

# CSCI 210: Computer Architecture

## Lecture 13: Procedures & The Stack

Stephen Checkoway

Oberlin College

Nov. 1, 2021

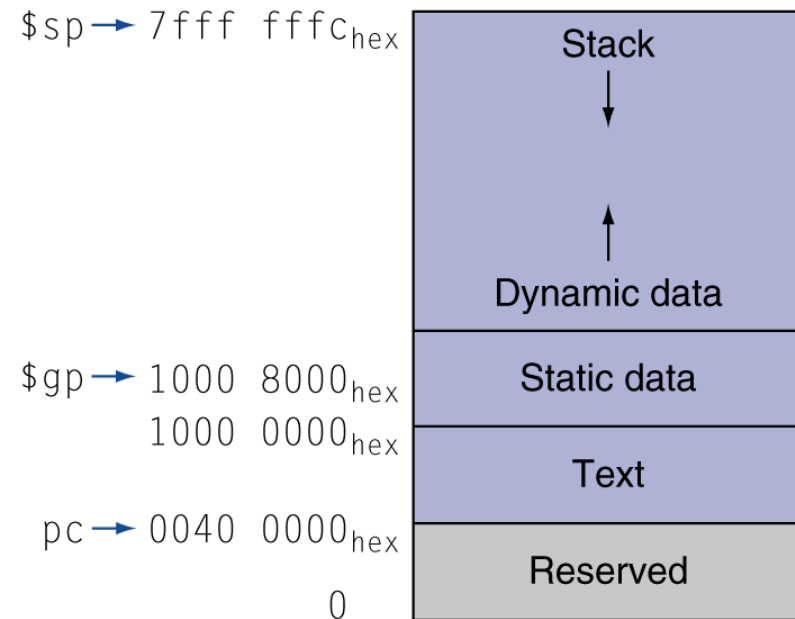
Slides from Cynthia Taylor

# Announcements

- Problem Set 4 due Friday
- Lab 3 due Sunday
- Office hours: Tuesday 13:30–14:30

# Last Class

- Spill and fill registers whose values we need to preserve
  - Return address \$ra
  - Any saved registers \$s0–\$s7
  - Any temporary registers over function calls
  - Any time we have more variables than fit in registers



# Leaf function Example

- C code:

```
int leaf_example(int g, int h, int i, int j) {  
    int f = (g + h) - (i + j);  
    return f;  
}
```

- Arguments g, ..., j in \$a0, ..., \$a3
- f in \$s0 (hence, need to save \$s0 on stack)
- Result in \$v0

# Leaf Procedure Example

- MIPS code:

- leaf\_example:

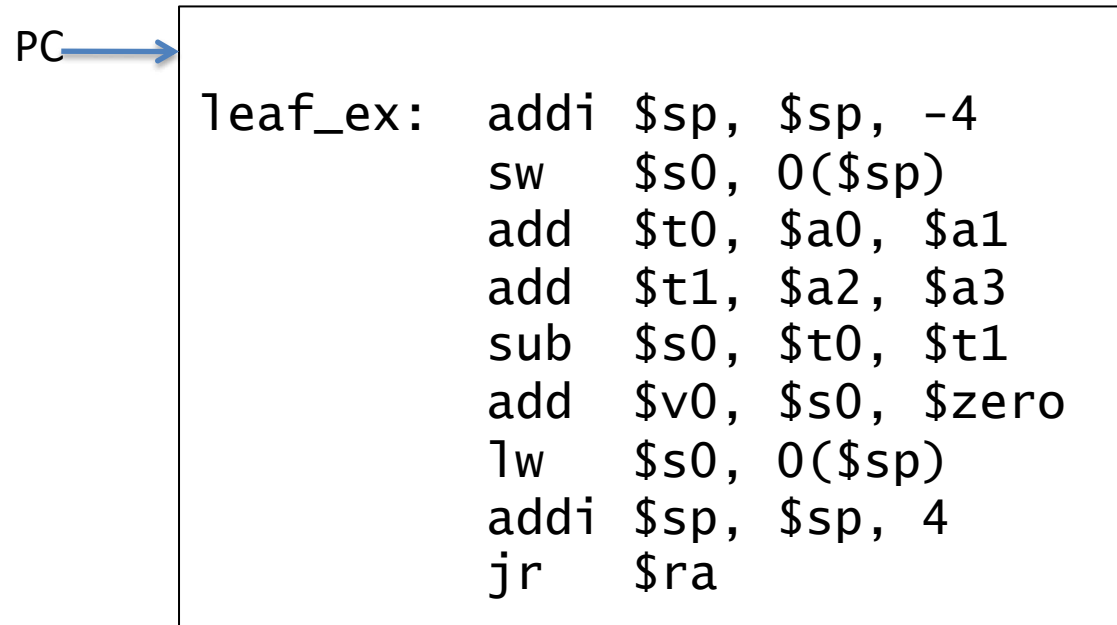
```
addi $sp, $sp, -4  
sw   $s0, 0($sp)
```

```
add  $t0, $a0, $a1  
add  $t1, $a2, $a3  
sub  $s0, $t0, $t1  
move $v0, $s0
```

```
lw   $s0, 0($sp)  
addi $sp, $sp, 4  
jr   $ra
```

```
int leaf_example(int g, int h,  
                 int i, int j) {  
    int f = (g + h) - (i + j);  
    return f;  
}
```

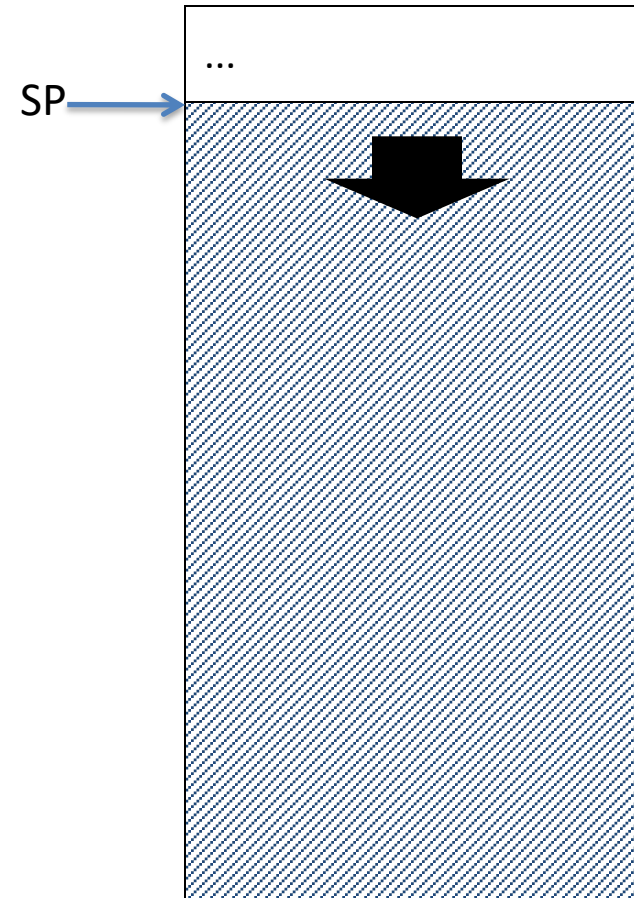
# Process Stack



Arguments g, ..., j in \$a0, ..., \$a3  
f in \$s0 (hence, need to save \$s0 on stack)  
Result in \$v0

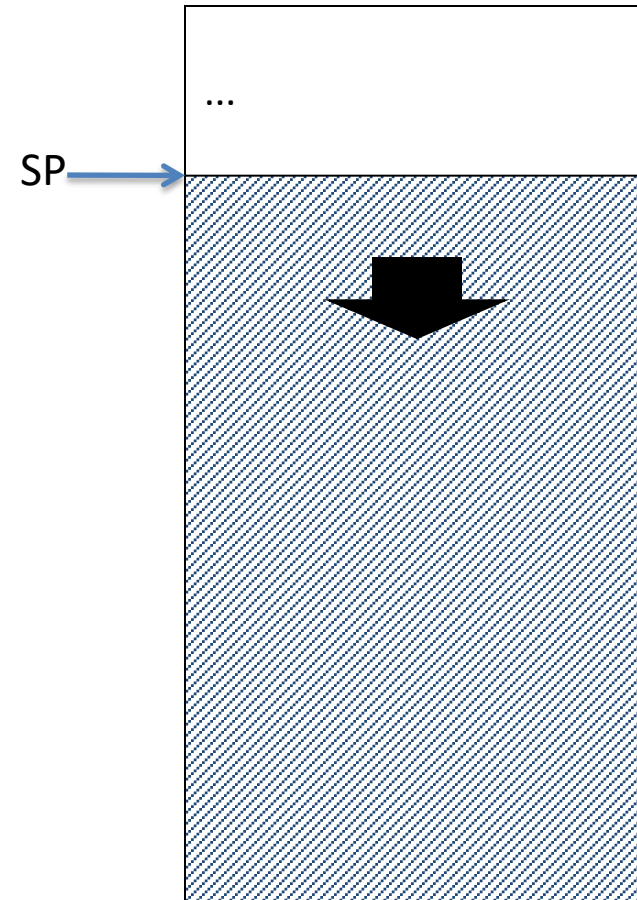
## Registers

\$s0: 25    \$a0: 5    \$a1: 2    \$a2: 3    \$a3: 1  
\$t0:        \$t1:        \$v0:



# Process Stack

```
PC → leaf_ex:  addi $sp, $sp, -4
                sw   $s0, 0($sp)
                add  $t0, $a0, $a1
                add  $t1, $a2, $a3
                sub  $s0, $t0, $t1
                add  $v0, $s0, $zero
                lw   $s0, 0($sp)
                addi $sp, $sp, 4
                jr   $ra
```



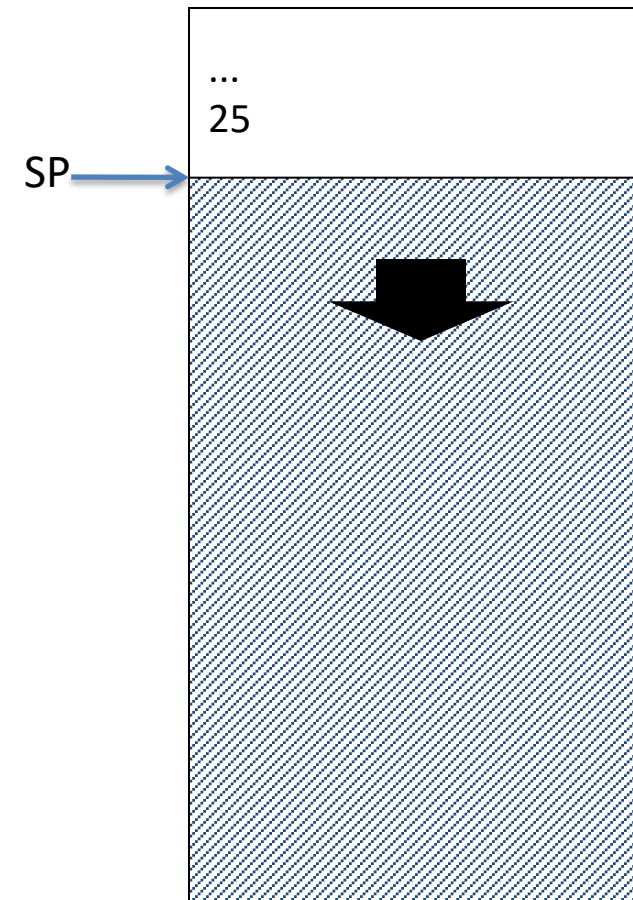
Arguments g, ..., j in \$a0, ..., \$a3  
f in \$s0 (hence, need to save \$s0 on stack)  
Result in \$v0

## Registers

\$s0: 25    \$a0: 5    \$a1: 2    \$a2: 3    \$a3: 1  
\$t0:        \$t1:        \$v0:

# Process Stack

```
leaf_ex:  addi  $sp, $sp, -4
          PC → sw   $s0, 0($sp)
          add  $t0, $a0, $a1
          add  $t1, $a2, $a3
          sub  $s0, $t0, $t1
          add  $v0, $s0, $zero
          lw   $s0, 0($sp)
          addi $sp, $sp, 4
          jr   $ra
```



Arguments g, ..., j in \$a0, ..., \$a3  
f in \$s0 (hence, need to save \$s0 on stack)  
Result in \$v0

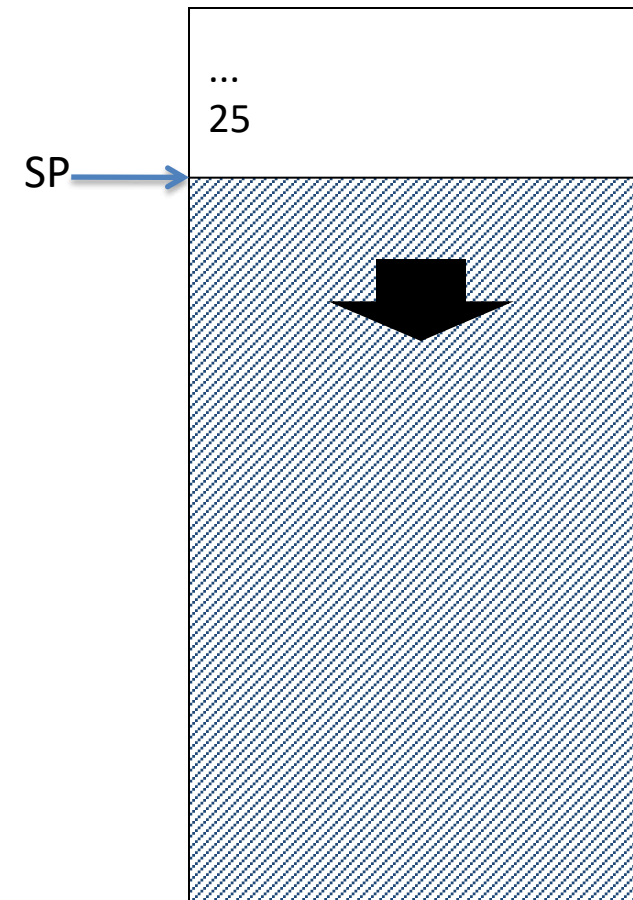
## Registers

\$s0: 25    \$a0: 5    \$a1: 2    \$a2: 3    \$a3: 1  
\$t0:        \$t1:        \$v0:



# Process Stack

```
leaf_ex:  addi $sp, $sp, -4
          sw   $s0, 0($sp)
          PC → add  $t0, $a0, $a1
          add  $t1, $a2, $a3
          sub  $s0, $t0, $t1
          add  $v0, $s0, $zero
          lw   $s0, 0($sp)
          addi $sp, $sp, 4
          jr   $ra
```



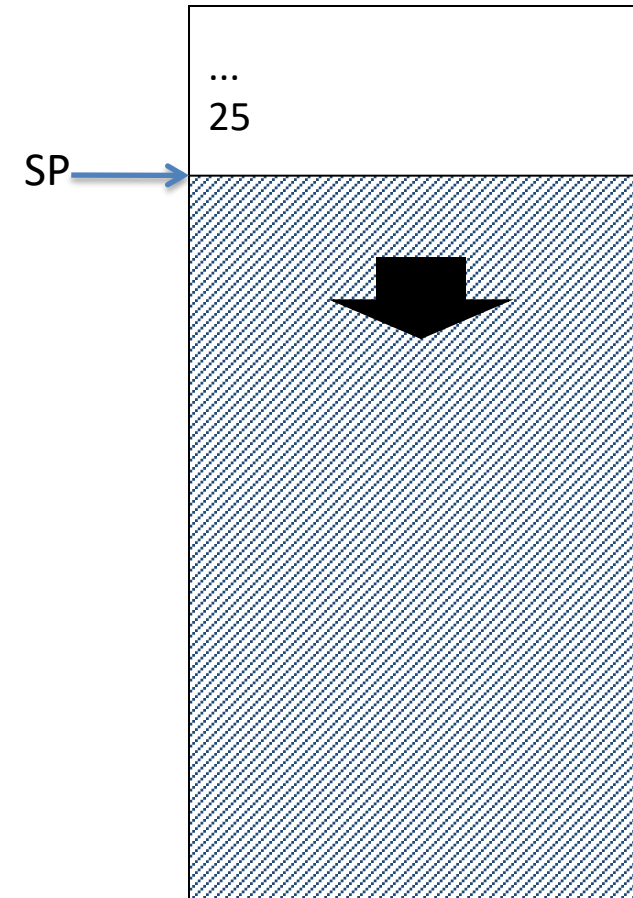
Arguments g, ..., j in \$a0, ..., \$a3  
f in \$s0 (hence, need to save \$s0 on stack)  
Result in \$v0

## Registers

\$s0: 25    \$a0: 5    \$a1: 2    \$a2: 3    \$a3: 1  
**\$t0: 7**    \$t1:    \$v0:

# Process Stack

```
leaf_ex:  addi $sp, $sp, -4
          sw   $s0, 0($sp)
          add  $t0, $a0, $a1
          PC → add  $t1, $a2, $a3
          sub  $s0, $t0, $t1
          add  $v0, $s0, $zero
          lw   $s0, 0($sp)
          addi $sp, $sp, 4
          jr   $ra
```



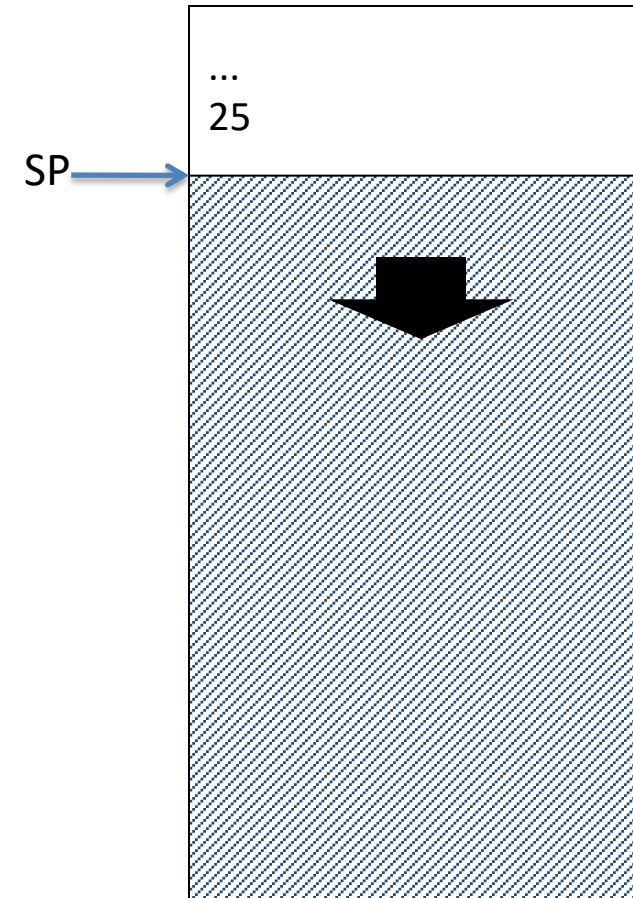
Arguments g, ..., j in \$a0, ..., \$a3  
f in \$s0 (hence, need to save \$s0 on stack)  
Result in \$v0

## Registers

\$s0: 25    \$a0: 5    \$a1: 2    \$a2: 3    \$a3: 1  
\$t0: 7    **\$t1: 4**    \$v0:

# Process Stack

```
leaf_ex:  addi  $sp, $sp, -4
          sw   $s0, 0($sp)
          add  $t0, $a0, $a1
          add  $t1, $a2, $a3
          PC → sub  $s0, $t0, $t1
          add  $v0, $s0, $zero
          lw   $s0, 0($sp)
          addi $sp, $sp, 4
          jr   $ra
```



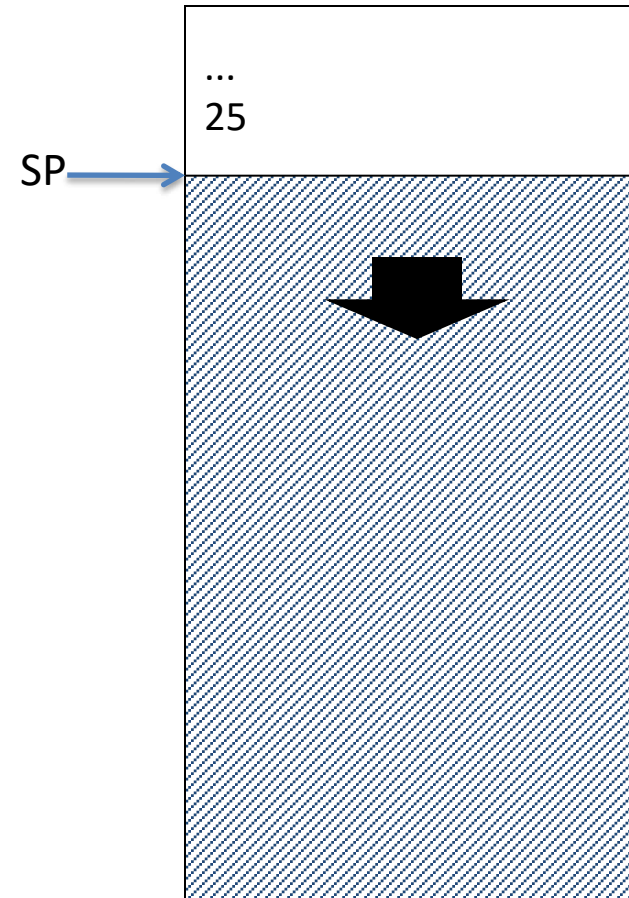
Arguments g, ..., j in \$a0, ..., \$a3  
f in \$s0 (hence, need to save \$s0 on stack)  
Result in \$v0

## Registers

**\$s0: 3**    \$a0: 5    \$a1: 2    \$a2: 3    \$a3: 1  
\$t0: 7    \$t1: 4    \$v0:

# Process Stack

```
leaf_ex:  addi $sp, $sp, -4
          sw   $s0, 0($sp)
          add  $t0, $a0, $a1
          add  $t1, $a2, $a3
          sub  $s0, $t0, $t1
PC →      add  $v0, $s0, $zero
          lw   $s0, 0($sp)
          addi $sp, $sp, 4
          jr   $ra
```



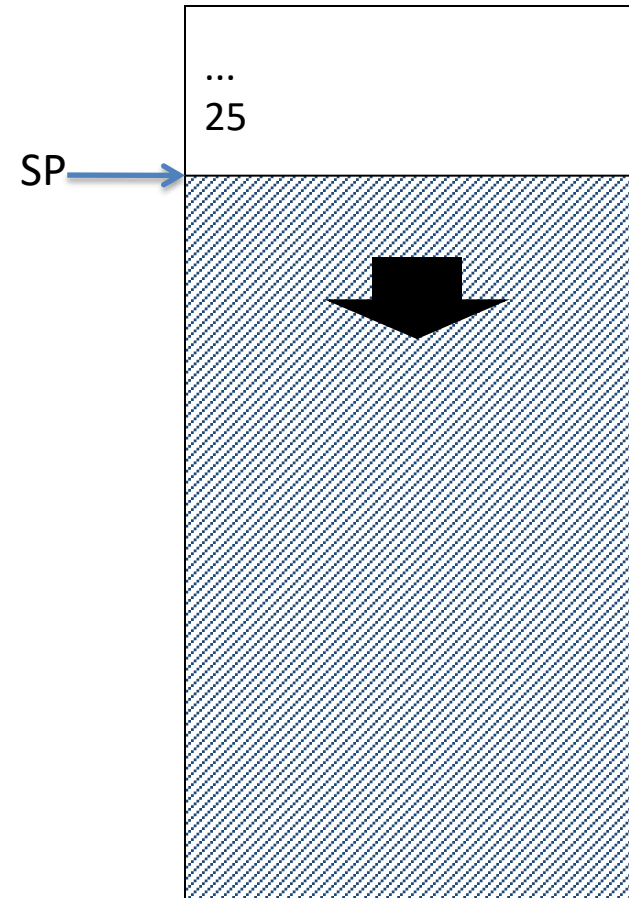
Arguments g, ..., j in \$a0, ..., \$a3  
f in \$s0 (hence, need to save \$s0 on stack)  
Result in \$v0

## Registers

\$s0: 3    \$a0: 5    \$a1: 2    \$a2: 3    \$a3: 1  
\$t0: 7    \$t1: 4    **\$v0: 3**

# Process Stack

```
leaf_ex:  addi $sp, $sp, -4
          sw   $s0, 0($sp)
          add  $t0, $a0, $a1
          add  $t1, $a2, $a3
          sub  $s0, $t0, $t1
          add  $v0, $s0, $zero
PC →      lw   $s0, 0($sp)
          addi $sp, $sp, 4
          jr   $ra
```



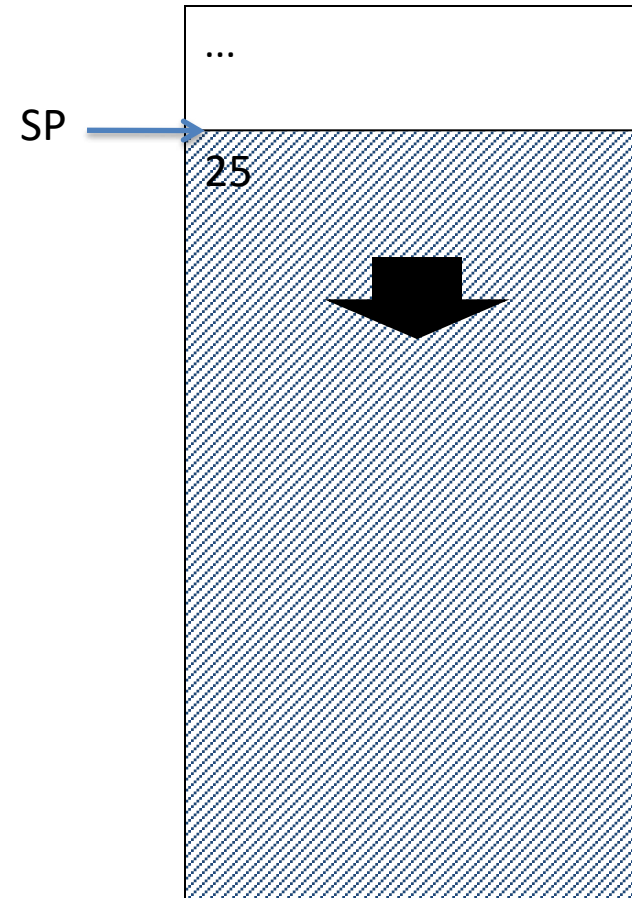
Arguments g, ..., j in \$a0, ..., \$a3  
f in \$s0 (hence, need to save \$s0 on stack)  
Result in \$v0

## Registers

**\$s0: 25**    \$a0: 5    \$a1: 2    \$a2: 3    \$a3: 1  
\$t0: 7    \$t1: 4    \$v0: 3

# Process Stack

```
leaf_ex:  addi $sp, $sp, -4
          sw   $s0, 0($sp)
          add  $t0, $a0, $a1
          add  $t1, $a2, $a3
          sub  $s0, $t0, $t1
          add  $v0, $s0, $zero
          lw   $s0, 0($sp)
PC →      addi $sp, $sp, 4
          jr   $ra
```



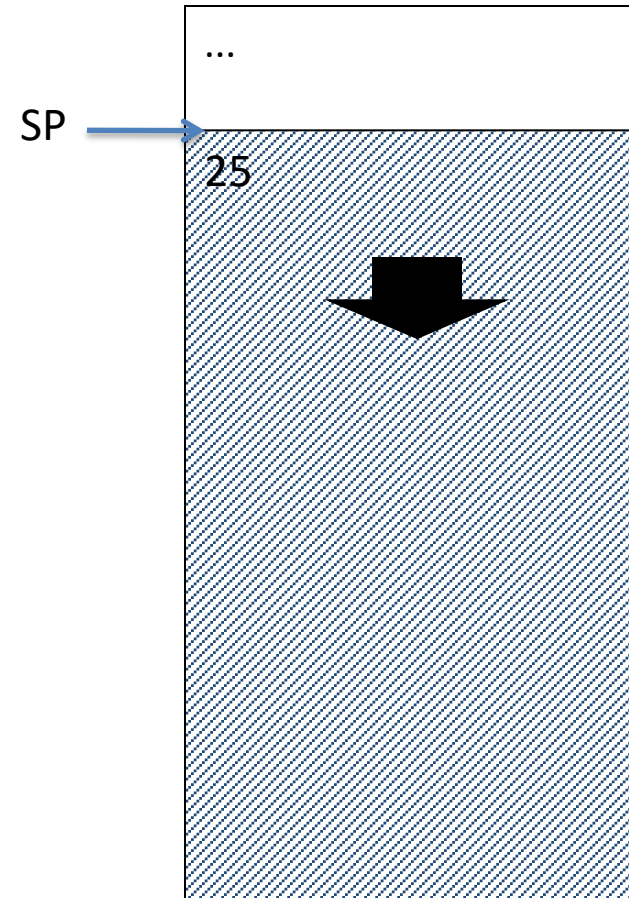
Arguments g, ..., j in \$a0, ..., \$a3  
f in \$s0 (hence, need to save \$s0 on stack)  
Result in \$v0

## Registers

\$s0: 25    \$a0: 5    \$a1: 2    \$a2: 3    \$a3: 1  
\$t0: 7    \$t1: 4    \$v0: 3

# Process Stack

```
leaf_ex:  addi $sp, $sp, -4
          sw   $s0, 0($sp)
          add  $t0, $a0, $a1
          add  $t1, $a2, $a3
          sub  $s0, $t0, $t1
          add  $v0, $s0, $zero
          lw   $s0, 0($sp)
          addi $sp, $sp, 4
PC →      jr   $ra
```



Arguments g, ..., j in \$a0, ..., \$a3  
f in \$s0 (hence, need to save \$s0 on stack)  
Result in \$v0

## Registers

\$s0: 25    \$a0: 5    \$a1: 2    \$a2: 3    \$a3: 1  
\$t0: 7    \$t1: 4    \$v0: 3

# Non-Leaf Procedures

- Procedures that call other procedures
- For nested call, caller needs to save on the stack:
  - Its return address
  - Any arguments and temporaries needed after the call
- Restore from the stack after the call



# Non-Leaf Procedure Example

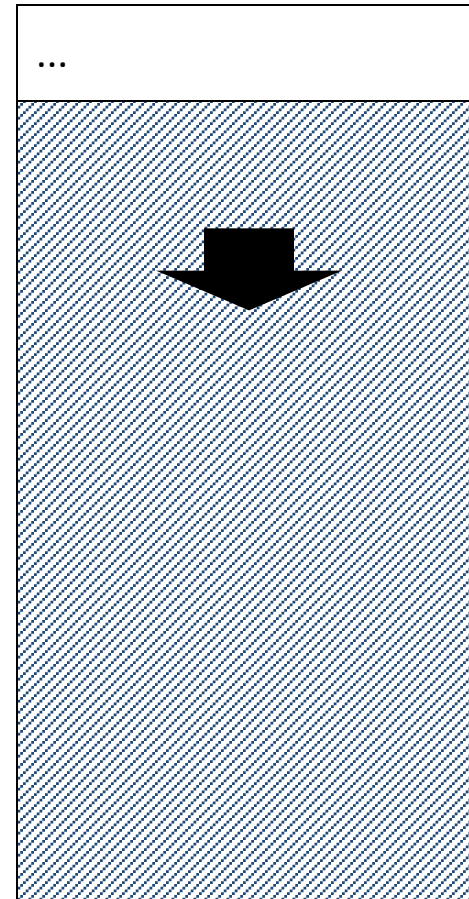
- C code:

```
int fact (int n) {  
    if (n < 2)  
        return 1;  
    else  
        return n * fact(n - 1);  
}
```

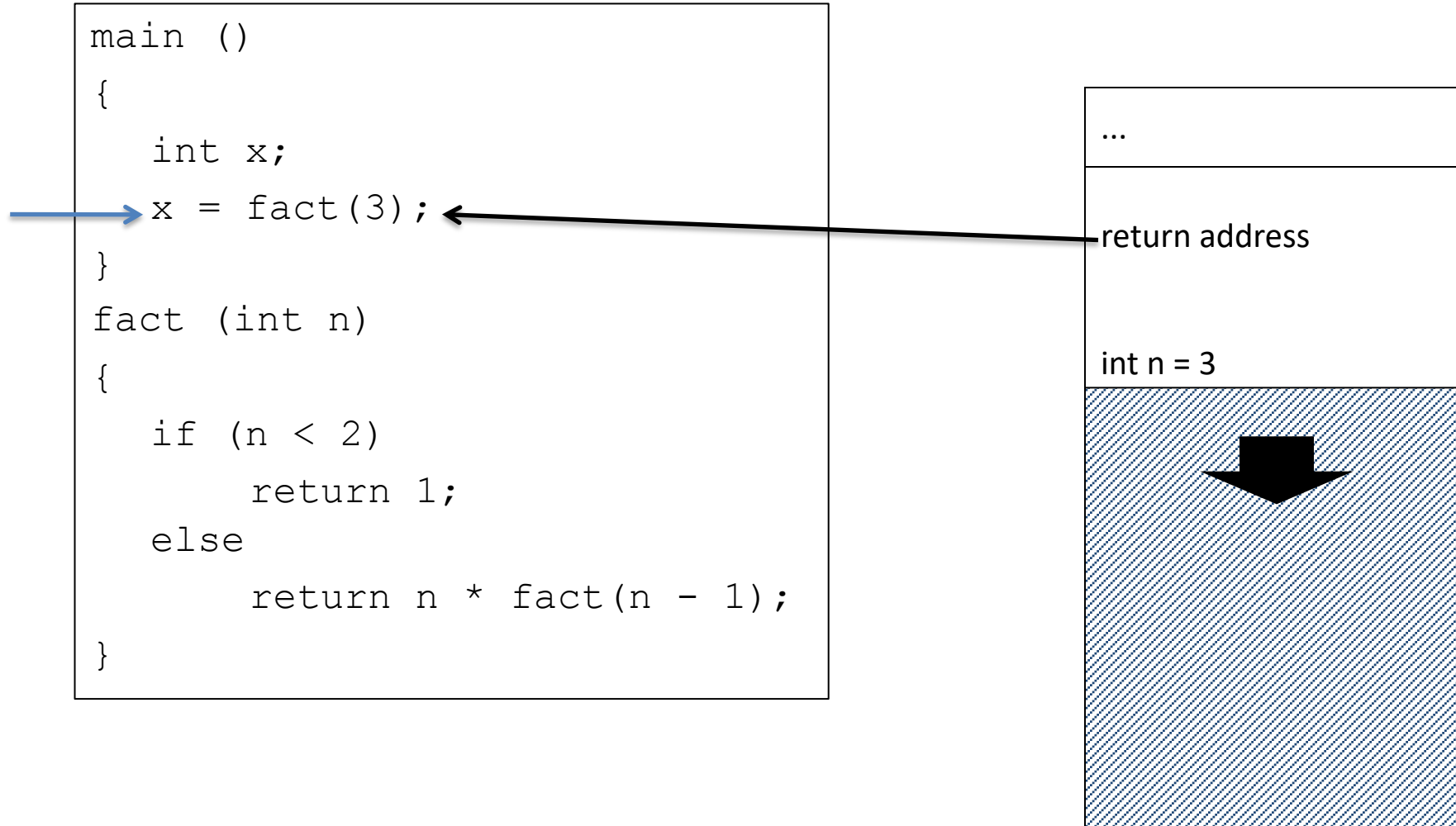
- Argument n in \$a0
- Result in \$v0

# Process Stack

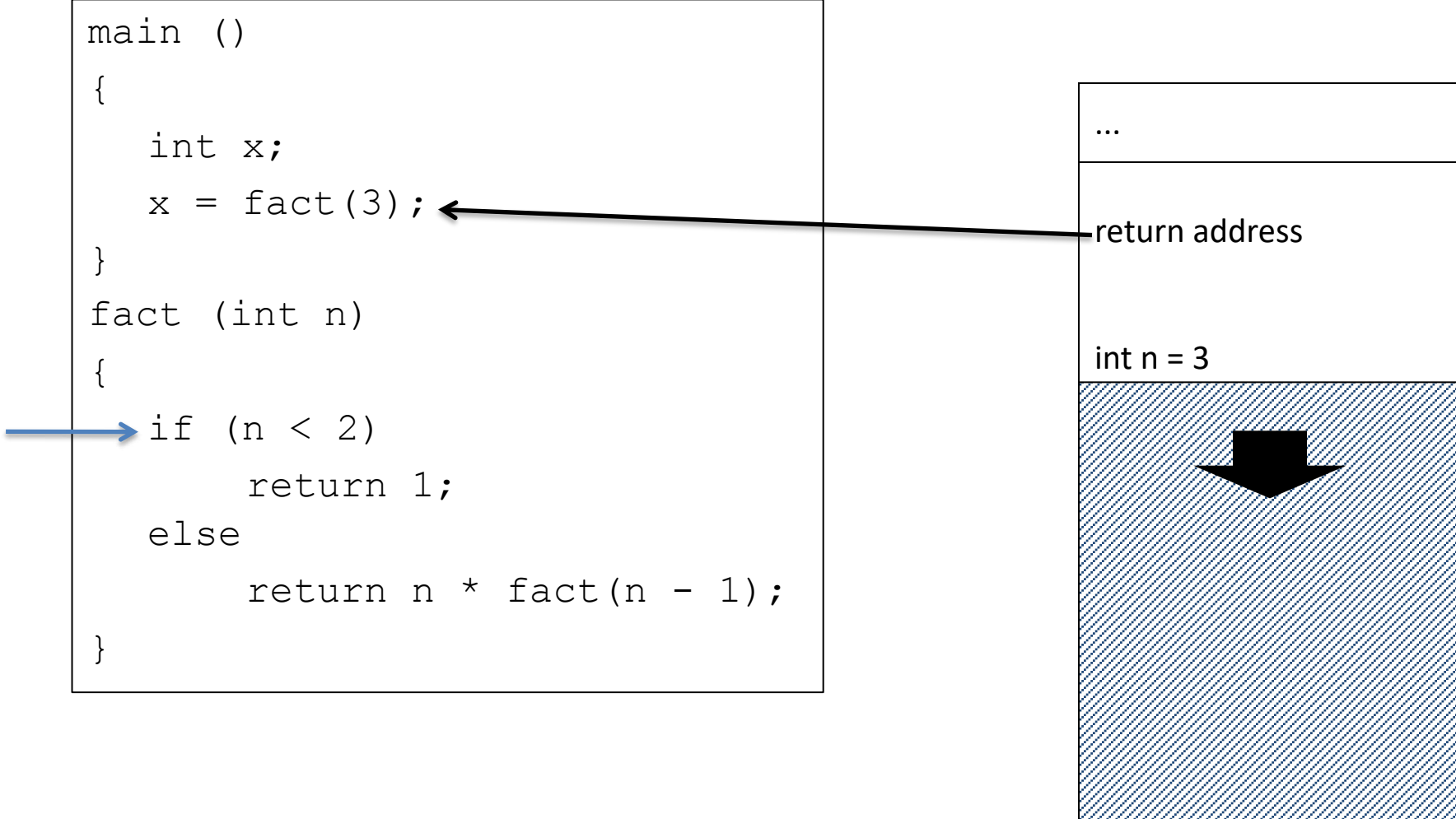
```
main ()
{
  → int x;
    x = fact(3);
}
fact (int n)
{
  if (n < 2)
    return 1;
  else
    return n * fact(n - 1);
}
```



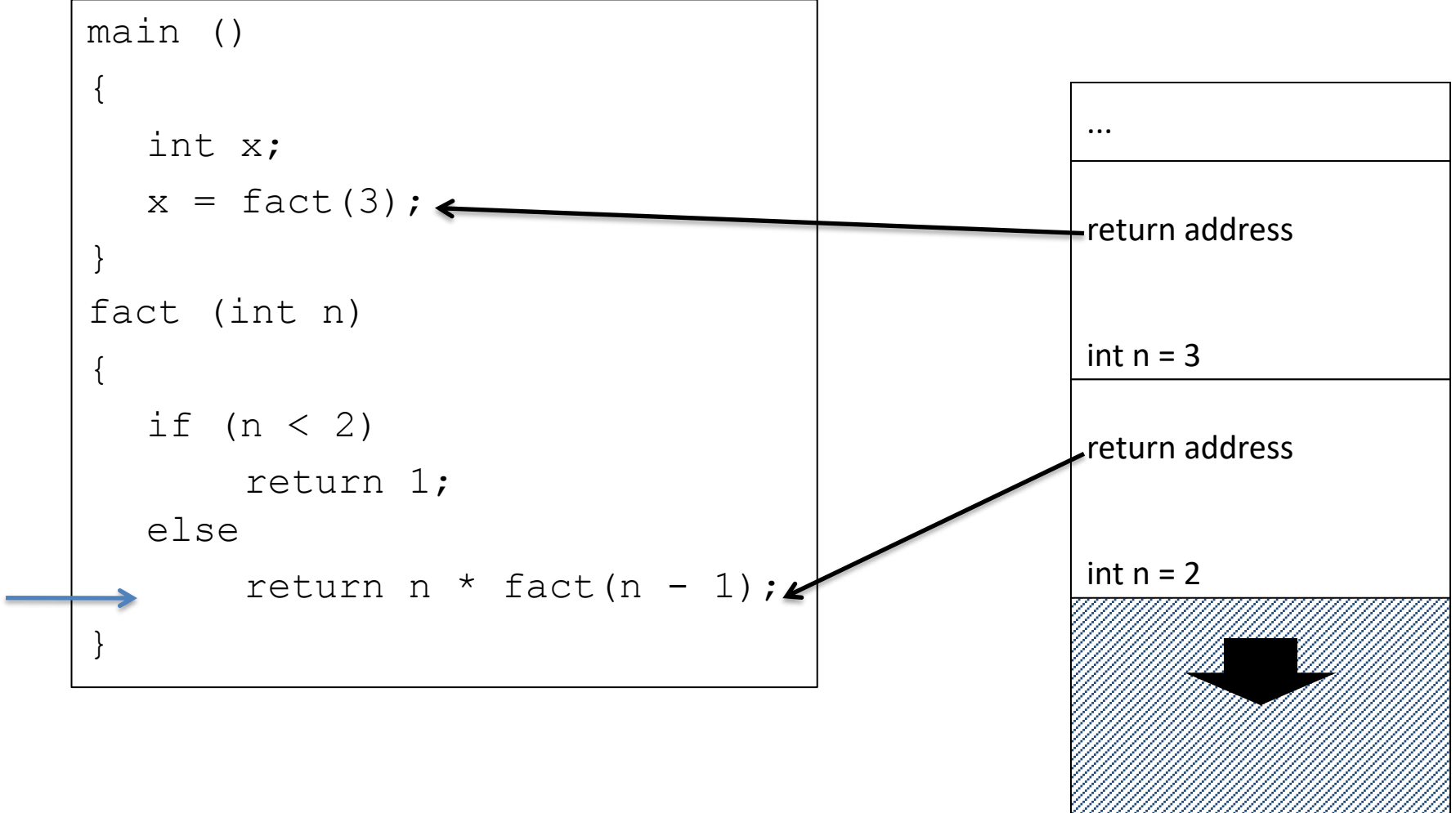
# Process Stack



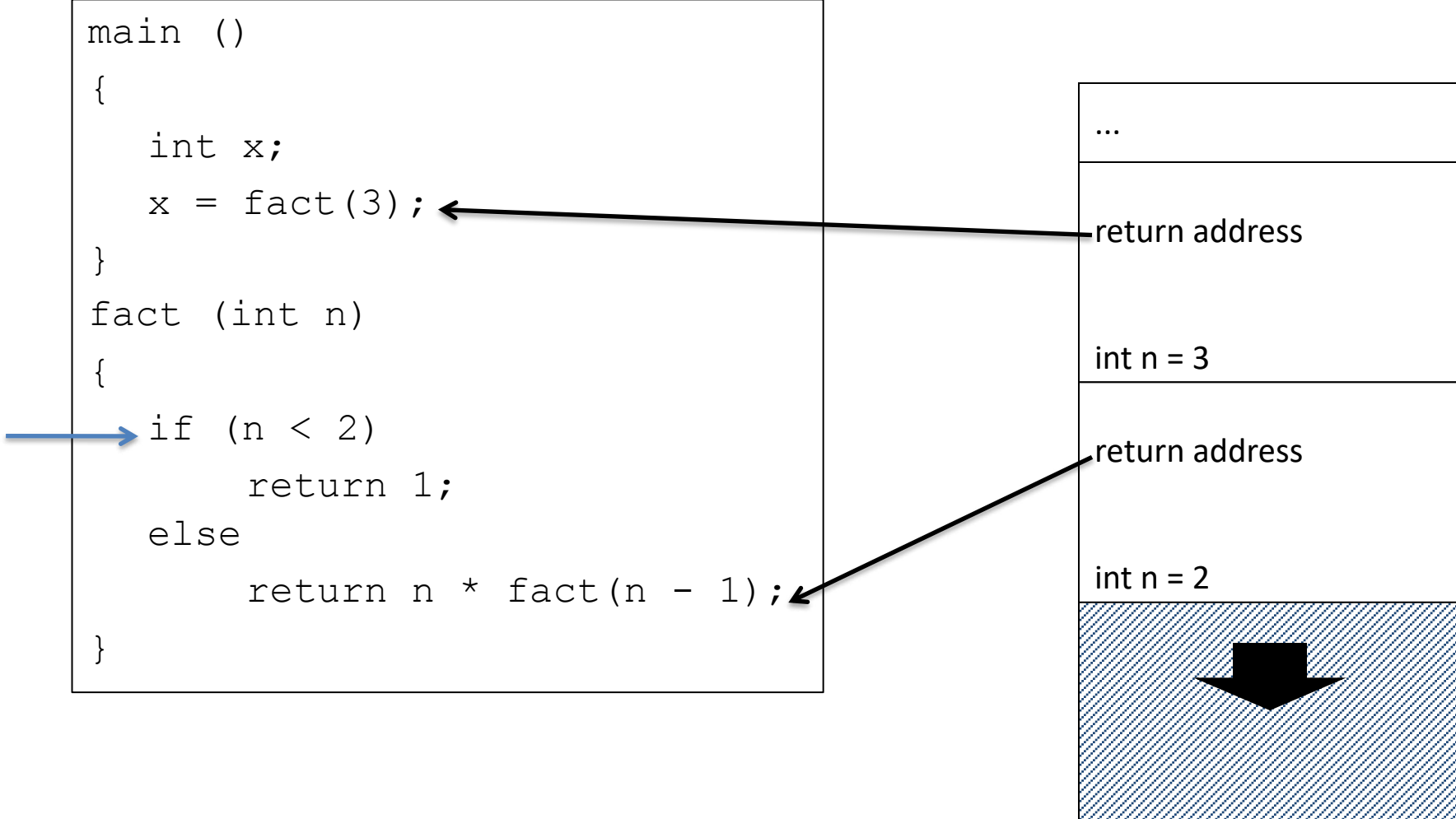
# Process Stack



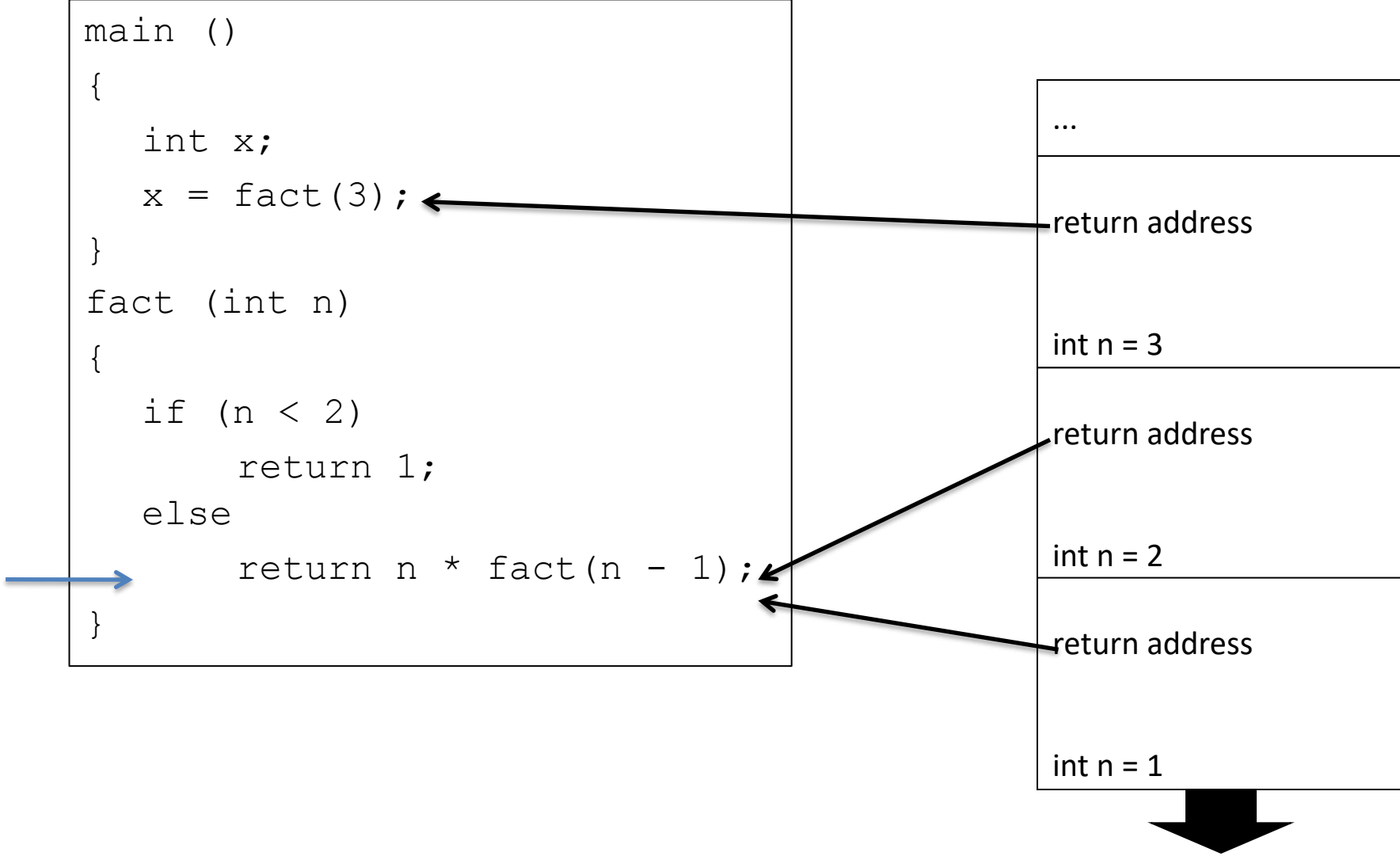
# Process Stack



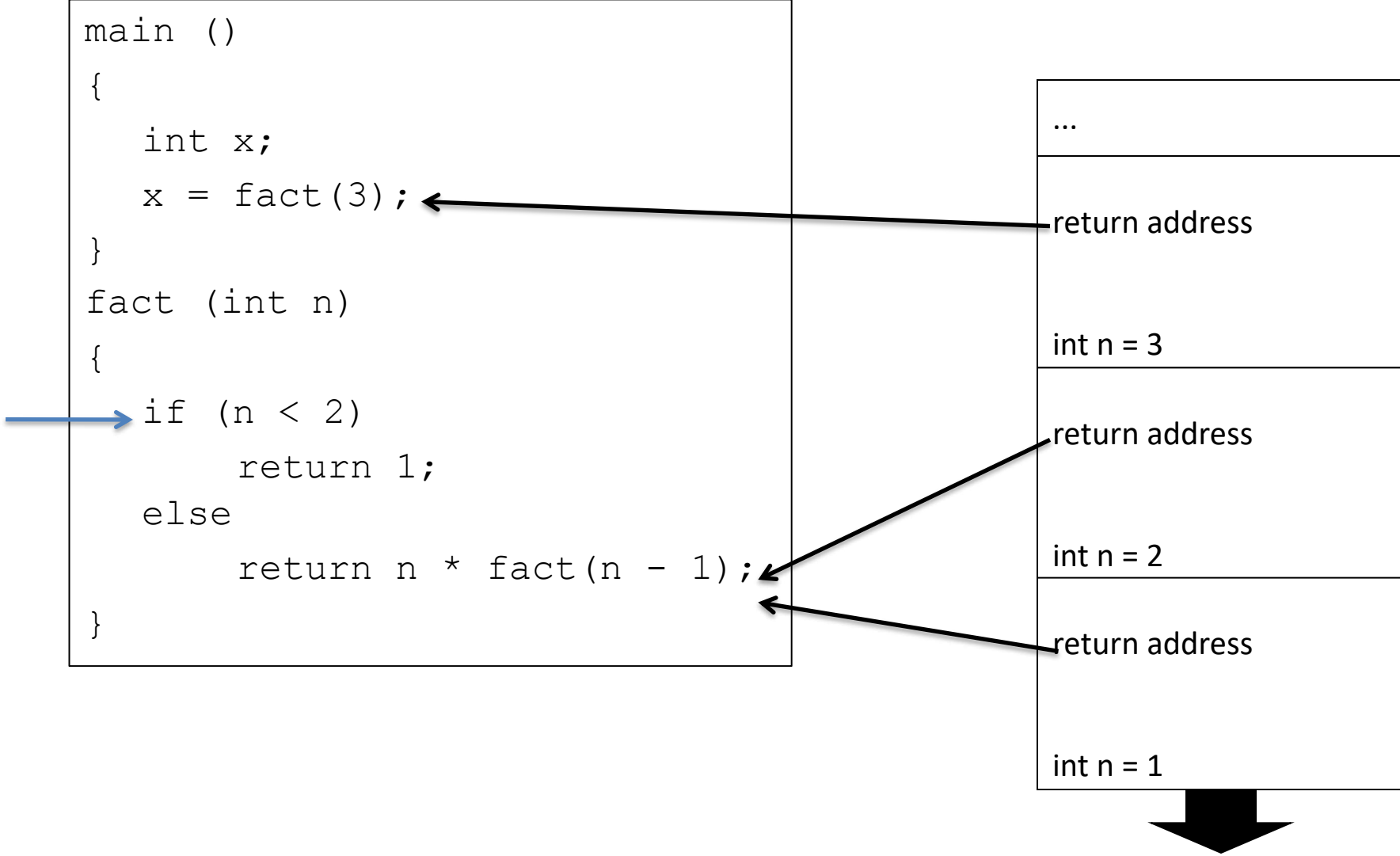
# Process Stack



# Process Stack

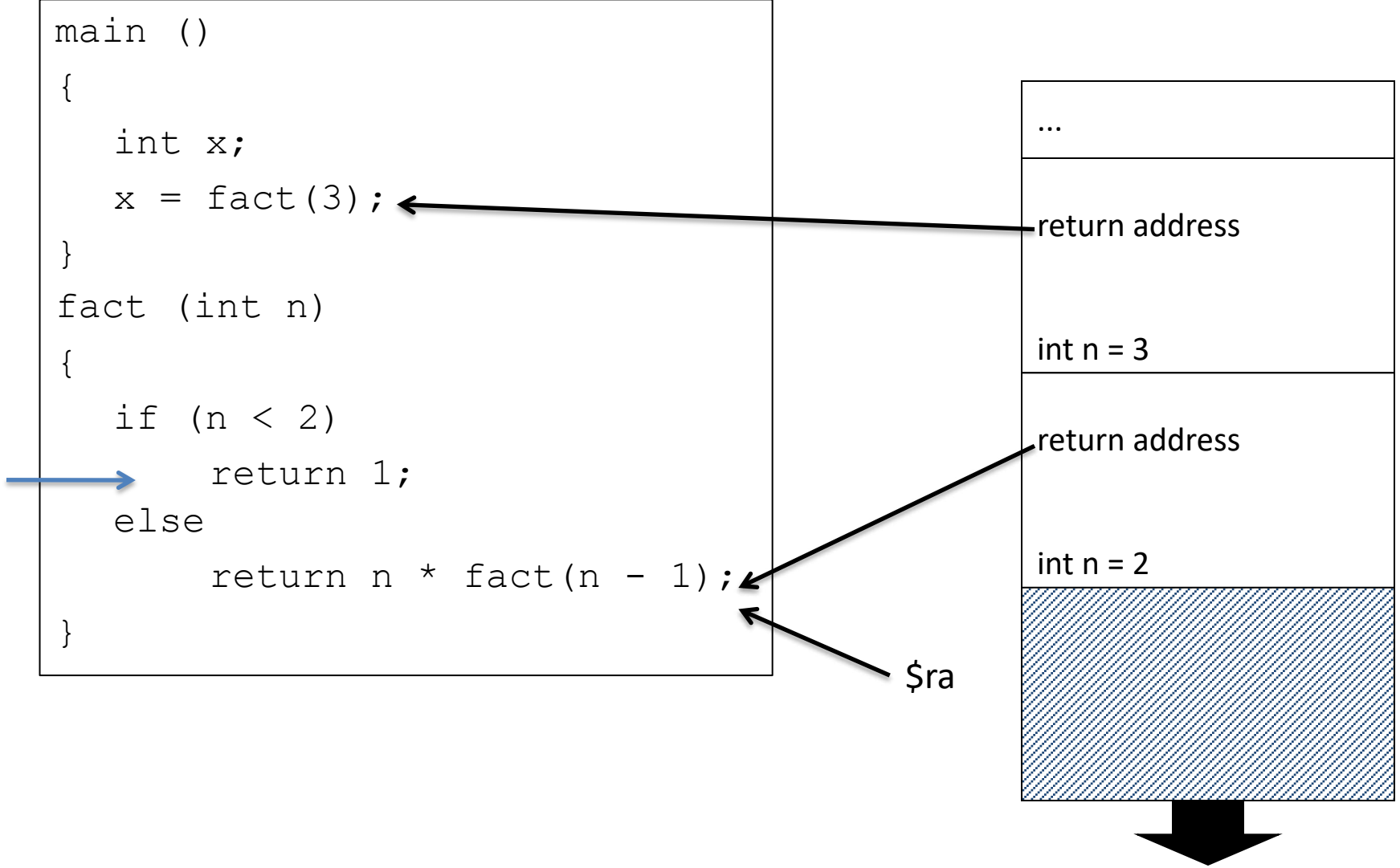


# Process Stack

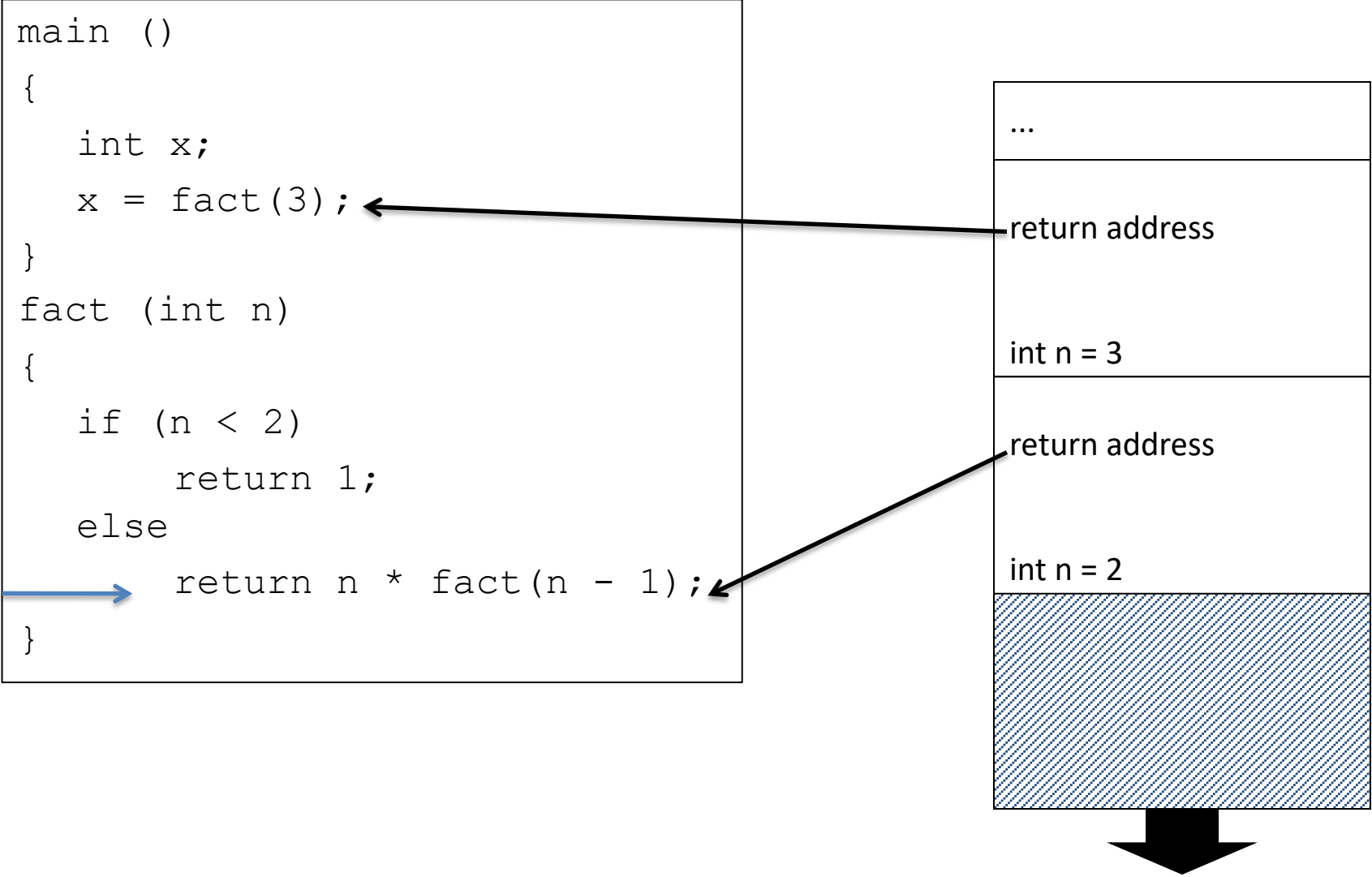




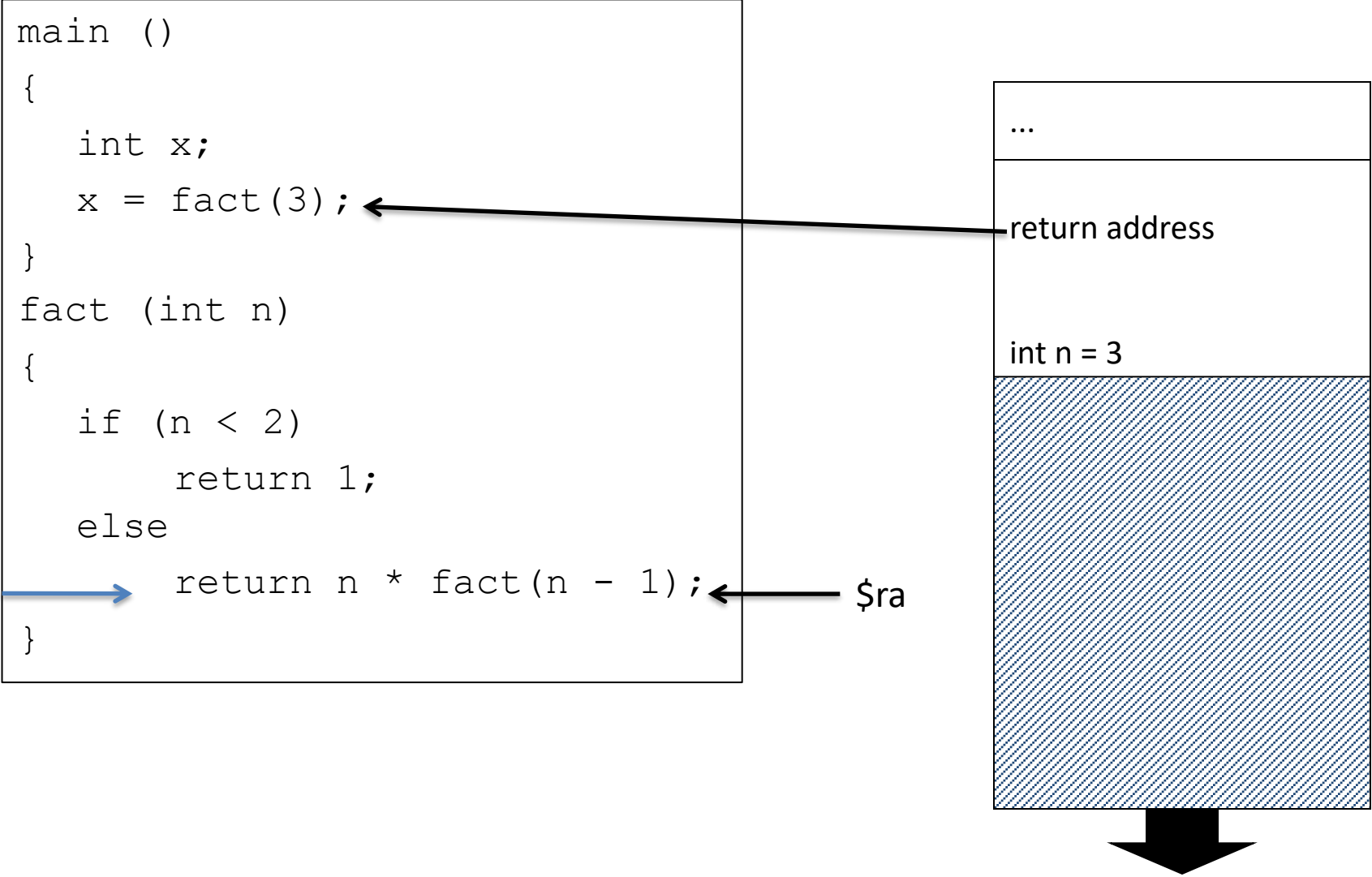
# Process Stack



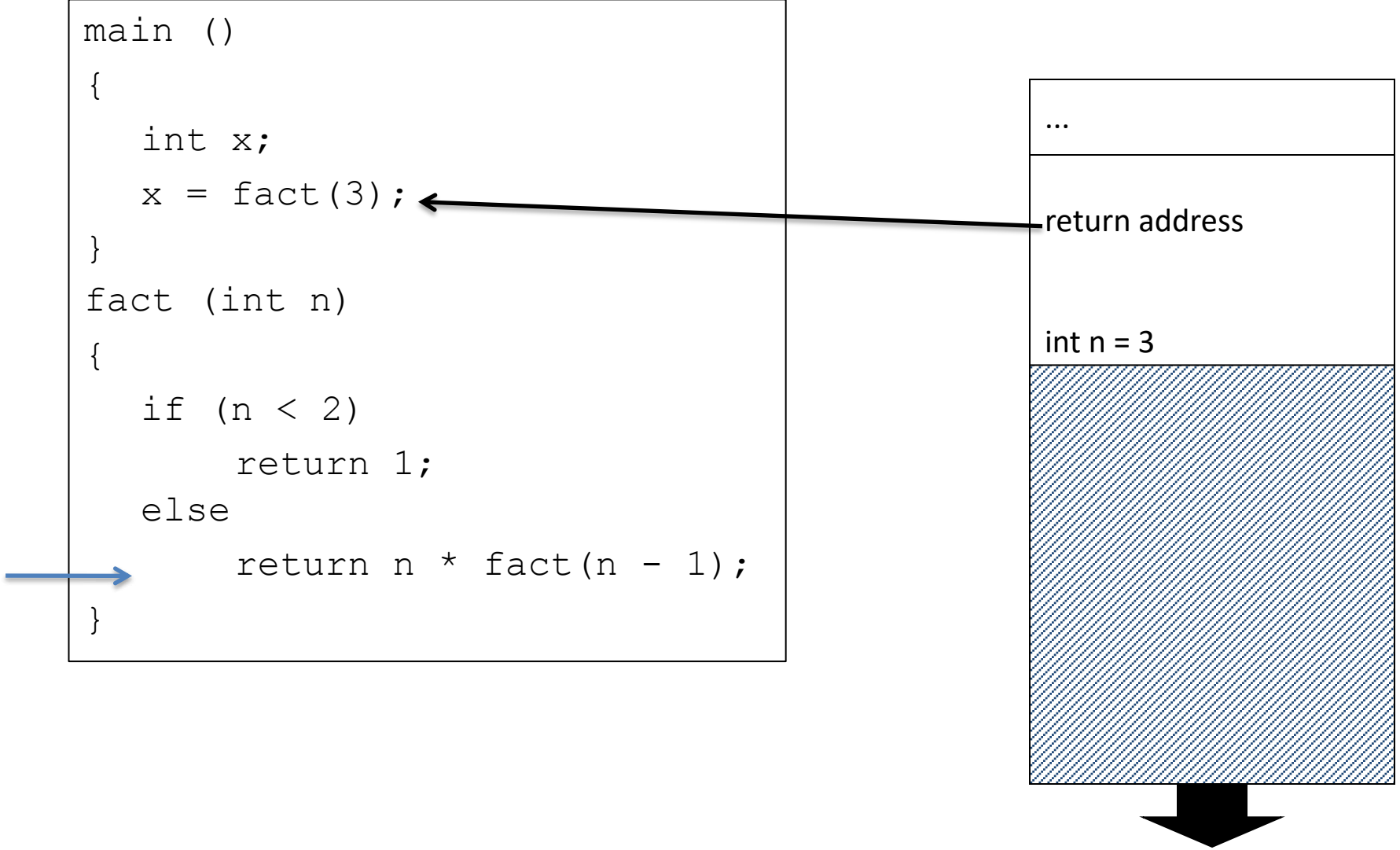
# Process Stack



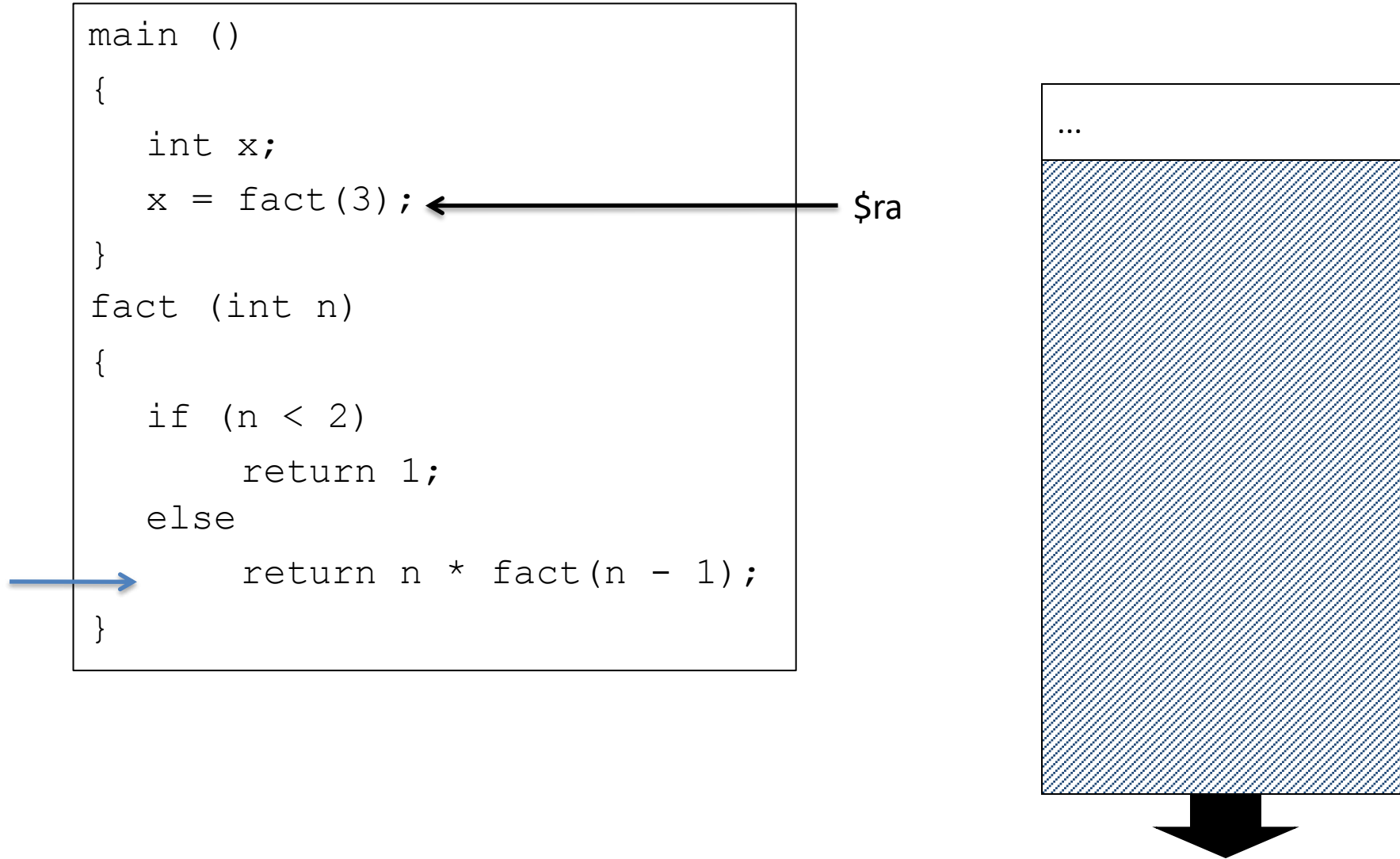
# Process Stack



# Process Stack

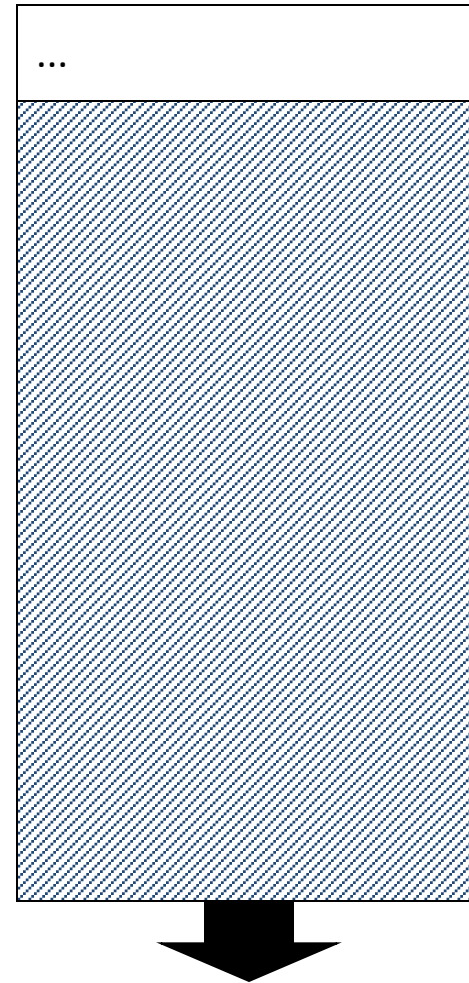


# Process Stack



# Process Stack

```
main ()
{
    int x;
    → x = fact(3);
}
fact (int n)
{
    if (n < 2)
        return 1;
    else
        return n * fact(n - 1);
}
```



# Non-Leaf Procedure Example

- MIPS code:

```
fact:
    addi $sp, $sp, -8      # adjust stack for 2 items
    sw   $ra, 4($sp)      # save return address
    sw   $a0, 0($sp)      # save argument
    slti $t0, $a0, 2      # test for n < 2
    beq  $t0, $zero, L1   # if so, result is 1
    addi $v0, $zero, 1    #   pop 2 items from stack
    addi $sp, $sp, 8      #   and return
    jr   $ra
L1:   addi $a0, $a0, -1    # else decrement n
    jal  fact             # recursive call
    lw   $a0, 0($sp)      # restore original n
    lw   $ra, 4($sp)      #   and return address
    addi $sp, $sp, 8      # pop 2 items from stack
    mul  $v0, $a0, $v0    # multiply to get result
    jr   $ra             # and return
```

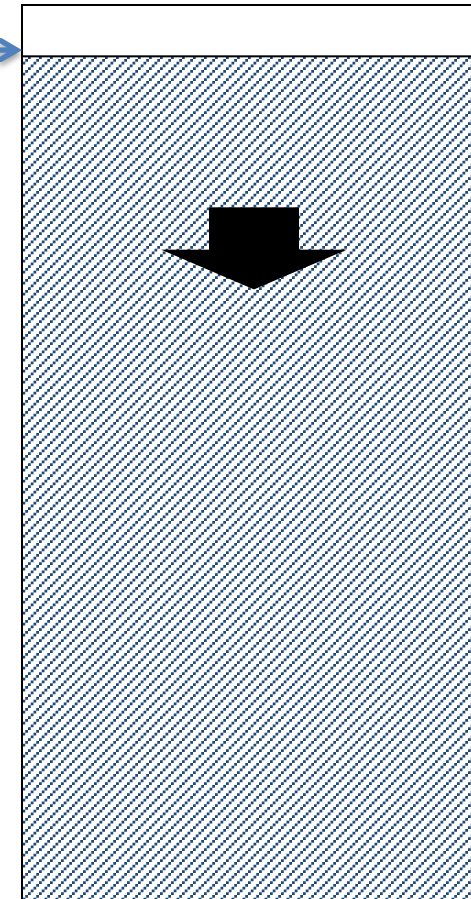
\$ra = 0x865  
\$a0 = 3  
\$v0 =  
\$t0 =

# fact(3)

PC →

```
fact:
    addi $sp, $sp, -8      # adjust stack for 2 items
    sw   $ra, 4($sp)      # save return address
    sw   $a0, 0($sp)      # save argument
    slti $t0, $a0, 2      # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1     # if so, result is 1
    addi $sp, $sp, 8      # pop 2 items from stack
    jr   $ra              # and return
L1:   addi $a0, $a0, -1    # else decrement n
    jal  fact             # recursive call
    lw   $a0, 0($sp)      # restore original n
    lw   $ra, 4($sp)      # and return address
    addi $sp, $sp, 8      # pop 2 items from stack
    mul  $v0, $a0, $v0    # multiply to get result
    jr   $ra             # and return
```

SP →

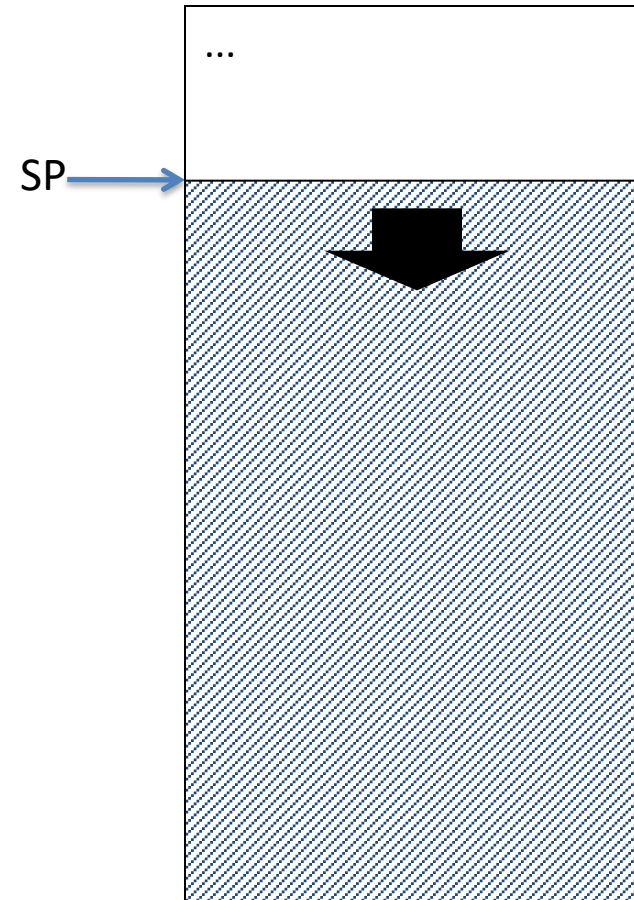




\$ra = 0x865  
\$a0 = 3  
\$v0 =  
\$t0 =

# fact

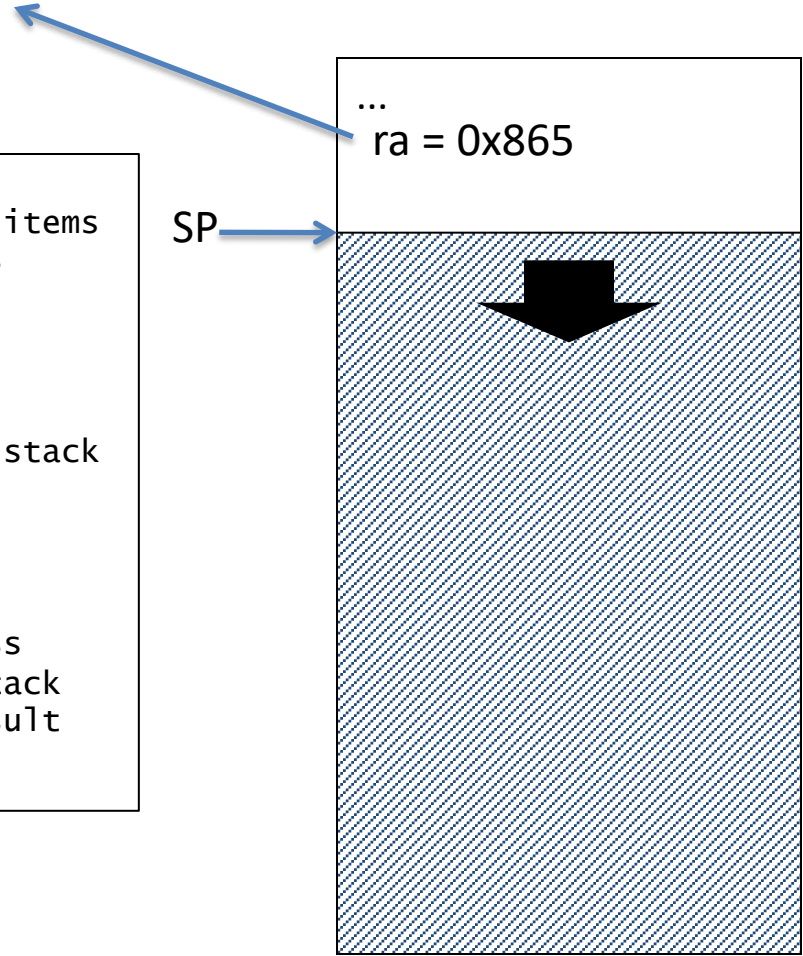
```
fact:
PC → addi $sp, $sp, -8      # adjust stack for 2 items
      sw  $ra, 4($sp)      # save return address
      sw  $a0, 0($sp)      # save argument
      slti $t0, $a0, 2     # test for n < 2
      beq $t0, $zero, L1
      addi $v0, $zero, 1   # if so, result is 1
      addi $sp, $sp, 8     # pop 2 items from stack
      jr  $ra              # and return
L1:   addi $a0, $a0, -1     # else decrement n
      jal fact             # recursive call
      lw  $a0, 0($sp)      # restore original n
      lw  $ra, 4($sp)      # and return address
      addi $sp, $sp, 8     # pop 2 items from stack
      mul $v0, $a0, $v0    # multiply to get result
      jr  $ra              # and return
```



\$ra = 0x865  
\$a0 = 3  
\$v0 =  
\$t0 =

# fact

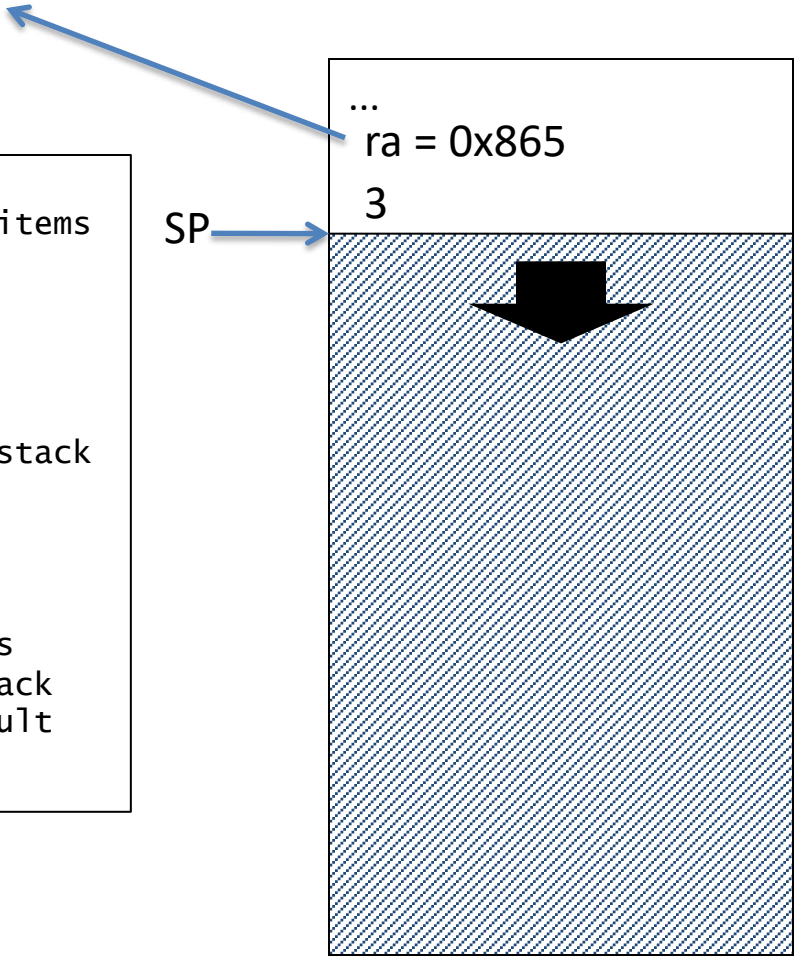
```
fact:
PC → addi $sp, $sp, -8      # adjust stack for 2 items
      sw   $ra, 4($sp)     # save return address
      sw   $a0, 0($sp)     # save argument
      slti $t0, $a0, 2     # test for n < 2
      beq $t0, $zero, L1
      addi $v0, $zero, 1   # if so, result is 1
      addi $sp, $sp, 8     # pop 2 items from stack
      jr   $ra             # and return
L1:   addi $a0, $a0, -1     # else decrement n
      jal  fact            # recursive call
      lw   $a0, 0($sp)     # restore original n
      lw   $ra, 4($sp)     # and return address
      addi $sp, $sp, 8     # pop 2 items from stack
      mul  $v0, $a0, $v0   # multiply to get result
      jr   $ra             # and return
```



\$ra = 0x865  
\$a0 = 3  
\$v0 =  
\$t0 =

# fact

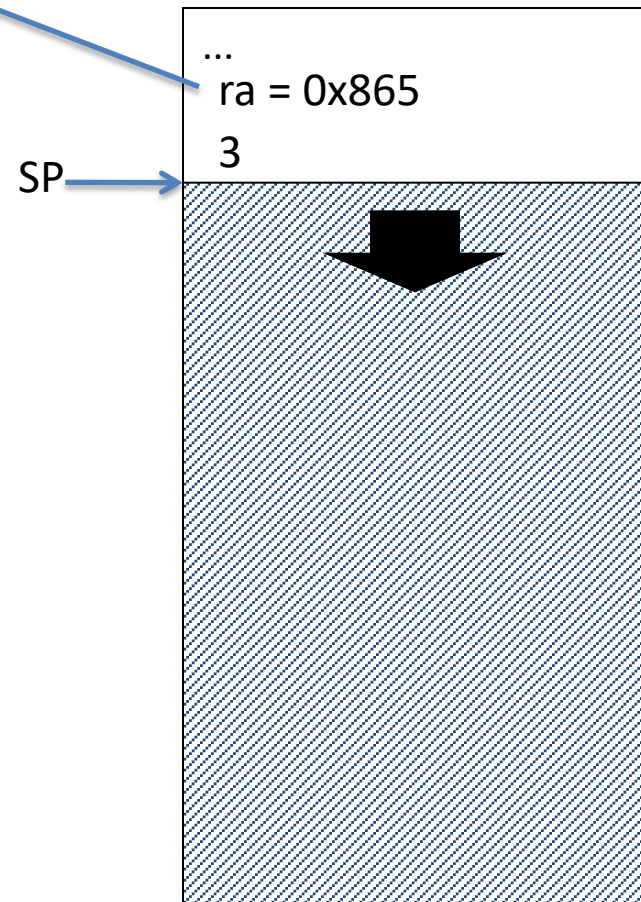
```
fact:
    addi $sp, $sp, -8      # adjust stack for 2 items
    sw   $ra, 4($sp)      # save return address
PC → sw   $a0, 0($sp)     # save argument
    slti $t0, $a0, 2      # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1    # if so, result is 1
    addi $sp, $sp, 8      # pop 2 items from stack
    jr   $ra              # and return
L1:  addi $a0, $a0, -1     # else decrement n
    jal  fact             # recursive call
    lw   $a0, 0($sp)      # restore original n
    lw   $ra, 4($sp)      # and return address
    addi $sp, $sp, 8      # pop 2 items from stack
    mul  $v0, $a0, $v0    # multiply to get result
    jr   $ra              # and return
```



\$ra = 0x865  
\$a0 = 3  
\$v0 =  
\$t0 = 0

# fact

```
fact:
    addi $sp, $sp, -8      # adjust stack for 2 items
    sw   $ra, 4($sp)      # save return address
    sw   $a0, 0($sp)      # save argument
PC → slti $t0, $a0, 2     # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1    # if so, result is 1
    addi $sp, $sp, 8      # pop 2 items from stack
    jr   $ra              # and return
L1:  addi $a0, $a0, -1    # else decrement n
    jal  fact             # recursive call
    lw   $a0, 0($sp)      # restore original n
    lw   $ra, 4($sp)      # and return address
    addi $sp, $sp, 8      # pop 2 items from stack
    mul  $v0, $a0, $v0    # multiply to get result
    jr   $ra              # and return
```



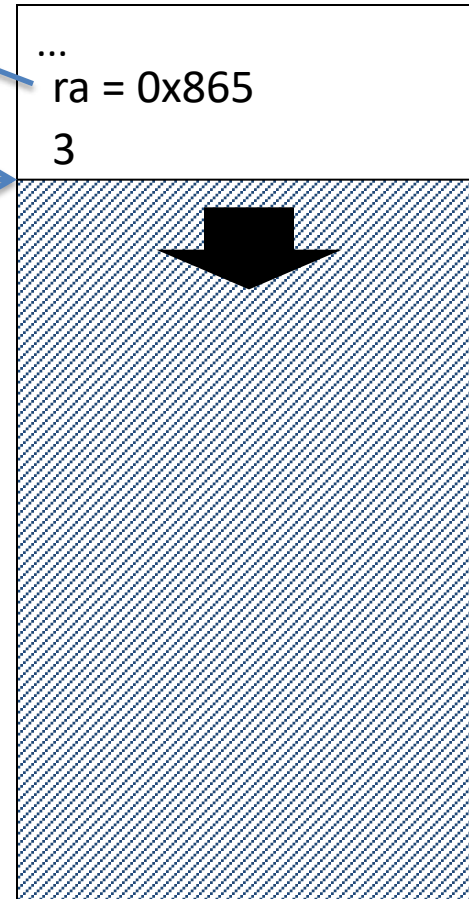
\$ra = 0x865  
\$a0 = 3  
\$v0 =  
\$t0 = 0

# fact

```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return
```

PC →

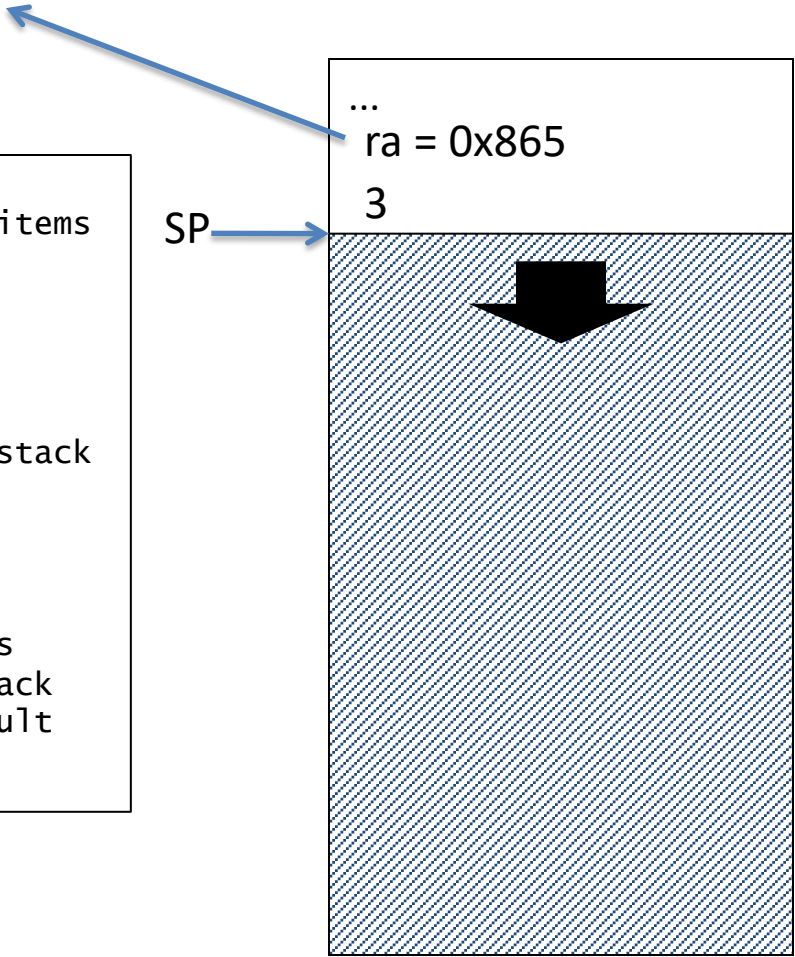
SP →



\$ra = 0x865  
\$a0 = 2  
\$v0 =  
\$t0 = 0

# fact

```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
PC → L1 → addi $a0, $a0, -1 # else decrement n
        jal  fact        # recursive call
        lw   $a0, 0($sp) # restore original n
        lw   $ra, 4($sp) # and return address
        addi $sp, $sp, 8 # pop 2 items from stack
        mul  $v0, $a0, $v0 # multiply to get result
        jr   $ra        # and return
```



After this line of code, the next line of code we run will be

```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
PC → L1: addi $a0, $a0, -1 # else decrement n
        jal  fact        # recursive call
        lw   $a0, 0($sp) # restore original n
        lw   $ra, 4($sp) # and return address
        addi $sp, $sp, 8 # pop 2 items from stack
        mul  $v0, $a0, $v0 # multiply to get result
        jr   $ra        # and return
```

```
$ra = 0x865
$a0 = 2
$v0 =
$t0 = 0
```

A. lw \$a0, 0(\$sp)

B. addi \$a0, \$a0, -1

C. addi \$sp, \$sp, -8

D. jr \$ra

E. None of the above

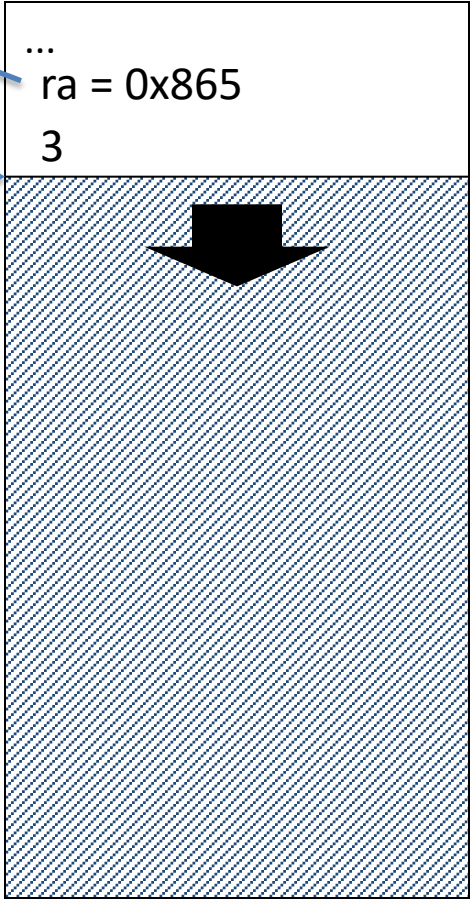
$\$ra = L1 + 2$   
 $\$a0 = 2$   
 $\$v0 =$   
 $\$t0 = 0$

# fact

```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra           # and return
```

PC →

SP →

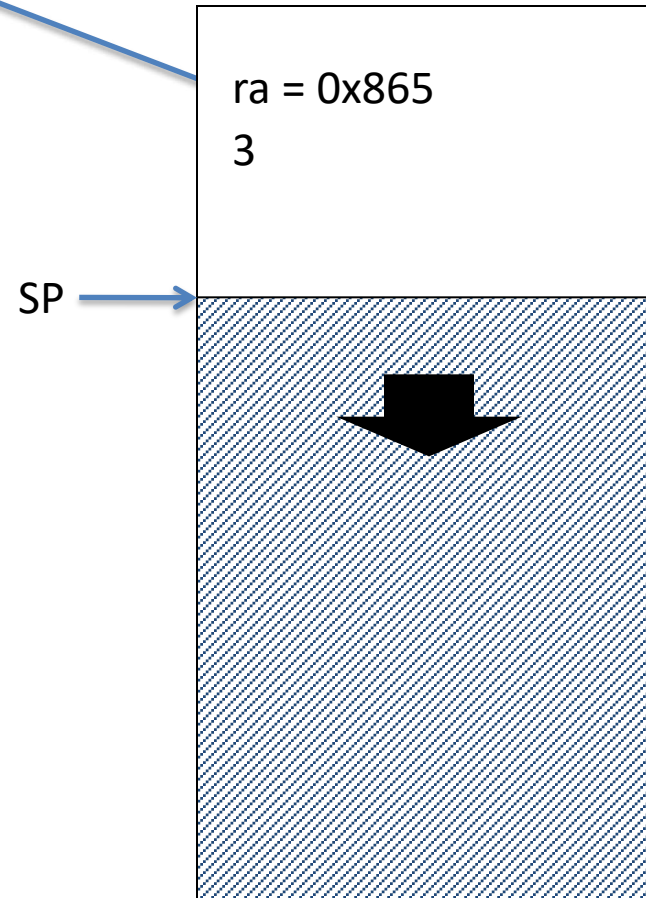




\$ra = L1 + 2  
\$a0 = 2  
\$v0 =  
\$t0 =

# fact

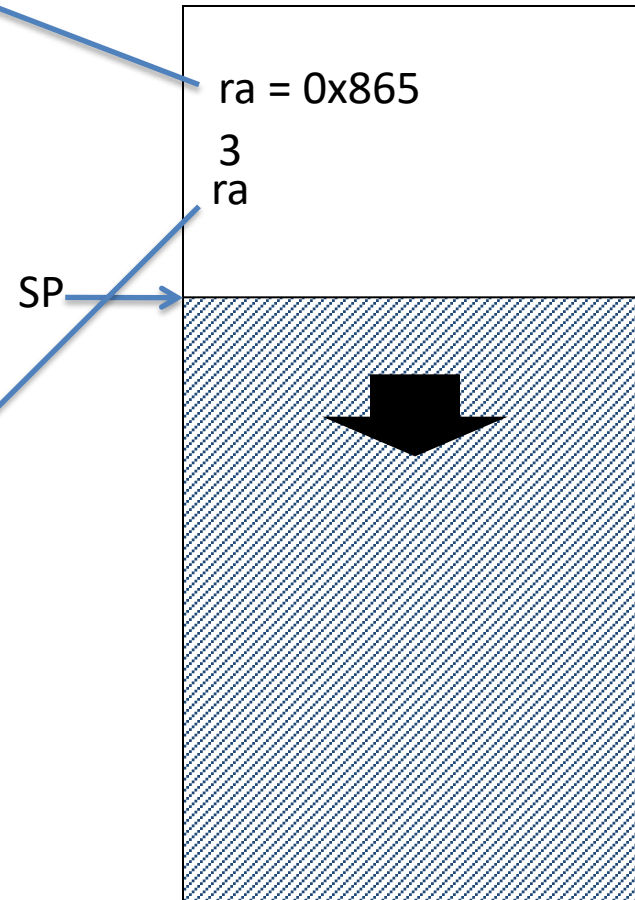
```
fact:
PC → addi $sp, $sp, -8      # adjust stack for 2 items
      sw  $ra, 4($sp)      # save return address
      sw  $a0, 0($sp)      # save argument
      slti $t0, $a0, 2     # test for n < 2
      beq $t0, $zero, L1
      addi $v0, $zero, 1   # if so, result is 1
      addi $sp, $sp, 8     # pop 2 items from stack
      jr  $ra              # and return
L1:   addi $a0, $a0, -1     # else decrement n
      jal fact             # recursive call
      lw  $a0, 0($sp)      # restore original n
      lw  $ra, 4($sp)      # and return address
      addi $sp, $sp, 8     # pop 2 items from stack
      mul $v0, $a0, $v0    # multiply to get result
      jr  $ra              # and return
```



\$ra = L1 + 2  
\$a0 = 2  
\$v0 =  
\$t0 = 0

# fact

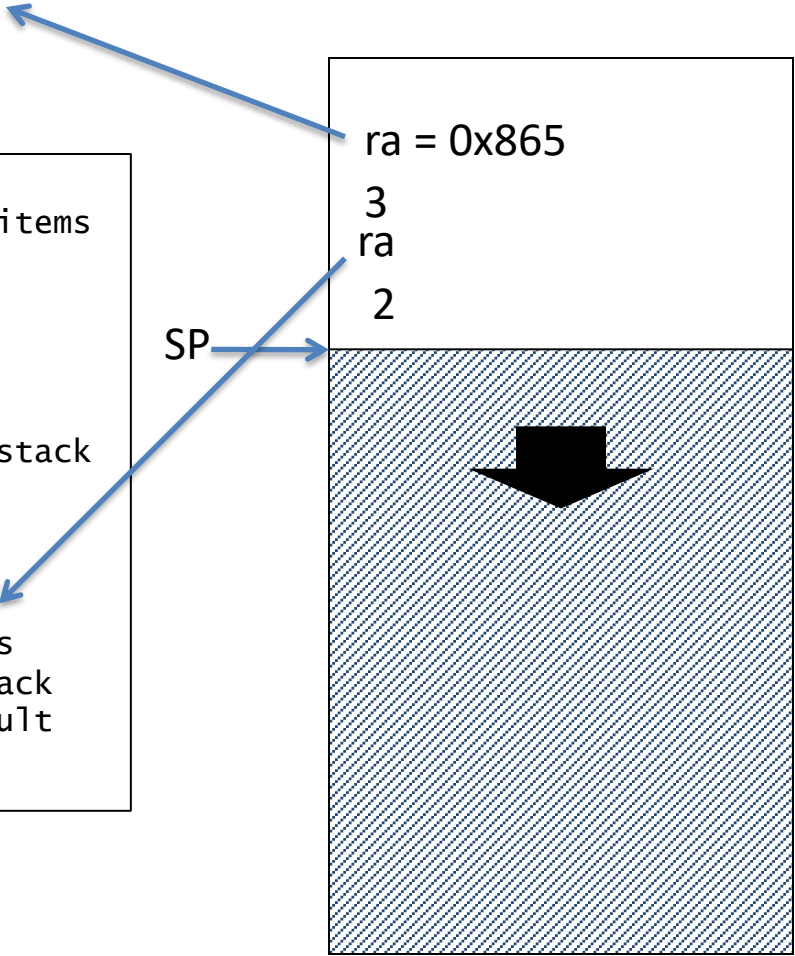
```
fact:
PC → addi $sp, $sp, -8      # adjust stack for 2 items
      sw  $ra, 4($sp)      # save return address
      sw  $a0, 0($sp)      # save argument
      slti $t0, $a0, 2     # test for n < 2
      beq $t0, $zero, L1
      addi $v0, $zero, 1   # if so, result is 1
      addi $sp, $sp, 8     # pop 2 items from stack
      jr  $ra              # and return
L1:   addi $a0, $a0, -1     # else decrement n
      jal fact             # recursive call
      lw  $a0, 0($sp)      # restore original n
      lw  $ra, 4($sp)      # and return address
      addi $sp, $sp, 8     # pop 2 items from stack
      mul $v0, $a0, $v0    # multiply to get result
      jr  $ra              # and return
```



\$ra = L1 + 2  
\$a0 = 2  
\$v0 =  
\$t0 = 0

# fact

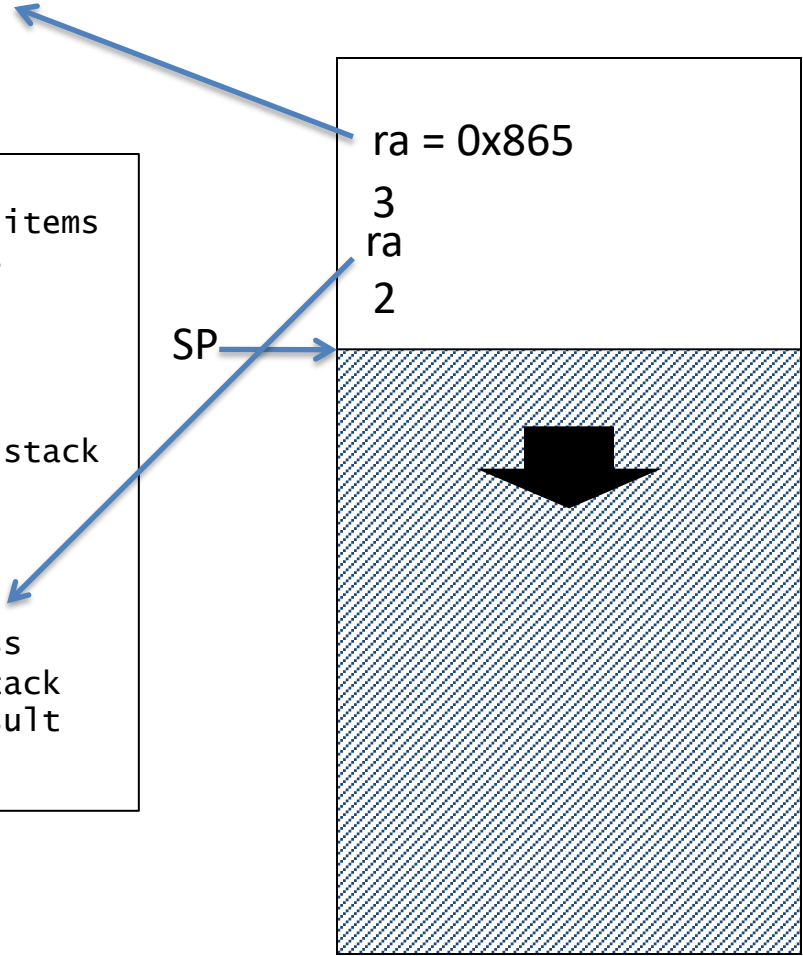
```
fact:
    addi $sp, $sp, -8      # adjust stack for 2 items
    sw   $ra, 4($sp)      # save return address
PC → sw   $a0, 0($sp)      # save argument
    slti $t0, $a0, 2      # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1     # if so, result is 1
    addi $sp, $sp, 8      # pop 2 items from stack
    jr   $ra              # and return
L1:  addi $a0, $a0, -1     # else decrement n
    jal  fact            # recursive call
    lw   $a0, 0($sp)      # restore original n
    lw   $ra, 4($sp)      # and return address
    addi $sp, $sp, 8      # pop 2 items from stack
    mul  $v0, $a0, $v0    # multiply to get result
    jr   $ra              # and return
```



\$ra = L1 + 2  
\$a0 = 2  
\$v0 =  
\$t0 = 0

# fact

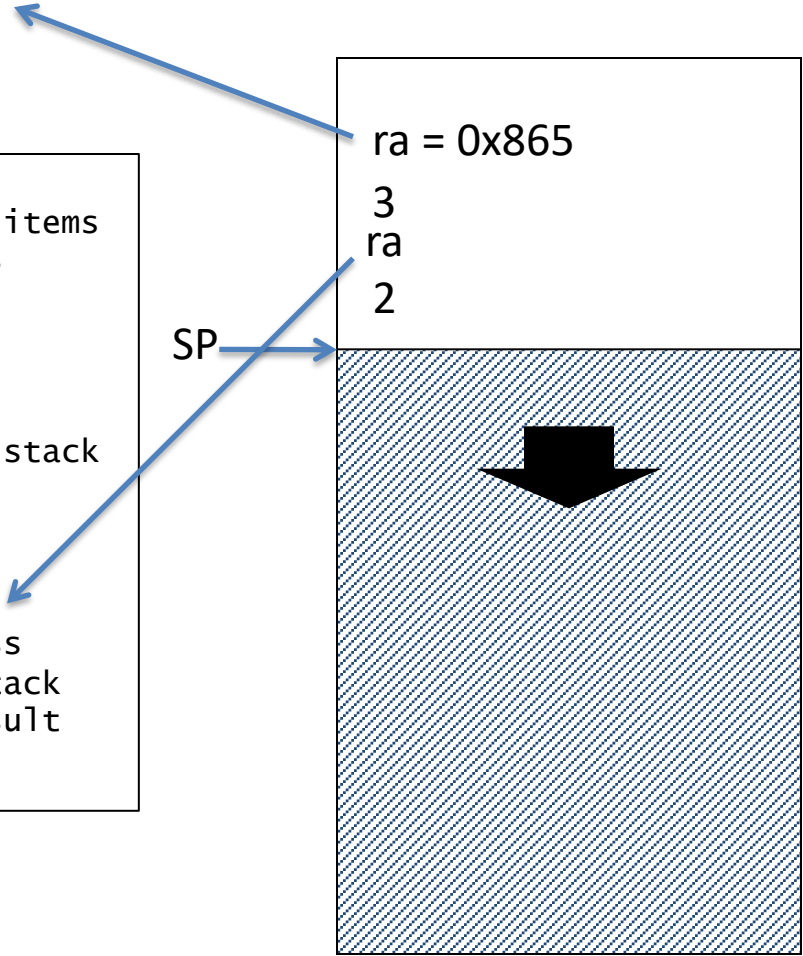
```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
PC → slti $t0, $a0, 2   # test for n < 2
    beq $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra           # and return
L1:  addi $a0, $a0, -1  # else decrement n
    jal fact          # recursive call
    lw   $a0, 0($sp)   # restore original n
    lw   $ra, 4($sp)   # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0 # multiply to get result
    jr   $ra           # and return
```



```
$ra = L1 + 2
$a0 = 2
$v0 =
$t0 = 0
```

# fact

```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
PC → beq $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra           # and return
L1:  addi $a0, $a0, -1  # else decrement n
    jal  fact          # recursive call
    lw   $a0, 0($sp)   # restore original n
    lw   $ra, 4($sp)   # and return address
    addi $sp, $sp, 8   # pop 2 items from stack
    mul  $v0, $a0, $v0 # multiply to get result
    jr   $ra           # and return
```



```

$ra = L1 + 2
$a0 = 1
$v0 =
$t0 = 0

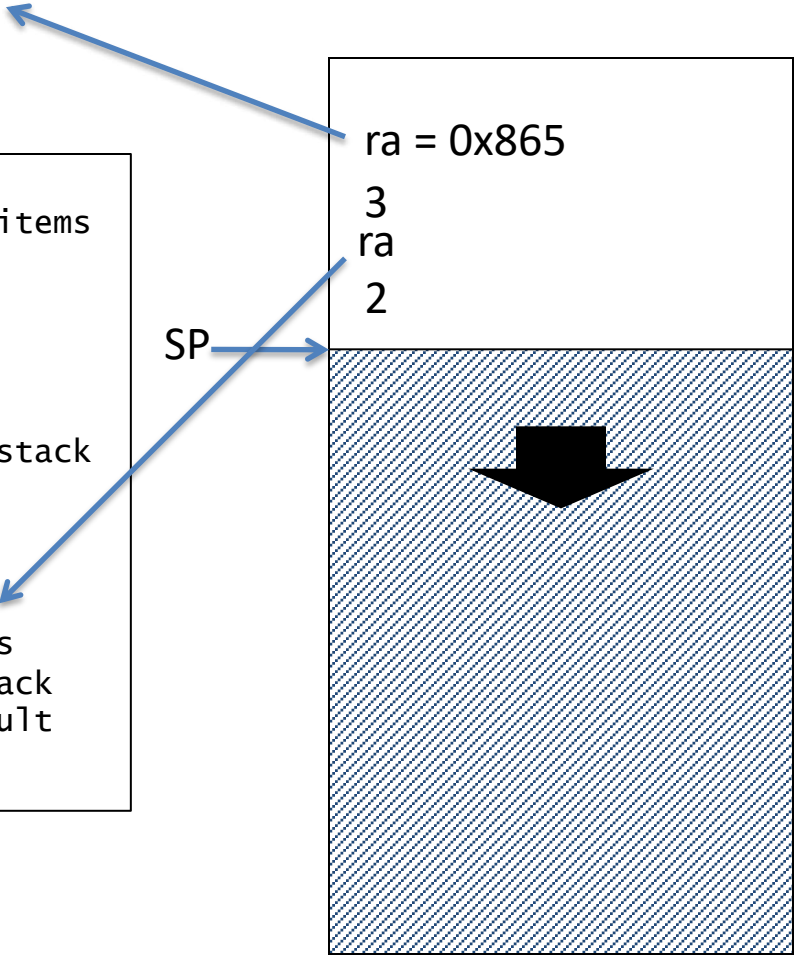
```

# fact

```

fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
PC → L1 → addi $a0, $a0, -1 # else decrement n
        jal  fact        # recursive call
        lw   $a0, 0($sp) # restore original n
        lw   $ra, 4($sp) # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return

```



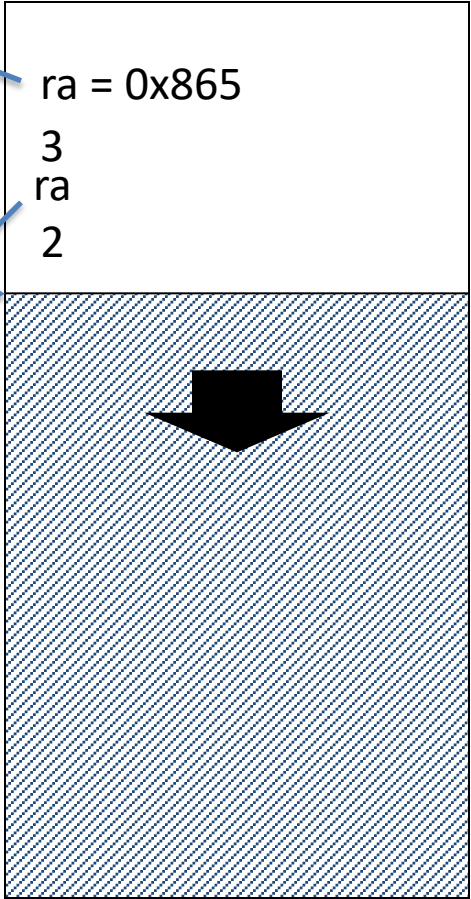
\$ra = L1 + 2  
\$a0 = 1  
\$v0 =  
\$t0 = 0

# fact

```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return
```

PC →

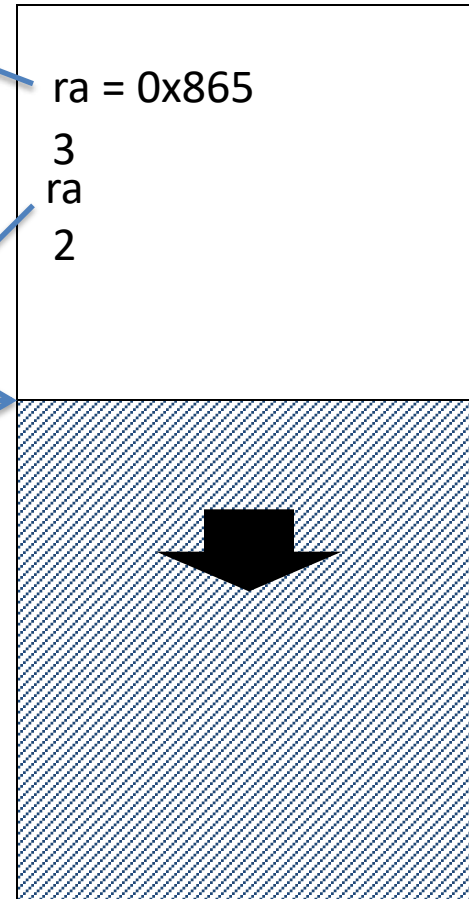
SP →



\$ra = L1 + 2  
\$a0 = 1  
\$v0 =  
\$t0 = 0

# fact

```
fact:
PC → addi $sp, $sp, -8      # adjust stack for 2 items
      sw  $ra, 4($sp)      # save return address
      sw  $a0, 0($sp)      # save argument
      slti $t0, $a0, 2     # test for n < 2
      beq $t0, $zero, L1
      addi $v0, $zero, 1   # if so, result is 1
      addi $sp, $sp, 8     # pop 2 items from stack
      jr  $ra              # and return
L1:   addi $a0, $a0, -1     # else decrement n
      jal fact             # recursive call
      lw  $a0, 0($sp)      # restore original n
      lw  $ra, 4($sp)      # and return address
      addi $sp, $sp, 8     # pop 2 items from stack
      mul $v0, $a0, $v0    # multiply to get result
      jr  $ra              # and return
```





```

$ra = L1 + 2
$a0 = 1
$v0 =
$t0 = 0

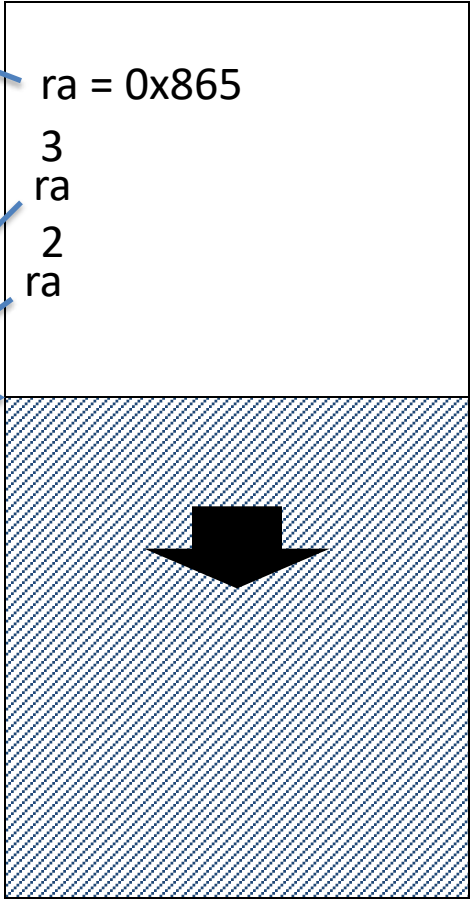
```

# fact

```

fact:
PC → addi $sp, $sp, -8      # adjust stack for 2 items
      sw  $ra, 4($sp)      # save return address
      sw  $a0, 0($sp)      # save argument
      slti $t0, $a0, 2     # test for n < 2
      beq $t0, $zero, L1
      addi $v0, $zero, 1   # if so, result is 1
      addi $sp, $sp, 8     # pop 2 items from stack
      jr  $ra              # and return
L1:   addi $a0, $a0, -1     # else decrement n
      jal fact             # recursive call
      lw  $a0, 0($sp)      # restore original n
      lw  $ra, 4($sp)      # and return address
      addi $sp, $sp, 8     # pop 2 items from stack
      mul $v0, $a0, $v0    # multiply to get result
      jr  $ra              # and return

```



```

$ra = L1 + 2
$a0 = 1
$v0 =
$t0 = 0

```

# fact

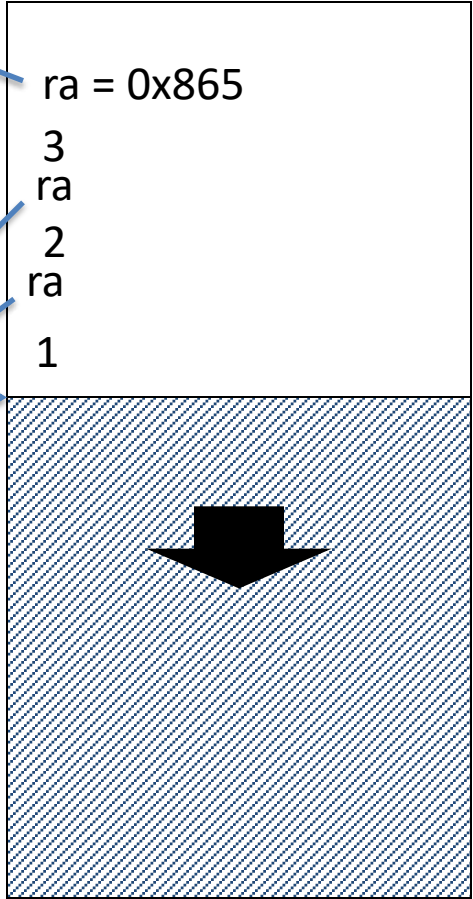
```

fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1 # if so, result is 1
    addi $v0, $zero, 1  # pop 2 items from stack
    addi $sp, $sp, 8    # and return
    jr   $ra
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra           # and return

```

PC →

SP →



```

$ra = L1 + 2
$a0 = 1
$v0 =
$t0 = 1

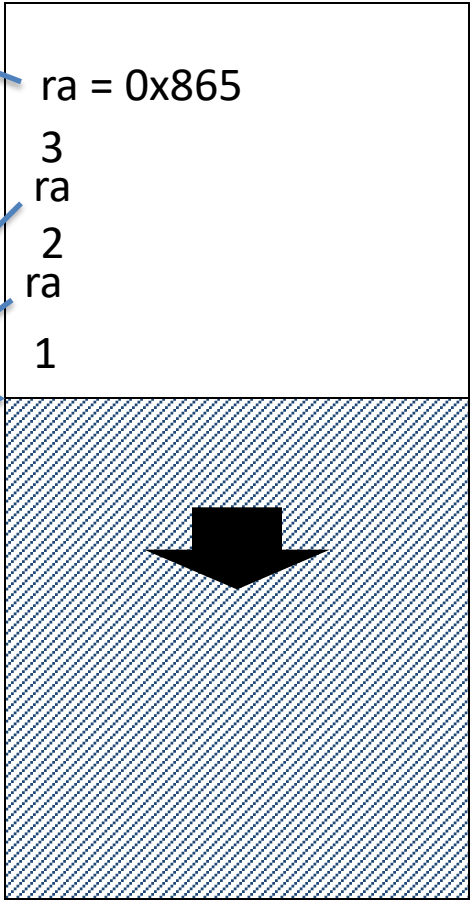
```

# fact

```

fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
PC →   slti $t0, $a0, 2 # test for n < 2
    beq $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra           # and return
L1:   addi $a0, $a0, -1 # else decrement n
    jal fact          # recursive call
    lw   $a0, 0($sp)   # restore original n
    lw   $ra, 4($sp)   # and return address
    addi $sp, $sp, 8   # pop 2 items from stack
    mul  $v0, $a0, $v0 # multiply to get result
    jr   $ra           # and return

```



```

$ra = L1 + 2
$a0 = 1
$v0 =
$t0 = 1

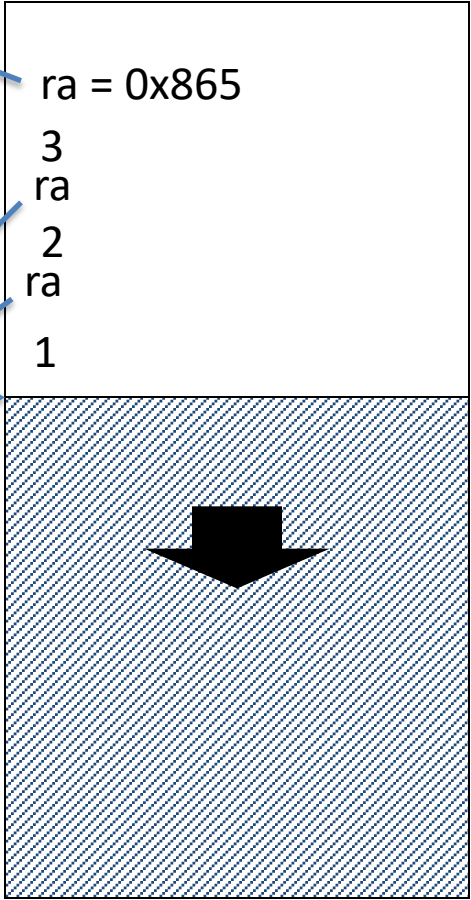
```

# fact

```

fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
PC → beq $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:  addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return

```



```

$ra = L1 + 2
$a0 = 1
$v0 = 1
$t0 = 1

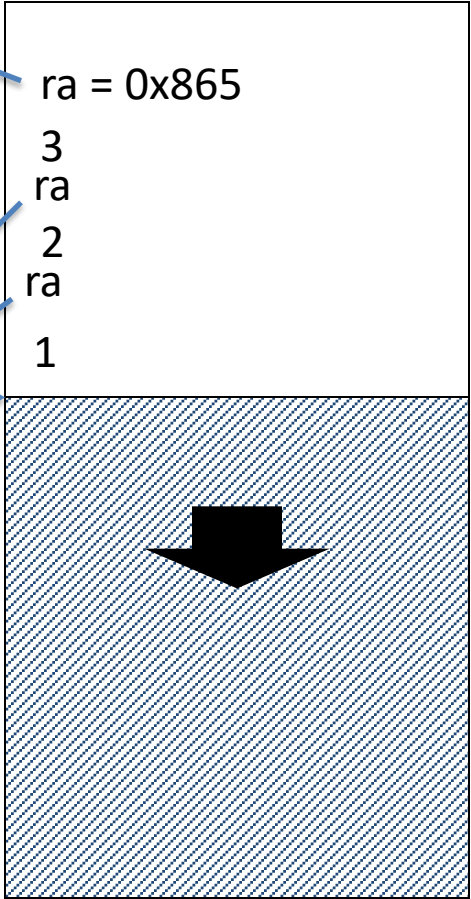
```

# fact

```

fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    PC → addi $v0, $zero, 1 # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1 # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return

```



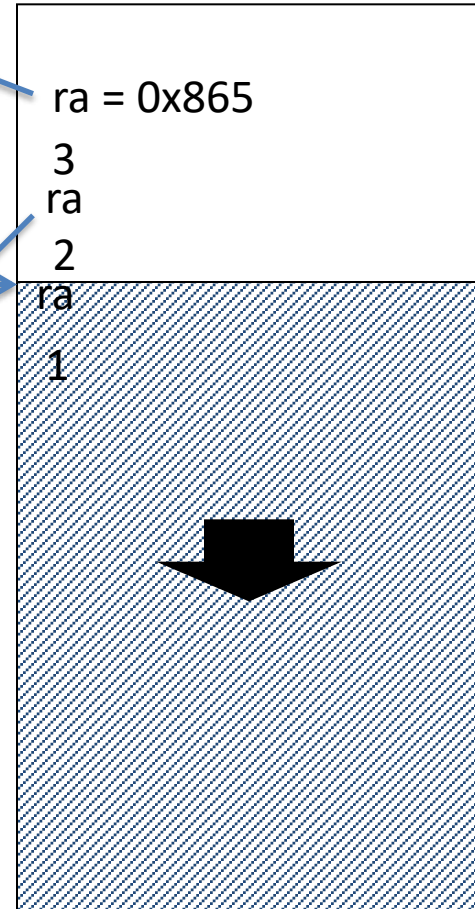
\$ra = L1 + 2  
\$a0 = 1  
\$v0 = 1  
\$t0 = 1

# fact

```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return
```

PC →

SP →



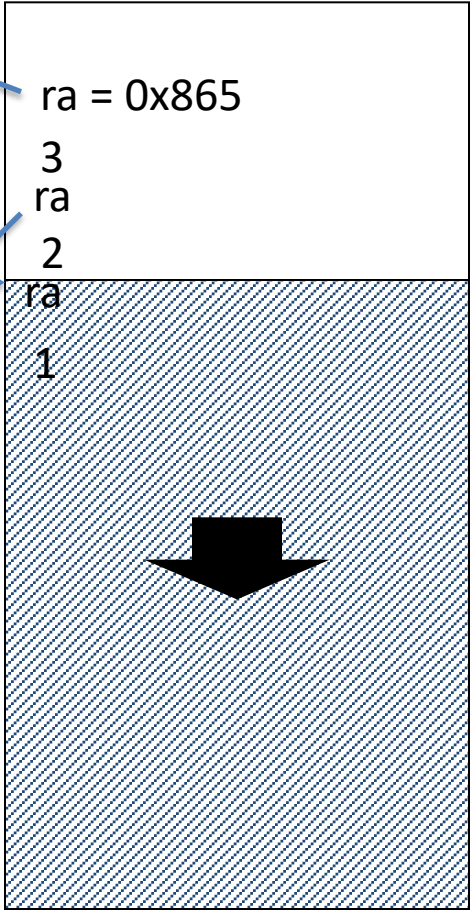
\$ra = L1 + 2  
\$a0 = 1  
\$v0 = 1  
\$t0 = 1

# fact

```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return
```

PC →

SP →



```

$ra = L1 + 2
$a0 = 1
$v0 = 1
$t0 = 1

```

# We will return to

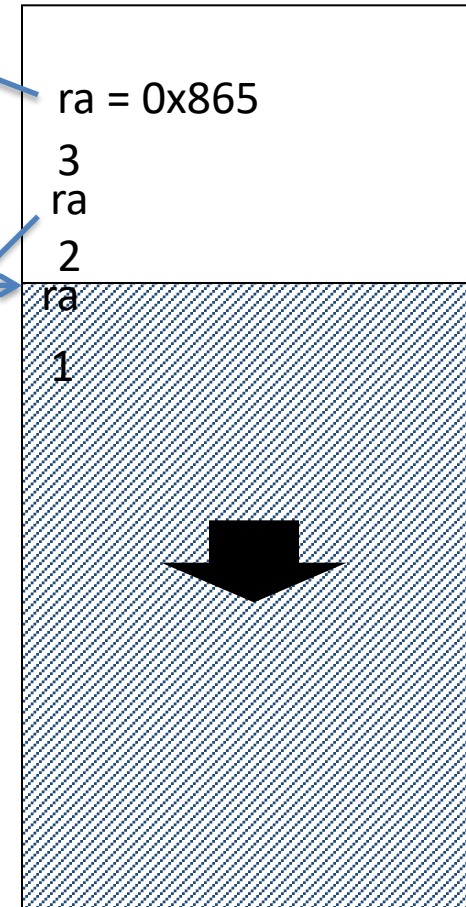
```

fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return

```

PC →

SP →



- A. L1 + 2, because it in \$ra
- B. L1 + 2, because it's the most recent value on the stack
- C. 0x865, because it's the top value on the stack
- D. fact, because it's the procedure call
- E. None of the above



```

$ra = L1 + 2
$a0 = 2
$v0 = 1
$t0 = 1

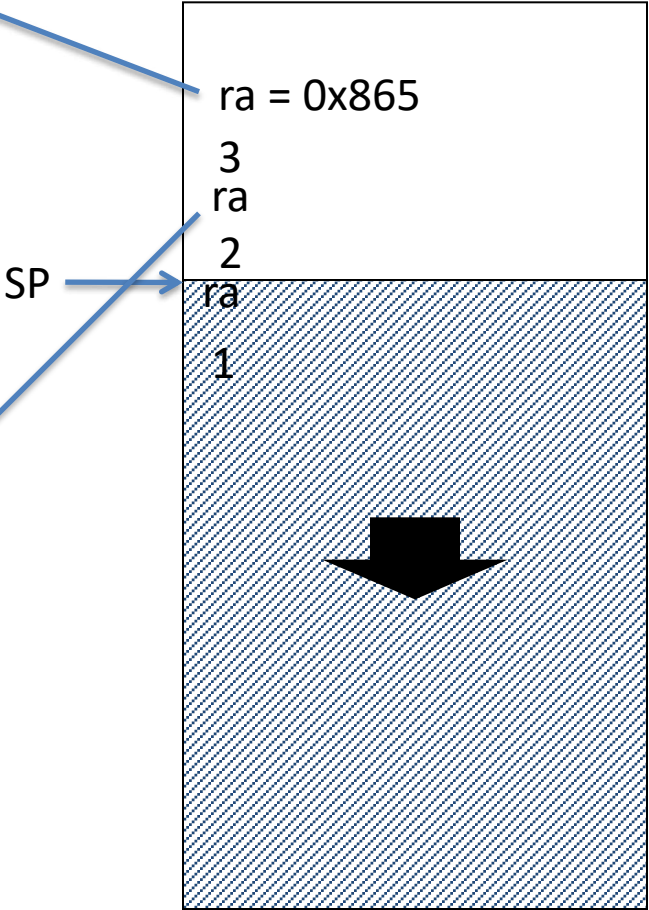
```

# fact

```

fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
PC → lw   $a0, 0($sp)  # restore original n
     lw   $ra, 4($sp)  # and return address
     addi $sp, $sp, 8  # pop 2 items from stack
     mul  $v0, $a0, $v0 # multiply to get result
     jr   $ra         # and return

```



$\$ra = L1 + 2$

$\$a0 = 2$

$\$v0 = 1$

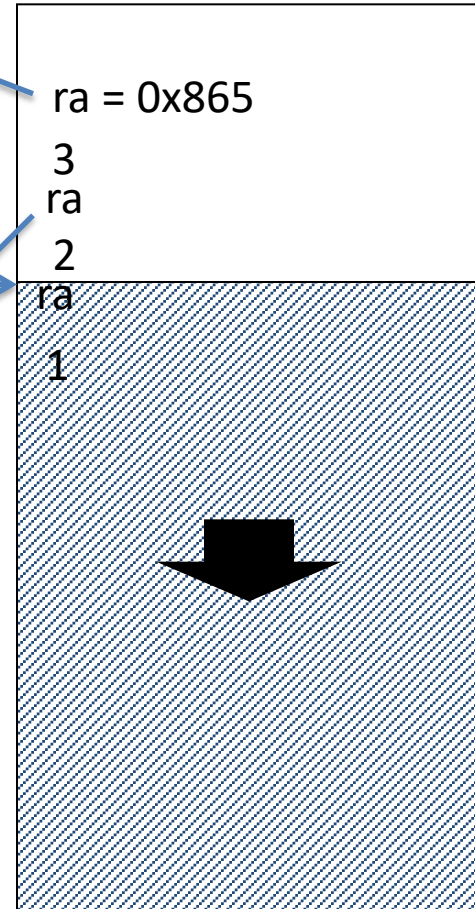
$\$t0 = 1$

# fact

```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return
```

PC →

SP →



```

$ra = L1 + 2
$a0 = 2
$v0 = 1
$t0 = 1

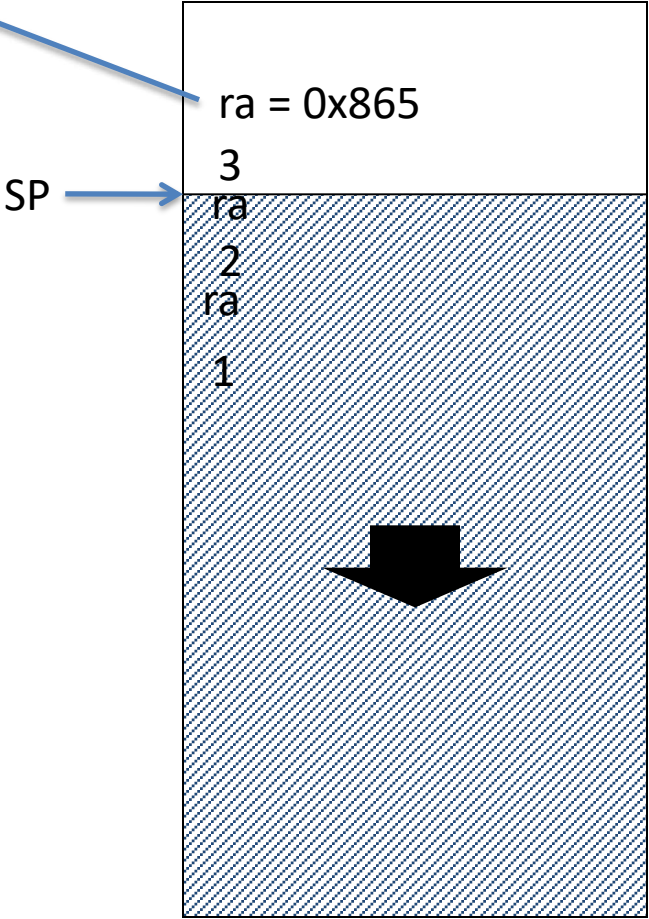
```

# fact

```

fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
PC → addi $sp, $sp, 8  # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return

```



```

$ra = L1 + 2
$a0 = 2
$v0 = 2
$t0 = 1

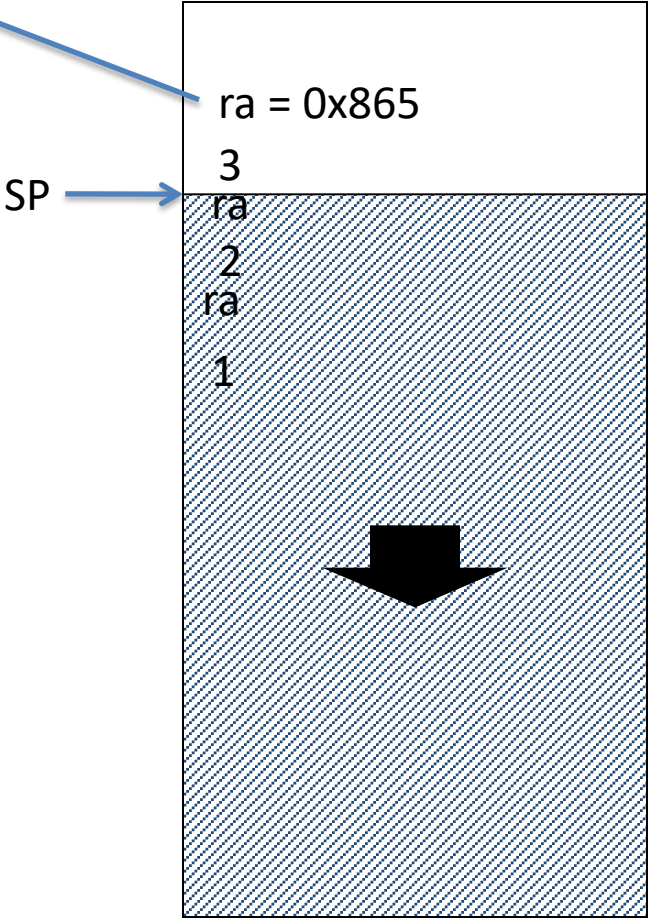
```

# fact

```

fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return

```



```

$ra = L1 + 2
$a0 = 2
$v0 = 2
$t0 = 1

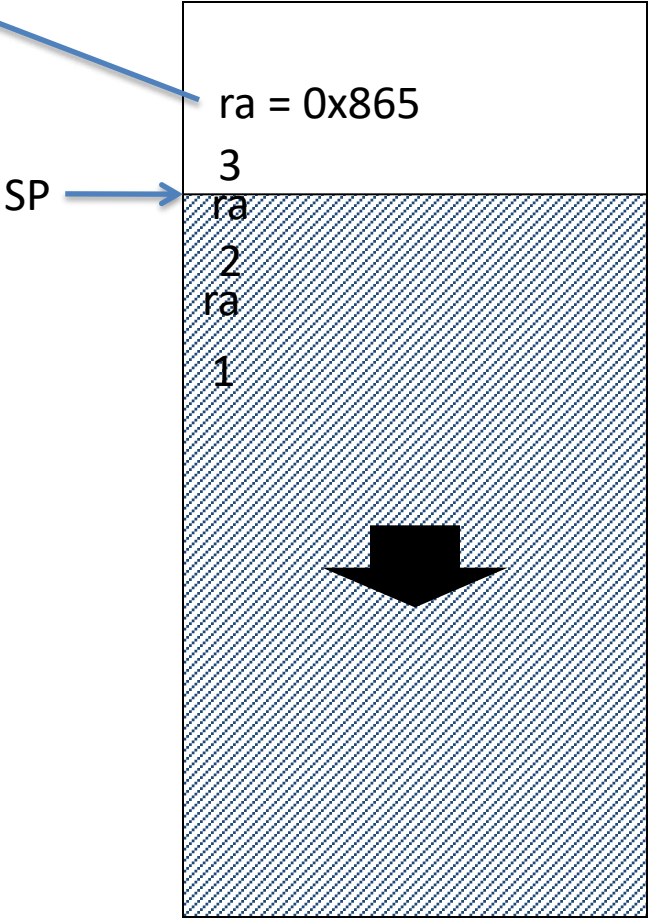
```

# fact

```

fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return

```



```

$ra = L1 + 2
$a0 = 3
$v0 = 2
$t0 = 1

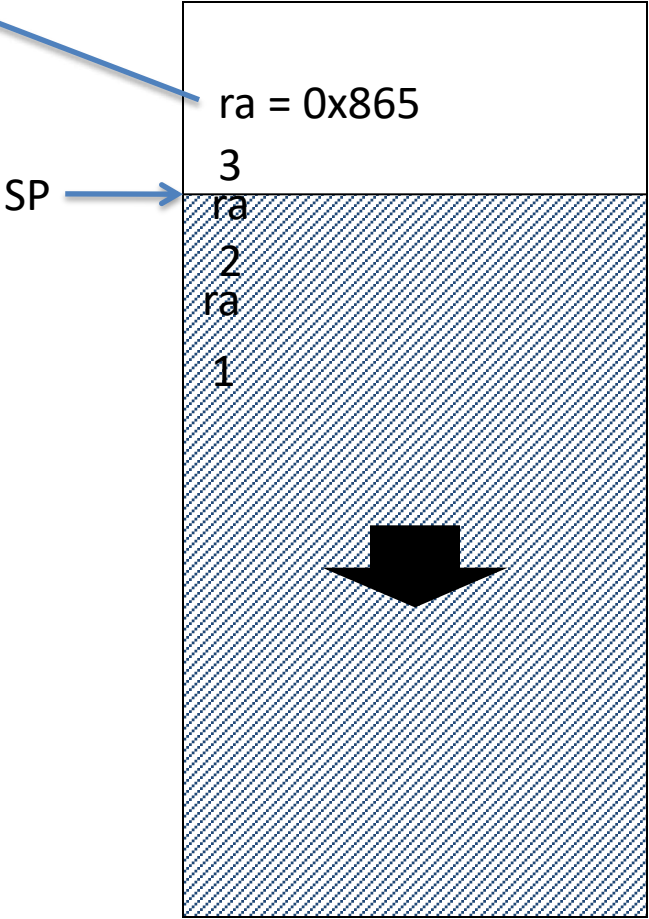
```

# fact

```

fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
PC → lw   $a0, 0($sp)   # restore original n
     lw   $ra, 4($sp)   # and return address
     addi $sp, $sp, 8   # pop 2 items from stack
     mul  $v0, $a0, $v0 # multiply to get result
     jr   $ra          # and return

```



**\$ra = 0x865**

\$a0 = 3

\$v0 = 2

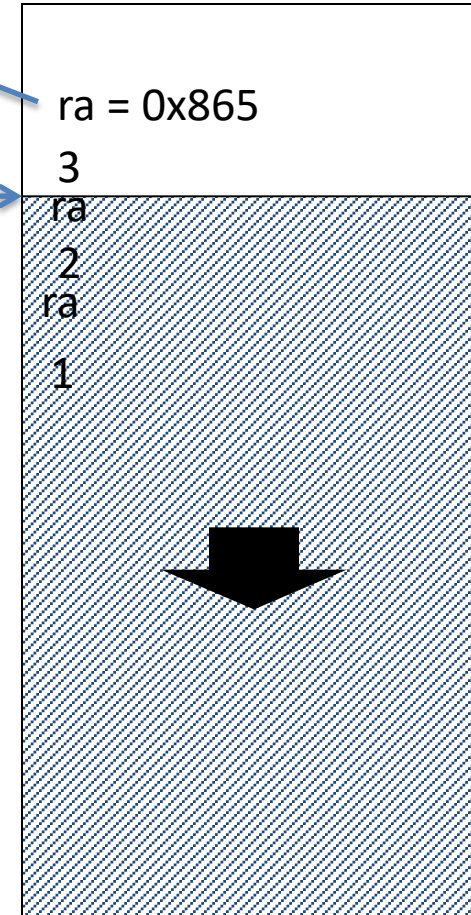
\$t0 = 1

# fact

```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return
```

PC →

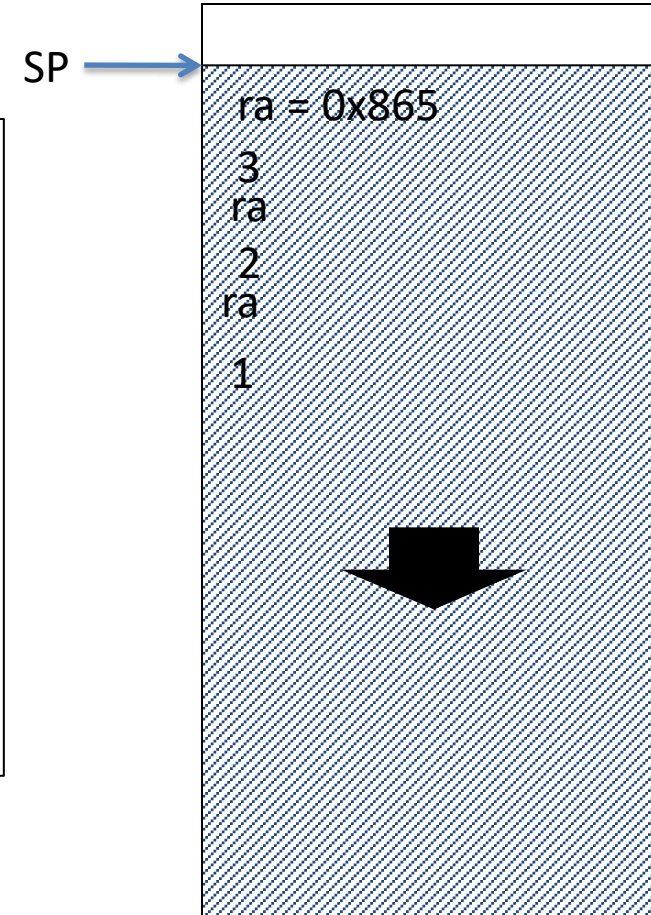
SP →



\$ra = 0x865  
\$a0 = 3  
\$v0 = 2  
\$t0 = 1

# fact

```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
PC → addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return
```





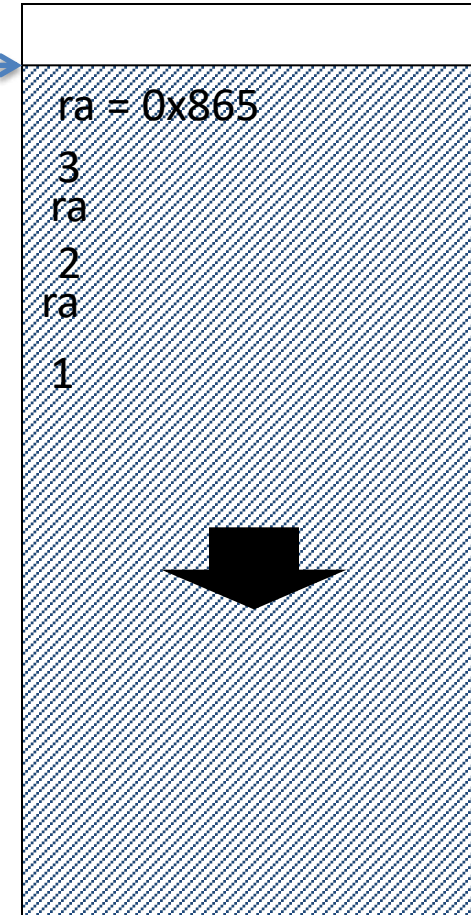
\$ra = 0x865  
\$a0 = 3  
\$v0 = 6  
\$t0 = 1

# fact

```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return
```

PC →

SP →

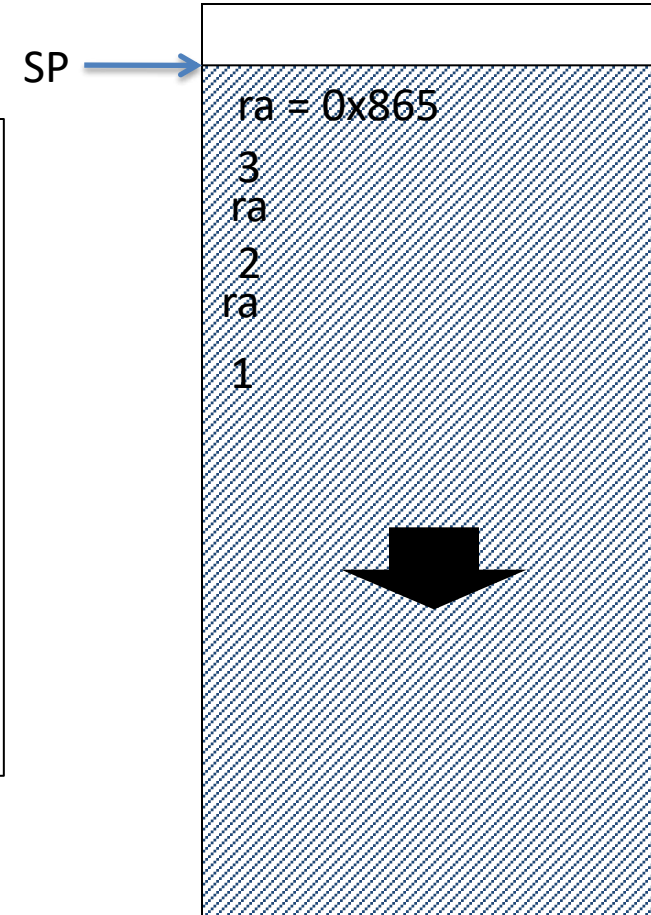


\$ra = 0x865  
\$a0 = 3  
\$v0 = 6  
\$t0 = 1

# fact

```
fact:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw   $ra, 4($sp)    # save return address
    sw   $a0, 0($sp)    # save argument
    slti $t0, $a0, 2    # test for n < 2
    beq  $t0, $zero, L1
    addi $v0, $zero, 1  # if so, result is 1
    addi $sp, $sp, 8    # pop 2 items from stack
    jr   $ra            # and return
L1:   addi $a0, $a0, -1  # else decrement n
    jal  fact           # recursive call
    lw   $a0, 0($sp)    # restore original n
    lw   $ra, 4($sp)    # and return address
    addi $sp, $sp, 8    # pop 2 items from stack
    mul  $v0, $a0, $v0  # multiply to get result
    jr   $ra            # and return
```

PC →



Why store registers relative to the stack pointer, rather than at some set memory location?

- A. Saves space.
- B. Easier to figure out where we stored things.
- C. Methods won't overwrite each other's saves.
- D. None of the above

# Reading

- Next lecture: More Stack
- Problem set 4 due Friday
- Lab 3 due Sunday