# Introduction to Google's Go

## Stratio / Datio (May 2016)

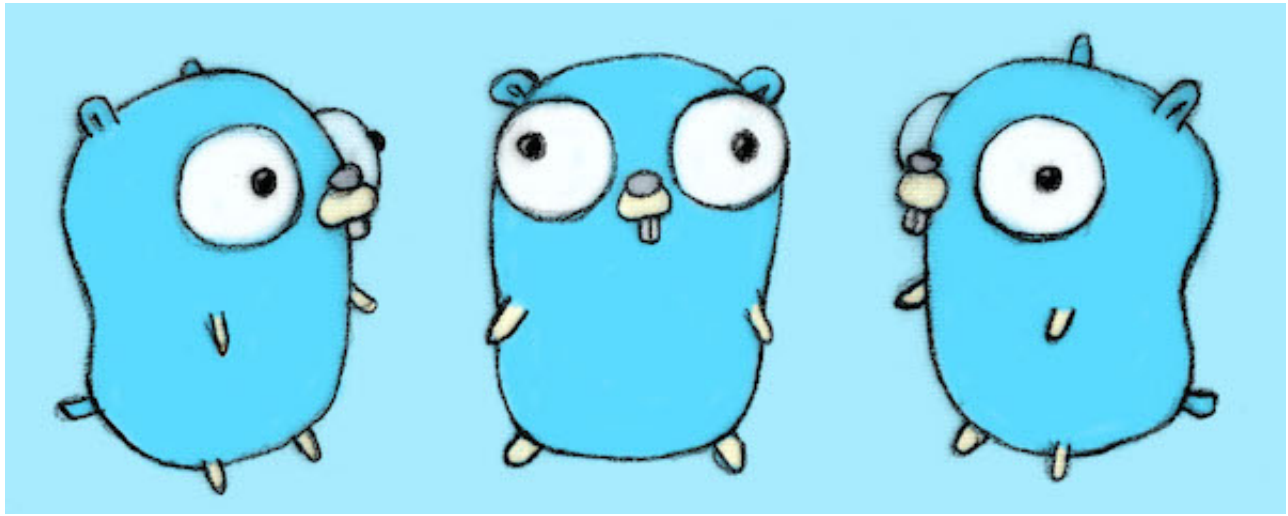
Mario Castro

# For those who don’t know me…

## For those who don’t know me...

- **Mario Castro**
- DevOps @ Stratio
- Started coding with 13 yo
- Java (J++) since 2000
- PHP, Python, Java Android, Objective-C, Node.js
- Pixar's Certified in Renderman's render engine
- Other scripting languages (MEL, HScript)

# Let's Go

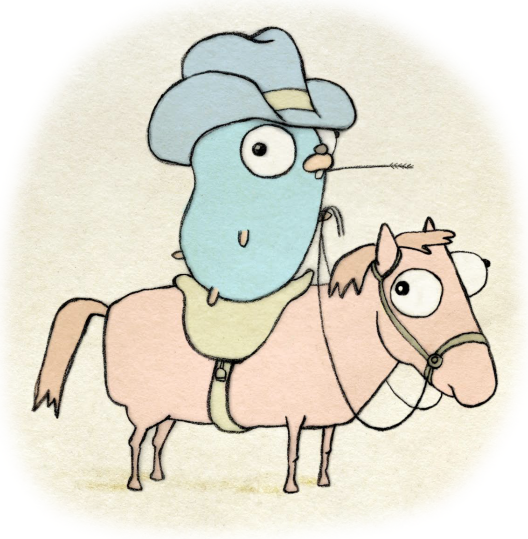


# When and who...

## History

Design began in late 2007.

- Robert Griesemer, Rob Pike, and Ken Thompson.
- Ian Lance Taylor and Russ Cox.
- Open source since 2009 with a very active community.
- Language stable as of Go 1, early 2012.



## Who's Go

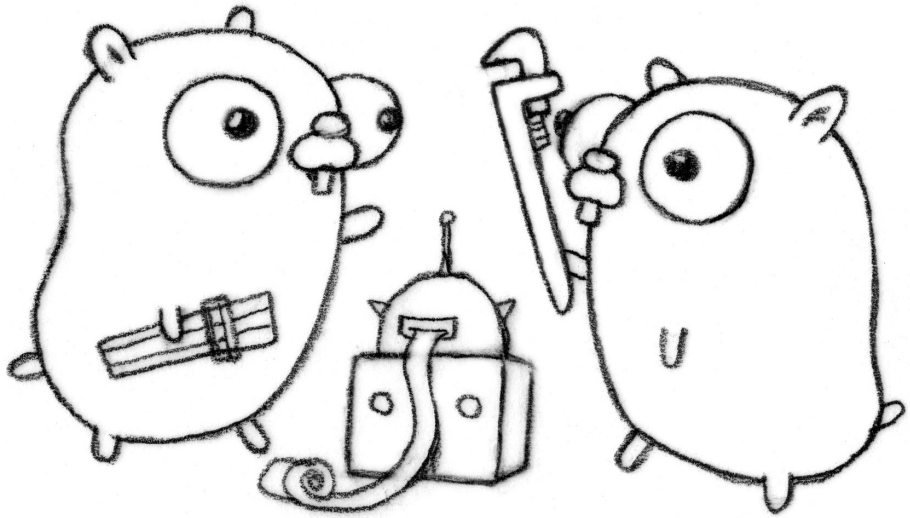
The men behind it...

- Ken Thompson (Ex Bell labs, Unix designer, B programming language)
- Rob Pike (Ex Bell labs, Unix Team, Inferno, Plan 9, UTF-8)
- Robert Griesemer (V8 Chrome engine, Sawzall)


# Why a new language

## Why a new language

- slow builds
- uncontrolled dependencies
- poor program understanding (code hard to read, poorly documented...)
- Computers are enormously quicker but software development is not faster.
- Some fundamental concepts such as garbage collection and parallel computation are not well supported by popular systems languages.

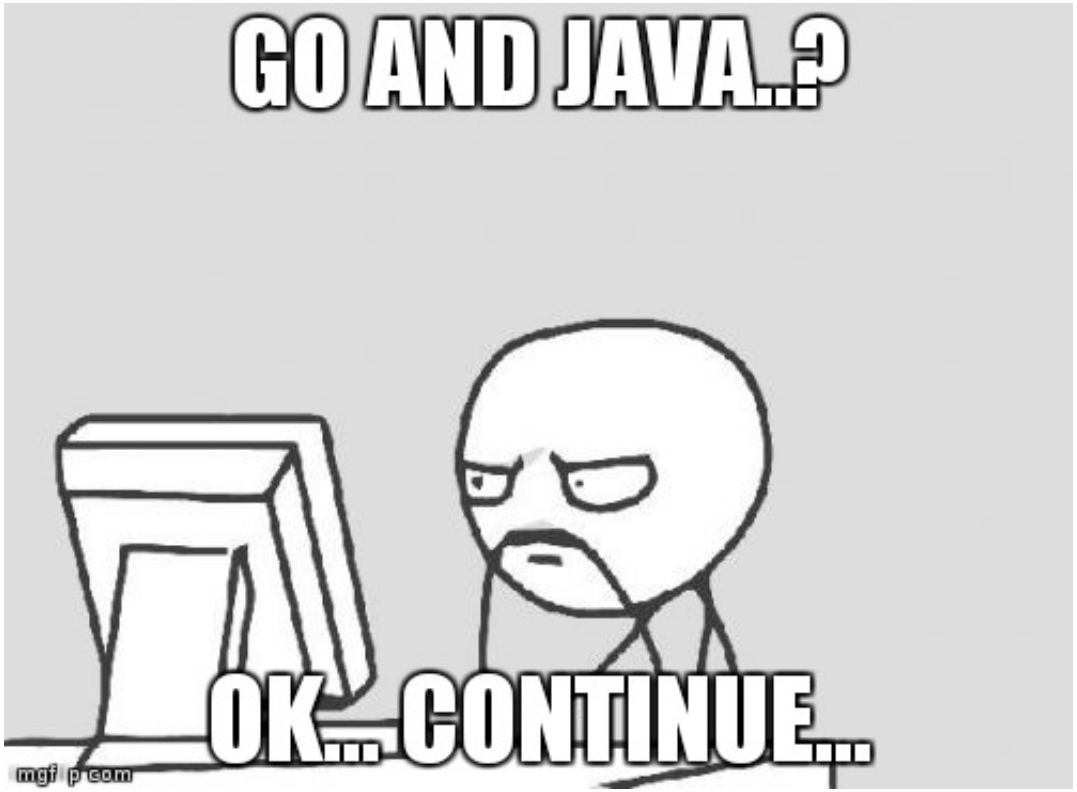


## What's Go (as a programming language)

- Statically typed (duck typing)
- Compiled language (no more virtual machines... thanks...)
- Structure oriented (no inheritance)
- Concise and simple syntax, easy to get started with
- Good facilities for writing concurrent programs that share state by communicating
- Unified formatting style for the language
- Compilation is fast even with large projects
- No need to know a new paradigm or awkward syntax

# Comparing Go and Java

# Go and Java have much in common



## Go and Java have much in common

- C family (imperative, braces)
- Statically typed
- Garbage collected
- Methods
- Interfaces
- Type assertions (`instanceof`)
- Reflection

# Classes vs Structs. Interfaces vs... Interfaces?

```java
public interface Animal {
  public void move();
}

public class Dog implements Animal {
  private int numberOfLegs;
  private boolean hasOwner;

  public Dog(int numberOfLegs, boolean hasOwner) {
    numberOfLegs = numberOfLegs;
    hasOwner = hasOwner;
  }

  private void bark() {
    System.out.println("Woof!");
  }

  @Override
  public void move() {
    System.out.println("Running!");
  }
}
```

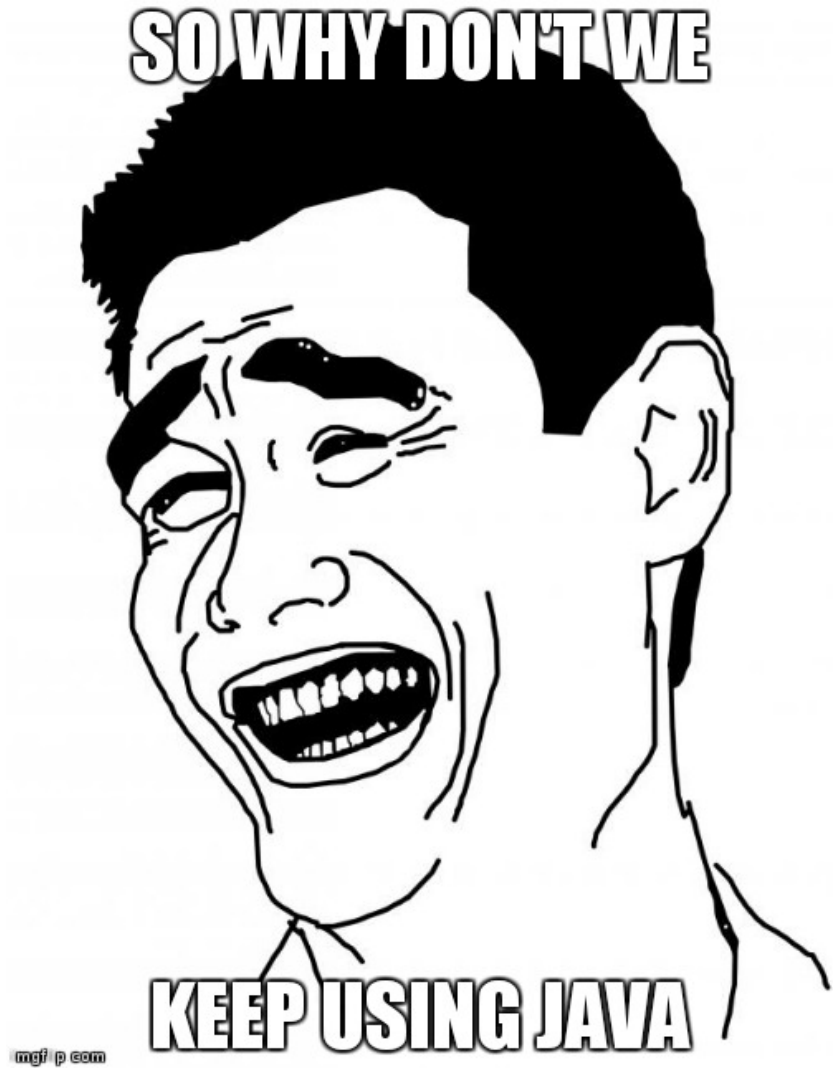
```go
type Animal interface {
  Move()
}

type Dog struct {
  numberOfLegs int
  hasOwner     bool
}

func (d *Dog) bark() {
  println("Woof!")
}

func (d *Dog) Move() {
  println("Running")
}
```

# Go and Java have much in common



## This is not a competition

- Go is not trying to replace Java
- Go has focus on productivity and consolidating Google's experience in large distributed systems
- Google uses Java, Javascript, C++, Python, Go, Sawzal and probably a few other languages are supported (Jeff Nelson, inventor of Chrome OS)

## Go differs from Java in several ways

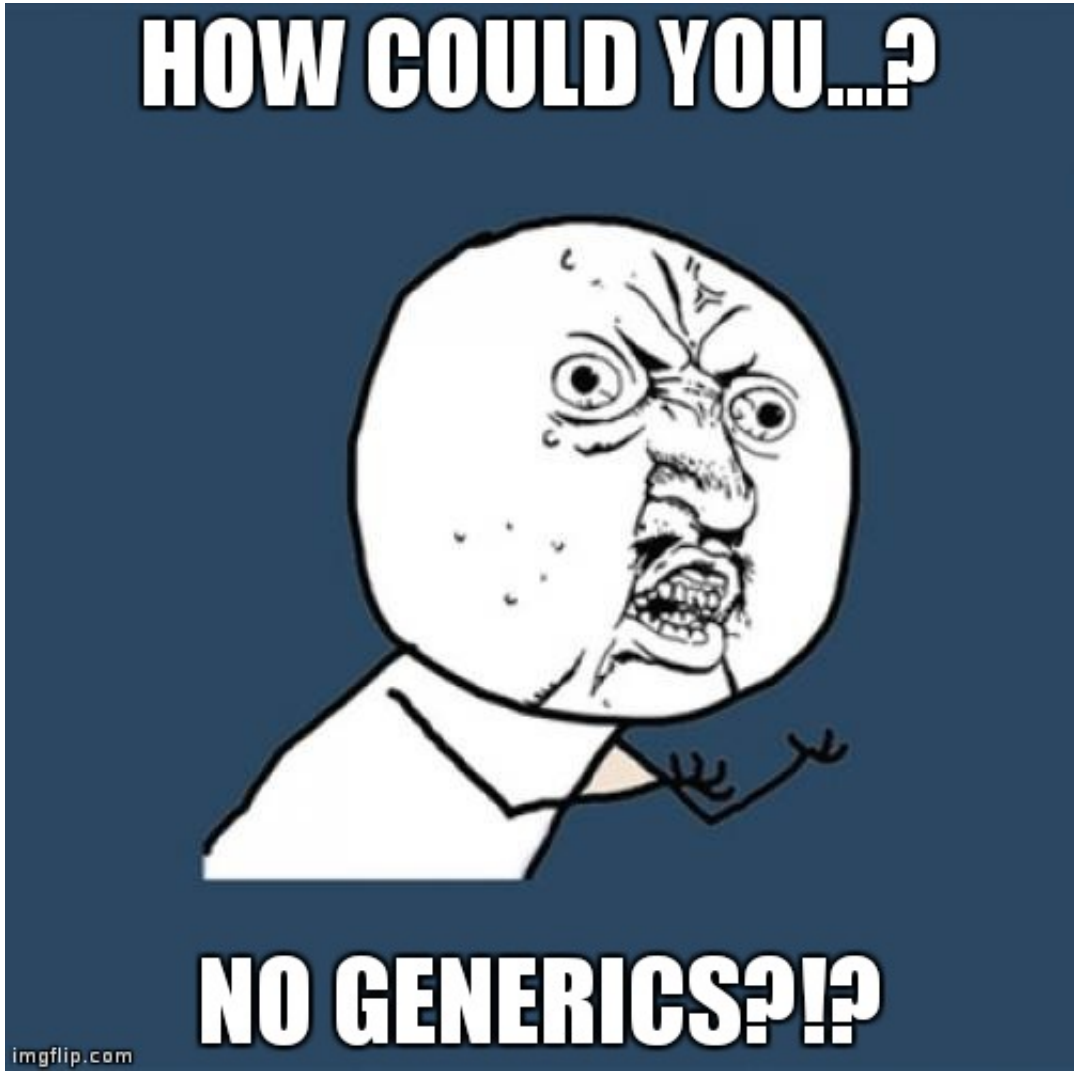
- Programs compile to machine code. There's no VM.
- Statically linked binaries
- Built-in strings (UTF-8)
- Built-in generic maps and arrays/slices
- Built-in concurrency

## Go intentionally leaves out many features

- No classes
- No constructors
- No inheritance
- **No user-defined generics**

No `final`, exceptions, annotations...

# Go intentionally leaves out many features



## Seriously...

No, there are not generics nor algebraic types, monads...

## Why does Go leave out those features?

- Clarity is critical.

```
class Foo[F[+_] : Monad, A, B](val execute: Foo.Request[A] => F[B], val joins: Foo.Request[A] =>

def bar: Foo[({type l[+a]=WriterT[F, Log[A, B], a]})#l, A, B] = {
    type TraceW[FF[+_], +AA] = WriterT[FF, Log[A, B], AA]
    def execute(request: Request[A]): WriterT[F, Log[A, B], B] =
      self.execute(request).liftM[TraceW] :++>> (repr => List(request -> request.response(repr,
```

- When reading code, it should be clear what the program will do.
- When writing code, it should be clear how to make the program do what you want.
- Sometimes this means writing out a loop instead of invoking an obscure function.

## So, Go is not...

- ...a functional language
- ...a new fancy way of coding
- ...a completely open-source project (no pull-requests)
- ...a super language that will replace us
- ...talking about replacement... is not the replacement of Java... or C++... or Cobol...

# Why Go

- Make programming fast
- Safety: type-safe and memory-safe
- Efficient gargage collector
- Reduce typing. No more:

```
foo.Foo *myFoo = new foo.Foo(foo.FOO_INIT)
```

- Focus on productivity and concurrency of distributed systems with CSP concurrency model

# CSP Concurrency

## CSP Concurrency

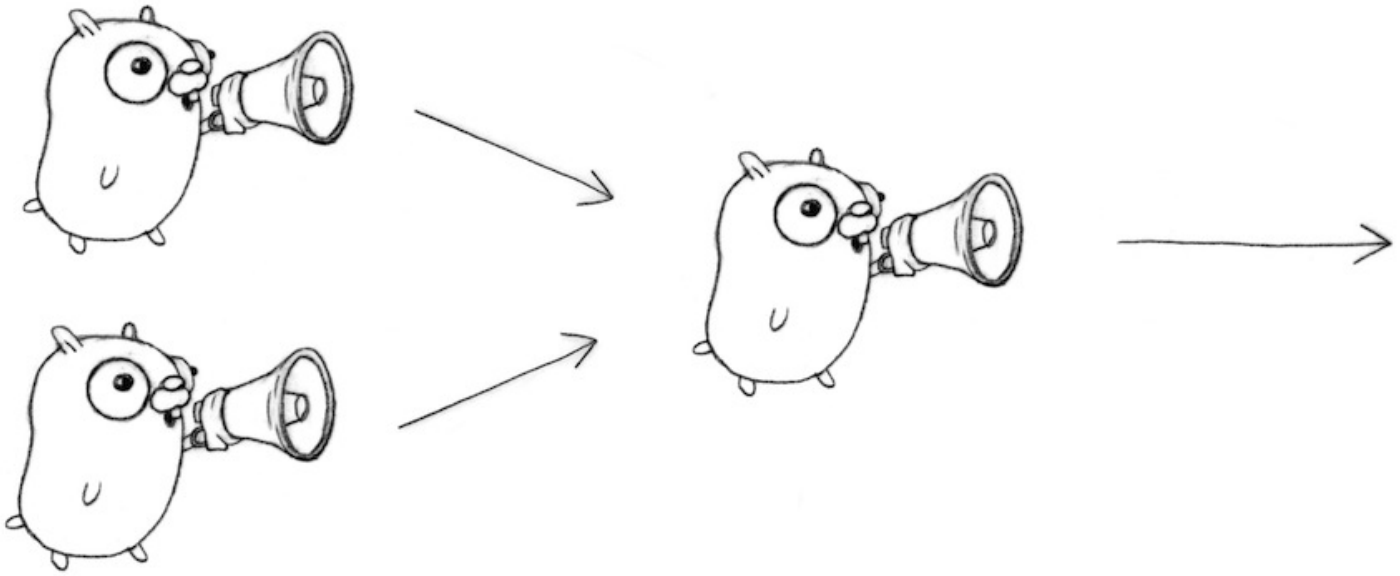
- **Actor** model: Entities passing messages to each other that are queued (Erlang, Scala, Java...)
- **CSP** (Communicating sequential processes) model. Proccesses passing messages to channels that blocks until the messages are taken but with possibility of using queues

CSP represents the best known way to write software and organize services together to form a system. Nature itself provides the best examples; even the human body is a system of interconnected services — respiratory, cardiovascular, nervous, immune, etc.

> The Tao of Hashicorp https://www.hashicorp.com/blog/tao-of-hashicorp.html (https://www.hashicorp.com/blog/tao-of-hashicorp.html)

- This was my face when I read the Tao of Hashicorp...

# CSP Concurrency



- "Gophers" = processes
- Arrows = channels
- One "Gopher" could listen to zero or more channels
- One "Gopher" can send messages to zero or more channels

## CSP Concurrency

- Process are anonymous (you can just reach them using a channel)
- Channels can be buffered or unbeffered
- Channels can be bi-directional or uni-directional
- More than one proccess could be listening the same channel
- One process could listen more than one channel
- More about this in the workshop

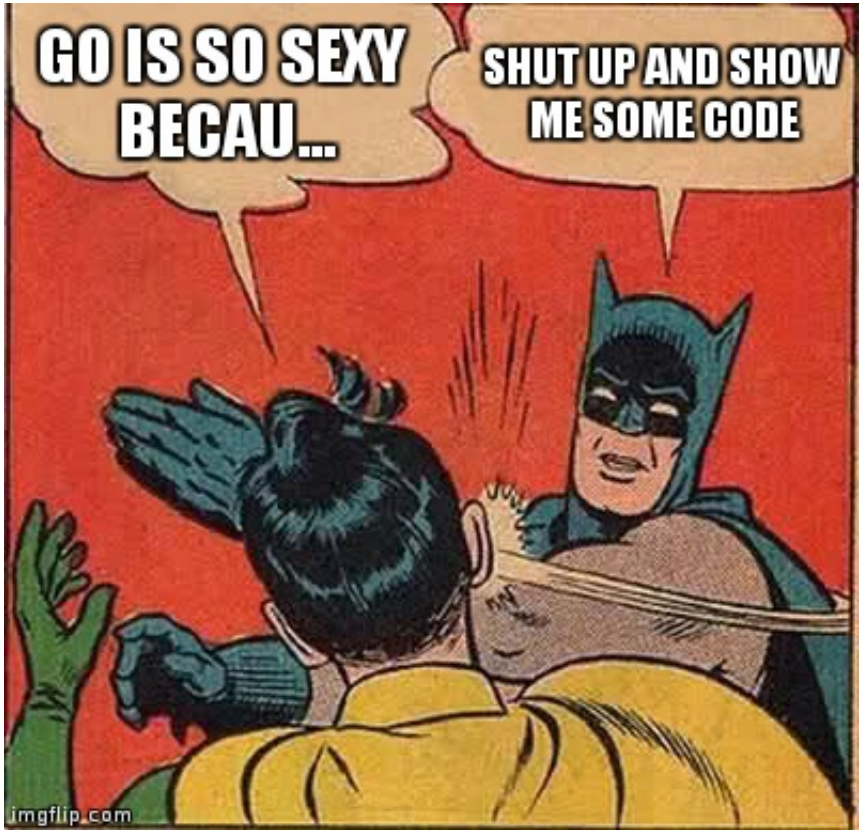
# An opinionated language

## An opinionated language

Can be weird at the beginning

- No brackets in new line (formatter will put them up again)
- An Uppercase name means that some method or variable is public, lowercase for private
- Comments of public methods must start with the name of the method
- Your project must reside within $GOPATH
- Parameters in the definition of a function must be one character
- If you return more than one value, last must be an error
- No more than one file with "package main" even if you only have one "main" function (just works with `go build` but not with `go run [file]`)
- Imports points to some kind of URL that must match a path within your $GOPATH

# An opinionated language



# Hello World

# Hello World

```
package main

func main() {
    println("Hello World!")
}
```

Run

# Hello world

# Hello web-server

```
package main

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

func main() {
    http.HandleFunc("/hello", handleHello)
    fmt.Println("serving on http://localhost:7777/hello")
    log.Fatal(http.ListenAndServe("localhost:7777", nil))
}

func handleHello(w http.ResponseWriter, req *http.Request) {
    fmt.Fprintln(w, "Hello, Stratio")
}
```

Run

# More examples: Native Unix piping

```
func main() {
    c1 := exec.Command("ls", "-lh", "/")
    c2 := exec.Command("awk", "{print $5, \"\011\" $9}")

    r1, w1 := io.Pipe()
    c1.Stdout = w1
    c2.Stdin = r1

    c2.Stdout = os.Stdout

    c1.Start()
    c2.Start()
    c1.Wait()
    w1.Close()
}
```

Run

# Inferred types

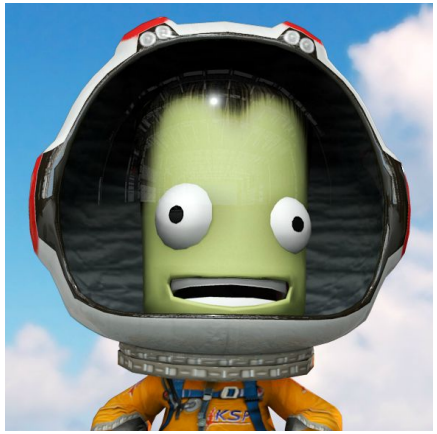
- No need to write variables types

```
package main

import (
    "fmt"
    "reflect"
)

func main() {
    me := "Jebediah Kerman"
    fmt.Println(reflect.TypeOf(me))
}
```

Run



# Some common coding features and patterns

- More than one object on return (error always at the end)

```
func myFunc() (struct1, struct2, error)
```

- Delegate error pattern

```
return nil, err       //Error ocurred, delegate to caller

return o, nil         //No error
```

- Pointers and references

```
o := myObject{}               //"o" is an object

doSomething(&o)               //Passing a pointer. "o" is still an object

o := &myObject{}              //Pointer to object. "o" is a pointer

func useObject(o *myObject)   //A received pointer "o" is a pointer
```

# Some commong coding features and patterns

- Handle every error

```
if err != nil {
  log.Fatal(err)
}
```

- Ignore a returned variable (very bad practice, don't trust code that do this)

```
myobj, _ := myFuncThatReturnsAnObjectAndAnError()
```

- Return anything (quite common in Consul code)

```
func myFunc() interface{}
```

- Do some type assertion

```
obj := myFunc()
myString, ok := obj.(string)
if ok {
  println(myString)
}
```

# Testing

- Comes with its own test suite

```
$ go test .
$ ok      github.com/thehivecorporation/raccoon/parser    0.003s
```

- Getting output...

```
$ go test -v .
$ INFO[0000] ------------------------------> Reading zombiebook file  zombiebook=/tmp/i-do-not-e
$ INFO[0000] ------------------------------> Reading zombiebook file  zombiebook=/tmp/wrongJson8
$ INFO[0000] ------------------------------> Reading zombiebook file  zombiebook=../examples/exa
$ INFO[0000] ------------------------------> Reading zombiebook file  zombiebook=/tmp/no-conten
$ INFO[0000] ------------------------------> Reading zombiebook file  zombiebook=/tmp/incorrect-
$ INFO[0000] ------------------------------> Reading zombiebook file  zombiebook=/tmp/incorrect-
$ INFO[0000] ------------------------------> Reading zombiebook file  zombiebook=/tmp/incorrect-
$ INFO[0000] ------------------------------> Reading zombiebook file  zombiebook=/tmp/incorrect-
$ --- PASS: TestReadZbookFile (0.00s)
$     zbook_test.go:18: Error reading zombiebook file: open /tmp/i-do-not-exist: no such file or
$     zbook_test.go:32: Error parsing JSON: invalid character 'w' looking for beginning of objec
$ PASS
$ ok      github.com/thehivecorporation/raccoon/parser    0.003s
```

# Testing


BE CAREFUL WITH RACE CONDITIONS
DON'T WORRY. I HAVE A RACE DETECTOR
imgflip.com

# Testing

- Race condition evaluation integrated

```
$ go test -race .
$ ok      github.com/thehivecorporation/raccoon/parser    1.013s
```

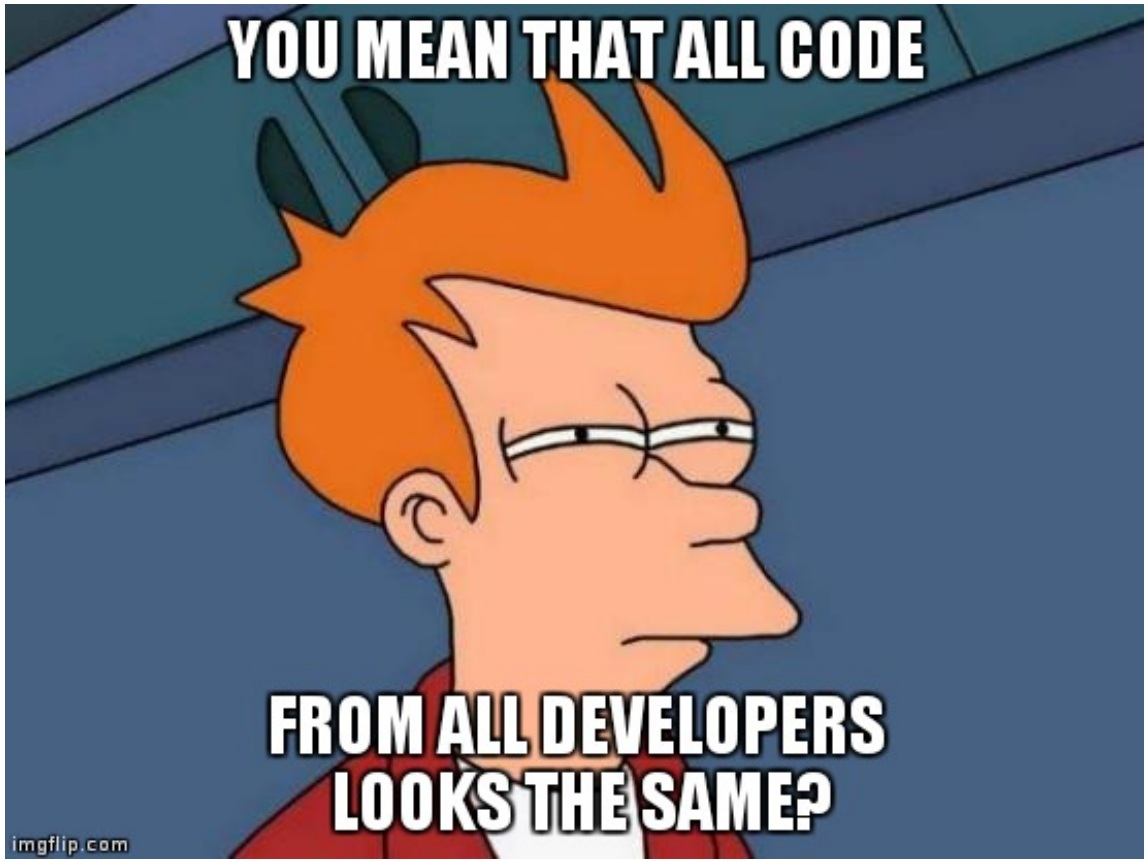
- Comes with its own coverage support

```
$ go test -race -cover .
$ ok      github.com/thehivecorporation/raccoon/parser    1.011s    coverage: 68.3% of statemen
```

# Other tools

# gofmt

- `gofmt` formats indentation, spaces, newlines and syntax
- Makes most code familiar.



# golint

- `golint` warns you about good practices and suggestions
- Example:

```
type Executor interface {
    Execute()
}
```

```
linter.go:3:6: exported type Executor should have comment or be unexported
```

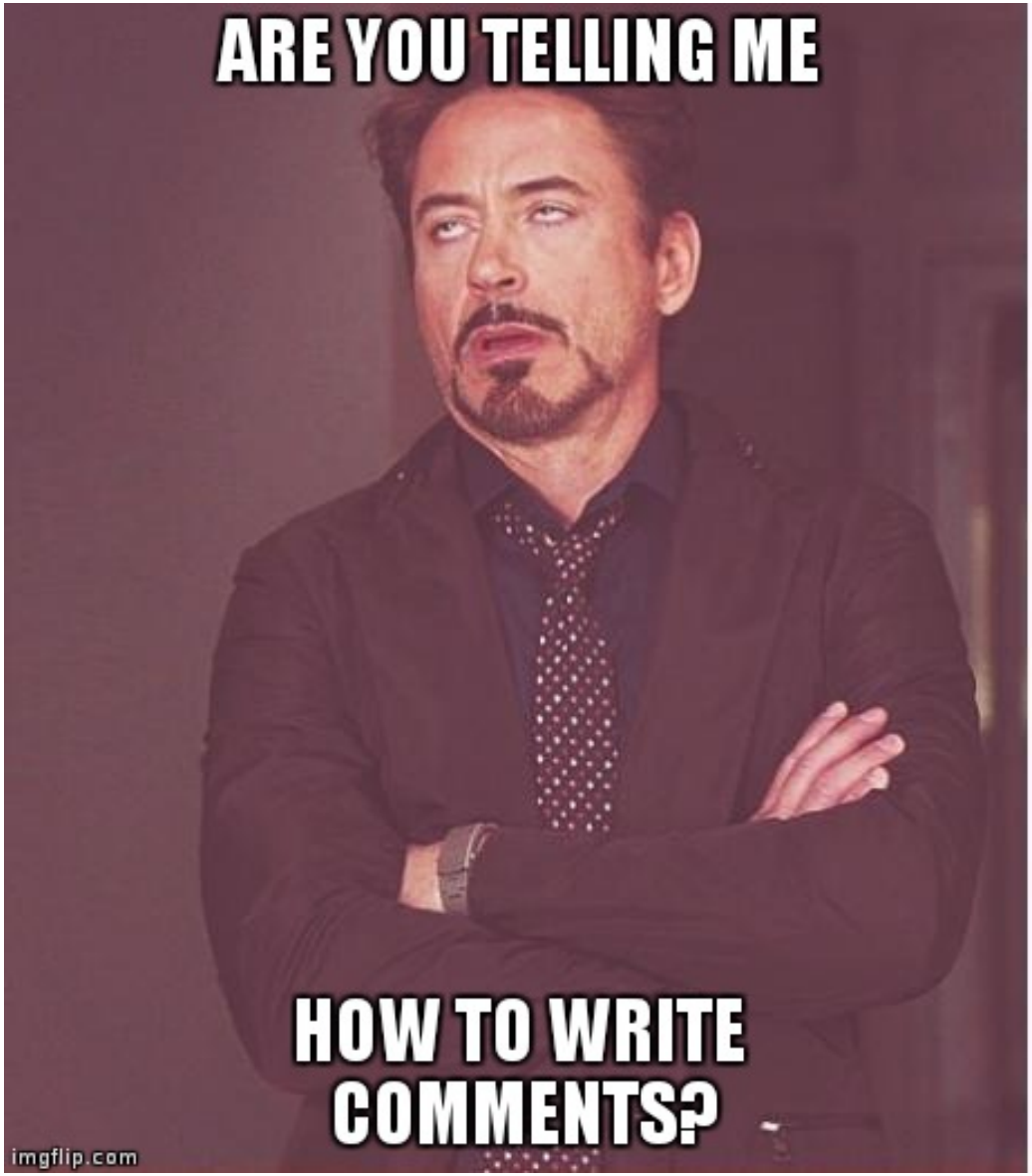
- Ok so...

```
//Interface for execution
type Executor interface {
    Execute()
}
```

```
linter2.go:3:1: comment on exported type Executor should be of the form "Executor ..." (with op
```

```
//Executor is the strategy pattern for a...
type Executor interface {
    Execute()
}
```

# golint



# golint

- If the name always begins the comment, the output of godoc can usefully be run through grep. Imagine you couldn't remember the name "Compile" but were looking for the parsing function for regular expressions, so you ran the command,

```
$ godoc regexp | grep parse
  Compile parses a regular expression and returns, if successful, a Regexp
  parsed. It simplifies safe initialization of global variables holding
  cannot be parsed. It simplifies safe initialization of global variables
$
```


Found in "Effective Go" (https://golang.org/doc/effective_go.html#commentary)

# GoDoc

- All core packages at your fingertips
- You can perform queries in command line
- Or open a web server to search

```
godoc index=true -http=:6060
```

- Search in a specific package

```
godoc fmt | grep -i read
func Fscanln(r io.Reader, a ...interface{}) (n int, err error)
    Scan scans text read from standard input, storing successive
    Scanf scans text read from standard input, storing successive
    // ReadRune reads the next rune (Unicode code point) from the input.
    // If invoked during Scanln, Fscanln, or Sscanln, ReadRune() will
    // return EOF after returning the first '\n' or when reading beyond
    ReadRune() (r rune, size int, err error)
    // UnreadRune causes the next call to ReadRune to return the same rune.
    UnreadRune() error
    // Because ReadRune is implemented by the interface, Read should never be
    // ScanState may choose always to return an error from Read.
    Read(buf []byte) (n int, err error)
```

# Go Get

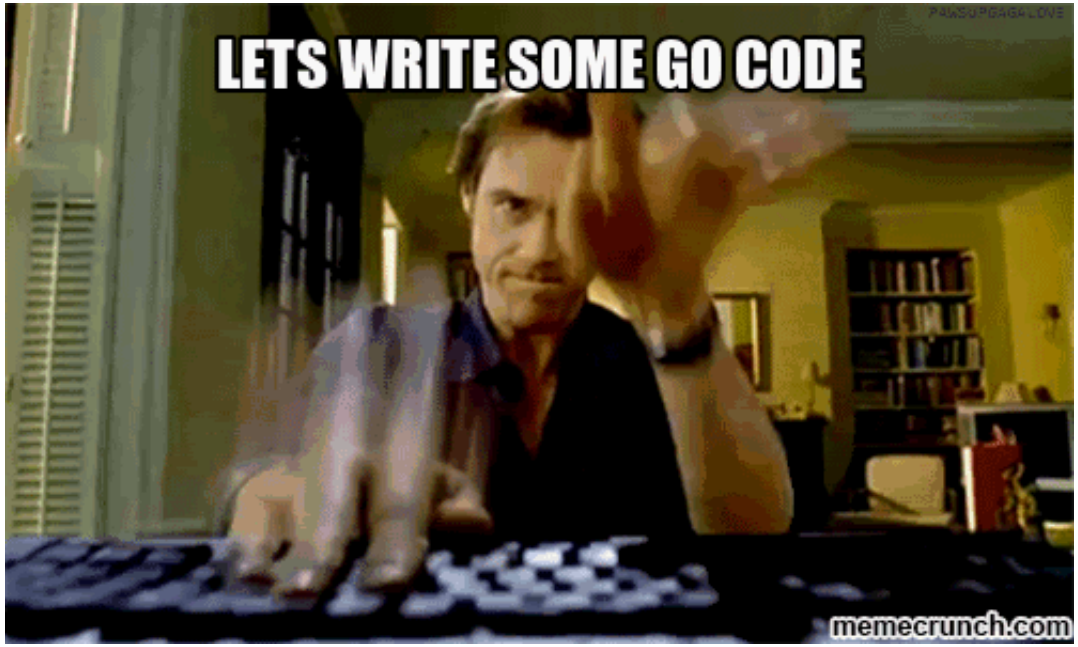
- Useful for dependency management
- As simple as

```
$ go get github.com/kubernetes/kubernetes
import "github.com/4ad/doozer"
var client doozer.Conn
```

- Most IDE's has an auto-import (on save) features

# Workflow

# Workflow



## Installing Go

From repositories
- Ubuntu users: `sudo apt-get install -y golang`
- Centos/RHEL users: `sudo yum install -y golang`
- Fedora: `sudo dnf install -y golang`

# Installing Go

# Ok... the slightly harder one: Installing Go

Latest build

golang.org (http://golang.org)

- Select your distro
- unpack it `tar -zxvf go1.6.2.linux-amd64.tar.gz`
- Move the go folder to your favorite destination (assign permissions according to destination)
- Add a $GOROOT environment variable pointing to your go folder (**not the bin folder within your go folder**) to /etc/profile, ${HOME}/.bashrc, etc.
- Add a $GOPATH environment variable pointing to an empty folder that will represent your global workspace for Go
- Add $GOPATH/bin and $GOROOT/bin to your $PATH

## The workspace

Workspace is global and it's defined as an environment variable called $GOPATH.

```
$ export GOPATH=${HOME}/go
```

So a `go get github.com/docker/docker` will put the source in

```
$ go get github.com/docker/docker
$ cd $GOPATH/src/github.com/docker/docker
```

Easy, uh?

# Our first Go App

```
package main

func main() {
    println("Hello world")
}
```

Run

# Our second Go App

```
package main

import "fmt"

func main() {
    fmt.Printf("(P1)Hello")

    go World()

}

func World() {
    fmt.Printf(" world(P2)\n")
}
```

# Our third Go App

```
package main

import "fmt"

func main() {
    for i := 0; i < 10000; i++ {
        go func(j int) {
            fmt.Printf("Proccess #%d\n", j)
        }(i)
    }

    var input string
    fmt.Scanln(&input)
}
```

# Our fourth Go App

```
func main() {
    workersCh := make(chan int, 20)
    for i := 0; i < 20; i++ {
        go worker(i, workersCh)
    }

    iter := 0
    for {
        workersCh <- iter
        iter++
        time.Sleep(100 * time.Millisecond)
    }
}

func worker(id int, w chan int) {
    for {
        select {
        case number := <-w:
            fmt.Printf("Worker %d got the number %d\n", id, number)
            time.Sleep(3 * time.Second)
        }
    }
} //END
```

# Our fifth Go App

```
var delay time.Duration = 5000

func main() {
    dispatcherCh := make(chan int, 100)
    workersCh := make(chan int)

    for i := 0; i < 20; i++ {
        go worker(i, workersCh)
    }

    go dispatcher(dispatcherCh, workersCh)

    iter := 0
    for {
        fmt.Printf("Queue has size %d\n", len(dispatcherCh))
        if len(dispatcherCh) == 100 {
            delay = 100
        } else if len(dispatcherCh) == 0 {
            delay = 5000
        }
        dispatcherCh <- iter
        iter++
        time.Sleep(100 * time.Millisecond)
    }
}
```

```
func dispatcher(c, w chan int) {
    for {
        v := <-c
        w <- v
    }
}

func worker(id int, w chan int) {
    for {
        select {
        case number := <-w:
            fmt.Printf("Worker %d got the number %d\n", id, number)
            time.Sleep(delay * time.Millisecond)
        }
    }
} //END
```

# Our sixth...

## Contributing in Github

Workflow is slightly different

- Fork the project into your own github account
- **Don't clone your fork!** Use `go get` of the original git project
- As mentioned earlier, it will create a folder within your `$GOPATH/src/[project_owner]/[repo]`
- Add your fork as a differente remote (`git remote add my_fork https://github.com/[my_account]/[my_fork]`)
- Work as usual
- Push your changes to your fork
- Open pull request as usual

(At least, this is how I do it)

## Part of Tens

- No inheritance
- Strongly typed
- No “git clone”. Use “go get”
- Strongly opinionated
- No generics
- Has pointers and references
- No unused variables nor imports
- Strings and structs can’t be nil
- You must work always in $GOPATH
- No need of locks or semaphores with channels

## IDE's and other editing tools

- vim + vim-go
- emacs + go-model.el
- Intellij Idea (has debugging support already. Thanks Abel!)
- Atom + go-plus (Has debugging support using Delve plugin)
- Sublime Text + GoSublime
- Visual Studio Code + vscode-go
- LiteIde

## Companies using Go

- Google
- Docker
- Netflix
- Parse
- Digital Ocean
- Ebay
- AirBnB
- Dropbox
- Uber
- VMWare

## Famous software written in Go

- Docker
- Kubernetes (Google)
- Hashicorp's stack (Consul, Terraform, Vault, Serf, Otto, Nomad, Packer)
- Prometheus (SoundClound)
- CoreOS stack (ETCD, Fleet, Rkt, Flannel)
- InfluxDB
- Grafana

# Best resources to learn Go (ordered)

- A Tour of Go (interactive tutorial)

  https://tour.golang.org/list (https://tour.golang.org/list)

- Go By Example

  https://gobyexample.com/ (https://gobyexample.com/)

- How to install Go workspace

  https://golang.org/doc/code.html (https://golang.org/doc/code.html)

- Go bootcamp (free ebook)

  http://www.golangbootcamp.com/ (http://www.golangbootcamp.com/)

- "Effective Go" Wait a week or two to read

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

# Other references

- Docker and Go: Why we decide to write Docker in Go?

http://www.slideshare.net/jpetazzo/docker-and-go-why-did-we-decide-to-write-docker-in-go (http://www.slideshare.net/jpetazzo/docker-and-go-why-did-we-decide-to-write-docker-in-go)

# By the way

# This presentation...

```
...is also done with a Go tool
```

# Questions?

# Thank you

Mario Castro
Gopher @ StratioBD (mailto:Gopher%20@%20StratioBD)
@110010111101011 (http://twitter.com/110010111101011)
github.com/sayden
mcastro@stratio.com (mailto:mcastro@stratio.com)