Funktsioonide definitsioonid

Funktsiooni definitsiooni transleerimisel genereeritakse kood, mis loob kuhjas *funktsionaalse väärtuse*:

- luuakse globaalvektor globaalsete muutujate sidumiseks;
- luuakse (algselt tühi) argumentvektor;
- luuakse F-objekt, mis sisaldab viitu neile vektoreile ning funktsiooni kehale vastava koodi algusaadressile.

Eraldi genereeritakse funktsiooni definitsiooni kehale vastav kood.

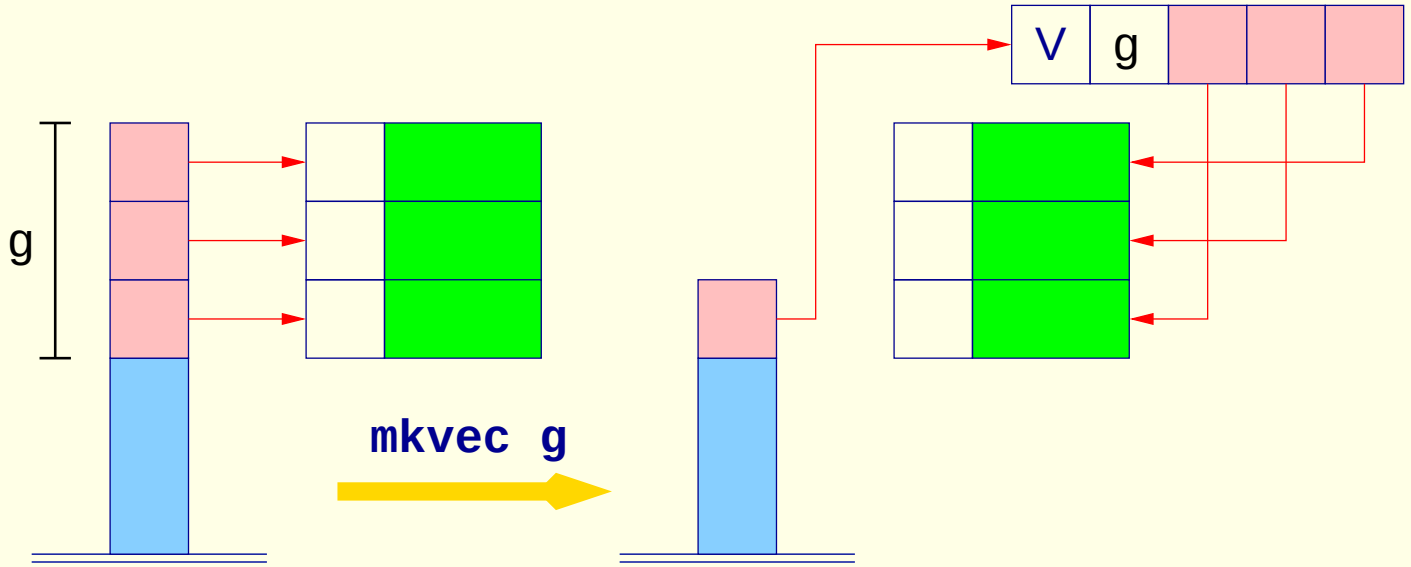
Funktsioonide definitsioonid

$$
\begin{array}{lll}
\text{code}_V\ (\textbf{fn}\ x_0,\ldots,x_{k-1} \Rightarrow e)\ \rho\ \text{sd} & = & \\
\quad \text{getvar}\ z_0\ \rho\ \text{sd} & \texttt{mkvec g} & \texttt{A: targ k} \\
\quad \text{getvar}\ z_1\ \rho\ (\text{sd}+1) & \texttt{mkfunval A} & \quad \text{code}_V\ e\ \rho'\ 0 \\
\quad \ldots & \texttt{jump B} & \quad \texttt{return k} \\
\quad \text{getvar}\ z_{g-1}\ \rho\ (\text{sd}+g-1) & & \texttt{B:}\ \ldots
\end{array}
$$

$$
\begin{array}{lrcl}
\text{kus} & \{z_0,\ldots,z_{g-1}\} & = & \mathit{free}(\textbf{fn}\ x_0,\ldots,x_{k-1} \Rightarrow e) \\
 & \rho' & = & \{x_i \mapsto (L,-i) \mid i=0,\ldots,k-1\} \\
 & & \cup & \{z_j \mapsto (G,\ j) \mid j=0,\ldots,g-1\}
\end{array}
$$

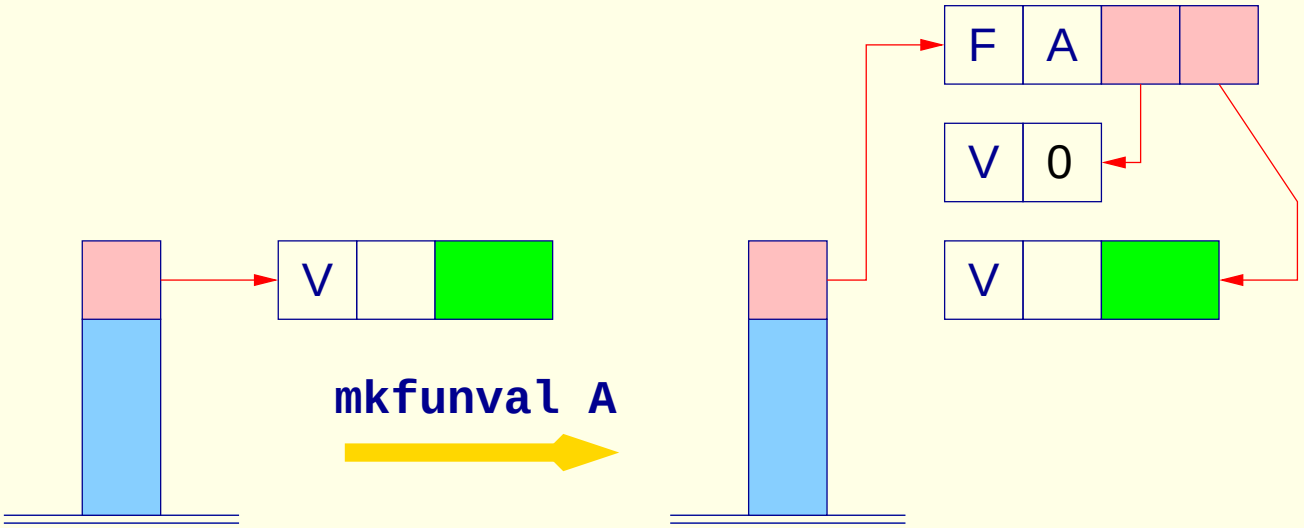
$$
\text{getvar}\ y\ \rho\ \text{sd} \quad = \quad \begin{cases} \texttt{pushloc}\ (\text{sd}-i) & \text{kui}\ \rho\ y = (L,i) \\ \texttt{pushglob i} & \text{kui}\ \rho\ y = (G,i) \end{cases}
$$

Funktsioonide definitsioonid




```
h = new (V, g);
SP = SP - g + 1;
for (i=0; i ≤ g; i++)
      h→v[i] = S[SP+i];
S[SP] = h;
```

Funktsioonide definitsioonid



a = new (V, 0);
S[SP] = new(F,A,a,S[SP]);

Funktsioonide definitsioonid

Näide: olgu $f \equiv \mathbf{fn}\ b \Rightarrow a + b$ keskkonnas $\rho = \{\ a \mapsto (L, 1)\ \}$ ja $sd = 2$.

$\text{code}_V\ f\ \rho\ sd$ emiteerib koodi:

```
2      pushloc 1
3      mkvec 1
3      mkfunval A
3      jump B
0   A: targ 1
```

```
0   pushglob 0
1   eval
1   getbasic
1   pushloc 1
2   eval
```

```
2      getbasic
2      add
1      mkbasic
1      return 1
3   B: ...
```

Käske `targ k` ja `return k` vaatame hiljem.

Funktsiooni aplikatsioon

Funktsiooni aplikatsiooni $e'\ e_0\ \ldots\ e_{m-1}$ korral genereeritakse kood, mis:

- loob magasinis uue freimi;
- kannab üle aktuaalsed parameetrid; so.

  CBV: väärtustab aktuaalsed parameetrid;

  CBN: loob aktuaalsete parameetrite sulundid;
- väärtustab funktsiooni e' F-objektiks;
- rakendab funktsiooni argumentidele.

Funktsiooni aplikatsioon

CBN korral genereeritakse kood:

$$
\begin{aligned}
\text{code}_V\ (e'\ e_0\ \ldots\ e_{m-1})\ \rho\ sd \quad = \quad & \texttt{mark A} \\
& \text{code}_C\ e_{m-1}\ \rho\ (sd+3) \\
& \text{code}_C\ e_{m-2}\ \rho\ (sd+4) \\
& \ldots \\
& \text{code}_C\ e_0\ \rho\ (sd+m+2) \\
& \text{code}_V\ e'\ \rho\ (sd+m+3) \\
& \texttt{apply} \\
\text{A:} \quad & \ldots
\end{aligned}
$$

CBV korral on argumentide e_i jaoks code_C asemel code_V.

Funktsiooni aplikatsioon

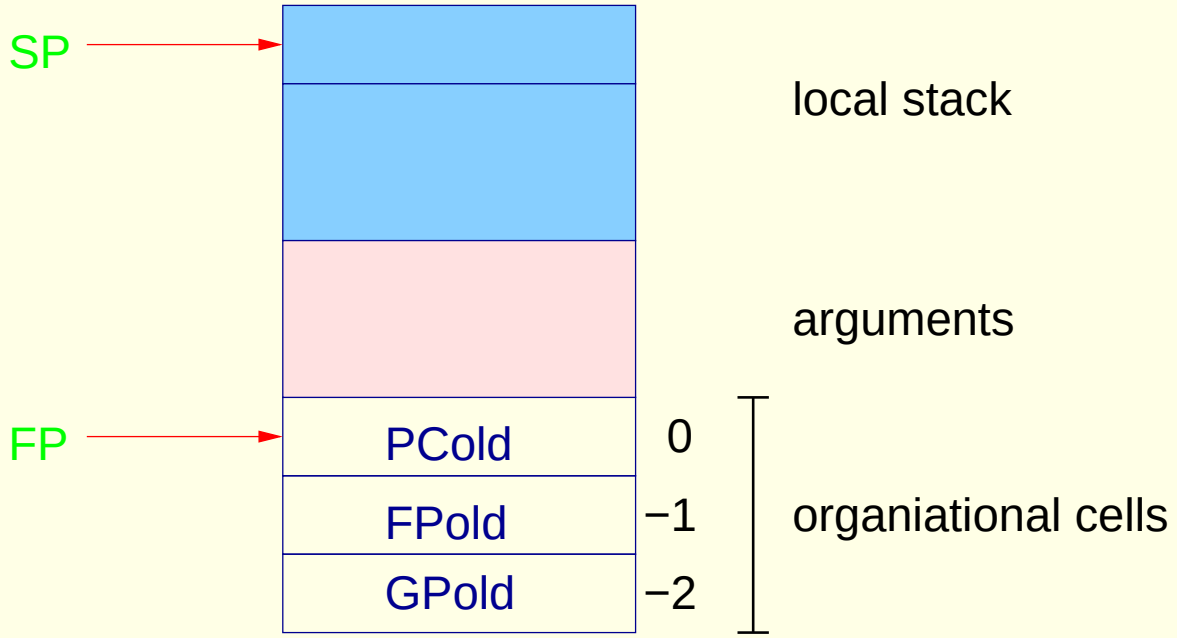
Näide: olgu $e \equiv f\ 42$ keskkonnas $\rho = \{\ f \mapsto (L, 2)\ \}$ ja $sd = 2$.

$\text{code}_V\ e\ \rho$ sd emiteerib CBV jaoks koodi:

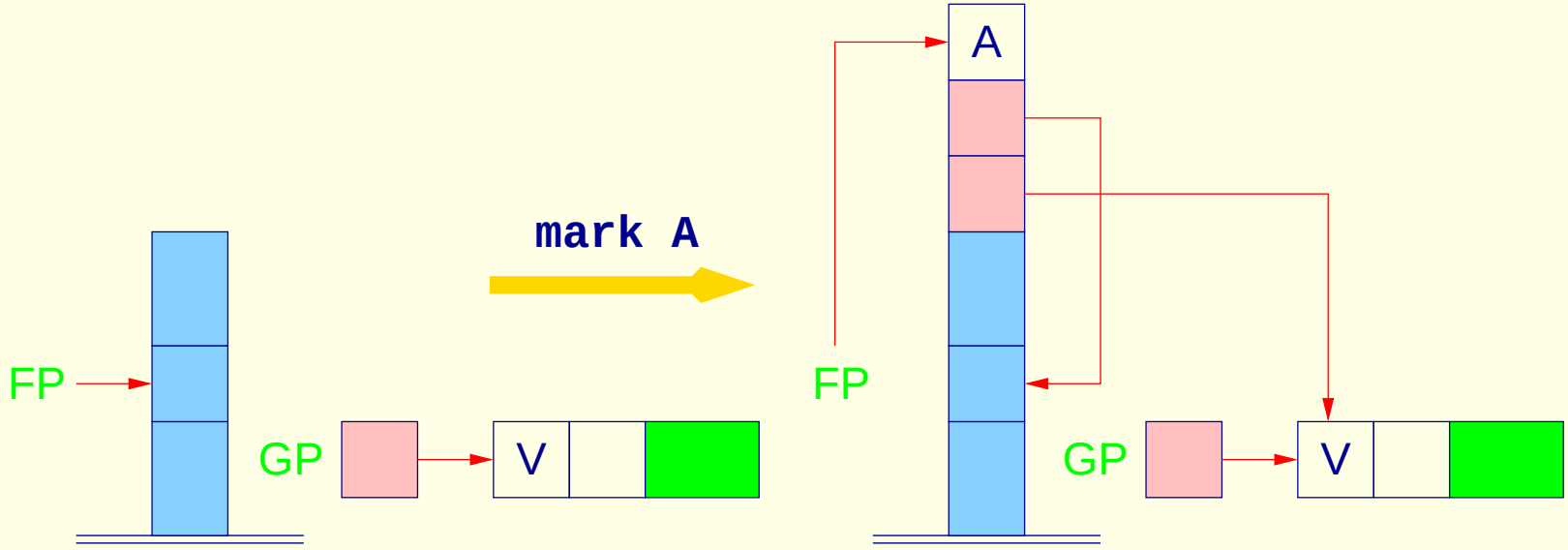
```
2   mark A          6     pushloc 4
5   loadc 42        7     apply
6   mkbasic         3  A: ...
```

Funktsiooni aplikatsioon

Freimi struktuur:

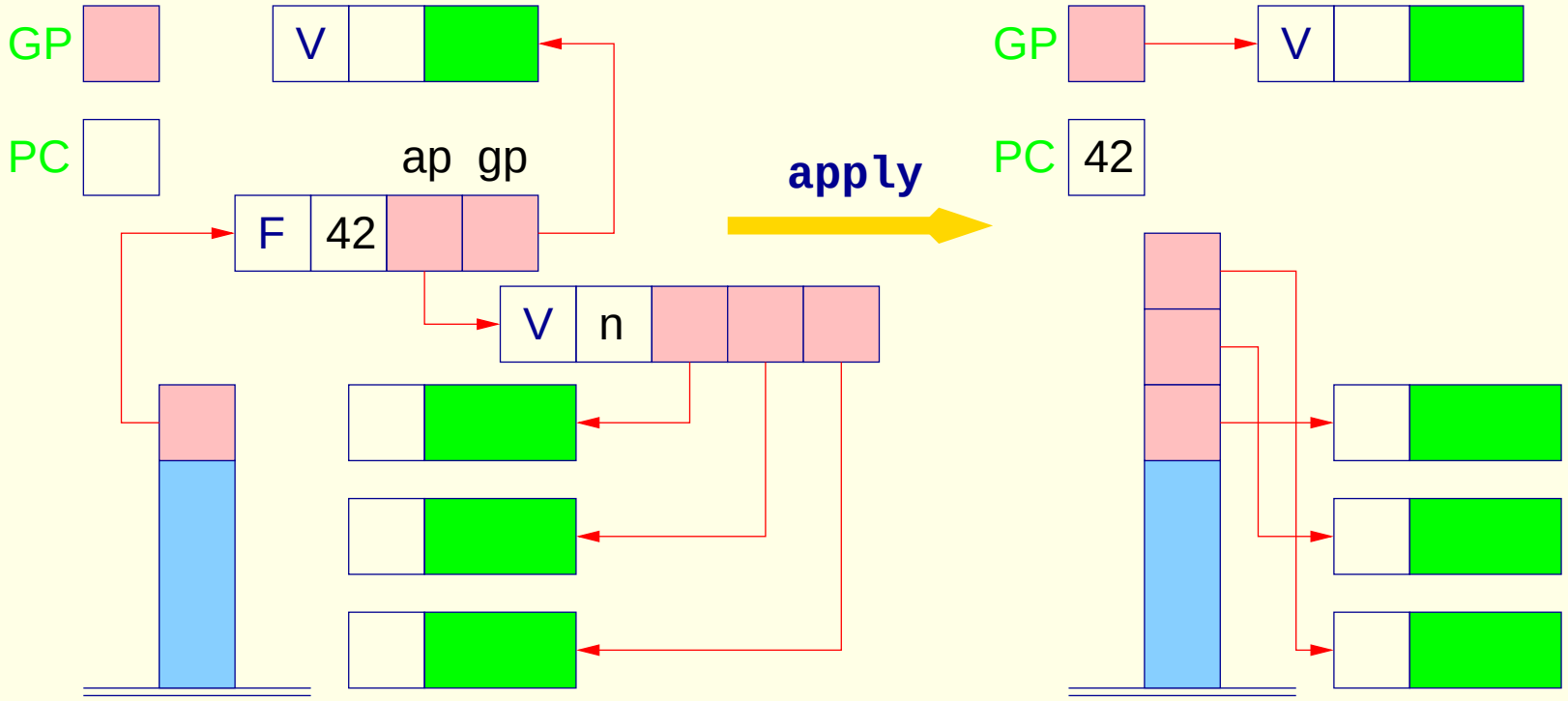


Funktsiooni aplikatsioon



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

Funktsiooni aplikatsioon



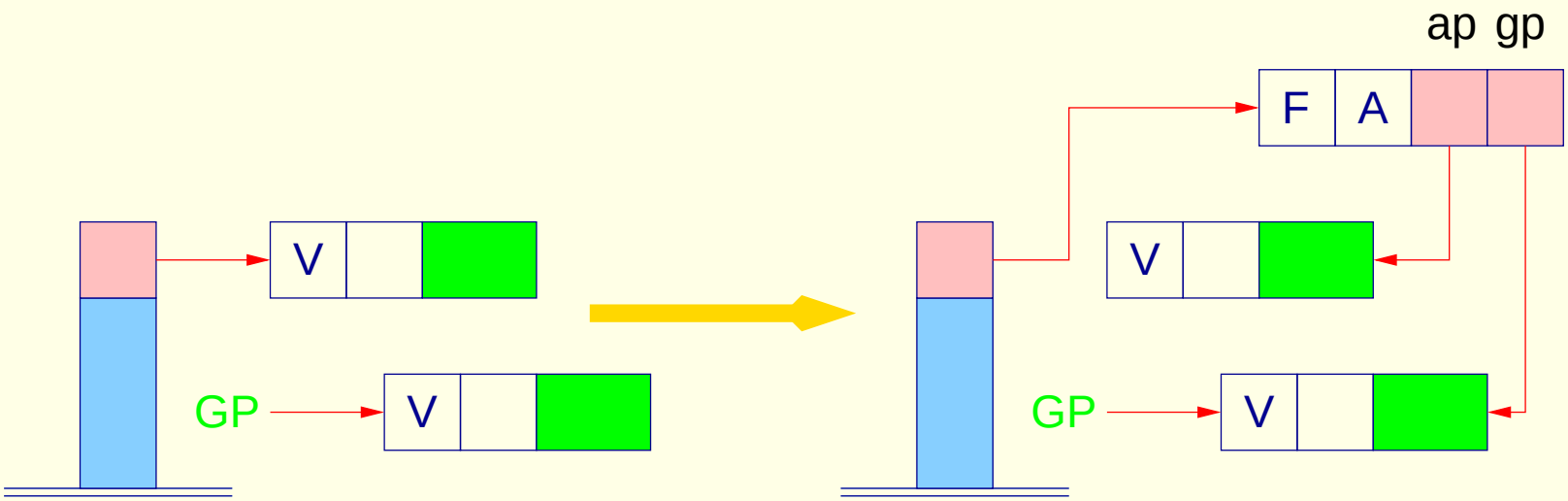

```
h = S[SP];
if (h→tag ≠ F)
   Error ("Not Function");
else {
   GP = h→gp; PC = h→cp;
   for (i=0; i < h→ap→n; i++)
      S[SP+i] = h→ap→v[i];
   SP = SP + h→ap→n - 1;
}
```

Argumentidega ala- ja ülevarustatus

- Esimene käsk pärast käsku `apply` on `targ k`.
- Kontrollib, et oleks funktsiooni väärtustamiseks piisavalt argumente
  - kontrollimiseks kasutab tingimust $\mathrm{SP} - \mathrm{FP} \geq k$.
- Kui on, siis alustatakse funktsiooni kehale vastava koodi täitmist.
- Vastasel korral väljastatakse uus funktsionaalne väärtus:
  - luuakse argumentvektor;
  - luuakse uus F-objekt;
  - magasinis vabastatakse freim.

Argumentidega ala- ja ülevarustatus

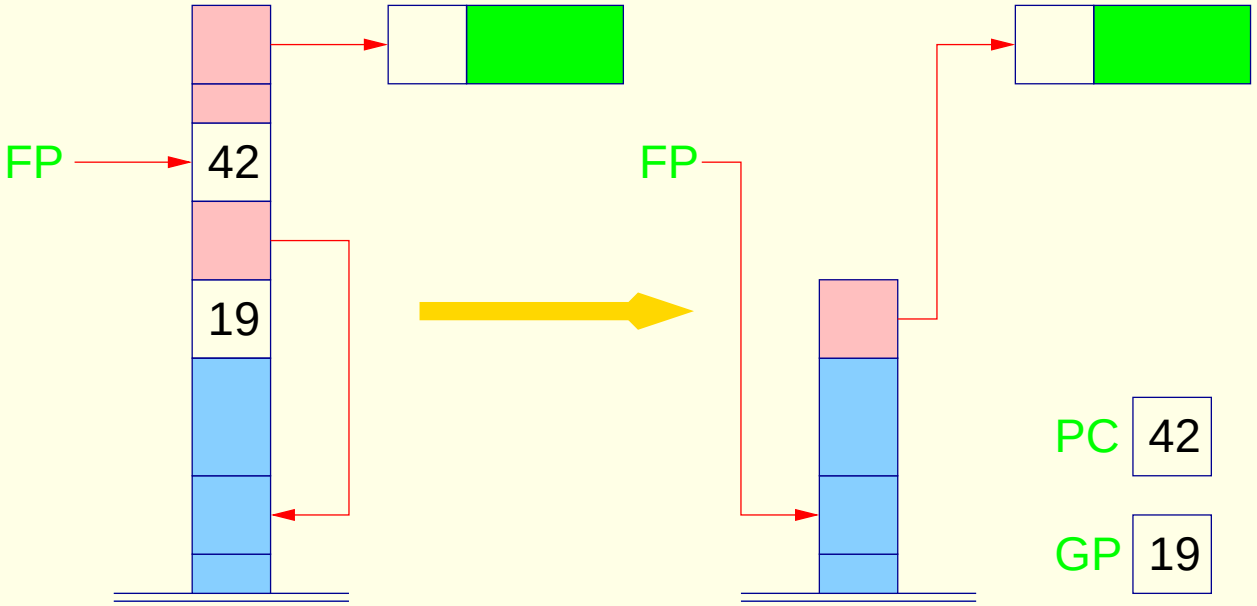
F-objekti konstrueerimine:



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

Argumentidega ala- ja ülevarustatus

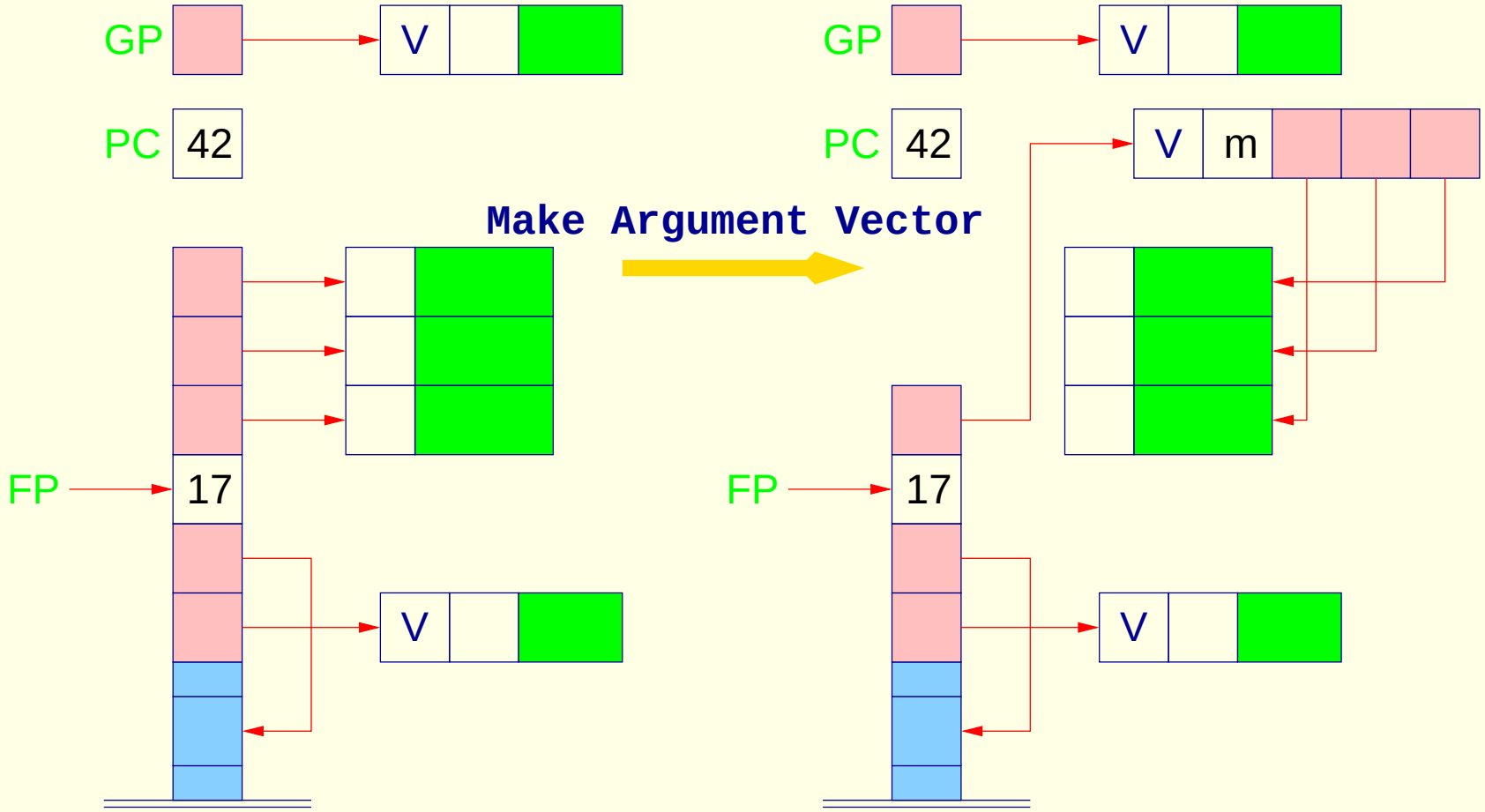
Freimi vabastamine:




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

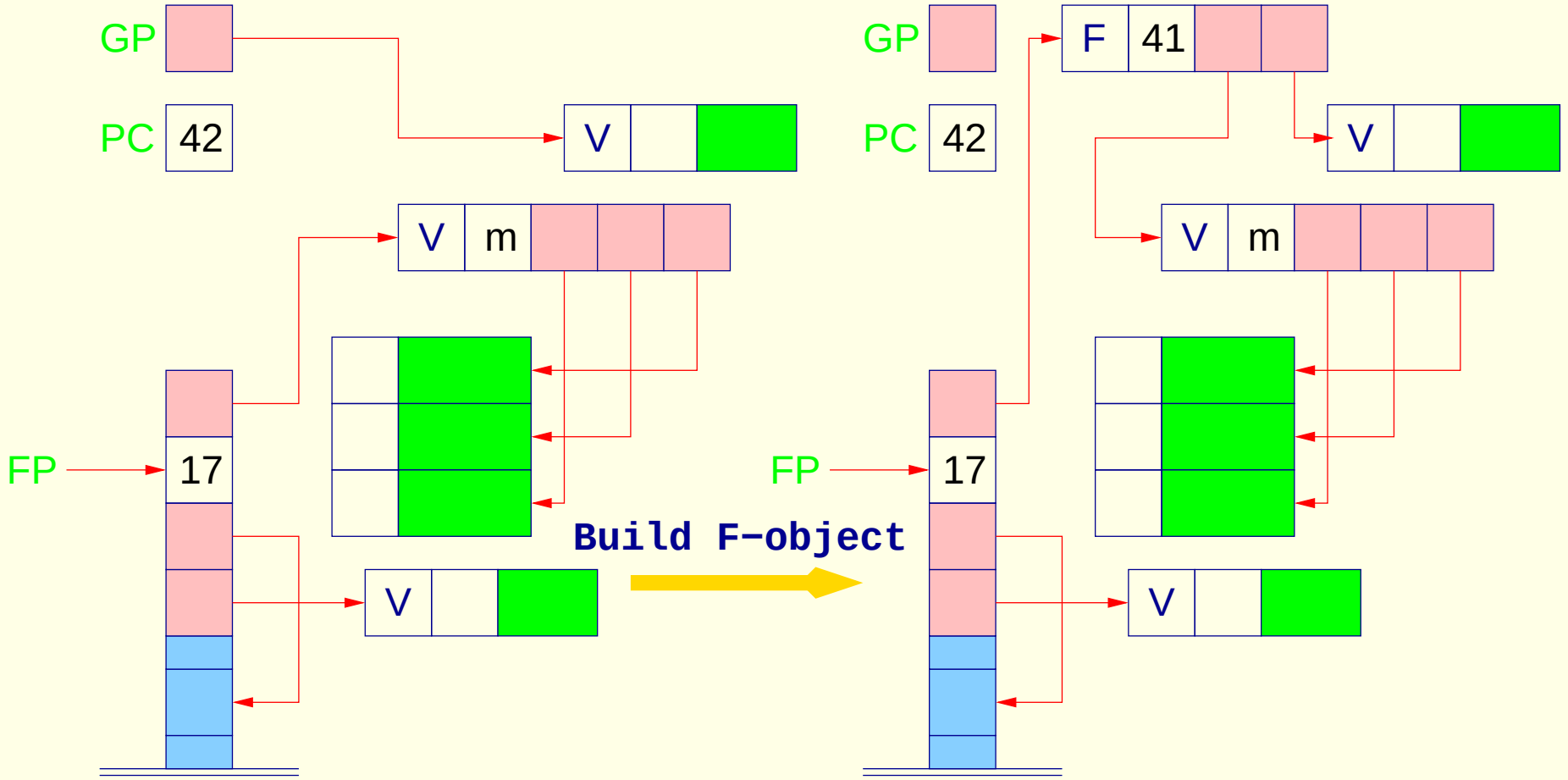
Argumentidega ala- ja ülevarustatus

`targ k`, kui argumente on $m < k$



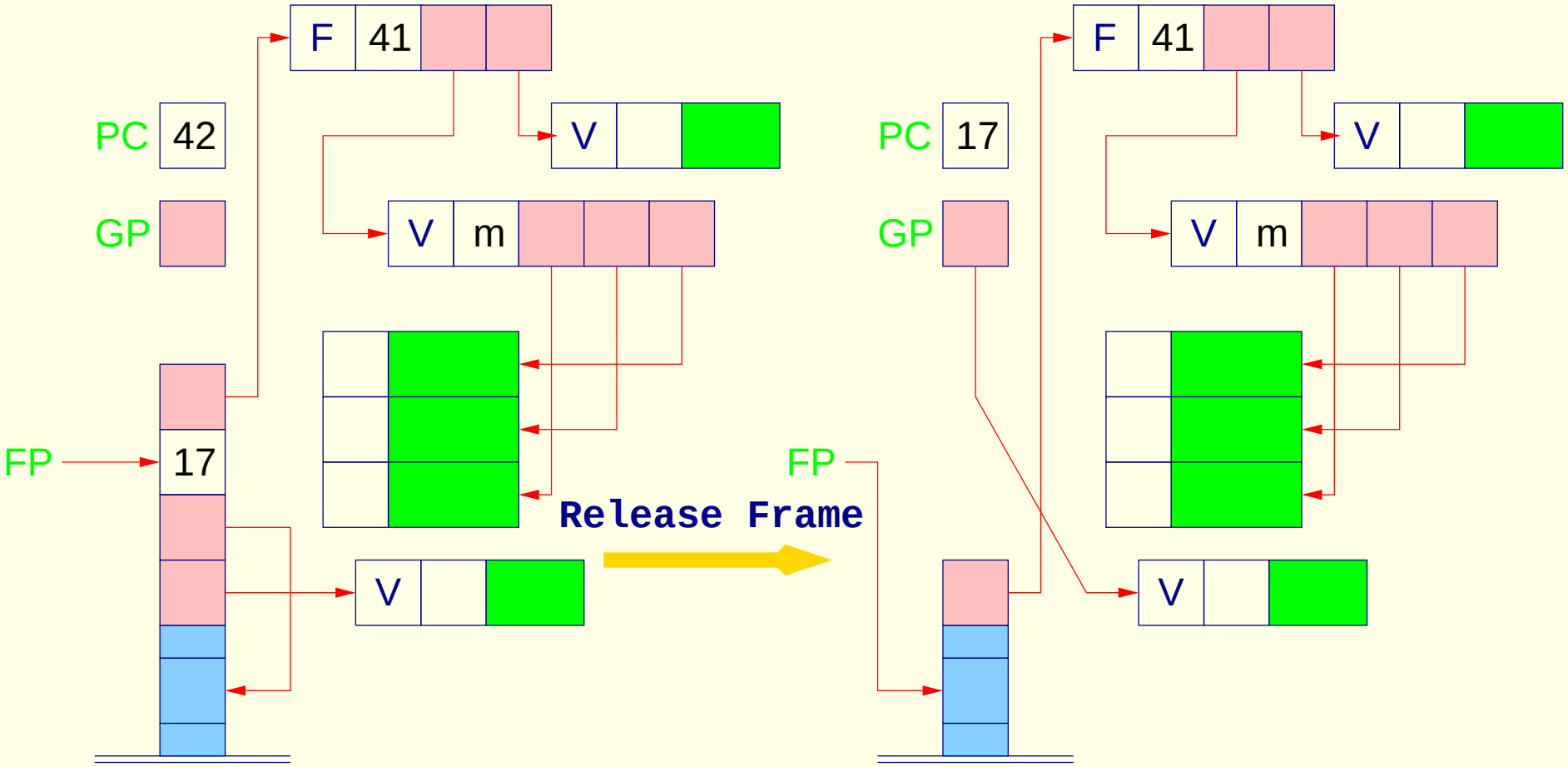
Argumentidega ala- ja ülevarustatus

`targ k`, kui argumente on $m < k$



Argumentidega ala- ja ülevarustatus

`targ k`, kui argumente on $m < k$



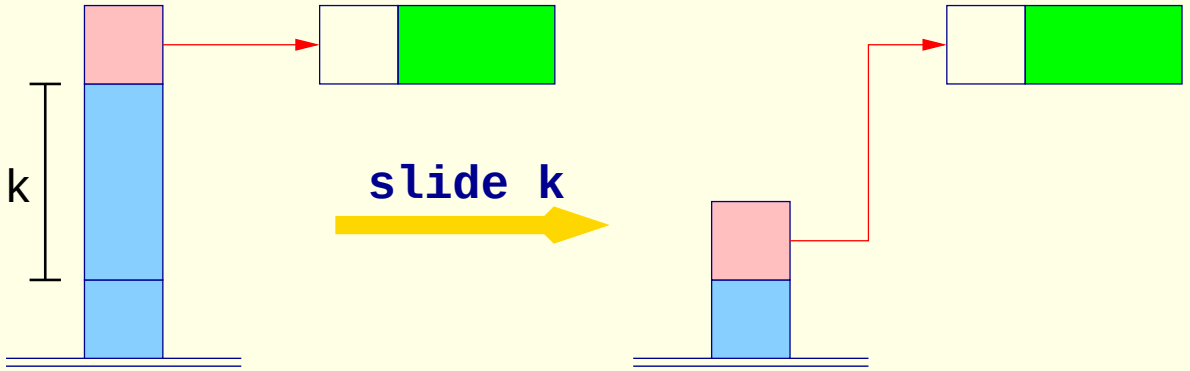
Argumentidega ala- ja ülevarustatus

- Funktsiooni keha kõige viimane käsk `return k` kontrollib, kas argumente oli täpselt õige arv.
- Kui oli, siis freim magasinis vabastatakse.
- Vastasel korral pidi funktsioon väärtustuma uueks funktsiooniks, mis kasutab ära ülejäänud argumandid.

```
return k   =   if (SP − FP = k − 1)
                 Release Stack Frame;
               else {
                 slide k;
                 apply;
               }
```

Argumentidega ala- ja ülevarustatus

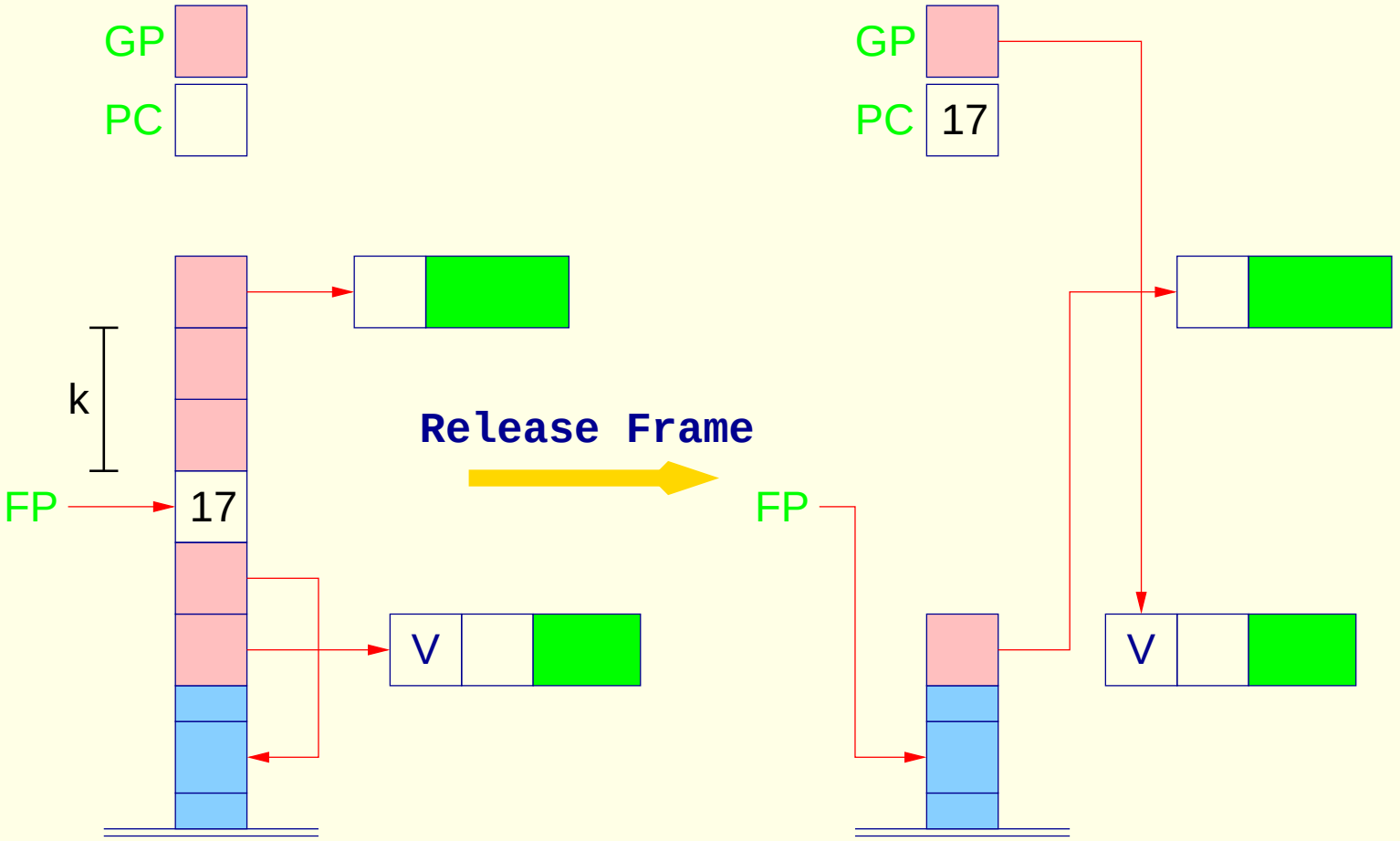
Käsk `slide k` nihutab magasini tipu *k* pesa allapoole, eemaldades vahepealsed pesad:




```
S[SP-k] = S[SP];
SP = SP - k;
```

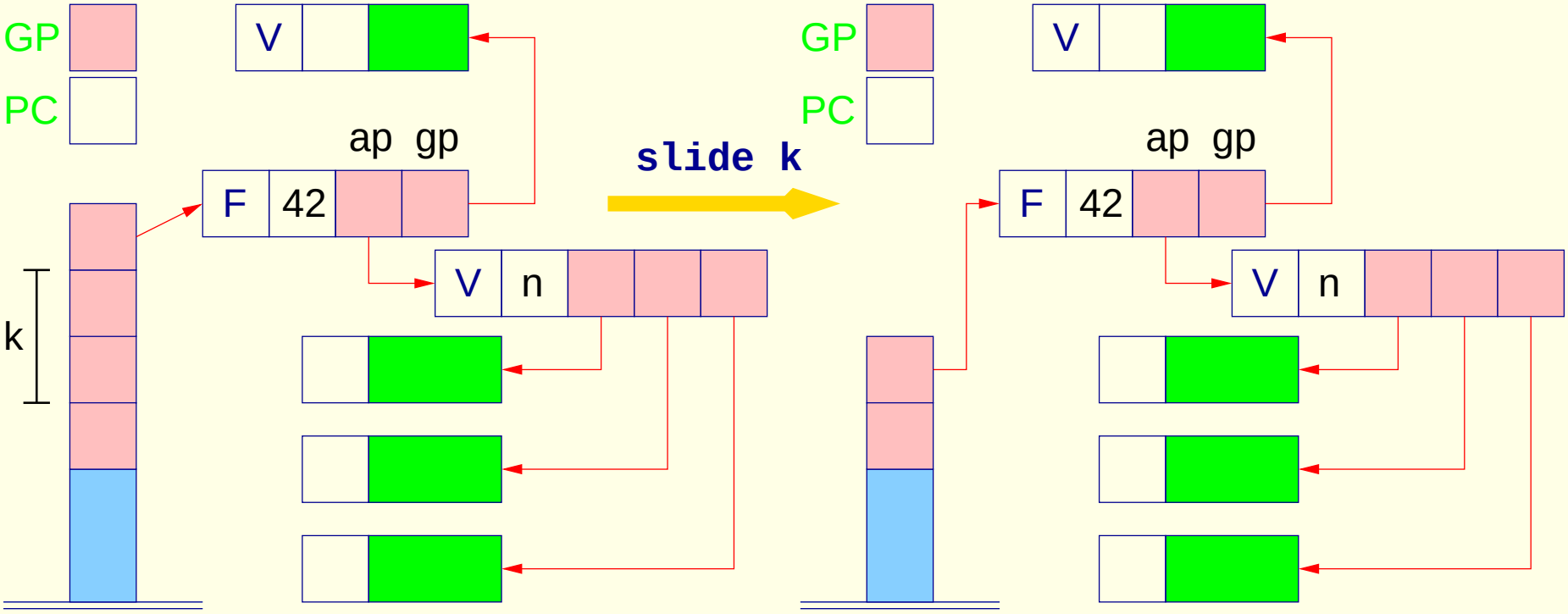
Argumentidega ala- ja ülevarustatus

`return k`, kui argumente on *k*



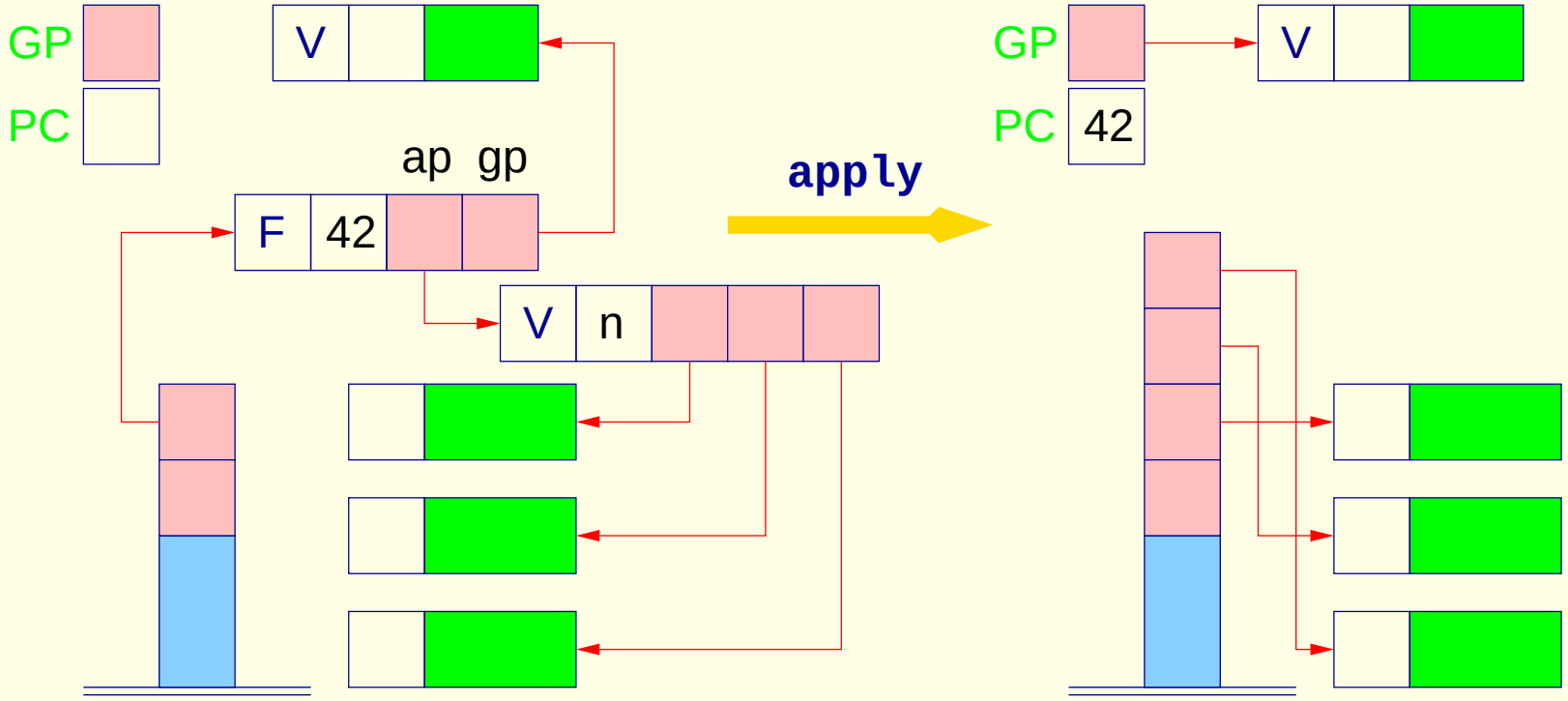
Argumentidega ala- ja ülevarustatus

`return k`, kui argumente on $m > k$



Argumentidega ala- ja ülevarustatus

`return k`, kui argumente on $m > k$



Lokaalsete definitsioonide transleerimine

Let-avaldise **let** $y_1 = e_1; \ \ldots; \ y_n = e_n$ **in** e_0 korral genereeritakse kood, mis:

- seob muutujad $y_1, \ldots, y_n$ vastavate väärtustega; so.

  CBV: väärtustab avaldised $e_1, \ldots, e_n$ ja seob muutujad nende väärtustega;

  CBN: loob avaldiste $e_1, \ldots, e_n$ sulundid ja seob muutujad nendega;

- väärtustab avaldise e_0 ja väljastab selle väärtuse.

Letrec-avaldise **letrec** $y_1 = e_1; \ \ldots; \ y_n = e_n$ **in** e_0 võivad avaldised e_i kasutada muutujaid y_j enne vastavate seoste loomist:

- muutujad seotakse algul fiktiivsetete väärtustega, mis hiljem muudetakse ära.

Lokaalsete definitsioonide transleerimine

CBN korral genereeritakse kood:

$$\begin{aligned}\text{code}_V\ (\textbf{let}\ y_1 = e_1;\ \ldots;\ y_n = e_n\ \textbf{in}\ e_0)\ \rho\ sd \quad = \quad & \text{code}_C\ e_1\ \rho\ sd \\ & \text{code}_C\ e_2\ \rho_1\ (sd+1) \\ & \ldots \\ & \text{code}_C\ e_n\ \rho_{n-1}\ (sd+n-1) \\ & \text{code}_V\ e_0\ \rho_n\ (sd+n) \\ & \texttt{slide n}\end{aligned}$$

kus $\rho_i = \rho \oplus \{y_i \mapsto (L,\ sd+i) \mid i = 1,\ldots,i\}$.

CBV korral on avaldiste e_i jaoks code_C asemel code_V.

NB! Kõigi avaldistega e_i on seotud sama globaalkeskkond.

Lokaalsete definitsioonide transleerimine

Näide: olgu $e \equiv$ **let** $a = 19;\ b = a * a$ **in** $a + b$ keskkonnas $\rho = \emptyset$ ja $sd = 0$.
$\text{code}_V\ e\ \rho\ sd$ emiteerib CBV jaoks koodi:

```
0  loadc 19
1  mkbasic
1  pushloc 0
2  getbasic
2  pushloc 1
3  getbasic
3  mul
2  mkbasic
2  pushloc 1
2  getbasic
3  pushloc 1
4  getbasic
4  add
3  mkbasic
3  slide 2
```

Lokaalsete definitsioonide transleerimine

CBN korral genereeritakse kood:

$\text{code}_V\ (\textbf{letrec}\ y_1 = e_1;\ \ldots;\ y_n = e_n\ \textbf{in}\ e_0)\ \rho\ sd$ =

```
alloc n
```
$\text{code}_C\ e_1\ \rho'\ (sd + n)$
```
rewrite n
```
...
$\text{code}_C\ e_n\ \rho'\ (sd + n)$
```
rewrite 1
```
$\text{code}_V\ e_0\ \rho'\ (sd + n)$
```
slide n
```

kus $\rho' = \rho \oplus \{y_i \mapsto (L,\ sd + i) \mid i = 1, \ldots, n\}$.

CBV korral on avaldiste e_i jaoks code_C asemel code_V.

NB! e_i ei tohi CBV korral olla baasväärtus.

Lokaalsete definitsioonide transleerimine

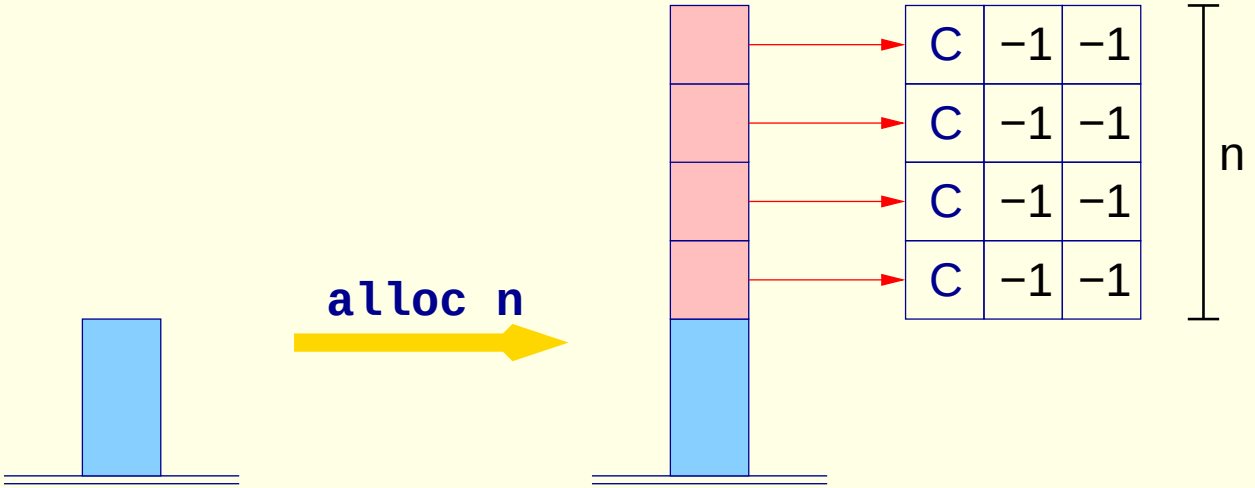
Näide:

$$e \equiv \textbf{letrec } f = \textbf{fn } x, y \Rightarrow \textbf{if } y \leq 1 \textbf{ then } x \textbf{ else } f\,(x * y)\,(y-1) \textbf{ in } f\,1$$

$\text{code}_V\ e\ \rho\ sd$, kus $\rho = \emptyset$ ja $sd = 0$, emiteerib CBV jaoks koodi:

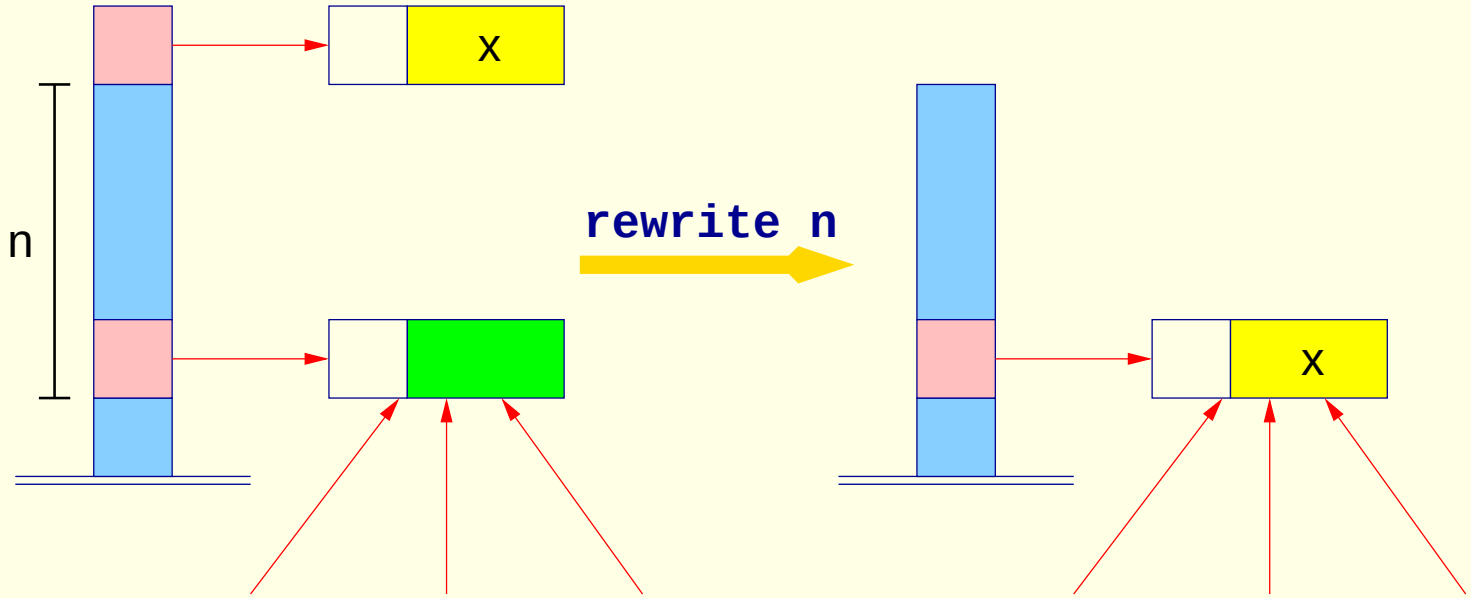
```
0   alloc 1          0   A: targ 2        4     loadc 1
1   pushloc 0        0      ...           5     mkbasic
2   mkvec 1          0      return 2      5     pushloc 4
2   mkfunval A       2   B: rewrite 1     6     apply
2   jump B           1      mark C        2  C: slide 1
```

Lokaalsete definitsioonide transleerimine




```
for (i=1; i≤n; i++)
   S[SP+i] = new(C,-1,-1);
SP = SP + n;
```

Lokaalsete definitsioonide transleerimine




```
H[S[SP-n]] = H[S[SP]];
SP = SP - 1;
```

- Viit $S[SP - n]$ jääb samaks!
- Ainult tema *sisu* muutub!