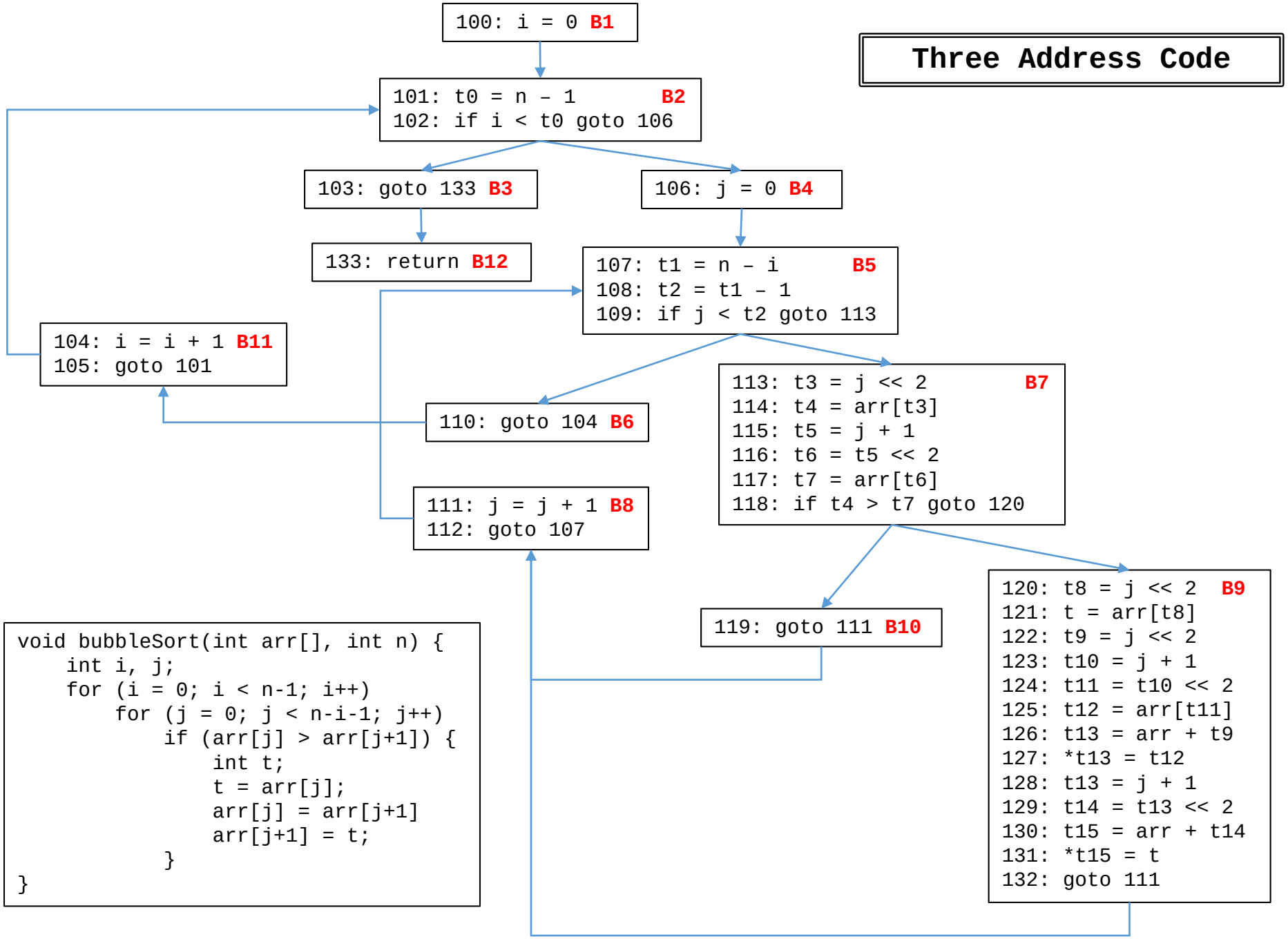
Solution for Tutorial 7

Optimization and Register Allocation

The Case of Bubble Sort Code

Three Address Code

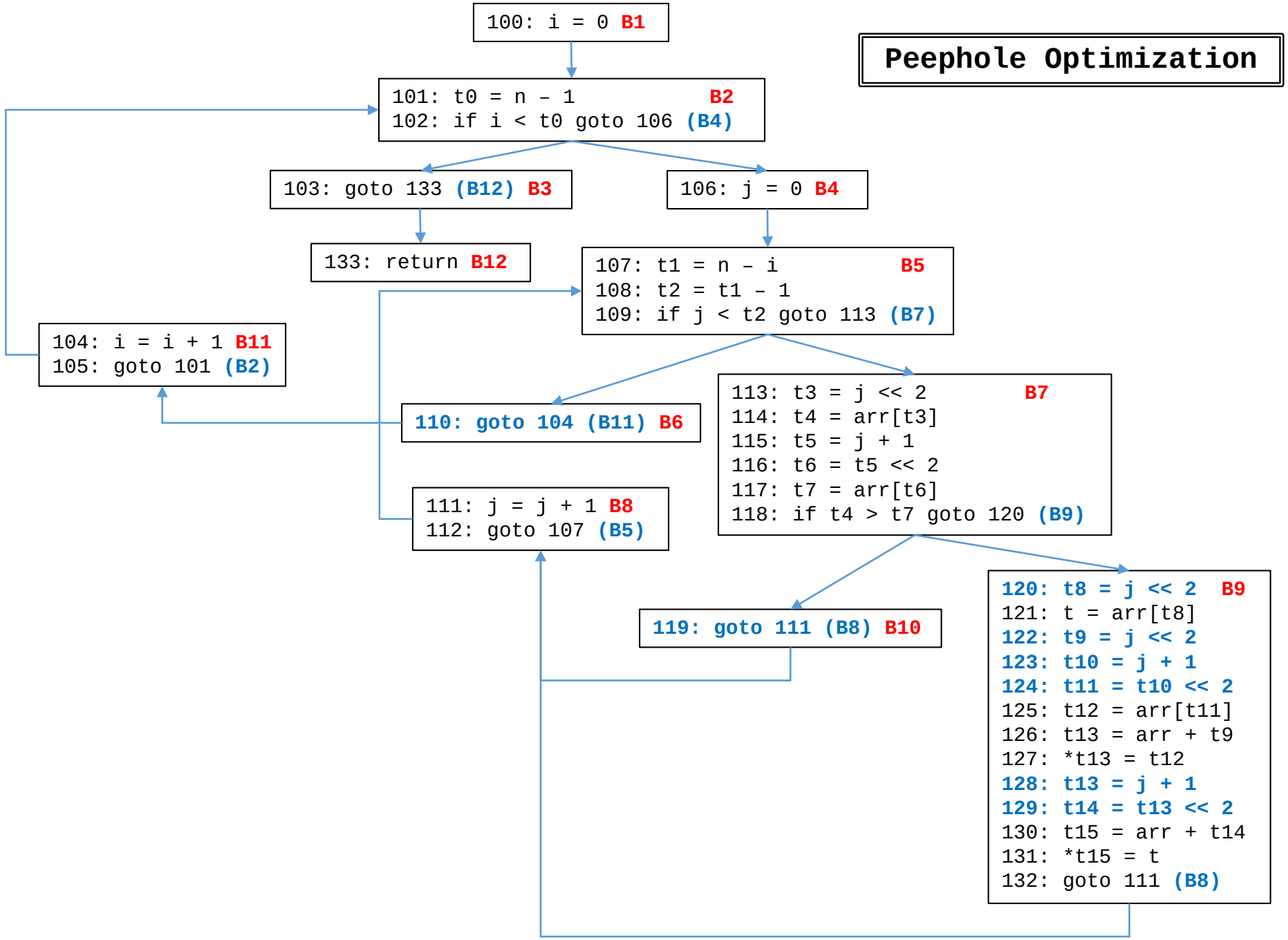
```
100*: i = 0
101*: t0 = n - 1
102 : if i < t0 goto 106
103*: goto 133
104*: i = i + 1
105 : goto 101
106*: j = 0
107*: t1 = n - i
108 : t2 = t1 - 1
109 : if j < t2 goto 113
110*: goto 104
111*: j = j + 1
112 : goto 107
113*: t3 = j << 2
114 : t4 = arr[t3]
115 : t5 = j + 1
116 : t6 = t5 << 2
117 : t7 = arr[t6]
118 : if t4 > t7 goto 120
119*: goto 111
120*: t8 = j << 2
121 : t = arr[t8]
122 : t9 = j << 2
123 : t10 = j + 1
124 : t11 = t10 << 2
125 : t12 = arr[t11]
126 : t13 = arr + t9
127 : *t13 = t12
128 : t13 = j + 1
129 : t14 = t13 << 2
130 : t15 = arr + t14
131 : *t15 = t
132 : goto 111
133*: return
```



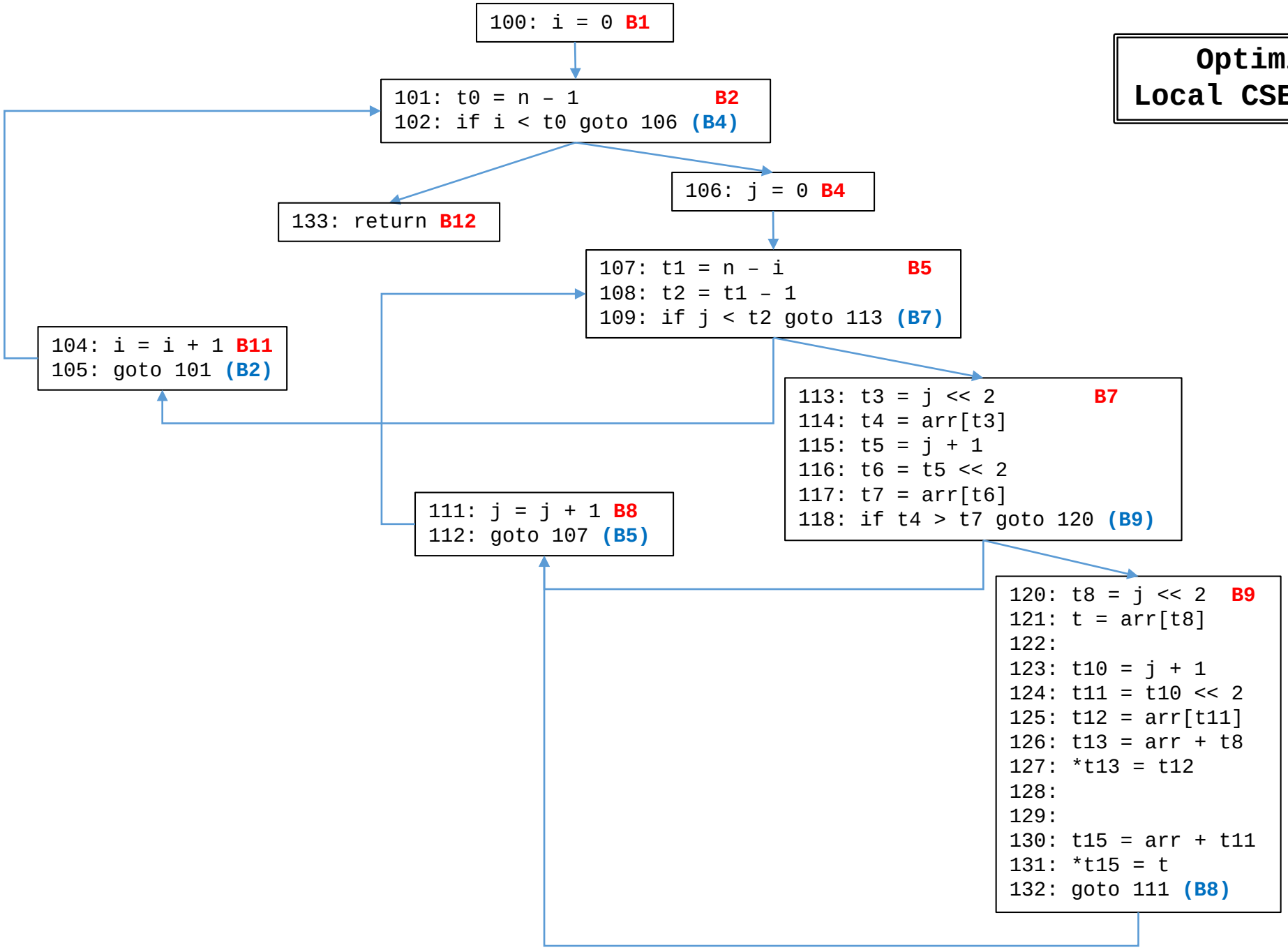

```
void bubbleSort(int arr[], int n) {
    int i, j;
    for (i = 0; i < n-1; i++)
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1]) {
                int t;
                t = arr[j];
                arr[j] = arr[j+1]
                arr[j+1] = t;
            }
}
```

Peephole Optimization

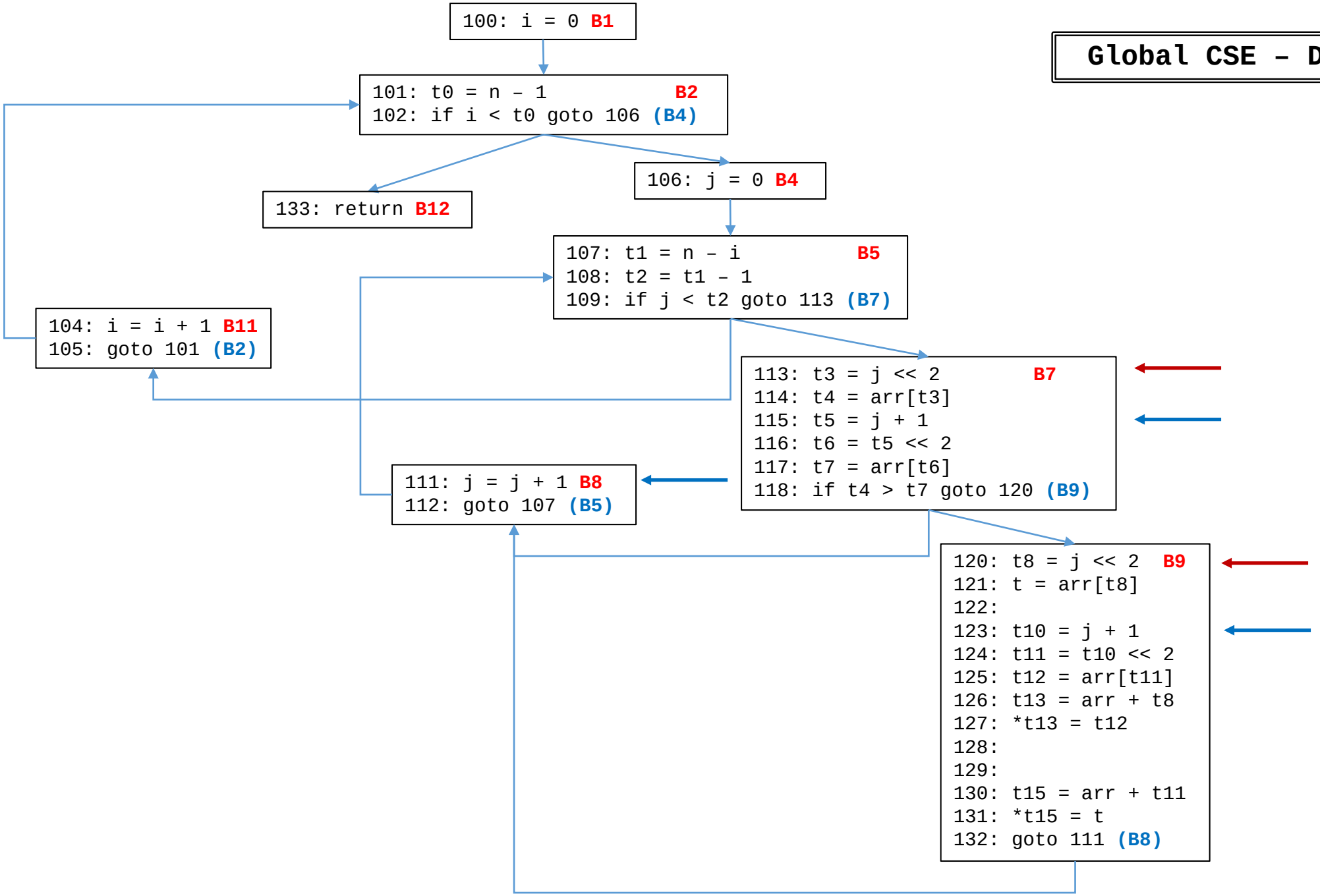
```
100*: i = 0
101*: t0 = n - 1
102 : if i < t0 goto 106
103*: goto 133
104*: i = i + 1
105 : goto 101
106*: j = 0
107*: t1 = n - i
108 : t2 = t1 - 1
109 : if j < t2 goto 113
110*: goto 104
111*: j = j + 1
112 : goto 107
113*: t3 = j << 2
114 : t4 = arr[t3]
115 : t5 = j + 1
116 : t6 = t5 << 2
117 : t7 = arr[t6]
118 : if t4 > t7 goto 120
119*: goto 111
120*: t8 = j << 2
121 : t = arr[t8]
122 : t9 = j << 2
123 : t10 = j + 1
124 : t11 = t10 << 2
125 : t12 = arr[t11]
126 : t13 = arr + t9
127 : *t13 = t12
128 : t13 = j + 1
129 : t14 = t13 << 2
130 : t15 = arr + t14
131 : *t15 = t
132 : goto 111
133*: return
```



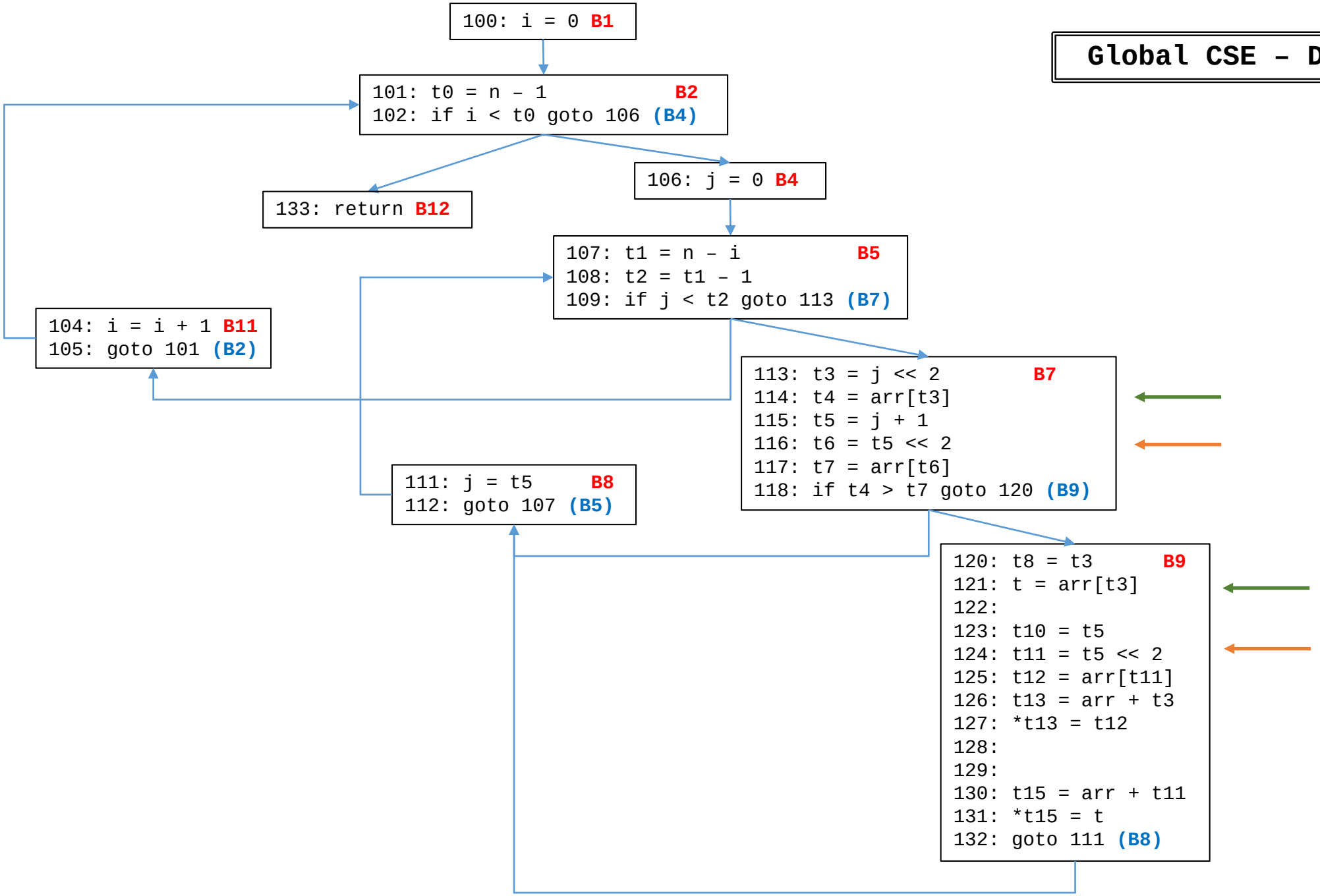
Optimize Flow
Local CSE – Block B9



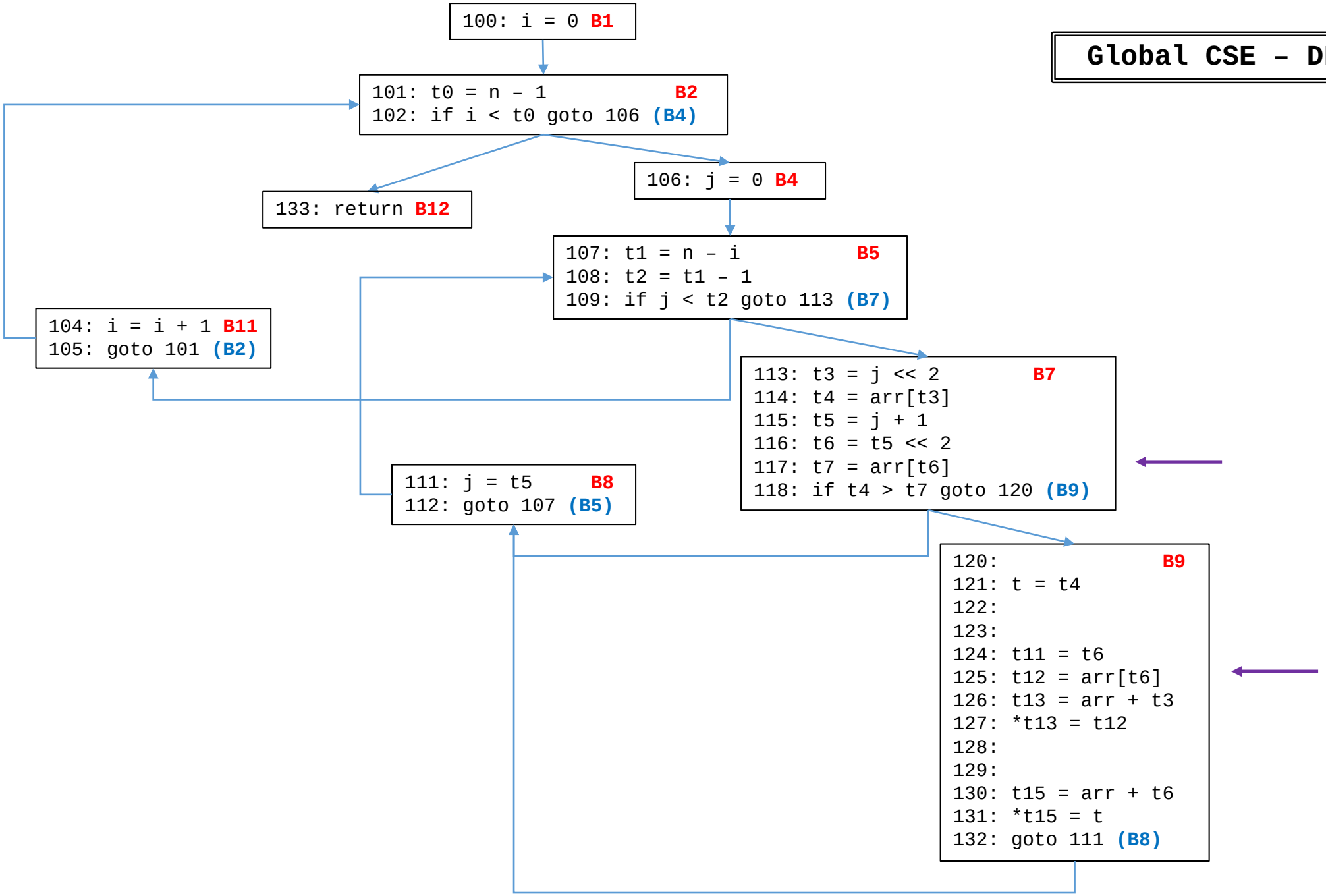
Global CSE – DFA 1



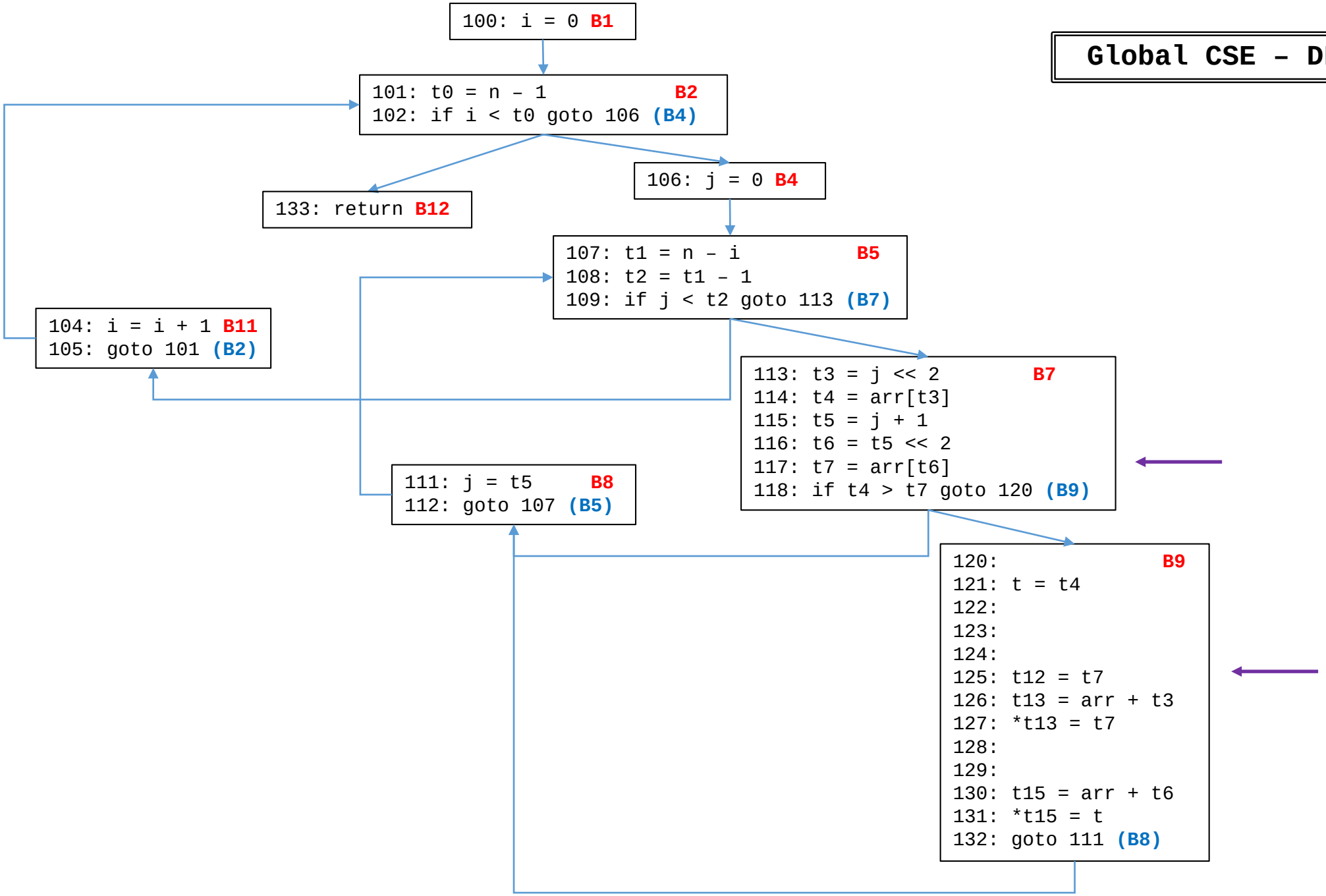
Global CSE – DFA 2



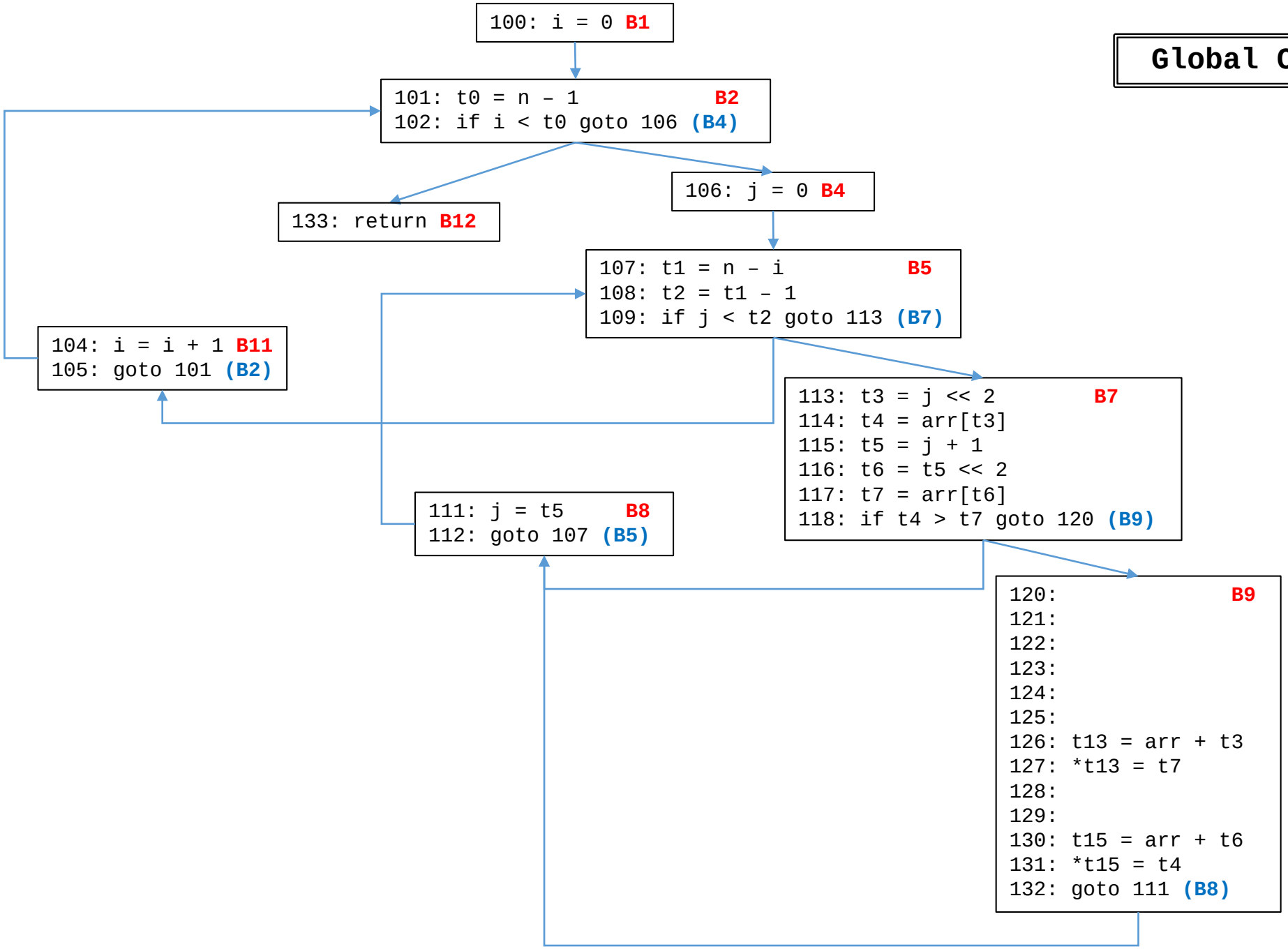
Global CSE – DFA 3



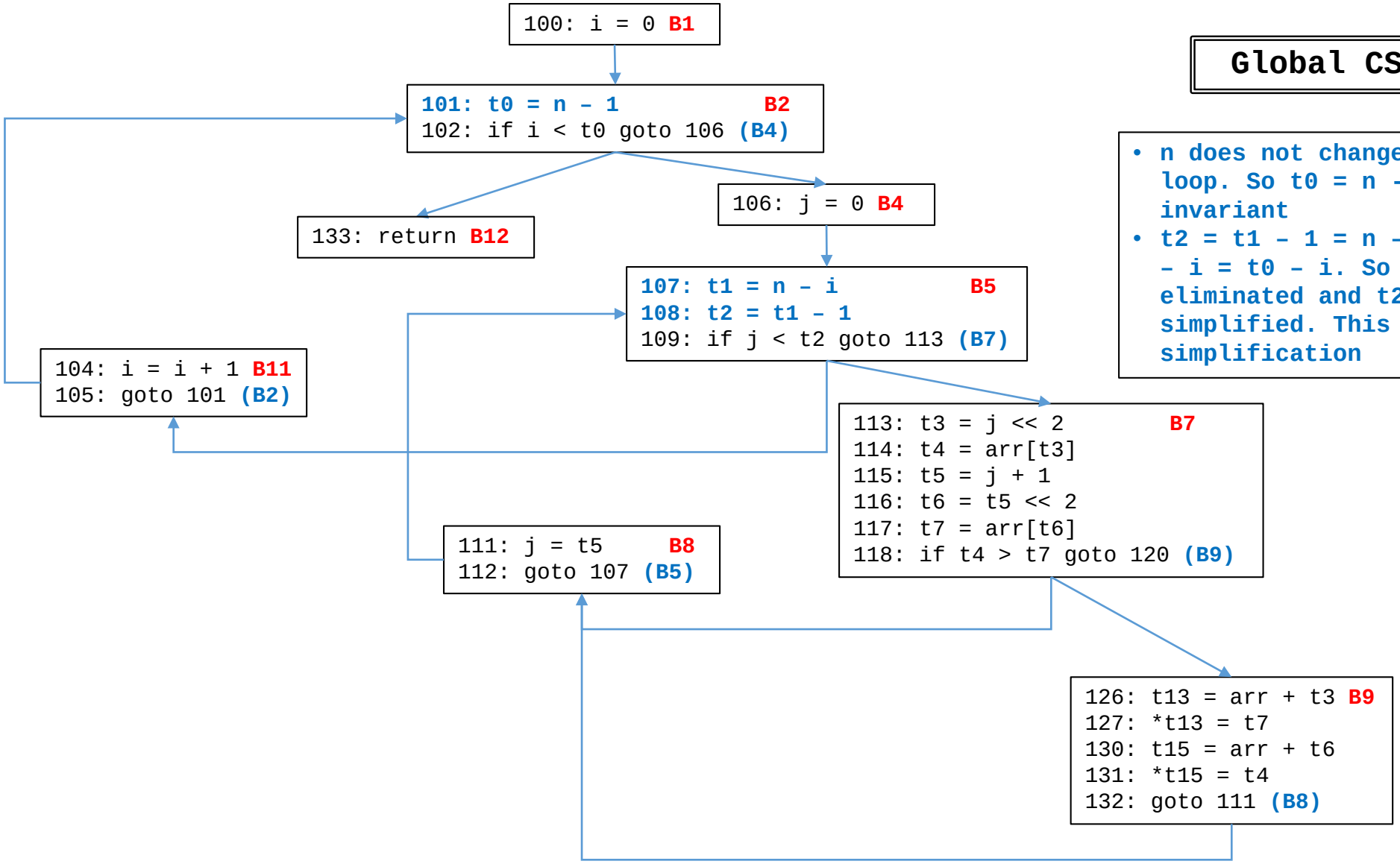
Global CSE – DFA 4



Global CSE – Clean

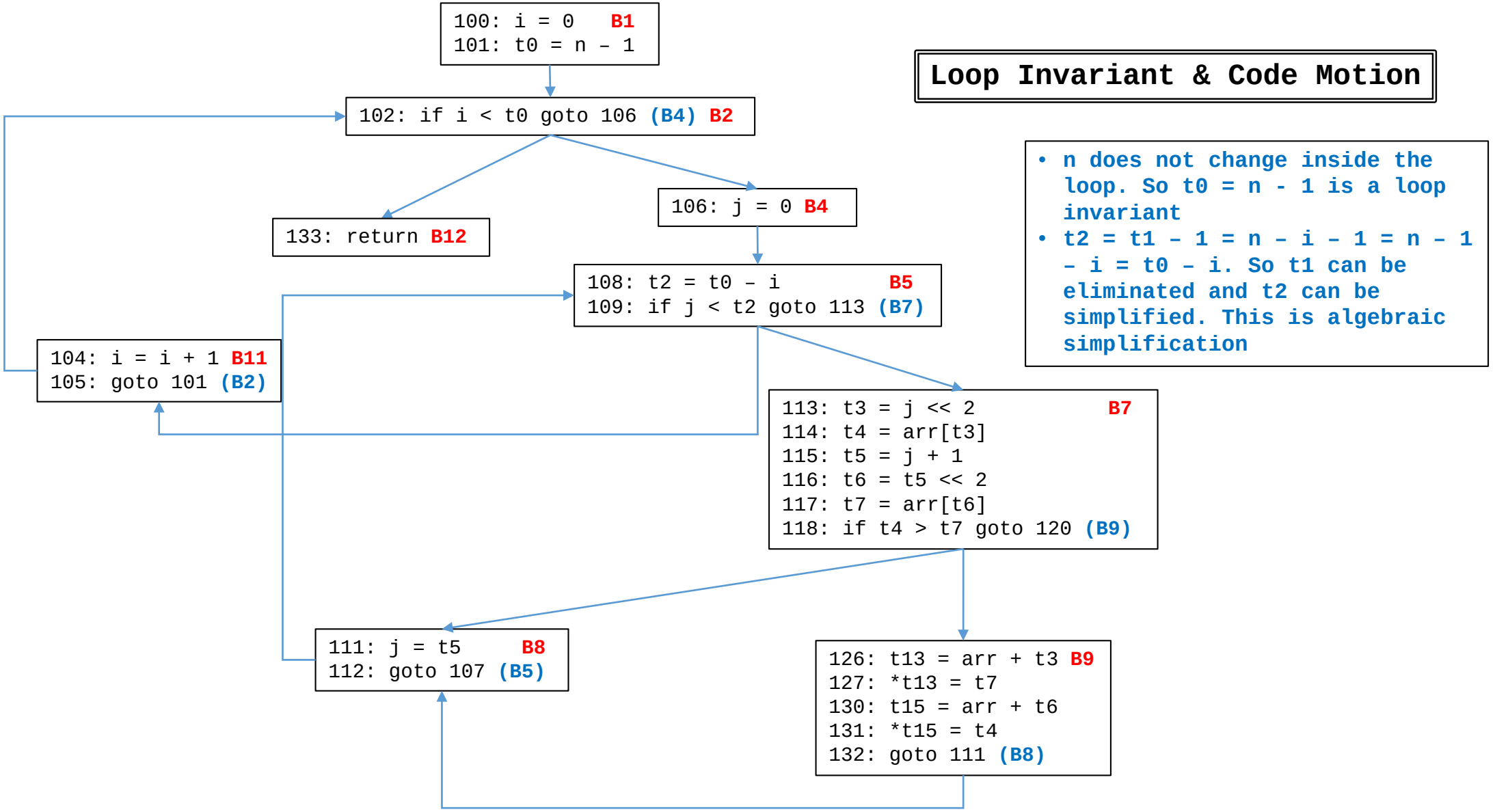


Global CSE – Clean



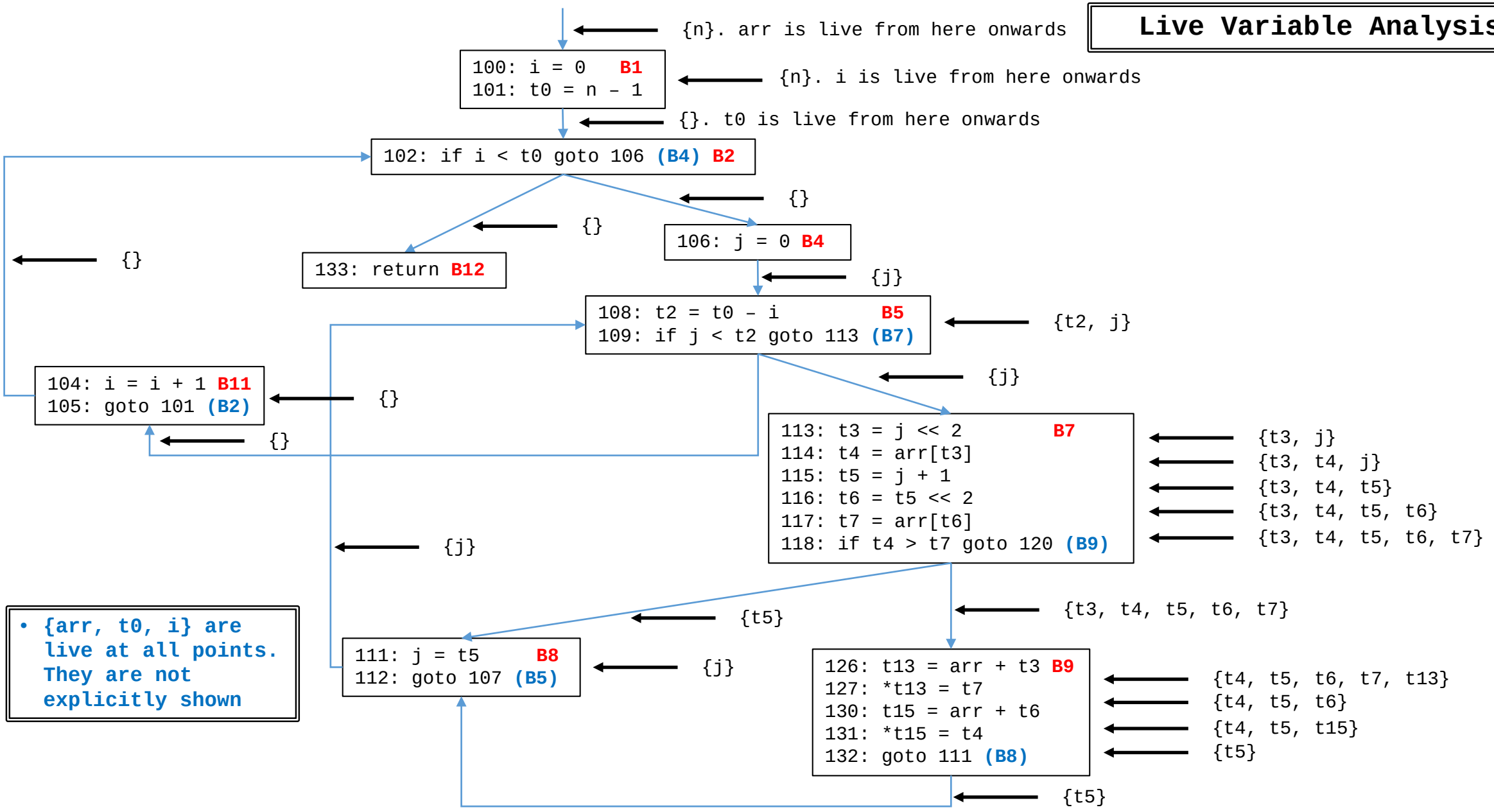
- n does not change inside the loop. So t0 = n - 1 is a loop invariant
- t2 = t1 – 1 = n – i – 1 = n – 1 – i = t0 – i. So t1 can be eliminated and t2 can be simplified. This is algebraic simplification

Loop Invariant & Code Motion

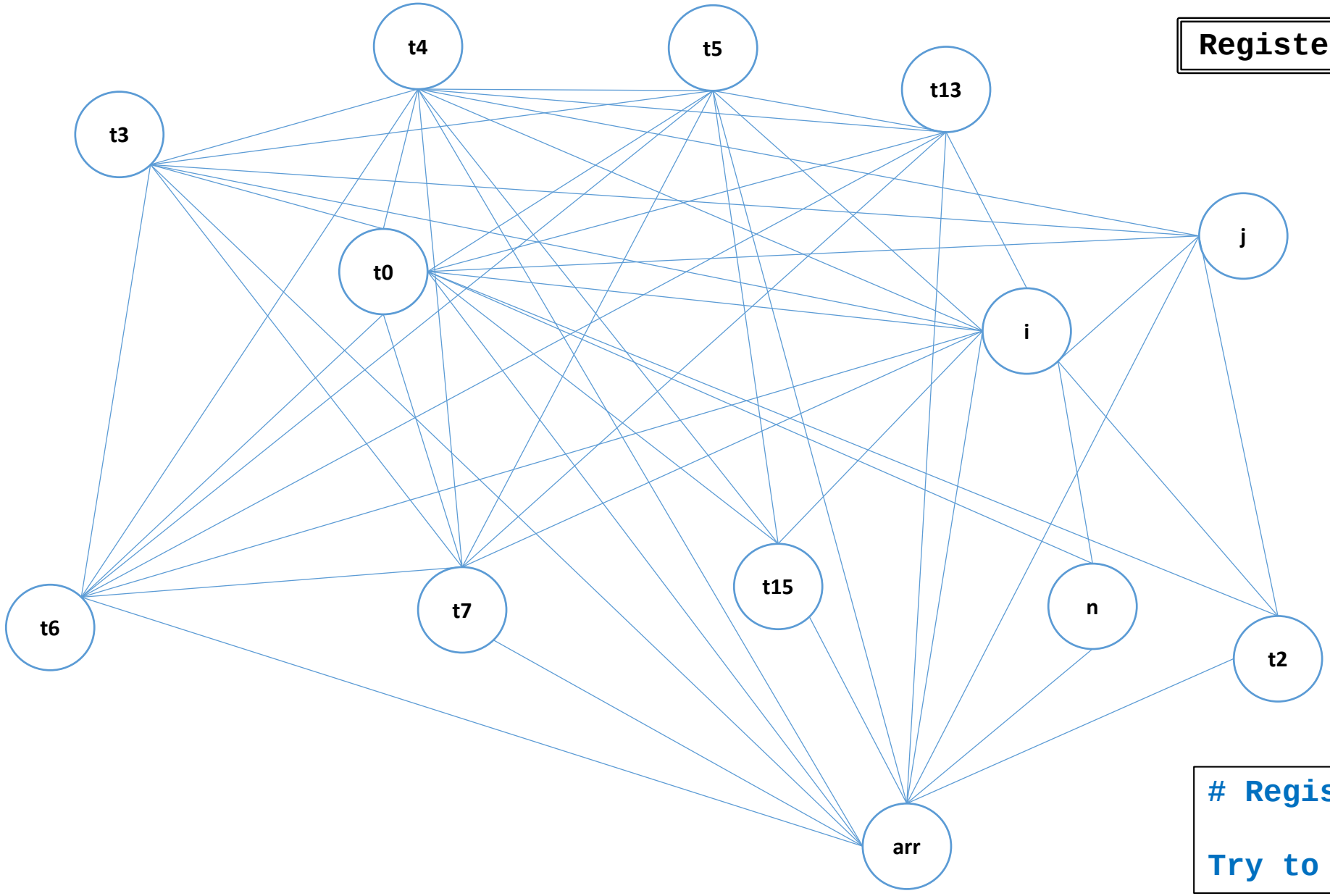


- n does not change inside the loop. So t0 = n - 1 is a loop invariant
- t2 = t1 – 1 = n – i – 1 = n – 1 – i = t0 – i. So t1 can be eliminated and t2 can be simplified. This is algebraic simplification

Live Variable Analysis



Register Interference Graph

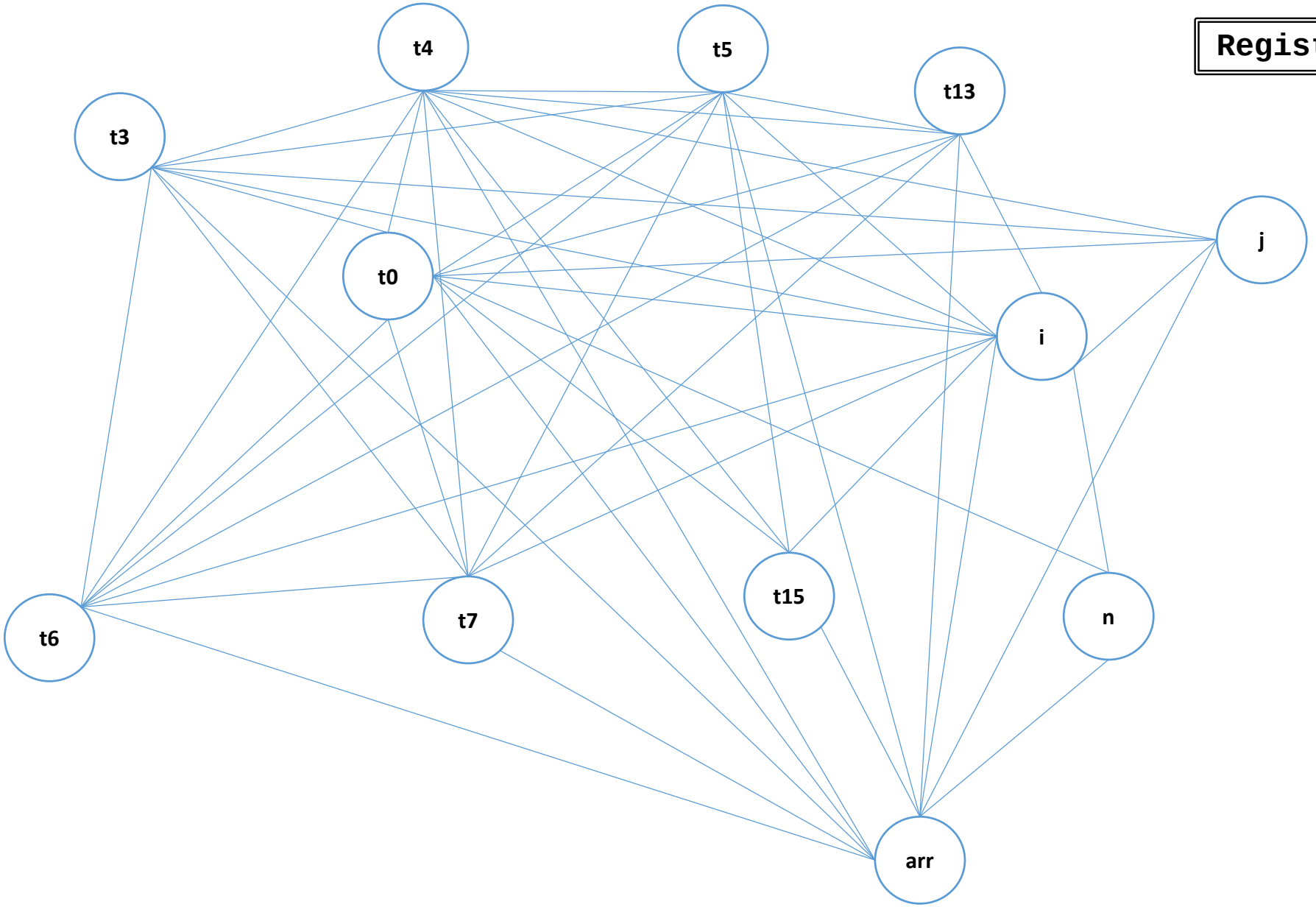


Registers k = 6

Try to color with 6 colors

Register Interference Graph

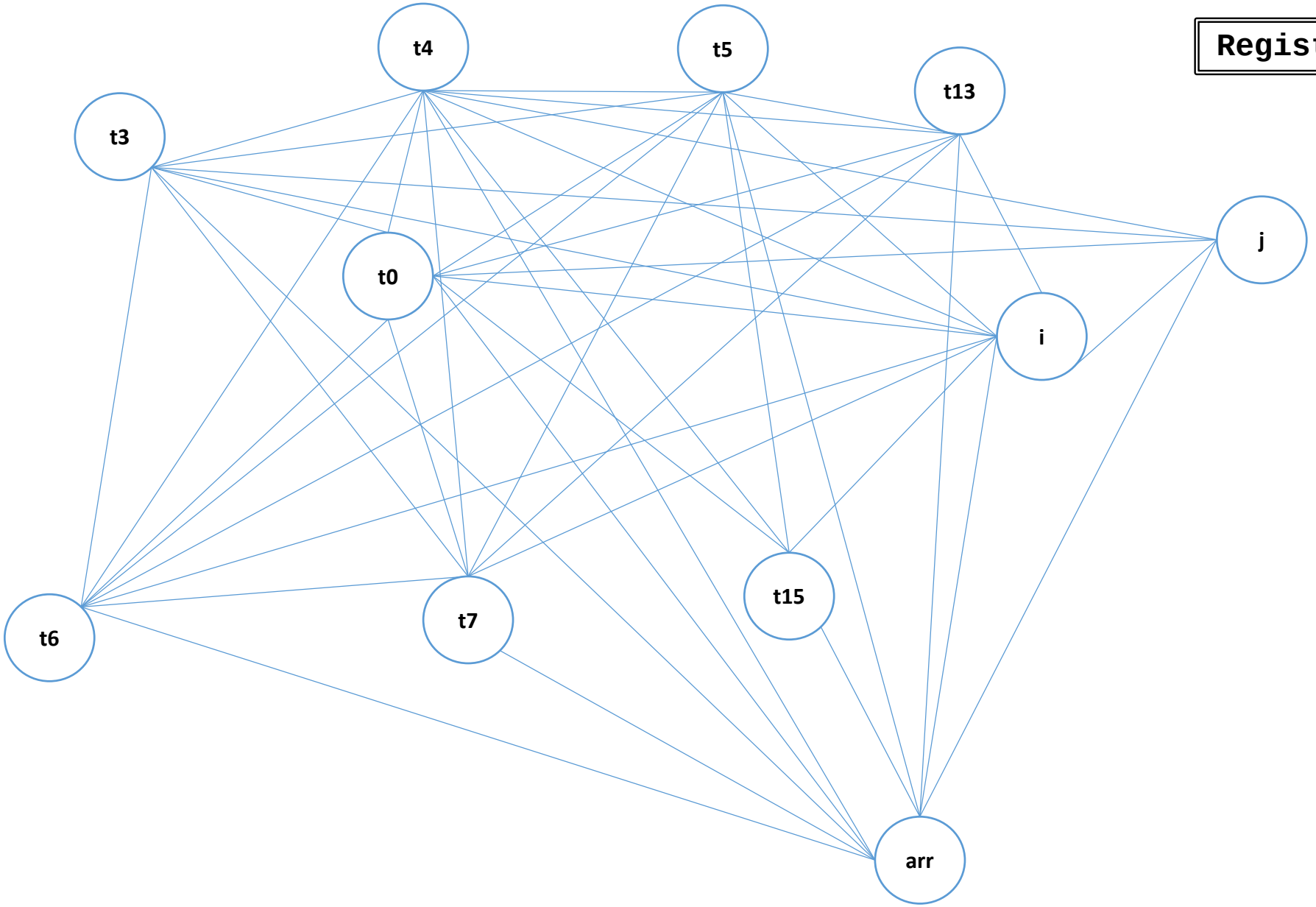
 # Registers k = 6



Stack: t2

Register Interference Graph

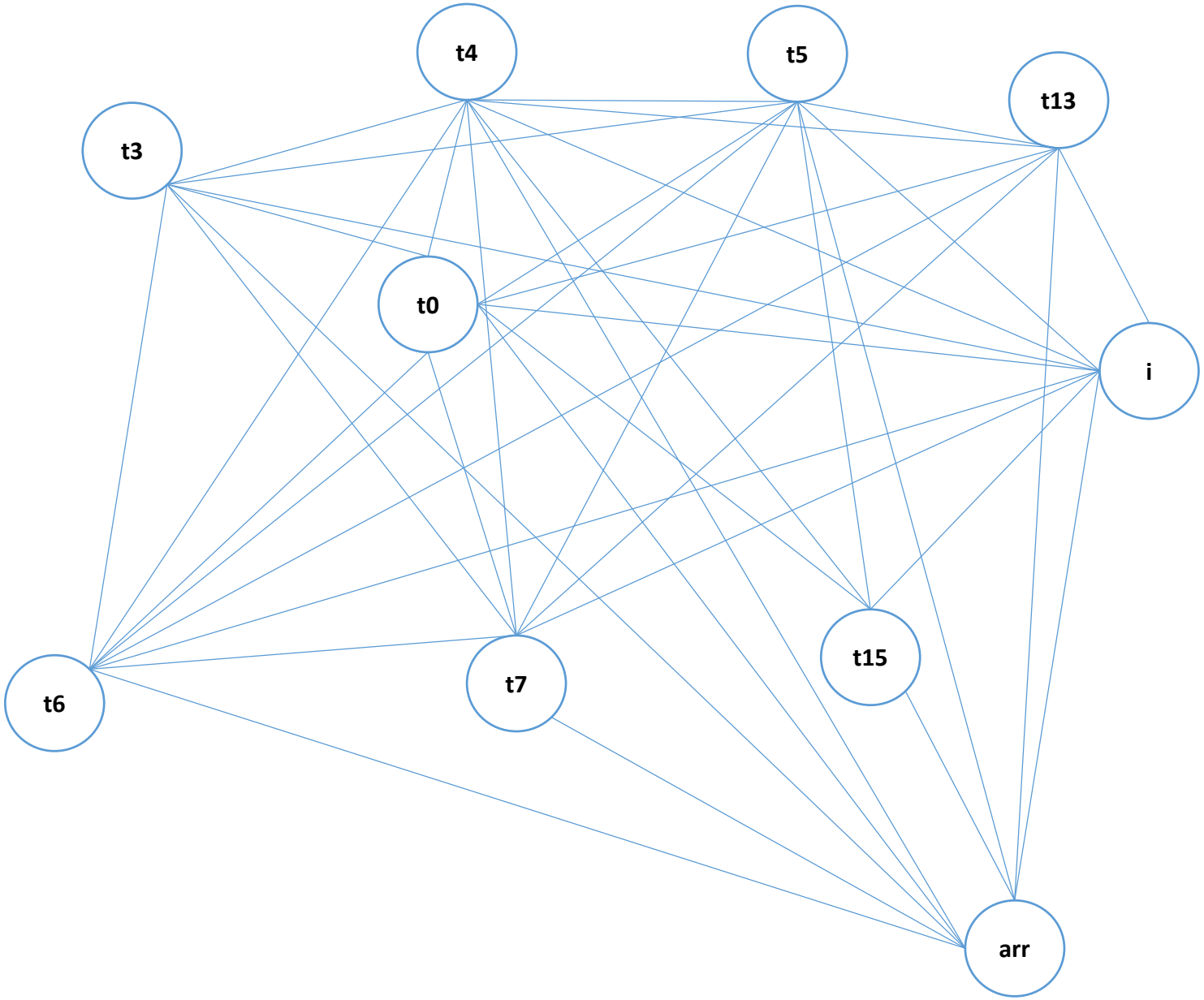
 # Registers k = 6



Stack: n, t2

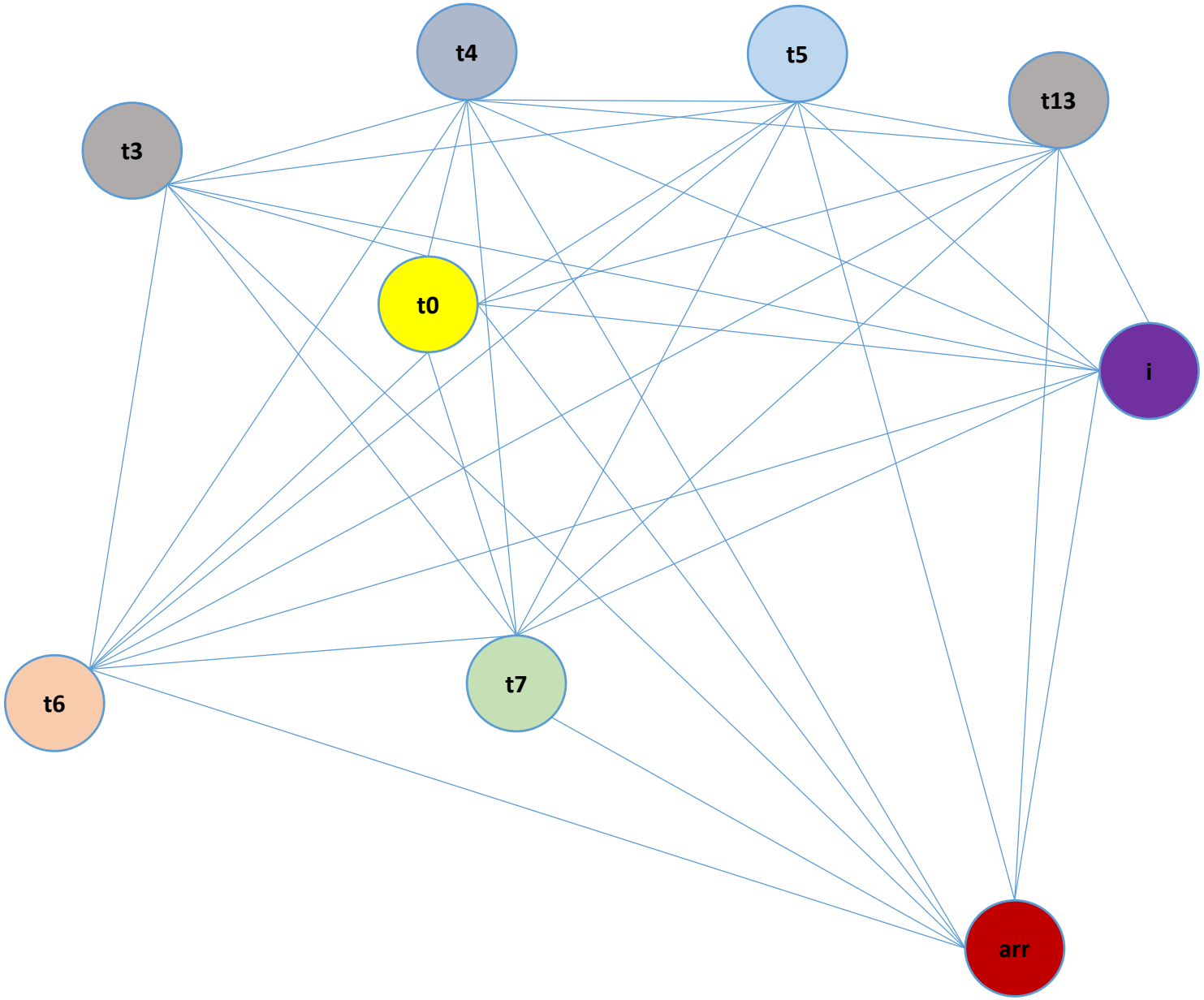
Register Interference Graph

 # Registers k = 6



Stack: j, n, t2

Register Interference Graph



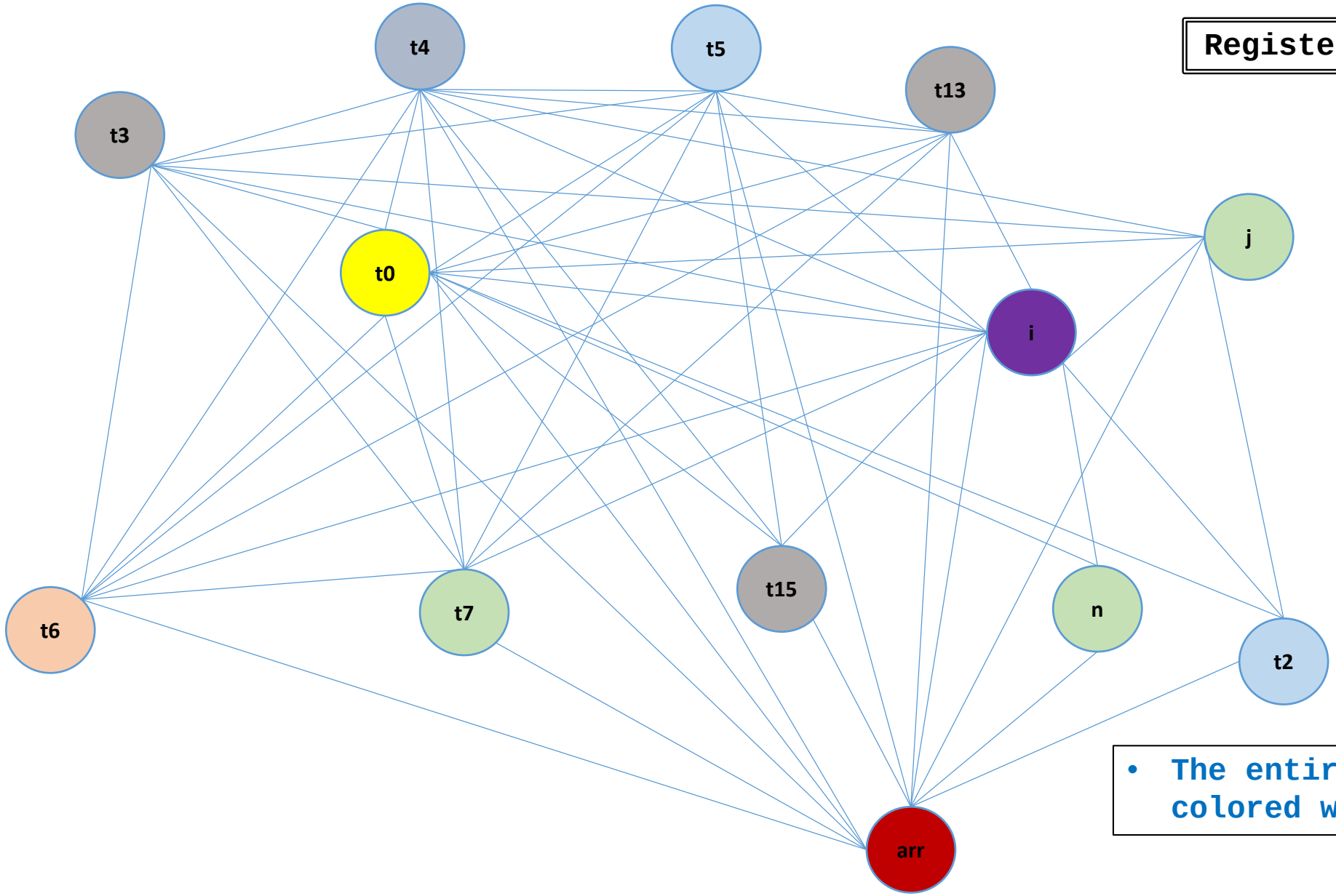
 # Registers k = 6

- No node can be removed
- We need at least 8 colors
- We will spill 2 variables

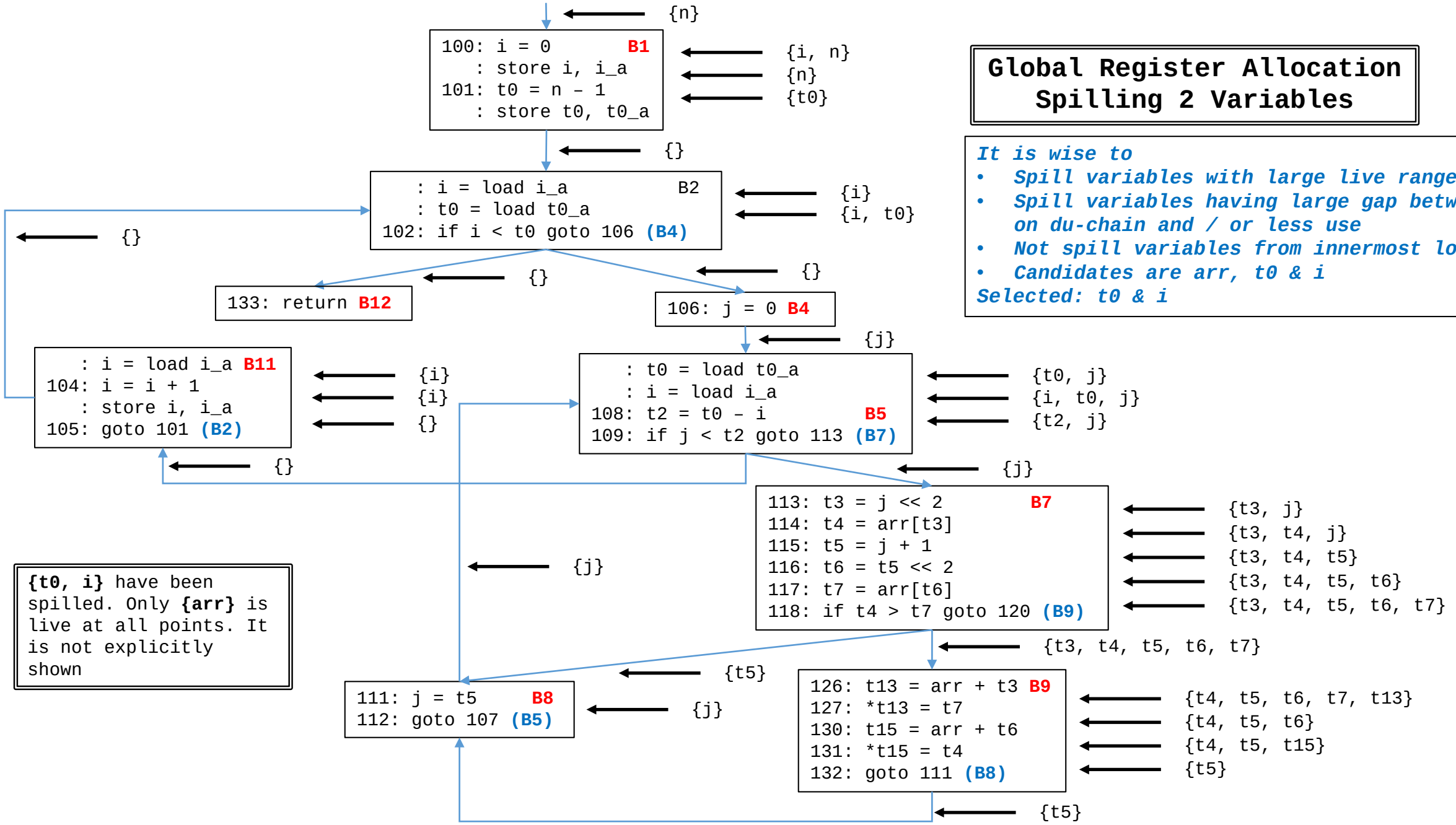
Stack: t15, j, n, t2

Register Interference Graph

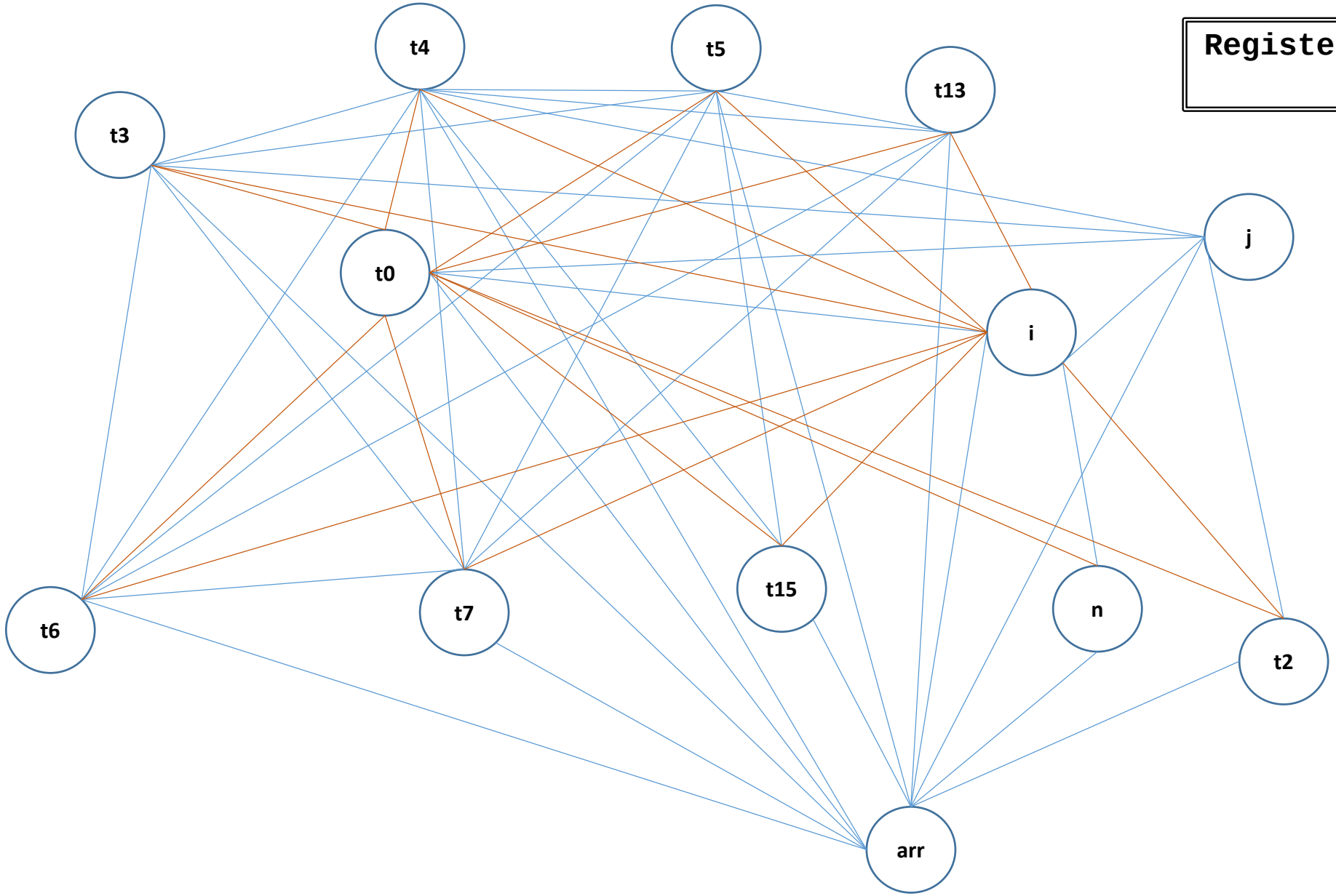
 # Registers k = 8



- **The entire graph can be colored with 8 colors**


{n}
100: i = 0 B1
: store i, i_a
101: t0 = n – 1
: store t0, t0_a
{i, n}
{n}
{t0}
{}
: i = load i_a B2
: t0 = load t0_a
102: if i < t0 goto 106 (B4)
{i}
{i, t0}
{}
{}
{}
133: return B12
106: j = 0 B4
{j}
: i = load i_a B11
104: i = i + 1
: store i, i_a
105: goto 101 (B2)
{i}
{i}
{}
{}
: t0 = load t0_a
: i = load i_a
108: t2 = t0 – i B5
109: if j < t2 goto 113 (B7)
{t0, j}
{i, t0, j}
{t2, j}
{j}
113: t3 = j << 2 B7
114: t4 = arr[t3]
115: t5 = j + 1
116: t6 = t5 << 2
117: t7 = arr[t6]
118: if t4 > t7 goto 120 (B9)
{t3, j}
{t3, t4, j}
{t3, t4, t5}
{t3, t4, t5, t6}
{t3, t4, t5, t6, t7}
{t3, t4, t5, t6, t7}
{j}
{t5}
111: j = t5 B8
112: goto 107 (B5)
{j}
126: t13 = arr + t3 B9
127: *t13 = t7
130: t15 = arr + t6
131: *t15 = t4
132: goto 111 (B8)
{t4, t5, t6, t7, t13}
{t4, t5, t6}
{t4, t5, t15}
{t5}
{t5}
{t0, i} have been spilled. Only {arr} is live at all points. It is not explicitly shown
Global Register Allocation
Spilling 2 Variables
It is wise to
• Spill variables with large live range
• Spill variables having large gap between on du-chain and / or less use
• Not spill variables from innermost loop
• Candidates are arr, t0 & i
Selected: t0 & i

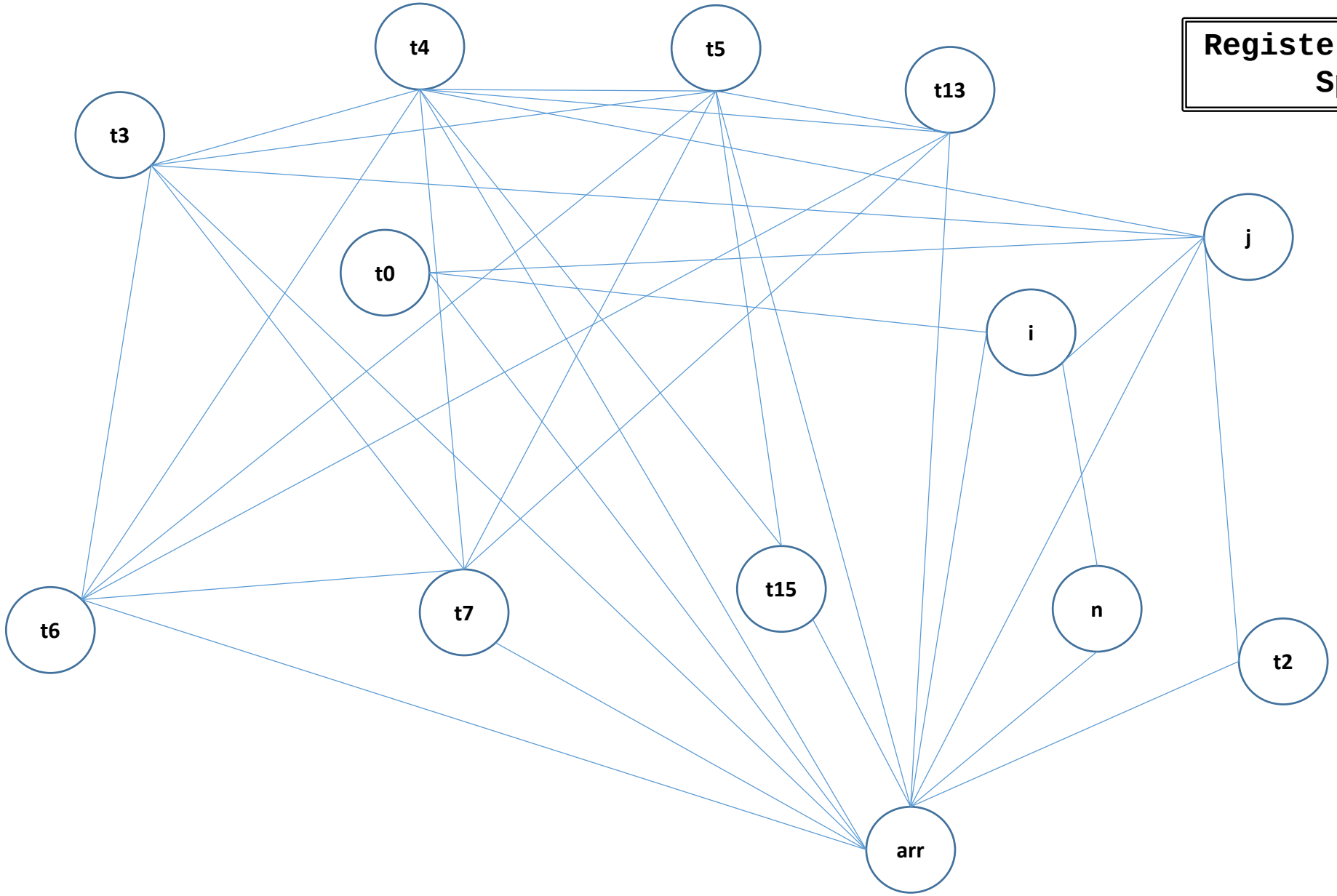
Register Interference Graph
No Spill



The red edges get removed by spilling

Register Interference Graph
Spilled: t0 & i

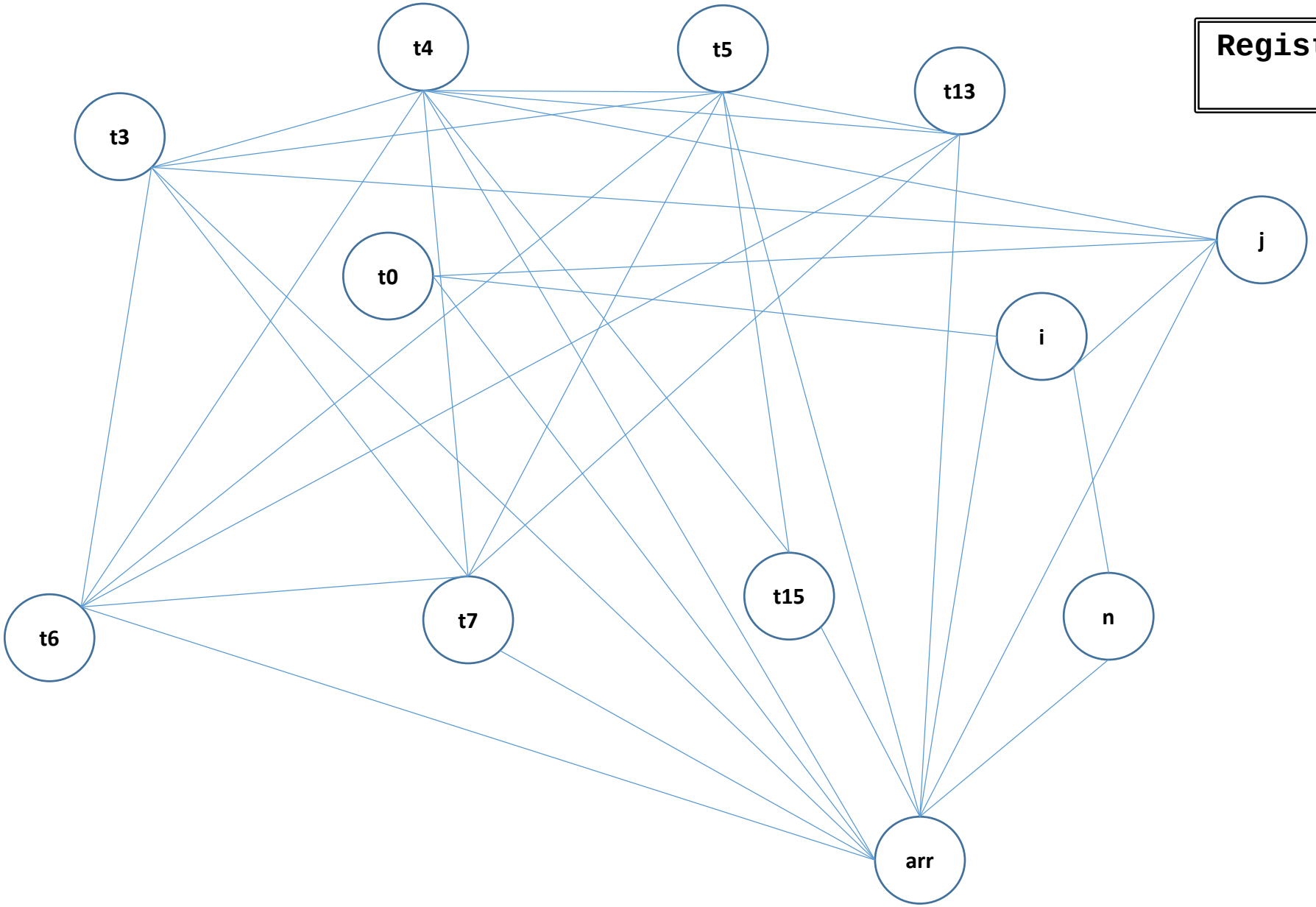
 # Registers k = 6



Stack:

Register Interference Graph
Spilled: t0 & i

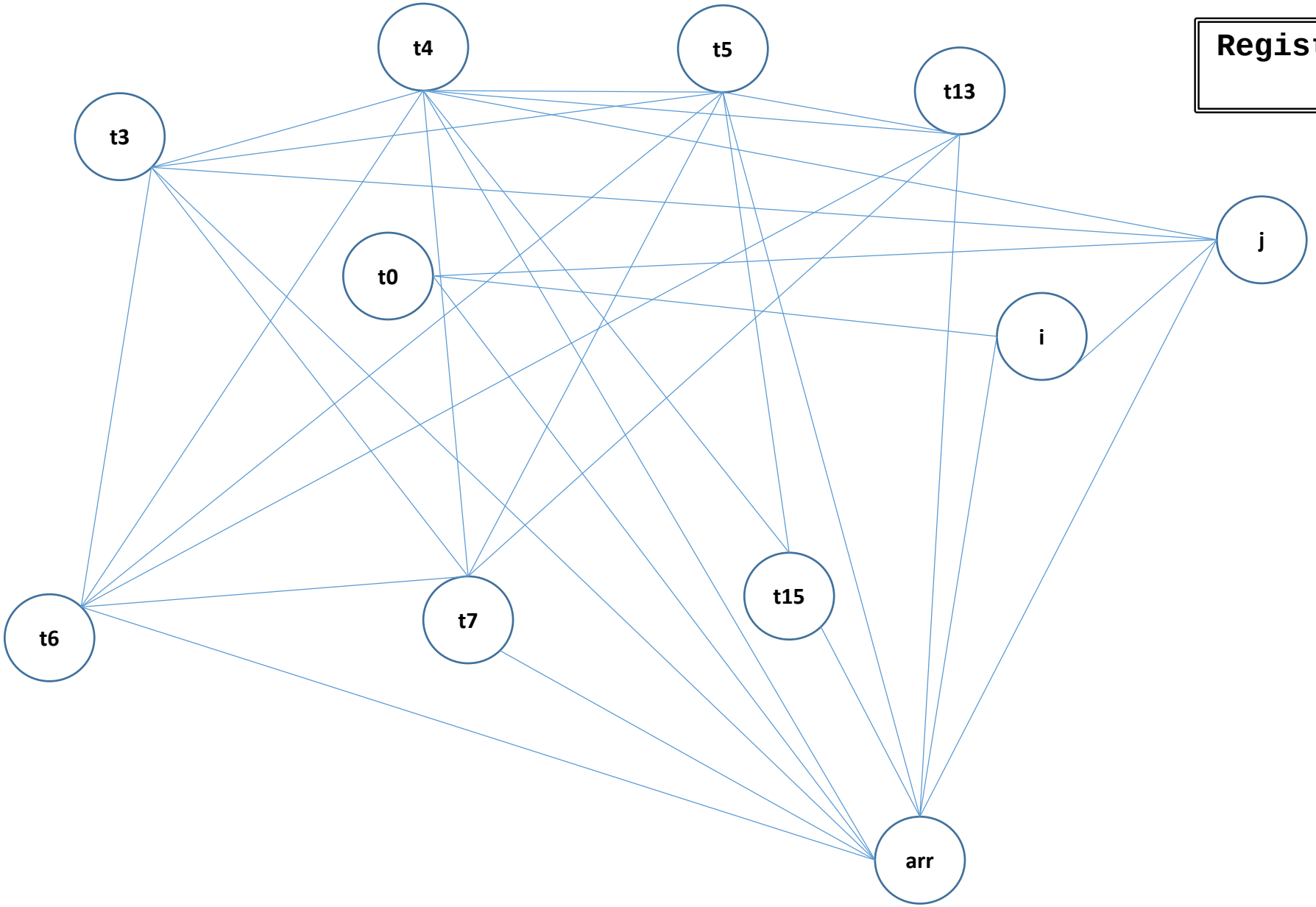
 # Registers k = 6



Stack: t2

Register Interference Graph
Spilled: t0 & i

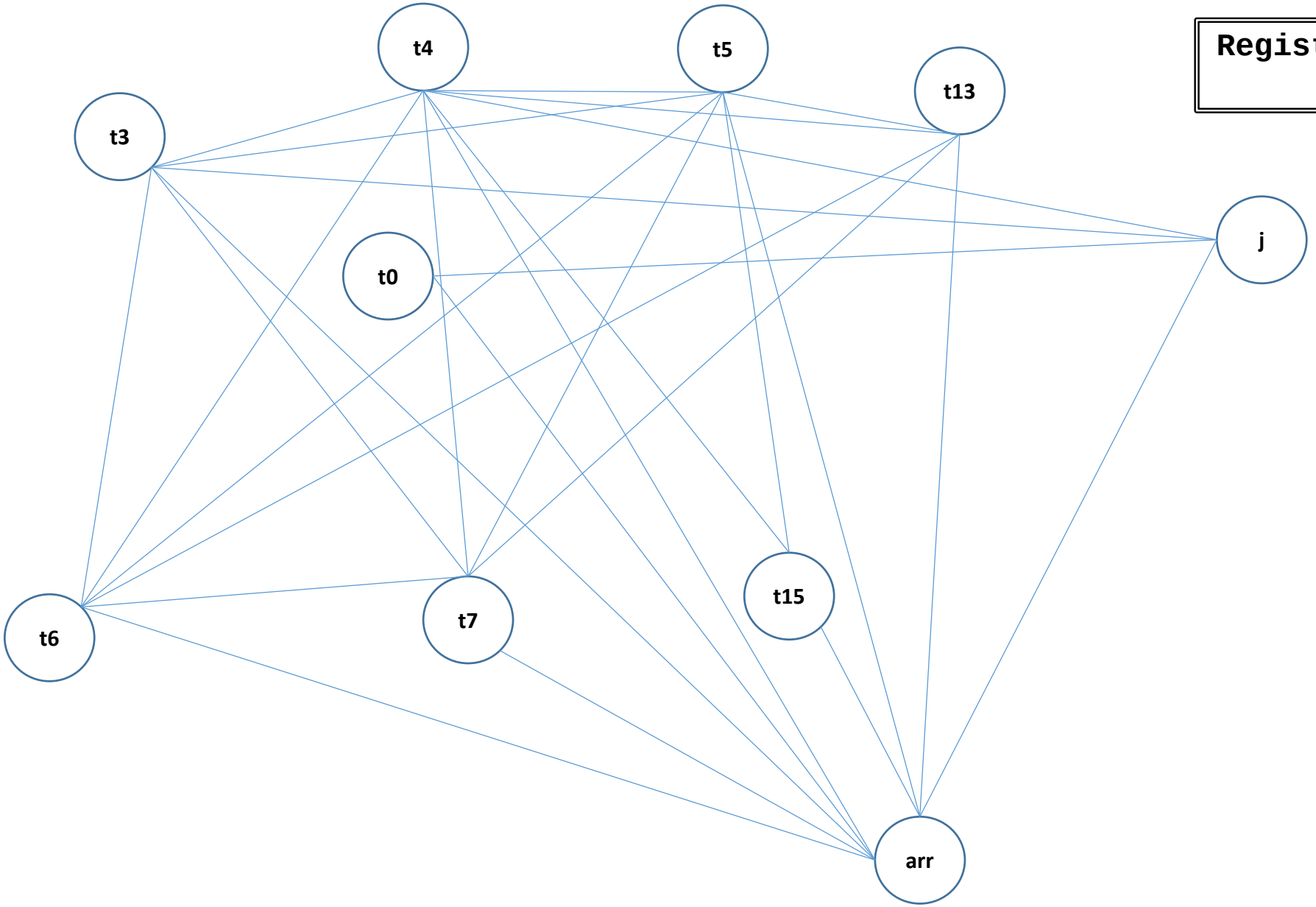
 # Registers k = 6



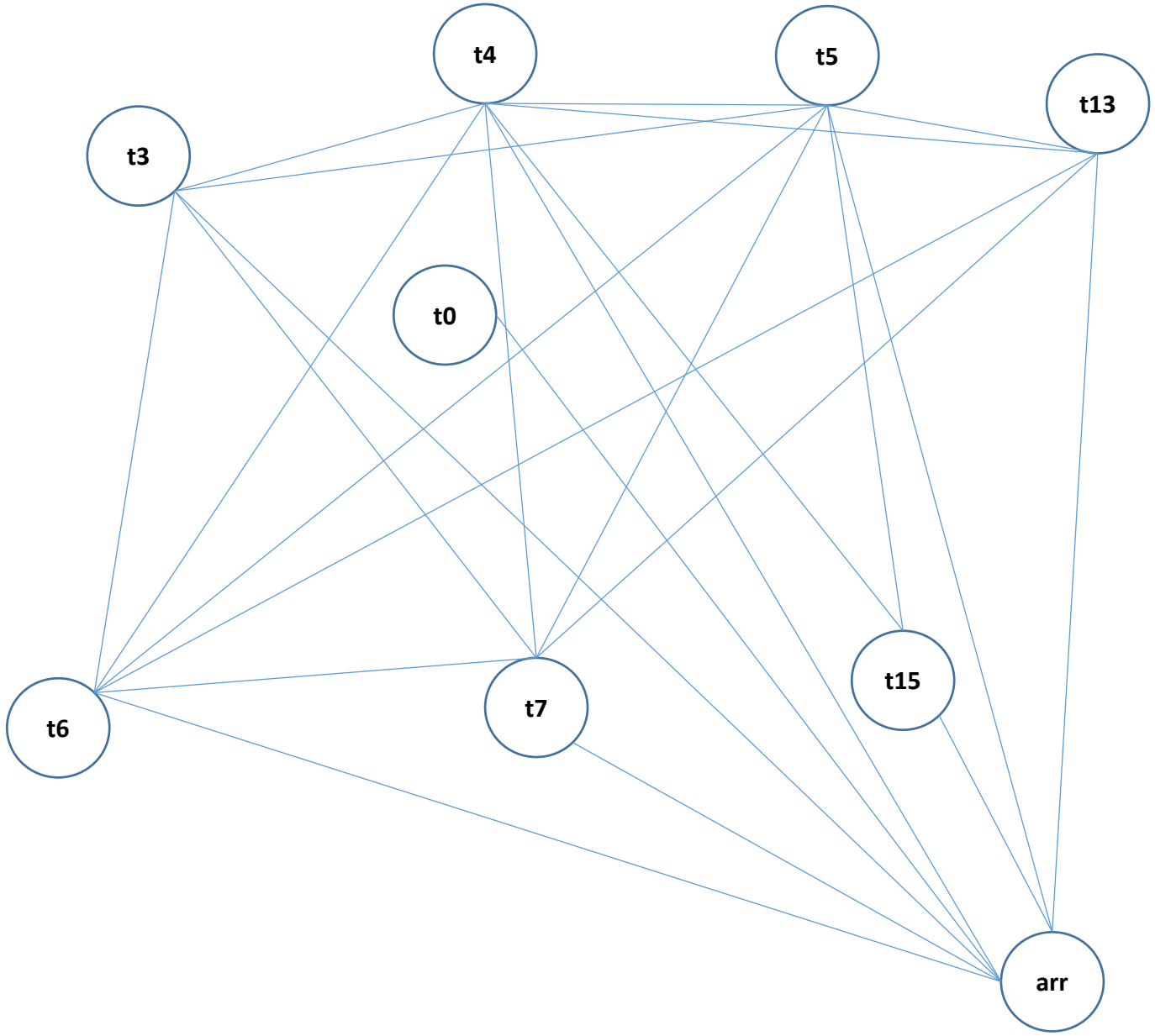
Stack: n, t2

Register Interference Graph
Spilled: t0 & i

 # Registers k = 6



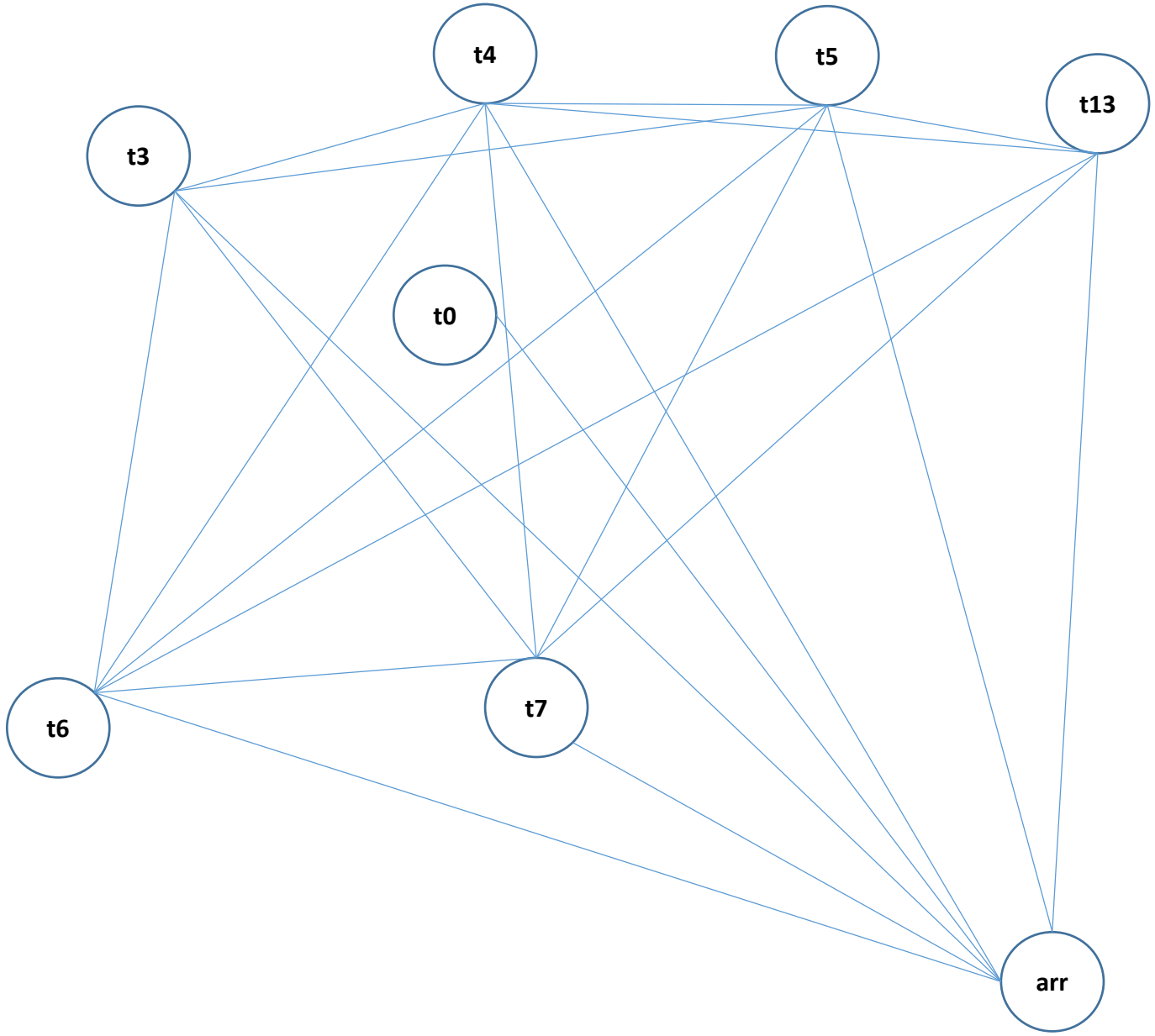
Stack: i, n, t2



Register Interference Graph
Spilled: t0 & i

 # Registers k = 6

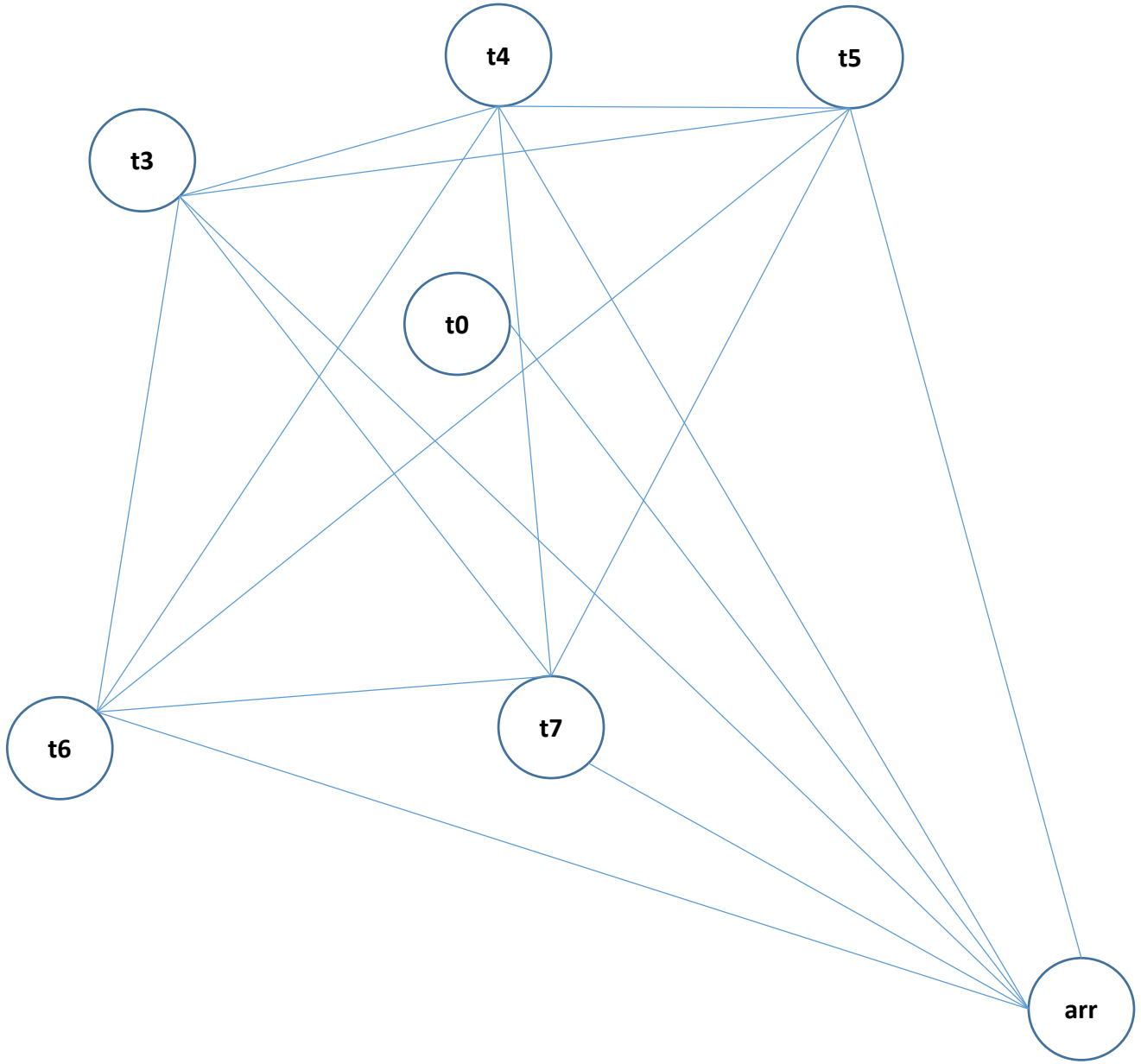
Stack: j, i, n, t2



Register Interference Graph
Spilled: t0 & i

 # Registers k = 6

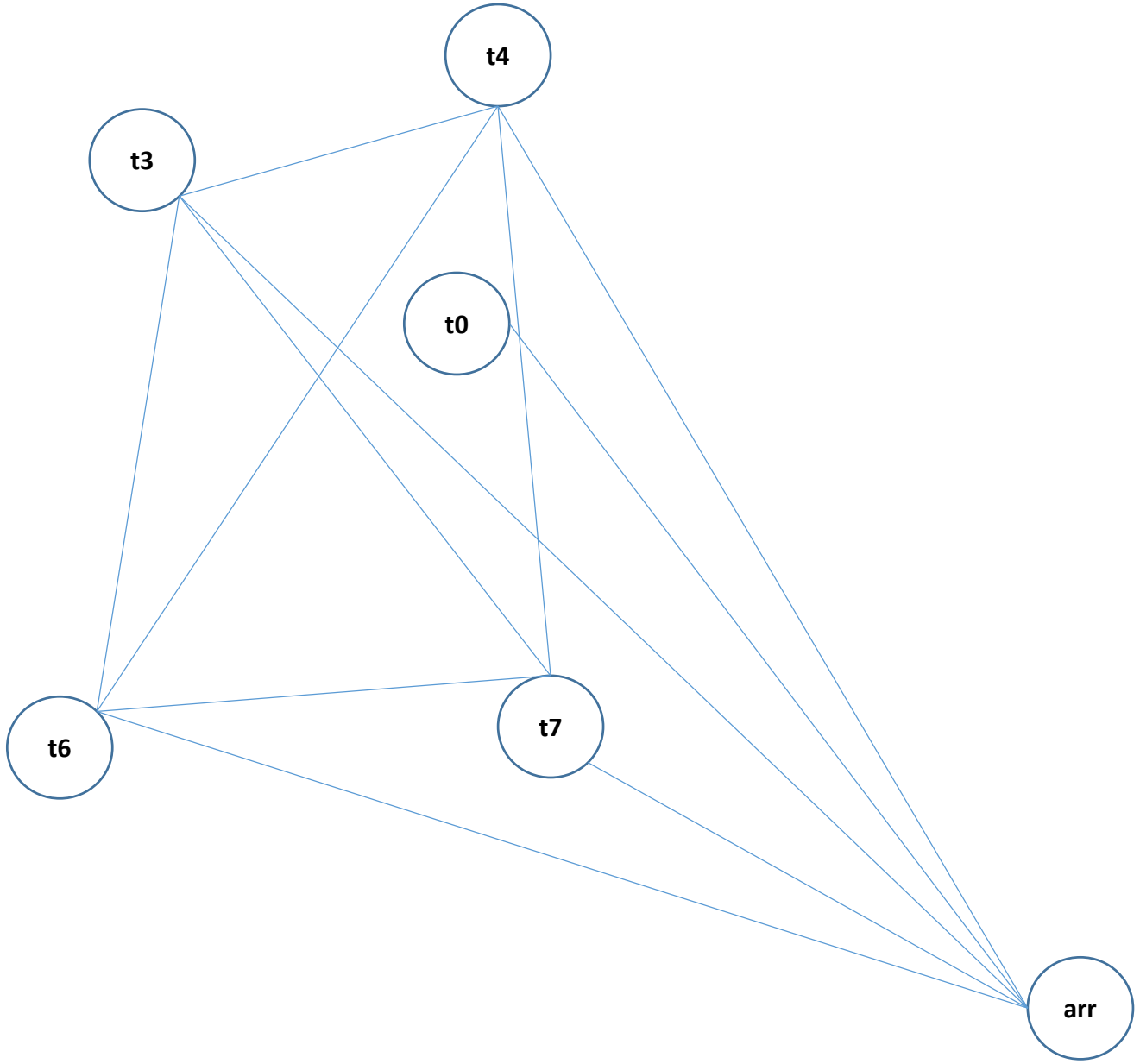
Stack: t15, j, i, n, t2



Register Interference Graph
Spilled: t0 & i

 # Registers k = 6

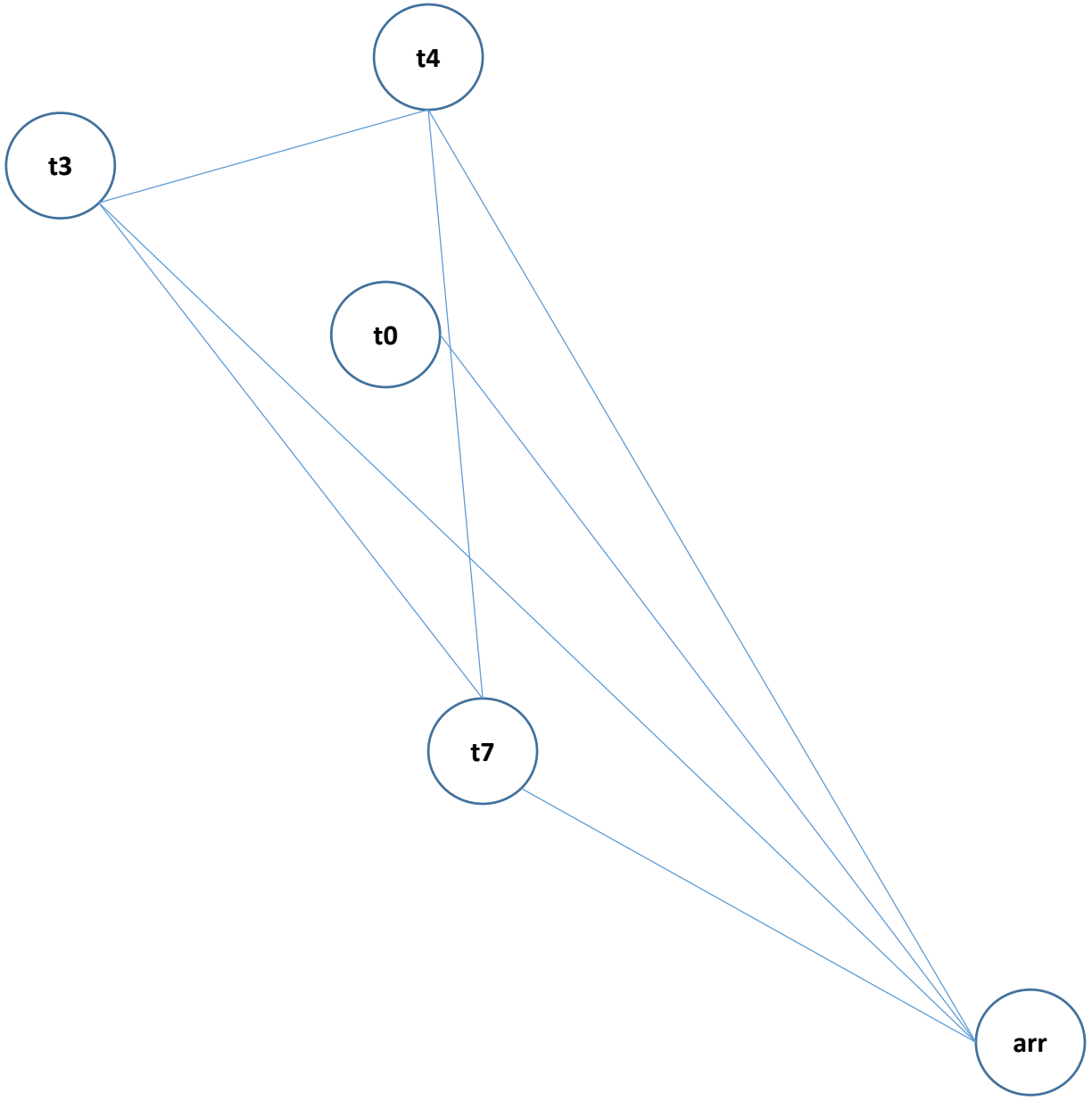
Stack: t13, t15, j, i, n, t2



Register Interference Graph
Spilled: t0 & i

 # Registers k = 6

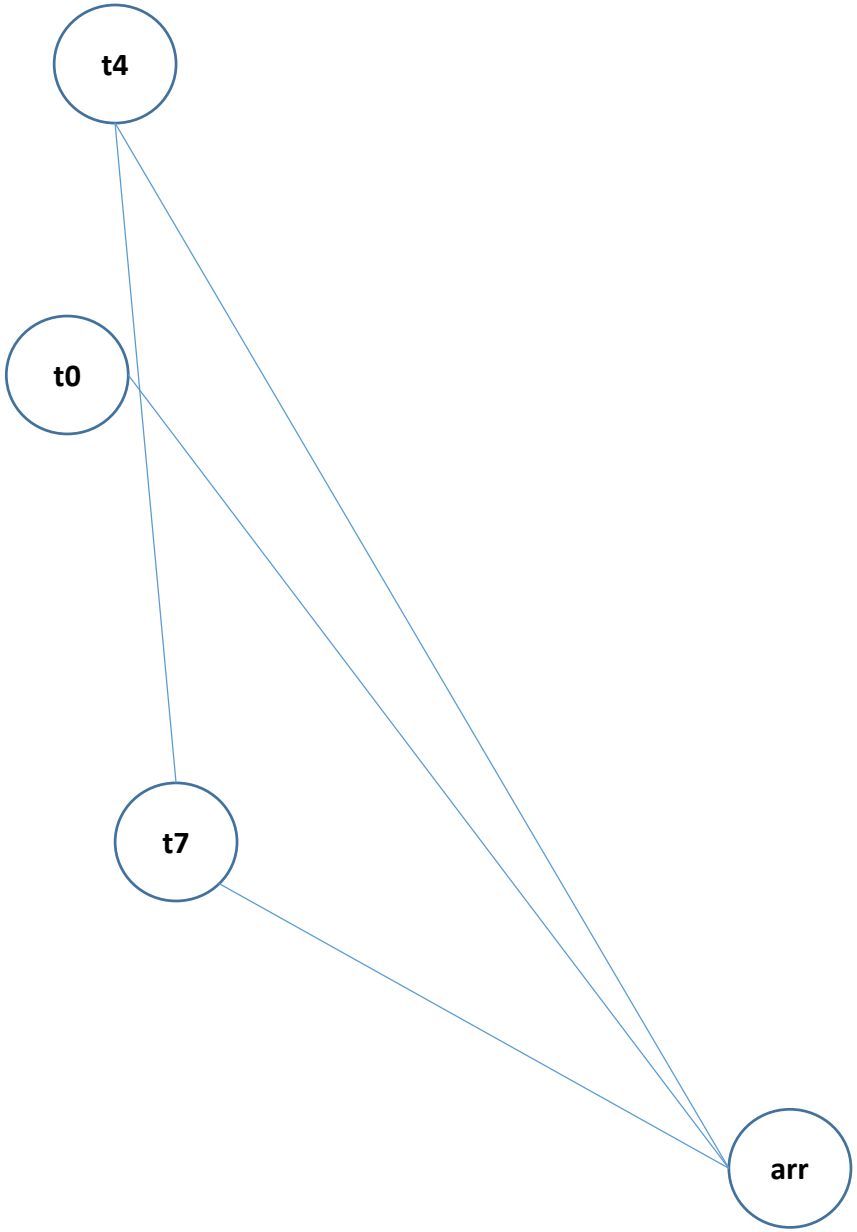
Stack: t5, t13, t15, j, i, n, t2



Register Interference Graph
Spilled: t0 & i

 # Registers k = 6

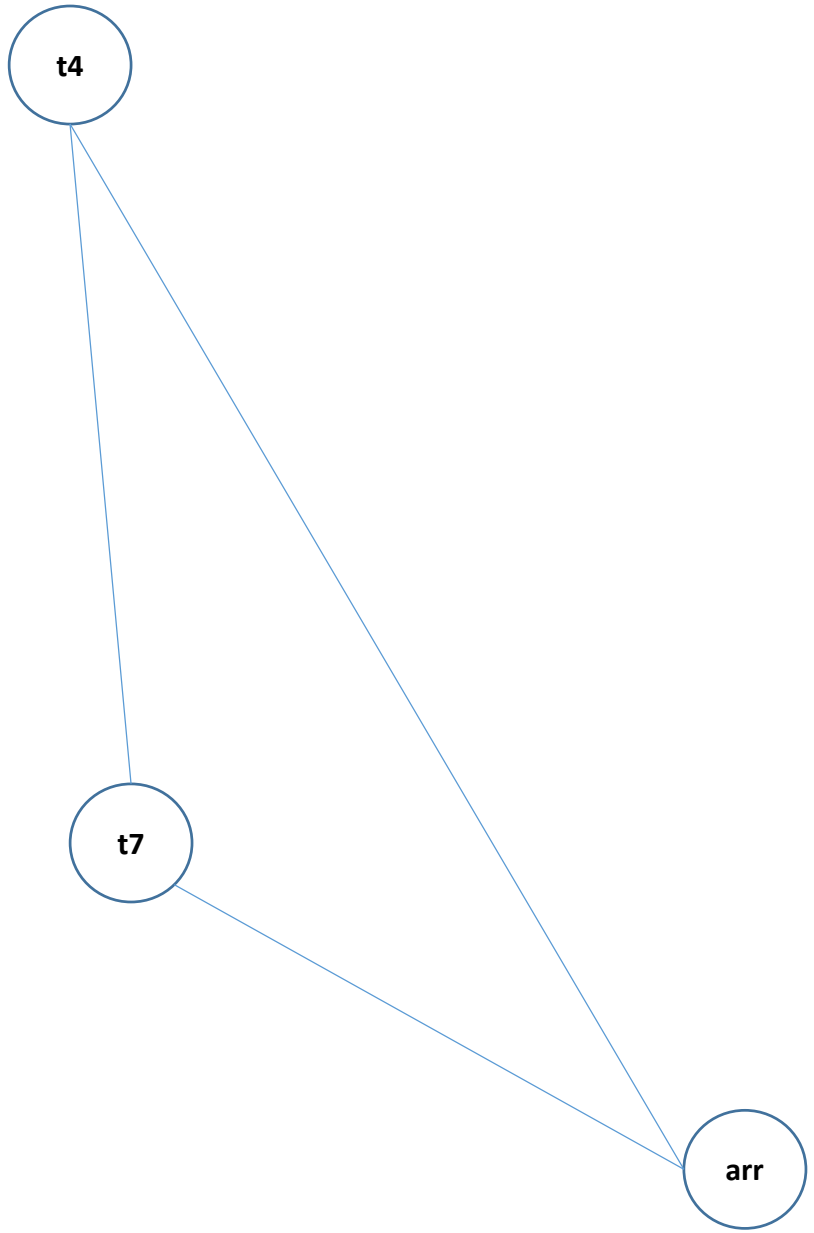
Stack: t6, t5, t13, t15, j, i, n, t2



Register Interference Graph
Spilled: t0 & i

 # Registers k = 6

Stack: t3, t6, t5, t13, t15, j, i, n, t2



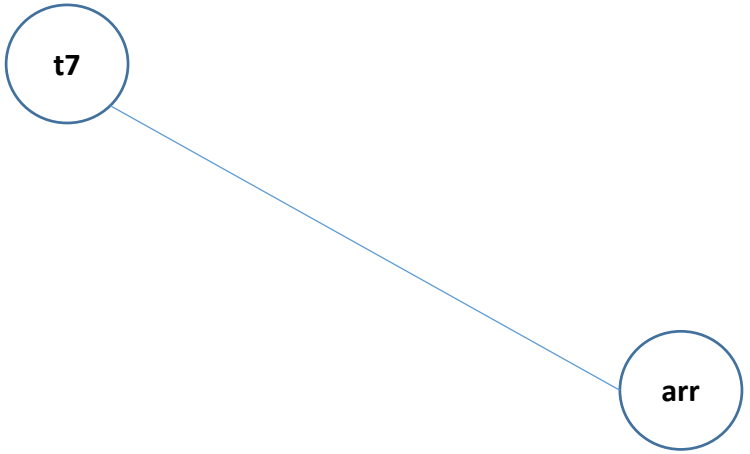
Register Interference Graph
Spilled: t0 & i

 # Registers k = 6

Stack: t0, t3, t6, t5, t13, t15, j, i, n, t2

Register Interference Graph
Spilled: t0 & i

 # Registers k = 6



Stack: t4, t0, t3, t6, t5, t13, t15, j, i, n, t2

Register Interference Graph
Spilled: t0 & i

 # Registers k = 6

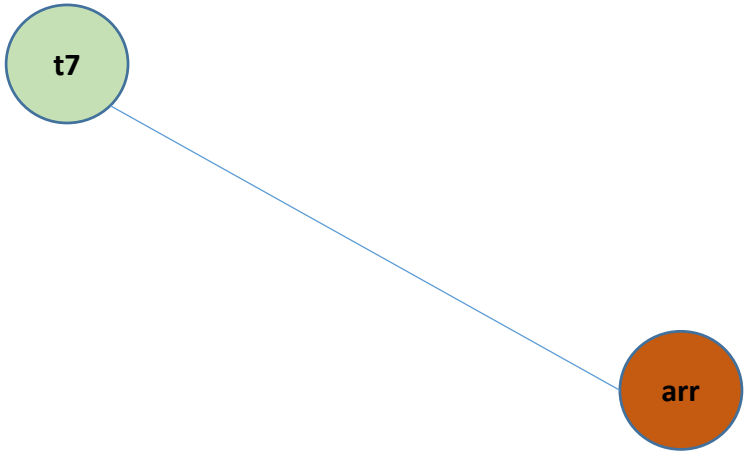
- **The entire graph can be colored with 6 colors**

Stack: t7, t4, t0, t3, t6, t5, t13, t15, j, i, n, t2

Colored Interference Graph
Spilled: t0 & i

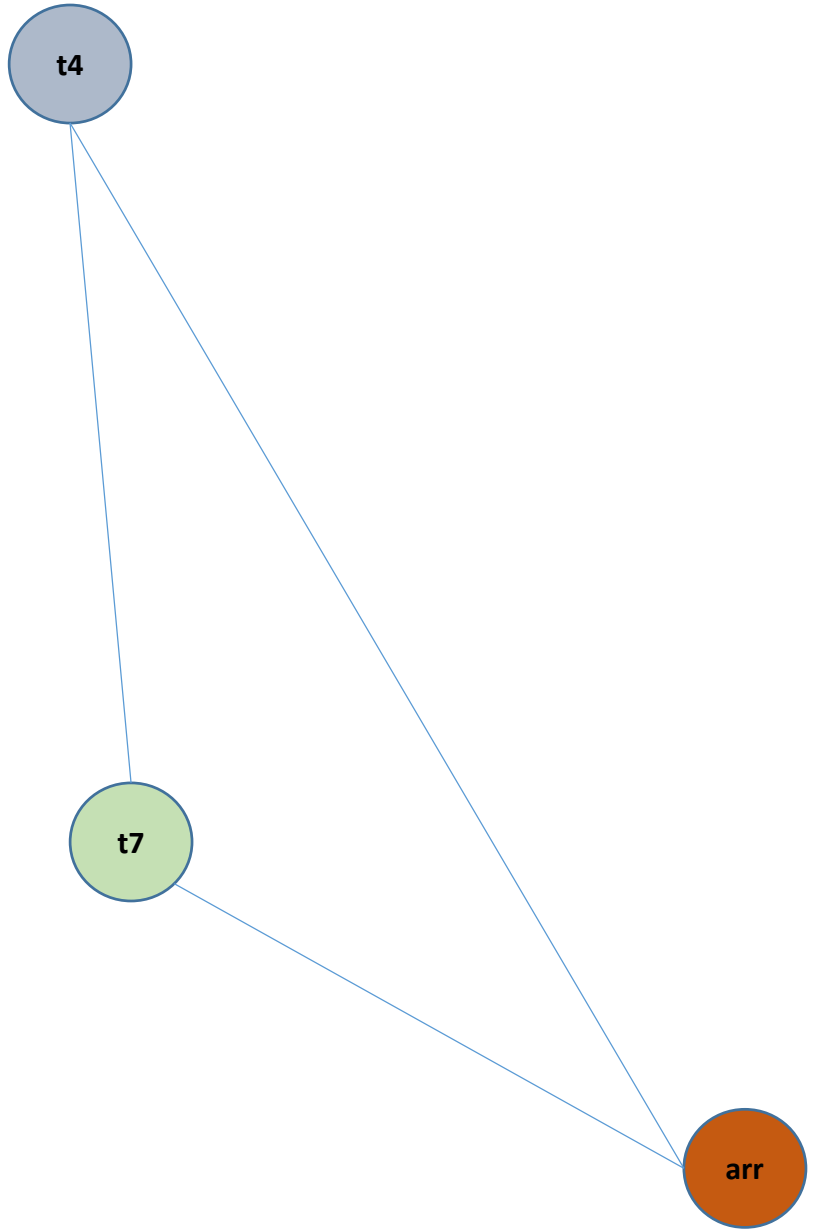
Stack: t7, t4, t0, t3, t6, t5, t13, t15, j, i, n, t2

Colored Interference Graph
Spilled: t0 & i



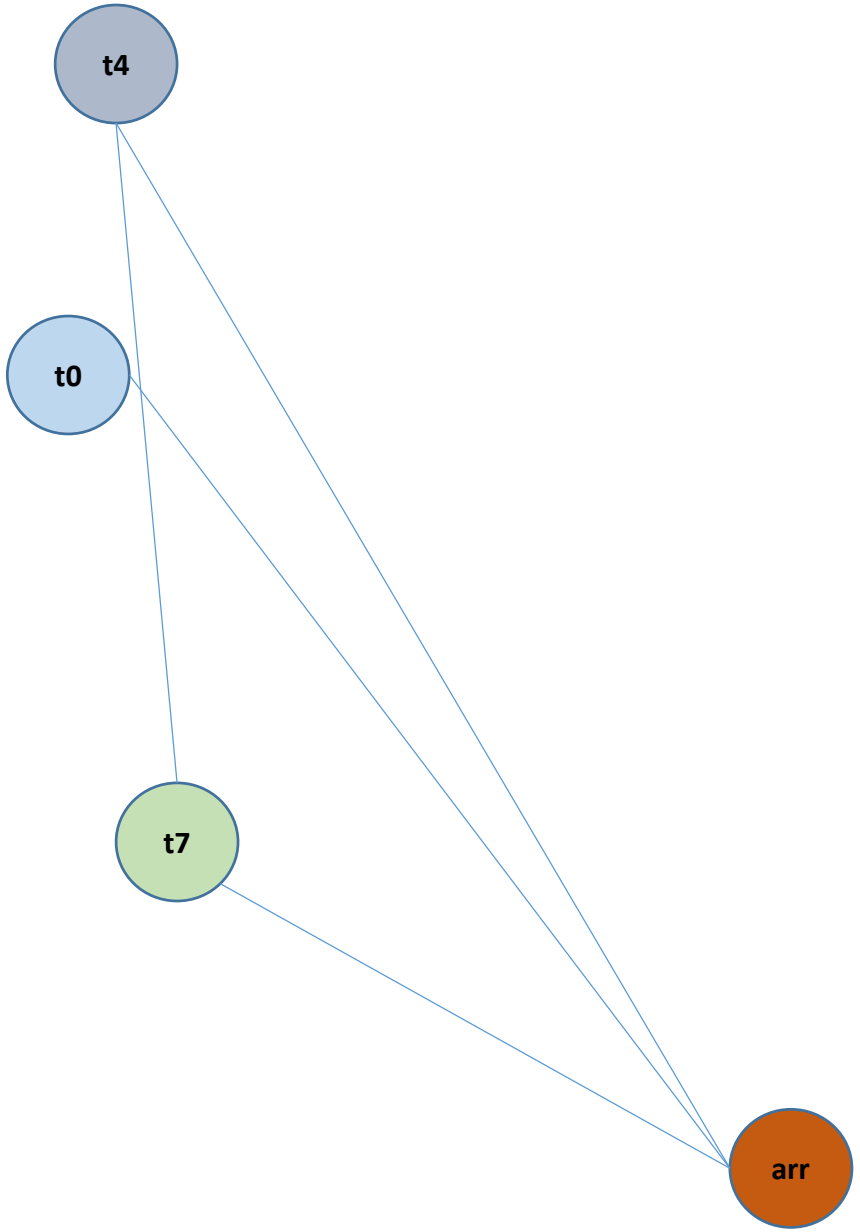
Stack: t4, t0, t3, t6, t5, t13, t15, j, i, n, t2

Colored Interference Graph
Spilled: t0 & i



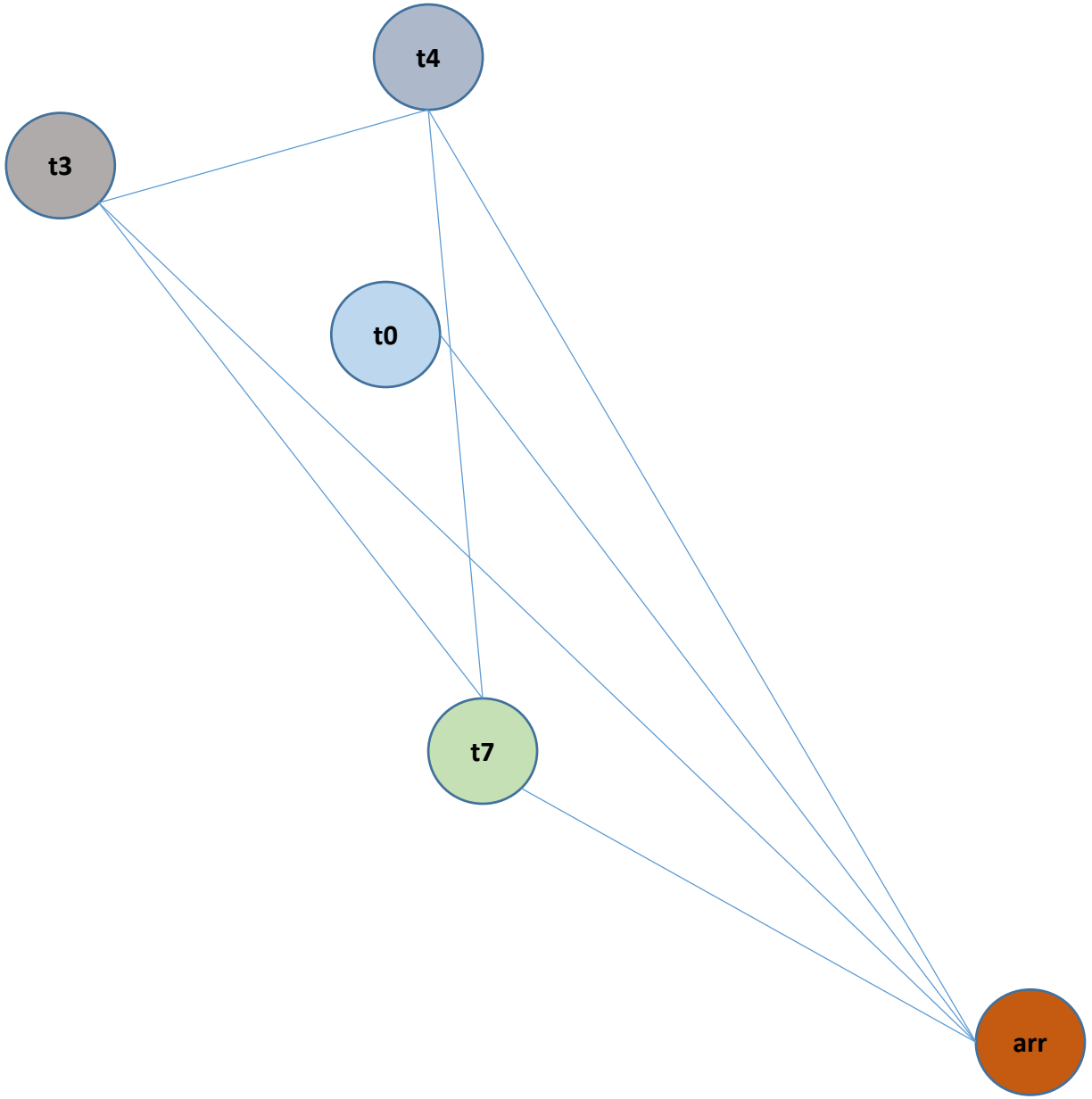
Stack: t0, t3, t6, t5, t13, t15, j, i, n, t2

Colored Interference Graph
Spilled: t0 & i



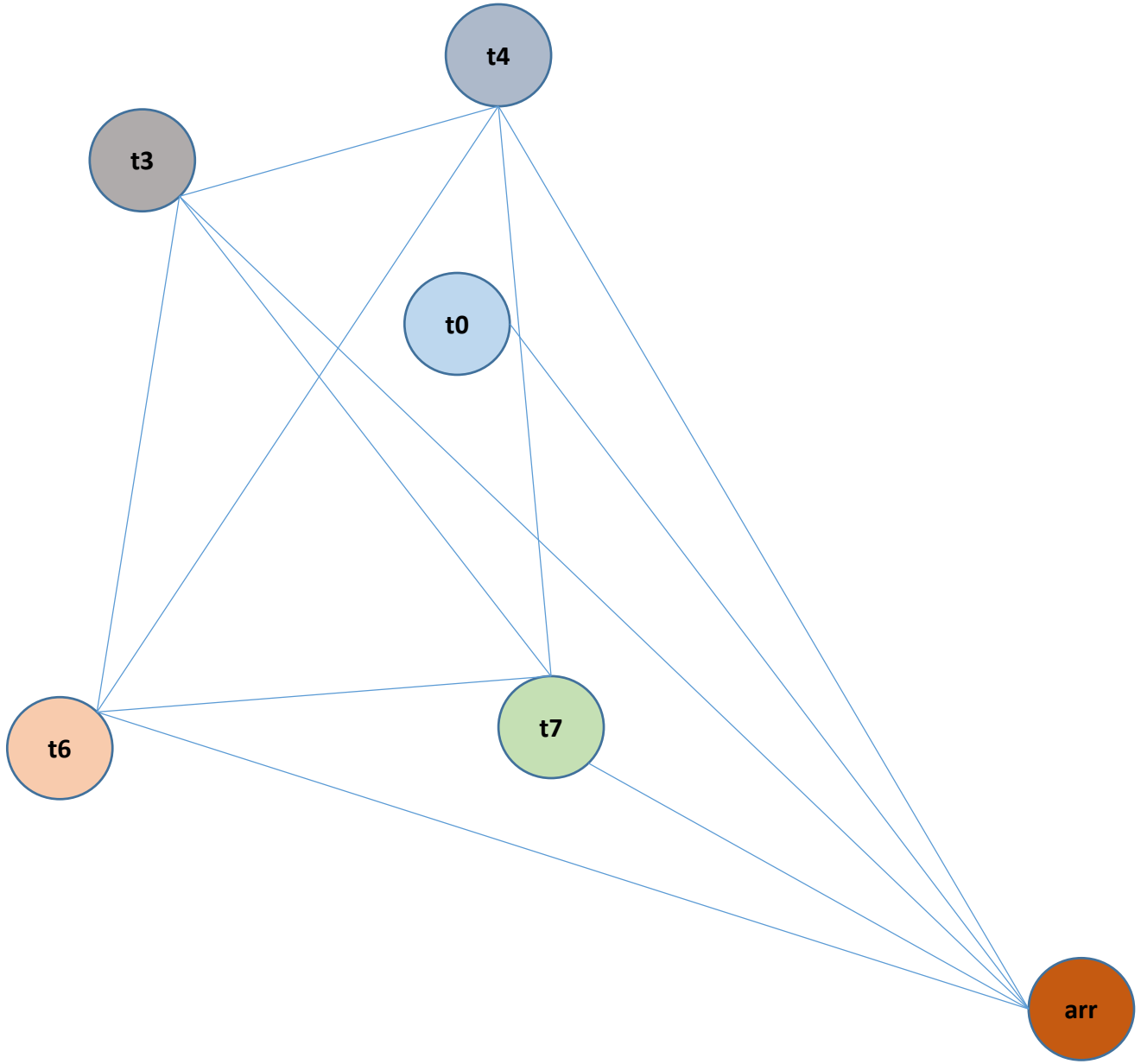
Stack: t3, t6, t5, t13, t15, j, i, n, t2

Colored Interference Graph
Spilled: t0 & i



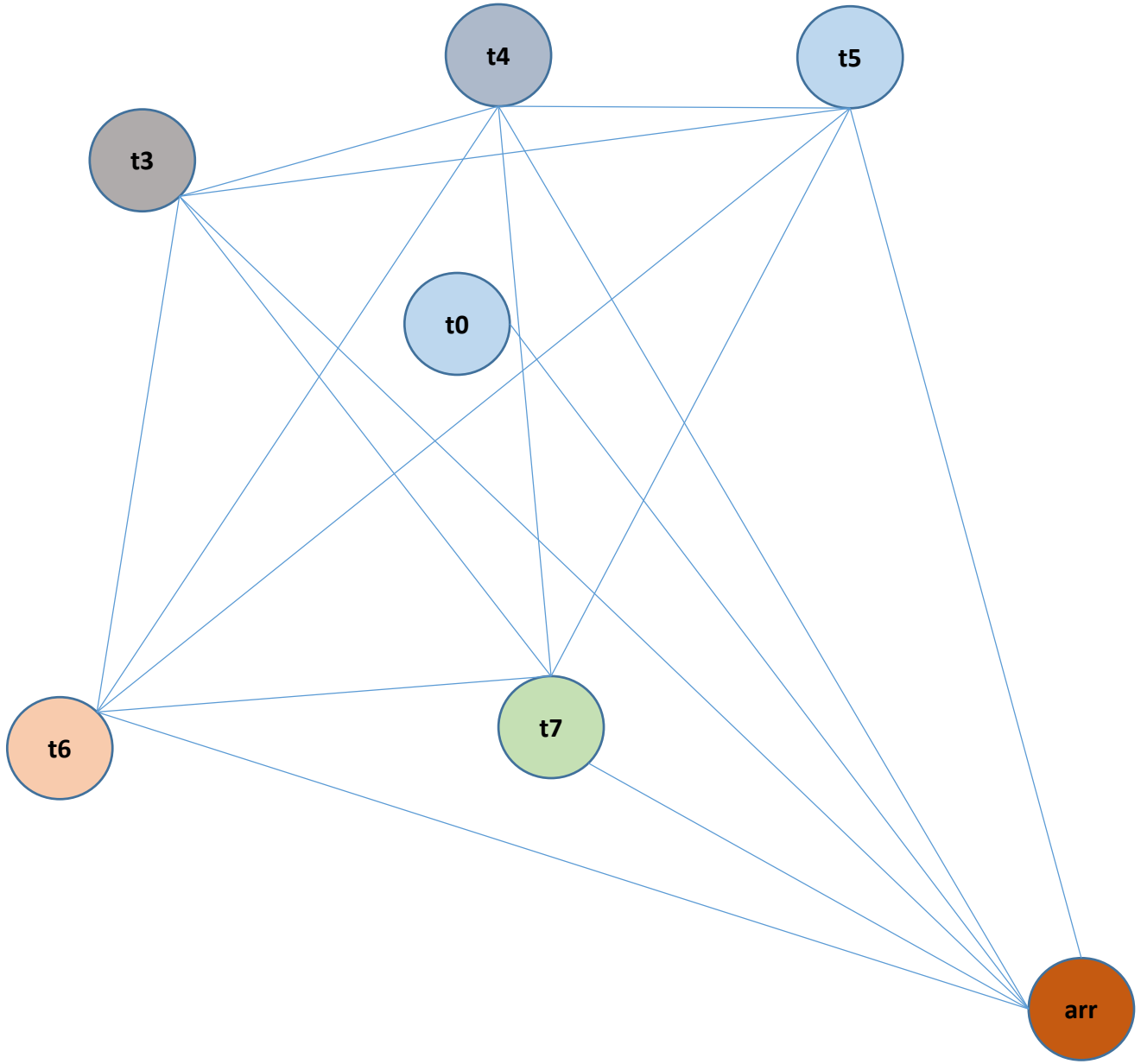
Stack: t6, t5, t13, t15, j, i, n, t2

Colored Interference Graph
Spilled: t0 & i



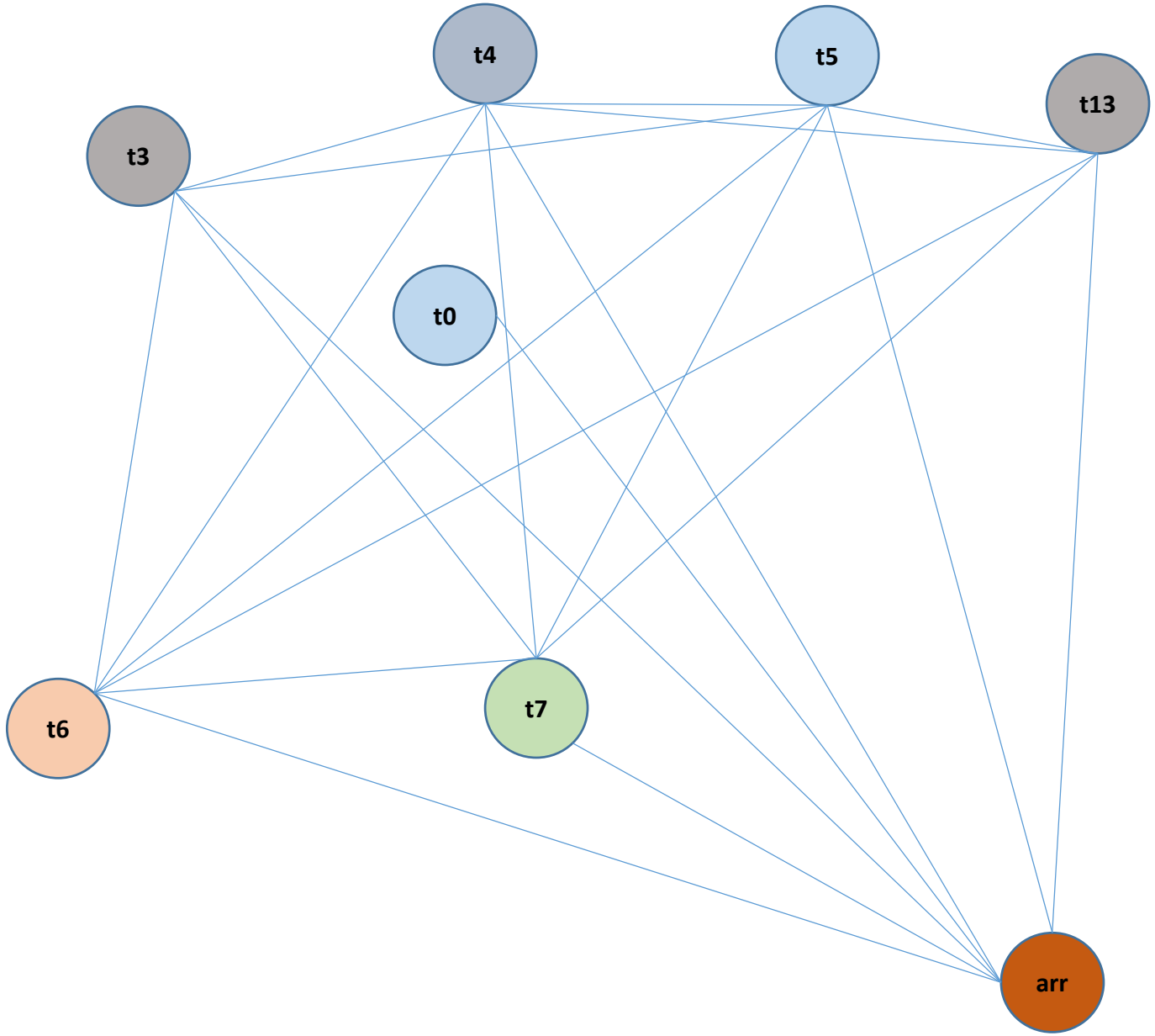
Stack: t5, t13, t15, j, i, n, t2

Colored Interference Graph
Spilled: t0 & i



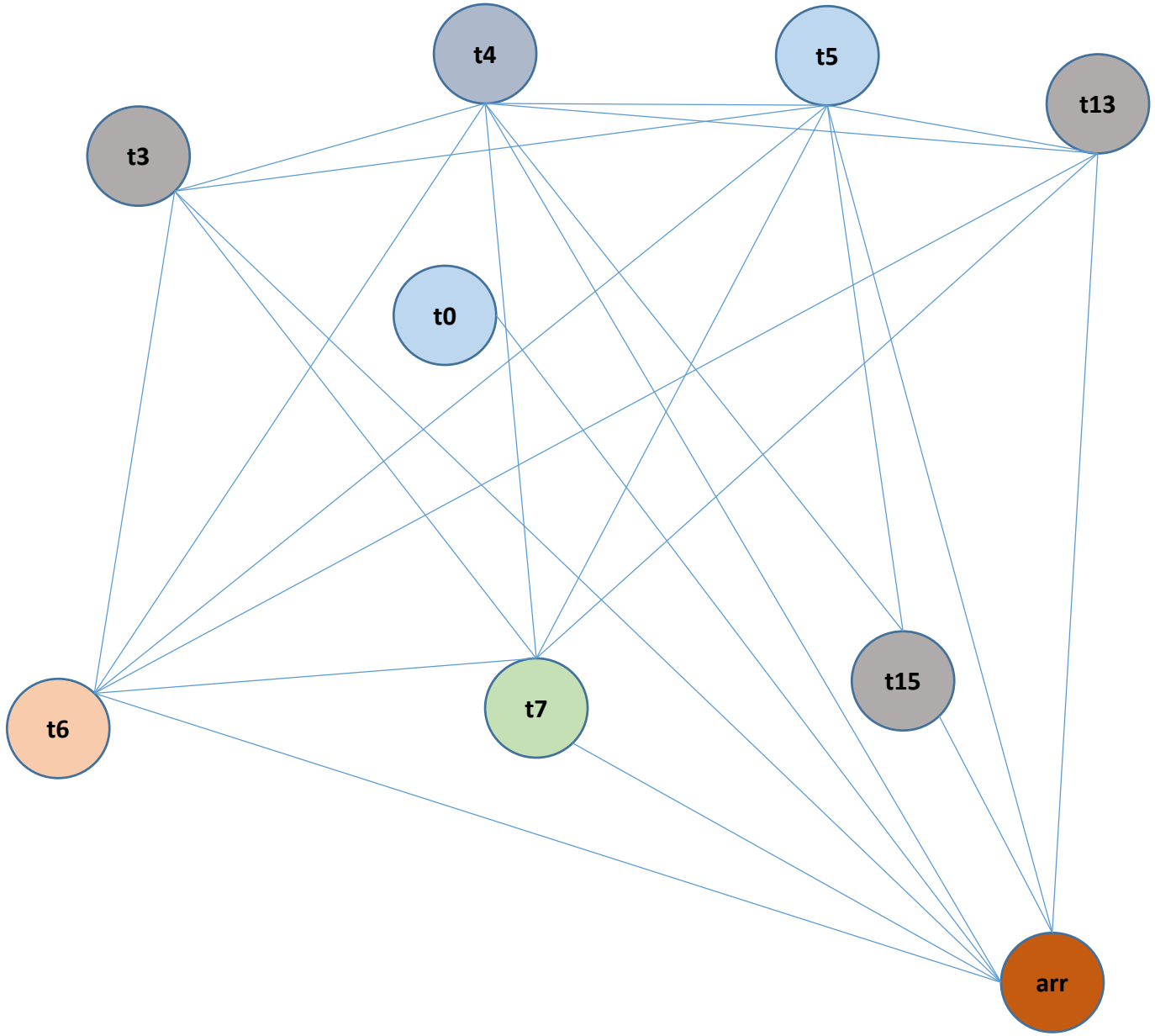
Stack: t13, t15, j, i, n, t2

Colored Interference Graph
Spilled: t0 & i



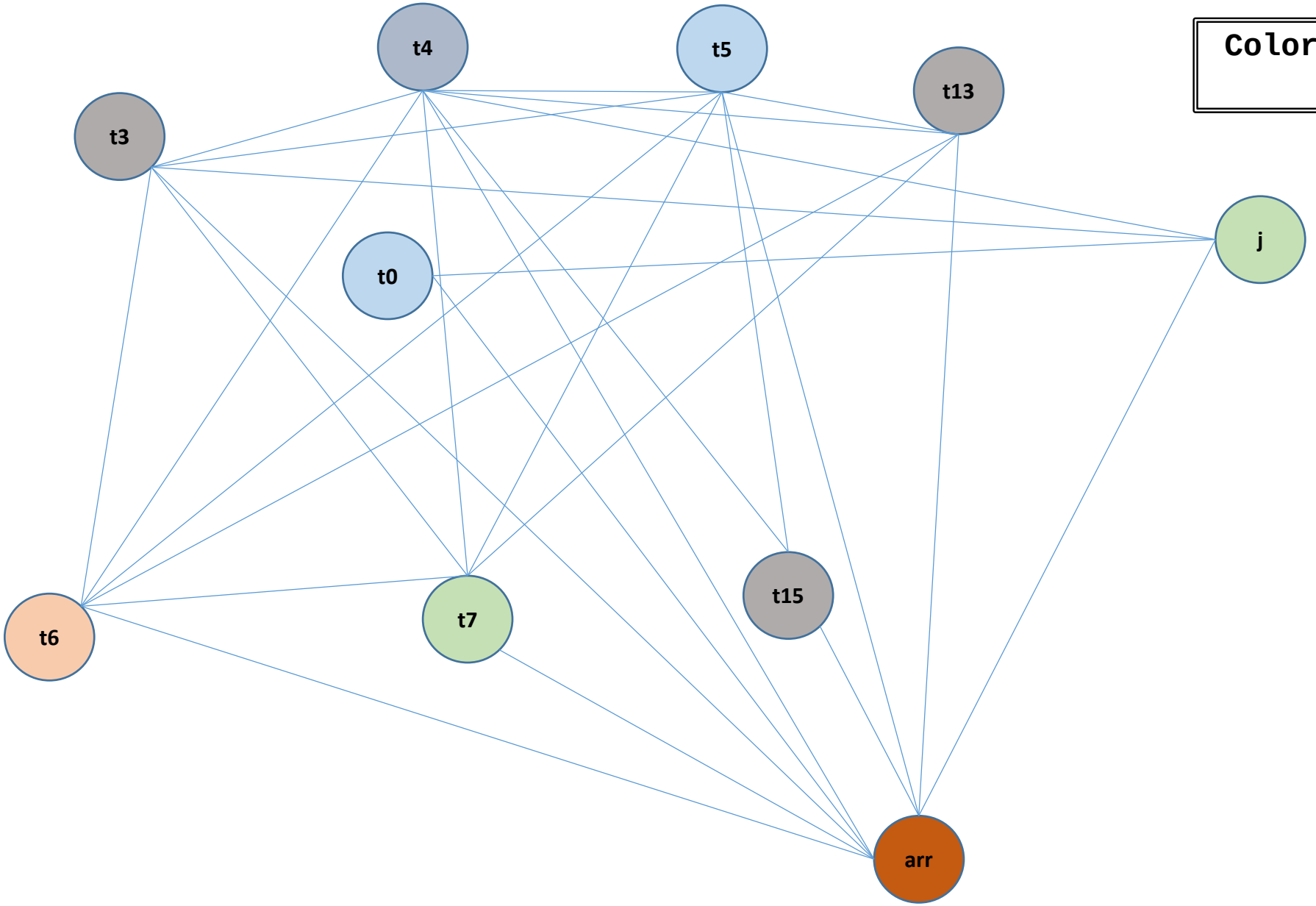
Stack: t15, j, i, n, t2

Colored Interference Graph
Spilled: t0 & i



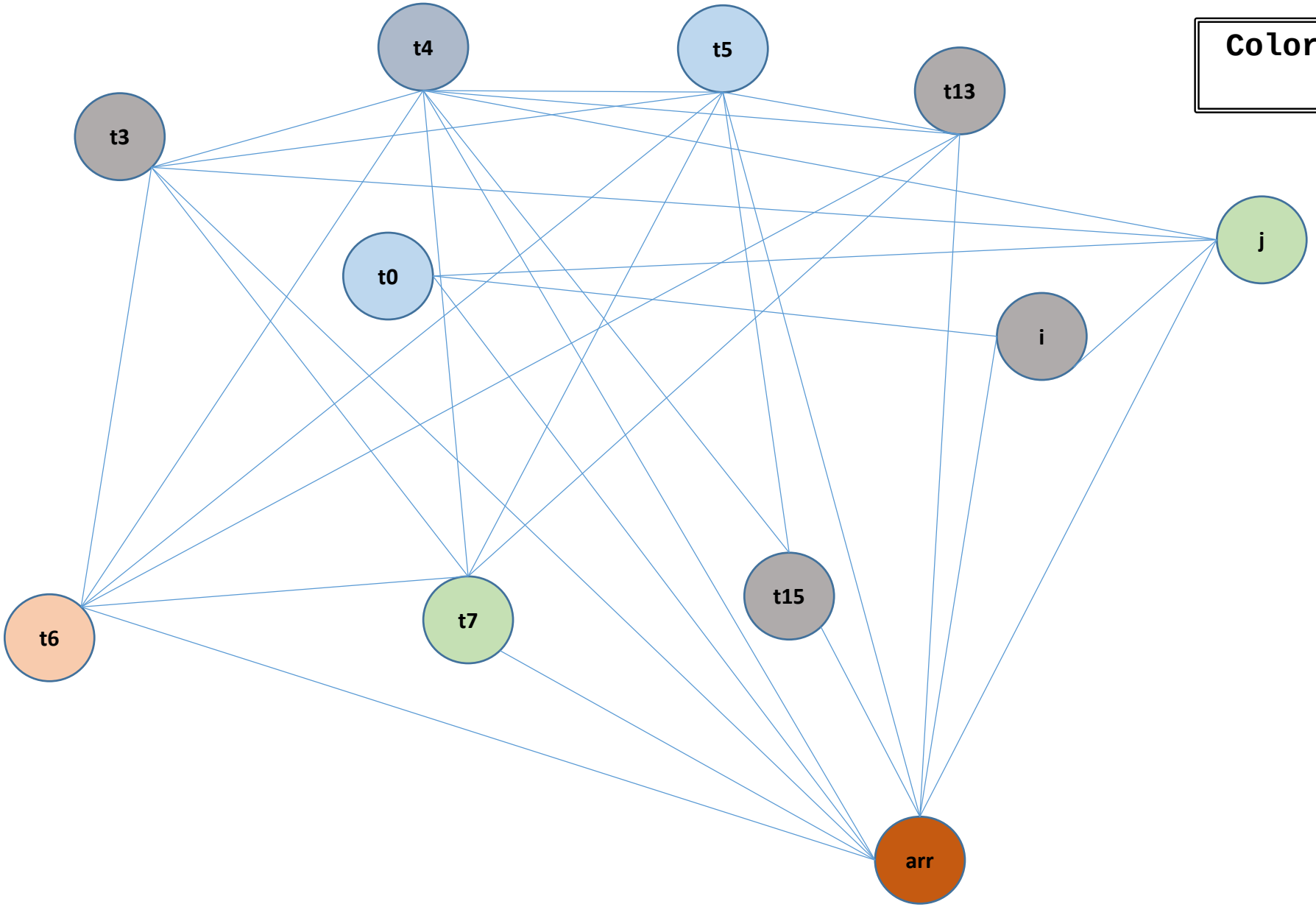
Stack: j, i, n, t2

Colored Interference Graph
Spilled: t0 & i



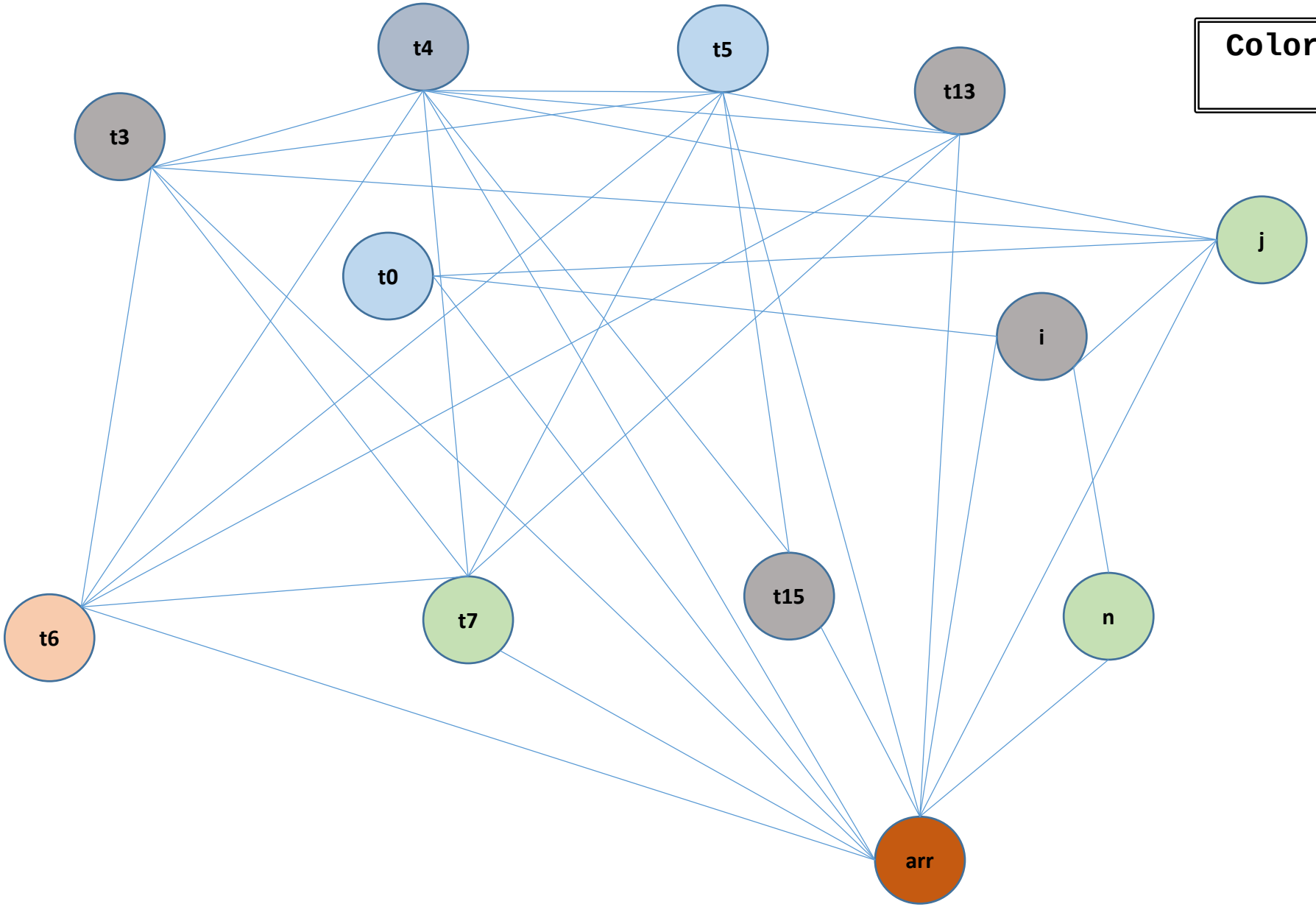
Stack: i, n, t2

Colored Interference Graph
Spilled: t0 & i



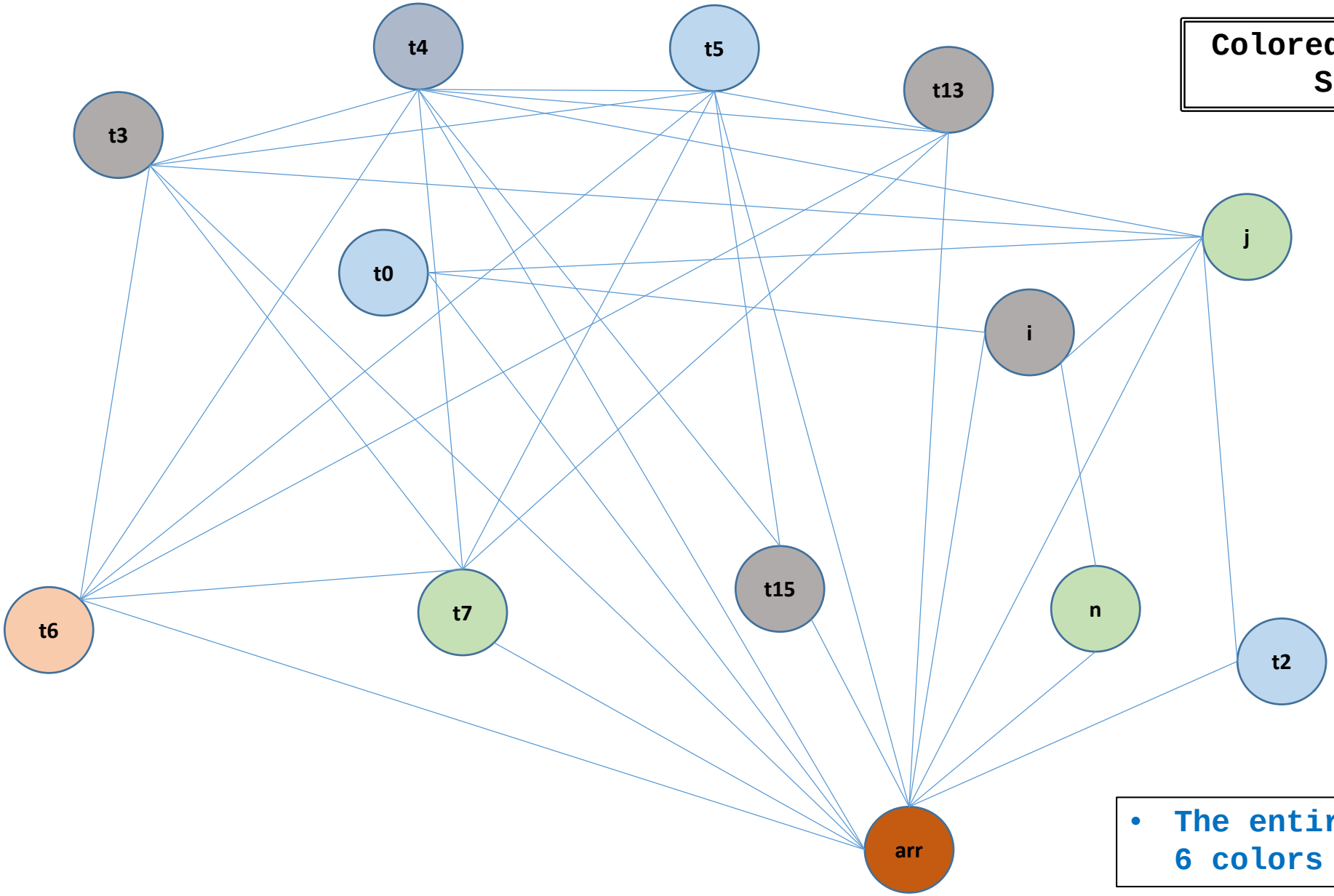
Stack: n, t2

Colored Interference Graph
Spilled: t0 & i



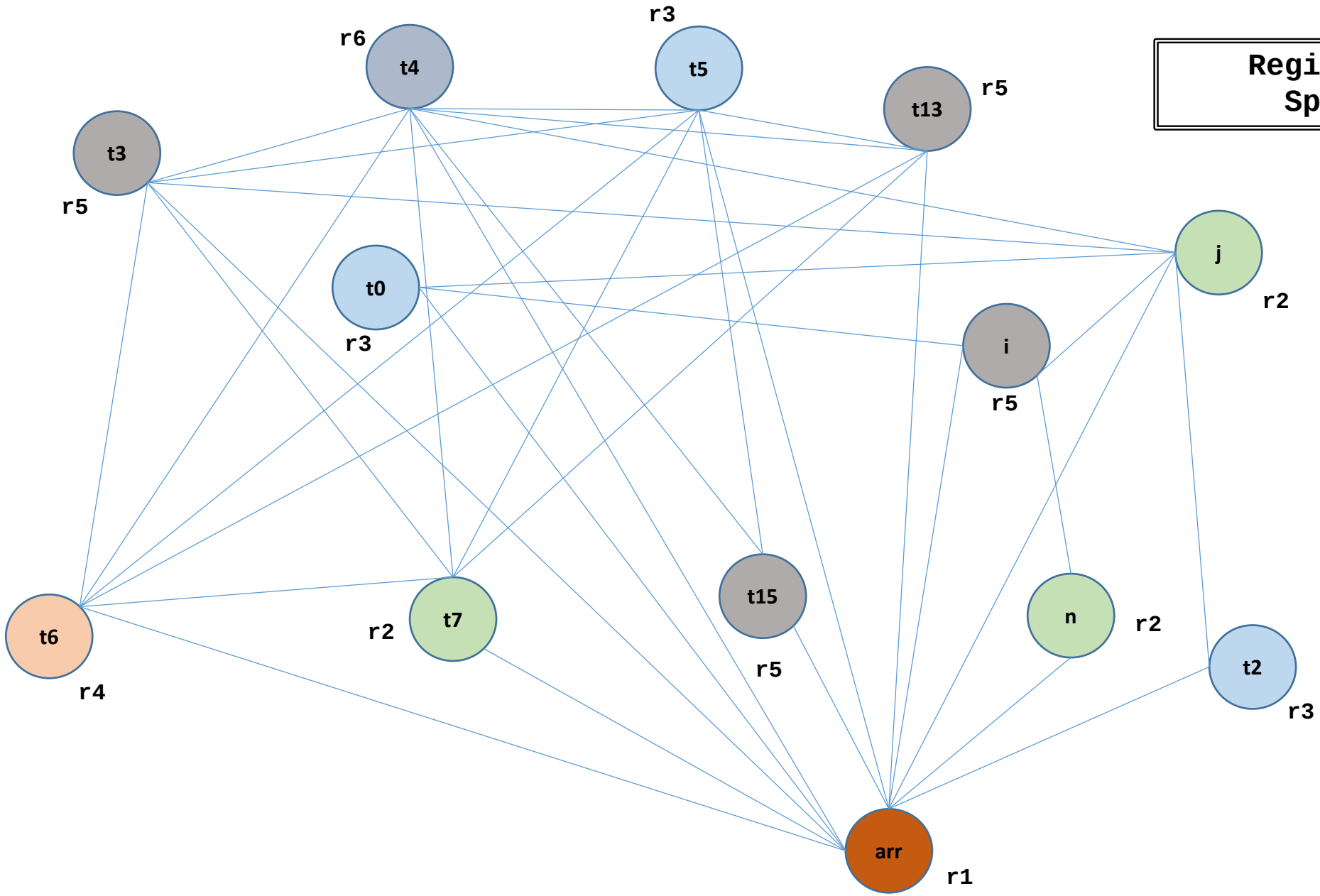
Stack: t2

Colored Interference Graph
Spilled: t0 & i



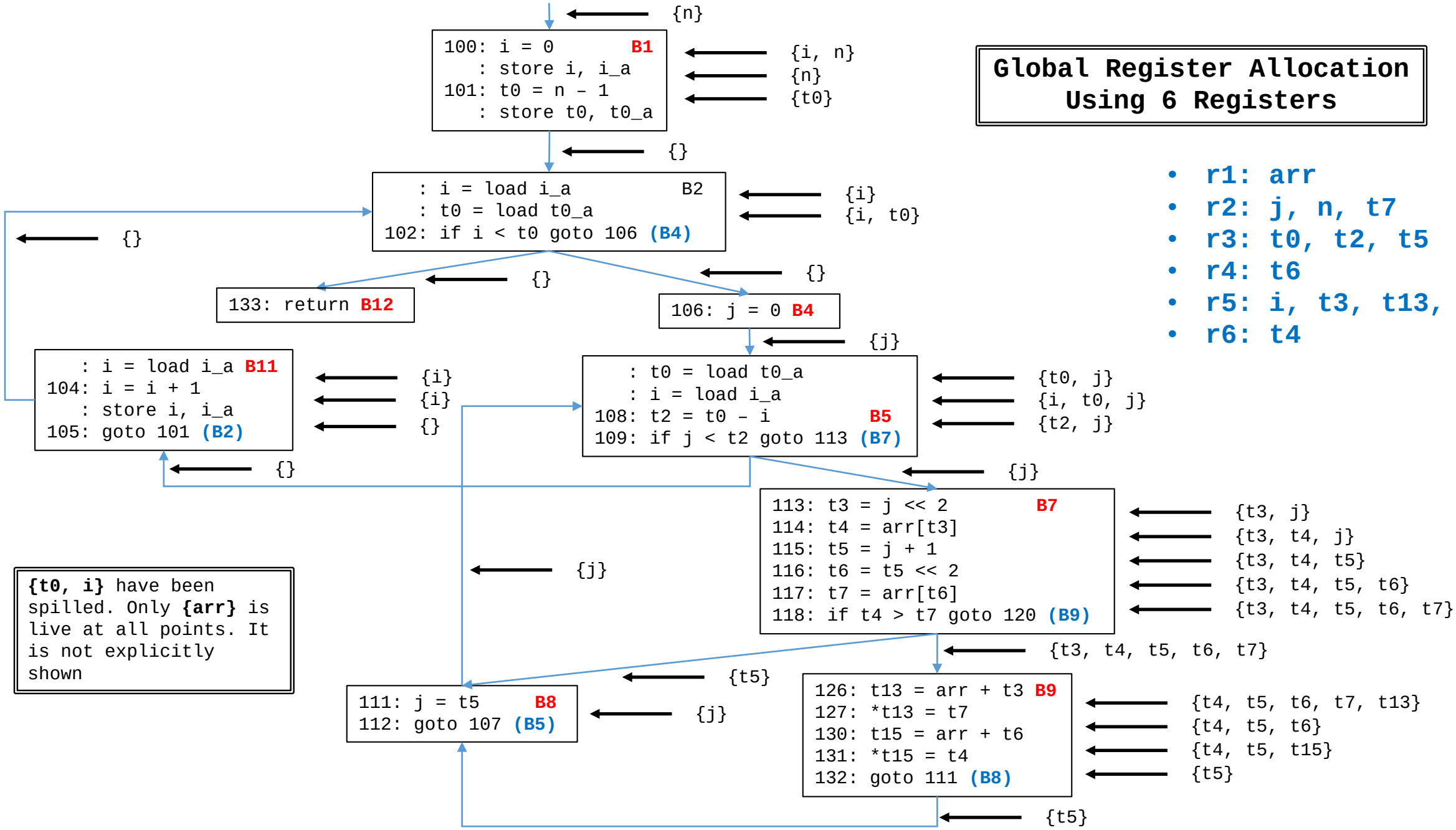
- **The entire graph colored with 6 colors**

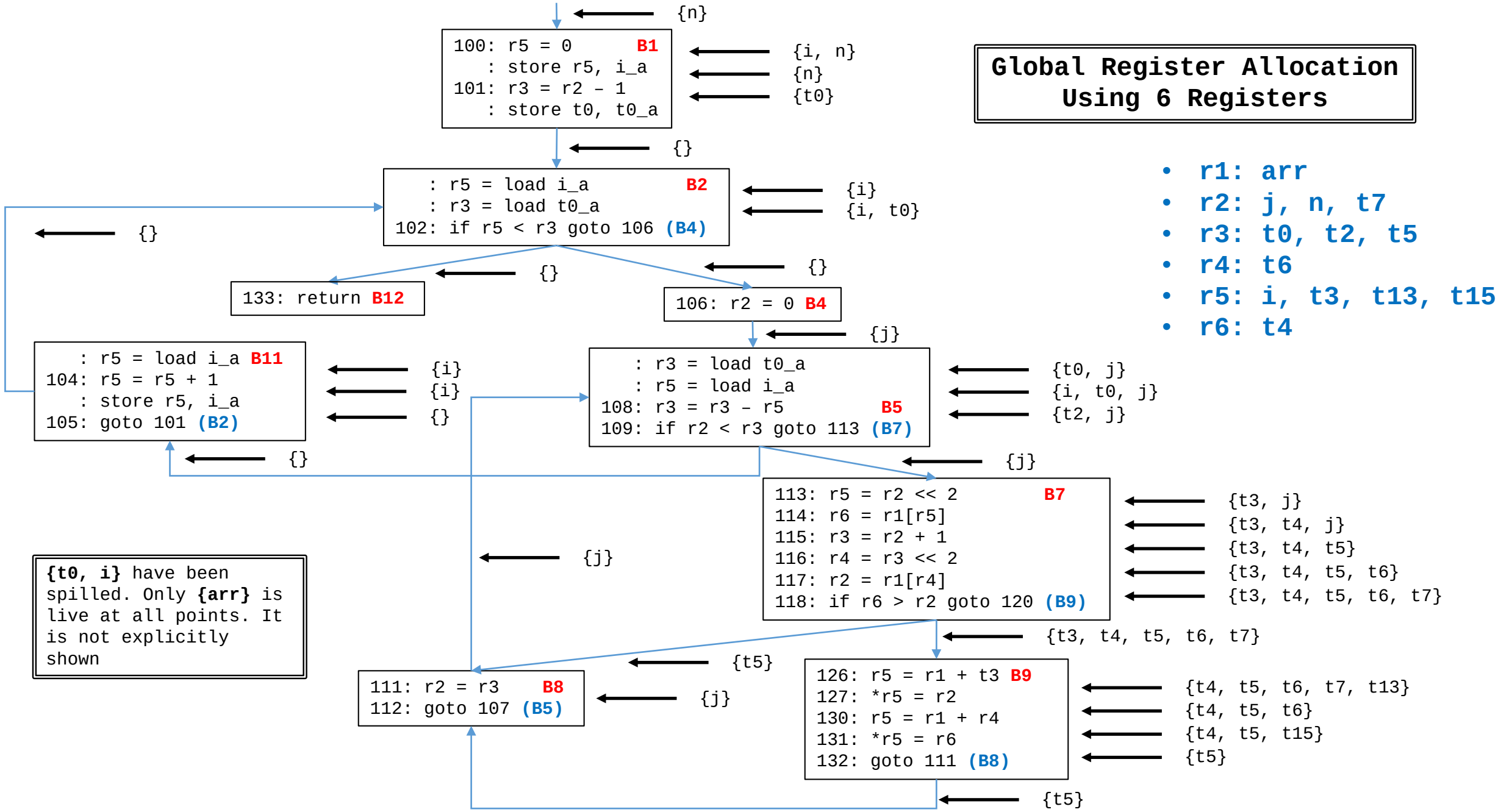
Stack:



Register Allocation
Spilled: t0 & i

- r1: arr
- r2: j, n, t7
- r3: t0, t2, t5
- r4: t6
- r5: i, t3, t13, t15
- r6: t4


{n}
100: i = 0 B1
: store i, i_a
101: t0 = n - 1
: store t0, t0_a
{i, n}
{n}
{t0}
{}
: i = load i_a B2
: t0 = load t0_a
102: if i < t0 goto 106 (B4)
{i}
{i, t0}
{}
{}
{}
133: return B12
106: j = 0 B4
{j}
: i = load i_a B11
104: i = i + 1
: store i, i_a
105: goto 101 (B2)
{i}
{i}
{}
{}
: t0 = load t0_a
: i = load i_a
108: t2 = t0 - i B5
109: if j < t2 goto 113 (B7)
{t0, j}
{i, t0, j}
{t2, j}
{j}
113: t3 = j << 2 B7
114: t4 = arr[t3]
115: t5 = j + 1
116: t6 = t5 << 2
117: t7 = arr[t6]
118: if t4 > t7 goto 120 (B9)
{t3, j}
{t3, t4, j}
{t3, t4, t5}
{t3, t4, t5, t6}
{t3, t4, t5, t6, t7}
{t3, t4, t5, t6, t7}
{j}
{t5}
111: j = t5 B8
112: goto 107 (B5)
{j}
126: t13 = arr + t3 B9
127: *t13 = t7
130: t15 = arr + t6
131: *t15 = t4
132: goto 111 (B8)
{t4, t5, t6, t7, t13}
{t4, t5, t6}
{t4, t5, t15}
{t5}
{t5}
{t0, i} have been spilled. Only {arr} is live at all points. It is not explicitly shown
Global Register Allocation Using 6 Registers
r1: arr
r2: j, n, t7
r3: t0, t2, t5
r4: t6
r5: i, t3, t13, t15
r6: t4



Global Register Allocation Using 6 Registers

- r1: arr
- r2: j, n, t7
- r3: t0, t2, t5
- r4: t6
- r5: i, t3, t13, t15
- r6: t4

THANK YOU