

# Register Allocation

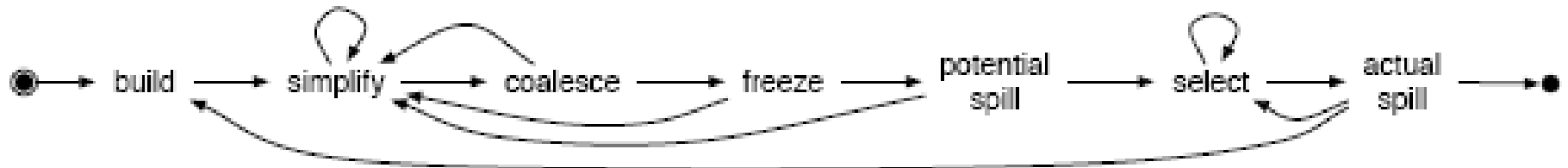
**Ajay Mathew**

- *Pereira and Palsberg. Register allocation via coloring of chordal graphs. APLoS'05*
- *Pereira and Palsberg. Register allocation after classical SSA elimination is NP-complete. FOSSACS'06*

# Register Allocation – *Recap*

- Interference graph
- Graph coloring
- NP Complete- Chaitin's proof
- Heuristics- priority coloring, Kempe's method

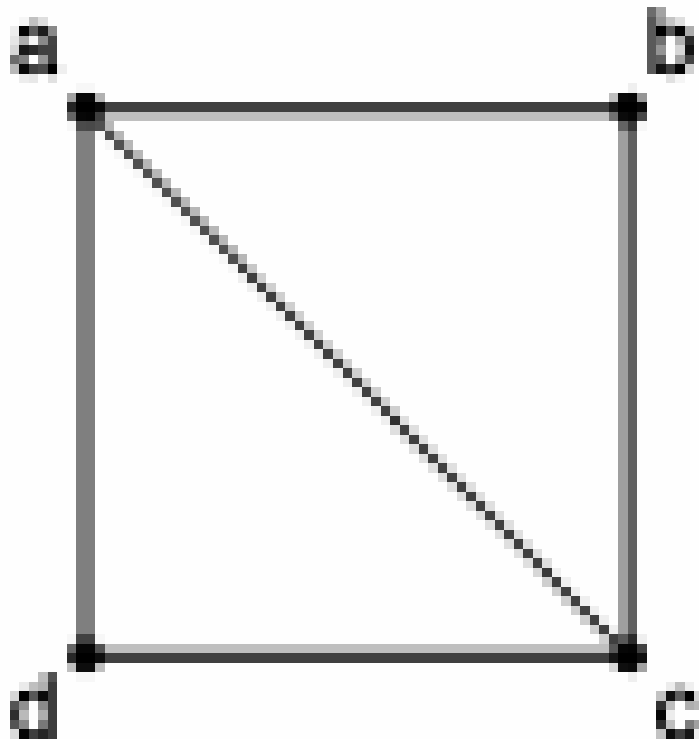
# *George and Appel's iterative spilling algorithm*



- *Drawback of George and Appel's iterative spilling algorithm- spilling and coalescing very complex.*
- The interference relations b/w temporary variables can form any possible graph-  
*Chaitin*

# Chordal Graph

- **Chord**- an edge which is not part of the cycle but which connects two vertices on the cycle.
- A graph is chordal if **every cycle with four or more edges** has a chord.



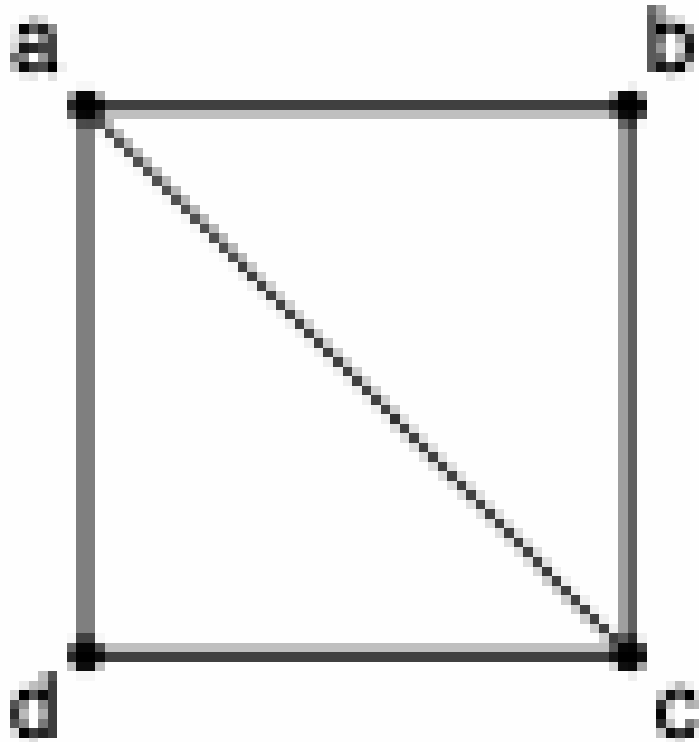
# useful properties

- **Minimum coloring--  $O(E + V)$  time**
- maximum clique
- maximum independent set
- minimum covering by cliques

All these NP complete problems in general graphs are now solvable in polynomial time.

# Simplicial Elimination Ordering

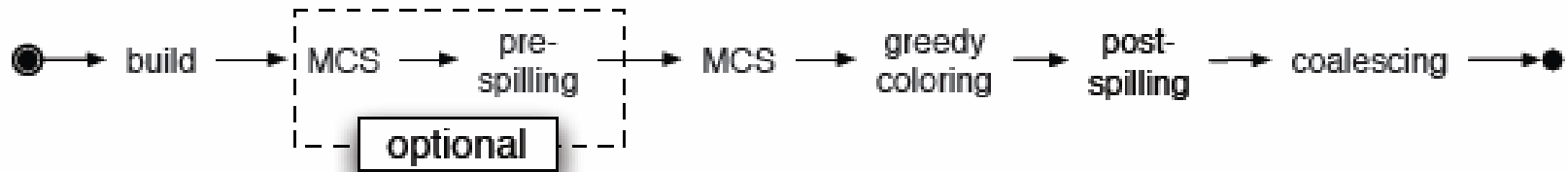
- A vertex  $v$  is called simplicial if its neighborhood in  $G$  is a clique.
- A SEO of  $G$  is a bijection  $V(G) \rightarrow \{1 \dots V\}$ , such that every vertex  $v_i$  is a simplicial vertex in the subgraph induced by  $\{v_1, \dots, v_i\}$



$\{b; a; c; d\}$  is a simplicial elimination ordering.

- An undirected graph without self-loops is chordal if and only if it has a simplicial elimination ordering. (Dirac)

# Algorithm



# The maximum cardinality search algorithm(MCS)

**procedure** MCS

1     *input:*  $G = (V, E)$

2     *output:* a simplicial elimination ordering  $\sigma = v_1, \dots, v_n$

3     **For all**  $v \in V$  **do**  $\lambda(v) \leftarrow 0$

4     **For**  $i \leftarrow 1$  **to**  $|V|$  **do**

5         **let**  $v \in V$  **be** a vertex such that  $\forall u \in V, \lambda(v) \geq \lambda(u)$  **in**

6              $\sigma(i) \leftarrow v$

7             **For all**  $u \in V \cap N(v)$  **do**  $\lambda(u) \leftarrow \lambda(u) + 1$

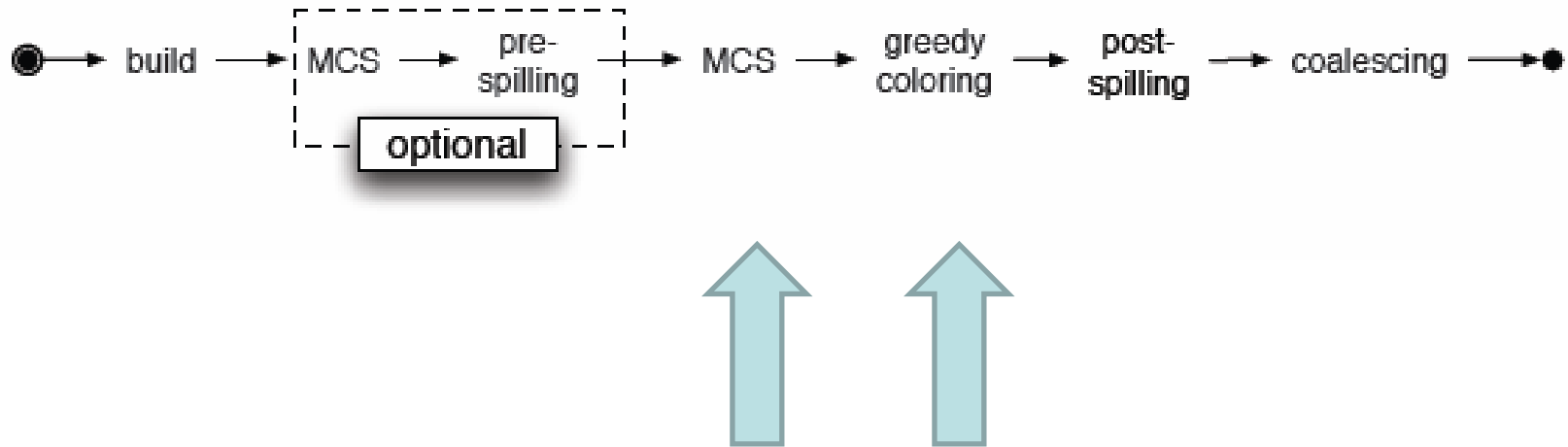
8              $V \leftarrow V - \{v\}$

# The greedy coloring algorithm

**procedure greedy coloring**

1     *input:*  $G = (V, E)$ , a sequence of vertices  $\nu$   
2     *output:* a mapping  $m$ ,  $m(v) = c$ ,  $0 \leq c \leq \Delta(G) + 1$ ,  $v \in V$   
3     **For all**  $v \in \nu$  **do**  $m(v) \leftarrow \perp$   
4     **For**  $i \leftarrow 1$  **to**  $|\nu|$  **do**  
5         **let**  $c$  **be** the lowest color not used in  $N(\nu(i))$  **in**  
6              $m(\nu(i)) \leftarrow c$

# Algorithm



# Post spilling

- polynomial algorithm--  $O(V^k)$
- the greedy coloring tends to use the lower colors first.

# Coalescing

- For each instruction  $a := b$ ,  
look for a color  $c$  not used in  $N(a) \cup N(b)$ ,
- If such a color exists, then the temporaries  $a$  and  $b$  are coalesced into a single register with the color  $c$ .

# Coalescing

**procedure** coalescing

```
1   input: list  $l$  of copy instructions,  $G = (V, E)$ ,  $K$ 
2   output:  $G'$ , the coalesced graph  $G$ 
3   let  $G' = G$  in
4   for all  $x := y \in l$  do
5       let  $S_x$  be the set of colors in  $N(x)$ 
6       let  $S_y$  be the set of colors in  $N(y)$ 
7       if there exists  $c, c < K, c \notin S_x \cup S_y$  then
8           let  $xy, xy \notin V$  be a new node
9           add  $xy$  to  $G'$  with color  $c$ 
10          make  $xy$  adjacent to every  $v, v \in N(x) \cup N(y)$ 
11          replace occurrences of  $x$  or  $y$  in  $l$  by  $xy$ 
12          remove  $x$  from  $G'$ 
13          remove  $y$  from  $G'$ 
```

# Pre-spilling

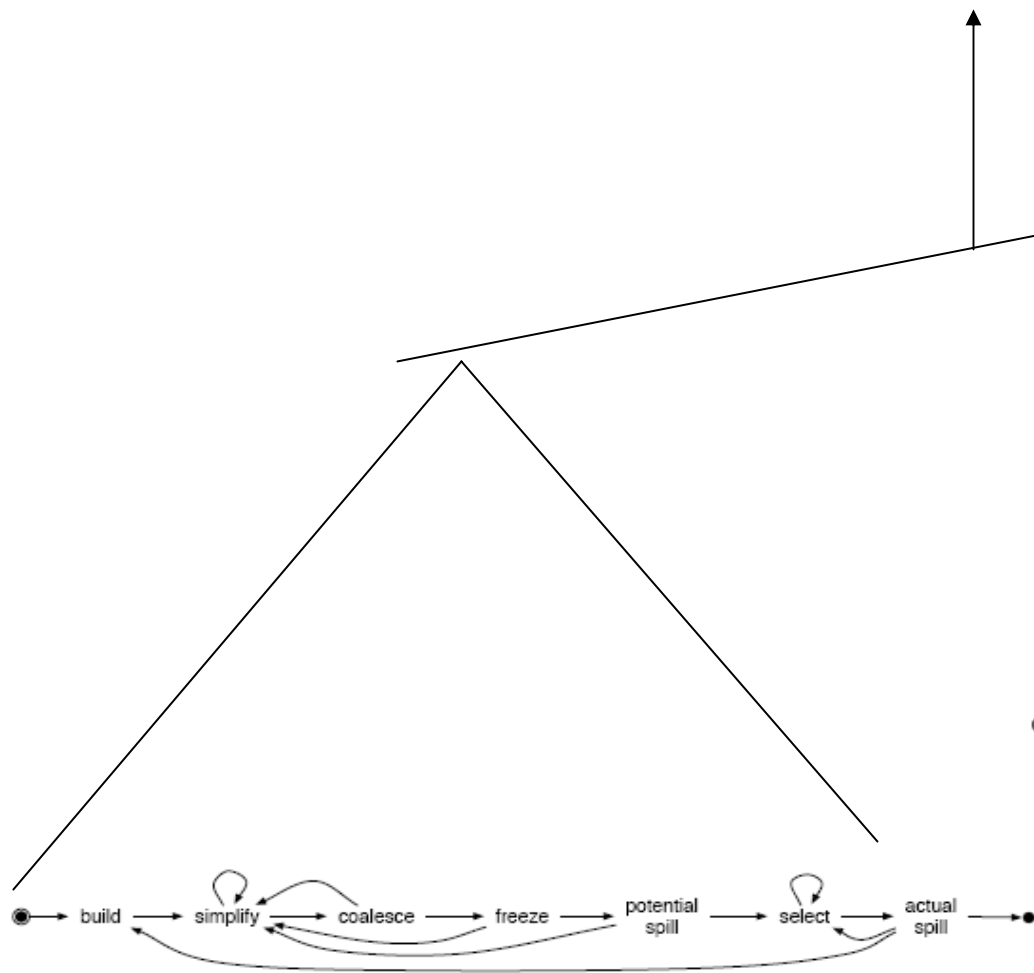
- Maintains  $k$ -colorable property
- removes nodes to bring the size of the largest clique down to the number of available colors

**procedure maximalCl**

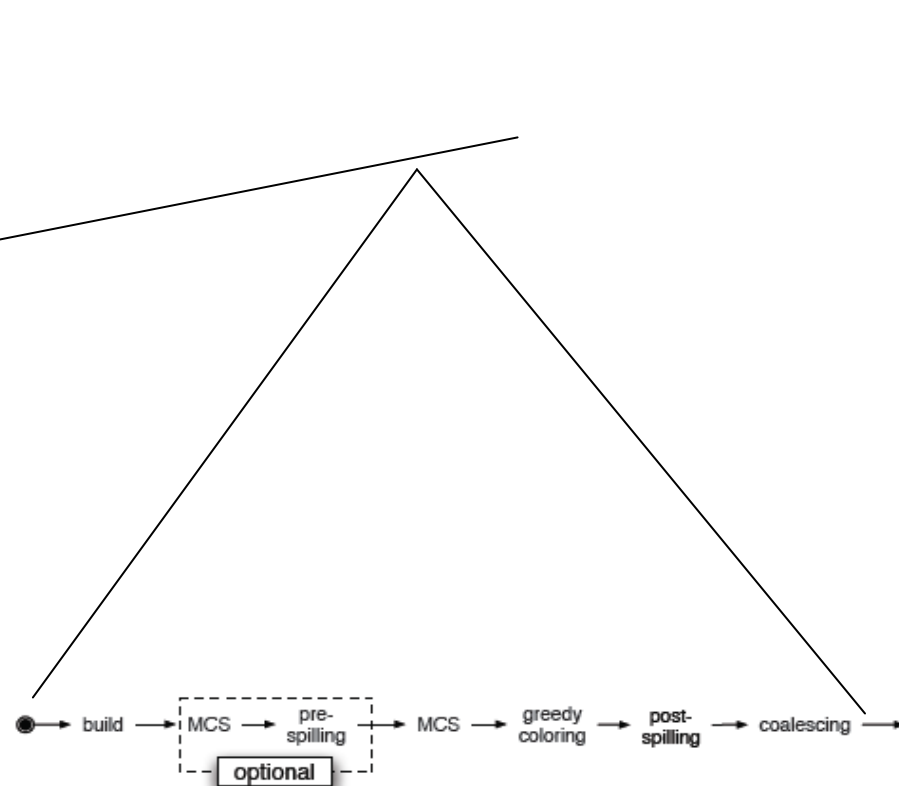
1     *input:*  $G = (V, E)$   
2     *output:* a list of cliques  $\xi = \langle Q_1, Q_2, \dots, Q_n \rangle$   
3      $\sigma \leftarrow \mathbf{MCS}(G)$   
4     **For**  $i \leftarrow 1$  **to**  $n$  **do**  
5         **Let**  $v \leftarrow \sigma[i]$  **in**  
6              $Q_i \leftarrow \{v\} \cup \{u \mid (u, v) \in E, u \in \{\sigma[1], \dots, \sigma[i-1]\}\}$

**procedure pre-spilling**

1     *input:*  $G = (V, E)$ , a list of subgraphs of  $G$ :  $\xi = \langle Q_1, Q_2, \dots, Q_n \rangle$ ,  
          a number of available colors  $K$ , a mapping  $\omega$   
2     *output:* a  $K$ -colorable subgraph of  $G$   
3      $R_1 = Q_1; R_2 = Q_2; \dots R_n = Q_n$   
4     **while** there is  $R_i$  with more than  $K$  nodes **do**  
5         **let**  $v \in R_i$  **be** a vertex such that  $\forall u \in R_i, \omega(v) \geq \omega(u)$  **in**  
6             remove  $v$  from all the graphs  $R_1, R_2, \dots, R_n$   
7     **return**  $R_1 \cup R_2 \cup \dots \cup R_n$



iterated register coalescing algorithm.



New algorithm

# Caveats

- entire run-time library of the standard Java 1.5 distribution
- Loop variables – spilling ?
- JoeQ compiler(John Whaley)
- IR- a set of instructions (quads- operator + 4 operands) organized into a control flow graph
- control flow can potentially exit from the middle of a basic block

# Register Allocation after Classical SSA Elimination is NP-complete

Sumit Kumar Jha

Some examples and graphics are borrowed from the FOSSACS'06 paper and talk by the author Fernando M Q Pereira.

# Talk Outline

- Background on “old” complexity results in register allocation
- SSA form and the register allocation problem
- Circular Graphs and Post-SSA Circular Graphs
- The Reduction of coloring Circular graphs to register allocation after SSA elimination.
- The Big Picture

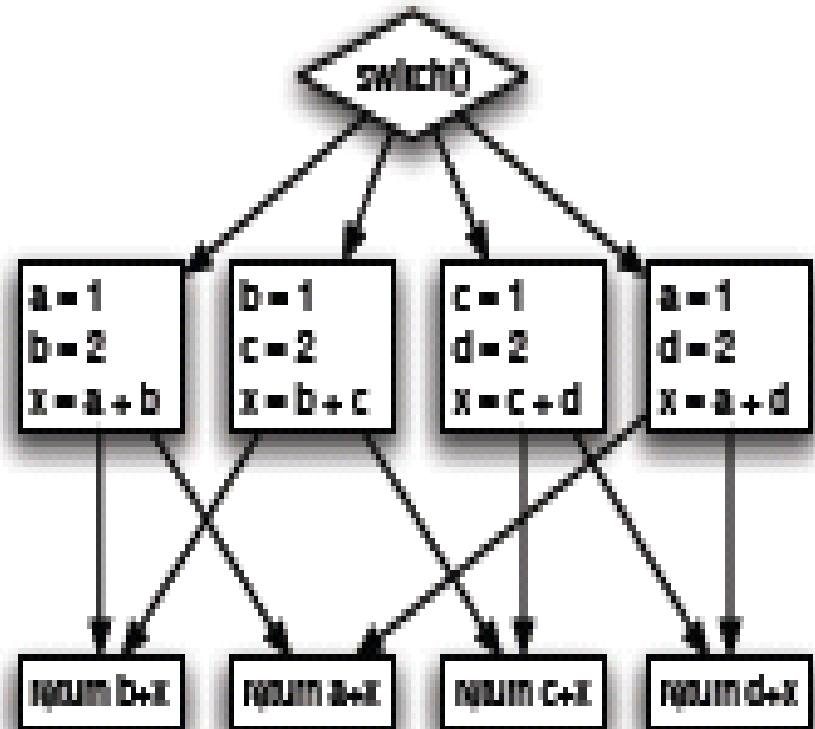
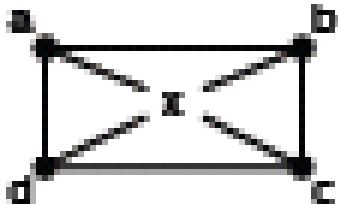
# Core register allocation problem

- Instance: a program  $P$  and a number  $N$  of available registers.
- Problem: Can each of the temporaries of  $P$  be mapped to one of the  $N$  registers such that temporary variables with interfering live ranges are assigned to different registers?

**NP Complete**

# Chaitin's Proof

- Chaitin et al. showed in 1981 that the core register allocation problem is **NP-complete**
- They used a reduction from the **graph coloring** problem.
  - The essence of Chaitin et al.'s proof is that every graph is the interference graph of some program.



# Static Single Assignment (SSA)

- SSA form. Static single assignment (SSA) form is an intermediate representation used in many compilers like gcc 4.
- If a program is in SSA form, then every variable is assigned exactly once, and each use refers to exactly one definition.

# Register Allocation for SSA Programs

- Bouchez and Hack (2006) proved the result that **strict programs** in **SSA** form have chordal interference graphs.
- Chordal graphs can be colored in **polynomial time!**
- **Strict** program: Every path from the initial block to the use of a variable  $v$  passes through a definition of  $v$ .

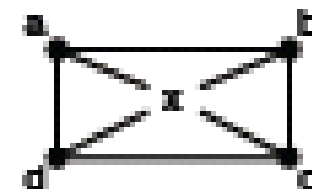
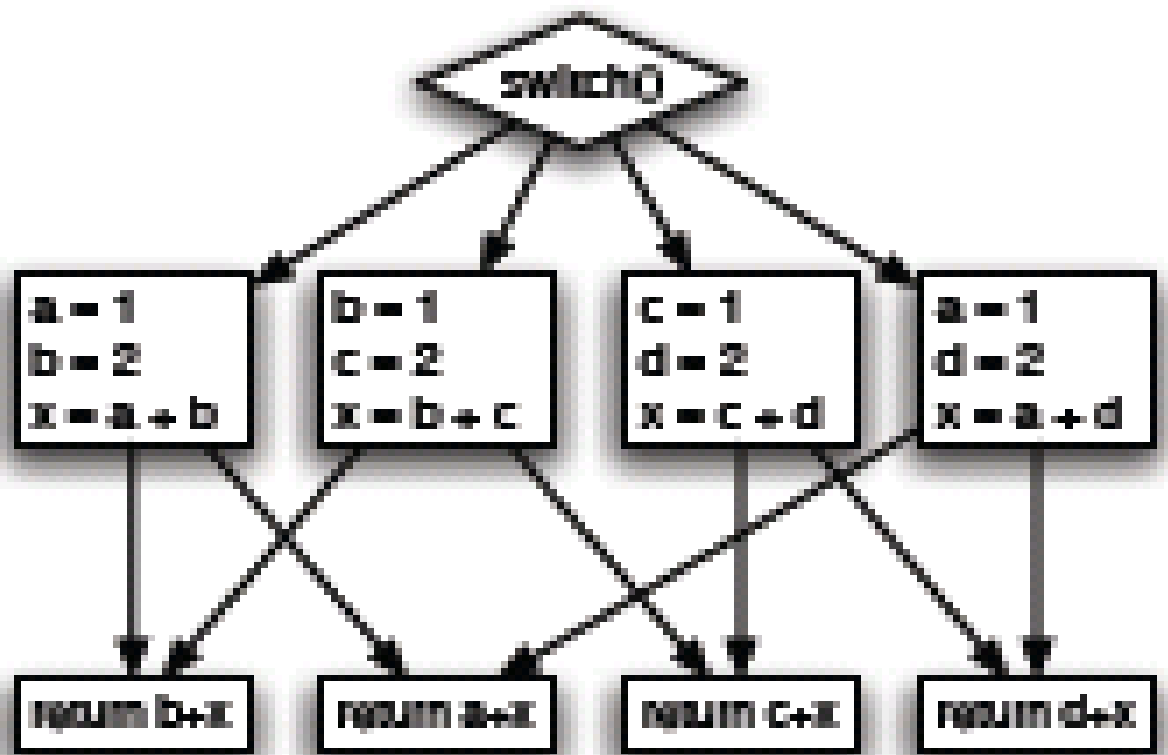
# SSA and Chordal interference graphs

- The core register allocation problem is **NP-complete**. [Chaitin 1981]
- Also, a compiler can transform a given program into SSA form in cubic time.
- We can color a chordal graph in **linear time** so we can solve the core register allocation problem for programs in SSA form in linear time.
- A contradiction!! Not really.

# Register Allocation after conversion to SSA may be easier.

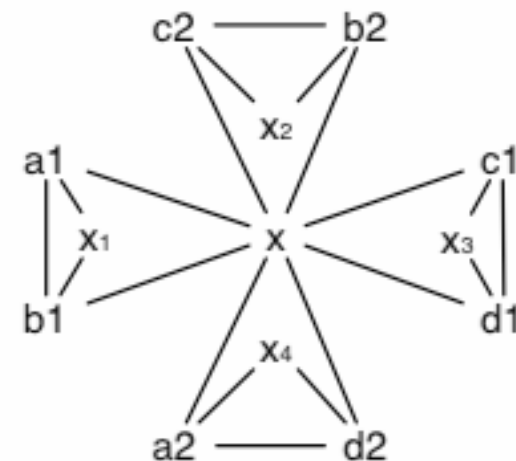
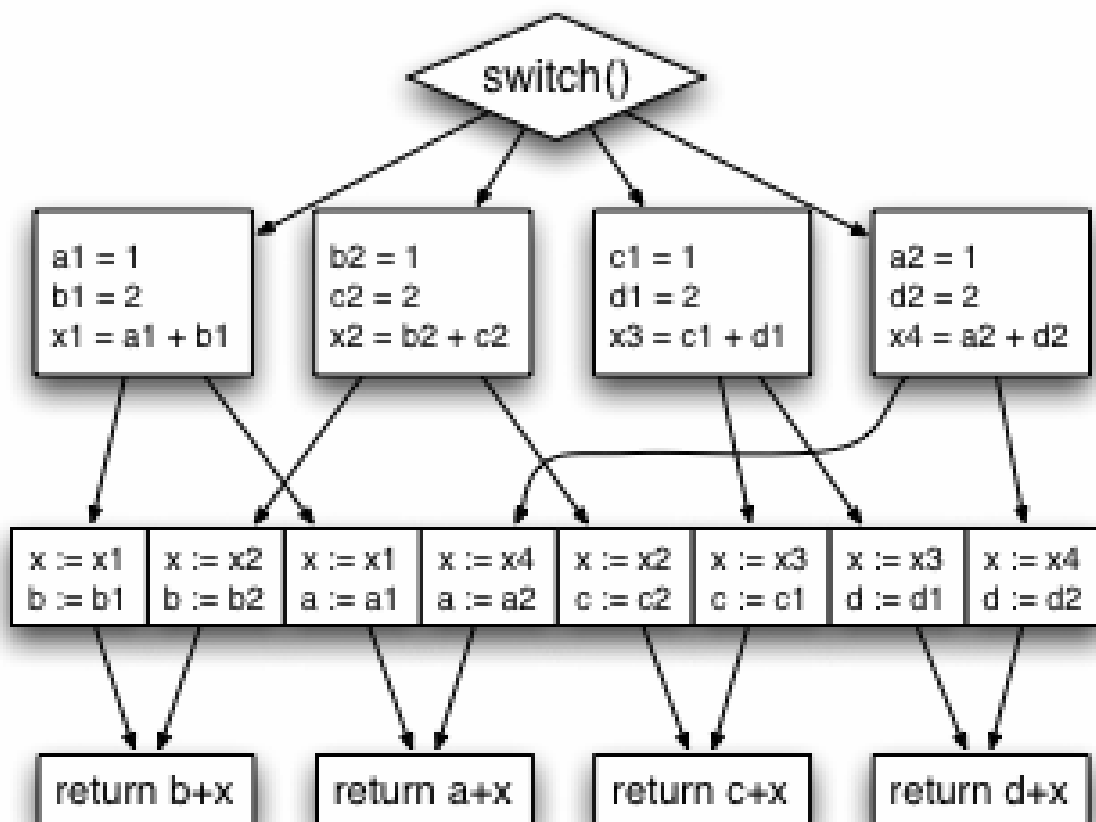
- Given a program  $P$ , its SSA-form version  $P_0$ , and a number of registers  $K$ ,
- the core register allocation problem  $(P, K)$  is not equivalent to  $(P_0, K)$ .
  - we can map a  $(P; K)$ -solution to a  $(P_0; K)$ -solution,
  - we can **not necessarily** map a  $(P_0; K)$ -solution to a  $(P; K)$ -solution.
    - The SSA transformation splits the live ranges of temporaries in  $P$  in such a way that  $P_0$  may need fewer registers than  $P$ .

# Why Chatin's proof does not work after SSA elimination?



(a) Chaitin et al.'s program to represent C4. (b) The interference graph of the original program

# Chatin's proof does not work after SSA elimination



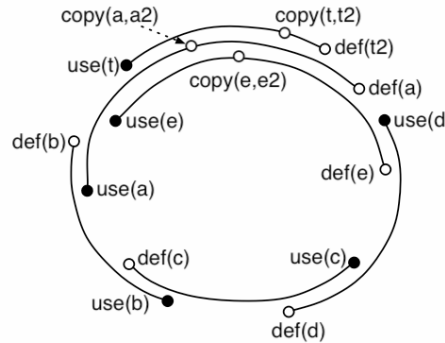
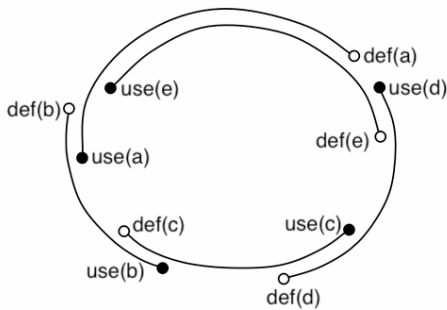
# What we learnt till now...

- Core Register Allocation is NP Complete.
- Chaitin's NP completeness proof does not work after classical SSA elimination.
- The solution obtained by analyzing a SSA form program [chordal graphs] in polynomial time can not be mapped back to the original program.
- So, the question if Register Allocation after SSA elimination is NP complete remains open.

# The current approach

- The authors identify a subset of graphs called circular graphs such that
  - They are NP-hard to color
  - It is possible to write a program  $P(C)$  such that its core register allocation uses  $N+a$  registers after SSA elimination if and only if the circular graph  $C$  is  $N$ -colorable.
- The authors introduce an intermediate step SSA-Circular graphs to move from the circular graph  $C$  to the program  $P(C)$

# The current approach



```
int m(int a1,int e1, int t1,int i1){
  int a = a1;
  int e = e1;
  int t = t1;
  int i = i1;
  while(i < 100) {
    int i2 = i + 1;
    << main loop >>
    i = i2;
    a = a2;
    t = t2;
    e = e2;
  }
  return a;
}
```

Circular-arc  
graph

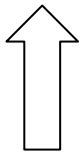
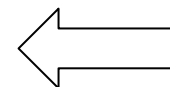
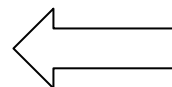
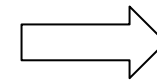
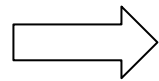
Post-SSA  
graph

Simple  
Post-SSA  
Program

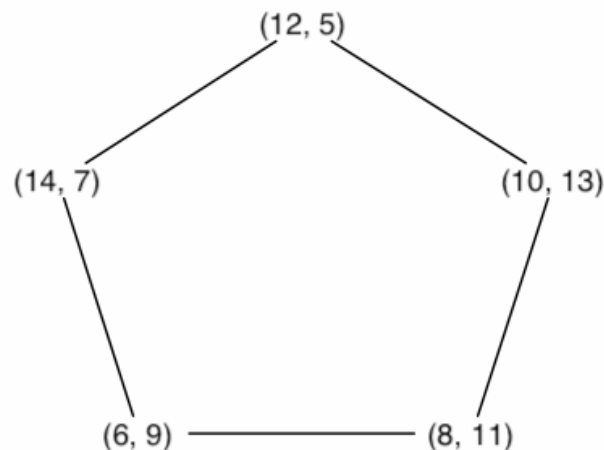
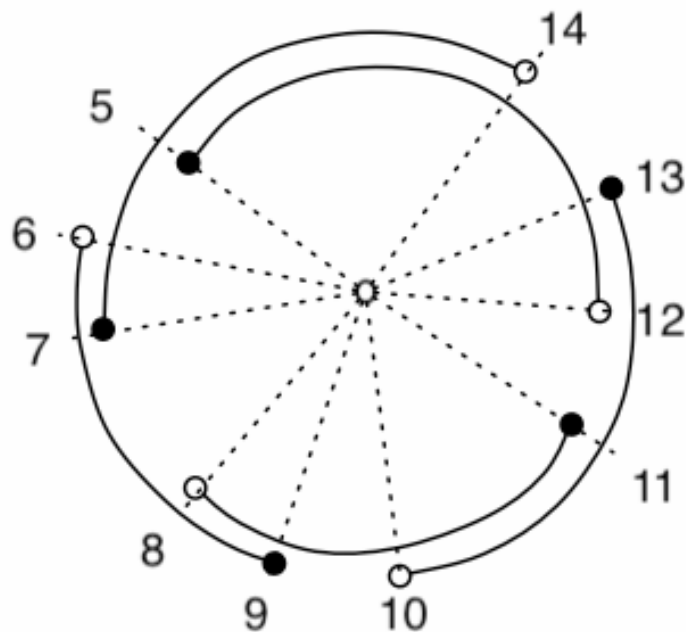
Arcs to arcs

Register  
To colors

Register  
Assignment



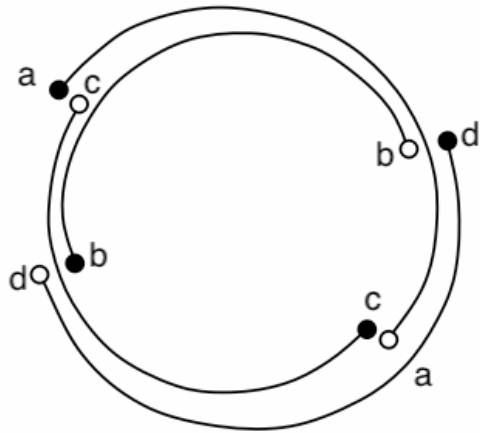
# Circular Graphs and SSA-Circular Graphs - I



**Theorem:** Finding a minimal coloring for a circular-arc graph is NP-complete.

# Circular Graphs and SSA-Circular Graphs - II

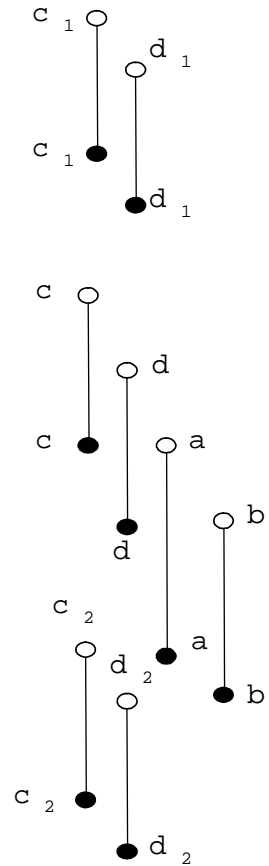
SSA - form



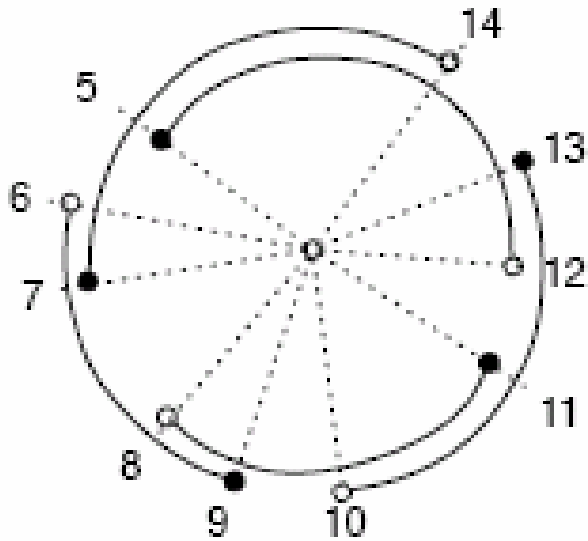
```
d1 = ... ;
c1 = ... ;
```

$$\begin{pmatrix} d \\ c \end{pmatrix} = \begin{pmatrix} d_1, d_2 \\ c_1, c_2 \end{pmatrix}$$

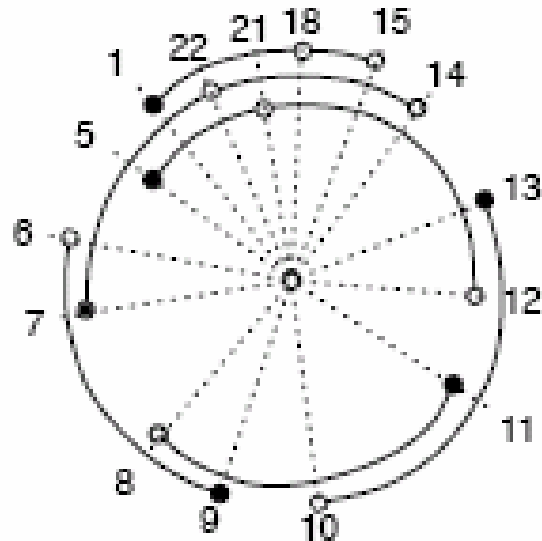
```
a = c;
b = d;
c2 = a+1;
d2 = b+1;
```



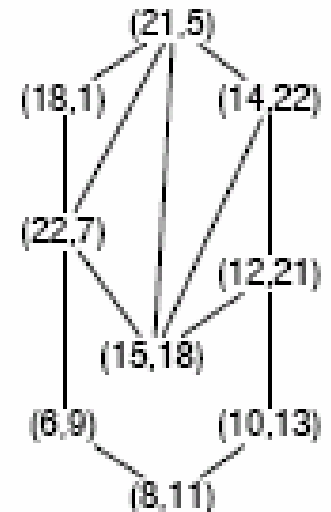
# SSA and Chordal interference graphs



(a)



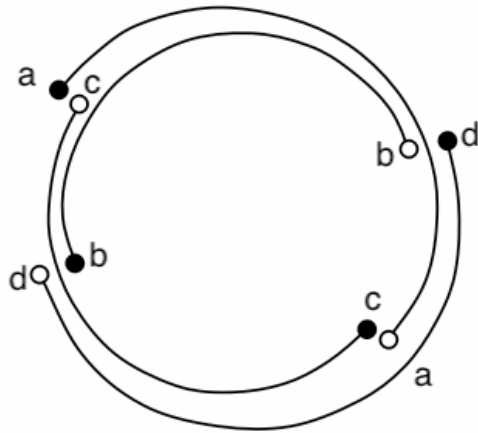
(b)



(c)

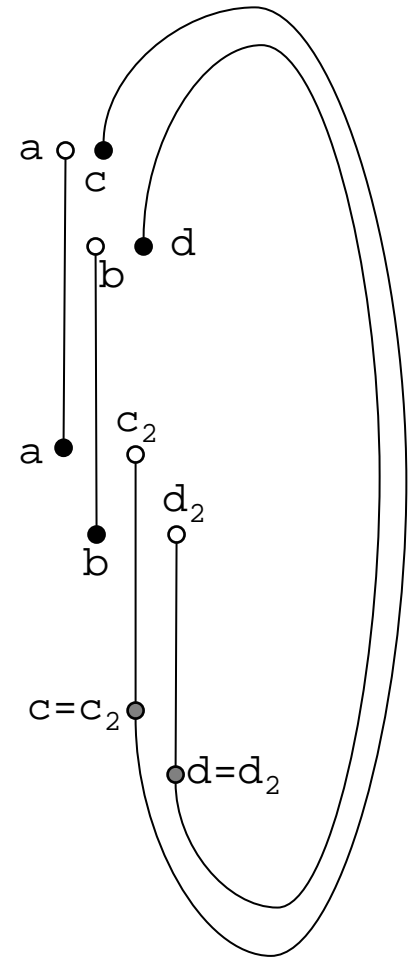
(a)  $C_5$  represented as a set of intervals. (b) The set of intervals that represent  $W = F(C_5; 3)$ . (c)  $W$  represented as a graph.

# Circular Graphs and SSA-Circular Graphs - III



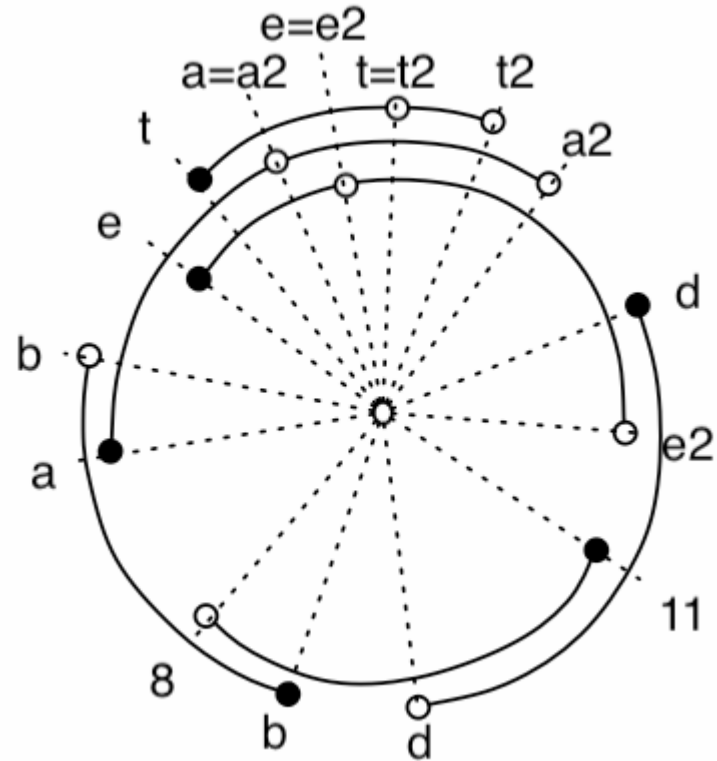
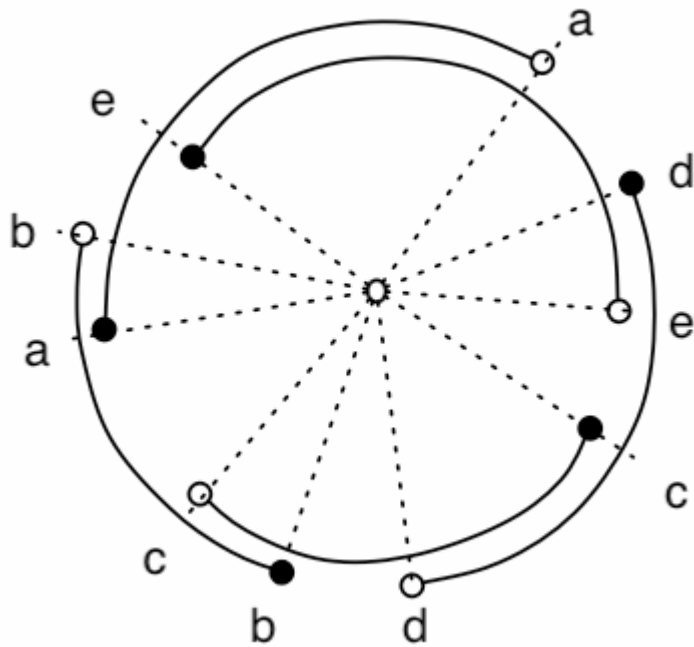
## Post-SSA program

```
1) int d1 = ...;
2) int c1 = ...;
3) d = d1;
4) c = c1;
5) while ( ... ) {
6)     int a = c;
7)     int b = d;
8)     c2 = a+1;
9)     d2 = b+1;
10)    d = d2;
11)    c = c2;
12) }
```



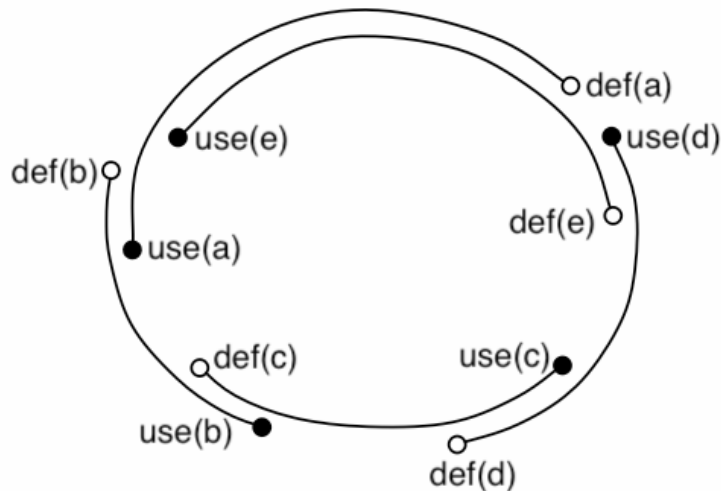
Post-SSA graph

# Circular Graphs and SSA-Circular Graphs - IV



**Result In Paper: Circular-arc graph has N coloring iff SSA-graph has N coloring.**

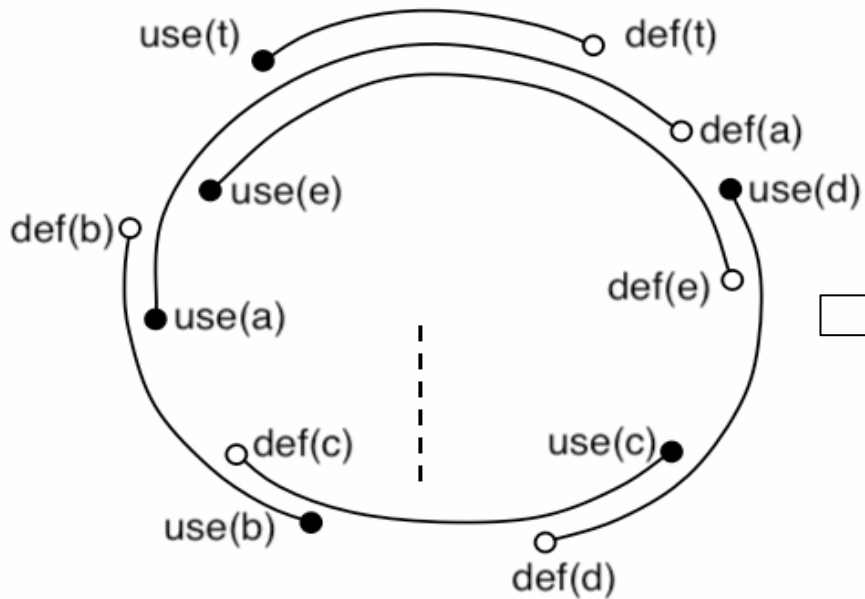
# The Reduction - I



```
int m(int a,int e,int i){
    while(i < 100) {
        i = i + 1;
        if (e > 10) break;
        int b = i + 11;
        if (a > 11) break;
        int c = i + 12;
        if(b > 12) break;
        int d = i + 13;
        if(c > 13) break;
        e = i + 14;
        if(d > 14) break;
        a = i + 15;
    }
    return a;
}
```

Given a circular graph, can we find a program such that the program needs  $K$  registers iff the circular graph is  $k$ -colorable?

# The Reduction - II



```
int m(int a,int e, int t ,int i){
    while(i < 100) {
        i = i + 1;
        if(t > 9) break;
        if(e > 10) break;
        int b = i + 11;
        if (a > 11) break;
        int c = i + 12;
        if(b > 12) break;
        int d = i + 13;
        if(c > 13) break;
        e = i + 14;
        if(d > 14) break;
        a = i + 15;
        t = i + 16;
    }
    return a;
}
```

# After SSA-elimination ...

```
int m(int a1, int e1, int t1, int i1) {
```

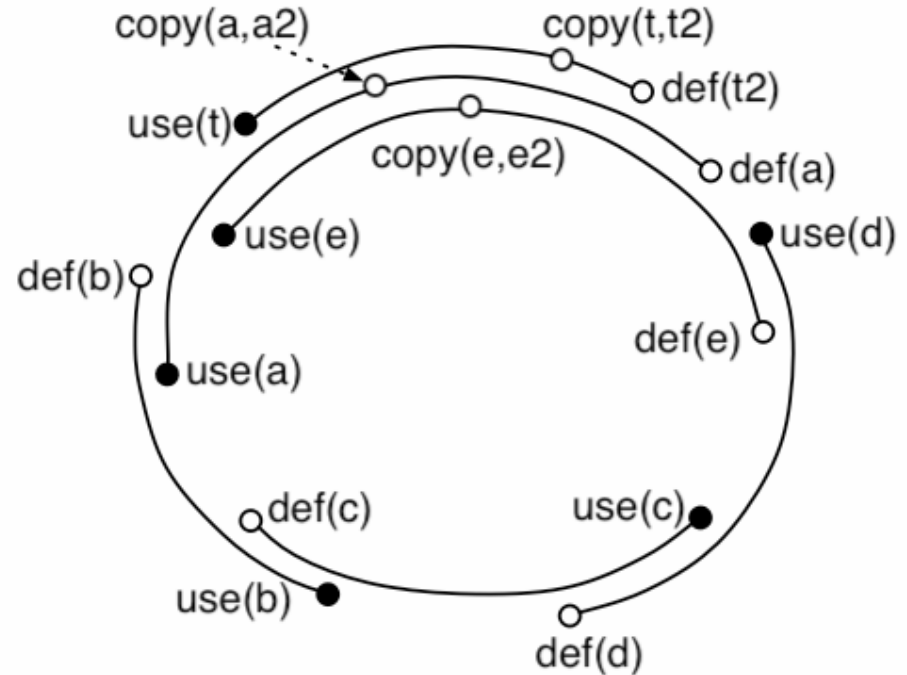
```
int a = a1;  
int e = e1;  
int t = t1;  
int i = i1;
```

```
while(i > 10) {  
    int i2 = i + 1;  
    << main loop >>
```

```
    i = i2;  
    a = a2;  
    t = t2;  
    e = e2;
```

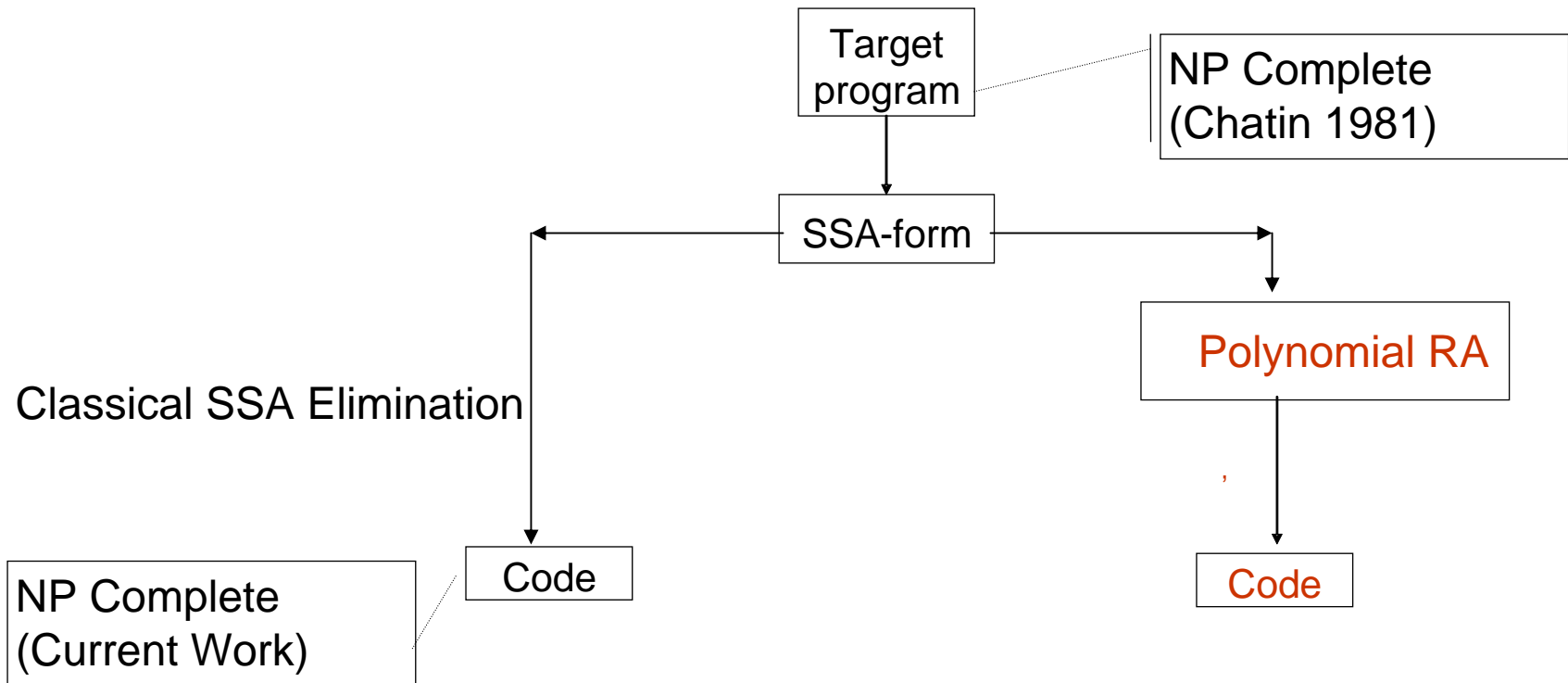
```
}  
return a;
```

```
}
```



N + a register assignment  
⇒ N coloring

# Current View – Register Allocation



Questions?

Thanks 😊