On the surface it may seem as though local and nonlocal objects of static storage duration are effectively interchangeable, but clearly they are not. Even when clients cannot directly access the nonlocal object due to giving it **internal linkage** by marking it **`static`** or putting it in an **unnamed namespace**, ~~the initialization behaviors make~~ such objects behave differently.

Dangerous recursive initialization

As with all other initialization, control flow does not continue *past* the definition of a **`static`** local object until after the initialization is complete, making recursive **`static`** initialization — or any initializer that might eventually call back to the same function — dangerous:

```
int fz(int i)  // The behavior of the first call is undefined unless i is 0.
{
    static int dz = i ? fz(i - 1) : 0;  // Initialize recursively. (BAD IDEA)
    return dz;
}

int main()  // The program is ill formed.
{
    int x = fz(5);  // Bug, e.g., due to possible deadlock
}
```

In the example above, the second recursive call of `fz` to initialize `dz` has undefined behavior because the control flow reached the same definition again before the initialization of the **`static`** object was completed; hence, control flow cannot continue to the **`return`** statement in `fz`. Given a likely implementation with a nonrecursive mutex or similar lock, the program can potentially deadlock, though many implementations provide better diagnostics with an exception or assertion violation when this form of error is encountered.[6]

Subtleties with recursion

Even when not recursing within the initializer itself, the rule for the initialization of **`static`** objects at function scope becomes more subtle for self-recursive functions. Notably, the initialization happens based on when flow of control first passes the variable definition and *not* based on the first invocation of the containing function. Due to this, when a recursive call happens in relation to the definition of a **`static`** local variable impacts which values might be used for the initialization:

[6] Prior to standardization (see **ellis90**, section 6.7, "Declaration Statement," pp. 91–92), C++ allowed control to flow past a **`static`** function-scope variable even during a recursive call made as part of the initialization of that variable. This behavior would result in the rest of such a function executing with a zero-initialized and possibly partially constructed local object. Even modern compilers, such as GCC with `-fno-threadsafe-statics`, allow turning off the locking and protection from concurrent initialization and retaining some of the pre-C++98 behavior. This optional behavior is, however, dangerous and unsupported in any standard version of C++.