Recursion

What is recursion?

- Sometimes, the best way to solve a problem is by solving a ***smaller version*** of the exact same problem first
- Recursion is a technique that solves a problem by solving a ***smaller problem*** of the same type

Functions that call themselves (*recursive functions*)

```
int f(int x)
{
 int y;
  if(x==0)
    return 1;
  else {
    y = 2 * f(x-1);
    return y+1;
 }
}
```

Problems defined recursively

- There are many problems whose solution can be defined recursively

Example: *n factorial*

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ (n-1)!*n & \text{if } n > 0 \end{cases}$$ (*recursive* solution)

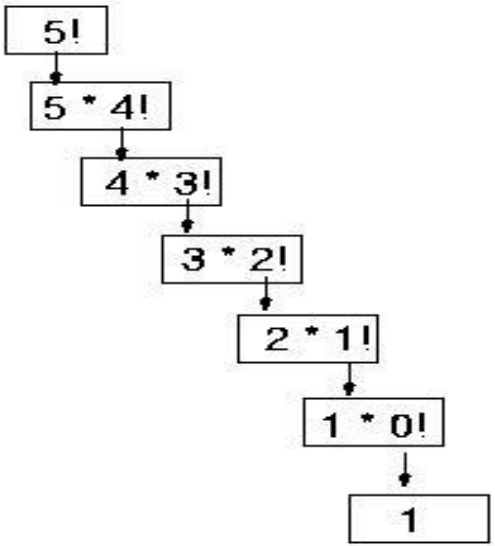
$$n! = \begin{cases} 1 & \text{if } n = 0 \\ 1*2*3*\ldots*(n-1)*n & \text{if } n > 0 \end{cases}$$ (*closed form* solution)

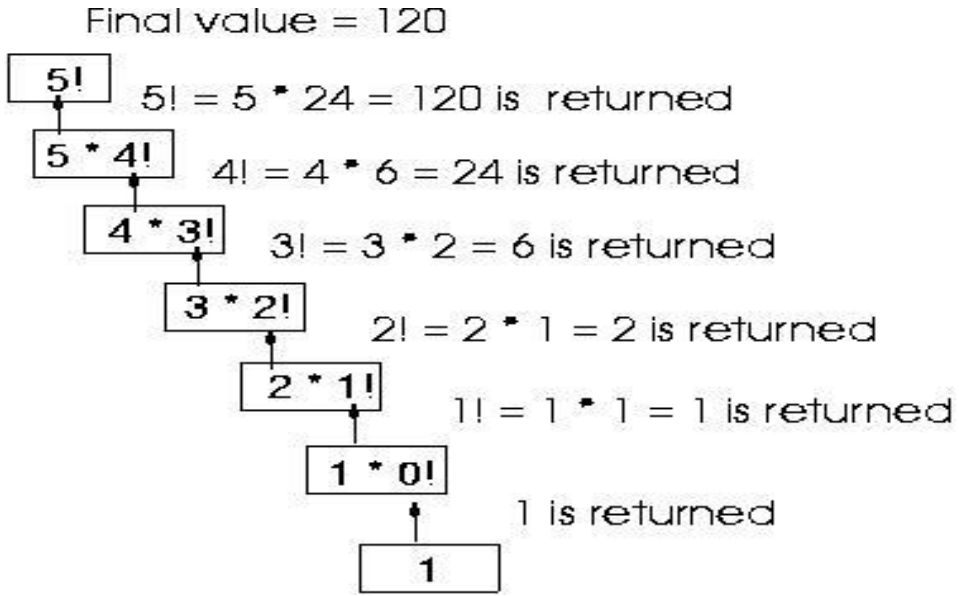
Factorial function

- Recursive implementation

```
int Factorial(int n)
{
 if (n==0)  // base case
   return 1;
 else
   return n * Factorial(n-1);
}
```

Recursive calls





Factorial function (cont.)

- Iterative implementation

```
int Factorial(int n)
{
 int fact = 1;
  for(int count = 2; count <= n; count++)
     fact = fact * count;
 return fact;
}
```

Another example: *n* choose *k* (combinations)

- Given *n* things, how many different sets of size *k* can be chosen?

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \quad ,1 < k < n \qquad \textit{(recursive solution)}$$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad ,1 < k < n \qquad \textit{(closed-form solution)}$$

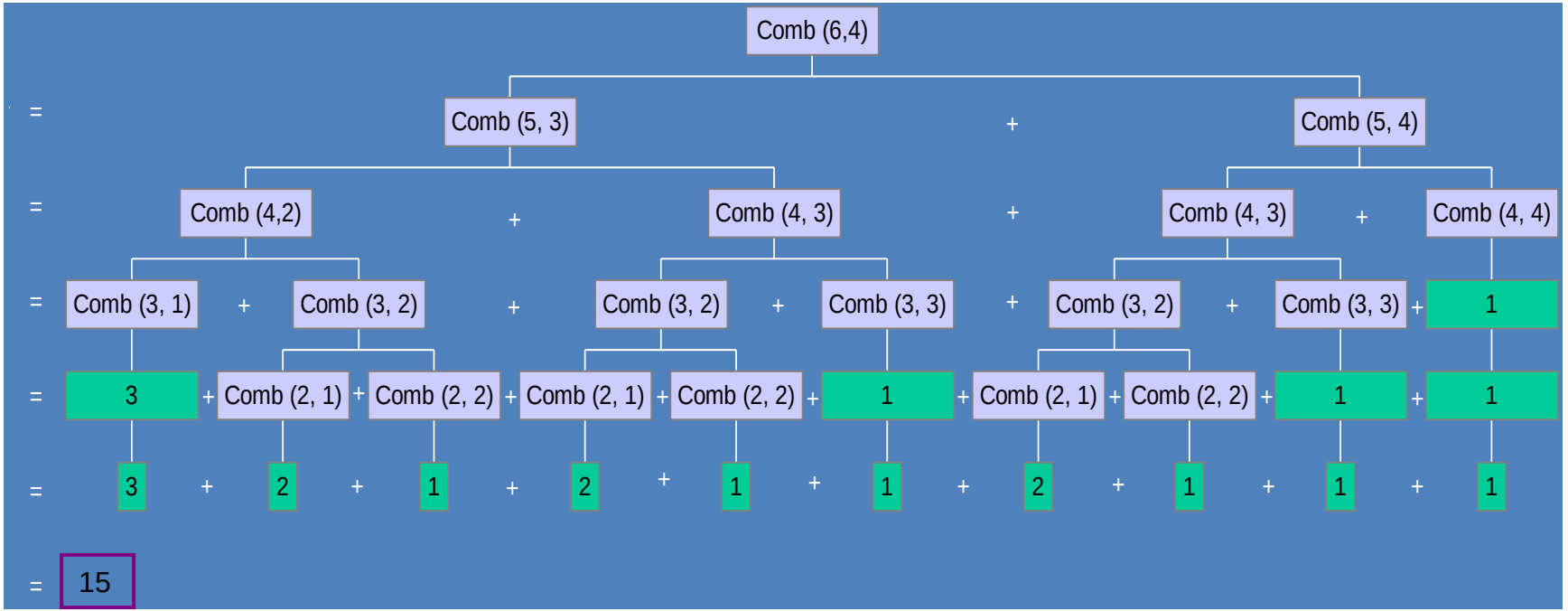
with base cases:

$$\binom{n}{1} = n \ (k = 1), \quad \binom{n}{n} = 1 \ (k = n)$$

n choose *k* (combinations)

```
int Comb(int n, int k)
{
 if(k == 1)  // base case 1
   return n;
 else if (n == k)  // base case 2
   return 1;
 else
   return(Comb(n-1, k) + Comb(n-1, k-1));
}
```

Recursion can be very inefficient is some cases



Recursion vs. iteration

- Iteration can be used in place of recursion
  - An iterative algorithm uses a *looping construct*
  - A recursive algorithm uses a *branching structure*
- Recursive solutions are often less efficient, in terms of both *time* and *space*, than iterative solutions
- Recursion can simplify the solution of a problem, often resulting in shorter, more easily understood source code

How do I write a recursive function?

- Determine the size factor
- Determine the base case(s)
  - the one for which you know the answer
- Determine the general case(s)
  - the one where the problem is expressed as a smaller version of itself

Three-Question Verification Method

The Base-Case Question:

- Is there a nonrecursive way out of the function, and does the routine work correctly for this "base" case?

The Smaller-Caller Question:

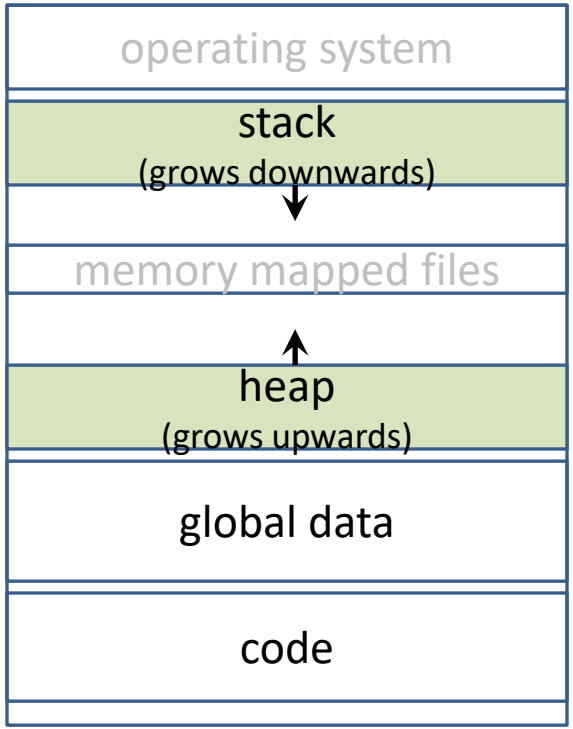
- Does each recursive call to the function involve a smaller case of the original problem, leading certainly to the base case?

The General-Case Question:

- Assuming that the recursive call(s) work correctly, does the whole function work correctly?

Background: Runtime Memory Organization

Layout of an executing process's virtual memory:



0xffffffff

high addresses

0x00000000

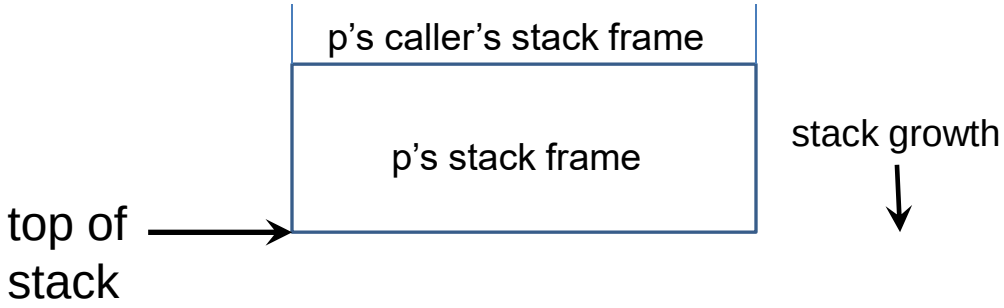
low addresses

Background: Runtime Memory Organization

Code:

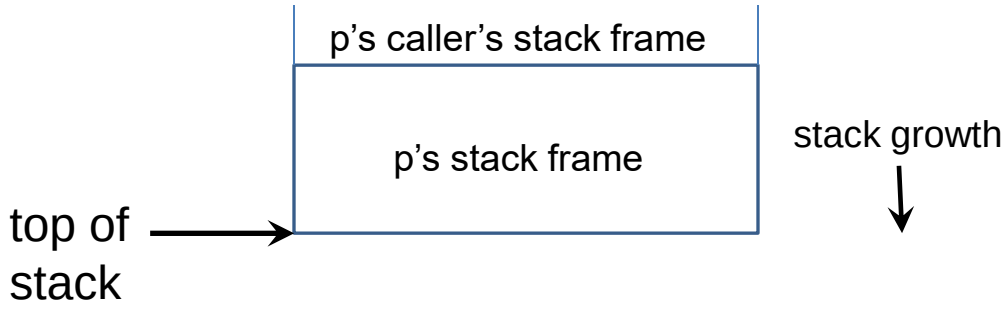
```
p(…) {
  …
  q(…);
  s(…);
}

q(…) {
  …
  r(…);
}

r(…)
{
  …
}

s(…) {
  …
}
```

Runtime stack:



Background: Runtime Memory Organization

```
p(…) {
  …
  q(…);
  s(…);
}

q(…) {
  …
  r(…);
}

r(…)
{
  …
}

s(…) {
  …
}
```



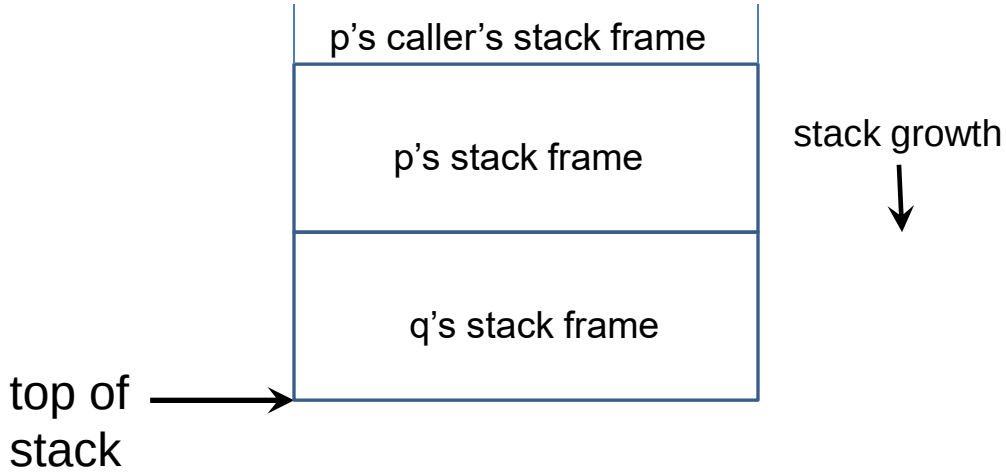
Background: Runtime Memory Organization

Code:

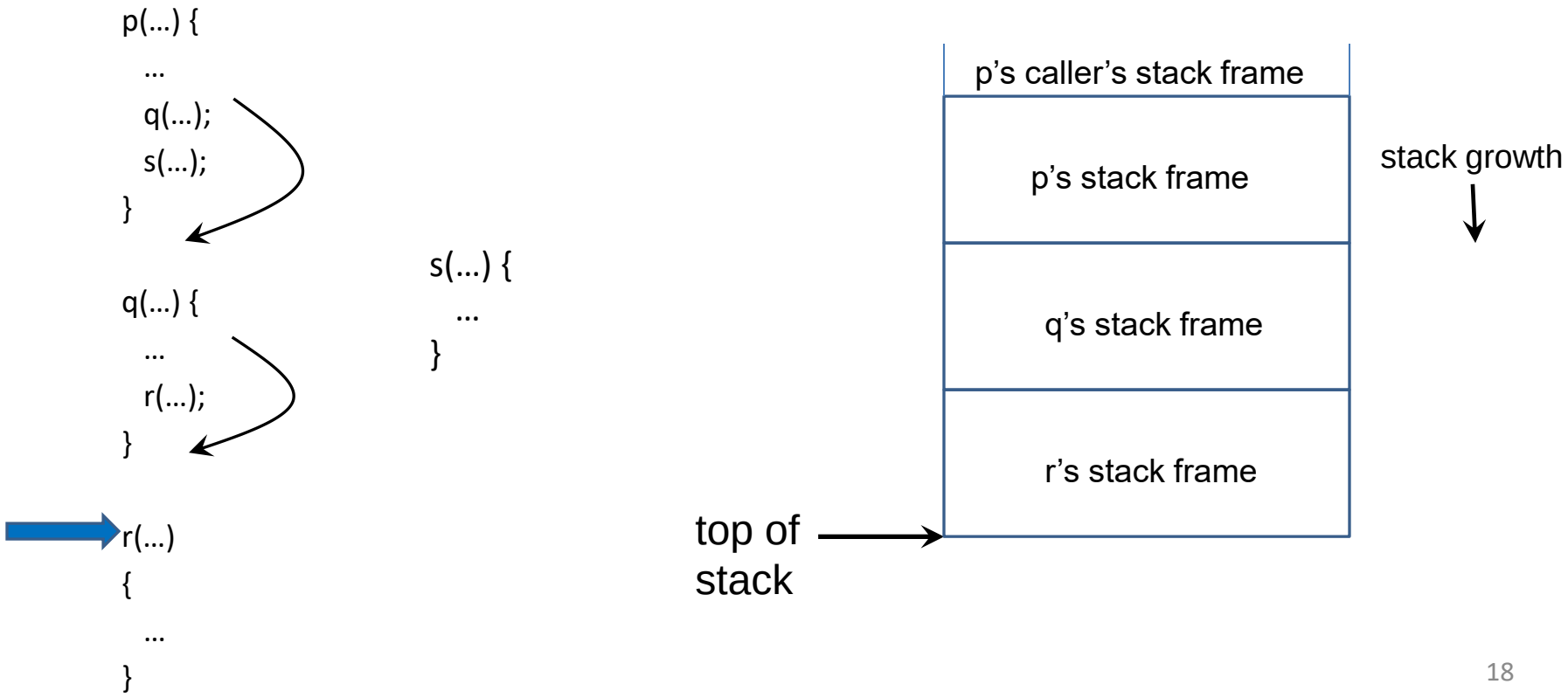
```
p(…) {
  …
  q(…);
  s(…);
}

q(…) {
  …
  r(…);
}

r(…)
{
```

```
s(…) {
  …
}
```

Runtime stack:



Background: Runtime Memory Organization

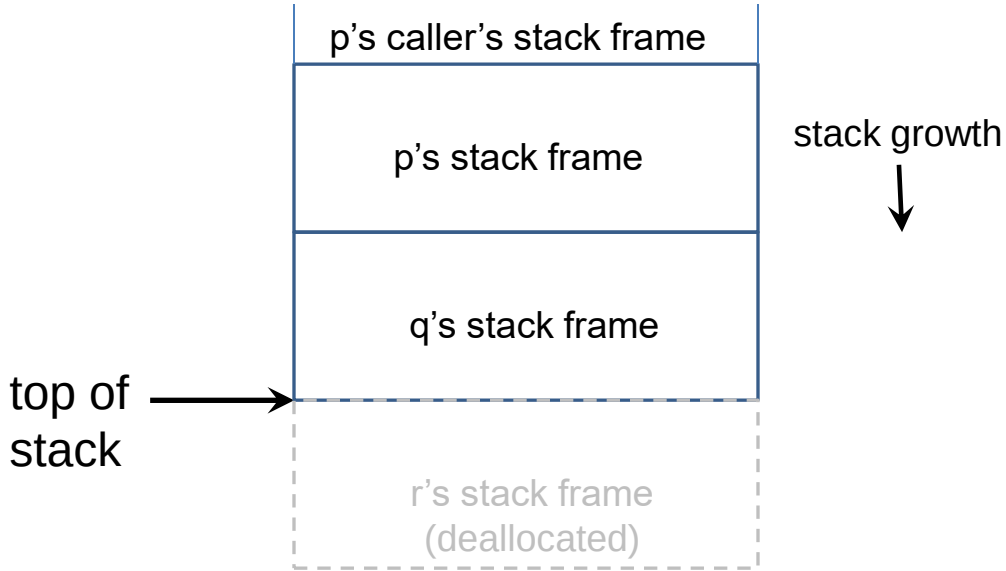


Background: Runtime Memory Organization

Code:

```
p(…) {
  …
  q(…);
  s(…);
}

s(…) {
  …
}

q(…) {
  …
  r(…);
}

r(…)
{
```

Runtime stack:



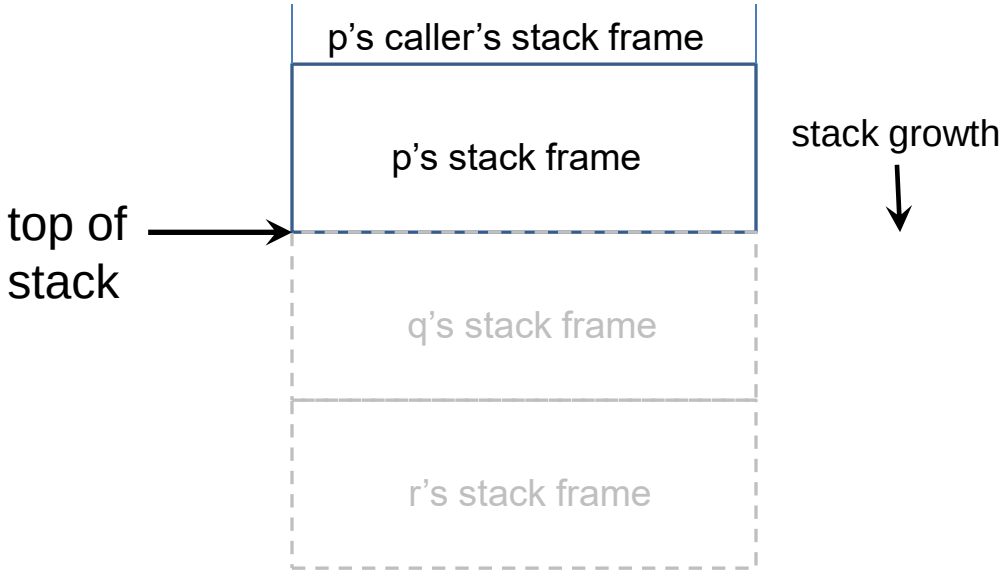
Background: Runtime Memory Organization

Code:

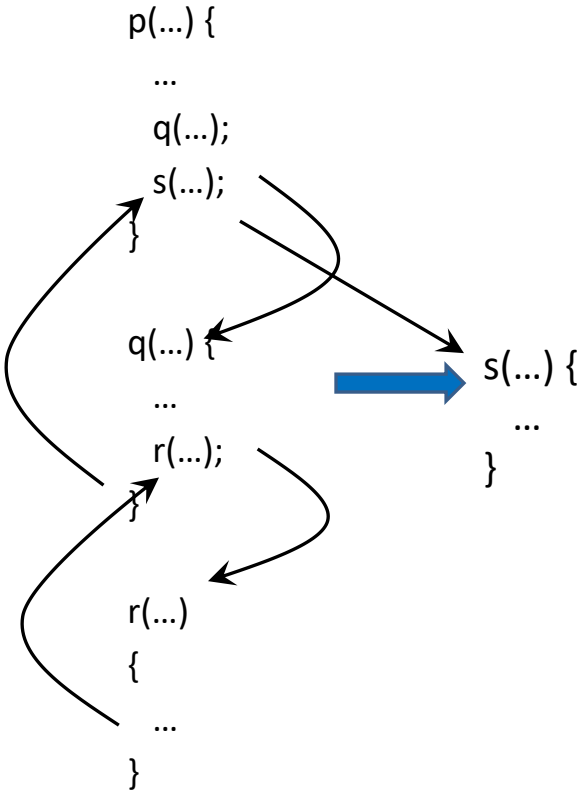
```
p(…) {
  …
  q(…);
  s(…);
}

q(…) {
  …
  r(…);
}

r(…)
{
```

```
s(…) {
  …
}
```

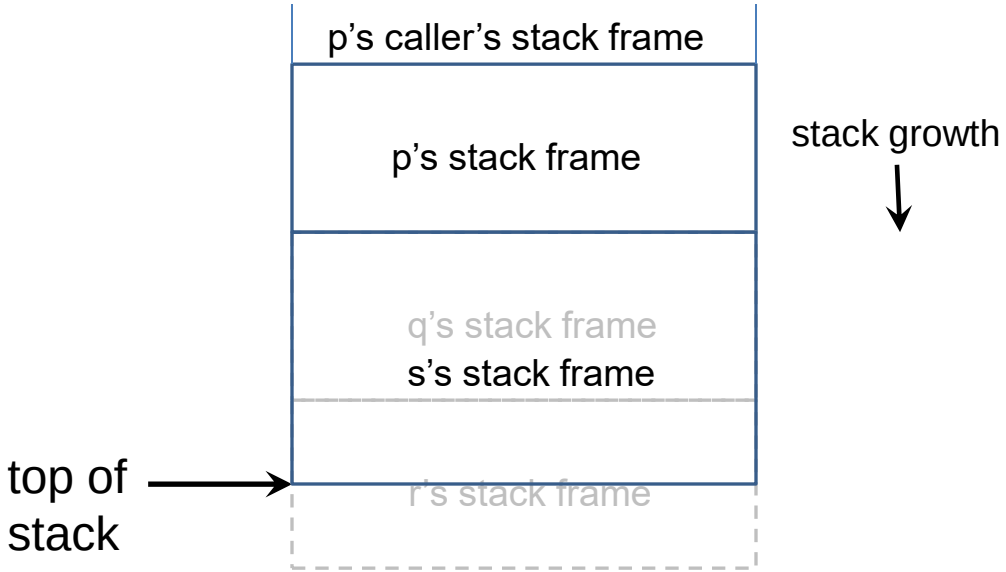
Runtime stack:



Background: Runtime Memory Organization



Runtime stack:



How is recursion implemented?

- What happens when a function gets called?

```
int A(int w)
{
 return w+w;
}

int B(int x)
{
 int z,y;
 …… // other statements
 z = A(x) + y;

 return z;
}
```

What happens when a function is called? (cont.)

An **activation** record is stored into a stack **(run-time stack)**

1) The computer must stop executing function ***B*** and starts executing function **A**
2) Since it needs to come back to function ***B*** later, it needs to store everything about function **B** that is going to need (**x, y, z**, and the place to start executing upon return)
3) Then, **x** from **B** is bounded to **w** from **A**
4) Control is transferred to function **A**

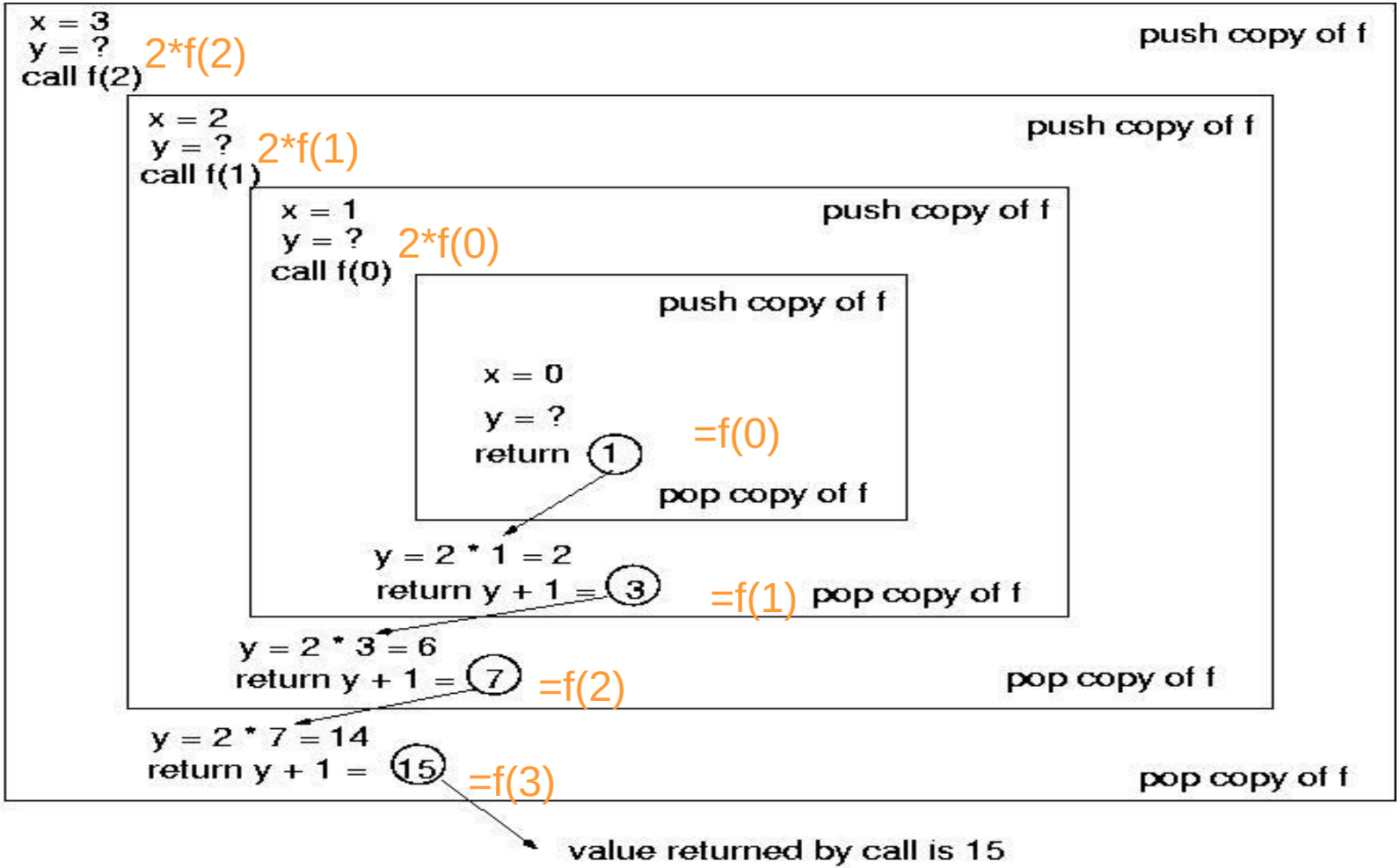
What happens when a function is called? (cont.)

- After function **A** is executed, the activation record is popped out of the run-time stack
- All the old values of the parameters and variables in function **B** are restored and the return value of function **A** replaces **A(x)** in the assignment statement

What happens when a recursive function is called?

- Except the fact that the calling and called functions have the same name, there is really no difference between recursive and nonrecursive calls

```
int f(int x)
{
    int y;

    if(x==0)
                    return 1;
   else {
                     y = 2 * f(x-1);
                    return y+1;
   }
}
```


x = 3
y = ?
call f(2)
2*f(2)
push copy of f
x = 2
y = ?
call f(1)
2*f(1)
push copy of f
x = 1
y = ?
call f(0)
2*f(0)
push copy of f
x = 0
y = ?
return 1
=f(0)
pop copy of f
y = 2 * 1 = 2
return y + 1 = 3
=f(1)
pop copy of f
y = 2 * 3 = 6
return y + 1 = 7
=f(2)
pop copy of f
y = 2 * 7 = 14
return y + 1 = 15
=f(3)
pop copy of f
value returned by call is 15

Deciding whether to use a recursive solution

- When the depth of recursive calls is relatively "shallow"
- The recursive version does about the same amount of work as the nonrecursive version
- The recursive version is shorter and simpler than the nonrecursive solution

Slides and figures have been collected from various publicly available Internet sources for preparing the lecture slides of IT2001 course. I acknowledge and thank all the original authors for their contribution to prepare the content