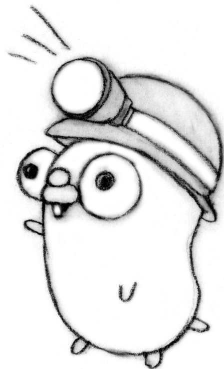
Dropping down

Go functions in assembly language

Michael Munday
Linux on IBM z Systems Open Source Ecosystem
18th August 2016



IBM

Agenda

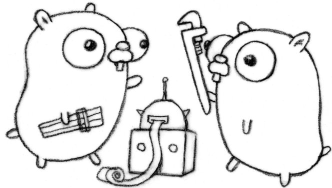
Introduction

Instructions

Functions & Stacks

Testing & Portability

IBM LinuxONE™



s390x in Go 1.7

What is the Go toolchain’s assembly language?

- Originates from the Plan 9 toolchain
- Has evolved mainly to meet the needs of the Go toolchain
- High-level (for an assembly language!)
  - Architecture independent mnemonics such as CALL and RET
  - Instructions may be expanded by the assembler
  - Assembler may insert prologues, optimize away ‘unreachable’ instructions
- Does not work in gccgo

```
#include "textflag.h"

DATA  text<>+0(SB)/8,$"Hello Wo"
DATA  text<>+8(SB)/8,$"rld!\n"
GLOBL text<>(SB),NOPTR,$16

// func printHelloWorld()
TEXT ·printHelloWorld(SB),$56-0
    NO_LOCAL_POINTERS
    MOVQ    $1, fd-56(SB)
    MOVQ    $text<>+0(SB), AX
    MOVQ    AX, ptr-48(SP)
    MOVQ    $13, len-40(SP)
    MOVQ    $16, cap-32(SP)
    CALL    syscall·Write(SB)
    RET
```

What is assembly used for in Go’s standard library?

crypto/aes
crypto/elliptic
crypto/md5
crypto/rc4
crypto/sha1
crypto/sha256
crypto/sha512
math
math/big

reflect
runtime
runtime/cgo
runtime/internal/atomic
sync/atomic
syscall

Terminology

- Mnemonic
  - CALL, MOVW, ADD, ...
- Register
  - R1, AX, V0, X3, F0, ...
- Immediate
  - $1, $0x100, ...
- Memory
  - (R1), 8(R3), ...

Agenda

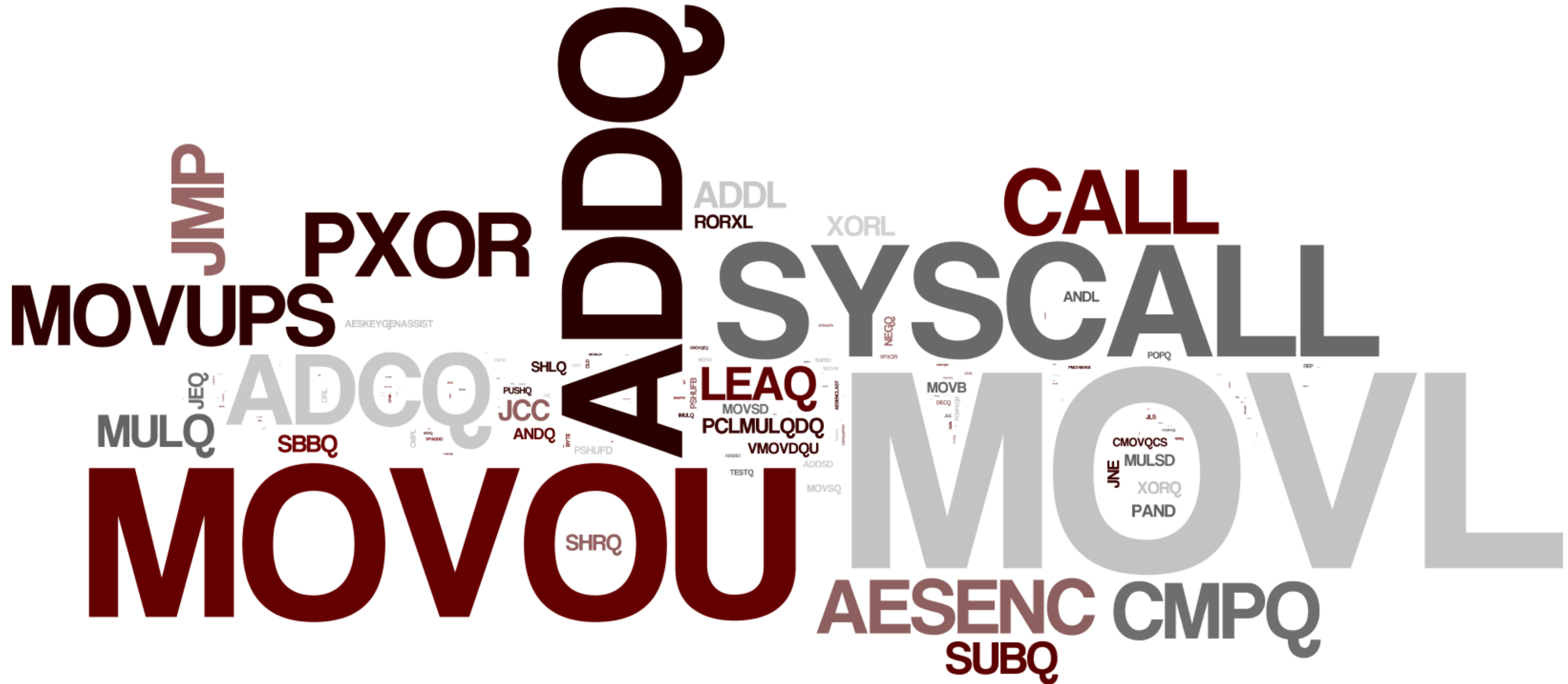
Introduction

Instructions

Functions & Stacks

Testing & Portability

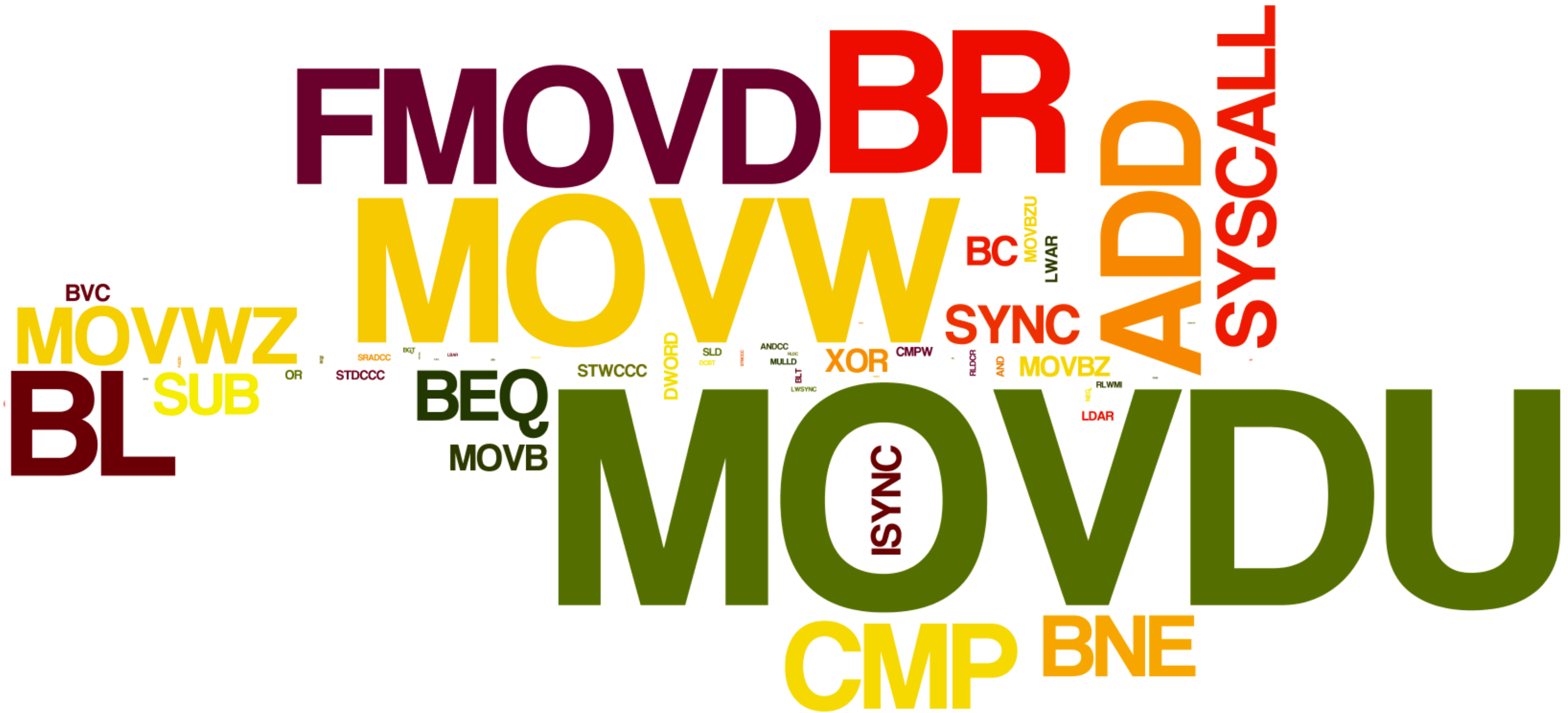
amd64 mnemonics (excluding MOVQ)



arm64 mnemonics (excluding plain MOVD)

MOVW
MOVD.W
BL
B
SVC
FMOVD
CMP
ADD
SUB
BCC
BNE
CMN
BEQ
MOVD.P
MOVB
MUL
MOVN
MOVBU.P
MOVWU
AND
LDAXRW
CBNZ
CSET
CBZ
LDAXR
STLXR
STLXRW
MOVBU.W
NEG

ppc64(le) mnemonics (excluding MOVD)



s390x mnemonics (excluding MOVD)

Move instructions

	386	amd64	arm	arm64	mips64	ppc64	s390x
1-byte	MOVB		MOVB	-	-	-	-
1-byte sign extend	MOVBLSX	MOVBQSX	MOVBS	MOVB	MOVB	MOVB	MOVB
1-byte zero extend	MOVBLZX	MOVBQZX	MOVBU	MOVBU	MOVBU	MOVBZ	MOVBZ
2-byte	MOVW		MOVH	-	-	-	-
2-byte sign extend	MOVWLSX	MOVWQSX	MOVHS	MOVH	MOVH	MOVH	MOVH
2-byte zero extend	MOVWLZX	MOVWQZX	MOVHU	MOVHU	MOVHU	MOVHZ	MOVHZ
4-byte	MOVL		MOVW	-	-	-	-
4-byte sign extend	-	MOVLQSX	-	MOVW	MOVW	MOVW	MOVW
4-byte zero extend	-	MOVLQZX	-	MOVWU	MOVWU	MOVWZ	MOVWZ
8-byte	-	MOVQ	-	MOVD	MOVV	MOVD	MOVD

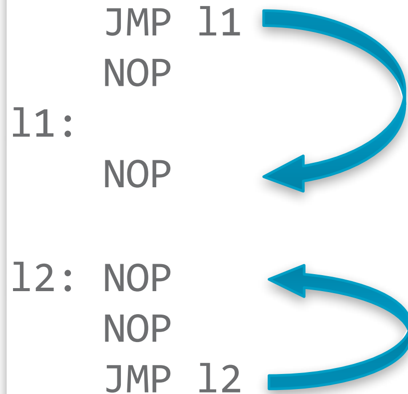
Instructions

- Data moves from left to right
  - `ADD R1, R2                => R2 += R1`
  - `SUB R3, R4, R5            => R5 = R4 - R3`
  - `MUL $7, R6                => R6 *= 7`
- Memory operands: offset + reg1 + reg2*scale
  - `MOV (R1), R2              => R2 = *R1`
  - `MOV 8(R3), R4             => R4 = *(8 + R3)`
  - `MOV 16(R5)(R6*1), R7 => R7 = *(16 + R5 + R6*1)`
  - `MOV ·myvar(SB), R8   => R8 = *myvar`
- Addresses
  - `MOV $8(R1)(R2*1), R3 => R3 = 8 + R1 + R2`
  - `MOV $·myvar(SB), R4  => R4 = &myvar`

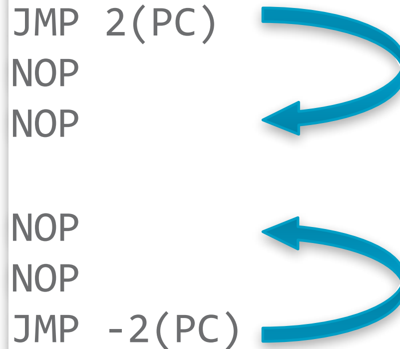
```
package ...

var myvar int64
```

Branches

Jump to labels.



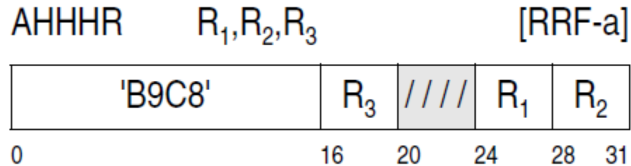
Jump relative to current position (a.k.a program counter or PC).



Useful in macros!

⚠ B & BR are aliases for JMP on some architectures

Missing instructions




```
// AHHHR R2,R3,R1
// R1 = R2 + R3 (high bits only)

// WORD (32 bits)
WORD $0xB9C83012

// BYTE (8 bits)
BYTE $0xB9; BYTE $0xC8
BYTE $0x30; BYTE $0x12
```

⚠ be careful of endianness (especially on mips64/ppc64)

Agenda

Introduction

Instructions

Functions & Stacks

Testing & Portability

Function declaration

sqrt_decl.go

```
func Sqrt(x float64) float64
```

package (optional)

function name

stack frame size

arguments size (optional)

sqrt_s390x.s

```
TEXT math·Sqrt(SB),$0-16
     FMOVD    x+0(FP), F0
     FSQRT    F0, F1
     FMOVD    F1, ret+8(FP)
     RET
```

Pseudo-registers

- FP: Frame Pointer
  - Points to the **bottom** of the argument list
  - Offsets are **positive**
  - Offsets must include a name, e.g. `arg+0(FP)`
- SP: Stack Pointer
  - Points to the **top** of the space allocated for local variables
  - Offsets are **negative**
  - Offsets must include a name, e.g. `ptr-8(SP)`
  - **Positive offsets refer to hardware register on `386/amd64`!**
- SB: Static Base
  - Named offsets from a global base
- PC: Program Counter
  - Used for branches
  - Offsets in **number of pseudo-instructions**

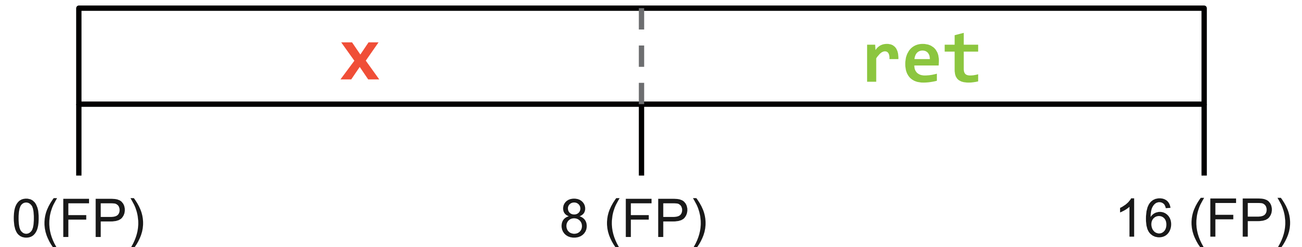
Calling convention

- All arguments passed on the stack
  - Offsets from FP
- Return arguments follow input arguments
  - Start of return arguments aligned to pointer size
- All registers are caller saved, except:
  - Stack pointer register
  - Zero register (if there is one)
  - G context pointer register (if there is one)
  - Frame pointer (if there is one)

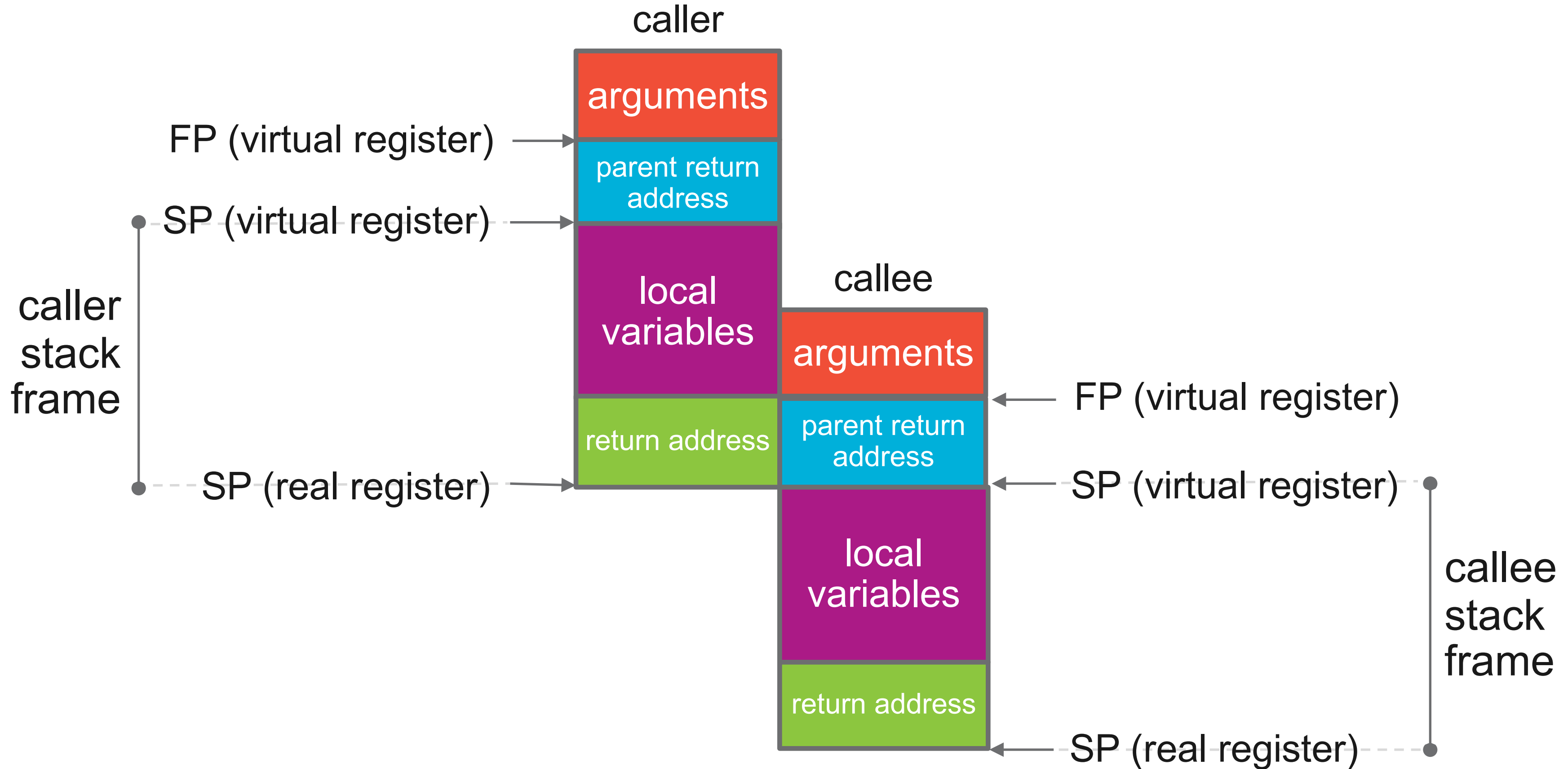
Function arguments

sqrt_s390x.s

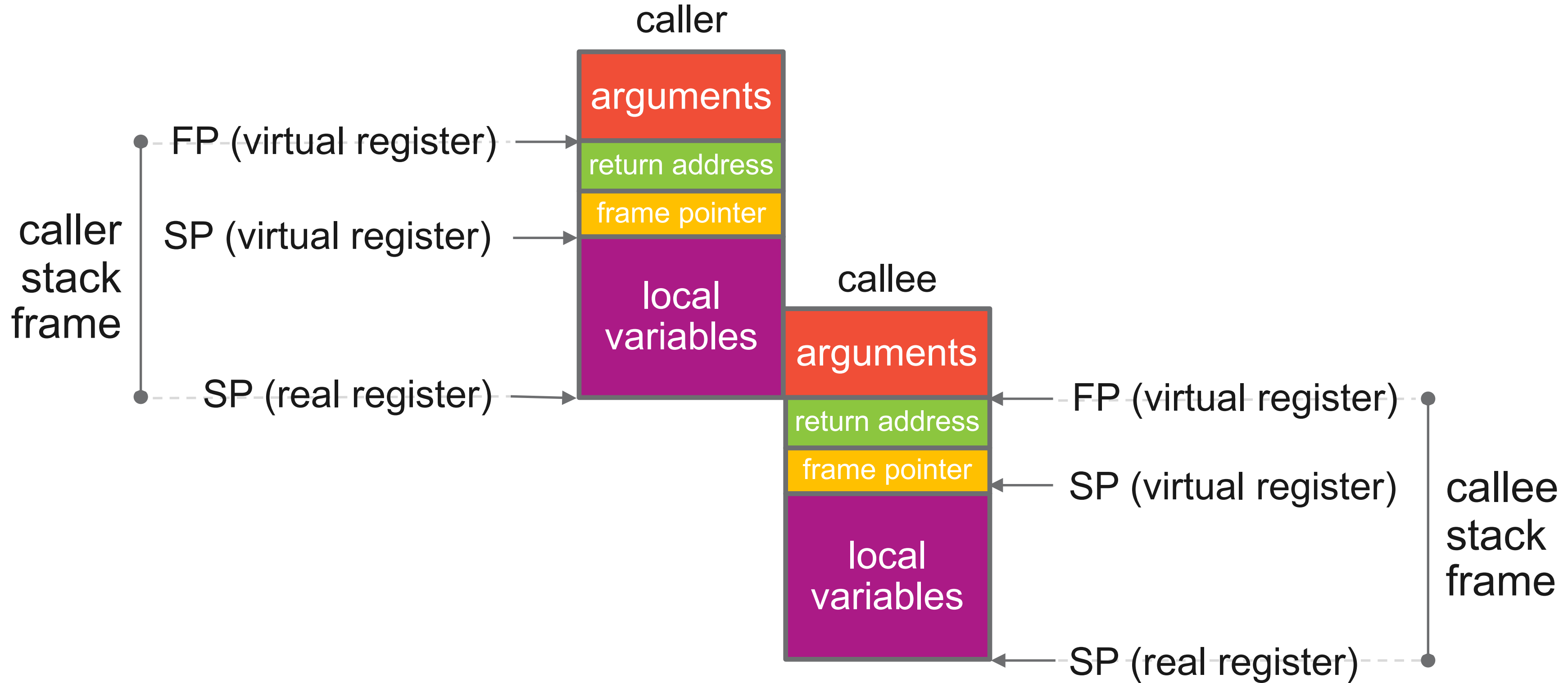
```
TEXT ·Sqrt(SB),$0-16
	FMOVD	x+0(FP), F0
	FSQRT	F0, F1
	FMOVD	F1, ret+8(FP)
	RET
```



Stack frame (link register, no frame pointer)

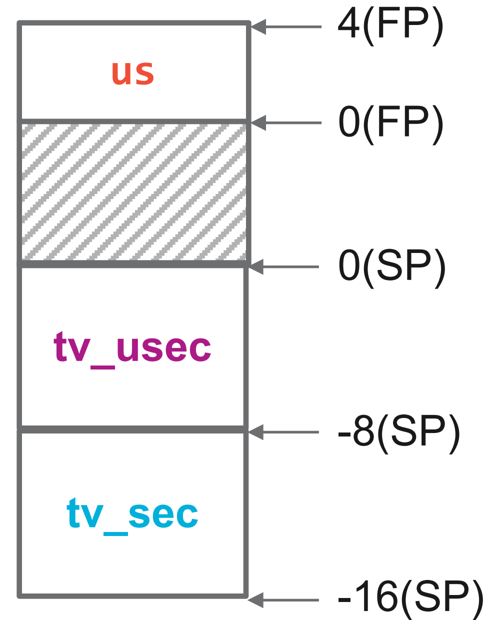


Stack frame (386/amd64, frame pointers enabled)



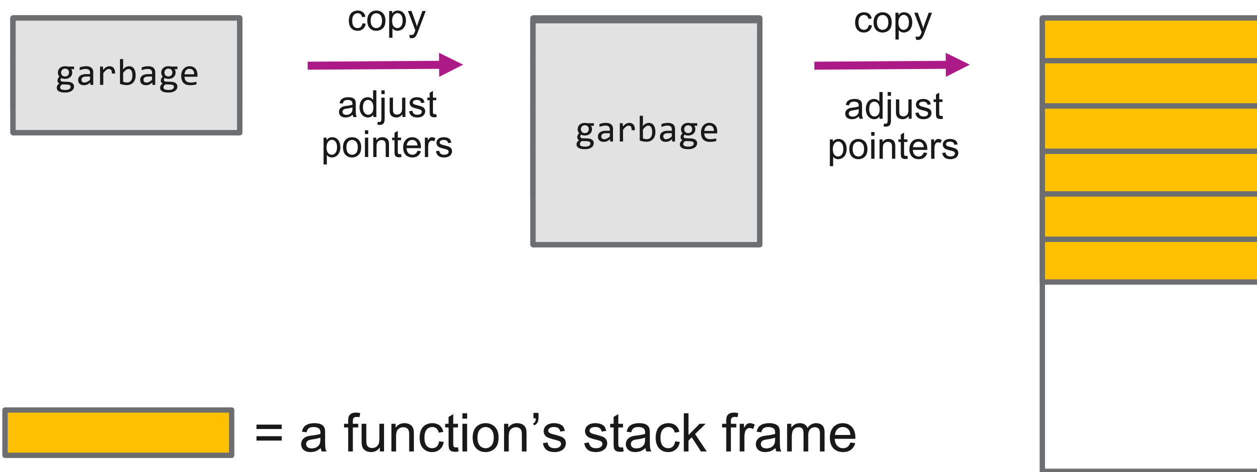
Local variables

```
// func usleep(usec int32)
TEXT ·usleep(SB),$16-4
    MOVL    $0, DX
    MOVL    usec+0(FP), AX
    MOVL    $1000000, CX
    DIVL    CX
    MOVQ    AX, tv_sec-16(SP)
    MOVQ    DX, tv_usec-8(SP)
    // select(0, 0, 0, 0, &tv)
    ...
    MOVQ    $tv-16(SP), R8
    MOVL    $23, AX
    SYSCALL
    RET
```



⚠ here SP is the pseudo-register, not the hardware register!

Stack growth



Flags

- NOSPLIT: don't insert a stack check
  - Reduces function call overhead
  - Limits size of stack
  - Use on leaf functions (unless a large stack is needed)
- NOFRAME: don't allocate a stack frame
  - Function must be a leaf
  - Function must be declared with a stack size of 0 (i.e. TEXT `...,NOFRAME,$0-...`)
  - No frame pointer (or return address on link register architectures) saved

```
TEXT ·Sqrt(SB),NOSPLIT|NOFRAME,$0-16
```

Escape analysis

stubs.go

```go
package runtime

// memmove copies n bytes from "from" to "to".
// in memmove_*.s
//go:noescape
func memmove(to, from unsafe.Pointer, n uintptr)
```

memmove_ppc64x.s

```
TEXT runtime·memmove(SB), NOSPLIT|NOFRAME,$0-24
    MOVD    to+0(FP), R3
    MOVD    from+8(FP), R4
    MOVD    n+16(FP), R5
    ...
    RET
```

Agenda

Introduction

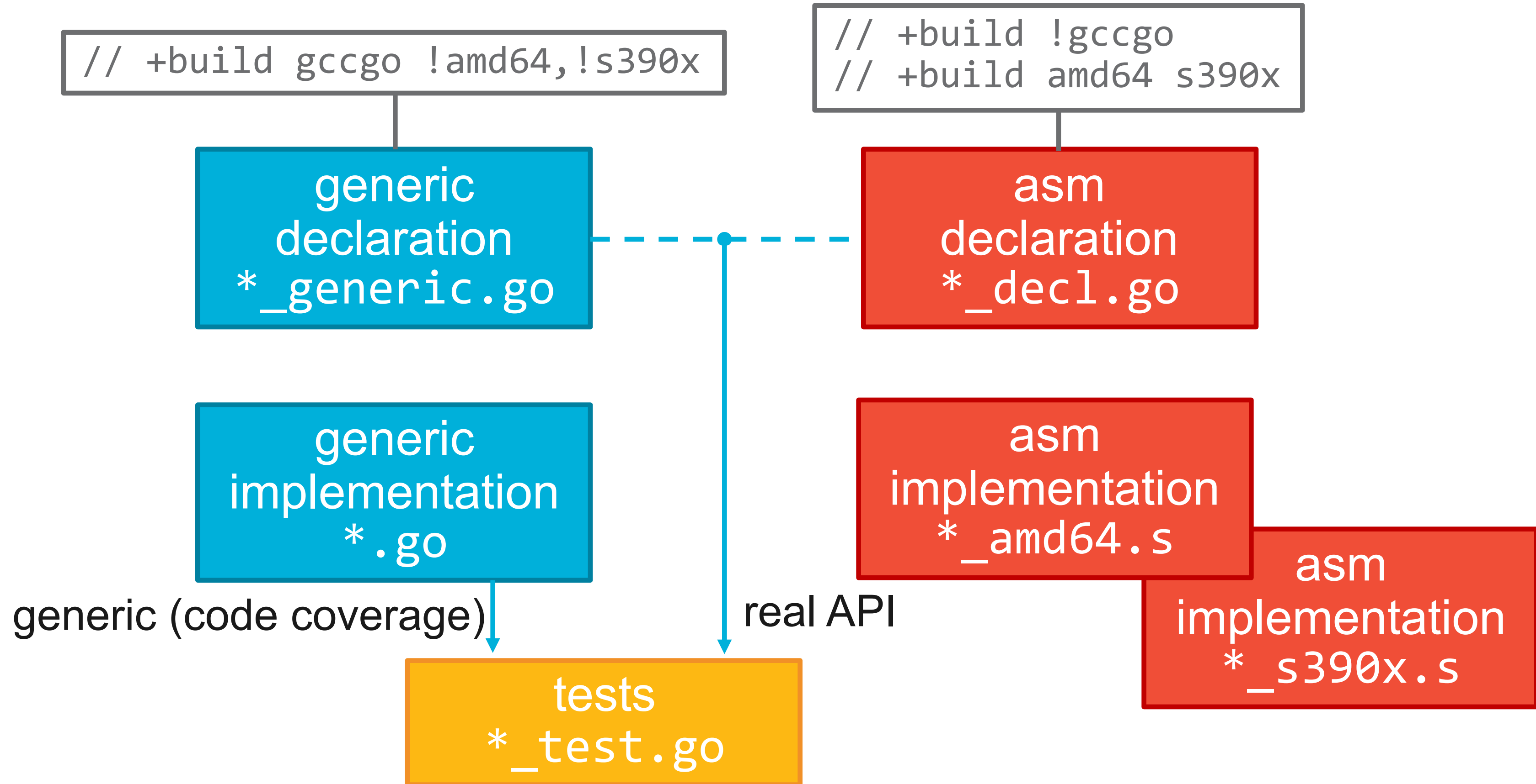
Instructions

Functions & Stacks

Testing & Portability

Code layout

example: https://github.com/mundaym/vector



Thanks for listening!

Go gophers © Renee French, used under Creative Commons license

Static data

```
// Package-level data
DATA  math·pi+0(SB)/8,$3
GLOBL math·pi(SB),RODATA,$8
```

```
// File-private data is appended with ‘<>’
DATA  text<>+0(SB)/8,$"Hello Wo"
DATA  text<>+8(SB)/8,$"rld!\n"
GLOBL text<>(SB),RODATA,$16
```