Leaf and Non-Leaf Procedures

A *leaf procedure* is one that doesn't all any other procedures.

A *non-leaf procedure* is one that does call another procedure.

Non-leaf procedures pose an additional, but simple, challenge; we make procedure calls by executing a jump-and-link instruction:

```
      jal   procedure_0  # puts PC+4 into $ra for return
```

But, if procedure_0 also makes a call, say

```
      jal   procedure_1  # puts PC+4 into $ra for return
```

then the original return address just got overwritten… the effect is fascinating…

Preserving the Return Address

Non-leaf procedures must back up the value of their return address before making a call to another procedure:

```
       addi   $sp, $sp, -4       # make room on stack
       sw     $ra, 0($sp)        # save return address
```

And they must restore the return address before they attempt to return:

```
       lw     $ra, 0($sp)        # retrieve return address
       addi   $sp, $sp, 4        # pop it off the stack
```

Failure to do this will almost certainly lead to a catastrophic runtime failure.

The safest way to do this is to back up the address immediately when the procedure is entered, and to restore it immediately before the return is executed. Of course, you must keep careful track of the stack pointer during all of this…

Factorial: First Version

```
###################################################################
# Returns factorial of parameter.
#
# Pre:
#    $a0 stores N
# Post:
#    $v0 stores N!
#
# Modifies: $t0, $t1, $v0, $a0
#
fac1:
        li    $t0, 1                     # check for base case
        bgt   $a0, $t0, recurse
        li    $v0, 1                     # if so, set $v0
        jr    $ra                        #         and return

recurse:
        move  $t1, $a0                   # save N
        addi  $a0, $a0, -1               # calc N-1 for recursive call
        jal   fac1                       # calc (N-1)!
        mul   $v0, $v0, $t1              # multiply that by N

        jr    $ra                        #    and return
```

Unfortunately, fac1 falls into an infinite loop when it's called with any value larger than 1 for $a0.

What went wrong?

```
fac1:
        li    $t0, 1                  # check for base case
        bgt   $a0, $t0, recurse
        li    $v0, 1                  # if so, set $v0
        jr    $ra                     #         and return

recurse:
        move  $t1, $a0                # save N
        addi  $a0, $a0, -1            # calc N-1 for recursive call
        jal   fac1                    # calc (N-1)!
        mul   $v0, $v0, $t1           # multiply that by N

        jr    $ra                     #     and return
```

Making the recursive call overwrites the original return address with the address of *what*?

And the effect of that is….? An infinite loop in a pgm with no loops.

And the moral of that is….? Back up `$ra` before a call in a non-leaf proc.

Factorial: Second Version

```
fac2:
        li    $t0, 1                 # check for base case
        bgt   $a0, $t0, recurse
        li    $v0, 1                 # if so, set return value
        jr    $ra                    #         and return

recurse:
        move  $t1, $a0               # save N
        addi  $a0, $a0, -1           # calc N-1 for recursive call
        addi  $sp, $sp, -4           # save return address on stack
        sw    $ra, ($sp)
        jal   fac2                   # calc (N-1)!
        mul   $v0, $v0, $t1          # multiply that by N

        lw    $ra, ($sp)             # restore return address
        addi  $sp, $sp, 4
        jr    $ra                    #     and return
```

Unfortunately, fac2 returns 32 when called with $a0 == 6.

What went wrong?

```
fac2:
. . .

recurse:
        move  $t1, $a0                  # save N
        addi  $a0, $a0, -1              # calc N-1 for recursive call
        addi  $sp, $sp, -4              # save return address on stack
        sw    $ra, ($sp)
        jal   fac2                      # calc (N-1)!
        mul   $v0, $v0, $t1             # multiply that by N

        lw    $ra, ($sp)                # restore return address
        addi  $sp, $sp, 4

        jr    $ra                       #    and return
```

During the recursive call, the previous contents of `$t1` and `$a0` are overwritten.

Moral: before making a call, back up your registers as necessary.

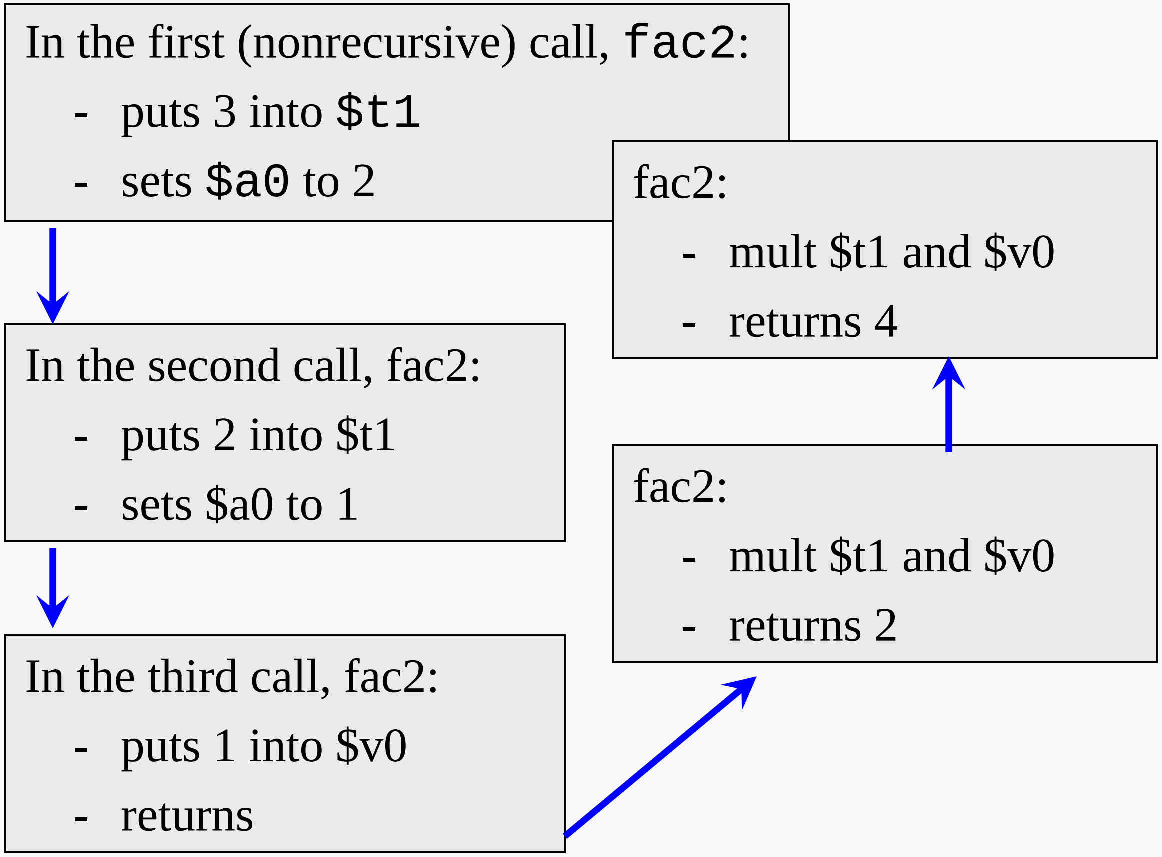
What went wrong: Details

Let's say we call this with `$a0` set to 3:

```
fac2:
. . .

recurse:
   move  $t1, $a0
   addi  $a0, $a0, -1
   addi  $sp, $sp, -4
   sw    $ra, ($sp)
   jal   fac2
   mul   $v0, $v0, $t1

   lw    $ra, ($sp)
   addi  $sp, $sp, 4
   jr    $ra
```



In the first (nonrecursive) call, `fac2`:

- puts 3 into `$t1`
- sets `$a0` to 2

In the second call, fac2:

- puts 2 into $t1
- sets $a0 to 1

In the third call, fac2:

- puts 1 into $v0
- returns

fac2:

- mult $t1 and $v0
- returns 2

fac2:

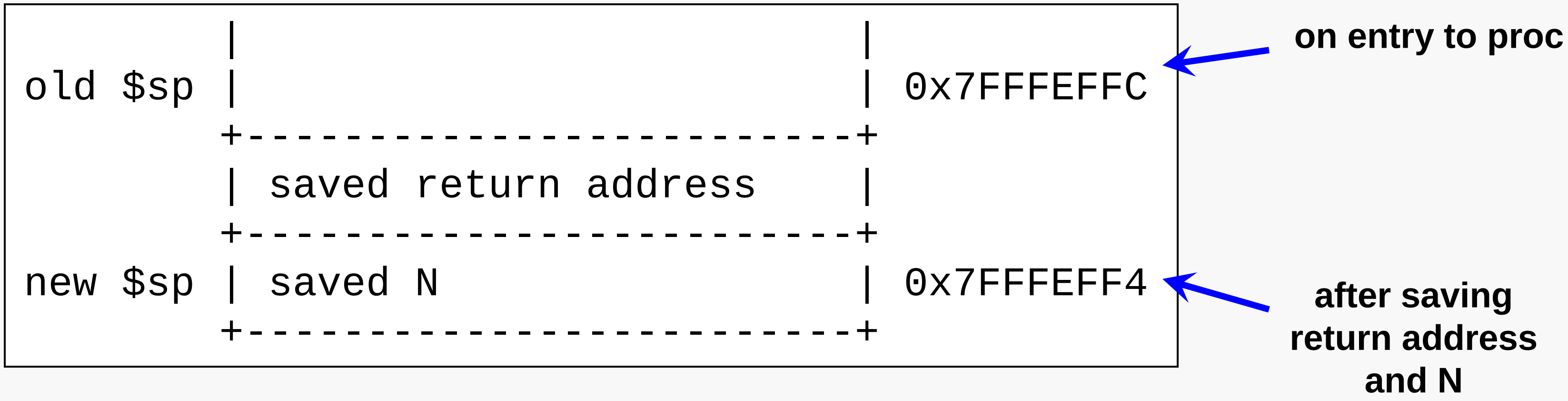
- mult $t1 and $v0
- returns 4

Moral: before making a call, back up your registers as necessary.

Factorial: Stack Organization

In order to fix the execution of the recursive factorial procedure, we need to use the stack to save values that would otherwise be overwritten when a recursive call takes place.

Here's one idea for organizing the stack:



```
         |                          |
old $sp  |                          | 0x7FFFEFFC
         +--------------------------+
         | saved return address     |
         +--------------------------+
new $sp  | saved N                  | 0x7FFFEFF4
         +--------------------------+
```

on entry to proc

after saving return address and N

Factorial: Third Version

```
fac3:
        li    $t0, 1                 # check for base case
        bgt   $a0, $t0, recurse
        li    $v0, 1                 # if so, set return value
        jr    $ra                    #         and return

recurse:
        addi  $sp, $sp, -8           # make room on stack for
        sw    $ra, 4($sp)            #       return address, and
        sw    $a0, 0($sp)            #       N

        addi  $a0, $a0, -1           # calc N-1 for recursive call
        jal   fac3                   # calc (N-1)!

        lw    $t1, 0($sp)            # restore N from stack
        mul   $v0, $v0, $t1          # multiply (N-1)! by N

        lw    $ra, 4($sp)            # restore return address from
                                     #    stack
        addi  $sp, $sp, 8            #    and restore stack pointer
        jr    $ra                    #    and return
```

Factorial: Stack Trace

Say we call factorial(3):

```
        |                          |
old $sp |                          | 0x7FFFEFFC
        +--------------------------+
        | saved $ra to main        |
        +--------------------------+
        | saved 3                  | 0x7FFFEFF4
        +--------------------------+
        | saved $ra to main        |
        +--------------------------+
new $sp | saved 2                  | 0x7FFFEFEC
        +--------------------------+
```

on first call

(not recursive)

on second call

(recursive)

Third call triggers base case and returns with $v0 == 1

Saved value of N (2) is retrieved from stack and multiplied to $v0; 2*1 is returned to from second call.

Saved value of N (3) is retrieved from stack and multipled to $v0; 3*2*1 is returned from first call.