ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

# Unit12
# Code Optimization

# Introduction

- Criteria for Code-Improving Transformation:
  - Meaning must be preserved (correctness)
  - Speedup must occur on average.
  - Work done must be worth the effort.

- Opportunities:
  - Programmer (algorithm, directives)
  - Intermediate code
  - Target code

# Peephole Optimizations

1. A Simple but effective technique for locally improving the code is peephole optimization,
2. a method for trying to improve the performance of the program
3. by examining a short sequence of instructions and replacing these instructions by a shorter or faster sequence whenever possible.

Characteristics of peephole optimization

1. Redundant instruction elimination
2. Flow of control information
3. Algebraic Simplification
4. Use of machine Idioms

# Peephole Optimizations

- **Constant Folding**

  `x := 32` becomes `x := 64`

  `x := x + 32`

- **Unreachable Code**

  `goto L2`

  `x := x + 1` ← No need

- **Flow of control optimizations**

  `goto L1` becomes `goto L2`

  ...

  `L1: goto L2` ← No needed if no other L1 branch

# Peephole Optimizations

- **Algebraic Simplification**

  `x := x + 0` ← No needed

- **Dead code**

  `x := 32` ← where x not used after statement

  `y := x + y` → `y := y + 32`

- **Reduction in strength**

  `x := x * 2` → `x := x + x`

  → x := x << 1

# Basic Block Level

1. Common subexpression elimination
2. Constant Propagation
3. Copy Propagation
4. Dead code elimination
5. …

# Flow Graphs

- A *flow graph* is a graphical depiction of a sequence of instructions with control flow edges
- A flow graph can be defined at the intermediate code level or target code level

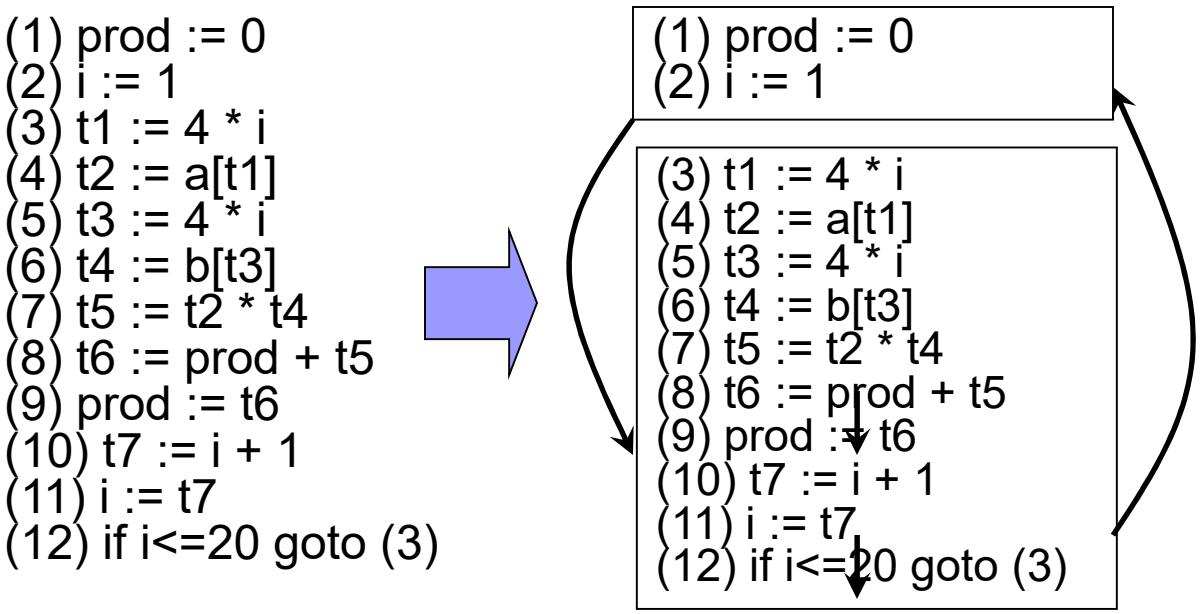
# Basic Blocks

- A *basic block* is a sequence of consecutive instructions with exactly one entry point and one exit point (with natural flow or a branch instruction)

Example
```
t1 := a * a
t2 := a * b
t3 := 2 * t2
t4 := t1 + t2
t5 := b * b
t6 := t4 + t5
```

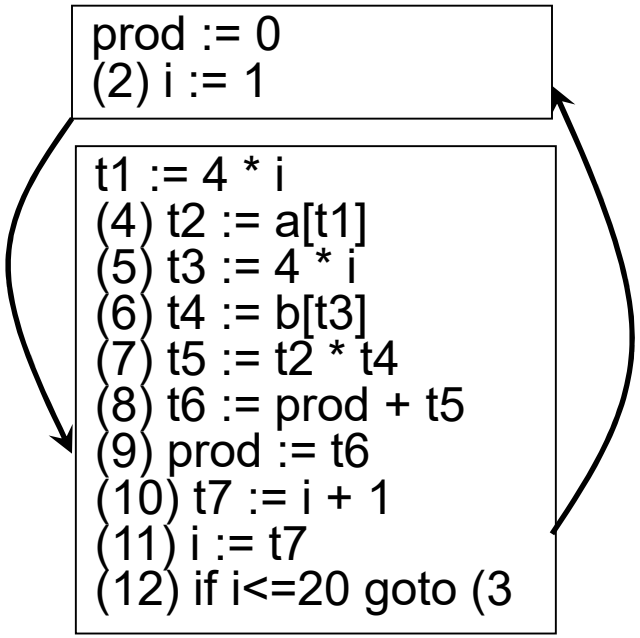
# Basic Blocks and Control Flow Graphs

- A *control flow graph* (CFG) is a directed graph with basic blocks $B_i$ as vertices and with edges $B_i \rightarrow B_j$ iff $B_j$ can be executed immediately after $B_i$



# Successor and Predecessor Blocks

- Suppose the CFG has an edge $B_1 \rightarrow B_2$
  - Basic block $B_1$ is a *predecessor* of $B_2$
  - Basic block $B_2$ is a *successor* of $B_1$



# Partition Algorithm for Basic Blocks

*Input*: A sequence of three-address statements
*Output*: A list of basic blocks with each three-address statement in exactly one block

1. Determine the set of *leaders*, the first statements if basic blocks
   a) The first statement is the leader
   b) Any statement that is the target of a goto is a leader
   c) Any statement that immediately follows a goto is a leader
2. For each leader, its basic block consist of the leader and all statements up to but not including the next leader or the end of the program

# Common expression can be eliminated

Simple example: a[i+1] = b[i+1]

- t1 = i+1
- t2 = b[t1]
- t3 = i + 1
- a[t3] = t2

- t1 = i + 1
- t2 = b[t1]
- t3 = i + 1 ←
  *no longer live*
- a[t1] = t2

# Constant propagation

## Now, suppose i is a constant:

```
i = 4
t1 = i+1
t2 = b[t1]
a[t1] = t2
```

```
i = 4
t1 = 5
t2 = b[t1]
a[t1] = t2
```

```
i = 4
t1 = 5
t2 = b[5]
a[5] = t2
```

Final Code:

```
i = 4
t2 = b[5]
a[5] = t2
```

# Optimizations on CFG

- Must take control flow into account
  - Common Sub-expression Elimination
  - Constant Propagation
  - Dead Code Elimination
  - Partial redundancy Elimination
  - …
- Applying one optimization may raise opportunities for other optimizations.

# Three Address Code of Quick Sort

| | |
|---|---|
| 1 | i = m - 1 |
| 2 | j = n |
| 3 | $t_1$ =4 * n |
| 4 | v = a[$t_1$] |
| 5 | i = i + 1 |
| 6 | $t_2$ = 4 * i |
| 7 | $t_3$ = a[$t_2$] |
| 8 | if $t_3$ < v goto (5) |
| 9 | j = j – 1 |
| 10 | $t_4$ = 4 * j |
| 11 | $t_5$ = a[$t_4$] |
| 12 | if $t_5$ > v goto (9) |
| 13 | if i >= j goto (23) |
| 14 | $t_6$ = 4 * i |
| 15 | x = a[$t_6$] |

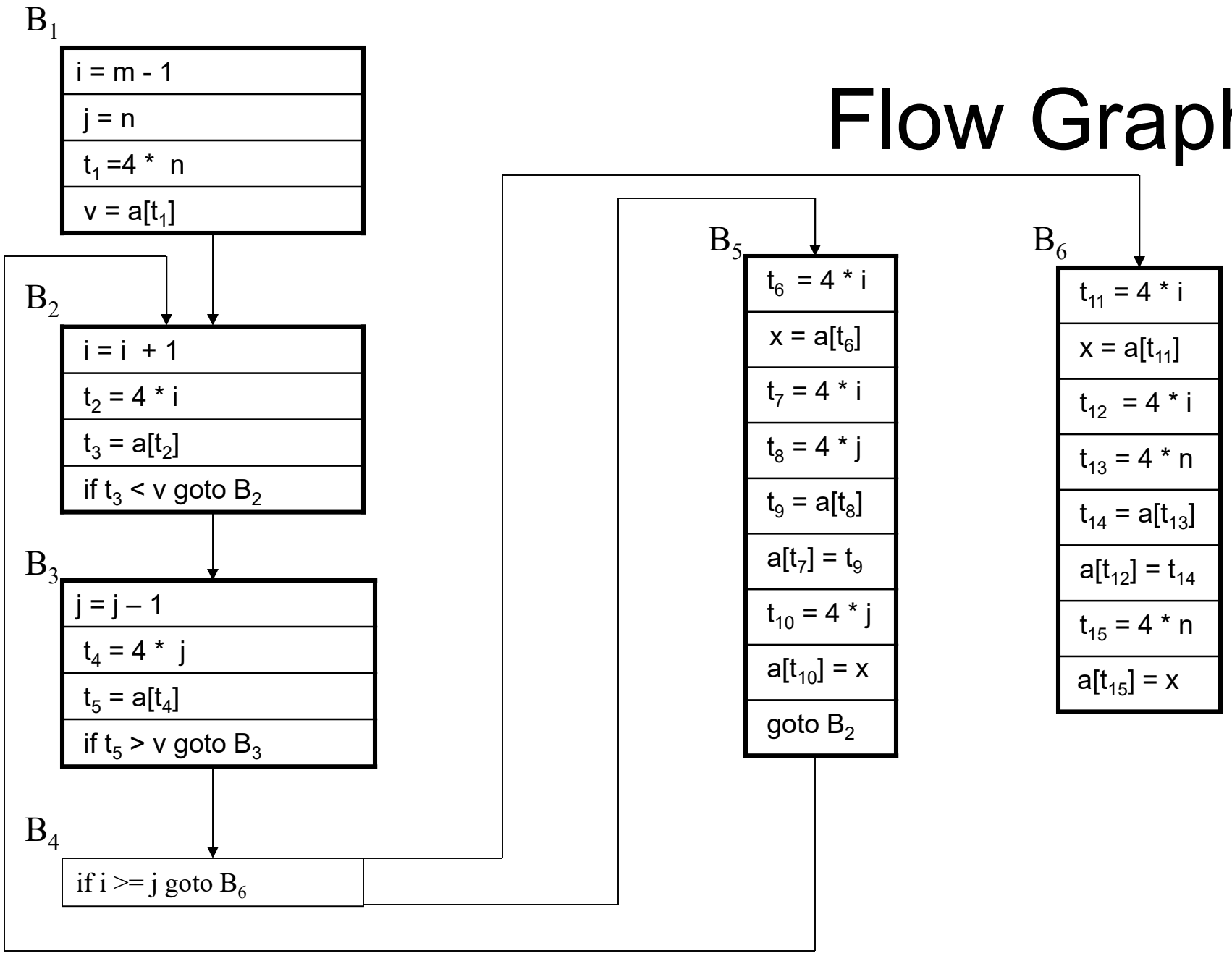
| | |
|---|---|
| 16 | $t_7$ = 4 * I |
| 17 | $t_8$ = 4 * j |
| 18 | $t_9$ = a[$t_8$] |
| 19 | a[$t_7$] = $t_9$ |
| 20 | $t_{10}$ = 4 * j |
| 21 | a[$t_{10}$] = x |
| 22 | goto (5) |
| 23 | $t_{11}$ = 4 * I |
| 24 | x = a[$t_{11}$] |
| 25 | $t_{12}$ = 4 * i |
| 26 | $t_{13}$ = 4 * n |
| 27 | $t_{14}$ = a[$t_{13}$] |
| 28 | a[$t_{12}$] = $t_{14}$ |
| 29 | $t_{15}$ = 4 * n |
| 30 | a[$t_{15}$] = x |

# Find The Basic Block

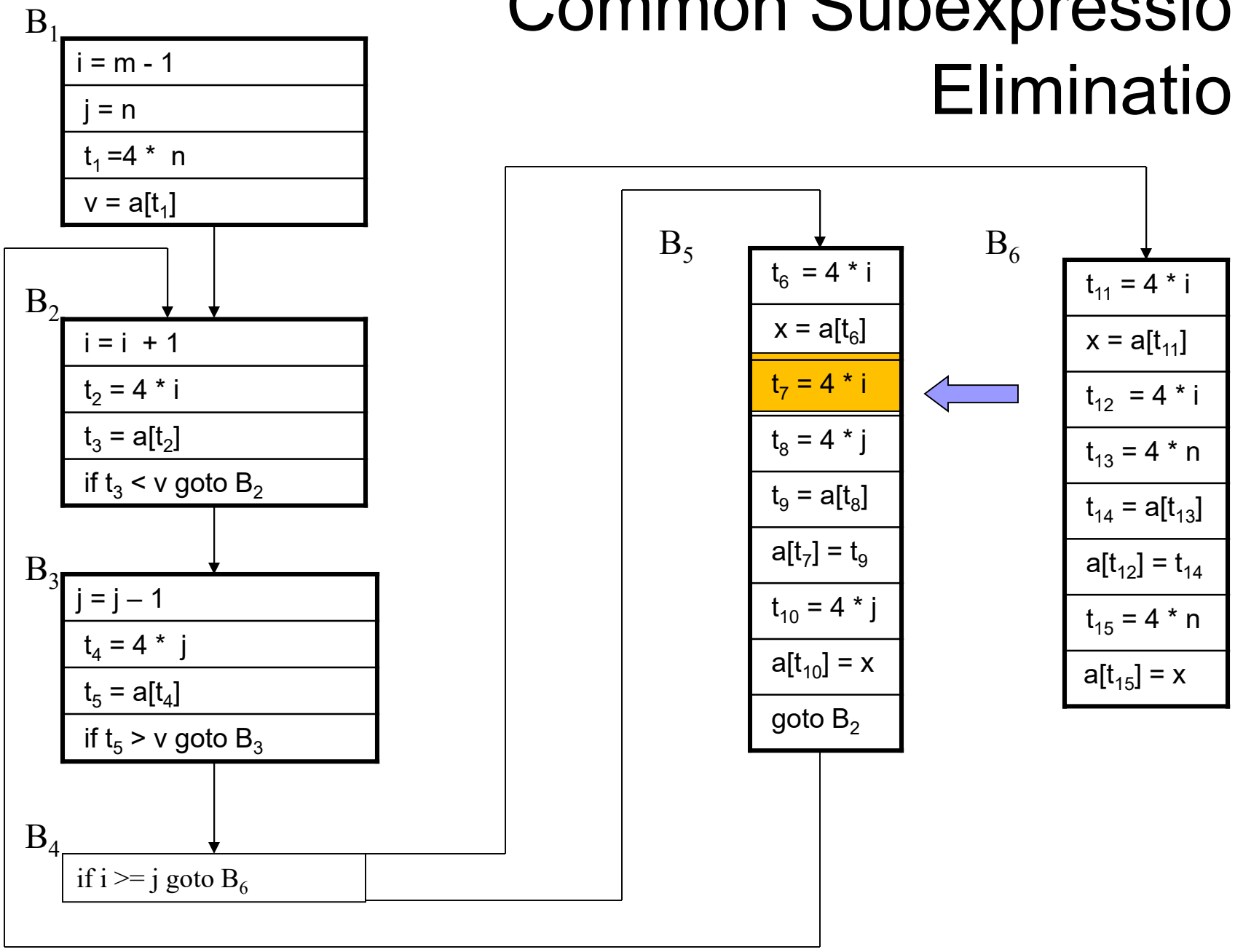
| | |
|---|---|
| 1 | i = m - 1 |
| 2 | j = n |
| 3 | $t_1$ =4 * n |
| 4 | v = a[$t_1$] |
| 5 | i = i + 1 |
| 6 | $t_2$ = 4 * i |
| 7 | $t_3$ = a[$t_2$] |
| 8 | if $t_3$ < v goto (5) |
| 9 | j = j – 1 |
| 10 | $t_4$ = 4 * j |
| 11 | $t_5$ = a[$t_4$] |
| 12 | if $t_5$ > v goto (9) |
| 13 | if i >= j goto (23) |
| 14 | $t_6$ = 4 * i |
| 15 | x = a[$t_6$] |

| | |
|---|---|
| 16 | $t_7$ = 4 * I |
| 17 | $t_8$ = 4 * j |
| 18 | $t_9$ = a[$t_8$] |
| 19 | a[$t_7$] = $t_9$ |
| 20 | $t_{10}$ = 4 * j |
| 21 | a[$t_{10}$] = x |
| 22 | goto (5) |
| 23 | $t_{11}$ = 4 * i |
| 24 | x = a[$t_{11}$] |
| 25 | $t_{12}$ = 4 * i |
| 26 | $t_{13}$ = 4 * n |
| 27 | $t_{14}$ = a[$t_{13}$] |
| 28 | a[$t_{12}$] = $t_{14}$ |
| 29 | $t_{15}$ = 4 * n |
| 30 | a[$t_{15}$] = x |

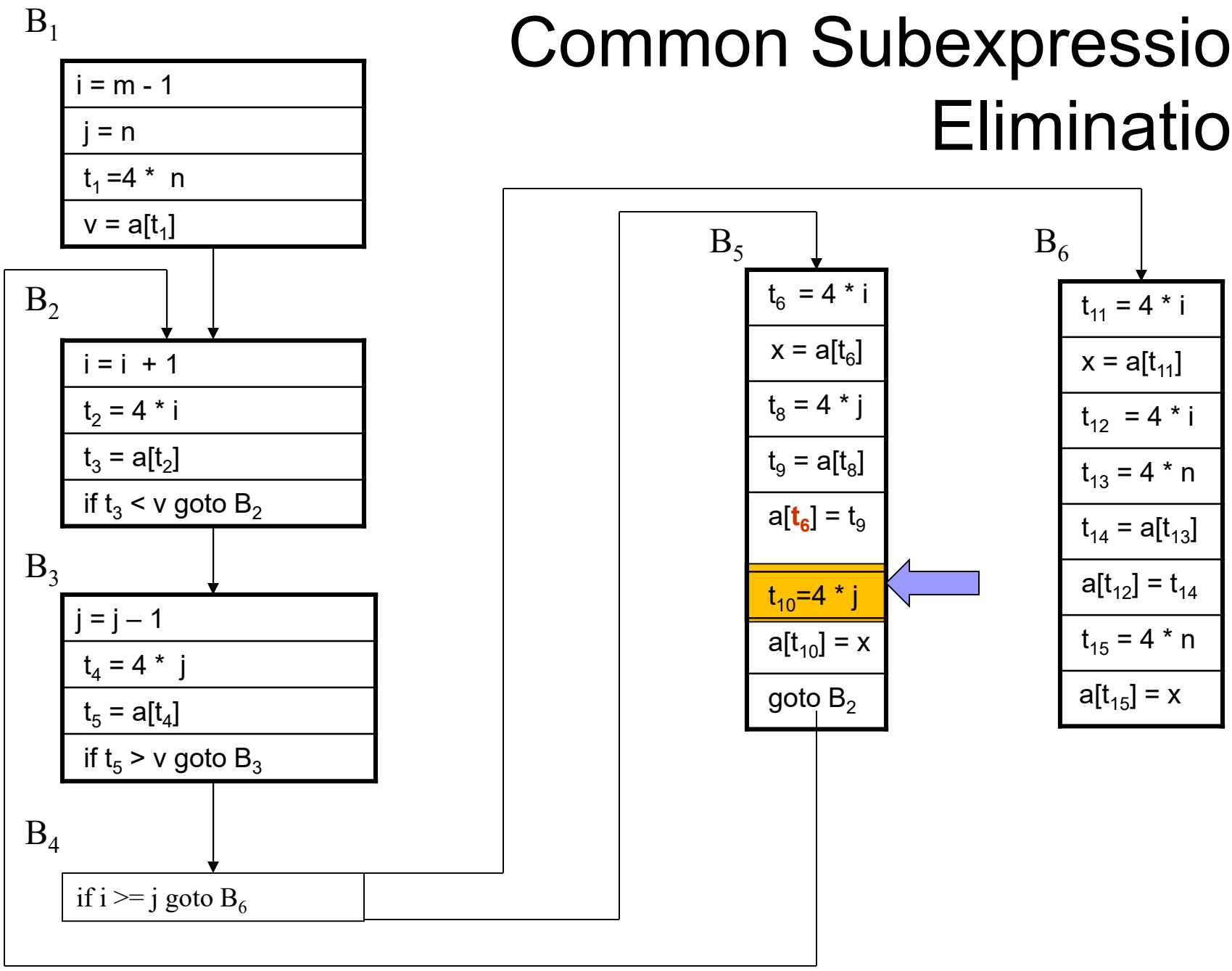
# Flow Graph


B1
i = m - 1
j = n
t1 =4 * n
v = a[t1]
B2
i = i + 1
t2 = 4 * i
t3 = a[t2]
if t3 < v goto B2
B3
j = j – 1
t4 = 4 * j
t5 = a[t4]
if t5 > v goto B3
B4
if i >= j goto B6
B5
t6 = 4 * i
x = a[t6]
t7 = 4 * i
t8 = 4 * j
t9 = a[t8]
a[t7] = t9
t10 = 4 * j
a[t10] = x
goto B2
B6
t11 = 4 * i
x = a[t11]
t12 = 4 * i
t13 = 4 * n
t14 = a[t13]
a[t12] = t14
t15 = 4 * n
a[t15] = x

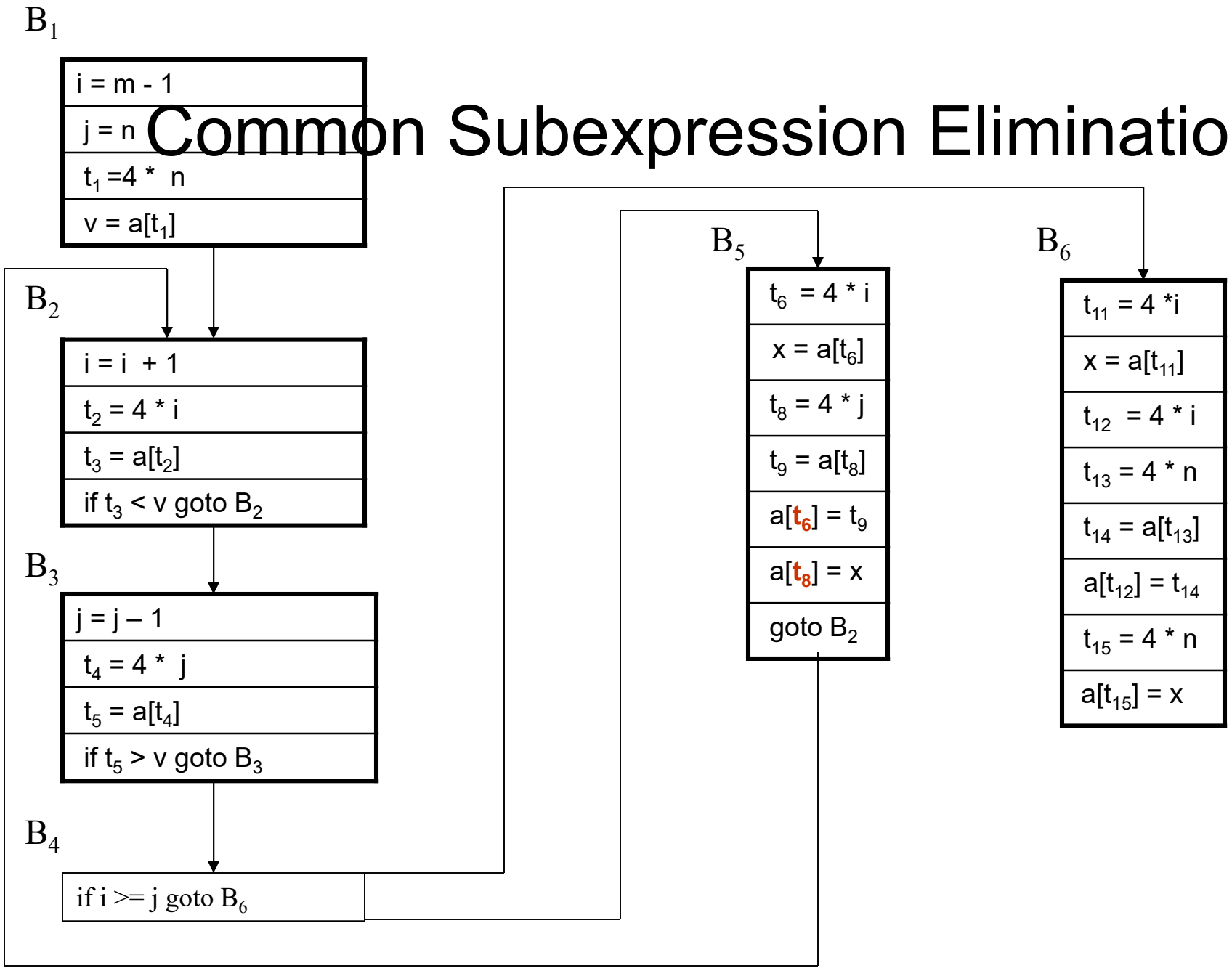
# Common Subexpression Elimination



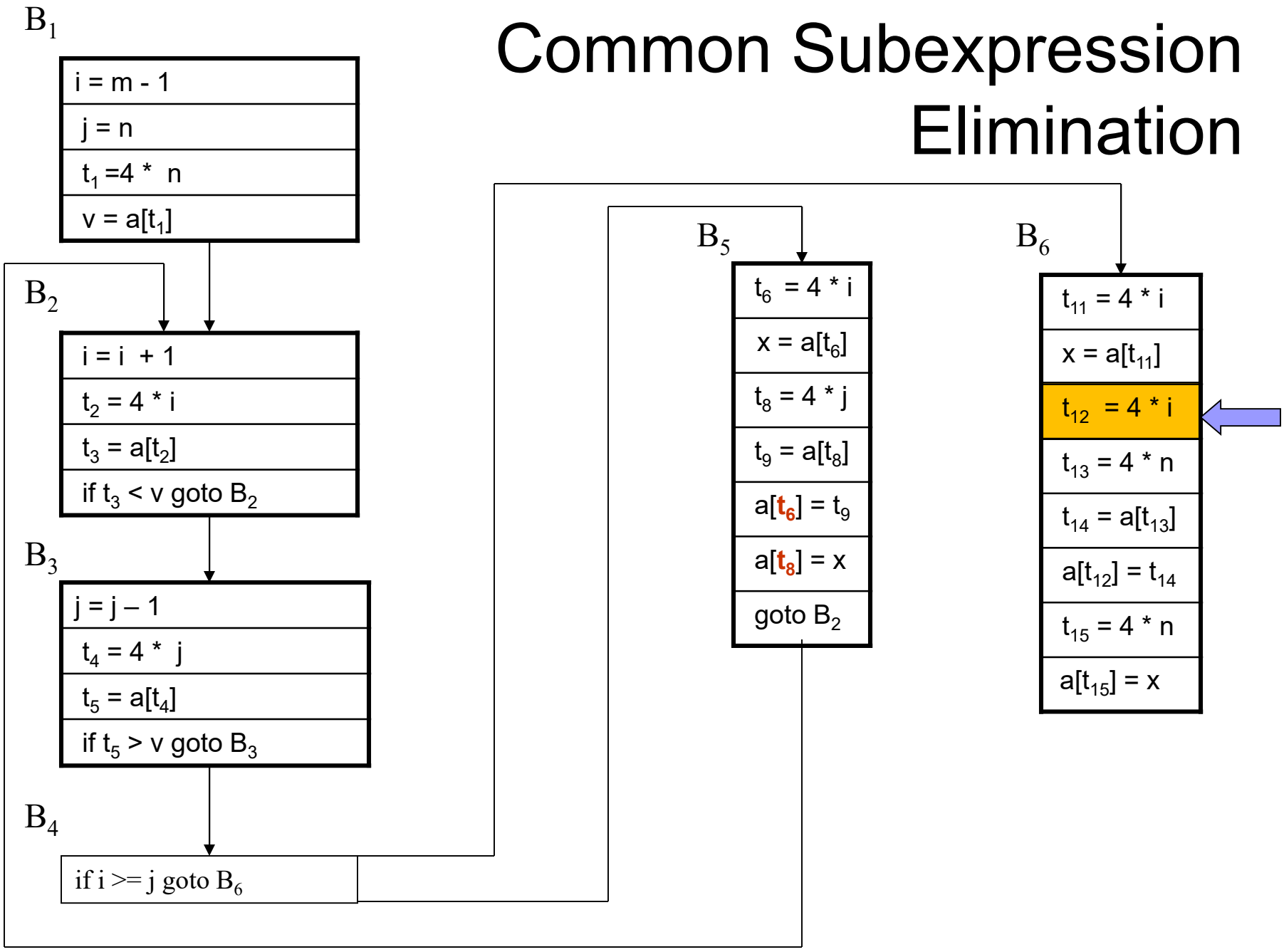
# Common Subexpression Elimination



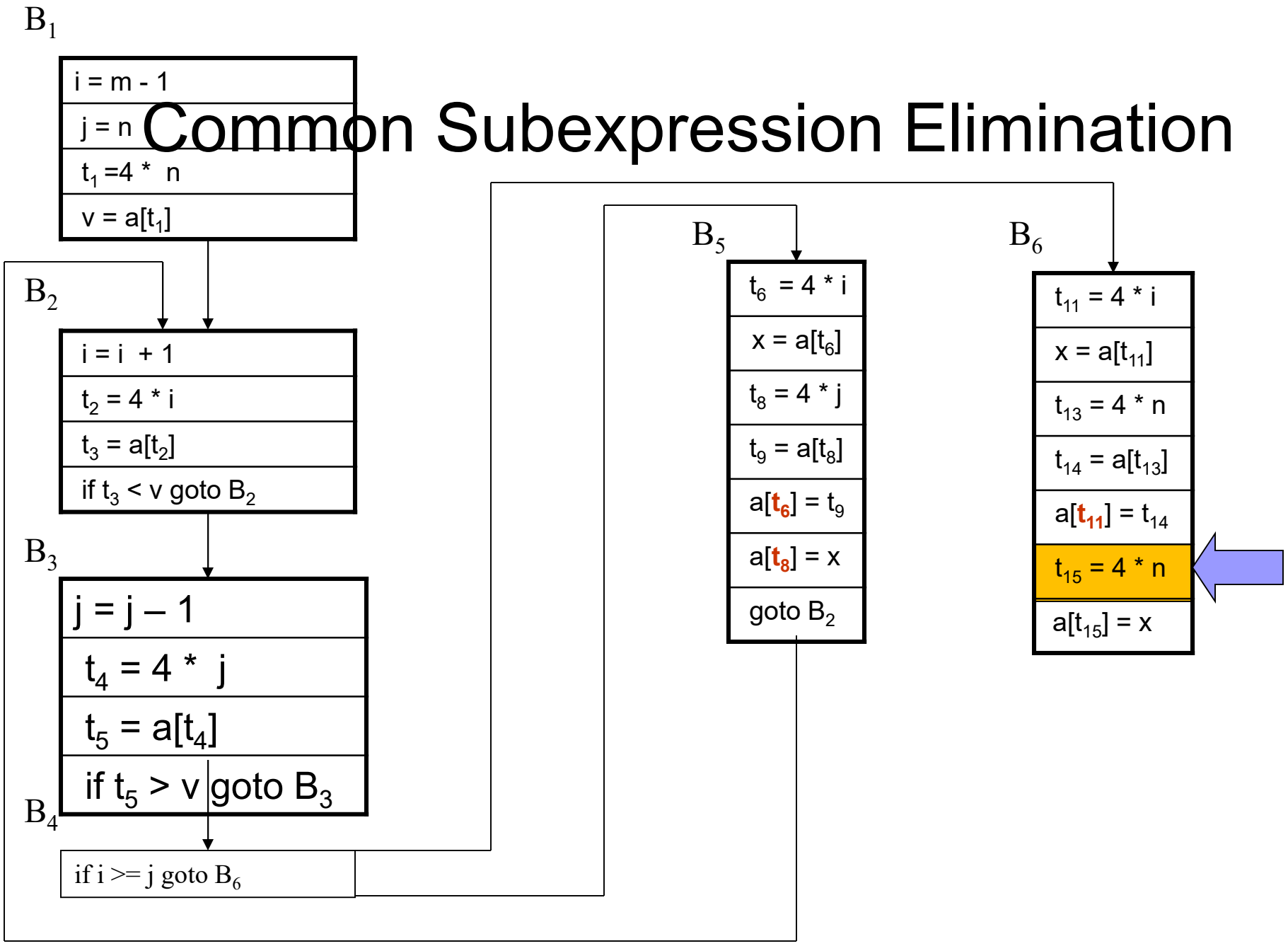
# Common Subexpression Elimination



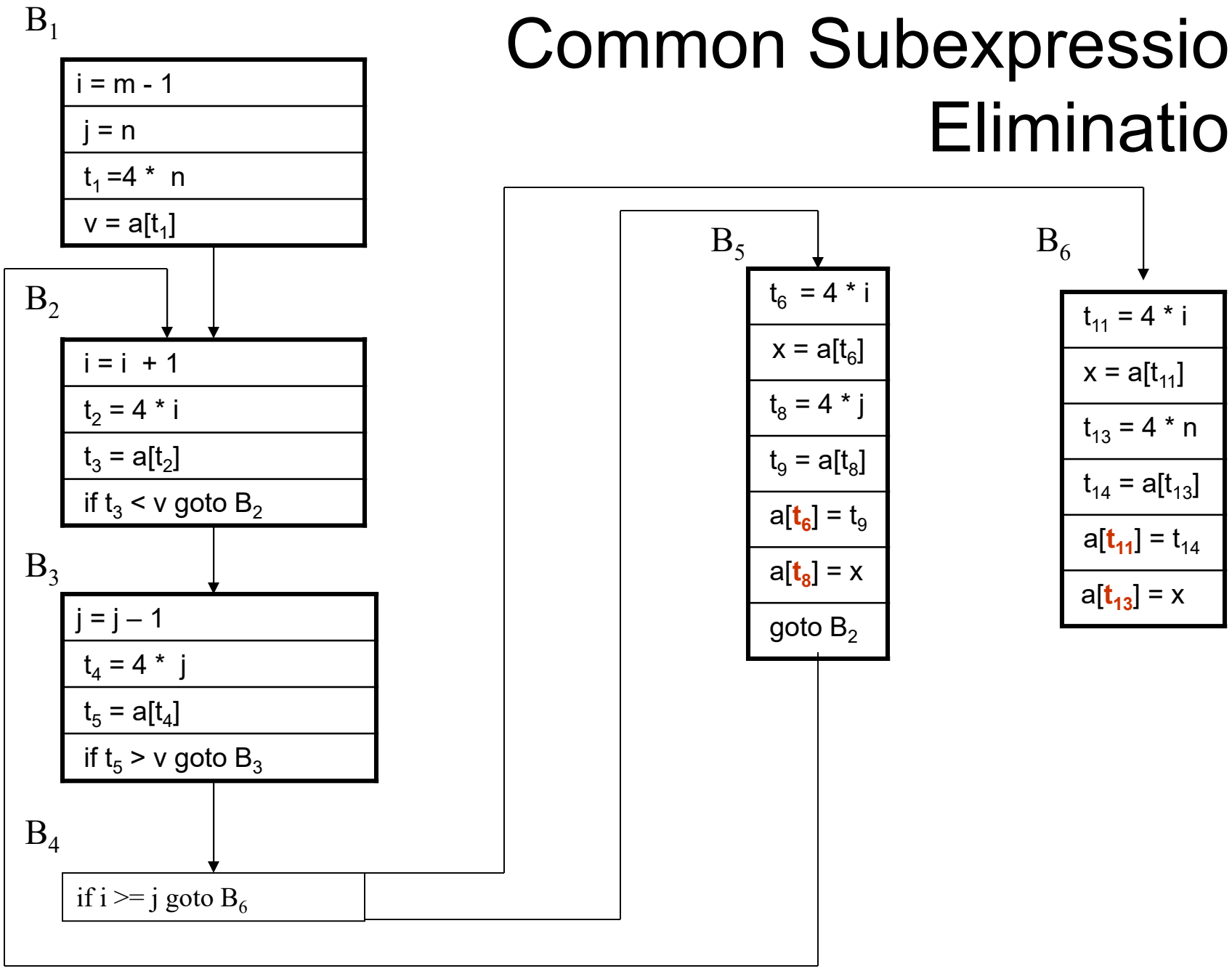
# Common Subexpression Elimination



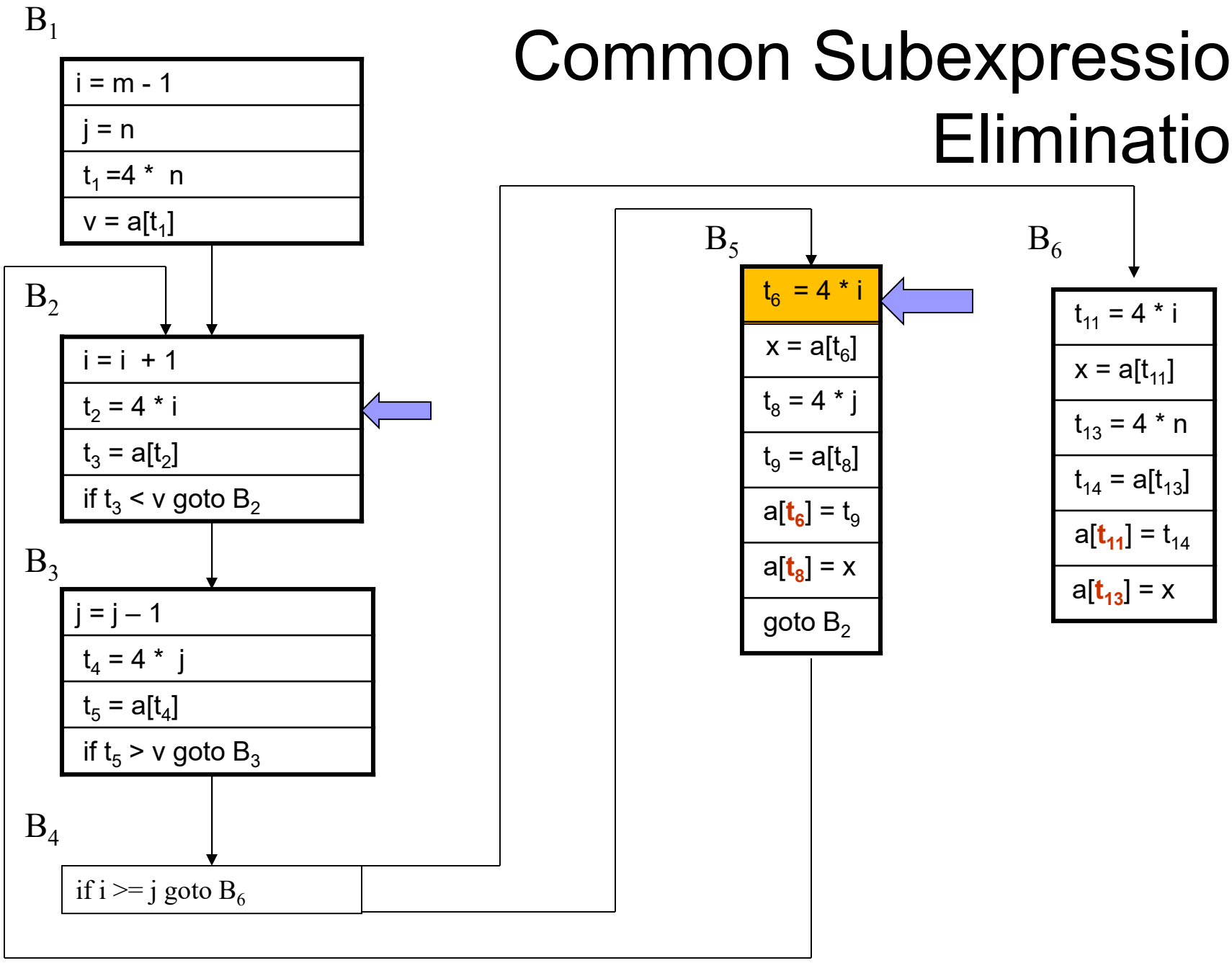
# Common Subexpression Elimination



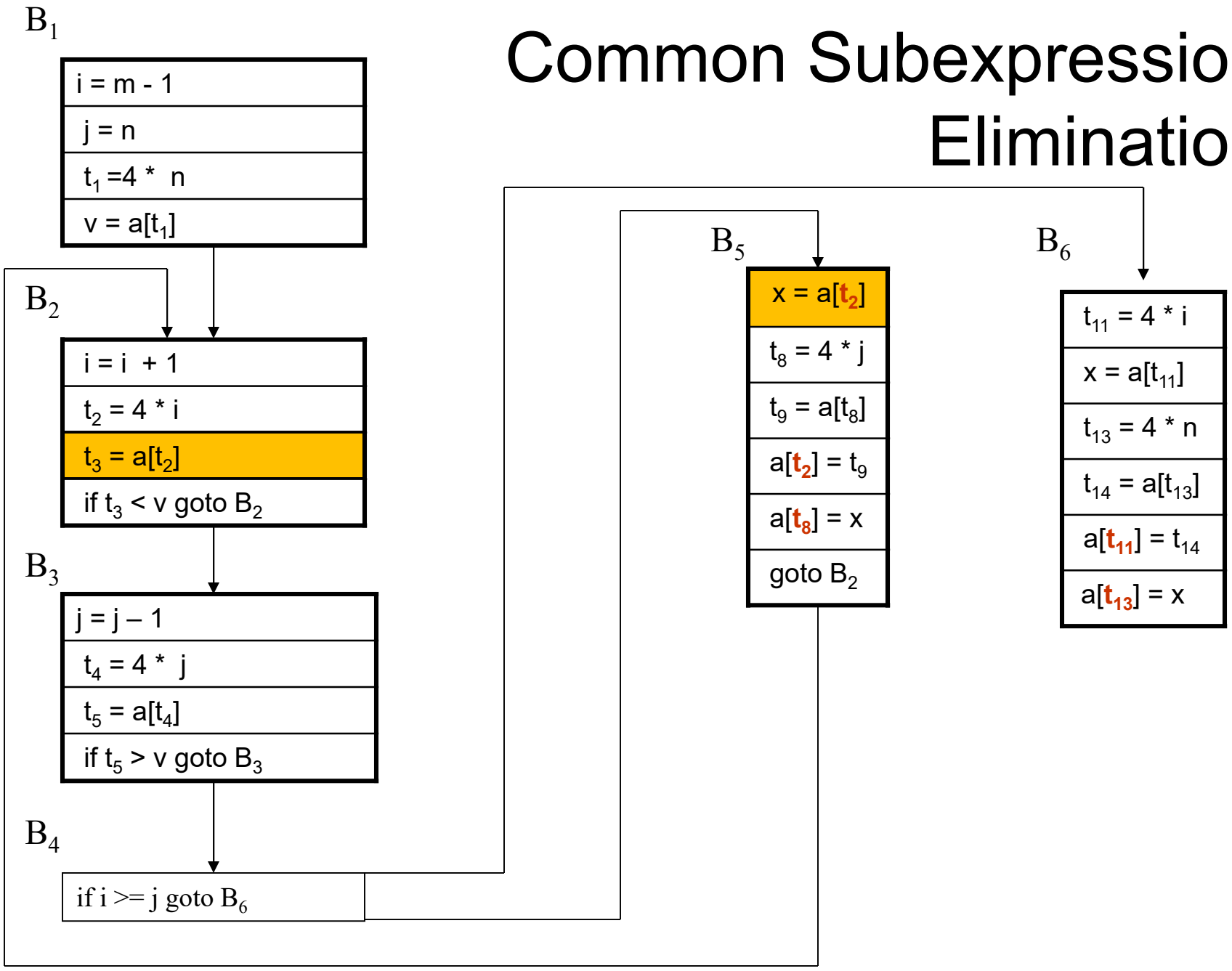
# Common Subexpression Elimination



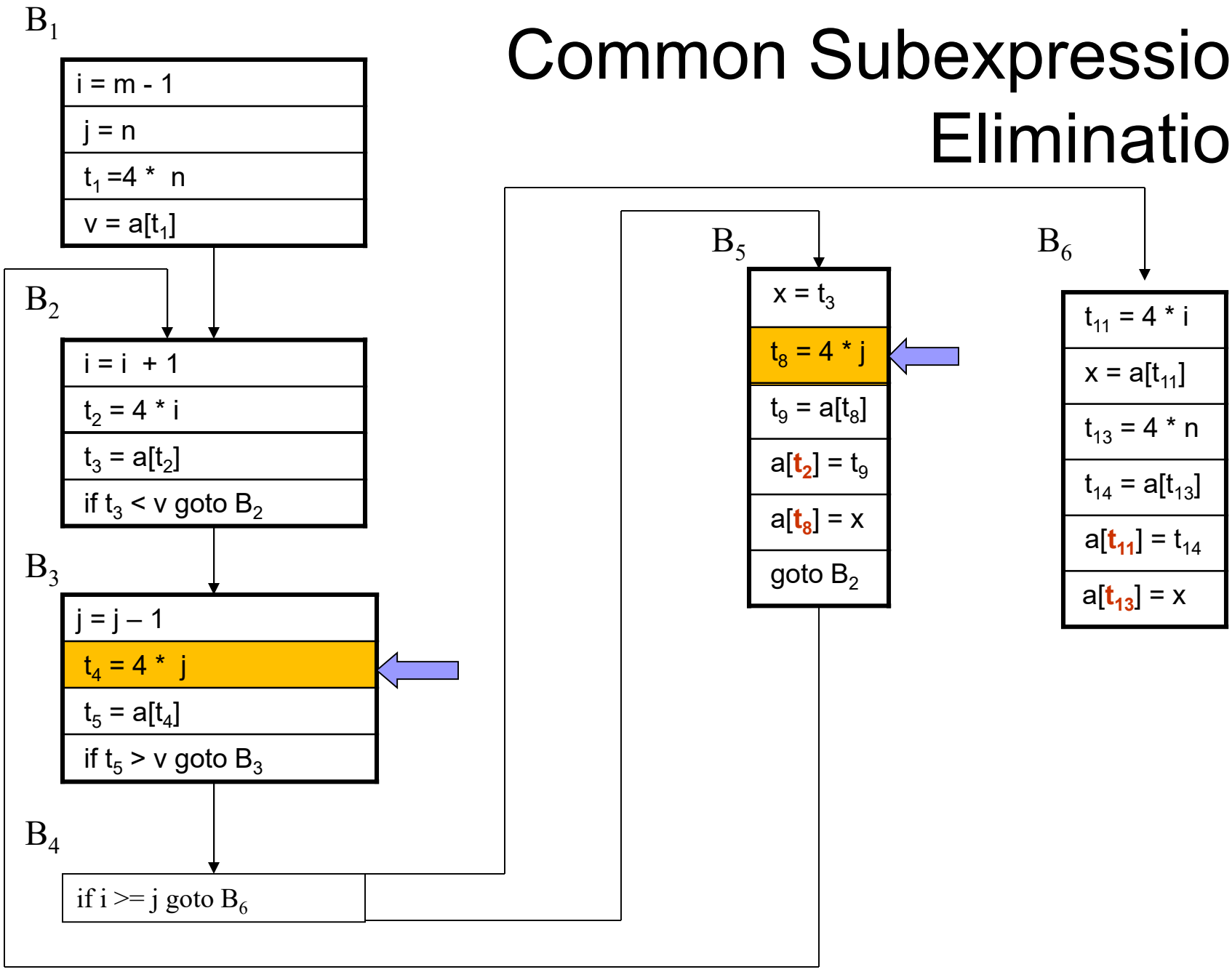
# Common Subexpression Elimination



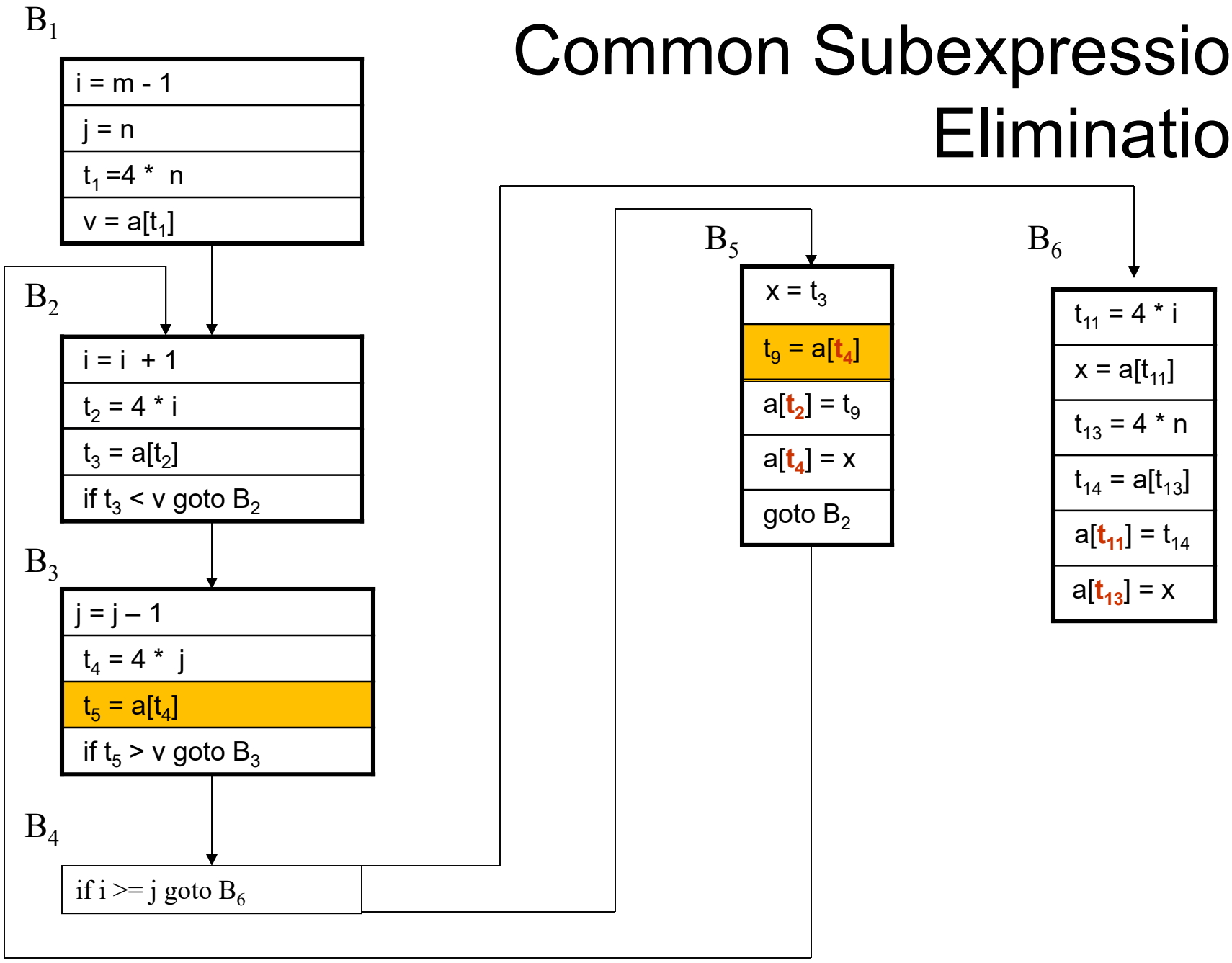
# Common Subexpression Elimination



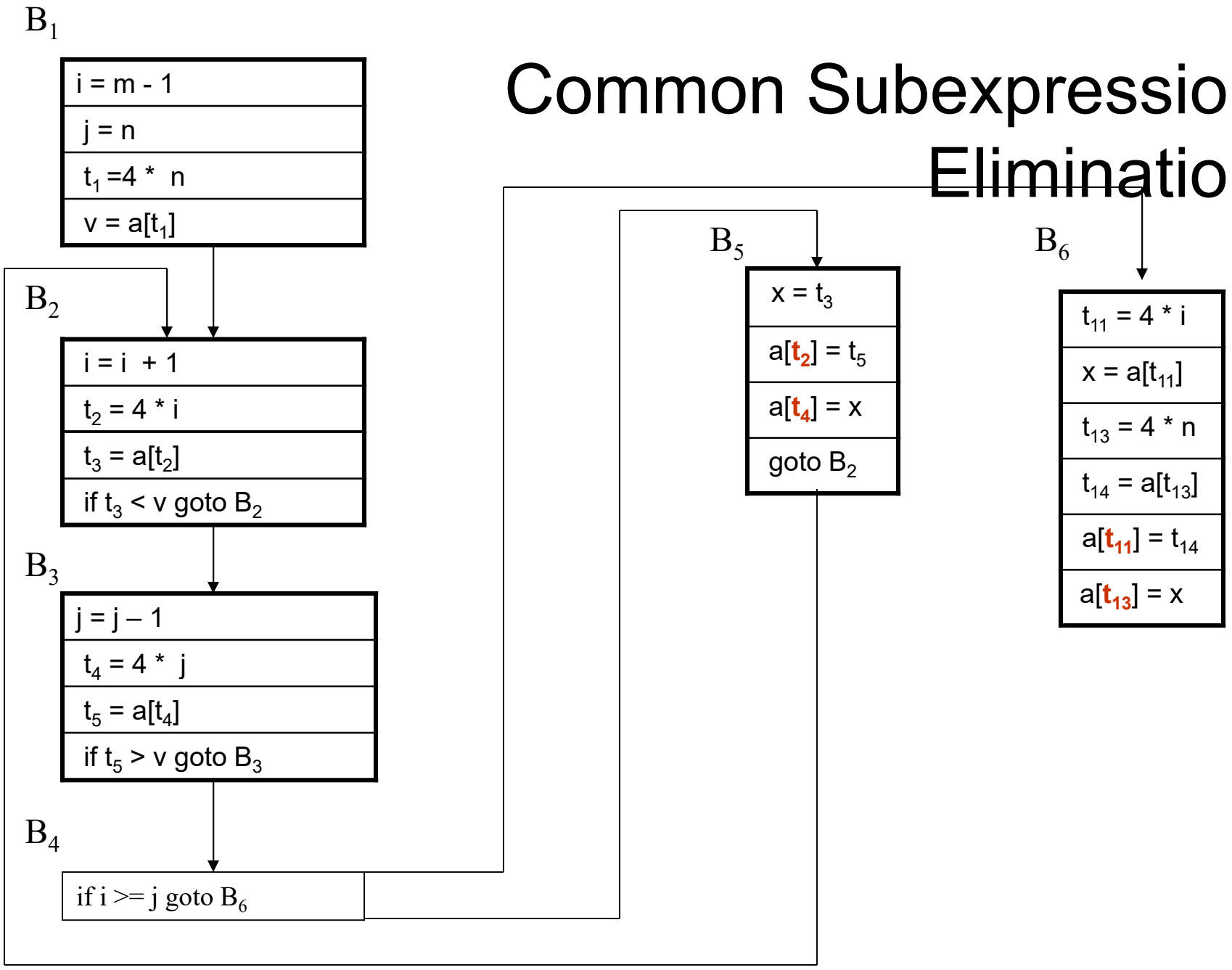
# Common Subexpression Elimination



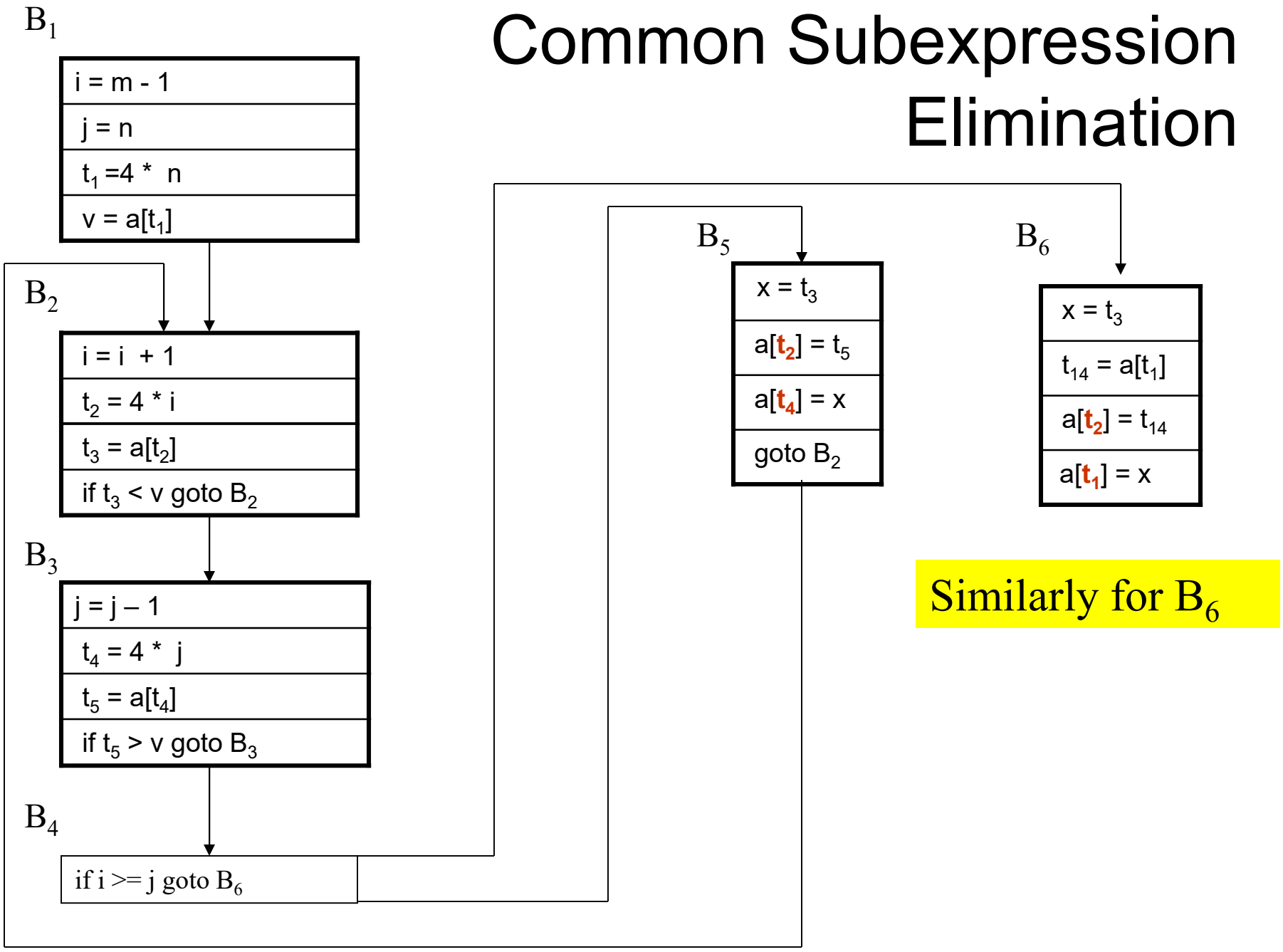
# Common Subexpression Elimination



# Common Subexpression Elimination

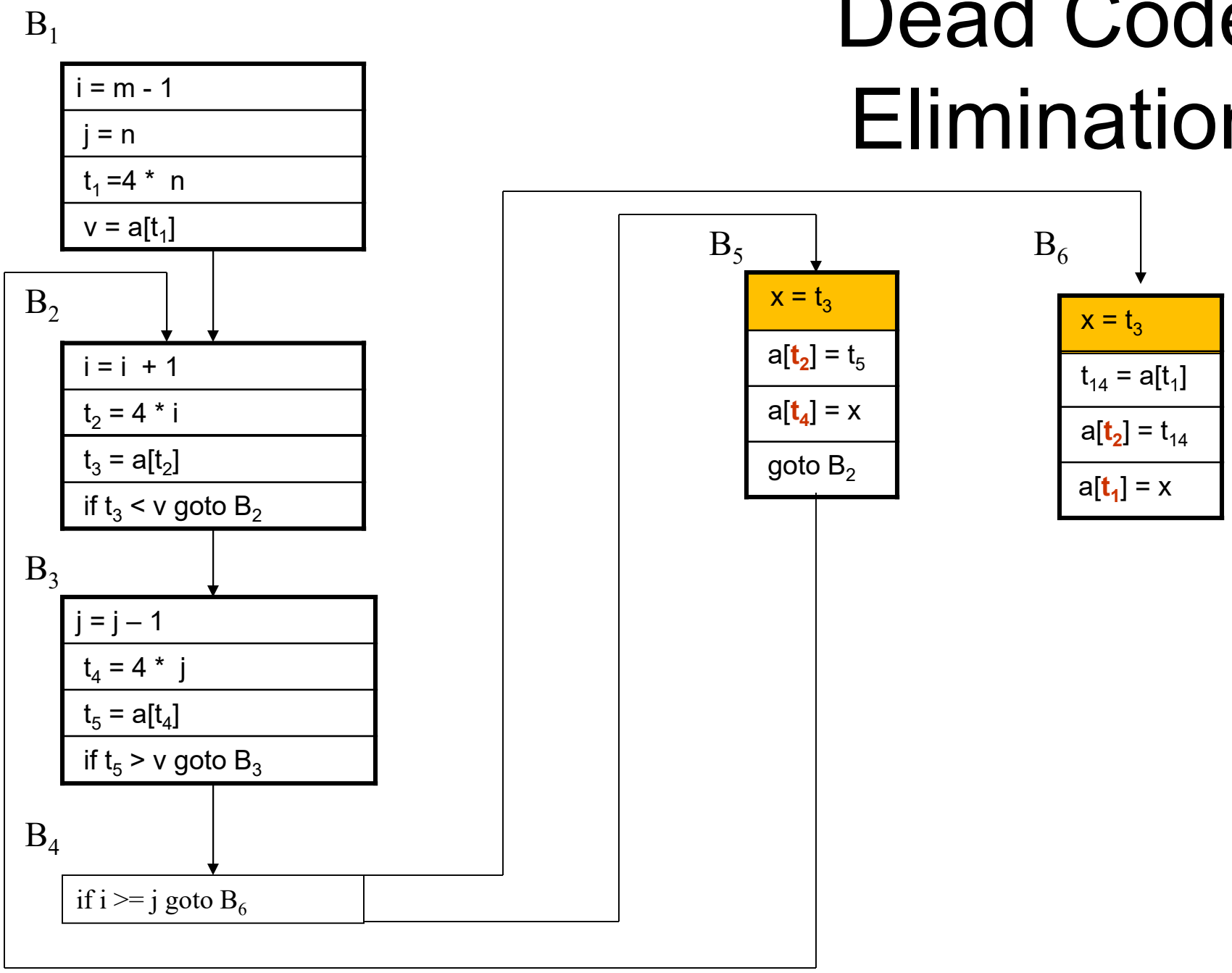


# Common Subexpression Elimination

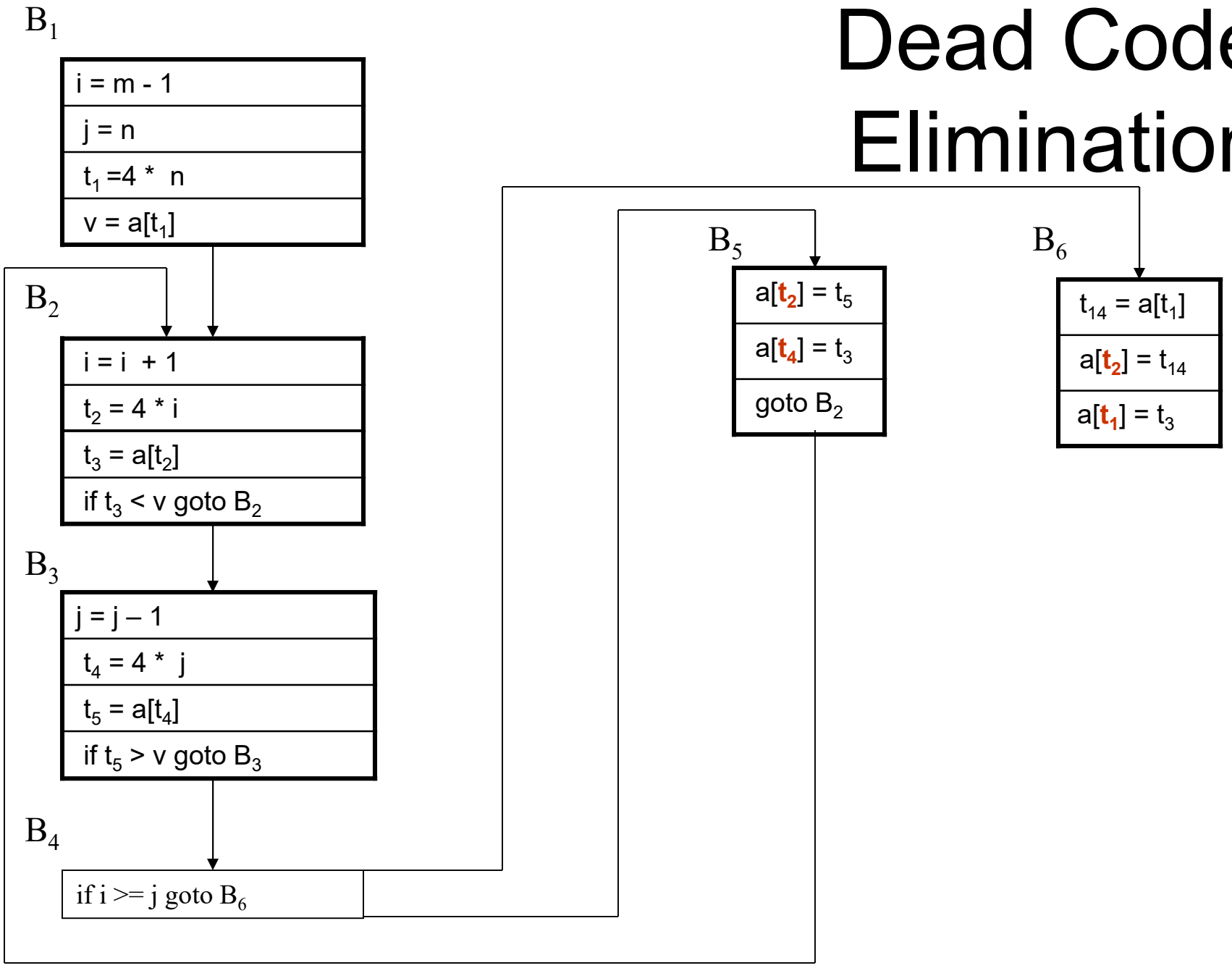



Similarly for $B_6$

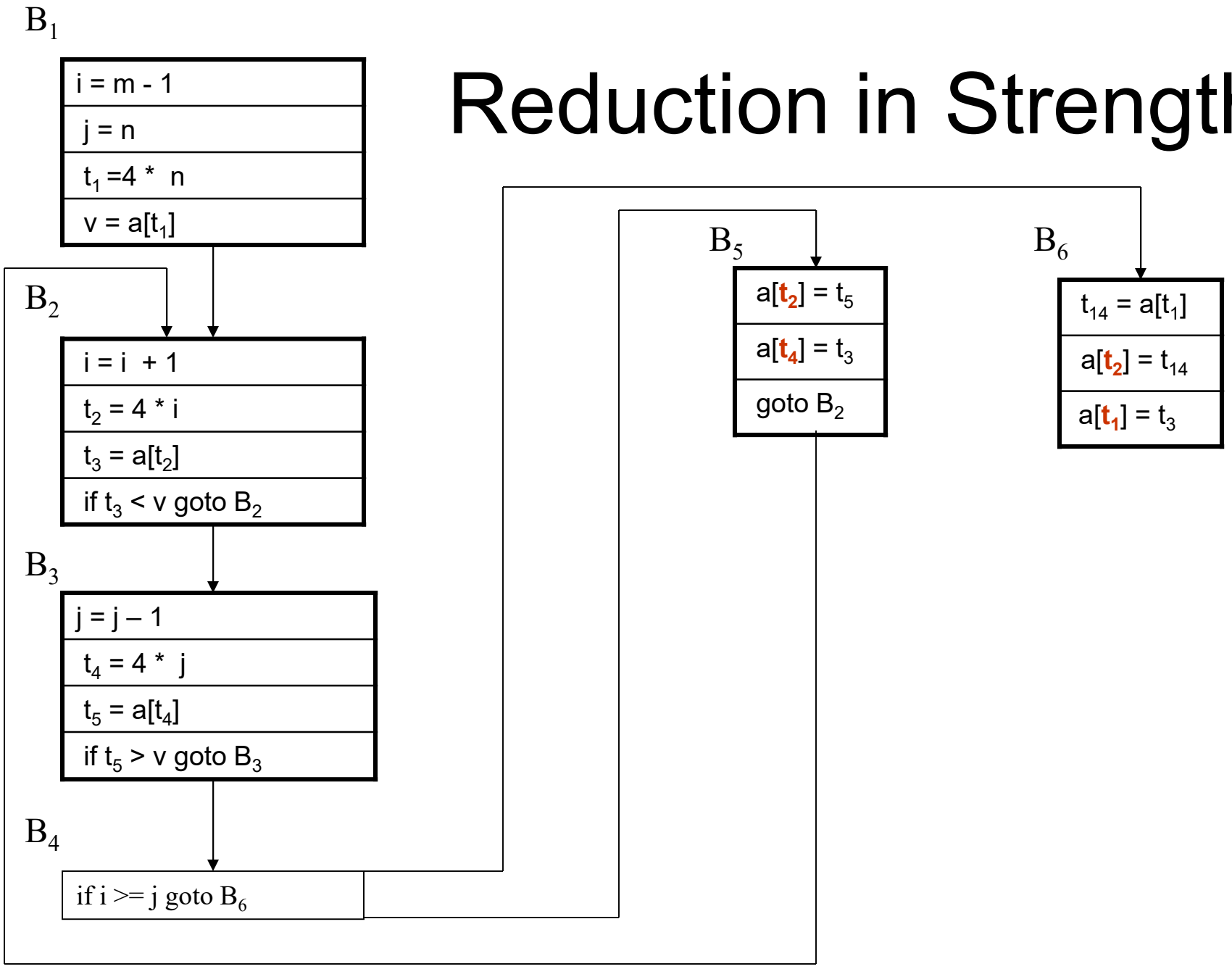
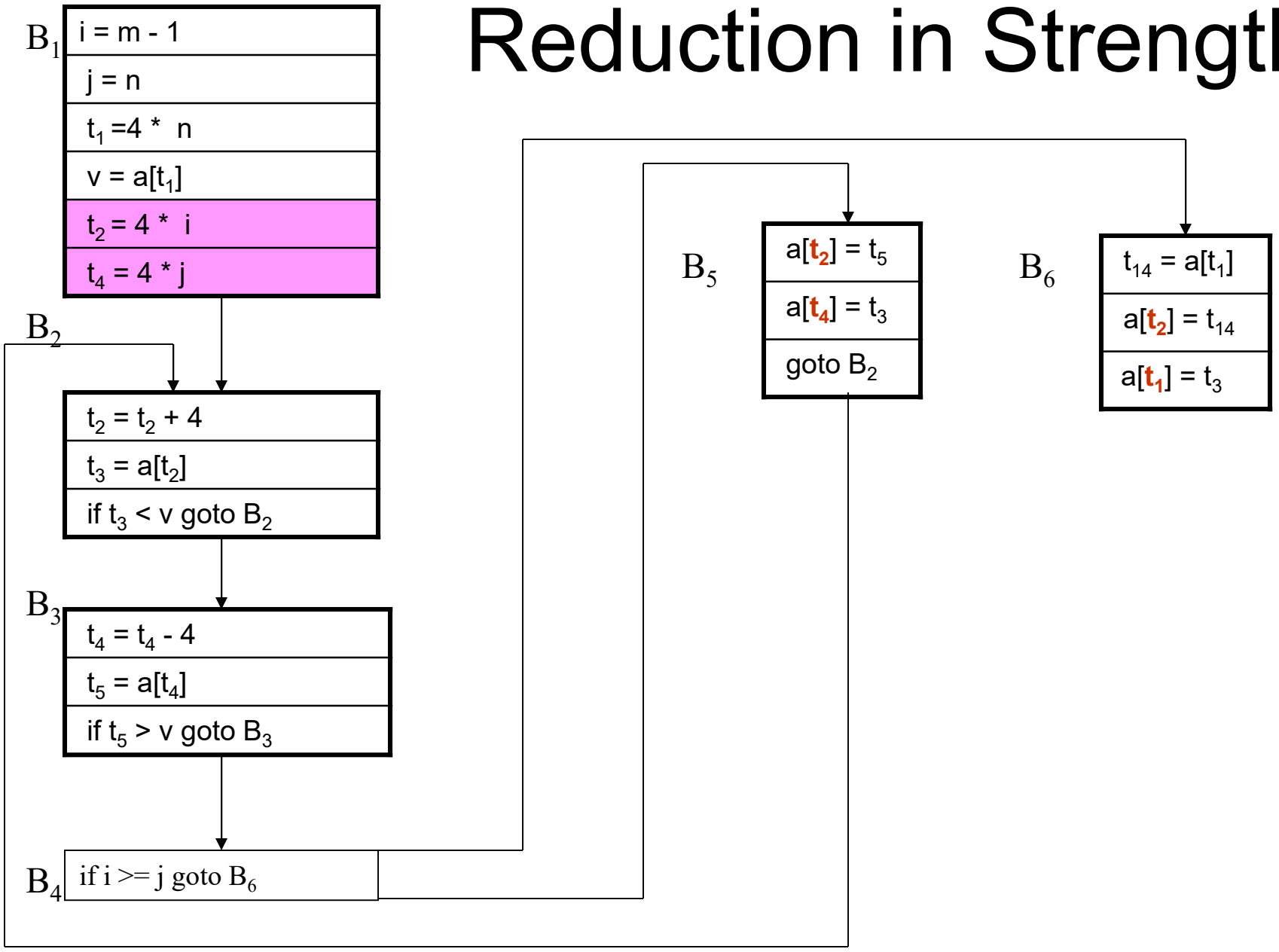
# Dead Code Elimination



# Dead Code Elimination



# Reduction in Strength



# Reduction in Strength