A hitchhicker's guide to Go

What is Go and why should you care?

25 January 2016

George Paraskevopoulos
Software Engineer, Intracom Telecom

Pavlos Antoniou
Software Engineer, Intracom Telecom

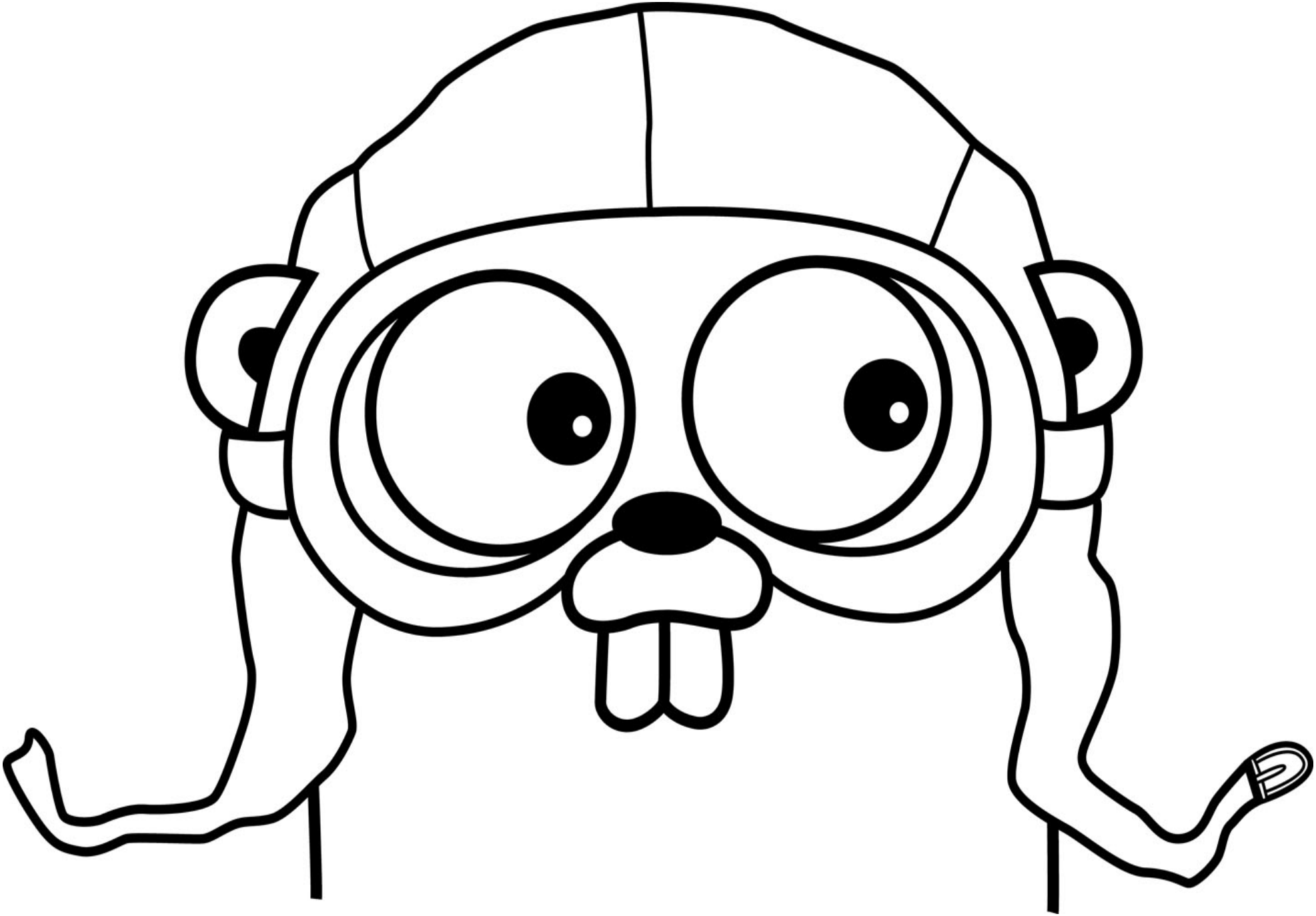
Nikos Anastopoulos
Technical Expert / Product Owner, Intracom Telecom

whoami

- We are the ODLP[2] team
- We work on the performance optimization of SDN/NFV applications
- In the past we have studied and doubled the performance of the OpenDaylight SDN controller
- working on solution for automated tuning of NFV deployments on private clouds for performace/energy

Starting Go

play.golang.org/ (https://play.golang.org/)



Go is

golang.org (https://golang.org): "Go is an open source programming language that makes it easy to build simple, reliable, and efficient software."

- open source
- concurrent
- low learning curve
- garbage collected
- simple (total of 25 keywords)
- for software engineers not for programming language designers
- opinionated
- developed by Google

Go is

- statically typed
- compiled
- object oriented
- memory safe (no pointer arithmetic)
- type safe (explicit type conversion)

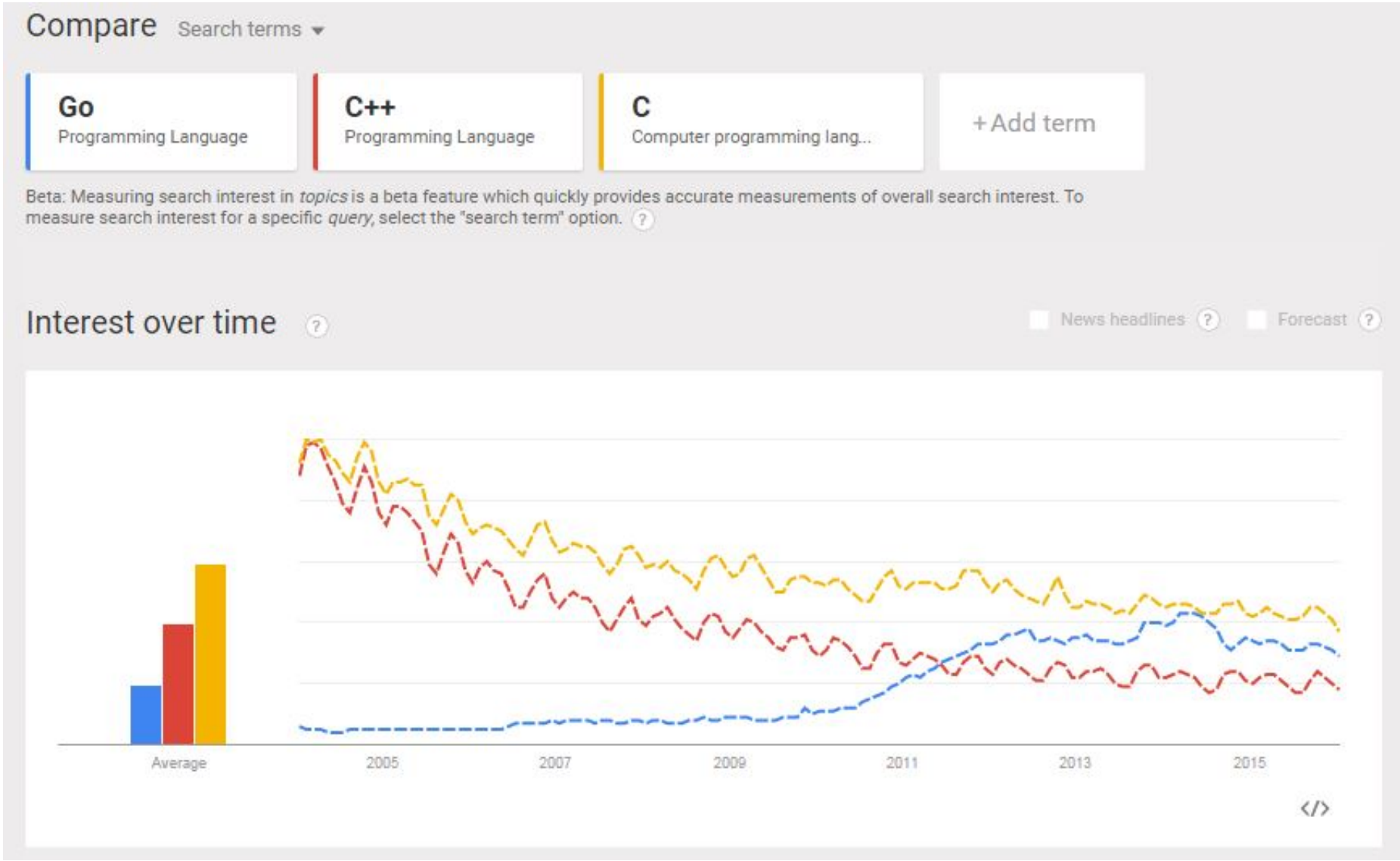
Who uses Go?

- Google (obviously)
- Github
- Mozilla
- Dropbox
- Heroku
- Docker
- CoreOS
- Canonical
- New York Times
- SoundCloud
- CloudFlare

Go in open source projects

- Docker
- Kubernetes
- Flynn
- InfluxDB
- etcd / Fleet
- Drone CI
- CorkroachDB

Trend



Compare Go, C and C++ in Google Trends

Let's Start

Technical disclaimer: *The following content is rated* **T** *for technical*

- The main concepts are made as simple as possible
- Can't avoid technical topics

Hello World

```
package main

import "fmt"

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

Run

Killer Features: Tooling

Go Busybox

```
Go is a tool for managing Go source code.

Usage:

      go command [arguments]

The commands are:

        build       compile packages and dependencies
        clean       remove object files
        doc         show documentation for package or symbol
        env         print Go environment information
        fix         run go tool fix on packages
        fmt         run gofmt on package sources
        generate    generate Go files by processing source
        get         download and install packages and dependencies
        install     compile and install packages and dependencies
        list        list packages
        run         compile and run Go program
        test        test packages
        tool        run specified go tool
        version     print Go version
        vet         run go tool vet on packages
```

gofmt

- Reported as the single most important go tool
- Enforces coding standards and uniformity
- With great uniformity comes great readability
- gofmt-ed packages are enabled for semantic tools processing
- >80% of open source Go code is gofmt-ed

godoc

godoc.org (godoc.org)

- Great resource to search for packages
- Indexed and searchable packages from Github, BitBucket, golang.org etc.
- Documentation is generated from the doc-style comments
- Documentation is coupled with the source code
- What you see is what you get

Data Race detector

Go runtime module is equipped with a race detection capabilities.

The runtime module tracks

- Memory accesses
- function calls
- goroutine creation/exit
- synchronization

Then it creates a precedence model to find the race conditions.

- No false positives possible
- Can have false negatives

Data Race Detector: Usage

```
$ go test -race mypkg    // to test the package
$ go run -race mysrc.go  // to run the source file
$ go build -race mycmd   // to build the command
$ go install -race mypkg // to install the package
```

Data Race Detector: Example output

```
WARNING: DATA RACE
Read by goroutine 7:
  main.incrementCounter()
      ..../.gopath/presentations/gotour-intracom-telecom/code/concurrency/race.go:30 +0x4c

Previous write by goroutine 10:
  main.incrementCounter()
      ..../.gopath/presentations/gotour-intracom-telecom/code/concurrency/race.go:30 +0x68

Goroutine 7 (running) created at:
  main.main()
      ..../.gopath/presentations/gotour-intracom-telecom/code/concurrency/race.go:15 +0xca

Goroutine 10 (finished) created at:
  main.main()
      ..../.gopath/presentations/gotour-intracom-telecom/code/concurrency/race.go:15 +0xca
==================
```

go test

- Use testing package in std lib
- Tests in regular go
- Can run recursively
- Tests live in **_test.go** files

Example output:

```
$ go test -v
=== RUN TestReverse
--- PASS: TestReverse (0.00s)
=== RUN: ExampleReverse
--- PASS: ExampleReverse (0.00s)
PASS
ok      github.com/golang/example/stringutil    0.009s
```

golint and govet

- Check for common mistakes
- Suggestions and warnings
- Check for idiomaticity
- Integration with many editors
- Opinionated

Other tools

- Deadlock detection: Can find when the entire application deadlocks
- gdb compatibility
- pprof: runtime profiling and visualisation
- goimports: sanitize imported packages
- many many more...

Killer Features: Concurrency

Running applications in Python, JS, Ruby

Concurrency in Go

- This is the main feature of the language
- Go was created for building scalable applications that run on single multicore machines or even on multiple machines

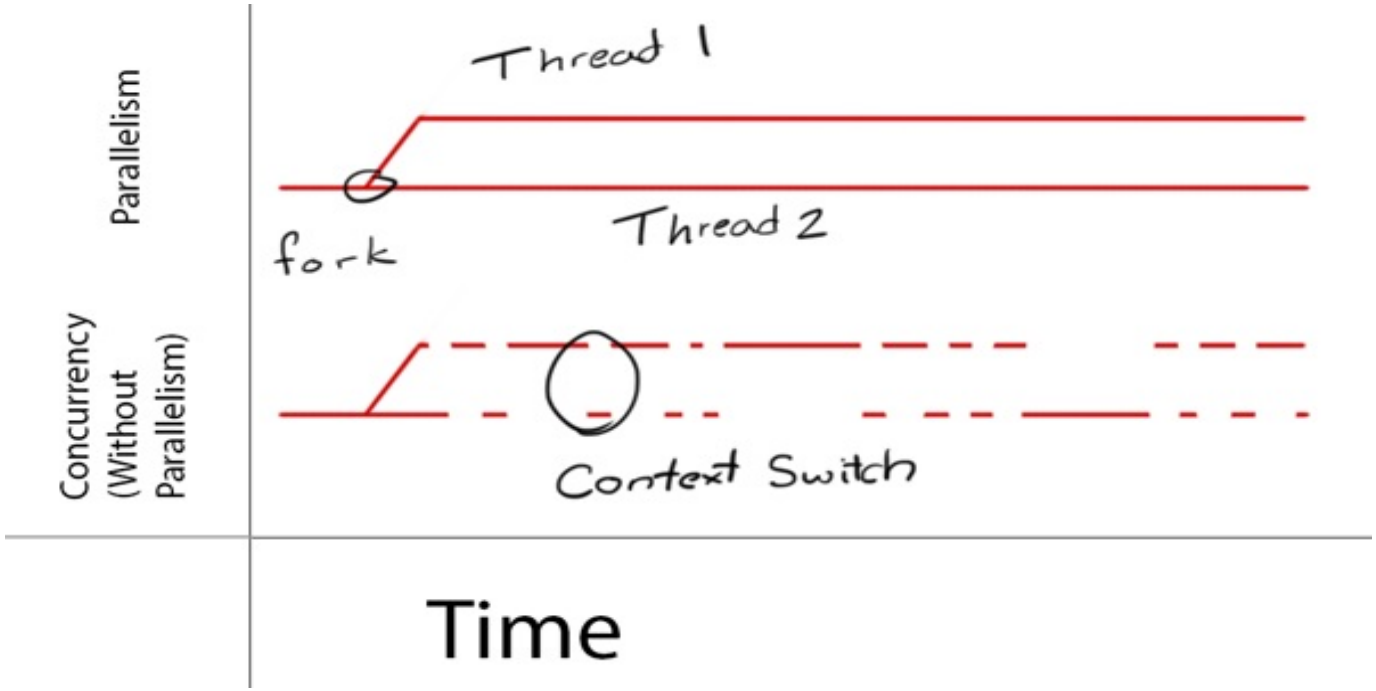
The concurrency mechanism is part of the core language, not facilitated by a library. It is based on three primitives

- **Goroutines**: You can think of goroutines as extremely lightweight threads (order of 4Kb). They have their own dynamic stack and get multiplexed in the system threads.
- **Channels**: Channels are 2-way typed pipes and are the main communication and synchronization mechanism in Go
- **Select** statement: Like a switch-case statement to wait for and handle input from multiple channels

Concurrency and Parallelism

Disclaimer: *Concurrency != Parallelism*

- Concurrent programs may or may not run in parallel
- Concurrency is a way to structure software that deals with multiple things at once
- Parallelism is a way to execute multiple things at once



Concurrency vs Parallelism

Communicating Sequential Processes

- The concurrency model in Go is based on the concept of communicating sequential processes
- The concept was first illustrated in a rigorous algebraic formulation by Tony Hoare (the same guy who invented quicksort)
- It sums up to using synchronous communication mechanisms (a.k.a. pipes) for information sharing instead of sharing memory

```
PHIL = *[... during ith lifetime ... →
        THINK;
        room!enter( );
        fork(i)!pickup( ); fork((i + 1) mod 5)!pickup( );
        EAT;
        fork(i)!putdown( ); fork((i + 1) mod 5)!putdown( );
        room!exit( )
        ]
```

Dining philosophers with CSP

Don't panic, no math today

Why CSP?

Coding Time

Example: Synchronous and Asynchronous I/O

We want to send requests repeatedly to a server

```
numRequests := 42

for i := 0; i < numRequests; i++ {
    resp, _ := http.Get("http://127.0.0.1:4444")
    respBody, _ := ioutil.ReadAll(resp.Body)
    fmt.Println("Response came after", string(respBody), "seconds")
}
```

Example: Synchronous and Asynchronous I/O

Specifically the following REST server

```
        router := mux.NewRouter().StrictSlash(true)
        router.HandleFunc("/", serverSleep)
        router.HandleFunc("/stop", serverStop)
        http.ListenAndServe(":4444", router)
```

Which sends back delayed responses

```
func serverSleep(w http.ResponseWriter, r *http.Request) {
    sleepTime := rand.Int() % 5
    time.Sleep(time.Duration(sleepTime) * time.Second)
    fmt.Fprintf(w, "%d", sleepTime)
}
```

Example: Synchronous I/O

```
func main() {
    go func() {
        router := mux.NewRouter().StrictSlash(true)
        router.HandleFunc("/", serverSleep)
        router.HandleFunc("/stop", serverStop)
        http.ListenAndServe(":4444", router)
    }()
    fmt.Println("Waiting for server to start...") // Don't do this
    time.Sleep(time.Duration(5) * time.Second)

    numRequests := 42

    for i := 0; i < numRequests; i++ {
        resp, _ := http.Get("http://127.0.0.1:4444")
        respBody, _ := ioutil.ReadAll(resp.Body)
        fmt.Println("Response came after", string(respBody), "seconds")
    }

    http.Get("http://127.0.0.1:4444/stop")
}
```

Run

Example: Asynchronous I/O (SleepSort)

```
func main() {
     go func() {
        router := mux.NewRouter().StrictSlash(true)
        router.HandleFunc("/", serverSleep)
        router.HandleFunc("/stop", serverStop)
        http.ListenAndServe(":4444", router)
    }()
    fmt.Println("Waiting for server to start...") // Don't do this
    time.Sleep(time.Duration(5) * time.Second)

    numRequests := 42
    done := make(chan bool)

    for i := 0; i < numRequests; i++ {
        go func() {
            resp, _ := http.Get("http://127.0.0.1:4444")
            respBody, _ := ioutil.ReadAll(resp.Body)
            fmt.Println("Response came after", string(respBody), "seconds")
            done <- true
        }()
    }
    for i := 0; i < numRequests; i++ { <-done }
    http.Get("http://127.0.0.1:4444/stop")
}
```

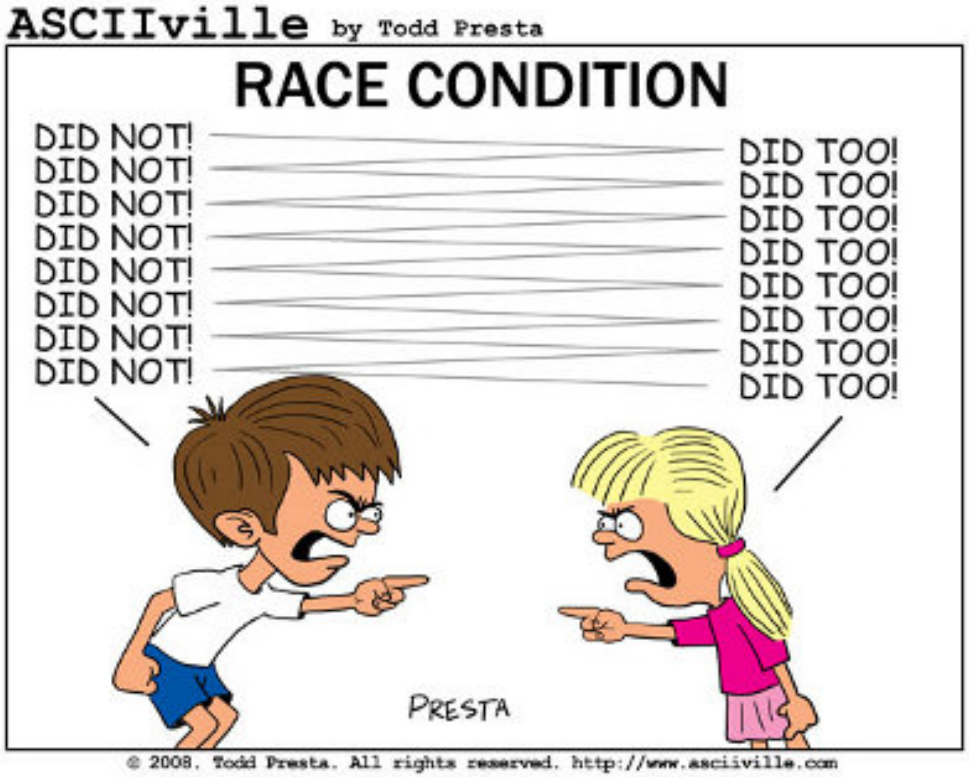
Run

Example: Timeout

```
func main() {
    service1 := make(chan string, 1)
    go func() {
        time.Sleep(time.Second * 1)
        service1 <- "Service 1 returned a result after 1 second."
    }()
    service2 := make(chan string, 1)
    go func() {
        time.Sleep(time.Second * 42)
        service2 <- "Service 2 returned a result after 42 seconds."
    }()

    for {
        select {
        case res := <-service1:
            fmt.Println(res)
        case res := <-service2:
            fmt.Println(res)
        case <-time.After(time.Second * 2):
            fmt.Println("Timeout waiting for services to complete after 2 seconds.")
            return
        }
    }
}
```

Run

Example: Race Condition



Example: Race Condition

```
var counter int

func main() {
    done := make(chan bool)
    numGoroutines := 42
    counterInc := 1000

    for i := 0; i < numGoroutines; i++ {
        go incrementCounter(counterInc, done)
    }

    for i := 0; i < numGoroutines; i++ { <-done }

    fmt.Println("Counter =", counter)
}

func incrementCounter(N int, done chan bool) {
    for i := 0; i < N; i++ {
        counter++
    }
    done <- true
}
```

Run

Example: Race Condition Detection

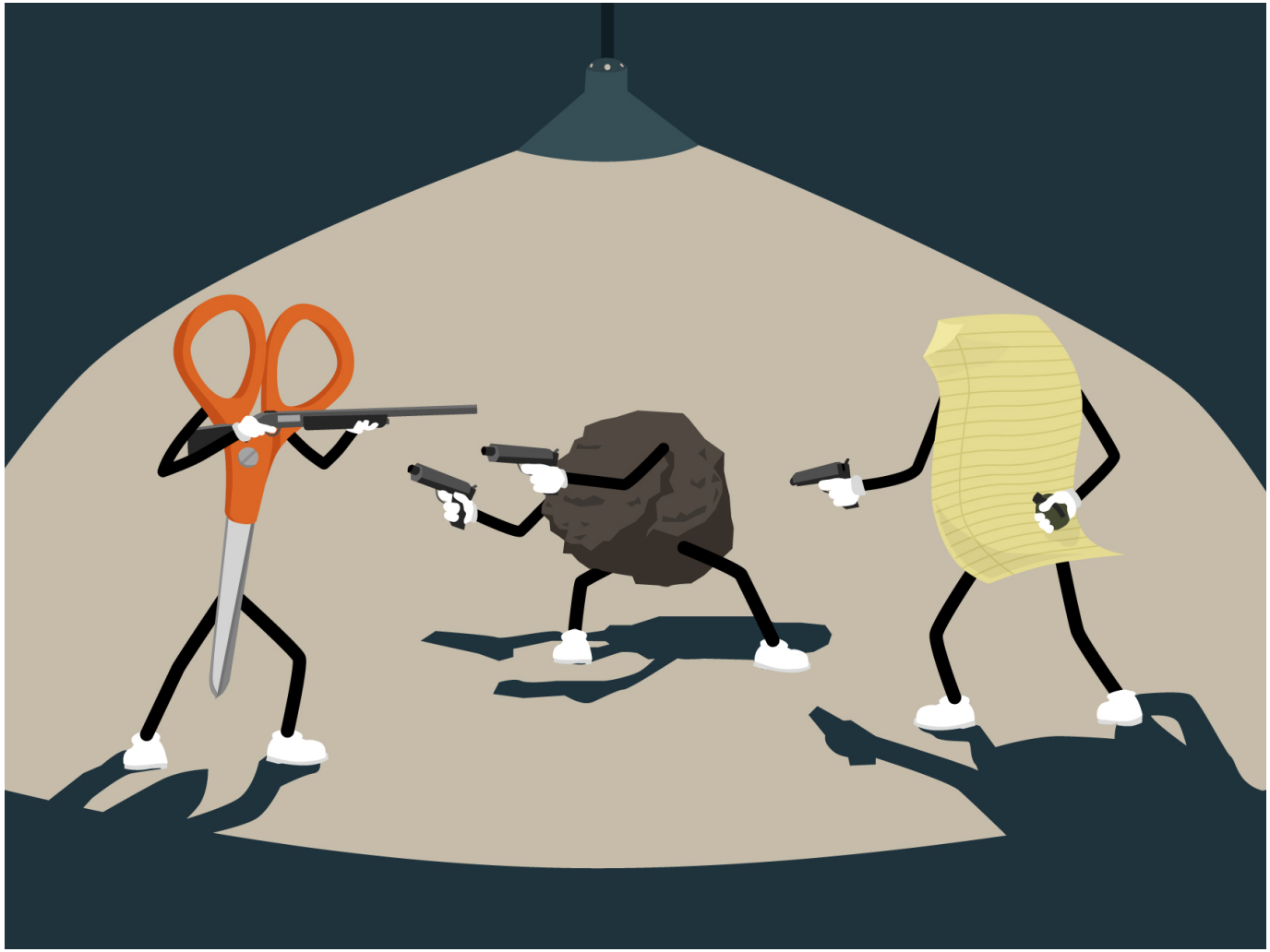
```
==================
WARNING: DATA RACE
Read by goroutine 7:
  main.incrementCounter()
      ..../.gopath/presentations/gotour-intracom-telecom/code/concurrency/race.go:24 +0x4c

Previous write by goroutine 9:
  main.incrementCounter()
      ..../.gopath/presentations/gotour-intracom-telecom/code/concurrency/race.go:24 +0x68

Goroutine 7 (running) created at:
  main.main()
      ..../.gopath/presentations/gotour-intracom-telecom/code/concurrency/race.go:14 +0xa8

Goroutine 9 (running) created at:
  main.main()
      ..../.gopath/presentations/gotour-intracom-telecom/code/concurrency/race.go:14 +0xa8
==================
Counter = 30044
Found 1 data race(s)
exit status 66
```

Example: Race Condition Resolution

```
func main() {
    done := make(chan bool)
    counterChan := make(chan int, 1)
    numGoroutines := 42
    counterInc := 1000

    for i := 0; i < numGoroutines; i++ {
        go incrementCounter(counterInc, done, counterChan)
    }
    counterChan <- 0

    for i := 0; i < numGoroutines; i++ { <-done }

    fmt.Println("Counter =", <-counterChan)
}

func incrementCounter(N int, done chan bool, cntChan chan int) {
    for i := 0; i < N; i++ {
        c := <-cntChan
        c++
        cntChan <- c
    }
    done <- true
}
```

Run

Example: Deadlock



Example: Deadlock

```
func main() {
    done := make(chan bool)
    counterChan := make(chan int, 1)
    numGoroutines := 42
    counterInc := 1000

    for i := 0; i < numGoroutines; i++ {
        go incrementCounter(counterInc, done, counterChan)
    }
    // counterChan <- 0

    for i := 0; i < numGoroutines; i++ { <-done }

    fmt.Println("Counter =", <-counterChan)
}

func incrementCounter(N int, done chan bool, cntChan chan int) {
    for i := 0; i < N; i++ {
        c := <-cntChan
        c++
        cntChan <- c
    }
    done <- true
}
```

Run

Example: A More Subtle Deadlock

```
func main() {
    done := make(chan bool)
    counterChan := make(chan int)
    numGoroutines := 42
    counterInc := 1000

    for i := 0; i < numGoroutines; i++ {
        go incrementCounter(counterInc, done, counterChan)
    }
    counterChan <- 0

    for i := 0; i < numGoroutines; i++ { <-done }

    fmt.Println("Counter =", <-counterChan)
}

func incrementCounter(N int, done chan bool, cntChan chan int) {
    for i := 0; i < N; i++ {
        c := <-cntChan
        c++
        cntChan <- c
    }
    done <- true
}
```

Run

Example: Fine Grained Concurrency / More Message Passing

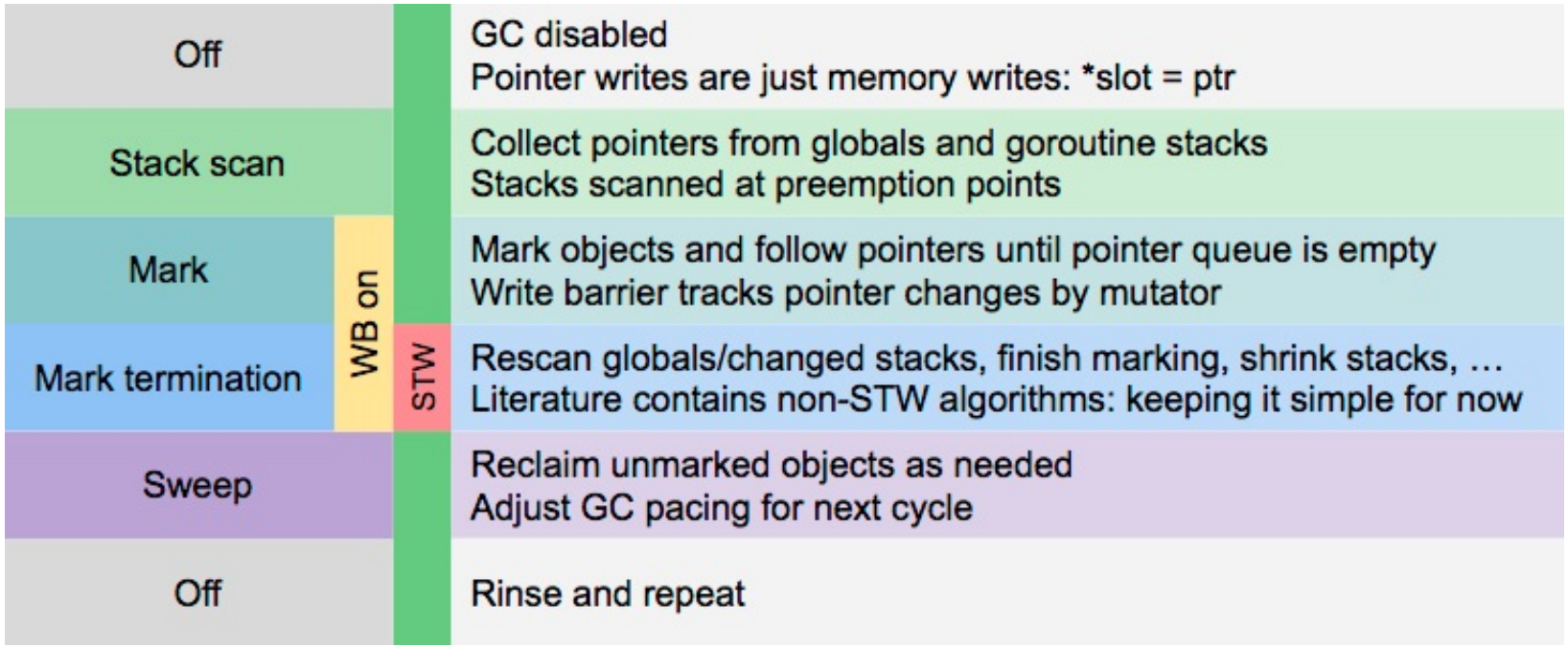
- Natural Language Detection

```
for i := range phrases {
    go func(idx int, phrase string) {
        results := make(chan PairSI)
        for _, lang := range languages {
            go func(lang string, results *chan PairSI) {
                eval := phrase + corpus[lang]
                eval_sz := compressedSize(eval) - corpus_sz[lang]
                *results <- PairSI{S: lang, I: eval_sz}
            }(lang, &results)
        }

        class := PairSI{S: "", I: 1<<32 - 1}
        for i := 0; i < len(languages); i++ {
            curr_class := <- results
            if curr_class.I < class.I  {
                class.S = curr_class.S
                class.I = curr_class.I
            }
        }
        class.I = idx
        classification <- class
      }(i, phrases[i])
}
```

Killer Features: Garbage Collection

Go Garbage Collector

- Mark and sweep algorithm
- (+) Low impact: Just marks unreferenced objects
- (-) Pauses: Application execution stops to reclaim marked objects in the heap



Garbage Collector runtime states

Garbage Collector Performance

- The performance in older versions of Go was not great
- Significant pauses were observed at runtime

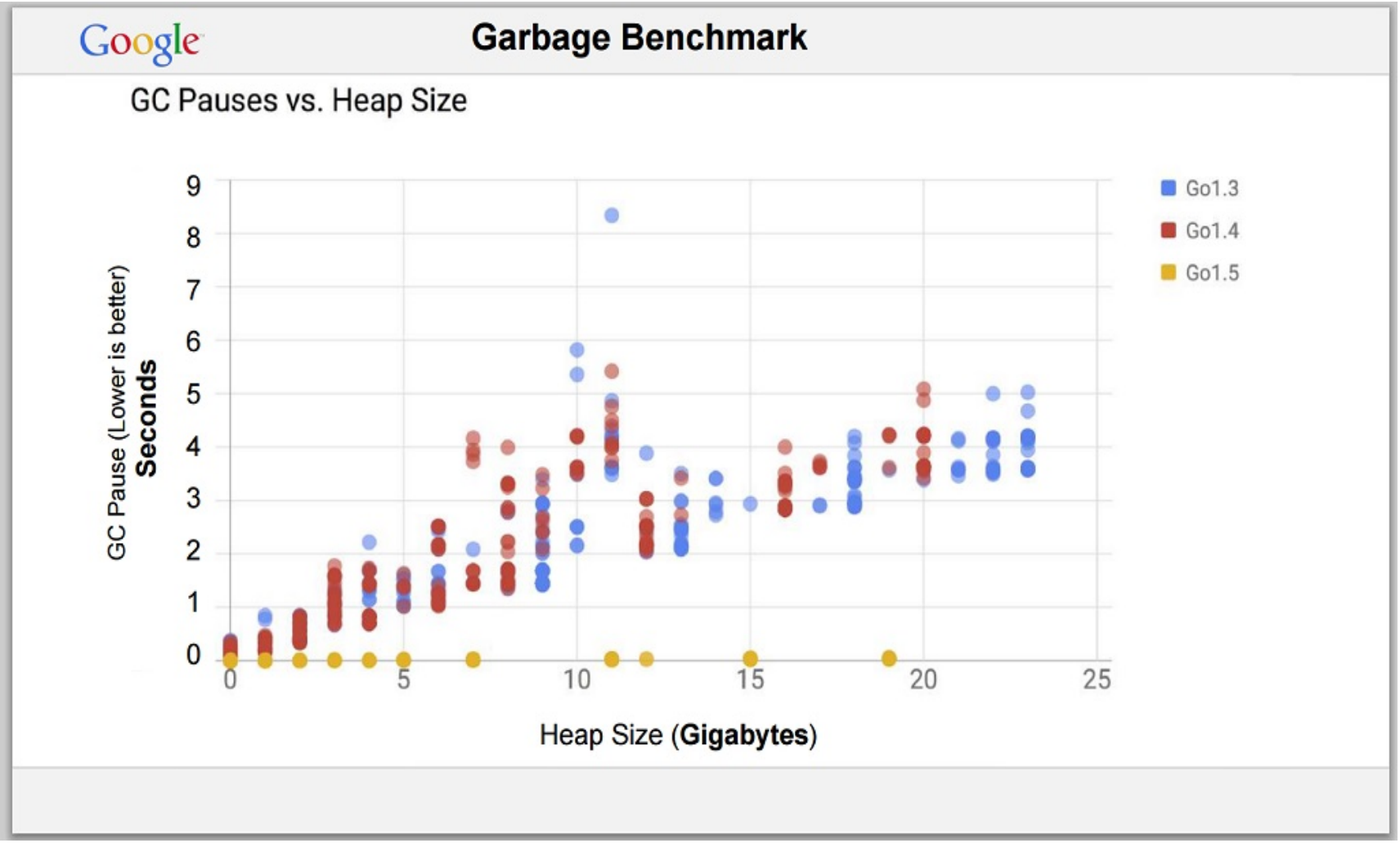
As of Go 1.5:

- Concurrent implementation
- GC latency limited to less than *10 ms*
- Assurance: Application code runs for at least *40 ms* out of every *50 ms*

Conclusion

- If the performance hit wasn't a worthwhile tradeoff in Go 1.4 maybe it is now

Garbage Collector Performance



Garbage Collector cross-version bencharks

Killer Features: Fast Compilation

Fast Compilation

Go programs compile much faster (order of 50x) than C or C++ programs of equivalent size.

- Low compilation time is an explicit design target
- Language simplicity
- Compact grammar
- No symbol table
- **Dependency Management** is the key

Dependency Management

- Unused dependencies trigger a *compile time error*
- No circular imports
- Each file is opened only once
- Exported data in the object file

Example: Transitive dependency compilation

- package A imports package B
- package B imports package C
- package A does *not* import package C

Dependent packages must be built before the packages that depend on them

- C is compiled first
- B is compiled second
- A is compiled last

Example: Transitive dependency linking

- The compiler reads the object file for B to compile A, not its source code
- The object file contains all the necessary type information to compile A
- The generated object file of B includes type information for all dependencies of B that affect the public interface of B

Killer Features: Easy Deployment

Application Deployment in Go

Deploying applications in Go is made easy due to the following

- Static linking of the output binaries
- No external dependencies required in the target (except libc on Linux)
- The necessary runtime components are compiled into the binary
- Can cross compile to different platforms (Linux, Mac OSX, Windows)
- Can cross compile to different architectures (x86, x64, ARMv5, ARMv6, ARMv7, ARMv8)

Showcase: Jscheduler

Jscheduler

github.com/intracom-telecom-sdn/jscheduler-go (https://github.com/intracom-telecom-sdn/jscheduler-go)

A testing tool to change the CPU affinity and the priority of Java threads at runtime

Features

- Live monitoring of JVM processes using JStack
- Dynamic thread dump parsing
- Dynamic name based thread matching
- Dynamic CPU affinity enforcement
- Dynamic thread priority enforcement
- Low execution footprint

The idea is simple: Parse the thread dump, get the native thread ids and enforce the policies on selected threads

Monitoring: Get JStack Thread Dump

```
func GetJstackThreadDump(java_home string, pid string) (string, error) {
    user := os.Getenv("SUDO_USER")
    if user == "" {
        user = os.Getenv("USER")
    }
    cmd := fmt.Sprintf("sudo -u %s %s/bin/%s -l %s", user, java_home, "jstack", pid)
    out, err := exec.Command("/bin/sh", "-c", cmd).Output()
    return string(out), err
}
```

- Interface with system commands
- Interface with system environment
- Multiple returns (get used to it)

Thread Dump Parsing: Regex Matching

```
const THREAD_DESCRIPTOR1 string = `^"(?P<name>[^"]+)".+prio=(?P<prio>[0-9]+)\s+os_prio=(?P<os_pr
io>[0-9]+)\s+tid=(?P<tid>0x[0-9a-f]+)\s+nid=(?P<nid>0x[0-9a-f]+).+`
```

```
    r1 := regexp.MustCompile(THREAD_DESCRIPTOR1)
```

```
func decomposeTreadDumpLineRe(threadDumpLine string, r *regexp.Regexp) (groups map[string]string
, err error) {
    matches := r.FindStringSubmatch(threadDumpLine)
    names := r.SubexpNames()

    groups = make(map[string]string)

    for i, name := range names {
        groups[name] = matches[i]
    }
    return
}
```

- Standard regex syntax
- Separate matched elements into groups

Thread Management: Some Type Declarations

```
type Thread struct {
    Name      string
    Tid       int
    Prio      int
    Cpus      CpuPool
    HasPolicy bool
}

func NewThread(name string, tid int) Thread {
    return Thread{
        Name:      name,
        Tid:       tid,
        Prio:      0,
        Cpus:      NewCpuPool(runtime.NumCPU()),
        HasPolicy: false,
    }
}
func (t *Thread) SetPolicy(policy ThreadPolicy) {
    t.Prio = policy.Prio
    t.Cpus = policy.Cpus
    t.HasPolicy = true
}
```

```
type ThreadList []Thread
```

Thread Management: Excluding Previous Threads

```
func ExcludeThreads(threads *ThreadList, excluded map[string]struct{}) *ThreadList {
    threadsRemain := NewThreadList()
    for _, thread := range *threads {
        if _, ignore := excluded[thread.Name]; !ignore {
            threadsRemain = append(threadsRemain, thread)
        }
    }
    return &threadsRemain
}
```

- Maps (and how to use them as sets)
- Pass by reference
- Idiomatic syntax
- Notice how we wrap the *[]Thread* type in *ThreadList* but we can still use *range* and *append* on it

Affinity Enforcement: Some low level code

```
func SetAffinity(pid int, cpus []int) error {
    var mask [1024 / 64]uintptr
    if pid <= 0 {
        pidget, _, _ := syscall.RawSyscall(unix.SYS_GETPID, 0, 0, 0)
        pid = int(pidget)
    }
    for _, cpuIdx := range cpus {
        cpuIndex := uint(cpuIdx)
        mask[cpuIndex/64] |= 1 << (cpuIndex % 64)
    }
    syscall.RawSyscall(unix.SYS_SCHED_SETAFFINITY,
        uintptr(pid),
        uintptr(len(mask)*8),
        uintptr(unsafe.Pointer(&mask[0])))
    return nil
}
```

- Calling raw system calls
- Some bitmask magic
- Unsafe

More Low Level Code: Signal Handling

```
    c := make(chan os.Signal, 1)
    signal.Notify(c, os.Interrupt)
    signal.Notify(c, syscall.SIGTERM)
    go func() {
        <-c
        printThreadCount(threadCount)
        os.Exit(1)
    }()
```

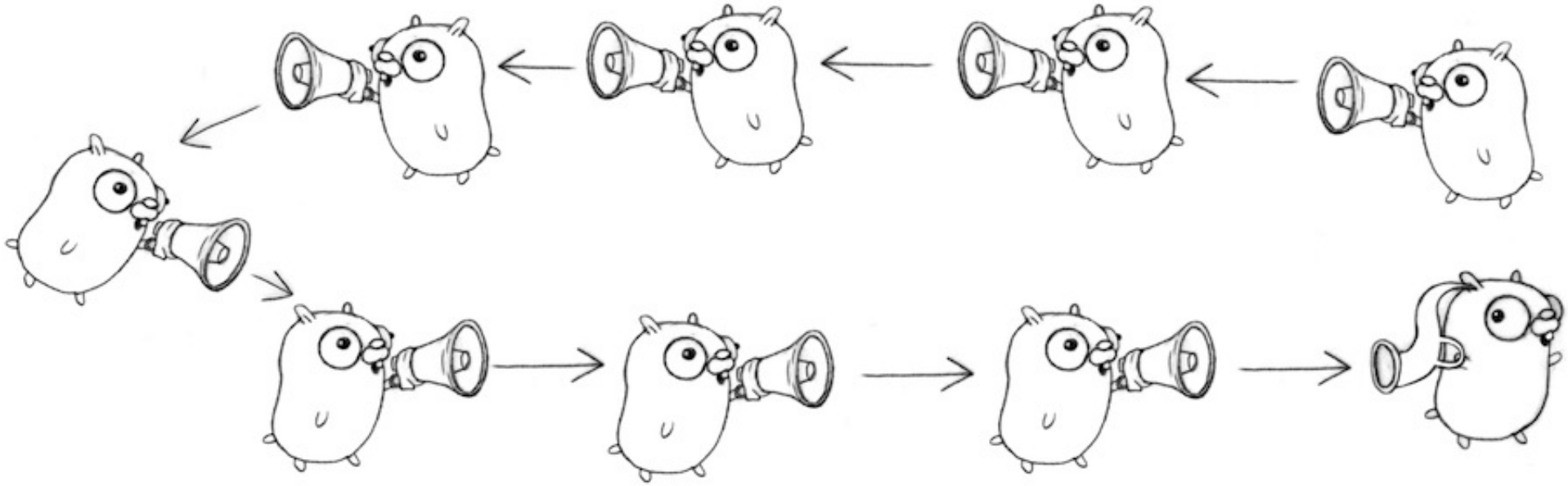
- Run in the background
- Notification through channel

Program Loop: Sequential Processes

```
    for {
        threadDump, _ := jscheduler.GetJstackThreadDump(os.Getenv("JAVA_HOME"), pid)

        allThreads, _ := jscheduler.ParseThreadDump(threadDump)

        for _, t := range *allThreads {
            threadCount[t.Name]++
        }

        threads := jscheduler.ExcludeThreads(allThreads, modifiedThreads)

        jscheduler.AdjustThreadPolicies(threads, policies.Get())

        jscheduler.RescheduleThreadGroup(threads)

        for _, t := range *threads {
            if t.HasPolicy {
                modifiedThreads[t.Name] = struct{}{}
            }
        }

        time.Sleep(time.Duration(interval) * time.Millisecond)
    }
```

Opportunity for Improvement: Communicating Sequential Processes

Use concurrency to improve the execution of the previous code



Thank you

George Paraskevopoulos
Software Engineer, Intracom Telecom
geopar@intracom-telecom.com (mailto:geopar@intracom-telecom.com)

Pavlos Antoniou
Software Engineer, Intracom Telecom
pant@intracom-telecom.com (mailto:pant@intracom-telecom.com)

Nikos Anastopoulos
Technical Expert / Product Owner, Intracom Telecom
nanast@intracom-telecom.com (mailto:nanast@intracom-telecom.com)