第八讲　代码优化

Code Optimization

主要内容

- 代码优化的常用方法
- 局部优化：基本块的优化
- 局部优化：窥孔优化
- 全局优化：数据流分析

- 对应章节：第 9 章

主要内容

- **代码优化的常用方法**
- 局部优化：基本块的优化
- 局部优化：窥孔优化
- 全局优化：数据流分析

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

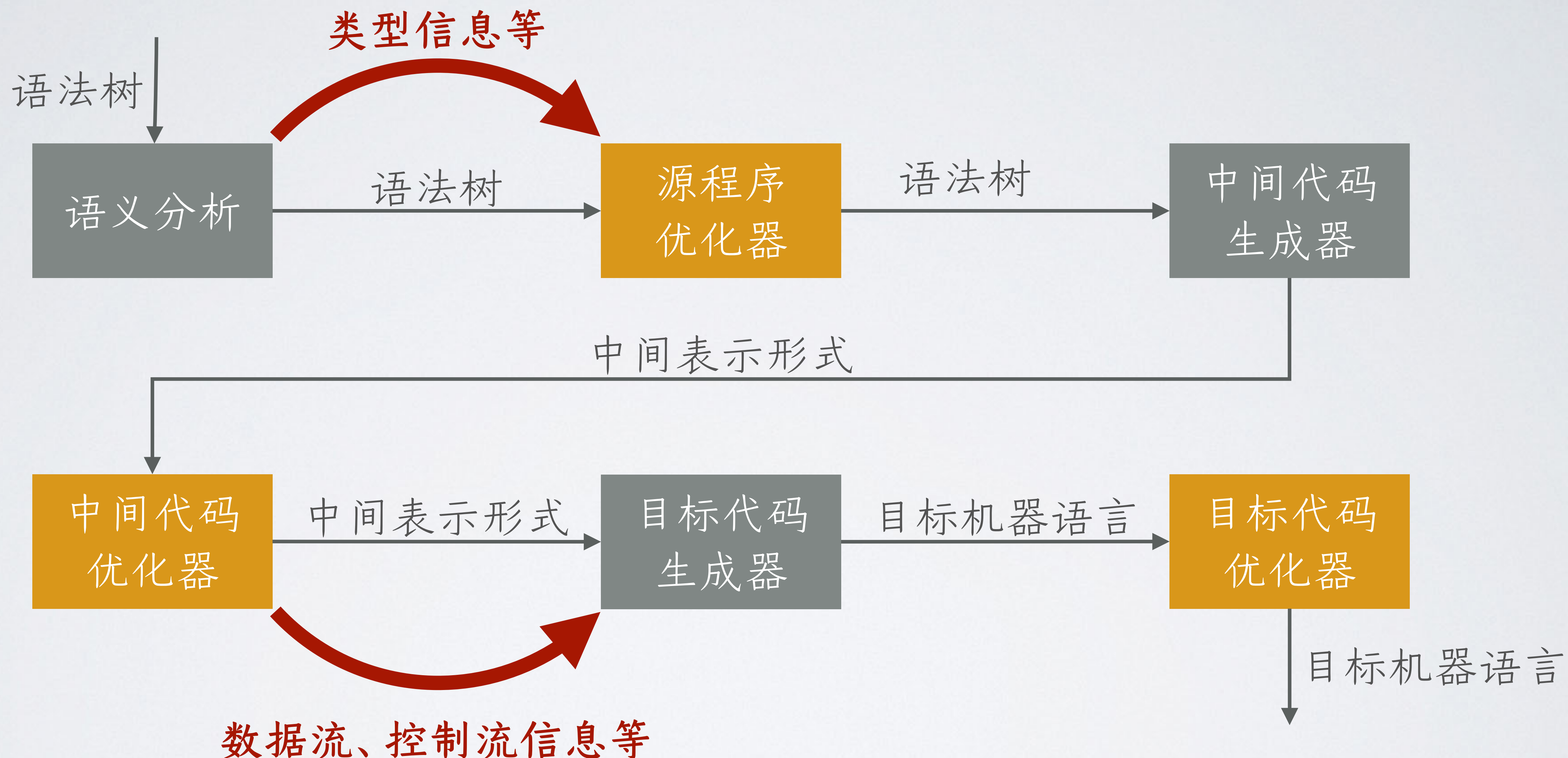
代码优化应遵循的原则

- **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

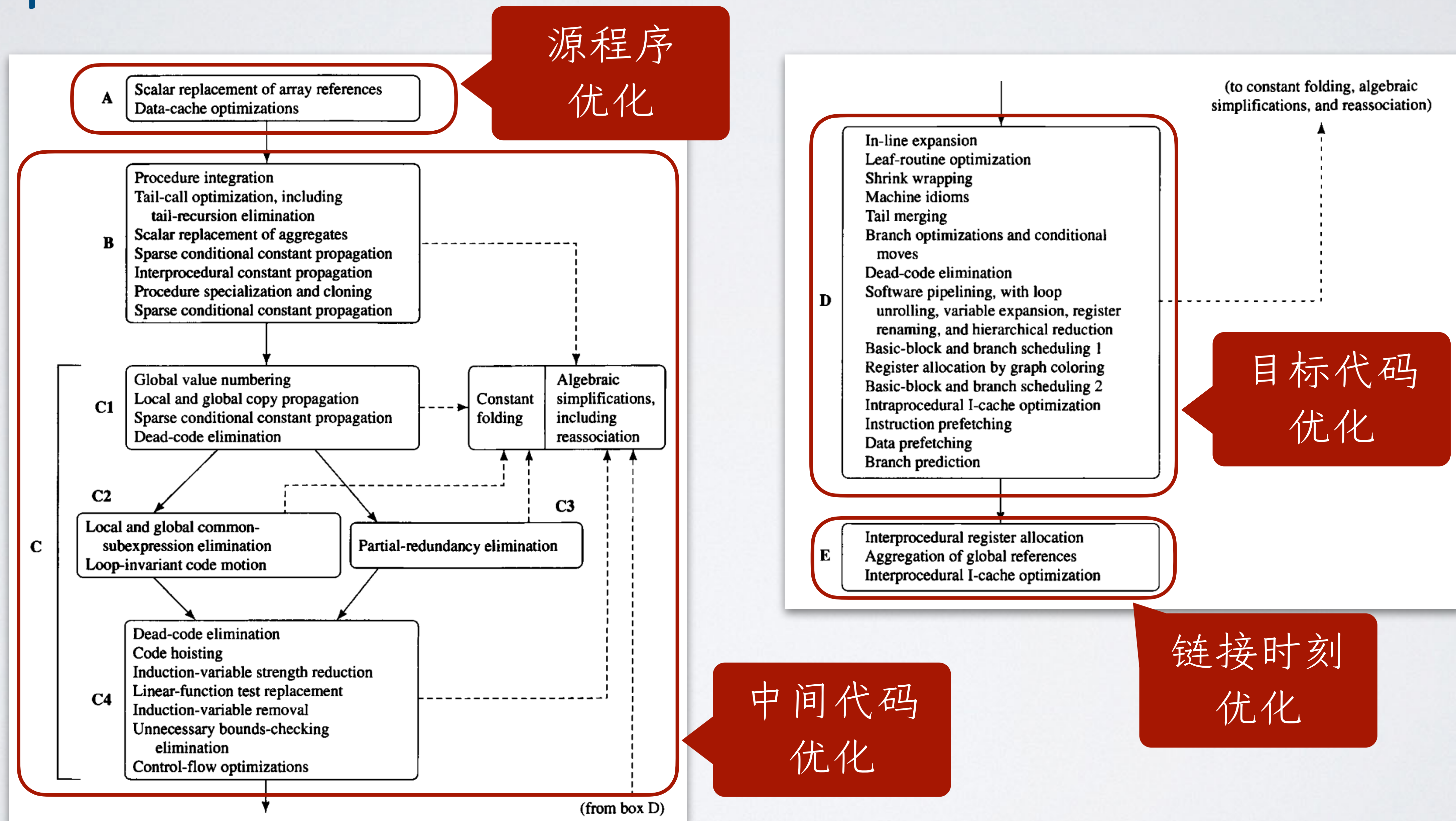
- **算法设计阶段的优化**
  - 通常由程序员进行
  - 比如：把选择排序换成快速排序
- 编译阶段的优化
  - **语义分析阶段**
    - 通过静态检查等的信息，在源程序上进行优化
  - **中间代码生成阶段**
    - 机器无关优化：在中间代码上进行优化
  - **目标代码生成阶段**
    - 机器有关优化：在目标代码上进行优化
  - **(链接时刻优化，link-time optimization)**

代码优化器的结构



现代代码优化器的复杂性（1）

- Steven S. Muchnick, **Advanced Compiler Design and Implementation**



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	Function Integration/Inlining	Unify function exit nodes
Profile Guided Block Placement	Combine redundant instructions	Remove unused exception handling
Break critical edges in CFG	Internalize Global Symbols	Reassociate expressions
Merge Duplicate Global	Interprocedural constant propa.	Demote all values to stack slots
Simple constant propagation	Jump Threading	Sparse Conditional Cons. Propaga.
Dead Code Elimination	Loop-Closed SSA Form Pass	Simplify the CFG
Dead Argument Elimination	**Loop Strength Reduction**	Code sinking
Dead Type Elimination	Rotate Loops	Strip all symbols from a module
Dead Instruction Elimination	**Loop Invariant Code Motion**	Strip debug info for unused symbols
Dead Store Elimination	Canonicalize natural loops	Strip Unused Function Prototypes
Deduce function attributes	Unroll loops	Strip all `llvm.dbg.declare` intrinsics
Dead Global Elimination	Unswitch loops	Tail Call Elimination
Global Variable Optimizer	Promote Memory to Register	Delete dead loops
Global Value Numbering	MemCpy Optimization	Extract loops into new functions
Canonicalize Induction Variables	Merge Functions	……

代码优化的范围

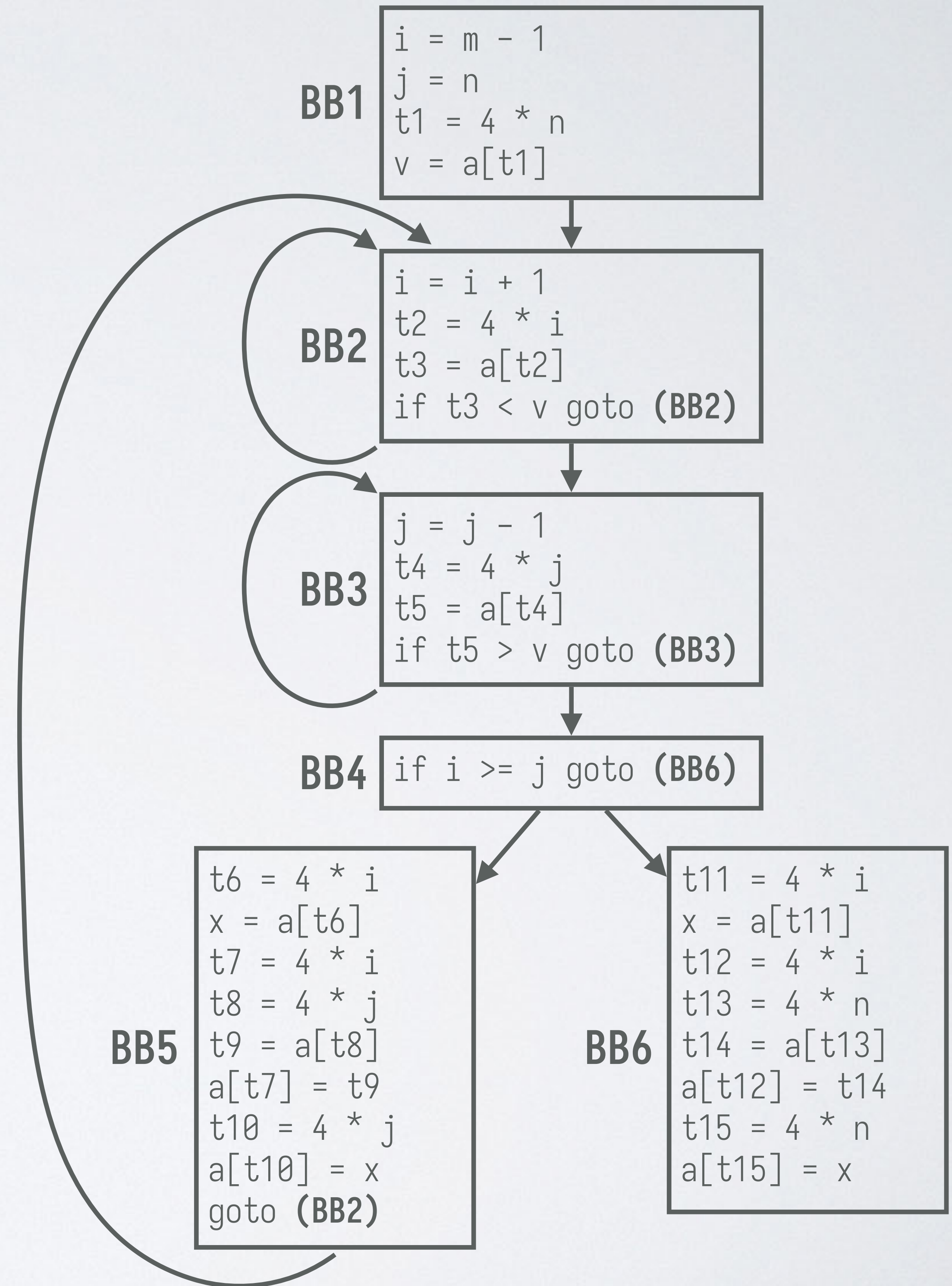
- 局部优化(local optimization)
  - 只对基本块内的语句进行分析，在基本块内进行优化
- 区域性优化(regional optimization)
  - 对若干个基本块构成的区域进行分析，比如对循环的优化
- 全局优化(global optimization)
  - 对一个过程所有基本块的信息和它们的关系进行分析，在此基础上在整个过程范围内进行优化
- 过程间优化(interprocedural optimization)
  - 对一个程序所有过程及其调用信息进行分析，对整个程序整体优化

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
- 常量传播和折叠
- 循环优化
  - 代码外提
  - 强度消减
  - 归纳变量消除

例子：快速排序

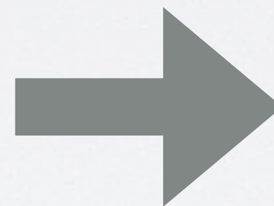
```
void quicksort(int m, int n)
{
  int i, j;
  int v, x;
  if (n <= m) return;
  /* 片段由此开始 */
  i = m - 1; j = n; v = a[n];
  while (1) {
    do i = i + 1; while (a[i] < v);
    do j = j - 1; while (a[j] > v);
    if (i >= j) break;
    x = a[i]; a[i] = a[j]; a[j] = x;
  }
  x = a[i]; a[i] = a[n];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

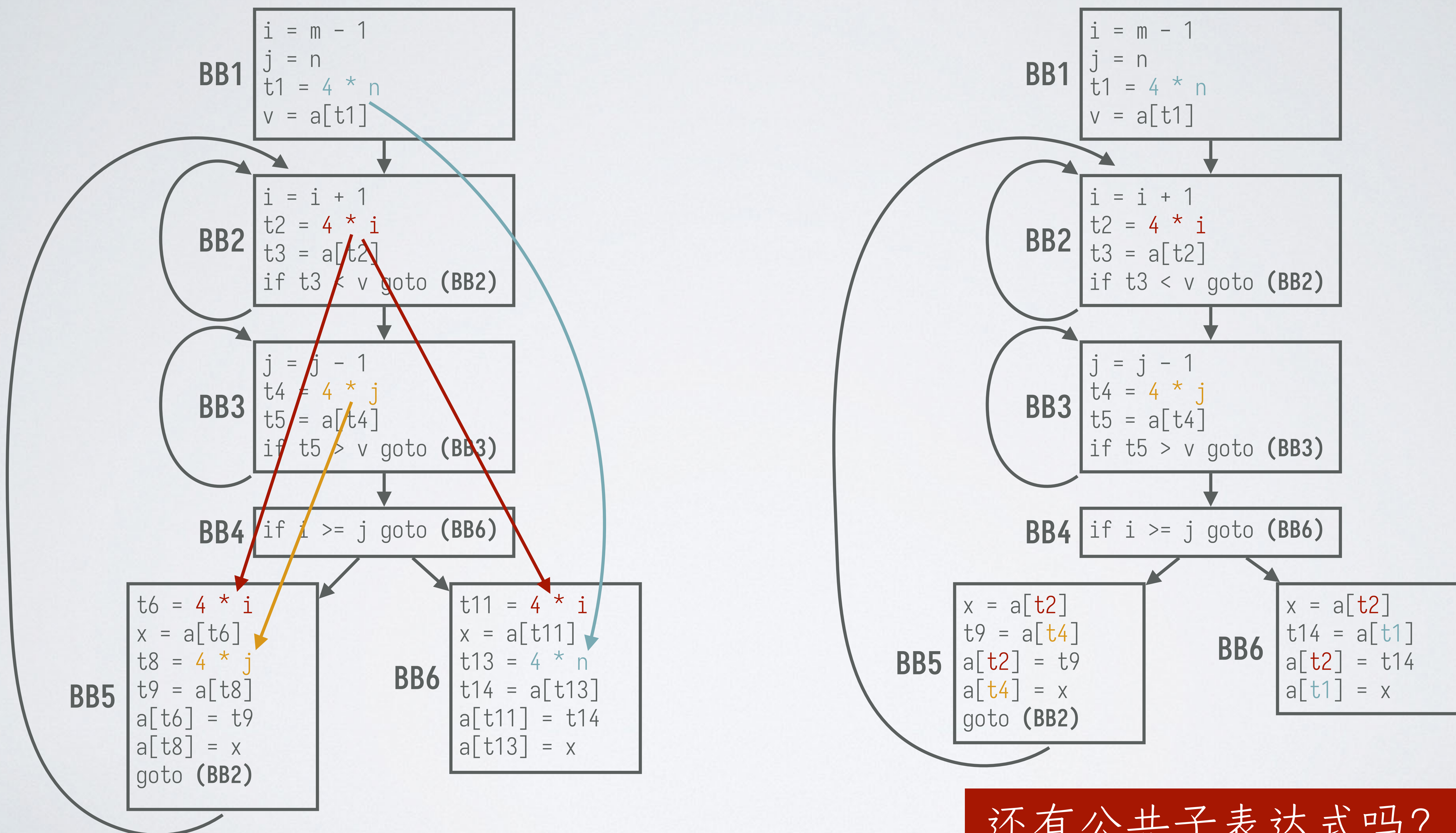
BB5



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

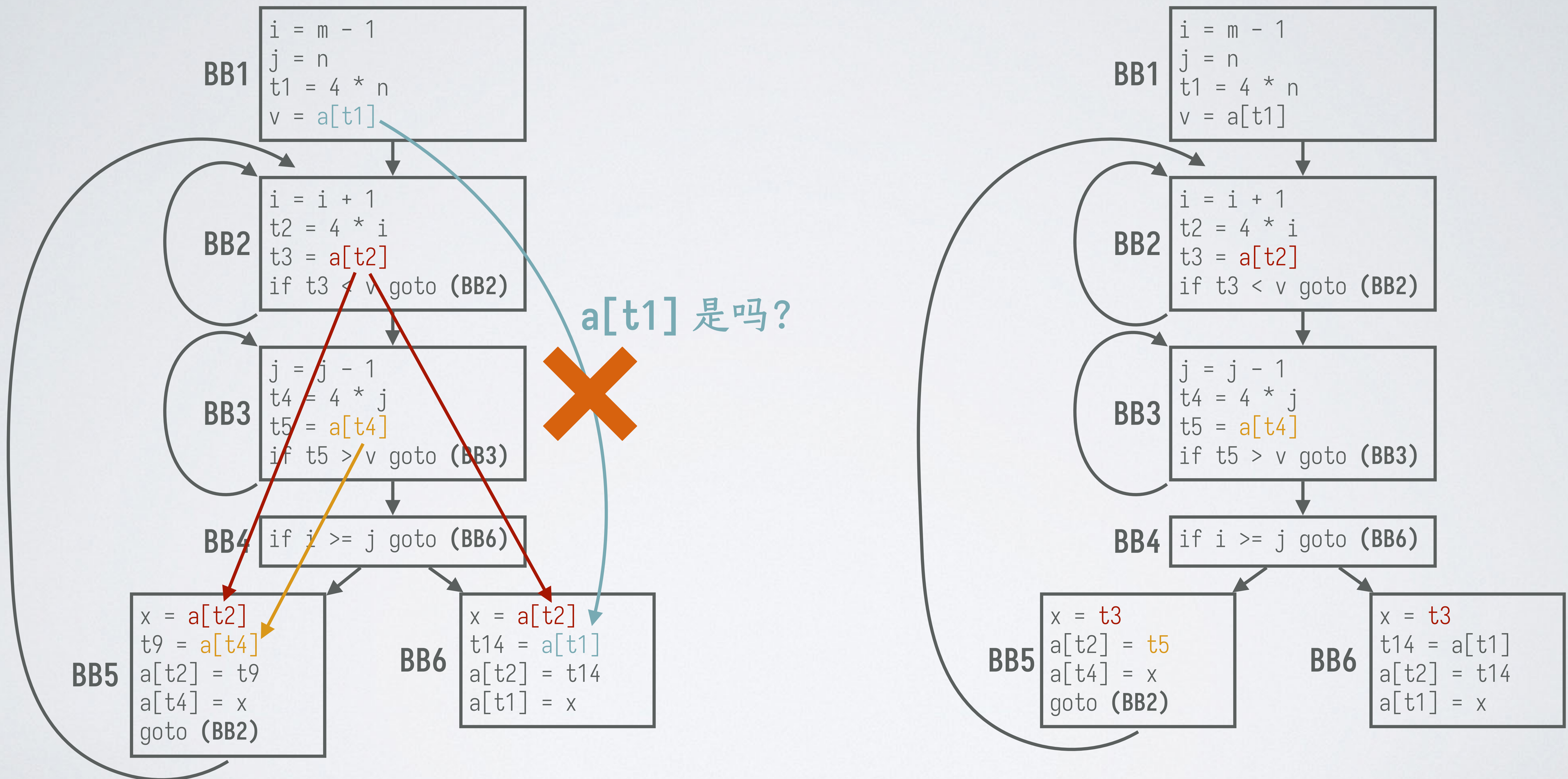
BB5

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）

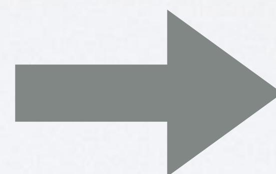


复写传播

- 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 $x = y$

```
x = t3
a[t2] = t5
a[t4] = x
goto (BB2)
```

BB5



```
x = t3
a[t2] = t5
a[t4] = t3
goto (BB2)
```

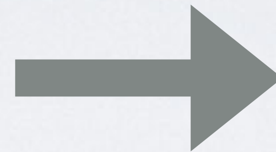
BB5

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

```
x = t3
/* x 在此不活跃 */
a[t2] = t5
a[t4] = t3
goto (BB2)
```

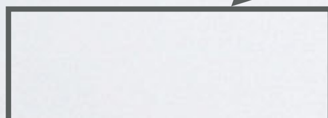
BB5

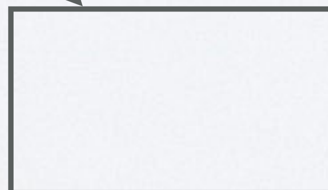


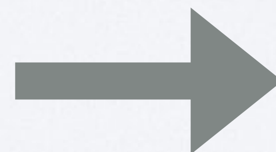
```
a[t2] = t5
a[t4] = t3
goto (BB2)
```

BB5

```
if false goto (BB2)
```



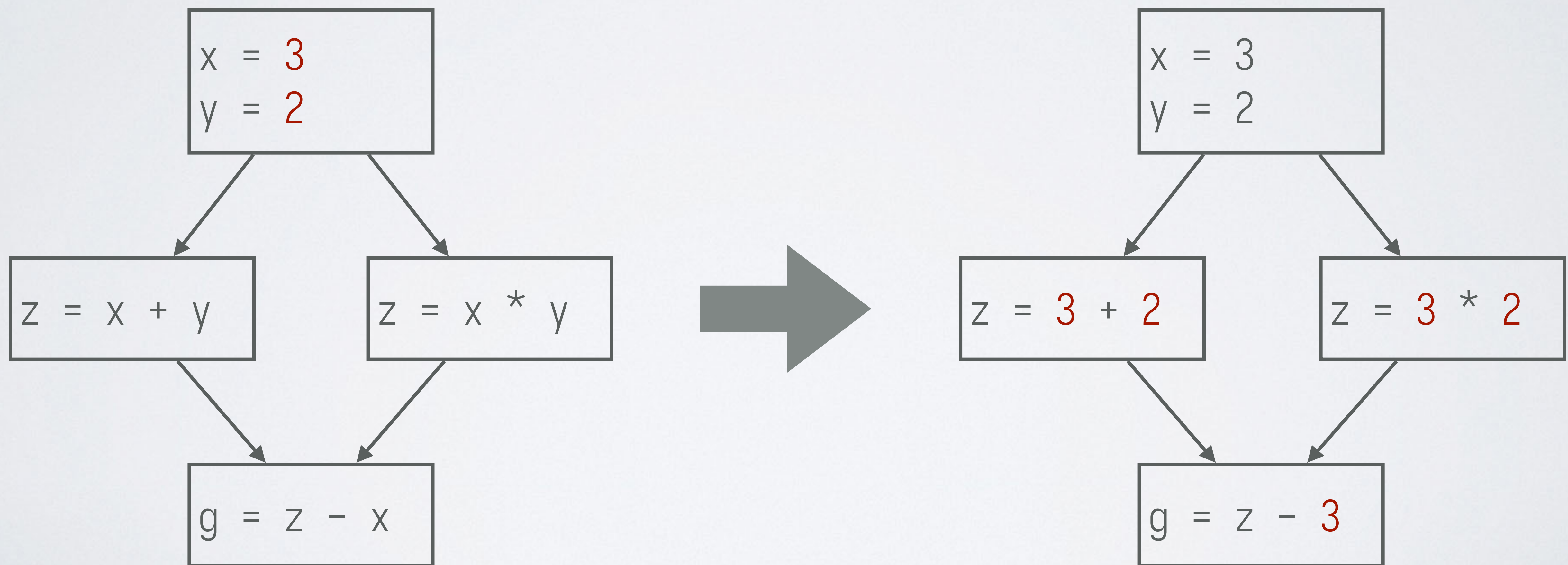




```
goto (BB1)
```

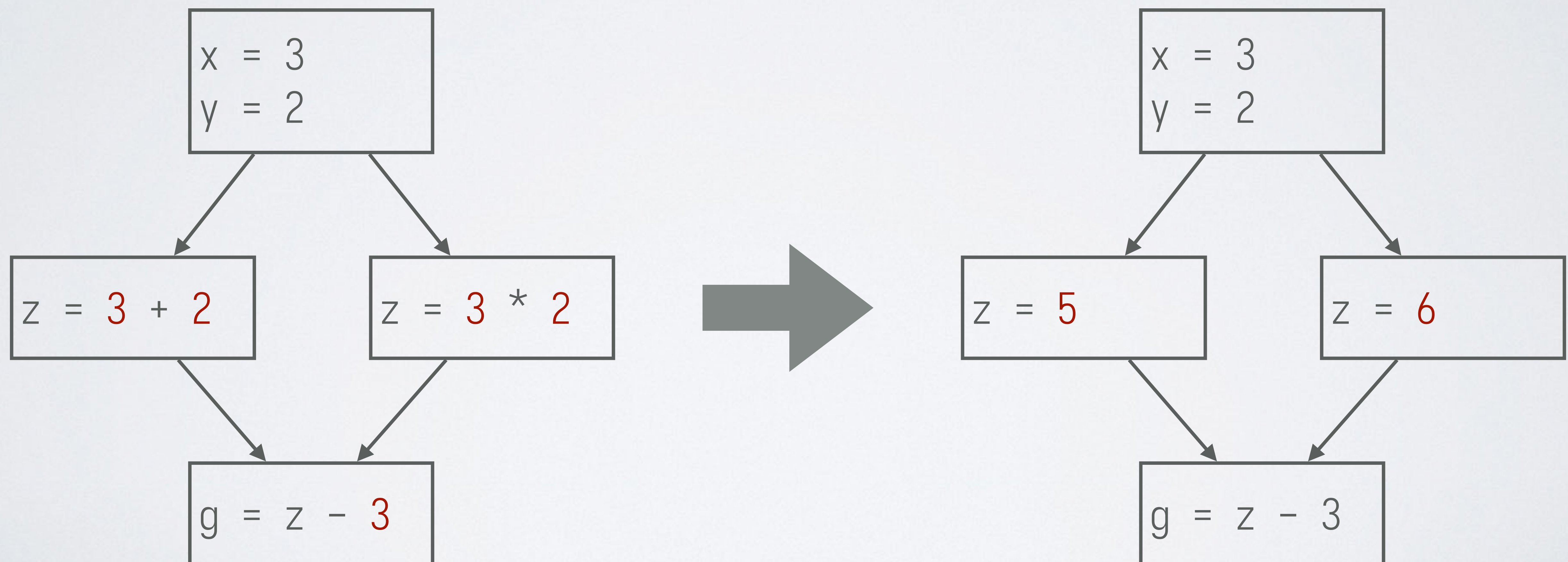
常量传播

- constant propagation
- 如果编译时刻推导出一个变量 x 的值是常量，就可以把用到 x 的地方换成该常量值



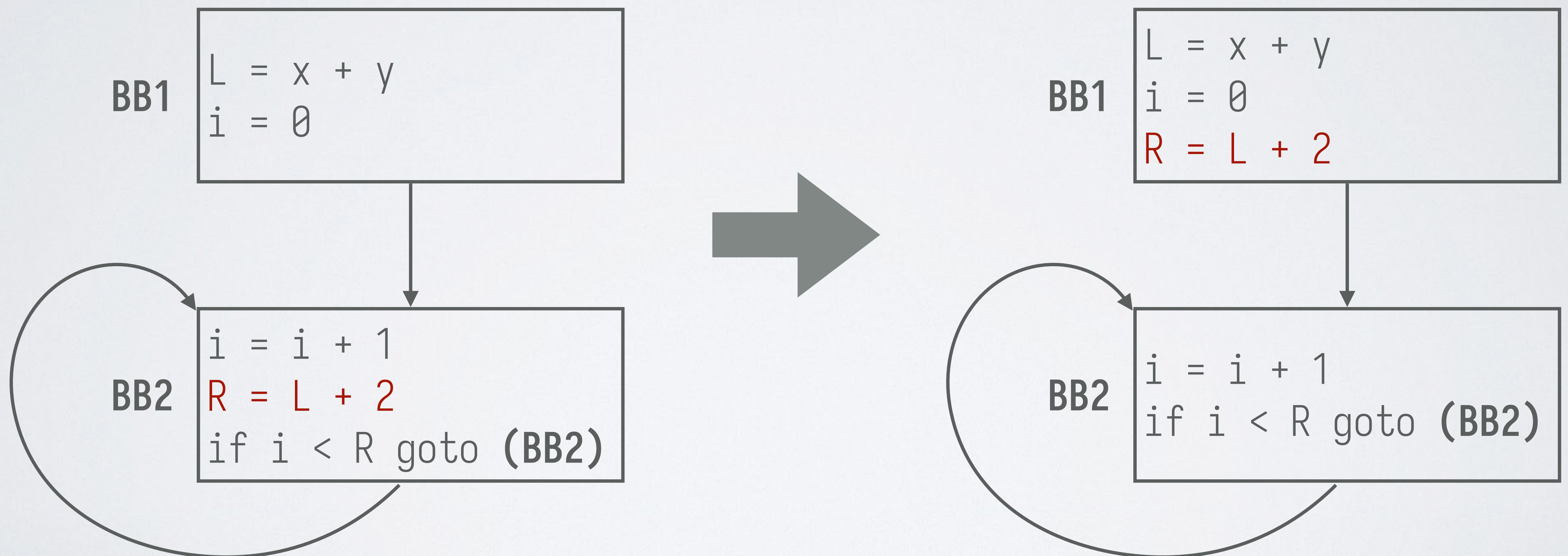
常量折叠

- constant folding
- 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



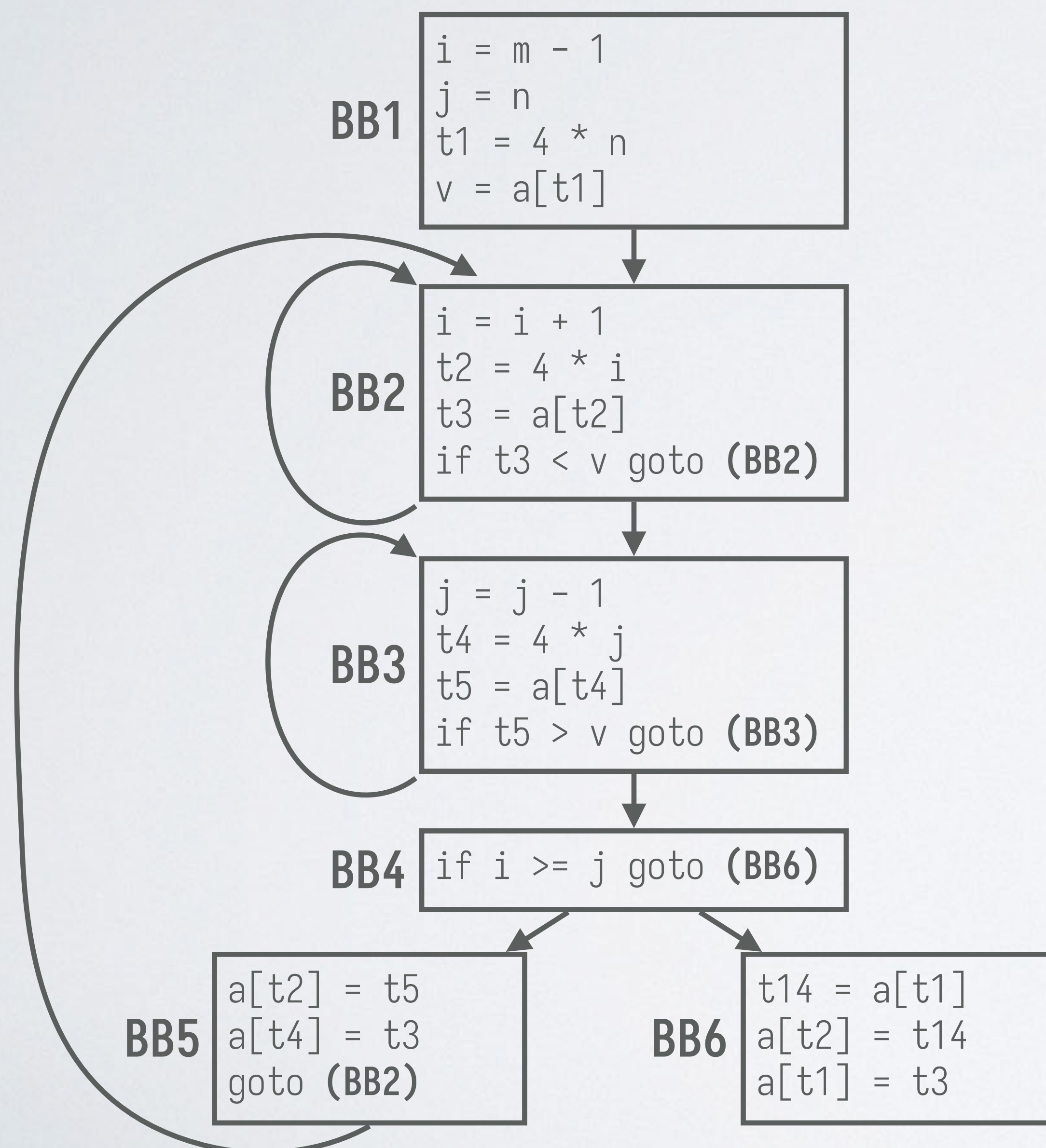
代码外提

- 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）

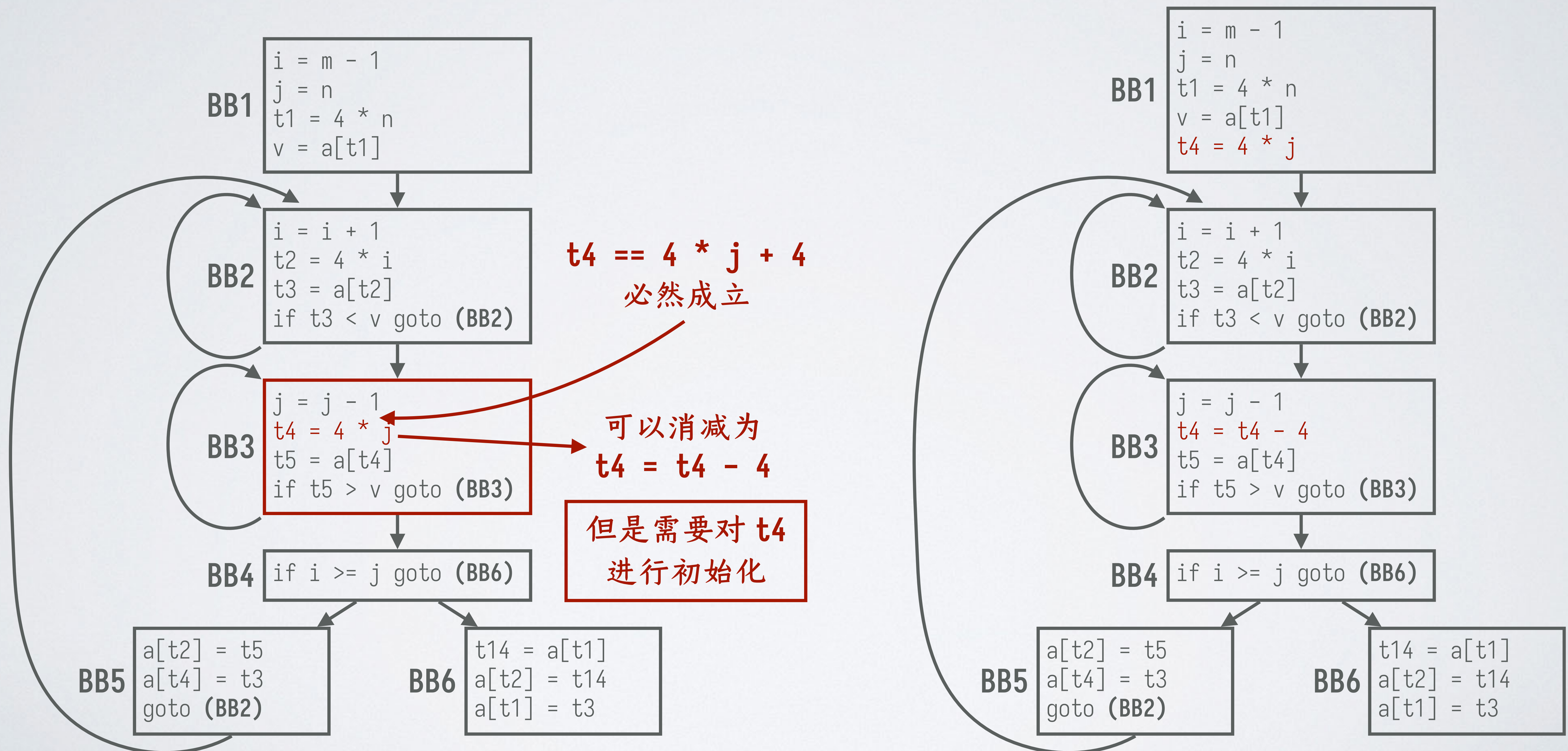
- **归纳变量(induction variable)**：循环中每次 x 被赋值时总是增加一个常数，则称 x 是该循环的一个归纳变量



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

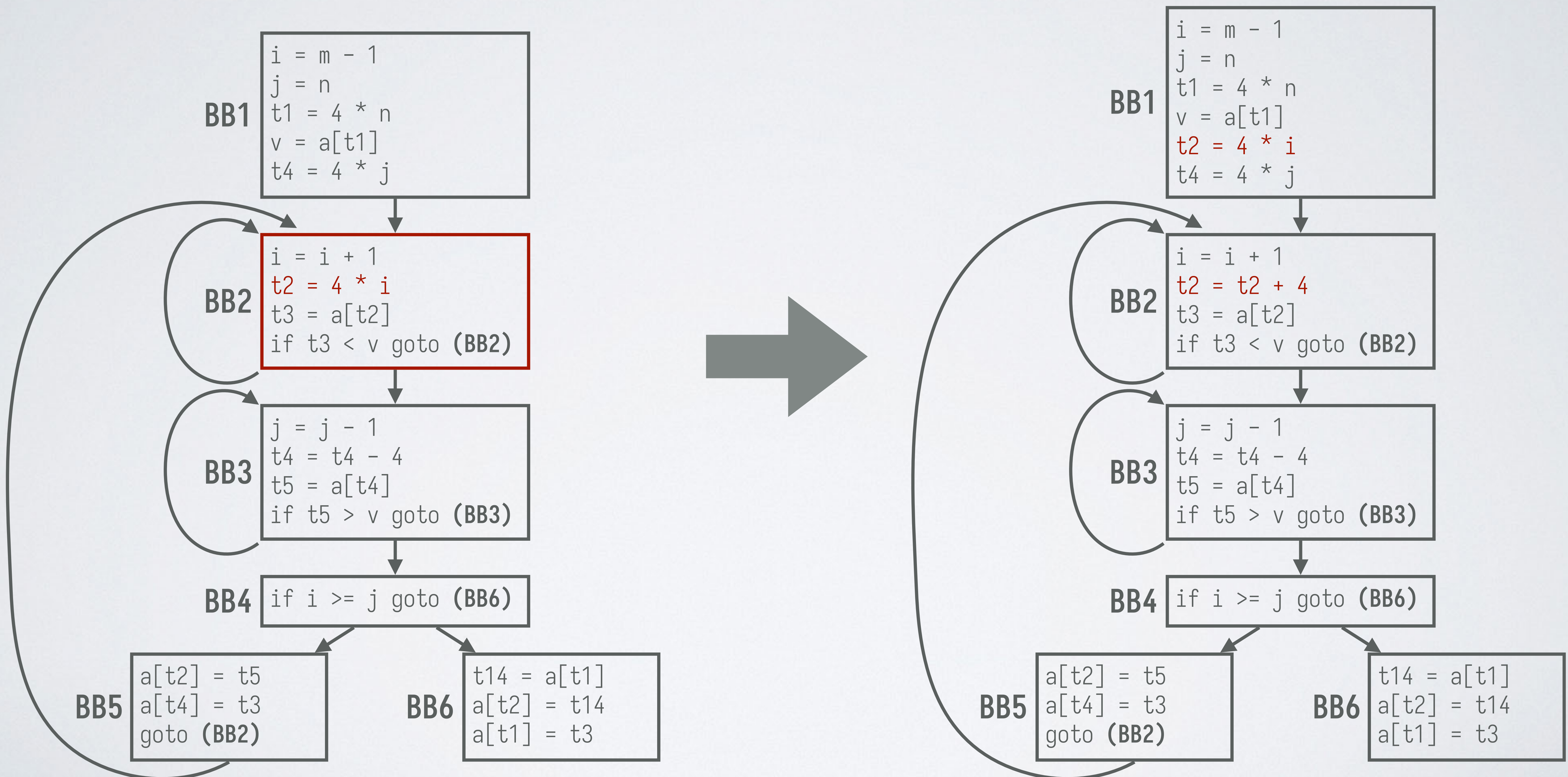
强度消减（2）

- 处理循环时，按照「从里到外」的方式进行工作



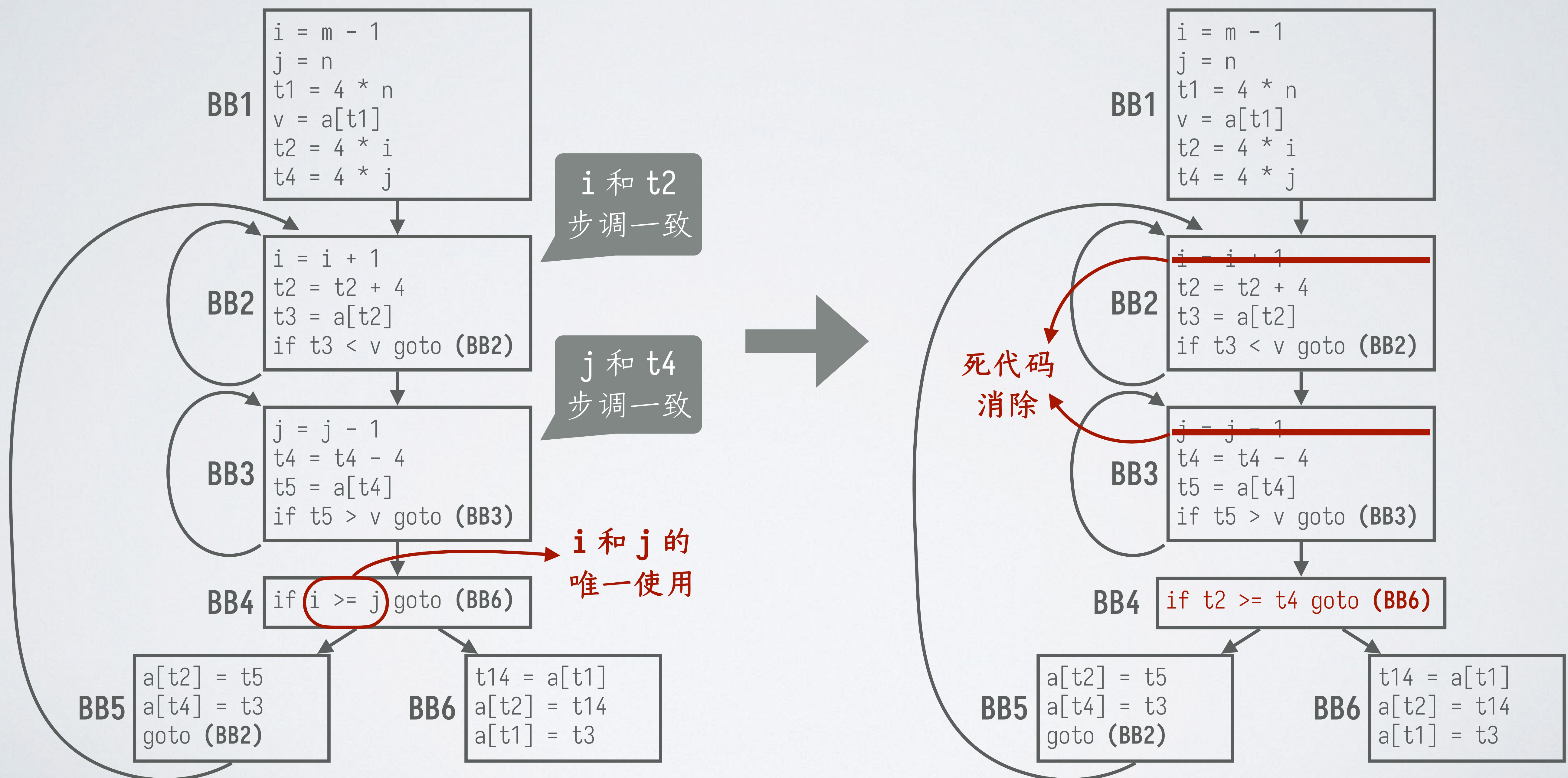
强度消减（3）

- 处理循环时，按照「从里到外」的方式进行工作

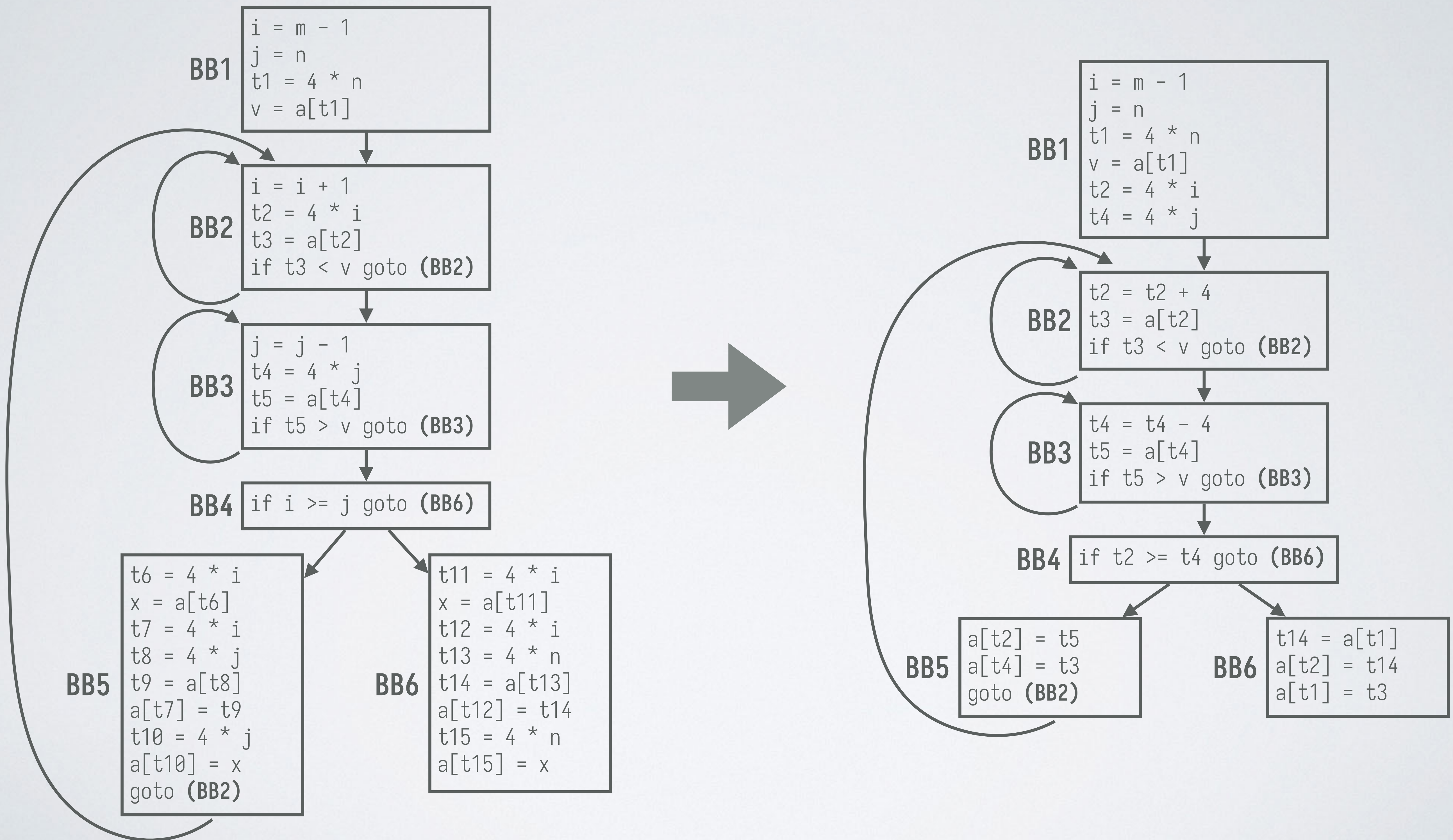


归纳变量消除

- 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



主要内容

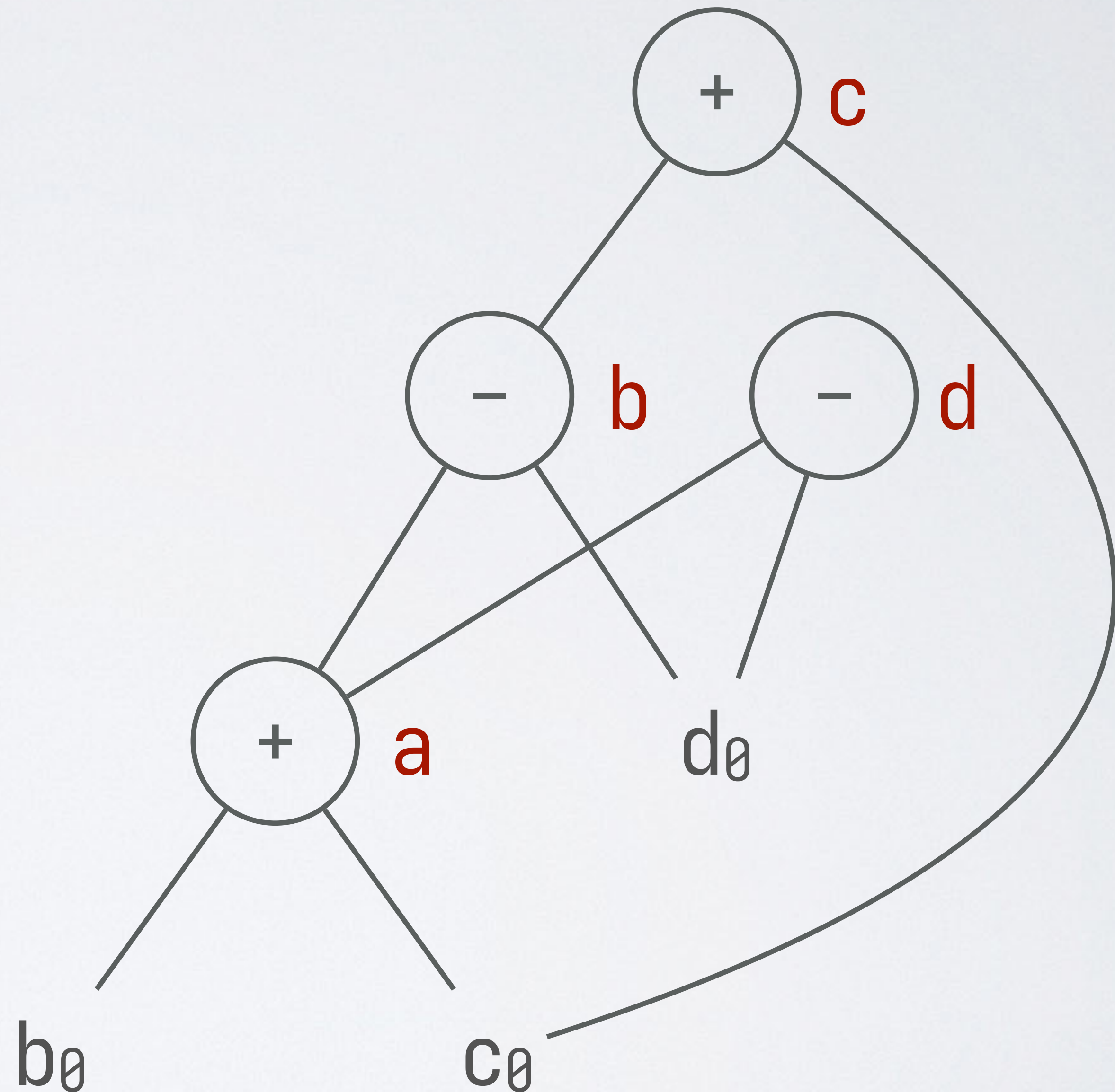
- 代码优化的常用方法
- **局部优化：基本块的优化**
- 局部优化：窥孔优化
- 全局优化：数据流分析

基本块的优化

- 对各个基本块进行**局部**优化
  - 回顾: **全局**优化指的是跨基本块的优化
- 基本块可以用**有向无环图(DAG)**表示
  - 每个变量对应 DAG 的一个结点, 代表其初值
  - 每个语句 s 对应一个结点 N, 代表语句计算得到的值
    - N 的孩子结点对应(其运算分量当前值)的其它语句
    - N 的标号是 s 的运算符
    - N 和一组变量关联, 表示 s 是在基本块内最晚对它们定值的语句
  - **输出结点:** 结点关联的变量在基本块出口处活跃
- DAG 描述了各个变量最后值与初始值的关系

基本块 DAG 的例子

```
a = b + c
b = a - d
c = b + c
d = a - d
```

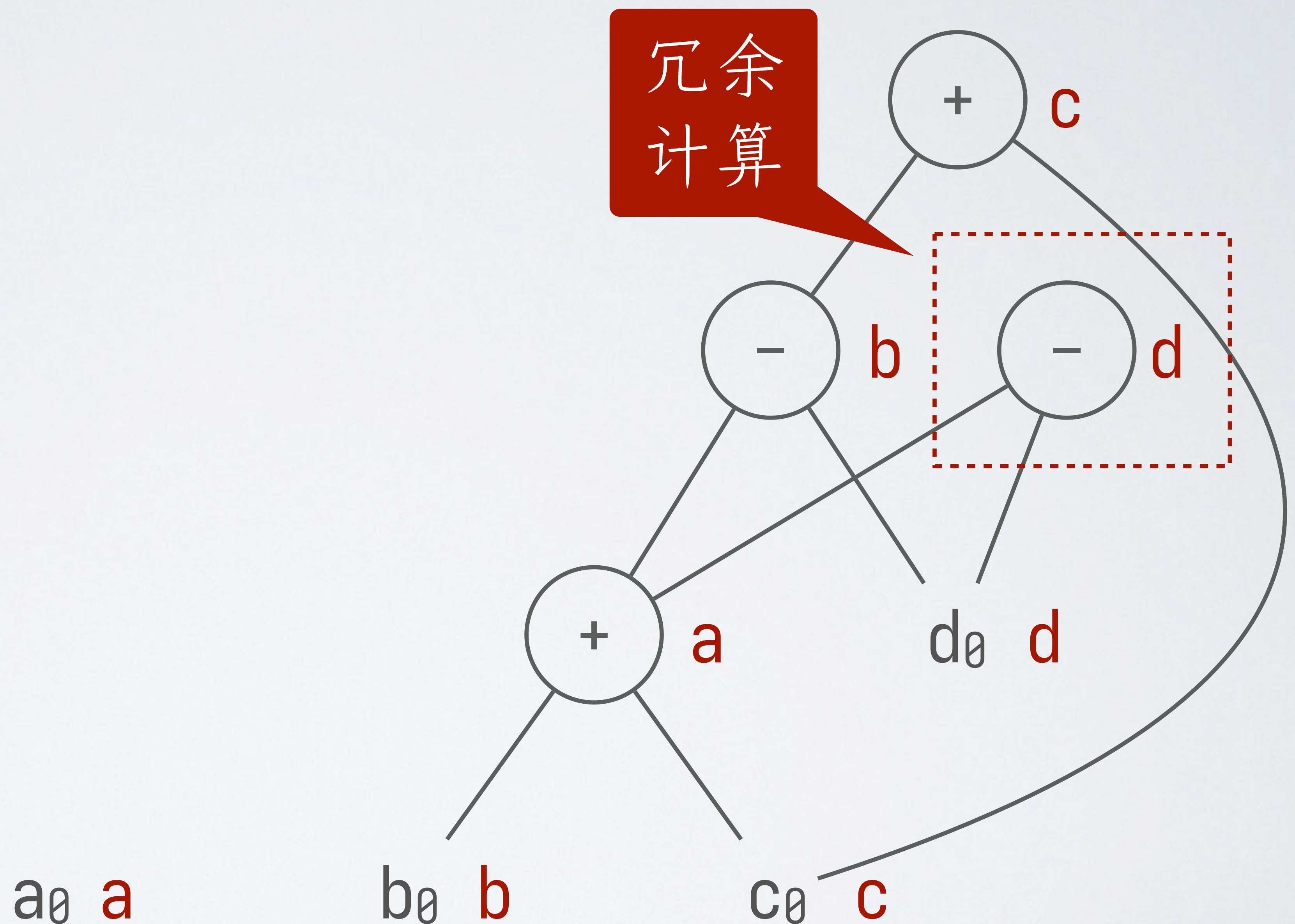


基本块 DAG 的构造算法

- 为基本块中出现的每个变量建立结点(表示**初始值**)，各变量和对应结点关联(表示**最后定值**)
- 顺序扫描每个三地址语句，进行如下处理：
  - 语句为 $x = y\ op\ z$:
    - 为这个语句建立结点 N, 标号为 op
    - N 的孩子结点为 y、z 当前关联的结点
    - 令 x 和 N 关联
  - 语句为 $x = y$:
    - 不建立新结点
    - 若 y 关联到 N, 那么 x 现在也关联到 N
- 扫描结束后，对于所有在出口处活跃的变量 x, 把 x 关联的结点设置为**输出结点**

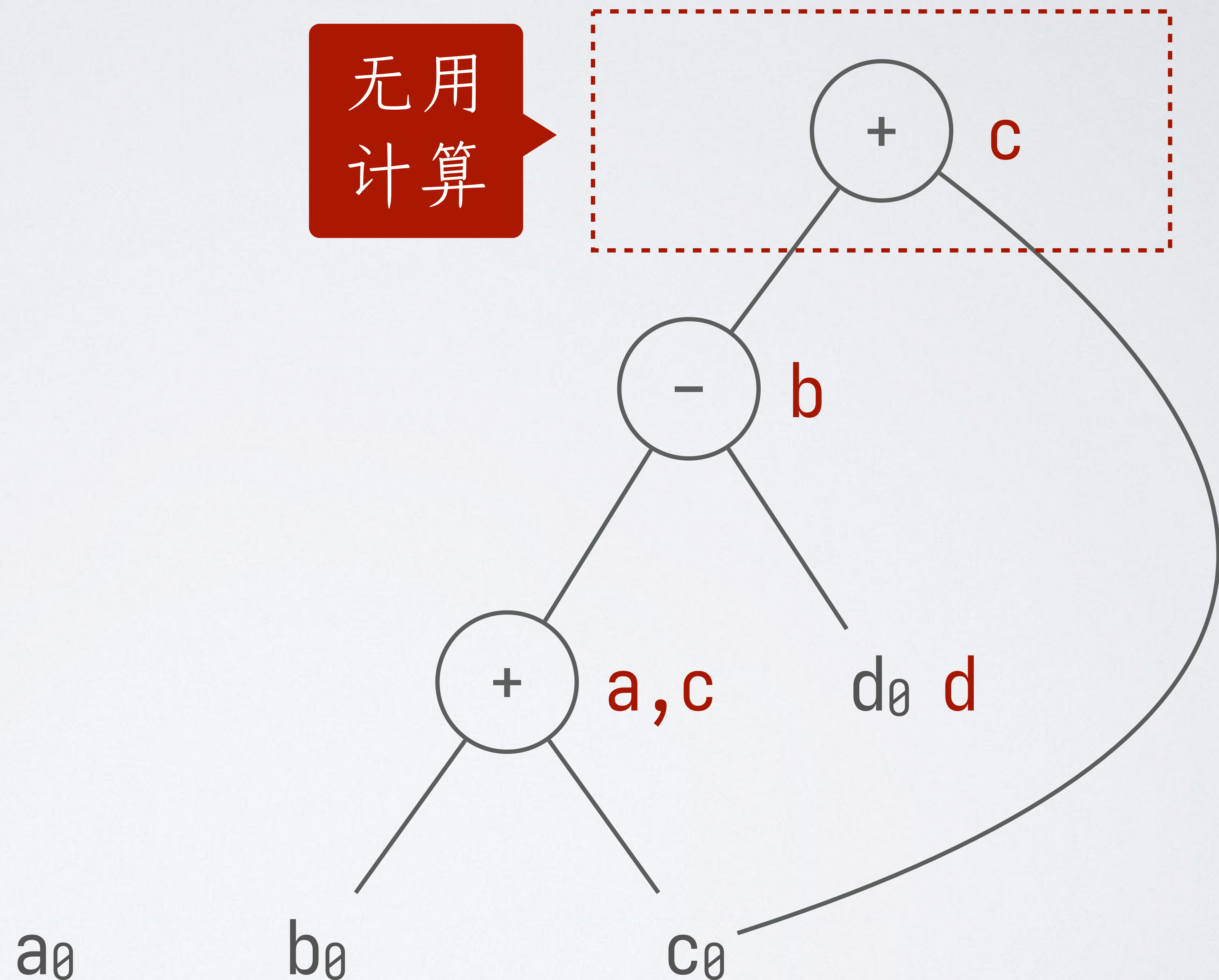
基本块 DAG 构造的例子（1）

```
a = b + c
b = a - d
c = b + c
d = a - d
```



基本块 DAG 构造的例子（2）

```
a = b + c
b = a - d
c = b + c
c = a
```



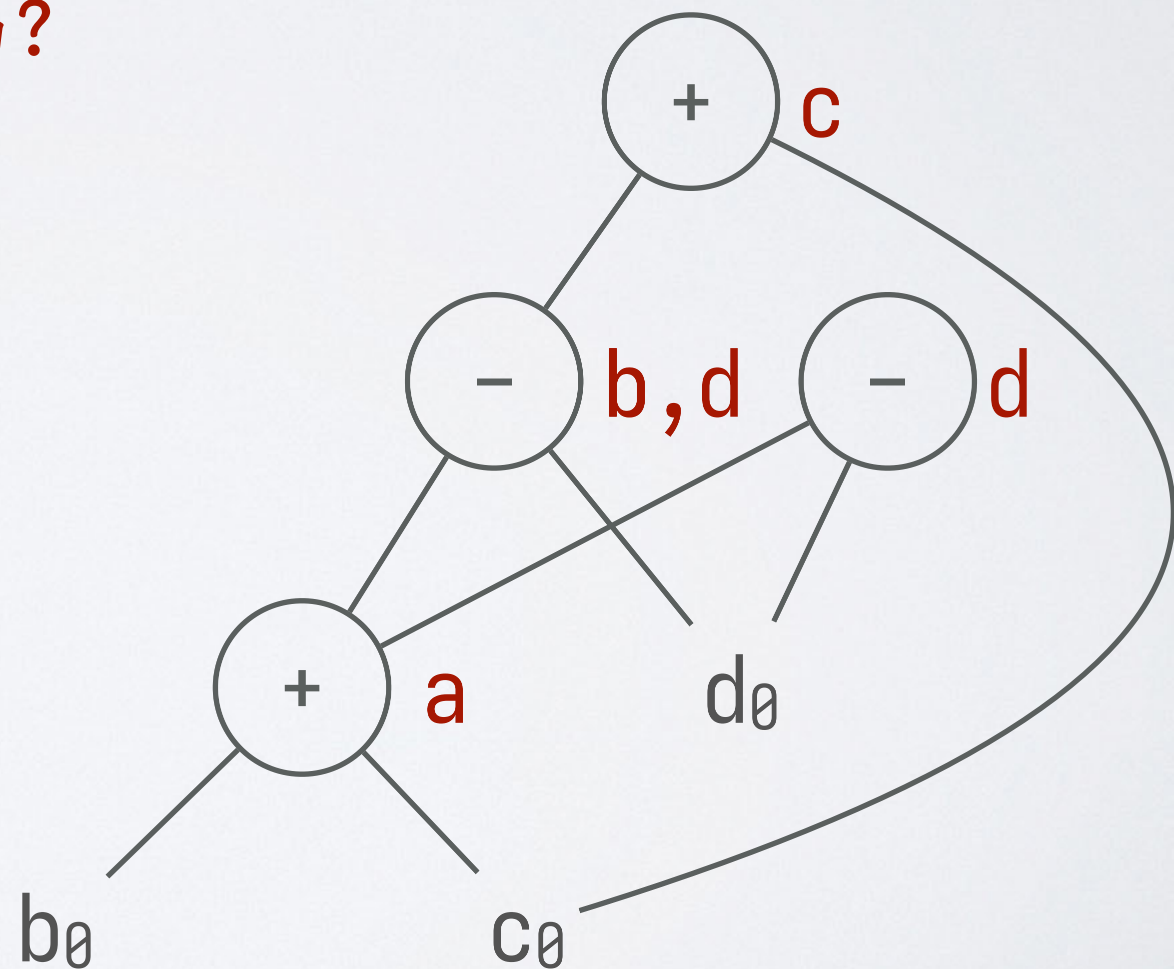
DAG 的作用

- DAG 描述了各个变量最后值与初始值的关系
- 以 DAG 为基础，可以对代码进行变换和优化
  - 消除局部公共子表达式
  - 消除死代码
  - 使用代数恒等式简化计算

消除局部公共子表达式

- 寻找**局部公共子表达式(local common subexpression)**
  - 建立某个结点 M 之前，首先检查是否存在一个结点 N，它和 M 具有相同的运算符和孩子结点(顺序也相同)
  - 如果存在，则不需要生成新的结点，用 N 表示 M
  - **问：b + c 是公共子表达式吗？**

```
a = b + c
b = a - d
c = b + c
d = a - d
```

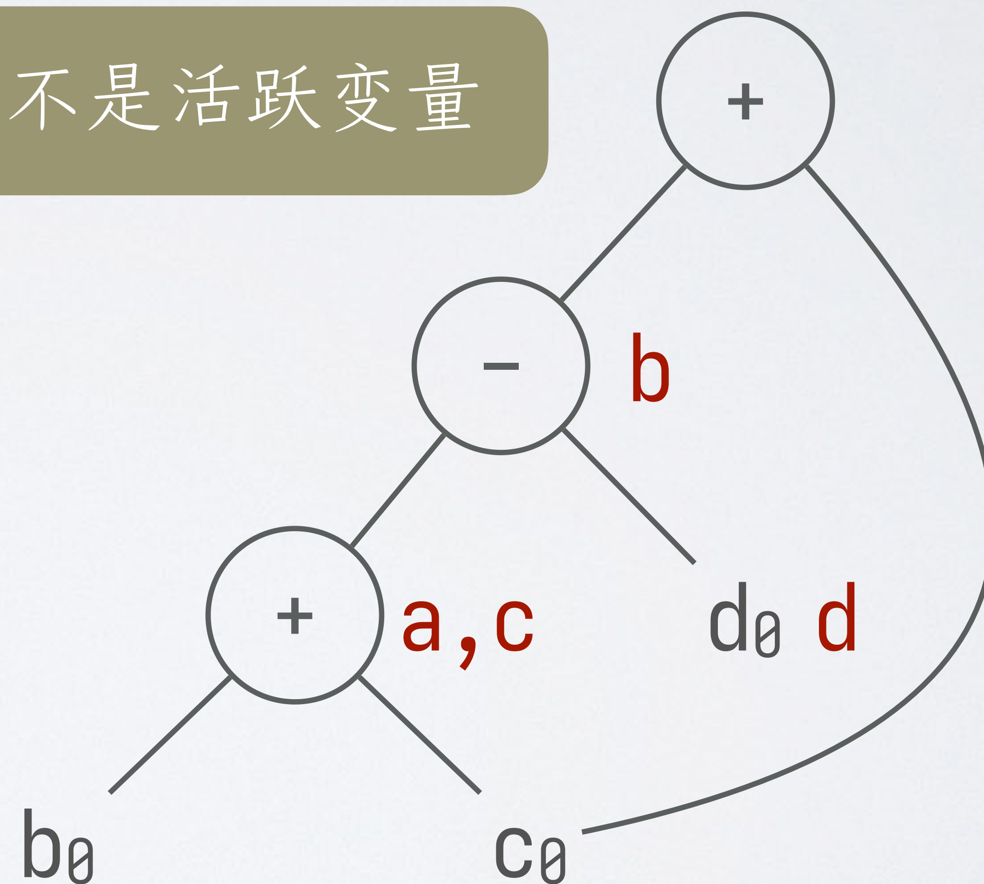


消除死代码

- 消除**死代码**(dead code)
  - 在 DAG 上消除没有附加活跃变量的根结点
  - 重复这一处理过程直到没有结点能再被消除

```
a = b + c
b = a - d
c = b + c
c = a
```

如果 b 不是活跃变量



使用代数恒等式简化计算

- **代数恒等式(algebraic identity)**
- 消除计算步骤
  - 例如：$x + 0 = 0 + x = x, x \times 1 = 1 \times x = x, x - 0 = x, x/1 = x$
- 强度消减(reduction in strength)
  - 例如：$x^2 = x \times x, 2 \times x = x + x, x/2 = x \times 0.5$
- 常量折叠(constant folding)
  - 例如：表达式 `2 * 3.14` 可以被替换为 `6.28`
- 实现这些优化时，只需要在 DAG 上寻找特定的模式

数组引用的注意事项

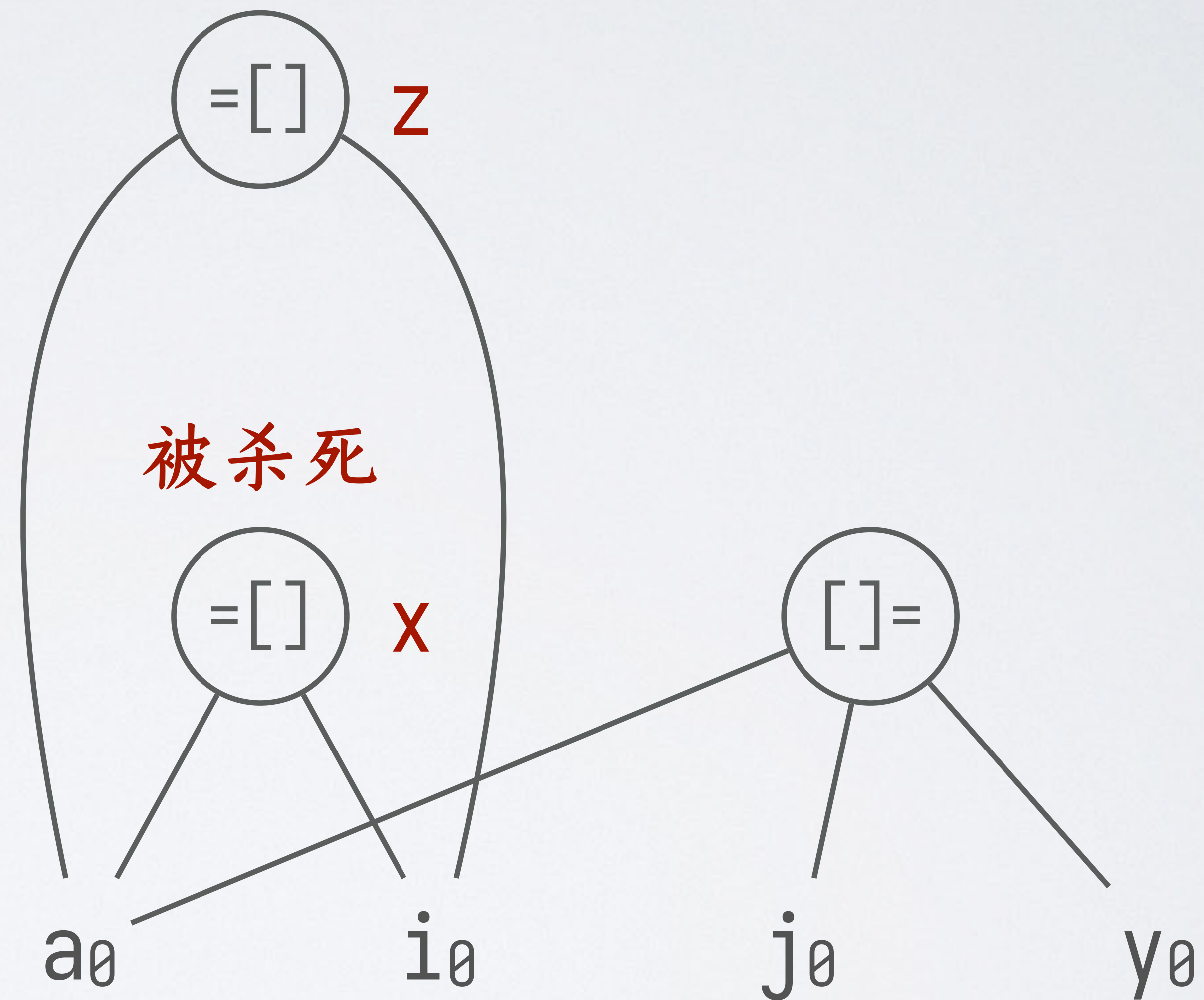
- **问题：下列代码中，a[i] 是公共子表达式吗？**

```
x = a[i]
a[j] = y
z = a[i]
```

- a[i] 和 a[j] 可能存在**别名(alias)**关系，对 a[j] 的赋值可能改变 a[i] 的值
- 数组取值的运算 x = a[i] 对应于一个运算符为 **=[]** 的结点，其孩子结点为 a、i，而变量 x 作为关联变量
- 数组赋值的运算 a[j] = y 对应于一个运算符为 **[]=** 的结点，其孩子结点为 a、j、y，并**杀死**所有依赖于 a 的结点

数组引用 DAG 的例子

```
x = a[i]
a[j] = y
z = a[i]
```

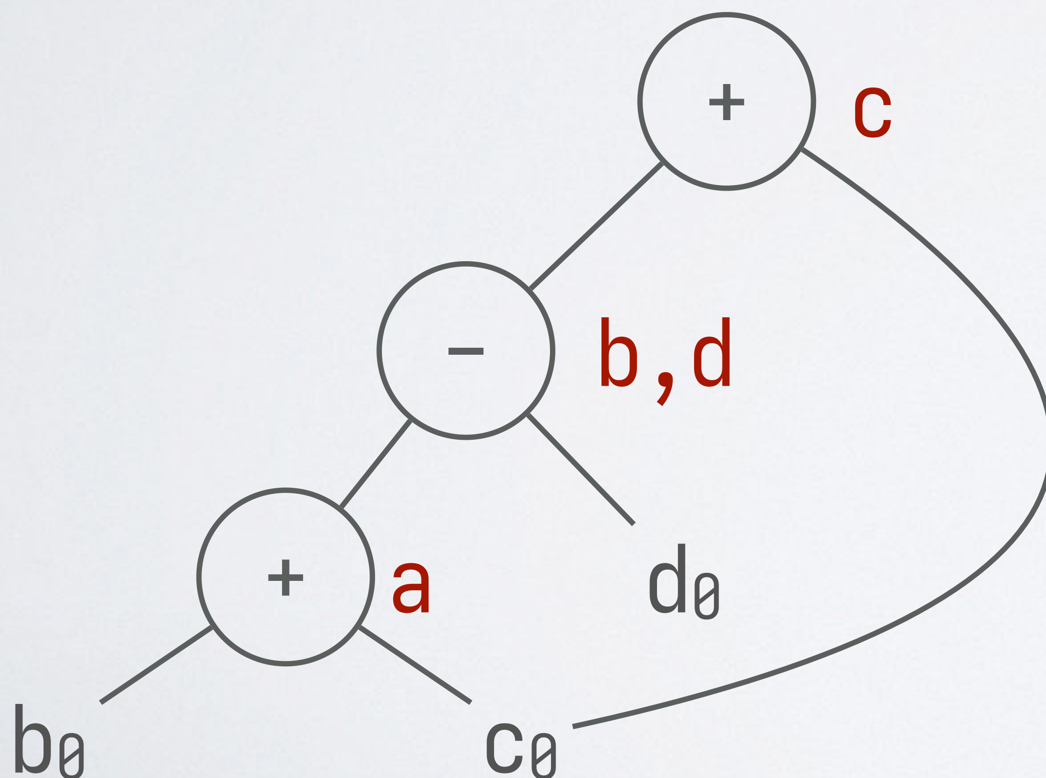


指针赋值和过程调用

- **问题：除了数组以外，还有哪些行为能够杀死结点？**
- 原则：对程序行为加以**保守的假设**
- 指针赋值
  - 代码 **x = *p; *q = y** 中，编译器并不知道 p 和 q 是否指向同一位置
  - 保守假设：
    - x = *p 可能使用任意变量，因此会影响死代码的消除
    - *q = y 可能对任意变量赋值，因此会杀死所有其它结点
  - 可以通过**指针分析(pointer analysis)**部分解决这个问题
- 过程调用
  - 保守假设：一个过程调用使用和改变了所有它能访问的数据

从 DAG 到基本块的重组

- **方法：**
  - 为每个结点构造一个三地址语句，计算对应的值
  - 结果应该尽量赋值给一个活跃变量
  - 如果结点有多个关联的变量，则用复制语句进行赋值
  - 处理数组引用、指针赋值、过程调用时，要**注意生成语句的顺序**



如果 b、c、d 在出口处活跃	如果 c、d 在出口处活跃
a = b + c d = a - d b = d c = d + c	a = b + c d = a - d c = d + c

主要内容

- 代码优化的常用方法
- 局部优化：基本块的优化
- **局部优化：窥孔优化**
- 全局优化：数据流分析

窥孔优化（peephole optimization）

- 使用一个滑动窗口(即**窥孔, peephole**)来检查目标指令，在窥孔内用更短或更快的指令来替换窗口中的指令序列
  - 也可以在中间代码上进行
- 常见的窥孔优化:
  - 冗余指令消除
  - 控制流优化
  - 代数化简
  - 机器特有指令的使用

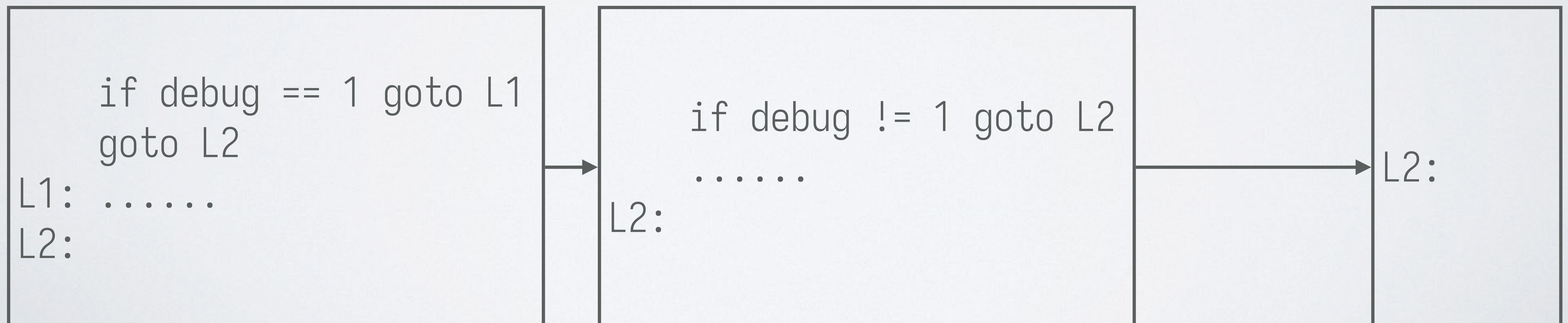
消除冗余指令

- 多余的 LD 和 ST 指令

没有指令跳转到第二条指令处

- 不可达代码

如果已知 debug 一定是 0

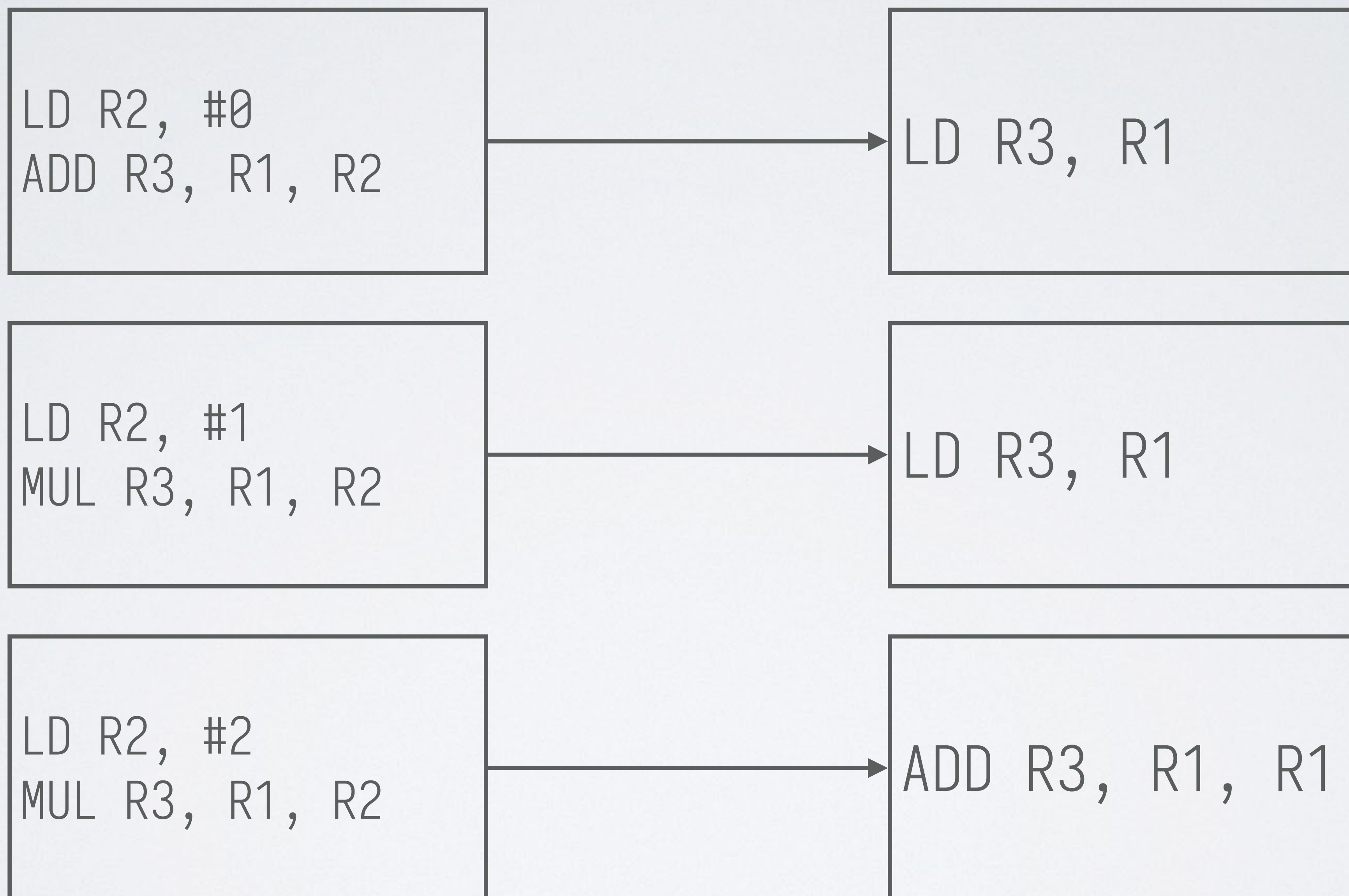


控制流优化

如果只有一个到达 L1 的跳转

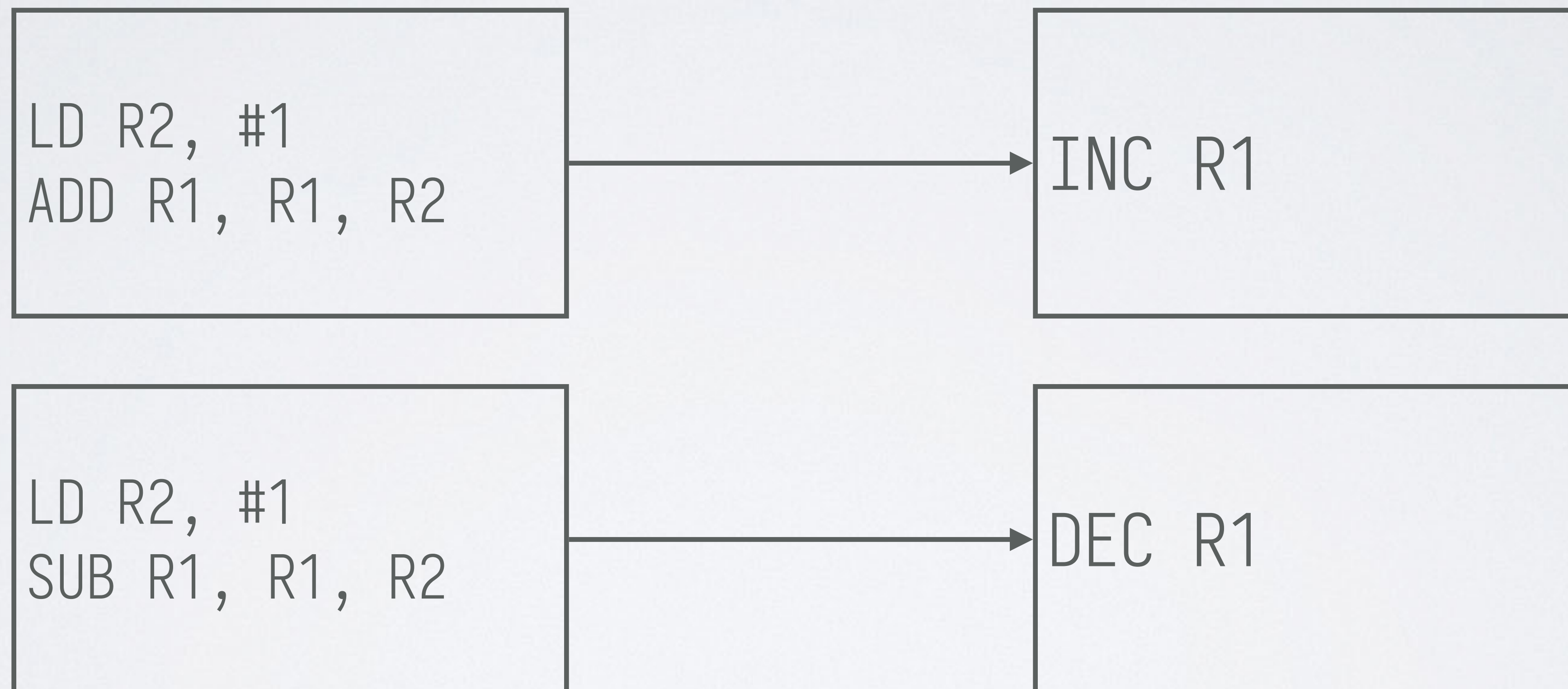
代数化简

- 类似于使用代数恒等式简化计算



机器特有指令的使用

- CISC 指令集中，实现一个运算的指令(序列)可能有多种
- 比如，假设有 INC 和 DEC 指令完成「加一」和「减一」运算



主要内容

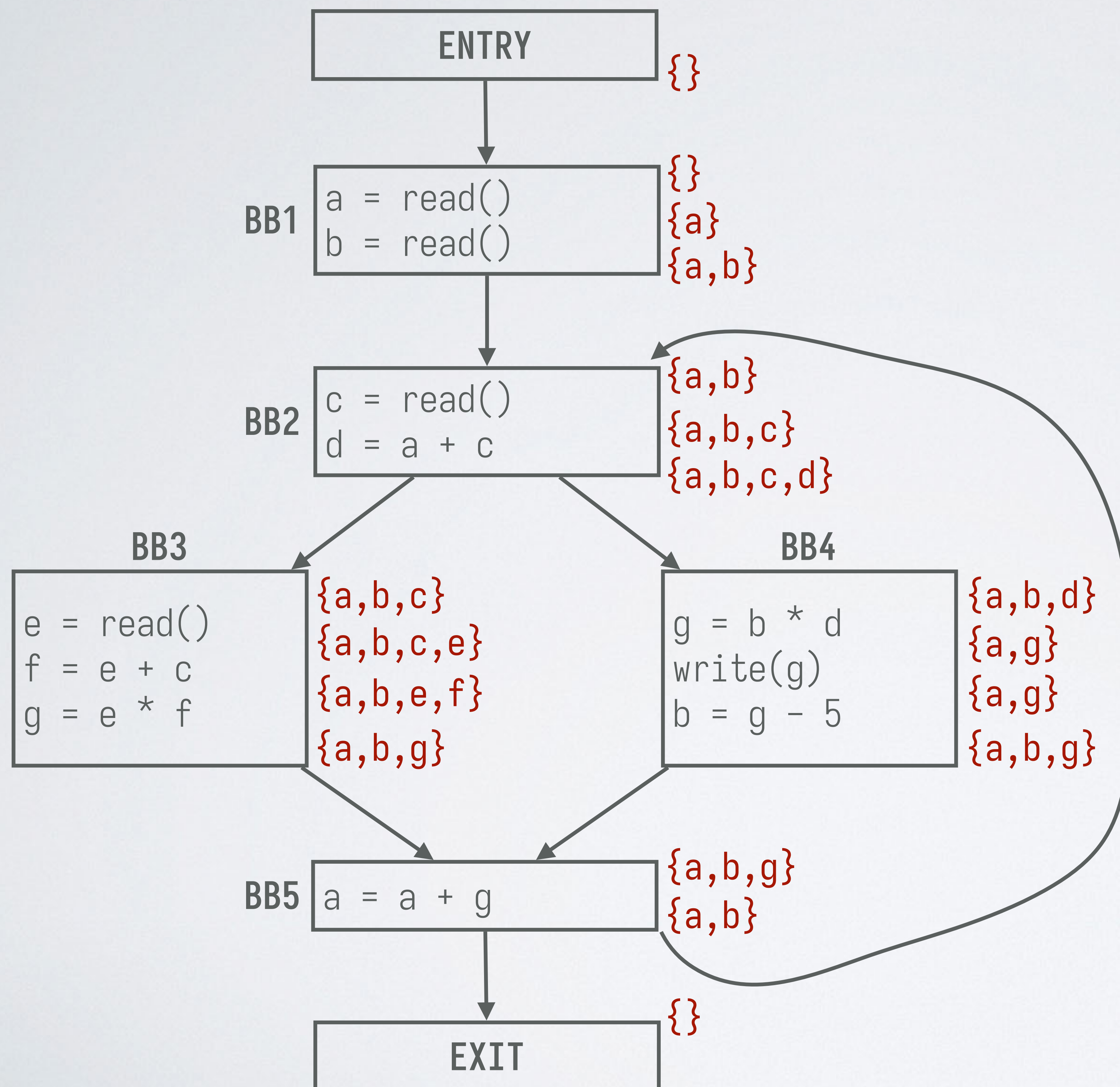
- 代码优化的常用方法
- 局部优化：基本块的优化
- 局部优化：窥孔优化
- **全局优化：数据流分析**

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？

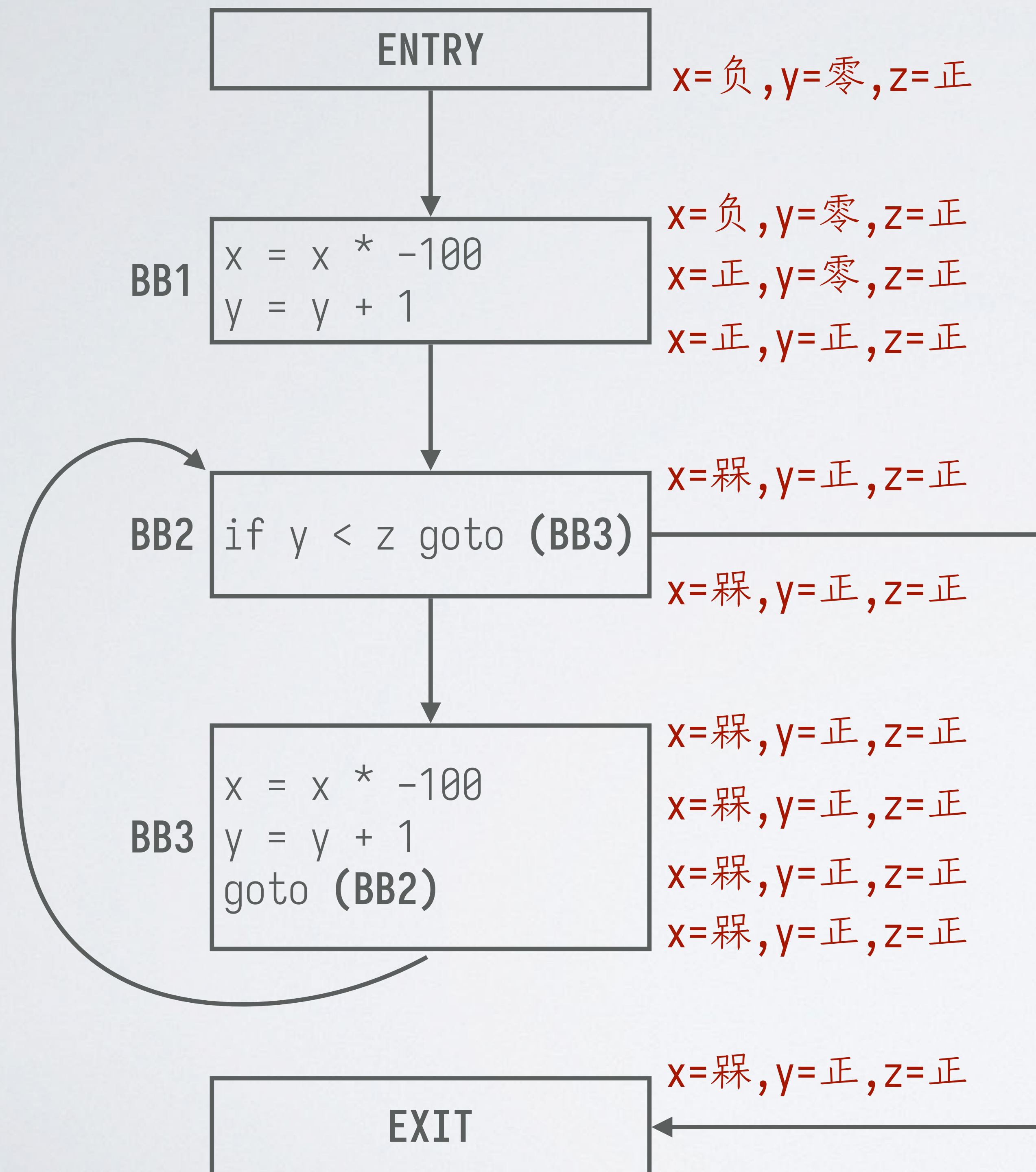
- 回顾：局部优化中的分析
  - 分析每条语句的作用
  - 把多条语句的作用组合起来
  - 推导出每条语句前或后的信息
- **数据流分析是一种全局分析**
  - 分析每个**基本块**的作用
  - 把多个**基本块**的作用组合起来
  - 推导出在每个**基本块入口或出口处**的信息
    - 基于这些信息，在基本块内使用局部分析推导出语句粒度的信息
  - 通过这些信息来进行全局优化

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**的程序执行完后**可能**具备的符号

正 = {所有的正数}

零 = {0}

负 = {所有的负数}

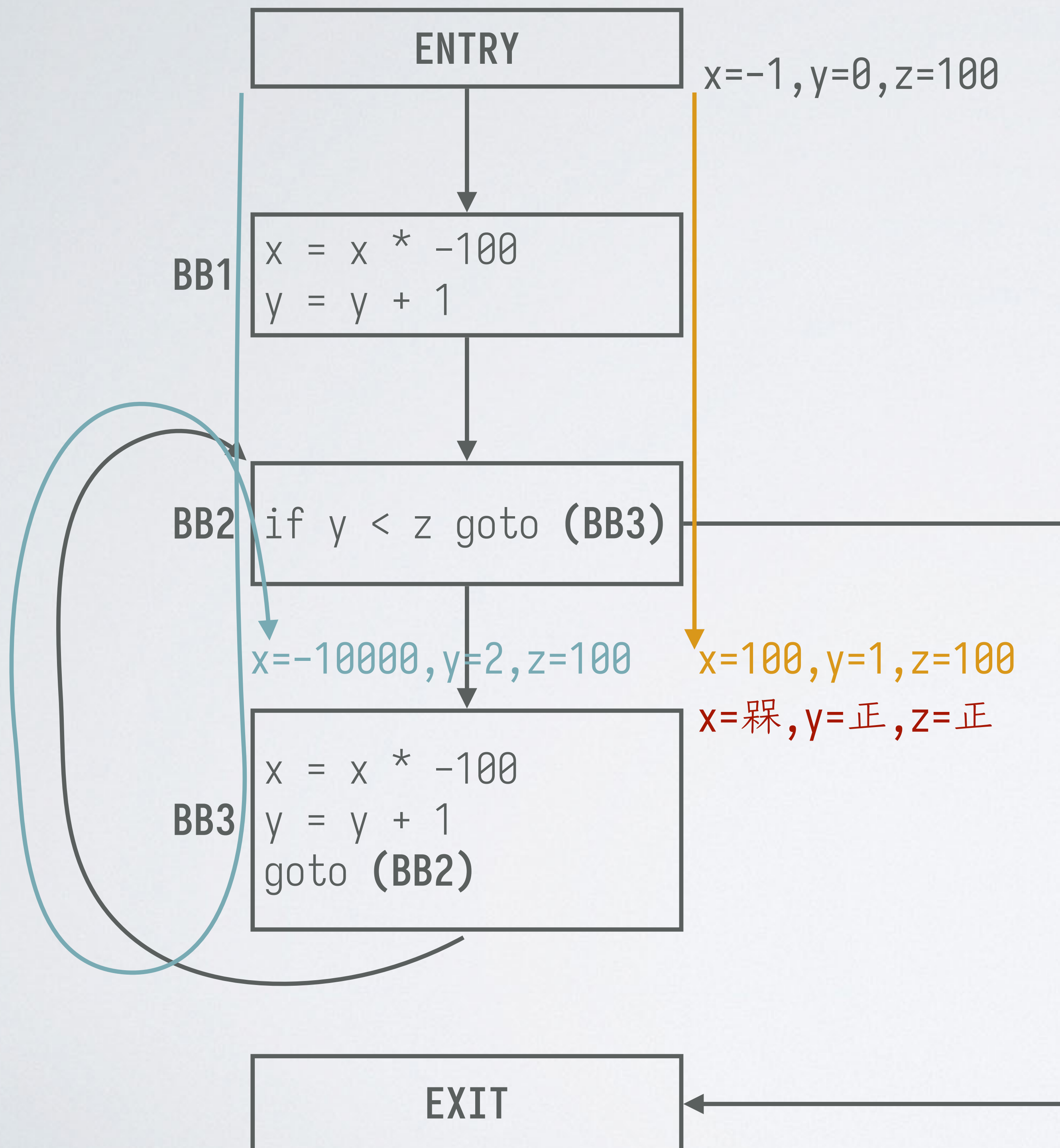
槑 = {所有的数}

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 $1 \leq i < n$:
    - 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

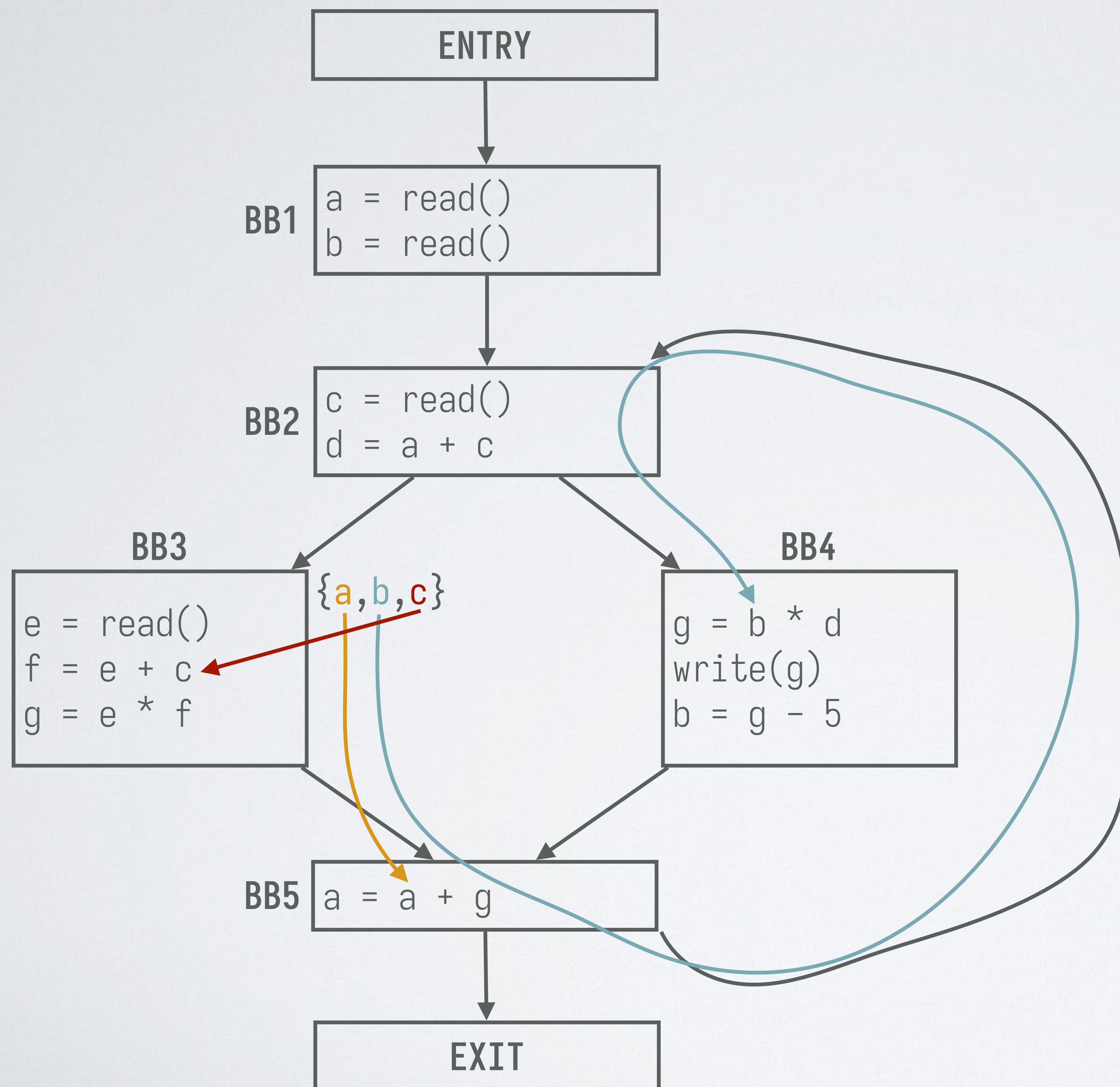
正 = {所有的正数}

零 = {0}

负 = {所有的负数}

槑 = {所有的数}

后向分析的例子：活跃变量

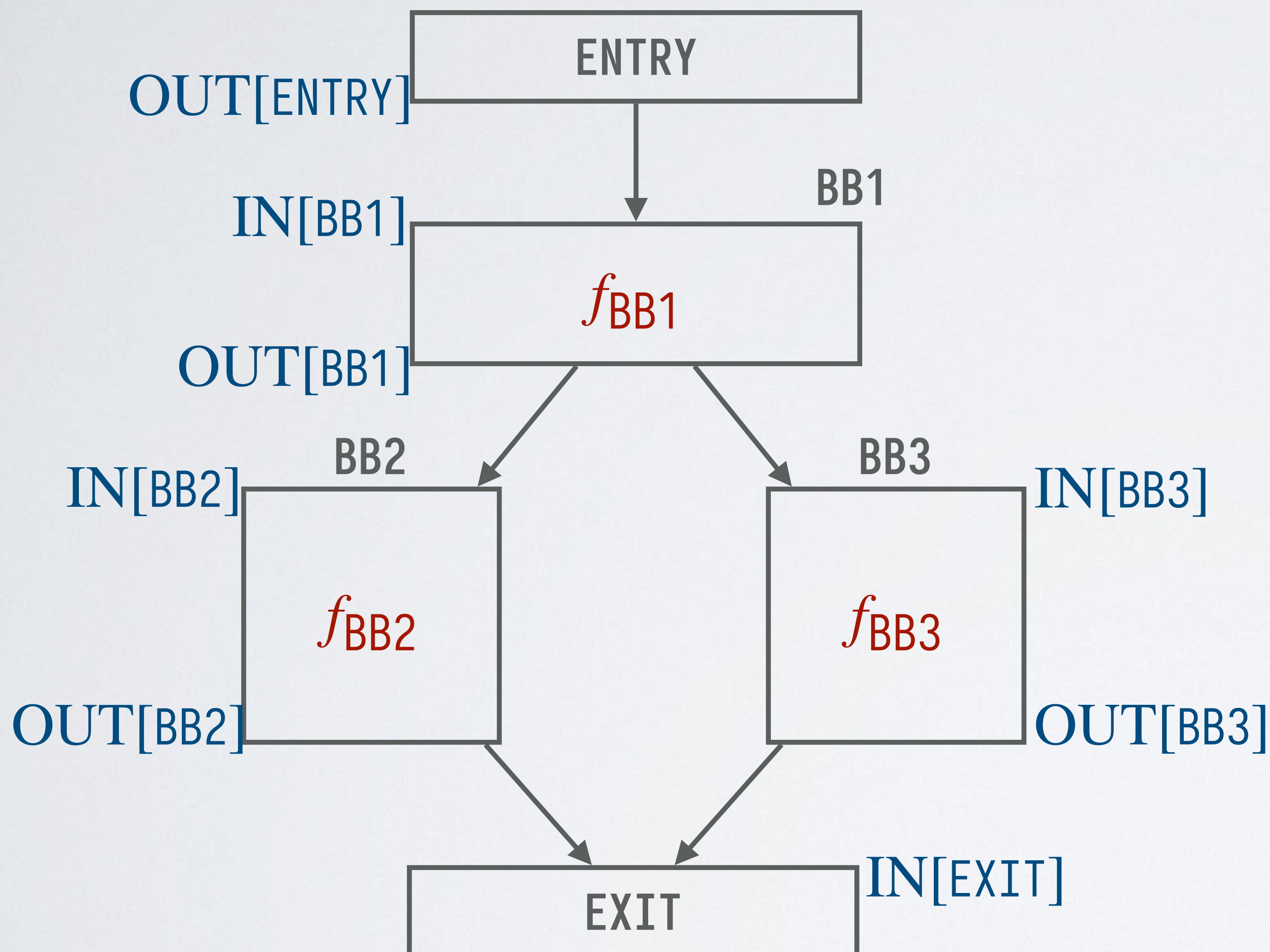


在基本块边界处的信息：哪些变量的值**可能**会在**之后**的程序执行中使用

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」、「槑」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

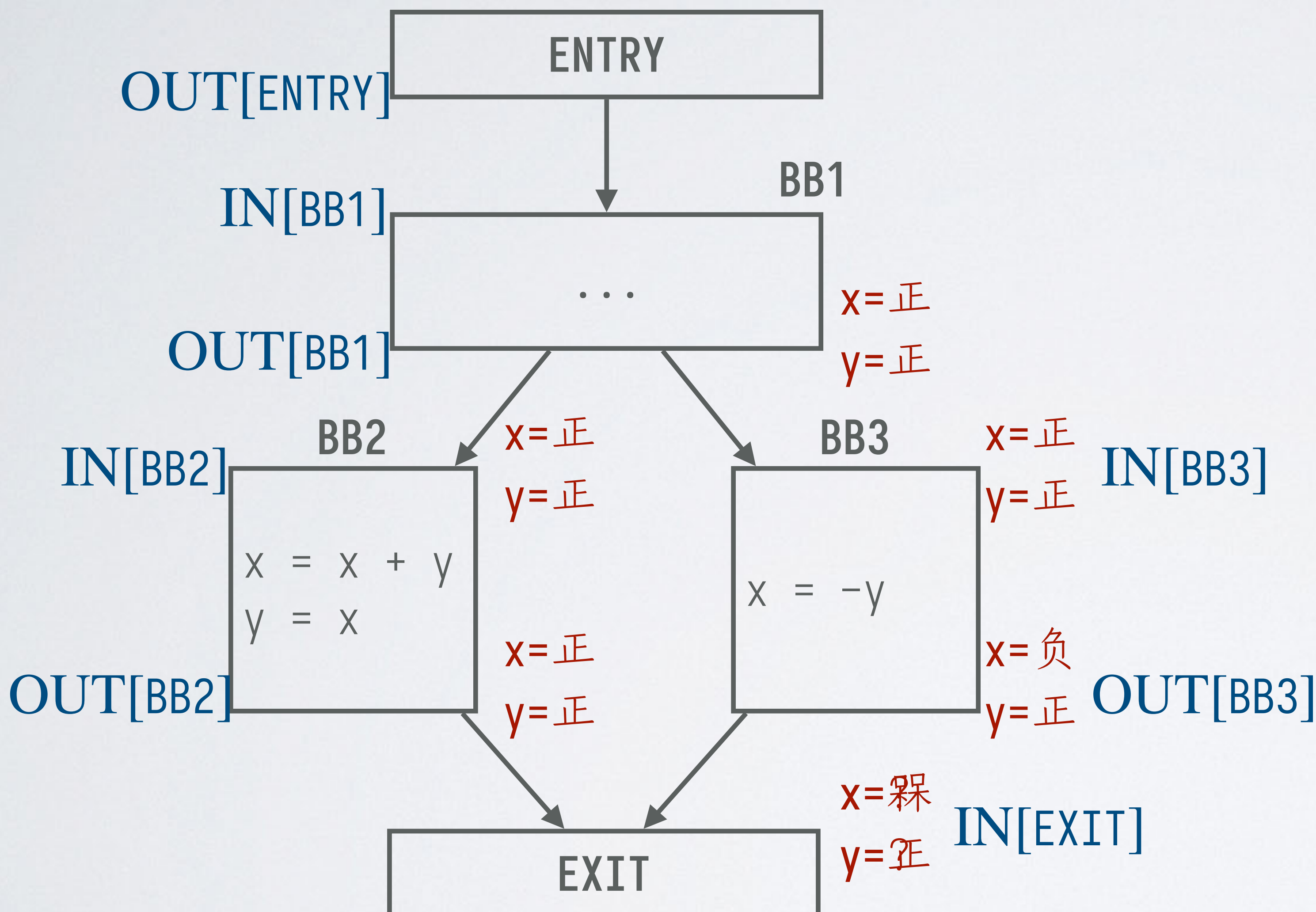
B

```
s1:  x = -y
s2:  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=槑
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=槑 y=负	x=槑 y=负	x=槑 y=负	x=槑 y=负
y=槑	x=槑 y=槑	x=槑 y=槑	x=槑 y=槑	x=槑 y=槑

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** $\wedge$：

  $v_1 \wedge v_2 = v_1$, 若 $v_1 = v_2$

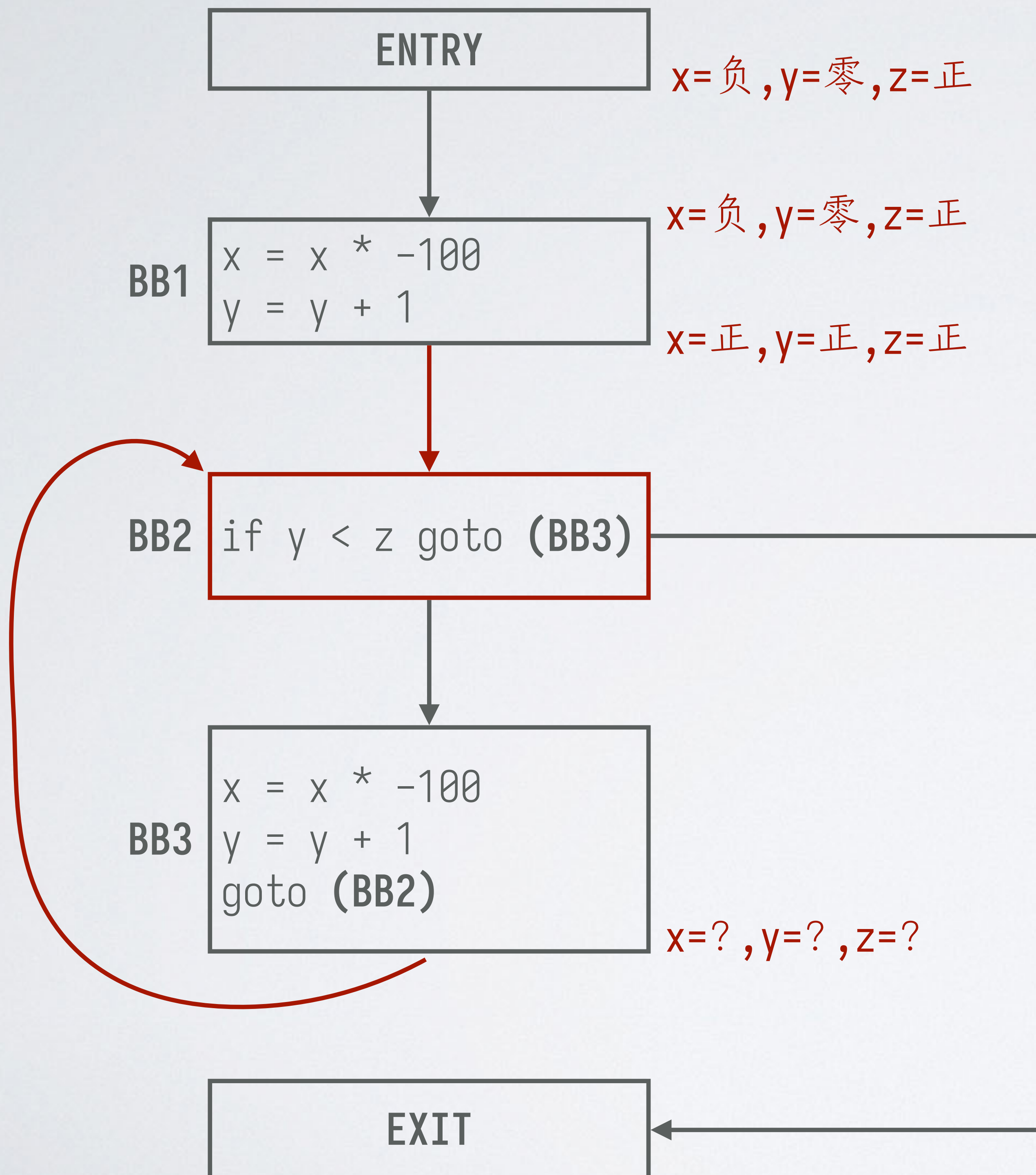
  $v_1 \wedge v_2 =$ 槑, 其它情况

- 分支的处理：

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

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

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：

  $\mathrm{IN[BB2]} = \mathrm{OUT[BB1]} \wedge \mathrm{OUT[BB3]}$

  $\mathrm{OUT[BB2]} = f_{\mathrm{BB2}}(\mathrm{IN[BB2]})$

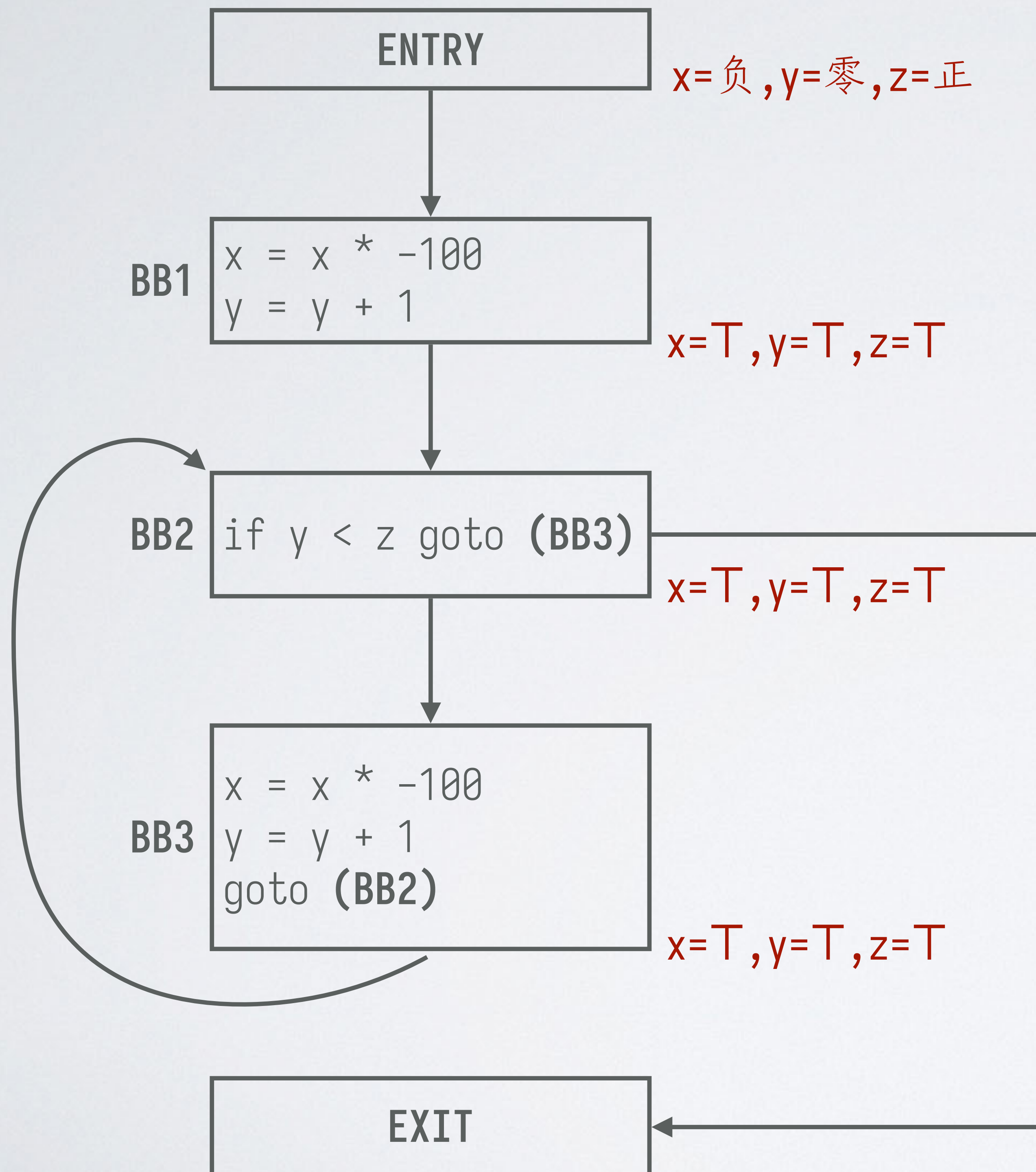
  $\mathrm{IN[BB3]} = \mathrm{OUT[BB2]}$

  $\mathrm{OUT[BB3]} = f_{\mathrm{BB3}}(\mathrm{IN[BB3]})$

  $\mathrm{IN[EXIT]} = \mathrm{OUT[BB2]}$

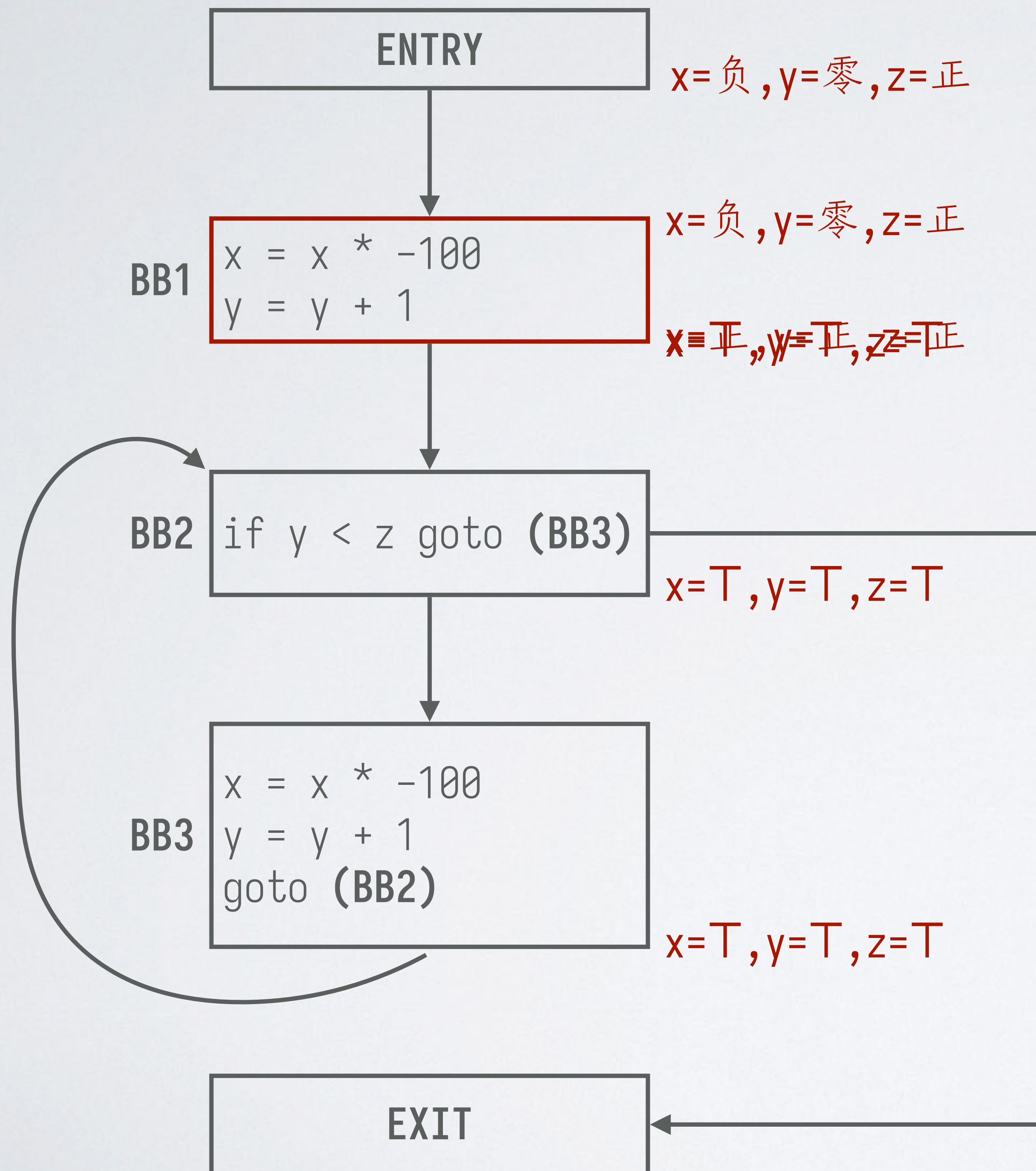
- **目标：找到「最精确」的解**

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）



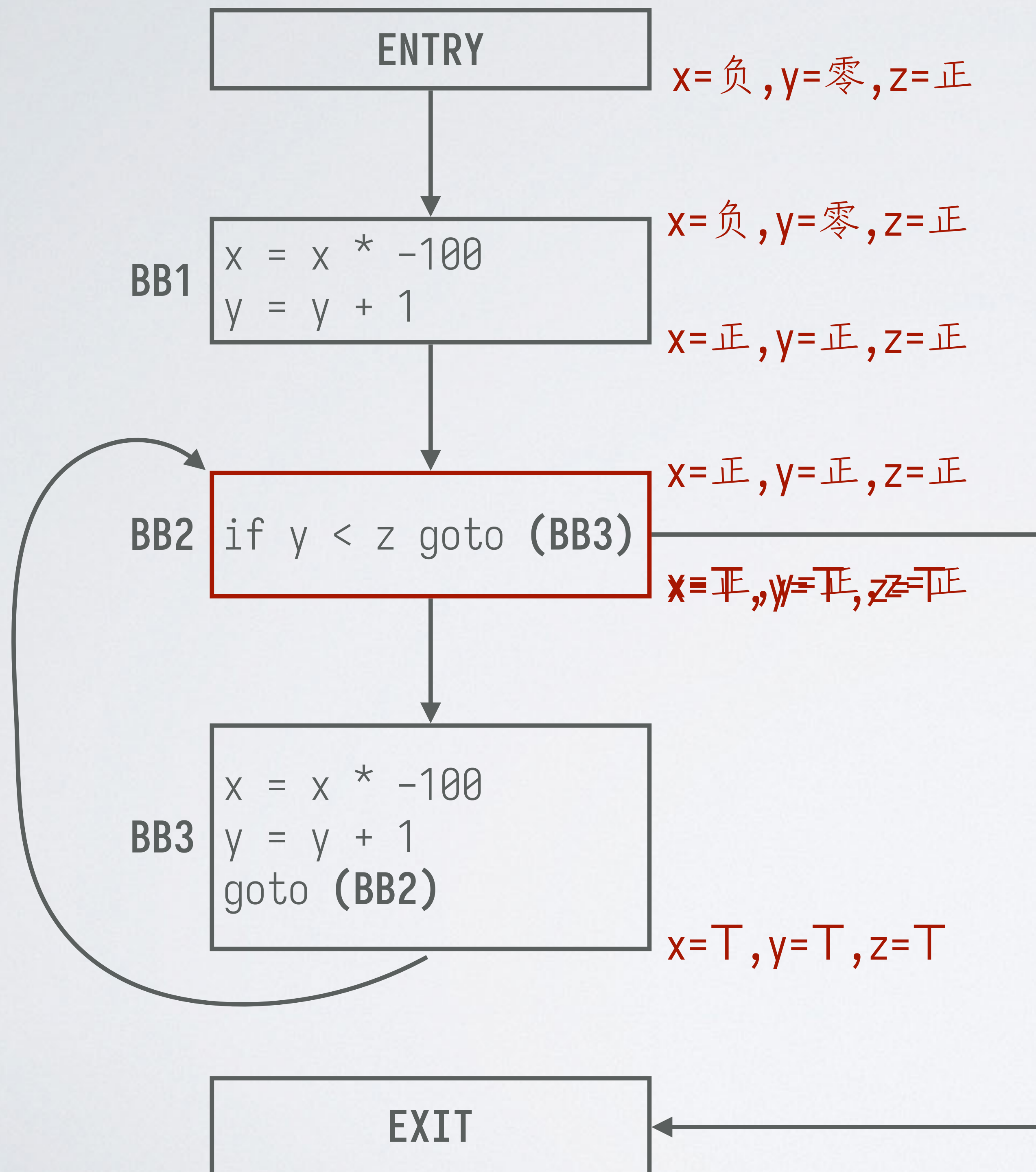
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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）



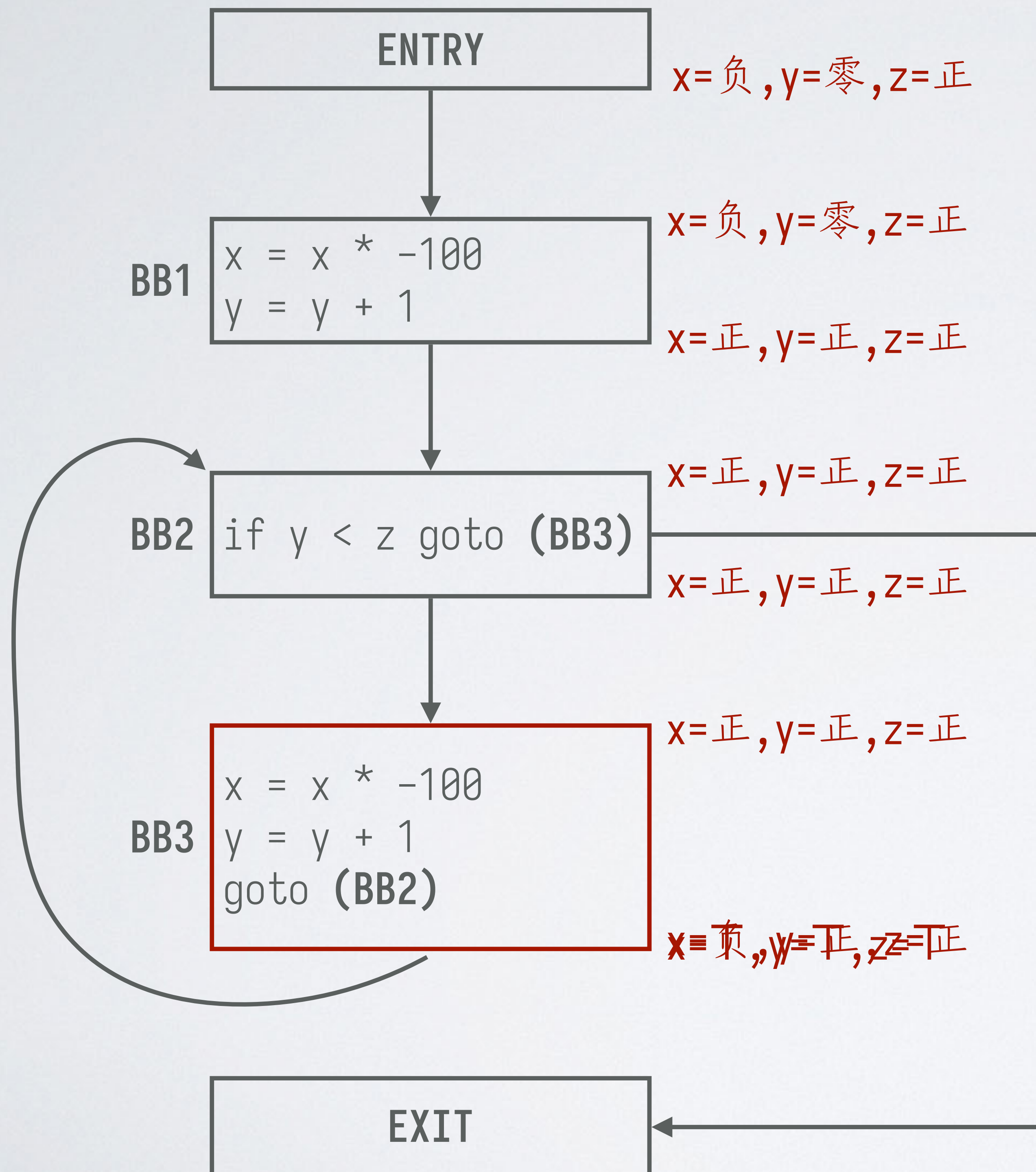
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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）



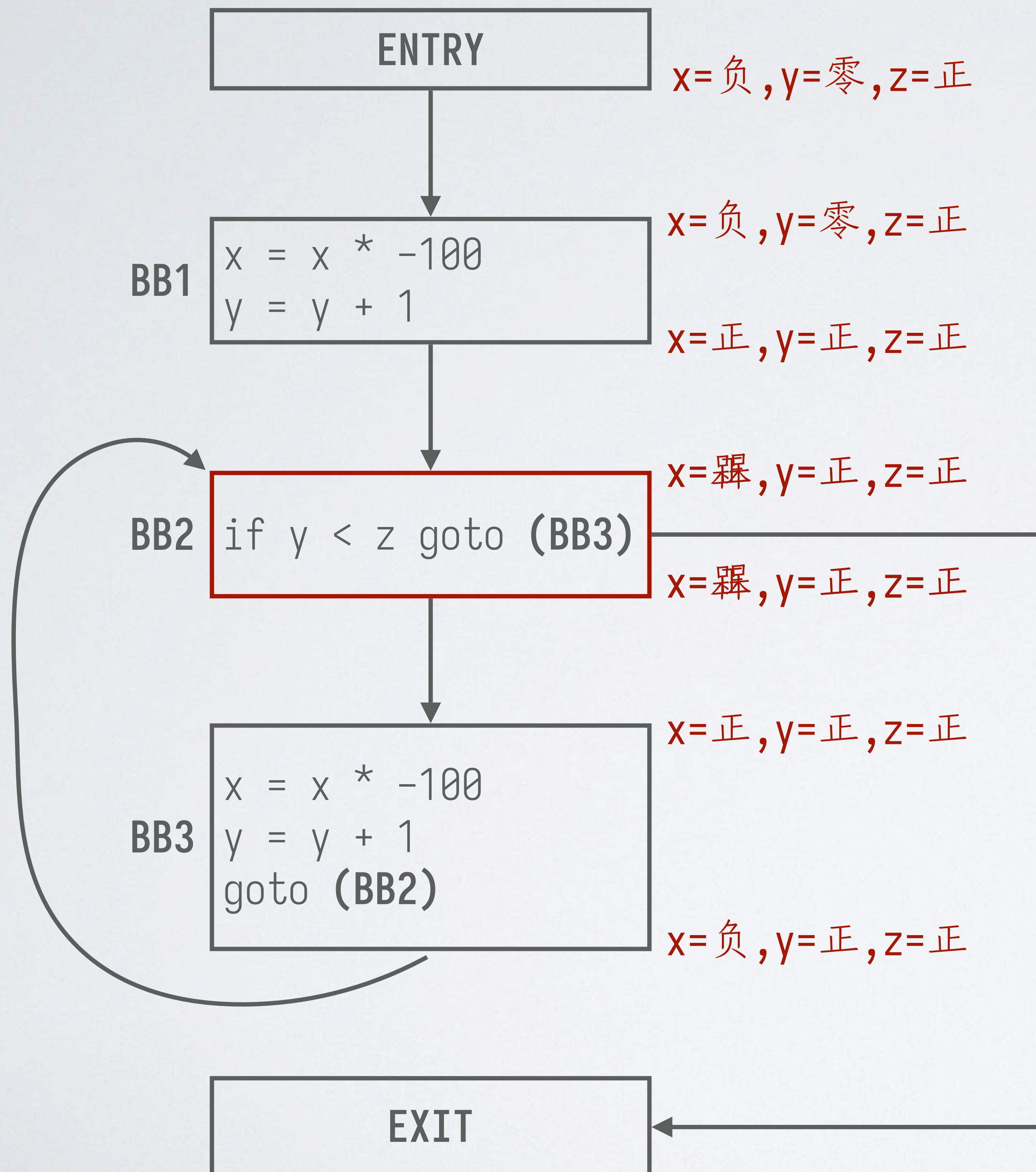
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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）



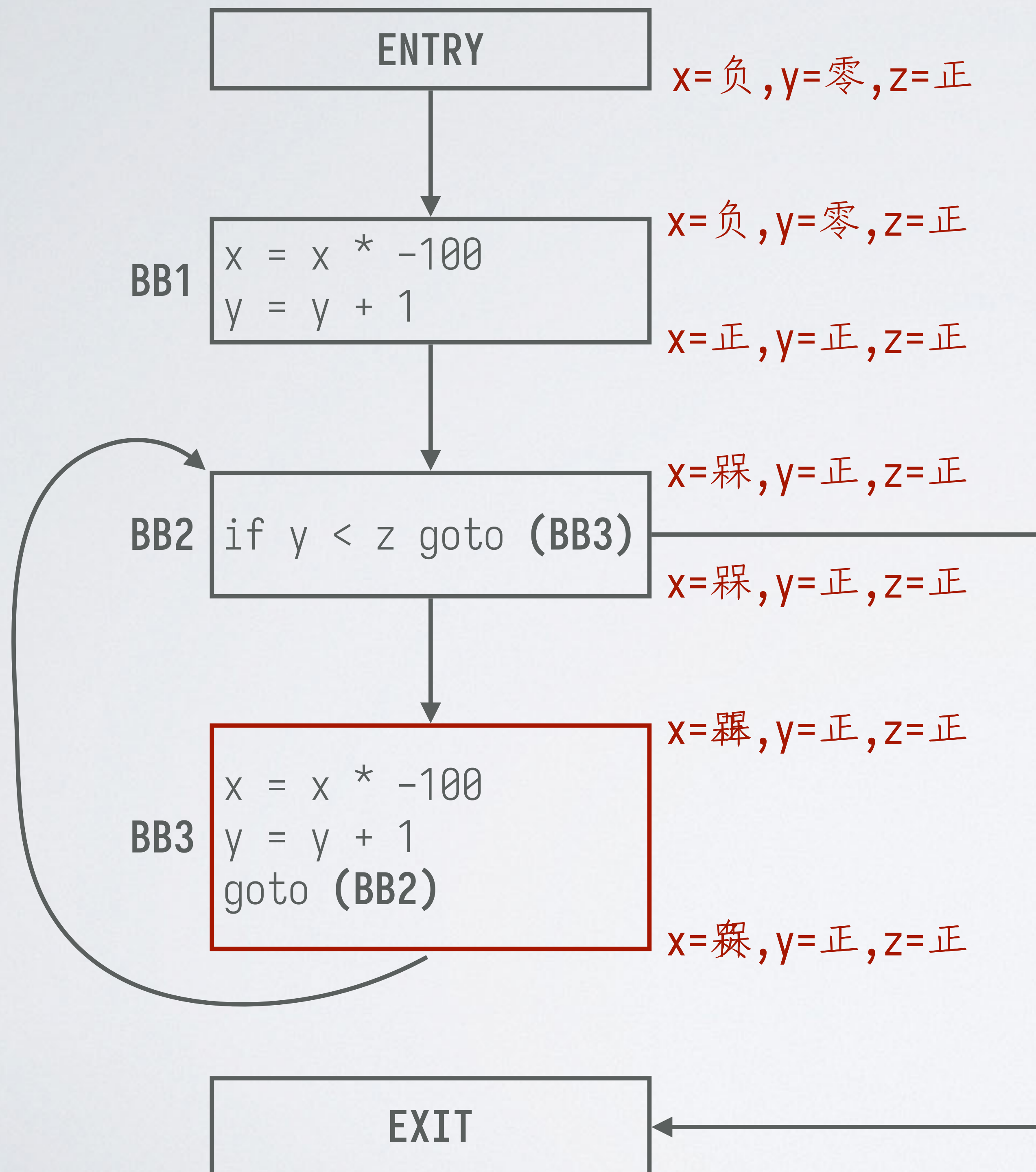
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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）



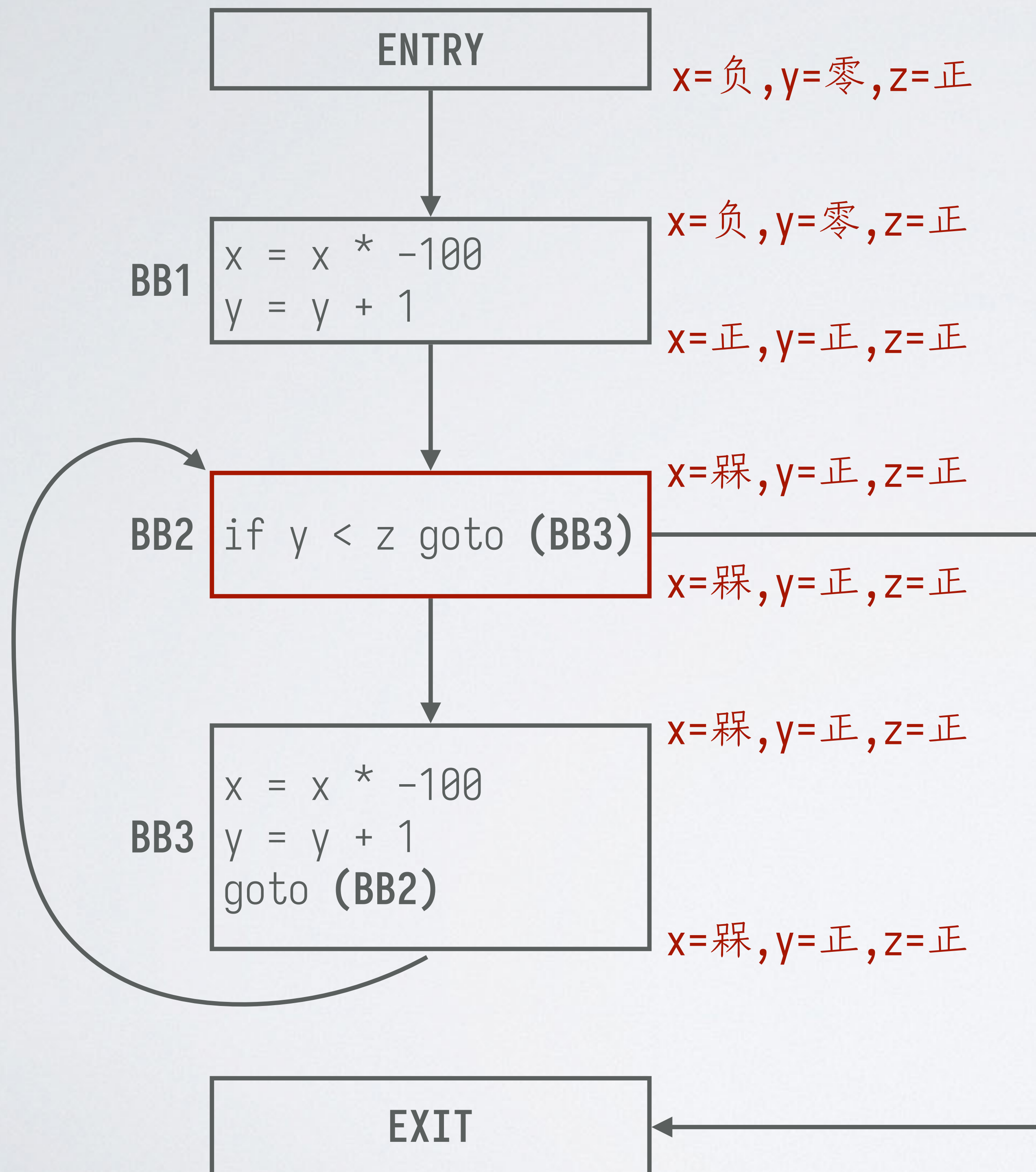
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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）



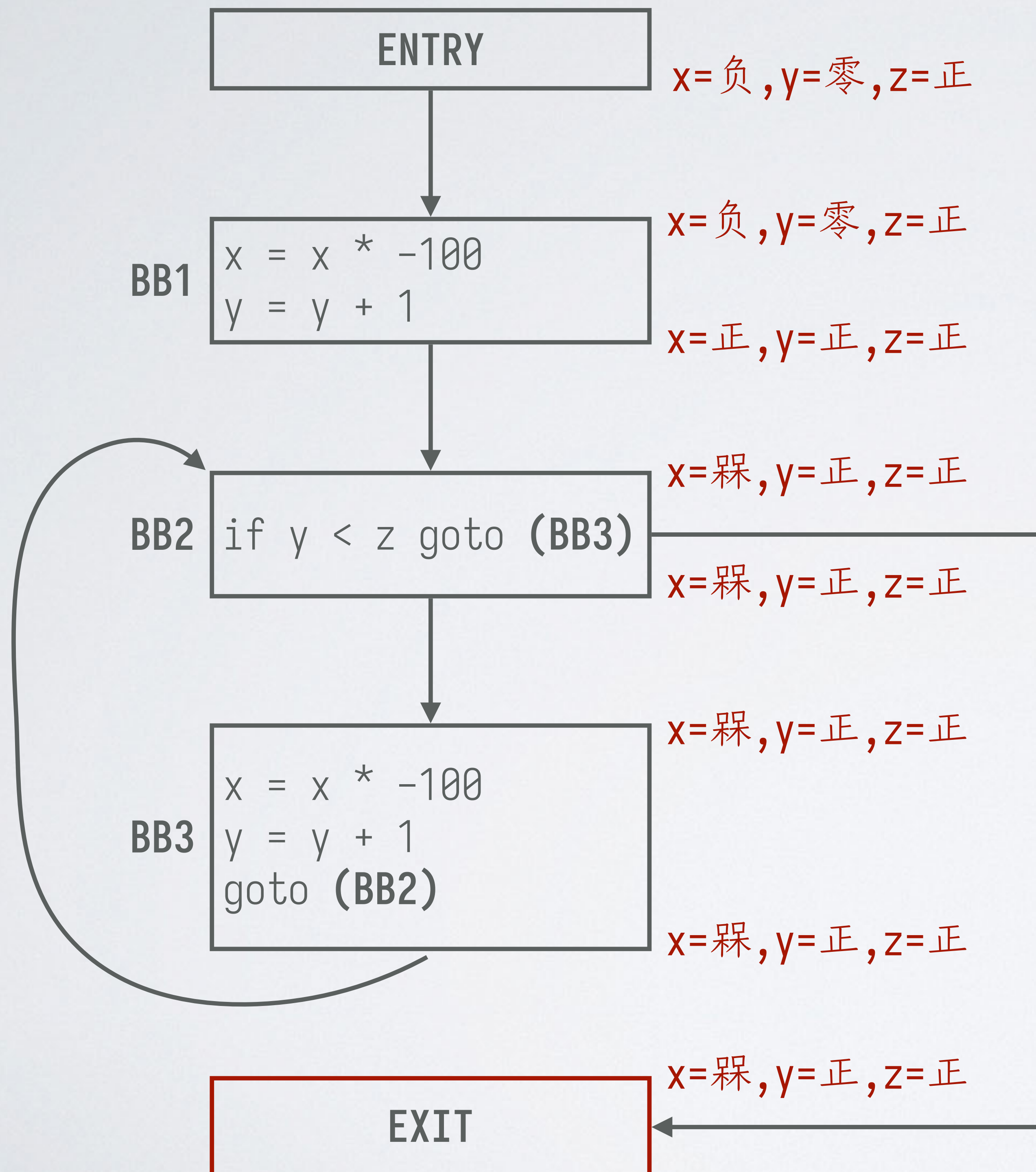
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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：$\text{OUT}[\text{ENTRY}] = v_{\text{ENTRY}}$
- 初始值：$\text{OUT}[B] = \top \ (B \neq \text{ENTRY})$
- 方程组：对任意 $B \neq \text{ENTRY}$，有

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

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

活跃变量分析：语句

B

```
s1:  e = read()
s2:  f = e + c
s3:  g = e * f
```

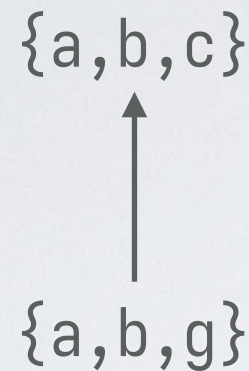
$def_{s_1} = \{e\}, use_{s_1} = \varnothing$

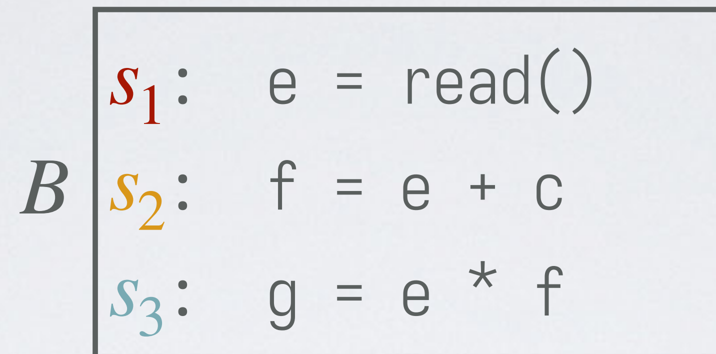
$def_{s_2} = \{f\}, use_{s_2} = \{e, c\}$

$def_{s_3} = \{g\}, use_{s_3} = \{e, f\}$

- 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





$def_{s_1} = \{e\}, use_{s_1} = \varnothing$

$def_{s_2} = \{f\}, use_{s_2} = \{e, c\}$

$def_{s_3} = \{g\}, use_{s_3} = \{e, f\}$

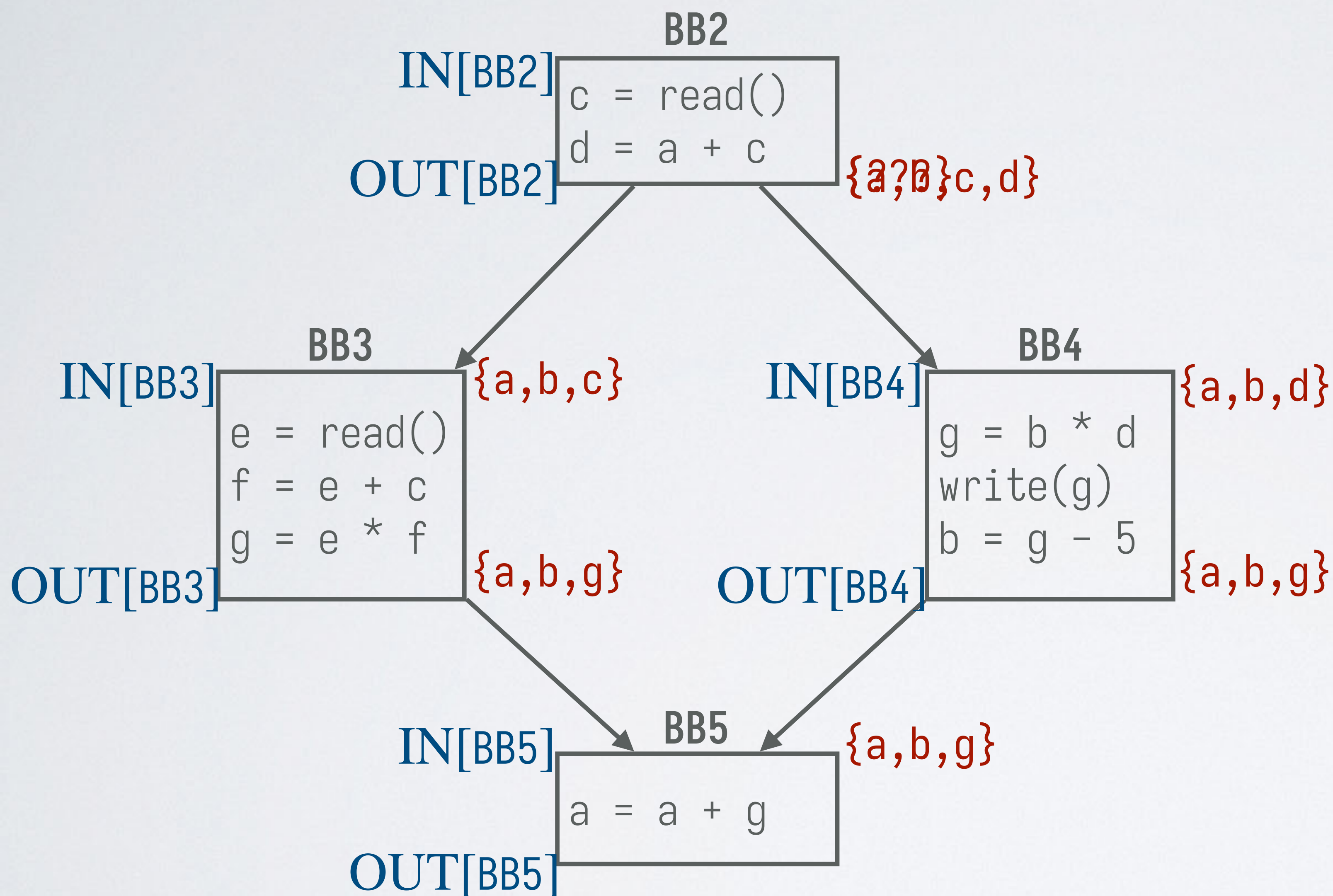
$def_B = \{e, f, g\}, use_B = \{c\}$

- 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：

  OUT[BB3] = IN[BB5]

  OUT[BB4] = IN[BB5]

  OUT[BB2] = ?

- 定义**交汇运算** ∧：

  $O_1 \wedge O_2 = O_1 \cup O_2$

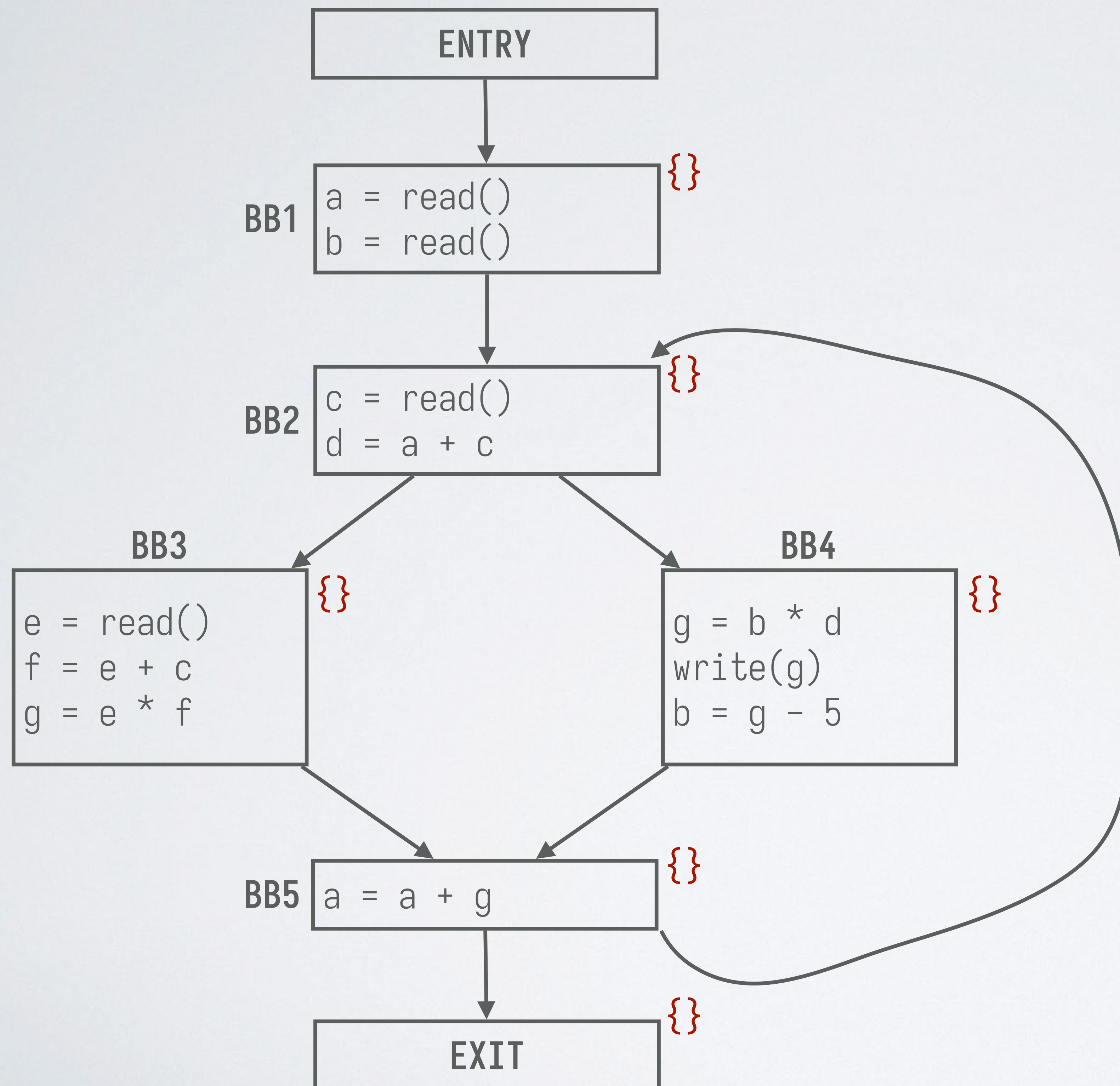
  即：在任意后继中活跃则认为是活跃的

- 分支的处理：

  OUT[BB2] = IN[BB3] ∧ IN[BB4]

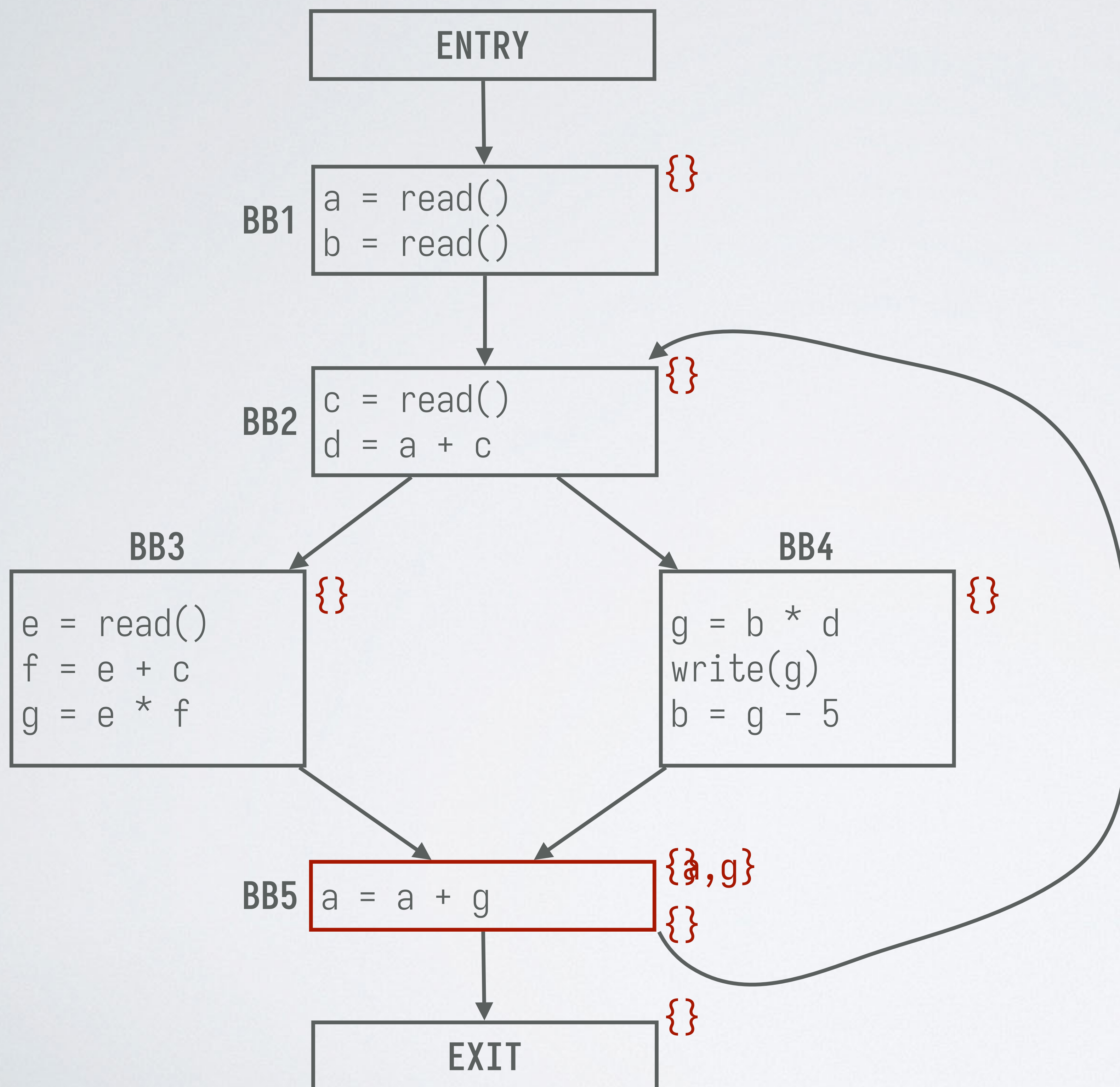
$$\mathrm{OUT}[B] = \bigwedge_{S\text{是}B\text{的后继}} \mathrm{IN}[S]$$

活跃变量分析：循环（1）



- **顶值(top)**为空集

活跃变量分析：循环（2）



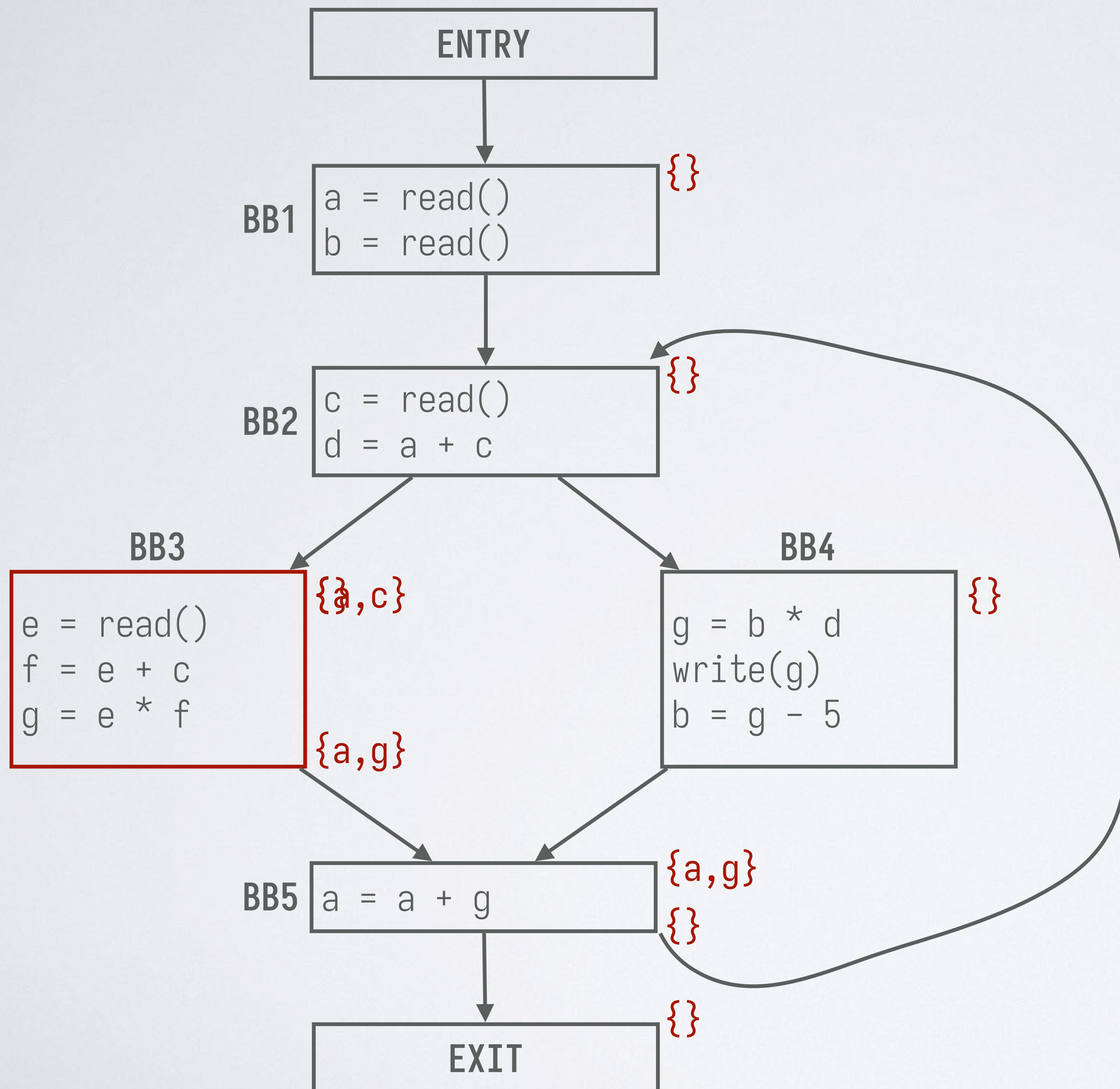
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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）



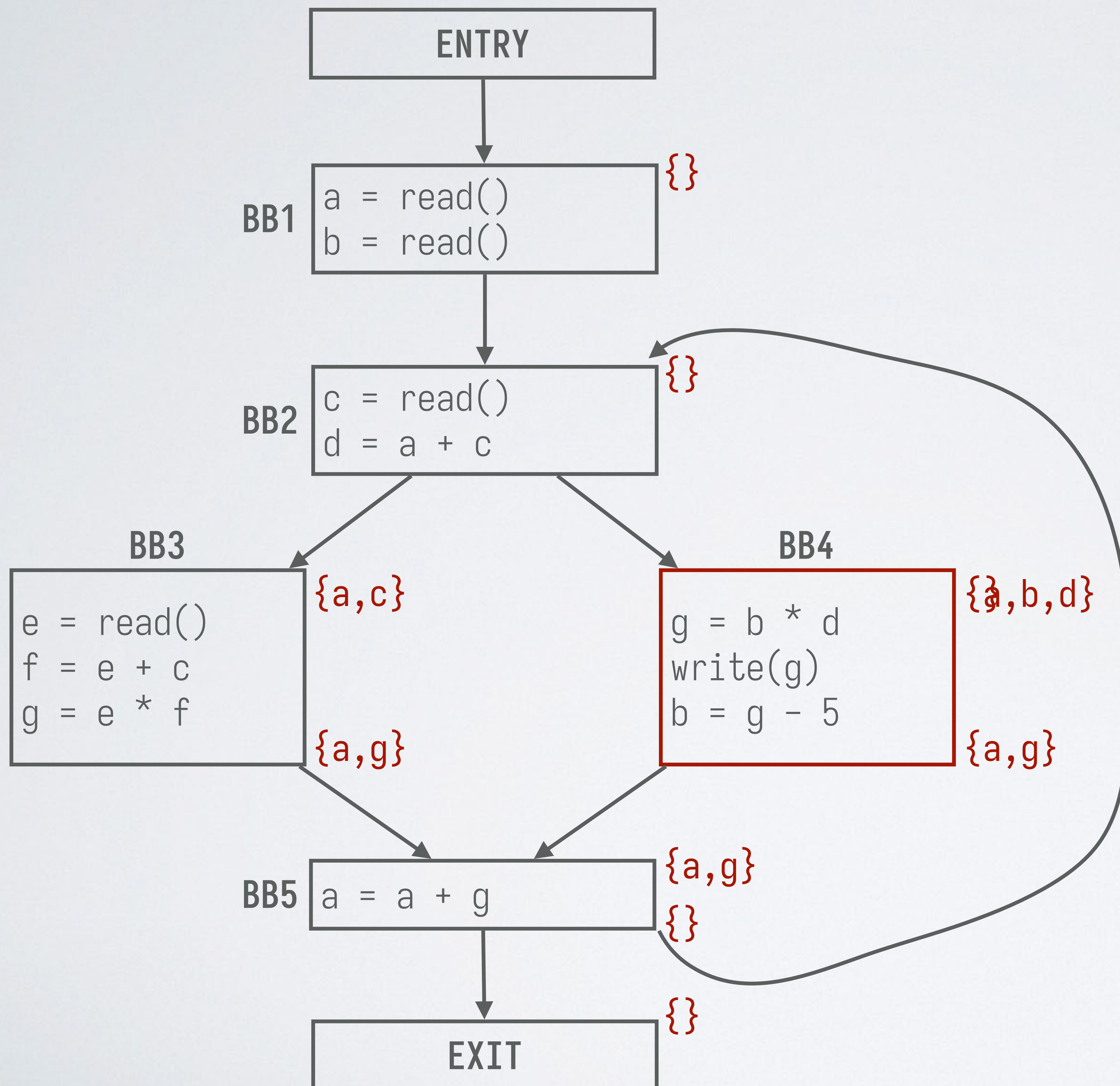
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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）



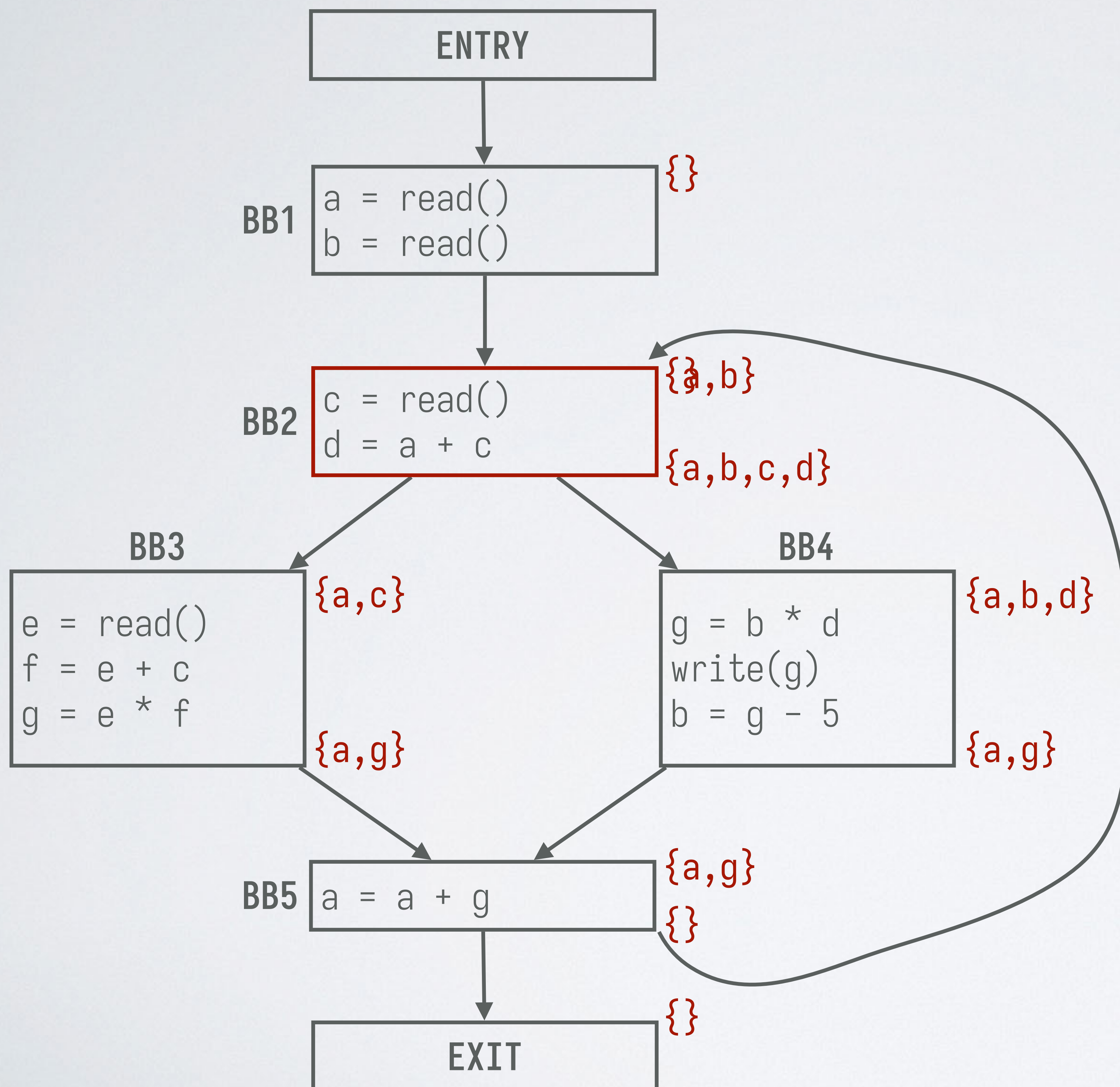
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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）



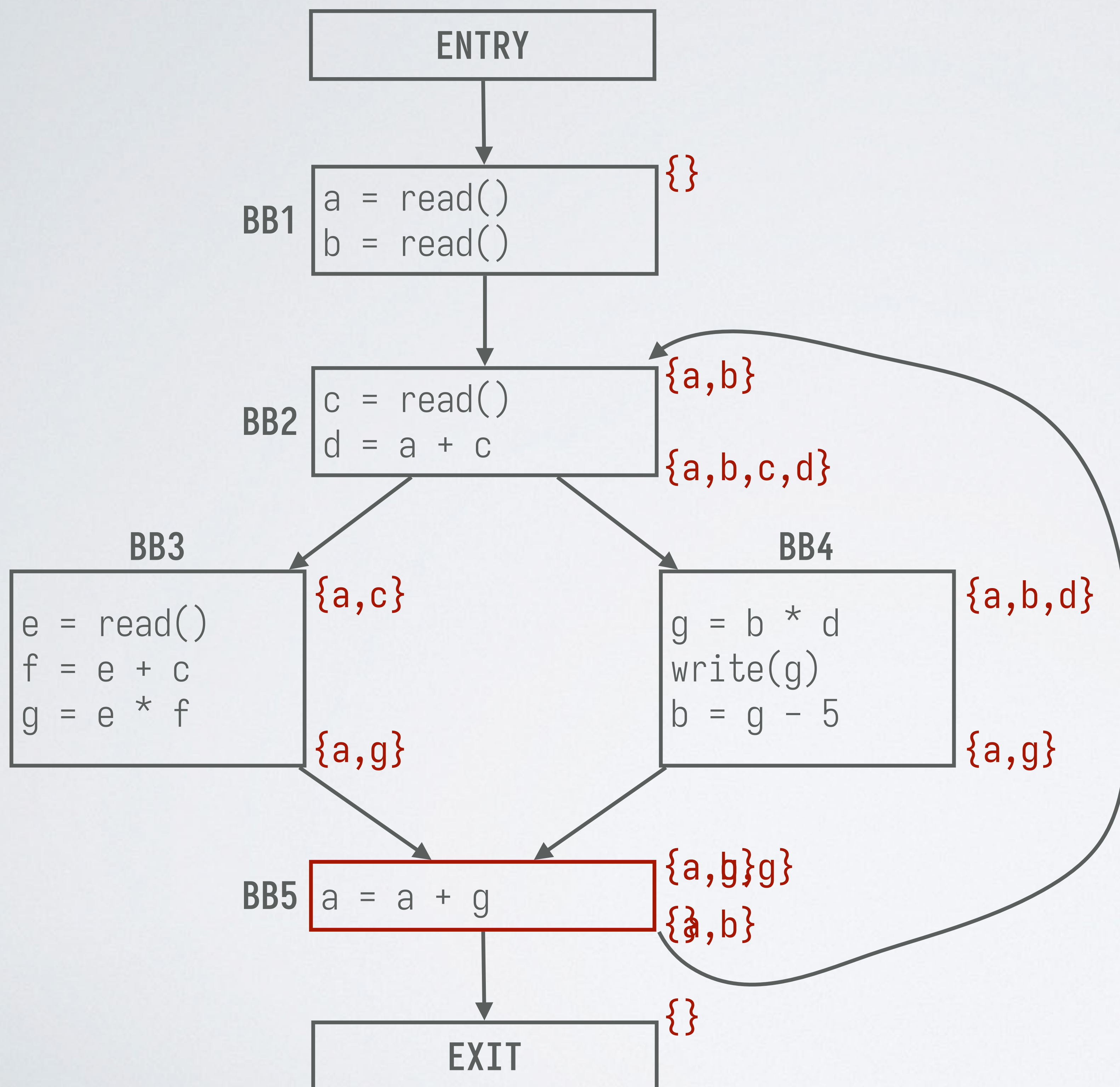
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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）



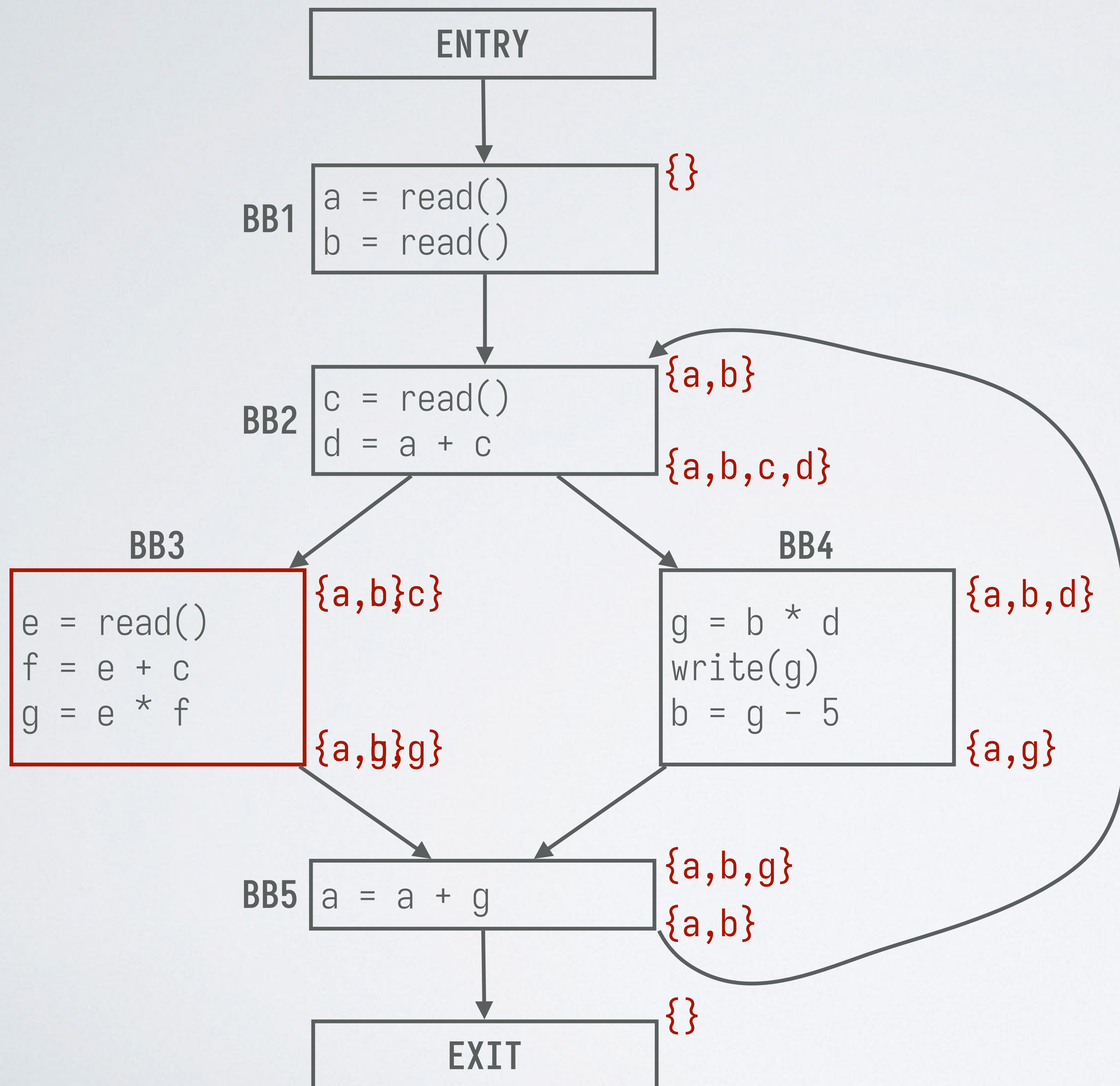
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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）



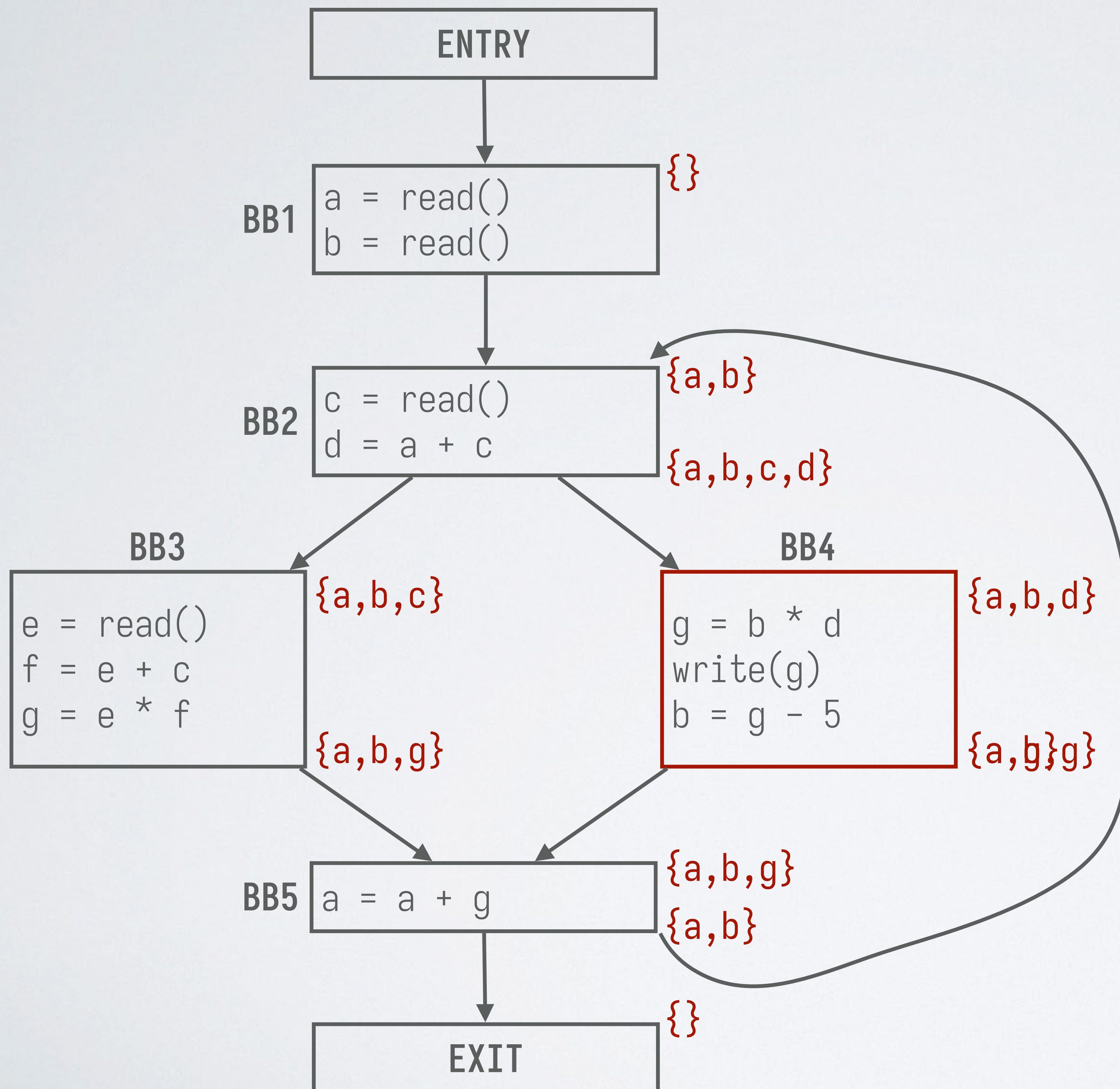
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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）



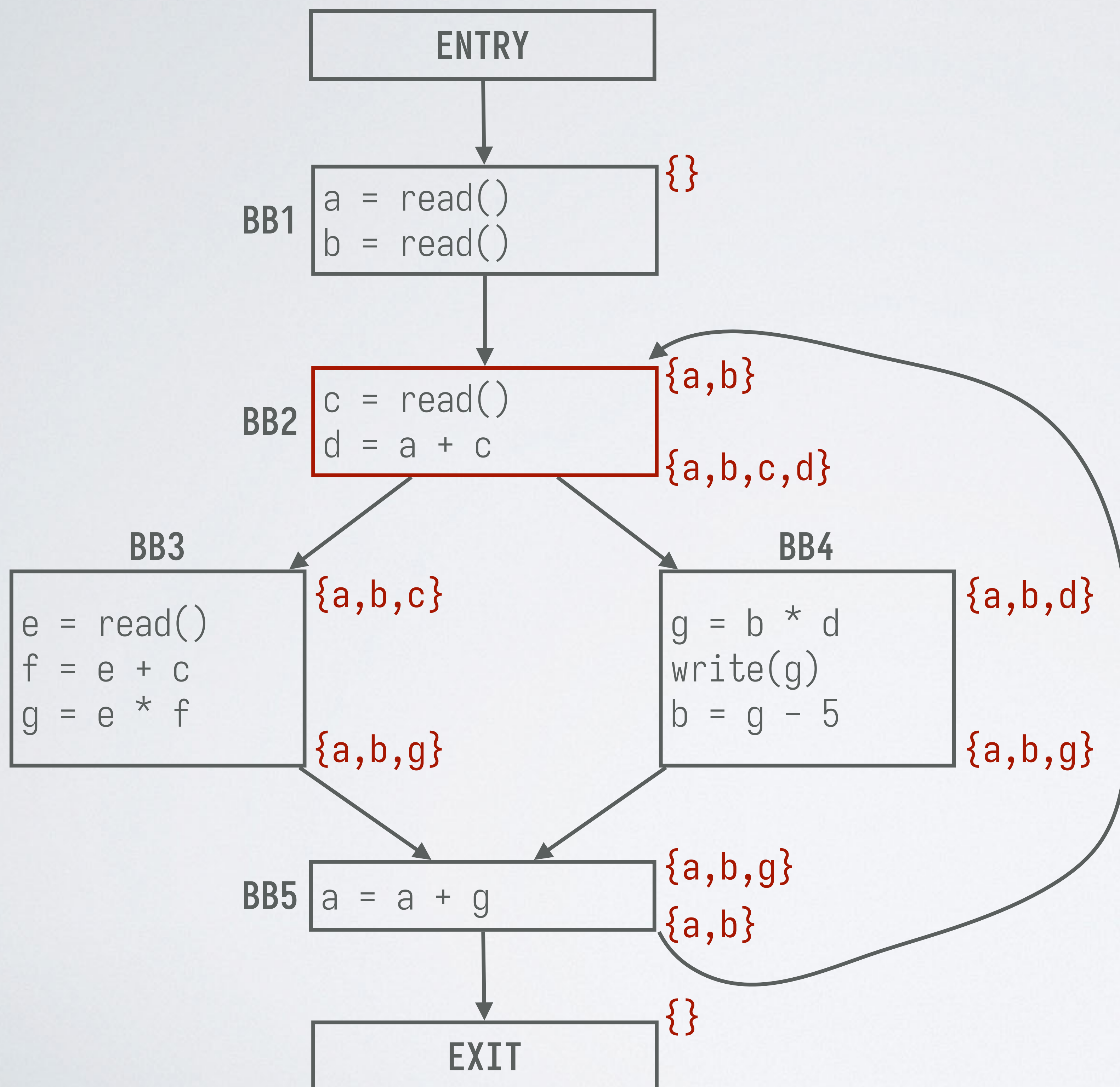
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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）



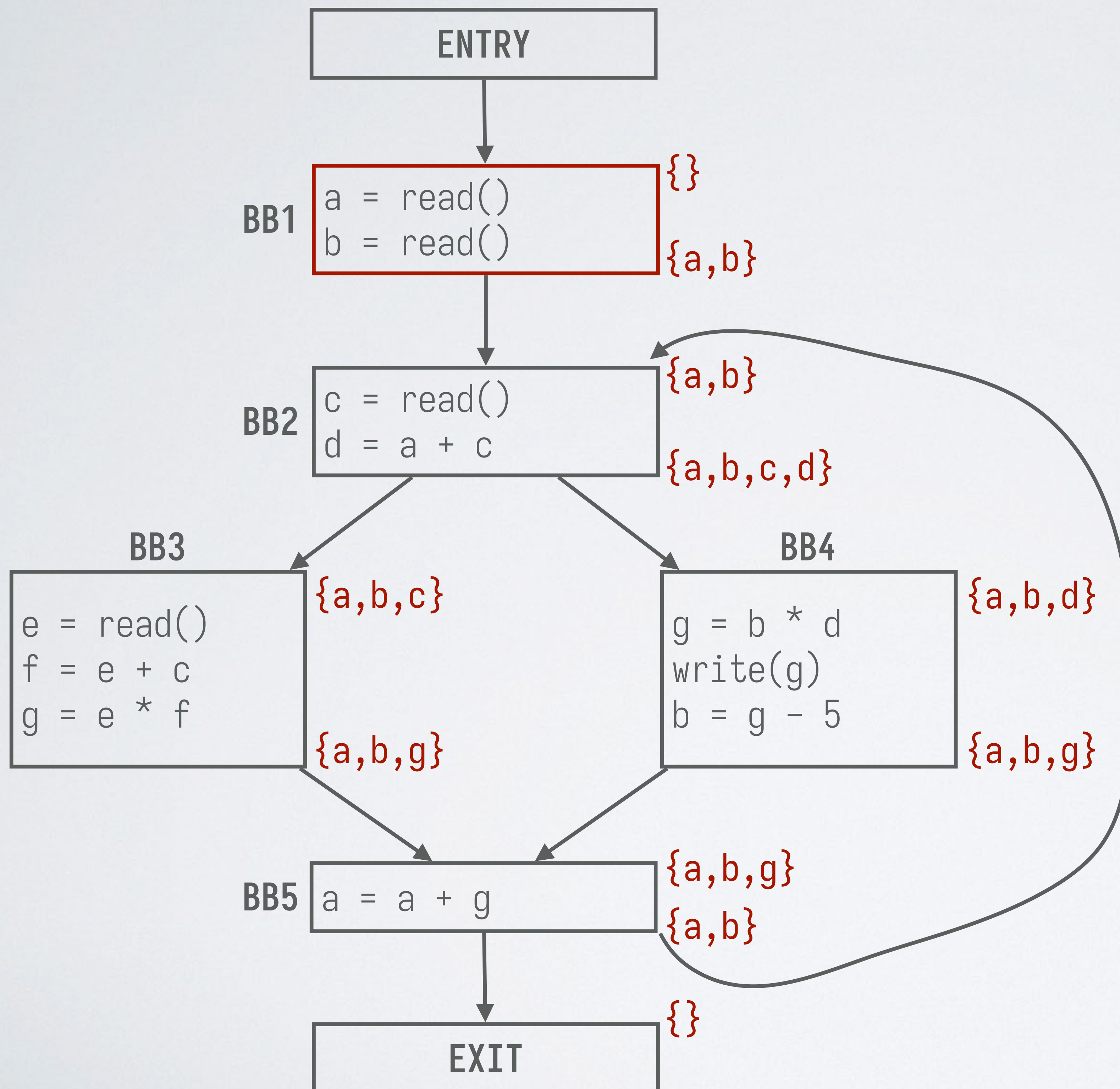
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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）



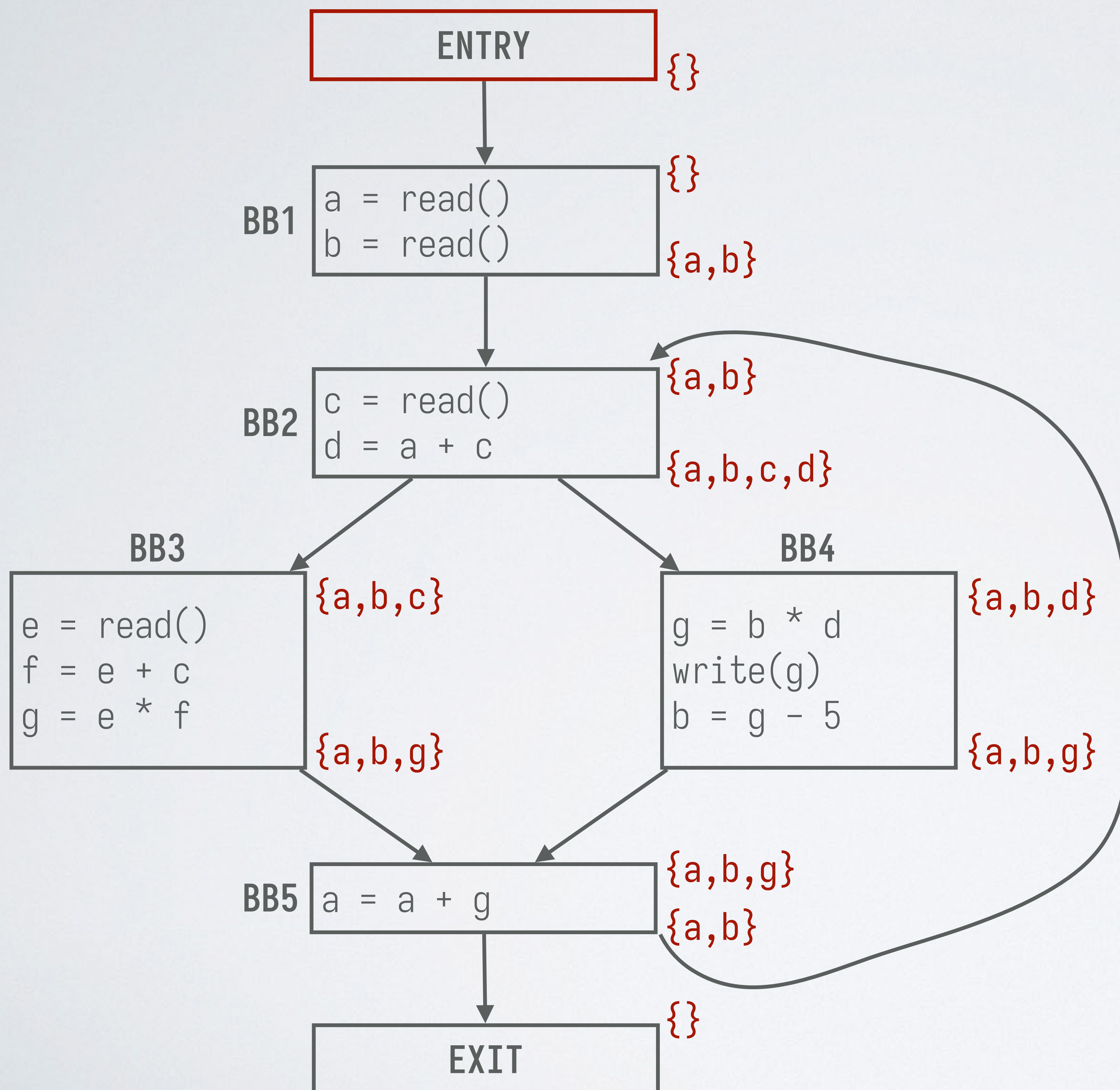
- 通过**迭代法**解方程
- 每次选择一个基本块，更新它的 IN 和 OUT

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

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

- 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

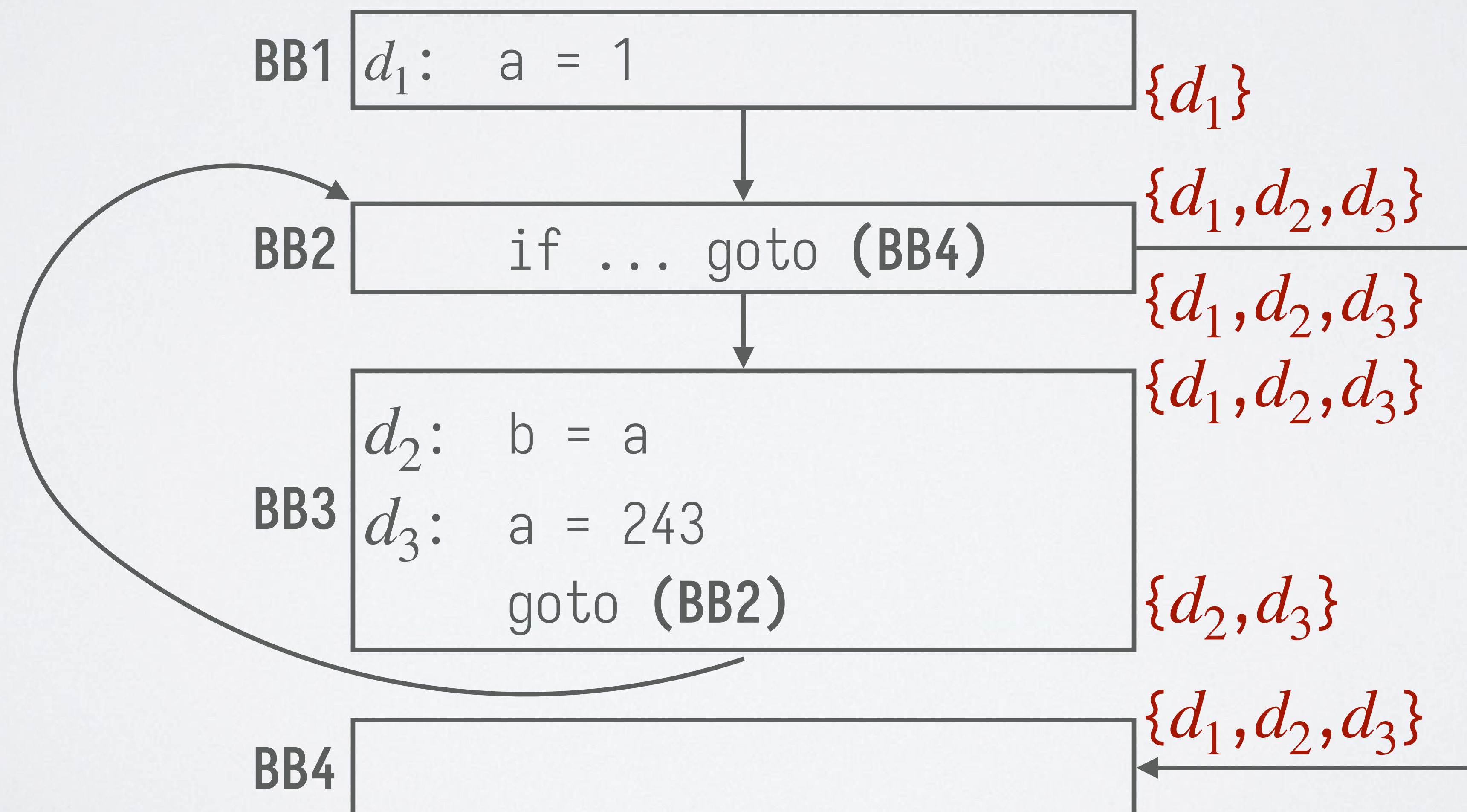
- 数据流分析的域 V, 交汇运算 $\wedge : V \times V \rightarrow V$, 顶值 $\top \in V$
- 每个基本块 B 的传递函数 $f_B : V \rightarrow V$ (从出口到入口)
- 边界条件: $\text{IN}[\texttt{EXIT}] = v_{\texttt{EXIT}}$
- 初始值: $\text{IN}[B] = \top$ $(B \neq \texttt{EXIT})$
- 方程组: 对任意 $B \neq \texttt{EXIT}$, 有

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

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

可达定值分析（1）

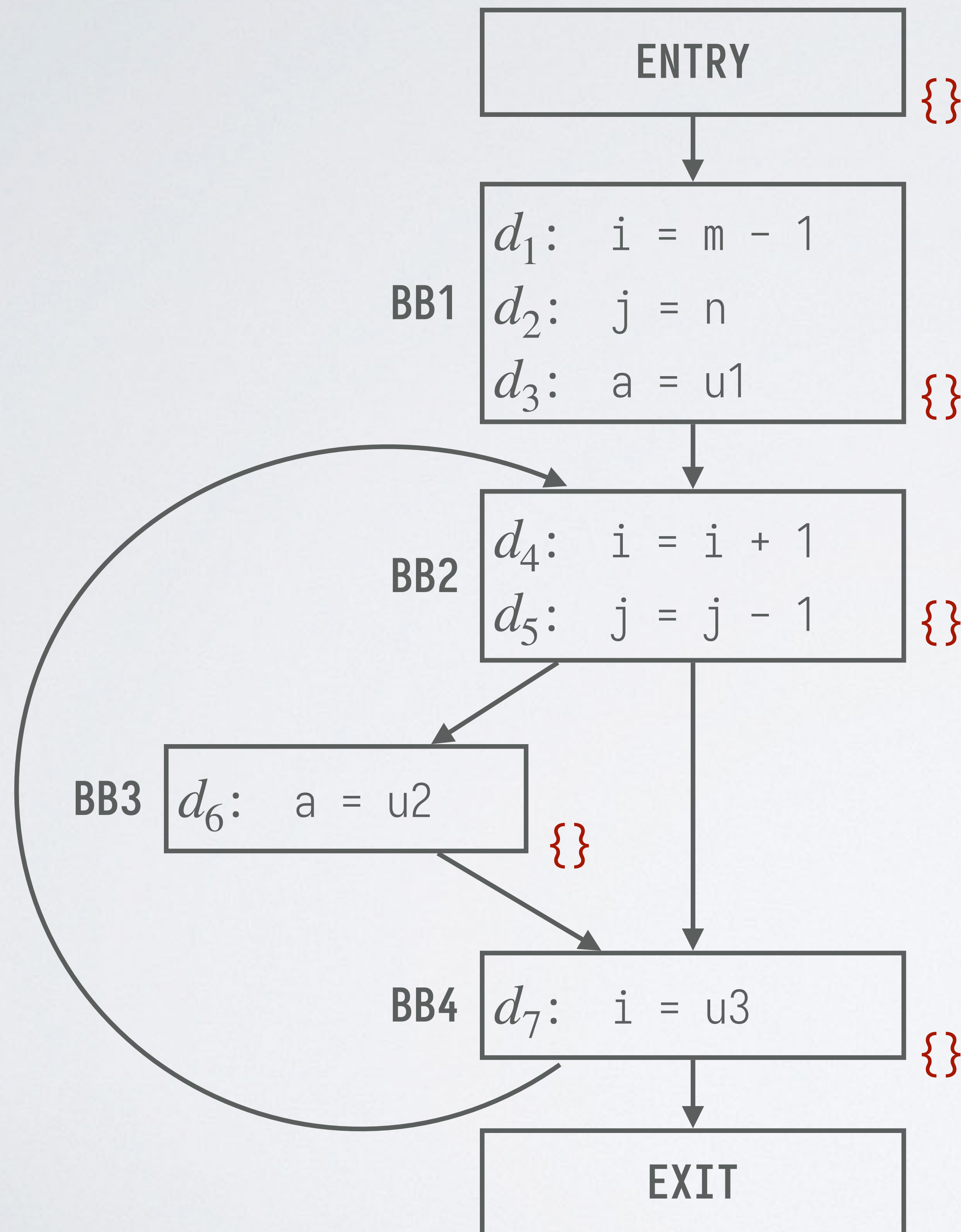
- **可达定值(reaching definition)**
  - 可能沿着某条路径到达一个程序点的定值(形如 $x = \ldots$ 的语句)
  - **前向**分析，值域为**定值的集合**
  - 用途：常量传播



可达定值分析（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

可达定值分析（3）



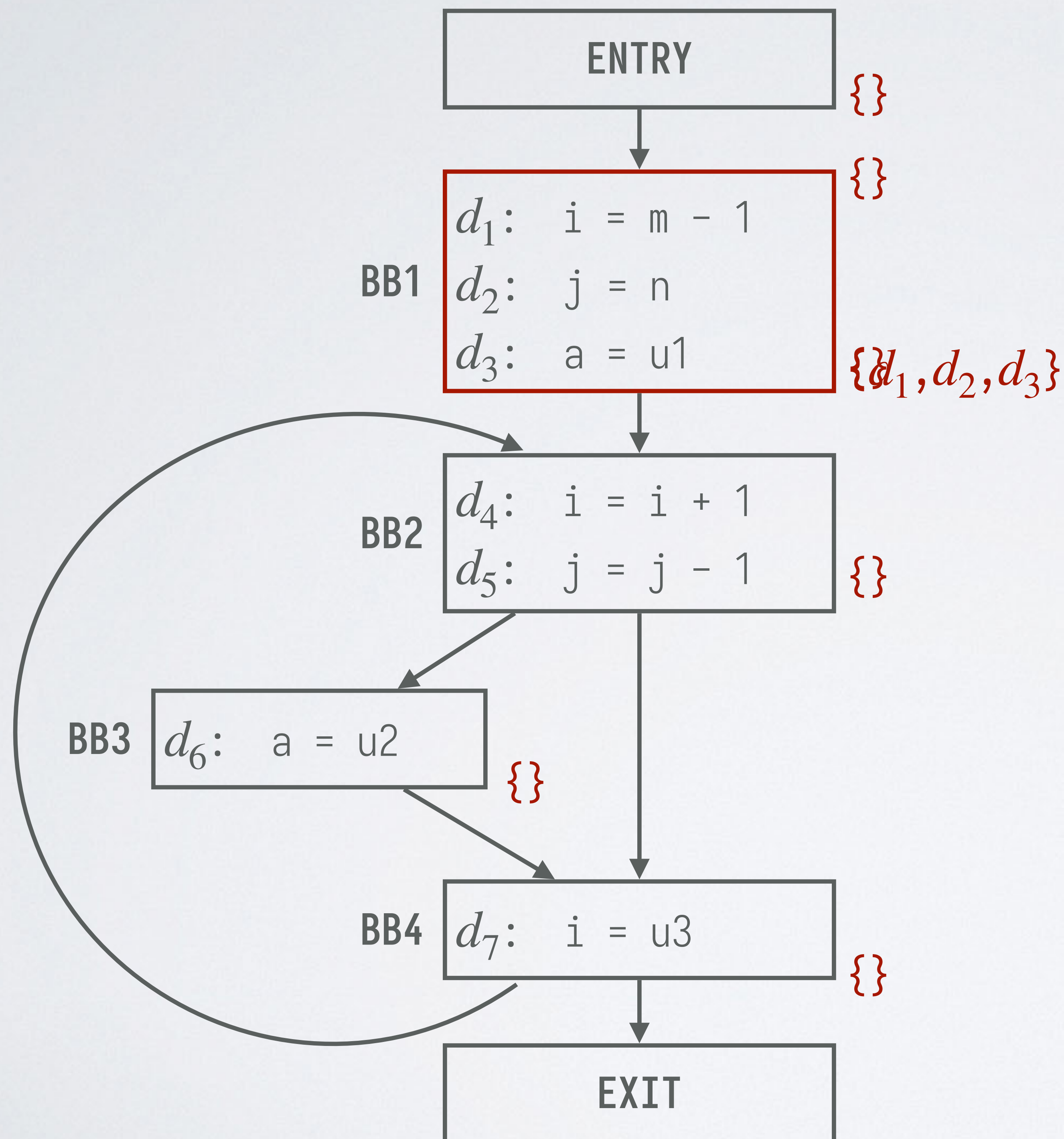
$gen_{\mathrm{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\mathrm{BB1}} = \{d_4, d_5, d_6, d_7\}$

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

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

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

可达定值分析（4）



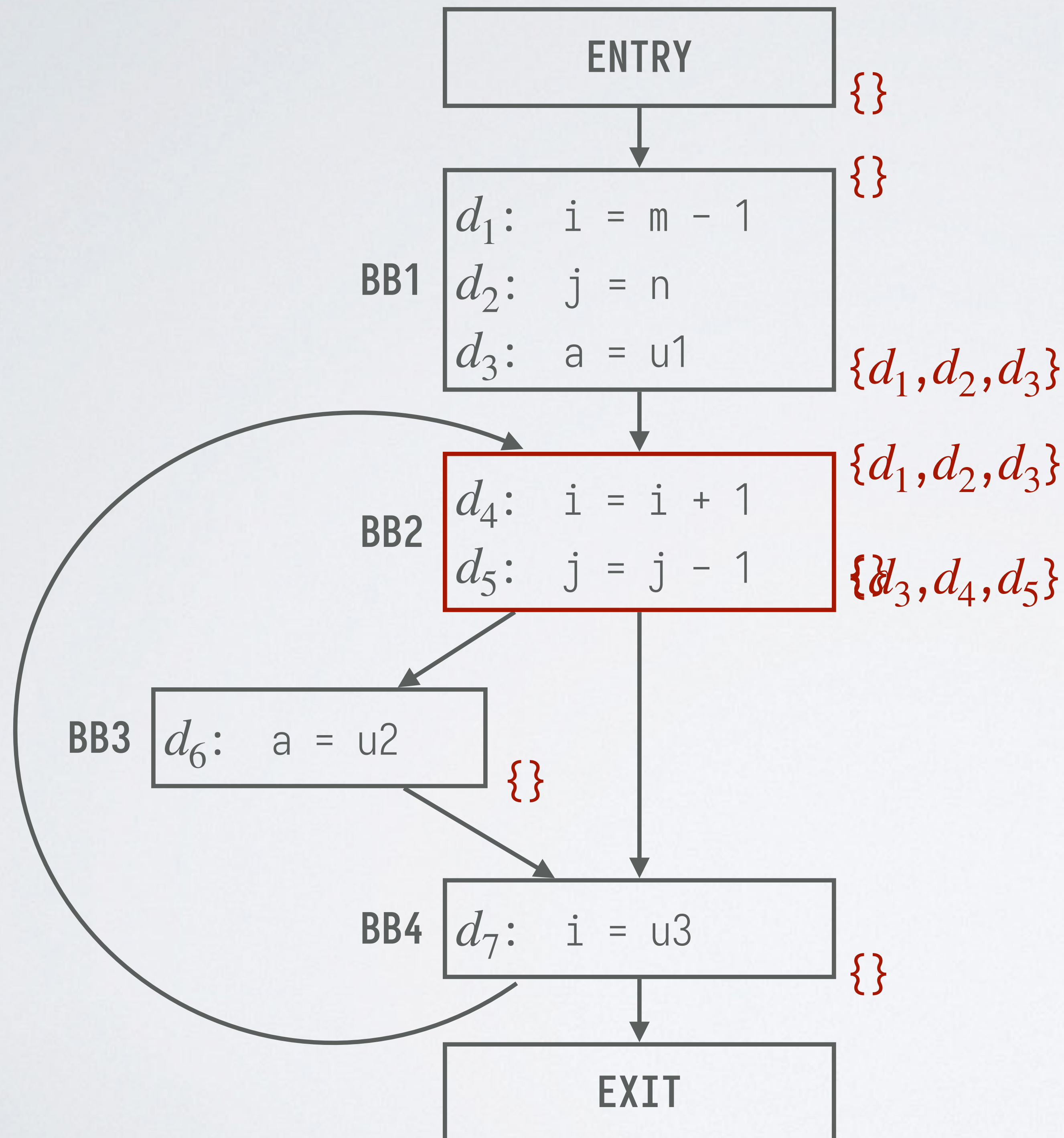
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

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

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

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

可达定值分析（5）



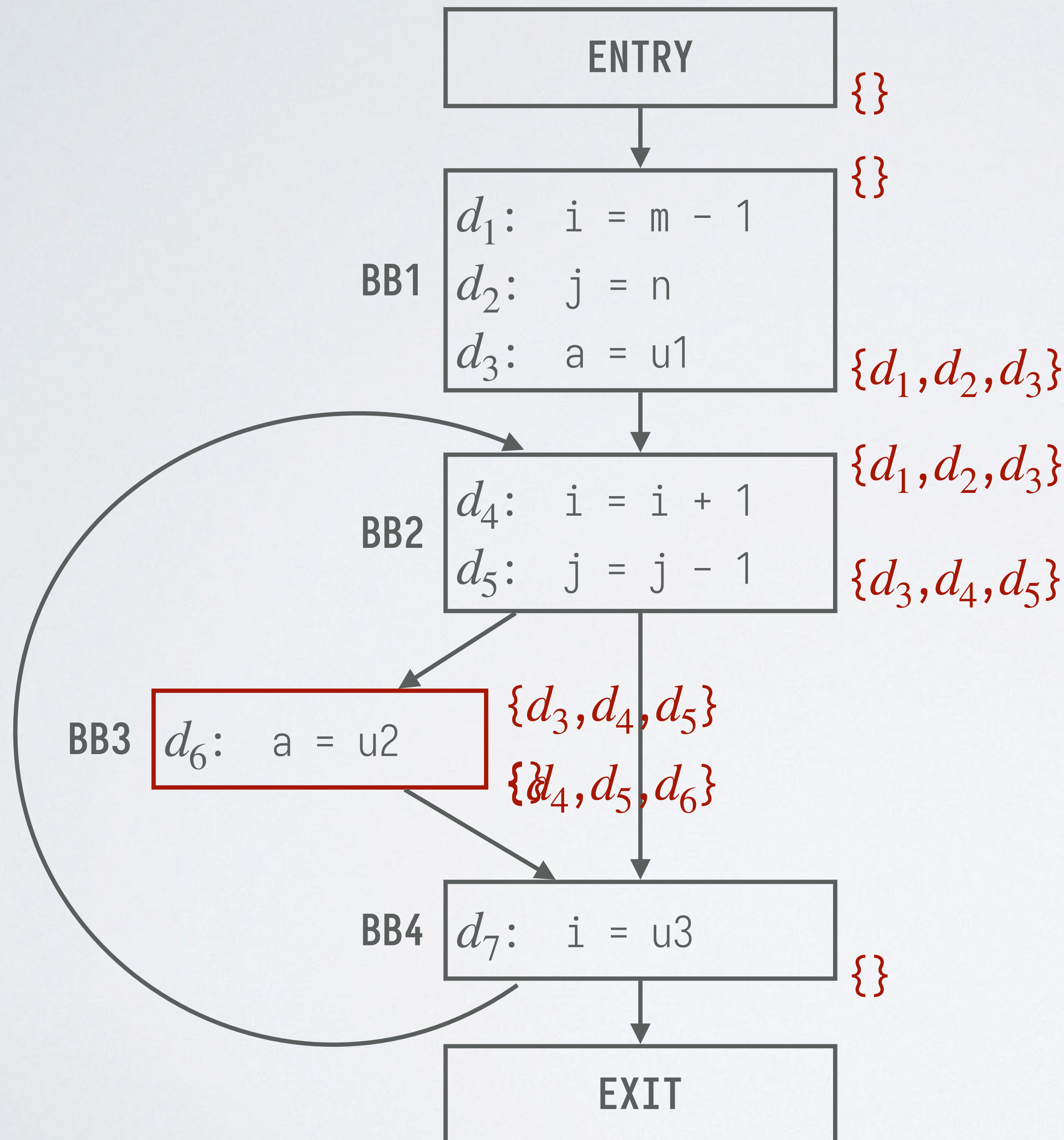
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

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

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

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

可达定值分析（6）



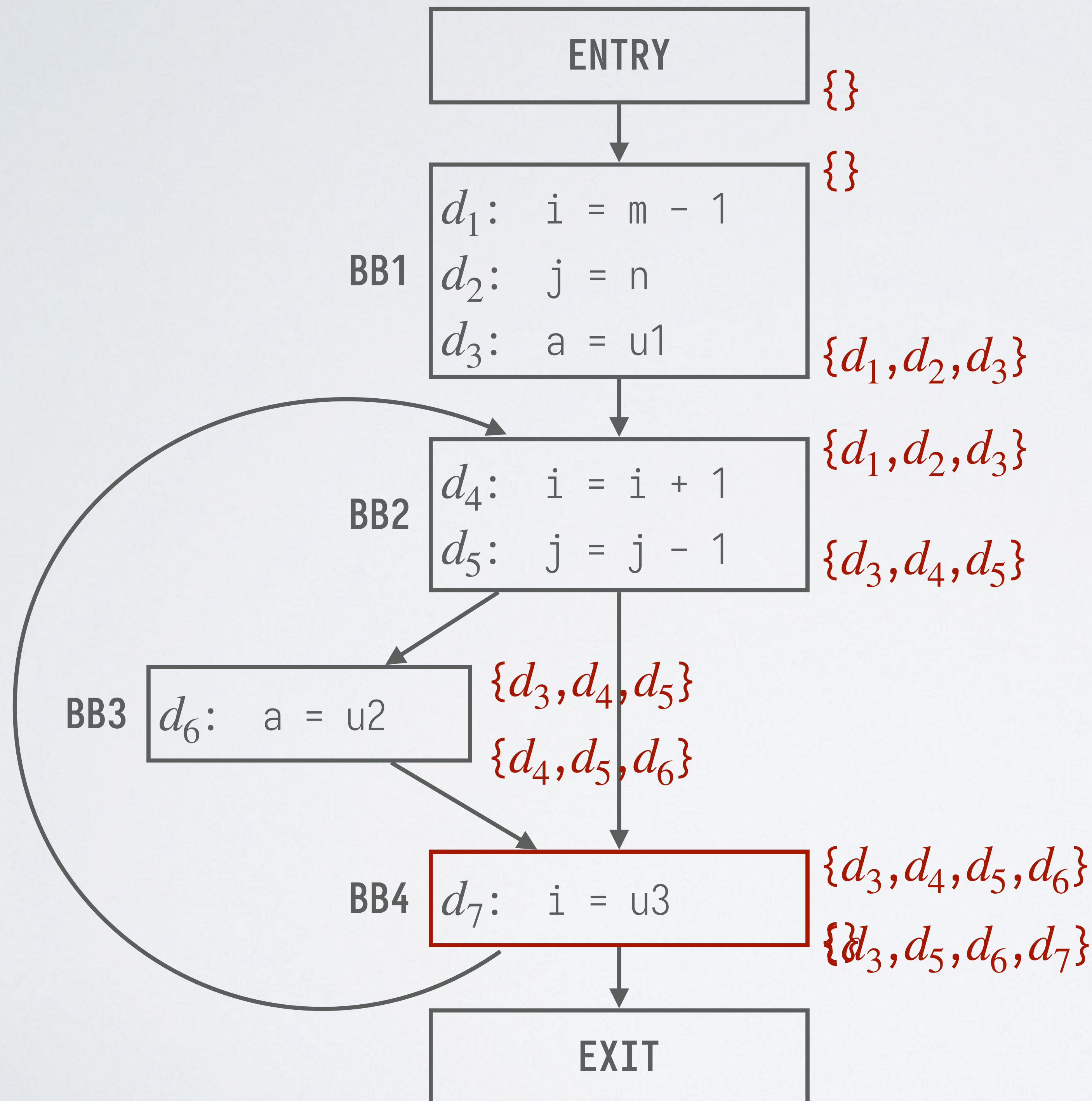
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

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

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

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

可达定值分析（7）



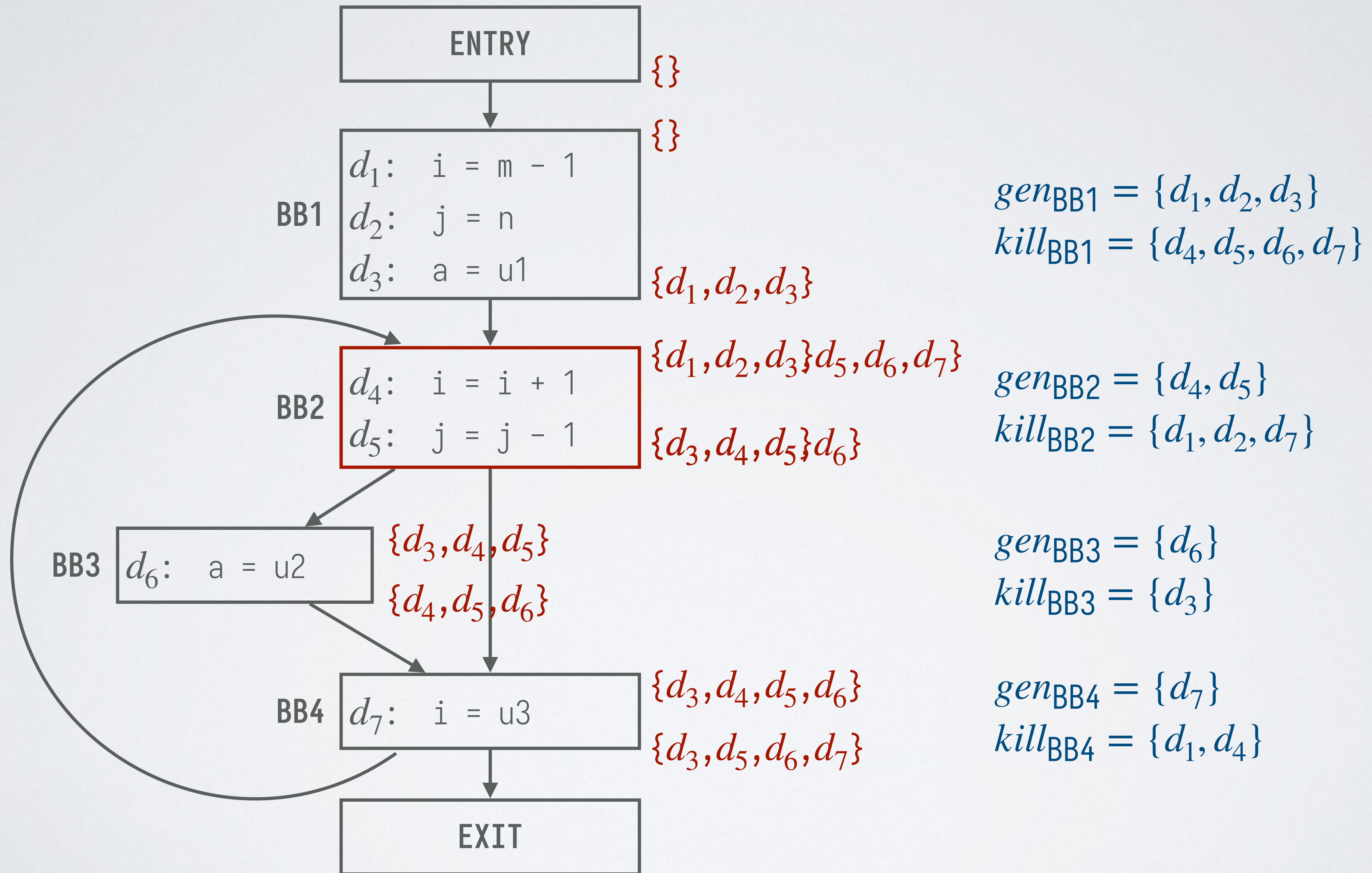
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

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

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

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

可达定值分析（8）



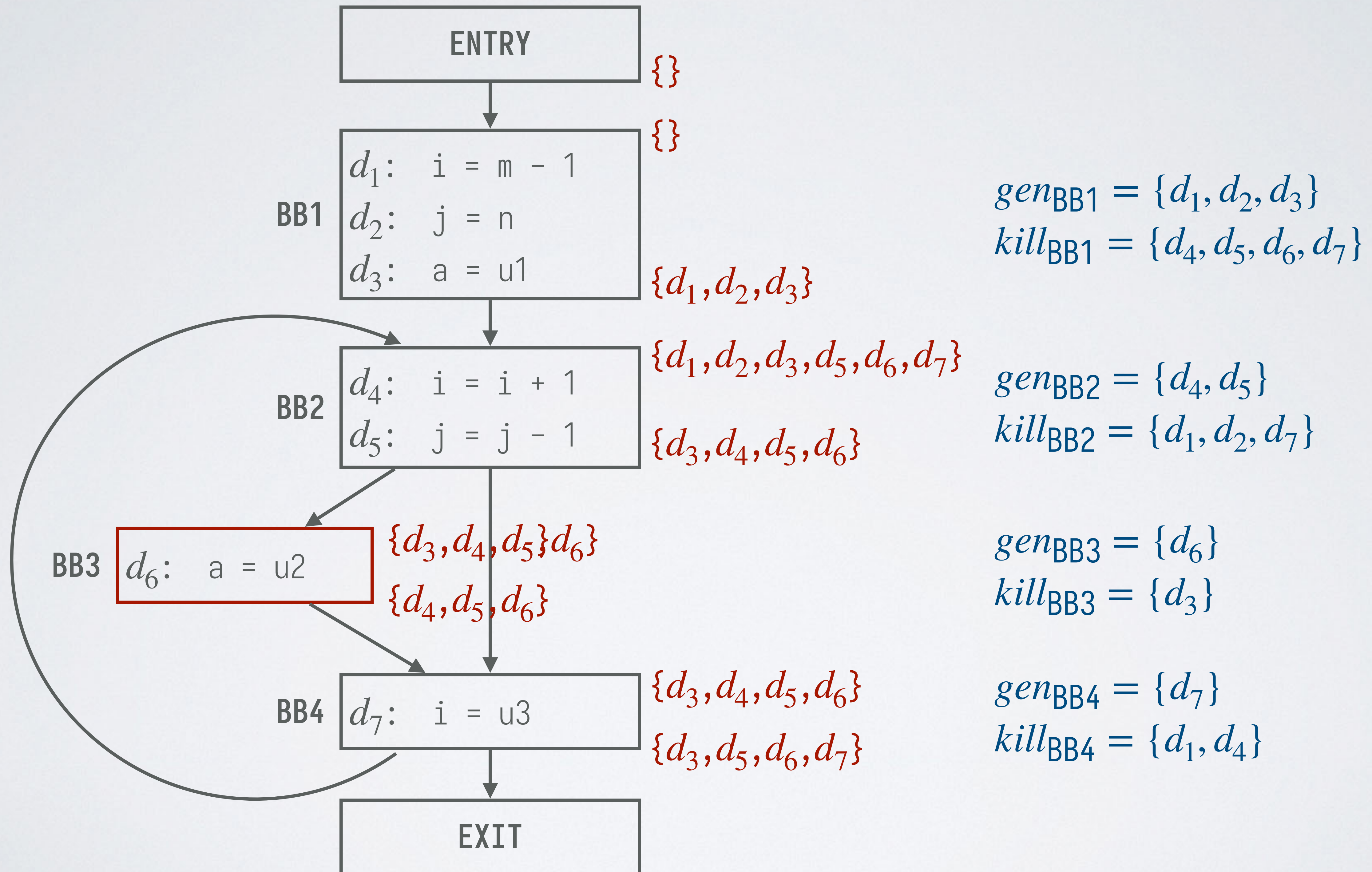
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

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

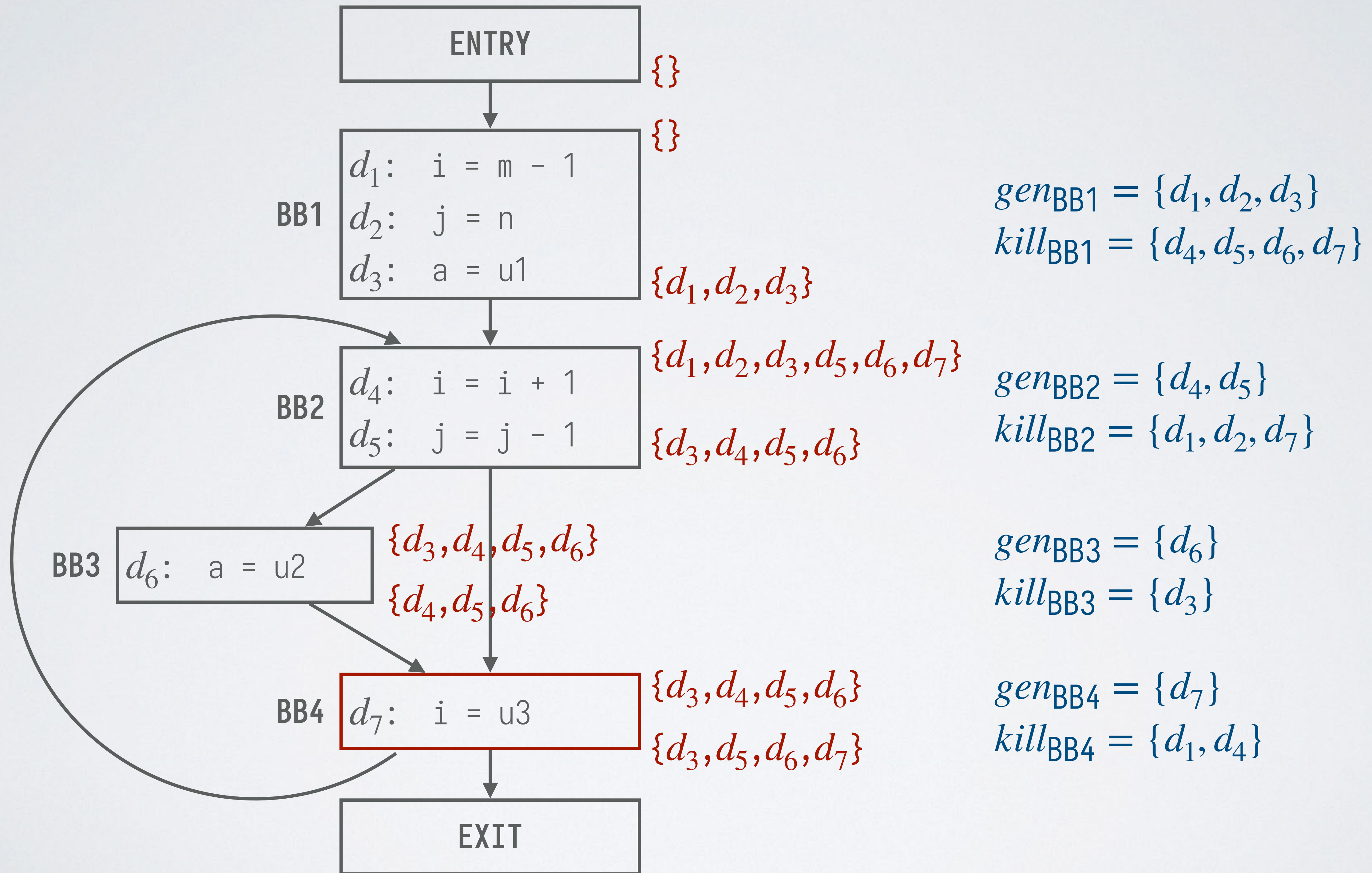
$gen_{\text{BB3}} = \{d_6\}$
$kill_{\text{BB3}} = \{d_3\}$

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

可达定值分析（9）



可达定值分析（10）



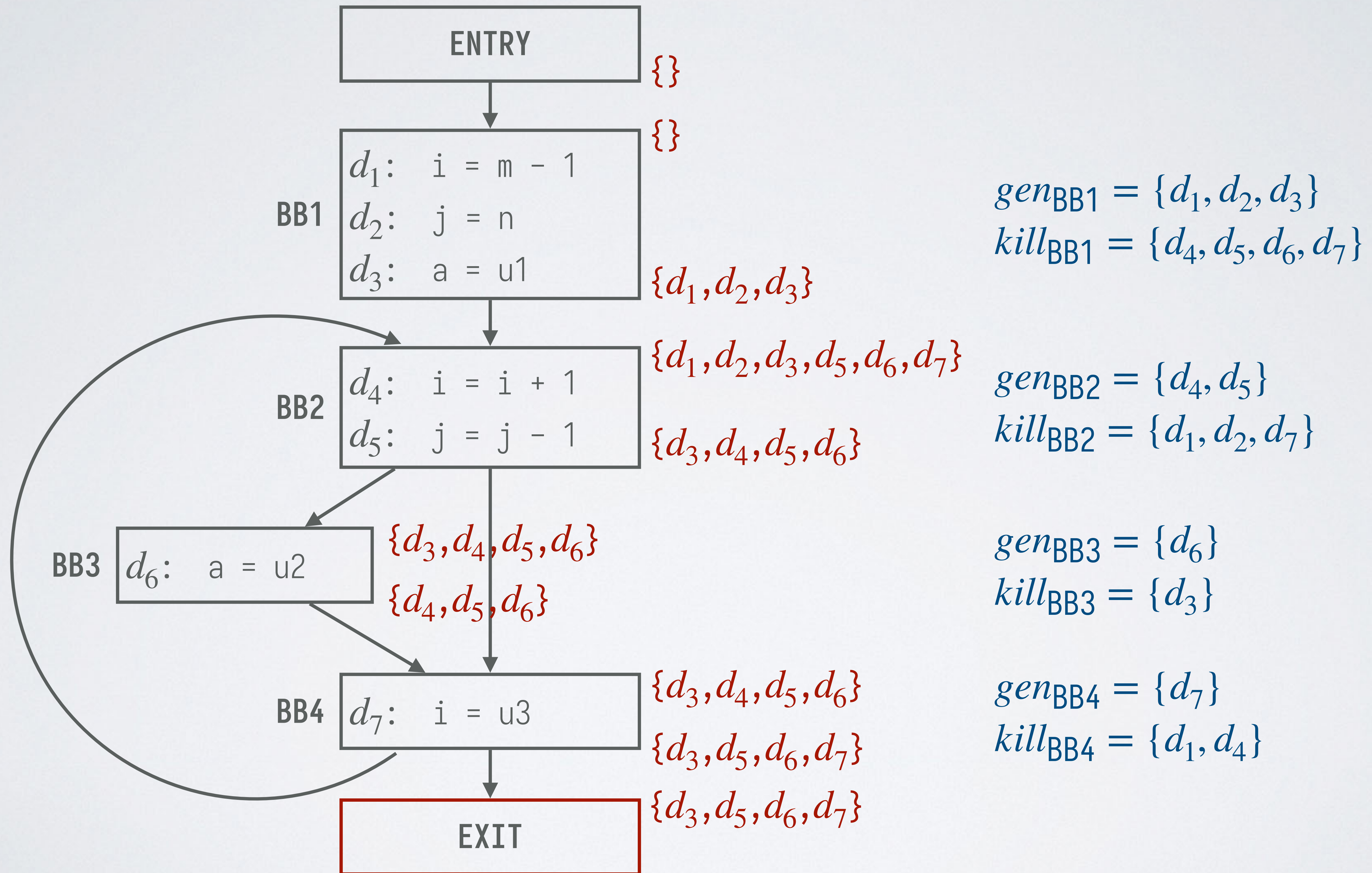
$gen_{\text{BB1}} = \{d_1, d_2, d_3\}$
$kill_{\text{BB1}} = \{d_4, d_5, d_6, d_7\}$

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

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

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

可达定值分析（11）



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

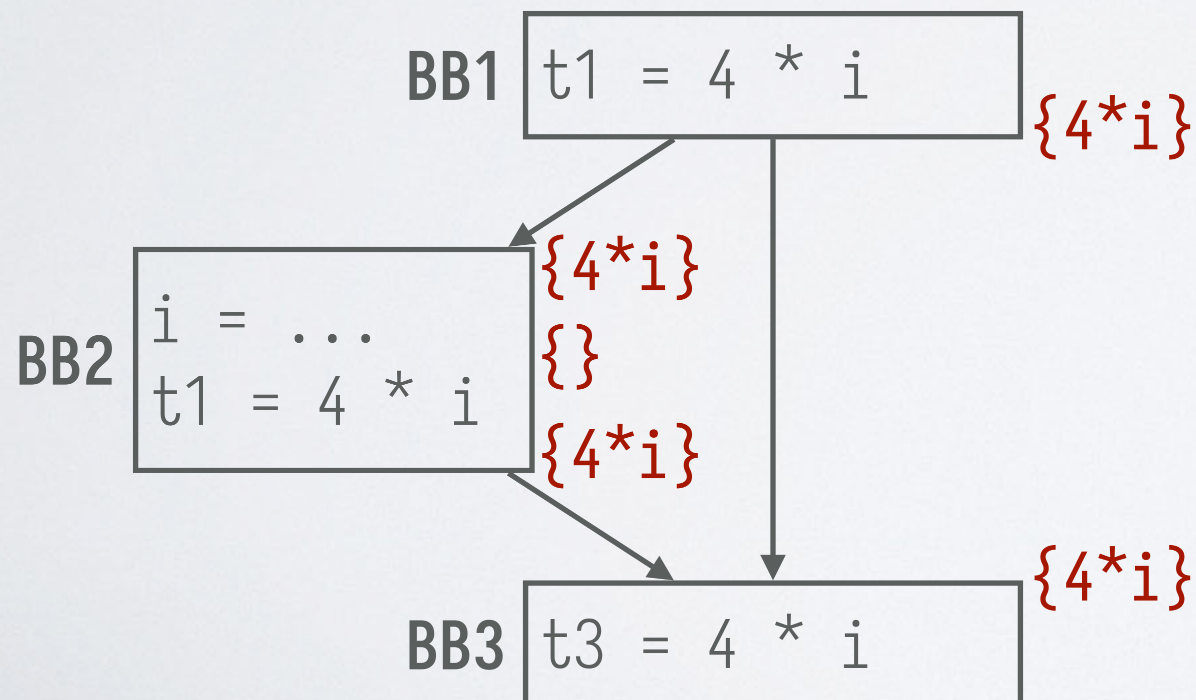
$gen_{\mathrm{BB2}} = \{d_4, d_5\}$
$kill_{\mathrm{BB2}} = \{d_1, d_2, d_7\}$

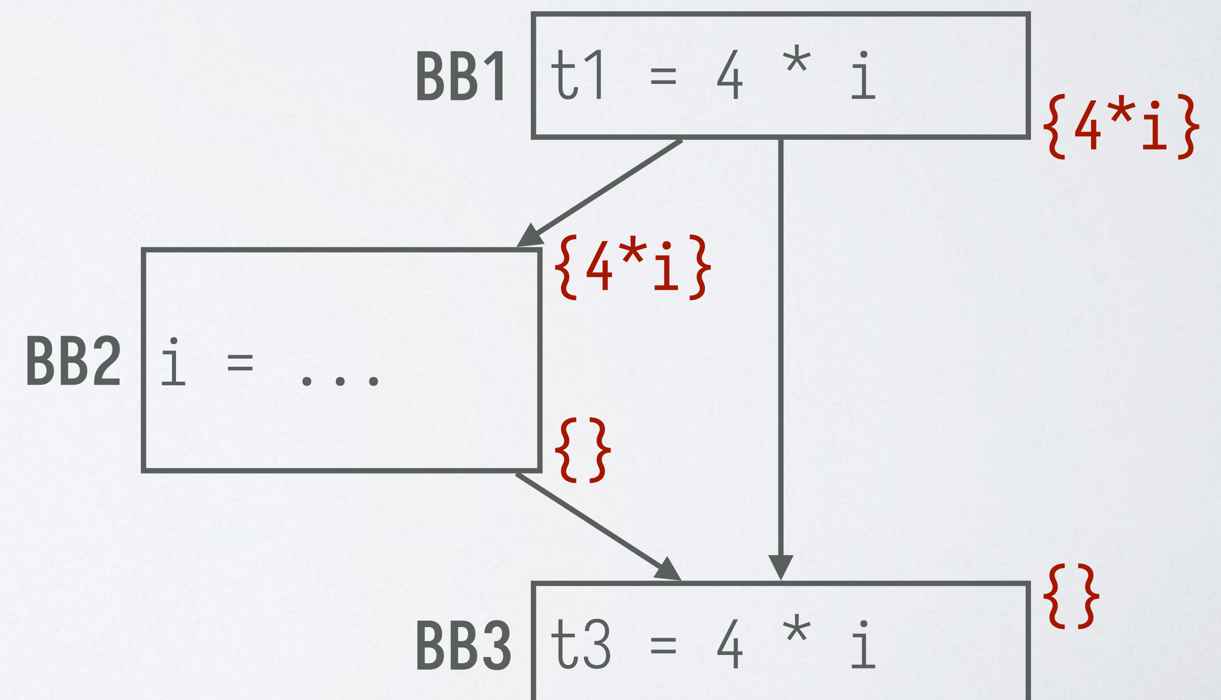
$gen_{\mathrm{BB3}} = \{d_6\}$
$kill_{\mathrm{BB3}} = \{d_3\}$

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

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

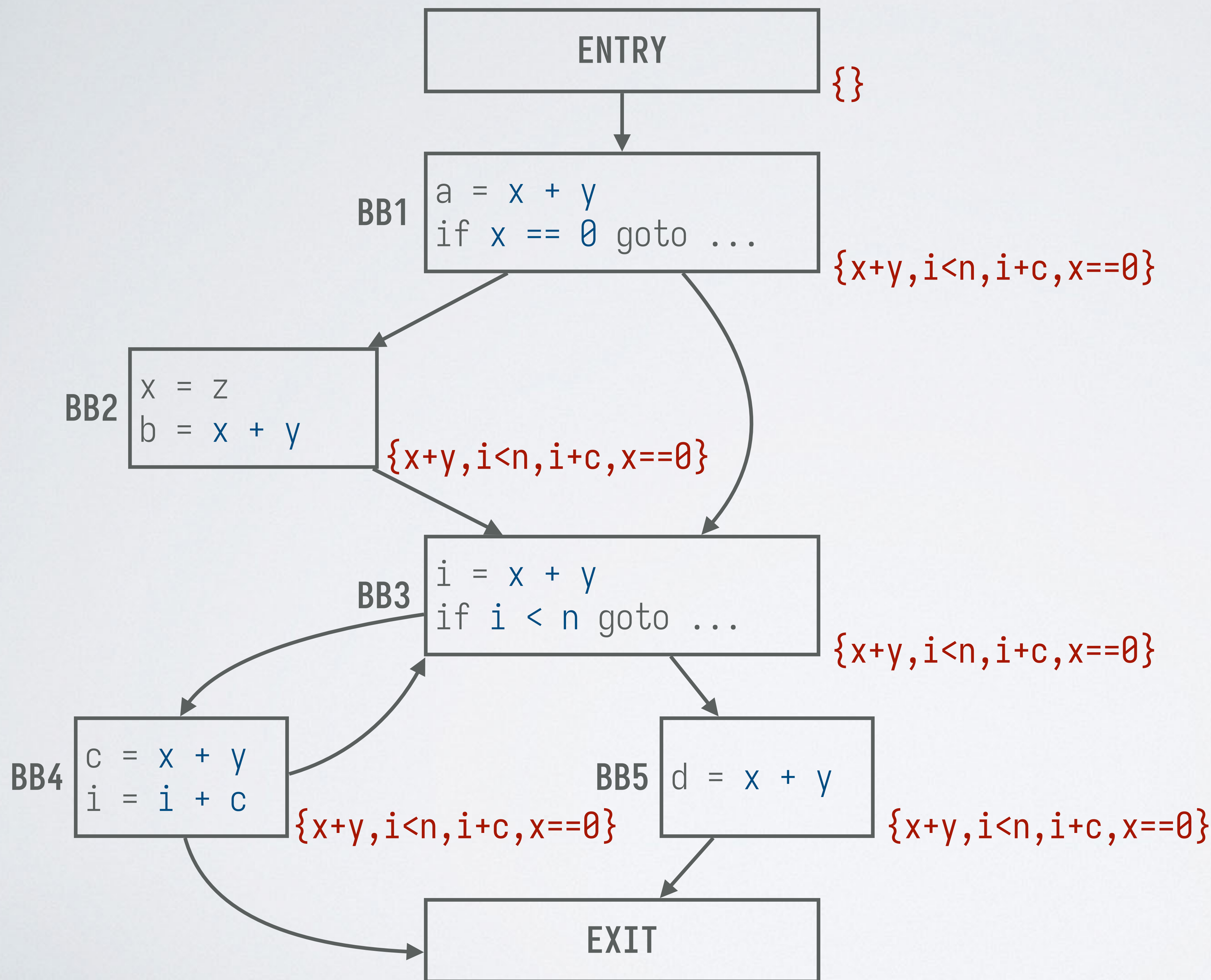




可用表达式分析（2）

- 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*、可达定值分析中的 *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）



$e_gen_{\text{BB1}} = \{\texttt{x+y}, \texttt{x==0}\}$
$e_kill_{\text{BB1}} = \{\}$

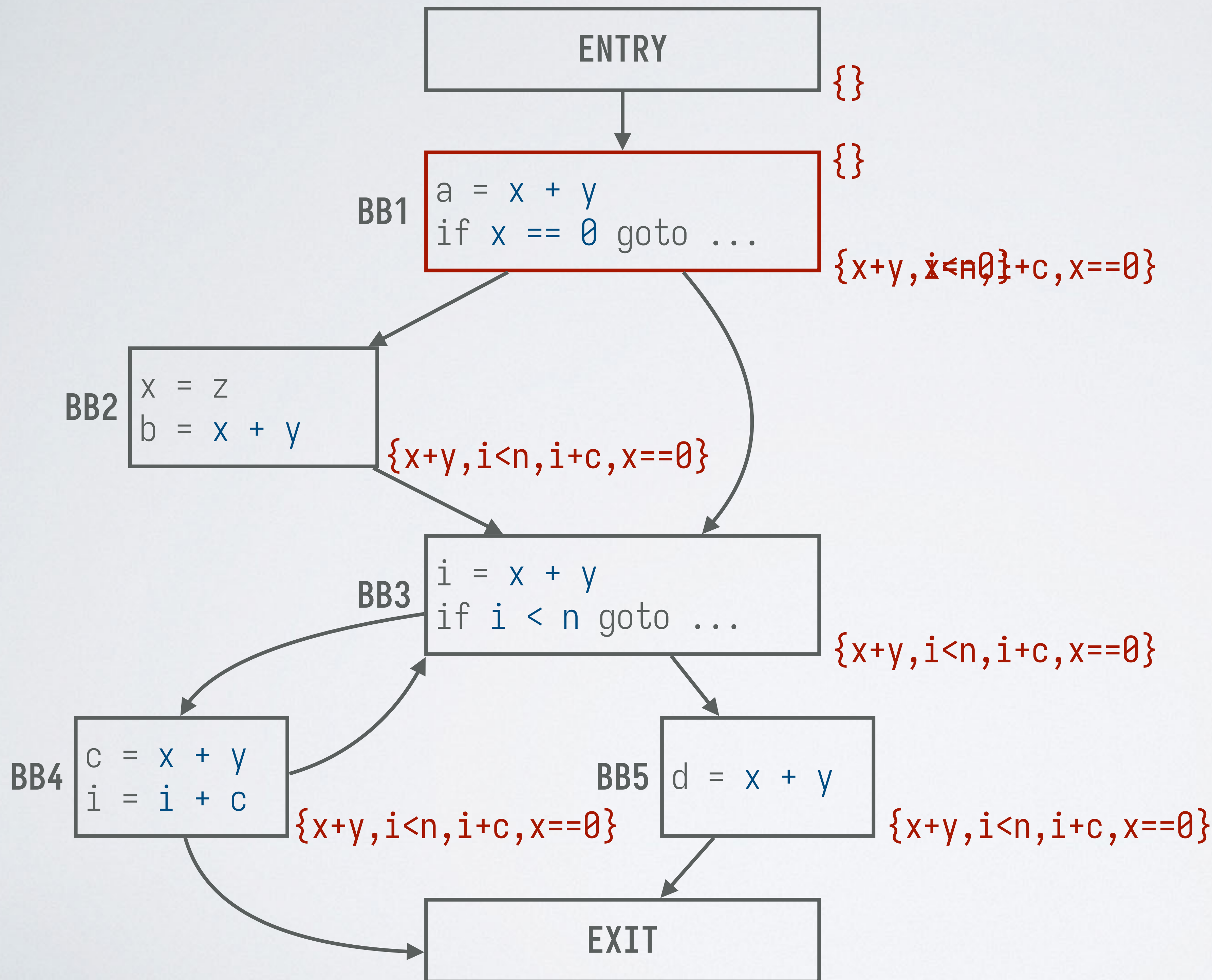
$e_gen_{\text{BB2}} = \{\texttt{x+y}\}$
$e_kill_{\text{BB2}} = \{\texttt{x+y}, \texttt{x==0}\}$

$e_gen_{\text{BB3}} = \{\texttt{x+y}, \texttt{i<n}\}$
$e_kill_{\text{BB3}} = \{\texttt{i<n}, \texttt{i+c}\}$

$e_gen_{\text{BB4}} = \{\texttt{x+y}\}$
$e_kill_{\text{BB4}} = \{\texttt{i<n}, \texttt{i+c}\}$

$e_gen_{\text{BB5}} = \{\texttt{x+y}\}$
$e_kill_{\text{BB5}} = \{\}$

可用表达式分析（4）



$e_gen_{\text{BB1}} = \{\texttt{x+y}, \texttt{x==0}\}$

$e_kill_{\text{BB1}} = \{\}$

$e_gen_{\text{BB2}} = \{\texttt{x+y}\}$

$e_kill_{\text{BB2}} = \{\texttt{x+y}, \texttt{x==0}\}$

$e_gen_{\text{BB3}} = \{\texttt{x+y}, \texttt{i<n}\}$

$e_kill_{\text{BB3}} = \{\texttt{i<n}, \texttt{i+c}\}$

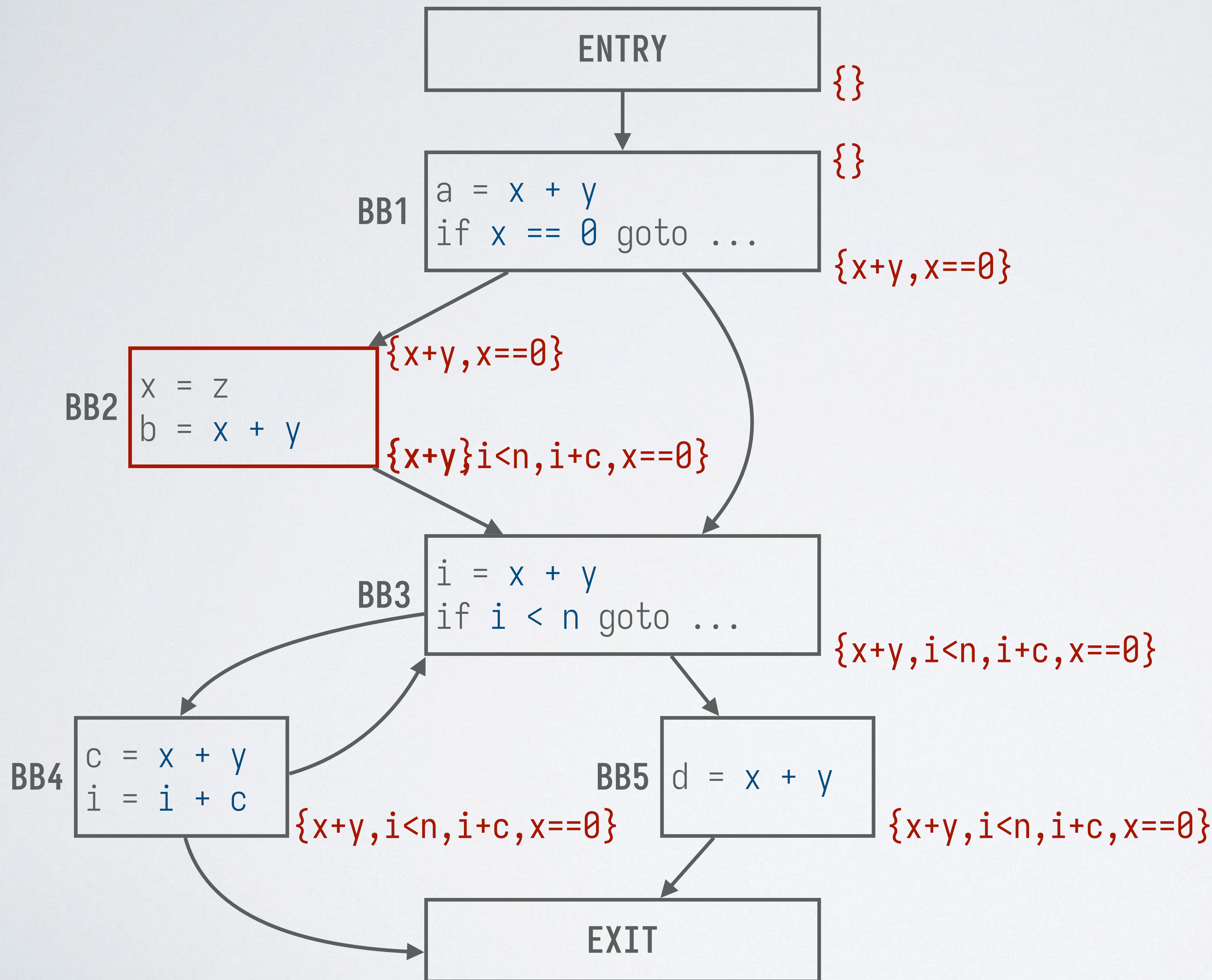
$e_gen_{\text{BB4}} = \{\texttt{x+y}\}$

$e_kill_{\text{BB4}} = \{\texttt{i<n}, \texttt{i+c}\}$

$e_gen_{\text{BB5}} = \{\texttt{x+y}\}$

$e_kill_{\text{BB5}} = \{\}$

可用表达式分析（5）



$e_gen_{\mathrm{BB1}} = \{\texttt{x+y}, \texttt{x==0}\}$

$e_kill_{\mathrm{BB1}} = \{\}$

$e_gen_{\mathrm{BB2}} = \{\texttt{x+y}\}$

$e_kill_{\mathrm{BB2}} = \{\texttt{x+y}, \texttt{x==0}\}$

$e_gen_{\mathrm{BB3}} = \{\texttt{x+y}, \texttt{i<n}\}$

$e_kill_{\mathrm{BB3}} = \{\texttt{i<n}, \texttt{i+c}\}$

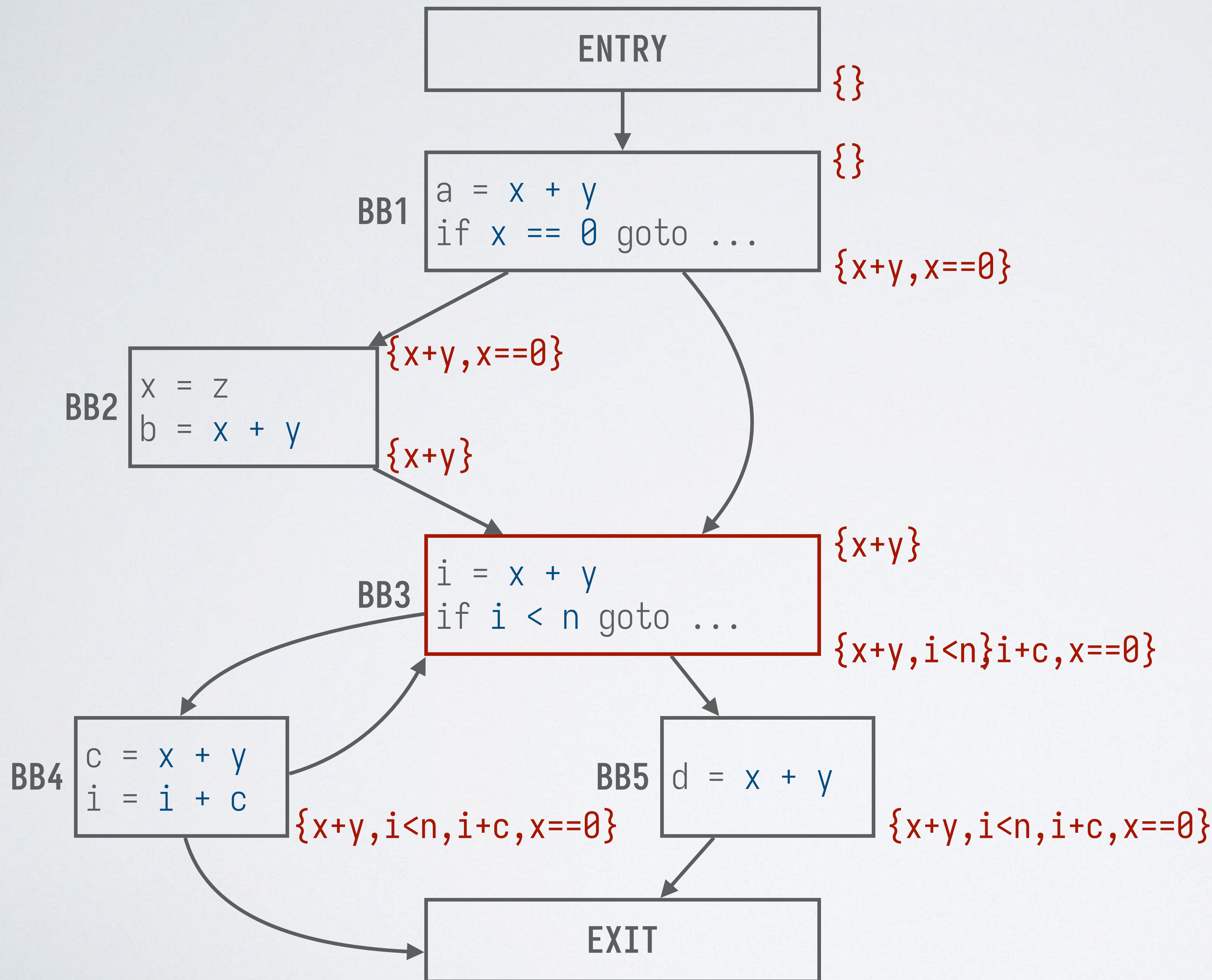
$e_gen_{\mathrm{BB4}} = \{\texttt{x+y}\}$

$e_kill_{\mathrm{BB4}} = \{\texttt{i<n}, \texttt{i+c}\}$

$e_gen_{\mathrm{BB5}} = \{\texttt{x+y}\}$

$e_kill_{\mathrm{BB5}} = \{\}$

可用表达式分析（6）



$e_gen_{\text{BB1}} = \{\text{x+y, x==0}\}$

$e_kill_{\text{BB1}} = \{\}$

$e_gen_{\text{BB2}} = \{\text{x+y}\}$

$e_kill_{\text{BB2}} = \{\text{x+y, x==0}\}$

$e_gen_{\text{BB3}} = \{\text{x+y, i<n}\}$

$e_kill_{\text{BB3}} = \{\text{i<n, i+c}\}$

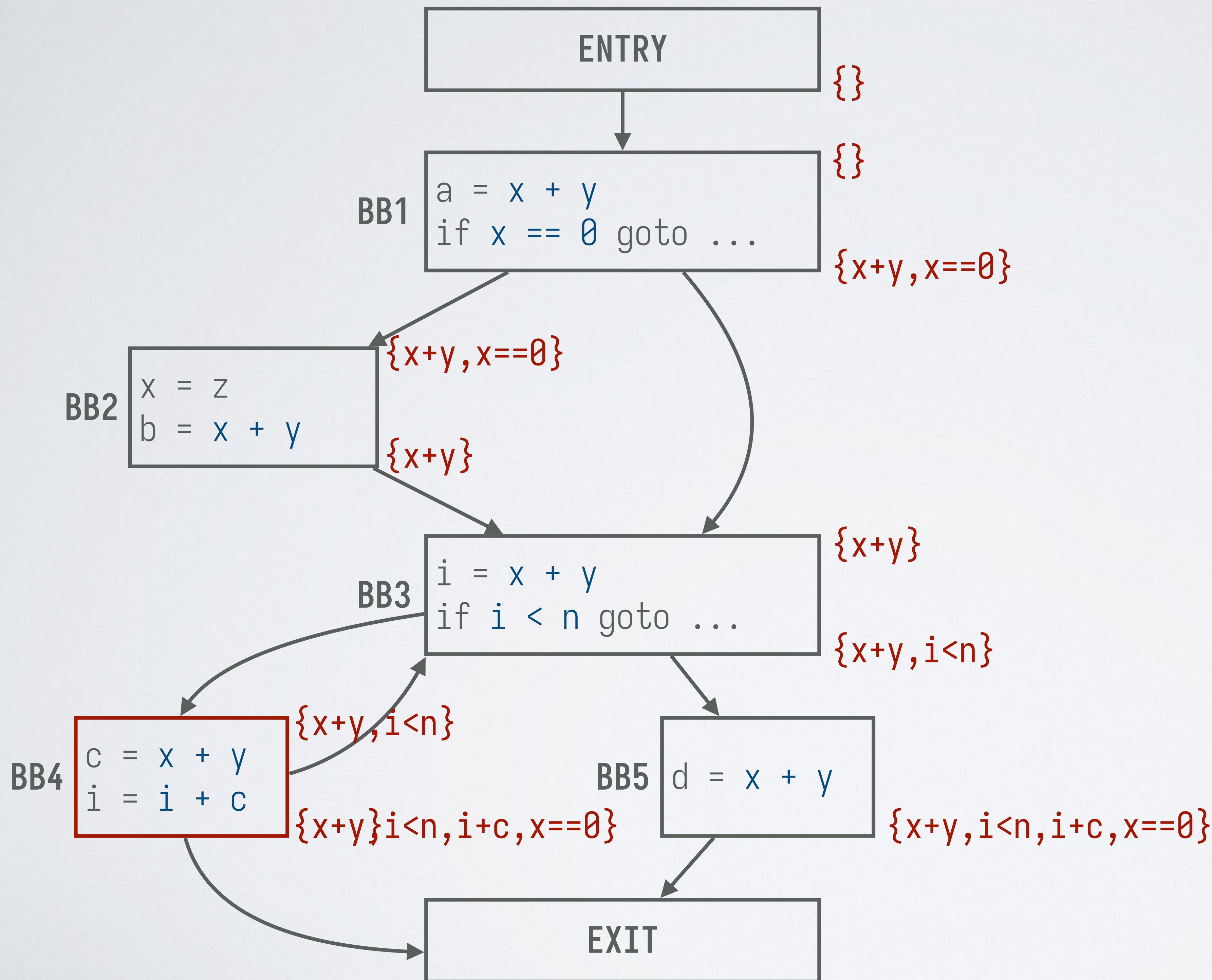
$e_gen_{\text{BB4}} = \{\text{x+y}\}$

$e_kill_{\text{BB4}} = \{\text{i<n, i+c}\}$

$e_gen_{\text{BB5}} = \{\text{x+y}\}$

$e_kill_{\text{BB5}} = \{\}$

可用表达式分析（7）



$e_gen_{\text{BB1}} = \{\text{x+y}, \text{x==0}\}$

$e_kill_{\text{BB1}} = \{\}$

$e_gen_{\text{BB2}} = \{\text{x+y}\}$

$e_kill_{\text{BB2}} = \{\text{x+y}, \text{x==0}\}$

$e_gen_{\text{BB3}} = \{\text{x+y}, \text{i<n}\}$

$e_kill_{\text{BB3}} = \{\text{i<n}, \text{i+c}\}$

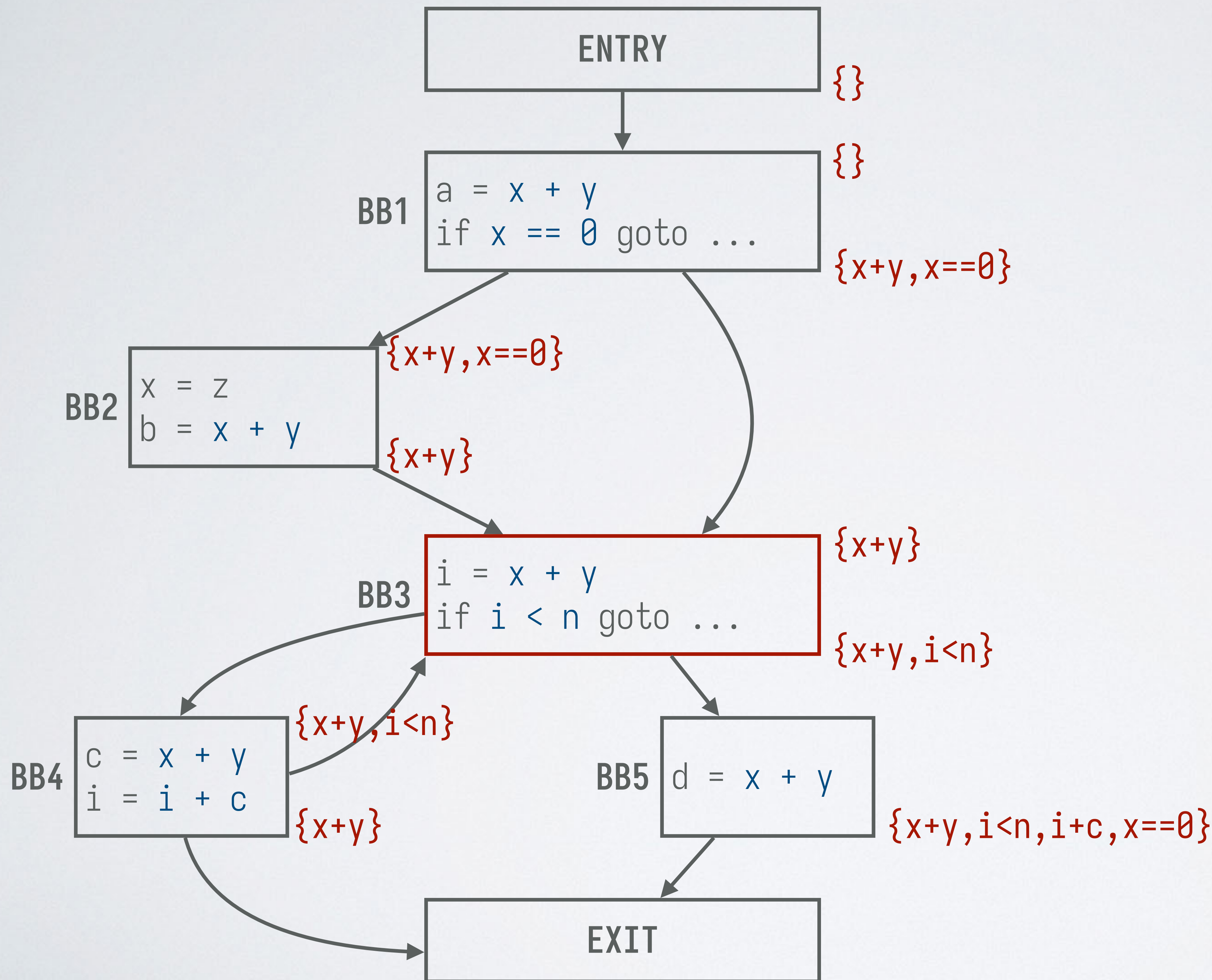
$e_gen_{\text{BB4}} = \{\text{x+y}\}$

$e_kill_{\text{BB4}} = \{\text{i<n}, \text{i+c}\}$

$e_gen_{\text{BB5}} = \{\text{x+y}\}$

$e_kill_{\text{BB5}} = \{\}$

可用表达式分析（8）



$e_gen_{BB1} = \{x+y, x==0\}$

$e_kill_{BB1} = \{\}$

$e_gen_{BB2} = \{x+y\}$

$e_kill_{BB2} = \{x+y, x==0\}$

$e_gen_{BB3} = \{x+y, i<n\}$

$e_kill_{BB3} = \{i<n, i+c\}$

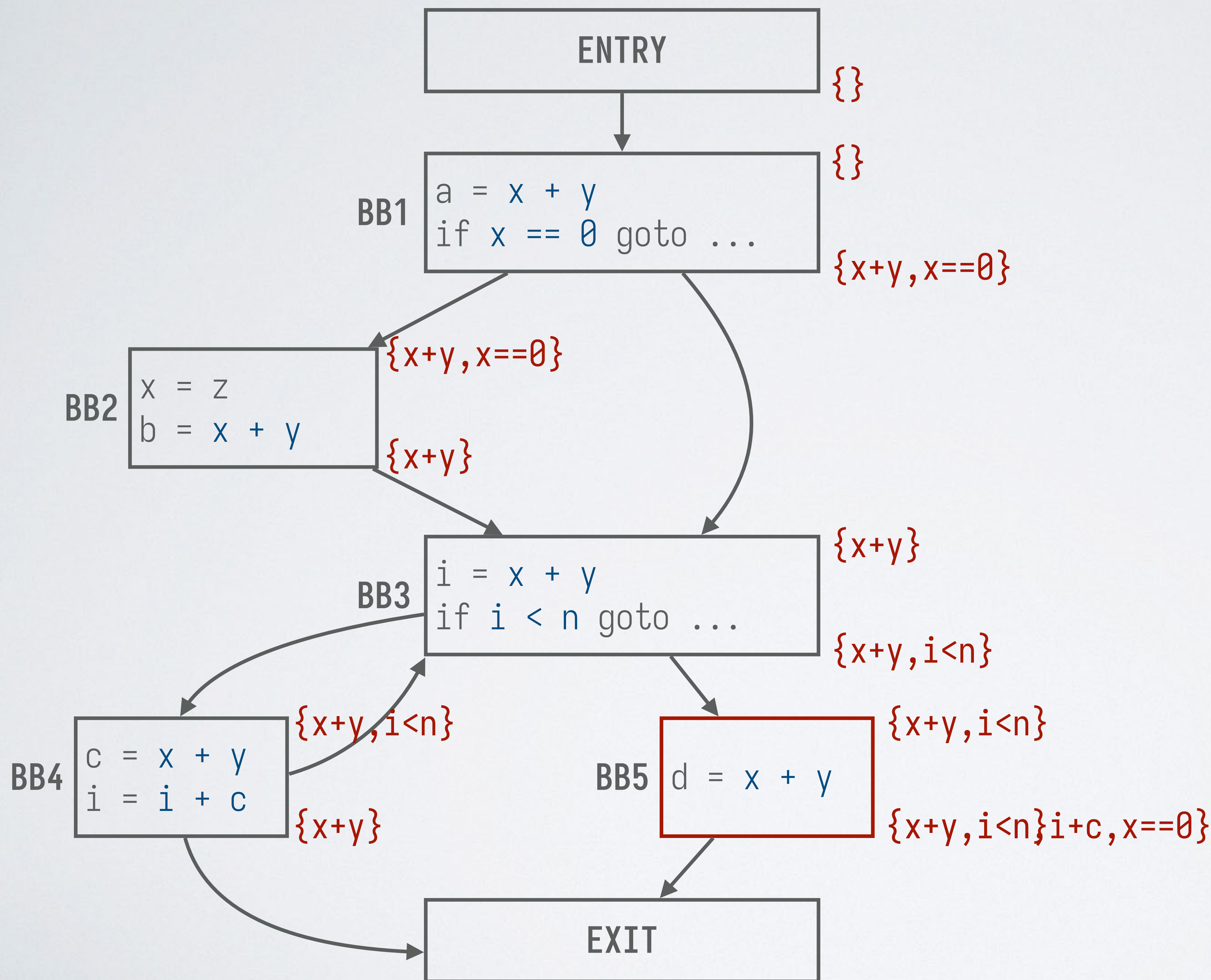
$e_gen_{BB4} = \{x+y\}$

$e_kill_{BB4} = \{i<n, i+c\}$

$e_gen_{BB5} = \{x+y\}$

$e_kill_{BB5} = \{\}$

可用表达式分析（9）



$e_gen_{\text{BB1}} = \{\texttt{x+y}, \texttt{x==0}\}$

$e_kill_{\text{BB1}} = \{\}$

$e_gen_{\text{BB2}} = \{\texttt{x+y}\}$

$e_kill_{\text{BB2}} = \{\texttt{x+y}, \texttt{x==0}\}$

$e_gen_{\text{BB3}} = \{\texttt{x+y}, \texttt{i<n}\}$

$e_kill_{\text{BB3}} = \{\texttt{i<n}, \texttt{i+c}\}$

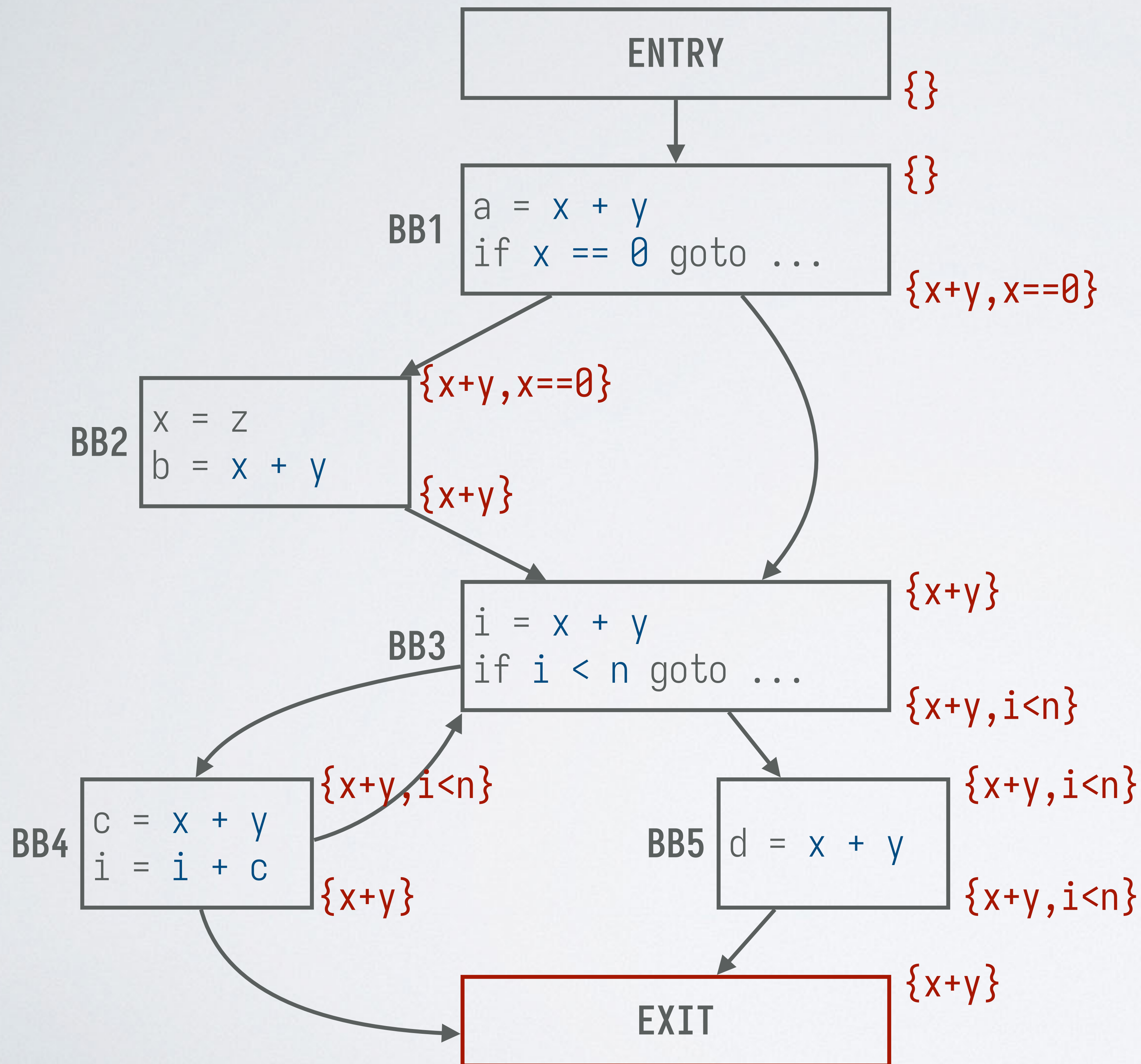
$e_gen_{\text{BB4}} = \{\texttt{x+y}\}$

$e_kill_{\text{BB4}} = \{\texttt{i<n}, \texttt{i+c}\}$

$e_gen_{\text{BB5}} = \{\texttt{x+y}\}$

$e_kill_{\text{BB5}} = \{\}$

可用表达式分析（10）



$e_gen_{\mathrm{BB1}} = \{$x+y, x==0$\}$
$e_kill_{\mathrm{BB1}} = \{\}$

$e_gen_{\mathrm{BB2}} = \{$x+y$\}$
$e_kill_{\mathrm{BB2}} = \{$x+y, x==0$\}$

$e_gen_{\mathrm{BB3}} = \{$x+y, i<n$\}$
$e_kill_{\mathrm{BB3}} = \{$i<n, i+c$\}$

$e_gen_{\mathrm{BB4}} = \{$x+y$\}$
$e_kill_{\mathrm{BB4}} = \{$i<n, i+c$\}$

$e_gen_{\mathrm{BB5}} = \{$x+y$\}$
$e_kill_{\mathrm{BB5}} = \{\}$

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

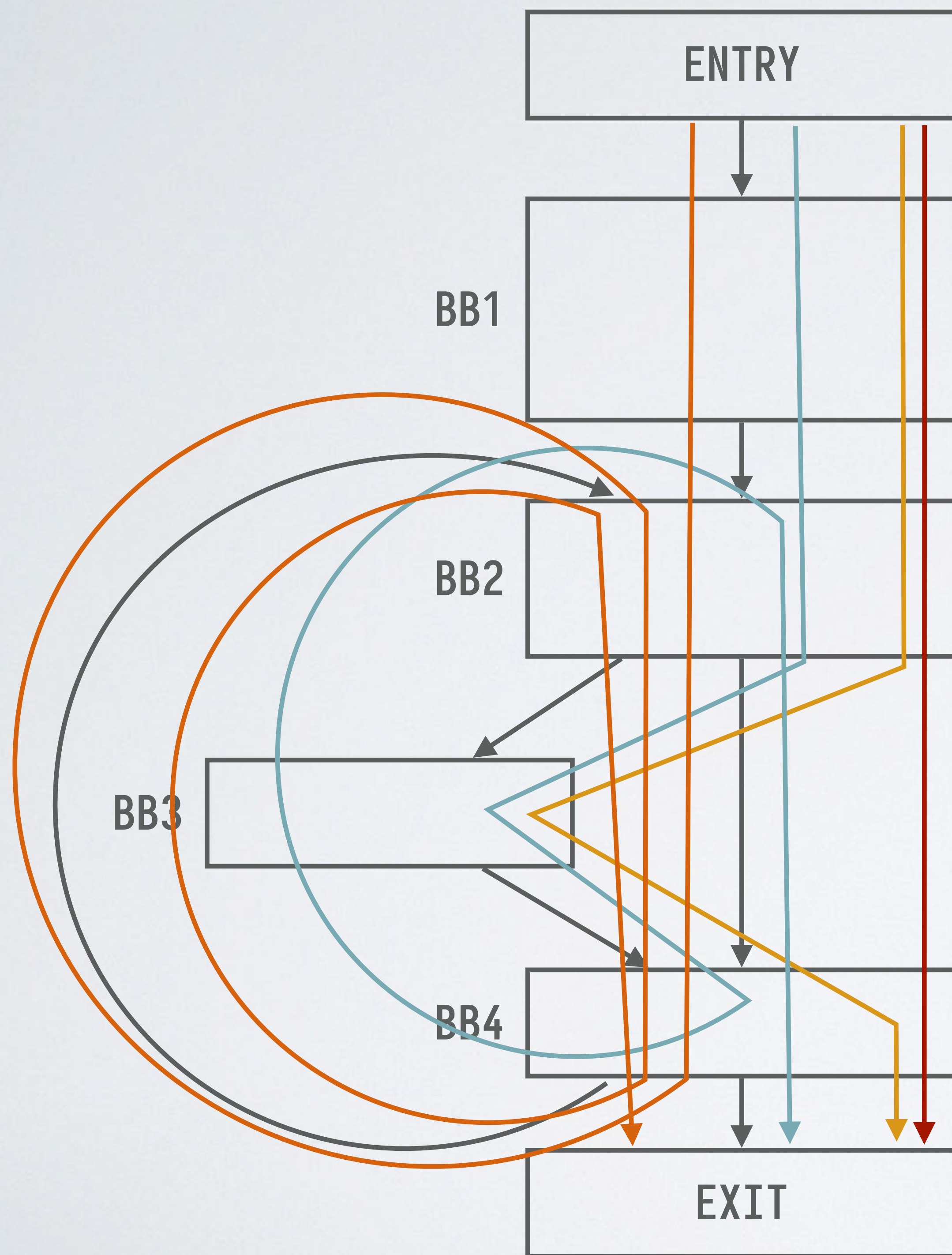
	活跃变量	可达定值	可用表达式
域	变量的集合	定值的集合	表达式的集合
方向	后向	前向	前向
传递函数	$(O - def_B) \cup use_B$	$(I - kill_B) \cup gen_B$	$(I - e_kill_B) \cup e_gen_B$
边界条件	$\mathrm{IN}[\mathrm{EXIT}] = \emptyset$	$\mathrm{OUT}[\mathrm{ENTRY}] = \emptyset$	$\mathrm{OUT}[\mathrm{ENTRY}] = \emptyset$
交汇运算	$\cup$	$\cup$	$\cap$
方程组	$\mathrm{OUT}[B]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

主要内容

- 代码优化的常用方法
- 局部优化：基本块的优化
- 局部优化：窥孔优化
- 全局优化：数据流分析
- One More Thing

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

  ……

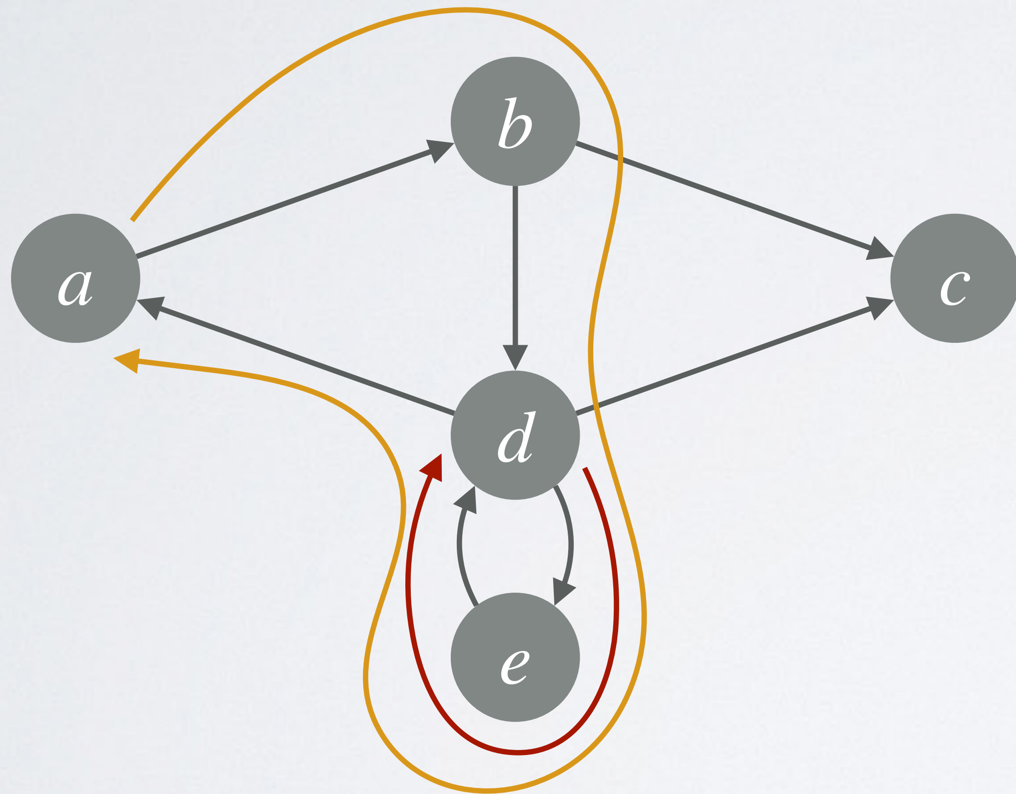
- 流图上可以有**无限多条**路径
- **是否可以用一个有限的形式刻画一个无限的集合？**

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 $L(\epsilon) = \{\epsilon\}$
  - 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 $L(r_1 \mid r_2) = L(r_1) \cup L(r_2)$
    - $r_1 r_2$ 表示语言 $L(r_1 r_2) = \{xy \mid x \in L(r_1), y \in L(r_2)\}$
    - r_1^* 表示语言 $L(r_1^*) = \{x_1 x_2 \cdots x_n \mid n \geq 0, x_1, x_2, \cdots, x_n \in L(r_1)\}$
- **例子：** 0(0 | 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**



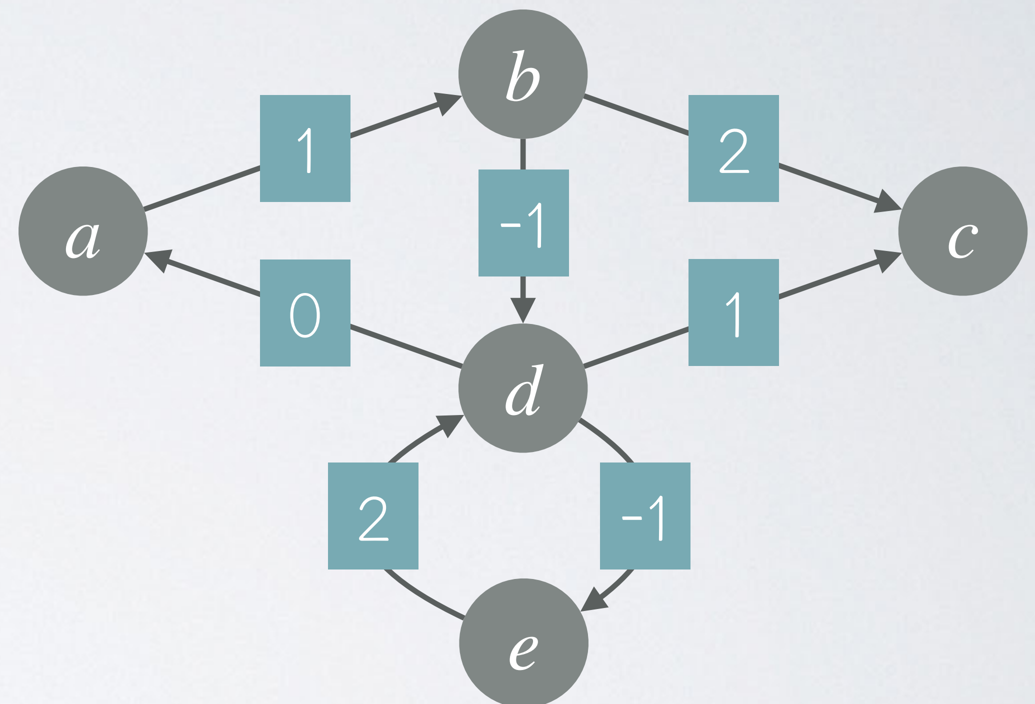
字母表 $\Sigma = \{\langle a,b\rangle, \langle b,d\rangle, \langle b,c\rangle, \langle d,a\rangle, \langle d,c\rangle, \langle d,e\rangle, \langle e,d\rangle\}$

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）

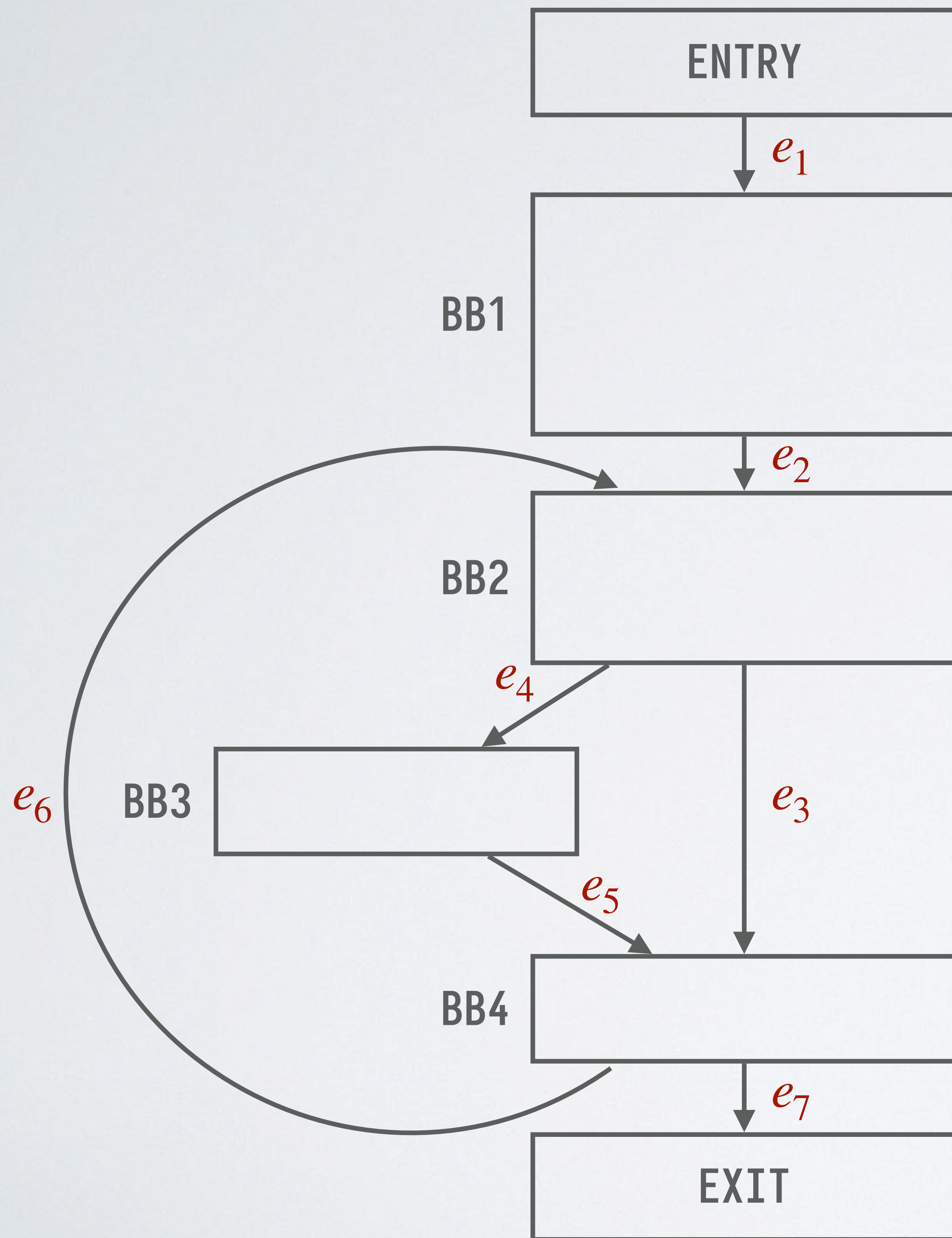
- 用 $F(r)$ 表示 r 能识别的路径的长度的最小值
  - $F(\epsilon) = 0$, $F(\langle u_1, u_2 \rangle) =$ 边 $\langle u_1, u_2 \rangle$ 的长度
  - $F(r_1 \mid r_2) = \min(F(r_1), F(r_2))$
  - $F(r_1 r_2) = F(r_1) + F(r_2)$
  - $F(r_1^*) = ?$
    - 若 $F(r_1) < 0$, 则 $F(r_1^*) = -\infty$
    - 否则 $F(r_1^*) = 0$
- **例子：**
  - $F(\langle d, e \rangle) = -1$, $F(\langle e, d \rangle) = 2$
  - $F(\langle d, e \rangle \langle e, d \rangle) = (-1) + 2 = 1$
  - $F((\langle d, e \rangle \langle e, d \rangle)^*) = 0$



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

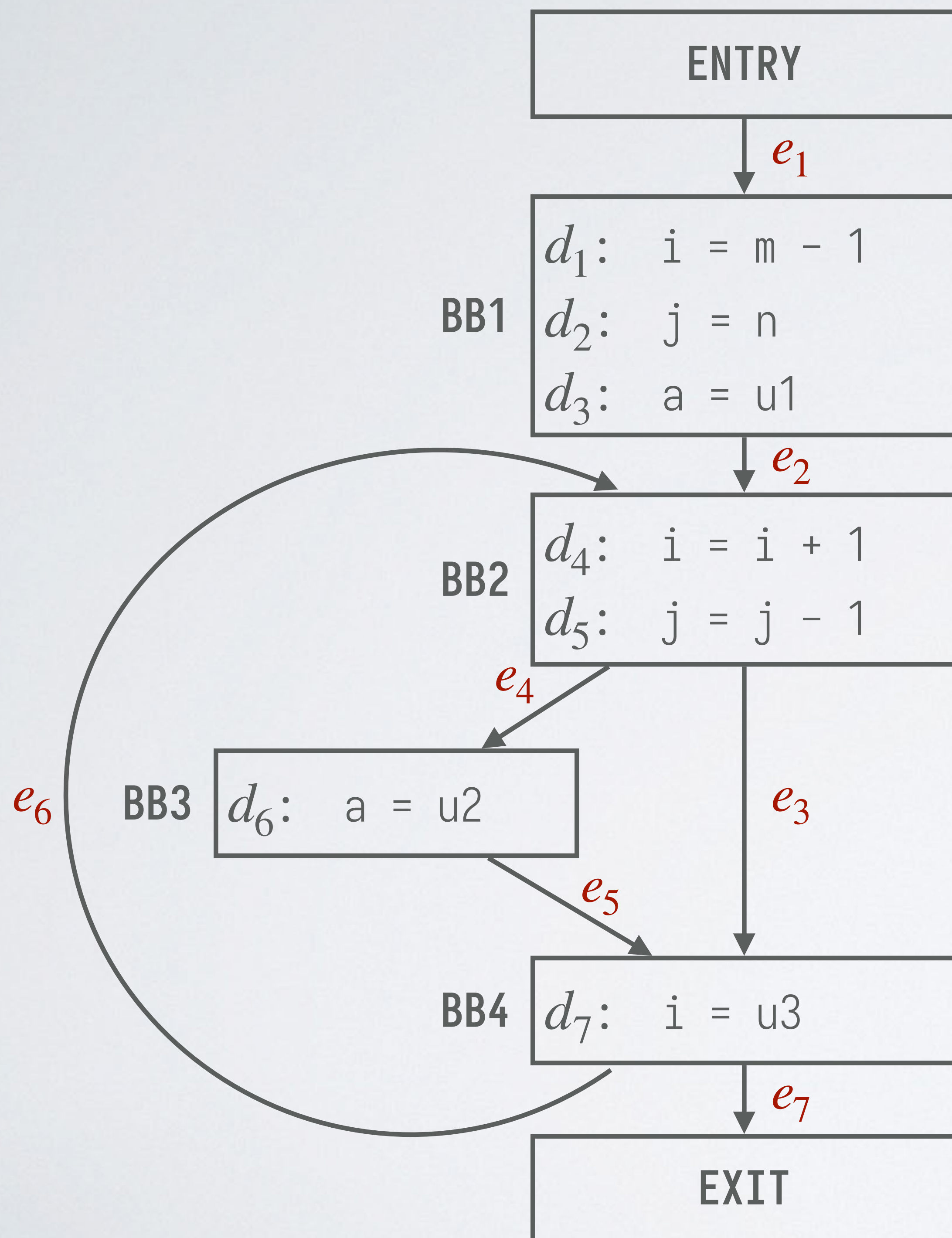


- 给每条边一个**标号**
- **前向分析**：从 ENTRY 出发的路径
  - BB1: e_1
  - BB2: $e_1e_2((e_3 \mid e_4e_5)e_6)^*$
  - BB3: $e_1e_2((e_3 \mid e_4e_5)e_6)^*e_4$
  - BB4: $e_1e_2((e_3 \mid e_4e_5)e_6)^*(e_3 \mid e_4e_5)$
  - EXIT: $e_1e_2((e_3 \mid e_4e_5)e_6)^*(e_3 \mid e_4e_5)e_7$
- **后向分析**：到达 EXIT 的路径
  - BB4: $(e_6(e_3 \mid e_4e_5))^*e_7$
  - 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
  - $F(\epsilon) =$ 恒等函数, $F(e) = f_{h(e)}$ 其中 $h(e)$ 是边 e 的**起点基本块**
  - $F(r_1 | r_2) = F(r_1) \wedge F(r_2)$
  - $F(r_1 r_2) = F(r_2) \circ F(r_1)$
  - $F(r_1^*) = \bigwedge_{i \geq 0} F(r_1)^i$, 不过有时能找到更高效的算法

例子：可达定值分析



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

本讲小结

- 代码优化的常用方法
  - 常量传播和折叠、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)
- 基本块的优化
  - 基本块 DAG 的构造
  - 局部公共子表达式消除、死代码消除、代数恒等式
- 窥孔优化
  - 目标代码优化的常用方法
- 数据流分析
  - 可达定值、活跃变量、可用表达式的基本算法

思考问题

- 现代编译器通常提供不同的优化级别(如 -O1, -O2, -O3)，它们有何不同之处？实际使用时怎么选择合适的优化级别？
- 代码优化有时需要在优化后代码的执行时间和大小之间做出权衡。在什么场景下需要考虑这种权衡？
- 局部优化和全局优化的区别是什么？不同的编译阶段应该倾向于采取哪种策略？是否有优化只能在全局范围内有效？
- 为什么循环优化对于提升程序性能尤为重要？除了代码外提、强度消减外，你还能想到什么循环优化策略？
- 人工智能时代，是否能够面向机器学习代码设计新的优化？