Compilation

0368-3133

Tutorial 12:

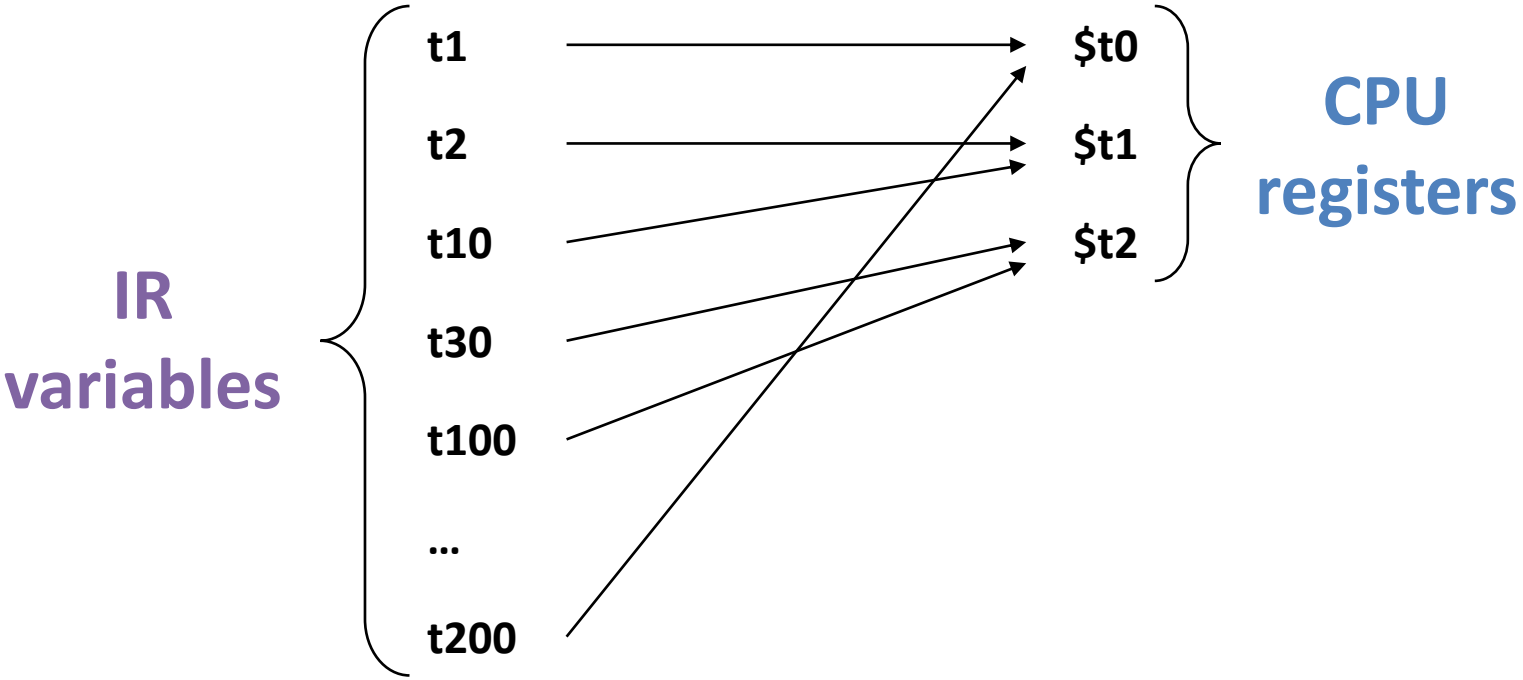
Register Allocation

Today's Plan

- **Part I:** Register allocation for the project
  - We may give up and abort
- **Part II:** General register allocation
  - Spilling & coalescing
  - Never give up!

Register Allocation (in the project)

- Typically, more IR variables than CPU registers
- Reduce the number of IR variables:
  - Without affecting the behavior of the program



Register Allocation

For each function:

1. Construct the CFG (from the IR)
2. Run liveness analysis
3. Construct the interference graph
4. Compute a *k-coloring* of the graph
5. Use the coloring to build the required mapping

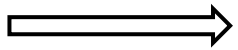
Register Allocation

For each function:

1. **Construct the CFG (from the IR)**
2. Run liveness analysis
3. Construct the interference graph
4. Compute a *k-coloring* of the graph
5. Use the coloring to build the required mapping

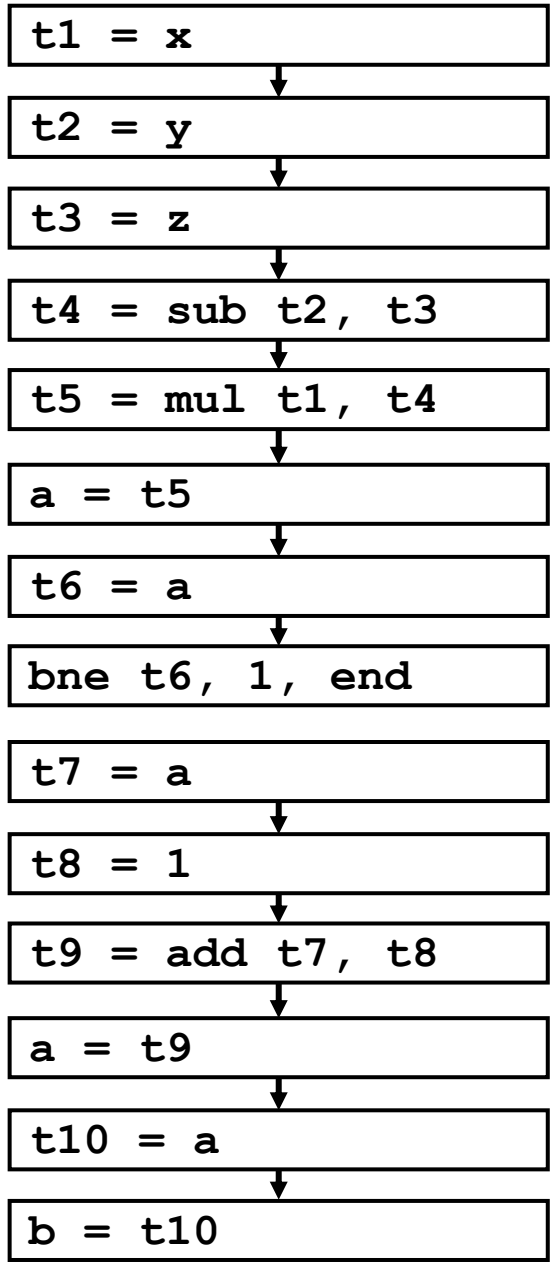
Control Flow Graph

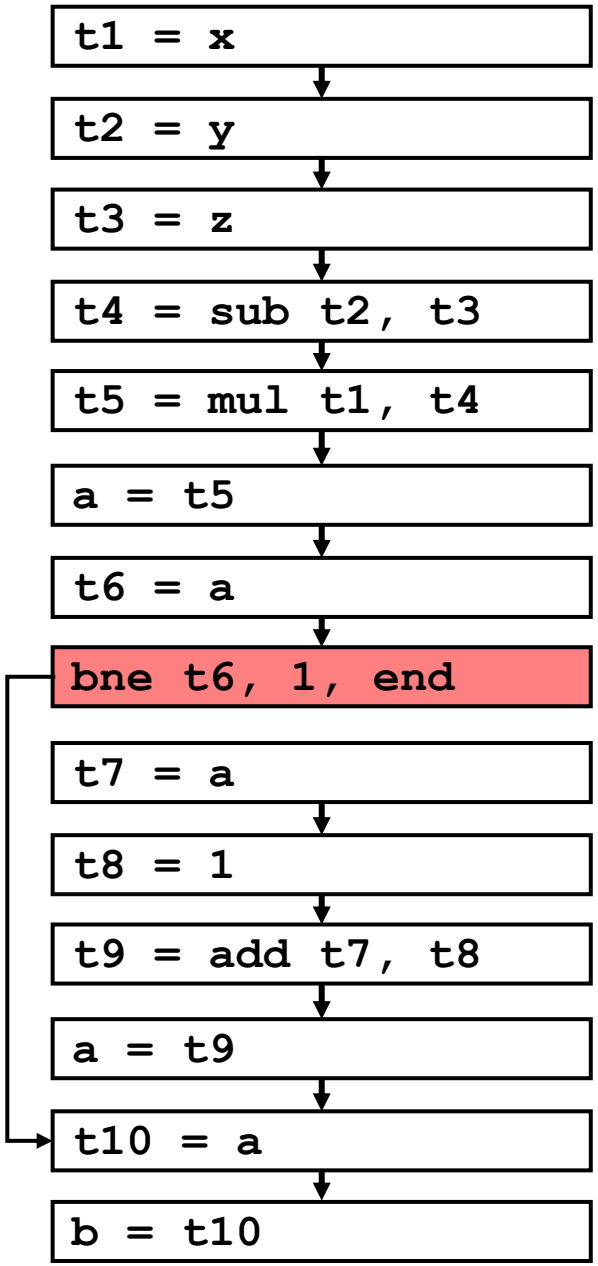
```
void f(int x, int y, int z) {
  int a = x * (y - z)
  if (a) {
    a = a + 1;
  }
  int b = a;
}
```

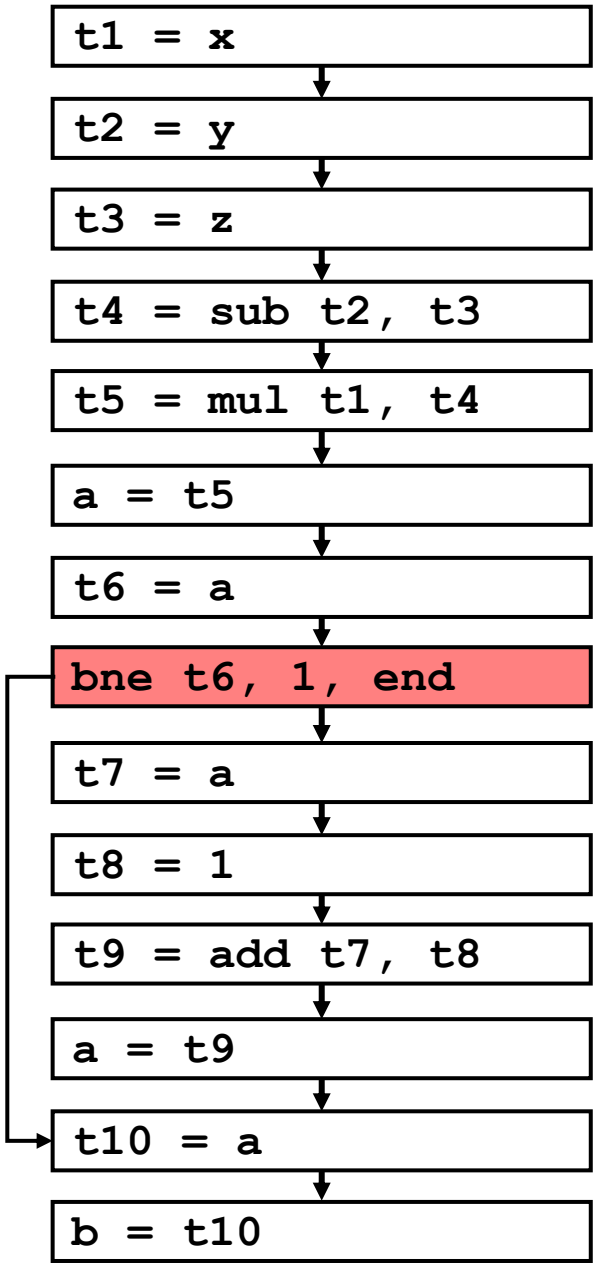


```
t1 = x
t2 = y
t3 = z
t4 = sub t2, t3
t5 = mul t1, t4
a = t5
t6 = a
bne t6, 1, end
t7 = a
t8 = 1
t9 = add t7, t8
a = t9
end:
t10 = a
b = t10
```

```
t1 = x
t2 = y
t3 = z
t4 = sub t2, t3
t5 = mul t1, t4
a = t5
t6 = a
bne t6, 1, end
t7 = a
t8 = 1
t9 = add t7, t8
a = t9
t10 = a
b = t10
```


t1 = x
t2 = y
t3 = z
t4 = sub t2, t3
t5 = mul t1, t4
a = t5
t6 = a
bne t6, 1, end
t7 = a
t8 = 1
t9 = add t7, t8
a = t9
t10 = a
b = t10


t1 = x
t2 = y
t3 = z
t4 = sub t2, t3
t5 = mul t1, t4
a = t5
t6 = a
bne t6, 1, end
t7 = a
t8 = 1
t9 = add t7, t8
a = t9
t10 = a
b = t10


t1 = x
t2 = y
t3 = z
t4 = sub t2, t3
t5 = mul t1, t4
a = t5
t6 = a
bne t6, 1, end
t7 = a
t8 = 1
t9 = add t7, t8
a = t9
t10 = a
b = t10

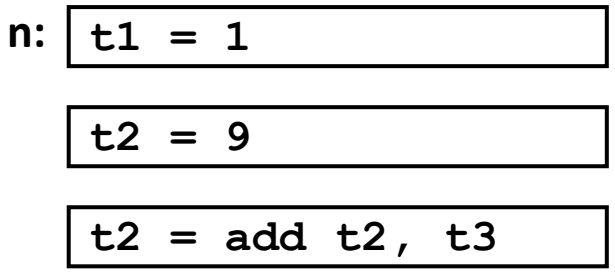
Register Allocation

For each function:

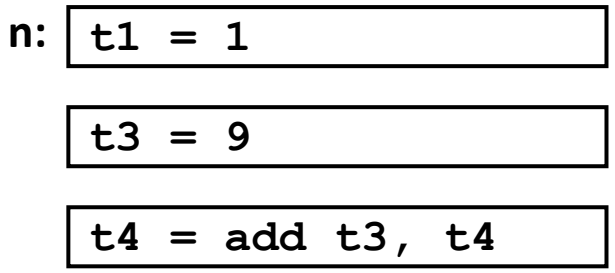
1. Construct the CFG (from the IR)
2. **Run liveness analysis**
3. Construct the interference graph
4. Compute a *k-coloring* of the graph
5. Use the coloring to build the required mapping

Liveness Analysis

- Determine *live variables* at each program location
- Variable x is live at location n if:
  - There is a path from n where x is read before it is overwritten



t3 is **live**



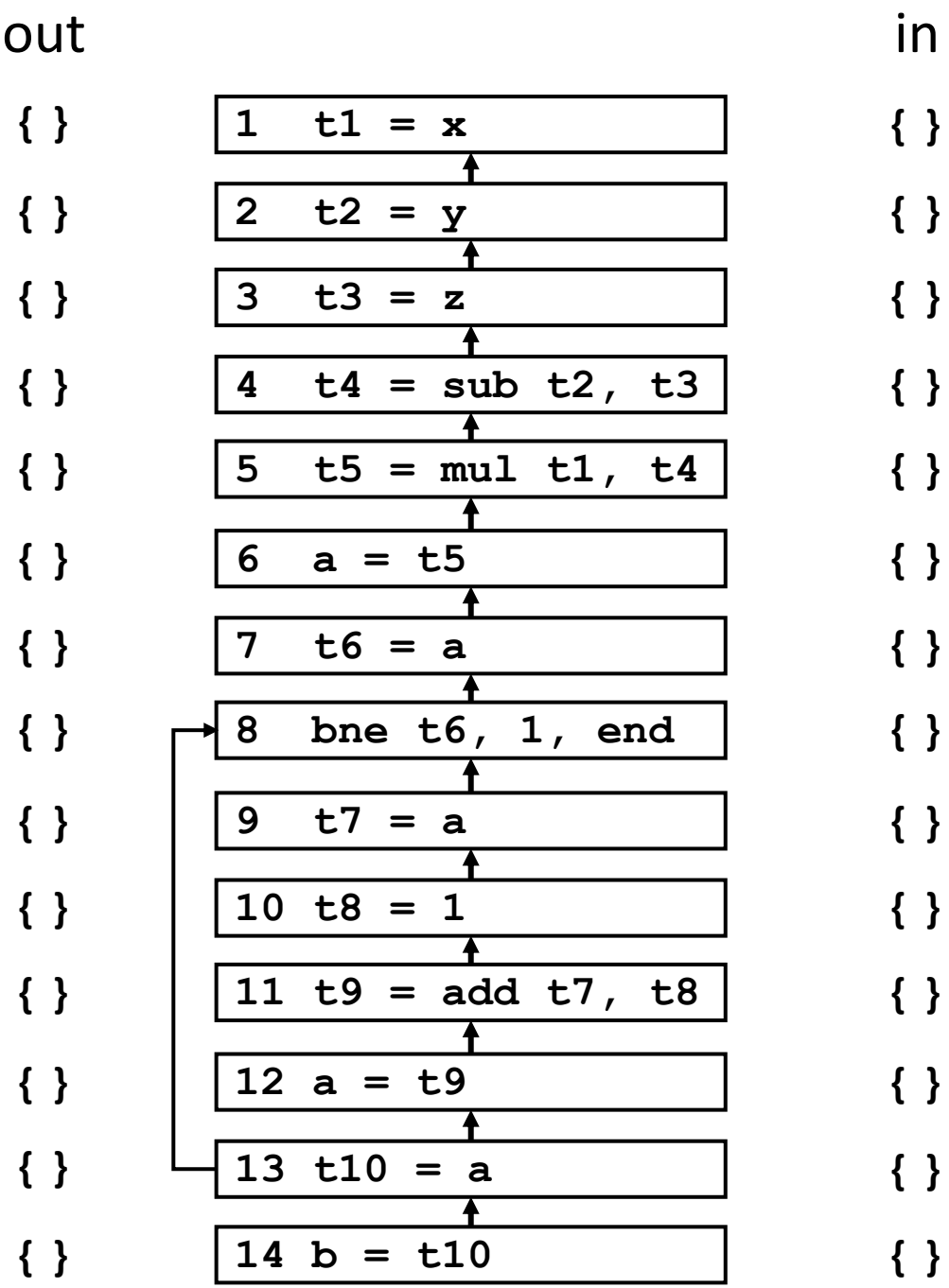
t3 is **dead**

Liveness Analysis

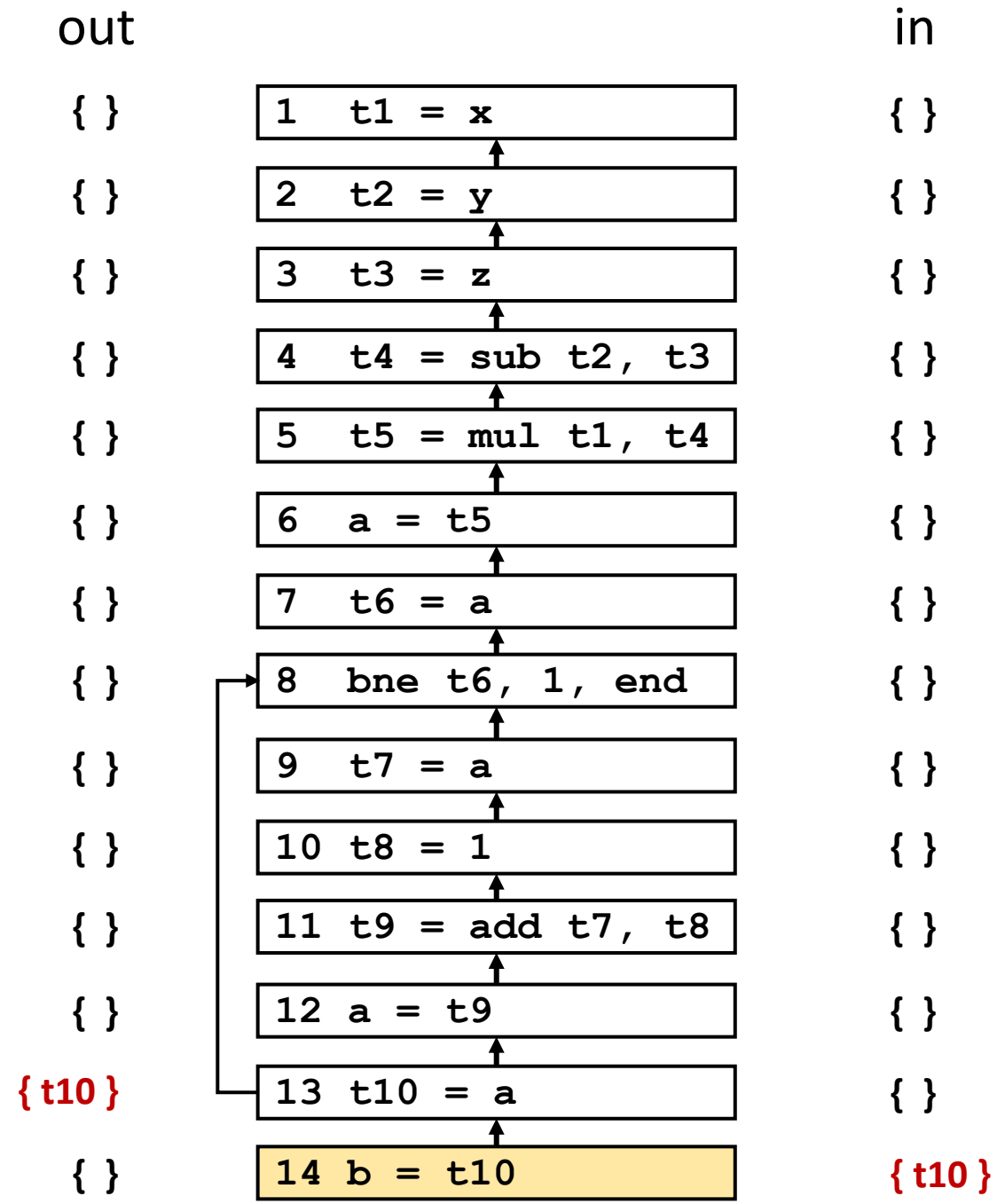
- Dom = $\wp$(Temps) (In our context, in general Dom = $\wp$(Vars))
- $\sqsubseteq = \subseteq$, $\sqcup = \cup$
- D = backward
- I = $\emptyset$

Statement	**$kill_i$**	**gen_i**
x = expr	{x}	Temporaries used in exp
stmt	$\emptyset$	Temporaries used in stmt

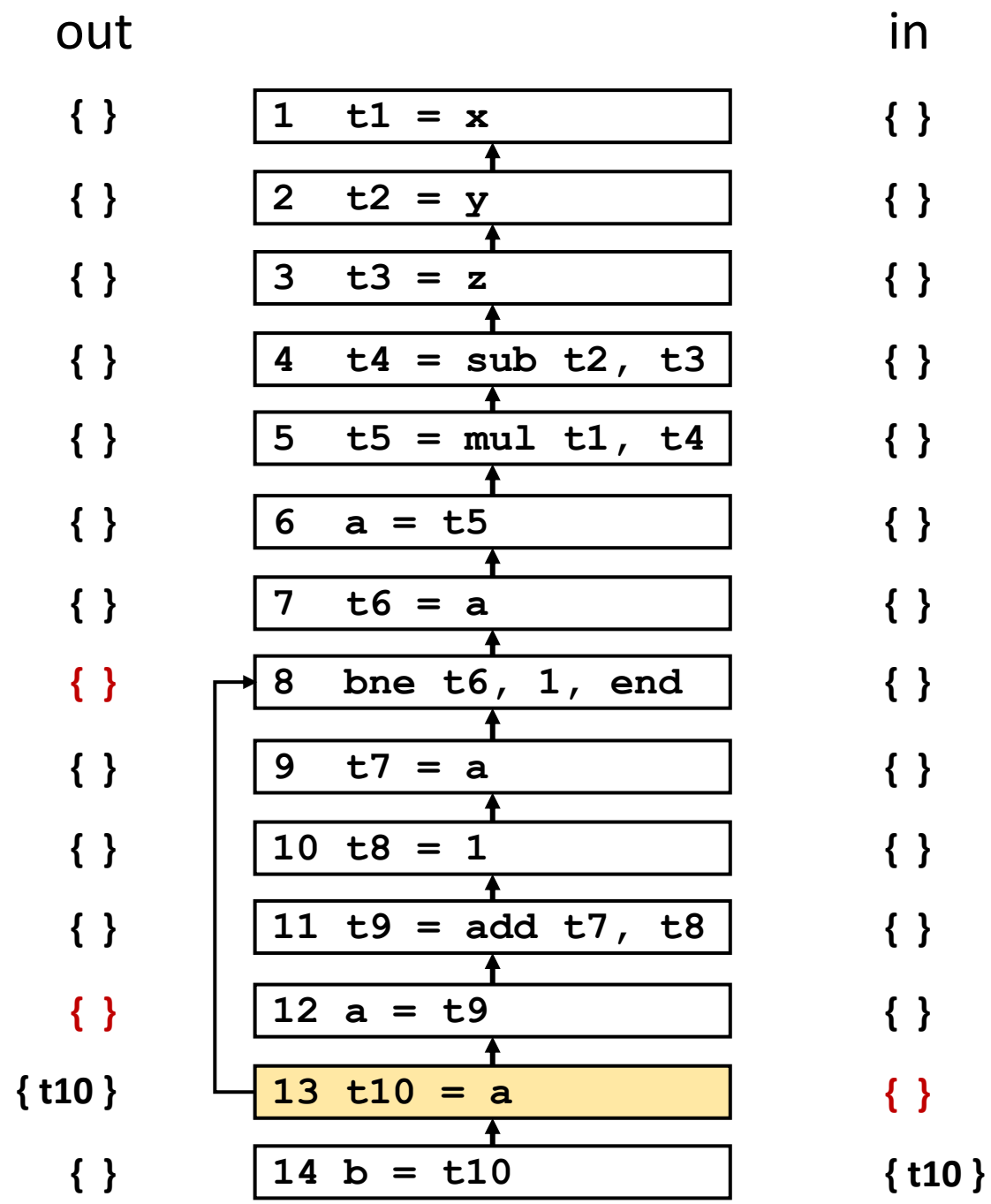
Work list = {14}


out
in
{ }
1 t1 = x
2 t2 = y
3 t3 = z
4 t4 = sub t2, t3
5 t5 = mul t1, t4
6 a = t5
7 t6 = a
8 bne t6, 1, end
9 t7 = a
10 t8 = 1
11 t9 = add t7, t8
12 a = t9
13 t10 = a
14 b = t10

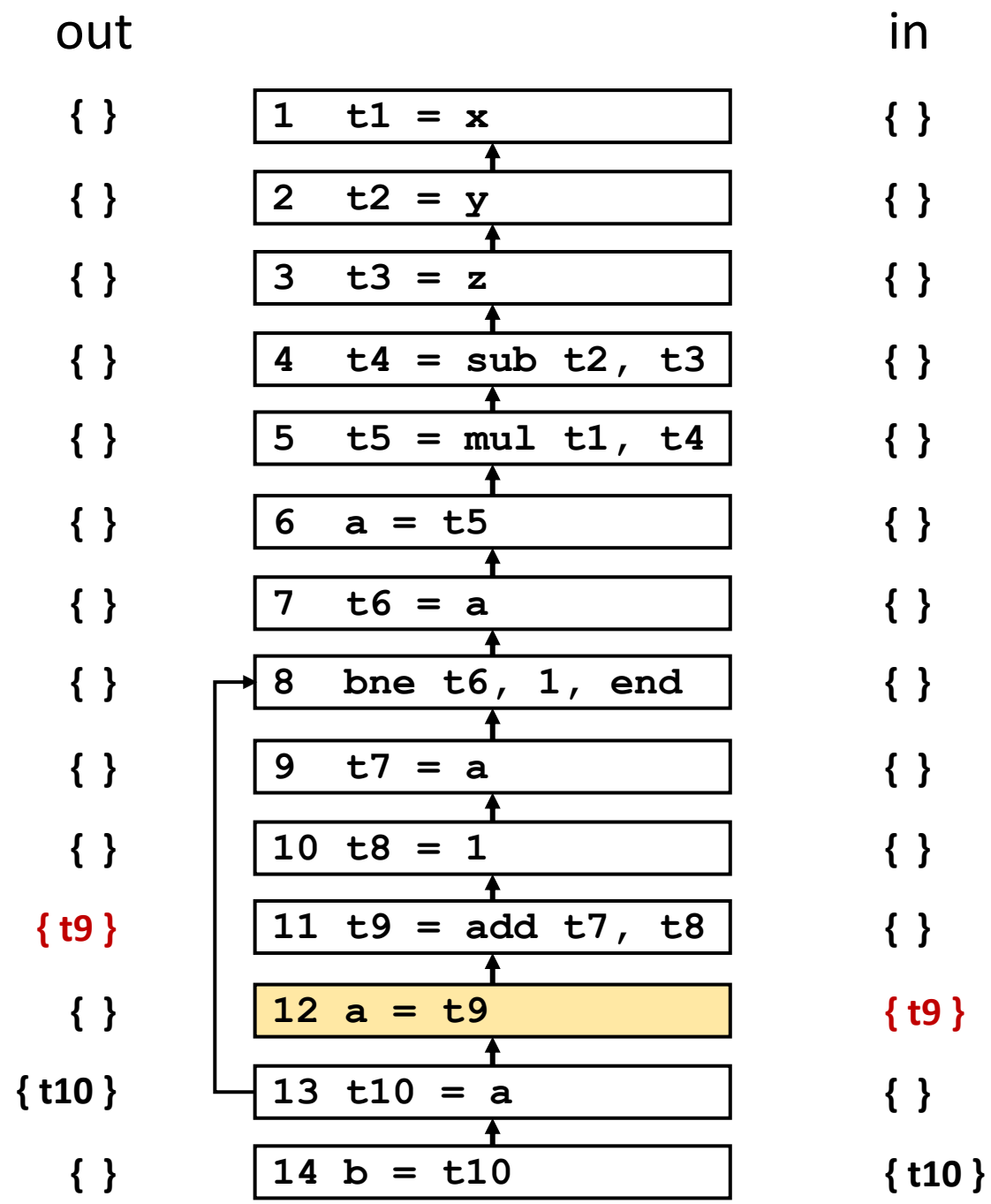
Work list = {13}


out
in
1 t1 = x
2 t2 = y
3 t3 = z
4 t4 = sub t2, t3
5 t5 = mul t1, t4
6 a = t5
7 t6 = a
8 bne t6, 1, end
9 t7 = a
10 t8 = 1
11 t9 = add t7, t8
12 a = t9
13 t10 = a
14 b = t10
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ t10 }
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ t10 }

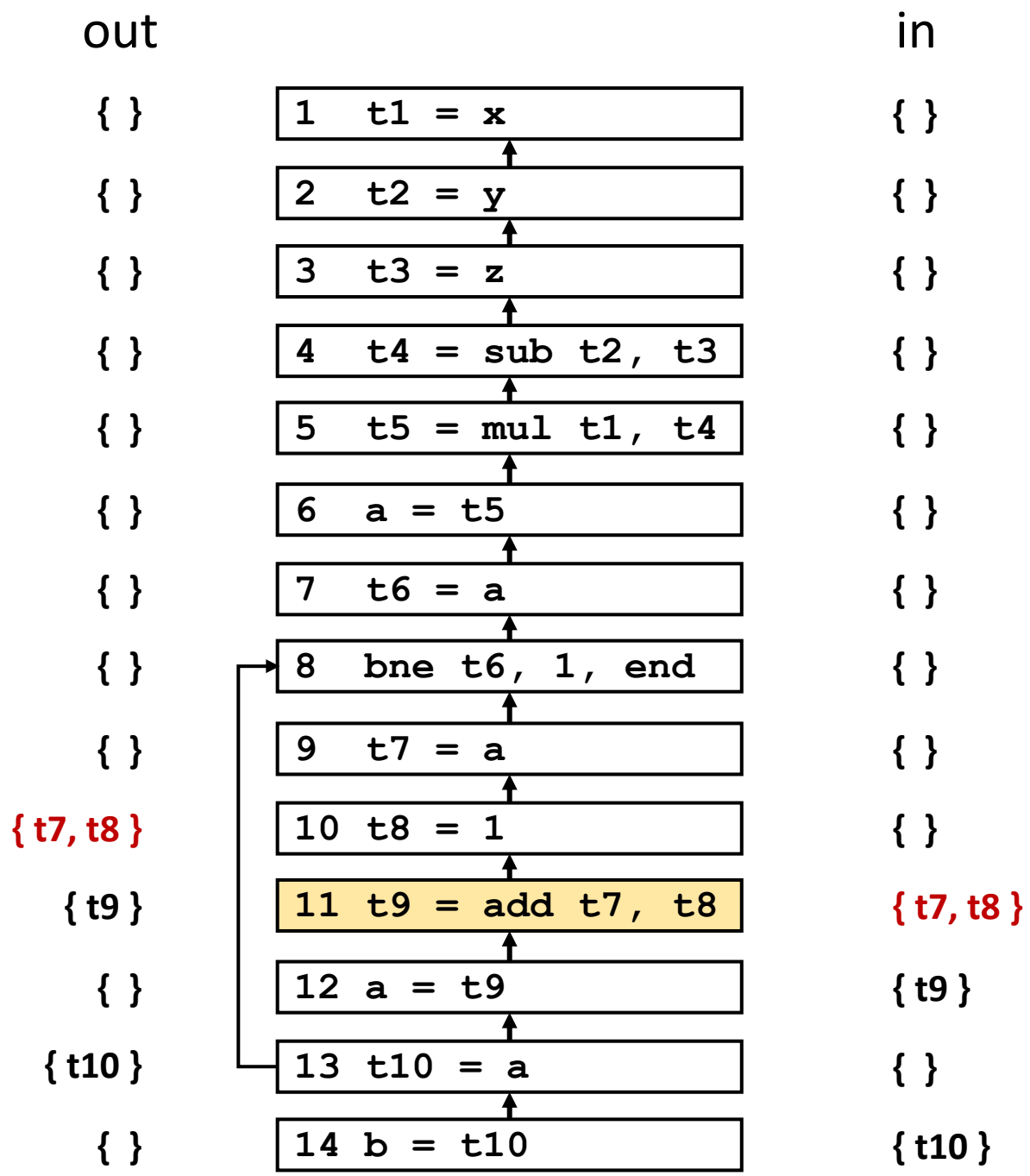
Work list = {12,8}



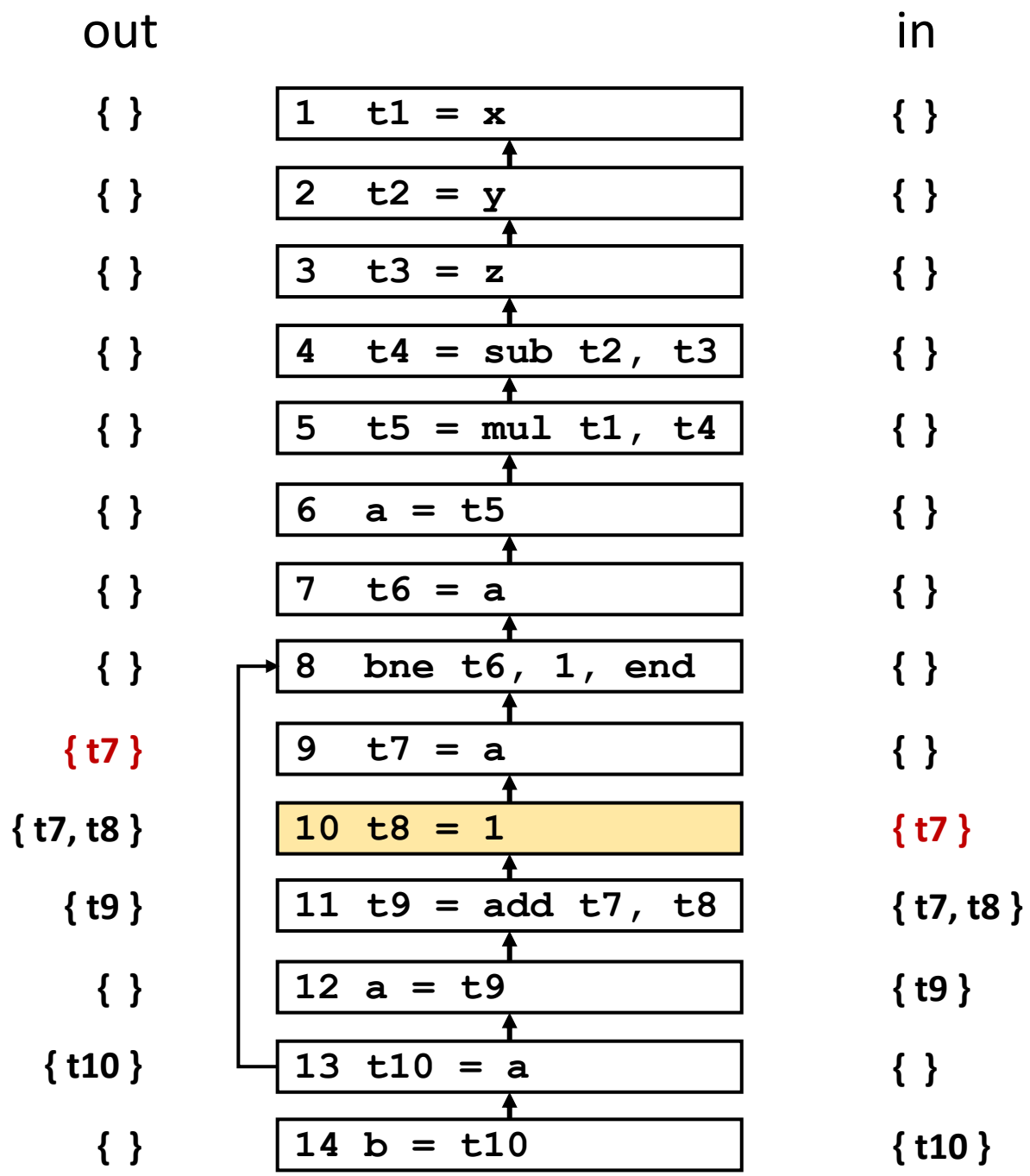
Work list = {11,8}



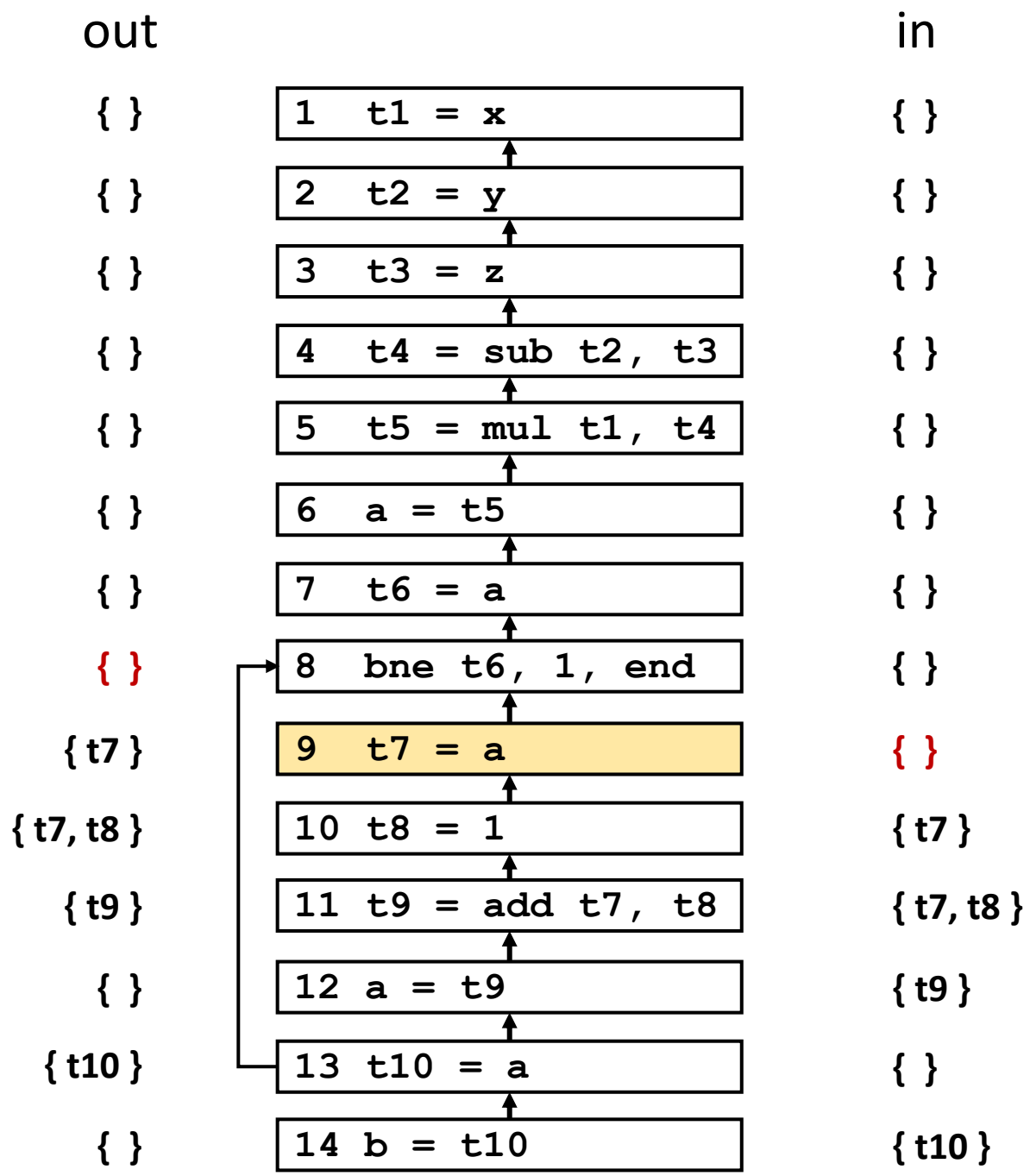
Work list = {10,8}



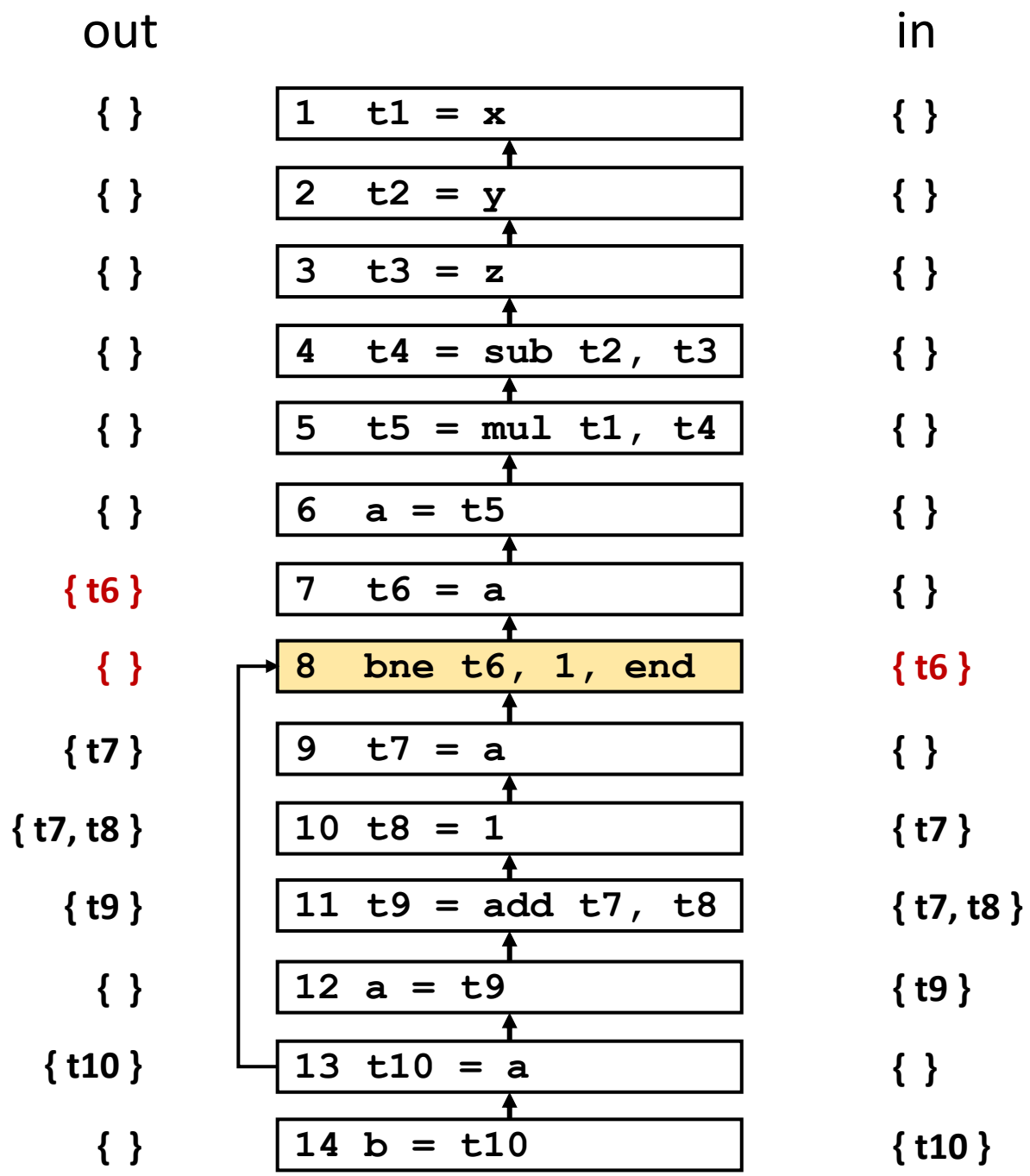
Work list = {9,8}


out
in
{ }
1 t1 = x
{ }
{ }
2 t2 = y
{ }
{ }
3 t3 = z
{ }
{ }
4 t4 = sub t2, t3
{ }
{ }
5 t5 = mul t1, t4
{ }
{ }
6 a = t5
{ }
{ }
7 t6 = a
{ }
{ }
8 bne t6, 1, end
{ }
{ t7 }
9 t7 = a
{ }
{ t7, t8 }
10 t8 = 1
{ t7 }
{ t9 }
11 t9 = add t7, t8
{ t7, t8 }
{ }
12 a = t9
{ t9 }
{ t10 }
13 t10 = a
{ }
{ }
14 b = t10
{ t10 }

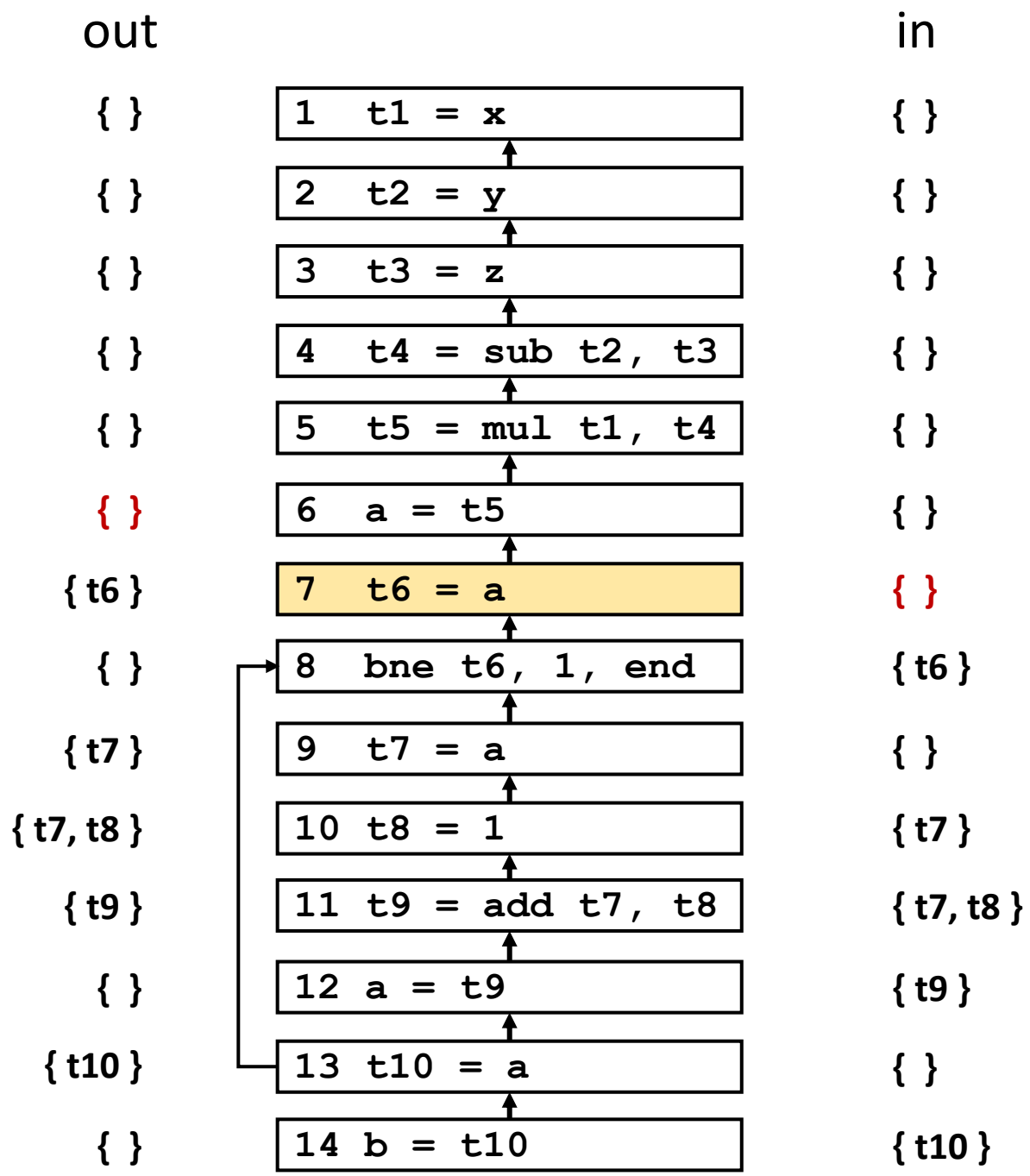
Work list = {8}


out
in
{ }
1 t1 = x
{ }
{ }
2 t2 = y
{ }
{ }
3 t3 = z
{ }
{ }
4 t4 = sub t2, t3
{ }
{ }
5 t5 = mul t1, t4
{ }
{ }
6 a = t5
{ }
{ }
7 t6 = a
{ }
{ }
8 bne t6, 1, end
{ }
{t7}
9 t7 = a
{ }
{t7, t8}
10 t8 = 1
{t7}
{t9}
11 t9 = add t7, t8
{t7, t8}
{ }
12 a = t9
{t9}
{t10}
13 t10 = a
{ }
{ }
14 b = t10
{t10}

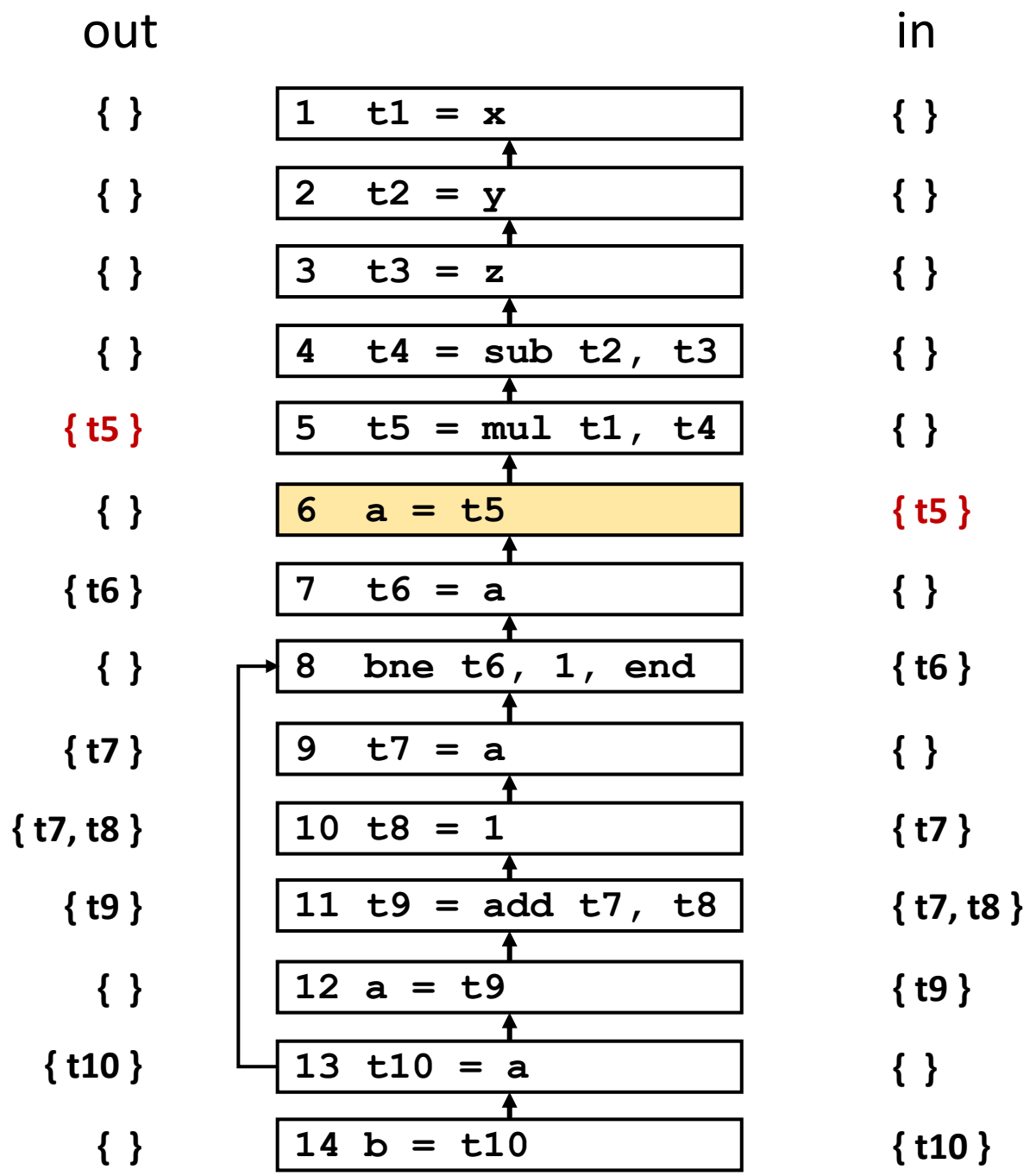
Work list = {7}


out
in
{ }
1 t1 = x
{ }
{ }
2 t2 = y
{ }
{ }
3 t3 = z
{ }
{ }
4 t4 = sub t2, t3
{ }
{ }
5 t5 = mul t1, t4
{ }
{ }
6 a = t5
{ }
{ t6 }
7 t6 = a
{ }
{ }
8 bne t6, 1, end
{ t6 }
{ t7 }
9 t7 = a
{ }
{ t7, t8 }
10 t8 = 1
{ t7 }
{ t9 }
11 t9 = add t7, t8
{ t7, t8 }
{ }
12 a = t9
{ t9 }
{ t10 }
13 t10 = a
{ }
{ }
14 b = t10
{ t10 }

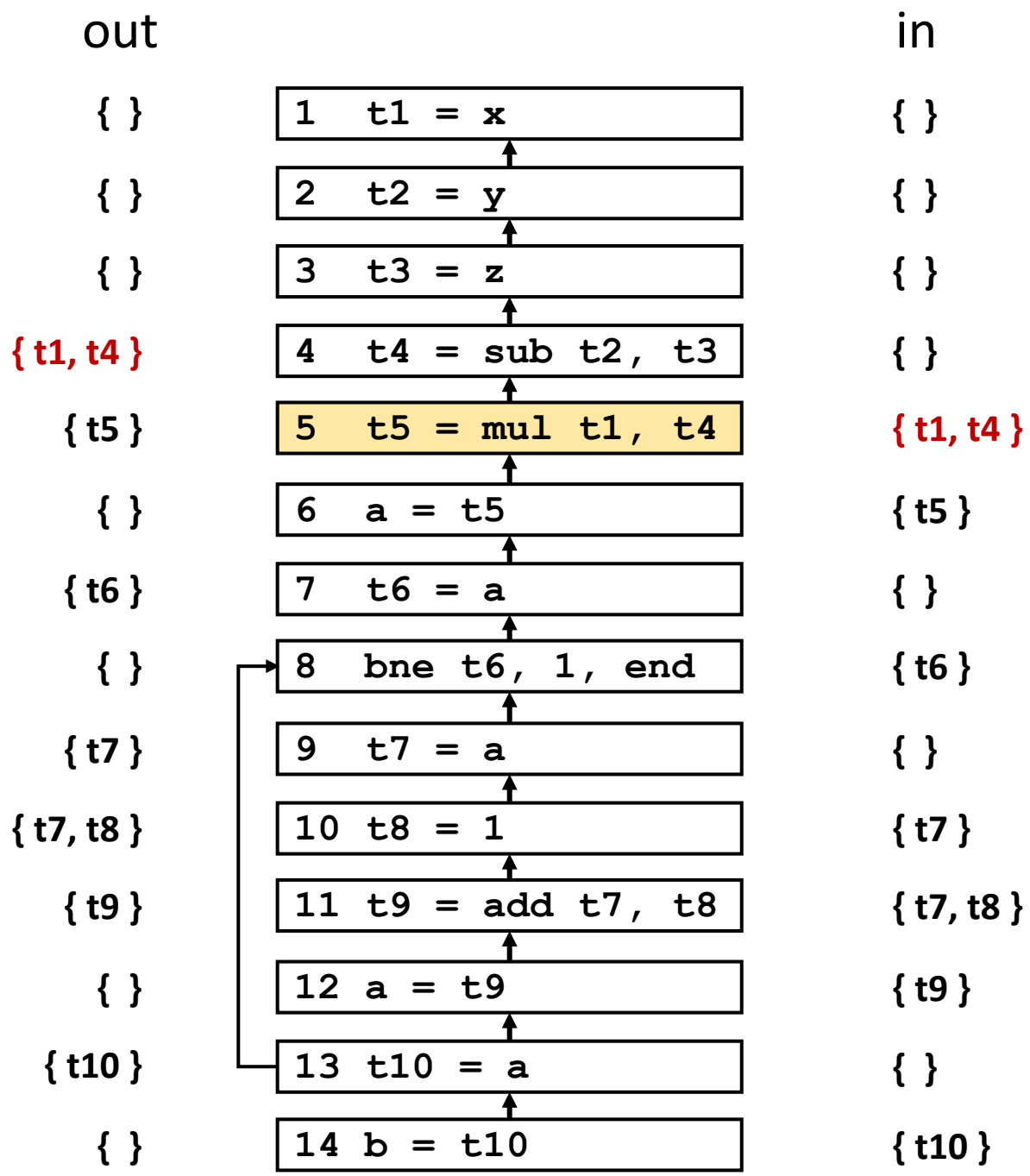
Work list = {6}


out
in
{ }
1 t1 = x
{ }
{ }
2 t2 = y
{ }
{ }
3 t3 = z
{ }
{ }
4 t4 = sub t2, t3
{ }
{ }
5 t5 = mul t1, t4
{ }
{ }
6 a = t5
{ }
{ t6 }
7 t6 = a
{ }
{ }
8 bne t6, 1, end
{ t6 }
{ t7 }
9 t7 = a
{ }
{ t7, t8 }
10 t8 = 1
{ t7 }
{ t9 }
11 t9 = add t7, t8
{ t7, t8 }
{ }
12 a = t9
{ t9 }
{ t10 }
13 t10 = a
{ }
{ }
14 b = t10
{ t10 }

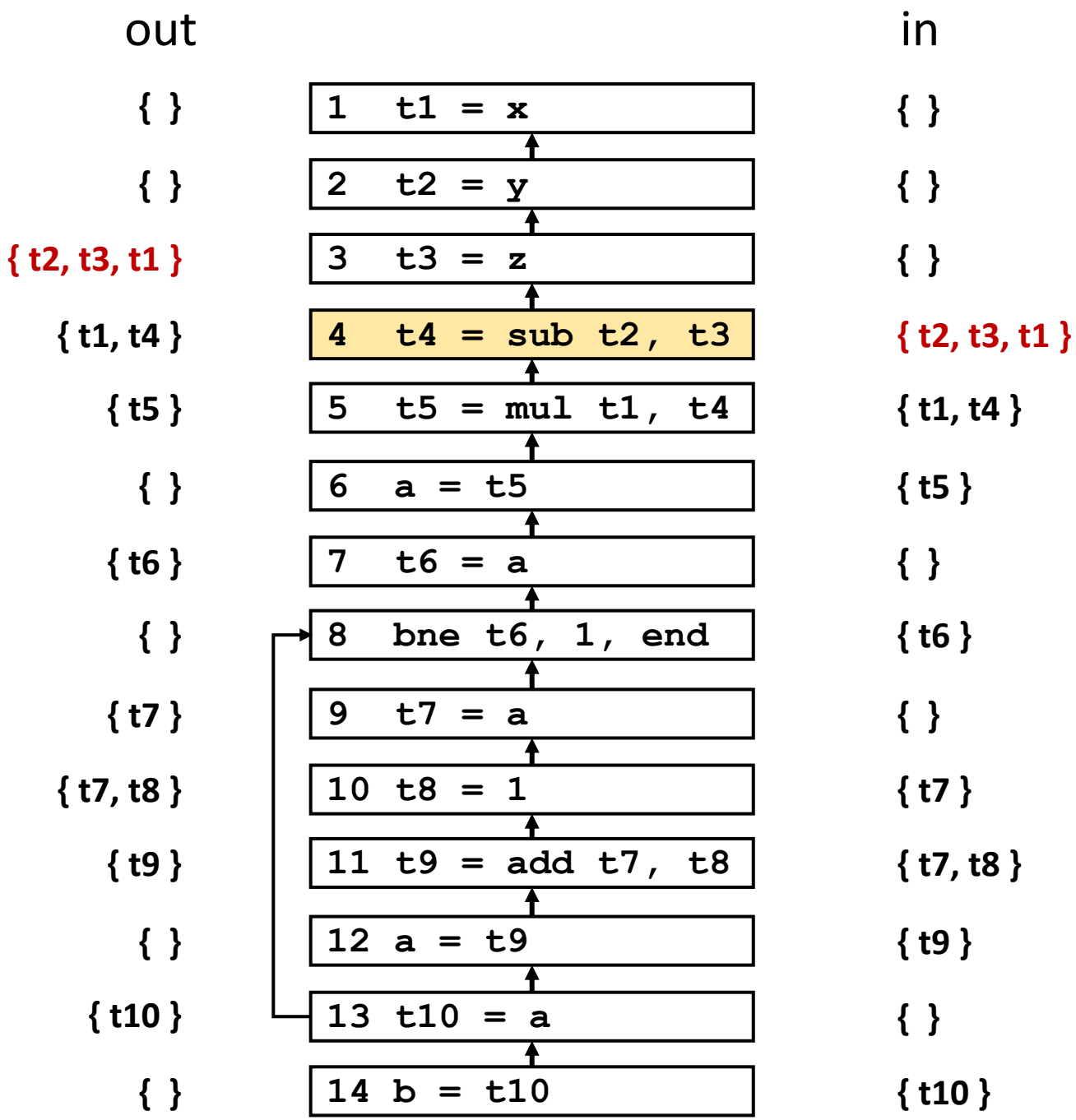
Work list = {5}


out
in
{ }
1 t1 = x
{ }
{ }
2 t2 = y
{ }
{ }
3 t3 = z
{ }
{ }
4 t4 = sub t2, t3
{ }
{ t5 }
5 t5 = mul t1, t4
{ }
{ }
6 a = t5
{ t5 }
{ t6 }
7 t6 = a
{ }
{ }
8 bne t6, 1, end
{ t6 }
{ t7 }
9 t7 = a
{ }
{ t7, t8 }
10 t8 = 1
{ t7 }
{ t9 }
11 t9 = add t7, t8
{ t7, t8 }
{ }
12 a = t9
{ t9 }
{ t10 }
13 t10 = a
{ }
{ }
14 b = t10
{ t10 }

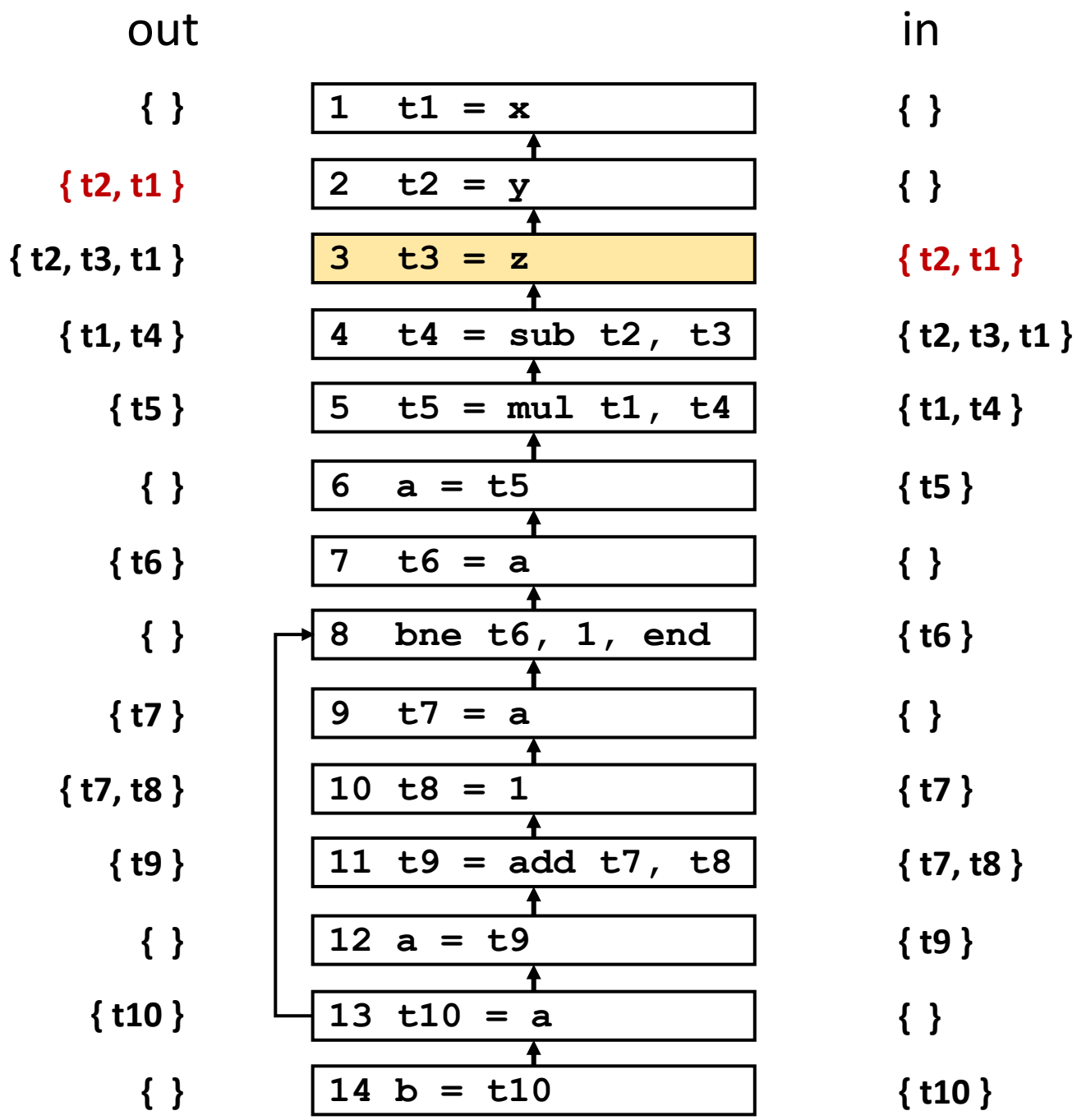
Work list = {4}


out
in
{ }
1 t1 = x
{ }
{ }
2 t2 = y
{ }
{ }
3 t3 = z
{ }
{ t1, t4 }
4 t4 = sub t2, t3
{ }
{ t5 }
5 t5 = mul t1, t4
{ t1, t4 }
{ }
6 a = t5
{ t5 }
{ t6 }
7 t6 = a
{ }
{ }
8 bne t6, 1, end
{ t6 }
{ t7 }
9 t7 = a
{ }
{ t7, t8 }
10 t8 = 1
{ t7 }
{ t9 }
11 t9 = add t7, t8
{ t7, t8 }
{ }
12 a = t9
{ t9 }
{ t10 }
13 t10 = a
{ }
{ }
14 b = t10
{ t10 }

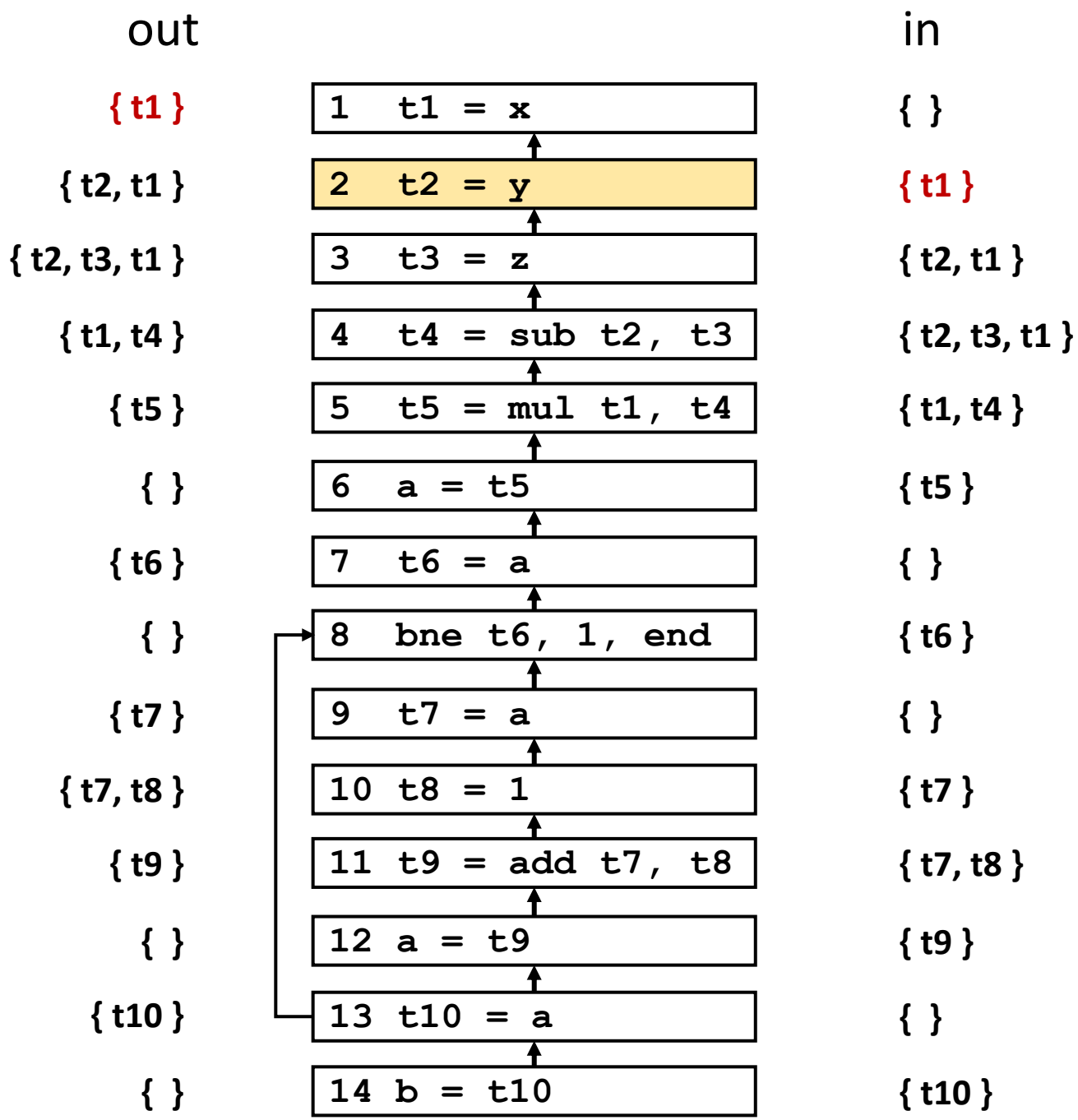
Work list = {3}


out
in
{ }
1 t1 = x
{ }
{ }
2 t2 = y
{ }
{ t2, t3, t1 }
3 t3 = z
{ }
{ t1, t4 }
4 t4 = sub t2, t3
{ t2, t3, t1 }
{ t5 }
5 t5 = mul t1, t4
{ t1, t4 }
{ }
6 a = t5
{ t5 }
{ t6 }
7 t6 = a
{ }
{ }
8 bne t6, 1, end
{ t6 }
{ t7 }
9 t7 = a
{ }
{ t7, t8 }
10 t8 = 1
{ t7 }
{ t9 }
11 t9 = add t7, t8
{ t7, t8 }
{ }
12 a = t9
{ t9 }
{ t10 }
13 t10 = a
{ }
{ }
14 b = t10
{ t10 }

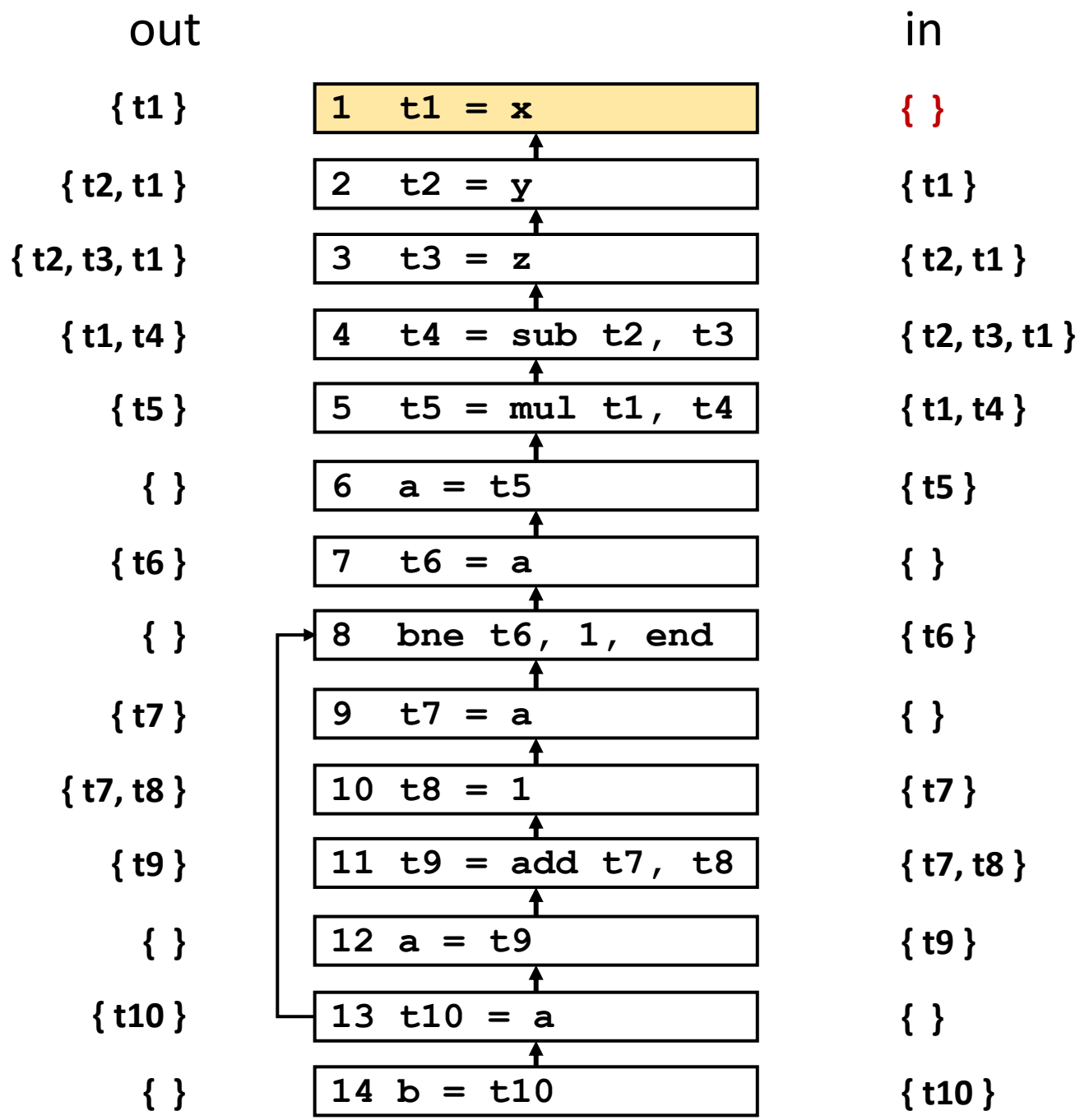
Work list = {2}

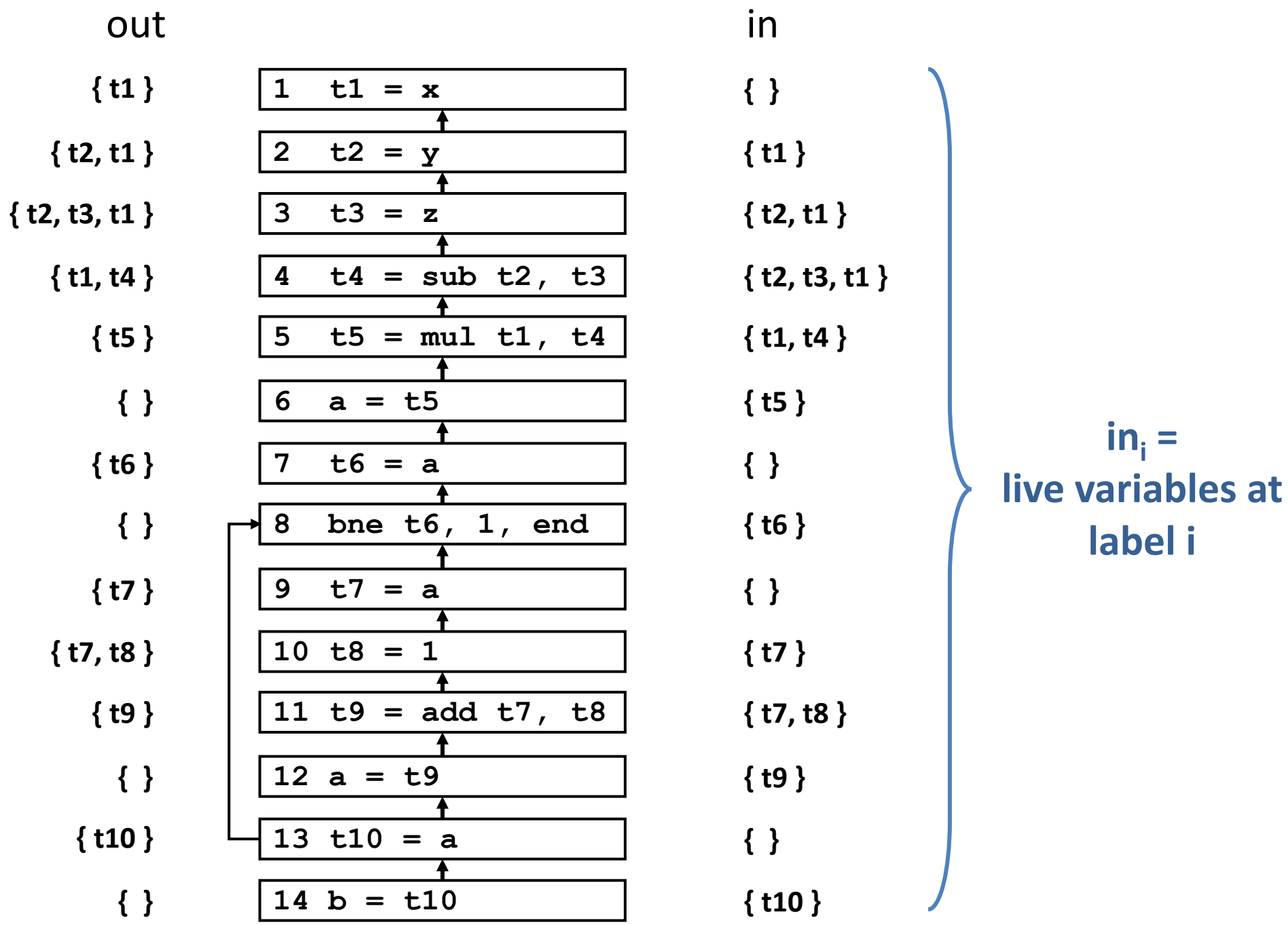

out
in
{ }
1 t1 = x
{ }
{ t2, t1 }
2 t2 = y
{ }
{ t2, t3, t1 }
3 t3 = z
{ t2, t1 }
{ t1, t4 }
4 t4 = sub t2, t3
{ t2, t3, t1 }
{ t5 }
5 t5 = mul t1, t4
{ t1, t4 }
{ }
6 a = t5
{ t5 }
{ t6 }
7 t6 = a
{ }
{ }
8 bne t6, 1, end
{ t6 }
{ t7 }
9 t7 = a
{ }
{ t7, t8 }
10 t8 = 1
{ t7 }
{ t9 }
11 t9 = add t7, t8
{ t7, t8 }
{ }
12 a = t9
{ t9 }
{ t10 }
13 t10 = a
{ }
{ }
14 b = t10
{ t10 }

Work list = {1}


out
in
{ t1 }
1 t1 = x
{ }
{ t2, t1 }
2 t2 = y
{ t1 }
{ t2, t3, t1 }
3 t3 = z
{ t2, t1 }
{ t1, t4 }
4 t4 = sub t2, t3
{ t2, t3, t1 }
{ t5 }
5 t5 = mul t1, t4
{ t1, t4 }
{ }
6 a = t5
{ t5 }
{ t6 }
7 t6 = a
{ }
{ }
8 bne t6, 1, end
{ t6 }
{ t7 }
9 t7 = a
{ }
{ t7, t8 }
10 t8 = 1
{ t7 }
{ t9 }
11 t9 = add t7, t8
{ t7, t8 }
{ }
12 a = t9
{ t9 }
{ t10 }
13 t10 = a
{ }
{ }
14 b = t10
{ t10 }

Work list = {}


out
in
{ t1 }
1 t1 = x
{ }
{ t2, t1 }
2 t2 = y
{ t1 }
{ t2, t3, t1 }
3 t3 = z
{ t2, t1 }
{ t1, t4 }
4 t4 = sub t2, t3
{ t2, t3, t1 }
{ t5 }
5 t5 = mul t1, t4
{ t1, t4 }
{ }
6 a = t5
{ t5 }
{ t6 }
7 t6 = a
{ }
{ }
8 bne t6, 1, end
{ t6 }
{ t7 }
9 t7 = a
{ }
{ t7, t8 }
10 t8 = 1
{ t7 }
{ t9 }
11 t9 = add t7, t8
{ t7, t8 }
{ }
12 a = t9
{ t9 }
{ t10 }
13 t10 = a
{ }
{ }
14 b = t10
{ t10 }


out
in
{ t1 }
1 t1 = x
{ }
{ t2, t1 }
2 t2 = y
{ t1 }
{ t2, t3, t1 }
3 t3 = z
{ t2, t1 }
{ t1, t4 }
4 t4 = sub t2, t3
{ t2, t3, t1 }
{ t5 }
5 t5 = mul t1, t4
{ t1, t4 }
{ }
6 a = t5
{ t5 }
{ t6 }
7 t6 = a
{ }
{ }
8 bne t6, 1, end
{ t6 }
{ t7 }
9 t7 = a
{ }
{ t7, t8 }
10 t8 = 1
{ t7 }
{ t9 }
11 t9 = add t7, t8
{ t7, t8 }
{ }
12 a = t9
{ t9 }
{ t10 }
13 t10 = a
{ }
{ }
14 b = t10
{ t10 }
in_i = live variables at label i

Register Allocation

For each function:

1. Construct the CFG (from the IR)
2. Run liveness analysis
3. **Construct the interference graph**
4. Compute a *k-coloring* of the graph
5. Use the coloring to build the required mapping

Interference Graph

- Use the liveness analysis output to construct the **interference graph**
- Create a node for each IR variable (*t1*, *t2*, …)
- If *t1* and *t2* appear in the same liveness sets (in_i for some i):
  - Create an edge between *t1* and *t2*

Interference Graph

{ t1 }

{ t2, t1 }

{ t2, t3, t1 }

{ t1, t4 }

{ t5 }

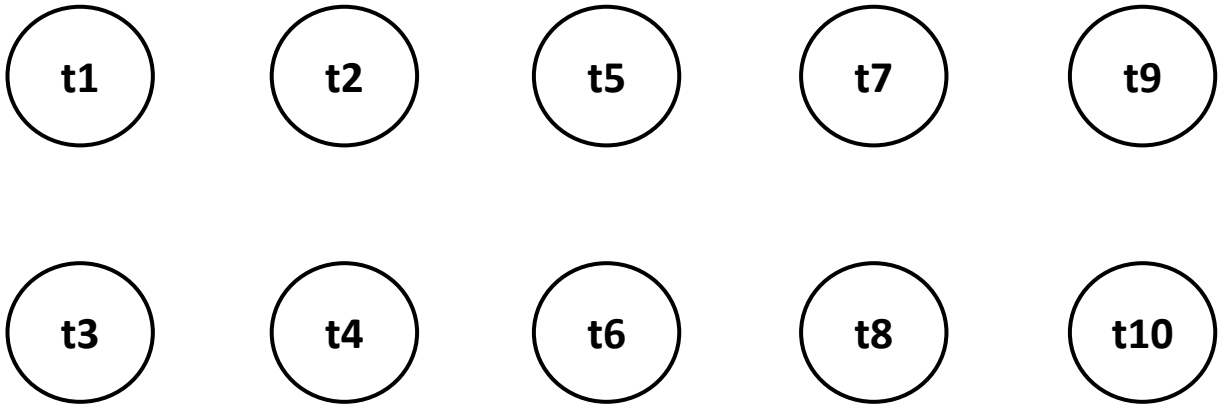
{ t6 }

{ t7 }

{ t7, t8 }

{ t9 }

{ t10 }



Interference Graph

{ t1 }

{ t2, t1 }

{ t2, t3, t1 }

{ t1, t4 }

{ t5 }

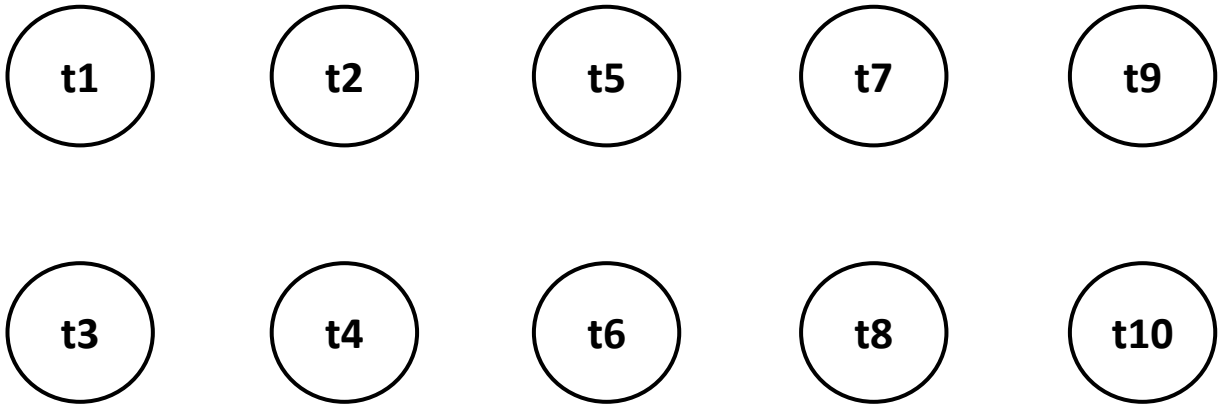
{ t6 }

{ t7 }

{ t7, t8 }

{ t9 }

{ t10 }



Interference Graph

{ t1 }

{ t2, t1 }

{ t2, t3, t1 }

{ t1, t4 }

{ t5 }

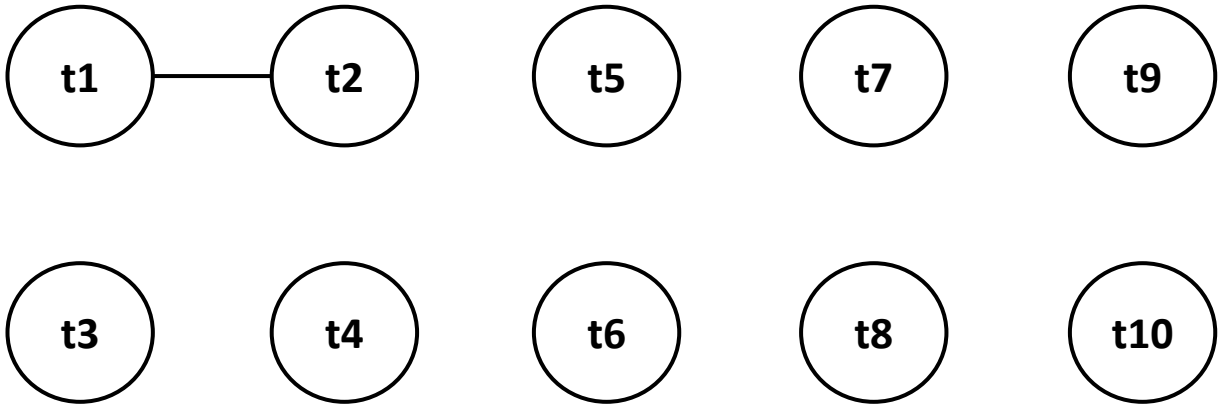
{ t6 }

{ t7 }

{ t7, t8 }

{ t9 }

{ t10 }



Interference Graph

{ t1 }

{ t2, t1 }

{ t2, t3, t1 }

{ t1, t4 }

{ t5 }

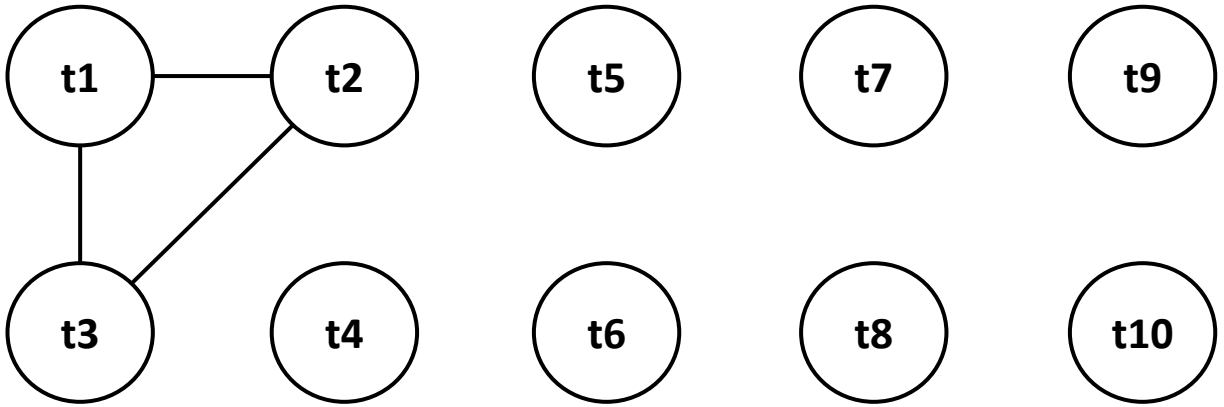
{ t6 }

{ t7 }

{ t7, t8 }

{ t9 }

{ t10 }



Interference Graph

{ t1 }

{ t2, t1 }

{ t2, t3, t1 }

{ t1, t4 }

{ t5 }

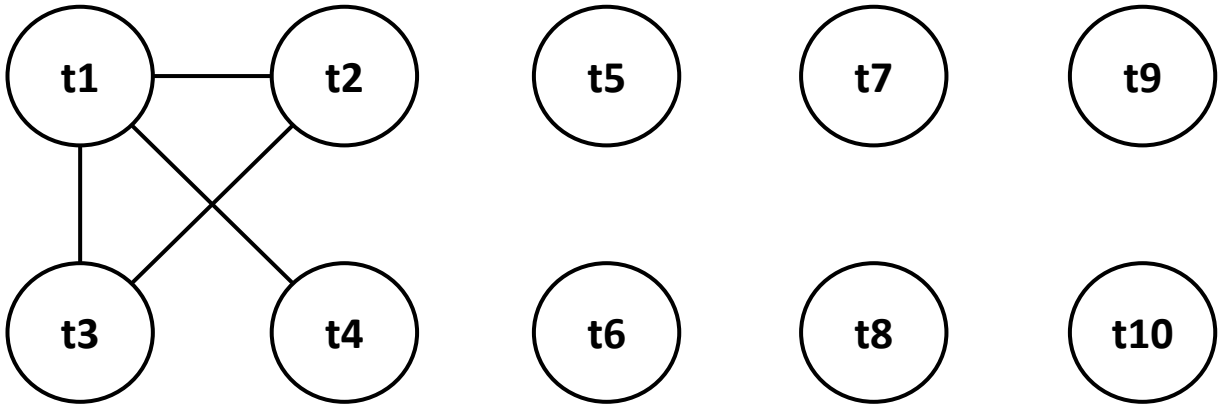
{ t6 }

{ t7 }

{ t7, t8 }

{ t9 }

{ t10 }



Interference Graph

{ t1 }

{ t2, t1 }

{ t2, t3, t1 }

{ t1, t4 }

{ t5 }

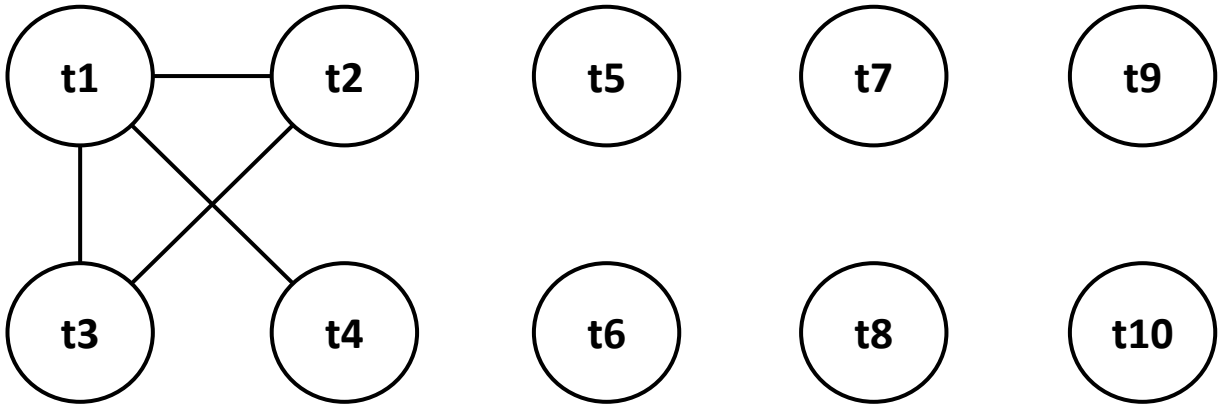
{ t6 }

{ t7 }

{ t7, t8 }

{ t9 }

{ t10 }



Interference Graph

{ t1 }

{ t2, t1 }

{ t2, t3, t1 }

{ t1, t4 }

{ t5 }

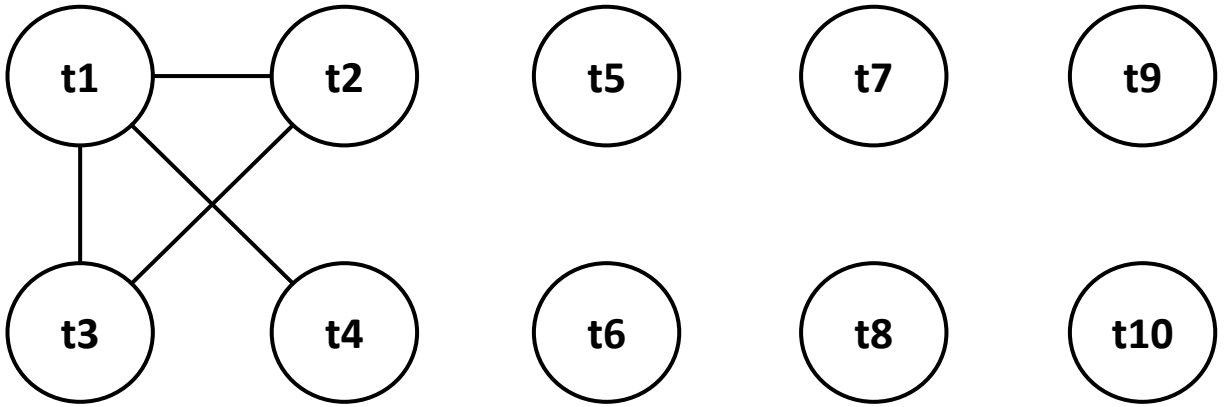
{ t6 }

{ t7 }

{ t7, t8 }

{ t9 }

{ t10 }



Interference Graph

{ t1 }

{ t2, t1 }

{ t2, t3, t1 }

{ t1, t4 }

{ t5 }

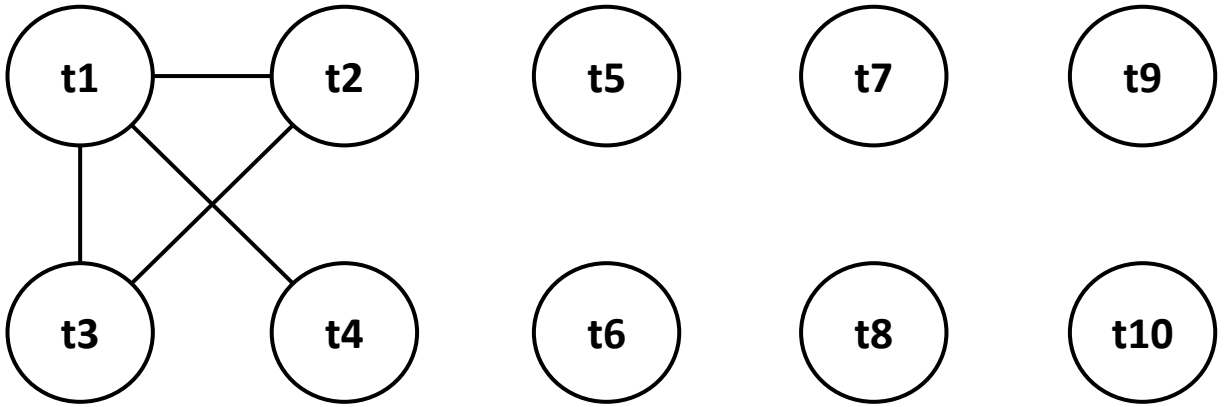
{ t6 }

{ t7 }

{ t7, t8 }

{ t9 }

{ t10 }



Interference Graph

{ t1 }

{ t2, t1 }

{ t2, t3, t1 }

{ t1, t4 }

{ t5 }

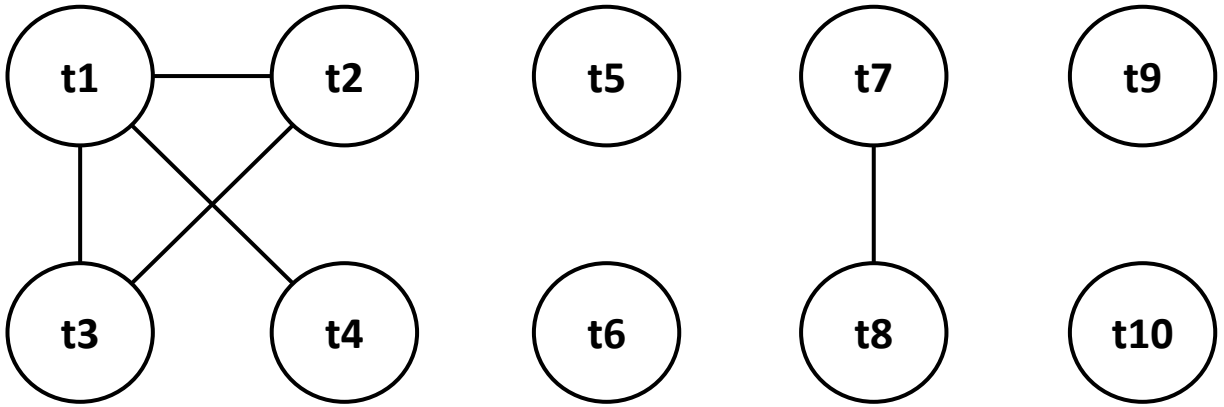
{ t6 }

{ t7 }

{ t7, t8 }

{ t9 }

{ t10 }



Interference Graph

{ t1 }

{ t2, t1 }

{ t2, t3, t1 }

{ t1, t4 }

{ t5 }

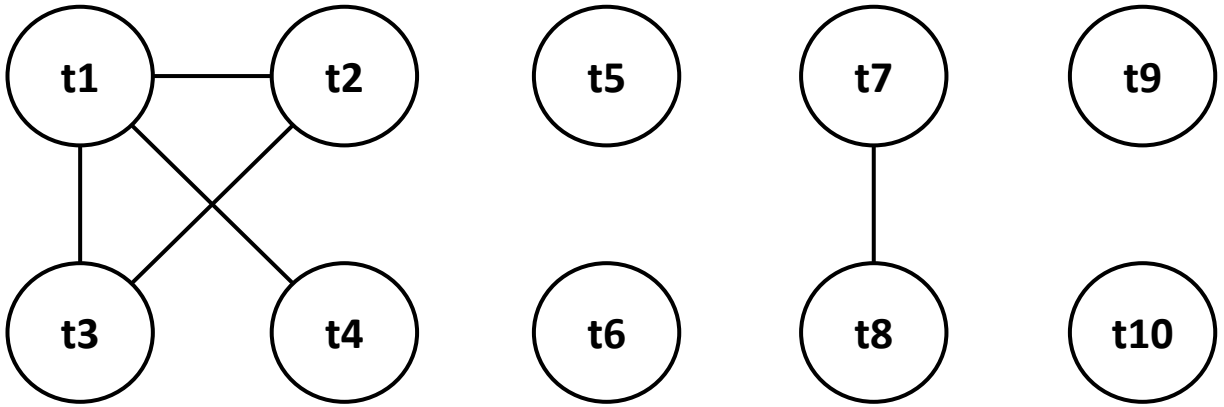
{ t6 }

{ t7 }

{ t7, t8 }

{ t9 }

{ t10 }



Interference Graph

{ t1 }

{ t2, t1 }

{ t2, t3, t1 }

{ t1, t4 }

{ t5 }

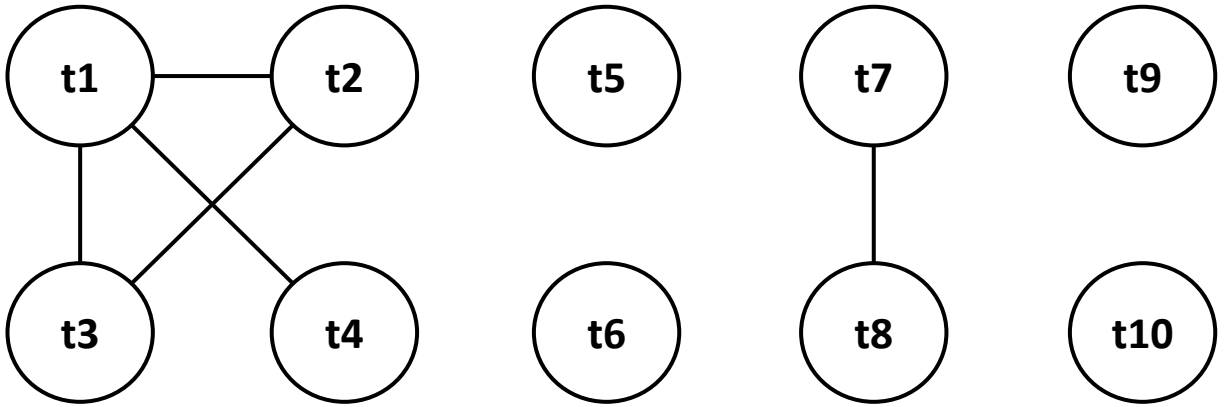
{ t6 }

{ t7 }

{ t7, t8 }

{ t9 }

{ t10 }



Register Allocation

1. For each function:
    1. Construct the CFG (from the IR)
    2. Run liveness analysis
    3. Construct the interference graph
    4. **Compute a *k-coloring* of the graph**
    5. Use the coloring to build the required mapping

Graph Coloring

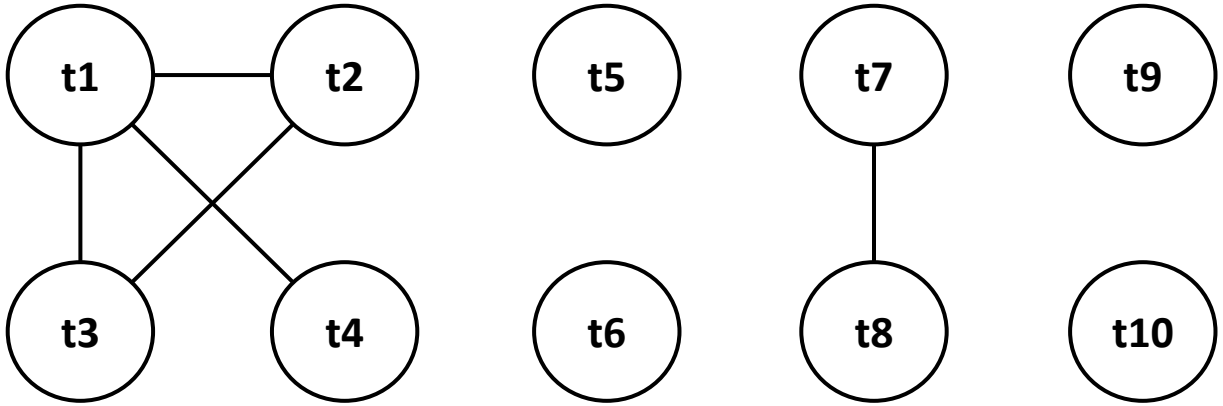
- Graph coloring is hard (NP-complete problem)
- We need a heuristic…

Graph Coloring

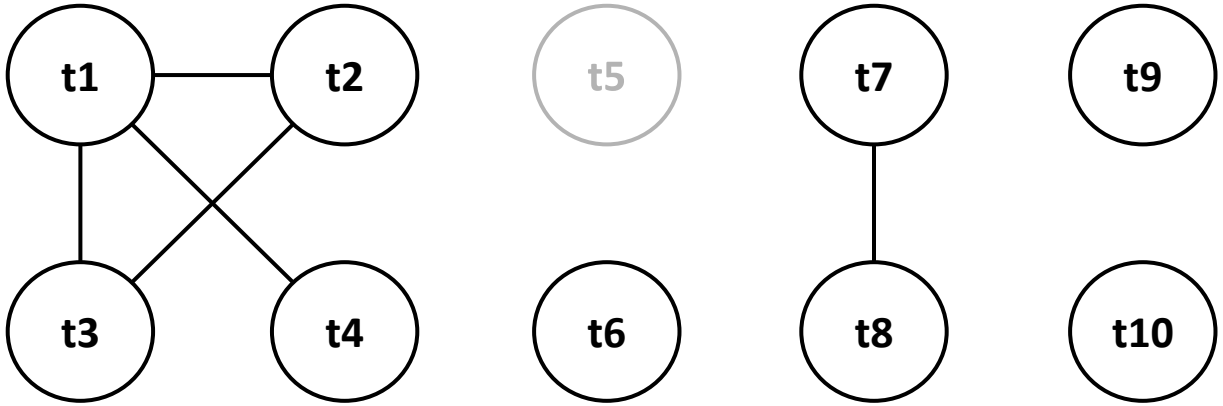
Chaitin's Algorithm for *k-coloring* (simplified):

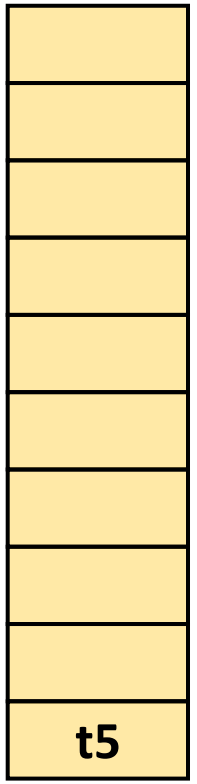
- While there is a node with less than *k* neighbors:
  - Remove it and its edges from the graph
  - Push it on the stack
- If the entire graph was removed, then it is *k*-colorable
- While the stack is not empty:
  - Pop a node from the stack and assign it an available color

Graph Coloring (k = 3)

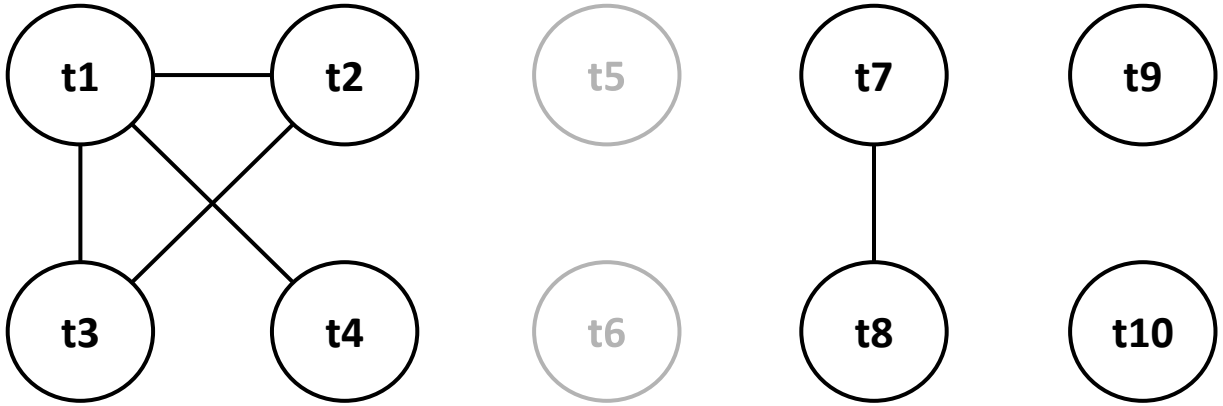


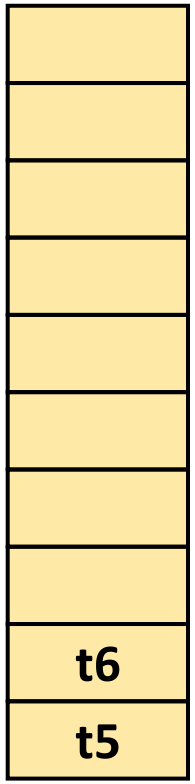
Graph Coloring (k = 3)



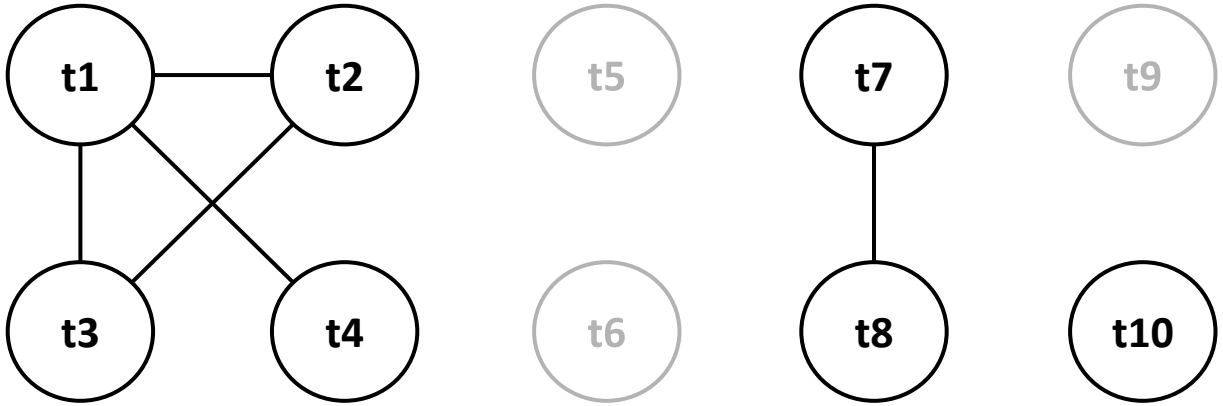


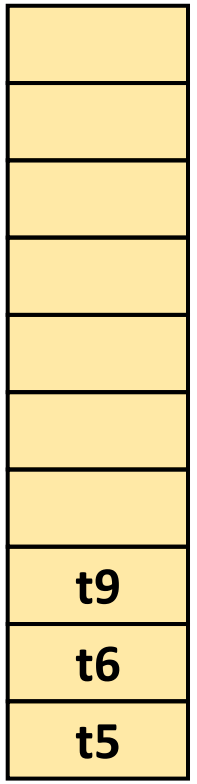
Graph Coloring (k = 3)



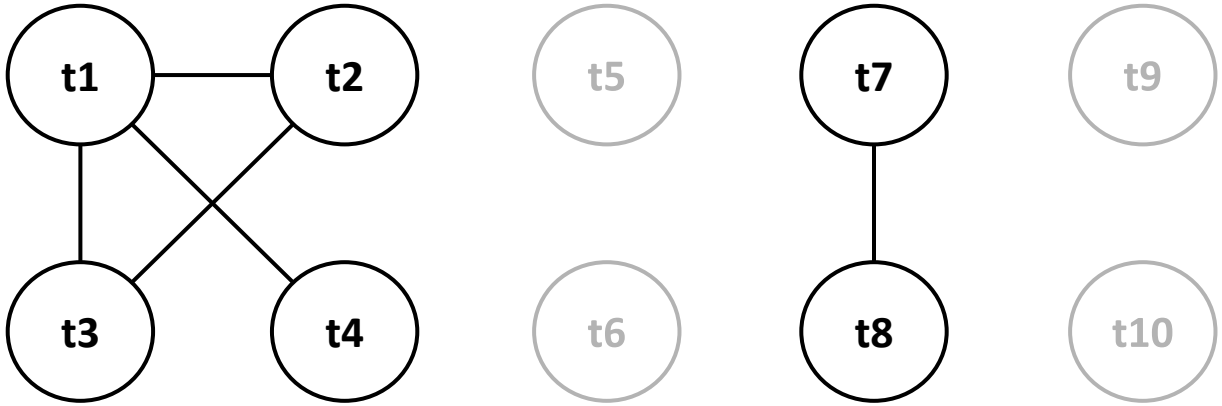


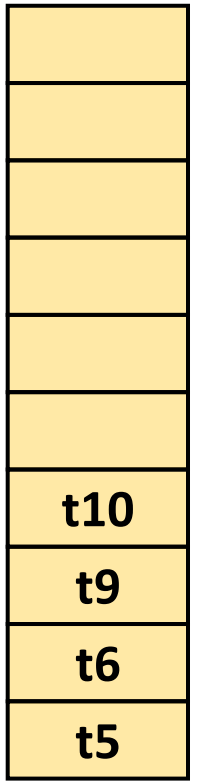
Graph Coloring (k = 3)



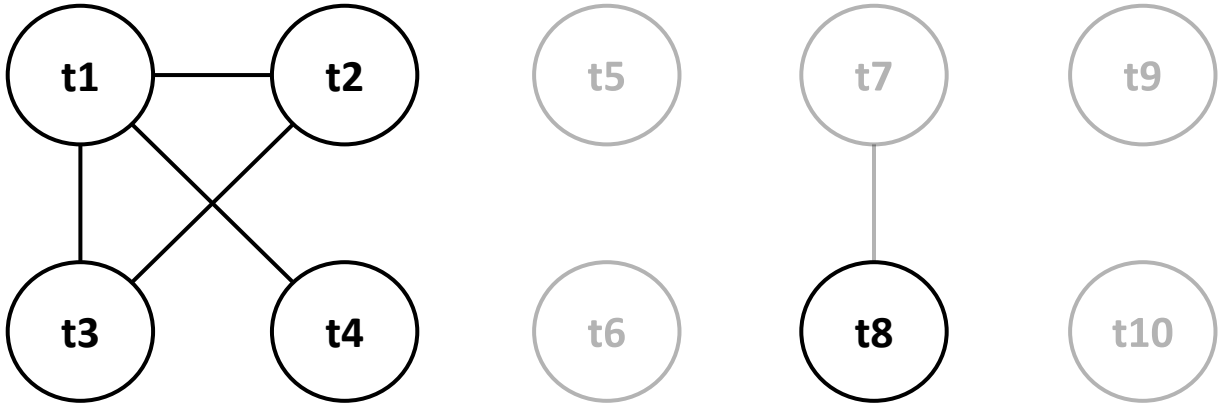


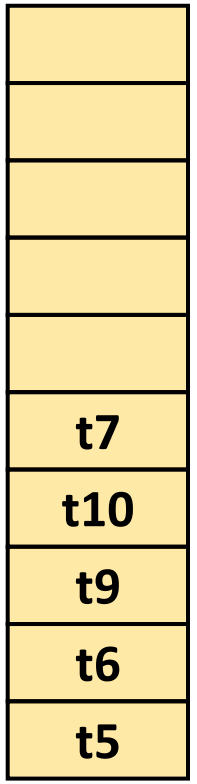
Graph Coloring (k = 3)



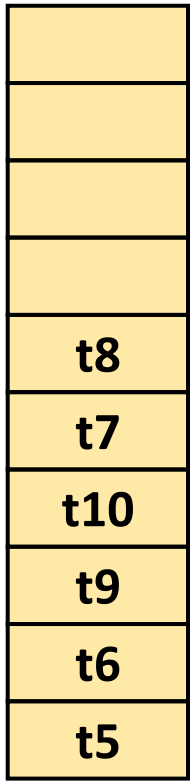


Graph Coloring (k = 3)

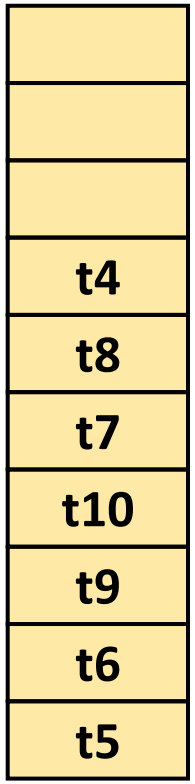




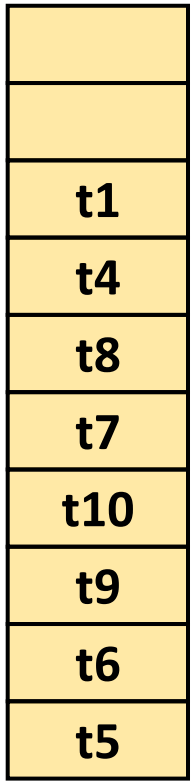
Graph Coloring (k = 3)



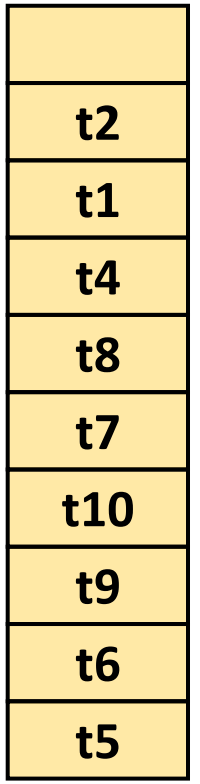
Graph Coloring (k = 3)



Graph Coloring (k = 3)



Graph Coloring (k = 3)



Graph Coloring (k = 3)

t3
t2
t1
t4
t8
t7
t10
t9
t6
t5

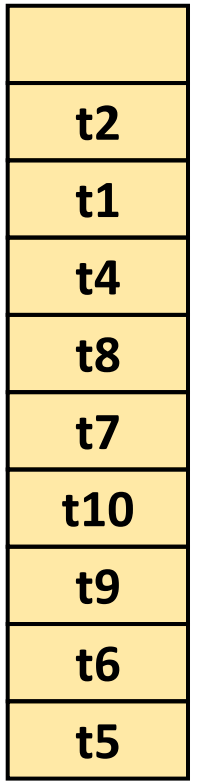
Graph Coloring (k = 3)

Pop nodes and assign colors…

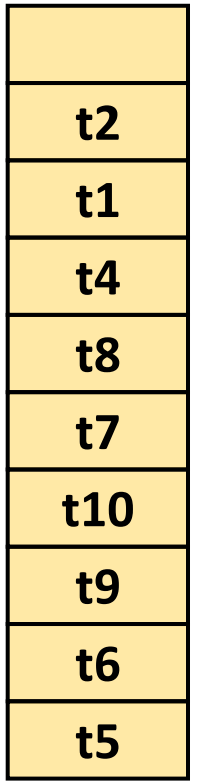
Graph Coloring (k = 3)

t3
t2
t1
t4
t8
t7
t10
t9
t6
t5

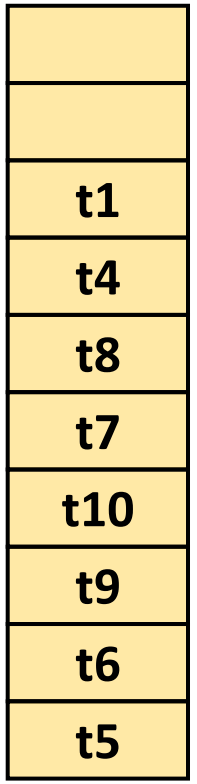
Graph Coloring (k = 3)



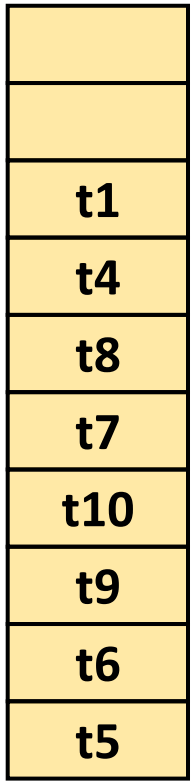
Graph Coloring (k = 3)



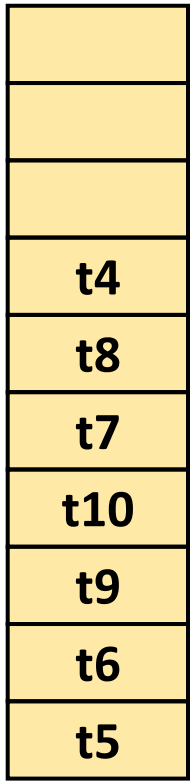
Graph Coloring (k = 3)



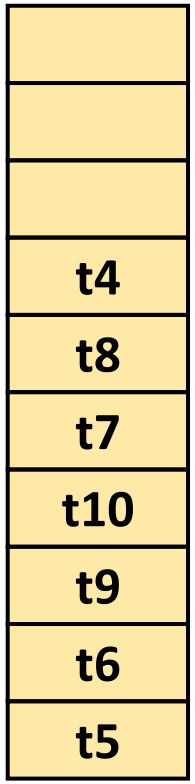
Graph Coloring (k = 3)



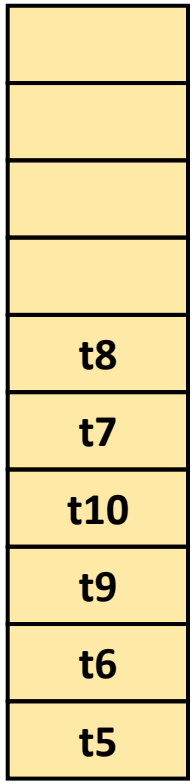
Graph Coloring (k = 3)



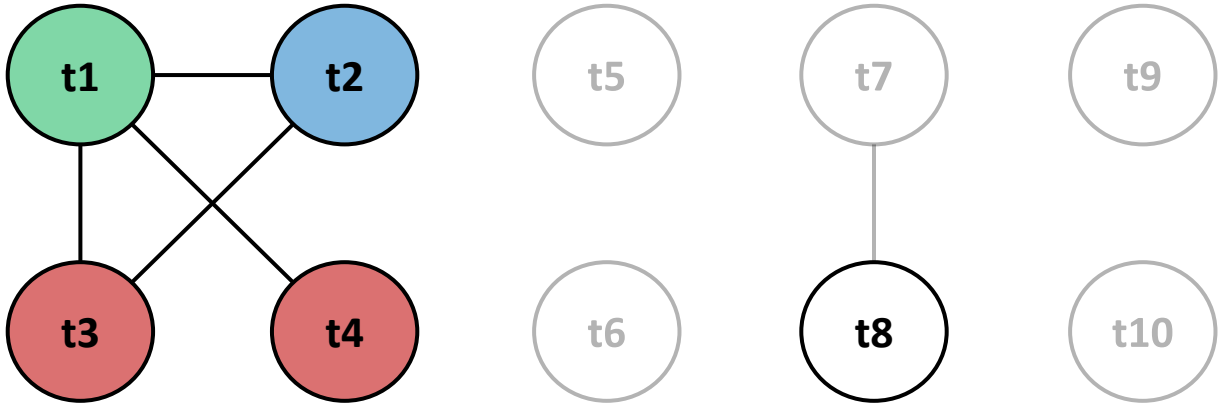
Graph Coloring (k = 3)

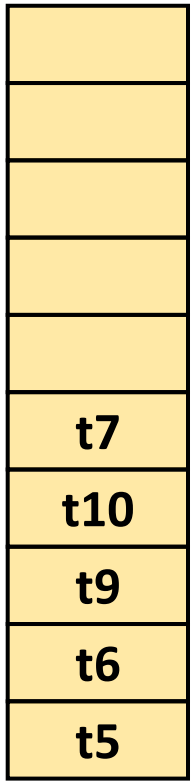


Graph Coloring (k = 3)

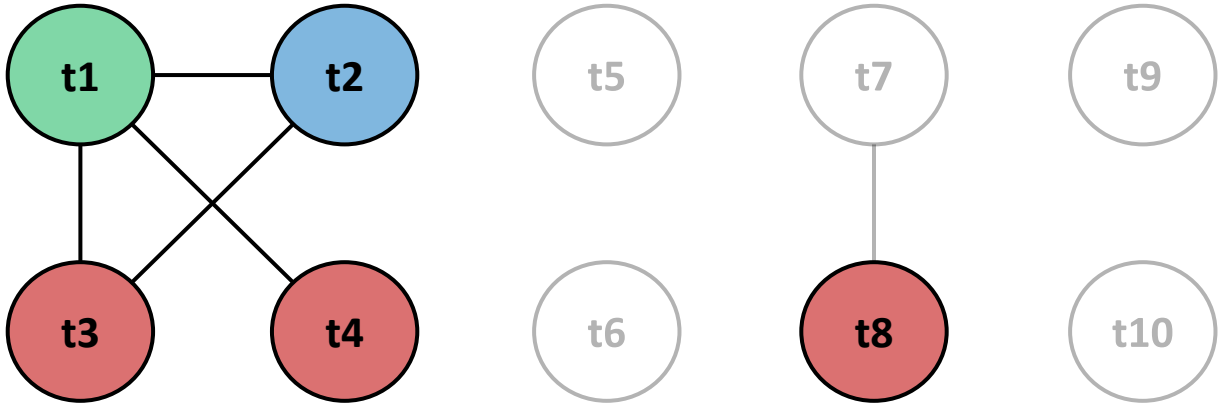


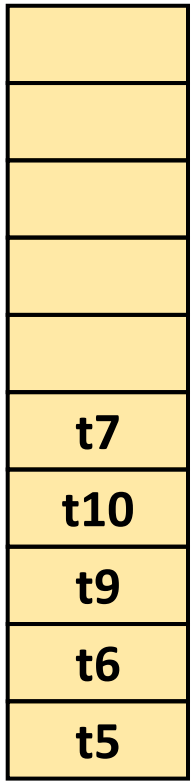
Graph Coloring (k = 3)



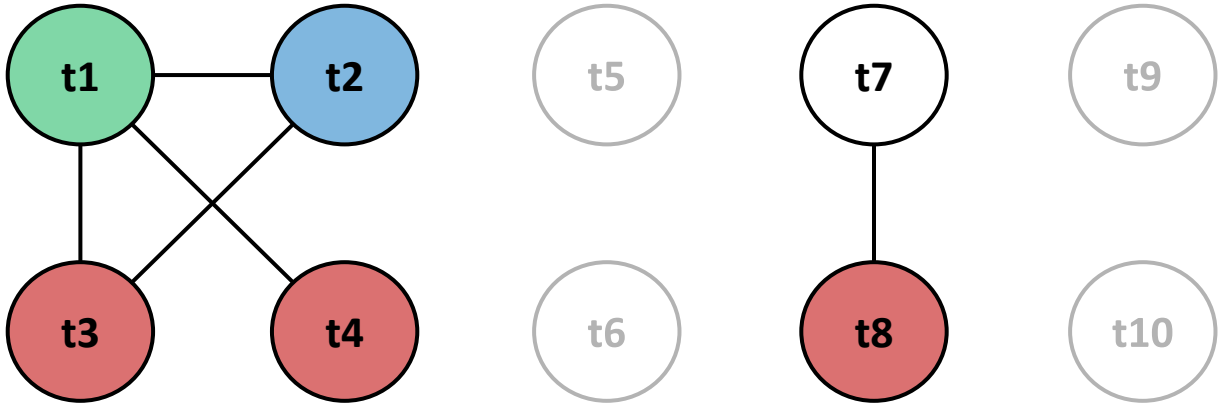


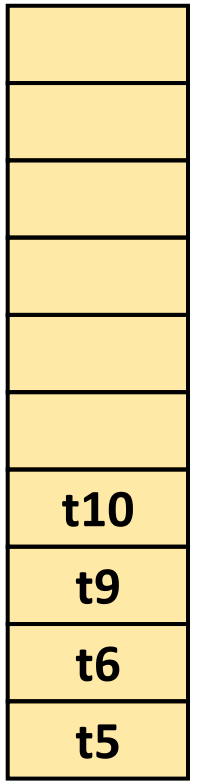
Graph Coloring (k = 3)



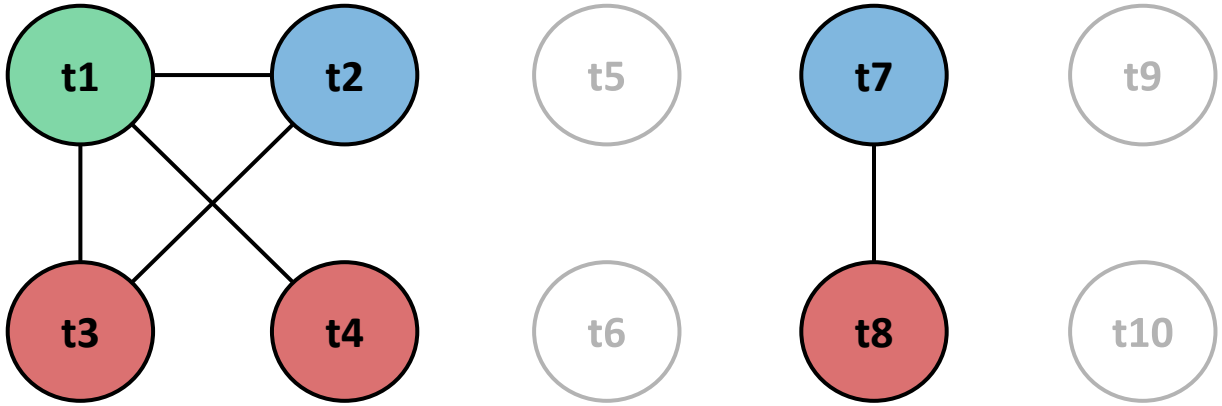


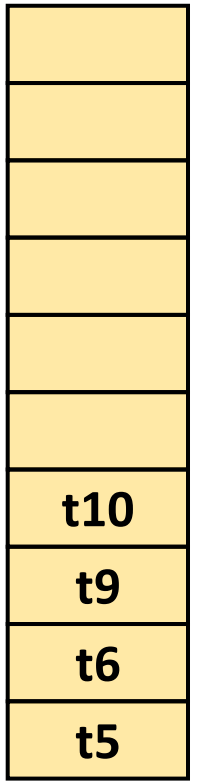
Graph Coloring (k = 3)



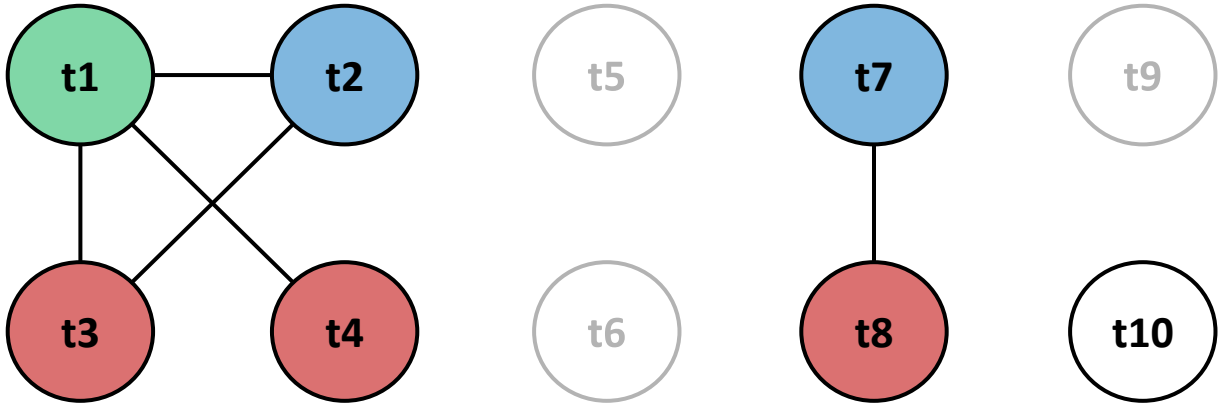


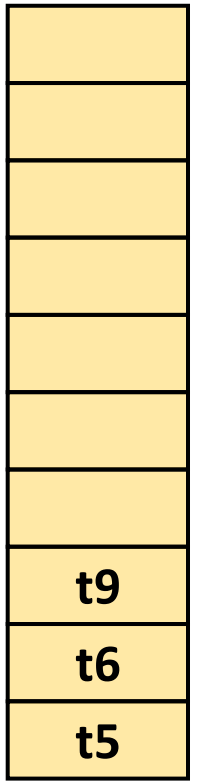
Graph Coloring (k = 3)



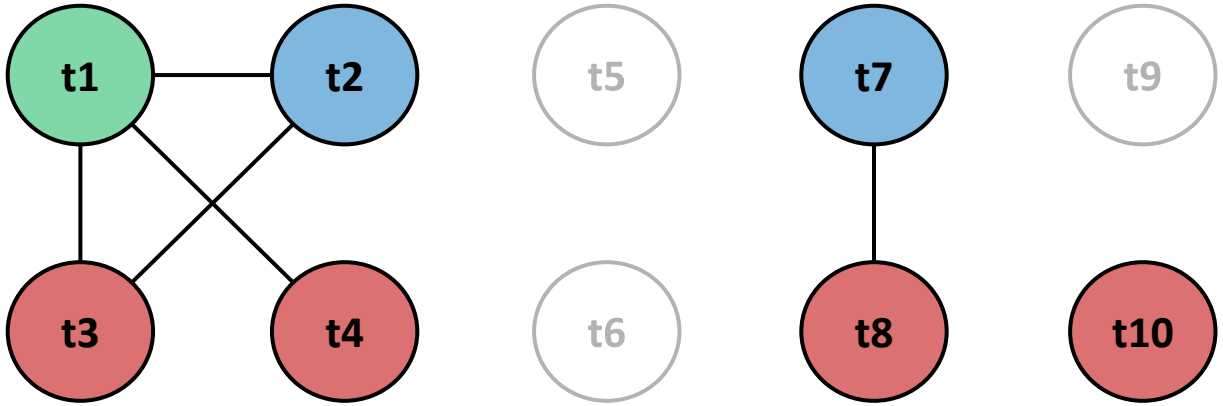


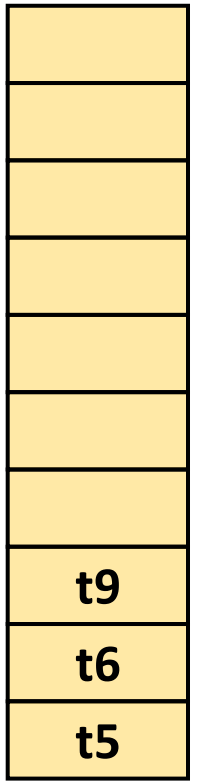
Graph Coloring (k = 3)



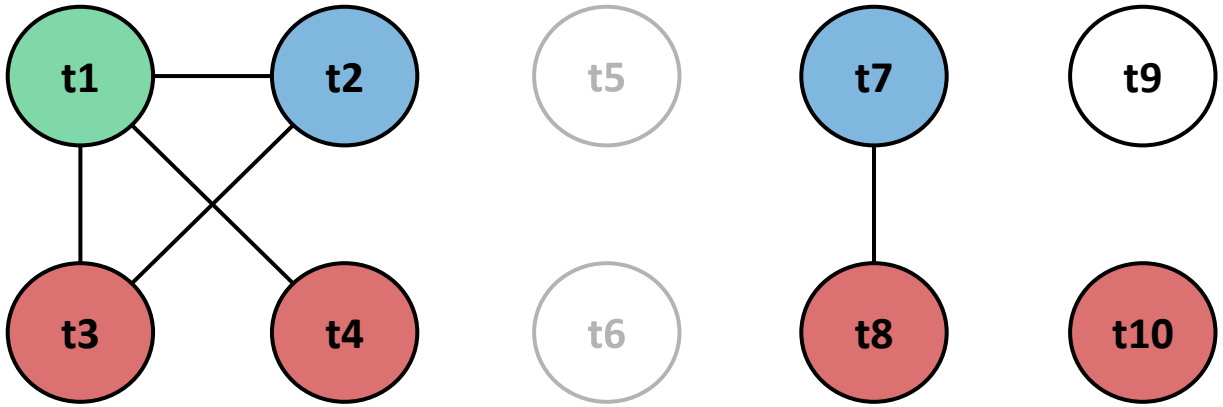


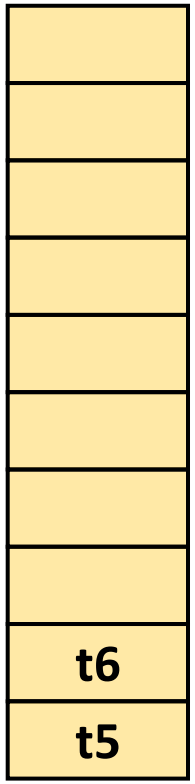
Graph Coloring (k = 3)



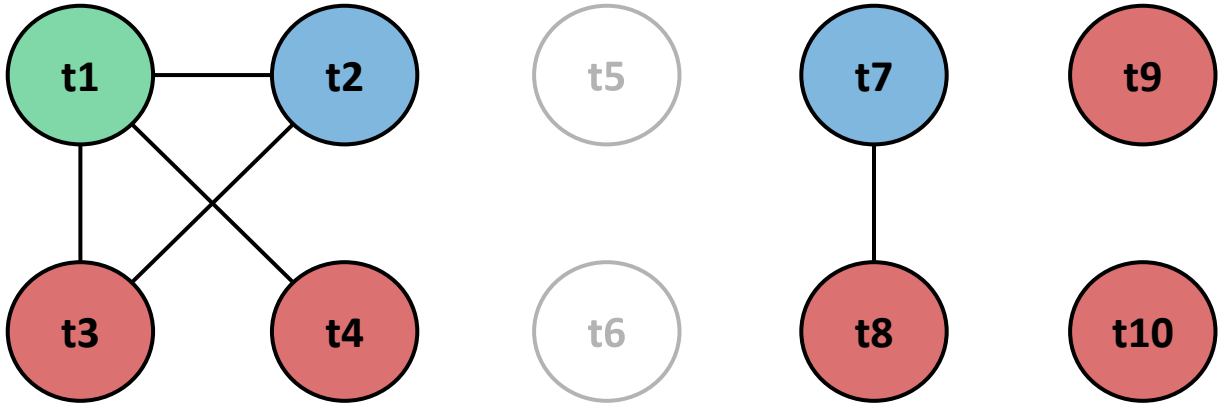


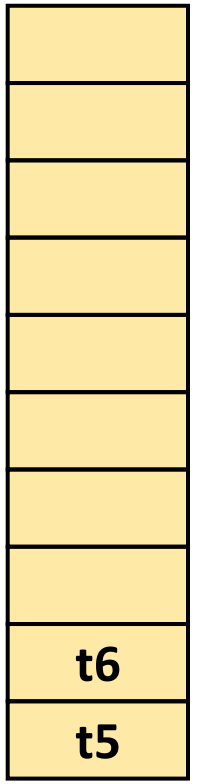
Graph Coloring (k = 3)



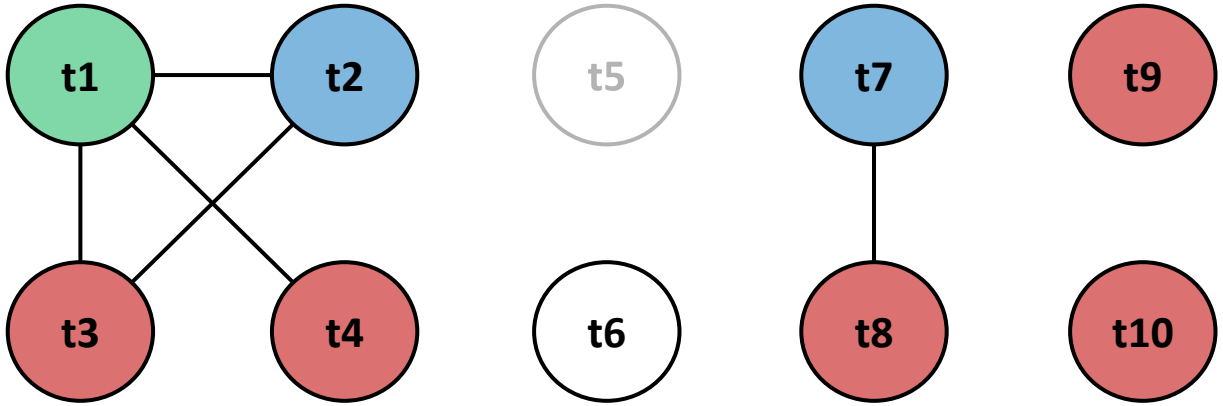


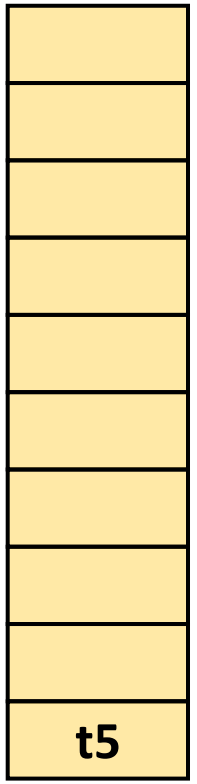
Graph Coloring (k = 3)



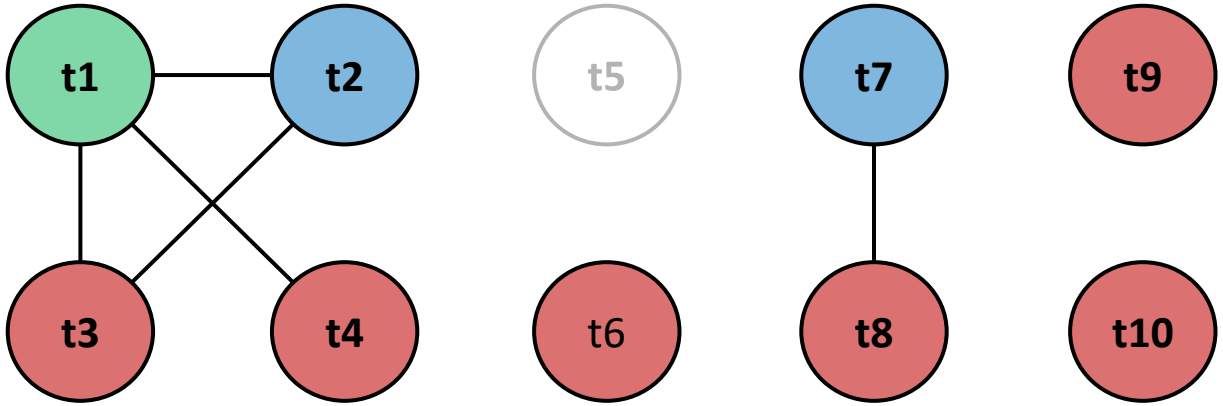


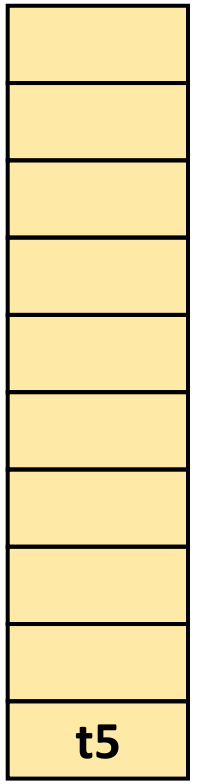
Graph Coloring (k = 3)



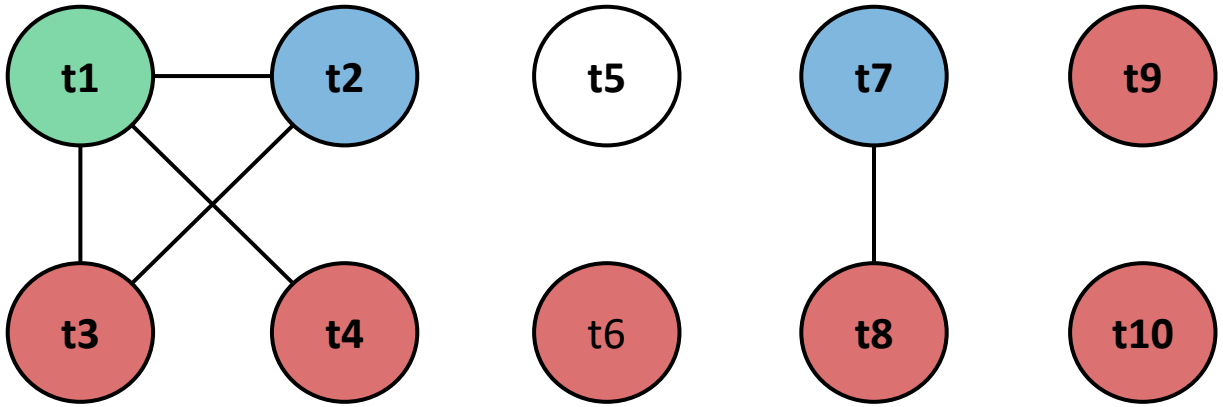


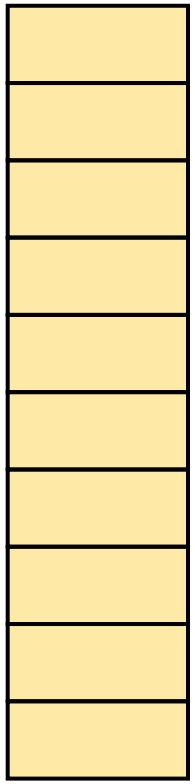
Graph Coloring (k = 3)



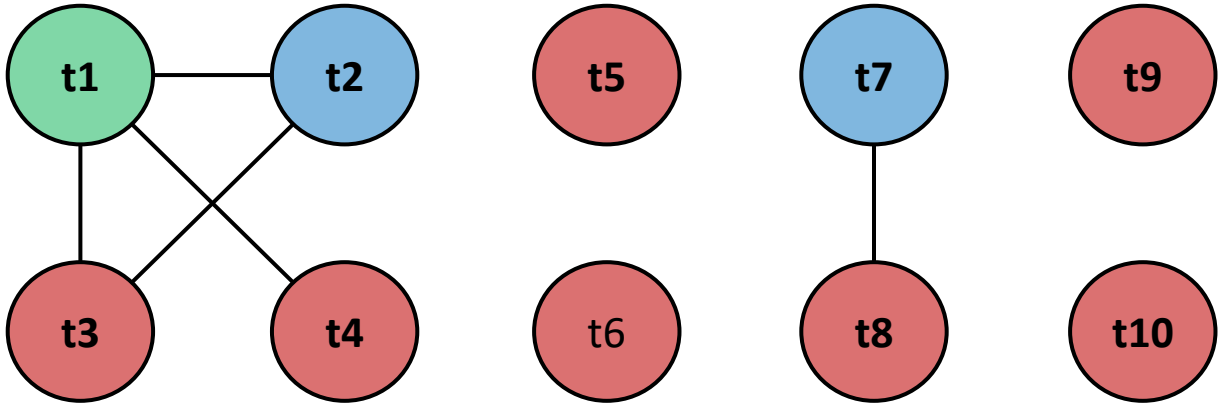


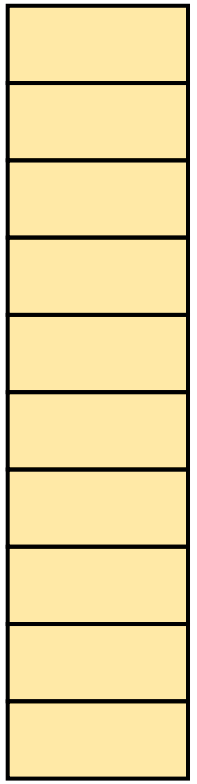
Graph Coloring (k = 3)





Graph Coloring (k = 3)





Register Allocation

For each function:

1. Construct the CFG (from the IR)
2. Run liveness analysis
3. Construct the interference graph
4. Compute a *k-coloring* of the graph
5. **Use the coloring to build the required mapping**

Register Allocation

- According to the coloring, our register allocation is:

IR Register	Color	MIPS Register
t1	R3	$t2
t2	R2	$t1
t3	R1	$t0
t4	R1	$t0
t5	R1	$t0
t6	R1	$t0
t7	R2	$t1
t8	R1	$t0
t9	R1	$t0
t10	R1	$t0

Register Allocation

```
t1 = x
t2 = y
t3 = z
t4 = sub t2, t3
t5 = mul t1, t4
a = t5
t6 = a
bne t6, 1, end
t7 = a
t8 = 1
t9 = add t7, t8
a = t9
end:
t10 = a
b = t10
```

IR Register	MIPS Register
t1	$t2
t2	$t1
t3	$t0
t4	$t0
t5	$t0
t6	$t0
t7	$t1
t8	$t0
t9	$t0
t10	$t0

```
lw $t1, 8($fp)
lw $t2, 12($fp)
lw $t3, 16($fp)
sub $t4, $t2, $t3
mul $t5, $t1, $t4
sw $t5, -44($fp)
lw $t6, -44($fp)
bne $t6, 1, end
lw $t7, -44($fp)
li $t8, 1
add $t9, $t7, $t8
sw $t9, -44($fp)
end:
lw $t10, -44($fp)
sw $t10, -48($fp)
```

Register Allocation

```
t1 = x
t2 = y
t3 = z
t4 = sub t2, t3
t5 = mul t1, t4
a = t5
t6 = a
bne t6, 1, end
t7 = a
t8 = 1
t9 = add t7, t8
a = t9
end:
t10 = a
b = t10
```

IR Register	MIPS Register
t1	$t2
t2	$t1
t3	$t0
t4	$t0
t5	$t0
t6	$t0
t7	$t1
t8	$t0
t9	$t0
t10	$t0

```
lw $t2, 8($fp)
lw $t1, 12($fp)
lw $t0, 16($fp)
sub $t0, $t1, $t0
mul $t0, $t2, $t0
sw $t0, -44($fp)
lw $t0, -44($fp)
bne $t0, 1, end
lw $t1, -44($fp)
li $t0, 1
add $t0, $t1, $t0
sw $t0, -44($fp)
end:
lw $t0, -44($fp)
sw $t0, -48($fp)
```

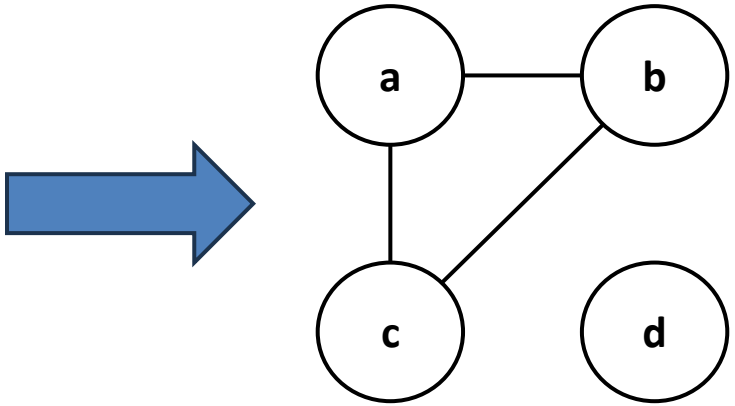
General Algorithm

- Spilling
  - We choose a “victim” variable to be stored in memory
  - Remove nodes from the graph

- Coalescing
  - Get rid or redundant MOV instructions (x = y)
  - Merge nodes in the graph

Example: Interference Graph

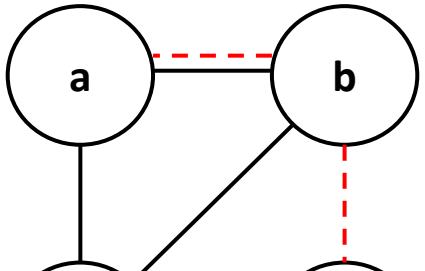
```
a = b
b = a + b
d = b
c = c + a
```



Example: MOV Edges

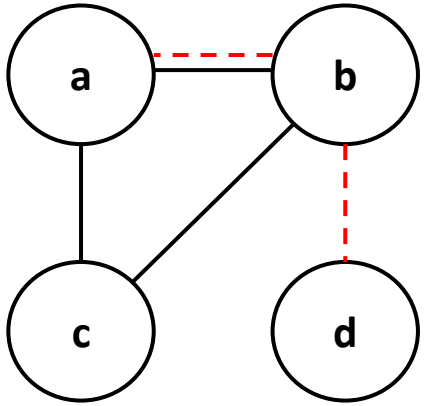
- Add dashed edges to MOV instructions

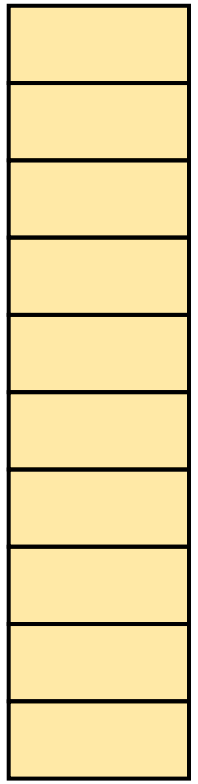
```
a = b
b = a + b
d = b
c = c + a
```



Example

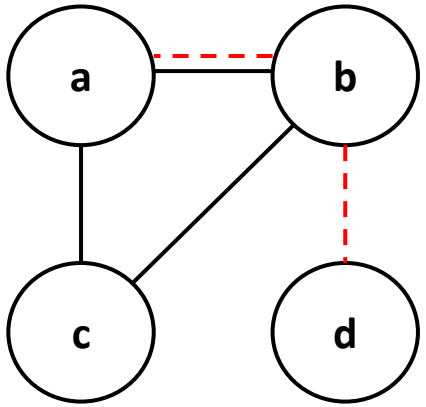
- We want to compute $k = 2$ coloring

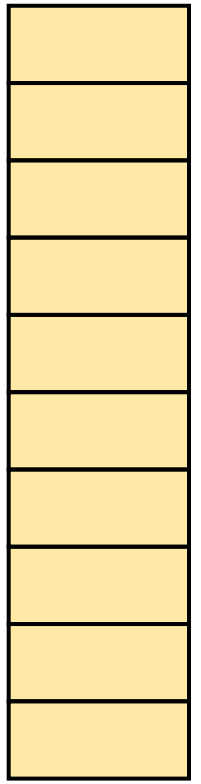




Example: Simplify

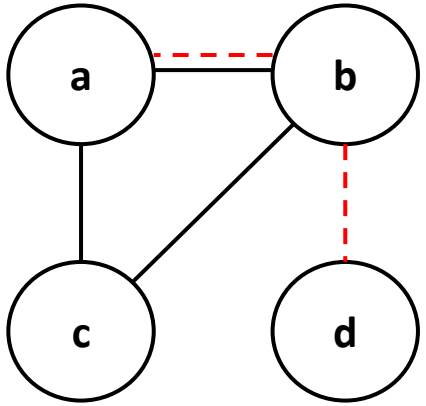
- We want to compute k = 2 coloring
- No node to simplify
  - All nodes have degree ≥ 2 or connected with a MOV edge

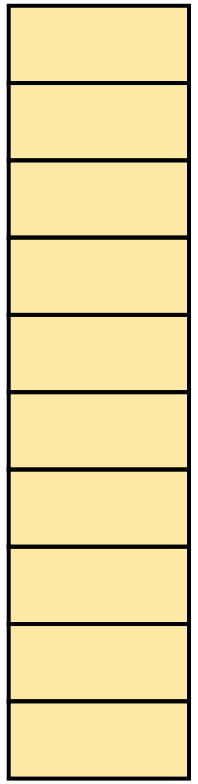




Example: Coalesce

- We want to compute k = 2 coloring
- Cannot coalesce a & b – constrained MOV
- Can we coalesce b & d?

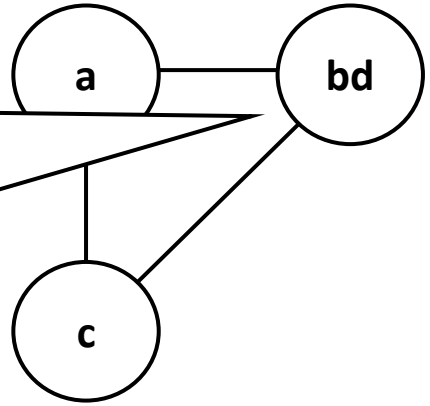


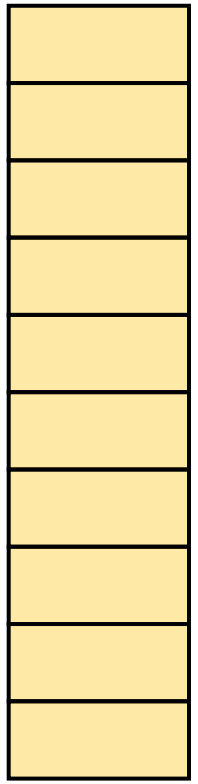


Example: Coalesce

- We want to compute k = 2 coloring
- Cannot coalesce a & b – constrained MOV
- Can we coalesce b & d?

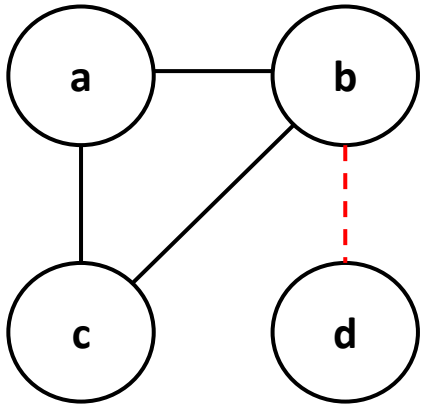
Briggs heuristic:
coalesce nodes x and y if the resulting node xy has less than k neighbors with degree ≥ k

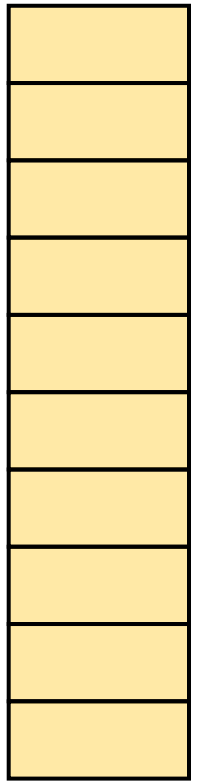




Example: Conservative Coalescing

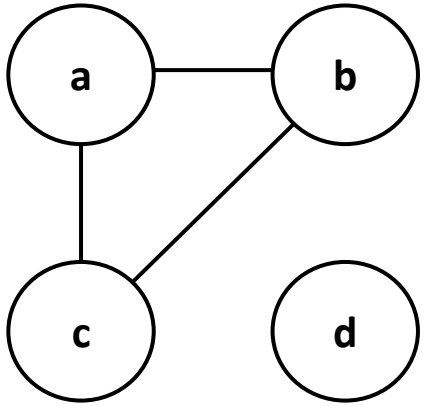
- We want to compute k = 2 coloring
- Following Briggs heuristic, we don't Coalesce b & d

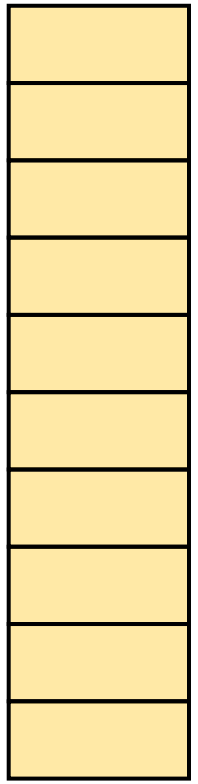




Example: Freeze

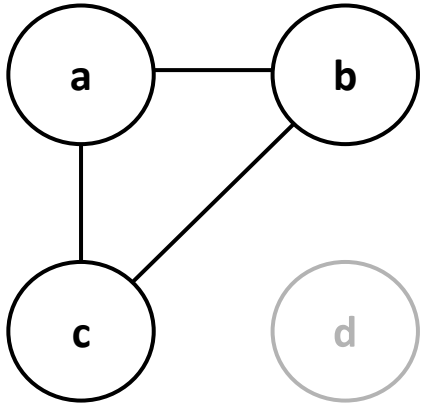
- We want to compute k = 2 coloring
- Remove MOV edge between b & d

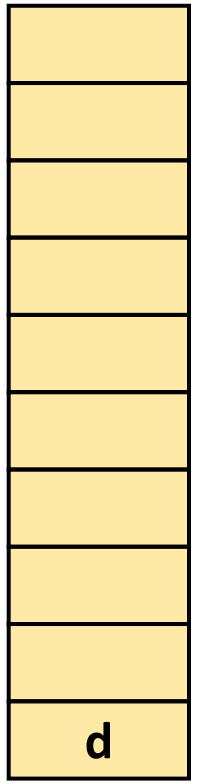




Example: Simplify

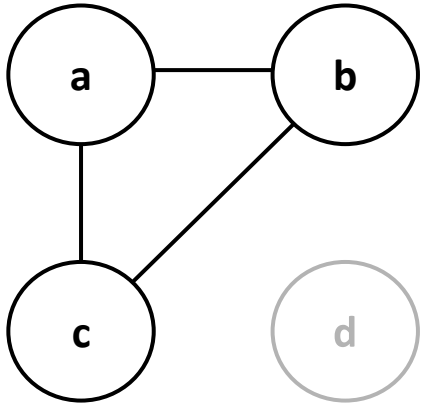
- We want to compute k = 2 coloring
- Simplify
- No more nodes to simplify…

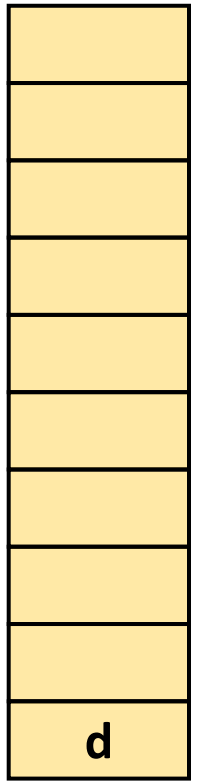




Example: Spilling

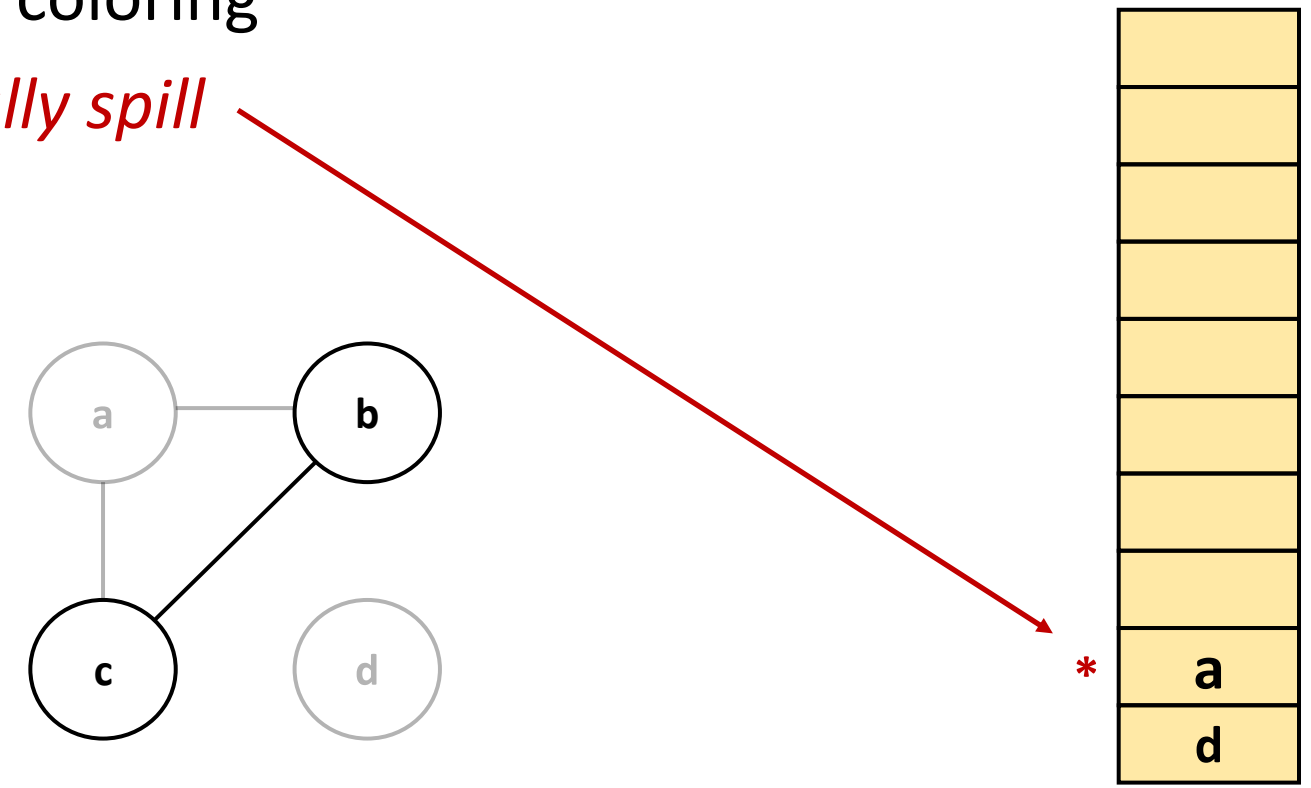
- We want to compute k = 2 coloring
- Choose a node to *potentially spill*





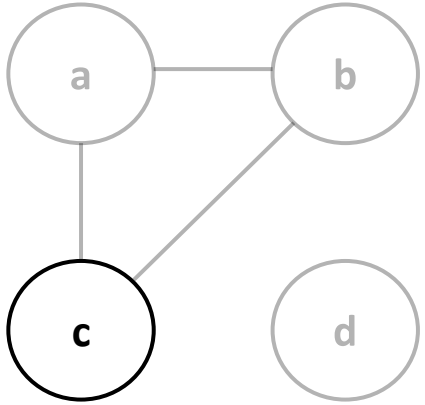
Example: Spilling

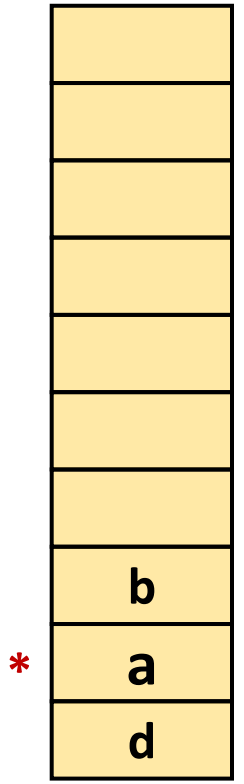
- We want to compute k = 2 coloring
- Choose a node to *potentially spill*



Example: Simplify

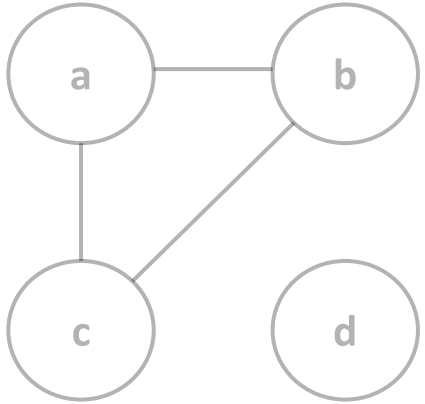
- We want to compute k = 2 coloring

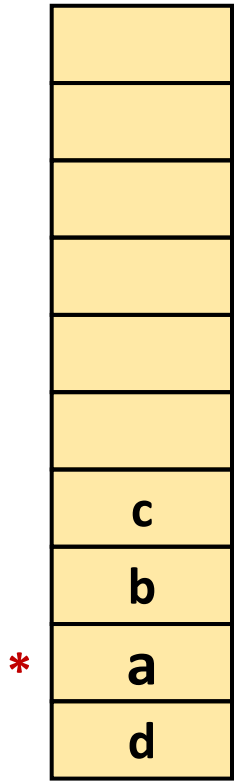




Example: Simplify

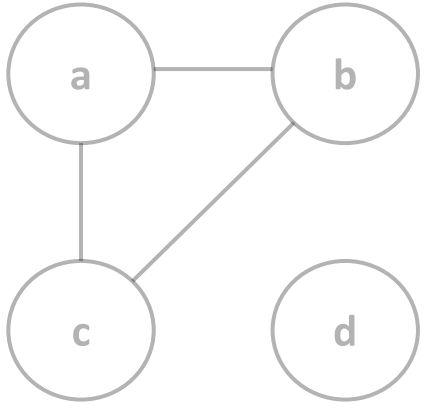
- We want to compute k = 2 coloring

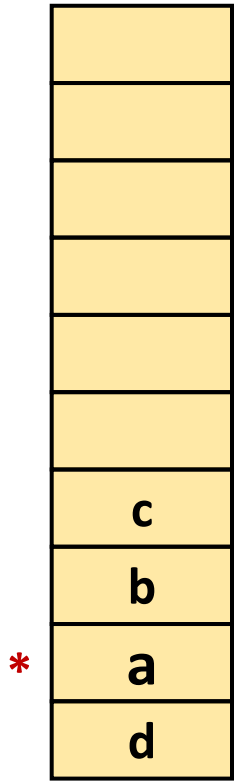




Example: Select

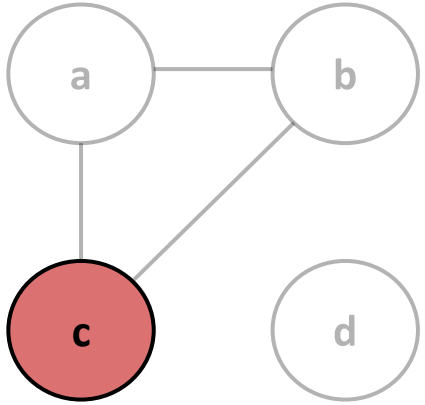
- We want to compute k = 2 coloring
- Pop and (try to) assign colors

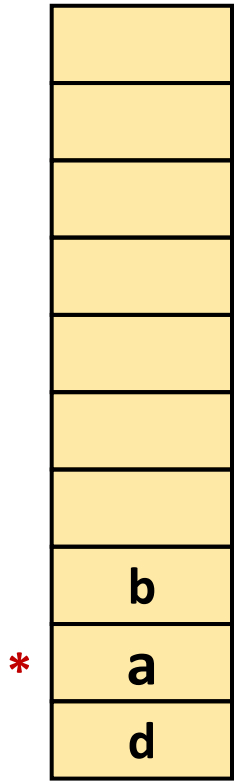




Example: Select

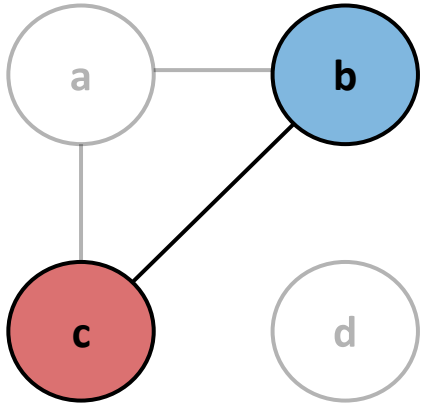
- We want to compute k = 2 coloring
- Pop and (try to) assign colors

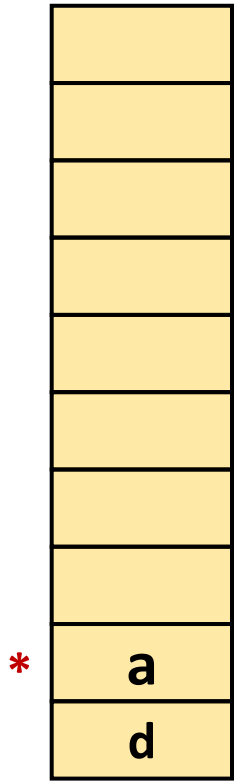




Example: Select

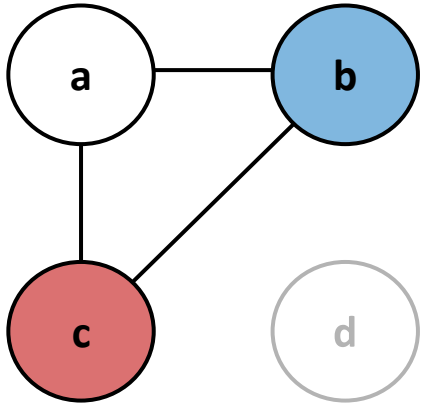
- We want to compute k = 2 coloring
- Pop and (try to) assign colors

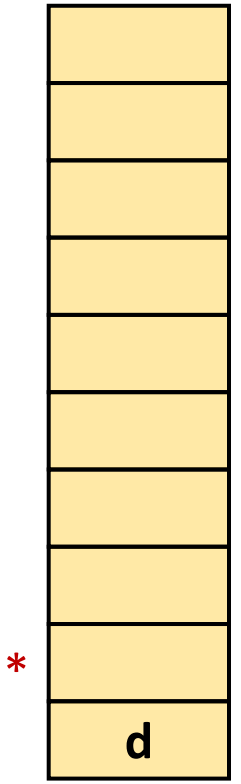




Example: Actually Spill

- We want to compute k = 2 coloring
- Pop and (try to) assign colors
- No color to assign a, need to *actually spill*





Example: Actually Spill

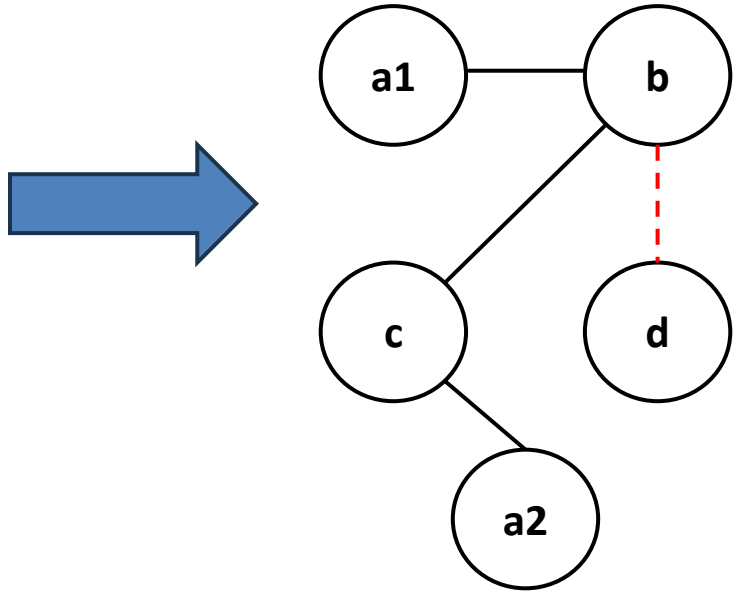
- Rewrite the IR & repeat

```
a = b
b = a + b
d = b
c = c + a
```

Example: Actually Spill

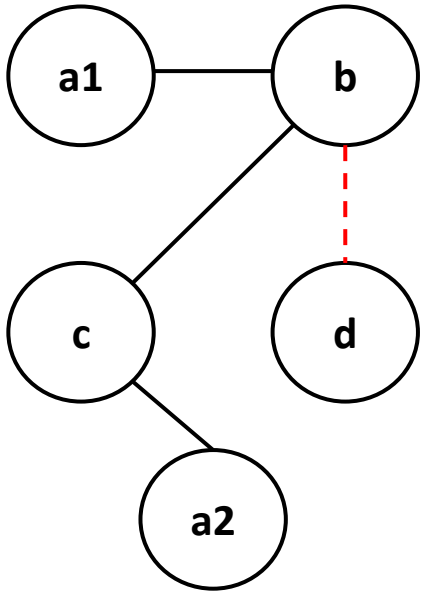
- Rewrite the IR & repeat

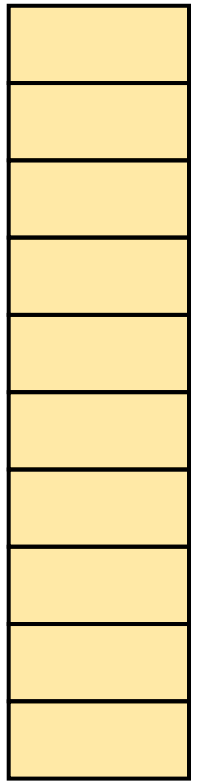
```
a1 = b
b = a1 + b
*(@a) = a1
d = b
a2 = *(@a)
c = c + a2
```



Example: Try Again

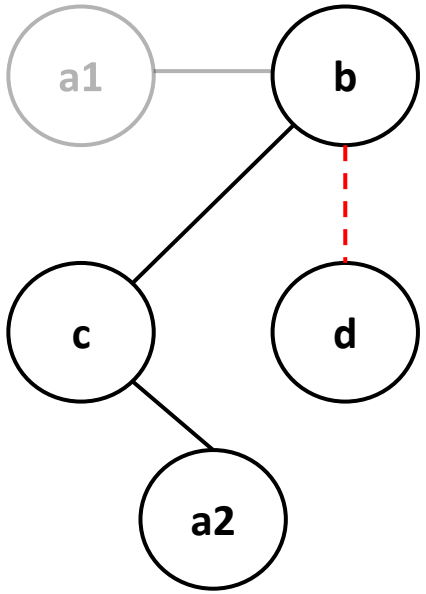
- We want to compute k = 2 coloring

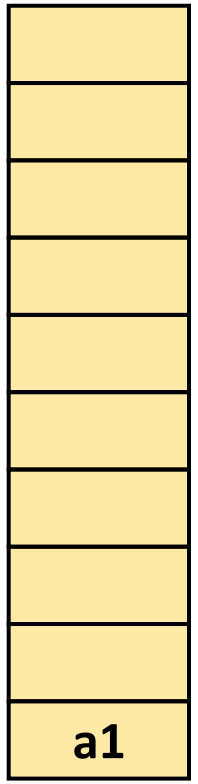




Example: Simplify

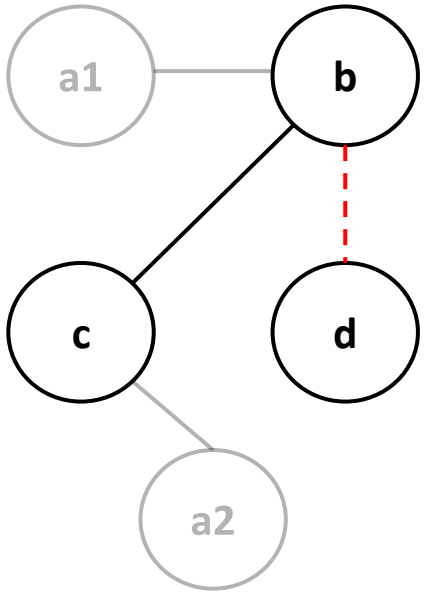
- We want to compute k = 2 coloring
- Simplify

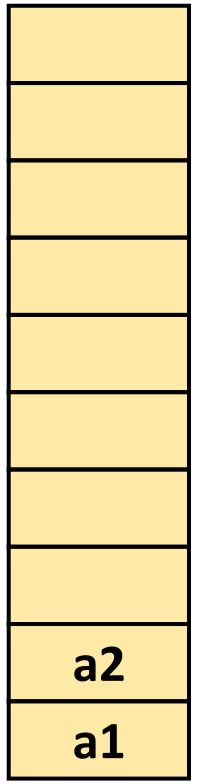




Example: Simplify

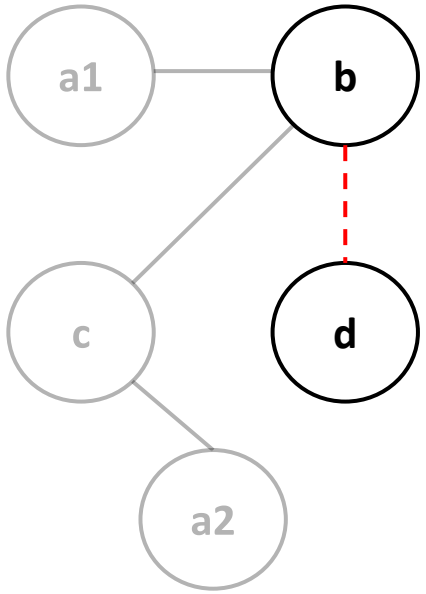
- We want to compute k = 2 coloring
- Simplify

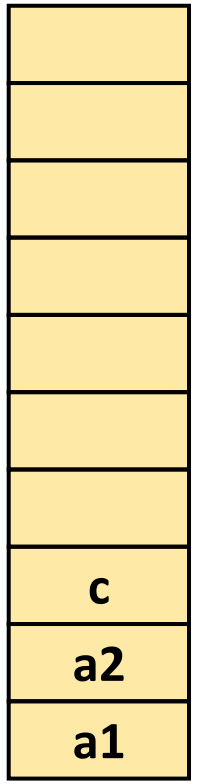




Example: Simplify

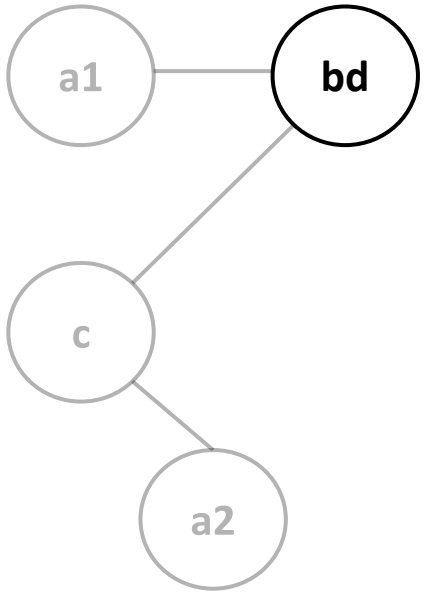
- We want to compute k = 2 coloring
- Simplify

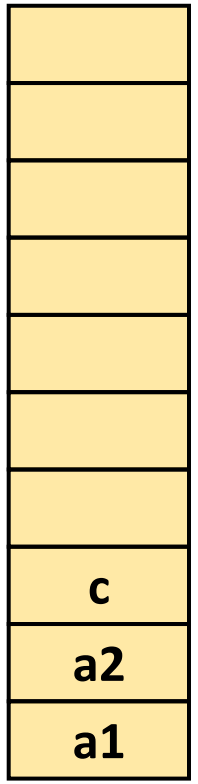




Example: Coalesce

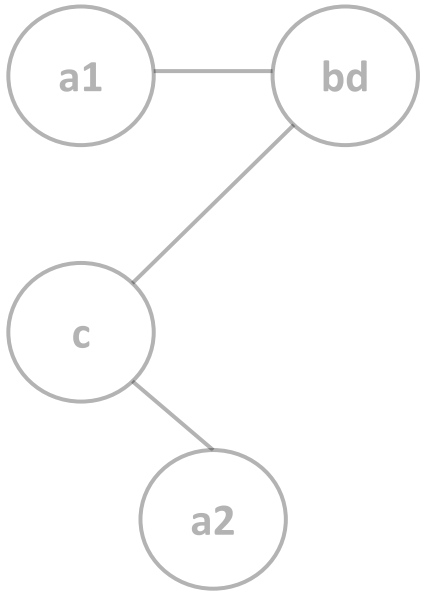
- We want to compute k = 2 coloring
- Coalesce b & d

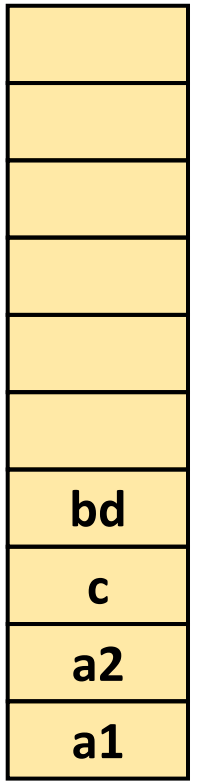




Example: Simplify

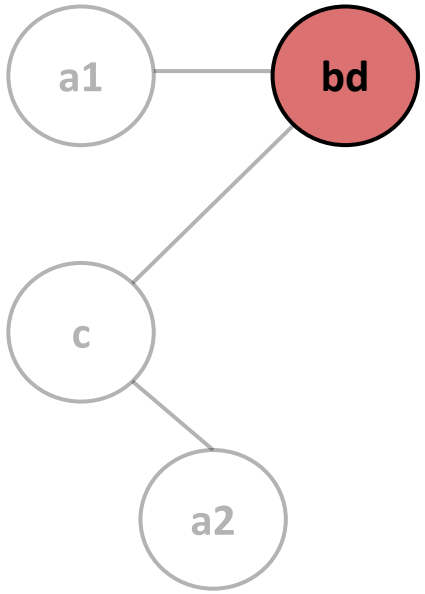
- We want to compute k = 2 coloring
- Simplify

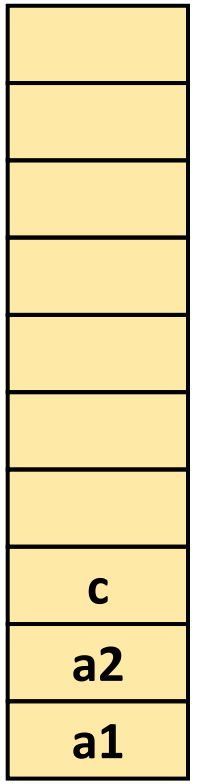




Example: Select

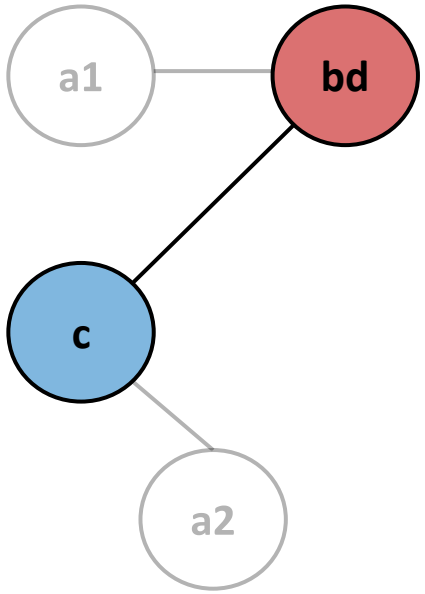
- We want to compute k = 2 coloring
- Pop and assign colors

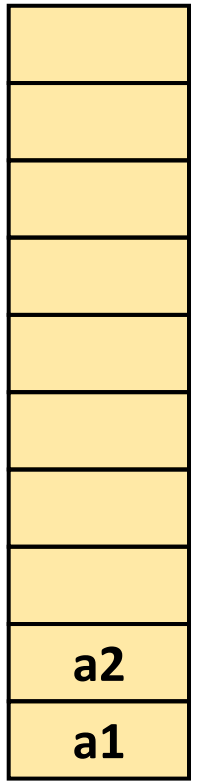




Example: Select

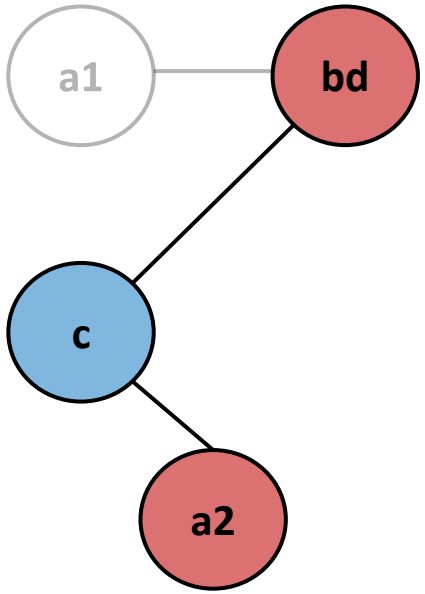
- We want to compute k = 2 coloring
- Pop and assign colors

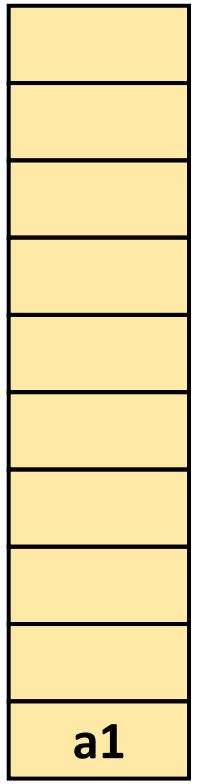




Example: Select

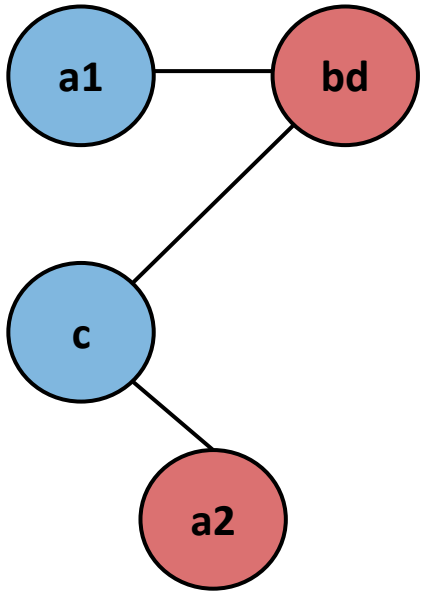
- We want to compute k = 2 coloring
- Pop and assign colors

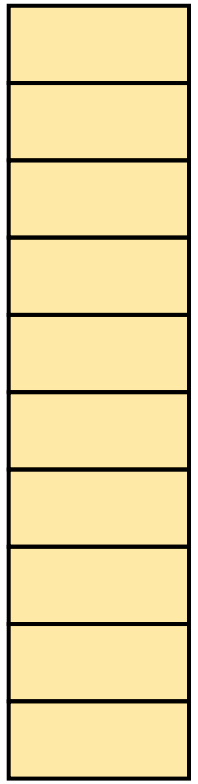




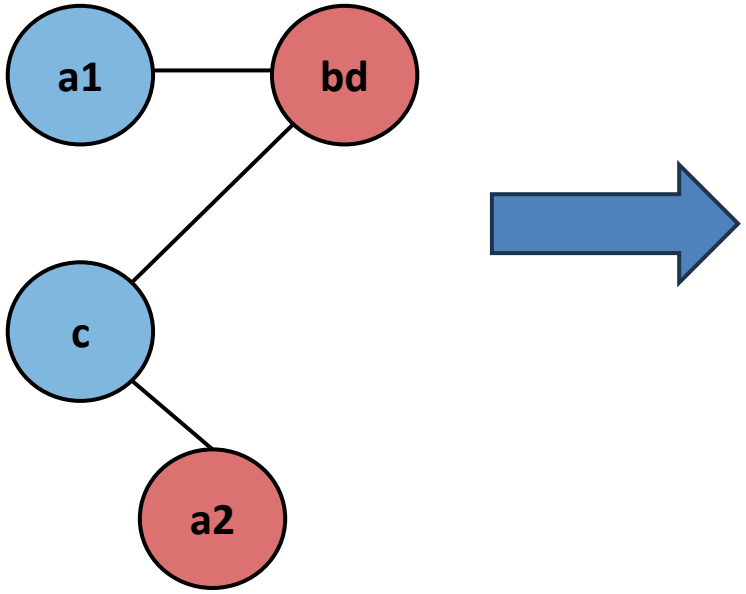
Example: Select

- We want to compute k = 2 coloring
- Pop and assign colors





Example: Final Register Allocation




```
R2 = R1
R1 = R2 + R1
*(@a) = R2
R1 = *(@a)
R2 = R2 + R1
```

In the Project

- We do register allocation on the IR
- Allocate physical registers only to IR registers
- Implement only the simplified register allocation algorithm without spilling or coalescing
- If need to actually spill (coloring failed) output an error message