北京大学计算机学院

2024 年秋季学期　　《编译原理》

第九章　代码优化

Code Optimization

本章内容

- 代码优化的常用方法
- 数据流分析简介
  - 到达定值
  - 活跃变量
  - 可用表达式
- 基于路径表达式的分析

代码优化概述

- 为了设计一个好的代码优化器，要考虑如下几个方面：
  - 代码优化应遵循的原则
  - 代码优化的阶段
  - 代码优化器的结构
  - 代码优化的范围

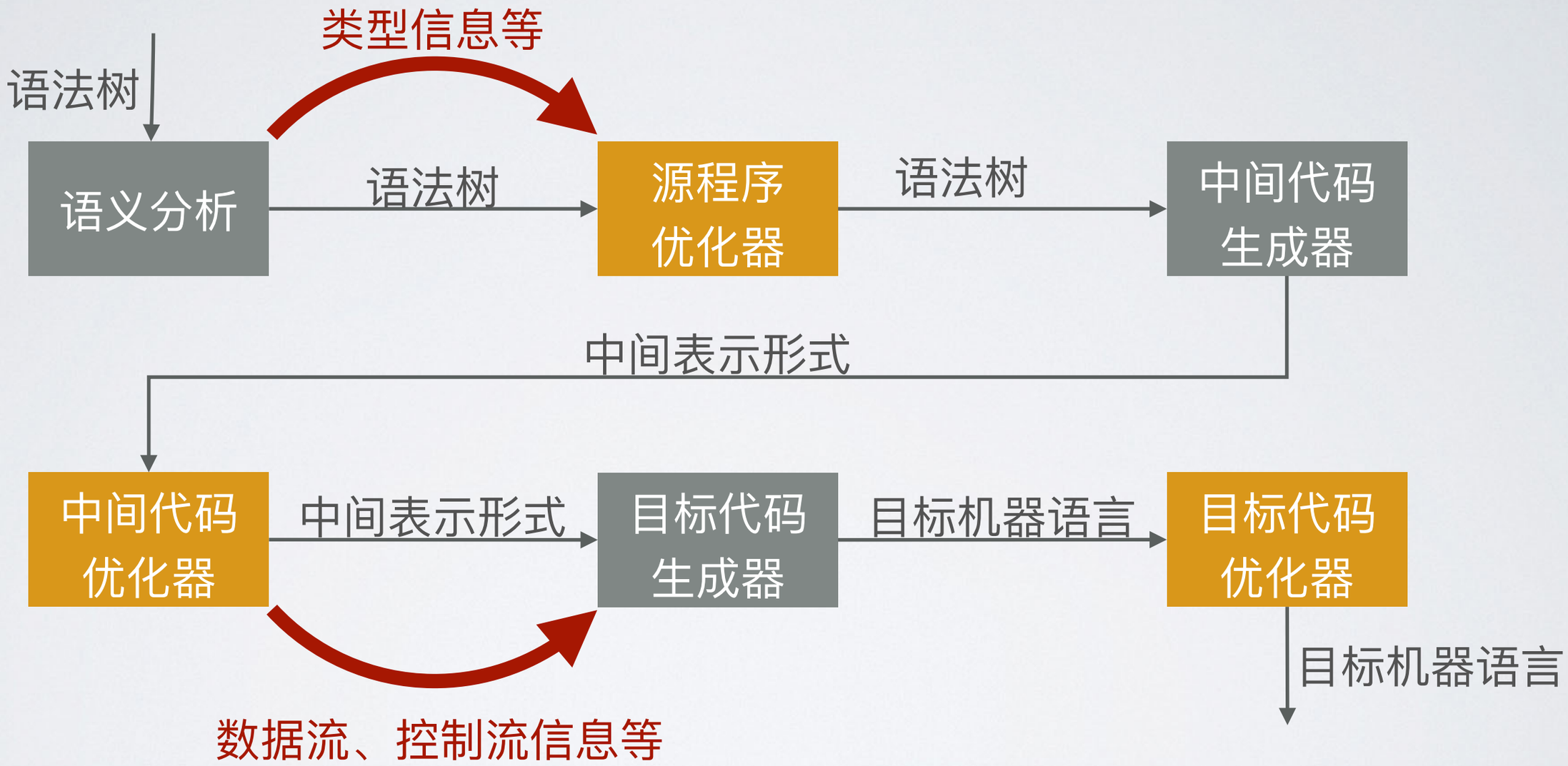
代码优化应遵循的原则

- 保证安全
  - 编译器的优化必须保持源程序的语义
  - 语义通常定义为程序的可观察行为
- 提高收益
  - 优化的目标通常是更快，也可能是更短、能耗更低等
  - “二八法则”：一个程序 80% 的运行时间花费在 20% 的代码上
  - 编译器的优化应当关注这最重要的 20% 的代码

代码优化的阶段

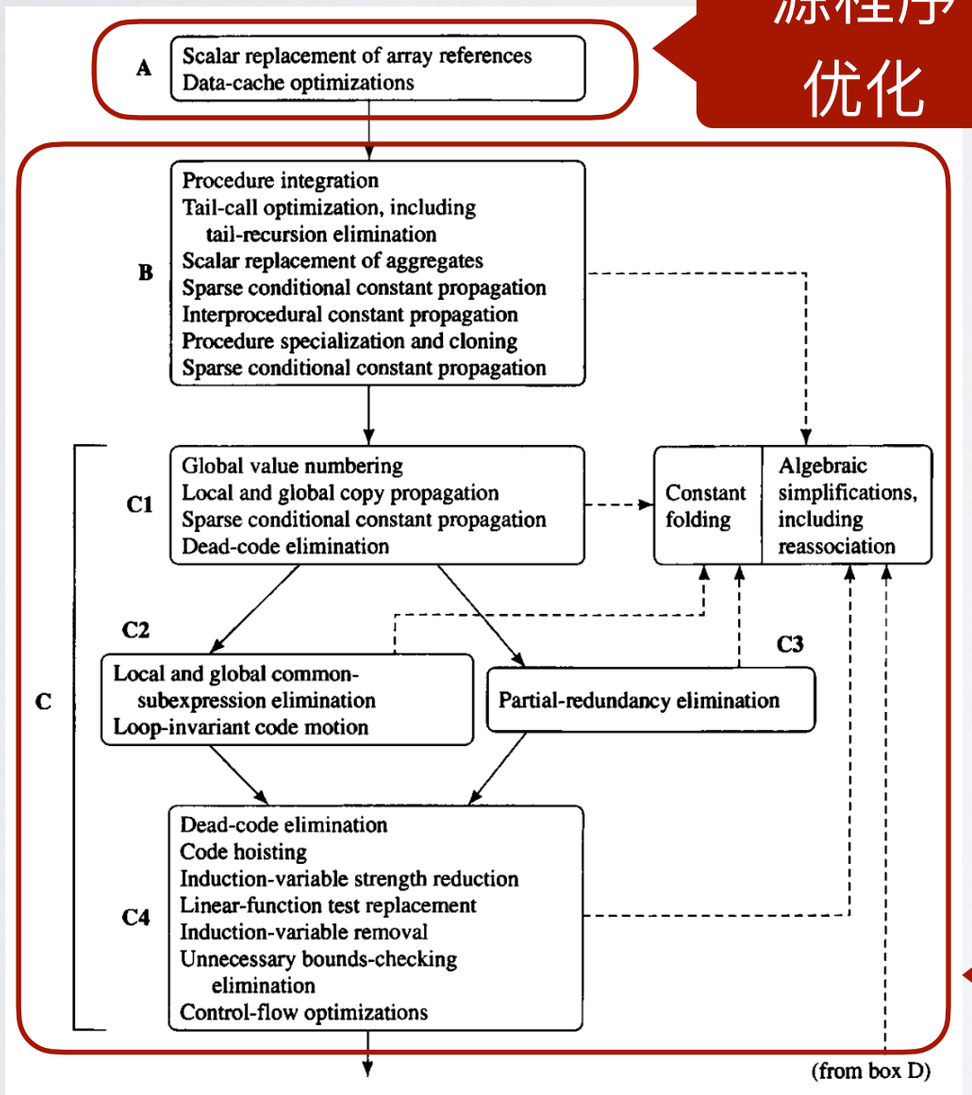
- 算法设计阶段的优化
  - 通常由程序员进行
  - 比如：把选择排序换成快速排序
- 编译阶段的优化
  - 语义分析阶段
    - 通过静态检查等的信息，在源程序上进行优化
  - 中间代码生成阶段
    - 机器无关优化：在中间代码上进行优化
  - 目标代码生成阶段
    - 机器有关优化：在目标代码上进行优化
  - （链接时刻优化，link-time optimization）

代码优化器的结构


语法树
类型信息等
语义分析
语法树
源程序
优化器
语法树
中间代码
生成器
中间表示形式
中间代码
优化器
中间表示形式
目标代码
生成器
目标机器语言
目标代码
优化器
目标机器语言
数据流、控制流信息等

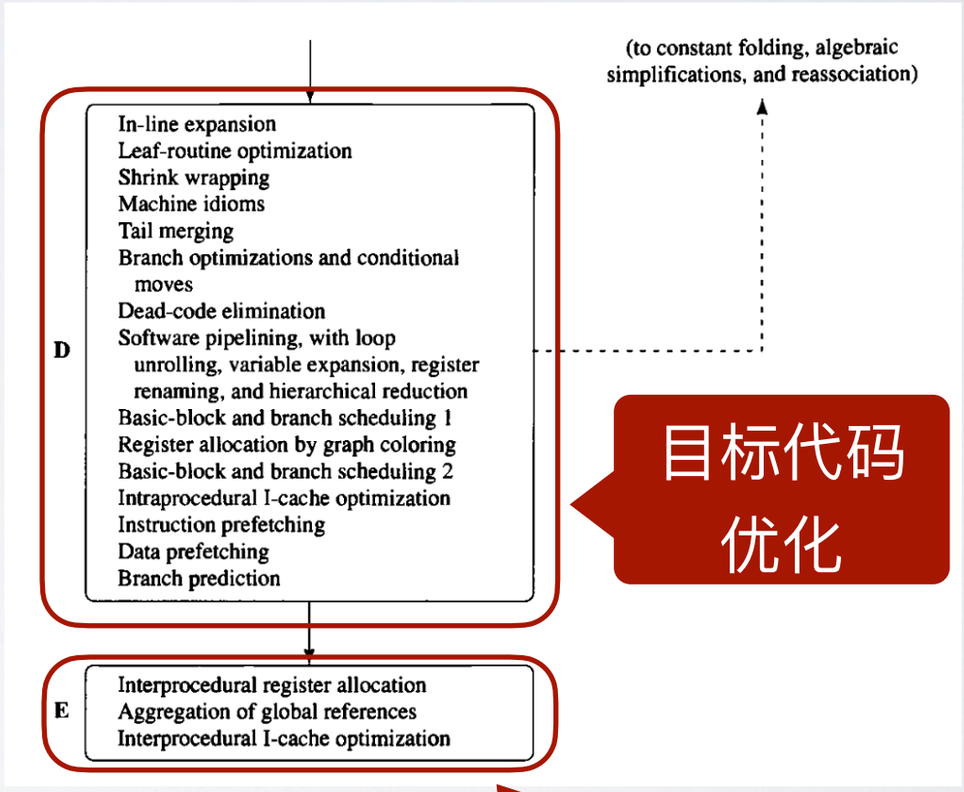
现代代码优化器的复杂性（1）

- Steven S. Muchnick, Advanced Compiler Design and Implementation



源程序优化

中间代码优化



目标代码优化

链接时刻优化

现代代码优化器的复杂性（2）

- LLVM 中的分析/优化趟（pass）
  - 多趟优化器：一趟只做一件事

Basic-Block Vectorization
Profile Guided Block Placement
Break critical edges in CFG
Merge Duplicate Global
Simple constant propagation
Dead Code Elimination
Dead Argument Elimination
Dead Type Elimination
Dead Instruction Elimination
Dead Store Elimination
Deduce function attributes
Dead Global Elimination
Global Variable Optimizer
Global Value Numbering
Canonicalize Induction Variables

Function Integration/Inlining
Combine redundant instructions
Internalize Global Symbols
Interprocedural constant propa.
Jump Threading
Loop-Closed SSA Form Pass
Loop Strength Reduction
Rotate Loops
Loop Invariant Code Motion
Canonicalize natural loops
Unroll loops
Unswitch loops
Promote Memory to Register
MemCpy Optimization
Merge Functions

Unify function exit nodes
Remove unused exception handling
Reassociate expressions
Demote all values to stack slots
Sparse Conditional Cons. Propaga.
Simplify the CFG
Code sinking
Strip all symbols from a module
Strip debug info for unused symbols
Strip Unused Function Prototypes
Strip all llvm.dbg.declare intrinsics
Tail Call Elimination
Delete dead loops
Extract loops into new functions
……

代码优化的范围

- 局部优化（local optimization）
  - 只对基本块内的语句进行分析，在基本块内进行优化
- 区域性优化（regional optimization）
  - 对若干个基本块构成的区域进行分析，比如对循环的优化
- 全局优化（global optimization）
  - 对一个过程所有基本块的信息和它们的关系进行分析，在此基础上在整个过程范围内进行优化
- 过程间优化（interprocedural optimization）
  - 对一个程序所有过程及其调用信息进行分析，对整个程序整体优化

本章内容

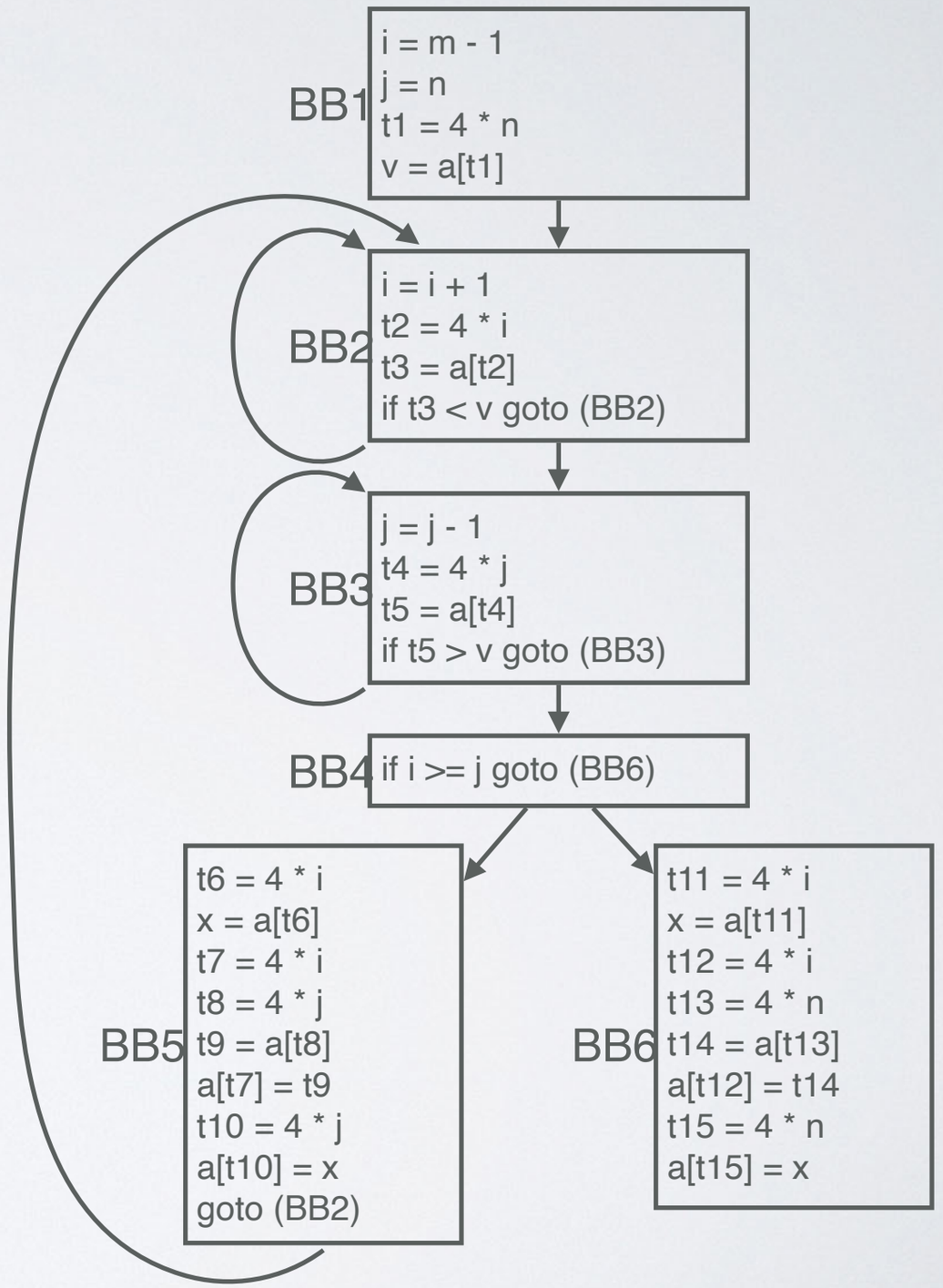
- 代码优化的常用方法
- 数据流分析简介
  - 到达定值
  - 活跃变量
  - 可用表达式
- 基于路径表达式的分析

代码优化的常用方法

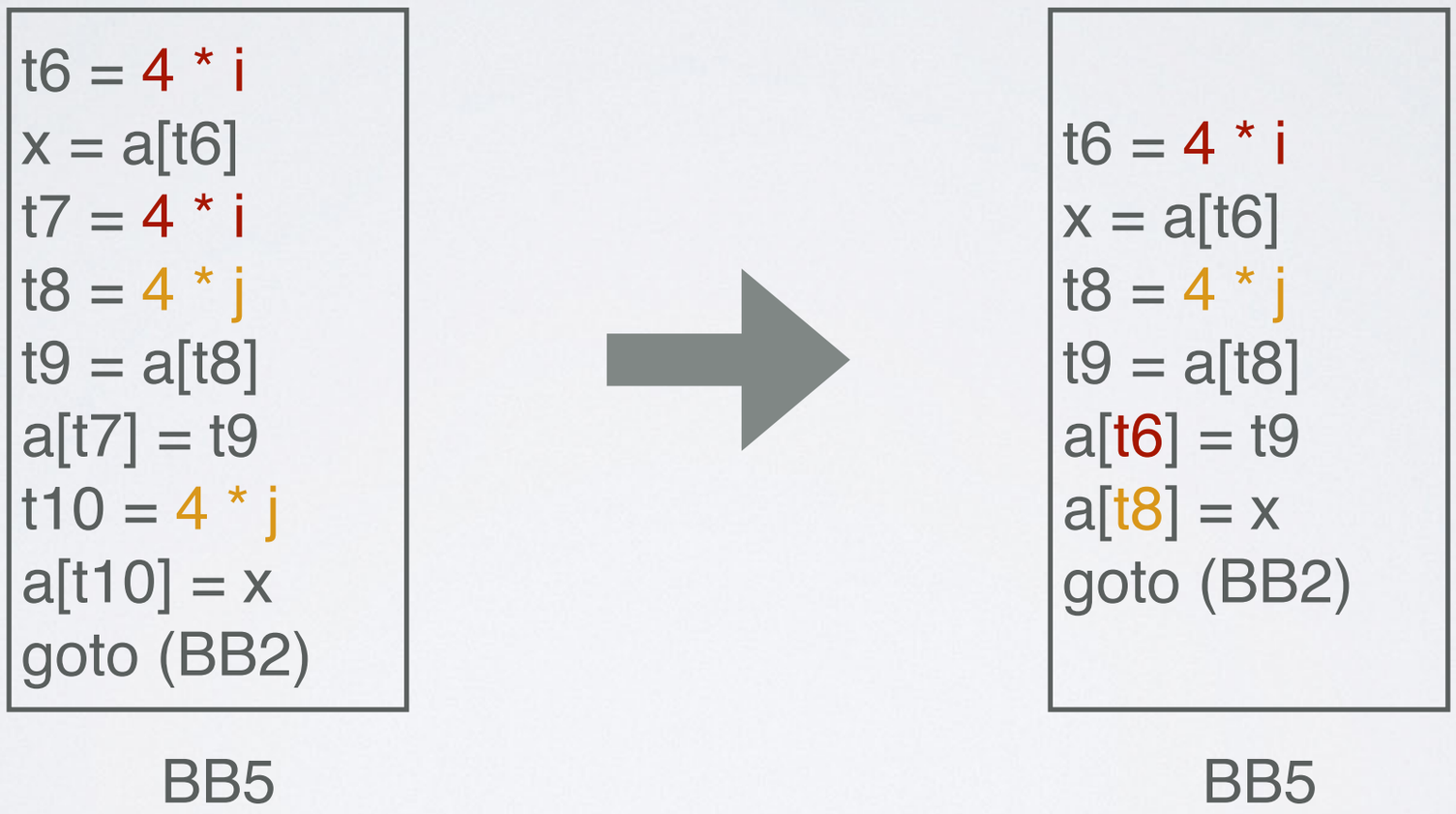
- 公共子表达式消除
- 复写传播
- 死代码消除
- 常量折叠
- 循环优化
  - 代码外提
  - 强度消减
  - 归纳变量消除

例子：快速排序

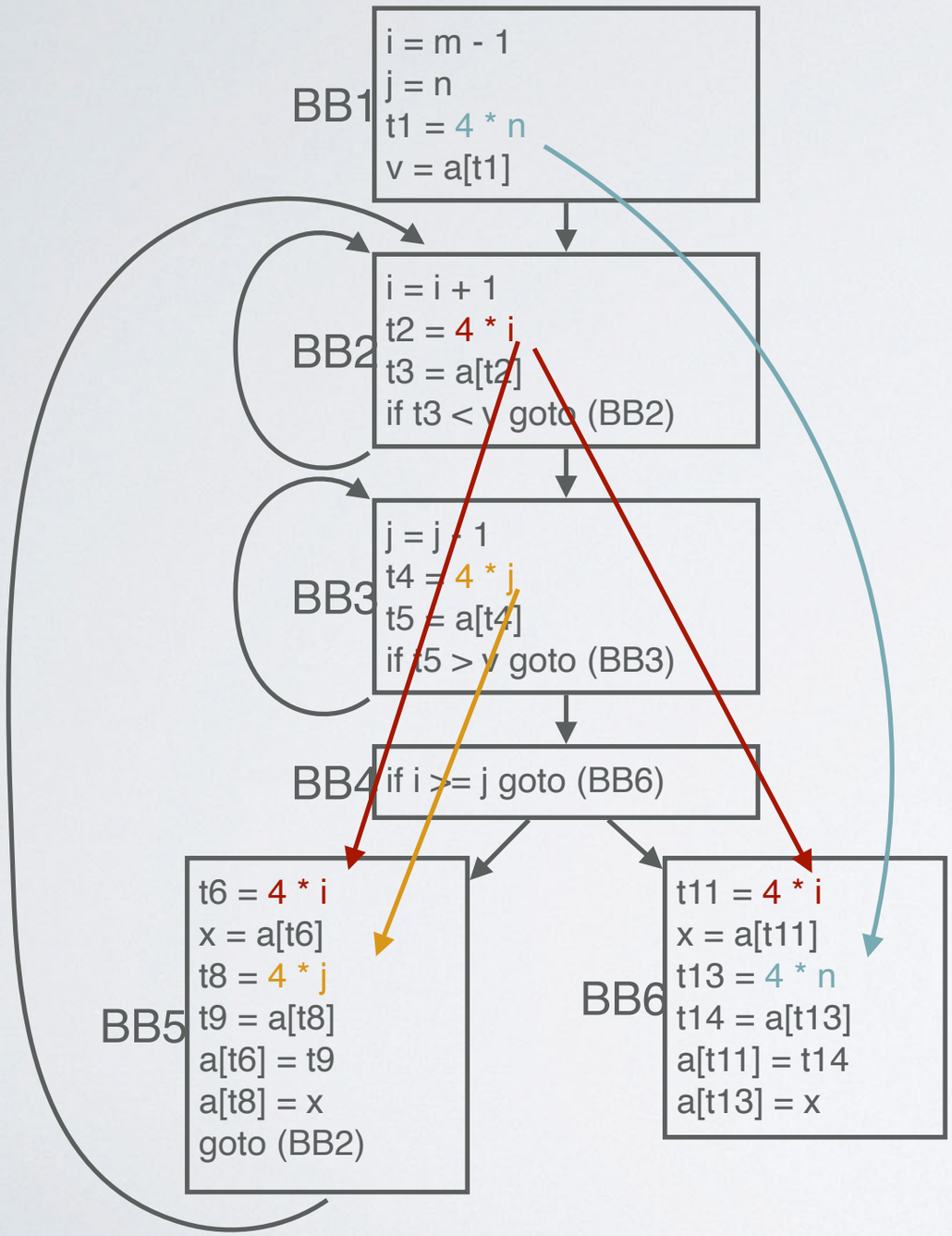
```
void quicksort(int m, int n)
{
  int i, j;
  int v, x;
  if (n <= m) return;
  /* 片段由此开始 */
  i = m - 1; j = n; v = a[n];
  while (1) {
    do i = i + 1; while (a[i] < v);
    do j = j - 1; while (a[j] > v);
    if (i >= j) break;
    x = a[i]; a[i] = a[j]; a[j] = x;
  }
  x = a[i]; a[i] = a[n]; a[n] = x;
  /* 片段在此结束 */
  quicksort(m, j); quicksort(i + 1, n);
}
```

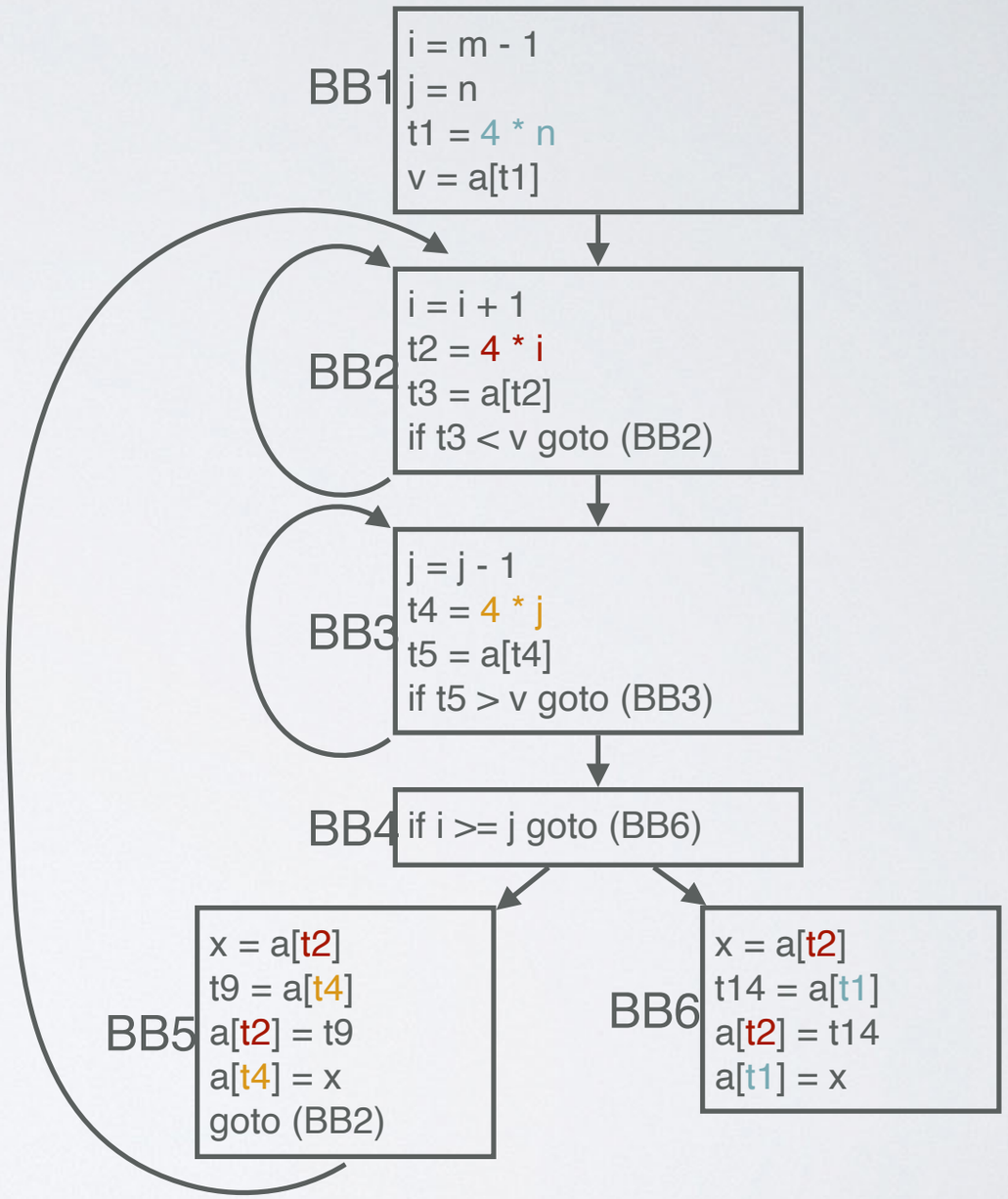


公共子表达式消除（1）


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 (BB2)
BB5
t6 = 4 * i
x = a[t6]
t8 = 4 * j
t9 = a[t8]
a[t6] = t9
a[t8] = x
goto (BB2)
BB5

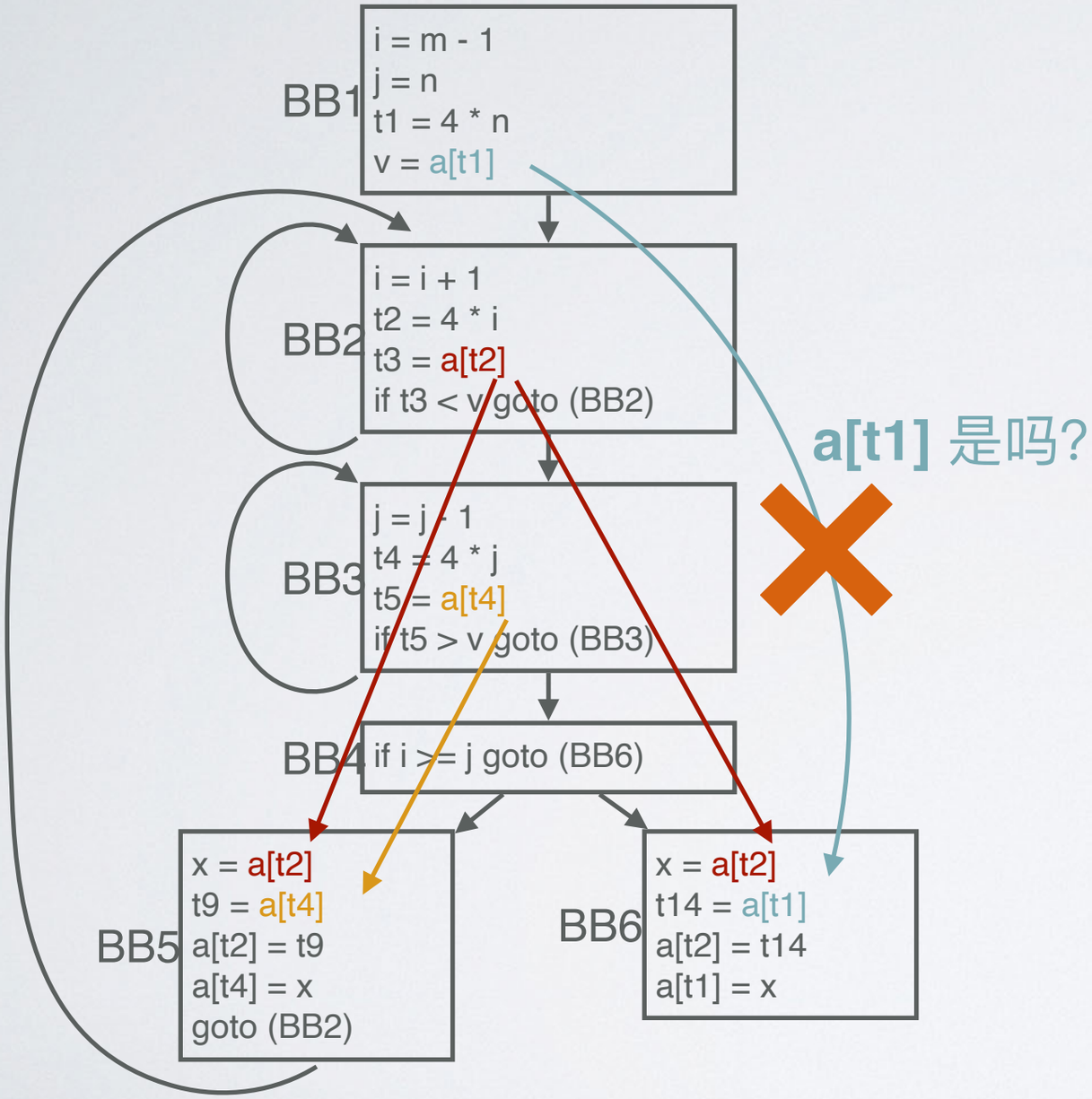
公共子表达式消除（2）

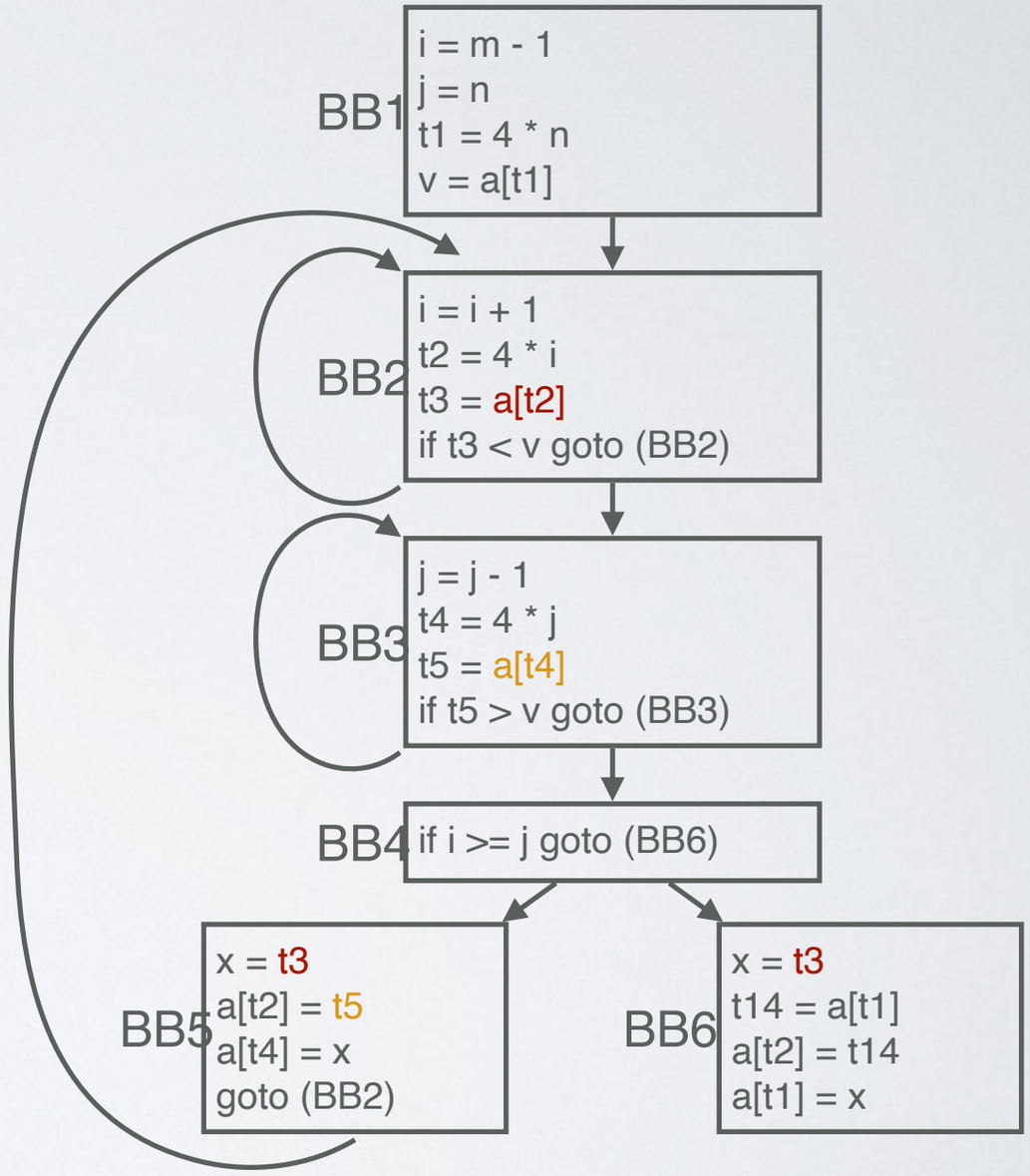




还有公共子表达式吗?

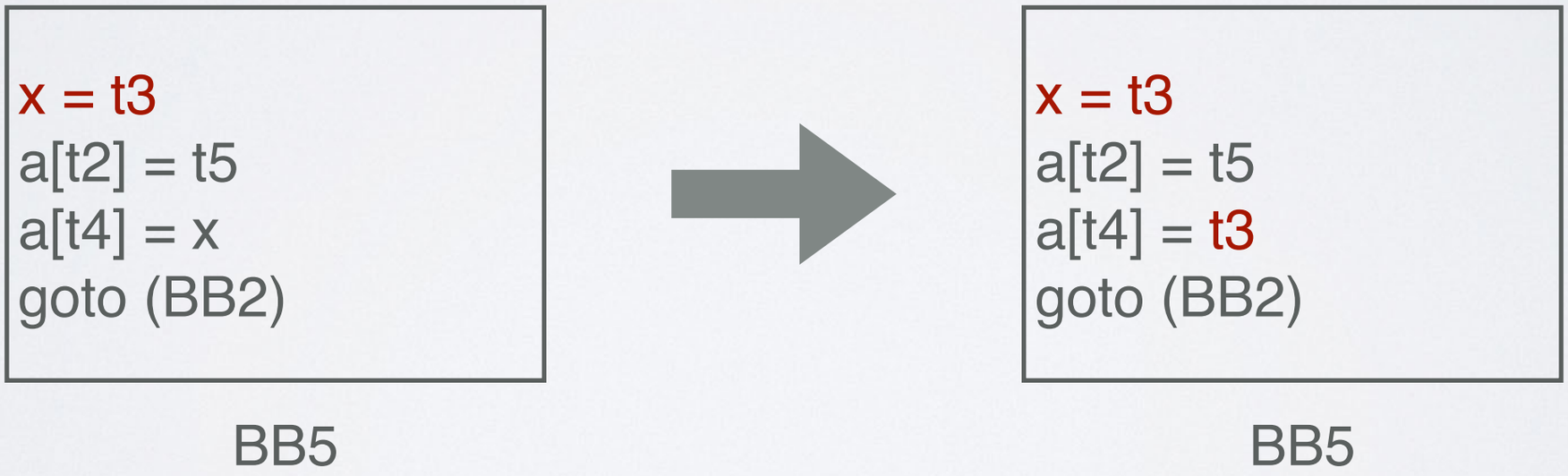
公共子表达式消除（3）


BB1
i = m - 1
j = n
t1 = 4 * n
v = a[t1]
BB2
i = i + 1
t2 = 4 * i
t3 = a[t2]
if t3 < v goto (BB2)
BB3
j = j - 1
t4 = 4 * j
t5 = a[t4]
if t5 > v goto (BB3)
BB4
if i >= j goto (BB6)
BB5
x = a[t2]
t9 = a[t4]
a[t2] = t9
a[t4] = x
goto (BB2)
BB6
x = a[t2]
t14 = a[t1]
a[t2] = t14
a[t1] = x
a[t1] 是吗?


BB1
i = m - 1
j = n
t1 = 4 * n
v = a[t1]
BB2
i = i + 1
t2 = 4 * i
t3 = a[t2]
if t3 < v goto (BB2)
BB3
j = j - 1
t4 = 4 * j
t5 = a[t4]
if t5 > v goto (BB3)
BB4
if i >= j goto (BB6)
BB5
x = t3
a[t2] = t5
a[t4] = x
goto (BB2)
BB6
x = t3
t14 = a[t1]
a[t2] = t14
a[t1] = x

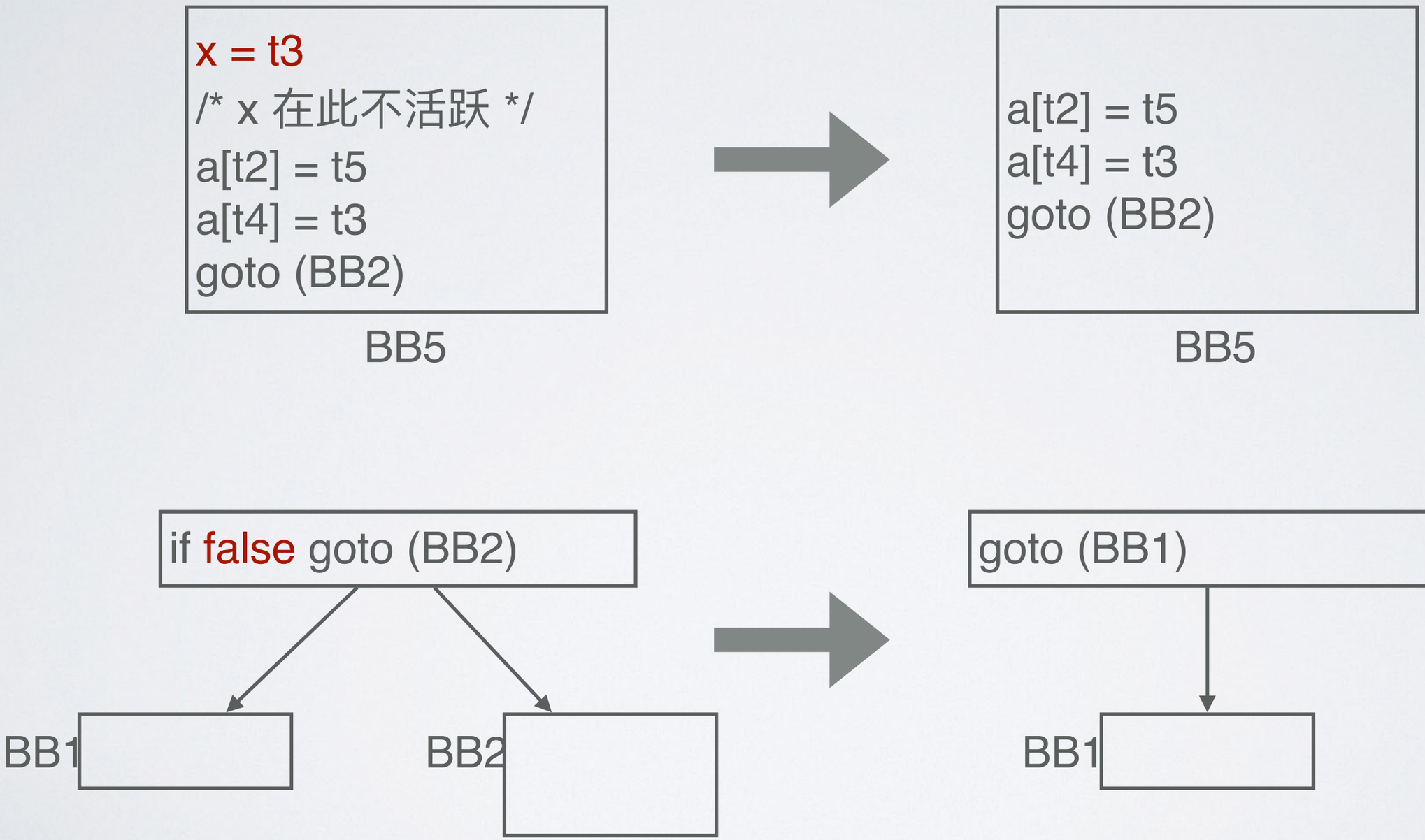
复写传播

- copy propagation
- 形如 $x = y$ 的赋值语句称为复写语句
- 复写传播把用到 x 的地方换成 y，从而最终删除 $x = y$



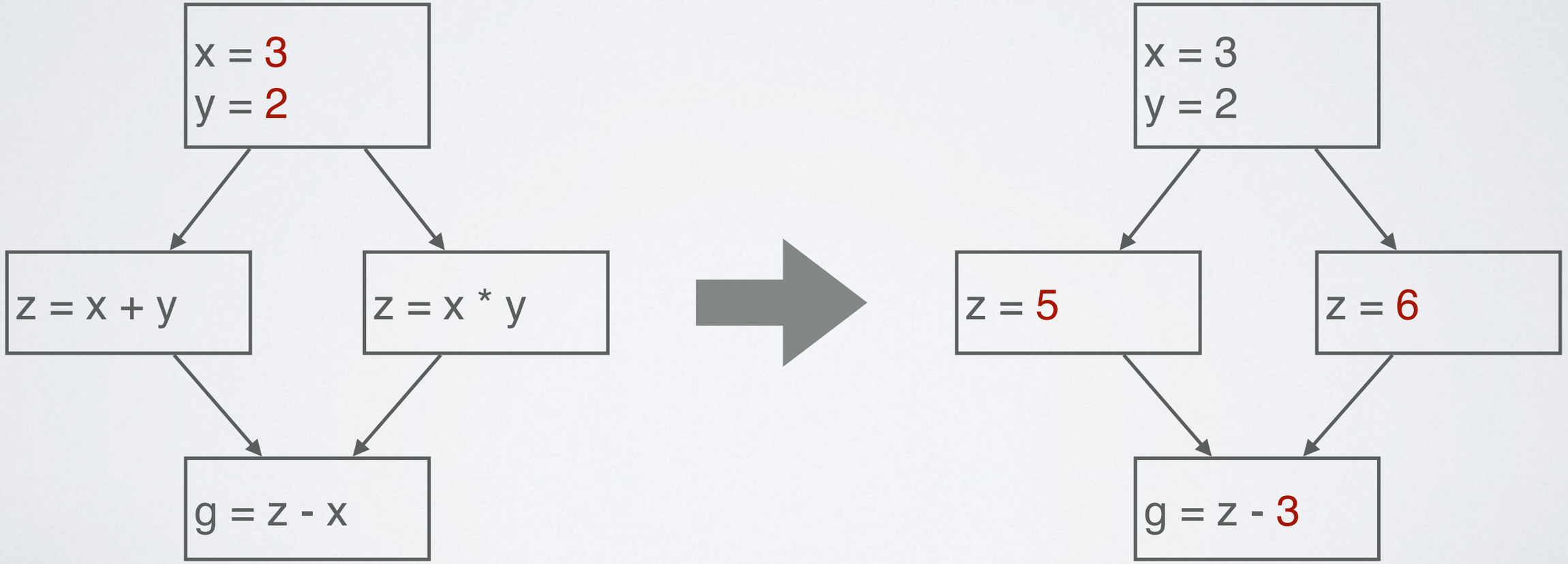
死代码消除

- 死代码（dead code）：其计算结果永远不会被使用的语句



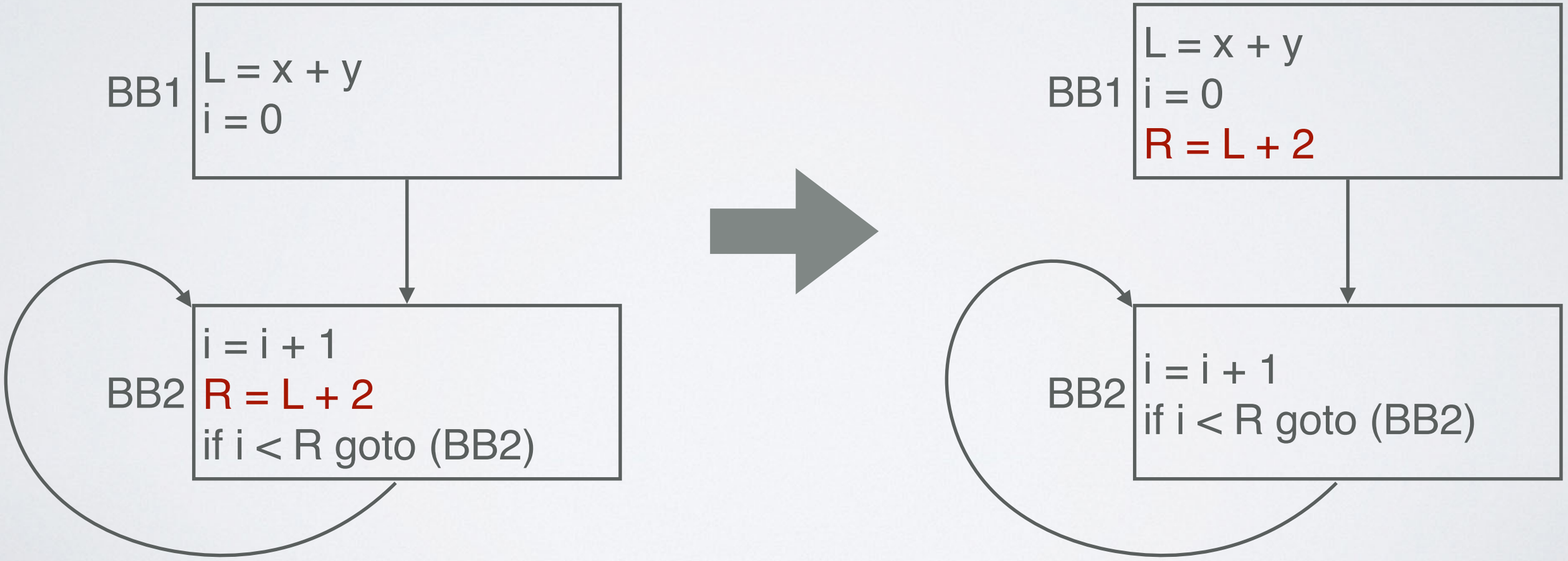
常量折叠

- 也称常量传播（constant propagation）
- 如果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替代这个表达式



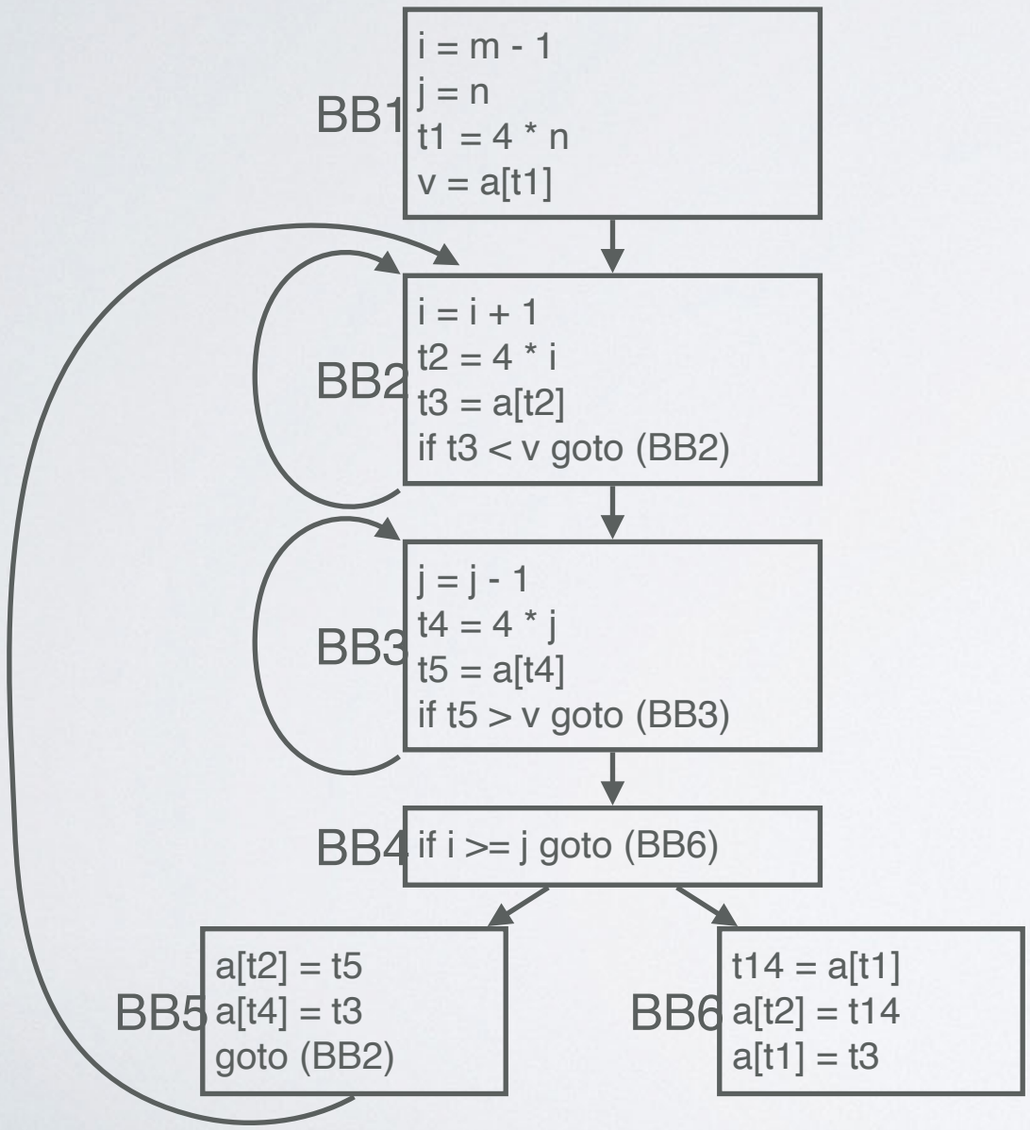
代码外提

- 也称循环不变式代码外提（loop-invariant code motion, LICM）
- 循环不变式：不管循环执行多少次都得到相同结果的表达式



强度消减（1）

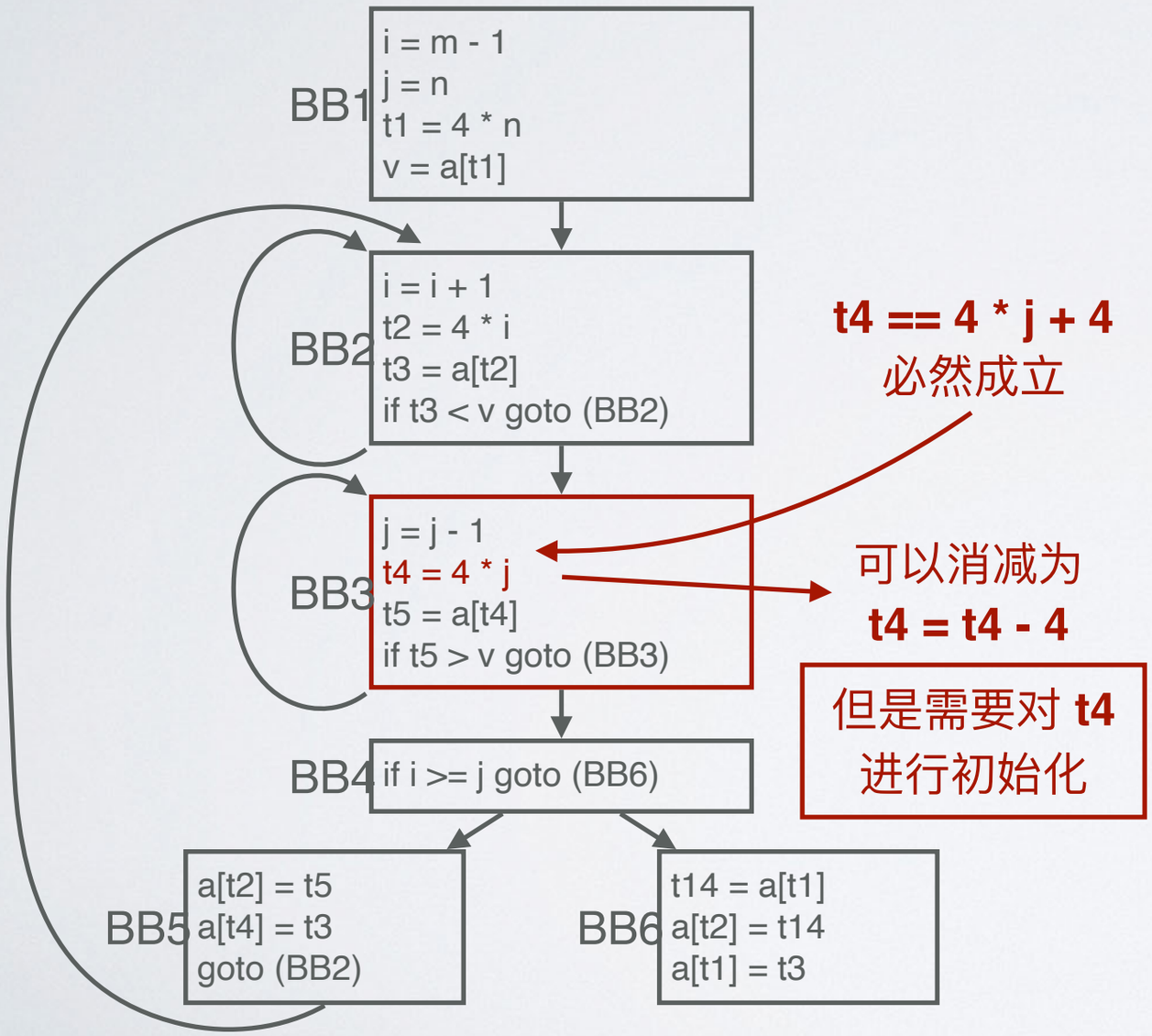
- 归纳变量（induction variable）：循环中每次 x 被赋值时总是增加一个常数，则称 x 是该循环的一个归纳变量

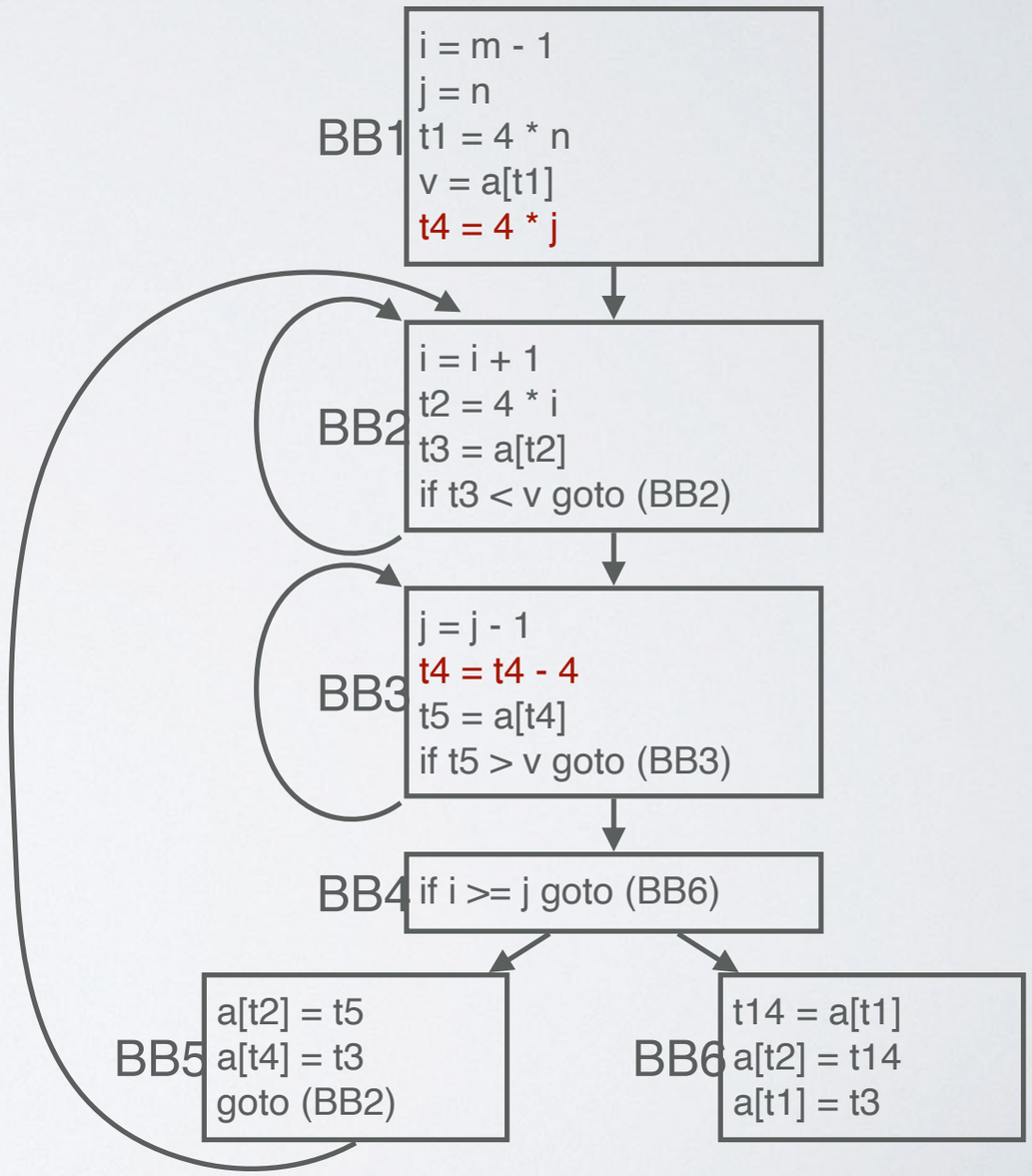


- 循环 {BB2}：
  - i，t2
- 循环 {BB3}：
  - j，t4
- 循环 {BB2, BB3, BB4, BB5}：
  - 无

强度消减（2）

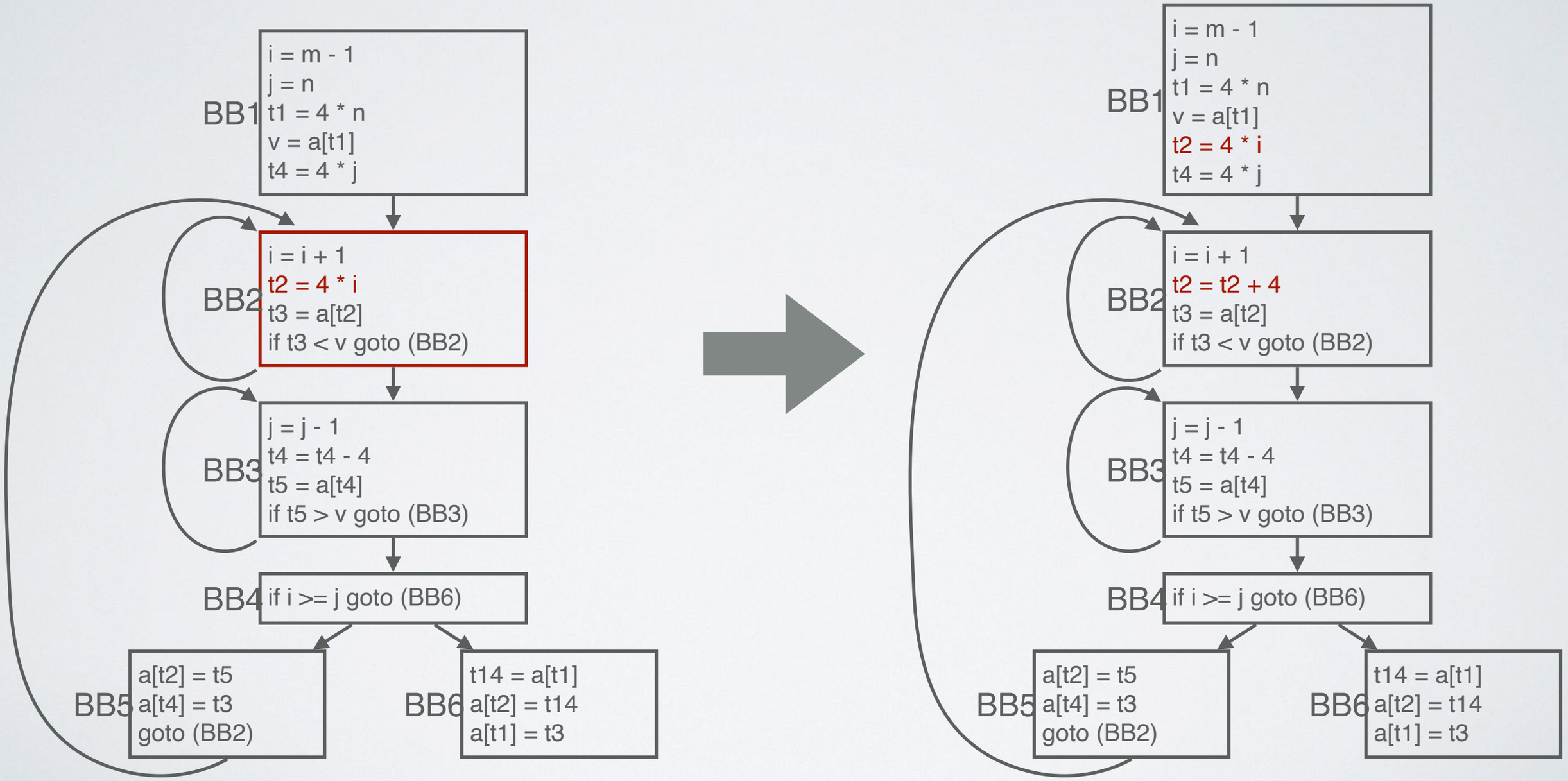
- 处理循环时，按照“从里到外”的方式进行工作





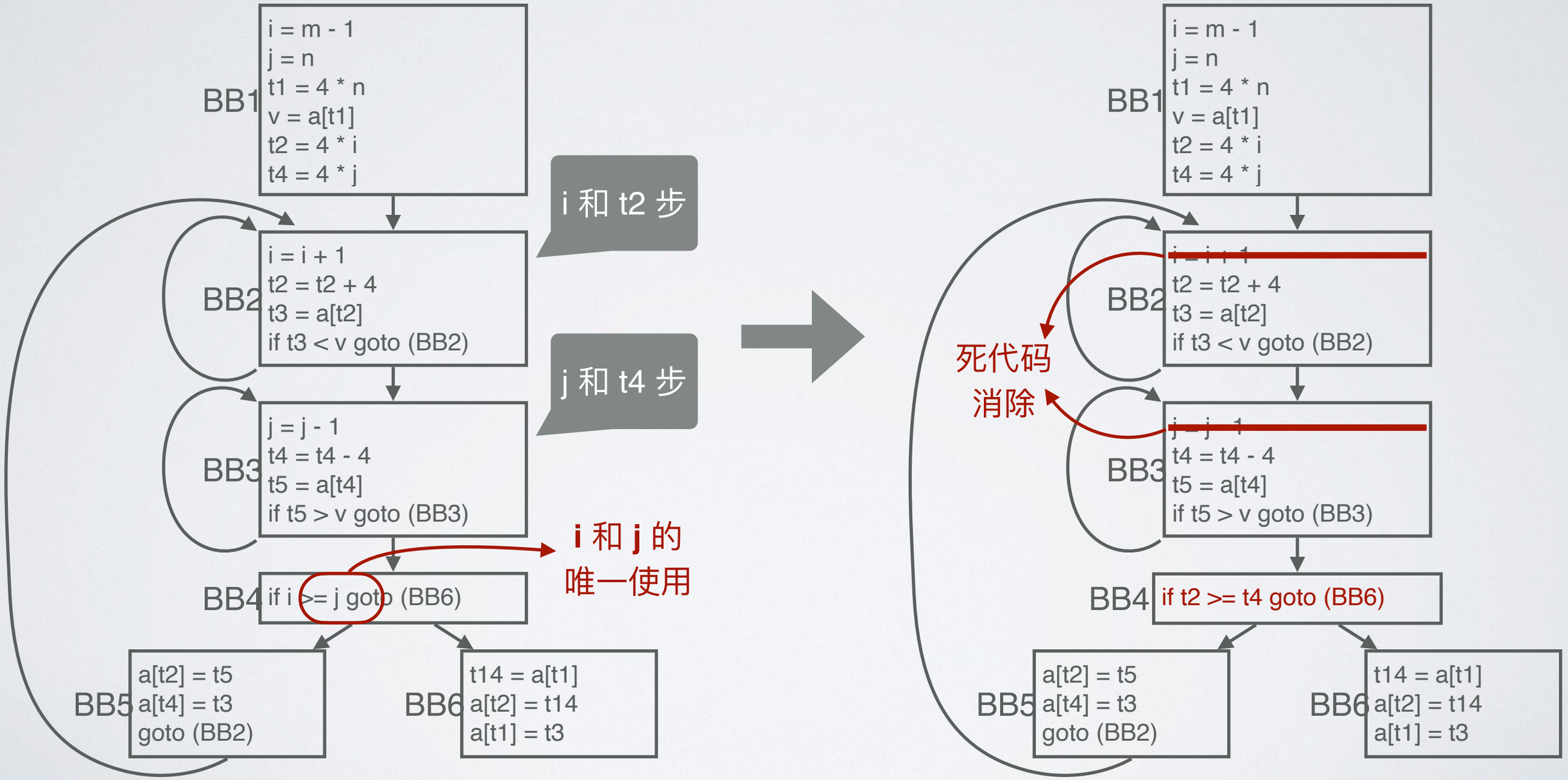
强度消减（3）

- 处理循环时，按照“从里到外”的方式进行工作

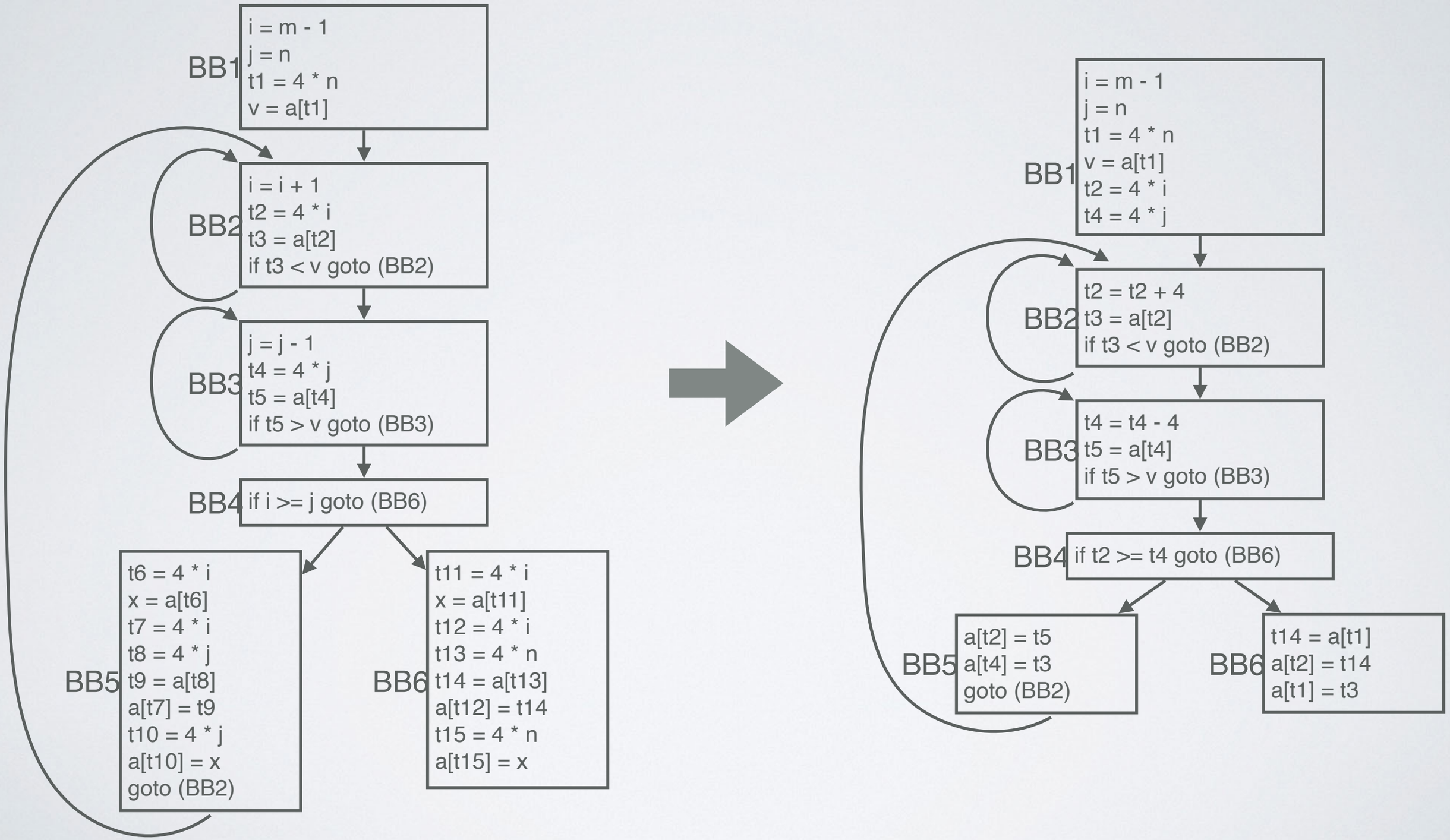


归纳变量消除

- 如果一组归纳变量的变化步调一致，则可以考虑消除一些



多项优化的结果


BB1
i = m - 1
j = n
t1 = 4 * n
v = a[t1]
BB2
i = i + 1
t2 = 4 * i
t3 = a[t2]
if t3 < v goto (BB2)
BB3
j = j - 1
t4 = 4 * j
t5 = a[t4]
if t5 > v goto (BB3)
BB4
if i >= j goto (BB6)
BB5
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 (BB2)
BB6
t11 = 4 * i
x = a[t11]
t12 = 4 * i
t13 = 4 * n
t14 = a[t13]
a[t12] = t14
t15 = 4 * n
a[t15] = x
BB1
i = m - 1
j = n
t1 = 4 * n
v = a[t1]
t2 = 4 * i
t4 = 4 * j
BB2
t2 = t2 + 4
t3 = a[t2]
if t3 < v goto (BB2)
BB3
t4 = t4 - 4
t5 = a[t4]
if t5 > v goto (BB3)
BB4
if t2 >= t4 goto (BB6)
BB5
a[t2] = t5
a[t4] = t3
goto (BB2)
BB6
t14 = a[t1]
a[t2] = t14
a[t1] = t3

本章内容

- 代码优化的常用方法
- 数据流分析简介
  - 到达定值
  - 活跃变量
  - 可用表达式
- 基于路径表达式的分析

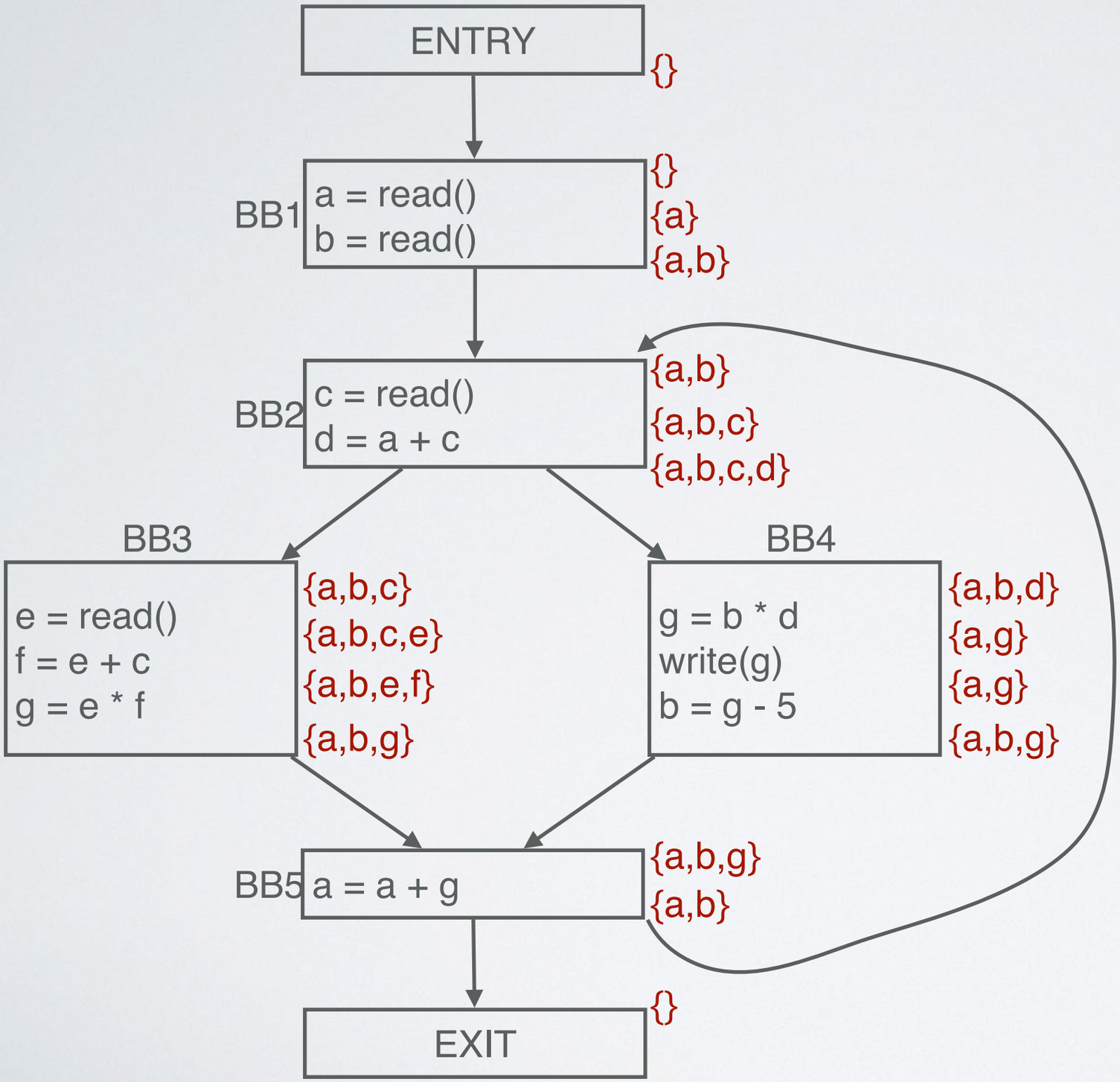
本部分课件参考了熊英飞老师开设的《软件分析技术》课程的课件。

数据流分析是什么？

- 回顾：局部分析(比如基本块 DAG 的构造)
  - 分析每条语句的作用
  - 把多条语句的作用组合起来
  - 推导出每条语句前或后的信息

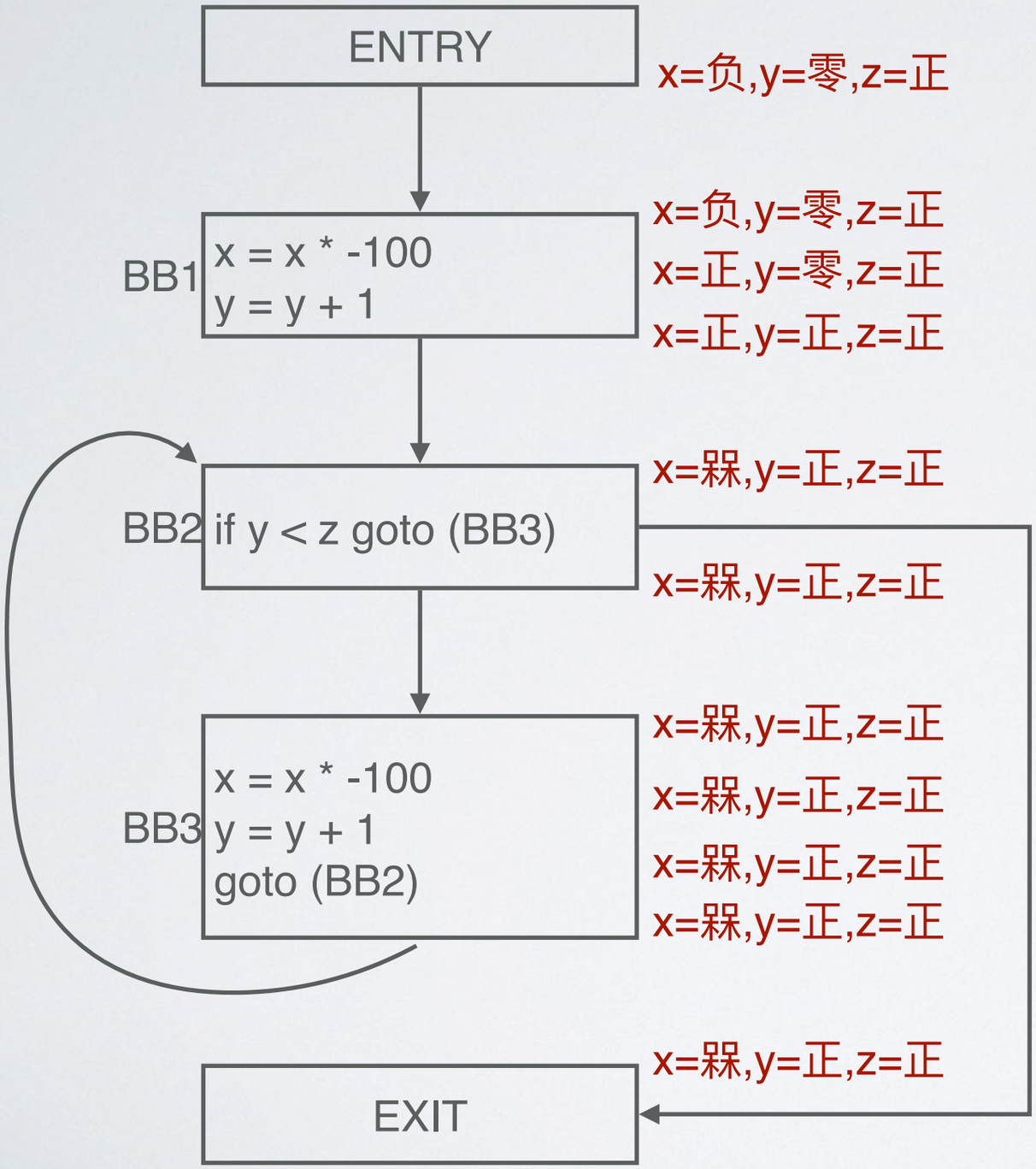
- 数据流分析是一种全局分析
  - 分析每个基本块的作用
  - 把多个基本块的作用组合起来
  - 推导出在每个基本块入口或出口处的信息
    - 基于这些信息，在基本块内使用局部分析推导出语句粒度的信息

例子：活跃变量分析



在基本块边界处的信息：
哪些变量的值可能会在之后的程序执行中使用

例子：变量符号分析



在基本块边界处的信息：
每个变量在之前的程序执行行完后可能具备的符号

正 = {所有的正数}

零 = {0}

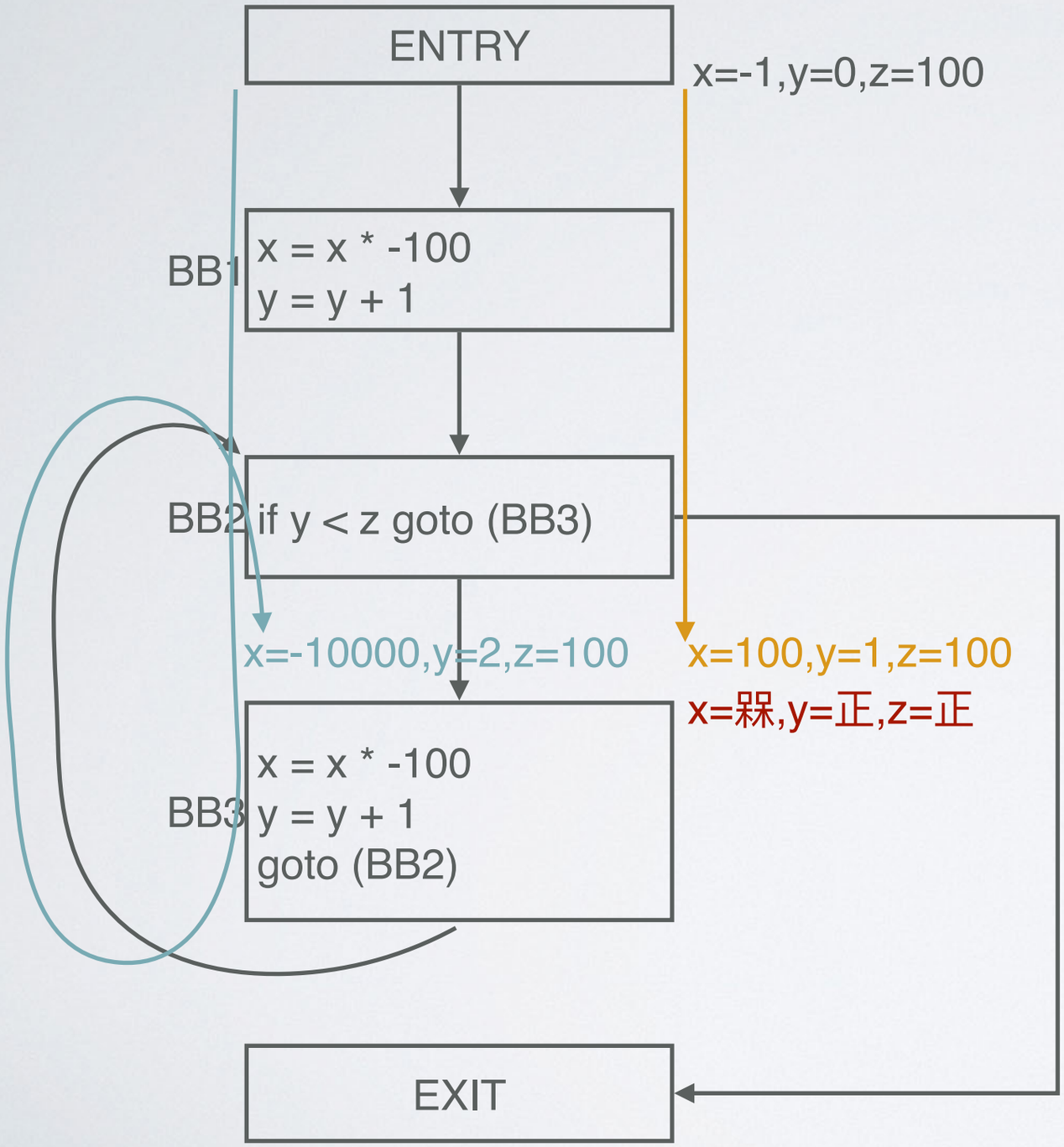
负 = {所有的负数}

⾮ = {所有的数}

数据流抽象

- 基本概念：
  - 程序点（program point）：每条语句对应其前、后两个程序点
    - 基本块内两条语句 $s_1; s_2$，s_1 后的程序点与 s_2 前的程序点相同
  - 路径（path）：程序点 $p_1, p_2, \ldots, p_n$ 构成的序列，对于任意 $1 < i < n$：
    - 点 p_i 和点 p_{i+1} 是一条语句前、后的两个程序点；或者
    - 点 p_i 指向基本块的结尾，点 p_{i+1} 指向该基本块某后继的开头
- 数据流分析推导对于每个程序点 p：
  - 前向（forward）分析：以 p 为终点的所有路径的集合的性质
  - 后向（backward）分析：以 p 为起点的所有路径的集合的性质

前向分析的例子：变量符号



在基本块边界处的信息：
每个变量在之前的程序执行完后可能具备的符号

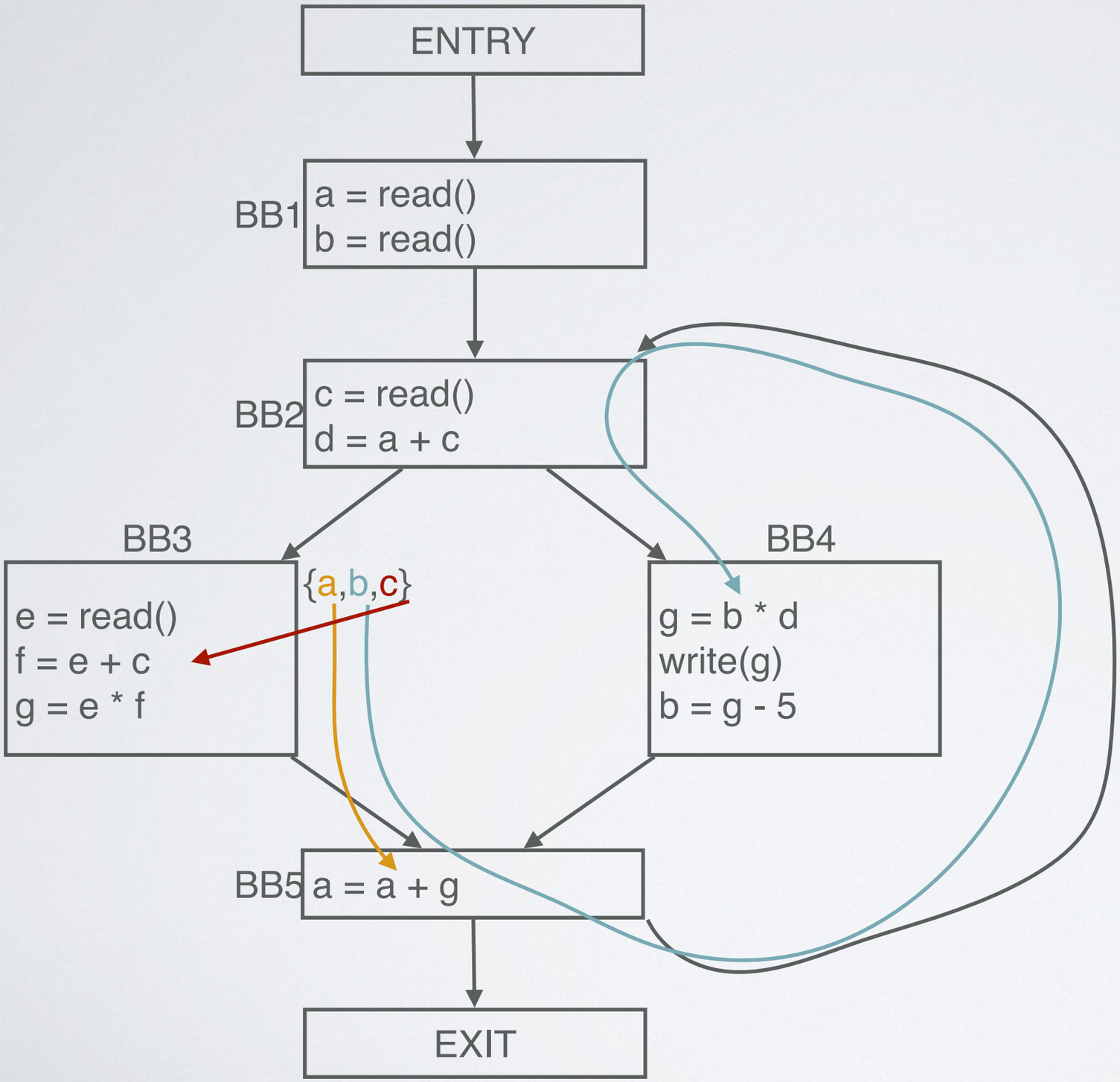
正 = {所有的正数}

零 = {0}

负 = {所有的负数}

粿 = {所有的数}

后向分析的例子：活跃变量

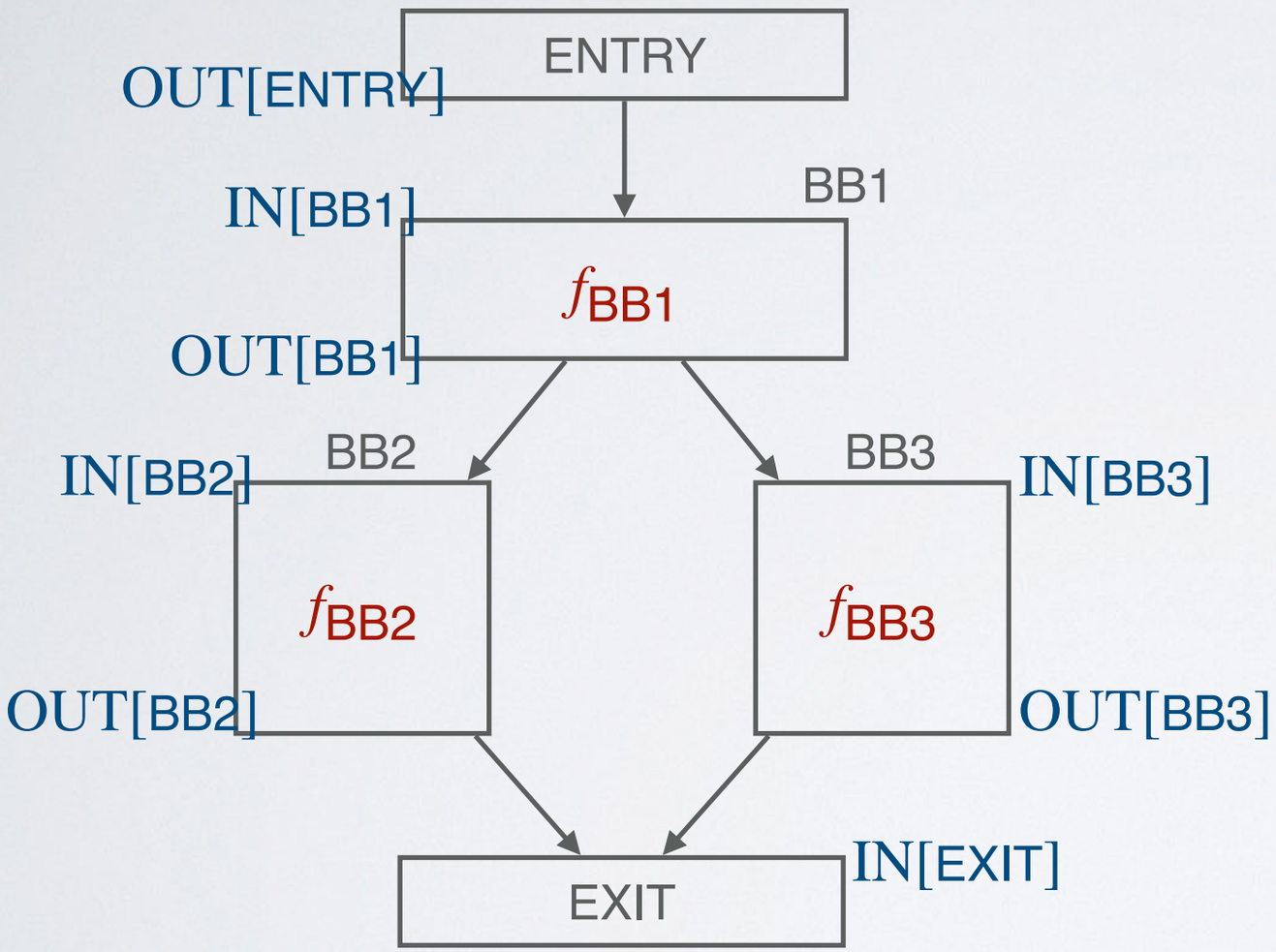


在基本块边界处的信息：
哪些变量的值可能会在之后的程序执行中使用

数据流分析模式（1）

- 把每个程序点和一个数据流值（data-flow value）关联起来，所有可能得数据流值构成的集合成为该分析的域（domain）
  - 活跃变量分析：变量的集合
  - 变量符号分析：变量到“正”、“零”、“负”、“踩”的映射
- 对每个基本块 B，用 $\mathrm{IN}[B]$ 和 $\mathrm{OUT}[B]$ 记录其入口和出口处的数据流值
  - 对于每个语句 s，用 $\mathrm{IN}[s]$ 和 $\mathrm{OUT}[s]$ 记录其前、后的数据流值
  - 如果 B 的首、尾语句为 s_1、s_n，那么 $\mathrm{IN}[B] = \mathrm{IN}[s_1]$、$\mathrm{OUT}[B] = \mathrm{OUT}[s_n]$
- 数据流问题（data-flow problem）：求出所有的 IN 和 OUT

数据流分析模式（2）



- 传递函数（transfer function）f_B 建立起基本块 B 入口和出口处数据流值的关系
- 前向分析：
  - $\mathrm{OUT}[B] = f_B(\mathrm{IN}[B])$
- 后向分析：
  - $\mathrm{IN}[B] = f_B(\mathrm{OUT}[B])$
- f_B 分析了基本块 B 的作用，其通常用基本块内的语句的传递函数 f_s 组合得到

变量符号分析：语句

B

s_1: x = -y
s_2: x = x - 1

数据流分析的安全性：当信息不充分时，采取保守处理

语句 s_1 的传递函数 f_{s_1}

	x=正	x=零	x=负	x=槑
y=正	x=负 y=正	x=负 y=正	x=负 y=正	x=负 y=正
y=零	x=零 y=零	x=零 y=零	x=零 y=零	x=零 y=零
y=负	x=正 y=负	x=正 y=负	x=正 y=负	x=正 y=负
y=槑	x=槑 y=槑	x=槑 y=槑	x=槑 y=槑	x=槑 y=槑

语句 s_2 的传递函数 f_{s_2}

	x=正	x=零	x=负	x=槑
y=正	x=槑 y=正	x=负 y=正	x=负 y=正	x=槑 y=正
y=零	x=槑 y=零	x=负 y=零	x=负 y=零	x=槑 y=零
y=负	x=槑 y=负	x=负 y=负	x=负 y=负	x=槑 y=负
y=槑	x=槑 y=槑	x=负 y=槑	x=负 y=槑	x=槑 y=槑

变量符号分析：基本块

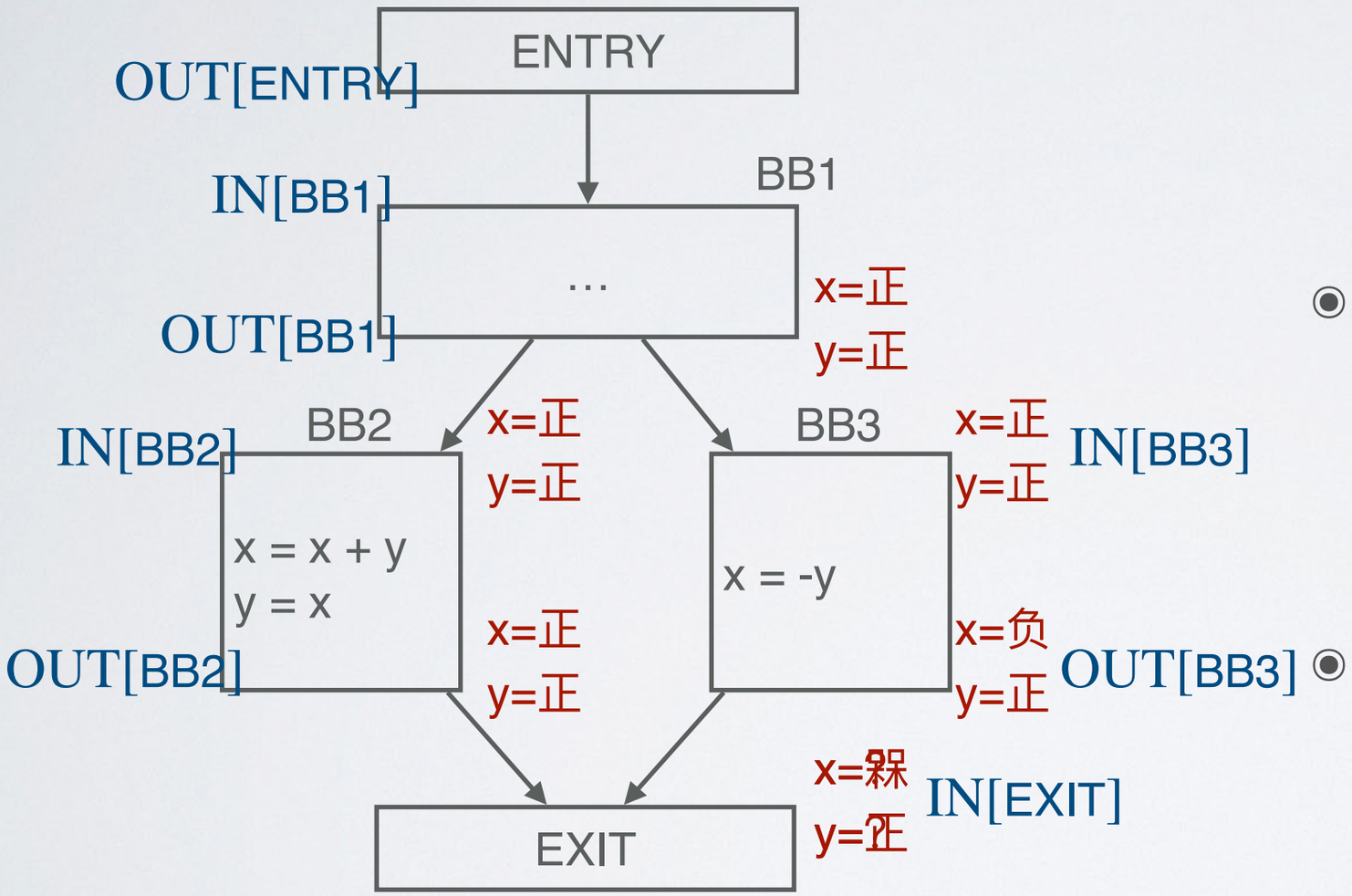
B

s_1: x = -y
s_2: x = x - 1

- 基本块 B 的传递函数 $f_B = f_{s_2} \circ f_{s_1}$，其中 $\circ$ 表示函数复合，也就是说 $(f_{s_2} \circ f_{s_1})(i) = f_{s_2}(f_{s_1}(i))$；这也体现出变量符号分析是一个前向分析

	x=正	x=零	x=负	x=㚔
y=正	x=负 y=正	x=负 y=正	x=负 y=正	x=负 y=正
y=零	x=负 y=零	x=负 y=零	x=负 y=零	x=负 y=零
y=负	x=㚔 y=负	x=㚔 y=负	x=㚔 y=负	x=㚔 y=负
y=㚔	x=㚔 y=㚔	x=㚔 y=㚔	x=㚔 y=㚔	x=㚔 y=㚔

变量符号分析：分支



- 控制流约束：

$$IN[BB2] = OUT[BB1]$$
$$IN[BB3] = OUT[BB1]$$
$$IN[EXIT] = ?$$

- 定义交汇运算 $\wedge$：

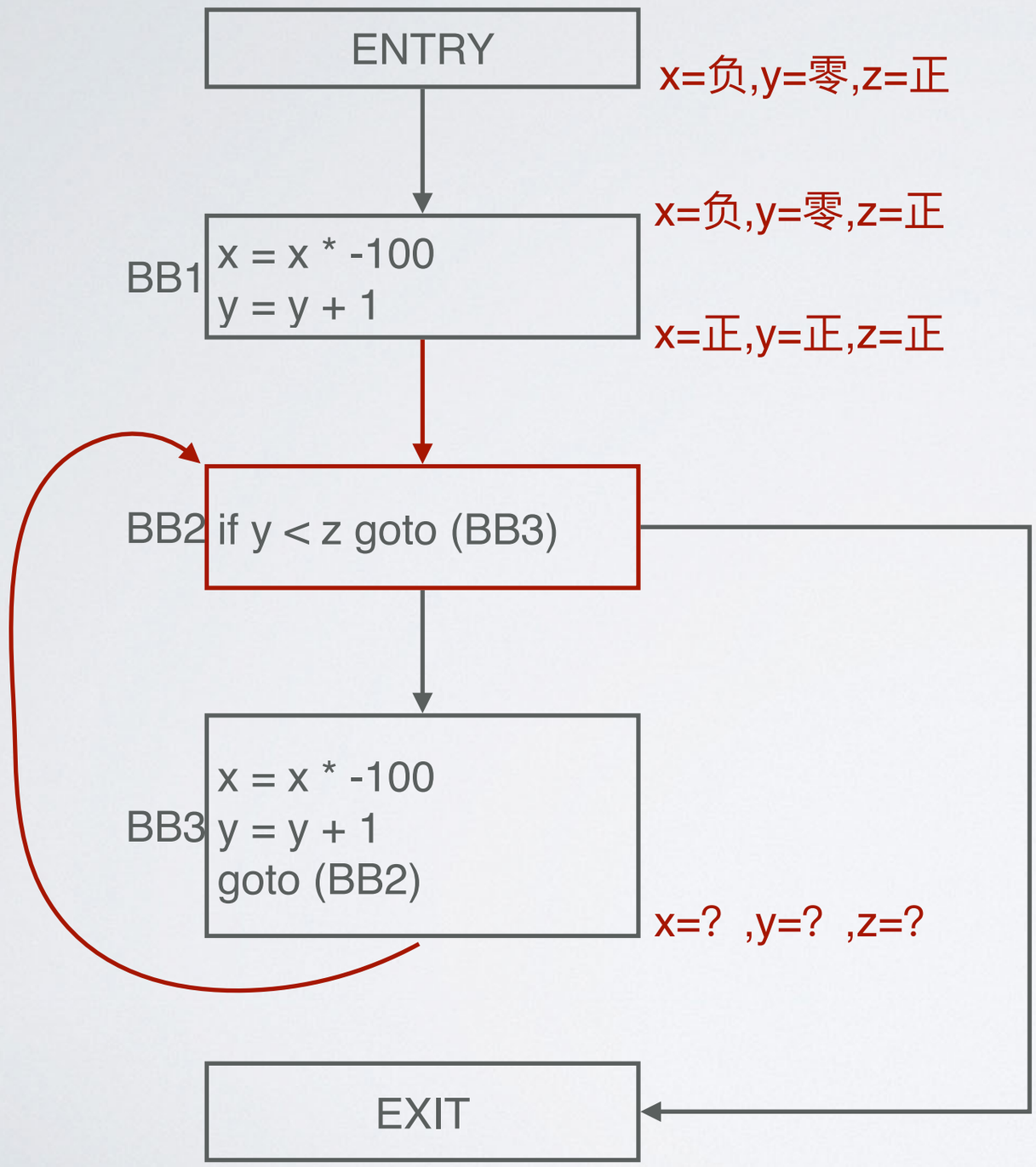
$$v_1 \wedge v_2 = v_1, \text{ 若 } v_1 = v_2$$
$$v_1 \wedge v_2 = \bot, \text{ 其它情况}$$

分支的处理：

$$IN[EXIT] = OUT[BB2] \wedge OUT[BB3]$$

$$IN[B] = \bigwedge_{P\text{是}B\text{的前驱}} OUT[P]$$

变量符号分析：循环（1）

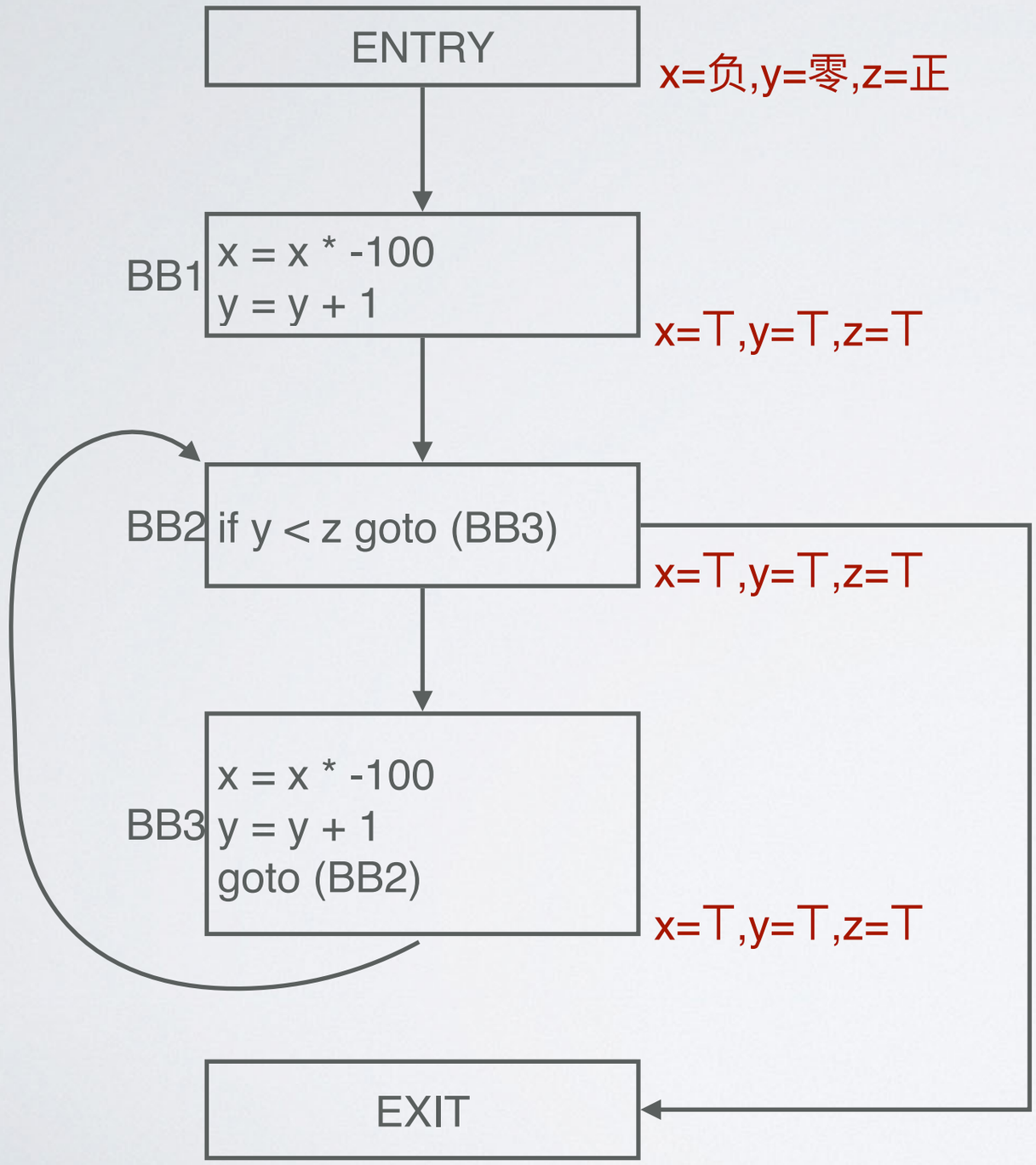


- 问题：**BB2** 有两个前驱 **BB1** 和 **BB3**，而 **BB3** 的值未知
- 看作一个解方程问题：

$$\mathrm{IN[BB2]} = \mathrm{OUT[BB1]} \wedge \mathrm{OUT[BB3]}$$
$$\mathrm{OUT[BB2]} = f_{\mathrm{BB2}}(\mathrm{IN[BB2]})$$
$$\mathrm{IN[BB3]} = \mathrm{OUT[BB2]}$$
$$\mathrm{OUT[BB3]} = f_{\mathrm{BB3}}(\mathrm{IN[BB3]})$$
$$\mathrm{IN[EXIT]} = \mathrm{OUT[BB2]}$$

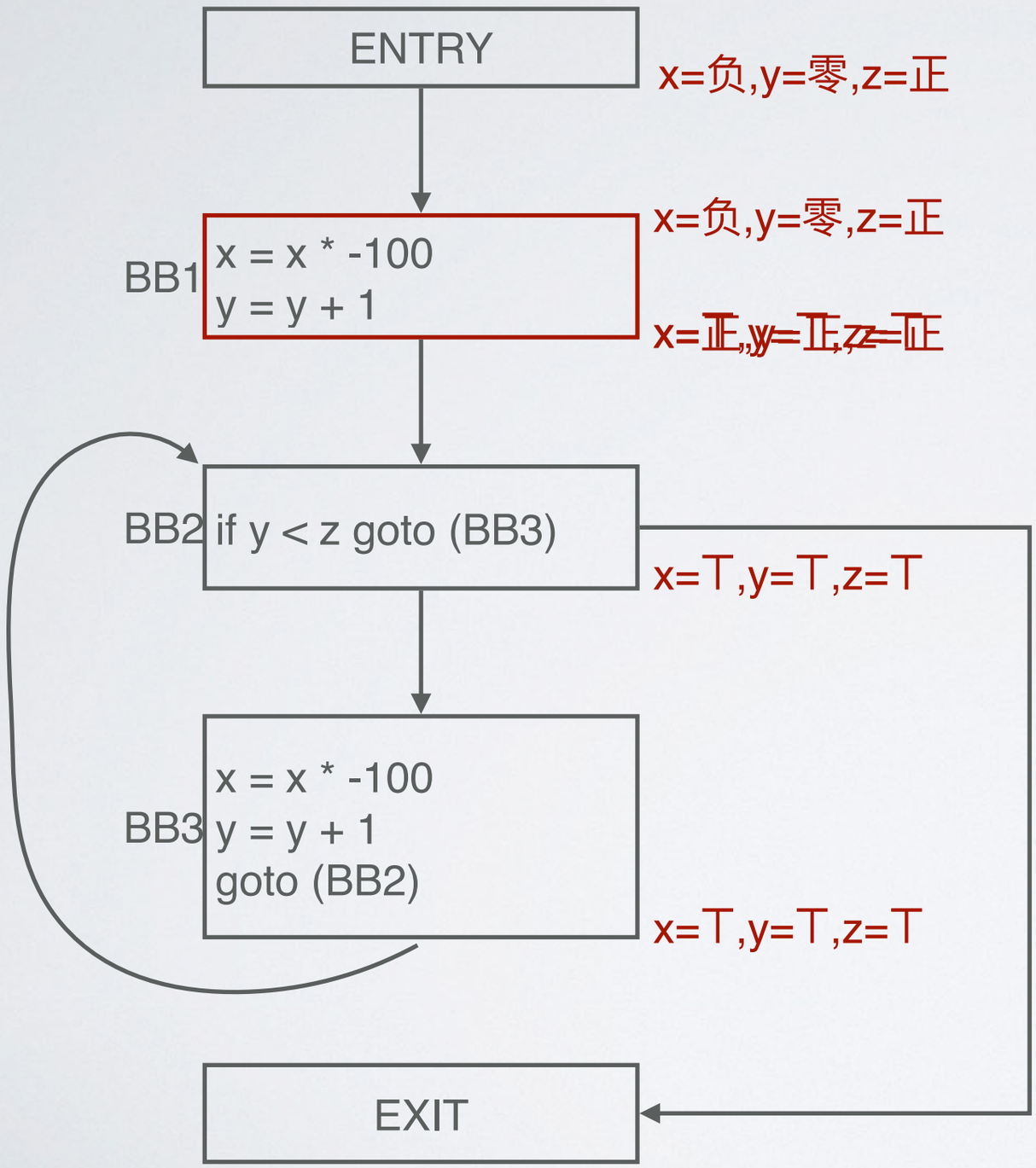
- 目标：找到“最精确”的解

变量符号分析：循环（2）



- 引入一种新的数据流值 T，称为顶值（top），表示还没有分析出任何信息
- 任意数据流值与 T 的交汇等于其自身：$v \wedge \top = v$
- 使用 T 来初始化程序点的数据流值
- 在分析过程中，通过得到的信息逐步精化数据流值

变量符号分析：循环（3）



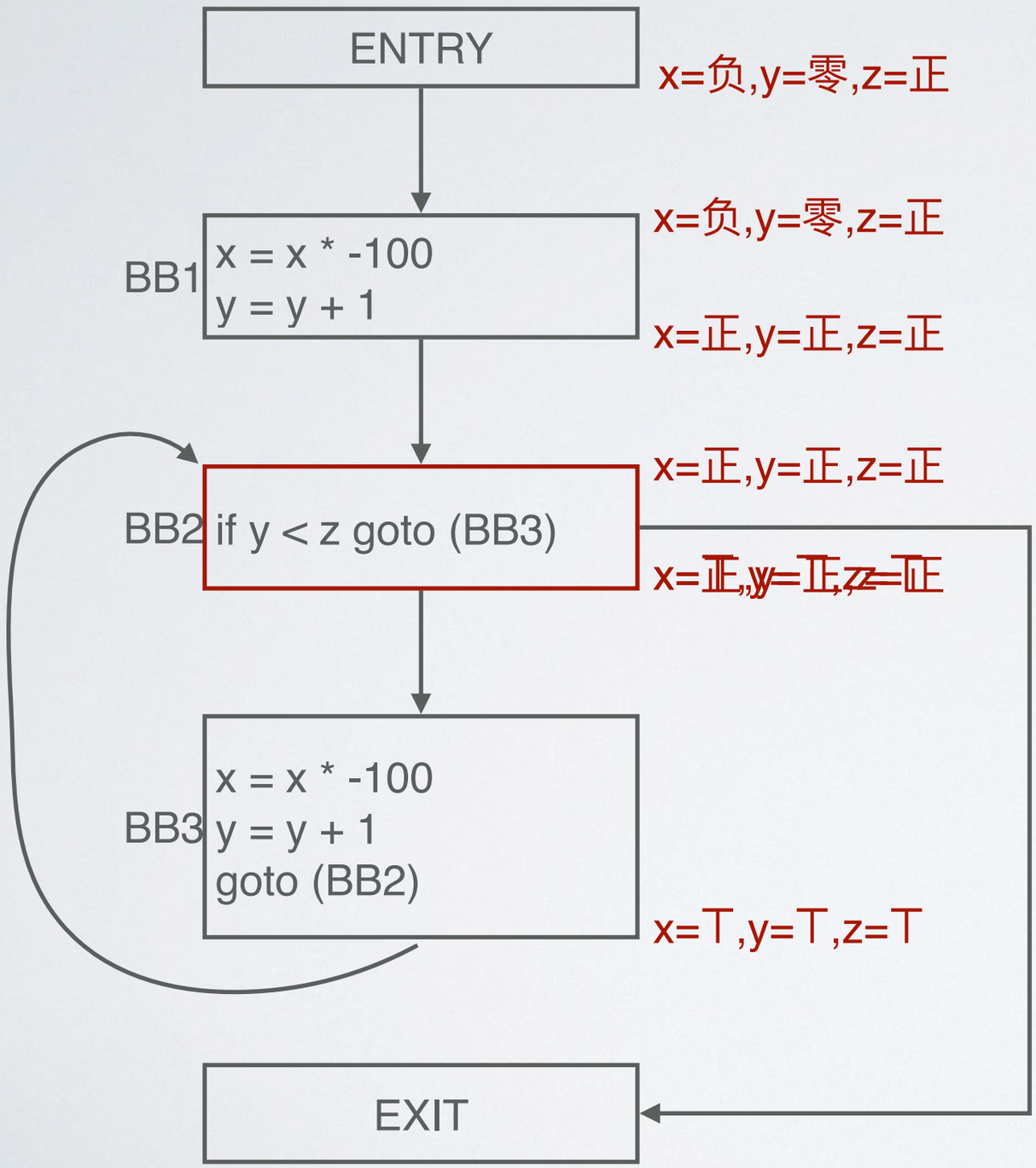
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{IN}[B] = \bigwedge_{P\text{是}B\text{的前驱}} \mathrm{OUT}[P]$$

$$\mathrm{OUT}[B] = f_B(\mathrm{IN}[B])$$

- 重复该过程直到没有基本块需要更新

变量符号分析：循环（4）



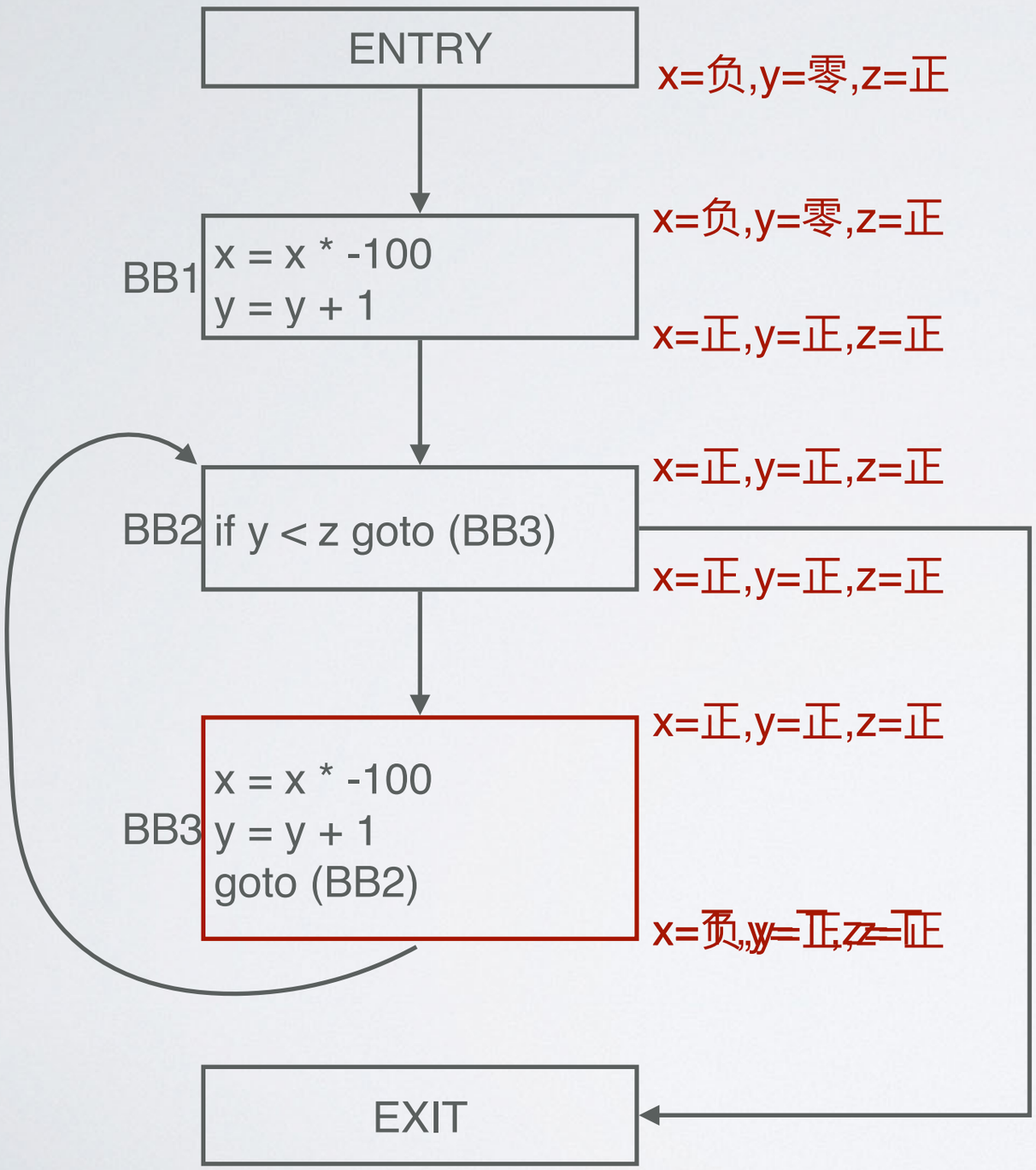
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{IN}[B] = \bigwedge_{P\text{是}B\text{的前驱}} \mathrm{OUT}[P]$$

$$\mathrm{OUT}[B] = f_B(\mathrm{IN}[B])$$

- 重复该过程直到没有基本块需要更新

变量符号分析：循环（5）



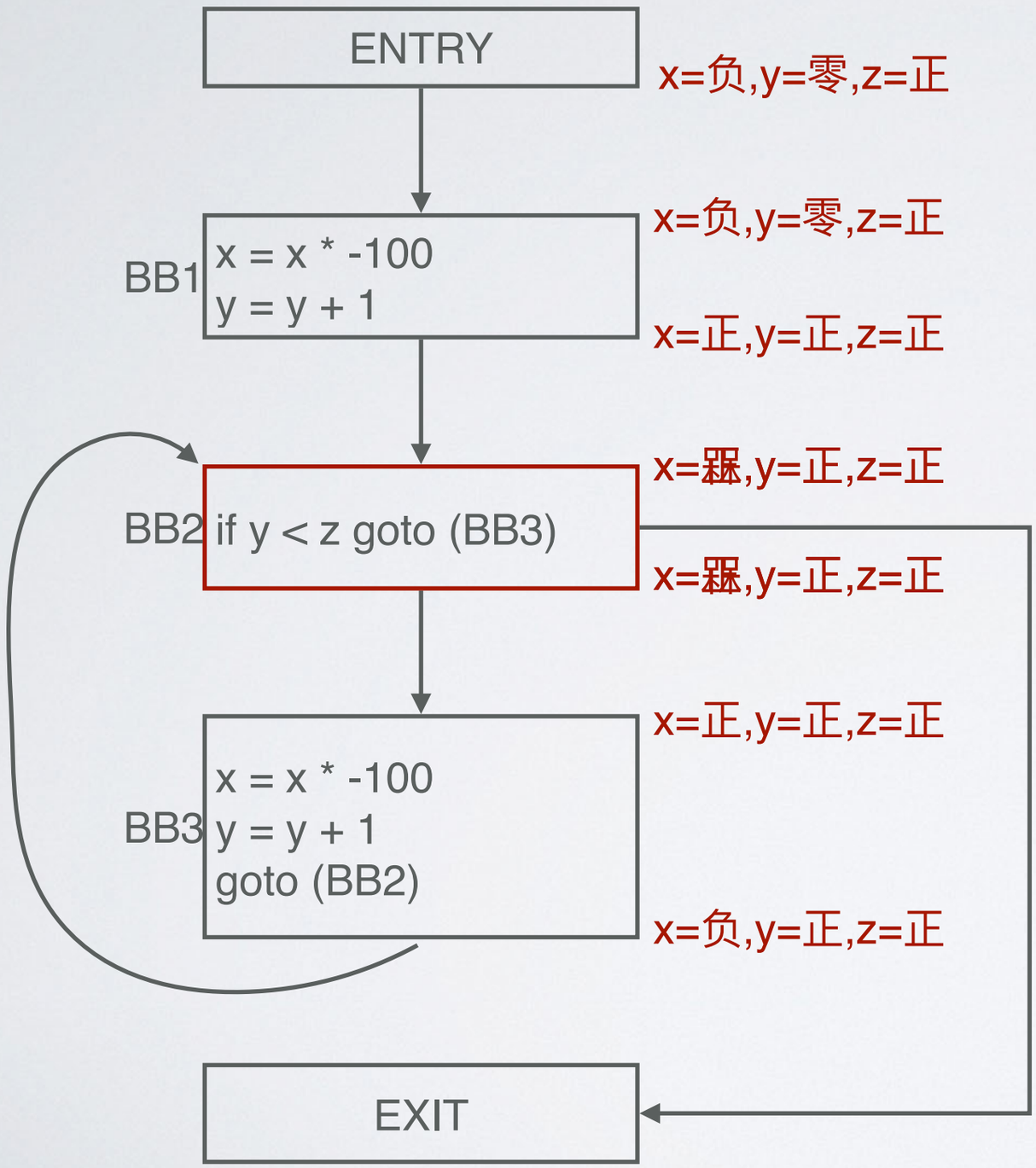
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{IN}[B] = \bigwedge_{P\text{是}B\text{的前驱}} \mathrm{OUT}[P]$$

$$\mathrm{OUT}[B] = f_B(\mathrm{IN}[B])$$

- 重复该过程直到没有基本块需要更新

变量符号分析：循环（6）



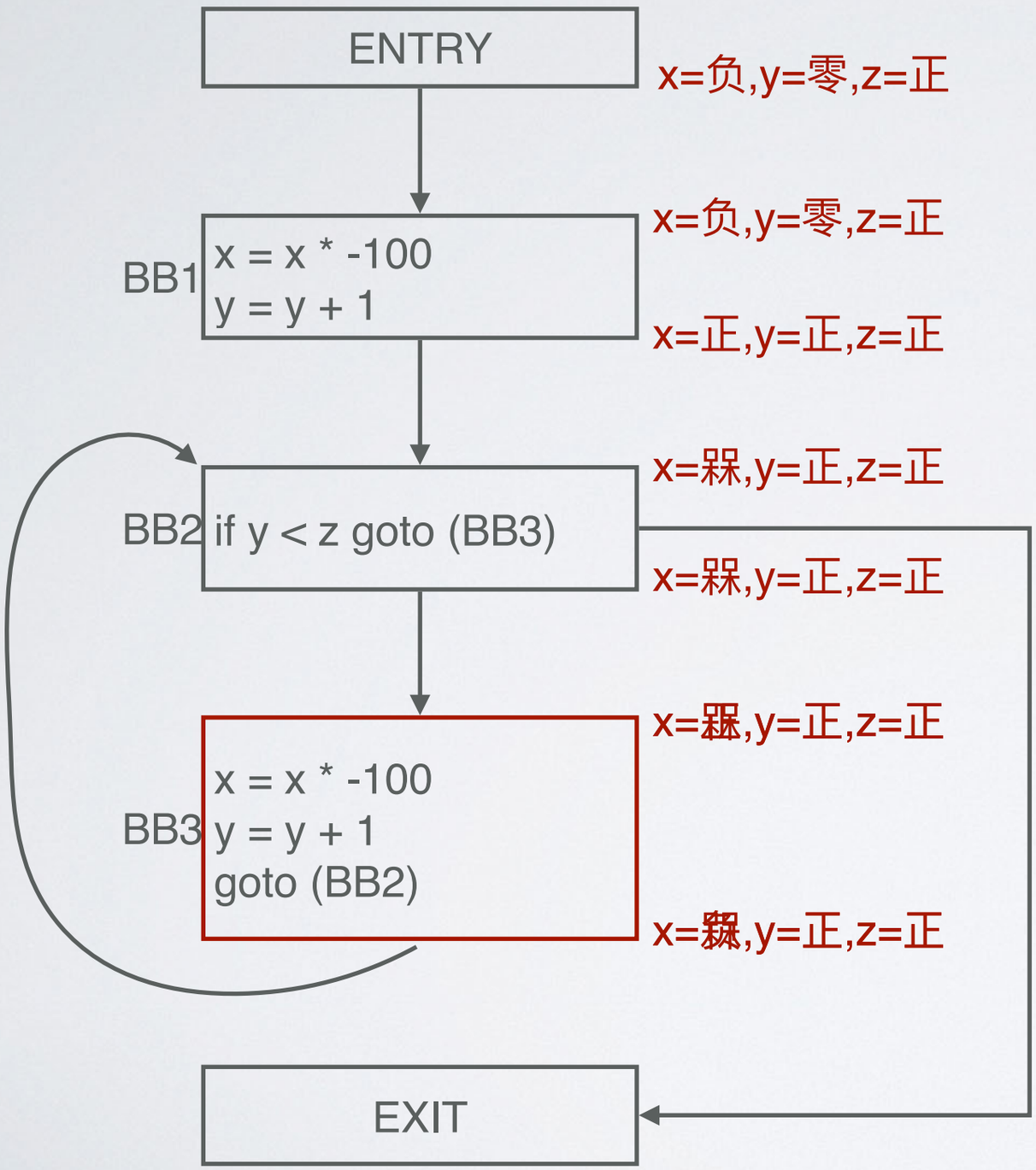
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{IN}[B] = \bigwedge_{P\text{是}B\text{的前驱}} \mathrm{OUT}[P]$$

$$\mathrm{OUT}[B] = f_B(\mathrm{IN}[B])$$

- 重复该过程直到没有基本块需要更新

变量符号分析：循环（7）



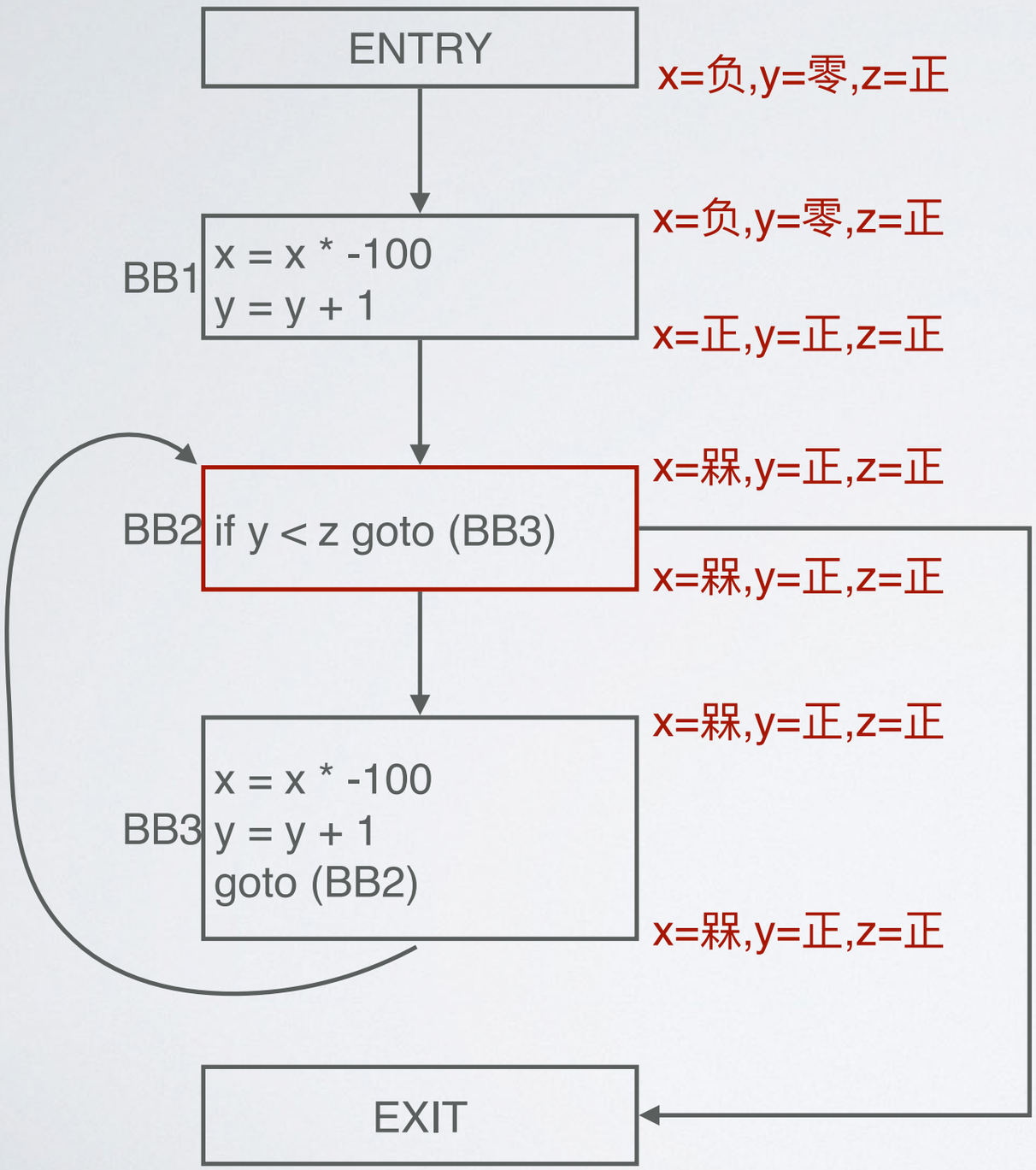
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{IN}[B] = \bigwedge_{P\text{是}B\text{的前驱}} \mathrm{OUT}[P]$$

$$\mathrm{OUT}[B] = f_B(\mathrm{IN}[B])$$

- 重复该过程直到没有基本块需要更新

变量符号分析：循环（8）



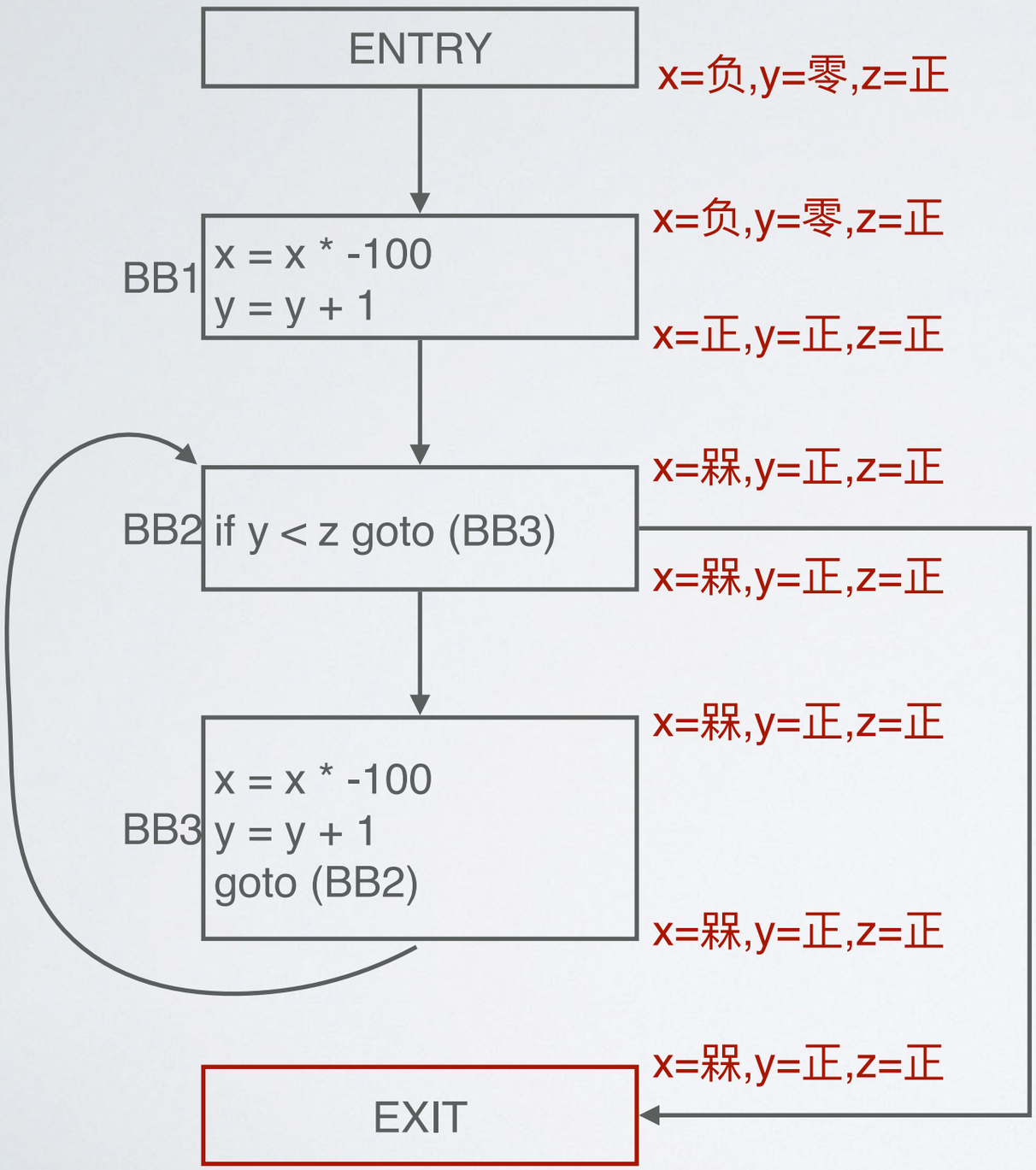
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{IN}[B] = \bigwedge_{P\text{是}B\text{的前驱}} \mathrm{OUT}[P]$$

$$\mathrm{OUT}[B] = f_B(\mathrm{IN}[B])$$

- 重复该过程直到没有基本块需要更新

变量符号分析：循环（9）



- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{IN}[B] = \bigwedge_{P\text{是}B\text{的前驱}} \mathrm{OUT}[P]$$

$$\mathrm{OUT}[B] = f_B(\mathrm{IN}[B])$$

- 重复该过程直到没有基本块需要更新

小结：前向分析模式

- 数据流分析的域 V，交汇运算 $\wedge : V \times V \to V$，顶值 $\top \in V$
- 每个基本块 B 的传递函数 $f_B : V \to V$（从入口到出口）
- 边界条件：$\mathrm{OUT}[\mathrm{ENTRY}] = v_{\mathrm{ENTRY}}$
- 初始值：$\mathrm{OUT}[B] = \top \quad (B \neq \mathrm{ENTRY})$
- 方程组：对任意 $B \neq \mathrm{ENTRY}$，有

$$\mathrm{IN}[B] = \bigwedge_{P\text{是}B\text{的前驱}} \mathrm{OUT}[P]$$

$$\mathrm{OUT}[B] = f_B(\mathrm{IN}[B])$$

活跃变量分析：语句

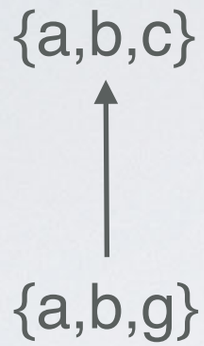
B

s_1: e = read()
s_2: f = e + c
s_3: g = e * f

$def_{s_1} = \{e\}, use_{s_1} = \varnothing$
$def_{s_2} = \{f\}, use_{s_2} = \{e, c\}$
$def_{s_3} = \{g\}, use_{s_3} = \{e, f\}$

- 对每条语句 s，定义如下集合：
  - def_s：语句 s 定值的变量
  - use_s：语句 s 使用的变量
- 语句 s 的传递函数 f_s 的输入为 s 后的活跃变量集合，输出为 s 前的活跃变量集合（活跃变量分析是一个后向分析）
  - $f_s(O) = (O - def_s) \cup use_s$

活跃变量分析：基本块



B

s_1: e = read()
s_2: f = e + c
s_3: g = e * f

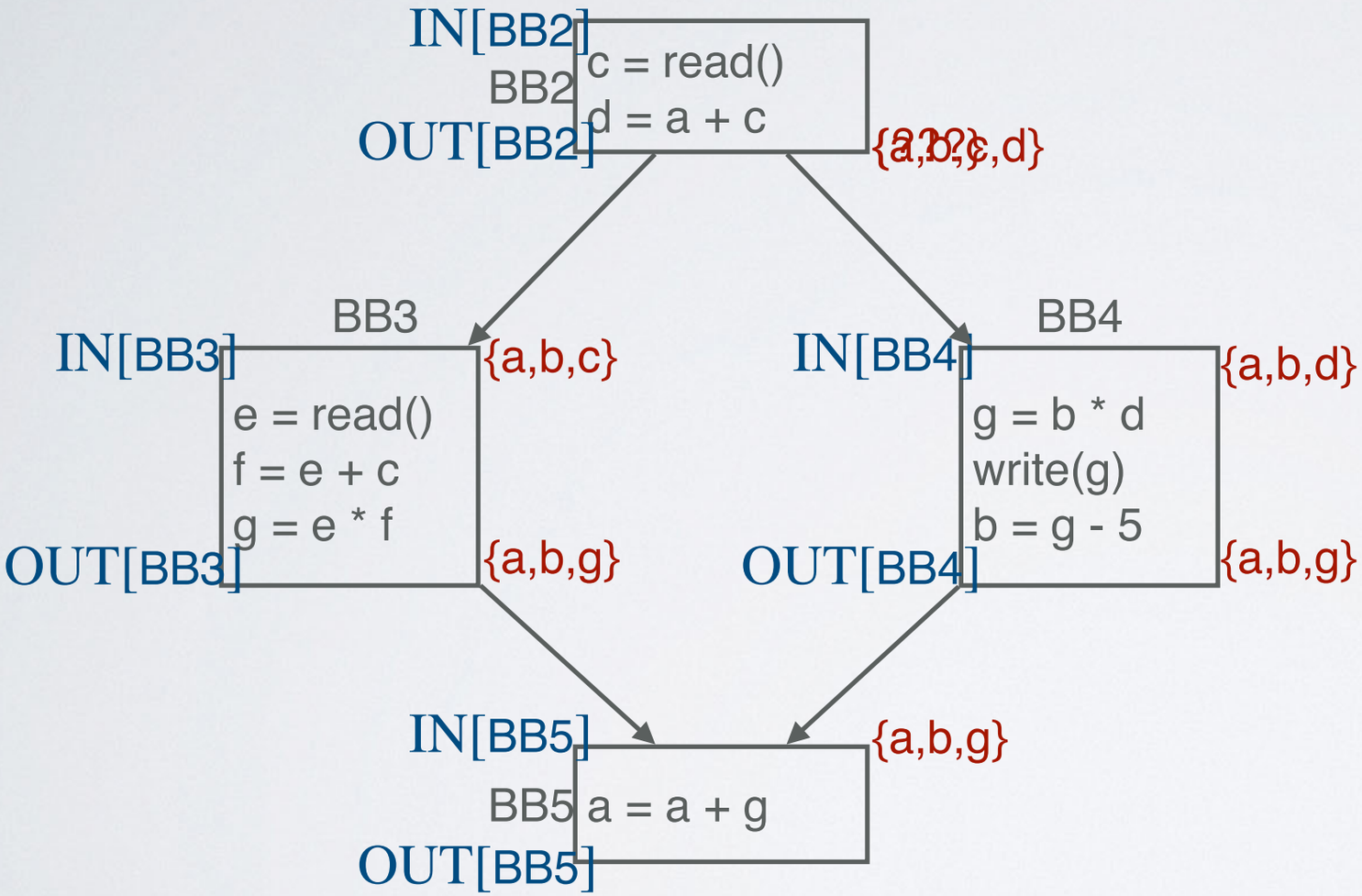
$$def_{s_1} = \{e\}, use_{s_1} = \varnothing$$
$$def_{s_2} = \{f\}, use_{s_2} = \{e, c\}$$
$$def_{s_3} = \{g\}, use_{s_3} = \{e, f\}$$

$$def_B = \{e,f,g\}, use_B = \{c\}$$

- 语句 s 的传递函数：$f_s(O) = (O - def_s) \cup use_s$
- 基本块 B 的传递函数 $f_B = f_{s_1} \circ f_{s_2} \circ f_{s_3}$；再次体现后向分析

$$f_B(O) = f_{s_1}(f_{s_2}(f_{s_3}(O)))$$
$$= ((((O - def_{s_3}) \cup use_{s_3}) - def_{s_2}) \cup use_{s_2}) - def_{s_1}) \cup use_{s_1}$$
$$= (O - (def_{s_3} \cup def_{s_2} \cup def_{s_1})) \cup ((((use_{s_3} - def_{s_2}) \cup use_{s_2}) - def_{s_1}) \cup use_{s_1})$$
$$f_B(O) = (O - def_B) \cup use_B$$

活跃变量分析：分支



- 控制流约束：

  $\mathrm{OUT}[\mathrm{BB3}] = \mathrm{IN}[\mathrm{BB5}]$

  $\mathrm{OUT}[\mathrm{BB4}] = \mathrm{IN}[\mathrm{BB5}]$

  $\mathrm{OUT}[\mathrm{BB2}] = ?$

- 定义交汇运算 $\wedge$：

  $O_1 \wedge O_2 = O_1 \cup O_2$

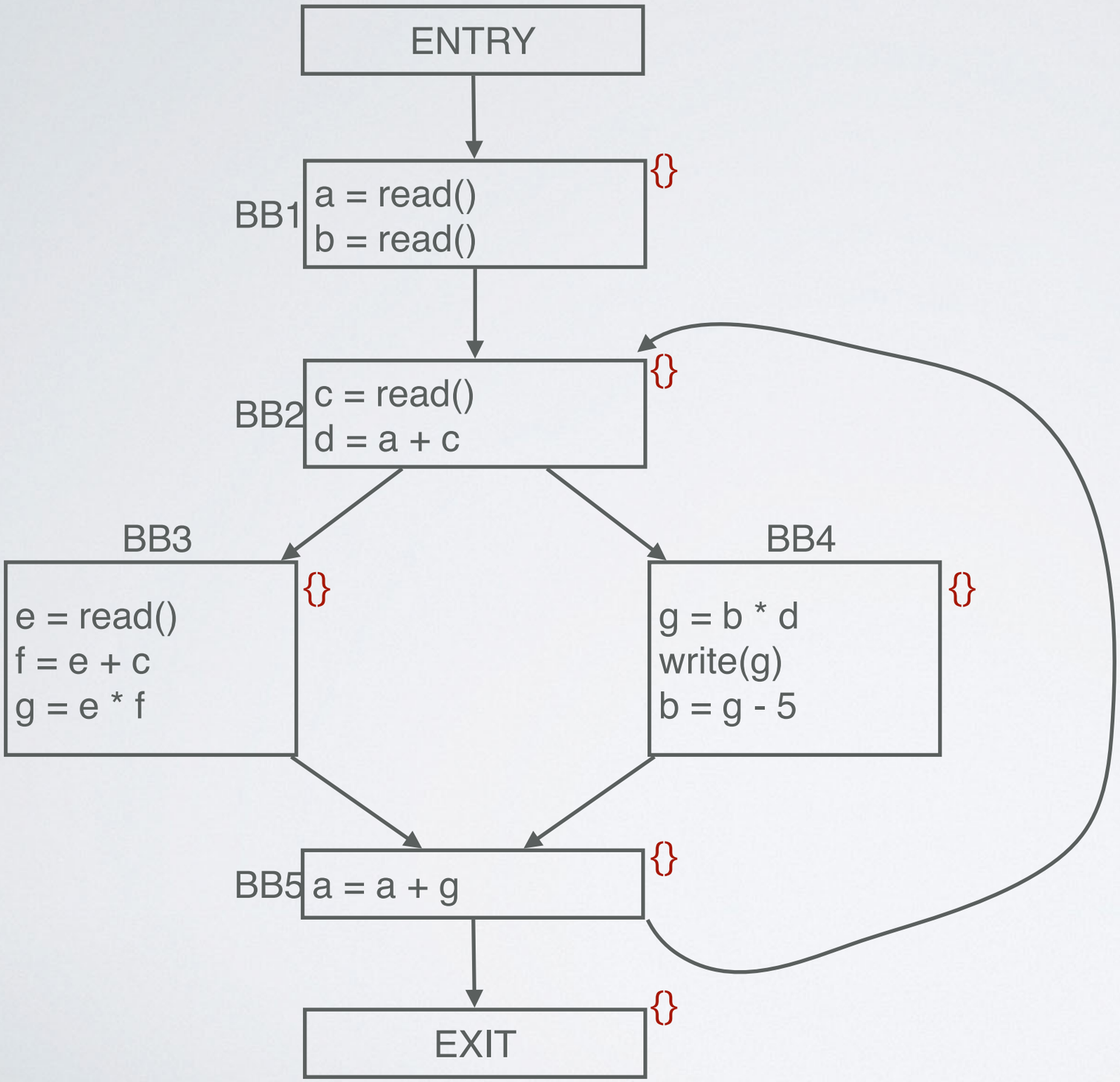
  即：在任意后继中活跃则认为是活跃的

- 分支的处理：

  $\mathrm{OUT}[\mathrm{BB2}] = \mathrm{IN}[\mathrm{BB3}] \wedge \mathrm{IN}[\mathrm{BB4}]$

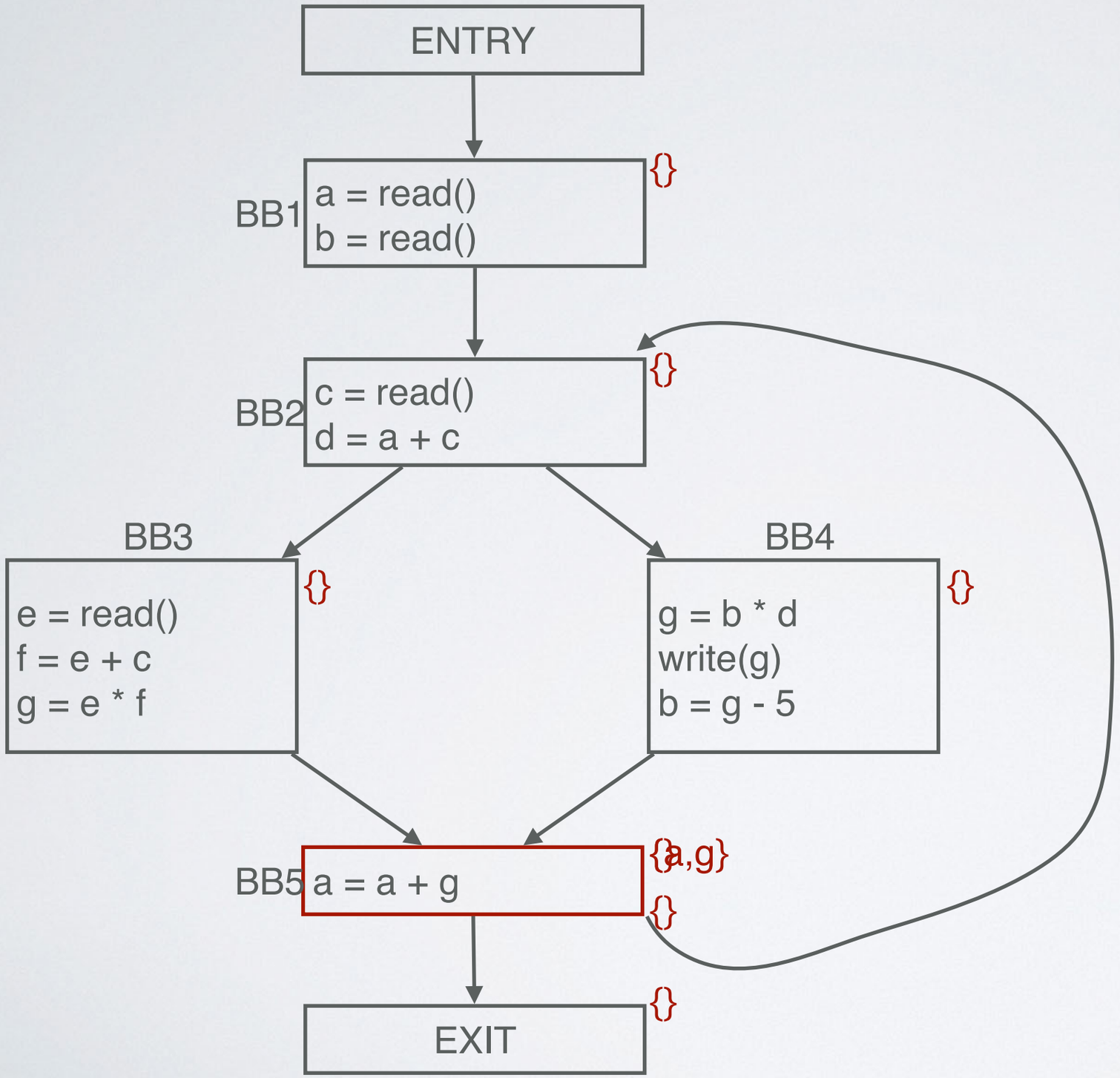
$$\mathrm{OUT}[B] = \bigwedge_{S\text{是}B\text{的后继}} \mathrm{IN}[S]$$

活跃变量分析：循环（1）



- 顶值（top)为空集

活跃变量分析：循环（2）



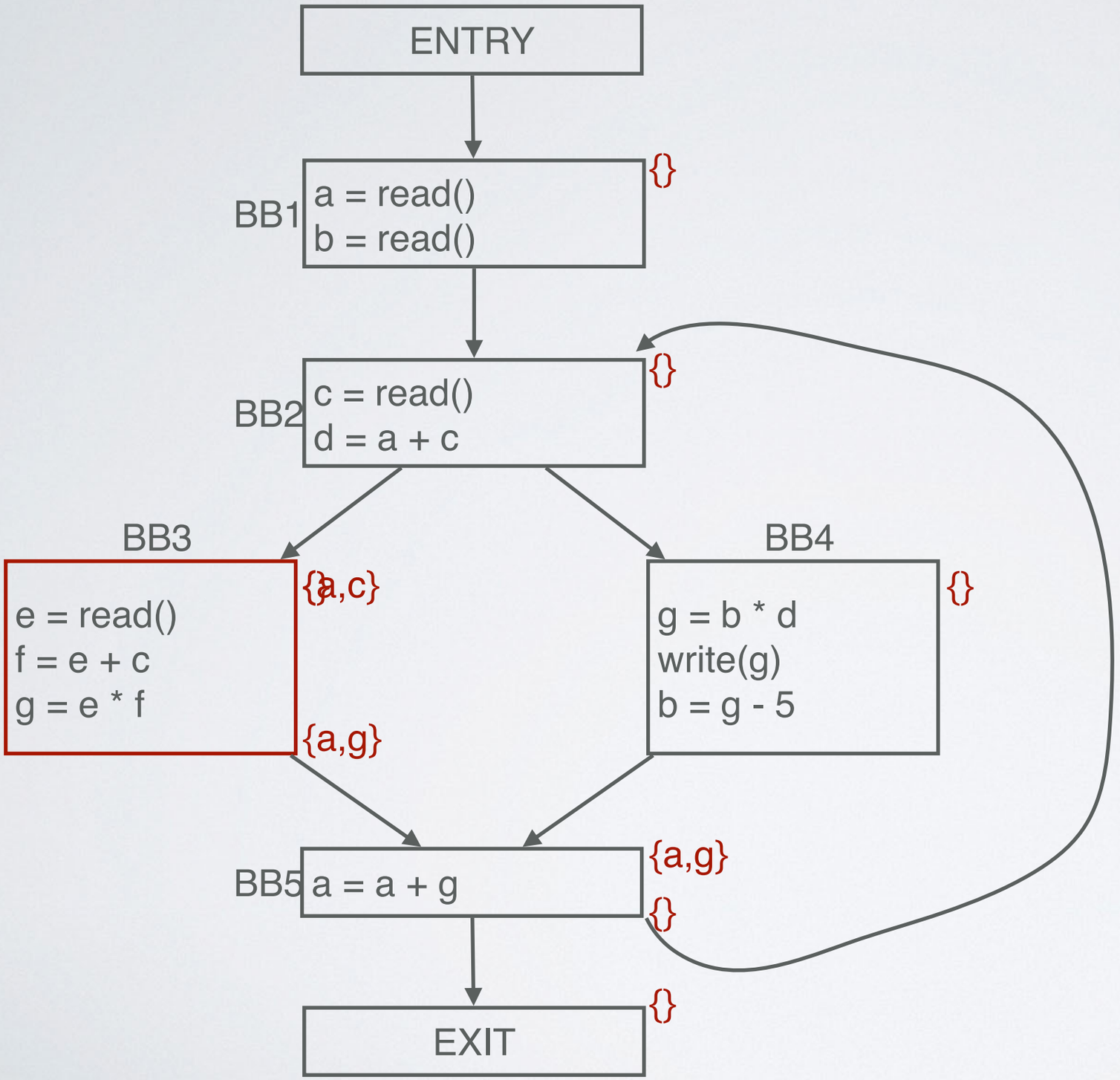
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{OUT}[B] = \bigwedge_{S\text{是}B\text{后继}} \mathrm{IN}[S]$$

$$\mathrm{IN}[B] = f_B(\mathrm{OUT}[B])$$

- 重复该过程直到没有基本块需要更新

活跃变量分析：循环（3）



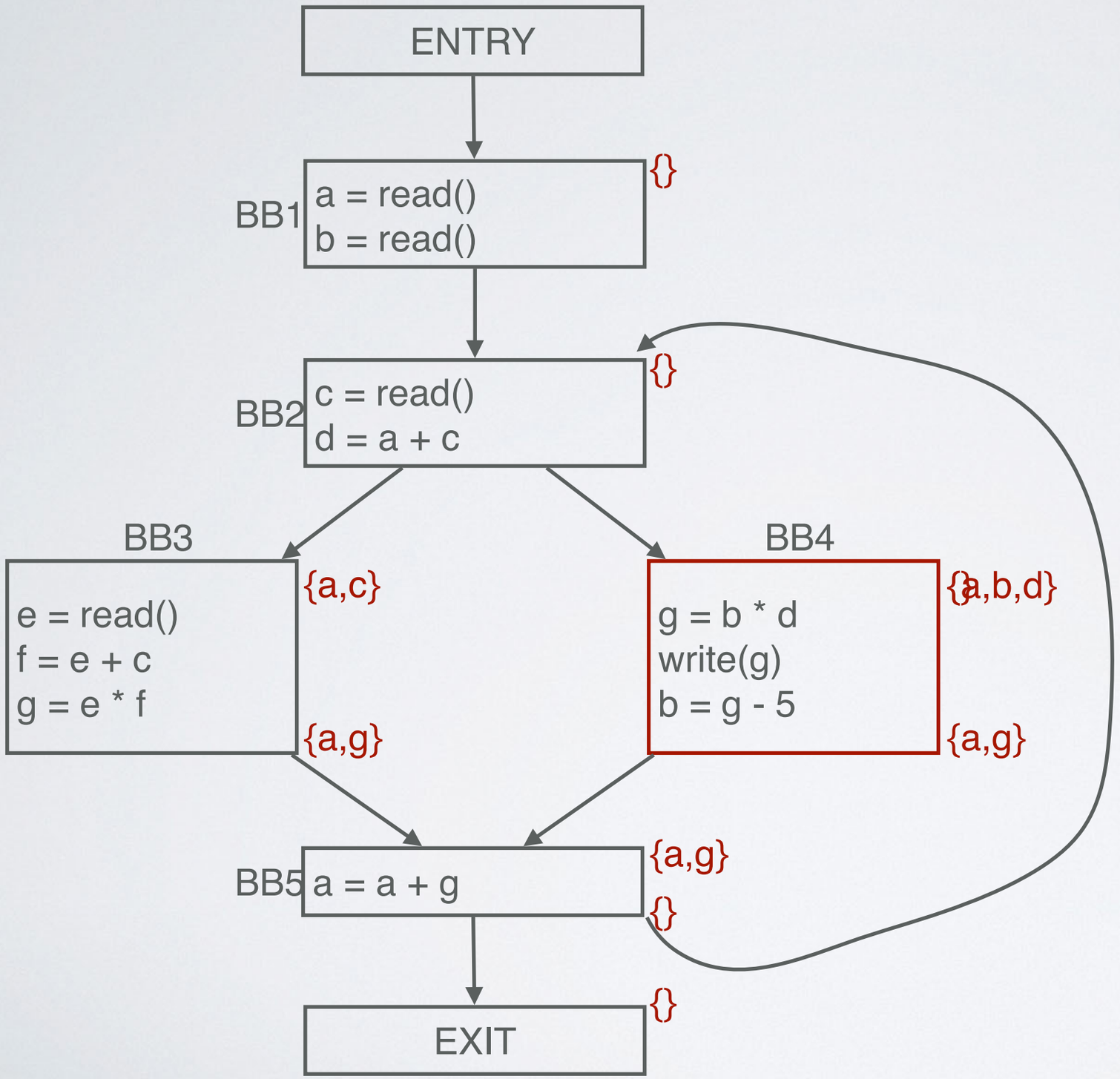
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{OUT}[B] = \bigwedge_{S\text{是}B\text{后继}} \mathrm{IN}[S]$$

$$\mathrm{IN}[B] = f_B(\mathrm{OUT}[B])$$

- 重复该过程直到没有基本块需要更新

活跃变量分析：循环（4）



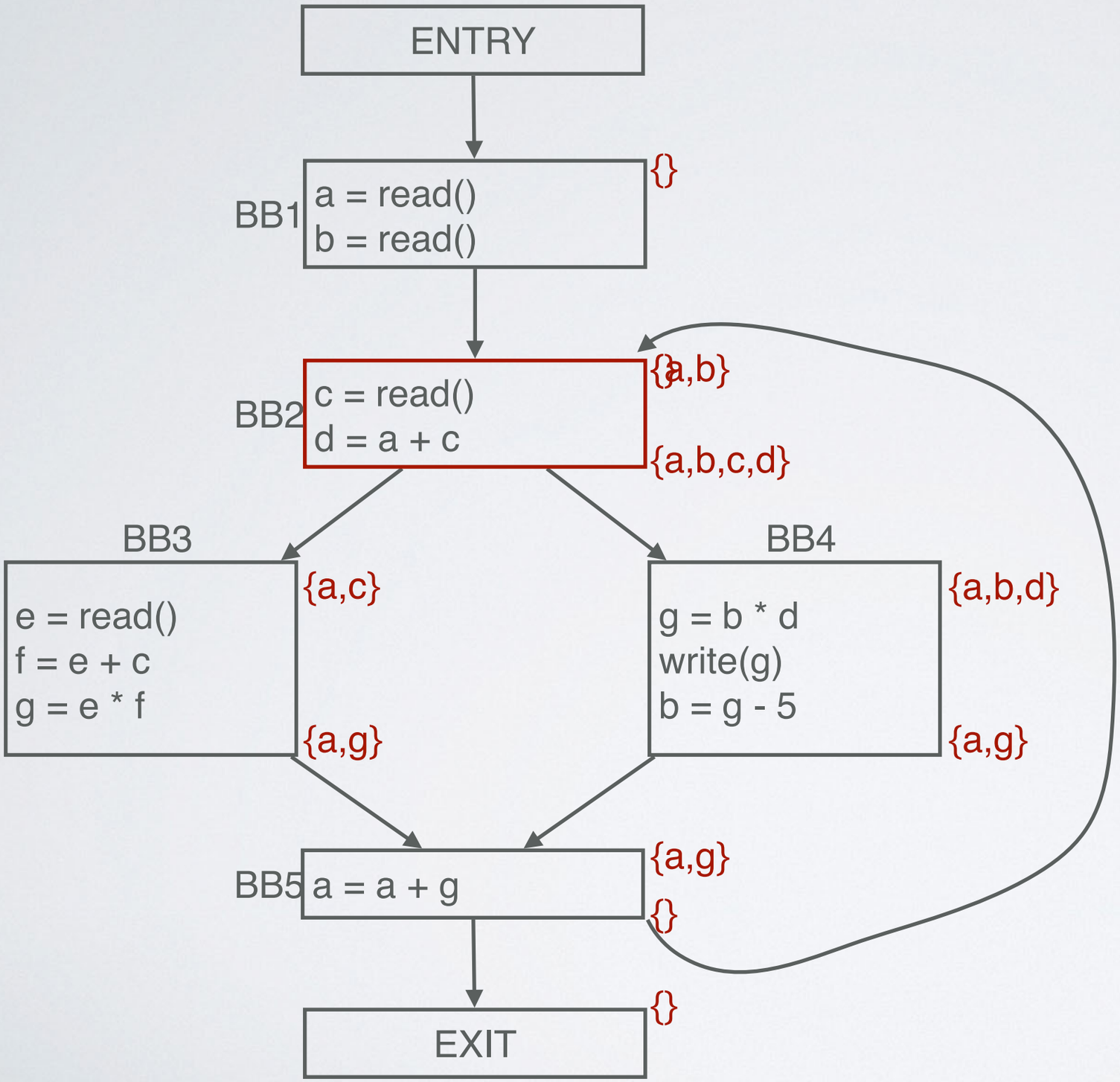
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{OUT}[B] = \bigwedge_{S\text{是}B\text{后继}} \mathrm{IN}[S]$$

$$\mathrm{IN}[B] = f_B(\mathrm{OUT}[B])$$

- 重复该过程直到没有基本块需要更新

活跃变量分析：循环（5）



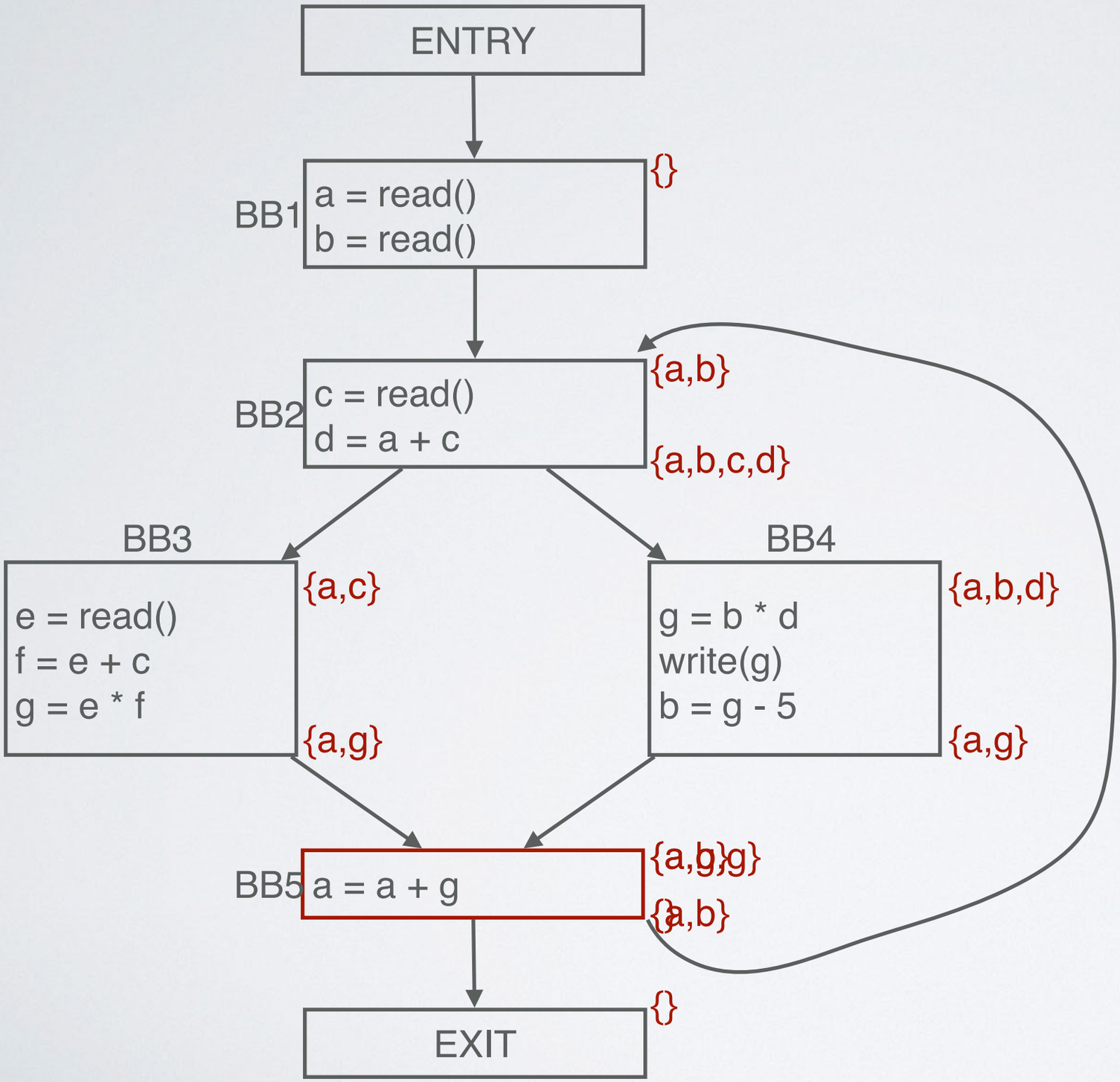
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{OUT}[B] = \bigwedge_{S\text{是}B\text{后继}} \mathrm{IN}[S]$$

$$\mathrm{IN}[B] = f_B(\mathrm{OUT}[B])$$

- 重复该过程直到没有基本块需要更新

活跃变量分析：循环（6）



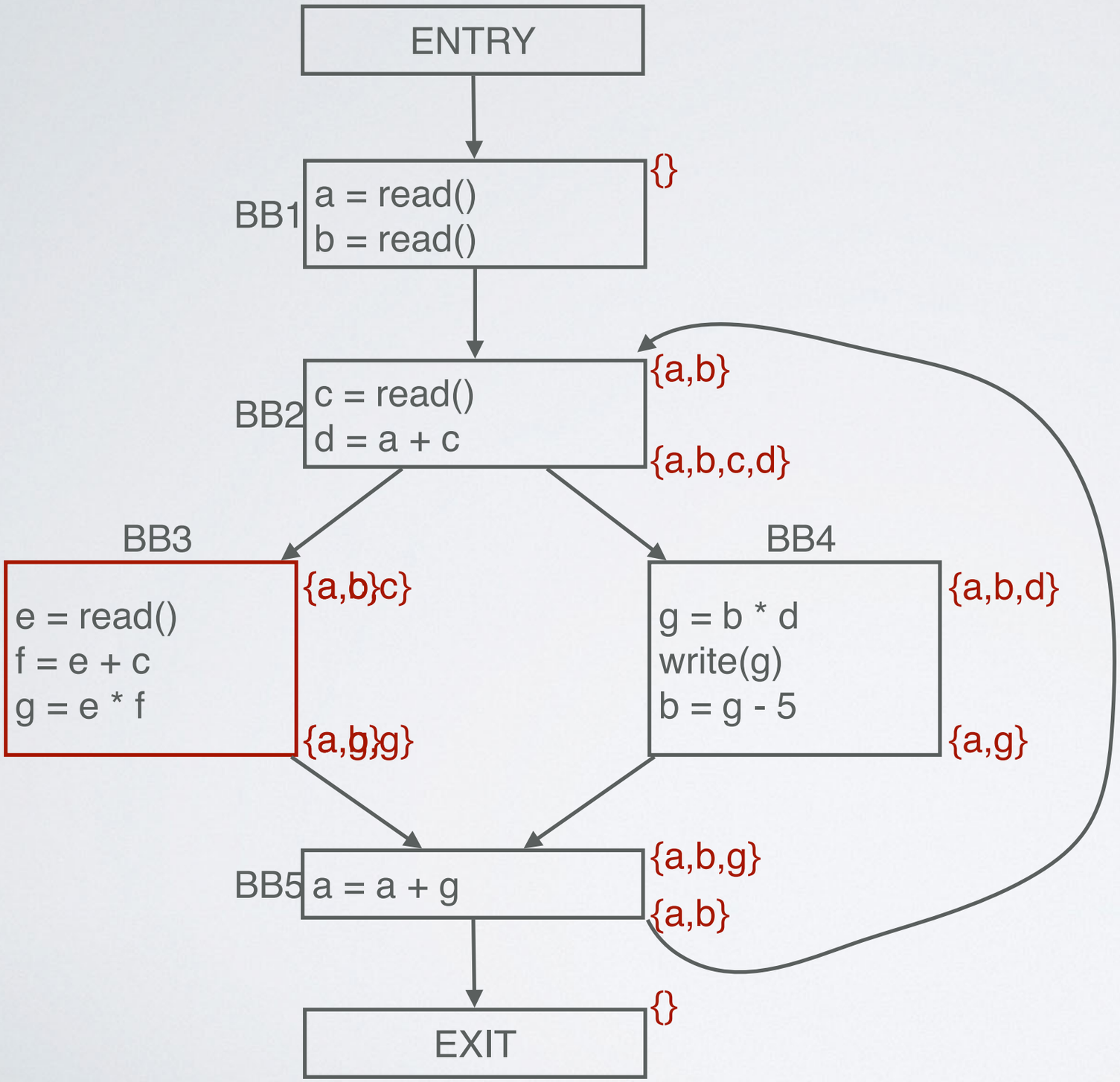
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{OUT}[B] = \bigwedge_{S\text{是}B\text{后继}} \mathrm{IN}[S]$$

$$\mathrm{IN}[B] = f_B(\mathrm{OUT}[B])$$

- 重复该过程直到没有基本块需要更新

活跃变量分析：循环（7）



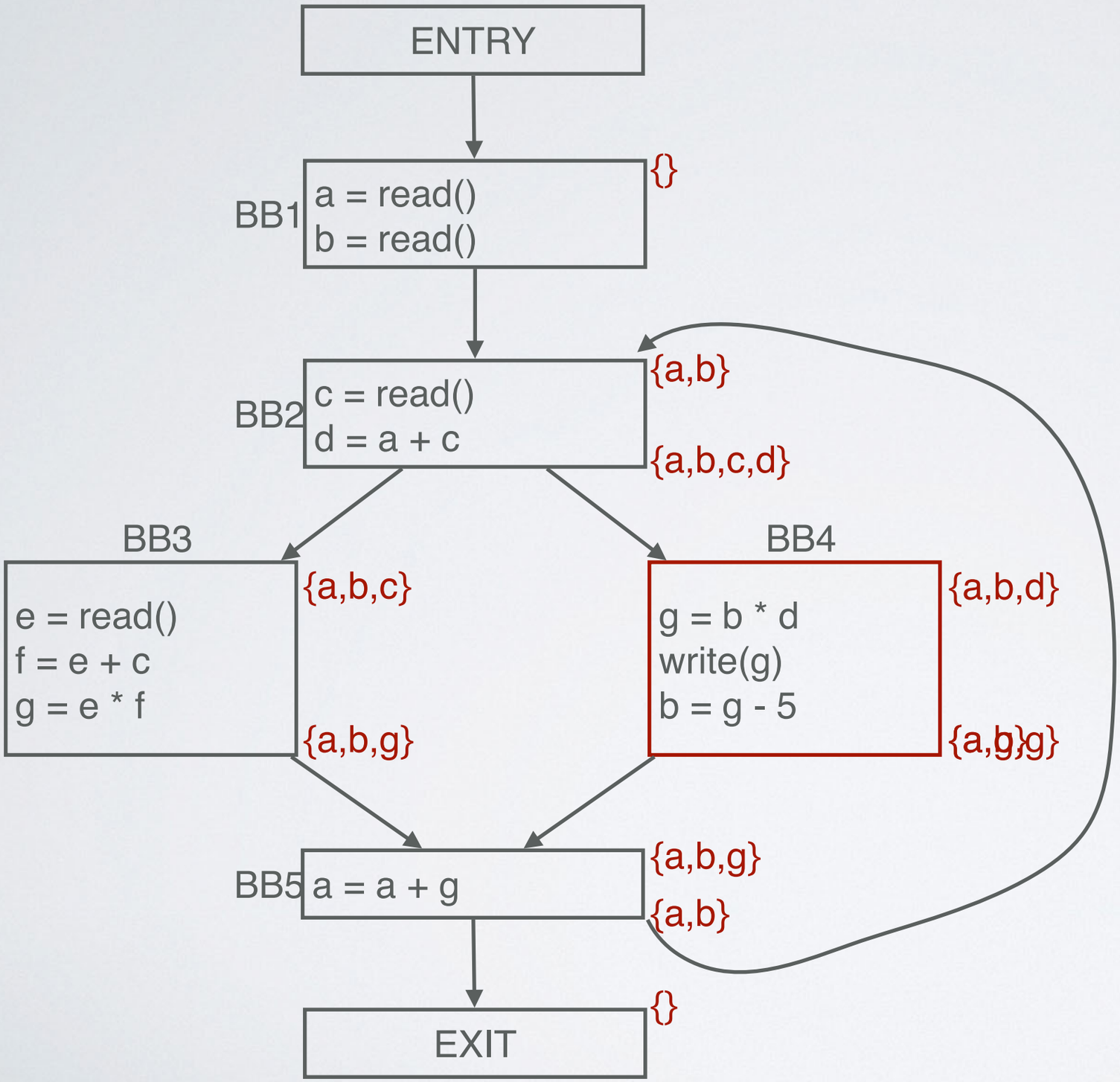
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{OUT}[B] = \bigwedge_{S\text{是}B\text{后继}} \mathrm{IN}[S]$$

$$\mathrm{IN}[B] = f_B(\mathrm{OUT}[B])$$

- 重复该过程直到没有基本块需要更新

活跃变量分析：循环（8）



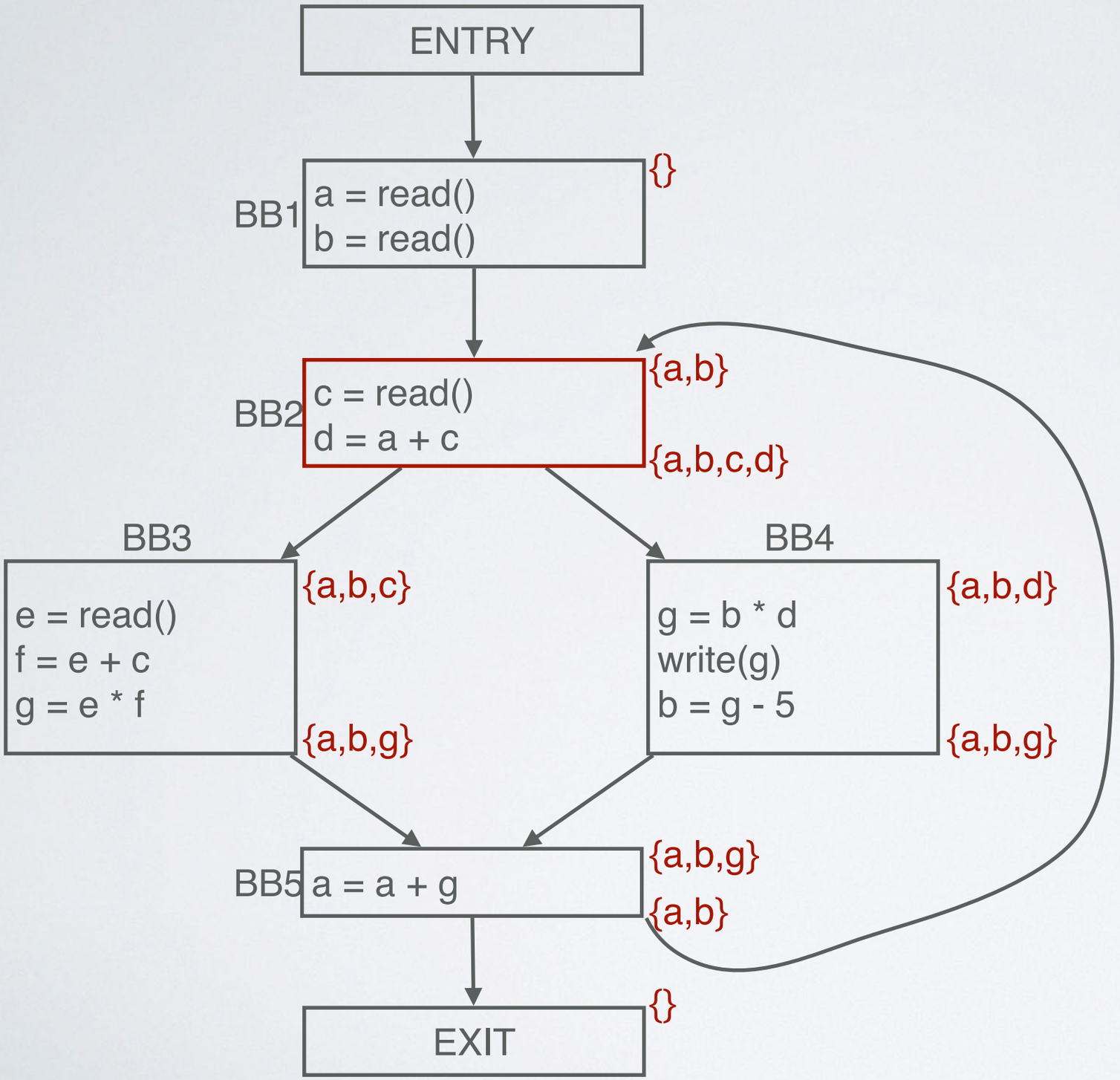
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{OUT}[B] = \bigwedge_{S\text{是}B\text{后继}} \mathrm{IN}[S]$$

$$\mathrm{IN}[B] = f_B(\mathrm{OUT}[B])$$

- 重复该过程直到没有基本块需要更新

活跃变量分析：循环（9）



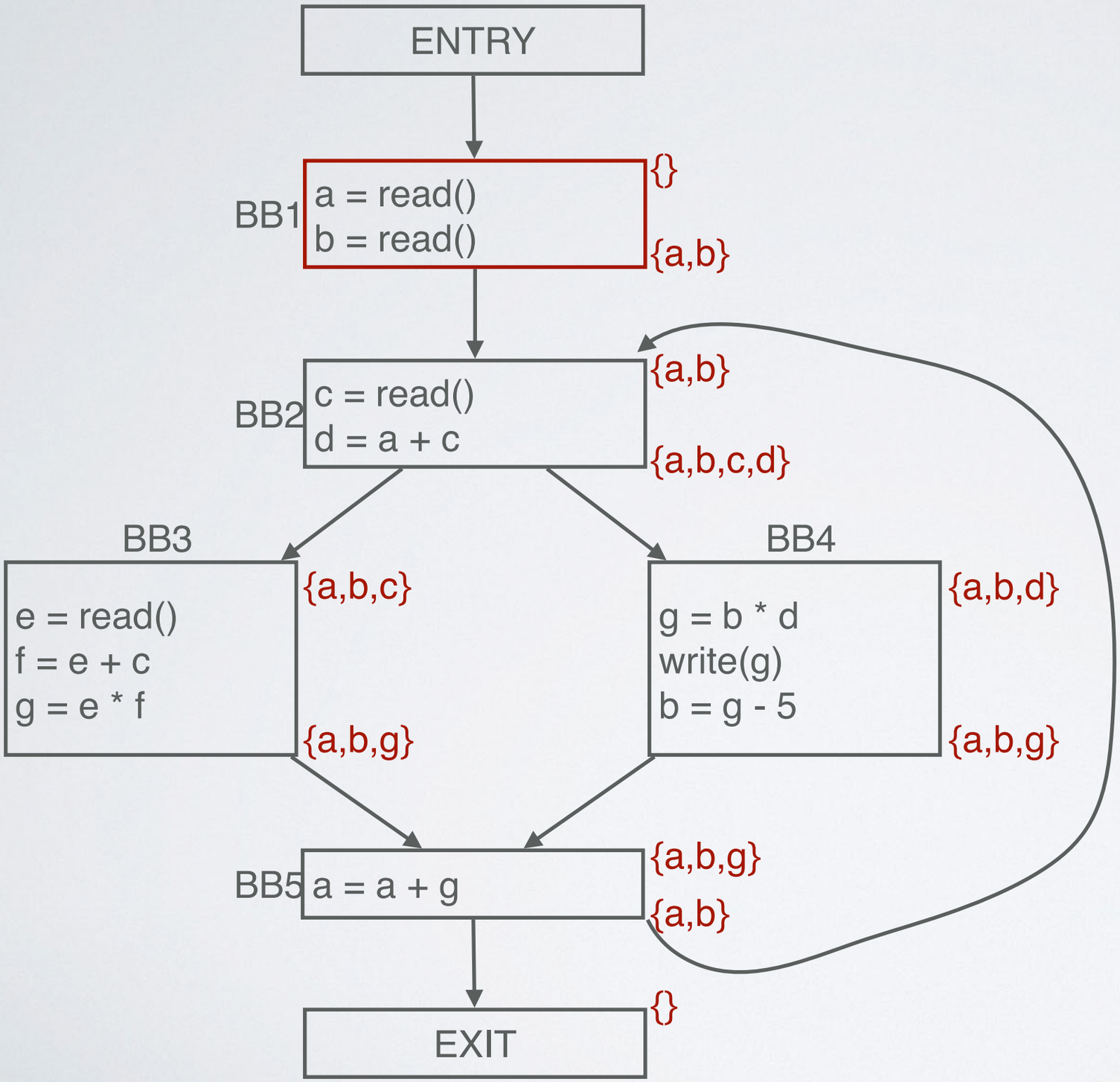
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{OUT}[B] = \bigwedge_{S\text{是}B\text{后继}} \mathrm{IN}[S]$$

$$\mathrm{IN}[B] = f_B(\mathrm{OUT}[B])$$

- 重复该过程直到没有基本块需要更新

活跃变量分析：循环（10）



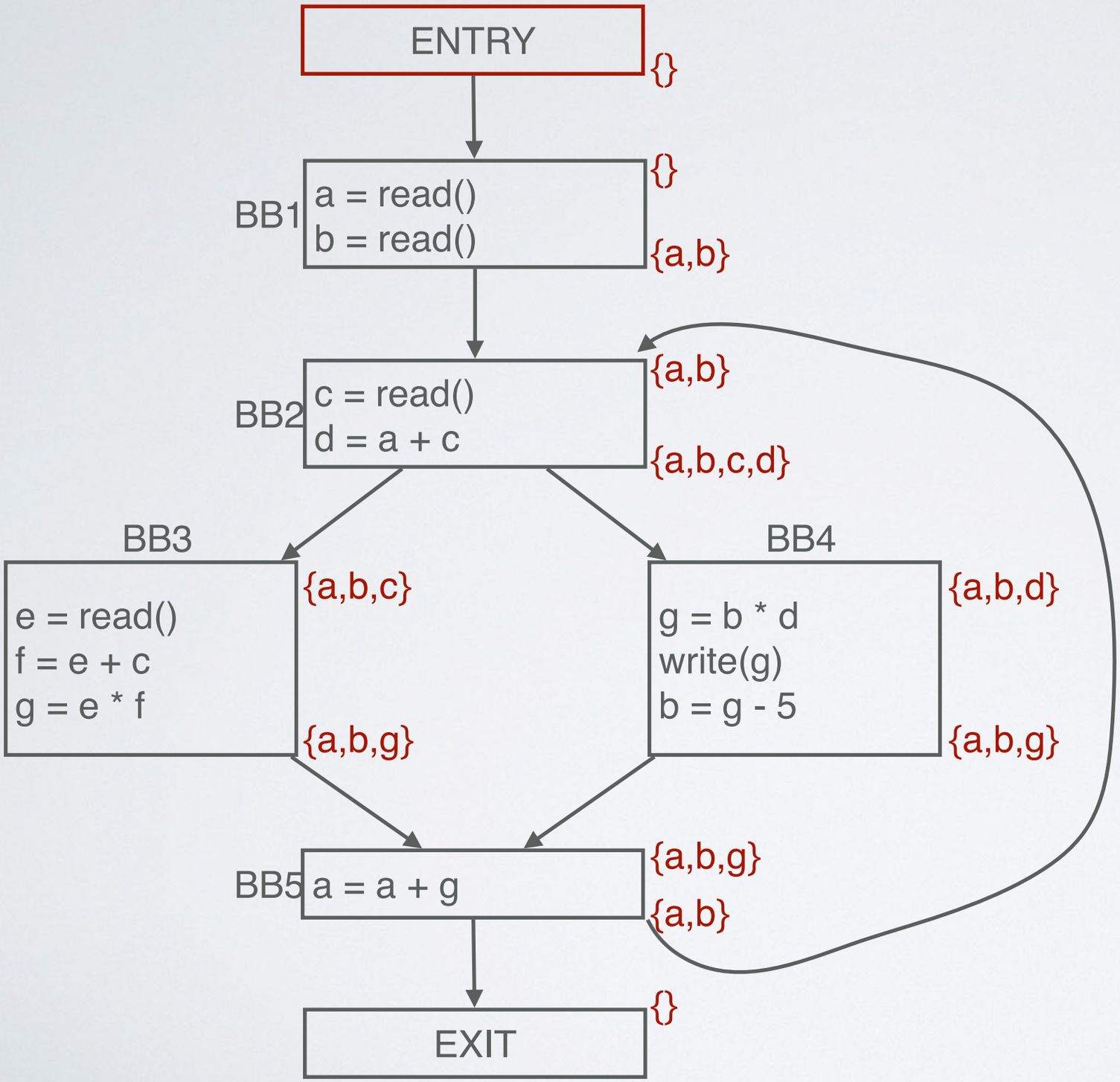
- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{OUT}[B] = \bigwedge_{S\text{是}B\text{后继}} \mathrm{IN}[S]$$

$$\mathrm{IN}[B] = f_B(\mathrm{OUT}[B])$$

- 重复该过程直到没有基本块需要更新

活跃变量分析：循环（11）



- 通过迭代法解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

$$\mathrm{OUT}[B] = \bigwedge_{S\text{是}B\text{后继}} \mathrm{IN}[S]$$

$$\mathrm{IN}[B] = f_B(\mathrm{OUT}[B])$$

- 重复该过程直到没有基本块需要更新

小结：后向分析模式

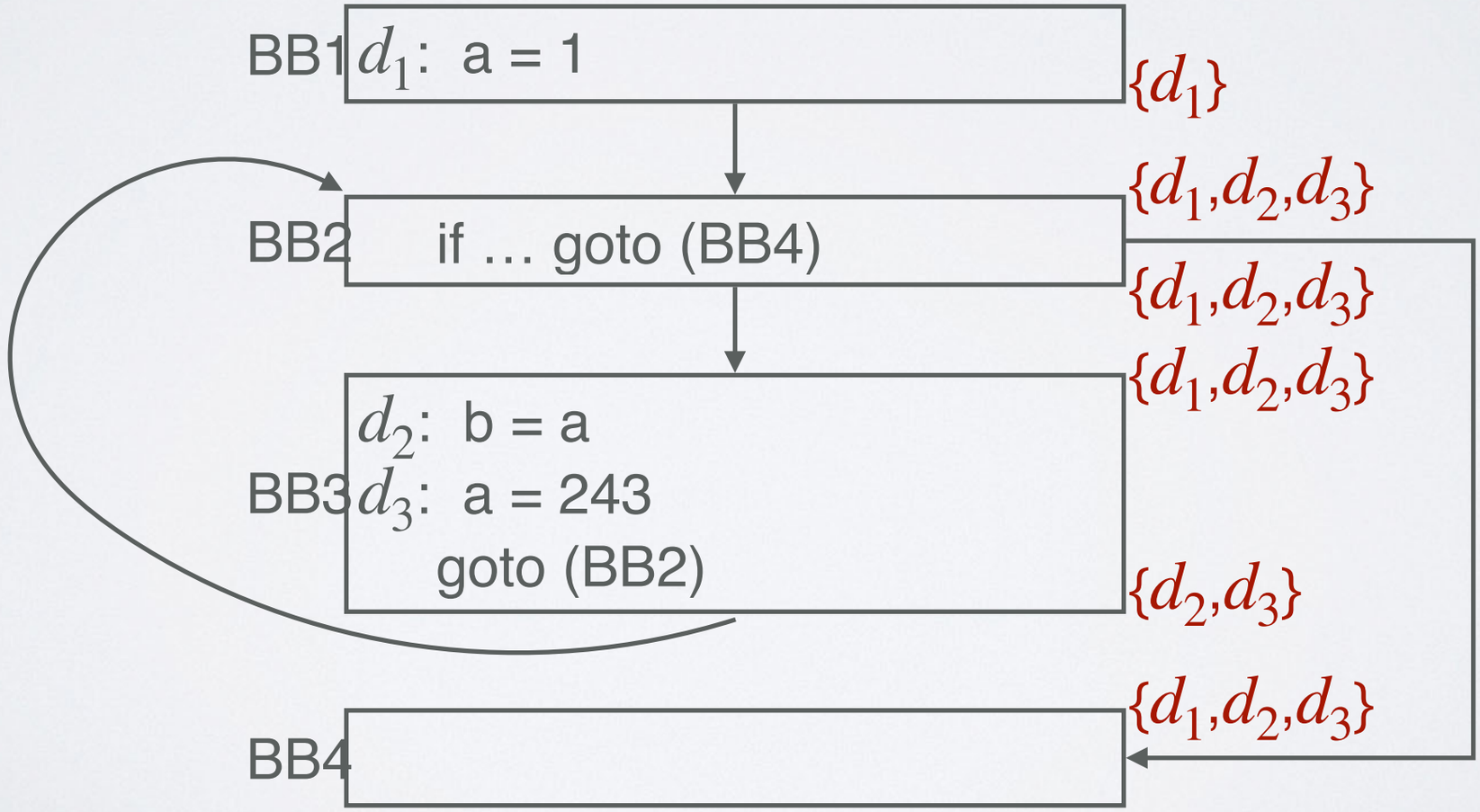
- 数据流分析的域 V，交汇运算 $\wedge : V \times V \to V$，顶值 $\top \in V$
- 每个基本块 B 的传递函数 $f_B : V \to V$ （从出口到入口）
- 边界条件：$\mathrm{IN}[\mathrm{EXIT}] = v_{\mathrm{EXIT}}$
- 初始值：$\mathrm{IN}[B] = \top \quad (B \neq \mathrm{EXIT})$
- 方程组：对任意 $B \neq \mathrm{EXIT}$，有

$$\mathrm{OUT}[B] = \bigwedge_{S\text{是}B\text{的后继}} \mathrm{IN}[S]$$

$$\mathrm{IN}[B] = f_B(\mathrm{OUT}[B])$$

到达定值分析（1）

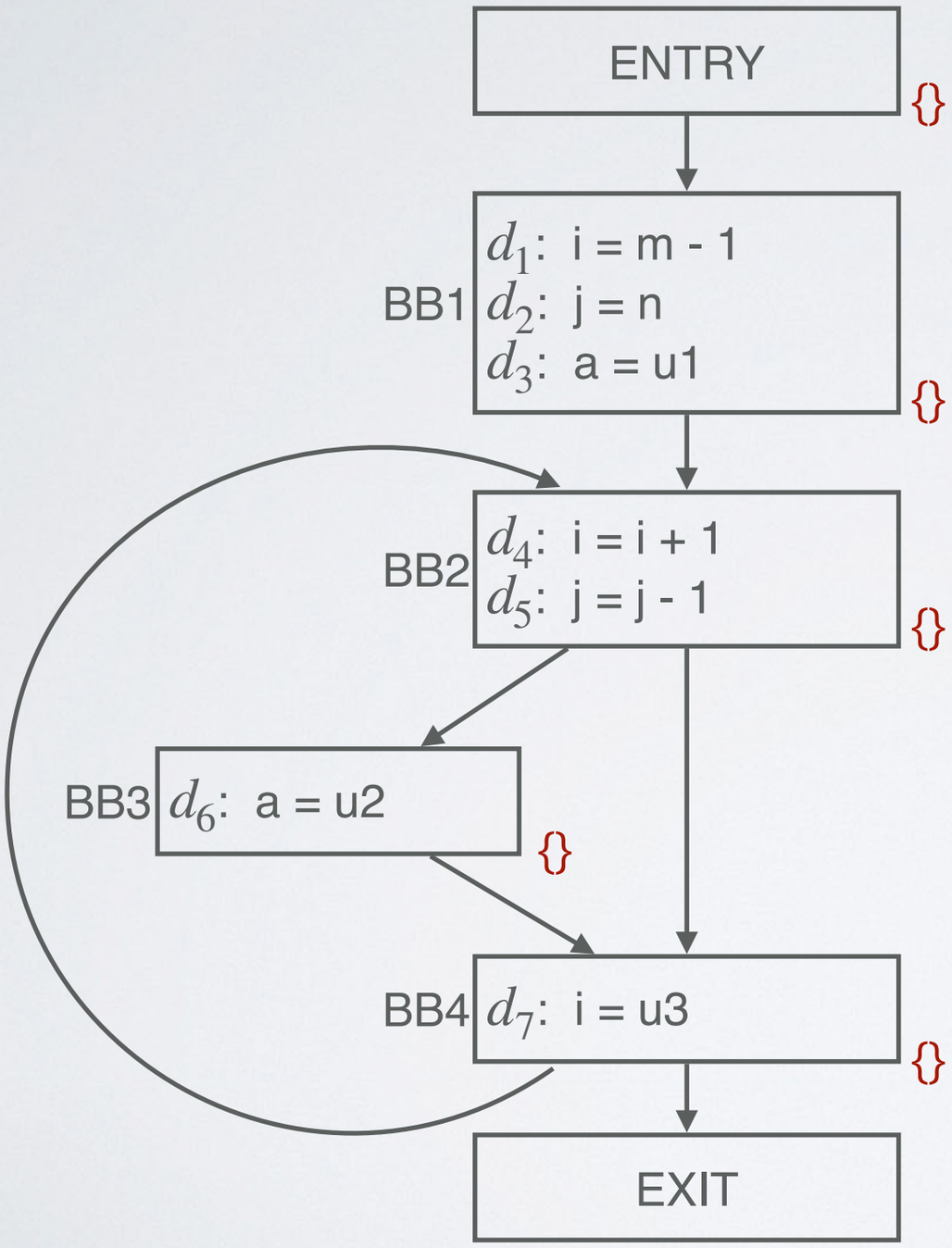
- 到达定值（reaching definition）
  - 可能沿着某条路径到达一个程序点的定值（形如 $x = \ldots$ 的语句）
  - 前向分析，值域为定值的集合
  - 用途：常量折叠



到达定值分析（2）

- 对每条语句 s，定义如下集合：
  - gen_s：语句 s 生成的定值的集合
  - $kill_s$：语句 s “杀死”的定值的集合
    - 若 s 对 x 赋值，则“杀死”了所有其它的对 x 的定值
  - 传递函数 $f_s(I) = (I - kill_s) \cup gen_s$
- 基本块 B 的传递函数 $f_B = f_{s_n} \circ \ldots \circ f_{s_2} \circ f_{s_1}$
  - 可以表示为 $f_B(I) = (I - kill_B) \cup gen_B$
  - 类似于活跃变量分析中的 def 和 use
- 交汇运算：$I_1 \wedge I_2 = I_1 \cup I_2$，从任意前驱可达则认为是可达的
- 顶值（top）：空集

到达定值分析（3）



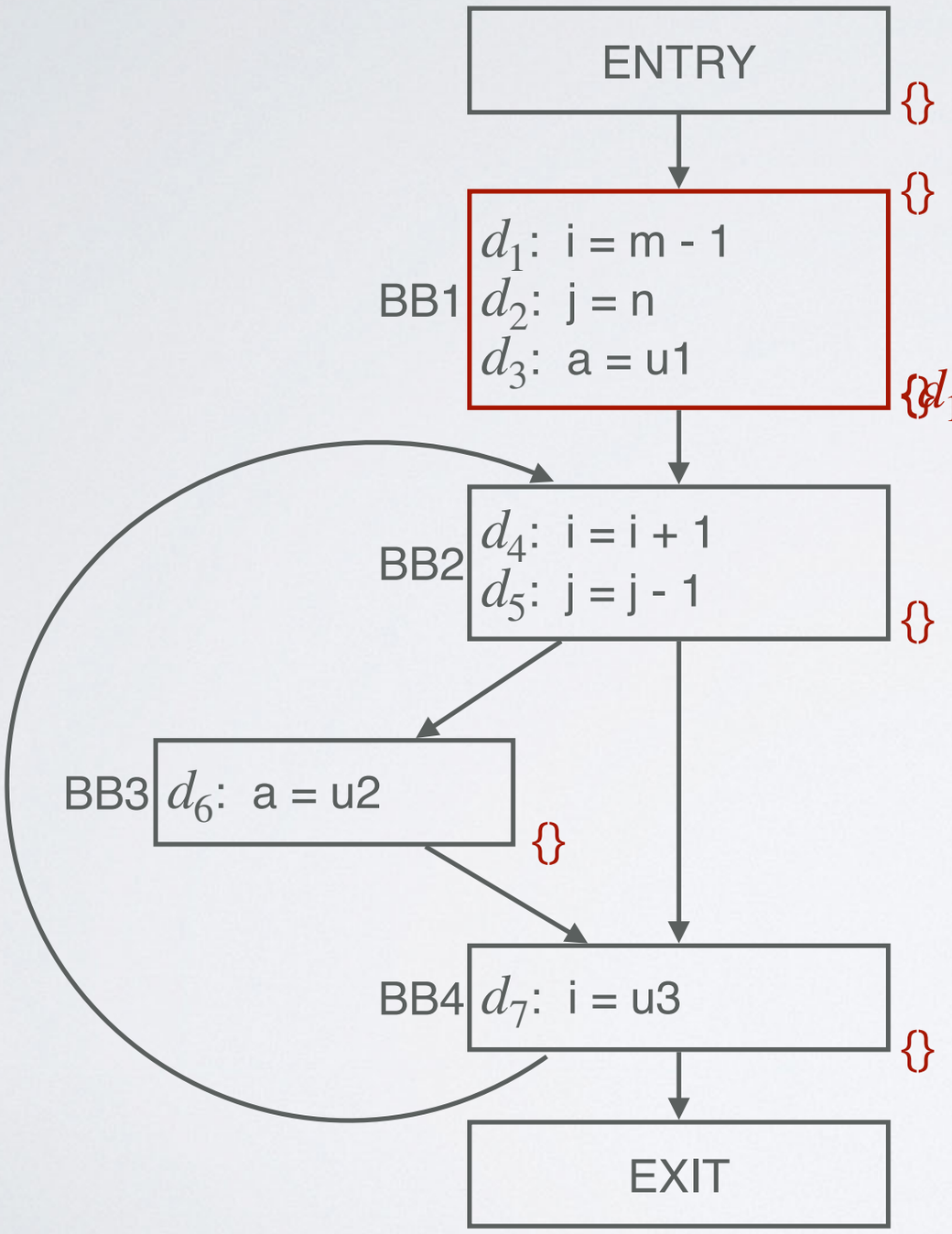
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

$gen_{\text{BB2}} = \{d_4, d_5\}$
$kill_{\text{BB2}} = \{d_1, d_2, d_7\}$

$gen_{\text{BB3}} = \{d_6\}$
$kill_{\text{BB3}} = \{d_3\}$

$gen_{\text{BB4}} = \{d_7\}$
$kill_{\text{BB4}} = \{d_1, d_4\}$

到达定值分析（4）



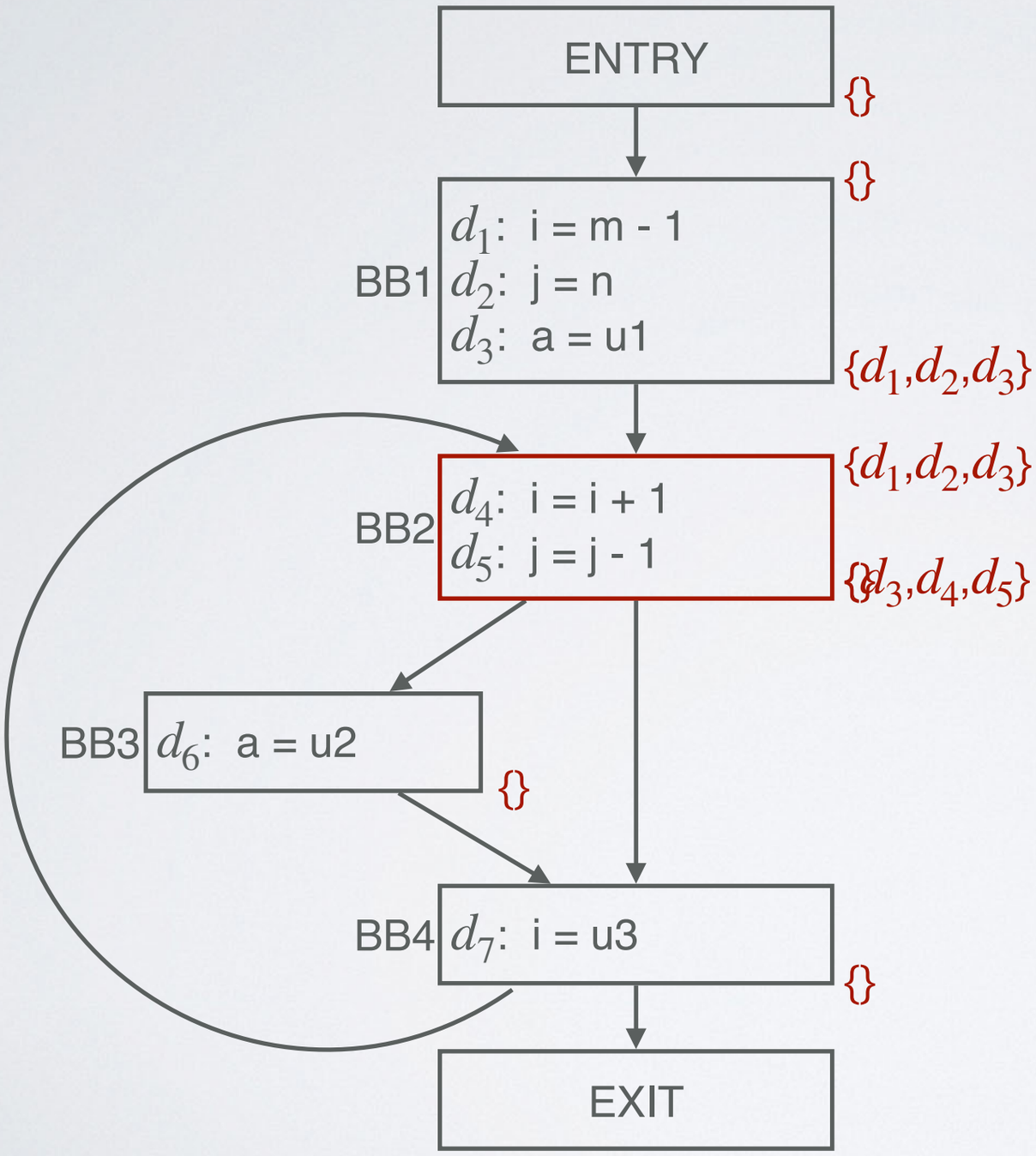
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

$gen_{\text{BB2}} = \{d_4, d_5\}$
$kill_{\text{BB2}} = \{d_1, d_2, d_7\}$

$gen_{\text{BB3}} = \{d_6\}$
$kill_{\text{BB3}} = \{d_3\}$

$gen_{\text{BB4}} = \{d_7\}$
$kill_{\text{BB4}} = \{d_1, d_4\}$

到达定值分析（5）



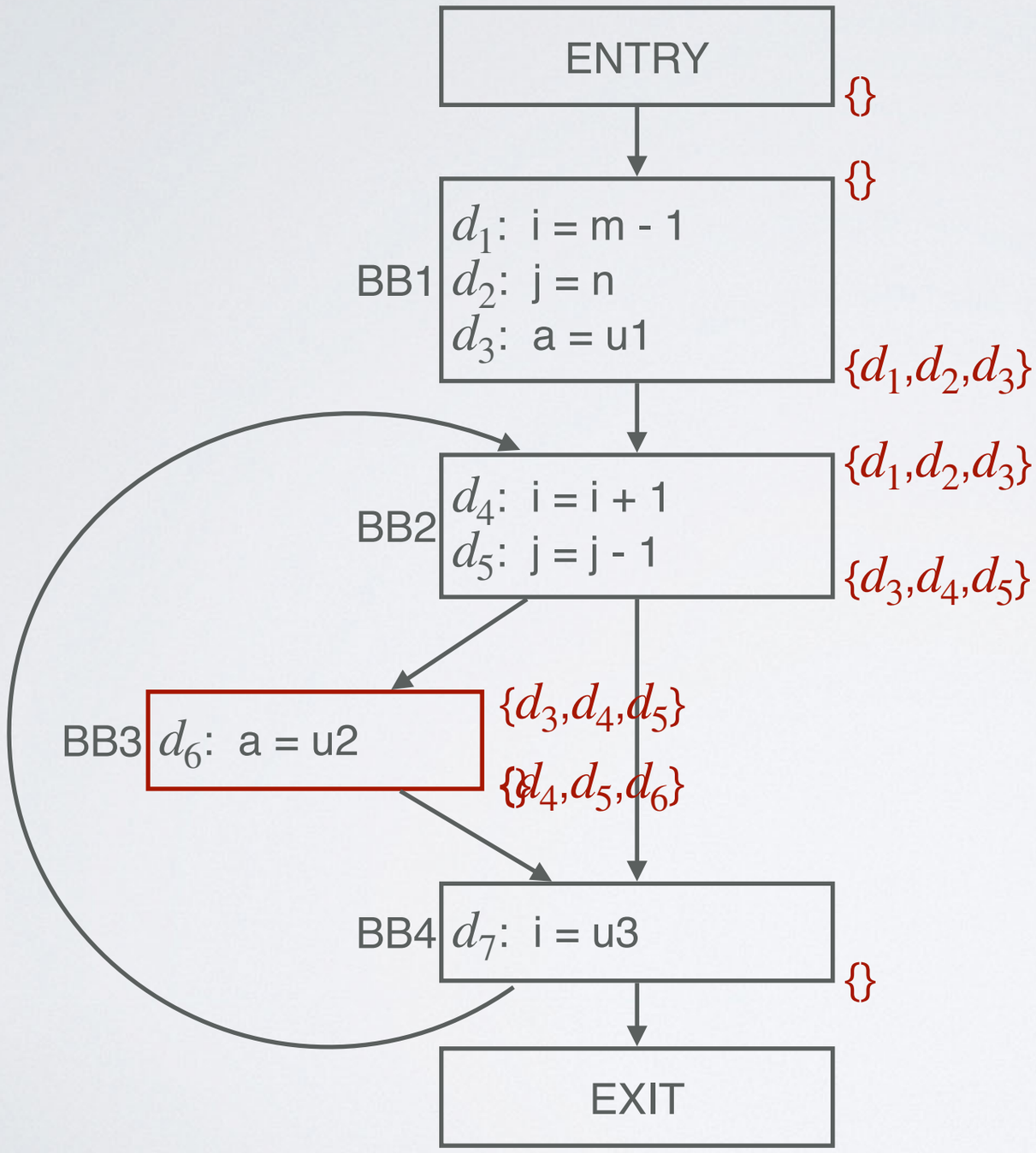
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

$gen_{\text{BB2}} = \{d_4, d_5\}$
$kill_{\text{BB2}} = \{d_1, d_2, d_7\}$

$gen_{\text{BB3}} = \{d_6\}$
$kill_{\text{BB3}} = \{d_3\}$

$gen_{\text{BB4}} = \{d_7\}$
$kill_{\text{BB4}} = \{d_1, d_4\}$

到达定值分析（6）



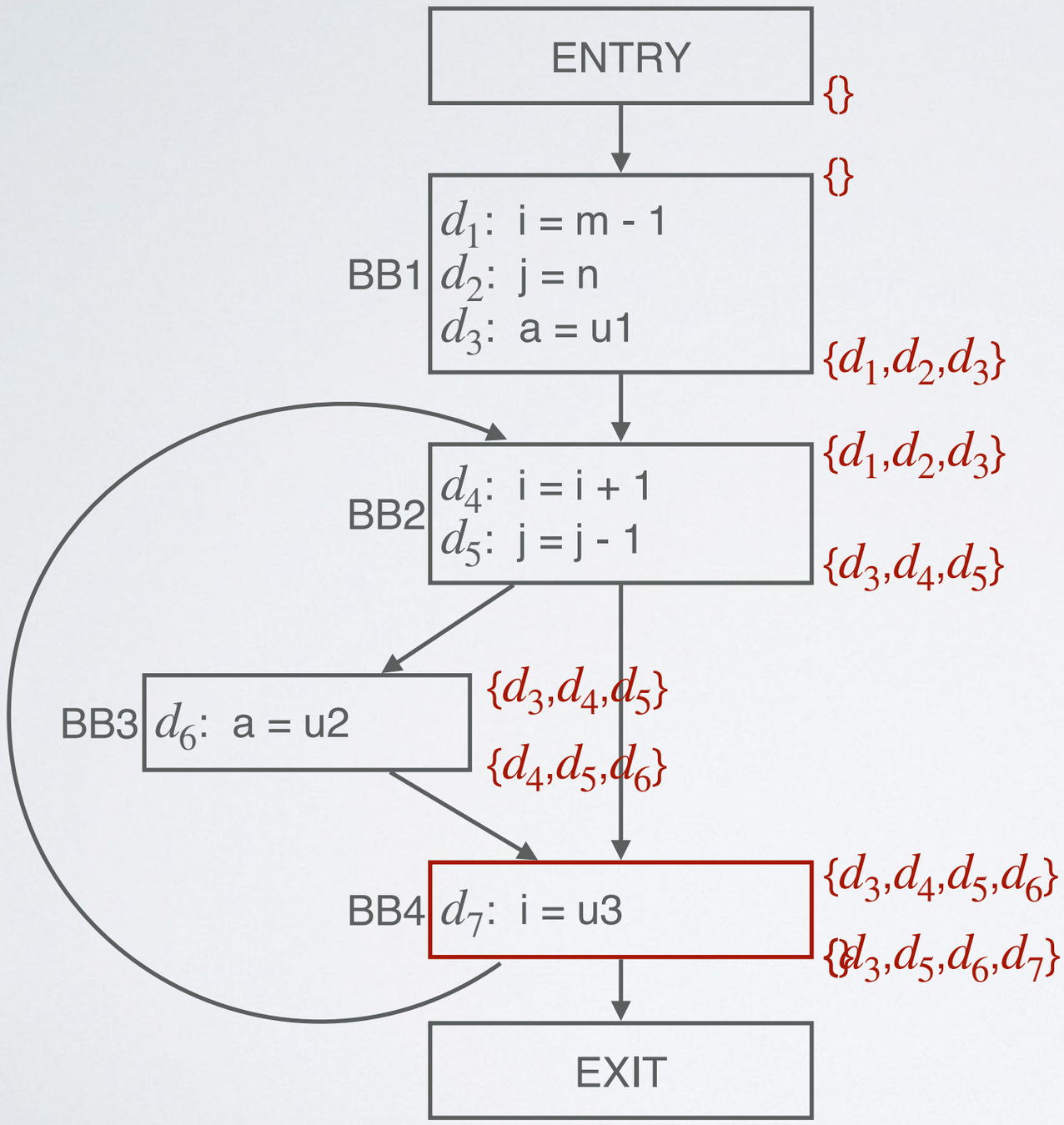
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

$gen_{\text{BB2}} = \{d_4, d_5\}$
$kill_{\text{BB2}} = \{d_1, d_2, d_7\}$

$gen_{\text{BB3}} = \{d_6\}$
$kill_{\text{BB3}} = \{d_3\}$

$gen_{\text{BB4}} = \{d_7\}$
$kill_{\text{BB4}} = \{d_1, d_4\}$

到达定值分析（7）



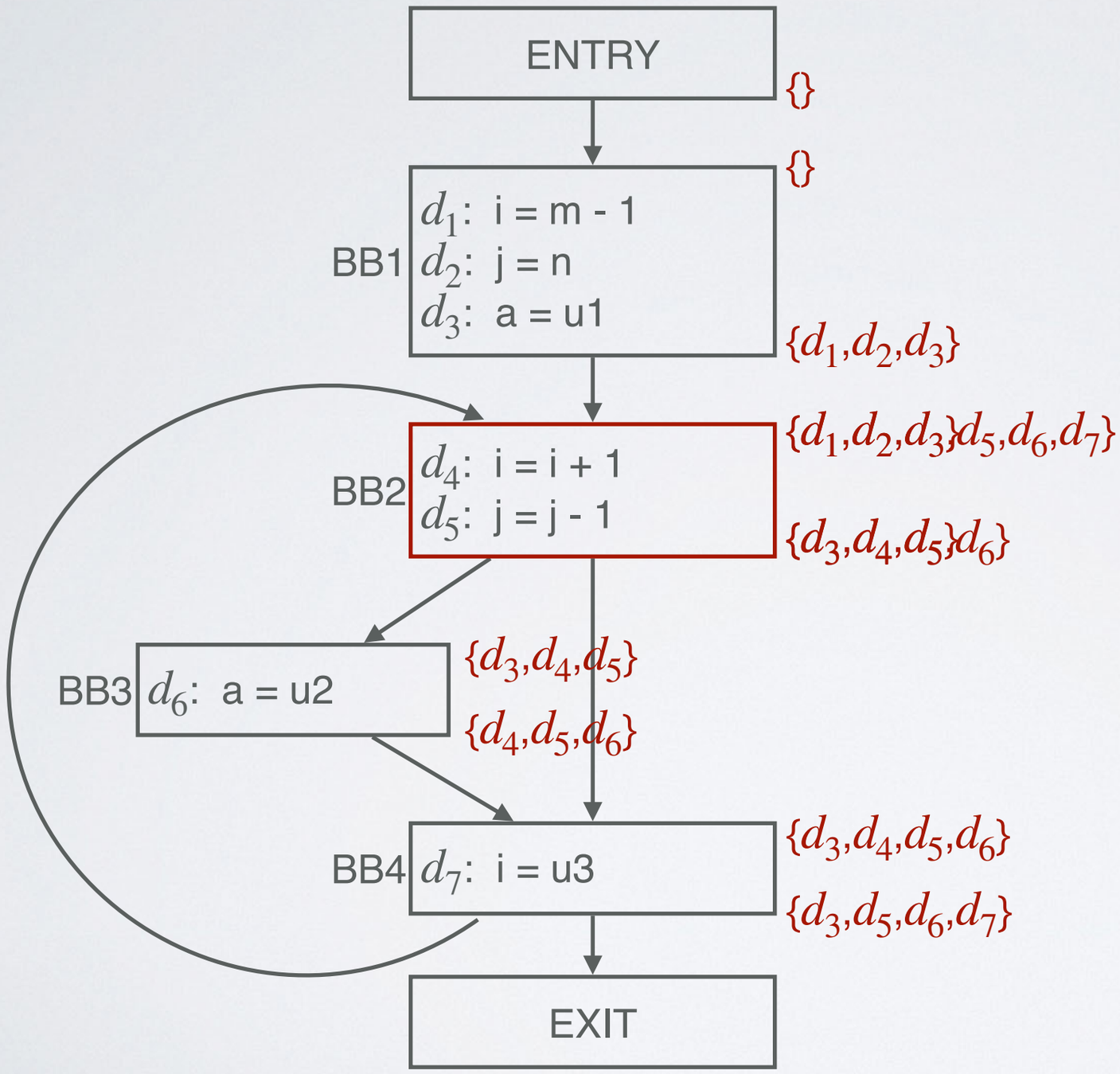
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

$gen_{\text{BB2}} = \{d_4, d_5\}$
$kill_{\text{BB2}} = \{d_1, d_2, d_7\}$

$gen_{\text{BB3}} = \{d_6\}$
$kill_{\text{BB3}} = \{d_3\}$

$gen_{\text{BB4}} = \{d_7\}$
$kill_{\text{BB4}} = \{d_1, d_4\}$

到达定值分析（8）



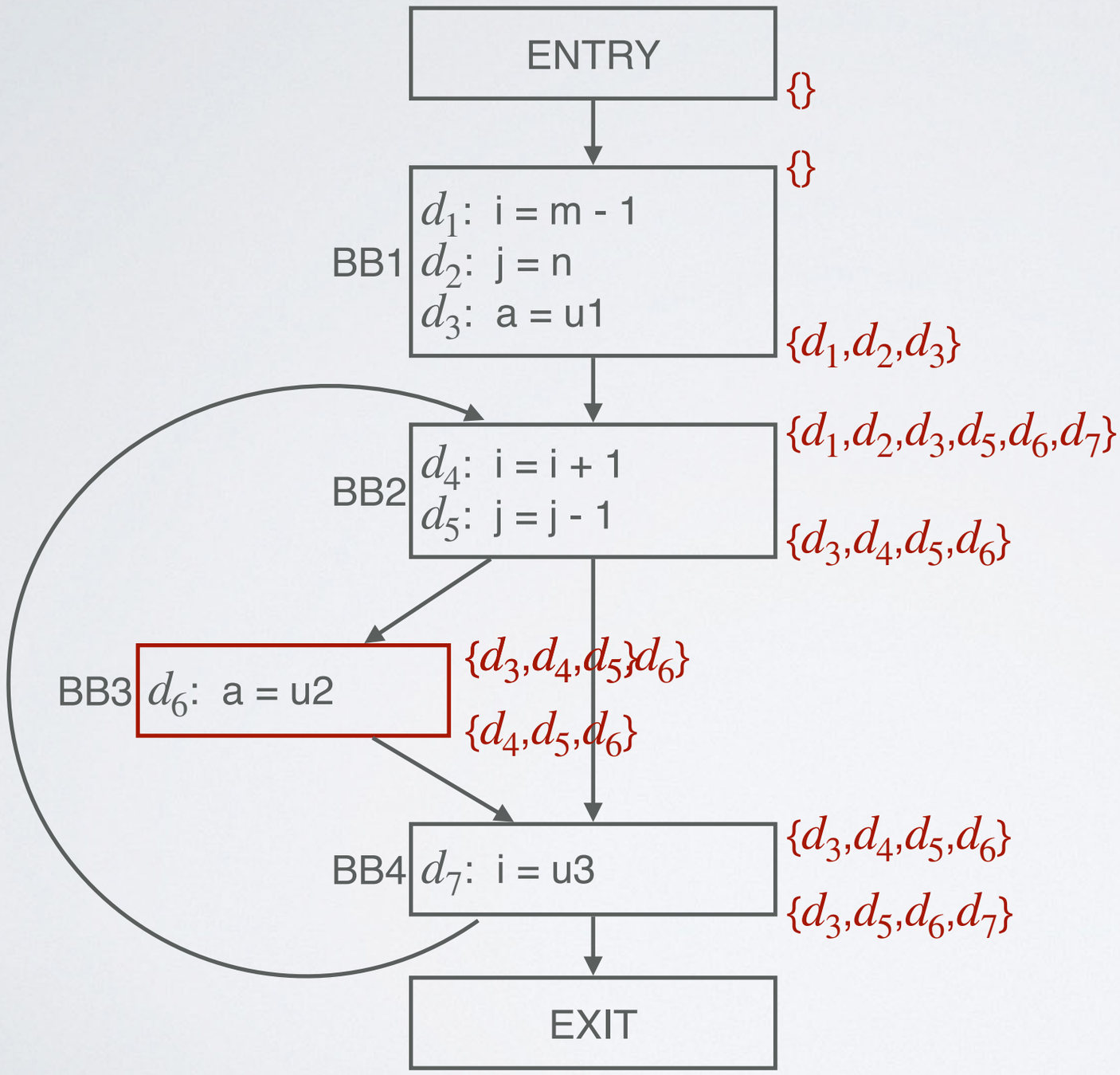
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

$gen_{\text{BB2}} = \{d_4, d_5\}$
$kill_{\text{BB2}} = \{d_1, d_2, d_7\}$

$gen_{\text{BB3}} = \{d_6\}$
$kill_{\text{BB3}} = \{d_3\}$

$gen_{\text{BB4}} = \{d_7\}$
$kill_{\text{BB4}} = \{d_1, d_4\}$

到达定值分析（9）



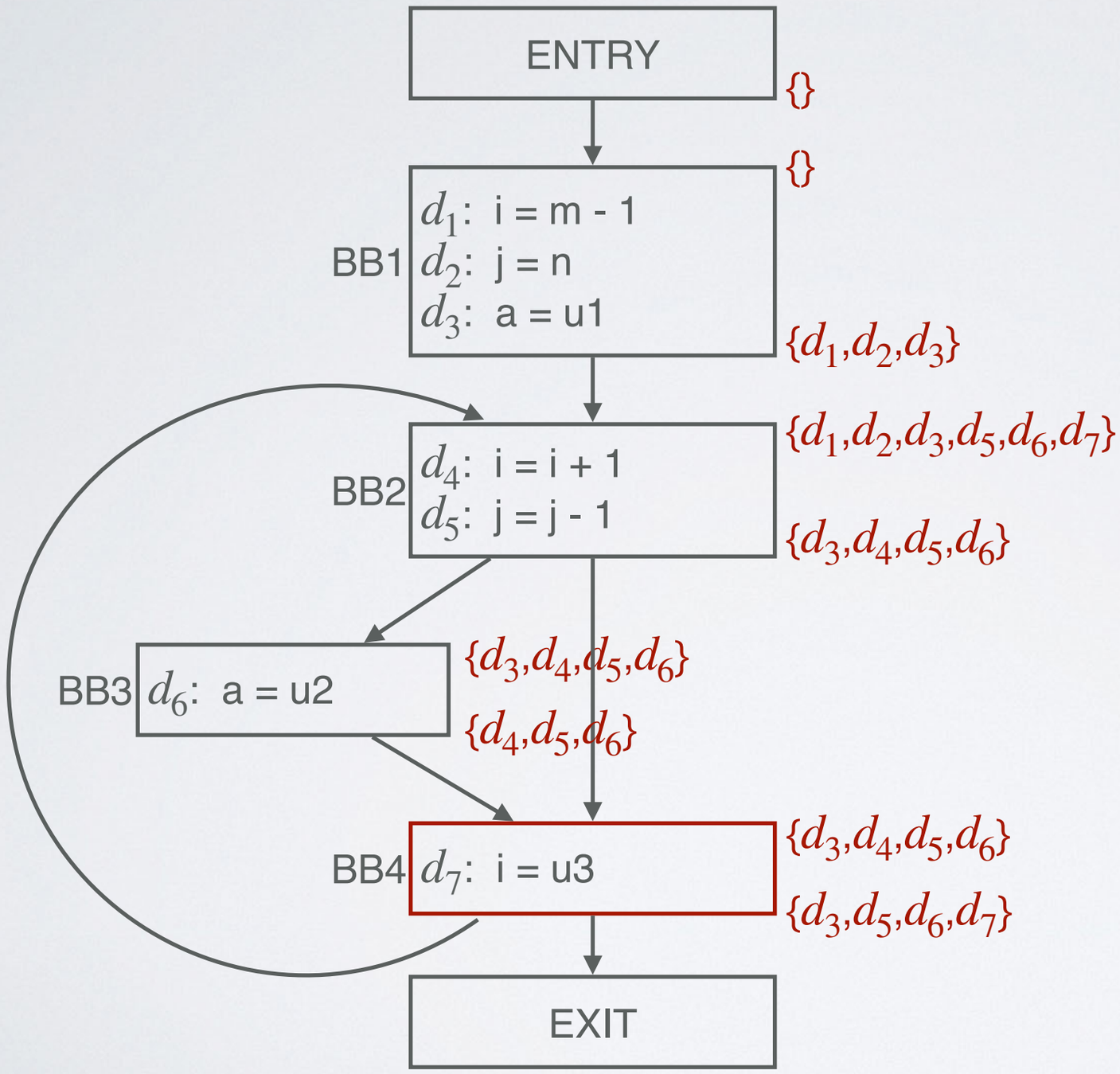
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

$gen_{\text{BB2}} = \{d_4, d_5\}$
$kill_{\text{BB2}} = \{d_1, d_2, d_7\}$

$gen_{\text{BB3}} = \{d_6\}$
$kill_{\text{BB3}} = \{d_3\}$

$gen_{\text{BB4}} = \{d_7\}$
$kill_{\text{BB4}} = \{d_1, d_4\}$

到达定值分析（10）



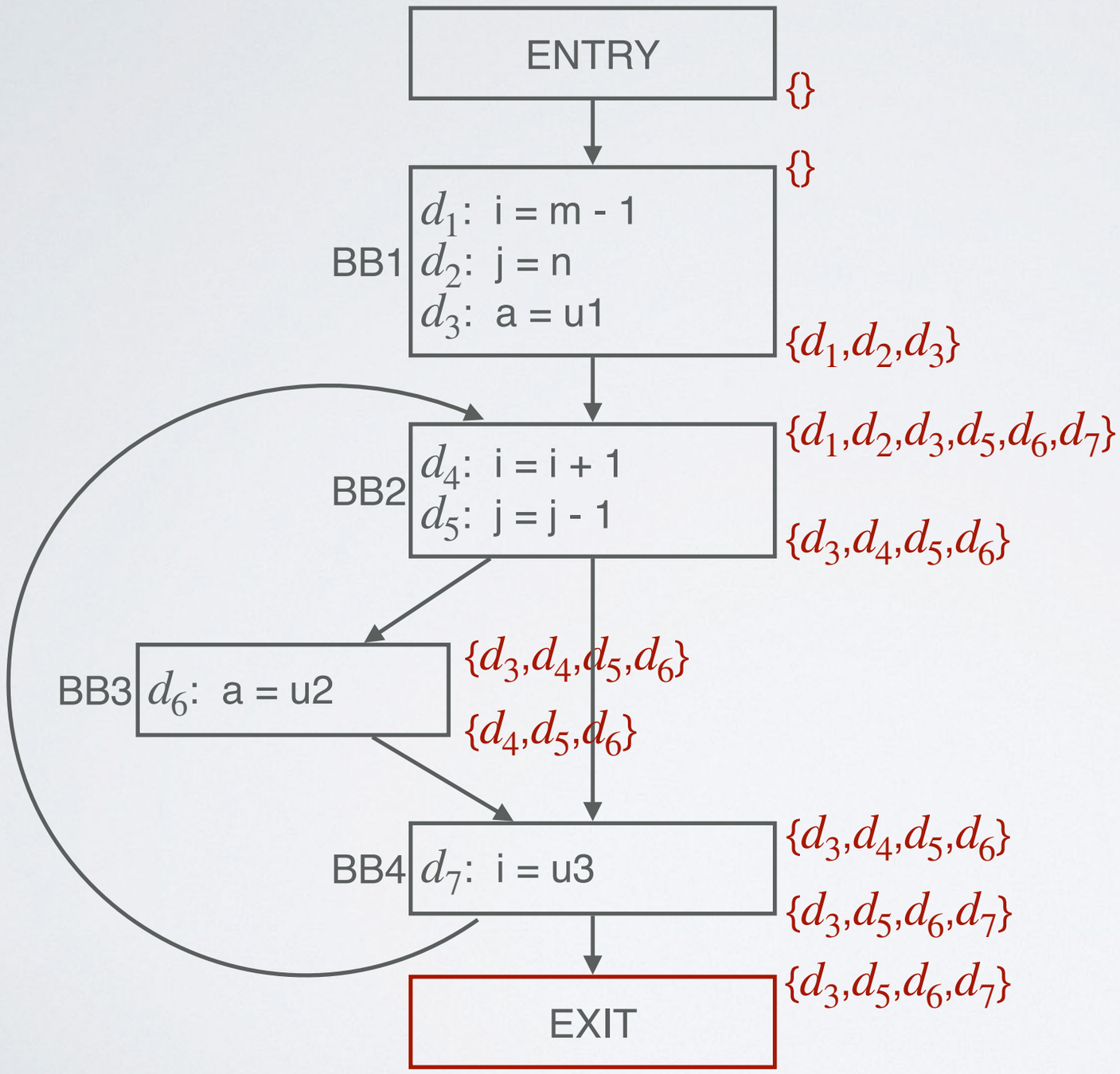
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

$gen_{\text{BB2}} = \{d_4, d_5\}$
$kill_{\text{BB2}} = \{d_1, d_2, d_7\}$

$gen_{\text{BB3}} = \{d_6\}$
$kill_{\text{BB3}} = \{d_3\}$

$gen_{\text{BB4}} = \{d_7\}$
$kill_{\text{BB4}} = \{d_1, d_4\}$

到达定值分析（11）



$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

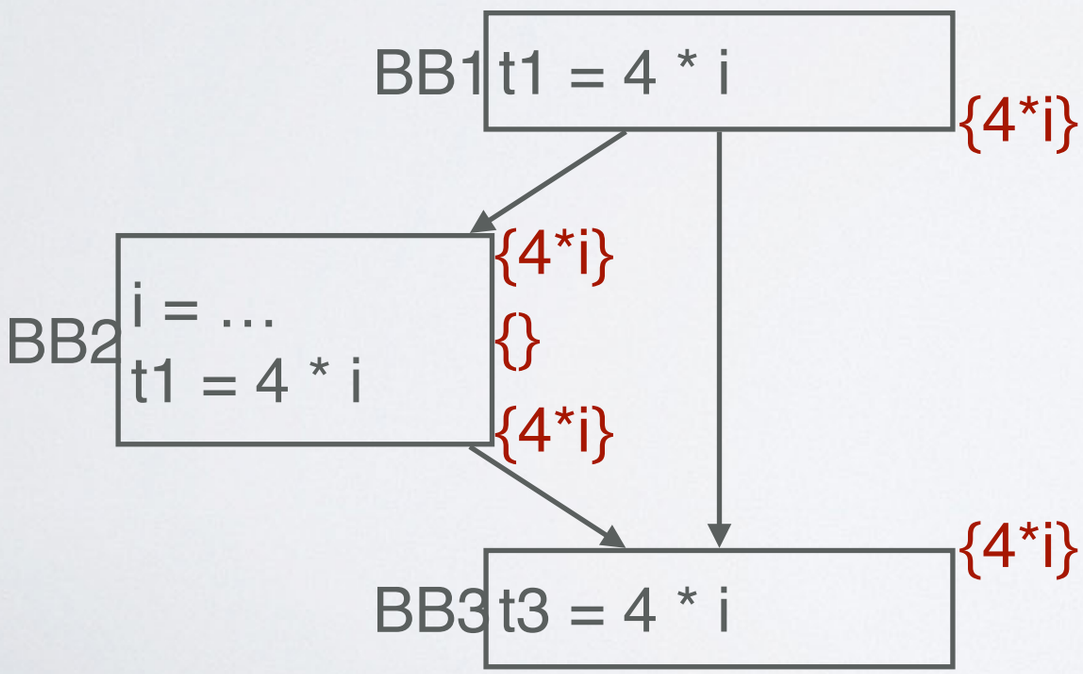
$gen_{\text{BB2}} = \{d_4, d_5\}$
$kill_{\text{BB2}} = \{d_1, d_2, d_7\}$

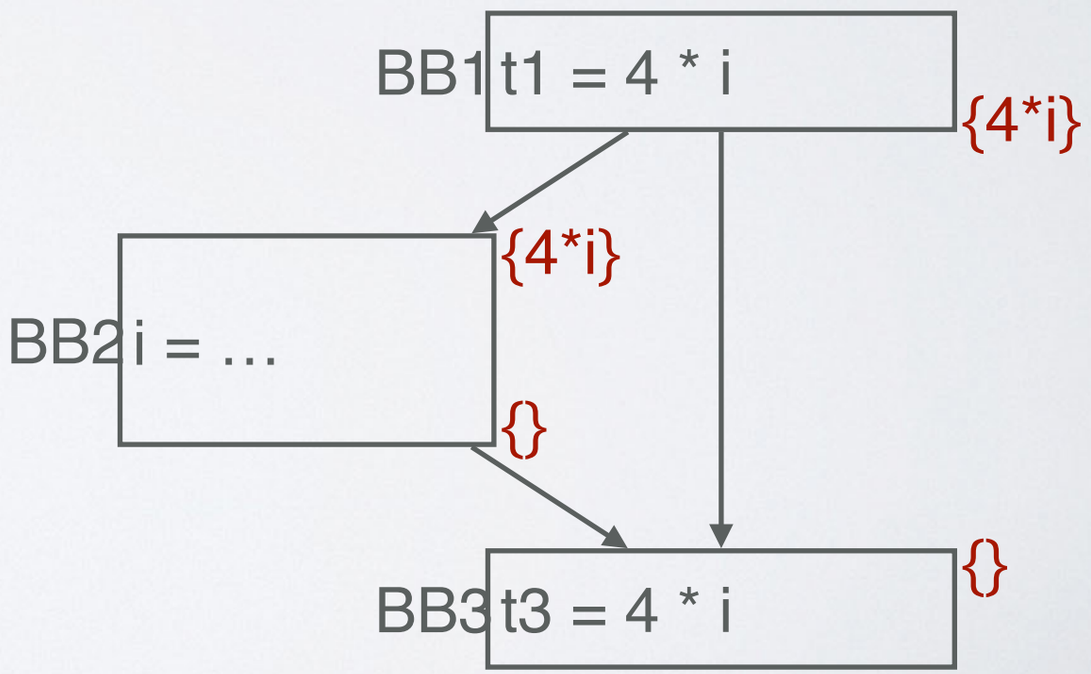
$gen_{\text{BB3}} = \{d_6\}$
$kill_{\text{BB3}} = \{d_3\}$

$gen_{\text{BB4}} = \{d_7\}$
$kill_{\text{BB4}} = \{d_1, d_4\}$

可用表达式分析（1）

- 可用表达式（available expression）
  - 到达一个程序点的每条路径都对表达式 E 求值，并且该表达式最近一次求值后其使用的变量没有被修改，则称 E 是一个可用表达式
  - 前向分析，值域为表达式的集合
  - 用途：公共子表达式消除

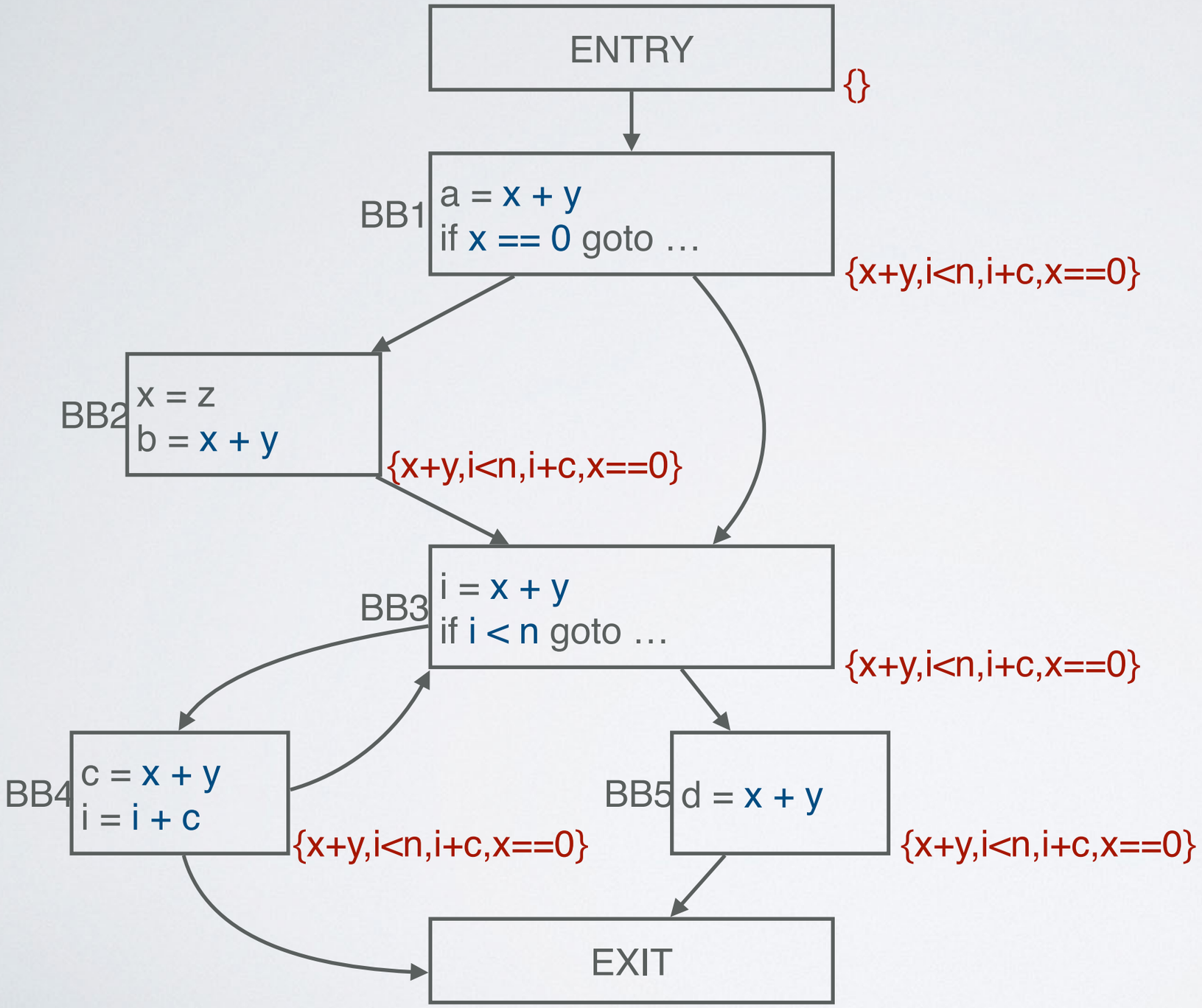




可用表达式分析（2）

- 对每条语句 s，定义如下集合：
  - e_gen_s：语句 s 生成的表达式的集合
  - e_kill_s：语句 s “杀死”的表达式的集合
    - 若 s 对 x 赋值，则“杀死”了所有其它使用 x 的表达式
  - 传递函数 $f_s(I) = (I - e\_kill_s) \cup e\_gen_s$
- 基本块 B 的传递函数 $f_B = f_{s_n} \circ \ldots \circ f_{s_2} \circ f_{s_1}$
  - 可以表示为 $f_B(I) = (I - e\_kill_B) \cup e\_gen_B$
  - 类似于活跃变量分析中的 def 和 use、到达定值分析中的 $kill$ 和 gen
- 交汇运算：$I_1 \wedge I_2 = I_1 \cap I_2$，要求任意前驱中都要可用才认为可用
- 顶值（top）：全集（为什么？）

可用表达式分析（3）



$e\_gen_{BB1} = \{x+y, x==0\}$
$e\_kill_{BB1} = \{\}$

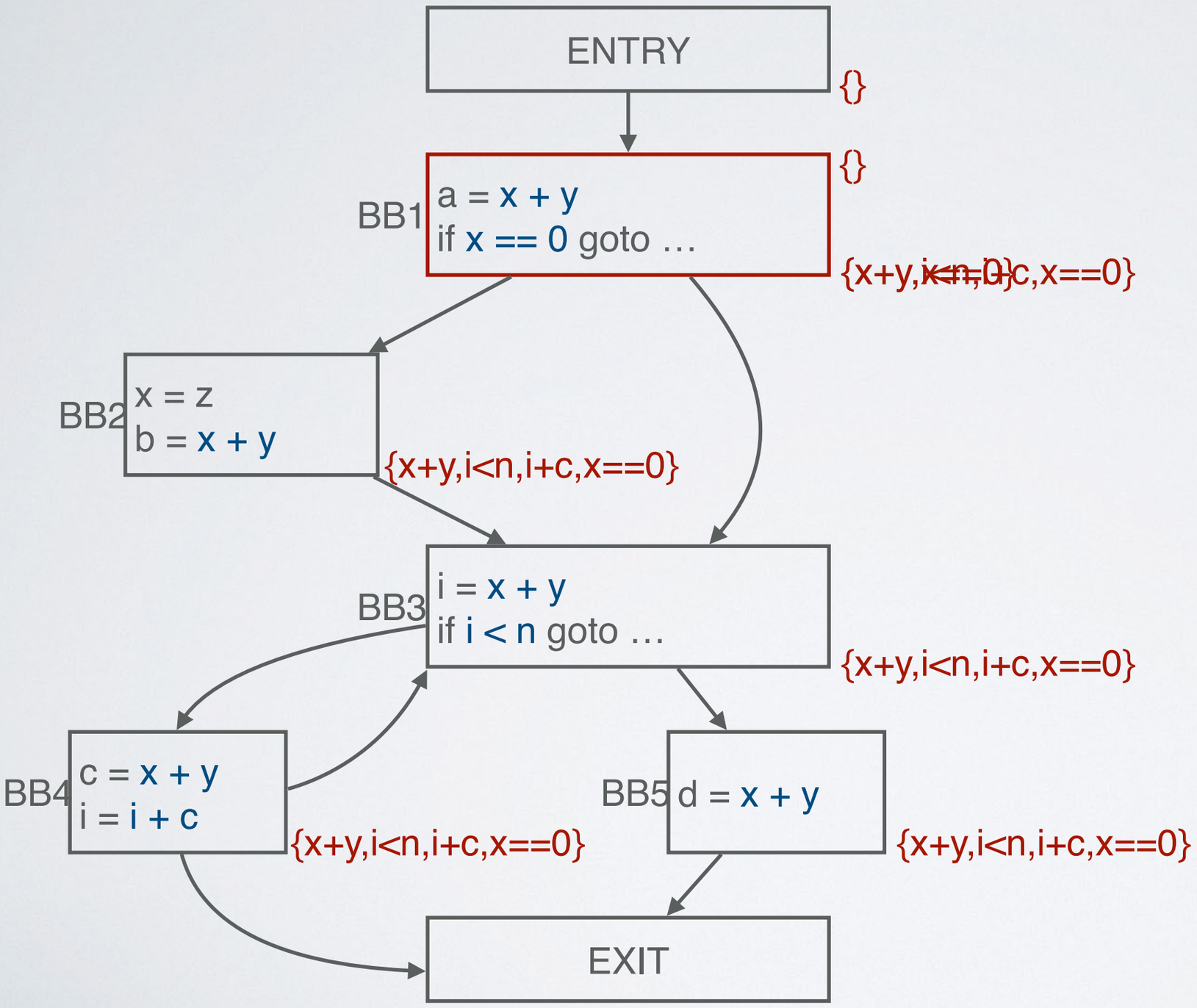
$e\_gen_{BB2} = \{x+y\}$
$e\_kill_{BB2} = \{x+y, x==0\}$

$e\_gen_{BB3} = \{x+y, i<n\}$
$e\_kill_{BB3} = \{i<n, i+c\}$

$e\_gen_{BB4} = \{x+y\}$
$e\_kill_{BB4} = \{i<n, i+c\}$

$e\_gen_{BB5} = \{x+y\}$
$e\_kill_{BB5} = \{\}$

可用表达式分析（4）



$e\_gen_{BB1} = \{x+y, x==0\}$
$e\_kill_{BB1} = \{\}$

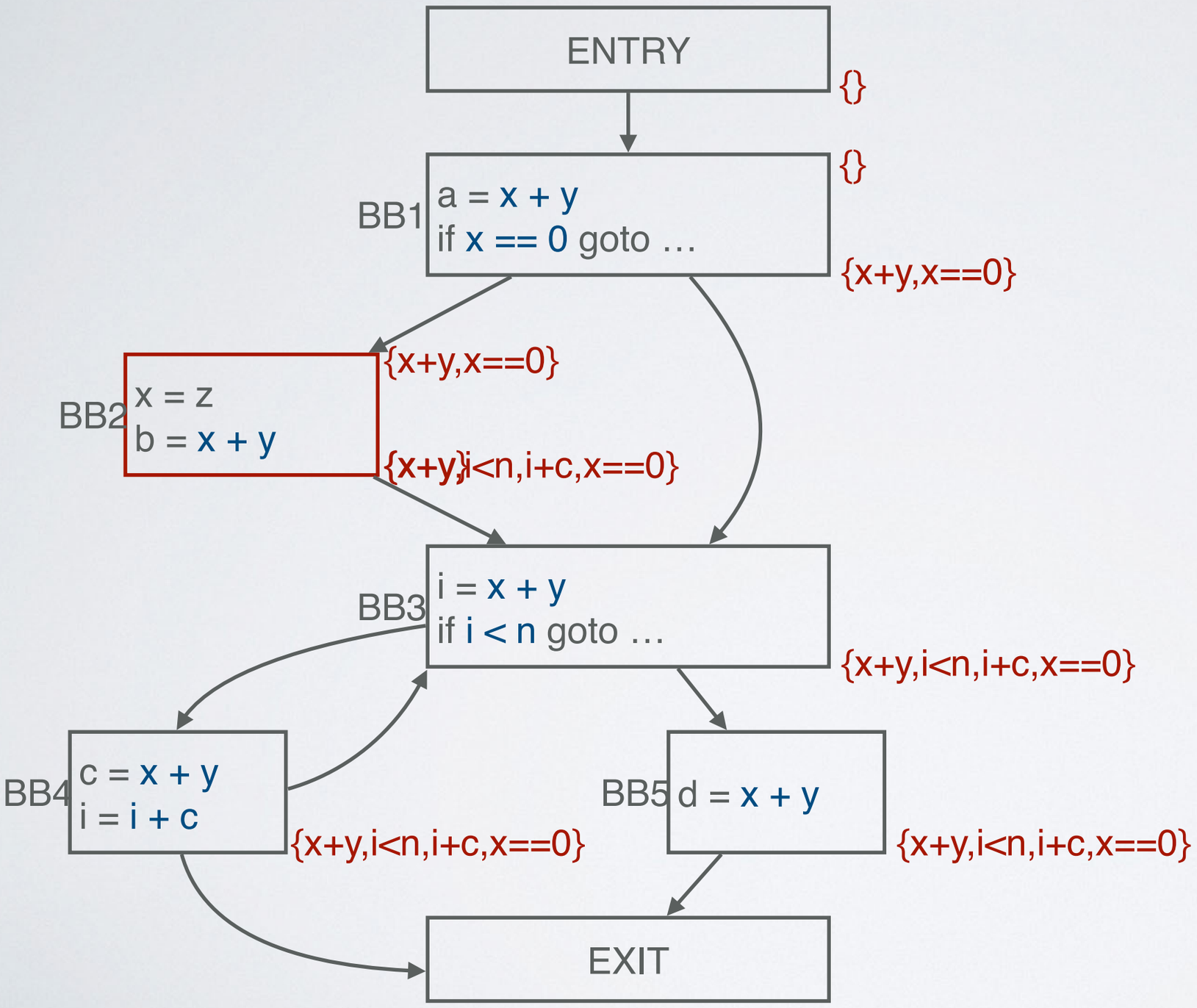
$e\_gen_{BB2} = \{x+y\}$
$e\_kill_{BB2} = \{x+y, x==0\}$

$e\_gen_{BB3} = \{x+y, i<n\}$
$e\_kill_{BB3} = \{i<n, i+c\}$

$e\_gen_{BB4} = \{x+y\}$
$e\_kill_{BB4} = \{i<n, i+c\}$

$e\_gen_{BB5} = \{x+y\}$
$e\_kill_{BB5} = \{\}$

可用表达式分析（5）



$e\_gen_{BB1} = \{x+y, x==0\}$
$e\_kill_{BB1} = \{\}$

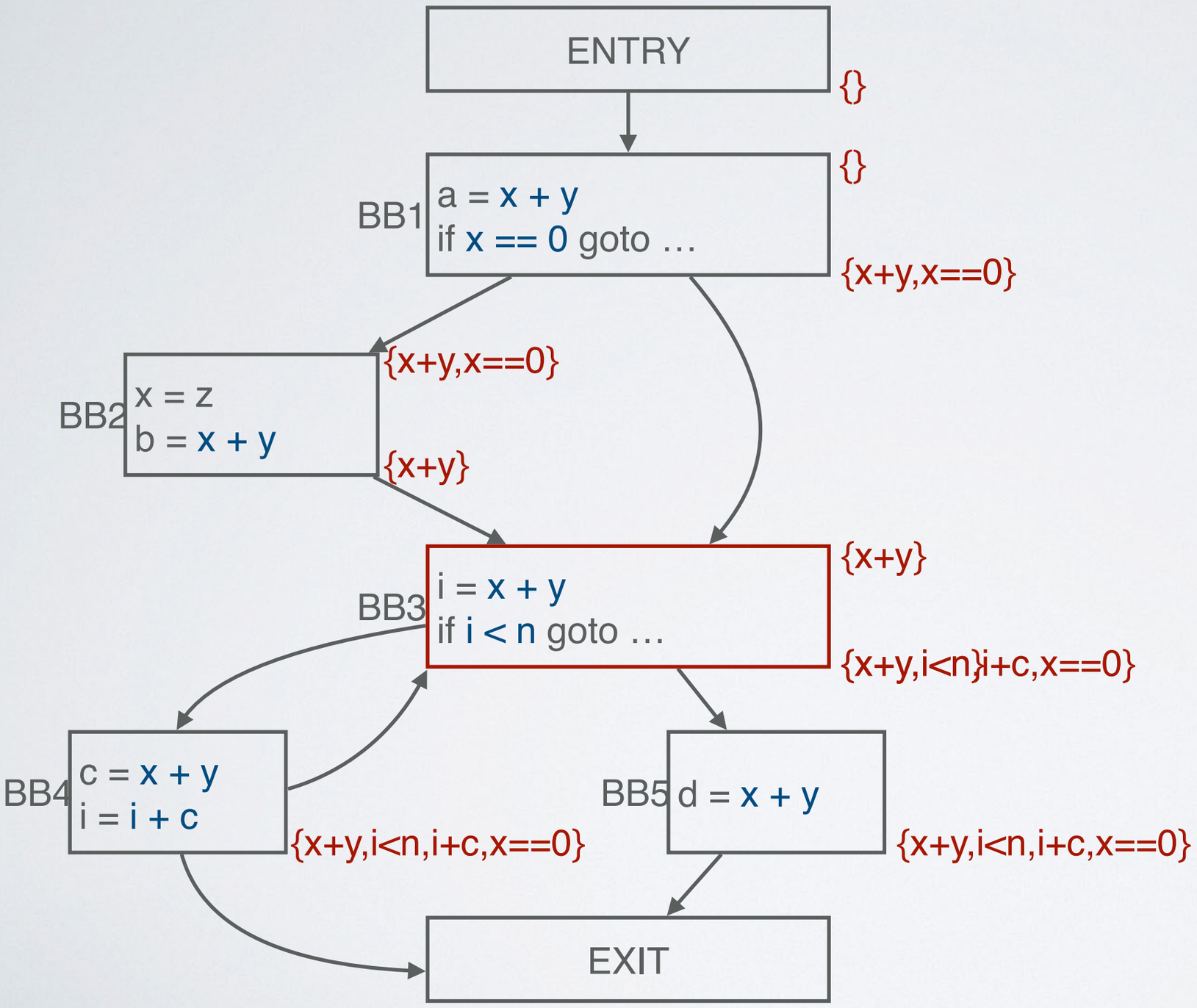
$e\_gen_{BB2} = \{x+y\}$
$e\_kill_{BB2} = \{x+y, x==0\}$

$e\_gen_{BB3} = \{x+y, i<n\}$
$e\_kill_{BB3} = \{i<n, i+c\}$

$e\_gen_{BB4} = \{x+y\}$
$e\_kill_{BB4} = \{i<n, i+c\}$

$e\_gen_{BB5} = \{x+y\}$
$e\_kill_{BB5} = \{\}$

可用表达式分析（6）



$e\_gen_{BB1} = \{x+y, x==0\}$
$e\_kill_{BB1} = \{\}$

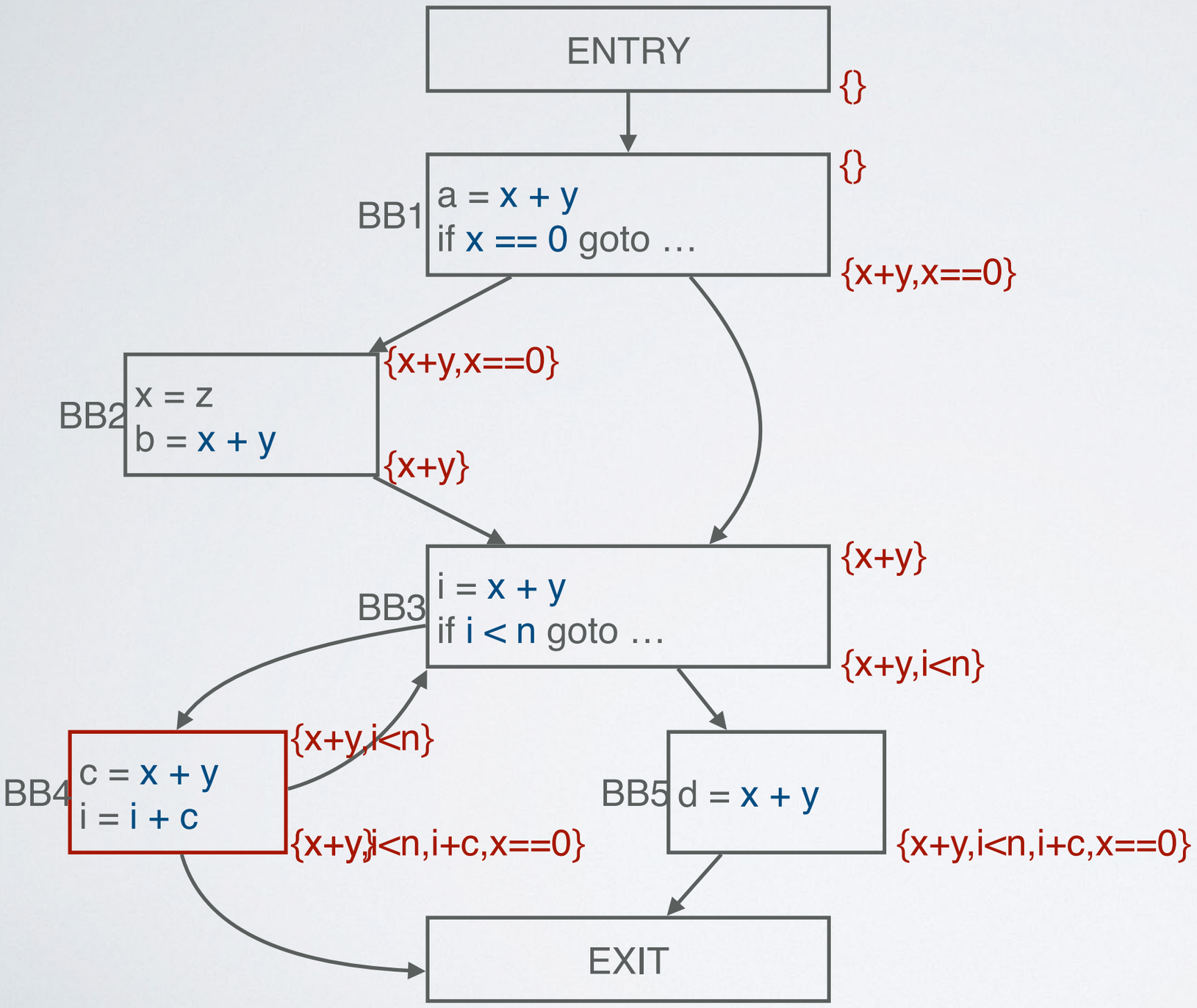
$e\_gen_{BB2} = \{x+y\}$
$e\_kill_{BB2} = \{x+y, x==0\}$

$e\_gen_{BB3} = \{x+y, i<n\}$
$e\_kill_{BB3} = \{i<n, i+c\}$

$e\_gen_{BB4} = \{x+y\}$
$e\_kill_{BB4} = \{i<n, i+c\}$

$e\_gen_{BB5} = \{x+y\}$
$e\_kill_{BB5} = \{\}$

可用表达式分析（7）



$e\_gen_{BB1} = \{x+y, x==0\}$
$e\_kill_{BB1} = \{\}$

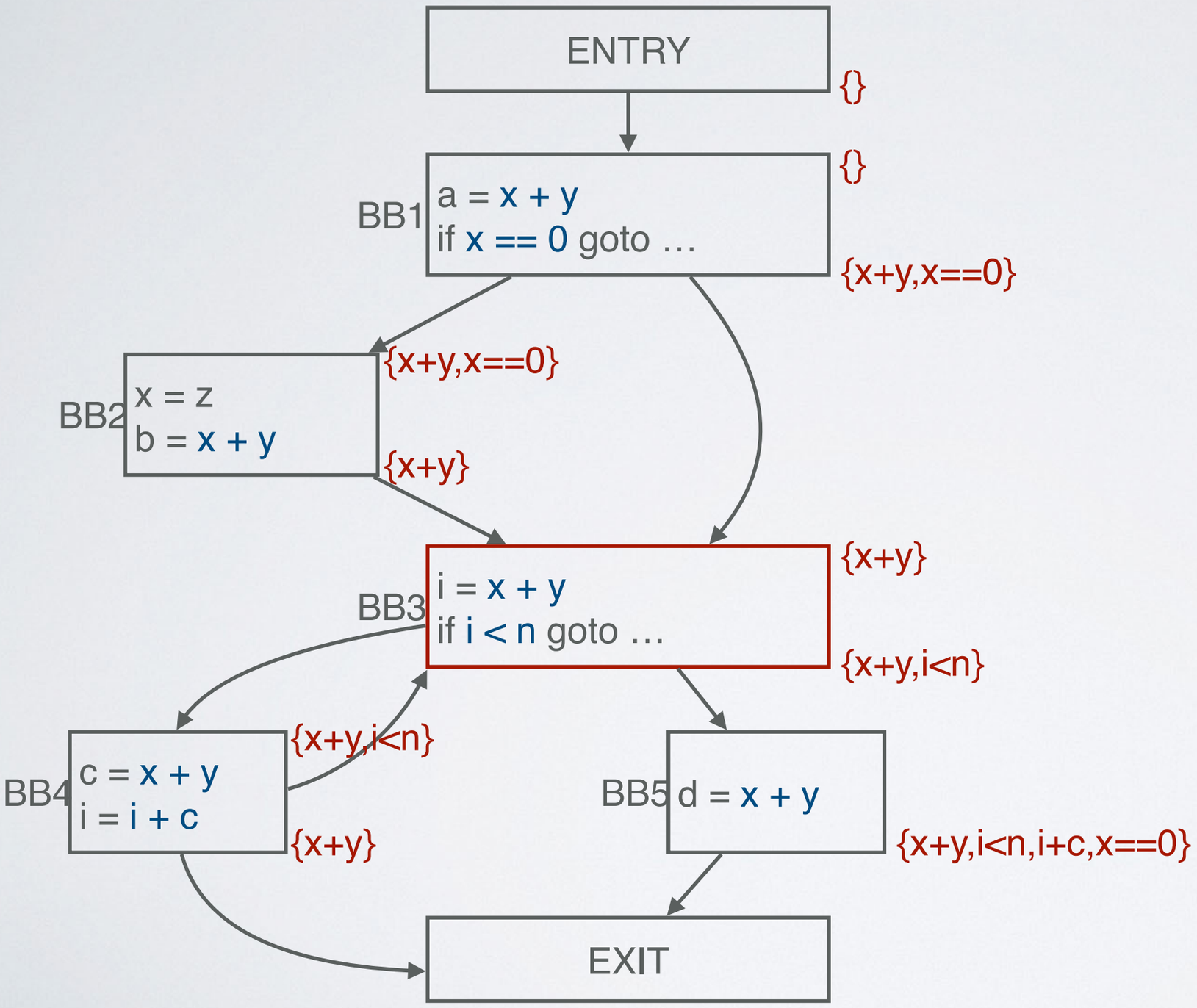
$e\_gen_{BB2} = \{x+y\}$
$e\_kill_{BB2} = \{x+y, x==0\}$

$e\_gen_{BB3} = \{x+y, i<n\}$
$e\_kill_{BB3} = \{i<n, i+c\}$

$e\_gen_{BB4} = \{x+y\}$
$e\_kill_{BB4} = \{i<n, i+c\}$

$e\_gen_{BB5} = \{x+y\}$
$e\_kill_{BB5} = \{\}$

可用表达式分析（8）



$e\_gen_{BB1} = \{x+y, x==0\}$
$e\_kill_{BB1} = \{\}$

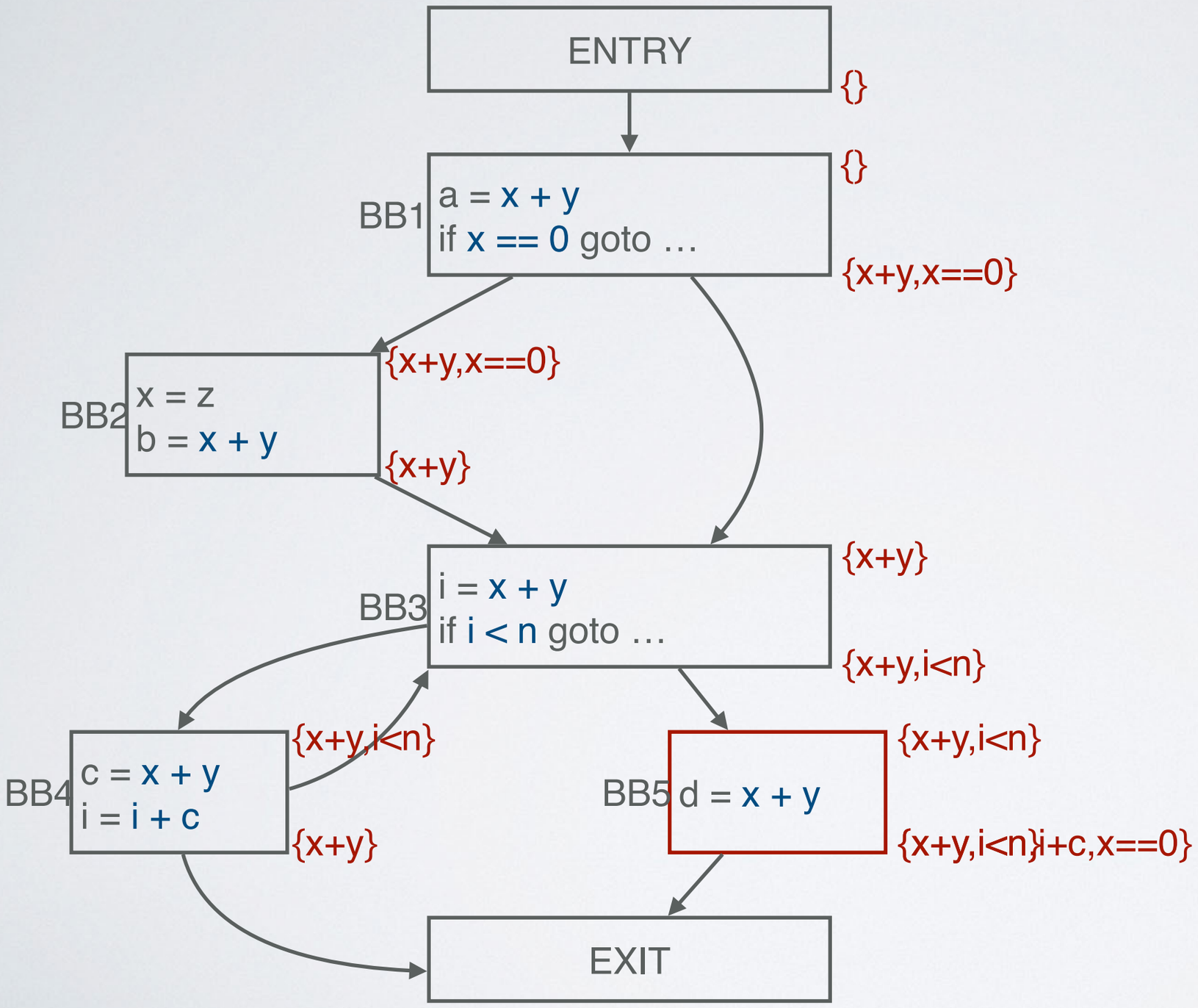
$e\_gen_{BB2} = \{x+y\}$
$e\_kill_{BB2} = \{x+y, x==0\}$

$e\_gen_{BB3} = \{x+y, i<n\}$
$e\_kill_{BB3} = \{i<n, i+c\}$

$e\_gen_{BB4} = \{x+y\}$
$e\_kill_{BB4} = \{i<n, i+c\}$

$e\_gen_{BB5} = \{x+y\}$
$e\_kill_{BB5} = \{\}$

可用表达式分析（9）



$e\_gen_{BB1} = \{x+y, x==0\}$
$e\_kill_{BB1} = \{\}$

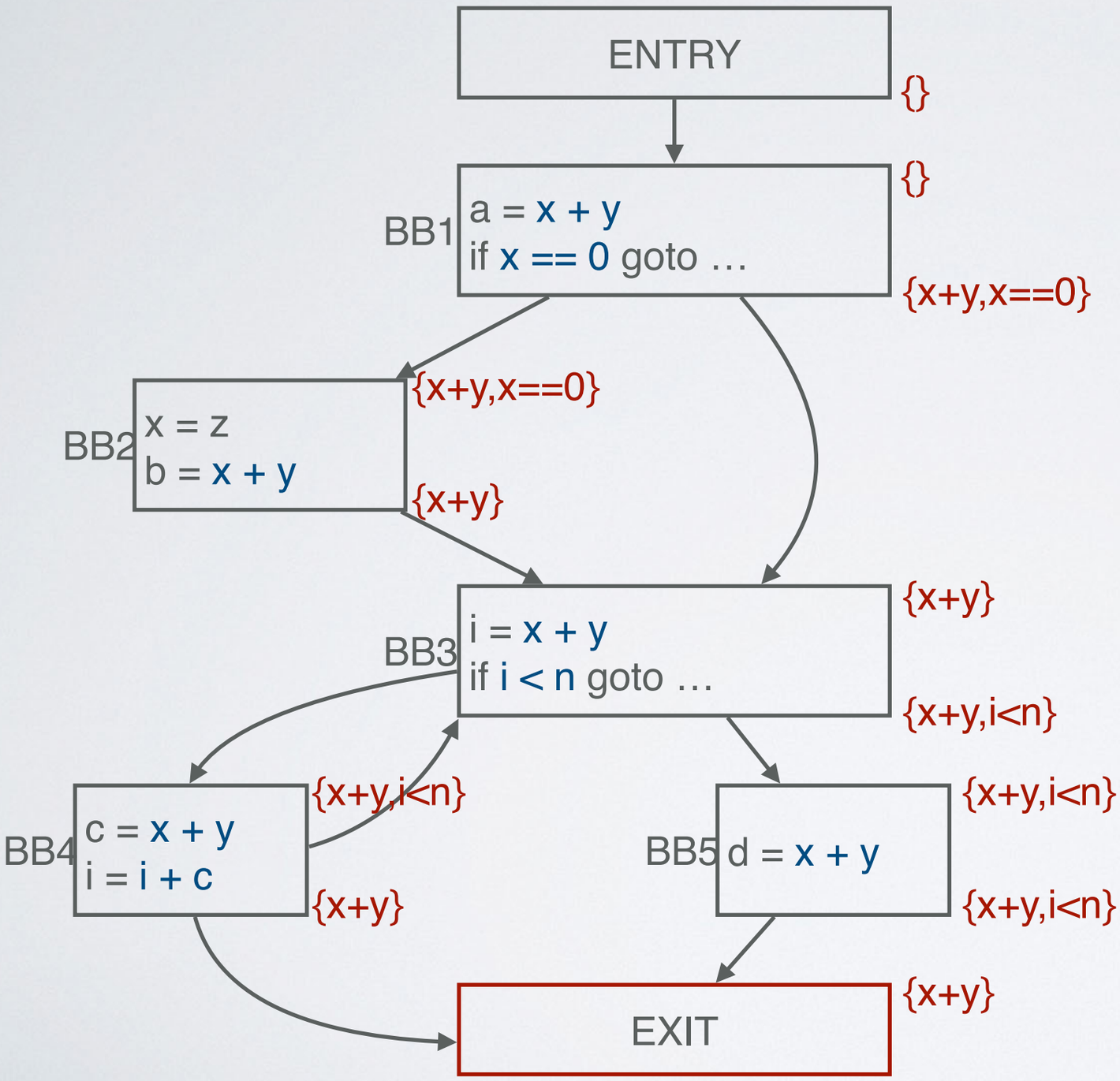
$e\_gen_{BB2} = \{x+y\}$
$e\_kill_{BB2} = \{x+y, x==0\}$

$e\_gen_{BB3} = \{x+y, i<n\}$
$e\_kill_{BB3} = \{i<n, i+c\}$

$e\_gen_{BB4} = \{x+y\}$
$e\_kill_{BB4} = \{i<n, i+c\}$

$e\_gen_{BB5} = \{x+y\}$
$e\_kill_{BB5} = \{\}$

可用表达式分析（10）



$e\_gen_{BB1} = \{x+y, x==0\}$
$e\_kill_{BB1} = \{\}$

$e\_gen_{BB2} = \{x+y\}$
$e\_kill_{BB2} = \{x+y, x==0\}$

$e\_gen_{BB3} = \{x+y, i<n\}$
$e\_kill_{BB3} = \{i<n, i+c\}$

$e\_gen_{BB4} = \{x+y\}$
$e\_kill_{BB4} = \{i<n, i+c\}$

$e\_gen_{BB5} = \{x+y\}$
$e\_kill_{BB5} = \{\}$

小结：数据流分析

- 分析以某程序点为终点/起点的所有路径的集合满足的性质

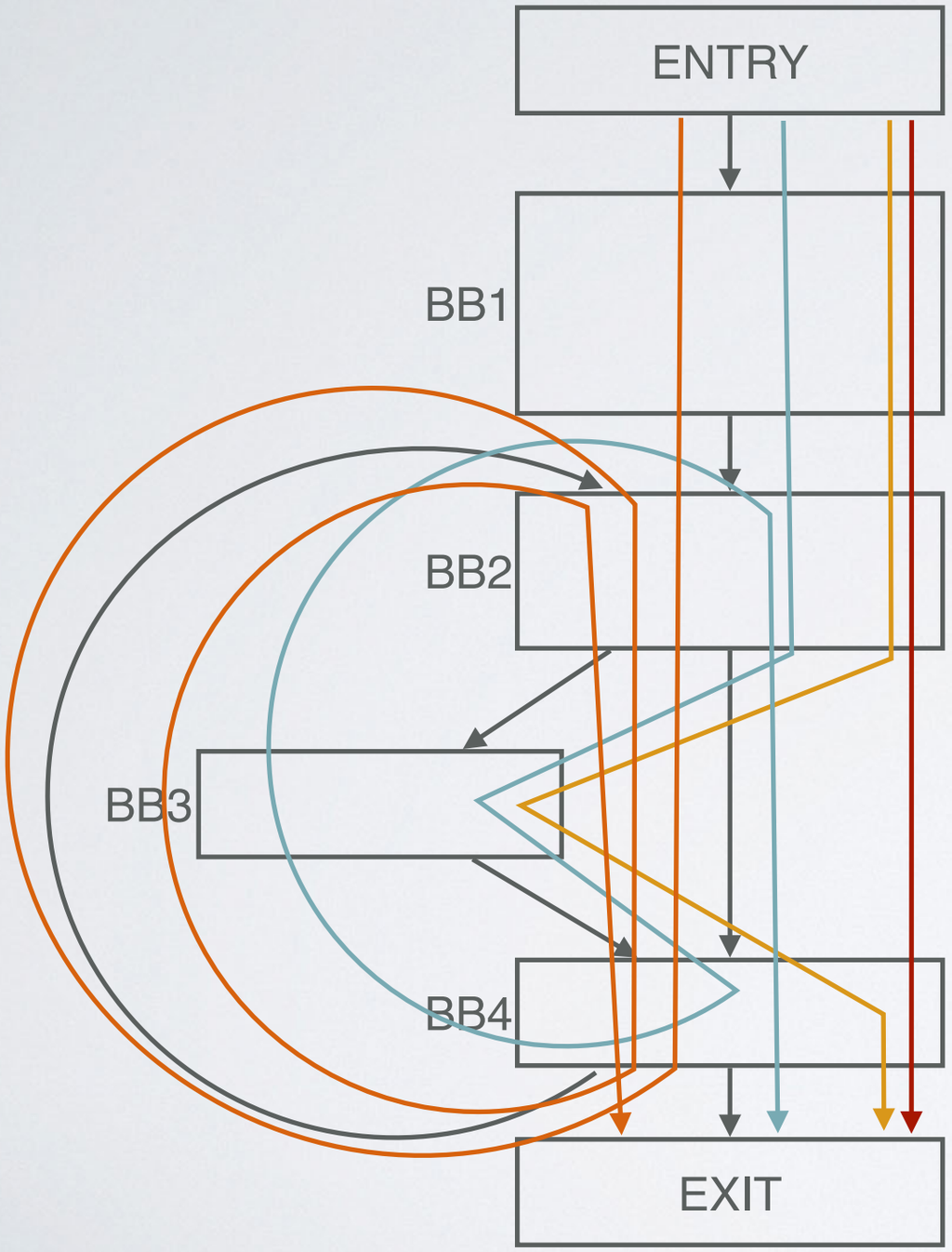
	活跃变量	到达定值	可用表达式
域	变量的集合	定值的集合	表达式的集合
方向	后向	前向	前向
传递函数	$(O - def_B) \cup use_B$	$(I - kill_B) \cup gen_B$	$(I - e\_kill_B) \cup e\_gen_B$
边界条件	$\text{IN[EXIT]} = \varnothing$	$\text{OUT[ENTRY]} = \varnothing$	$\text{OUT[ENTRY]} = \varnothing$
交汇运算	$\cup$	$\cup$	$\cap$
方程组	$\text{OUT}[B] = \bigcup_{S,succ(B)} \text{IN}[S]$ $\text{IN}[B] = f_B(\text{OUT}[B])$	$\text{IN}[B] = \bigcup_{P,pred(B)} \text{OUT}[P]$ $\text{OUT}[B] = f_B(\text{IN}[B])$	$\text{IN}[B] = \bigcap_{P,pred(B)} \text{OUT}[P]$ $\text{OUT}[B] = f_B(\text{IN}[B])$
初始值	$\text{IN}[B] = \varnothing$	$\text{OUT}[B] = \varnothing$	$\text{OUT}[B] =$ 全集

- 数据流分析理论框架参见 9.3 节：正确性、精确性、收敛性

本章内容

- 代码优化的常用方法
- 数据流分析简介
  - 到达定值
  - 活跃变量
  - 可用表达式
- 基于路径表达式的分析

数据流分析是什么?



- 分析流图上的路径集合的性质
  - ENTRY->BB1->BB2->BB4->EXIT
  - ENTRY->BB1->BB2->BB3->BB4->EXIT
  - ENTRY->BB1->BB2->BB3->BB4->BB2->BB4->EXIT
  - ENTRY->BB1->BB2->BB4->BB2->BB4->BB2->BB4->EXIT
  - ……
- 流图上可以有无限多条路径
- 是否可以用一个有限的形式刻画一个无限的集合?

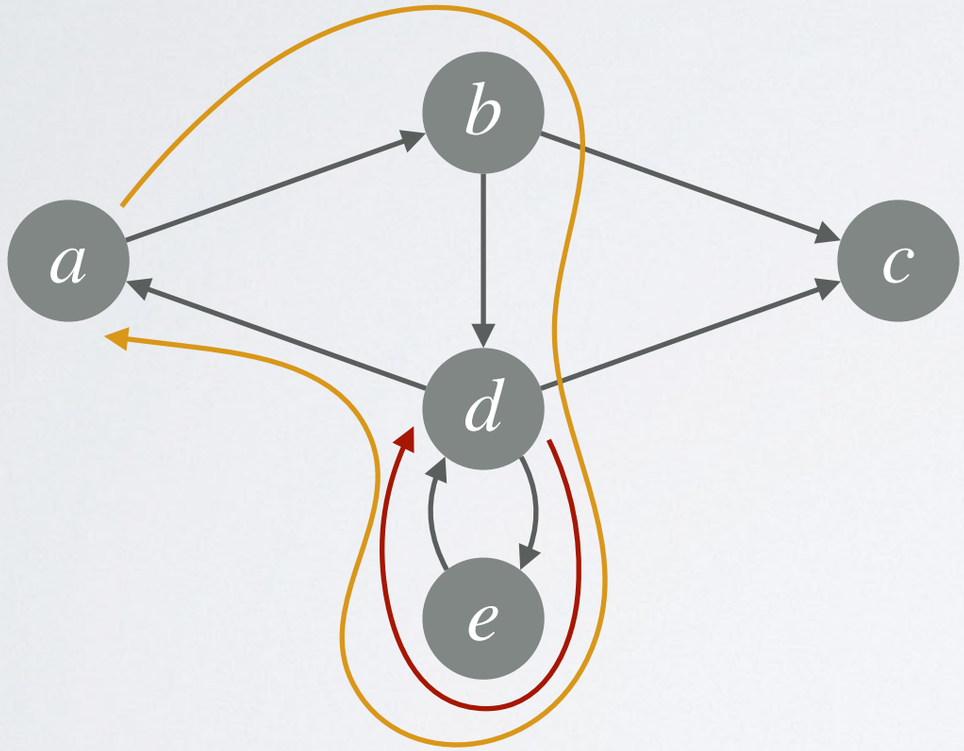
回顾：正则表达式

- 用有限形式（正则表达式）来刻画可能无限的集合（正则语言）
  - ε 表示语言 $L(\varepsilon) = \{\varepsilon\}$
  - 如果 a 是字母表中的符号，则 a 表示语言 $L(a) = \{a\}$
  - 设 R_1 和 R_2 是正则表达式
    - $R_1 \mid R_2$ 表示语言 $L(R_1 \mid R_2) = L(R_1) \cup L(R_2)$
    - R_1R_2 表示语言 $L(R_1R_2) = L(R_1)L(R_2)$
    - R_1^* 表示语言 $L(R_1^*) = L(R_1)^* = \bigcup_{i \geq 0} L(R_1)^i$

- 例子：$0(0 \mid 1)^*$ 刻画了所有以 0 开头的 0/1 符号串

有向图上的路径表达式

- 考虑有向图 $G = (V, E)$，一个路径表达式（path expression）是一个以 E 为字母表的正则表达式 R，且 R 识别的每个符号串都是图 G 中的一条路径



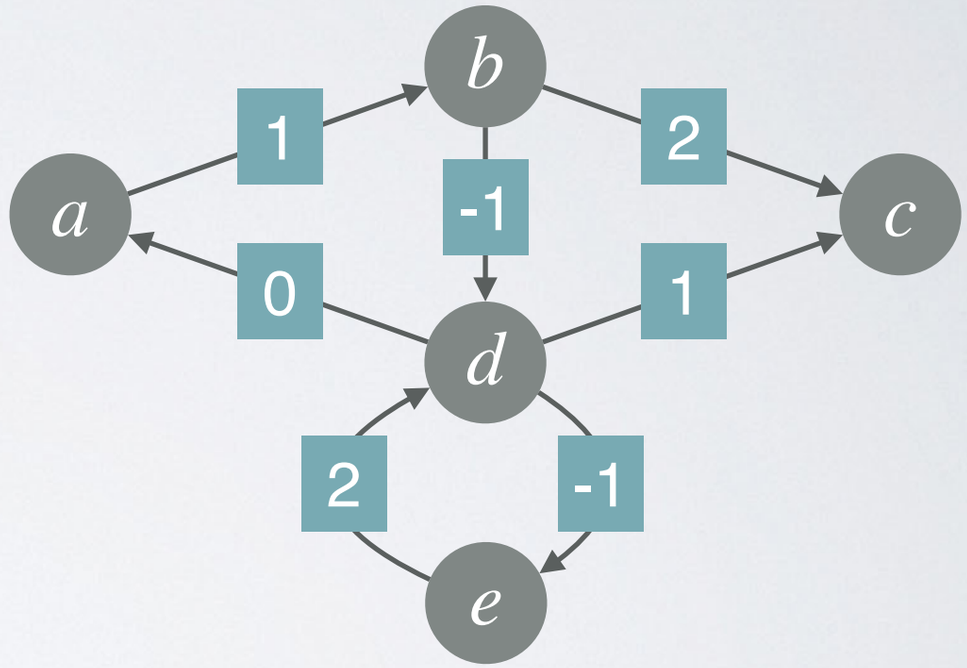
字母表 $\Sigma = \{\langle a, b\rangle, \langle b, d\rangle, \langle b, c\rangle, \langle d, a\rangle, \langle d, c\rangle, \langle d, e\rangle, \langle e, d\rangle, \langle e, d\rangle\}$

识别所有 a 到 c 的路径的路径表达式：

$$(\langle a,b\rangle\langle b,d\rangle(\langle d,e\rangle\langle e,d\rangle)^*\langle d,a\rangle)^*\langle a,b\rangle(\langle b,c\rangle \mid \langle b,d\rangle(\langle d,e\rangle\langle e,d\rangle)^*\langle d,c\rangle)$$

通过路径表达式求最短路（1）

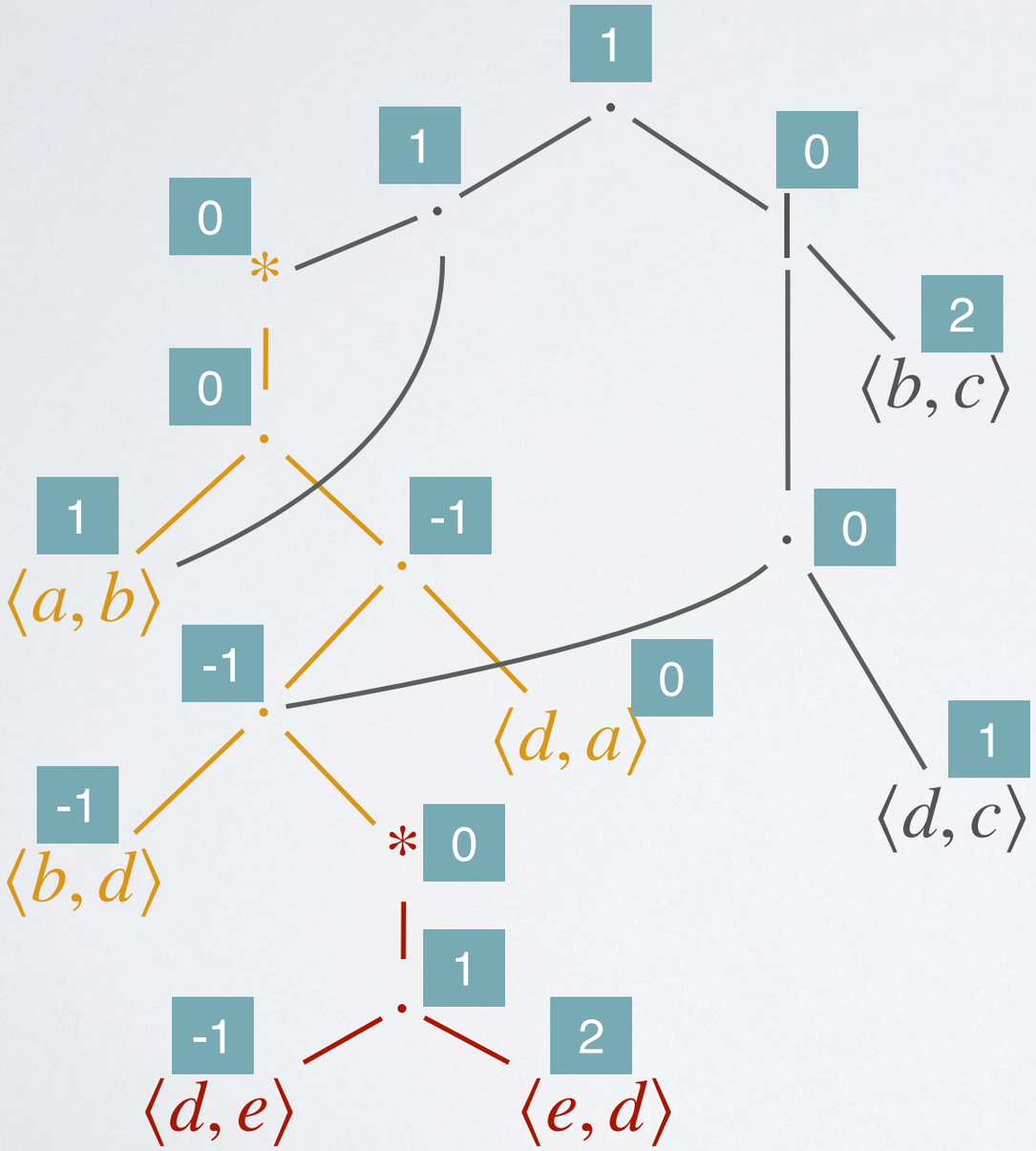
- 用 $F(R)$ 表示 R 能识别的路径的长度的最小值
  - $F(\varepsilon) = 0,\ \ F(\langle u_1, u_2 \rangle) =$ 边 $\langle u_1, u_2 \rangle$ 的长度
  - $F(R_1 \mid R_2) = \min(F(R_1), F(R_2))$
  - $F(R_1 R_2) = F(R_1) + F(R_2)$
  - $F(R_1^*) = ?$
    - 若 $F(R_1) < 0$，则 $F(R_1^*) = -\infty$
    - 否则 $F(R_1^*) = 0$

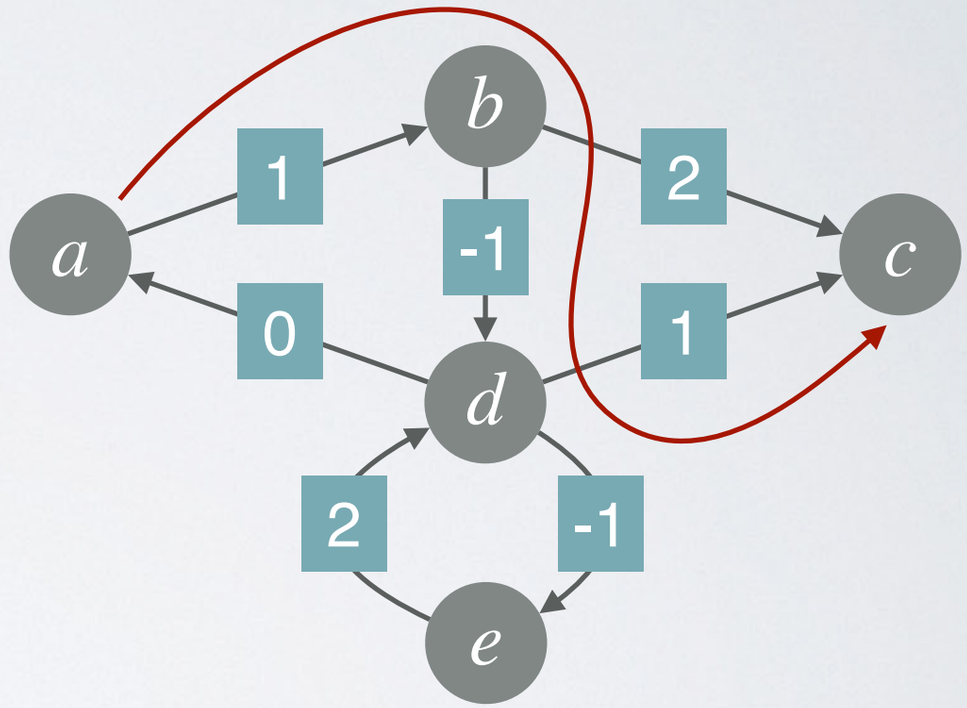


- 例子：
  - $F(\langle d, e \rangle) = -1,\ \ F(\langle e, d \rangle) = 2$
  - $F(\langle d, e \rangle \langle e, d \rangle) = (-1) + 2 = 1$
  - $F((\langle d, e \rangle \langle e, d \rangle)^*) = 0$

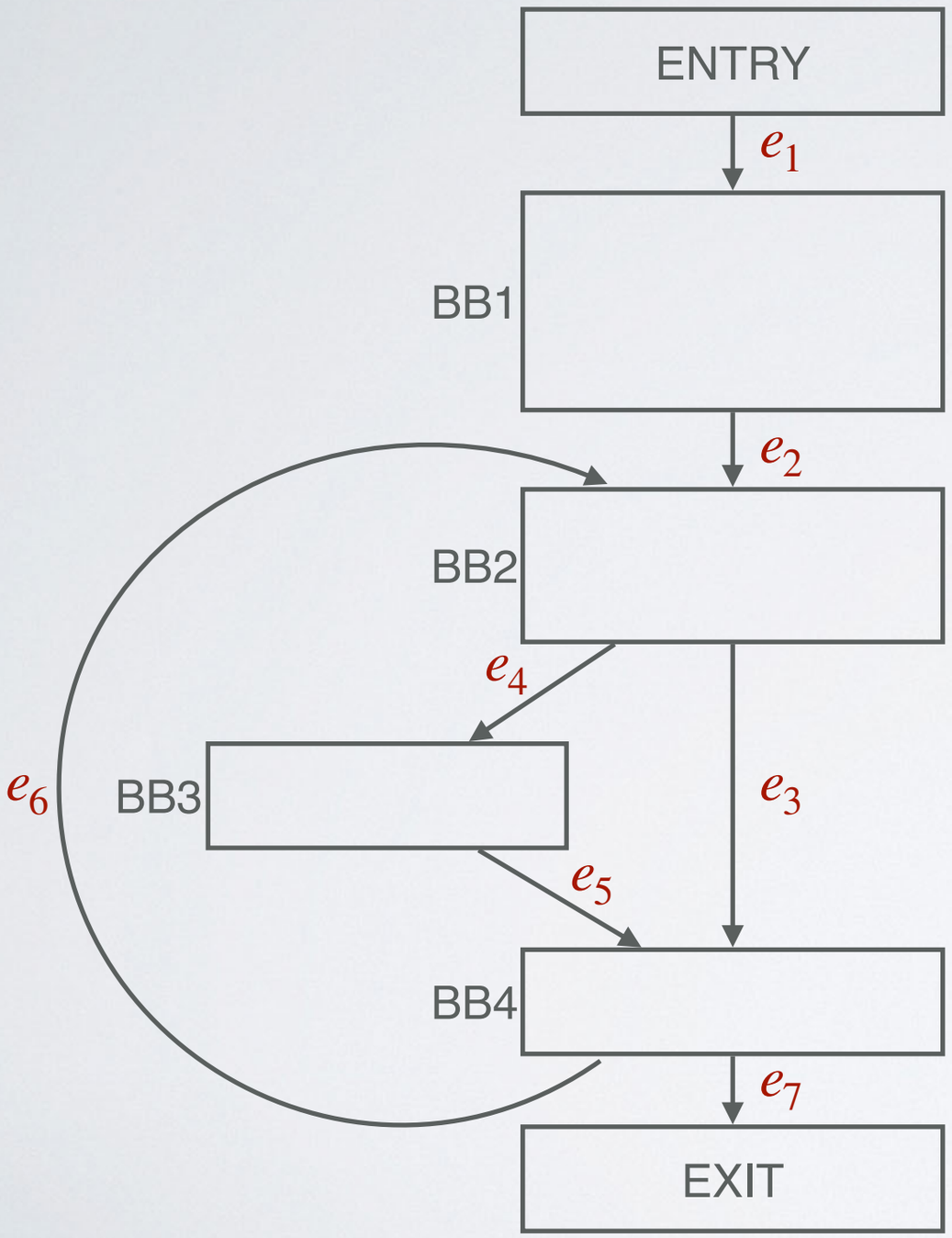
通过路径表达式求最短路（2）

$$F(\ (\langle a,b\rangle\langle b,d\rangle(\langle d,e\rangle\langle e,d\rangle)^*\langle d,a\rangle)^*\langle a,b\rangle(\langle b,c\rangle \mid \langle b,d\rangle(\langle d,e\rangle\langle e,d\rangle)^*\langle d,c\rangle)?$$





流图上的路径表达式



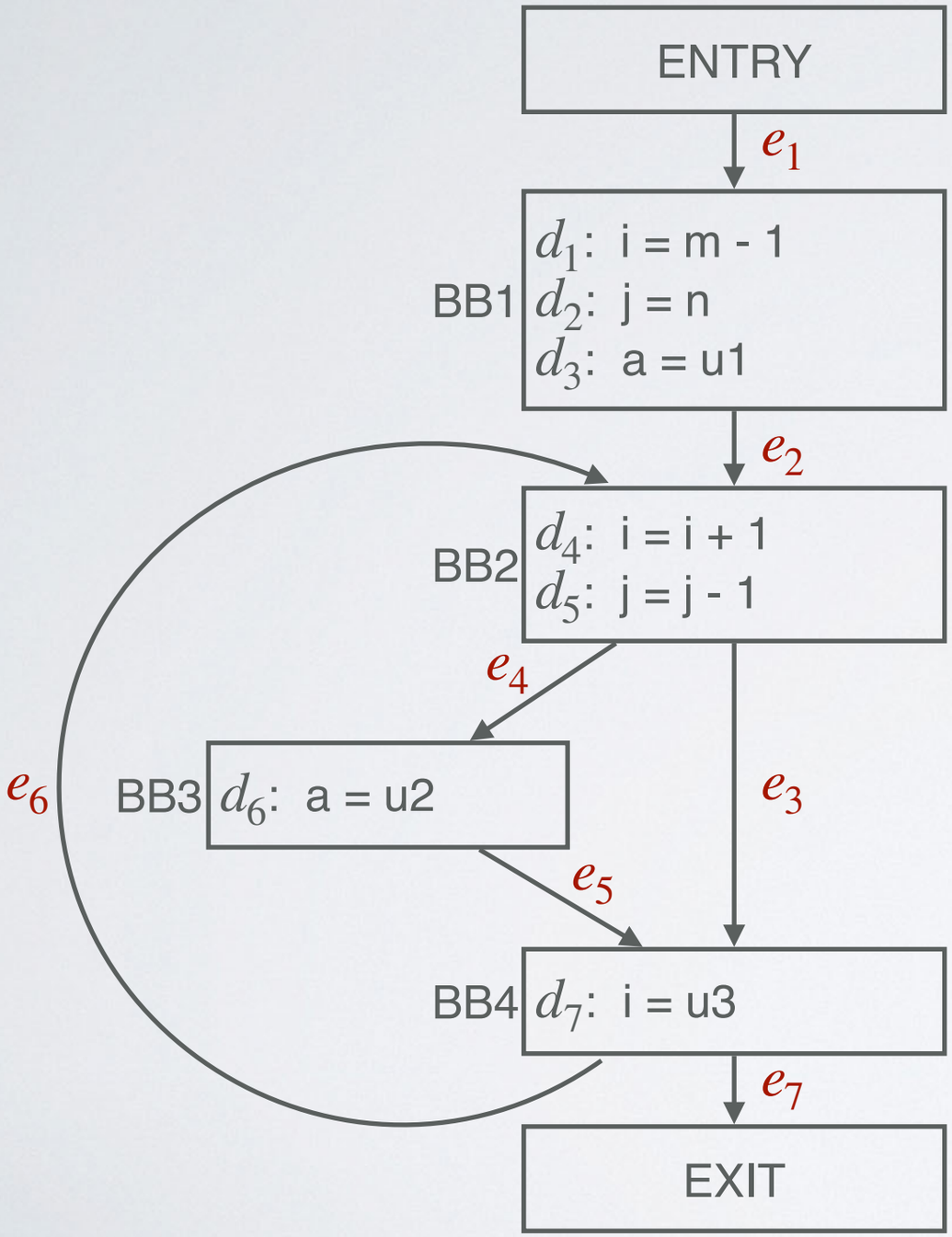
- 给每条边一个标号
- 前向分析：从 ENTRY 出发的路径
  - BB1: e_1
  - BB2: $e_1e_2((e_3 \mid e_4e_5)e_6)^*$
  - BB3: $e_1e_2((e_3 \mid e_4e_5)e_6)^*e_4$
  - BB4: $e_1e_2((e_3 \mid e_4e_5)e_6)^*(e_3 \mid e_4e_5)$
  - EXIT: $e_1e_2((e_3 \mid e_4e_5)e_6)^*(e_3 \mid e_4e_5)e_7$
- 后向分析：到达 EXIT 的路径
  - BB4: $(e_6(e_3 \mid e_4e_5))^*e_7$
  - BB3: $e_5(e_6(e_3 \mid e_4e_5))^*e_7$
  - BB2: $(e_3 \mid e_4e_5)(e_6(e_3 \mid e_4e_5))^*e_7$
  - BB1: $e_2(e_3 \mid e_4e_5)(e_6(e_3 \mid e_4e_5))^*e_7$
  - ENTRY: $e_1e_2(e_3 \mid e_4e_5)(e_6(e_3 \mid e_4e_5))^*e_7$

基于路径表达式的数据流分析

- 数据流分析的域 V，交汇运算 $\wedge : V \times V \to V$
- 每个基本块 B 的传递函数 $f_B : V \to V$
- 用 $F(R) : V \to V$ 表示 R 能识别的路径的数据流抽象

- 以前向分析为例
  - $F(\varepsilon) =$ 恒等函数，$F(e) = f_{h(e)}$ 其中 $h(e)$ 是边 e 的起点基本块
  - $F(R_1 \mid R_2) = F(R_1) \wedge F(R_2)$
  - $F(R_1 R_2) = F(R_2) \circ F(R_1)$
  - $F(R_1^*) = \bigwedge_{i \geq 0} F(R_1)^i$，不过有时能找到更高效的算法

例子：到达定值分析



数据流抽象：$F(R)(I) = (I - kill_R) \cup gen_R$

$gen_{e_1} = \{\}$ $gen_{e_2} = \{d_1, d_2, d_3\}$
$kill_{e_1} = \{\}$ $kill_{e_2} = \{d_4, d_5, d_6, d_7\}$

$gen_{e_3} = gen_{e_4} = \{d_4, d_5\}$ $gen_{e_5} = \{d_6\}$
$kill_{e_3} = kill_{e_4} = \{d_1, d_2, d_7\}$ $kill_{e_5} = \{d_3\}$

$gen_{e_6} = gen_{e_7} = \{d_7\}$
$kill_{e_6} = kill_{e_7} = \{d_1, d_4\}$

$gen_{e_4e_5} = \{d_4, d_5, d_6\}$
$kill_{e_4e_5} = \{d_1, d_2, d_3, d_7\}$

$gen_{e_3|e_4e_5} = \{d_4, d_5, d_6\}$
$kill_{e_3|e_4e_5} = \{d_1, d_2, d_7\}$

$gen_{(e_3|e_4e_5)^*} = \{d_4, d_5, d_6\}$
$kill_{(e_3|e_4e_5)^*} = \{\}$

不需要迭代

本章小结

- 理解代码优化的常用方法
  - 常量折叠、公共子表达式消除、复写传播、死代码消除、循环优化（代码外提、归纳变量消除、强度消减）
- 理解数据流分析框架
  - 掌握三种数据流分析（到达定值、活跃变量、可用表达式）的基本算法

作业

- 代码优化的常用方法
  - 9.2.1，9.2.2
- 数据流分析简介
  - 9.2.3

- 不设DDL、自行完成