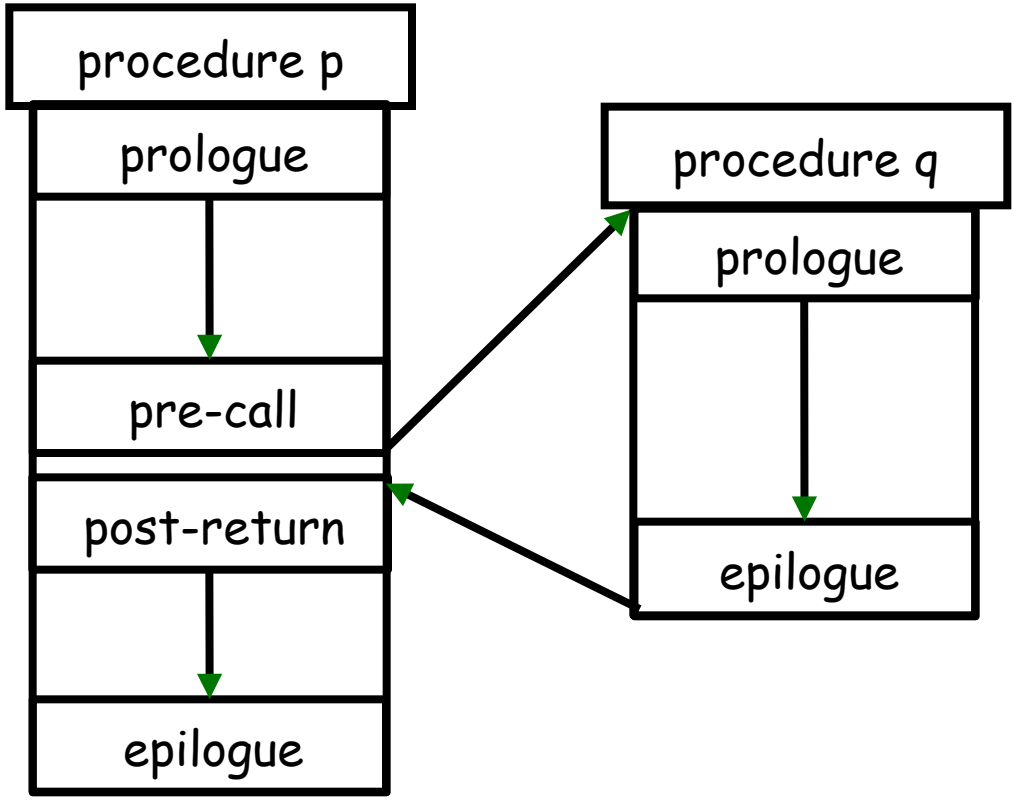
Function Calls

CS132

March 21, 2011

What does each part do?



ia32 Architecture Review

8 32-bit "general purpose" registers

```
%eax, %ebx, %ecx, %edx
%esi, %edi, %ebp, %esp
```

[8 floating-point registers, treated as a stack]

Caller-save: %eax, %ecx, %edx

Prologue

On entry

```
pushl %ebp
movl %esp, %ebp //
subl $16, %esp   // whatever space we want
```

On exit (return value already in `%eax or st0`)

```
leave  // movl %ebp, %esp
       // popl %ebp
ret
```

Prologue (with -fomit-frame-pointer)

On entry

```
subl $16, %esp   // whatever space we want
```

On exit (return value already in `%eax or st0`)

```
addl $16, %esp
ret
```

cdecl

Default for C and C++ (32-bit code)

```
foo(a,b,c)
```

```
pushl c
pushl b
pushl a
call foo
Remove a,b,c from stack
```

Alternative cdecl

Gcc tends to generate code like the following.

What else must be done to make this work?

```
foo(a,b,c)
```



```
movl c, 8(%esp)
movl b, 4(%esp)
movl a, 0(%esp)
call foo
```

Stdcall (Win32)

Callee removes arguments from stack too.

```
foo(a,b,c)
```

```
pushl c
pushl b
pushl a
call foo
```

Fastcall

Optional in Dos/Windows code

```
foo(a,b,c)
```

```
pushl c
movl b, %edx
movl a, %ecx
call foo
Remove c from stack
```

Exercise

How might we

- Pass an int to a function (by value)?
- Pass an int to a function (by reference)?
- Pass a char to a function?
- Return a char from a function?
- Pass a struct to a function?
- Return a struct from a function?

Alpha Architecture

GP Registers

32 general-purpose registers (`$r0` -- `$r31`)

`$0`	return value (caller-save)
`$[1-8]`	caller-save registers
`$[9-14]`	callee-save registers
`$15 = $fp`	frame pointer or callee-save
`$[16-21]`	integer arguments (caller-save)
`$[22-25]`	more caller-save registers
`$26 = $ra`	return address
`$27 = $pv`	address of procedure being called
`$28 = $at`	volatile scratch register
`$29 = $gp`	global pointer
`$30 = $sp`	stack pointer
`$31`	always zero

Alpha Call and Return

Caller:

```
lda $pv, foo
jsr $ra, 0($pv)
ldgp $gp, 0($ra)
```

Subroutine call to label "`foo`"

Address of the following instruction is put in $ra.

Program counter is set to address of label `foo`.

No delay slot

After code returns, fix up `$gp` using the address of the `ldgp`

Subroutine Call and Return

Callee prologue:

```
foo:  ldgp $gp, 0($pv)
      lda $sp, -16($sp)
      stq $26, 0($sp)
```

No requirement for stack allocation

Generally return address will go on stack, so normally we'll need at least 16 bytes (as above)

Need space/code for callee-save registers that are modified

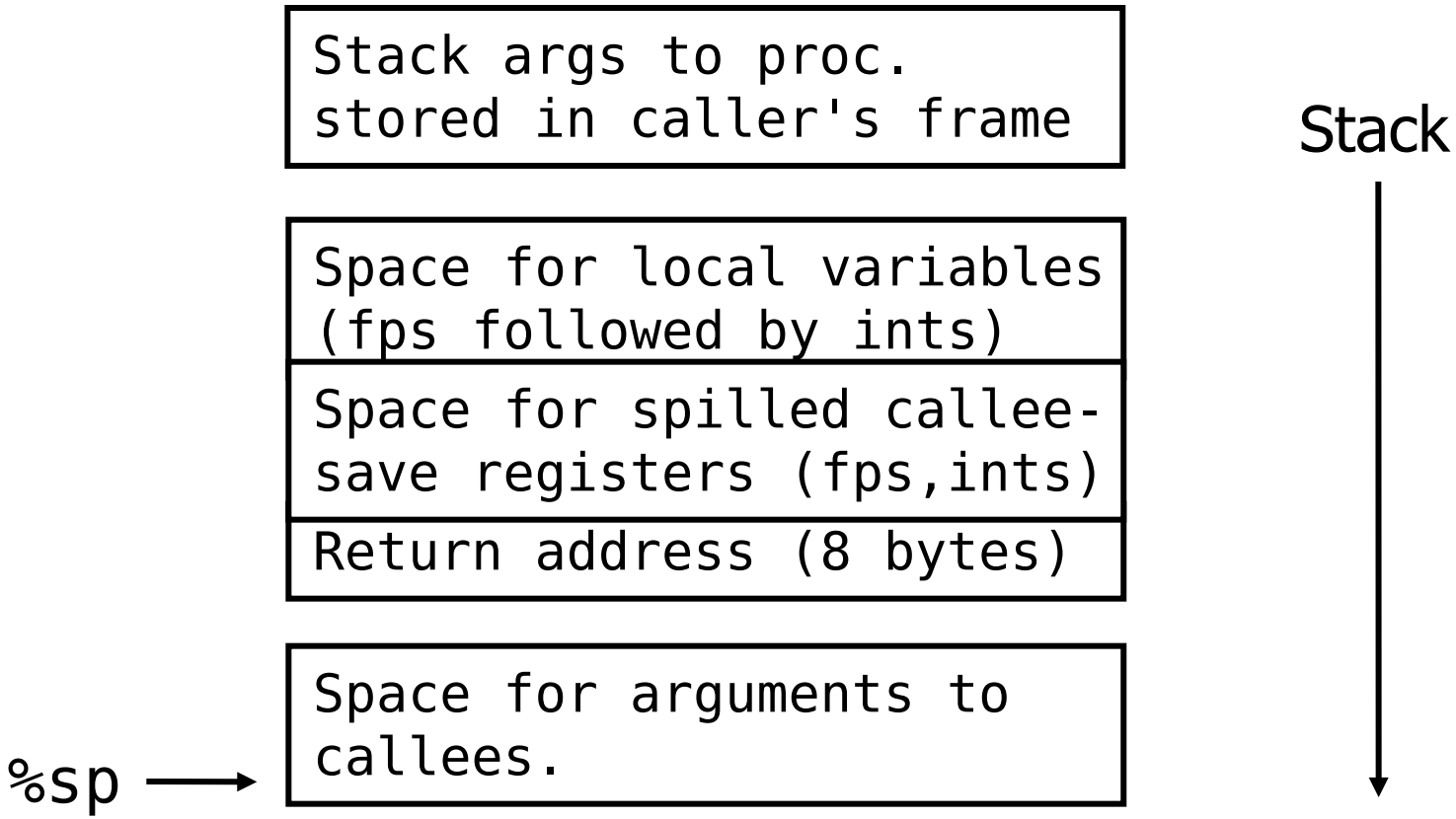
Subroutine Call and Return

Callee epilogue:

```
ldq $26, 0($sp)
lda $sp, 16($sp)
ret $31, ($26)
```

Again, no delay slots

Alpha Stack Frame



SPARC Architecture Review

Registers

32 general-purpose registers (`%r0` -- `%r31`)

Each register has at least two names

`%r0`	`= %g0`	always zero
`%r[1-7]`	`= %g[1-7]`	global data
`%r[8-13]`	`= %o[0-5]`	arguments to other routines
`%r14 = %o6`	`= %sp`	stack pointer
`%r15`	`= %o7`	ret. addr. of callees
`%r[16-23]`	`= %l[0-7]`	local data
`%r[24-29]`	`= %i[0-5]`	arguments to current routine
`%r30 = %i6`	`= %fp`	frame pointer
`%r31`	`= %i7`	ret. addr for current routine

Register Windows

The SPARC actually has ≥128 registers

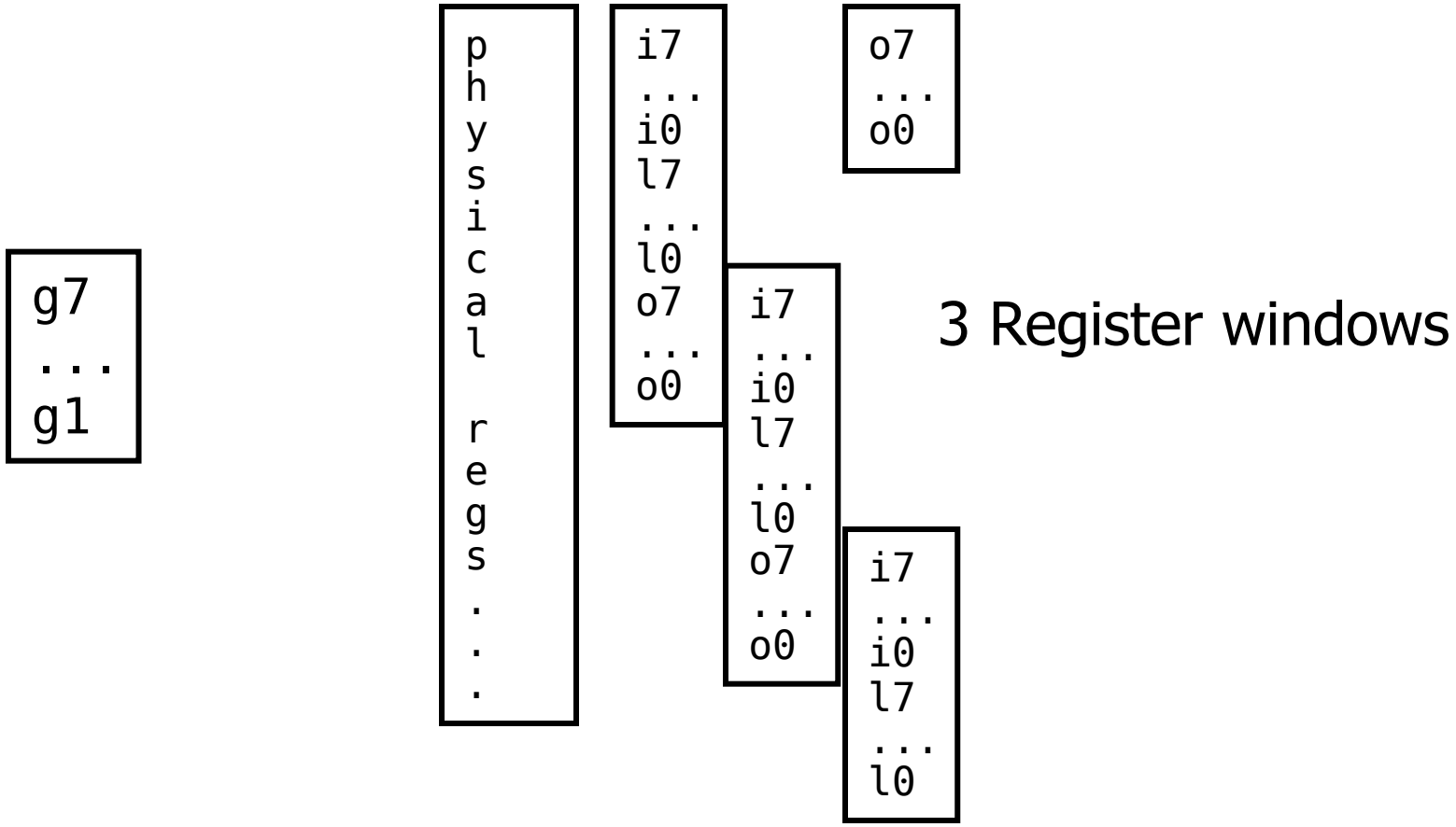
- But only 32 are visible at any one time.
- To access the rest, one must use *register windows*

Upon executing `save` instruction:

- "out" registers now accessible with "in" names
- 32 new registers allocated for "local" and "out"
- "global" registers unaffected

Opposite action on `restore`

Pictorial View



Subroutine Call and Return

Caller:

```
call foo
nop
```

Subroutine call to label "`foo`"

Address of the call is put in `%o7`.

Program counter is set to address of label `foo`.

Recall: SPARC has delay slots, here filled by `nop`.

Subroutine Call and Return

Callee prologue:

```
foo:  save %sp, -96, %sp
```

Instruction `save` updates register window

Also acts as an add (subtracts 96 from `%sp`)

- Arguments are taken from *old* window, result is put in *new* window. Afterwards old %sp is called %fp.
- In general, there will be some other number here depending on how much space the stack frame requires.

Subroutine Call and Return

Callee epilogue:

```
ret
restore
```

Instruction ret expands to `jmpl %i7+8, %g0`

Why `%i7` and not `%o7` ? Why `8` ?

Instruction `restore` is in the delay slot; pops register window.

Parameter Passing

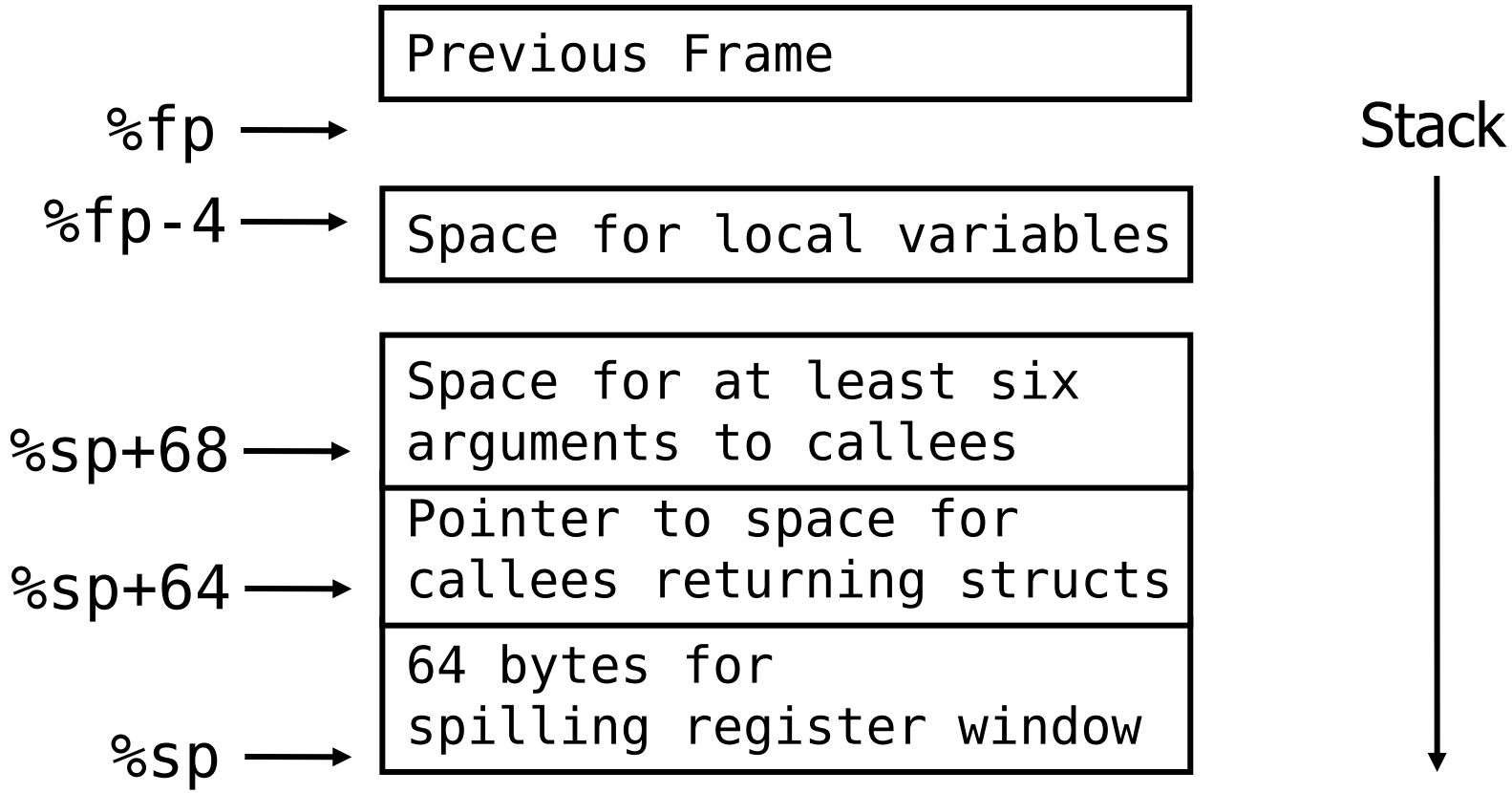
The first 6 arguments are passed in registers

- %o0 through %o5

Remaining arguments are passed on the stack.

- In the caller's stack frame!
- Caller: addresses %sp+92, %sp+96, ...
- Callee: addresses %fp+92, %fp+96, ...

SPARC Stack Frame



Sparc Stack

Minimum stack frame size of 96 bytes.

64 + 4 + 24 + 4*overflow args + local storage

64+4+24 = 92, but ...