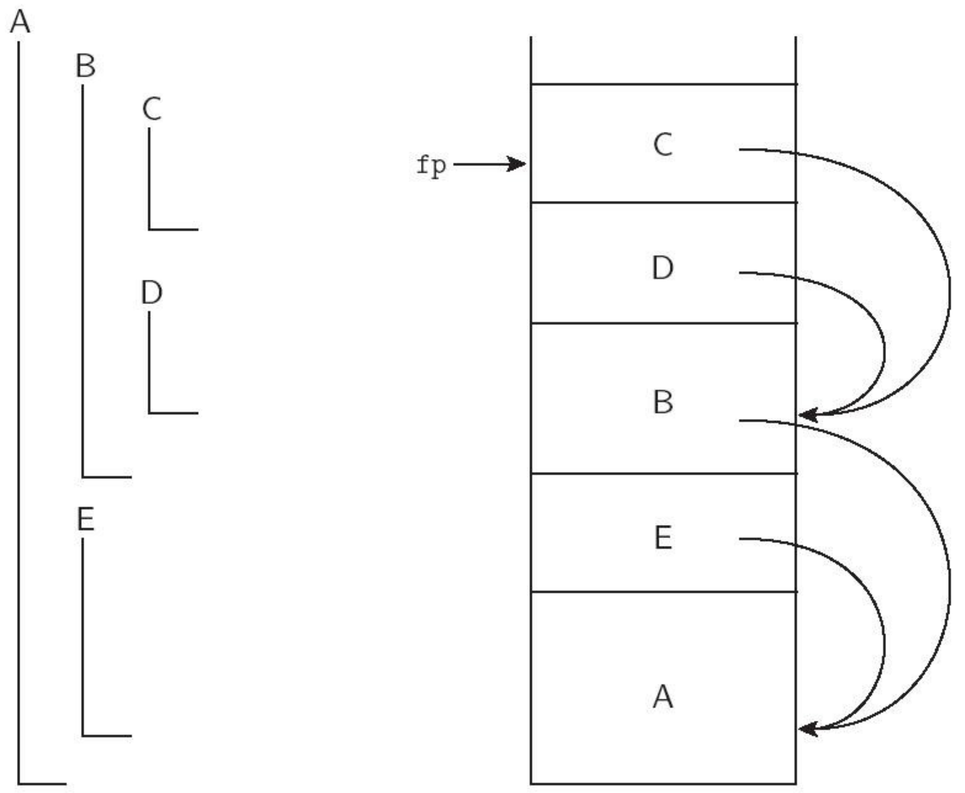
Scope Rules

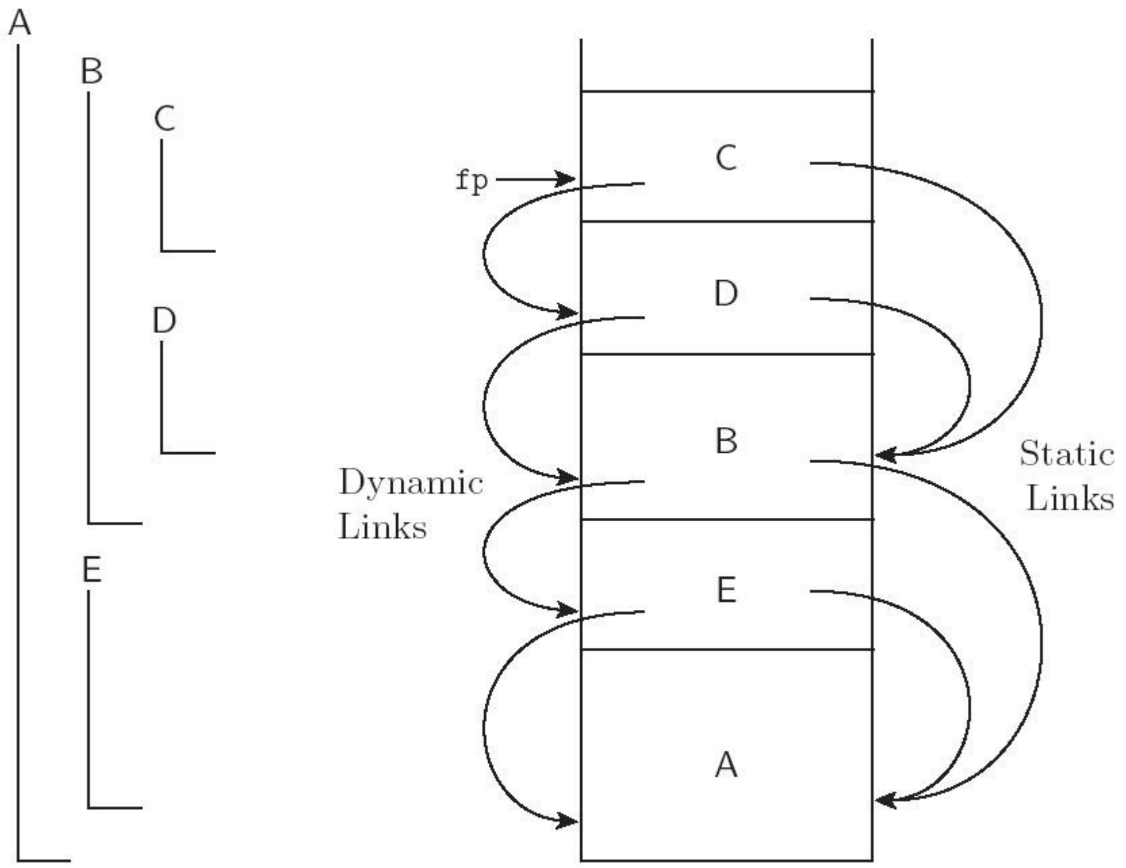


Python code outline

```
def A:
    def B:
        def C:
            ... # C stuff
        def D:
            ... # D stuff
        ... # B stuff

    def E:
        ... #E stuff

    ...#A stuff
```

Figure 3.5: **Static chains.** Subroutines A, B, C, D, and E are nested as shown on the left. If the sequence of nested calls at run time is A, E, B, D, and C, then the static links in the stack will look as shown on the right. The code for subroutine C can find local objects at known offsets from the frame pointer. It can find local objects of the surrounding scope, B, by dereferencing its static chain once and then applying an offset. It can find local objects in B's surrounding scope, A, by dereferencing its static chain twice and then applying an offset.

Review Of Stack Layout



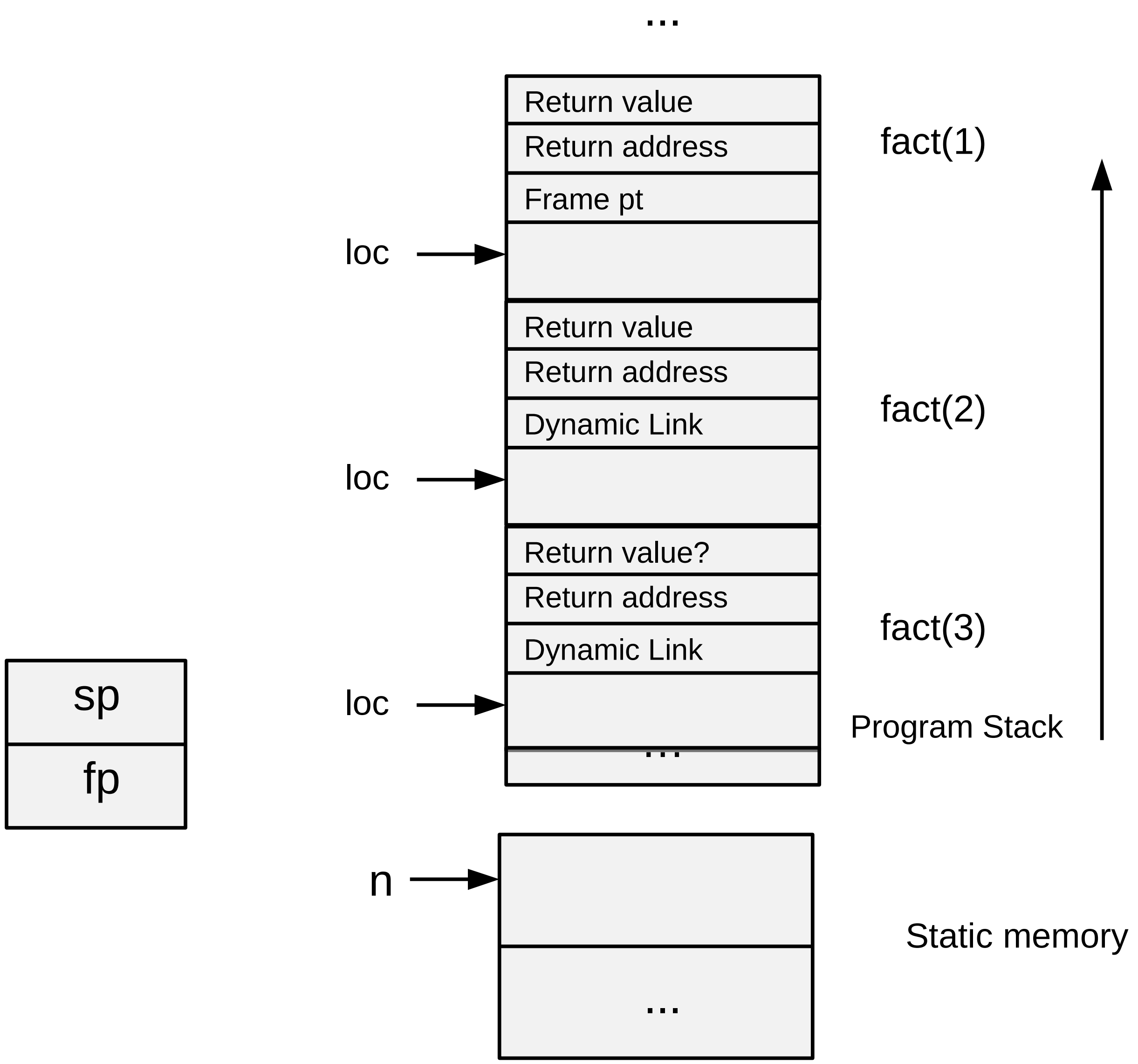
Python code outline

```
def A:
    def B:
        def C:
            ... # C stuff
        def D:
            ... # D stuff
        ... # B stuff
    def E:
        ... #E stuff

    ...#A stuff
```

Figure 8.1: **Example of subroutine nesting, taken from Figure 3.5.** Within B, C, and D, all five routines are visible. Within A and E, routines A, B, and E are visible, but C and D are not. Given the calling sequence A, E, B, D, C, in that order, frames will be allocated on the stack as shown at right, with the indicated static and dynamic links.

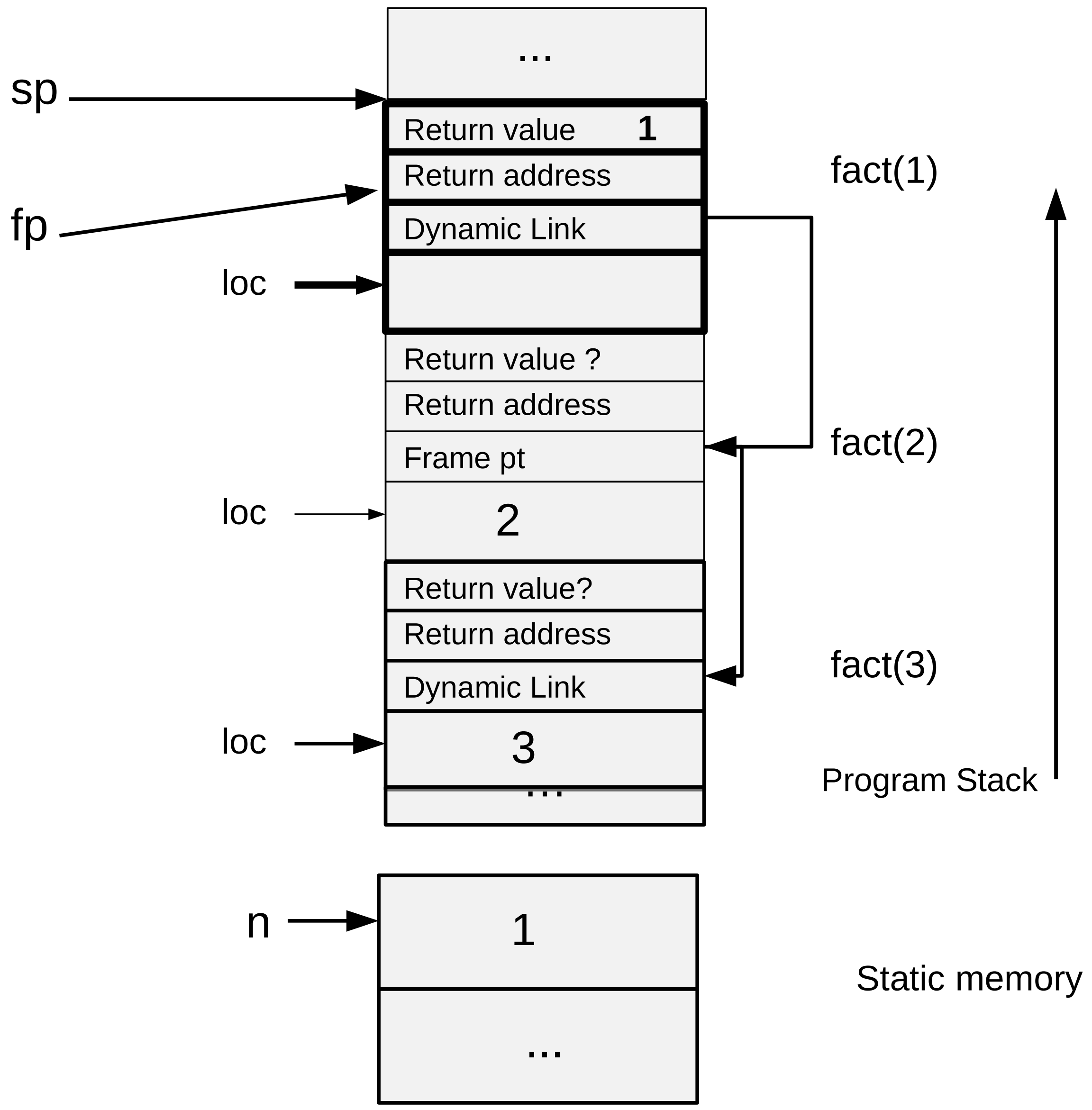
Simple recursive function call

```
int n;
int fact(void) {
    int loc;
    if (n>1) {
       loc = n--;
       return loc *
fact();
   }  else  {
      return 1;
   }
}

int n = 3;
…
  fact();
```



Simple recursive function call

```
int n;
int fact(void) {
    int loc;
    if (n>1) {
       loc = n--;
       return loc * fact();
   }  else  {
      return 1;
   }
}

int n = 3;
…
  fact();
```


...
sp
Return value 1
Return address
fp
Dynamic Link
loc
fact(1)
Return value ?
Return address
Frame pt
fact(2)
loc
2
Return value?
Return address
Dynamic Link
fact(3)
loc
3
...
Program Stack
n
1
...
Static memory