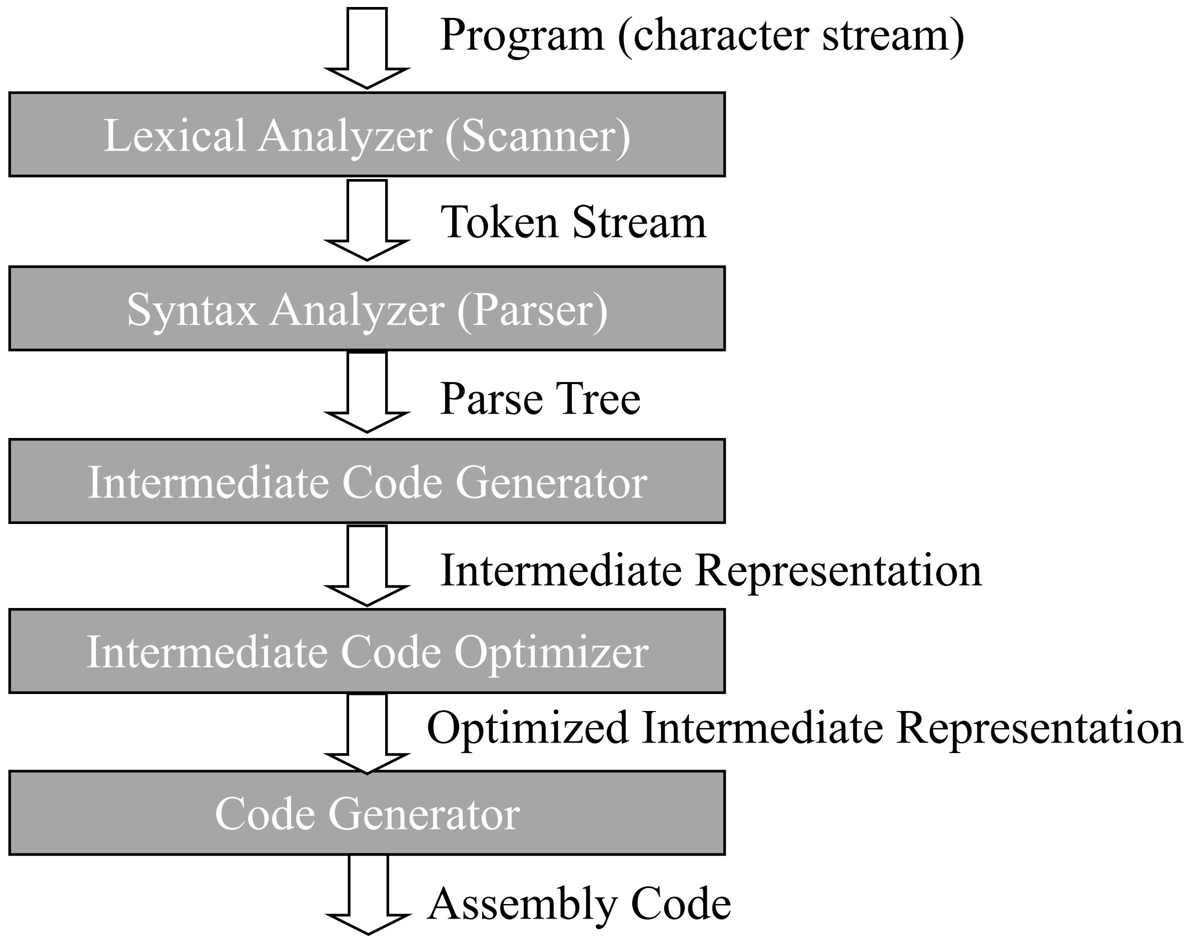
Introduction to Optimization

Control-Flow Analysis

Outline

- Overview of Optimizations
- Control-Flow Analysis
- Dominators
- Back Edges and Natural Loops

Anatomy of a Compiler



Example

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + b*y;
    }
    return x;
}
```

Example in Assembly

```
test:
   subu  $fp, 16
   sw    zero, 0($fp)          # x = 0
   sw    zero, 4($fp)          # y = 0
   sw    zero, 8($fp)          # i = 0
lab1:                          # for(i=0;i<N; i++)
   mul   $t0, $a0, 4           # a*4
   div   $t1, $t0, $a1         # a*4/b
   lw    $t2, 8($fp)           # i
   mul   $t3, $t1, $t2         # a*4/b*i
   lw    $t4, 8($fp)           # i
   addui $t4, $t4, 1           # i+1
   lw    $t5, 8($fp)           # i
   addui $t5, $t5, 1           # i+1
   mul   $t6, $t4, $t5         # (i+1)*(i+1)
   addu  $t7, $t3, $t6         # a*4/b*i + (i+1)*(i+1)
   lw    $t8, 0($fp)           # x
   add   $t8, $t7, $t8         # x = x + a*4/b*i + (i+1)*(i+1)
   sw    $t8, 0($fp)
   ...
```

Example in Assembly

```
...
lw    $t0, 4($fp)           # y
mul   $t1, $t0, a1          # b*y
lw    $t2, 0($fp)           # x
add   $t2, $t2, $t1         # x = x + b*y
sw    $t2, 0($fp)

lw    $t0, 8($fp)           # i
addui $t0, $t0, 1           # i+1
sw    $t0, 8($fp)
ble   $t0, $a3, lab1

lw    $v0, 0($fp)
addu  $fp, 16
b     $ra
```

Let's Optimize...

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + b*y;
    }
    return x;
}
```

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + b*y;
    }
    return x;
}
```

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + b*y;
    }
    return x;
}
```

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + b*y;
    }
    return x;
}
```

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + b*0;
    }
    return x;
}
```

Algebraic Simplification

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + b*0;
    }
    return x;
}
```

Algebraic Simplification

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + b*0;
    }
    return x;
}
```

Algebraic Simplification

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + 0;
    }
    return x;
}
```

Algebraic Simplification

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + 0;
    }
    return x;
}
```

Algebraic Simplification

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x;
    }
    return x;
}
```

Copy Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x;
    }
    return x;
}
```

Copy Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);

    }
    return x;
}
```

Common Sub-expression Elimination (CSE)

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);

    }
    return x;
}
```

Common Sub-expression Elimination (CSE)

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);

    }
    return x;
}
```

Common Sub-expression Elimination (CSE)

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y, t;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + (4*a/b)*i + t * t;

    }
    return x;
}
```

Dead Code Elimination

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y, t;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + (4*a/b)*i + t * t;

    }
    return x;
}
```

Dead Code Elimination

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y, t;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + (4*a/b)*i + t * t;

    }
    return x;
}
```

Dead Code Elimination

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t;
    x = 0;

    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + (4*a/b)*i + t * t;

    }
    return x;
}
```

Loop Invariant Removal

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t;
    x = 0;

    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + (4*a/b)*i + t * t;

    }
    return x;
}
```

Loop Invariant Removal

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t;
    x = 0;

    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + (4*a/b)*i + t * t;

    }
    return x;
}
```

Loop Invariant Removal

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u;
    x = 0;
    u = (4*a/b);
    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + u *i + t * t;

    }
    return x;
}
```

Strength Reduction

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u;
    x = 0;
    u = (4*a/b);
    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + u*i + t * t;

    }
    return x;
}
```

Strength Reduction

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u;
    x = 0;
    u = (4*a/b);
    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + u*i + t * t;

    }
    return x;
}
```

```
u*0,   v=0,
u*1,   v=v+u,
u*2,   v=v+u,
u*3,   v=v+u,
u*4,   v=v+u,
...    ...
```

Strength Reduction

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = (4*a/b);
    v = 0;
    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + u*i + t*t;
      v = v + u;
    }
    return x;
}
```

Strength Reduction

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = (4*a/b);
    v = 0;
    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + v + t*t;
      v = v + u;
    }
    return x;
}
```

Strength Reduction

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = (4*a/b);
    v = 0;
    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + v + t*t;
      v = v + u;
    }
    return x;
}
```

Strength Reduction

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = (4*a/b);
    v = 0;
    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + v + t*t;
      v = v + u;
    }
    return x;
}
```

Strength Reduction

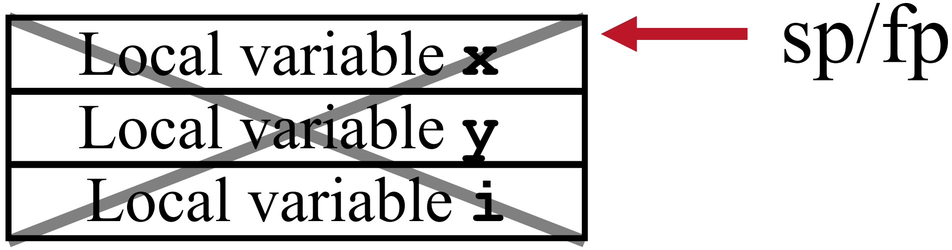
```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = ((a<<2)/b);
    v = 0;
    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + v + t*t;
      v = v + u;
    }
    return x;
}
```

Register Allocation

Local variable **`x`**
Local variable **`y`**
Local variable **`i`**

← sp/fp

Register Allocation




```
$t9 = X
$t8 = t
$t7 = u
$t6 = v
$t5 = i
```

Optimized Example

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = ((a<<2)/b);
    v = 0;
    for(i = 0; i <= N; i++) {
      t = i+1;
      x = x + v + t*t;
      v = v + u;
    }
    return x;
}
```

Optimized Example in Assembly

```
test:
   subu  $fp, 16
   add   $t9, zero, zero       # x = 0
   sll   $t0, $a0, 2           # a<<2
   div   $t7, $t0, $a1         # u = (a<<2)/b
   add   $t6, zero, zero       # v = 0
   add   $t5, zero, zero       # i = 0

lab1:                          # for(i=0;i<N; i++)
   addui$t8, $t5, 1            # t = i+1
   mul   $t0, $t8, $t8         # t*t
   addu  $t1, $t0, $t6         # v + t*t
   addu $t9, t9, $t1           # x = x + v + t*t

   addu $6, $6, $7             # v = v + u

   addui$t5, $t5, 1            # i = i+1
   ble   $t5, $a3, lab1

   addu  $v0, $t9, zero
   addu  $fp, 16
   b     $ra
```

Optimized Example in Assembly

Unoptimized Code

```
test:
      subu      $fp, 16
      sw        zero, 0($fp)
      sw        zero, 4($fp)
      sw        zero, 8($fp)
lab1:
      mul       $t0, $a0, 4
      div       $t1, $t0, $a1
      lw        $t2, 8($fp)
      mul       $t3, $t1, $t2
      lw        $t4, 8($fp)
      addui     $t4, $t4, 1
      lw        $t5, 8($fp)
      addui     $t5, $t5, 1
      mul       $t6, $t4, $t5
      addu      $t7, $t3, $t6
      lw        $t8, 0($fp)
      add       $t8, $t7, $t8
      sw        $t8, 0($fp)
      lw        $t0, 4($fp)
      mul       $t1, $t0, a1
      lw        $t2, 0($fp)
      add       $t2, $t2, $t1
      sw        $t2, 0($fp)
      lw        $t0, 8($fp)
      addui     $t0, $t0, 1
      sw        $t0, 8($fp)
      ble       $t0, $a3, lab1

      lw        $v0, 0($fp)
      addu      $fp, 16
      b         $ra
```

4*ld/st + 2*add/sub + br +
N*(9*ld/st + 6*add/sub + 4* mul + div + br)
= 7 + N*21

Optimized Code

```
test:
      subu      $fp, 16
      add       $t9, zero, zero

      sll       $t0, $a0, 2
      div       $t7, $t0, $a1
      add       $t6, zero, zero

      add       $t5, zero, zero

lab1:
      addui     $t8, $t5, 1

      mul       $t0, $t8, $t8
      addu      $t1, $t0, $t6
      addu      $t9, t9, $t1

      addu      $6, $6, $7

      addui     $t5, $t5, 1

      ble       $t5, $a3, lab1

      addu      $v0, $t9, zero
      addu      $fp, 16
      b         $ra
```

6*add/sub + shift + div + br +
N*(5*add/sub + mul + br)
= 9 + N*7

Outline

- Overview of Optimizations
- Control-Flow Analysis
- Dominators
- Back Edges and Natural Loops

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + b*y;
    }
    return x;
}
```

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + b*y;
    }
    return x;
}
```

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + b*y;
    }
    return x;
}
```

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + b*0;
    }
    return x;
}
```

Implementing Constant Propagation

- Find an RHS expression that is a Constant
- Replace the use of the LHS variable with the RHS Constant given that:
  - All paths to the use(s) of LHS variable pass through the assignment to the LHS with the constant
  - There are no intervening definition of the RHS variable

- Need to know the "Control-Flow" of the program

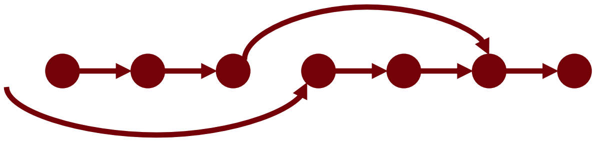
Implementing Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
      x = x + (4*a/b)*i + (i+1)*(i+1);
      x = x + b*y;
    }
    return x;
}
```

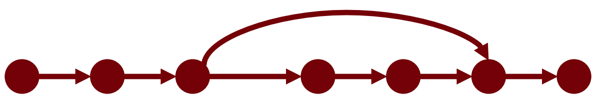
Representing the Control Flow of a Program

- Most instructions
  - execute the next instruction
  - straight line control-flow

- Jump instructions
  - execute form different location
  - jump in control-flow

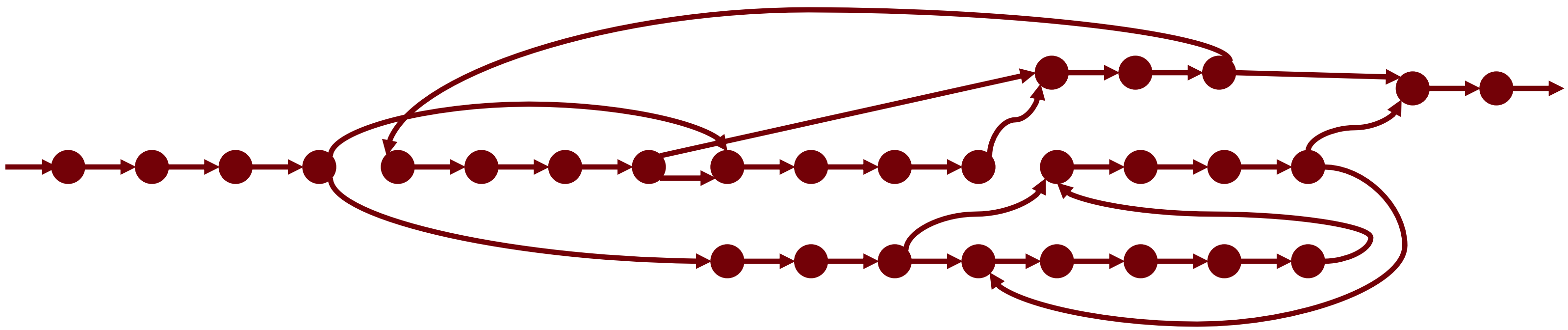


- Branch instructions
  - execute either the next instruction or from a different location
  - fork in the control-flow



Representing Control Flow

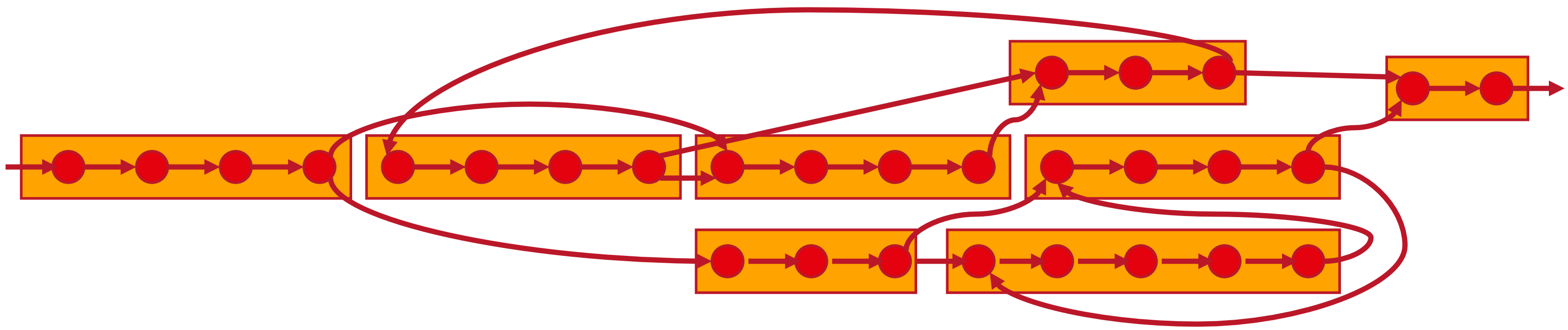
- Forms a Graph



- A Very Large Graph
- Observations:
  - lots of straight-line connections
  - simplify the graph by grouping some instructions

Representing Control Flow

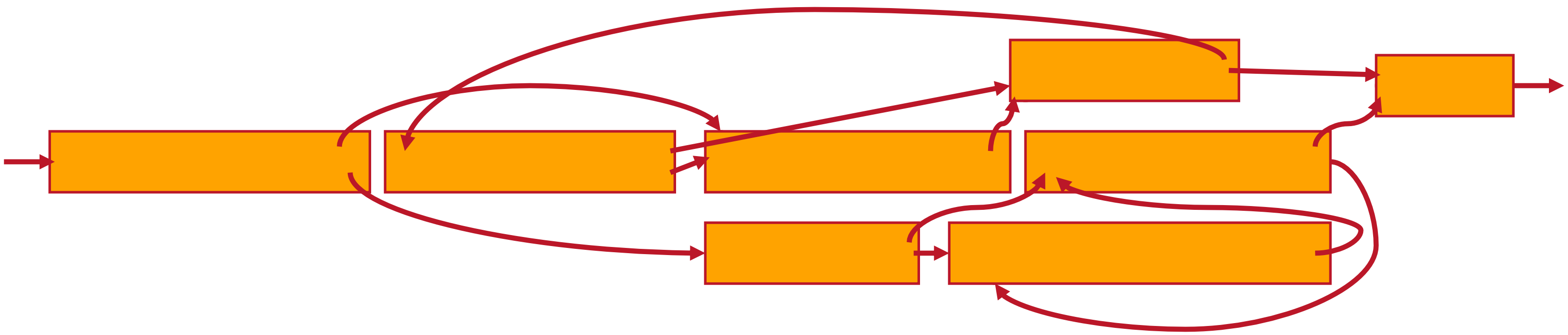
- Forms a Graph



- A Very Large Graph
- Observations:
  - lots of straight-line connections
  - simplify the graph by grouping some instructions

Representing Control Flow

- Forms a Graph



- A Very Large Graph
- Observations:
  - lots of straight-line connections
  - simplify the graph by grouping some instructions

Basic Blocks

- Def: A *Maximal Sequence* of Instructions such that:
  1. Only the first instruction can be reached from outside the basic block
  2. All the instructions are executed consecutively *iff* the first instruction is executed
     - No branch or jump instructions in the basic block
     - Except the last instruction
     - No labels within the basic block
     - Except before the first instruction

Basic Blocks: Algorithm

- Input: Sequence of Three-Address Instructions
- Output: A list of Basic Blocks
- Algorithm:
  - Determine the set of *leader instructions* - the head of each basic block - using the following:
    - The first statement of the program is a *leader*
    - Any statement that is the target of a goto (either conditional or not) is a *leader instructions*
    - Any statement that immediately follows a goto or unconditional goto statement is a *leader instruction*
  - For each *leader instruction*, its basic block consists of the *leader instruction* and all the statements up to but not including the next *leader instruction* or the end of the program.

Basic Blocks: Example

```
test:
     subu     $fp, 16
     sw       zero, 0($fp)
     sw       zero, 4($fp)
     sw       zero, 8($fp)
```

```
lab1:
     mul      $t0, $a0, 4
     div      $t1, $t0, $a1
     lw       $t2, 8($fp)
     mul      $t3, $t1, $t2
     lw       $t4, 8($fp)
     addui    $t4, $t4, 1
     lw       $t5, 8($fp)
     addui    $t5, $t5, 1
     mul      $t6, $t4, $t5
     addu     $t7, $t3, $t6
     lw       $t8, 0($fp)
     add      $t8, $t7, $t8
     sw       $t8, 0($fp)
     lw       $t0, 4($fp)
     mul      $t1, $t0, a1
     lw       $t2, 0($fp)
     add      $t2, $t2, $t1
     sw       $t2, 0($fp)
     lw       $t0, 8($fp)
     addui    $t0, $t0, 1
     sw       $t0, 8($fp)
     ble      $t0, $a3, lab1
```

```
     lw       $v0, 0($fp)
     addu     $fp, 16
     b        $ra
```

Control Flow Graph (CFG)

- Control-Flow Graph G = <N, E>
- Nodes(N): Basic Blocks
- Edges(E): (x,y) ∈ E iff first instruction in the Basic Block y follows the last instruction in the basic block x
  - First instruction in y is the target of branch or jump instruction (last instruction) in the basic block x
  - first instruction of y is next after the last instruction of x in memory and the last instruction of x is not a jump instruction

Control Flow Graph (CFG)

- Block with the first instruction of the procedure is the entry node (block with the procedure label)
- The blocks with the return instruction are exit nodes.
  – Can make a single exit node by adding a special node

Why Control-Flow Analysis ?

- Uncover Flow Structure:
  - Loops
  - Convergence and Divergence of Paths

- Loops are Important to Optimize
  - Programs spend a lot of times in loops and recursive cycles
  - Many special optimizations can be done on loops

- Programmers organize code using structured control-flow (if-then-else, for-loops *etc*)
  - Optimizer can exploit this
  - *but* need to discover them first

Challenges in Control-Flow Analysis

- Unstructured Control Flow
  - Use of goto's by the programmer
  - Only way to build certain control structures

```
L1: x = 0
    if (y > 0) goto L3
L2: if (y < 0) goto L1
L3: y = y + z
    goto L2
```

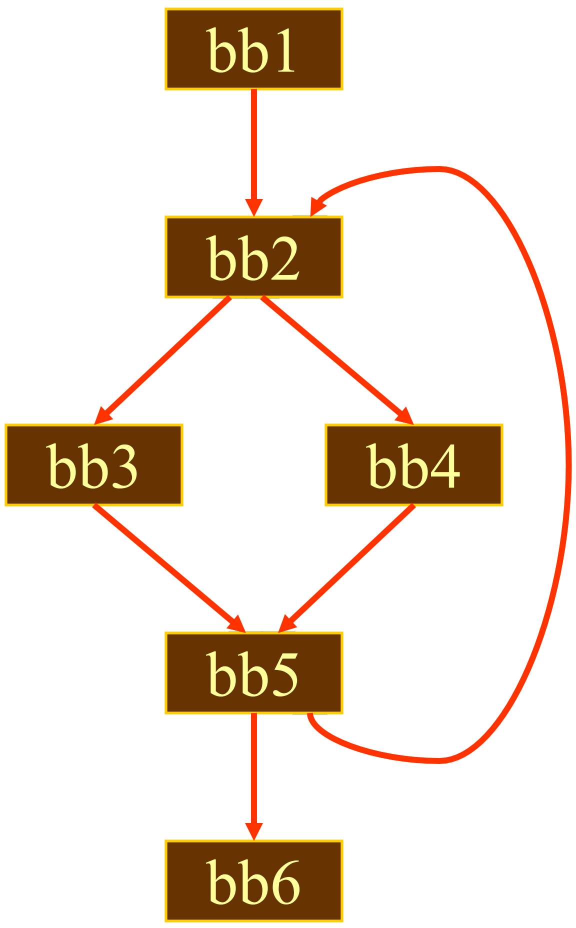
- Obscured Control Flow
  - Method Invocations
  - Procedure Variables
  - Higher-Order Functions
  - Jump Tables

```
Myobject->run()
```

Building CFGs

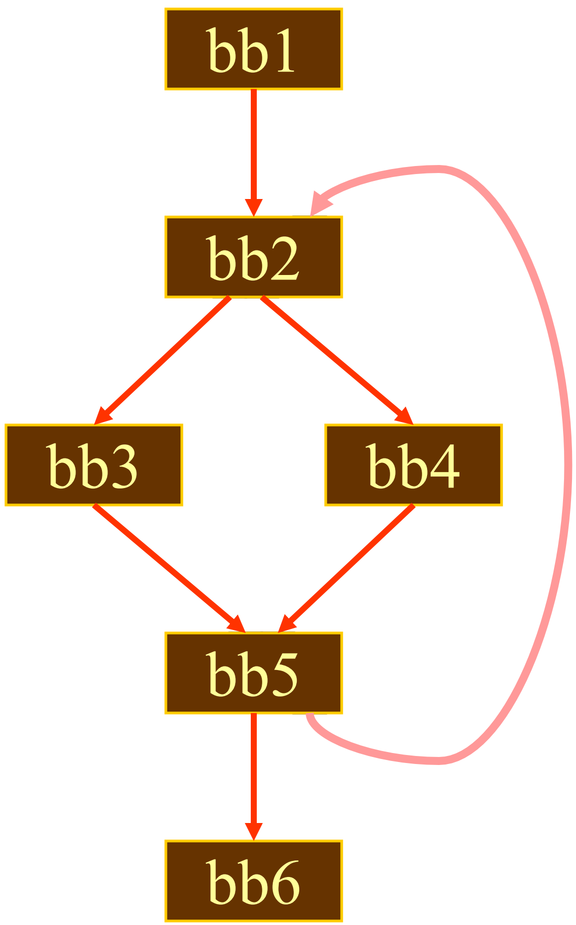
- Simple:
  - Programs are written in structured control flow
  - Has simple CFG patterns
- Not so!
  - *Gotos* can create different control-flow patterns than what is given by the structured control-flow
  - Need to perform analyses to identify true control-flow patterns

Identifying Recursive Structures Loops



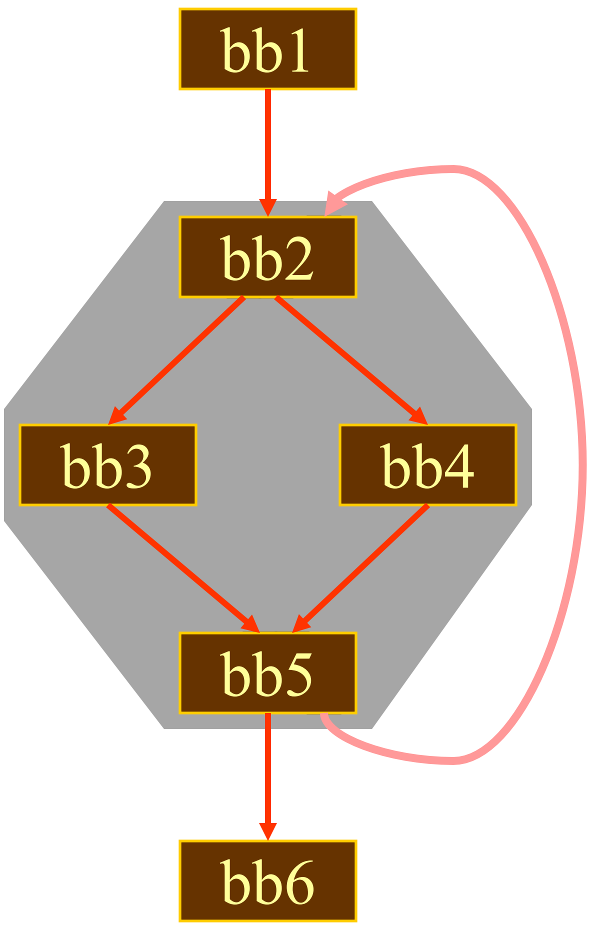
Identifying Recursive Structures Loops

- Identify **Back** Edges



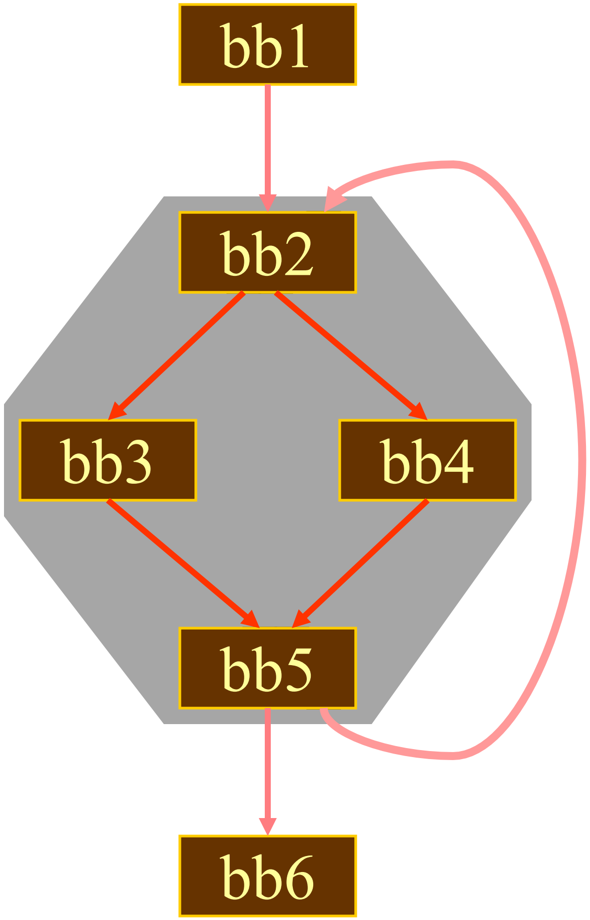
Identifying Recursive Structures Loops

- Identify **Back** Edges
- Find the nodes and edges in the loop given by the Back Edge



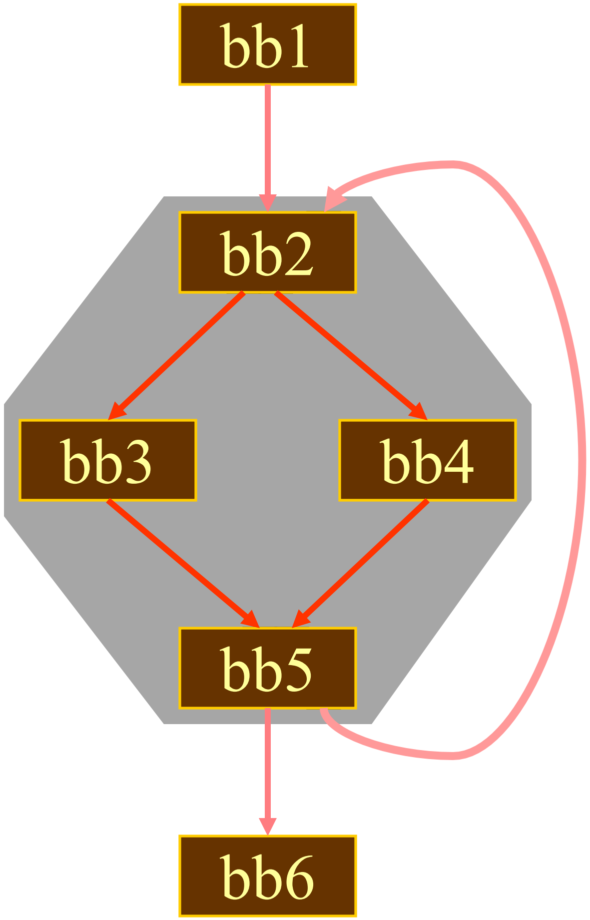
Identifying Recursive Structures Loops

- Identify **Back** Edges
- Find the nodes and edges in the loop given by the Back Edge
- Other than the Back Edge
  - Incoming edges only to the basic block with the back edge head
  - one outgoing edge from the basic block with the tail of the back edge



Identifying Recursive Structures Loops

- Identify **Back** Edges
- Find the nodes and edges in the loop given by the Back Edge
- Other than the Back Edge
  - Incoming edges only to the basic block with the back edge head
  - one outgoing edge from the basic block with the tail of the back edge
- How do I find the Back Edges?



Outline

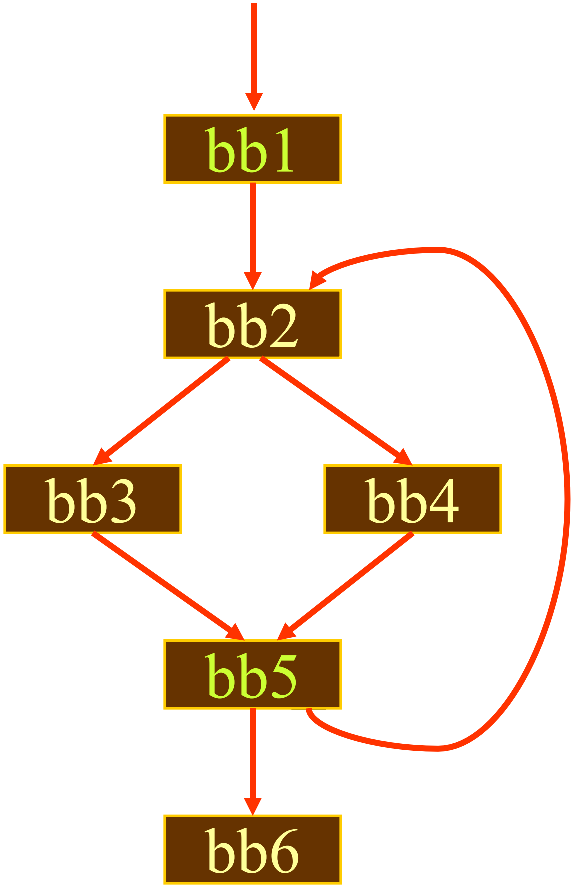
- Overview of Optimizations
- Control-Flow Analysis
- Dominators
- Back Edges and Natural Loops

Dominators

- Node x **Dominates** node y (x dom y) if every possible execution path from entry to node y includes node x

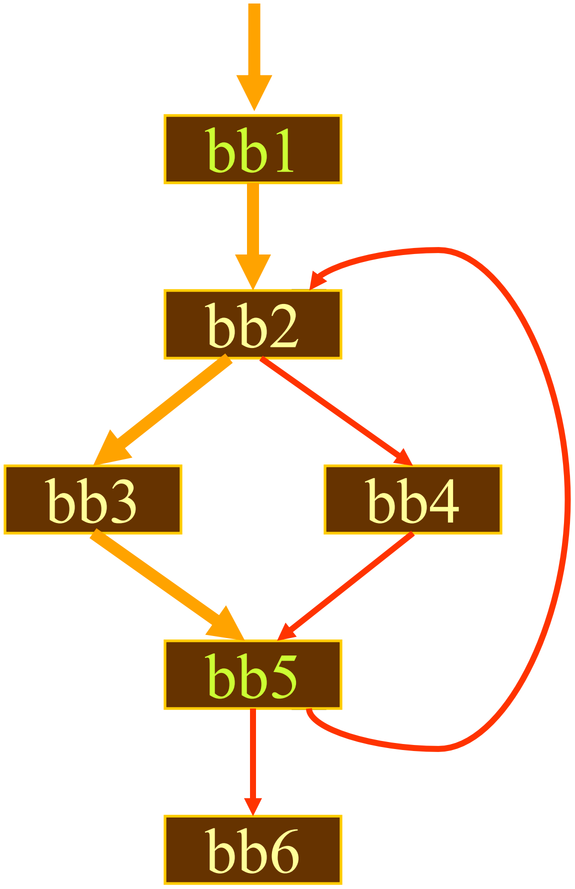
Dominators

- Does bb1 dom bb5?



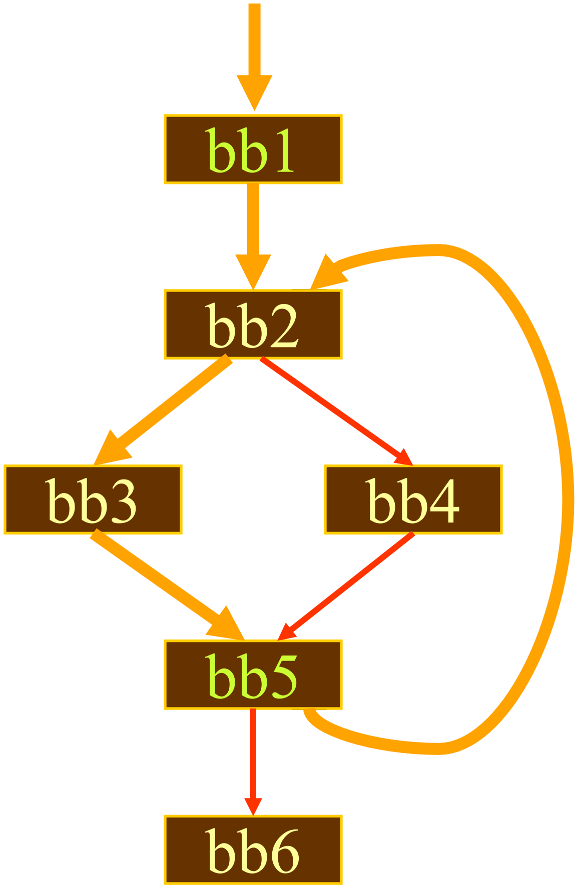
Dominators

- Does bb1 dom bb5?



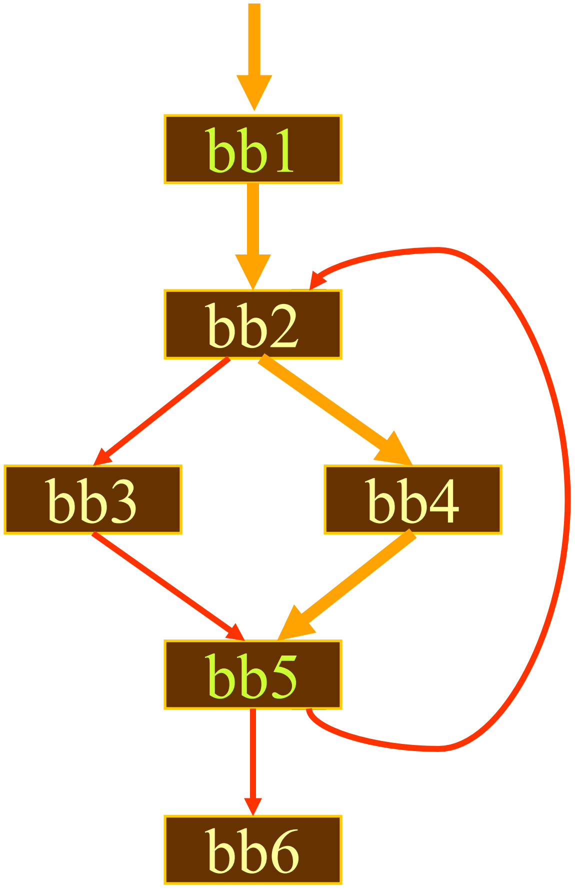
Dominators

- Does bb1 dom bb5?



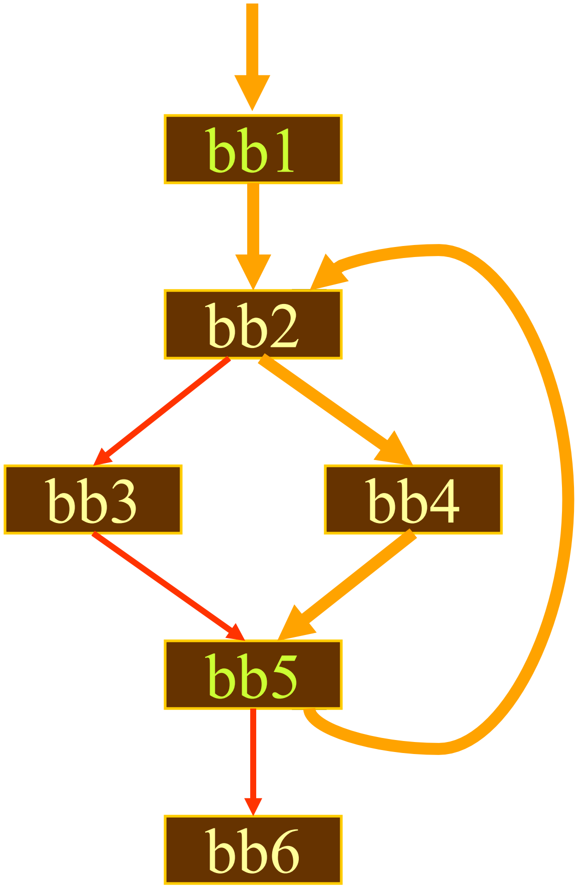
Dominators

- Does bb1 dom bb5?



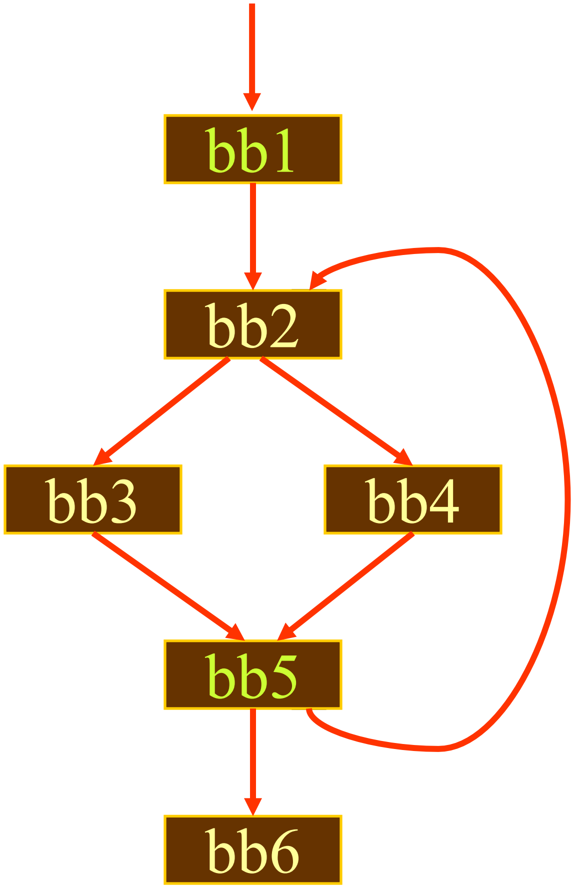
Dominators

- Does bb1 dom bb5?



Dominators

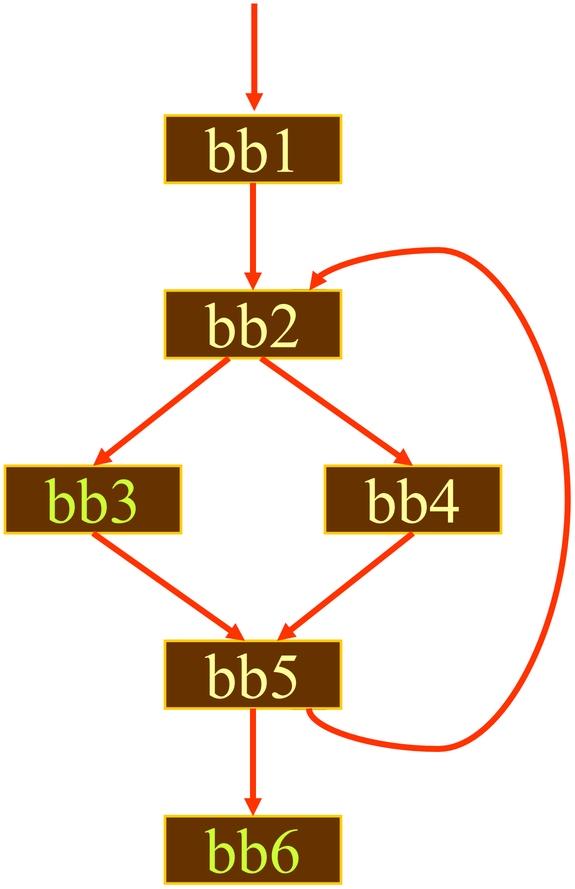
- Does bb1 dom bb5? ***Yes!***



Dominators

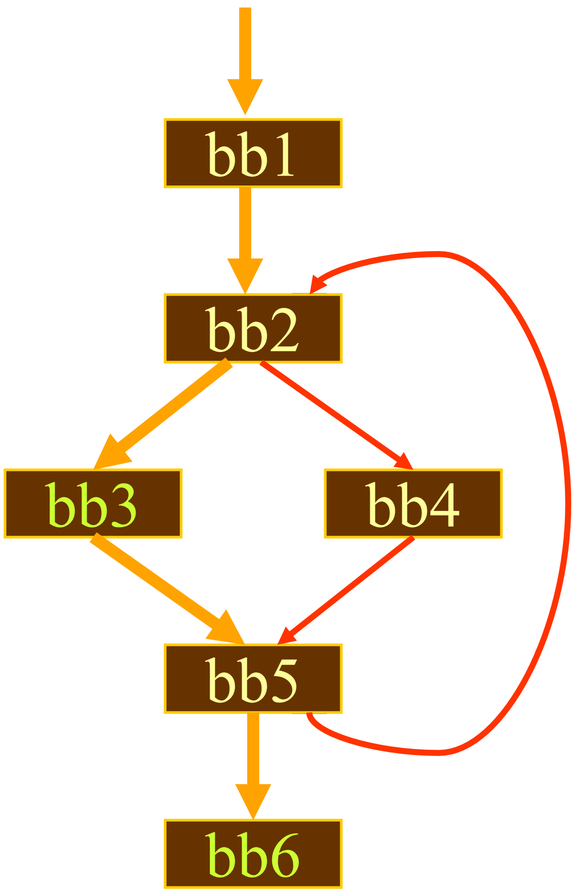
- Does bb1 dom bb5? ***Yes!***

- Does bb3 dom bb6?



Dominators

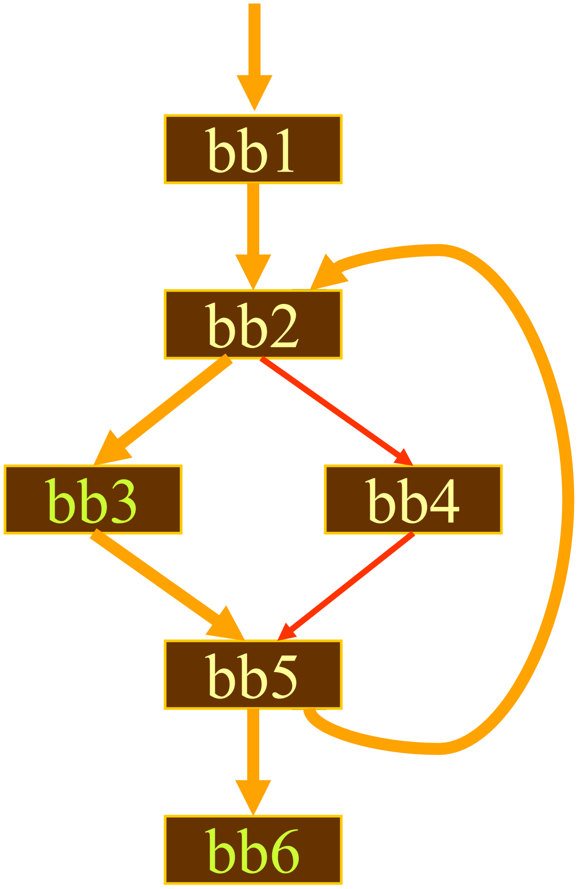
- Does bb1 dom bb5? ***Yes!***
- Does bb3 dom bb6?



Dominators

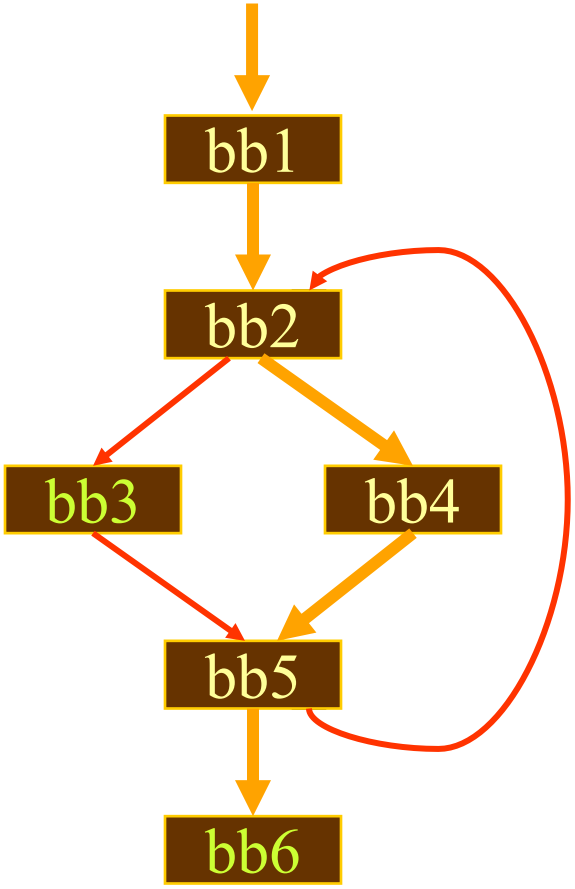
- Does bb1 dom bb5? ***Yes!***

- Does bb3 dom bb6?



Dominators

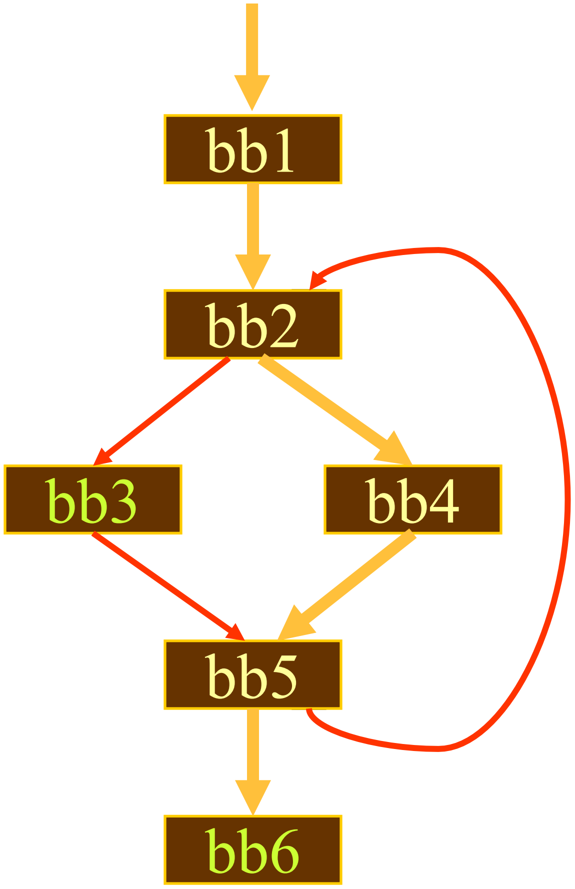
- Does bb1 dom bb5? ***Yes!***
- Does bb3 dom bb6?



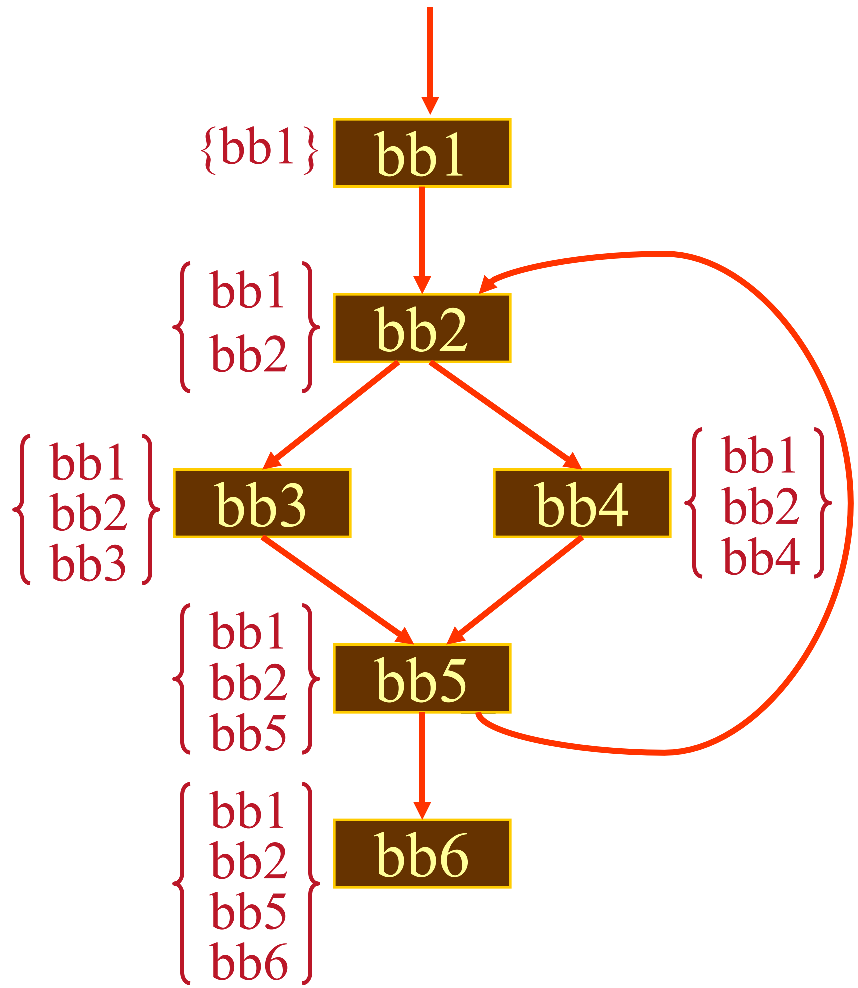
Dominators

- Does bb1 dom bb5? ***Yes!***

- Does bb3 dom bb6? ***No!***



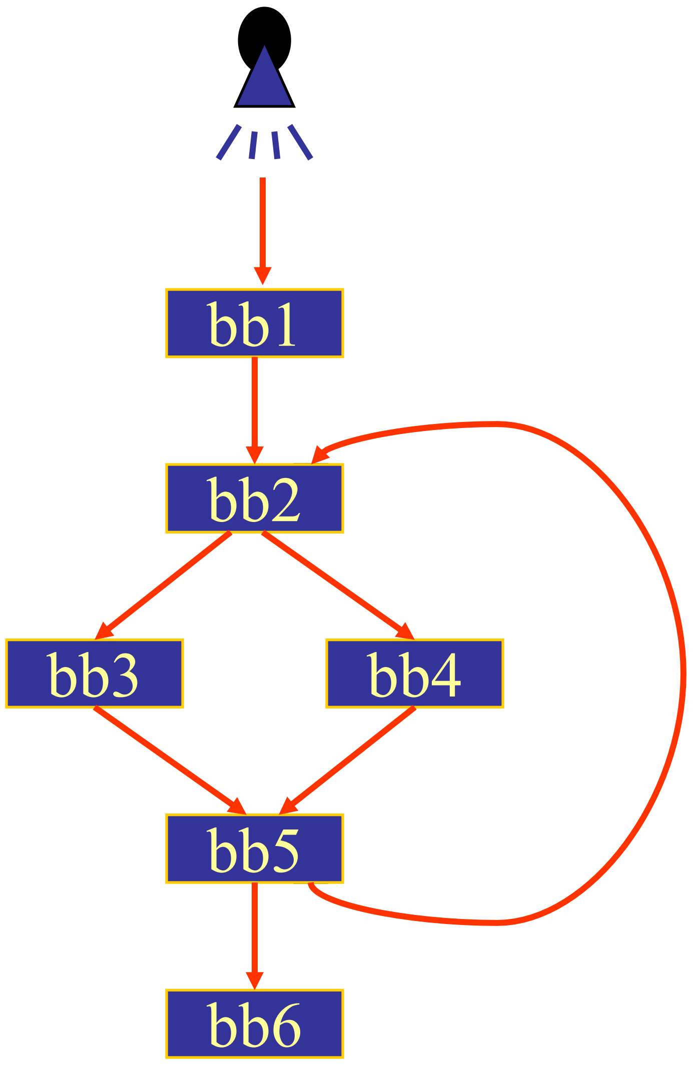
Dominators


{bb1}
bb1
{bb1 bb2}
bb2
{bb1 bb2 bb3}
bb3
bb4
{bb1 bb2 bb4}
{bb1 bb2 bb5}
bb5
{bb1 bb2 bb5 bb6}
bb6

Dominators - Intuition

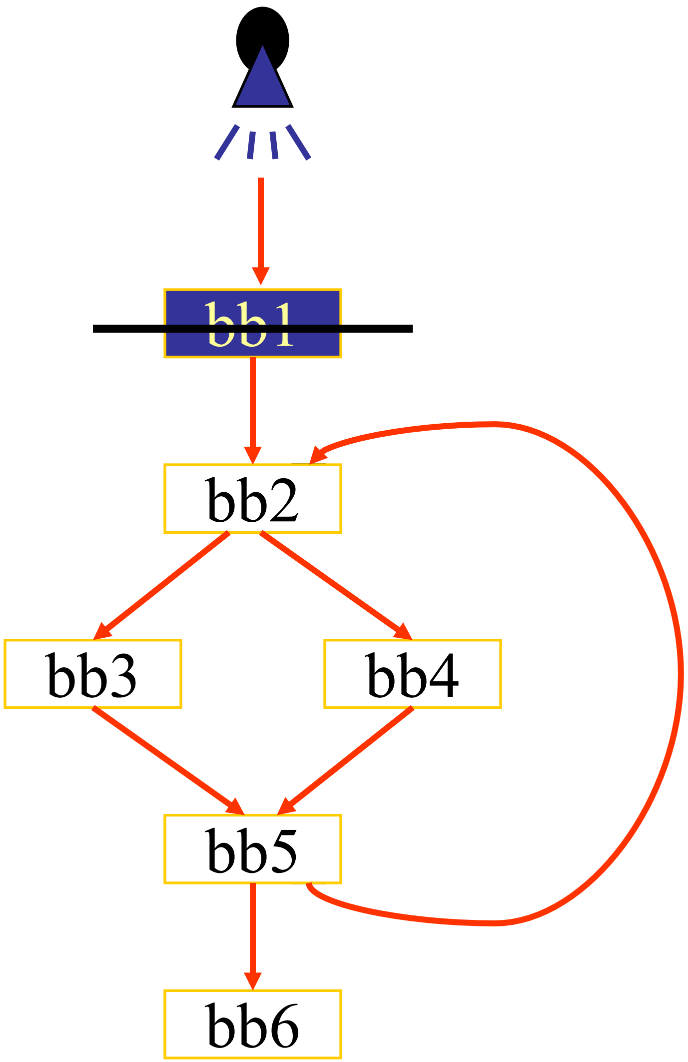
- Imagine a source of light at the start node and that edges are optical fibers

- To find which nodes are dominated by a given node a, place an **opaque barrier** at a and observe which nodes become dark.



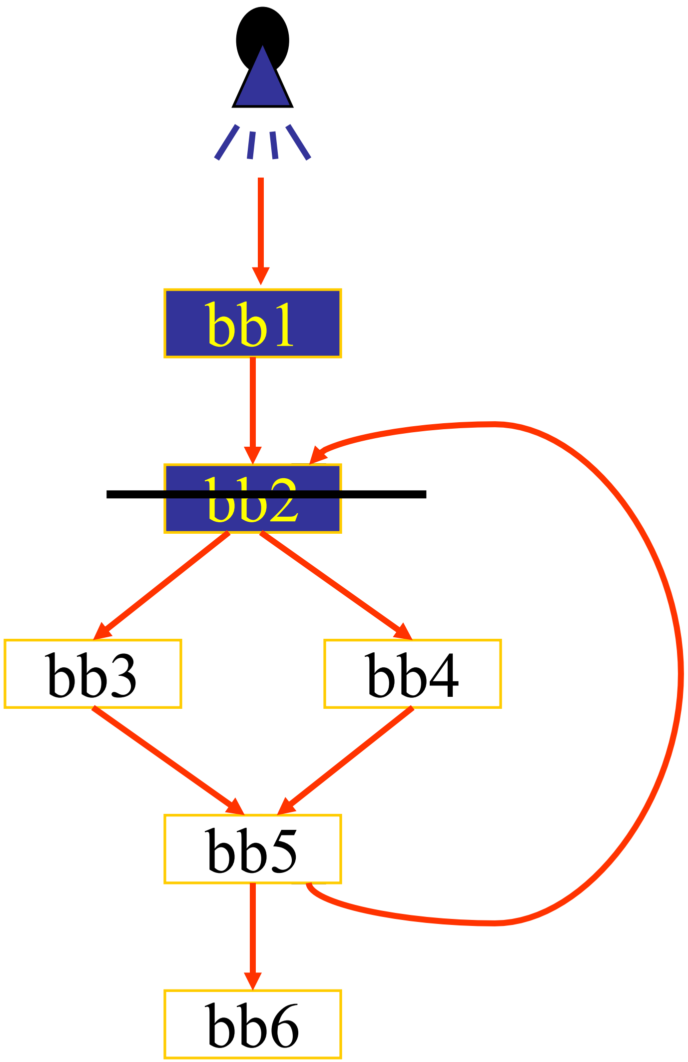
Dominators - Intuition

- Clearly, bb1 dominates all nodes in the graph



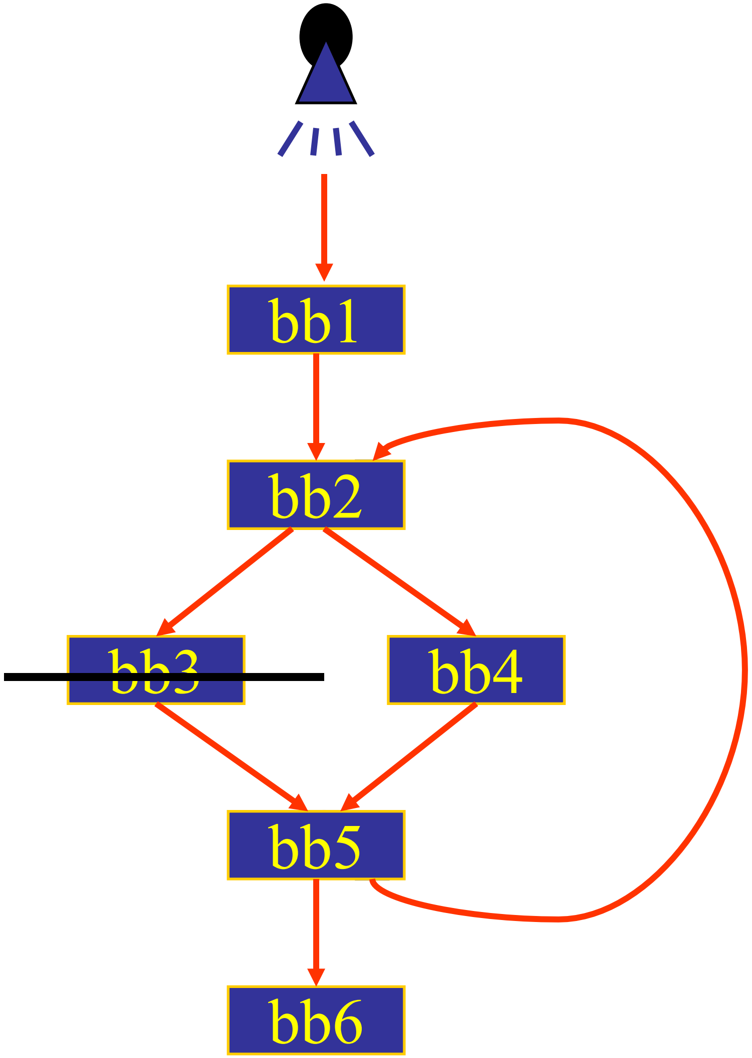
Dominators - Intuition

- Clearly, bb1 dominates all nodes in the graph
- Block bb2 dominates bb3, bb4, bb5 and bb6



Dominators - Intuition

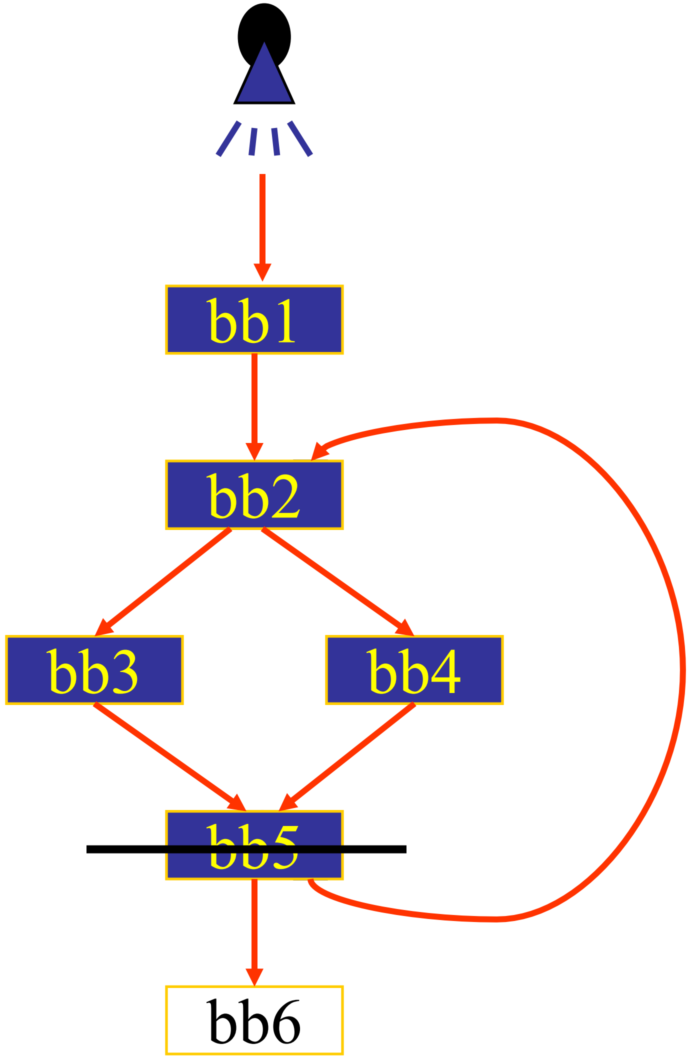
- Clearly, bb1 dominates all nodes in the graph
- Block bb2 dominates bb3, bb4, bb5 and bb6
- Block bb3 (bb4) dominates no other block



Pedro C. Diniz
pcd@fe.up.pt

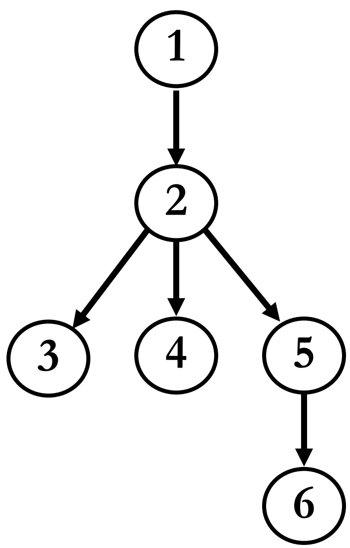
Dominators - Intuition

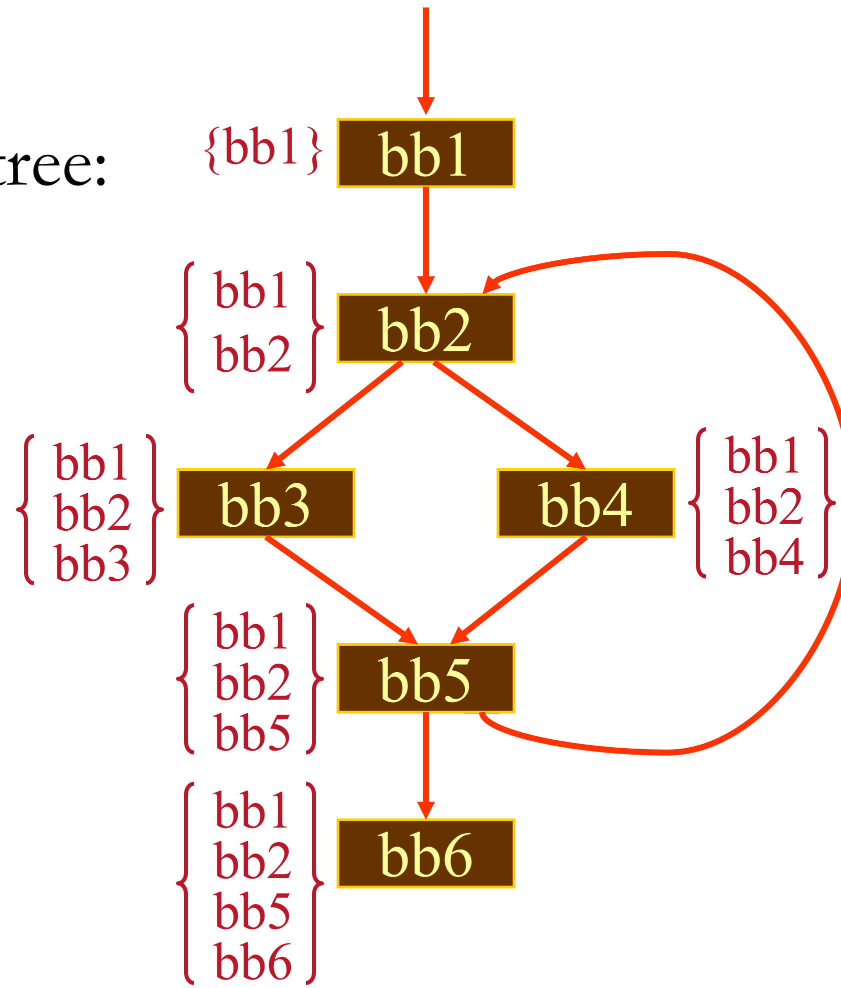
- Clearly, bb1 dominates all nodes in the graph
- Block bb2 dominates bb3, bb4, bb5 and bb6
- Block bb3 (bb4) dominates no other block
- Block bb5 dominates bb6



Dominators Tree

- Dominance Relationship, direct (or immediate) and indirect represented as a tree:



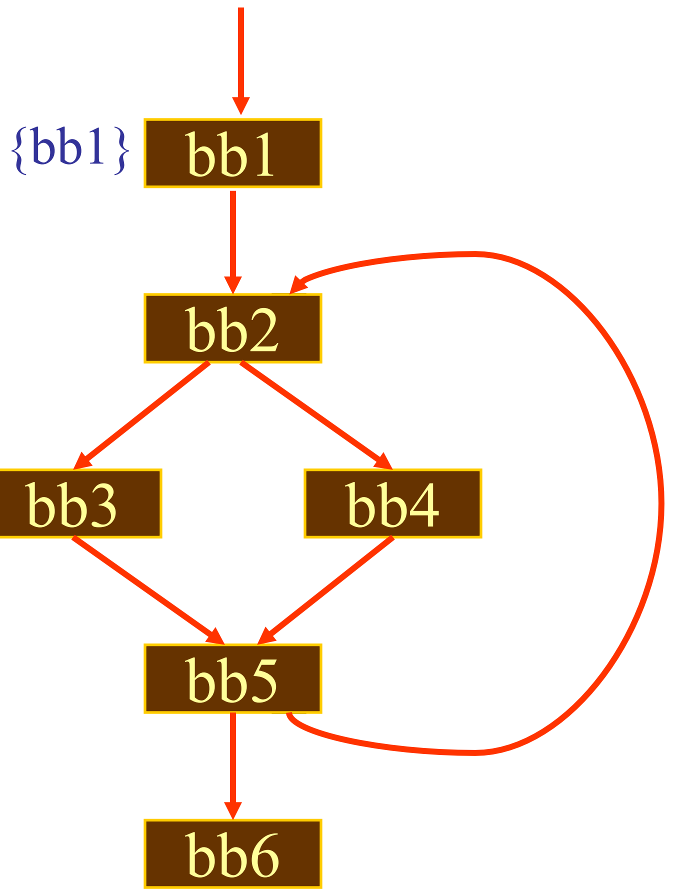


Computing Dominators

- a dom b iff
  - a = b or
  - a is the **unique immediate predecessor** of b or
  - a is a **dominator of all immediate predecessor** of b
- Algorithm
  - Make dominator set of the entry node itself
  - Make dominator set of the remainder node to be all graph nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

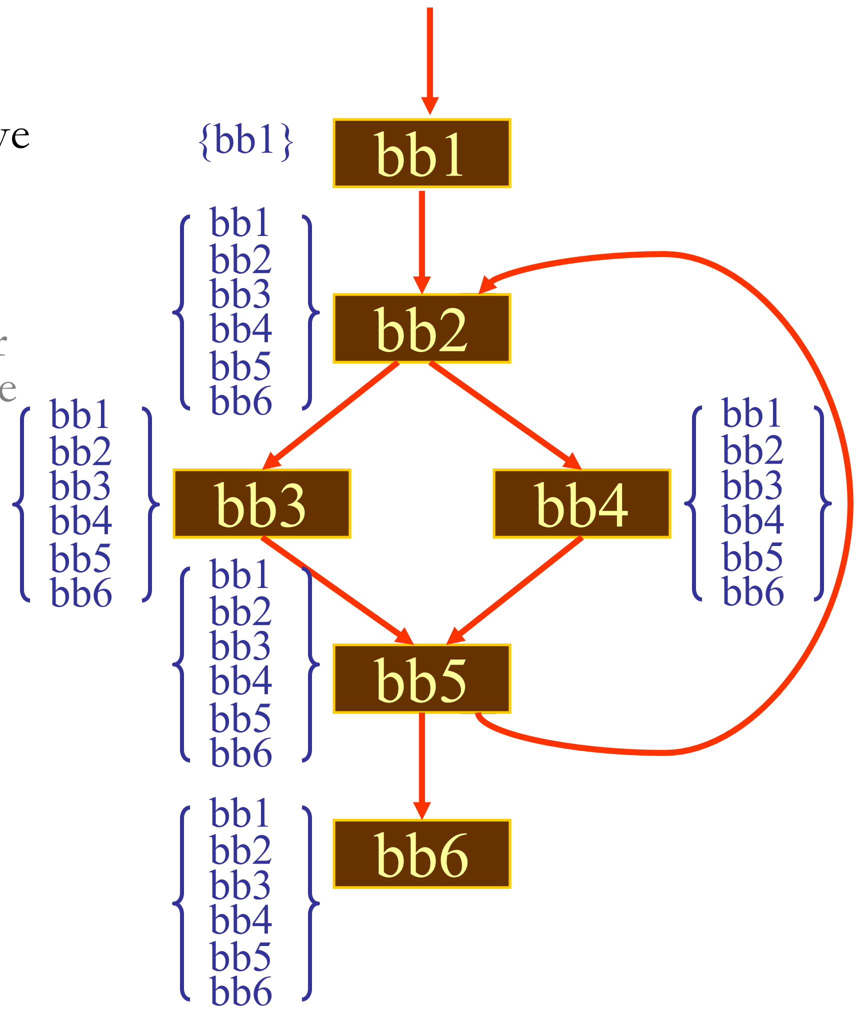
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



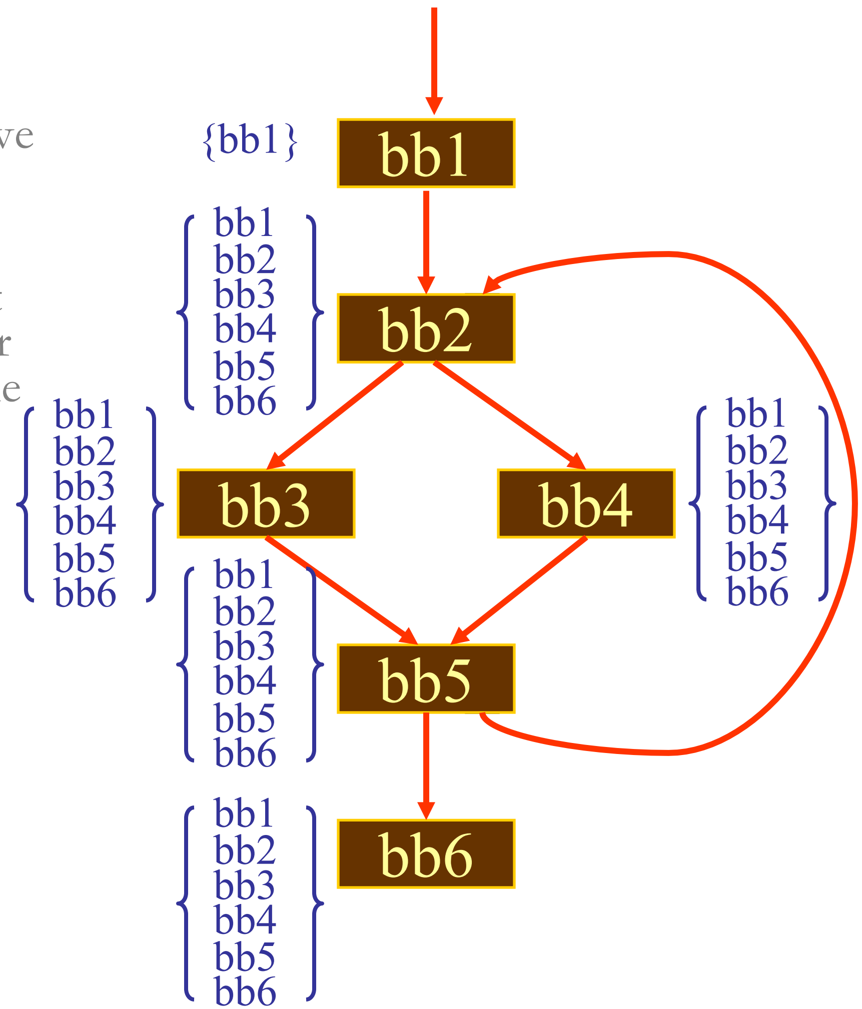
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



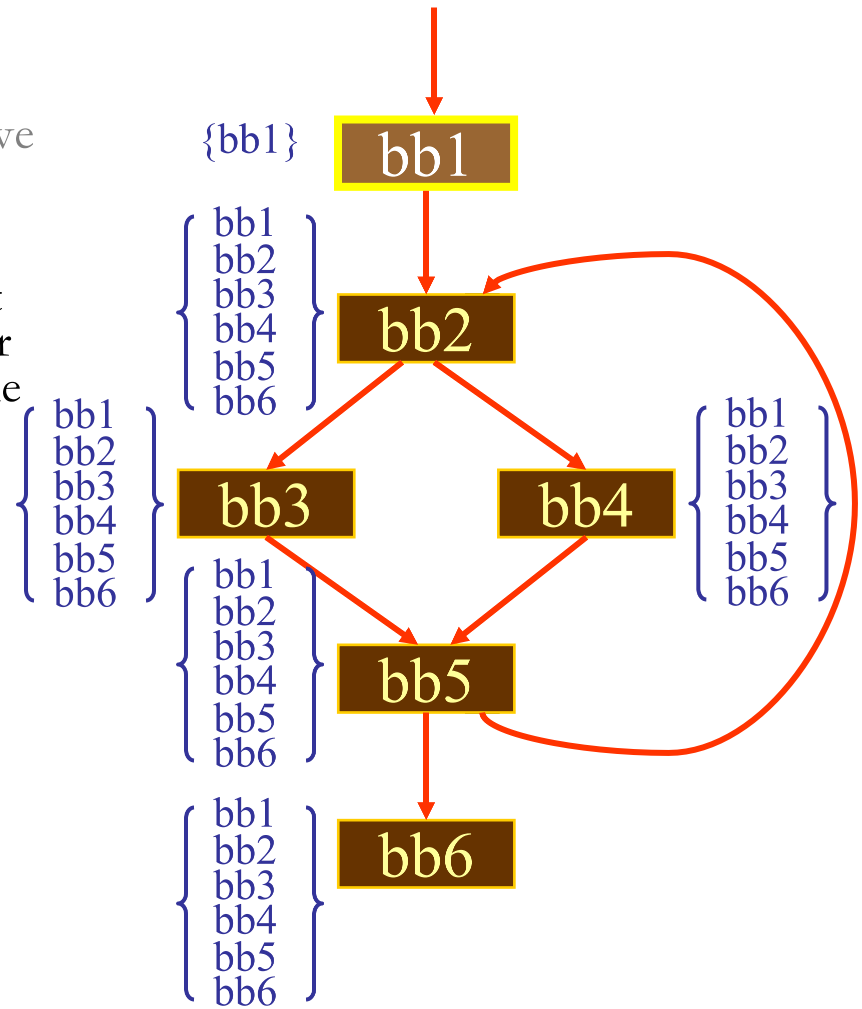
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



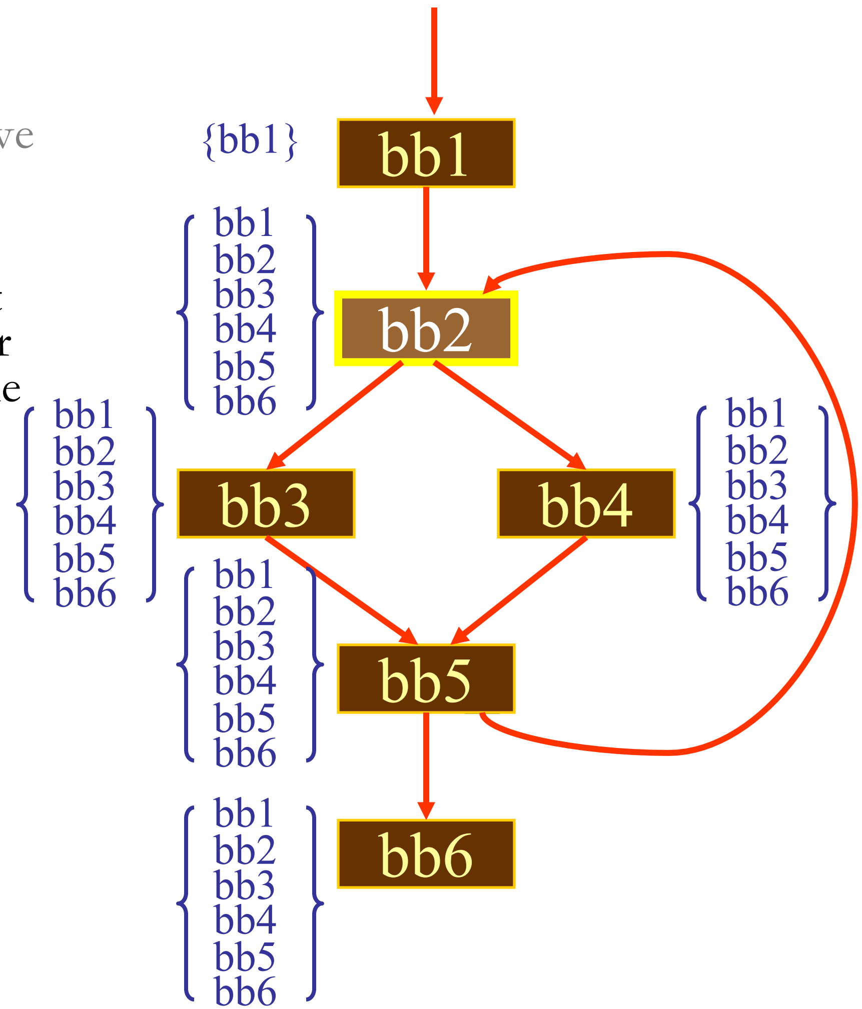
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



Computing Dominators

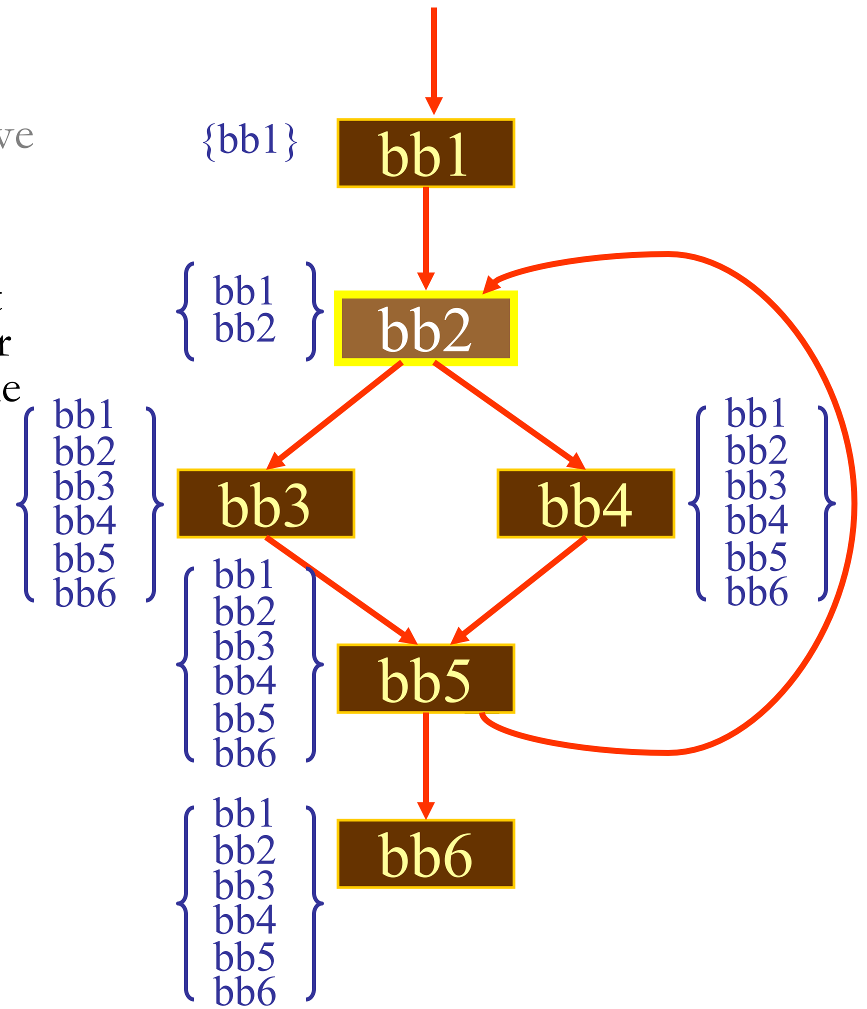
- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



Pedro C. Diniz
pcd@fe.up.pt

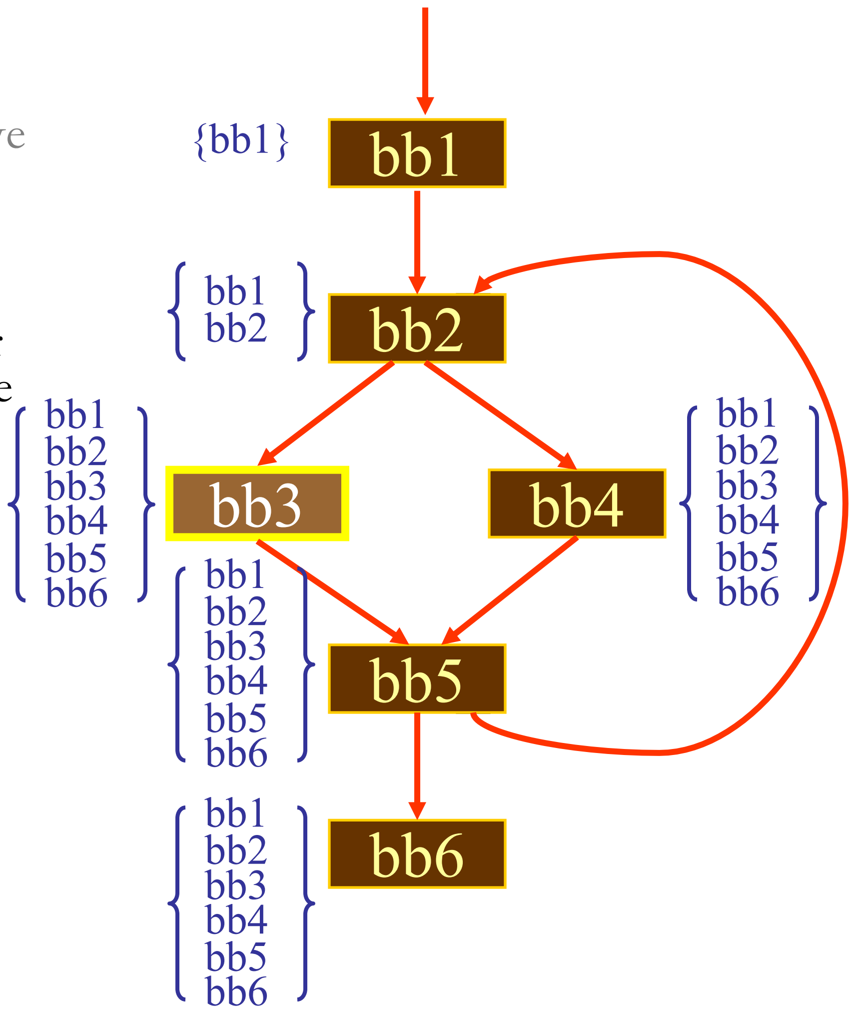
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



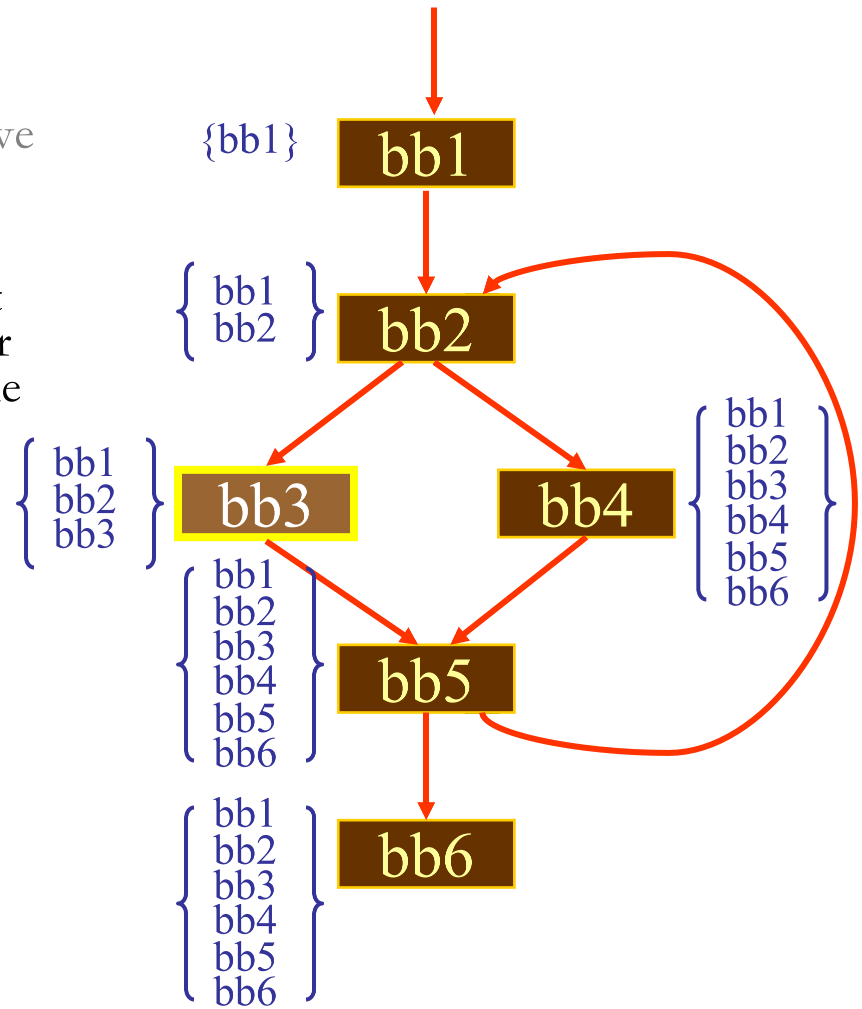
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



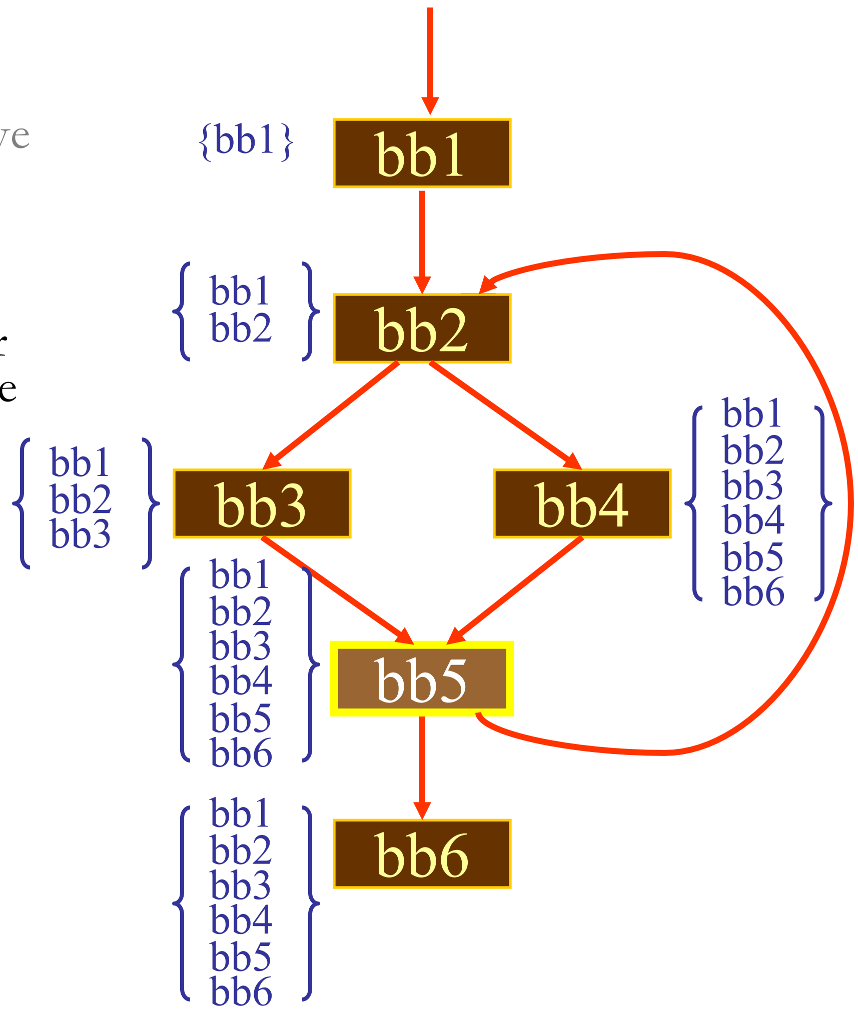
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



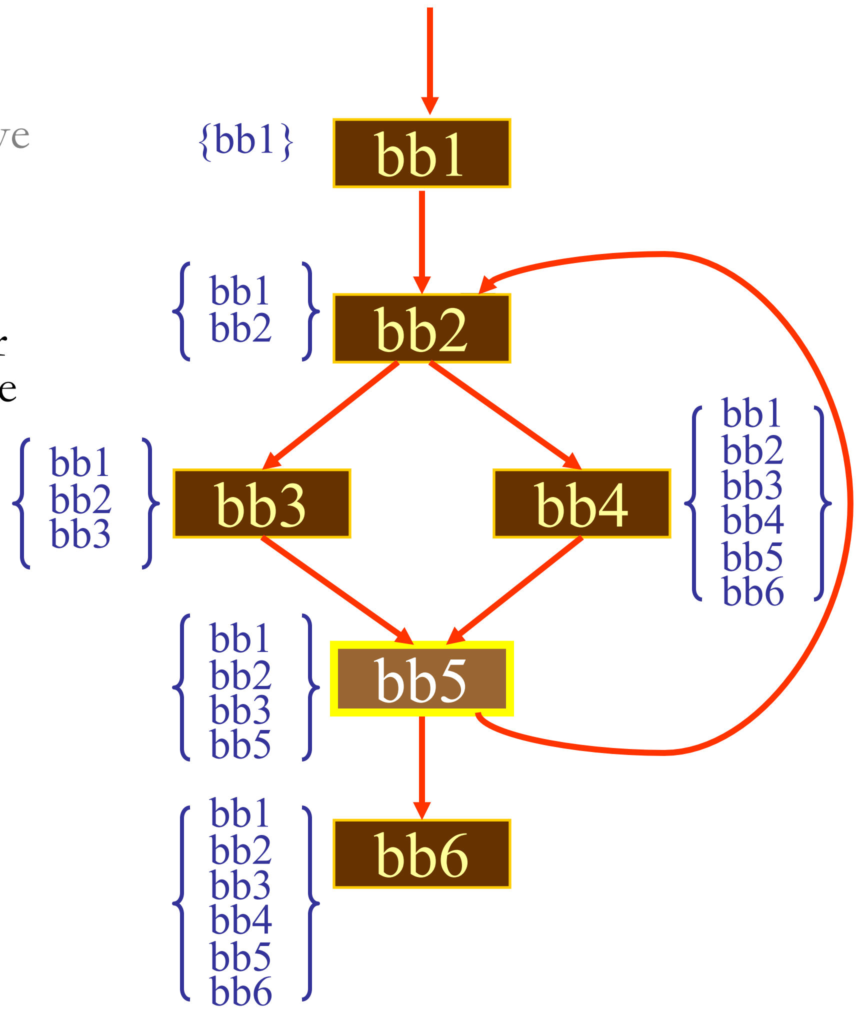
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



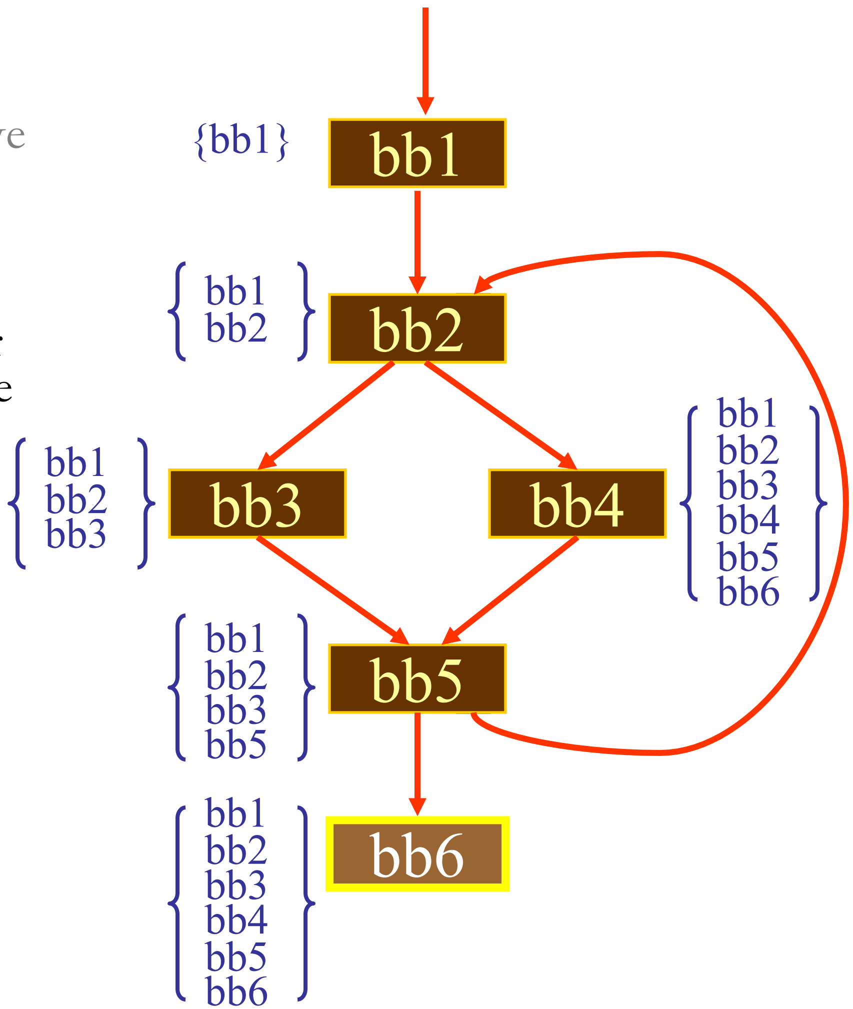
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



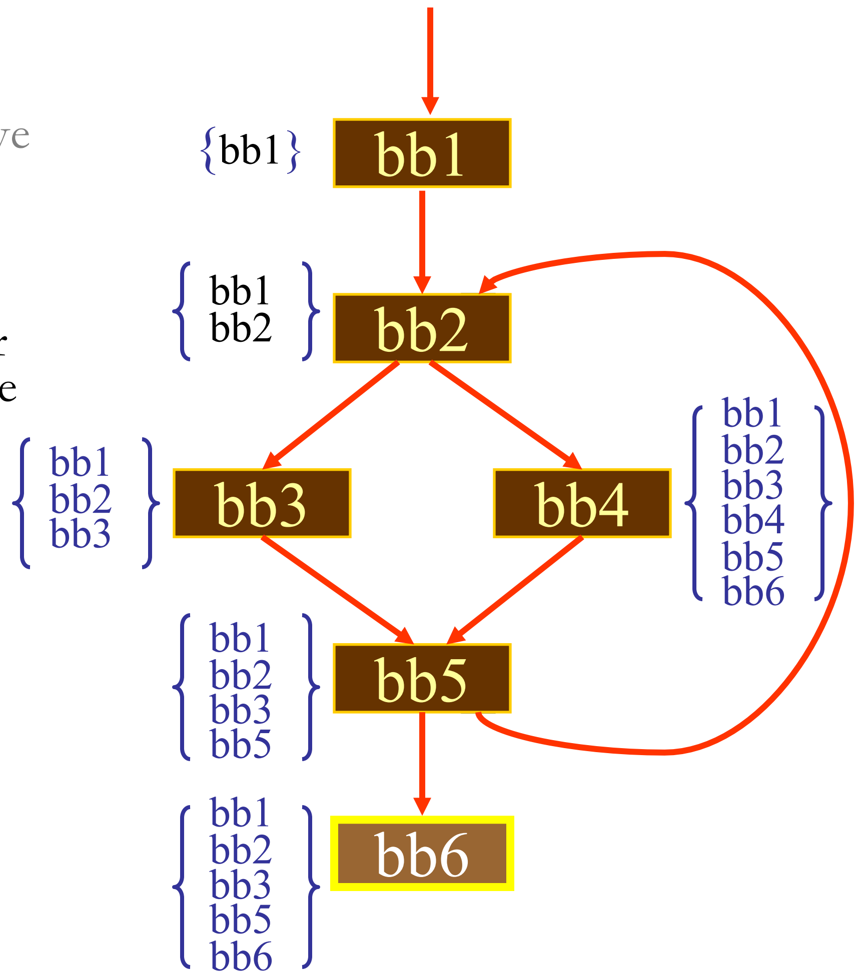
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



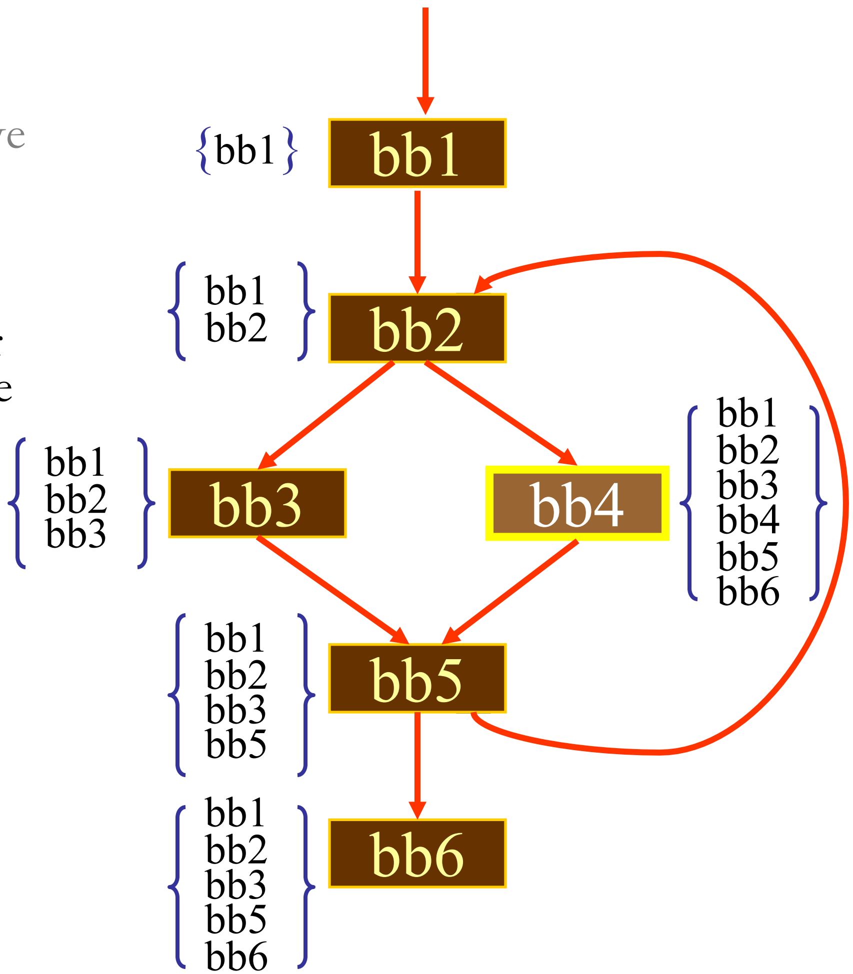
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



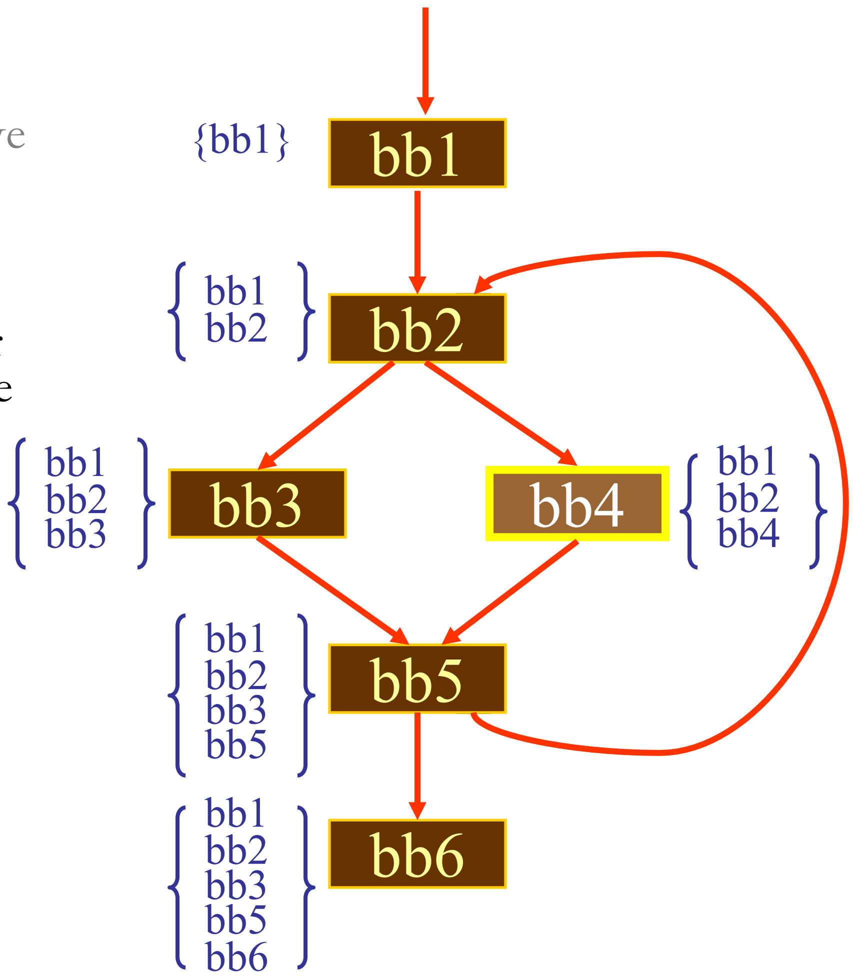
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



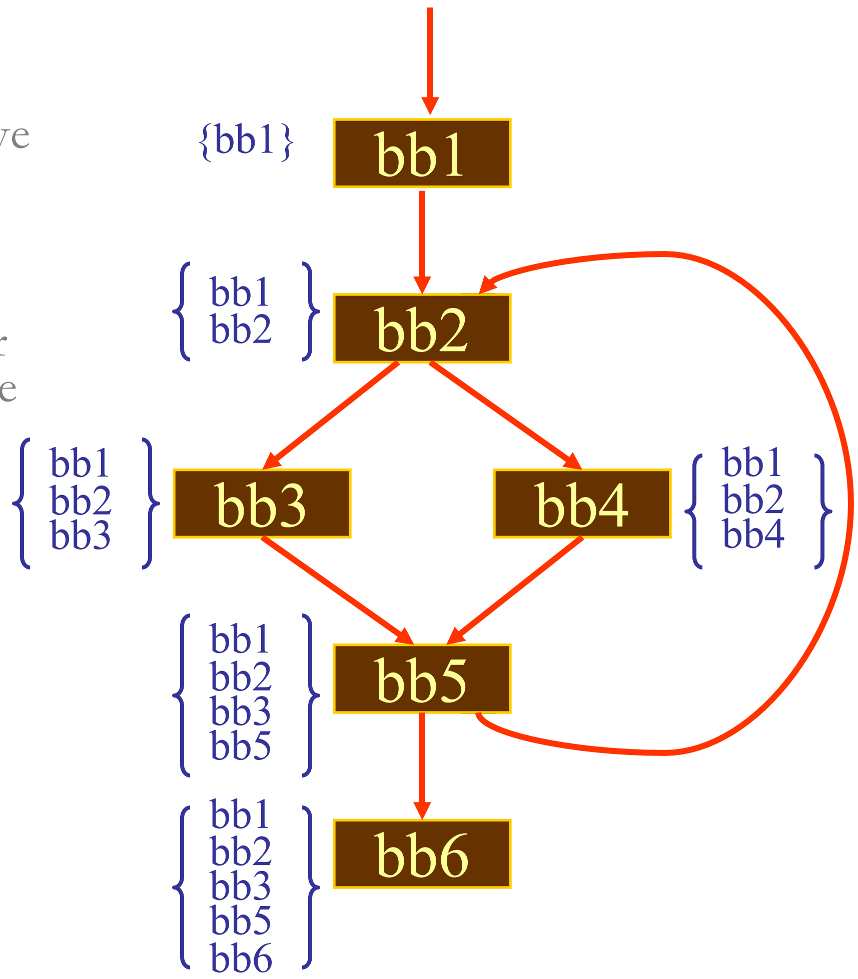
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



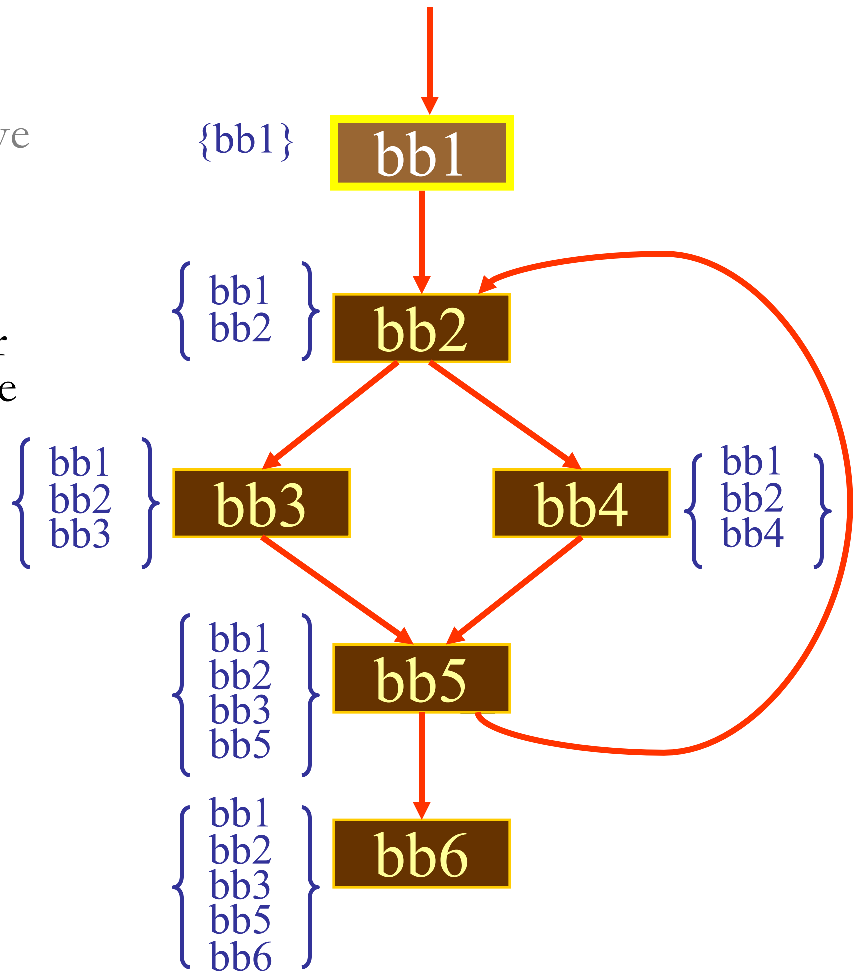
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



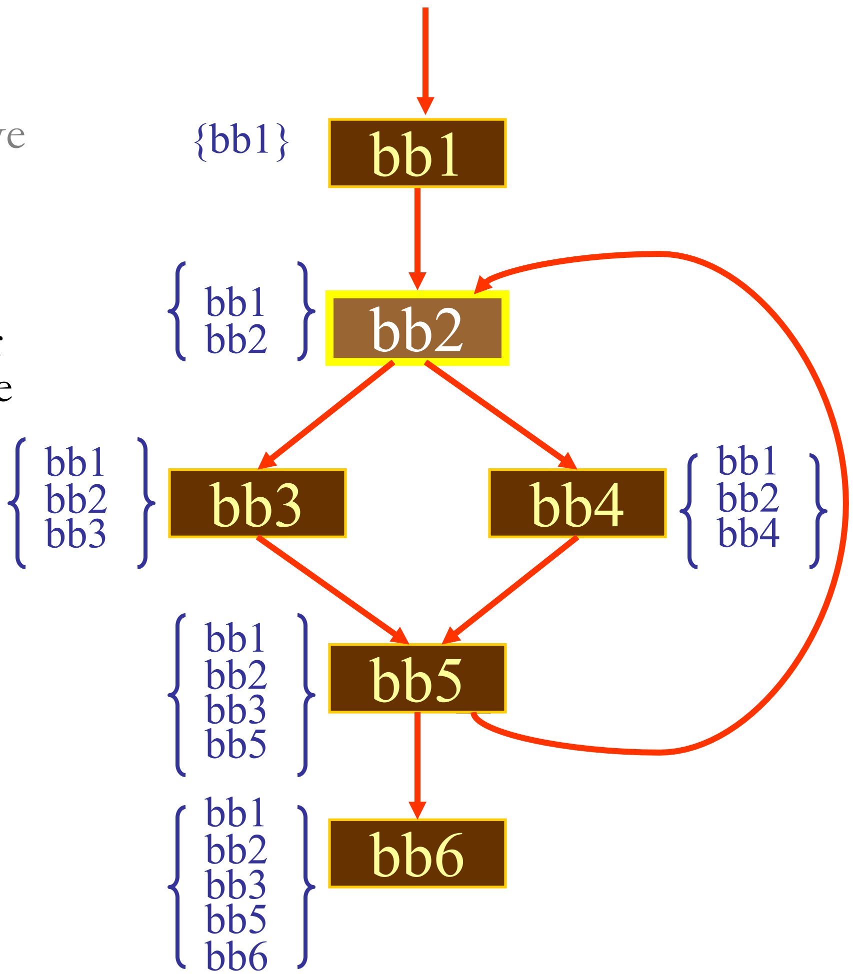
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



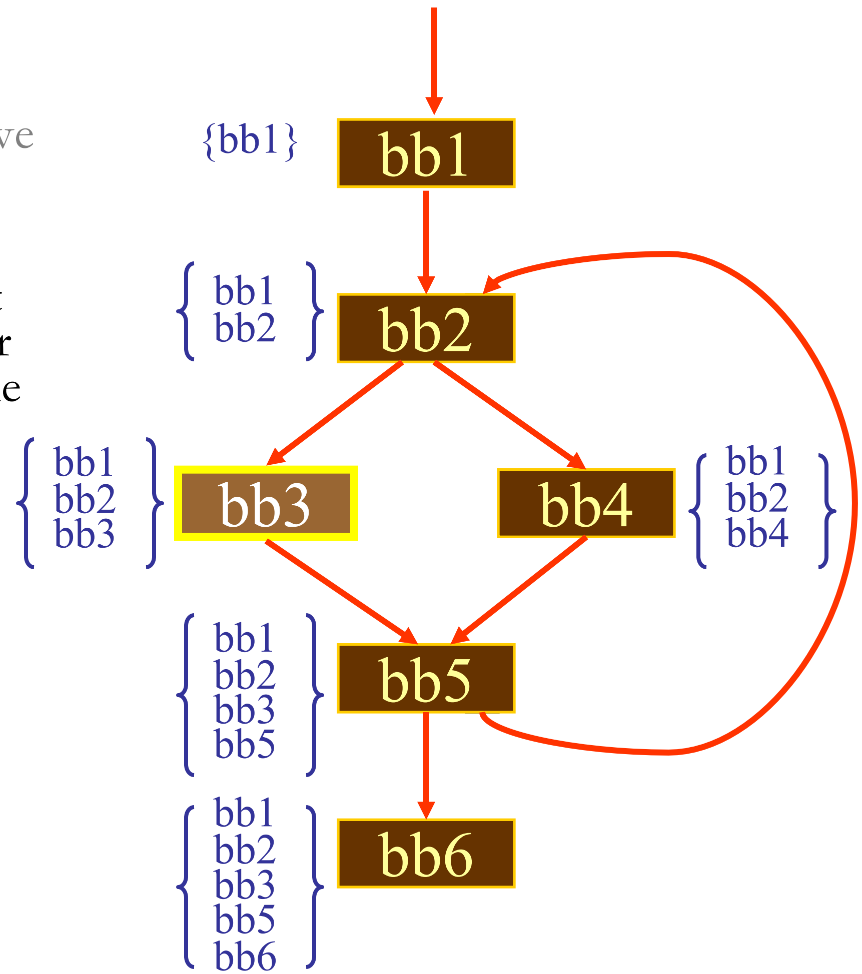
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



Computing Dominators

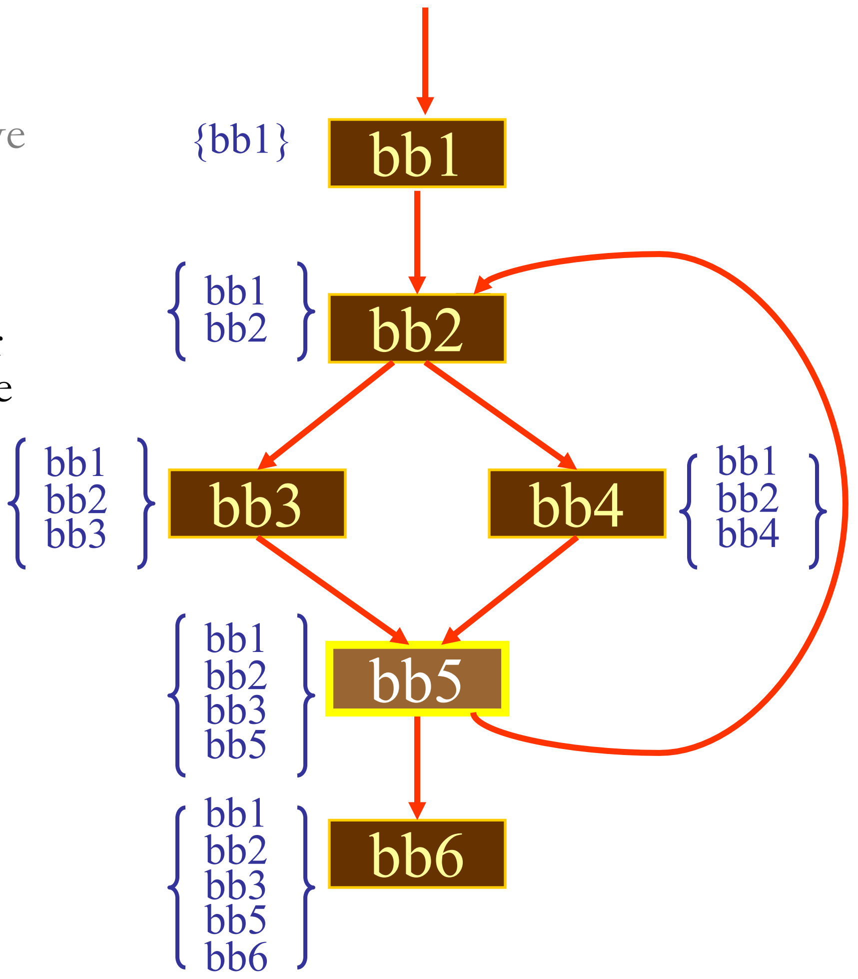
- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



Pedro C. Diniz
pcd@fe.up.pt

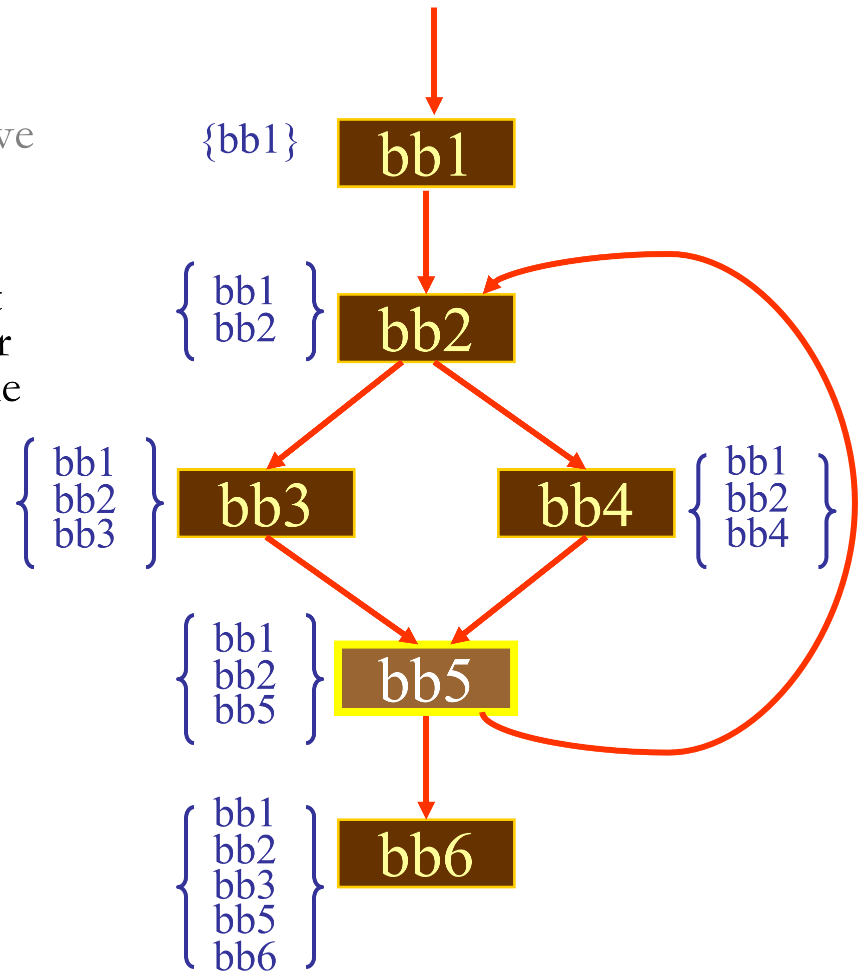
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



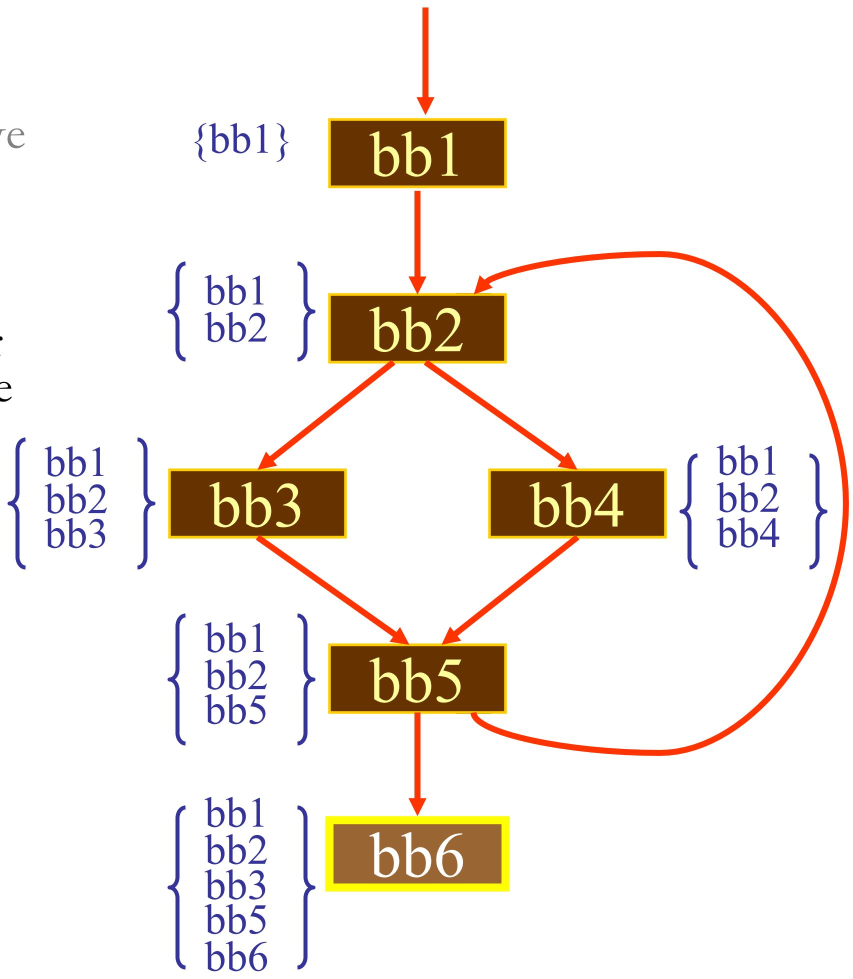
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



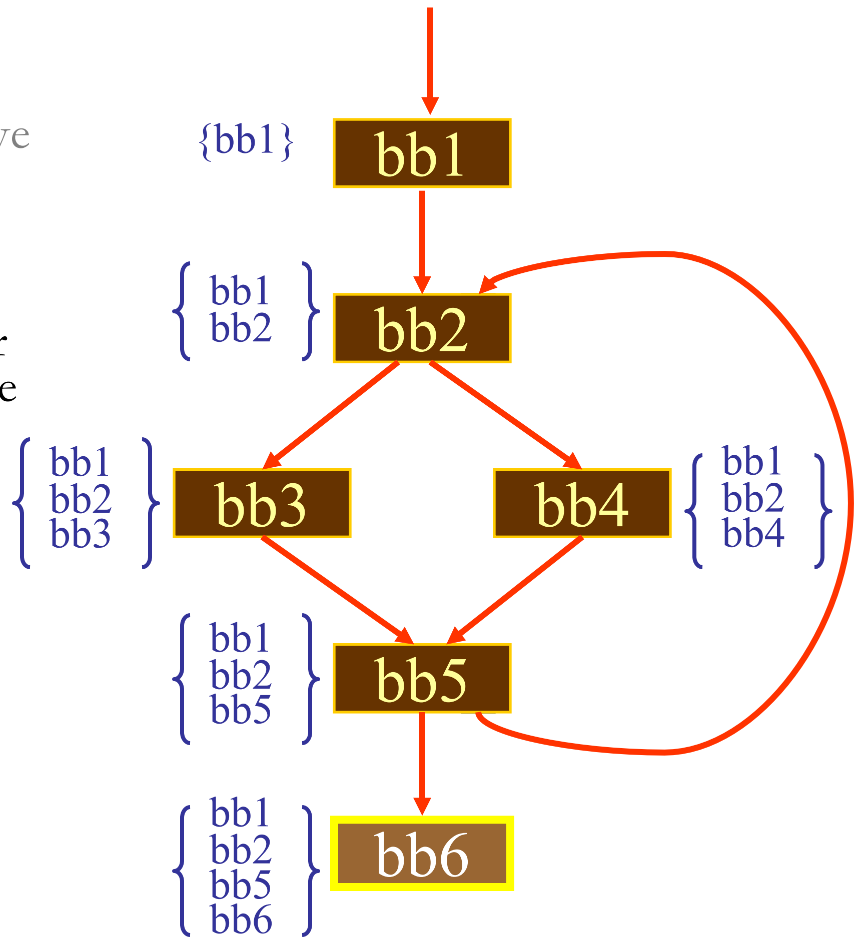
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



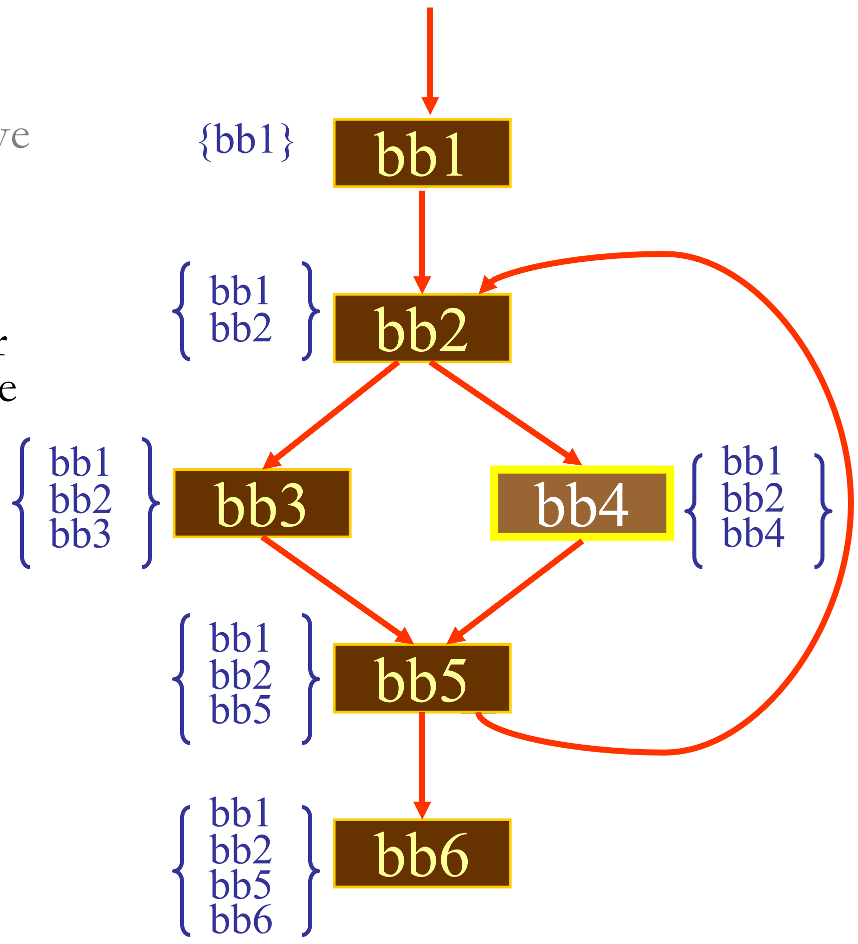
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



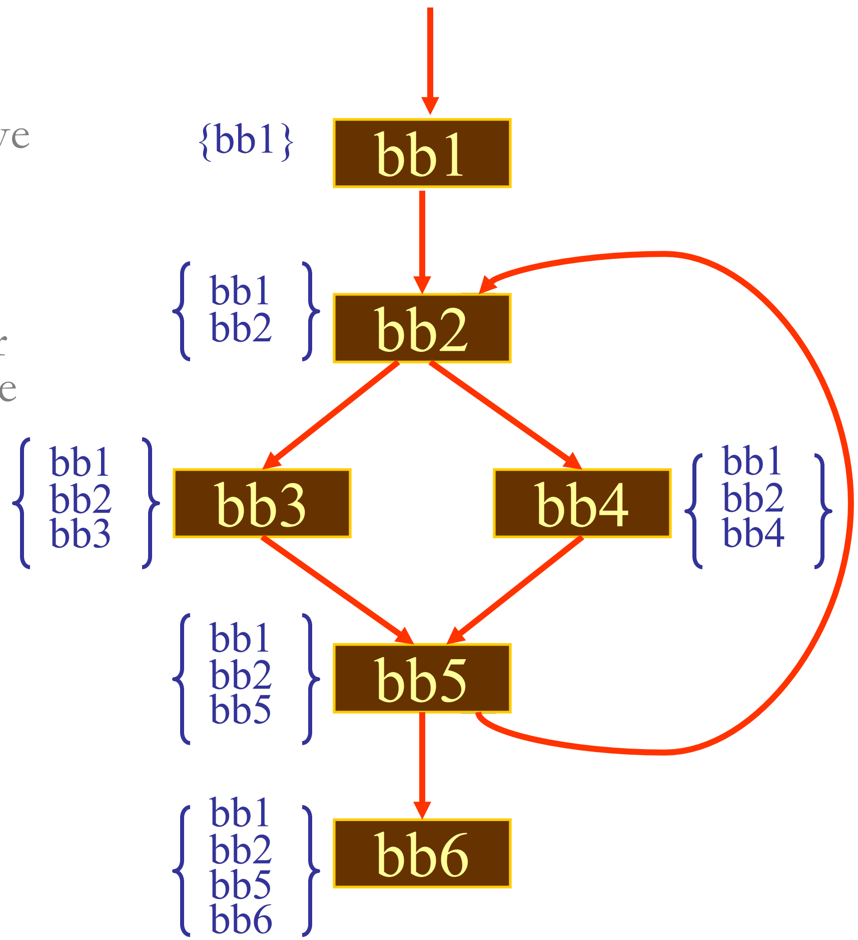
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



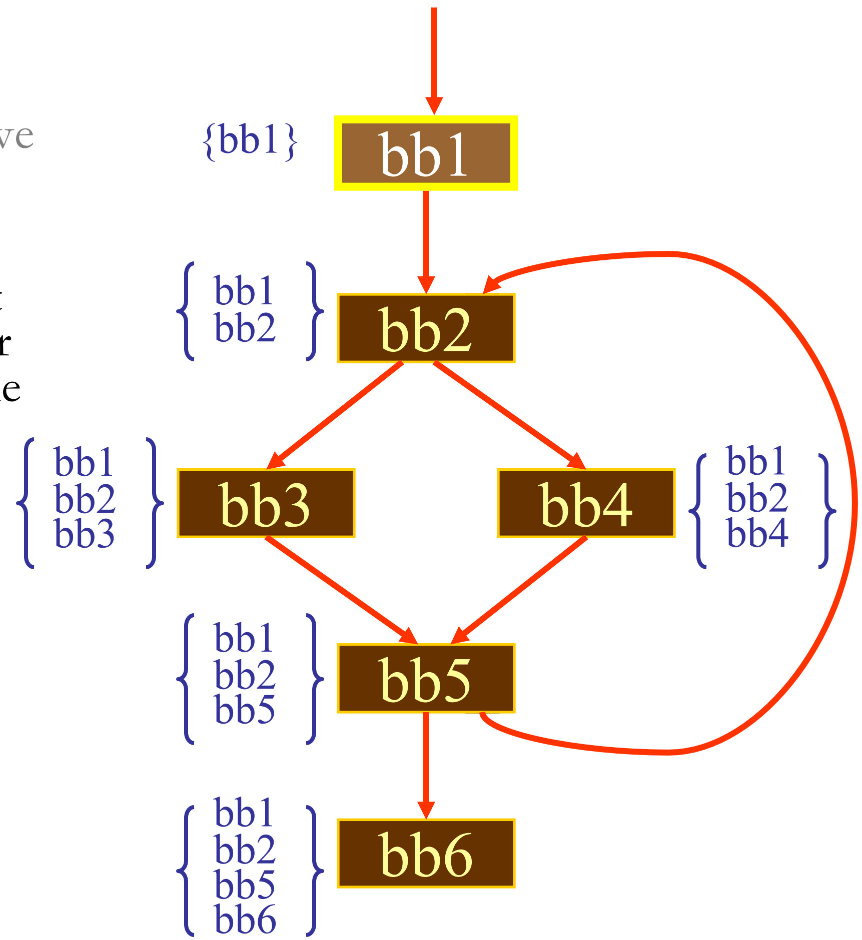
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



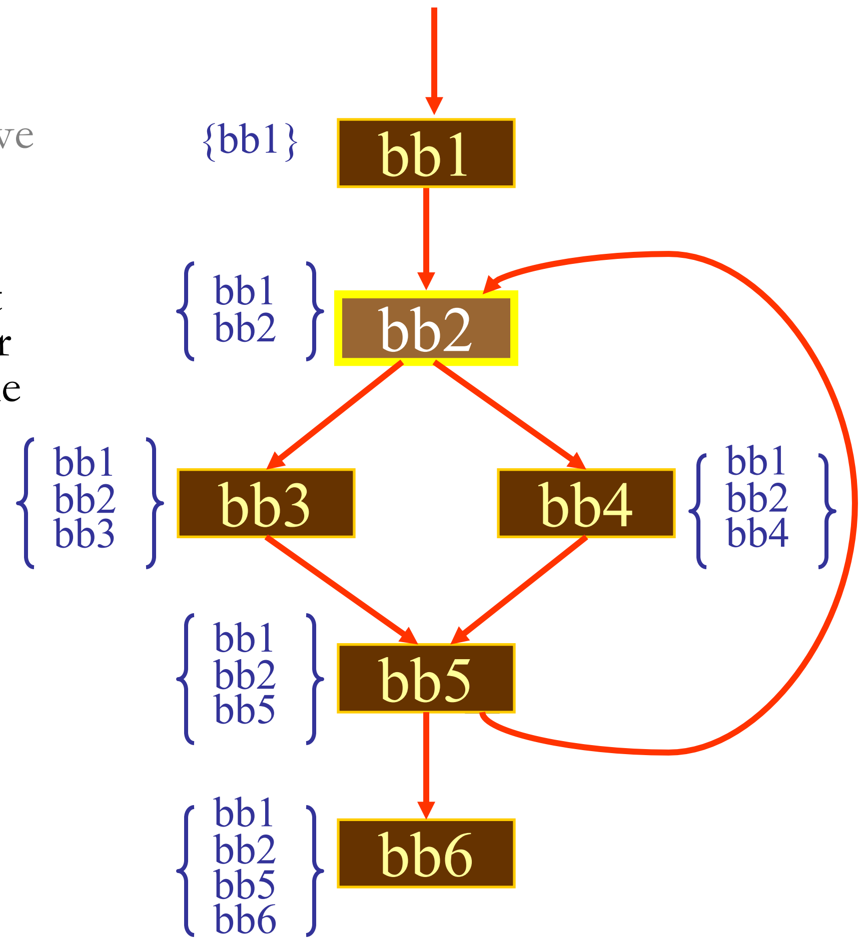
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



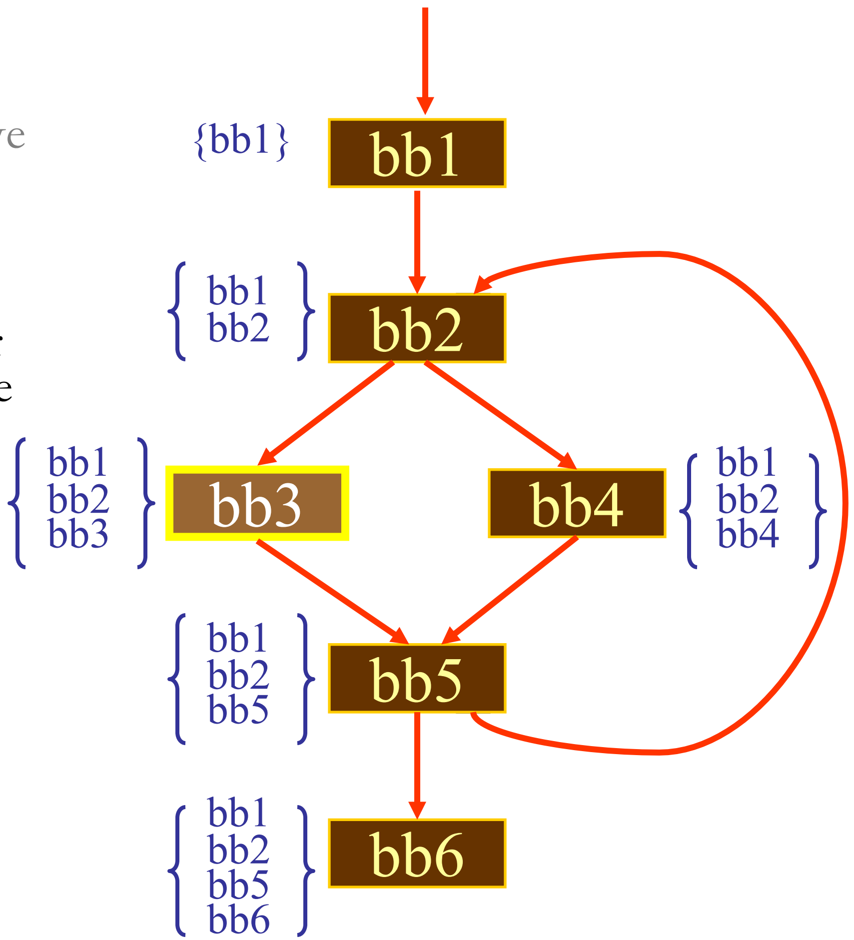
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



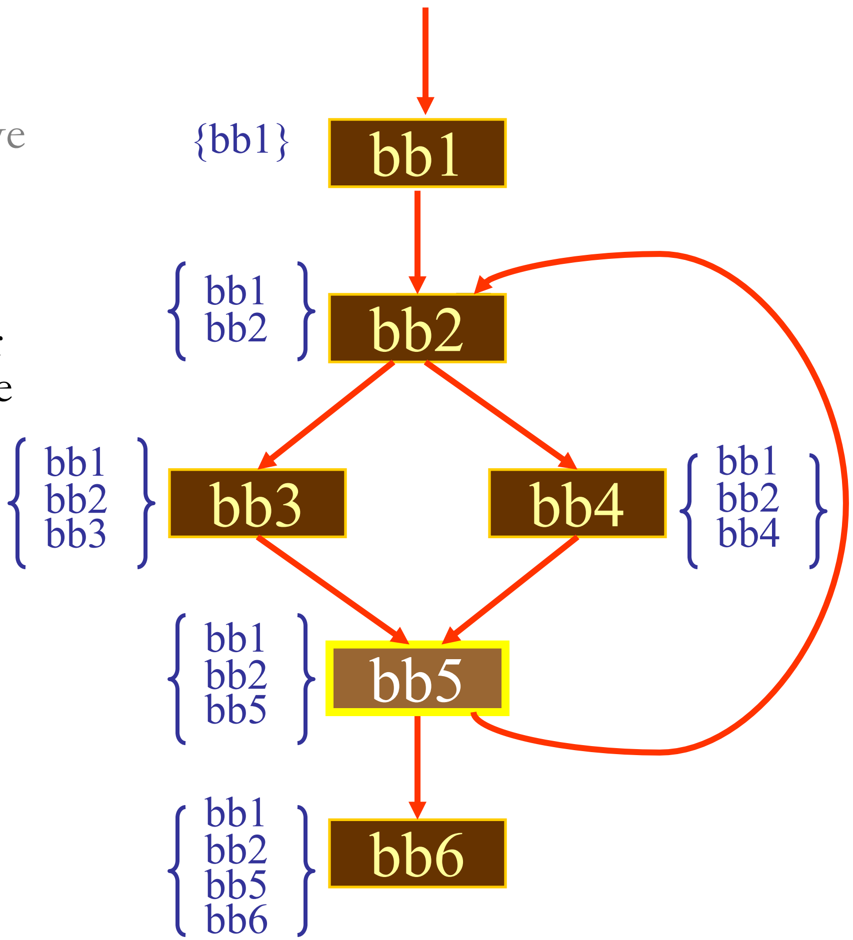
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



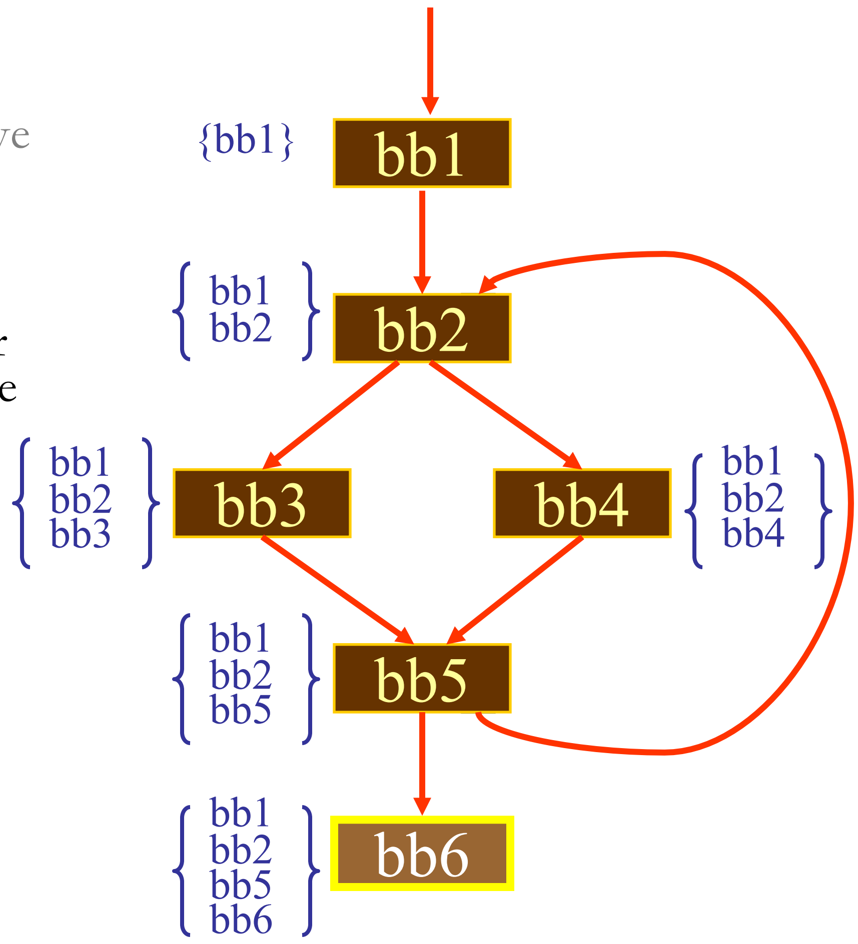
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



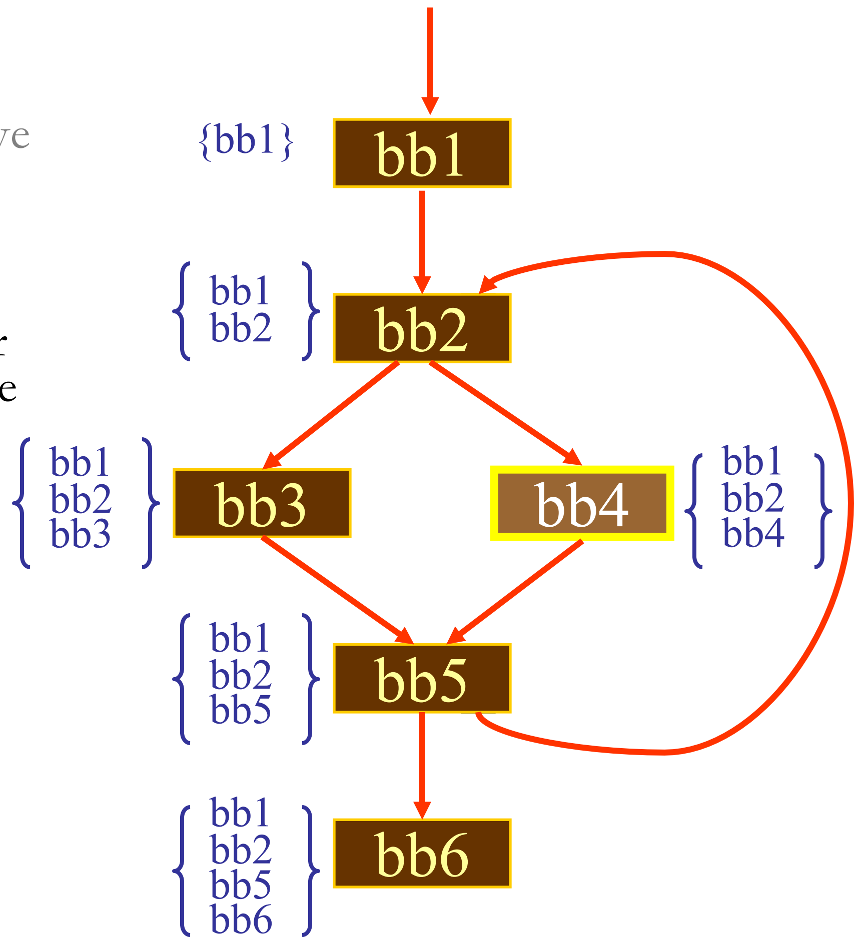
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



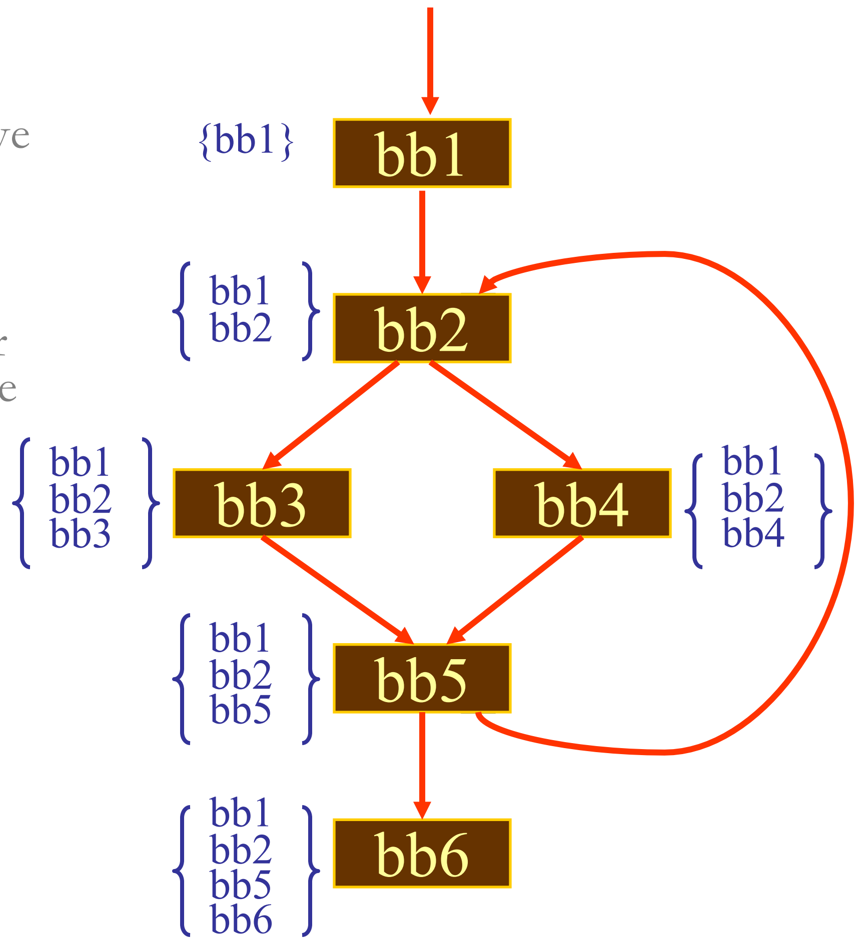
Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



Computing Dominators

- Algorithm
  - Make dominator set of the entry node has itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



Computing Dominators

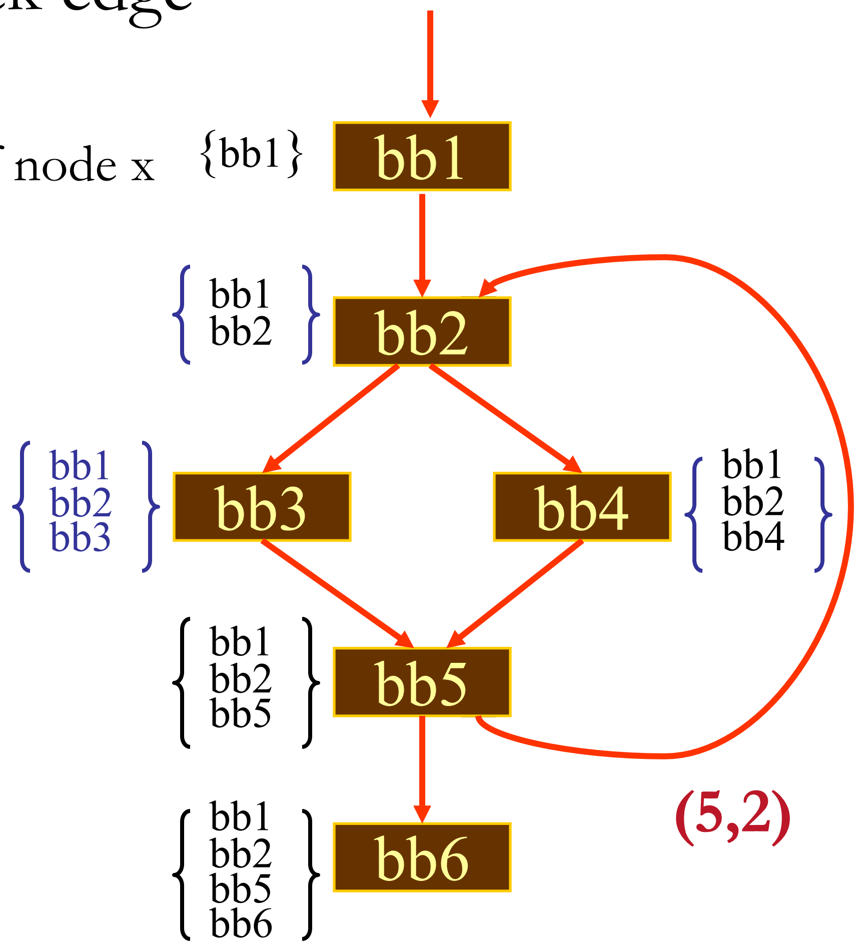
- What we just witnessed was an *iterative* data-flow analysis algorithm in action:
  - Initialize all the nodes to a given value
  - Visit nodes in some order
  - Calculate the node's value
  - Repeat until no value changes (fixed-point)
- Will talk about this in the coming lectures

What is a Back Edge?

- An edge (x, y) ∈ E is a back edge iff y dom x
  - is node y in the dominator set of node x

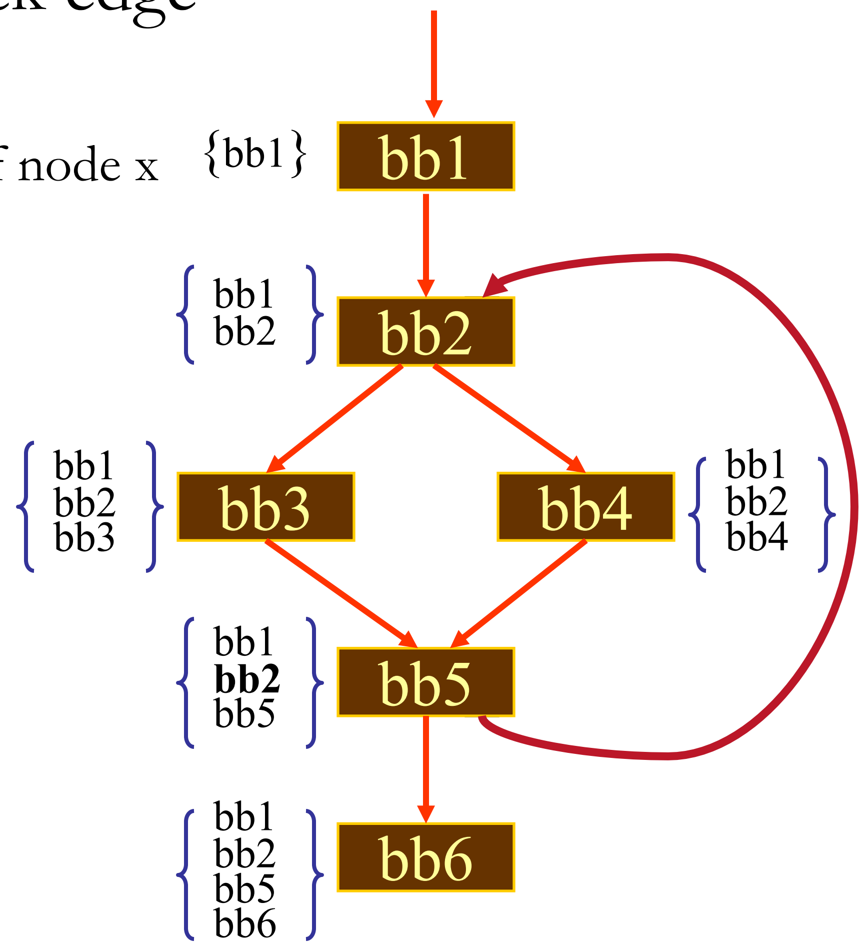
What is a Back Edge?

- An edge (x, y) ∈ E is a back edge iff y dom x
  - is node y in the dominator set of node x



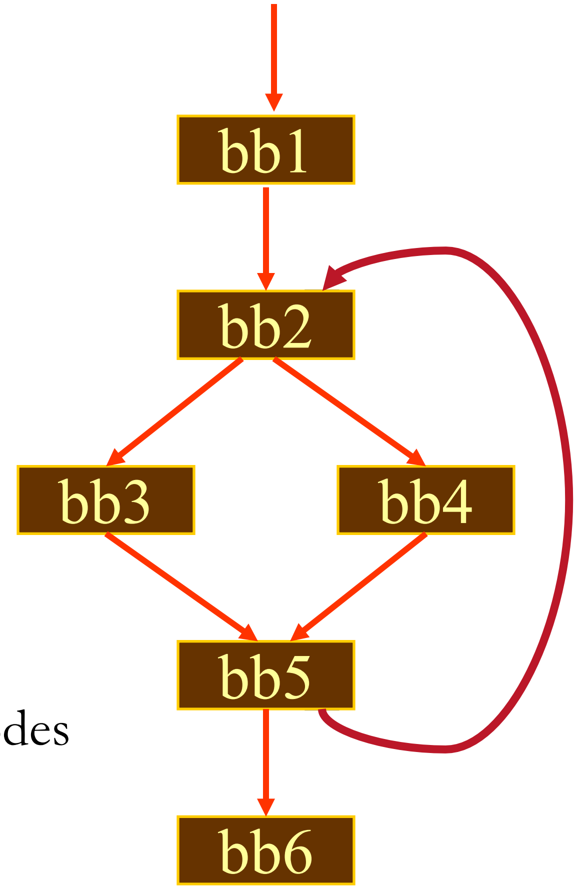
What is a Back Edge?

- An edge (x, y) ∈ E is a back edge iff y dom x
  - is node y in the dominator set of node x



Natural Loop

- In a CFG a Back Edge induces a Natural Loop
- Finding the Nodes of a Loop
  - Given Back Edge (s,d)
  - Traverse Backwards (against flow) from d
  - Until Reaching s
  - Collected nodes in CFG form Natural Loop



- In the Example: back edge is (5,2)
  - Natural Loop : { 2, 5, 3, 4 } why?
  - Trace Back Edge backward and collect all nodes
  - Until you reach head of the Back Edge

Summary

- Overview of Optimizations
- Control-Flow Analysis
- Dominators
- Back Edges and Natural Loops