编译原理

18. 循环优化

rainoftime.github.io
浙江大学
计算机科学与技术学院

Content

Outline

这些都
不考!

1. Loops and Dominators

- Loop optimizations
- Dominators
- Back to Loops
- Loop Preheader

Loop Optimizations

- Loops are pervasive in computer programs
- A great proportion of the execution time of a typical program is spent in one loop or another.
- So we want techniques to improve loops!

Examples of Loop Optimizations

- Loop invariant hoisting
- Induction variable reduction
- Loop unrolling
- Loop fusion
- Loop fission
- Loop interchange
- Loop peeling
- Loop tiling
- Loop parallelization
- …

Loop Optimizations

- **Low level optimization**
  - Moving code around in a single loop
  - Examples: loop invariant code motion, strength reduction, loop unrolling

- **High level optimization**
  - Restructuring loops, often affects multiple loops
  - Examples: loop fusion, loop interchange, loop tiling

Example: Loop Invariant Hoisting

- t = a + b is a loop invariant

```
L0:  t := 0


L1:  i := i + 1
     t := a + b
     *i := t
     if i<N goto L1 else L2

L2:  x := t
```

Example: Loop Invariant Hoisting

- Move t = a + b for optimization

```
L0:   t := 0
      t := a + b


L1:   i := i + 1

      *i := t
      if i<N goto L1 else L2

 L2:   x := t
```

Loops

- For optimizations, we define a special kind of loops

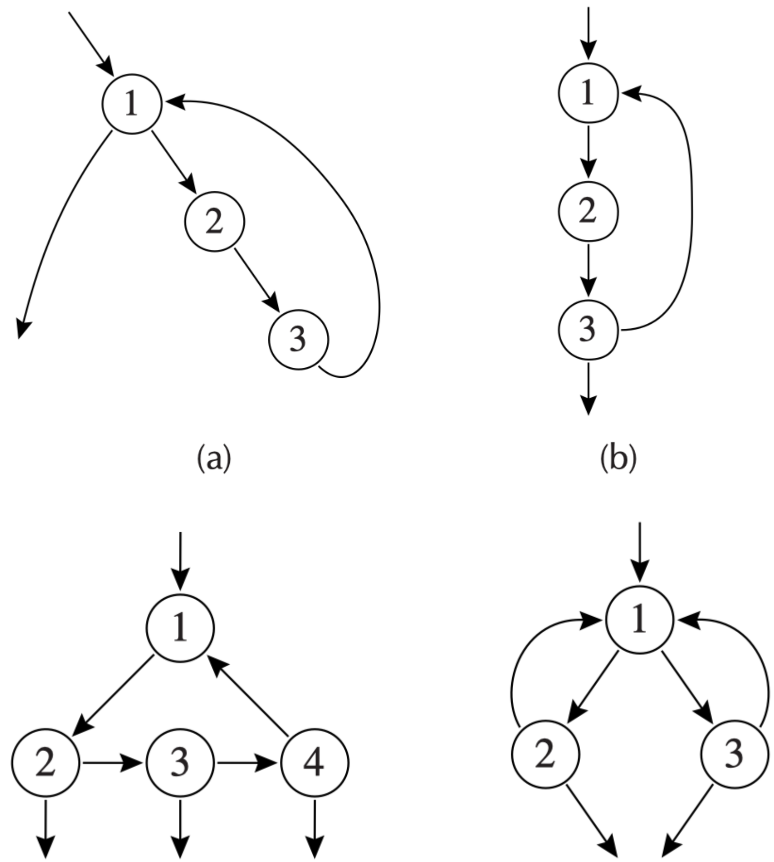
- A **loop** in a control-flow graph is a set of nodes S including a header node h with the following properties:
  1. From any node in S there is a **path** of directed edges leading to h.
  2. There is a **path** of directed edges from h to any node in S.
  3. There is no **edge** from any node outside S to any node in S other than h.

Example: Loops

- A **loop entry** node is one with predecessor outside the loop.
- A **loop exit** node is one with a successor outside the loop.

  - h -> S
  - S -> h
  - no other node -> S

Only **one** entry, but may have **multiple** exists



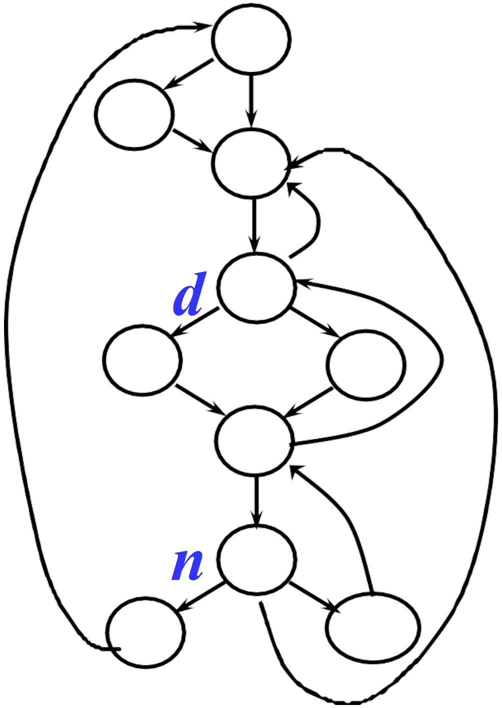
(a) (b) (c) (d)

1. Loops and Dominators

- Loop optimizations
- Dominators
- Back to Loops
- Loop Preheader

Dominator (支配结点)

- **Dominator**: ***d*是*n*的支配结点** **(*d* dom *n*)**: 从流图的入口结点s_0到结点n的每条路径都经过节点d
  - Every node dominates itself
  - *n* can have more than one dominators



Finding Dominators

- Let D[n] be the set of nodes that dominate n, Then

$$D[s_0] = \{s_0\} \qquad D[n] = \{n\} \cup \left(\bigcap_{p \in \mathrm{pred}[n]} D[p] \right) \qquad \text{for } n \neq s_0$$

- It's pretty easy to solve this equation:
  - Start off assuming D[n] is all nodes, except for the start node (which is dominated only by itself)
  - Iteratively update D[n] based on predecessors until you reach a fixed point

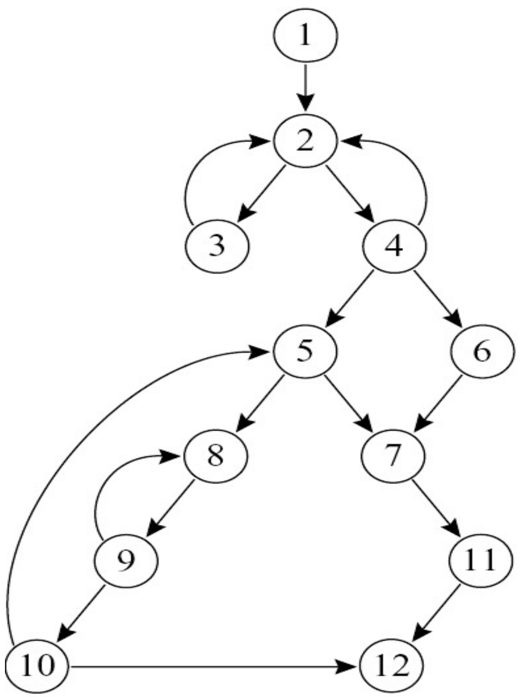
Immediate Dominators (直接支配结点)

- **直接支配结点** (immediate dominator)
  - 从入口结点到达n的任何路径 (不含n) 中，它是路径中最后一个支配n的结点
- Every node n (except s_0) has exactly one *immediate dominator, idom(n), such that:*
  - *idom(n)* is not the same node n
  - *idom(n)* dominates n, and
  - *idom(n)* does not dominate any other dominator of n

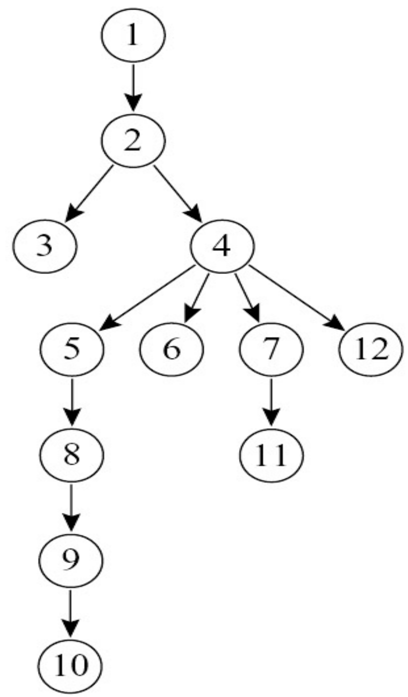
Theorem: Suppose d dominates n, and e dominates n.
Then it must be that either d dominates e or e dominates d.

Dominator Tree (支配结点树)

- **Dominator tree**
  - Containing every node of the flow graph, and
  - For every node *n*, there is an edge from *idom(n)* to n.

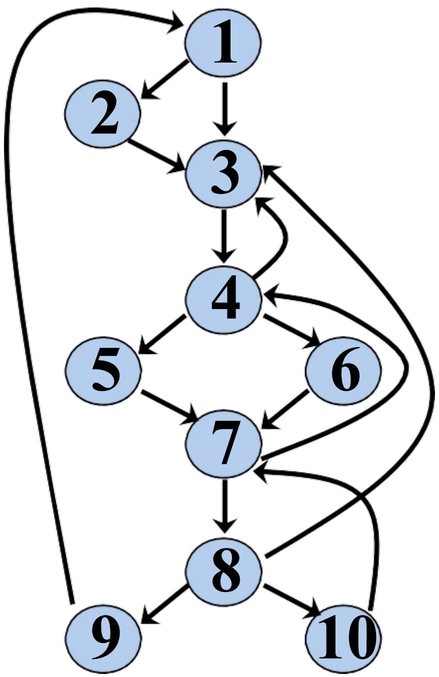


(a)



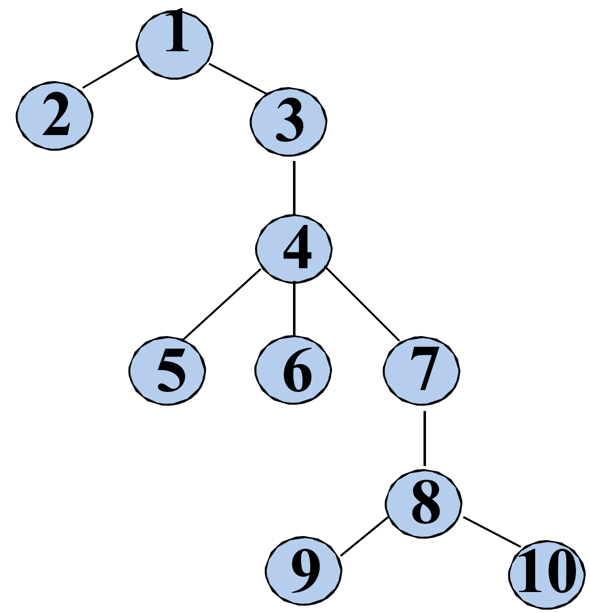
(b)

Example: Dominator Tree



支配结点	支配对象
1	1 ~ 10
2	2
3	3 ~ 10
4	4 ~ 10
5	5
6	6
7	7 ~ 10
8	8 ~ 10
9	9
10	10

➢ **支配结点树** (*Dominator Tree*)



在Dom tree中, 每个结点只支配它和它的后代结点

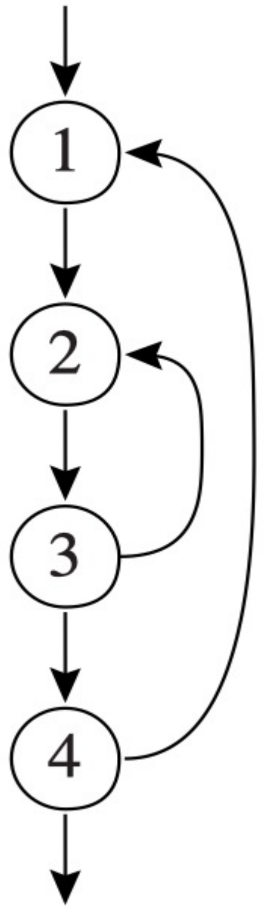
1. Loops and Dominators

- Loop optimizations
- Dominators
- Back to Loops
- Loop Preheader

Natural Loops (自然循环)

- **Back Edge**: an edge from n to h when h dominates n (h dom n)

- The **natural loop** of a back edge $n \rightarrow h$ is the set of nodes x such that
  1. h dominates x and
  2. there is a path from x to n not containing h



The target of a back edge (h) is a loop header!

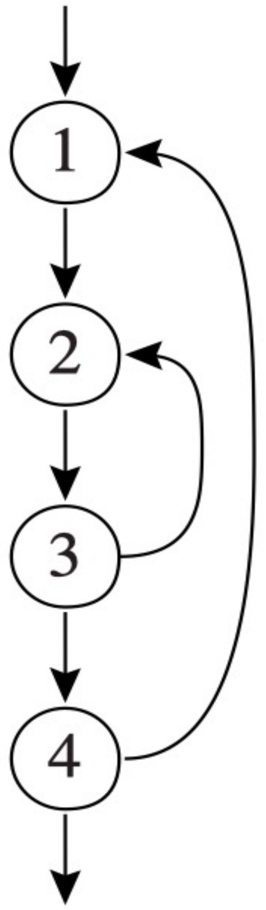
Short Summary

- Node a dominates node b if every possible execution path that gets to b must pass through a
- A back edge is an edge from b to a when a dominates b
- The target of a back edge is a loop header

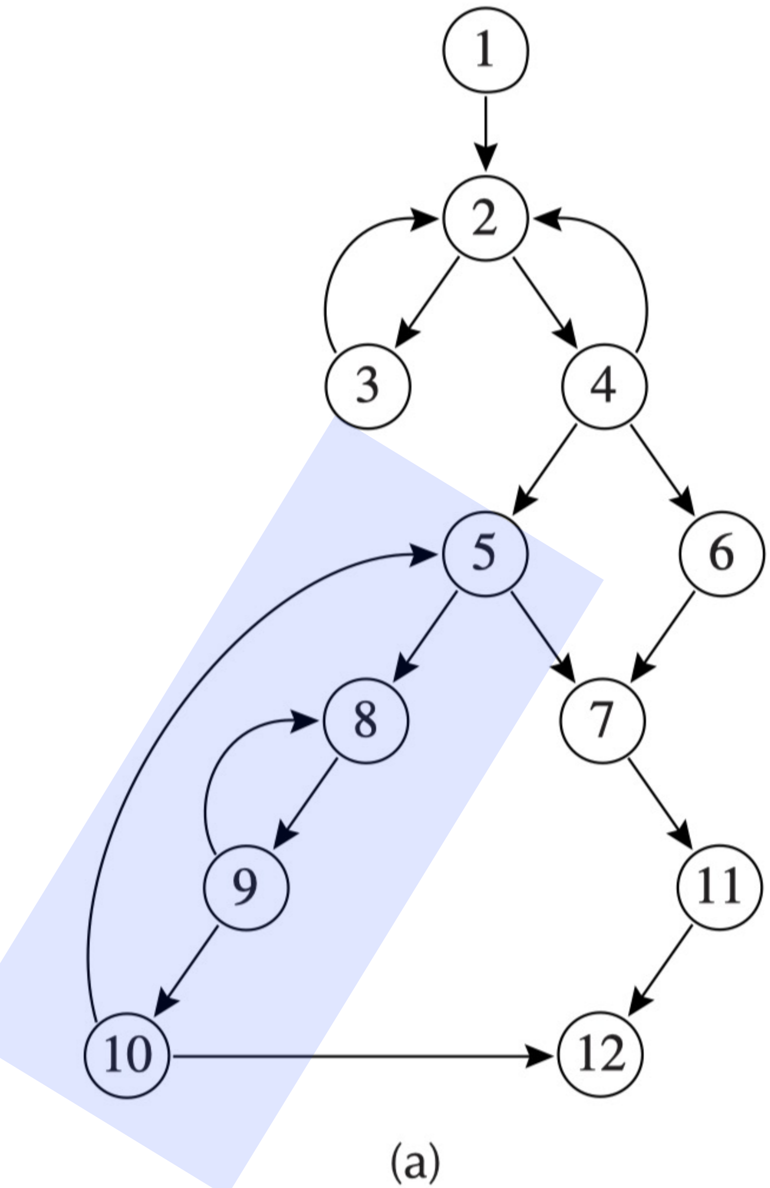
Natural Loops (自然循环)

- **从编译优化的角度看，循环在代码中以什么形式出现并不重要，重要的是它是否具有易于优化的性质**

- **自然循环满足以下性质**
  - 有唯一的入口结点，称为首结点(*header*)。首结点支配循环中的所有结点
  - 循环中至少有一条返回首结点的路径



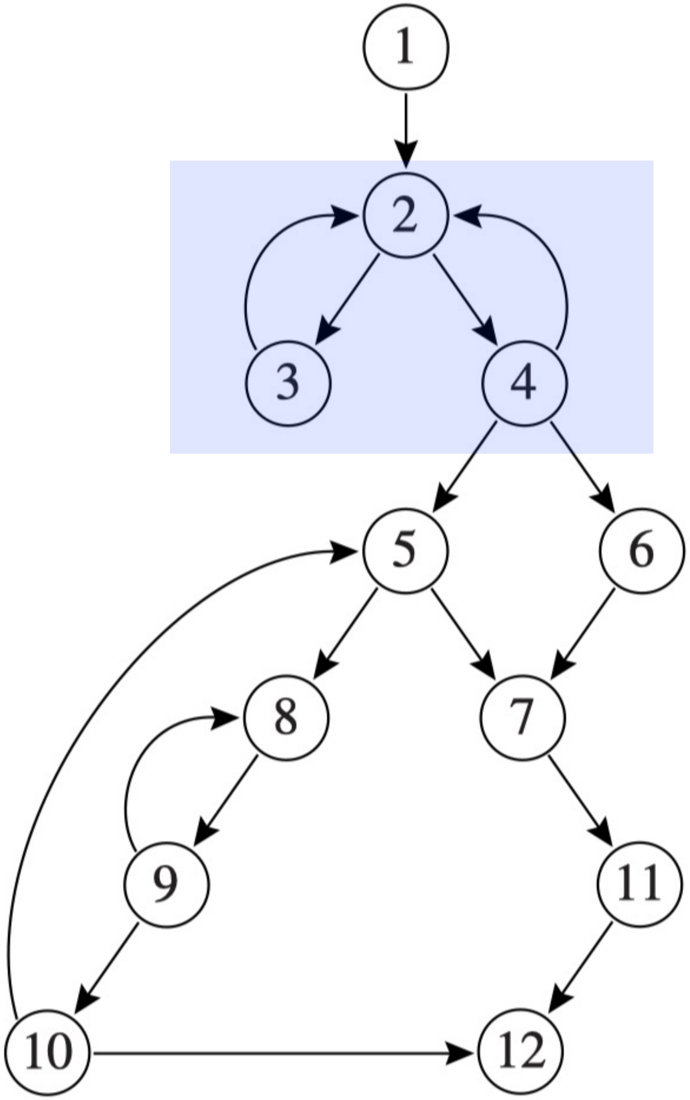
Example: Natural Loops



(a)

- The natural loop of the back edge 10 -> 5 includes nodes 5, 8, 9, 10, and has the loop 8, 9 nested within it.

Example: Natural Loops

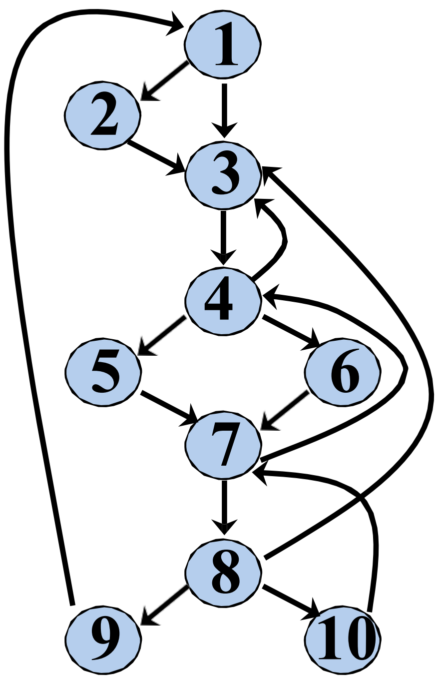


(a)

- The natural loop of the back edge 10 -> 5 includes nodes 5, 8, 9, 10, and has the loop 8, 9 nested within it.
- A node *h* can be the header of more than one natural loop
  - 3->2 => {3, 2}
  - 4->2 => {4, 2]

Properties of Natural Loops

- 除非两个自然循环的首结点相同，否则，它们或者**互不相交,** 或者一个**完全包含(嵌入)在另外一个里面**



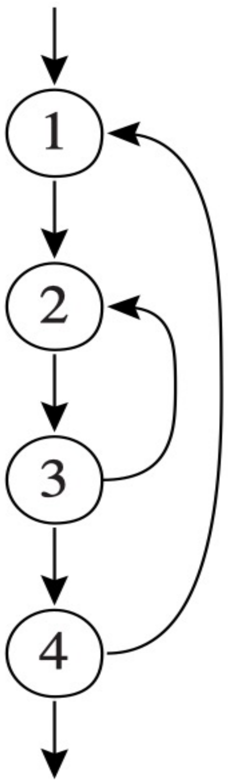
回边	自然循环
4->3	③④⑤⑥⑦⑧⑩
7->4	④⑤⑥⑦⑧⑩
8->3	③④⑤⑥⑦⑧⑩
9->1	① ~ ⑩
10->7	⑦⑧⑩

最内循环 (*Innermost Loops*):
不包含其它循环的循环

Nested Loop

- If loops A and B have distinct headers and all nodes in B are in A (i.e., B⊆A), then we say B is nested within A

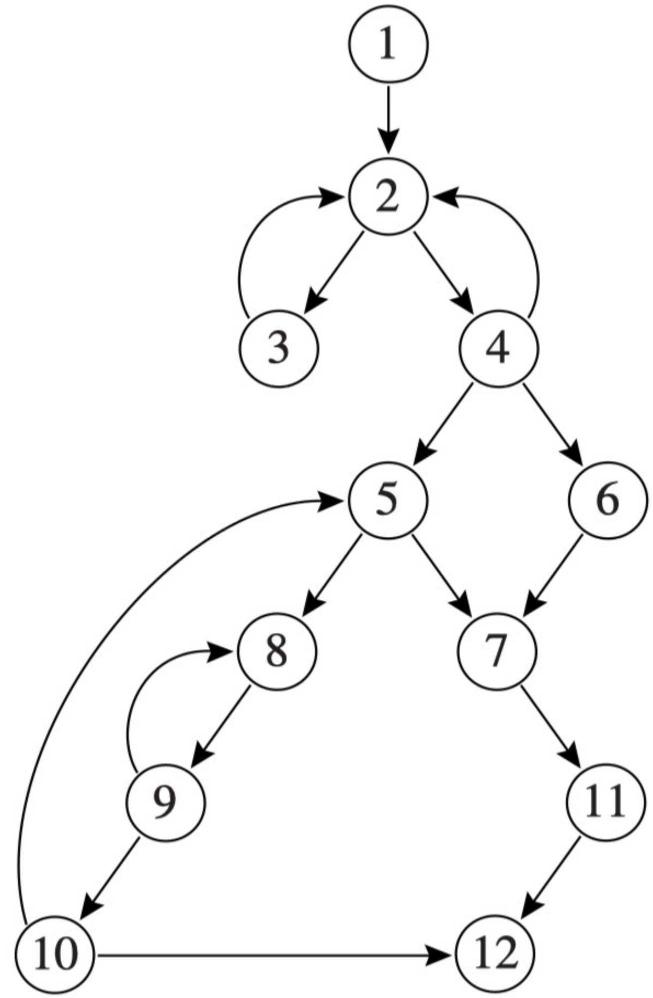
- B is an inner loop



Loop-Nest Tree

We can construct a **loop-nest tree** of loops in a program:

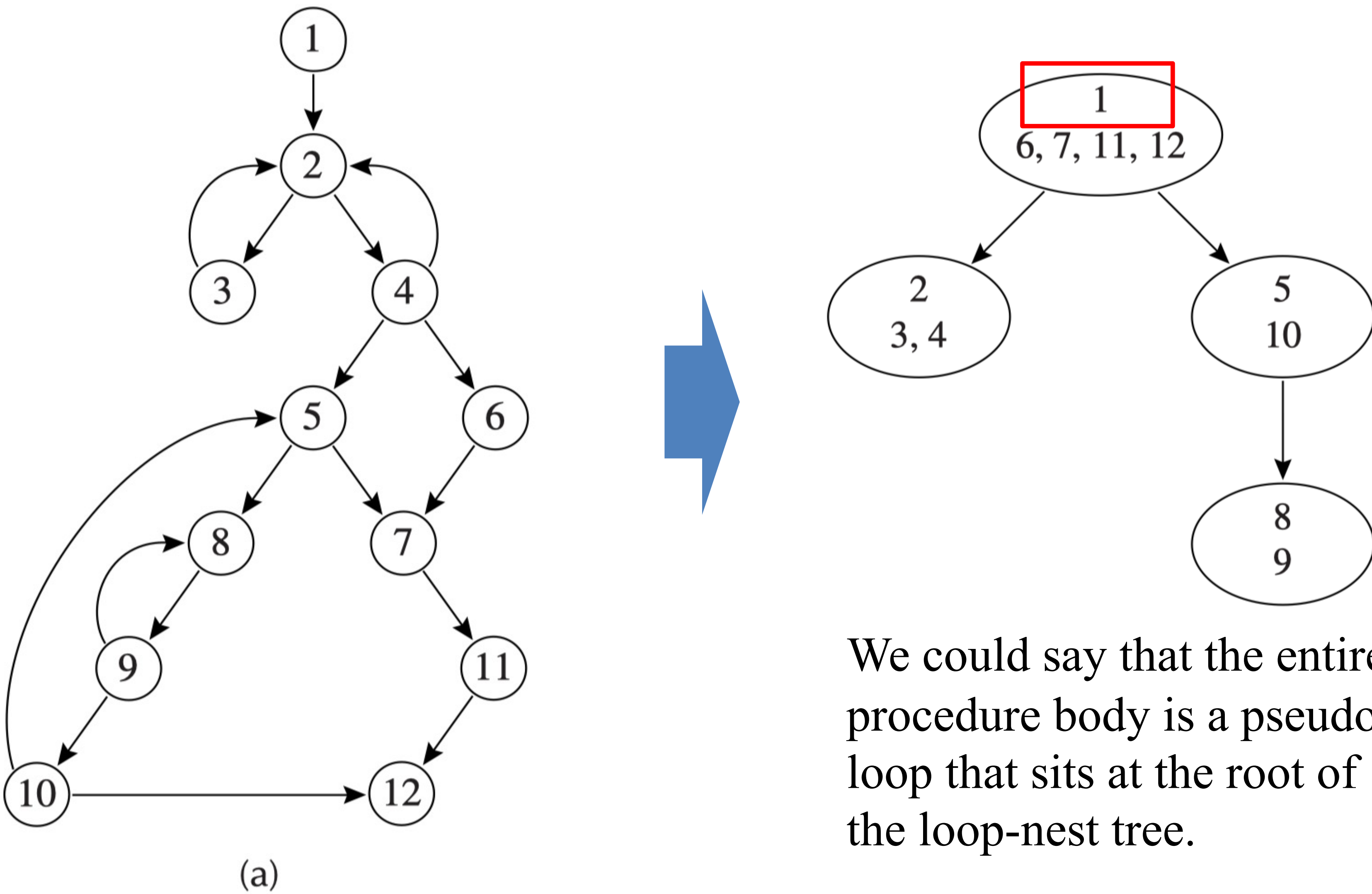
1. Compute dominators of the flow graph G.
2. Construct the dominator tree.
3. Find all the natural loops, and thus all the loop-header nodes.
4. For each loop header *h*, merge all the natural loops of *h* into a single loop, loop[h].
5. Construct the tree of loop headers (and implicitly loops), such that *h1* is above *h2* in the tree if *h2* is in *loop[h1]*.



(a)

Loop-Nest Tree

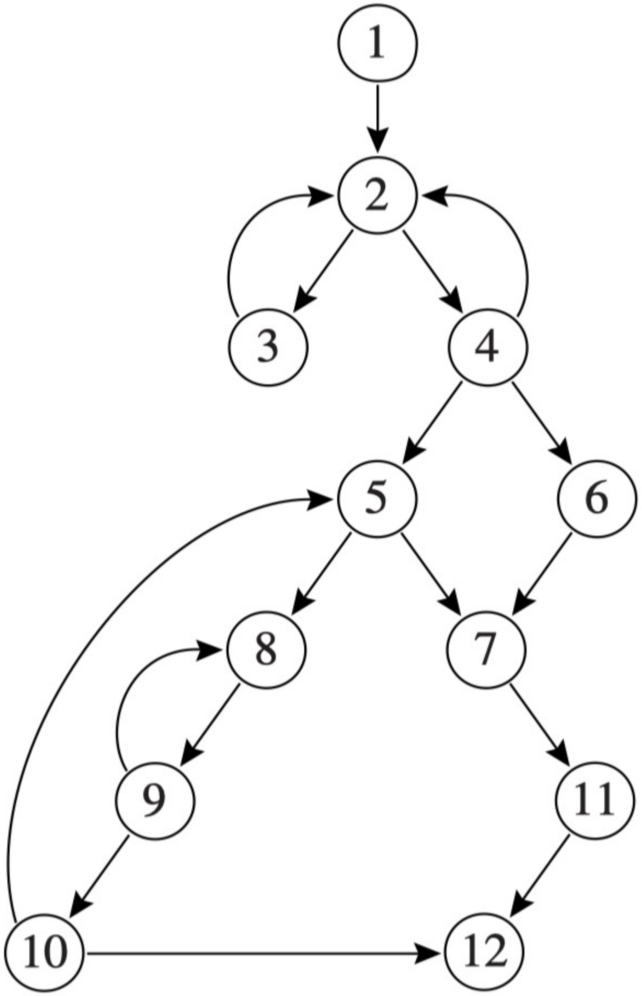
We can construct a **loop-nest tree** of loops in a program:



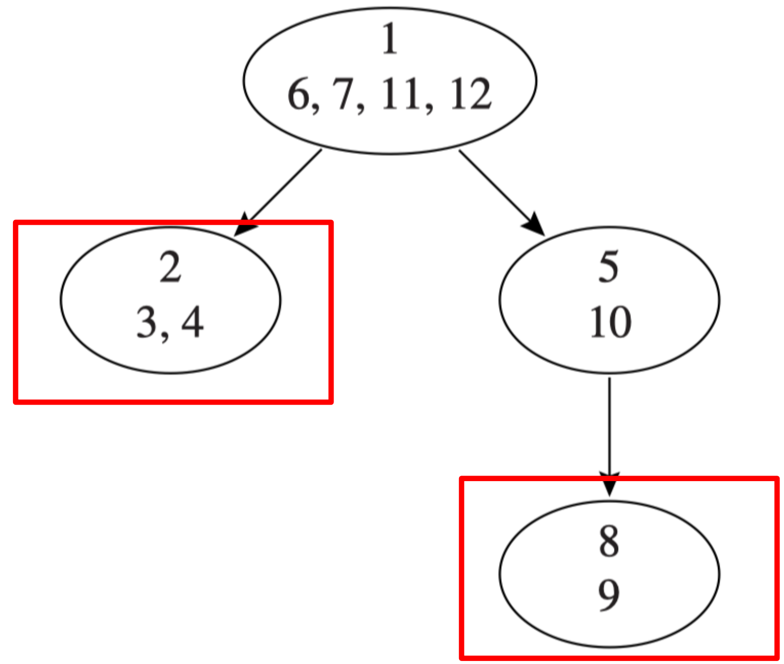
We could say that the entire procedure body is a pseudo-loop that sits at the root of the loop-nest tree.

Loop-Nest Tree

We can construct a **loop-nest tree** of loops in a program:



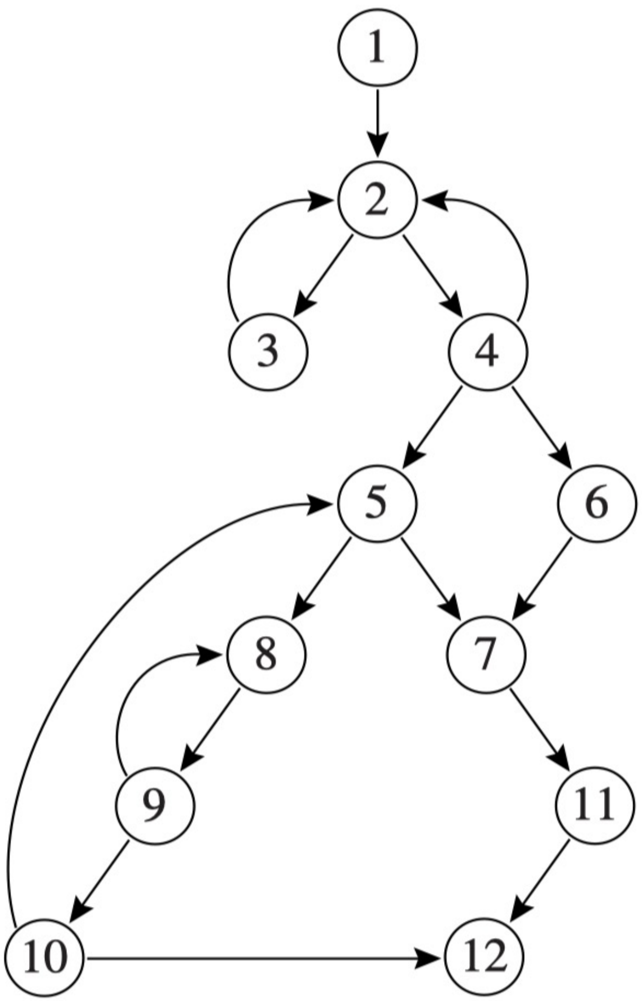
(a)



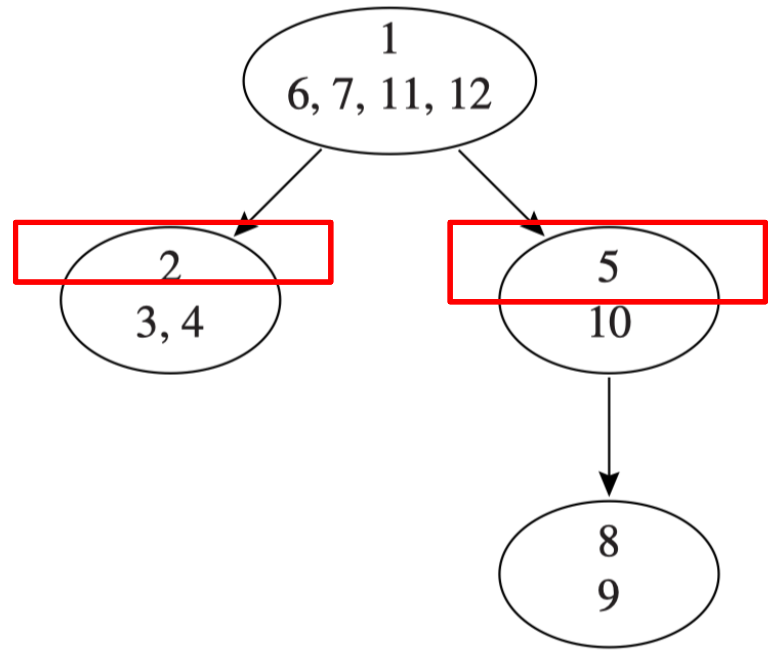
The leaves of the loop-nest tree are the *innermost loops*

Loop-Nest Tree

We can construct a **loop-nest tree** of loops in a program:



(a)



Each loop header is shown in the top half of each oval (nodes 1, 2, 5, 8)

1. Loops and Dominators

- Loop optimizations
- Dominators
- Back to Loops
- Loop Preheader

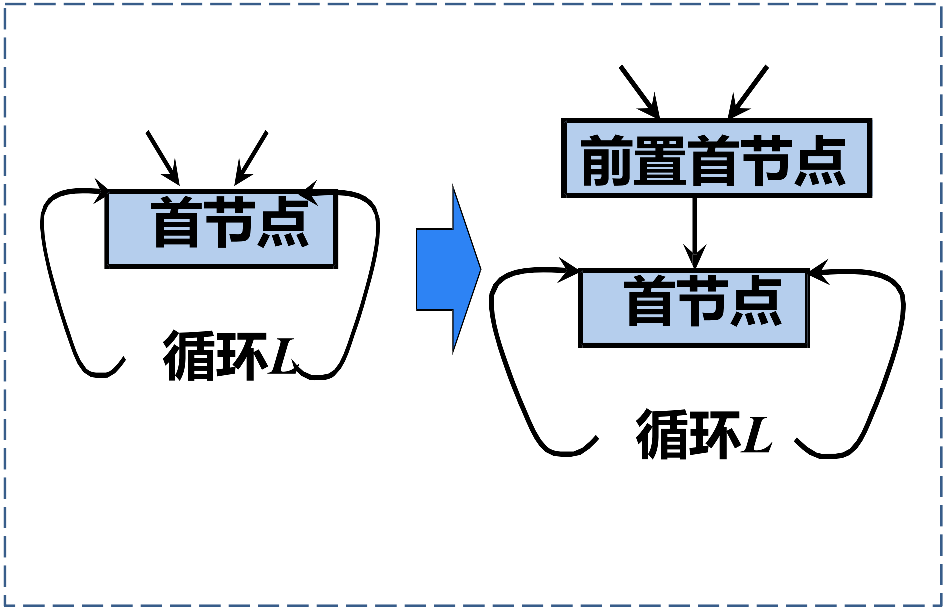
Loop Preheader (前置首结点)

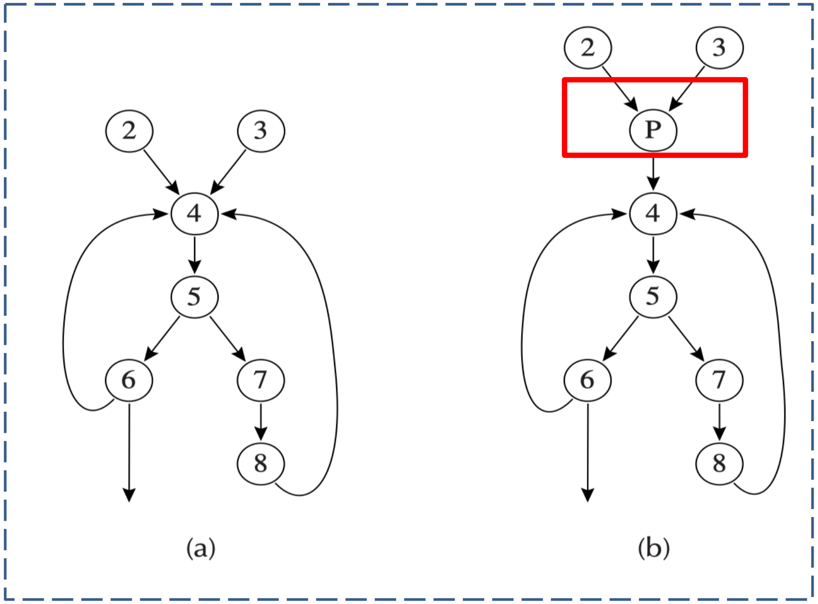
- Many loop optimizations will insert statements immediately before the loop executes.

- E.g., *loop-invariant hoisting*
  – move a statement from inside the loop to immediately before the loop

Loop Preheader (前置首结点)

- **前置首结点**的唯一后继是首结点
  1. 原来从循环L外到达L首结点的边都改成进入前置首结点
  2. 从循环 L里面到达首结点的边不变





2. Loop Invariant Hoisting

- Loop Invariant
- Loop Invariant Hoisting

Loop Invariant Code Motion

- **Idea**: some expressions evaluated in a loop never change; they are loop invariant
  - Can move loop invariant expressions outside the loop, store result in temporary and just use the temporary in each iteration
  - Why is this useful?

- How?

Loop Invariant Code Motion

- Two steps: analysis and transformations
- Step1: find invariant computations in loop
  - invariant: computes same result each time evaluated
- Step 2: move them outside loop
  - to top if used within loop: **code hoisting**
  - to bottom if used after loop: **code sinking**

Identifying Loop Invariants

- An assignment x := v1 op v2 is **invariant** for a loop if for each operand v1 and v2 either
  1. The operand is constant, or
  2. All of the definitions that reach the assignment are outside the loop, or
  3. Only one definition reaches the assignment and it is a loop invariant

Can use an iterative algorithm to compute

Identifying Loop Invariants (Another Definition)

- An expression x := v1 op v2 is invariant in a loop L iff:

(base cases)

  - it's a constant
  - it's a variable use, all of whose defs are outside of L

(inductive cases)

  - it's a pure computation all of whose arguments are loop-invariant
  - it's a variable use with only one reaching def, and the rhs of that def is loop-invariant

2. Loop Invariant Hoisting

- Loop Invariant
- Loop Invariant Hoisting

Example: Loop Invariant Hoisting

- t = a + b is a loop invariant

```
L0:   t := 0


L1:   i := i + 1
      t := a + b
      *i := t
      if i<N goto L1 else L2

L2:   x := t
```

Example: Loop Invariant Hoisting

- Move t = a + b for optimization

```
L0:   t := 0
      t := a + b


L1:   i := i + 1

      *i := t
      if i<N goto L1 else L2

 L2:   x := t
```

Example: Invalid Hoisting

- Can we always move t as long as it is an invariant?

```
L0:  t := 0

L1:  i := i + 1
     *i:= t
     t := a + b
     if i<N goto L1 else L2

L2:  x := t
```

Although t's definition is loop invariant, hoisting conflicts with this use of t

Criteria for Safe Hoisting

- Just because code is loop invariant doesn't mean we can move it! (should preserve the semantics)

- The **criteria** for hoisting $d : t \leftarrow a \oplus b$ to the end of the loop preheader:
  1. d dominates all loop exits where t is live-out
  2. and there is only one definition of t in the loop
  3. and t is not live-out of the loop preheader (that is, t is not live before the loop)

Example: Can We Hoist $t \leftarrow a \oplus b$?

1. d dominates all loop exits where t is live-out
2. and there is only one definition of t in the loop
3. and t is not live-out of the loop preheader

L_0 $t \leftarrow 0$ L_1 $i \leftarrow i+1$ $t \leftarrow a \oplus b$ $M[i] \leftarrow t$ if $i < N$ goto L_1 L_2 $x \leftarrow t$	L_0 $t \leftarrow 0$ L_1 if $i \geq N$ goto L_2 $i \leftarrow i+1$ $t \leftarrow a \oplus b$ $M[i] \leftarrow t$ goto L_1 L_2 $x \leftarrow t$	L_0 $t \leftarrow 0$ L_1 $i \leftarrow i+1$ $t \leftarrow a \oplus b$ $M[i] \leftarrow t$ $t \leftarrow 0$ $M[j] \leftarrow t$ if $i < N$ goto L_1 L_2	L_0 $t \leftarrow 0$ L_1 $M[j] \leftarrow t$ $i \leftarrow i+1$ $t \leftarrow a \oplus b$ $M[i] \leftarrow t$ if $i < N$ goto L_1 L_2 $x \leftarrow t$

correct
faster

Example: Can We Hoist $t \leftarrow a \oplus b$?

- **1. *d* dominates all loop exits where *t* is live-out**

  The original program does not *always* execute $t \leftarrow a \oplus b$, but the transformed program does, producing an incorrect value for x if $i \geq N$ initially

L_0 $t \leftarrow 0$ L_1 $i \leftarrow i+1$ $t \leftarrow a \oplus b$ $M[i] \leftarrow t$ if $i < N$ goto L_1 L_2 $x \leftarrow t$	L_0 $t \leftarrow 0$ L_1 if $i \geq N$ goto L_2 $i \leftarrow i+1$ $t \leftarrow a \oplus b$ $M[i] \leftarrow t$ goto L_1 L_2 $\boxed{x \leftarrow t}$	L_0 $t \leftarrow 0$ L_1 $i \leftarrow i+1$ $t \leftarrow a \oplus b$ $M[i] \leftarrow t$ $t \leftarrow 0$ $M[j] \leftarrow t$ if $i < N$ goto L_1 L_2	L_0 $t \leftarrow 0$ L_1 $M[j] \leftarrow t$ $i \leftarrow i+1$ $t \leftarrow a \oplus b$ $M[i] \leftarrow t$ if $i < N$ goto L_1 L_2 $x \leftarrow t$
correct faster	incorrect not always execute the def		

Example: Can We Hoist $t \leftarrow a \oplus b$?

- **2. and there is only one definition of t in the loop**

The original loop had more than one definition of t, and the transformed program interleaves the assignments to t in a different way

L_0	L_0	L_0	L_0
$t \leftarrow 0$	$t \leftarrow 0$	$t \leftarrow 0$	$t \leftarrow 0$
L_1	L_1	L_1	L_1
$i \leftarrow i + 1$	if $i \geq N$ goto L_2	$i \leftarrow i + 1$	$M[j] \leftarrow t$
$t \leftarrow a \oplus b$	$i \leftarrow i + 1$	$t \leftarrow a \oplus b$	$i \leftarrow i + 1$
$M[i] \leftarrow t$	$t \leftarrow a \oplus b$	$M[i] \leftarrow t$	$t \leftarrow a \oplus b$
if $i < N$ goto L_1	$M[i] \leftarrow t$	$\boxed{t \leftarrow 0}$	$M[i] \leftarrow t$
L_2	goto L_1	$M[j] \leftarrow t$	if $i < N$ goto L_1
$x \leftarrow t$	L_2	if $i < N$ goto L_1	L_2
	$x \leftarrow t$	L_2	$x \leftarrow t$

correct faster	incorrect not always execute the def	incorrect more than one def of t

Example: Can We Hoist $t \leftarrow a \oplus b$?

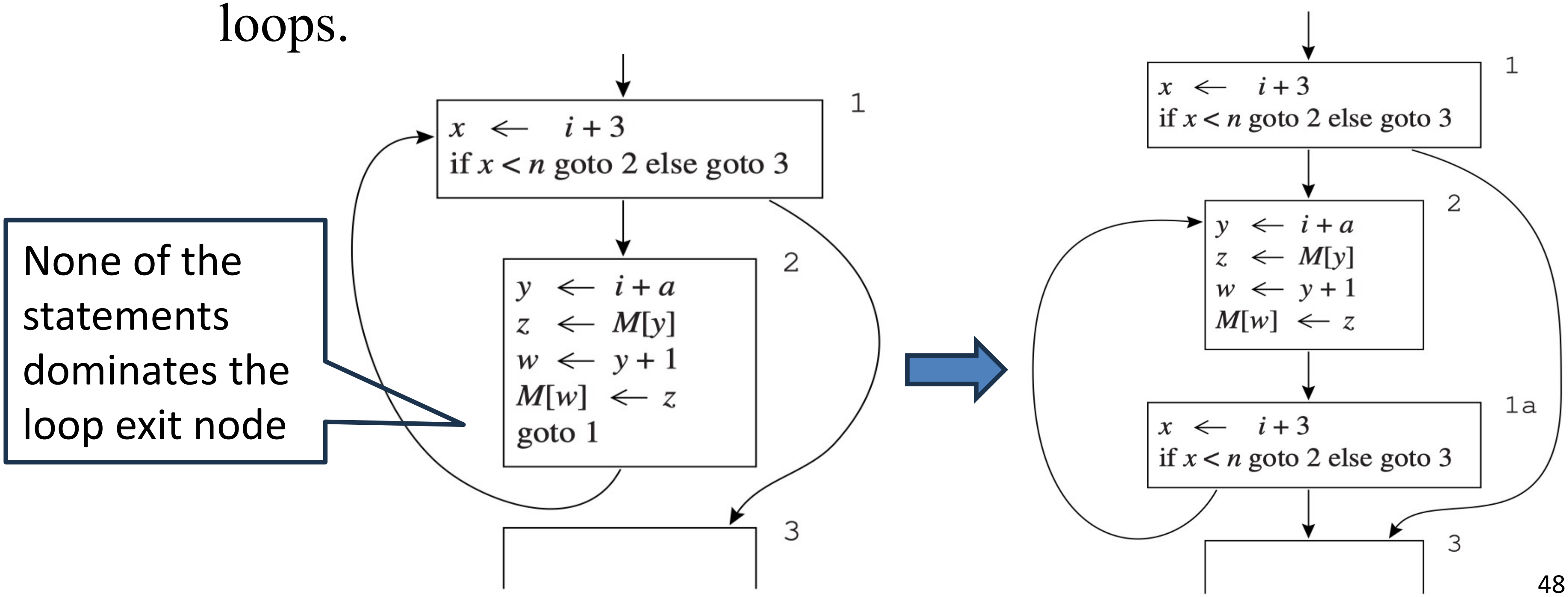
- **3. and t is not live-out of the loop preheader**

  There is a use of t before the loop-invariant definition, so after hoisting, this use will have the wrong value on the first iteration of the loop.

L_0 $t \leftarrow 0$ L_1 $i \leftarrow i+1$ $t \leftarrow a \oplus b$ $M[i] \leftarrow t$ if $i < N$ goto L_1 L_2 $x \leftarrow t$	L_0 $t \leftarrow 0$ L_1 if $i \geq N$ goto L_2 $i \leftarrow i+1$ $t \leftarrow a \oplus b$ $M[i] \leftarrow t$ goto L_1 L_2 $x \leftarrow t$	L_0 $t \leftarrow 0$ L_1 $i \leftarrow i+1$ $t \leftarrow a \oplus b$ $M[i] \leftarrow t$ $t \leftarrow 0$ $M[j] \leftarrow t$ if $i < N$ goto L_1 L_2	L_0 $t \leftarrow 0$ L_1 $\boxed{M[j] \leftarrow t}$ $i \leftarrow i+1$ $t \leftarrow a \oplus b$ $M[i] \leftarrow t$ if $i < N$ goto L_1 L_2 $x \leftarrow t$
correct faster	incorrect not always execute the def	incorrect more than one def of t	incorrect a use of t before the def

Hoisting

- **Implicit side effects.** These rules need modification if $t \leftarrow a \oplus b$ could raise some sort of arithmetic exception or have other side effects.
- **Turning while loops into repeat-until loops.**
  - "d dominates all loop exits at which t is live-out": tends to prevent many computations from being hoisted from **while** loops.



期末考试

- 时间：2024年6月25日(10:30-12:30)
- 地点：玉泉教7-102，教7-104
- 半开卷（允许带3张A4纸，打印或手写，可正反面）
- 题型：判断、选择、问答
- 考试范围：课程（14、18章无问答题）