

# Data Flow Analysis 1

## Liveness

Dr. William Harrison

[harrisonwl@missouri.edu](mailto:harrisonwl@missouri.edu)

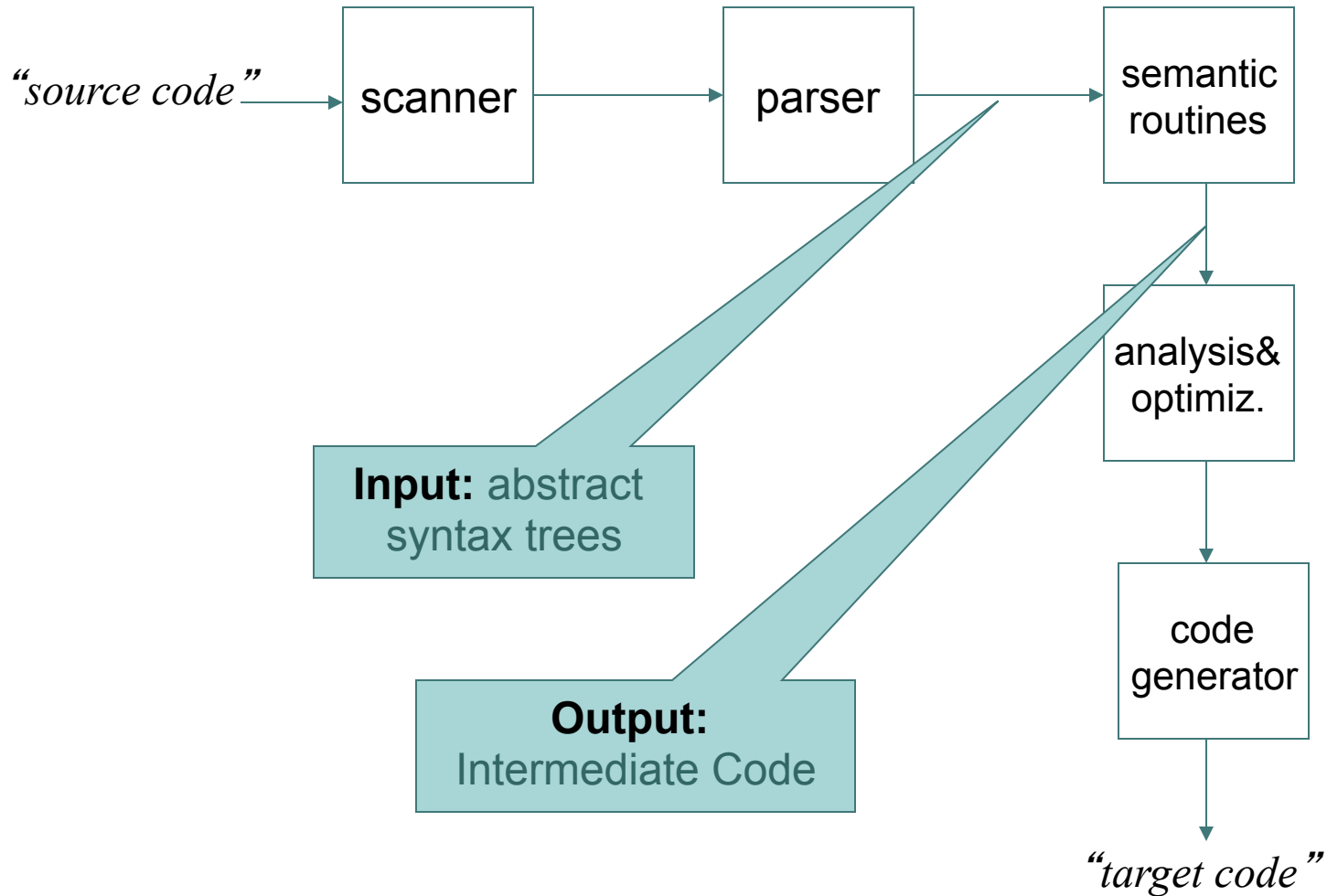
CS 4430 Compilers I



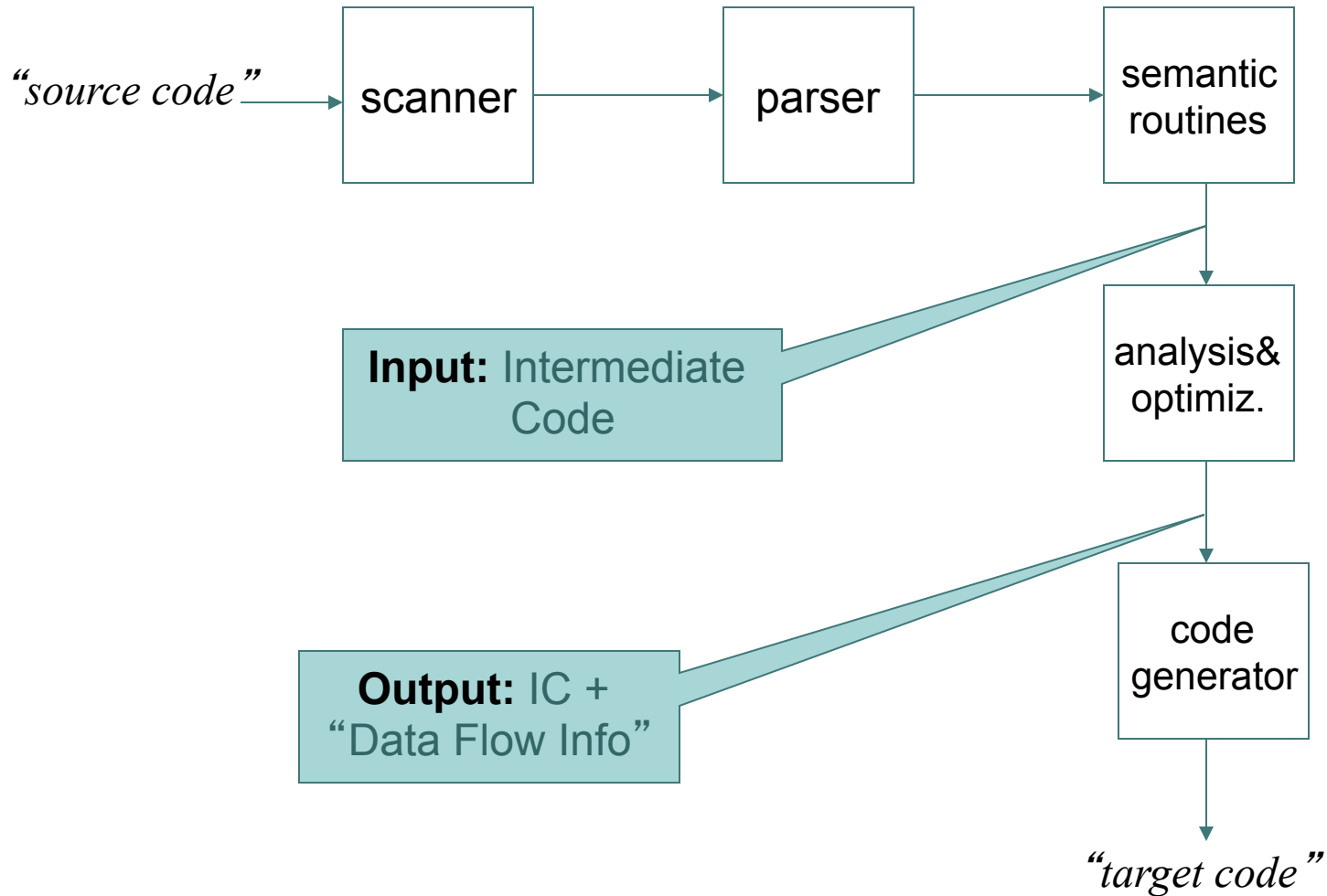
# This Week

- Start Optimization Ideas Today
  - particularly basic data flow analysis
    - liveness
  - and, later, its uses: register allocation

# Review: Where are we?



# Review: Where are we going?



# Review: Translation into IR via “translation schemas”

Source Code

```
begin
  read(A,B) ;
  if A > B then
    C := A + 5;
  else
    C := B + 5;
  end if;
  write(2 * (C - 1));
end
```



IR

```
(READI, A)
(READI, B)
(GT, A, B, t1)
(JUMP0, t1, L1)
(ADDI, A, 5, C)
(JUMP, L2)
(LABEL, L1)
(ADDI, B, 5, C)
(LABEL, L2)
(SUBI, C, 1, t2)
(MULTI, 2, t2, t3)
(WRITEI, t3)
```

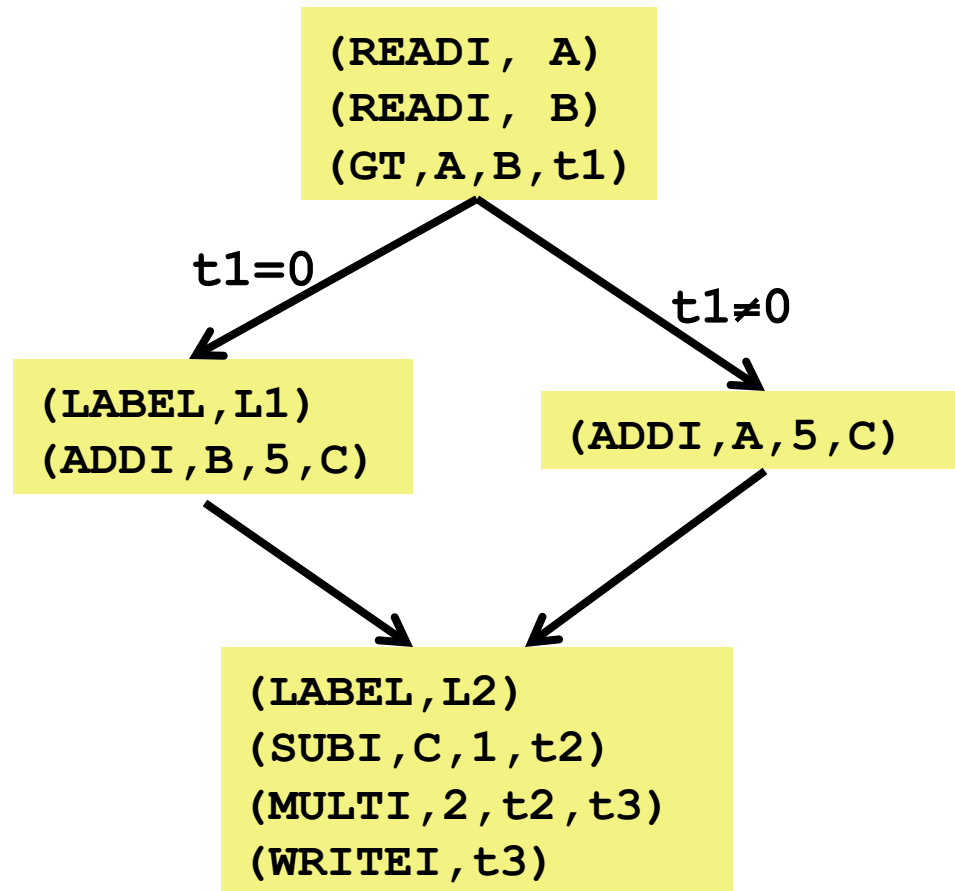
# Graphical Representation of IR

IR

```
(READI, A)
(READI, B)
(GT, A, B, t1)
(JUMP0, t1, L1)
(ADDI, A, 5, C)
(JUMP, L2)
(LABEL, L1)
(ADDI, B, 5, C)
(LABEL, L2)
(SUBI, C, 1, t2)
(MULTI, 2, t2, t3)
(WRITEI, t3)
```



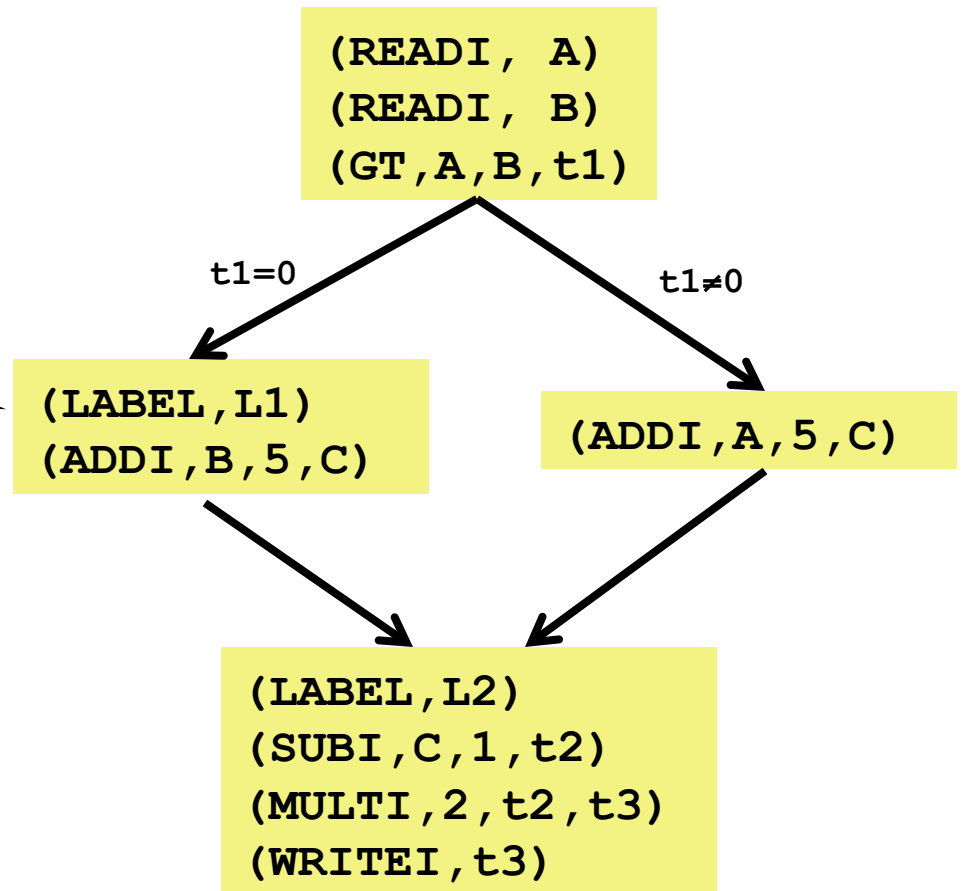
Control Flow Graph (CFG)



# Graphical Representation

Control Flow Graph (CFG)

**Basic Block:** *linear sequences of tuples containing no branches until the end. Such branches are usually represented as arrows.*





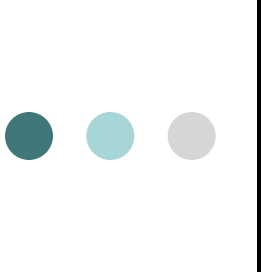
# Virtual Registers

Variables in the IR are sometimes referred to as “virtual registers”

```
a ← 0  
L: b ← a + 1  
  c ← c + b  
  a ← b * 2  
  if (a < N) goto L  
  return c
```

\* registers on a microprocessor are called “physical registers”





# Register allocation answers the question

```
a ← 0  
L: b ← a + 1  
  c ← c + b  
  a ← b * 2  
  if (a < N) goto L  
  return c
```

Should we store **c** in a physical register or on the run-time stack?



# Liveness Analysis

- ...determines when the value within a virtual register may still be used
  - a.k.a. its value is “live”
- ...and when it won't
  - a.k.a. its value is “dead”
- This property, “liveness”, may be **approximated** statically

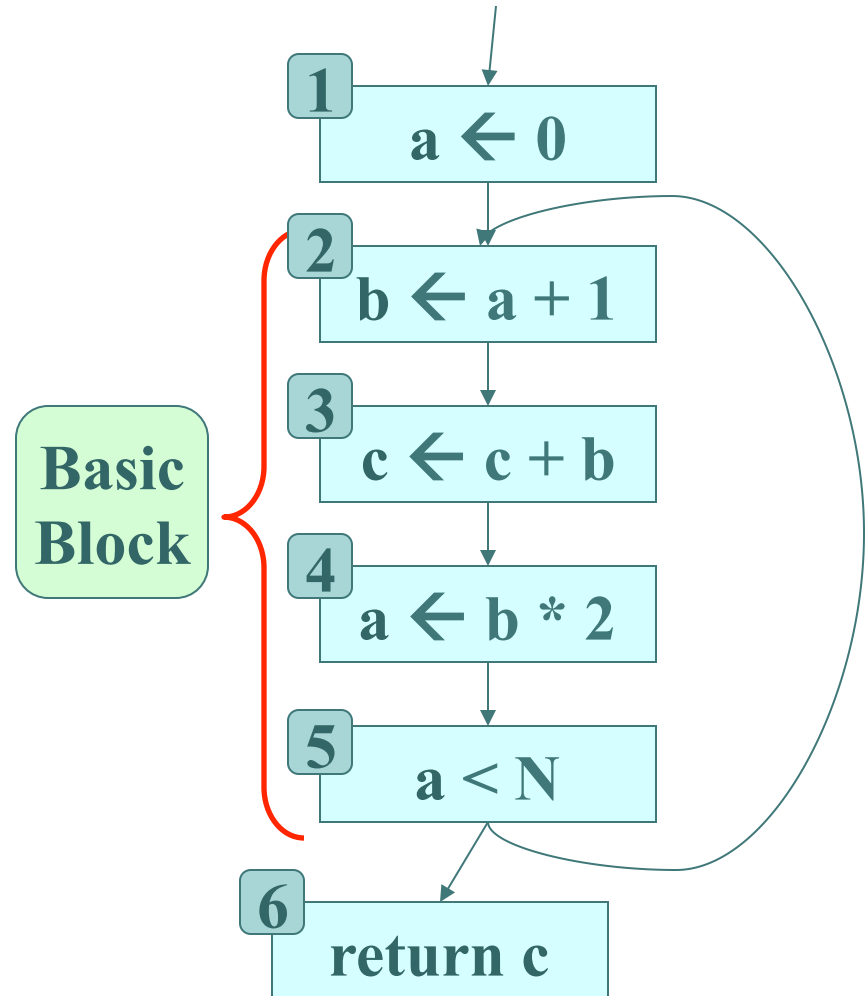


# Register Allocation

- ...depends on good liveness analysis
- Ex: if virtual registers A,B,C, and D are live at some point in the program
  - ...and you have only 3 physical registers
  - Then, at least one of these virtual registers must be maintained in the run-time stack
- Why is this important?
- When is it important to know this?

# Small Control Flow Graph

$a \leftarrow 0$   
L:  $b \leftarrow a + 1$   
 $c \leftarrow c + b$   
 $a \leftarrow b * 2$   
if ( $a < N$ ) goto L  
return c



# Liveness of **a**

**a** is assigned to in block 1 and 4.

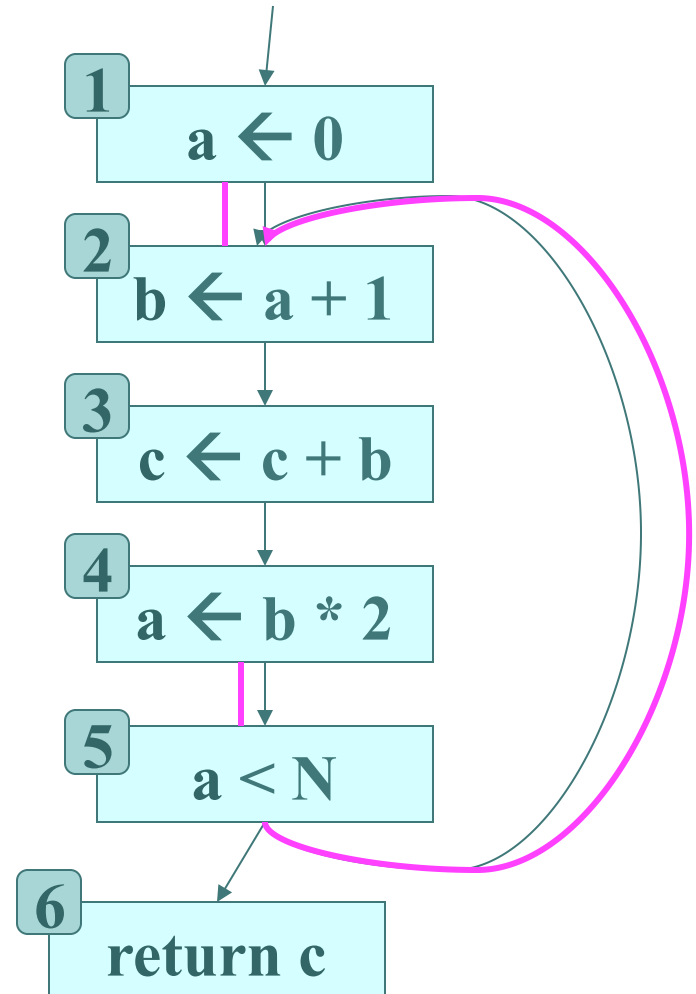
**a** is dead leaving block 2.

**a** is dead in block 3.

**a** is dead entering block 4.

**a** is dead entering block 6.

**a** is dead in block 6.



# Liveness of **b**

**b** is assigned to in block 2.

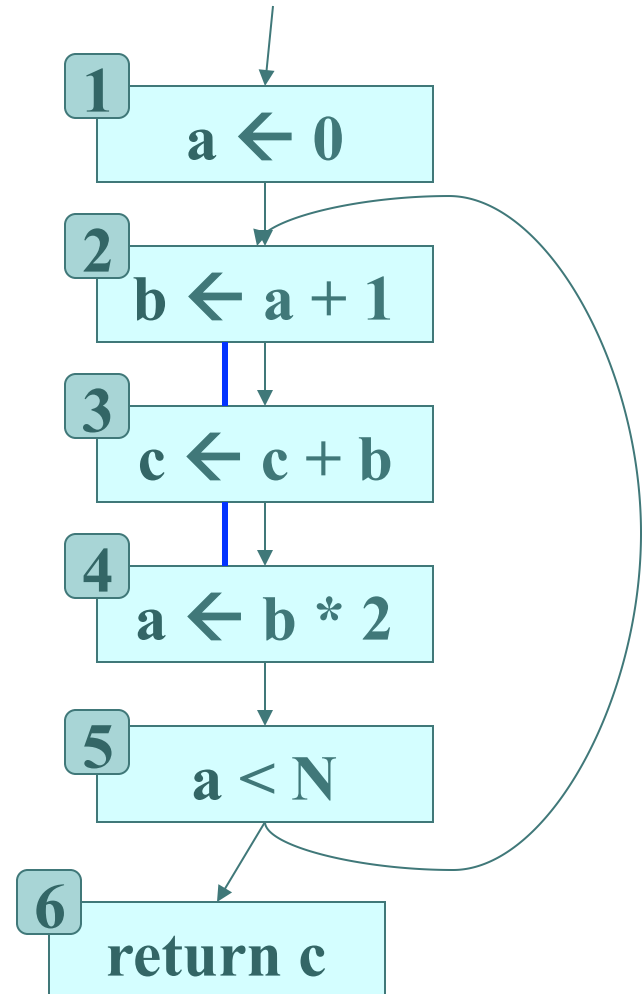
**b** is live entering block 3.

**b** is live in block 3.

**b** is dead leaving block 4.

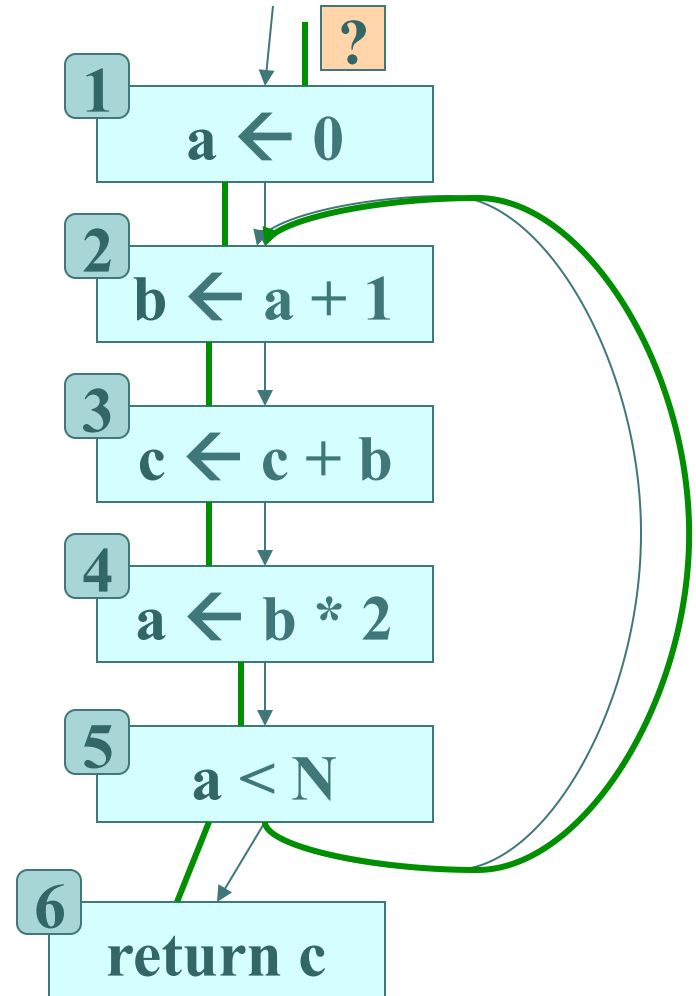
...etc...

Notice how the liveness of **a** and **b** do not overlap.

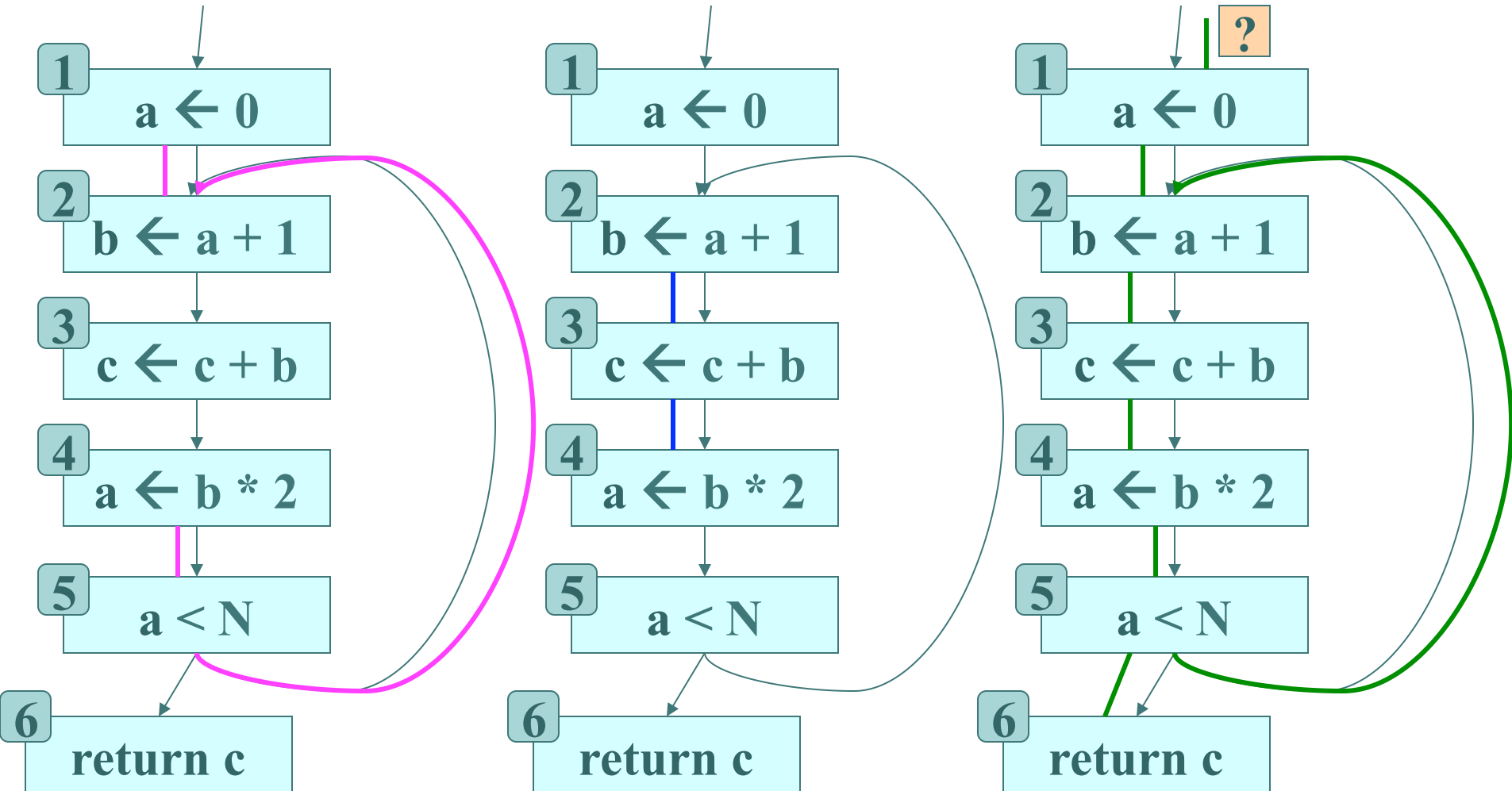


# Liveness of **c**

c is live everywhere!



# Liveness of **a**, **b** and **c**





# Some Flow Graph Definitions

Flow graph terminology

$\text{pred}[N]$  = set of immediate predecessors.

$\text{pred}[2] = \{ 1, 5 \}$

$\text{succ}[N]$  = set of successors.

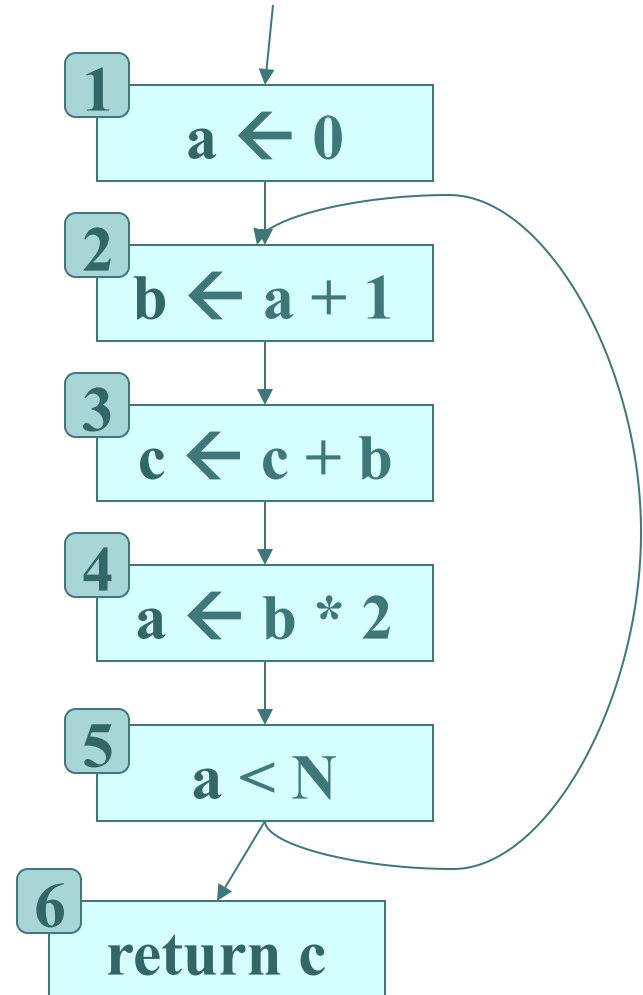
$\text{succ}[3] = \{ 4 \}$

$\text{def}[N]$  = set of registers assigned-to in  $N$ .

$\text{def}[3] = \{ c \}$

$\text{use}[N]$  = set of registers used in  $N$ .

$\text{use}[2] = \{ a \}$



# Ex: intra-block analysis

- Perform liveness analysis on this program
  - defined use and def for each instruction.
  - find liveness for each variable.

Variables are

b, c, d, e, f, g, h, j, k, m

Remember:

where is a variable **used**?  
work backwards to its  
**definition**.

live in: k j

```
(1)    g ← M[j + 12]
(2)    h ← k - 1
(3)    f ← g * h
(4)    e ← M[j + 8]
(5)    m ← M[j + 16]
(6)    b ← M[f]
(7)    c ← e + 8
(8)    d ← c
(9)    k ← m + 4
(10)   j ← b
```

live out: d k j



(1)  $g \leftarrow M[j + 12]$

(2)  $h \leftarrow k - 1$

(3)  $f \leftarrow g * h$

(4)  $e \leftarrow M[j + 8]$

(5)  $m \leftarrow M[j + 16]$

(6)  $b \leftarrow M[f]$

(7)  $c \leftarrow e + 8$

(8)  $d \leftarrow c$

(9)  $k \leftarrow m + 4$

(10)  $j \leftarrow b$

$\{d, k, b\}$

liveout={d, k, j}



(1)  $g \leftarrow M[j + 12]$

(2)  $h \leftarrow k - 1$

(3)  $f \leftarrow g * h$

(4)  $e \leftarrow M[j + 8]$

(5)  $m \leftarrow M[j + 16]$

(6)  $b \leftarrow M[f]$

(7)  $c \leftarrow e + 8$

(8)  $d \leftarrow c$

(9)  $k \leftarrow m + 4$

(10)  $j \leftarrow b$

$\{d, b, m\}$

$\{d, k, b\}$

liveout={d, k, j}



(1)  $g \leftarrow M[j + 12]$

(2)  $h \leftarrow k - 1$

(3)  $f \leftarrow g * h$

(4)  $e \leftarrow M[j + 8]$

(5)  $m \leftarrow M[j + 16]$

(6)  $b \leftarrow M[f]$

(7)  $c \leftarrow e + 8$

(8)  $d \leftarrow c$

(9)  $k \leftarrow m + 4$

(10)  $j \leftarrow b$

$\{b, m, c\}$

$\{d, b, m\}$

$\{d, k, b\}$

liveout={d, k, j}



(1)  $g \leftarrow M[j + 12]$

(2)  $h \leftarrow k - 1$

(3)  $f \leftarrow g * h$

(4)  $e \leftarrow M[j + 8]$

(5)  $m \leftarrow M[j + 16]$

(6)  $b \leftarrow M[f]$

(7)  $c \leftarrow e + 8$

(8)  $d \leftarrow c$

(9)  $k \leftarrow m + 4$

(10)  $j \leftarrow b$

$\{b, m, e\}$

$\{b, m, c\}$

$\{d, b, m\}$

$\{d, k, b\}$

liveout={d, k, j}



(1)  $g \leftarrow M[j + 12]$

(2)  $h \leftarrow k - 1$

(3)  $f \leftarrow g * h$

(4)  $e \leftarrow M[j + 8]$

(5)  $m \leftarrow M[j + 16]$

(6)  $b \leftarrow M[f]$

(7)  $c \leftarrow e + 8$

(8)  $d \leftarrow c$

(9)  $k \leftarrow m + 4$

(10)  $j \leftarrow b$

$\{m, e, f\}$

$\{b, m, e\}$

$\{b, m, c\}$

$\{d, b, m\}$

$\{d, k, b\}$

liveout={d, k, j}



(1)  $g \leftarrow M[j + 12]$

(2)  $h \leftarrow k - 1$

(3)  $f \leftarrow g * h$

(4)  $e \leftarrow M[j + 8]$

(5)  $m \leftarrow M[j + 16]$

(6)  $b \leftarrow M[f]$

(7)  $c \leftarrow e + 8$

(8)  $d \leftarrow c$

(9)  $k \leftarrow m + 4$

(10)  $j \leftarrow b$

$\{e, f, j\}$

$\{m, e, f\}$

$\{b, m, e\}$

$\{b, m, c\}$

$\{d, b, m\}$

$\{d, k, b\}$

liveout= $\{d, k, j\}$





(1)  $g \leftarrow M[j + 12]$

(2)  $h \leftarrow k - 1$

(3)  $f \leftarrow g * h$

(4)  $e \leftarrow M[j + 8]$  {f, j}

(5)  $m \leftarrow M[j + 16]$  {e, f, j}

(6)  $b \leftarrow M[f]$  {m, e, f}

(7)  $c \leftarrow e + 8$  {b, m, e}

(8)  $d \leftarrow c$  {b, m, c}

(9)  $k \leftarrow m + 4$  {d, b, m}

(10)  $j \leftarrow b$  {d, k, b}

liveout={d, k, j}



(1)  $g \leftarrow M[j + 12]$

(2)  $h \leftarrow k - 1$

(3)  $f \leftarrow g * h$

(4)  $e \leftarrow M[j + 8]$

(5)  $m \leftarrow M[j + 16]$

(6)  $b \leftarrow M[f]$

(7)  $c \leftarrow e + 8$

(8)  $d \leftarrow c$

(9)  $k \leftarrow m + 4$

(10)  $j \leftarrow b$

$\{j, g, h\}$

$\{f, j\}$

$\{e, f, j\}$

$\{m, e, f\}$

$\{b, m, e\}$

$\{b, m, c\}$

$\{d, b, m\}$

$\{d, k, b\}$

liveout= $\{d, k, j\}$

livein = {j, k}

(1)  $g \leftarrow M[j + 12]$

(2)  $h \leftarrow k - 1$

(3)  $f \leftarrow g * h$

(4)  $e \leftarrow M[j + 8]$

(5)  $m \leftarrow M[j + 16]$

(6)  $b \leftarrow M[f]$

(7)  $c \leftarrow e + 8$

(8)  $d \leftarrow c$

(9)  $k \leftarrow m + 4$

(10)  $j \leftarrow b$

{j, g, k}

{j, g, h}

{f, j}

{e, f, j}

{m, e, f}

{b, m, e}

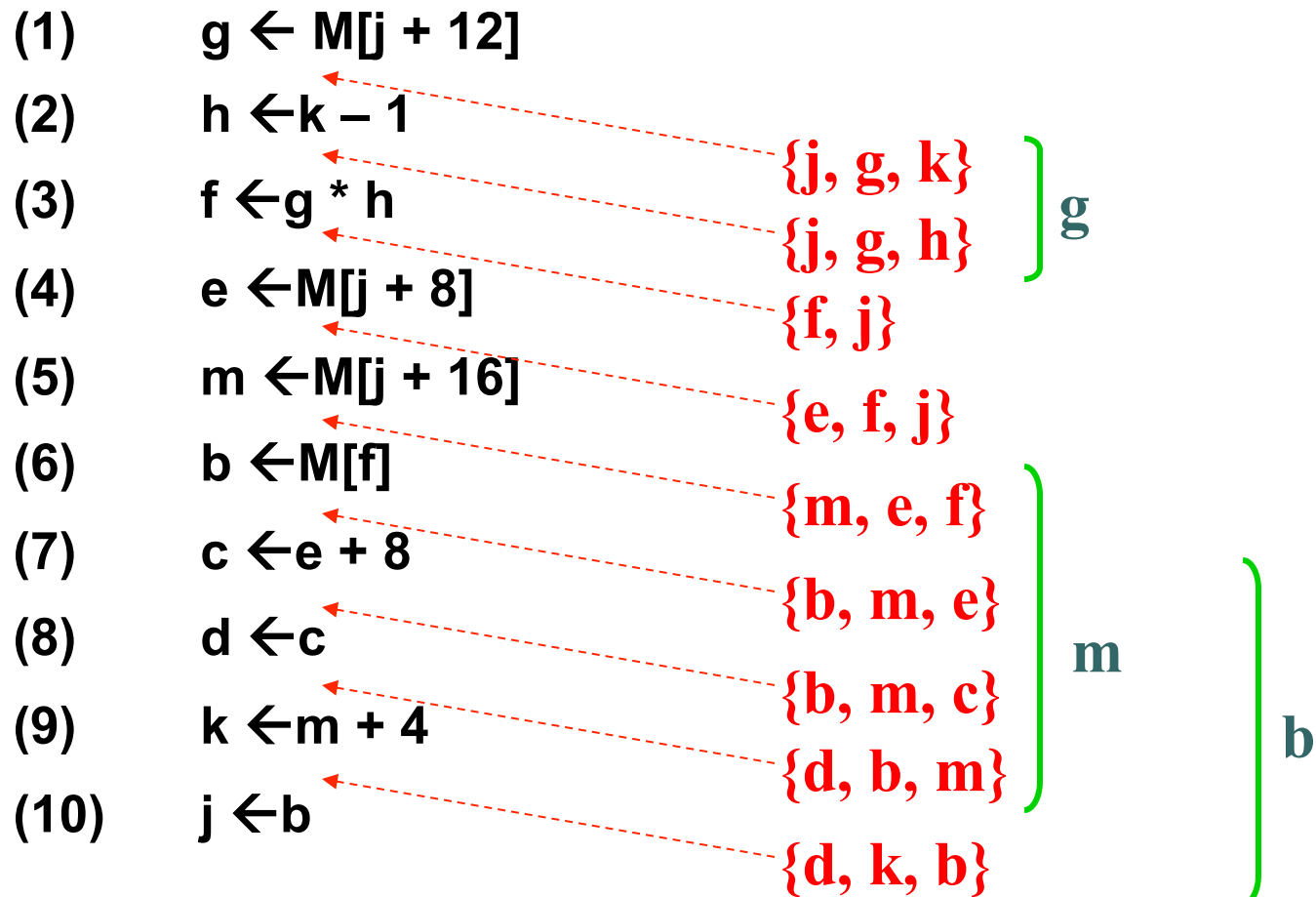
{b, m, c}

{d, b, m}

{d, k, b}

liveout = {d, k, j}

liveout = {j, k}

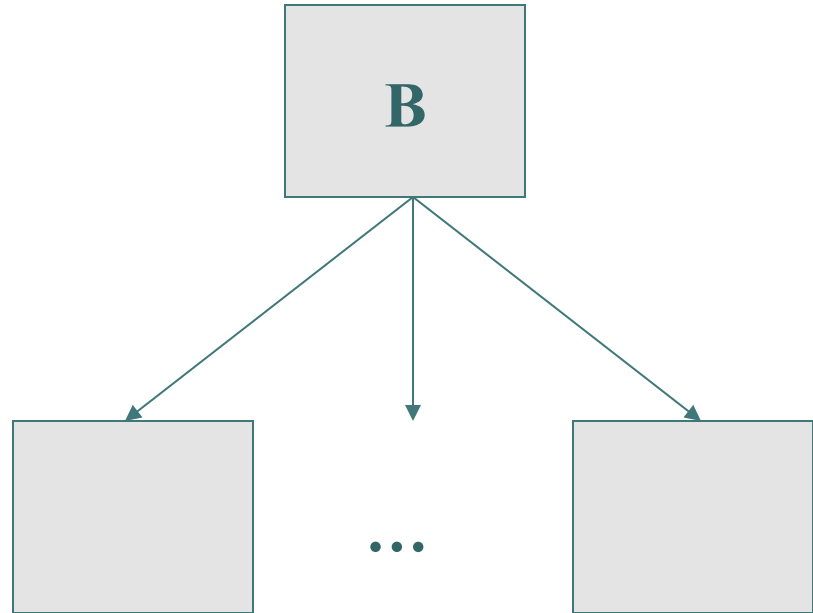


liveout={d, k, j}

# Inter-block data flow analysis

Q: Where does the initial “liveout” come from?

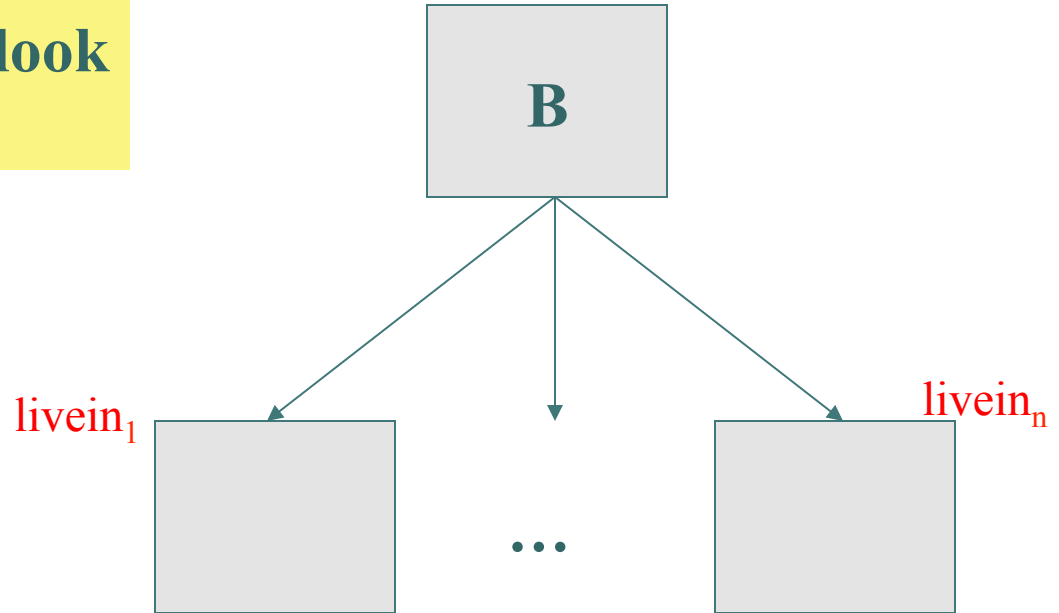
**In control-flow graph, look at all successors of B**



# Inter-block data flow analysis

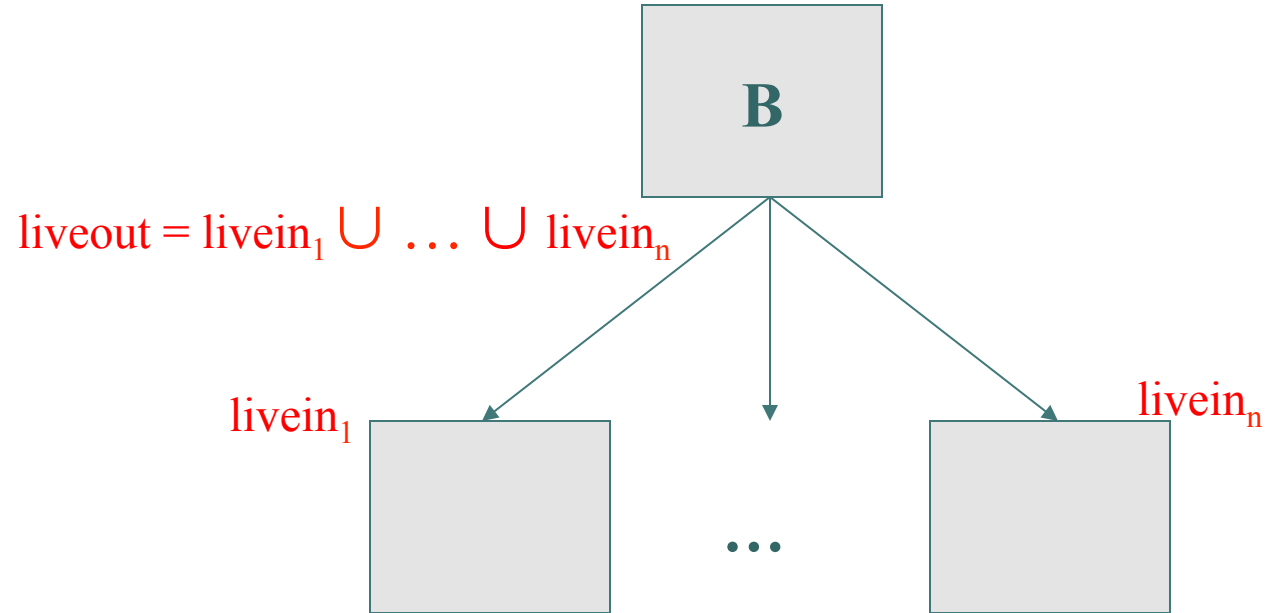
Q: Where does the initial “liveout” come from?

**In control-flow graph, look at all successors of B**



# Inter-block data flow analysis

Q: Where does the initial “liveout” come from?





# Liveness summary

- Start with the places that **use** registers
- Propagate liveness backwards to the **definitions**.
- This is typically done via an iterative process.
- Can be expensive →  $O(n^4)$
- It is always correct to approximate
  - for example - everything is live
    - Leads to poor register allocation
  - Some algorithms compromise.
    - Cheaper to compute
    - Useable liveness information.





# Next Time

- General form of liveness analysis
  - called sometimes “iterative data flow analysis”