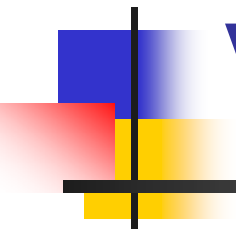
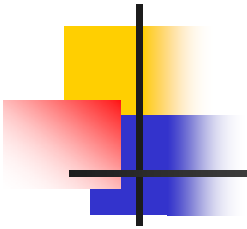
Einfach benutzbare mehrdimensionale Felder variabler Größe in C/C++



Seminar 2005

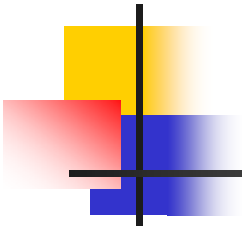
Annika Schiller

ZAM



Inhalt

- Motivation
- Library VarArray
  - Anwendung
  - Implementierung
- Ausblick



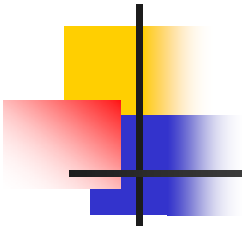
Motivation

Bibliothek VarArray

- steht für variable size arrays

Ziel:

- C-Bibliothek zur Bereitstellung dynamischer mehrdimensionaler Felder
- einfache Benutzung
- erweiterte Funktionalität



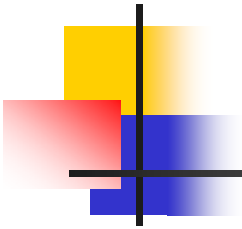
Motivation

Was sind die Nachteile der Standard-C/C++ Felder?

- konstante Dimensionslängen
- keine Parameter möglich, an die Felder unterschiedlicher Größe übergeben werden können

  (Ausnahme: 1. Dimension)

  Bsp.: `void transpose ( int m[][10] );`



Motivation

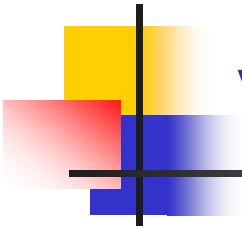
Beispiel: Größe eines Vektors zur Laufzeit festlegen

```
#include <stdlib.h>

void f (int veklen)
{
   /* unzulässig:
   int v[veklen];
                /* der gewünschte Effekt wird        */
                /* erreicht durch:                   */
   int * v = (int *) malloc(veklen * sizeof(int));
   ...
   for (i = 0; i < veklen; i++)
      v[i] = ...
   ...
   free(v);
}
```

VarArray: Anwendung

Jetzt mit VarArray:

```
#include “var_array.h“

void f (int veklen)
{
   vaArray_1d(int) v = vaCreate_1d(veklen,int,NULL);
   ...
   for (i = 0; i < veklen; i++)
      v[i] = ...
   ...
   vaDelete(v);
}
```



VarArray: Anwendung

Array Typen

- werden durch Makros angegeben:

  `vaArray_1d(`*`Basistyp`*`)`

  `vaArray_2d(`*`Basistyp`*`)`

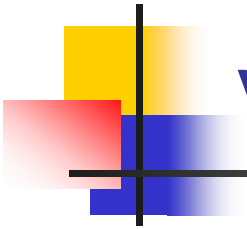
  ...

  `vaArray_5d(`*`Basistyp`*`)`

- sind Zeigertypen

Beispiel:

`vaArray_1d(int)` expandiert zu `int *`



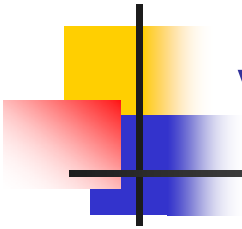
VarArray: Anwendung

- Nachteil:

```
vaArray_1d(int) a, b; /* expandiert zu:     */
int * a, b;           /* b ist kein Pointer */
```

- Abhilfe:

```
typedef vaArray_1d(int) int_vektor_t;
int_vektor_t a, b;    /* b ist ein Pointer */
```



VarArray: Anwendung

Vereinbarung einer 2-dimensionalen Matrix:

```
/* float m0[dim1][dim2]; */
vaArray_2d(float) m = vaCreate_2d(dim1,dim2,float,NULL);
```

Eigenschaften von m entsprechen Standard C-Matrix:

- m kann zweifach indiziert werden, um auf ein Element zuzugreifen:

  m[i][j]

- bei Funktionsaufruf f(m): Call by Reference
- m[i] entspricht einer Zeile der Matrix
- m[i] ist ein konstanter Zeiger

VarArray: Anwendung

Unterschiede:

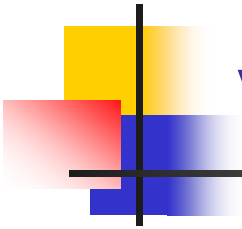
- Name des Feldes
  - C: Zeiger auf den Anfang des Feldes
  - VarArray: Zeiger auf ein Hilfselement

  ⇒ **Vorteil:**
  - Typ eines VarArrays enthält keine Größenangabe des Feldes
  - Löst Problem der Parametervereinbarung:

    ```
    vaArray_2d(float) transpose( vaArray_2d(float) m );
    ```

- VarArray muss explizit freigegeben werden:

  ```
  vaDelete(m);
  ```



VarArray:Anwendung

Struktur eines VarArray:

```
vaArray_3d(int) m;
```

- **m**: eindimensionaler Vektor, dessen Elemente 2-dimensionale VarArrays sind
- **m[i]**: vaArray_2d(int)
- **m[i][j]**: vaArray_1d(int)
- **m[i][j][k]**: int

VarArray: Anwendung

Zusätzliche Abfragefunktion:

int vaSize(*VarArray*, *dim*);

- liefert die Länge der Dimension *dim*
- -1, falls eine Dimension *dim* nicht existiert

VarArray: Anwendung

Transponieren einer Matrix:

```
#include “var_array.h“

vaArray_2d(int) transpose(vaArray_2d(int) m)
{
   vaArray_2d(int) t = vaCreate_2d(vaSize(m,2),
                                   vaSize(m,1), int,NULL);
   int i, j;

   for (i = 0; i < vaSize(m,1); i++)
      for (j = 0; j < vaSize(m,2); j++)
         t[j][i] = m[i][j];

   return t;
}
```

VarArray: Anwendung

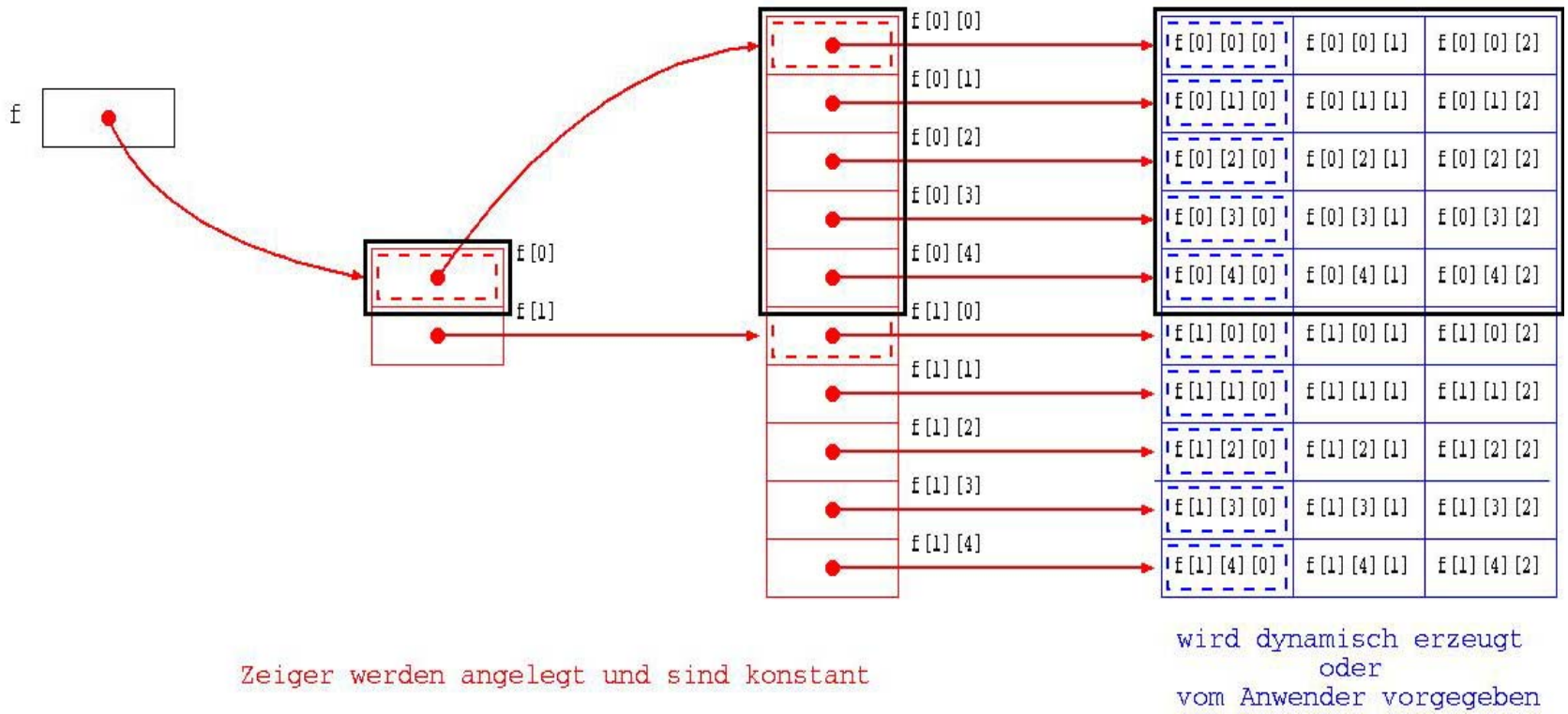
Übergabe eines C-Feldes an transpose?

```
int main (void)
{
   int a[2][3] = { {1,2,3},
                   {4,5,6} }
   vaArray_2d(int) t;

   vaArray_2d(int) va = vaCreate_2d(2,3,int,a);

   t = transpose(va);

   return 0;
}
```

VarArray: Implementierung

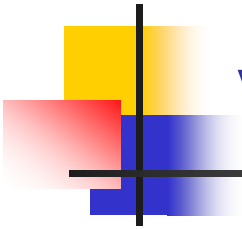
Beispiel:

```
vaArray_3d(float) f = vaCreate_3d(2,5,3,float,NULL);
```



Zeiger werden angelegt und sind konstant

wird dynamisch erzeugt
oder
vom Anwender vorgegeben



VarArray: Implementierung

Woher kennt vaSize() die Größe eines VarArrays a?

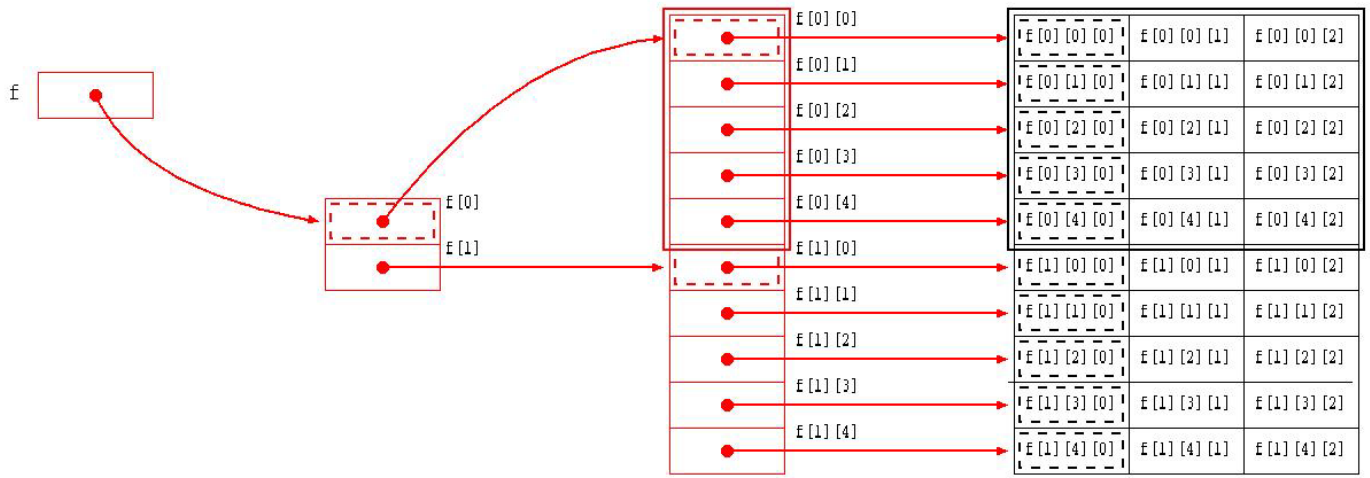
- `vaCreate_`*`n`*`d` kennt:
  - ✓ Dimensionslängen
  - ✓ Pointer a
- Idee: Informationen merken und vaSize() zugänglich machen

VarArray: Implementierung

Ursprüngliche Idee: Informationen in Hashtabelle speichern

- Pointer als Schlüssel verwenden

```
vaArray_3d(float) f = vaCreate_3d(2,5,3,float,NULL);
```



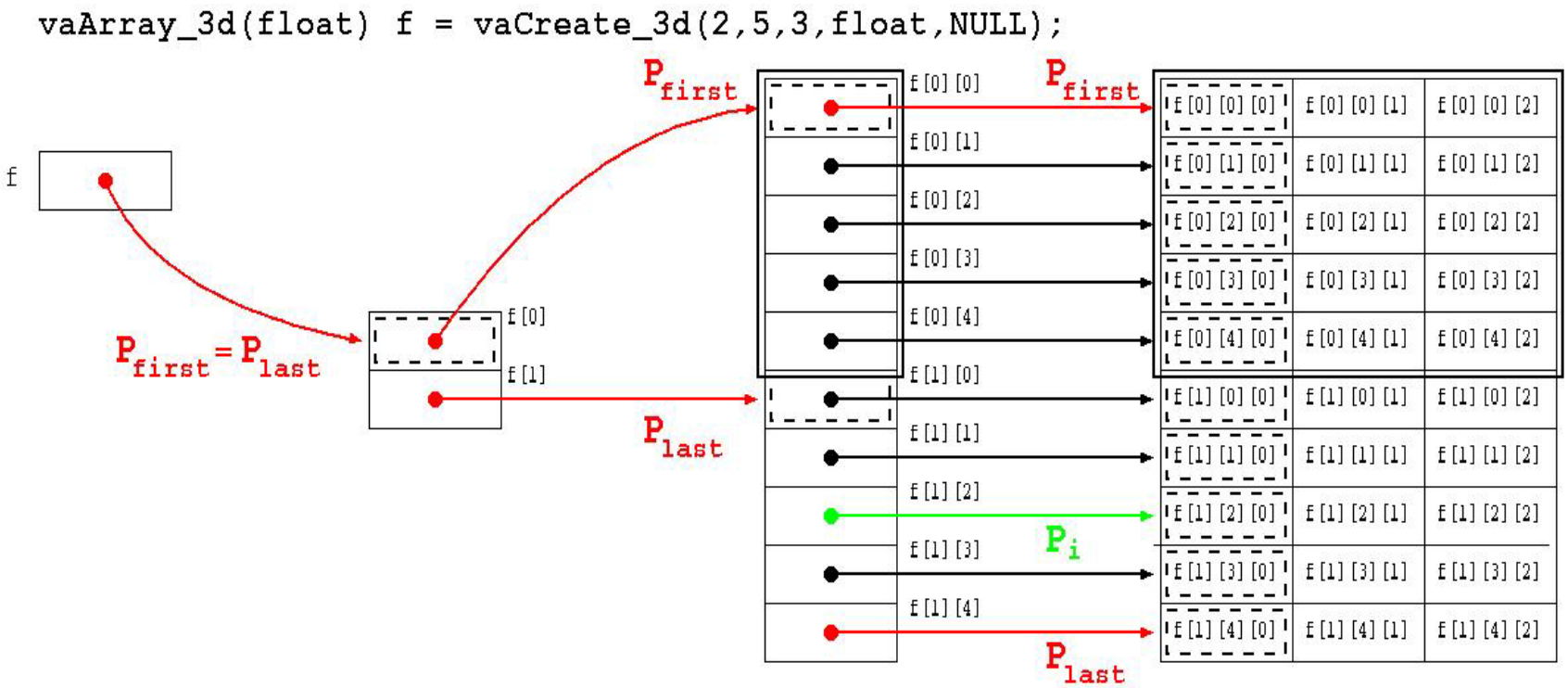

```
1       +       2       +       2*5     = 13    Pointer sind in Hashtabelle zu registrieren
```

- sehr viele Einträge
- Zeitkomplexität für a[n][n][n]: $0(n^2)$

$\Rightarrow$ Hoher Aufwand für wenige vaSize()-Aufrufe

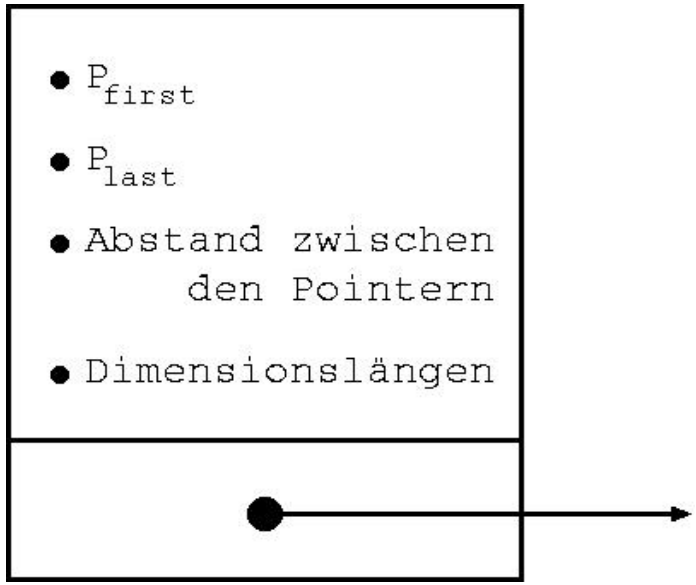
VarArray: Implementierung

neue Idee: Ausnutzen der Tatsache, dass VarArray einen zusammenhängenden Speicherbereich belegt



VarArray: Implementierung

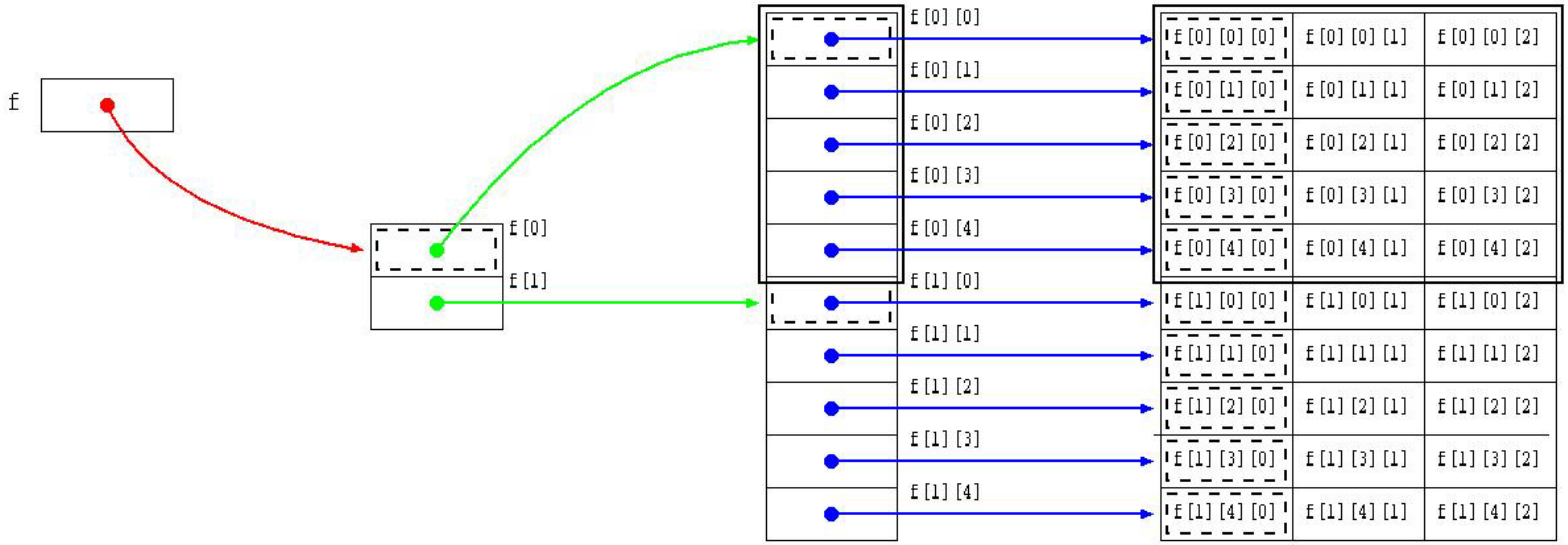
- zu merken:

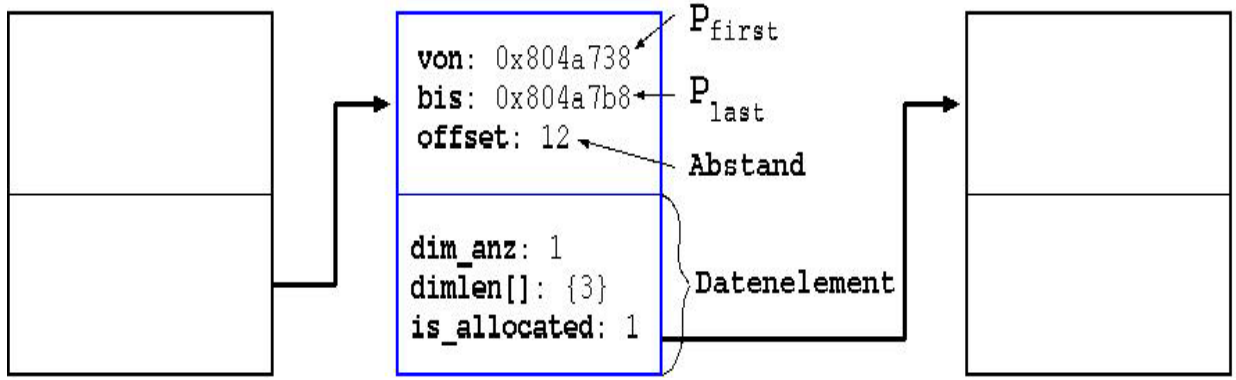


- Zeitkomplexität für a[n][n][n]: O(1)

VarArray: Implementierung

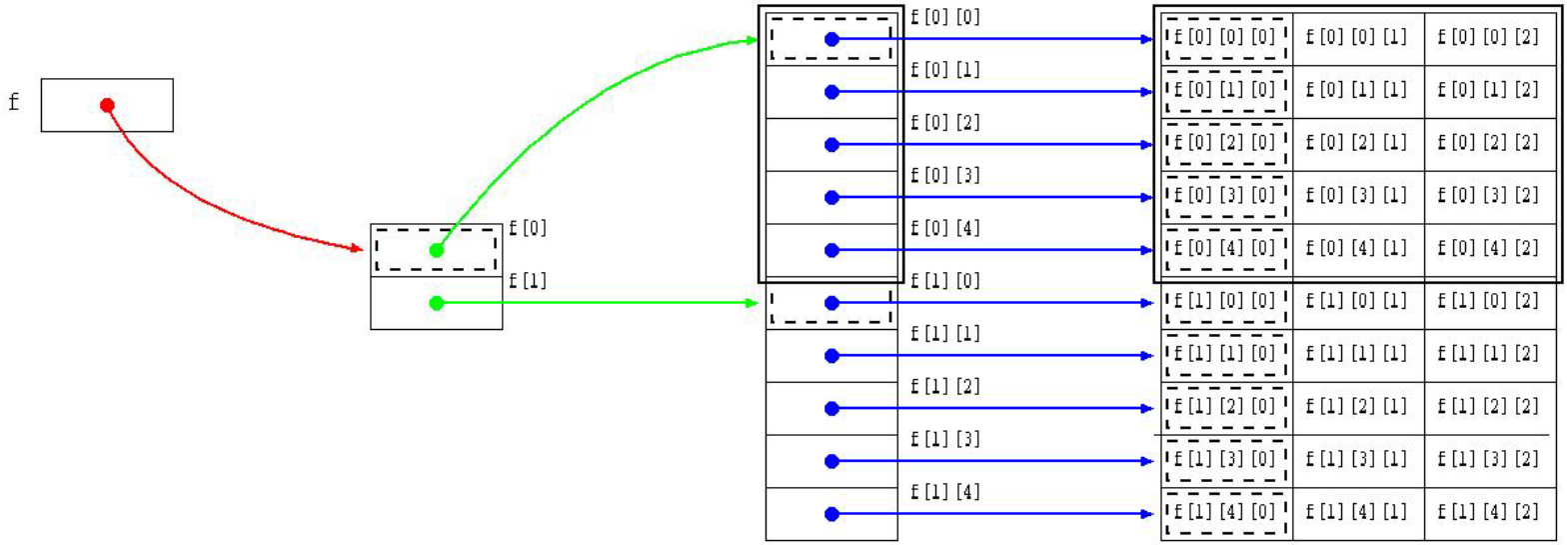
```
vaArray_3d(float) f = vaCreate_3d(2,5,3,float,NULL);
```

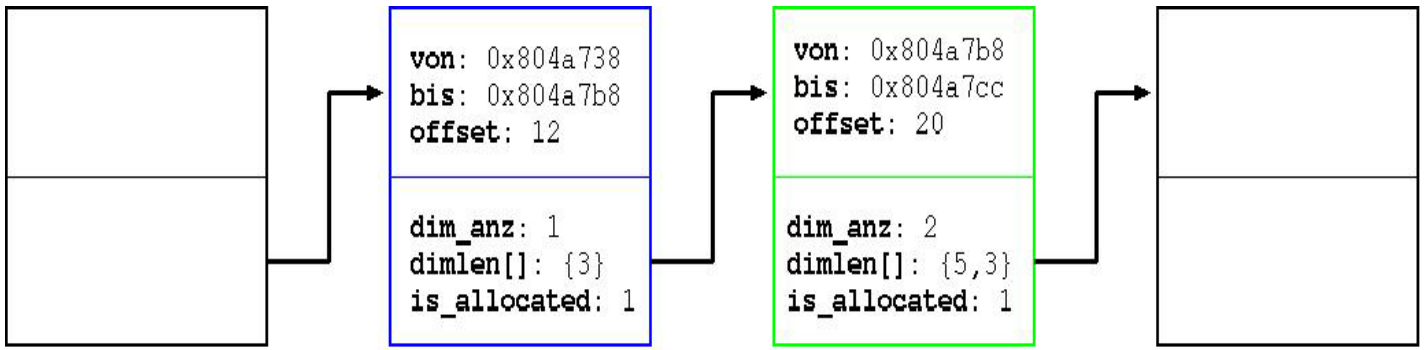




VarArray: Implementierung

```
vaArray_3d(float) f = vaCreate_3d(2,5,3,float,NULL);
```

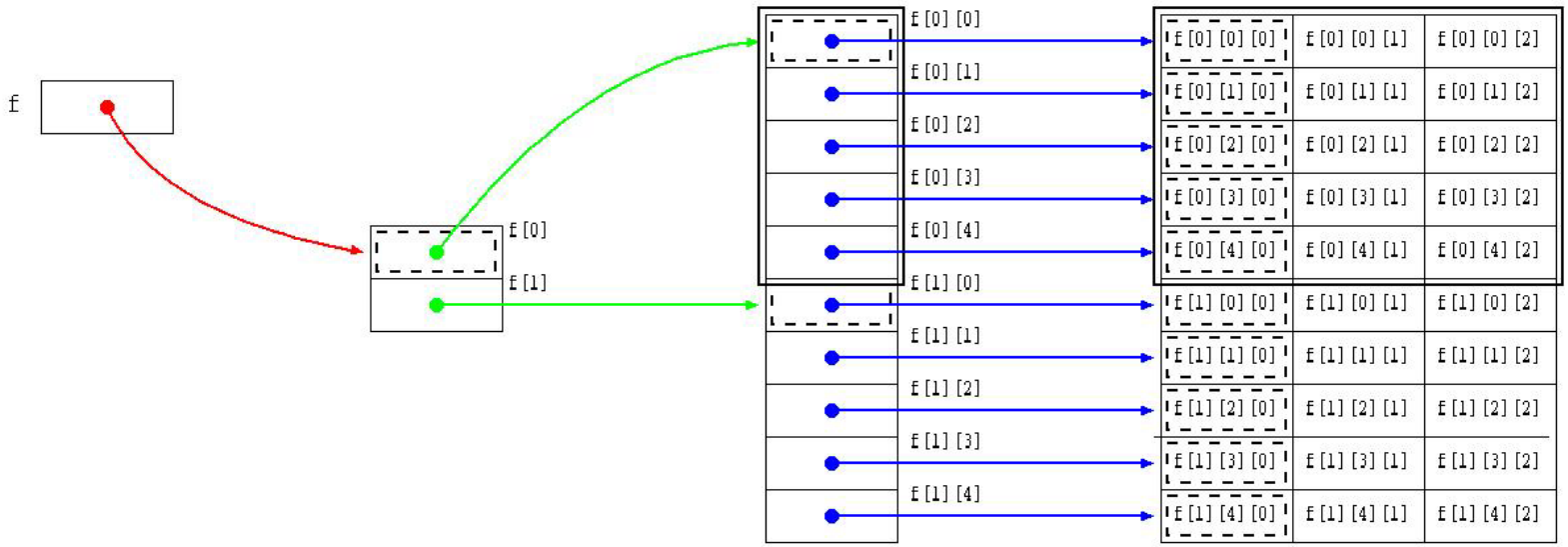


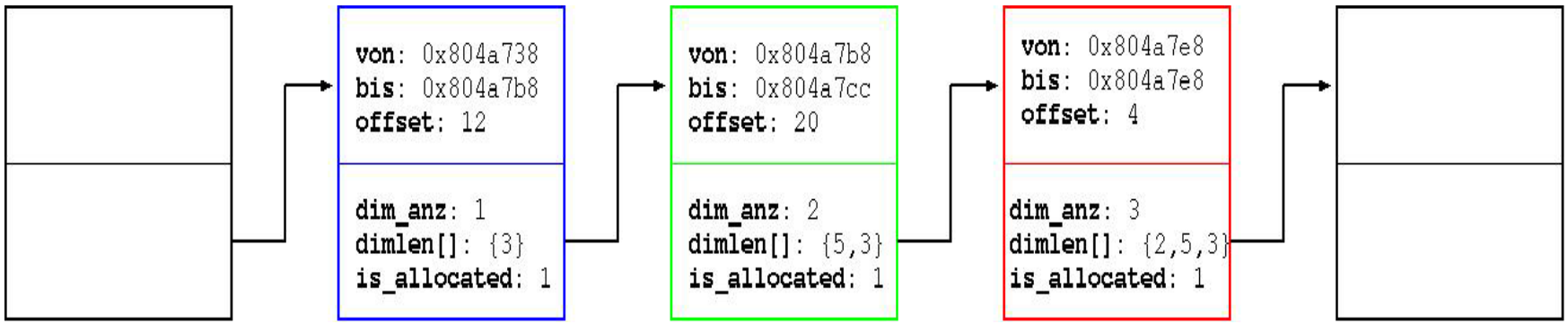


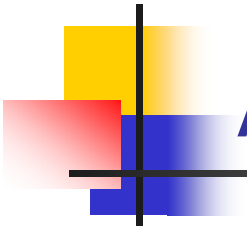
aufsteigend nach Adressen sortiert

VarArray: Implementierung

```
vaArray_3d(float) f = vaCreate_3d(2,5,3,float,NULL);
```







Ausblick

- Web-Seite erstellen
- Library in Source-Form zum Download bereitstellen
- C++: zusätzliche Templates für die Ein-/Ausgabe von VarArrays