

EECS 583 – Class 2

Control Flow Analysis

University of Michigan

September 12, 2011

Announcements & Reading Material

- ❖ Daya's office hours held in 1695 CSE not 1620 CSE!
 - » Tue/Thu/Fri: 3-5pm
- ❖ See reading list on course webpage
 - » Will add to it each week
- ❖ Topic 1 – Control flow analysis and optimization
 - » Today's class
 - Ch 9.4, 10.4 from Red Dragon book (Aho, Sethi, Ullman)
 - » Next class
 - “Trace Selection for Compiling Large C Applications to Microcode”, Chang and Hwu, MICRO-21, 1988.
 - “The Superblock: An Effective Technique for VLIW and Superscalar Compilation”, Hwu et al., Journal of Supercomputing, 1993

Homework 1

- ❖ Available on course website
 - » <http://www.eecs.umich.edu/~mahlke/courses/583f11/homeworks.html>
- ❖ ~2 weeks to do – so get started soon!
 - » Due week from Friday (Sept 23)
- ❖ 3 4-core Linux machines available for class use
 - » Andrew, hugo, and wilma
 - » Need EECS account to access – See me if you do not have one
- ❖ Part 1 – Download and build llvm
 - » See <http://www.llvm.org>
- ❖ Part 2 - Collect some opcode, branch, and memory statistics
 - » Must figure out how to profile an application
 - » And extract profile info

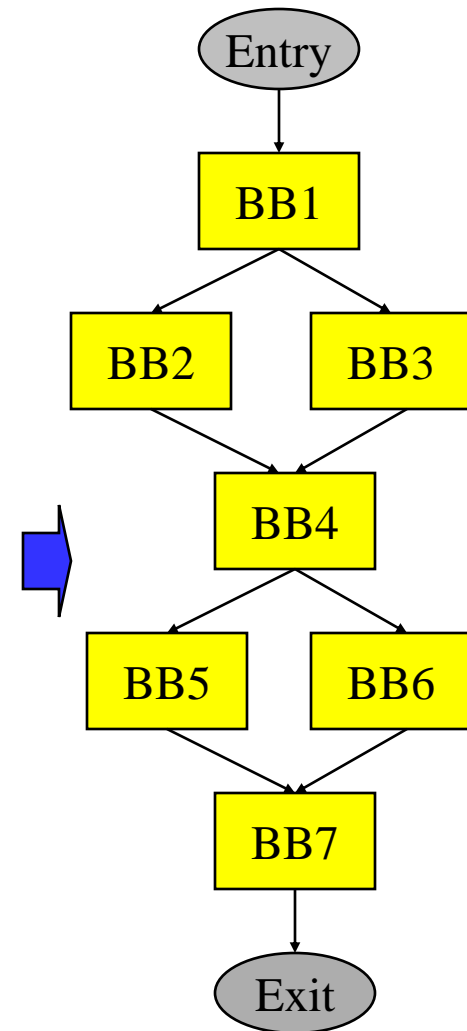
Homework 1 - Notes

- ❖ llvm-gcc already installed on hurricane machines
 - » `/y/583f11/llvm-gcc-install`
 - » Add `/y/583f11/llvm-gcc-install/bin` to your path so you can use the llvm-gcc executable
- ❖ If you want to use hurricane machines
 - » Create a directory for yourself on 1 of the machines
 - `cd /y/students`
 - `mkdir uniqueness`
 - `chmod 700 uniqueness`
 - `cd uniqueness`
 - **Don't accidentally put your llvm files in the `/y/students` directory!!**
 - » Then just download and build llvm backend
- ❖ If you use your other machines
 - » Then need to build both llvm-gcc and backend

From Last Time: BB and CFG

- ❖ Basic block – a sequence of consecutive operations in which flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end
- ❖ Control Flow Graph – Directed graph, $G = (V, E)$ where each vertex V is a basic block and there is an edge E , $v1 (BB1) \rightarrow v2 (BB2)$ if $BB2$ can immediately follow $BB1$ in some execution sequence

```
x = y+1;  
if (c)  
    x++;  
else  
    x--;;  
y = z + 1;  
if (a)  
    y++;  
else  
    y—;  
z++;
```

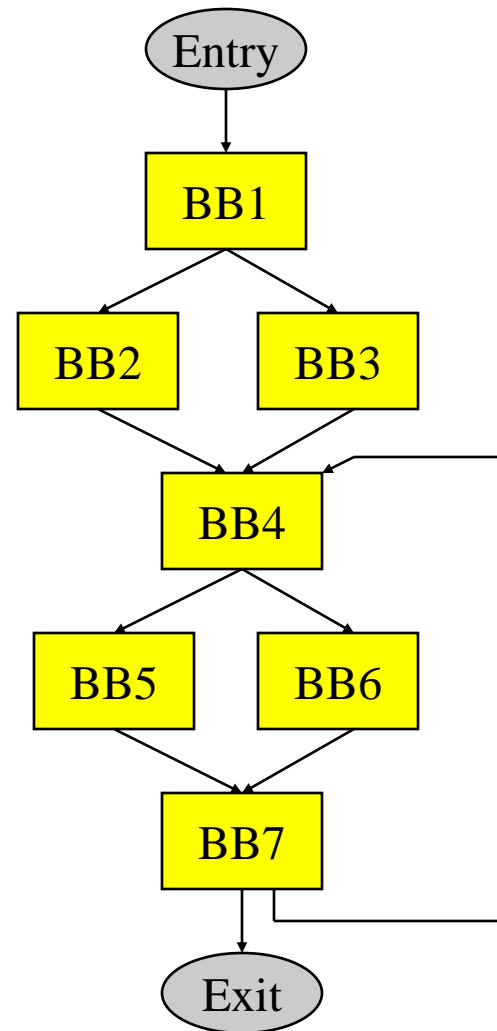


From Last Time: Dominator (DOM)

- ❖ 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

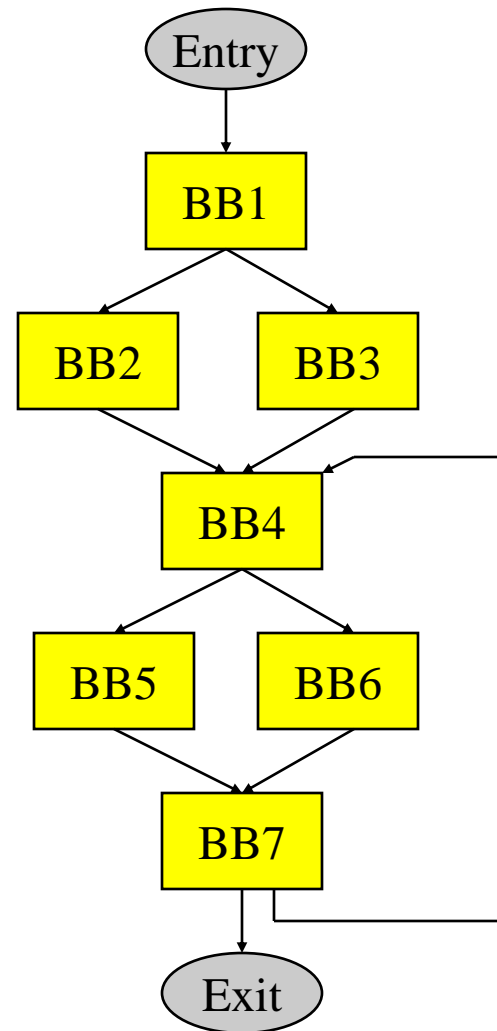
Dominator Analysis

- ❖ Compute $\text{dom}(\text{BB}_i)$ = set of BBs that dominate 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



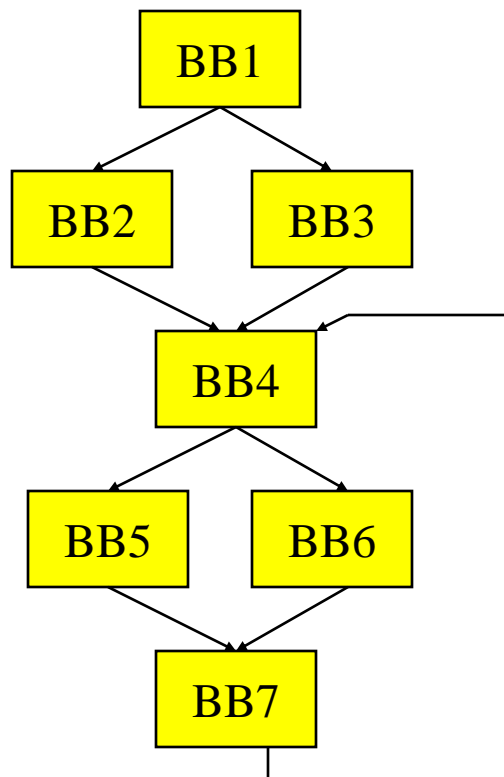
Immediate Dominator

- ❖ 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

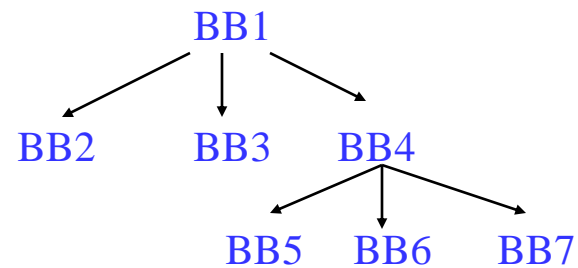


Dominator Tree

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



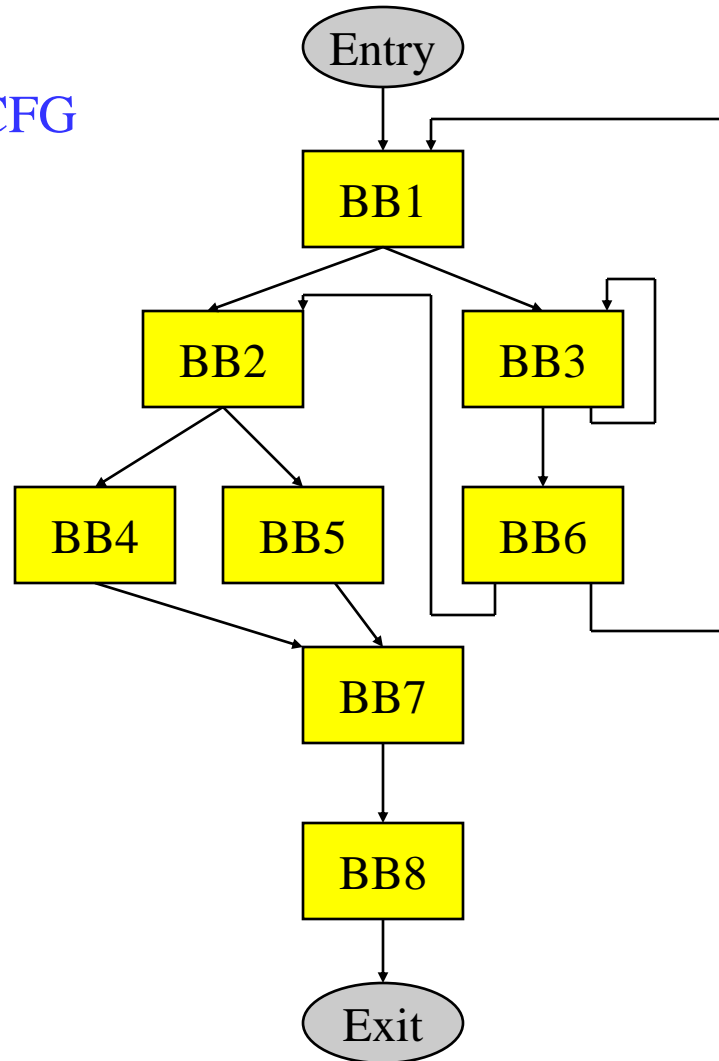
BB	DOM	BB	DOM
1	1	5	1,4,5
2	1,2	6	1,4,6
3	1,3	7	1,4,7
4	1,4		



Dom tree

Class Problem

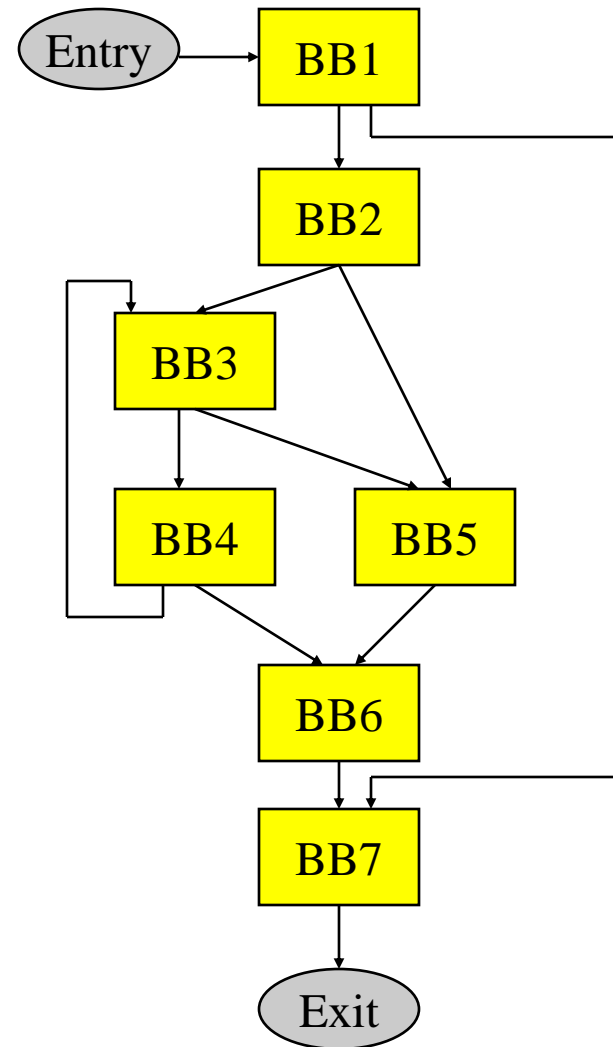
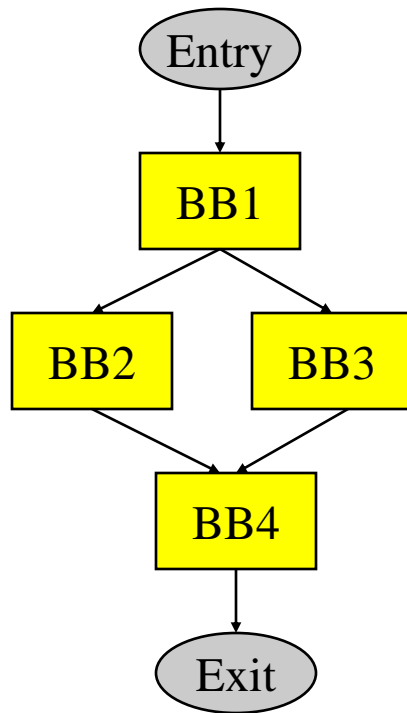
Draw the dominator
tree for the following CFG



Post Dominator (PDOM)

- ❖ Reverse of dominator
- ❖ Defn: Post Dominator –
Given a CFG($V, E, \text{Entry}, \text{Exit}$), 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
- ❖ $\text{pdom}(\text{BB}_i) = \text{set of BBs that post dominate BB}_i$
- ❖ Initialization
 - » $\text{Pdom}(\text{exit}) = \text{exit}$
 - » $\text{Pdom}(\text{everything else}) = \text{all nodes}$
- ❖ Iterative computation
 - » while change, do
 - $\text{change} = \text{false}$
 - for each BB (except the exit BB)
 - ♦ $\text{tmp}(\text{BB}) = \text{BB} + \{\text{intersect of pdom of all successor BB's}\}$
 - ♦ if $(\text{tmp}(\text{BB}) \neq \text{pdom}(\text{BB}))$
 $\text{pdom}(\text{BB}) = \text{tmp}(\text{BB})$
 $\text{change} = \text{true}$

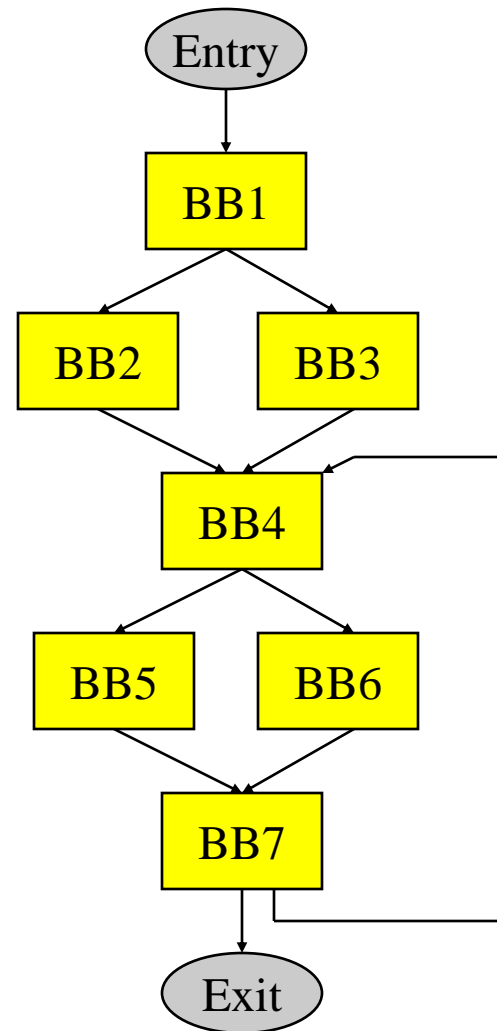
Post Dominator Examples



Immediate Post Dominator

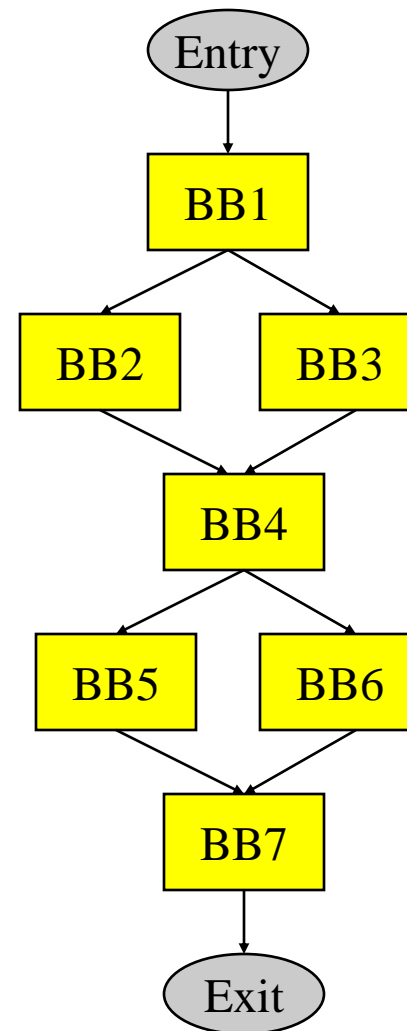
❖ Defn: Immediate post dominator (ipdom) –
Each node n has a unique immediate post dominator m that is the first post dominator of n on any path from n to the Exit

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



Why Do We Care About Dominators?

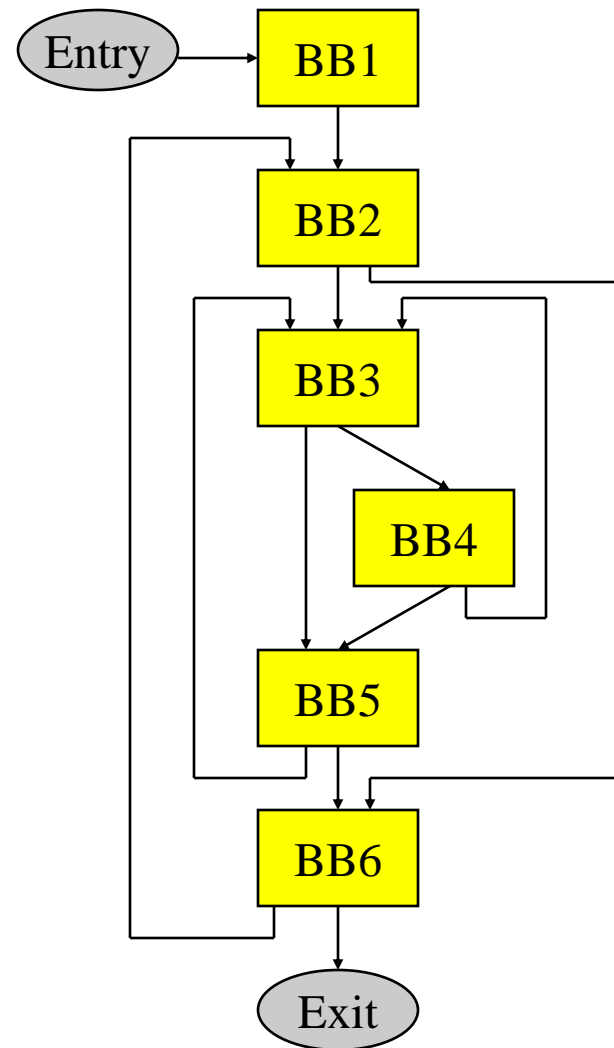
- ❖ Loop detection – next subject
- ❖ Dominator
 - » Guaranteed to execute before
 - » Redundant computation – an op is redundant if it is computed in a dominating BB
 - » Most global optimizations use dominance info
- ❖ Post dominator
 - » Guaranteed to execute after
 - » Make a guess (ie 2 pointers 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 optimizations 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)

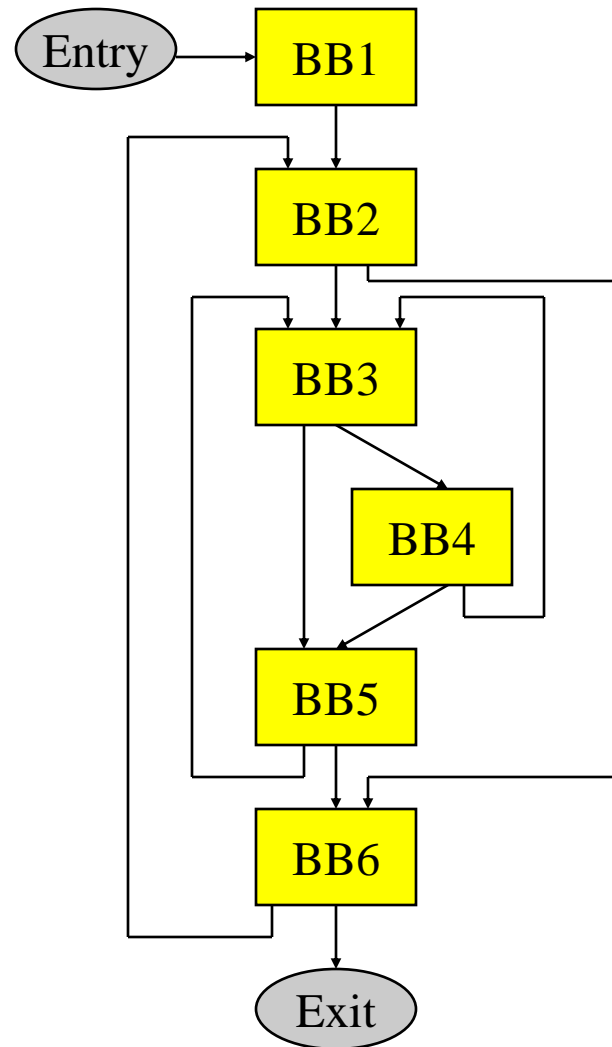
Backedge Example



Loop Detection

- ❖ Identify all backedges using Dom 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

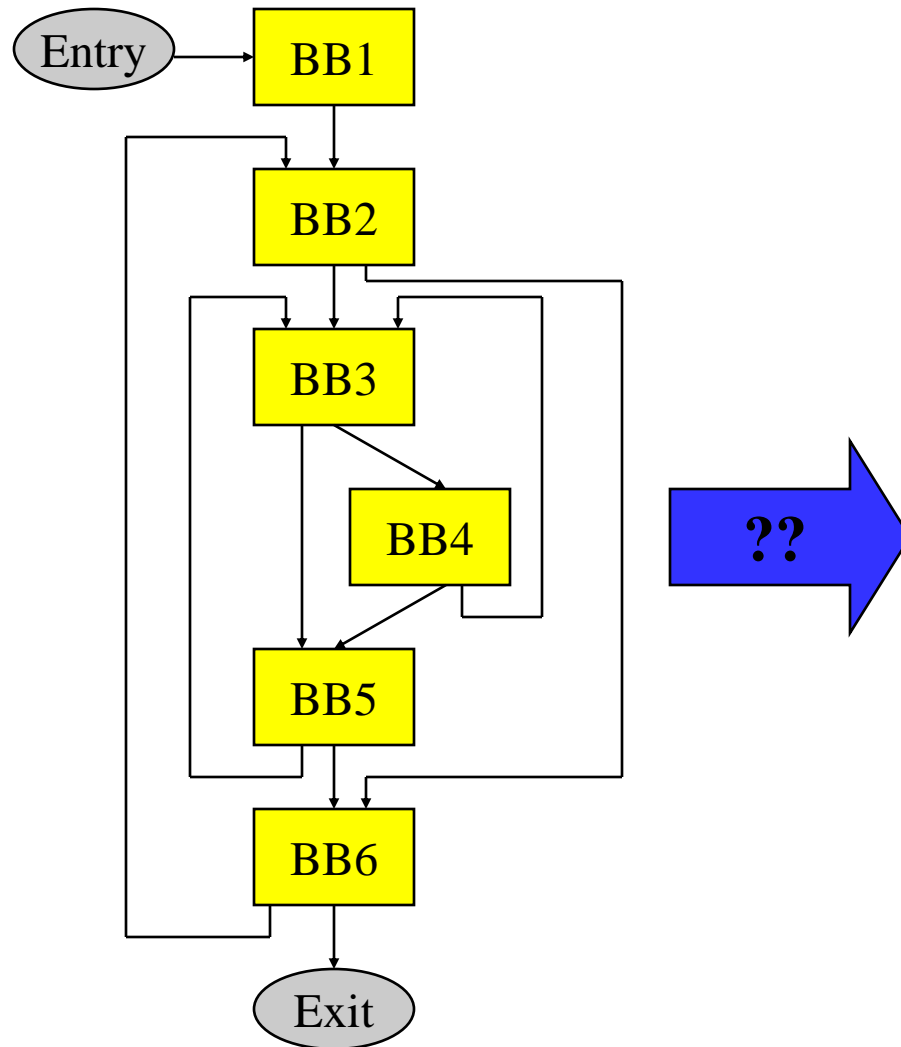
Loop Detection Example



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

Find the Preheaders for each Loop

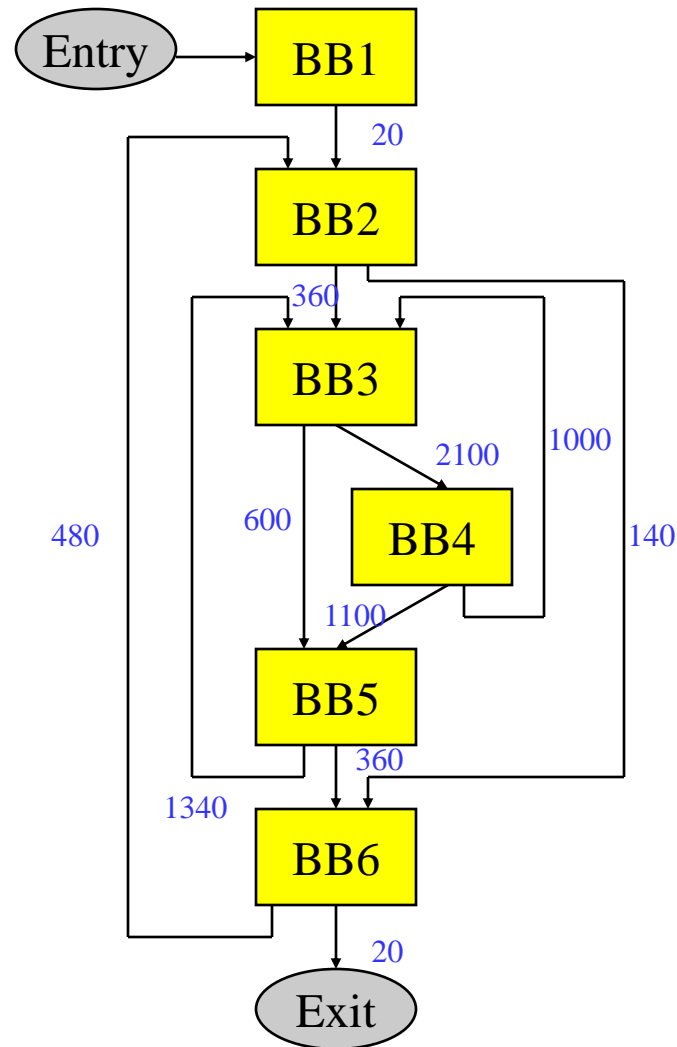


Characteristics of a Loop

- ❖ 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 = $\text{weight}(\text{header}) / \text{weight}(\text{preheader})$

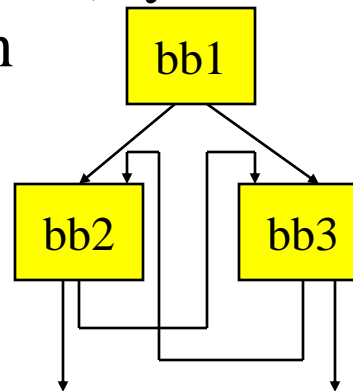
Trip Count Calculation Example

Calculate the trip counts for all the loops in the graph



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



Non-reducible!

Regions

- ❖ Region: A collection of operations that are treated as a single unit by the compiler
 - » Examples
 - Basic block
 - Procedure
 - Body of a loop
 - » Properties
 - Connected subgraph of operations
 - Control flow is the key parameter that defines regions
 - Hierarchically organized
- ❖ Problem
 - » Basic blocks are too small (3-5 operations)
 - Hard to extract sufficient parallelism
 - » Procedure control flow too complex for many compiler xforms
 - Plus only parts of a procedure are important (90/10 rule)

Regions (2)

❖ Want

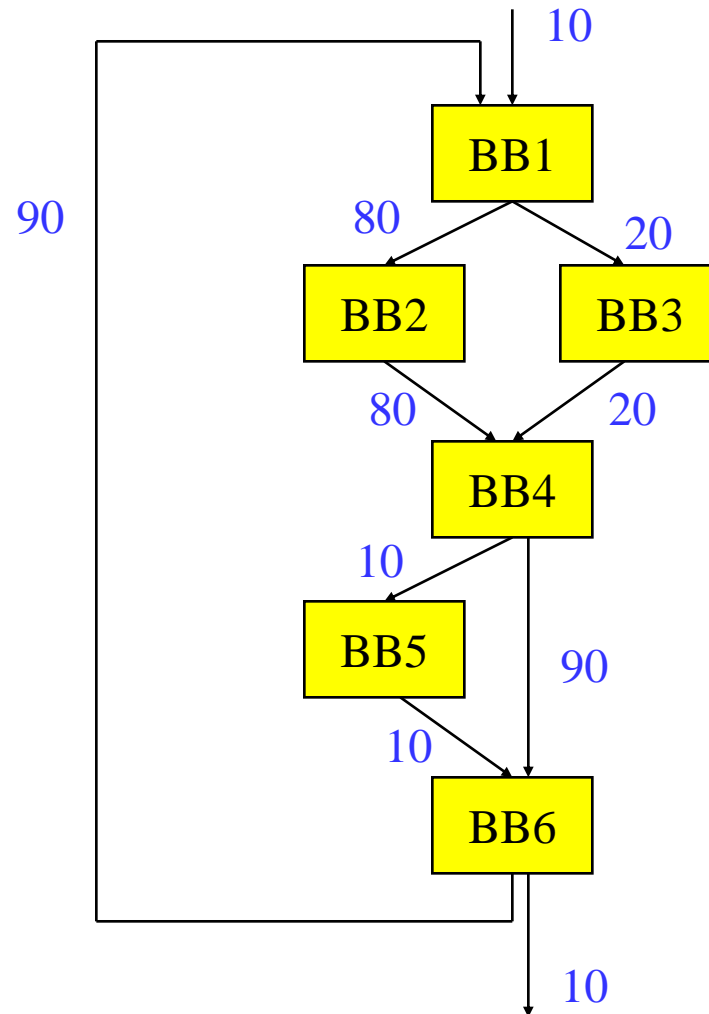
- » Intermediate sized regions with simple control flow
- » Bigger basic blocks would be ideal !!
- » Separate important code from less important
- » Optimize frequently executed code at the expense of the rest

❖ Solution

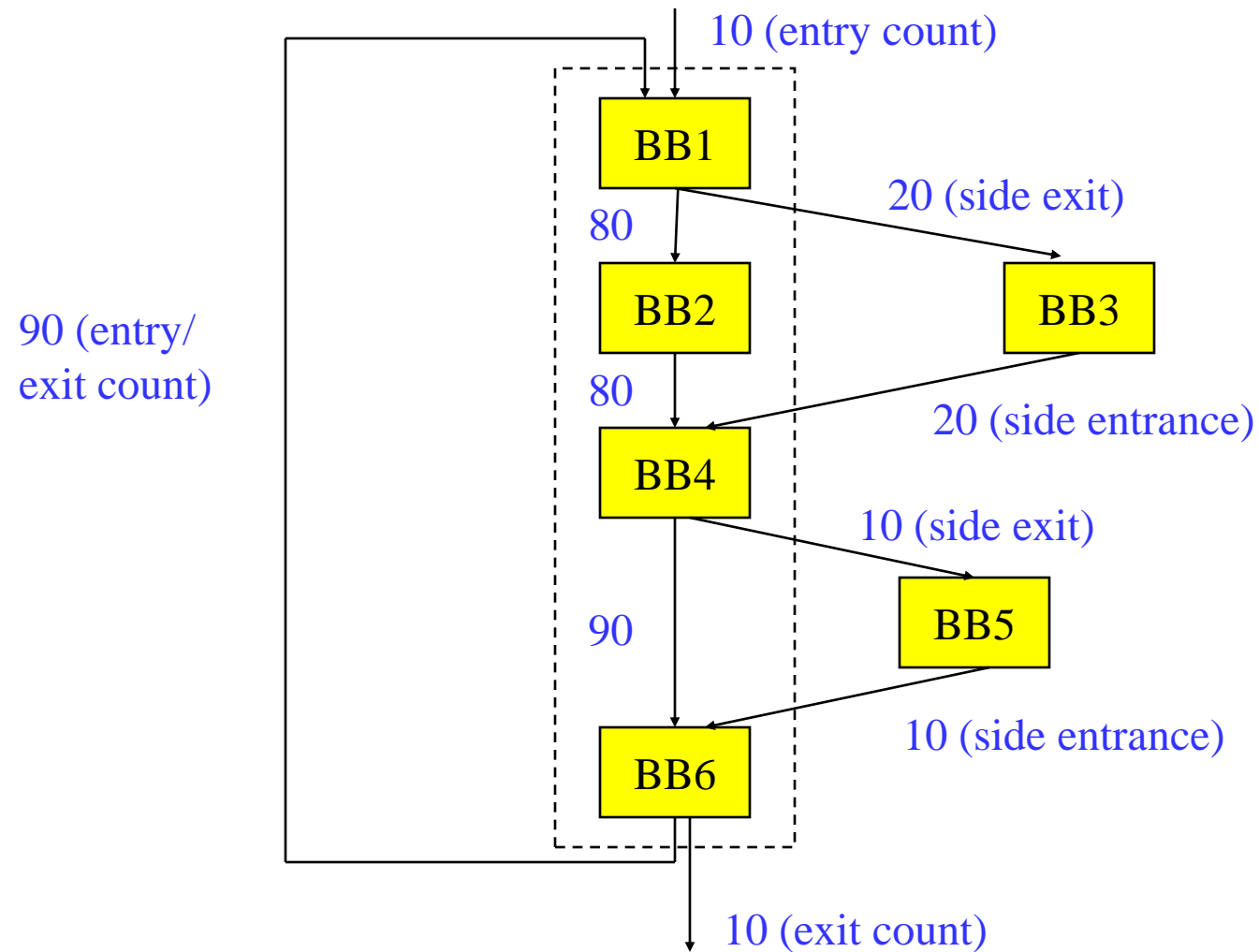
- » Define new region types that consist of multiple BBs
- » Profile information used in the identification
- » Sequential control flow (sorta)
- » Pretend the regions are basic blocks

Region Type 1 - Trace

- ❖ Trace - Linear collection of basic blocks that tend to execute in sequence
 - » “Likely control flow path”
 - » Acyclic (outer backedge ok)
- ❖ Side entrance – branch into the middle of a trace
- ❖ Side exit – branch out of the middle of a trace
- ❖ Compilation strategy
 - » Compile assuming path occurs 100% of the time
 - » Patch up side entrances and exits afterwards
- ❖ Motivated by scheduling (i.e., trace scheduling)



Linearizing a Trace



Intelligent Trace Layout for Icache Performance

