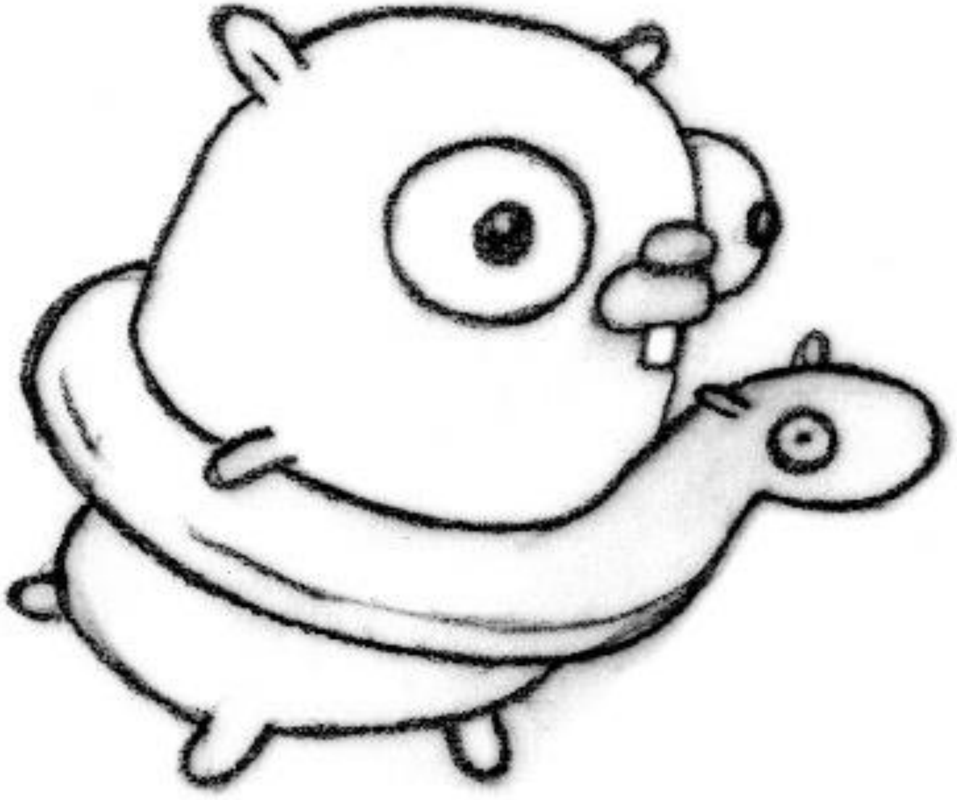
Intro to programming in **Go**

Vladimir Alekseichenko

Agenda

- History
- What is Go?
- What is **not** Go?
- Concurrency
- Organizations using Go
- Examples

Get ready



http://talks.golang.org/2013/advconc.slide#3

Introduction

- **Go** is a concurrent ***open source*** programming language developed at **Google**.

- Combines **native compilation** and **static types** with a lightweight dynamic feel.

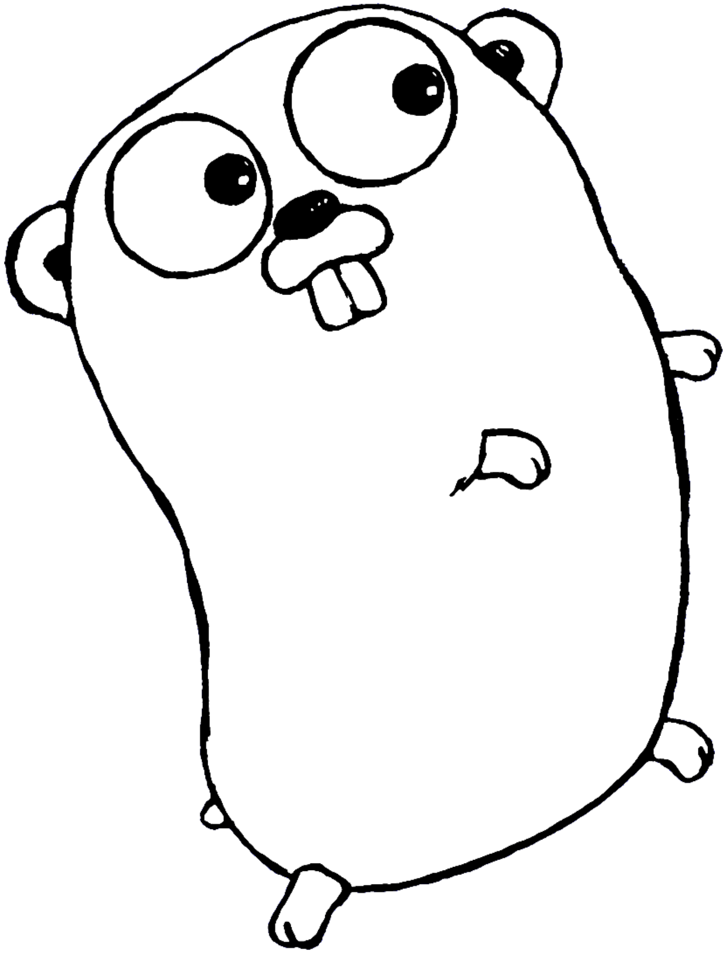
- Fast, fun, and productive.

http://talks.golang.org/2012/go1.slide#2

History

- Design began in late 2007.
- Became open source in November 2009.

Language stable as of Go 1, early 2012.



Go's mascot is a **gopher** designed by *Renée French*.

What is Go?

Go is a modern, general purpose language.

http://talks.golang.org/2011/Real_World_Go.pdf

What is Go?

- Native code generation (compiled)
- Statically typed
- Composition via interfaces
- Memory safe
- Garbage collected
- Native concurrency support
- Excellent standard library
- Great tools

http://talks.golang.org/2012/go1.slide#3

What is **not** Go?

- No type inheritance
- No method* or operator overloading
- No circular dependencies among packages
- No pointer arithmetic
- No assertions
- No generic programming

* *like in C# or Java ☺.*

http://en.wikipedia.org/wiki/Go_(programming_language)

Big hardware

http://www.dailymail.co.uk/sciencetech/article-2219188/Inside-Google-pictures-gives-look-8-vast-data-centres.html

Big software

- C++ (mostly) for servers, plus lots of Java and Python
- thousands of engineers
- gazillions of lines of code
- distributed build system
- one tree

http://talks.golang.org/2012/splash.slide

The reason for Go

Goals:

- eliminate slowness
- eliminate clumsiness
- improve effectiveness
- maintain (even improve) scale

http://talks.golang.org/2012/splash.slide

Pain

https://www.simple-talk.com/blogs/2008/01/08/the-pain-of-software-installation/

Pain

- slow builds
- uncontrolled dependencies
- each programmer using a different subset of the language
- poor program understanding (documentation, etc.)
- duplication of effort
- cross-language builds
- …

http://talks.golang.org/2012/splash.slide

Primary considerations

Must work at scale:

- large programs
- large teams
- large number of dependencies

- Must be familiar, roughly C-like

http://talks.golang.org/2012/splash.slide

Modernize

- The *old* ways are old.

Go should be:

- suitable for multicore machines
- suitable for networked machines
- suitable for web stuff

Install Go

golang.org/doc/install

Install from binary distributions or build from source 32- and 64-bit x86 and ARM processors Windows, Mac OS X, Linux, and FreeBSD.

Tools

- go build hello.go # Compile
- go run hello.go # Compile-and-go. (Ha!)
- go build package # Build everything in directory (and deps)
- go install # Install everything in dir and (and deps)
- go test archive/zip # Compile and run unit tests for package

http://talks.golang.org/2012/go1.slide#36

The go tool and remote repositories

Go tool downloads and installs all dependencies, transitively.

- **go get** *code.google.com/p/myrepo/mypack*

And to use the package in Go source:

- import "*code.google.com/p/myrepo/mypack*"

http://talks.golang.org/2012/go1.slide#37

Concurrency

Programming as the composition of independently executing processes.

(Processes in the general sense, not Linux processes. Famously hard to define.)

http://talks.golang.org/2012/waza.slide

Go supports concurrency

Go provides:

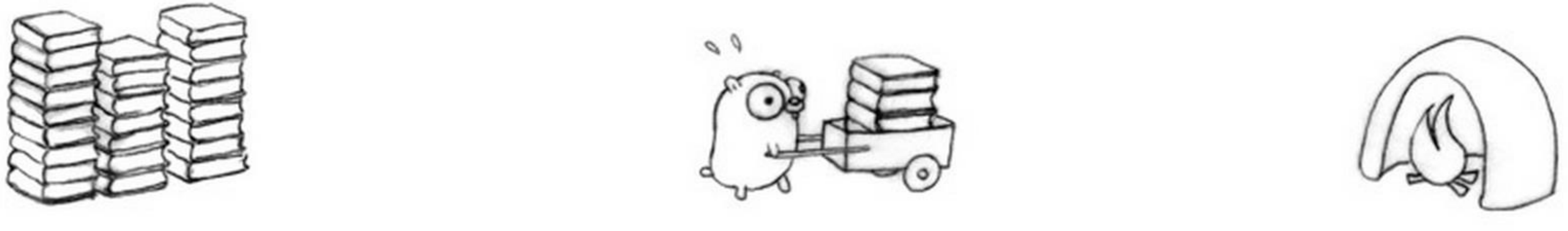
- concurrent execution (goroutines)
- synchronization and messaging (channels)
- multi-way concurrent control (select)

http://talks.golang.org/2012/waza.slide

http://talks.golang.org/2012/waza.slide

Our problem

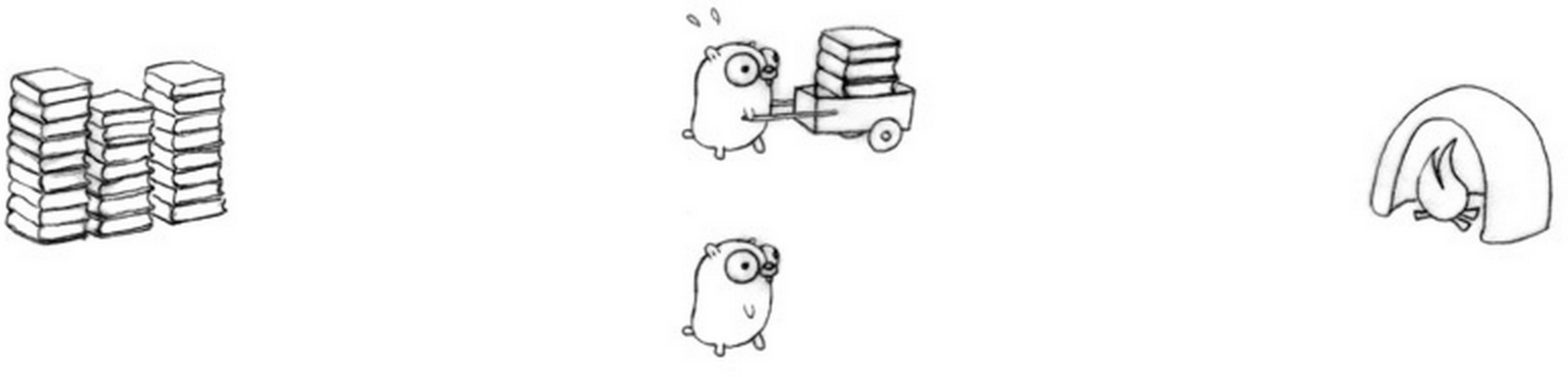
Move a pile of obsolete language manuals to the incinerator.



With only one gopher this will take too long.

http://talks.golang.org/2012/waza.slide

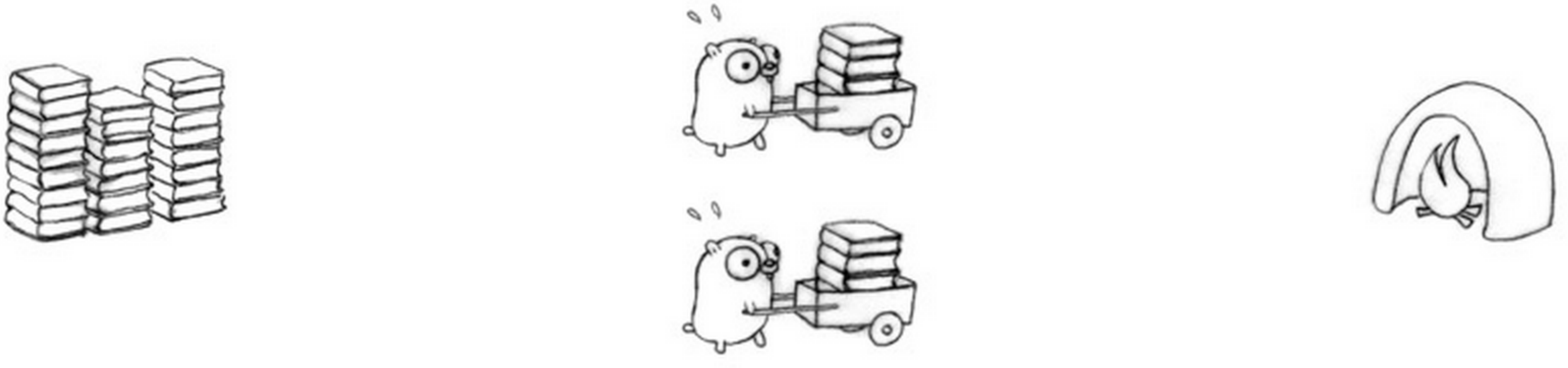
More gophers!



More gophers are not enough; they need more carts.

http://talks.golang.org/2012/waza.slide

More gophers and more carts



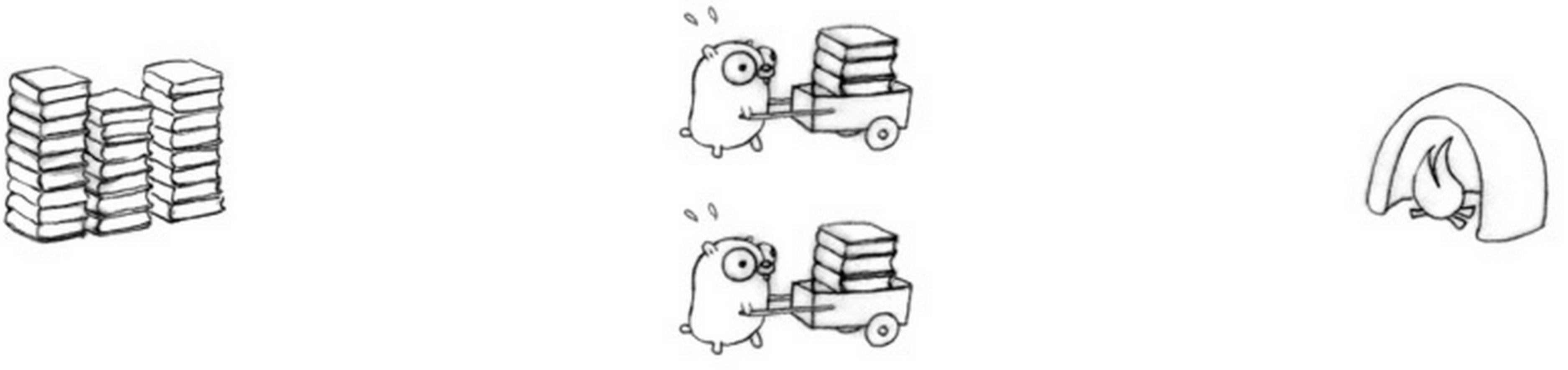
This will go faster, but there will be **bottlenecks** at the **pile** and **incinerator**.

Also need to **synchronize** the gophers.

http://talks.golang.org/2012/waza.slide

Double everything

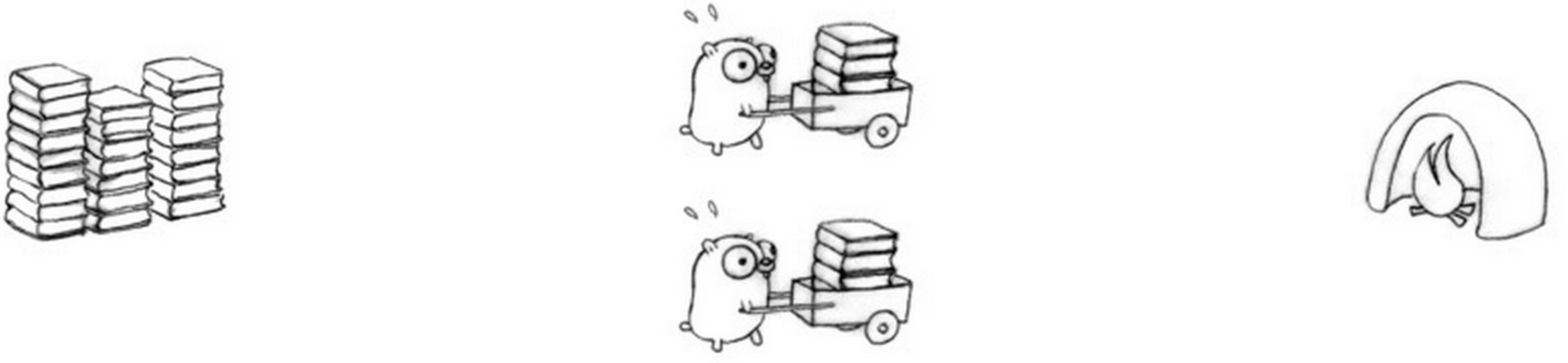
Remove the bottleneck; make them really independent.



This will consume input twice as fast.

http://talks.golang.org/2012/waza.slide

Concurrent composition



The concurrent composition of two gopher procedures.

http://talks.golang.org/2012/waza.slide

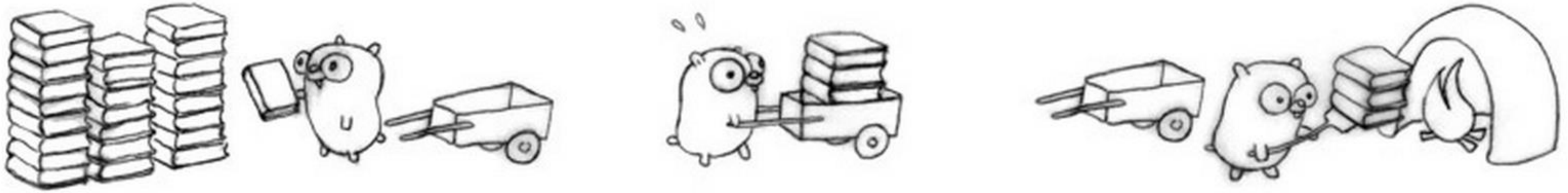
Concurrent composition

- This design is not automatically parallel!
- What if only one gopher is moving at a time? Then it's still concurrent (that's in the design), just not parallel.

- However, it's automatically parallelizable!

http://talks.golang.org/2012/waza.slide

Another design



Three gophers in action, but with likely delays.
Each gopher is an independently executing procedure, plus coordination (communication).

http://talks.golang.org/2012/waza.slide

Finer-grained concurrency

Add another gopher procedure to return the empty carts.

Four gophers in action for better flow, each doing one simple task.

http://talks.golang.org/2012/waza.slide

Concurrent procedures

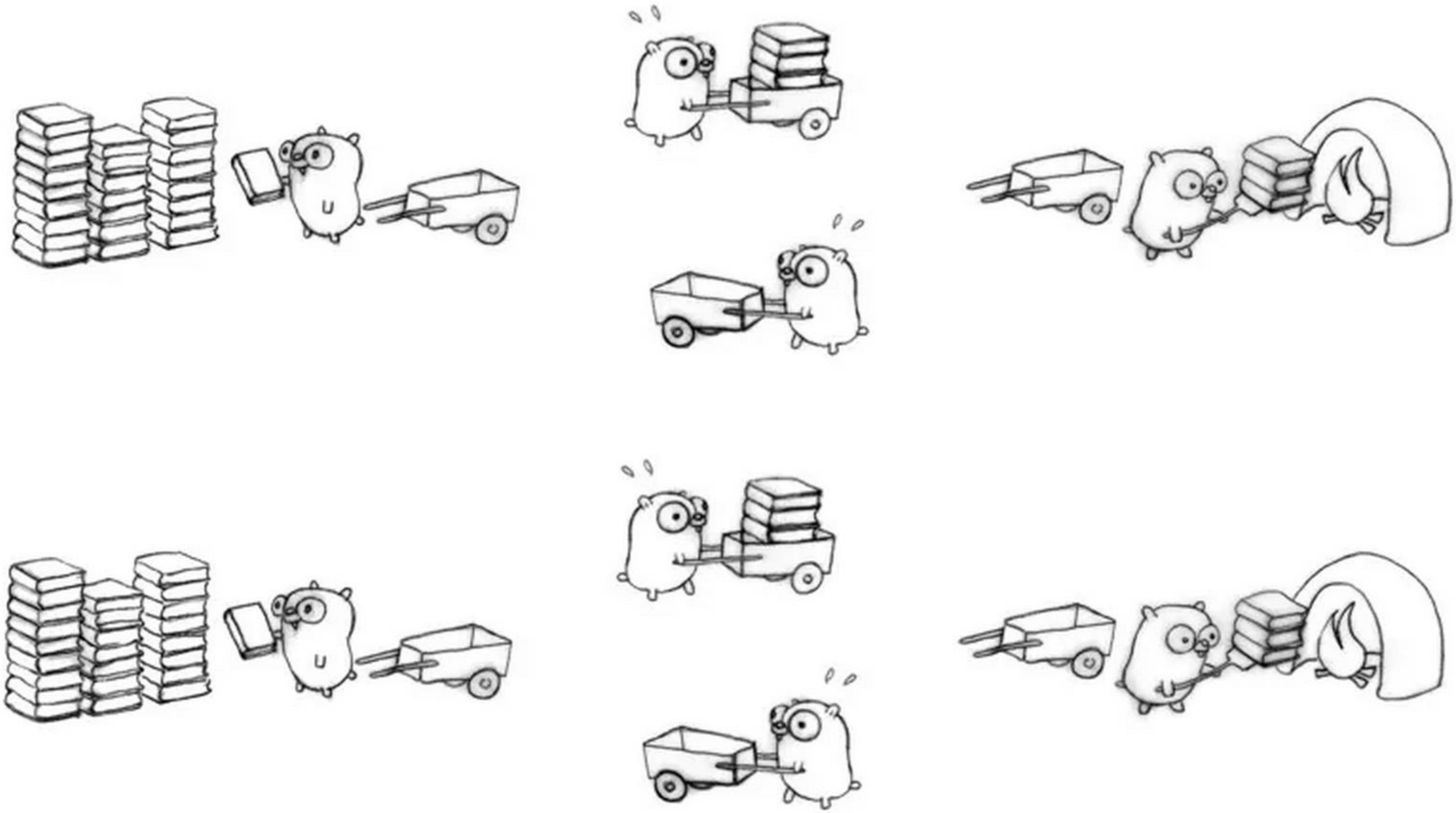
Four distinct gopher procedures:

- load books onto cart
- move cart to incinerator
- unload cart into incinerator
- return empty cart

Different concurrent designs enable different ways to parallelize.

http://talks.golang.org/2012/waza.slide

More parallelization!



http://talks.golang.org/2012/waza.slide

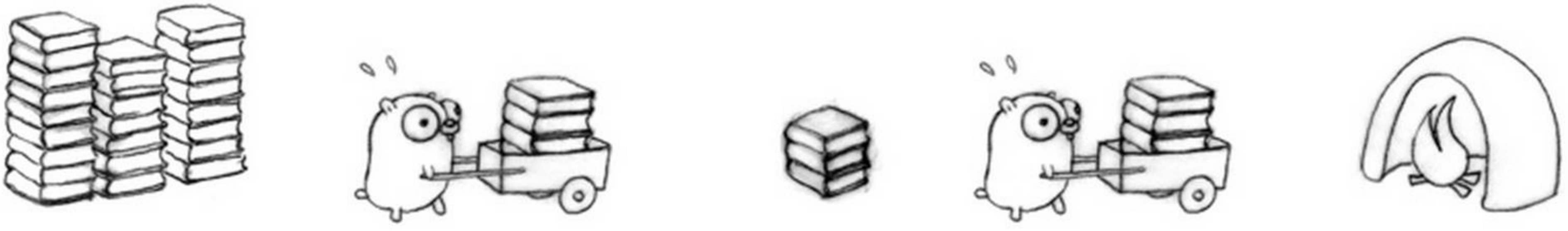
Or maybe no parallelization at all

Keep in mind, even if only one gopher is active at a time (zero parallelism), it's still a correct and concurrent solution.

http://talks.golang.org/2012/waza.slide

Another design

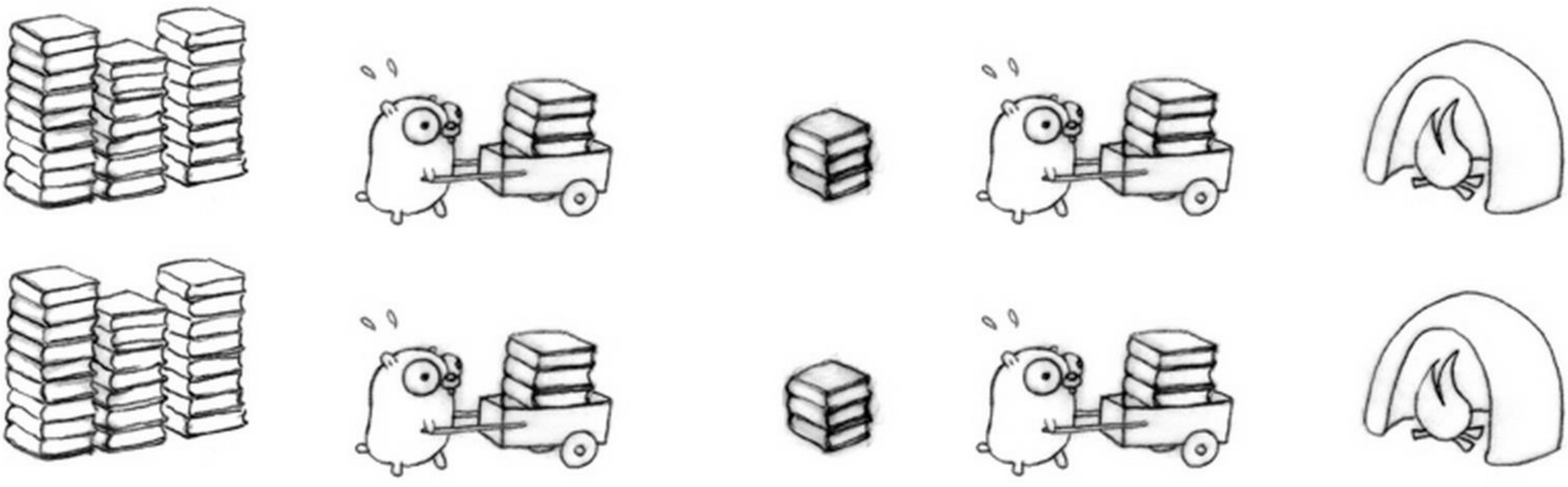
Two gopher procedures, plus a staging pile.



http://talks.golang.org/2012/waza.slide

Parallelize the usual way

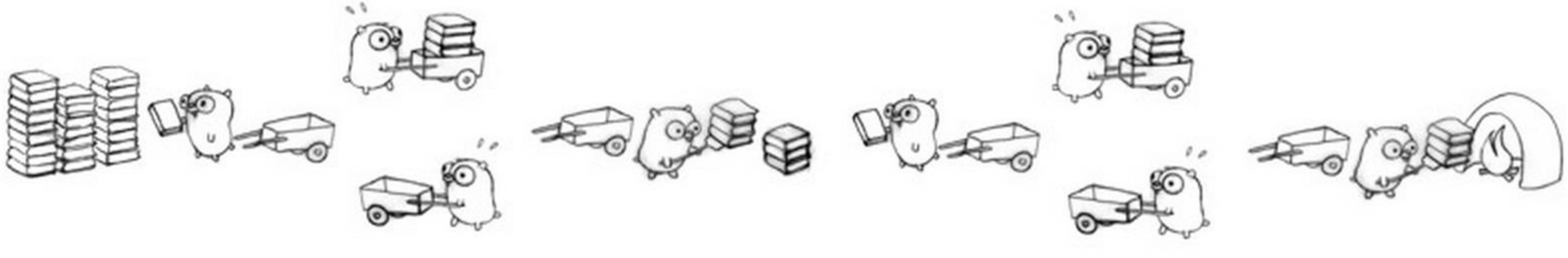
Run more concurrent procedures to get more throughput.



http://talks.golang.org/2012/waza.slide

Or a different way

Bring the staging pile to the multi-gopher concurrent model:

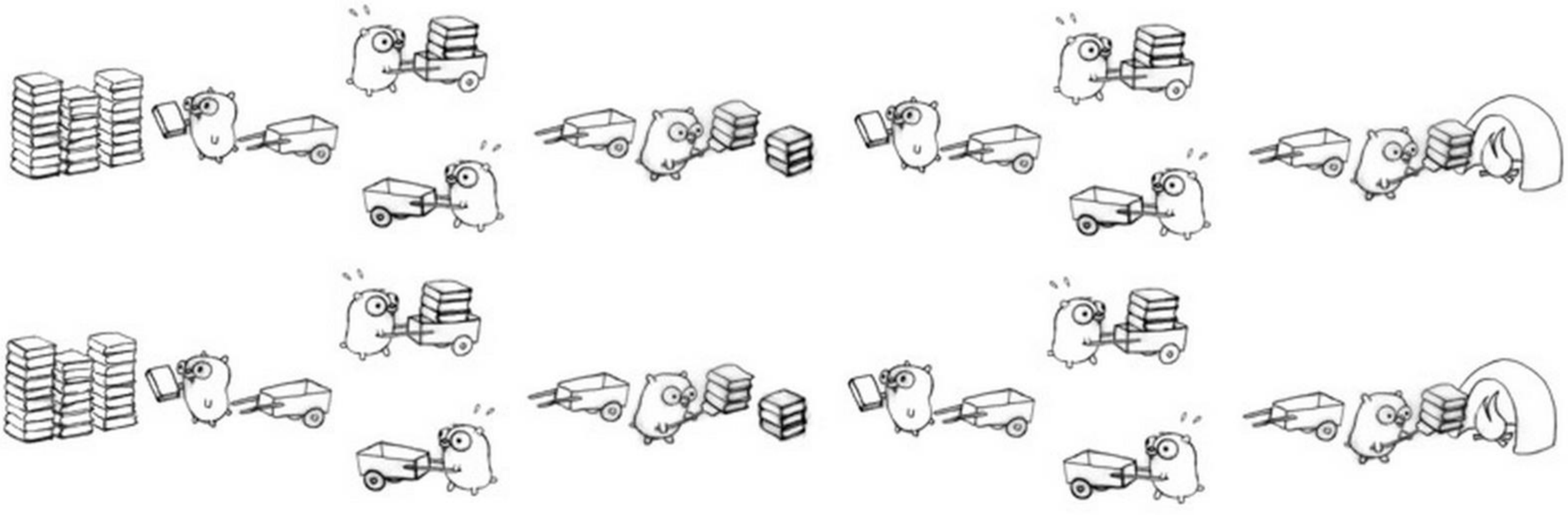


http://talks.golang.org/2012/waza.slide

Full on optimization

Use all our techniques.

Sixteen gophers hard at work!



http://talks.golang.org/2012/waza.slide

Back to computing

In our book transport problem, substitute:

- **book pile** => web content
- **gopher** => CPU
- **cart** => marshaling, rendering, or networking
- **incinerator** => proxy, browser, or other consumer

http://talks.golang.org/2012/waza.slide

Goroutines are not threads

- They're a bit like threads, but they're much cheaper.

- Goroutines are multiplexed onto OS threads as required.
- When a goroutine blocks, that thread blocks but no other goroutine blocks.

http://talks.golang.org/2012/waza.slide

Concurrency: philosophy

Think about the concurrency issues that matter:

Don’t communicate by sharing memory.

Instead, share memory by communicating.

http://talks.golang.org/2011/Real_World_Go.pdf

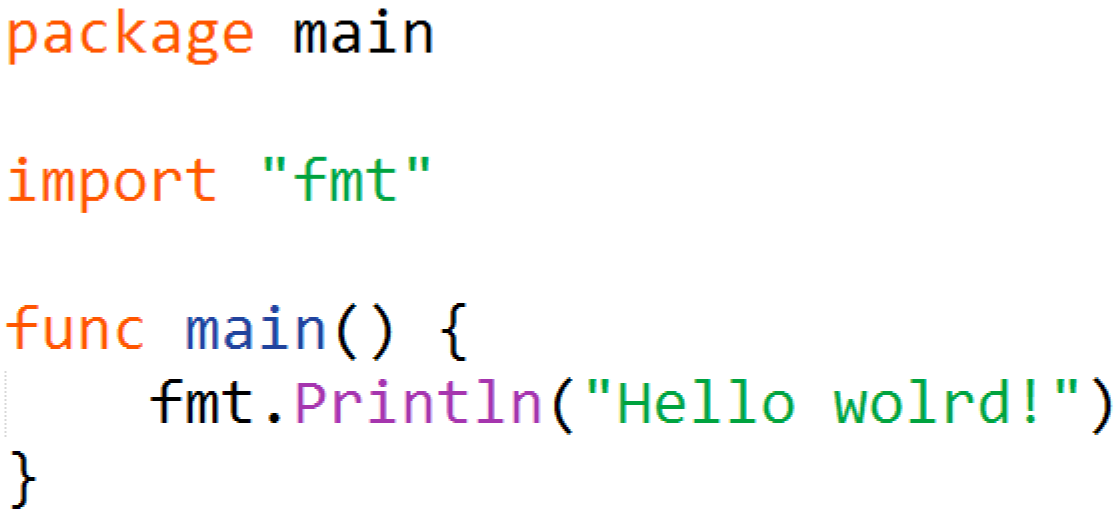
Organizations using Go

- Google
- bit.ly
- CloudFlare
- Canonical
- Heroku
- The BBC
- …

http://en.wikipedia.org/wiki/Go_(programming_language)

The first example

Hello world!

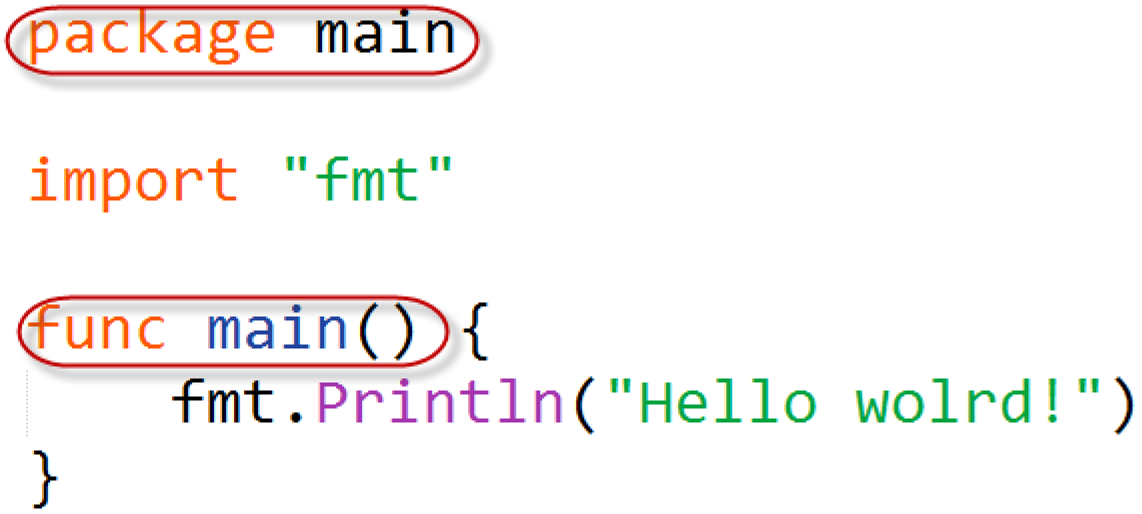


```go
package main

import "fmt"

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

Hello world!

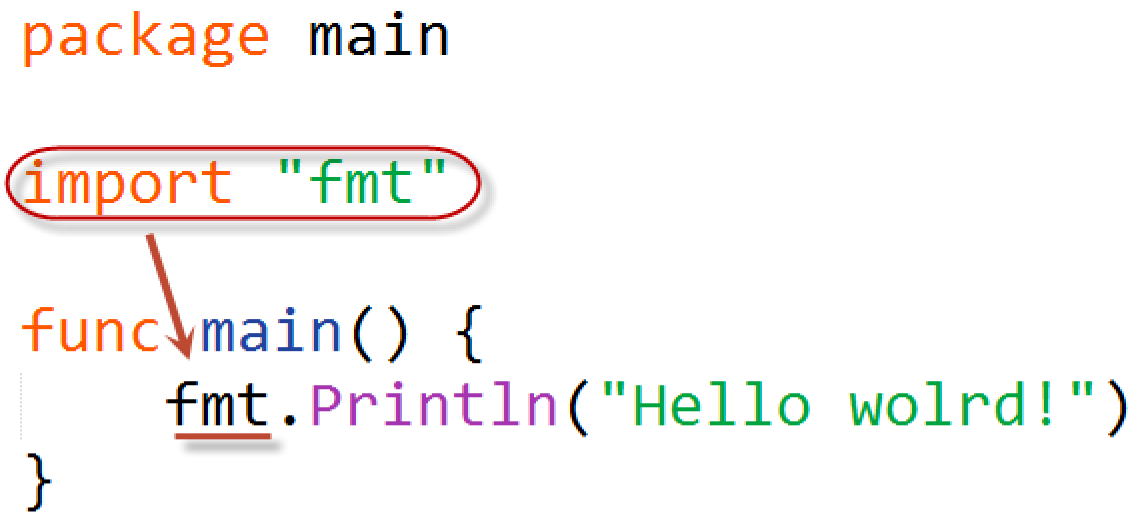


```go
package main

import "fmt"

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

Hello world!



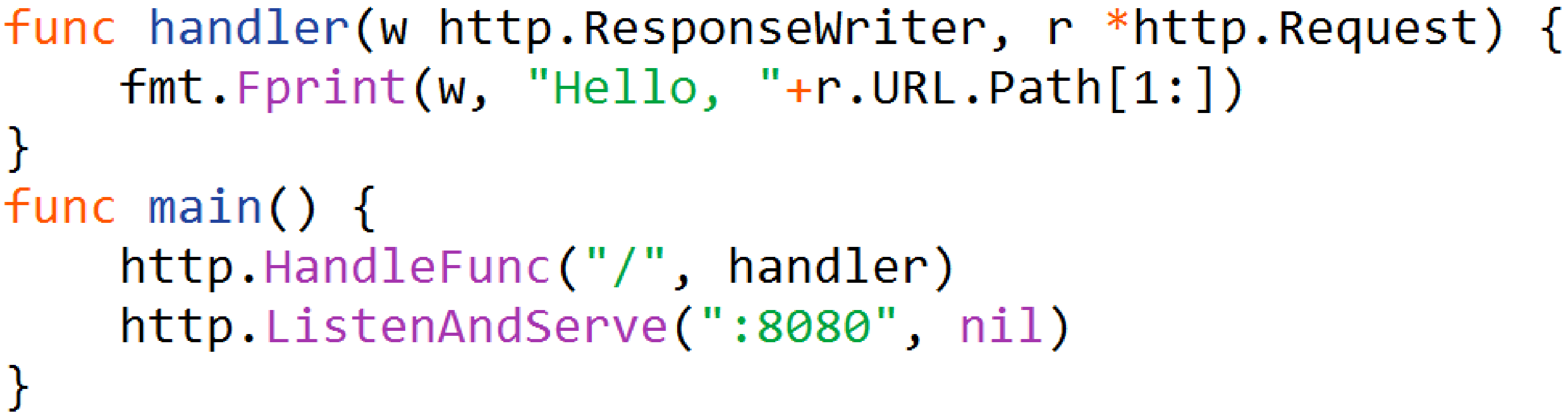
```
package main

import "fmt"

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

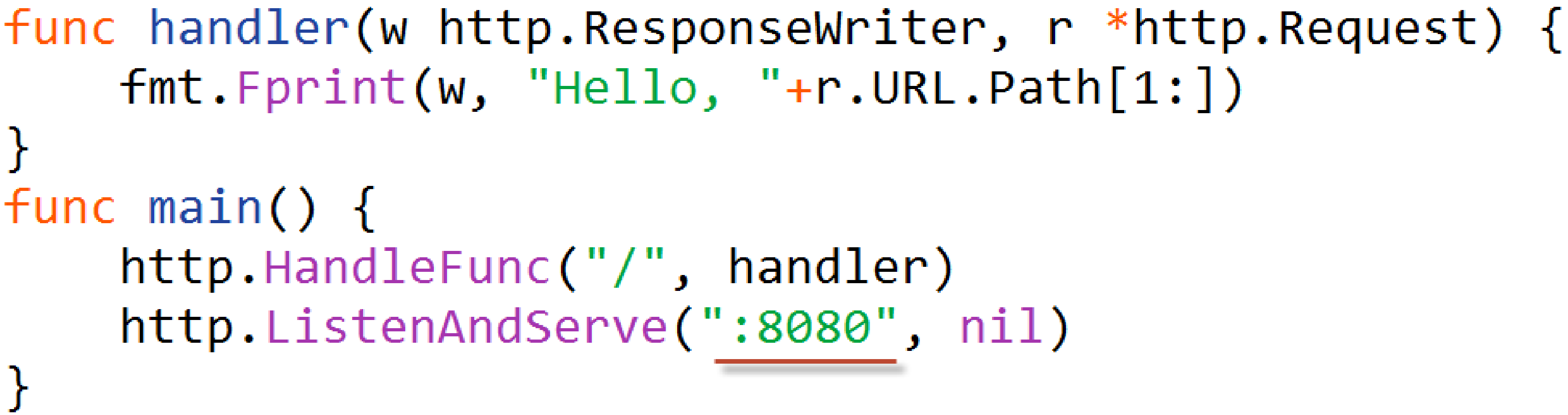
The second example

Hello web server



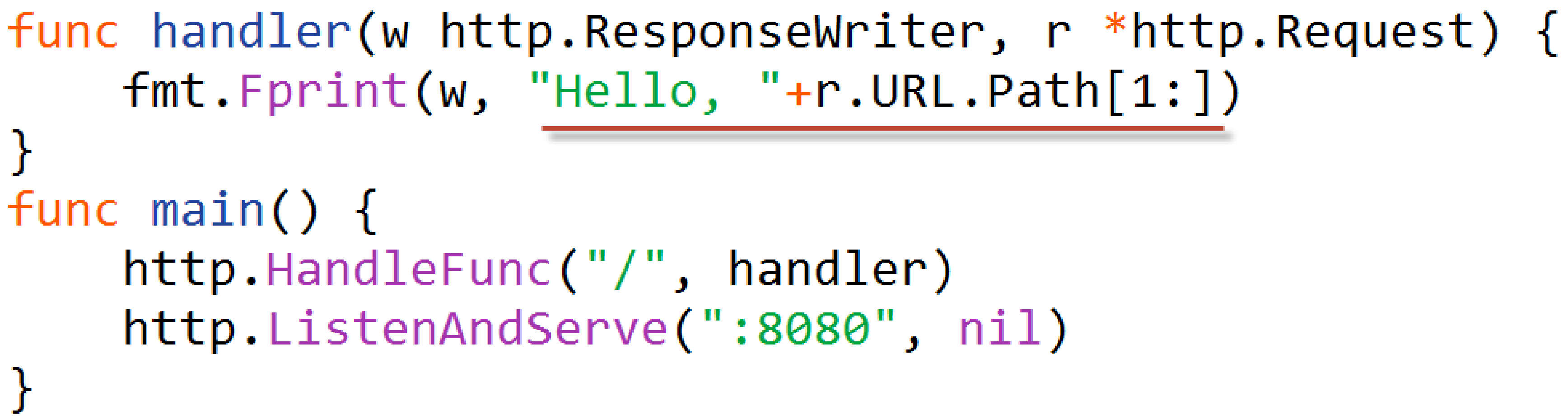
```
func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprint(w, "Hello, "+r.URL.Path[1:])
}
func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

Hello web server



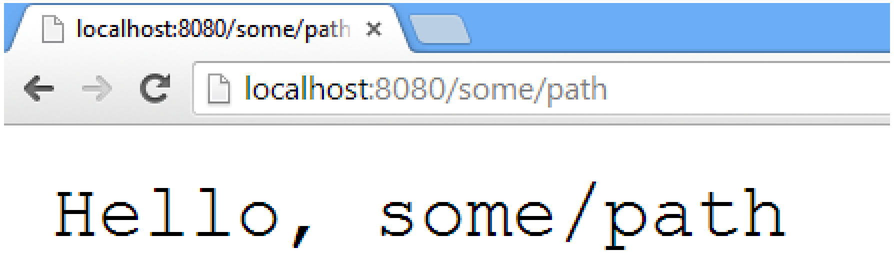
```
func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprint(w, "Hello, "+r.URL.Path[1:])
}
func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

Hello web server



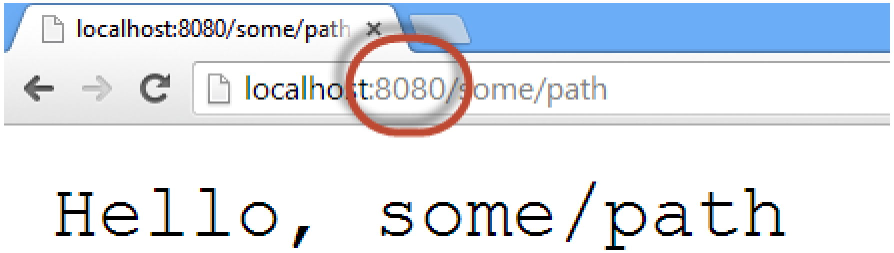
```go
func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprint(w, "Hello, "+r.URL.Path[1:])
}
func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

Hello web server



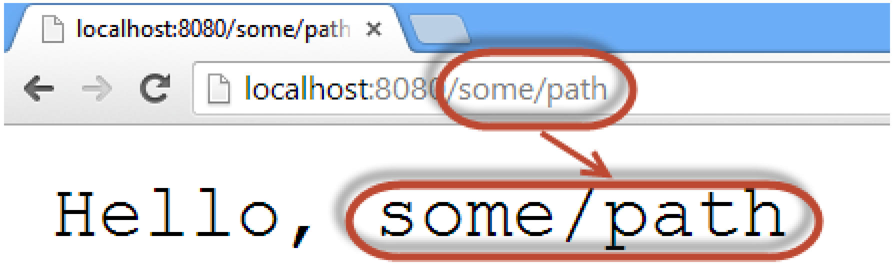
```
Hello, some/path
```

Hello web server



```
Hello, some/path
```

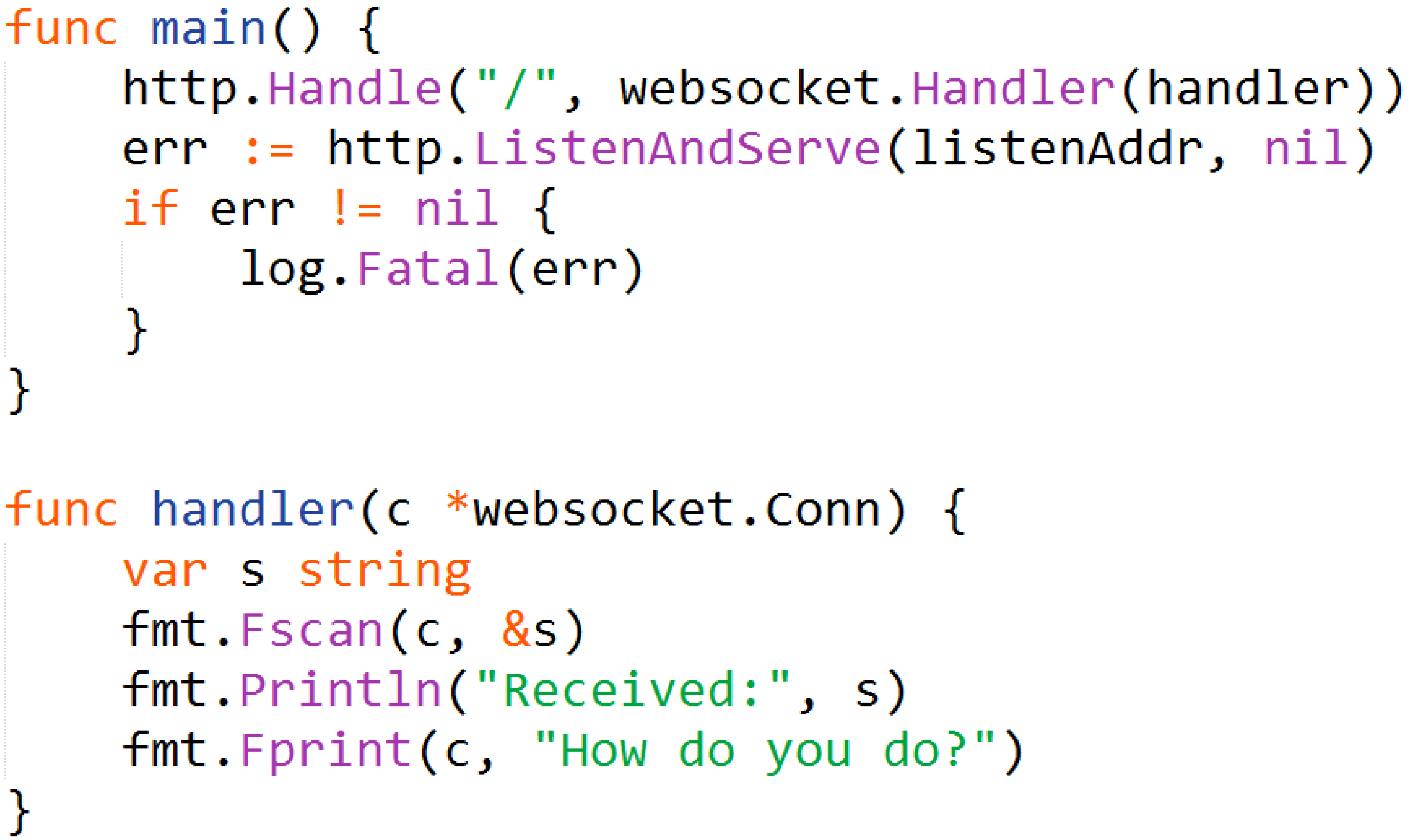
Hello web server



localhost:8080/some/path

```
Hello, some/path
```

Hello WebSocket

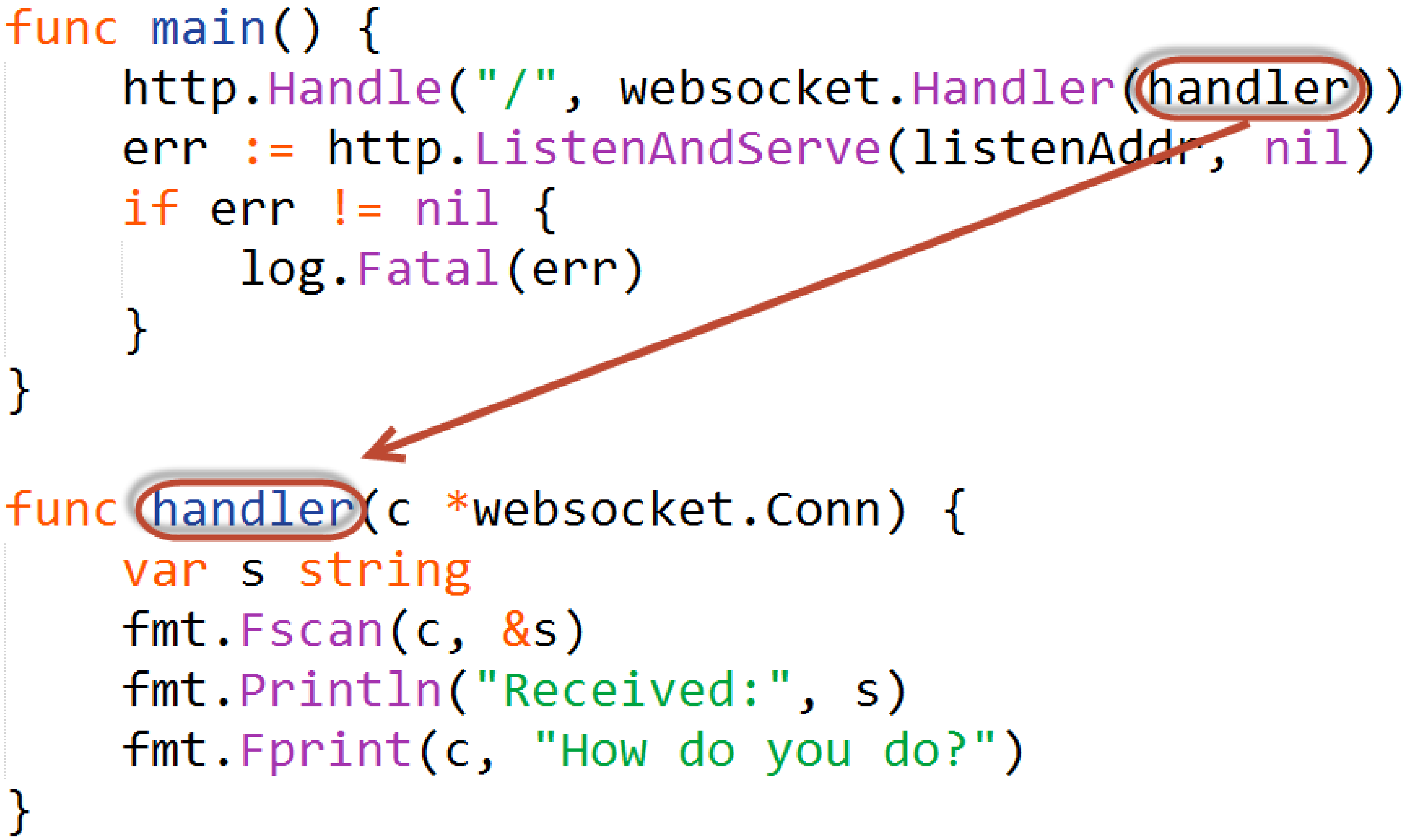


```
func main() {
    http.Handle("/", websocket.Handler(handler))
    err := http.ListenAndServe(listenAddr, nil)
    if err != nil {
        log.Fatal(err)
    }
}

func handler(c *websocket.Conn) {
    var s string
    fmt.Fscan(c, &s)
    fmt.Println("Received:", s)
    fmt.Fprint(c, "How do you do?")
}
```

http://talks.golang.org/2012/chat.slide#29

Hello WebSocket

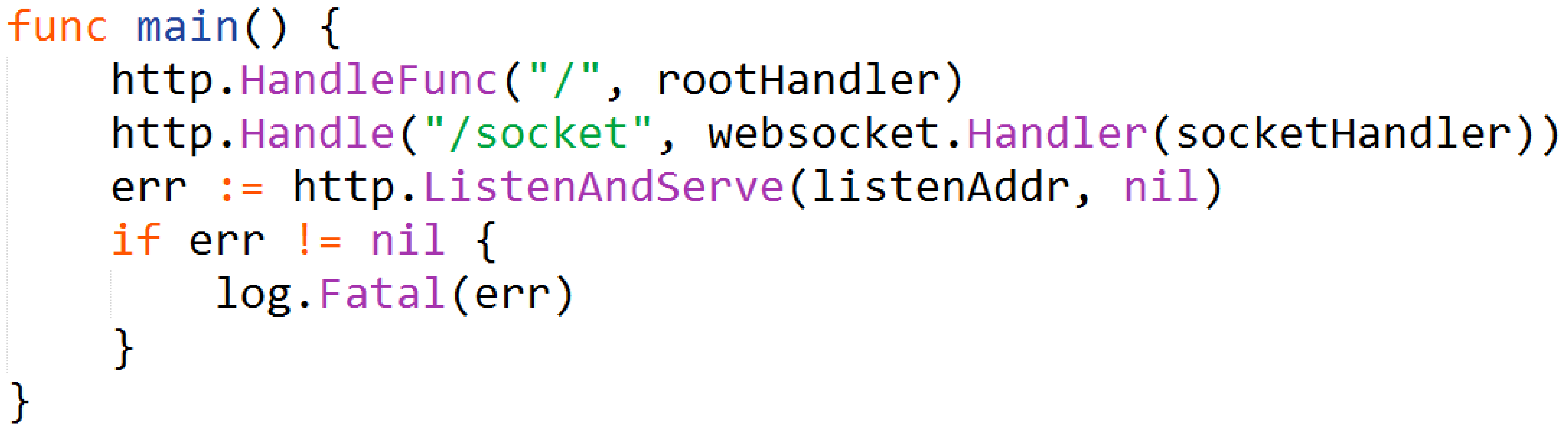


```
func main() {
    http.Handle("/", websocket.Handler(handler))
    err := http.ListenAndServe(listenAddr, nil)
    if err != nil {
        log.Fatal(err)
    }
}

func handler(c *websocket.Conn) {
    var s string
    fmt.Fscan(c, &s)
    fmt.Println("Received:", s)
    fmt.Fprint(c, "How do you do?")
}
```

http://talks.golang.org/2012/chat.slide#29

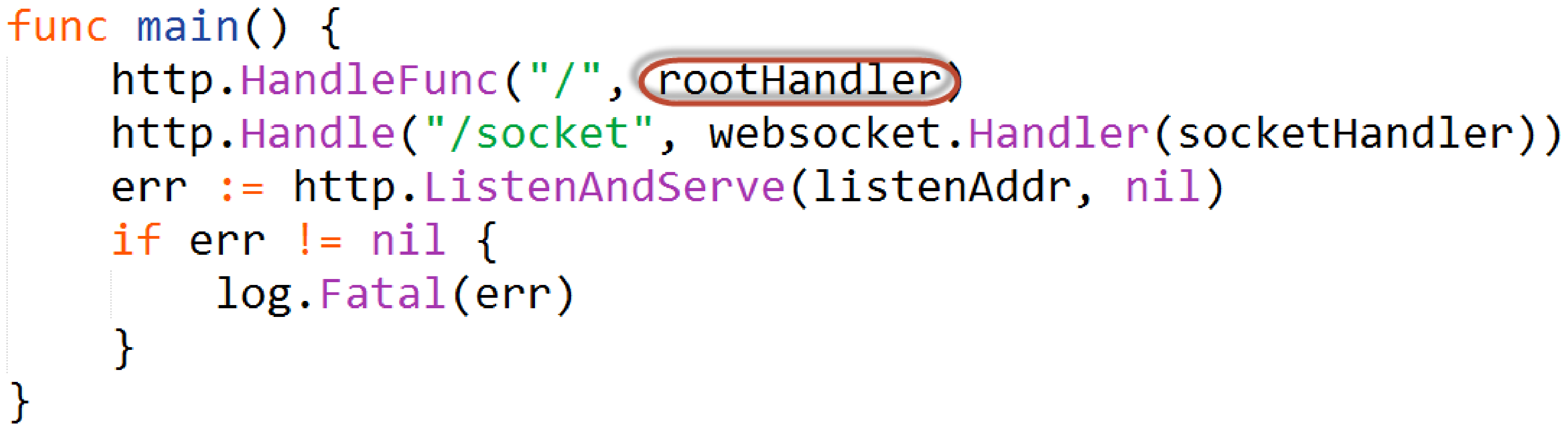
Using the **http** and **websocket** packages



```go
func main() {
    http.HandleFunc("/", rootHandler)
    http.Handle("/socket", websocket.Handler(socketHandler))
    err := http.ListenAndServe(listenAddr, nil)
    if err != nil {
        log.Fatal(err)
    }
}
```

http://talks.golang.org/2012/chat.slide#29

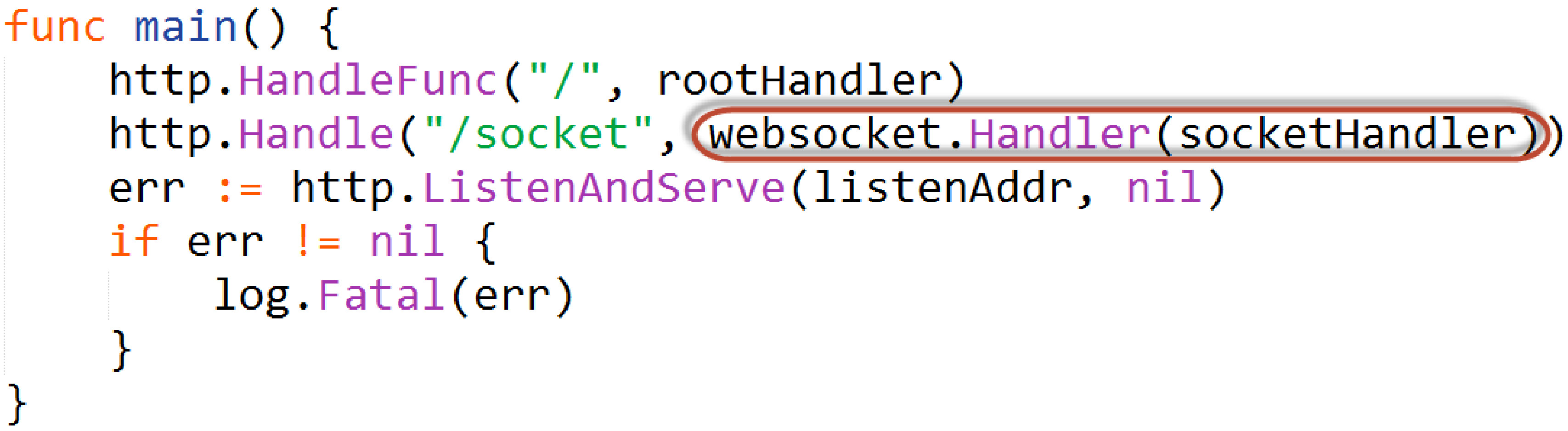
Using the **http** and **websocket** packages



```go
func main() {
	http.HandleFunc("/", rootHandler)
	http.Handle("/socket", websocket.Handler(socketHandler))
	err := http.ListenAndServe(listenAddr, nil)
	if err != nil {
		log.Fatal(err)
	}
}
```

http://talks.golang.org/2012/chat.slide#29

Using the **http** and **websocket** packages



```
func main() {
    http.HandleFunc("/", rootHandler)
    http.Handle("/socket", websocket.Handler(socketHandler))
    err := http.ListenAndServe(listenAddr, nil)
    if err != nil {
        log.Fatal(err)
    }
}
```

http://talks.golang.org/2012/chat.slide#29

The third example

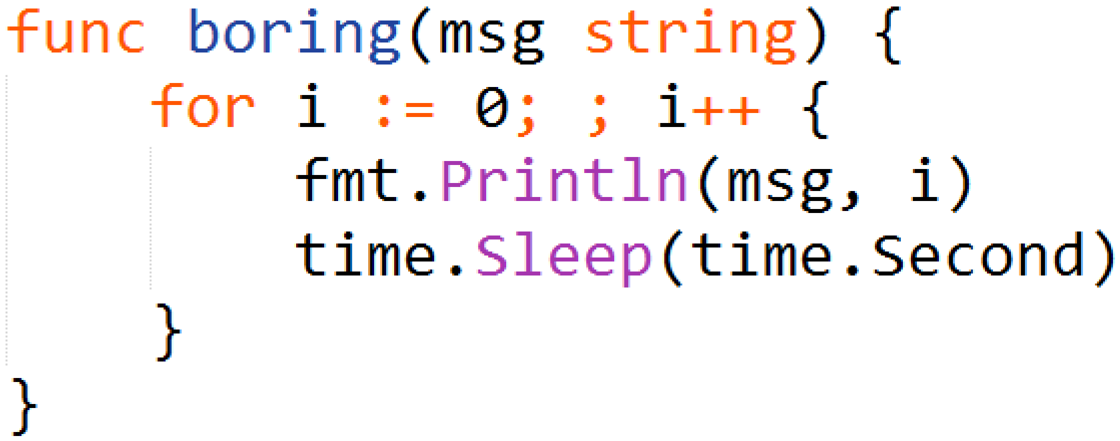
Goroutines

Goroutines are lightweight threads that are managed by the Go runtime.

To run a function in a new goroutine, just put "go" before the function call.

http://talks.golang.org/2012/chat.slide#14

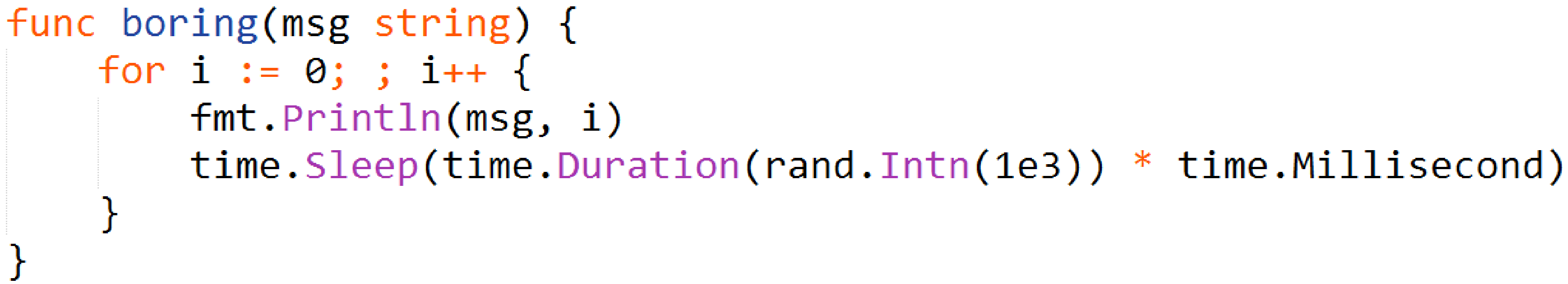
A boring function



```
func boring(msg string) {
    for i := 0; ; i++ {
        fmt.Println(msg, i)
        time.Sleep(time.Second)
    }
}
```

http://talks.golang.org/2012/concurrency.slide

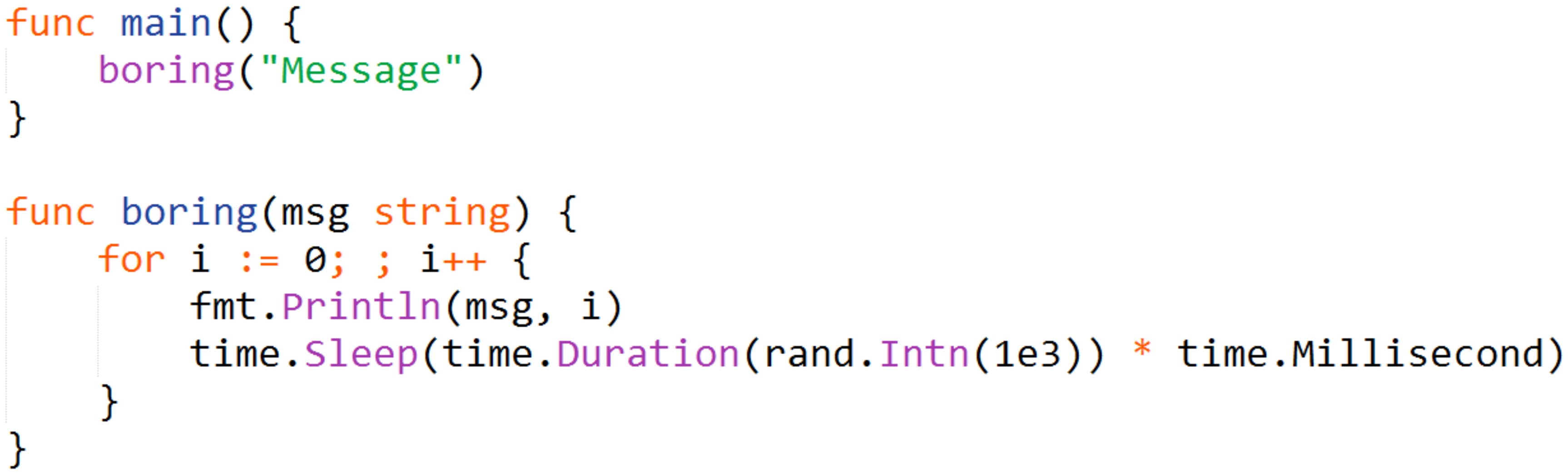
Slightly less boring



```
func boring(msg string) {
    for i := 0; ; i++ {
        fmt.Println(msg, i)
        time.Sleep(time.Duration(rand.Intn(1e3)) * time.Millisecond)
    }
}
```

http://talks.golang.org/2012/concurrency.slide

Running it

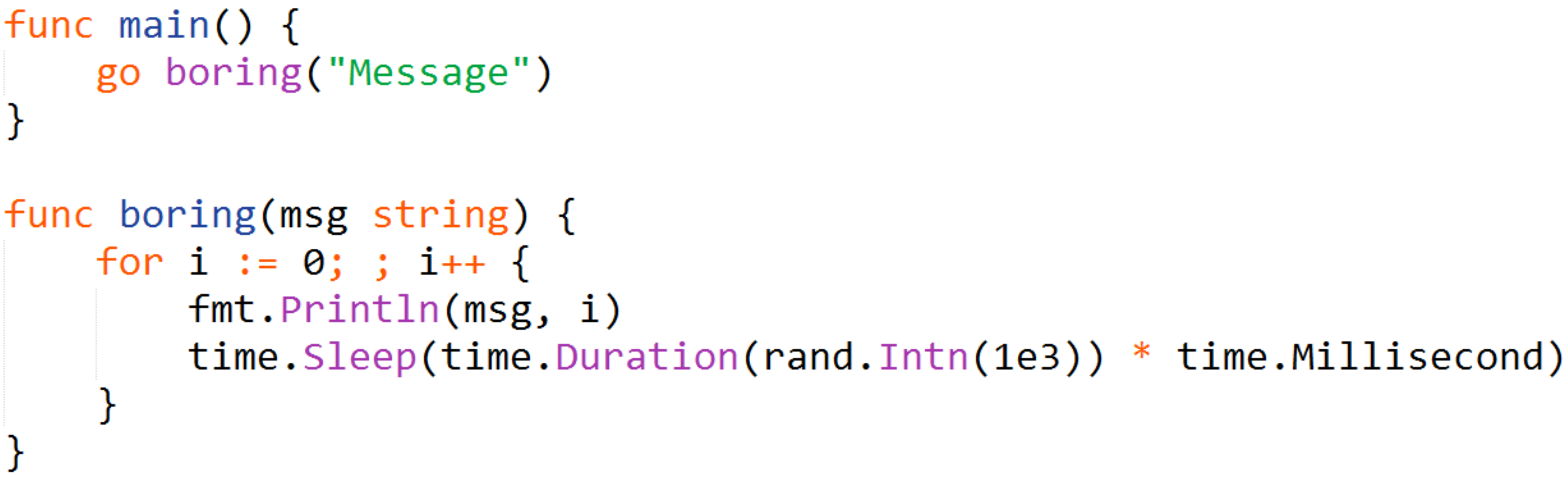


```go
func main() {
    boring("Message")
}

func boring(msg string) {
    for i := 0; ; i++ {
        fmt.Println(msg, i)
        time.Sleep(time.Duration(rand.Intn(1e3)) * time.Millisecond)
    }
}
```

http://talks.golang.org/2012/concurrency.slide

Ignoring it

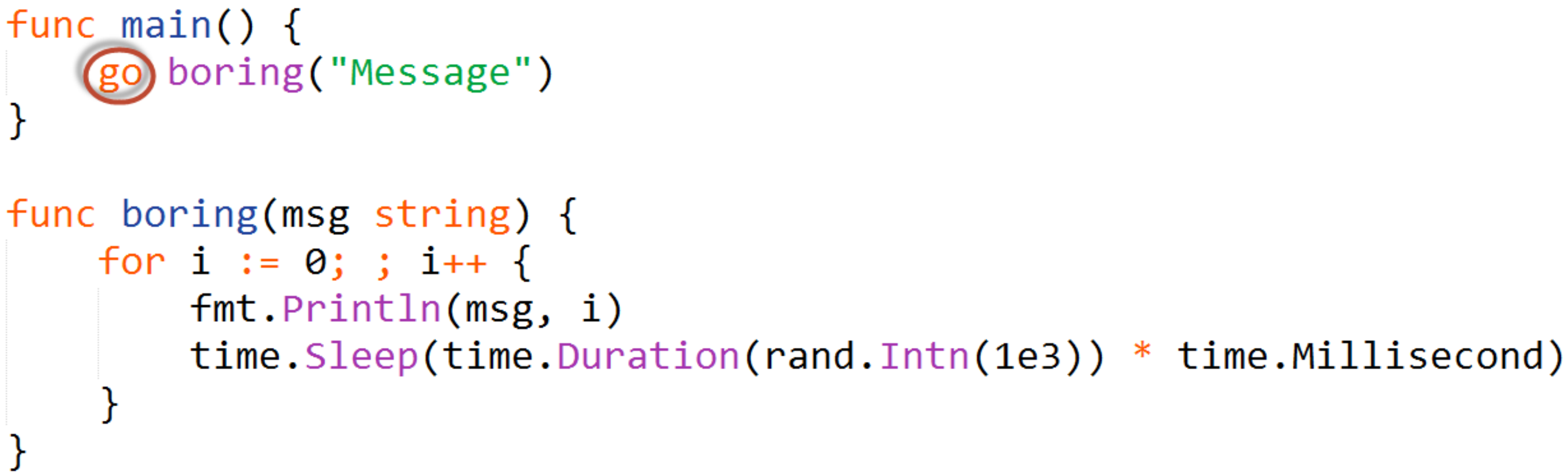


```go
func main() {
	go boring("Message")
}

func boring(msg string) {
	for i := 0; ; i++ {
		fmt.Println(msg, i)
		time.Sleep(time.Duration(rand.Intn(1e3)) * time.Millisecond)
	}
}
```

http://talks.golang.org/2012/concurrency.slide

Ignoring it

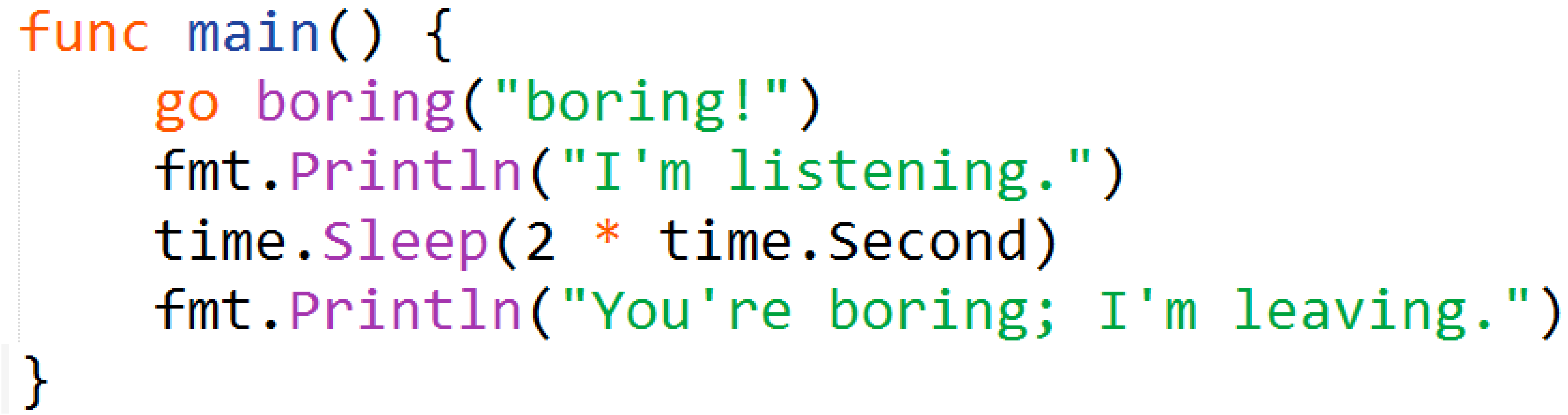


```go
func main() {
    go boring("Message")
}

func boring(msg string) {
    for i := 0; ; i++ {
        fmt.Println(msg, i)
        time.Sleep(time.Duration(rand.Intn(1e3)) * time.Millisecond)
    }
}
```

http://talks.golang.org/2012/concurrency.slide

Ignoring it a little less

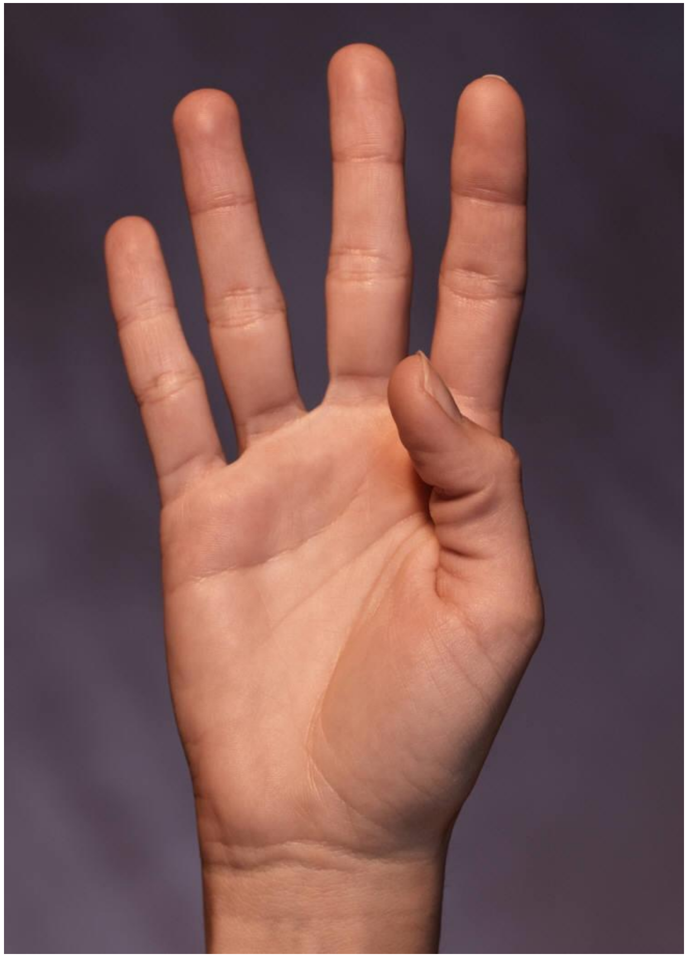


```go
func main() {
	go boring("boring!")
	fmt.Println("I'm listening.")
	time.Sleep(2 * time.Second)
	fmt.Println("You're boring; I'm leaving.")
}
```

```
I'm listening.
 boring! 0
 boring! 1
 boring! 2
 boring! 3
 boring! 4
 boring! 5
 You're boring; I'm leaving.
```

http://talks.golang.org/2012/concurrency.slide

The fourth example



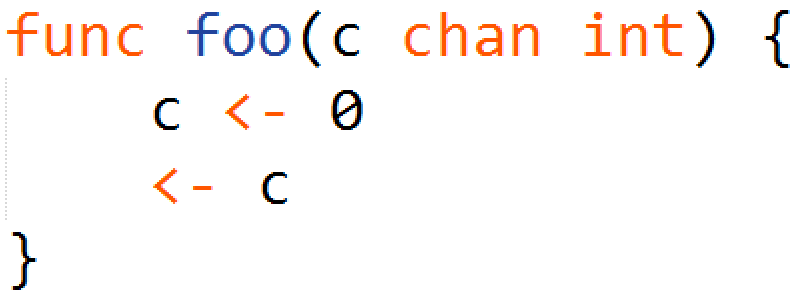
Channels

A channel in Go provides a connection between two goroutines, allowing them to communicate.

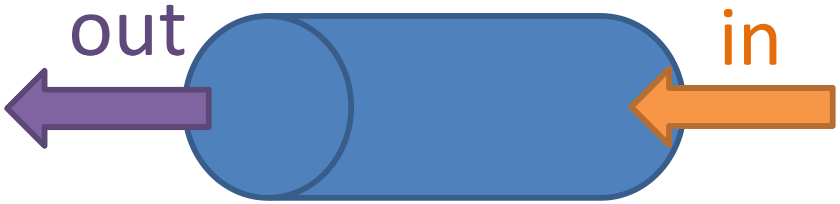
http://talks.golang.org/2012/concurrency.slide

Simple concurency

(use channel)

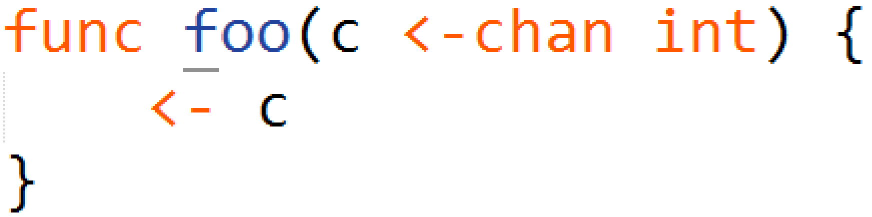


```
func foo(c chan int) {
    c <- 0
    <- c
}
```

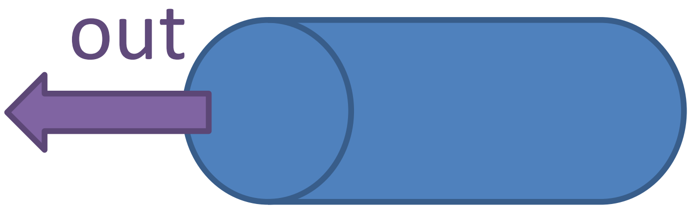


Simple concurency

(use channel)

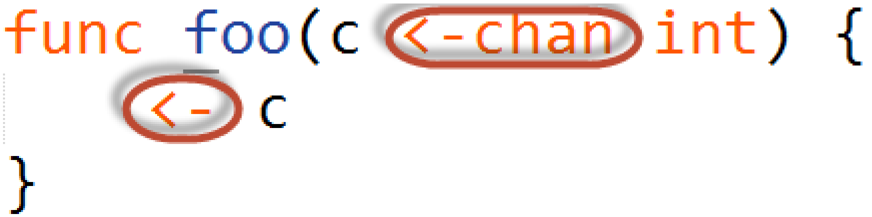


```
func foo(c <-chan int) {
    <- c
}
```

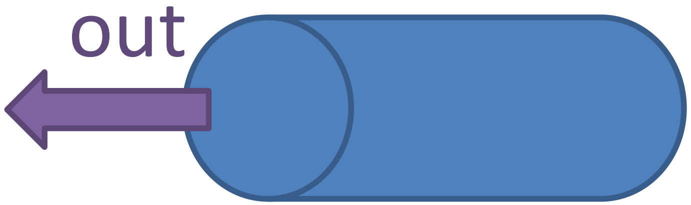


Simple concurency

(use channel)

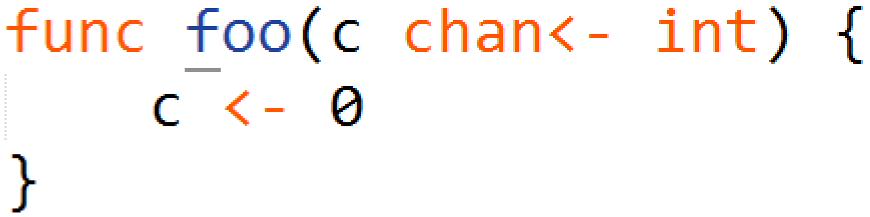


```
func foo(c <-chan int) {
    <- c
}
```

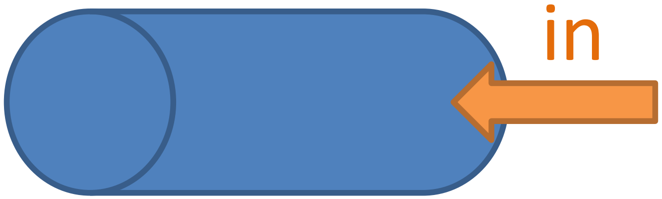


Simple concurency

(use channel)

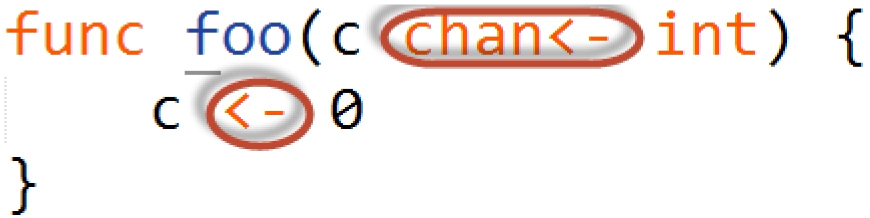


```
func foo(c chan<- int) {
    c <- 0
}
```

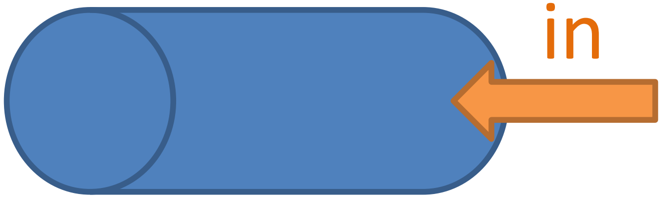


Simple concurency

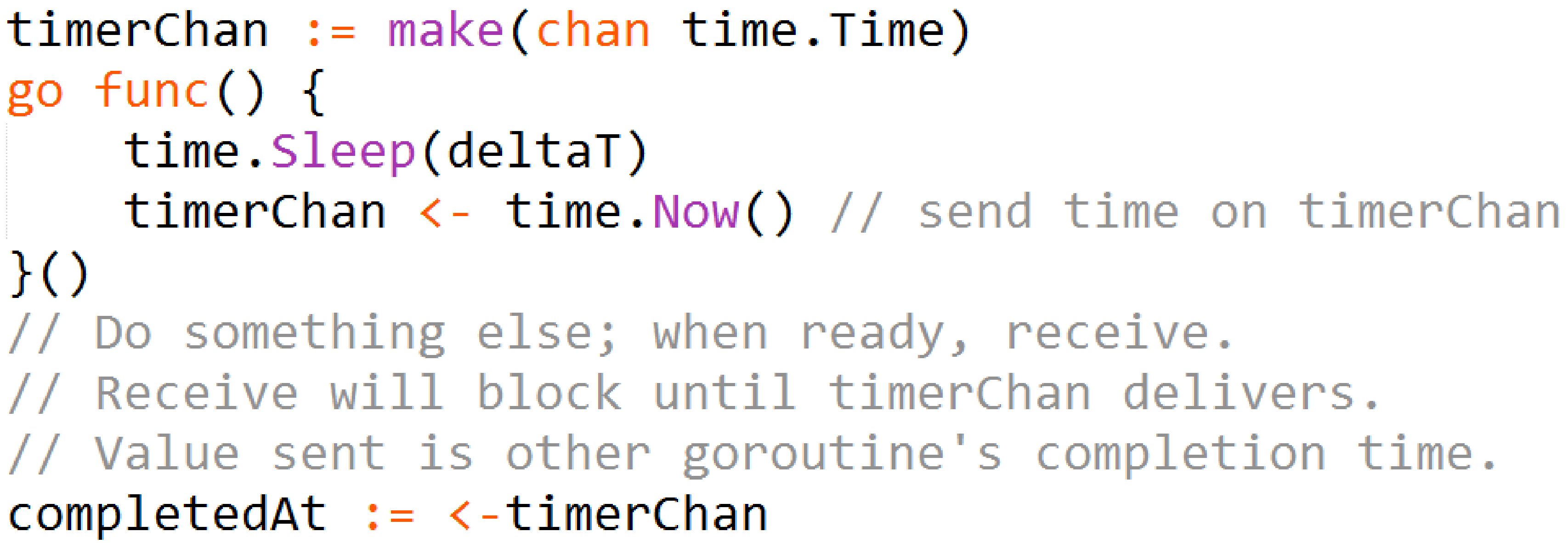
(use channel)



```
func foo(c chan<- int) {
	c <- 0
}
```



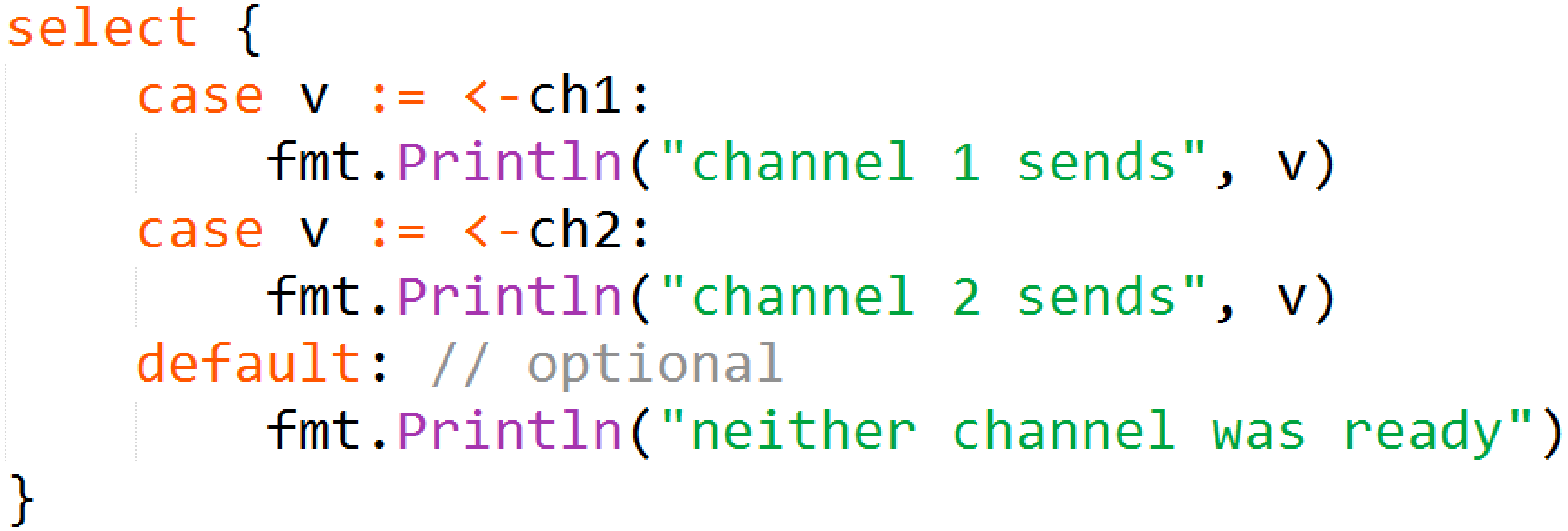
Channels



```go
timerChan := make(chan time.Time)
go func() {
    time.Sleep(deltaT)
    timerChan <- time.Now() // send time on timerChan
}()
// Do something else; when ready, receive.
// Receive will block until timerChan delivers.
// Value sent is other goroutine's completion time.
completedAt := <-timerChan
```

http://talks.golang.org/2012/waza.slide

Select



```go
select {
    case v := <-ch1:
        fmt.Println("channel 1 sends", v)
    case v := <-ch2:
        fmt.Println("channel 2 sends", v)
    default: // optional
        fmt.Println("neither channel was ready")
}
```

http://talks.golang.org/2012/waza.slide

Go really supports concurrency

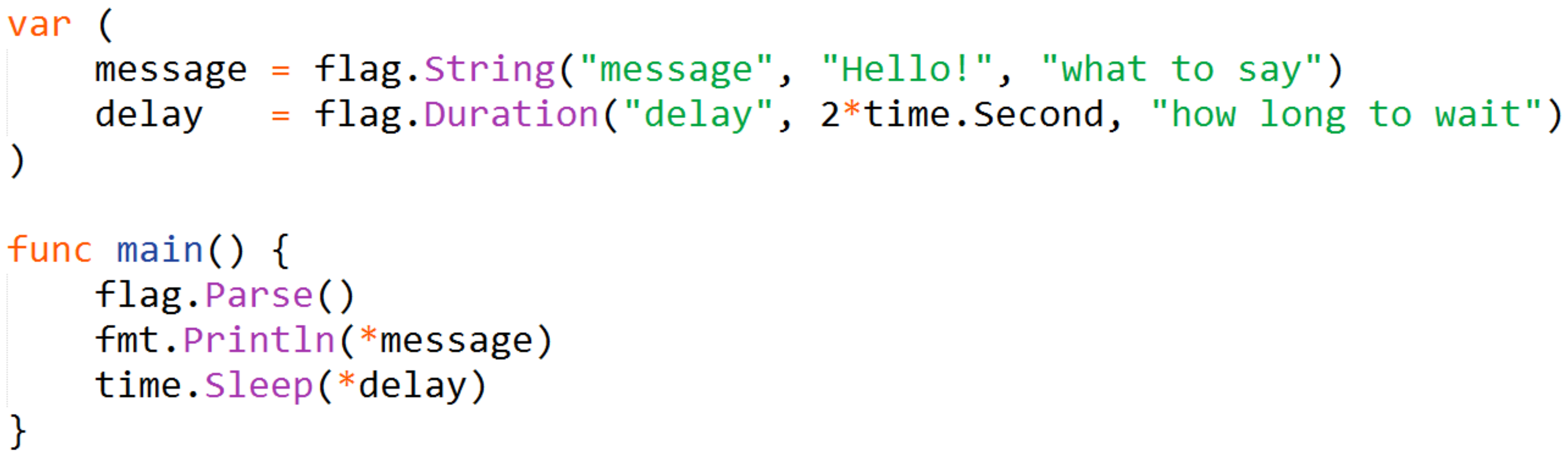
- It's routine to create thousands of goroutines in one program. (Once debugged a program after it had created 1.3 million.)
- Stacks start small, but grow and shrink as required.

Goroutines aren't free, but they're very cheap.

http://talks.golang.org/2012/waza.slide

The fifth example

Flag

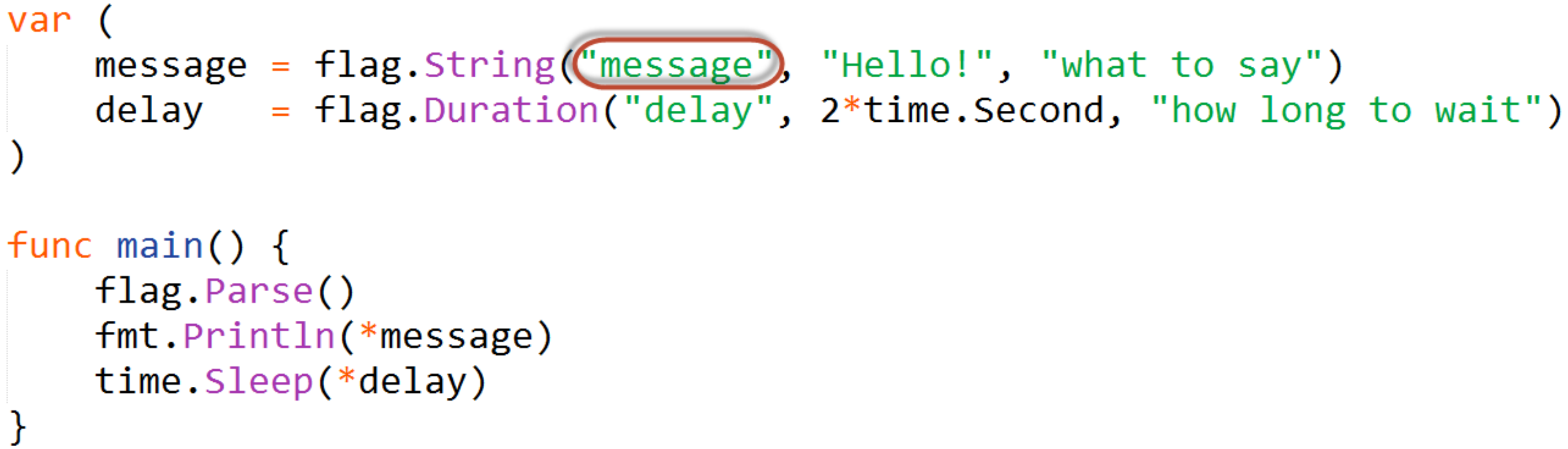


```
var (
    message = flag.String("message", "Hello!", "what to say")
    delay   = flag.Duration("delay", 2*time.Second, "how long to wait")
)

func main() {
    flag.Parse()
    fmt.Println(*message)
    time.Sleep(*delay)
}
```

http://talks.golang.org/2012/simple.slide

Flag

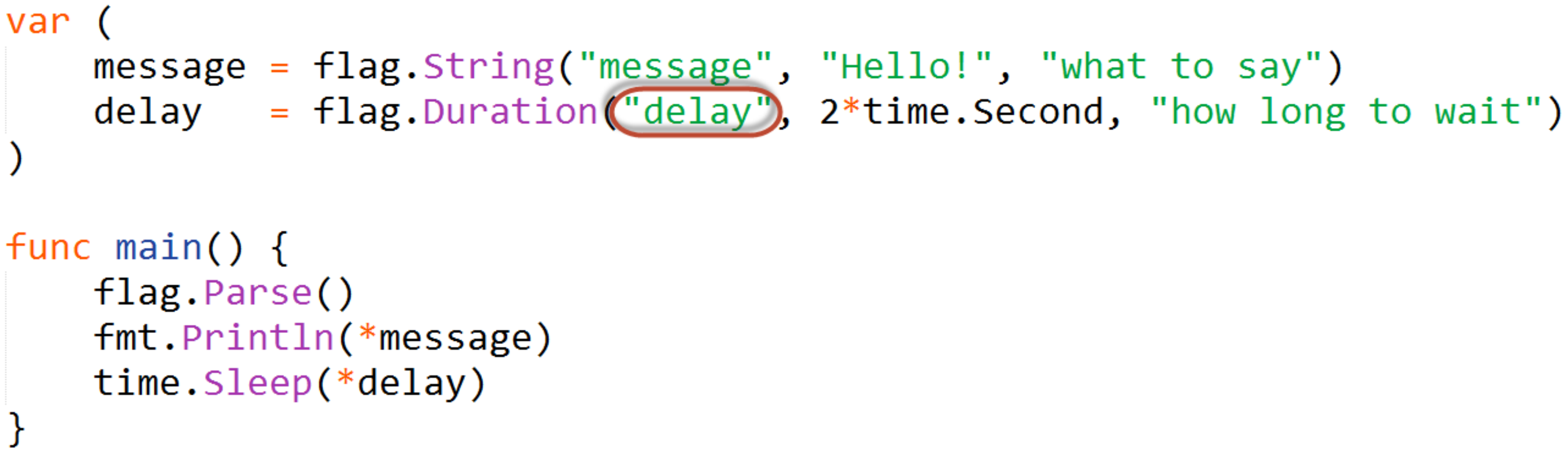


```
var (
    message = flag.String("message", "Hello!", "what to say")
    delay   = flag.Duration("delay", 2*time.Second, "how long to wait")
)

func main() {
    flag.Parse()
    fmt.Println(*message)
    time.Sleep(*delay)
}
```

http://talks.golang.org/2012/simple.slide

Flag

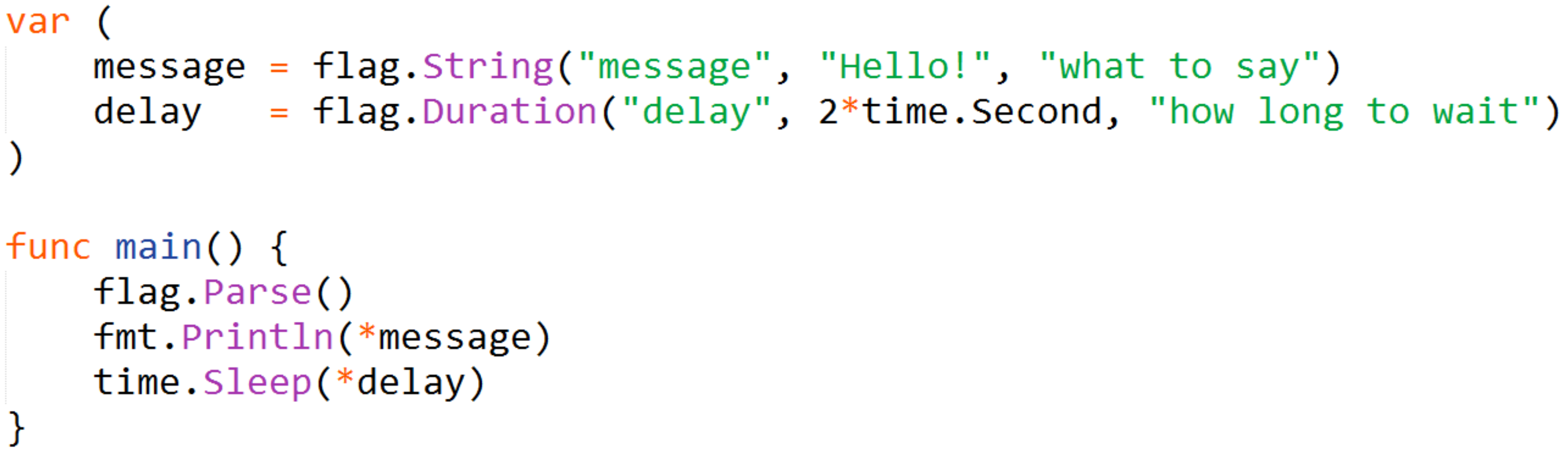


```go
var (
	message = flag.String("message", "Hello!", "what to say")
	delay   = flag.Duration("delay", 2*time.Second, "how long to wait")
)

func main() {
	flag.Parse()
	fmt.Println(*message)
	time.Sleep(*delay)
}
```

http://talks.golang.org/2012/simple.slide

Flag



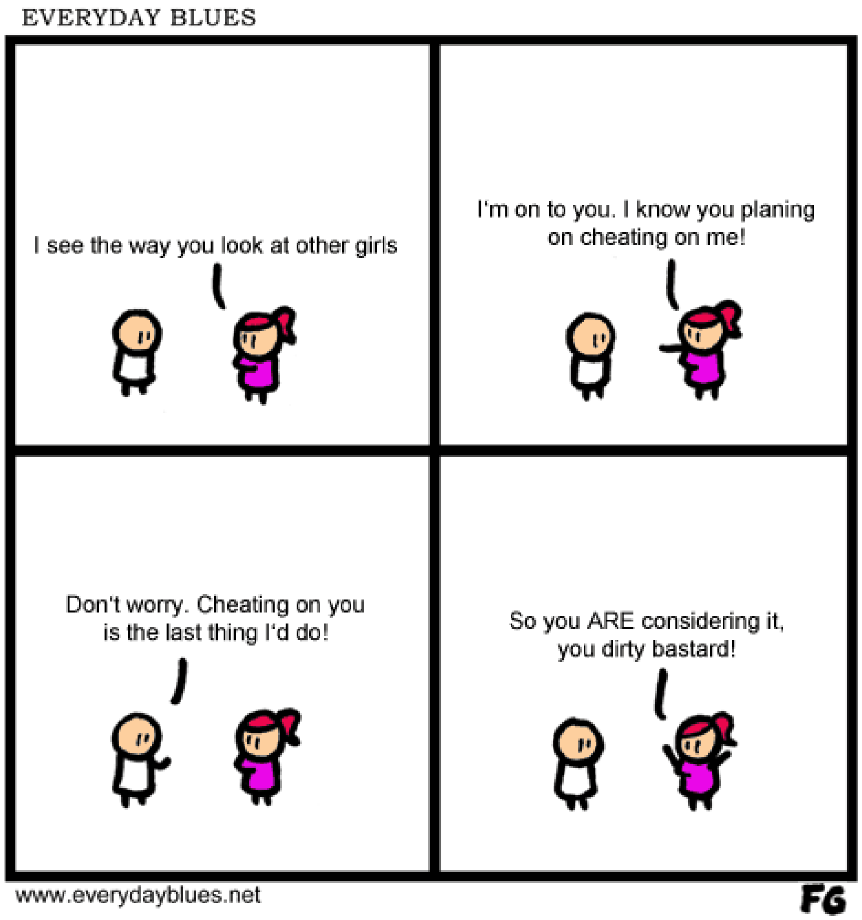
```go
var (
	message = flag.String("message", "Hello!", "what to say")
	delay   = flag.Duration("delay", 2*time.Second, "how long to wait")
)

func main() {
	flag.Parse()
	fmt.Println(*message)
	time.Sleep(*delay)
}
```

go run flag.go –message **Another** –delay **10s**

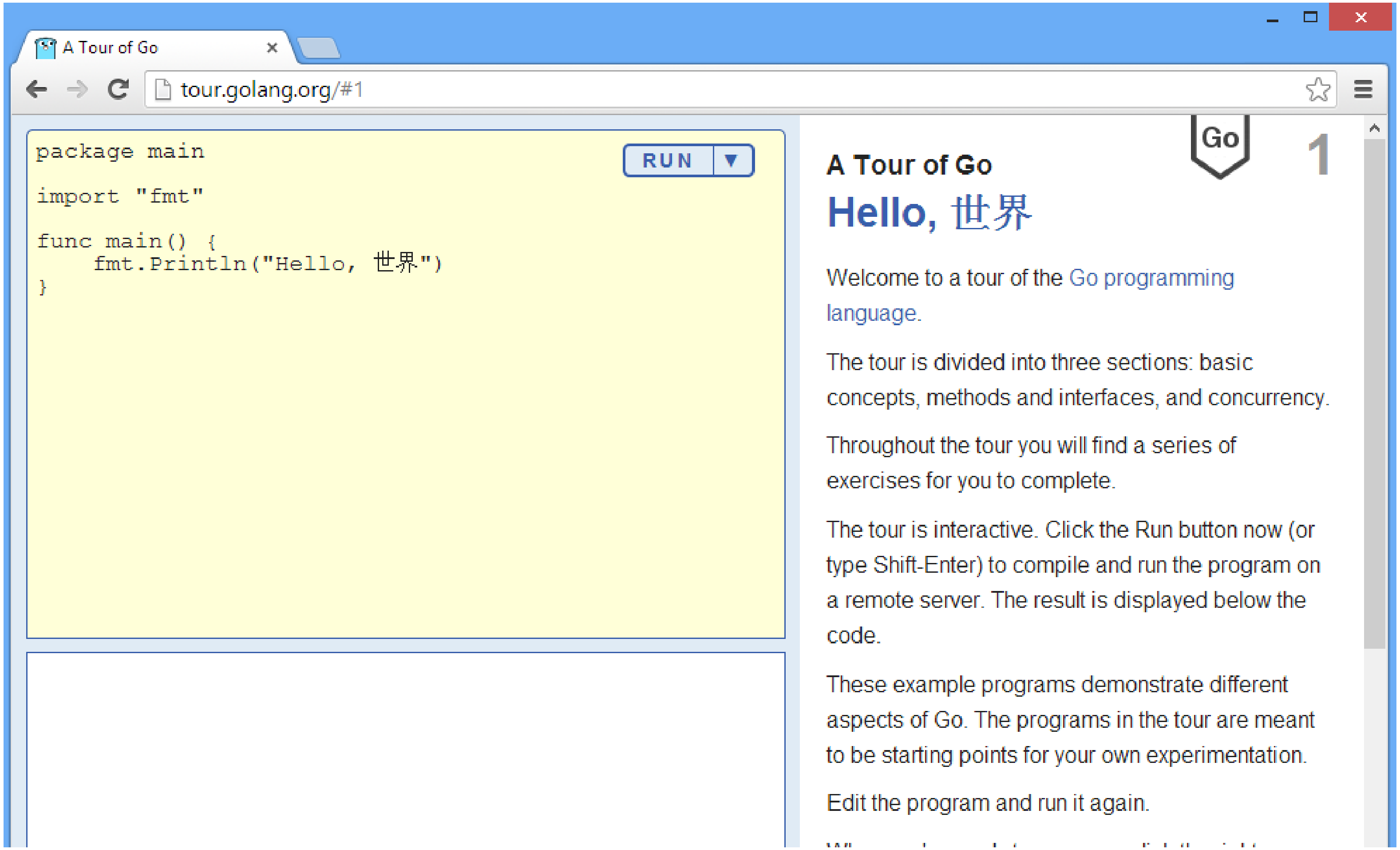
http://talks.golang.org/2012/simple.slide

Last but not least


EVERYDAY BLUES
I see the way you look at other girls
I'm on to you. I know you planing on cheating on me!
Don't worry. Cheating on you is the last thing I'd do!
So you ARE considering it, you dirty bastard!
www.everydayblues.net
FG

http://everydayblues.everydayblues.net/2011/04/27/last-but-not-least/

tour.golang.org



```
package main

import "fmt"

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

A Tour of Go

Hello, 世界

Welcome to a tour of the Go programming language.

The tour is divided into three sections: basic concepts, methods and interfaces, and concurrency.

Throughout the tour you will find a series of exercises for you to complete.

The tour is interactive. Click the Run button now (or type Shift-Enter) to compile and run the program on a remote server. The result is displayed below the code.

These example programs demonstrate different aspects of Go. The programs in the tour are meant to be starting points for your own experimentation.

Edit the program and run it again.

In summary

- Go was designed by and for people who write—and read and debug and **maintain—large software systems**.
- Go's purpose is not research into programming language design.
- Go's purpose is to make its designers' programming lives better.

http://talks.golang.org/2012/splash.slide

Examples

All examples of this presentation
(and even more) are available at

https://**github**.com/**slon1024/intro_to_go**

Resources

- **Effective Go** *(golang.org/doc/effective_go.html).*
- **An Introduction to Programming in Go** by *Caleb Doxsey (golang-book.com)*
- **Learning Go** by *Miek Gieben (miek.nl/files/go)*
- **Programming in Go: Creating Applications for the 21st Century (Developer's Library)** *by Mark Summerfield*
- **The Way To Go: A Thorough Introduction To The Go Programming Language** by *Ivo Balbaert*
- **The Go Programming Language** *(http://www.youtube.com/watch?v=rKnDgT73v8s)*
- **Concurrency is not Parallelism** by *Rob Pike*