

# Go - Introduction

29 March 2017

# SAP Hybris (v)

## Speakers

Piotr Mścichowski

piotr.mscichowski (at) hybris.com

Adam Szecówka

adam.szecowka (at) hybris.com

Brought to you by Hack Your Career

More at FB Page <https://www.facebook.com/Hack.your.Career>



Go is a programming language designed by Google to help solve Google's problems, and Google has big problems

# Problems at Google

- long builds
- each programmer using a different subset of the language
- popular languages are obsolete: C++ (1983), Java(1995), Python(1991)

**"When builds are slow, there is time to think"**

# Go - Key concepts

- \* Scalability

- \* Easy syntax

- similar to other known languages

- but without repetitions

```
final List<String> list = new ArrayList<String>();
```

Java

- \* Concurrency

- goroutines

- channels

- \* Garbage Collector

## Build time - examples

```
~/h/r/y/g/s/s/g/gandalf-supervisor (develop) $ time go build
0.84 real    0.86 user    0.18 sys
~/h/r/y/g/s/s/g/gandalf-supervisor (develop) $
```

Build time - production microservice ~4000 LOC

```
~/h/r/y/g/s/s/g/gandalf-supervisor (develop) $
~/h/r/y/g/s/s/g/gandalf-supervisor (develop) $ time go test ./...
?      stash.hybris.com/gopher/gandalf-supervisor      [no test files]
?      stash.hybris.com/gopher/gandalf-supervisor/internal      [no test files]
ok     stash.hybris.com/gopher/gandalf-supervisor/internal/deselector 0.078s
ok     stash.hybris.com/gopher/gandalf-supervisor/internal/expirator 0.010s
```

```
?      stash.hybris.com/gopher/gandalf-supervisor/vendor/stash.hybris
?      stash.hybris.com/gopher/gandalf-supervisor/vendor/stash.hybris
1.94 real    8.59 user    1.62 sys
~/h/r/y/g/s/s/g/gandalf-supervisor (develop) $
~/h/r/y/g/s/s/g/gandalf-supervisor (develop) $
```

Tests Execution



## Build time - Go vs Java

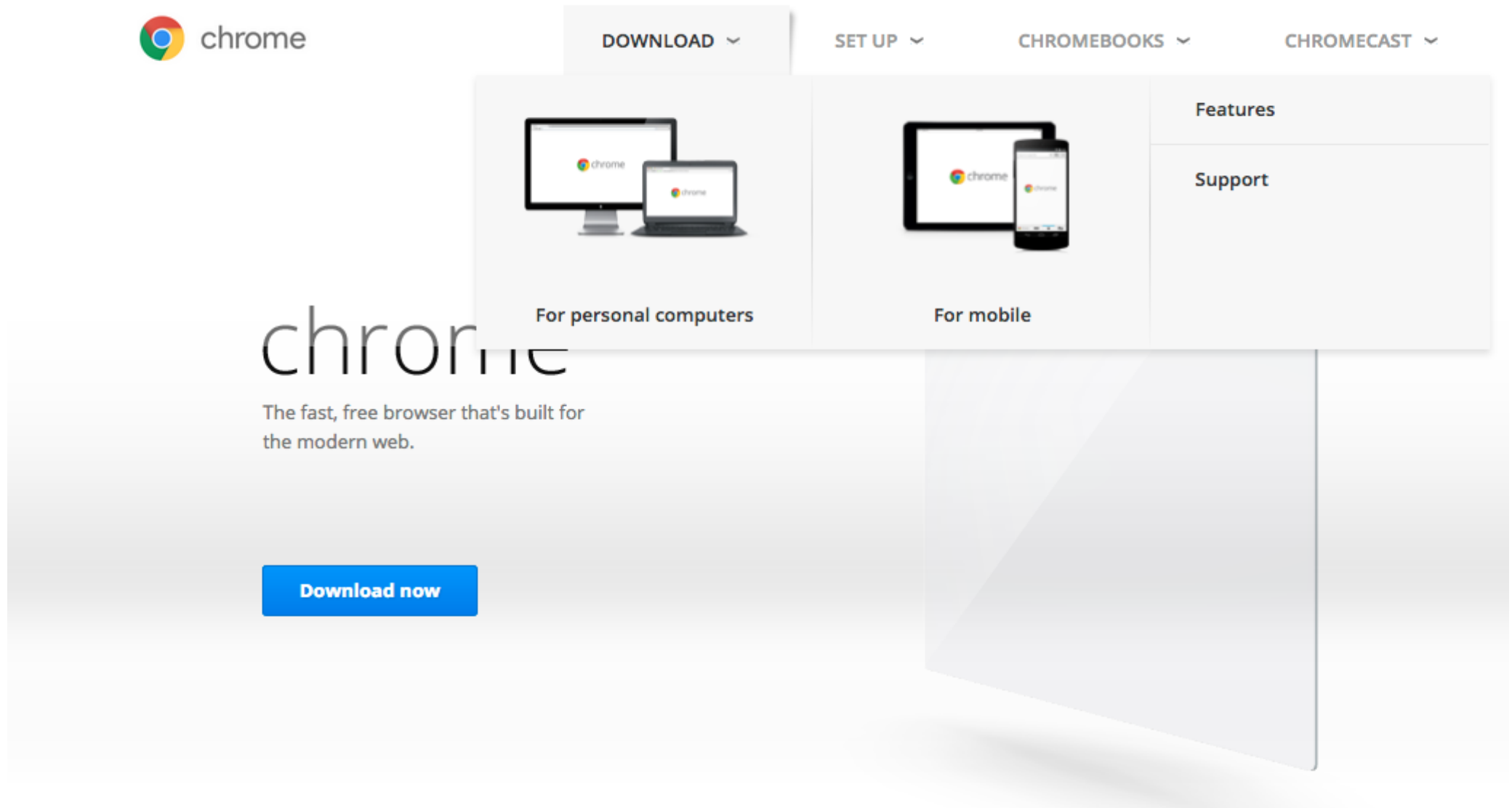
- Simple program: Quadratic equation (~150 LOC)
- Java: maven + junit
- Java: mvn clean package [s]: 3.19, 2.83, 3.48, 3.53, 3.57, 2.73. Avg: **3.22**
- go build [s]: 0.19 \* 3, 0.18 \* 2, 0.26. Avg: 0.198
- go test [s]: 0.41 \* 2, 0.40 \* 2, 0.43, 0.39. Avg: 0.406
- go total [s] = 0.198 + 0.406 = **0.604**
- Diff: 2.61 [s] per build. **5 times faster!**

## Fast build in go achieved by

- reduced number of open files
- no cyclic imports
- no types hierarchy
- unused import cause compilation error

# Who uses Go?

- Google: dl.google.com



# Who uses Go?

- Docker

## Under the hood

---

Under the hood, Docker is built on the following components:

- The [cgroups](#) and [namespaces](#) capabilities of the Linux kernel
- The [Go](#) programming language
- The [Docker Image Specification](#)
- The [Libcontainer Specification](#)

# Who uses Go?

- InfluxDB (Time-series data storage)

Influxdb V1.1

Search the docs...


## Key Features

Here are some of the features that InfluxDB currently supports that make it a great choice for working with time series data.

- Custom high performance datastore written specifically for time series data. The TSM engine allows for high ingest speed and data compression.
- Written entirely in Go. It compiles into a single binary with no external dependencies.

# Who uses Go?

- Cloud Foundry



## Cloud Foundry

Cloud Foundry Foundation active projects

Worldwide <https://www.cloudfoundry.org> [cf-dev@cloudfoundry.org](mailto:cf-dev@cloudfoundry.org)

Repositories

People 223

Type: All Language: Go

92 results for repositories written in Go Clear filter

### grootfs

Garden root file system

Go ★ 3 🍴 3 Updated 9 minutes ago

### diego-release

BOSH Release for Diego


Go ★ 147 🍴 143 Updated 3 hours ago

#### Top languages

Go Ruby Shell HTML Java

#### People

223 >



## Who uses Go?

- Uber

GENERAL ENGINEERING

# HOW WE BUILT UBER ENGINEERING'S HIGHEST QUERY PER SECOND SERVICE USING GO

FEBRUARY 24, 2016

BY KAI WEI

# Go getting started

- Get and install Go

From [here](https://golang.org/dl/) (https://golang.org/dl/)

- Set up GOPATH!

```
.
├── bin
├── pkg
│   └── darwin_amd64
└── src
    ├── 9fans.net
    ├── github.com
    ├── golang.org
    ├── gopkg.in
    ├── gorillas
    ├── hack-your-career
    ├── mongo-bongo
    ├── stash.hybris.com
    └── vimgotest
```



# Library imports

```
package myPackage

import (
    "testing"
    myname "strings"
    . "github.com/smartystreets/goconvey/convey"
)

func TestMe(t *testing.T) {
    Convey("'ala ma kota' should", t, func() {
        testString := "ala ma kota"

        Convey("contain 'kota", func() {
            zawiera := myname.Contains(testString, "kota")
            So(zawiera, ShouldBeTrue)
        })
    })
}
```

# Packages

- packages

```
package main
import "fmt"

func main() {
    fmt.Printf("Hello, world %d", 123)
}
```

Run

- to install package use `go get $REPOSITORY`

# Access modifiers

- Private, protected, public ?

## //Java style

```
public String modelowanieCyfrowe = "MC hammer"  
private String portretyFazoweUkładów = "Trolololo"  
protected Integer wynikEgzaminu = 2
```

## //Golang style

```
Name := "Piotr" //Exported  
id := "1kh2uss73x" //Unexported  
  
//function sayHello is unexported  
func sayHello(){  
    fmt.Println("Hello") //fmt.Println is exported  
}  
  
//Exported  
func ExporteMe(){  
  
}
```

**Types, functions, vars**

# Basic types + zero values

- bool	false
- string	""
- int int8 int16 int32 int64	0
- uint uint8 uint16 uint32 uint64 uintptr	0
- byte // alias for uint8	0
- rune // alias for int32	0
- float32 float64	0.00
- complex64 complex128	0+0i
- interface	nil
- pointer	nil

# Constants & vars

```
const (  
    Pi          = 3.14  
    Gandalf string = "you shall not pass!"  
)  
  
func main() {  
    fmt.Println(Pi)  
    fmt.Println(Gandalf)  
}
```

Run

```
func main() {  
    var xzibit, says string  
  
    xzibit = "Yodawg"  
    says = "I heard you like go"  
  
    so := "So I put some java code into your go code"  
  
    fmt.Println(xzibit,says,so)  
}
```

Run

# Functions

- Function recipe

```
func + [optional]name + ([optional]arguments) + ([optional] return type(s)) {  
    body  
}
```

- Functions receive arguments as copies
- Functions can return multiple values
- Functions can be assigned to variables
- Functions can return functions
- Functions can take variadic arguments
- Function can be defined as types

# Functions

```
package main

import "fmt"

func swap(x, y string) (string, string) {
    return y, x
}

func hello(name string) {
    fmt.Println("hello", name)
}

func main() {
    hello("Maciej")
    a, b := swap("hello", "world")
    fmt.Println(a, b)
}
```

Run



## Function part 2.

```
func namedValues(val int) (x, y int) {  
    x = val * 2  
    y = val * 3  
    return x,y  
}
```

```
func noRetArg(val int) (x, y int) {  
    x = val * 2  
    y = val * 3  
    return  
}
```

```
func main() {  
    fmt.Println(namedValues(2))  
    fmt.Println(noRetArg(2))  
}
```

Run

## Functions part 4 - Dude! really ?!

```
package main

import "fmt"

func main(){
    func (values ... int) {
        fmt.Println("I've been called with : ", values)
    }()
}
```

Run

# Structures

```
package main

import "fmt"

type Player struct {
    FirstName string
    LastName  string
}

func main(){
    goalKeeper := Player{
        FirstName: "Piotr",
        LastName: "Mścichowski", //-> comma required
    }

    proPlayer := Player{"Cristiano", "Ronaldo"}

    fmt.Println(goalKeeper, proPlayer)
}
```

Run

# Structures

- In go structures can have methods

```
package main

type Player struct {
    FirstName string
    LastName string
}

func (p *Player) Name() {
    println(p.FirstName)
}

func main(){
    p := new(Player) // Another way, new returns pointer,
    p.FirstName = "Piotr"
    p.LastName = "Mścichowski"

    p.Name()
}
```

Run

# Structures

- So basically structure is like class in Java/C# ?
- Does structure have constructor ?
- Overloading ?

# Pointers

- Wait, WAT?



# Pointers

- `var p *int` creates pointer to int
- `&` makes pointer to operand :

```
name := "Tom"  
ptr := &name
```

# Pointers

```
// method receiver
func (ptr *Person) modifyByPtr() {
    ptr.name = "Tom"
    ptr.lastName = "Hanks"
}

func (p Person) modifyByValue() {
    p.lastName = "Wick"
    p.name = "John"
}

func main() {
    me := Person{
        name:      "Piotr",
        lastName: "Mścichowski",
    }
    fmt.Println(me)
    ptr := &me
    ptr.modifyByPtr()
    fmt.Println(me)
    me.modifyByValue()
    fmt.Println(me)
}
```

Run



# Interface

- contract not an implementation

```
package main

import "fmt"

type Player interface {
    Hello(message string)
}

type Test struct {
}

func (t Test) Hello(message string) {
    fmt.Println("Hello", message)
}
```

- we do not need to 'tell' explicitly which interface we implement
- lot of interfaces in std lib like in package io
- duck typing

# Interface type - solution for missing generic types?

- In go you can use interfaces as type

```
type test struct {  
    val string  
}  
  
// interface{} has no methods  
func interfaceType(d interface{}) {  
    fmt.Println(d)  
    //work on copy of an interface type  
}  
  
func main() {  
    interfaceType(4)  
    interfaceType(test{"test value"})  
}
```

Run

- can a function return interface{} ?

## Errors and error handling

- Say goodbye to exceptions
- Error is an interface
- So it means that you can create your own errors easily

# Errors and error handling

```
type error interface {  
    Error() string  
}
```

```
func hello(input string) (string, error) {  
    if length := len(input); length > 0 {  
        return "Hello, " + input, nil  
    } else {  
        return "", errors.New("MISSING INPUT")  
    }  
}  
  
func main() {  
    val, err := hello("")  
  
    if err != nil {  
        println(err.Error())  
    } else {  
        println(val)  
    }  
}
```

Run

# Custom errors

```
type CustomError struct {
    Status  int
    Message string
    Type    string
}

func InternalServerError(message string) CustomError {
    return newError(500, "internal_server_error", message)
}

func (e CustomError) Error() string {
    return fmt.Sprintf("An error has occurred during processing! Details %s ", e.Message)
}

func newError(status int, errType string, message string) CustomError {
    ce := CustomError{status, errType, message}
    return ce
}
```

Run

## Custom errors

```
28 func process() error {  
29     //assume some processing is done here  
30     return InternalServerError("Error while calling upstream service")  
31 }  
32  
33 func main() {  
34     err := process()  
35     if err != nil {  
36         fmt.Printf(err.Error())  
37     }  
38 }
```

[Run](#)

# Arrays

```
1 package main
2 import "fmt"
3
4 func main() {
5     var words [2]string
6     words[0] = "hello"
7     words[1] = "world"
8     primes := [6]int{2, 3, 5, 7, 11, 13}
9     fmt.Println(primes)
10    //cannot use primes (type [6]int) as type [5]int in argument to sum
11    //sum(primes)
12    // cannot use primes (type [6]int) as type []int in argument to avg1
13    //avg(primes)
14 }
15
16 func sum(arr [5]int) int {
17     return 123
18 }
19 func avg(arr []int) float64 {
20     return 0.0
21 }
```

Run

# Slices

```
1 package main
2 import "fmt"
3
4 func main() {
5     var emptySlice []string
6     fmt.Printf("empty slice len: %d, cap: %d\n", len(emptySlice), cap(emptySlice))
7
8     letters := make([]string, 5, 10)
9     fmt.Printf("letters len: %d, cap: %d\n", len(letters), cap(letters))
10
11     // emptySlice[0] = "a" // panic: runtime error: index out of range
12     letters[0] = "a" // OK
13     // letters[10] = "b" // panic: runtime error: index out of range
14
15     letters = append(letters, "f")
16     fmt.Printf("letters after append: %v, len: %d, cap: %d\n", letters, len(letters), cap(letters))
17 }
```

[Run](#)



# Slices slicing

```
1 package main
2 import "fmt"
3
4 func main() {
5     src := []string{"a", "b", "c", "d", "e"}
6     fmt.Println("src:", src)
7
8     middle := src[1:4]
9     fmt.Println("middle:", middle)
10
11     first := src[:2]
12     fmt.Println("first:", first)
13
14     all := src[:]
15     fmt.Println("all:", all)
16
17     all[2] = "X"
18     fmt.Println("all:", all)
19     fmt.Println("src:", src)
20 }
```

Run

## Slices - Tricky part

```
1 package main
2 import "fmt"
3
4 func main() {
5     // insert in the middle
6     src := []string{"a", "b", "d", "e"}
7     fmt.Println(src)
8     src = append(src[:2], append([]string{"c"}, src[2:]...)...)
9     fmt.Println("src:", src)
10
11     //delete
12     copy(src[2:], src[3:])
13     fmt.Println("src after copy:", src)
14     src[len(src) - 1] = ""
15     src = src[:len(src) - 1]
16     fmt.Println("src:", src)
17 }
```

Run

# Maps

```
4 func main() {  
5     worldChampions := map[int]string{  
6         1994:"Brasil",  
7         1998:"France",  
8         2002:"Brasil",  
9         2006:"Italy",  
10        2010:"Spain",  
11    }  
12    worldChampions[2014] = "Germany"  
13    fmt.Println(worldChampions[1994])  
14    _, ex := worldChampions[1997]  
15    fmt.Println(ex)  
16    delete(worldChampions, 2006)  
17    fmt.Println(worldChampions)  
18 }
```

Run

# Loops

```
3 func main() {  
4     for i := 0; i < 10; i++ {  
5         //TODO  
6     }  
7     flag := true  
8     for {  
9         if flag {  
10             break  
11         }  
12     }  
13 }
```

Run

# If

```
6 func main() {  
7     str := "12"  
8     if n, err := strconv.Atoi(str); err != nil {  
9         fmt.Println("Got error when parsing ", str)  
10    } else {  
11        fmt.Printf("Number is %d\n", n)  
12    }  
13    // n = 14 - Unresolved reference  
14 }
```

Run

# Defer

```
1 package main
2 import "os"
3 import "fmt"
4
5 func main() {
6     f, err := os.Open("abc.txt")
7     if err != nil {
8         fmt.Println("Got error", err)
9         return
10    }
11    defer f.Close()
12    f.WriteString("sth")
13
14    i := 10
15    defer cleanup(i)
16    i = 20
17    defer cleanup(i)
18 }
19
20 func cleanup(i int) {
21     fmt.Printf("Cleanup %d\n", i)
22 }
```

Run

# Switch

```
6 func main() {  
7     var todayMeme string  
8     switch today := time.Now().Weekday(); today {  
9     case time.Monday:  
10        fallthrough  
11        case time.Tuesday:  
12            todayMeme = "grumpyCat"  
13        case time.Friday:  
14            todayMeme = "weekend"  
15        default:  
16            todayMeme = "randomMeme"  
17    }  
18    fmt.Println(todayMeme)  
19 }
```

Run

# Goroutines

\* Goroutine is not OS thread!

**Table 2–1** Thread creation costs

Item	Approximate cost
Kernel data structures	Approximately 1 KB
Stack space	512 KB (secondary threads) 8 MB (OS X main thread) 1 MB (iOS main thread)
Creation time	Approximately 90 microseconds

From

<https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/Multithreading/CreatingThreads/CreatingThre>



# Goroutines

```
1 package main
2 import "fmt"
3 import "time"
4
5 func main() {
6     go longComputation(1000)
7     fmt.Println("Will be printed immediately")
8 }
9
10 func longComputation(x int) {
11     time.Sleep(time.Hour)
12 }
```

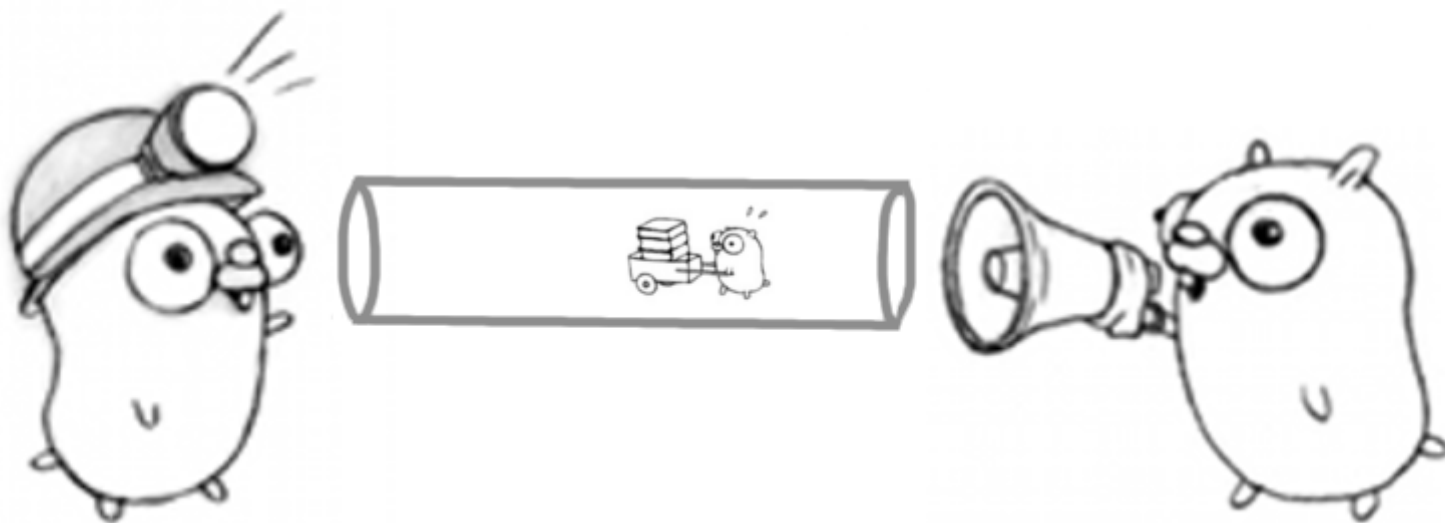
Run

# Goroutines

```
1 package main
2 import "sync"
3 import "time"
4 import "fmt"
5 import "math/rand"
6
7 func main() {
8     wg := sync.WaitGroup{}
9     bef := time.Now()
10    const len = 1000000 // million
11    wg.Add(len)
12    var out [len]int32
13    for i := 0; i < len; i++ {
14        go func(idx int) {
15            out[idx] = rand.Int31n(1000)
16            wg.Done()
17        }(i)
18    }
19    wg.Wait()
20    fmt.Println(time.Now().Sub(bef))
21 }
```

Run

Do not communicate by sharing memory; instead, share memory by communicating.



# Channels

```
1 package main
2
3 import (
4     "fmt"
5     // "time"
6 )
7
8 func main() {
9     msgChan := make(chan string)
10
11     go func() {
12         msgChan <- "message"
13     }()
14     go func() {
15         val := <-msgChan
16         fmt.Println(val)
17     }()
18     // time.Sleep(time.Second)
19     // msgChan <- "message 2"
20
21 }
```

Run

# Buffered channels

```
1 package main
2
3 func main() {
4     bufferedChan := make(chan string, 2)
5     bufferedChan <- "msg1"
6     bufferedChan <- "msg2"
7     //bufferedChan <- "msg3" // deadlock
8 }
```

Run

## Example: Producer Consumer

```
18 func producer(name string, msgChan chan string, wg *sync.WaitGroup) {
19     for i := 0; i < 3; i++ {
20         msgChan <- fmt.Sprintf("[%s] Message:%d", name, i+1)
21         time.Sleep(time.Second)
22     }
23     fmt.Printf("[%s] Done\n", name)
24     wg.Done()
25 }
```

Run

```
27 func consumer(name string, msgChan chan string) {
28     got := 0
29     for {
30         msg := <-msgChan
31         got++
32         fmt.Printf("[%s] Received message nr %d: %s\n", name, got, msg)
33         if msg == DONE_MSG {
34             return
35         }
36     }
37 }
```

Run

## Example: Producer Consumer cont.

```
8 func main() {  
9     msgChan := make(chan string)  
10    wg := sync.WaitGroup{}  
11    go producer("Producer",msgChan,&wg)  
12    wg.Add(1)  
13    go consumer(fmt.Sprintf("Consumer"), msgChan)  
14    wg.Wait()  
15    msgChan <- DONE_MSG  
16 }
```

Run



## Example: Timeouts

```
package time

// After waits for the duration to elapse and then sends the current time
// on the returned channel.
func After(d Duration) <-chan Time {
    return NewTimer(d).C
}
```

## Example: "I don't care" Producer

```
1 package main
2 import "sync"
3 import "fmt"
4 import "time"
5 func main() {
6     msgChan := make(chan string)
7     wg := sync.WaitGroup{}
8     wg.Add(2)
9     go func () {
10         fmt.Println("Got: ",<-msgChan)
11         wg.Done()
12     }()
13     go iDontCareProducer("Producer",msgChan,&wg)
14     wg.Wait()
15 }
16 func iDontCareProducer(name string, msgChan chan string, wg *sync.WaitGroup) {
17     for i := 0; i < 5; i++ {
18         select {
19             case msgChan <- fmt.Sprintf("[%s] Not so important message:%d", name, i+1):
20                 case <-time.After(time.Second): fmt.Println("Timeout")
21         }
22     }
23     fmt.Printf("[%s] Done\n", name)
24     wg.Done()
25 }
```



# Go tools

- Govendor

Govendor is similar a little bit to js npm

- Gofmt

Builtin tool used for formatting go code

- Go vet

Vet examines Go source code and reports suspicious constructs.

- Go present

Tool for making presentation (like this one )

- Lots of them (<https://godoc.org/golang.org/x/tools>)

# Hackin Gliwice



- More info : <https://www.facebook.com/Hackin-Gliwice-1852681074976881>

**Thank you for attention. Do you have questions?**

## Links:

- <https://tour.golang.org/>
- <https://golang.org/doc/faq>
- <https://talks.golang.org/2012/splash.article> (Go at Google: Language Design in the Service of Software Engineering)
- <http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang>
- <https://eng.uber.com/go-geofence> (How We Built Uber Engineering's Highest Query per Second Service Using Go)

Thank you



