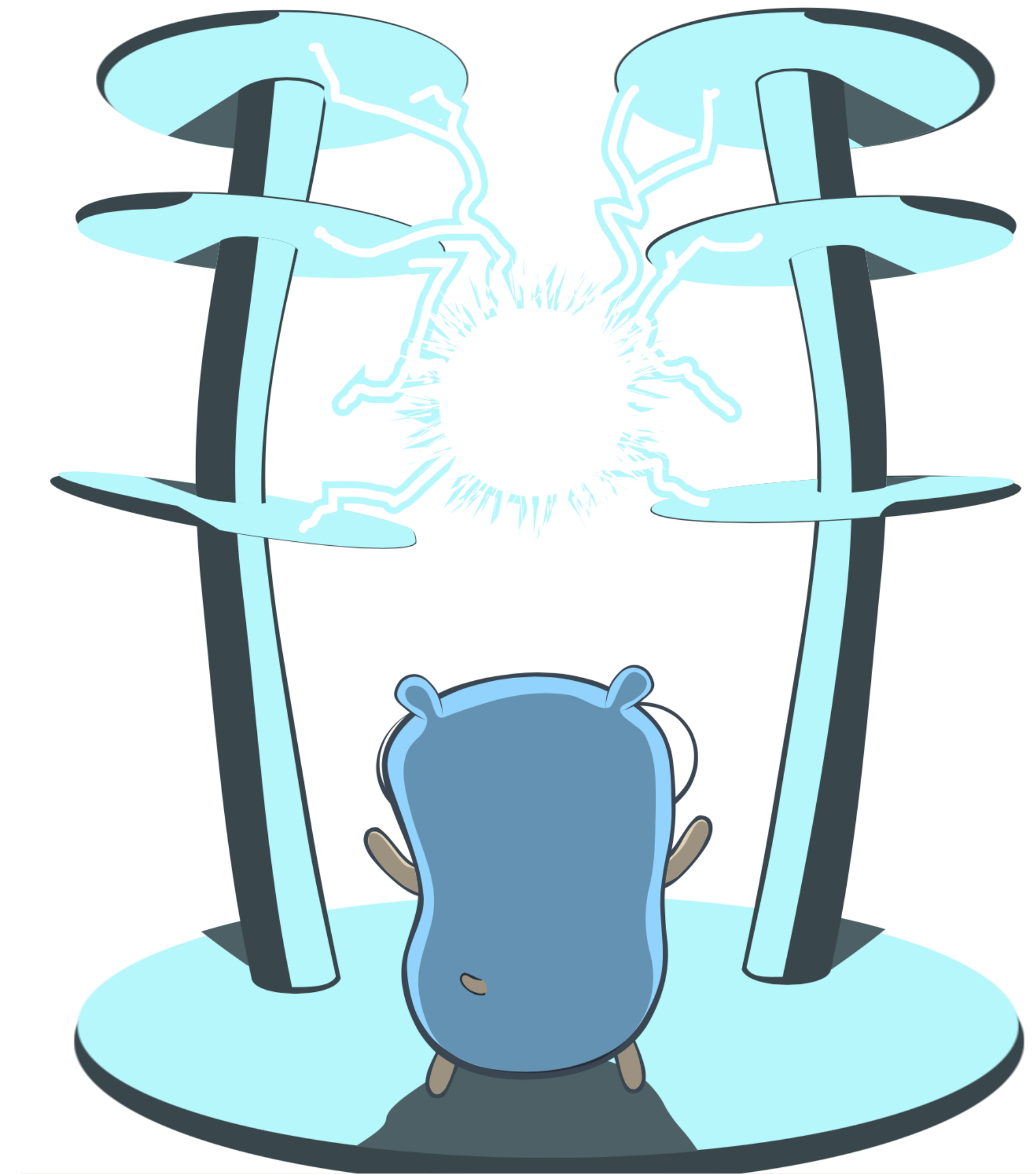# Go 1.9

Coolest Go release ever*

@idanyliuk
BCN Golang, Sep 28, 2017

*every Go release is coolest ever by definition

Performance

- As usual, most of the programs should be a bit faster

- Speedups in different libs

- GC optimizations
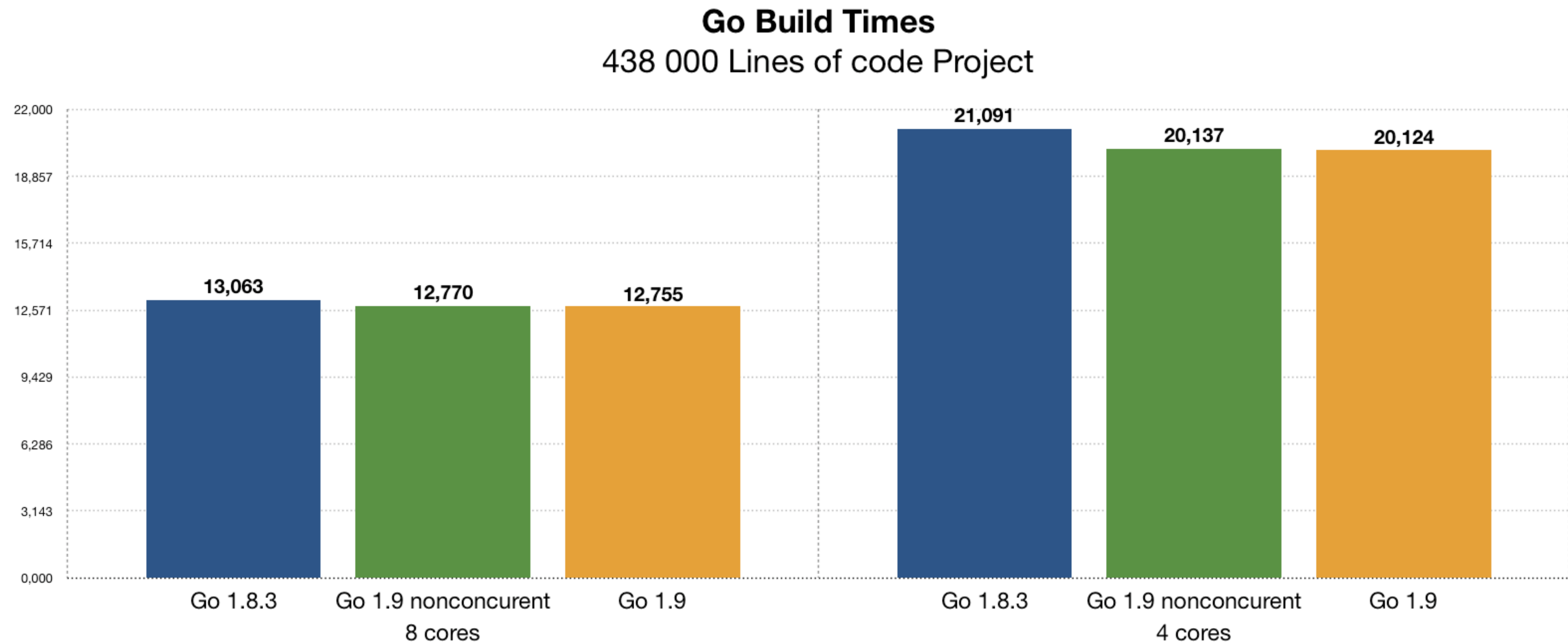
- Better generated code

Parallel compilation

- Go has always compiled files in parallel

- In Go1.9 functions also can be processes/ compiled in parallel

- Can be disabled by:

```
export GO19CONCURRENTCOMPILATION=0
```

Parallel compilation

- Benefit depends on the width and hight of your packages

- For "many packages/not so many functions" gain is small

Go Build Times

438 000 Lines of code Project

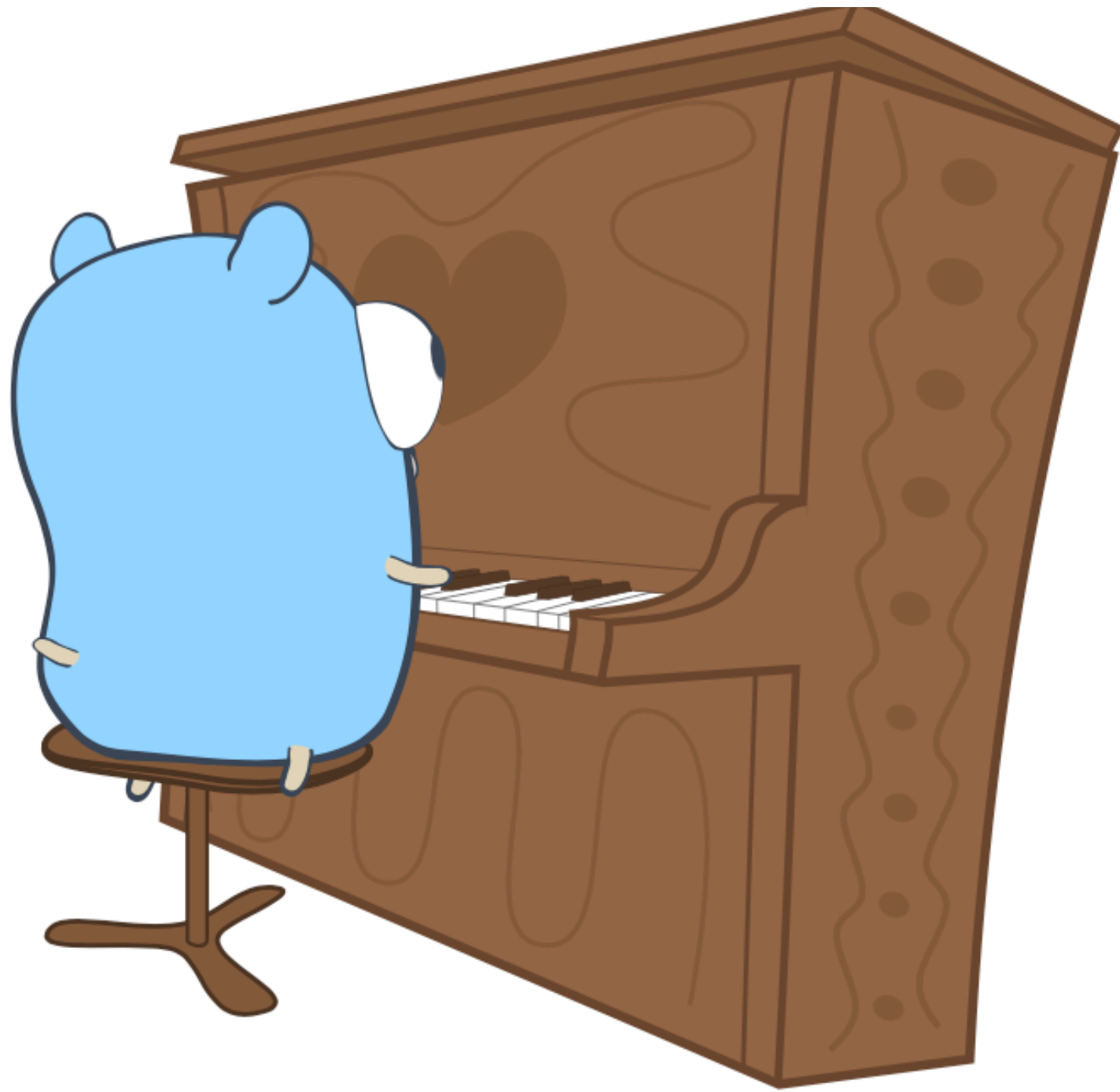|	8 cores			4 cores	
Go 1.8.3	Go 1.9 nonconcurent	Go 1.9	Go 1.8.3	Go 1.9 nonconcurent	Go 1.9
13,063	12,770	12,755	21,091	20,137	20,124

Type Aliases

- For refactoring large codebases

- T1 denotes **the same type** as T2

- Don't use it for anything else

```
type OldAPI = NewPackage.API
```
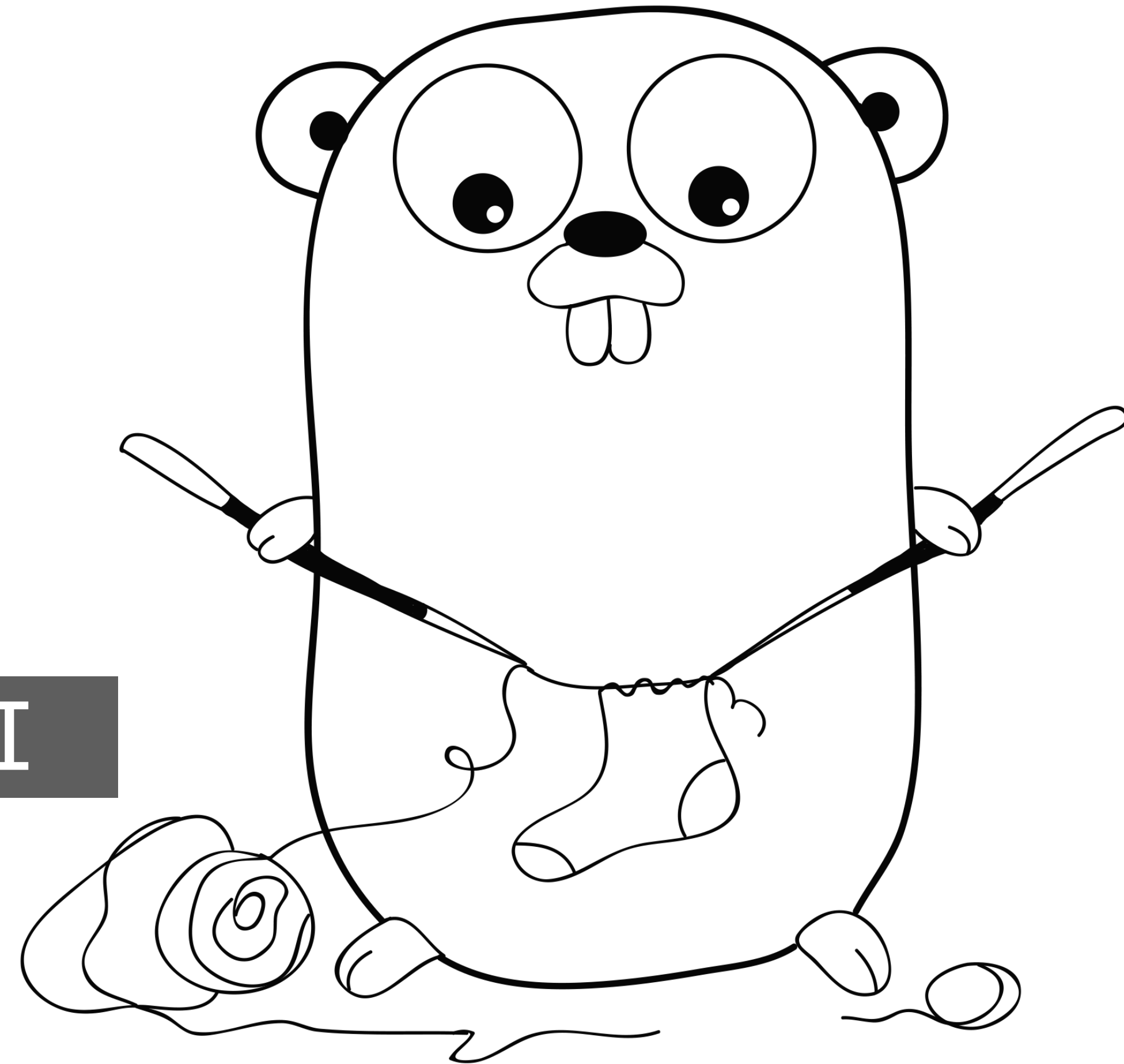
./... ignores vendor/ 🤘

- No more:

```
go test $(go list ./... | grep -v vendor)
```
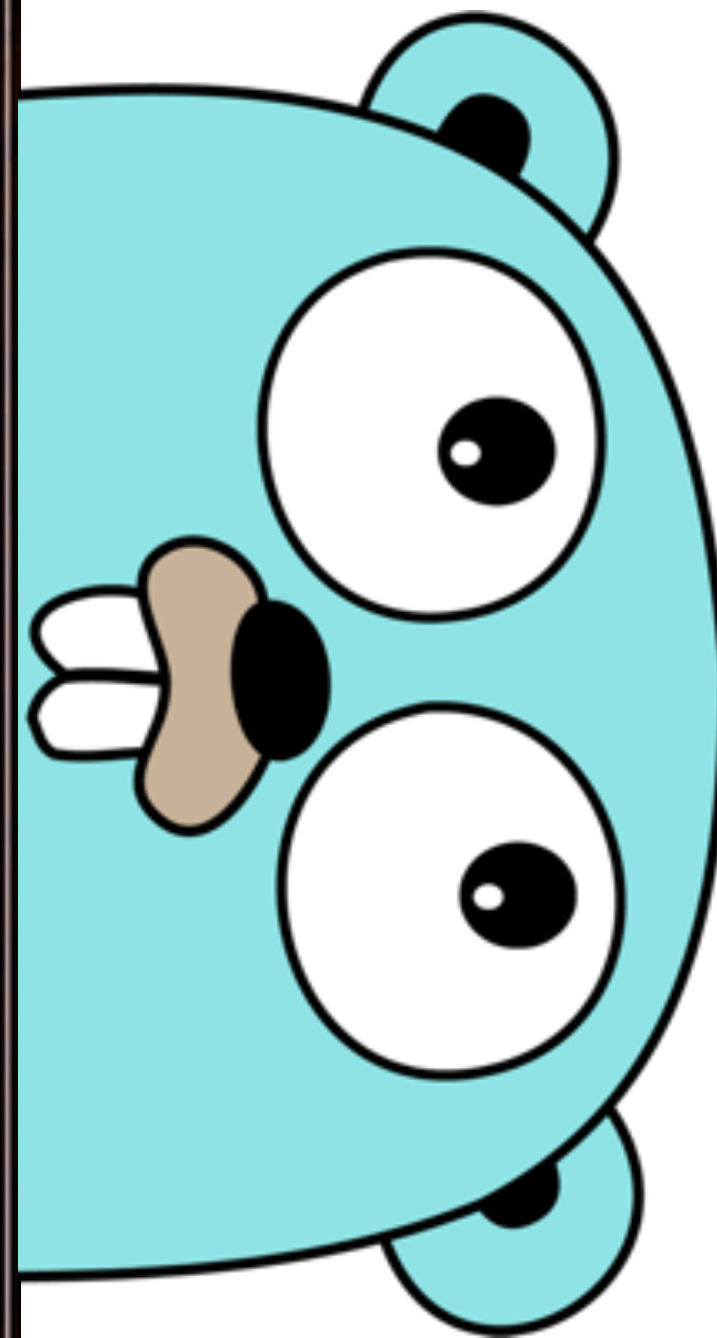
- Just run:

```
go test ./...
```
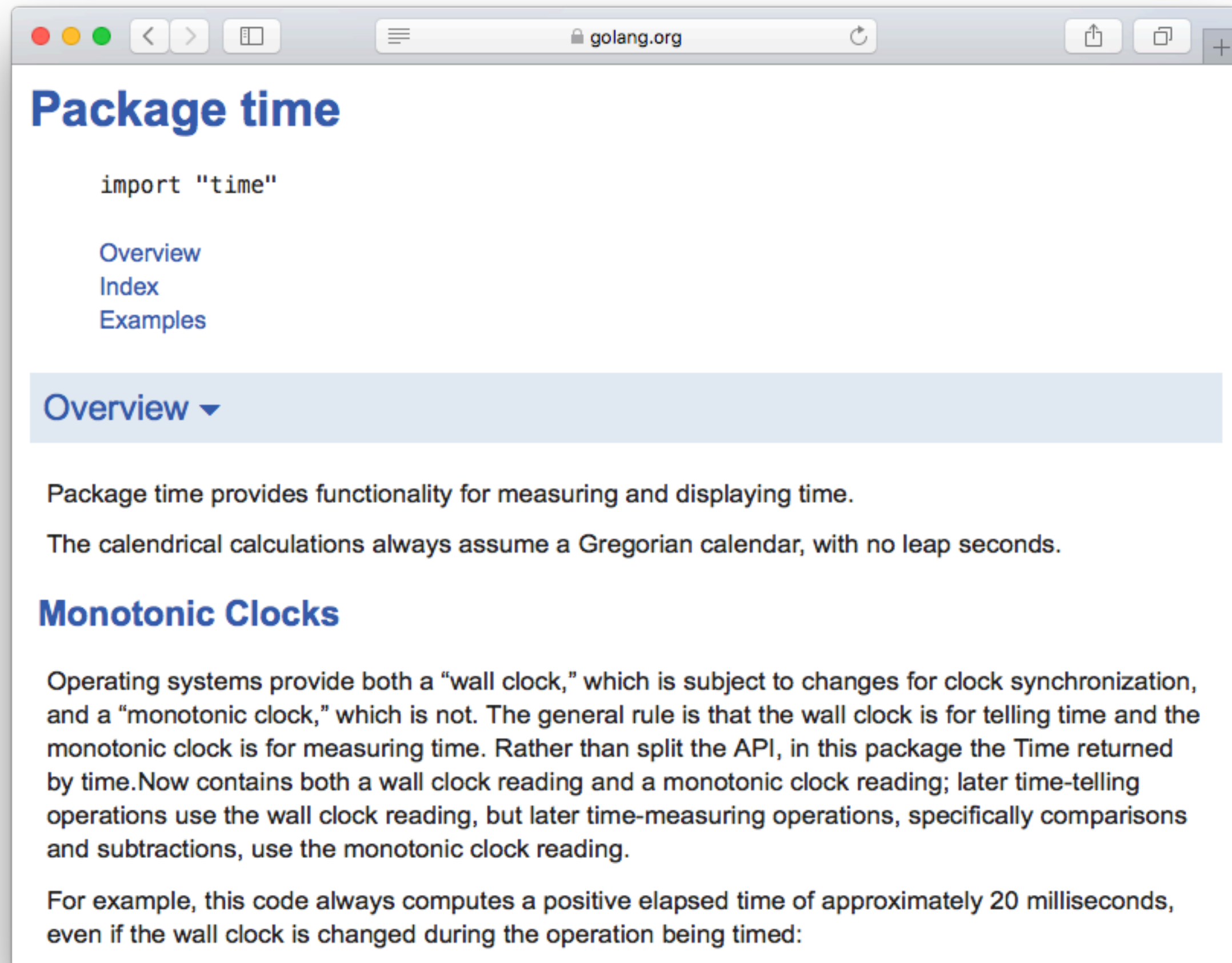
- If you need to also test vendor/:

```
go test ./vendor/...
```

Monotonic time support

- Each OS has two clocks - "*wall*" and "*monotonic*"

- *Wall* clock - for telling time

- *Monotonic* clock - for measuring time

- If time changes (sync, "leap second"), time.Duration() before Go 1.9 could return wrong measurement

Monotonic time support

- CloudFlare wrote a blog post of how this absence of monotonic clock support in Go causes serious outage.

- Rationale of using monotonic clocks are well described in the GoDoc for time package in Go 1.9

- What is cool: there is no change in API

- Go will use right clock for the right task

Monotonic time support

Don't use Time.String() for comparison:

```
now := time.Now()
now.String()
```

Go 1.8

2017-09-24 20:05:54.356882078 +0200 CEST

Go 1.9

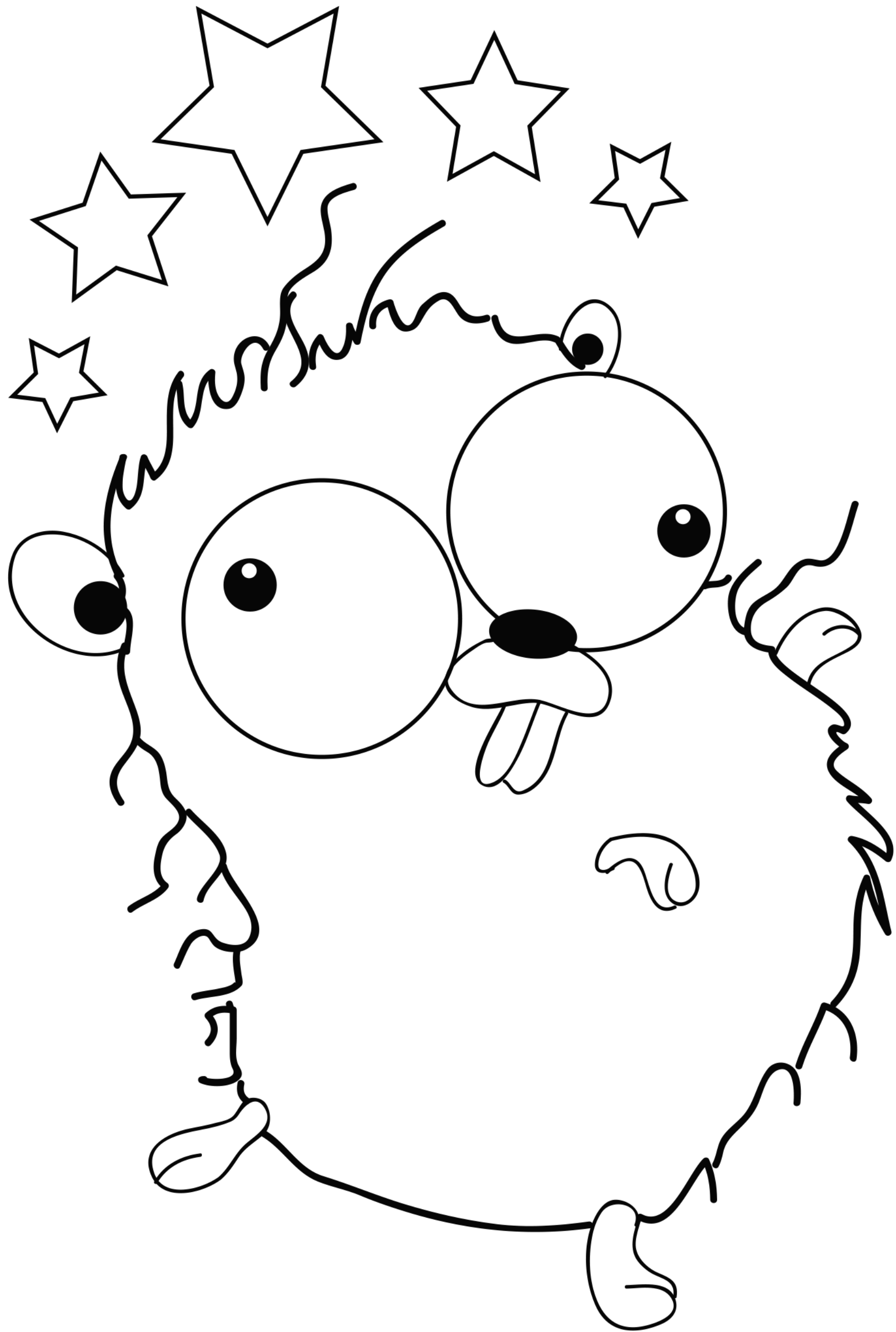2017-09-24 20:05:38.755165304 +0200 CEST **m=+0.000259428**

Use instead:

```
now.Format()
```

sync.Map

- Concurrent map that solves **specific case of cache contention**:

  - high-performance (ns makes a diff)

  - stable keys

  - many CPU cores (16 and more)

- In other cases, map+RWLock mutex is generally better

Cache Contention

sync.Map

- You use **encoding/json**...

- ...encoding/json uses **reflect**

- ...reflect uses sync.**RWMutex**

- ...RWMutex uses **atomic.AddInt32** to update readers counter

- ...each reader needs to **invalidate L2 cache and transfer value** from other core cache

- L2 cache transfer is around **40ns** on modern CPU

- O(1) task becomes **O(N)** by number of cores = **cache contention**

sync.Map

```
37
38  // RLock locks rw for reading.
39  //
40  // It should not be used for recursive read locking; a blocked Lock
41  // call excludes new readers from acquiring the lock. See the
42  // documentation on the RWMutex type.
43  func (rw *RWMutex) RLock() {
44          if race.Enabled {
45                  _ = rw.w.state
46                  race.Disable()
47          }
48          if atomic.AddInt32(&rw.readerCount, 1) < 0 {
49                  // A writer is pending, wait for it.
50                  runtime_Semacquire(&rw.readerSem)
51          }
52          if race.Enabled {
53                  race.Enable()
54                  race.Acquire(unsafe.Pointer(&rw.readerSem))
55          }
56  }
```

sync.Map

Benchmark from <u>original issue</u> on RWMutex's cache contention:

sync.Map

Performance of sync.Map vs builtin map (RWMutex)
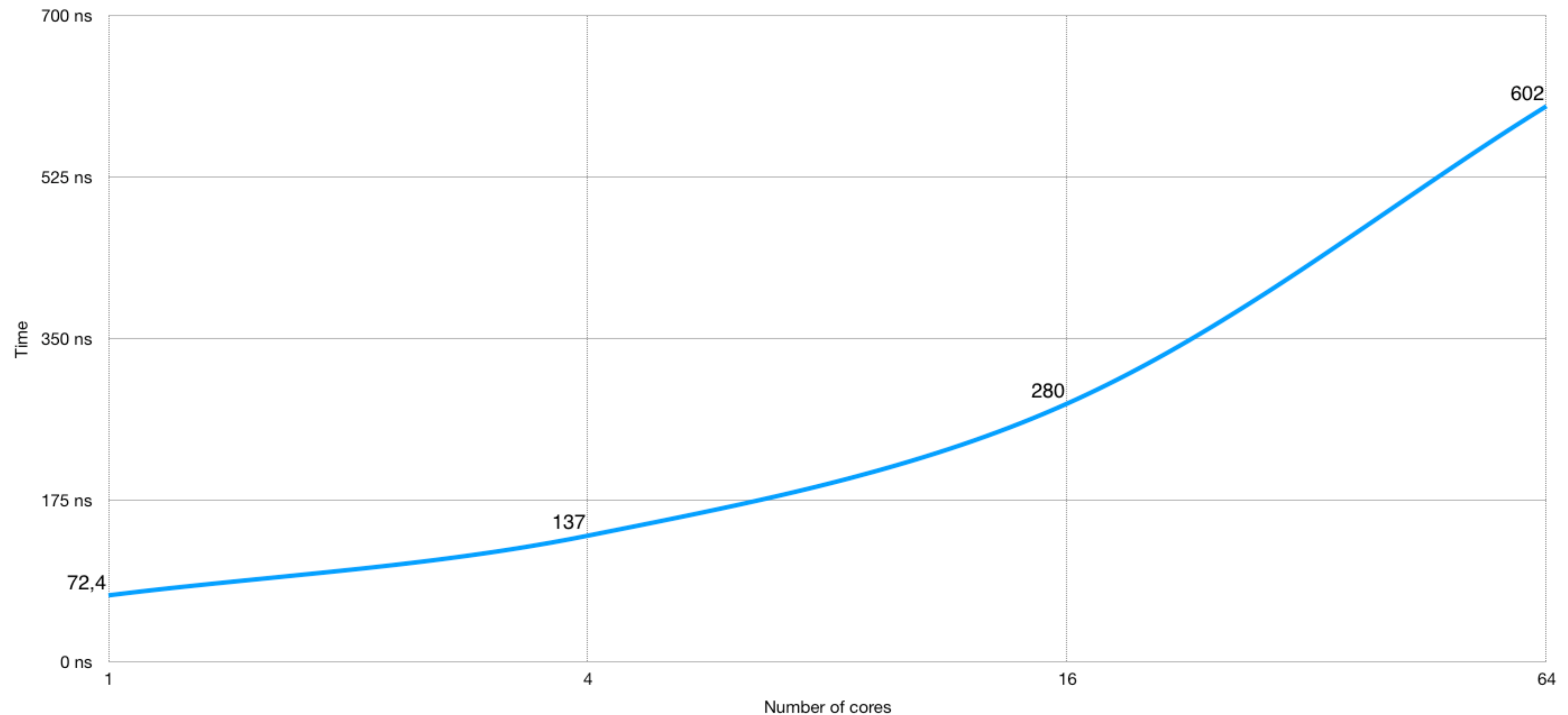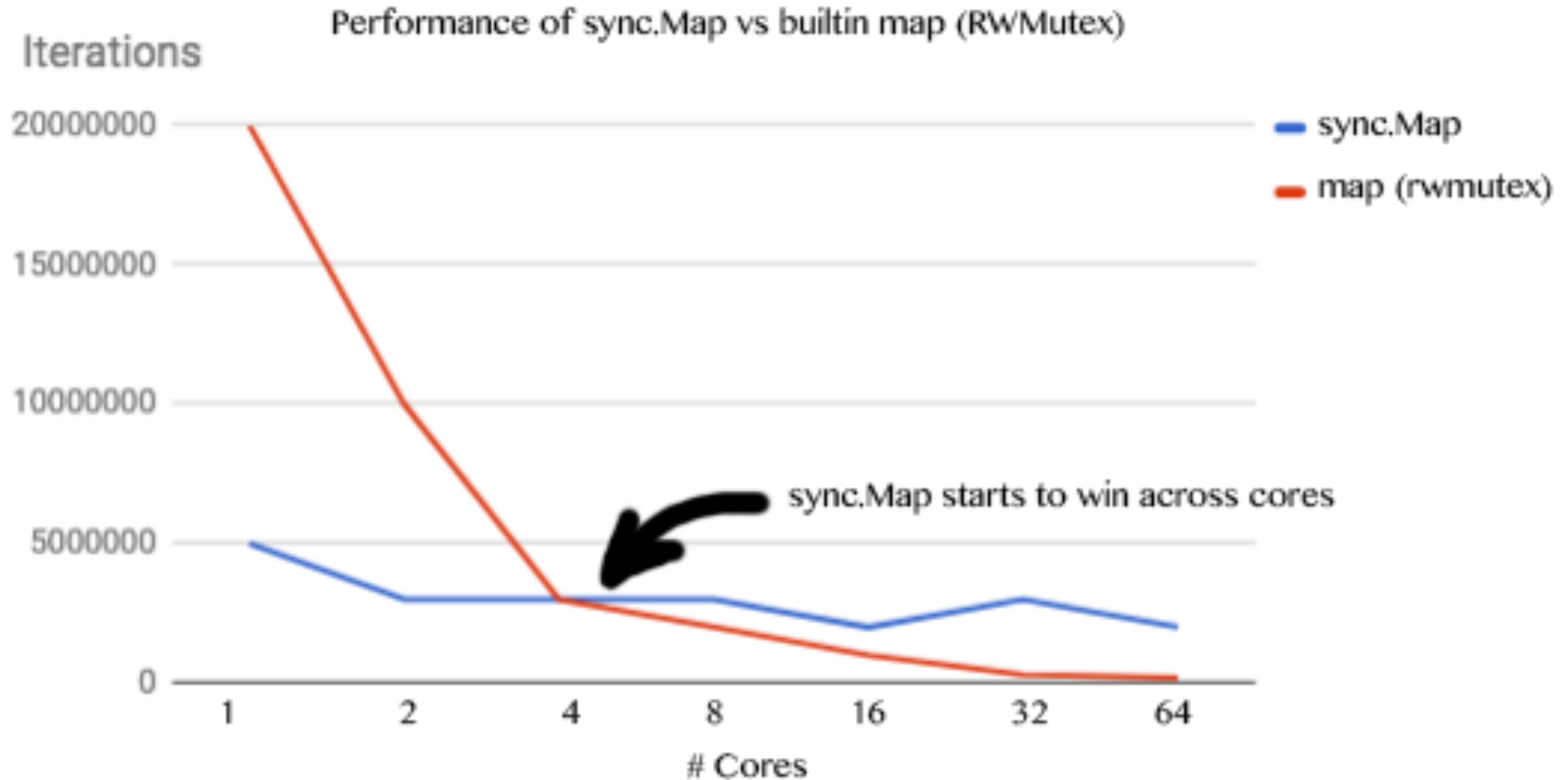
Map

```
m := make(map[string]int64)

m["key"] = 42

val, ok := m["key"]

delete(m, "key")
```

sync.Map

```
var m sync.Map

m.Store("key", 42)

val, ok := m.Load("key")

m.Delete("key")

val, ok := m.LoadOrStore("key")
m.Range(func(k, v interface{}) bool
{
    fmt.Println("key", k, "val", v)
    return true
})
```
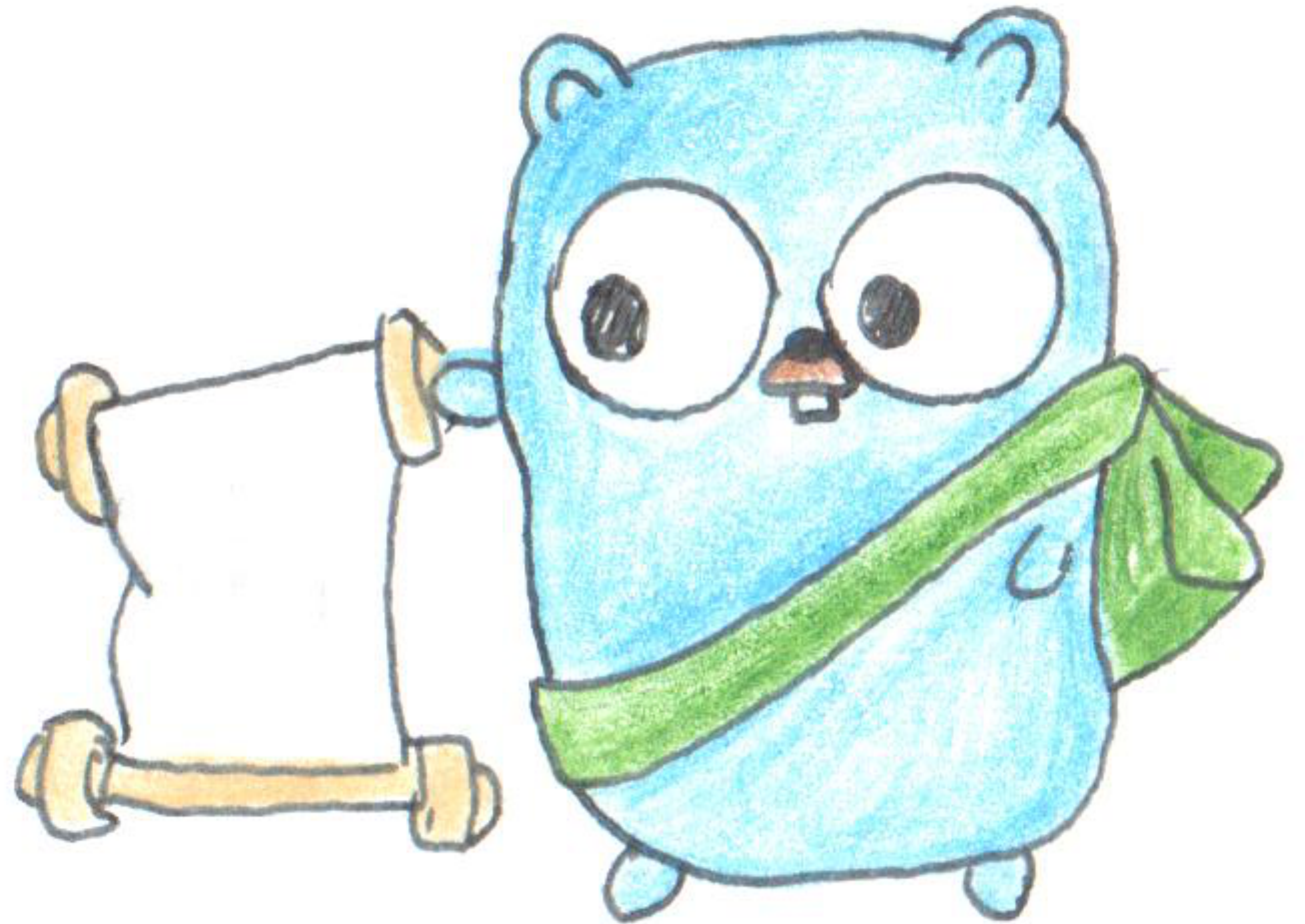
PProf

- Profile files now contain symbol information

- Means - no need to keep binaries

```
go tool pprof cpu.prof
```

- Super useful for profiling remote servers or cross compiled apps
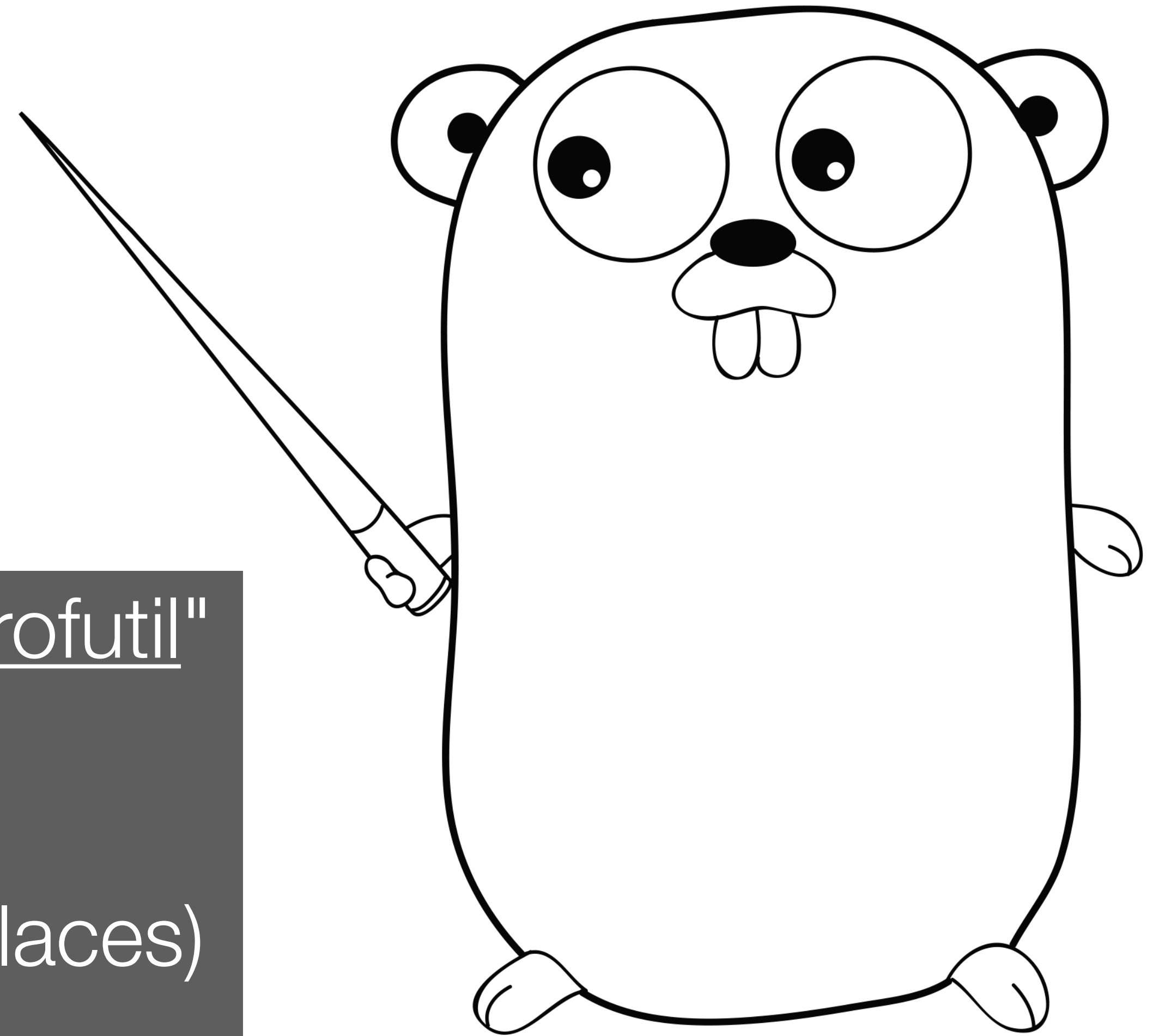
Profile Labels

- New feature in profile - custom labels

- Adds label to functions you profiling

```go
l := pprof.Labels("ext", "zip")
pprof.Do(ctx, l,
    func(ctx context.Context) {
        myFunc(ctx, args)
    }
)
```
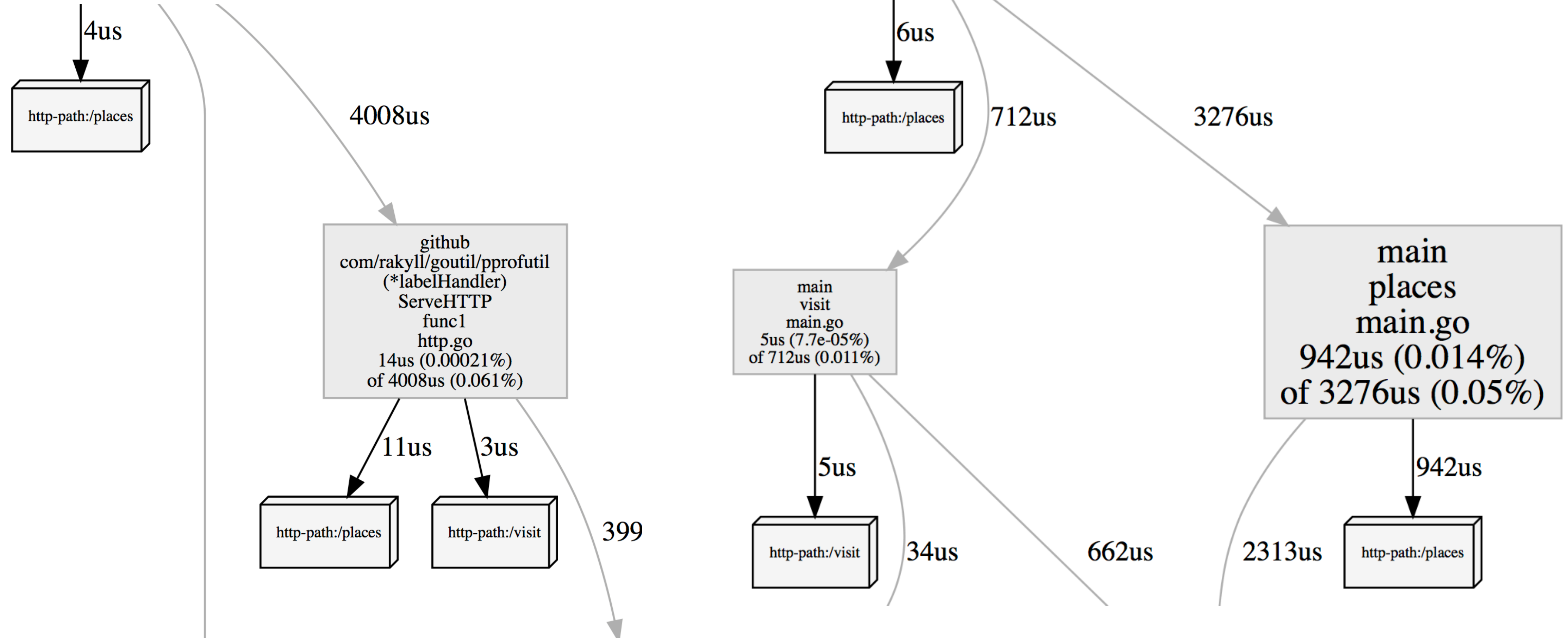
Profile Labels

- https://rakyll.org/profiler-labels/

- Example: pprofutil package,

- wrapper for http.Handler

- Adds "http-path" labels to each request

```
import "github.com/rakyll/goutil/pprofutil"

http.Handle("/places",
    pprofutil.LabelHandlerFunc(places)
)
```
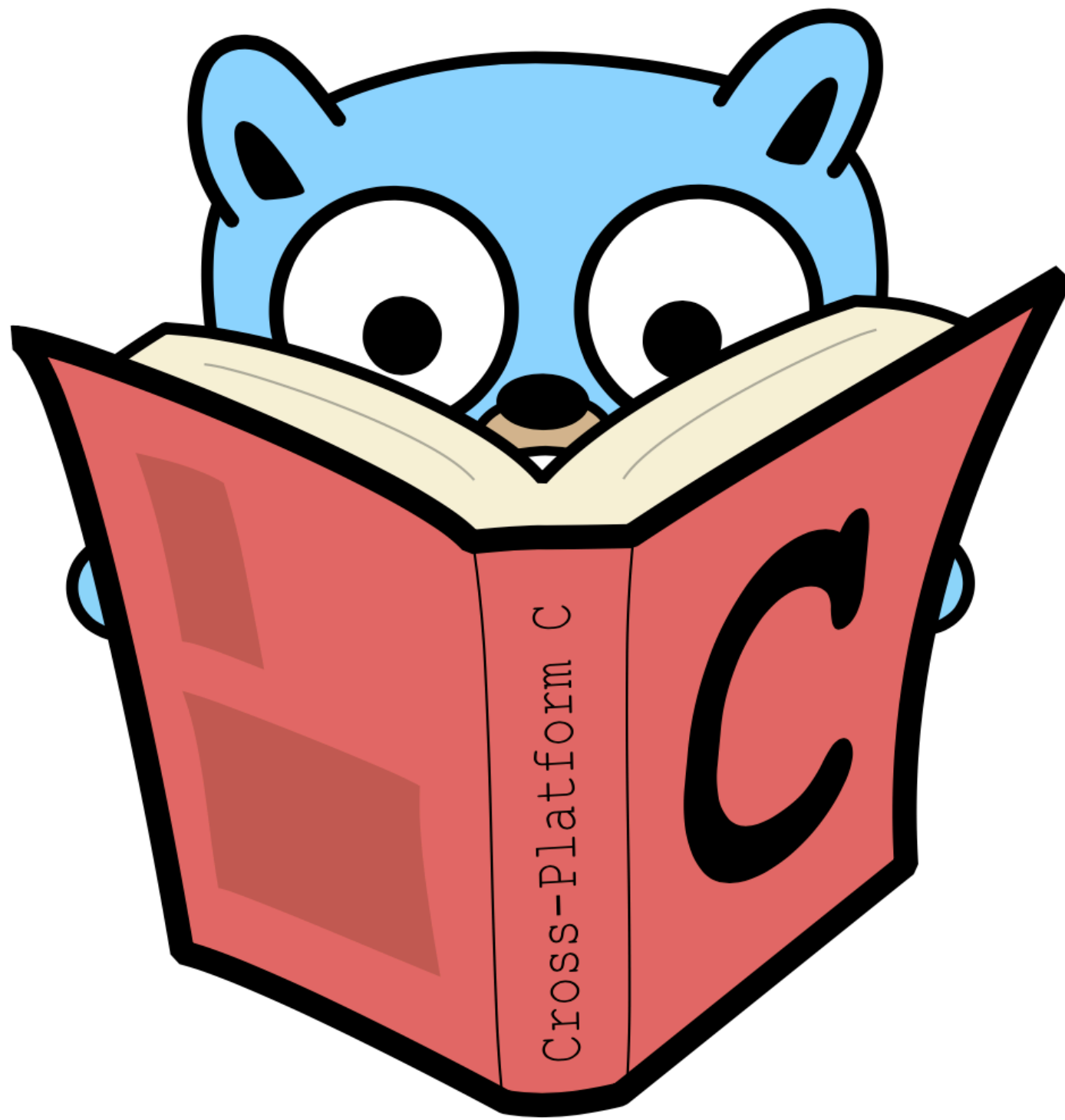
Profile Labels

`(pprof) tagfocus="http-path"`
`(pprof) web`

net/http
HandlerFunc
ServeHTTP
server.go
6us (9.2e-05%)
of 3994us (0.061%)

4us

http-path:/places

4008us

6us

http-path:/places

712us

3276us

github
com/rakyll/goutil/pprofutil
(*labelHandler)
ServeHTTP
func1
http.go
14us (0.00021%)
of 4008us (0.061%)

main
visit
main.go
5us (7.7e-05%)
of 712us (0.011%)

main
places
main.go
942us (0.014%)
of 3276us (0.05%)

11us   3us

5us

942us

http-path:/places   http-path:/visit

399

http-path:/visit

34us

662us

2313us

http-path:/places

# More stdlib changes

math/bits

- New package with bit manipulation functions

- Highly optimized for different CPU architectures

  - Rotate bits

  - Count zeroes / ones

  - Reverse bits/bytes

- If you're asked on interview how to reverse bits, use math/bits :)
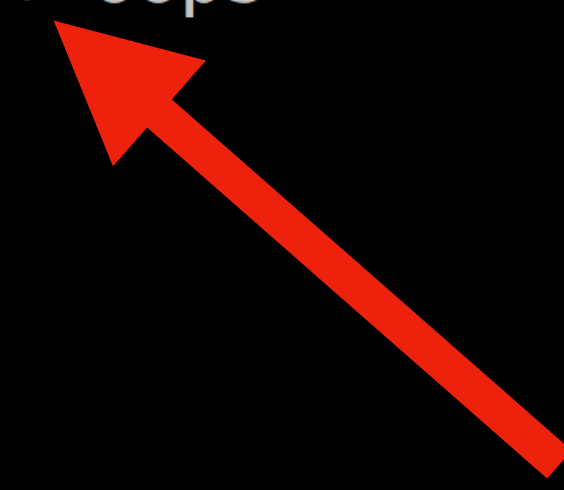
Test helpers functions

- In testing package:

  · **(*T).Helper()**

  · **(*B).Helper()**

· Marks functions as helper, so it's skipped when reports file:line information in log.
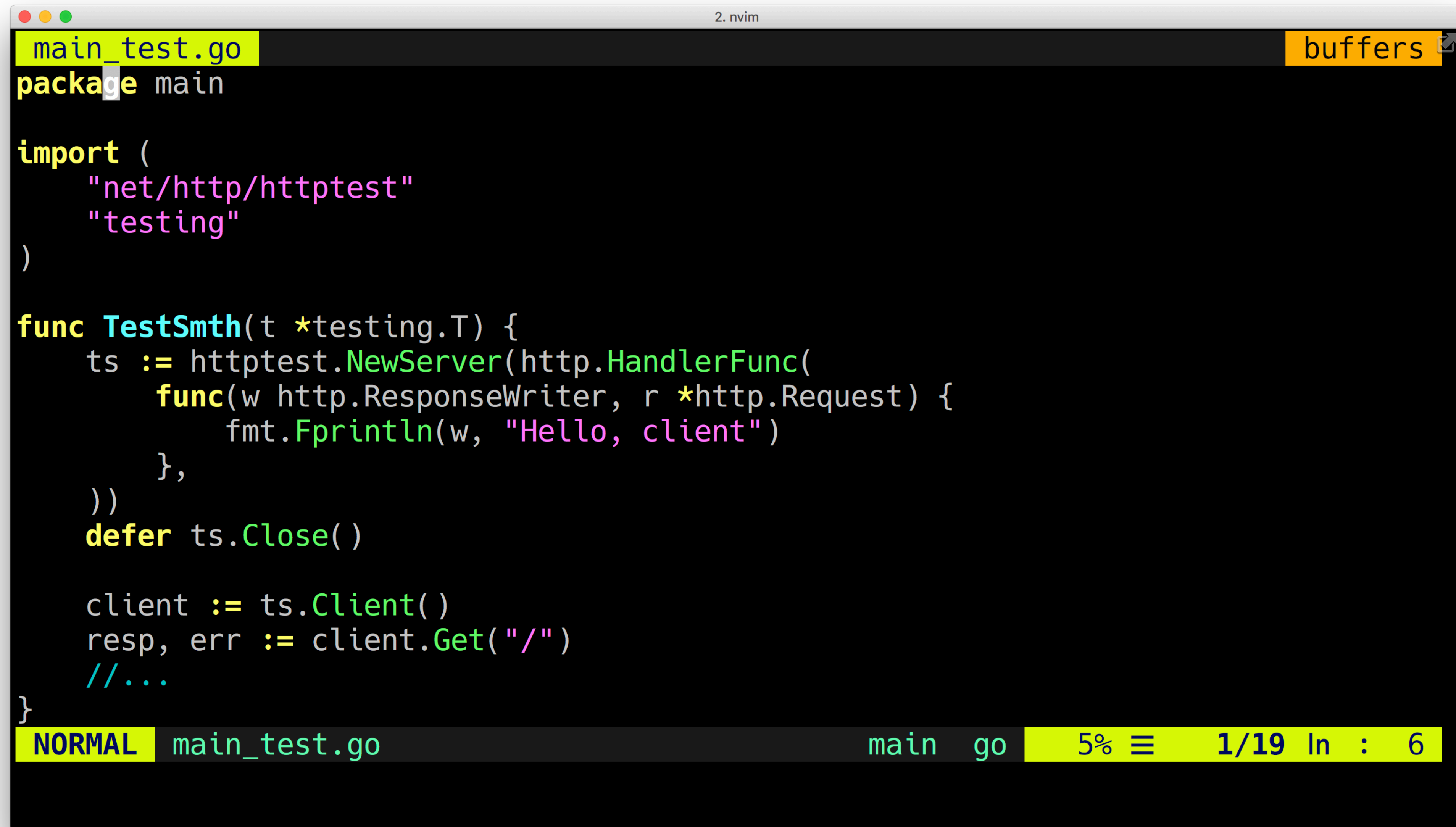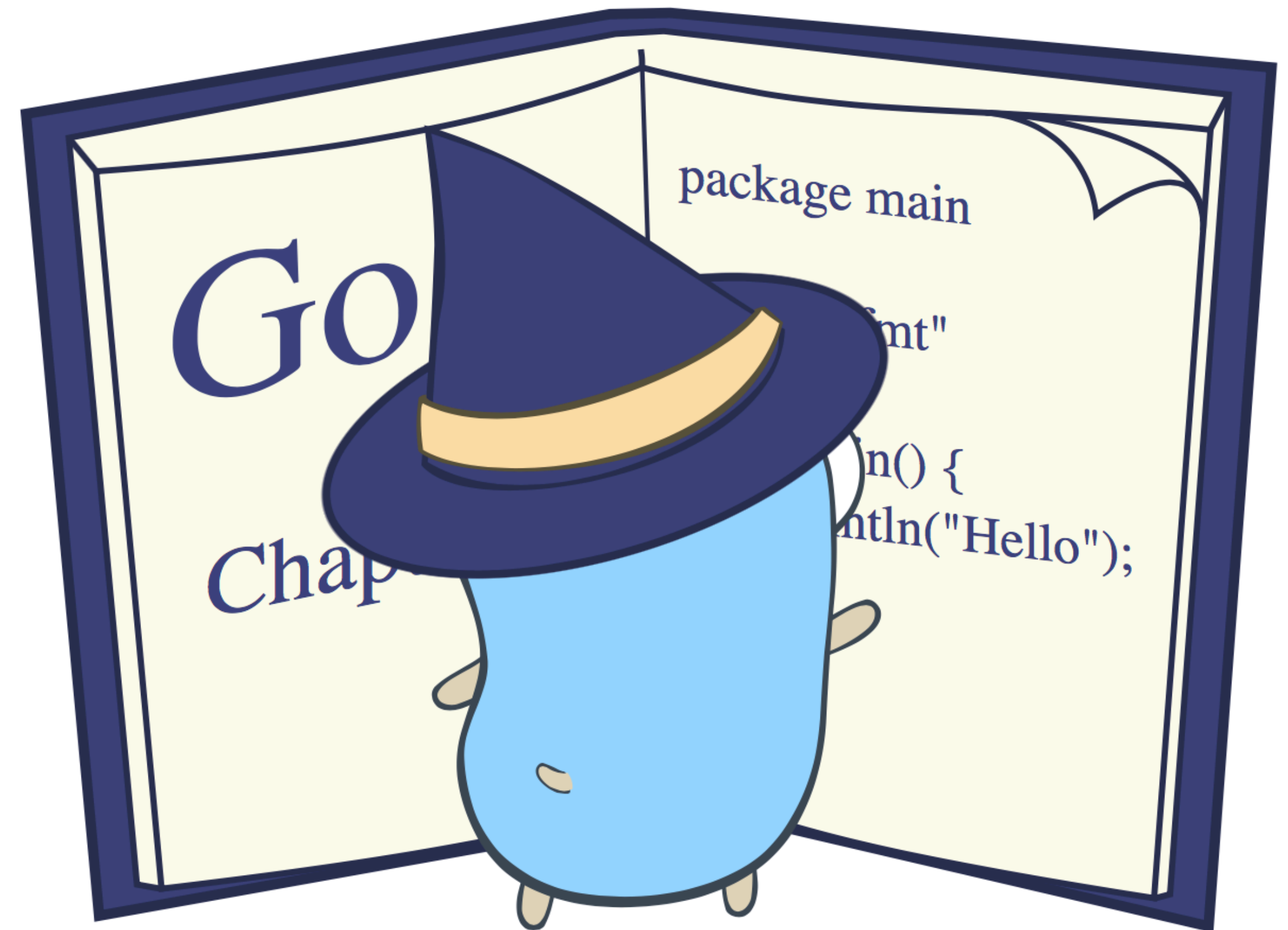
Test helpers functions

```go
package main

import (
    "errors"
    "testing"
)

func Something() error {
    return errors.New("oops")
}

func TestSomething(t *testing.T) {
    err := Something()
    checkErr(t, err) // line 14
}

func checkErr(t *testing.T, err error) {
    t.Helper()
    if err != nil {
        t.Fatal(err) // line 20
    }
}
```

```
Last login: Mon Sep 25 01:47:50 on ttys008
[mac@test ]$ go test -v
=== RUN   TestSomething
--- FAIL: TestSomething (0.00s)
        main_test.go:14: oops
FAIL
exit status 1
FAIL    test    0.006s
[mac@test ]$
```

httptest.Server.Client()


```go
package main

import (
    "net/http/httptest"
    "testing"
)

func TestSmth(t *testing.T) {
    ts := httptest.NewServer(http.HandlerFunc(
        func(w http.ResponseWriter, r *http.Request) {
            fmt.Fprintln(w, "Hello, client")
        },
    ))
    defer ts.Close()

    client := ts.Client()
    resp, err := client.Get("/")
    //...
}
```
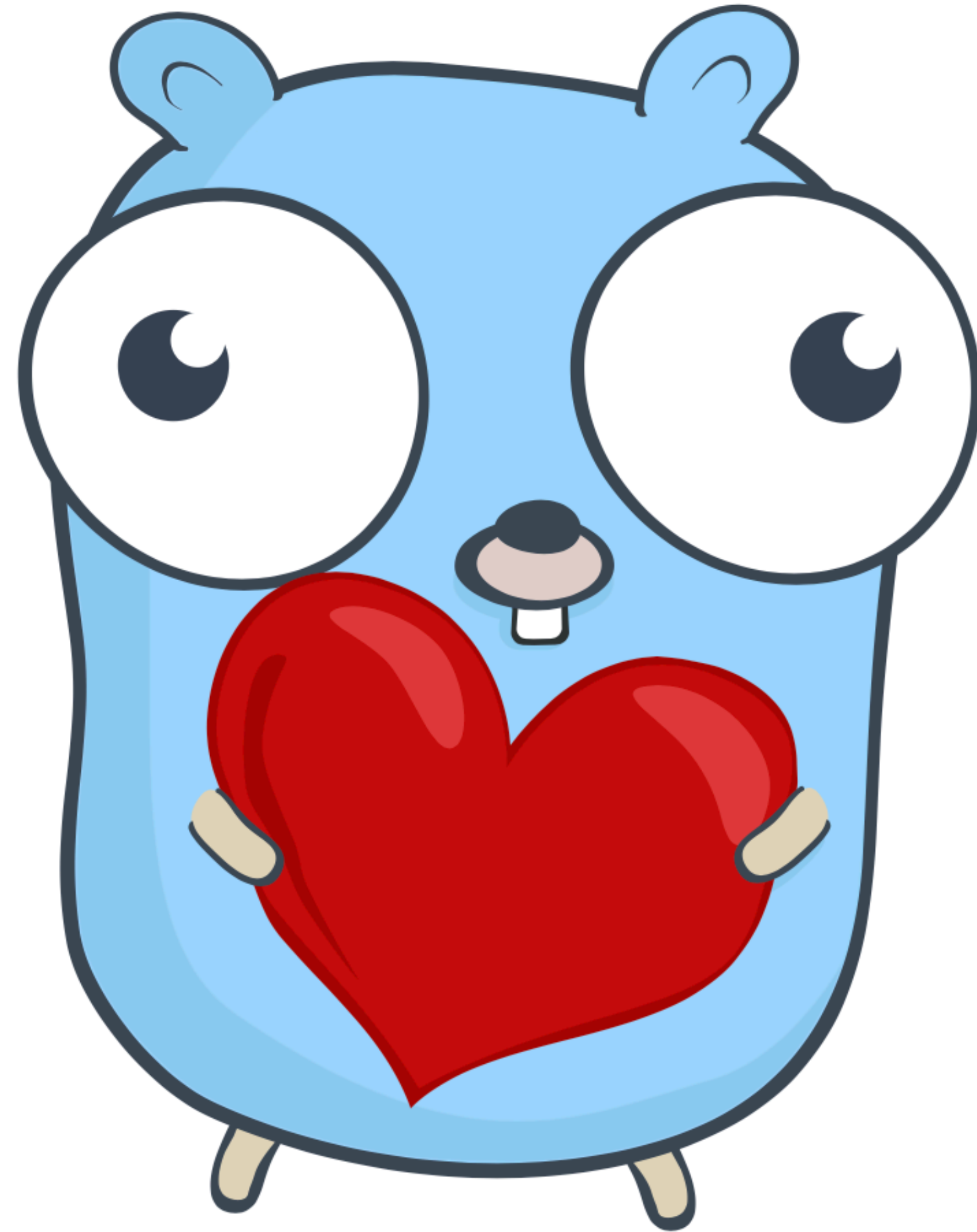
Go 1.9 is cool

- Upgrade now!

- Amazing artwork by @egonelbre and Olga Shalakhina

Thank you