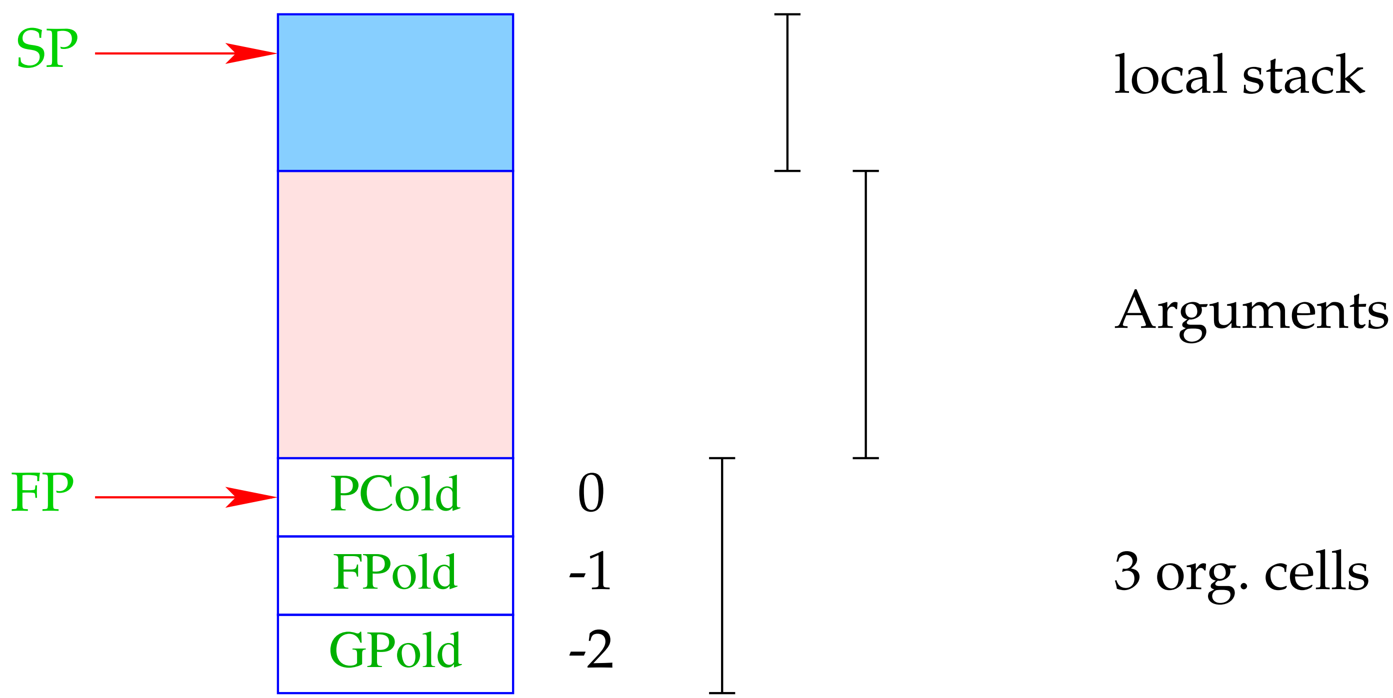
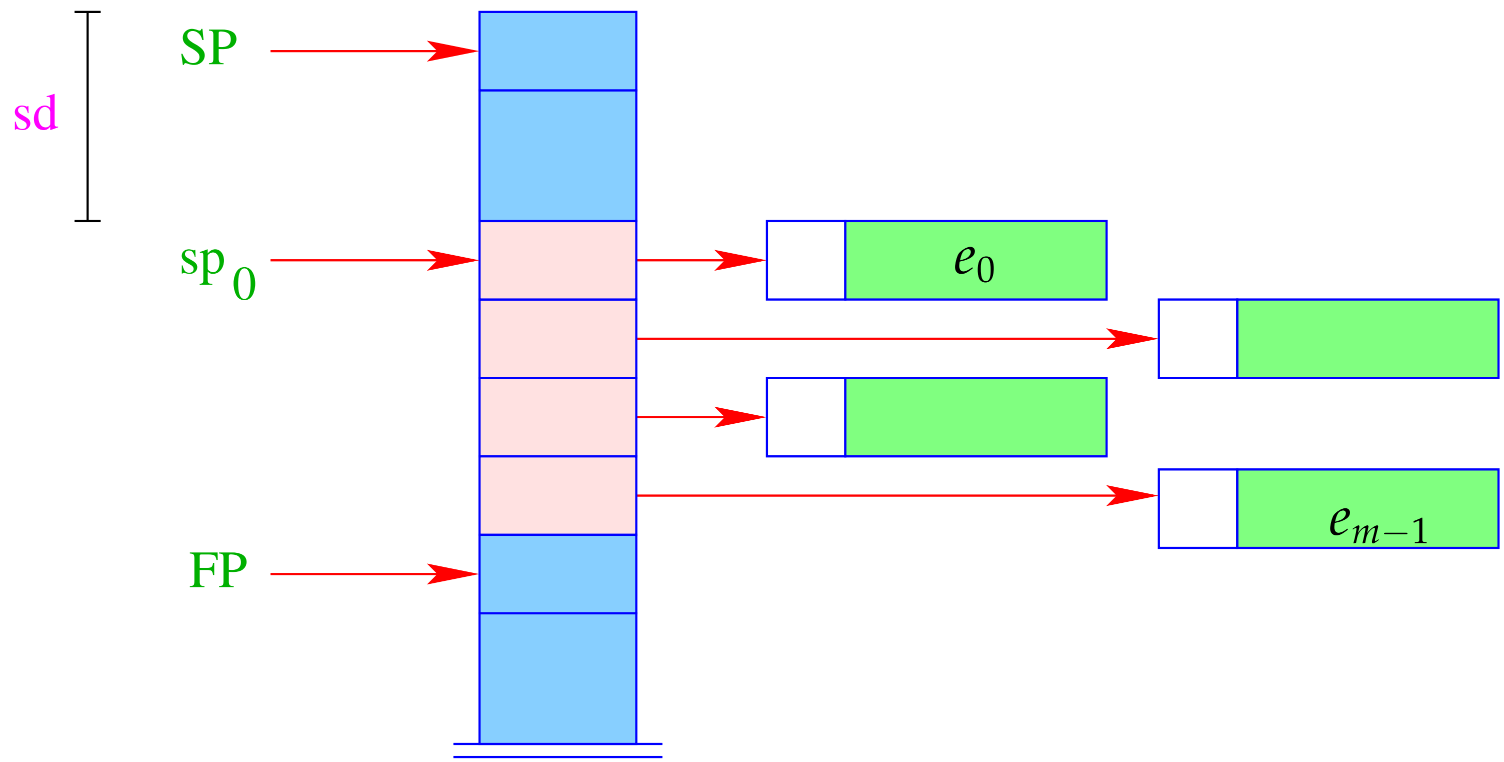
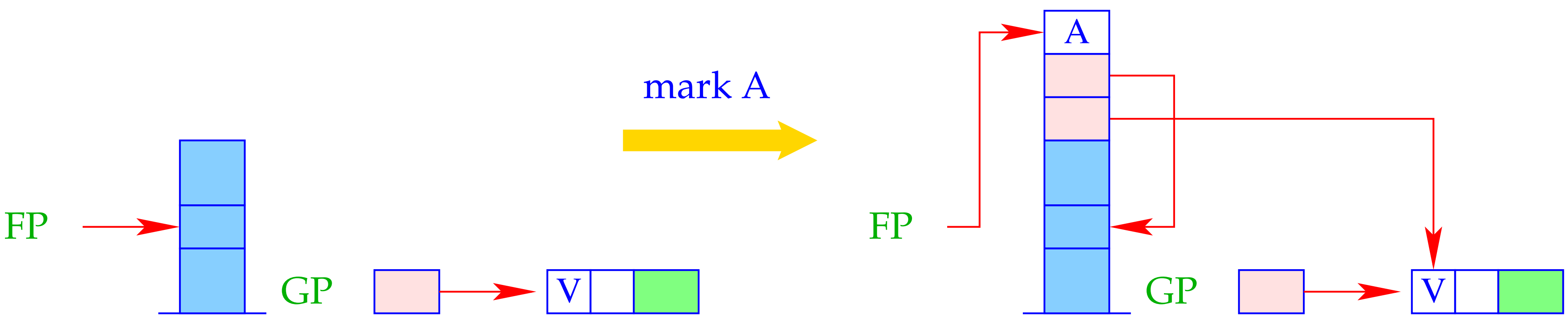
For the implementation of the new instruction, we must fix the organization of a stack frame:



Remember: Addressing of arguments and local variables

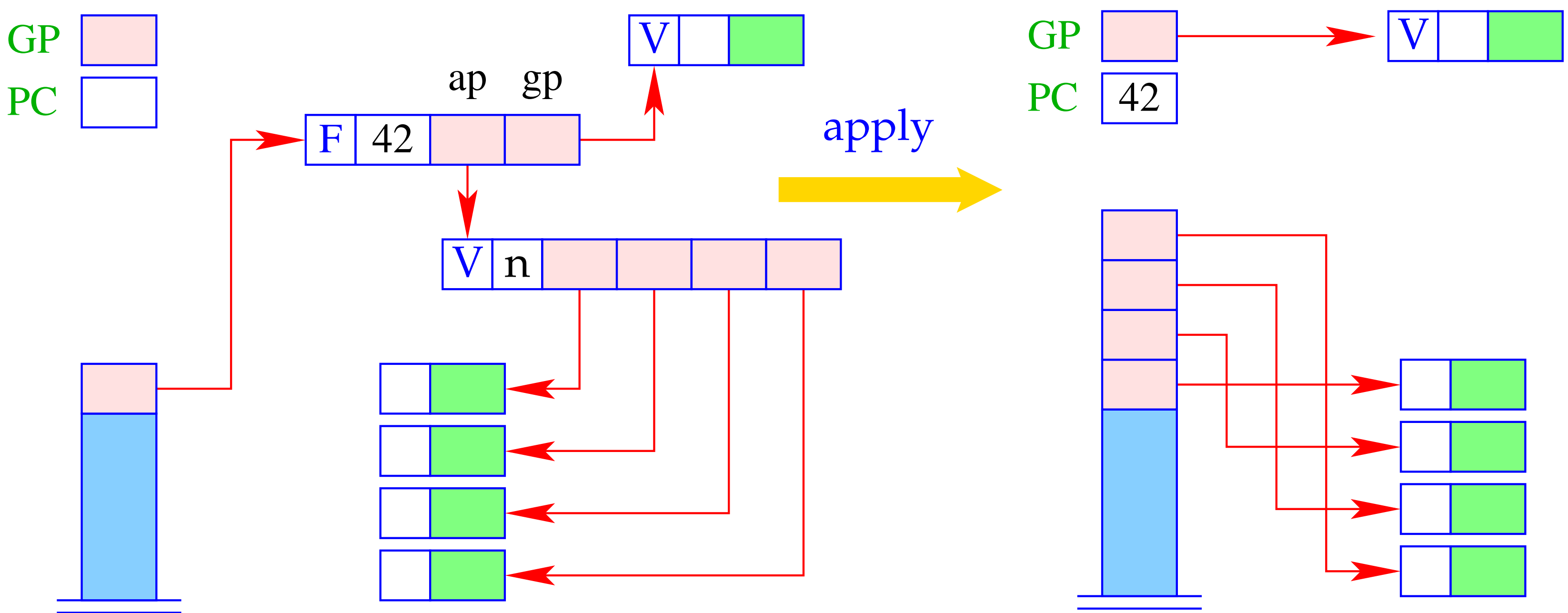


Different from the CMa, the instruction mark A already saves the return address:



S[SP+1] = GP;
S[SP+2] = FP;
S[SP+3] = A;
FP = SP = SP + 3;

The instruction apply unpacks the F-object, a reference to which (hopefully) resides on top of the stack, and continues execution at the address given there:




```
h = S[SP];
if (H[h] != (F,_,_))
    Error "no fun";
else {
```

```
GP = h→gp; PC = h→cp;
for (i=0; i< h→ap→n; i++)
    S[SP+i] = h→ap→v[i];
SP = SP + h→ap→n – 1;
}
```

Warning:

- The last element of the argument vector is the last to be put onto the stack. This must be the first argument reference.
- This should be kept in mind, when we treat the packing of arguments of an under-supplied function application into an F-object !!!

18 Over– and Undersupply of Arguments

The first instruction to be executed when entering a function body, i.e., after an apply is targ k .

This instruction checks whether there are enough arguments to evaluate the body.

Only if this is the case, the execution of the code for the body is started.

Otherwise, i.e. in the case of under-supply, a new F-object is returned.

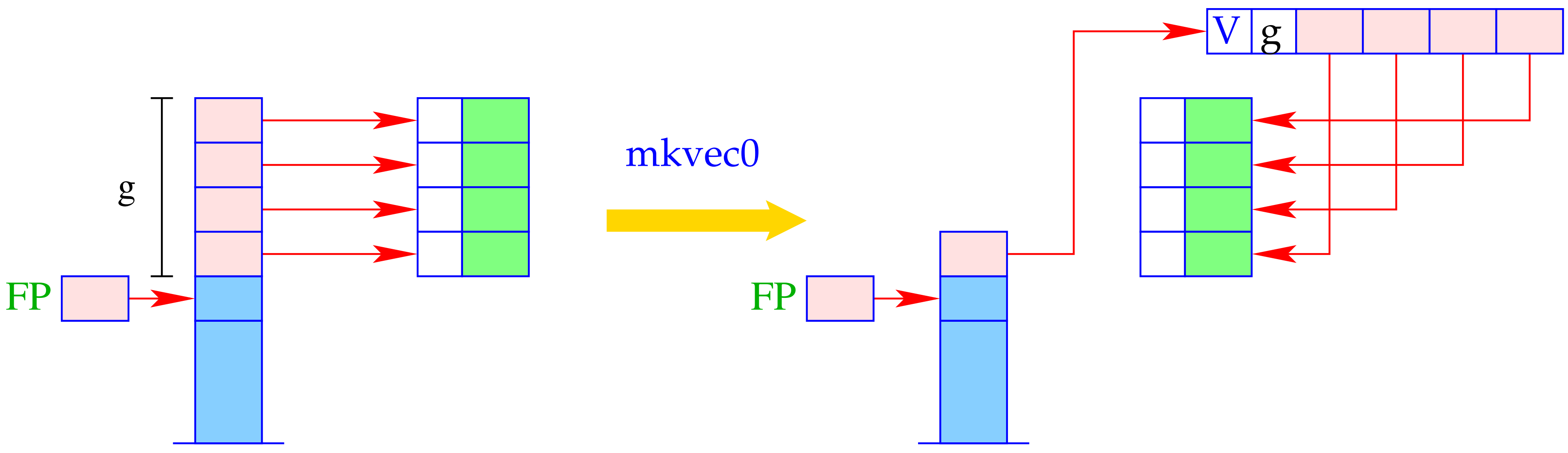
The test for number of arguments uses: SP – FP targ k   is a complex instruction.

We decompose its execution in the case of under-supply into several steps:

$$
\begin{array}{lcll}
\text{targ k} & = & \text{if } (\text{SP} - \text{FP} < \text{k}) \, \{ & \\
 & & \quad \text{mkvec0;} & // \text{ creating the argumentvector} \\
 & & \quad \text{wrap;} & // \text{ wrapping into an F} - \text{object} \\
 & & \quad \text{popenv;} & // \text{ popping the stack frame} \\
 & & \} &
\end{array}
$$

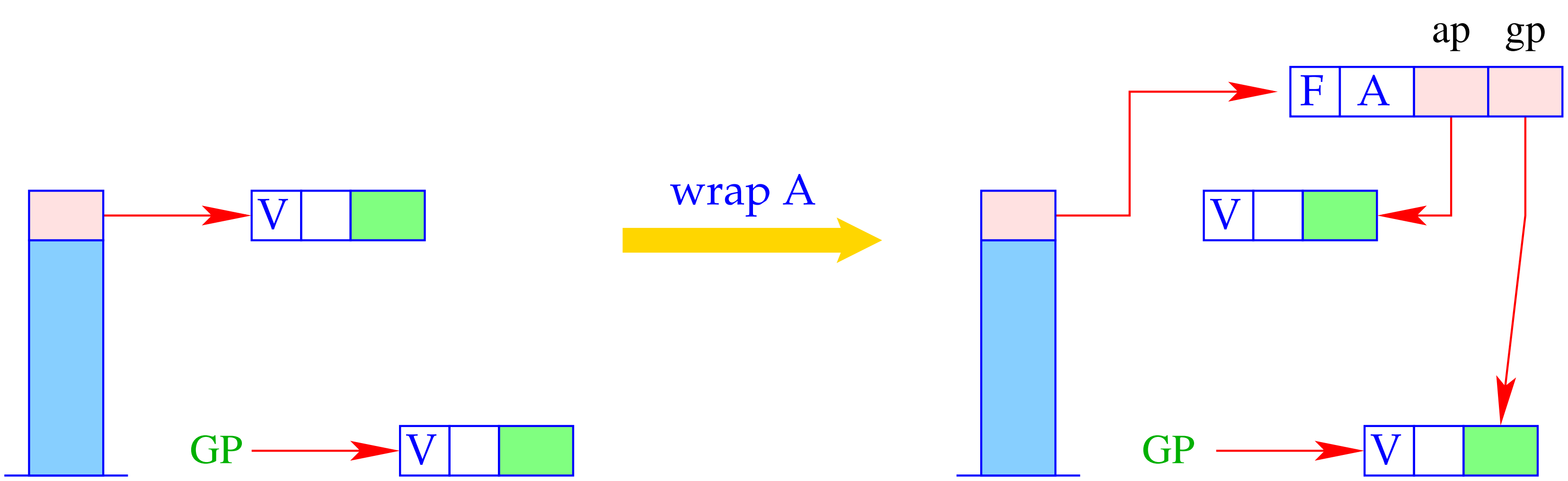
The combination of these steps into one instruction is a kind of optimization   :-)

The instruction mkvec0 takes all references from the stack above FP and stores them into a vector:



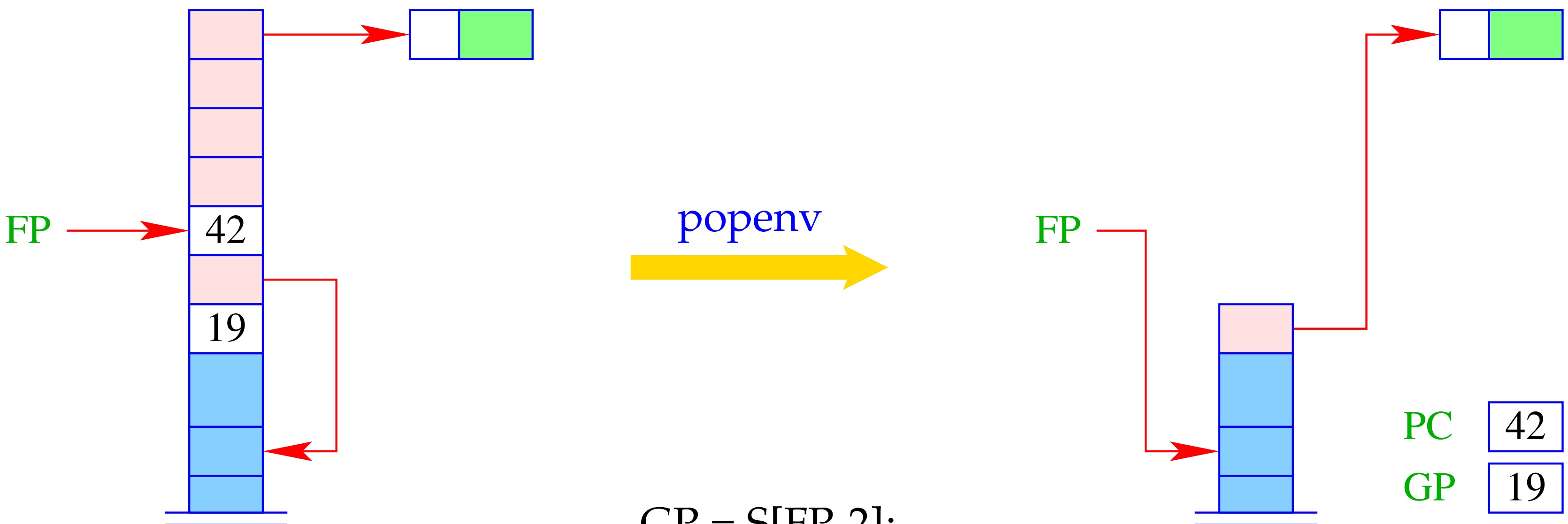
g = SP–FP; h = new (V, g);
SP = FP+1;
for (i=0; i<g; i++)
    h→v[i] = S[SP + i];
S[SP] = h;

The instruction wrap A wraps the argument vector together with the global vector into an F-object:



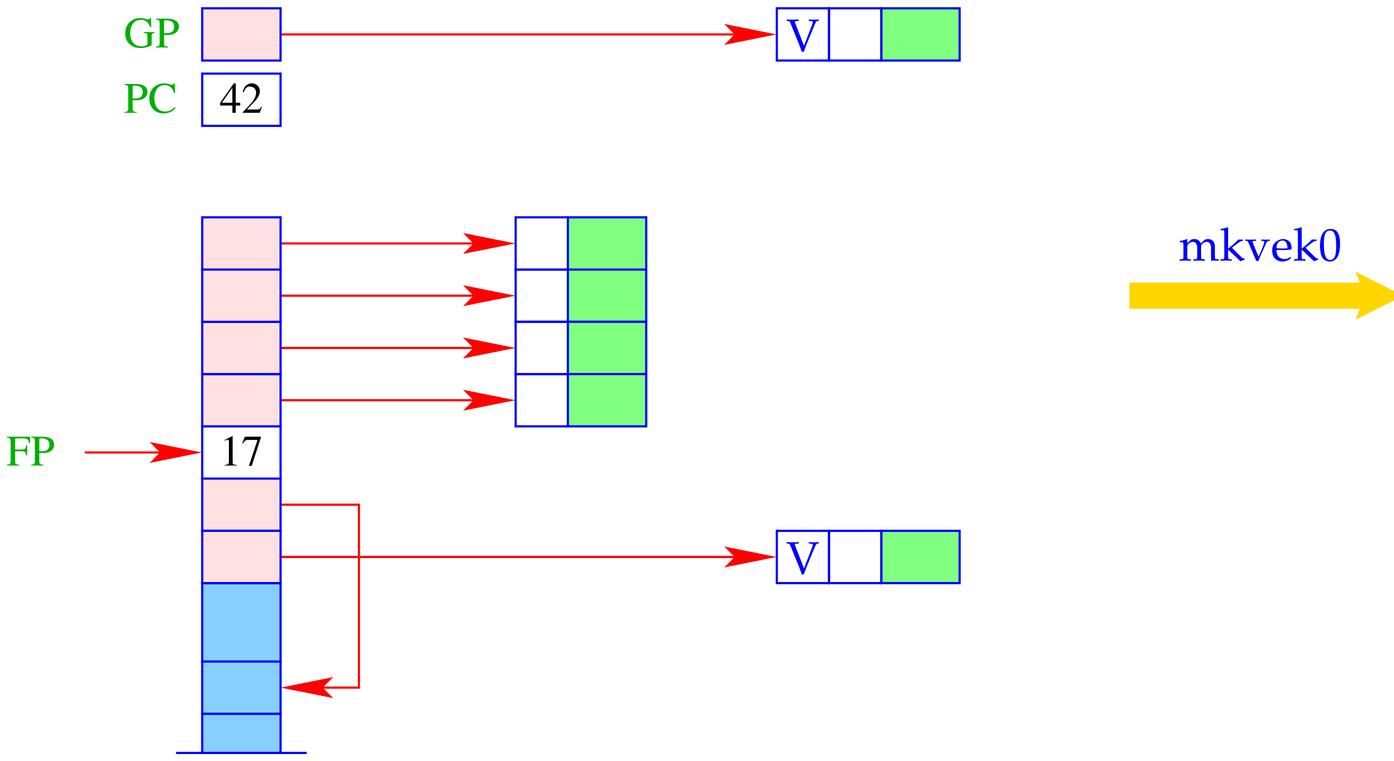
S[SP] = new (F, A, S[SP], GP);

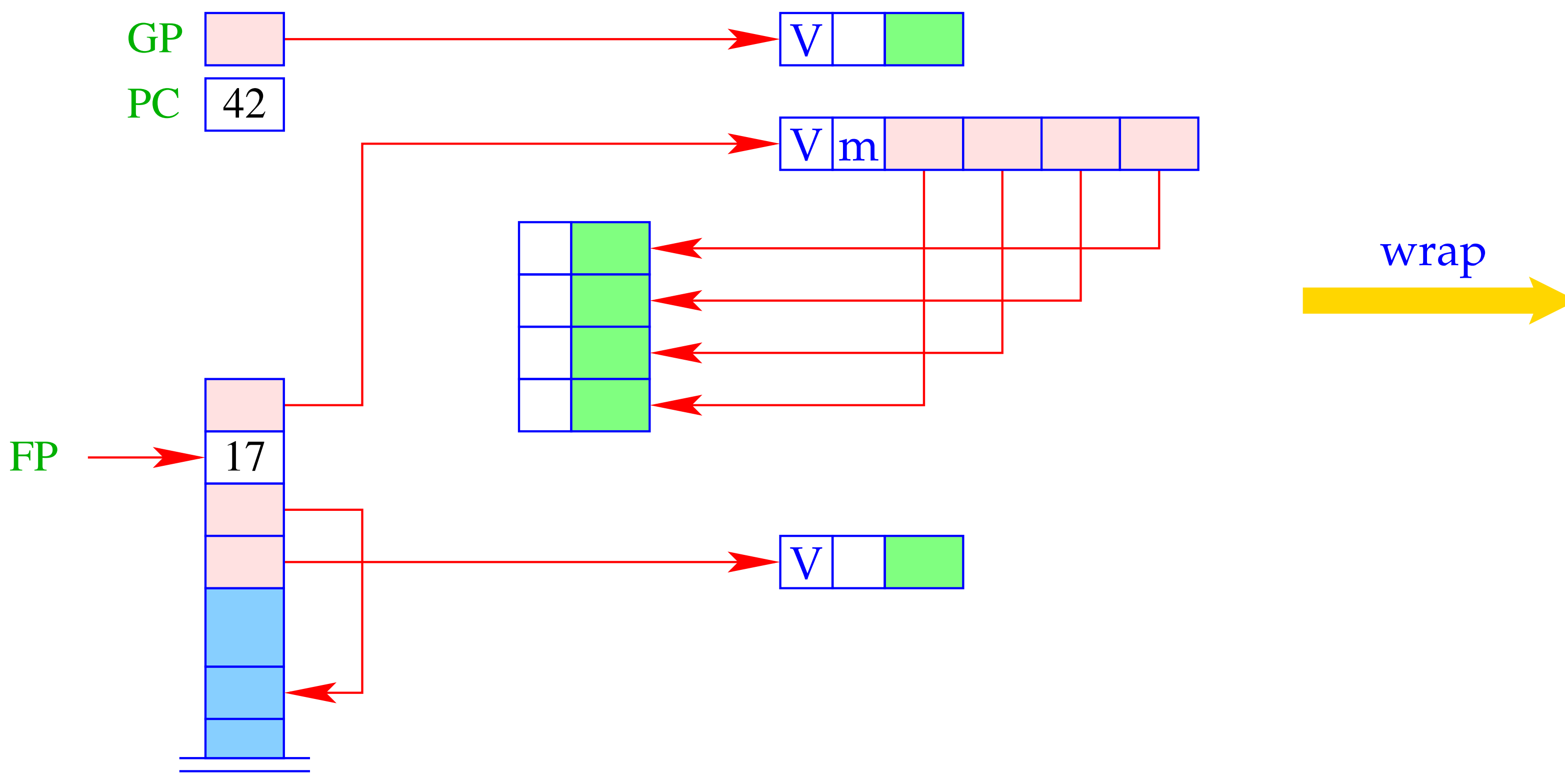
The instruction popenv finally releases the stack frame:

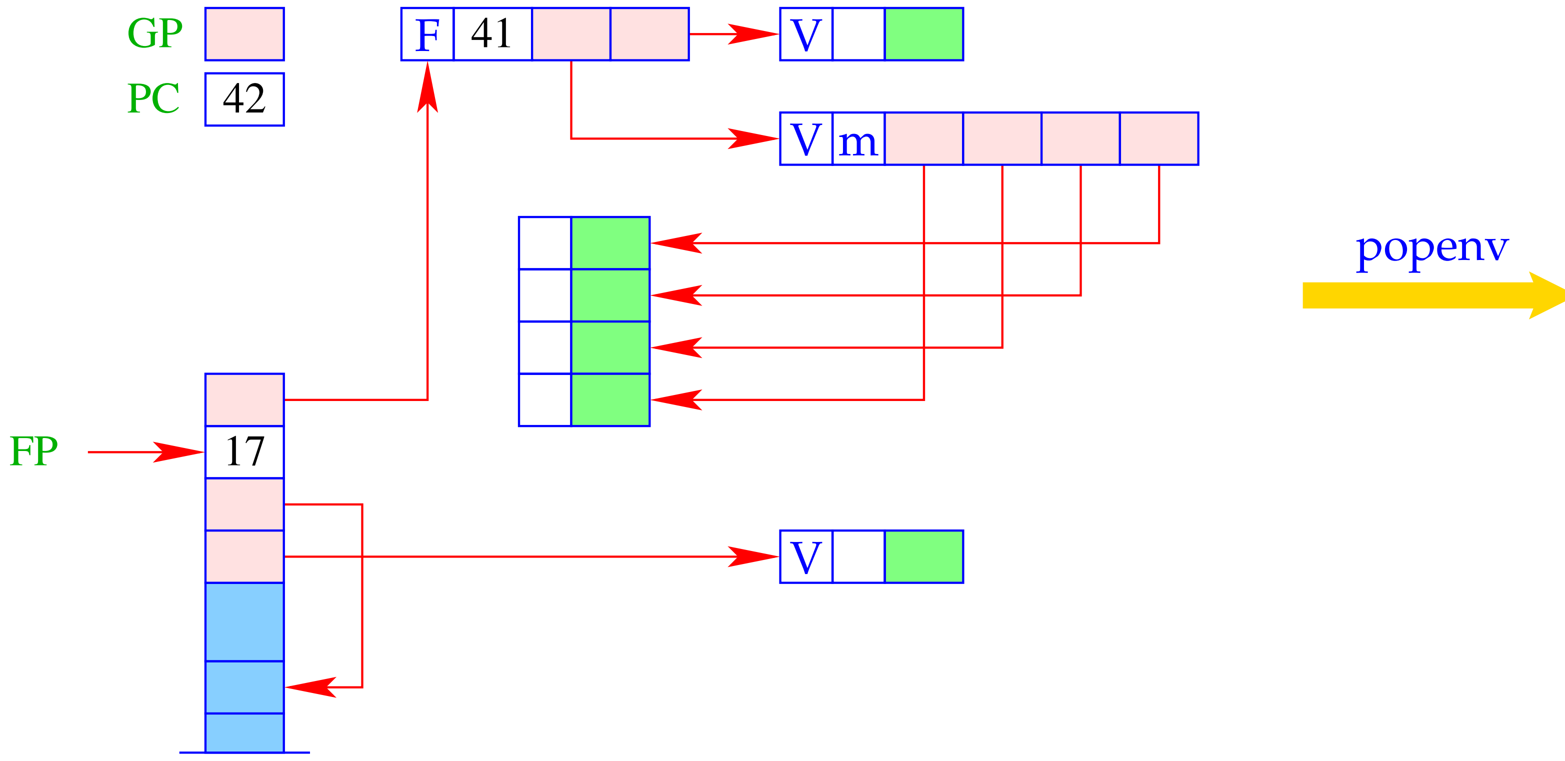


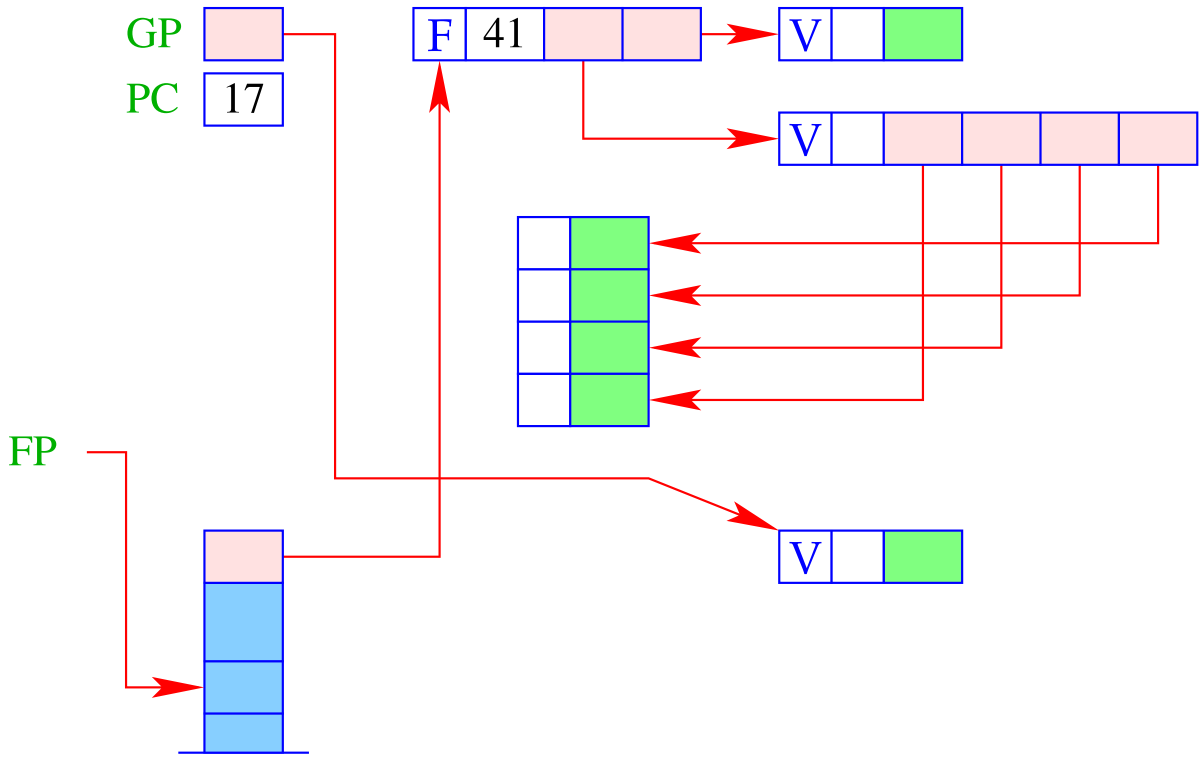

```
GP = S[FP-2];
S[FP-2] = S[SP];
PC = S[FP];
SP = FP - 2;
FP = S[FP-1];
```

Thus, we obtain for targ k in the case of under supply:

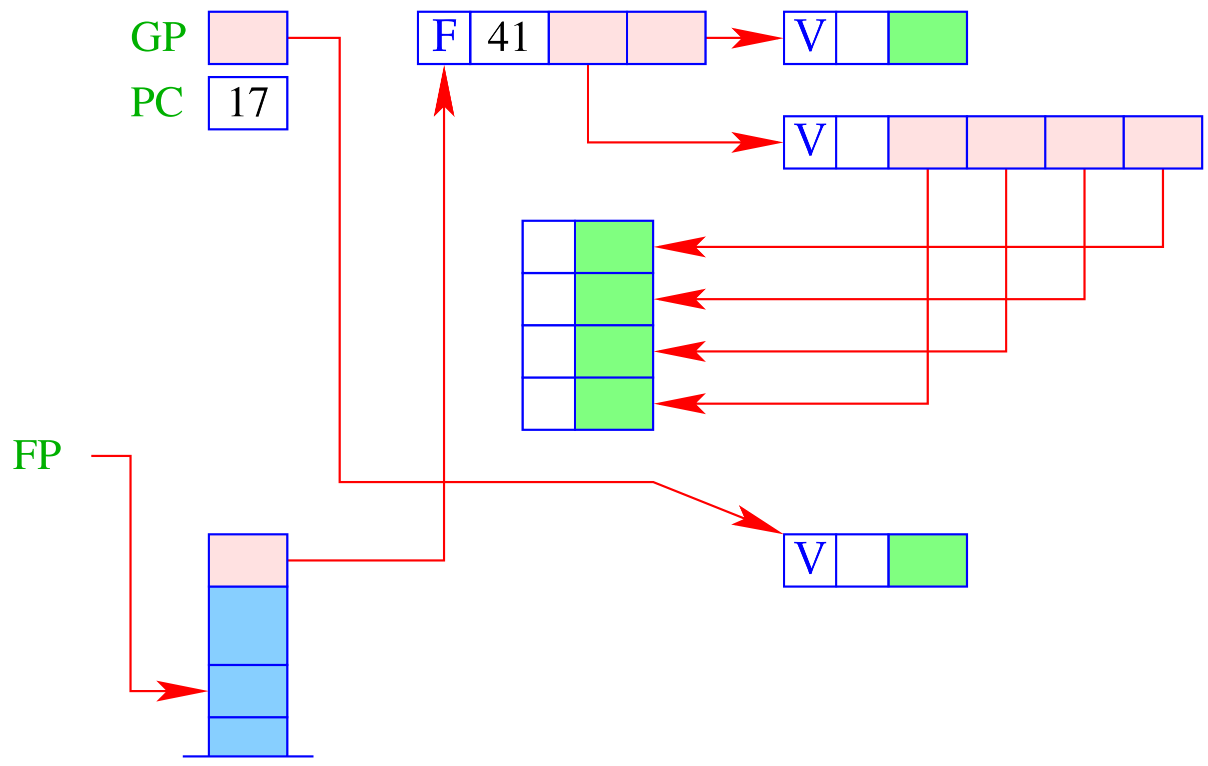



GP
PC
42
V
V
m
wrap
FP
17
V


GP
PC
42
F
41
V
V
m
FP
17
V
popenv



GP
PC
17
F
41
V
V
V
FP

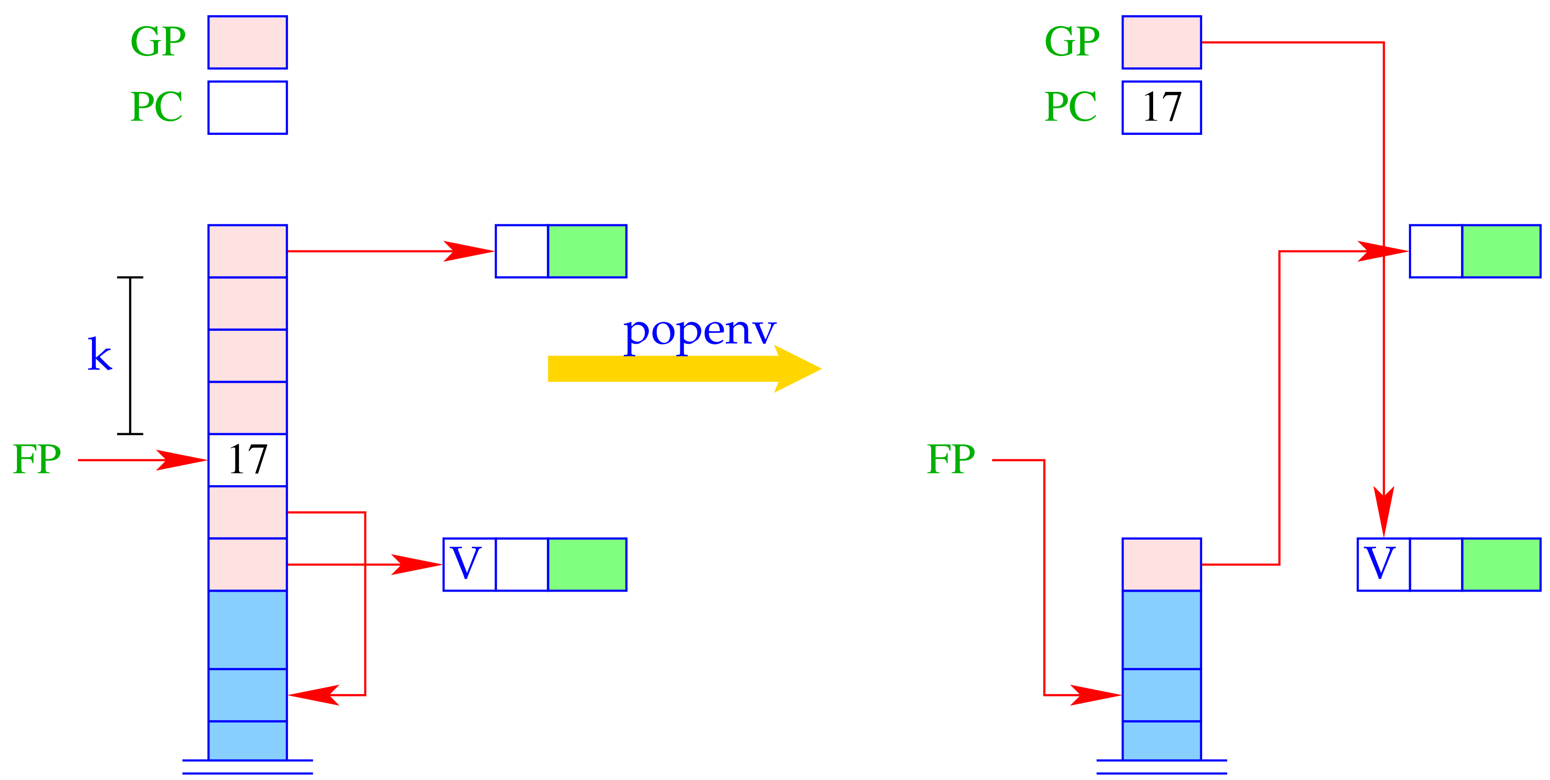

GP
PC
17
F
41
V
V
FP
V

- The stack frame can be released after the execution of the body if exactly the right number of arguments was available.
- If there is an oversupply of arguments, the body must evaluate to a function, which consumes the rest of the arguments ...
- The check for this is done by return k:

$$
\begin{array}{lll}
\text{return k} & = & \text{if } (\text{SP} - \text{FP} = \text{k} + 1) \\
 & & \quad \text{popenv;} \qquad\qquad\qquad // \text{ Done} \\
 & & \text{else } \{ \qquad\qquad\qquad\quad // \text{ There are more arguments} \\
 & & \quad \text{slide k;} \\
 & & \quad \text{apply;} \qquad\qquad\qquad // \text{ another application} \\
 & & \}
\end{array}
$$

The execution of return k results in:

Case: Done



Case: Over-supply