



中国科学技术大学  
University of Science and Technology of China

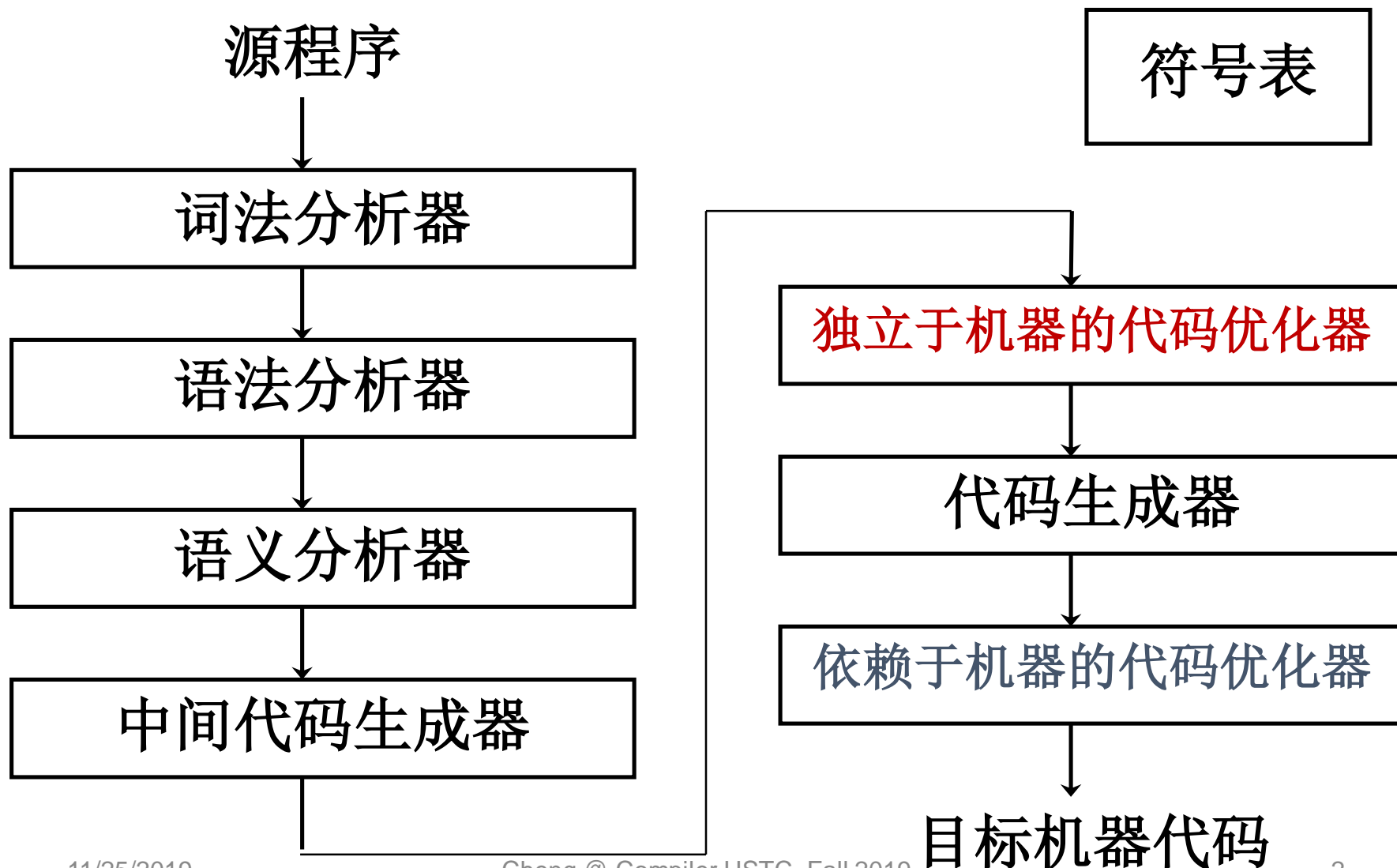


# 《编译原理与技术》 独立于机器的优化 I

计算机科学与技术学院

李 诚

25/11/2019





## □ 程序中存在许多程序员无法避免的冗余运算

如  $A[i][j]$  和  $X.f1$  这样访问数组元素和结构体的域的操作

- ❖ 编译后，这些访问操作展开成多步低级算术运算
- ❖ 对同一个数据结构多次访问导致许多公共低级运算



## □代码优化

❖通过程序变换（局部变换和全局变换）来改进程序，称为优化

## □代码优化的种类

- ❖基本块内优化、全局优化
- ❖公共子表达式删除、复制传播、死代码删除
- ❖循环优化

## □代码优化的实现方式

- ❖基本块的DAG表示与分析
- ❖数据流分析及其一般框架、循环的识别和分析



## □基本块与流图

## □优化策略

- ❖ 公共子表达式删除(common subexpression elimination)
- ❖ 复制传播(copy propagation)
- ❖ 常量合并(constant folding)
- ❖ 死代码删除(dead code elimination)
- ❖ 代码移动(code motion)
- ❖ 强度削弱(strength reduction)
- ❖ 删除归纳变量(induction variable elimination)

## 快速排序程序片段如下

$i = m - 1; j = n; v = a[n];$

**while (1) {**

**do  $i = i + 1$ ; while( $a[i] < v$ );**

**do  $j = j - 1$ ; while ( $a[j] > v$ );**

**if ( $i \geq j$ ) break;**

**$x = a[i]; a[i] = a[j]; a[j] = x;$**

**}**

**$x = a[i]; a[i] = a[n]; a[n] = x;$**

(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  $i \geq j$  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)  $a[t15] := x$



**□连续的三地址指令序列，控制流从它的开始进入，并从它的末尾离开，中间没有停止或分支的可能性（末尾除外）**



□输入：三地址指令序列

□输出：基本块列表

□算法：

❖首先确定基本块的第一个指令，即**首指令(leader)**

➤指令序列的**第一条三地址指令**是一个首指令

➤任意转移指令的**目标指令**是一个首指令

➤紧跟一个**转移指令的指令**是一个首指令

❖然后，每个首指令对应的基本块包括了从它自己开始，直到**下一个首指令(不含)或指令序列结尾**之间的所有指令





# 举例



```
(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
```



# 举例——首指令



(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  $i \geq j$  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)  $a[t15] := x$



# 举例——基本块



**B<sub>1</sub>**

(1)  $i := m - 1$   
(2)  $j := n$   
(3)  $t1 := 4 * n$   
(4)  $v := a[t1]$

-----

**B<sub>2</sub>**

(5)  $i := i + 1$   
(6)  $t2 := 4 * i$   
(7)  $t3 := a[t2]$   
(8) if  $t3 < v$  goto (5)

-----

**B<sub>3</sub>**

(9)  $j := j - 1$   
(10)  $t4 := 4 * j$   
(11)  $t5 := a[t4]$   
(12) if  $t5 > v$  goto (9)

-----

**B<sub>4</sub>**

(13) if  $i \geq j$  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)

-----

**B<sub>5</sub>**

(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)  $a[t15] := x$



- 流图的结点是一些基本块
- 从基本块 $B$ 到基本块 $C$ 之间有一条边，当且仅当 $C$ 的第一个指令可能紧跟在 $B$ 的最后一条指令之后执行
  - ❖  $B$ 是 $C$ 的前驱 (predecessor)
  - ❖  $C$ 是 $B$ 的后继 (successor)



- 流图的结点是一些基本块
- 从基本块 $B$ 到基本块 $C$ 之间有一条边，当且仅当 $C$ 的第一个指令可能紧跟在 $B$ 的最后一条指令之后执行，**判定方法如下**：
  - ❖ 有一个从 $B$ 的结尾跳转到 $C$ 的开头的跳转指令
  - ❖ 参考原来三地址指令序列中的顺序， $C$ 紧跟在 $B$ 之后，且 $B$ 的结尾没有无条件跳转指令

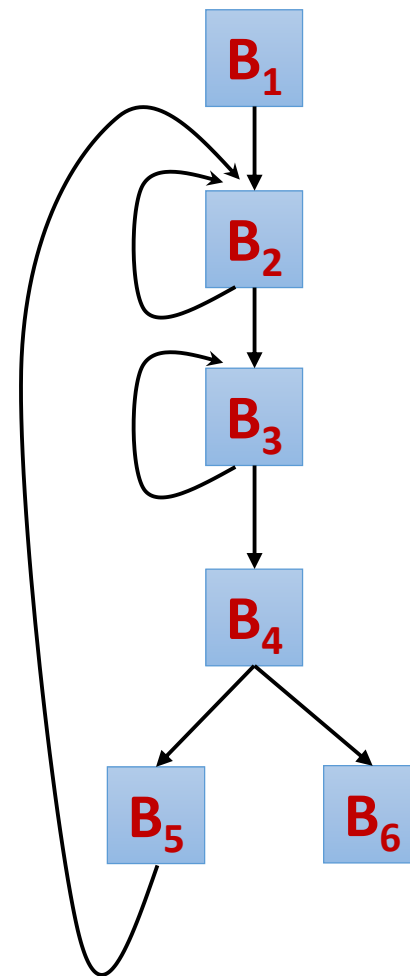


# 举例——流图



(1)  $i := m - 1$   
(2)  $j := n$   
 $B_1$  (3)  $t1 := 4 * n$   
(4)  $v := a[t1]$   
-----  
(5)  $i := i + 1$   
 $B_2$  (6)  $t2 := 4 * i$   
(7)  $t3 := a[t2]$   
(8) if  $t3 < v$  goto (5)  
-----  
(9)  $j := j - 1$   
 $B_3$  (10)  $t4 := 4 * j$   
(11)  $t5 := a[t4]$   
(12) if  $t5 > v$  goto (9)  
-----  
 $B_4$  (13) if  $i \geq j$  goto (23)  
(14)  $t6 := 4 * i$   
(15)  $x := a[t6]$

(16)  $t7 := 4 * i$   
(17)  $t8 := 4 * j$   
(18)  $t9 := a[t8]$   
(19)  $a[t7] := t9$   
 $B_5$  (20)  $t10 := 4 * j$   
(21)  $a[t10] := x$   
(22) goto (5)  
-----  
(23)  $t11 := 4 * i$   
(24)  $x := a[t11]$   
(25)  $t12 := 4 * i$   
 $B_6$  (26)  $t13 := 4 * n$   
(27)  $t14 := a[t13]$   
(28)  $a[t12] := t14$   
(29)  $t15 := 4 * n$   
(30)  $a[t15] := x$



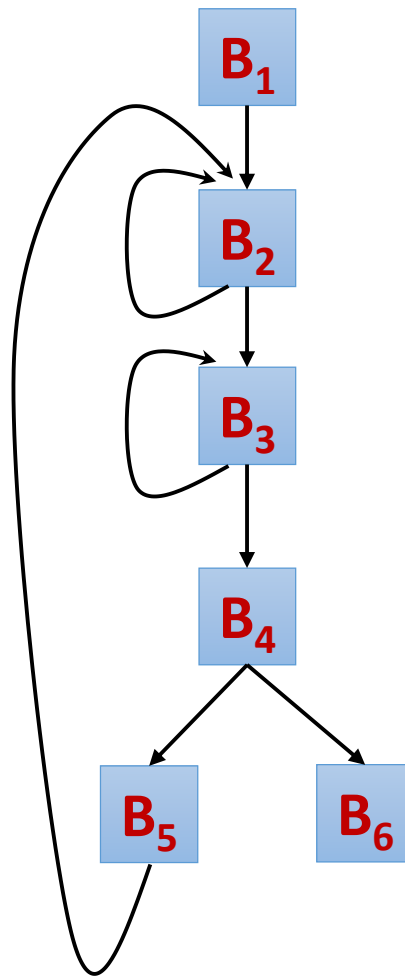
□ 流图中的一个结点集合L是一个循环，如果它满足：

- ❖ 该集合有唯一的入口结点
- ❖ 任意结点都有一个到达入口结点的非空路径，且该路径全部在L中

□ 不包含其他循环的循环叫做内循环

□ 右图中的循环

- ❖  $B_2$  自身
- ❖  $B_3$  自身
- ❖  $\{B_2, B_3, B_4, B_5\}$





## □基本块与流图

## □优化策略

- ❖ 公共子表达式删除(common subexpression elimination)
- ❖ 复制传播(copy propagation)
- ❖ 常量合并(constant folding)
- ❖ 死代码删除(dead code elimination)
- ❖ 代码移动(code motion)
- ❖ 强度削弱(strength reduction)
- ❖ 删除归纳变量(induction variable elimination)



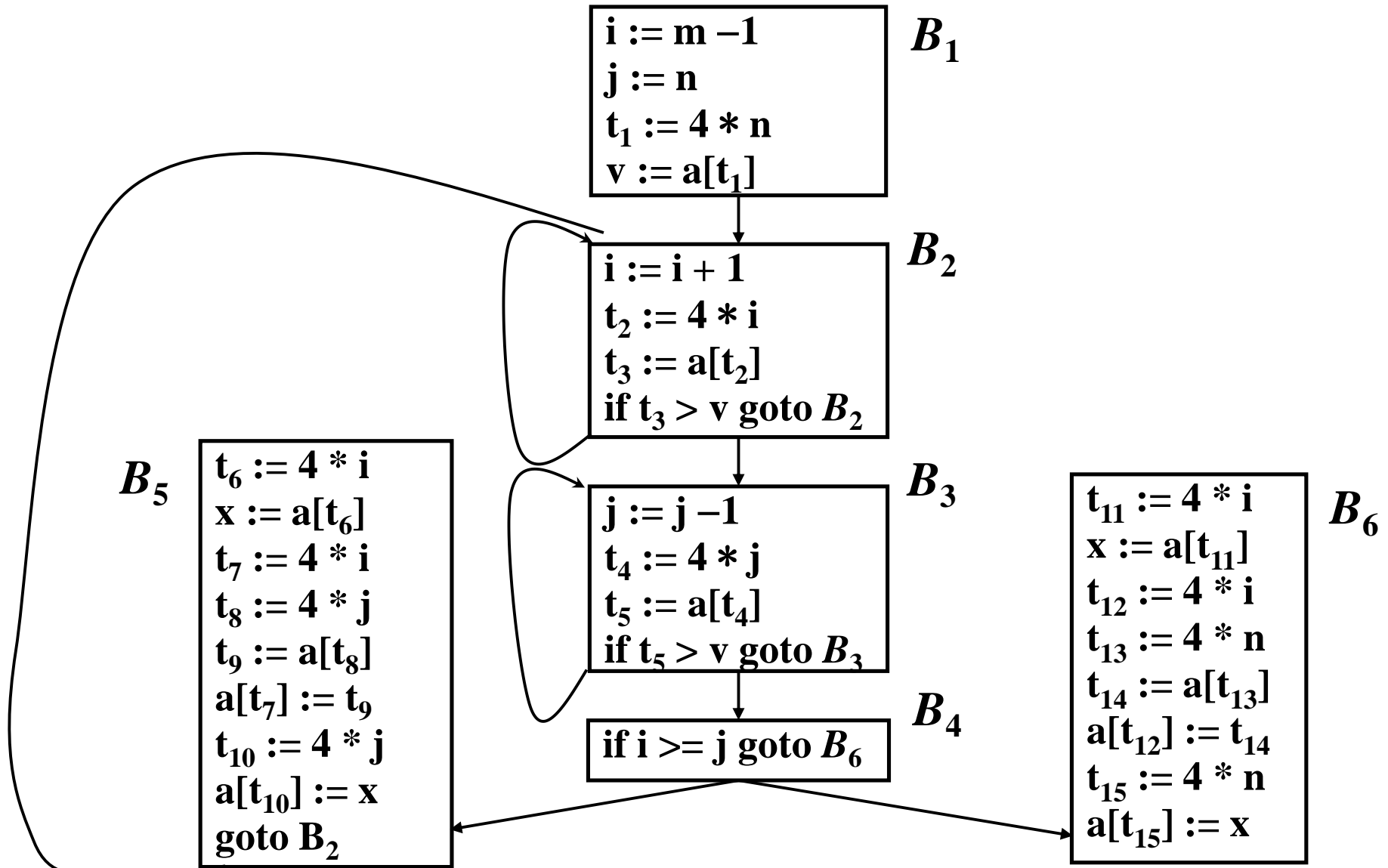


## □公共子表达式

❖如果表达式 $x \text{ op } y$ 之前已经被计算过，并且到现在为止， $x \text{ op } y$ 中的变量的值没有发生改变，那么该表达式的本次出现就成为公共子表达式

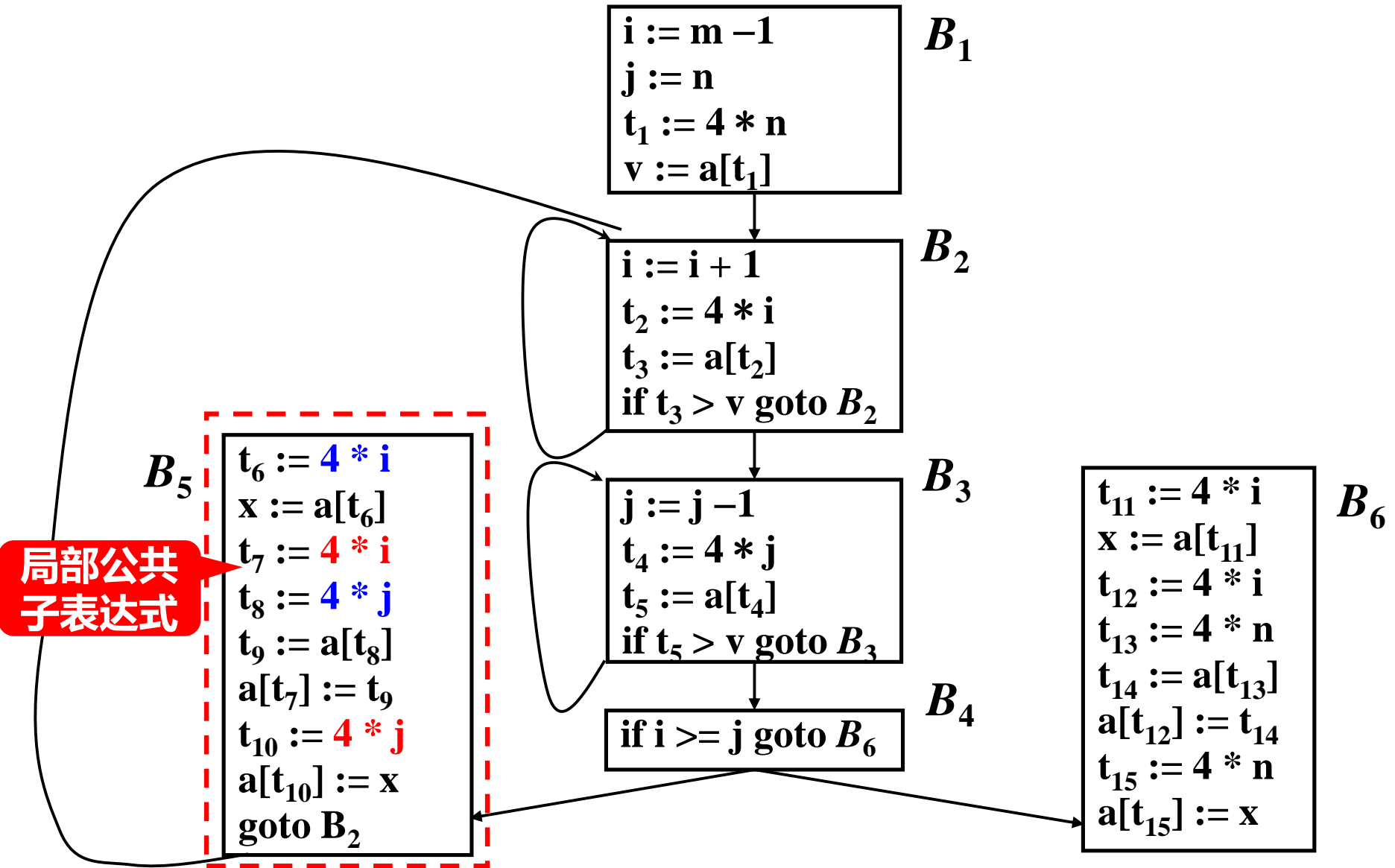


# 优化举例-公共子表达式删除



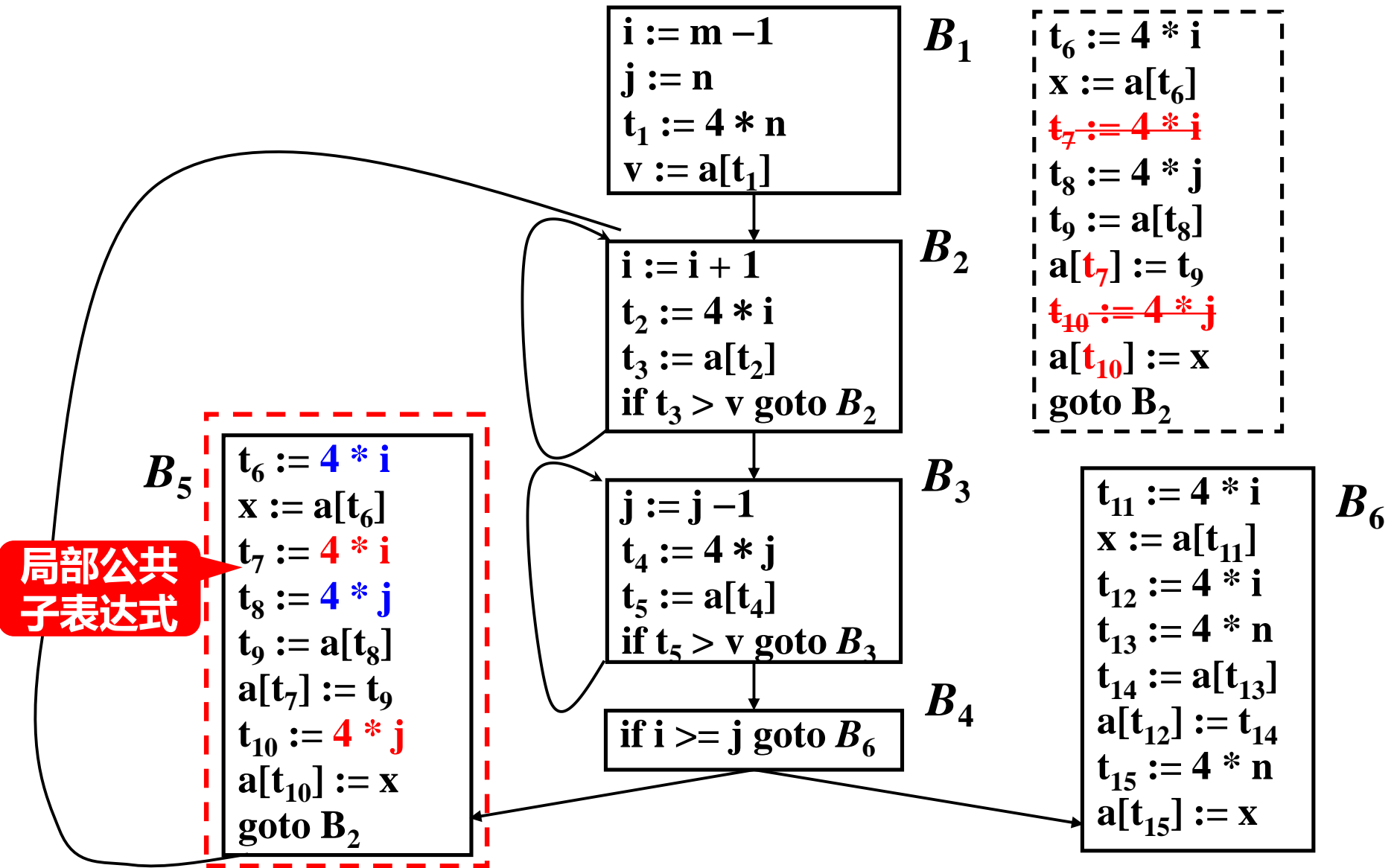


# 优化举例-公共子表达式删除



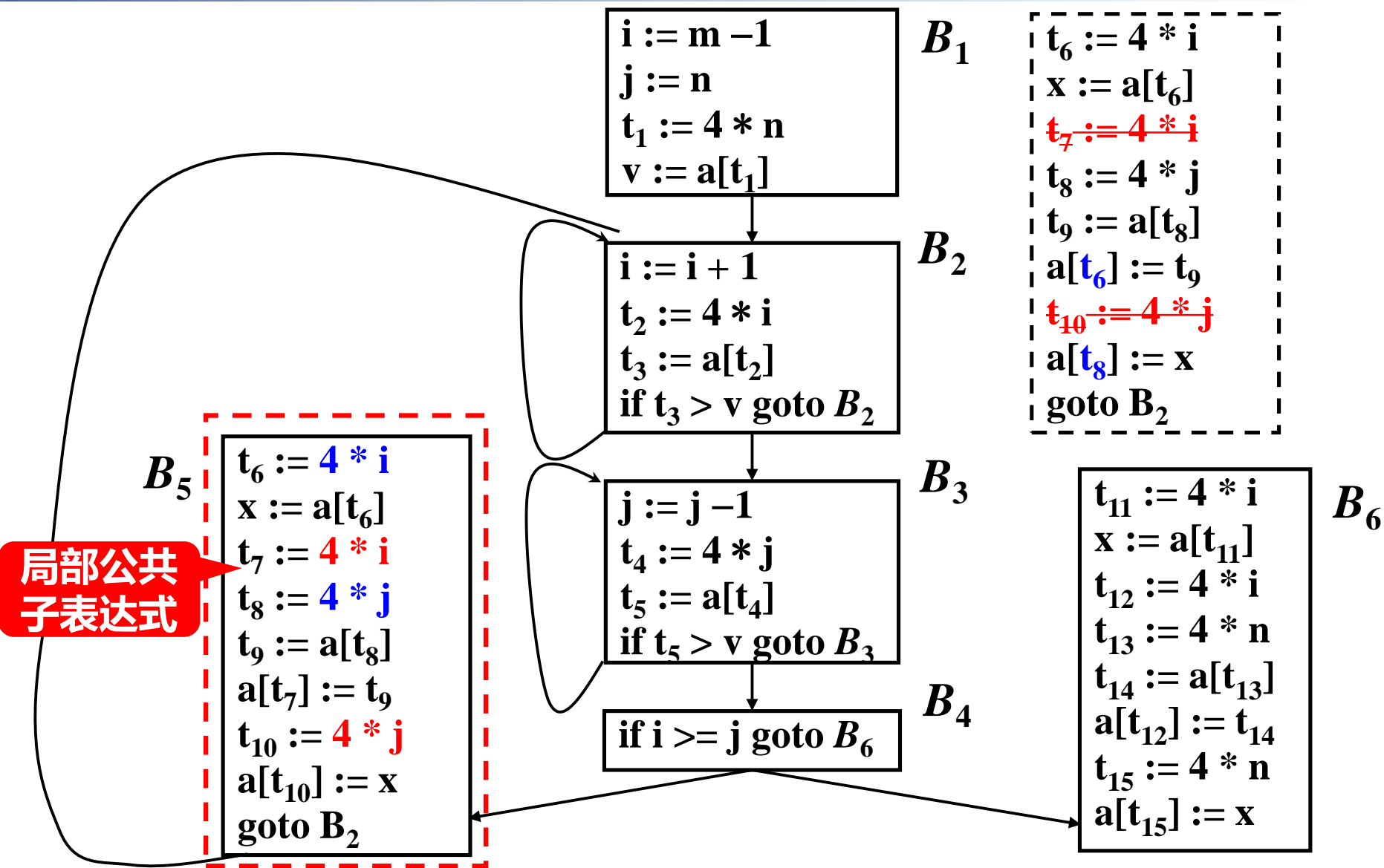


# 优化举例-公共子表达式删除



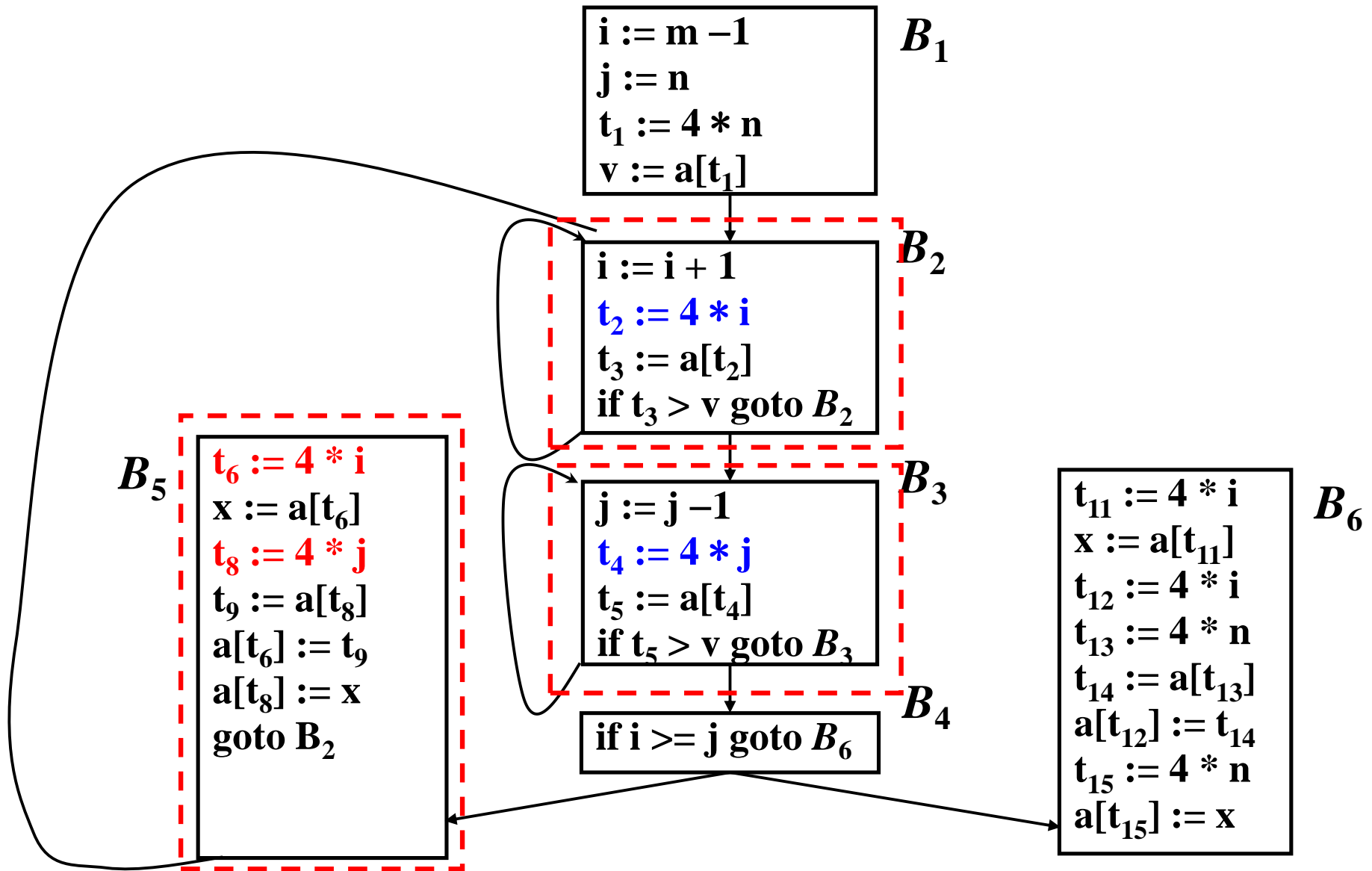


# 优化举例-公共子表达式删除



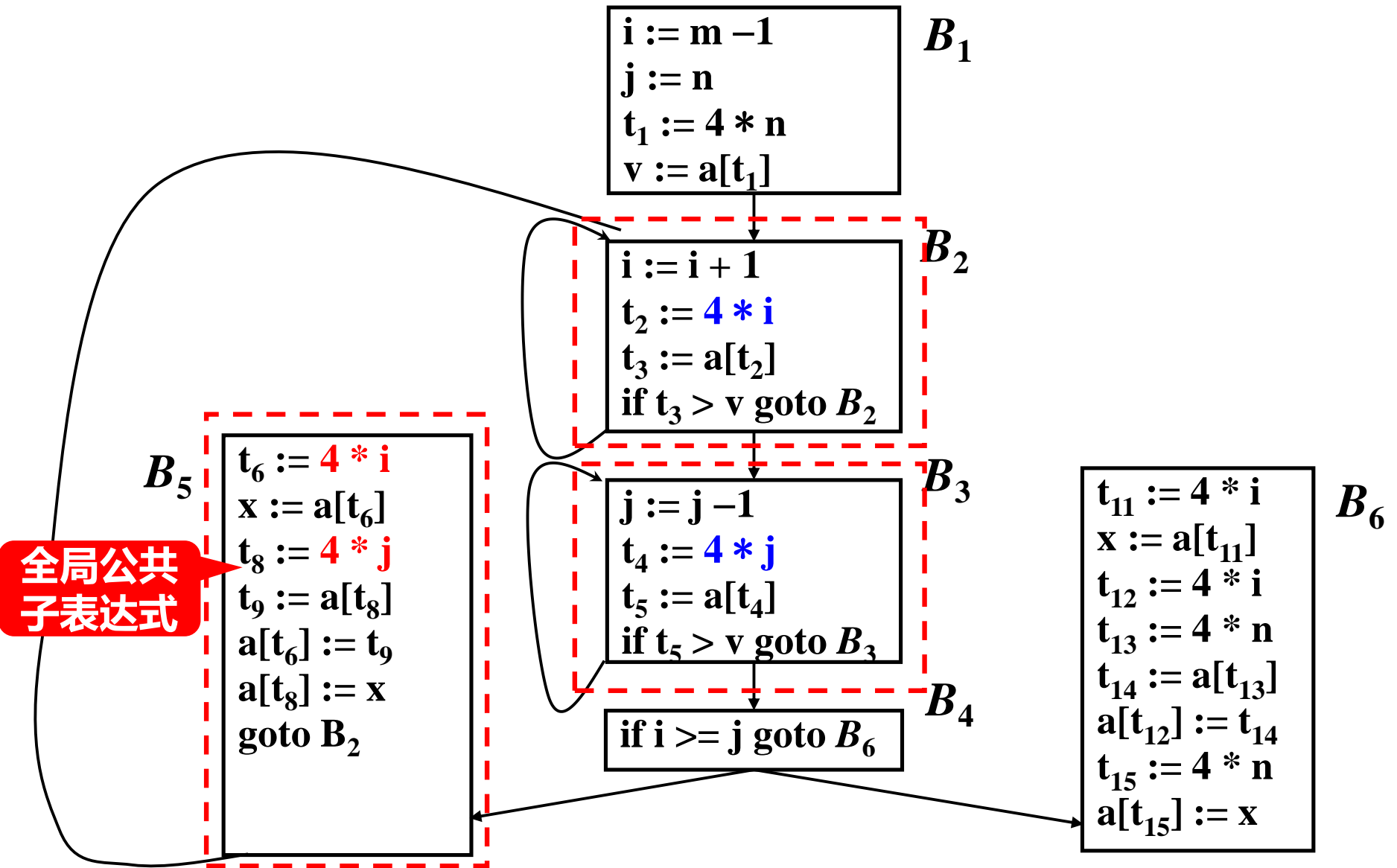


# 优化举例-公共子表达式删除



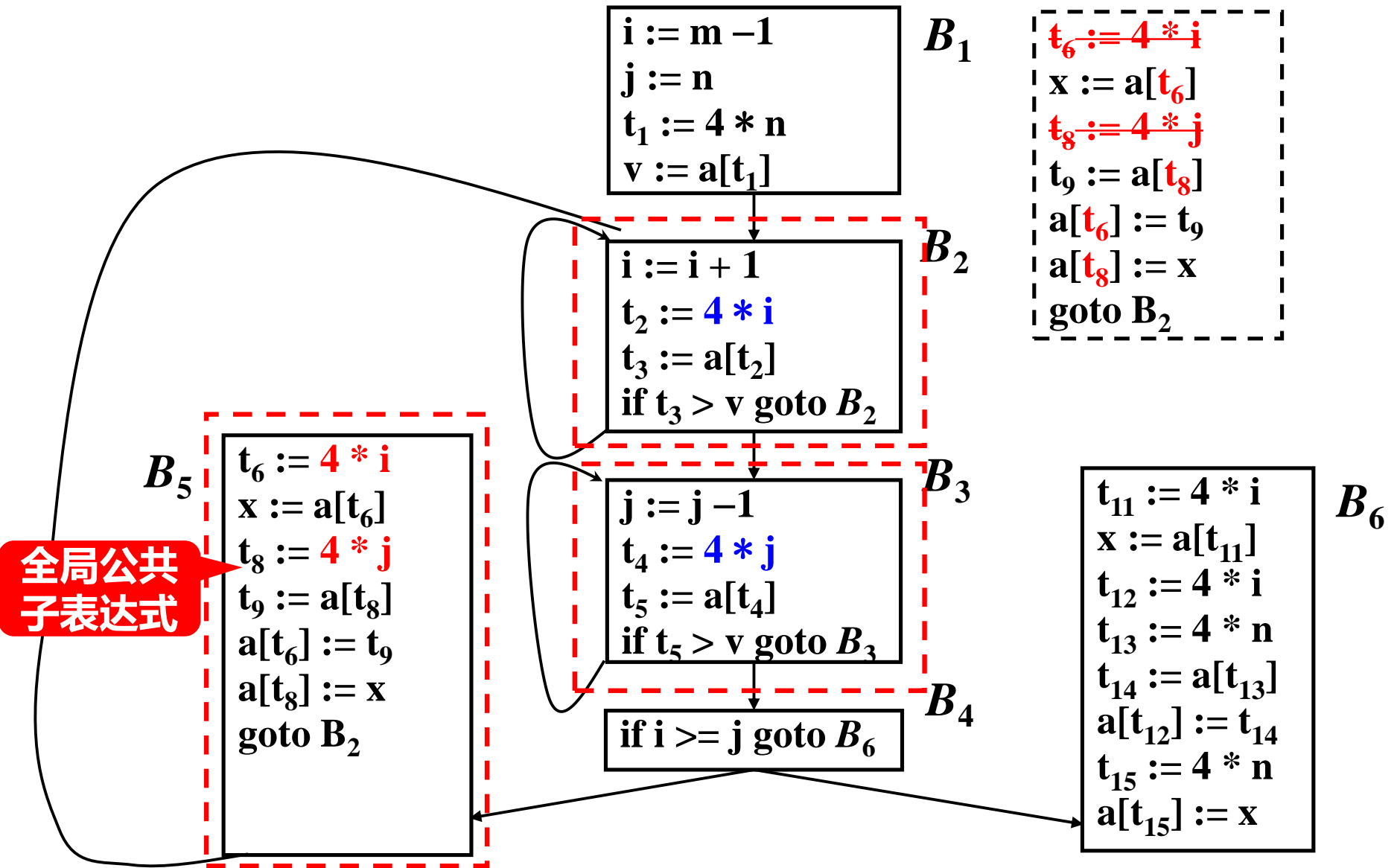


# 优化举例-公共子表达式删除





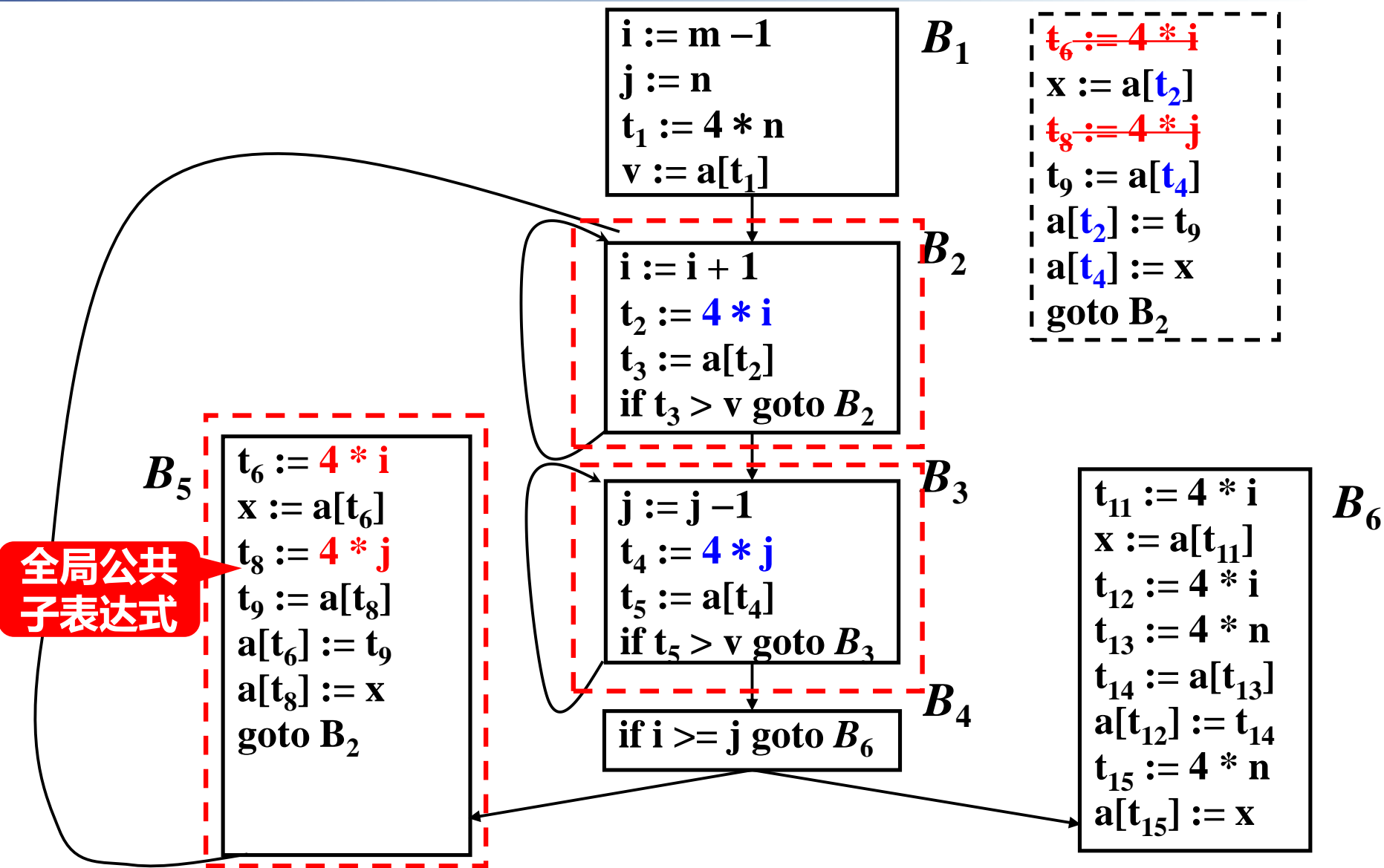
# 优化举例-公共子表达式删除





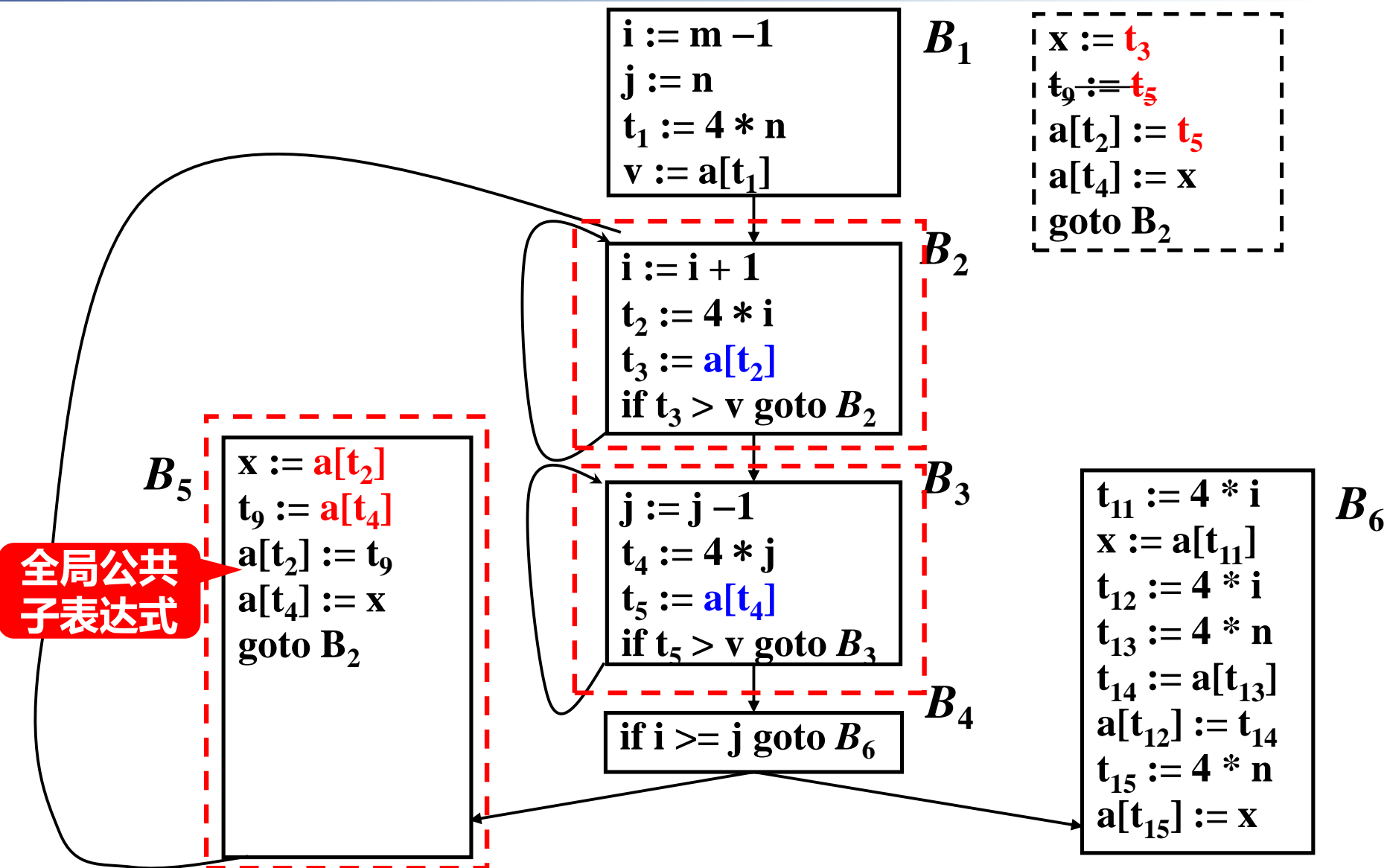


# 优化举例-公共子表达式删除



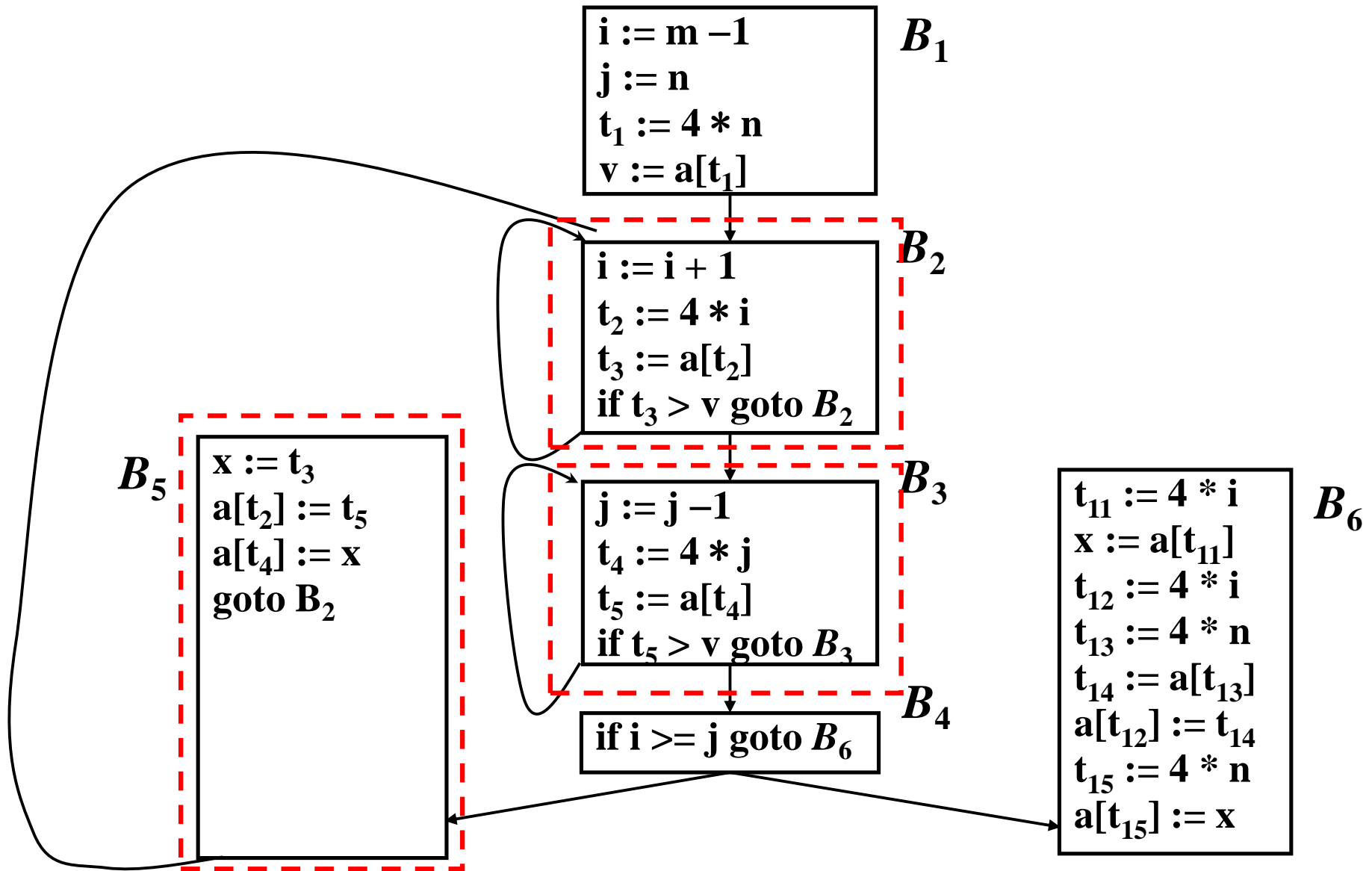


# 优化举例-公共子表达式删除



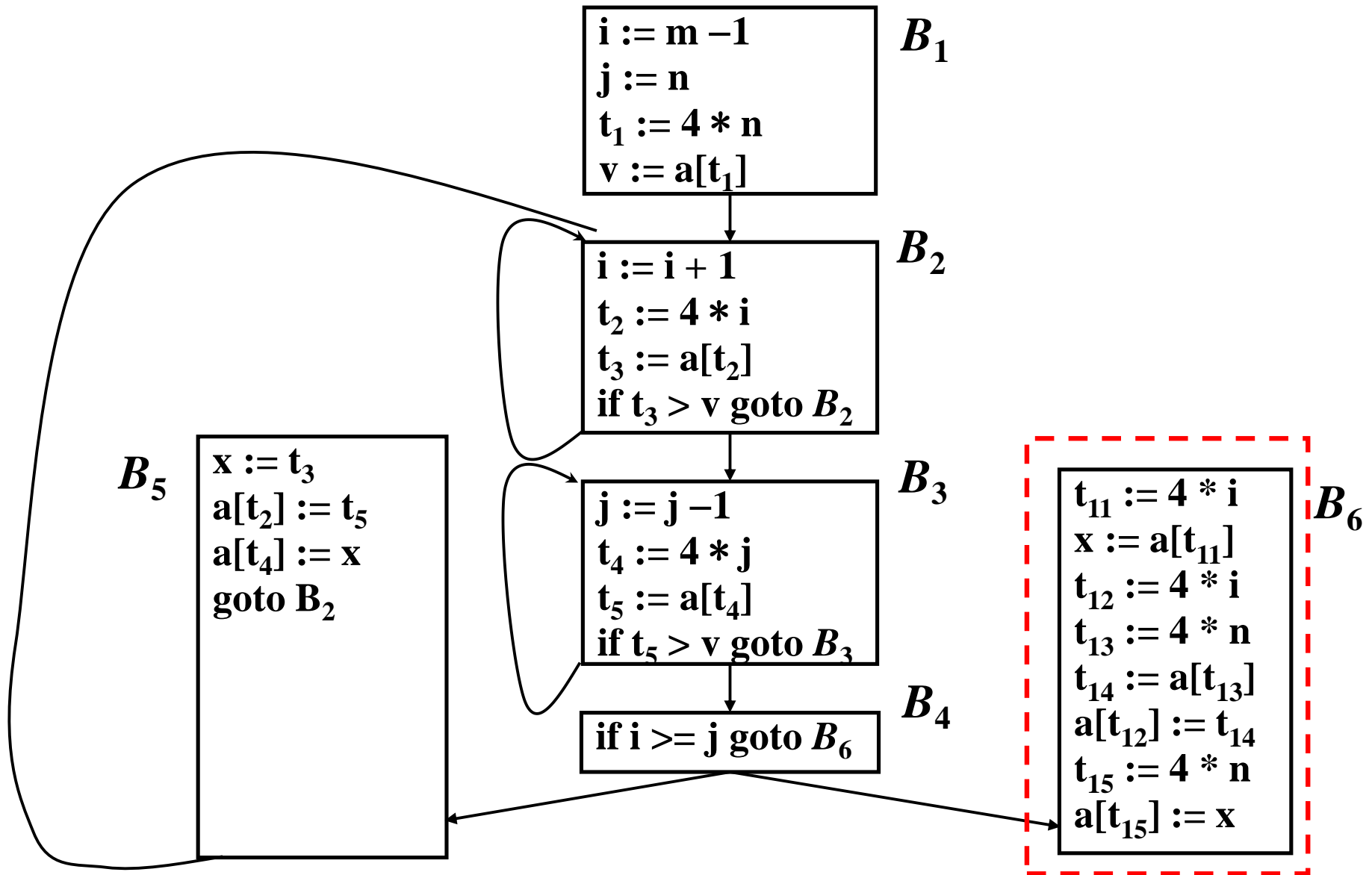


# 优化举例-公共子表达式删除



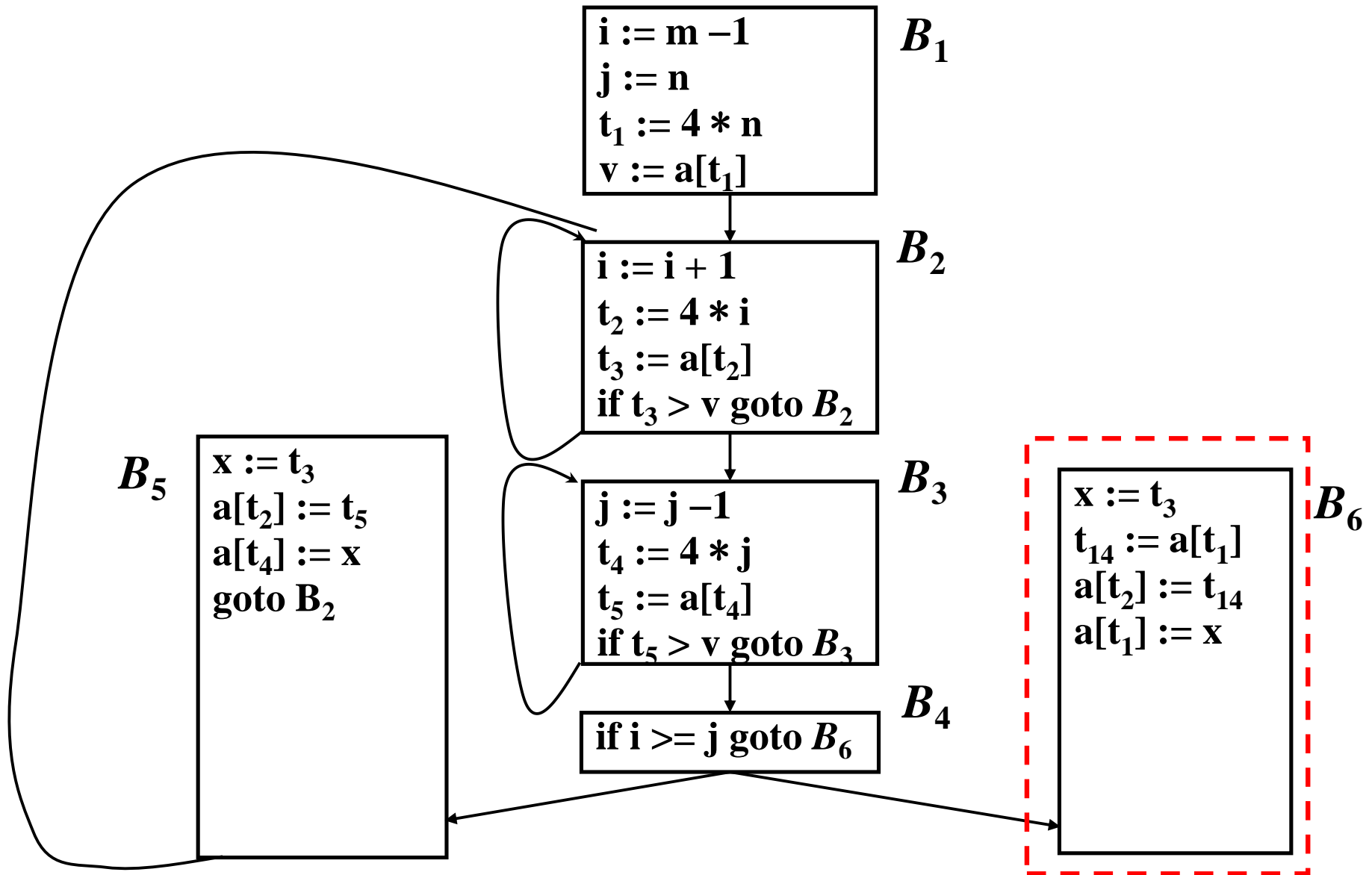


# 优化举例-公共子表达式删除



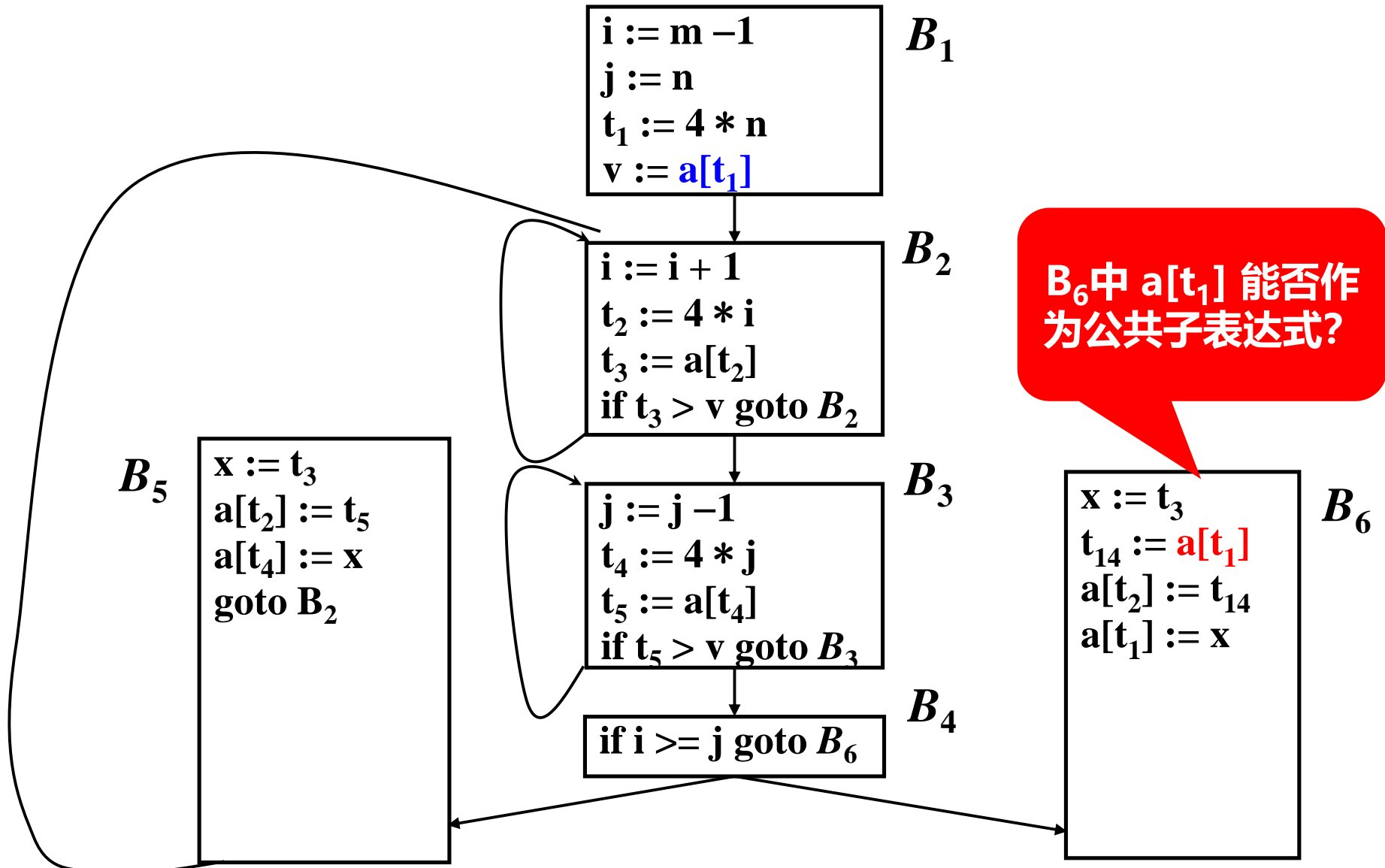


# 优化举例-公共子表达式删除



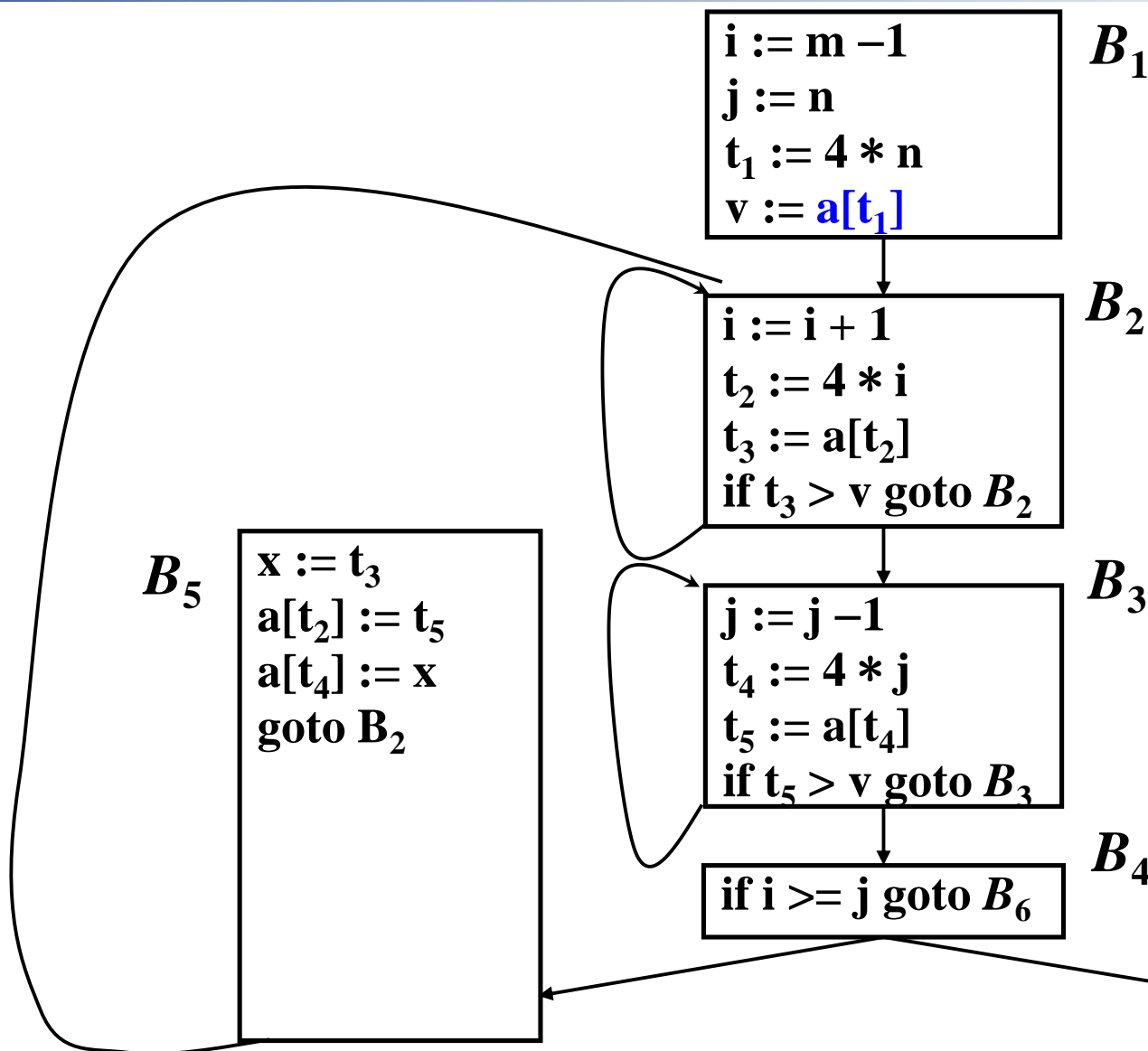


# 优化举例-公共子表达式删除





# 优化举例-公共子表达式删除



把 $a[t_1]$ 作为公共子表达式是不稳妥的

因为 $B_5$ 有对下标变量 $a[t_2]$ 和 $a[t_4]$ 的赋值



## □基本块与流图

## □优化策略

❖公共子表达式删除(common subexpression elimination)

❖复制传播(copy propagation)

❖常量合并(constant folding)

❖死代码删除(dead code elimination)

❖代码移动(code motion)

❖强度削弱(strength reduction)

❖删除归纳变量(induction variable elimination)

打包  
一起讲





□死代码是指计算的结果永远不被引用的语句

例： 为便于调试，可能在程序中加打印语句，测试后改成右边的形式

<code>debug = true;</code>		<code>debug = false;</code>
<code>...</code>		<code>...</code>
<code>if (debug) print ...</code>		<code>if (debug) print ...</code>



□死代码是指计算的结果永远不被引用的语句

□一些优化变换可能会引起死代码

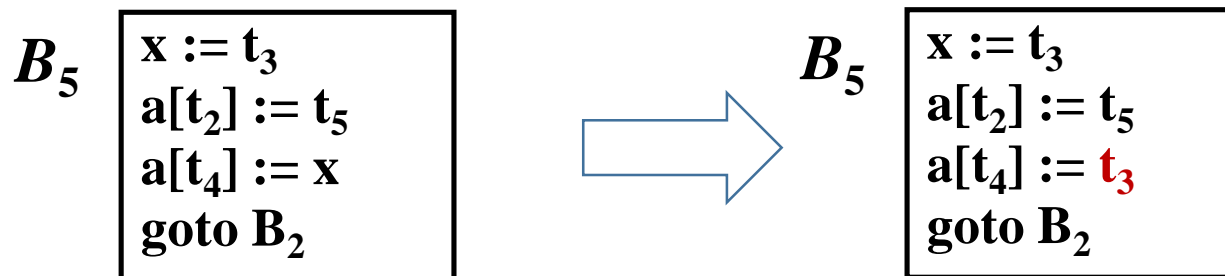
❖如：复制传播、常量合并

例： 为便于调试，可能在程序上加打印语句，测试后改成右边的形式

debug = true;		debug = false;
. . .		. . .
if (debug) print ...		if (debug) print ...

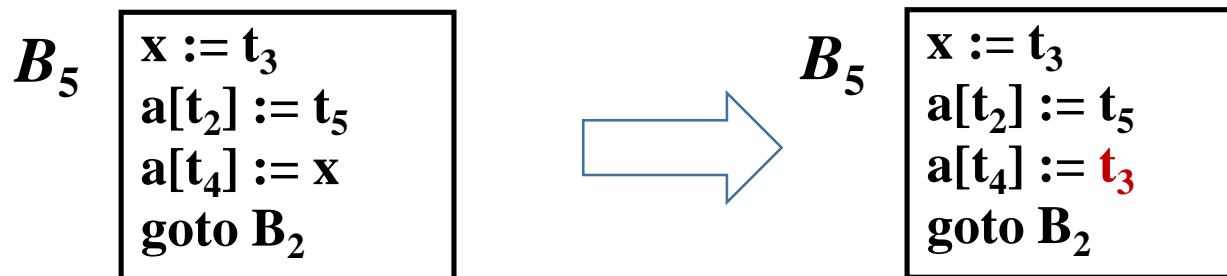


□定义：在复制语句 $x = y$ 之后尽可能用 $y$ 代替 $x$





□定义：在复制语句 $x = y$ 之后尽可能用 $y$ 代替 $x$



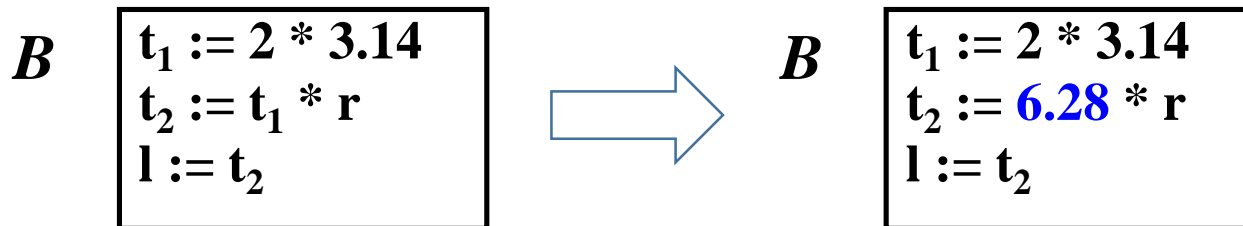
□常用的公共子表达式删除和其他一些优化会引入一些复制语句

□复制传播本身没有优化的意义，但可以给死代码删除创造机会



□如果在**编译时刻**推导出一个表达式的值是**常量**，就可以使用该常量来代替这个表达式。

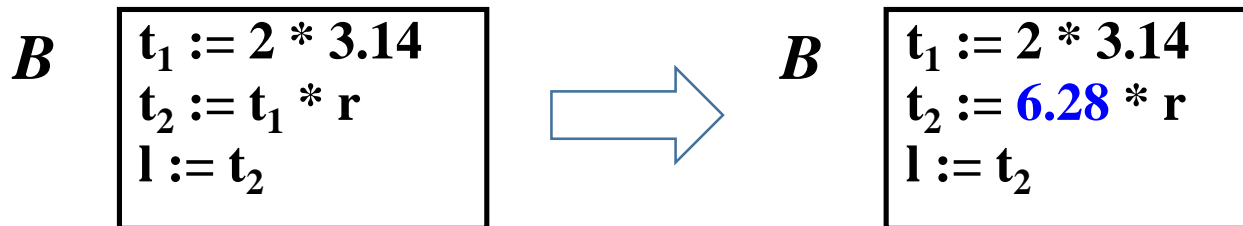
❖例：计算圆周长的表达式  $l = 2 * 3.14 * r$





□如果在**编译时刻**推导出一个表达式的值是**常量**，就可以使用该常量来代替这个表达式。

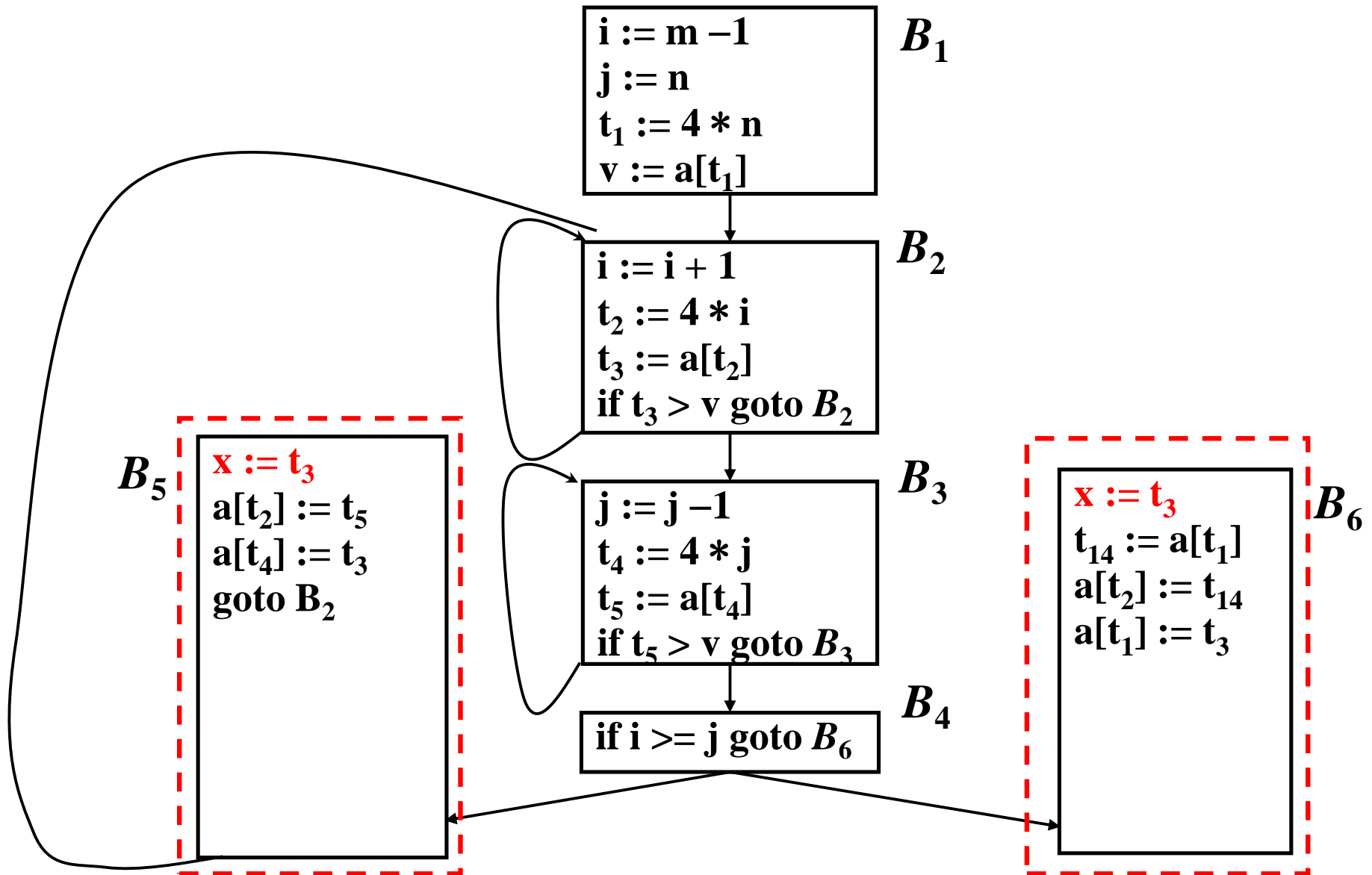
❖例：计算圆周长的表达式  $l = 2 * 3.14 * r$



□常量合并本身没有优化的意义，**但可以给死代码删除创造机会**

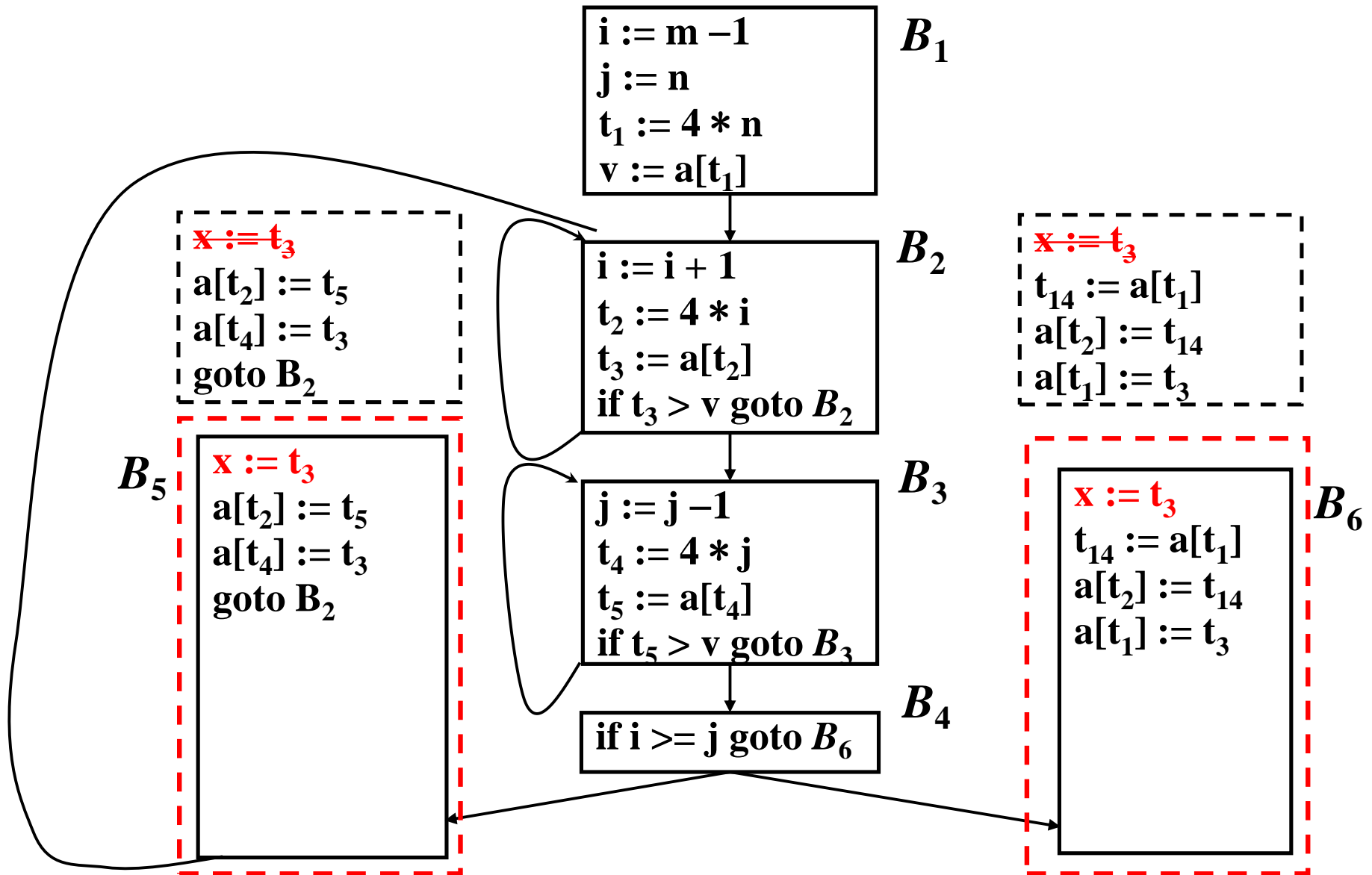


# 优化举例-死代码删除





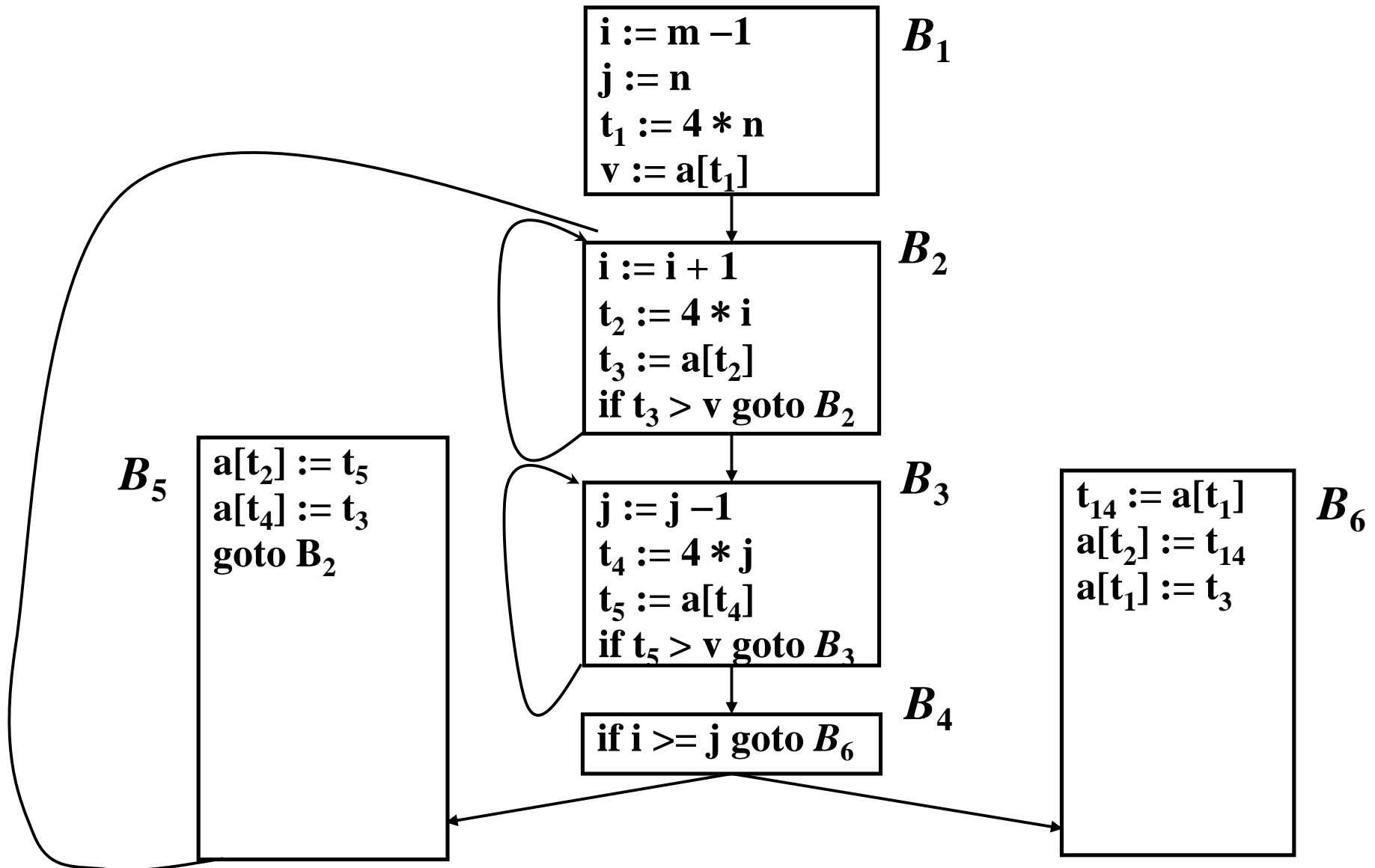
# 优化举例-死代码删除







# 优化举例-死代码删除





## □基本块与流图

## □优化策略

- ❖ 公共子表达式删除(common subexpression elimination)
- ❖ 复制传播(copy propagation)
- ❖ 常量合并(constant folding)
- ❖ 死代码删除(dead code elimination)
- ❖ 代码移动(code motion)
- ❖ 强度削弱(strength reduction)
- ❖ 删除归纳变量(induction variable elimination)

涉及循环优化打包一起讲



- 循环不变计算 (**loop-invariant computation**) 是指  
不管循环执行多少次都得到相同结果的表达式
- 代码移动是**循环优化**的一种，在进入循环前  
就对循环不变计算进行求值



- 循环不变计算 (**loop-invariant computation**) 是指  
不管循环执行多少次都得到相同结果的表达式
- 代码移动是**循环优化**的一种，在进入循环前  
就对循环不变计算进行求值

例：while (  $i \leq \text{limit} - 2$  ) ...

代码移动后变换成

$t = \text{limit} - 2;$

while (  $i \leq t$  ) ...



- 循环不变计算 (**loop-invariant computation**) 是指不管循环执行多少次都得到相同结果的表达式
- 代码移动是**循环优化**的一种，在进入循环前就对循环不变计算进行求值。
- 对于多重嵌套循环， **loop-invariant computation**是相对于某一个循环的，可能对于更加外层的循环，它就不成立了。
- 因此，处理循环时，按照由里到外的方式



## □用较快的操作代替较慢的操作

❖如用加代替乘

❖课外阅读：Booth算法

$2 * x$  或者  $2.0 * x$

$x/2$

$x^2$

$a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0$



$x + x$

$x * 0.5$

$x * x$

$((\dots (a_n x + a_{n-1})x + a_{n-2}) \dots)x + a_1) + a_0$

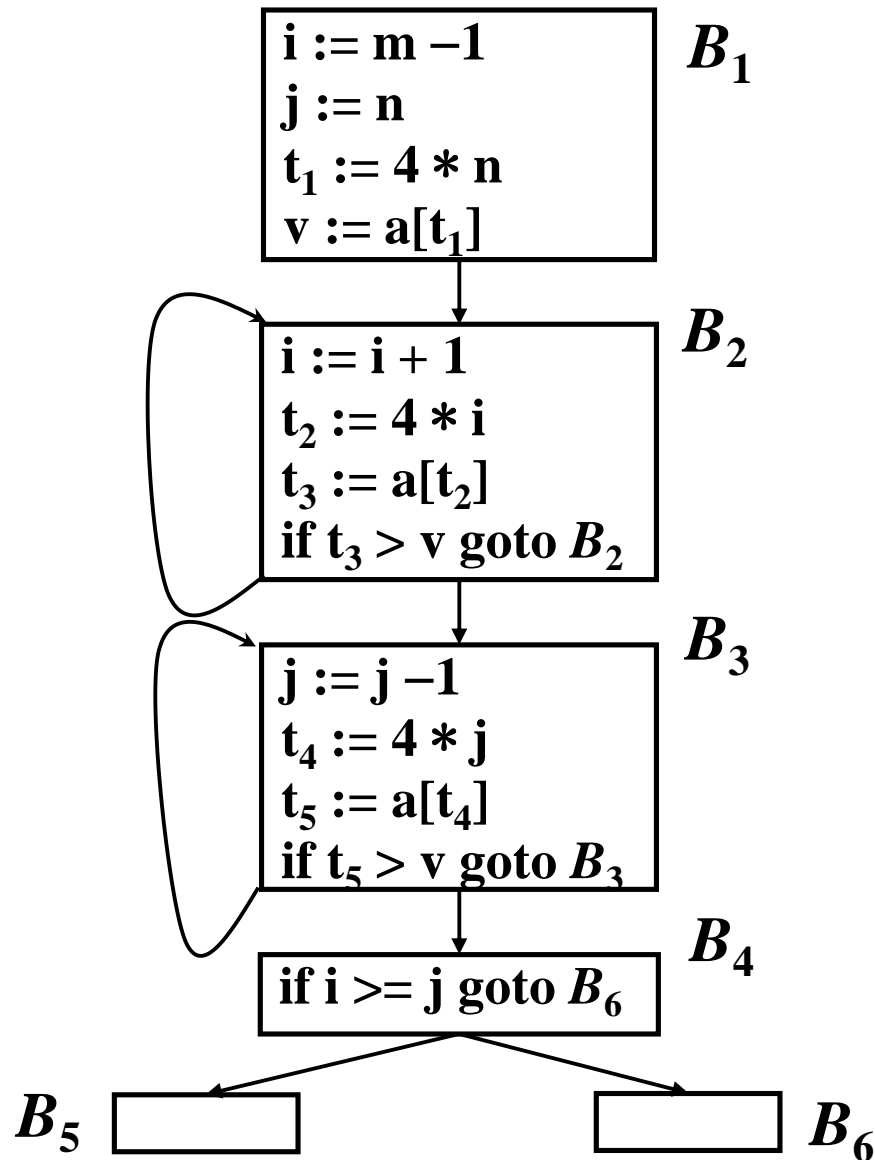


## □归纳变量:

❖ 对于一个变量 $x$ ，如果存在一个正的或者负的常数 $c$ 使得每次 $x$ 被赋值时它的值总增加 $c$ ，那么 $x$ 就成为归纳变量(Induction variable)



# 优化举例-循环中的强度削弱







# 优化举例-循环中的强度削弱



中国科学技术大学  
University of Science and Technology of China

$B_1$   
 $i := m - 1$   
 $j := n$   
 $t_1 := 4 * n$   
 $v := a[t_1]$

$B_2$   
 $i := i + 1$   
 $t_2 := 4 * i$   
 $t_3 := a[t_2]$   
if  $t_3 > v$  goto  $B_2$

$B_3$   
 $j := j - 1$   
 $t_4 := 4 * j$   
 $t_5 := a[t_4]$   
if  $t_5 > v$  goto  $B_3$

$B_4$   
if  $i \geq j$  goto  $B_6$

$B_5$

$B_6$



$B_1$   
 $i := m - 1$   
 $j := n$   
 $t_1 := 4 * n$   
 $v := a[t_1]$

$B_2$   
 $i := i + 1$   
 $t_2 := t_2 + 4$   
 $t_3 := a[t_2]$   
if  $t_3 > v$  goto  $B_2$

$B_3$   
 $j := j - 1$   
 $t_4 := t_4 - 4$   
 $t_5 := a[t_4]$   
if  $t_5 > v$  goto  $B_3$

$B_4$   
if  $i \geq j$  goto  $B_6$

$B_5$

$B_6$



# 优化举例-循环中的强度削弱



中国科学技术大学  
University of Science and Technology of China

$B_1$   
 $i := m - 1$   
 $j := n$   
 $t_1 := 4 * n$   
 $v := a[t_1]$

$B_2$   
 $i := i + 1$   
 $t_2 := 4 * i$   
 $t_3 := a[t_2]$   
if  $t_3 > v$  goto  $B_2$

$B_3$   
 $j := j - 1$   
 $t_4 := 4 * j$   
 $t_5 := a[t_4]$   
if  $t_5 > v$  goto  $B_3$

$B_4$   
if  $i \geq j$  goto  $B_6$

$B_5$

$B_6$

11/25/2019

Cheng @ Compiler USIC, Fall 2019

$B_1$   
 $i := m - 1$   
 $j := n$   
 $t_1 := 4 * n$   
 $v := a[t_1]$   
 $t_2 := 4 * i$   
 $t_4 := 4 * j$

$B_2$   
 $i := i + 1$   
 $t_2 := t_2 + 4$   
 $t_3 := a[t_2]$   
if  $t_3 > v$  goto  $B_2$

$B_3$   
 $j := j - 1$   
 $t_4 := t_4 - 4$   
 $t_5 := a[t_4]$   
if  $t_5 > v$  goto  $B_3$

$B_4$   
if  $i \geq j$  goto  $B_6$

$B_5$

$B_6$

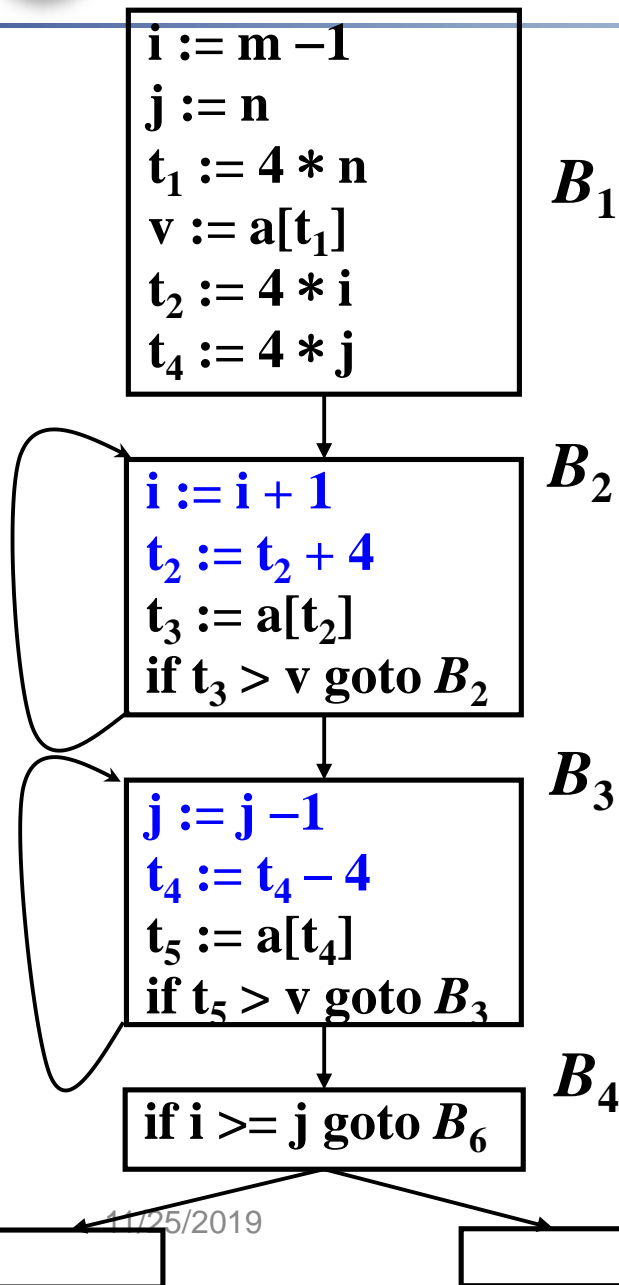
50



- ❖ 在程序循环的执行过程中，如果有一组归纳变量的值的变化保持步调一致，通常可以只保留一个。
- ❖ 强度削弱可以给删除归纳变量提供机会。



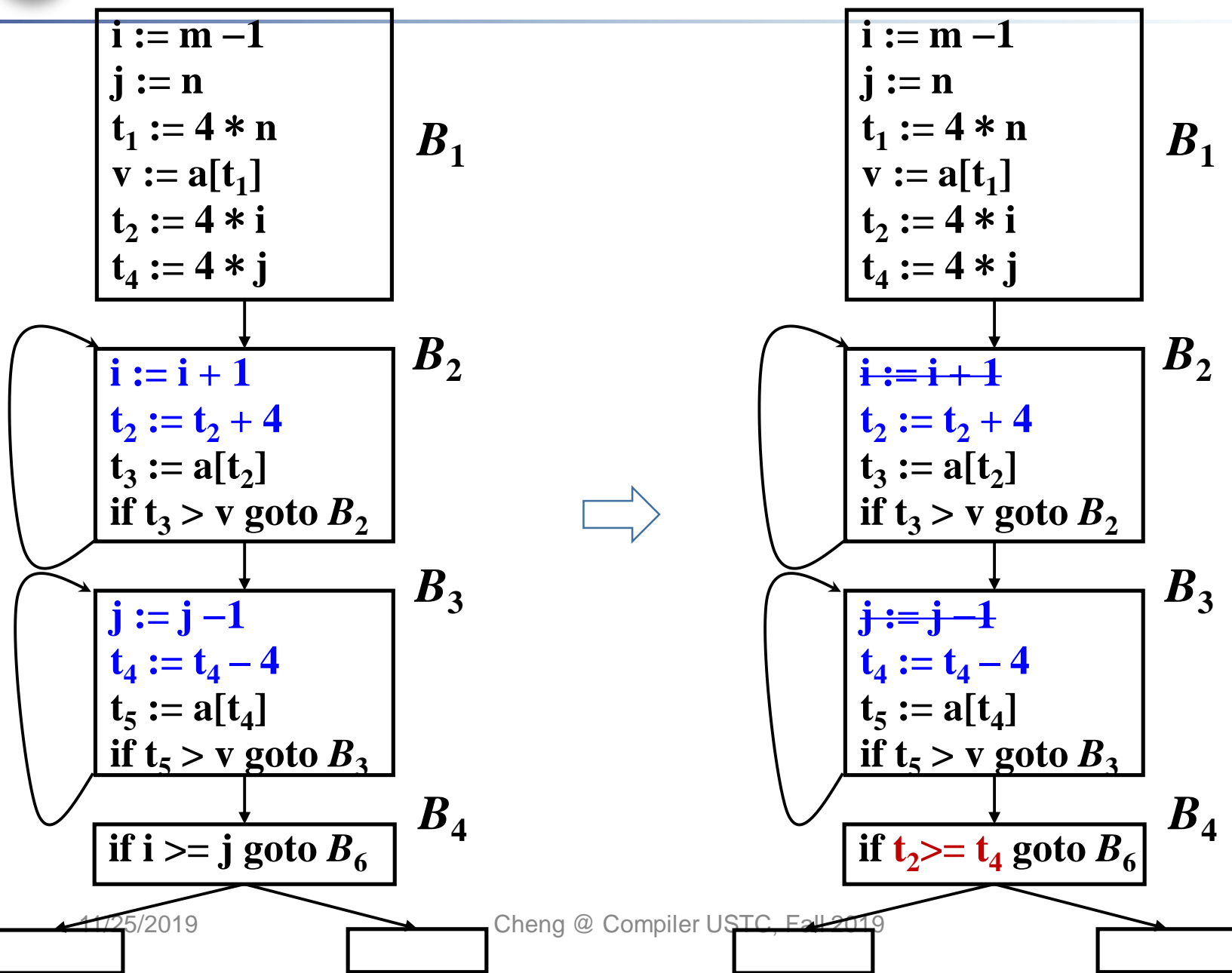
# 优化举例-删除归纳变量



- ❖ 在程序循环的执行过程中，如果有一组归纳变量的值的变化保持步调一致，通常可以只保留一个。
- ❖ 强度削弱可以给删除归纳变量提供机会。



# 优化举例-删除归纳变量





# 《编译原理与技术》

## 独立于机器的优化

**At the end, if you fail, at least you did something interesting, rather than doing something boring and also failing. Or doing something boring and then forgetting how to do something interesting.**

**—— Barbara Liskov (Turing Award 2008)**