Compiladores
(IF688)

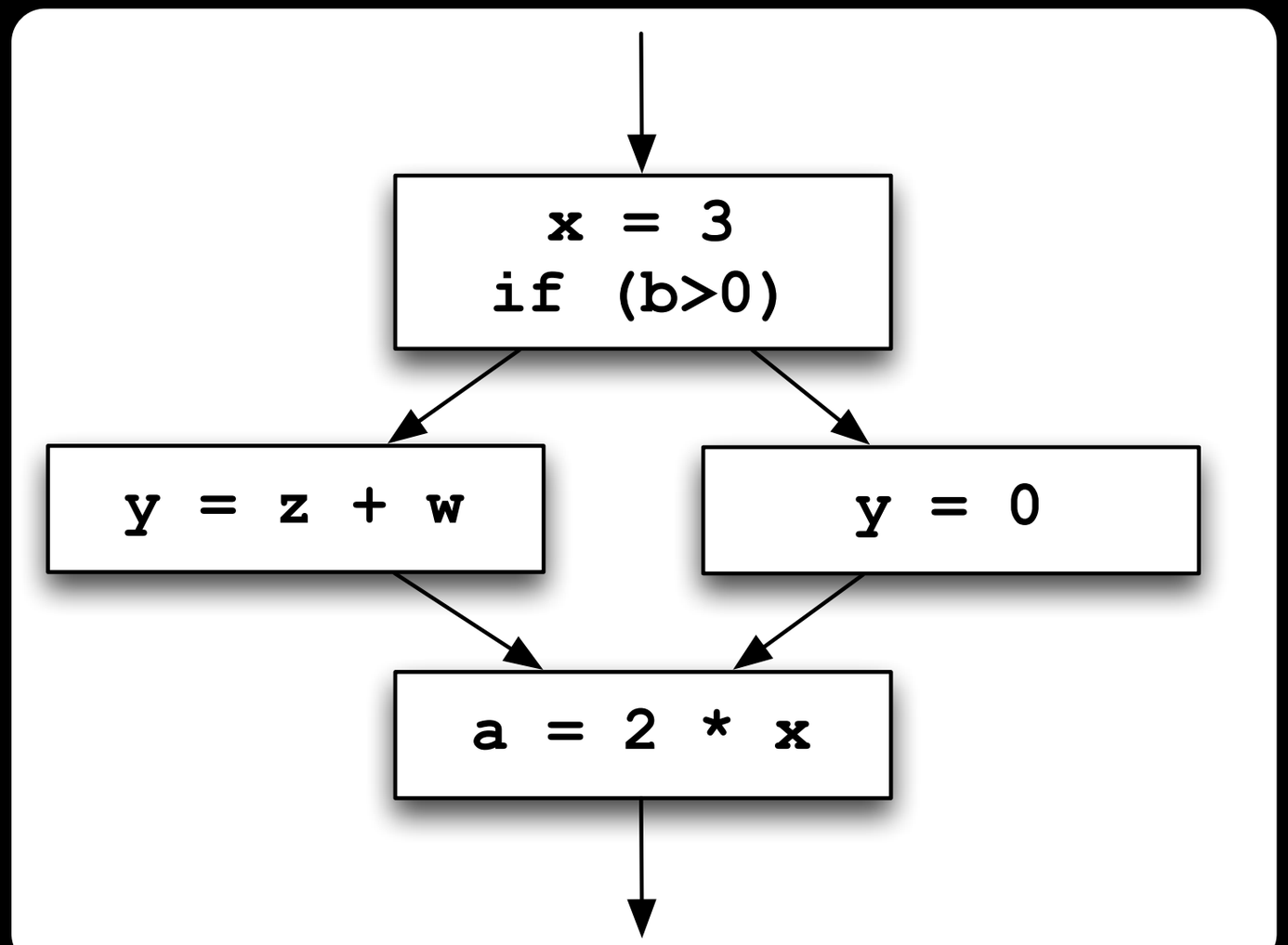
Leopoldo Teixeira
lmt@cin.ufpe.br | @leopoldomt

Forward-flow

- Toda a informação é propagada de maneira *forward*
- Ou seja, o *out* de um bloco é definido com base no *in*
  - na entrada de um bloco que é sucessor de vários, observamos se há alguma divergência

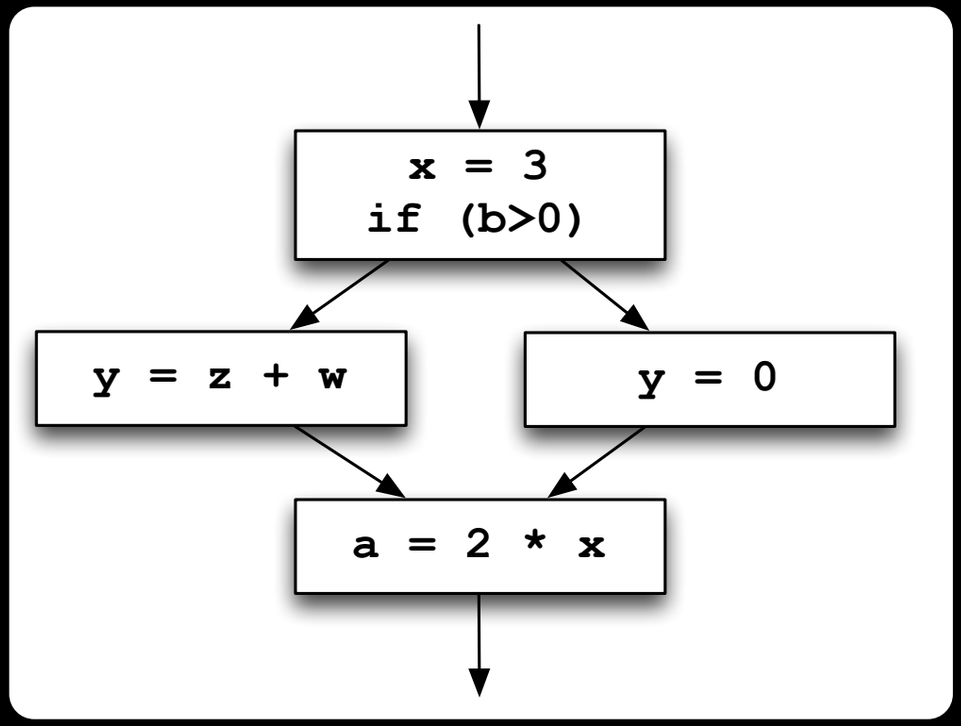
Liveness

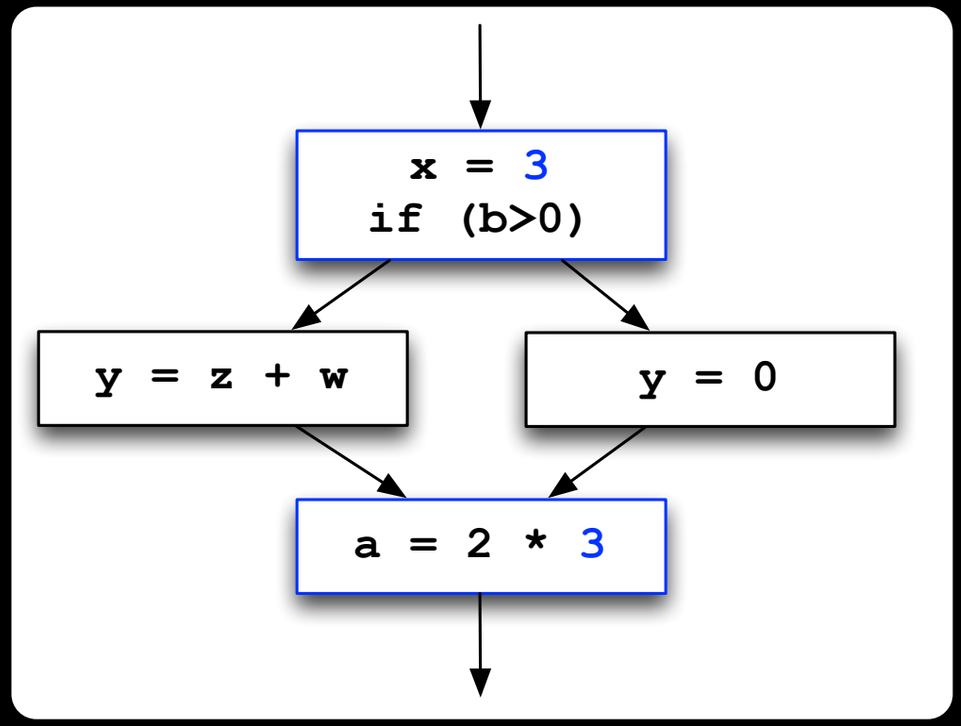
Uma vez que constantes tenham sido propagadas, gostaríamos de eliminar *dead code*



Liveness

Uma vez que constantes tenham sido propagadas, gostaríamos de eliminar *dead code*

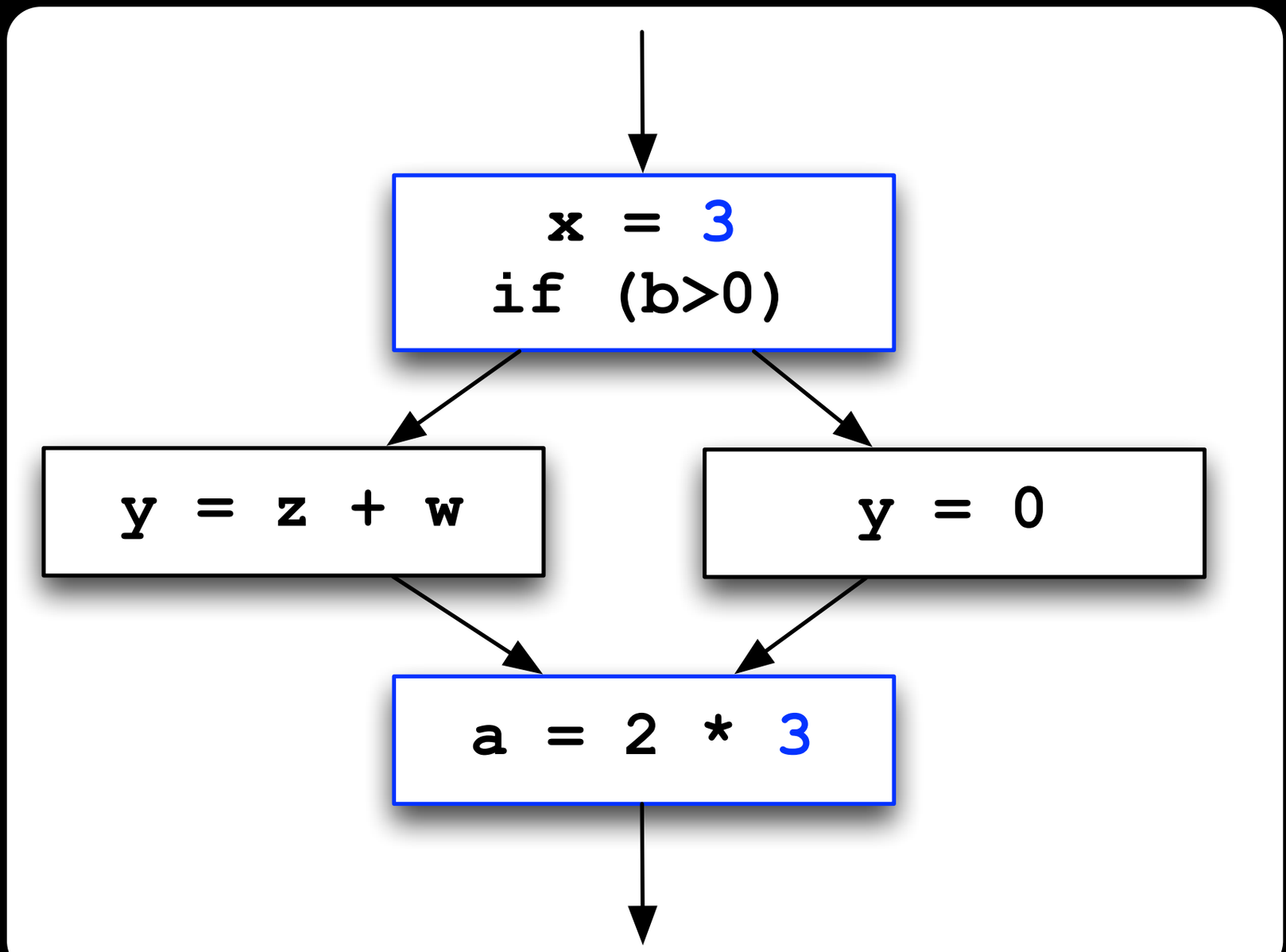




Após *constant propagation*, x=3 é *dead code*

(assumindo que este seja o CFG inteiro)

Liveness

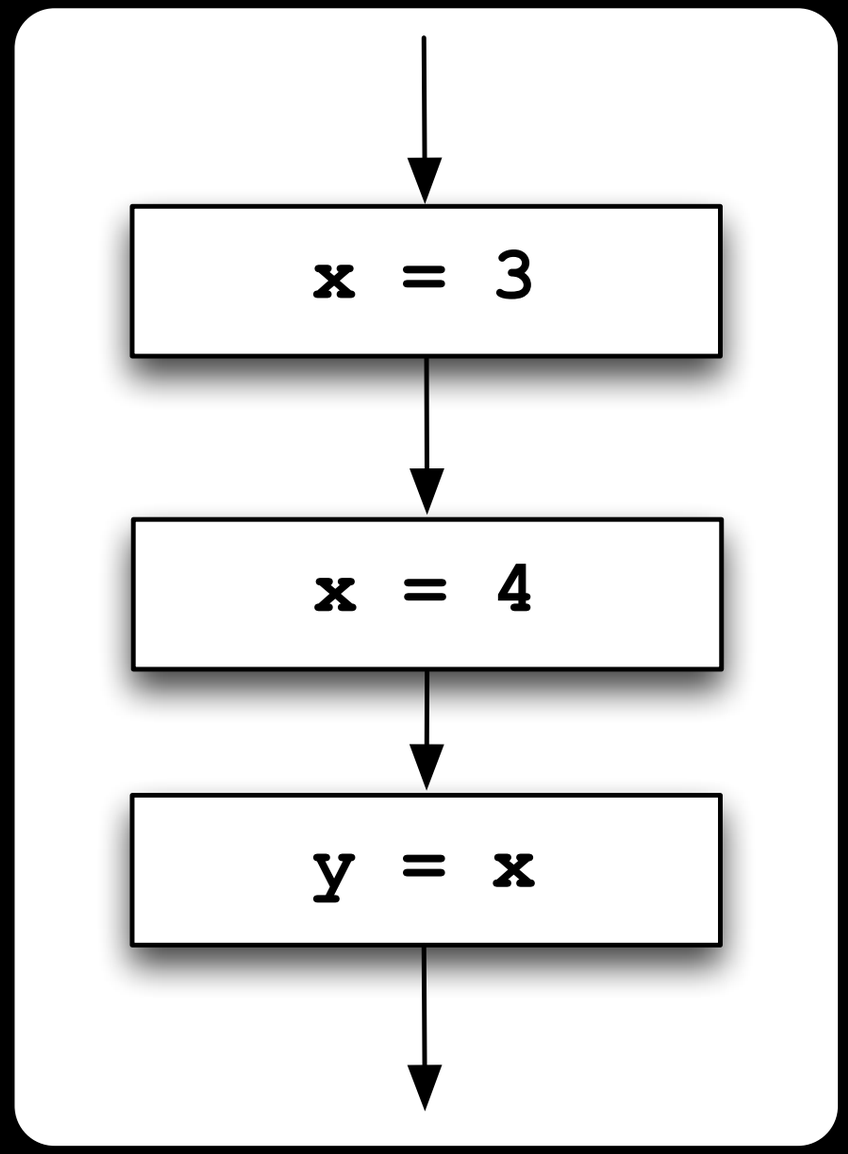


Liveness

- Utilizamos a análise de live variables para saber se em um dado ponto **p**, uma variável **x** pode ser usada em um dos caminhos a partir de **p**
  - dependendo da resposta **x** é *live* ou *dead*
  - análise é *backwards*

Live vs. Dead

- O primeiro valor de x é *dead*
  - (nunca usado)
- O segundo valor de x é *live*
  - (pode ser usado)
- *Liveness* é um conceito importante de compiladores e análise de programas



Como seria a definição de *liveness* para uma variável **`x`** em um *statement* **`s`**?

Liveness

- Uma variável **`x`** é viva em um *statement* **`s`** se:
  - Existe um statement **`s′`** que usa **`x`**
  - Existe um caminho de **`s′`** a **`s`**;
  - Este caminho não tem nenhuma atribuição a **`x`**

Global Dead Code Elimination

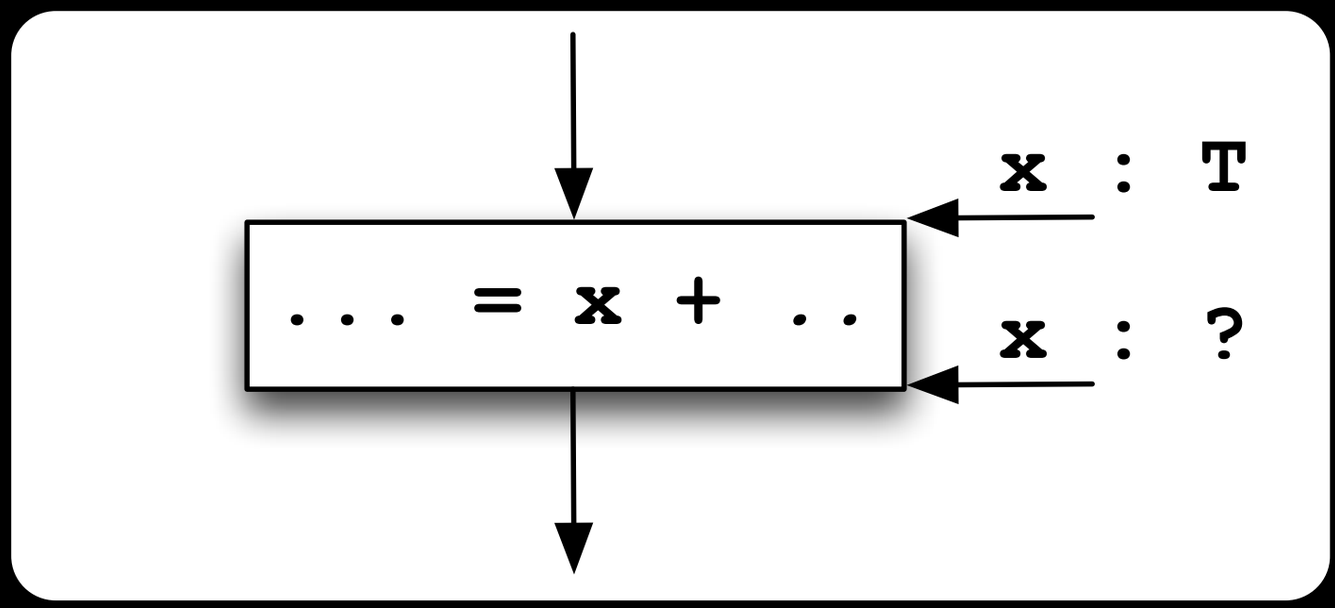
- Uma instrução **`x:=`**... é *dead code* se **`x`** está morta após esta atribuição
- Podemos eliminar código morto do programa
- No entanto, precisamos da informação de *liveness* a priori

Como computar
liveness?

Computando *Liveness*

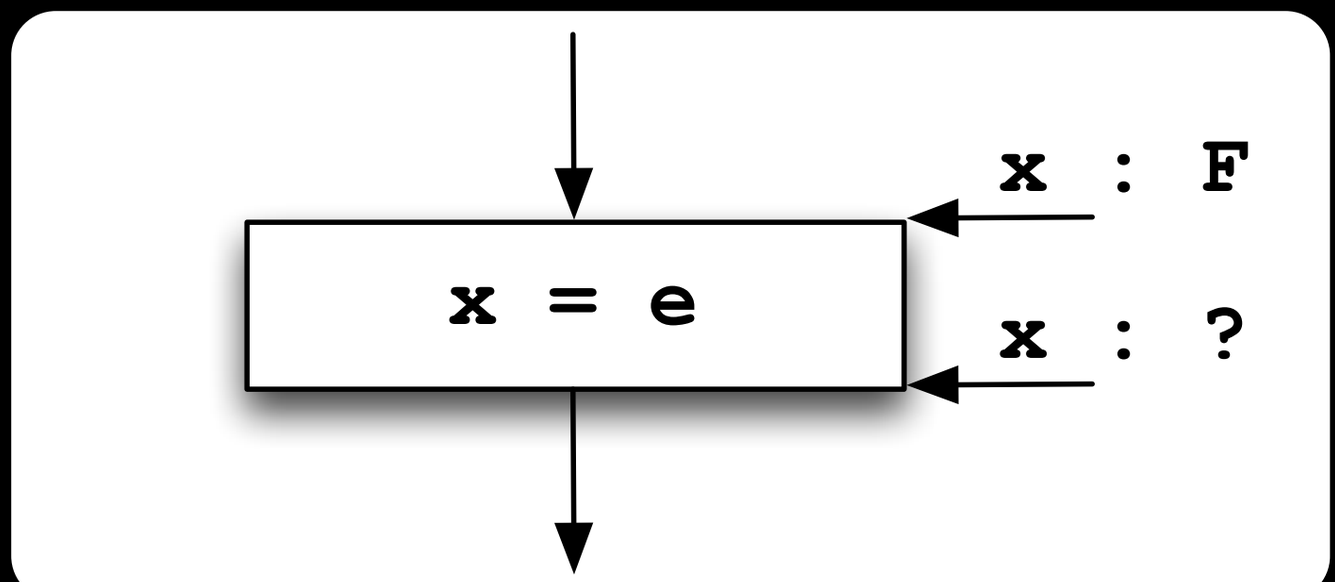
- Podemos expressar *liveness* em termos de informação transferida entre *statements*
  - assim como em *constant propagation*
- *Liveness* é até mais simples, por ser uma propriedade booleana

Regra 1



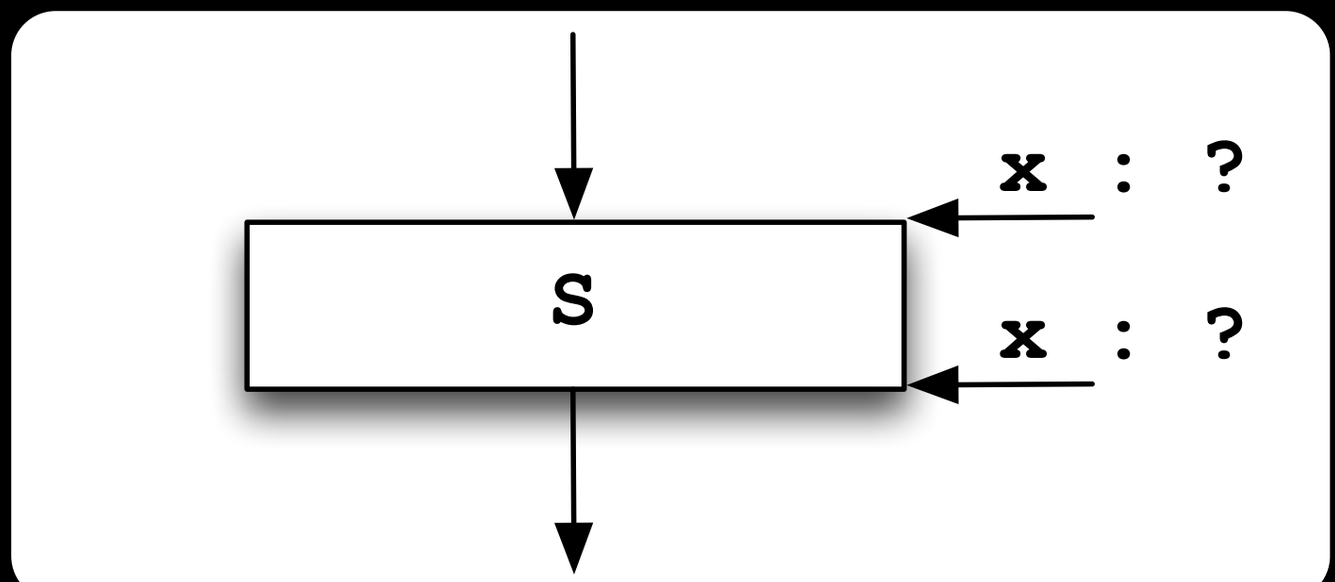
$L_{in}(x,s)$ = true,
se s referencia x no rhs

Regra 2



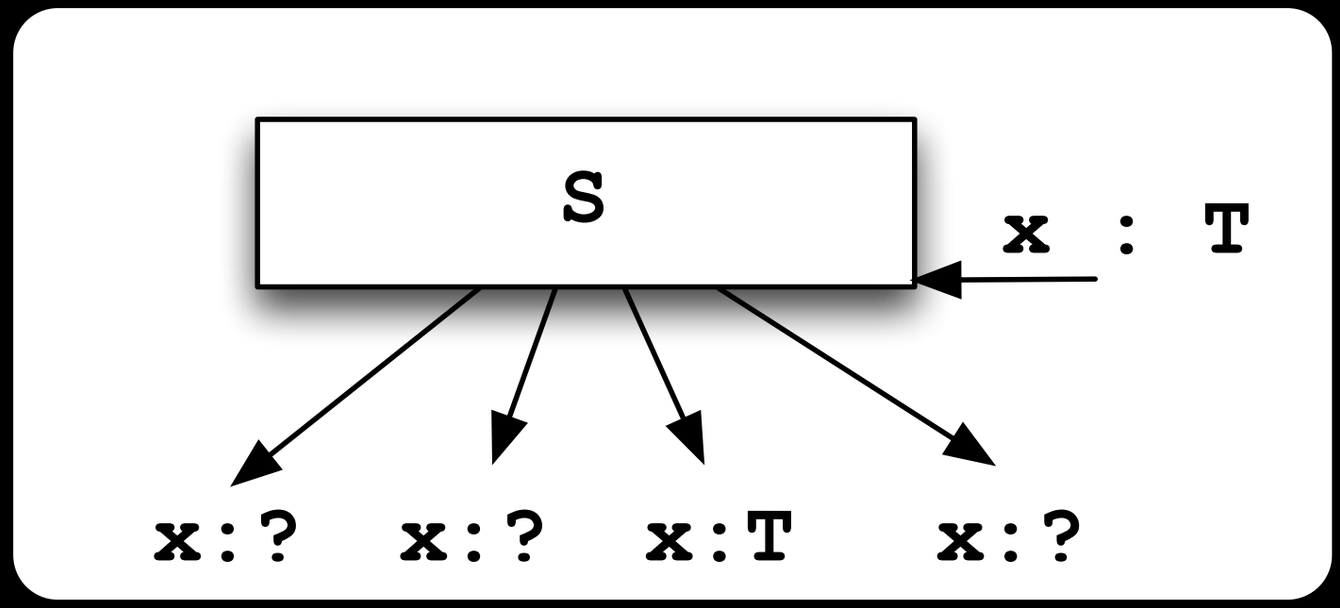
L_{in}(x,x:=e) = false,
se e não referencia x

Regra 3



$L_{in}(x,s) = L_{out}(x,s)$,
se s não referencia x

Regra 4

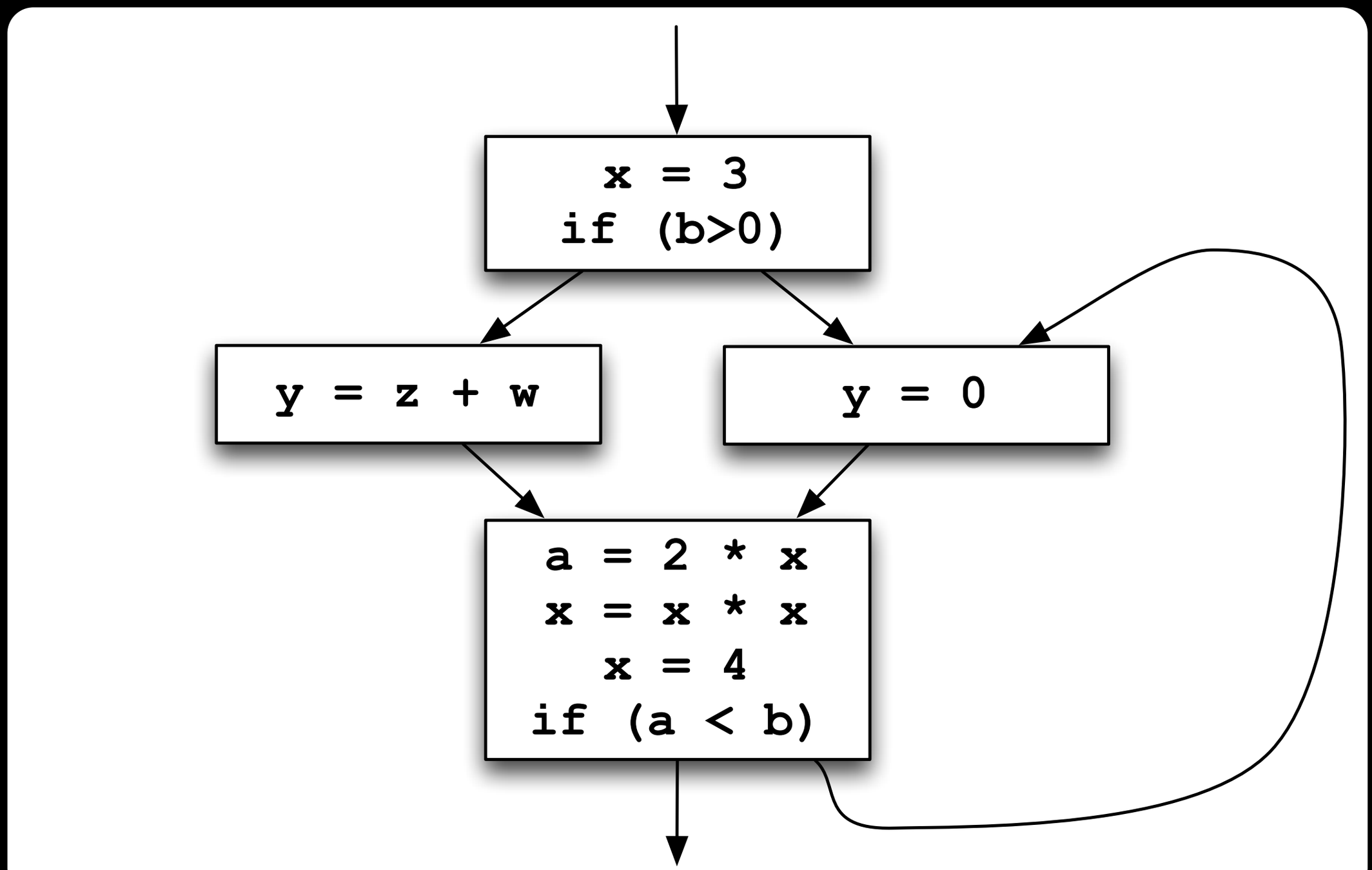


$L_{out}(x,p) = \bigvee\{L_{in}(x,s) \mid s \text{ é sucessor de } p\}$

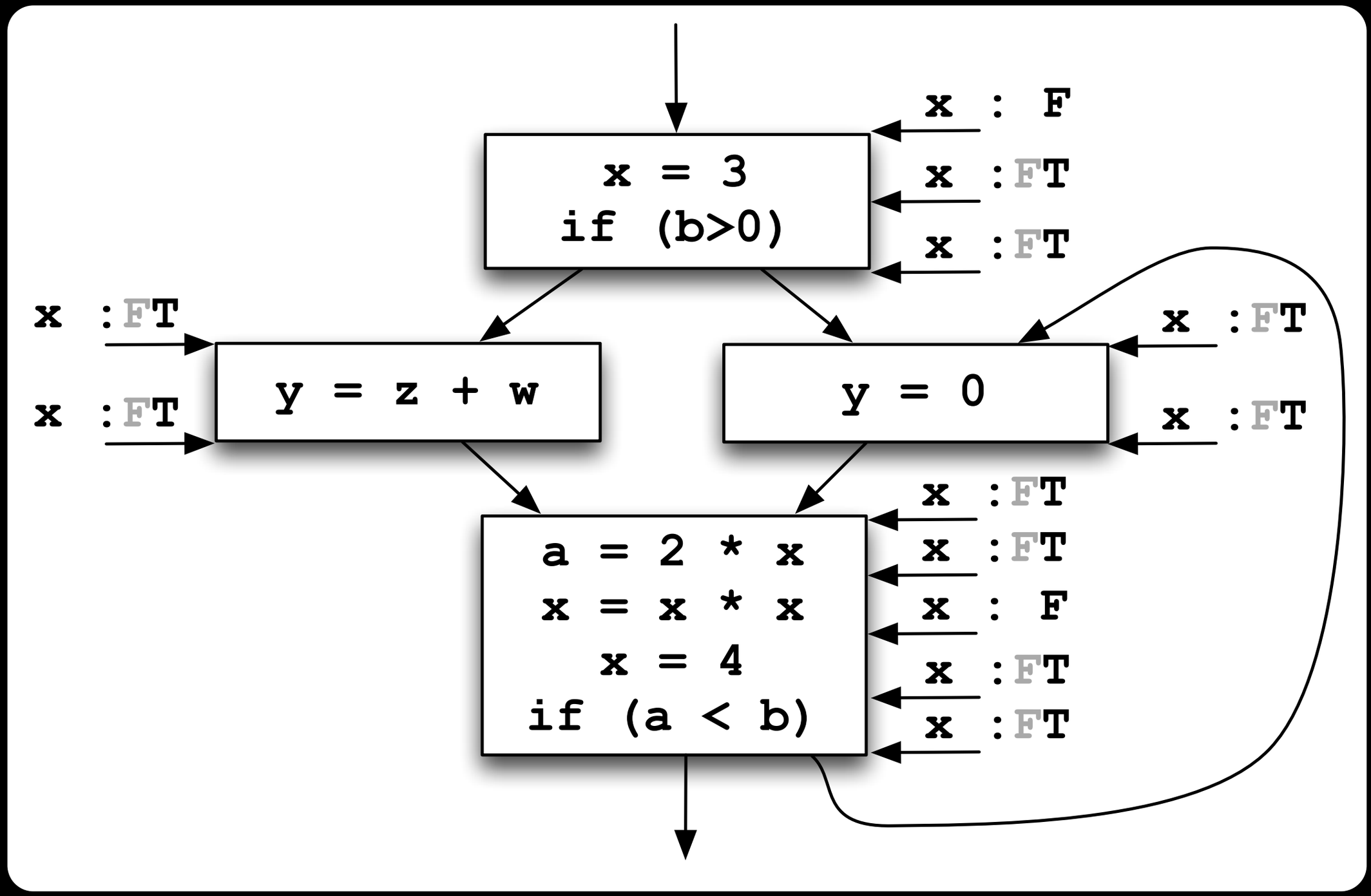
Algoritmo

- Para todos os pontos do programa, defina **`L_(…)=false`**
- Repita o processo abaixo até que todos os pontos satisfaçam as regras 1-4
    - dado um s que não satisfaça 1-4, atualize usando a regra apropriada

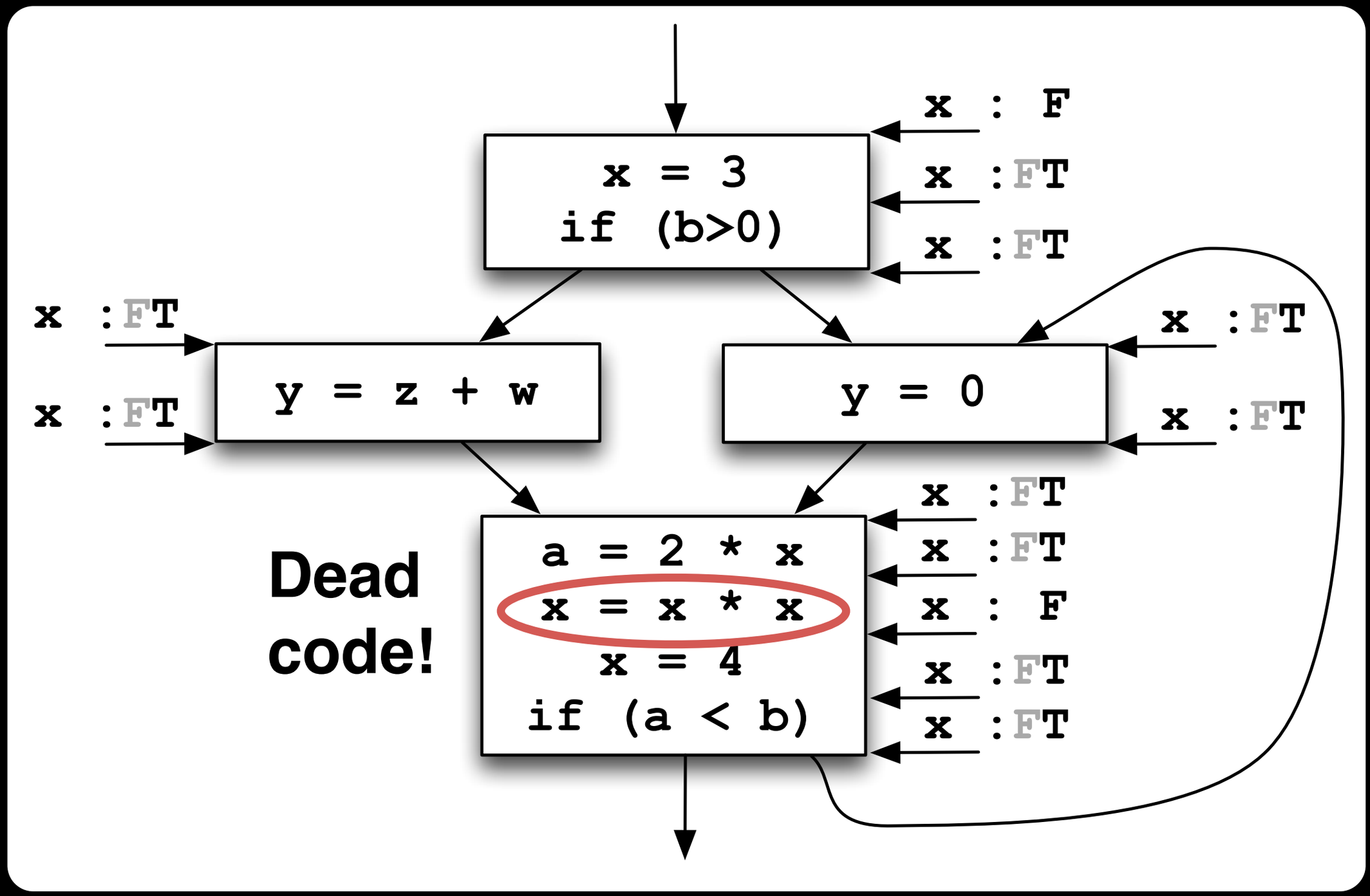
Liveness



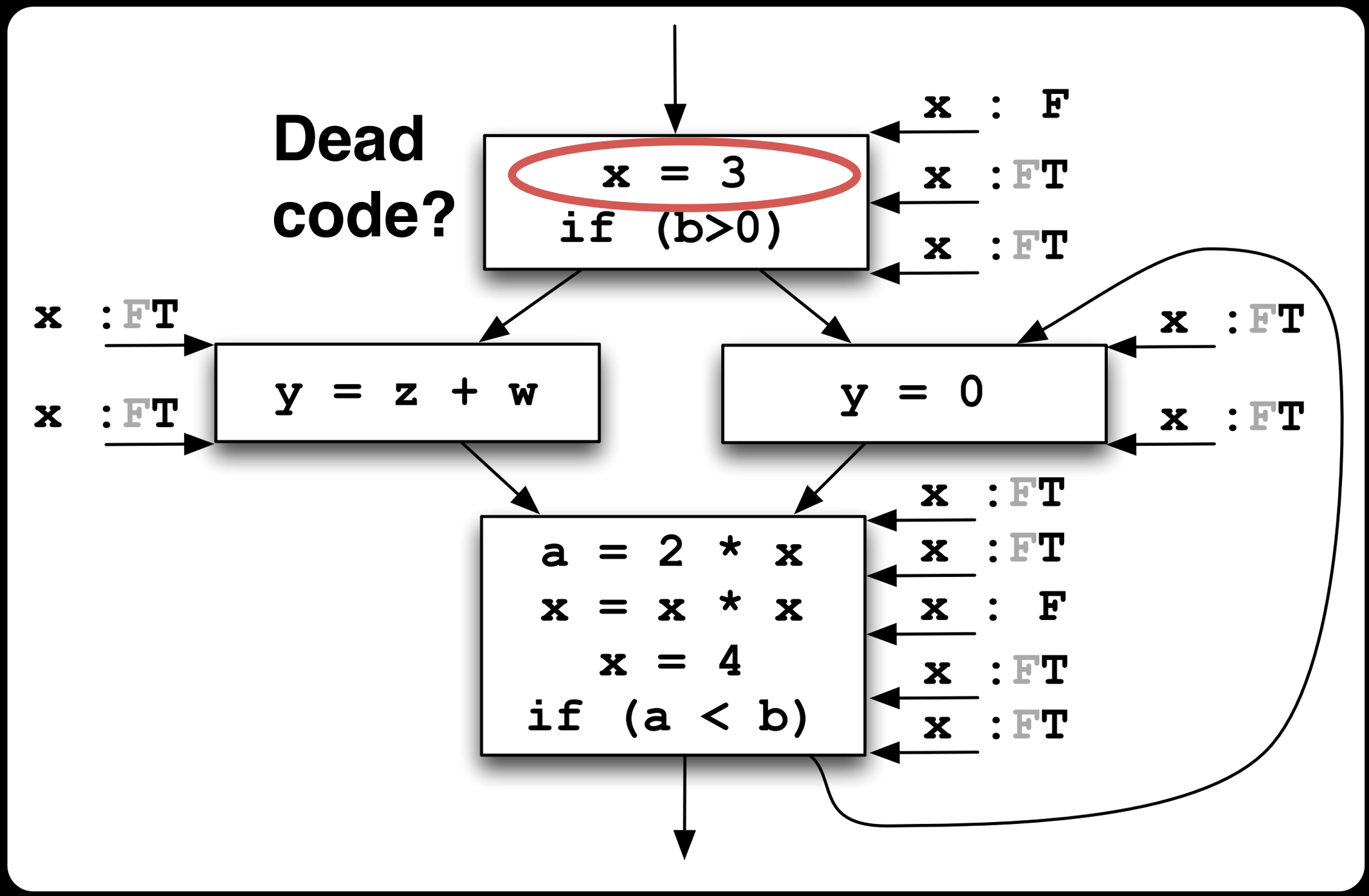
Liveness


x = 3
if (b>0)
y = z + w
y = 0
a = 2 * x
x = x * x
x = 4
if (a < b)
x : F
x :FT
x :FT
x :FT
x :FT
x :FT
x :FT
x :FT
x :FT
x : F
x :FT
x :FT

Liveness


x = 3
if (b>0)
x : F
x :FT
x :FT
x :FT
y = z + w
x :FT
x :FT
y = 0
x :FT
x :FT
a = 2 * x
x = x * x
x = 4
if (a < b)
x :FT
x :FT
x : F
x :FT
x :FT
Dead code!

Liveness


Dead code?
x = 3
if (b>0)
x : F
x :FT
x :FT
x :FT
y = z + w
x :FT
y = 0
x :FT
x :FT
a = 2 * x
x = x * x
x = 4
if (a < b)
x :FT
x :FT
x : F
x :FT
x :FT

Termination

- Um valor pode mudar de **false** para **true**, mas não o contrário
- Cada valor pode mudar apenas uma vez, então garantimos a terminação
- Uma vez que a análise tenha sido computada, é simples remover código morto
  - pode gerar nova análise

Representando análises com equações

Calculando *Liveness* de várias variáveis

- Método visto é útil para calcular *liveness* de uma variável específica, com relação a uma atribuição
- Cálculo a seguir especifica maneira alternativa de calcularmos *liveness* entre blocos básicos, para todas as variáveis do CFG

Calculando *Liveness*

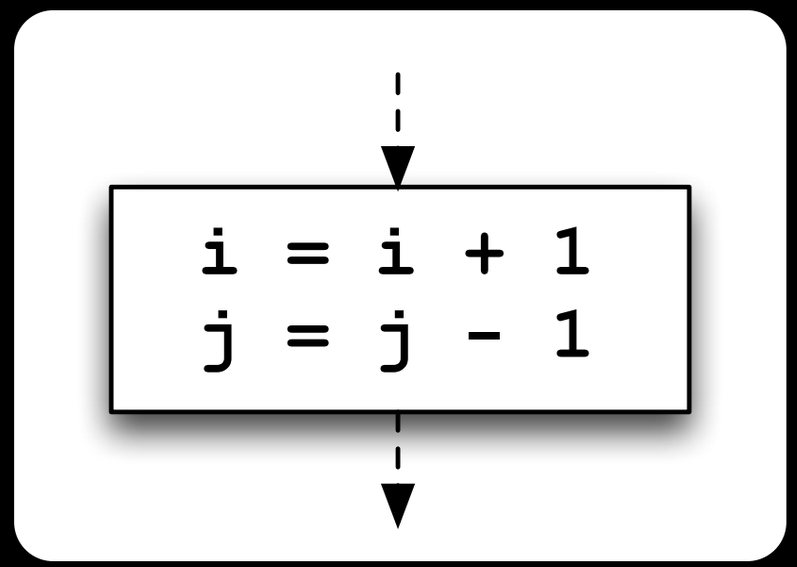
- Definimos as equações em termos das entradas e saídas de um bloco, isto é IN[B] e OUT[B]
  - conjunto de variáveis vivas nos pontos imediatamente antes e depois do bloco
- def_B é o conjunto de variáveis definidas em B antes de qualquer uso da variável em B
- use_B é o conjunto de variáveis cujos valores podem ser usados em B antes de qualquer definição

Exemplo de def/use

- Assumindo que `i` e `j` não são *aliases*

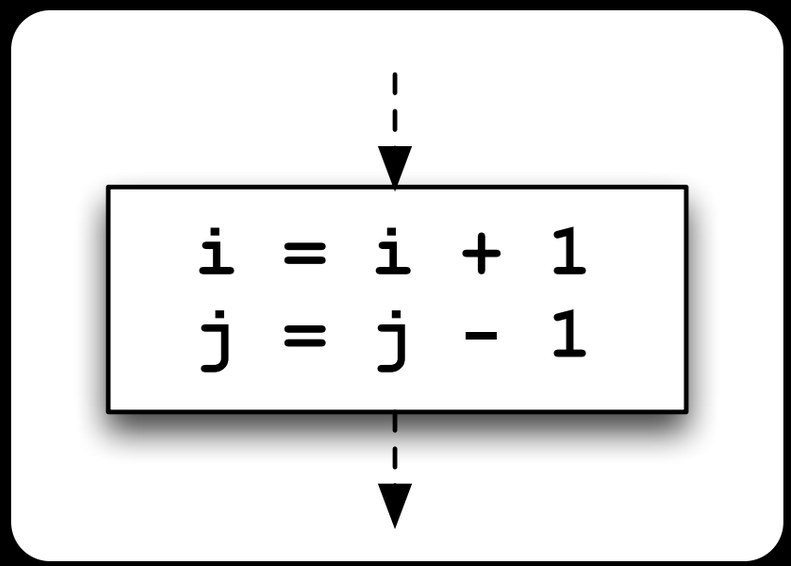
- $def_B = ?$

- $use_B = ?$



Exemplo de def/use

- Assumindo que `i` e `j` não são *aliases*



- $def_B = \{\}$
- $use_B = \{\texttt{i}, \texttt{j}\}$

Usando as definições

- Qualquer variável em use_B deve ser considerada *live* na entrada do bloco B
- Qualquer variável em def_B deve ser considerada *dead* na entrada do bloco B
  - estar em def_B mata qualquer oportunidade da variável ser considerada *live* em caminhos iniciados a partir do bloco B

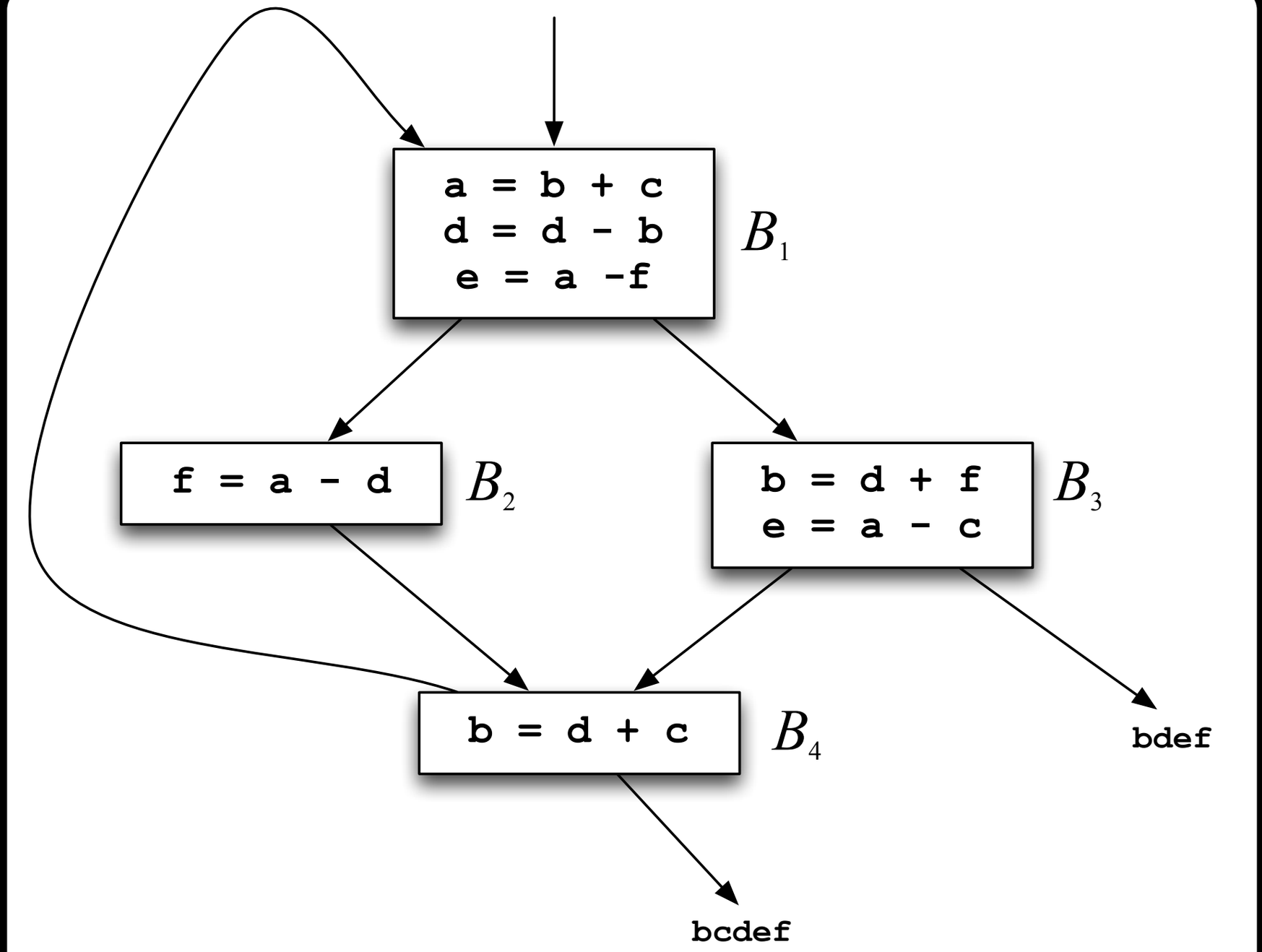
Equações

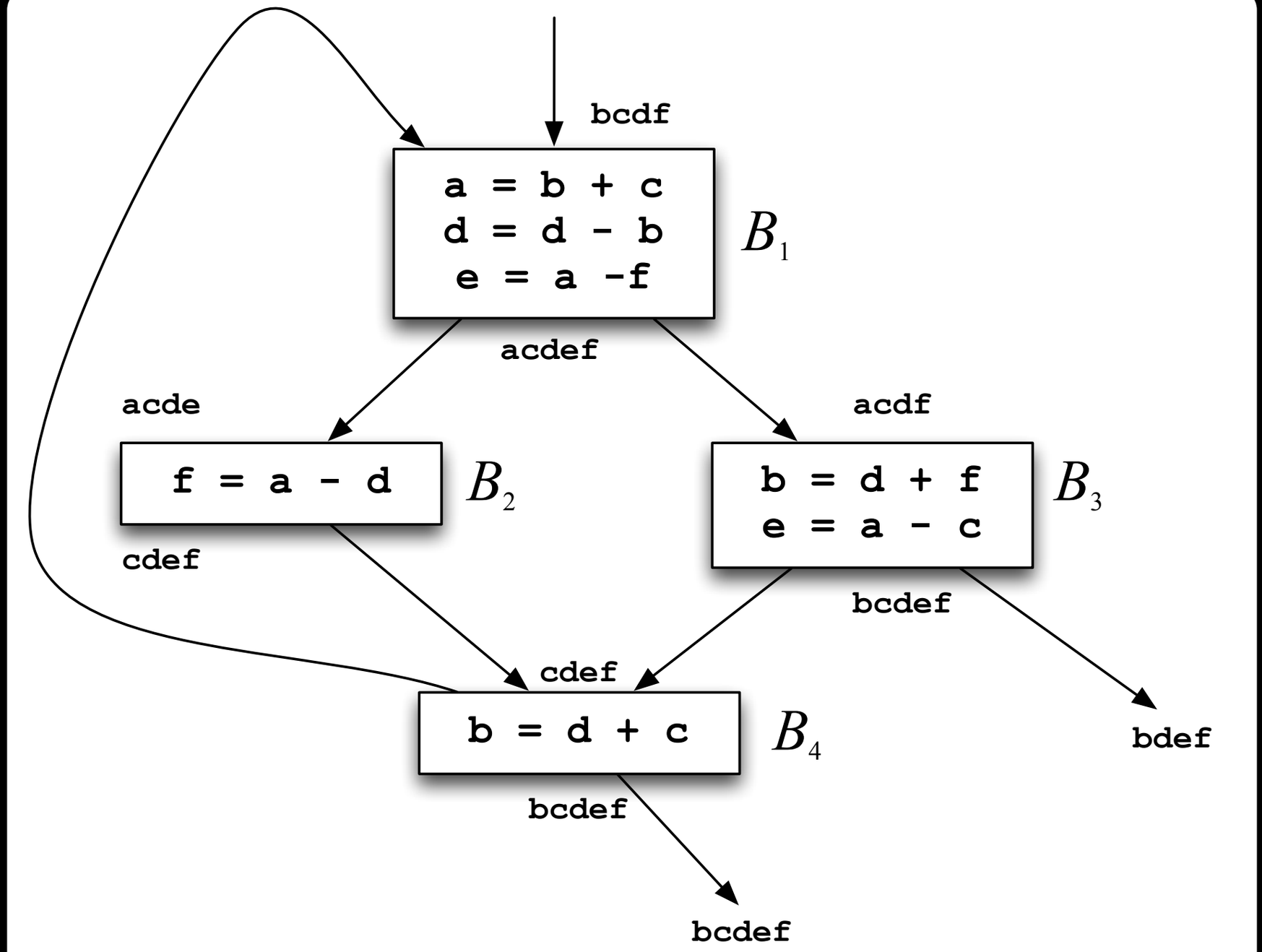
- $\text{Live}_{in}[B] = use_B \cup (\text{Live}_{out}[B] - def_B)$

- $\text{Live}_{out}[B] = \cup_{S \text{ é sucessor de } B} \text{Live}_{in}[S]$

Equações

- $\text{Live}_{in}[B] = use_B \cup (\text{Live}_{out}[B] - def_B)$
  - variável é *live* na entrada do bloco se é usada antes de ser redefinida, ou se está viva na saída e não é redefinida
- $\text{Live}_{out}[B] = \cup_{S \text{ é sucessor de } B} \text{Live}_{in}[S]$
  - variável é *live* na saída do bloco se e somente se é *live* na entrada de seus sucessores
- Opcionalmente: $\text{Live}_{in}[\text{EXIT}] = \varnothing$
  - condição de fronteira, nenhuma variável é *live* ao encerrarmos o programa

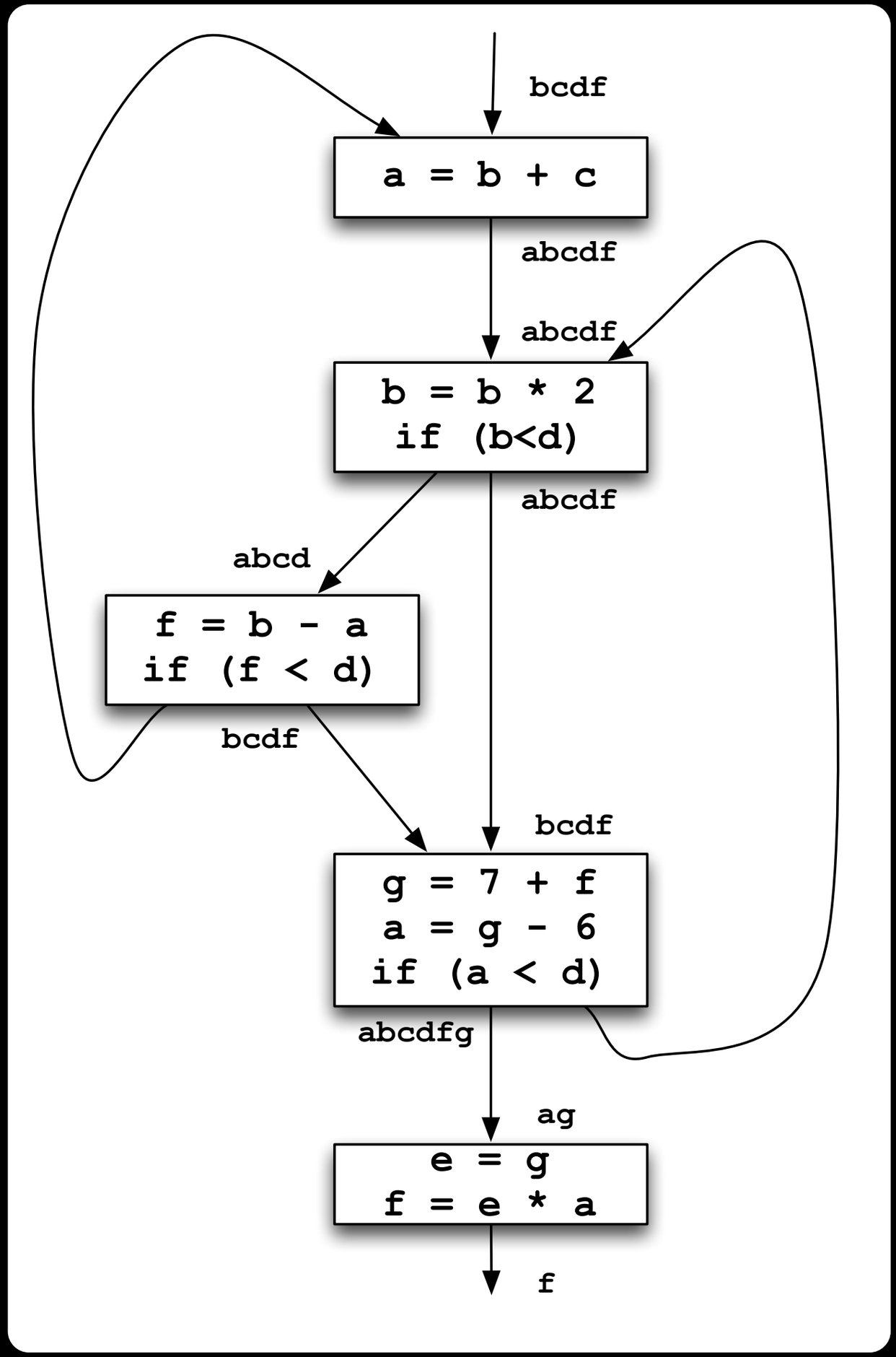

a = b + c
d = d - b
e = a -f
B_1
f = a - d
B_2
b = d + f
e = a - c
B_3
b = d + c
B_4
bdef
bcdef


bcdf
a = b + c
d = d - b
e = a -f
B_1
acdef
acde
acdf
f = a - d
B_2
b = d + f
e = a - c
B_3
cdef
bcdef
cdef
b = d + c
B_4
bdef
bcdef
bcdef

Considere o seguinte fragmento de código de três endereços:

- Construa o CFG usando as regras para criar blocos básicos;
- Informe o conjunto de variáveis vivas na entrada e saída de cada bloco básico do CFG, assumindo que apenas f é viva na saída do CFG.

```
1.  L1:   a = b + c
2.  L2:   b = b * 2
3.        if (b<d) goto L3
4.        f = b - a
5.        if (f<d) goto L1
6.  L3:   g = 7 + f
7.        a = g - 6
8.        if (a>d) goto L2
9.        e = g
10.       f = e*a
```


bcdf
a = b + c
abcdf
abcdf
b = b * 2
if (b<d)
abcdf
abcd
f = b - a
if (f < d)
bcdf
bcdf
g = 7 + f
a = g - 6
if (a < d)
abcdfg
ag
e = g
f = e * a
f

Considere o seguinte fragmento de código de três endereços:

- Construa o CFG usando as regras para criar blocos básicos;
- Informe o conjunto de variáveis vivas na entrada e saída de cada bloco básico do CFG, assumindo que apenas **a** é viva na saída do CFG.

```
     a = 1
     c = 3
     if a >= 2 goto L2
     b = c+1
     c = b+2
L1:  b = d
     if a >= d goto L2
     return a
L2:  a = c+2
     d = c+1
     goto L1
```