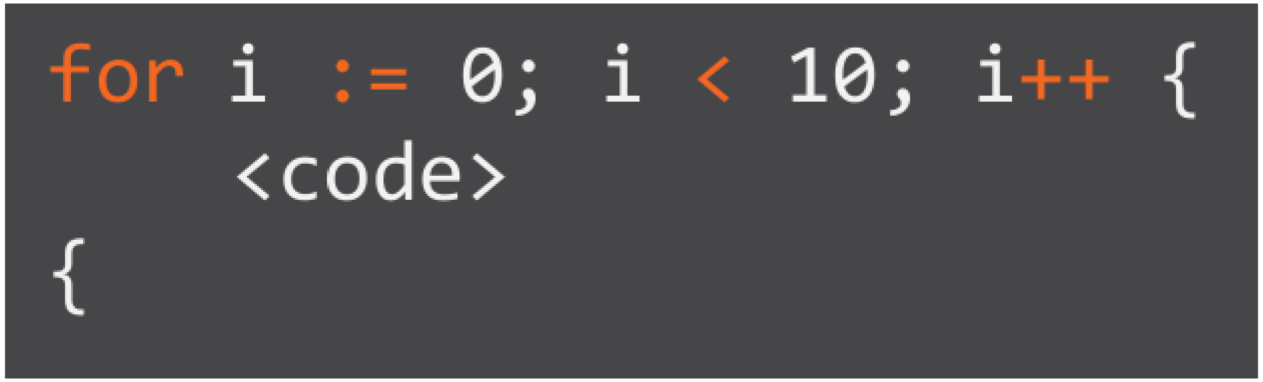
```
for {
<code>
}
```

Boolean expr

```
for 1 > 0 {
<code>
}
```

for …range

```
for  i := range list {
<code>
}
```



```
for i := 0; i < 10; i++ {
    <code>
{
```

Arrays and slices

- Arrays have fixed length
- Slices have flexible length
- Slices are passed by reference, arrays are not

Example

```
package main

import (
    "fmt"
)



func main() {

    var myName [2]string
    myName[0] = "Girish"
    myName[1] = "Ramnani"

    otherName  :=  [2]string {"Girish","Ramnani"}

    // slices with zeros

    mySlice := make([]int,5)
    otherSlice := []int{0,0,0,0,0}

    fmt.Println(myName,otherName,mySlice,otherSlice)
}
```

Run

Maps

- Unordered
- passed by reference
- you can use "for range" to interate over the map

```
for key,value := range map {

}
```

- example

```
var someMap map[string]float64
someMap = make(map[string]float64)

// shortcut
someMap := make(map[string]float64)

// delete a key
someMap = delete(someMap,<Key>)
```

Lets take a breather

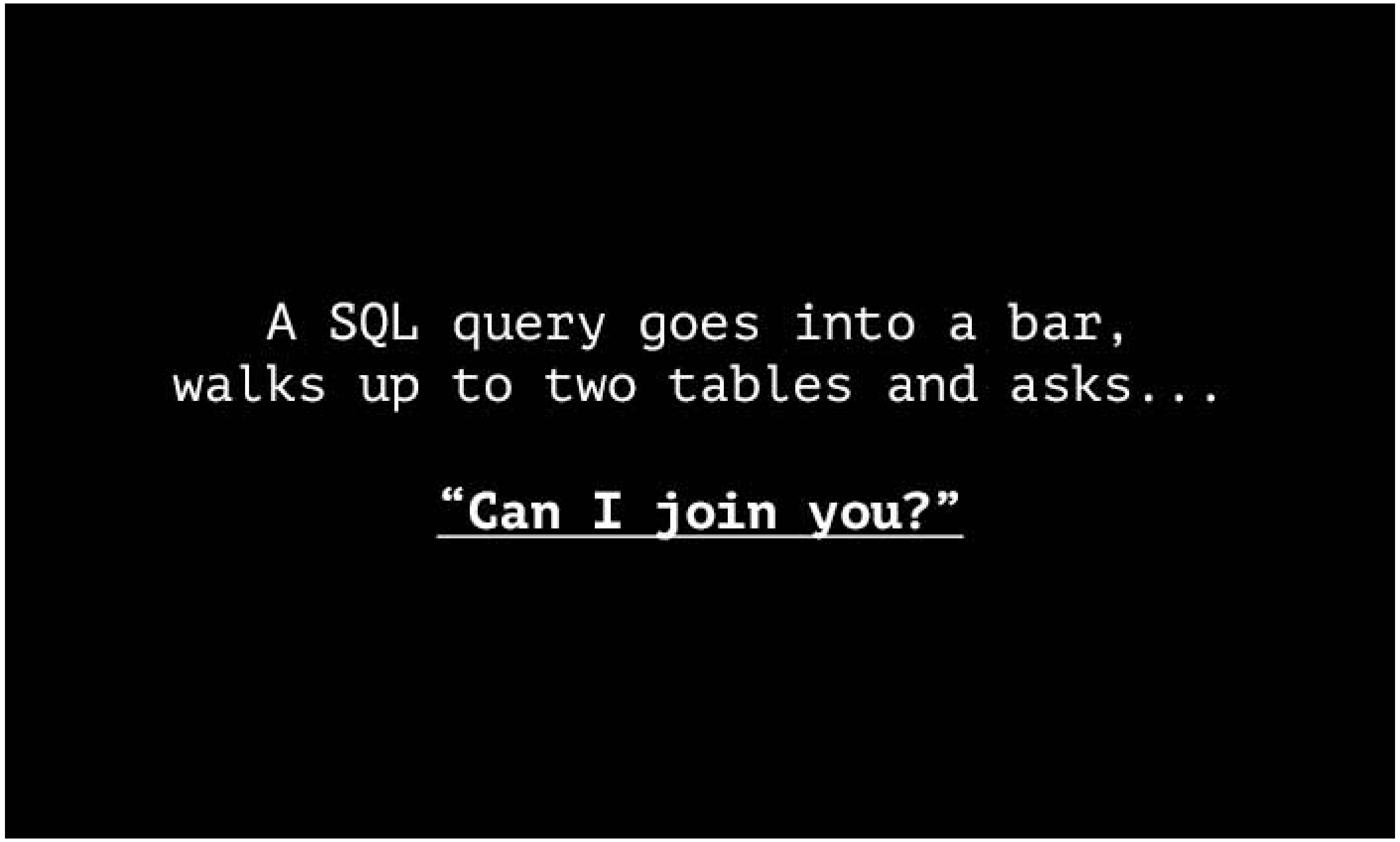
[–] sedmonster 12 points 3 hours ago
Go team!
permalink save report give gold reply pocket

[–] bradfitz 22 points 3 hours ago
Yes?
permalink save parent report give gold reply pocket

[–] enneff 17 points 2 hours ago
Yes?
permalink save parent report give gold reply pocket

[–] dsymonds 15 points 3 hours ago
yes?

More


A SQL query goes into a bar,
walks up to two tables and asks...

“Can I join you?”

Structs

```go
package main

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

type User struct {
    Firstname string
    Lastname  string
    Email     string
    Password  string
}

func (u *User) FullName() string {
    return fmt.Sprint(u.Firstname," ",u.Lastname)
}

func (u User) Name() string {
    return fmt.Sprint(u.Firstname," ",u.Lastname)
}



func main() {
        u := User{"girish", "ramnani", "", "girish"}
        u2 := &User{"Girish","Ramnani","","Girish"}
        u3 := new(User)
        fmt.Println(u,u.FullName())
        fmt.Println(u2,u2.FullName())
        fmt.Println(u3,u3.FullName())
        fmt.Println()
        b, _ := json.Marshal(u)
        os.Stdout.Write(b)

}
```

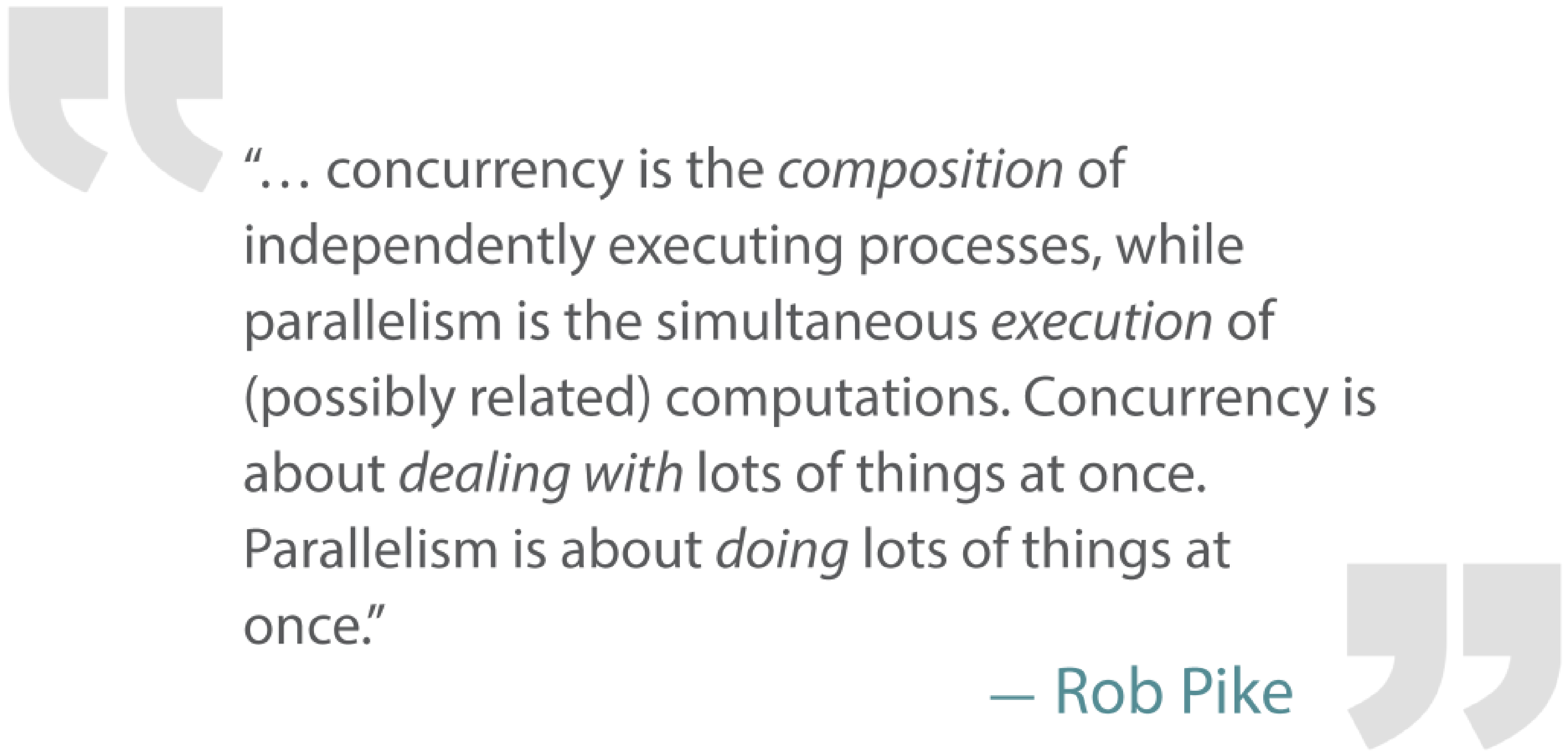
Try this on your laptop

```
func main() {
        u := User{"girish", "ramnani", "", "girish"}
        u2 := &User{"Girish","Ramnani","","Girish"}
        u3 := new(User)
        fmt.Println(u,u.FullName())
        fmt.Println(u2,u2.FullName())
        fmt.Println(u3,u3.FullName())
        fmt.Println()
        b, _ := json.Marshal(u)
        os.Stdout.Write(b)

}
```

Run

Concurrency



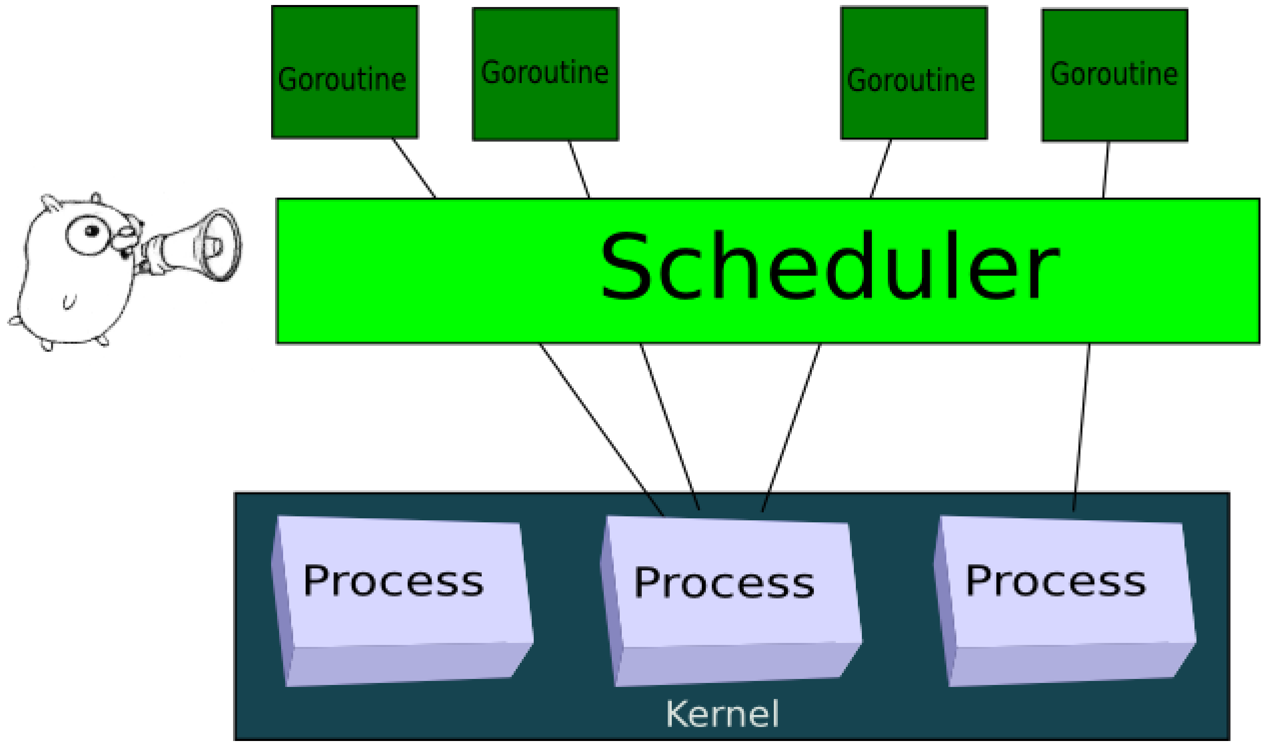
"… concurrency is the *composition* of independently executing processes, while parallelism is the simultaneous *execution* of (possibly related) computations. Concurrency is about *dealing with* lots of things at once. Parallelism is about *doing* lots of things at once."

— Rob Pike

Concurrency Model

Golang has one of the most powerful concurrency models

Go Routines


Goroutine
Goroutine
Goroutine
Goroutine
Scheduler
Process
Process
Process
Kernel

How to create a goroutine?

Just add "go" in front of a function

```go
package main

import (
    "fmt"
    "time"
)



func goroutine(i int) {
    fmt.Println("Waiting for",i,"ms")
    time.Sleep(time.Duration(i) * time.Millisecond)
}

func main() {

    go goroutine(200)
    go goroutine(300)
    go goroutine(400)

    fmt.Println("waiting for 1 second on main go routine")
    time.Sleep(1 * time.Second)
}
```

Run

Wait Groups

```
func main() {

    wg := sync.WaitGroup{}
    wg.Add(2)
    go func(wg *sync.WaitGroup) {
        defer wg.Done()
        fmt.Println("func 1")
        time.Sleep(2 * time.Second)

    }(&wg)

    go func(wg *sync.WaitGroup) {
        defer wg.Done()
        fmt.Println("func 2")
        time.Sleep(1 * time.Second)

    }(&wg)

    wg.Wait()
    fmt.Println("Fin")
}
```

Run

Exercise time

Error hunting

```go
package main

import (
    "fmt"
)

myvar := 1 //error

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

Run

Guess the output

```
package main

import "fmt"

func main() {
    x := [3]int{1,2,3}

    func(arr [3]int) {
        arr[0] = 7
        fmt.Println(arr)
    }(x)

    fmt.Println(x)
}
```

Run

Guess the output

```go
package main

import "fmt"

func main() {
    x := []int{1,2,3}

    func(arr []int) {
        arr[0] = 7
        fmt.Println(arr)
    }(x)

    fmt.Println(x)
}
```

Run

Guess the output

```go
package main

import "fmt"

func main() {
    data := "♥"
    fmt.Println(len(data))
}
```

Run

Guess the output

```
func main() {
    var a int8 = 3
    var b int16 = 4

    sum := a + b

    fmt.Println(sum)
}
```

Run

Guess the output

```
func main() {
    var wg sync.WaitGroup
    wg.Add(1)

    go func() {
        fmt.Println("1")
        wg.Done()
    }()
    wg.Add(1)

    go func() {
        fmt.Println("2")
        wg.Done()
    }()
    wg.Wait()
    fmt.Println("3")
}
```

Run

Code Smell

```go
package main

import "fmt"

func main() {
    x := map[string]string{"one":"a","two":"","three":"c"}

    if v := x["two"]; v == "" { //incorrect
        fmt.Println("no entry")
    }
}
```

Run

Guess the output (Intermediate)

```go
package main

import "fmt"

func doRecover() {
    fmt.Println("recovered =>",recover())
}

func main() {
    defer func() {
        doRecover()
    }()

    panic("not good")
}
```

Run

Thank you

Girish Ramnani
girishramnani95@gmail.com (mailto:girishramnani95@gmail.com)
https://girishramnani.github.io/ (https://girishramnani.github.io/)
@girishramnani95 (http://twitter.com/girishramnani95)