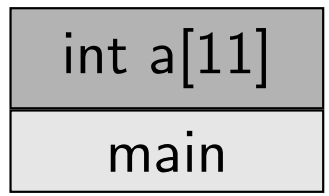
# Beispiel: Quicksort

```
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);
}
```

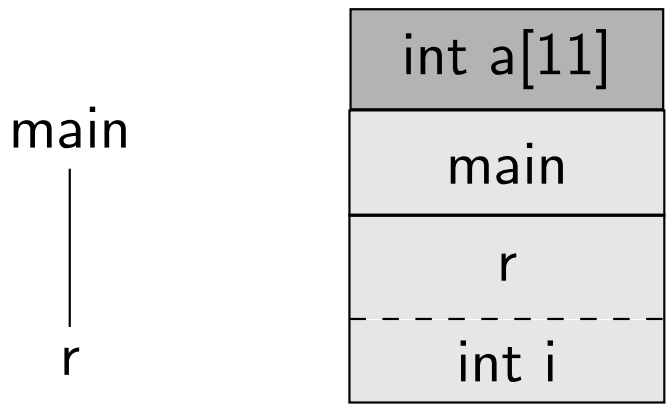


# Stack mit Activation Records

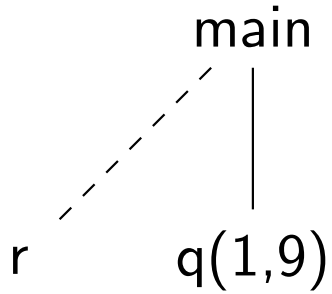
main

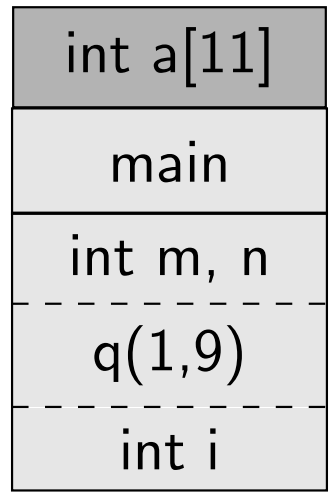


# Stack mit Activation Records

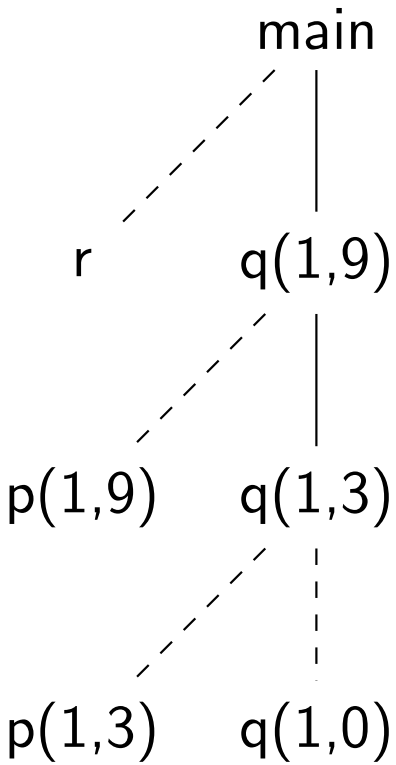


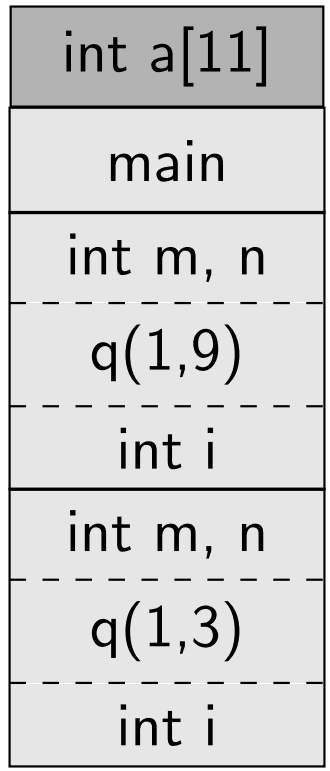
# Stack mit Activation Records



# Stack mit Activation Records



1 Speicherzuweisung auf dem Stack

2 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

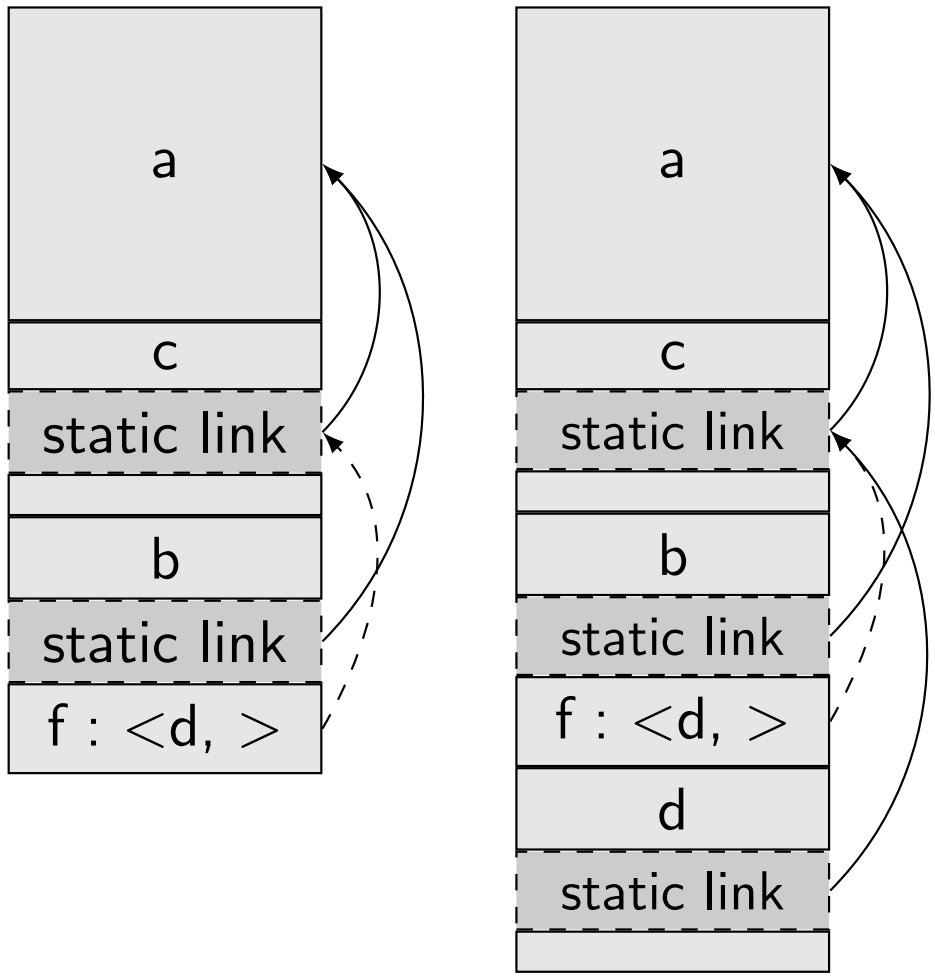
Prozeduren als Parameter werden als Closure implementiert.

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

Implementierung: Paar <Einsprungadresse, Link zur statischen Umgebug>



# Activation Record Stack mit Zugriffslinks



# Displays

Problem: bei Zugriffen auf äussere Variablen aus tief verschachtelten Prozeduren müssen viele Zugriffslinks 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

## 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:

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

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

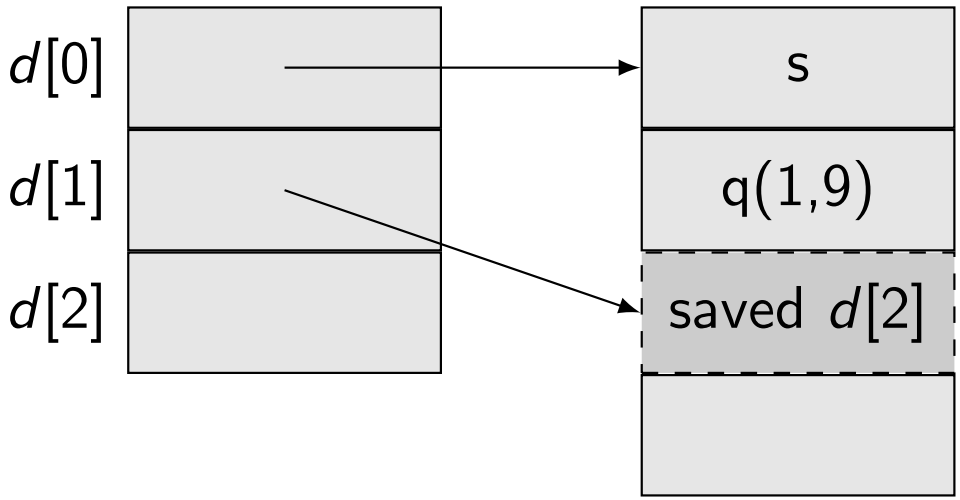


# Quicksort mit verschachtelten Prozeduren

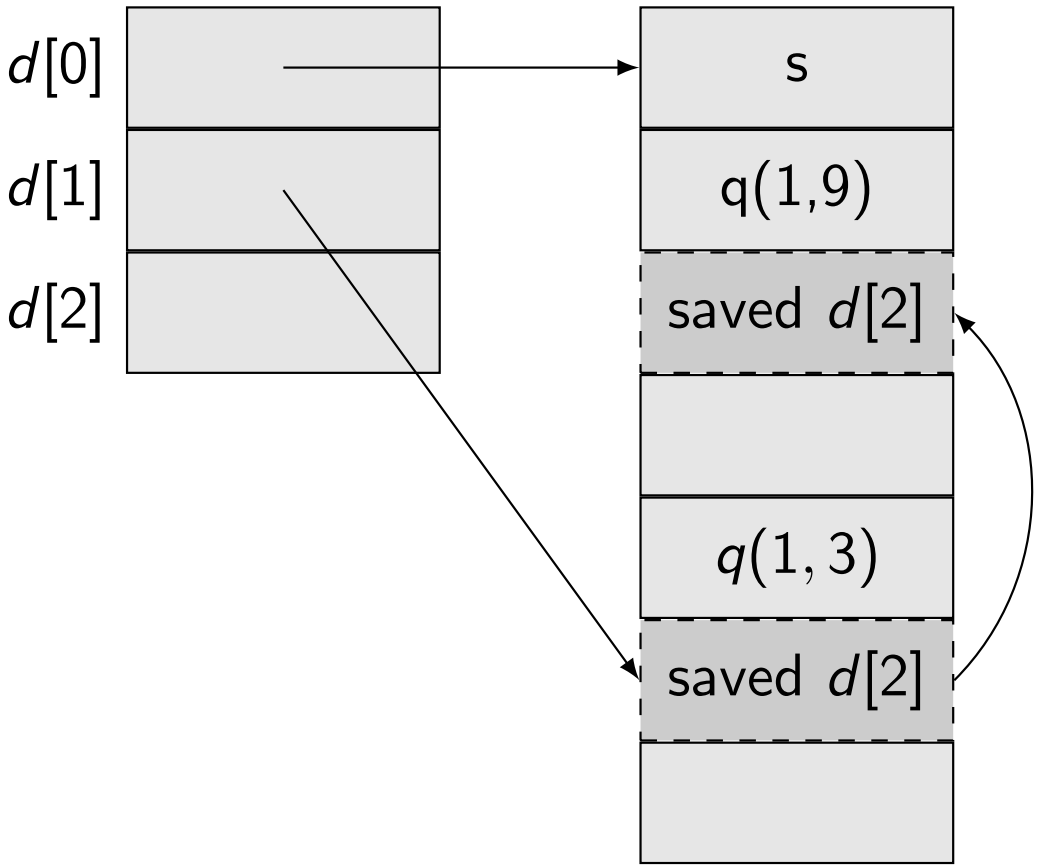
```
int main(void) {
    int a[11] = { INT_MIN,6,5,3,4,2,5,19,9,9,INT_MAX };
    void q(int l, int r) {
        void e(int e1, int e2) { /* exchange 2 elements */
            int t = a[e1]; a[e1] = a[e2]; a[e2] = t;
        }
        void p(int l, int r) {
            /* choose pivot element and partition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



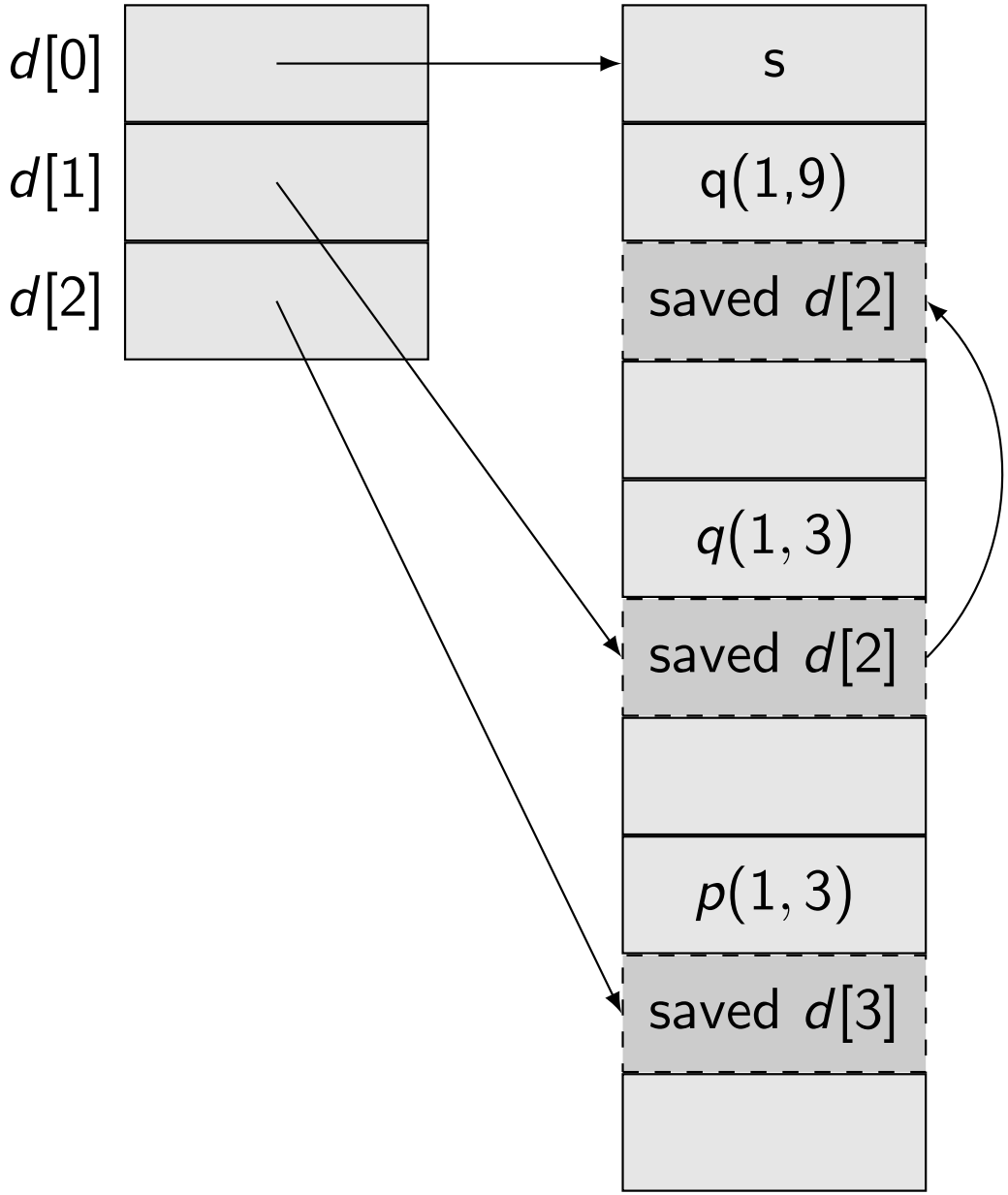
# Displays



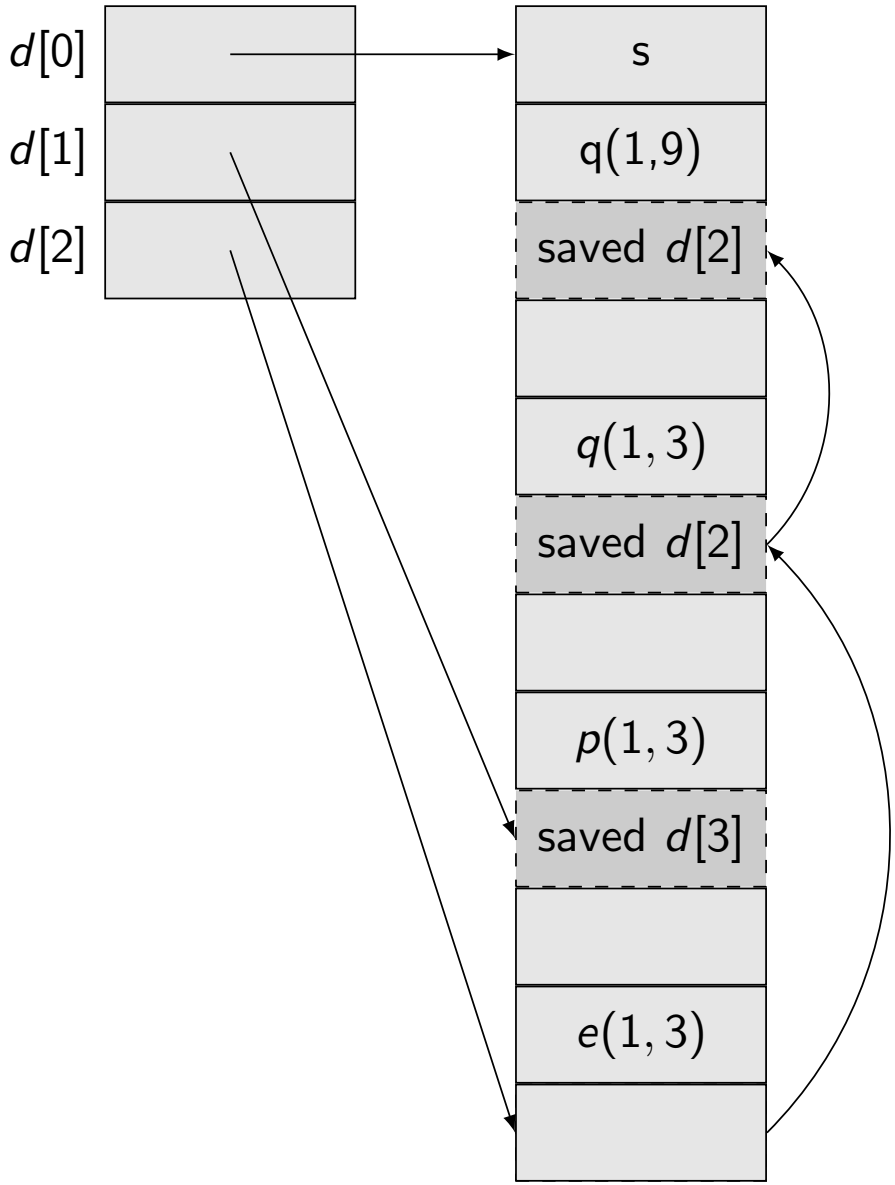
# Displays



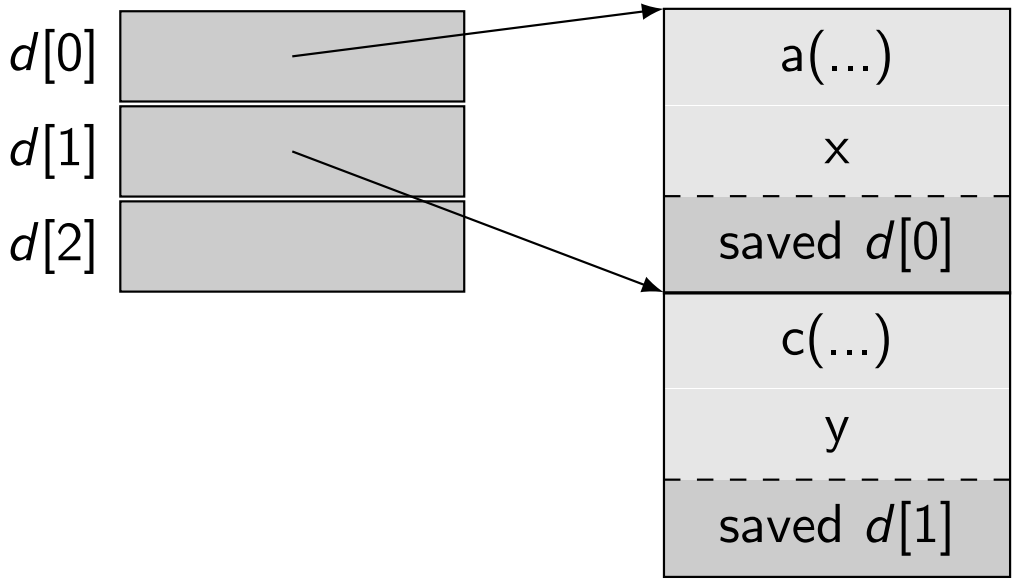
# Displays



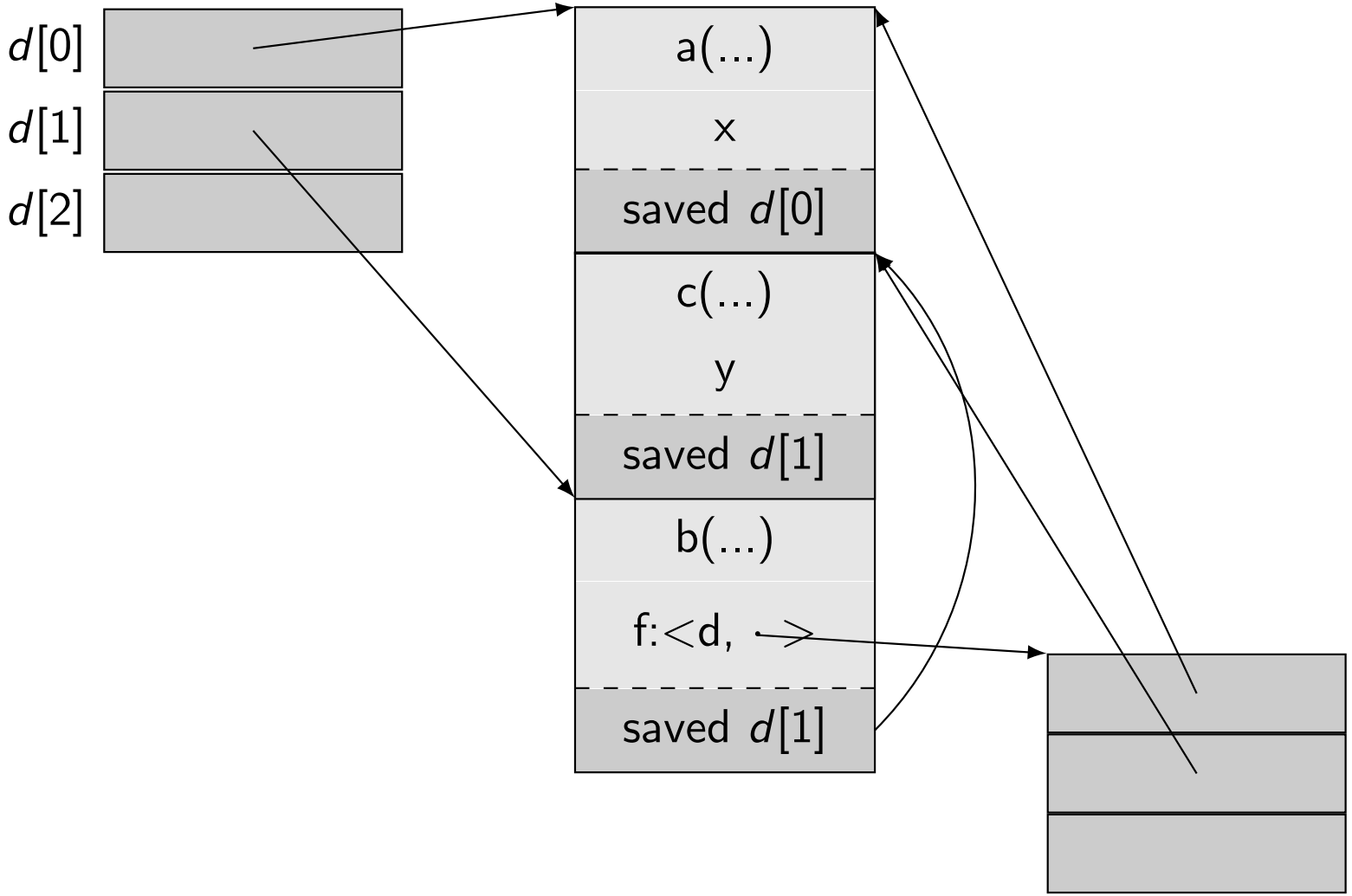
# Displays



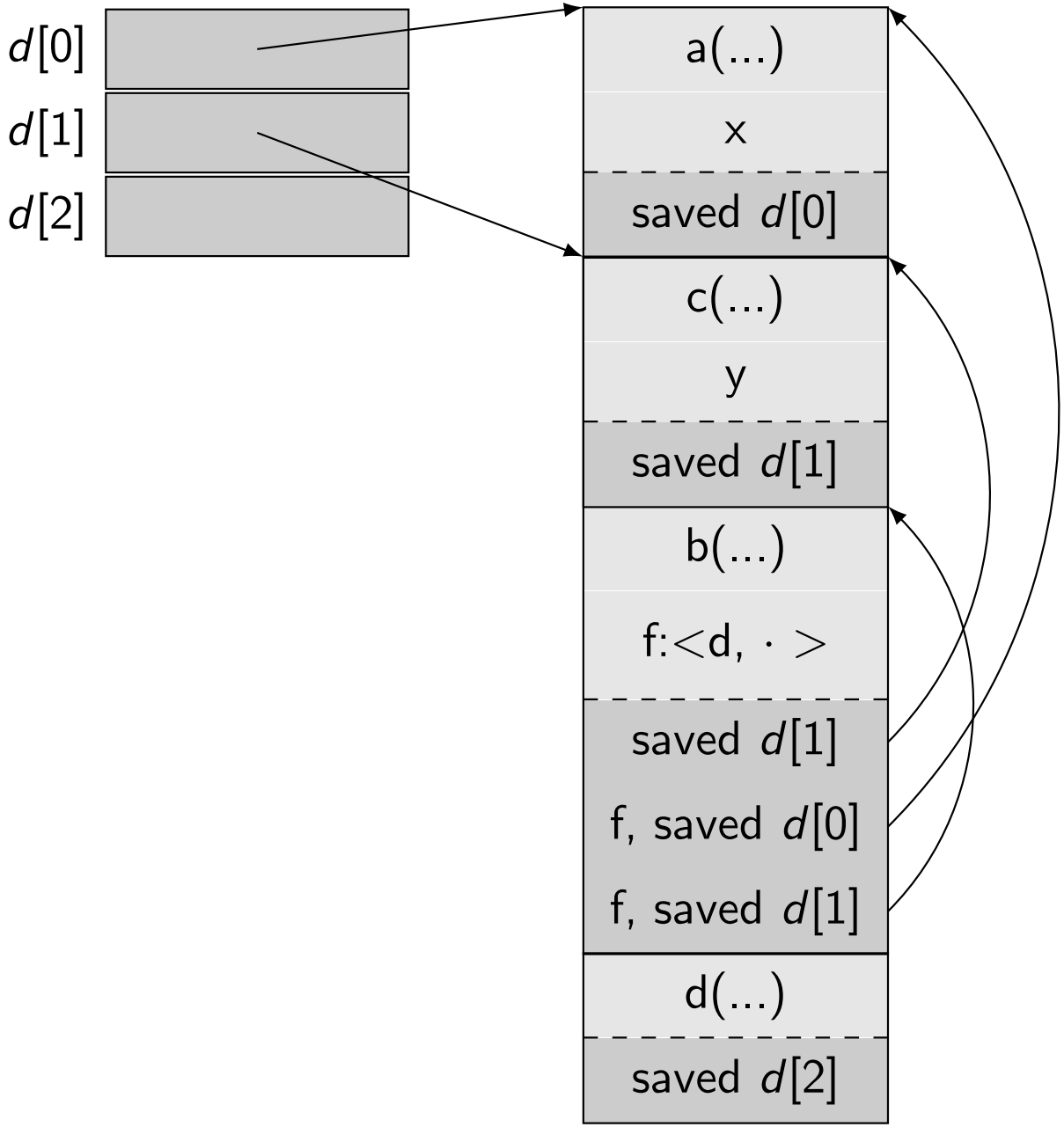
# Prozedurparameter mit Displays



# Prozedurparameter mit Displays



# Prozedurparameter mit Displays



# Literatur


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