Kapitel 7: Zusatzfolien Transformation

Array-Adressierung

- eindimensionales Array $a[u_1..o_1]$: $adr(a[i]) = adr(a[0]) + d * i$
- zweidimensionales Array $a[u_1..o_1, u_2..o_2]$: klassische zeilenorientierte Speicherung:

$a[u_1, u_2] \ldots a[u_1, o_2]$	$a[u_1 + 1, u_2] \ldots a[u_1 + 1, o_2]$	$\cdots$	$a[o_1, u_2] \ldots a[o_1, o_2]$

  $adr(a[i, j]) = adr(a[0, 0]) + d * (i * (o_2 - u_2 + 1) + j)$
- dreidimensionales Array $a[u_1..o_1, u_2..o_2, u_3..o_3]$: Sei $l_i = o_i - u_i + 1$
  $adr(a[i_1, i_2, i_3]) = adr(a[0, 0, 0]) + d * (i_1 * l_2 * l_3 + i_2 * l_3 + i_3)$
- Allgemein:

$$adr(a[i_1, \ldots, i_n]) = adr(a[0, \ldots, 0]) + d * \left(\sum_{\nu=1}^{n} \left(i_\nu * \prod_{\mu=\nu+1}^{n} l_\mu \right) \right)$$

Implementierung der Array-Adressierung

$$adr(a[i_1, \ldots, i_n]) = adr(a[0, \ldots, 0]) + d * \left(\sum_{\nu=1}^{n} \left(i_\nu * \prod_{\mu=\nu+1}^{n} l_\mu \right) \right)$$

effiziente Auswertung mit Hornerschema:

adr = i_1;
for k = 2 to n **do** {
    adr = adr * l_k;
    adr = adr + i_k;
}

Zwischencode durch „Ausrollen“ dieser Schleife

Beispiel (n=3, symbolischer Tripelcode):

load r, i1
mul r, l2
add r, i2
mul r, l3
add r, i3
mul r, d
add r, adr(a[0,0,0])

Kapitel 7: Zusatzfolien Transformation

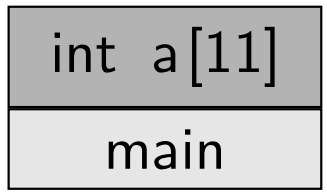
1. Array-Adressierung
2. Speicherzuweisung auf dem Stack
3. Zugriff auf nichtlokale Daten auf dem Stack
   - Static Links
   - Displays

Beispiel: Quicksort

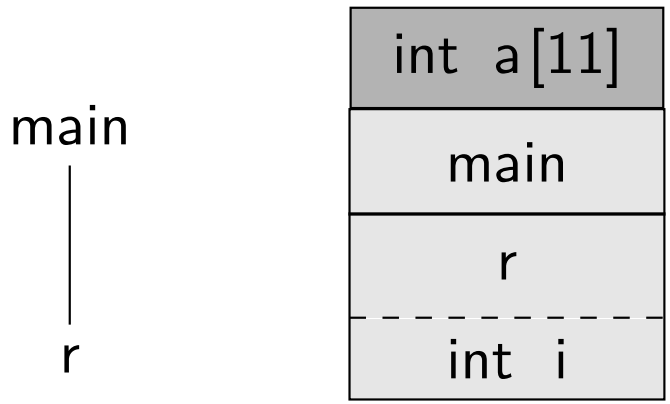
```
int a[11];
void r(void) { /* ... read integers into a[1] to a[9] ... */
    int i;
}
int p(int m, int n) {
    /* choose pivot element p, partition array into
     * { x | x < p }, { x | x >= p}; return position of p ... */
}
void q(int m, int n) {
    int i;
    if (n > m) {
        i = p(m, n); /* partition array */
        q(m, i-1); /* sort left part */
        q(i+1, n); /* sort right part */
    }
}
int main(void) {
    r(); a[0] = -INT_MIN; a[10] = INT_MAX;
    q(1,9);
}
```

Stack mit Activation Records

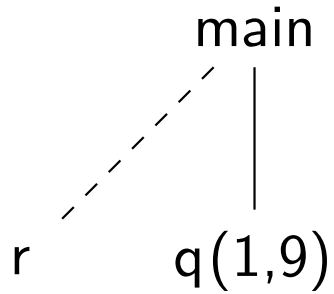
main

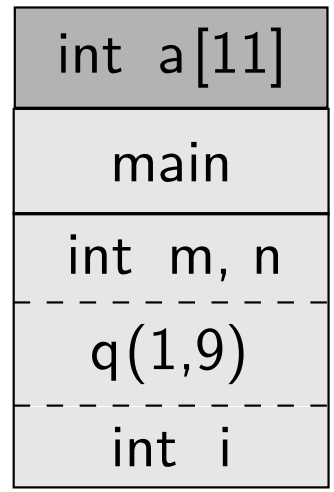


Stack mit Activation Records

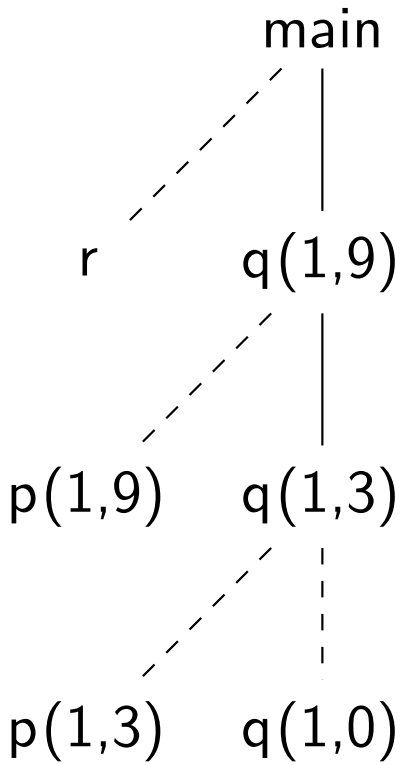


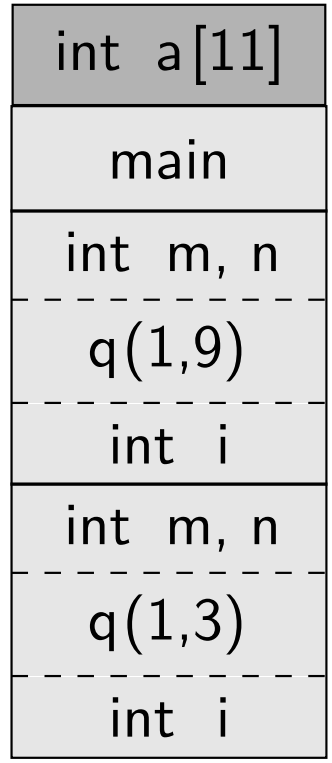
Stack mit Activation Records





Stack mit Activation Records





Kapitel 7: Zusatzfolien Transformation

1 Array-Adressierung

2 Speicherzuweisung auf dem Stack

3 Zugriff auf nichtlokale Daten auf dem Stack
- Static Links
- Displays

Skizze eines Programms mit geschachtelten Prozeduren

```
typedef int (*function_pointer)(int parameter);
void a(int x) {
    int b(function_pointer f) {
        /* ... */ int res = f(x); /* ... */
        return res;
    }
    void c(int y) {
        int d(int z) {
            /* ... */ int res = x*y + z; /* ... */
            return res;
        }
        /* ... */
        b(d);
        /* ... */
    }

    c(1);
}
```

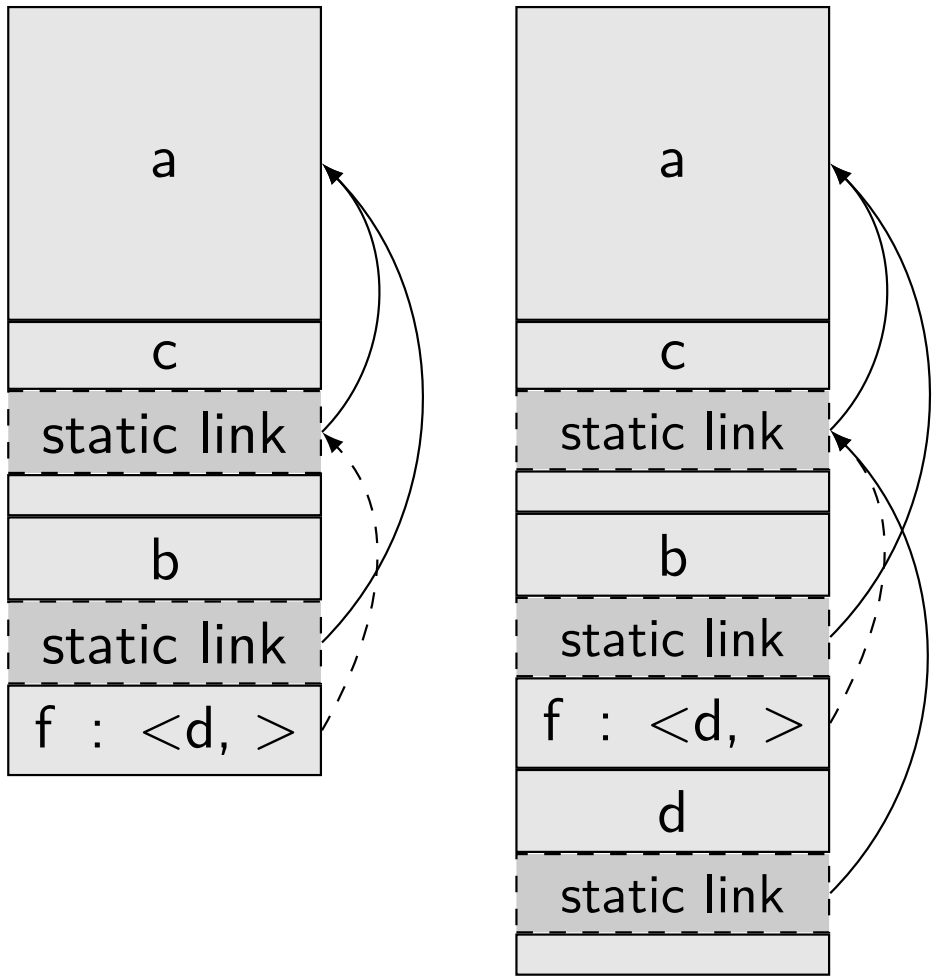
Prozedurparameter / Prozedurvariablen

Prozeduren als Parameter werden als Closure implementiert.

Definition Closure:
Paar aus <Funktion, statischer Umgebung>; erlaubt korrekten Zugriff auf nichtlokale Variablen.

Implementierung:
Paar <Einsprungadresse, Link zur statischen Umgebug>

Activation Record Stack mit Static Links



Displays

Problem: bei Zugriffen auf äußere Variablen aus tief verschachtelten Prozeduren müssen viele Static Links verfolgt werden.

Abhilfe: Displays (Dijkstra [1]).
Displays sind ein Hilfsarray d. Dieses enthält die Adressen der geschachtelten Activation Records. 1 Eintrag pro statische Tiefe:

$$
\begin{aligned}
d_i &= \mathrm{adr}(\mathrm{AR}_i) \\
AR_i &= \text{Activation Record des letzten Aufrufs der statischen Tiefe } i
\end{aligned}
$$

Vorteil: Schneller Zugriff auf nichtlokale Variablen.

Displays (1/2)

Implementierung:

- Bei Aufruf einer Funktion der Tiefe i:
  $d[i]$ sichern; $d[i]$ neu setzen auf $\mathrm{adr}(\mathrm{AR})$; Beim Rücksprung $d[i]$ wiederherstellen.
- Die Einträge des Displays werden in Registerbank organisiert.
  Der Rest mit static Link.
  Bei n Registern:

$$\left.\begin{array}{l} d[0] \\ d[1] \\ \vdots \\ d[n-1] \end{array}\right\} \text{in Registern}$$

$$\left.\begin{array}{l} d[n] \\ d[n+1] \\ \vdots \end{array}\right\} \text{static Links}$$

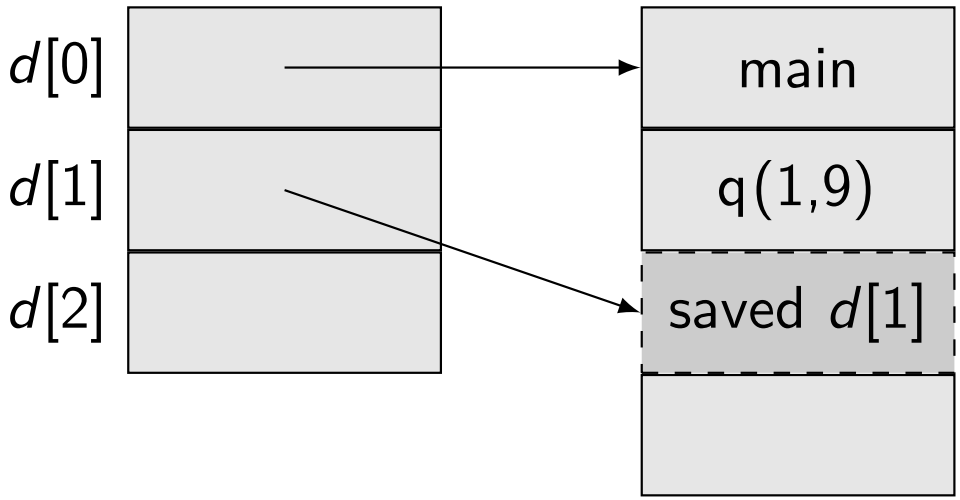
Displays (2/2)

- Falls nur wenige Register verfügbar (z.B. n=4):
  - verwende Static Link und zusätzlich d[0] für globale Variablen
  - d[3] (bzw Frame Register) für lokale Variablen (im aktuellen AR)
  - d[1/2] für statischen (Vor)Vorgänger der aktuellen Funktion

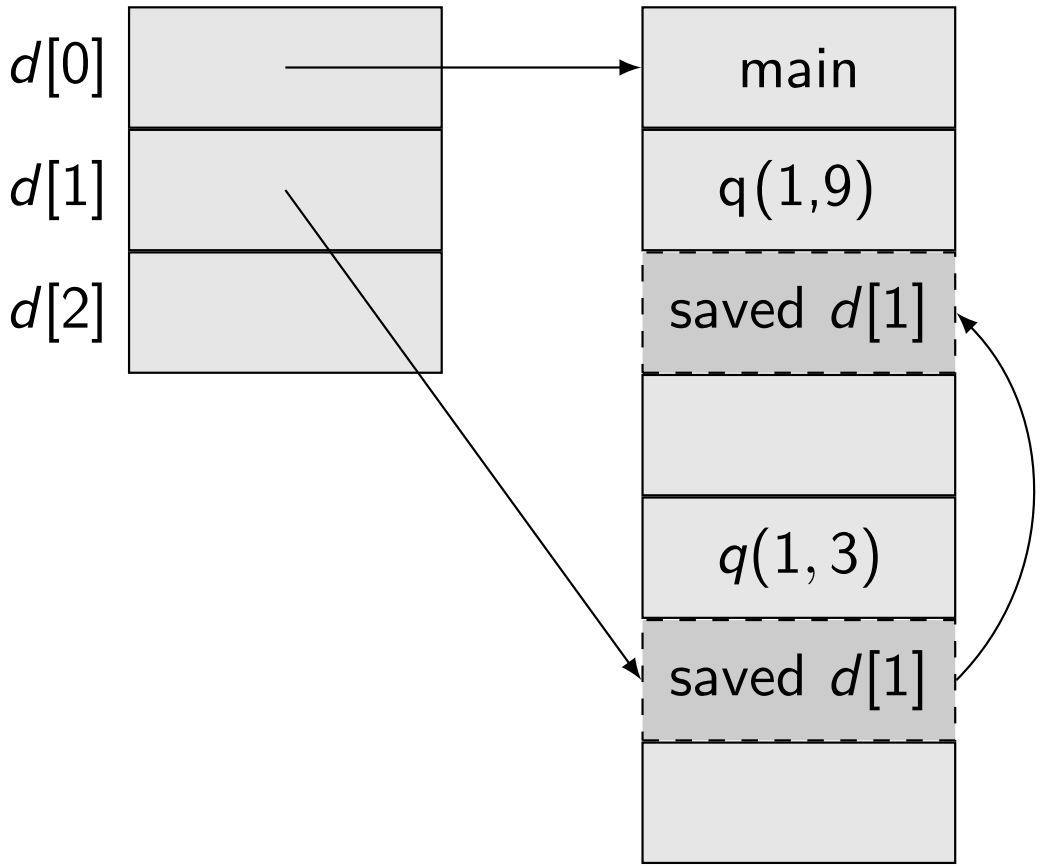
Quicksort mit verschachtelten Prozeduren

```
int main(void) {
    int a[11] = { INT_MIN,6,5,3,4,2,5,19,9,9,INT_MAX };
    void q(int l, int r) {
        void e(int e1, int e2) { /* exchange 2 elements */
            int t = a[e1]; a[e1] = a[e2]; a[e2] = t;
        }
        void p(int l, int r) {
            /* choose pivot element and partition */
            /* ... */ e(x, y); /* ... */
        }
        if (l > r)
            return;
        int i = p(m, n); /* partition array */
        q(l, i-1);
        q(i+1, r);
    }
    return 0;
}
```

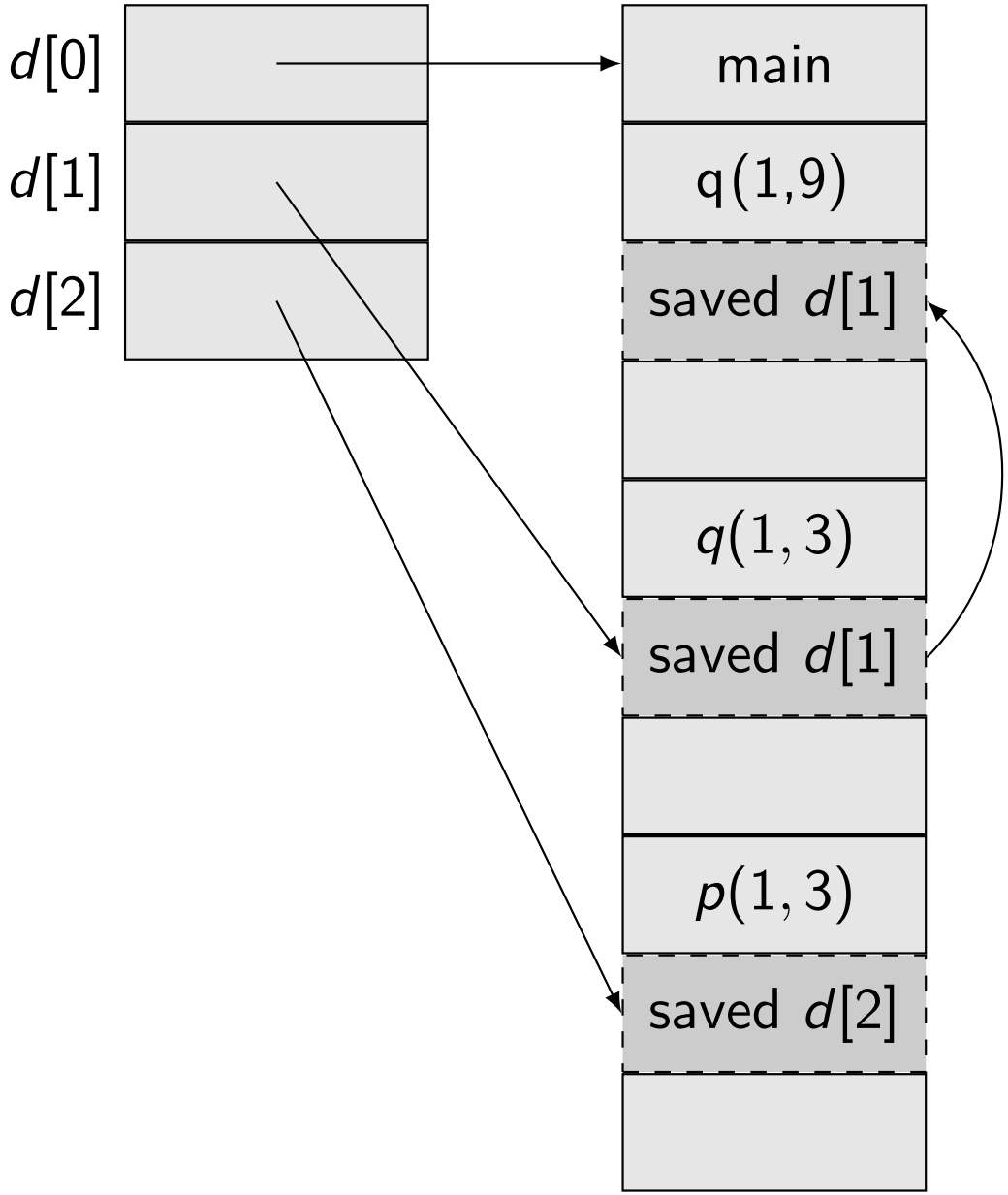
Displays



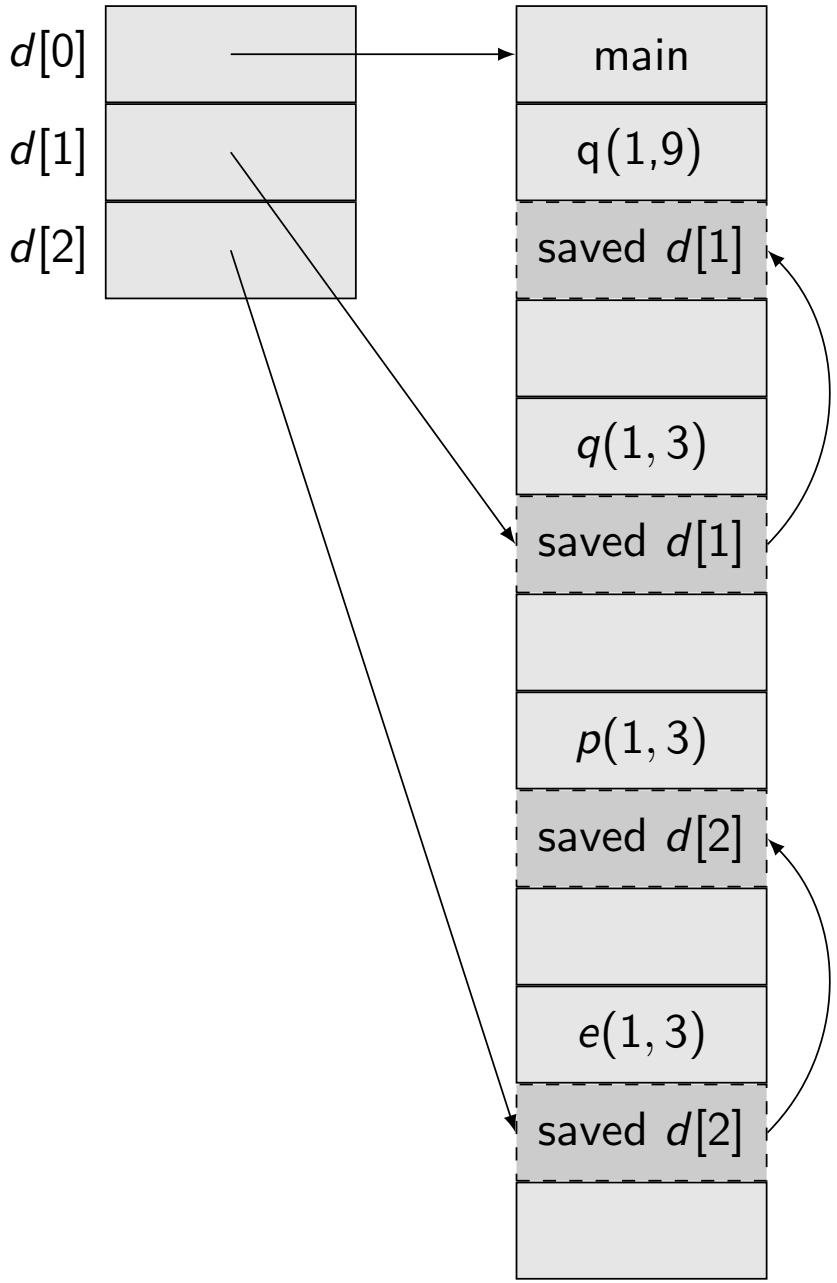
Displays



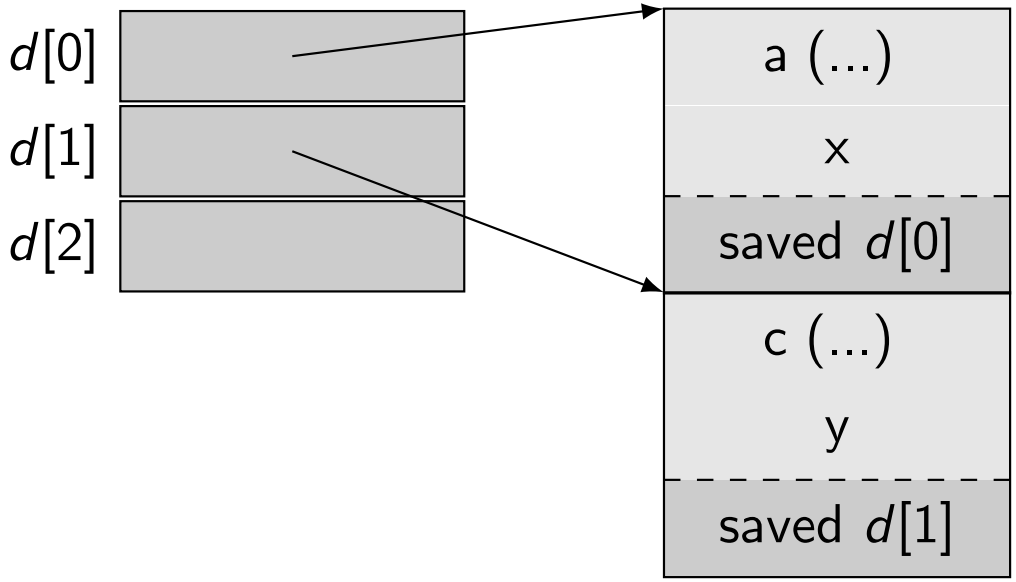
Displays



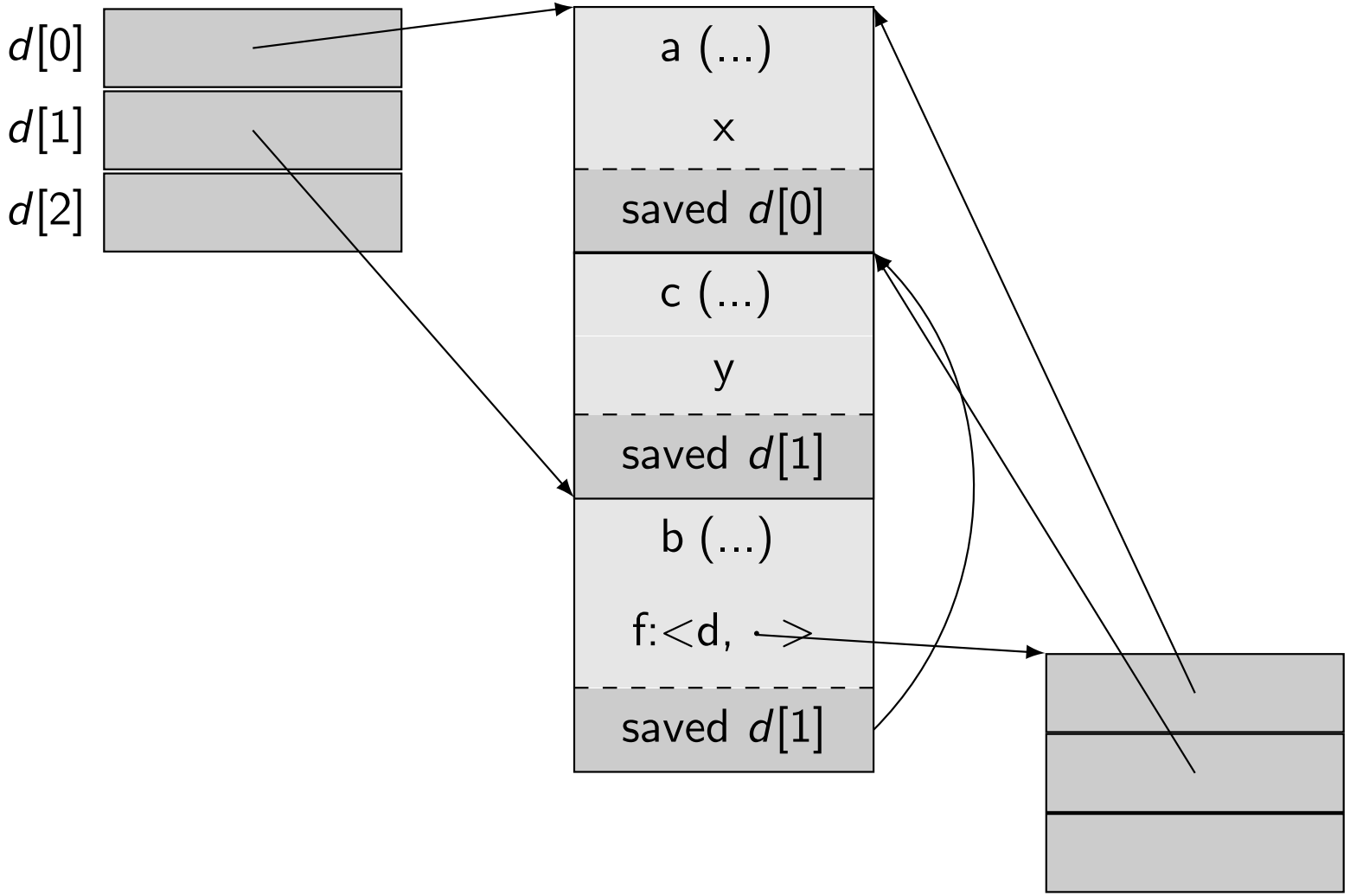
Displays


d[0]
d[1]
d[2]
main
q(1,9)
saved d[1]
q(1, 3)
saved d[1]
p(1, 3)
saved d[2]
e(1, 3)
saved d[2]

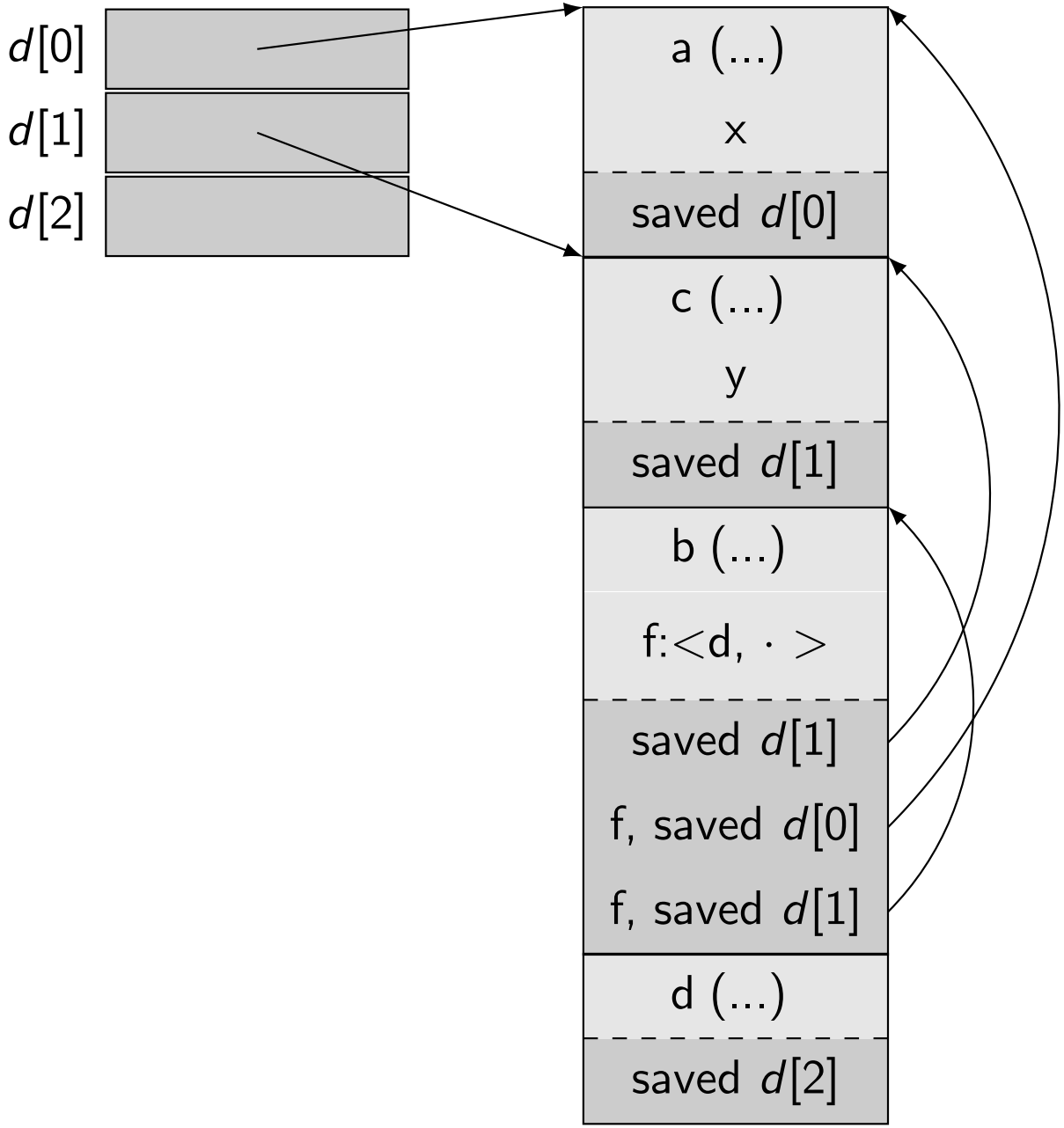
Prozedurparameter mit Displays



Prozedurparameter mit Displays



Prozedurparameter mit Displays



Literatur


[Dijkstra, 1960] E. W. Dijkstra.
Recursive Programming.
Numerische Mathematik 2, 312–318, 1960.