The Go Programming Language

Spencer Nelson

19 June 2020

Goals of the talk

What is Go?

What is Go good at?

What is Go bad at?

What does Go feel like?

How I know learned Go

- Web development at ZeroCater, 2011-2013: Python
- Data science at Twitch, 2013-2015: Python
- Streaming video software at Twitch/AWS, 2015-2020: Go

What is Go: Why you might care

Go is quickly becoming a lingua franca of web backends.

Things in Go you may have heard of:

- Docker
- Kubernetes
- InfluxDB, Prometheus
- Hashicorp Vault, Consul

What is Go: History

Rob Pike, Ken Thompson, Robert Griesemer started it in 2007 at Google

> I wasn’t allowed to use threads to solve a concurrent problem in [a big C++ program], because the C++ libraries didn’t work properly in that way, and the style rules forbid the use of threads in the binary. So I was doing gymnastics, which were very difficult to get right, to do what struck me as a very simple job...
>
> And every time I touched anything, I had to wait 45 minutes for another build, on a huge distributed compile cluster. At some point, my morale just broke. We had to do something... But I disctinctly remember turning the chair around and saying “Robert, help!”
>
> We talked for a few minutes, and then Ken was in the next office, so I ran and got Ken and said “Do you wanna help?” He said yes, and that was it.

What is Go: History

So they wrote a language in their spare time.

Wrote a formal spec and implemented a compiler in 2008.

First public release was in 2009.

Go 1.0 was released in 2012, and the language was declared stable.

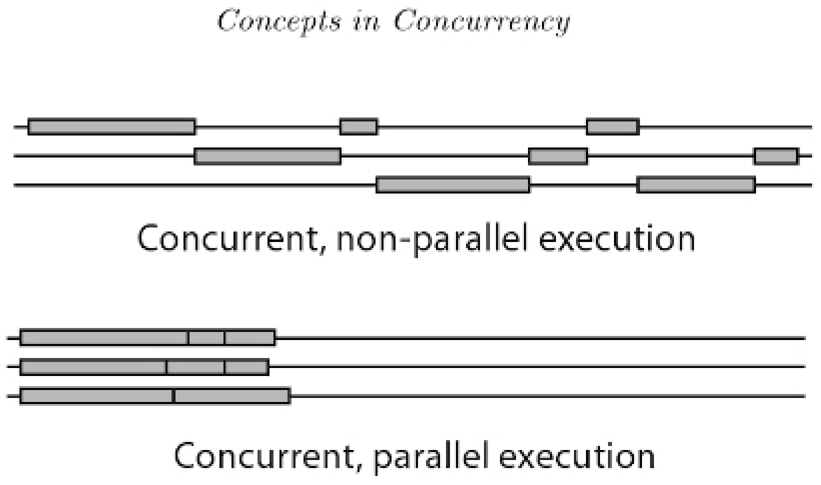
Today, in 2020, we're at Go 1.14.

What is Go: Design

Go is a language for whose main goal is productivity.

- Fast compilation
  - Most programs (<10k lines, few dependencies): Under 1 second
  - Kubernetes (~3M lines of code): 1m57s from a cold cache, 22.5s warm
- Simple builds
  - `go build k8s.io/cubernetes/kubectl` - no special config
- Very simple language, not many features
- Garbage collected
- Natively multicore
- Executables are fully self-contained and statically linked
- No inheritance, not really object-oriented

Sidebar: Concurrency vs parallelism



Concepts in Concurrency
Concurrent, non-parallel execution
Concurrent, parallel execution

Concurrency: Structure a program so it can switch between lots of different tasks

Parallelism: Do lots of tasks simultaneously

Concurrency: Wiggle your mouse while playing a youtube video

Parallelism: Compute a vector dot product

What is Go: Sample code

```go
func pageSize(url string) (size int64, err error) {
    // pageSize returns the size in bytes of the HTTP response for url
    resp, err := http.Head(url)
    if err != nil {
        return 0, err
    }
    if resp.StatusCode != 200 {
        return 0, fmt.Errorf("bad status code: %v", resp.StatusCode)
    }
    return resp.ContentLength, nil
}

func main() {
    start := time.Now()

    for id := 1; id < 150; id++ {
        url := fmt.Sprintf("https://dmtn-%03d.lsst.io", id)
        size, err := pageSize(url)
        if err != nil {
            fmt.Printf("err retrieving %s: %v\n", url, err)
        }
        fmt.Printf("%s: %s: %d\n", time.Since(start), url, size)
    }
}
```

Run

What is Go: Concurrency primitives

```
func main() {
    start := time.Now()
    dmtns := make(chan string)

    for id := 1; id < 150; id++ {
        url := fmt.Sprintf("https://dmtn-%03d.lsst.io", id)
        go func() {
            size, err := pageSize(url)
            if err != nil {
                dmtns <- fmt.Sprintf("err retrieving %s: %v", url, err)
            } else {
                dmtns <- fmt.Sprintf("%s: %d", url, size)
            }
        }()
    }

    for msg := range dmtns {
        fmt.Printf("%s: %s\n", time.Since(start), msg)
    }
}
```

Run

What is Go: Goroutines

If you call a function, `f(x, y)`, you have to wait for it to return.

If you call it with `go f(x, y)`, it goes off and executes - and you get to continue. Kind of like ending a command with `&` in bash. We call this a **goroutine**.

Goroutines are not threads. They're lighter-weight, and Go comes with its own scheduler.

It is routine to create thousands of Goroutines. Sometimes you create millions.

The work then becomes communicating between Goroutines - how do you get `f(x, y)`'s value back? You do this with **channels**, which are little queues for sending messages.

What is Go: Concurrency primitives are usually not enough

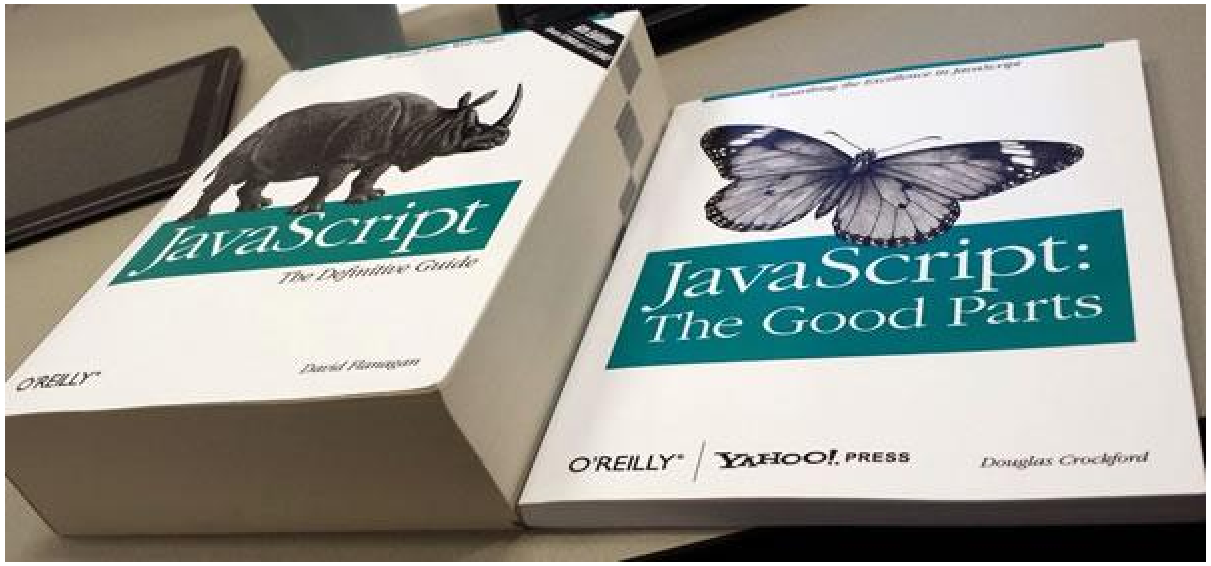
```
func main() {
    start := time.Now()
    dmtns := make(chan string)
    var wg sync.WaitGroup
    for id := 1; id < 150; id++ {
        url := fmt.Sprintf("https://dmtn-%03d.lsst.io", id)
        wg.Add(1)
        go func() {
            size, err := pageSize(url)
            if err != nil {
                dmtns <- fmt.Sprintf("err retrieving %s: %v", url, err)
            } else {
                dmtns <- fmt.Sprintf("%s: %d", url, size)
            }
            wg.Done()
        }()
    }
    go func() {
        wg.Wait()
        close(dmtns)
    }()

    for msg := range dmtns {
        fmt.Printf("%s: %s\n", time.Since(start), msg)
    }
}
```

Run

What is Go: Small language

Most languages add features over time, but rarely remove any.

You end up writing in a subset of the language.



Reading someone else's code gets hard: when did they write it, and what subset did *they* use?

What is Go: Small language

Go has very a small set of features, and adds features very slowly.

As of Go 1, the language is locked.

For example:

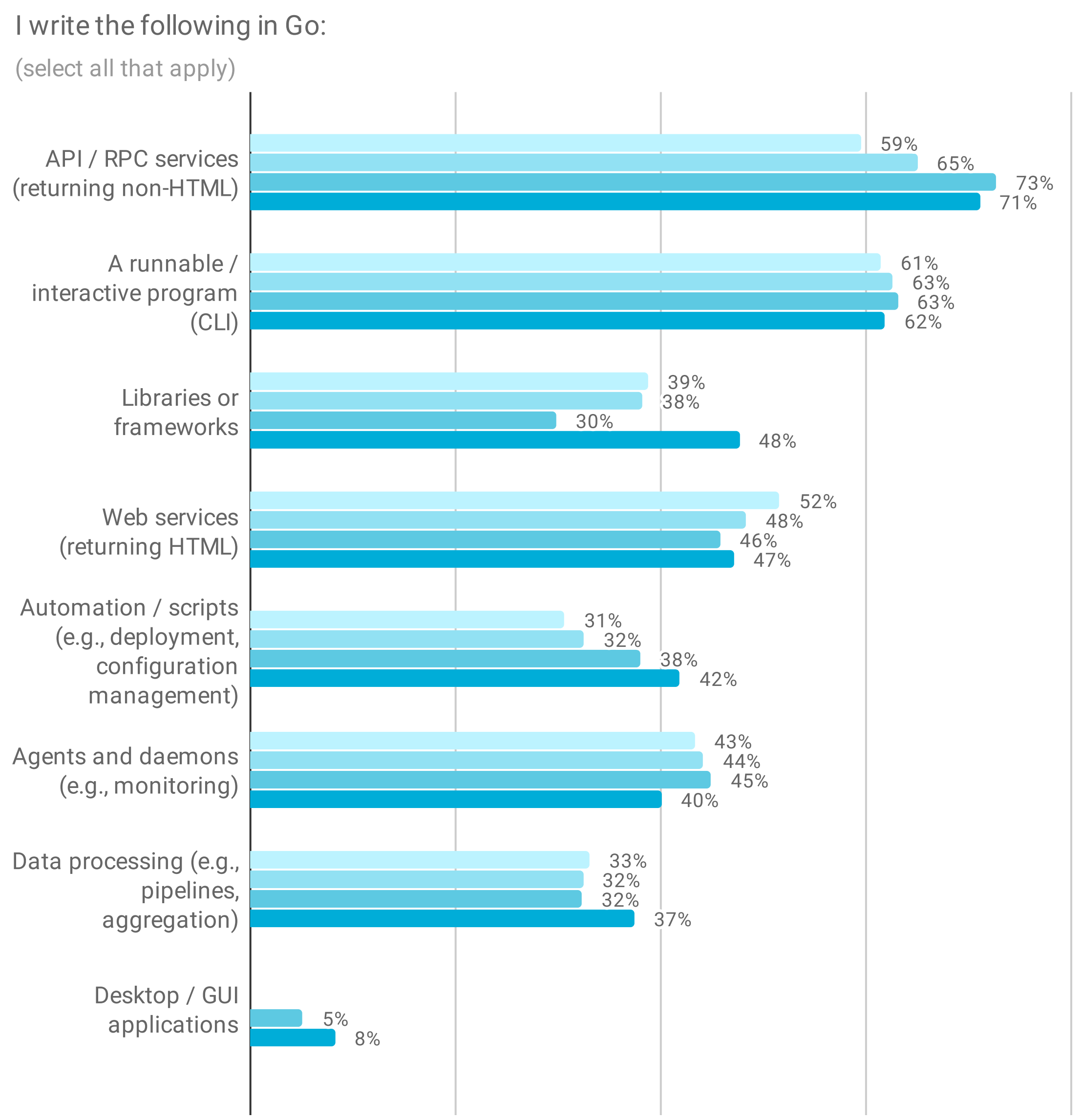
- barebones error handling
- no generics
- no complex type algebra
- no pattern matching
- no list comprehensions

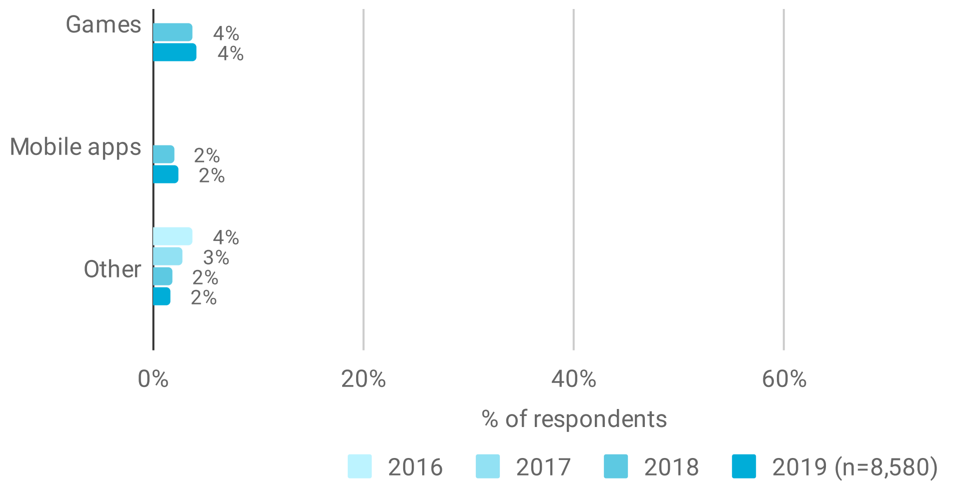
Basically no metaprogramming.

But: upgrading from go1.2 (2013) to go1.14 (2020) is easy, and I can read anyone's Go code and understand what's going on.

What is Go good at: Anything that benefits from concurrency

Go shines for anything that lives on a network.





Games
4%
4%
Mobile apps
2%
2%
Other
4%
3%
2%
2%
0%
20%
40%
60%
80%
% of respondents
2016
2017
2018
2019 (n=8,580)

What is Go good at: Software Engineering

Big software projects with lots of collaboration are easier because code is so readable.

Most C or Python code is terrible, but a small fraction is *beautiful*; pretty much all Go code is acceptable.

Tools are easy to write, too. Writing a new linter is about a day of work.

Stuff is fast enough to work in that I find myself reading for Go for simple scripts and using `go run whatever.go`.

What is Go good at: It's fast enough

The language doesn't have many abstractions to get in the way, unlike (for example) Python's objects. As a result, most of the time you're going at C speeds.

The language also comes with a scheduler built-in that will spread work across all the cores you give it. Parallelizing, and coordinating that parallelism, is simple and efficient.

The standard library takes performance very seriously. The emphasis is on *predictable* performance.

What is Go bad at

There's no REPL. Nothing even nearly as good as Jupyter.

Numeric routine libraries (like LAPACK or BLAS) are not very developed.

Nothing nearly as developed as the Python ecosystem for astronomy*

*Sebastien Binet, @sbinet in your local LSST slack, has written https://github.com/astrogo/fitsio and other packages related to astronomy!

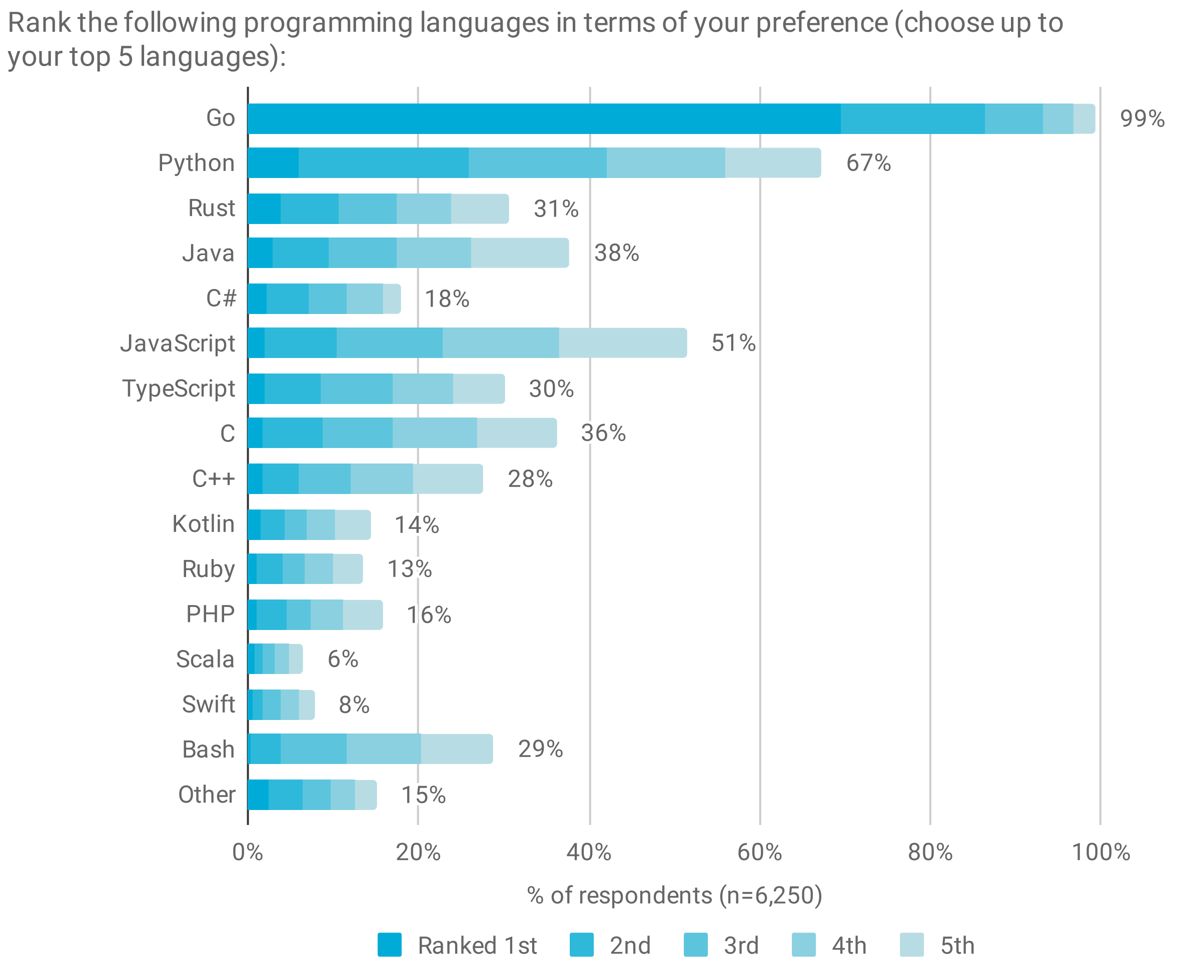
Package management is still a mess.

What does Go feel like?

Pretty boring. But boring can be good!

The designers of Go set out to make a simpler, more productive C++. I think they ended up making a simpler, more efficient, more boring Python.

What does Go feel like?



What does Go feel like?

"Some programmers find it fun to work in; others find it unimaginative, even boring. Those are not contradictory positions.

"Go was designed to address the problems faced in software development at Google, which led to a language that is not a breakthrough research language but is nonetheless an excellent tool for engineering large software projects."

> Rob Pike, "Go At Google: Language Design in the Service of Software Engineering (https://talks.golang.org/2012/splash.article)", 2012

Thank you

Spencer Nelson
19 June 2020
swnelson@uw.edu (mailto:swnelson@uw.edu)