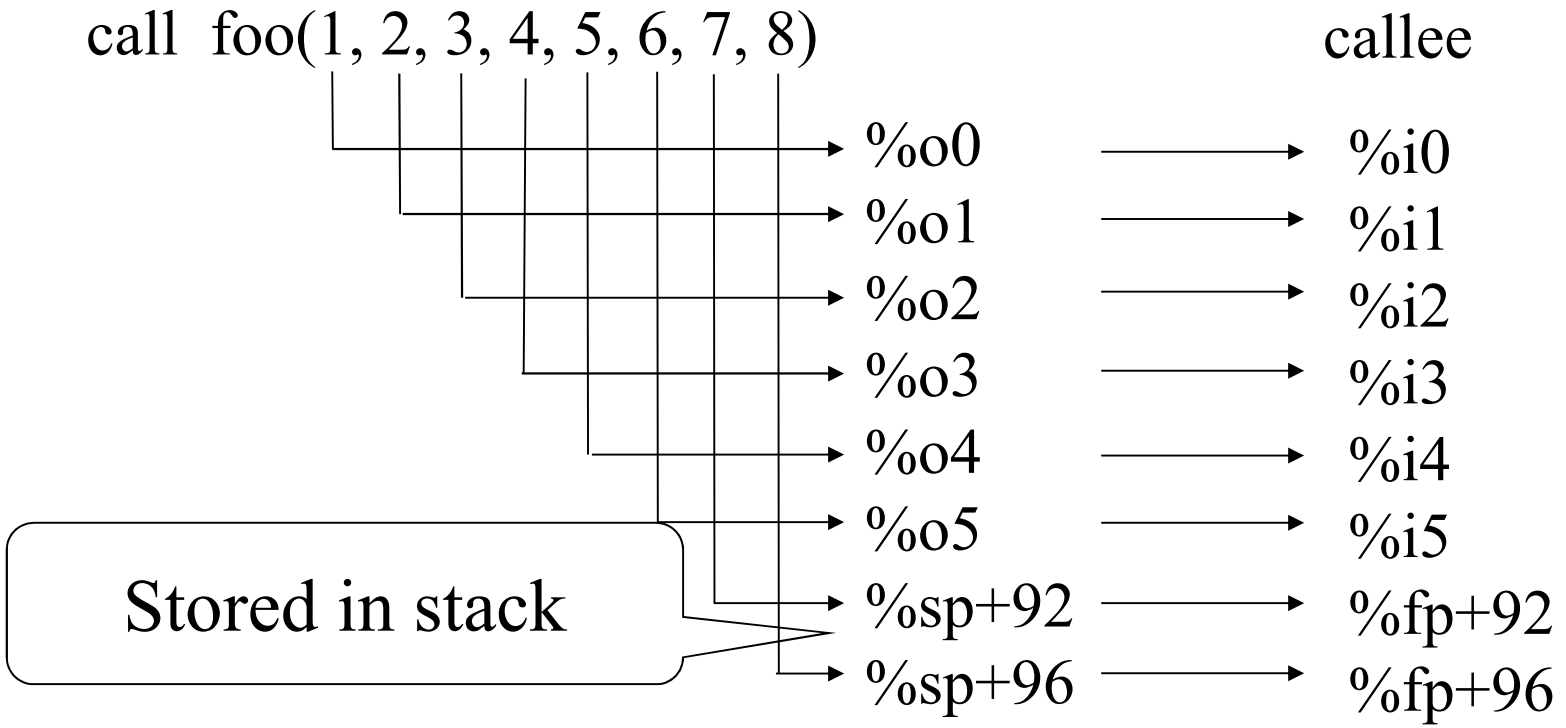
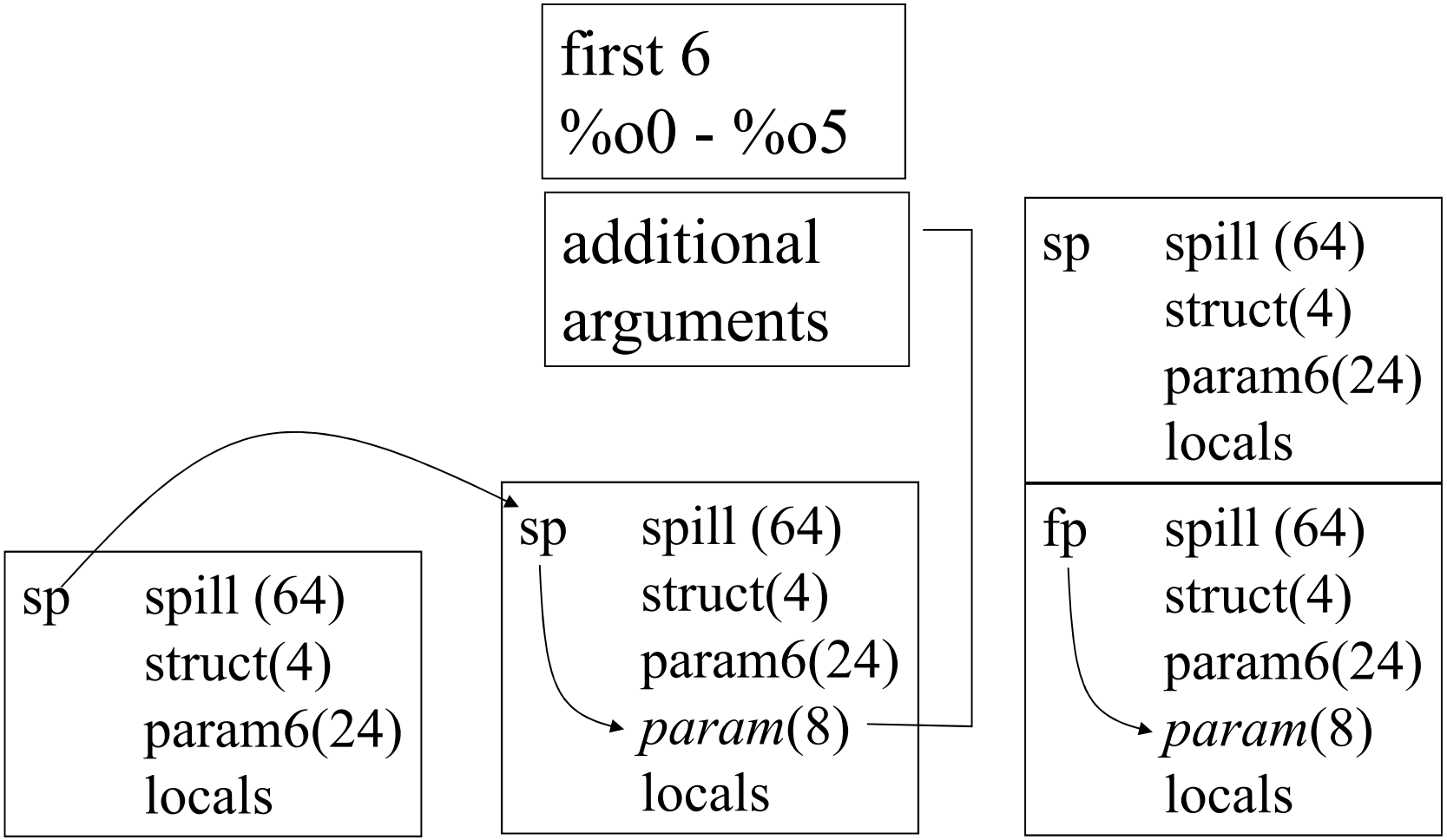
Using more than 6 arguments: store in stack

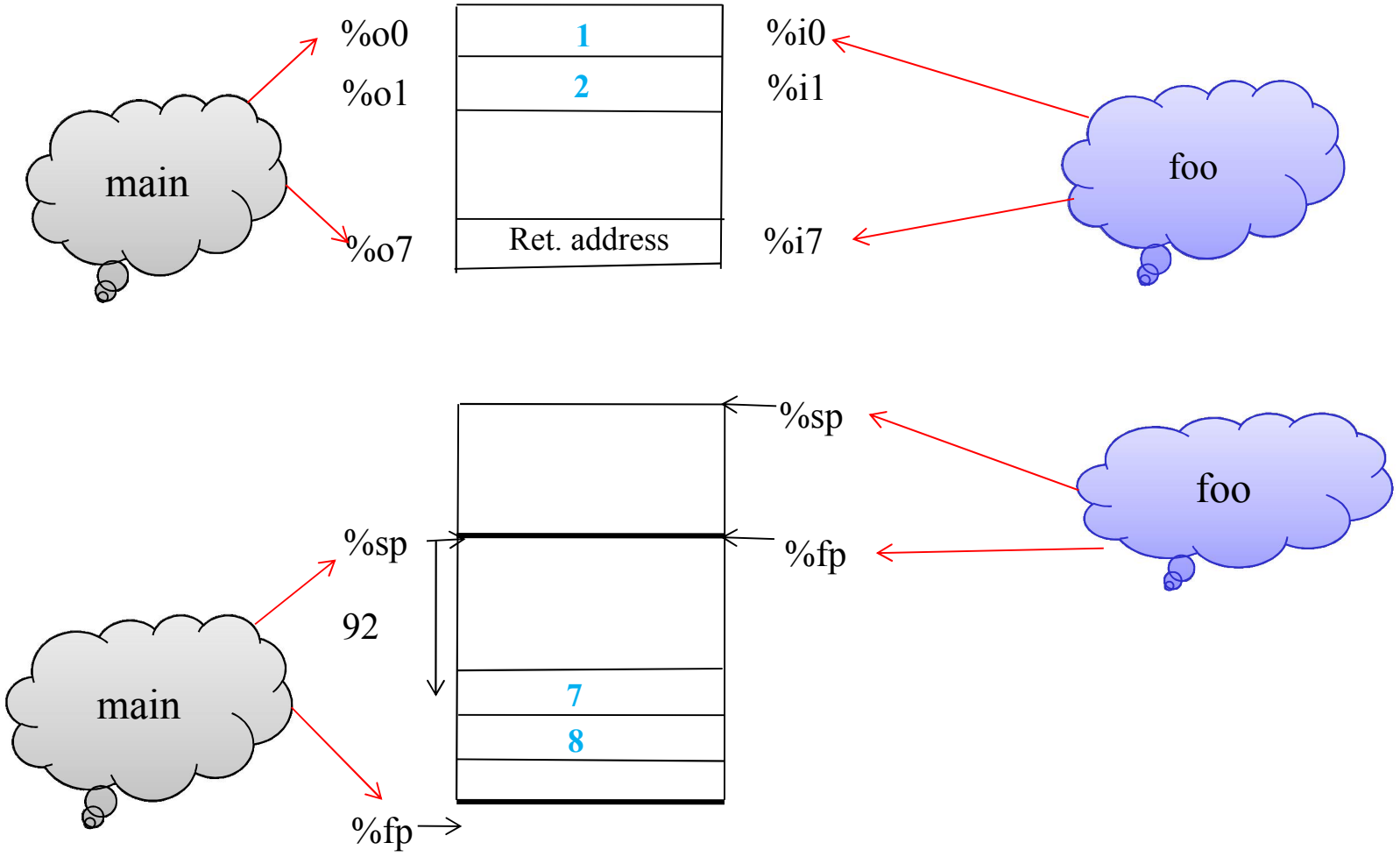
```
int foo(int a1, int a2, int a3, int a4, int a5, int a6, int a7, int a8){
   return a1+ a2+ a3+ a4+ a5+ a6+ a7+ a8;
}
```



Stack usage for additional arguments



- Main and foo


%o0
%o1
%o7
main
1
2
Ret. address
%i0
%i1
%i7
foo
%sp
%fp
foo
%sp
92
%fp
main
7
8

Code Example (1)

```
        .global  main
main:   save     %sp, -104, %sp    ! -(92 + 8) & -8
        mov  8, %o0   ! store args in reverse
        st       %o0, [%sp + 96]
        mov      7, %o0
        st       %o0, [%sp + 92]
        mov      6, %o5
        mov      5, %o4
        mov      4, %o3
        mov      3, %o2
        mov      2, %o1
        call     foo
        mov      1, %o0
        mov      1, %g1
        ta       0
```

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

        ld       [%fp + 96], %o0
        ld       [%fp + 92], %o1
        add      %o0, %o1, %o0
        add      %i5, %o0, %o0
        add      %i4, %o0, %o0
        add      %i3, %o0, %o0
        add      %i2, %o0, %o0
        add      %i1, %o0, %o0
        ret
        restore %i0, %o0, %o0
```

Code example (2)

- Caller

```
main: save   %sp, -96, %sp
      add    %sp, -2*4 & -8, %sp
      mov      8, %o0
      st       %o0, [%sp + 96]
      mov      7, %o0
      st       %o0, [%sp + 92]
      mov      6, %o5
      mov      5, %o4
      mov      4, %o3
      mov      3, %o2
      mov      2, %o1
      call     foo
      mov      1, %o0
      sub   %sp, -2*4 & -8, %sp
```

- callee

```
foo:  save     %sp, −96, %sp

      ld       [%fp + 96], %o0
      ld       [%fp + 92], %o1
      add    %o0, %o1, %o0
      add     %i5, %o0, %o0
      add     %i4, %o0, %o0
      add     %i3, %o0, %o0
      add      %i2, %o0, %o0
      add     %i1, %o0, %o0

      ret
      restore     %i0, %o0,%o0
```

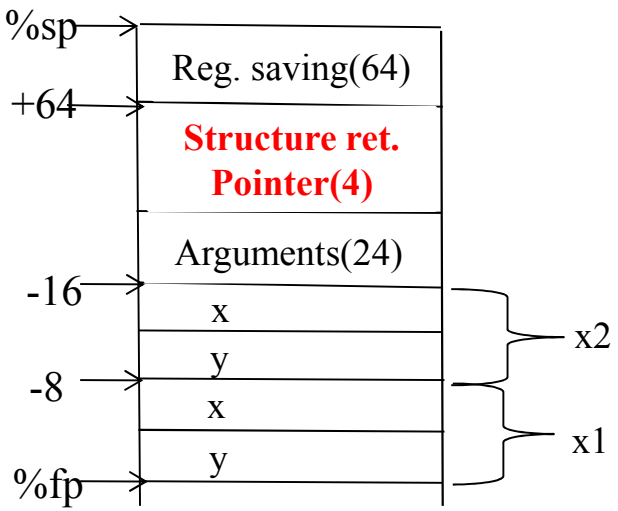
Return values

- Returning a 32-bit value: use caller's %o0 (callee's %i0)
- Returning structure
  - ✓ example

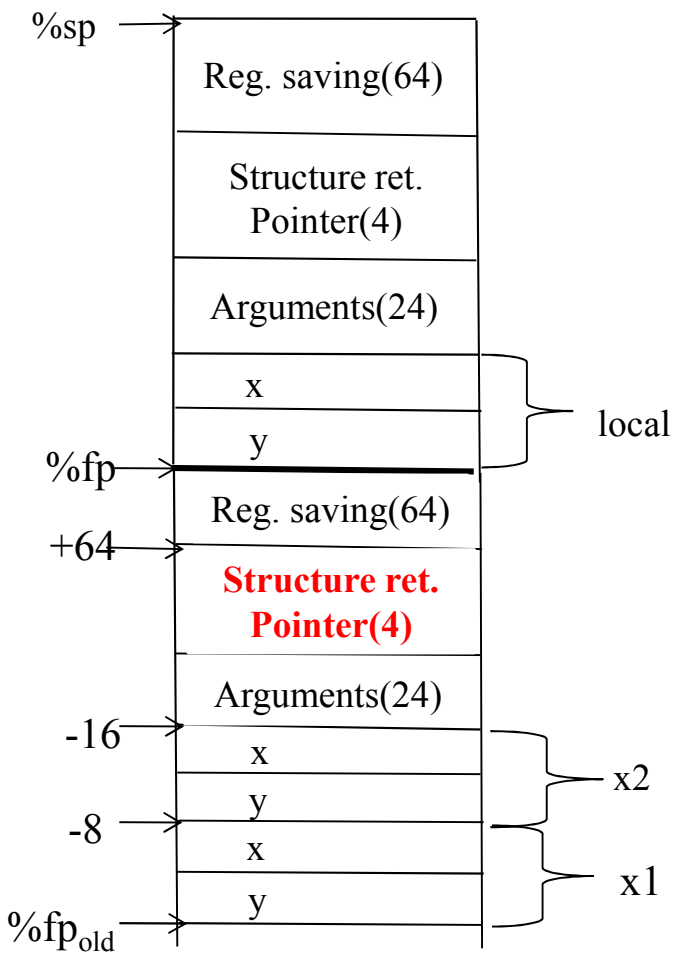
```
struct point { int x, y; };

struct point zero() {
  struct point local;
  local.x = 0;
  local.y = 0;
   return local;
}
```

```
main() {

   struct point x1, x2;

   x1 = zero();
   x2 = zero();
}
```

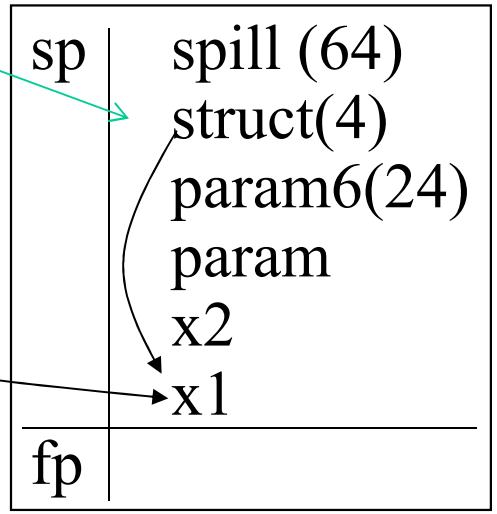
< in main >



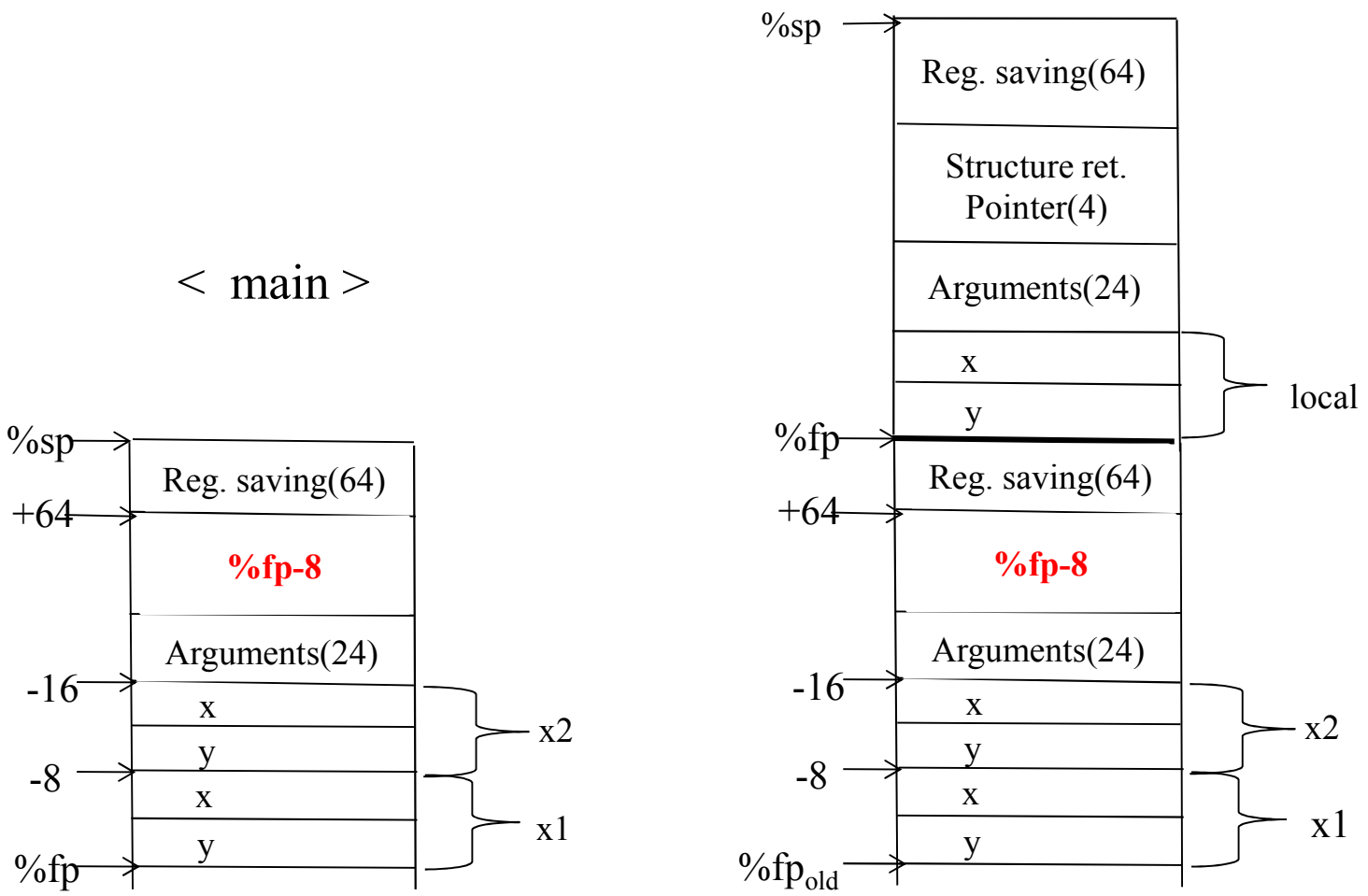
< subroutine zero is executed >



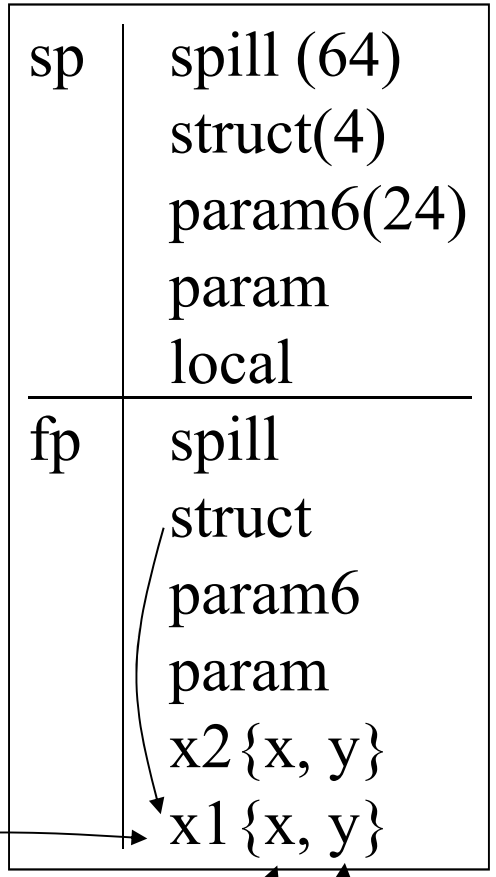
```
        !local variables
        x1 = -8
        x2 = -16

        .global main
main:   save    %sp, -112, %sp      ! –(64+4+24+16) &-8
        add     %fp, x1, %o0          ! address of point x1
        call    zero
        st      %o0, [%sp + 64]       ! saving the address
                                      ! return slot

        add     %fp, x2, %o0
        call    zero
        st      %o0, [%sp + 64]

        ret
        restore
```



- Before/after fist call instruction < subroutine zero is called>



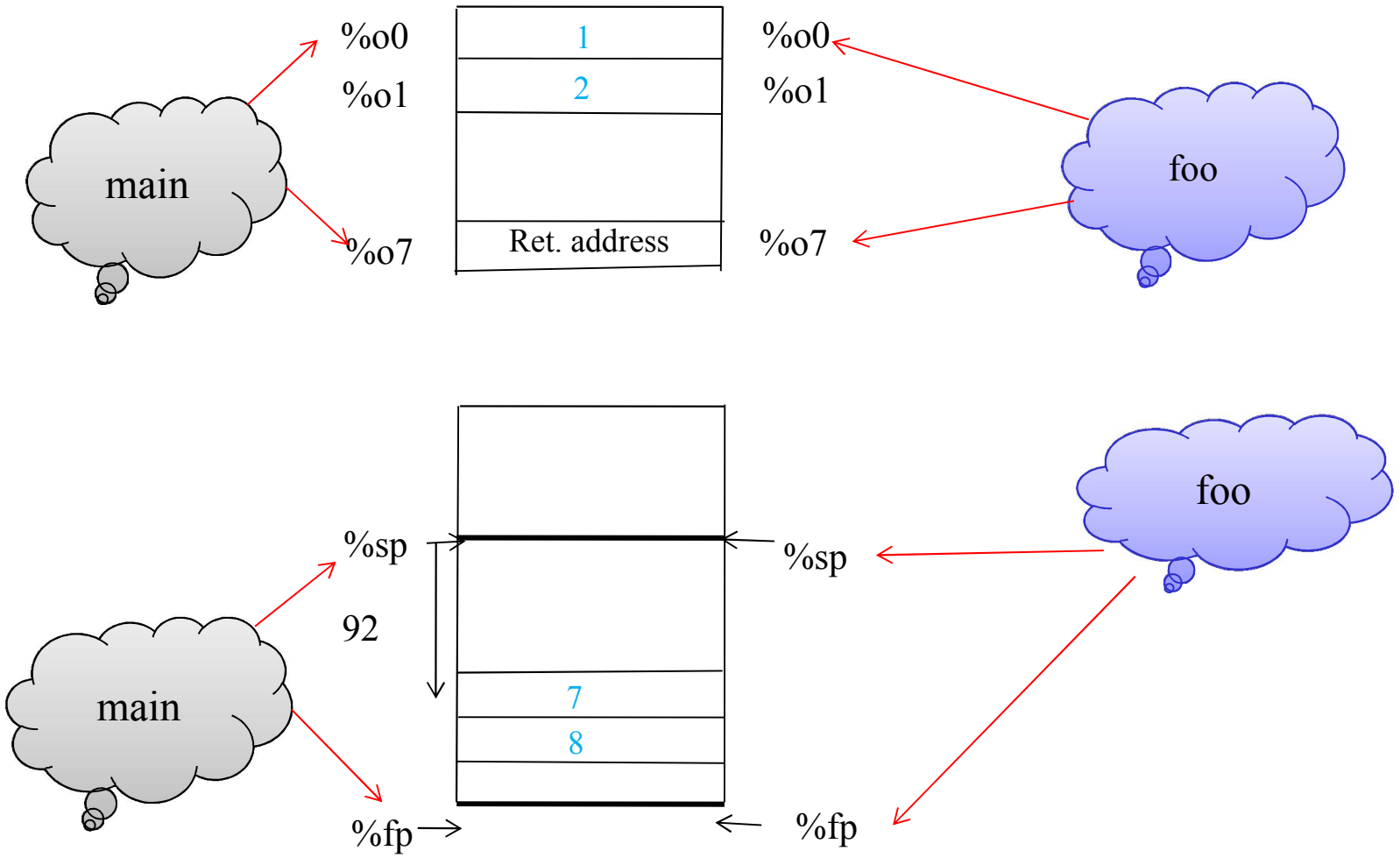
```
        !define structure point
        point_x = 0
        point_y = 4

        !align_of_point, 4 bytes
        !size_of_point, 8 bytes
        !local variables
        local = -8

        .global zero
zero:   save    %sp, -104, %sp
        ld      [%fp + 64], %o0          ! get pointer into %o0
        st      %g0, [%o0 + point_x]
        st      %g0, [%o0 + point_y]
        ret
        restore
```



Leaf subroutine

- A subroutine that does not call any other subroutine: A leaf node in a call graph
- Optimized leaf procedure/subroutine
  - No stack frame allocation
  - No save, restore instruction usage
  - Maintain CWP
  - Register usage restriction: %o0 - %o5, %g0, %g1
  - Return address: %o7 + 8
    - ✓ return instruction: retl (= jmpl %o7+8, %g0)

- Main and foo


%o0
%o1
%o7
main
1
2
Ret. address
%o0
%o1
%o7
foo
%sp
92
%fp
main
7
8
%sp
%fp
foo

Example

- Caller

```
main: save   %sp, -96, %sp
      add    %sp, -2*4 & -8, %sp
      mov       8, %o0
      st        %o0, [%sp + 96]
      mov       7, %o0
      st        %o0, [%sp + 92]
      mov       6, %o5
      mov       5, %o4
      mov       4, %o3
      mov       3, %o2
      mov       2, %o1
      call      foo
      mov       1, %o0
      sub   %sp, -2*4 & -8, %sp
```

- callee

```
foo:add  %o1, %o0, %o0
      add  %o2, %o0, %o0
      add  %o3, %o0, %o0
      add  %o4, %o0, %o0
      add  %o5, %o0, %o0
      ld   [%sp + 92], %o1
      add  %o1, %o0, %o0
      ld   [%sp + 96], %o1
      retl
      add  %o1, %o0, %o0
```

Pointer argument

- A pointer is an address (container)
- There is no pointer to register

```
swap(int *x, int *y){
    int t;
    t = *x;
    *x = *y;
    *y = t;
}
```

```
main() {
        int  i, j;
        i = 5;
        j = 7;
        swap(&i, &j);
        :
}
```

```
!  local variables
       x_s = -4        ! int x
       y_s = -8        ! int y

       .global main
main:   save    %sp, -104, %sp   !  -(92 + 8) & -8
       mov     5, %o0
       st      %o0, [%fp + x_s]        ! x = 5
       mov     7, %o0
       st      %o0, [%fp + y_s]        ! y = 7
       add     %fp, x_s, %o0      !pointer to x in %o0
       call    swap
       add     %fp, y_s, %o1      !pointer to y in %o1
       :
       ret
       restore
```

```
       ! a leaf subroutine
swap:   ld  [%o0], %o2
                        ! %o2 = x
       ld  [%o1], %o3
                        ! %o3 = y
       st      %o2, [%o1]
       retl
       st      %o3, [%o0]
```