

Adventures in
Dataflow Analysis + SSA

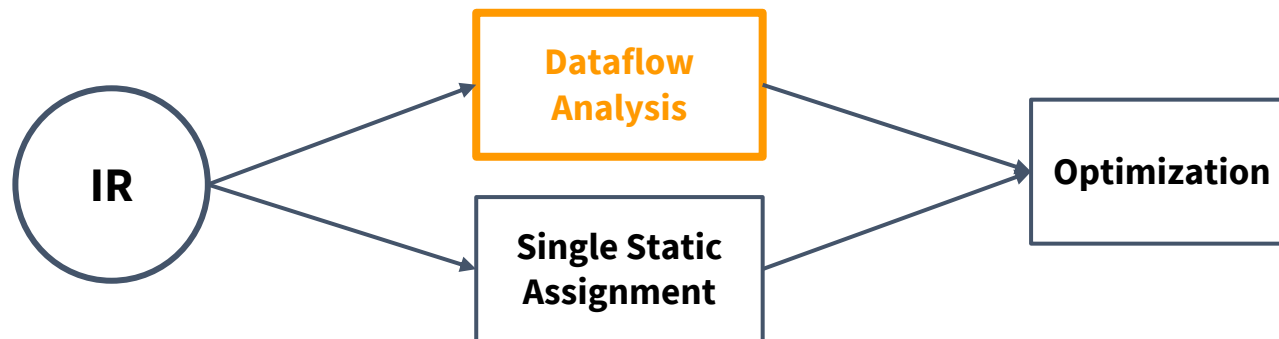
CSE 401 Section 9
Michael, Eunia, Kris, Anand, Gavin

Announcements

- Codegen due tonight, 12/03/20
- Report due next Tuesday, 12/08/20
- HW 4 due next Thursday, 12/10/20
- 2 free Late days can be used CodeGen and hw4, but not the CSE 401 report
- CSE M 501 final project and report due end of next week
 - No late days on these
- Email course staff if you need any additional resources or support!
- If you run into Office Hours trouble, please post on Ed!!

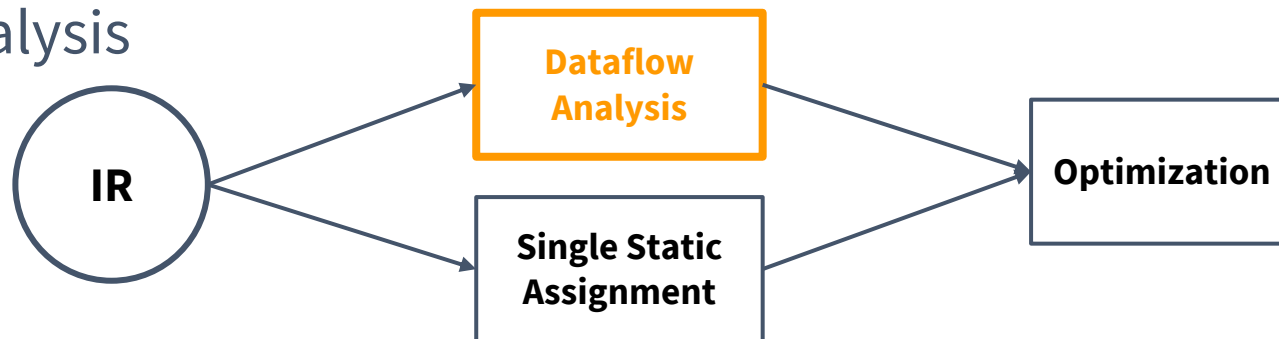
Overview of Dataflow Analysis

- A framework for exposing properties about programs
- Operates using sets of “facts”



Overview of Dataflow Analysis

- A framework for exposing properties about programs
- Operates using sets of “facts”
- Just the initial discovery phase
 - Changes can then be made to optimize based on the analysis



Overview of Dataflow Analysis

- Basic Framework of Set Definitions (for a Basic Block b):
 - $IN(b)$: facts true on entry to b
 - $OUT(b)$: facts true on exit from b
 - $GEN(b)$: facts created (and not killed) in b
 - $KILL(b)$: facts killed in b

Reaching Definitions (*A Dataflow Problem*)

“What definitions of each variable might reach this point”

- Could be used for:
 - Constant Propagation
 - Uninitialized Variables

```
int x;  
  
if (y > 0) {  
    x = y;  
} else {  
    x = 0;  
}  
  
System.out.println(x);
```

“x=y”, “x=0”

Reaching Definitions (A Dataflow Problem)

“What definitions of each variable might reach this point”

- **Be careful:** Does not involve the *value* of the definition
 - The dataflow problem “Available Expressions” is designed for that

```
int x;  
  
if (y > 0) {  
    x = y;  
} else {  
    x = 0;  
}  
  
y = -1;  
System.out.println(x);
```

still: “x=y”, “x=0”

Problems 1_a and 1_b

```

L0:  a = 0
L1:  b = a + 1
L2:  c = c + b
L3:  a = b * 2
L4:  if a < N goto L1
L5:  return c

```

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0					
L1	L1					
L2	L2					
L3	L3					
L4						
L5						

L0: a = 0
 L1: b = a + 1
 L2: c = c + b
 L3: a = b * 2
 L4: if a < N goto L1
 L5: return c

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0	L3				
L1	L1					
L2	L2					
L3	L3	L0				
L4						
L5						

L0: a = 0
 L1: b = a + 1
 L2: c = c + b
 L3: a = b * 2
 L4: if a < N goto L1
 L5: return c

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0	L3				
L1	L1		L0			
L2	L2		L0, L1			
L3	L3	L0	L0, L1, L2			
L4			L1, L2, L3			
L5			L1, L2, L3			

L0: a = 0
 L1: b = a + 1
 L2: c = c + b
 L3: a = b * 2
 L4: if a < N goto L1
 L5: return c

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0	L3		L0		
L1	L1		L0	L0, L1		
L2	L2		L0, L1	L0, L1, L2		
L3	L3	L0	L0, L1, L2	L1, L2, L3		
L4			L1, L2, L3	L1, L2, L3		
L5			L1, L2, L3	L1, L2, L3		

L0: a = 0
 L1: b = a + 1
 L2: c = c + b
 L3: a = b * 2
 L4: if a < N goto L1
 L5: return c

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0	L3		L0		L0
L1	L1		L0	L0, L1	L0, L1, L2, L3	L0, L1, L2, L3
L2	L2		L0, L1	L0, L1, L2	L0, L1, L2, L3	L0, L1, L2, L3
L3	L3	L0	L0, L1, L2	L1, L2, L3	L0, L1, L2, L3	L1, L2, L3
L4			L1, L2, L3	L1, L2, L3	L1, L2, L3	L1, L2, L3
L5			L1, L2, L3	L1, L2, L3	L1, L2, L3	L1, L2, L3

```

L0: a = 0
L1: b = a + 1
L2: c = c + b
L3: a = b * 2
L4: if a < N goto L1
L5: return c

```

Convergence!

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0	L3		L0		L0
L1	L1		L0	L0, L1	L0, L1, L2, L3	L0, L1, L2, L3
L2	L2		L0, L1	L0, L1, L2	L0, L1, L2, L3	L0, L1, L2, L3
L3	L3	L0	L0, L1, L2	L1, L2, L3	L0, L1, L2, L3	L1, L2, L3
L4			L1, L2, L3	L1, L2, L3	L1, L2, L3	L1, L2, L3
L5			L1, L2, L3	L1, L2, L3	L1, L2, L3	L1, L2, L3

Single Static Assignment

An intermediate representation where each variable has only one definition:

Original

```
a := x + y
b := a - 1
a := y + b
b := x * 4
a := a + b
```



SSA Form

```
a1 := x1 + y1
b1 := a1 - 1
a2 := y1 + b1
b2 := x1 * 4
a3 := a2 + b2
```

SSA: Why We Love It

- Without SSA, all definitions and uses of a variable get mixed together
 - Computing information about the definitions of a variable is an expensive but necessary part of many dataflow analyses

SSA: Why We Love It

- Without SSA, all definitions and uses of a variable get mixed together
 - Computing information about the definitions of a variable is an expensive but necessary part of many dataflow analyses
- Doing the work of converting to SSA once makes many analyses + optimizations more efficient
 - SSA can be thought of as an implicit representation of Definition/Use chains

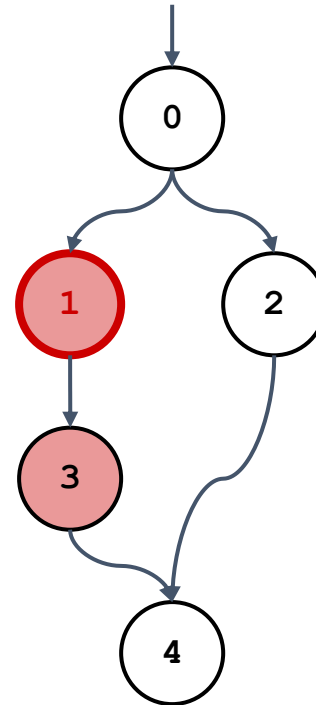
SSA: Why We Love It

- Ex: Dead Store Elimination
 - Without SSA: Compute live variables at every point, which requires working backwards and using the dataflow sets to check for *any path* that does not kill the variable, and eliminate any stores that are not to a live variable.
 - With SSA: Eliminate any store where the variable being assigned has 0 uses.
- Constant Propagation is another optimization made much easier by SSA

Dominators

- A node x *dominates* a node y iff every path from the entry point of the control flow graph to y includes x .

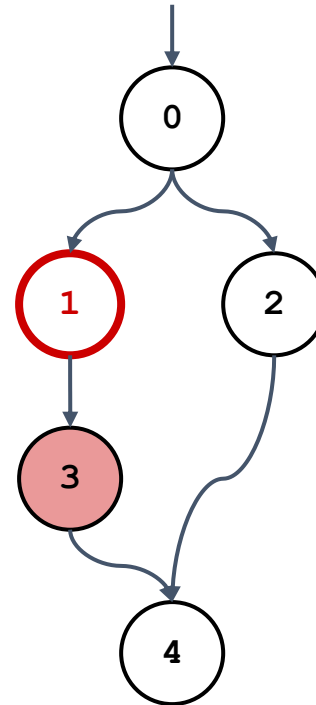
Node 1 dominates nodes 1 and 3.
It does not dominate 4 because
there is another path that reaches
it.



Strict Dominance

- A node x *strictly dominates* a node y iff x dominates y and $x \neq y$.

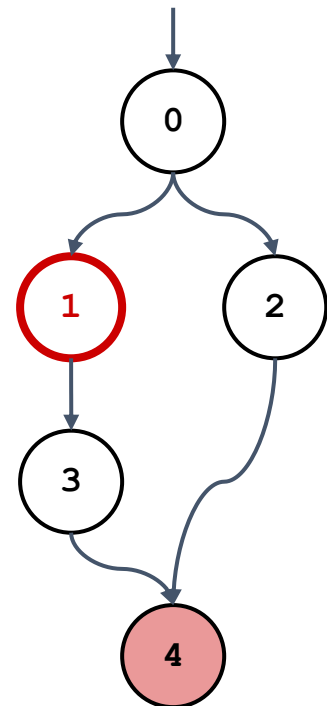
Node 1 only strictly dominates node 3 because it is the only dominated node that is not equal to 1.



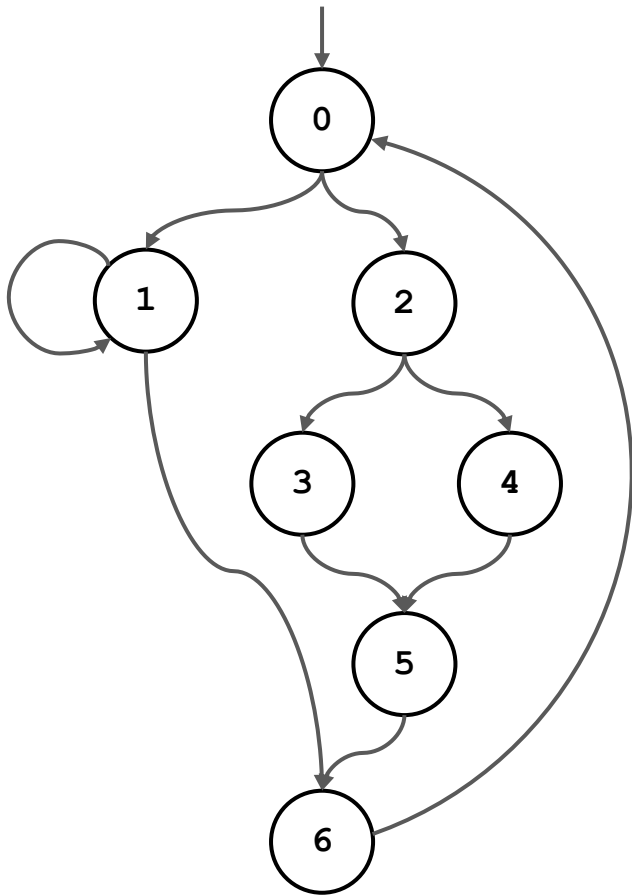
Dominance Frontiers

- A node \mathbf{y} is in the *dominance frontier* of node \mathbf{x} iff \mathbf{x} dominates an immediate predecessor of \mathbf{y} but \mathbf{x} does not strictly dominate \mathbf{y} .
- Essentially, the border between dominated and non-dominated nodes
 - Note: a node can be in its own dominance frontier!
- This is where phi function merging is necessary

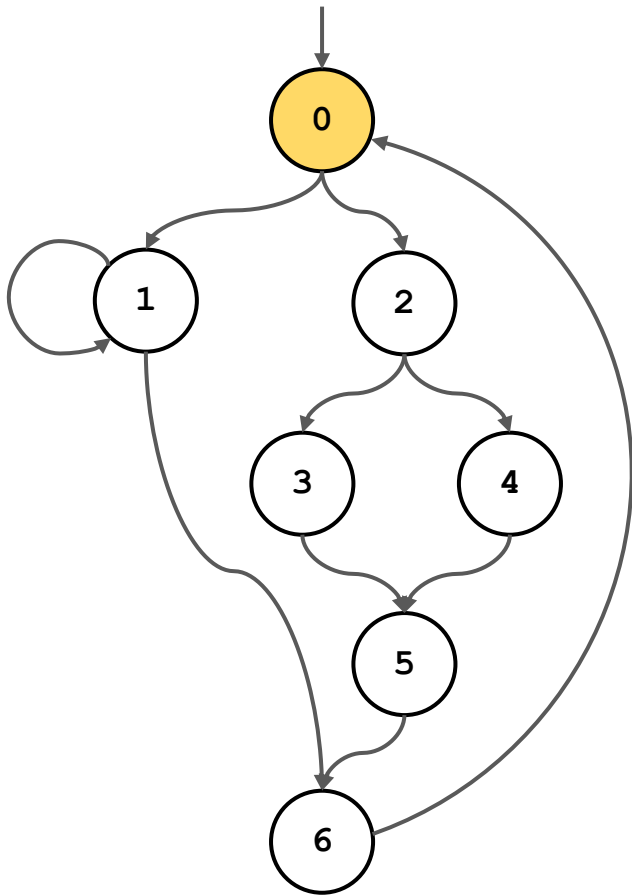
Node 4 is in the dominance frontier of node 1 because an immediate predecessor (node 3) is dominated by 1.



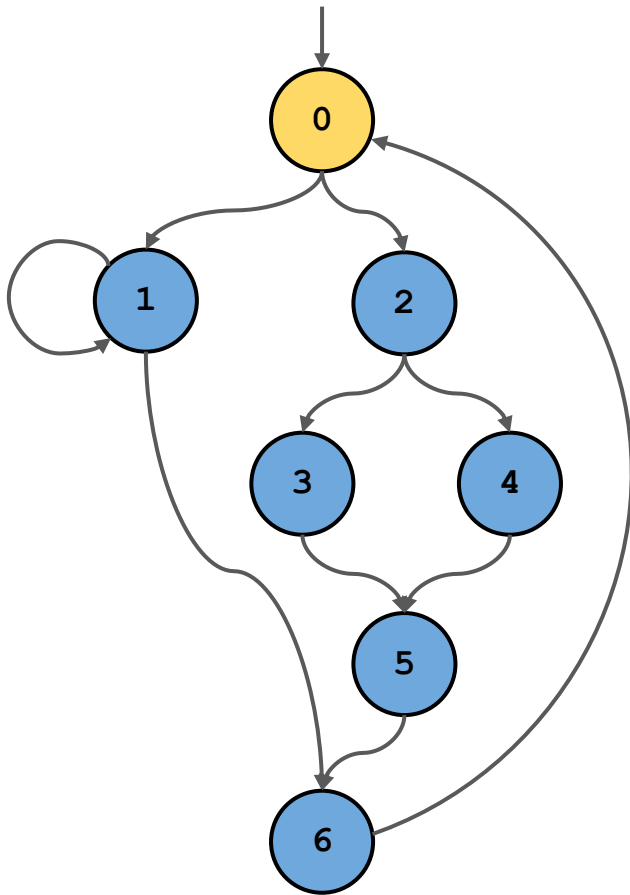
Problem 2(a)



NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0		
1		
2		
3		
4		
5		
6		



NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0		
1		
2		
3		
4		
5		
6		

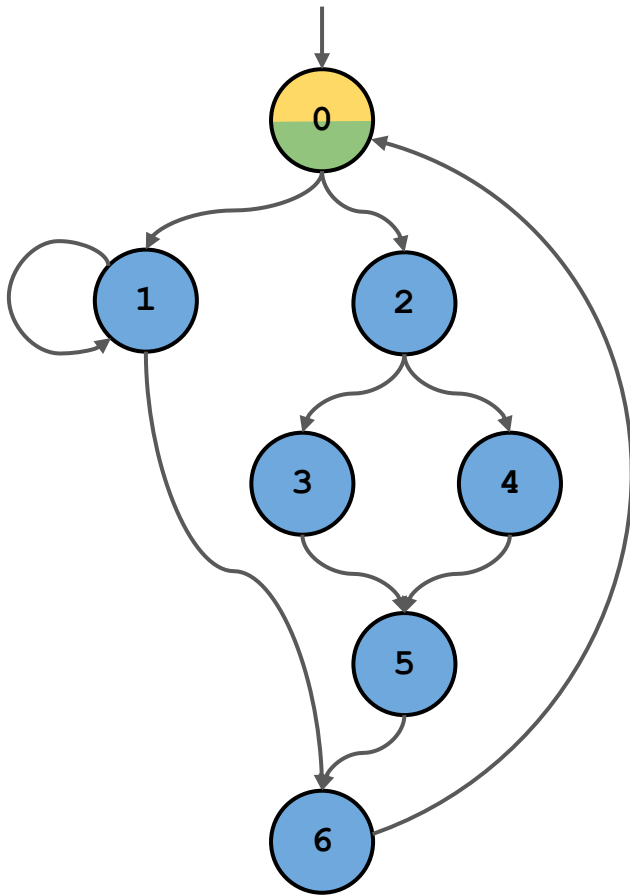


NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5, 6	
1		
2		
3		
4		
5		
6		

A node \mathbf{x} *dominates* a node \mathbf{y} iff every path from the entry point of the control flow graph to \mathbf{y} includes \mathbf{x} .

A node \mathbf{x} *strictly dominates* a node \mathbf{y} iff \mathbf{x} dominates \mathbf{y} and $\mathbf{x} \neq \mathbf{y}$

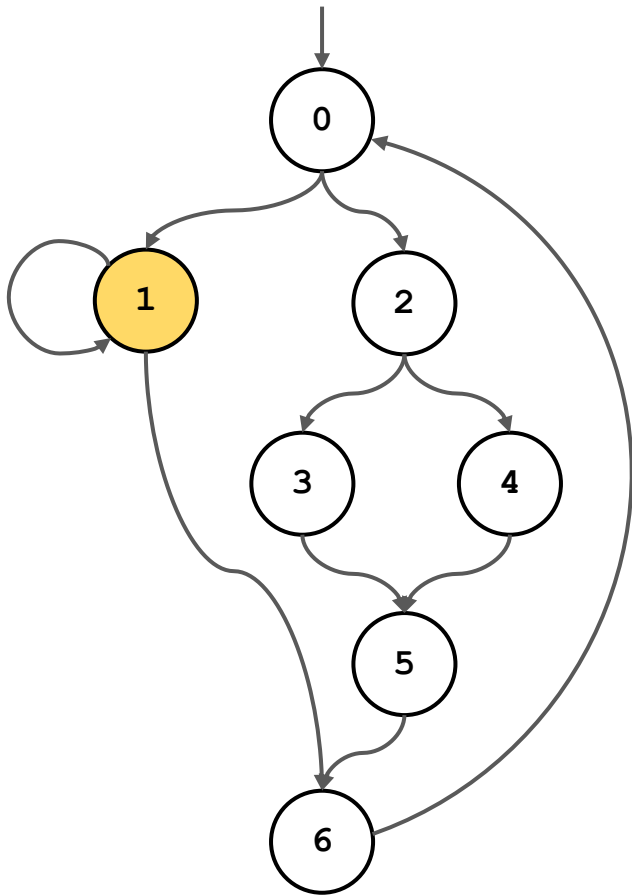
Need to go through 0 to get through 1, 2, 3, 4, 5, 6 and 0 cannot strictly dominate itself



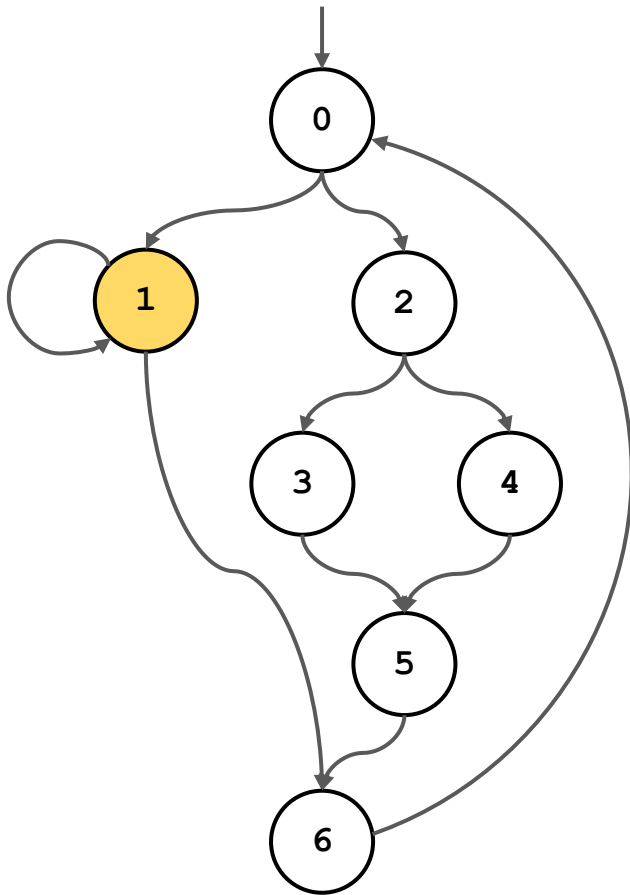
NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5, 6	0
1		
2		
3		
4		
5		
6		

A node Y is in the *dominance frontier* of node X iff X dominates an immediate predecessor of Y but X does not strictly dominate Y .
 A node 0 is in the *dominance frontier* of node 0 iff 0 dominates an immediate predecessor (6) of 0 but 0 does not strictly dominate 0

0 dominates 6, 6 is an immediate predecessor of 0, 0 does not strictly dominate 0



NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5, 6	0
1		
2		
3		
4		
5		
6		

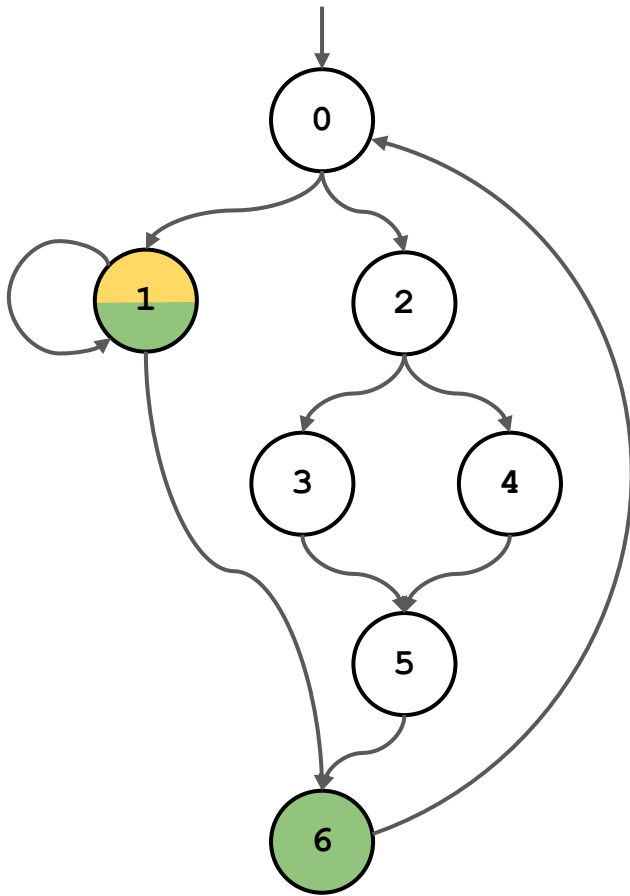


NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5, 6	0
1	\emptyset	
2		
3		
4		
5		
6		

A node \mathbf{x} *dominates* a node \mathbf{y} iff every path from the entry point of the control flow graph to \mathbf{y} includes \mathbf{x} .

A node \mathbf{x} *strictly dominates* a node \mathbf{y} iff \mathbf{x} dominates \mathbf{y} and $\mathbf{x} \neq \mathbf{y}$

1 does not dominate 6 because there is a path from 5 that doesn't include 1. 1 does not strictly dominate itself

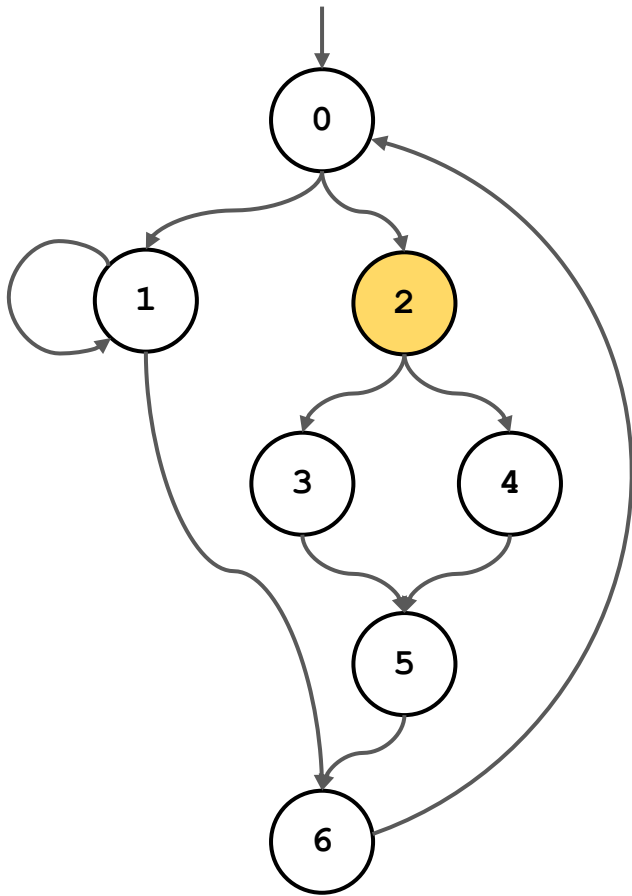


NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5, 6	0
1	\emptyset	1, 6
2		
3		
4		
5		
6		

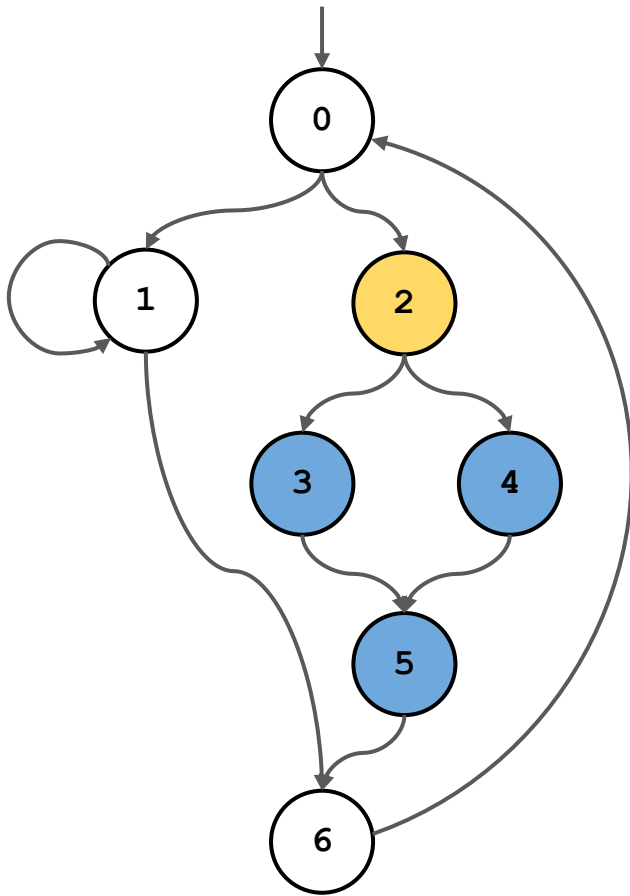
A node \mathbf{Y} is in the *dominance frontier* of node \mathbf{X} iff \mathbf{X} dominates an immediate predecessor of \mathbf{Y} but \mathbf{X} does not strictly dominate \mathbf{Y} .

$X = 1, Y = 6$, 1 dominates 1, 1 is an immediate predecessor of 6, 1 does not strictly dominate 6

$X = 1, Y = 1$, 1 dominates 1, 1 is an immediate predecessor of 1, 1 does not strictly dominate 1



NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5, 6	0
1	∅	1, 6
2		
3		
4		
5		
6		

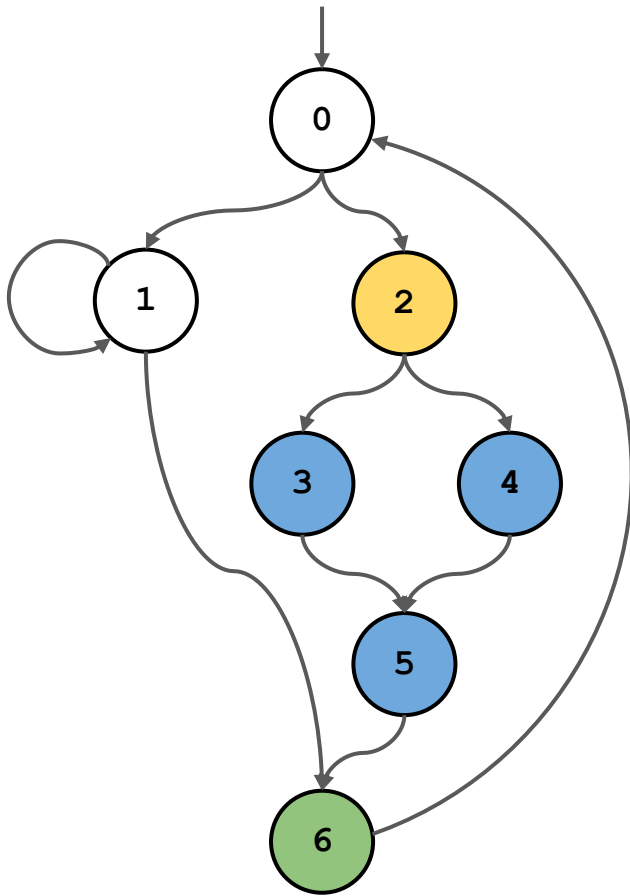


NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5, 6	0
1	\emptyset	1, 6
2	3, 4, 5	
3		
4		
5		
6		

A node \mathbf{x} *dominates* a node \mathbf{y} iff every path from the entry point of the control flow graph to \mathbf{y} includes \mathbf{x} .

A node \mathbf{x} *strictly dominates* a node \mathbf{y} iff \mathbf{x} dominates \mathbf{y} and $\mathbf{x} \neq \mathbf{y}$

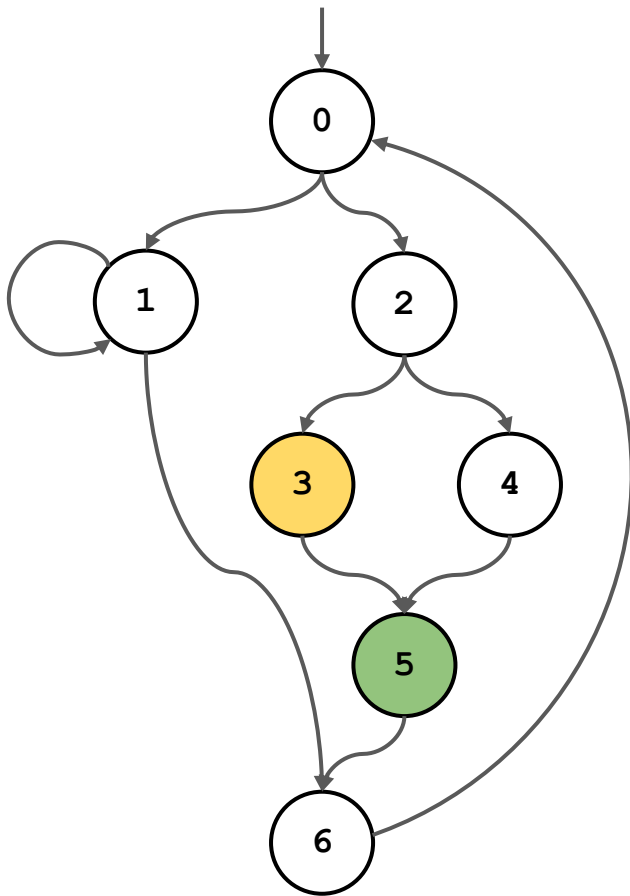
Need to go through 2 to get through 3, 4, 5 and 2 cannot strictly dominate itself



NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5, 6	0
1	\emptyset	1, 6
2	3, 4, 5	6
3		
4		
5		
6		

A node Y is in the *dominance frontier* of node X iff X dominates an immediate predecessor of Y but X does not strictly dominate Y .

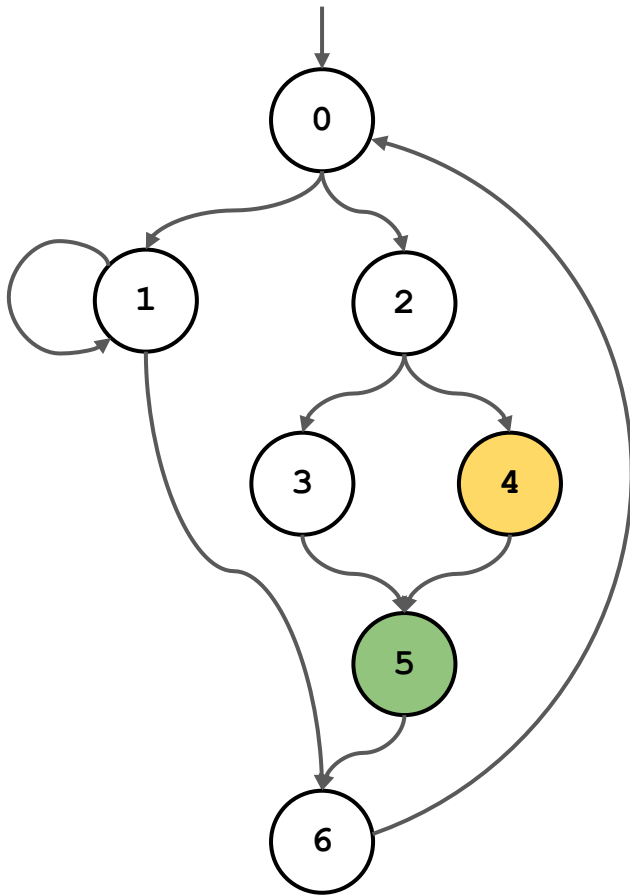
$X = 2, Y = 6$, 2 dominates 5, 5 is an immediate predecessor of 6, 2 does not strictly dominate 6



NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5, 6	0
1	∅	1, 6
2	3, 4, 5	6
3	∅	5
4		
5		
6		

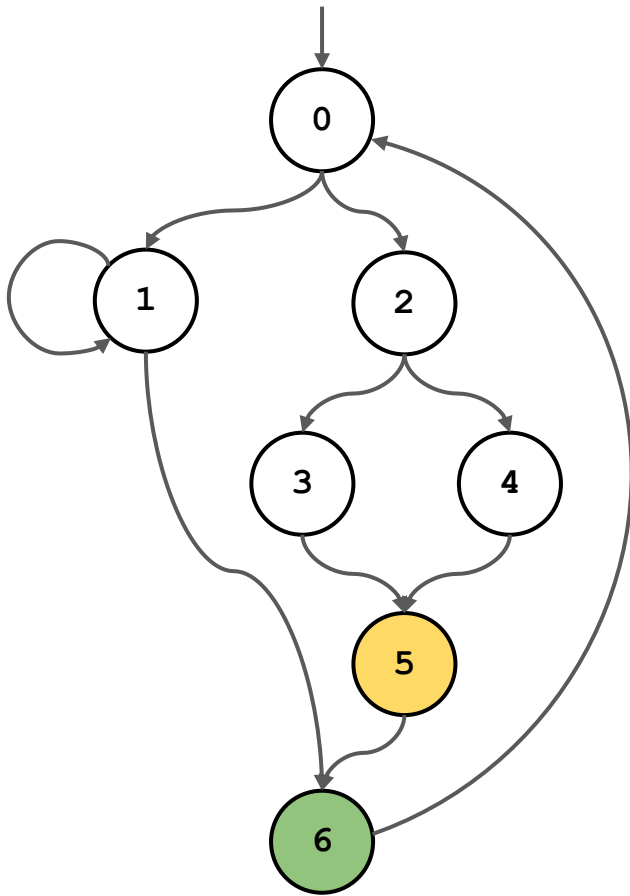
3 does not strictly dominate 5 (path through 4) and therefore does not strictly dominate anything else

3 dominates 3, 3 is an immediate predecessor of 5, 3 does not strictly dominate 5



NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5, 6	0
1	∅	1, 6
2	3, 4, 5	6
3	∅	5
4	∅	5
5		
6		

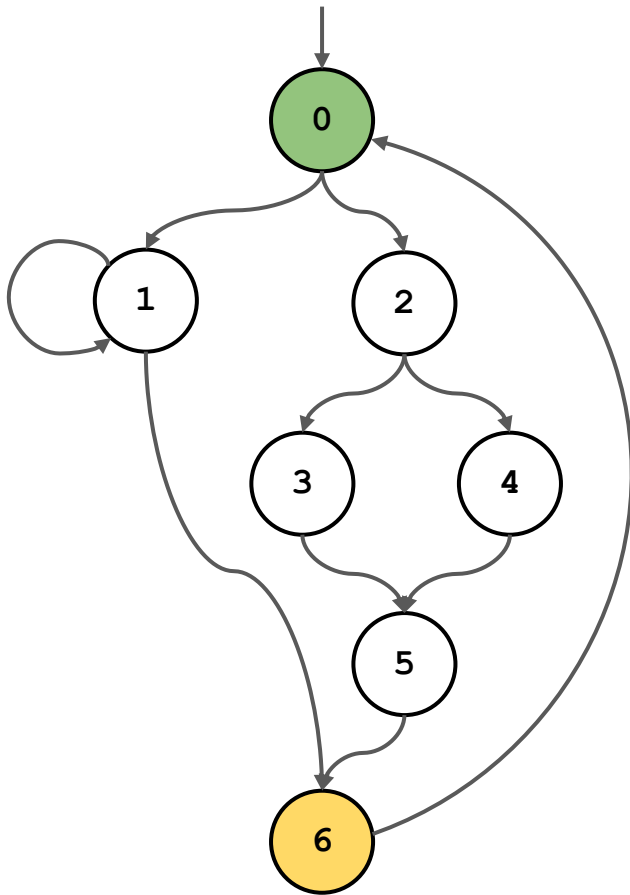
Same as previous slide but with 4 instead of 3



NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5, 6	0
1	∅	1, 6
2	3, 4, 5	6
3	∅	5
4	∅	5
5	∅	6
6		

5 does not strictly dominate 6 (path through 1) and therefore does not strictly dominate anything else

5 dominates 5, 5 is an immediate predecessor of 6, 5 does not strictly dominate 6



NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5, 6	0
1	∅	1, 6
2	3, 4, 5	6
3	∅	5
4	∅	5
5	∅	6
6	∅	0

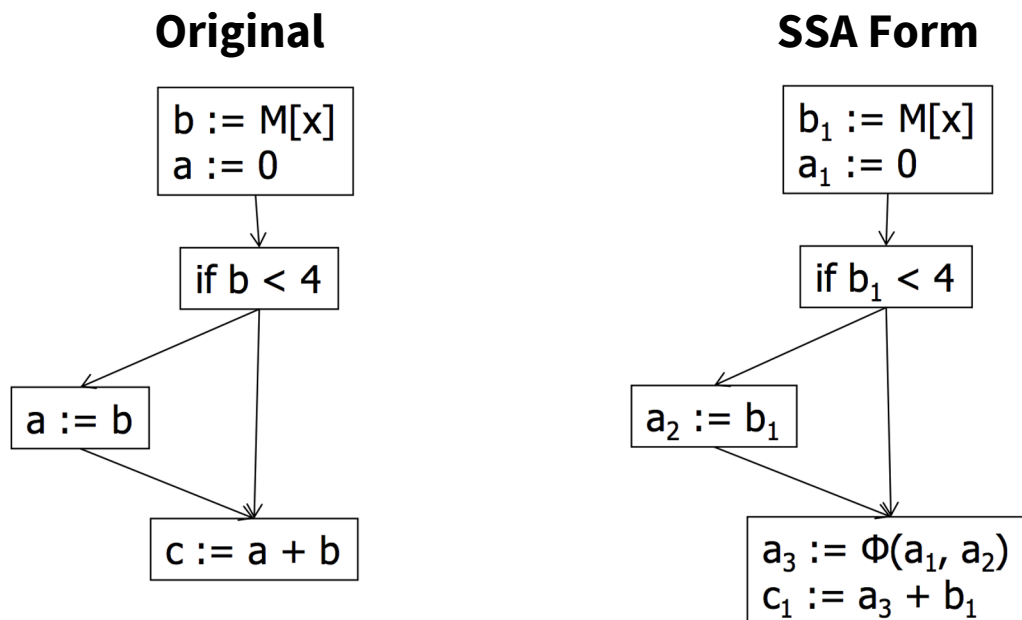
6 does not strictly dominate 0 (path through 0) and therefore does not strictly dominate anything else

6 dominates 6, 6 is an immediate predecessor of 0, 6 does not strictly dominate 0

Problem 2(b)

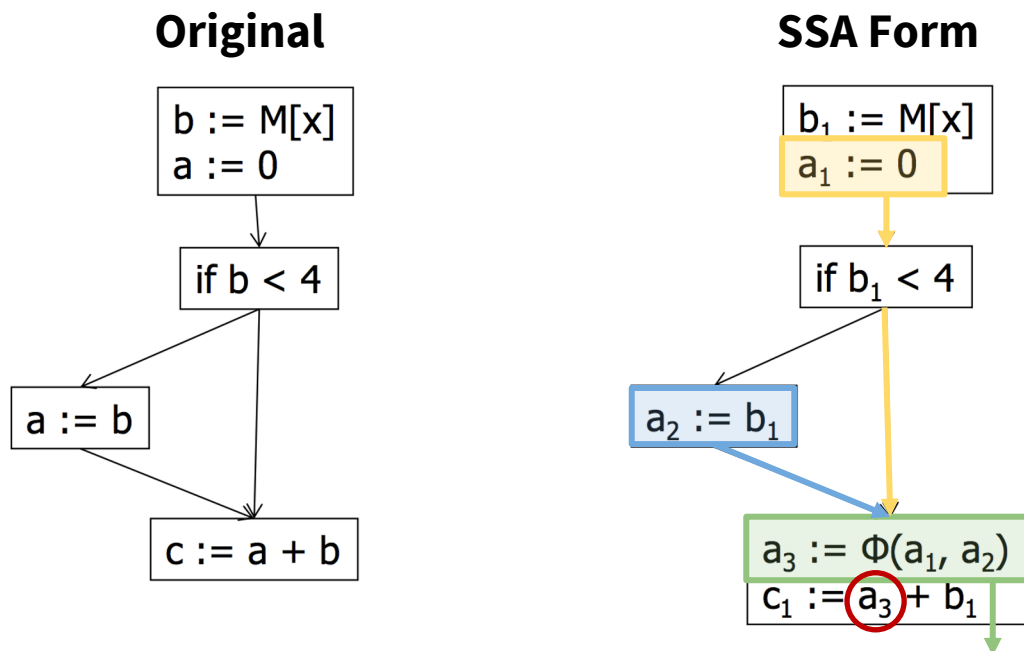
Phi-Functions

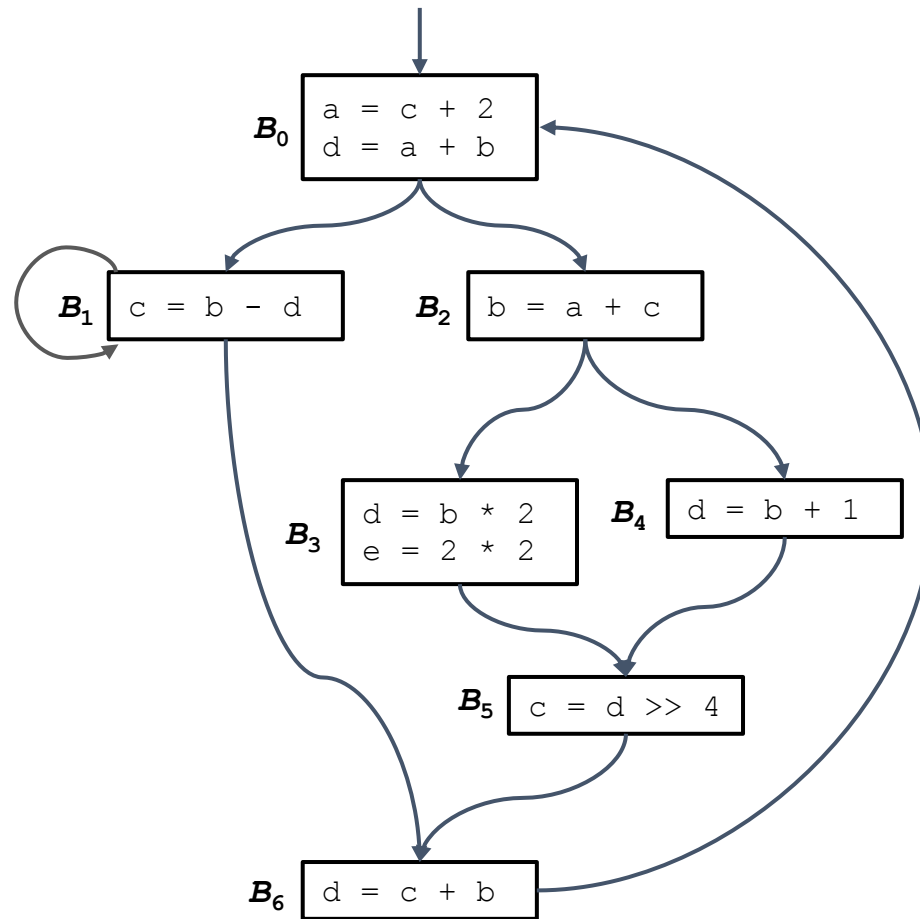
- A way to represent multiple possible values for a certain definition
 - Not a “real” instruction – just a form of bookkeeping needed for SSA



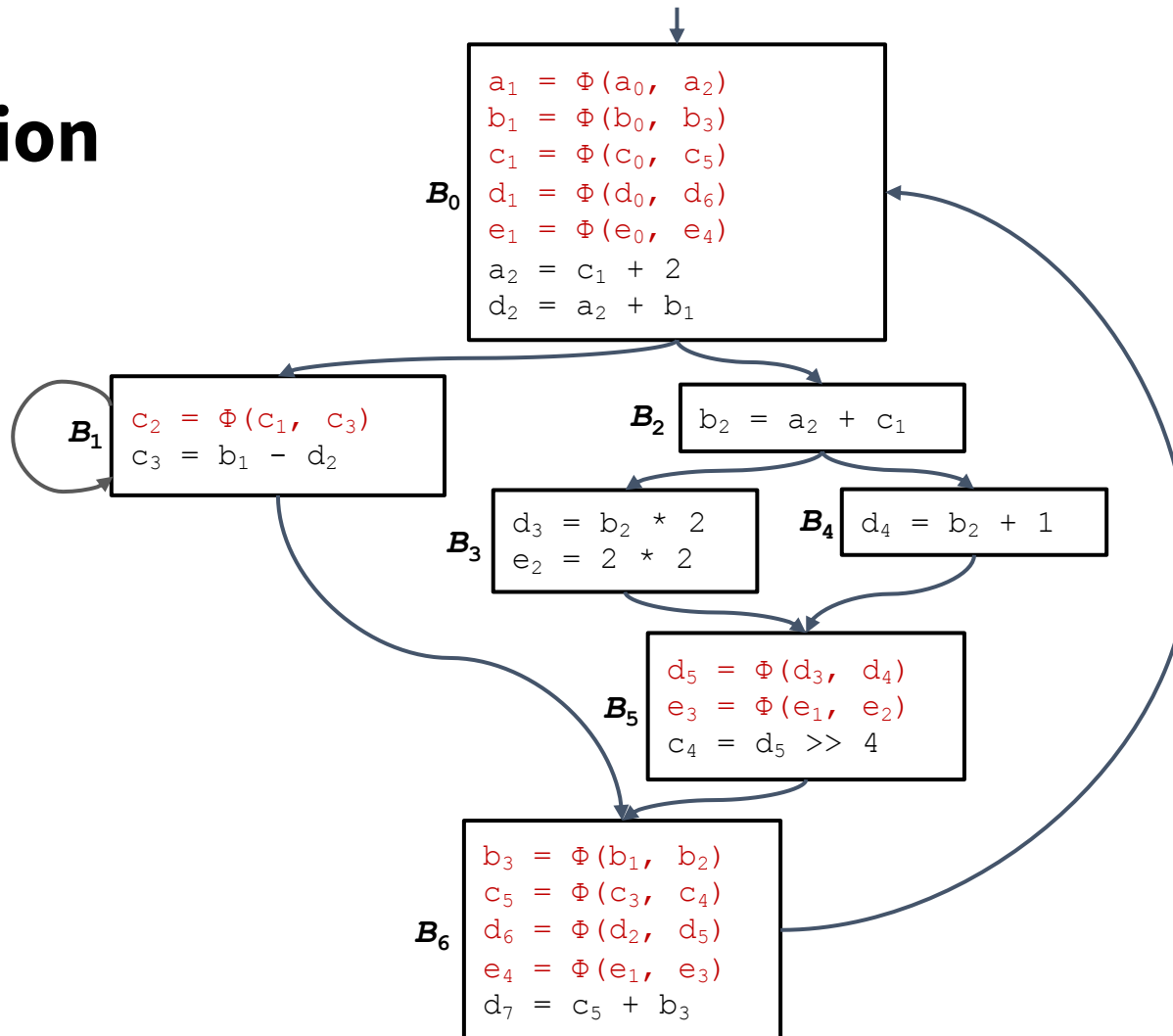
Where to place Phi-Functions?

- Wherever a variable has multiple possible definitions entering a block
 - Inefficient (and unnecessary!) to consider all possible phi-functions at the start of each block





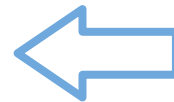
Solution



Converting to SSA

1

Compute the dominance frontier of each node



Already done (in problem 2a)

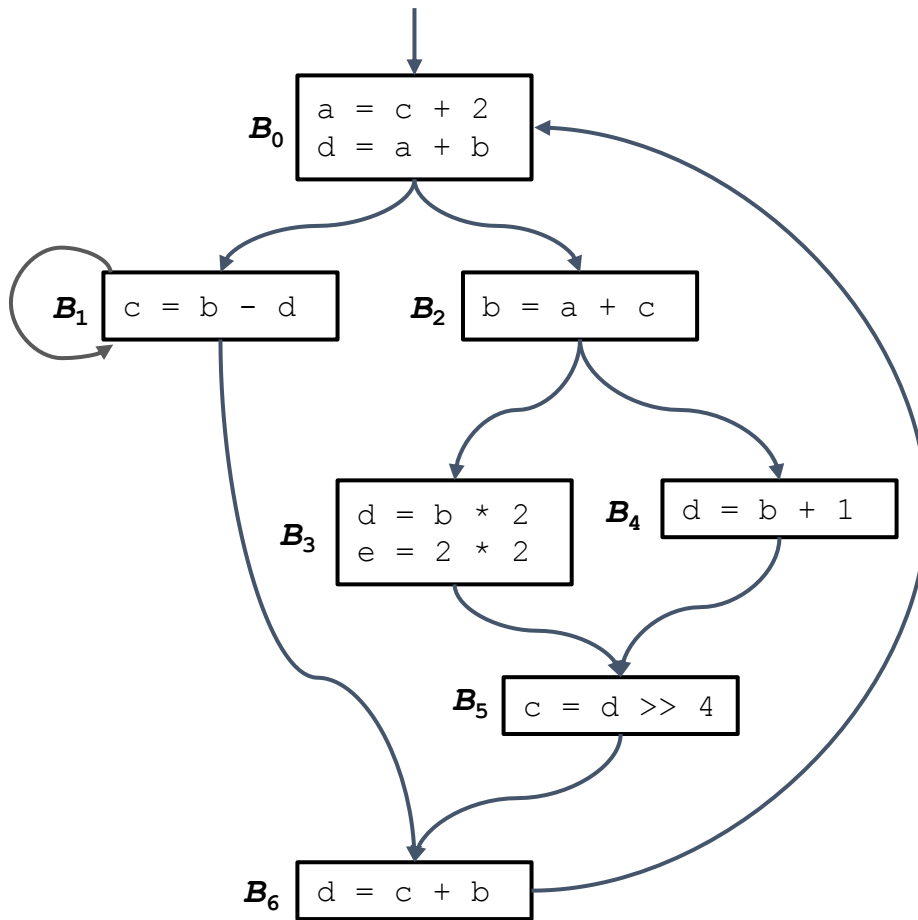
2

Determine which variables need merging in each node

3

Assign numbers to definitions and add phi functions

Step 1: Dominance Frontiers



NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5, 6	0
1	\emptyset	1, 6
2	3, 4, 5	6
3	\emptyset	5
4	\emptyset	5
5	\emptyset	6
6	\emptyset	0

Converting to SSA

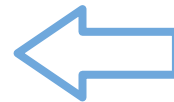
1

Compute the dominance frontier of each node



2

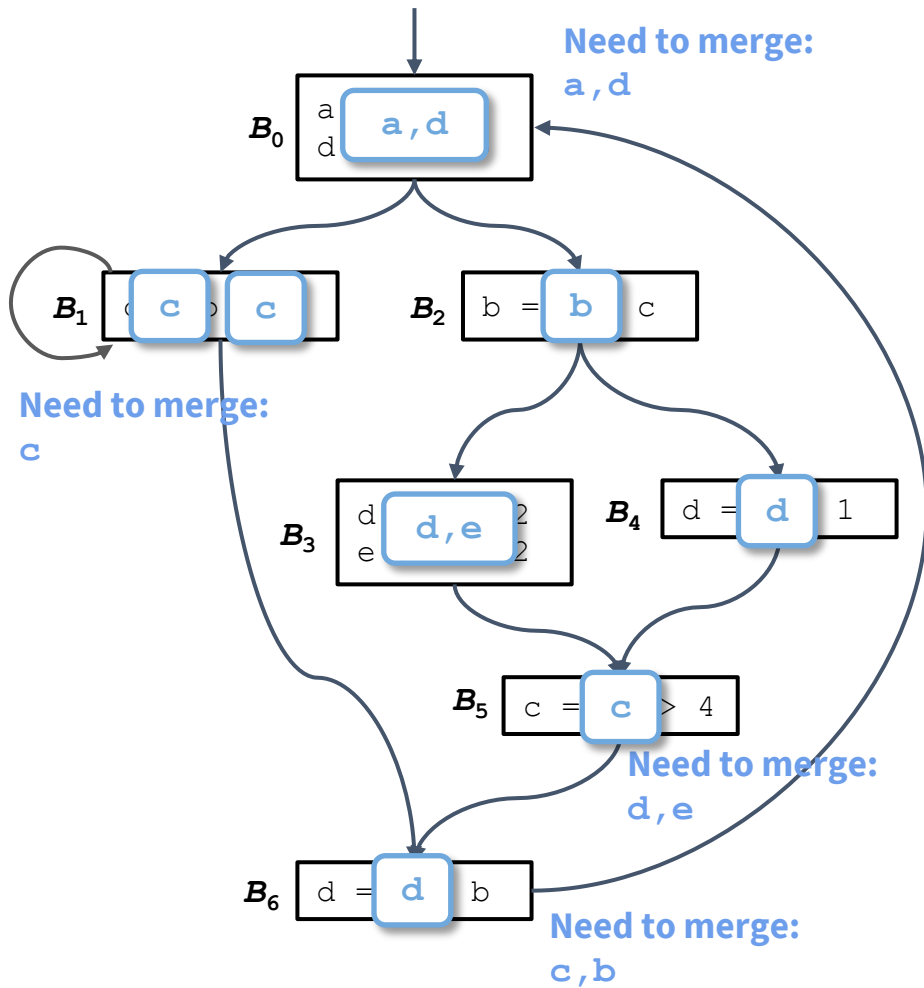
Determine which variables need merging in each node



We will compute using the dominance frontiers

3

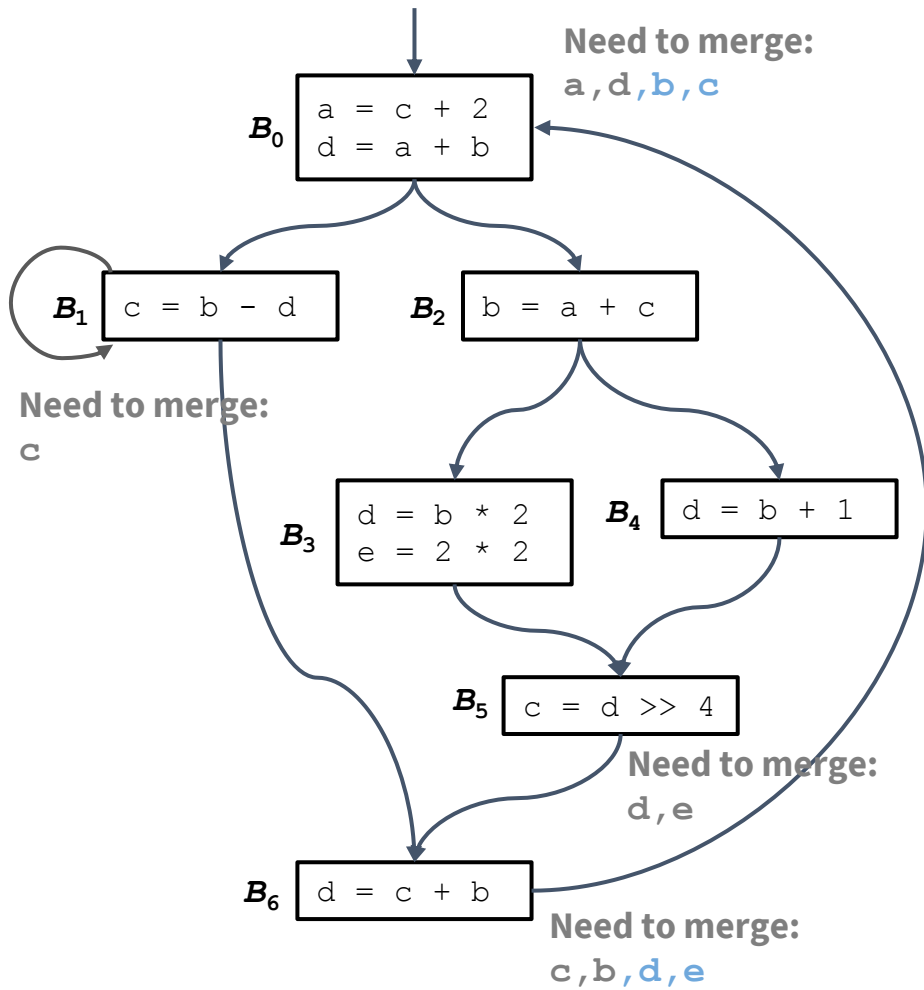
Assign numbers to definitions and add phi functions



Step 2: Determine Necessary Merges

ITERATION 1: Each node in the dominance frontier of node X will merge any definitions created in node X.

NODE		DOMINANCE FRONTIER
0	a, d	0
1	c	1, 6
2	b	6
3	d, e	5
4	d	5
5	c	6
6	d	0



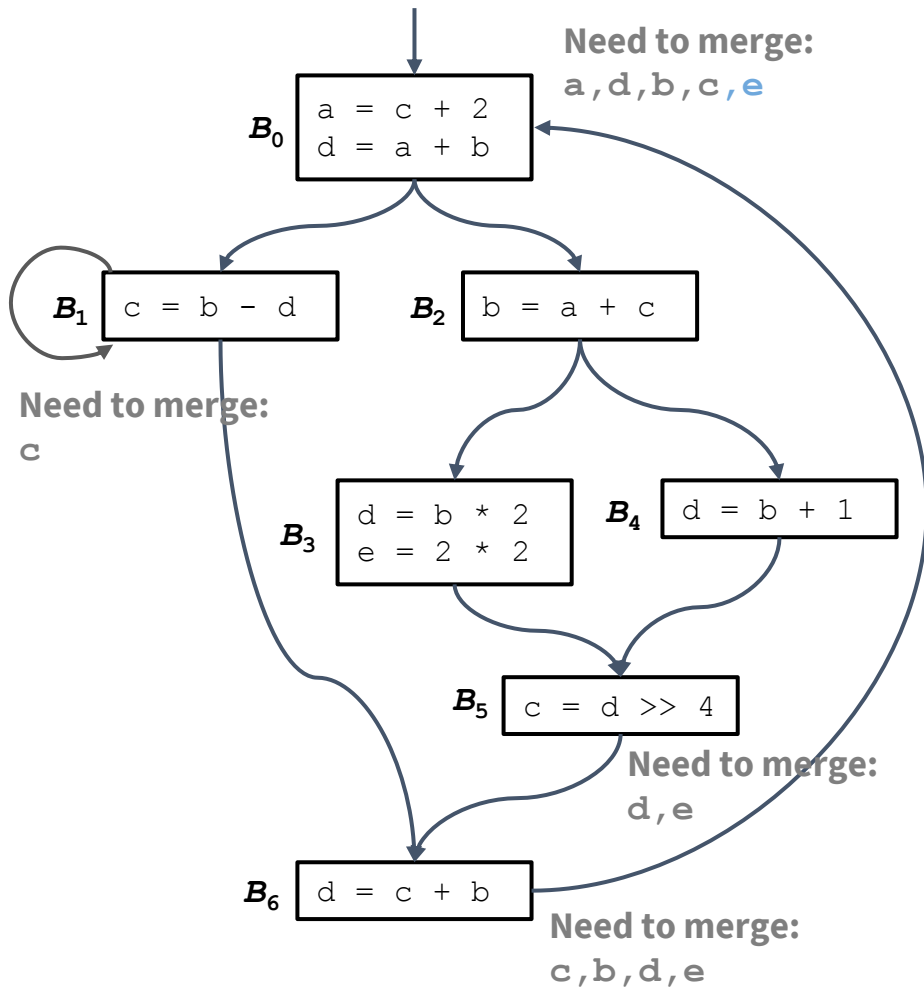
Step 2: Determine Necessary Merges

ITERATION 2: Each merge will create a new definition, which may need merging again.

NODE	DOMINANCE FRONTIER
0	0
1	1, 6
2	6
3	5
4	5
5	6
6	0

Annotations for Node 5: d, e

Annotations for Node 6: b, c



Step 2: Determine Necessary Merges

ITERATION 3: Each merge will create a new definition, which may need merging again.

NODE	DOMINANCE FRONTIER
0	0
1	1, 6
2	6
3	5
4	5
5	6
6	0

Annotation: A blue arrow points from Node 6 to Node 0, labeled d, e .

Converting to SSA

1

Compute the dominance frontier of each node



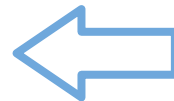
2

Determine which variables need merging in each node



3

Assign numbers to definitions and add phi functions



Place phi functions first, then increment subscripts

Step 3: Assign Definition Numbers

Merges go first, and each successive definition of a variable should increment its index by 1.

$$\mathbf{B}_0 \quad \begin{array}{l} a = c + 2 \\ d = a + b \end{array}$$

Need to merge:
a, b, c, d, e



$$\mathbf{B}_0 \quad \begin{array}{l} a_1 = \Phi(a_0, a_2) \\ b_1 = \Phi(b_0, b_3) \\ c_1 = \Phi(c_0, c_5) \\ d_1 = \Phi(d_0, d_7) \\ e_1 = \Phi(e_0, e_4) \\ a_2 = c_1 + 2 \\ d_2 = a_2 + b_1 \end{array}$$

Note: these subscripts determined after doing the rest of the CFG!

Step 3: Assign Definition Numbers

Merges go first, and each successive definition of a variable should increment its index by 1.

$$B_1 \quad c = b - d$$

Need to merge:

c



$$B_1 \quad \begin{array}{l} c_2 = \Phi(c_1, c_3) \\ c_3 = b_1 - d_2 \end{array}$$

Note: must merge its own (later) definition because of the back-edge!

Step 3: Assign Definition Numbers

Merges go first, and each successive definition of a variable should increment its index by 1.

$$B_2 \quad b = a + c$$



$$B_2 \quad b_2 = a_2 + c_1$$

Nothing to merge

Step 3: Assign Definition Numbers

Merges go first, and each successive definition of a variable should increment its index by 1.

$$\mathbf{B}_3 \begin{array}{l} d = b * 2 \\ e = 2 * 2 \end{array}$$



$$\mathbf{B}_3 \begin{array}{l} d_3 = b_2 * 2 \\ e_2 = 2 * 2 \end{array}$$

Nothing to merge

Step 3: Assign Definition Numbers

Merges go first, and each successive definition of a variable should increment its index by 1.

$$B_4 \quad d = b + 1$$



$$B_4 \quad d_4 = b_2 + 1$$

Nothing to merge

Step 3: Assign Definition Numbers

Merges go first, and each successive definition of a variable should increment its index by 1.

$$\mathbf{B}_5 \quad c = d \gg 4$$



$$\mathbf{B}_5 \quad \begin{array}{l} d_5 = \Phi(d_3, d_4) \\ e_3 = \Phi(e_1, e_2) \\ c_4 = d_5 \gg 4 \end{array}$$

Need to merge:
d, e

Step 3: Assign Definition Numbers

Merges go first, and each successive definition of a variable should increment its index by 1.

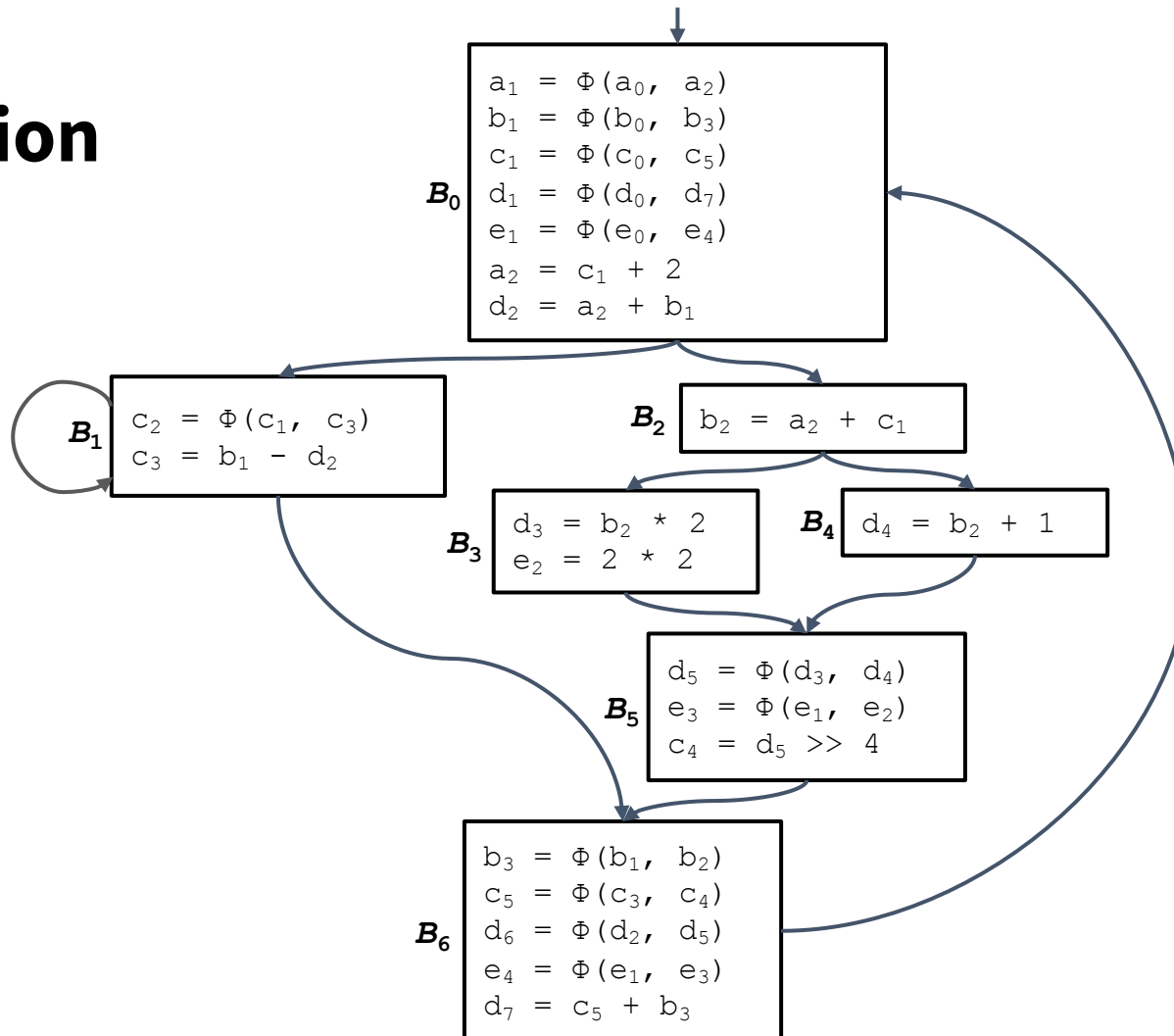
$$B_6 \quad \boxed{d = c + b}$$

Need to merge:
 b, c, d, e

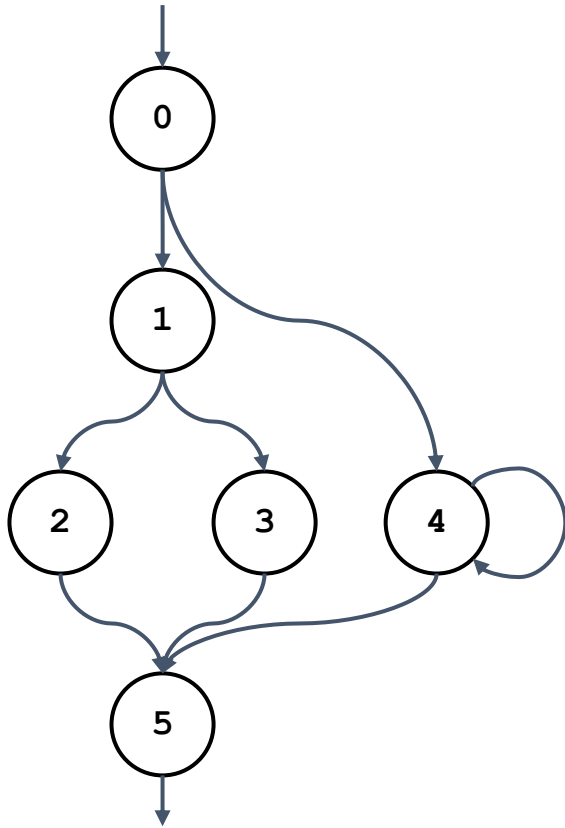


$$B_6 \quad \boxed{\begin{array}{l} b_3 = \Phi(b_1, b_2) \\ c_5 = \Phi(c_3, c_4) \\ d_6 = \Phi(d_2, d_5) \\ e_4 = \Phi(e_1, e_3) \\ d_7 = c_5 + b_3 \end{array}}$$

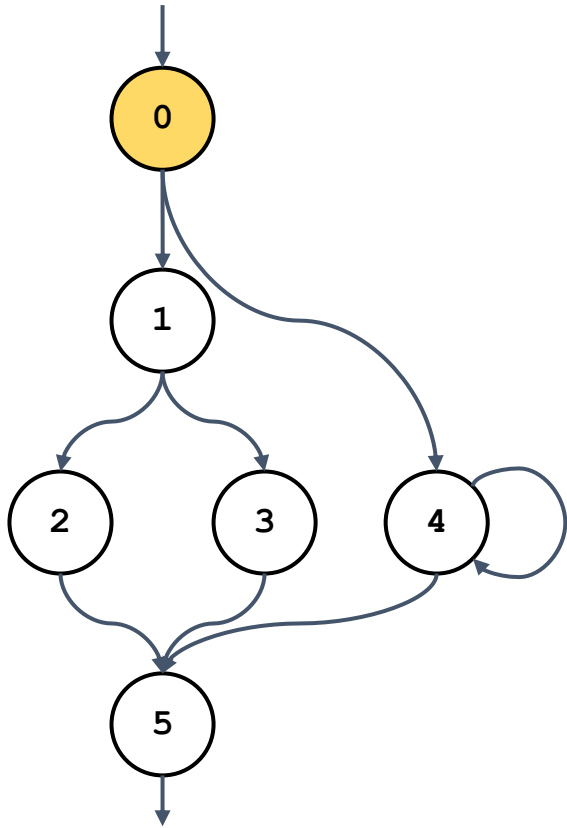
Solution



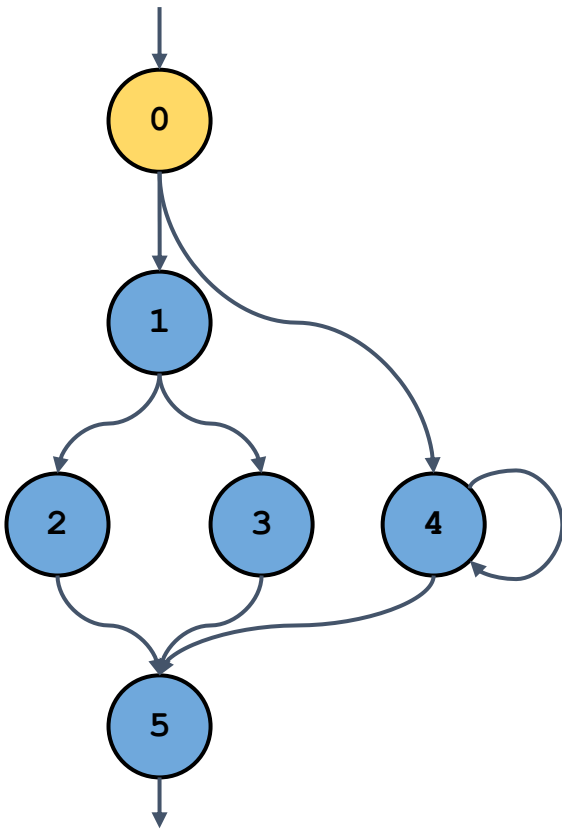
Problem 3



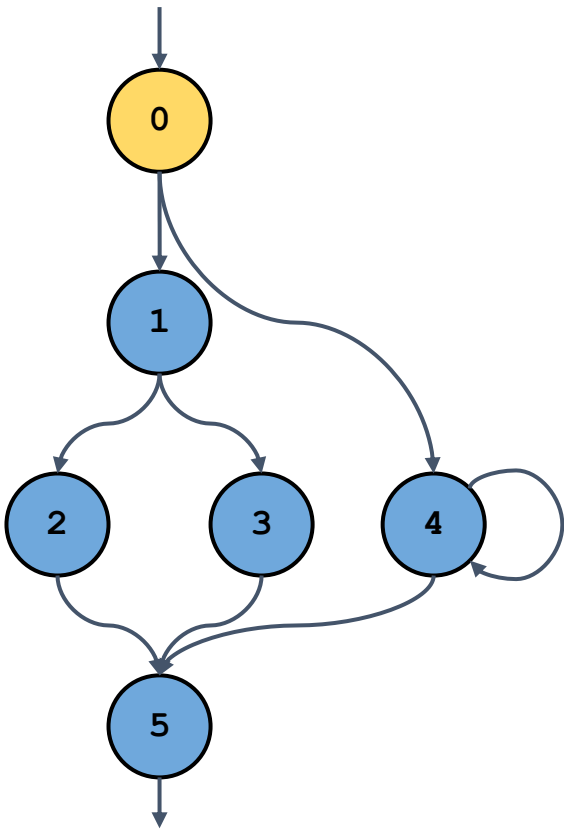
NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0		
1		
2		
3		
4		
5		



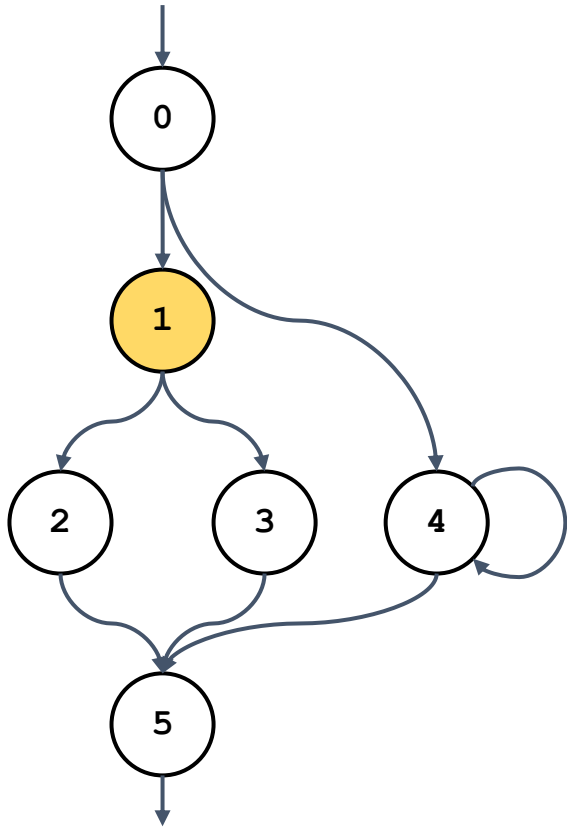
NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0		
1		
2		
3		
4		
5		



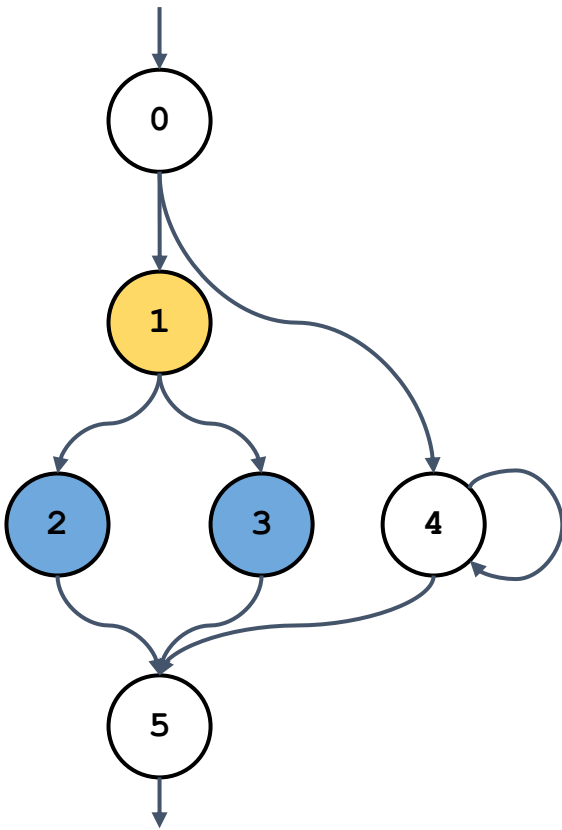
NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5	
1		
2		
3		
4		
5		



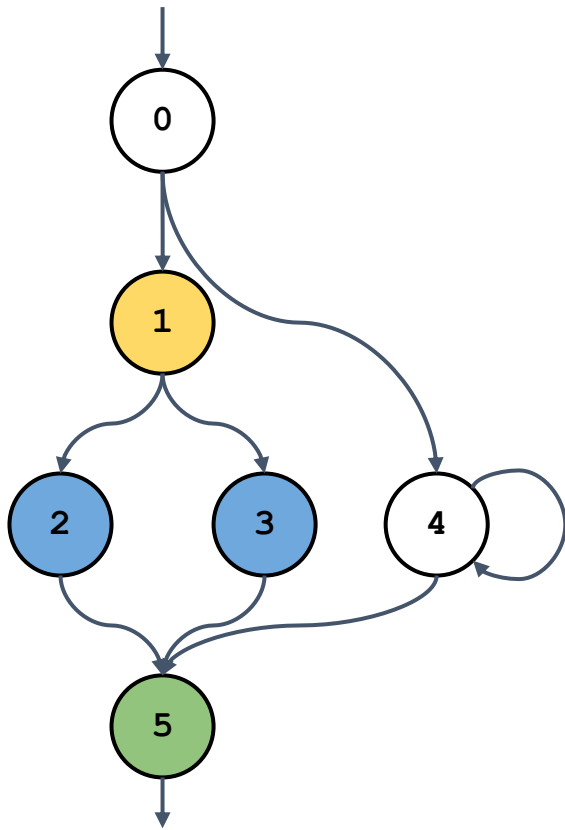
NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5	∅
1		
2		
3		
4		
5		



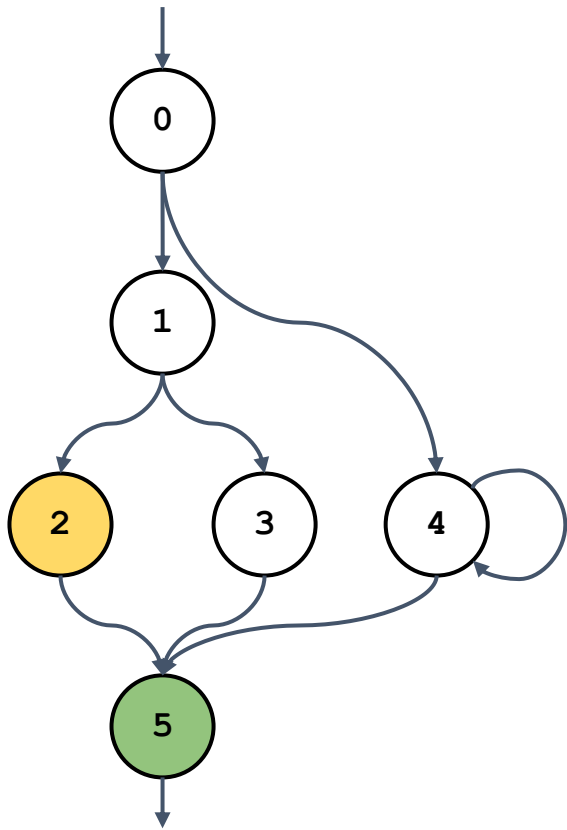
NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5	\emptyset
1		
2		
3		
4		
5		



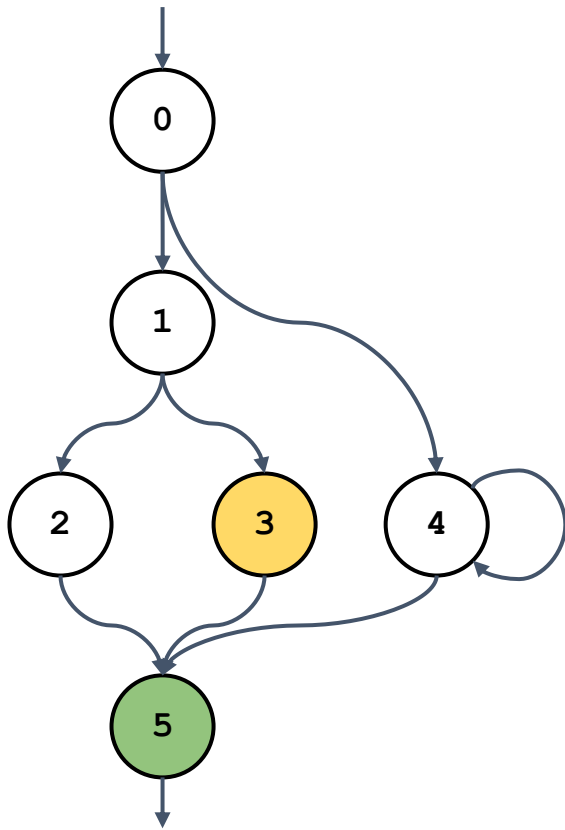
NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5	\emptyset
1	2, 3	
2		
3		
4		
5		



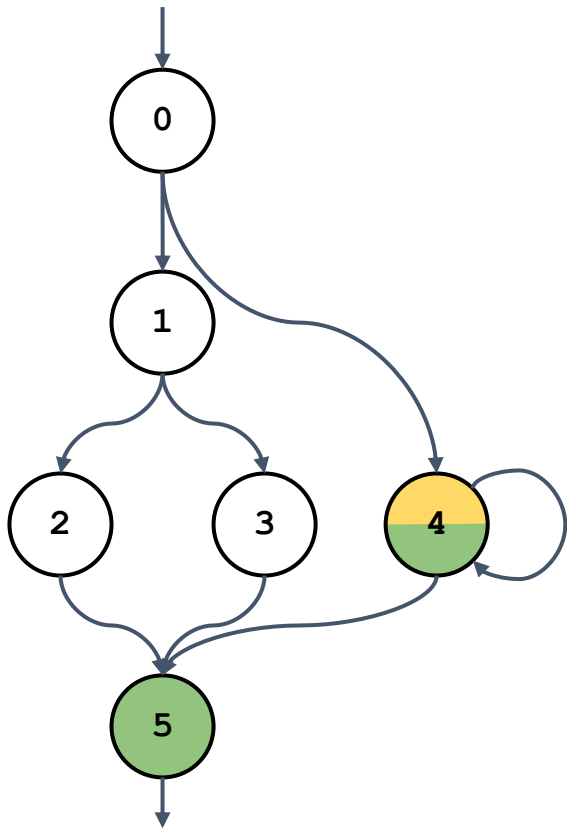
NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5	\emptyset
1	2, 3	5
2		
3		
4		
5		



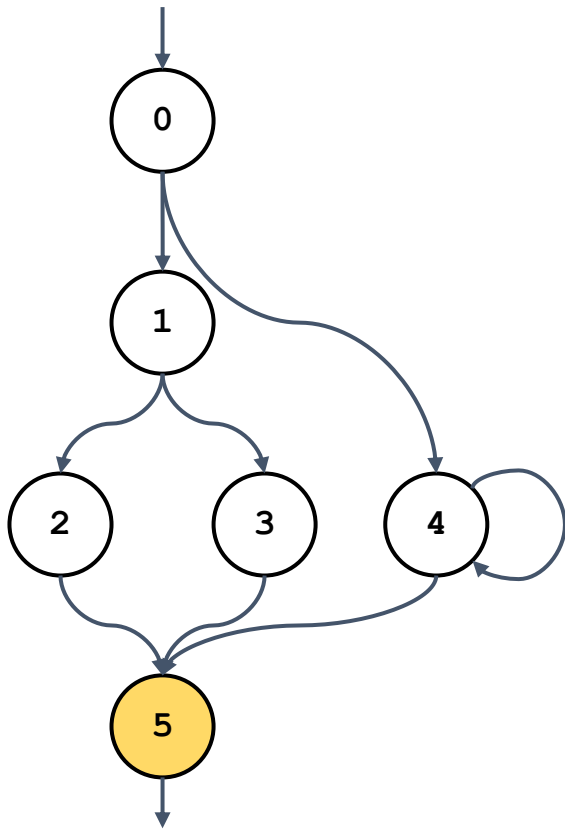
NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5	∅
1	2, 3	5
2	∅	5
3		
4		
5		



NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5	\emptyset
1	2, 3	5
2	\emptyset	5
3	\emptyset	5
4		
5		



NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5	∅
1	2, 3	5
2	∅	5
3	∅	5
4	∅	4, 5
5		



NODE	STRICTLY DOMINATES	DOMINANCE FRONTIER
0	1, 2, 3, 4, 5	\emptyset
1	2, 3	5
2	\emptyset	5
3	\emptyset	5
4	\emptyset	4, 5
5	\emptyset	\emptyset