Go

Berkeley CS 294-101
Mar 18, 2015

Rob Pike
Google

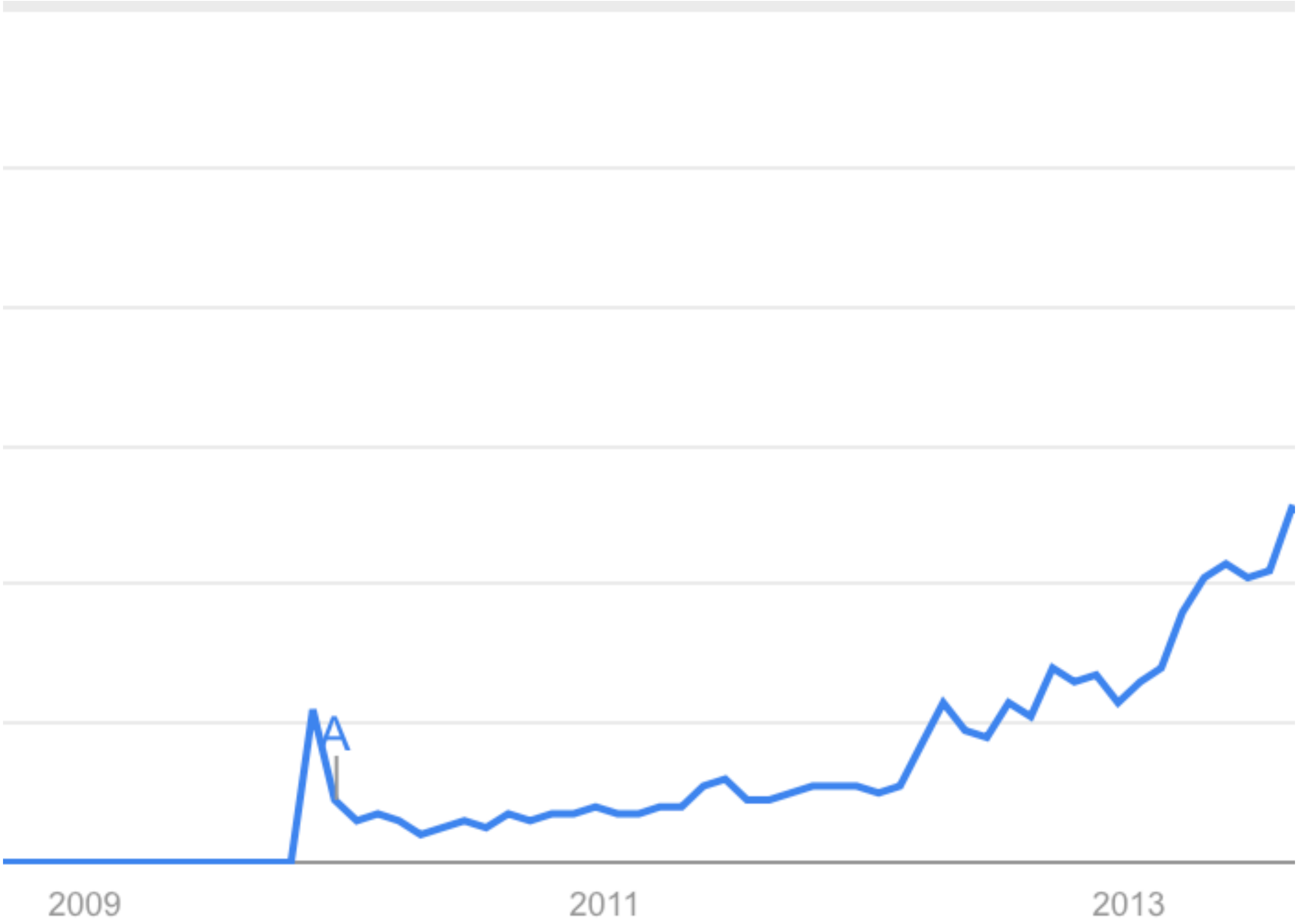
What is Go?

Go is:

- designed by Google
- open source
- concurrent
- garbage-collected
- compiled
- scalable
- simple
- fun
- boring (to some)

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

Adoption



Why?

Go is an answer to problems of scale at Google.

How?

By designing a language for software engineering.

What is important?

Properties

The "abilities":

- Readability
- Scalability
- Suitability
- Toolability

Readability

Overview

The readability of programs is immeasurably more important than their writeability.

Hints on Programming Language Design
C. A. R. Hoare 1973

Readability

The purpose of notation:

- clearly express what we care about

Clarity: Plan for the future

- program for someone else, years from now
- one-liners not the gold standard
- a balance between clarity and redundancy

Too cold

```
scoped_ptr<goscript::GoScript>
    goscript(goscript::GoScript::NewGoScript(FLAGS_goscript, goscript::
```

Too hot

```
(n: Int) => (2 to n) |> (r => r.foldLeft(r.toSet)((ps, x) =>
    if (ps(x)) ps -- (x * x to n by x) else ps))
```

Just right

```
t := time.Now()
switch {
case t.Hour() < 12:
    return "morning"
case t.Hour() < 18:
    return "afternoon"
default:
    return "evening"
}
```

Naming

How names work in a programming language is critical to readability.

Scope

Go has very simple scope hierarchy:

- universe
- package
- file (for imports only)
- function
- block

Locality of names

Nuances:

- upper case names for visibility: `name` *vs.* `Name`
- no implicit `this` in methods (receiver is explicit); always see `rcvr.Field`
- package qualifier always present for imported names
- (first component of) every name is always declared in current package

Locality scales

No surprises when importing:

- adding an exported name to my package cannot break your package!

Names do not leak across boundaries.

In C, C++, Java the name y could refer to anything.
In Go, y (or even Y) is always defined within the package.
In Go, x.Y is clear: find x locally, Y belongs to it.

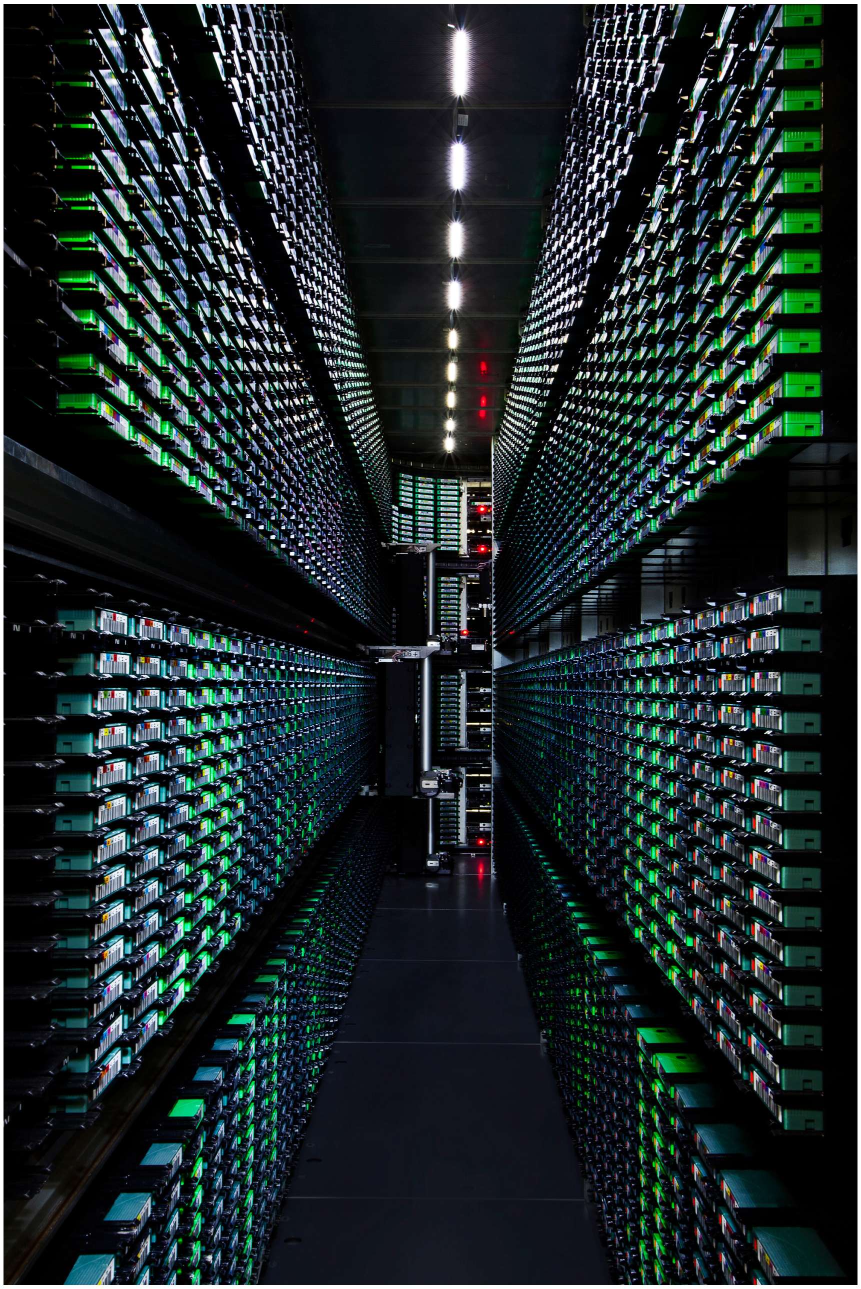
Function and method lookup

Method lookup by name only, not type.
A type cannot have two methods with the same name, ever.
Easy to identify which function/method is referred to.
Simple implementation, simpler program, fewer surprises.

Given a method `x.M`, there's only ever one `M` associated with `x`.

Scalability

19
20
21
F
22
23
24
25
26 26



Scalability

Google means scale in multiple dimensions

- computers
- cores
- data
- code
- engineers

Plus scaling has a big effect on:

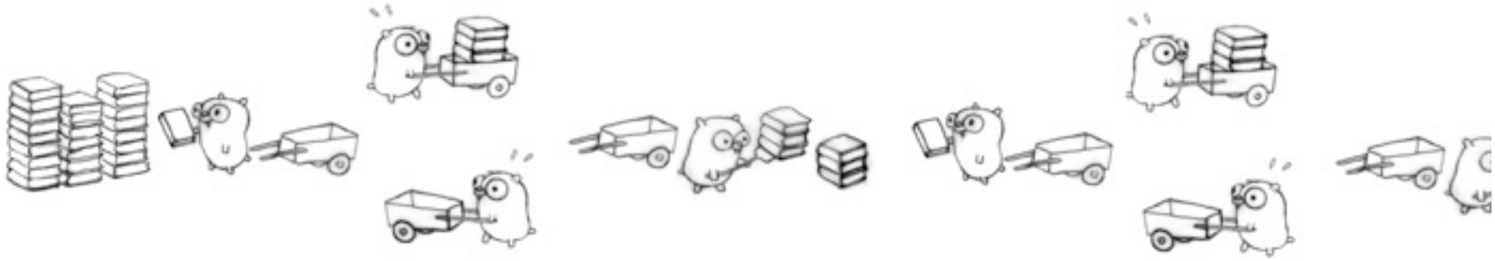
- speed of compilation
- speed of testing

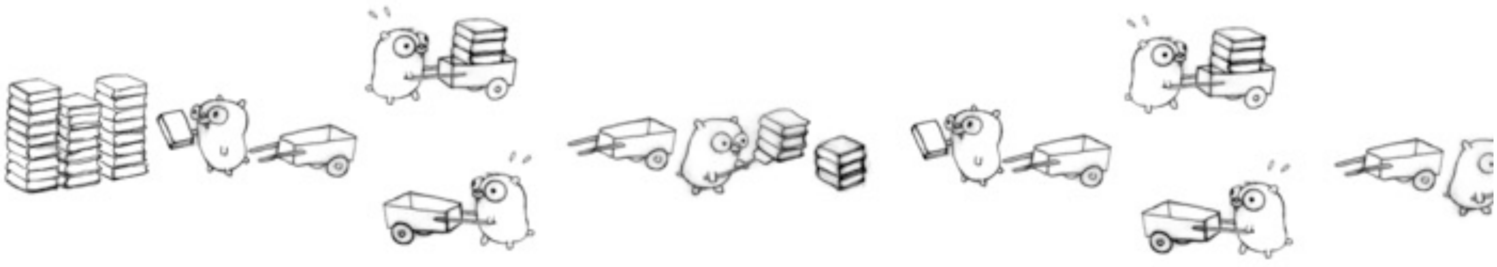
System scale

System scale

- 10^{6+} machines (design point)
- routine to be running on 1000 machines
- coordinating, interacting with other servers
- lots going on at once

Solution: great support for concurrency





Engineering scale

Engineering scale

In 2011 at Google:

- single code tree
- 5000+ developers across 40+ offices
- 20+ changes per minute
- 50% of source files change every month
- 50 million test cases executed per day

Solution: engineer language for large code bases

Software scale

Dependencies in C++

Explosive, exponential, almost non-computable.

In 2007, instrumented building a large Google web-serving binary:

- 2000 files
- 4.2 megabytes
- 8 gigabytes delivered to compiler
- 2000 bytes sent to compiler for every C++ source byte
- it's real work too: `<string>` for example
- hours to build

Dependencies in Go

Linguistically defined.

Efficient.

Computable.

Hoisting dependencies

Consider:
A imports B imports C but A does not directly import C.

The object code for B includes all the information about C needed to import B.
Therefore in A the line

```
import "B"
```

does not require the compiler to read C when compiling A.

Also, the object files are designed so the "export" information comes first; compiler doing import does not need to read whole file.

Exponentially less data read than with `#include` files.

With Go in Google, about 40X fanout (recall C++ was 2000x)
Plus in C++ it's general code that must be parsed; in Go it's just export data.

Scalability requires readability

For code to grow safely as time passes and staff changes:

- it must be readable
- it must be clear
- it must be adaptable
- it must be local

The themes resonate.

Suitability

Suitability

Can the language do the job?

Language is notation for a problem; not all languages are good for all problems.

Go was designed *for Google* to help solve *Google's problems.*

Concurrency is vital

Linguistic support for concurrent execution makes programming in the Google environment easier, safer, and more productive.

A key reason for Go's existence.

Go in production

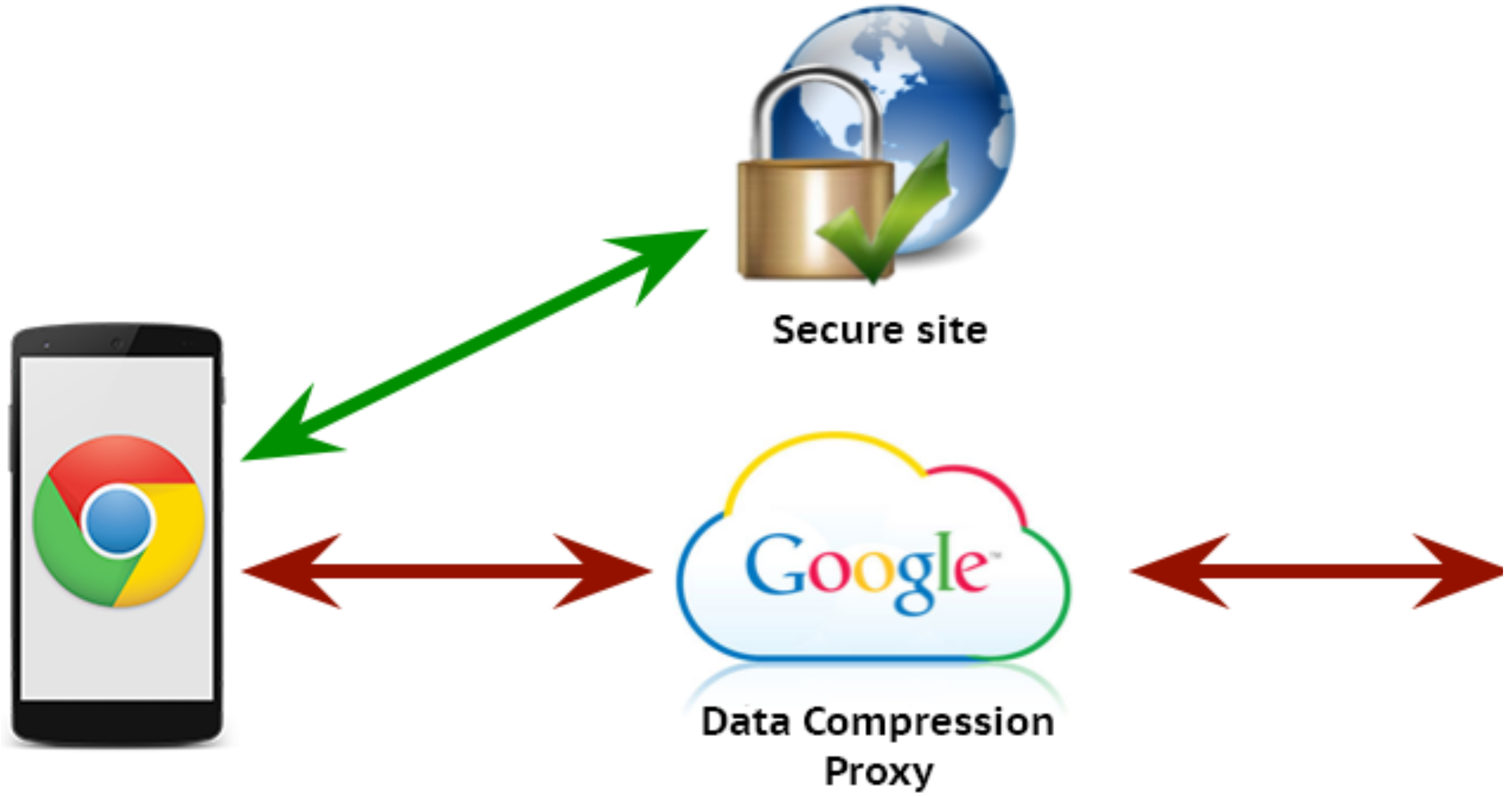
Several big services are written in Go:

- `golang.org`
- `dl.google.com`
- `vitess`, part of `youtube.com`
- ...

Adoption finds issues; they are resolved; adoption easier next time.

SPDY

SPDY proxy for Chrome on mobile devices



Toolability

Toolability

Software engineering requires tools.

Go's syntax, package design, naming, etc. make tools easy to write.

Library includes lexer, parser and type checker.

Gofmt

Always intended to do automatic code formatting.
Eliminates an entire class of argument.
Runs as a "presubmit" to the code repositories.

Training:

- The community has always seen `gofmt` output.

Sharing:

- Uniformity of presentation simplifies sharing.

Scaling:

- Less time spent on formatting, more on content.

Often cited as one of Go's best features.

Gofmt and other tools

Surprise: The existence of `gofmt` enabled *semantic* tools:
Can rewrite the tree; `gofmt` will clean up output.

Examples:

- `gofmt -r 'a[b:len(a)] -> a[b:]'`
- `gofix`

And good front-end libraries enable ancillary tools:

- godoc
- go `get`, go `build`, go `vet`, etc.
- `api`

Gofix

The `gofix` tool allowed us to make sweeping changes to APIs and language features leading up to the release of Go 1.

- changed syntax for deleting from a map
- new time API
- many more

Also allows us to *update* code even if the old code still works.

More recent example:

Changed Go's protocol buffer implementation to use getter functions; used `gofix` to update *all* google3 Go code.

Conclusion

Clarity is key.

Design for readability, not writeability.

Readability creates clarity, improving:

- productivity
- scale
- tooling

These effects multiply.

Questions?

Links:

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

talks.golang.org/2012/splash.article (http://talks.golang.org/2012/splash.article)

Thank you

Berkeley CS 294-101
Mar 18, 2015

Rob Pike
Google
r@golang.org (mailto:r@golang.org)
http://golang.org/ (http://golang.org/)