Lab Basic Blocks

J. Nelson Amaral

A Basic Block

Only the first instruction of a basic block can be the target of a branch.

Execution can only start here
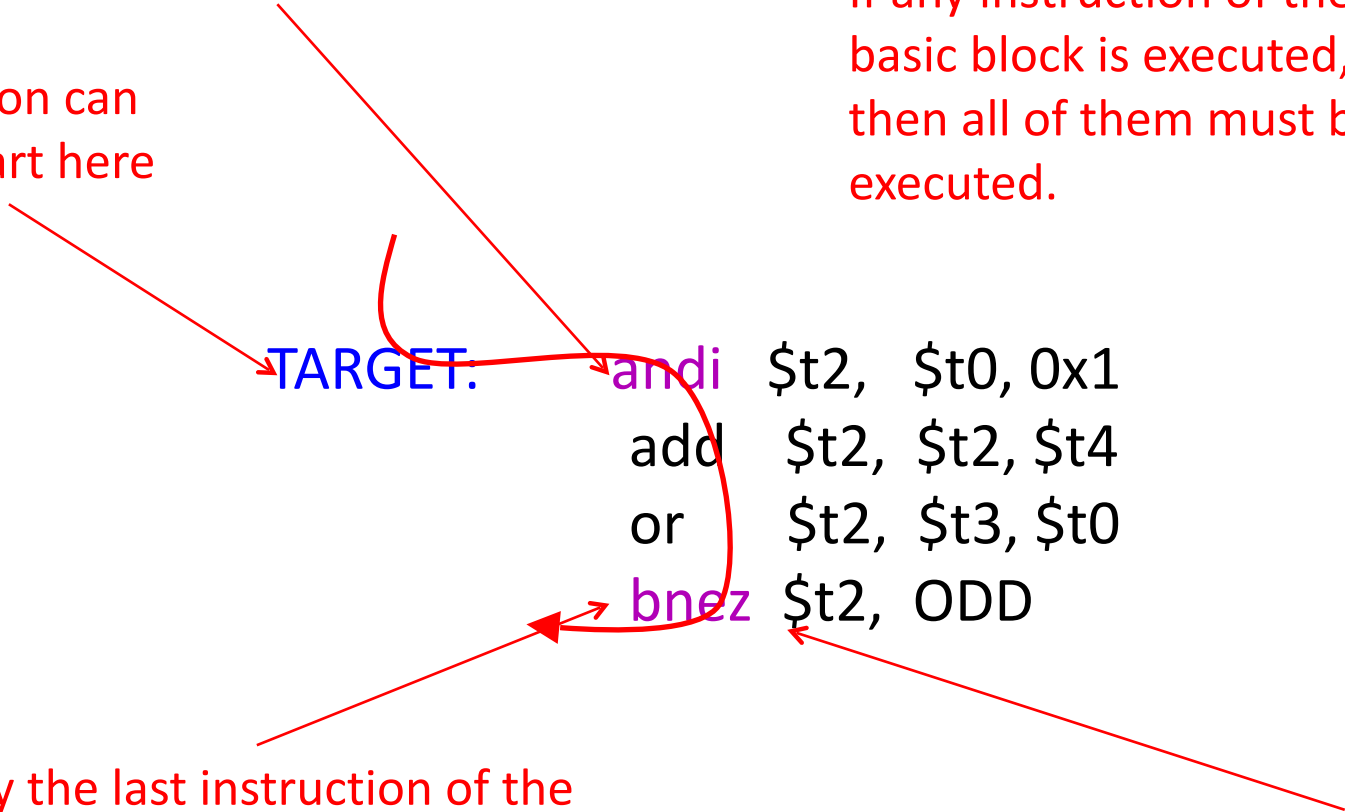
If any instruction of the basic block is executed, then all of them must be executed.

```
TARGET:   andi   $t2,  $t0, 0x1
          add    $t2,  $t2, $t4
          or     $t2,  $t3, $t0
          bnez   $t2,  ODD
```

Only the last instruction of the basic block can be a branch or a jump.

Execution can only stop here

A Program Example

```c
int odd_series(int p){
int j=0;
 for (int i=0 ; i<p; i++){
   if(i%2)
     j += i;
   else
     j++;
 }
 return j;
}
```

```mips
odd_series:
        li      $t0, 0          # i <-- 0
        li      $v0, 0          # j <-- 0
        blez    $a0, DONE       # if (p <= 0) goto DONE
LOOP:
        andi    $t2, $t0, 0x1   # $t2 <-- bit0 of i
        bnez    $t2 ODD         # if i is odd goto ODD
        add     $v0, $v0, $t0   # j <-- j+i
        j       REINIT
ODD:
        add     $v0, $v0, 1     # j <-- j+1
REINIT:
        add     $t0, $t0, 1     # i <-- i+1
        blt     $t0, $a0, LOOP  # if i<p goto LOOP
DONE:
        jr      $ra
```

Finding Leaders

Rule 1: The first statement of a procedure is a leader.

```
odd_series:
        li      $t0, 0          # i <-- 0
        li      $v0, 0          # j <-- 0
        blez    $a0, DONE       # if (p <= 0) goto DONE
LOOP:
        andi    $t2, $t0, 0x1   # $t2 <-- bit0 of i
        bnez    $t2 ODD         # if i is odd goto ODD
        add     $v0, $v0, $t0   # j <-- j+i
        j       REINIT
ODD:
        add     $v0, $v0, 1     # j <-- j+1
REINIT:
        add     $t0, $t0, 1     # i <-- i+1
        blt     $t0, $a0, LOOP  # if i<p goto LOOP
DONE:
        jr      $ra
```

Rule 2: Any target of a branch or jump is a leader.

```
odd_series:
        li      $t0, 0          # i <-- 0
        li      $v0, 0          # j <-- 0
        blez    $a0, DONE       # if (p <= 0) goto DONE
LOOP:
        andi    $t2, $t0, 0x1   # $t2 <-- bit0 of i
        bnez    $t2 ODD         # if i is odd goto ODD
        add     $v0, $v0, $t0   # j <-- j+i
        j       REINIT
ODD:
        add     $v0, $v0, 1     # j <-- j+1
REINIT:
        add     $t0, $t0, 1     # i <-- i+1
        blt     $t0, $a0, LOOP  # if i<p goto LOOP
DONE:
        jr      $ra
```

Rule 3: Any instruction that follows a branch or jump is a leader.

```
odd_series:
        li      $t0, 0          # i <-- 0
        li      $v0, 0          # j <-- 0
        blez    $a0, DONE       # if (p <= 0) goto DONE
LOOP:
        andi    $t2, $t0, 0x1   # $t2 <-- bit0 of i
        bnez    $t2 ODD         # if i is odd goto ODD
        add     $v0, $v0, $t0   # j <-- j+i
        j       REINIT
ODD:
        add     $v0, $v0, 1     # j <-- j+1
REINIT:
        add     $t0, $t0, 1     # i <-- i+1
        blt     $t0, $a0, LOOP  # if i<p goto LOOP
DONE:
        jr      $ra
```

Instruction	Encoding
beq $s, $t, offset	0001 00ss ssst tttt oooo oooo oooo oooo
bgez $s, offset	0000 01ss sss0 0001 oooo oooo oooo oooo
bgezal $s, offset	0000 01ss sss1 0001 oooo oooo oooo oooo
bgtz $s, offset	0001 11ss sss0 0000 oooo oooo oooo oooo
blez $s, offset	0001 10ss sss0 0000 oooo oooo oooo oooo
bltz $s, offset	0000 01ss sss0 0000 oooo oooo oooo oooo
bltzal $s, offset	0000 01ss sss1 0000 oooo oooo oooo oooo
bne $s, $t, offset	0001 01ss ssst tttt oooo oooo oooo oooo
j target	0000 10aa aaaa aaaa aaaa aaaa aaaa aaaa
jal target	0000 11aa aaaa aaaa aaaa aaaa aaaa aaaa
jr $s	0000 00ss sss0 0000 0000 0000 0000 1000

Finding the Leaders – Lab #2

Instruction	Encoding
beq $s, $t, offset	0001 00ss ssst tttt oooo oooo oooo oooo
bgez $s, offset	0000 01ss sss0 0001 oooo oooo oooo oooo
bgezal $s, offset	0000 01ss sss1 0001 oooo oooo oooo oooo
bgtz $s, offset	0001 11ss sss0 0000 oooo oooo oooo oooo
blez $s, offset	0001 10ss sss0 0000 oooo oooo oooo oooo
bltz $s, offset	0001 10ss sss0 0000 oooo oooo oooo oooo
bne $s, $t, offset	0001 01ss ssst tttt oooo oooo oooo oooo
j target	0000 10aa aaaa aaaa aaaa aaaa aaaa aaaa
jal target	0000 11aa aaaa aaaa aaaa aaaa aaaa aaaa
jr $s	0000 00ss sss0 0000 0000 0000 0000 1000

Given the Leaders,
How do we get the Basic Blocks?

A basic block is the leader followed by all subsequent instructions up to, but not including, the next leader.

```
odd_series:
        li      $t0, 0          # i <-- 0
        li      $v0, 0          # j <-- 0
        blez    $a0, DONE       # if (p <= 0) goto DONE
LOOP:
        andi    $t2, $t0, 0x1   # $t2 <-- bit0 of i
        bnez    $t2 ODD         # if i is odd goto ODD
        add     $v0, $v0, $t0   # j <-- j+i
        j       REINIT
ODD:
        add     $v0, $v0, 1     # j <-- j+1
REINIT:
        add     $t0, $t0, 1     # i <-- i+1
        blt     $t0, $a0, LOOP  # if i<p goto LOOP
DONE:
        jr      $ra
```

A basic block is the leader followed by all subsequent instructions up to, but not including, the next leader.

```
odd_series:
        li      $t0, 0          # i <-- 0
        li      $v0, 0          # j <-- 0
        blez    $a0, DONE       # if (p <= 0) goto DONE
LOOP:
        andi    $t2, $t0, 0x1   # $t2 <-- bit0 of i
        bnez    $t2 ODD         # if i is odd goto ODD
        add     $v0, $v0, $t0   # j <-- j+i
        j       REINIT
ODD:
        add     $v0, $v0, 1     # j <-- j+1
REINIT:
        add     $t0, $t0, 1     # i <-- i+1
        blt     $t0, $a0, LOOP  # if i<p goto LOOP
DONE:
        jr      $ra
```

The Actual Code

```
0  [10010000] 34080000   ori $8, $0, 0                    ; 52: li $t0, 0 # i
   [10010004] 34020000   ori $2, $0, 0                    ; 53: li $v0, 0 # j
   [10010008] 18800009   blez $4 36 [DONE-0x10010008]; 54: blez $a0, DONE #
   if (p

1  [1001000c] 310a0001   andi $10, $0, 1                  ; 56: andi $t2, $t0, 0, 1 #
   $t2
   [10010010] 15400003   bne $10, $
```

```
2  [10010014] 00481020   add $2, $2
   [10010018] 08100029   j 0x100100
```

```
3  [1001001c] 20420001   addi $2, $
```

```
4  [10010020] 21080001   addi $8, $
   [10010024] 0104082a   slt $1, $8
   if i
   [10010028] 1420fff9   bne $1, $0
```

```
5  [1001002c] 03e00008   jr $31                           ; 66: jr $ra
```

6 block(s) found.
Block Leader: 0x10010000, Size: 3
Block Leader: 0x1001000C, Size: 2
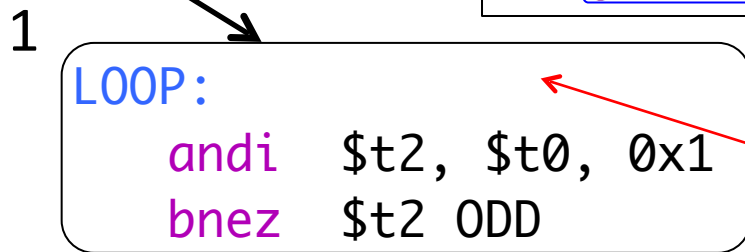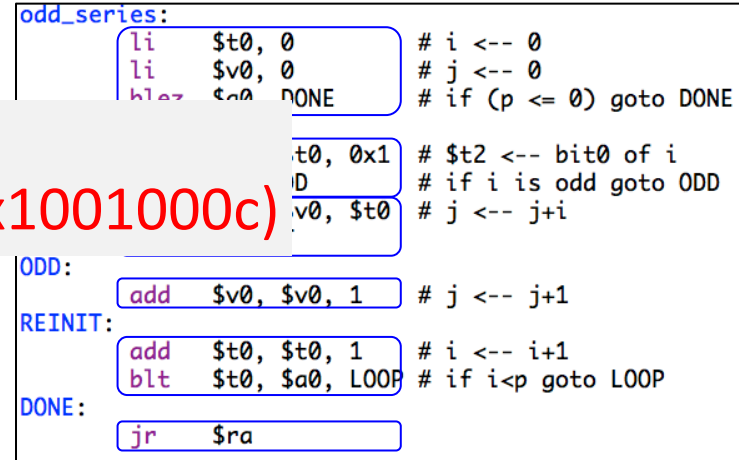Block Leader: 0x10010014, Size: 2
Block Leader: 0x1001001C, Size: 1
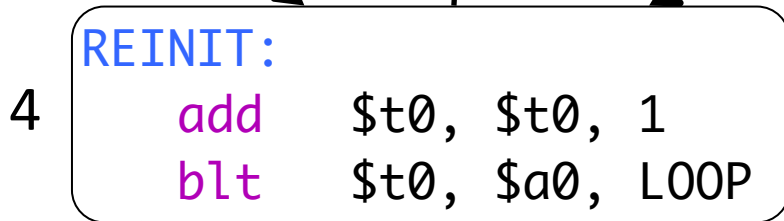Block Leader: 0x10010020, Size: 3
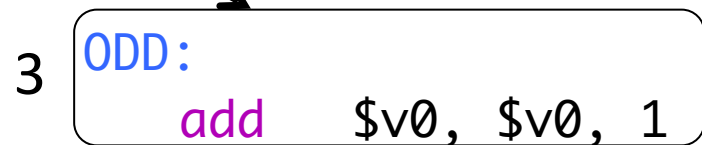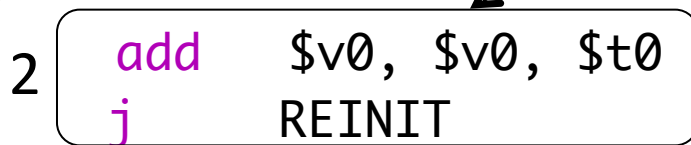Block Leader: 0x1001002C, Size: 1

Control Flow Graphs

0

```
odd_series:
    li      $t0, 0
    li      $v0, 0
    blez    $a0, DONE
```
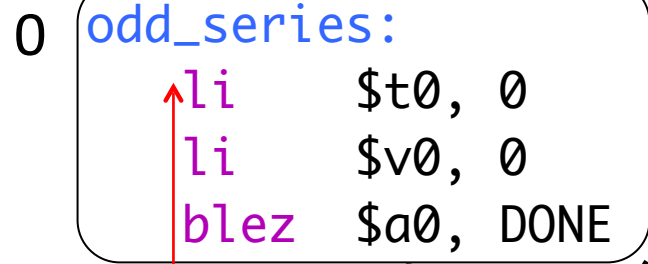
This is the edge
(0x10010000, 0x1001000c)

This is the
basic block at
address
0x10010000

```
odd_series:
    li      $t0, 0          # i <-- 0
    li      $v0, 0          # j <-- 0
    blez    $a0, DONE       # if (p <= 0) goto DONE

        t0, 0x1             # $t2 <-- bit0 of i
        DD                  # if i is odd goto ODD
        v0, $t0             # j <-- j+i

ODD:
    add     $v0, $v0, 1     # j <-- j+1
REINIT:
    add     $t0, $t0, 1     # i <-- i+1
    blt     $t0, $a0, LOOP  # if i<p goto LOOP
DONE:
    jr      $ra
```

1

```
LOOP:
    andi    $t2, $t0, 0x1
    bnez    $t2 ODD
```

This is the
basic block at
address
0x1001000c

2

```
add     $v0, $v0, $t0
j       REINIT
```

3

```
ODD:
    add     $v0, $v0, 1
```

4

```
REINIT:
    add     $t0, $t0, 1
    blt     $t0, $a0, LOOP
```

5

```
DONE:
    jr      $ra
```
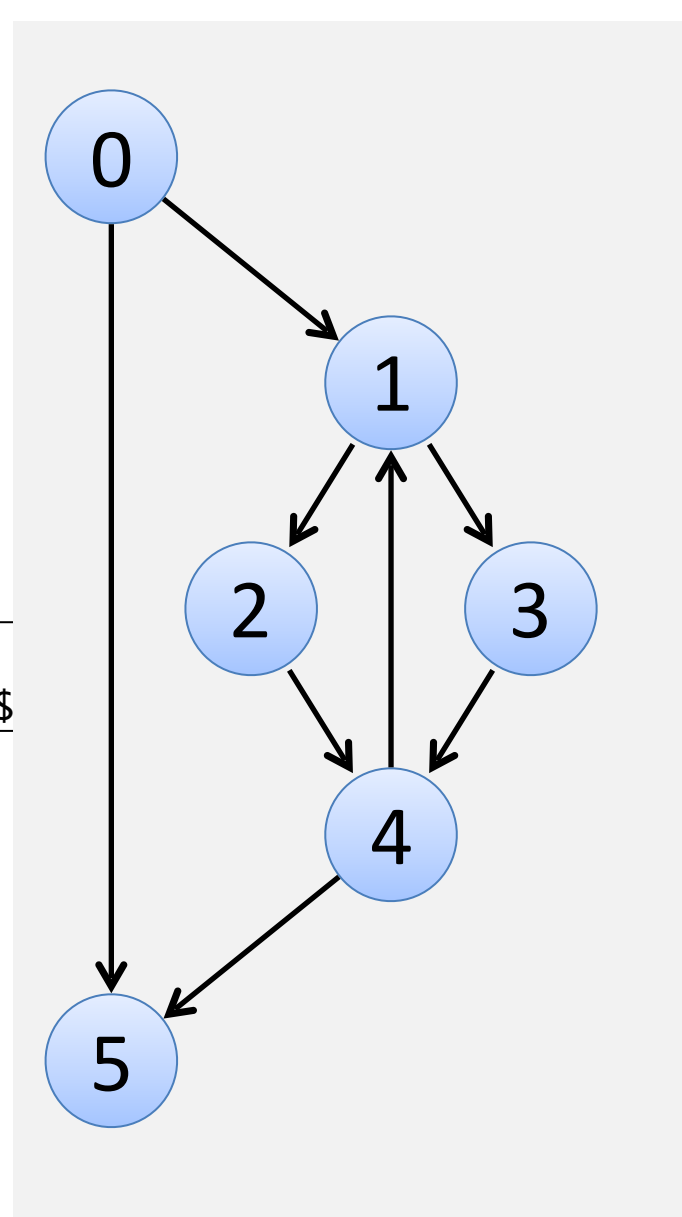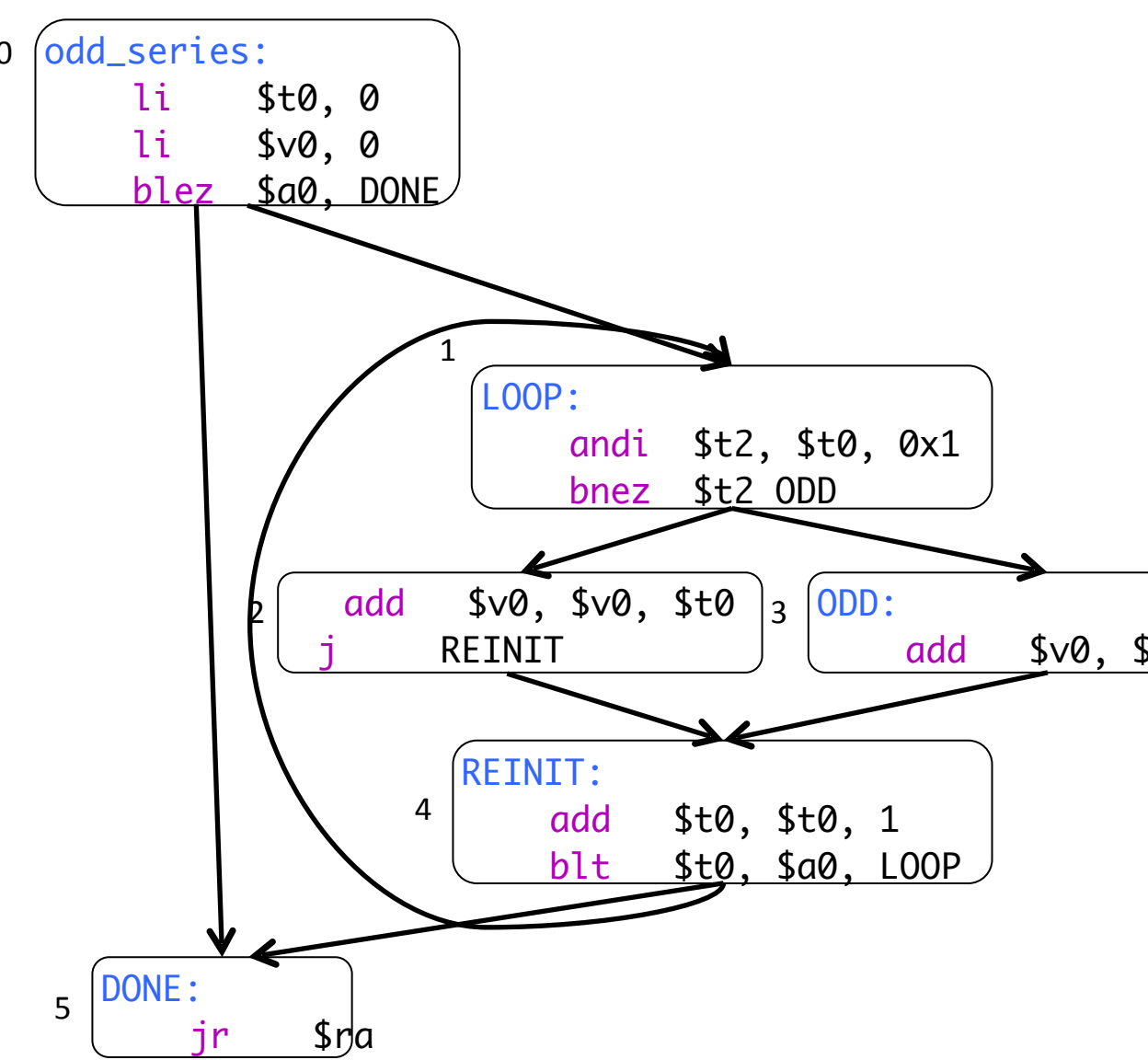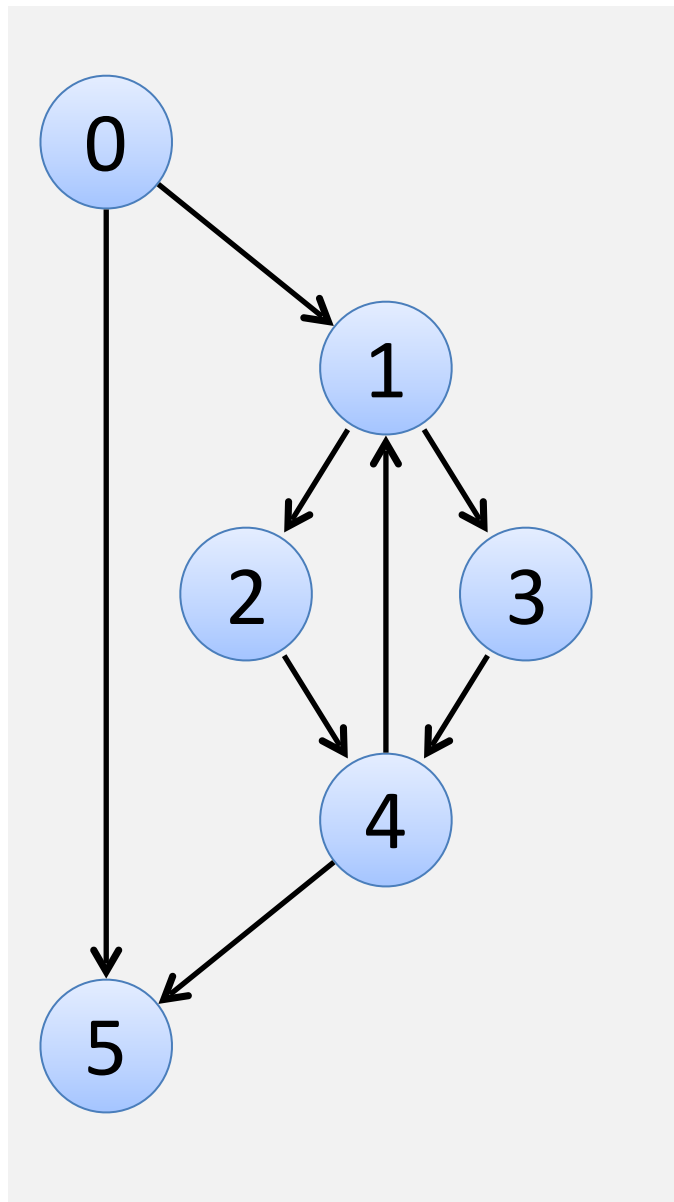
The Actual Code

```
0   [10010000] 34080000  ori $8, $0, 0              ; 52: li $t0, 0 # i
    [10010004] 34020000  ori $2, $0, 0              ; 53: li $v0, 0 # j
    [10010008] 18800009  blez $4 36 [DONE-0x10010008]; 54: blez $a0, DONE #
    if (p

1   [1001000c] 310a0001  andi $10, $8, 1            ; 56: andi $t2, $t0, 0x1 #
    $t2
    [10010010] 15400003  bne $10, $0, 12

2   [10010014] 00481020  add $2, $2, $8
    [10010018] 08100029  j 0x10010020 [RE

3   [1001001c] 20420001  addi $2, $2, 1

4   [10010020] 21080001  addi $8, $8, 1
    [10010024] 0104082a  slt $1, $8, $4
    if i
    [10010028] 1420fff9  bne $1, $0, -28

5   [1001002c] 03e00008  jr $31
```
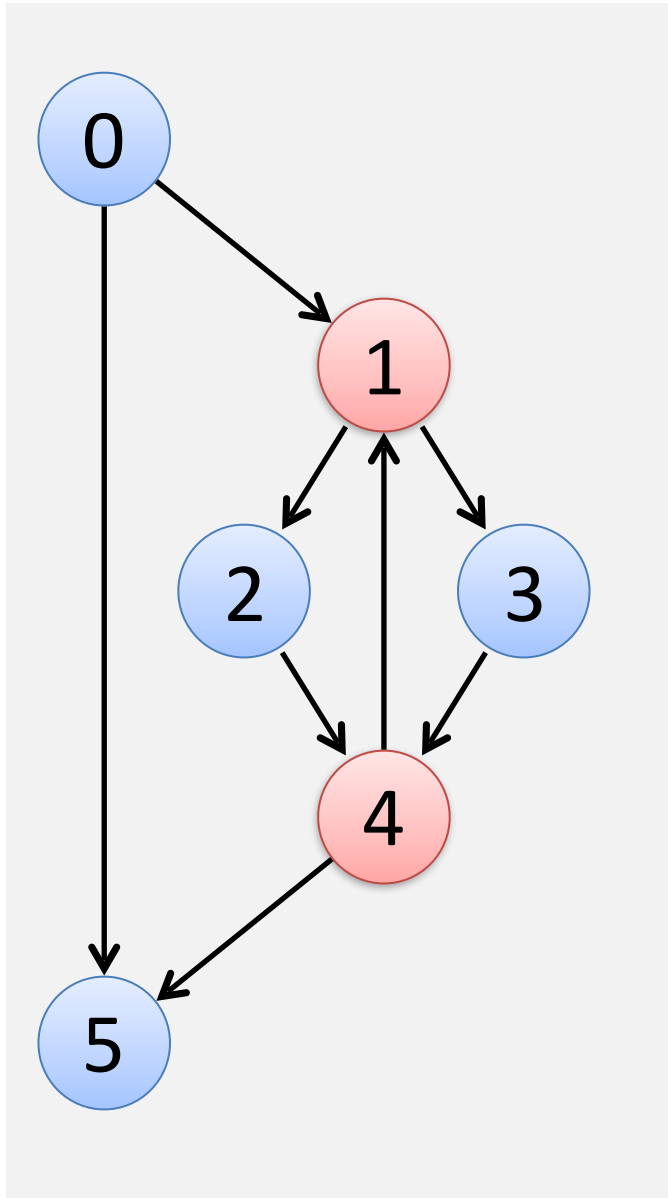
Edges:

0x10010000 --> 0x1001000C

0x10010000 --> 0x1001002C

0x1001000C --> 0x10010014

0x1001000C --> 0x1001001C

0x10010014 --> 0x10010020

0x1001001C --> 0x10010020

0x10010020 --> 0x1001000C
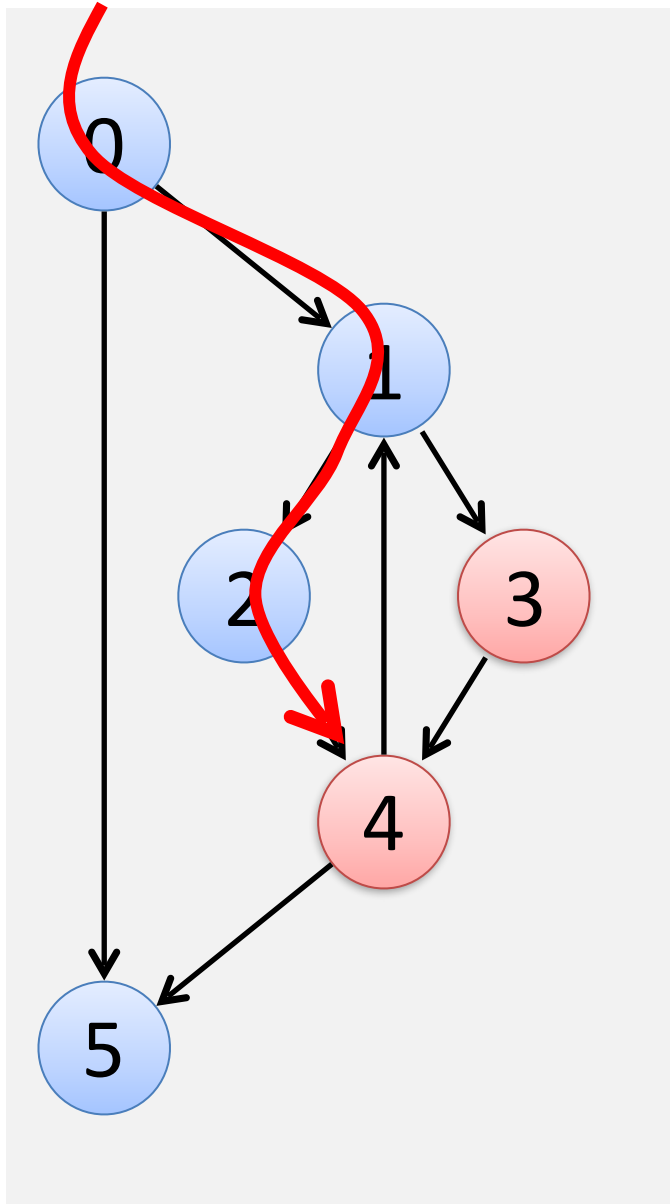
0x10010020 --> 0x1001002C

```
odd_series:
    li      $t0, 0
    li      $v0, 0
    blez    $a0, DONE
```

1
```
LOOP:
    andi    $t2, $t0, 0x1
    bnez    $t2 ODD
```

2
```
    add     $v0, $v0, $t0
    j       REINIT
```

3
```
ODD:
        add     $v0, $
```

4
```
REINIT:
    add     $t0, $t0, 1
    blt     $t0, $a0, LOOP
```

5
```
DONE:
    jr      $ra
```

Dominators

1 **dominates** 4 because you cannot execute 4 unless you have already executed 1

1 **dominates** 4 because you cannot execute 4 unless you have already executed 1

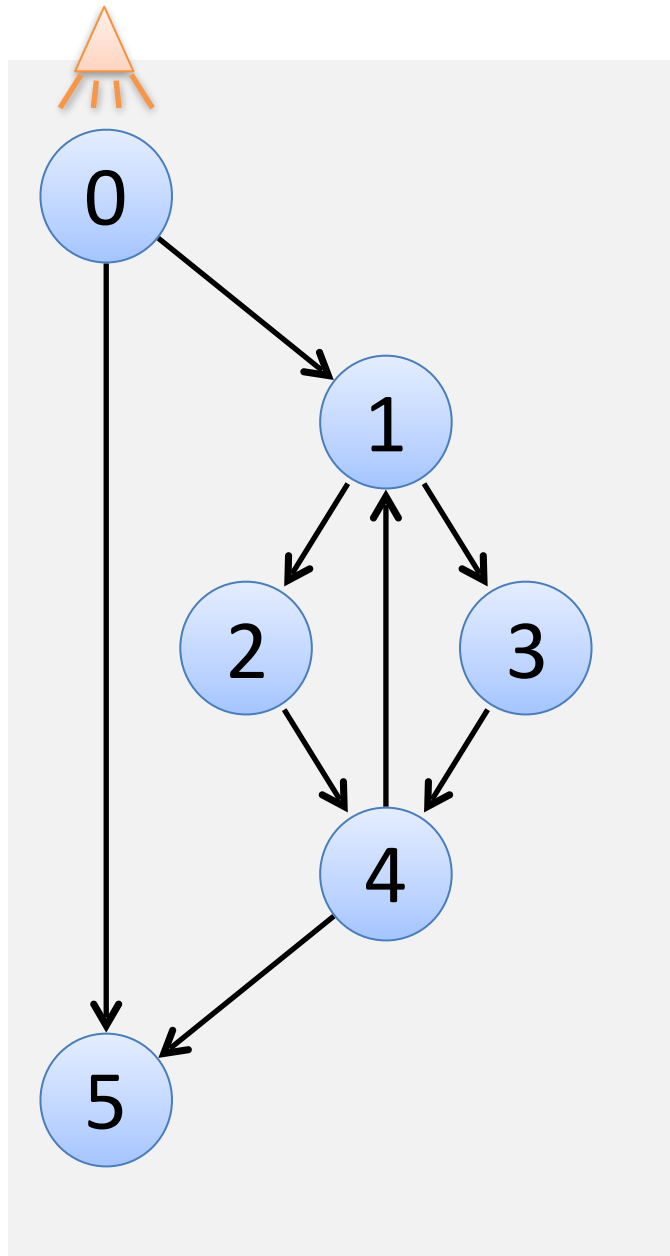3 **does not dominate** 4 because there is a path (0→1→2→4) that reaches 4 without executing 3.
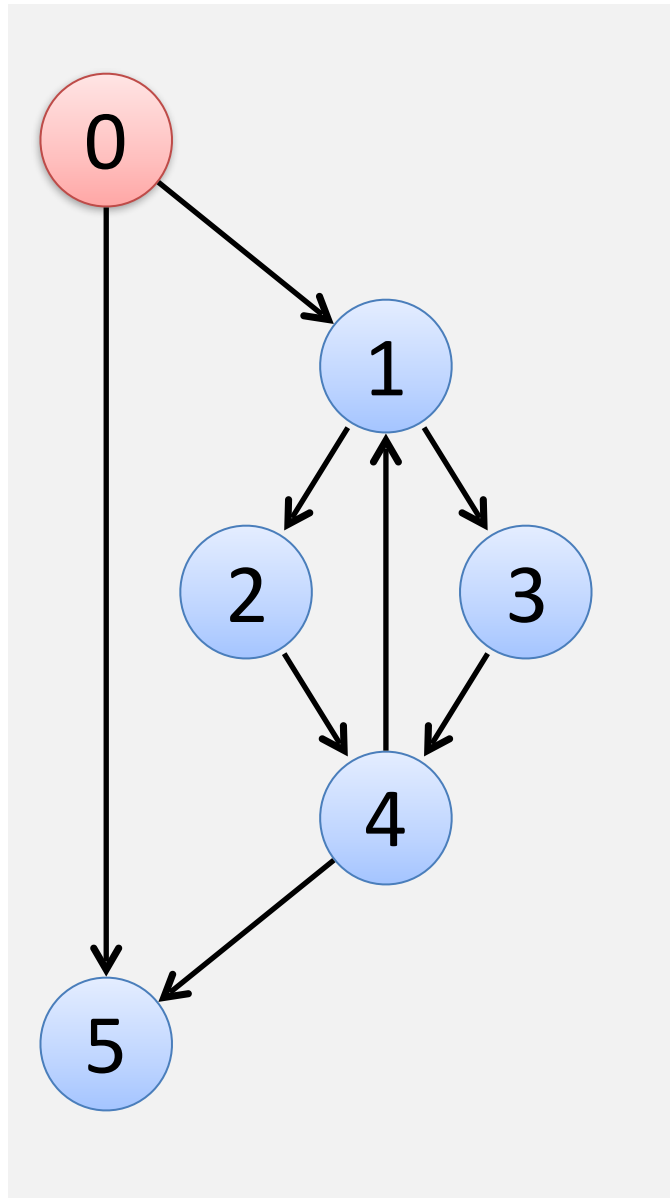
0 dominates all the basic blocks.

1 **dominates** 4 because you cannot execute 4 unless you have already executed 1
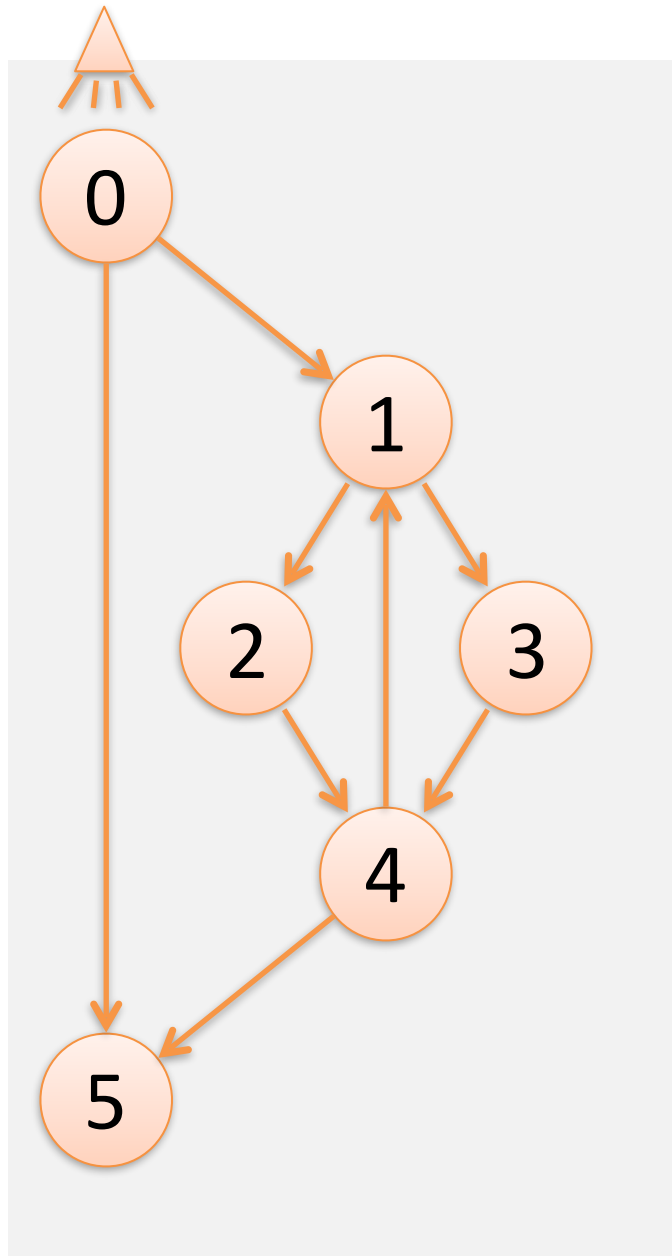
3 **does not dominate** 4 because there is a path (0→1→2→4) that reaches 4 without executing 3.
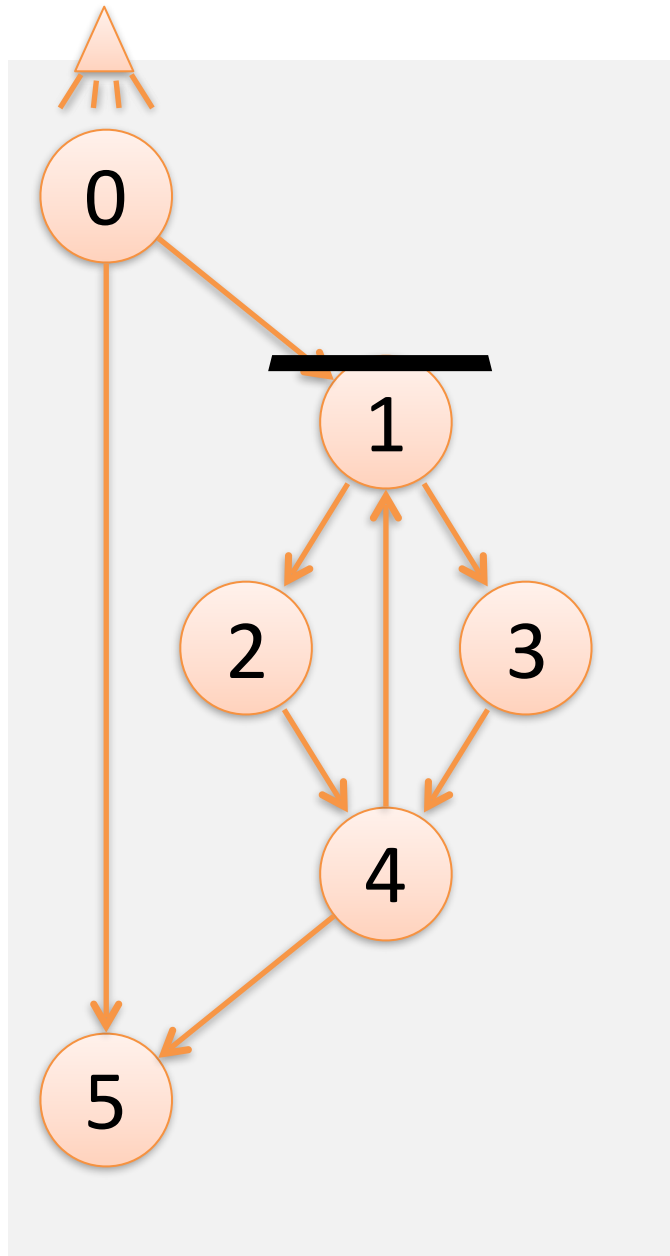
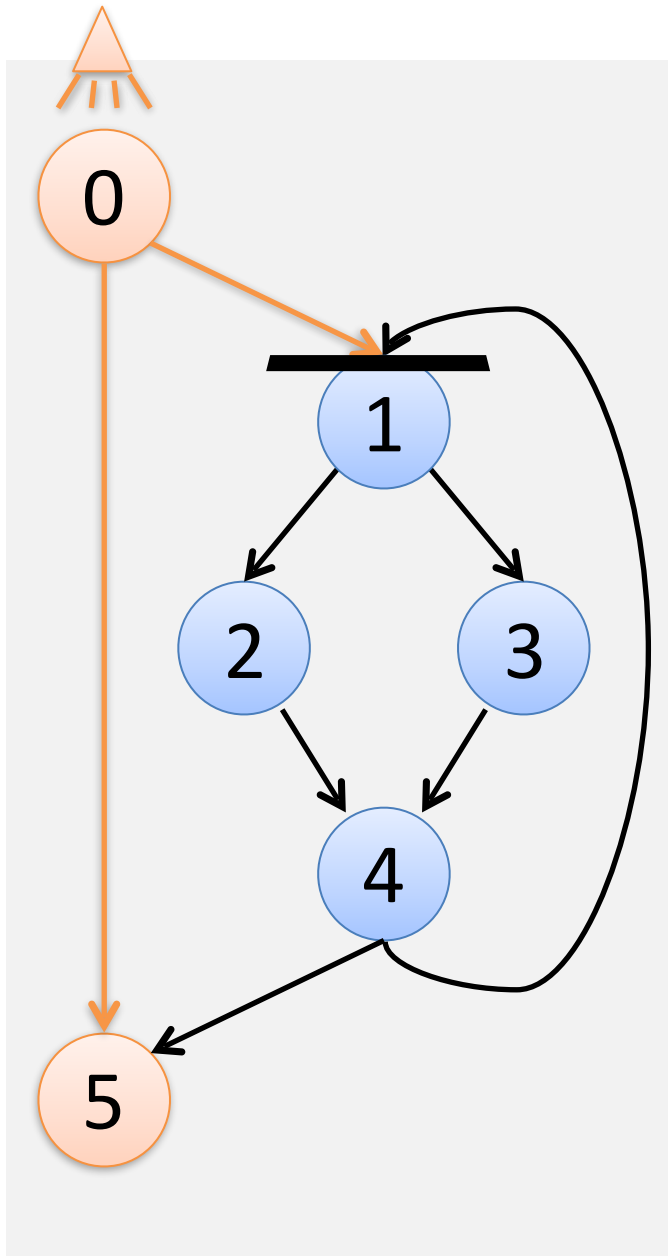A basic block dominates itself.

Intuitive Understanding of Dominance
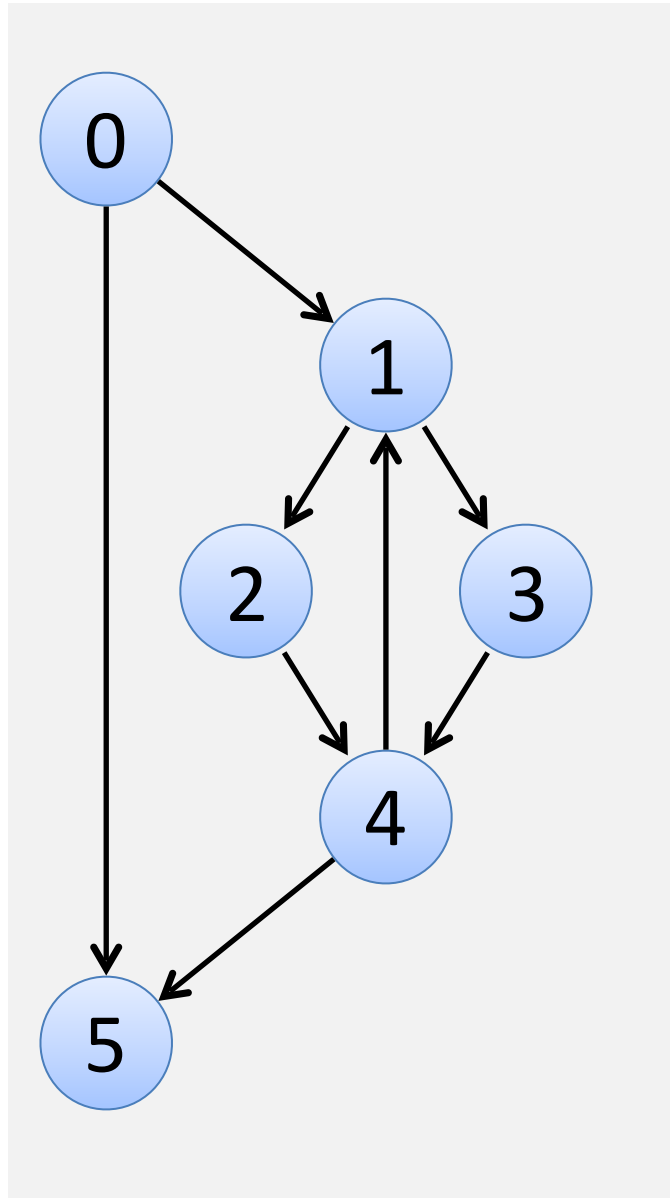
Intuitive Understanding of Dominance
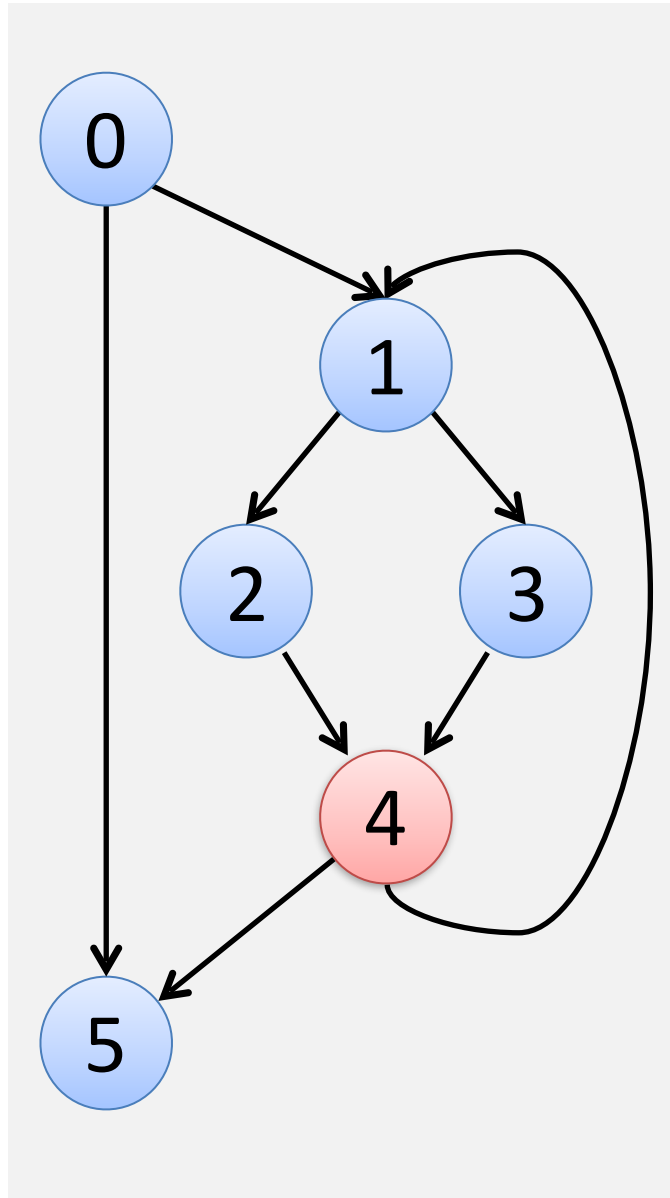
Intuitive Understanding of Dominance

Intuitive Understanding of Dominance

1 dominates nodes 1, 2, 3 and 4.

Which nodes dominate node 4?

Which nodes dominate node 4?
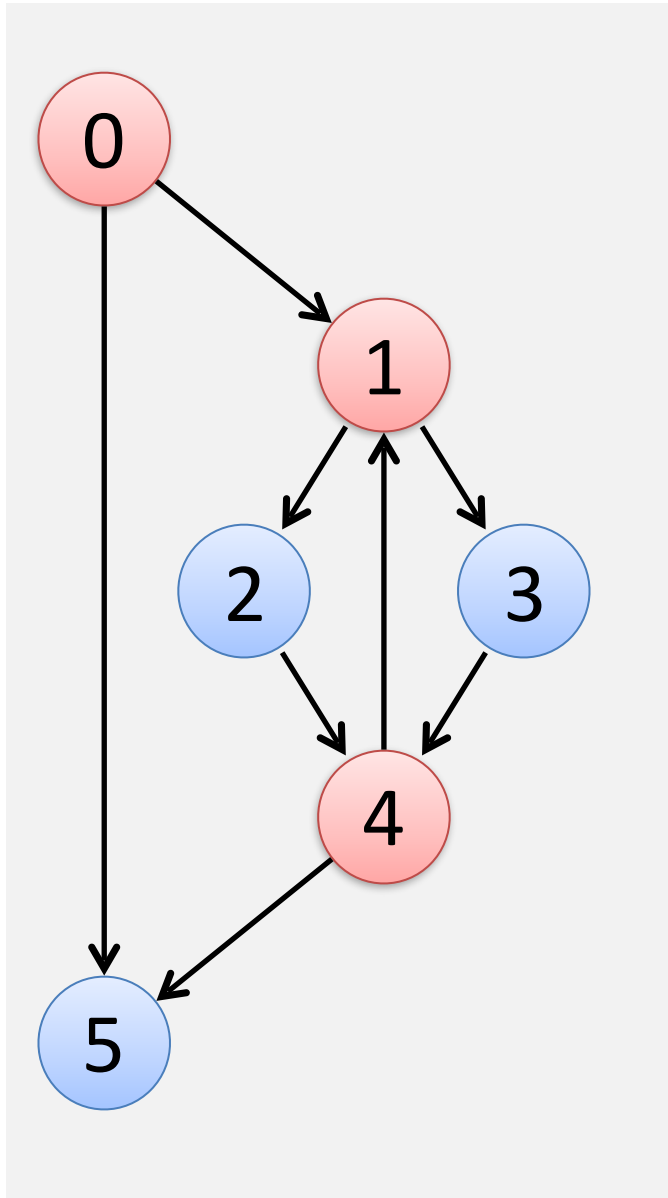
Which nodes dominate node 4?

Dom(4) = {0, 1, 4}

This is called the *Dominator Set* of 4
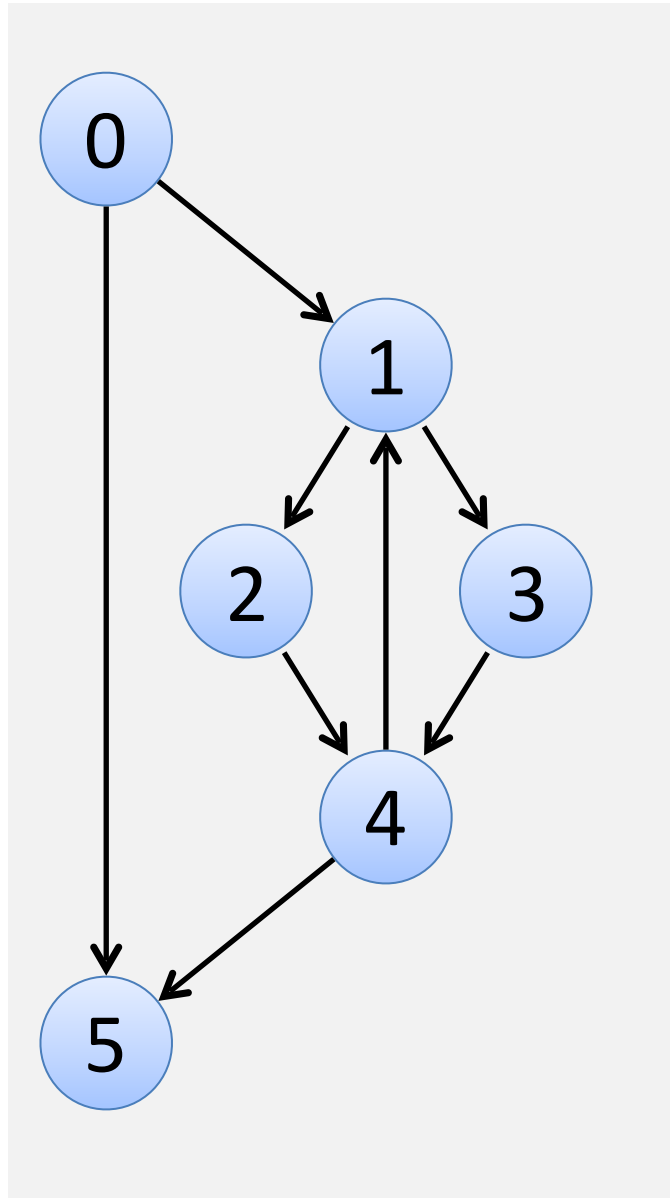
We can use a bit in a binary vector to represent each node in the CFG

Bit position in binary vector
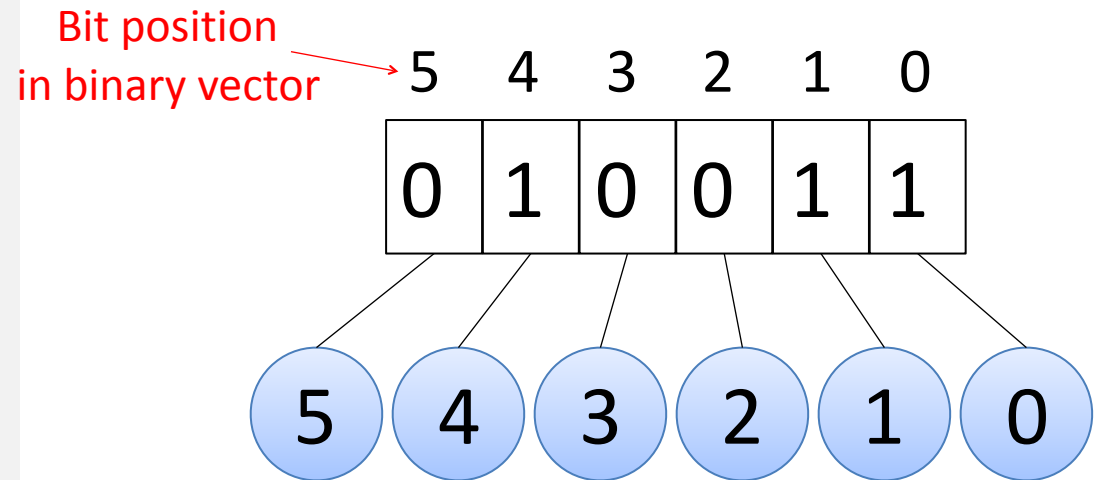
5	4	3	2	1	0
0	1	0	0	1	1

We can now represent the dominator set of 4 as a binary vector

Dom(4) = {0, 1, 4}

Dominator Bit Vectors:
0000 0000 0000 0000 0000 0000 0000 0001
0000 0000 0000 0000 0000 0000 0000 0011
0000 0000 0000 0000 0000 0000 0000 0111
0000 0000 0000 0000 0000 0000 0000 1011
0000 0000 0000 0000 0000 0000 0001 0011
0000 0000 0000 0000 0000 0000 0010 0001
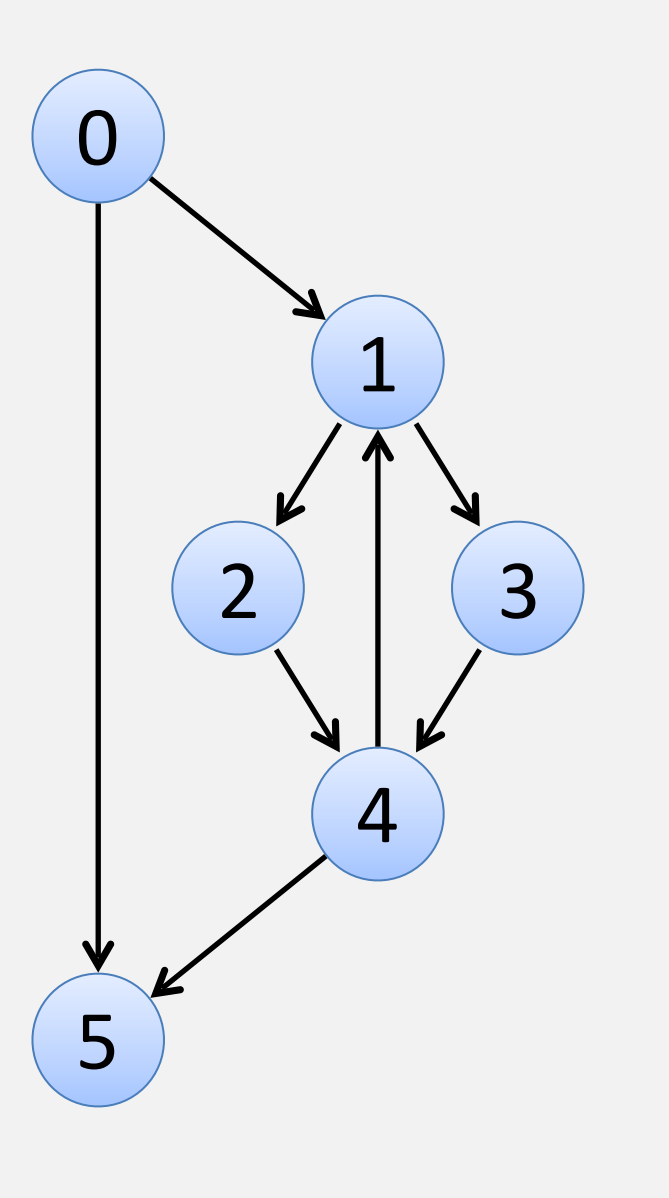
How to Compute Dominator Sets?
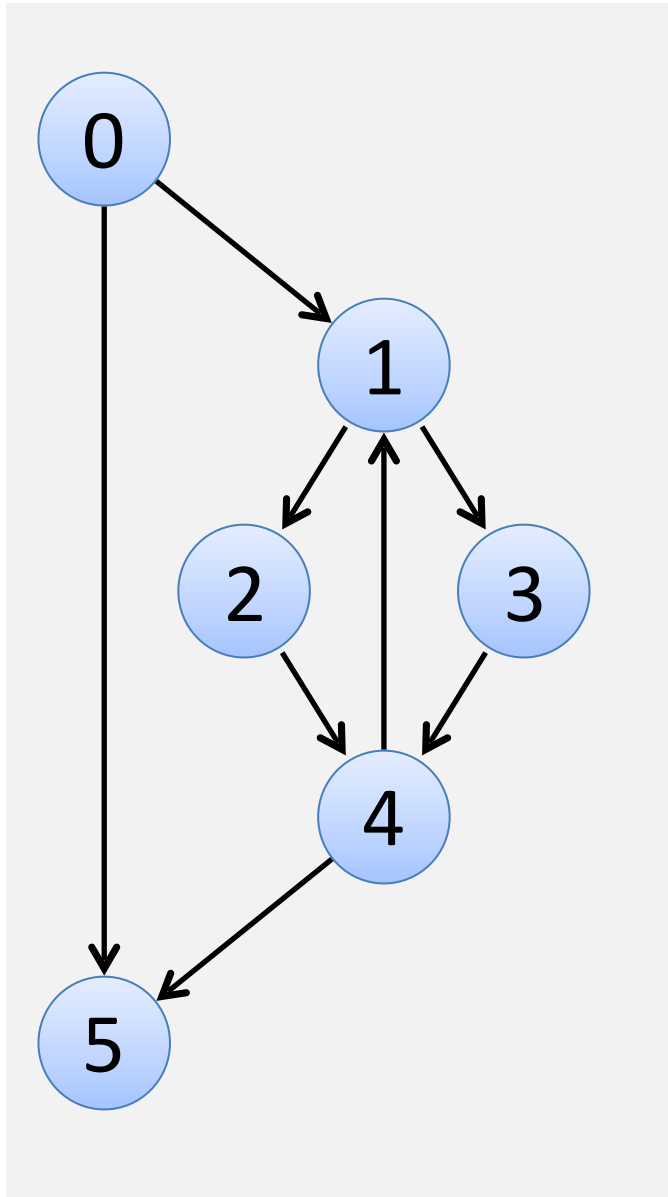
Initialization:

0 is the only dominator of 0

All other nodes are dominated by every node.

Node	Dominators					
	5	4	3	2	1	0
0	0	0	0	0	0	1
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1

For each edge (n_j, n_i) in CFG:

$$\text{Dom}(n_i) = \{n_i\} \cup (\text{Dom}(n_i) \cap \text{Dom}(n_j))$$

Node	Dominators					
	5	4	3	2	1	0
0	0	0	0	0	0	1
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1

For each edge (n_j, n_i) in CFG:

$$Dom(n_i) = \{n_i\} \cup (Dom(n_i) \cap Dom(n_j))$$

$$Dom(1) = \{1\} \cup (\{0,1,2,3,4,5\} \cap \{0\})$$

$$Dom(1) = \{1\} \cup \{0\} = \{0,1\}$$

Node	Dominators					
	5	4	3	2	1	0
0	0	0	0	0	0	1
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1

For each edge (n_j, n_i) in CFG:

$Dom(n_i) = \{n_i\} \cup (Dom(n_i) \cap Dom(n_j))$

$Dom(1) = \{1\} \cup (\{0,1,2,3,4,5\} \cap \{0\})$

$Dom(1) = \{1\} \cup \{0\} = \{0,1\}$

Node	Dominators					
	5	**4**	**3**	**2**	**1**	**0**
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1

For each edge (n_j, n_i) in CFG:

$$\text{Dom}(n_i) = \{n_i\} \cup (\text{Dom}(n_i) \cap \text{Dom}(n_j))$$

Node	Dominators					
	5	4	3	2	1	0
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1

For each edge (n_j, n_i) in CFG:

$$\text{Dom}(n_i) = \{n_i\} \cup (\text{Dom}(n_i) \cap \text{Dom}(n_j))$$

Node	Dominators					
	5	4	3	2	1	0
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	0	0	0	0	1

For each edge (n_j, n_i) in CFG:

$$\text{Dom}(n_i) = \{n_i\} \cup (\text{Dom}(n_i) \cap \text{Dom}(n_j))$$

Node	Dominators					
	5	4	3	2	1	0
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	0	0	0	0	1

For each edge (n_j, n_i) in CFG:

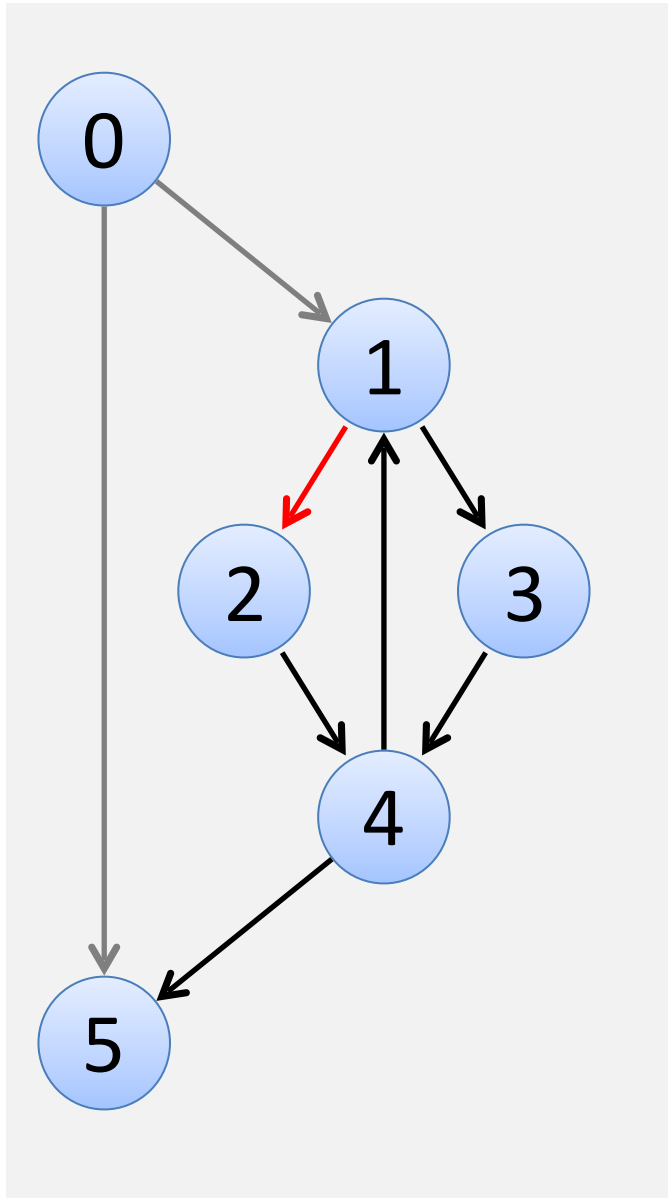$$\text{Dom}(n_i) = \{n_i\} \cup (\text{Dom}(n_i) \cap \text{Dom}(n_j))$$

Node	Dominators					
	5	**4**	**3**	**2**	**1**	**0**
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	0	0	0	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	0	0	0	0	1

For each edge (n_j, n_i) in CFG:

$$\mathrm{Dom}(n_i) = \{n_i\} \cup (\mathrm{Dom}(n_i) \cap \mathrm{Dom}(n_j))$$

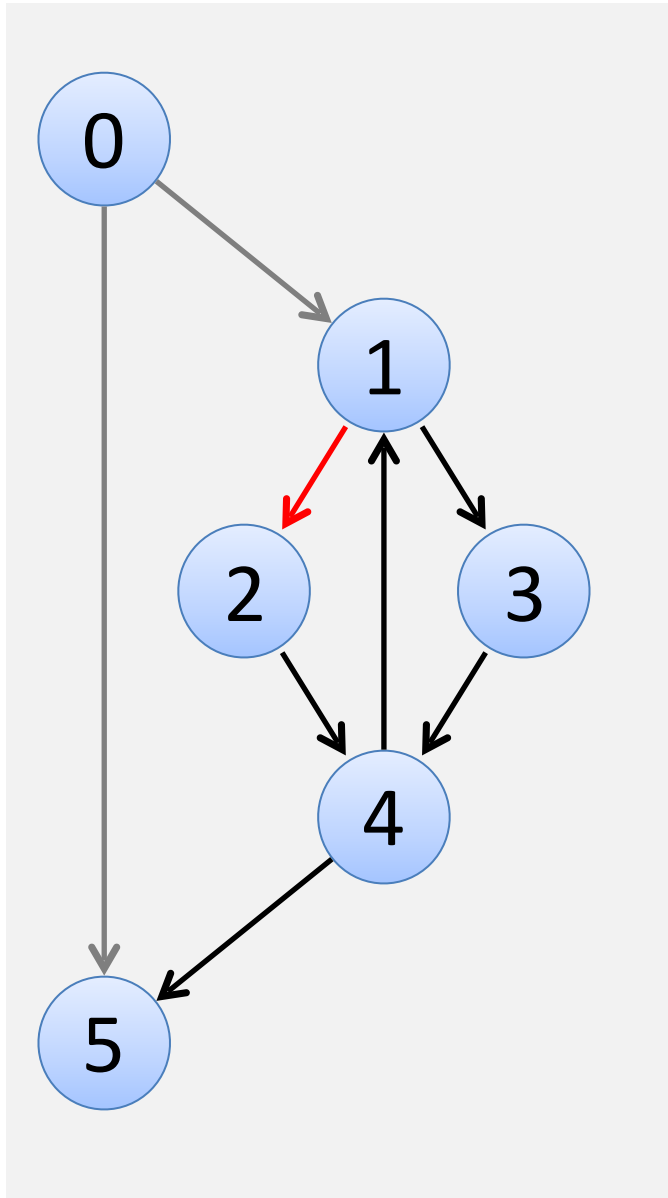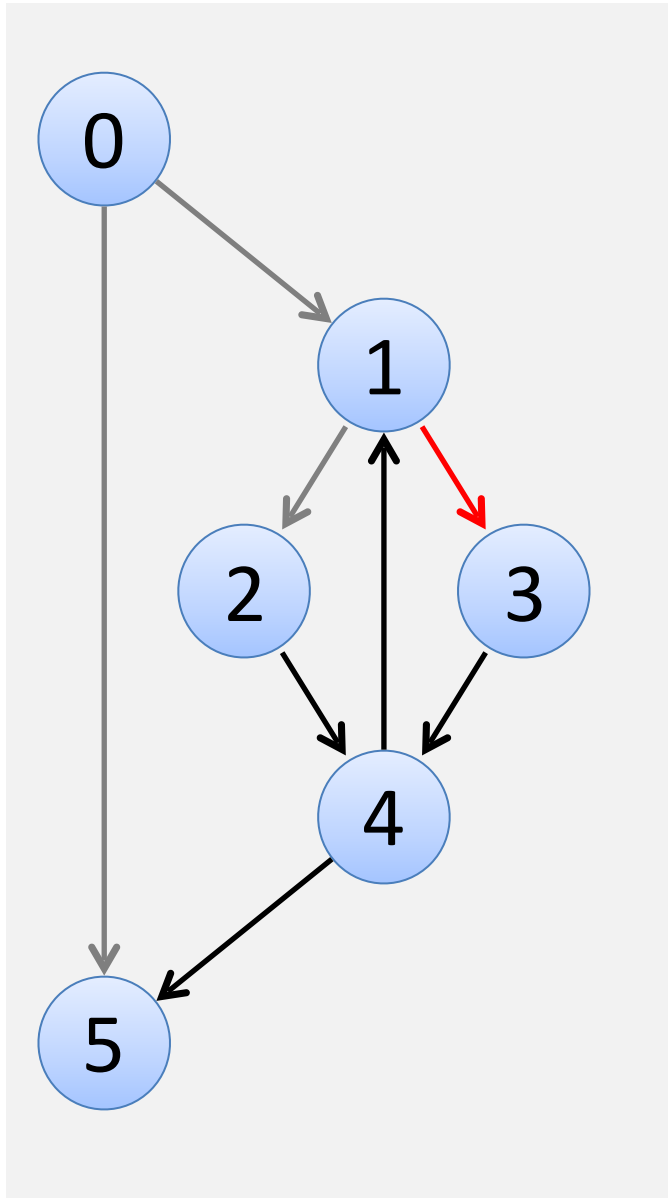| Node	Dominators					
	5	4	3	2	1	0
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	0	0	0	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	0	0	0	0	1

For each edge (n_j, n_i) in CFG:

$$\text{Dom}(n_i) = \{n_i\} \cup (\text{Dom}(n_i) \cap \text{Dom}(n_j))$$

Node	Dominators					
	5	4	3	2	1	0
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	0	0	0	1	1	1
3	0	0	1	0	1	1
4	1	1	1	1	1	1
5	1	0	0	0	0	1

For each edge (n_j, n_i) in CFG:

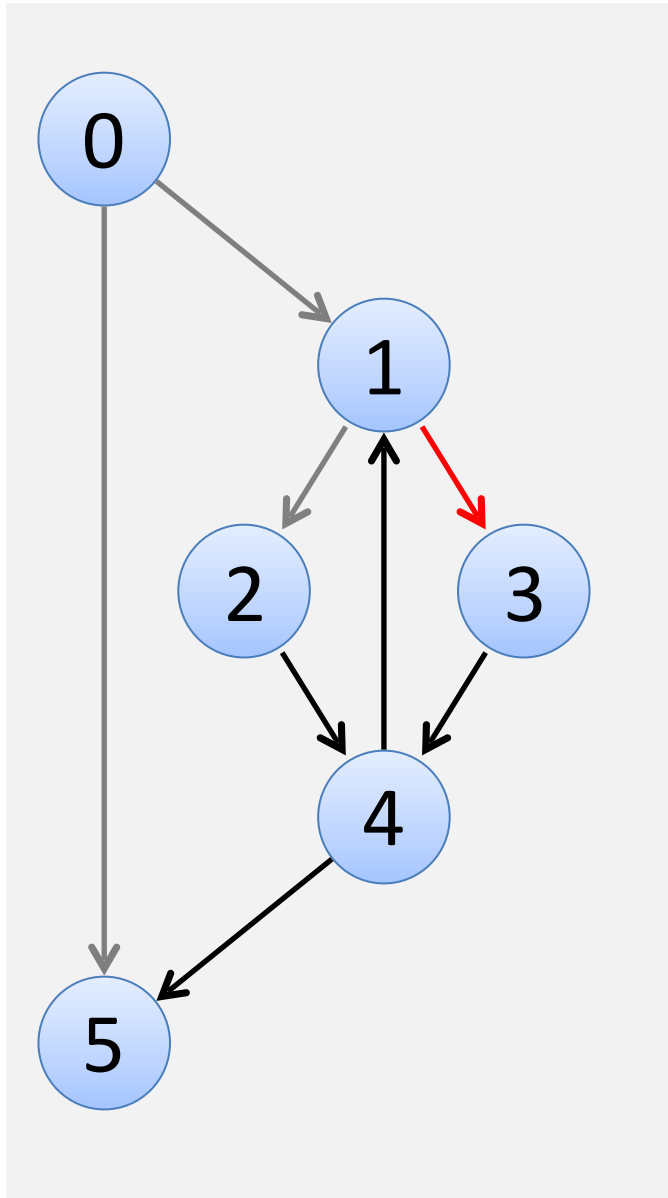$$\text{Dom}(n_i) = \{n_i\} \cup (\text{Dom}(n_i) \cap \text{Dom}(n_j))$$

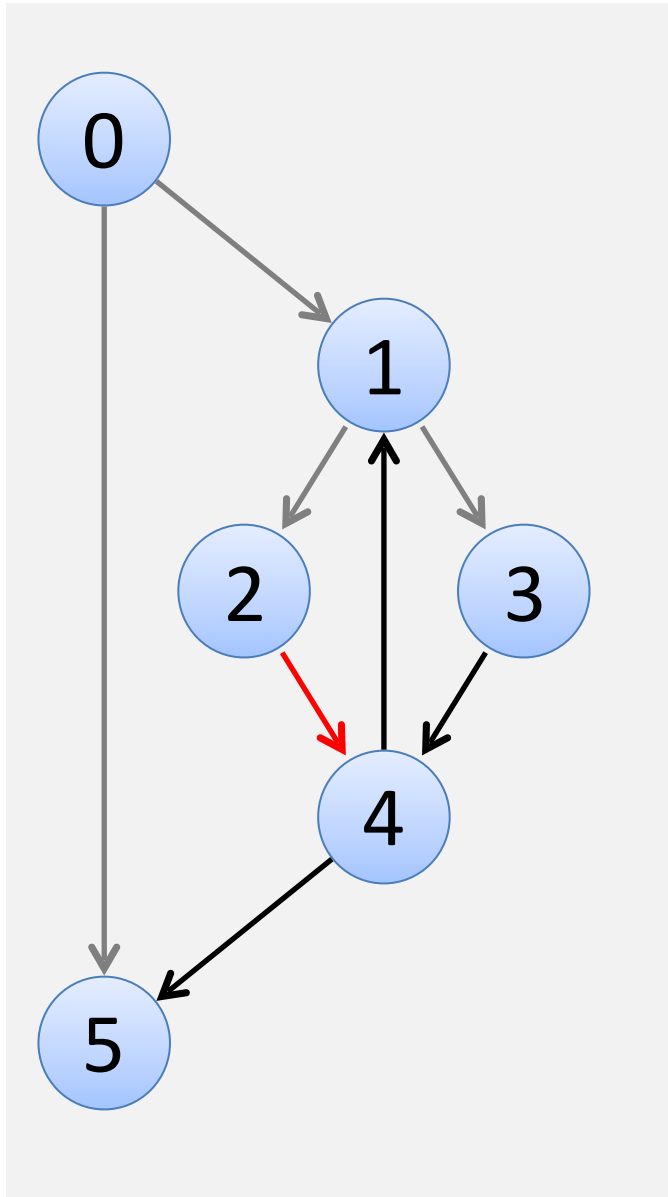| Node	Dominators					
	5	4	3	2	1	0
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	0	0	0	1	1	1
3	0	0	1	0	1	1
4	1	1	1	1	1	1
5	1	0	0	0	0	1

For each edge (n_j, n_i) in CFG:

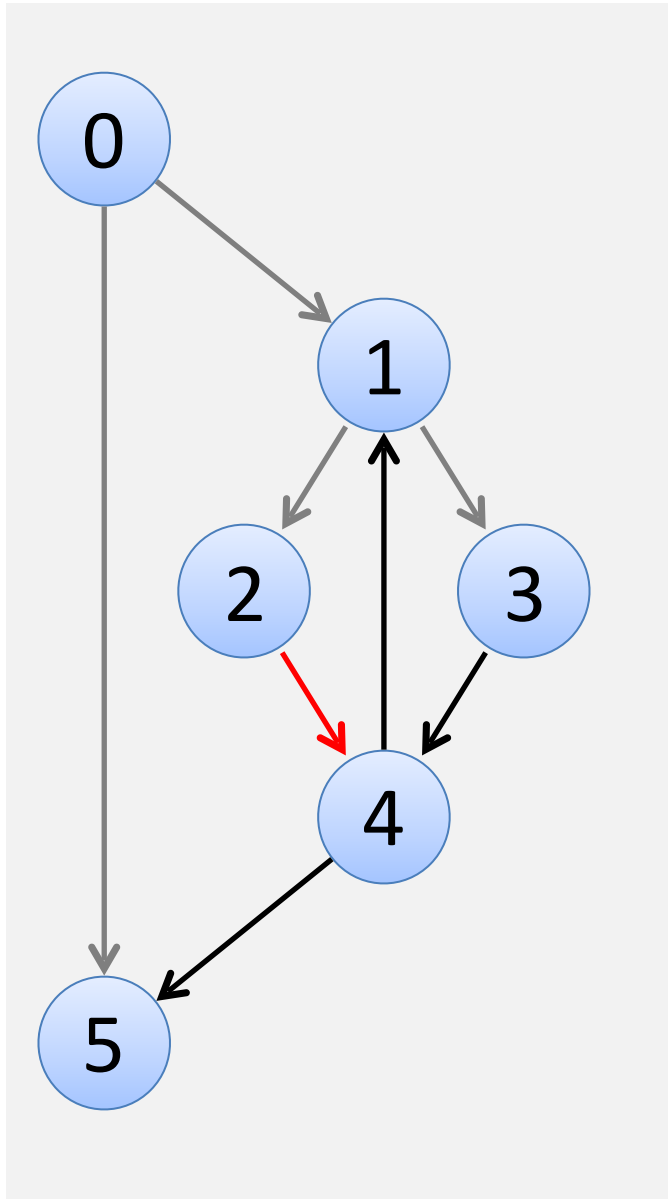$$\text{Dom}(n_i) = \{n_i\} \cup (\text{Dom}(n_i) \cap \text{Dom}(n_j))$$

Node	Dominators					
	5	4	3	2	1	0
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	0	0	0	1	1	1
3	0	0	1	0	1	1
4	0	1	0	1	1	1
5	1	0	0	0	0	1

For each edge (n_j, n_i) in CFG:

$$Dom(n_i) = \{n_i\} \cup (Dom(n_i) \cap Dom(n_j))$$

Node	Dominators					
	5	**4**	**3**	**2**	**1**	**0**
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	0	0	0	1	1	1
3	0	0	1	0	1	1
4	0	1	0	1	1	1
5	1	0	0	0	0	1

For each edge (n_j, n_i) in CFG:

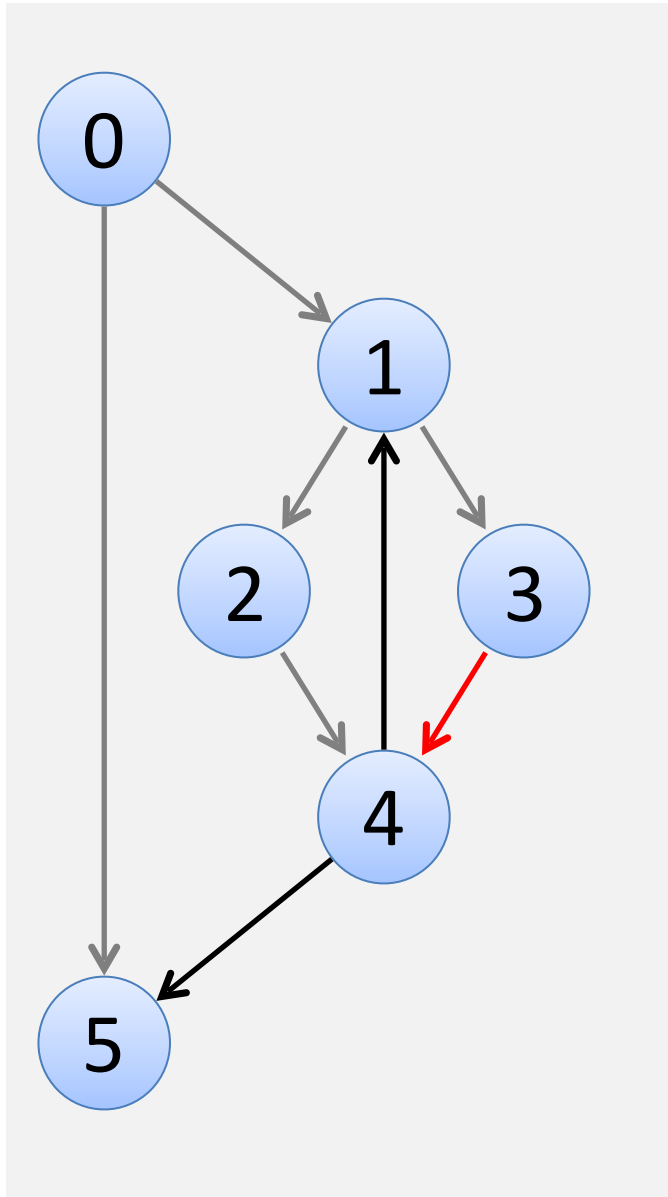$$\text{Dom}(n_i) = \{n_i\} \cup (\text{Dom}(n_i) \cap \text{Dom}(n_j))$$

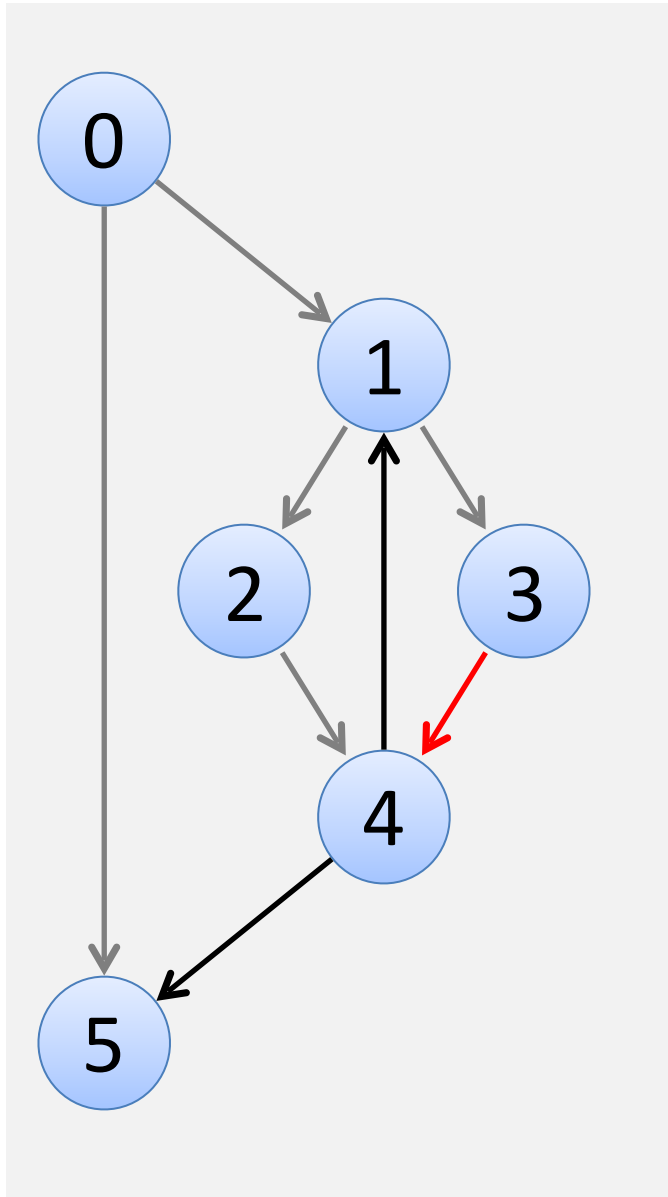| Node	Dominators					
	5	4	3	2	1	0
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	0	0	0	1	1	1
3	0	0	1	0	1	1
4	0	1	0	0	1	1
5	1	0	0	0	0	1

For each edge (n_j, n_i) in CFG:

$$\text{Dom}(n_i) = \{n_i\} \cup (\text{Dom}(n_i) \cap \text{Dom}(n_j))$$

Node	Dominators					
	5	**4**	**3**	**2**	**1**	**0**
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	0	0	0	1	1	1
3	0	0	1	0	1	1
4	0	1	0	0	1	1
5	1	0	0	0	0	1

For each edge (n_j, n_i) in CFG:

$$\text{Dom}(n_i) = \{n_i\} \cup (\text{Dom}(n_i) \cap \text{Dom}(n_j))$$

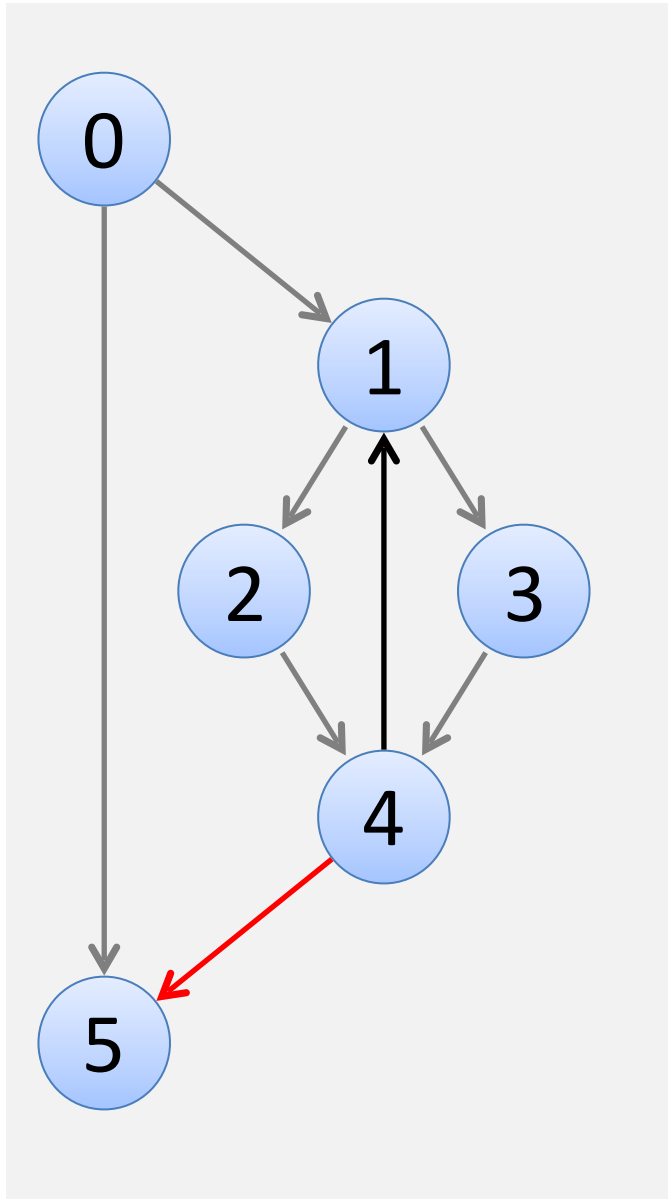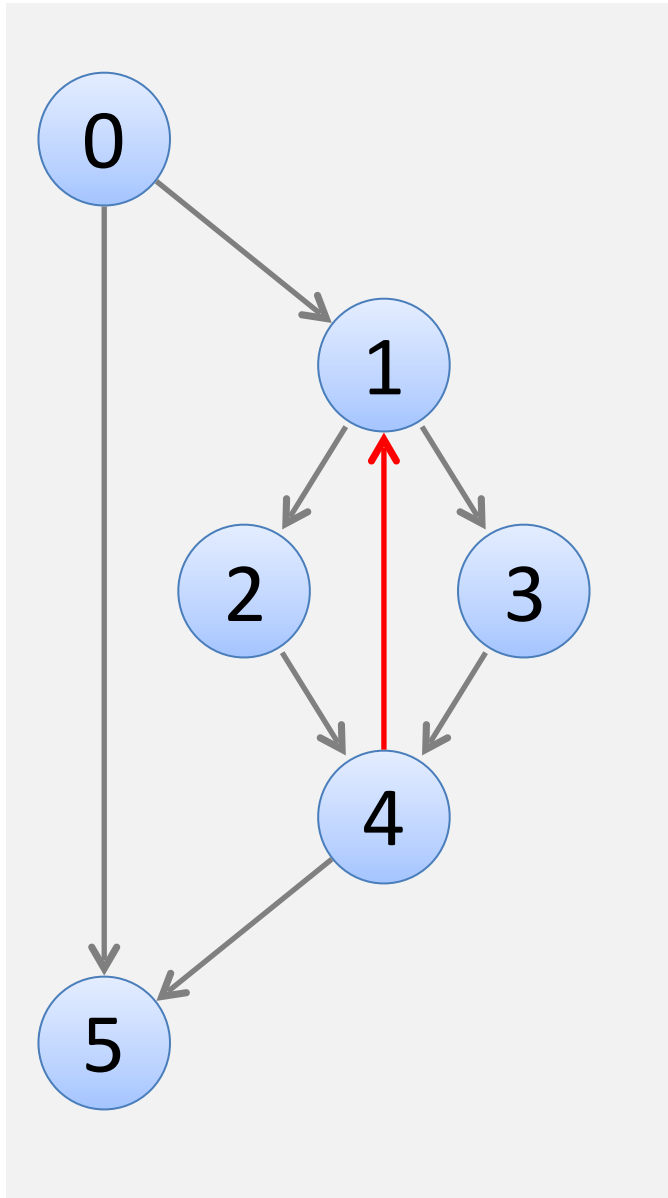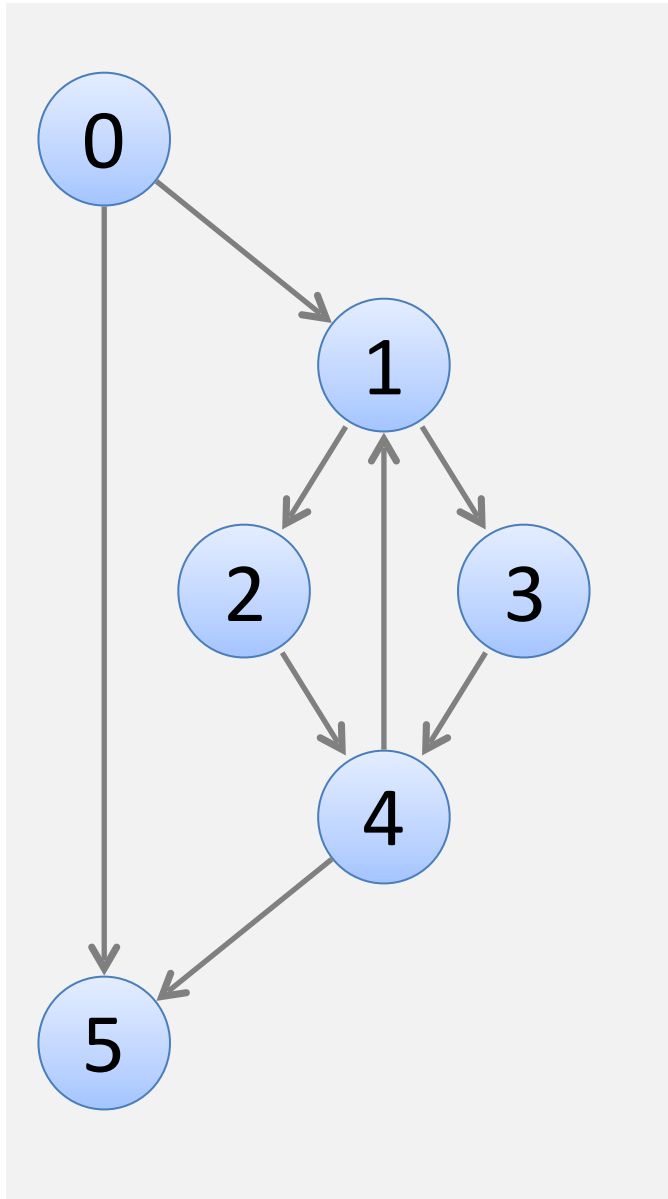| Node	Dominators					
	5	4	3	2	1	0
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	0	0	0	1	1	1
3	0	0	1	0	1	1
4	0	1	0	0	1	1
5	1	0	0	0	0	1

For each edge (n_j, n_i) in CFG:

$$\text{Dom}(n_i) = \{n_i\} \cup (\text{Dom}(n_i) \cap \text{Dom}(n_j))$$

Node	Dominators					
	5	4	3	2	1	0
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	0	0	0	1	1	1
3	0	0	1	0	1	1
4	0	1	0	0	1	1
5	1	0	0	0	0	1

Bit Vectors

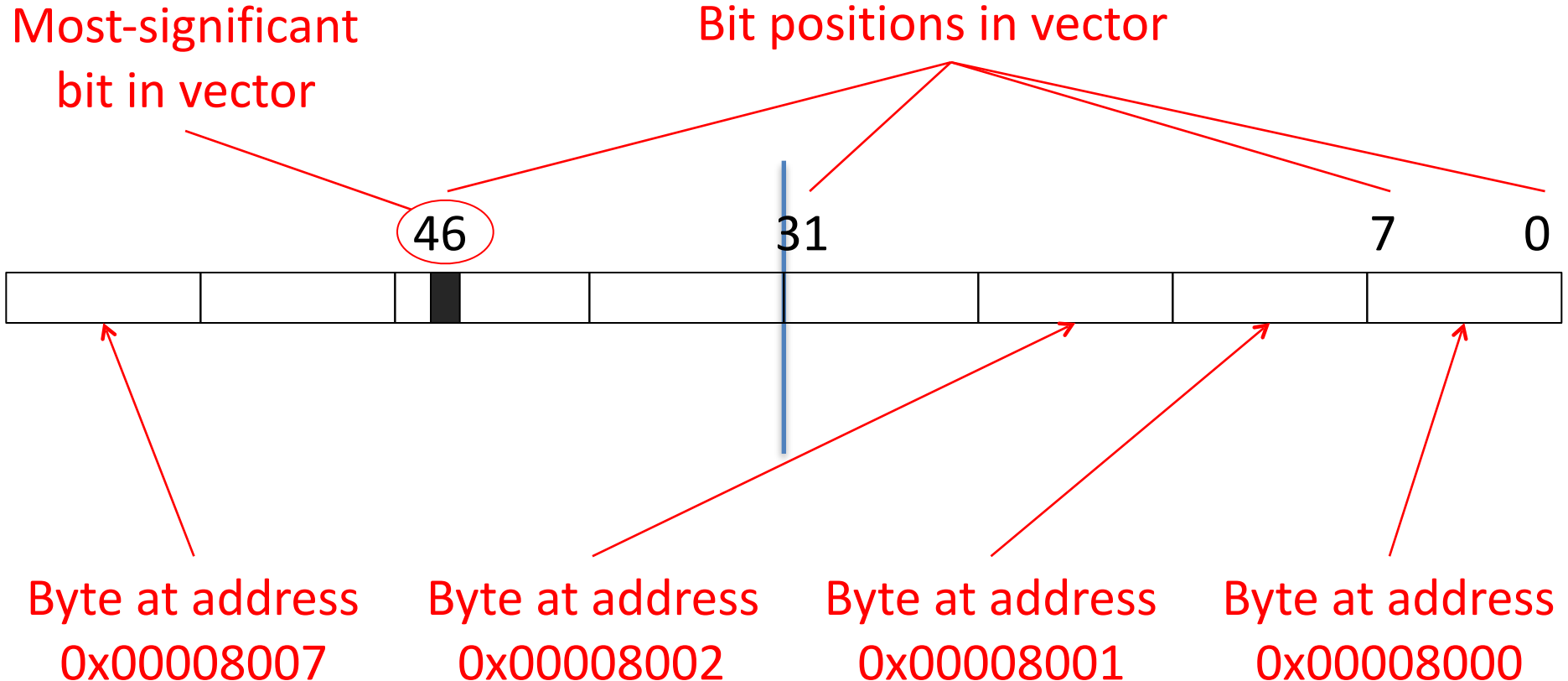Use as many words as necessary to store the bit vector in memory.

Examples:

A 17-bit vector occupies one word.

A 42-bit vector occupies two words.

A 65-bit vector occupies three words.

Bit Ordering

Consider a 47-bit vector stored at address 0x00008000.

Most-significant bit in vector

Bit positions in vector

46

31

7

0

Byte at address 0x00008007

Byte at address 0x00008002

Byte at address 0x00008001

Byte at address 0x00008000

The Assignment (input)

The MIPS code is guaranteed to contain a single procedure.
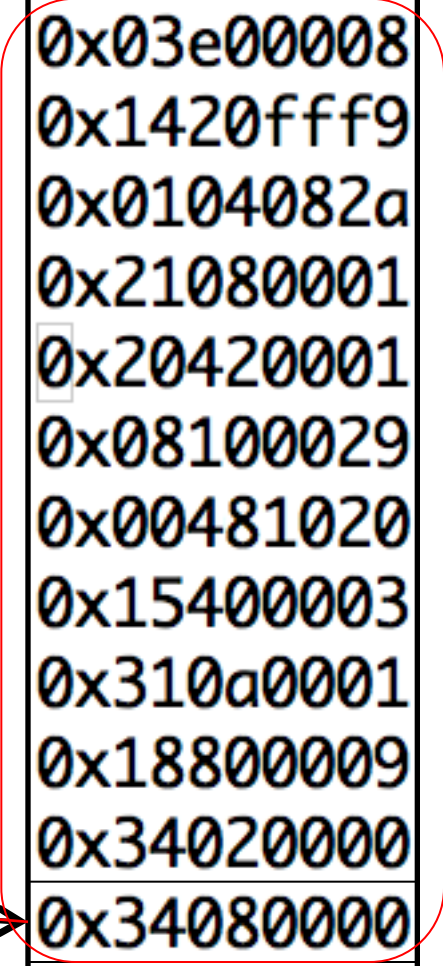
This is the sentinel indicating the end of the procedure.

$a0

This is the binary code for the odd_series example in this presentation.

$a0 contains a memory address

At that address is the binary representation of the first instruction

Memory

```
0xffffffff
0x03e00008
0x1420fff9
0x0104082a
0x21080001
0x20420001
0x08100029
0x00481020
0x15400003
0x310a0001
0x18800009
0x34020000
0x34080000
```

The Assignment (output)

$v0: number of basic blocks in the procedure.

$v1: number of edges in the CFG of the procedure.

Three additional memory addresses returned into the stack.

Returning Addresses in Stack

Memory

$fp →

Caller
Stack
Frame

$sp →

Address of list
of dominator
sets

Address of list
of CFG edges

Address of list
of basic blocks

Before getControlFlow

Memory

$fp →

Caller
Stack
Frame

$sp →

After getControlFlow

List of Basic Blocks
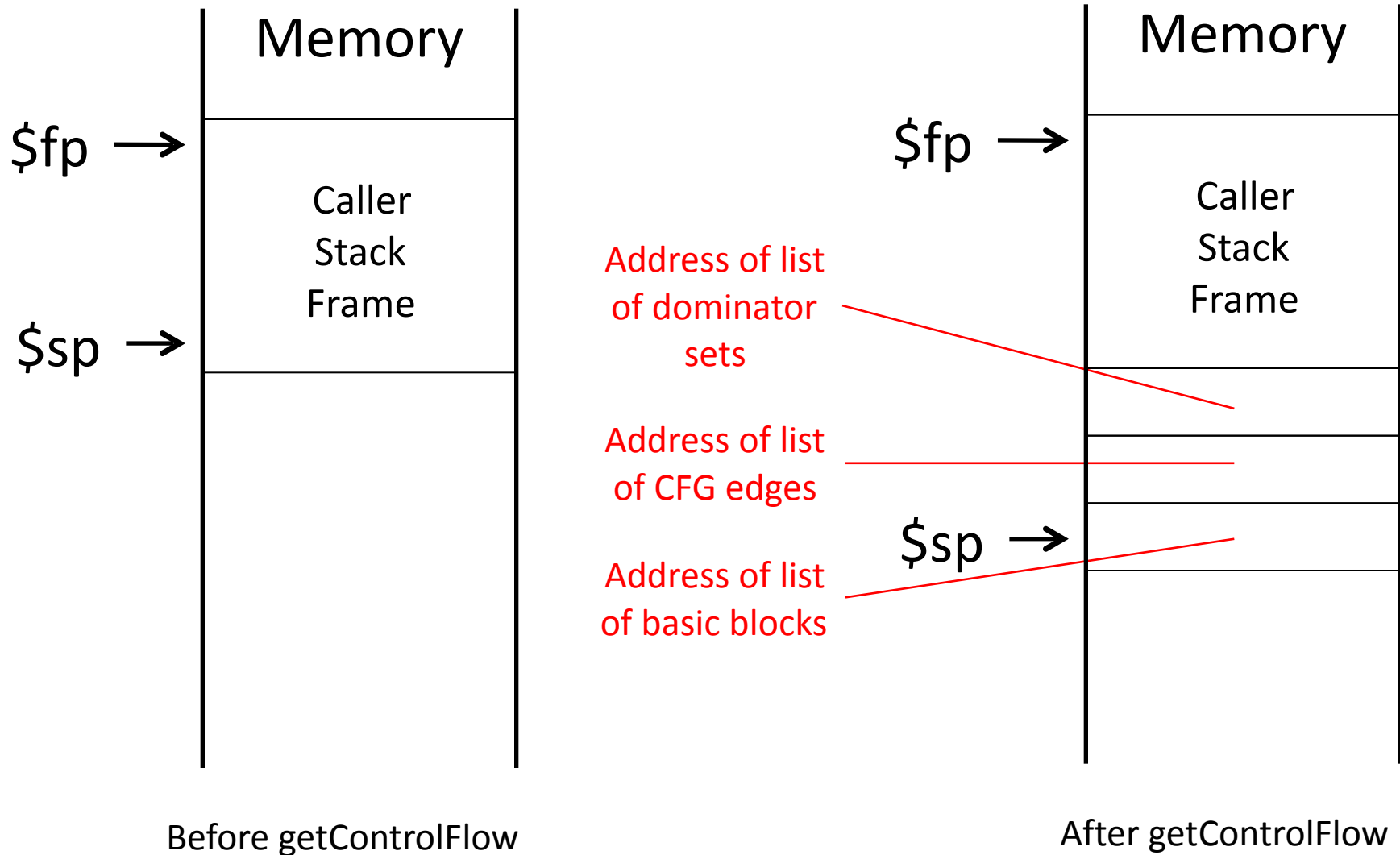
Basic blocks in ascending order of the address of their leaders.

Basic Block List

0x0000 0001
0x1001 002c
0x0000 0003
0x1001 0020
0x0000 0001
0x1001 001c
0x0000 0002
0x1001 0014
0x0000 0002
0x1001 000c
0x0000 0003
0x1001 0000

higher address ↑

lower address ↓

```
[                    $0, 0              ; 52: li $t0, 0 # i
[                    $0, 0              ; 53: li $v0, 0 # j
[              4 36 [DONE-0x10010008]; 54: blez $a0, DONE #
```

6 block(s) found.
Block Leader: 0x10010000, Size: 3
Block Leader: 0x1001000C, Size: 2
Block Leader: 0x10010014, Size: 2
Block Leader: 0x1001001C, Size: 1
Block Leader: 0x10010020, Size: 3
Block Leader: 0x1001002C, Size: 1

```
[10010028] 1420fff9  bne $1, $0
[1001002c] 03e00008  jr $31              ; 66: jr $ra
```

After getControlFlow address in $sp+0 is the address of this word.

# of CFG Edges

CFG Edge List

higher address ↑

0x1001 002c
0x1001 0020
0x1001 000c
0x1001 0020
0x1001 0020
0x1001 001c
0x1001 0020
0x1001 0014
0x1001 001c
0x1001 000c
0x1001 0014
0x1001 000c
0x1001 002c
0x1001 0000
0x1001 000c
0x1001 0000

lower address ↓

Edges with the same source
in ascending order of
the address of the target leader.

$0, 0 # i
$0, 0 # j
36 [DONE-0x10010008]; 54: blez $a0, DONE #

l0, $8, 1   ; 56: andi $t2, $t0, 0x1 #

0, $0, 12

, $2, $8

010020 [RE

2, $2, 1

3, $8, 1
$8, $4

[10010028] 1420fff9 bne $1, $0, -28

[1001002c] 03e00008 jr $31

After getControlFlow address in $sp+4
is the address of this word.

Edges:
0x10010000 --> 0x1001000C
0x10010000 --> 0x1001002C
0x1001000C --> 0x10010014
0x1001000C --> 0x1001001C
0x10010014 --> 0x10010020
0x1001001C --> 0x10010020
0x10010020 --> 0x1001000C
0x10010020 --> 0x1001002C

List of Dominator Sets

Basic Block List

higher address ↑
- 0x0000 0021
- 0x0000 0013
- 0x0000 000b
- 0x0000 0007
- 0x0000 0003

lower address ↓ 0x0000 0001

Dominator Bit Vectors:
0000 0000 0000 0000 0000 0000 0000 0001
0000 0000 0000 0000 0000 0000 0000 0011
0000 0000 0000 0000 0000 0000 0000 0111
0000 0000 0000 0000 0000 0000 0000 1011
0000 0000 0000 0000 0000 0000 0001 0011
0000 0000 0000 0000 0000 0000 0010 0001

2   3

After getControlFlow address in $sp+8 is the address of this word.

5

If the CFG has more than 32 basic blocks, then each binary vector will occupy more than one word.

Dominator sets in ascending order of the address of the corresponding basic-block's leader.

Testing

- Test Cases
  - A few test cases are provided under Resources
- Student-Generated Test Cases
  - TestCaseGeneration
- Printing the Output of your solution
  - MIPS code provided for printing

University of Alberta
Code of Student Behavior

http://www.governance.ualberta.ca/en/CodesofConductandResidenceCommunityStandards/CodeofStudentBehaviour.aspx

30.3.2(2) Cheating

30.3.2(2) a No Student shall in the course of an examination or other similar activity, obtain or attempt to obtain information from another Student or other unauthorized source, give or attempt to give information to another Student, or use, attempt to use or possess for the purposes of use any unauthorized material.