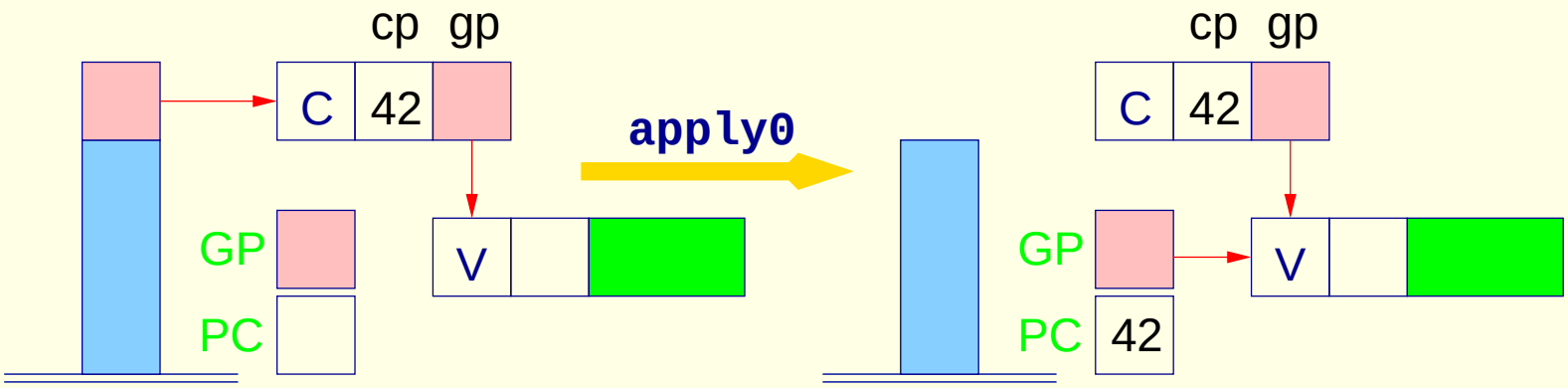
Sulundid ja nende väärtustamine

- Sulundeid on tarvis CBN semantika korral.
- Enne muutuja poole pöördumist peab tema väärtus olema saadaval.
- Vastasel korral tuleb temaga seotud sulund väärtustada.
- Sulund on sisuliselt parameetriteta funktsioon.
- Seega, sulundi väärtustamine on tema rakendamine 0 argumendile.
- Sulundi väärtustamine toimub käsu `eval` abil.

```
eval  =  if (S[SP] → tag = C) {
             mark PC;
             pushloc 3;
             apply0;
         }
```

Sulundid ja nende väärtustamine



h = S[SP];

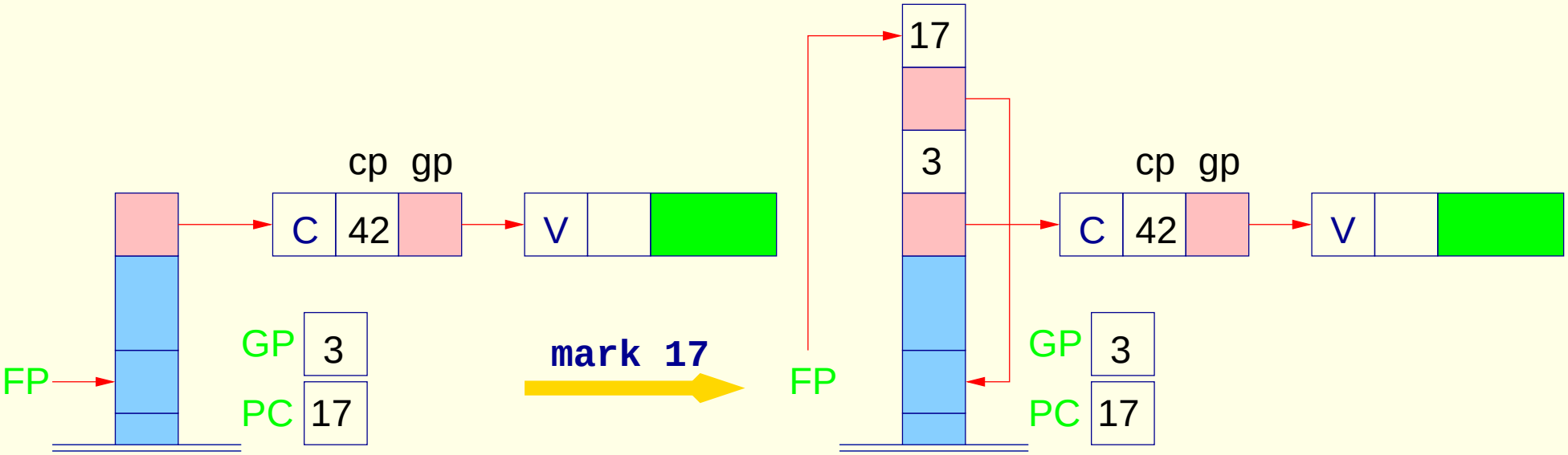
SP–;

GP = h→gp;

PC = h→cp;

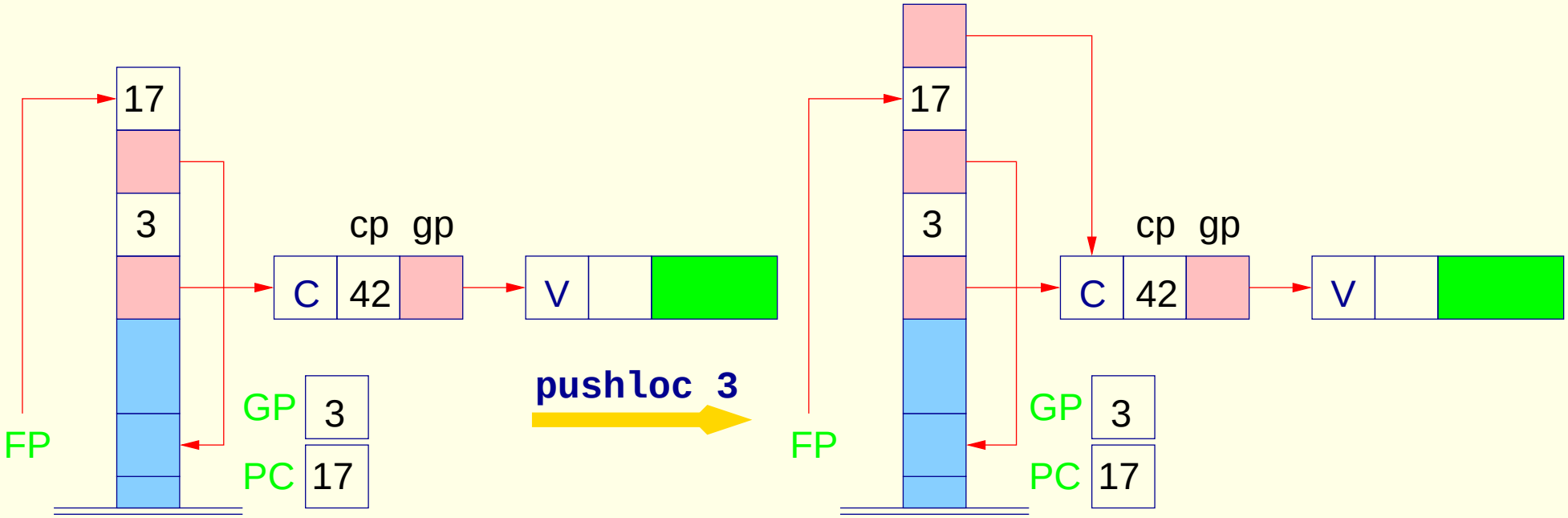
Sulundid ja nende väärtustamine

Sulundi väärtustamine käsu `eval` abil:



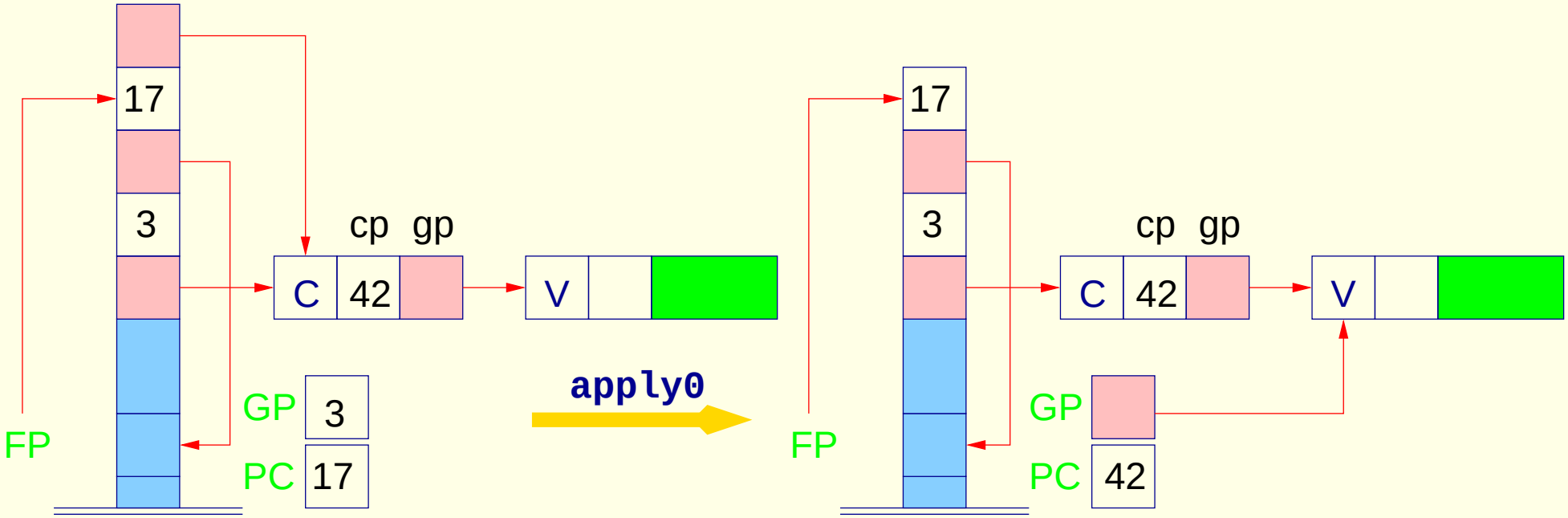
Sulundid ja nende väärtustamine

Sulundi väärtustamine käsu `eval` abil:



Sulundid ja nende väärtustamine

Sulundi väärtustamine käsu `eval` abil:



Sulundid ja nende väärtustamine

Avaldise e sulundi konstrueerimiseks:

- pakitakse temas esinevad vabad muutujad globaalvektorisse;
- luuakse C-objekt, mis viitab sellele vektorile ja avaldise väärtustamisele vastava koodi algusse.

$\text{code}_C\ e\ \rho\ \text{sd}\quad =$

getvar z_0 ρ sd	`mkvec g`	A: $\text{code}_V\ e\ \rho'\ 0$
getvar z_1 ρ $(\text{sd}+1)$	`mkclos A`	`update`
$\ldots$	`jump B`	B: $\ldots$
getvar z_{g-1} ρ $(\text{sd}+g-1)$		

kus $\{z_0,\ldots,z_{g-1}\} = \mathit{free}(\mathbf{fn}\ x_0,\ldots,x_{k-1} \Rightarrow e)$

ja $\rho' = \{z_i \mapsto (G,\ i) \mid i = 0,\ldots,g-1\}$

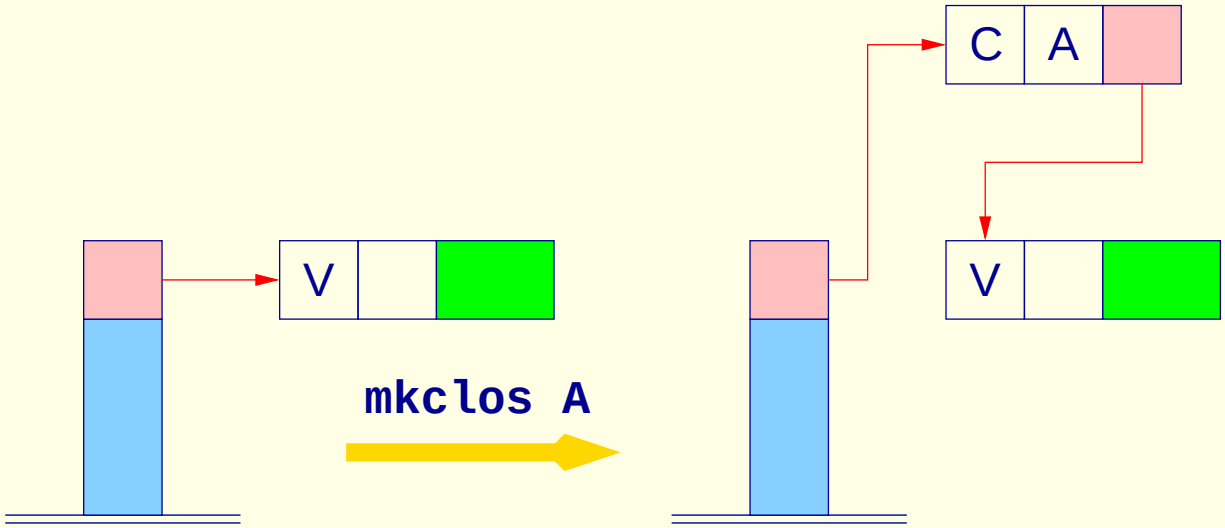
Sulundid ja nende väärtustamine

Näide: olgu $e \equiv a * a$ keskkonnas $\rho = \{ a \mapsto (L, 0) \}$ ja $sd = 1$.

$\text{code}_C\ e\ \rho\ sd$ emiteerib koodi:

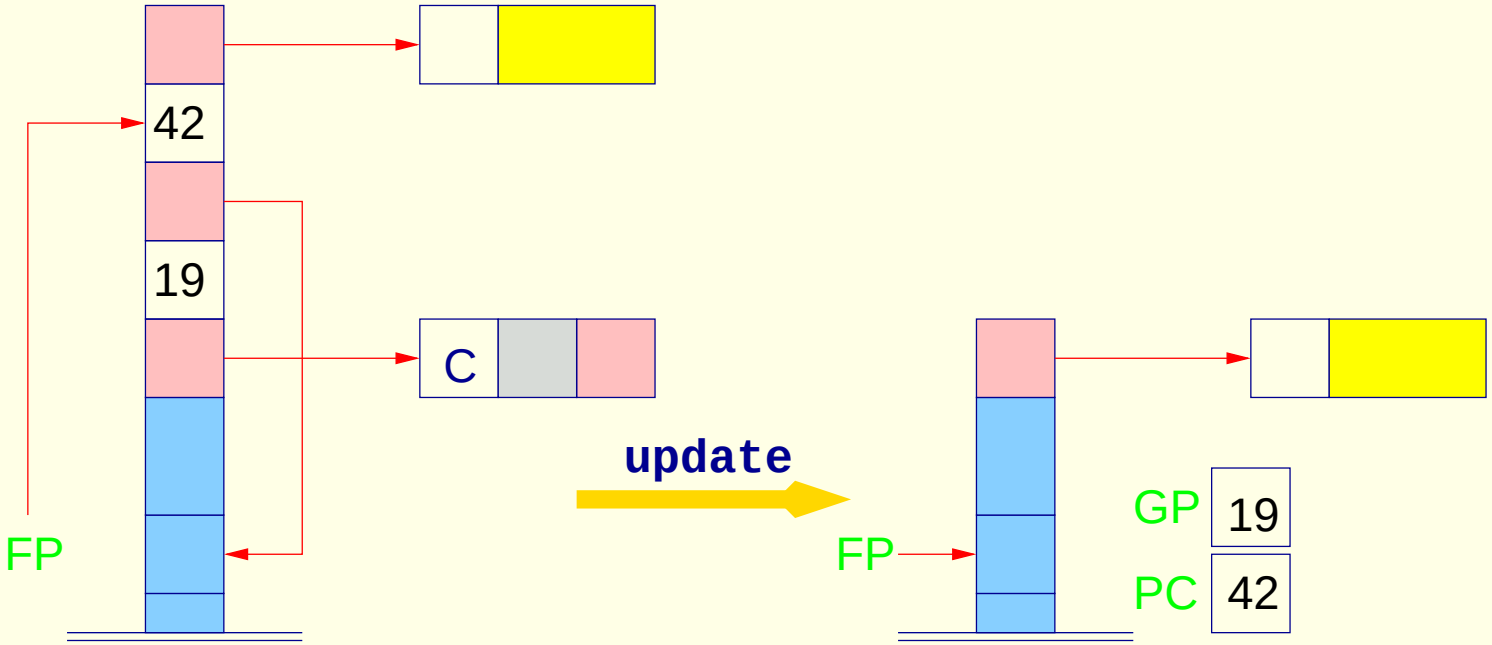
```
1   pushloc 1
2   mkvec 1
2   mkclos A
2   jump B
0   A: pushglob 0
1      eval
1      getbasic
2      pushglob 0
2      eval
2      getbasic
2      mul
1      mkbasic
1      update
2   B: ...
```

Sulundid ja nende väärtustamine



S[SP] = new(C,A,S[SP]);

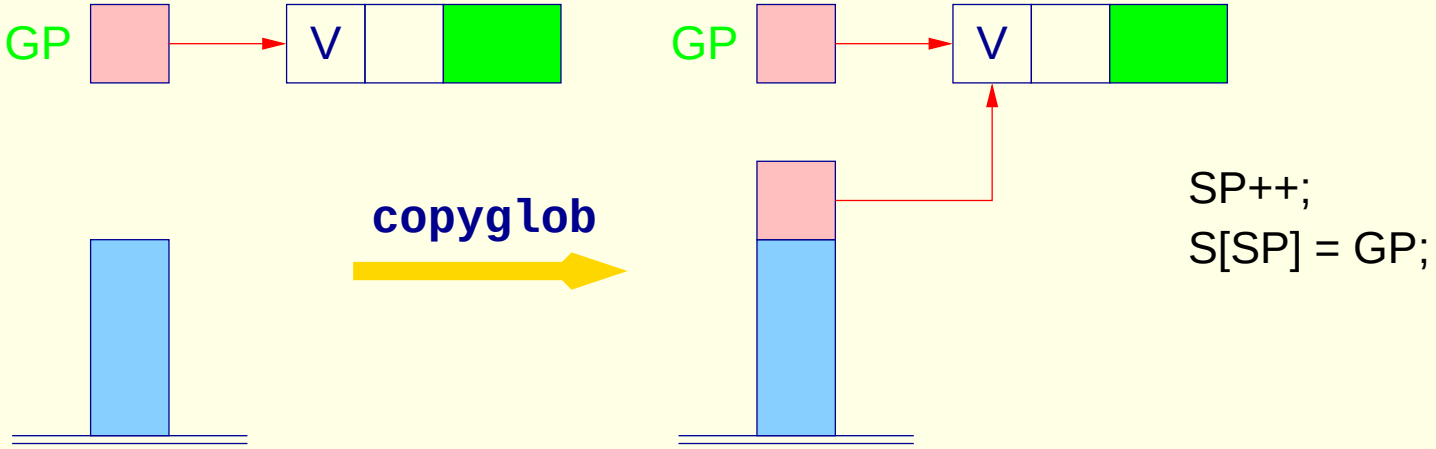
Sulundid ja nende väärtustamine



Optimiseerimine I: Globaalsed muutujad

- Funktsionaalsed programmid konstrueerivad palju F- ja C-objekte.
- Muuhulgas tähendab see uute globaalvektorite loomist.
- ”Globaalselt defineeritud” (top level) muutujatele võime staatiliselt määrata absoluutaadressid, mille kaudu neile ligi pääseb.
- Kuna need absoluutaadressid on teada transleerimisajal, siis pole neid muutujaid vaja globaalvektoritesse lisada.
- Tihti on võimalik globaalvektoreid ka korduvkasutada.
- Kasulik näiteks let-avaldiste või funktsiooni aplikatsioonide transleerimisel, kus võib konstrueerida ühe globaalvektori, mis sisaldab kõigi definitsioonide või argumentide vabu muutujaid.
- Selleks, analoogselt lokaalsetele muutujatele, salvestame korduvkasutatavad globaalvektorid magasini.

Optimiseerimine I: Globaalsed muutujad



- Jagatavad globaalvektorid sisaldavad tihti rohkem muutujaid, kui antud avaldises on vabu muutujaid:
  - mida rohkem on vektoris muutujaid, seda suurema tõenäosusega on teda võimalik korduvkasutada.
- Liigsed muutujad globaalvektorites võivad takistada mälu vabastamist (space leaks).
- Võimalik lahendus: kustutada viit pärast tema ”eluea” lõppu.

Optimiseerimine II: Sulundid

- Avaldise *e* sulundi konstrueerimine lükkab antud avaldise väärtustamise edasi ajani kuni selle avaldise väärtust on tegelikult vaja.
- Kui avaldise väärtust pole tarvis, siis talle vastavat sulundit ka ei väärtustata (*laisk väärtustamine*).
- Kui aga staatiliselt on teada, et avaldise väärtust on kindlasti vaja (strictness analysis), siis on vahepealne sulundi konstrueerimine asjatu lisatöö.
- Seega, kui avaldis *e* on *agaras kontekstis*, siis:

$$\text{code}_C\ e\ \rho\ sd \quad = \quad \text{code}_V\ e\ \rho\ sd$$

- Sulundi konstrueerimine võib olla mõtetu ka juhul, kui avaldis on väga lihtne.

Optimiseerimine II: Sulundid

Baasväärtused:

Sulundi konstrueerimine baasväärtusele on vähemalt sama kulukas, kui B-objekti enda konstrueerimine!

Seega:

$$\text{code}_C\ b\ \rho\ \text{sd} \quad = \quad \text{code}_V\ b\ \rho\ \text{sd} \quad = \quad \begin{array}{l}\texttt{loadc b}\\ \texttt{mkbasic}\end{array}$$

See asendab koodijada:

```
mkvec 0        A: loadc b      B: ...
mkclos A          mkbasic
jump B            update
```

Optimiseerimine II: Sulundid

Muutujad:

Muutuja on seotud kas väärtuse või C-objektiga ning uue sulundi konstrueerimine on üleliigne. Seega:

$$\text{code}_C\ x\ \rho\ sd \quad = \quad \text{getvar}\ x\ \rho\ sd$$

See asendab koodijada:

```
getvar x ρ sd      mkclos A      A: pushglob 0      update
mkvec 1            jump B           eval          B: ...
```

Näide: olgu $e \equiv$ **letrec** $a = b;\ b = 7$ **in** a, siis $\text{code}_V\ e\ \emptyset\ 0$ genereerib:

```
0  alloc 2        2  loadc 7        2  pushloc 1
2  pushloc 0      3  mkbasic        3  eval
3  rewrite 2      3  rewrite 1      3  slide 2
```

Optimiseerimine II: Sulundid

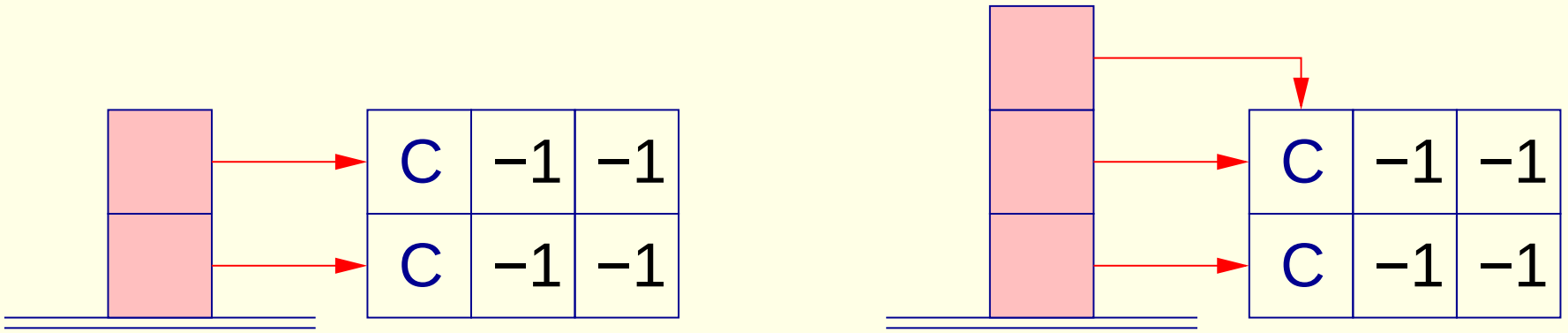
```
0  alloc 2        2  loadc 7       2  pushloc 1
2  pushloc 0      3  mkbasic       3  eval
3  rewrite 2      3  rewrite 1     3  slide 2
```

alloc 2

Optimiseerimine II: Sulundid

```
0  alloc 2          2  loadc 7         2  pushloc 1
2  pushloc 0        3  mkbasic         3  eval
3  rewrite 2        3  rewrite 1       3  slide 2
```

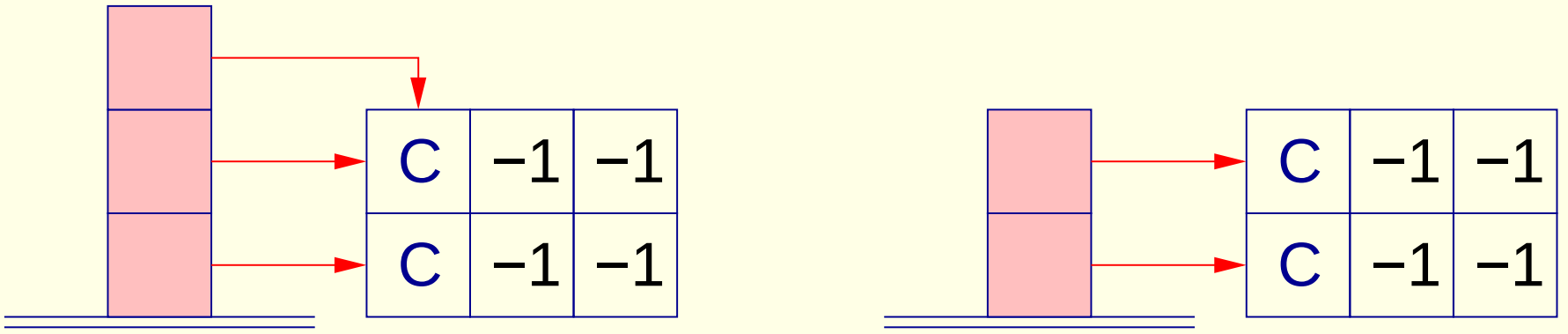
pushloc 0



Optimiseerimine II: Sulundid

```
0  alloc 2          2  loadc 7         2  pushloc 1
2  pushloc 0        3  mkbasic         3  eval
3  rewrite 2        3  rewrite 1       3  slide 2
```

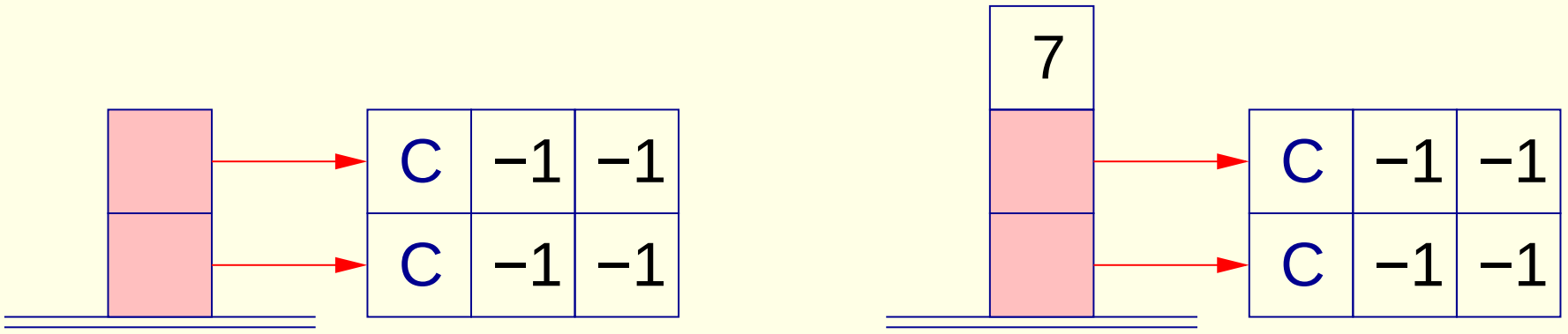
rewrite 2



Optimiseerimine II: Sulundid

```
0   alloc 2          2   loadc 7          2   pushloc 1
2   pushloc 0        3   mkbasic          3   eval
3   rewrite 2        3   rewrite 1        3   slide 2
```

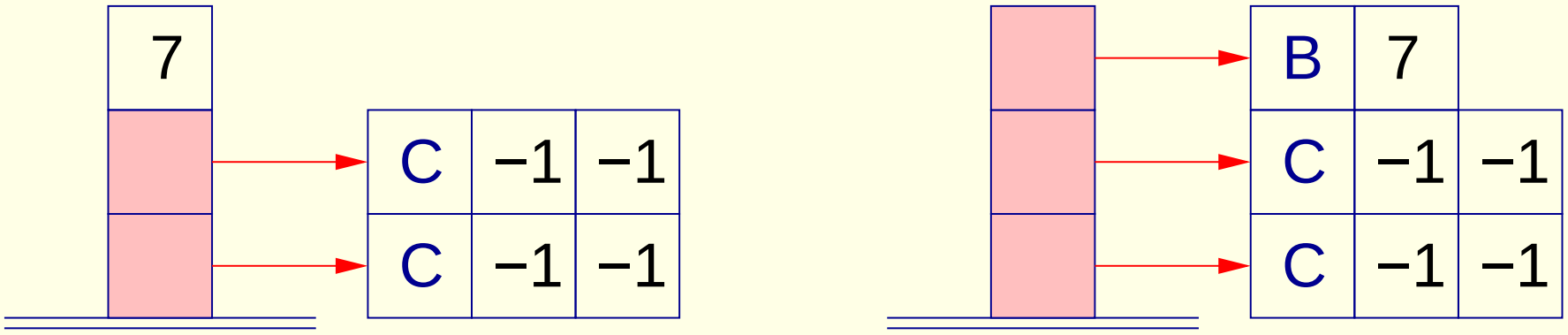
loadc 7



Optimiseerimine II: Sulundid

```
0  alloc 2        2  loadc 7       2  pushloc 1
2  pushloc 0      3  mkbasic       3  eval
3  rewrite 2      3  rewrite 1     3  slide 2
```

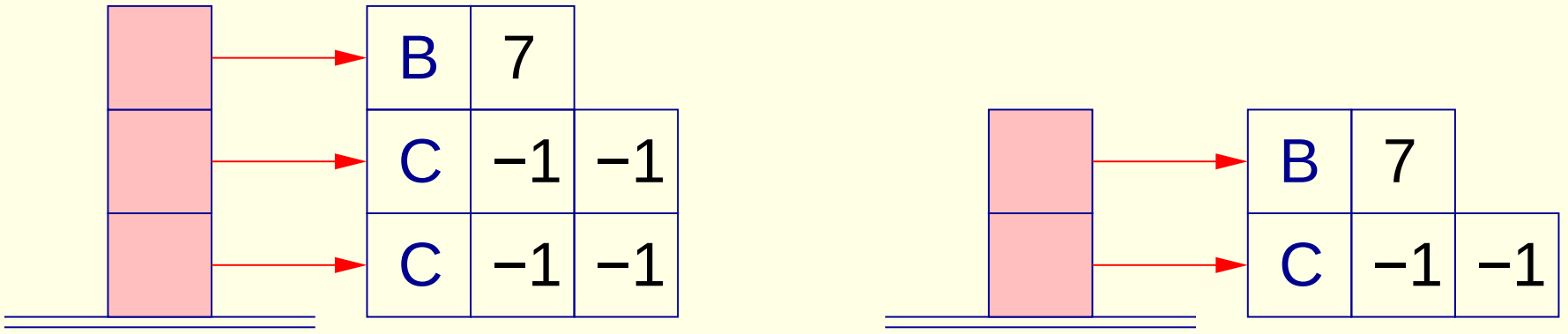
mkbasic



Optimiseerimine II: Sulundid

```
0  alloc 2          2  loadc 7         2  pushloc 1
2  pushloc 0        3  mkbasic         3  eval
3  rewrite 2        3  rewrite 1       3  slide 2
```

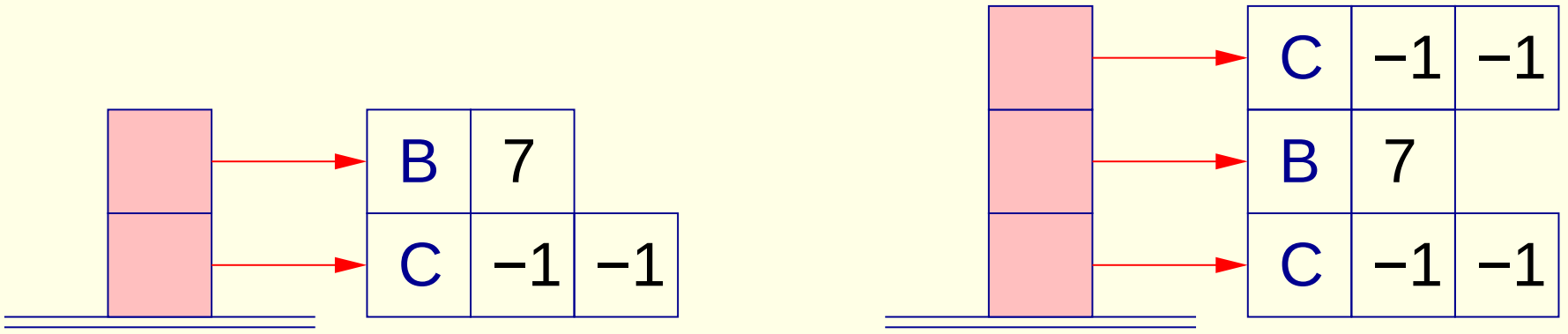
rewrite 1



Optimiseerimine II: Sulundid

```
0  alloc 2          2  loadc 7         2  pushloc 1
2  pushloc 0        3  mkbasic         3  eval
3  rewrite 2        3  rewrite 1       3  slide 2
```

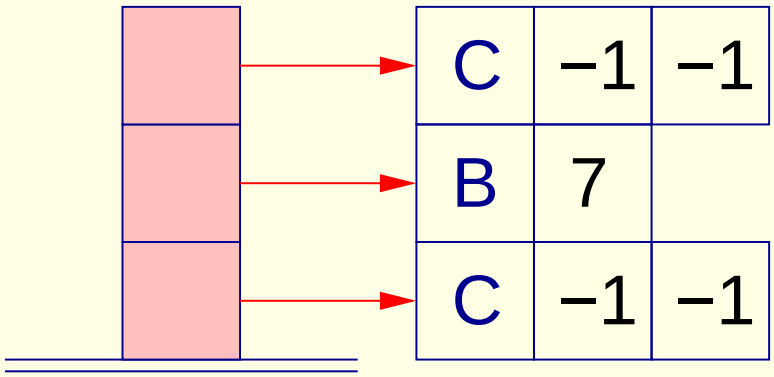
pushloc 1



Optimiseerimine II: Sulundid

```
0  alloc 2          2  loadc 7         2  pushloc 1
2  pushloc 0        3  mkbasic         3  eval
3  rewrite 2        3  rewrite 1       3  slide 2
```

eval



Segmentation Fault!!

Optimiseerimine II: Sulundid

Ilmselt polnud optimiseerimine päris korrektne!

Probleem:

Muutuja x seoti muutuja y väärtusega enne, kui see oli tegeliku väärtusega asendatud!!

Lahendus:

tsüklilised definitsioonid: mitte lubada definitsioone kujul

letrec $a = b;\ \ldots;\ b = a$ **in** $\ldots$

atsüklilised definitsioonid: järjestada definitsioonid ringi nii, et teistest defineeritavatest muutujatest sõltuvad definitsioonid oleks tagapool.

Optimiseerimine II: Sulundid

Funktsioonid:

Funktsioonid on väärtused, mida edasi väärtustata. Selle asemel, et genereerida kood, mis konstrueerib F-objekti sulundi, võime otse konstrueerida F-objekti.

Seega:

$$\text{code}_C\ (\mathbf{fn}\ x_0, \ldots, x_{k-1} \Rightarrow e)\ \rho\ \text{sd} \quad = \quad \text{code}_V\ (\mathbf{fn}\ x_0, \ldots, x_{k-1} \Rightarrow e)\ \rho\ \text{sd}$$

Kogu programmi transleerimine

Abstraktse masina olek enne programmi käivitamist:

$$PC = 0 \qquad SP = FP = GP = -1$$

Programm (so. avaldis) *e* ei tohi sisaldada *vabu muutujaid*.

Genereeritakse kood, mis väärtustab avaldise *e* ja seejärel lõpetab masina töö käsuga `halt`:

$$\begin{aligned} \text{code } e \quad = \quad & \text{code}_V\ e\ \emptyset\ 0 \\ & \texttt{halt} \end{aligned}$$

Kogu programmi transleerimine

- Toodud transleerimisskeemid genereerivad ”*spageti-koodi*”.
- Põhjus: funktsioonikehade ja sulundite kood paigutatakse vahetult pärast `mkfunval` ja `mkclos` käske, hüpates üle selle koodi.
- Alternatiiv: panna see kood kuhugi mujale; näiteks pärast `halt` käsku:

  Eelis: saame lahti hüpetest pärst `mkfunval` ja `mkclos` käske.

  Puudus: transleerimisskeemid muutuvad keerulisemaks.
- Lahendus: eemaldame ”spageti-koodi” koodi genereerimisele järgnevas optimiseerimisfaasis.

Kogu programmi transleerimine

Näide: **let** $a = 17;\ f = \textbf{fn}\ b \Rightarrow a + b\ \textbf{in}\ f\ 42$

”Spageti-koodi” elimineerimisel saame:

```
0  loadc 17
1  mkbasic
1  pushloc 0
2  mkvec 1
2  mkfunval A
2  mark B
5  loadc 42
6  mkbasic
6  pushloc 4
7  eval
7  apply
3  B: slide 2
1     halt
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
```