Go (Golang) for Java Developers

Java.IL Meetup
3 April 2016

Uri Shamay
Lead Developer, Juno

The Go programming language

- Written by Google
- Written in the 21st century :: Design: 2007, Open Source: 2009, Stable Go 1.0: 2012
- 6 Years old
- Influenced by: C, occam, Limbo, Modula, Newsqueak, Oberon, Pascal, Smalltalk
- Current Version: Go 1.6
- The target is networked servers, but it's a great general-purpose language

Continue..

Supported OS ($GOOS) & ARCH ($GOARCH)

- darwin (386, amd64, arm, arm64)
- dragonfly (amd64)
- freebsd (386, amd64, arm)
- linux (386, amd64, arm, arm64, ppc64, ppc64le)
- netbsd (386, amd64, arm)
- openbsd (386, amd64, arm)
- plan9 (386, amd64)
- solaris (amd64)
- windows (386, amd64)

Motivation for Go

Started as an answer to software problems at Google:

- multicore processors
- networked systems
- massive computation clusters
- scale: 10^7 lines of code
- scale: 10^3 programmers
- scale: 10^{6+} machines (design point)

Who uses Go at Google?

Lots of projects. Thousands of Go programmers. Millions of lines of Go code.

Public examples:

- SPDY proxy for Chrome on mobile devices
- Download server for Chrome, ChromeOS, Android SDK, Earth, etc.
- YouTube Vitess MySQL balancer

Who uses Go besides Google?

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

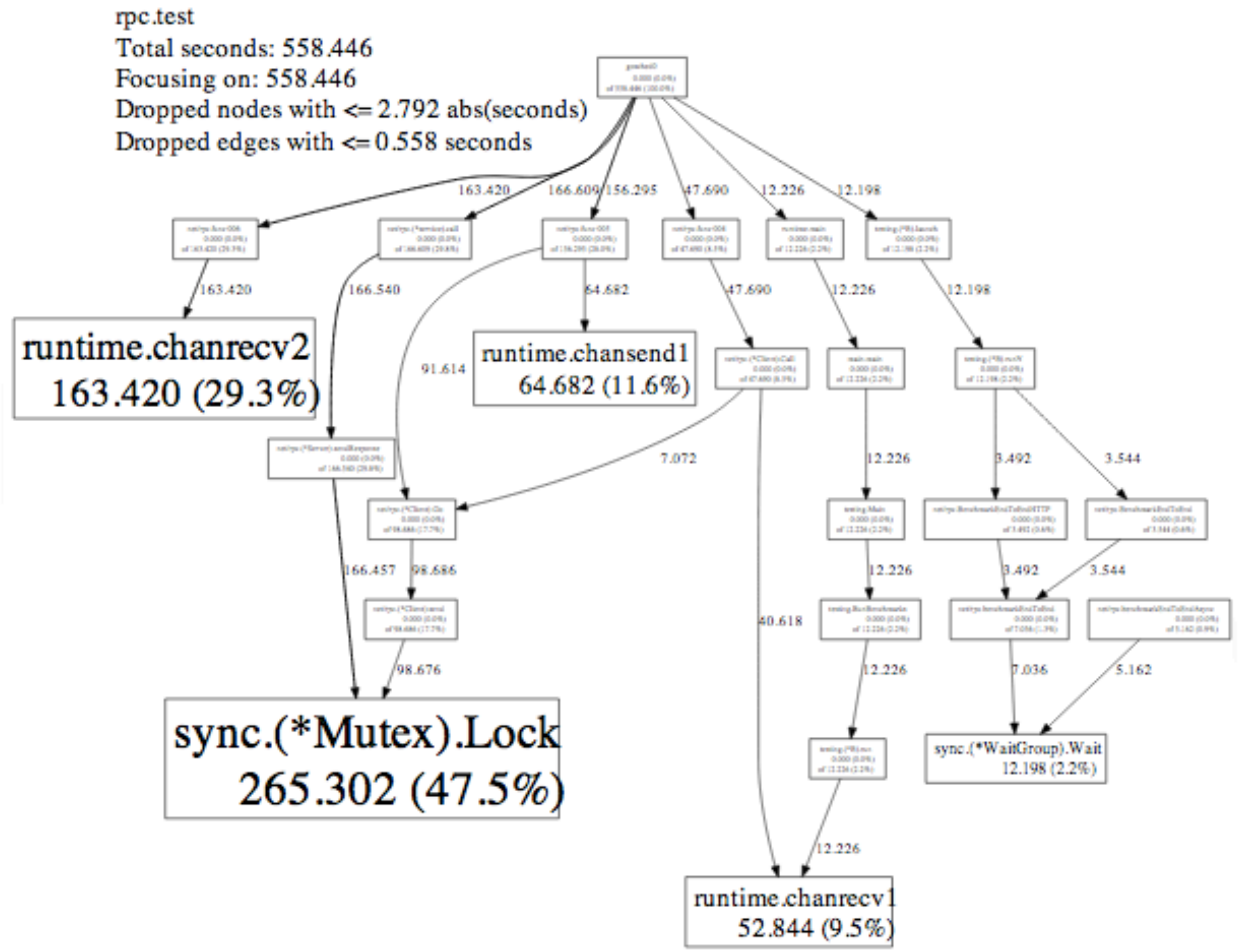
Apcera, Bitbucket, bitly, Canonical, CloudFlare, Core OS, Digital Ocean, Docker, Dropbox, Facebook, Getty Images, GitHub, Heroku, Iron.io, Kubernetes, Medium, MongoDB services, Mozilla services, New York Times, pool.ntp.org, Secret, SmugMug, SoundCloud, Stripe, Square, Thomson Reuters, Tumblr, ...

Great Tools

- go [build] [run]
- go get
- gofmt
- go test [-cover] [-race]
- go vet
- golint
- godoc

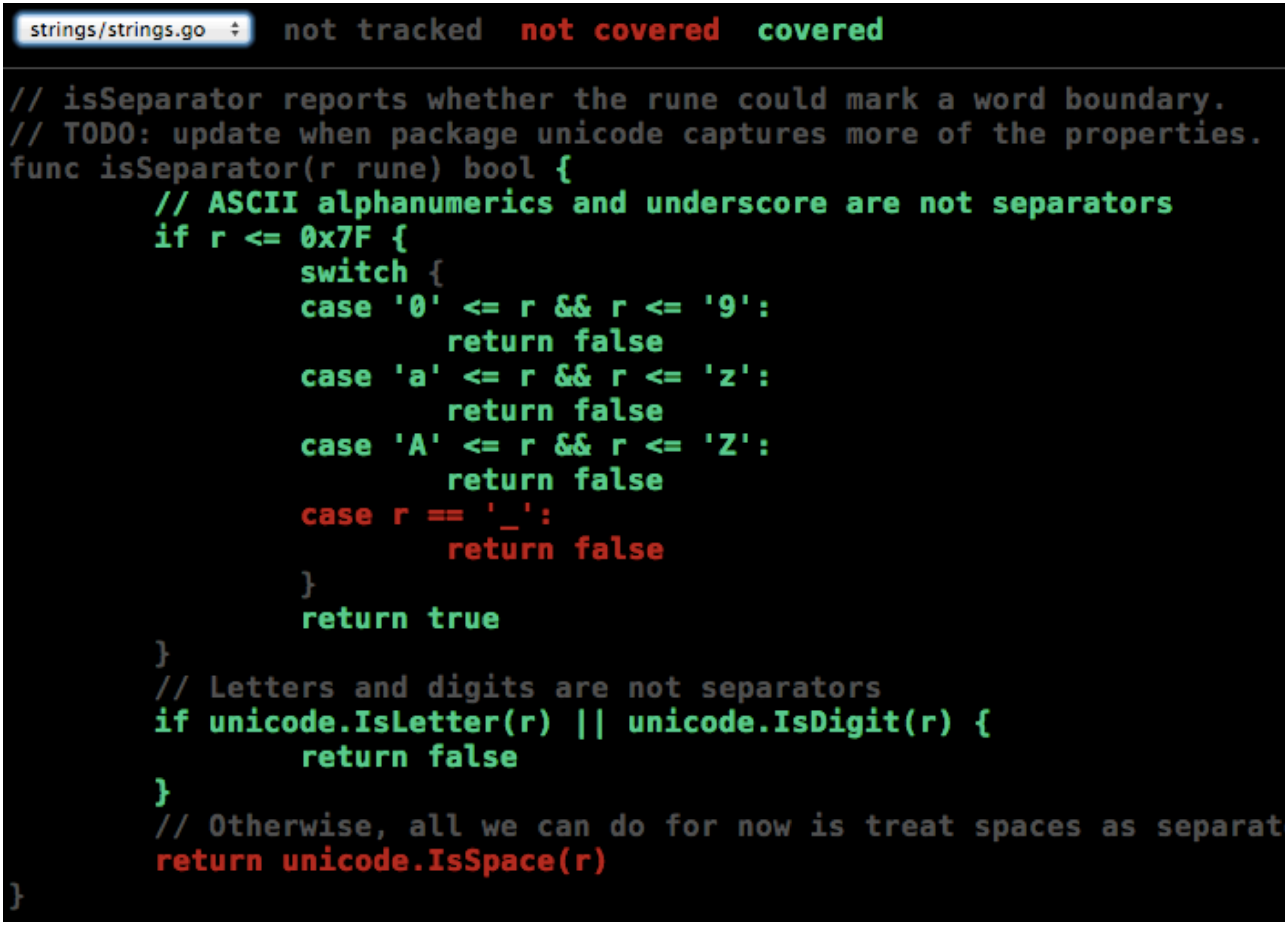
Profiling visualization (web)

```
$ go tool pprof
```



Coverage visualization

```
$ go tool cover -html=cover.out
```



```
// isSeparator reports whether the rune could mark a word boundary.
// TODO: update when package unicode captures more of the properties.
func isSeparator(r rune) bool {
        // ASCII alphanumerics and underscore are not separators
        if r <= 0x7F {
                switch {
                case '0' <= r && r <= '9':
                        return false
                case 'a' <= r && r <= 'z':
                        return false
                case 'A' <= r && r <= 'Z':
                        return false
                case r == '_':
                        return false
                }
                return true
        }
        // Letters and digits are not separators
        if unicode.IsLetter(r) || unicode.IsDigit(r) {
                return false
        }
        // Otherwise, all we can do for now is treat spaces as separat
        return unicode.IsSpace(r)
}
```

Comparing Go and Java

Go and Java have much in common

- Rich standard library
- Cross platform & CPU architecture
- Garbage collected, since Go 1.5 - GC latencies well below 10 milliseconds
- C family (imperative, braces)
- Statically typed. Looks like a script

Continue...

- Memory safe (nil references, runtime bounds checks)
- Variables are always initialized (zero/nil/false)
- Methods
- Interfaces
- Type assertions (`instanceof`)
- Reflection

Go differs from Java in several ways

- Programs compile to machine code. There's no VM.
- Statically linked binaries, since Go 1.5 also dynamically
- Control over memory layout, has pointers, **BUT - no buffer overflows, no pointer arithmetic**
- Function values and lexical closures
- Functions can return multiple values
- Memory efficient: concrete primitives: int & uint **BUT also** int8 & uint8, float32 & float64
- Built-in generic maps and arrays/slices
- Built-in concurrency from first version

Go intentionally leaves out many features

- No classes, has structs
- No constructors
- No direct inheritance. OOP based on interfaces & composition
- No `final`
- No exceptions, has errors
- No annotations
- No user-defined generics
- No public/protected/package/private, visibility is at the package level

Go looks familiar to Java programmers

Main.java

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

hello.go

```go
package main

import "fmt"

func main() {
    hello := "Hello, world."
    fmt.Println(hello)
}
```

Run Run

```
$>go run hello.go
Hello, world.
```

Concurrency

Goroutines

A goroutine is a thread of control within the program, with its own local variables and stack. Cheap, easy to create.

- Concurrency is not parallelism, although it enables parallelism.
- A goroutine runs concurrently (but not necessarily in parallel).
- If you have only one processor, your program can still be concurrent but it cannot be parallel.

You can dispatch millions of goroutines - and OS native threads NOT !

Without goroutines - single thread

```go
func DoGet(url string) {
    resp, err := http.Get(url)
    if err != nil {
        // handle error
    }

    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)

    if err != nil {
        // handle error
    }

    fmt.Printf("url: %v, status code: %v, body len: %v\n", url, resp.StatusCode, len(body))
}

var i int

func f1() {
    for {
        fmt.Println(i)
        DoGet("https://golang.org")
    }
}

func f2() {
    for {
        i++
        DoGet("https://github.com/golang")
    }
}

func main() {
    runtime.GOMAXPROCS(1) // only one OS native thread
    f1()
    f2()
    time.Sleep(3 * time.Second)
}
```

Run Run

Output:

```
$>go run goroutines101.1.go
0
url: https://golang.org, status code: 200, body len: 7856
0
url: https://golang.org, status code: 200, body len: 7856
0
url: https://golang.org, status code: 200, body len: 7856
0
url: https://golang.org, status code: 200, body len: 7856
0
url: https://golang.org, status code: 200, body len: 7856
0
url: https://golang.org, status code: 200, body len: 7856
0
url: https://golang.org, status code: 200, body len: 7856
0
...
```

With cooperative goroutines - single thread

```
func DoGet(url string) {
    resp, err := http.Get(url)
    if err != nil {
        // handle error
    }

    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)

    if err != nil {
        // handle error
    }

    fmt.Printf("url: %v, status code: %v, body len: %v\n", url, resp.
StatusCode, len(body))
}

var i int

func f1() {
    for {
        DoGet("https://golang.org")
        fmt.Println(i)
    }
}

func f2() {
    for {
        i++
        DoGet("https://github.com/golang")
    }
}

func main() {
    runtime.GOMAXPROCS(1) // only one OS native thread
    go f1()
    go f2()
    time.Sleep(3 * time.Second)
}
```

Run Run

Output:

```
$>go run goroutines101.2.go
url: https://golang.org, status code: 200, body len: 7856
1
url: https://golang.org, status code: 200, body len: 7856
1
url: https://golang.org, status code: 200, body len: 7856
1
url: https://github.com/golang, status code: 200, body len: 72600
url: https://golang.org, status code: 200, body len: 7856
2
url: https://github.com/golang, status code: 200, body len: 72600
url: https://golang.org, status code: 200, body len: 7856
3
url: https://github.com/golang, status code: 200, body len: 72600
url: https://golang.org, status code: 200, body len: 7856
4
url: https://github.com/golang, status code: 200, body len: 72600
url: https://golang.org, status code: 200, body len: 7856
5
url: https://github.com/golang, status code: 200, body len: 72600
url: https://golang.org, status code: 200, body len: 7856

...
```

With goroutines - no cooperative - single thread

```
func DoGet(url string) {
    resp, err := http.Get(url)
    if err != nil {
        // handle error
    }

    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)

    if err != nil {
        // handle error
    }

    fmt.Printf("url: %v, status code: %v, body len: %v\n", url, resp.
StatusCode, len(body))
}

var i int

func f1() {
    for {
        DoGet("https://golang.org")
        fmt.Println(i)
    }
}

func f2() {
    for {
        i++
    }
}

func main() {
    runtime.GOMAXPROCS(1) // only one OS native thread
    go f1()
    time.Sleep(time.Second)
    go f2()
    time.Sleep(3 * time.Second)
}
```

Run Run

Output:

```
$>go run goroutines101.3.go
url: https://golang.org, status code: 200, body len: 7856
0
url: https://golang.org, status code: 200, body len: 7856
0
url: https://golang.org, status code: 200, body len: 7856
0

***PROGRAM HALTED***
```

With goroutines - multi threads

```
func DoGet(url string) {
    resp, err := http.Get(url)
    if err != nil {
        // handle error
    }

    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)

    if err != nil {
        // handle error
    }

    fmt.Printf("url: %v, status code: %v, body len: %v\n", url, resp.
StatusCode, len(body))
}

var i int

func f1() {
    for {
        DoGet("https://golang.org")
        fmt.Println(i)
    }
}

func f2() {
    for {
        i++
    }
}

func main() {
    runtime.GOMAXPROCS(runtime.NumCPU()) // use all computer lo
gical CPUs - *DEFAULT*
```

```
    go f1()
    go f2()
    time.Sleep(3 * time.Second)
}
```

Run Run

Output:

```
$>go run goroutines101.3.go
url: https://golang.org, status code: 200, body len: 7856
242475467
url: https://golang.org, status code: 200, body len: 7856
335142144
url: https://golang.org, status code: 200, body len: 7856
434719265
url: https://golang.org, status code: 200, body len: 7856
538521801
url: https://golang.org, status code: 200, body len: 7856
626111870
url: https://golang.org, status code: 200, body len: 7856
721214452
url: https://golang.org, status code: 200, body len: 7856
825607211
url: https://golang.org, status code: 200, body len: 7856
919156439
url: https://golang.org, status code: 200, body len: 7856
1020298919
```

Channels

Channels

Problem: Prime sieve

Problem specification from
Communicating Sequential Processes, by C. A. R. Hoare, 1978

"Problem: To print in ascending order all primes less than 10000. Use an array of processes, SIEVE, in which each process inputs a prime from its predecessor and prints it. The process then inputs an ascending stream of numbers from its predecessor and passes them on to its successor, suppressing any that are multiples of the original prime. "

Solution

Defined in the 1978 CSP paper.

Channels

Channel communication is the main method of synchronization between goroutines.

Don't communicate by sharing memory, share memory by communicating.

Construction:

```
ch1  := make(chan int)     // make unbuffered channel (synchronous). 0
ch2  := make(chan int, 10) // make buffered channel (asynchronous)
ch1 <- 1 // send data to channel
<- ch2   // receive data from channel
```

Select

A `select` statement blocks until communication can proceed.

No locks. No condition variables. No callbacks.

```
    c := make(chan string, 3)

    go func() { c <- DoGet("https://golang.org") }()
    go func() { c <- DoGet("https://tour.golang.org") }()
    go func() { c <- DoGet("https://blog.golang.org") }()

    timeout := time.After(100 * time.Millisecond)
    for i := 0; i < 3; i++ {
        select {
        case result := <-c:
            results = append(results, result)
        case <-timeout:
            fmt.Println("timed out")
            return
        case <-cancel:
            fmt.Println("canceled")
            return
        }
    }
```

Run Run

Let's build web crawler!

Web Crawler

```
// Don't Try This at Home ;D
// const count int = 1000000
const count int = 1000

func main() {

    var done sync.WaitGroup
    done.Add(count)

    responses := make(chan *MyResponse, count)

    for i := 0; i < count; i++ {
        go DoGet("https://golang.org", &done, responses)
    }

    // will wait till all the goroutines notify Done
    done.Wait()
    close(responses)

    for response := range responses {
        fmt.Println(response)
    }
}

type MyResponse struct {
    Body  string
    Error error
}

func DoGet(url string, done *sync.WaitGroup, reponseChannel chan<- *MyResponse) {
    defer done.Done()

    my := new(MyResponse)
    resp, err := http.Get(url)
    if err != nil {
        my.Error = err
        reponseChannel <- my
        return
    }

    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)

    my.Error = err
    my.Body = string(body)
    reponseChannel <- my
}
```

Run Run

Let's build http upload server!!

You can use it as a Log-Receiver/Images-Upload-Server etc...

```
// global shared counter, used by all goroutines,
// should synchronized - using atomic.Add and not a mutex
var x int64

// request handler - each handler run in a dedicated goroutine
func upload(w http.ResponseWriter, r *http.Request) {
    out, err := os.Create("/tmp/" + strconv.FormatInt(atomic.AddInt64
(&x, 1), 10))

    if err != nil {
        fmt.Println(err)
    }

    // synchronus heaven - get rid of my back caLLbACK HeLL!O_o
    io.Copy(out, r.Body)
}

func main() {
    runtime.GOMAXPROCS(runtime.NumCPU())
    http.HandleFunc("/upload", upload)
    http.ListenAndServe(":8080", nil)
}
```

Run Run

Another Great Tool: Race Detector

```
package main

import "runtime"

//obvious race O_o
var i int

func main() {
    runtime.GOMAXPROCS(runtime.NumCPU()) // use all computer logical CPUs

    go func() {
        for {
            i++
        }
    }()
    go func() {
        for {
            i--
        }
    }()

    for {
    }
}
```

Run Run

Race detector example

```
$ go run -race race.go
==================
WARNING: DATA RACE
Read by goroutine 6:
  main.main.func2()
    /Users/urishamay/sandbox/codemotion2015/race.go:10 +0x30

Previous write by goroutine 5:
  main.main.func1()
    /Users/urishamay/sandbox/codemotion2015/race.go:7 +0x4c

Goroutine 6 (running) created at:
  main.main()
    /Users/urishamay/sandbox/codemotion2015/race.go:11 +0x50

Goroutine 5 (finished) created at:
  main.main()
    /Users/urishamay/sandbox/codemotion2015/race.go:8 +0x38
==================
Found 1 data race(s)
exit status 66
```

Resources

The Go Programming Language

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

A Tour Of Go

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

Go Playground

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

The Go Blog

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

Go wiki

github.com/golang/go/wiki (https://github.com/golang/go/wiki)

Concurrency Is Not Parallelism

golang.org/s/concurrency-is-not-parallelism (http://golang.org/s/concurrency-is-not-parallelism)

Thank you

Uri Shamay
Lead Developer, Juno
shamayuri@gmail.com (mailto:shamayuri@gmail.com)

(#ZgotmplZ) github.com/cmpxchg16 (https://github.com/cmpxchg16)

@cmpxchg16 (http://twitter.com/cmpxchg16)

(#ZgotmplZ) cmpxchg16.me (http://cmpxchg16.me)

Standing on the shoulders of giants