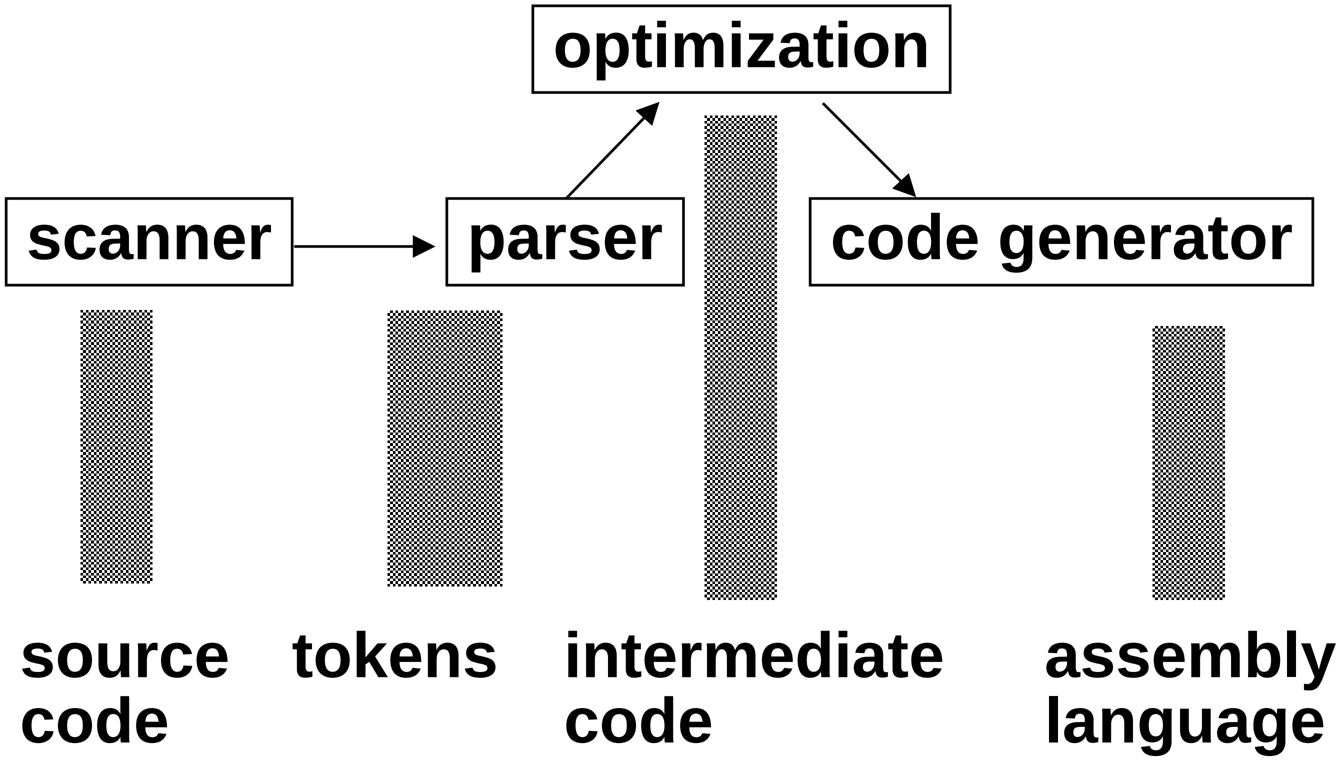
Machine Independent Optimizations - 2

- **What is optimization - an exploration through examples**
- **Machine independent optimizations**
  - **General code motion**
  - **Global common subexpression elimination**

Compilation



optimization

scanner

parser

code generator

source code

tokens

intermediate code

assembly language

Optimization is a semantics preserving operation

Example

Fortran Source Code:

```
          .
          .
          .
          sum = 0
          do 10 i = 1, n
10        sum = sum + a(i) * a(i)
          .
          .
          .
```

3 Address Code

```
1.  sum = 0
2.  i = 1
3.  if i > n goto 15
4.  t1 = addr(a) - 4
5.  t2 = i * 4
6.  t3 = t1[t2]
7.  t4 = addr(a) - 4
8.  t5 = i * 4
9.  t6 = t4[t5]
10. t7 = t3 * t6
11. t8 = sum + t7
12. sum = t8
13. i = i + 1
14. goto 3
15.
```

Machine Independent Optimization

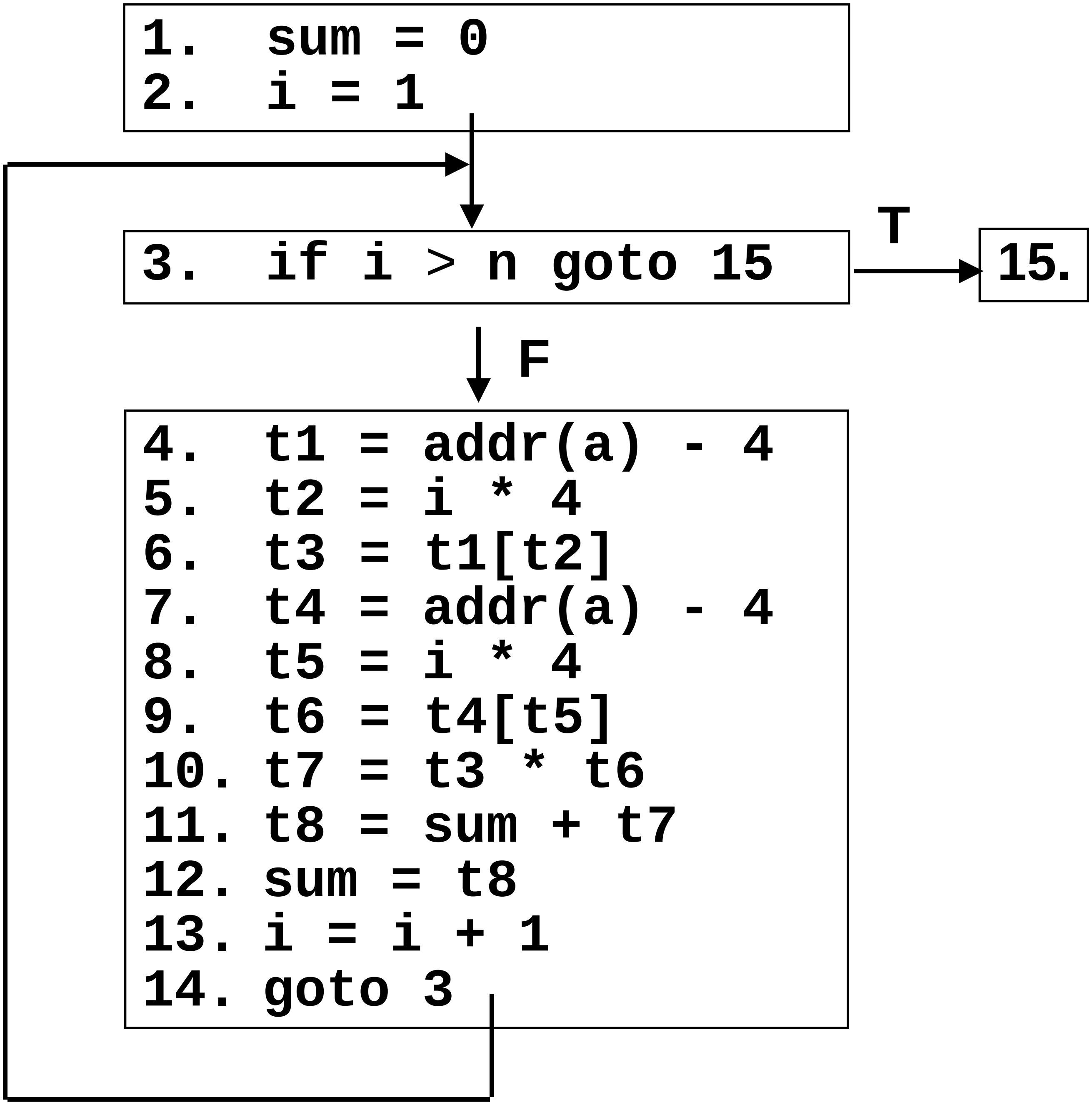
3-Address Code	Source
`1. sum = 0` `2. i = 1`	sum = 0
`3. if i > n goto 15` `4. t1 = addr(a) - 4`	init for loop and check limit

`5. t2 = i * 4`
`6. t3 = t1[t2]`
`7. t4 = addr(a) - 4`
	a[i]

`8. t5 = i * 4` `9. t6 = t4[t5]`	a[i]
`10. t7 = t3 * t6` `11. t8 = sum + t7`	a[i] * a[i]
`12. sum = t8`	increment sum

`13. i = i + 1`
`14. goto 3`
`15.`
	increment loop counter

Control Flow Graph



```
1.  sum = 0
2.  i = 1
```

```
3.  if i > n goto 15
```

```
4.  t1 = addr(a) - 4
5.  t2 = i * 4
6.  t3 = t1[t2]
7.  t4 = addr(a) - 4
8.  t5 = i * 4
9.  t6 = t4[t5]
10. t7 = t3 * t6
11. t8 = sum + t7
12. sum = t8
13. i = i + 1
14. goto 3
```

Example

```
1.    sum = 0
2.    i = 1
3.    if i > n goto 15
4.    t1 = addr(a) - 4
5.    t2 = i * 4
6.    t3 = t1[t2]
~~7.    t4 = addr(a) - 4~~
~~8.    t5 = i * 4~~
~~9.    t6 = t4[t5]~~
~~10.   t7 = t3 * t6~~
10a.  t7 = t3 * t3
~~11.   t8 = sum + t7~~
11a.  sum = sum + t7
~~12.   sum = t8~~
13.   i = i + 1
14.   goto 3
15.
```

Local Common Subexpression Elimination

Example

1. sum = 0
2. i = 1
2a. t1 = addr[a] - 4
3. if i > n goto 15
~~**4. t1 = addr(a) - 4**~~
5. t2 = i * 4
6. t3 = t1[t2]
10a. t7 = t3 * t3
11a. sum = sum + t7
13. i = i + 1
14. goto 3
15.

Invariant Code Motion

Example

```
1.   sum = 0
2.   i = 1
2a.  t1 = addr[a] - 4
2b.  t2 = i * 4
3.   if i > n goto 15
~~5.   t2 = i * 4~~
6.   t3 = t1[t2]
10a. t7 = t3 * t3
11a. sum = sum + t7
12a. t2 = t2 + 4
13.  i = i + 1
14.  goto 3
15.
```

Reduction in Strength

Example

1. sum = 0
2. i = 1
2a. t1 = addr[a] - 4
2b. t2 = i * 4
2c. t9 = 4 * n
~~**3. if i > n goto 15**~~
3a. if t2 > t9 goto 15
6. t3 = t1[t2]
10a. t7 = t3 * t3
11a. sum = sum + t7
12a. t2 = t2 + 4
~~**13. i = i + 1**~~
14. goto **3a**
15.

Test Elision
and
Elimination of Induction Variables

Example

1. sum = 0
~~**2. i = 1**~~
2a. t1 = addr[a] - 4
~~**2b. t2 = i * 4**~~
2d. t2 = 4
2c. t9 = 4 * n
3a. if t2 > t9 goto 15
6. t3 = t1[t2]
10a. t7 = t3 * t3
11a. sum = sum + t7
12a. t2 = t2 + 4
14. goto 3a
15.

**Constant Propagation
and
Dead Code Elimination**

Example

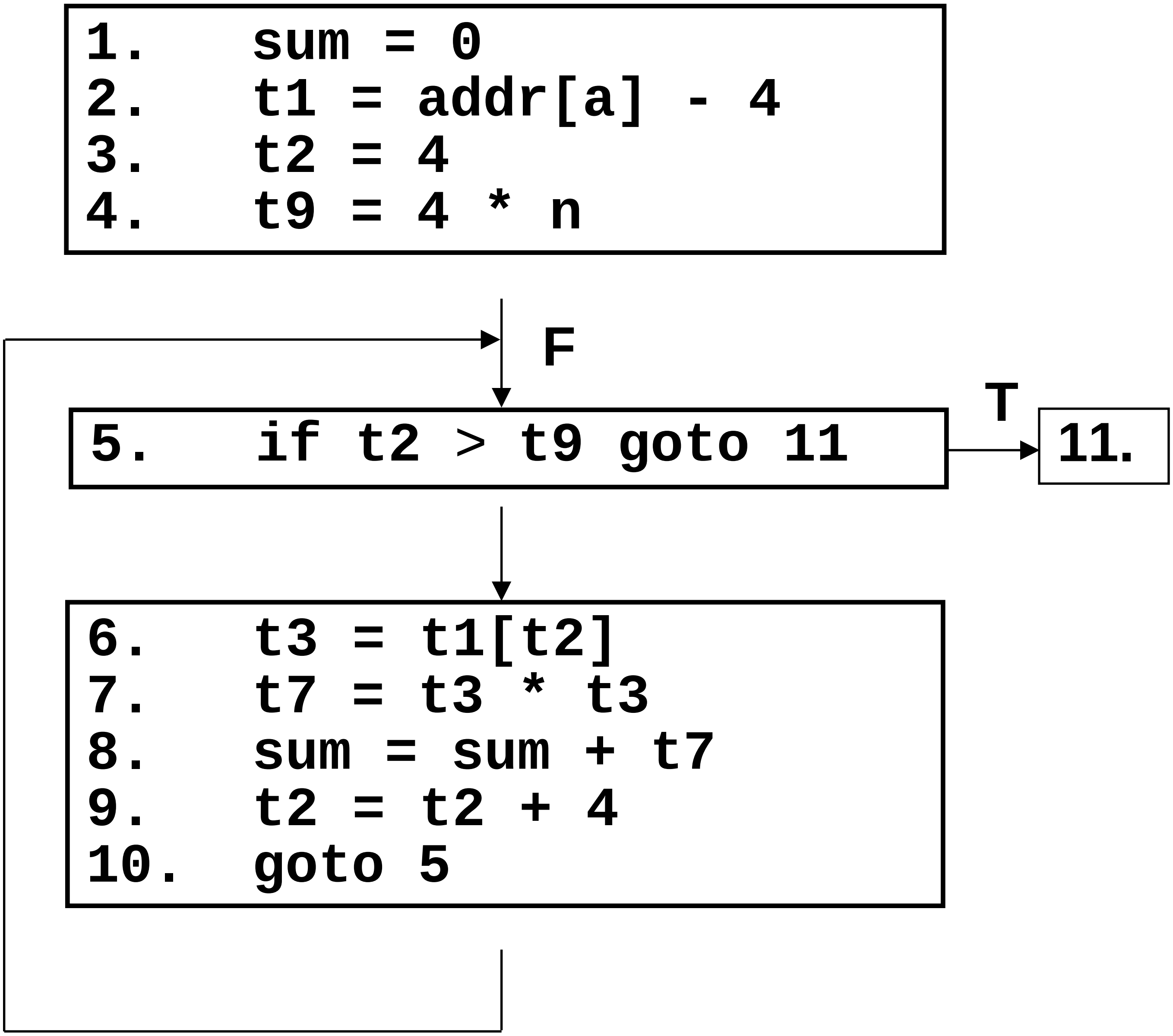
Optimized Code (renumbered)

```
1.    sum = 0
2.    t1 = addr[a] - 4
3.    t2 = 4
4.    t9 = 4 * n
5.    if t2 > t9 goto 11
6.    t3 = t1[t2]
7.    t7 = t3 * t3
8.    sum = sum + t7
9.    t2 = t2 + 4
10.   goto 5
11.
```

Unoptimized:
8 temps, 11 stmts in loop

Optimized:
5 temps, 5 stmts in loop

Example



```
1.    sum = 0
2.    t1 = addr[a] - 4
3.    t2 = 4
4.    t9 = 4 * n
```

```
5.    if t2 > t9 goto 11
```

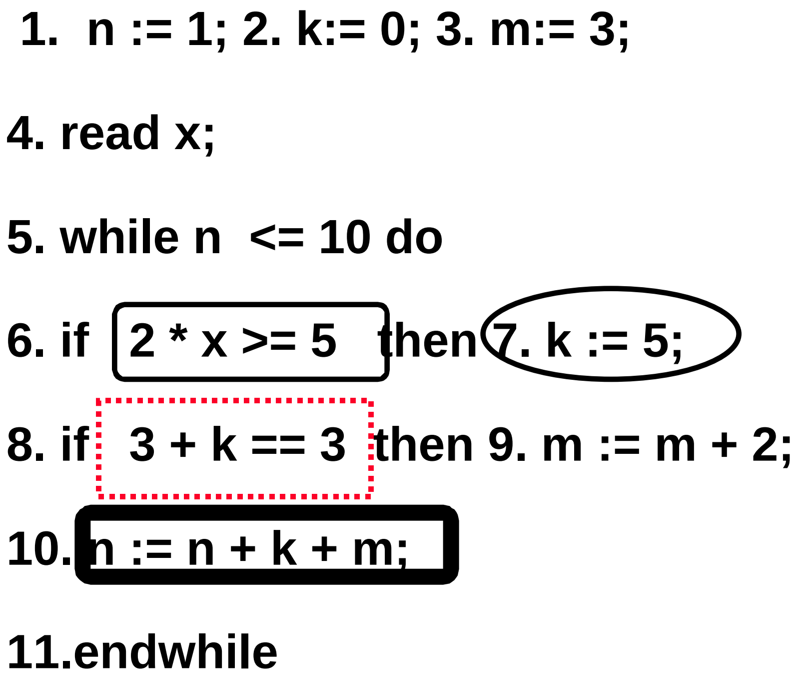
```
11.
```

```
6.    t3 = t1[t2]
7.    t7 = t3 * t3
8.    sum = sum + t7
9.    t2 = t2 + 4
10.   goto 5
```

General Code Motion

```
n := 1; k := 0; m := 3; read x;
while n ≤ 10 do
  if 2 + x ≥ 5 then k := 5;
  if 3 + k = 3 then m := m + 2;
  n := n + k + m;
endwhile;
```

General Code Motion



```
1.  n := 1; 2. k:= 0; 3. m:= 3;
4. read x;
5. while n  <= 10 do
6. if  2 * x >= 5   then 7. k := 5;
8. if  3 + k == 3  then 9. m := m + 2;
10. n := n + k + m;
11.endwhile
```

▭ Invariant within loop and therefore moveable.

⬭ Unaffected by definitions in loop therefore moveable.

⬚ Moveable after we move statement 7.

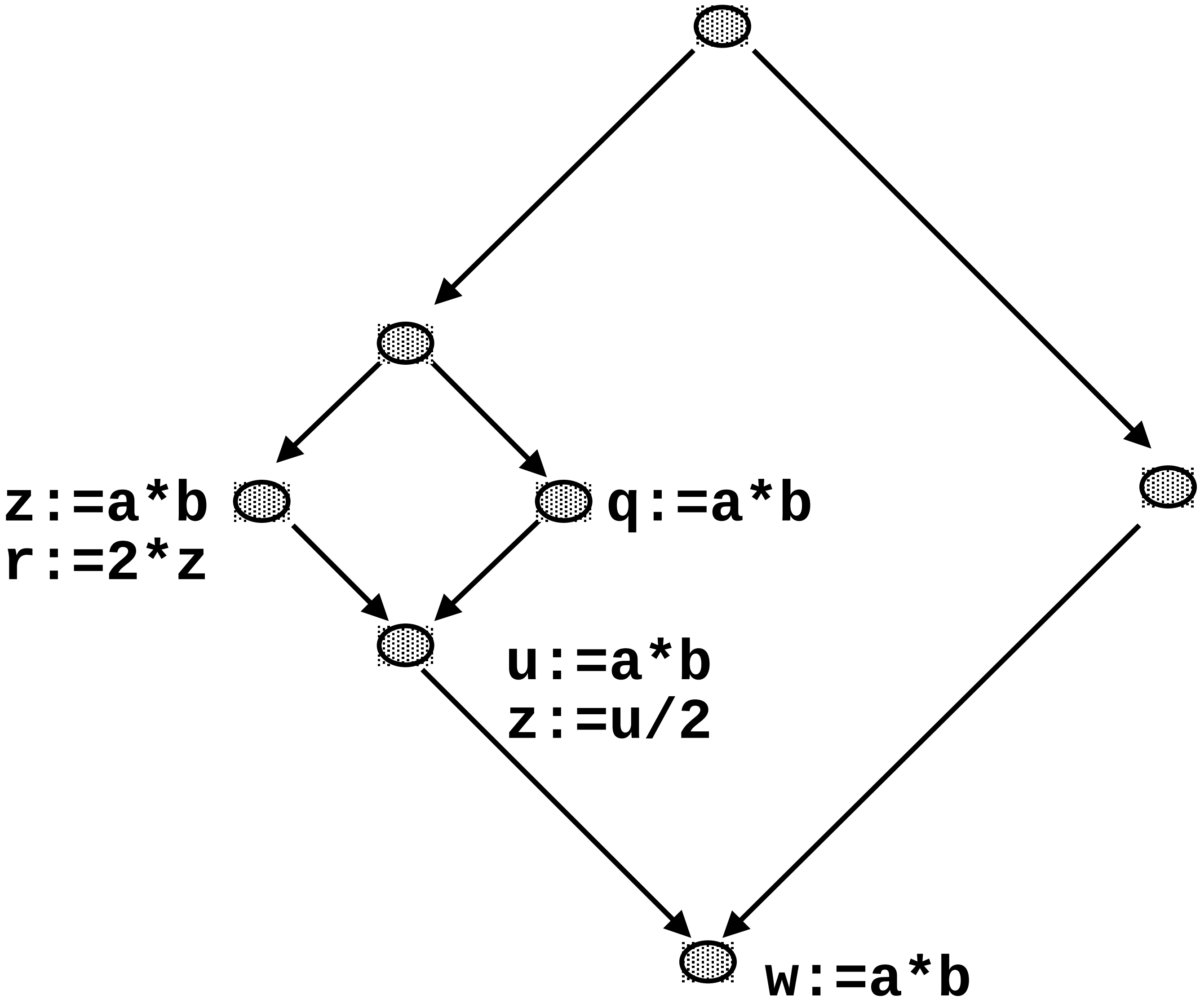
▣ Not moveable because may use def of m from statement 9 on previous iteration.

General Code Motion

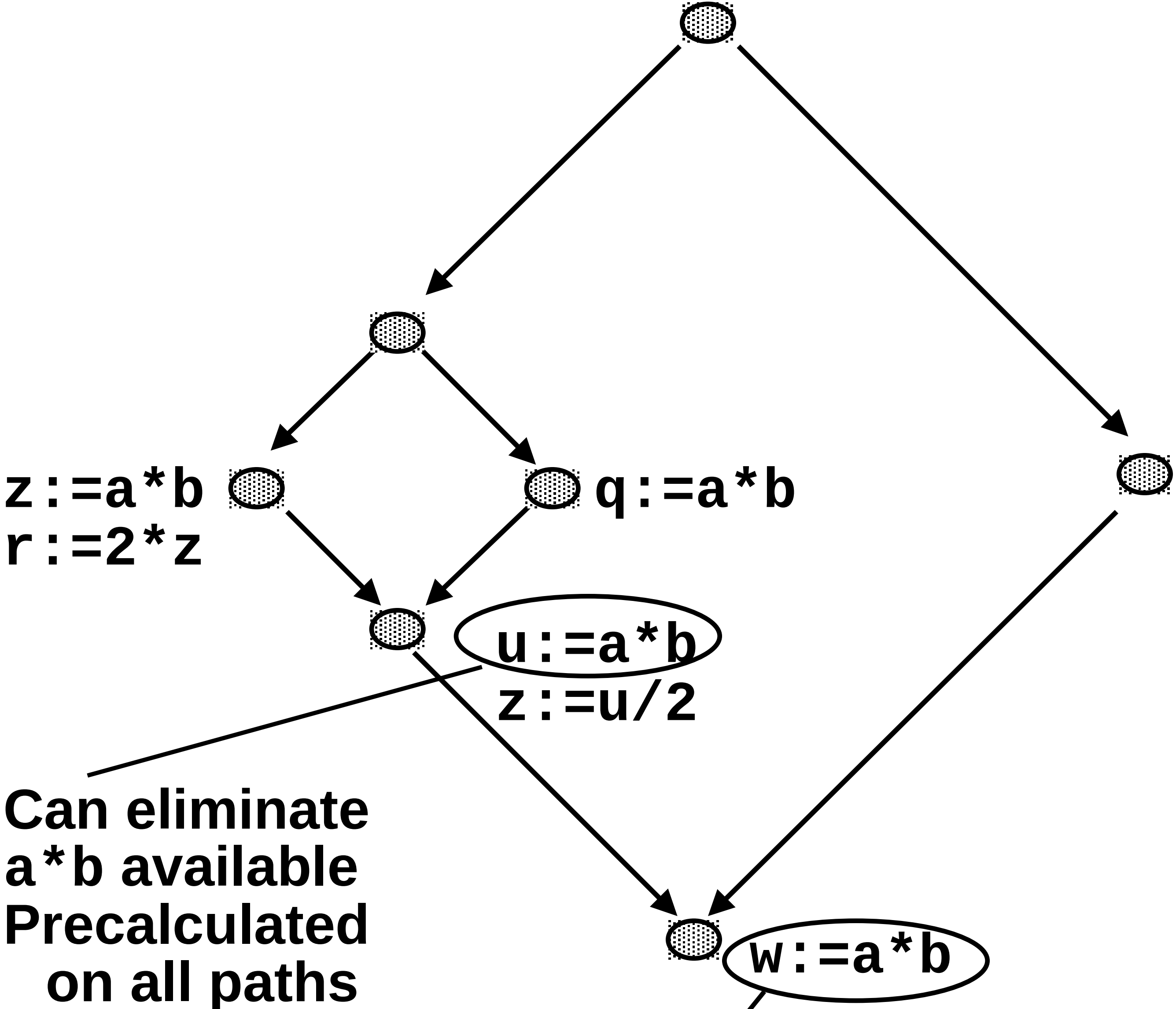
```
n := 1; k := 0; m := 3; read x;
while n ≤ 10 do
  if 2 * x ≥ 5 then k := 5;
  if 3 + k = 3 then m := m + 2;
  n := n + k + m;
endwhile;
```

```
n := 1; k := 0; m := 3; read x;
if 2 * x ≥ 5 then k := 5;
t1 := 3 + k = 3;
while n ≤ 10 do
  if t1 then m := m + 2;
  n := n + k + m;
endwhile;
```

Global Common Subexpression Elimination



Global Common Subexpression Elimination



```
z:=a*b
r:=2*z
```

```
q:=a*b
```

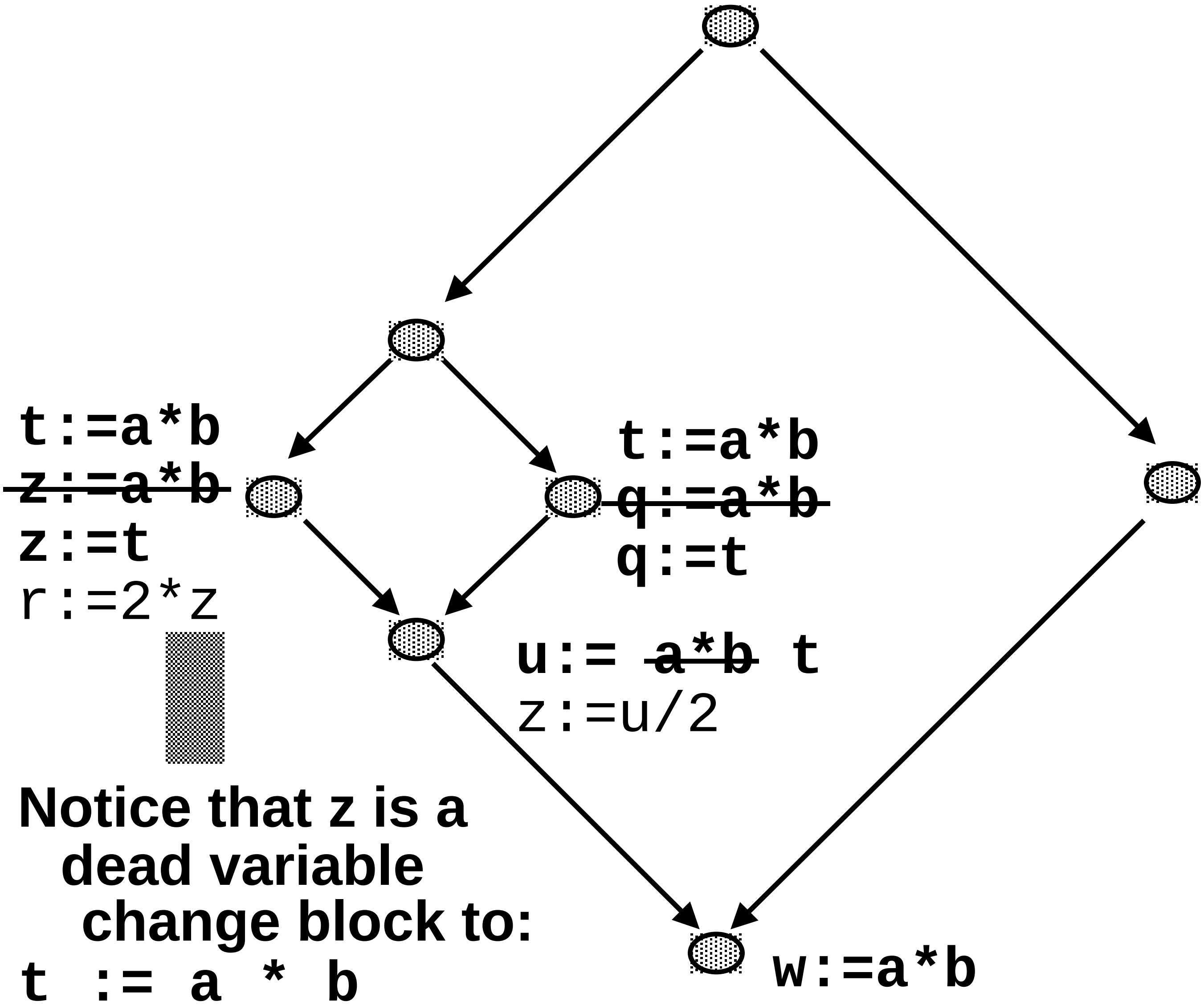
```
u:=a*b
z:=u/2
```

```
w:=a*b
```

Can eliminate
a*b available
Precalculated
on all paths
to this point

Can't eliminate
a*b not available at
this point

Global Common Subexpression Elimination



`t:=a*b`
~~`z:=a*b`~~
`z:=t`
`r:=2*z`

`t:=a*b`
~~`q:=a*b`~~
`q:=t`

`u:=` ~~`a*b`~~ `t`
`z:=u/2`

`w:=a*b`

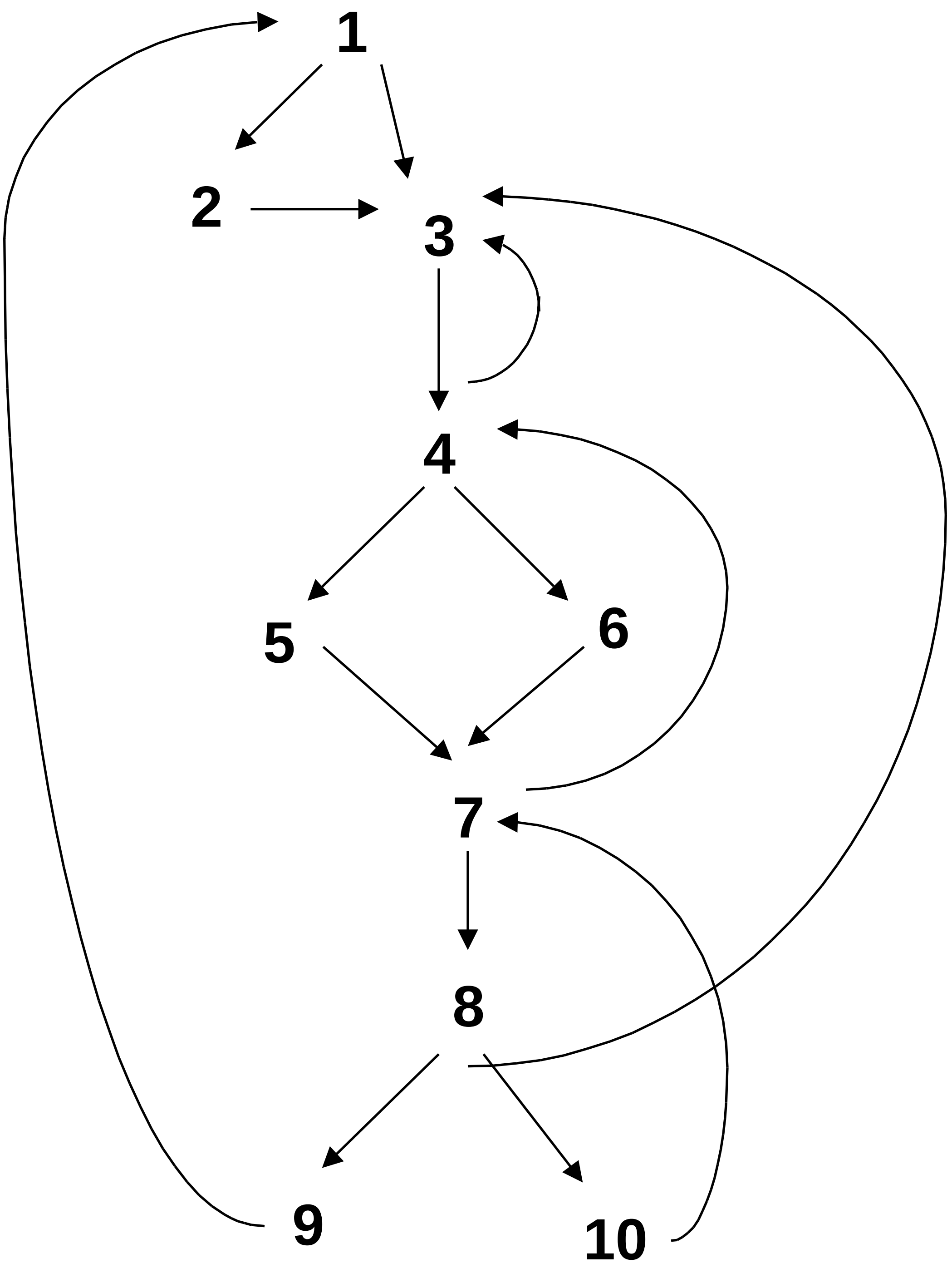
Notice that z is a dead variable change block to:

```
t  :=  a  *  b
r  :=  2  *  t
```

Natural Loops

- Single entry node (header) that dominates all other nodes in loop
- From every node there is at least one path back to the header
- Back edge: edge whose target node dominates its source node.
- Loop construction:
  - Find back edge. Traverse edges in reverse execution direction until back edge target is reached. All nodes encountered in traversal are in corresponding natural loop. (ASU Alg 10.1)
- If 2 back edges go to same header then all nodes in natural loop sets for these edges are in same loop
- If each pair of nodes, one in loop(k) and one in loop(n) are reachable one from the other and header(n) dominates header(k), then loop(k) is nested within loop(n).

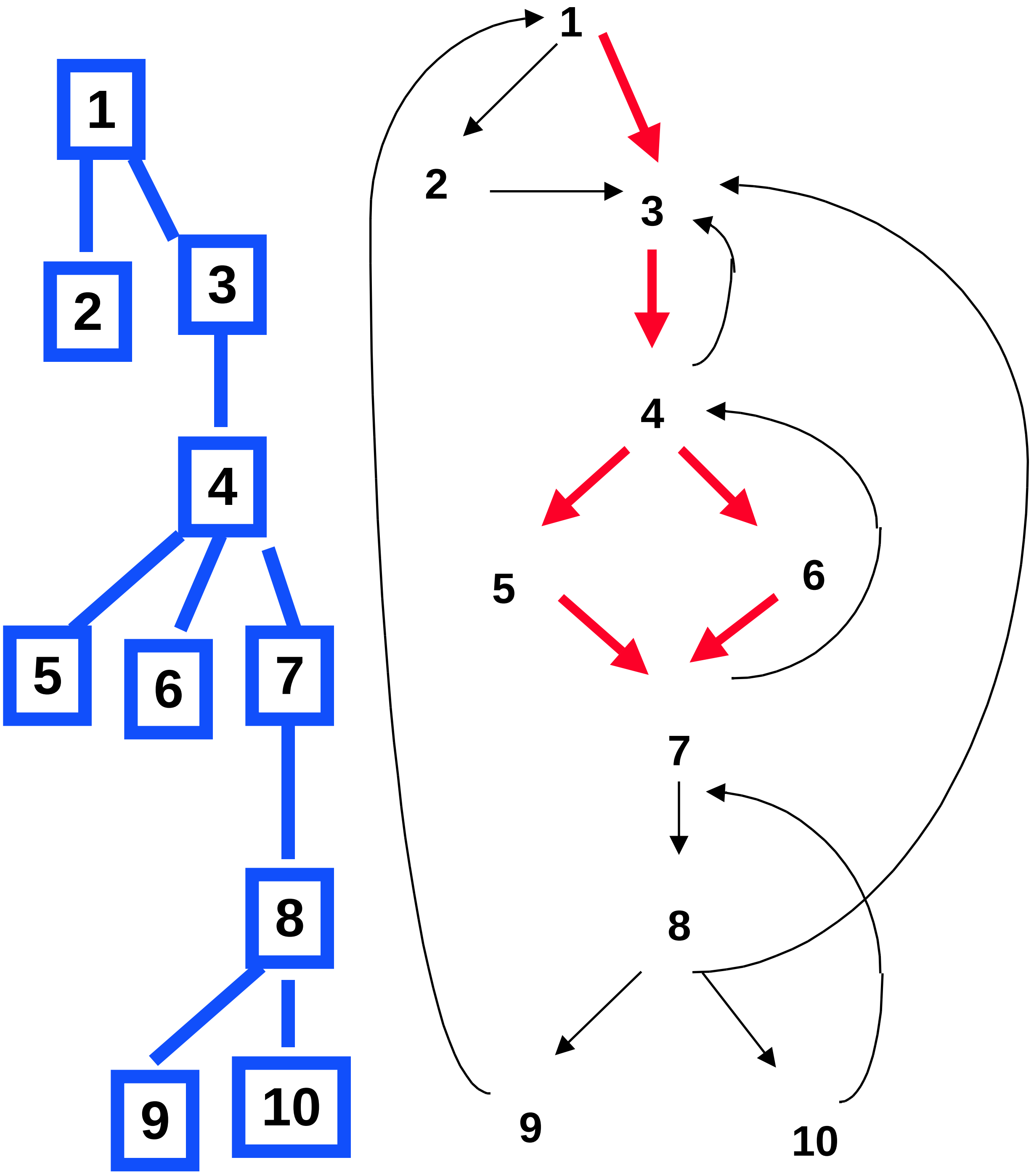
Natural Loops



ASU, P 603

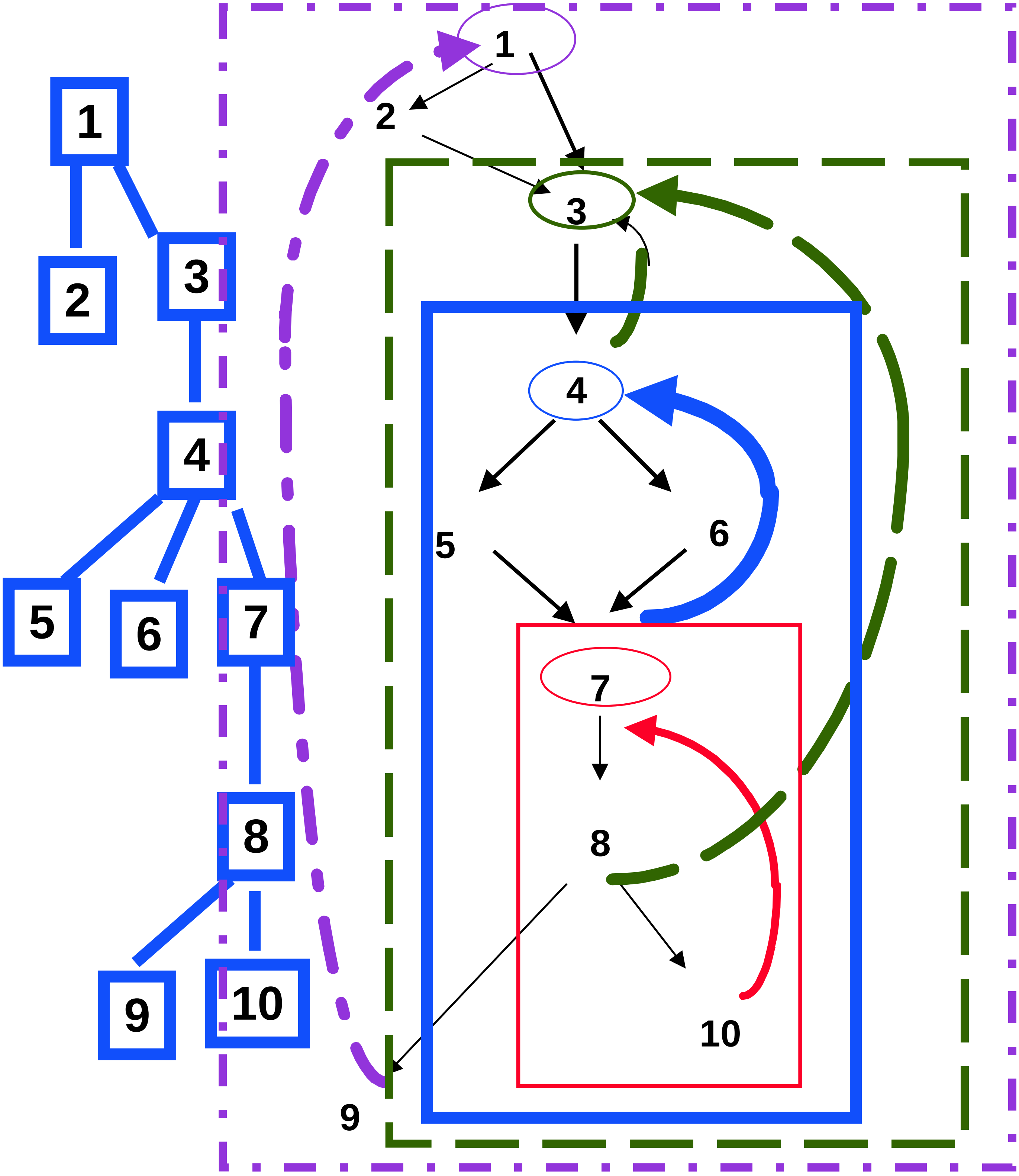
Natural Loops

1 dominates 7



Dominator Tree

Back Edges and Their Loops



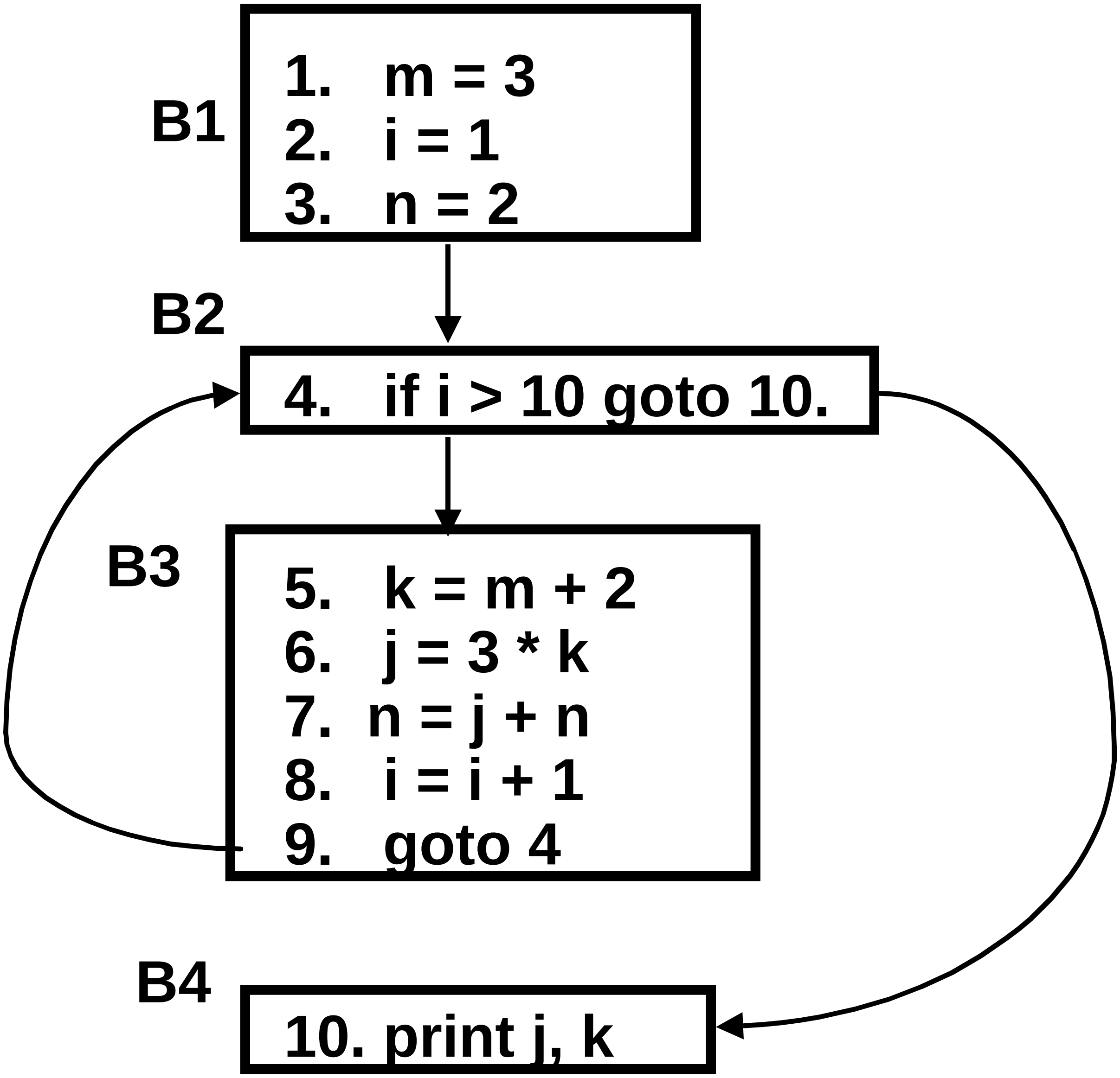
Invariant Code Motion

- **Computation is loop invariant if its value does not change while control stays within the loop**
- **Algm (needs use-def chains):**
  - **Mark invariant all 3 address statements whose operands are constant or have all reaching definitions from outside the loop.**
  - **Mark invariant all 3 address statements not previously marked such that all operands are constant or all operand reaching definitions are outside the loop or 1 reaching definition in loop is marked invariant already. REPEAT.**
- **Create loop preheader node (immediate predecessor of header) as destination for moved code.**
- **Q: why only 1 reaching invariant definition?**

Example of Invariant Code Motion

```
1.   m = 3
2.   i = 1
3.   n = 2

4.   if i > 10 goto 10.

5.   k = m + 2
6.   j = 3 * k
7.  n = j + n
8.   i = i + 1
9.   goto 4

10. print j, k
```

Invariant Code Motion



```
1.   m = 3
2.   i = 1
3.   n = 2
```

```
4.   if i > 10 goto 10.
```

```
5.   k = m + 2
6.   j = 3 * k
7.  n = j + n
8.   i = i + 1
9.   goto 4
```

```
10. print j, k
```

Control Flow Graph

Determine Def-Use Links

- **Definitions:**

  {< m, B1 >, < i, B1 >, <n,B1 >, < k, B3 >, < j, B3 >, < n, B3 >, < i, B3 >}

- **Reaching Definitions**

  REACH (B1) = empty

  REACH (B2) = REACH (B3) = REACH (B4) =

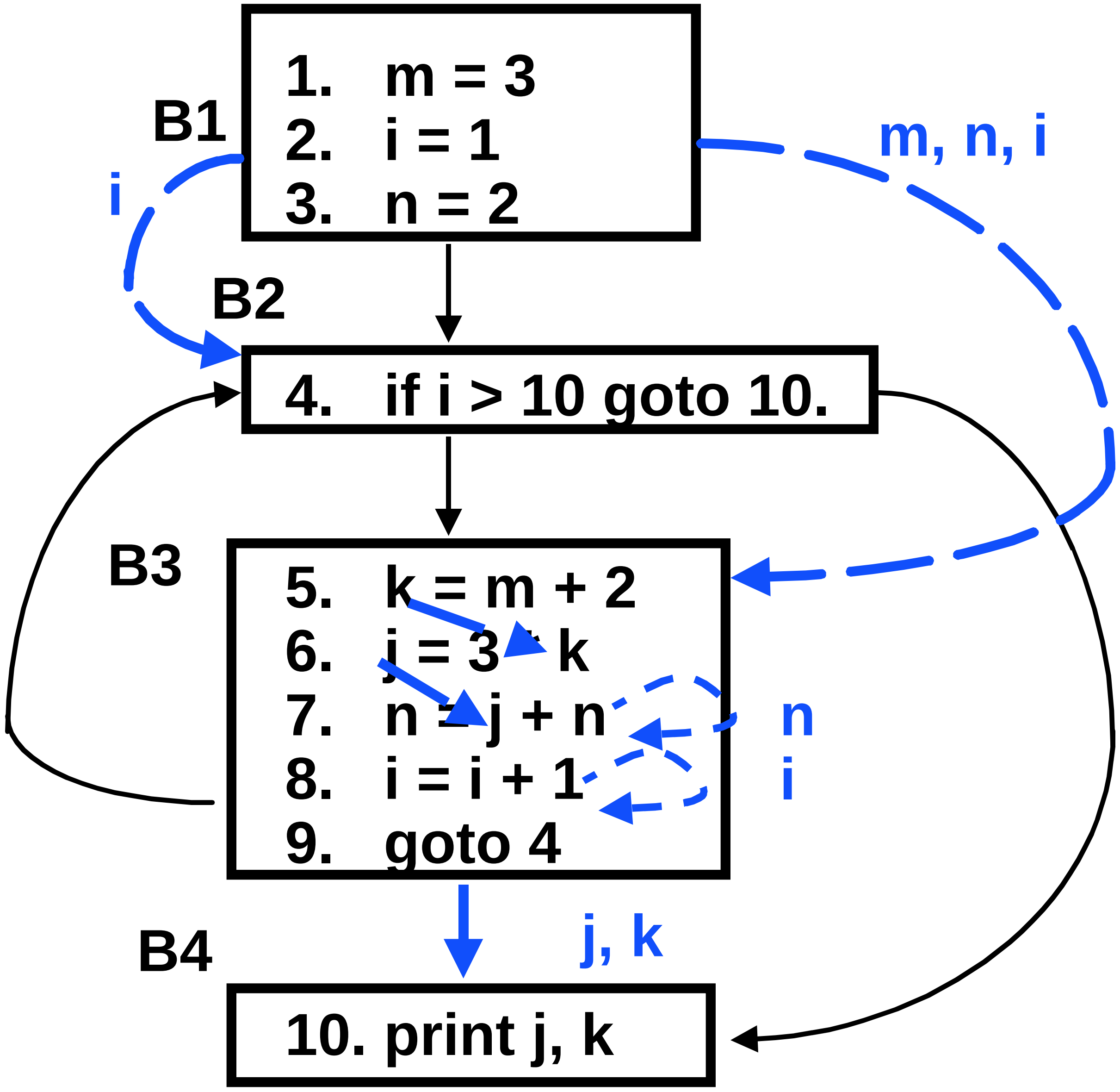
  all defs

  because the loop can iterate 0 or more than 0 times and therefore all defs reach node B4.

- **Def-Use Chains**

  At a use, link to all reaching definitions.

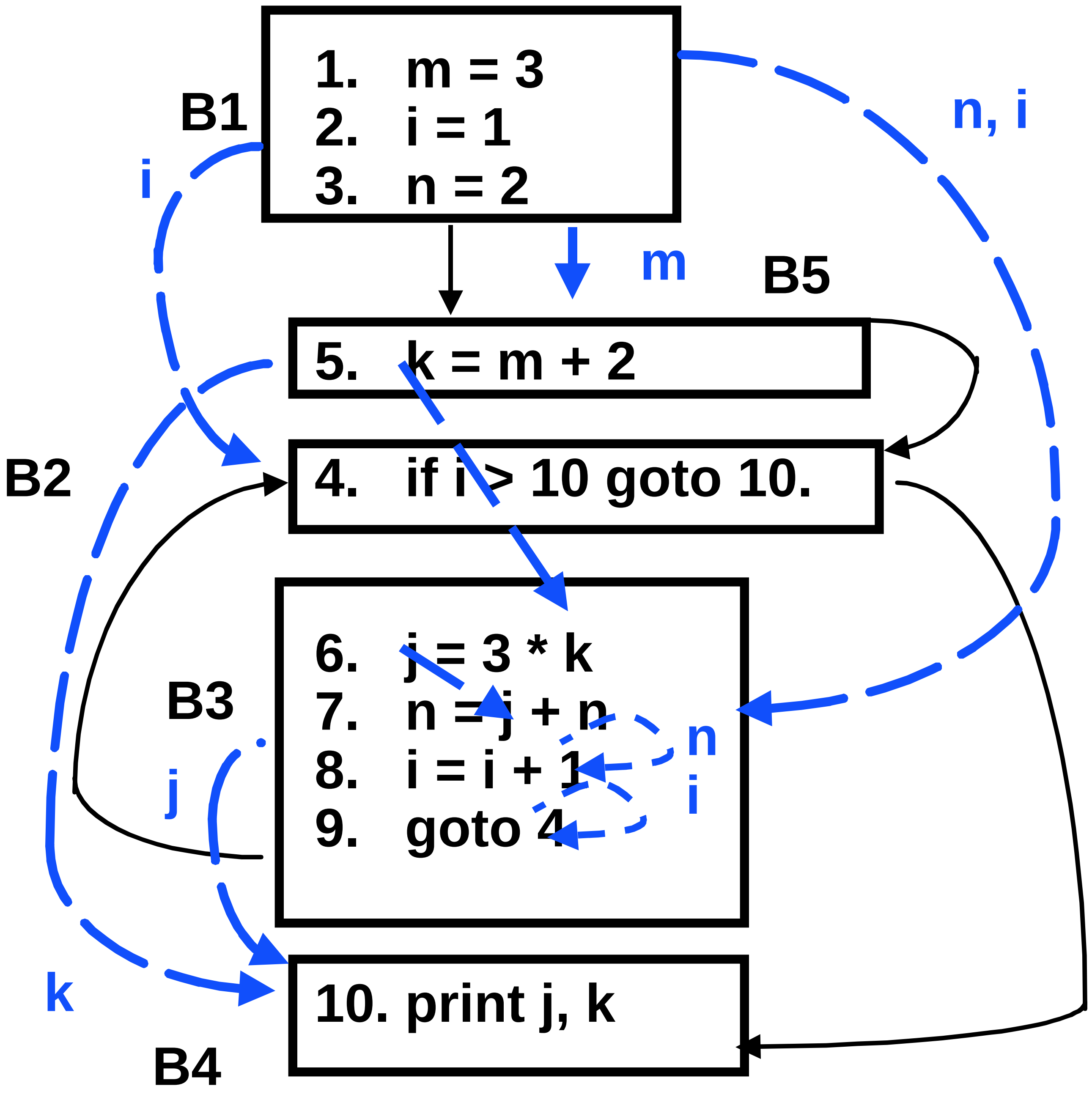
Def-Use Links



```
1.  m = 3
2.  i = 1
3.  n = 2

4.  if i > 10 goto 10.

5.  k = m + 2
6.  j = 3 * k
7.  n = j + n
8.  i = i + 1
9.  goto 4

10. print j, k
```

Def-Use Links - invariant code is code which has no def-use links from within the loop (B2,B3)

Invariant Code Motion



B1

1. m = 3
2. i = 1
3. n = 2

B5

5. k = m + 2

B2

4. if i > 10 goto 10.

B3

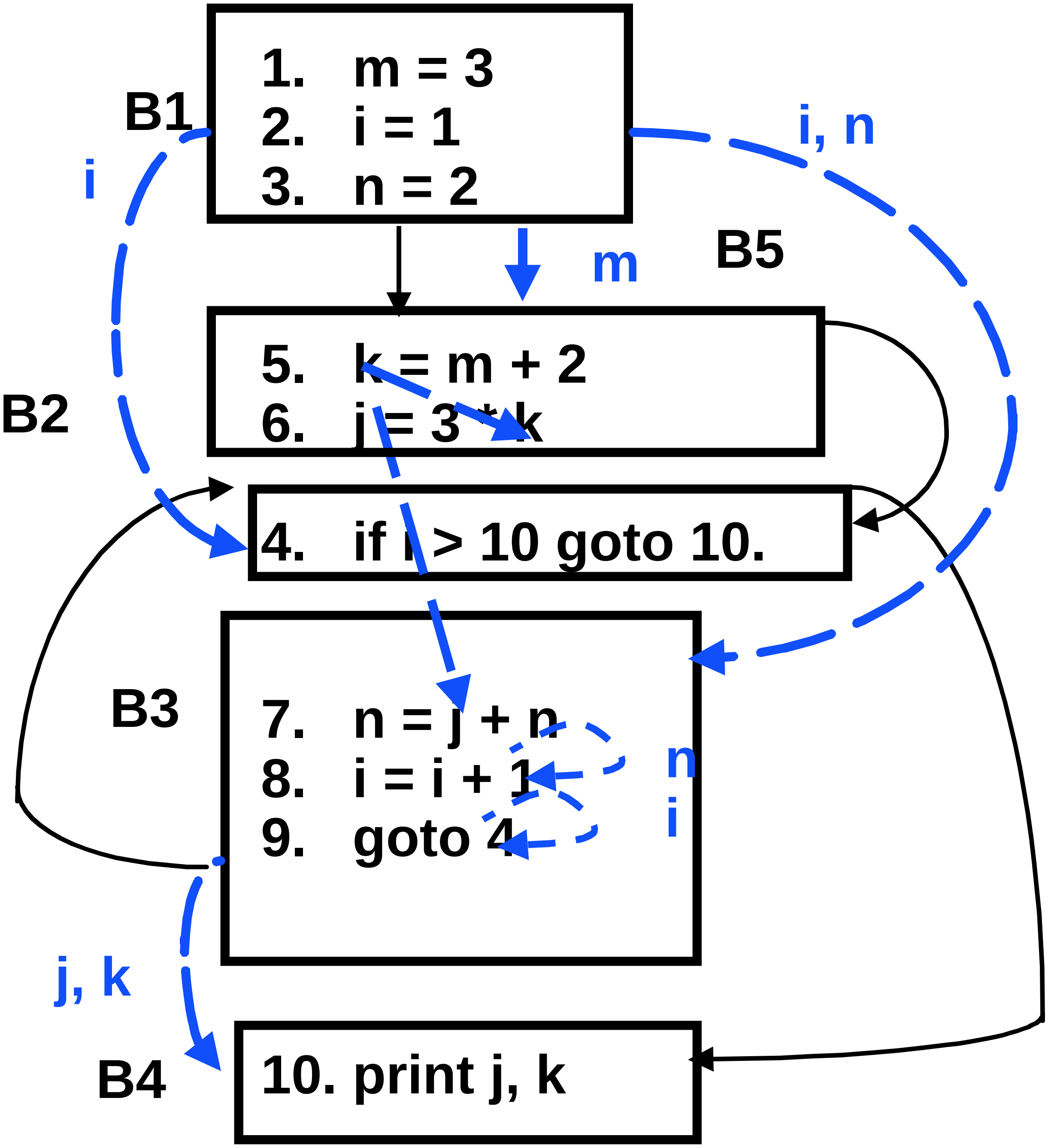
6. j = 3 * k
7. n = j + n
8. i = i + 1
9. goto 4

B4

10. print j, k

m+2 is invariant so statement 5 can be moved to new cfg node (loop preheader) B5.
now statement 6 can be moved to B5 as well, because it is now invariant in the loop.

Invariant Code Motion



No more code motion is possible because neither n nor i are invariant in the loop.