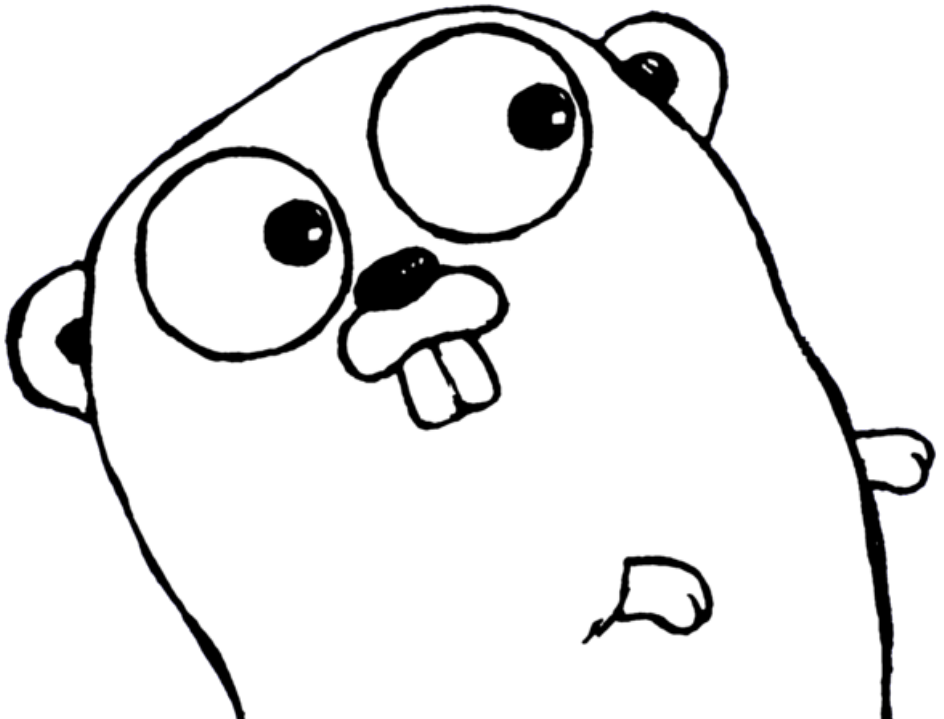
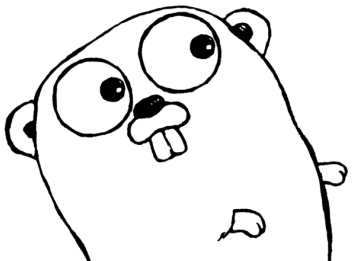
Go

Thomas Richards



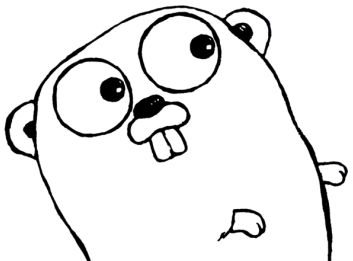
Facts About The Language

- Published in 2009
- Developed at Google (golang.org)
- Systems language
- C-like
- Strong static typing
- Type inference
- Garbage collected
- Concurrency focused
- Has a gopher mascot



First Glance

```go
package main

import "fmt"

func main() {
    var x = fib(10)
    fmt.Println(x)
}

func fib(n int) int {
    if n == 0 || n == 1 {
        return n
    }

    return fib(n - 2) + fib(n - 1)
}
```



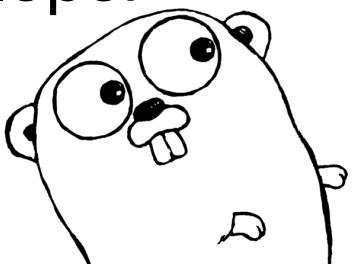
```go
type Animal struct {
    age int
    name string
}

type Cat struct {
    Animal
    length int
}

type AnimalI interface {
    feed(food string)
}

func (c *Cat) lick(a *Animal) {
    fmt.Println("*" + c.name + " licks " + a.name + "*")
}

func (c *Cat) feed(food string) {
    fmt.Println("\"This " + food + " is delicious.\"")
}

func (c *Cat) feedTo(a AnimalI) {
    a.feed(c.name)
}

func main() {
    var myCat *Cat = &Cat{Animal{5, "Kit"}, 40}
    fmt.Println(myCat.name)
    myCat.feedTo(myCat)
    myCat.lick(&myCat.Animal)
}
```

Concurrency

- Main focus of Go
- Time independent events
- Not the same as parallelism

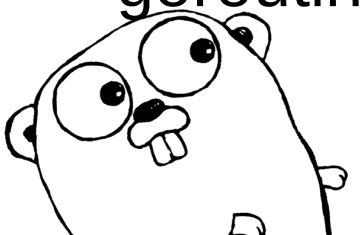
“By concurrency, I mean a way to structure your programs so that they can cope with having to do many things at once and having to deal with many simultaneous sources of input like network connections, and do that in a way that still lets you write a simple and well-structured program.”

—Russ Cox, Go Language Designer and Developer



Goroutines and Channels

- `go` starts a function concurrently
- Channels are a data queue
- `channel <- data`
- `data = <-channel`
- If no data is available, the goroutine “waits” until it is.
- A waiting goroutine is switched for another at runtime.
- Channels can be raced
- A thread pool is used to run goroutines

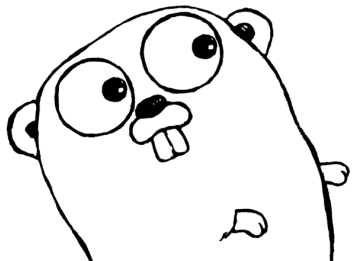


```go
package main

import "fmt"
import "time"

func wait(t int, result *chan int) {
    time.Sleep(time.Second * time.Duration(t))
    *result <- t
}

func output(c *chan int) {
    for {
        fmt.Println(<-*c)
    }
}

func main() {
    result := make(chan int)
    list := []int{1,4,8,3,2,9,6,7,5,0}

    for i:=0; i < 10; i++ {
        go wait(list[i], &result)
    }

    go output(&result)

    //don't immediately end the program
    for {}
}
```

Sleep Sort Example

The Mantra of Go

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



A Language Comparison Between Parallel Programming Features of Go and C

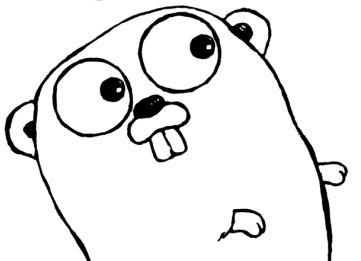
by Skyler Hawthorne

Advantages of Go over C

- Goroutine arguments can be typed
- Whereas C's pthreads take and return `void*`
- Many aspects of concurrency are simpler
- Faster Compile Time

Advantages of C over Go

- C gives more fine grain control
- e.g. options for thread scheduling algorithms



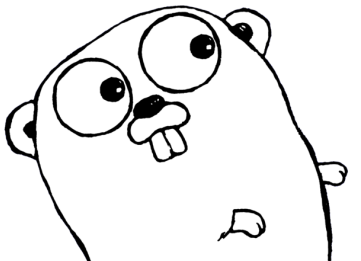
A Language Comparison Between Parallel Programming Features of Go and C

by Skyler Hawthorne

Criticisms

“The Go team says that “This approach can be taken too far. Reference counts may be best done by putting a mutex around an integer variable, for instance””

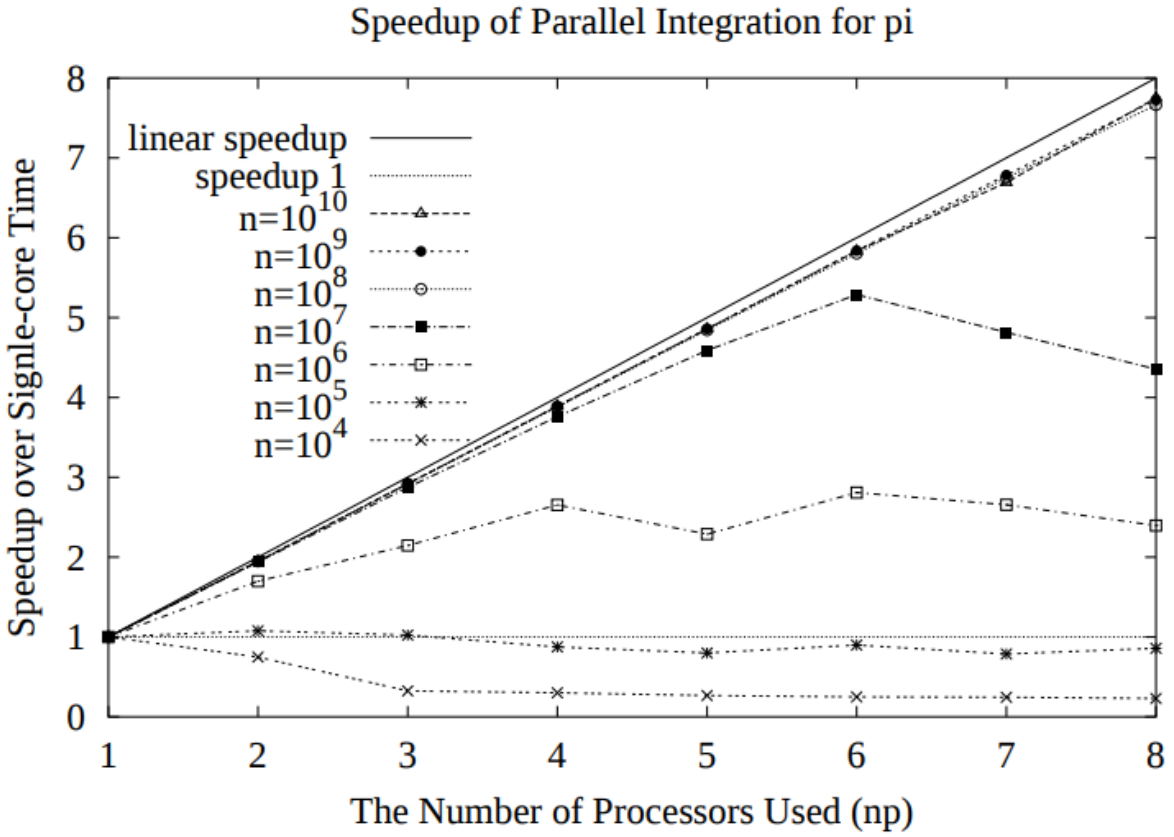
- Doesn’t compare the difference in how mutexes function

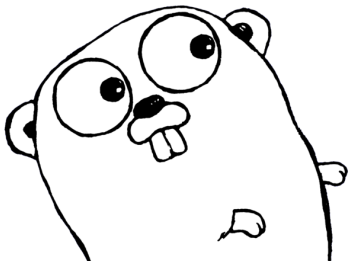


Multi-Core Parallel Programming in Go

by Peiyi Tang

- Provides two different programming problems
- Analyses the ease of writing the solutions
- And their speeds on different numbers of cores





Multi-Core Parallel Programming in Go

by Peiyi Tang

- Good diagrams
- Thourough explanation of problems and solutions

Criticism

- No comparison with another language