# Dataflow Analysis + SSA

CSE 401/M501 Sections

# Announcements

- **401 CodeGen** deadline **– TUESDAY 11/26 at 11:59 pm**.  Commit/push/tag by Tuesday 11:59pm. If using late days, may submit at most two days late (depending on number of remaining late days).

  ➢ Once codegen is finished, we'll do an overall evaluation of your compiler, **all phases**, and rerun a comprehensive set of tests.  This **final evaluation is the major part** (half!) **of the overall project grade**.  So you need to fix any remaining bugs, from semantics/type checking all the way back to the scanner! (And your final evaluation will reflect these fixes)  🐛🕷🐜🦂🐞

- **401 report** due Monday, 12/02 at 11:59 pm **(no late days)**; M501 project/report due end of the last week of classes (please see Assignment page on course web for more details – report details will be posted shortly).

- HW4 due Thursday, 12/05 at 11:59 pm

# Review of Optimizations
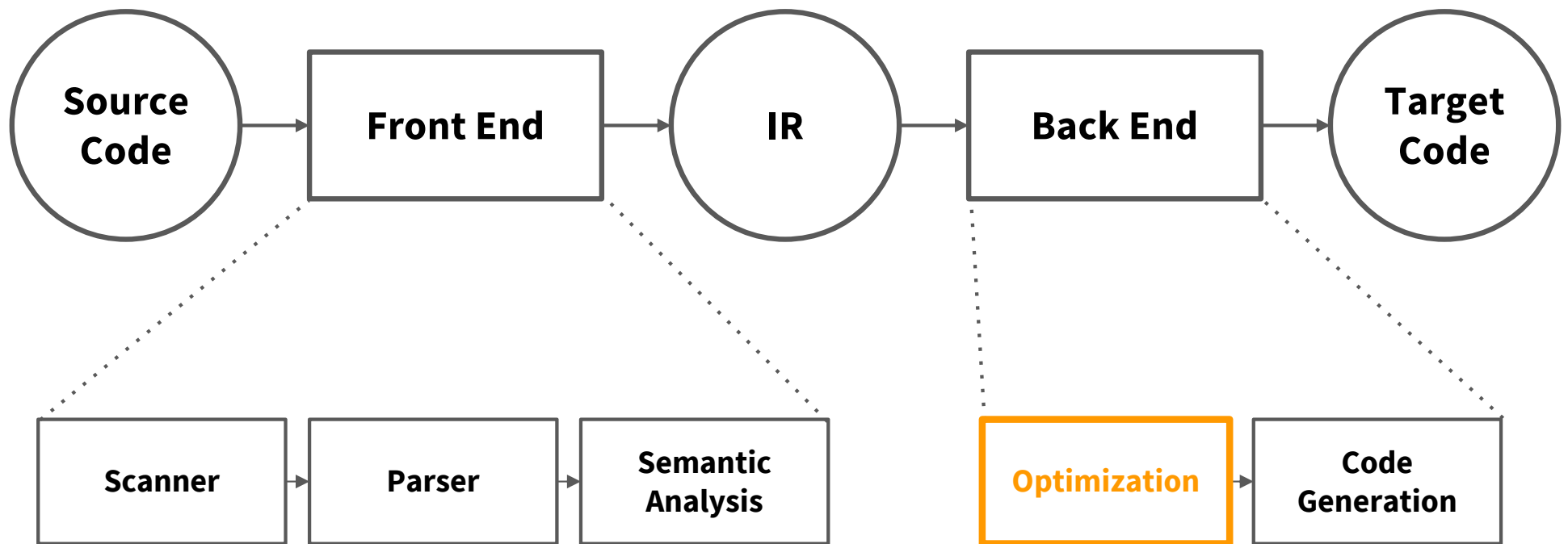
# Review of Optimizations

**Peephole**     A few Instructions

**Local**

**Intraprocedural / Global**

**Interprocedural**

# Review of Optimizations

| | |
|---:|:---|
| **Peephole** | A few Instructions |
| **Local** | A Basic Block |
| **Intraprocedural / Global** | |
| **Interprocedural** | |

# Review of Optimizations

|  |  |
|---|---|
| **Peephole** | A few Instructions |
| **Local** | A Basic Block |
| **Intraprocedural / Global** | A Function/Method |
| **Interprocedural** |  |

# Review of Optimizations

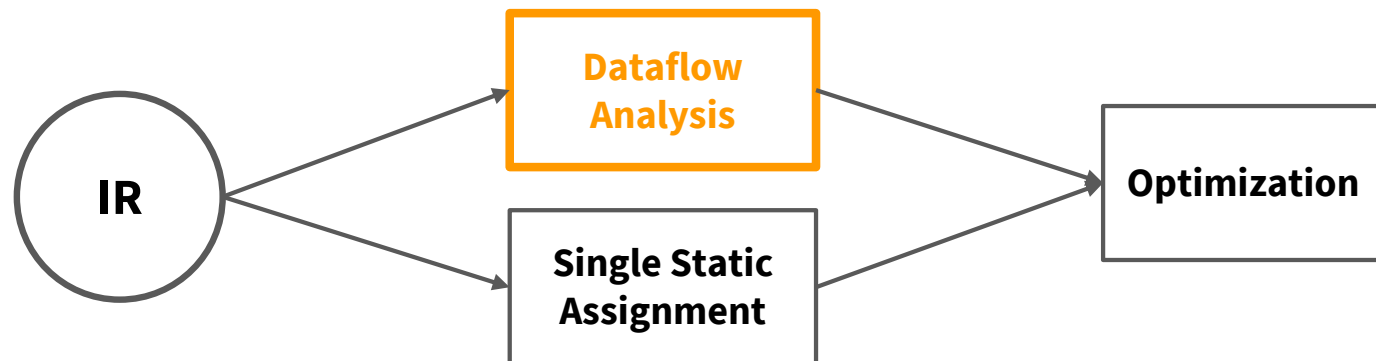|  |  |
|---|---|
| **Peephole** | A few Instructions |
| **Local** | A Basic Block |
| **Intraprocedural / Global** | A Function/Method |
| **Interprocedural** | A Program |

# 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
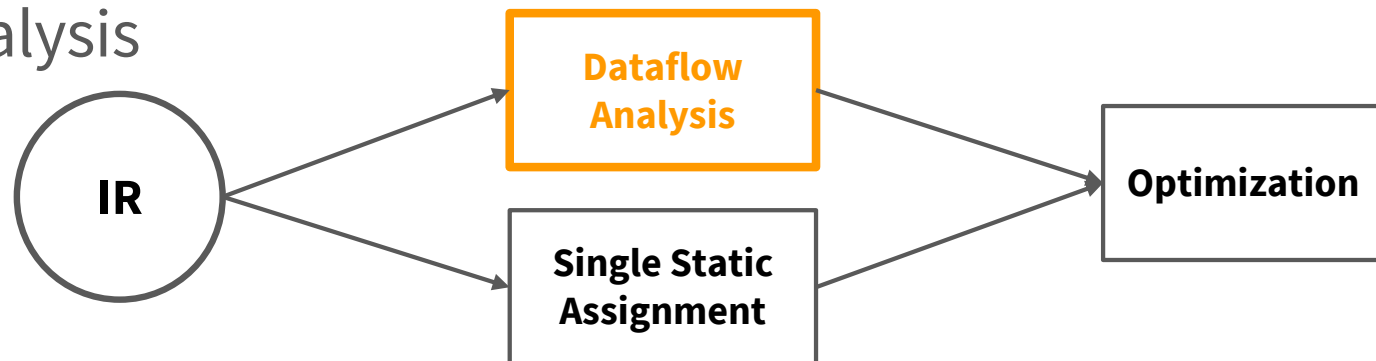
# Overview of Dataflow Analysis

- Basic Framework of Set Definitions (for a Basic Block $b$):

  - $\text{IN}(b)$: facts true on entry to $b$

  - $\text{OUT}(b)$: facts true on exit from $b$

  - $\text{GEN}(b)$: facts created (and not killed) in $b$

  - $\text{KILL}(b)$: facts killed in $b$

# Overview of Dataflow Analysis (continued)

- These are related by the equation
  - `OUT(`$b$`) = GEN(`$b$`) ` $\cup$ ` (IN(`$b$`) - KILL(`$b$`))`
  - Solve this iteratively for all blocks
  - Sometimes information propagates forward; sometimes backward
    - But will reach correct solution (fixed point) regardless of order in which blocks are considered

# 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 variable definition
  - The dataflow problem "Available Expressions" is designed for that

still: "x=y", "x=0"

```java
int x;

if (y > 0) {
    x = y;
} else {
    x = 0;
}

y = -1;
System.out.println(x);
```

# Equations for Reaching Definitions

- $\mathrm{IN}(b)$: the definitions reaching upon entering block b

- $\mathrm{OUT}(b)$: the definitions reaching upon exiting block b

- $\mathrm{GEN}(b)$: the definitions assigned and not killed in block b

- $\mathrm{KILL}(b)$: the definitions of variables overwritten in block b

$$\mathrm{IN}(b) = \bigcup_{p \in \mathrm{pred}(b)} \mathrm{OUT}(p)$$

$$\mathrm{OUT}(b) = \mathrm{GEN}(b) \cup \left( \mathrm{IN}(b) - \mathrm{KILL}(b) \right)$$

# 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 | | | | | |
| 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 | | | | | |
| 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 | | | 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 | | | 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 | | | 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
```

Is it possible to replace the use of *a* in block L1 with the constant 0?

| Block | GEN | KILL | IN (1) | OUT (1) | IN (2) | OUT (2) |
|-------|-----|------|--------|---------|--------|---------|
| L0 | L0 | | | 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
```

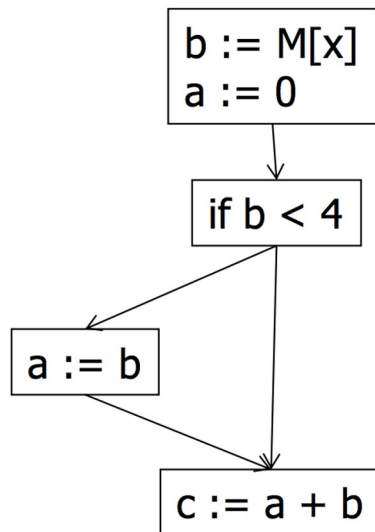Is it possible to replace the use of *a* in block L1 with the constant 0?

No. To determine this, we would look at the IN set for block L1 -- the fact that the IN set contains two definitions of 'a' (L0 and L3) means we cannot perform this constant propagation. In other words, more than one definition of 'a' is a reaching definition to block L1, and therefore performing constant propagation would only preserve one possible value of 'a' and the generated code would not be equivalent.

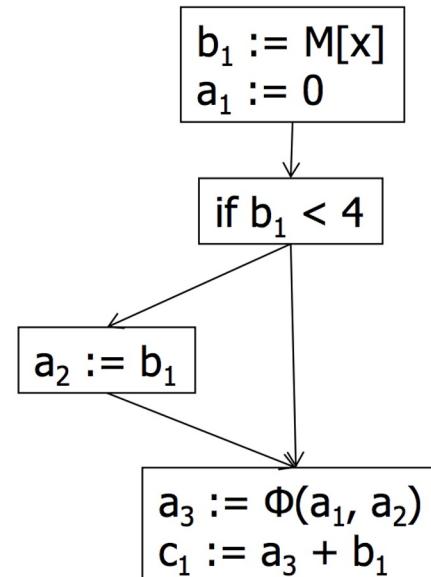| Block | GEN | KILL | IN (1) | OUT (1) | IN (2) | OUT (2) |
|-------|-----|------|--------|---------|--------|---------|
| L0 | L0 | | | L0 | | L0 |
| L1 | L1 | | L0 | L0, L1 | L0, L1, L2, L3 | L0, L1, L2, L3 |
| L2 | L2 | | L0, L1 | L0, L1, L2 | L0, L1, L2, L3 | L0, L1, L2, L3 |
| L3 | L3 | L0 | L0, 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 |

# Phi-Functions

- A way to represent <u>multiple possible values</u> for a certain definition
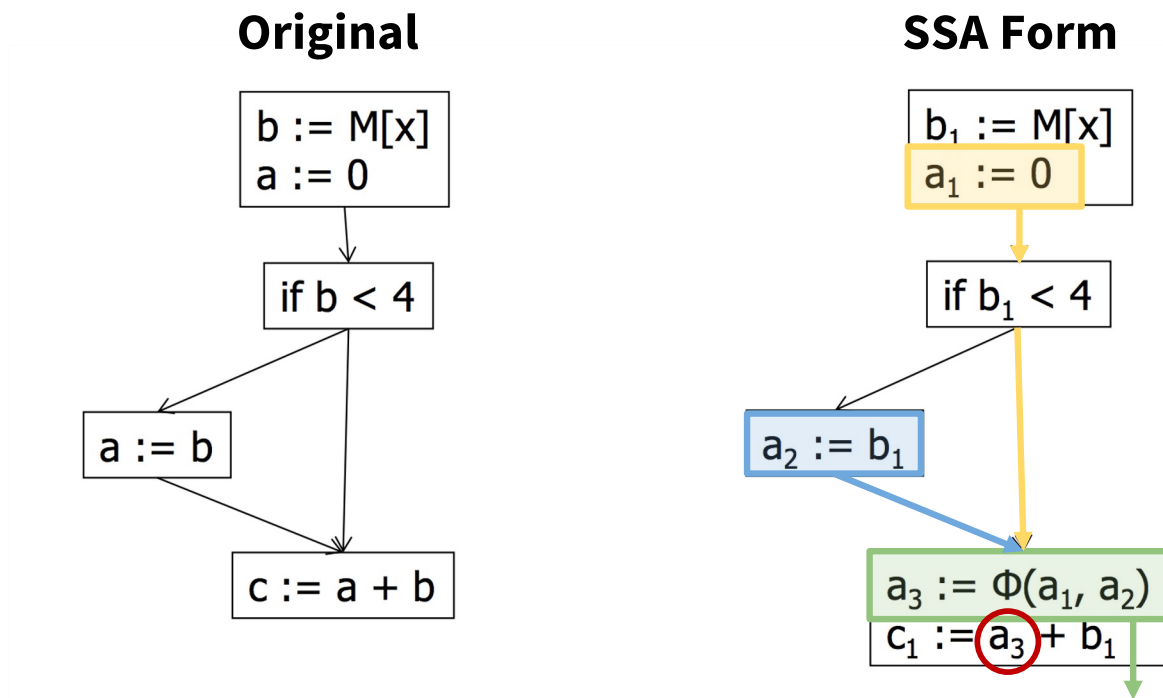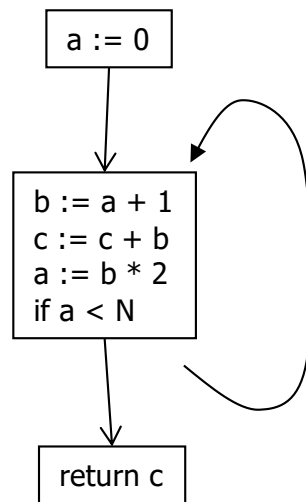  - Not a "real" instruction – just a form of bookkeeping needed for SSA

**Original.**

$b := M[x]$
$a := 0$

↓

if $b < 4$

$a := b$

$c := a + b$

**SSA Form**

$b_1 := M[x]$
$a_1 := 0$

↓

if $b_1 < 4$

$a_2 := b_1$

$a_3 := \Phi(a_1, a_2)$
$c_1 := a_3 + b_1$

# 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
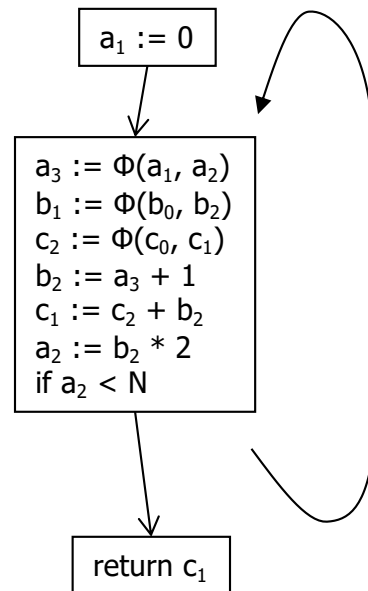
**Original**

```
b := M[x]
a := 0
```

```
if b < 4
```

```
a := b
```

```
c := a + b
```

**SSA Form**

```
b₁ := M[x]
a₁ := 0
```
$b_1 := M[x]$
$a_1 := 0$

```
if b₁ < 4
```
$if\ b_1 < 4$

```
a₂ := b₁
```
$a_2 := b_1$

```
a₃ := Φ(a₁, a₂)
c₁ := a₃ + b₁
```
$a_3 := \Phi(a_1, a_2)$
$c_1 := a_3 + b_1$

# Example With a Loop

### Original

a := 0

b := a + 1
c := c + b
a := b * 2
if a < N

return c

### SSA

$a_1 := 0$

$a_3 := \Phi(a_1, a_2)$
$b_1 := \Phi(b_0, b_2)$
$c_2 := \Phi(c_0, c_1)$
$b_2 := a_3 + 1$
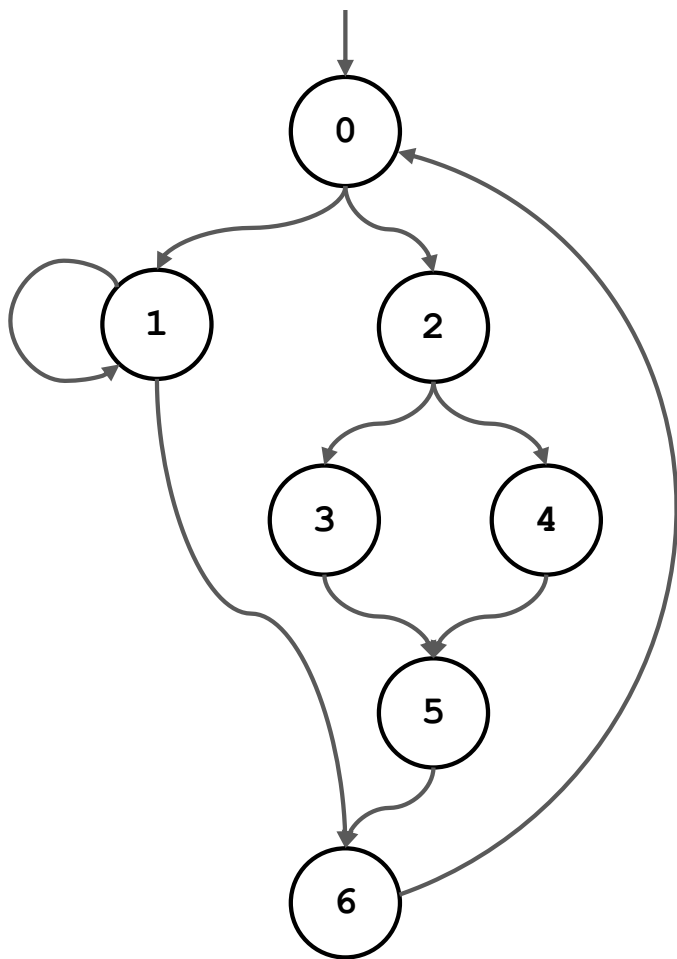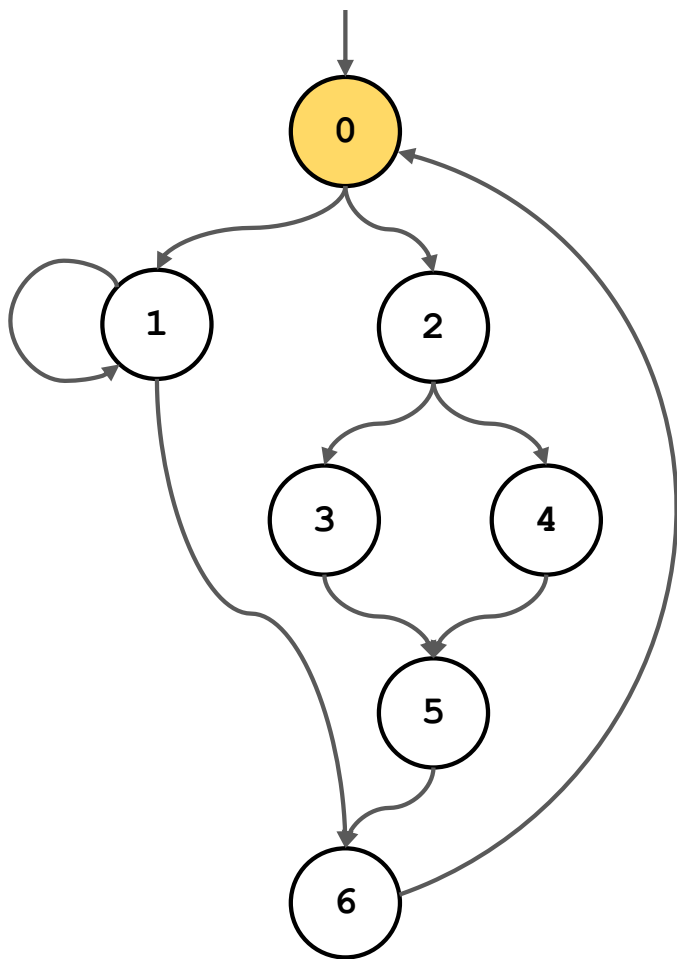$c_1 := c_2 + b_2$
$a_2 := b_2 * 2$
if $a_2 < N$

return $c_1$

Notes:
- Loop-back edges are also merge points, so require $\Phi$-functions
- $a_0, b_0, c_0$ are initial values of a, b, c on entry to initial block
- $b_1$ is dead – can delete later
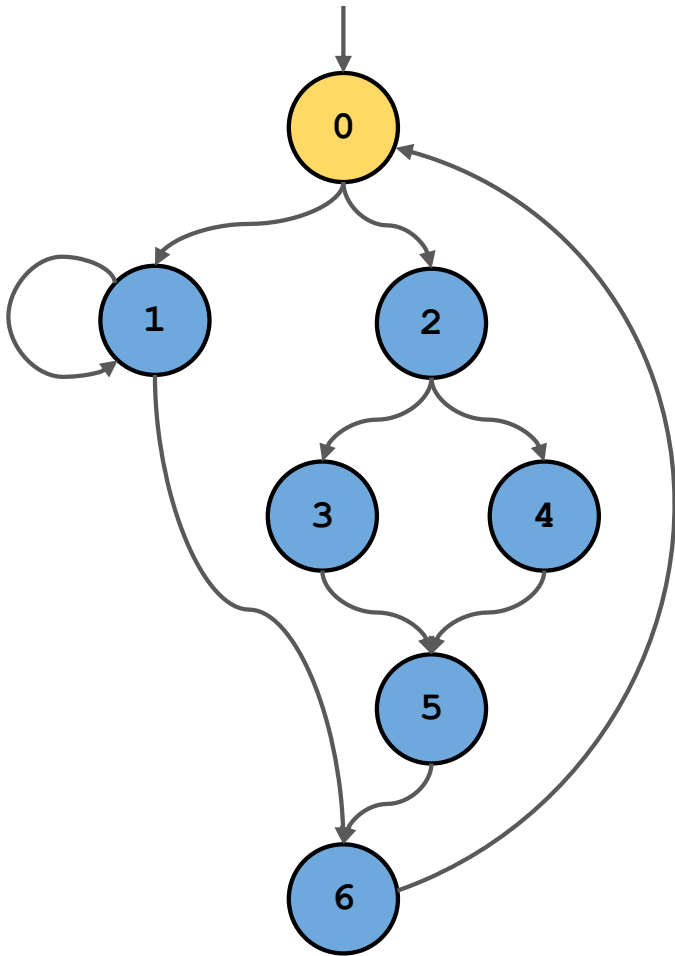- c is live on entry – either input parameter or uninitialized

# 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 **X** *dominates* a node **Y** iff every path from the entry point of the control flow graph to **Y** includes **X**.
A node **X** *strictly dominates* a node **Y** iff **X** dominates **Y** and **X** ≠ **Y**

Need to go through 0 to get through 1, 2, 3, 4, 5, 6 and 0 cannot strictly dominate itself
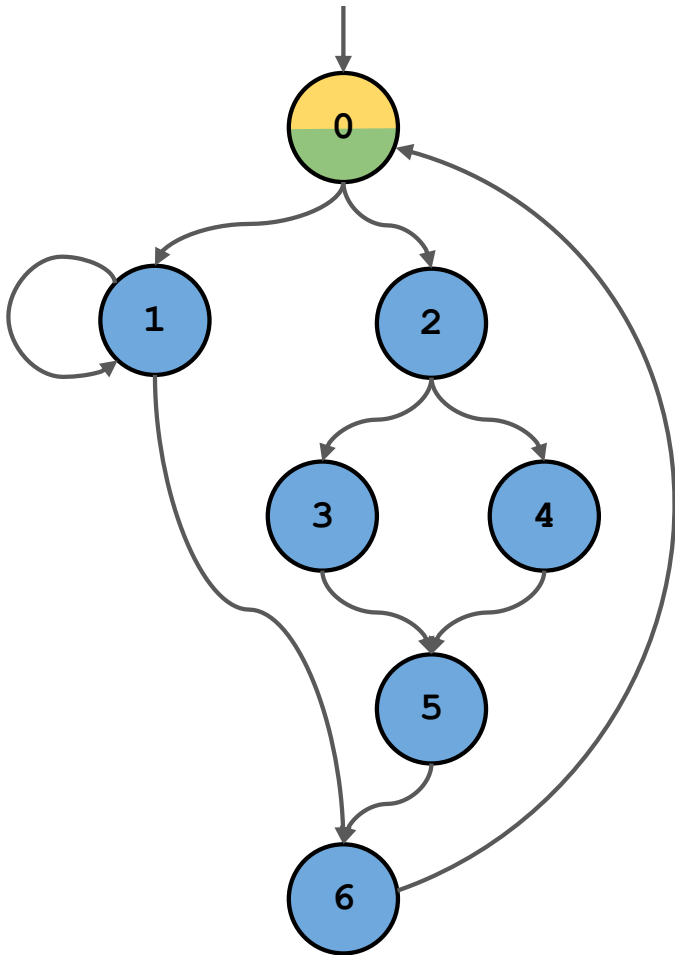
| NODE | STRICTLY DOMINATES | DOMINANCE FRONTIER |
|------|--------------------|--------------------|
| 0 | 1, 2, 3, 4, 5, 6 | 0 |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |

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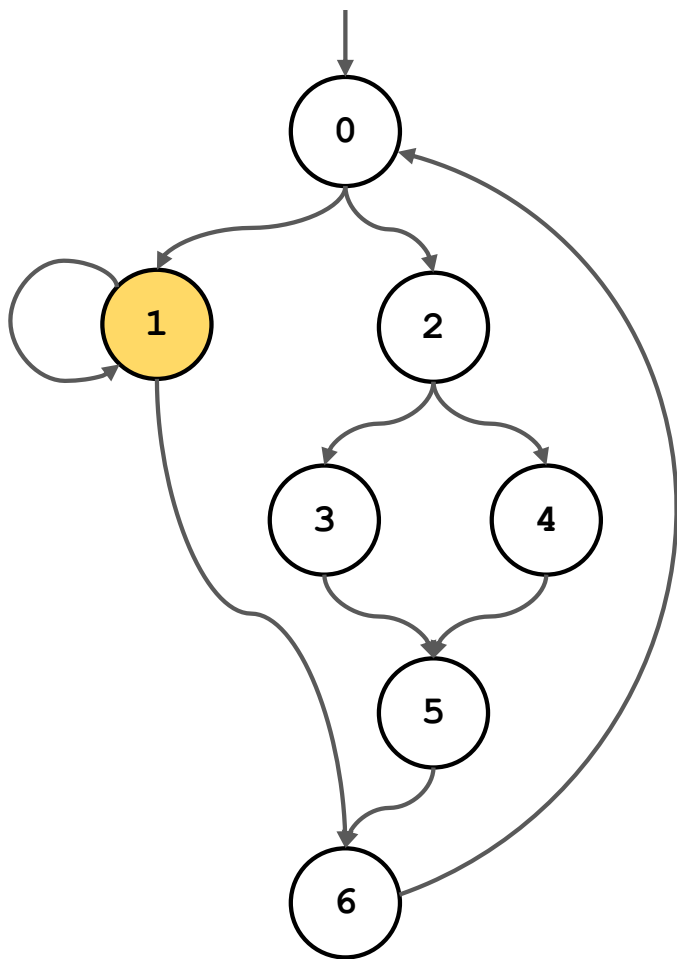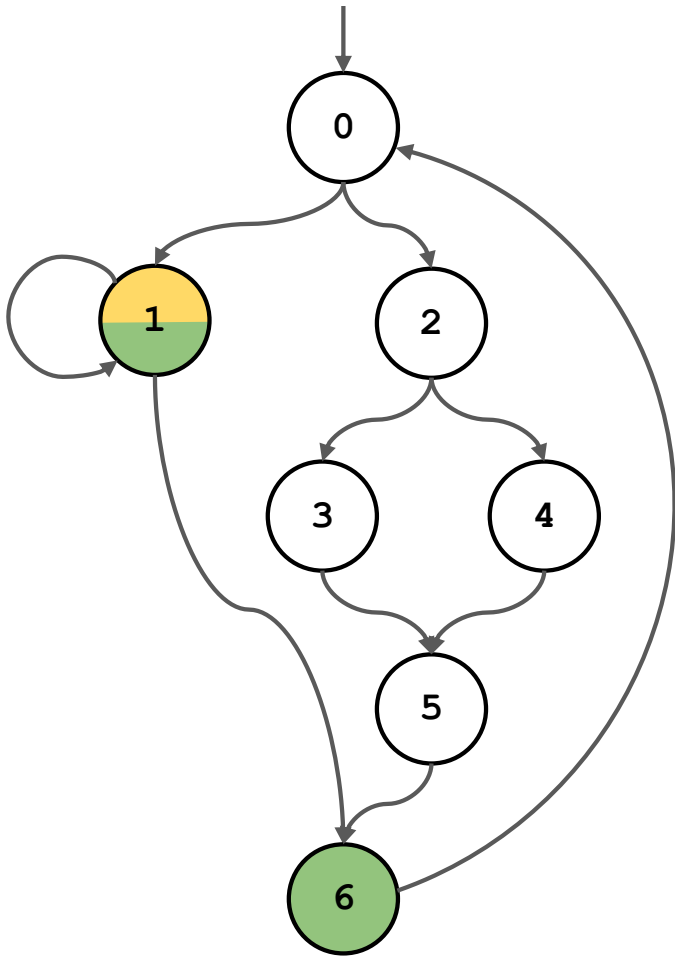
| NODE | STRICTLY DOMINATES | DOMINANCE FRONTIER |
|------|--------------------|--------------------|
| 0 | 1, 2, 3, 4, 5, 6 | 0 |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |

| NODE | STRICTLY DOMINATES | DOMINANCE FRONTIER |
|------|--------------------|--------------------|
| 0 | 1, 2, 3, 4, 5, 6 | 0 |
| 1 | Ø | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
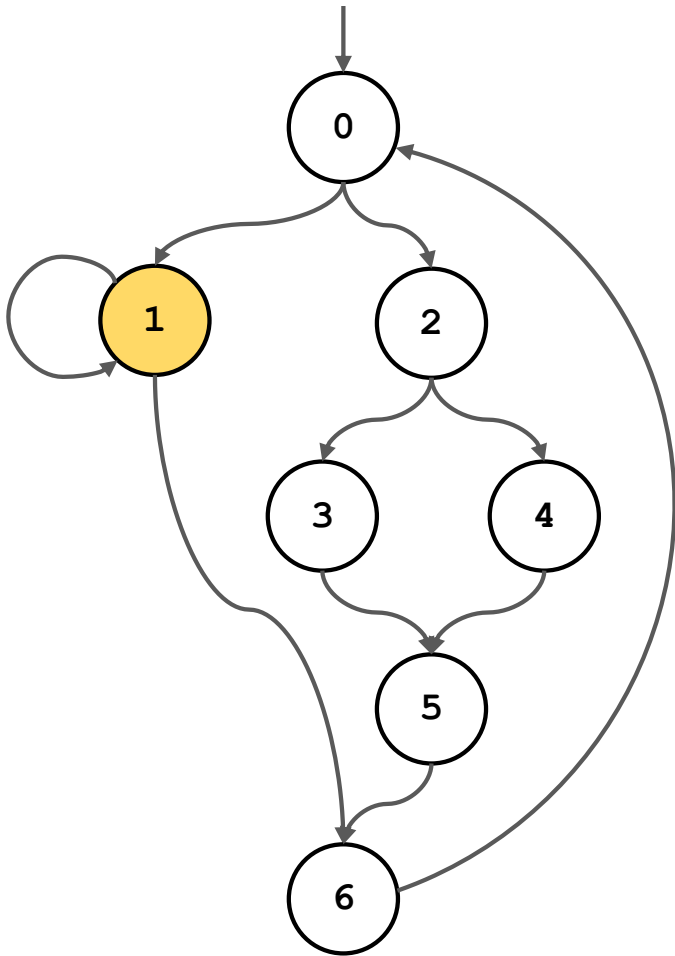
A node **X** *dominates* a node **Y** iff every path from the entry point of the control flow graph to **Y** includes **X** .

A node **X** *strictly dominates* a node **Y** iff **X** dominates **Y** and **X** ≠ **Y**

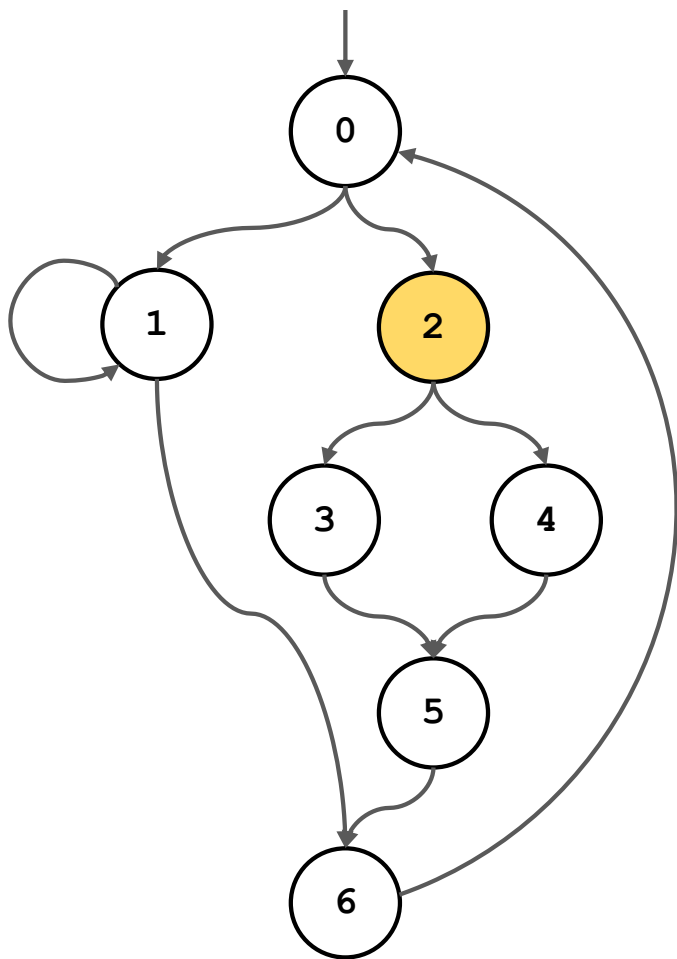1 does not dominate 6 because there is a path from 5 that doesn't include 1. 1 does not strictly dominate itself

| NODE | STRICTLY DOMINATES | DOMINANCE FRONTIER |
|------|--------------------|--------------------|
| 0 | 1, 2, 3, 4, 5, 6 | 0 |
| 1 | Ø | 1, 6 |
| 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** .

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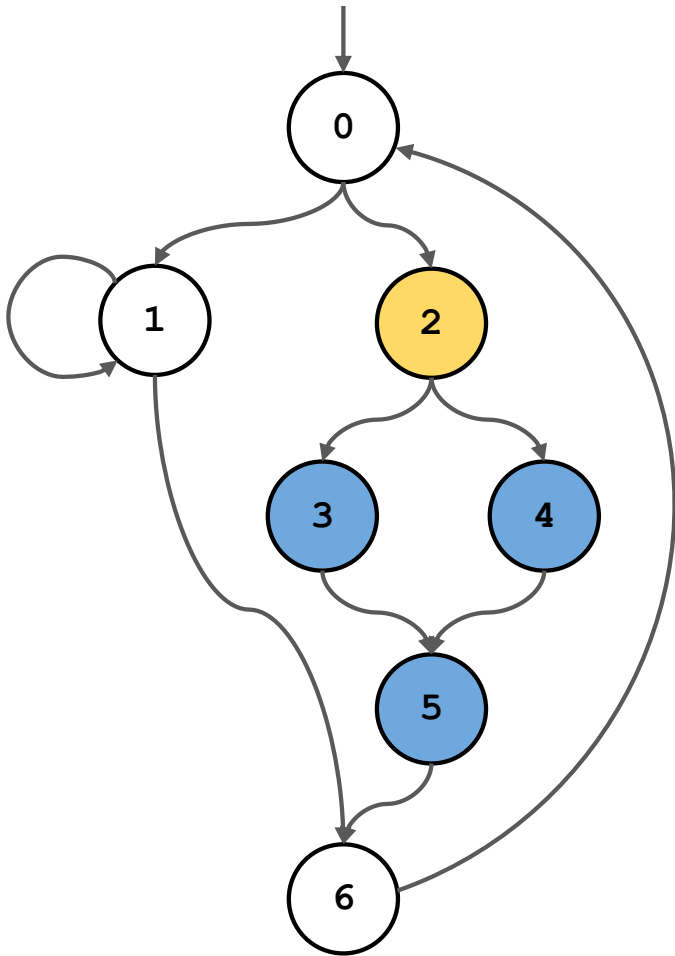
| NODE | STRICTLY DOMINATES | DOMINANCE FRONTIER |
|------|--------------------|--------------------|
| 0 | 1, 2, 3, 4, 5, 6 | 0 |
| 1 | ø | 1, 6 |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |

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

A node **X** *dominates* a node **Y** iff every path from the entry point of the control flow graph to **Y** includes **X**.

A node **X** *strictly dominates* a node **Y** iff **X** dominates **Y** and **X** ≠ **Y**

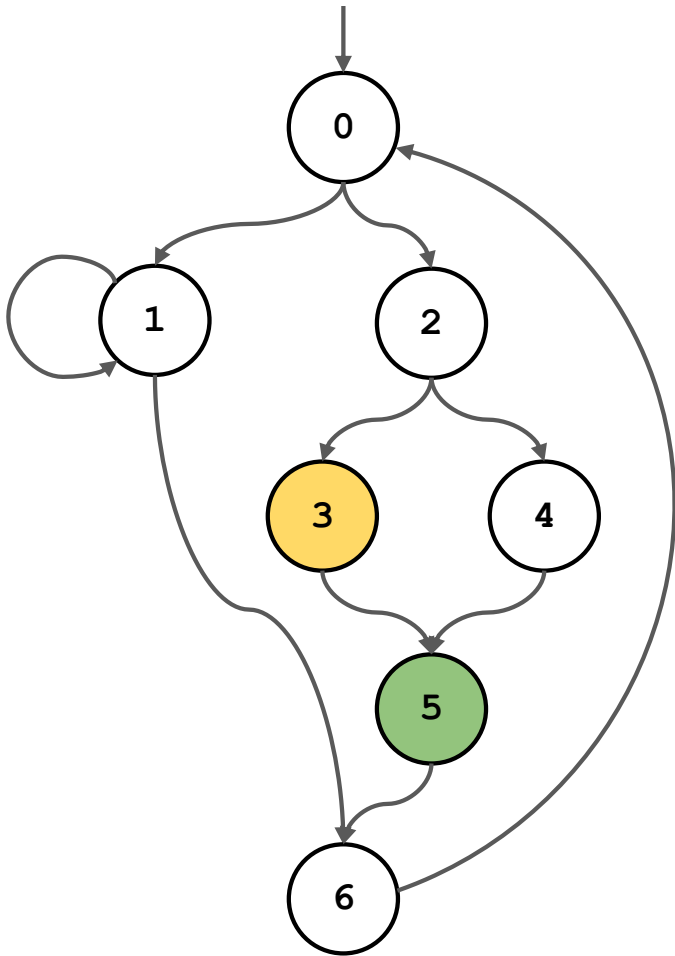Need to go through 2 to get through 3, 4, 5 and 2 cannot strictly dominate itself
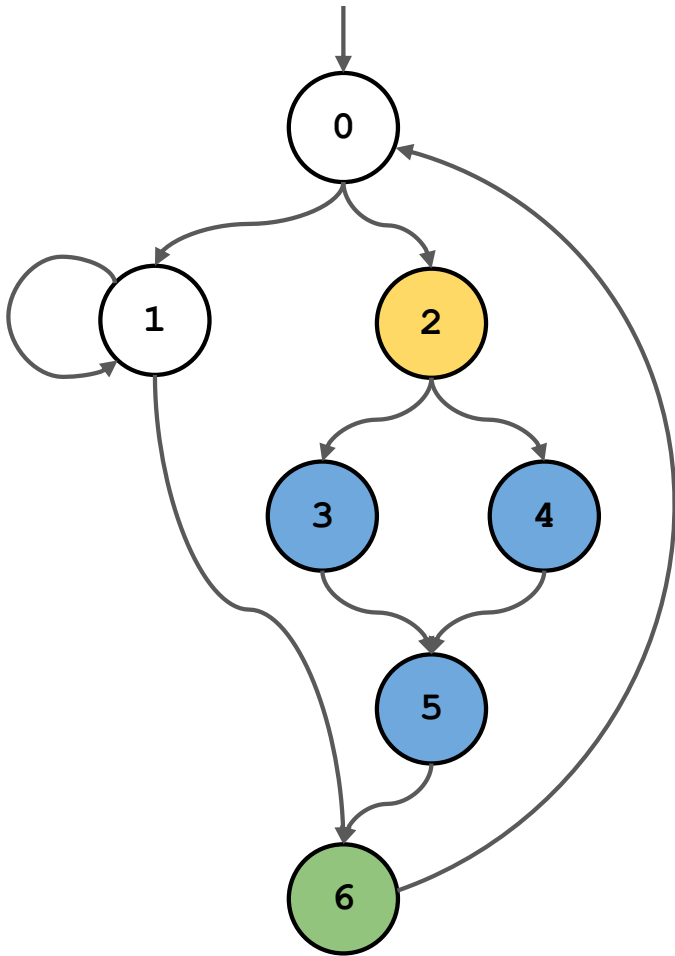
| NODE | STRICTLY DOMINATES | DOMINANCE FRONTIER |
|---|---|---|
| 0 | 1, 2, 3, 4, 5, 6 | 0 |
| 1 | ∅ | 1, 6 |
| 2 | 3, 4, 5 | 6 |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |

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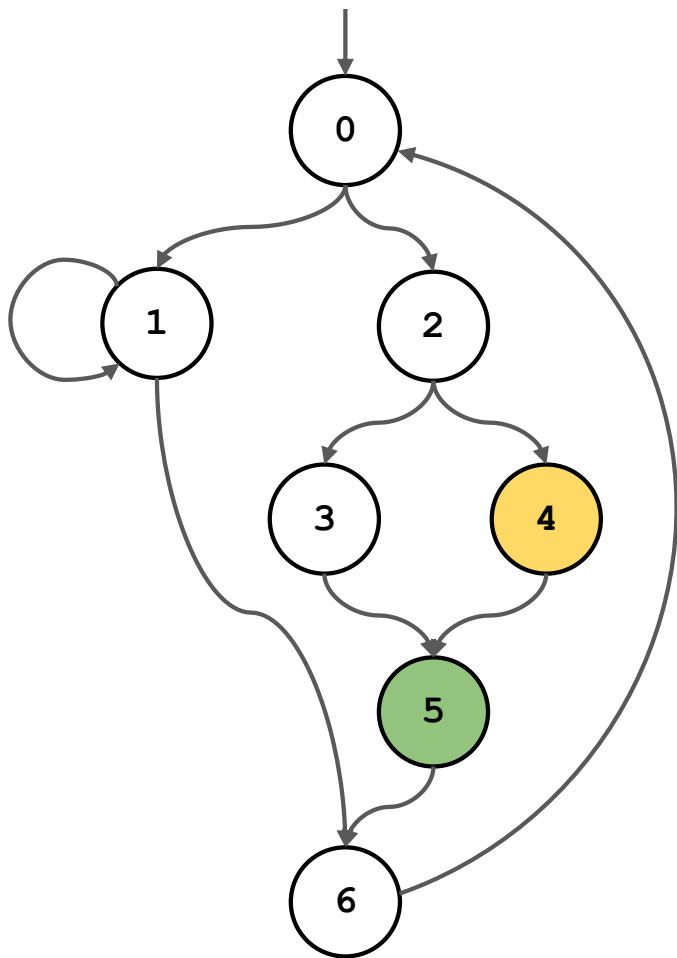
X = 2, Y = 6, 2 dominates 5, 5 is an immediate predecessor of 6, 2 does not strictly dominate 6

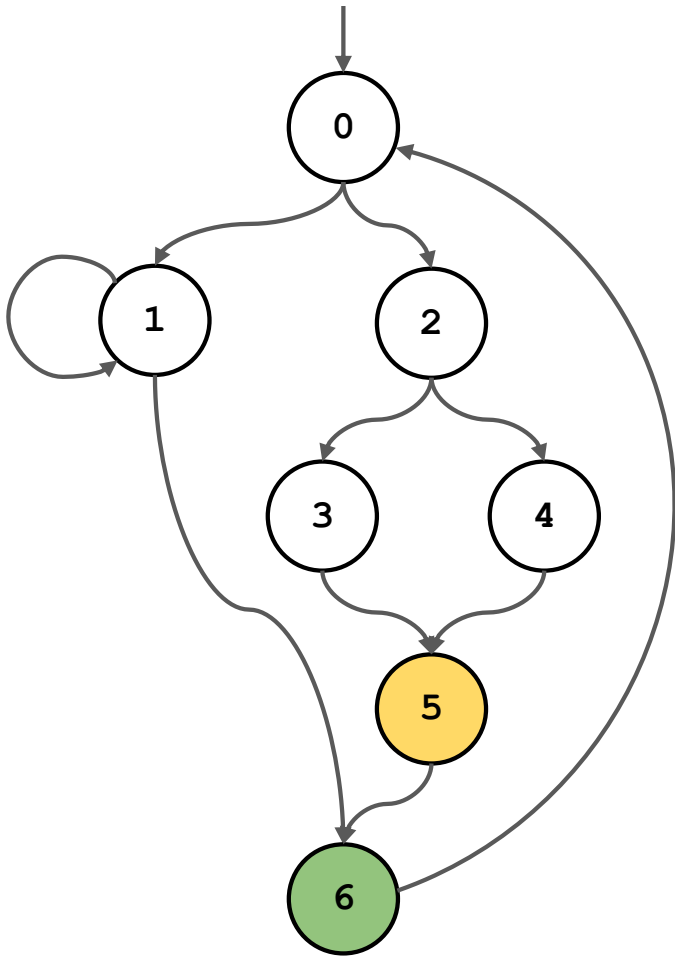| NODE | STRICTLY DOMINATES | DOMINANCE FRONTIER |
|---|---|---|
| 0 | 1, 2, 3, 4, 5, 6 | 0 |
| 1 | ∅ | 1, 6 |
| 2 | 3, 4, 5 | 6 |
| 3 | ∅ | 5 |
| 4 | | |
| 5 | | |
| 6 | | |

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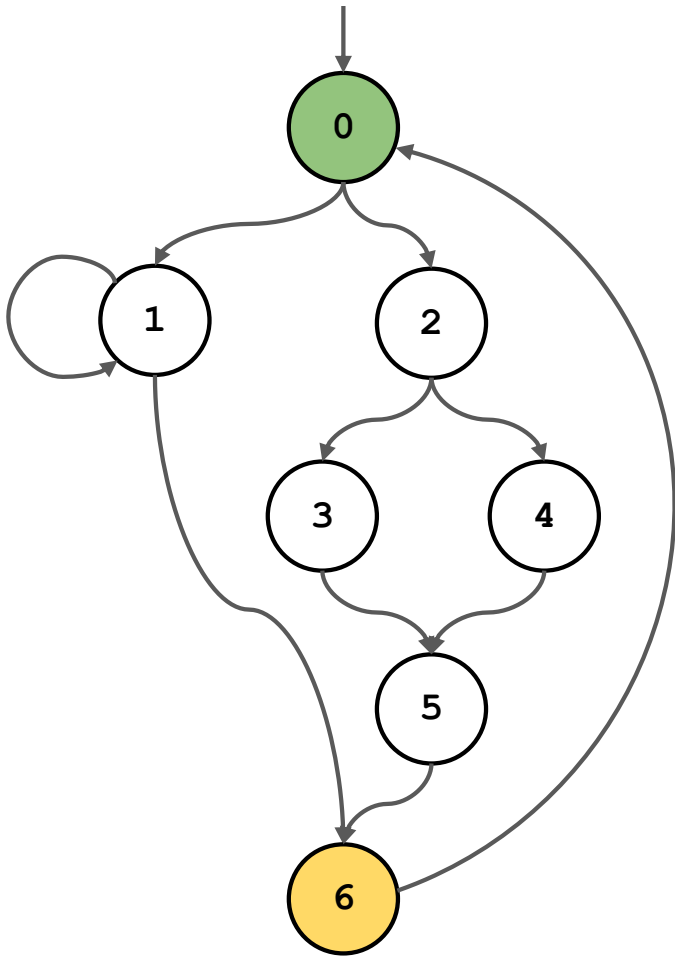
| NODE | STRICTLY DOMINATES | DOMINANCE FRONTIER |
|------|--------------------|--------------------|
| 0 | 1, 2, 3, 4, 5, 6 | 0 |
| 1 | ∅ | 1, 6 |
| 2 | 3, 4, 5 | 6 |
| 3 | ∅ | 5 |
| 4 | ∅ | 5 |
| 5 | ∅ | 6 |
| 6 | ∅ | 0 |

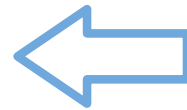6 does not strictly dominate 0 (path through 0) and therefore does not strictly dominate anything else

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

# Problem 2(b)

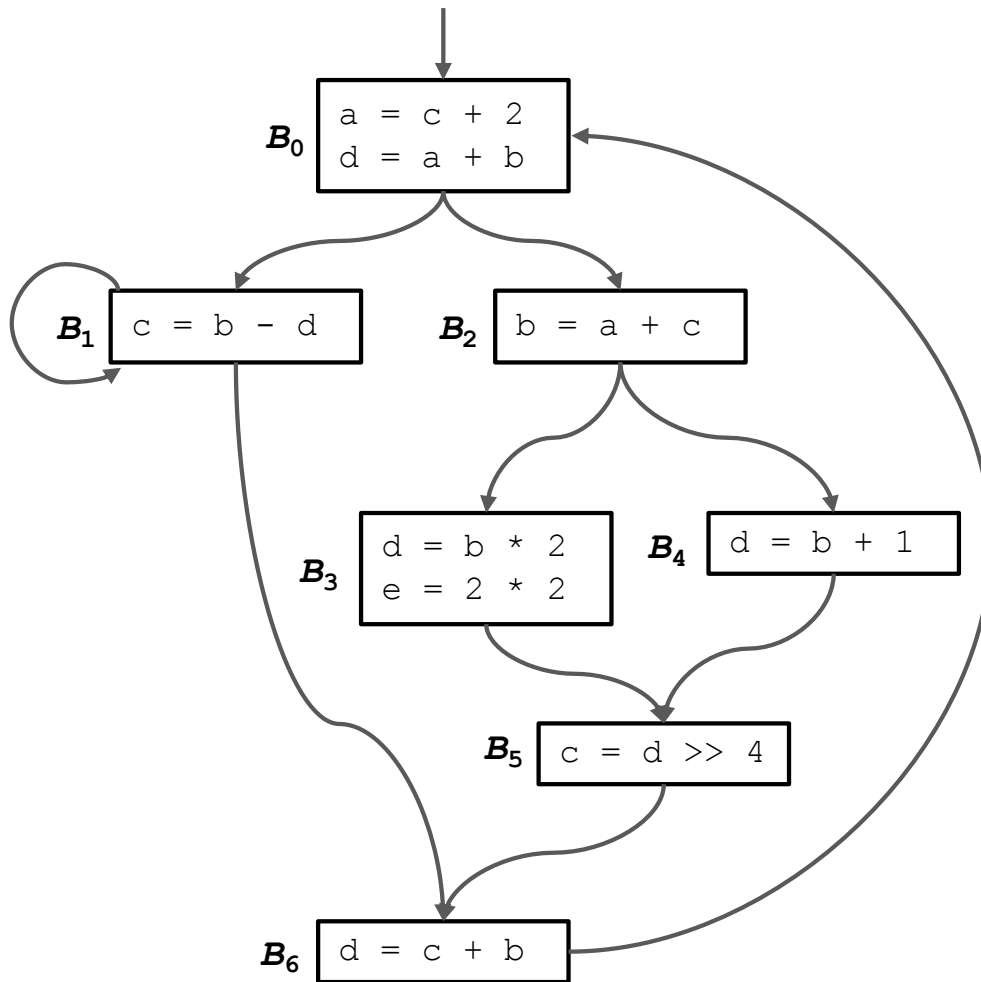# Converting to SSA

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 | Ø | 1, 6 |
| 2 | 3, 4, 5 | 6 |
| 3 | Ø | 5 |
| 4 | Ø | 5 |
| 5 | Ø | 6 |
| 6 | Ø | 0 |

Control flow graph:

$B_0$: a = c + 2; d = a + b

$B_1$: c = b - d

$B_2$: b = a + c

$B_3$: d = b * 2; e = 2 * 2

$B_4$: d = b + 1

$B_5$: c = d >> 4

$B_6$: d = c + b
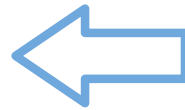
# Converting to SSA

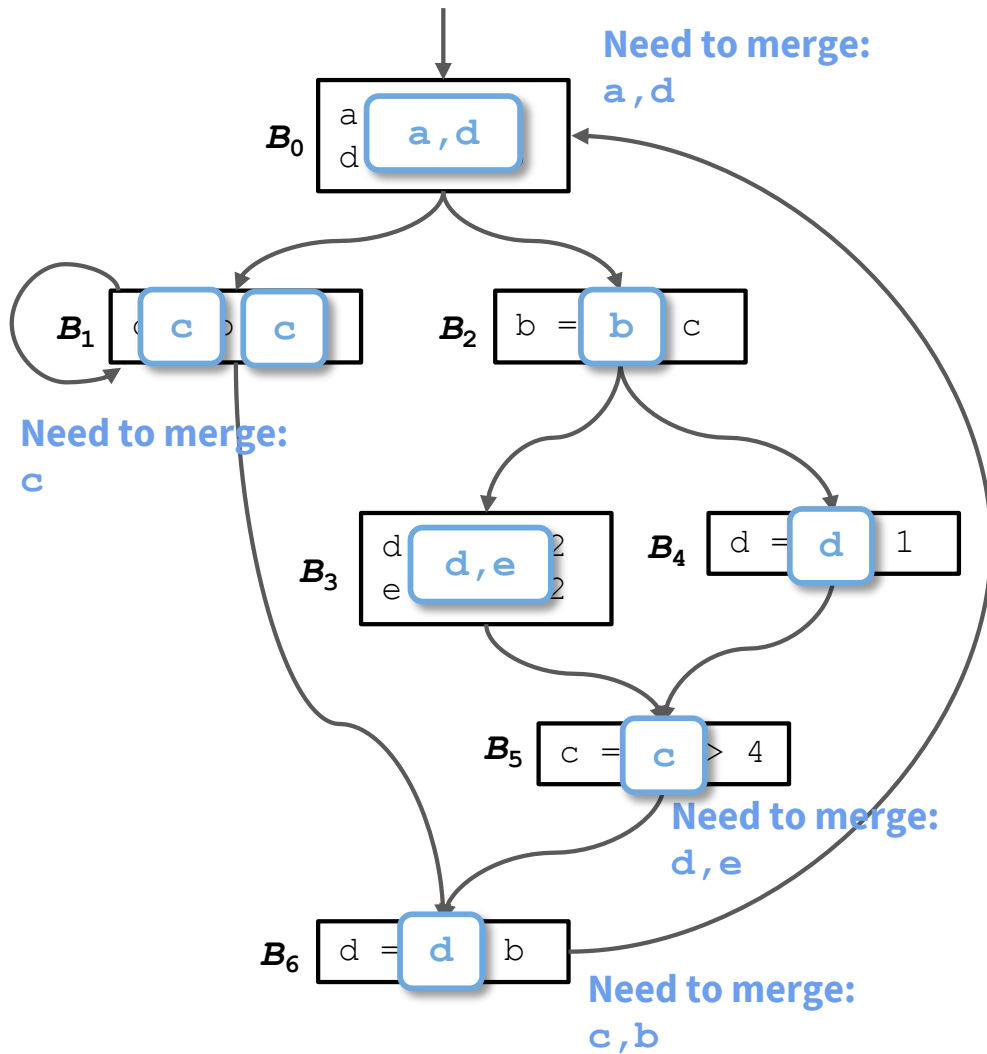**1**    Compute the dominance frontier of each node    ✓

**2**    **Determine which variables need merging in each node**   ⬅ **We will compute using the dominance frontiers**
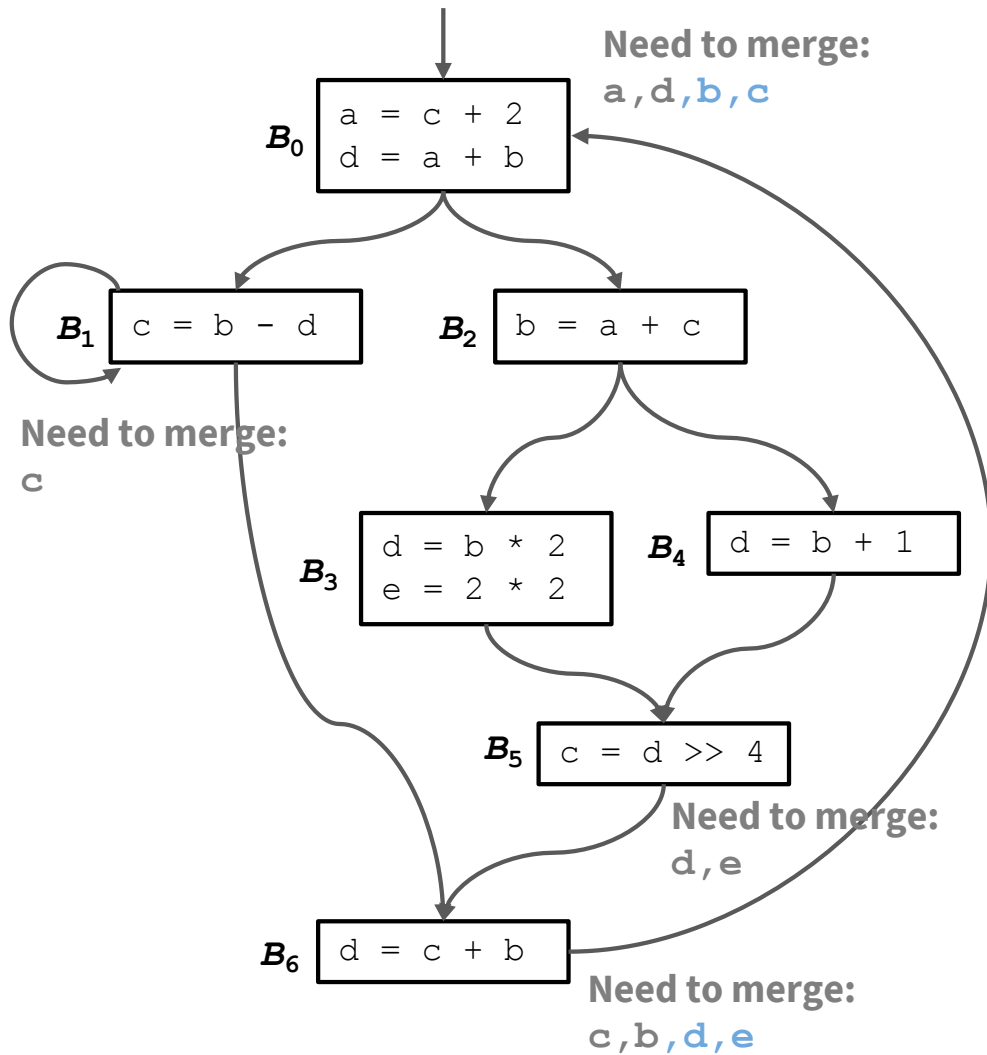
**3**    Assign numbers to definitions and add phi functions

**Step 2**: Determine Necessary Merges

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

Need to merge:
a,d

Need to merge:
c

Need to merge:
d,e

Need to merge:
c,b

| 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.

Diagram labels:

$B_0$: a = c + 2; d = a + b — Need to merge: a,d,b,c

$B_1$: c = b - d — Need to merge: c

$B_2$: b = a + c

$B_3$: d = b * 2; e = 2 * 2

$B_4$: d = b + 1

$B_5$: c = d >> 4 — Need to merge: d,e

$B_6$: d = c + b — Need to merge: c,b,d,e

| NODE | | DOMINANCE FRONTIER |
|------|------|------|
| 0 | | 0 |
| 1 | | 1, 6 |
| 2 | | 6 |
| 3 | | 5 |
| 4 | | 5 |
| 5 | d,e | 6 |
| 6 | b,c | 0 |
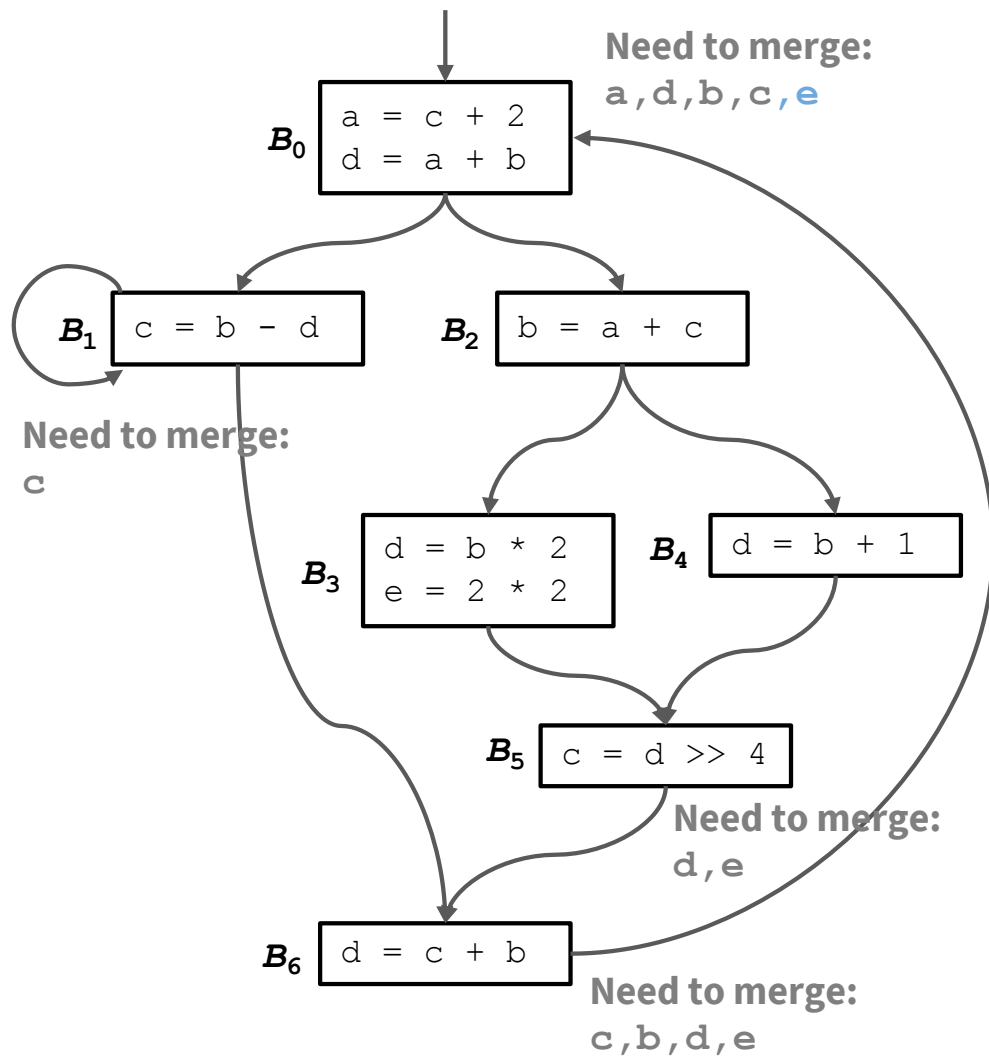
**Step 2**: Determine Necessary Merges

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

Diagram labels:

$B_0$: 
```
a = c + 2
d = a + b
```
Need to merge: a,d,b,c,e

$B_1$: `c = b - d`
Need to merge: c

$B_2$: `b = a + c`

$B_3$:
```
d = b * 2
e = 2 * 2
```

$B_4$: `d = b + 1`

$B_5$: `c = d >> 4`
Need to merge: d,e

$B_6$: `d = c + b`
Need to merge: c,b,d,e

| NODE | | DOMINANCE FRONTIER |
|------|---|--------------------|
| 0 | | 0 |
| 1 | | 1, 6 |
| 2 | | 6 |
| 3 | | 5 |
| 4 | | 5 |
| 5 | | 6 |
| 6 | d,e | 0 |

# Converting to SSA

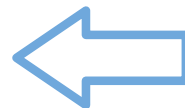1. Compute the dominance frontier of each node ✔

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.

$B_0$

$$a = c + 2$$
$$d = a + b$$

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

$\Longrightarrow$

$B_0$

$$a_1 = \Phi(a_0, a_2)$$
$$b_1 = \Phi(b_0, b_3)$$
$$c_1 = \Phi(c_0, c_5)$$
$$d_1 = \Phi(d_0, d_7)$$
$$e_1 = \Phi(e_0, e_4)$$
$$a_2 = c_1 + 2$$
$$d_2 = a_2 + b_1$$

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

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

$B_1$ | c = b - d

Need to merge:

c

$B_1$ | $c_2 = \Phi(c_1, c_3)$
$c_3 = b_1 - d_2$

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

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

$B_2$ | $b = a + c$

$\Longrightarrow$

$B_2$ | $b_2 = a_2 + c_1$

Nothing to merge

**Step 3**: Assign Definition Numbers

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

$B_3$
```
d = b * 2
e = 2 * 2
```

Nothing to merge

$\Rightarrow$

$B_3$
```
d₃ = b₂ * 2
e₂ = 2 * 2
```

$B_3$
$$d_3 = b_2 * 2$$
$$e_2 = 2 * 2$$

**Step 3**: Assign Definition Numbers

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

$B_4$ | $d = b + 1$



$B_4$ | $d_4 = b_2 + 1$

Nothing to merge

**Step 3**: Assign Definition Numbers

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

$B_5$ | $c = d >> 4$

Need to merge:
$d, e$

$B_5$ |
$d_5 = \Phi(d_3, d_4)$
$e_3 = \Phi(e_1, e_2)$
$c_4 = d_5 >> 4$

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

$B_6$ | $d = c + b$

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

$\Longrightarrow$

$B_6$

$$b_3 = \Phi(b_1, b_2)$$
$$c_5 = \Phi(c_3, c_4)$$
$$d_6 = \Phi(d_2, d_5)$$
$$e_4 = \Phi(e_1, e_3)$$
$$d_7 = c_5 + b_3$$

**Solution**

$B_0$
$$a_1 = \Phi(a_0, a_2)$$
$$b_1 = \Phi(b_0, b_3)$$
$$c_1 = \Phi(c_0, c_5)$$
$$d_1 = \Phi(d_0, d_7)$$
$$e_1 = \Phi(e_0, e_4)$$
$$a_2 = c_1 + 2$$
$$d_2 = a_2 + b_1$$

$B_1$
$$c_2 = \Phi(c_1, c_3)$$
$$c_3 = b_1 - d_2$$

$B_2$
$$b_2 = a_2 + c_1$$

$B_3$
$$d_3 = b_2 * 2$$
$$e_2 = 2 * 2$$

$B_4$
$$d_4 = b_2 + 1$$

$B_5$
$$d_5 = \Phi(d_3, d_4)$$
$$e_3 = \Phi(e_1, e_2)$$
$$c_4 = d_5 >> 4$$

$B_6$
$$b_3 = \Phi(b_1, b_2)$$
$$c_5 = \Phi(c_3, c_4)$$
$$d_6 = \Phi(d_2, d_5)$$
$$e_4 = \Phi(e_1, e_3)$$
$$d_7 = c_5 + b_3$$