7. Optimierung

Verschiedene „Hilfsmittel“ zur Entscheidung der Optimierungsfrage:

- abstrakte Interpretation
- Datenflußanalysen wie:
- „lebendige Variablen“
- „ankommende Definitionen“
- „verfügbare Ausdrücke“
- „wichtige Ausdrücke“

7. Optimierung

Beispiel 1:

```
i := 0;
while i < n do
  a[i] := b[i];
  i := i+1
od
```

Zwischen-Code →

```
i := 0;
while i < n do
  t1 := i*csize(b);
  t2 := addr(b) + t1;
  t3 := i*csize(a);
  t4 := addr(a) + t3;
  mem(t4) := mem(t2);
  i := i+1;
od
```

Wo kann gemäß 1.-4. optimiert werden?

7. Optimierung

Beispiel 1 (Fortsetzung):

```
i := 0;
while i < n do
  t1 := i*csize(b);
  t2 := addr(b) + t1;
  t3 := t1;
  t4 := addr(a) + t1;
  mem(t4) := mem(t2);
  i := i+1;
od
```

Datenflußanalyse „lebende Variabeln“ liefert →

```
..
while i < n do
  {}
  t1 := i*csize(b);
  {t1}
  t2 := addr(b) + t1;
  {t2,t1}
  t3 := t1;
  {t2,t1}
  t4 := addr(a) + t1;
  {t4,t2}
  mem(t4) := mem(t2);
  {}
  i:=i+1;
od
```

7. Optimierung

Der Drei-Adress-Code

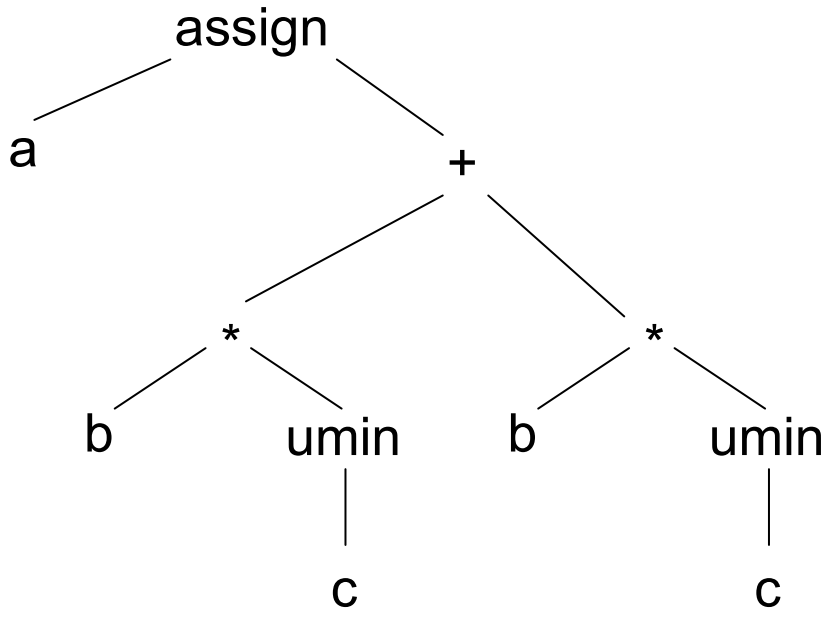
- Folge von Anweisungen der allgemeinen Form
- `x := y` *op* `z`
- x, y, z: Namen, Konstanten, temporäre Werte
- *op* beliebiger Operator (z.B. arithmetischer, logischer,..)
- Bsp: `x+y*z` würde übersetzt in
  ```
  t1:= y*z
  t2:= x+t1
  ```

- stellt linearisierte Darstellung des Syntaxbaum dar
- gut geeignet für Optimierung und Zielcode-Generierung

7. Optimierung

Beispiel 2:

abstrakter Syntaxbaum
zu a:= b * -c + b * -c



Felddarstellung

0	id	b	
1	id	c	
2	umin	1	
3	*	0	2
4	id	b	
5	id	c	
6	umin	5	
7	*	4	6
8	+	3	7
9	id	a	
10	assign	9	8
11	…	…	…

7. Optimierung

Beispiel 2 (Fortsetzung):

Drei-Adress-Code zu a:= b * -c + b * -c

t1 := - c
t2 := b * t1
t3 := - c
t4 := b * t3
t5 := t2 + t1
a := t5

7. Optimierung

Gebräuchlichste Drei-Adress-Code-Anweisungen:

- Zuweisungsanweisungen `x := y op z`
- Zuweisungsanweisungen `x =: op y`
- Kopieranweisungen `x := y`
- unbedingter Sprung `goto L`
- bedingter Sprung `if x relop y  goto L`
- Prozeduraufrufe mit

```
param x1
…
param xn
call p,n
return y
```

- indizierte Zuweisungen

```
x := y[i]
x[i] := y
```

- Adress- und Zeigerzuweisungen

```
x := &y
x := *y
*x := y
```

7. Optimierung

Beispiel 3:

```
void quicksort (int m, int n)
{   int i,j,v,x;
    if (n <= m) return;
    /* ab hier wichtig für Beispiel */
    i = m-1; j = n; v = a[n];
    while (1) {
          do i=i+1;  while (a[i] < v);
          do j=j-1;  while (a[j] > v);
          if (i >= j) break;
          x = a[i]; a[i] = a[j]; a[j] = x;
    }
    x = a[i]; a[i] = a[n]; a[n] = x;
    /* bis hierher */
    quicksort(m,j); quicksort(i+1,n);
}
```

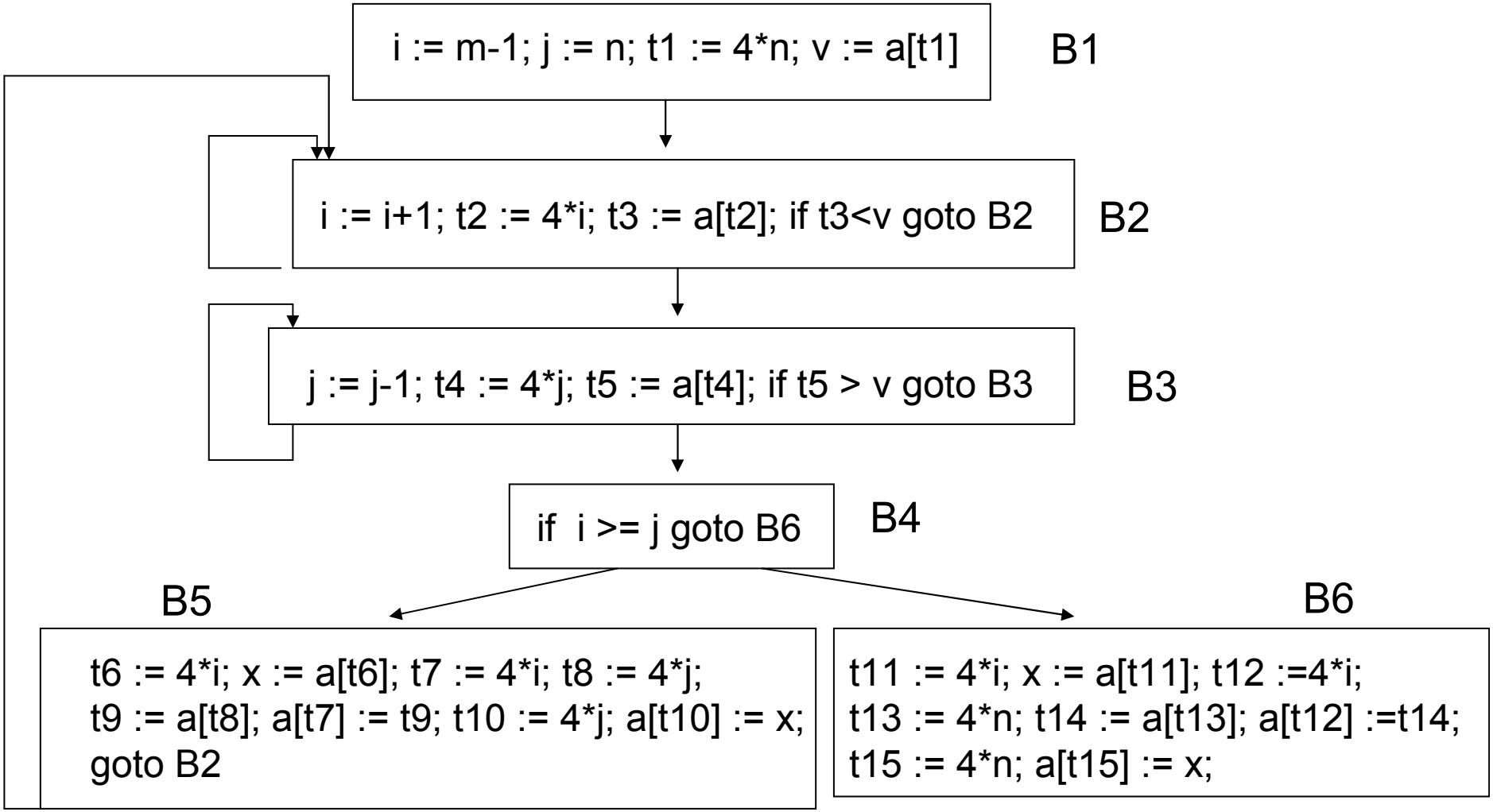
7. Optimierung

Beispiel 3 (Fortsetzung):

```
(1) i := m -1
(2) j := n;
(3) t1 := 4 * n
(4) v := a[t1]
(5) i := i+1
(6) t2 := 4*i;
(7) t3 := a[t2]
(8) if t3 < v goto (5)
(9) j := j-1
(10)t4 := 4*j
(11)t5 := a[t4]
(12)if t5 > v goto (9)
(13)if i >=j goto (23)
(14)t6 :=4*i
(15)x := a[t6]
(16) t7 := 4*i
(17) t8 := 4*j
(18) t9 := a[t8]
(19) a[t7] := t9
(20) t10 := 4*j
(21) a[t10] := x
(22) goto (5)
(23) t11 := 4*i
(24) x := a[t11]
(25) t12 := 4*i
(26) t13 := 4*n
(27) t14 := a[t13]
(28) a[t12] := t14
(29) t15 := 4*n
(30) a[t15] := x
```

7. Optimierung

Beispiel 3 (Fortsetzung): Flußgraph zum Drei-Adress-Code



7. Optimierung

Beispiel 3 (Fortsetzung):

Eliminierung lokaler gemeinsamer Teilausdrücke

B5

```
t6 := 4*i; x := a[t6]; t7 := 4*i; t8 := 4*j;
t9 := a[t8]; a[t7] := t9; t10 := 4*j; a[t10] := x;
goto B2
```



```
t6 := 4*i
x := a[t6]
t8 := 4*j
t9 := a[t8]
a[t6] := t9
a[t8] := x
goto B2
```

B6

```
t11 := 4*i; x := a[t11]; t12 :=4*i;
t13 := 4*n; t14 := a[t13]; a[t12] :=t14;
t15 := 4*n; a[t15] := x;
```



```
t11 := 4*i;
x := a[t11];
t13 := 4*n;
t14 := a[t13];
a[t11] :=t14;
a[t13] := x;
```

7. Optimierung

Beispiel 3 (Fortsetzung):

Eliminierung (globaler) gemeinsamer Teilausdrücke

Ausdruck E gemeinsamer Teilausdruck, wenn

– E schon zuvor berechnet wurde
– die Werte der in E enthaltenen Variablen sich seitdem nicht geändert haben

in B1 : t1 := 4*n } n zwischenzeitlich unverändert: in B6
In B6: t13:= 4*n } t13 := 4*n eliminieren, t14 :=a[t1];..;a[t1] := x

In B2: t2 := 4*i; t3:= a[t2]; } in B3,B4 ändern sich i, a[4*i] nicht
In B5: t6 := 4*i; x := a[t6]; } → in B5 x := t3;.. ;a[t2] := t9;
a[t6] := t9;

Aufgabe: Welche Ausdrücke eliminieren sich in B5 durch B3?
Welche in B6 durch B2, B3?

7. Optimierung

Beispiel 3 (Lösung):

Aufgabe: Welche Ausdrücke eliminieren sich in B5 durch B3?

Welche in B6 durch B2, B3?

in B2 : t2 := 4*i; t3 := a[t2]

In B6: t11 := 4*i; x := a[t11]; t12 := 4*i;..;a[t12] := t14;

ergibt sich in B6 zu

x := t3; …; a[t2] := t14;

In B3: t4 := 4*j; t5 := a[t4]

In B5: t8 := 4*j; t9 := a[t8]; a[t2] := t9; a[t8] := x (nach Folie 165)

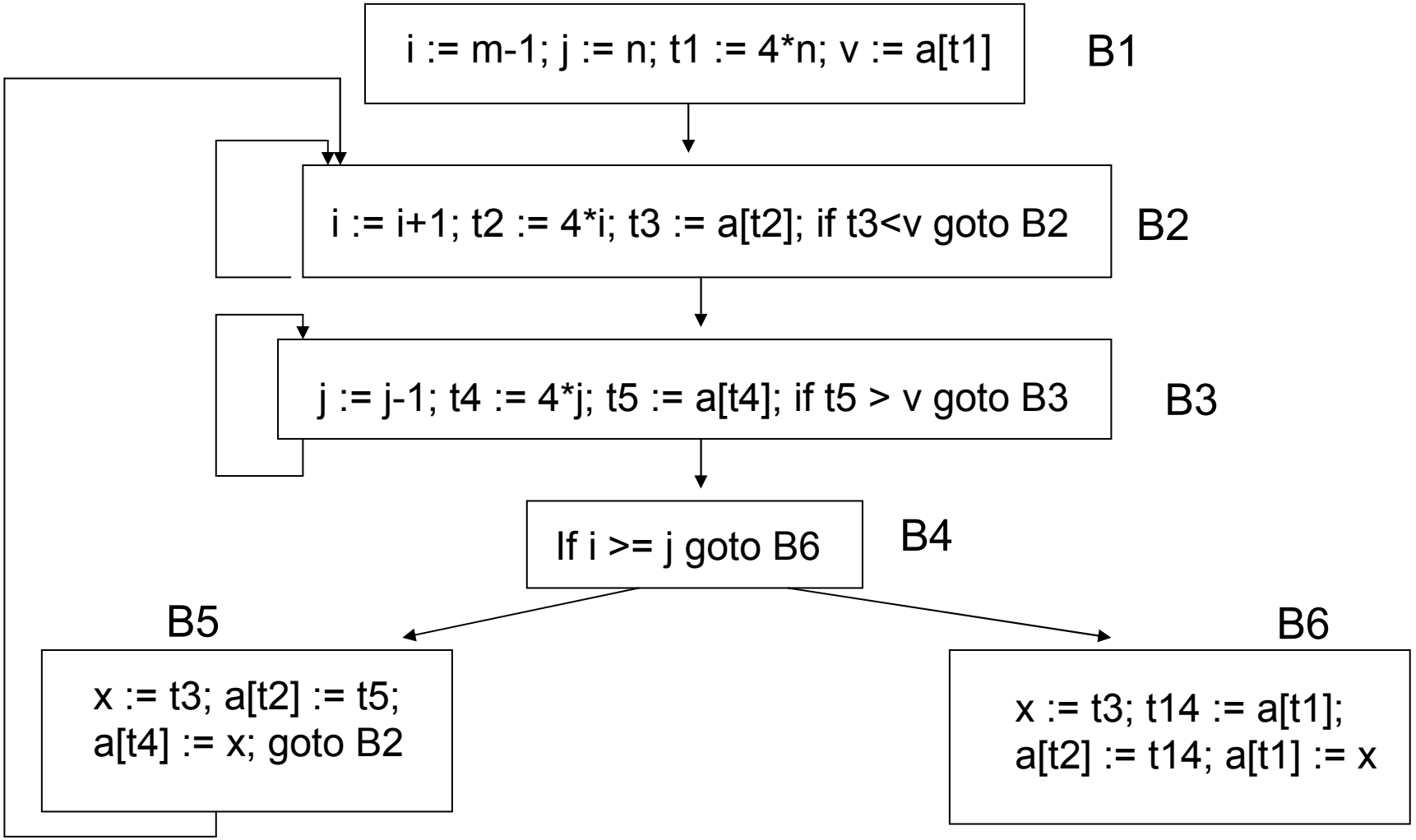
ergibt sich in B5 zu

a[t2] := t5; a[t4] :=x

In B6 durch B3 keine Änderungen.

7. Optimierung

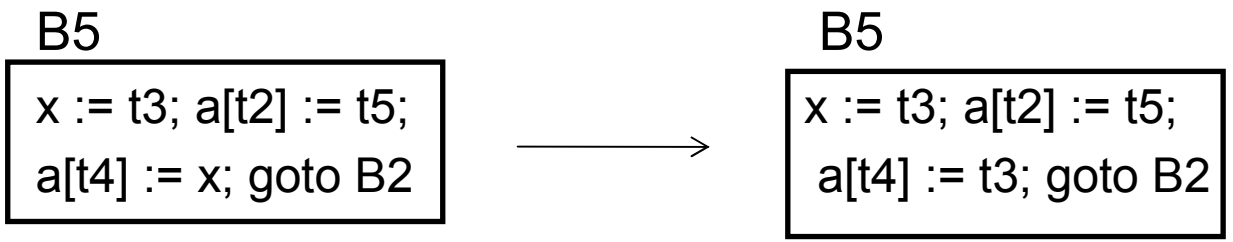
Beispiel 3 (Fortsetzung): Ergebnis nach Elimination



7. Optimierung

Weitergabe von Kopien:

- weitere Optimierungstransformation zur Eliminierung von Ausdrücken
- bezogen auf Kopier-Anweisungen f:=g
- Grundidee: nach Kopieranweisung f:=g so weit wie möglich g statt f zu nutzen
- evtl. Verzicht auf f möglich
- Beispiel: in B5 statt x t3 nutzen



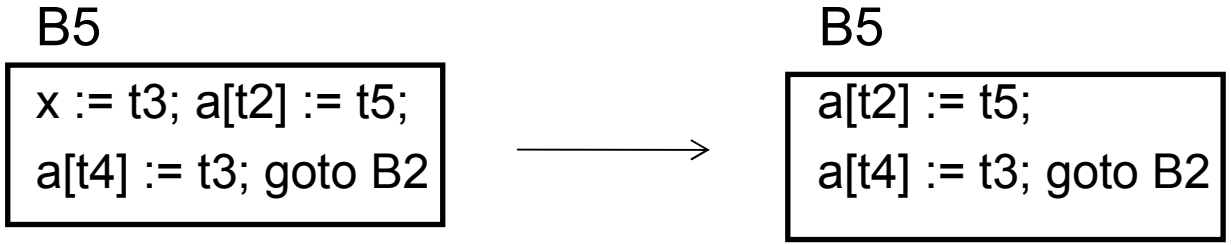
- wirklich Verbesserung? Ja,..

7. Optimierung

Entfernen von passivem Code:

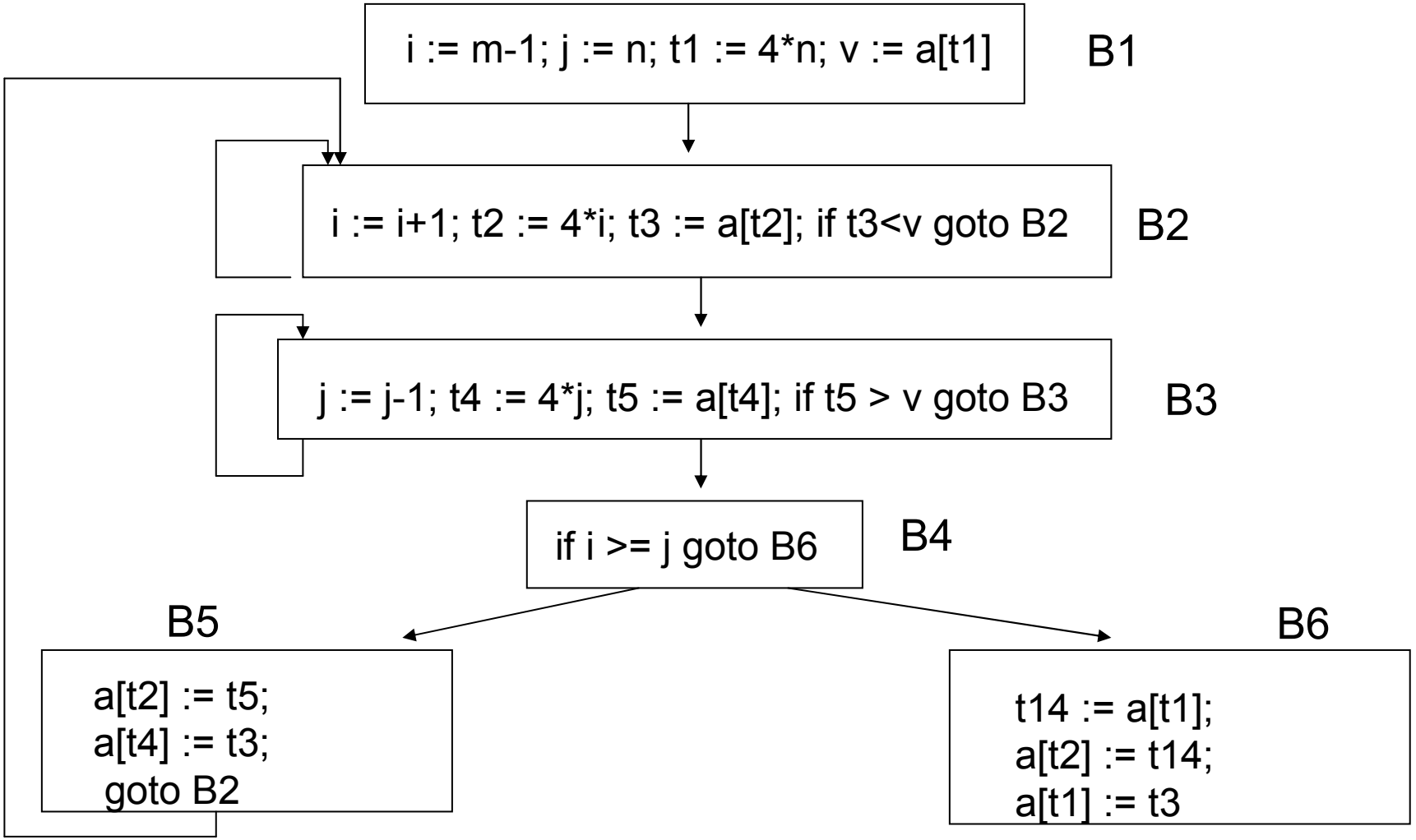
- 4. Fragestellung bzgl. Optimierungsalgorithmen
- *aktive* Variable: ihr Wert wird nachfolgend noch benutzt
- sonst *passive* Variable
- selten durch Programmierer:
- öfter durch vorangegangene Transformationen
- Kopieranweisung oft passiver Code

```
debug :=false;
…
if (debug) { … }
```



7. Optimierung

Beispiel 3 (Fortsetzung): Ergebnis nach allen Eliminationen



7. Optimierung

Schleifenoptimierungen (1):

– Code-Verschiebung:

Schleifen-invariante Berechnungen vor die Schleife ziehen

```
while (i <= limit – 2) {
       /*limit wird in Schleife nicht veraendert*/
       a:= limit*3.14;
       …}
```

→

```
t1:= limit – 2;
t2:= limit * 3.14;
while (i <= t1) {
       /*limit wird in Schleife nicht veraendert*/
       a:= t2;
       …}
```

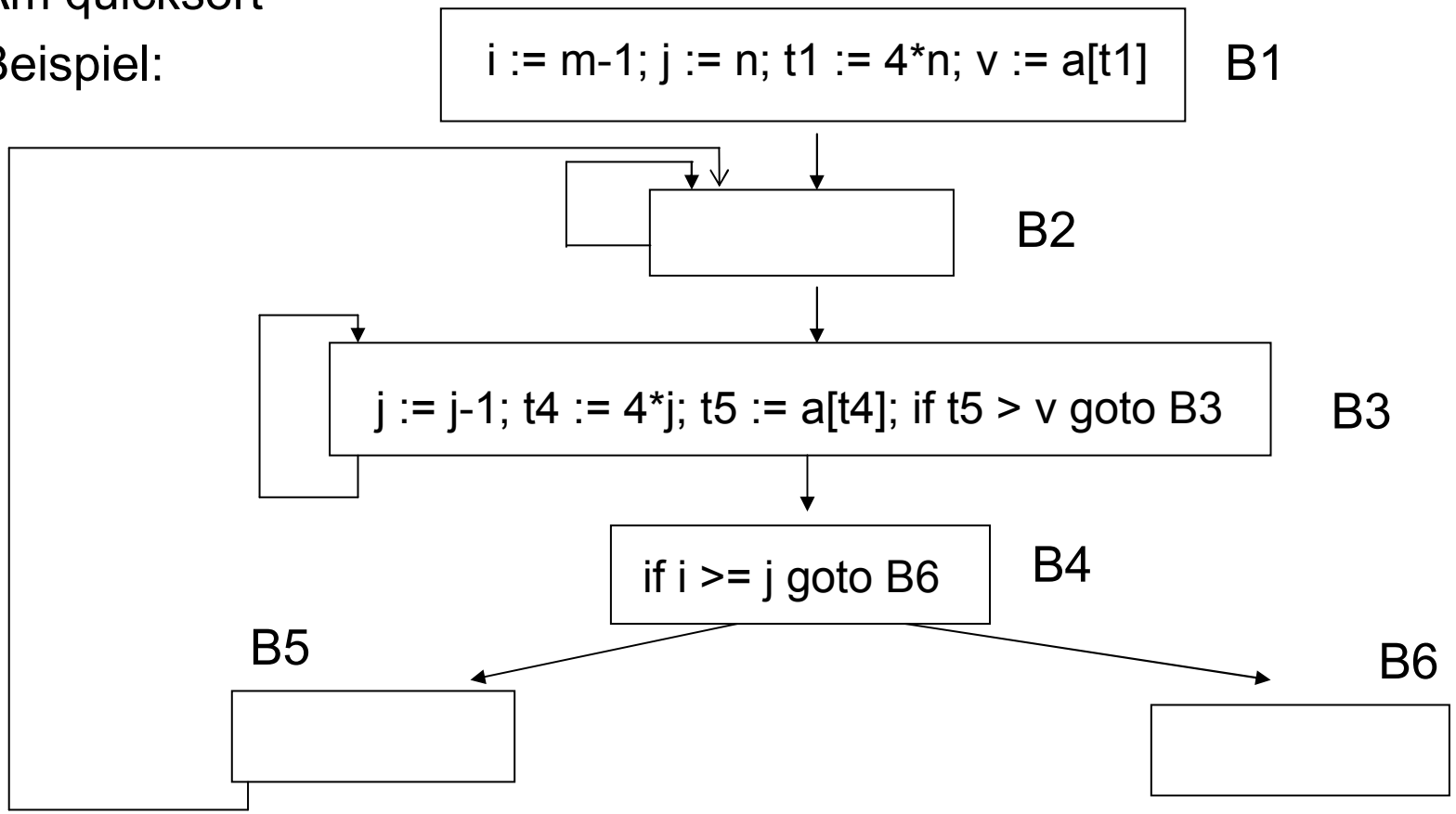
7. Optimierung

Schleifenoptimierungen (2):

– Elimination von Induktionsvariablen

– Verringerung von Operationskosten

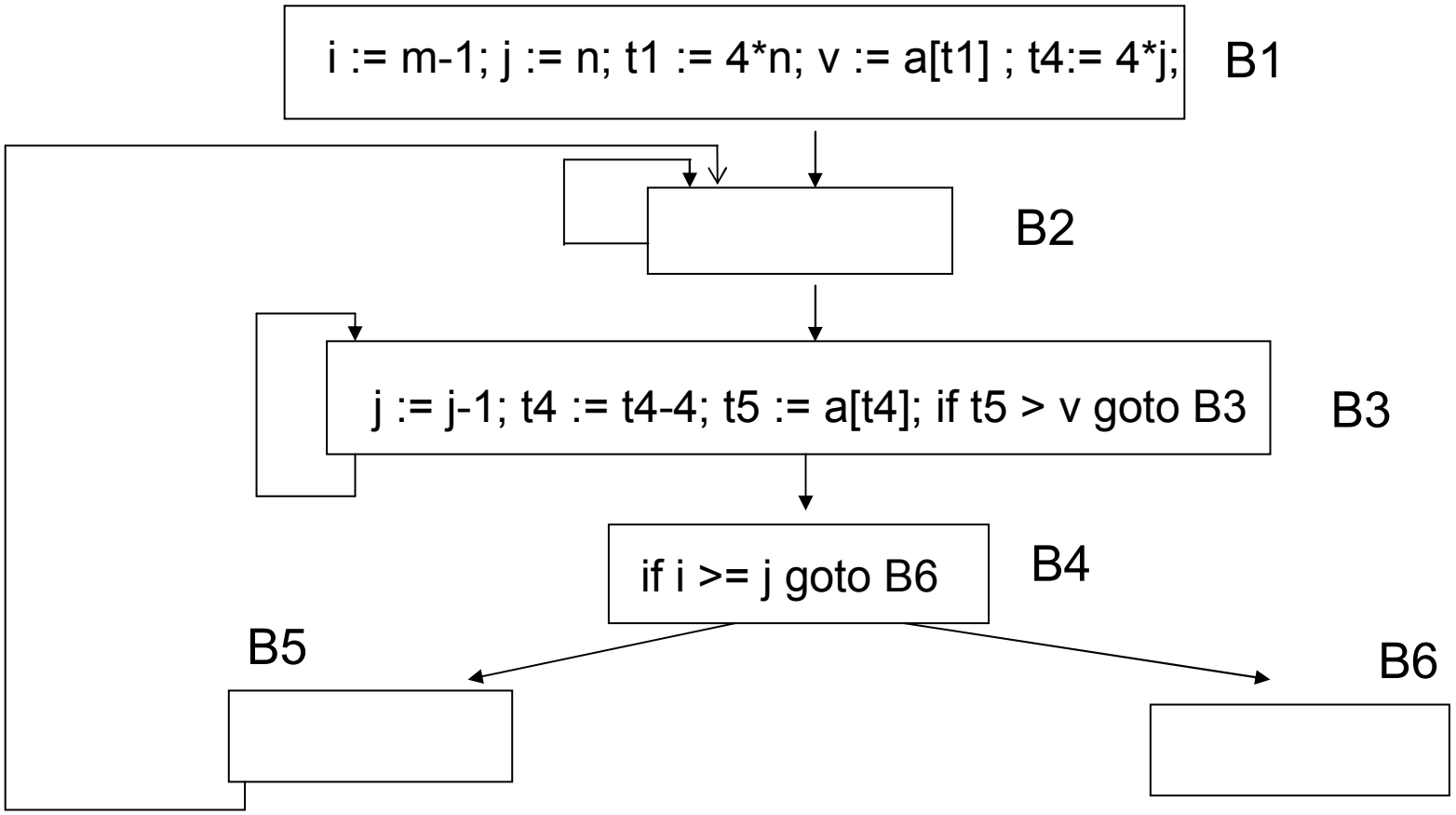
Am quicksort-Beispiel:



7. Optimierung

Schleifenoptimierungen (3):

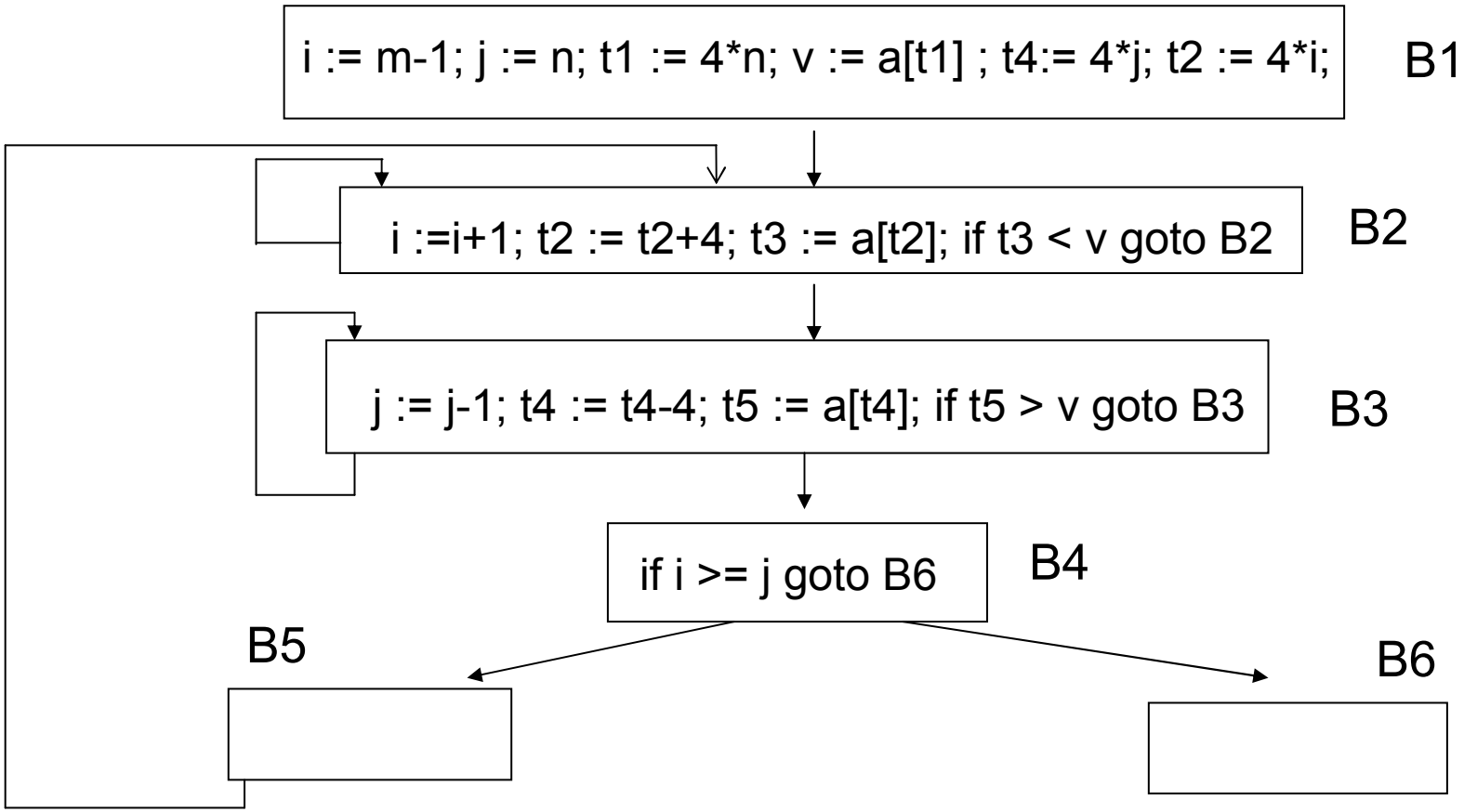
Nach Verringerung der Operationskosten in B3



7. Optimierung

Schleifenoptimierungen (4):

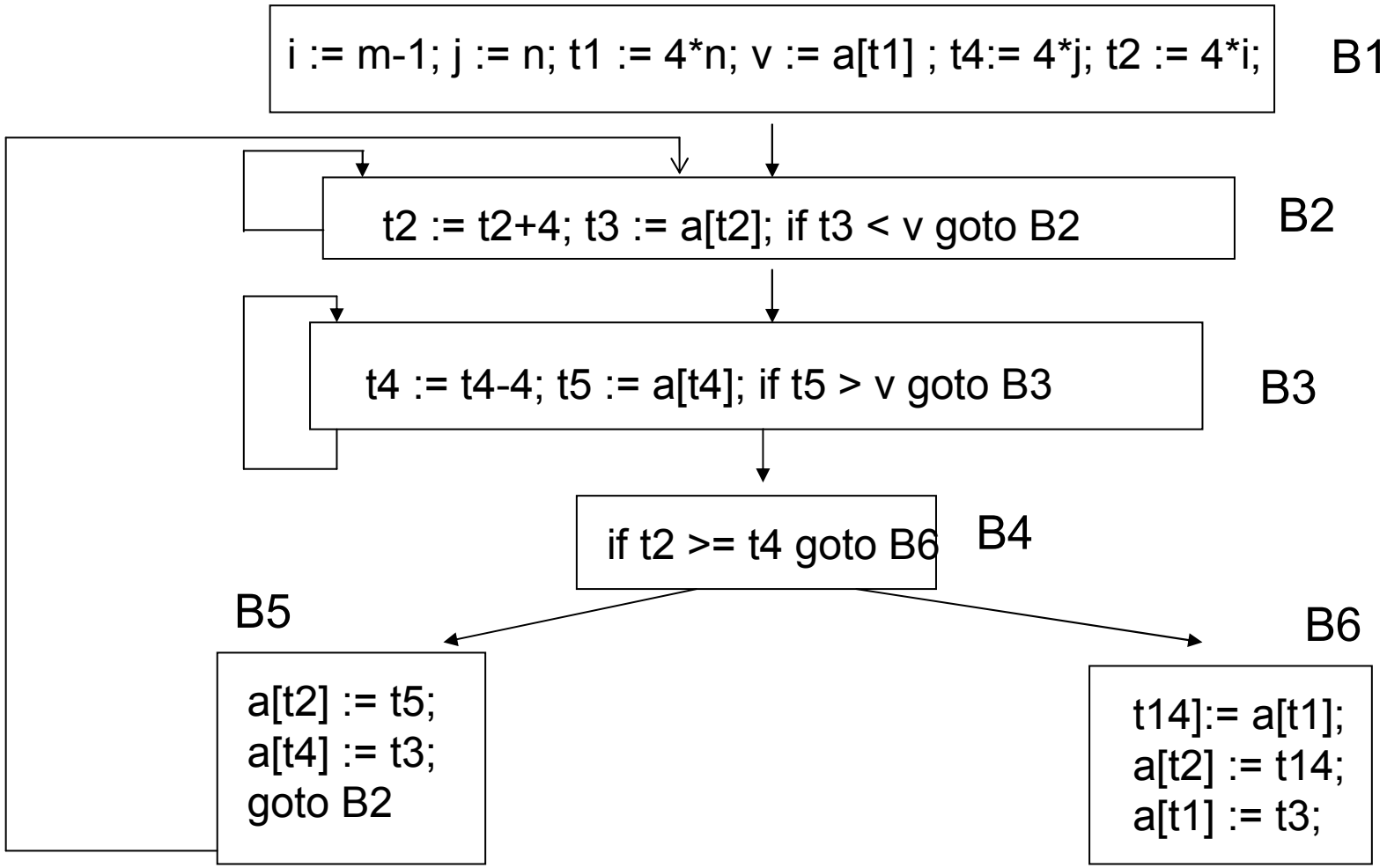
Nach Verringerung der Operationskosten in B2 und B3



7. Optimierung

Schleifenoptimierungen (5):

Nach anschließender Elimination von Induktionsvariablen



7. Optimierung

Beispiele für andere Optimierungen:

- Vereinfachung von typischen Ausdrücken:
  - x+0 = 0+x = x
  - x-0 = x
  - x*1 = 1*x = x
  - x/1 = x
- Reduktion von Operationskosten
  - x**2 = x*x
  - 2.0 * x = 2*x = x+x
  - x/2 = x*0.5
- Ausnutzen von Kommutativität/Assoziativität
  - a:= b+c; e := c+d+b; → a := b+c; t := c+d; e := t+b;
    → a := b+c; e := a + d; (falls t außerhalb nicht gebraucht wird)

7. Optimierung

Fragestellungen zu Kapitel 7:

- Wer kann optimieren? An welchen Stellen der Compilierung?
- Ziel der Optimierung?
- Wo im Programm macht Optimierung am meisten Sinn?
- Welche Optimierungsverfahren haben Sie kennengelernt?
- Drei-Adress-Code u. Datenflussgraph (erstellen)
- konkretes Verfahren anwenden können

Zwischenstand: