EECS 583 – Class 7
Static Single Assignment Form

University of Michigan

September 26, 2018

Announcements & Reading Material

- HW2 is out – Available on course webpage
  - See piazza for guide/hints
  - Benchmarks will be posted soon, but you can start w/o them!
- Today's class
  - "Practical Improvements to the Construction and Destruction of Static Single Assignment Form," P. Briggs, K. Cooper, T. Harvey, and L. Simpson, *Software--Practice and Experience*, 28(8), July 1998, pp. 859-891.
- Next class – Optimization
  - *Compilers: Principles, Techniques, and Tools,*
    A. Aho, R. Sethi, and J. Ullman, Addison-Wesley, 1988,
    9.9, 10.2, 10.3, 10.7 Edition 1; 8.5, 8.7, 9.1, 9.4, 9.5 Edition 2

Homework 2 – Frequent Path LICM

```
j = 99;
for (i=0; i<100; i++) {
    B[i] = A[j] * 23 + i;
    if (i % 32 == 0)
        j = i;  /* infrequent */
}
```

Basic

```
j = 99;
t = j;
for (i=0; i<100; i++) {
    B[i] = A[t] * 23 + i;
    if (i % 32 == 0)
        j = i;  /* infrequent */
}
```

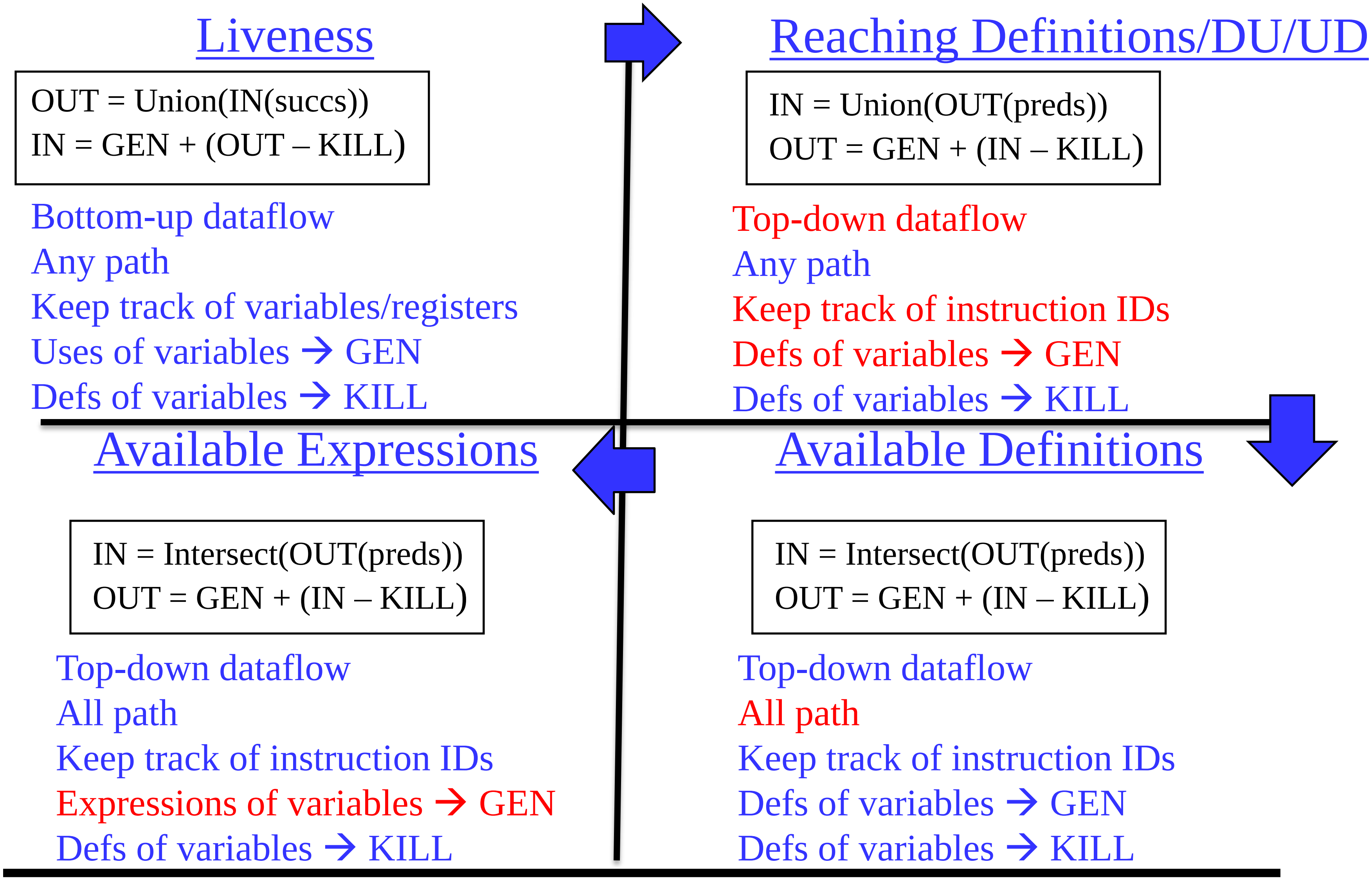
Insert
repair code

```
j = 99;
t = j;
for (i=0; i<100; i++) {
    B[i] = A[t] * 23 + i;
    if (i % 32 == 0) {
        j = i;  /* infrequent */
        t = j;  /* fixup */
    }
}
```

Bonus,
Hoist invariant uses,
update repair code

```
j = 99;
t = A[j] * 23;
for (i=0; i<100; i++) {
    B[i] = t + i;
    if (i % 32 == 0) {
        j = i;  /* infrequent */
        t = A[j] * 23;  /* fixup */
    }
}
```

Dataflow Analyses in 1 Slide



Liveness

OUT = Union(IN(succs))
IN = GEN + (OUT – KILL)

Bottom-up dataflow
Any path
Keep track of variables/registers
Uses of variables → GEN
Defs of variables → KILL

Reaching Definitions/DU/UD

IN = Union(OUT(preds))
OUT = GEN + (IN – KILL)

Top-down dataflow
Any path
Keep track of instruction IDs
Defs of variables → GEN
Defs of variables → KILL

Available Expressions

IN = Intersect(OUT(preds))
OUT = GEN + (IN – KILL)

Top-down dataflow
All path
Keep track of instruction IDs
Expressions of variables → GEN
Defs of variables → KILL

Available Definitions

IN = Intersect(OUT(preds))
OUT = GEN + (IN – KILL)

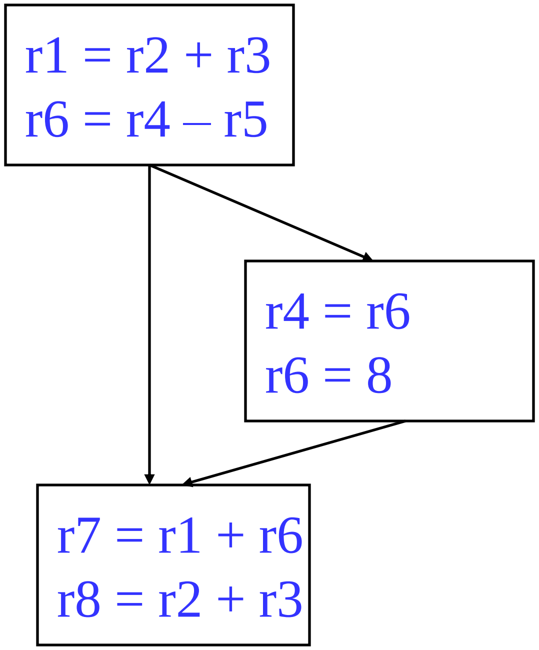
Top-down dataflow
All path
Keep track of instruction IDs
Defs of variables → GEN
Defs of variables → KILL

Some Things to Think About

- Liveness and rdefs are basically the same thing
  - All dataflow is basically the same with a few parameters
    - Meaning of gen/kill – src vs dest, variable vs operation
    - Backward / Forward
    - All paths / some paths (must/may)
    - What other dataflow analysis problems can be formulated?
- Dataflow can be slow
  - How to implement it efficiently?
    - Forward analysis – DFS order
    - Backward analysis – PostDFS order
  - How to represent the info?
- Predicates
  - Throw a monkey wrench into this stuff
  - So, how are predicates handled?

Static Single Assignment (SSA) Form

- Difficulty with optimization
  - Multiple definitions of the same register
  - Which definition reaches
  - Is expression available?



r1 = r2 + r3
r6 = r4 – r5

r4 = r6
r6 = 8

r7 = r1 + r6
r8 = r2 + r3

- Static single assignment
  - Each assignment to a variable is given a unique name
  - All of the uses reached by that assignment are renamed
  - DU chains become obvious based on the register name!

Converting to SSA Form

- Trivial for straight line code

```
x = -1
y = x
x = 5
z = x
```

→

```
x0 = -1
y = x0
x1 = 5
z = x1
```

- More complex with control flow – Must use Phi nodes

```
if ( … )
    x = -1
else
    x = 5
y = x
```

→

```
if ( … )
    x0 = -1
else
    x1 = 5
x2 = Phi(x0,x1)
y = x2
```

Converting to SSA Form (2)

- What about loops?
  - No problem!, use Phi nodes again

```
i = 0
do {
    i = i + 1
}
while (i < 50)
```

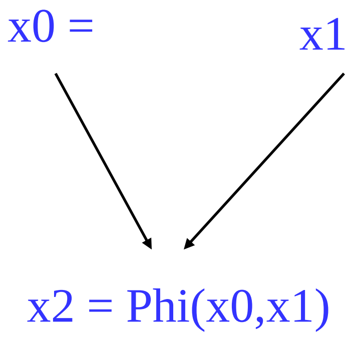
```
i0 = 0
do {
    i1 = Phi(i0, i2)
    i2 = i1 + 1
}
while (i2 < 50)
```

SSA Plusses and Minuses

- Advantages of SSA
  - Explicit DU chains – Trivial to figure out what defs reach a use
    - Each use has exactly 1 definition!!!
  - Explicit merging of values
  - Makes optimizations easier
- Disadvantages
  - When transform the code, must either recompute (slow) or incrementally update (tedious)

Phi Nodes (aka Phi Functions)

- Special kind of copy that selects one of its inputs
- Choice of input is governed by the CFG edge along which control flow reached the Phi node



x0 =

x1 =

x2 = Phi(x0,x1)

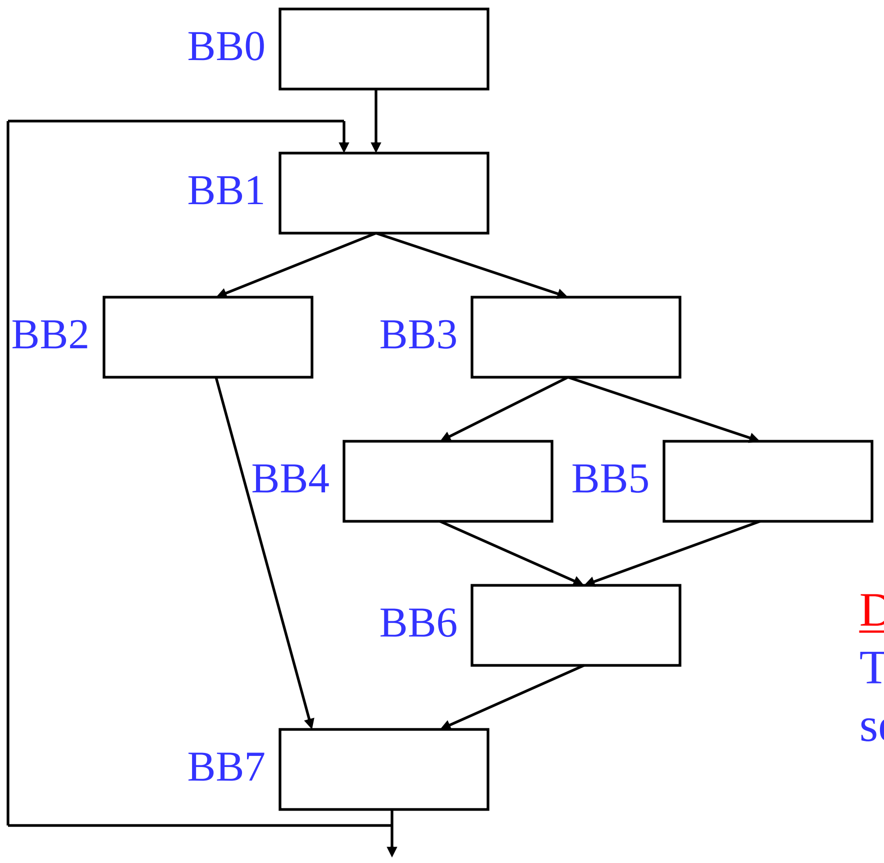
- Phi nodes are required when 2 non-null paths X→Z and Y→Z converge at node Z, and nodes X and Y contain assignments to V

SSA Construction

- High-level algorithm
    1. Insert Phi nodes
    2. Rename variables
- A dumb algorithm
    - Insert Phi functions at every join for every variable
    - Solve reaching definitions
    - Rename each use to the def that reaches it (will be unique)
- Problems with the dumb algorithm
    - Too many Phi functions (precision)
    - Too many Phi functions (space)
    - Too many Phi functions (time)

Need Better Phi Node Insertion Algorithm

- A definition at n forces a Phi node at m iff n not in DOM(m), but n in DOM(p) for some predecessors p of m



def in BB4 forces Phi in BB6
def in BB6 forces Phi in BB7
def in BB7 forces Phi in BB1

Phi is placed in the block that
is just outside the dominated region
of the definition BB

Dominance frontier

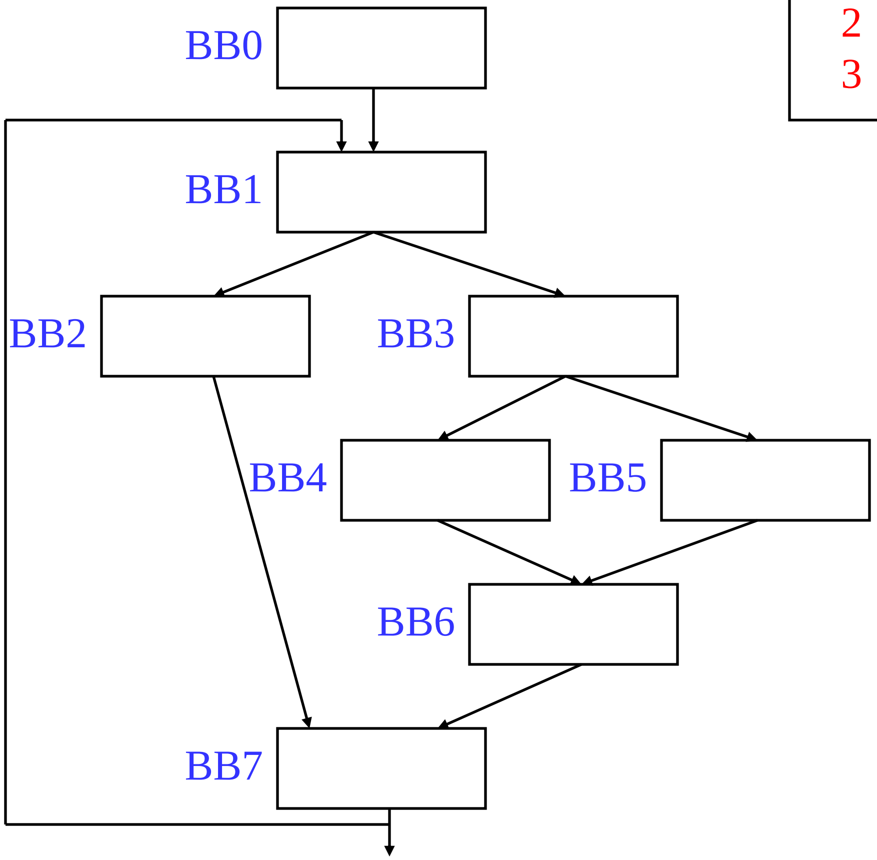
The dominance frontier of node X is the set of nodes Y such that

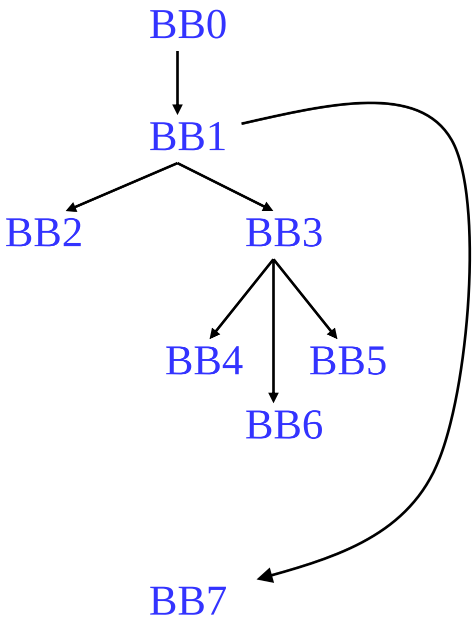
  * X dominates a predecessor of Y, but
  * X does not strictly dominate Y

Recall: Dominator Tree

First BB is the root node, each node dominates all of its descendants

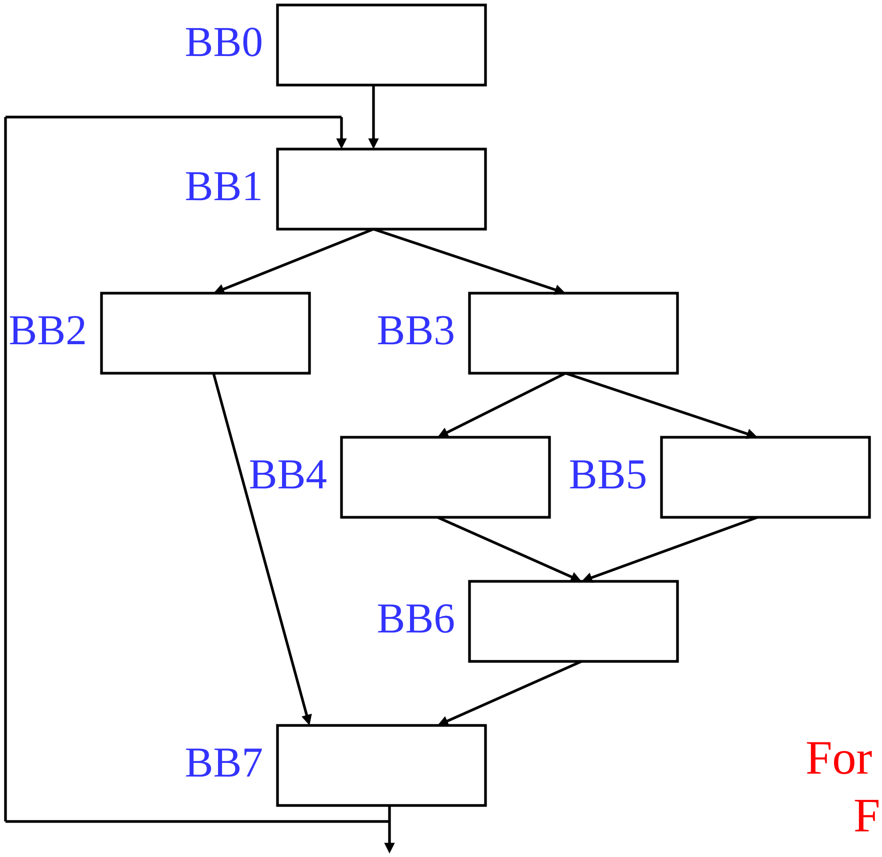
BB	DOM	BB	DOM
0	0	4	0,1,3,4
1	0,1	5	0,1,3,5
2	0,1,2	6	0,1,3,6
3	0,1,3	7	0,1,7

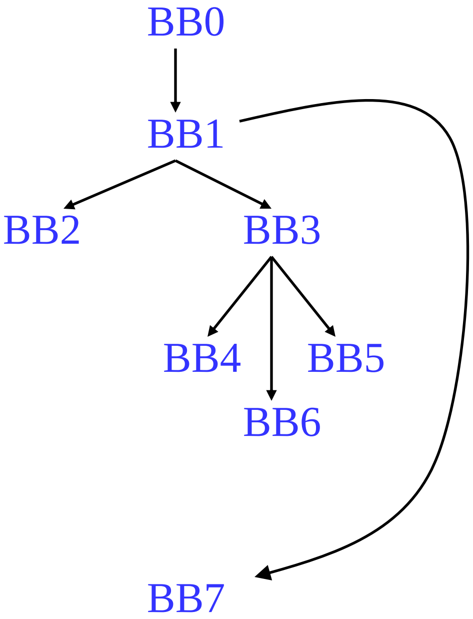




Dom tree

Computing Dominance Frontiers



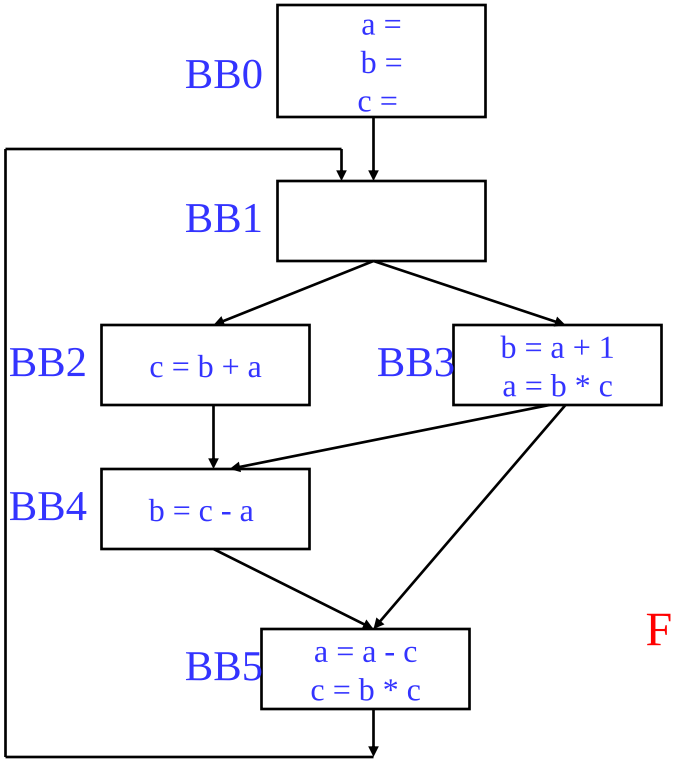


BB	DF
0	
1	
2	
3	
4	
5	
6	
7	

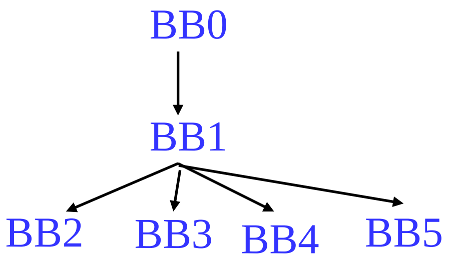
```
For each join point X in the CFG
    For each predecessor, Y, of X in the CFG
        Run up to the IDOM(X) in the dominator tree,
        adding X to DF(N) for each N between Y and
        IDOM(X) (or X, whichever is encountered first)
```

Class Problem

Compute dominance frontiers for each BB



Dominator Tree



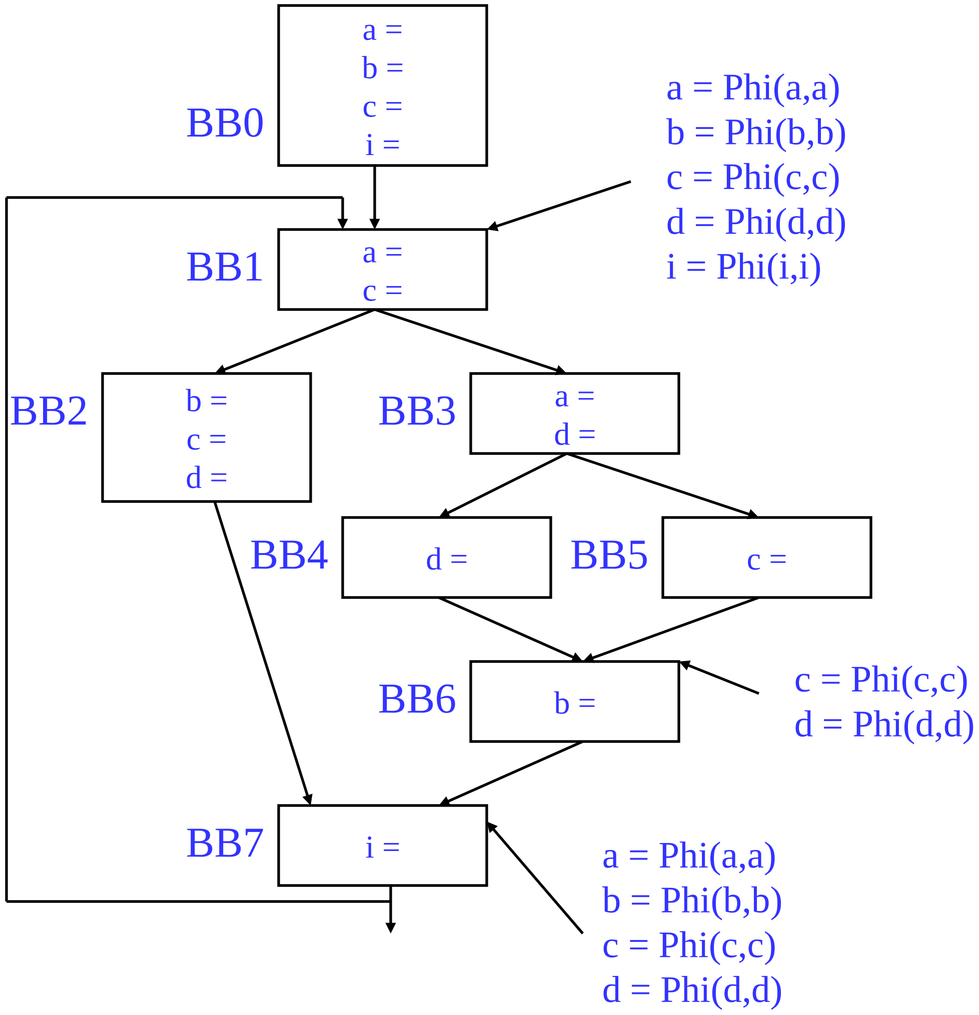
```
For each join point X in the CFG
    For each predecessor, Y, of X in the CFG
        Run up to the IDOM(X) in the dominator tree,
        adding X to DF(N) for each N between Y and
        IDOM(X) (or X, whichever is encountered first)
```

SSA Step 1 - Phi Node Insertion

- Compute dominance frontiers
- Find global names (aka virtual registers)
  - Global if name live on entry to some block
  - For each name, build a list of blocks that define it
- Insert Phi nodes
  - For each global name n
    - For each BB b in which n is defined
      - For each BB d in b’s dominance frontier
        - Insert a Phi node for n in d
        - Add d to n’s list of defining BBs

Phi Node Insertion - Example

BB	DF
0	-
1	-
2	7
3	7
4	6
5	6
6	7
7	1



BB0
a =
b =
c =
i =

a = Phi(a,a)
b = Phi(b,b)
c = Phi(c,c)
d = Phi(d,d)
i = Phi(i,i)

BB1
a =
c =

BB2
b =
c =
d =

BB3
a =
d =

BB4
d =

BB5
c =

BB6
b =

c = Phi(c,c)
d = Phi(d,d)

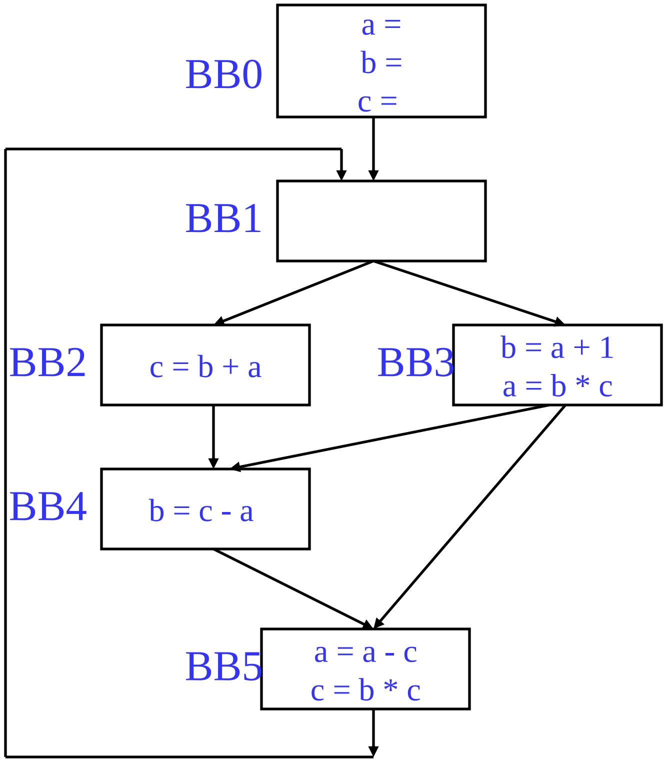
BB7
i =

a = Phi(a,a)
b = Phi(b,b)
c = Phi(c,c)
d = Phi(d,d)

a is defined in 0,1,3
need Phi in 7
then a is defined in 7
need Phi in 1
b is defined in 0, 2, 6
need Phi in 7
then b is defined in 7
need Phi in 1
c is defined in 0,1,2,5
need Phi in 6,7
then c is defined in 7
need Phi in 1
d is defined in 2,3,4
need Phi in 6,7
then d is defined in 7
need Phi in 1
i is defined in BB7
need Phi in BB1

Class Problem

Insert the Phi nodes



BB0
a =
b =
c =

BB1

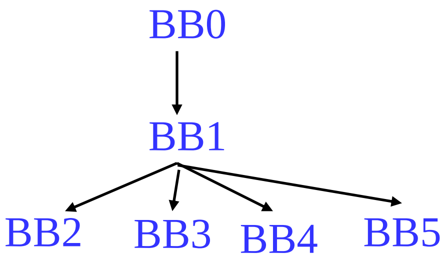
BB2
c = b + a

BB3
b = a + 1
a = b * c

BB4
b = c - a

BB5
a = a - c
c = b * c

Dominator tree



BB0
BB1
BB2
BB3
BB4
BB5

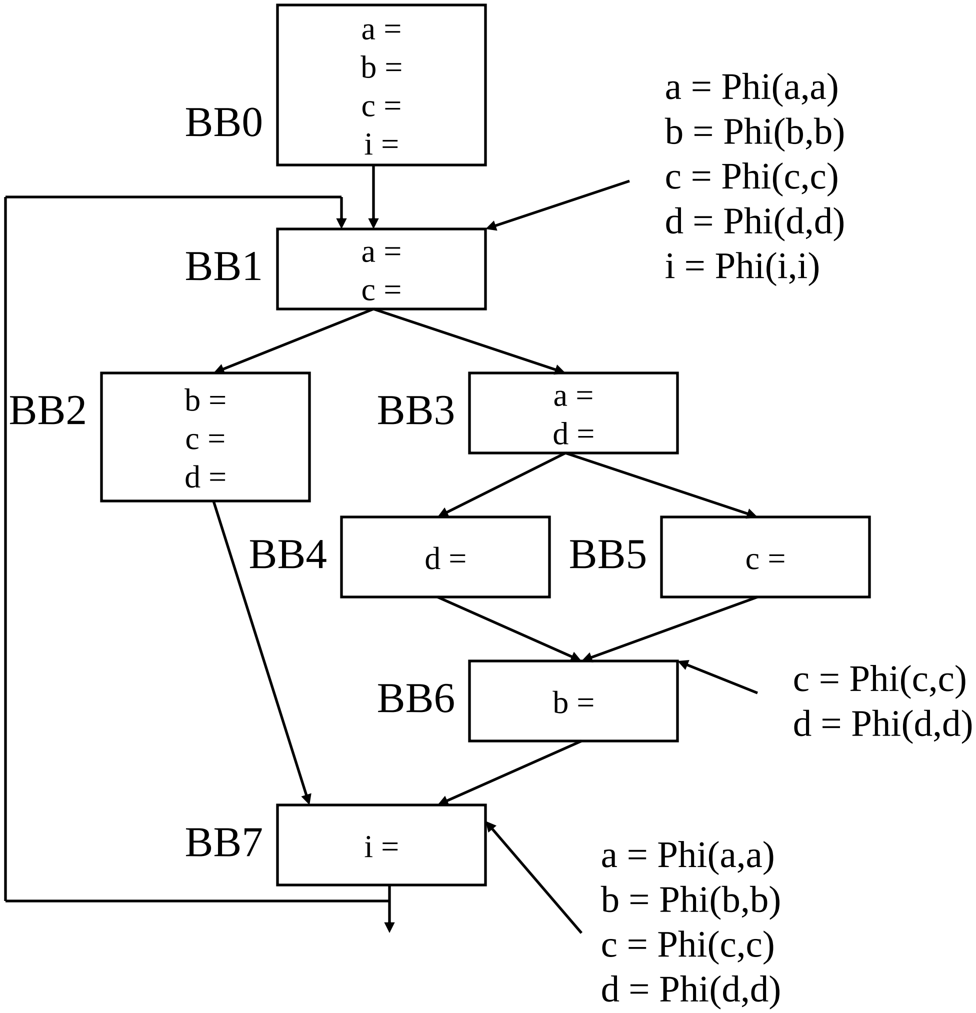
Dominance frontier

BB	DF
0	-
1	-
2	4
3	4, 5
4	5
5	1

SSA Step 2 – Renaming Variables

- Use an array of stacks, one stack per global variable (VR)
- Algorithm sketch
  - For each BB b in a preorder traversal of the dominator tree
    - Generate unique names for each Phi node
    - Rewrite each operation in the BB
      - Uses of global name: current name from stack
      - Defs of global name: create and push new name
    - Fill in Phi node parameters of successor blocks
    - Recurse on b's children in the dominator tree
    - <on exit from b> pop names generated in b from stacks

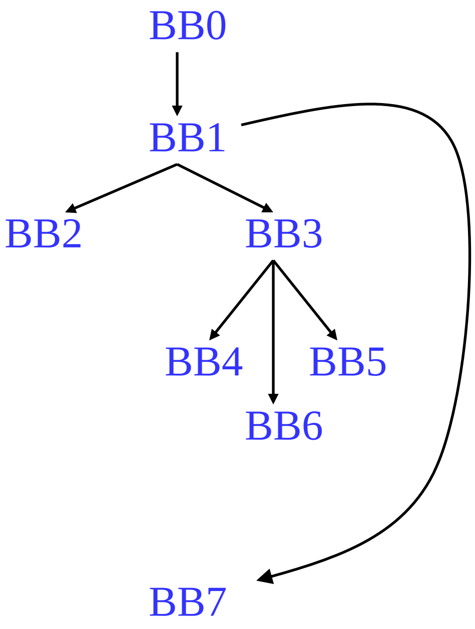
Renaming – Example (Initial State)



a = Phi(a,a)
b = Phi(b,b)
c = Phi(c,c)
d = Phi(d,d)
i = Phi(i,i)

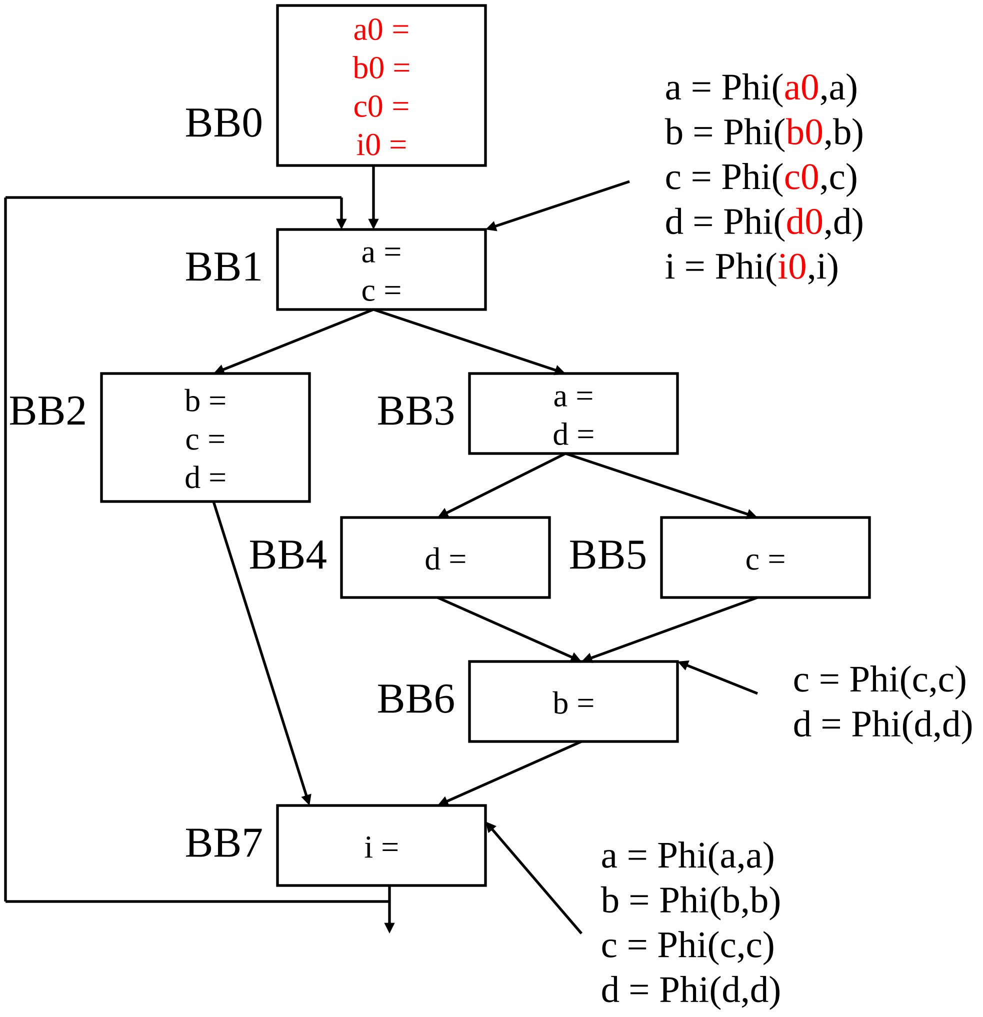
c = Phi(c,c)
d = Phi(d,d)

a = Phi(a,a)
b = Phi(b,b)
c = Phi(c,c)
d = Phi(d,d)



var:	a	b	c	d	i
ctr:	0	0	0	0	0
stk:	a0	b0	c0	d0	i0

Renaming – Example (After BB0)



BB0
a0 =
b0 =
c0 =
i0 =

BB1
a =
c =

a = Phi(a0,a)
b = Phi(b0,b)
c = Phi(c0,c)
d = Phi(d0,d)
i = Phi(i0,i)

BB2
b =
c =
d =

BB3
a =
d =

BB4
d =

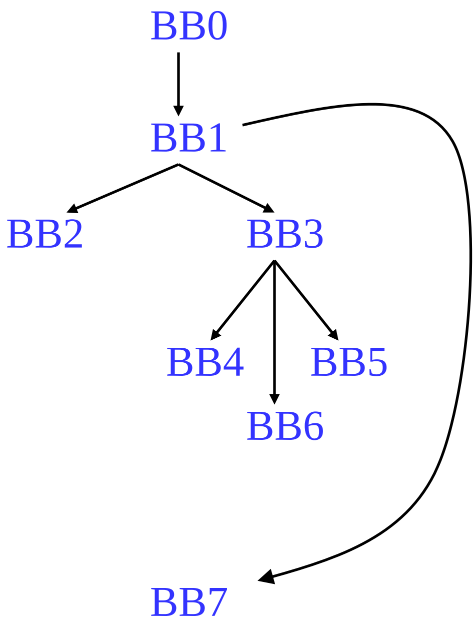
BB5
c =

BB6
b =

c = Phi(c,c)
d = Phi(d,d)

BB7
i =

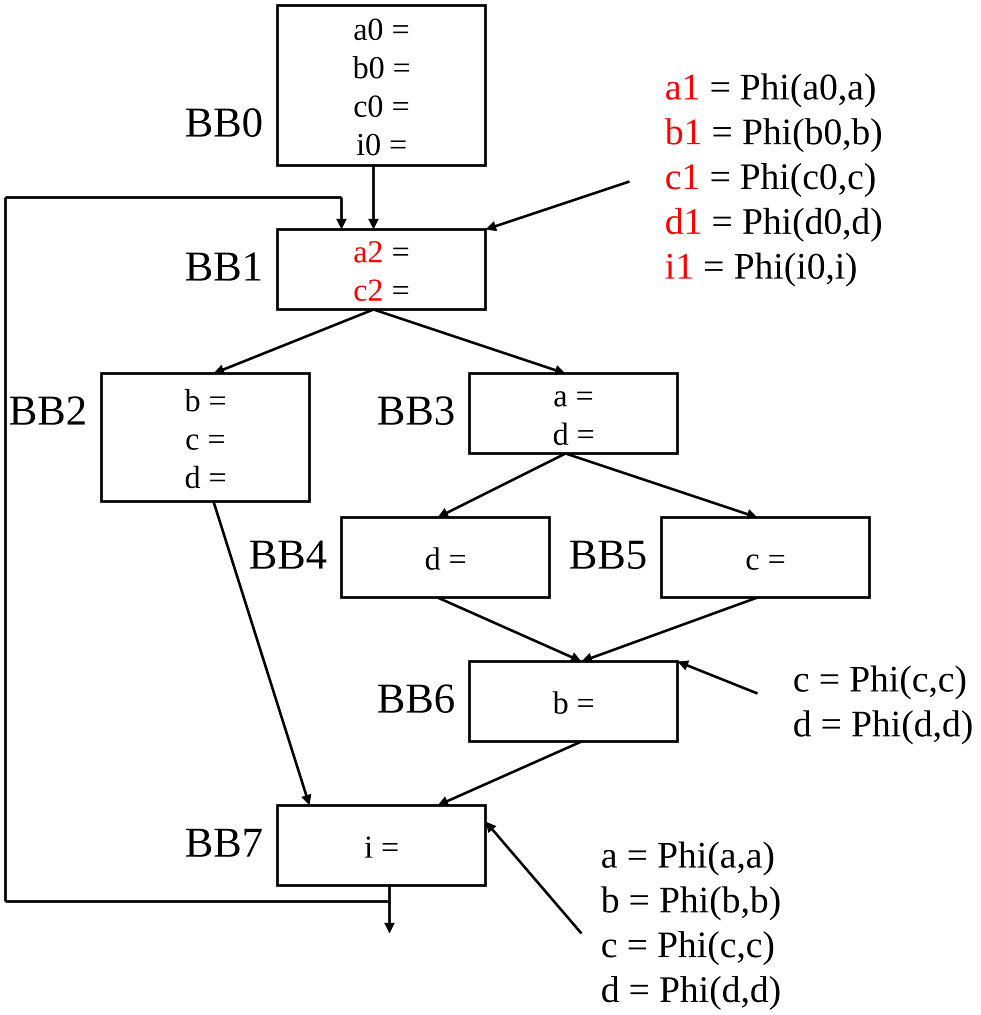
a = Phi(a,a)
b = Phi(b,b)
c = Phi(c,c)
d = Phi(d,d)



BB0
BB1
BB2
BB3
BB4
BB5
BB6
BB7

var:	a	b	c	d	i
ctr:	1	1	1	1	1
stk:	a0	b0	c0	d0	i0

Renaming – Example (After BB1)



BB0

a0 =
b0 =
c0 =
i0 =

a1 = Phi(a0,a)
b1 = Phi(b0,b)
c1 = Phi(c0,c)
d1 = Phi(d0,d)
i1 = Phi(i0,i)

BB1

a2 =
c2 =

BB2

b =
c =
d =

BB3

a =
d =

BB4

d =

BB5

c =

BB6

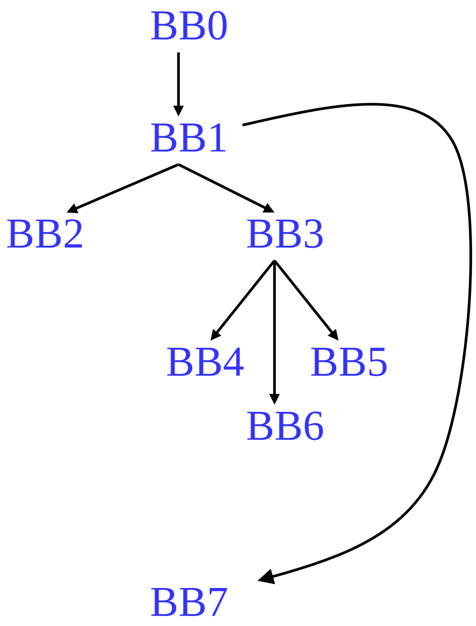
b =

c = Phi(c,c)
d = Phi(d,d)

BB7

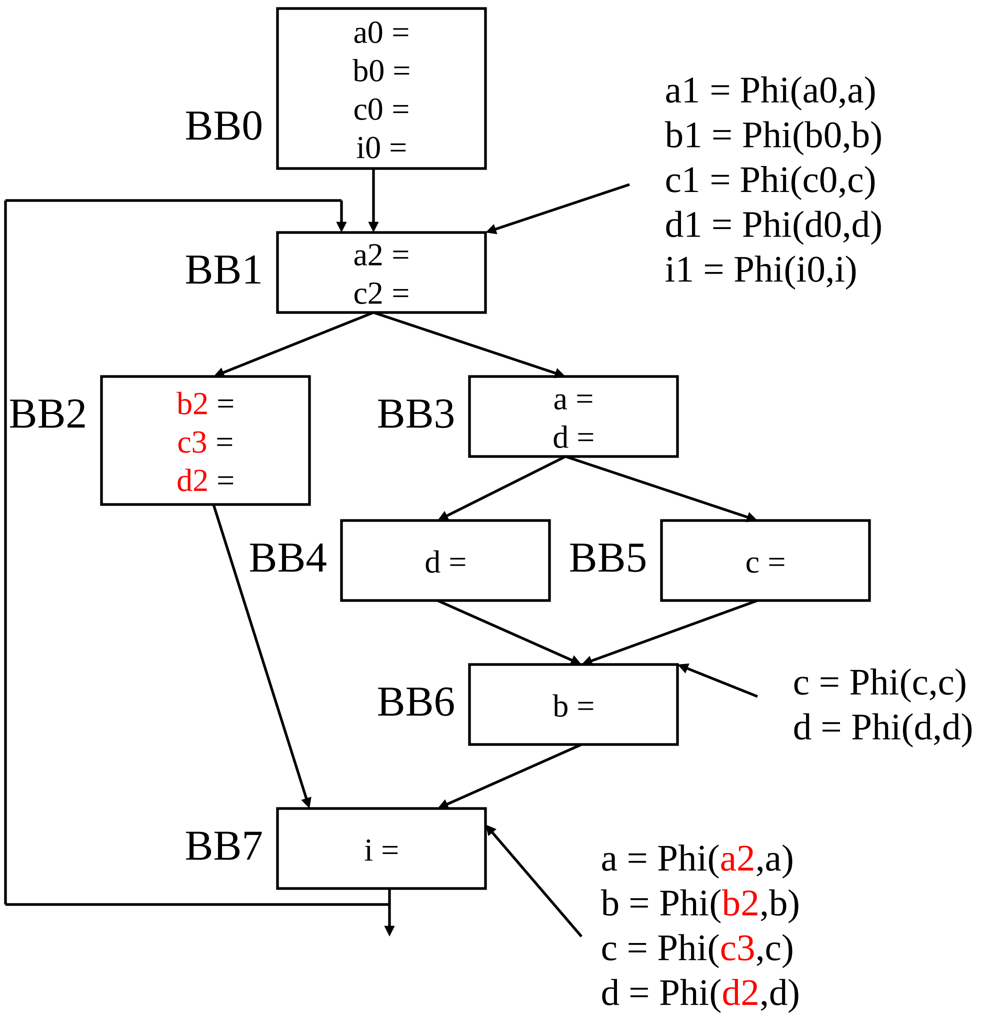
i =

a = Phi(a,a)
b = Phi(b,b)
c = Phi(c,c)
d = Phi(d,d)



var:	a	b	c	d	i
ctr:	3	2	3	2	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2		c2		

Renaming – Example (After BB2)



BB0

a0 =
b0 =
c0 =
i0 =

a1 = Phi(a0,a)
b1 = Phi(b0,b)
c1 = Phi(c0,c)
d1 = Phi(d0,d)
i1 = Phi(i0,i)

BB1

a2 =
c2 =

BB2

b2 =
c3 =
d2 =

BB3

a =
d =

BB4

d =

BB5

c =

BB6

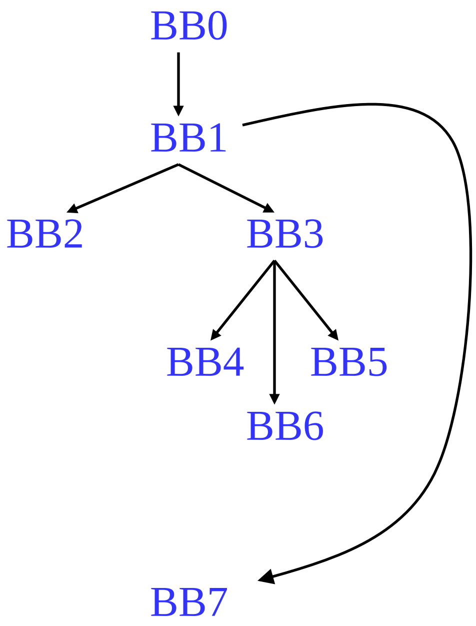
b =

c = Phi(c,c)
d = Phi(d,d)

BB7

i =

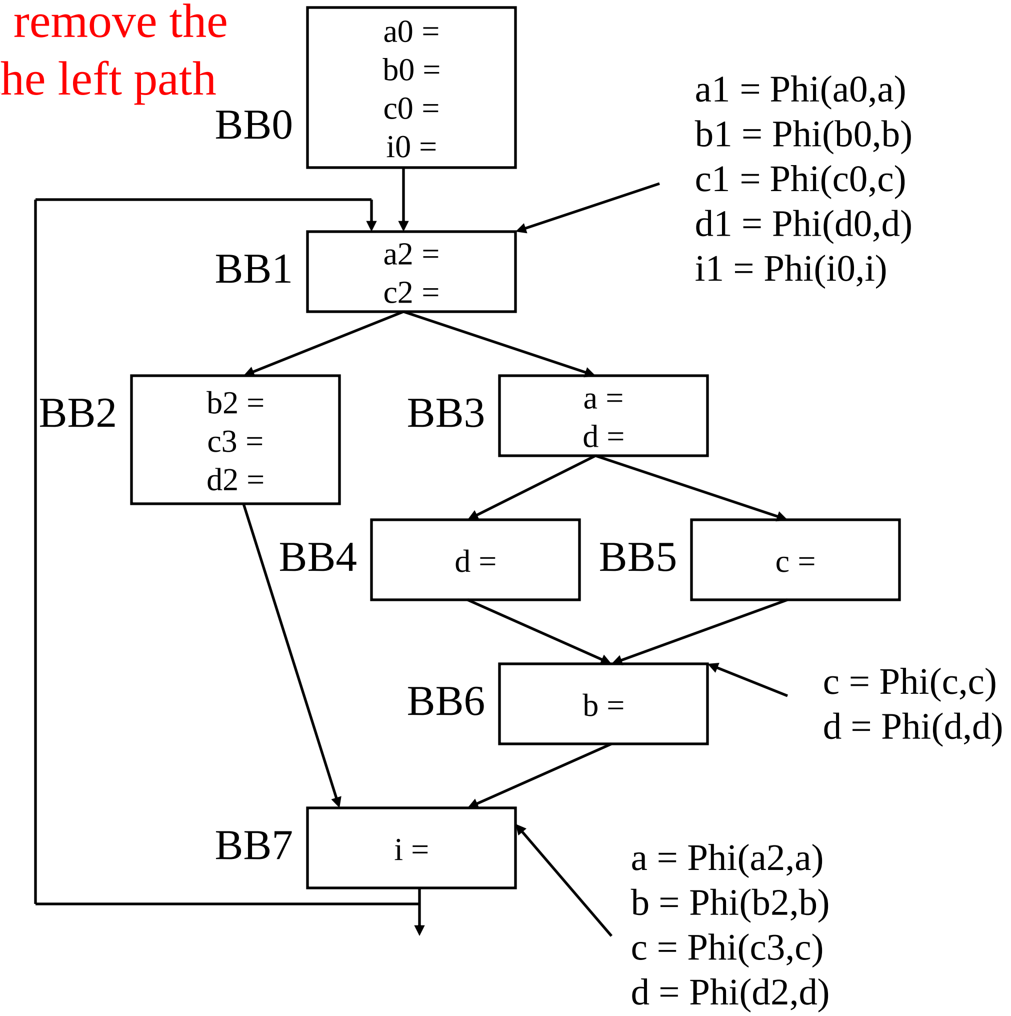
a = Phi(a2,a)
b = Phi(b2,b)
c = Phi(c3,c)
d = Phi(d2,d)



var:	a	b	c	d	i
ctr:	3	3	4	3	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2	b2	c2	d2	
			c3		

Renaming – Example (Before BB3)

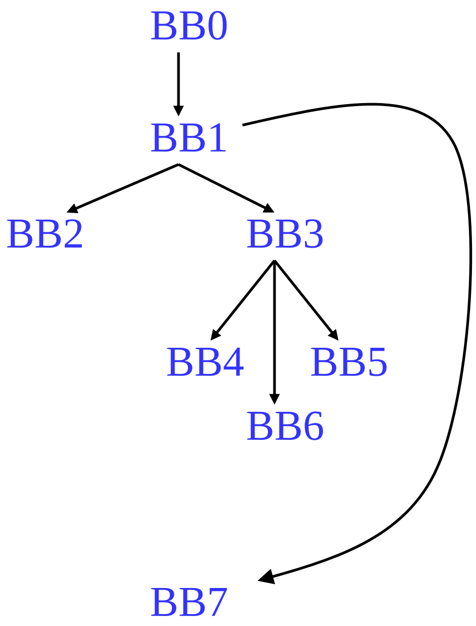
This just updates the stack to remove the stuff from the left path out of BB1



a1 = Phi(a0,a)
b1 = Phi(b0,b)
c1 = Phi(c0,c)
d1 = Phi(d0,d)
i1 = Phi(i0,i)

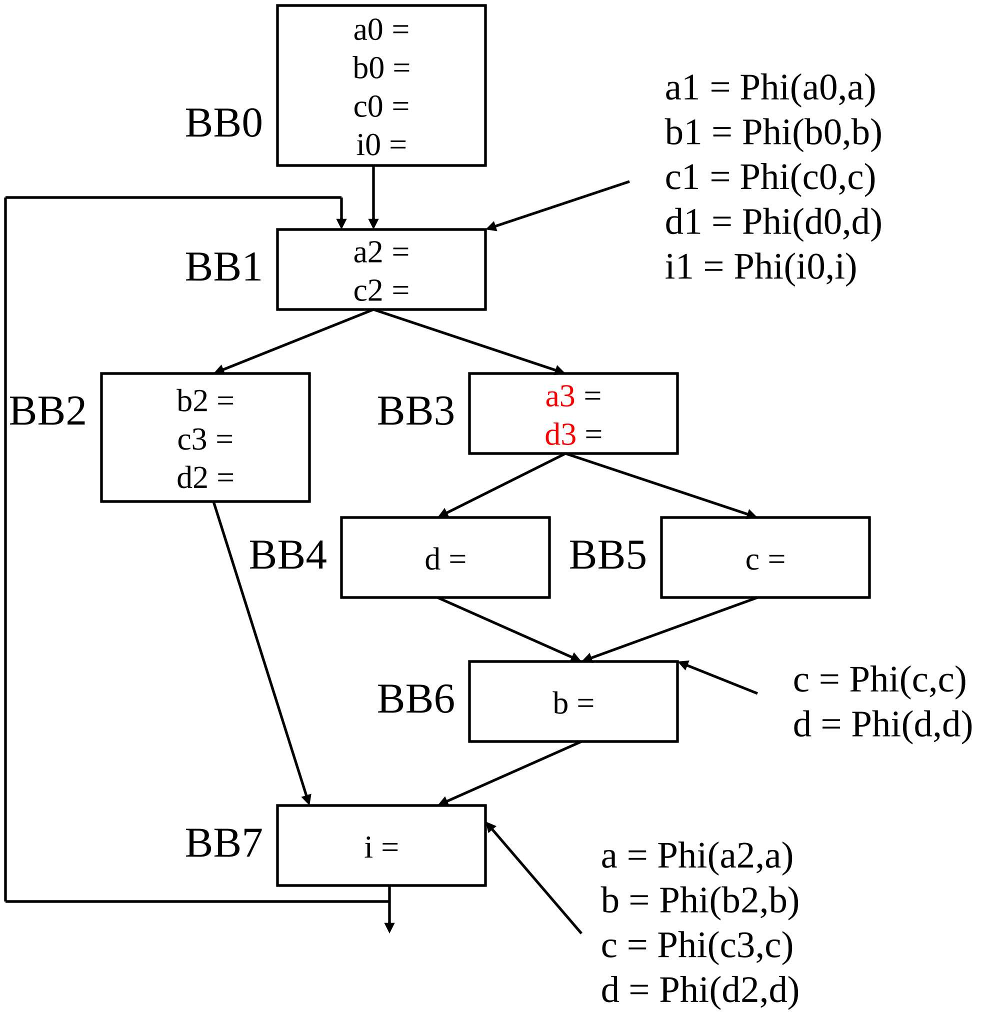
c = Phi(c,c)
d = Phi(d,d)

a = Phi(a2,a)
b = Phi(b2,b)
c = Phi(c3,c)
d = Phi(d2,d)



var:	a	b	c	d	i
ctr:	3	3	4	3	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2		c2		

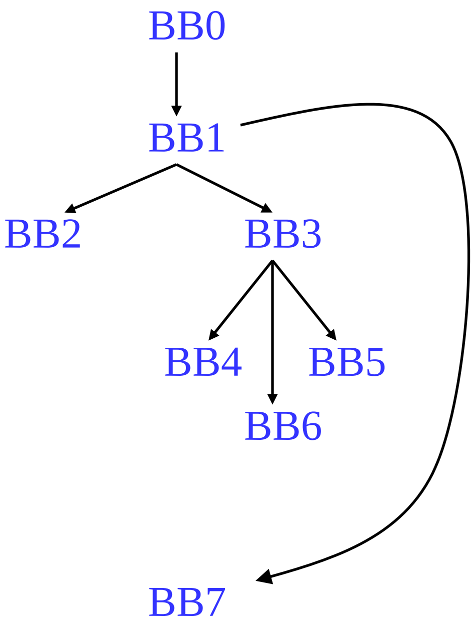
Renaming – Example (After BB3)



a1 = Phi(a0,a)
b1 = Phi(b0,b)
c1 = Phi(c0,c)
d1 = Phi(d0,d)
i1 = Phi(i0,i)

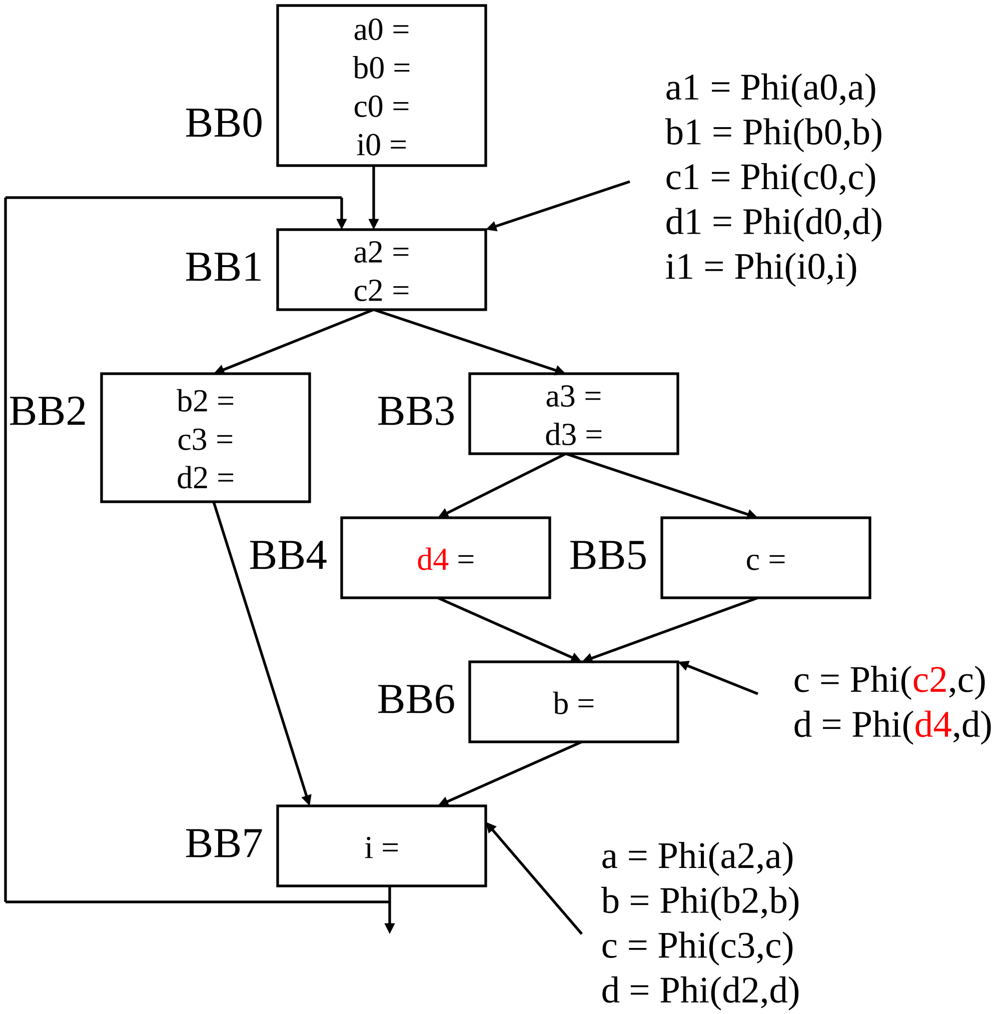
c = Phi(c,c)
d = Phi(d,d)

a = Phi(a2,a)
b = Phi(b2,b)
c = Phi(c3,c)
d = Phi(d2,d)



var:	a	b	c	d	i
ctr:	4	3	4	4	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2		c2	d3	
	a3				

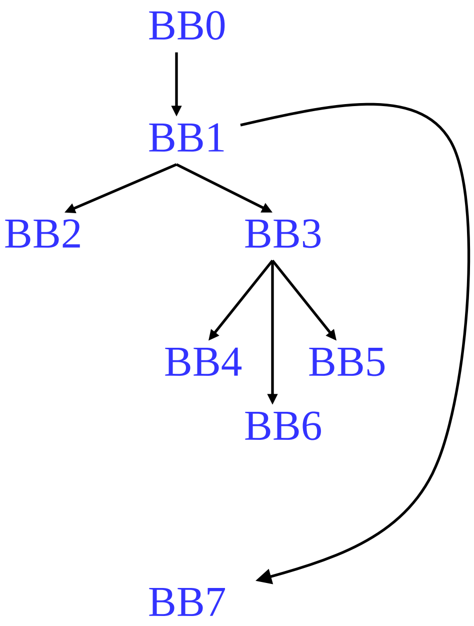
Renaming – Example (After BB4)



a1 = Phi(a0,a)
b1 = Phi(b0,b)
c1 = Phi(c0,c)
d1 = Phi(d0,d)
i1 = Phi(i0,i)

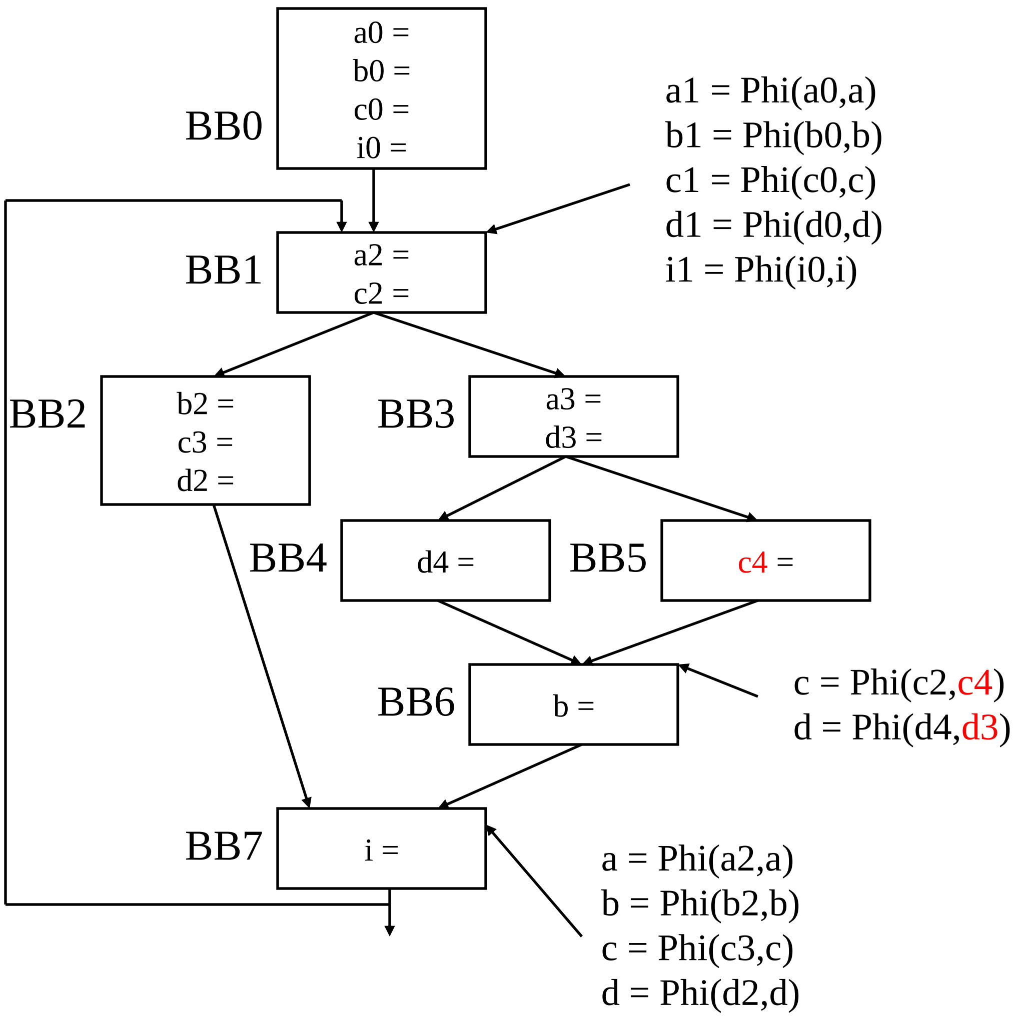
c = Phi(c2,c)
d = Phi(d4,d)

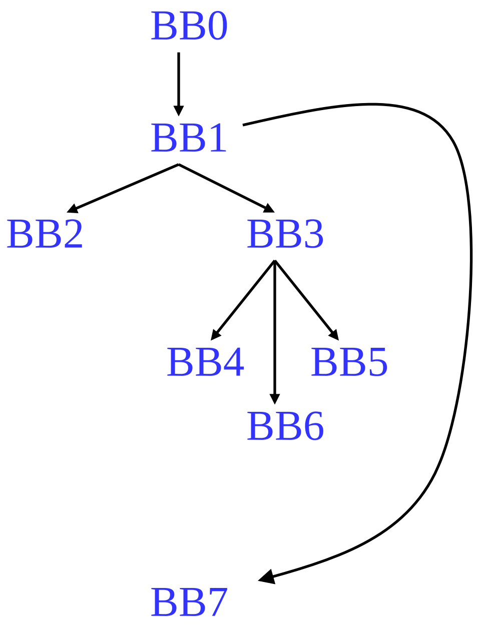
a = Phi(a2,a)
b = Phi(b2,b)
c = Phi(c3,c)
d = Phi(d2,d)



var:	a	b	c	d	i
ctr:	4	3	4	5	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2		c2	d3	
	a3			d4	

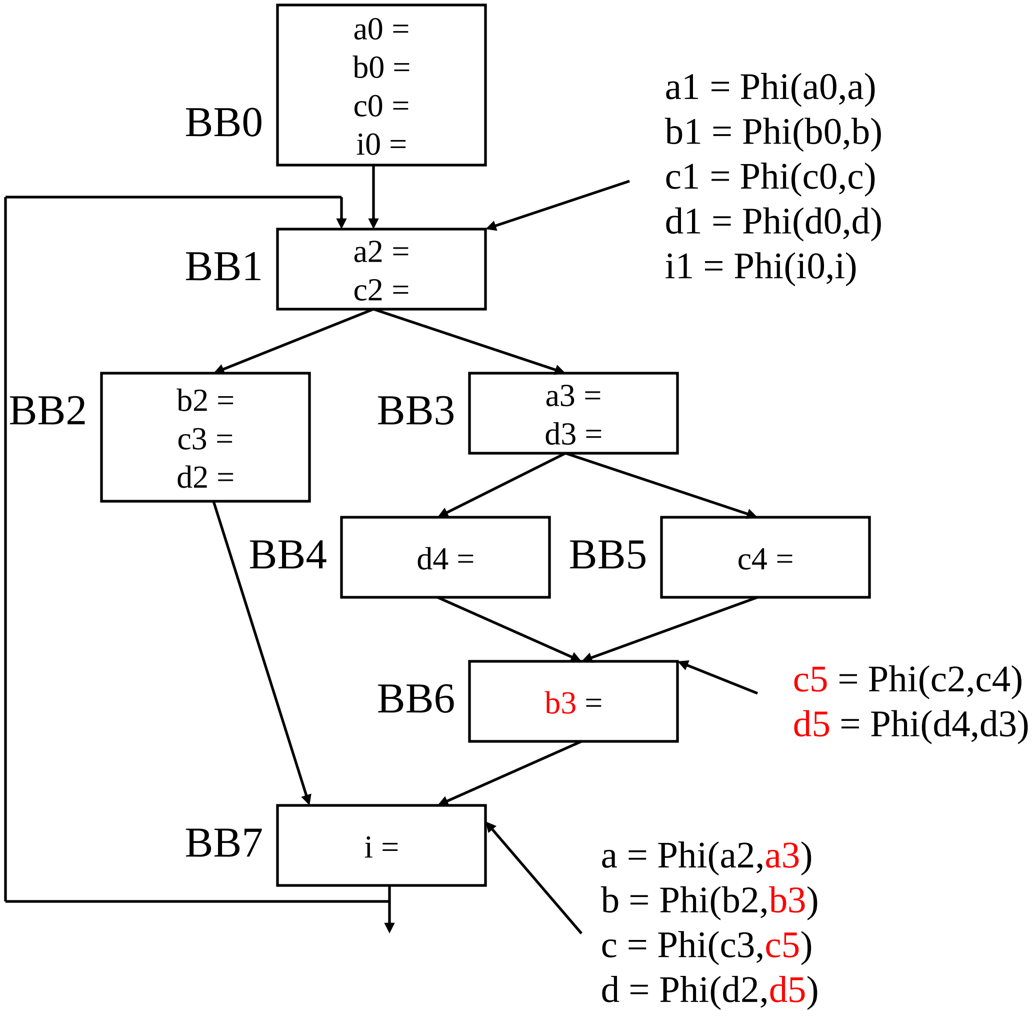
Renaming – Example (After BB5)





var:	a	b	c	d	i
ctr:	4	3	5	5	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2		c2	d3	
	a3		c4		

Renaming – Example (After BB6)



BB0: a0 = b0 = c0 = i0 =

BB1: a2 = c2 =

a1 = Phi(a0,a)
b1 = Phi(b0,b)
c1 = Phi(c0,c)
d1 = Phi(d0,d)
i1 = Phi(i0,i)

BB2: b2 = c3 = d2 =

BB3: a3 = d3 =

BB4: d4 =

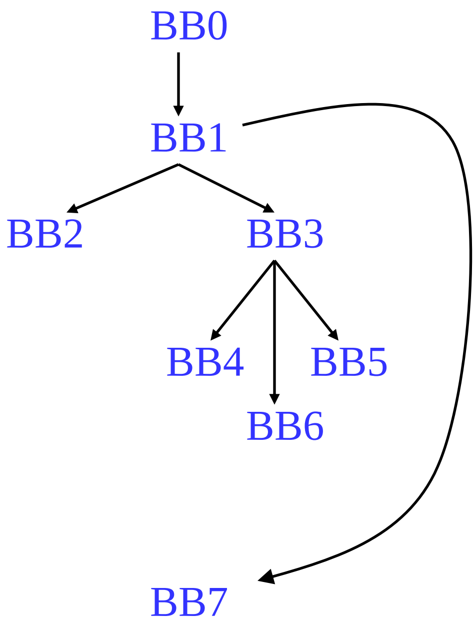
BB5: c4 =

BB6: b3 =

c5 = Phi(c2,c4)
d5 = Phi(d4,d3)

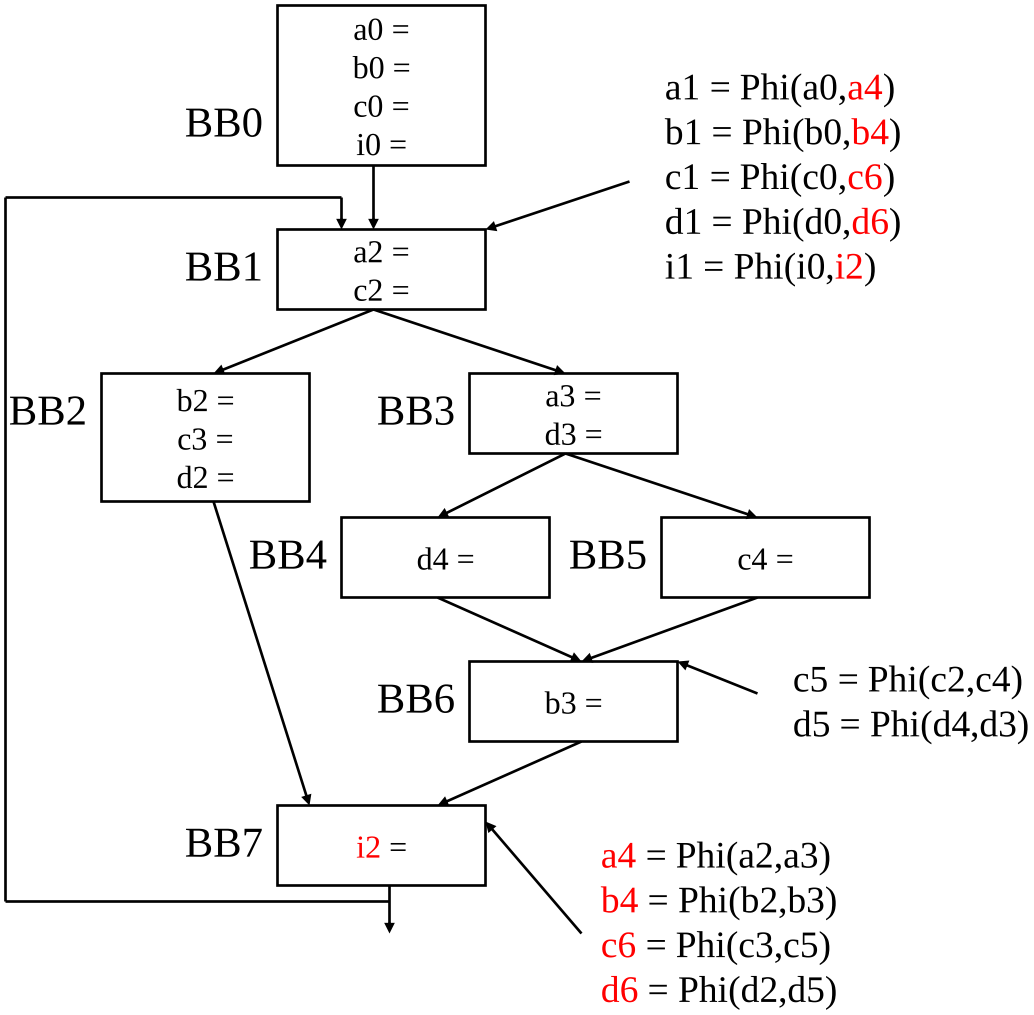
BB7: i =

a = Phi(a2,a3)
b = Phi(b2,b3)
c = Phi(c3,c5)
d = Phi(d2,d5)



var:	a	b	c	d	i
ctr:	4	4	6	6	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2	b3	c2	d3	
	a3		c5	d5	

Renaming – Example (After BB7)



BB0
a0 =
b0 =
c0 =
i0 =

BB1
a2 =
c2 =

a1 = Phi(a0,a4)
b1 = Phi(b0,b4)
c1 = Phi(c0,c6)
d1 = Phi(d0,d6)
i1 = Phi(i0,i2)

BB2
b2 =
c3 =
d2 =

BB3
a3 =
d3 =

BB4
d4 =

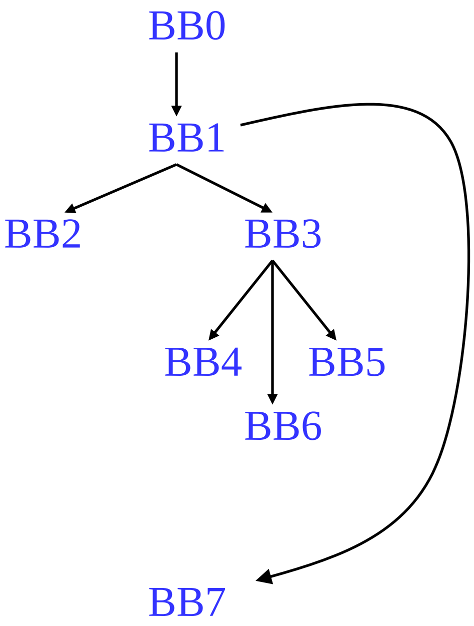
BB5
c4 =

BB6
b3 =

c5 = Phi(c2,c4)
d5 = Phi(d4,d3)

BB7
i2 =

a4 = Phi(a2,a3)
b4 = Phi(b2,b3)
c6 = Phi(c3,c5)
d6 = Phi(d2,d5)



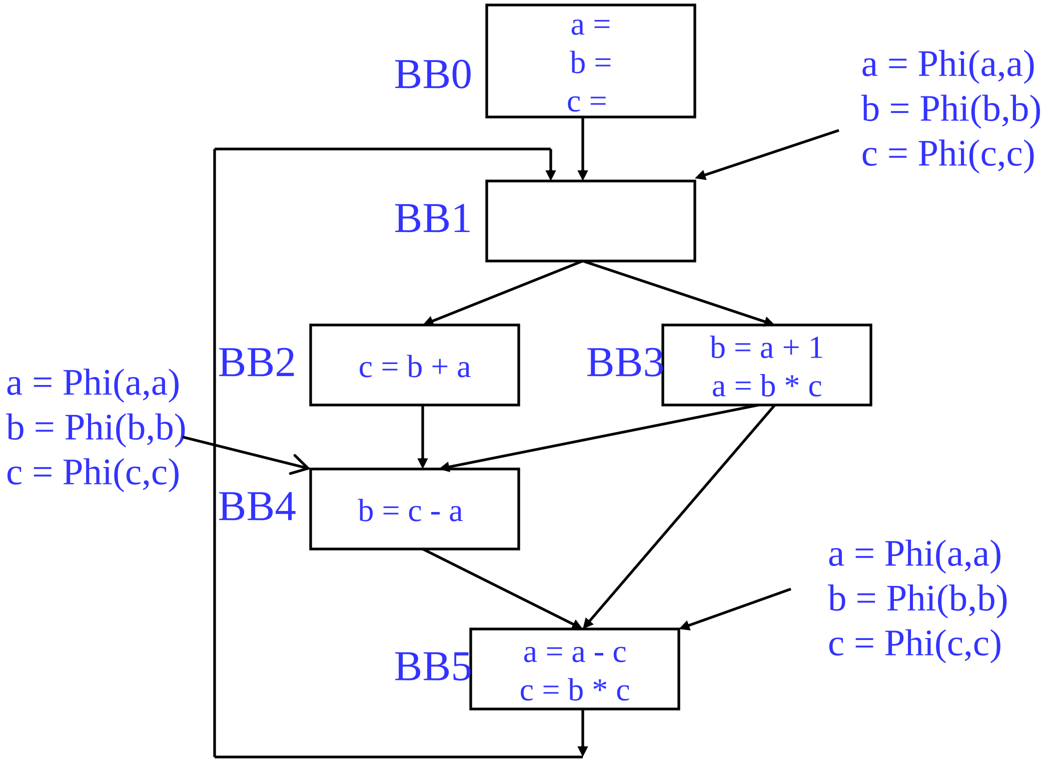
BB0
BB1
BB2
BB3
BB4
BB5
BB6
BB7

var:	a	b	c	d	i
ctr:	5	5	7	7	3
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2	b4	c2	d6	i2
	a4		c6		

Fin!

Class Problem

Rename the variables



Dominance frontier

BB	DF
0	-
1	-
2	4
3	4, 5
4	5
5	1