Compilers Course
Lecture 11: Recursion and Call Stacks

When a procedure executes:

- The program counter register (PC) points to the current instruction
- The state contains:
  - Its input parameters
  - Its local variables
  - A return address referring to its caller's code
  - A "dynamic link" referring to its caller's state

The concrete representation of this state is called an "activation record".

Call Tree

```
main() { f(2); g(); }
f(n) { if (n <= 1) h(); else f(n-1); }
g() { h(); }
h() { }
```

$main()_1$

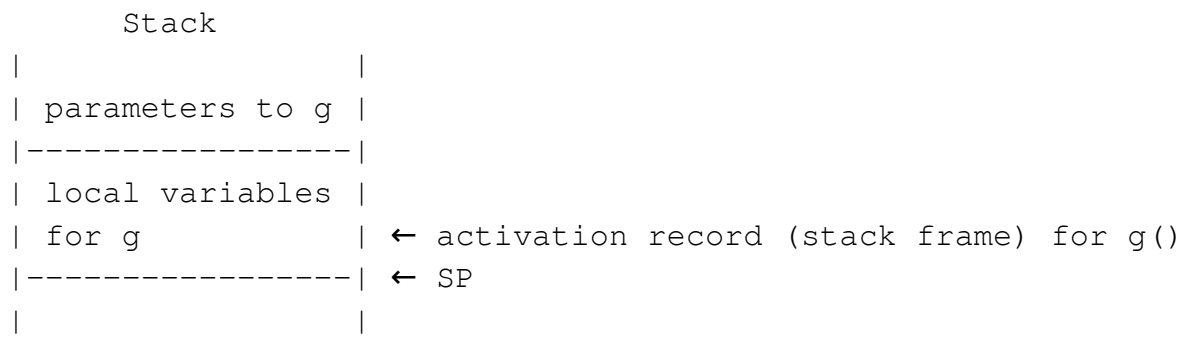
- $f(2)_2$
  - $f(1)_3$
    - $h()_4$
- $g()_5$
  - $h()_6$

Calling a procedure involves creating a new activation record and switching to it. Returning from a procedure involves leaving the current activation record, restoring the activation record via the dynamic link, and restoring the caller's PC via the return address.

Call Stacks

Most languages specify that when a procedure returns its activation record ceases to exist. The live part of the call tree thus forms a stack, so activation records are typically implemented using a variable-sized memory area (the stack) and a pointer to its current end (the stack pointer, SP).

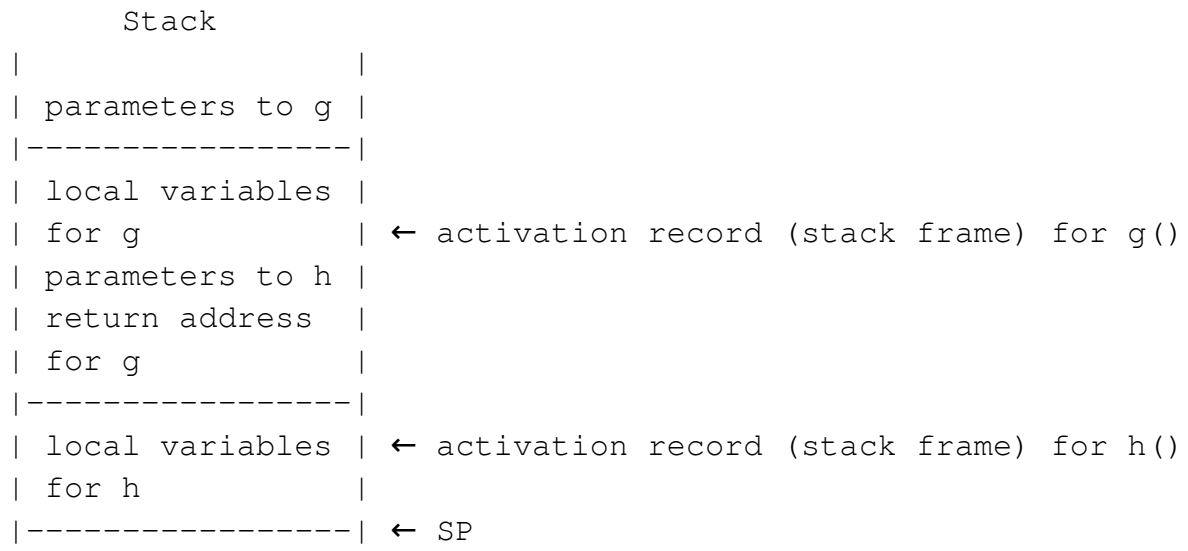
Before g() calls h():



When h() is called it creates its own activation record:

h:
SP := SP - H_FRAME_SIZE (prologue)
...

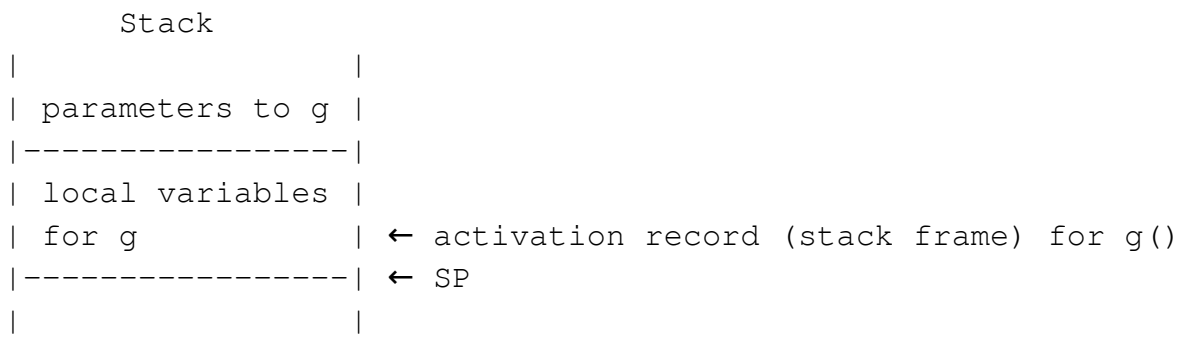
which results in:



When h() returns, it removes its activation record and reinstates g()'s activation record:

SP := SP + H_FRAME_SIZE (epilogue)
return

which results in:



and g() can then resume its execution.

+ Handles recursion easily
+ Very cheap allocation and deallocation of activation records
+ Easy access to local variables at the top of your own activation record (SP + FRAME_SIZE - offset)
+ Easy access to actual parameters at the bottom of the caller's activation record (SP + FRAME_SIZE + offset)

Bad Implementation Alternatives

- Store a procedure's activation record in global variables.
  - Simple, but cannot handle recursion.
  - Some CPUs supported this in hardware.
  - Used in the 1960s for FORTRAN.
- Use dynamic allocation on the heap for activation records.
  - Handles recursion, but with high runtime costs.
  - Allows activation records to "survive" returns and be reactivated later on. Used in some languages to implement threads and exceptions (Scheme, Smalltalk).