

# Control Flow II: Dominators, Loop Detection

---

EECS 483 – Lecture 20

University of Michigan

Wednesday, November 15, 2006

# Announcements and Reading

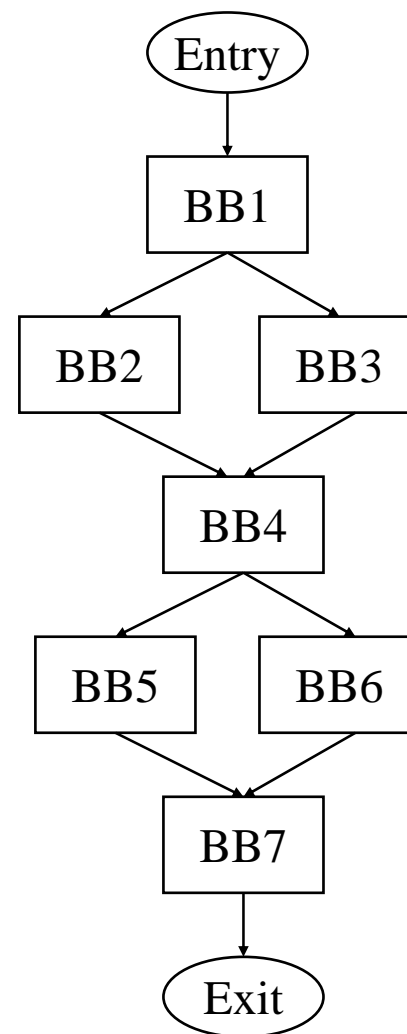
---

- ❖ Simon's office hours on Thurs (11/16)
  - » Moved to 10am-12pm, 4817 CSE
  - » Exam 1 problem 11 was incorrectly graded
    - Follow set for B should contain z
  - » See Simon to get your points back
- ❖ Project 3/4 – postponed until Monday
  - » Group formation for Project 3/4
- ❖ Today's class material:
  - » 10.4, end of 10.9 (dominator algorithm)

# From Last Time: Control Flow Graph (CFG)

---

- ❖ Defn Control Flow Graph –  
Directed graph,  $G = (V, E)$   
where each vertex  $V$  is a basic block and there is an edge  $E$ ,  
 $v_1 (BB1) \rightarrow v_2 (BB2)$  if  $BB2$   
can immediately follow  $BB1$   
in some execution sequence
  - » A BB has an edge to all blocks it can branch to
  - » Standard representation used by many compilers
  - » Often have 2 pseudo vertices
    - entry node
    - exit node



# Control Flow Analysis

---

- ❖ Determining properties of the program branch structure
  - » Static properties → Not executing the code
  - » Properties that exist regardless of the run-time branch directions
  - » Use CFG
  - » Optimize efficiency of control flow structure
- ❖ Determine instruction execution properties
  - » Global optimization of computation operations
  - » Discuss this later

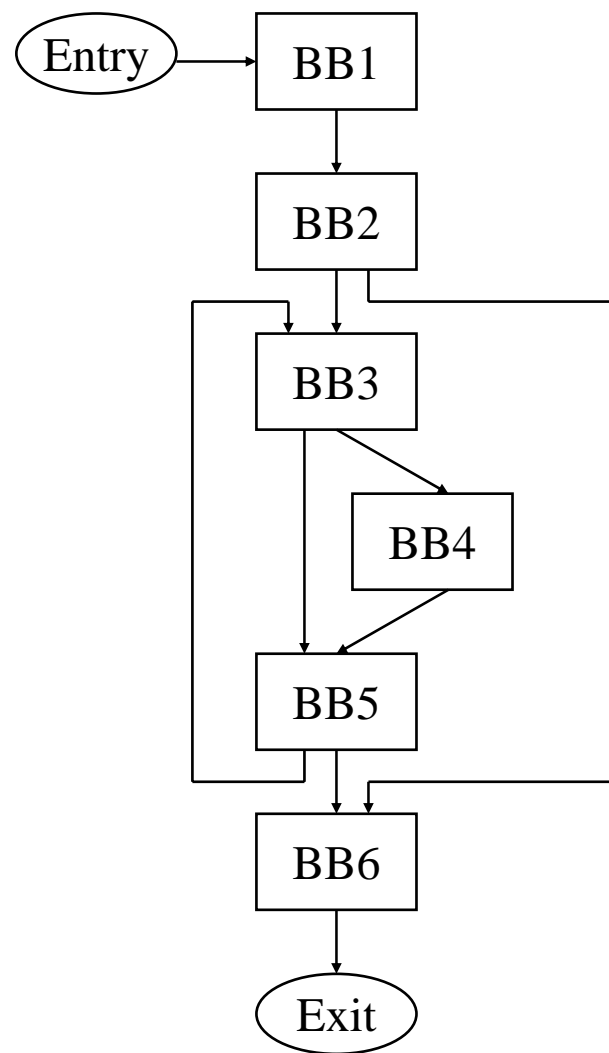
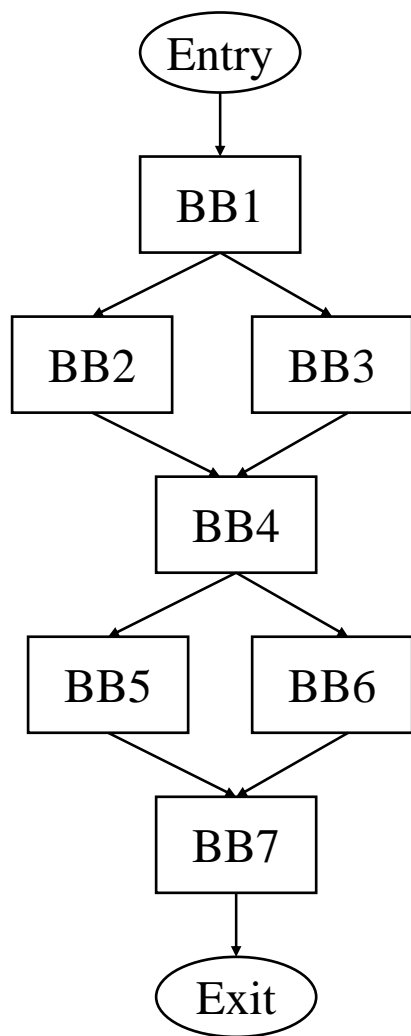
# Dominator

---

- ❖ **Defn: Dominator** – Given a CFG( $V, E, \text{Entry}, \text{Exit}$ ), a node  $x$  dominates a node  $y$ , if every path from the Entry block to  $y$  contains  $x$
- ❖ 3 properties of dominators
  - » Each BB dominates itself
  - » If  $x$  dominates  $y$ , and  $y$  dominates  $z$ , then  $x$  dominates  $z$
  - » If  $x$  dominates  $z$  and  $y$  dominates  $z$ , then either  $x$  dominates  $y$  or  $y$  dominates  $x$
- ❖ **Intuition**
  - » Given some BB, which blocks are guaranteed to have executed prior to executing the BB

# Dominator Examples

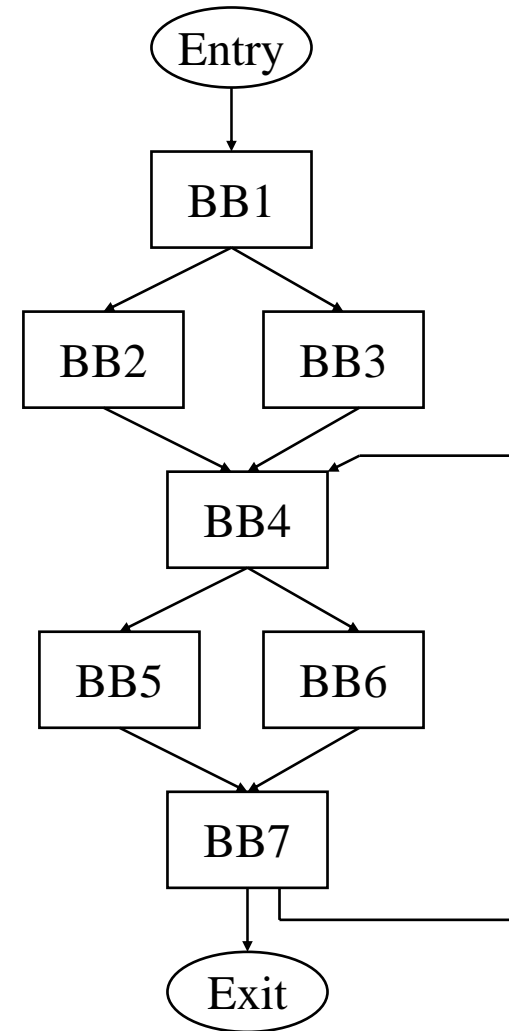
---



# Dominator Analysis

---

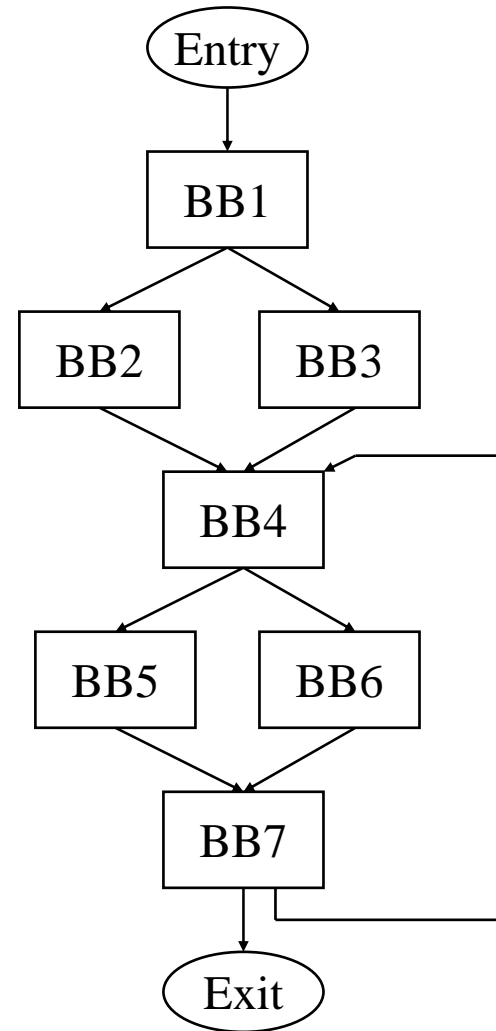
- ❖ Compute  $\text{dom}(\text{BB}_i)$  = set of BBs that dominate  $\text{BB}_i$
- ❖ Initialization
  - »  $\text{Dom}(\text{entry}) = \text{entry}$
  - »  $\text{Dom}(\text{everything else}) = \text{all nodes}$
- ❖ Iterative computation
  - » while change, do
    - change = false
    - for each BB (except the entry BB)
      - ◆  $\text{tmp}(\text{BB}) = \text{BB} + \{\text{intersect of Dom of all predecessor BB's}\}$
      - ◆ if ( $\text{tmp}(\text{BB}) \neq \text{dom}(\text{BB})$ )
        - ↓  $\text{dom}(\text{BB}) = \text{tmp}(\text{BB})$
        - ↓ change = true



# Immediate Dominator

---

- ❖ Defn: Immediate dominator (idom)– Each node  $n$  has a unique immediate dominator  $m$  that is the last dominator of  $n$  on any path from the initial node to  $n$ 
  - » Closest node that dominates

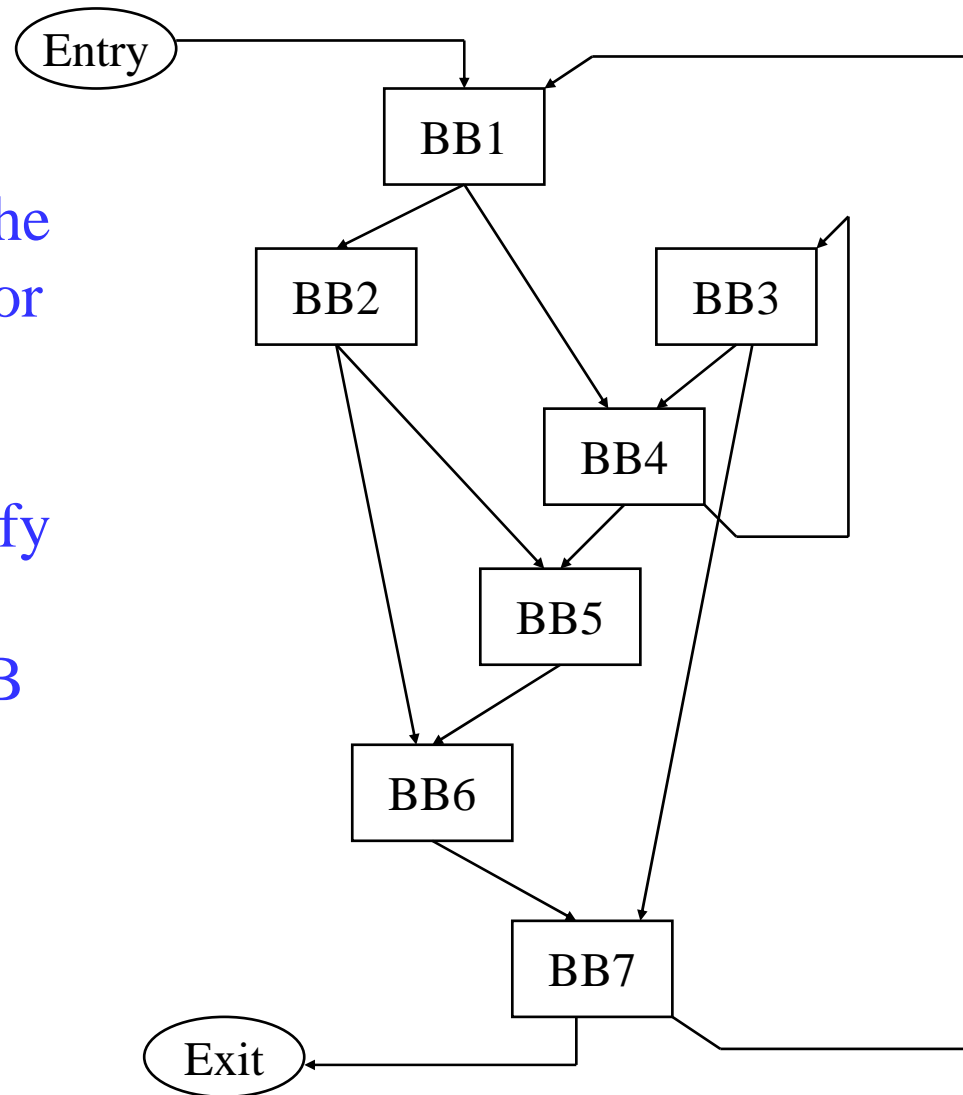


# Class Problem

---

Calculate the  
DOM set for  
each BB

Also identify  
the iDOM  
for each BB



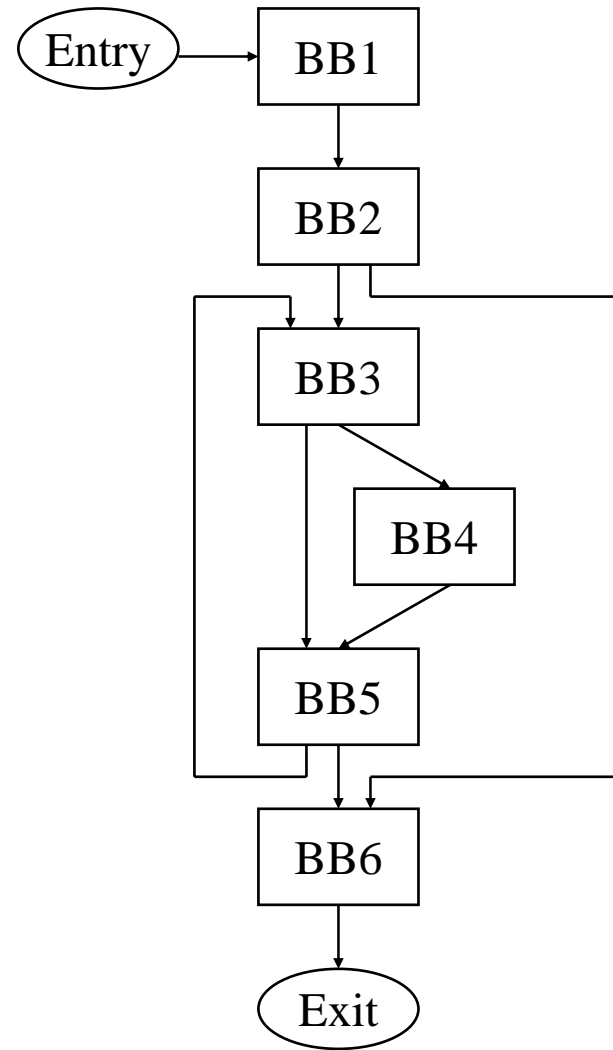
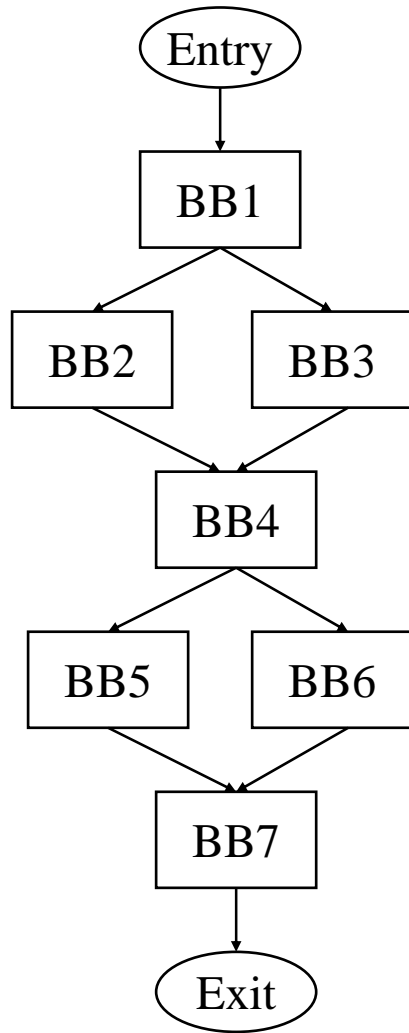
# Post Dominator

---

- ❖ Reverse of dominator
- ❖ Defn: Post Dominator – Given a CFG( $V$ ,  $E$ , Entry, Exit), a node  $x$  post dominates a node  $y$ , if every path from  $y$  to the Exit contains  $x$
- ❖ Intuition
  - » Given some BB, which blocks are guaranteed to have executed after executing the BB

# Post Dominator Examples

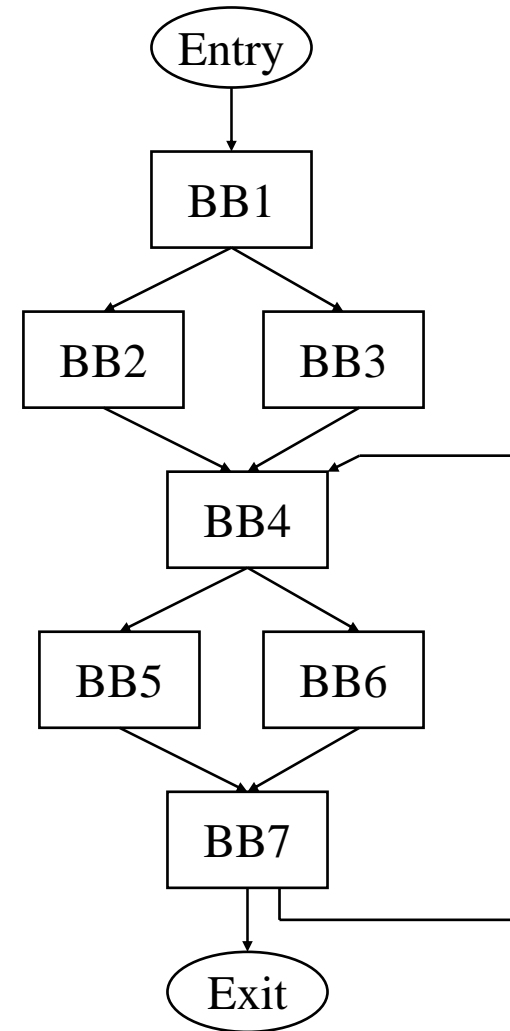
---



# Post Dominator Analysis

---

- ❖ Compute  $\text{pdom}(\text{BB}_i) = \text{set of BBs that post dominate BB}_i$
- ❖ Initialization
  - »  $\text{Pdom}(\text{exit}) = \text{exit}$
  - »  $\text{Pdom}(\text{everything else}) = \text{all nodes}$
- ❖ Iterative computation
  - » while change, do
    - change = false
    - for each BB (except the exit BB)
      - ◆  $\text{tmp}(\text{BB}) = \text{BB} + \{\text{intersect of pdom of all successor BB's}\}$
      - ◆ if ( $\text{tmp}(\text{BB}) \neq \text{pdom}(\text{BB})$ )
        - ↓  $\text{pdom}(\text{BB}) = \text{tmp}(\text{BB})$
        - ↓ change = true

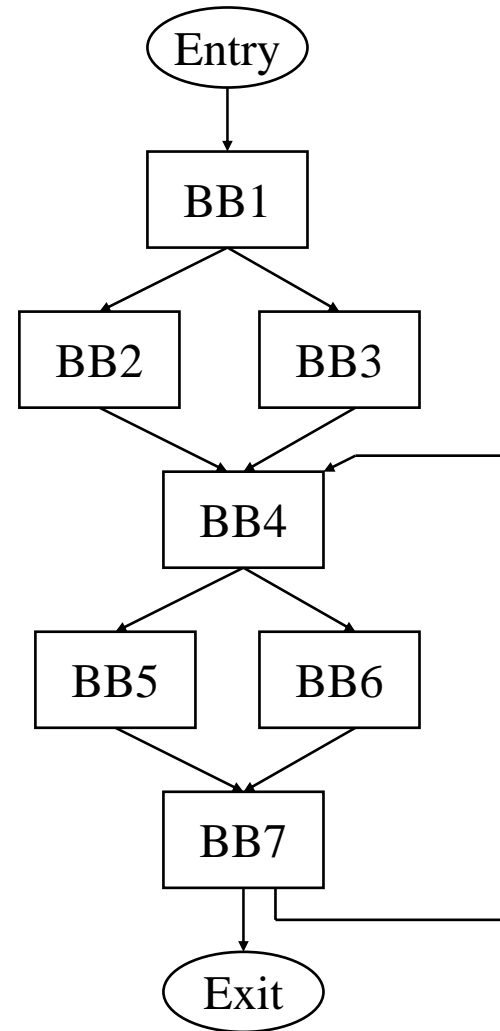


# Immediate Post Dominator

---

❖ Defn: Immediate post dominator (ipdom) –  
Each node  $n$  has a unique immediate post dominator  $m$  that is the first post dominator of  $n$  on any path from  $n$  to the Exit

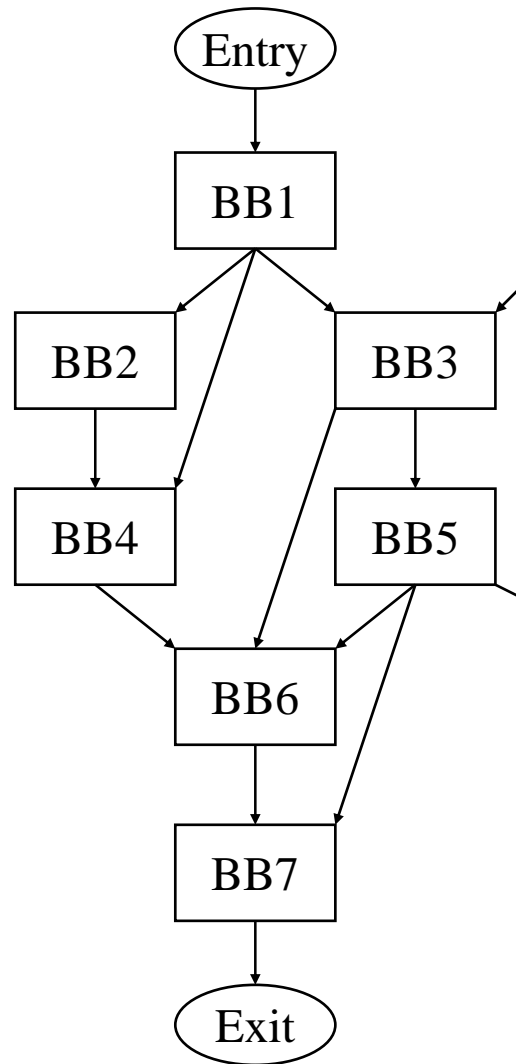
- » Closest node that post dominates
- » First breadth-first successor that post dominates a node



# Class Problem

---

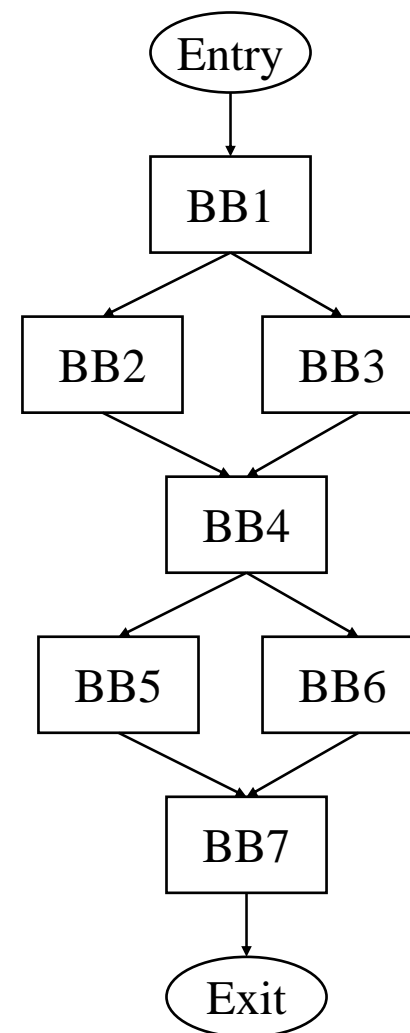
Calculate the  
PDOM set for  
each BB



# Why Do We Care About Dominators?

---

- ❖ Loop detection – next subject
- ❖ Dominator
  - » Guaranteed to execute before
  - » Redundant computation – an op can only be redundant if it is computed in a dominating BB
  - » Most global optimizations use dominance info
- ❖ Post dominator
  - » Guaranteed to execute after
  - » Make a guess (ie 2 pointers do not point to the same locn)
  - » Check they really do not point to one another in the post dominating BB



# Natural Loops

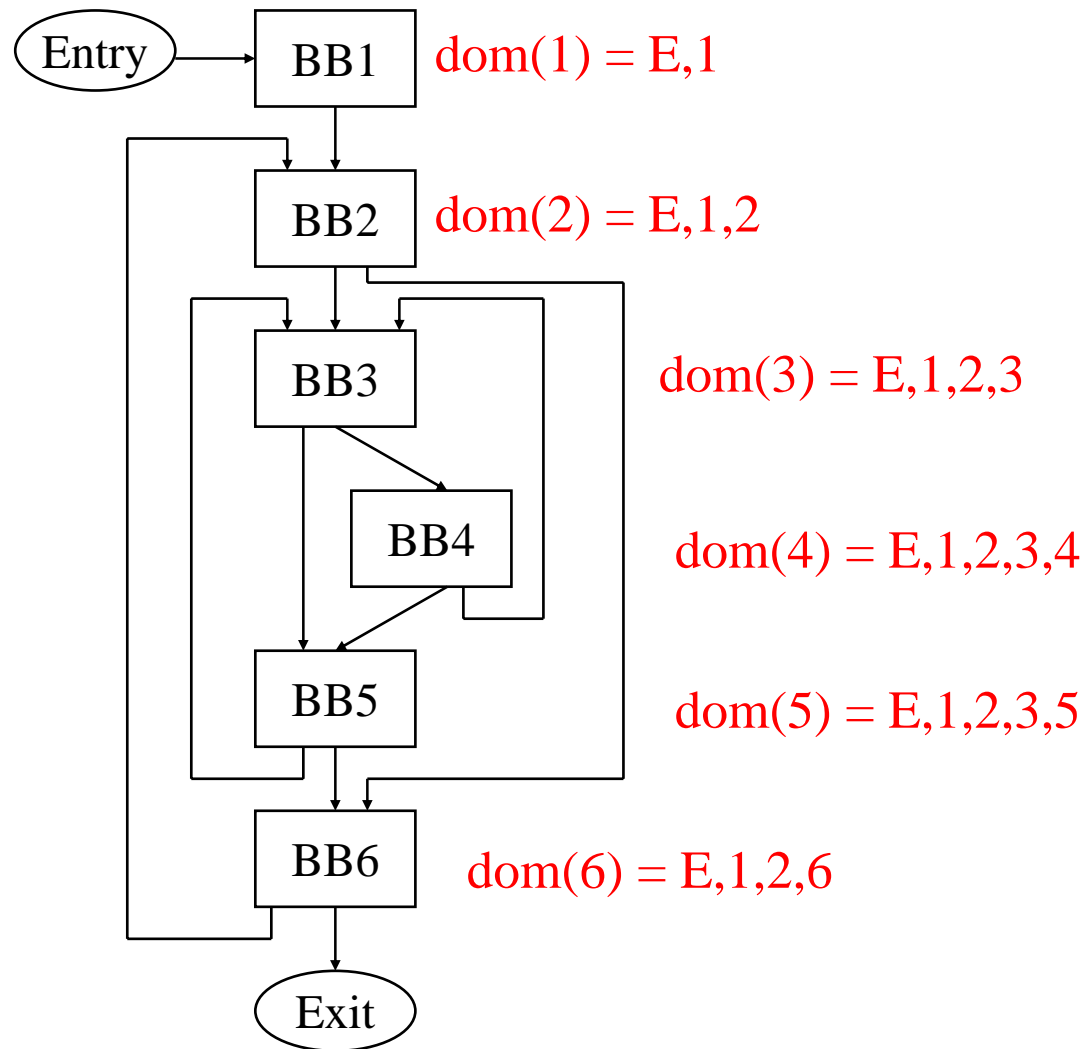
---

- ❖ Cycle suitable for optimization
  - » Discuss opti later
- ❖ 2 properties:
  - » Single entry point called the header
    - Header dominates all blocks in the loop
  - » Must be one way to iterate the loop (ie at least 1 path back to the header from within the loop) called a backedge
- ❖ Backedge detection
  - » Edge,  $x \rightarrow y$  where the target (y) dominates the source (x)

# Backedge Example

BE = target dominates  
source

- E → 1 : No
- 1 → 2 : No
- 2 → 3 : No
- 2 → 6 : No
- 3 → 4 : No
- 3 → 5 : No
- 4 → 3 : Yes
- 4 → 5 : No
- 5 → 3 : Yes
- 5 → 6 : No
- 6 → 2 : Yes
- 6 → X : No



In this example, BE = edge from higher BB to lower BB, not always this easy!

# Loop Detection

---

- ❖ Identify all backedges using dominance info
- ❖ Each backedge ( $x \rightarrow y$ ) defines a loop
  - » Loop header is the backedge target ( $y$ )
  - » Loop BB – basic blocks that comprise the loop
    - All predecessor blocks of  $x$  for which control can reach  $x$  without going through  $y$  are in the loop
- ❖ Merge loops with the same header
  - » I.e., a loop with 2 continues
  - »  $\text{LoopBackedge} = \text{LoopBackedge1} + \text{LoopBackedge2}$
  - »  $\text{LoopBB} = \text{LoopBB1} + \text{LoopBB2}$
- ❖ Important property
  - » Header dominates all LoopBB

# Loop Detection Example

Loop detection: 3 steps:

- Identify backedges
- Compute LoopBB
- Merge loops with the same header

Loop1: defined by  $6 \rightarrow 2$

LoopBB = 2,3,4,5,6

Loop2: defined by  $4 \rightarrow 3$

LoopBB = 3,4

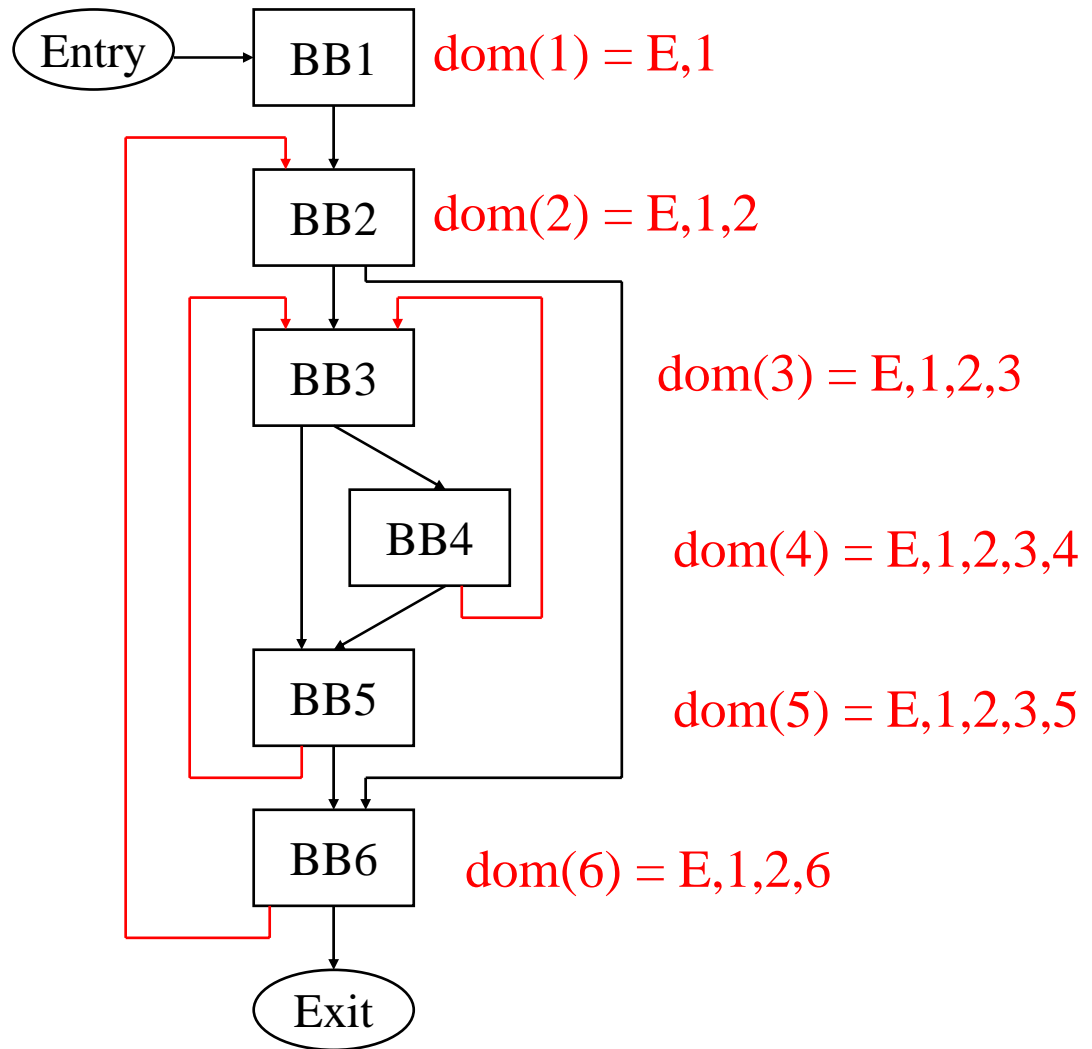
Loop3: defined by  $5 \rightarrow 3$

LoopBB = 3,4,5

Merge loops 2,3

LoopBB = 3,4,5

Backedges =  $4 \rightarrow 3$ ,  $5 \rightarrow 3$



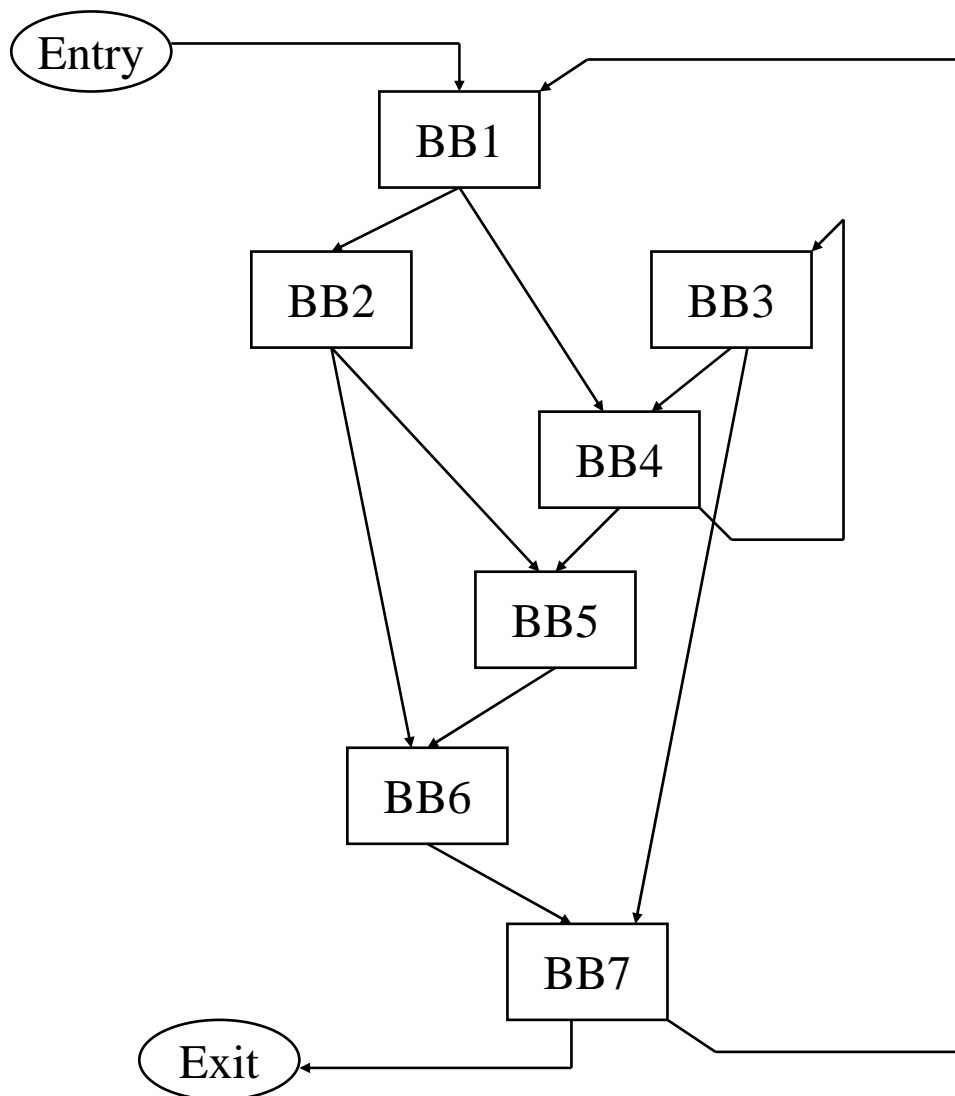
# Class Problem

---

Find the loops

What are the header(s)?

What are the backedge(s)?

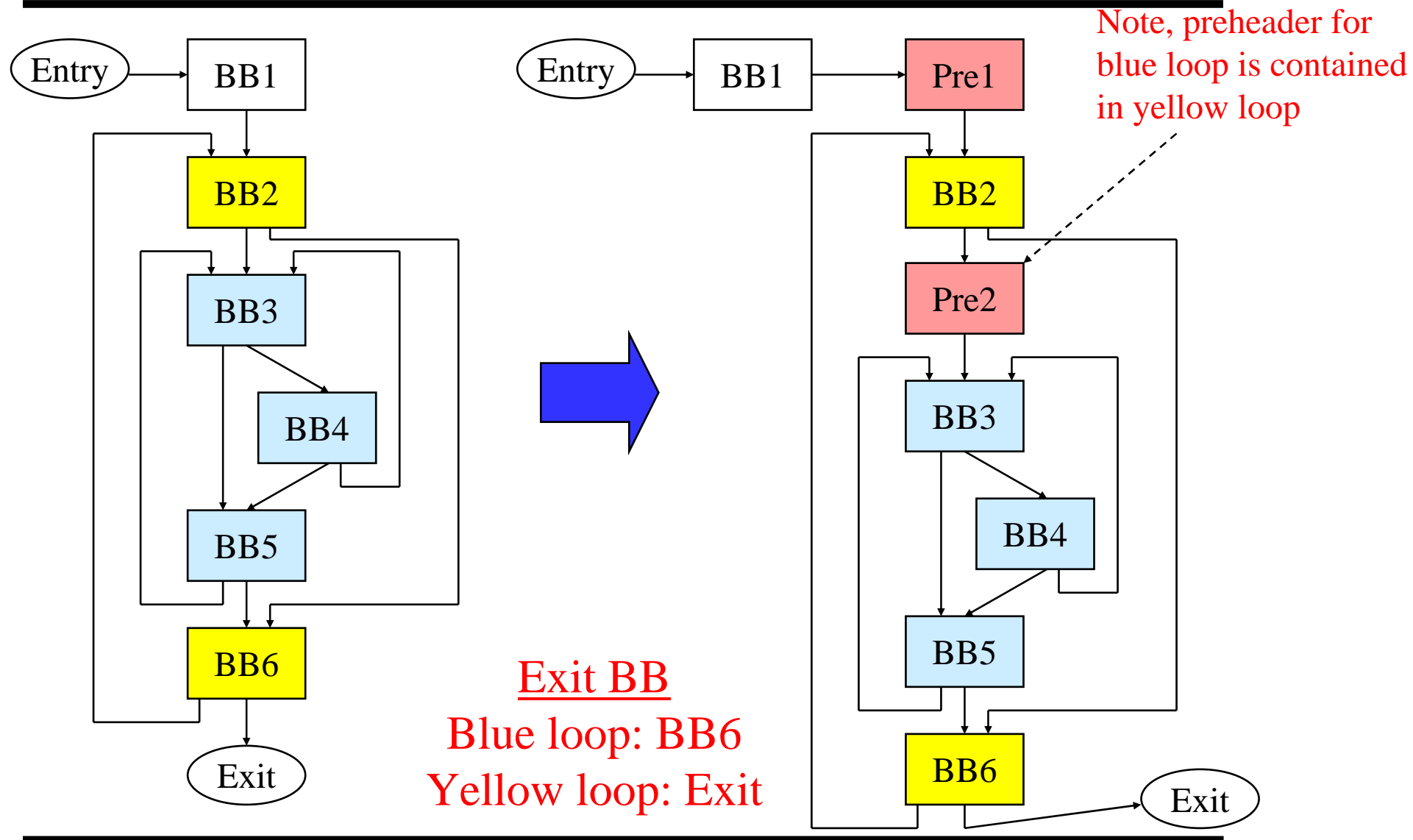


# Important Parts of a Loop

---

- ❖ Header, LoopBB
- ❖ Backedges, BackedgeBB
- ❖ Exitedges, ExitBB
  - » For each LoopBB, examine each outgoing edge
  - » If the edge is to a BB not in LoopBB, then its an exit
- ❖ Preheader (Preloop)
  - » New block before the header (falls through to header)
  - » Whenever you invoke the loop, preheader executed
  - » Whenever you iterate the loop, preheader NOT executed
  - » All edges entering header
    - Backedges – no change, All others - retarget to preheader
- ❖ Postheader (Postloop) - analogous

# ExitBB/Preheader Example



# Characteristics of a Loop

---

- ❖ **Nesting** (generally within a procedure scope)
  - » Inner loop – Loop with no loops contained within it
  - » Outer loop – Loop contained within no other loops
  - » Nesting depth
    - $\text{depth}(\text{outer loop}) = 1$
    - $\text{depth} = \text{depth}(\text{parent or containing loop}) + 1$
- ❖ **Trip count** (average trip count)
  - » How many times (on average) does the loop iterate
  - » `for (I=0; I<100; I++)` → trip count = 100
  - » Ave trip count =  $\text{weight}(\text{header}) / \text{weight}(\text{preheader})$

# Trip Count Calculation Example

Calculate the trip counts for all the loops in the graph

Blue loop:

$$\begin{aligned}w(\text{header}) &= w(\text{BB3}) \\ &= 1240 + 60 + 700 = 2000\end{aligned}$$

$$\begin{aligned}w(\text{preheader}) &= w(\text{BB2}) \\ &= 60 \text{ ( why not 100???)}\end{aligned}$$

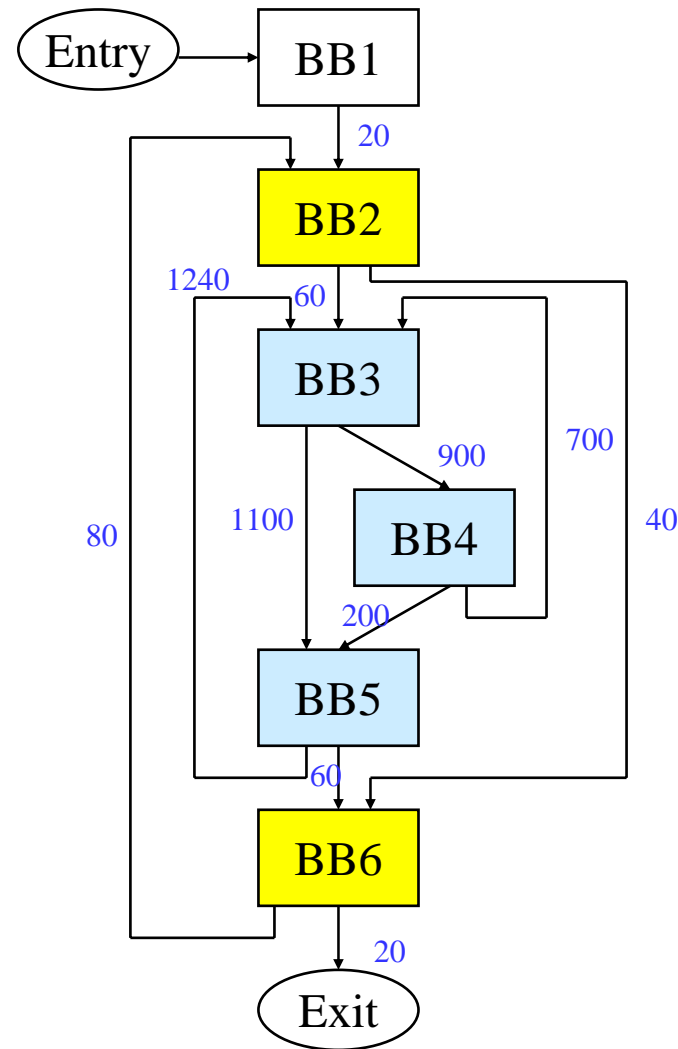
$$\text{avg trip count} = 2000 / 60 = 33.3$$

Yellow loop:

$$\begin{aligned}w(\text{header}) &= w(\text{BB2}) \\ &= 80 + 20 = 100\end{aligned}$$

$$w(\text{preheader}) = w(\text{BB1}) = 20$$

$$\text{avg trip count} = 100 / 20 = 5$$



# Loop Induction Variables

---

- ❖ **Induction variables** are variables such that every time they change value, they are incremented/decremented by some constant
- ❖ **Basic induction variable** – induction variable whose only assignments within a loop are of the form  $j = j \pm C$ , where  $C$  is a constant
- ❖ **Primary induction variable** – basic induction variable that controls the loop execution (for  $i=0$ ;  $i<100$ ;  $i++$ ),  $i$  (virtual register holding  $i$ ) is the primary induction variable
- ❖ **Derived induction variable** – variable that is a linear function of a basic induction variable

# Class Problem

---

Identify the basic, primary, and derived induction variables in this loop.

Loop:

r1 = 0 r7 = &A
r2 = r1 * 4 r4 = r7 + 3 r7 = r7 + 1 r1 = load(r2) r3 = load(r4) r9 = r1 * r3 r10 = r9 >> 4 store (r10, r2) r1 = r1 + 4 blt r1 100 Loop

# Reducible Flow Graphs

---

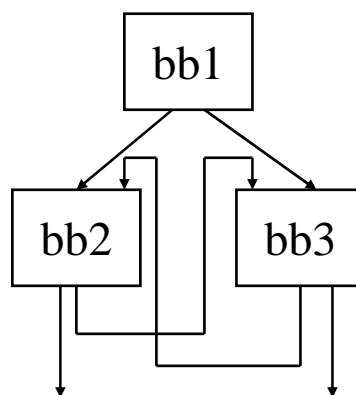
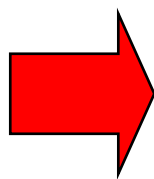
- ❖ A flow graph is reducible if and only if we can partition the edges into 2 disjoint groups often called forward and back edges with the following properties
  - » The forward edges form an acyclic graph in which every node can be reached from the Entry
  - » The back edges consist only of edges whose destinations dominate their sources
- ❖ More simply – Take a CFG, remove all the backedges ( $x \rightarrow y$  where  $y$  dominates  $x$ ), you should have a connected, acyclic graph

# Irreducible Flow Graph Example

---

- \* In C/C++, its not possible to create an irreducible flow graph without using goto's
- \* Cyclic graphs that are NOT natural loops cannot be optimized by the compiler

```
L1: x = x + 1
if (x) {
  L2: y = y + 1
  if (y > 10) goto L3
} else {
  L3: z = z + 1
  if (z > 0) goto L2
}
```



Non-reducible!