Die Funktion Down() dekrementiert den Zähler.

Rutscht dieser dadurch ins Negative, wird **wait** aufgerufen:

```
void Down (Sema * s) {
        Mutex *me;
        me = s→me;
        lock (me);
        s→count– –;
        if (s→count < 0)    wait (s→cv,me);
        unlock (me);
    }
```

Die Übersetzung liefert für den Rumpf:

```
alloc 1      loadc 2      add          loadc 1
loadr 1      add          store        add
load         load         loadc 0      load
storer 2     loadc 1      le           wait
lock         sub          jumpz A  A:  loadr 2
             loadr 1      loadr 2      unlock
loadr 1      loadc 2      loadr 1      return
```

Die Funktion Up() inkrementiert den Zähler wieder.

Ist dieser danach noch nicht positiv, gibt es wartende Threads, von denen einer ein Signal erhält:

```
void Up (Sema * s) {
        Mutex *me;
        me = s→me;
        lock (me);
        s→count++;
        if (s→count ≤ 0)   signal (s→cv);
        unlock (me);
    }
```

Die Übersetzung liefert für den Rumpf:

```
alloc 1      loadc 2      add          loadc 1
loadr 1      add          store        add
load         load         loadc 0      load
storer 2     loadc 1      le           signal
lock         add          jumpz A  A:  loadr 2
             loadr 1                   unlock
loadr 1      loadc 2      loadr 1      return
```

49 Stack-Management

Problem:

- Alle Threads leben in einem gemeinsamen Speicher.
- Jeder Thread benötigt (konzeptuell) einen eigenen Stack.

1. Idee:

Allokiere für jeden neuen Thread einen festen Speicherbereich auf der Halde!

Dann implementieren wir:

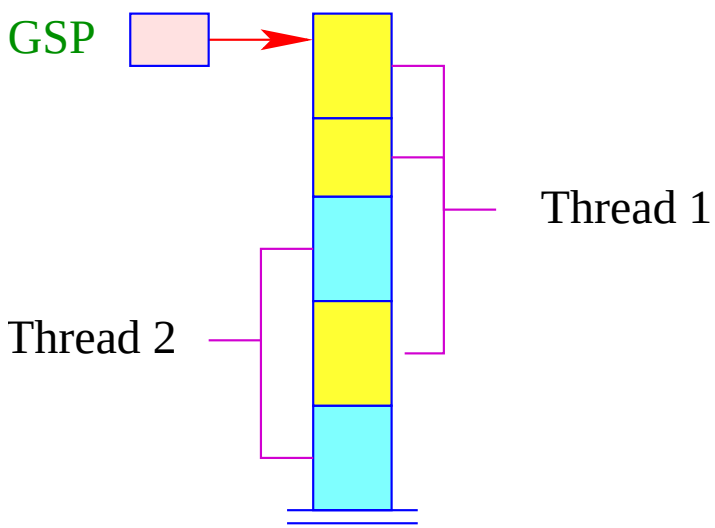
```
void *newStack() { return malloc(M); }
void freeStack(void *adr) { free(adr); }
```

Problem:

- Manche Threads brauchen viel, manche weniger Stack-Space.
- Evt. ist der nötige Platz statisch gar nicht bekannt :-(

2. Idee:

- Verwalte sämtliche Keller zusammen in einem Frame-Heap FH :-)
- Sorge dafür, dass der Platz im Rahmen zumindest ausreicht zum Abarbeiten des aktuellen Funktionsaufrufs.
- Ein globaler Stack-Pointer GSP gibt an, wieviel Platz bereits vergeben ist...



Allokation und De-Allokation eines Frames erfolgt mittels Laufzeitfunktionen:

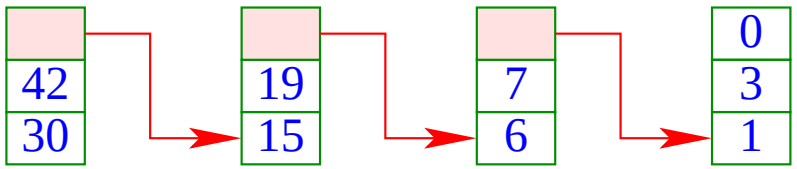
```
int newFrame(int size) {
    int result = GSP;
    GSP = GSP+size;
    return result;
    }

void freeFrame(int sp, int size);
```

Achtung:

Der frei zu gebende Block kann im Innern des Stacks liegen :-(

$\Longrightarrow$

Wir verwalten eine Liste der freigegebenen Abschnitte des Stacks :-)

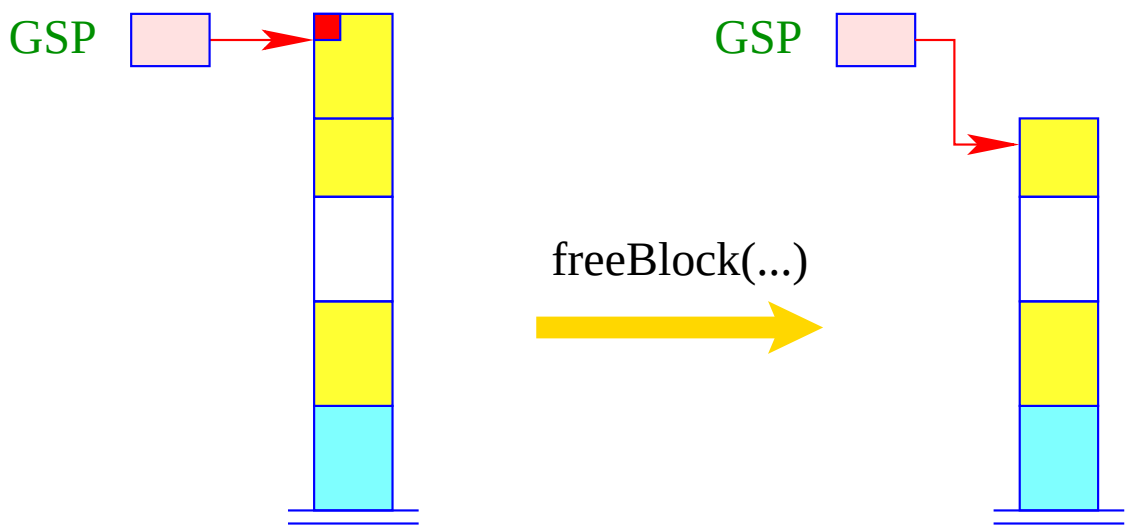


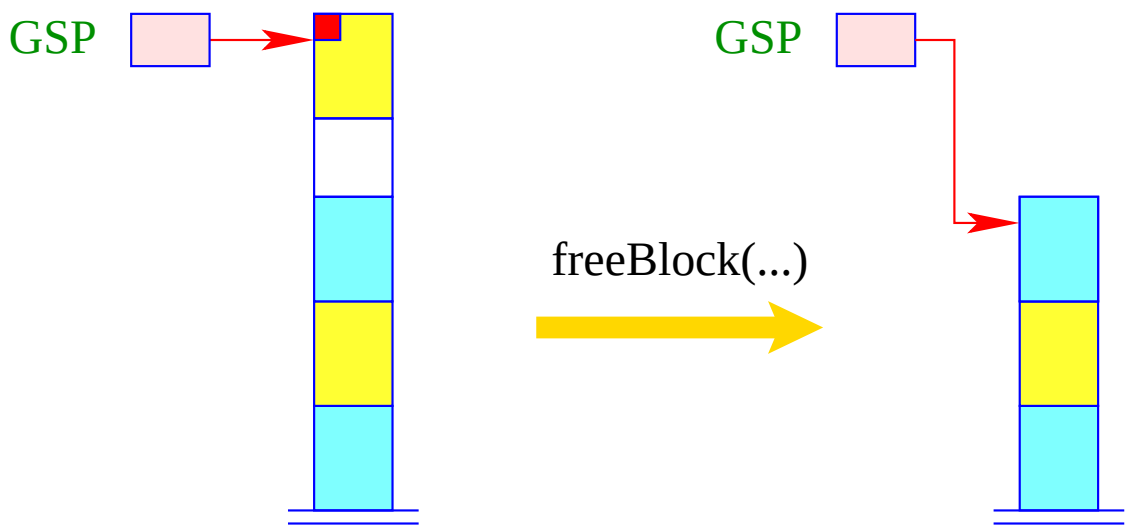
Diese Liste unterstützt eine Funktion

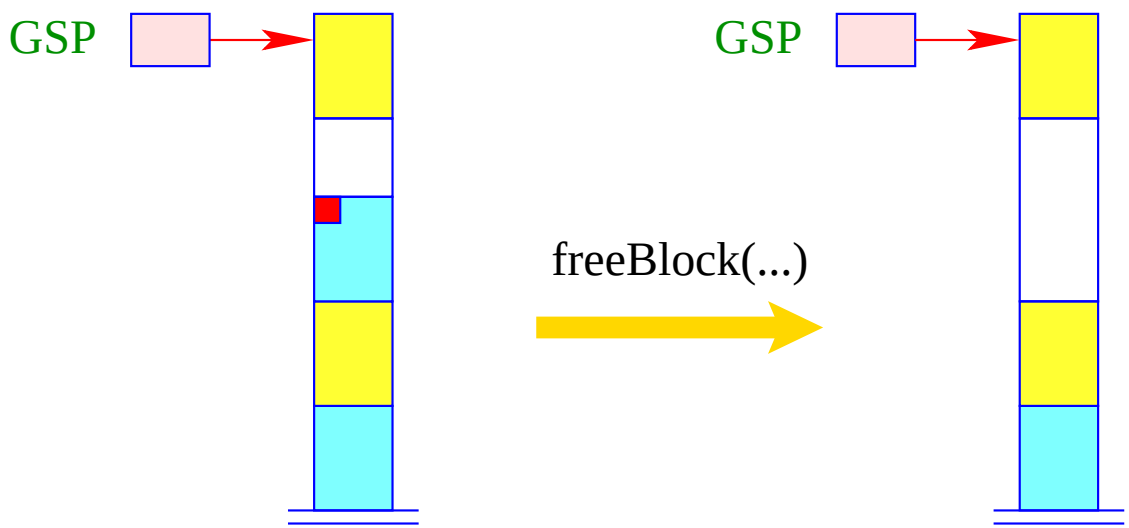
```
void insertBlock(int max, int min)
```

die es gestattet, einzelne Blocks frei zugeben.

- Liegt der Block am oberen Ende des Stacks geben wir ihn sofort frei;
- ... Wie den darunter liegenden Abschnitt – falls dieser bereits de-allokiert ist.
- Liegt er im Innern, verschmelzen wir ihn mit angrenzenden freien Blöcken:


GSP
freeBlock(...)
GSP


GSP
GSP
freeBlock(...)


GSP
freeBlock(...)
GSP

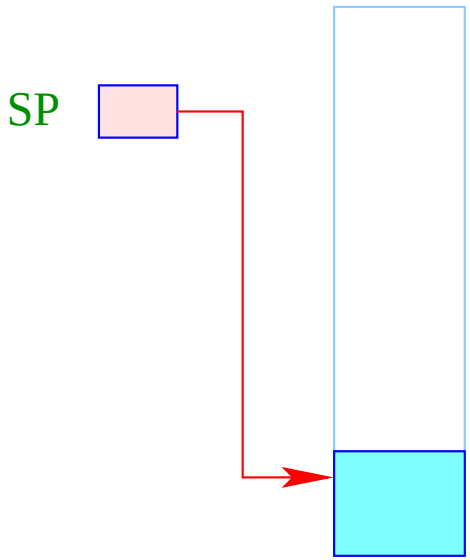
Ansatz:

Wir allokieren einen neuen Block für jeden Funktions-Aufruf ...

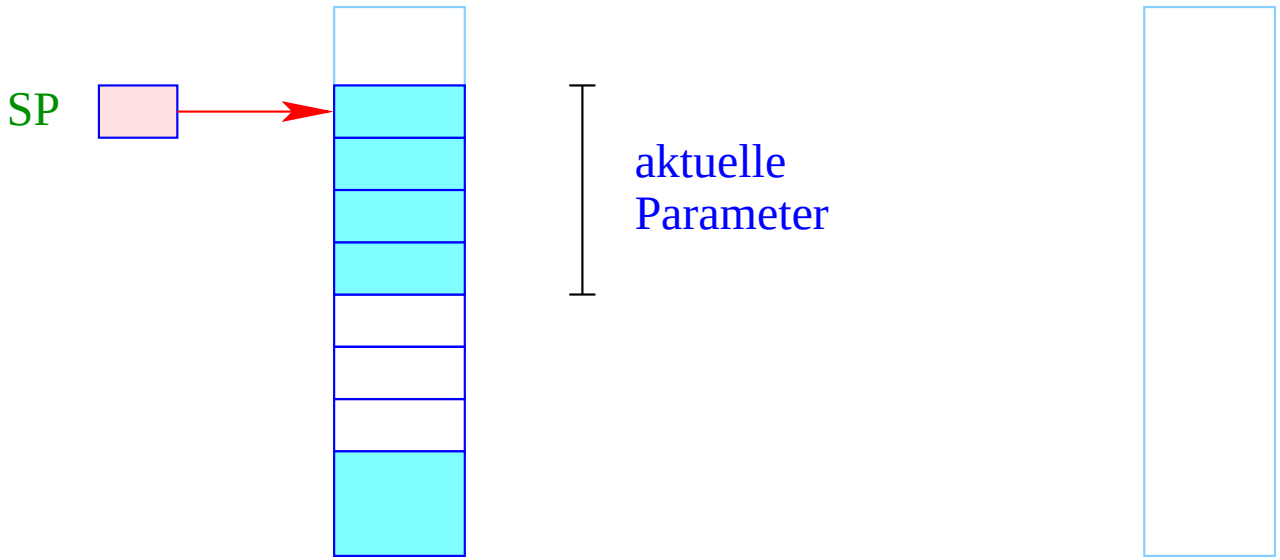
Problem:

Bei Anforderung des neuen Blocks vor dem Aufruf ist der Speicherbedarf der aufgerufenen Funktion noch gar nicht bekannt :-(

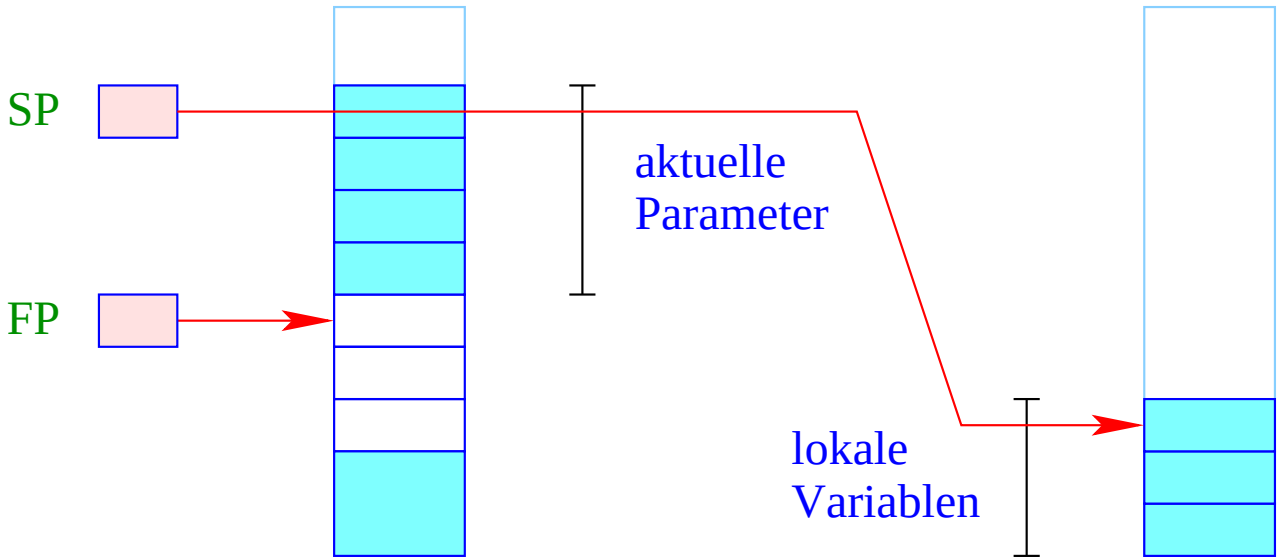
$\Longrightarrow$ Wir können den neuen Block erst bei Betreten des Funktions-Rumpfs anfordern!



Organisatorische Zellen wie aktuelle Parameter müssen noch im alten Block angelegt werden ...



Bei Betreten der neuen Funktion allokieren wir auch den neuen Block ...



Insbesondere liegen jetzt die lokalen Variablen im neuen Block ...

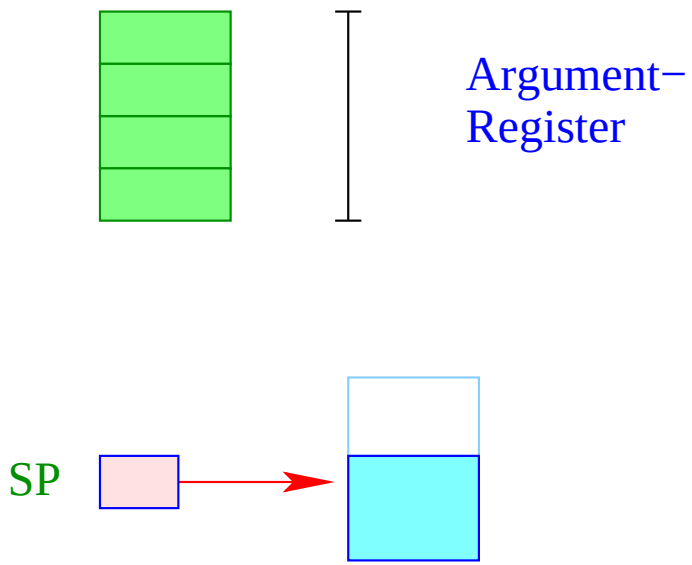
$\Longrightarrow$ Wir adressieren ...

- die formalen Parameter relativ zum Frame-Pointer;
- die lokalen Variablen relativ zum Stack-Pointer :-)

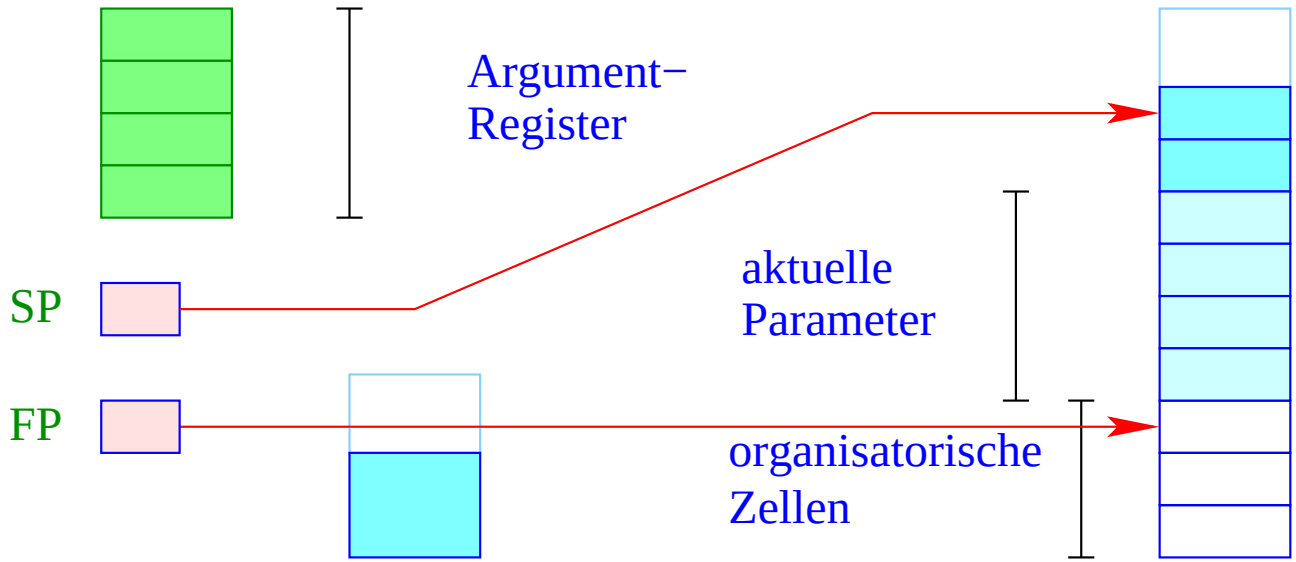
$\Longrightarrow$ Wir müssen die gesamte Code-Erzeugung umstellen ... :-(

Ausweg:

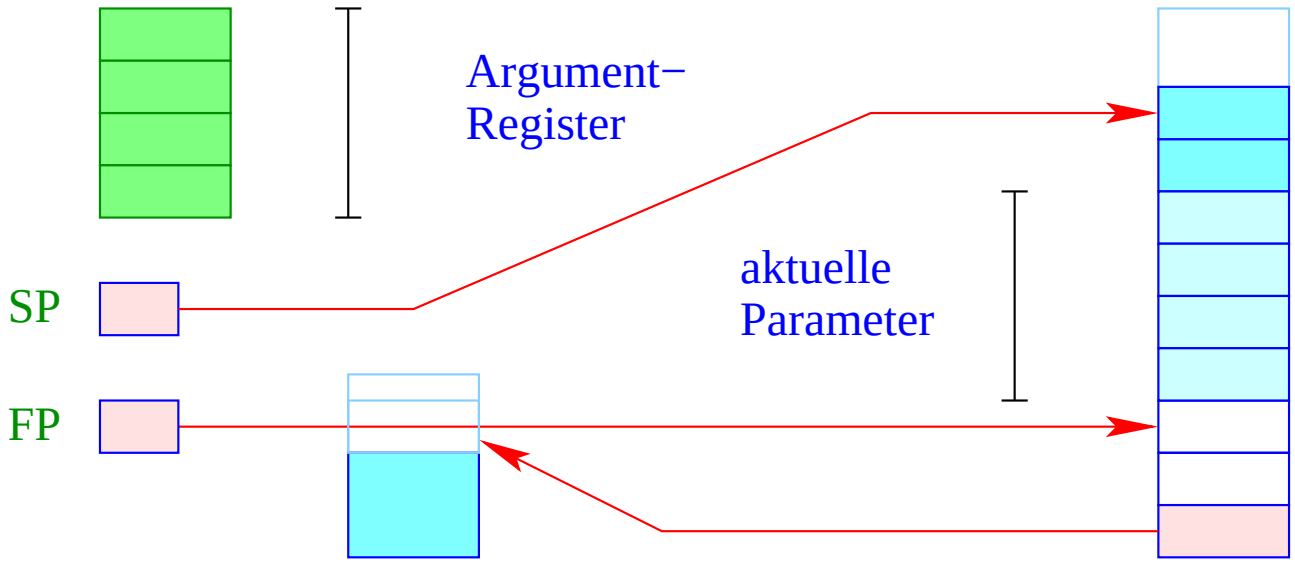
Übergabe von Parametern in Registern ... :-)



Die Werte der aktuellen Parameter werden vor Anlegen des neuen Keller-Rahmens ermittelt.



Der gesamte Rahmen wird im neuen Block angelegt – inclusive Platz für die aktuellen Parameter.



Im neuen Block müssen wir allerdings uns auch den alten SP (evt. +1) merken, damit das Ergebnis korrekt zurück geliefert werden kann ...

3. Idee: Hybrid-Lösung

- Für die ersten k Threads lege jeweils einen eigenen Speicherbereich an!
- Für alle weiteren benutze reihum einen der bereits vorhandenen ...

- Für wenige Threads extrem einfach und effizient;
- Für viele Threads amortisierte Speicher-Ausnutzung ...