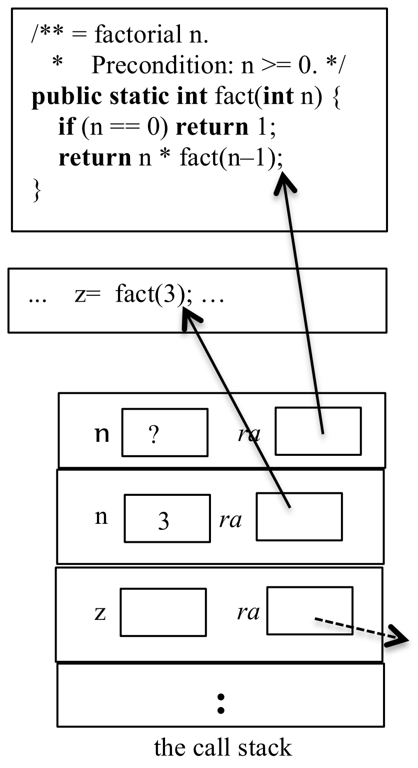
# Now we see why recursion works

David Gries and Scott Wehrwein

We execute a call on the recursive factorial function.

Assume that a method body contains the assignment z= fact(3); and a frame for a call on this method is at the top of the call stack.

We carry out function call fact(3).



```
/** = factorial n.
  *   Precondition: n >= 0. */
public static int fact(int n) {
   if (n == 0) return 1;
   return n * fact(n–1);
}
```

```
...   z=  fact(3); …
```

**Algorithm**

1. **Push a frame for the call onto the call stack**.
2. **Assign the values of the arguments of the call to the parameters**.
   The value of the argument is 3, so we store 3 in n.
3. **Execute the method body, using the frame for the call to access parameters and local variables**.
   n is 3, so execution of the if-statement does nothing.
   Evaluating the return-expression requires evaluating the function call fact(n-1), so we push a frame for the call onto the call stack.

**Now you can see why recursion works!** Each call has its own space — on the call stack— for its parameters, local variables, and return address. It doesn't matter whether a call is recursive or not, it has its own space, and its method body is executed independently of all other calls.

If you are still uneasy with executing recursive calls, we encourage you to execute the call fact(3) to completion yourself, following the algorithm for executing a method call slowly and carefully. The best way to learn and understand is to *do*.

4. **Pop the frame for the call from the call stack. If this is a function call** (and it is)**, push the value to be returned onto the call stack.**