THEORY OF COMPUTATION AND COMPILERS

Unit - IV

Code Optimization | Data Flow Analysis

Code Optimization

- Consideration for Optimization
- Scope of Optimization
- *Basic blocks and Local Optimization
- Loop Optimization
- Frequency Reduction
- Folding
- DAG Representation

Data Flow Analysis

- Flow Graph
- Data Flow Equation
- Global Optimization
- Redundant Sub Expression Elimination
- Induction Variable Elements
- Live Variable Analysis
- Copy Propagation

Dr. R. Madana Mohana

Professor, Artificial Intelligence & Data Science | I/c-Head, Artificial Intelligence & Machine Learning

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

Hyderabad - 500 075, Telangana, INDIA

www.cbit.ac.in

Code Optimization Classification

- By Scope:
  - Local Optimization: within a single basic block.
  - Peephole Optimization : on a window of instructions (usually local)
  - Loop-level Optimization : on one or more loops or loop nests.
  - Global: for an entire procedure
  - Interprocedural: across multiple procedures or whole program.
- By machine information used:
  - Machine-independent versus machine-dependent.
- By effect on program structure:
  - Algebraic transformations (e.g., `x+0`, `x*1`, `3*z*4`, …)
  - Reordering transformations (change the order of 2 computations)
    - Loop transformations: loop-level reordering transformations.

Global Optimization

- Global Optimization: across blocks
  - Common sub-expression elimination
  - Strength reduction
  - Data flow analysis

Global Optimization

A Running Example (Quicksort)

```
void quicksort(int m, int n)
    /* recursively sorts a[m] through a[n] */
{
    int i, j;
    int v, x;
    if (n <= m) return;
    /* fragment begins here */
    i = m-1; j = n; v = a[n];
    while (1) {
        do i = i+1; while (a[i] < v);
        do j = j-1; while (a[j] > v);
        if (i >= j) break;
        x = a[i]; a[i] = a[j]; a[j] = x; /* swap a[i], a[j] */
    }
    x = a[i]; a[i] = a[n]; a[n] = x; /* swap a[i], a[n] */
    /* fragment ends here */
    quicksort(m,j); quicksort(i+1,n);
}
```

Fig: C code for quicksort

Global Optimization

A Running Example (Quicksort)

```
(1)   i = m-1
(2)   j = n
(3)   t1 = 4*n
(4)   v = a[t1]
(5)   i = i+1
(6)   t2 = 4*i
(7)   t3 = a[t2]
(8)   if t3<v goto (5)
(9)   j = j-1
(10)  t4 = 4*j
(11)  t5 = a[t4]
(12)  if t5>v goto (9)
(13)  if i>=j goto (23)
(14)  t6 = 4*i
(15)  x = a[t6]
(16)  t7 = 4*i
(17)  t8 = 4*j
(18)  t9 = a[t8]
(19)  a[t7] = t9
(20)  t10 = 4*j
(21)  a[t10] = x
(22)  goto (5)
(23)  t11 = 4*i
(24)  x = a[t11]
(25)  t12 = 4*i
(26)  t13 = 4*n
(27)  t14 = a[t13]
(28)  a[t12] = t14
(29)  t15 = 4*n
(30)  a[t15] = x
```

Fig: Three- address code for above C fragment

Global Optimization

A Running Example (Quicksort)

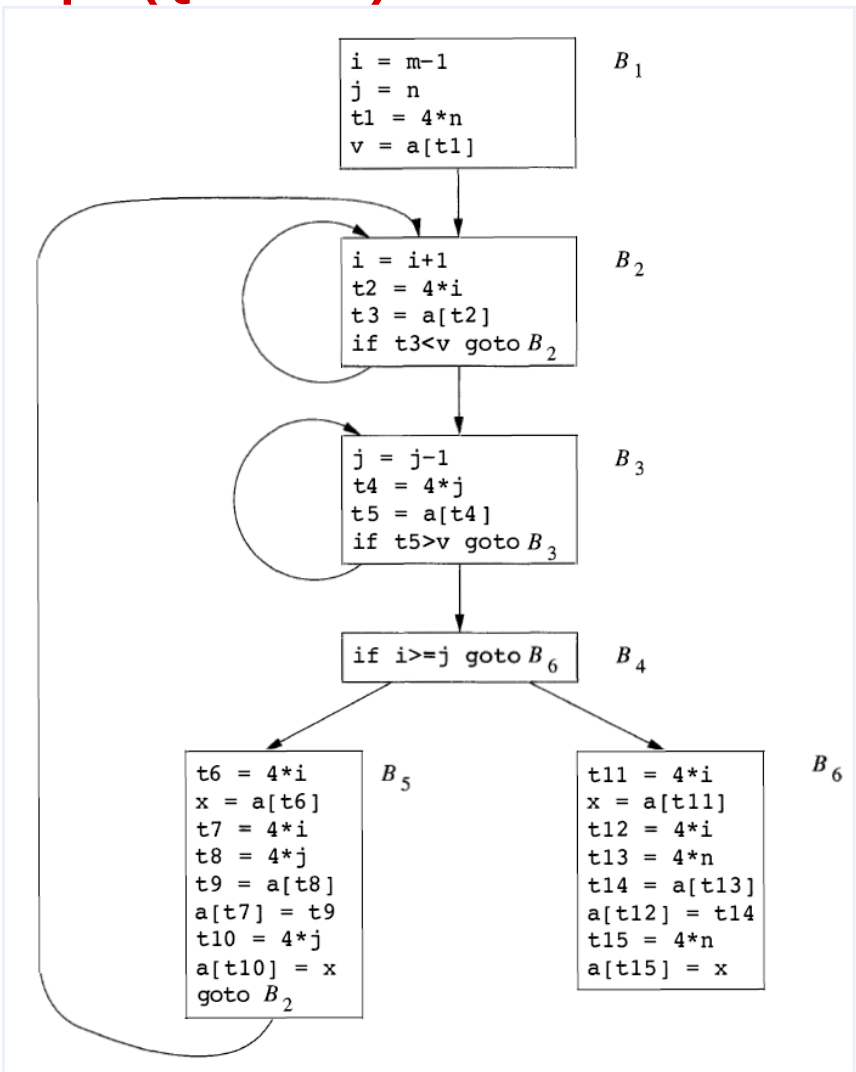


Fig: Flowgraph for above three-address code

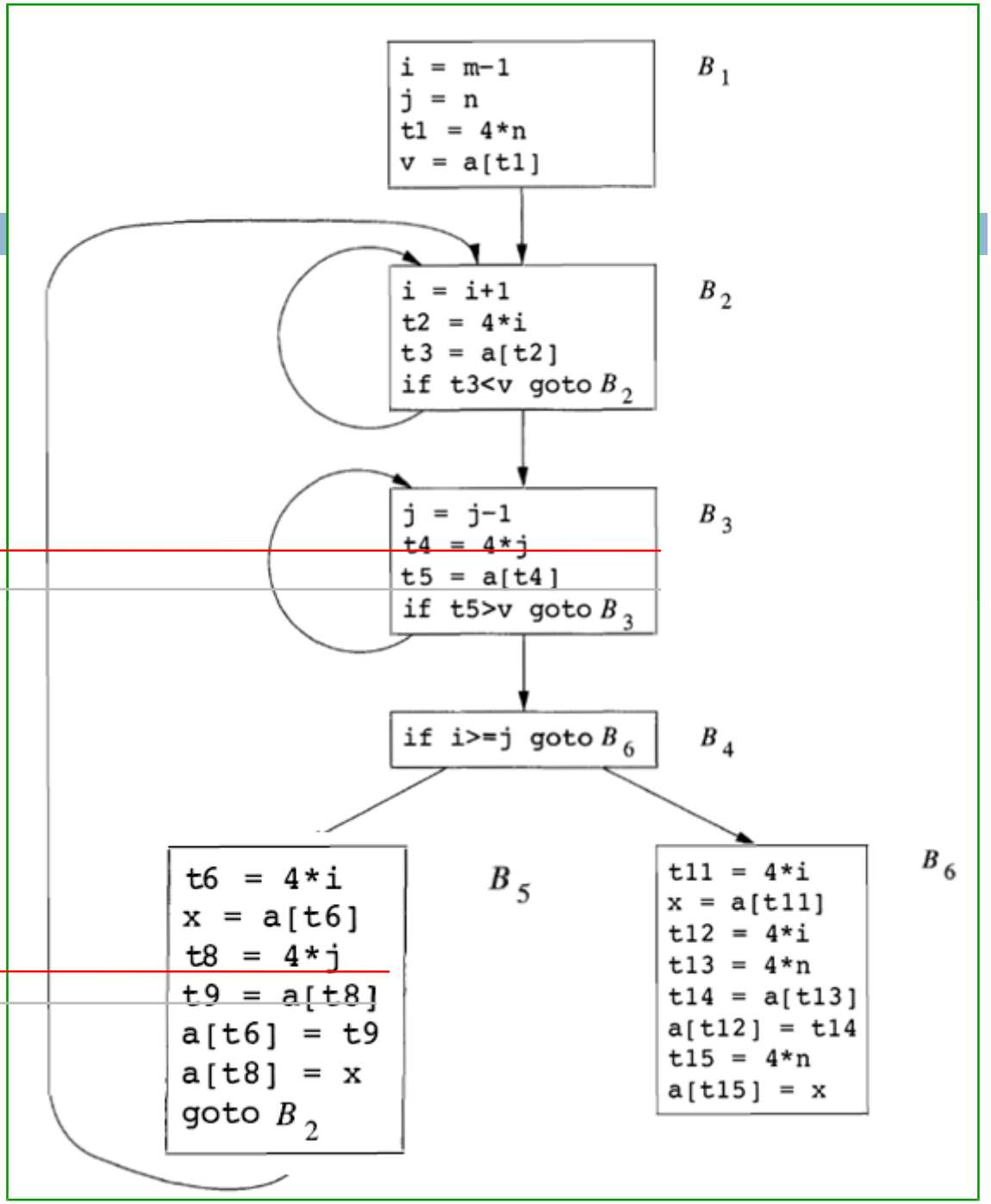
Global Optimization

Common Subexpression Elimination

- An occurrence of an expression E is called a *common subexpression* if E was previously computed and the values of the variables in E have not changed since the previous computation. We avoid recomputing E if we can use its previously computed value; that is, the variable x to which the previous computation of E was assigned has not changed in the interim.
- An expression, say x+y, is redundant iff along every path from the procedure's entry it has been evaluated and its constituent subexpressions (x, y) have not been redefined.

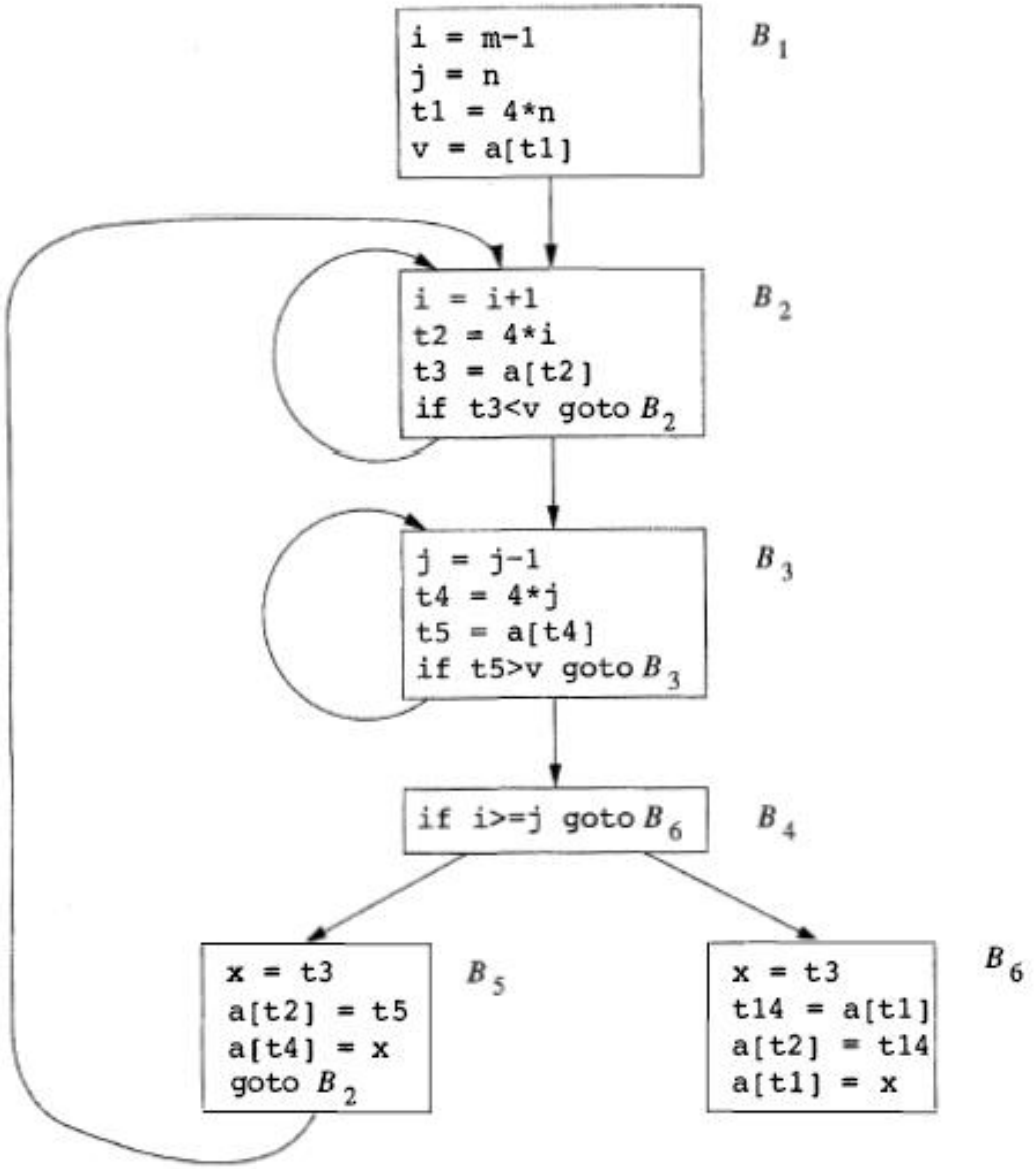
Common Subexpression Elimination

Global:



Global Optimization

Common Subexpression Elimination



Global Optimization

Reduction in Strength

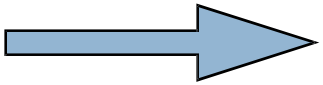
- Strength reduction
  - The transformation of replacing an expensive operation, such as multiplication, by a cheaper one, such as addition
- Induction variables lead to
  - strength reduction
  - eliminate computation

Global Optimization

Strength Reduction

- Replace expensive operations with simpler ones
- Example: Multiplications replaced by additions

```
y := x * 2
```

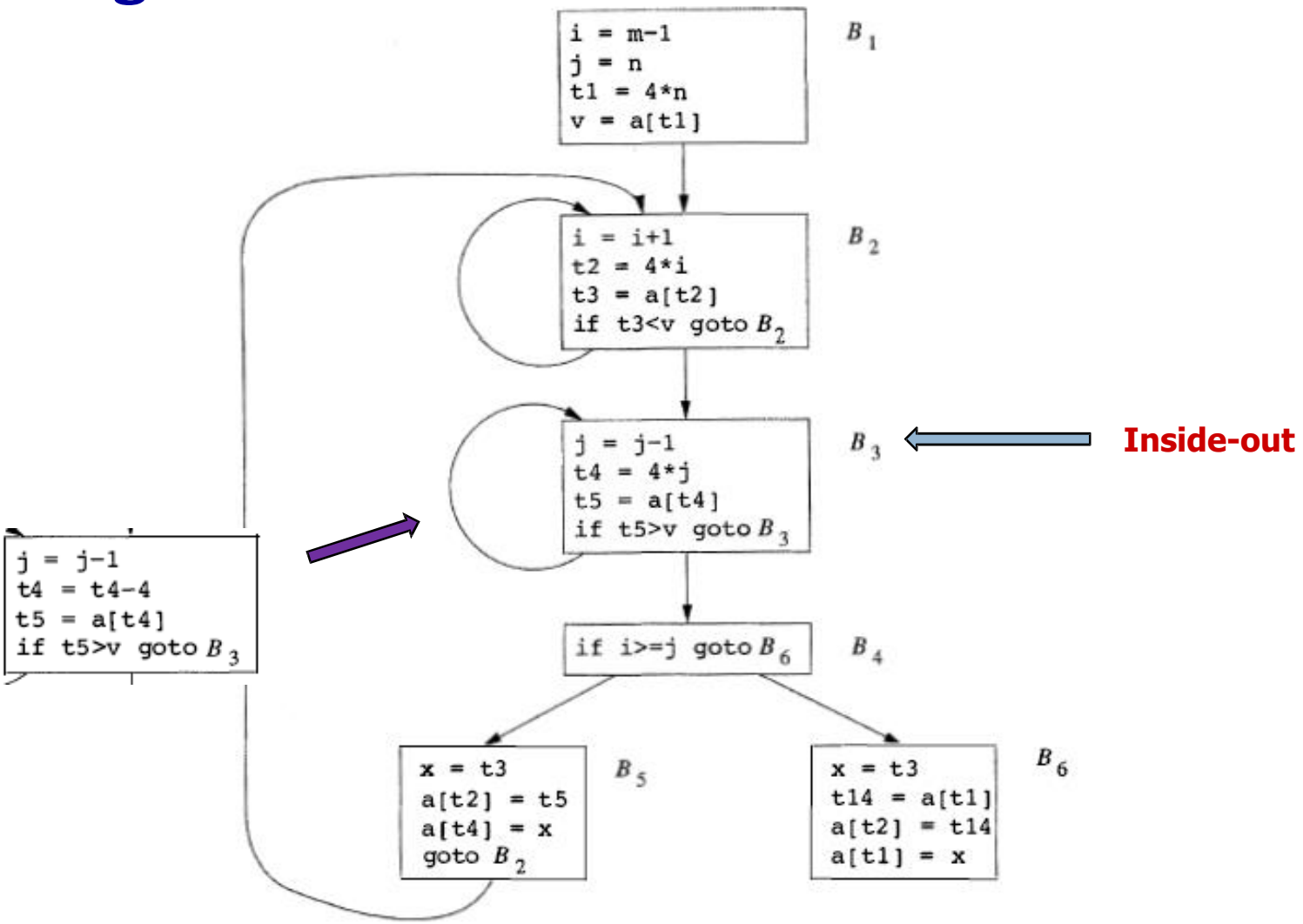


```
y := x + x
```

Peephole optimizations are often strength reductions

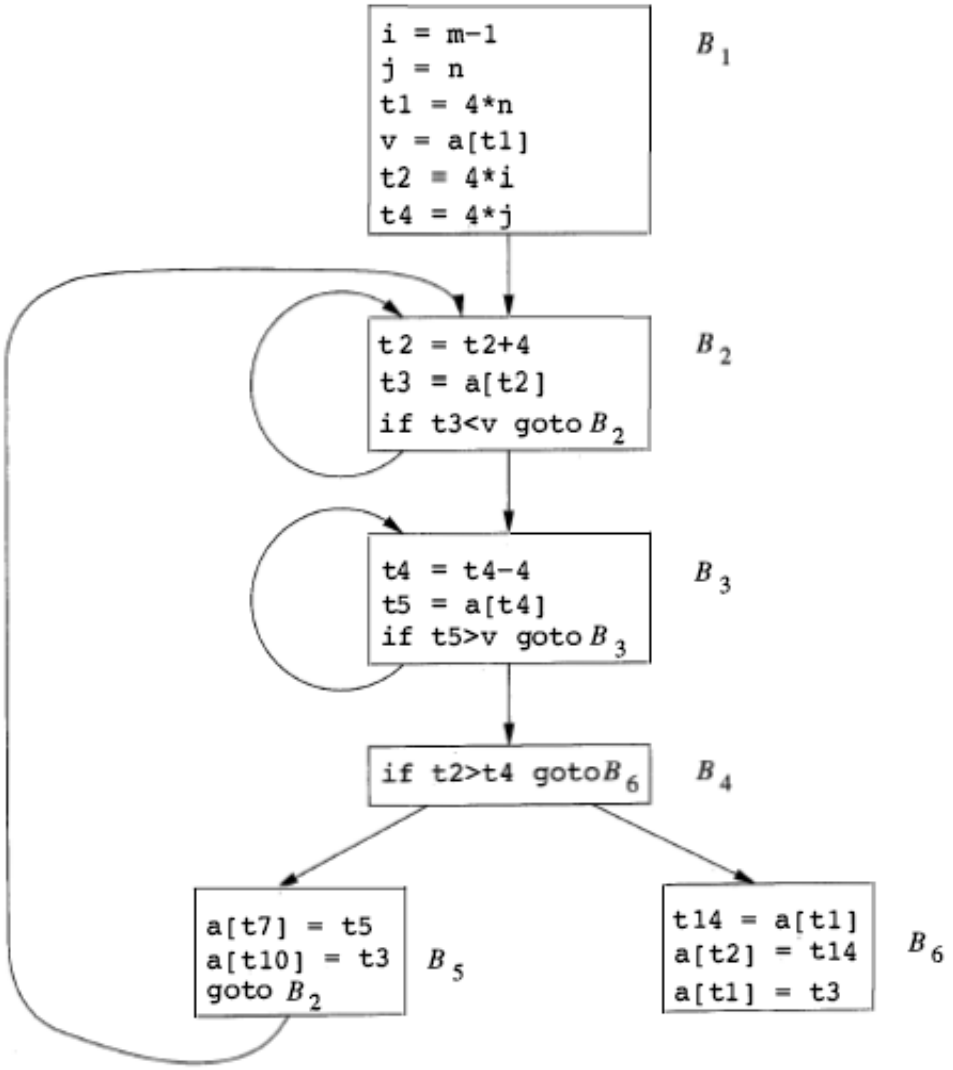
Global Optimization

Strength Reduction



Global Optimization

Strength Reduction



Induction Variable Elements

Induction Variables and Reduction in Strength

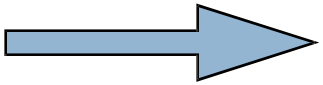
- Induction variable
  - For an induction variable x, there is a positive or negative constant c such that each time x is assigned, its value increases by c
- Induction variables can be computed with a single increment (addition or subtraction) per loop iteration
- Strength reduction
  - The transformation of replacing an expensive operation, such as multiplication, by a cheaper one, such as addition
- Induction variables lead to
  - strength reduction
  - eliminate computation

Induction Variable Elements

Strength Reduction

- Replace expensive operations with simpler ones
- Example: Multiplications replaced by additions

```
y := x * 2
```

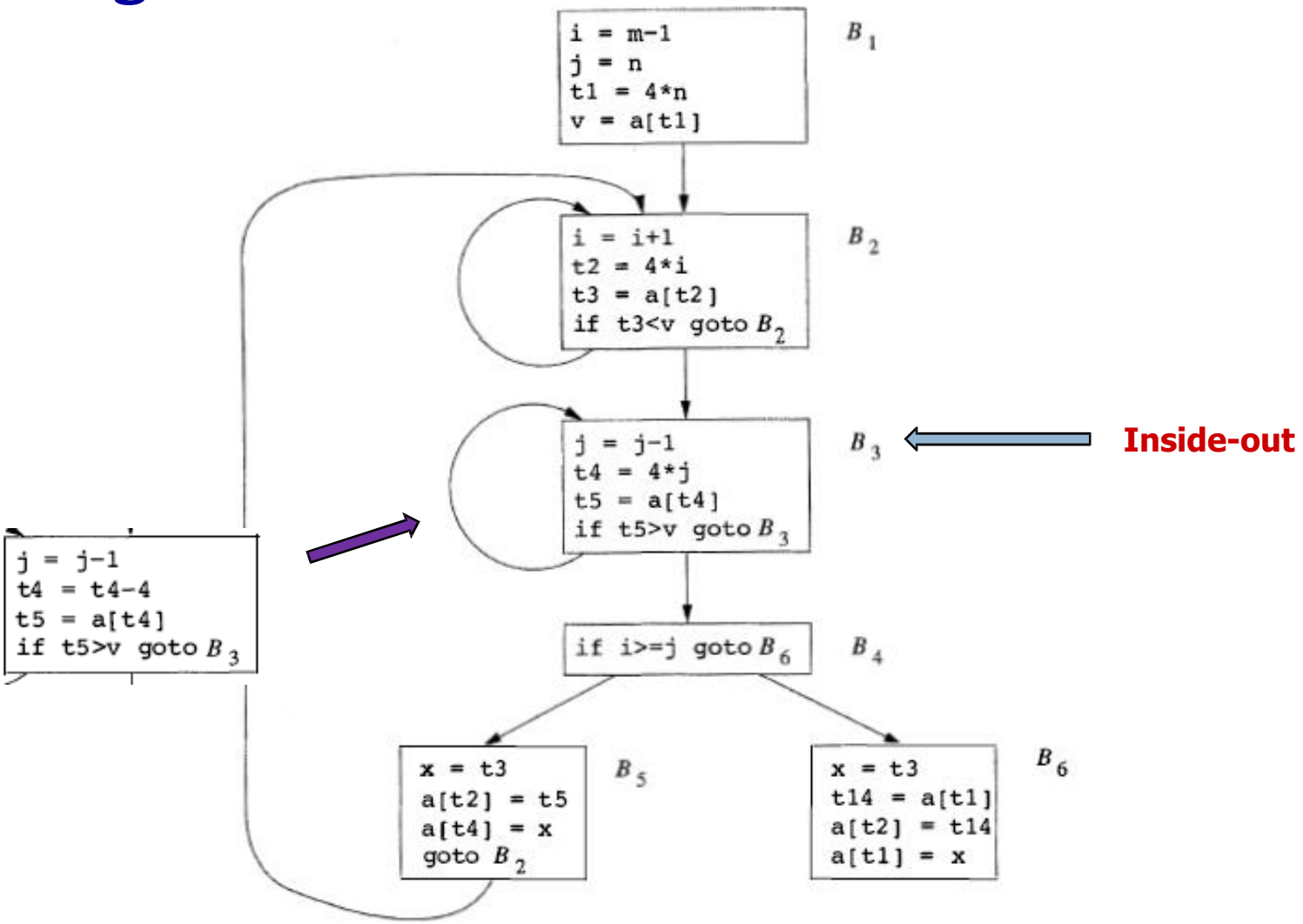


```
y := x + x
```

Peephole optimizations are often strength reductions

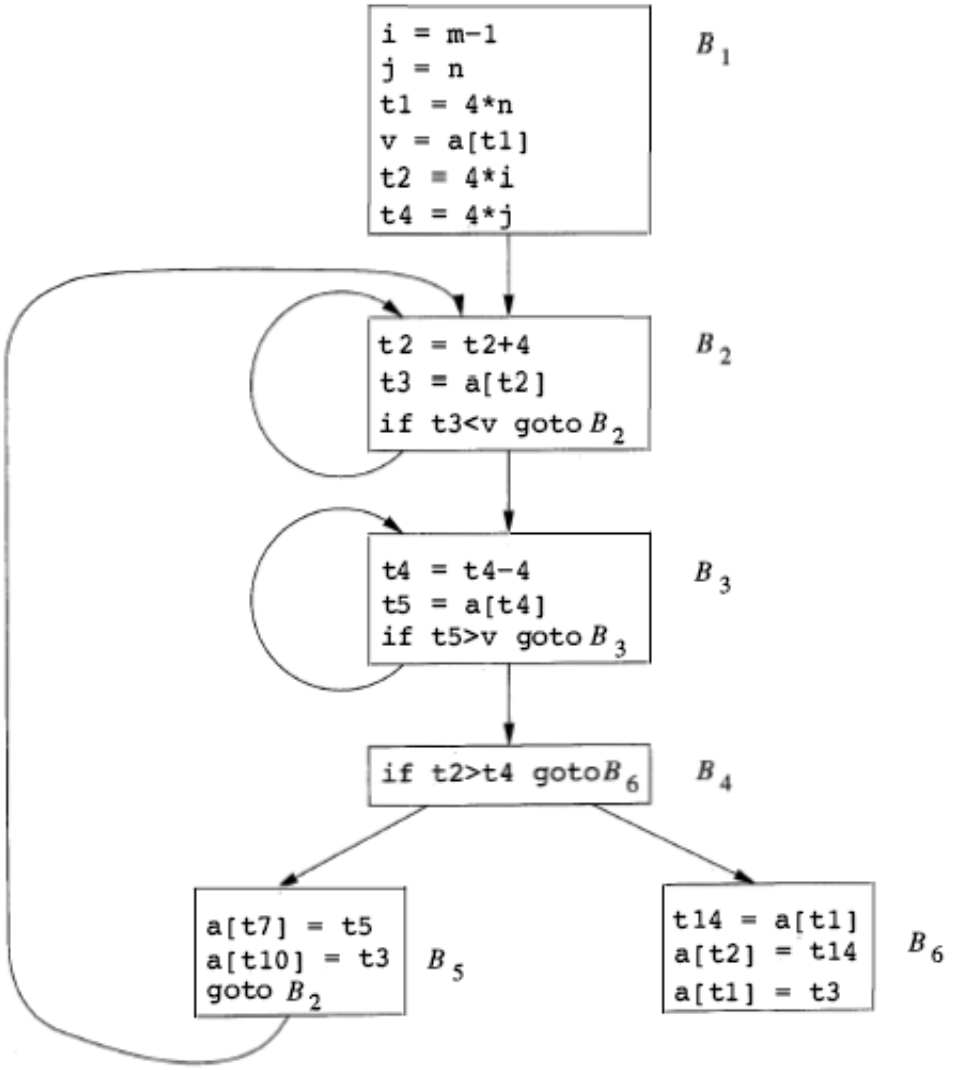
Induction Variable Elements

Strength Reduction



Induction Variable Elements

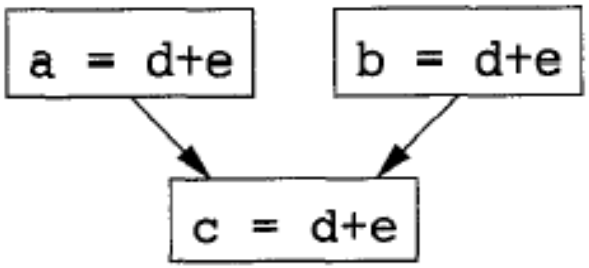
Strength Reduction



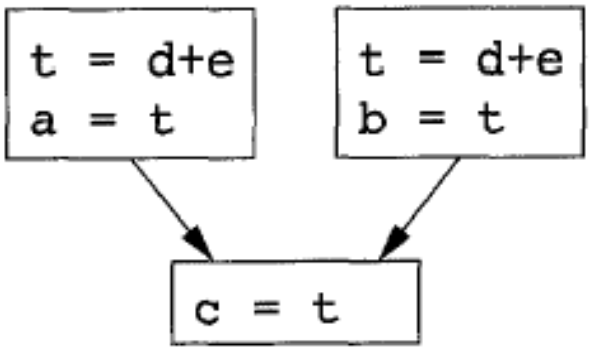
Copy Propagation

Copy Propagation

- Copy statements or Copies
  - u = v



(a)

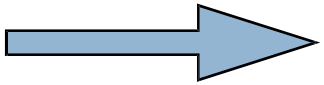


(b)

Copy Propagation

- for a statement x := y
- replace later uses of x with y, if x and y have not been changed.

```
x := y
c := 1 + x
d := x + c
```



```
x := y
c := 1 + y
d := y + c
```

Analysis needed, as y and x can be assigned more than once!