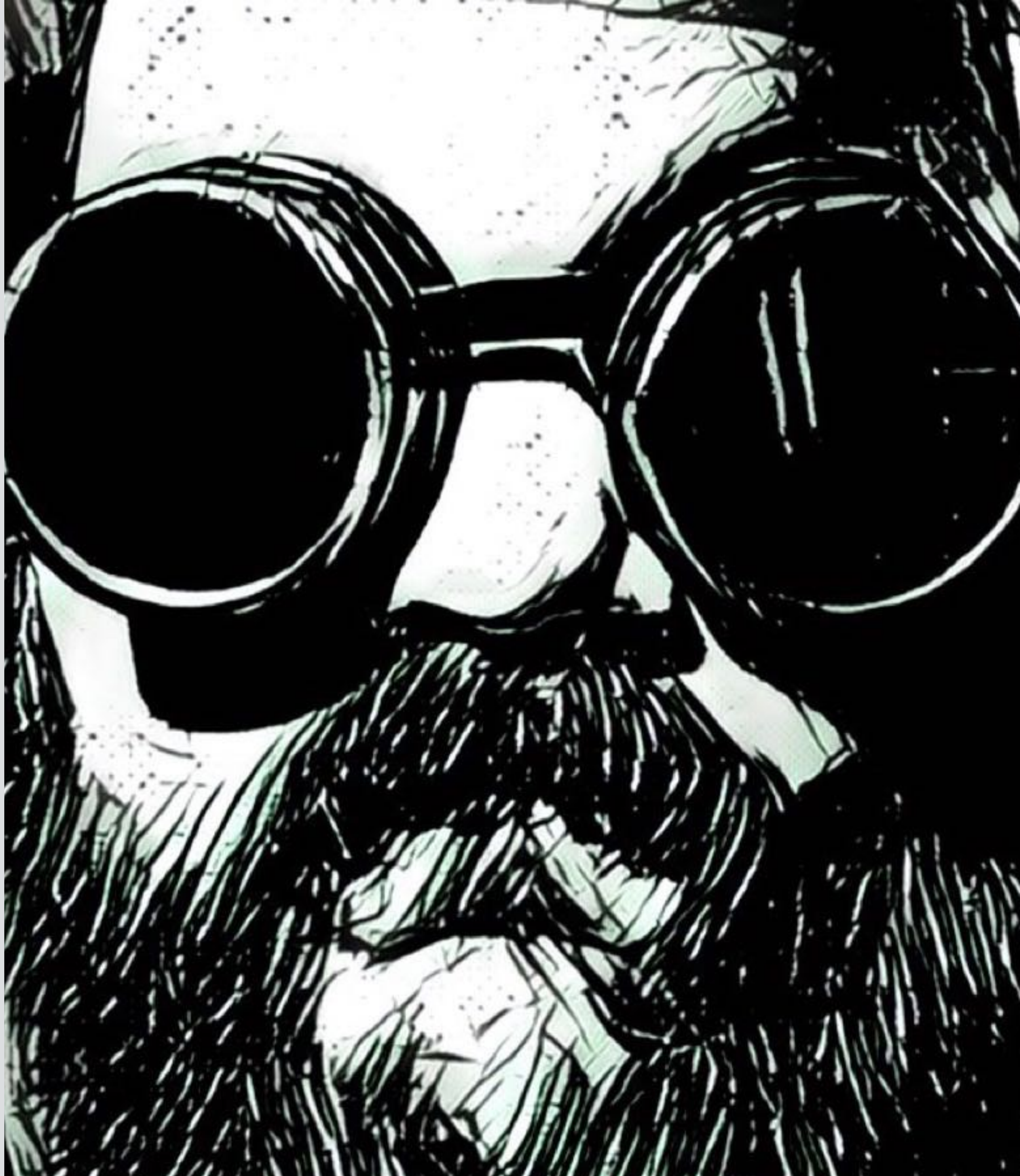
Go’s Hidden #pragmas

GopherChina 2017

@davecheney

Go programmer from Sydney, Australia



But first, a history lesson.

Perl

```
use strict;
use strict "vars";
use strict "refs";
use strict "subs";
use strict;
no strict "vars";
```

Javascript

```
"use strict";
```

Rust

```
#[inline(always)]
fn super_fast_fn() {

#[cfg(target_os = "macos")]
mod macos_only {
```

Warning: History time

ALGOL 68

pragmat

C

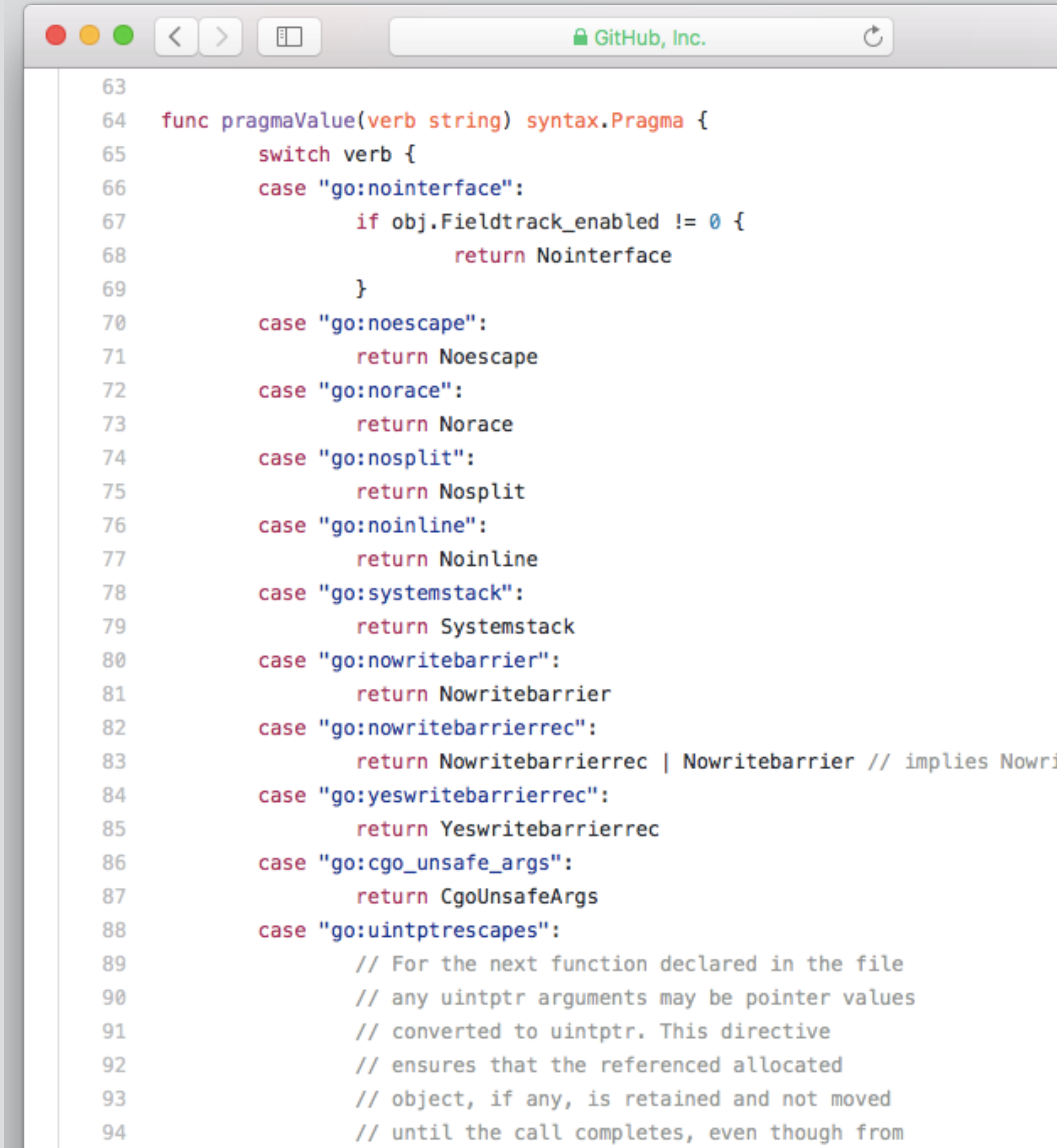
```
#pragma pack(2)
struct T {
    int i;
    short j;
    double k;
};
```

Does Go have pragmas?

Yes. Go has pragmas

//

They're actually called *pragmas* in the source



```
func pragmaValue(verb string) syntax.Pragma {
	switch verb {
	case "go:nointerface":
		if obj.Fieldtrack_enabled != 0 {
			return Nointerface
		}
	case "go:noescape":
		return Noescape
	case "go:norace":
		return Norace
	case "go:nosplit":
		return Nosplit
	case "go:noinline":
		return Noinline
	case "go:systemstack":
		return Systemstack
	case "go:nowritebarrier":
		return Nowritebarrier
	case "go:nowritebarrierrec":
		return Nowritebarrierrec | Nowritebarrier // implies Nowri
	case "go:yeswritebarrierrec":
		return Yeswritebarrierrec
	case "go:cgo_unsafe_args":
		return CgoUnsafeArgs
	case "go:uintptrescapes":
		// For the next function declared in the file
		// any uintptr arguments may be pointer values
		// converted to uintptr. This directive
		// ensures that the referenced allocated
		// object, if any, is retained and not moved
		// until the call completes, even though from
```

syscall/syscall_linux_amd64.go

```
//go:noescape
func gettimeofday(tv *Timeval) (err Errno)
```

cmd/compile/internal/gc/testdata/arith.go

```
//go:noinline
func lshNop1(x uint64) uint64 {
	// two outer shifts should be removed
	return (((x << 5) >> 2) << 2)
}
```

runtime/atomic_pointer.go

```
//go:nosplit
func atomicstorep(ptr unsafe.Pointer, new unsafe.Pointer) {
	writebarrierptr_prewrite((*uintptr)(ptr), uintptr(new))
	atomic.StorepNoWB(noescape(ptr), new)
}
```

A word of caution

🚨

"Useful" is always true for a feature request. The question is, does the usefulness justify the cost? The cost here is continued proliferation of magic comments, which are becoming too numerous already.

–Rob Pike

```
//go:noescape
```

Escape Analysis

```
func NewBook() *Book {
	b := Book{Mice: 12, Men: 9}
	return &b
}
```

Escape Analysis (cont.)

```
func BuildLibrary() {
	b := Book{Mice: 99: Men: 3}
	AddToCollection(&b)
}
```

Answer: it depends 🤔

b does not escape

```
func AddToCollection(b *Book) {
	b.Classification = "fiction"
}
```

b *escapes*

```
var AvailableForLoan []*Book

func AddToCollection(b *Book) {
	AvailableForLoan = append(AvailableForLoan, b)
}
```

os.File.Read

```
f, _ := os.Open("/tmp/foo")
buf := make([]byte, 4096)
n, _ := f.Read(buf)
```

os.File.Read

```
// Read reads up to len(b) bytes from the File.
// It returns the number of bytes read and any error encountered.
// At end of file, Read returns 0, io.EOF.
func (f *File) Read(b []byte) (n int, err error) {
	if err := f.checkValid("read"); err != nil {
		return 0, err
	}
	n, e := f.read(b)
	if e != nil {
		if e == io.EOF {
			err = e
		} else {
			err = &PathError{"read", f.name, e}
		}
	}
	return n, err
}
```

golang.org/issue/4099

```
commit fd178d6a7e62796c71258ba155b957616be86ff4
Author: Russ Cox <rsc@golang.org>
Date:   Tue Feb 5 07:00:38 2013 -0500

    cmd/gc: add way to specify 'noescape' for extern funcs

    A new comment directive //go:noescape instructs the compiler
    that the following external (no body) func declaration should be
    treated as if none of its arguments escape to the heap.

    Fixes #4099.

    R=golang-dev, dave, minux.ma, daniel.morsing, remyoudompheng, adg, agl, iant
    CC=golang-dev
    https://golang.org/cl/7289048
```

bytes.IndexByte (circa Go 1.5)

```
package bytes

//go:noescape
// IndexByte returns the index of the first instance of c in s,
// or -1 if c is not present in s.
func IndexByte(s []byte, c byte) int // ../runtime/asm_$GOARCH.s
```

Can you use //go:noescape in your code?

//go:norace

8c195bdf

```
// TODO(rsc): Remove. Put //go:norace on forkAndExecInChild instead.
func isforkfunc(fn *Node) bool {
	// Special case for syscall.forkAndExecInChild.
	// In the child, this function must not acquire any locks, because
	// they might have been locked at the time of the fork. This means
	// no rescheduling, no malloc calls, and no new stack segments.
	// Race instrumentation does all of the above.
	return myimportpath != "" && myimportpath == "syscall" &&
		fn.Func.Nname.Sym.Name == "forkAndExecInChild"
}
```

syscall/exec_bsd.go

```
// Fork, dup fd onto 0..len(fd), and exec(argv0, argvv, envv) in child.
// If a dup or exec fails, write the errno error to pipe.
// (Pipe is close-on-exec so if exec succeeds, it will be closed.)
// In the child, this function must not acquire any locks, because
// they might have been locked at the time of the fork. This means
// no rescheduling, no malloc calls, and no new stack segments.
// For the same reason compiler does not race instrument it.
// The calls to RawSyscall are okay because they are assembly
// functions that do not grow the stack.
//go:norace
func forkAndExecInChild(argv0 *byte, argv, envv []*byte, chroot, dir
        *byte, attr *ProcAttr, sys *SysProcAttr, pipe int)
        (pid int, err Errno) {
```

Should you use //go:norace in your own code?

//go:nosplit

Function preamble

```
"".fn t=1 size=120 args=0x0 locals=0x80
	0x0000 00000 (main.go:5)  TEXT    "".fn(SB), $128-0
	0x0000 00000 (main.go:5)  MOVQ    (TLS), CX
	0x0009 00009 (main.go:5)  CMPQ    SP, 16(CX)
	0x000d 00013 (main.go:5)  JLS     113
```

Warning: nerdy, technical, digression

#pragma textflag

```
// All reads and writes of g's status go through readgstatus, casgstatus
// castogscanstatus, casfromgscanstatus.
#pragma textflag NOSPLIT
uint32
runtime·readgstatus(G *gp)
{
	return runtime·atomicload(&gp->atomicstatus);
}
```

#pragma textflag

```
// All reads and writes of g's status go through
// readgstatus, casgstatus, castogscanstatus,
// casfrom_Gscanstatus.
//go:nosplit
func readgstatus(gp *g) uint32 {
        return atomic.Load(&gp.atomicstatus)
}
```

What happens when you run out of stack with //go:nosplit?

Can you use //go:nosplit in your own code?

//go:noinline

We particularly need this feature on the SSA branch because if a function is inlined, the code contained in that function might switch from being SSA-compiled to old-compiler-compiled. Without some sort of noinline mark the SSA-specific tests might not be testing the SSA backend at all.

–Keith Randall

```
func ishairy(n *Node, budget *int32, reason *string) bool
```

cmd/compile/internal/gc.ishairy()

```
case OCLOSURE,
        OCALLPART,
        ORANGE,
        OFOR,
        OSELECT,
        OSWITCH,
        OPROC,
        ODEFER,
        ODCLTYPE,  // can't print yet
        ODCLCONST, // can't print yet
        ORETJMP:
        return true
```

```
func f3a_ssa(x int) *int {
        switch {
        }
        return &x
}
```

Can you use //go:noinline in your own code?

//go:linkname

runtime/timeasm.go

```
// Declarations for operating systems implementing time.now directly in assembly.
// Those systems are also expected to have nanotime subtract startNano,
// so that time.now and nanotime return the same monotonic clock readings.

// +build darwin,amd64 darwin,386 windows

package runtime

import _ “unsafe"

//go:linkname time_now time.now
func time_now() (sec int64, nsec int32, mono int64)
```

time/time.go

```
// Provided by package runtime.
func now() (sec int64, nsec int32, mono int64)

// Now returns the current local time.
func Now() Time {
	sec, nsec, mono := now()
	sec += unixToInternal - minWall
	if uint64(sec)>>33 != 0 {
		return Time{uint64(nsec), sec + minWall, Local}
	}
	return Time{hasMonotonic | uint64(sec)<<nsecShift |
	    uint64(nsec), mono, Local}
}
```

Can you use //go:linkname in your own code?

Finding a coroutine’s id

Never, ever, do this. Seriously.

(hold my beer)

But what about …

```
// +build
```

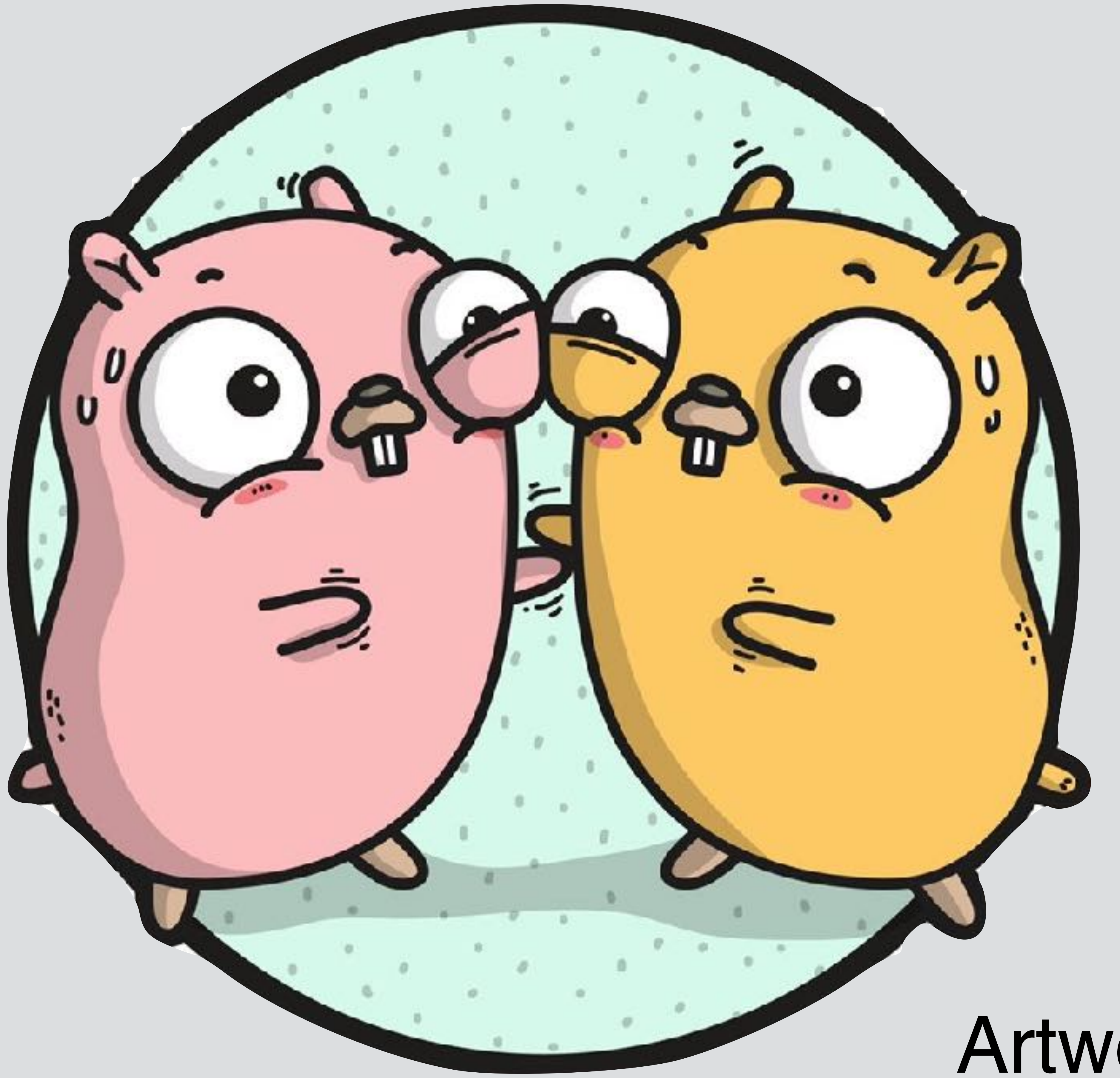
//go:generate

```
package pdf // import "rsc.io/pdf"
```

```
//line /foo/bar.go:123
```

Conclusion

Thank you!



Artwork @ashleymacnamara
Gopher design @reneefrench