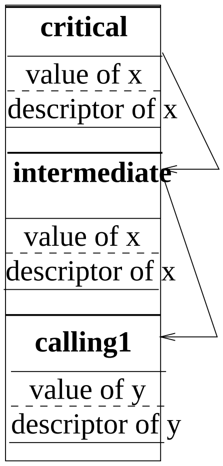
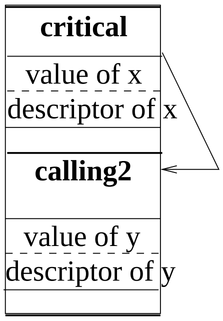
Consider a program in a language which supports dynamic scoping:

```
Procedure critical(x: integer);
begin
   if x == 3 then y := 5 else y[3] := 5
end
procedure intermediate(x: integer)
begin
   critical(x)
end
procedure calling1
var y: array [1..10] of integer;
begin
   intermediate(4)
end
procedure calling2
var y: integer;
begin
   critical(3)
end
```

First observe that it is a well typed program. Each `y` declared has a type which is statically determinable. However, the use occurrences of `y` in `critical` will be bound to different declarations at runtime, and the compiler has to generate code to verify that (for different execution paths) the usages of `y` are consistent with the declarations they are bound to at that moment.
The activation stack of the two call sequences `calling1 -> intermediate -> critical` and `calling2 -> critical` will look as in figures (a) and (b) respectively.



(a)



(b)

In the first case, a reference to `y` in `critical` will first be looked up in the activation record of `critical`. To do this, the activation record should also contain other information about a variable than just its value; the place for this information is called the *descriptor* of the variable. Thus one of the things that the descriptor will have to contain is the name of the variable. Since the variable `y` will not be found in the descriptor of `critical`, following the dynamic link, it will be searched for in `intermediate`. It will be finally found in `calling1` whence the statement `y[3] := 5` will be type checked. Thus the descriptor of `y`, should also contain its type. Similarly, in the second case the descriptor of `y` in `calling2` will be use to type check `y = 5`.
This example should give you some idea about how much information has to be shifted from the symboltable to activation records in case of languages with dynamic scoping.