

INF 212

ANALYSIS OF PROG. LANGS  
*PROCEDURES & FUNCTIONS*

Instructors: Crista Lopes  
Copyright © Instructors.

# Subroutines aka Procedures

- Historically: blocks of instructions executed several times during program execution
- May have 0 or more input arguments
- May have 0 or more output arguments
- May perform IO, side effects
- Mid-50s

# Functions

- Take 0 or more input arguments
- Return one value
- Used as *expressions*
- Additional constraint for **pure** functions:
  - ▣ No IO, no side effects

# Procedures vs. Functions

- Distinction existed as early as 1958 (FORTRAN)

```
subroutine square_cube(i,isquare,icube)
  integer, intent(in)  :: i                ! input
  integer, intent(out) :: isquare,icube    ! output
  isquare = i**2
  icube   = i**3
end subroutine square_cube
```

```
program xx
  implicit none
  integer :: i,isq,icub
  i = 4
  call square_cube(i,isq,icub)
  print*, "i,i^2,i^3=", i,isq,icub
end program xx
```

# Procedures vs. Functions

- Distinction existed as early as 1958 (FORTRAN)

```
function func(i) result(j)
    integer, intent(in) :: i ! input
    integer              :: j ! output
    j = i**2 + i**3
end function func
```

```
program xfunc
    implicit none
    integer :: i
    integer :: func
    i = 3
    print*, "sum of the square and cube of", i, " is", func(i)
end program xfunc
```

Additionally, Fortran has a **pure** keyword for pure functions

# Procedures vs. Functions

- Distinction was lost at some point, mainstream PLs merged the two concepts into one
  - ▣ C/C++, Java, Python, Perl, PHP, ... No distinction:
    - Procedures can also return values
  - ▣ Lisp, ML, Haskell, ... Only functions, but:
    - Functions can be pure or impure

# “Pure” Functional Programming

- Mathematical functions
  - ▣ No side effects
  - ▣ No IO (other than at the beginning and the end)
- “High-order” functions
  - ▣ Functions can take functions as arguments
  - ▣ Functions can return functions as values
- More on this later...



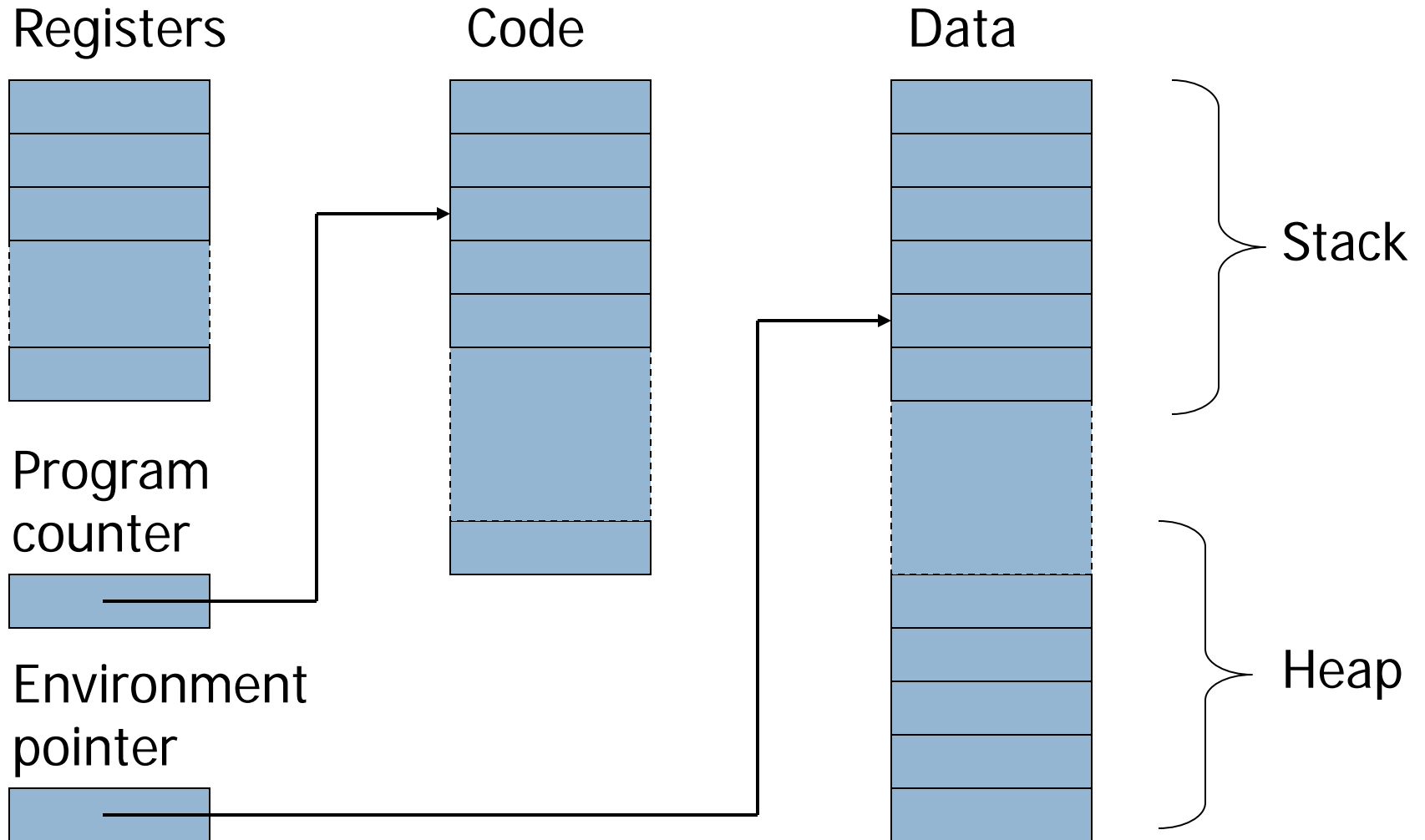
# Function/procedure calls

Implementation details



# Simplified Machine Model


slide 9



# Function definition

```
def: fact(n) = if n<=1 then 1  
              else n * fact(n-1)
```

```
...  
call: fact(3)  
...  
?
```



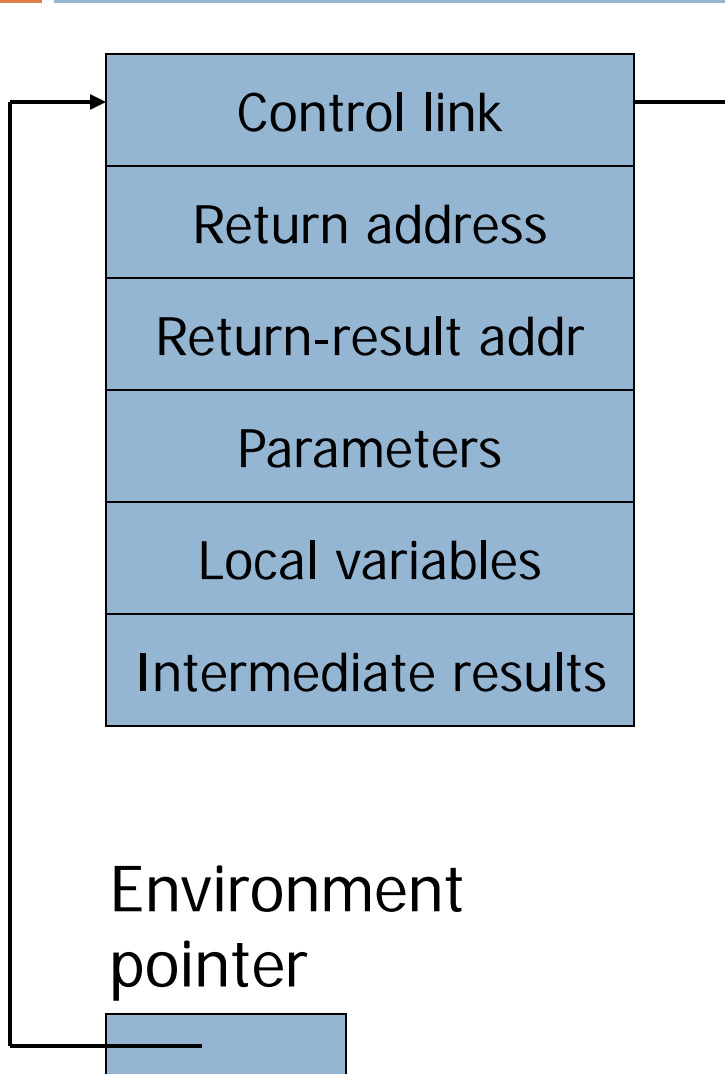
# Activation Records for Functions

slide 11

- Block of information (“frame”) associated with each function call, including:
  - ▣ Parameters
  - ▣ Local variables
  - ▣ Return address
  - ▣ Location to put return value when function exits
  - ▣ Control link to the caller’s activation record
  - ▣ Saved registers
  - ▣ Temporary variables and intermediate results
  - ▣ (not always) Access link to the function’s static parent

# Activation Record Layout

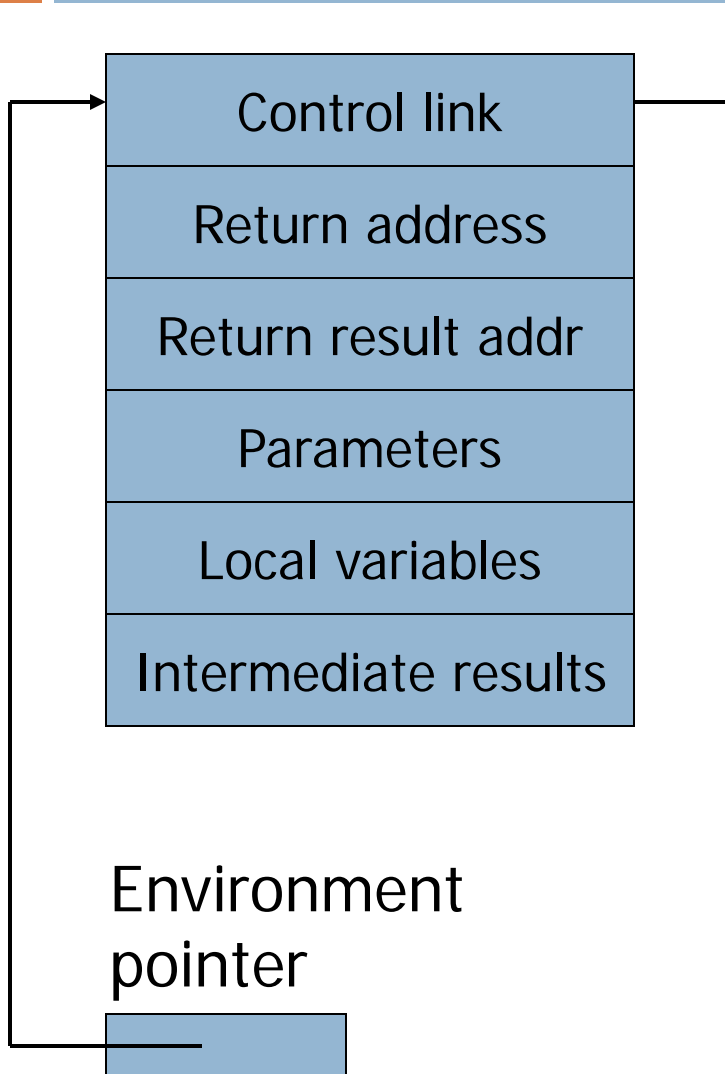
slide 12



- Return address
  - ▣ Location of code to execute on function return
- Return-result address
  - ▣ Address in activation record of calling block to receive returned value
- Parameters
  - ▣ Locations to contain data from calling block

# Example

slide 13



## □ Function

$\text{fact}(n) = \text{if } n \leq 1 \text{ then } 1$   
 $\text{else } n * \text{fact}(n-1)$

▣ Return result address:  
location to put  $\text{fact}(n)$

## □ Parameter

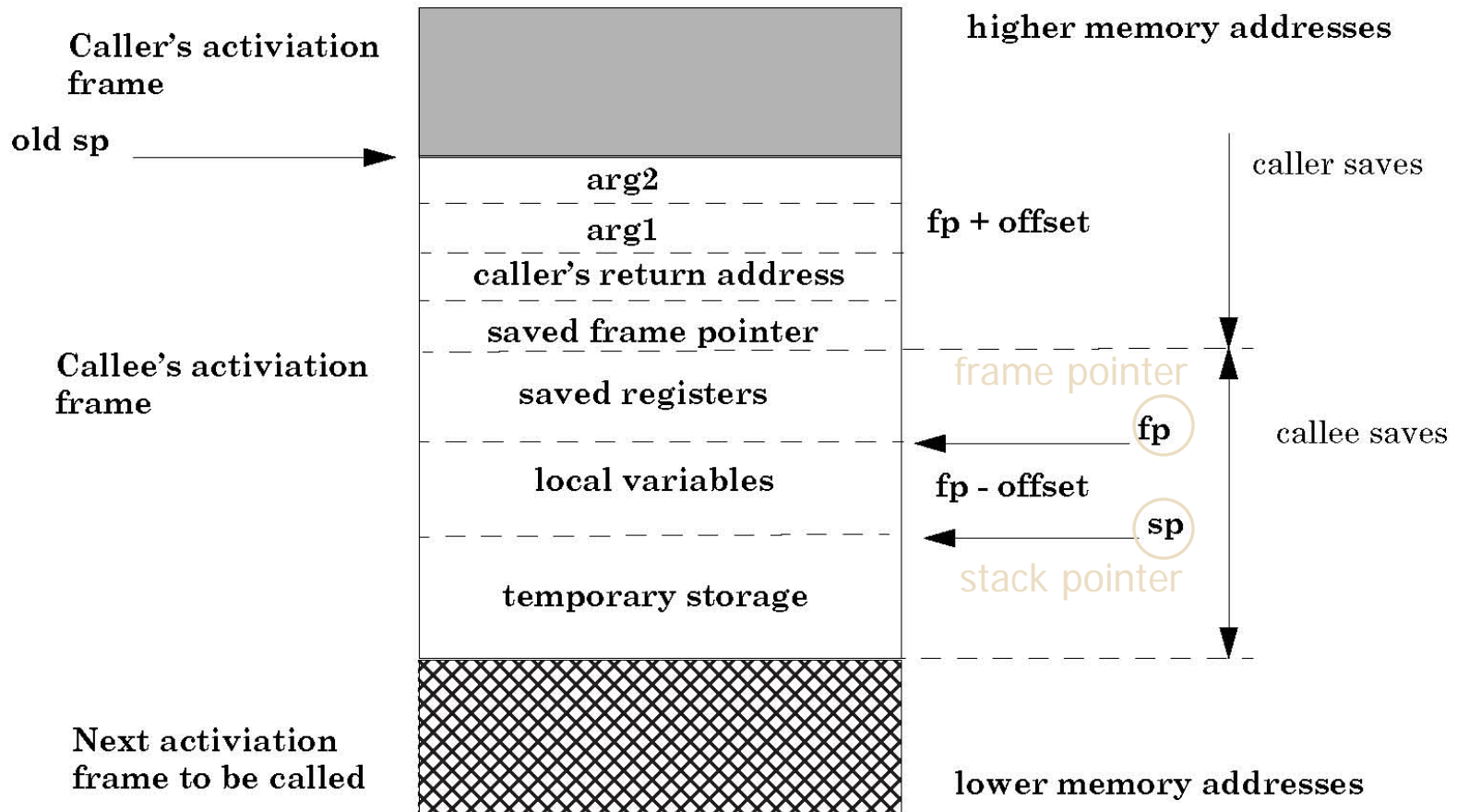
▣ Set to value of  $n$  by calling  
sequence

## □ Intermediate result

▣ Locations to contain value of  
 $\text{fact}(n-1)$

# Typical x86 Activation Record

slide 14



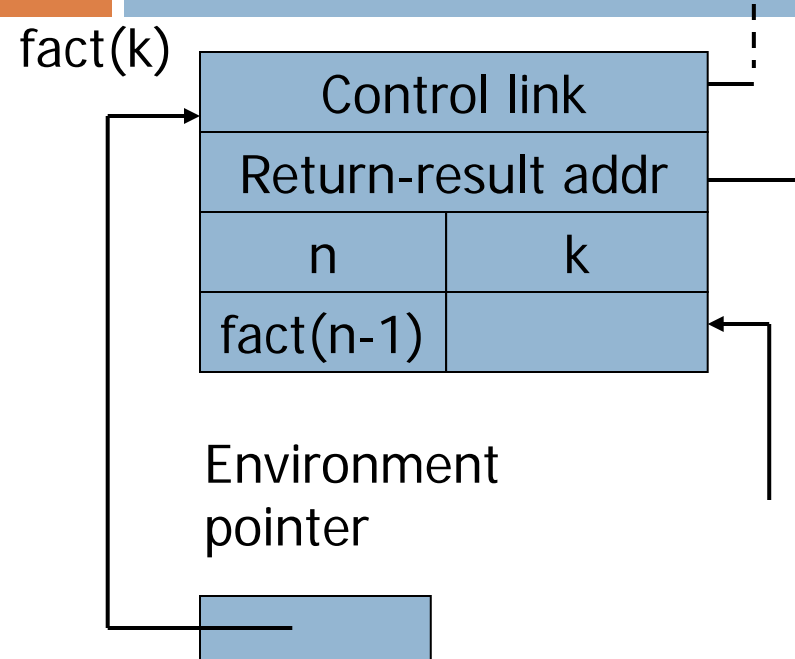
# Run-Time Stack

slide 15

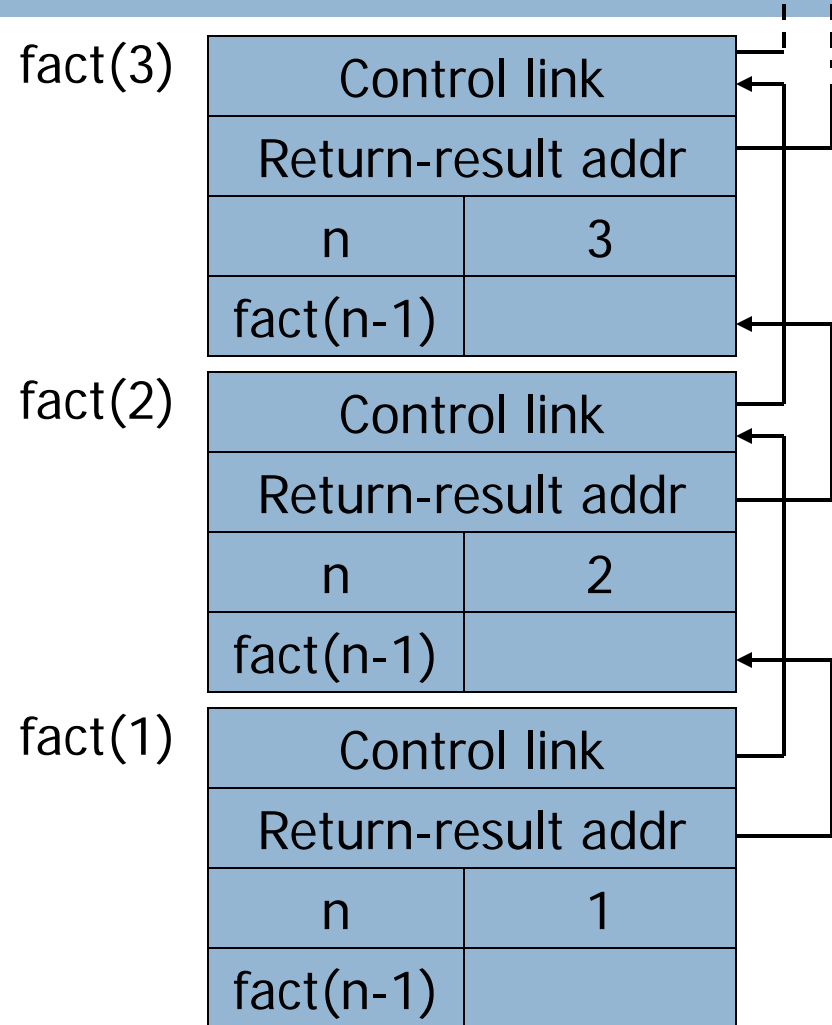
- Activation records are kept on the **stack**
  - ▣ Each new call pushes an activation record
  - ▣ Each completing call pops the topmost one
  - ▣ Stack has all records of all active calls at any moment during execution (topmost record = most recent call)
- Example: `fact(3)`
  - ▣ Pushes one activation record on the stack, calls `fact(2)`
  - ▣ This call pushes another record, calls `fact(1)`
  - ▣ This call pushes another record, resulting in three activation records on the stack

# Function Call

slide 16



$\text{fact}(n) = \text{if } n \leq 1 \text{ then } 1$   
 $\text{else } n * \text{fact}(n-1)$

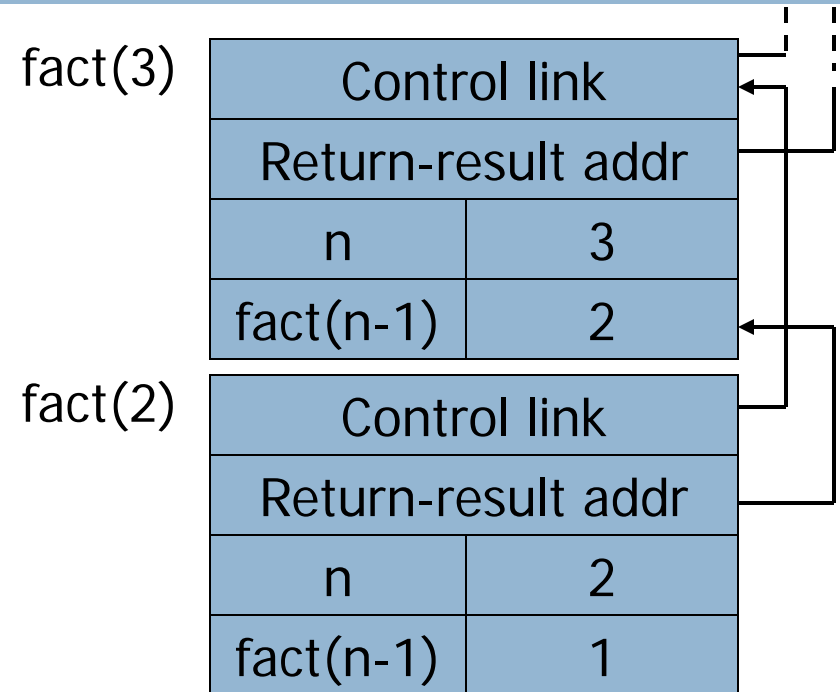
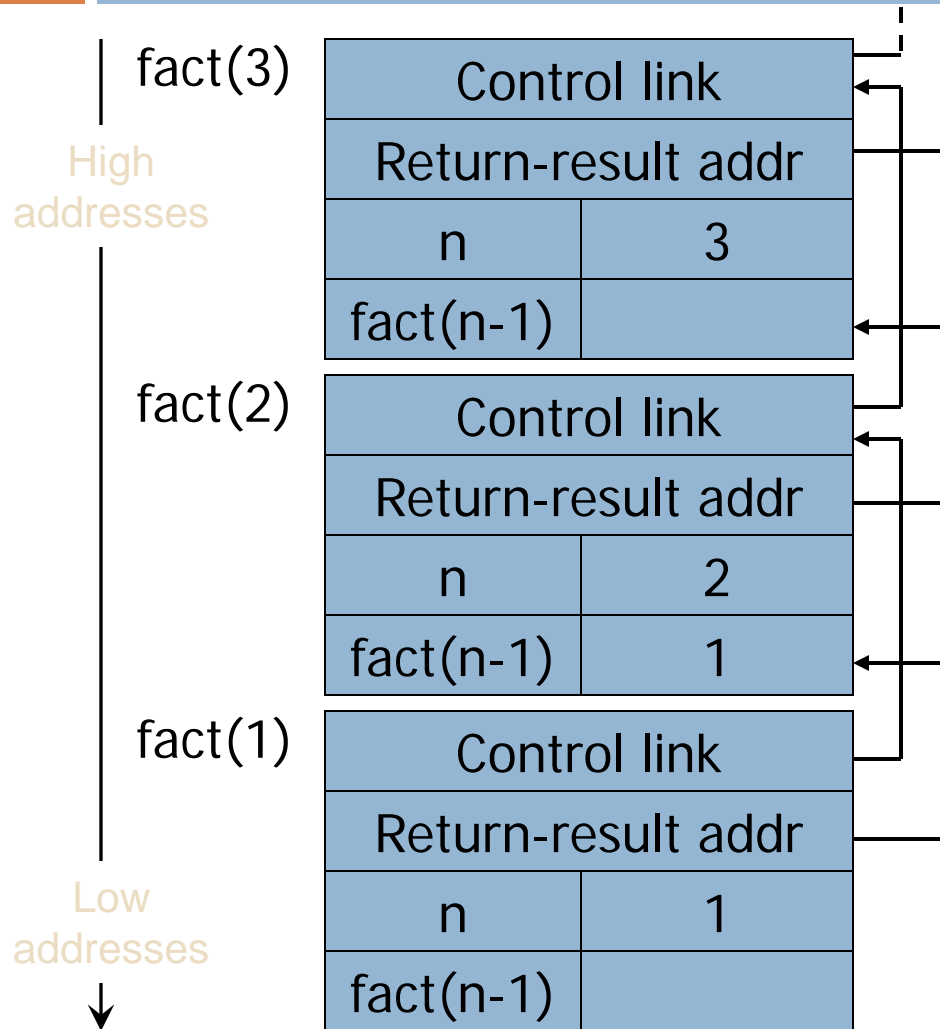


Return address omitted; would be a  
pointer into code segment



# Function Return

slide 17



$\text{fact}(n) = \text{if } n \leq 1 \text{ then } 1$   
 $\text{else } n * \text{fact}(n-1)$