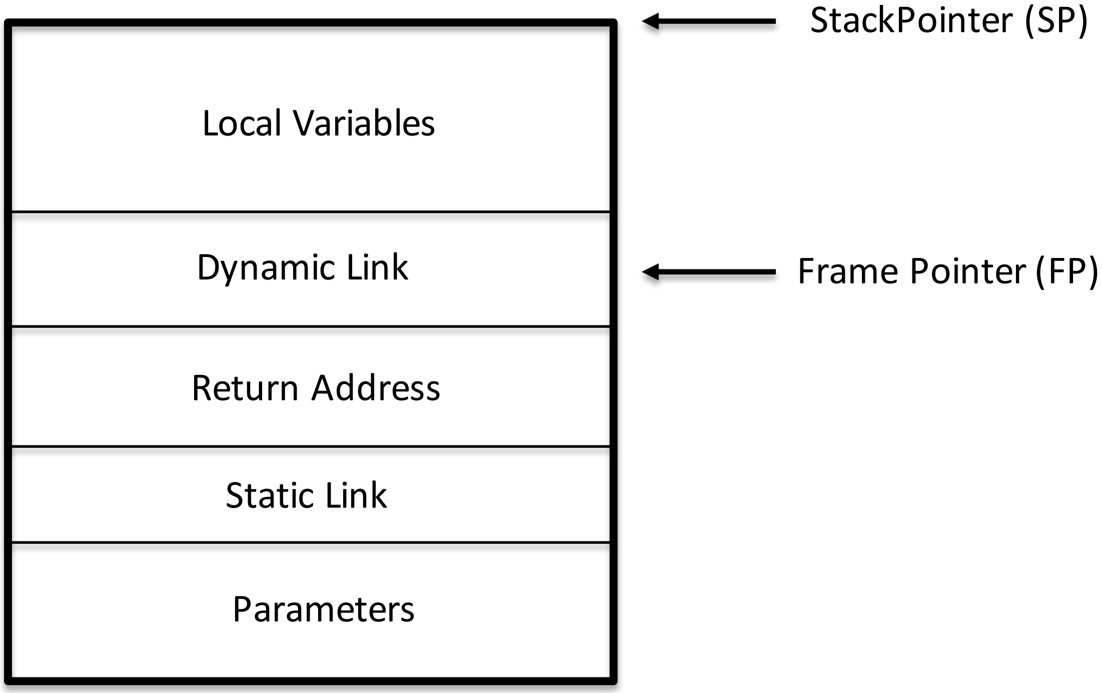
The illustration to the right shows the stack frame (aka activation record) for a single procedure. A stack frame is pushed onto the stack every time a procedure is called and is popped off the stack when the procedure returns. The stack frame for the currently executing procedure is at the top of the stack and is pointed to by the Frame Pointer (FP) register.

Return Address: Points to the place in the code of the calling procedure to return to.

Static Link:
Points into the stack frame of the defining procedure. A static link is used to support static scoping in a language that allows nested procedures.

Dynamic Link:
Points into the stack frame of the calling procedure (i.e. the old value of the FP when the calling procedure was executing).



StackPointer (SP)

Local Variables

Dynamic Link

Frame Pointer (FP)

Return Address

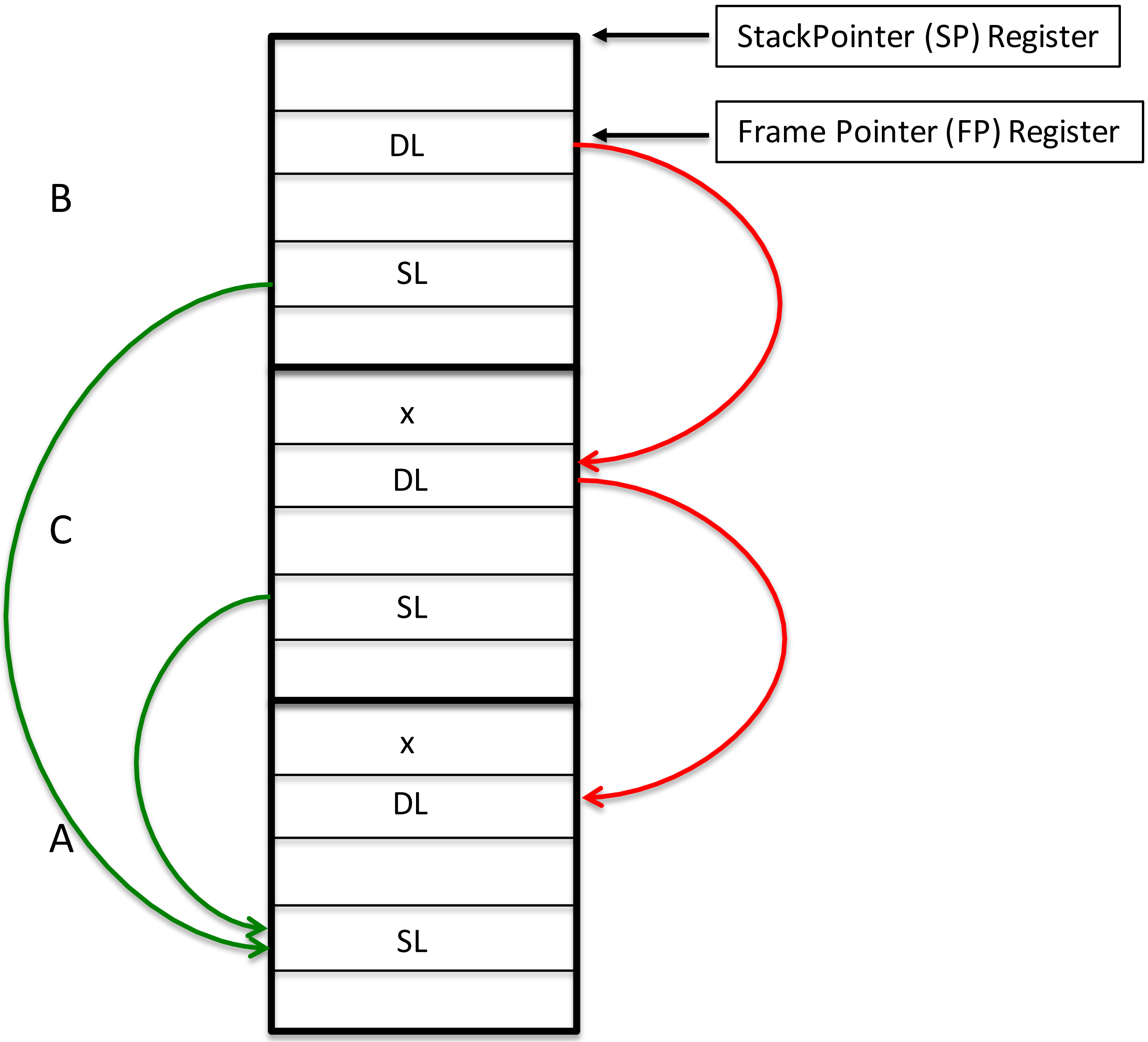
Static Link

Parameters

Notes

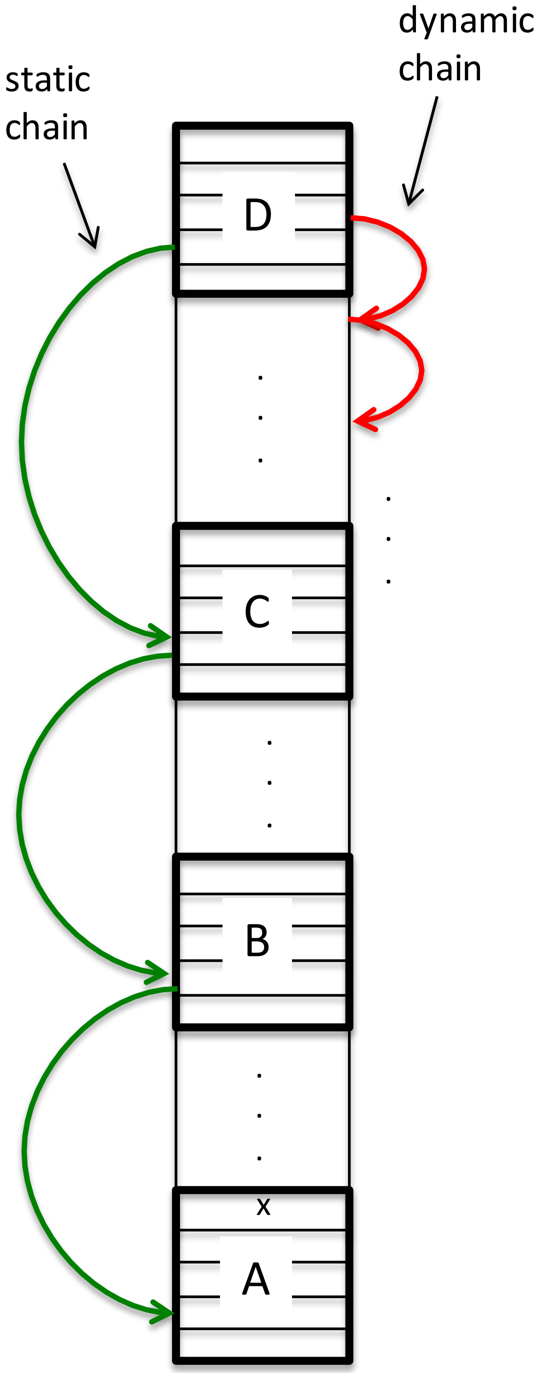
- For languages that do not support nested procedures (e.g. C), no static link is needed.
- Parameters and local veriables are accessed using offsets from the frame pointer, e.g. [FP+16].
- If a procedure is recursive, there may be many stack frames on the stack for that procedure, one stack frame for each call to that procedure that has not yet returned.
- On Intel x86 processors, the stack grows down in memory. That is, the push instruction decrements the address contained in the SP.

```
procedure A()
    x: integer;

    procedure B()
    begin
        x: = 1;
     end;

    procedure C()
        x: integer;
    begin
        B();
     end;

begin (* A *)
  C();
end;
```



Since B is defined within A, static scoping requires that the x referenced by B is the x declared in A. B must follow its static link one hop to get to A's stack frame. Thus, to access x, B will follow its static link one hop, then use an offset to get the location of x.

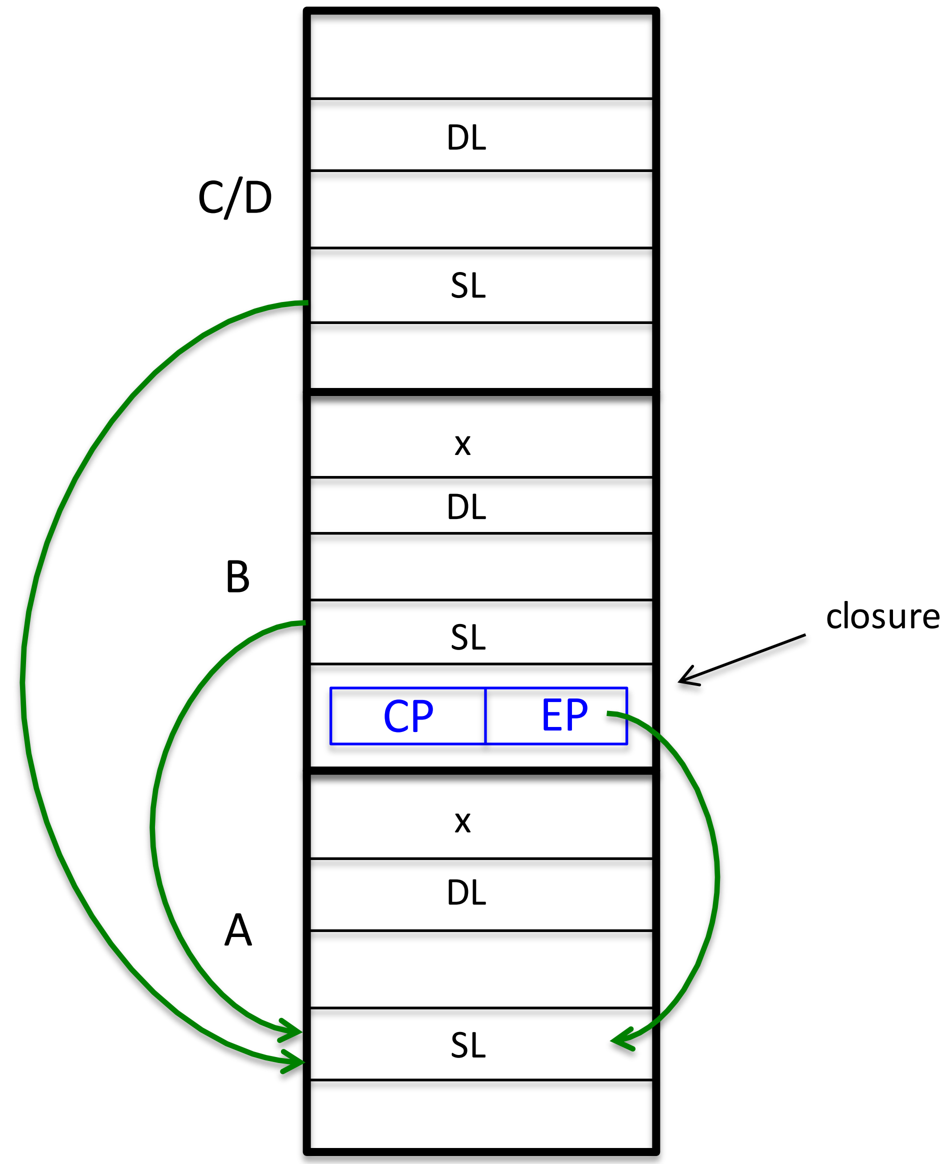
```
procedure A()
    x: integer;
    procedure B()
       procedure C()
          procedure D()
          begin (* D *)
             x: = 1;
          end;
       begin (* C *)
        ...
       end;
    begin (* B *)
      ...
    end;
begin (* A *)
...
end;
```



Notes

- On the stack, there may be stack frames for many other procedures in between the stack frames for A, B, C and D, which is why the dynamic chain (sequence of dynamic links) cannot be used to access the variable x in A.
- The code that the compiler generates for D follows D’s static chain (sequence of static links) for three hops to get to A’s stack frame, and then uses an offset to access x.
- The number of hops (3 n this case) is known at compile time and is the difference between the nesting level at which the variable is defined (nesting level 1, in this case) and the nesting level at which the variables is accessed (nesting level 4, in this case). No searching for variables in the stack frames along the static chain is required.

```
procedure A()
    x: integer;

    procedure B(procedure D)
        x: integer;
    begin
        D();
     end;

    procedure C()
    begin
        x := 1;
     end;

begin (* A *)
   B(C);
end;
```



- When A calls B, it passes a <u>closure</u> to represent the procedure C being passed as the parameter. A closure contains a code pointer (CP), pointing to the code for the procedure being passed, and an environment pointer (EP), which is the static link to be used when the passed procedure is called.
- When B calls D (which is really C), B copies the static link from the EP portion of the closure and writes it to D's stack frame. D is executed by jumping to the code pointed to by the CP portion of the closure.