Go 1.8 Release Party

Golang Brno meetup
28 February 2017

Rodolfo Carvalho
Red Hat

License and Materials

The materials for this presentation are available on GitHub:

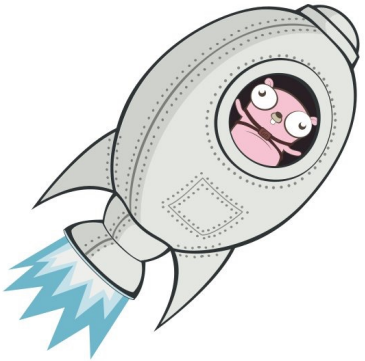
github.com/davecheney/go-1.8-release-party

You are encouraged to remix, transform, or build upon the material, providing you distribute your contributions under the same license.

If you have suggestions or corrections to this presentation, please raise an issue on the GitHub project.

Go 1.8

Go 1.8 is released!



Go 1.8 Announcement

Go 1.8 is the 9th release in the Go 1 series. It follows from the previous version, Go 1.7, released on the 16th of August, 2016

Go 1.8 Release Notes

What's changed?

So, what's happened in the last six months?

- Performance
- Compiler changes
- Tool changes
- Runtime changes
- Changes to the standard library

Performance

Performance

As always, the changes are so general and varied that precise statements about performance are difficult to make.

Most programs should run a bit faster, due to speedups in the garbage collector and optimizations in the standard library.

The new back end, based on static single assignment form (SSA), generates more compact, more efficient code and provides a better platform for optimizations such as bounds check elimination.

- The new back end reduces the CPU time required by our benchmark programs by 20-30% on 32-bit ARM systems.
- For 64-bit x86 systems, which already used the SSA back end in Go 1.7, the gains are a more modest 0-10%.
- Other architectures will likely see improvements closer to the 32-bit ARM numbers.

Go 1.8 Toolchain Improvements

Garbage Collector

Garbage collection pauses should be significantly shorter than they were in Go 1.7, usually under 100 microseconds and often as low as 10 microseconds.

> Garbage Collector Improvements Over Time (ht. @francesc)

Defer

The overhead of deferred function calls has been reduced by nearly half.

```
name           old time/op  new time/op  delta
Defer-4        75.1ns ± 1%  43.3ns ± 1%  -42.31%   (p=0.000 n=8+10)

In reality, this speeds up defer by about 2.2X. The two benchmarks
below compare a Lock/defer Unlock pair (DeferLock) with a Lock/Unlock
pair (NoDeferLock). NoDeferLock establishes a baseline cost, so these
two benchmarks together show that this change reduces the overhead of
defer from 61.4ns to 27.9ns.

name           old time/op  new time/op  delta
DeferLock-4    77.4ns ± 1%  43.9ns ± 1%  -43.31%  (p=0.000 n=10+10)
NoDeferLock-4  16.0ns ± 0%  15.9ns ± 0%   -0.39%    (p=0.000 n=9+8)
```

Source: CL 29656

Defer and cgo improvements

Cgo

The overhead of cgo calls is reduced by more than half:

```
name       old time/op  new time/op  delta
CgoNoop-8   146ns ± 1%    56ns ± 6%  -61.57%  (p=0.000 n=25+30)
```

This is the result of merging two defers on the standard cgo calling path.

Less cgo overhead in Go 1.8

Miscellaneous performance improvements

Lots of small updates to the standard library.

There have been optimizations to implementations in the `bytes`, `crypto/aes`, `crypto/cipher`, `crypto/elliptic`, `crypto/sha256`, `crypto/sha512`, `encoding/asn1`, `encoding/csv`, `encoding/hex`, `encoding/json`, `hash/crc32`, `image/color`, `image/draw`, `math`, `math/big`, `reflect`, `regexp`, `runtime`, `strconv`, `strings`, `syscall`, `text/template`, and `unicode/utf8` packages.

Go 1.8 Standard Library Changes

Compiler changes

SSA backend

- All backends updated to use the new SSA form. Old "plan9" style compiler backend deleted.
- Big performance improvements for 32-bit ARM.
- Easy to add new SSA backends (for values of easy compiler engineers work in). eg, adding MIPS backend was straight forward, SPARC and RISC-V are rumored to be next.

New parser

Robert Griesemer and Matthew Dempsky replaced the parser.

The new parser removes many of the package level variables which came along for the ride from the previous YACC based parser.

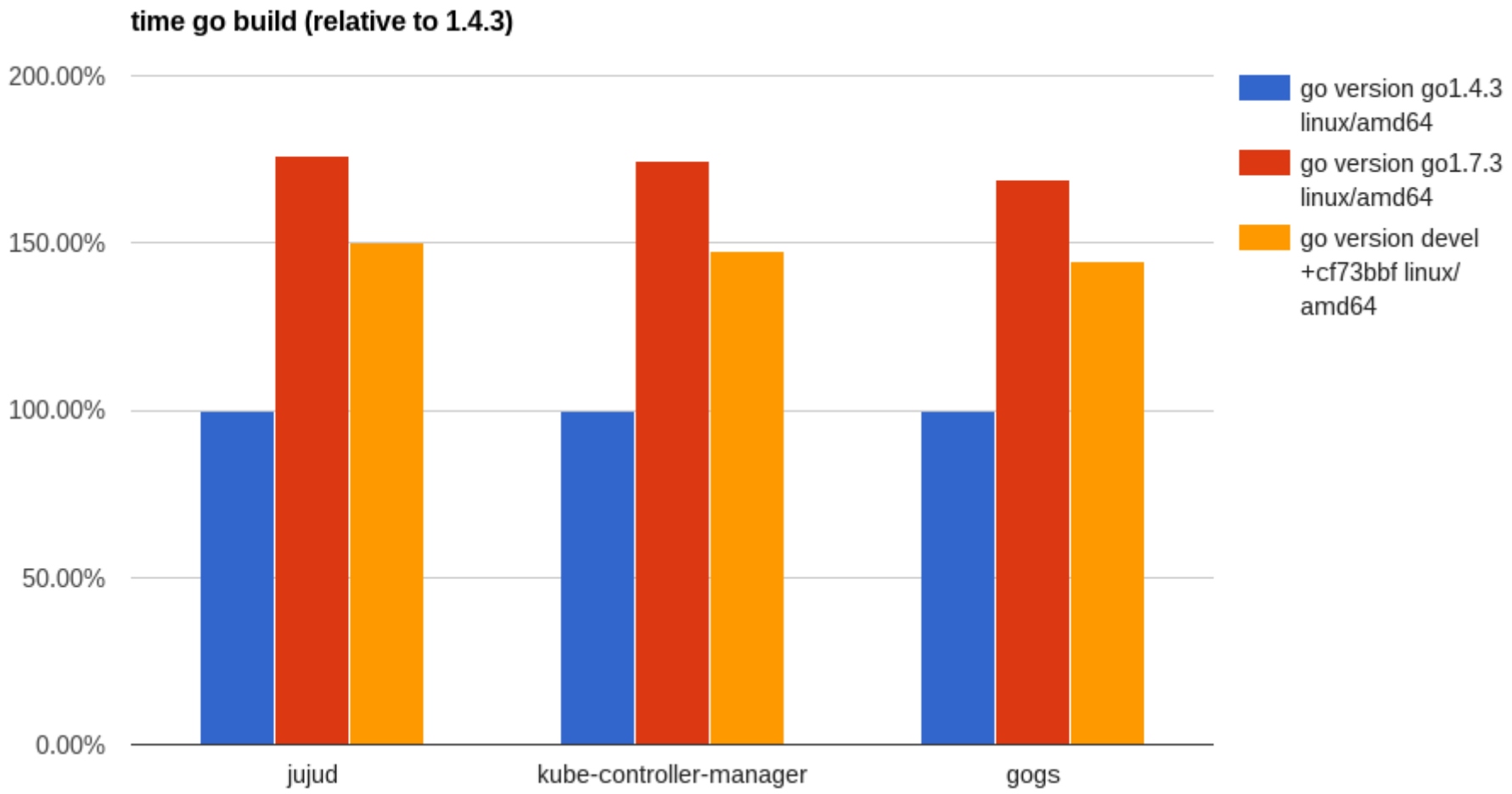
Removing package level globals is a stepping stone to making the compiler more parallel. In fact this work has started in the development branch.

This new parser is considerably faster, but in 1.8 is hamstrung by having to convert between the new style AST and the old version the rest of the compiler expects.

Expect this AST conversion to be removed in Go 1.9.

Compiler speed

About a 12-15% improvement compared to Go 1.7



Struct conversion

When explicitly converting a value from one struct type to another, as of Go 1.8 the tags are ignored.

```
func example() {
    type T1 struct {
        X int `json:"foo"`
    }
    type T2 struct {
        X int `json:"bar"`
    }
    var v1 T1
    var v2 T2
    v1 = T1(v2) // now legal
    _ = v1
}
```

Run

Ports

The number of platforms Go supports, on balance, continues to grow.

- Go now supports 32-bit MIPS on Linux for both big-endian (linux/mips) and little-endian machines (linux/mipsle) that implement the MIPS32r1 instruction set with FPU or kernel FPU emulation. Note that many common MIPS-based routers lack an FPU and have firmware that doesn't enable kernel FPU emulation; Go won't run on such machines.
- On DragonFly BSD, Go now requires DragonFly 4.4.4 or later.
- On OpenBSD, Go now requires OpenBSD 5.9 or later.
- The Plan 9 port's networking support is now much more complete and matches the behavior of Unix and Windows with respect to deadlines and cancelation.
- Go 1.8 now only supports OS X 10.8 or later. *This is likely the last Go release to support 10.8*. Compiling Go or running binaries on older OS X versions is untested.

Go 1.8 may be the last to support ARMv5

Supporting ARMv5 has been a problem for several releases.

Currently we have no ARMv5 builders so have no idea if Go 1.8 actually works on real ARMv5 hardware (we test on ARMv7).

Several companies have complained about loosing ARMv5 support, and several have offered to run a builder -- but there are *still* no real ARMv5 builders on the dashboard.

If you or your company care about ARMv5, you have to shoulder some of the burden. Contact @bradfitz and set up a builder.

> Proposal: runtime: drop support for linux/armv5E and linux/armv6
>
> Dashboard Builders (wiki)

No ARMv5 builders means ARMv5 support will be dropped in Go 1.9.

Tool changes

go tool yacc

The yacc tool (previously available by running “go tool yacc”) has been removed. As of Go 1.7 it was no longer used by the Go compiler.

It has moved to the “tools” repository and is now available at golang.org/x/tools/cmd/goyacc.

go tool asm

`go tool asm` now supports a flotilla of Intel and PPC vector instructions.

For 64-bit x86 systems, the following instructions have been added: `VBROADCASTSD`, `BROADCASTSS`, `MOVDDUP`, `MOVSHDUP`, `MOVSLDUP`, `VMOVDDUP`, `VMOVSHDUP`, and `VMOVSLDUP`.

For 64-bit PPC systems, the common vector scalar instructions have been added: `LXS`, `LXSDX`, `LXSI`, `LXSIWAX`, `LXSIWZX`, `LXV`, `LXVD2X`, `LXVDSX`, `LXVW4X`, `MFVSR`, `MFVSRD`, `MFVSRWZ`, `MTVSR`, `MTVSRD`, `MTVSRWA`, `MTVSRWZ`, `STXS`, `STXSDX`, `STXSI`, `STXSIWX`, `STXV`, `STXVD2X`, `STXVW4X`, `XSCV`, `XSCVDPSP`, `XSCVDPSPN`, `XSCVDPSXDS`, `XSCVDPSXWS`, `XSCVDPUXDS`, `XSCVDPUXWS`, `XSCVSPDP`, `XSCVSPDPN`, `XSCVSXDDP`, `XSCVSXDSP`, `XSCVUXDDP`, `XSCVUXDSP`, `XSCVX`, `XSCVXP`, `XVCV`, `XVCVDPSP`, `XVCVDPSXDS`, `XVCVDPSXWS`, `XVCVDPUXDS`, `XVCVDPUXWS`, `XVCVSPDP`, `XVCVSPSXDS`, `XVCVSPSXWS`, `XVCVSPUXDS`, `XVCVSPUXWS`, `XVCVSXDDP`, `XVCVSXDSP`, `XVCVSXWDP`, `XVCVSXWSP`, `XVCVUXDDP`, `XVCVUXDSP`, `XVCVUXWDP`, `XVCVUXWSP`, `XVCVX`, `XVCVXP`, `XXLAND`, `XXLANDC`, `XXLANDQ`, `XXLEQV`, `XXLNAND`, `XXLNOR`, `XXLOR`, `XXLORC`, `XXLORQ`, `XXLXOR`, `XXMRG`, `XXMRGHW`, `XXMRGLW`, `XXPERM`, `XXPERMDI`, `XXSEL`, `XXSI`, `XXSLDWI`, `XXSPLT`, and `XXSPLTW`.

go tool trace

The `trace` tool has a new `-pprof` flag for producing pprof-compatible blocking and latency profiles from an execution trace.

Garbage collection events are now shown more clearly in the execution trace viewer.

Garbage collection activity is shown on its own row and GC helper goroutines are annotated with their roles.

Rhys Hiltner's execution tracer presentation (dotGo 2016)

go tool vet

Vet is stricter in some ways and looser where it previously caused false positives. Vet now checks for:

- copying an array of locks
- duplicate JSON and XML struct field tags
- non-space-separated struct tags
- deferred calls to HTTP `Response.Body.Close` before checking errors
- indexed arguments in `Printf`.

```
package main

type T struct {
    A string `json:"A"`
    B string `json:"A"`
    C string `json:"C"xml:"C"`
}
```

```
go tool vet examples/vet_repeated_json_tags.go
```

Run

go fix

`go fix` now rewrites references to `golang.org/x/net/context` to the standard library provided `context` package.

go bug

The new go bug command starts a bug report on GitHub, prefilled with information about the current system.

Example:

```
go bug
```

Run

Default $GOPATH

Lastly, it is no longer necessary to set `$GOPATH` before using the go command.

When `GOPATH` is not defined, the tool will use:

- `$HOME/go` on Unix
- `%USERPROFILE%\go` on Windows

If you don't like these defaults, just set `$GOPATH` to your preferred location.

Runtime changes

Mutex contention profile

Peter Wienberger has added a new pprof profile, mutex contention.

```
func BenchmarkMutex(b *testing.B) {
    const N = 16
    for n := 0; n < b.N; n++ {
        var done sync.WaitGroup
        done.Add(N)
        m := Map{m: make(map[string]int)}
        for i := 0; i < N; i++ {
            go func(i int) {
                defer done.Done()
                m.Put("foo", i)
            }(i)
        }
        done.Wait()
    }
}
```

note: Due to an oversight this only works with `sync.Mutex`, not `sync.RWMutex`, this oversight will be addressed early in Go 1.9.

runtime: contended mutex profiling doesn't work for sync.RWMutex

Mutex contention profile

```
% go test examples/mutex_test.go -bench=. -mutexprofile=mutex.out
% go tool pprof mutex.test mutex.out
(pprof) list
Total: 290.81ms
ROUTINE ======================== command-line-arguments.(*Map).Put in /Users/dfc/devel/go-1.8-release-pa
     0   290.81ms (flat, cum)   100% of Total
     .          .      9:}
     .          .     10:
     .          .     11:func (m *Map) Put(key string, val int) {
     .          .     12:   m.Lock()
     .          .     13:   m.m[key] = val
     .   290.81ms     14:   m.Unlock()
     .          .     15:}
     .          .     16:
     .          .     17:func BenchmarkMutex(b *testing.B) {
     .          .     18:   const N = 16
     .          .     19:   for n := 0; n < b.N; n++ {
```

Plugins

Go now supports a “plugin” build mode for generating plugins written in Go, and a new plugin package for loading such plugins at run time.

Plugin support is currently only available on Linux.

Plugin Example

```
package main

// // No C code needed.
import "C"

import "fmt"

var V int

func init() {
    fmt.Println("Plugin loading")
}

func F() {
    fmt.Printf("Hello, number %d\n", V)
}
```

Then to build plugin.so

```
$ go build -buildmode=plugin plugin.go
```

Plugin Example - Using Plugin

```
    fmt.Printf("main started...\n")
    p, err := plugin.Open("plugin.so")
    if err != nil {
        panic(err)
    }

    v, err := p.Lookup("V")
    if err != nil {
        panic(err)
    }

    f, err := p.Lookup("F")
    if err != nil {
        panic(err)
    }

    *v.(*int) = 7
    f.(func())() // prints "Hello, number 7"
```

```
$ go run main.go
main started...
Plugin loading
Hello, number 7
```

Plugins demo

Demo video: twitter.com/francesc

Source code: github.com/campoy/golang-plugins

os.Executable

`os.Executable` returns an absolute path to the currently running program.

```
    fmt.Printf("os.Args[0]: %q\n", os.Args[0])
    exec, _ := os.Executable()
    fmt.Printf("os.Executable(): %q\n", exec)
```

Run

Executable returns the path name for the executable that started the current process. There is no guarantee that the path is still pointing to the correct executable.

If a symlink was used to start the process, depending on the operating system, the result might be the symlink or the path it pointed to. If a stable result is needed, path/filepath.EvalSymlinks might help.

os.Executable (godoc.org)

Detection of concurrent map accesses

In Go 1.6, the runtime added lightweight, best-effort detection of concurrent misuse of maps. Go 1.8 improves that detector with support for detecting programs that concurrently write to and iterate over a map.

```
func main() {
    m := map[int]int{4: 4, 8: 8, 15: 15, 16: 16, 23: 23, 42: 42}
    go func() {
        for {
            for _, v := range m {
                _ = v
            }
        }
    }()
    for {
        for k, _ := range m {
            m[k]++
        }
    }
}
```

Run

If the runtime detects this condition, it prints a diagnosis and crashes the program.

Changes to the standard library

sort.Slice

The `sort` package has a new convenice method `sort.Slice`

```
    switch strings.ToLower(field) {
    case "updated":
        sort.Slice(subs, func(i, j int) bool { return subs[i].Updated.After(subs[j].Updated) })
    case "trust":
        sort.Slice(subs, func(i, j int) bool { return subs[i].Trust > subs[j].Trust })
    case "stddev":
        sort.Slice(subs, func(i, j int) bool {
            var s1, s2 stats.Sample
            for _, r := range subs[i].Ratings {
                s1.Xs = append(s1.Xs, float64(r.Value))
            }
            for _, r := range subs[j].Ratings {
                s2.Xs = append(s2.Xs, float64(r.Value))
            }
            return s1.StdDev() < s2.StdDev()
        })
    case "rating":
        fallthrough
    default:
        sort.Slice(subs, func(i, j int) bool { return subs[i].Rating > subs[j].Rating })

    }
```

HTTP shutdown

A long requested feature, graceful shutdown of a `http.Server` was added in Go 1.8.

Call `Shutdown` when a signal is received:

```
func main() {
    srv := &http.Server{Addr: ":8080", Handler: http.DefaultServeMux}

    go func() {
        fmt.Println("Press enter to shutdown server")
        fmt.Scanln()
        log.Println("Shutting down server...")
        if err := srv.Shutdown(context.Background()); err != nil {
            log.Fatalf("could not shutdown: %v", err)
        }
    }()

    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintln(w, "Happy Go 1.8'th")
    })
    if err := srv.ListenAndServe(); err != http.ErrServerClosed {
        log.Fatalf("listen: %s\n", err)
    }
}
```

HTTP/2

`http.Response` now satisfies the `http.Pusher` interface.

```
type Pusher interface {
    Push(target string, opts *PushOptions) error
}
```

More context support

Continuing Go 1.7's adoption of `context.Context` into the standard library, Go 1.8 adds more context support to existing packages:

- The new `Server.Shutdown` takes a context argument.
- There have been significant additions to the `database/sql` package with context support.
- All nine of the new `Lookup` methods on the new `net.Resolver` now take a context.

Minor changes

Minor library changes

What's coming up in Go 1.9

What's coming up in Go 1.9

Before we close, let's quickly touch on some of the things coming up in Go 1.9

note: All of these are speculation, nothing is set in stone until code hits the repo.

Aliases

After being rolled back before the Go 1.8 freeze, aliases are being re-proposed for Go 1.9 in a more limited fashion.

golang.org/design/18130-type-alias

```
package main

type A struct { a int }

type B struct { a int }

type C B

type D = B

func main() {
    var ( a A; b B; c C; d D )
    a = b    // nope
    b = c    // also nope
    d = b    // new in Go 1.9!
}
```

Run

Faster / cheaper runtime.MemStats

The cost of calling `runtime.MemStats` is propotional to the size of the heap; Austin recently timed it at ~1.7ms per Gb.

There is a CL ready to land that reduces it to 20 us per proc (thread servicing a goroutine) which is a much smaller upper bound.

golang.org/issue/13613

Improvements to the inliner

Inlining has historically been limited to leaf functions because of the concern of agressive inlining on the call graph.

In support of this Robert Griesemer and Matthew Dempsky have been improving the line number tracking in Go 1.9 to make it flexible enough to lift the restriction on non-leaf functions.

An unnamed intern has been hired to work on the inliner for 1.9.

Josh Bleecher-Snyder has also suggest that the cost model for inlining should be revised to make sure inlining is paying for itself.

Issue 17566

A user level poller

Go has used epoll/kqueue/poll/select for *network sockets* for years.

Reads/Writes to other file descriptors have traditionally consumed an OS thread during operation

Recently Ian Lance Taylor landed a refactor that broken our the `runtime` polling subsystem and extended to work for the rest of the `os` package.

> net: refactor poller into new internal/poll package
>
> os: use poller for file I/O

There is no sign of a programmer accessible poller, yet.

Official dependency management tool

Finally!

Peter Bourgon, Edward Muller, Sam Boyer, Jess Frazelle, and Steve Francia have been working on an *official* dependency management tool, dep.

If you're used to Glide or govendor, you won't be too surprised.

> Mailing List Announcement

It's described as *pre alpha* at the moment, but you should check it out, and let them know what you think.

> github.com/golang/dep

Conclusion

Upgrade to Go 1.8, now!

It's literally the best version of Go, *ever*.

Thank you

Rodolfo Carvalho
Red Hat
rhcarvalho@gmail.com
http://rodolfocarvalho.net
https://dojo-brno.github.io