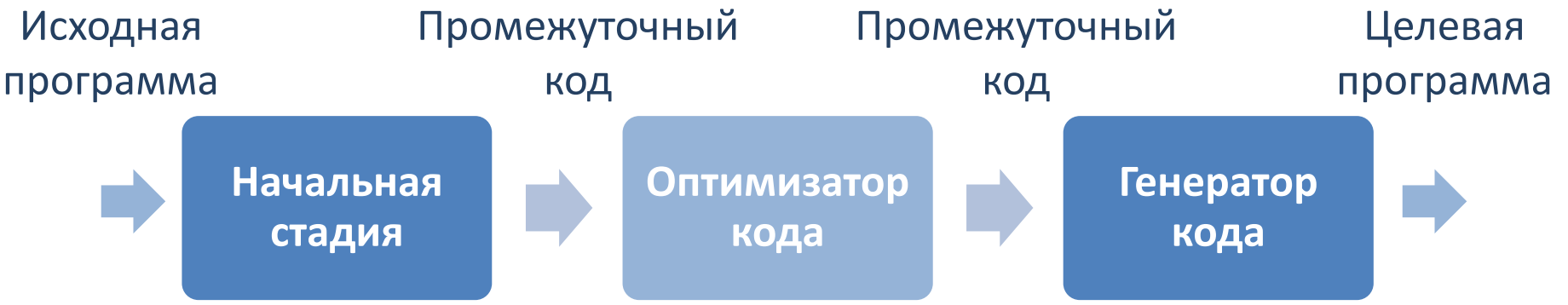
Оптимизация кода



Оптимизация программы

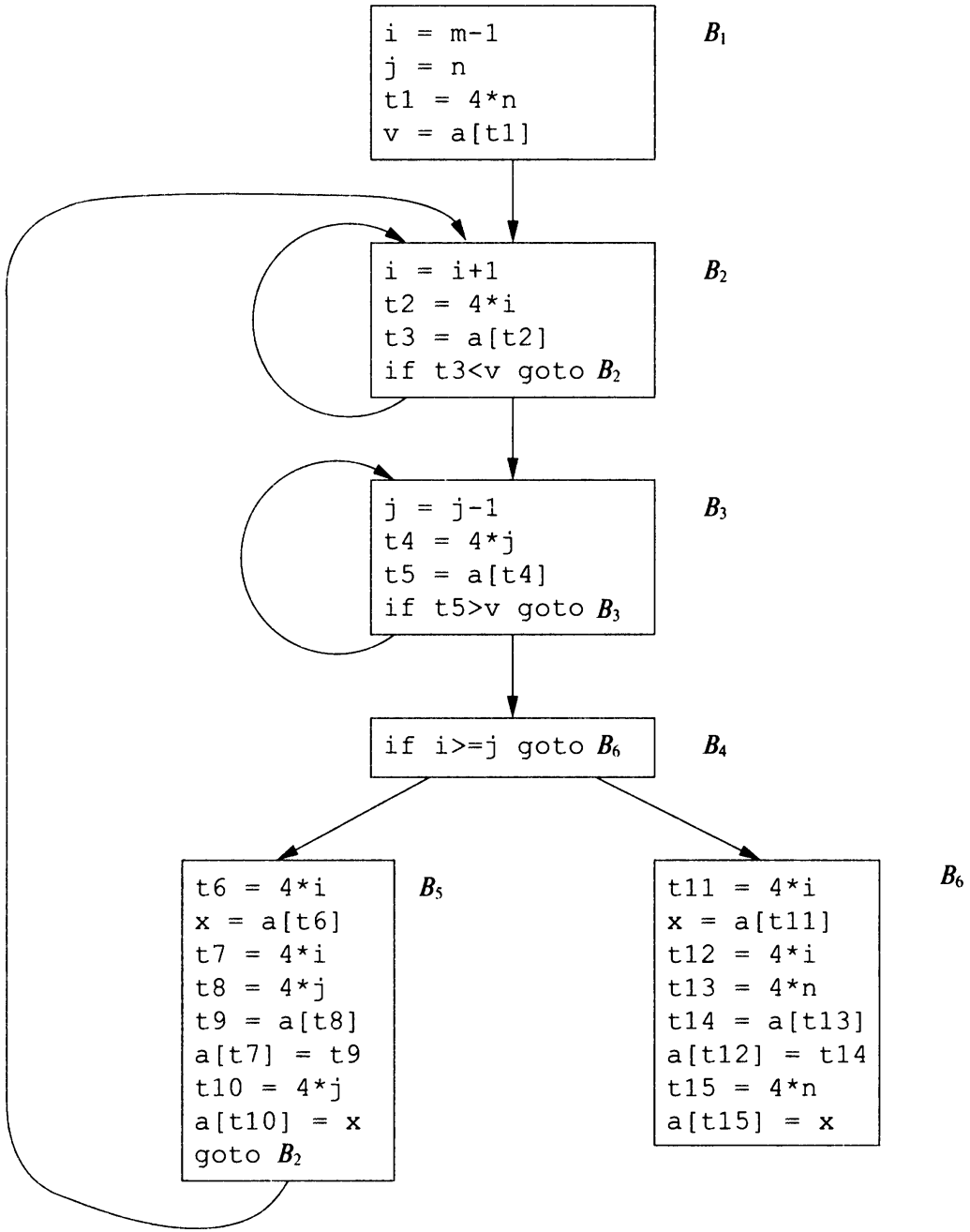
- *Оптимизация программы* – обработка, связанная с переупорядочиванием и изменением операций в компилируемой программе с целью получения более эффективной результирующей объектной программы

Пример (быстрая сортировка)

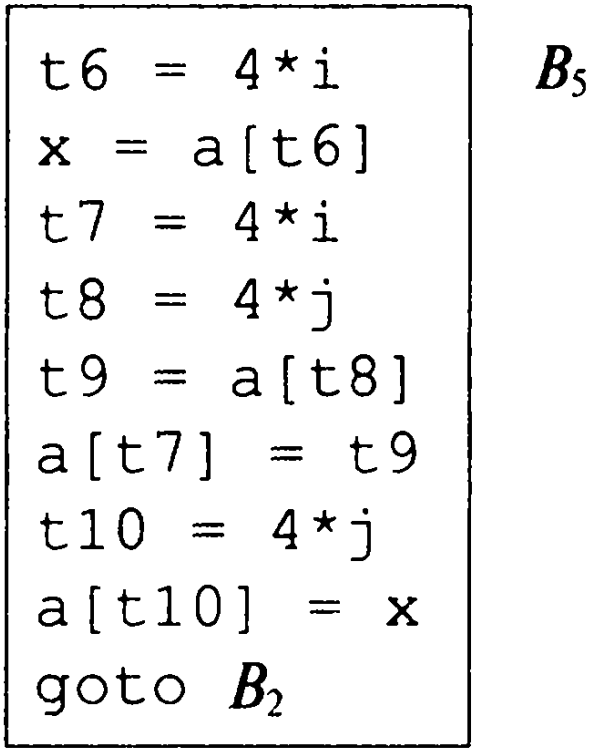
```
void quicksort(int m, int n)
    /* Рекурсивная сортировка диапазона от a[m] до a[n] */
{
    int i, j;
    int v, x;
    if (n <= m) return;
    /* Начало фрагмента */
    i = m-1; j = n; v = a[n];
    while (1) {
        do i = i+1; while (a[i] < v);
        do j = j-1; while (a[j] > v);
        if (i >= j) break;
        x = a[i]; a[i] = a[j]; a[j] = x;
            /* Обмен a[i], a[j] */
    }
    x = a[i]; a[i] = a[n]; a[n] = x;
        /* Обмен a[i], a[n] */
    /* Конец фрагмента */
    quicksort(m,j); quicksort(i+1,n);
}
```

```
(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
```

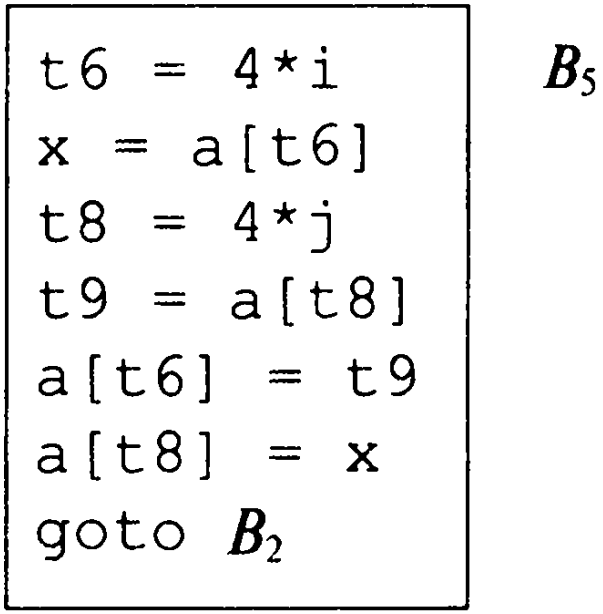
Граф потока для программы быстрой сортировки



Изменения, сохраняющие семантику



а) До



б) После

Глобальные общие подвыражения

- Выражение **E** называется *общим подвыражением*, если **E** было ранее вычислено и значения переменных в **E** с того времени не изменились

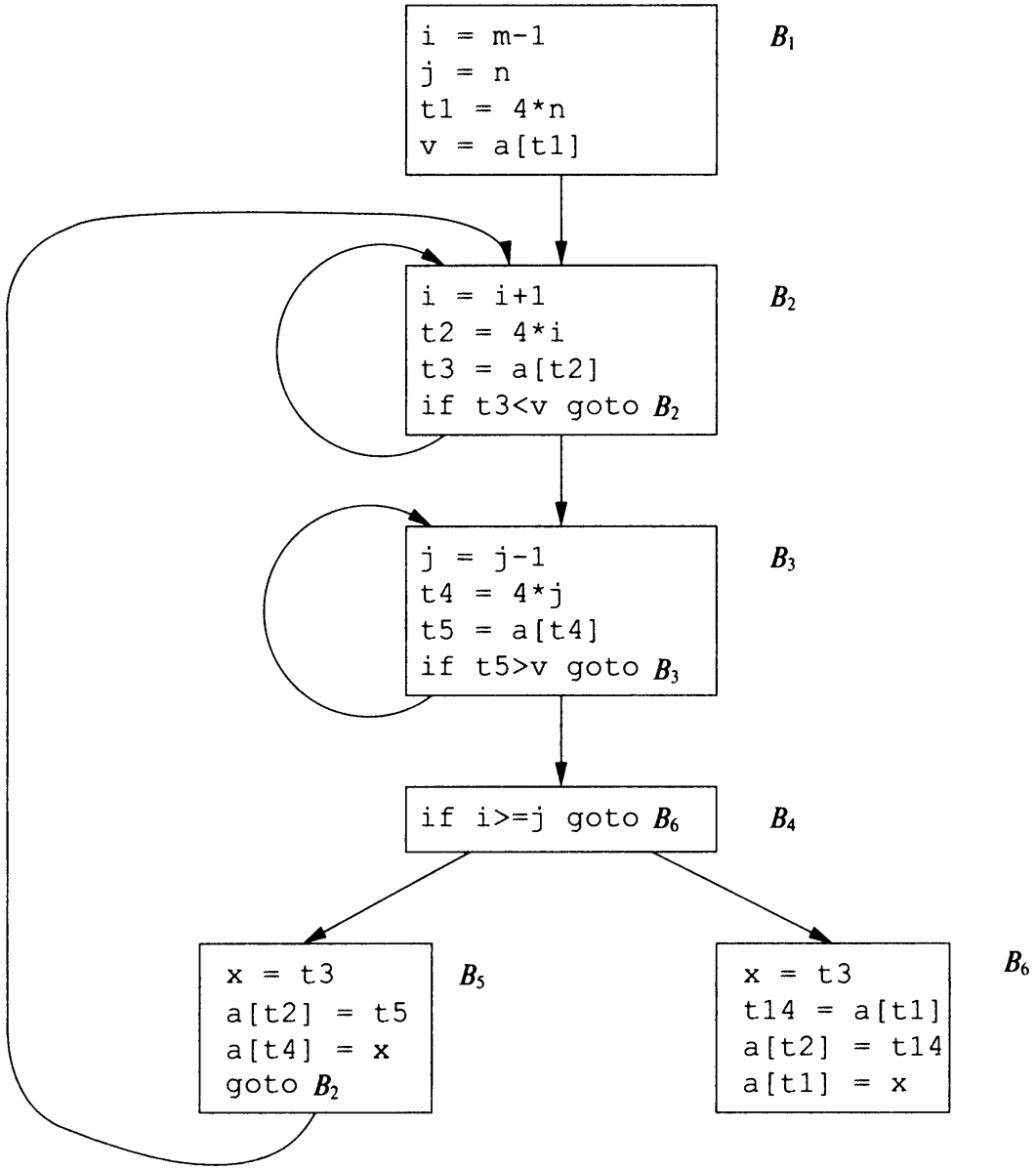
```
t8 = 4*j
t9 = a[t8]
a[t8] = x
```

```
t9 = a[t4]
a[t4] = x
```

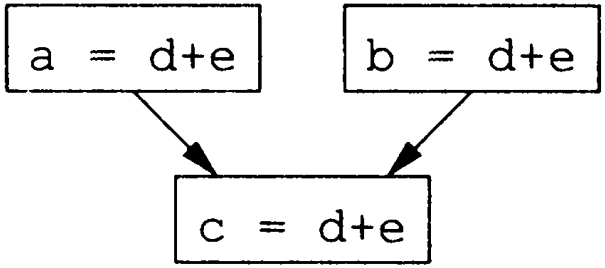
```
t9 = a[t4]
a[t6] = t9
```

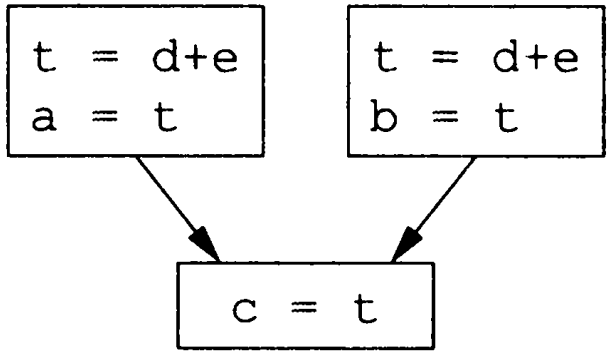
```
a[t6] = t5
```

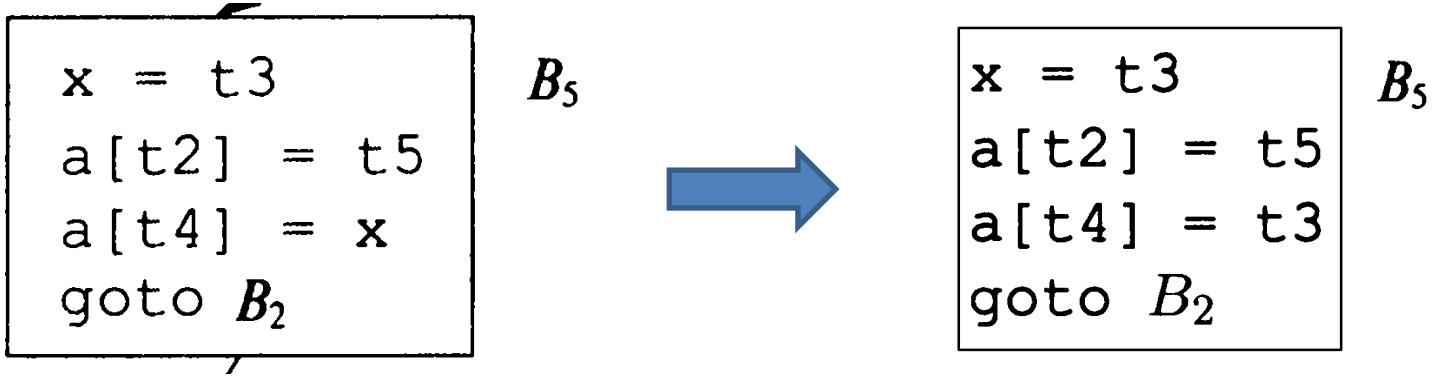
Граф потока после устранения локальных и глобальных общих подвыражений



Распространение копий

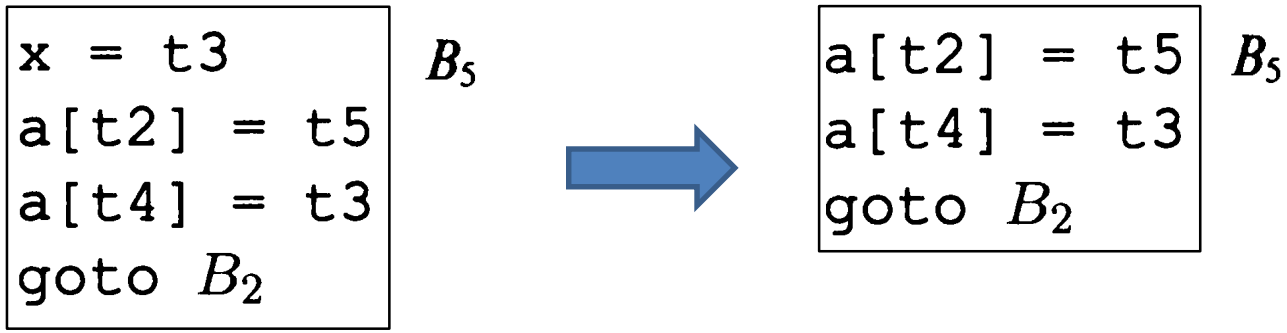






Удаление бесполезного кода

- Переменная в некоторой точке программы считается *живой* или активной, если ее значение будет использовано в программе в последующем; в противном случае она считается *мертвой*



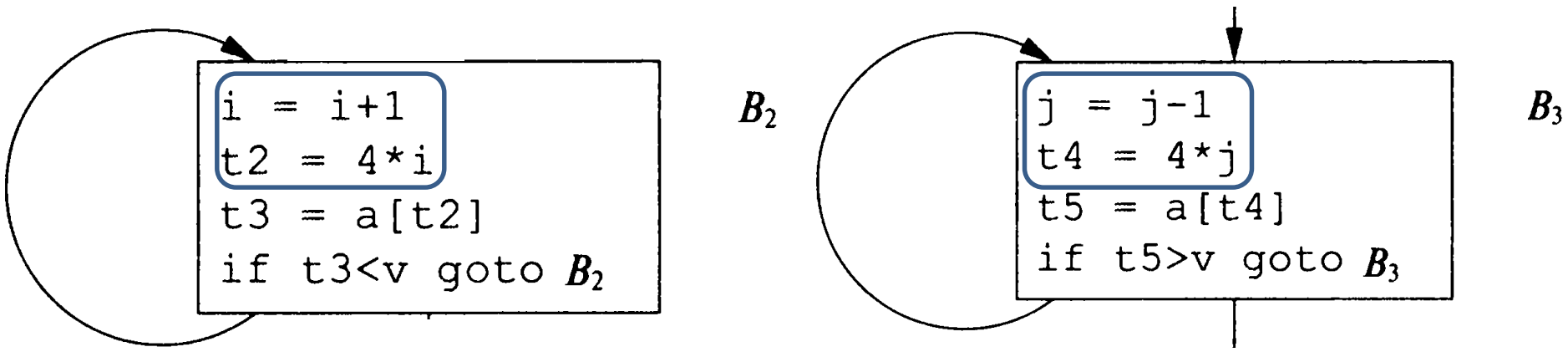
Перемещение кода

```
while(i <= limit - 2)
```

```
t = limit - 2;
while(i <= t)
```

Переменные индукции

- Переменная **x** называется *переменной индукции*, если существует положительная или отрицательная константа **c**, такая, что всякий раз при присваивании **x** ее значение изменяется на **c**
- Переменные индукции могут быть вычислены при помощи единственного увеличения (сложения или вычитания) в каждой итерации цикла



Преобразование снижения стоимости

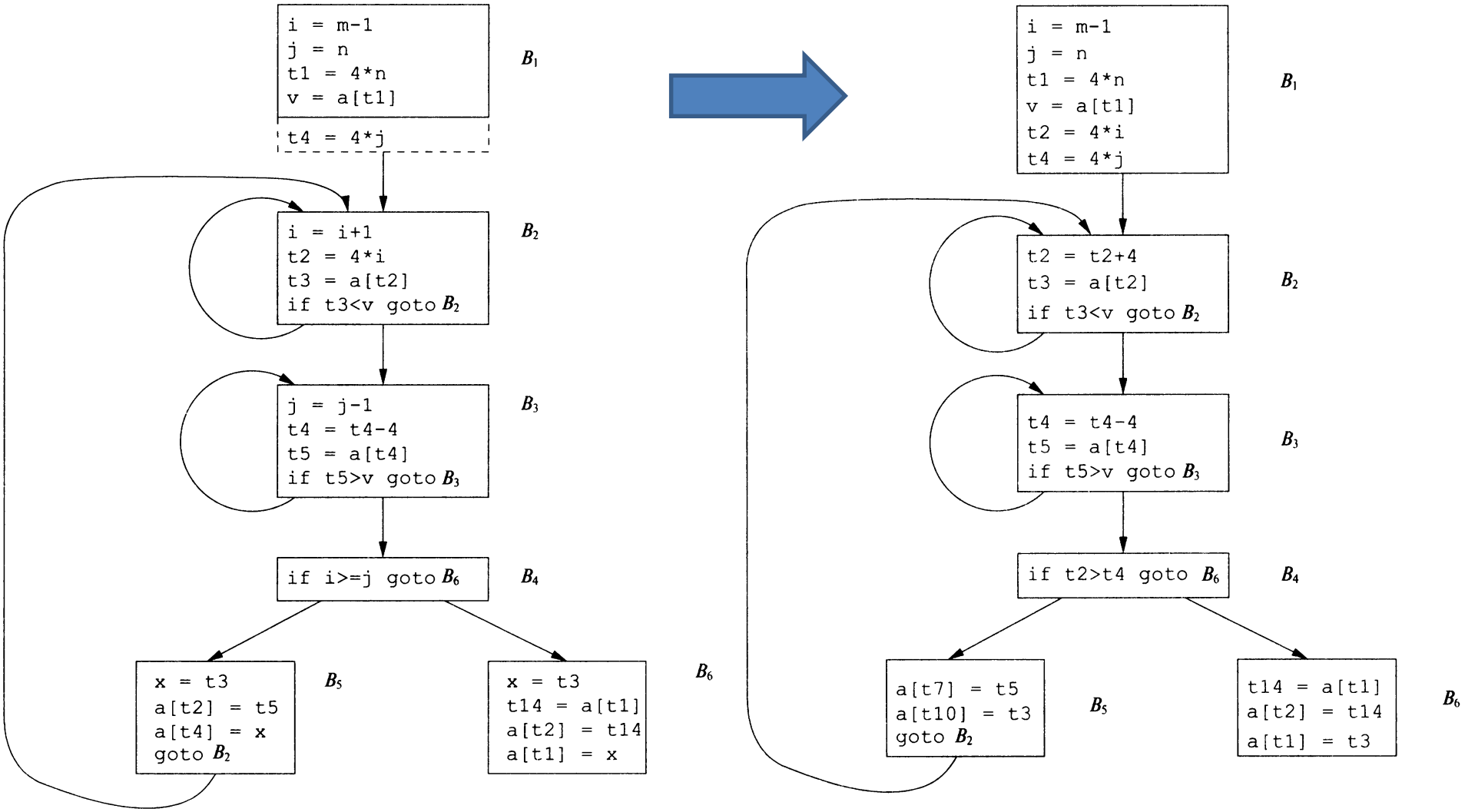
- Преобразование *снижения стоимости* состоит в замене дорогой операции, такой как умножение, более дешевой, такой как сложение

```
j = j - 1
t4 = 4 * j
```

```
1) j = 4
   t4 = 4 * 4 = 16
2) j = 3                  -4
   t4 = 4 * 3 = 12
3) j = 2                  -4
   t4 = 4 * 2 = 8
```

```
t4 = 4 * j        t4 = t4 - 4
```

Преобразование снижения стоимости



Исключение избыточных вычислений

```
D := D + C * B;
A := D + C * B;
C := D + C * B;
```

1. * (C, B)
2. + (D, ^1)
3. := (D, ^2)
4. * (C, B)
5. + (D, ^4)
6. := (A, ^5)
7. * (C, B)
8. + (D, ^7)
9. := (C, ^8)

Исключение избыточных вычислений

Триада	Зависимости переменных				Зависимости триад	Результат
	A	B	C	D		
1. * (C, B)	0	0	0	0	1	1. * (C, B)
2. + (D, ^1)	0	0	0	0	2	2. + (D, ^1)
3. := (D, ^2)	0	0	0	3	3	3. := (D, ^2)
4. * (C, B)	0	0	0	3	1	4. SAME(1, 0)
5. + (D, ^4)	0	0	0	3	4	5. + (D, ^4)
6. := (A, ^5)	6	0	0	3	5	6. := (A, ^5)
7. * (C, B)	6	0	0	3	1	7. SAME(1, 0)
8. + (D, ^7)	6	0	0	3	4	8. SAME(5, 0)
9. := (C, ^8)	6	0	9	3	5	9. := (C, ^8)

1. * (C, B)
2. + (D, ^1)
3. := (D, ^2)
4. + (D, ^1)
5. := (A, ^4)
6. := (C, ^4)

Свертка объектного кода

- Выполнение во время компиляции тех операций исходной программы, для которых значения операндов уже известны

```
I := 1 + 1;
I := 3;
J := 6 * I + I;
```

```
1) + (1, 1)
2) := (I, ^1)
3) := (I, 3)
4) * (6, ^3)
5) + (^4, I)
6) := (J, ^5)
```

Свертка объектного кода

1) + (1, 1)

2) := (I, ^1)

3) := (I, 3)

4) * (6,^3)

5) + (^4, I)

6) := (J, ^5)

Триада	Шаг 1	Шаг 2	Шаг 3	Шаг 4	Шаг 5	
1	C(2,0)	C(2,0)	C(2,0)	C(2,0)	C(2,0)	:= (I, 2) := (I, 3) := (J, 21)
2	:= (I, ^1)	:= (I, 2)	:= (I, 2)	:= (I, 2)	:= (I, 2)	
3	:= (I, 3)	:= (I, 3)	:= (I, 3)	:= (I, 3)	:= (I, 3)	
4	* (6,^3)	* (6, I)	*(6, 3)	C(18, 0)	C(18, 0)	
5	+ (^4, I)	+ (^4, I)	+ (^4, I)	+(18, 3)	C(21, 0)	
6	:= (J, ^5)	:= (J, ^5)	:= (J, ^5)	:= (J, ^5)	:= (J, 21)	

Перестановка операций

- Изменение порядка следования операций, которое может повысить эффективность программы, но не будет влиять на конечный результат вычислений

A = 2 * B * 3 * C; => A = 2 * 3 * B * C;

можно выполнить свёртку !

A = (B + C) + (D + E); => A = B + (C + (D + E));

Арифметические преобразования

- Выполнение изменения характера и порядка следования операций на основании известных алгебраических и логических тождеств

  A = B * C + B * D; => A = B * (C + D);

- Замена возведения в степень на умножение, а целочисленного умножения на выполнение операций сдвига

Оптимизация вычисления логических выражений

- Не всегда необходимо полностью вычислять всё выражение для того, чтобы знать его результат

  A or B or C or D

  Если A = true, не имеет смысла вычислять выражение
- Операция – *предопределённая для некоторого значения операнда*, если её результат зависит только от этого операнда и остаётся неизменным относительно значений других операндов
- *Операция логического сложения* (or) является предопределенной для логического значения «истина» (true)
- *Операция логического умножения* (and) является предопределенной для логического значения «ложь» (false)

Компиляторы строят объектный код вычисления логических выражений таким образом, что вычисление выражения прекращается сразу же, как только его значение становится предопределённым

Оптимизация передачи параметров в процедуры и функции

- Передача параметров через регистры процессора
- Подстановка кода функции в вызывающий объектный код

Слияние и развертывание циклов

- Слияние двух циклов в один

```
for i := 1 to N do
    for j := 1 to M do
        A[i, j] := 0;
```

```
K := N * M;
for i := 1 to K do
    A[i] := 0;
```

- Развертывание циклов

```
for i := 1 to 3 do
    A[i] := i;
```

```
A[1] := 1;
A[2] := 2;
A[3] := 3;
```