Program Analysis

https://www.cse.iitb.ac.in/~karkare/cs618/

Welcome & Introduction

Amey Karkare
Dept of Computer Science and Engg
IIT Kanpur
Visiting IIT Bombay
karkare@cse.iitk.ac.in
karkare@cse.iitb.ac.in

Program Analysis: About the course

Program Analysis: About the course

- Analysis of a Program, by a Program, for a Program

"Democracy is the government of the people, by the people, for the people" - Abraham Lincoln

Program Analysis: About the course

- Analysis of a Program, by a Program, for a Program
- of a Program - User Program

"Democracy is the government of the people, by the people, for the people" - Abraham Lincoln

Program Analysis: About the course

- Analysis of a Program, by a Program, for a Program
- of a Program - User Program
- by a Program - Analyzer (Compiler, Runtime)

"Democracy is the government of the people, by the people, for the people" - Abraham Lincoln

Program Analysis: About the course

- Analysis of a Program, by a Program, for a Program
- of a Program - User Program
- by a Program - Analyzer (Compiler, Runtime)
- for a Program - Optimizer, Verifier, ...

"Democracy is the government of the people, by the people, for the people" - Abraham Lincoln

Expectations from You

Expectations from You

- Basic Compilers Knowledge

Expectations from You

- Basic Compilers Knowledge
- Write code
  - C/C++ for Assignments
  - C/C++/Java/Python for Project (Tentative)

Expectations from You

- Basic Compilers Knowledge
- Write code
  - C/C++ for Assignments
  - C/C++/Java/Python for Project (Tentative)
- Understand and modify large code base
  - GCC, LLVM, SOOT

Expectations from You

- Basic Compilers Knowledge
- Write code
  - C/C++ for Assignments
  - C/C++/Java/Python for Project (Tentative)
- Understand and modify large code base
  - GCC, LLVM, SOOT
- Read state of the art research papers
  - Discussions in class

Your Expectations

Your Expectations

- ?

Quick Quizzes (QQs)

Quick Quizzes (QQs)

- There will be small quizzes (10-15 min duration) during the class.

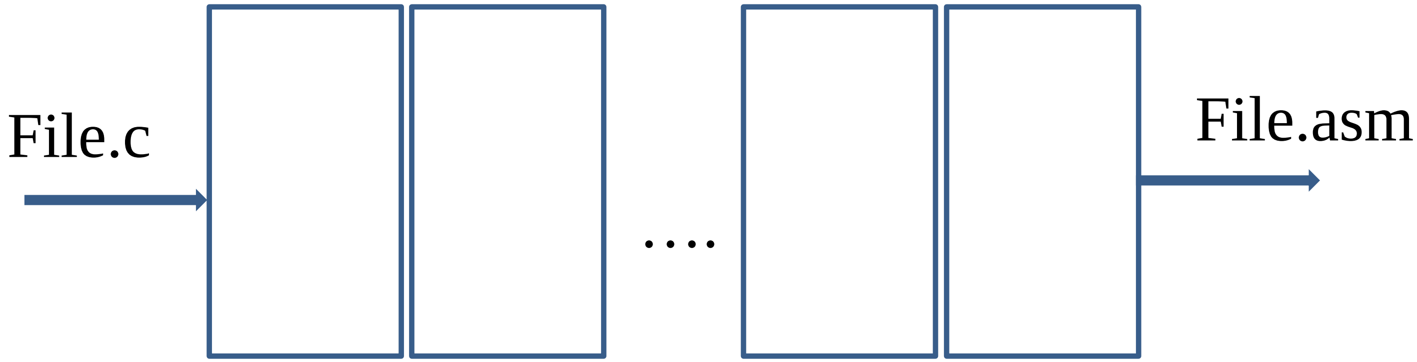
Quick Quizzes (QQs)

- There will be small quizzes (10-15 min duration) during the class.
- These can be announced or **un-announced (surprize** quizzes).

Quick Quizzes (QQs)

- There will be small quizzes (10-15 min duration) during the class.
- These can be announced or **un-announced (surprize** quizzes).
- Always bring a pen and some loose papers to the class

QQ #1 (Ungraded)



- What are the various phases of compilers that you know? (5 minutes)

Assignments / Exercises

Assignments / Exercises

- Short assignments to apply the lecture material.

Assignments / Exercises

- Short assignments to apply the lecture material.
- Assignments will have some written and some programming tasks.

Assignments / Exercises

- Short assignments to apply the lecture material.
- Assignments will have some written and some programming tasks.
- 4-5 Assignments for the semester

Using Program Analysis

Using Program Analysis

- Compiler Code Optimizations

Using Program Analysis

- Compiler Code Optimizations
- Why are optimizations important?

Using Program Analysis

- Compiler Code Optimizations
- Why are optimizations important?
- Why not write optimized code to begin with?

Using Program Analysis

- Compiler Code Optimizations
- Why are optimizations important?
- Why not write optimized code to begin with?
- Where do optimizations fit in the compiler flow?

Code Optimization

Code Optimization

- Machine Independent
  - Remove redundancy introduced by the Programmer
  - Remove redundancy not required by later phases of compiler
  - Take advantage of algebraic properties of operators

Code Optimization

- Machine Independent
  - Remove redundancy introduced by the Programmer
  - Remove redundancy not required by later phases of compiler
  - Take advantage of algebraic properties of operators
- Machine dependent
  - Take advantage of the properties of target machine

Code Optimization

- Machine Independent
  - Remove redundancy introduced by the Programmer
  - Remove redundancy not required by later phases of compiler
  - Take advantage of algebraic properties of operators
- Machine dependent
  - Take advantage of the properties of target machine

Optimization must preserve the semantics of the original program!

Machine Independent Optimizations

Motivational Example

```
void quicksort(int m, int n)
/* recursively sort a[m] through a[n] */
{
  int i, j;
  int v, x;
  if(n <= m) return;
  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;
  }
  x = a[i]; a[i] = a[n]; a[n] = x;
  quicksort(m,j); quicksort(i+1,n);
}
```

Motivational Example

```
void quicksort(int m, int n)
/* recursively sort a[m] through a[n] */
{
  int i, j;
  int v, x;
  if(n <= m) return;
  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;
  }
  x = a[i]; a[i] = a[n]; a[n] = x;
}
```

```
(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
```

```
(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
```

```
(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
```

```
(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
```

```
(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
```

```
(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
```

```
(1)  i = m-1
(2)  j = n
(3)  t1 = 4*n
(4)  v = a[t1]
```

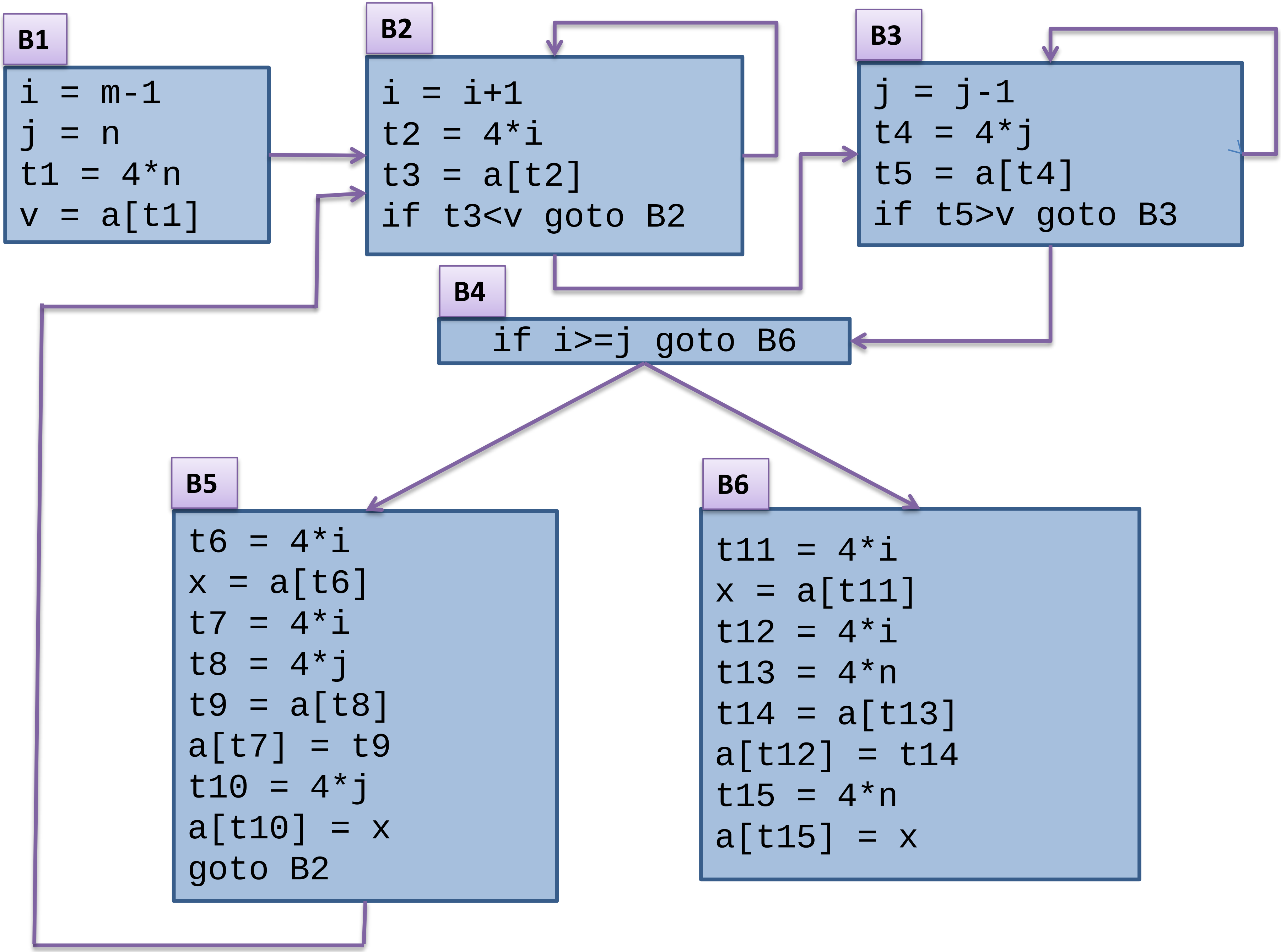
```
(5)  i = i+1
(6)  t2 = 4*i
(7)  t3 = a[t2]
(8)  if t3<v goto (5)
```

```
(9)  j = j-1
(10) t4 = 4*j
(11) t5 = a[t4]
(12) if t5>v goto (9)
```

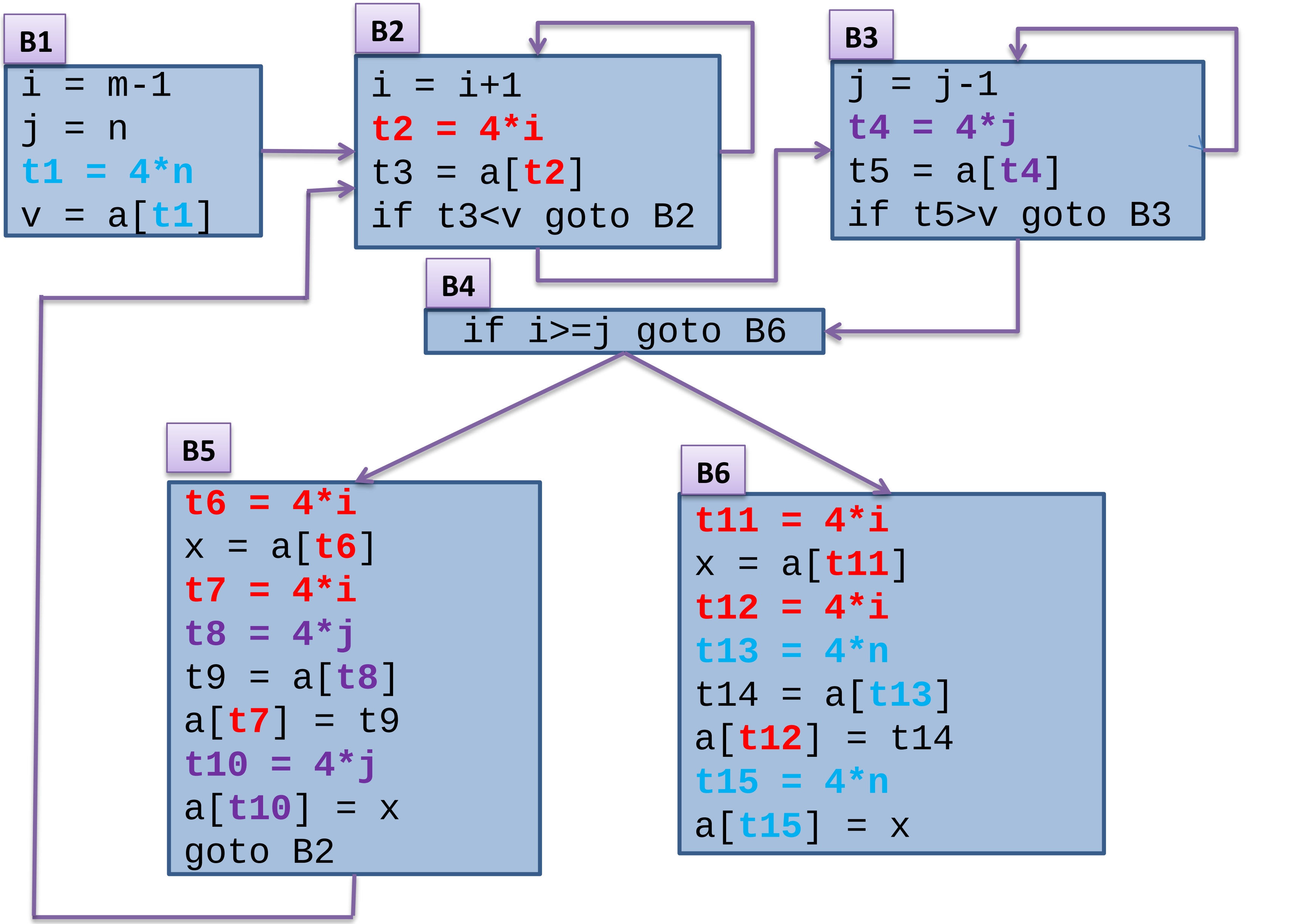
```
(13) if i>=j goto(23)
```

```
(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)
```

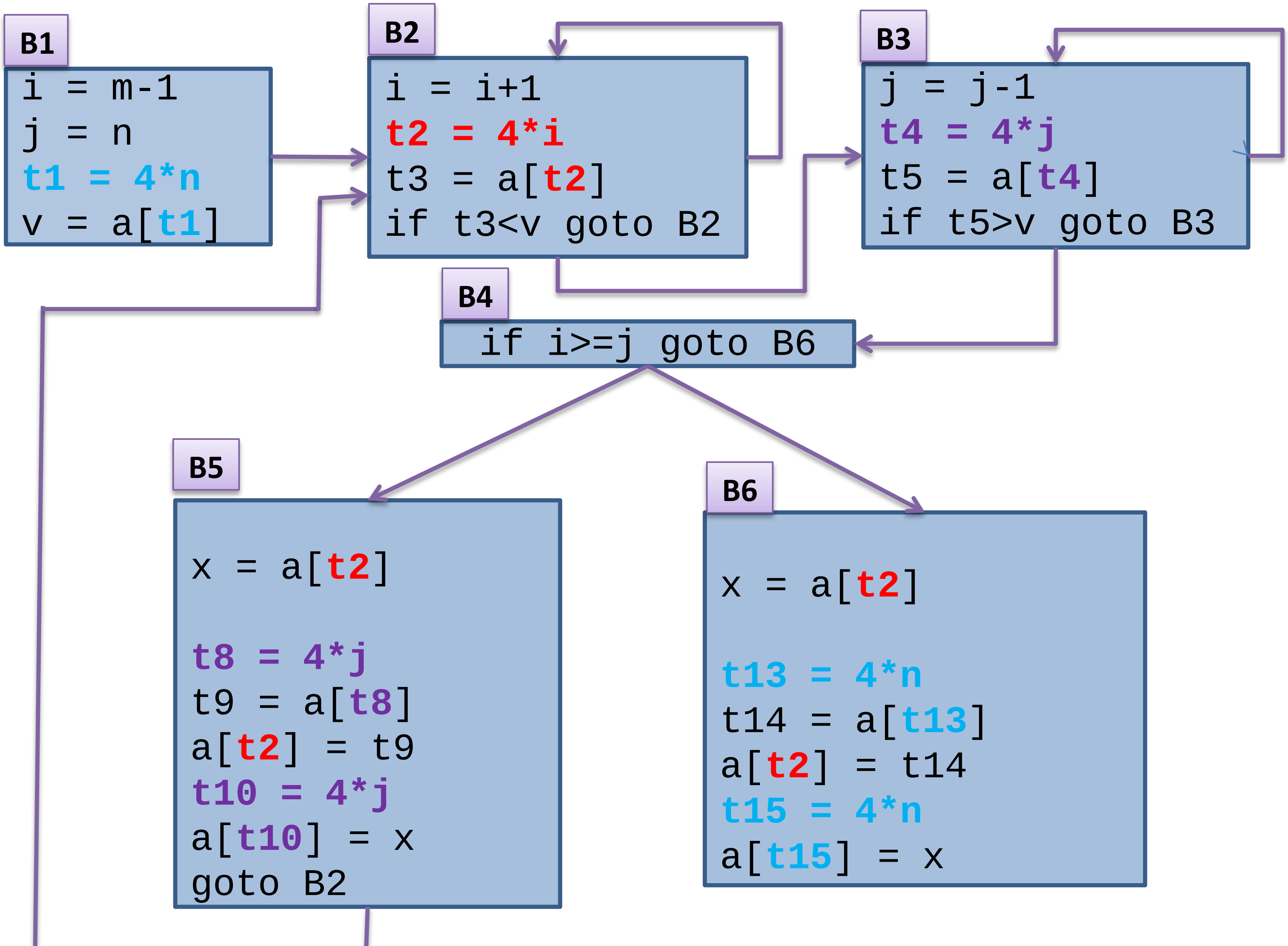
```
(23) t11 = 4*i
(24) x = a[t11]
(25) t12 = 4*i
(26) t13 = 4*n
(27) t14 = a[t13]
(28) a[t12] = t14
(29) t15 = 4*n
(30) a[t15] = x
```


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

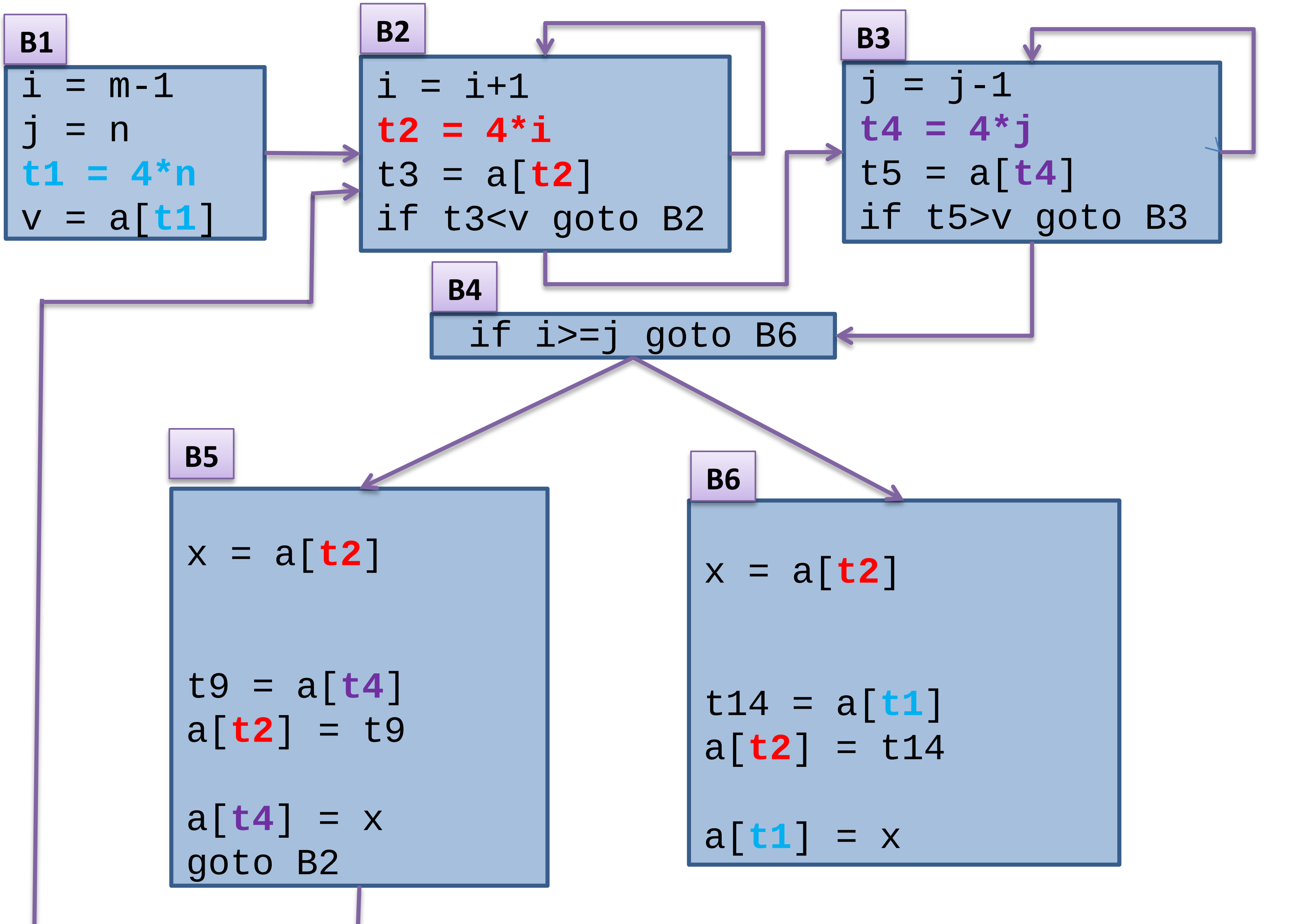
Common Subexpression Elimination



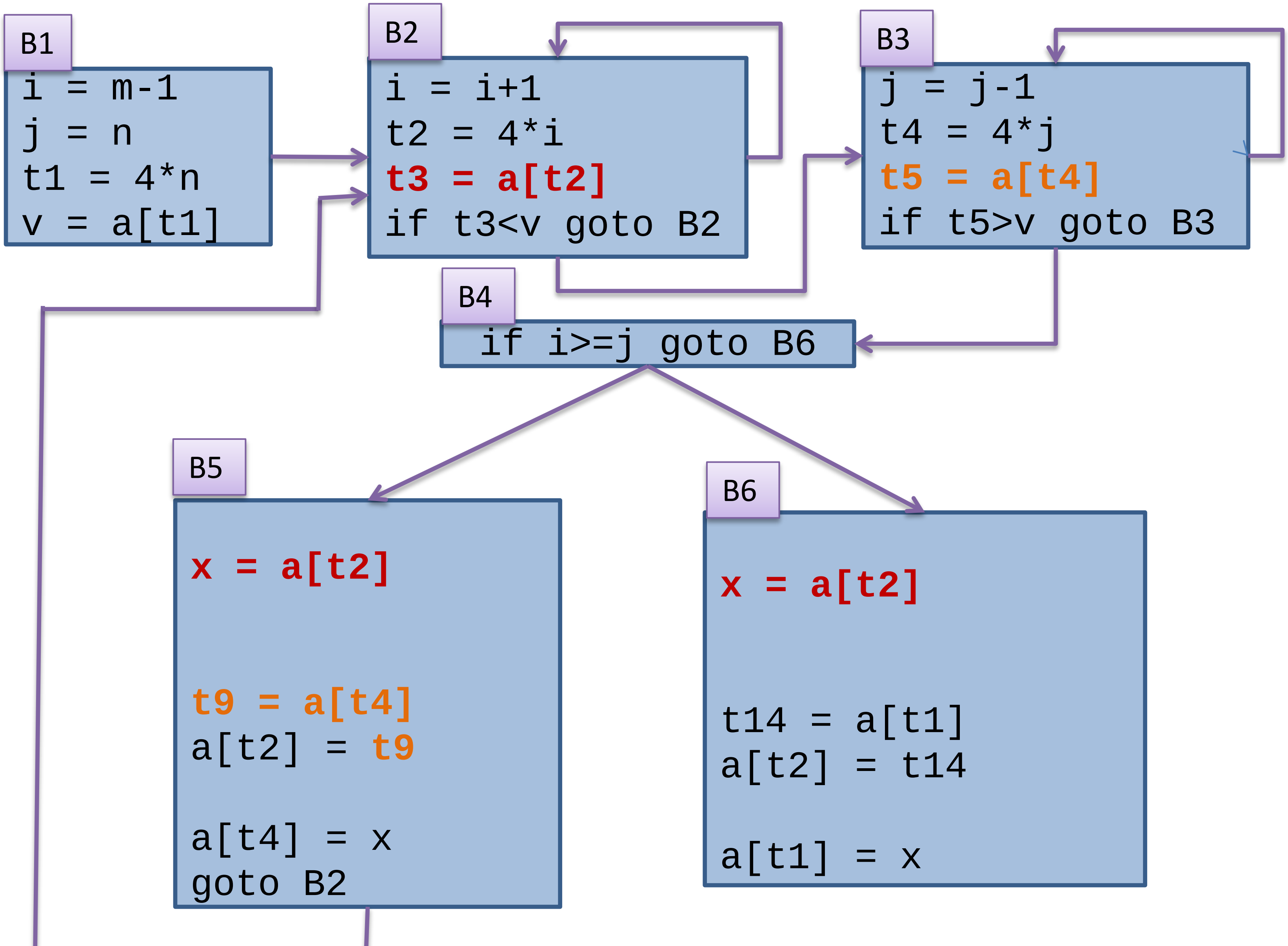
Common Subexpression Elimination



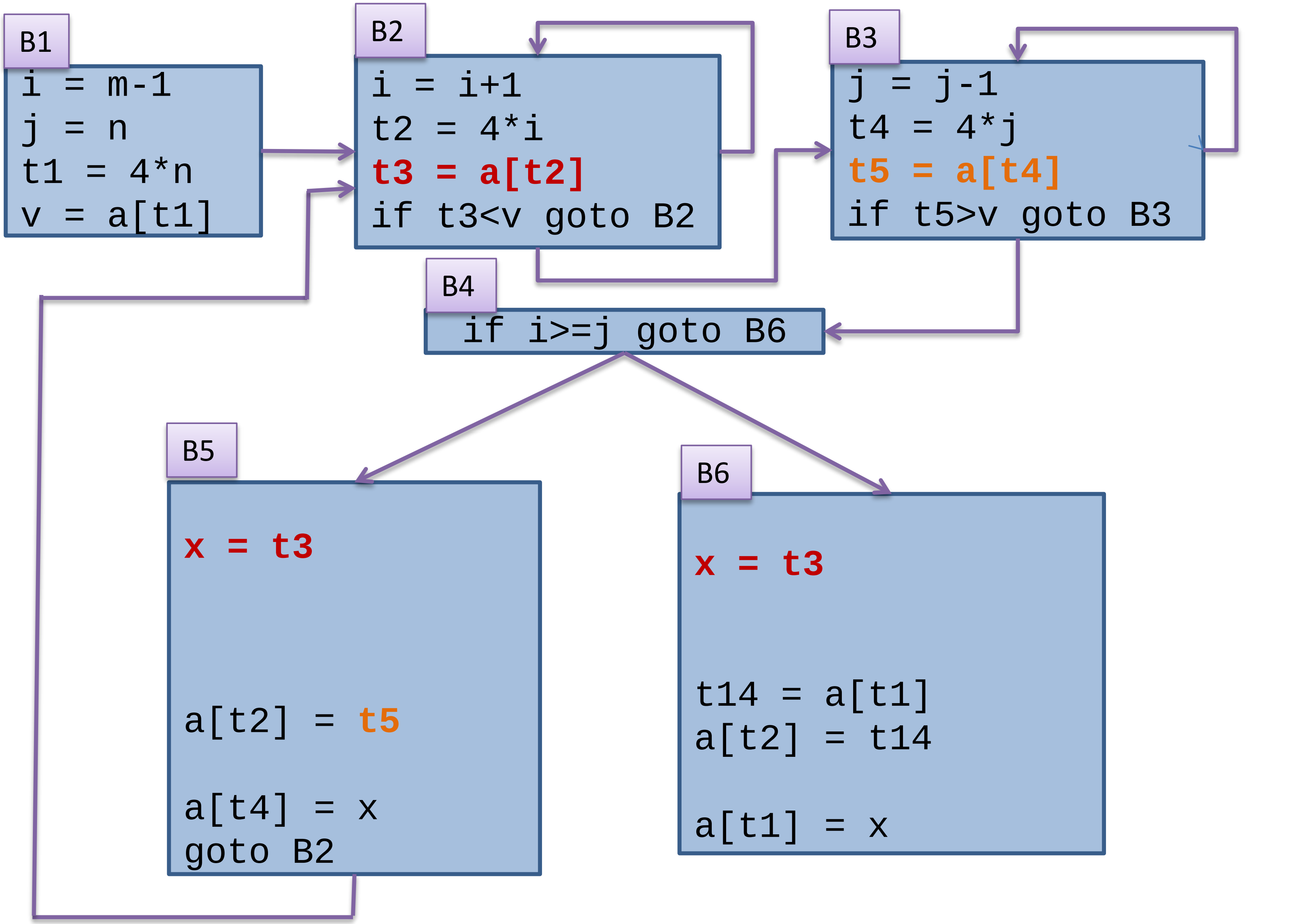
Common Subexpression Elimination



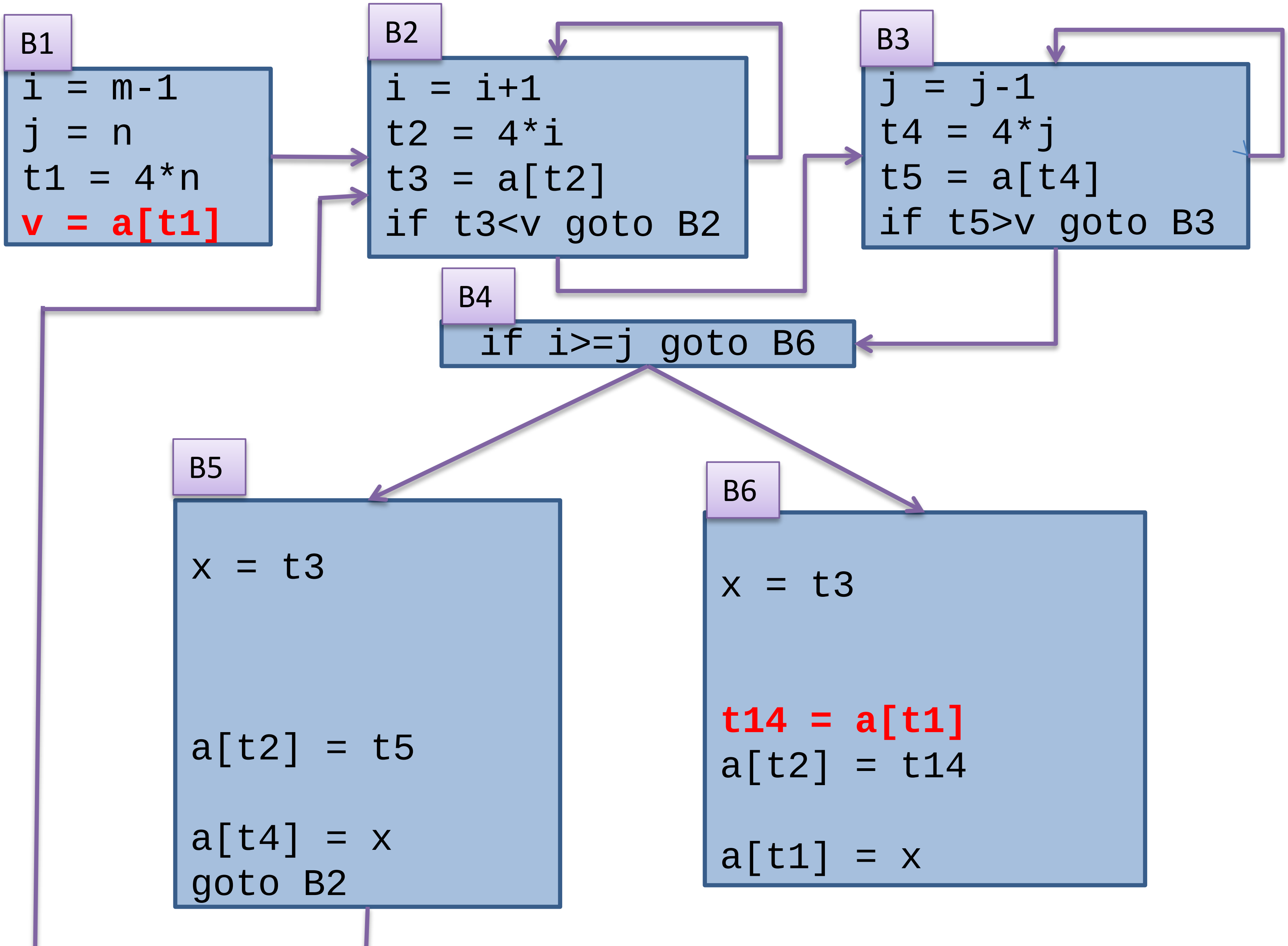
Common Subexpression Elimination

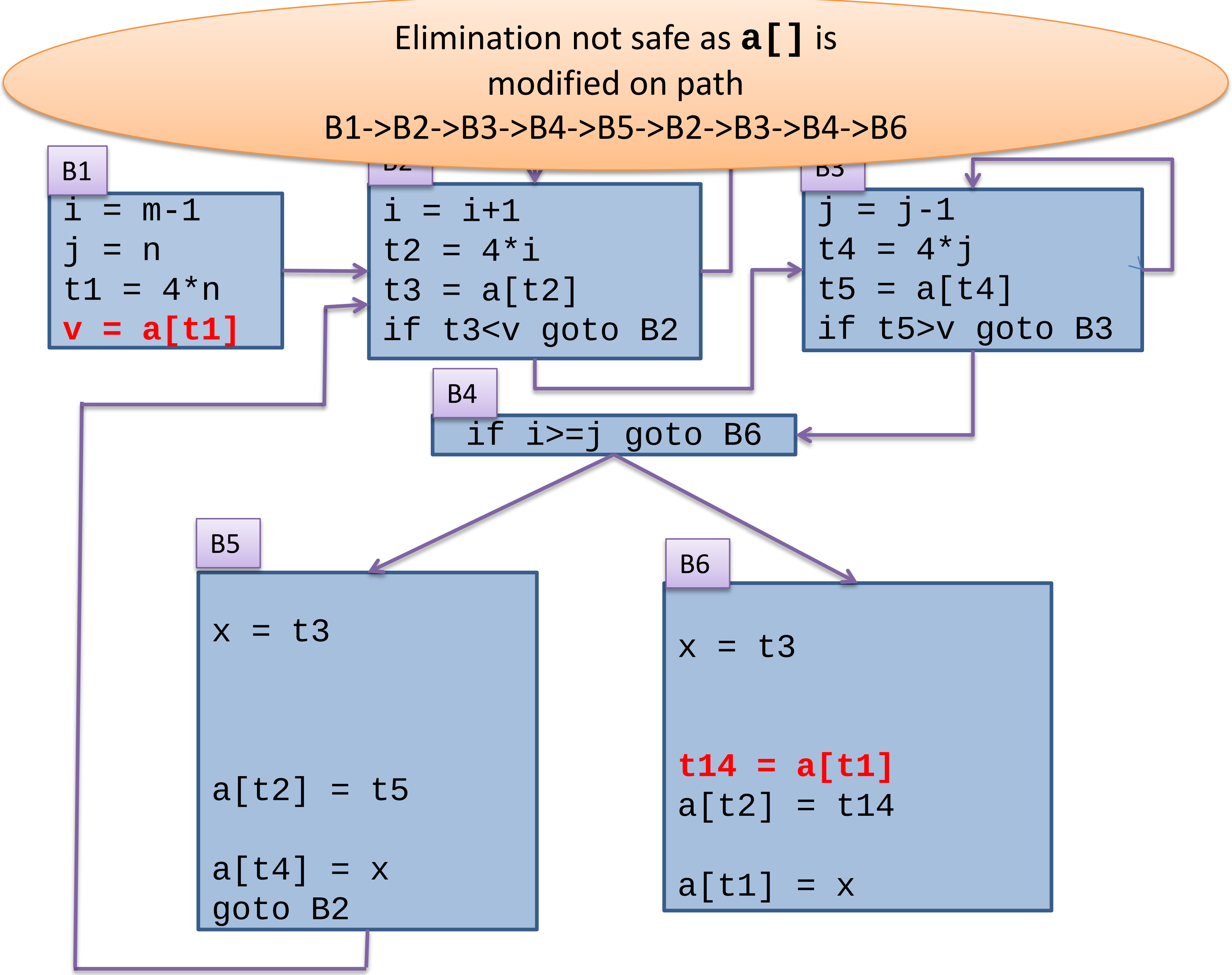


Common Subexpression Elimination

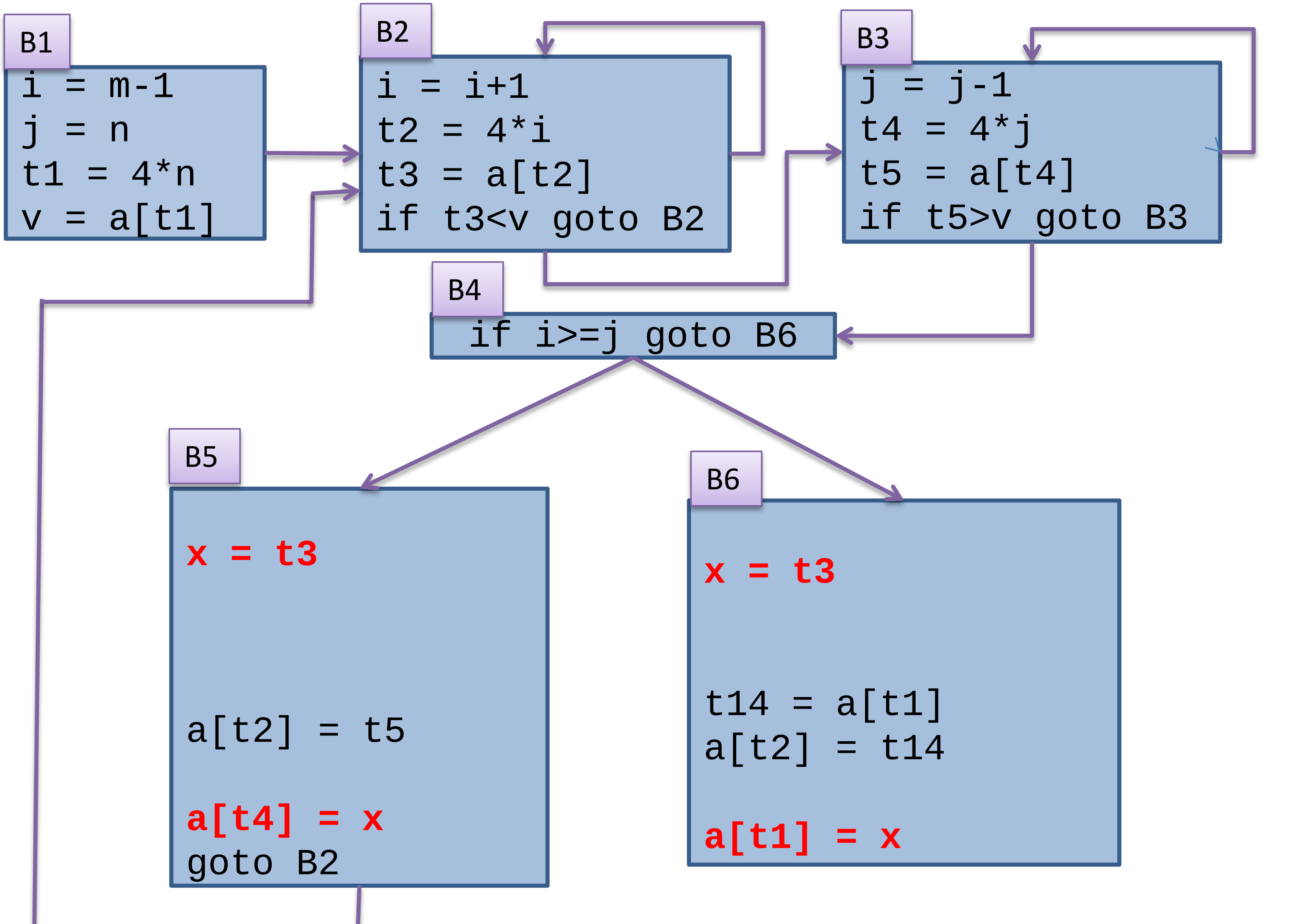


Did I miss one common subexpression?

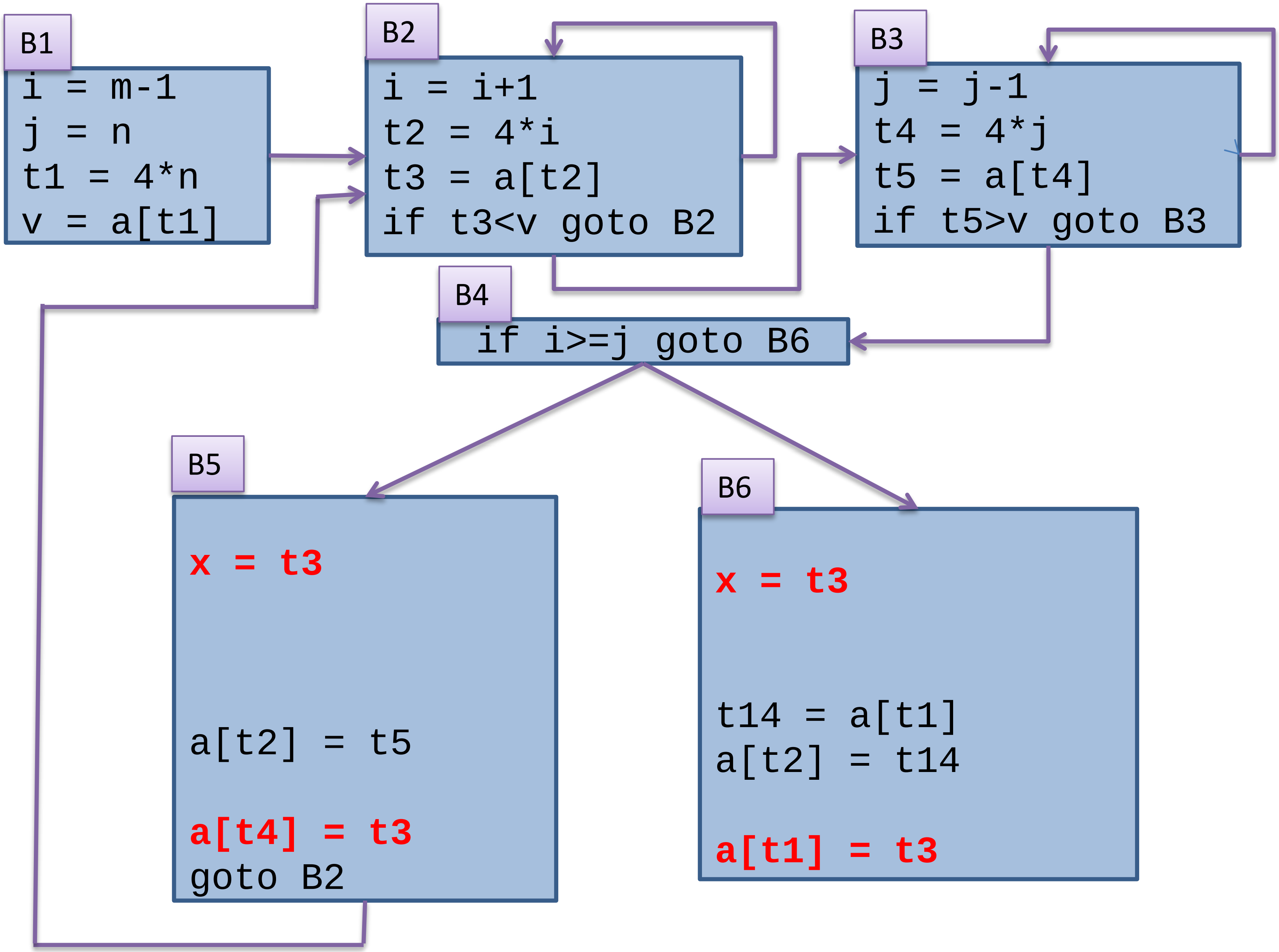



Elimination not safe as a[] is modified on path B1->B2->B3->B4->B5->B2->B3->B4->B6
B1
i = m-1
j = n
t1 = 4*n
v = a[t1]
i = i+1
t2 = 4*i
t3 = a[t2]
if t3<v goto B2
j = j-1
t4 = 4*j
t5 = a[t4]
if t5>v goto B3
B4
if i>=j goto B6
B5
x = t3
a[t2] = t5
a[t4] = x
goto B2
B6
x = t3
t14 = a[t1]
a[t2] = t14
a[t1] = x

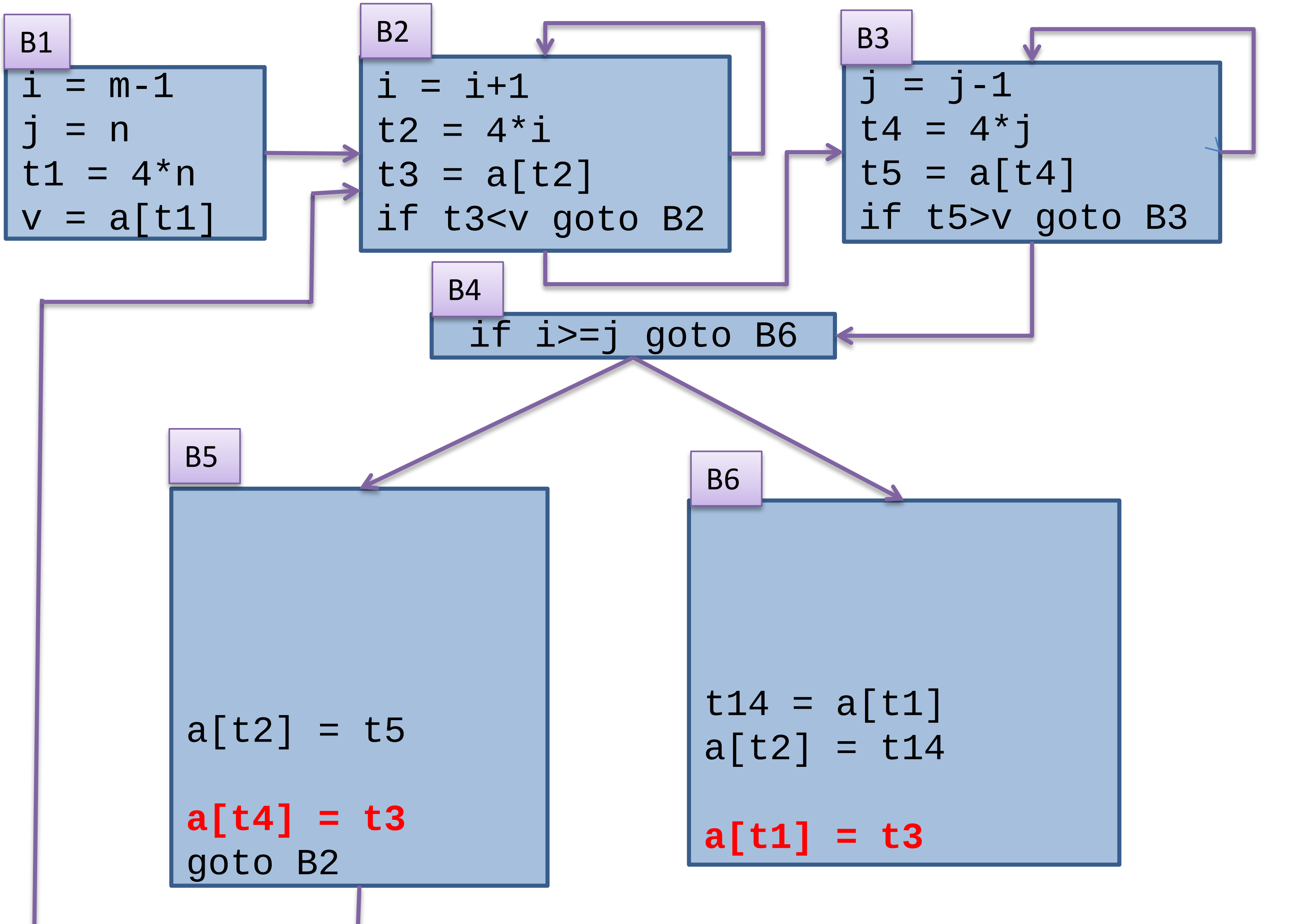
Copy Propagation



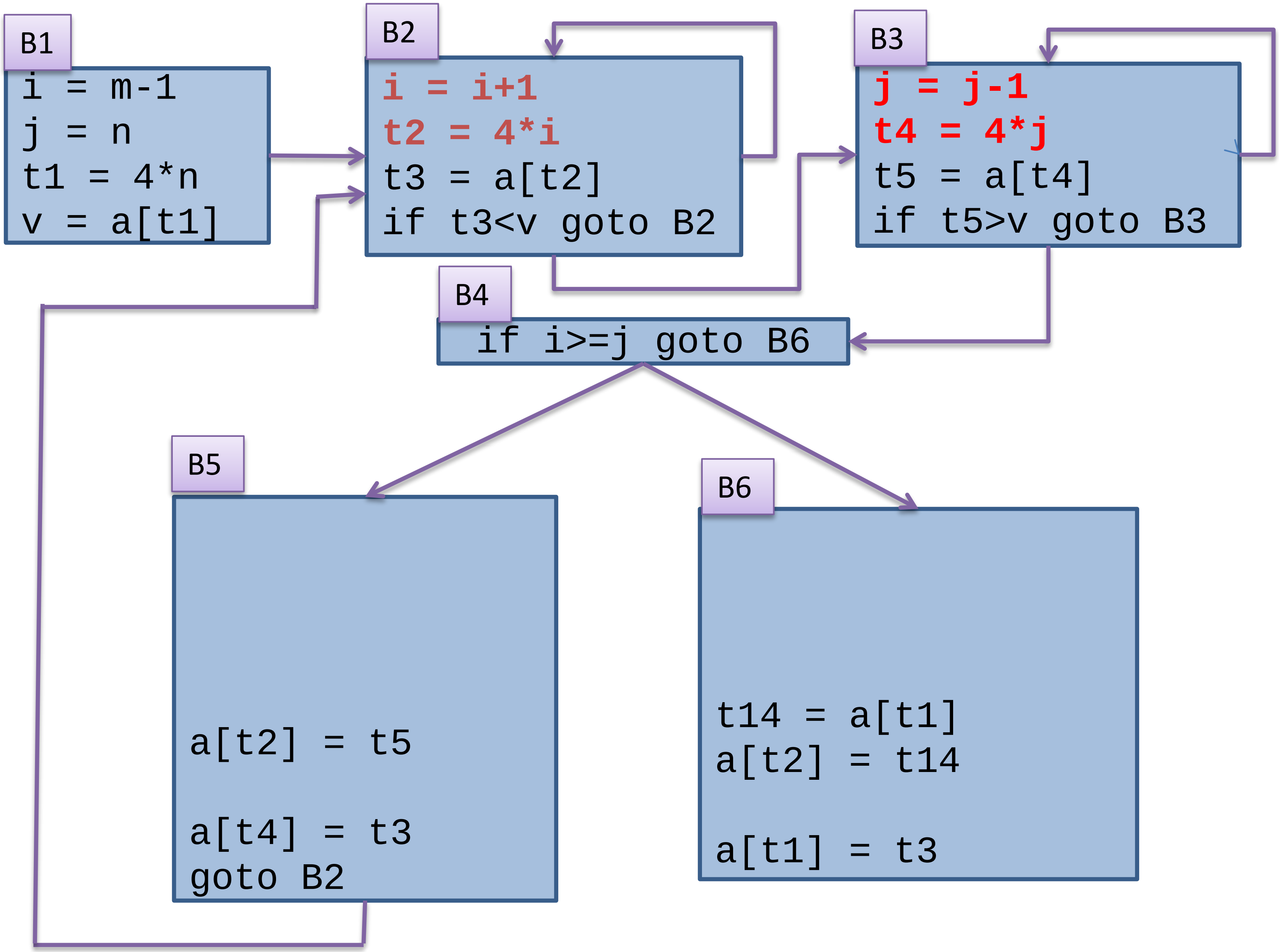
Copy Propagation



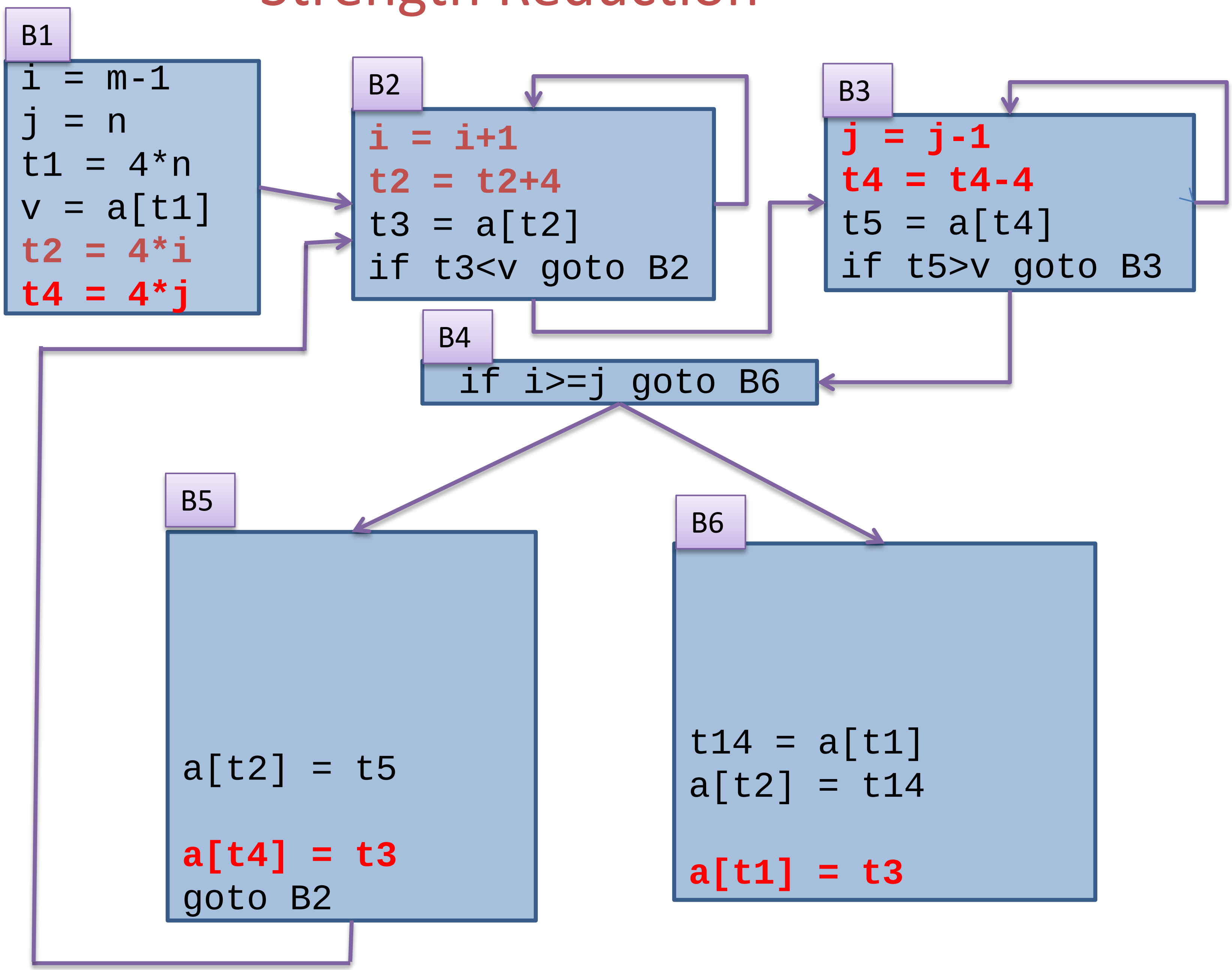
Dead Code Elimination



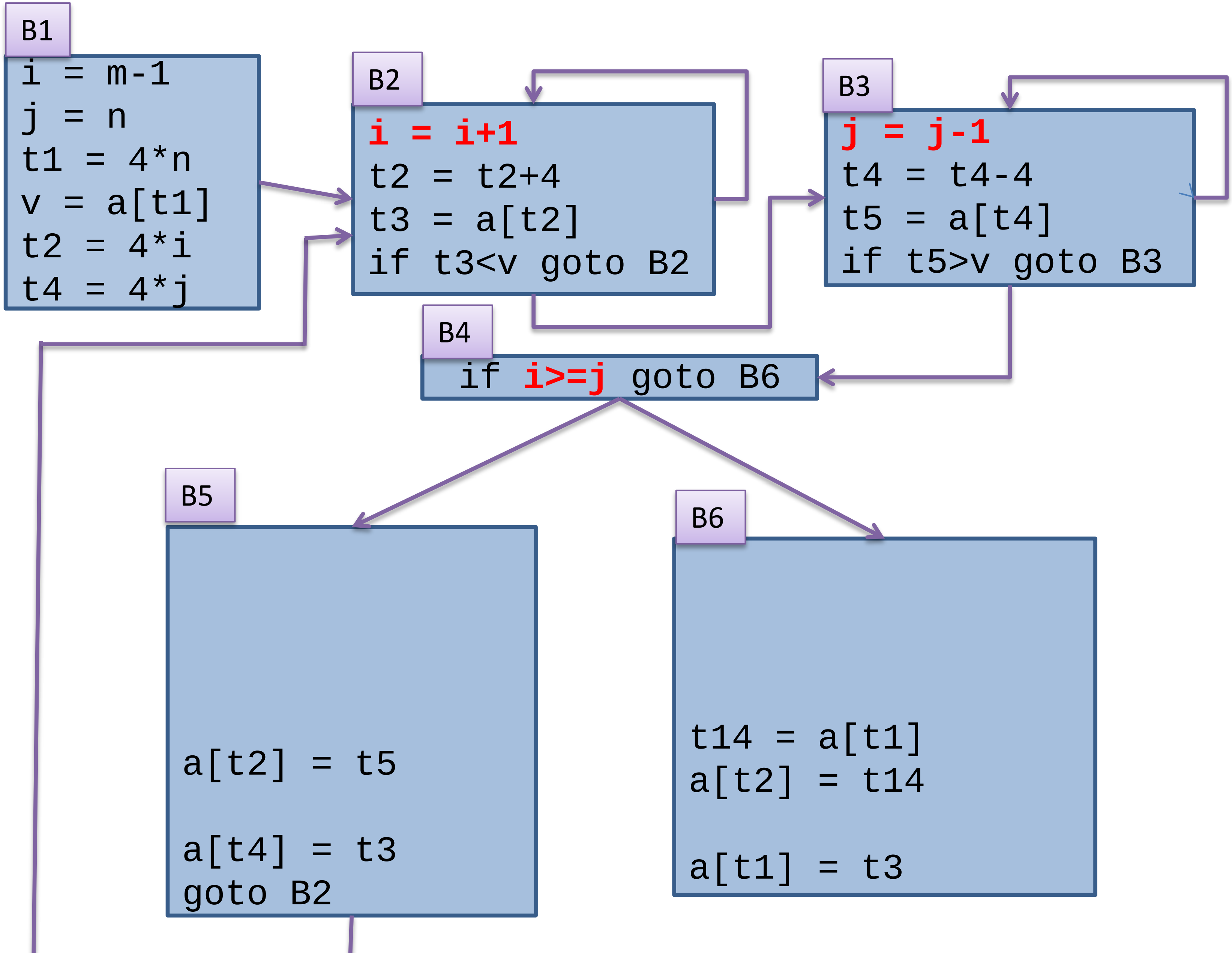
Strength Reduction



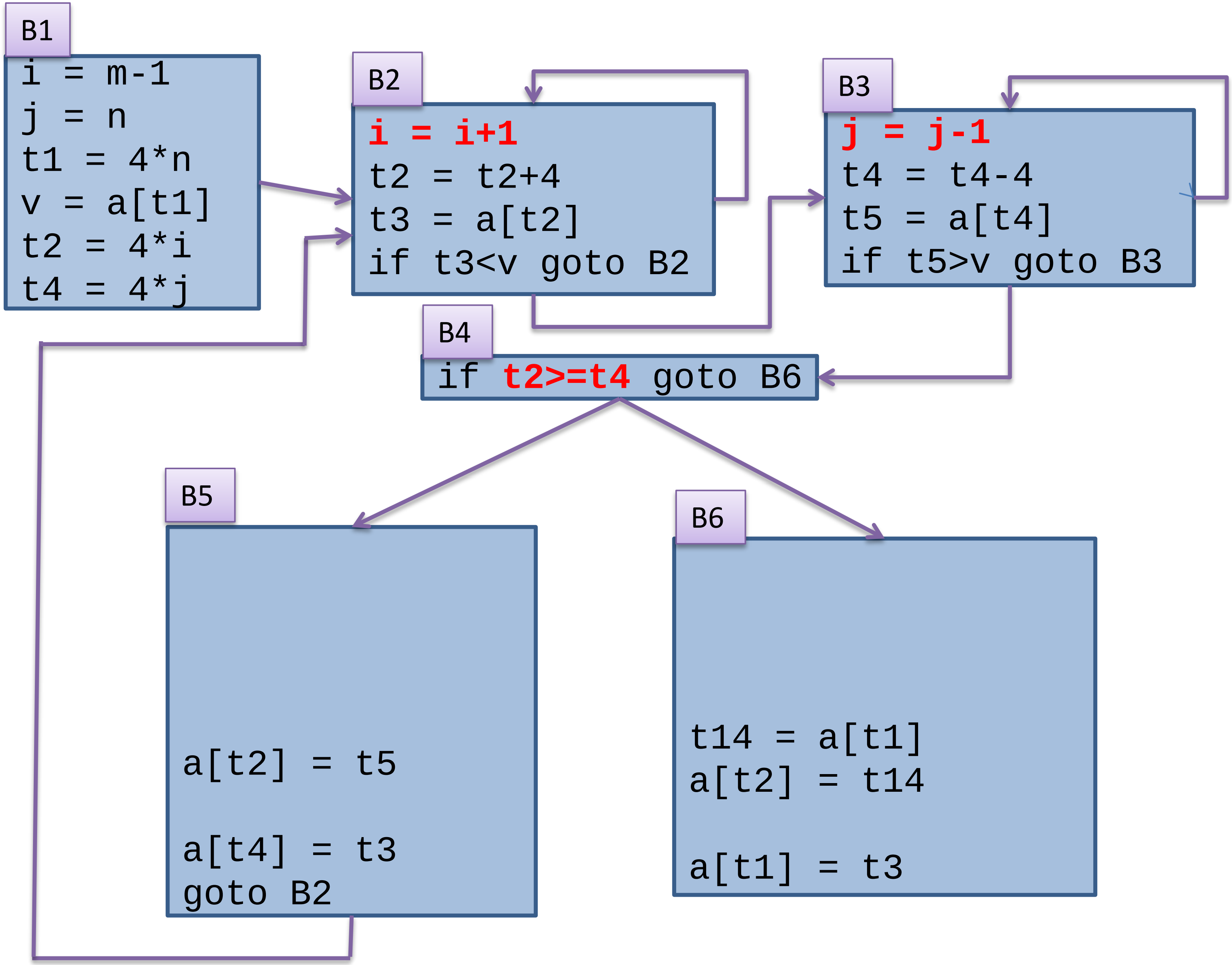
Strength Reduction



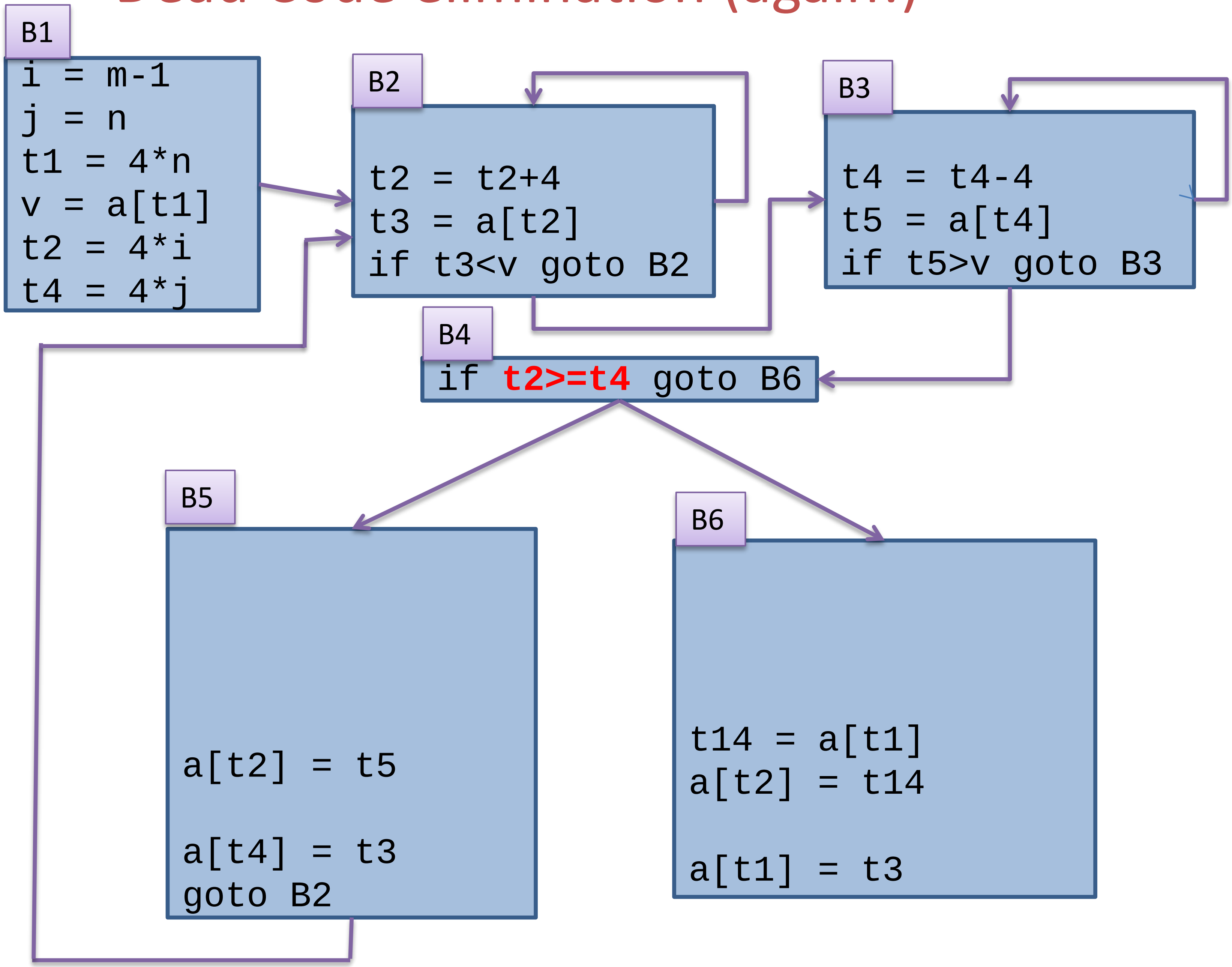
Induction Variable elimination



Induction Variable elimination



Dead Code elimination (again!)



Benefits

Block	# Stmt before Optimizations	# Stmt after Optimizations
B1	4	**6**
B2	4	**3**
B3	4	**3**
B4	1	**1**
B5	9	**3**
B6	8	**3**

Benefits

Block	# Stmt before Optimizations	# Stmt after Optimizations
B1	4	6
B2	4	3
B3	4	3
B4	1	**1**
B5	9	3
B6	8	3

Assume: Unit cost for statements, 10 iterations of outer loop, 100 iterations of each inner loop

Cost of execution:

ORIGINAL: 1*4 + 100*4 + 100*4 + 10*1 + 10*9 + 1*8 = 912

OPTIMIZED: 1*6 + 100*3 + 100*3 + 10*1 + 10*3 + 1*3 = 649

Machine Dependent Optimizations

Peephole Optimization

Peephole Optimization

- target code often contains redundant instructions and suboptimal constructs

Peephole Optimization

- target code often contains redundant instructions and suboptimal constructs

- examine a short sequence of target instruction (peephole) and replace by a shorter or faster sequence

Peephole Optimization

- target code often contains redundant instructions and suboptimal constructs
- examine a short sequence of target instruction (peephole) and replace by a shorter or faster sequence
- peephole is a small moving window on the target systems

Peephole optimization examples…

Redundant loads and stores

- Consider the code sequence

Move R_0, a

Move a, R_0

Peephole optimization examples…

Redundant loads and stores

- Consider the code sequence

Move R_0, a

Move a, R_0

- Instruction 2 can always be removed if it does not have a label.

Peephole optimization examples…

Unreachable code

- Consider following code sequence

```
#define debug 0
if (debug) {
        print debugging info
}
```

Peephole optimization examples…

Unreachable code

- Consider following code sequence

```
#define debug 0
if (debug) {
        print debugging info
}
```

this may be translated as

```
        if debug == 1 goto L1
        goto L2
L1: print debugging info
L2:
```

Peephole optimization examples…

Unreachable code

- Consider following code sequence

```
#define debug 0
if (debug) {
        print debugging info
}
```

this may be translated as

```
        if debug == 1 goto L1
        goto L2
L1: print debugging info
L2:
```

Eliminate jumps

```
        if debug != 1 goto L2
                print debugging information
L2:
```

Unreachable code example …

Unreachable code example …

constant propagation

```
        if 0 <> 1 goto L2
            print debugging information
L2:
```

Unreachable code example …

constant propagation

```
        if 0 <> 1 goto L2
           print debugging information
L2:
```

Evaluate boolean expression. Since if condition is always true the code becomes

```
        goto L2
        print debugging information
L2:
```

Unreachable code example …

constant propagation

```
        if 0 <> 1 goto L2
            print debugging information
L2:
```

Evaluate boolean expression. Since if condition is always true the code becomes

```
        goto L2
        print debugging information
L2:
```

The print statement is now unreachable. Therefore, the code becomes

```
L2:
```

Peephole optimization examples…

- flow of control: replace jump over jumps

```
     goto L1                         goto L2
                   by                     …
     …                                    …
                                  L1: goto L2
     …
L1 : goto L2
```

Peephole optimization examples…

- flow of control: replace jump over jumps

```
     goto L1                    goto L2
     …           by                ...
     …                             ...
L1 : goto L2                L1: goto L2
```

- Simplify algebraic expressions

  remove x := x+0 or x:=x*1

Peephole optimization examples…

Peephole optimization examples…

- Strength reduction
  - Replace X^2 by X*X
  - Replace multiplication by left shift
  - Replace division by right shift

Peephole optimization examples…

- Strength reduction
  - Replace X^2 by X*X
  - Replace multiplication by left shift
  - Replace division by right shift

- Use faster machine instructions

  replace Add #1,R

  by Inc R

Course Logistics

Proposed Evaluation

Assignments	**5%-10%**
Course Project	**30%-40%**
(Proposal	**5%)**
(Report	**15%)**
(Presentation	**15%)**
Mid semester exam	**10%-20%**
End semester exam	**25%-35%**
Quizzes/Class Participation	**5%**

Project

Project

- Major part of the course

Project

- Major part of the course
- You need to implement some non-trivial analysis/optimization using one of the open source infrastructure
  - For e.g., some paper published in last 10 years

Project

- Major part of the course
- You need to implement some non-trivial analysis/optimization using one of the open source infrastructure
  - For e.g., some paper published in last 10 years
- You are encouraged to suggest your own projects

Project

- Major part of the course
- You need to implement some non-trivial analysis/optimization using one of the open source infrastructure
  - For e.g., some paper published in last 10 years
- You are encouraged to suggest your own projects
- **Bonus** marks for publishable results

Project

- Major part of the course
- You need to implement some non-trivial analysis/optimization using one of the open source infrastructure
  - For e.g., some paper published in last 10 years
- You are encouraged to suggest your own projects
- **Bonus** marks for publishable results
- Individual OR Group of **2**

Assignment #1

Assignment #1

- Select one of the compiler infrastructure mentioned on the course webpage and
  a) Download it
  b) Build it
  c) Submit a report
  d) one page about the infrastructure, and the optimizations present in it.
  e) one page about the most interesting optimization found, with example

Assignment #1

- You can try more than one tool, even something not mentioned on the webpage.
- But submit report for only **one**.
  - Preferably the one you plan to use for your project.

Assignment #1

- You can try more than one tool, even something not mentioned on the webpage.
- But submit report for only **one**.
  - Preferably the one you plan to use for your project.
- DEADLINE: July 30th, End of Day (before Midnight)
- See course website for submission details (TBD)

https://www.cse.iitb.ac.in/~karkare/cs618/