# Registers and Stack Frames

Lecture 6

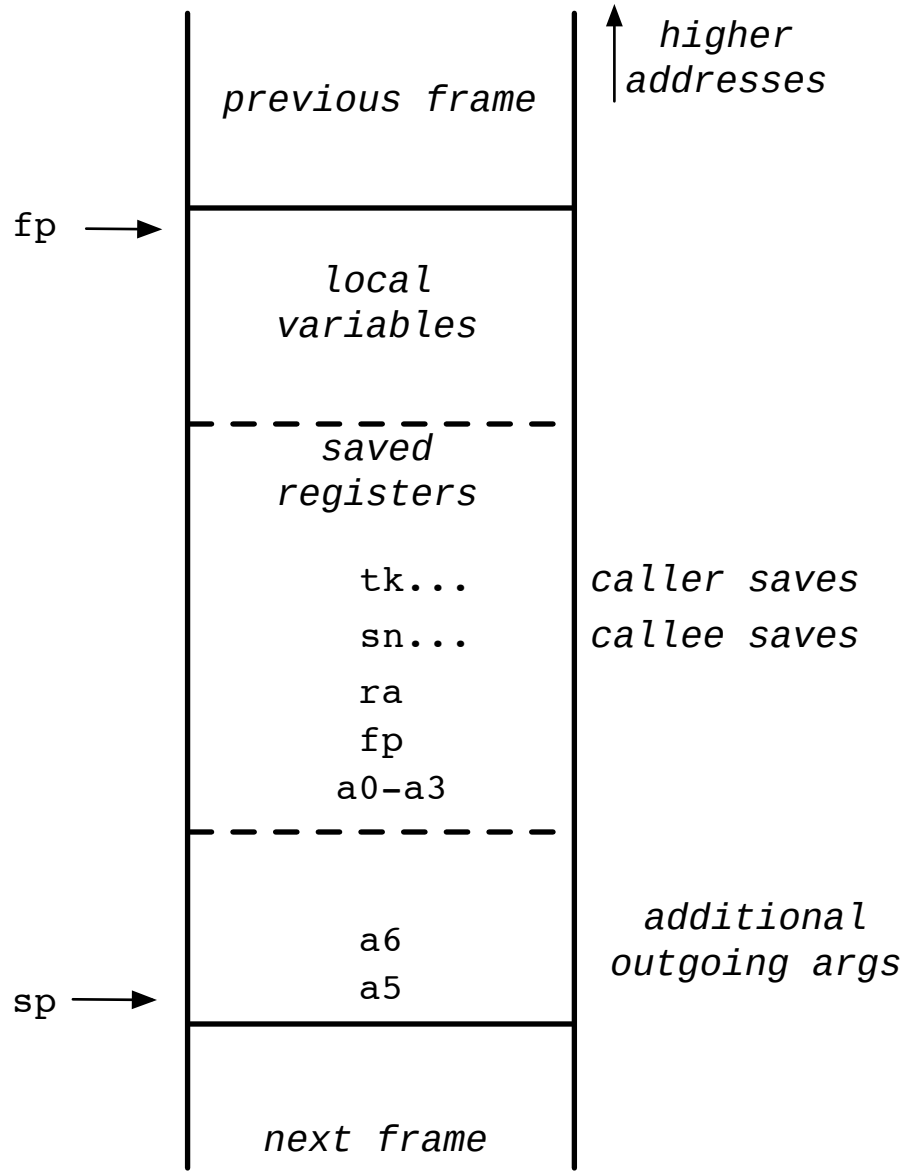
# Function calls and stack frames

- *function invocation contexts are strictly nested*
  - stack mechanism is adequate for managing parameters and locals
- *Information needed for function calls*
  - arguments
    - in registers
    - in memory (via stack)
  - function return address
  - return values?
  - local variables of the function
  - temp space for saving registers

# MIPS Registers

| Hardware | Name | Description |
|---|---|---|
| $0 | zero | constant zero |
| $1 | at | assembler temporary |
| $2-$3 | v0-v1 | function return value |
| $4-$7 | a0-a3 | incoming args |
| $8-$15 | t0-t7 | temporaries |
| $16-$23 | s0-s7 | callee-saves temporaries |
| $24-$25 | t8-t9 | temporaries |
| $26-$27 | k0-k1 | exception handling |
| $28 | gp | global data pointer |
| $29 | sp | stack pointer |
| $30 | s8,fp | frame pointer |
| $31 | ra | return address |

# Stack Frame Layout



# Assembly Example 1

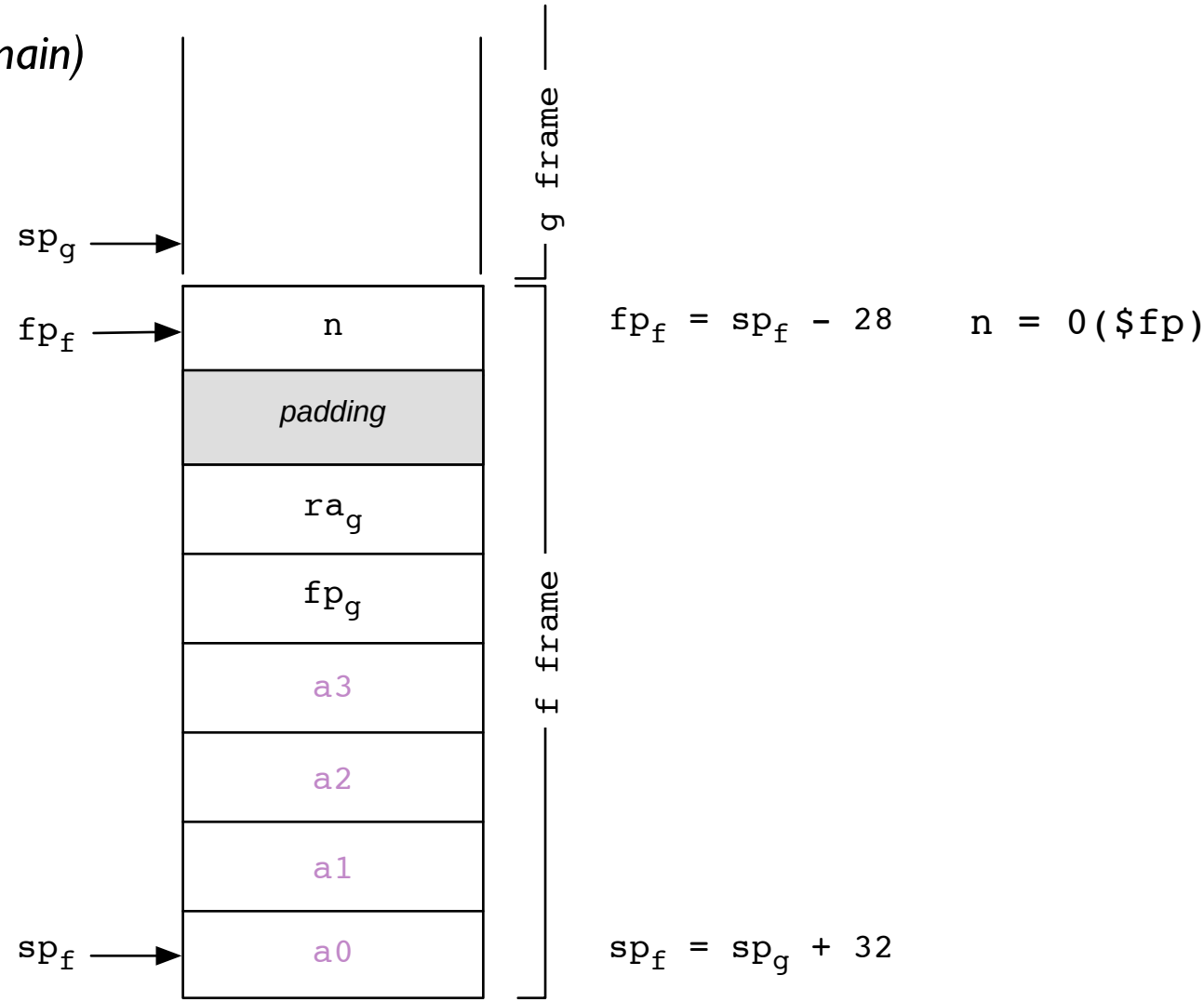
```
        .text
        .globl main
main:
        subu    $sp,$sp,24      # stack frame is 6 words
        sw      $ra,20($sp)     # save return address
        sw      $fp,16($sp)     # save frame pointer
        addu    $fp,$sp,20      # set new frame pointer

        li      $a0,6           # arg0: 6
        jal     f               # call f

        move    $a0,$v0         # arg0: return value from f
        jal     print_int       # call print_int

        lw      $ra,20($sp)     # restore return address
        lw      $fp,16($sp)     # restore frame pointer
        addu    $sp,$sp,24      # pop current frame
        j       $ra             # return to caller
```

# Assembly Example 2

```
        .text
f:
        subu    $sp,$sp,32      # allocate new frame (8 words)
        sw      $ra,20($sp)     # save return address
        sw      $fp,16($sp)     # save previous frame pointer
        addu    $fp,$sp,28      # define new frame pointer

        sw      $a0,0($fp)      # store n
        lw      $v0,0($fp)      # load n into v0
        bgtz    $v0,$L2         # branch to $L2 if n > 0
        li      $v0,1           # otherwise, return 1
        j       $L1             # jump to return code

$L2:    lw      $v1,0($fp)      # load n into v1
        subu    $a0,$v1,1       # a0 := v1 - 1  (= n - 1)
        jal     f               # call f recursively
        lw      $v1,0($fp)      # load n into v1
        mul     $v0,$v0,$v1     # v0 := f(n-1) * n

$L1:    lw      $ra,20($sp)     # restore return address
        lw      $fp,16($sp)     # restore frame pointer
        addu    $sp,$sp,32      # pop frame frome stack
        j       $ra             # return to caller (main)
```

# Example frame

*f called from within g (main)*




Stack pointer must be aligned on two word boundary.

# Escaping Variables

```
type tree = {key: string, left: tree, right: tree}

function prettyprint(tree: tree) : string =
  let
      var output := ""

      function write(s: string) =
        output := concat(output,s)

      function show(n: int, t: tree) =
        let function indent(s: string) =
             (for i := 1 to n
               do write(" ");
              output := concat(output,s); write("\n"))
         in if t=nil then indent(".")
            else (indent(t.key);
                  show(n+1,t.left);
                  show(n+1,t.left))
        end

   in show(0,tree);
      output
  end
```

An *escaping variable* is a local variable of a function that occurs in the body of a nested function definition. E.g. output, n.

# Computing Escaping Variables

- An escaping variable must be declared before the nested function that it appears in.

- The escaping property can be determined by comparing the function nesting depth of the variable's declaration with the nesting depths of its applied occurrences.

- The escaping property is recorded in the escape field (a `bool ref`) of the abstract syntax construct that declares it.
  - `VarDec` for variable declarations
  - ForExp for for loop index variables
  - `field` record for function parameters

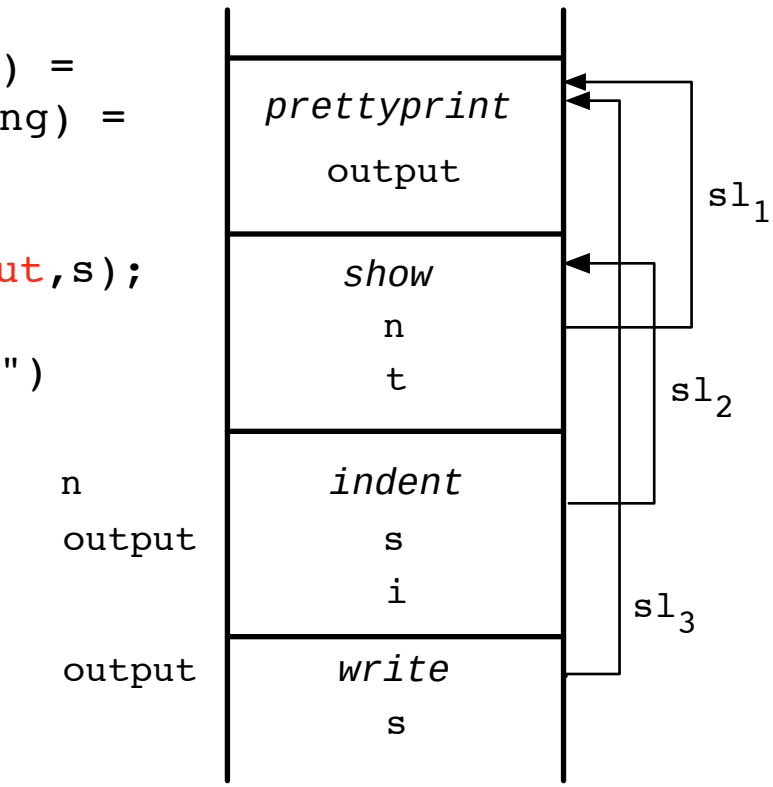
- An environment can be used to record function nesting depth and escape field ref for each variable within its scope.

# Static Links

- To access an escaping variable while executing the body of a nested function that accesses it, we will use a *static link*.
- A static link is the frame pointer of a frame (function activation record) for the latest call of the function statically enclosing the function currently being called.
- The static link will be passed as an added parameter, conventionally as the first parameter (a0).
- To access an escaping variable more than one nesting level deep, we need to follow a chain of static links.

# Static Links Example

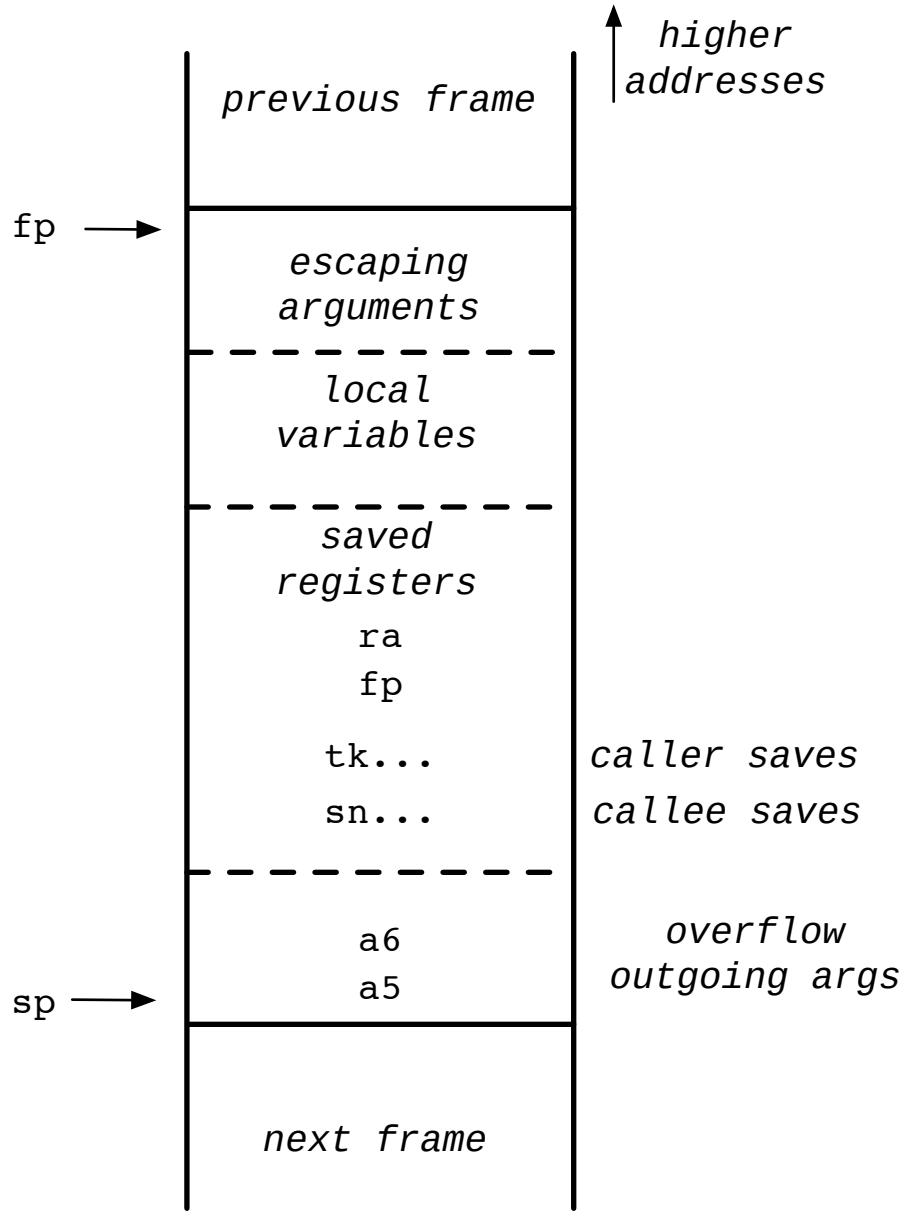
```
function prettyprint(tree: tree) : string =
  let
      var output := ""

      function write(s: string) =
        output := concat(output,s)

      function show(n: int, t: tree) =
        let function indent(s: string) =
             (for i := 1 to n
               do write(" ");
              output := concat(output,s);
              write("\n"))
         in if t=nil then indent(".")
            else (indent(t.key);
                  show(n+1,t.left);
                  show(n+1,t.left))
        end
   in show(0,tree);
      output
  end
```



# Frames for Tiger

```
signature FRAME =
sig

  type frame
  type access

  val newFrame : {name: Temp.label,  (* label of called fun *)
                  formals : (bool * string) list} -> frame
    (* formals specified with an escape flag, and a description
     * string *)

  val name : frame -> Temp.label
    (* the function label of a frame *)
  val formals : frame -> access list
    (* access info for formal parameters *)
  val allocLocal : frame -> (bool * string) -> access

  (* more to come ... *)

end (* signature FRAME *)
```

# Frame Layout (2)



# Incoming Arguments

```
function f(x0,x1,x2,x3,y4,y5) =
  let function g(u: int): int =
        x0 + x2 + u    /* x0 and x2 escaping */
   in g(3)
  end
```

Arguments `x0,...,x3` start in registers `$a0,...,$a3`.
They are moved to temp registers or frame slots:

```
x0:$a0  ⇒  0($fp)    -- escaping
x1:$a1  ⇒  t17
x2:$a2  ⇒  -4($fp)   -- escaping
x3:$a3  ⇒  t18
```

Arguments `y4,y5` are stored in previous frame:

```
y4: 4($fp)
y5: 8($fp)
```