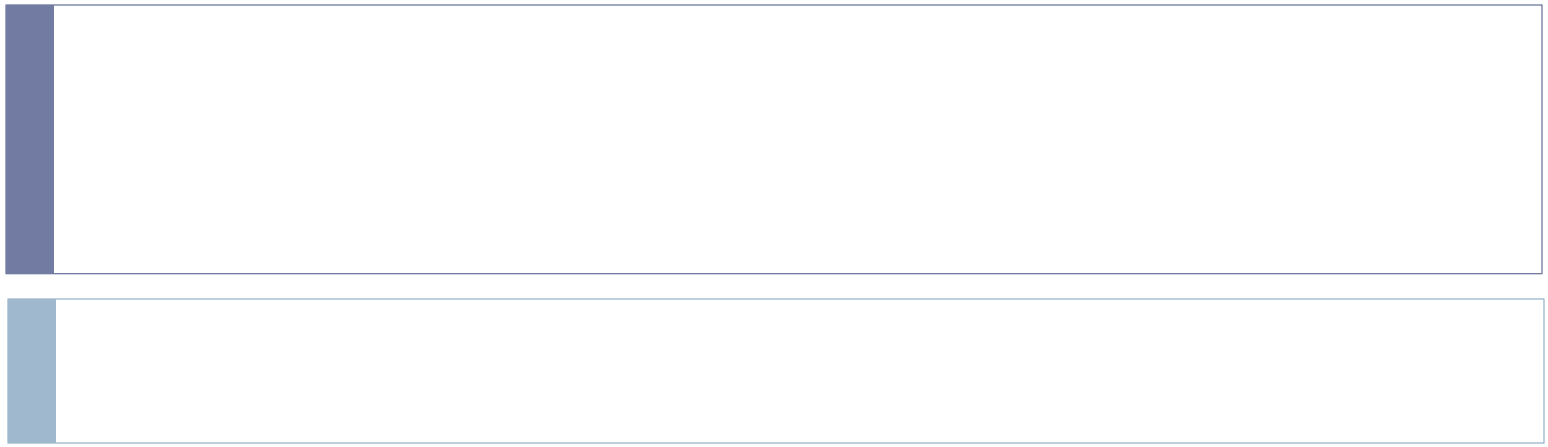
Chapter 9 优化和目标代码生成



内容与重点

- 内容
  - 概述
  - 优化技术
  - 局部优化
  - 循环优化

重点

优化原则

优化类型

优化技术

对程序进行各种等价变换，用于生成更高效的目标代码

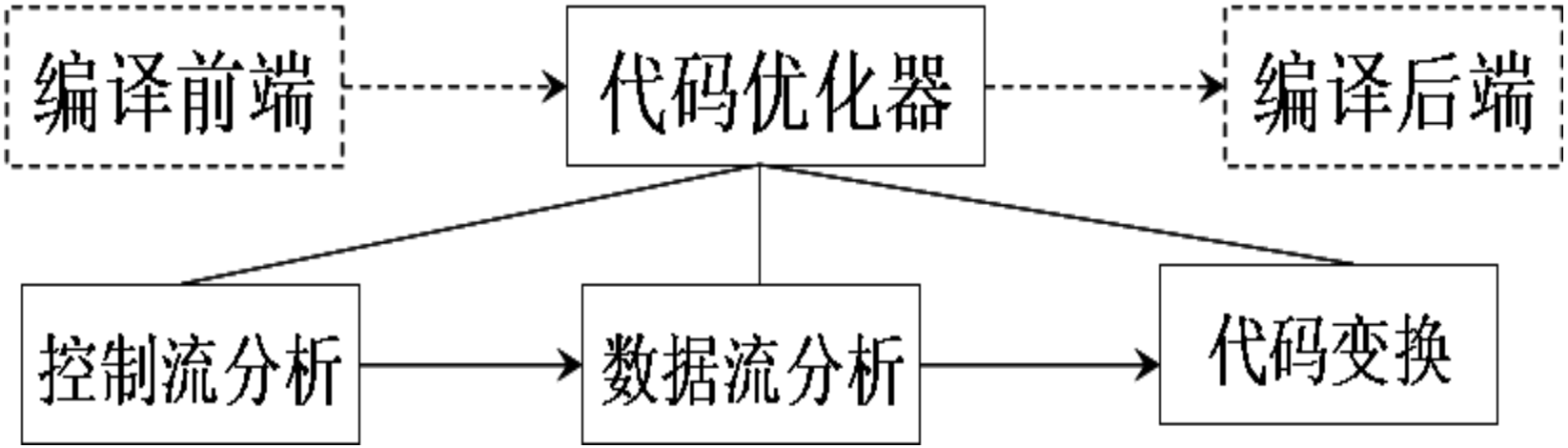
等价原则、有效原则和合算原则

依据优化所涉及的程序范围，可以分为局部优化、循环优化和全局优化三种类型

最常用的有删除公共子表达式、复写传播、删除无用代码、代码外提、强度削弱和删除归纳变量等

1.1 代码优化器

- 为了生成更高效的目标代码，对程序进行各种等价变换，这种变换称为优化



代码优化器的地位和结构

1.2 优化目的与优化原则

- 优化的目的
  - 产生更高效的代码
- 遵循一定的原则
  - 等价原则
    - 经过优化后不应改变程序运行的结果
  - 有效原则
    - 使优化后所产生的目标代码运行时间较短，占用的存储空间较小
  - 合算原则
    - 尽可能以较低的代价取得较好的优化效果

2 优化技术

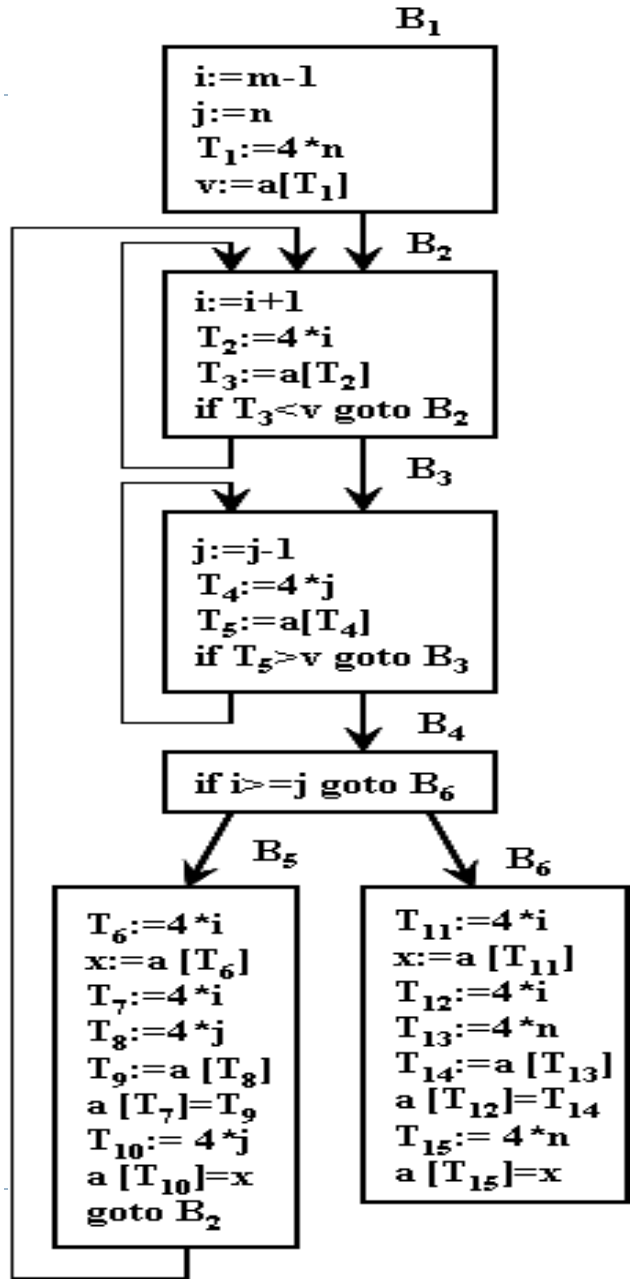
- 概述
- 优化技术
  - 删除公共子表达式
  - 复写传播
  - 删除无用代码
  - 代码外提
  - 强度削弱
  - 删除归纳变量
- 局部优化
- 循环优化

基本块

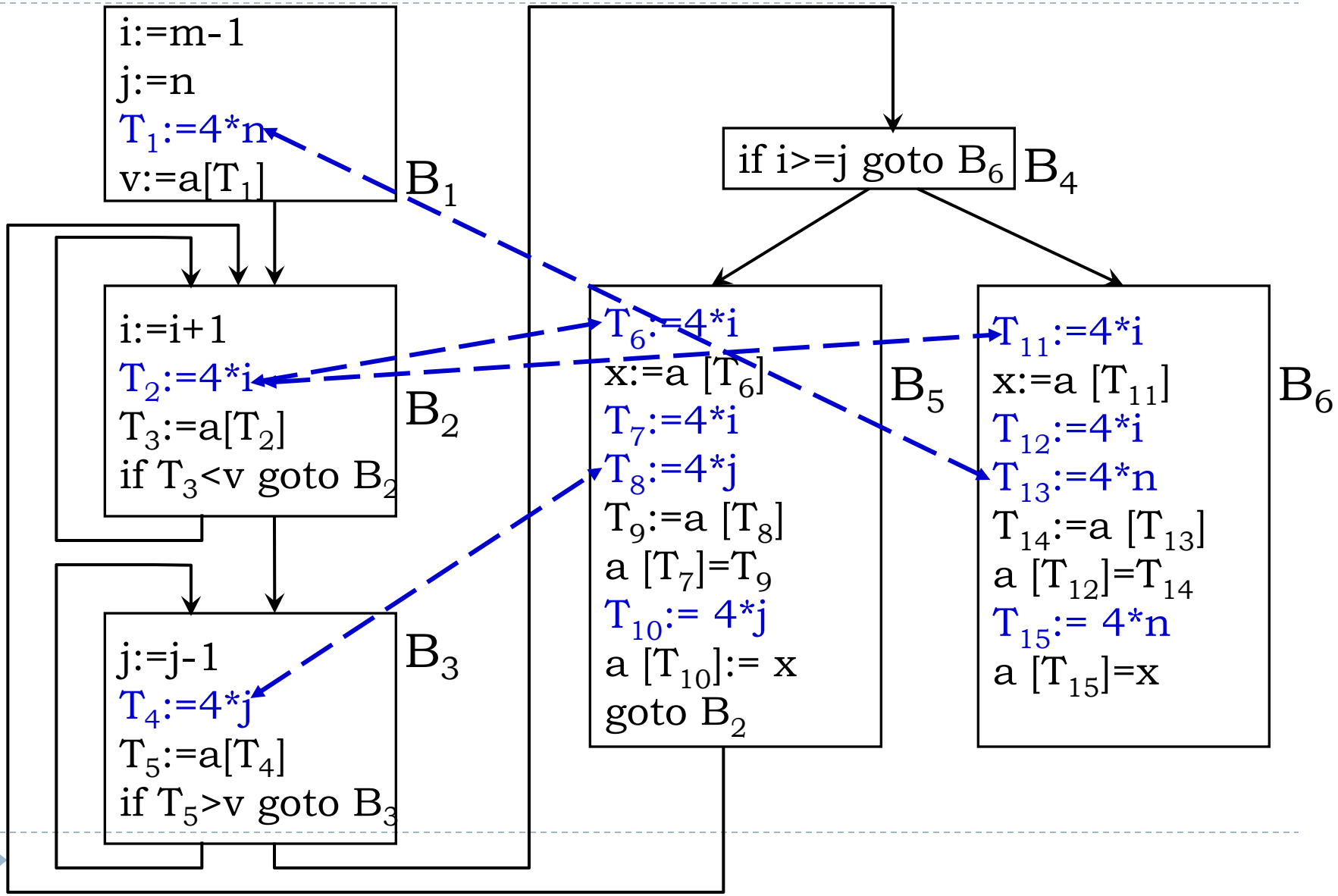
- 定义
  - 一段代码
  - 一个入口，一个出口
  - 入口为第一条语句
  - 出口为最后一条语句
- 计算入口
  - 程序第一条语句
  - 条件转移或者无条件转移到的语句
  - 条件转移的下一条语句
- 程序流图

Example

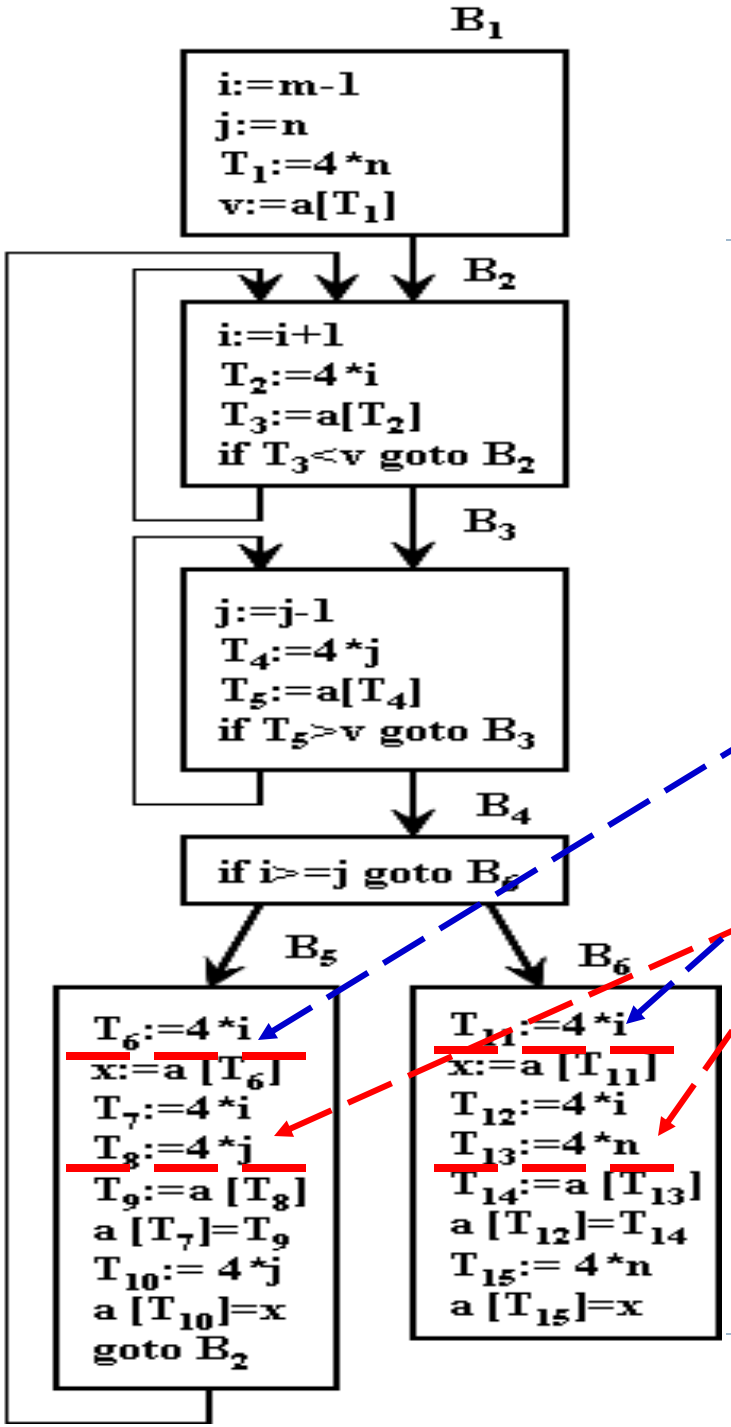
```
void quicksort (m, n);
int m, n;
{
        int i, j;
        int v, x;
        if (n<=m) return;
        /* fragment begins here*/
        i=m-1; j=n; v=a[n];
        while (1) {
                do  i=i+1;  while (a[i]<v);
                do  j=j-1;   while (a[j]>v);
                if (i>=j) break;
                x=a[i]; a[i]=a [j]; a[j]=x;
        }
        x=a[i]; a[i]=a[n]; a[n]=x;
        /*fragment ends here*/
        quicksort (m, j); quicksort (i+1, n);
}
```

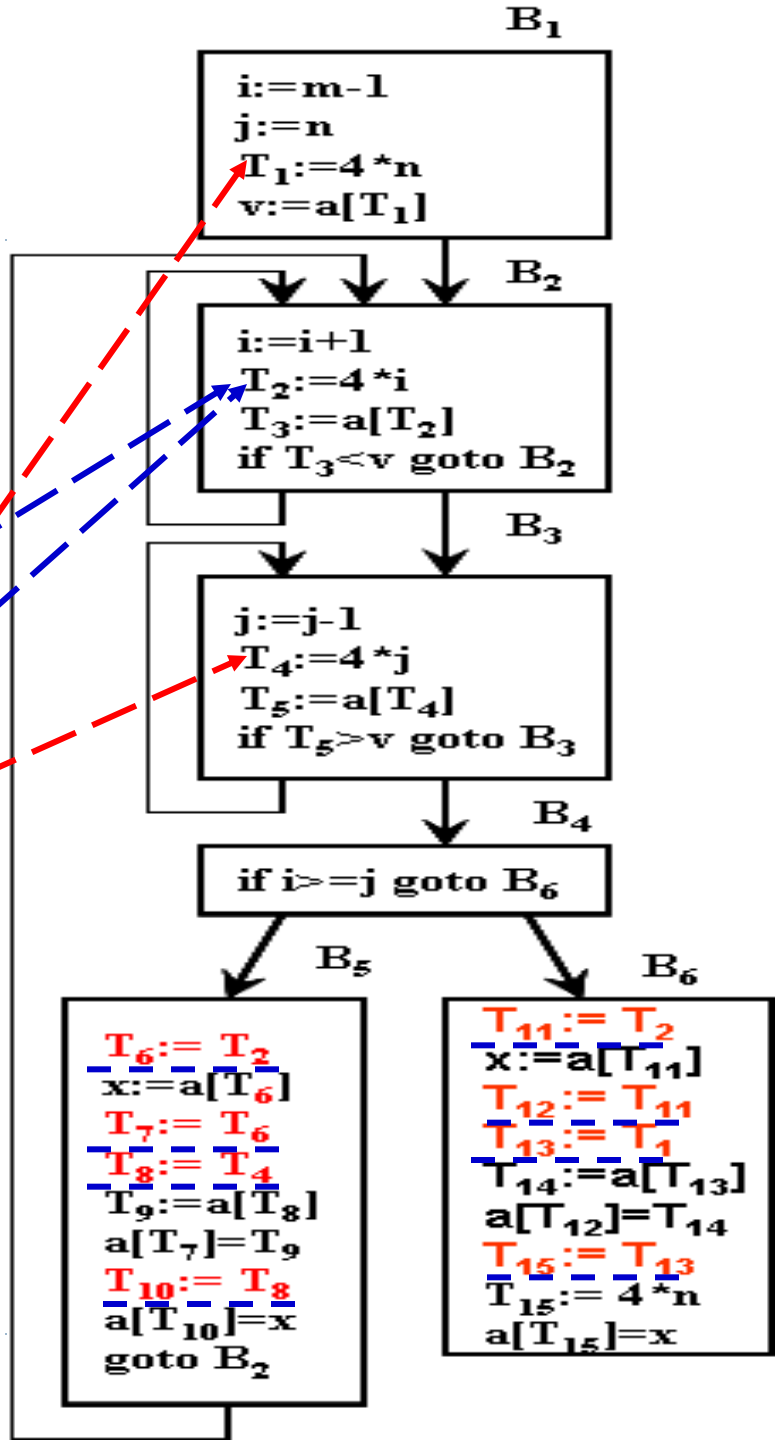


分析一：删除公共子表达式

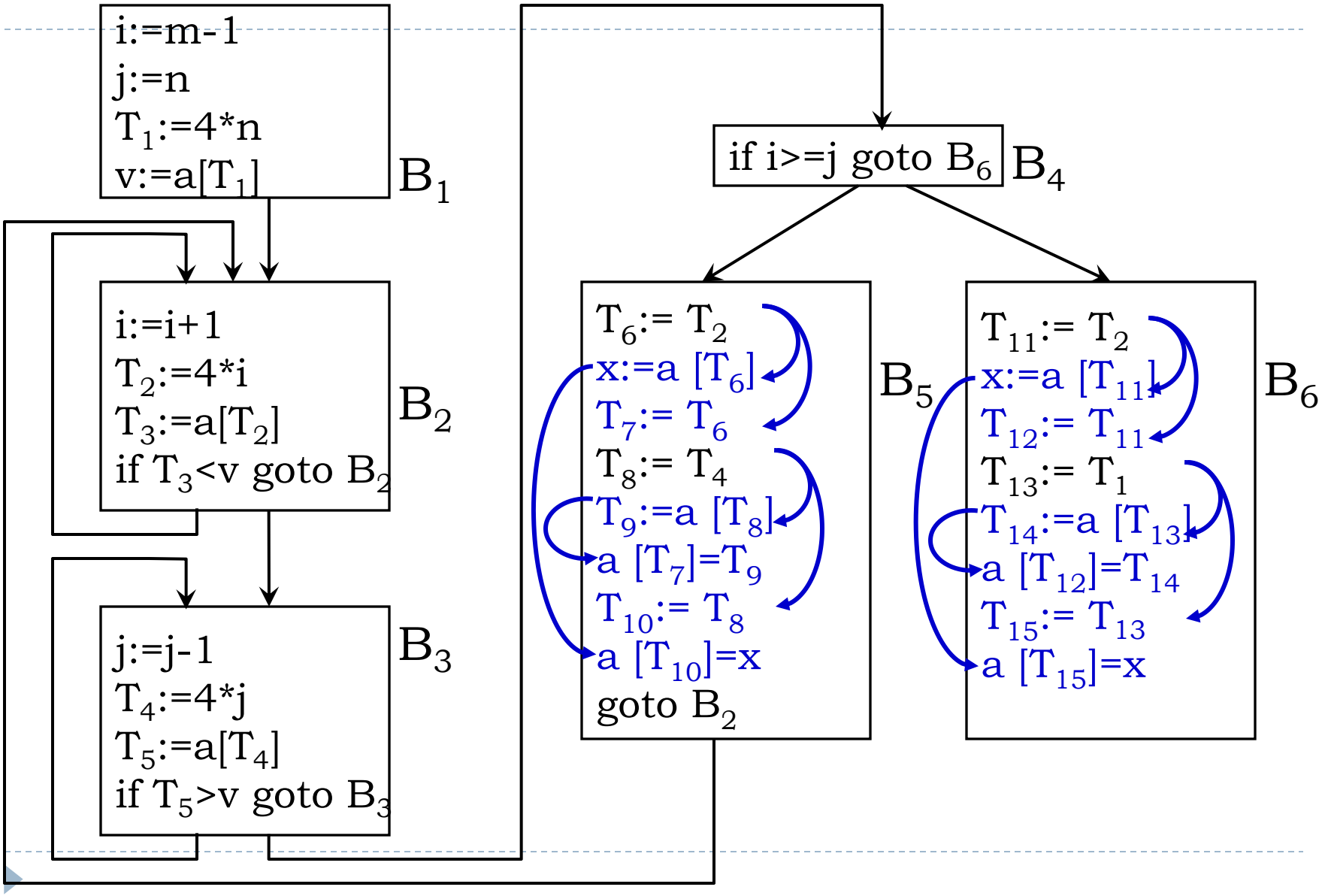


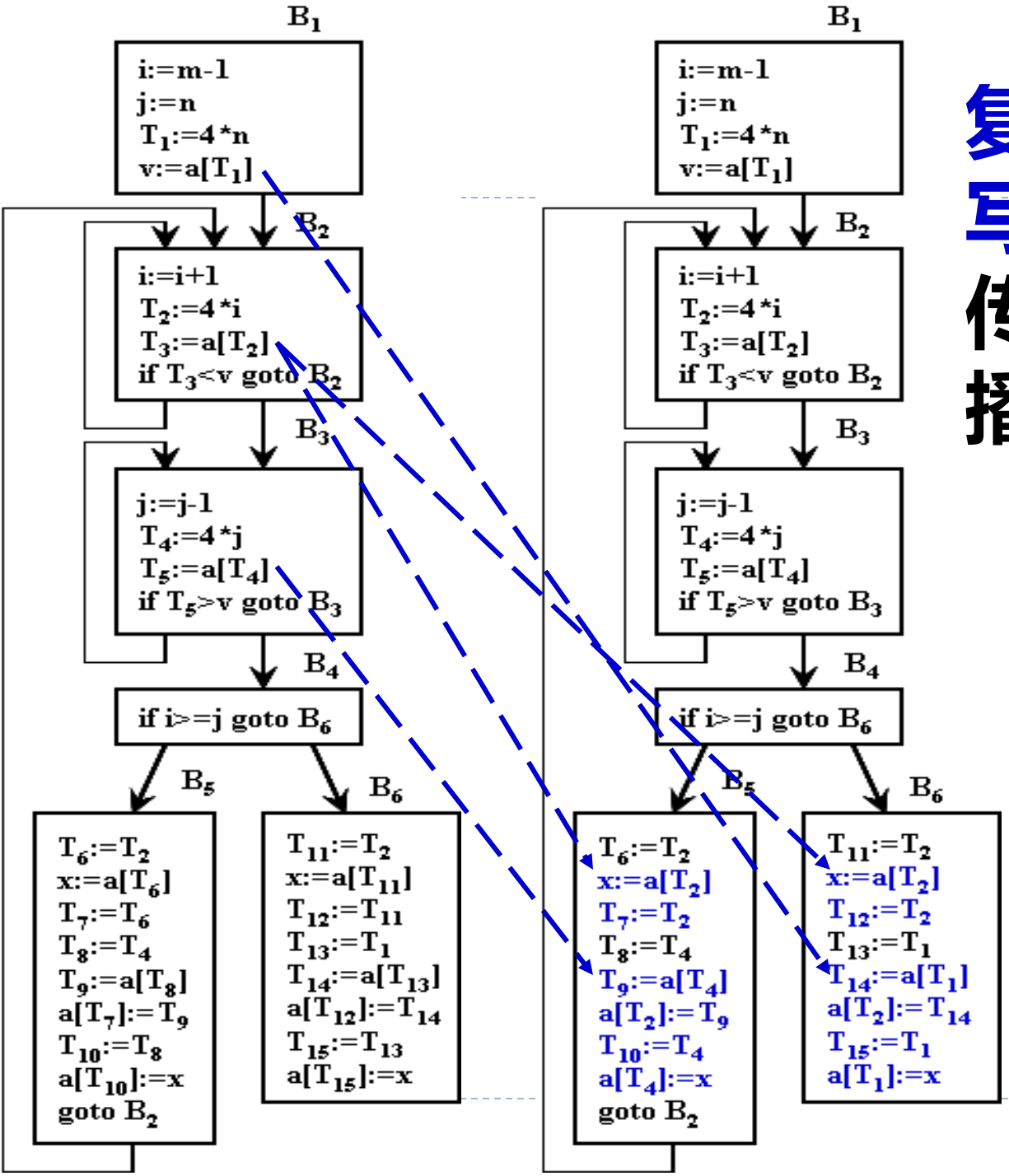
删除公共子表达式


B1
i:=m-1
j:=n
T1:=4*n
v:=a[T1]
B2
i:=i+1
T2:=4*i
T3:=a[T2]
if T3<v goto B2
B3
j:=j-1
T4:=4*j
T5:=a[T4]
if T5>v goto B3
B4
if i>=j goto B6
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
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

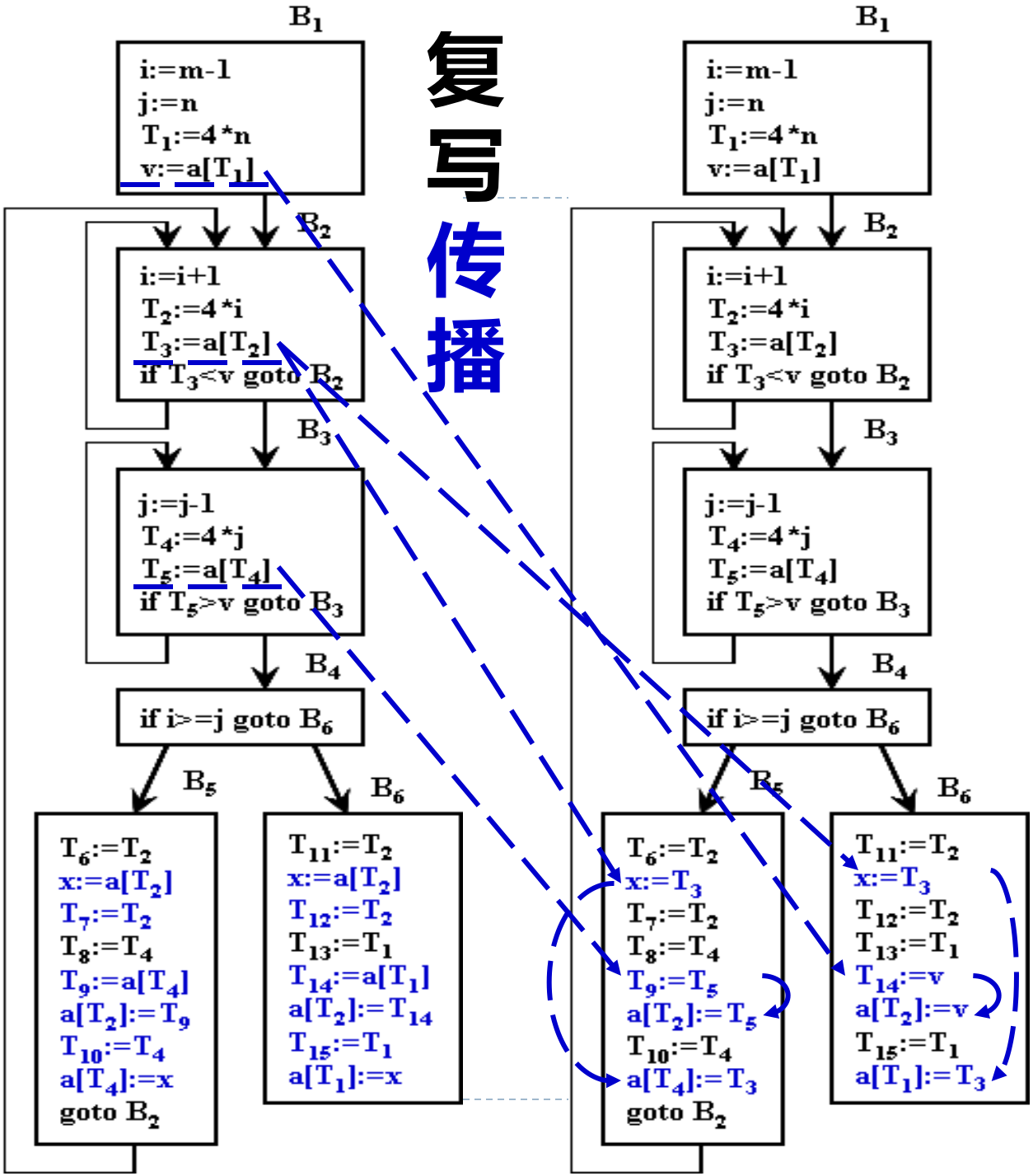

B1
i:=m-1
j:=n
T1:=4*n
v:=a[T1]
B2
i:=i+1
T2:=4*i
T3:=a[T2]
if T3<v goto B2
B3
j:=j-1
T4:=4*j
T5:=a[T4]
if T5>v goto B3
B4
if i>=j goto B6
B5
T6:=T2
x:=a[T6]
T7:=T6
T8:=T4
T9:=a[T8]
a[T7]=T9
T10:=T8
a[T10]=x
goto B2
B6
T11:=T2
x:=a[T11]
T12:=T11
T13:=T1
T14:=a[T13]
a[T12]=T14
T15:=T13
T15:=4*n
a[T15]=x

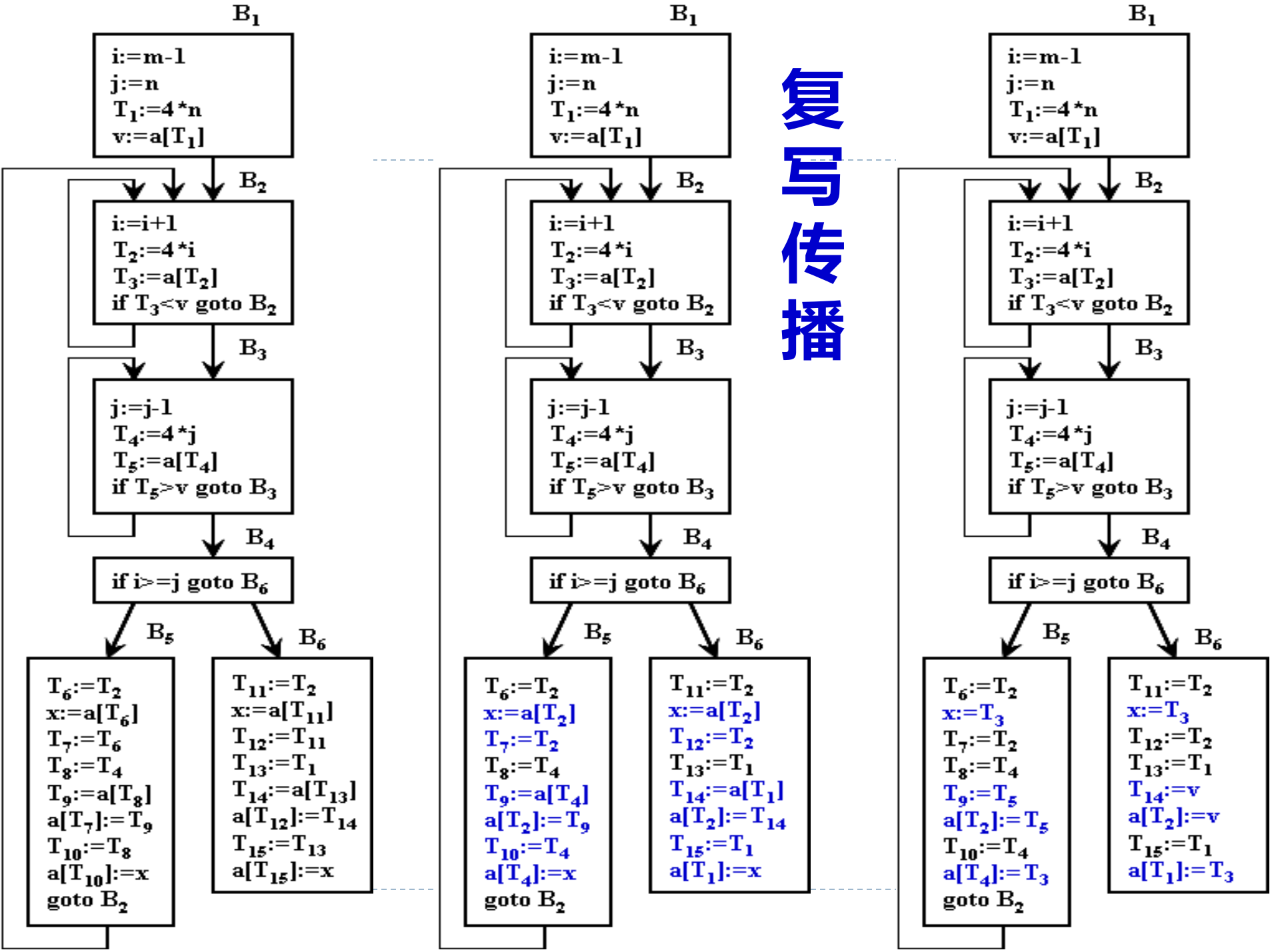
分析二：复写传播


i:=m-1
j:=n
T1:=4*n
v:=a[T1]
B1
i:=i+1
T2:=4*i
T3:=a[T2]
if T3<v goto B2
B2
j:=j-1
T4:=4*j
T5:=a[T4]
if T5>v goto B3
B3
if i>=j goto B6
B4
T6:= T2
x:=a [T6]
T7:= T6
T8:= T4
T9:=a [T8]
a [T7]=T9
T10:= T8
a [T10]=x
goto B2
B5
T11:= T2
x:=a [T11]
T12:= T11
T13:= T1
T14:=a [T13]
a [T12]=T14
T15:= T13
a [T15]=x
B6

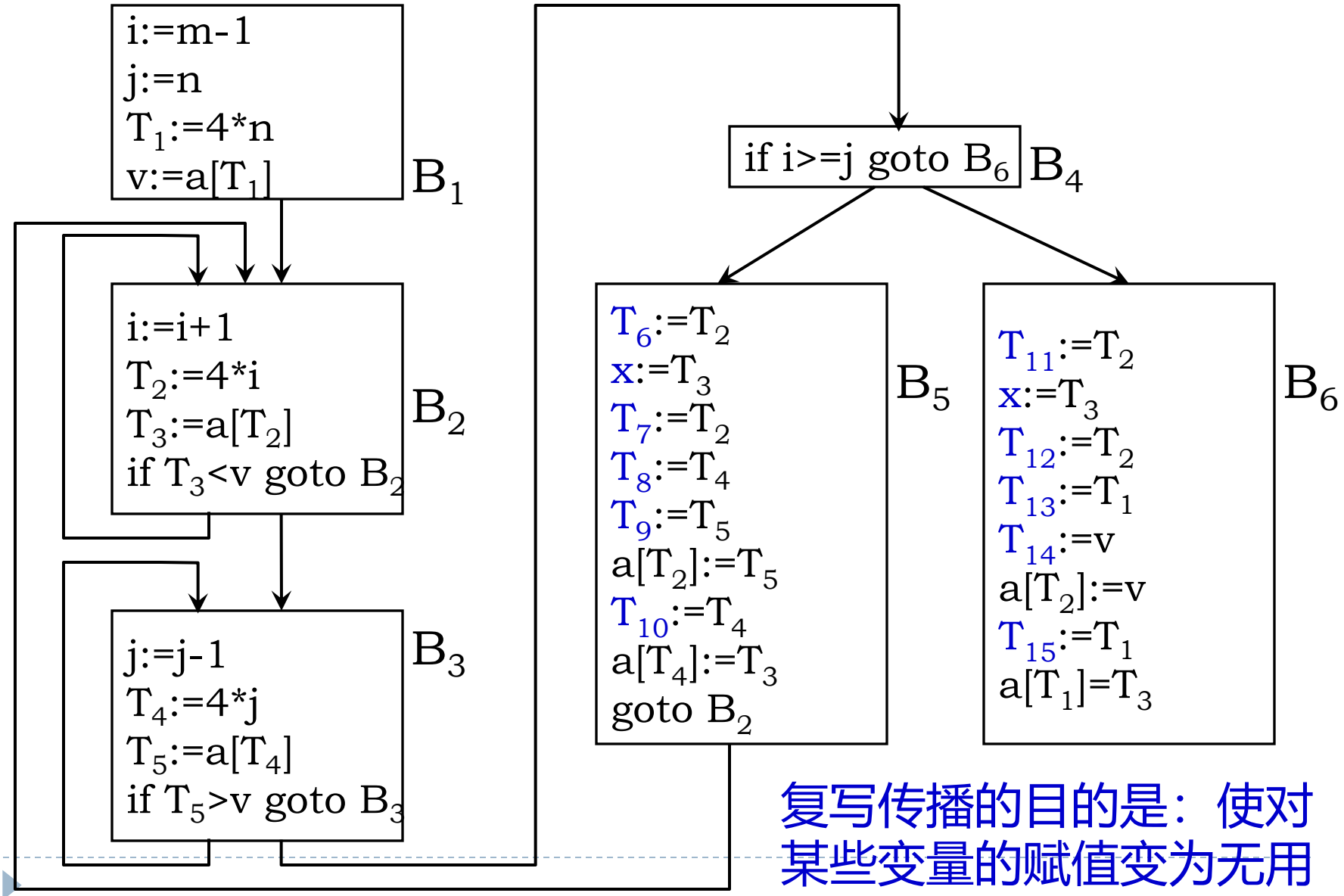

B1
i:=m-1
j:=n
T1:=4*n
v:=a[T1]
B2
i:=i+1
T2:=4*i
T3:=a[T2]
if T3<v goto B2
B3
j:=j-1
T4:=4*j
T5:=a[T4]
if T5>v goto B3
B4
if i>=j goto B6
B5
T6:=T2
x:=a[T6]
T7:=T6
T8:=T4
T9:=a[T8]
a[T7]:=T9
T10:=T8
a[T10]:=x
goto B2
B6
T11:=T2
x:=a[T11]
T12:=T11
T13:=T1
T14:=a[T13]
a[T12]:=T14
T15:=T13
a[T15]:=x
B1
i:=m-1
j:=n
T1:=4*n
v:=a[T1]
B2
i:=i+1
T2:=4*i
T3:=a[T2]
if T3<v goto B2
B3
j:=j-1
T4:=4*j
T5:=a[T4]
if T5>v goto B3
B4
if i>=j goto B6
B5
T6:=T2
x:=a[T2]
T7:=T2
T8:=T4
T9:=a[T4]
a[T2]:=T9
T10:=T4
a[T4]:=x
goto B2
B6
T11:=T2
x:=a[T2]
T12:=T2
T13:=T1
T14:=a[T1]
a[T2]:=T14
T15:=T1
a[T1]:=x

复写传播

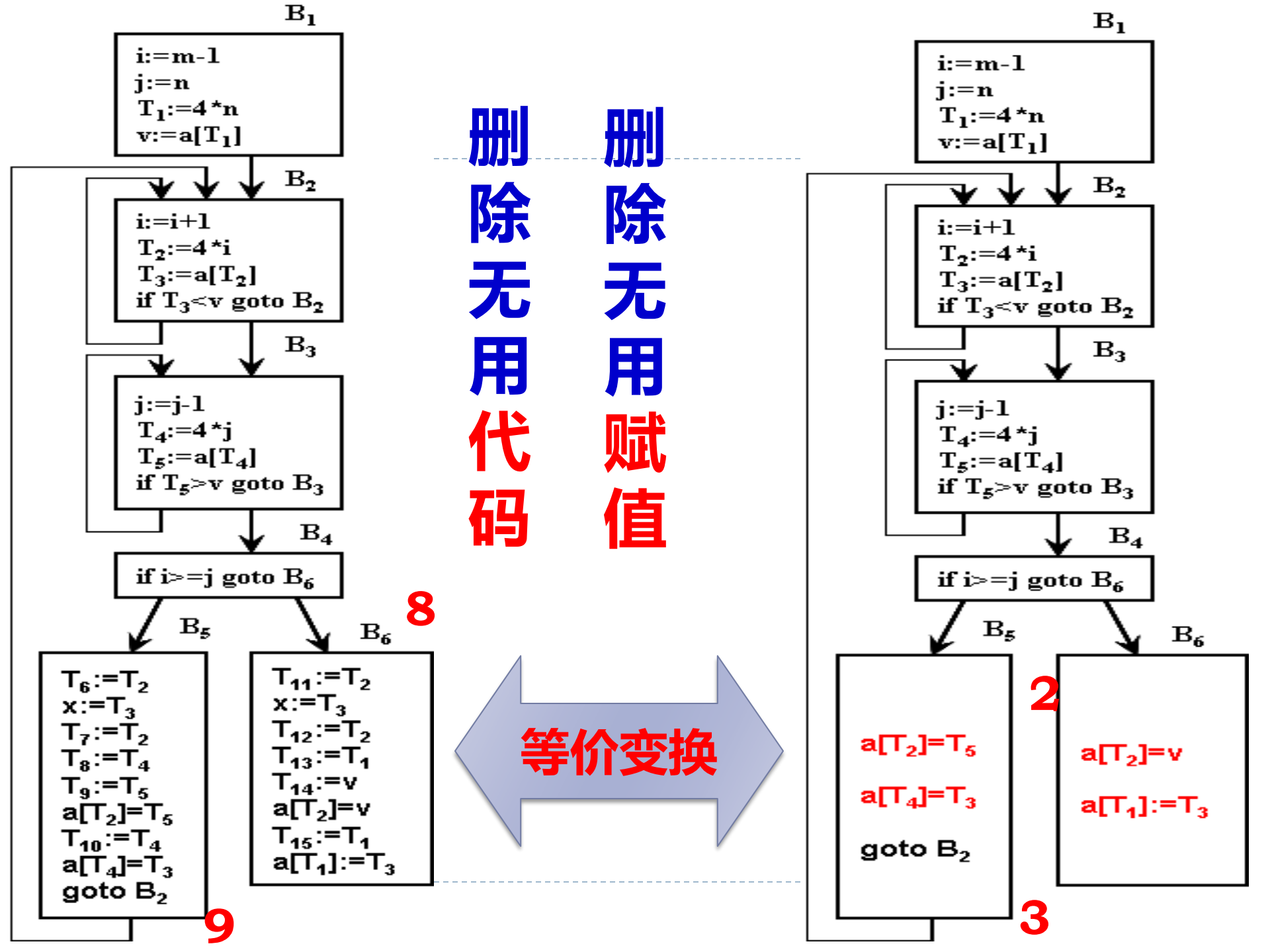

复写传播
B1
i:=m-1
j:=n
T1:=4*n
v:=a[T1]
B2
i:=i+1
T2:=4*i
T3:=a[T2]
if T3<v goto B2
B3
j:=j-1
T4:=4*j
T5:=a[T4]
if T5>v goto B3
B4
if i>=j goto B6
B5
T6:=T2
x:=a[T2]
T7:=T2
T8:=T4
T9:=a[T4]
a[T2]:=T9
T10:=T4
a[T4]:=x
goto B2
B6
T11:=T2
x:=a[T2]
T12:=T2
T13:=T1
T14:=a[T1]
a[T2]:=T14
T15:=T1
a[T1]:=x
B1
i:=m-1
j:=n
T1:=4*n
v:=a[T1]
B2
i:=i+1
T2:=4*i
T3:=a[T2]
if T3<v goto B2
B3
j:=j-1
T4:=4*j
T5:=a[T4]
if T5>v goto B3
B4
if i>=j goto B6
B5
T6:=T2
x:=T3
T7:=T2
T8:=T4
T9:=T5
a[T2]:=T5
T10:=T4
a[T4]:=T3
goto B2
B6
T11:=T2
x:=T3
T12:=T2
T13:=T1
T14:=v
a[T2]:=v
T15:=T1
a[T1]:=T3


B1
i:=m-1
j:=n
T1:=4*n
v:=a[T1]
B2
i:=i+1
T2:=4*i
T3:=a[T2]
if T3<v goto B2
B3
j:=j-1
T4:=4*j
T5:=a[T4]
if T5>v goto B3
B4
if i>=j goto B6
B5
T6:=T2
x:=a[T6]
T7:=T6
T8:=T4
T9:=a[T8]
a[T7]:=T9
T10:=T8
a[T10]:=x
goto B2
B6
T11:=T2
x:=a[T11]
T12:=T11
T13:=T1
T14:=a[T13]
a[T12]:=T14
T15:=T13
a[T15]:=x
复写传播
B1
i:=m-1
j:=n
T1:=4*n
v:=a[T1]
B2
i:=i+1
T2:=4*i
T3:=a[T2]
if T3<v goto B2
B3
j:=j-1
T4:=4*j
T5:=a[T4]
if T5>v goto B3
B4
if i>=j goto B6
B5
T6:=T2
x:=a[T2]
T7:=T2
T8:=T4
T9:=a[T4]
a[T2]:=T9
T10:=T4
a[T4]:=x
goto B2
B6
T11:=T2
x:=a[T2]
T12:=T2
T13:=T1
T14:=a[T1]
a[T2]:=T14
T15:=T1
a[T1]:=x
B1
i:=m-1
j:=n
T1:=4*n
v:=a[T1]
B2
i:=i+1
T2:=4*i
T3:=a[T2]
if T3<v goto B2
B3
j:=j-1
T4:=4*j
T5:=a[T4]
if T5>v goto B3
B4
if i>=j goto B6
B5
T6:=T2
x:=T3
T7:=T2
T8:=T4
T9:=T5
a[T2]:=T5
T10:=T4
a[T4]:=T3
goto B2
B6
T11:=T2
x:=T3
T12:=T2
T13:=T1
T14:=v
a[T2]:=v
T15:=T1
a[T1]:=T3

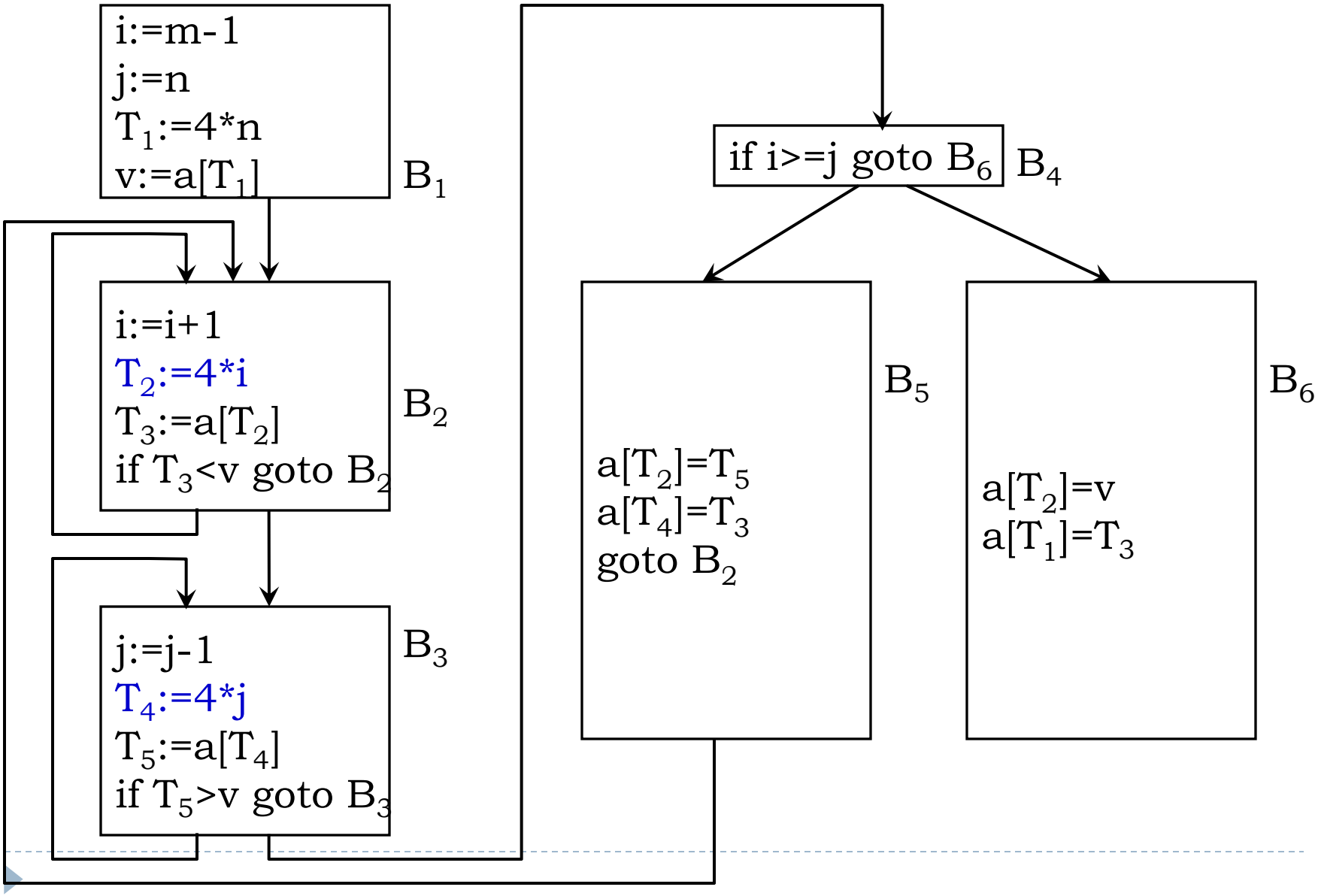
分析三：删除无用代码

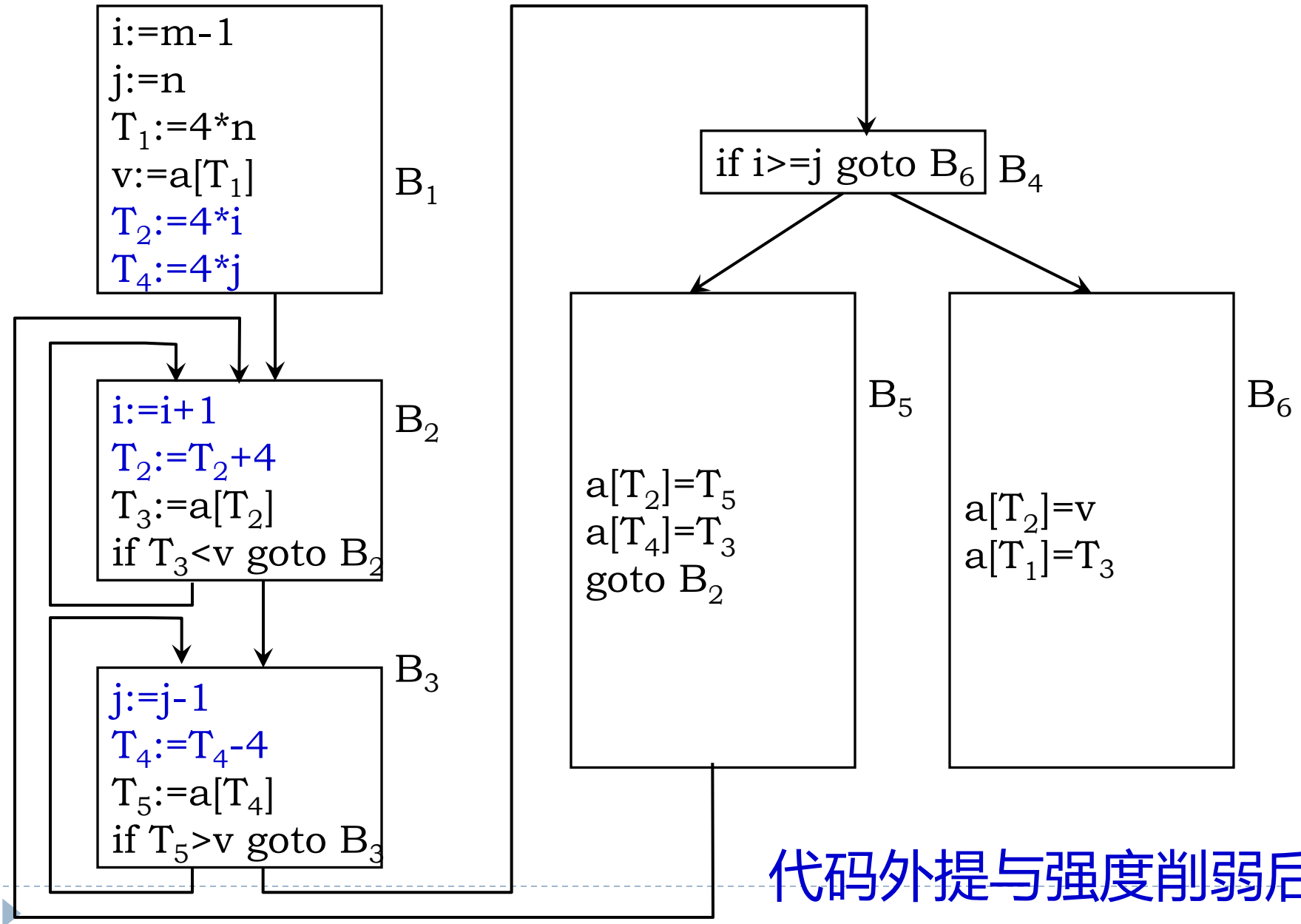


复写传播的目的是：使对某些变量的赋值变为无用


B1
i:=m-1
j:=n
T1:=4*n
v:=a[T1]
B2
i:=i+1
T2:=4*i
T3:=a[T2]
if T3<v goto B2
B3
j:=j-1
T4:=4*j
T5:=a[T4]
if T5>v goto B3
B4
if i>=j goto B6
B5
T6:=T2
x:=T3
T7:=T2
T8:=T4
T9:=T5
a[T2]=T5
T10:=T4
a[T4]=T3
goto B2
9
B6
8
T11:=T2
x:=T3
T12:=T2
T13:=T1
T14:=v
a[T2]=v
T15:=T1
a[T1]:=T3
删除无用代码
删除无用赋值
等价变换
B1
i:=m-1
j:=n
T1:=4*n
v:=a[T1]
B2
i:=i+1
T2:=4*i
T3:=a[T2]
if T3<v goto B2
B3
j:=j-1
T4:=4*j
T5:=a[T4]
if T5>v goto B3
B4
if i>=j goto B6
B5
a[T2]=T5
a[T4]=T3
goto B2
3
B6
2
a[T2]=v
a[T1]:=T3

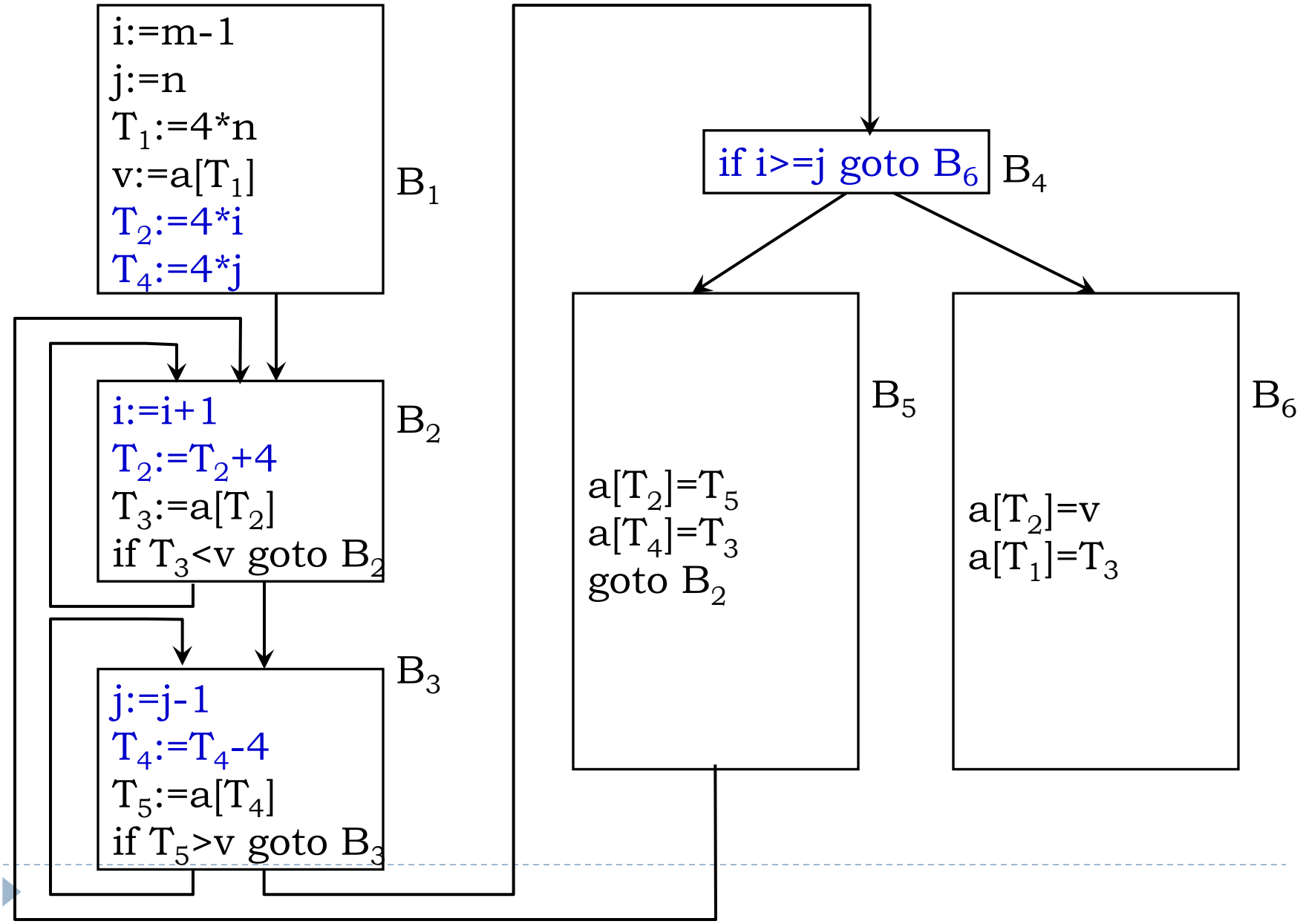
分析四：代码外提与强度削弱

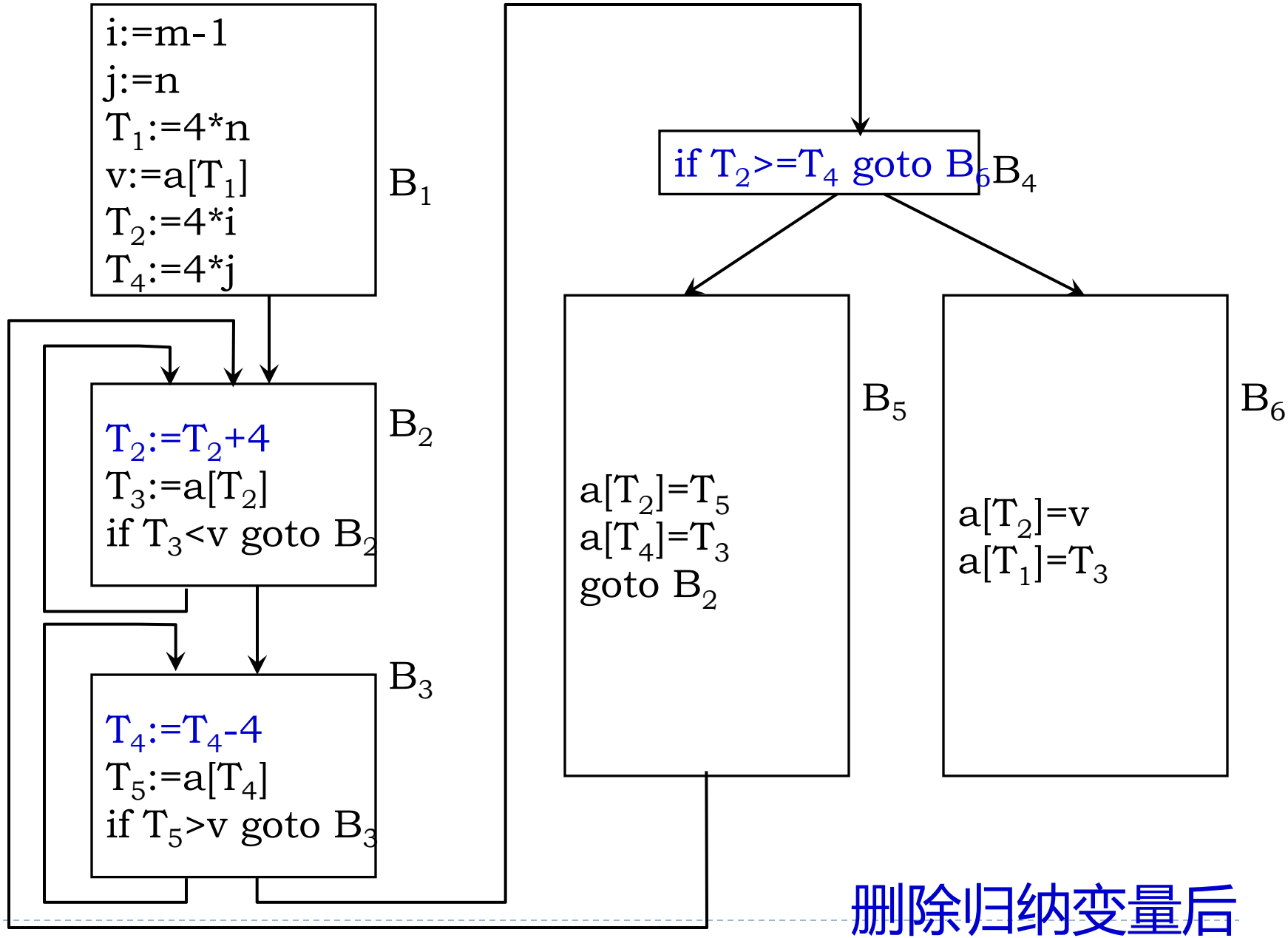



i:=m-1
j:=n
T1:=4*n
v:=a[T1]
T2:=4*i
T4:=4*j
B1
i:=i+1
T2:=T2+4
T3:=a[T2]
if T3<v goto B2
B2
j:=j-1
T4:=T4-4
T5:=a[T4]
if T5>v goto B3
B3
if i>=j goto B6
B4
a[T2]=T5
a[T4]=T3
goto B2
B5
a[T2]=v
a[T1]=T3
B6

代码外提与强度削弱后

分析五：删除归纳变量


i:=m-1
j:=n
T1:=4*n
v:=a[T1]
T2:=4*i
T4:=4*j
B1
i:=i+1
T2:=T2+4
T3:=a[T2]
if T3<v goto B2
B2
j:=j-1
T4:=T4-4
T5:=a[T4]
if T5>v goto B3
B3
if i>=j goto B6
B4
a[T2]=T5
a[T4]=T3
goto B2
B5
a[T2]=v
a[T1]=T3
B6


i:=m-1
j:=n
T1:=4*n
v:=a[T1]
T2:=4*i
T4:=4*j
B1
T2:=T2+4
T3:=a[T2]
if T3<v goto B2
B2
T4:=T4-4
T5:=a[T4]
if T5>v goto B3
B3
if T2>=T4 goto B6
B4
a[T2]=T5
a[T4]=T3
goto B2
B5
a[T2]=v
a[T1]=T3
B6

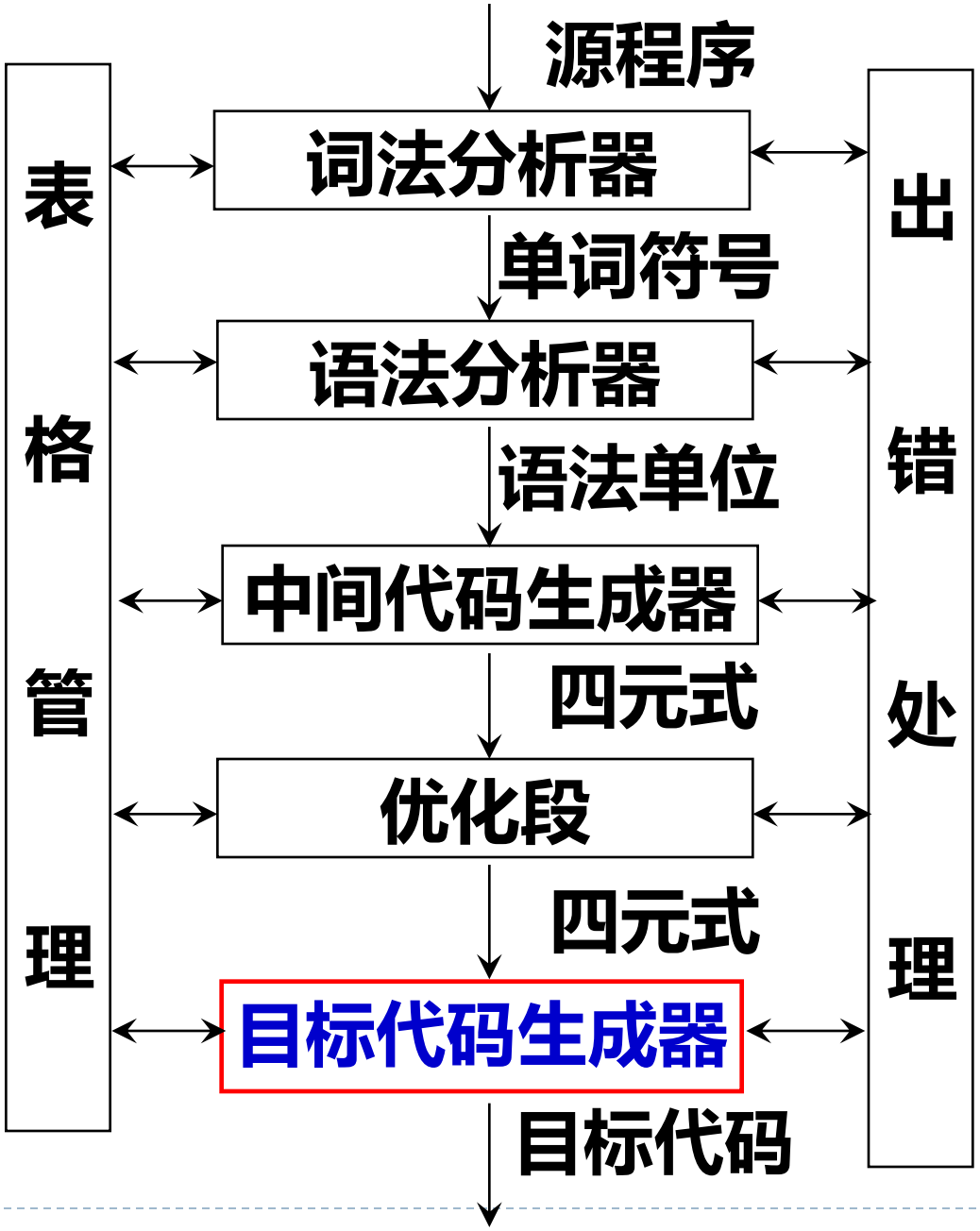
删除归纳变量后

Exercise

- 编译过程中可进行的优化如何分类？最常用的代码优化技术有哪些?
- 解:
  - 依据优化所涉及的程序范围，可以分为
    - 全局优化
    - 局部优化
    - 循环优化
  - 最常用的代码优化技术有
    - 删除公共子表达式
    - 复写传播
    - 删除无用代码
    - 代码外提
    - 强度削弱
    - 删除归纳变量等

Code generation

- 代码生成是把语法分析后或优化后的中间代码变换成目标代码。
- 目标代码一般有以下三种形式：
  - 能够立即执行的机器语言代码，所有地址已经定位；
  - 待装配的机器语言模块。执行时，由连接装配程序把它们和某些运行程序连接起来，转换成能执行的机器语言代码；
  - 汇编语言代码。尚须经过汇编程序汇编，转换成可执行的机器语言代码。


源程序
词法分析器
单词符号
语法分析器
语法单位
中间代码生成器
四元式
优化段
四元式
目标代码生成器
目标代码
表格管理
出错处理

Code generation

- 代码生成着重考虑的问题：
  - 每一个语法成分的语义
  - 目标代码中需要哪些信息，怎样截取这些信息

Code generation

- 设计代码生成器时要考虑的一般问题：
  - 代码生成器的输入
    - 代码生成器的输入包括源程序的中间表示，以及符号表中的信息
    - 类型检查
  - 目标程序
    - 绝对机器代码、可再定位机器语言、汇编语言
  - 指令选择
  - 寄存器分配
  - 计算顺序选择

Example

- 不考虑代码的执行效率，目标代码生成是不难的，例如：
  - A:=(B+C)*D+E
- 翻译为四元式：
  - T_1:=B+C
  - T_2:=T_1*D
  - T_3:=T_2+E
  - A:=T_3

假设只有一个寄存器可供使用

- 四元式

T_1:=B+C

T_2:=T_1*D

T_3:=T_2+E

A:=T_3

- 目标代码:

```
LD    R0, B
ADD   R0, C
ST    R0, T1
LD    R0, T1
MUL   R0, D
ST    R0, T2
LD    R0, T2
ADD   R0, E
ST    R0, T3
LD    R0, T3
ST    R0, A
```

假设T_1，T_2，T_3在基本块之后不再引用:

```
LD    R0, B
ADD   R0, C
MUL   R0, D
ADD   R0, E
ST    R0, A
```