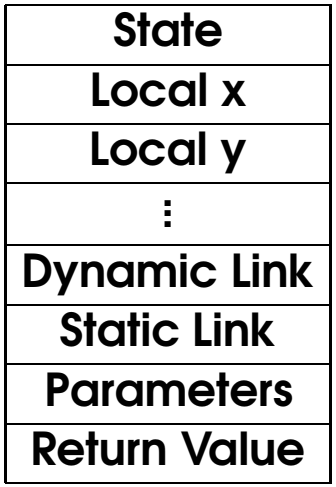
Runtime storage management

Most block-structured languages require manipulation of runtime structures to maintain efficient access to appropriate data and machine resources. For our purposes, a procedure is either a named function or an inline (parameterless) block.

Let's examine the activity normally associated with invoking a procedure P:

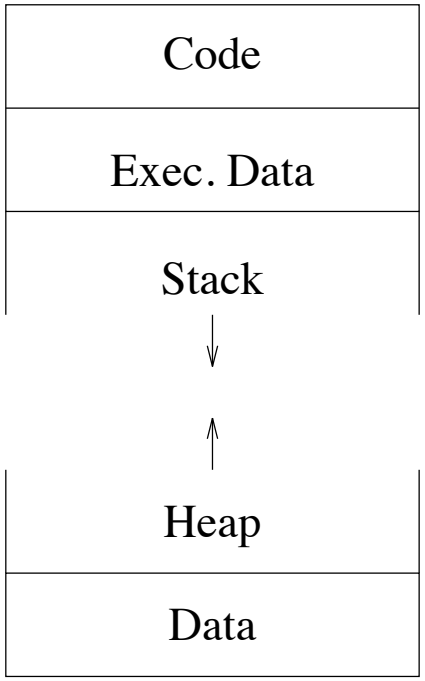
1. **Some machine state might be saved: general registers, vector registers, condition codes, interrupt masks, etc.**
2. **Access must be established to P's local variables and compiler-generated temporaries.**
3. **Access must be established to outer scope variables (but not for C).**
4. **The caller of P must be recorded so that P can return when done.**
5. **Parameters might be received prior to executing P.**
6. **A return value might be prepared prior to returning from P.**

Each procedure invocation causes creation of an *activation record* or *frame* to hold such runtime information.



It's convenient to have each local occupy a fixed amount of storage in the frame. Therefore, arrays and other large objects are often indirectly accessed from a procedure's frame, with the actual storage allocated on stack after the frame.

A simple runtime storage layout



Since there are two dynamically growing areas, a simple scheme is to place these at opposite ends of the address space.

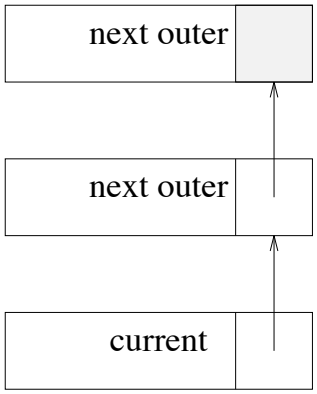
Following the contour of procedure entry and exit, activation records are usually allocated on a stack. Where languages allow suspension and resumption of procedures (*e.g.*, via *continuations*), then frames are garbage-collected from the heap when dead.

The stack can also be used for performing intermediate computations.

The heap is generally managed by some form of explicit or implicit garbage collection (4).

Access to nonlocals

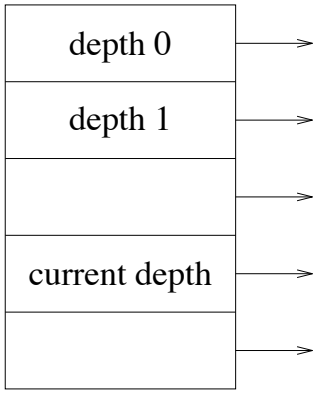
Static Links



At procedure entry, a link is inserted in the frame to a procedure's next outer scope, whose frame is linked to its outer scope, and so on. The static link is deallocated along with the frame.

Establishing the link is fast, but accessing the kth enclosing scope requires k indirections using static links. However, a good register allocator would cache these in the procedure's registers.

Displays



A display is an array of frame pointers. At procedure entry, the display is adjusted so that the frame at static depth d is accessed via entry d of the display. The display must be reset at procedure return.

Maintaining the display takes more time than with static links, but access to outer scopes is faster once the display is established.

Most programs make almost exclusive use of local and outermost scopes, with scant use of intermediate scopes. This is especially true in C, where the language offers no access to intermediate scopes except by explicit pointers.