EECS 583 – Class 4

If-conversion

University of Michigan

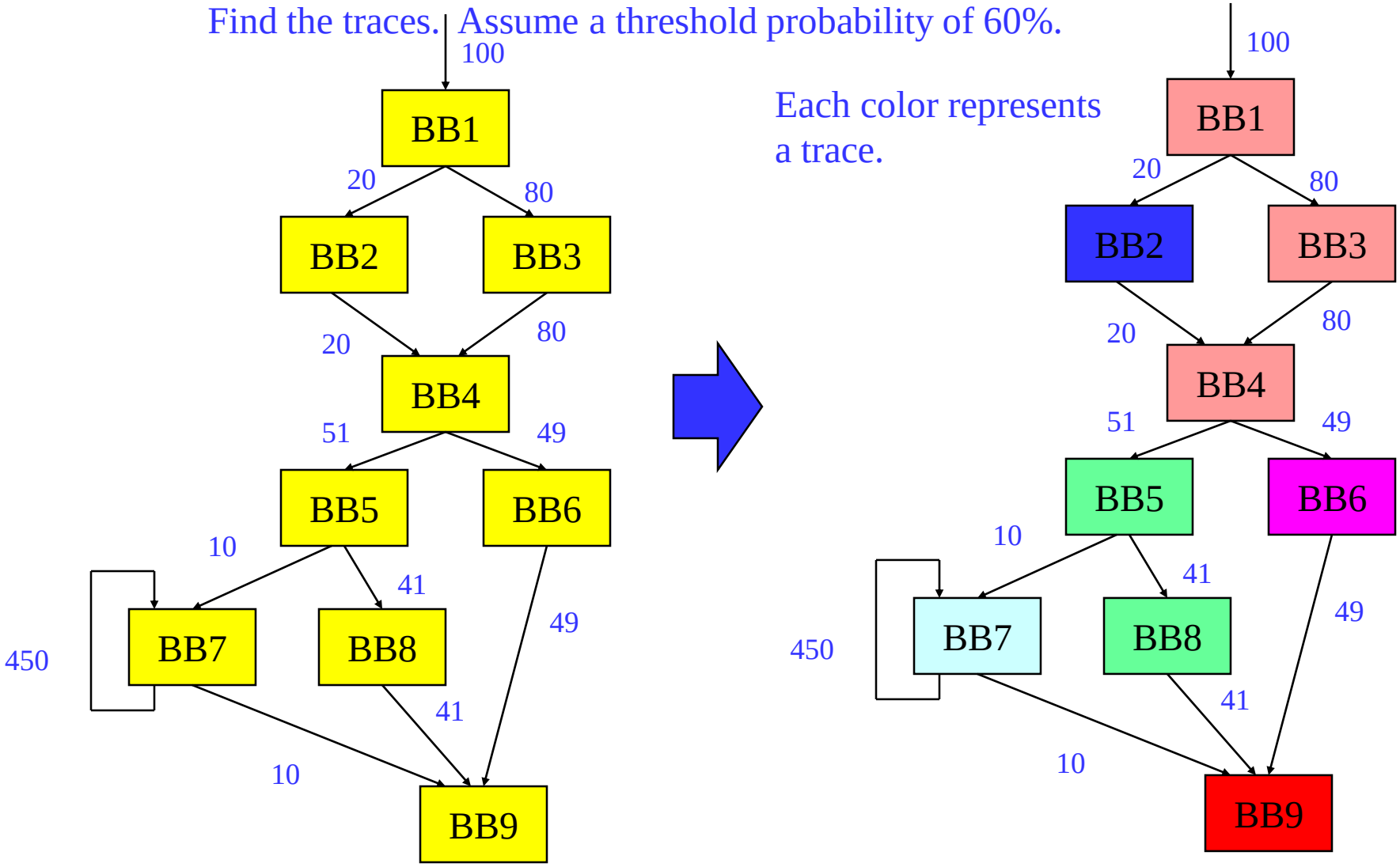
September 16, 2019

Announcements & Reading Material

- HW 1 – Deadline Friday Sept 20, midnight
  - Talk to Sung this week if you are having troubles with LLVM
  - Refer to EECS 583 piazza group for tips and answers to questions
- Today's class
  - "The Program Dependence Graph and Its Use in Optimization", J. Ferrante, K. Ottenstein, and J. Warren, ACM TOPLAS, 1987
    - This is a long paper – the part we care about is the control dependence stuff. The PDG is interesting and you should skim it over.
    - "On Predicated Execution", Park and Schlansker, HPL Technical Report, 1991.
- Material for Wednesday
  - *Compilers: Principles, Techniques, and Tools*, A. Aho, R. Sethi, and J. Ullman, Addison-Wesley, 1988. (Sections: 10.5, 10.6 Edition 1) (Sections 9.2 Edition 2)

From Last Time: Class Problem - Answer

Find the traces. Assume a threshold probability of 60%.



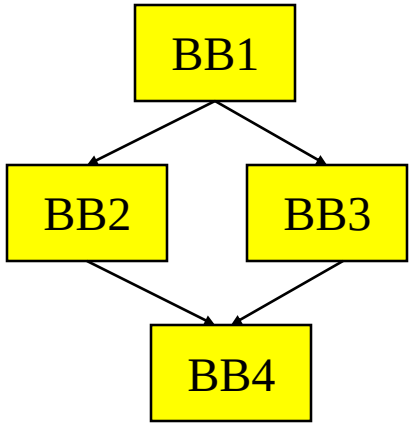
An Alternative to Branches: Predicated Execution

- Hardware mechanism that allows operations to be conditionally executed
- Add an additional boolean source operand (predicate)
  - ADD r1, r2, r3 if p1
    - if (p1 is True), r1 = r2 + r3
    - else if (p1 is False), do nothing (Add treated like a NOP)
    - p1 referred to as the guarding predicate
    - Predicated on True means always executed
    - Omitted predicated also means always executed
- Provides compiler with an alternative to using branches to selectively execute operations
  - If statements in the source
  - Realize with branches in the assembly code
  - Could also realize with conditional instructions
  - Or use a combination of both

Predicated Execution Example

```
a = b + c
if (a > 0)
    e = f + g
else
    e = f / g
h = i - j
```

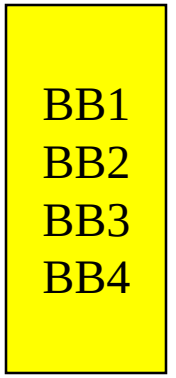
```
BB1    add a, b, c
BB1    bgt a, 0, L1
BB3    div e, f, g
BB3    jump L2
BB2    L1: add e, f, g
BB4    L2: sub h, i, j
```



Traditional branching code

p2 → BB2
p3 → BB3

```
BB1    add a, b, c if T
BB1    p2 = a > 0 if T
BB1    p3 = a <= 0 if T
BB3    div e, f, g if p3
BB2    add e, f, g if p2
BB4    sub h, i, j if T
```

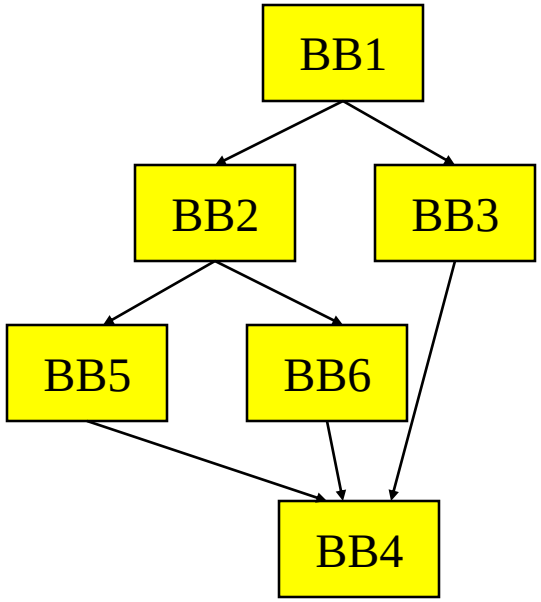


Predicated code

What About Nested If-then-else's?

```
a = b + c
if (a > 0)
   if (a > 25)
      e = f + g
   else
      e = f * g
else
   e = f / g
h = i - j
```

```
BB1   add a, b, c
BB1   bgt a, 0, L1
BB3   div e, f, g
BB3   jump L2
BB2   L1: bgt a, 25, L3
BB6   mpy e, f, g
BB6   jump L2
BB5   L3: add e, f, g
BB4   L2: sub h, i, j
```



Traditional branching code

Nested If-then-else's – No Problem

```
a = b + c
if (a > 0)
   if (a > 25)
      e = f + g
   else
      e = f * g
else
   e = f / g
h = i - j
```

```
BB1   add a, b, c if T
BB1   p2 = a > 0 if T
BB1   p3 = a <= 0 if T
BB3   div e, f, g if p3
BB3   p5 = a > 25 if p2
BB3   p6 = a <= 25 if p2
BB6   mpy e, f, g if p6
BB5   add e, f, g if p5
BB4   sub h, i, j if T
```

BB1
BB2
BB3
BB4
BB5
BB6

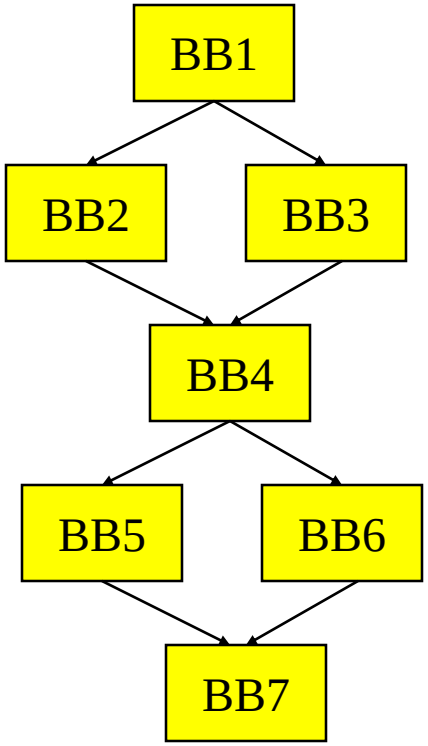
Predicated code

What do we assume to make this work ??

if p2 is False, both p5 and p6 are False

So, predicate setting instruction should set result to False if guarding predicate is false!!!

Benefits/Costs of Predicated Execution



Benefits:
- No branches, no mispredicts
- Can freely reorder independent operations in the predicated block
- Overlap BB2 with BB5 and BB6

Costs (execute all paths)
-worst case schedule length
-worst case resources required

HPL-PD Compare-to-Predicate Operations (CMPPs)

- How do we compute predicates
  - Compare registers/literals like a branch would do
  - Efficiency, code size, nested conditionals, etc
- 2 targets for computing taken/fall-through conditions with 1 operation

p1, p2 = CMPP.cond.D1a.D2a (r1, r2) if p3

p1 = first destination predicate
p2 = second destination predicate
cond = compare condition (ie EQ, LT, GE, …)
D1a = action specifier for first destination
D2a = action specifier for second destination
(r1,r2) = data inputs to be compared (ie r1 < r2)
p3 = guarding predicate

CMPP Action Specifiers

Guarding predicate	Compare Result	UN	UC	ON	OC	AN	AC
0	0	0	0	-	-	-	-
0	1	0	0	-	-	-	-
1	0	0	1	-	1	0	-
1	1	1	0	1	-	-	0

UN/UC = Unconditional normal/complement

This is what we used in the earlier examples

guard = 0, both outputs are 0

guard = 1, UN = Compare result, UC = opposite

ON/OC = OR-type normal/complement

AN/AC = AND-type normal/complement

OR-type, AND-type Predicates

p1 = 0
p1 = cmpp_ON (r1 < r2) if T
p1 = cmpp_OC (r3 < r4) if T
p1 = cmpp_ON (r5 < r6) if T

p1 = (r1 < r2) | (!(r3 < r4)) |
(r5 < r5)

Wired-OR into p1

Generating predicated code
for some source code requires
OR-type predicates

p1 = 1
p1 = cmpp_AN (r1 < r2) if T
p1 = cmpp_AC (r3 < r4) if T
p1 = cmpp_AN (r5 < r6) if T

p1 = (r1 < r2) & (!(r3 < r4)) &
(r5 < r5)

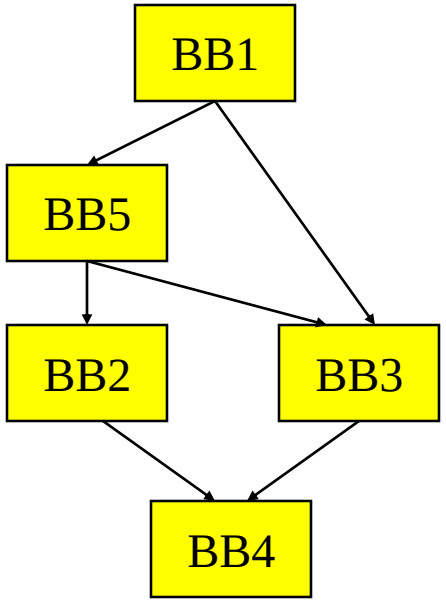
Wired-AND into p1

Talk about these later – used
for control height reduction

Use of OR-type Predicates

```
a = b + c
if (a > 0 && b > 0)
    e = f + g
else
    e = f / g
h = i - j
```

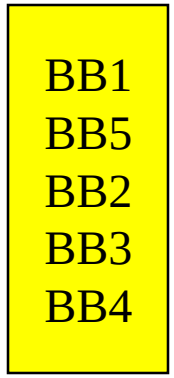
```
BB1    add a, b, c
BB1    ble a, 0, L1
BB5    ble b, 0, L1
BB2    add e, f, g
BB2    jump L2
BB3    L1: div e, f, g
BB4    L2: sub h, i, j
```



Traditional branching code

p2 → BB2
p3 → BB3
p5 → BB5

```
BB1    add a, b, c if T
BB1    p3, p5 = cmpp.ON.UC a <= 0 if T
BB5    p3, p2 = cmpp.ON.UC b <= 0 if p5
BB3    div e, f, g if p3
BB2    add e, f, g if p2
BB4    sub h, i, j if T
```



Predicated code

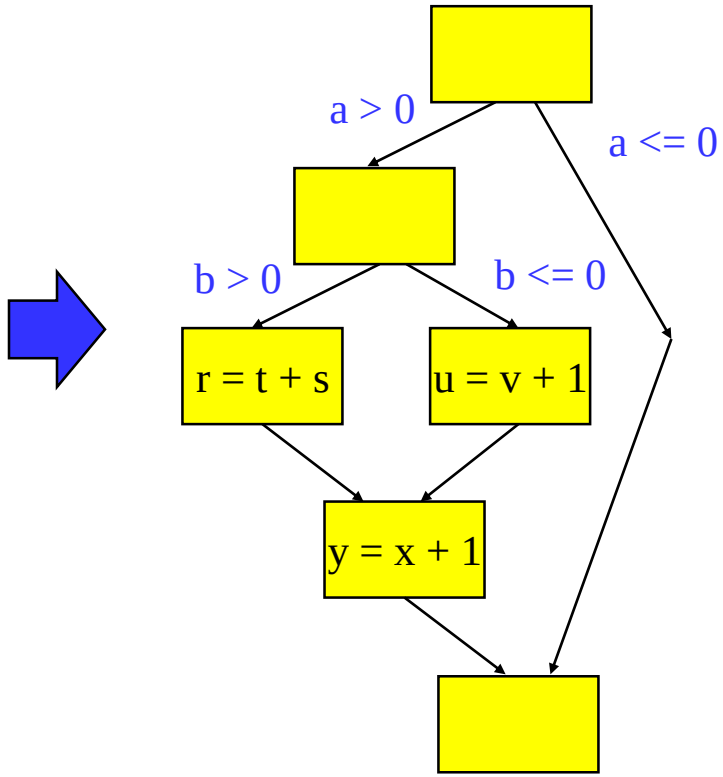
Homework Problem – Answer on next slide but don't cheat!

```
if (a > 0) {
    if (b > 0)
        r = t + s
    else
        u = v + 1
    y = x + 1
}
```

a. Draw the CFG
b. Predicate the code removing all branches

Homework Problem Answer

```
if (a > 0) {
    if (b > 0)
        r = t + s
    else
        u = v + 1
    y = x + 1
}
```




```
p1 = cmpp.UN(a > 0) if T
p2, p3 = cmpp.UNUC(b > 0) if p1
r = t + s if p2
u = v + 1 if p3
y = x + 1 if p1
```

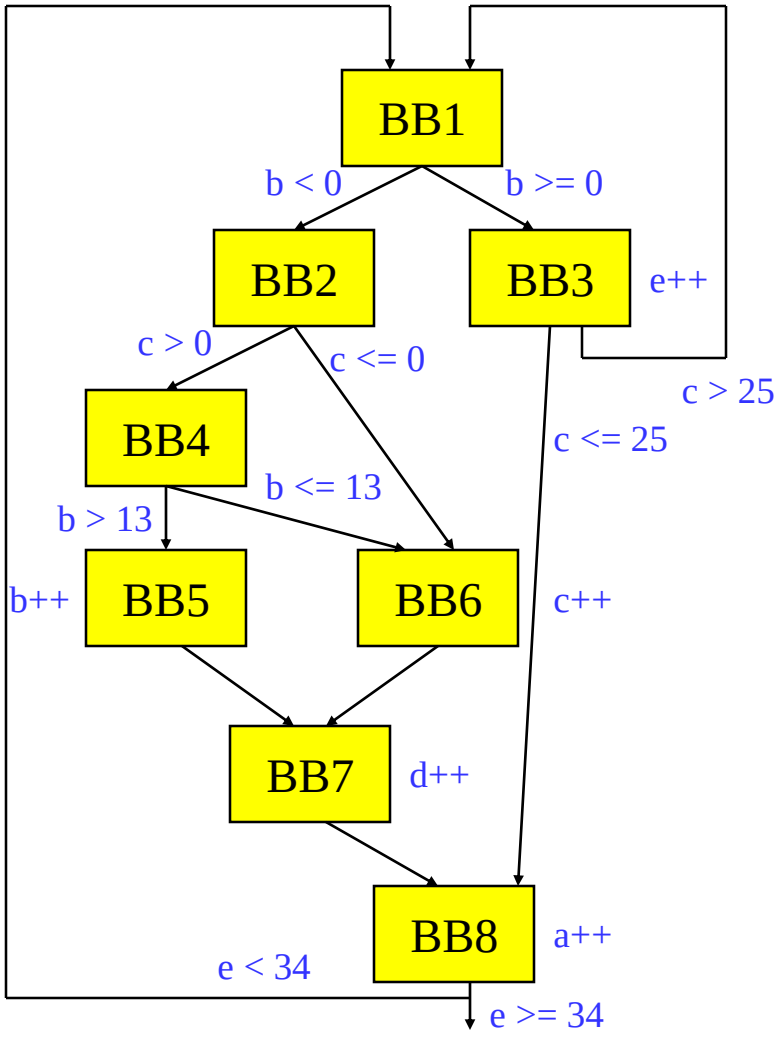
a. Draw the CFG
b. Predicate the code removing all branches

If-conversion

- Algorithm for generating predicated code
  - Automate what we've been doing by hand
  - Handle arbitrary complex graphs
    - But, acyclic subgraph only!!
    - Need a branch to get you back to the top of a loop
  - Efficient
- Roots are from Vector computer days
  - Vectorize a loop with an if-statement in the body
- 4 steps
  - 1. Loop backedge coalescing
  - 2. Control dependence analysis
  - 3. Control flow substitution
  - 4. CMPP compaction
- My version of Park & Schlansker

Running Example – Initial State

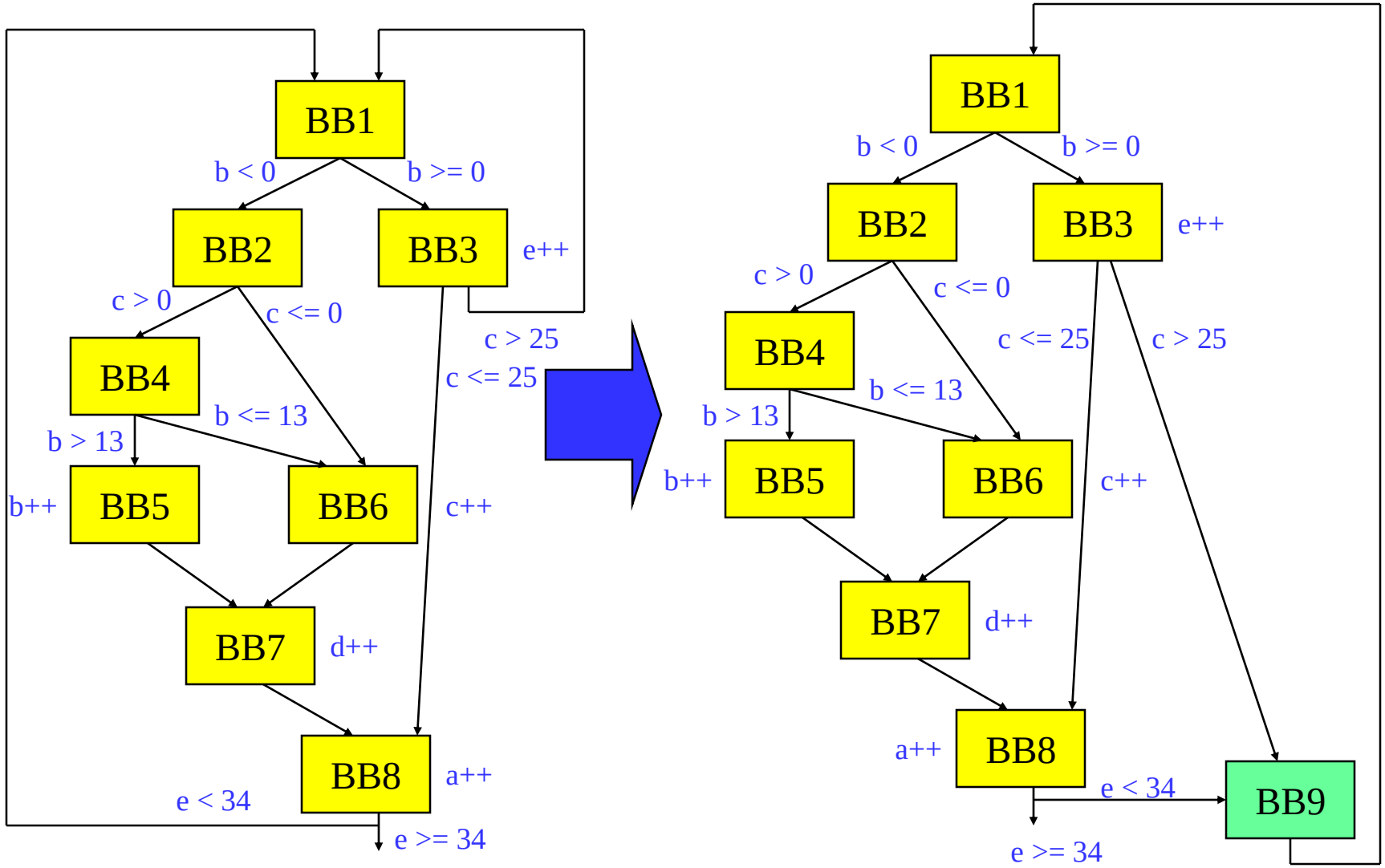
```
do {
    b = load(a)
    if (b < 0) {
        if ((c > 0) && (b > 13))
            b = b + 1
        else
            c = c + 1
        d = d + 1
    }
    else {
        e = e + 1
        if (c > 25) continue
    }
    a = a + 1
} while (e < 34)
```



Step 1: Backedge Coalescing

- Recall – Loop backedge is branch from inside the loop back to the loop header
- This step only applicable for a loop body
  - If not a loop body → skip this step
- Process
  - Create a new basic block
    - New BB contains an unconditional branch to the loop header
  - Adjust all other backedges to go to new BB rather than header
- Why do this?
  - Heuristic step – Not essential for correctness
    - If-conversion cannot remove backedges (only forward edges)
    - But this allows the control logic to figure out which backedge you take to be eliminated
  - Generally this is a good thing to do

Running Example – Backedge Coalescing



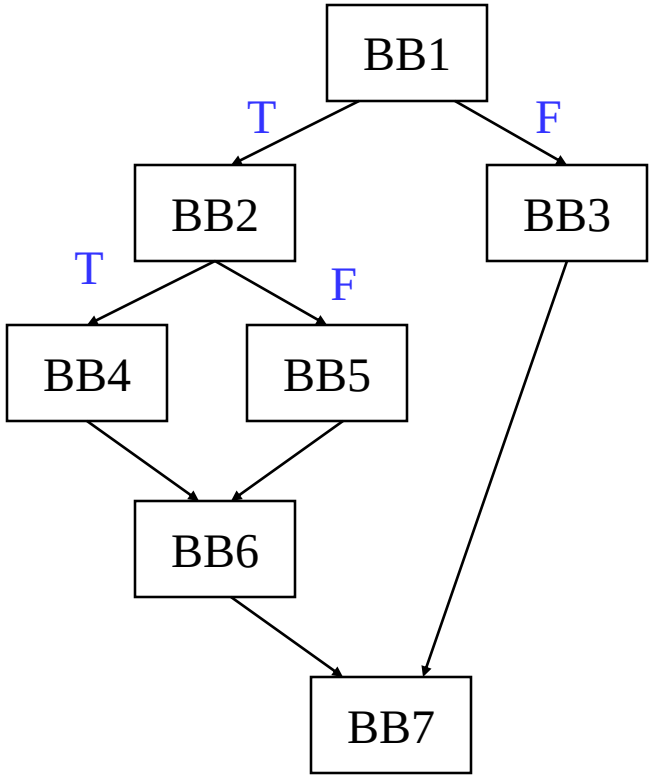
Step 2: Control Dependence Analysis (CD)

- Control flow – Execution transfer from 1 BB to another via a taken branch or fallthrough path
- Dependence – Ordering constraint between 2 operations
  - Must execute in proper order to achieve the correct result
  - O1: a = b + c
  - O2: d = a – e
  - O2 dependent on O1
- Control dependence – One operation controls the execution of another
  - O1: blt a, 0, SKIP
  - O2: b = c + d
  - SKIP:
  - O2 control dependent on O1
- Control dependence analysis derives these dependences

Control Dependences

- Recall
  - Post dominator – BBX is post dominated by BBY if every path from BBX to EXIT contains BBY
  - Immediate post dominator – First breadth first successor of a block that is a post dominator
- Control dependence – BBY is control dependent on BBX iff
  - 1. There exists a directed path P from BBX to BBY with any BBZ in P (excluding BBX and BBY) post dominated by BBY
  - 2. BBX is not post dominated by BBY
- In English,
  - A BB is control dependent on the closest BB(s) that determine(s) its execution
  - Its actually not a BB, it's a control flow edge coming out of a BB

Control Dependence Example



Control dependences

BB1:

BB2:

BB3:

BB4:

BB5:

BB6:

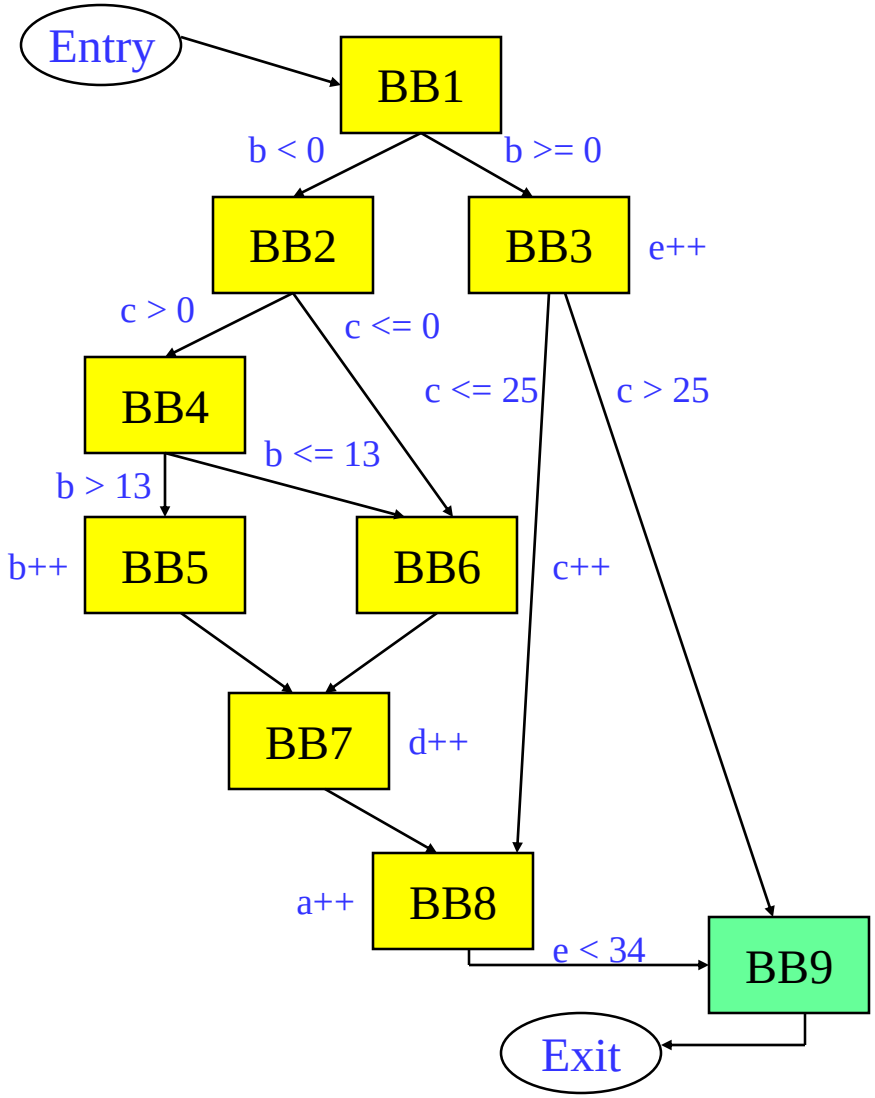
BB7:

Notation

positive BB number = fallthru direction

negative BB number = taken direction

Running Example – CDs



First, nuke backedge(s)
Second, nuke exit edges
Then, Add pseudo entry/exit nodes
- Entry → nodes with no predecessors
- Exit → nodes with no successors

Control deps (left is taken)

BB1:
BB2:
BB3:
BB4:
BB5:
BB6:
BB7:
BB8:
BB9:

Algorithm for Control Dependence Analysis

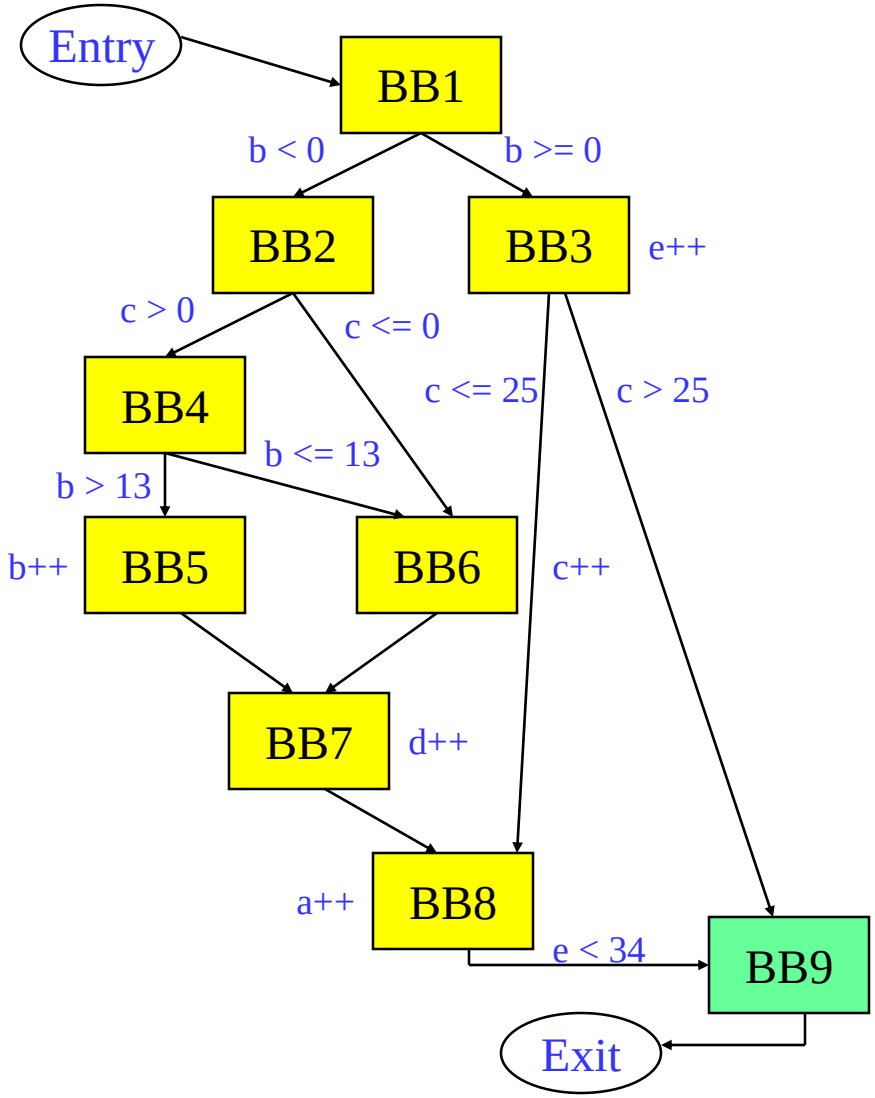
```
for each basic block x in region
   for each outgoing control flow edge e of x
      y = destination basic block of e
      if (y not in pdom(x)) then
         lub = ipdom(x)
         if (e corresponds to a taken branch) then
            x_id = -x.id
         else
            x_id = x.id
         endif
         t = y
         while (t != lub) do
            cd(t) += x_id;
            t = ipdom(t)
         endwhile
      endif
   endfor
endfor
```

Notes

Compute cd(x) which contains those BBs which x is control dependent on

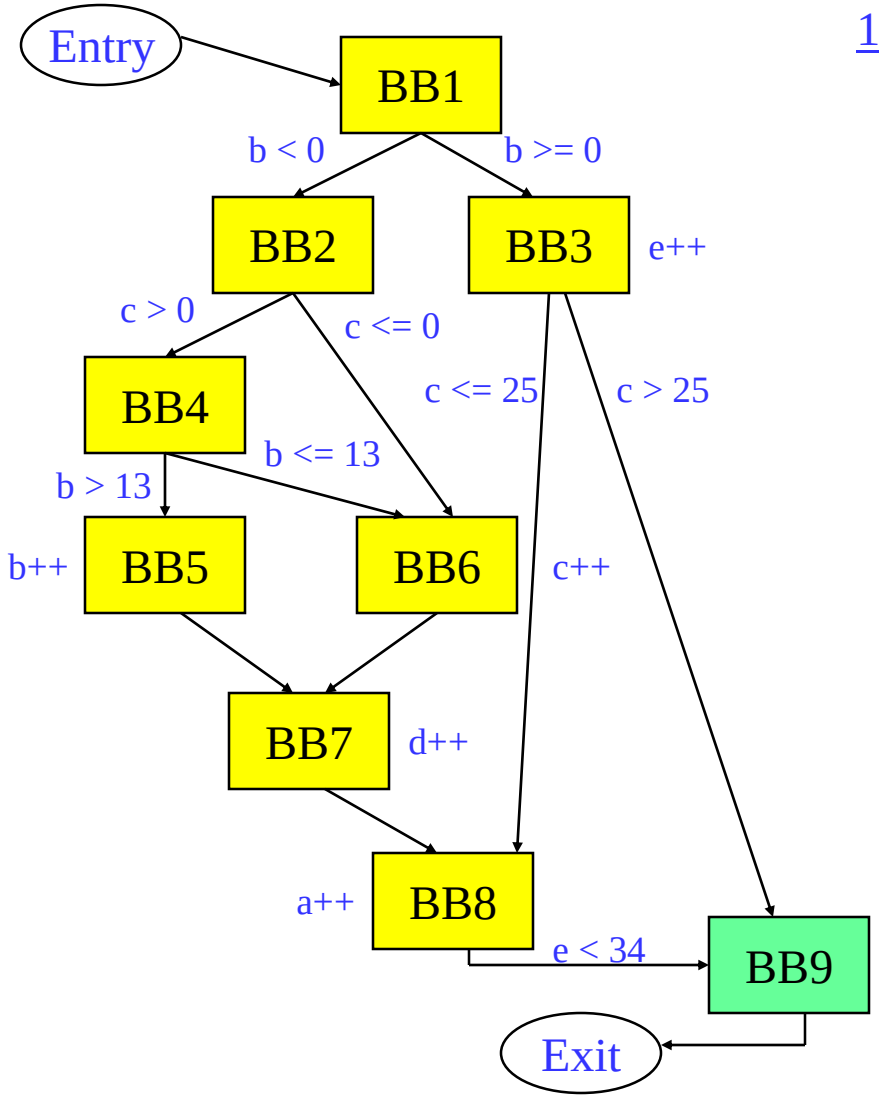
Iterate on per edge basis, adding edge to each cd set it is a member of

Running Example – Post Dominators



	pdom	ipdom
BB1:	1, 9, ex	9
BB2:	2, 7, 8, 9, ex	7
BB3:	3, 9, ex	9
BB4:	4, 7, 8, 9, ex	7
BB5:	5, 7, 8, 9, ex	7
BB6:	6, 7, 8, 9, ex	7
BB7:	7, 8, 9, ex	8
BB8:	8, 9, ex	9
BB9:	9, ex	ex

Running Example – CDs Via Algorithm

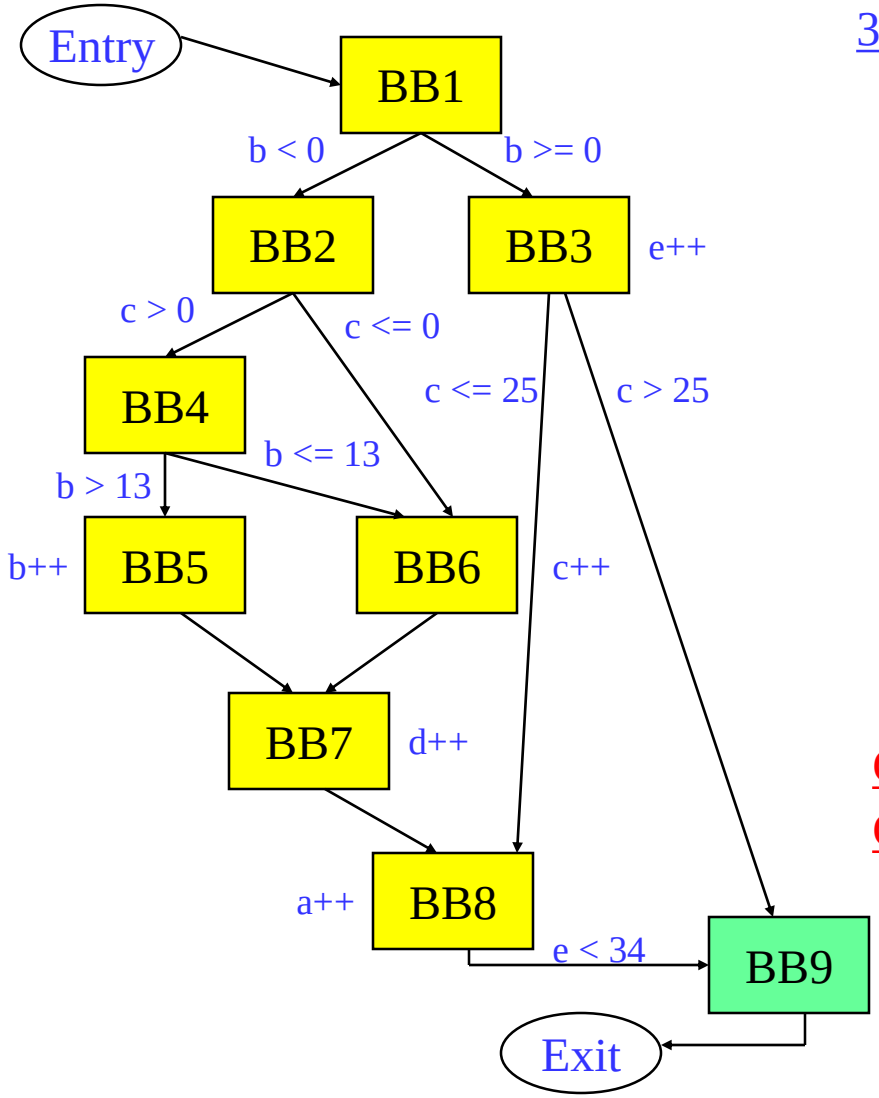


1 → 2 edge (aka –1)

x = 1
e = taken edge 1 → 2
y = 2
y not in pdom(x)
lub = 9
x_id = -1
t = 2

2 != 9 cd(2) += -1 t = 7
7 != 9 cd(7) += -1 t = 8
8 != 9 cd(8) += -1 t = 9
9 == 9

Running Example – CDs Via Algorithm (2)



3 → 8 edge (aka -3)

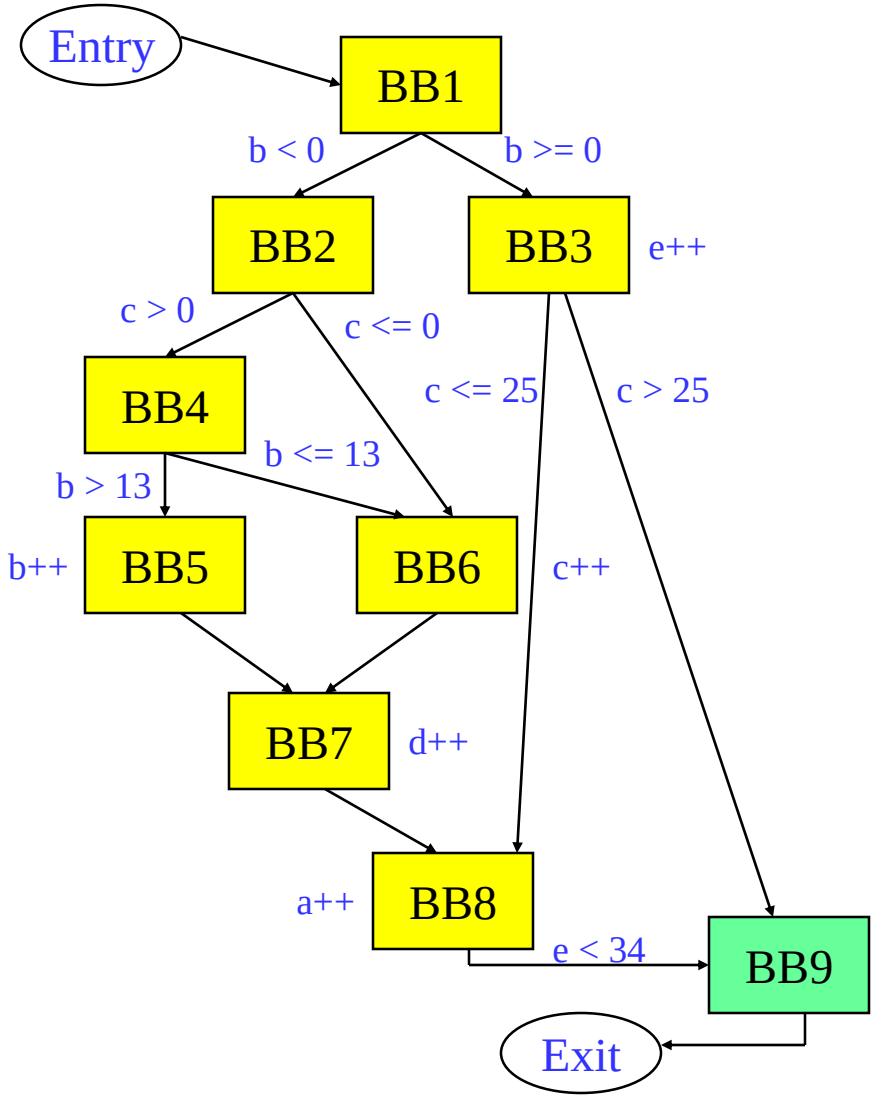
x = 3
e = taken edge 3 → 8
y = 8
y not in pdom(x)
lub = 9
x_id = -3
t = 8

8 != 9 cd(8) += -3 t = 9
9 == 9

Class ProblemA: 1 → 3 edge (aka 1)

Class ProblemB: 7 → 8 edge (aka -7)

Running Example – CDs Via Algorithm (3)



Control deps (left is taken)

BB1: none

BB2: -1

BB3: 1

BB4: -2

BB5: -4

BB6: 2, 4

BB7: -1

BB8: -1, -3

BB9: none

Step 3: Control Flow Substitution

- Go from branching code → sequential predicated code
- 5 baby steps
  - 1. Create predicates
  - 2. CMPP insertion
  - 3. Guard operations
  - 4. Remove branches
  - 5. Initialize predicates

Predicate Creation

- R/K calculation – Mapping predicates to blocks
  - Paper more complicated than it really is
  - K = unique sets of control dependences
  - Create a new predicate for each element of K
  - R(bb) = predicate that represents CD set for bb, ie the bb's assigned predicate (all ops in that bb guarded by R(bb))

K = {{-1}, {1}, {-2}, {-4}, {2,4}, {-1,-3}}
predicates = p1, p2, p3, p4, p5, p6

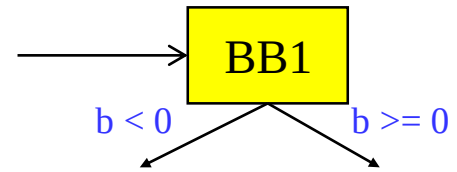
bb	=	1	2	3	4	5	6	7	8	9
CD(bb)	=	{none}	{-1}	{1}	{-2}	{-4}	{2,4}	{-1}	{-1,-3}	{none}
R(bb)	=	T	p1	p2	p3	p4	p5	p1	p6	T

CMPP Creation/Insertion

- For each control dependence set
  - For each edge in the control dependence set
    - Identify branch condition that causes edge to be traversed
    - Create CMPP to compute corresponding branch condition
      - OR-type – handles worst case
      - guard = True
      - destination = predicate assigned to that CD set
      - Insert at end of BB that is the source of the edge

K = {{-1}, {1}, {-2}, {-4}, {2,4}, {-1,-3}}
predicates = p1, p2, p3, p4, p5, p6

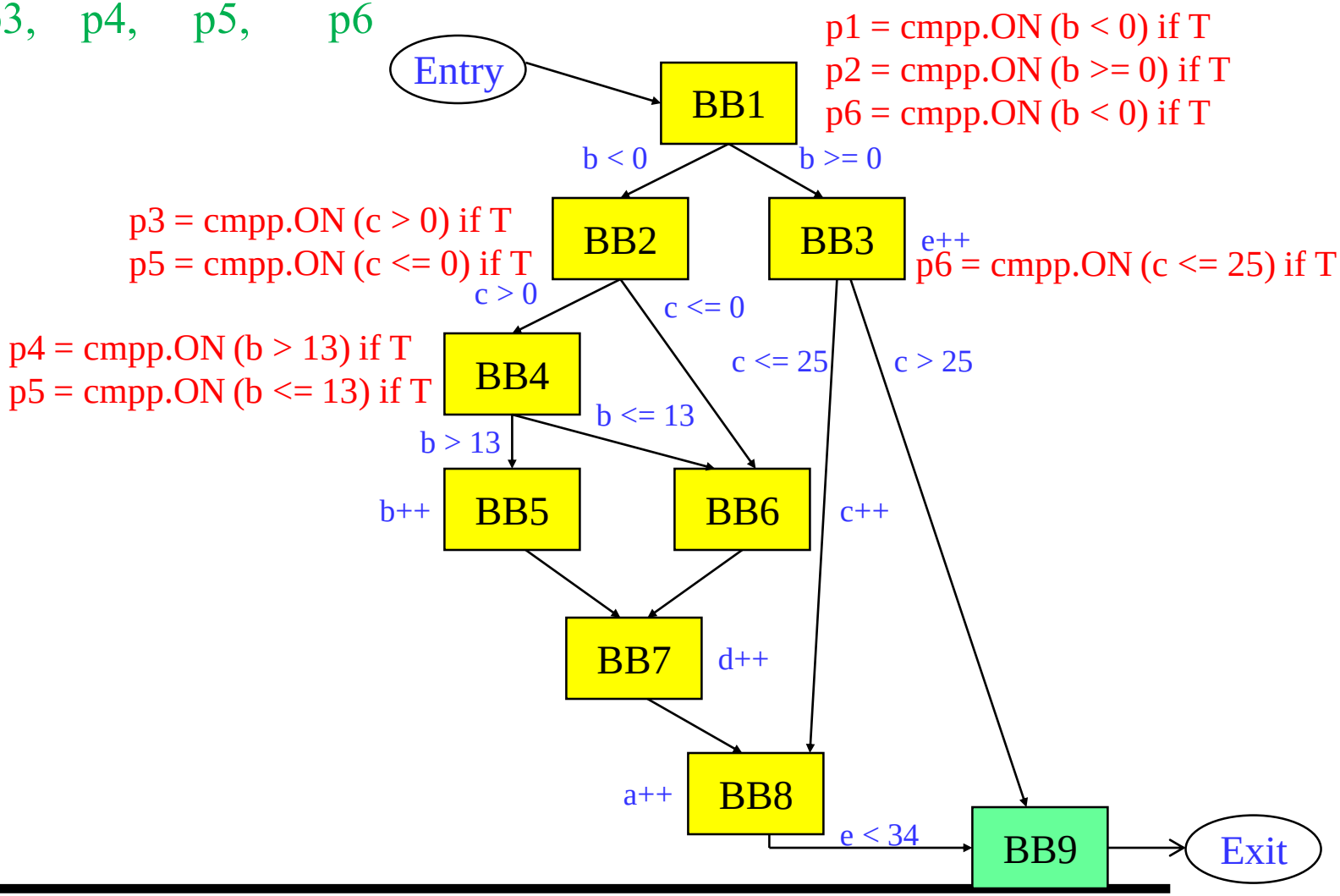
Example: p1 = cmpp.ON (b < 0) if T



Running Example – CMPP Creation

K = {{-1}, {1}, {-2}, {-4}, {2,4}, {-1,-3}}

p's = p1, p2, p3, p4, p5, p6

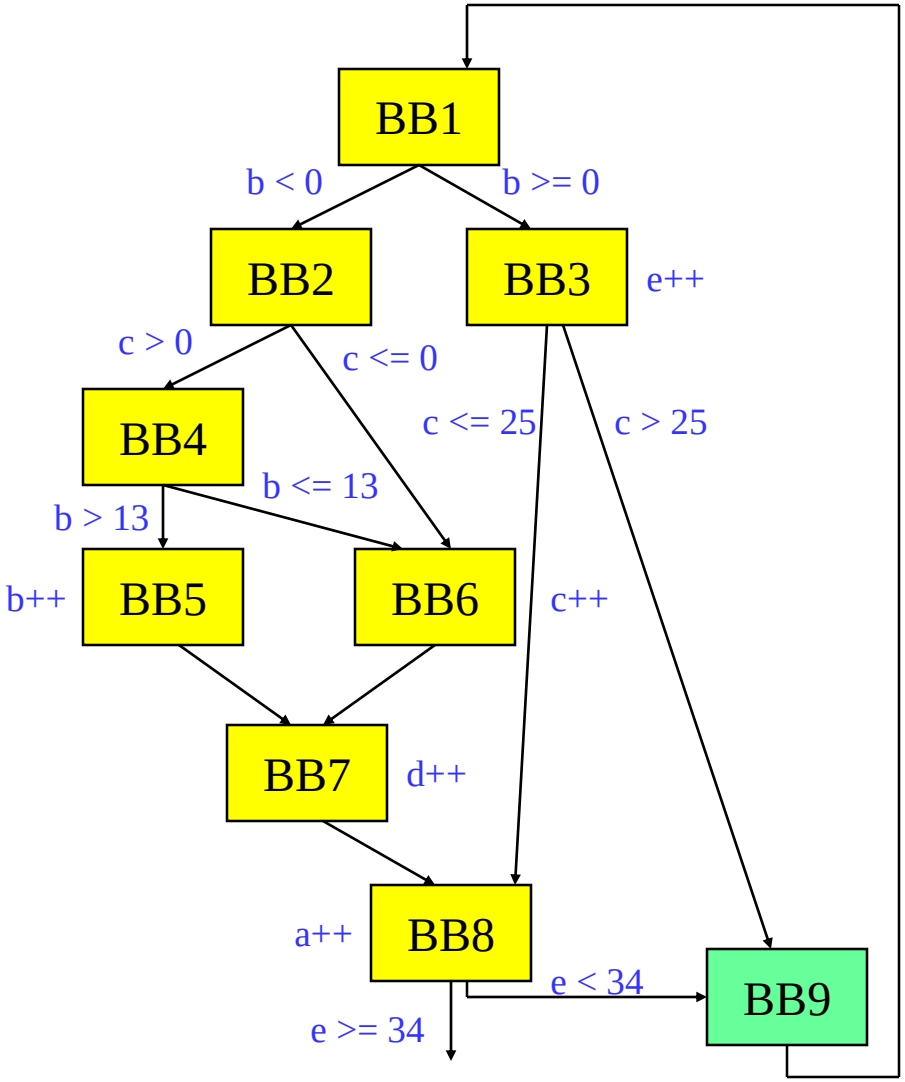


Control Flow Substitution – The Rest

- Guard all operations in each bb by R(bb)
  - » Including the newly inserted CMPPs
- Nuke all the branches
  - » Except exit edges and backedges
- Initialize each predicate to 0 in first BB

bb	=	1,	2,	3,	4,	5,	6,	7,	8,	9
CD(bb)	=	{{none},	{-1},	{1},	{-2},	{-4},	{2,4},	{-1},	{-1,-3},	{none}
R(bb)	=	T	p1	p2	p3	p4	p5	p1	p6	T

Running Example – Control Flow Substitution



Loop:

p1 = p2 = p3 = p4 = p5 = p6 = 0
b = load(a) if T
p1 = cmpp.ON (b < 0) if T
p2 = cmpp.ON (b >= 0) if T
p6 = cmpp.ON (b < 0) if T
p3 = cmpp.ON (c > 0) if p1
p5 = cmpp.ON (c <= 0) if p1
p4 = cmpp.ON (b > 13) if p3
p5 = cmpp.ON (b <= 13) if p3
b = b + 1 if p4
c = c + 1 if p5
d = d + 1 if p1
p6 = cmpp.ON (c <= 25) if p2
e = e + 1 if p2
a = a + 1 if p6
bge e, 34, Done if p6
jump Loop if T

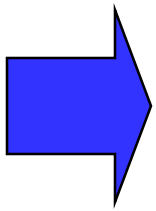
Done:

Step 4: CMPP Compaction

- Convert ON CMPPs to UN
  - All singly defined predicates don't need to be OR-type
  - OR of 1 condition → Just compute it !!!
  - Remove initialization (Unconditional don't require init)
- Reduce number of CMPPs
  - Utilize 2nd destination slot
  - Combine any 2 CMPPs with:
    - Same source operands
    - Same guarding predicate
    - Same or opposite compare conditions

Running Example - CMPP Compaction

```
Loop:
   p1 = p2 = p3 = p4 = p5 = p6 = 0
   b = load(a) if T
   p1 = cmpp.ON (b < 0) if T
   p2 = cmpp.ON (b >= 0) if T
   p6 = cmpp.ON (b < 0) if T
   p3 = cmpp.ON (c > 0) if p1
   p5 = cmpp.ON (c <= 0) if p1
   p4 = cmpp.ON (b > 13) if p3
   p5 = cmpp.ON (b <= 13) if p3
   b = b + 1 if p4
   c = c + 1 if p5
   d = d + 1 if p1
   p6 = cmpp.ON (c <= 25) if p2
   e = e + 1 if p2
   a = a + 1 if p6
   bge e, 34, Done if p6
   jump Loop if T
Done:
```



```
Loop:
   p5 = p6 = 0
   b = load(a) if T
   p1,p2 = cmpp.UN.UC (b < 0) if T
   p6 = cmpp.ON (b < 0) if T
   p3,p5 = cmpp.UN.OC (c > 0) if p1
   p4,p5 = cmpp.UN.OC (b > 13) if p3
   b = b + 1 if p4
   c = c + 1 if p5
   d = d + 1 if p1
   p6 = cmpp.ON (c <= 25) if p2
   e = e + 1 if p2
   a = a + 1 if p6
   bge e, 34, Done if p6
   jump Loop if T
Done:
```

Homework Problem – Answer Next Time

```
if (a > 0) {
    r = t + s
    if (b > 0 || c > 0)
        u = v + 1
    else if (d > 0)
        x = y + 1
    else
        z = z + 1
}
```

a. Draw the CFG
b. Compute CD
c. If-convert the code