

Register Allocation

Eelco Visser



CS4200 | Compiler Construction | November 25, 2021

Allocate Minimal Number of Registers

```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Interference graphs

- construction during liveness analysis

Interference graphs

- construction during liveness analysis

Graph Coloring

- assign registers to local variables and compiler temporaries
- store local variables and temporaries in memory

Interference graphs

- construction during liveness analysis

Graph Coloring

- assign registers to local variables and compiler temporaries
- store local variables and temporaries in memory

Coalescing

- handle move instructions

Interference graphs

- construction during liveness analysis

Graph Coloring

- assign registers to local variables and compiler temporaries
- store local variables and temporaries in memory

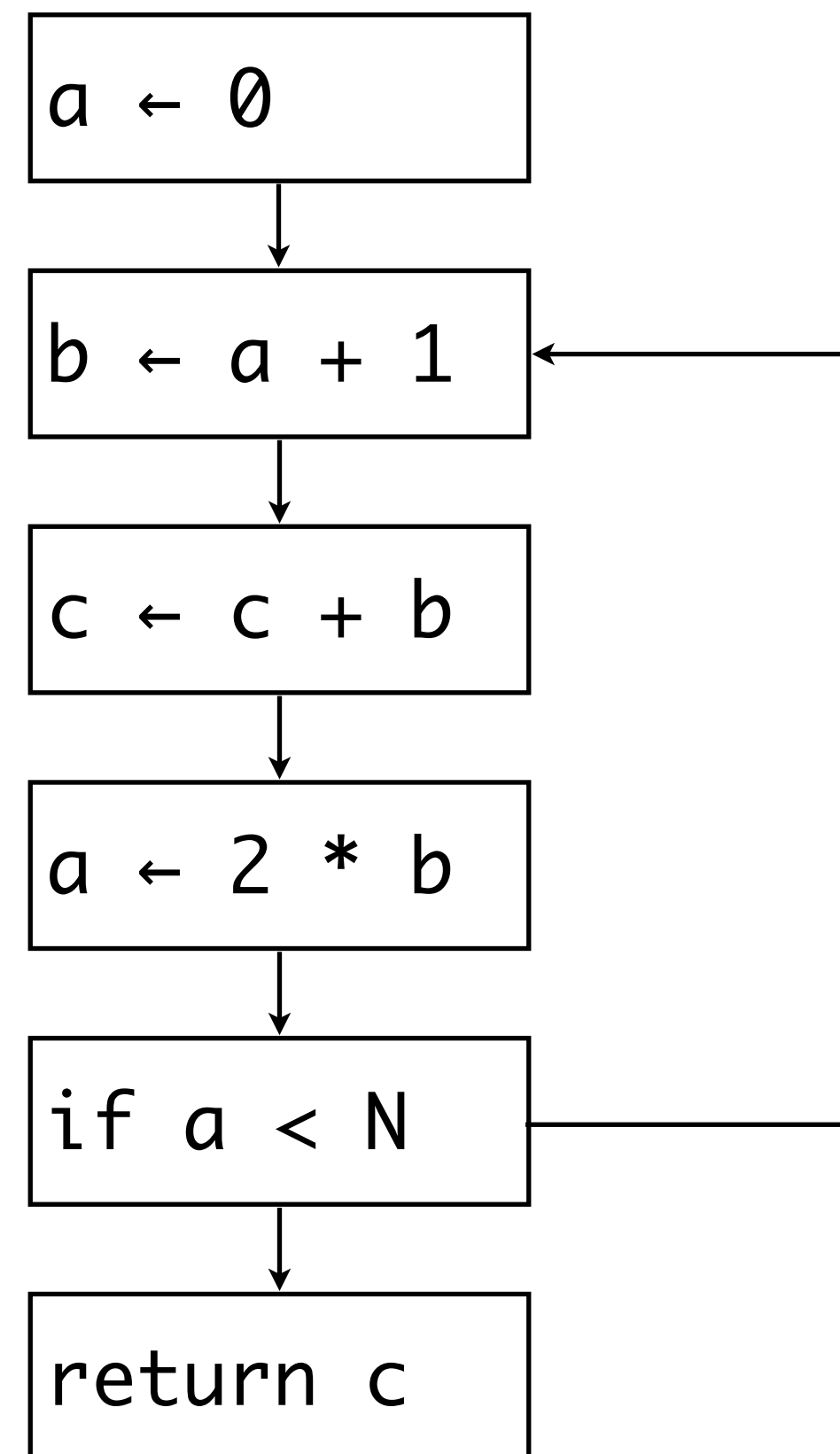
Coalescing

- handle move instructions

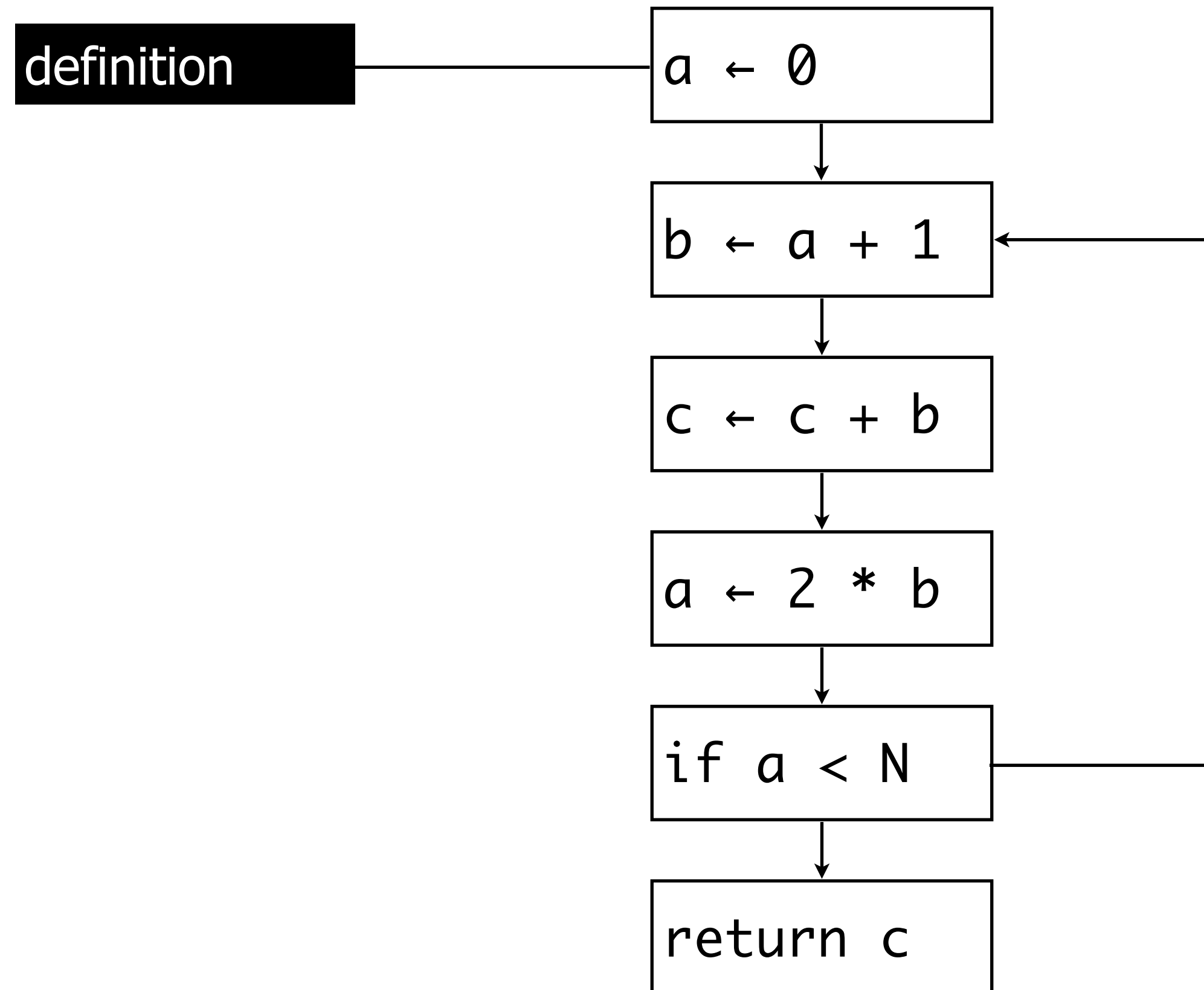
Pre-colored nodes

Interference Graphs

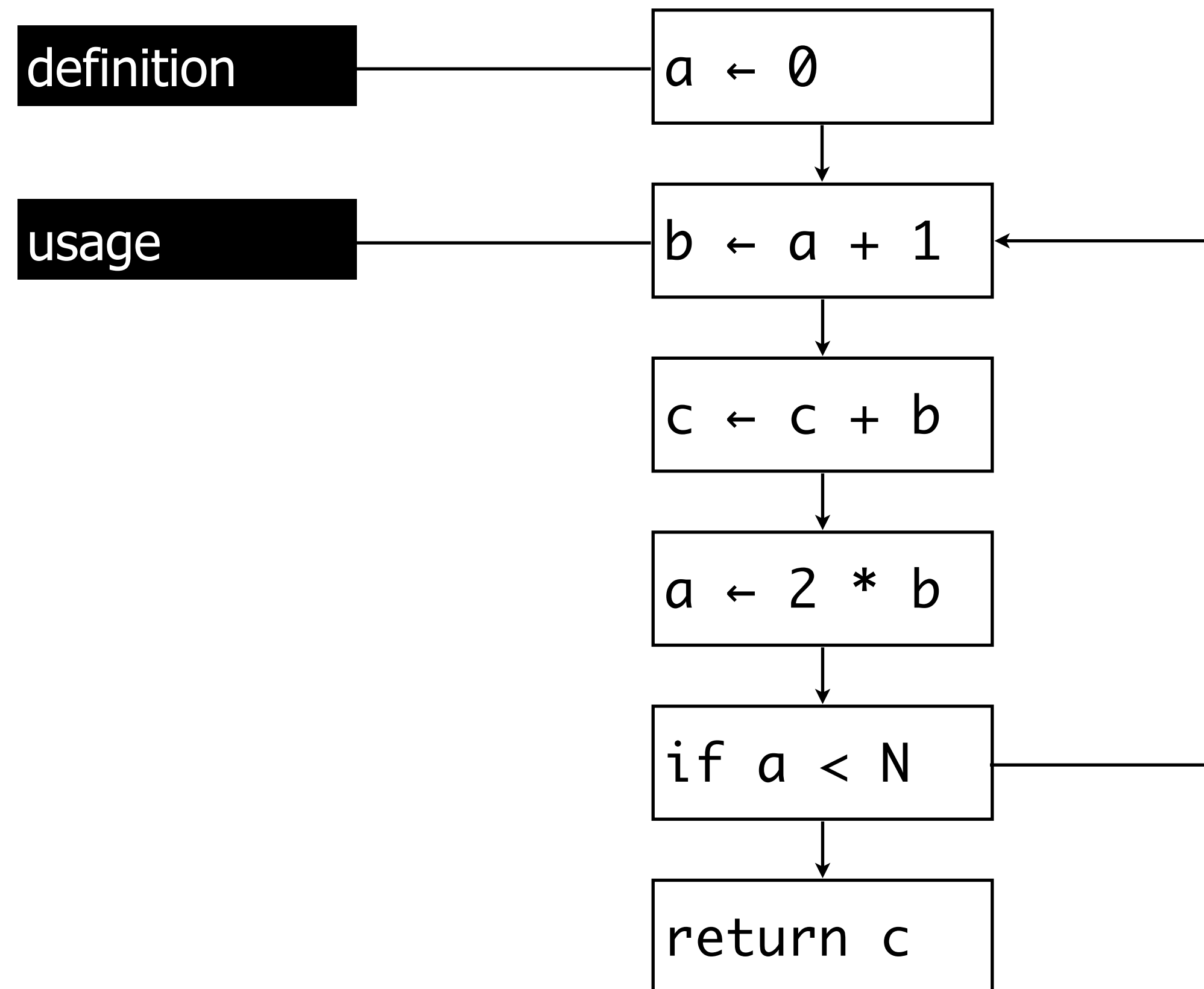
Liveness Analysis



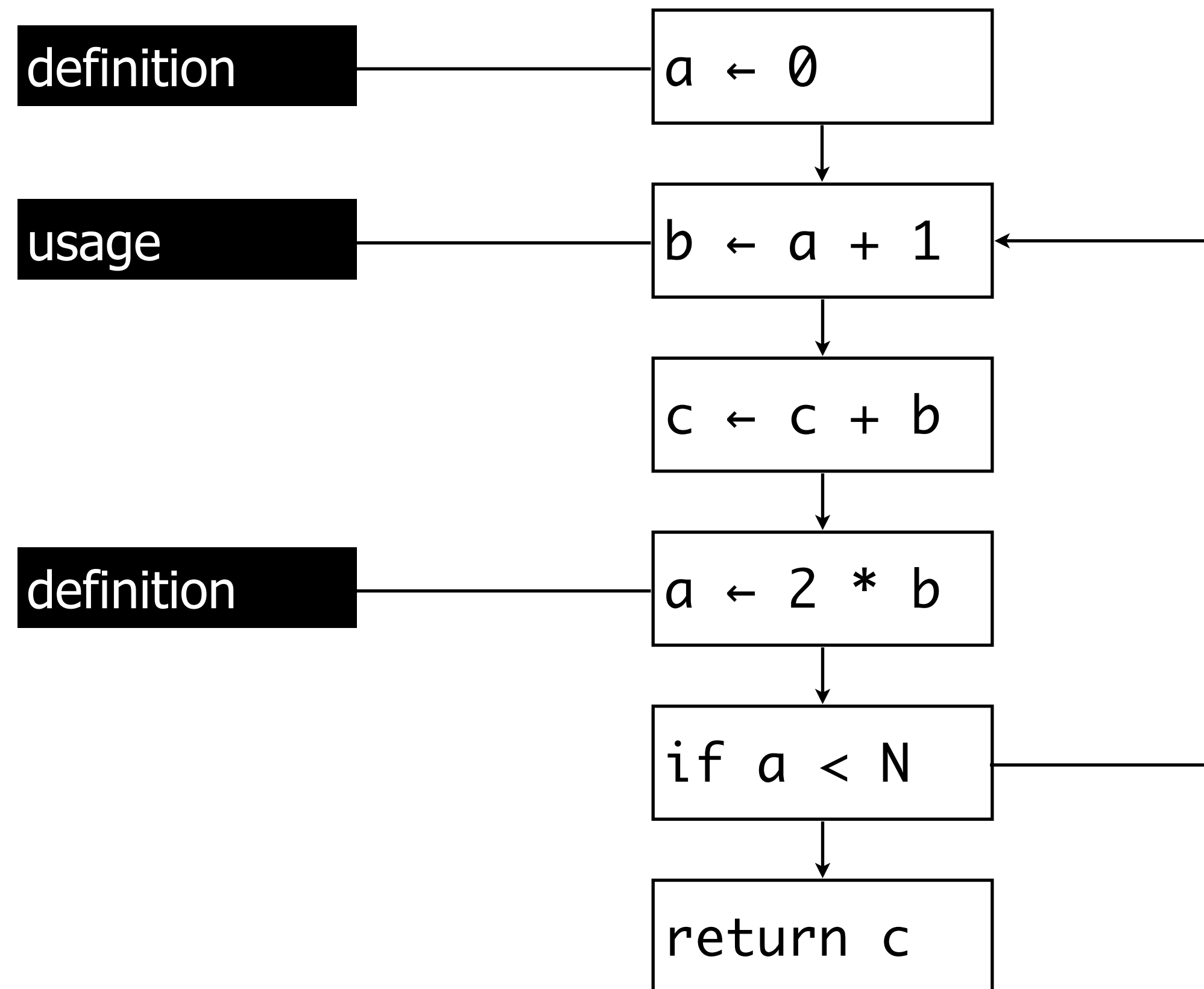
Liveness Analysis



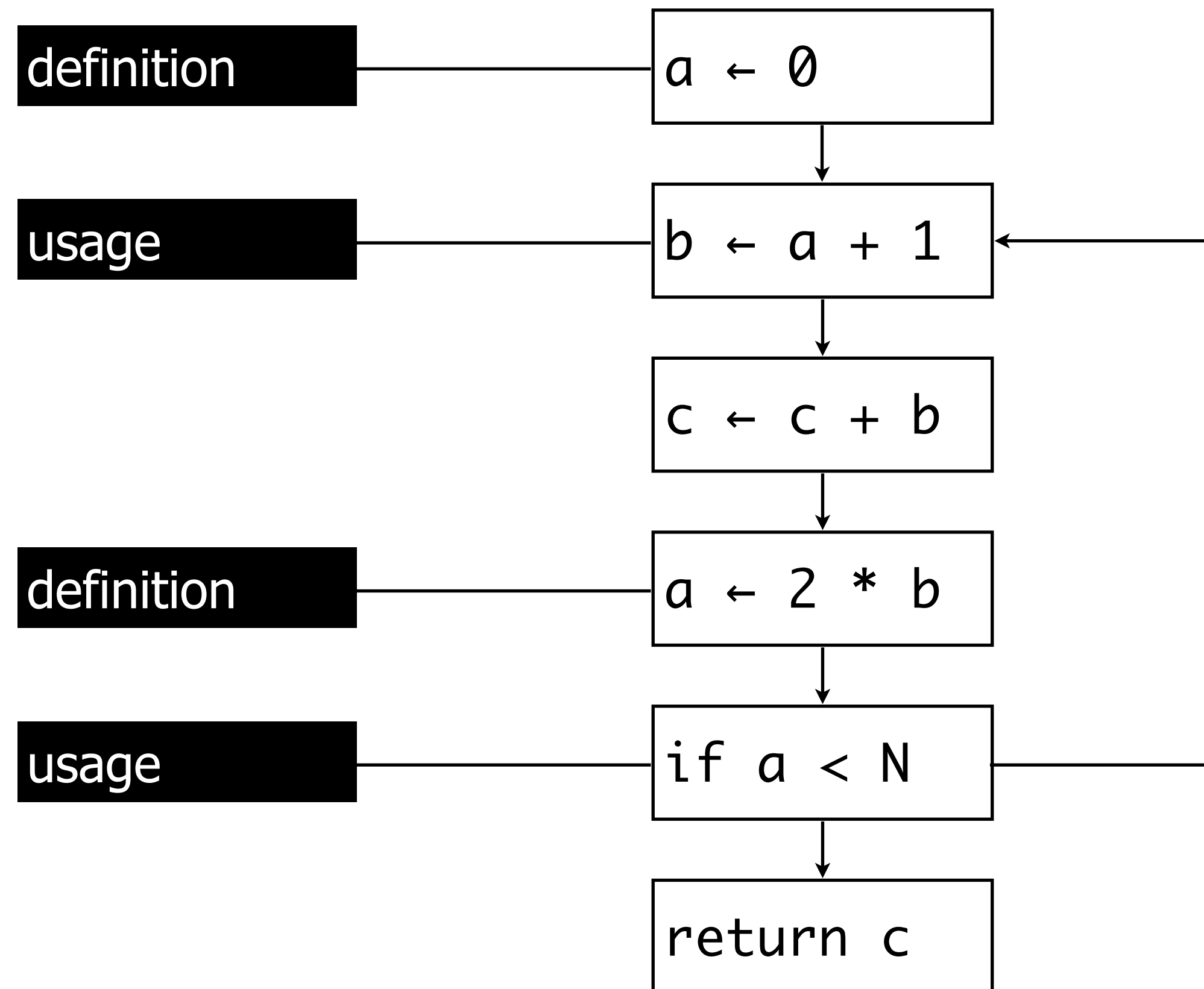
Liveness Analysis



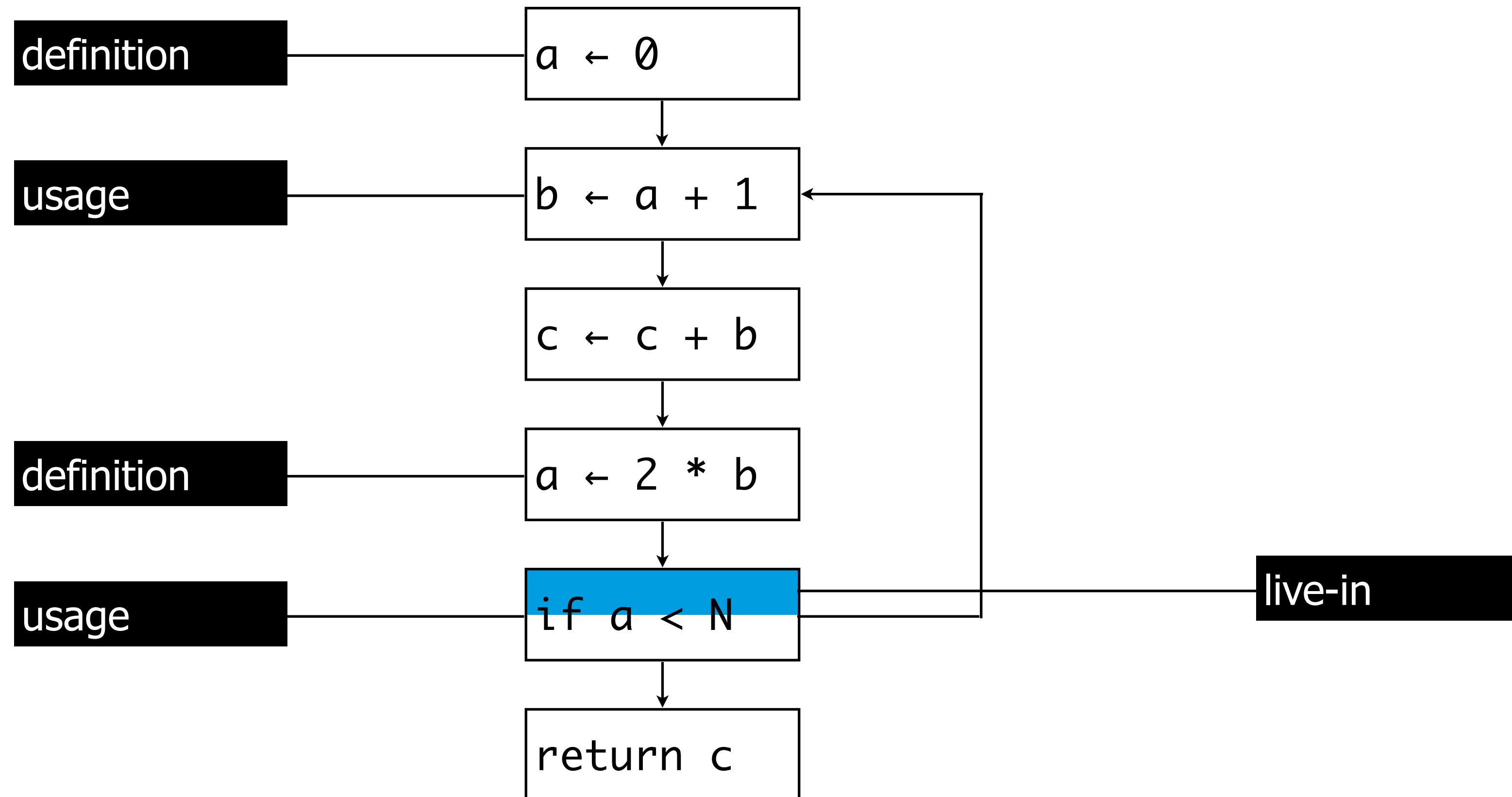
Liveness Analysis



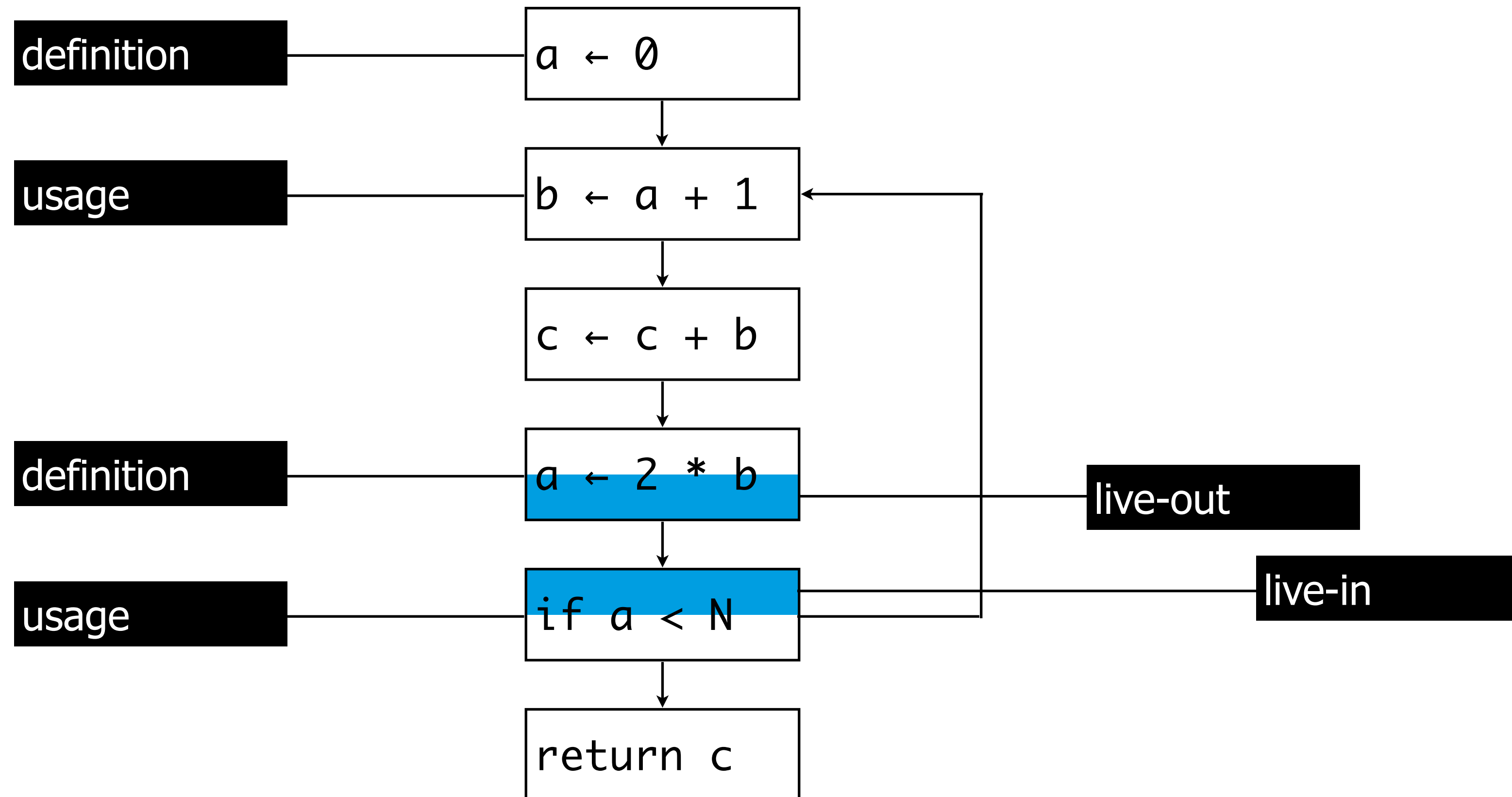
Liveness Analysis



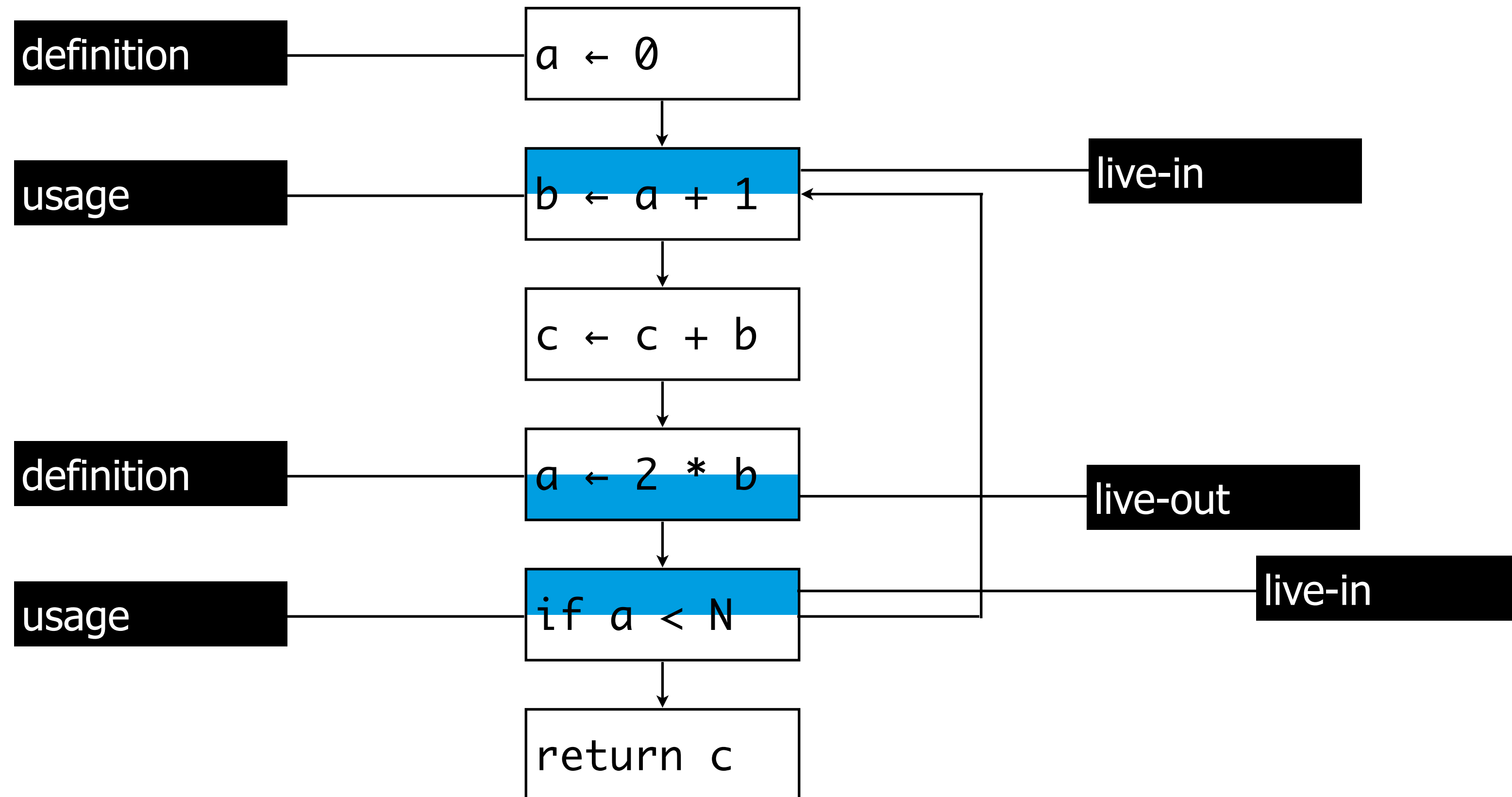
Liveness Analysis



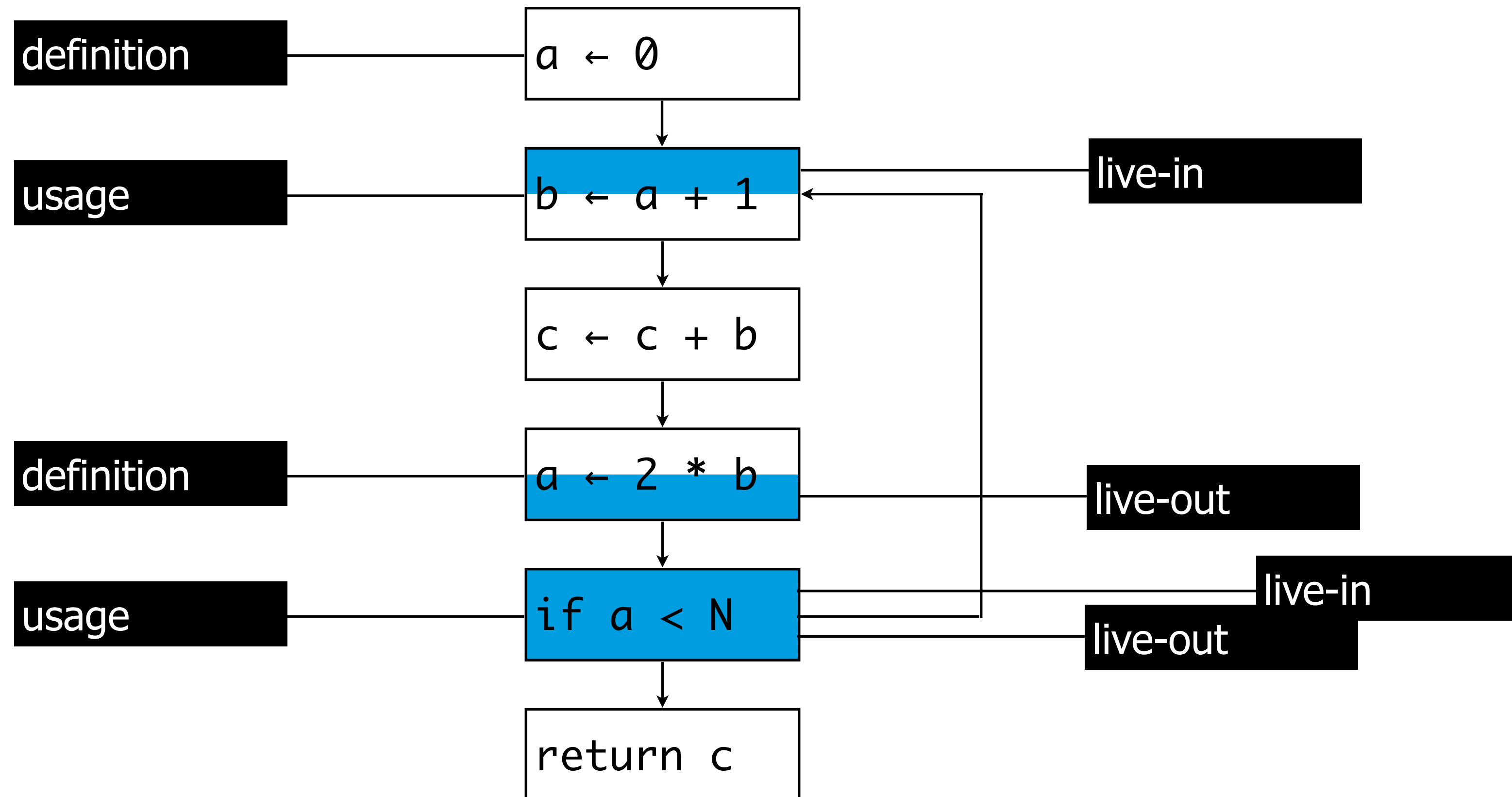
Liveness Analysis



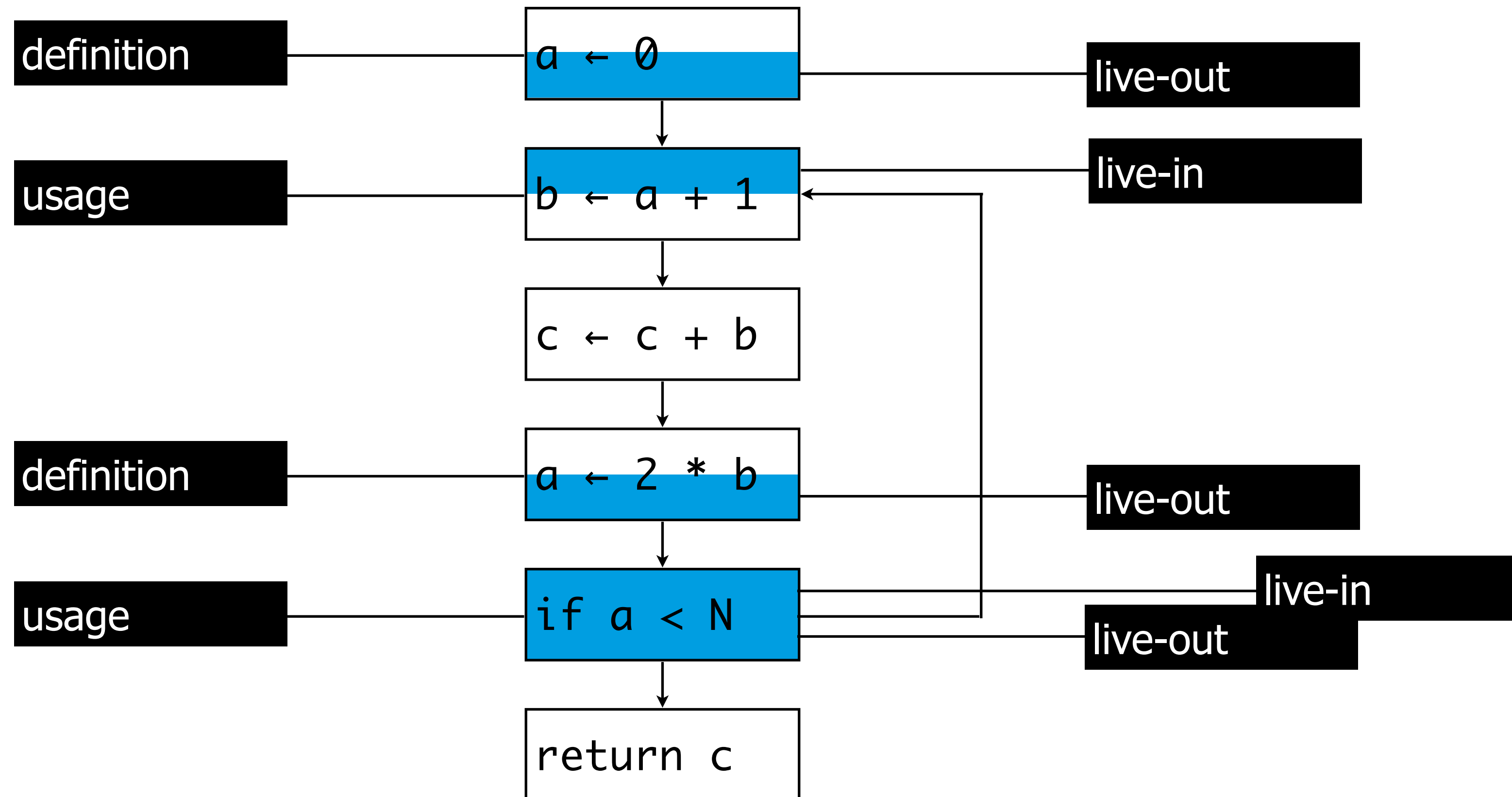
Liveness Analysis



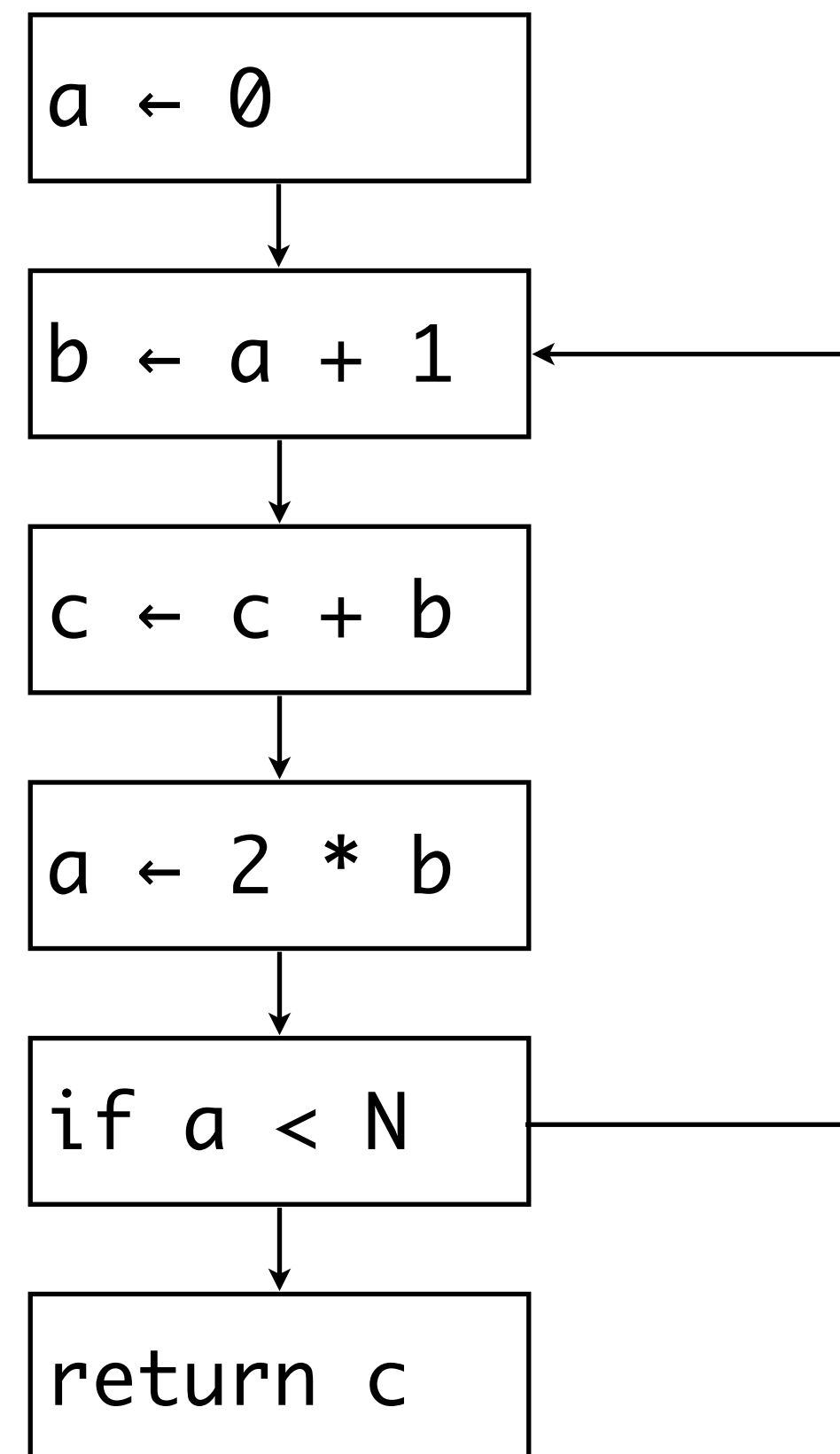
Liveness Analysis



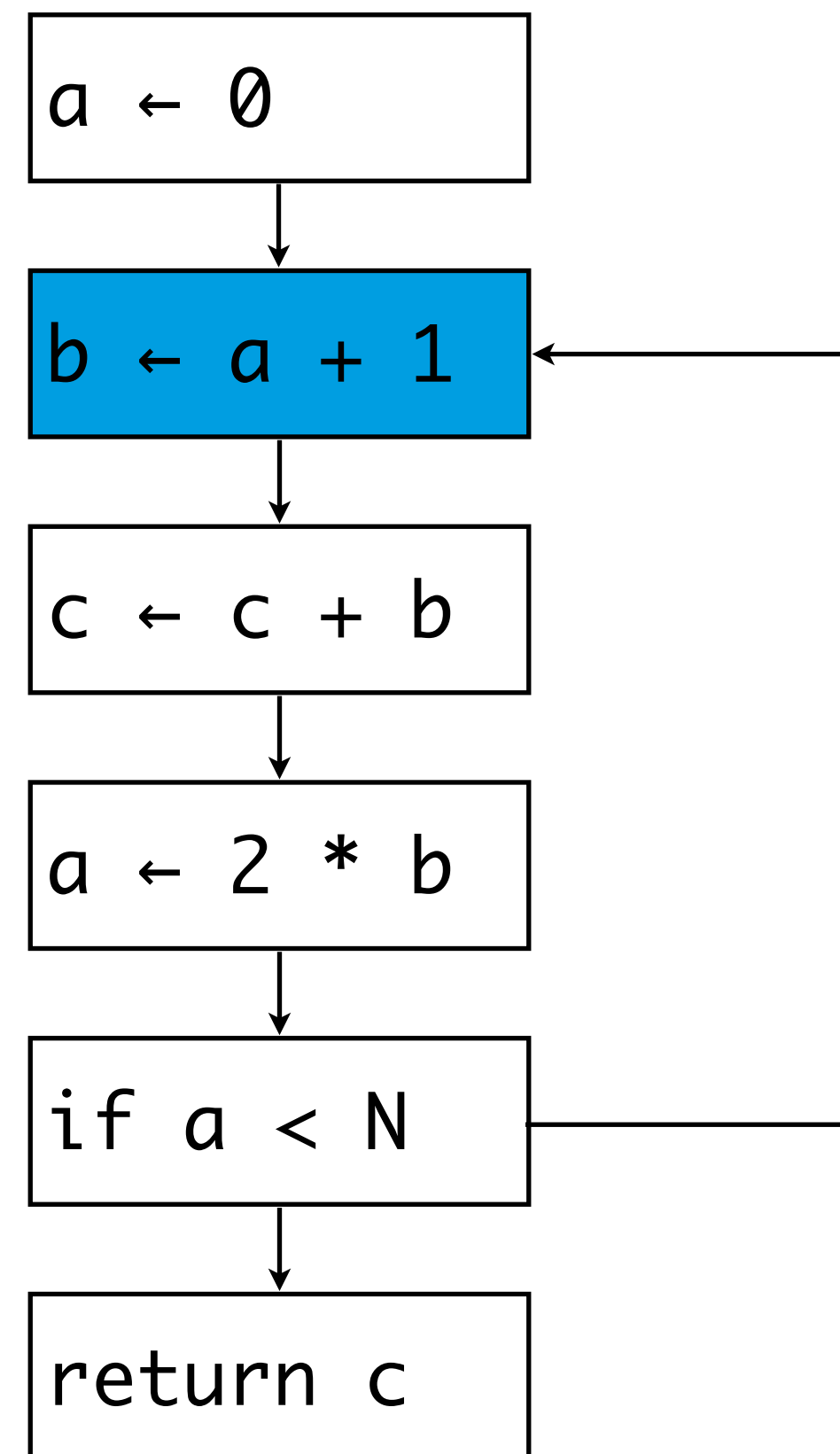
Liveness Analysis



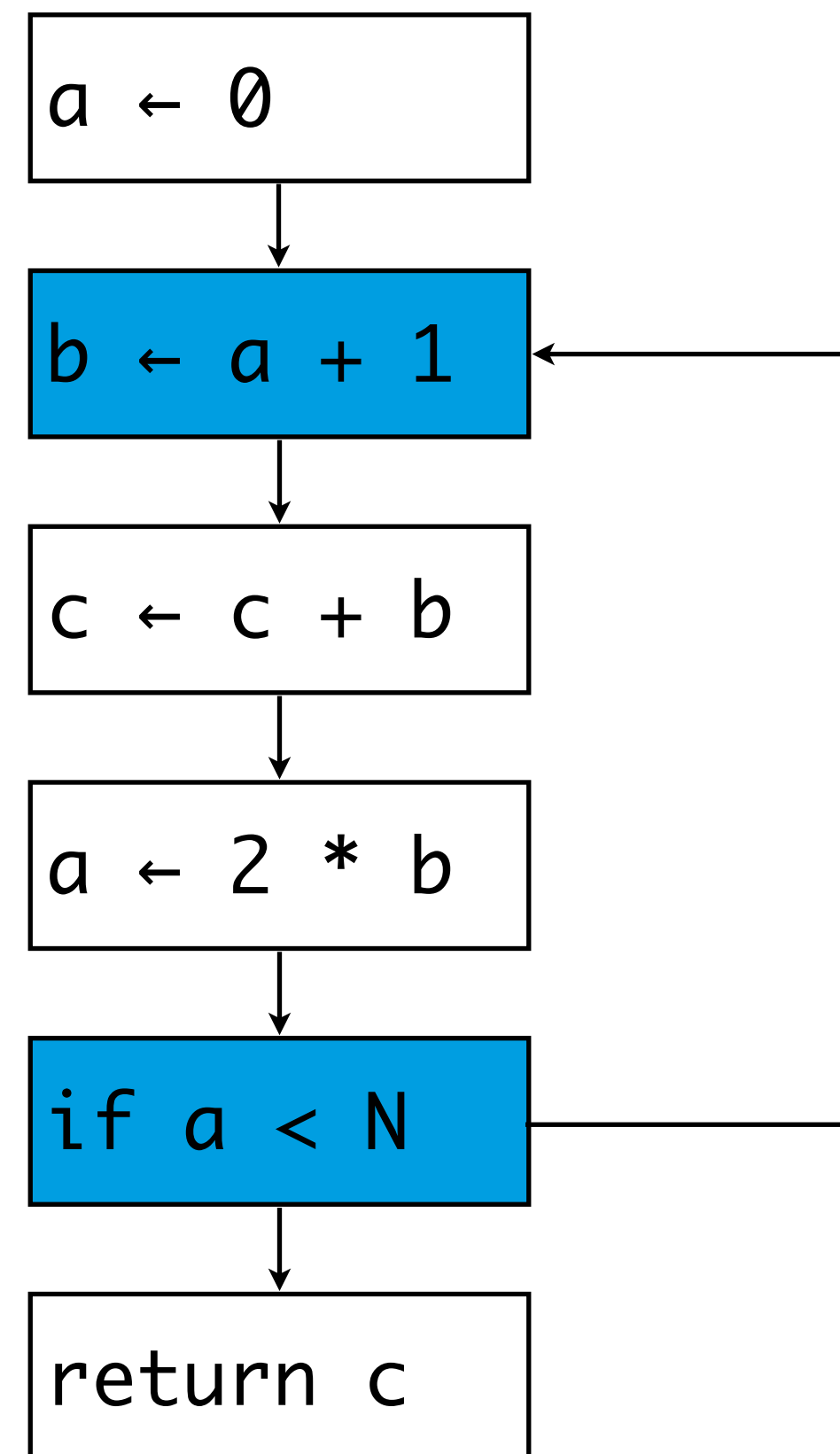
Liveness Analysis



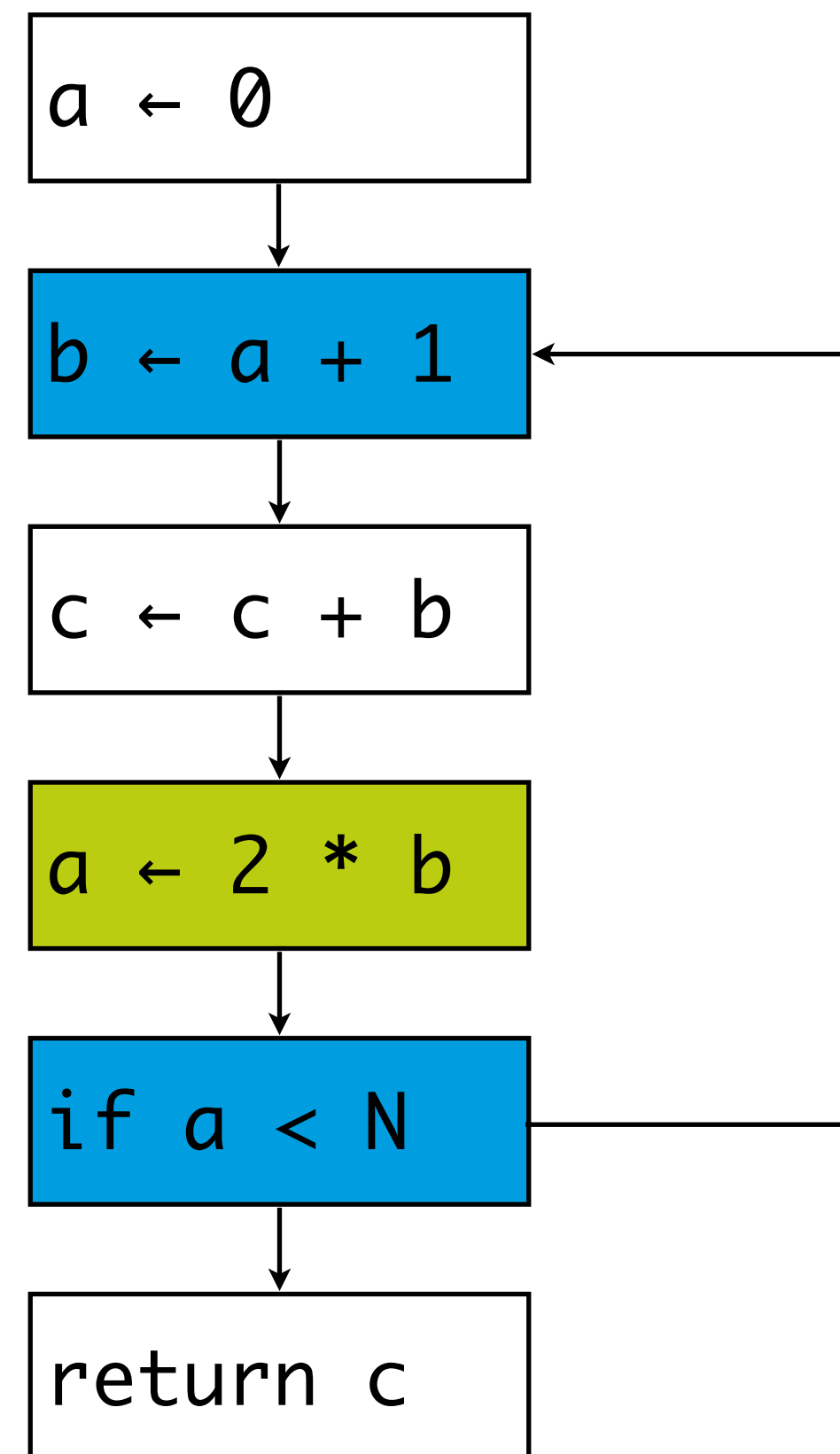
Liveness Analysis



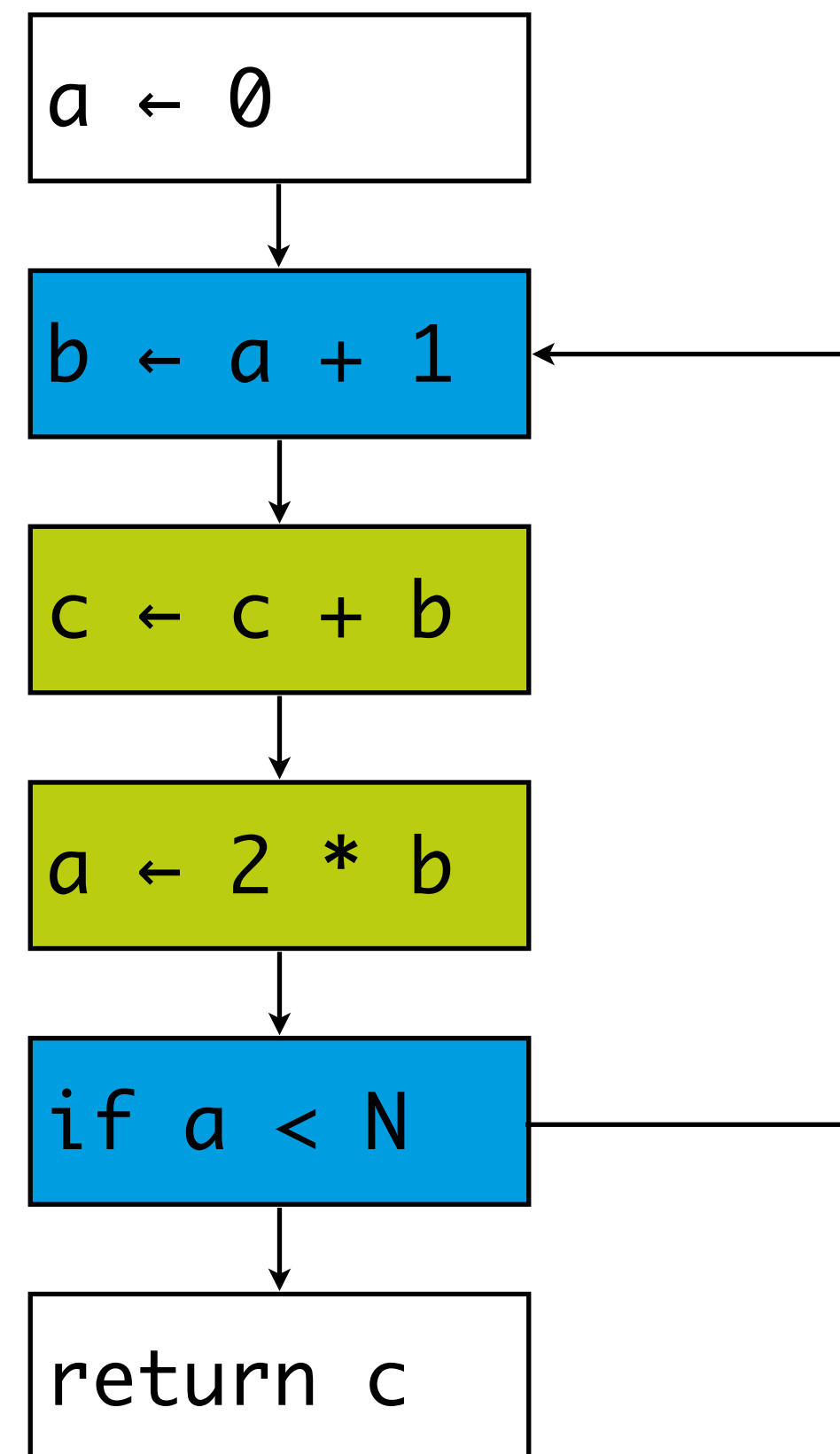
Liveness Analysis



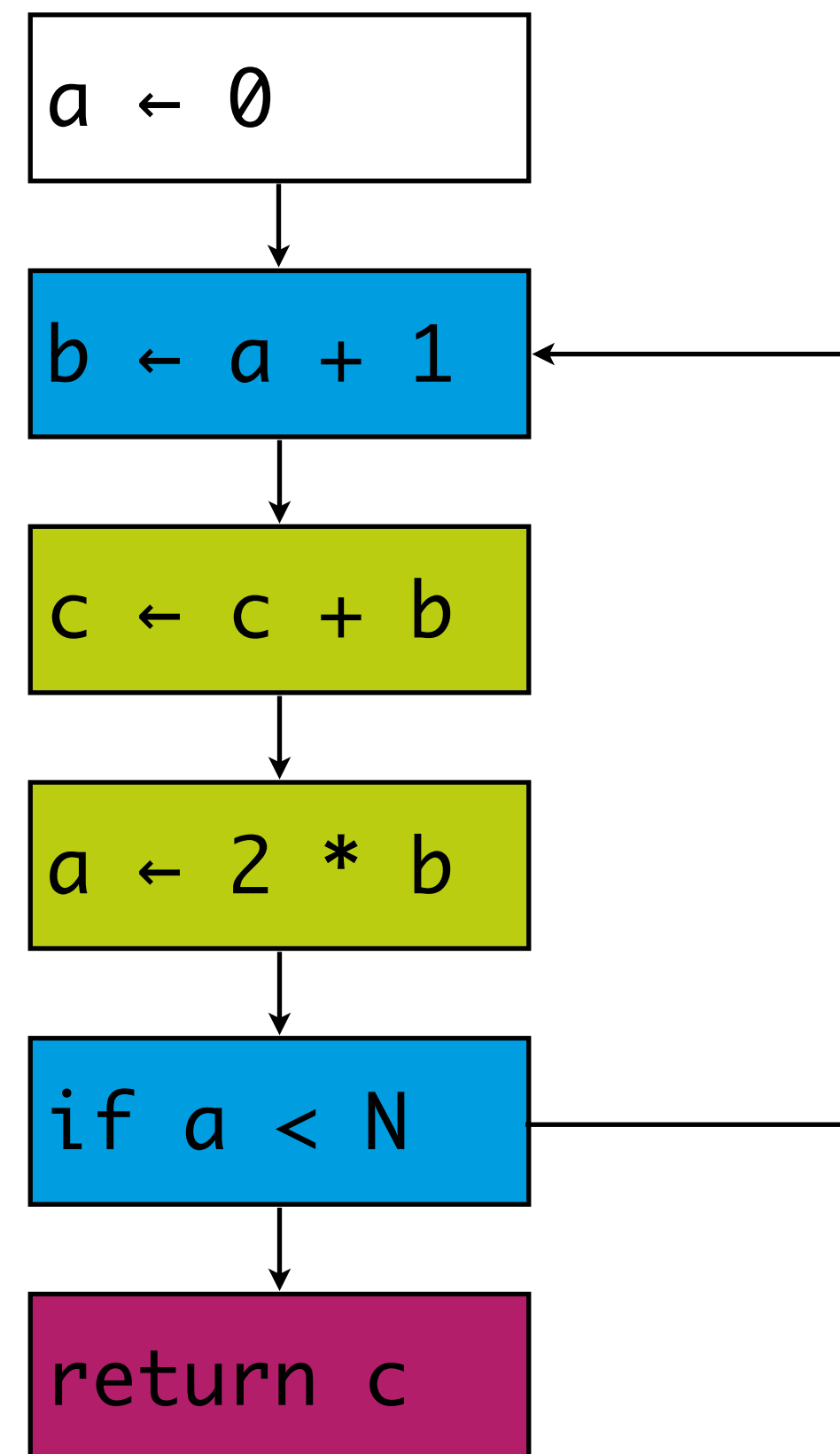
Liveness Analysis



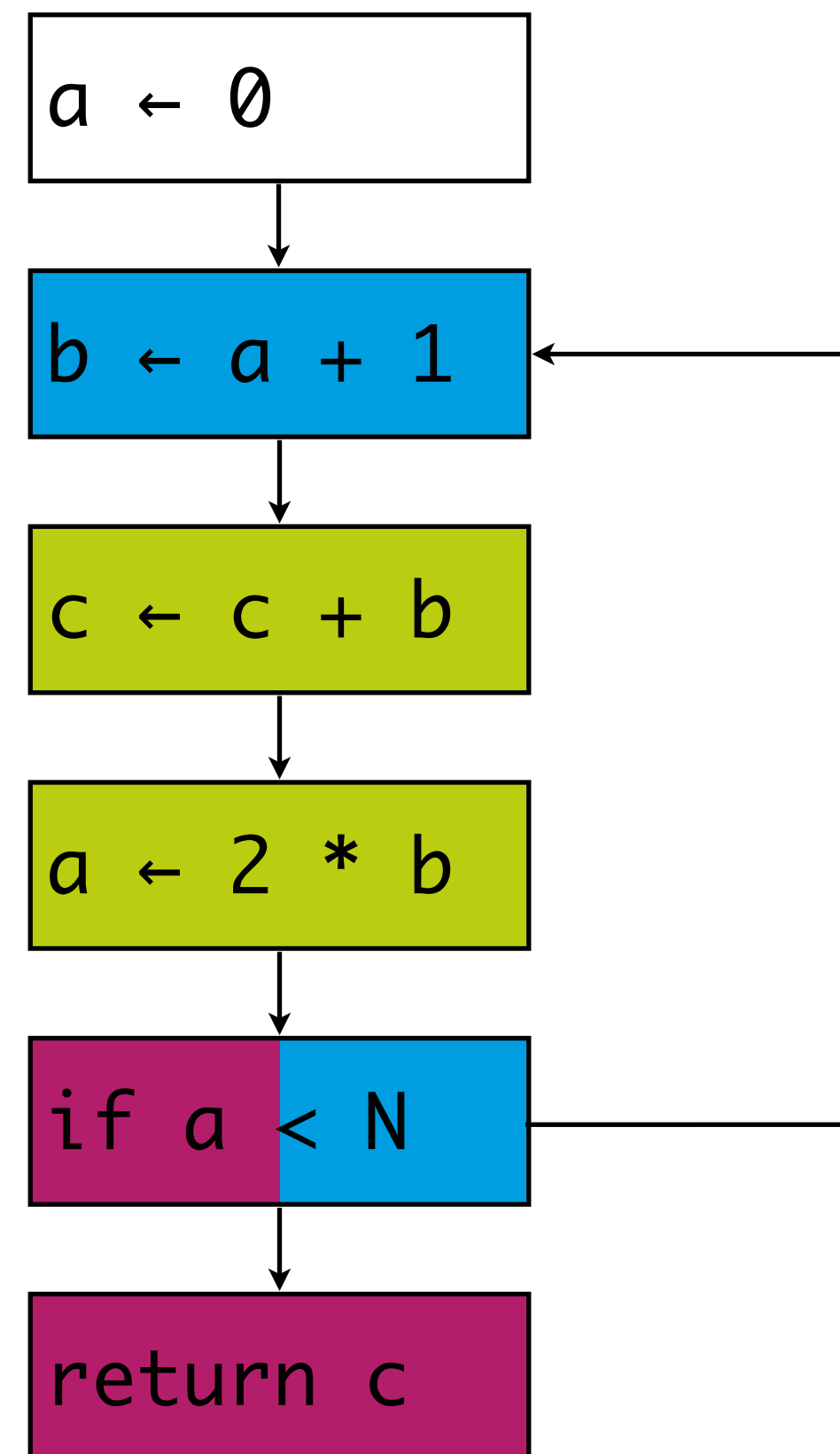
Liveness Analysis



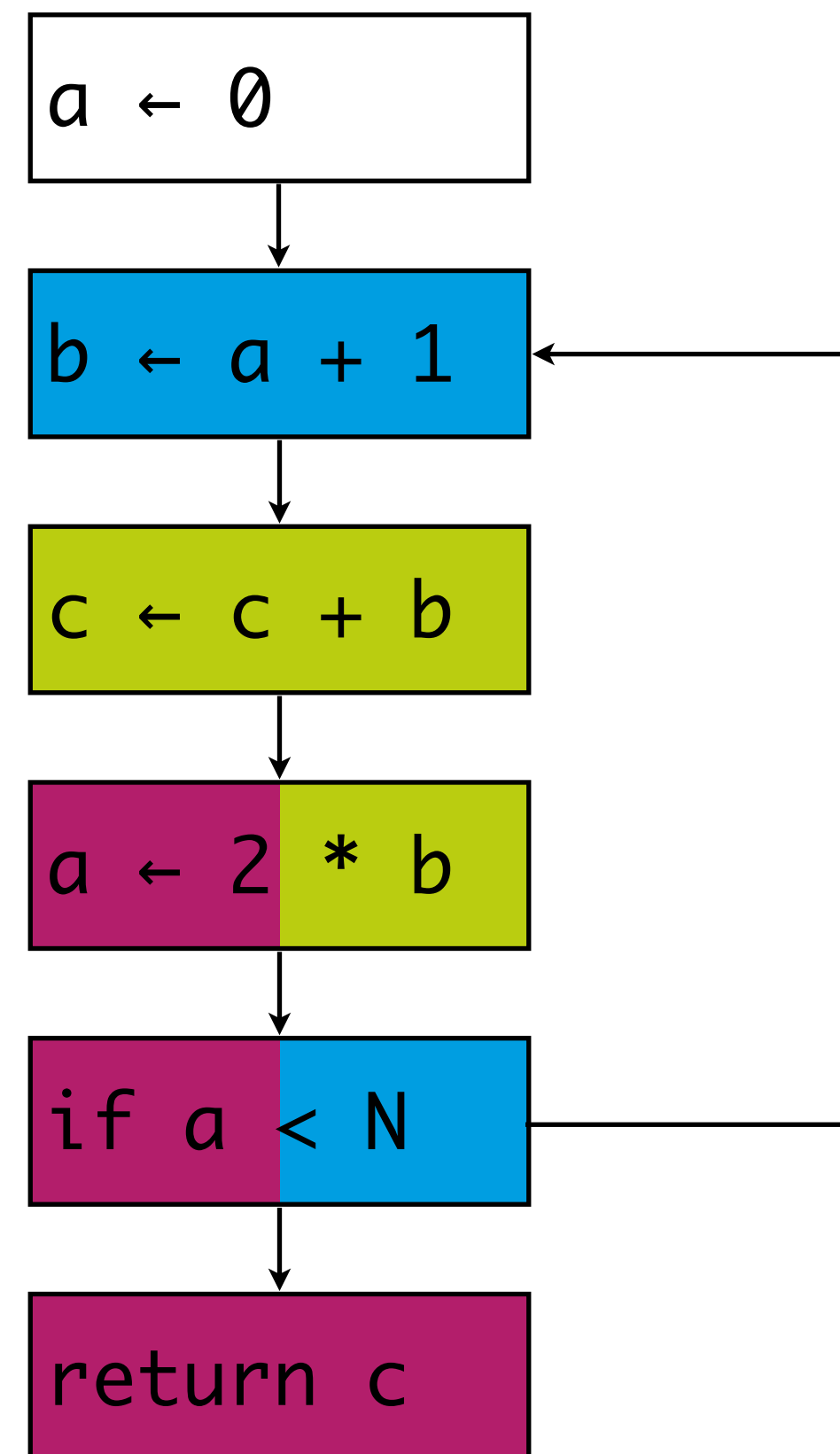
Liveness Analysis



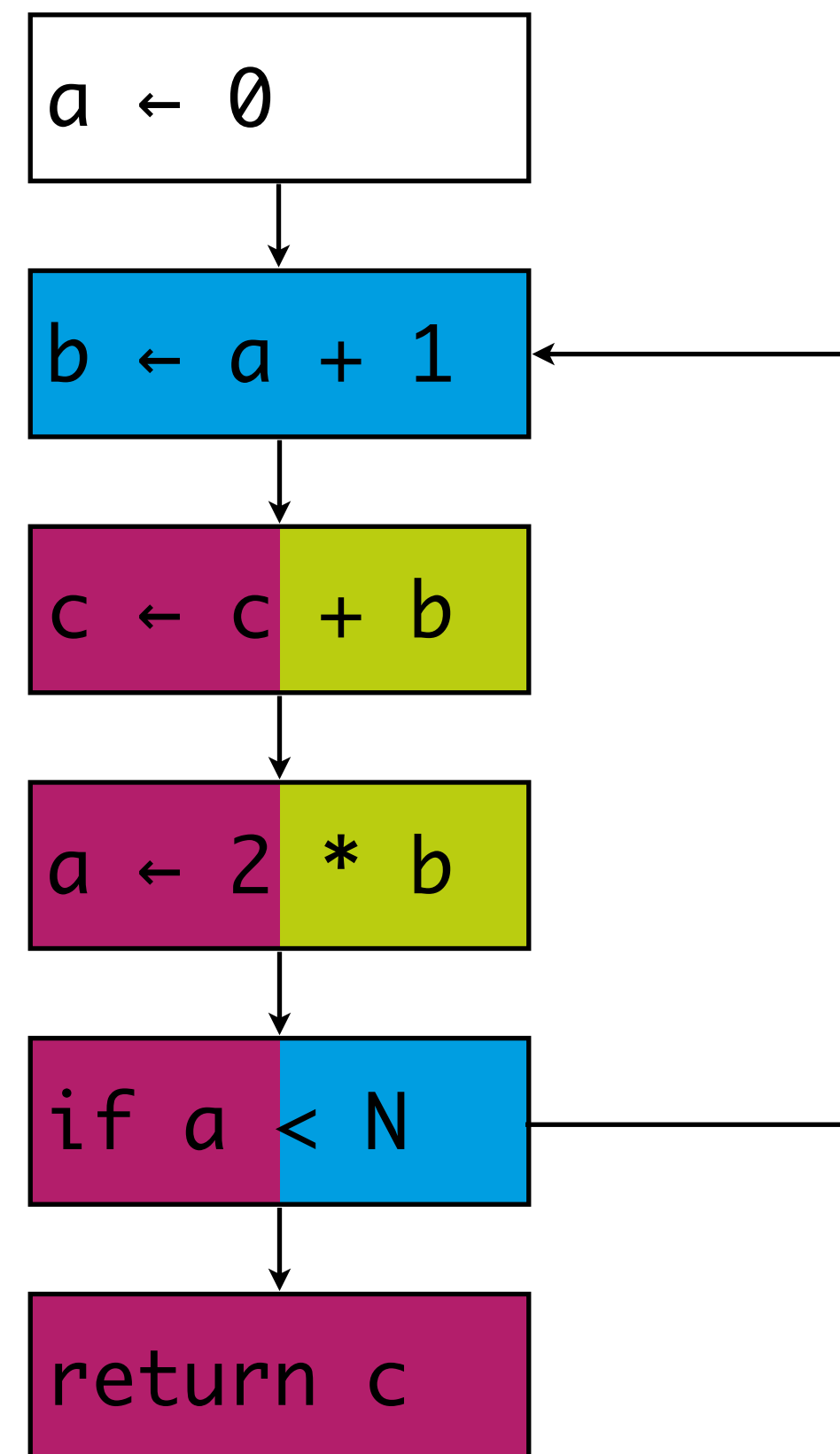
Liveness Analysis



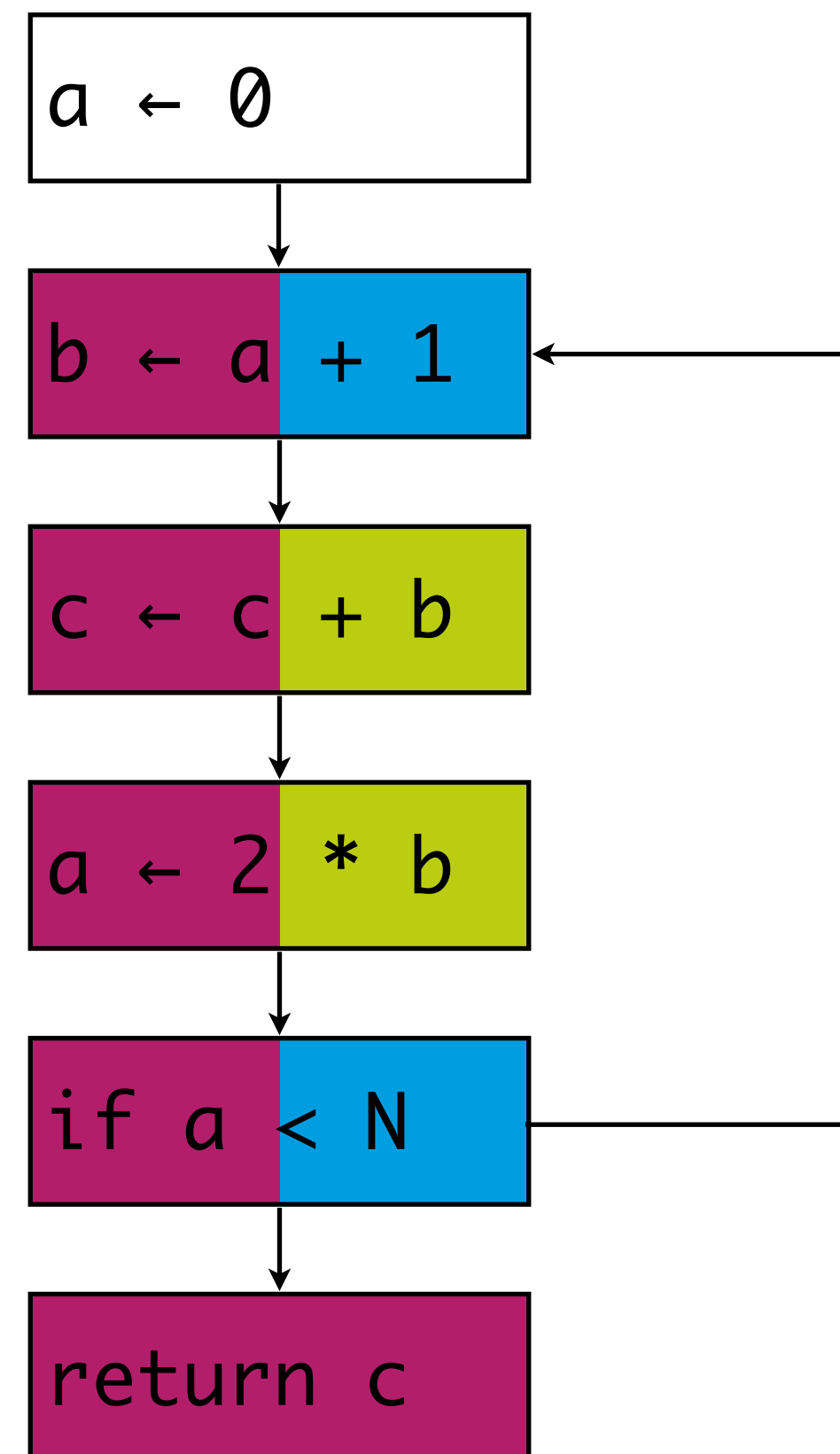
Liveness Analysis



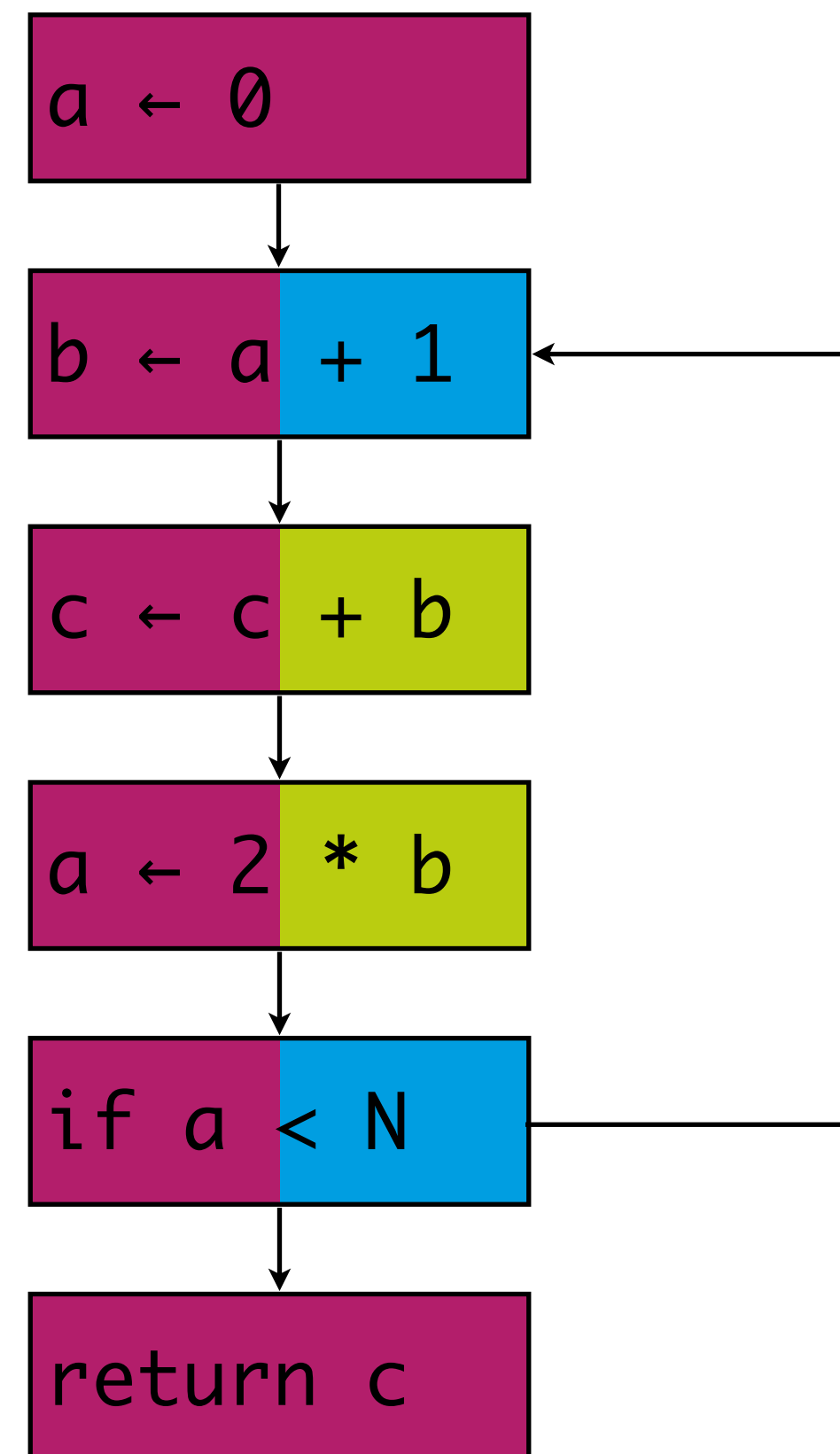
Liveness Analysis



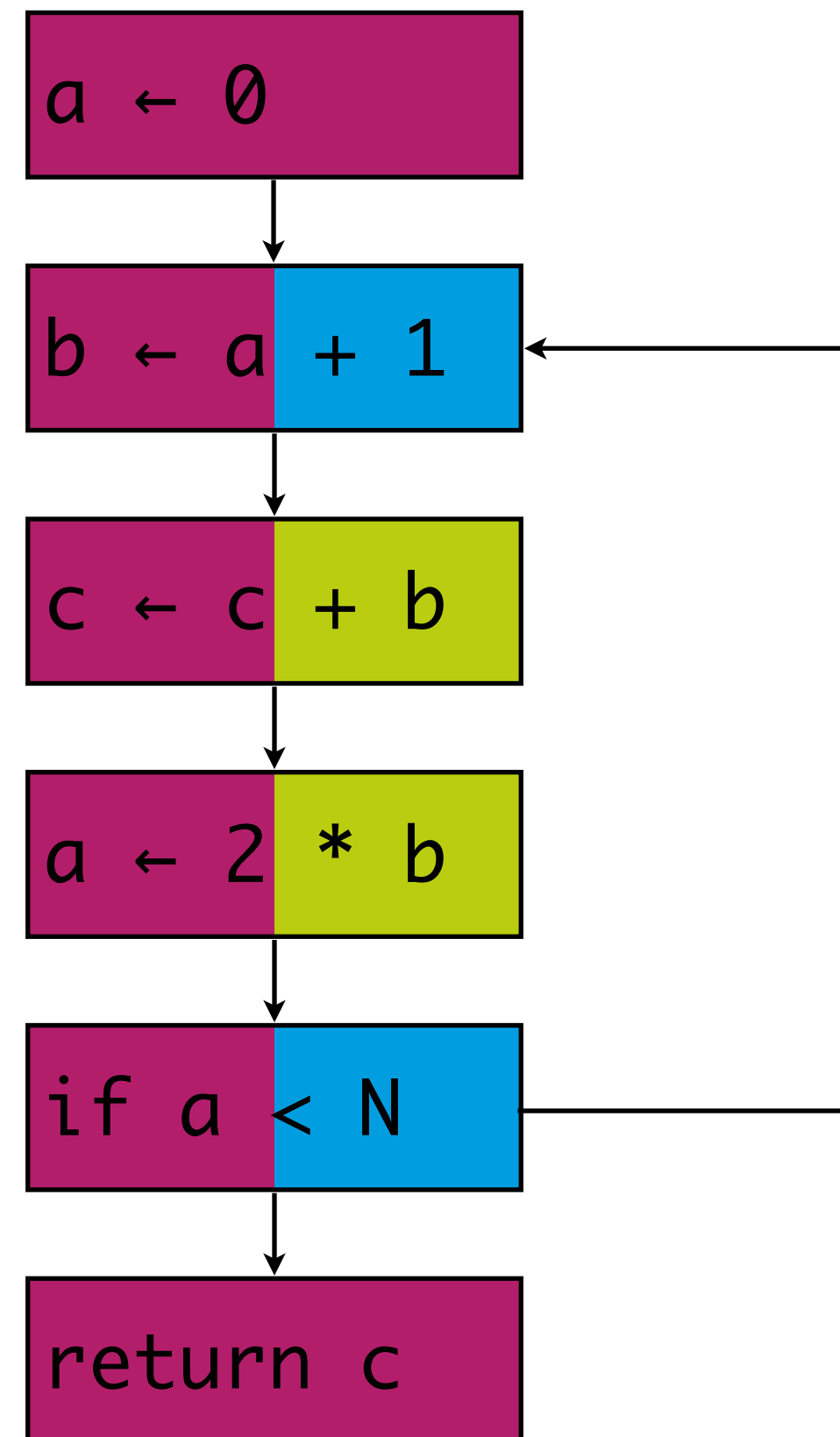
Liveness Analysis



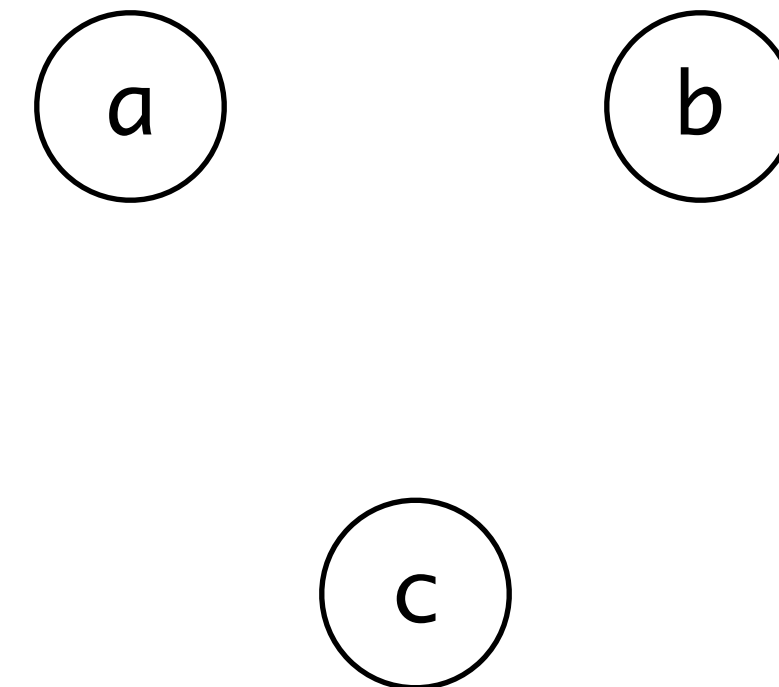
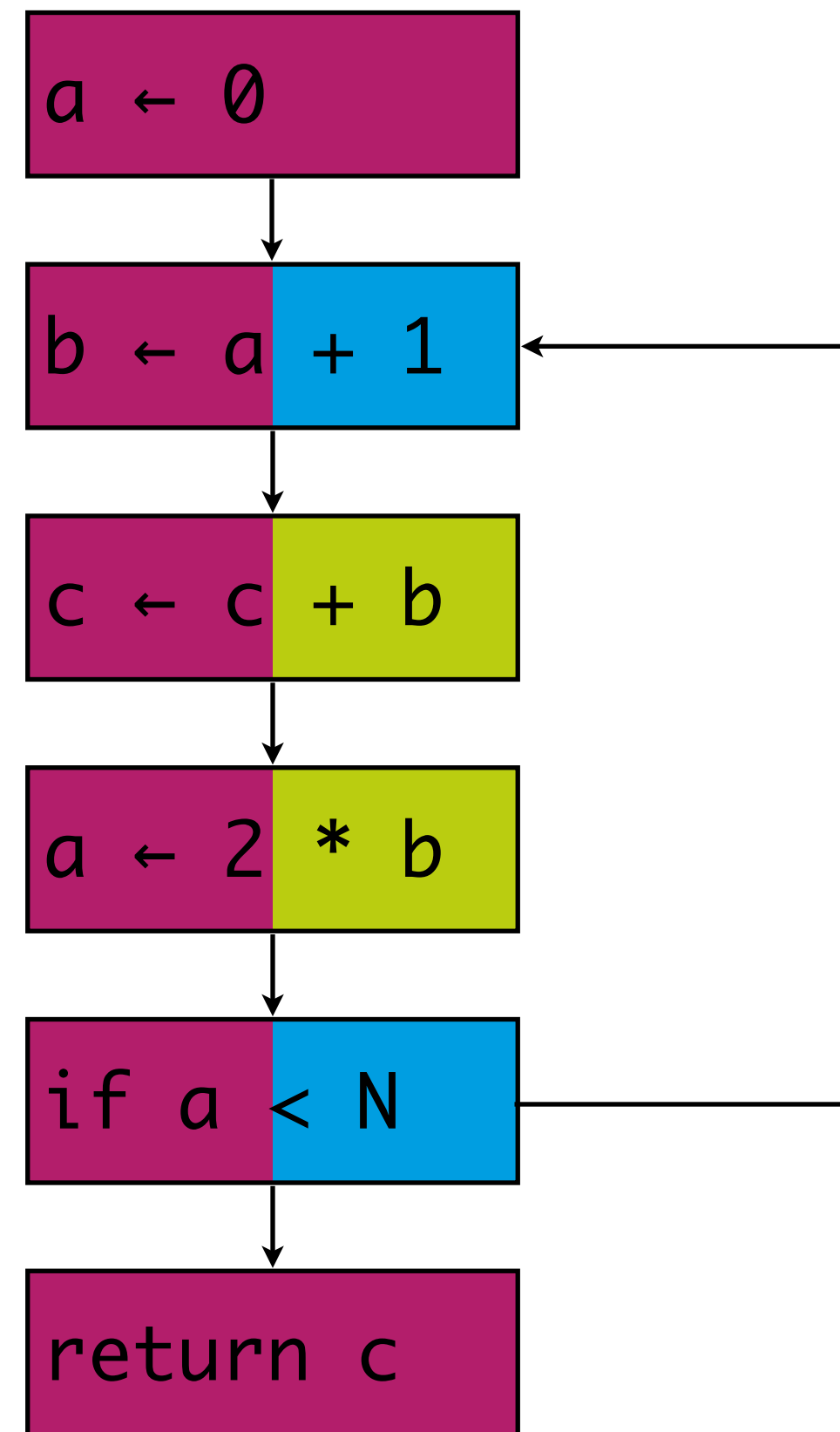
Liveness Analysis



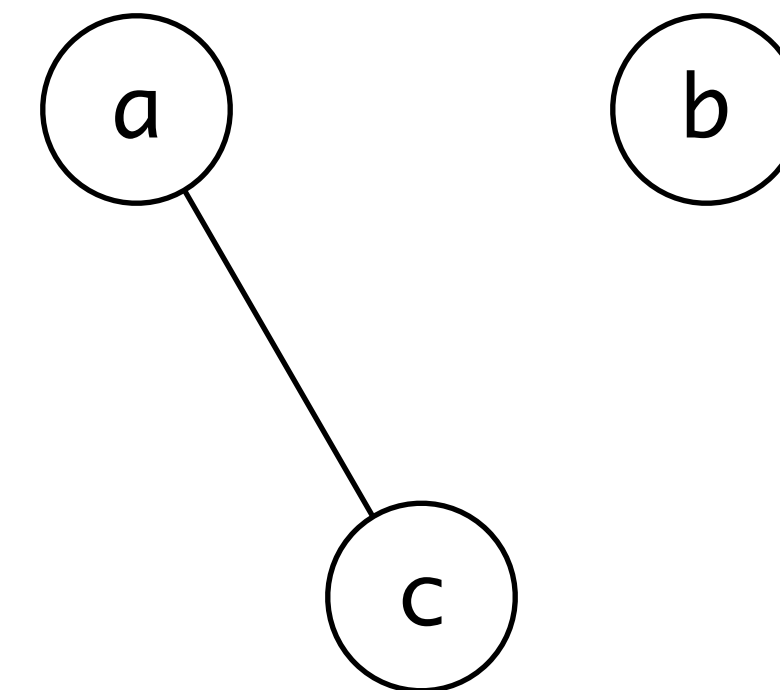
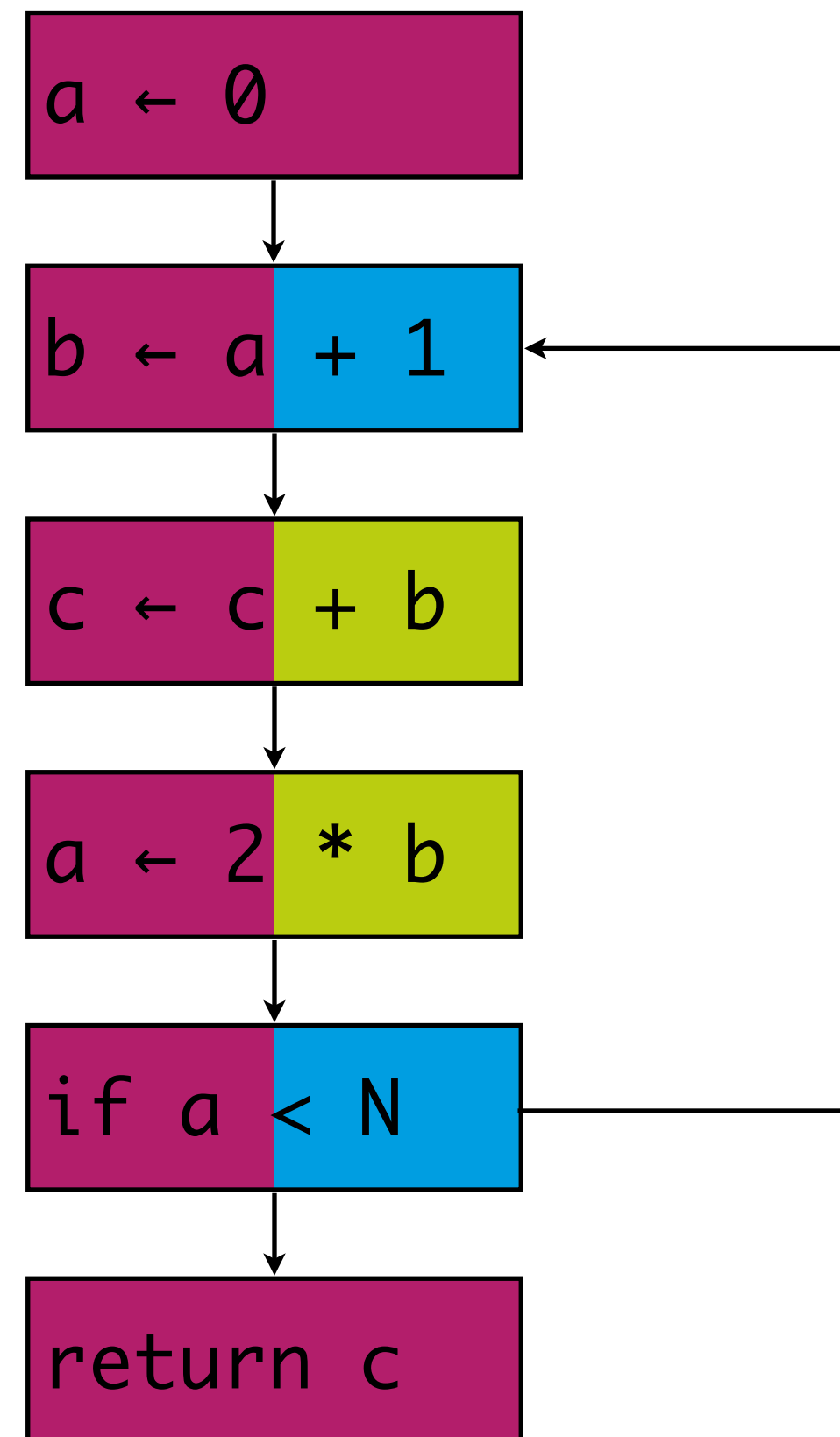
Interference Graphs



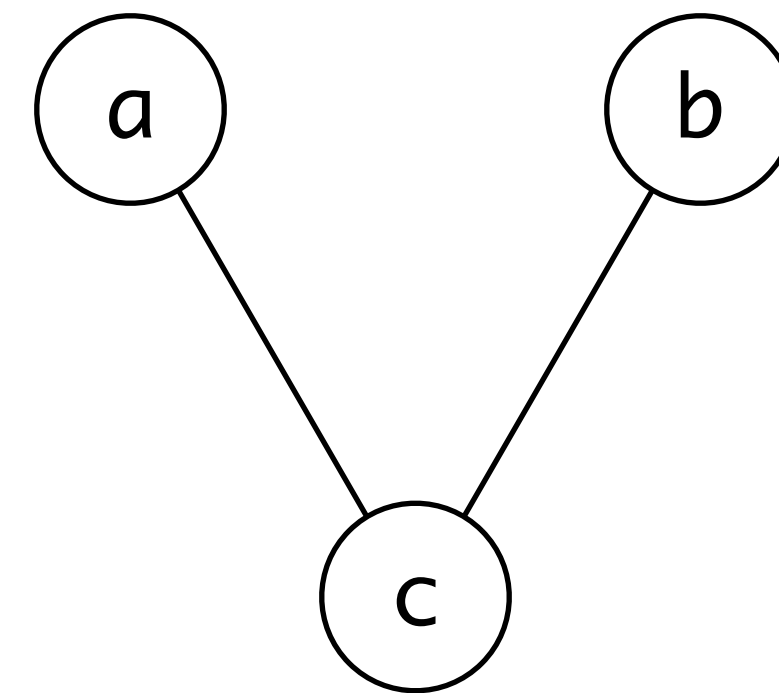
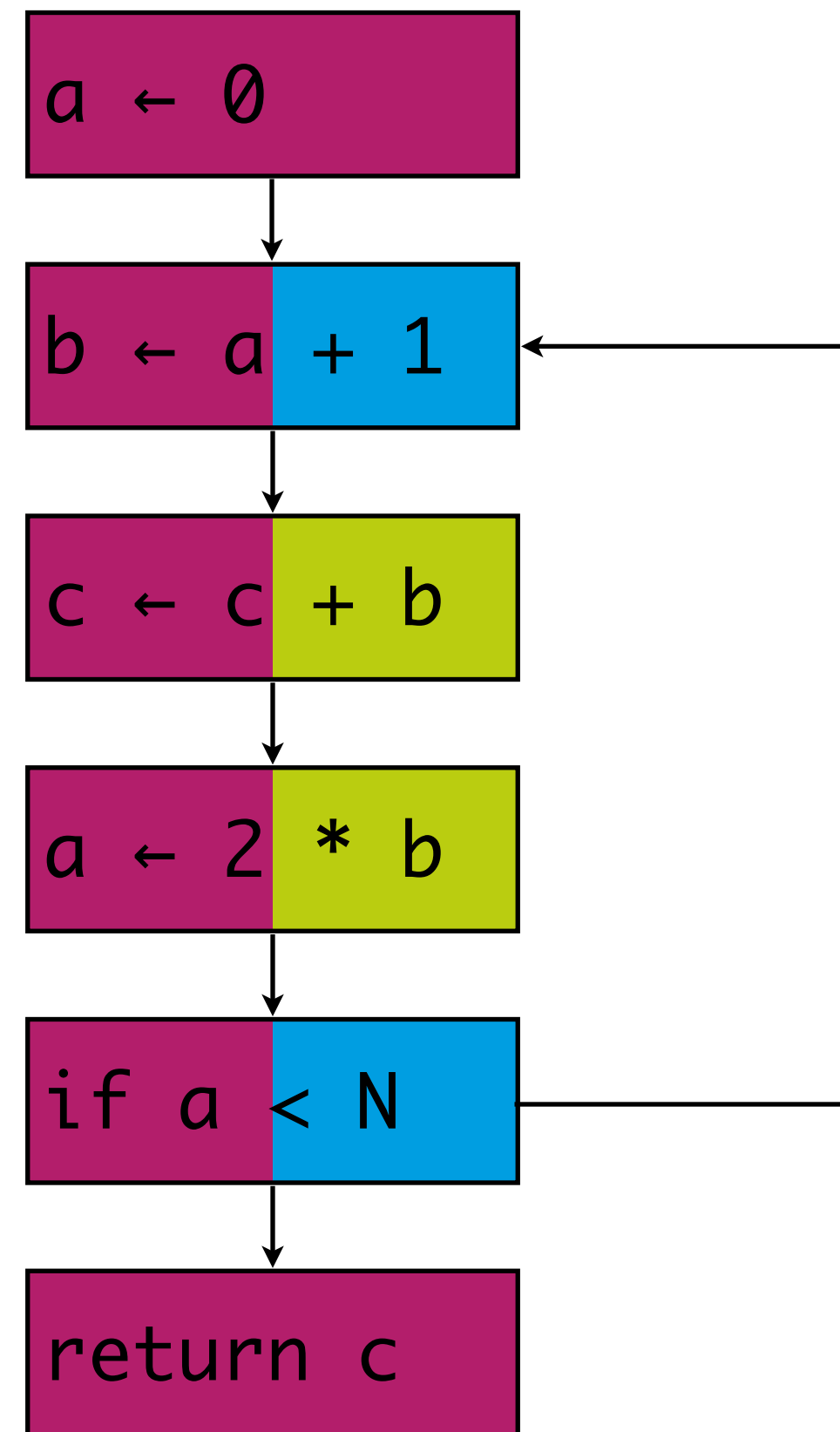
Interference Graphs



Interference Graphs

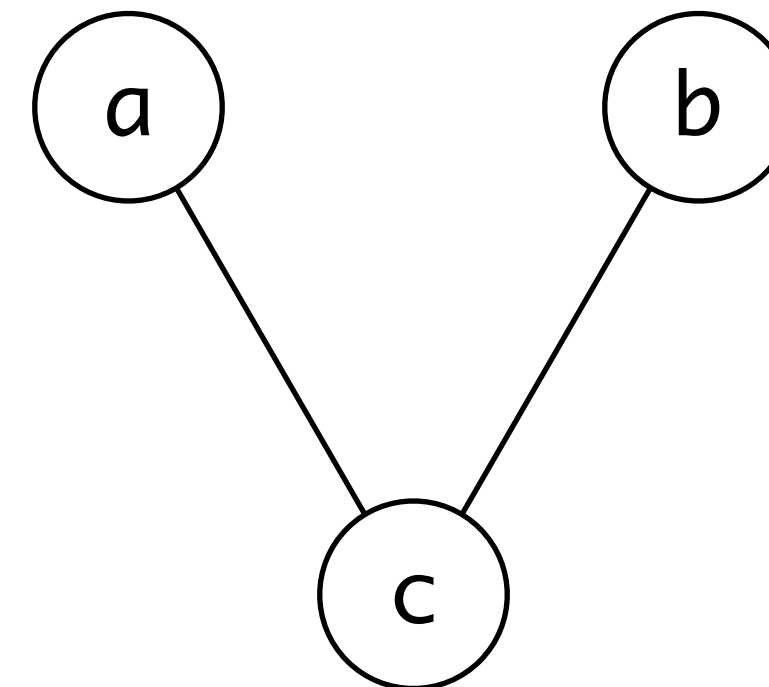
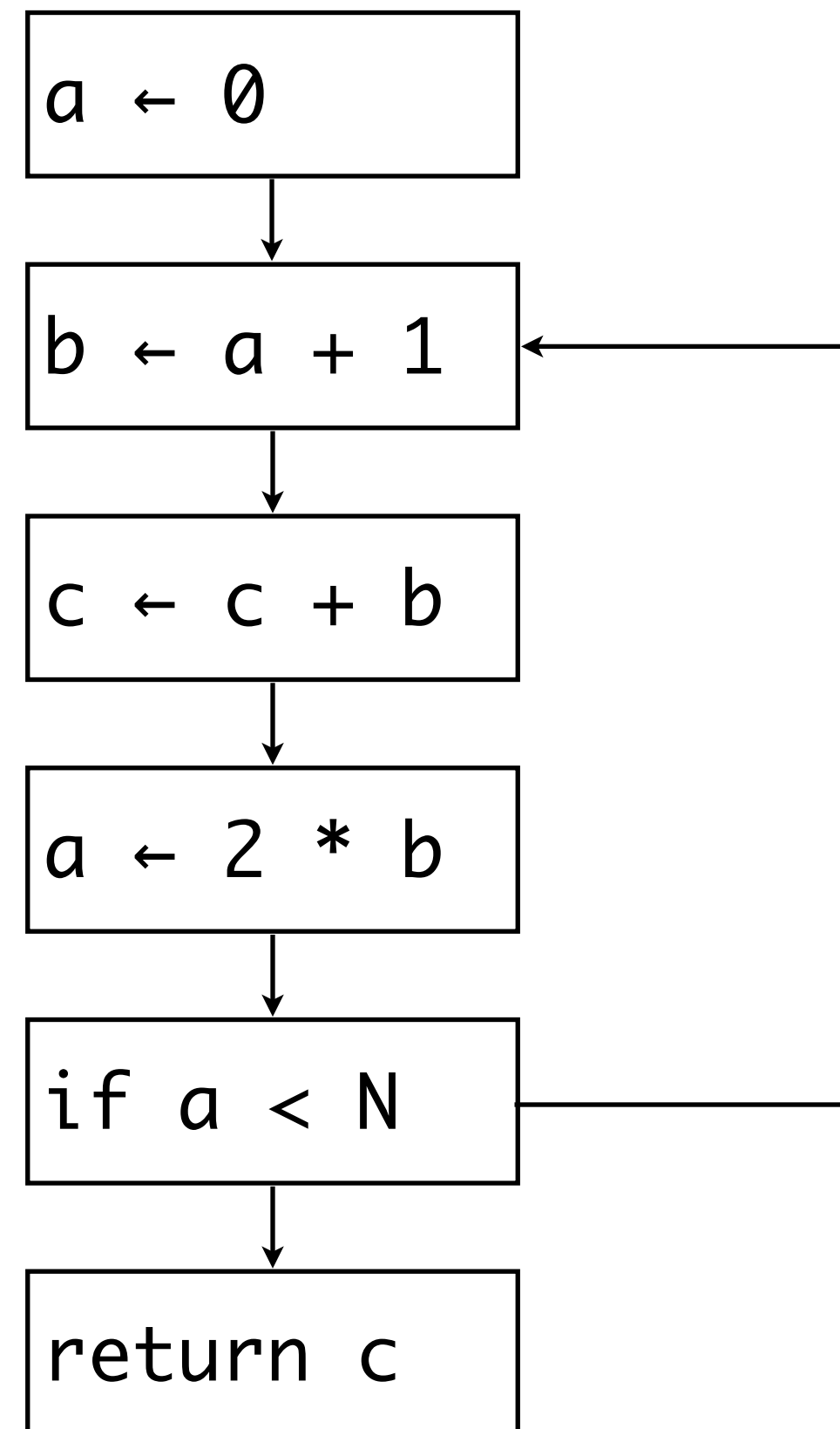


Interference Graphs

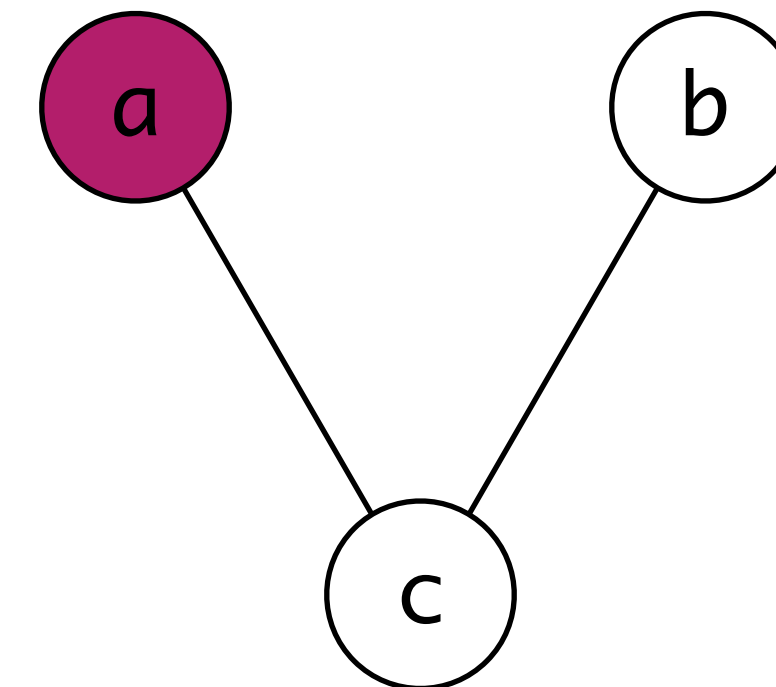
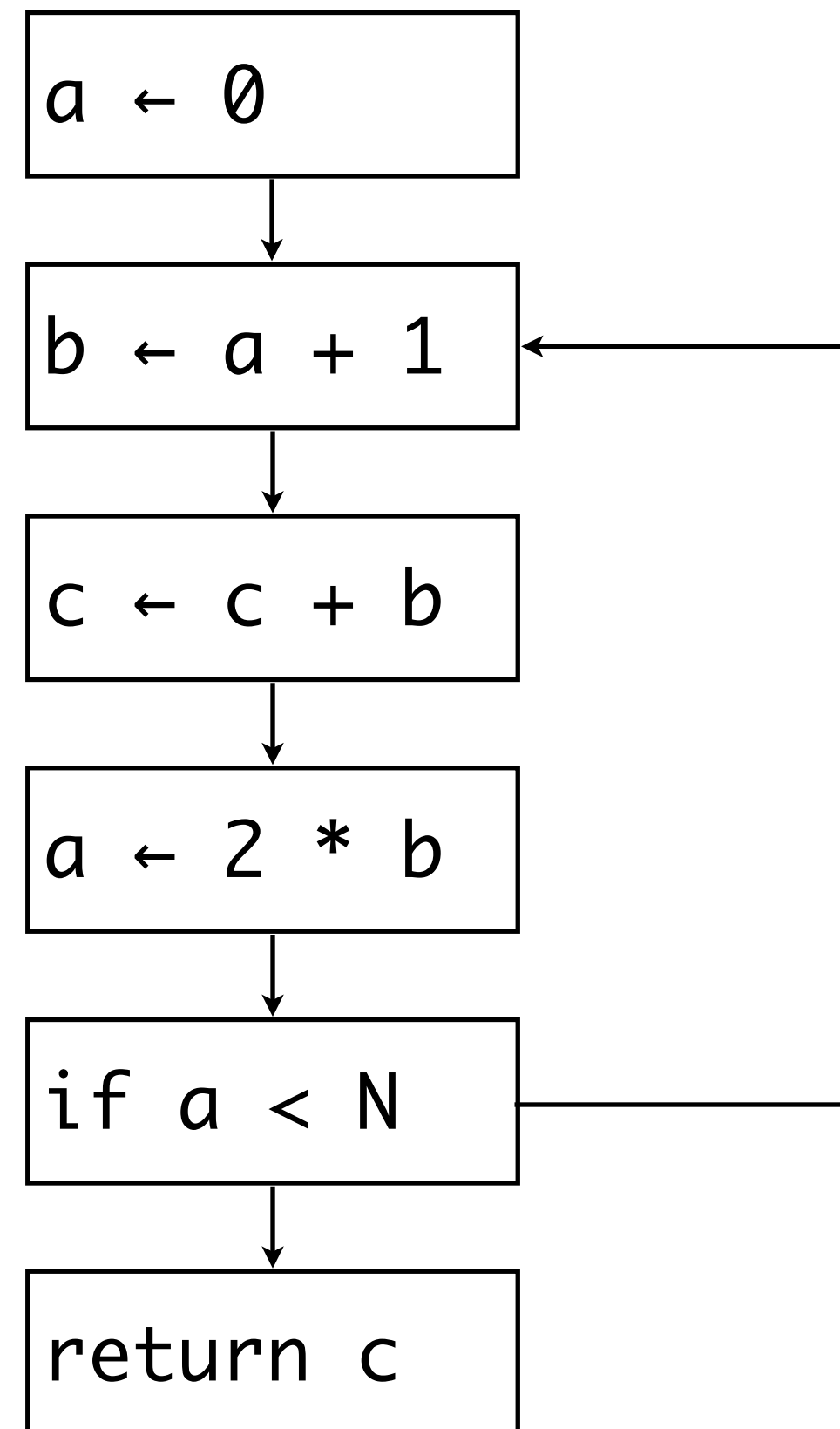


Graph Coloring

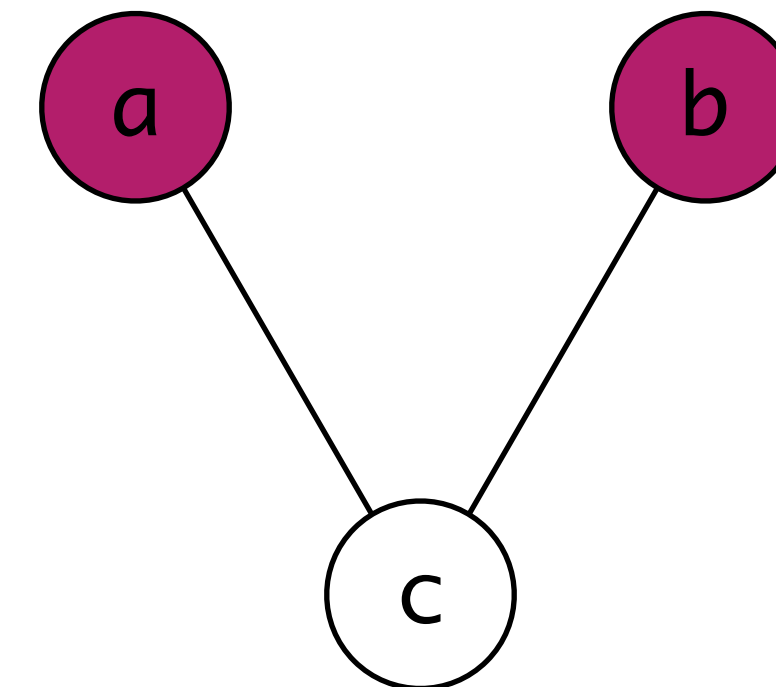
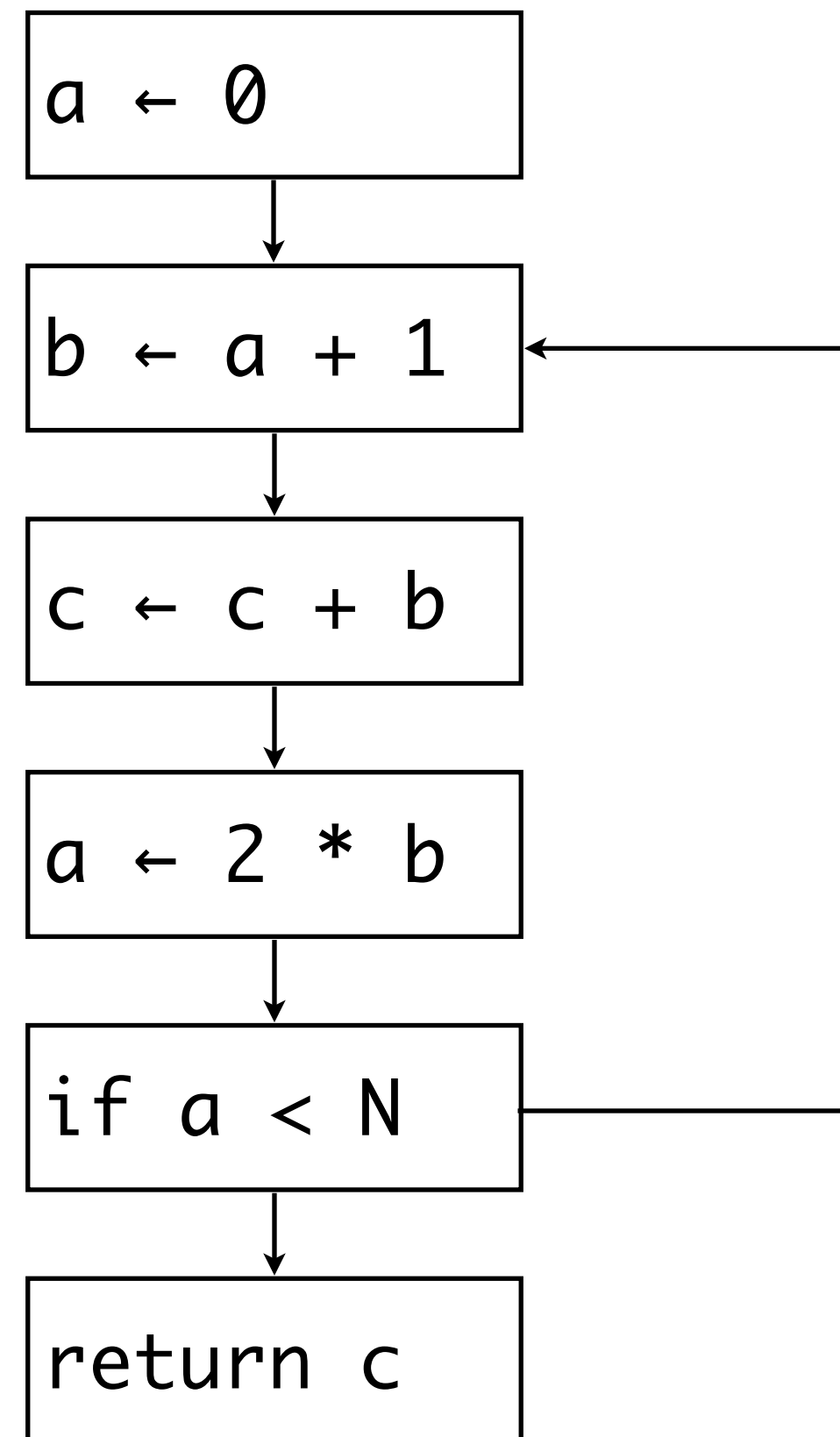
Graph Coloring



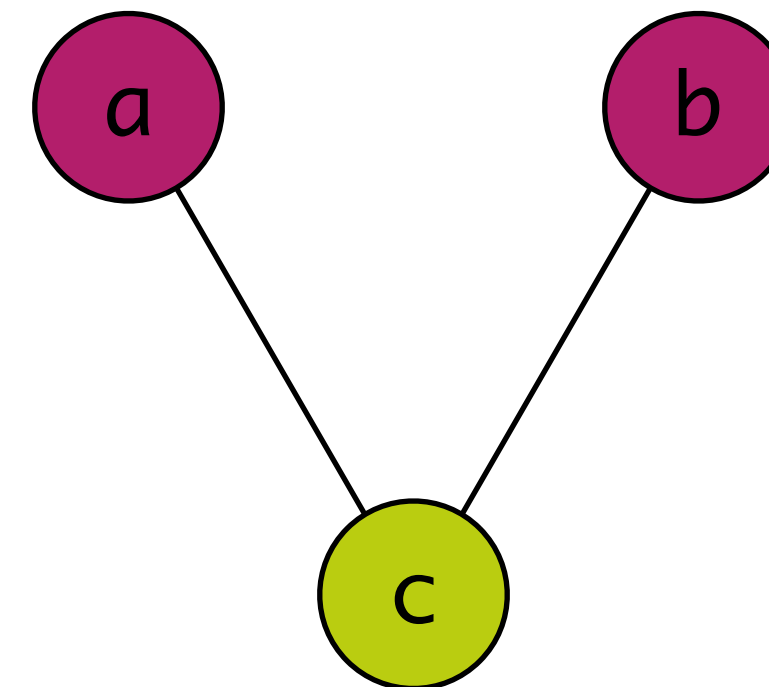
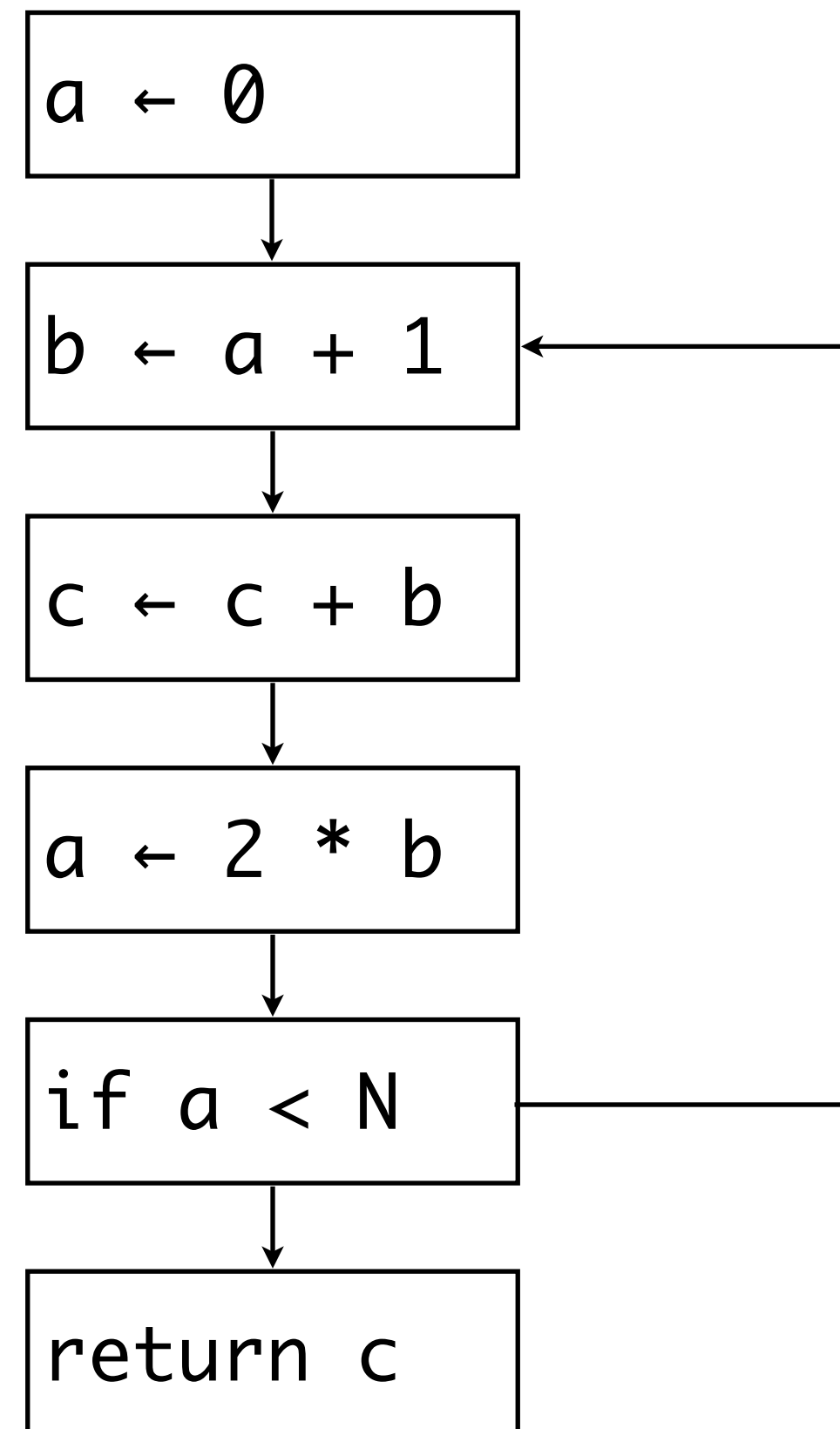
Graph Coloring



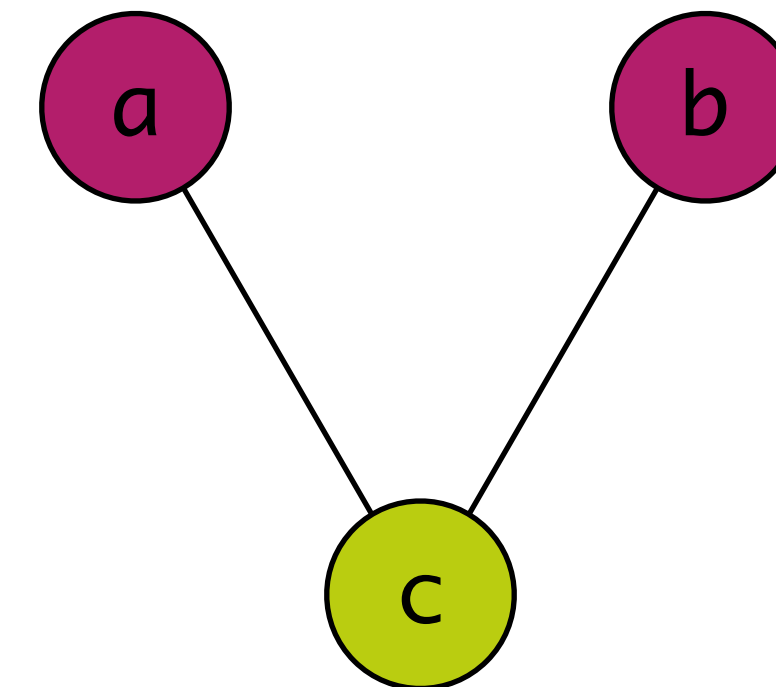
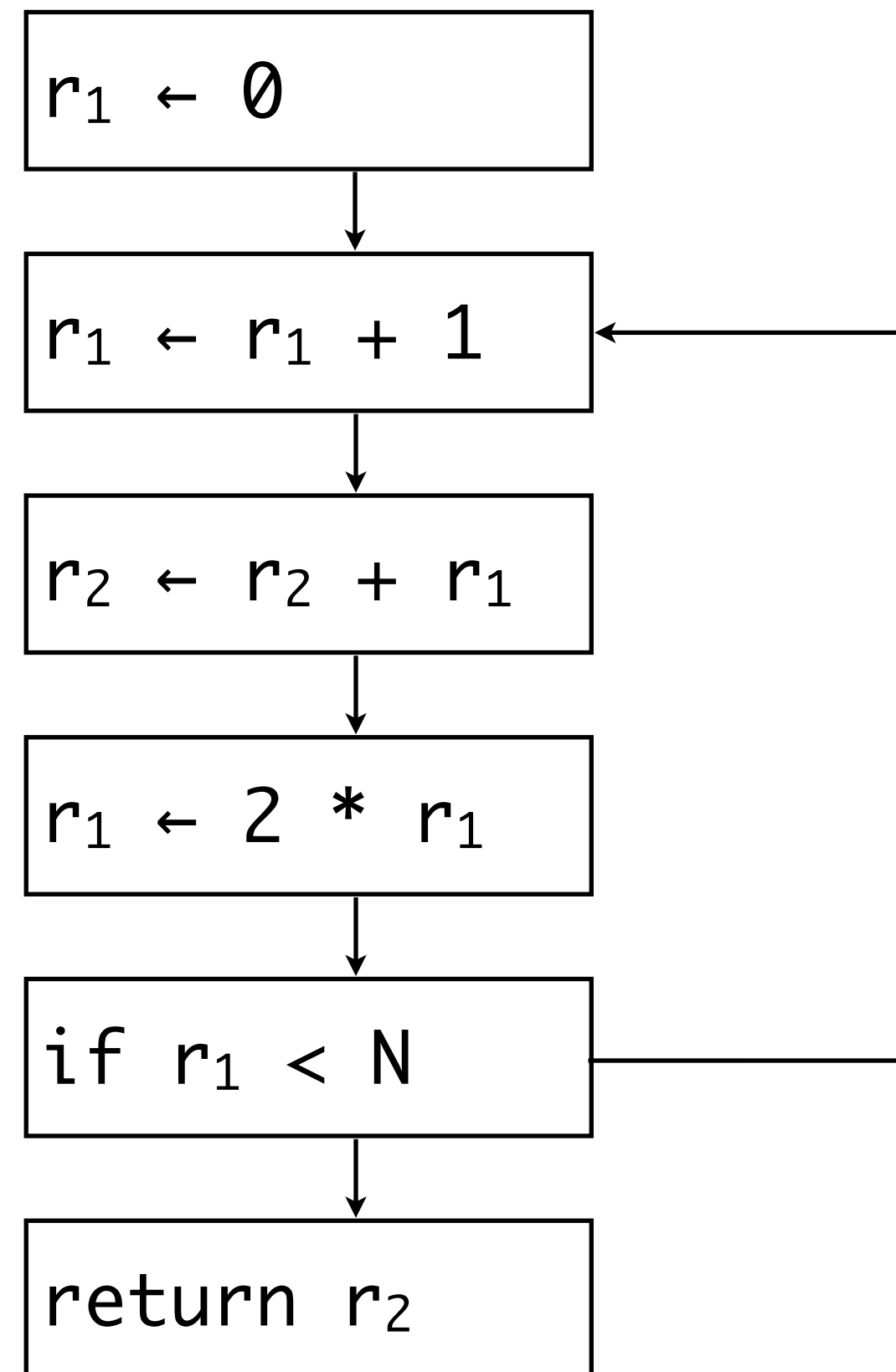
Graph Coloring



Graph Coloring



Graph Coloring



Graph Coloring: Steps

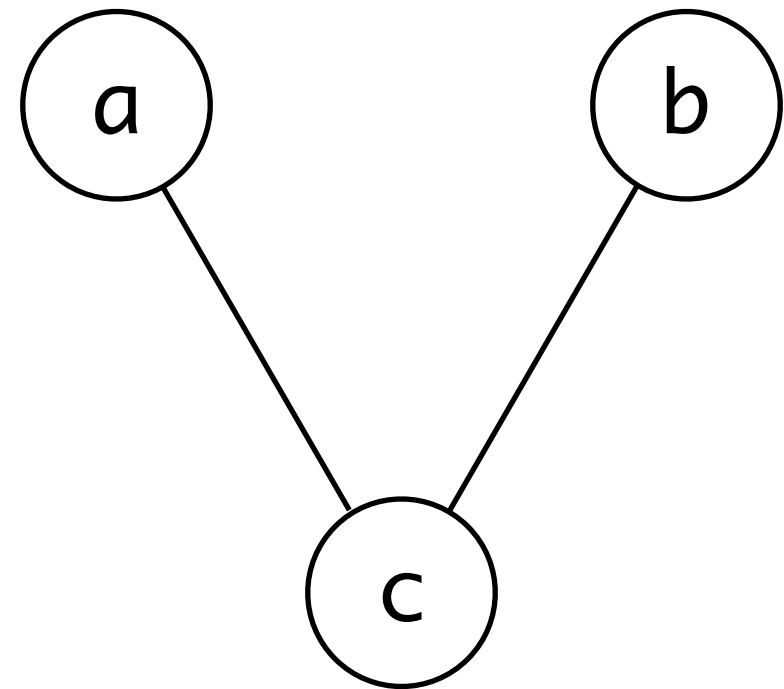
Simplify

- remove node of insignificant degree (fewer than k edges)

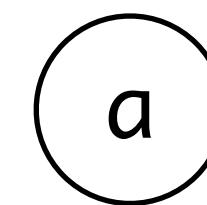
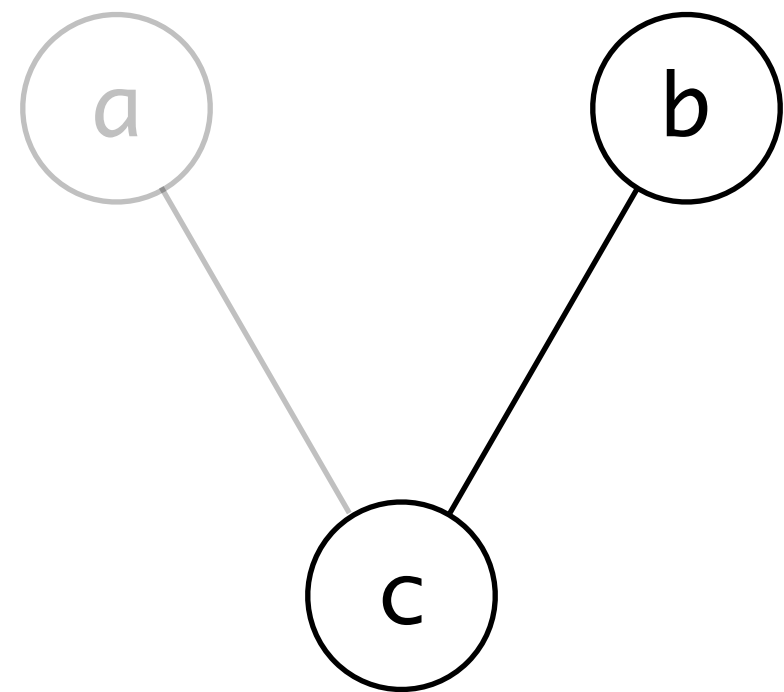
Select

- add node, select color

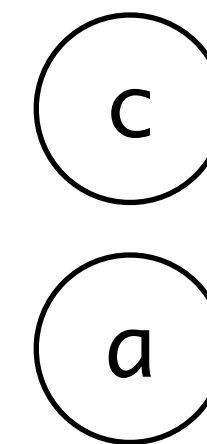
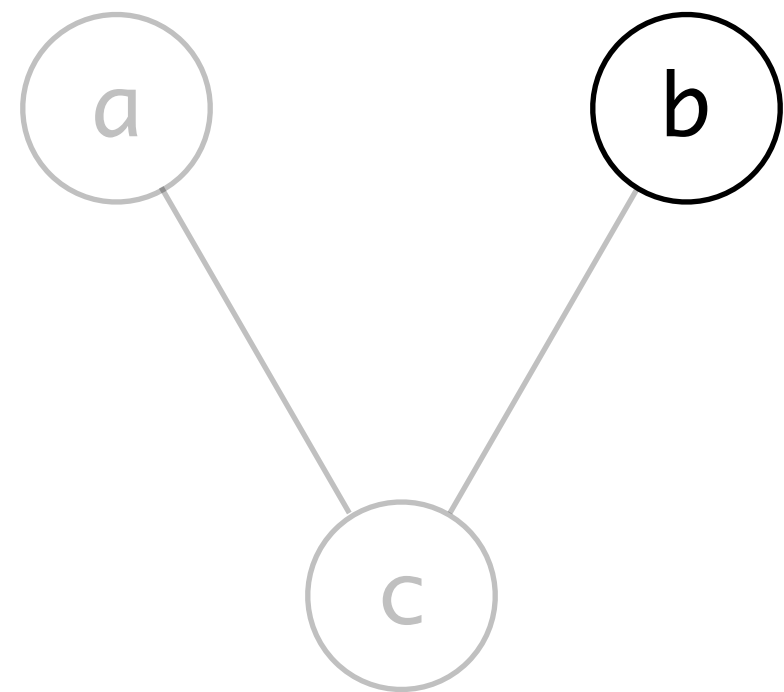
Graph Coloring: Example with 2 Colors



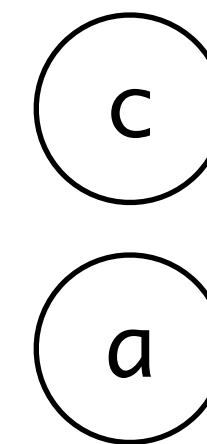
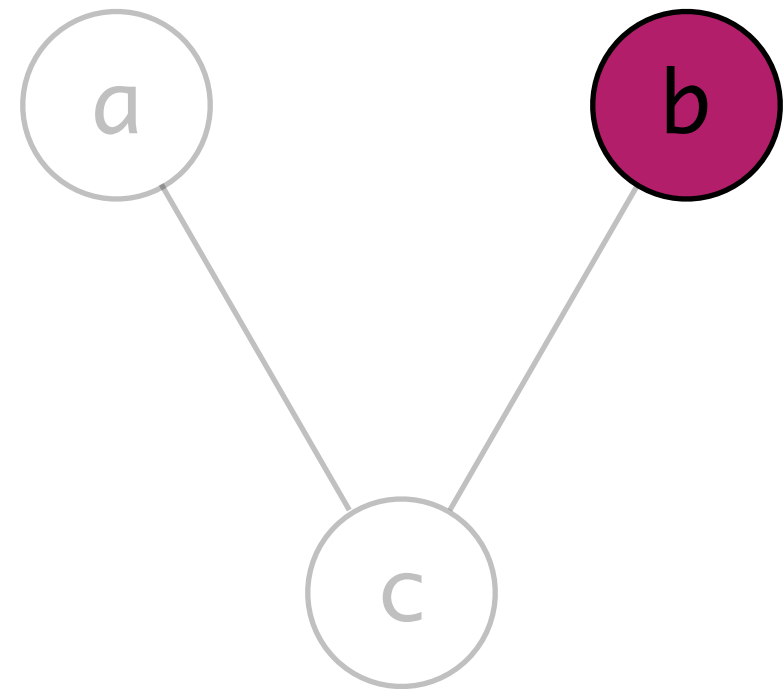
Graph Coloring: Example with 2 Colors



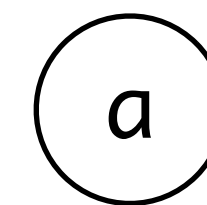
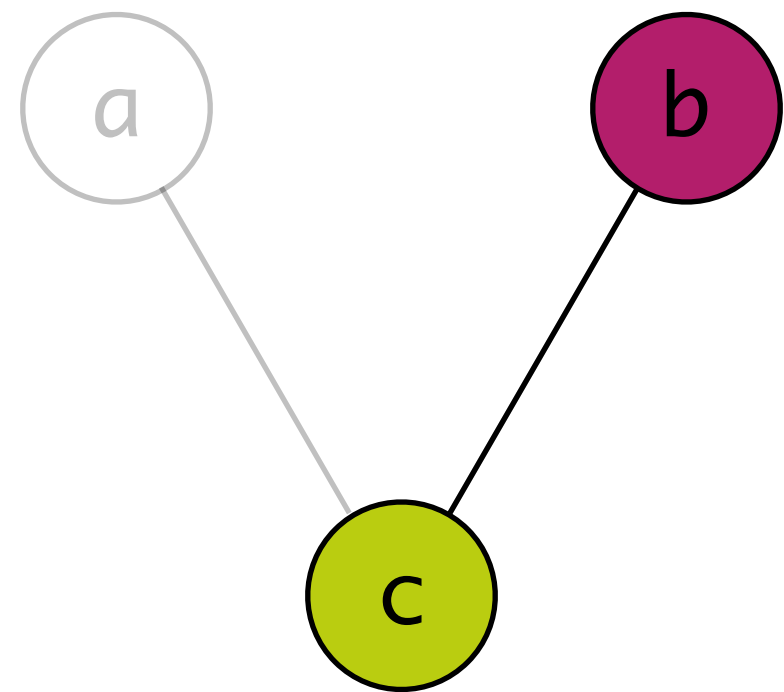
Graph Coloring: Example with 2 Colors



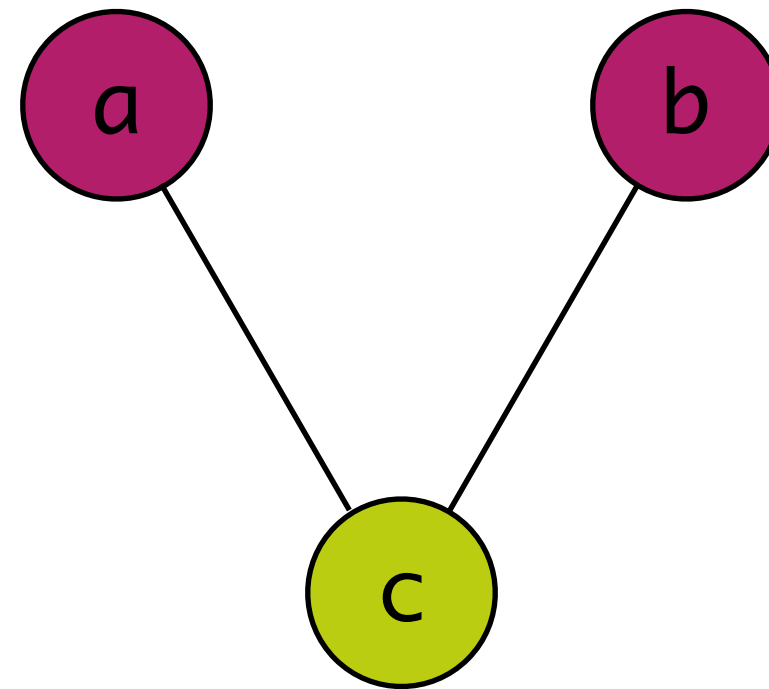
Graph Coloring: Example with 2 Colors



Graph Coloring: Example with 2 Colors



Graph Coloring: Example with 2 Colors

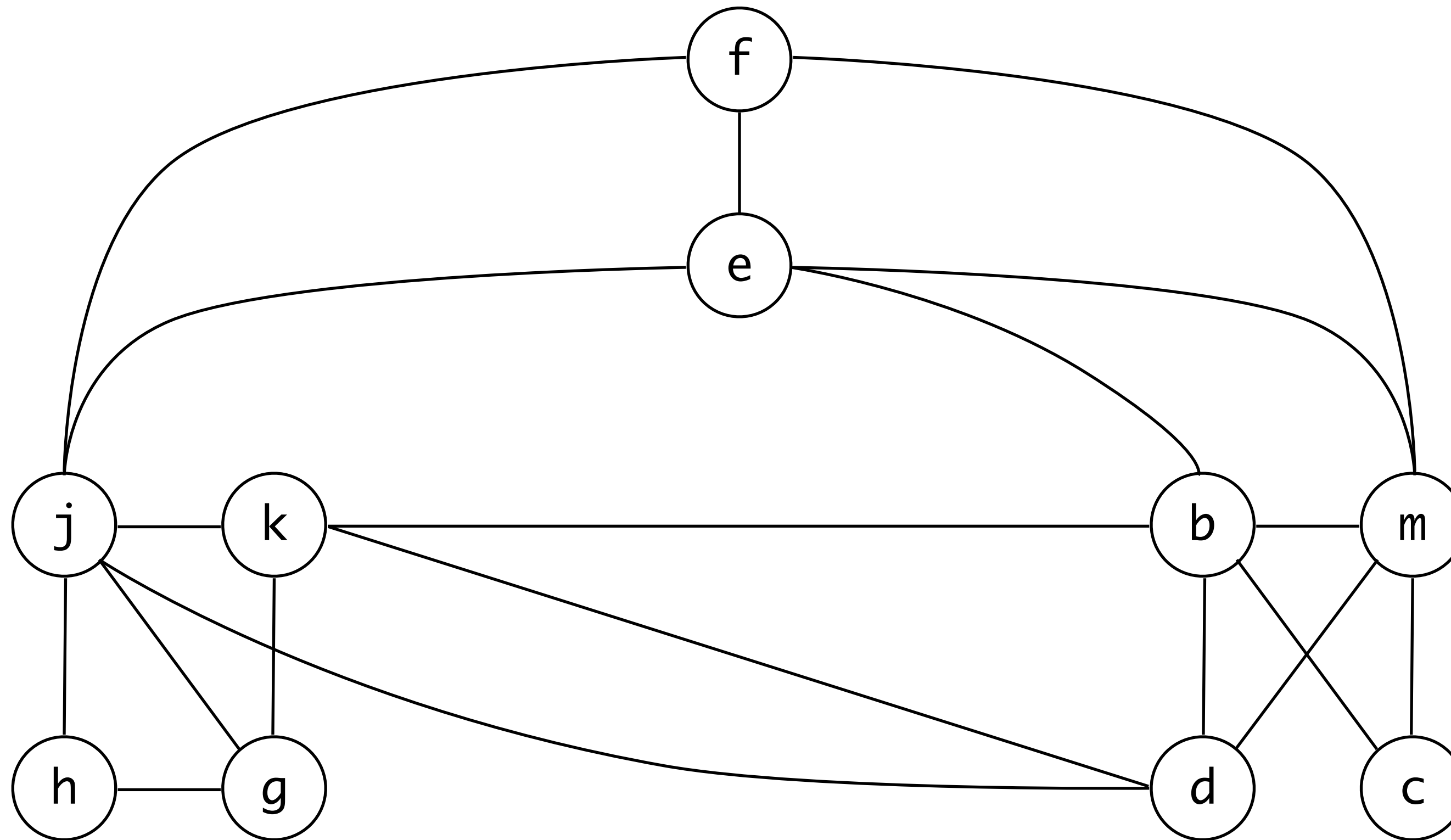


Graph Coloring

```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Graph Coloring: Example with 4 Colors

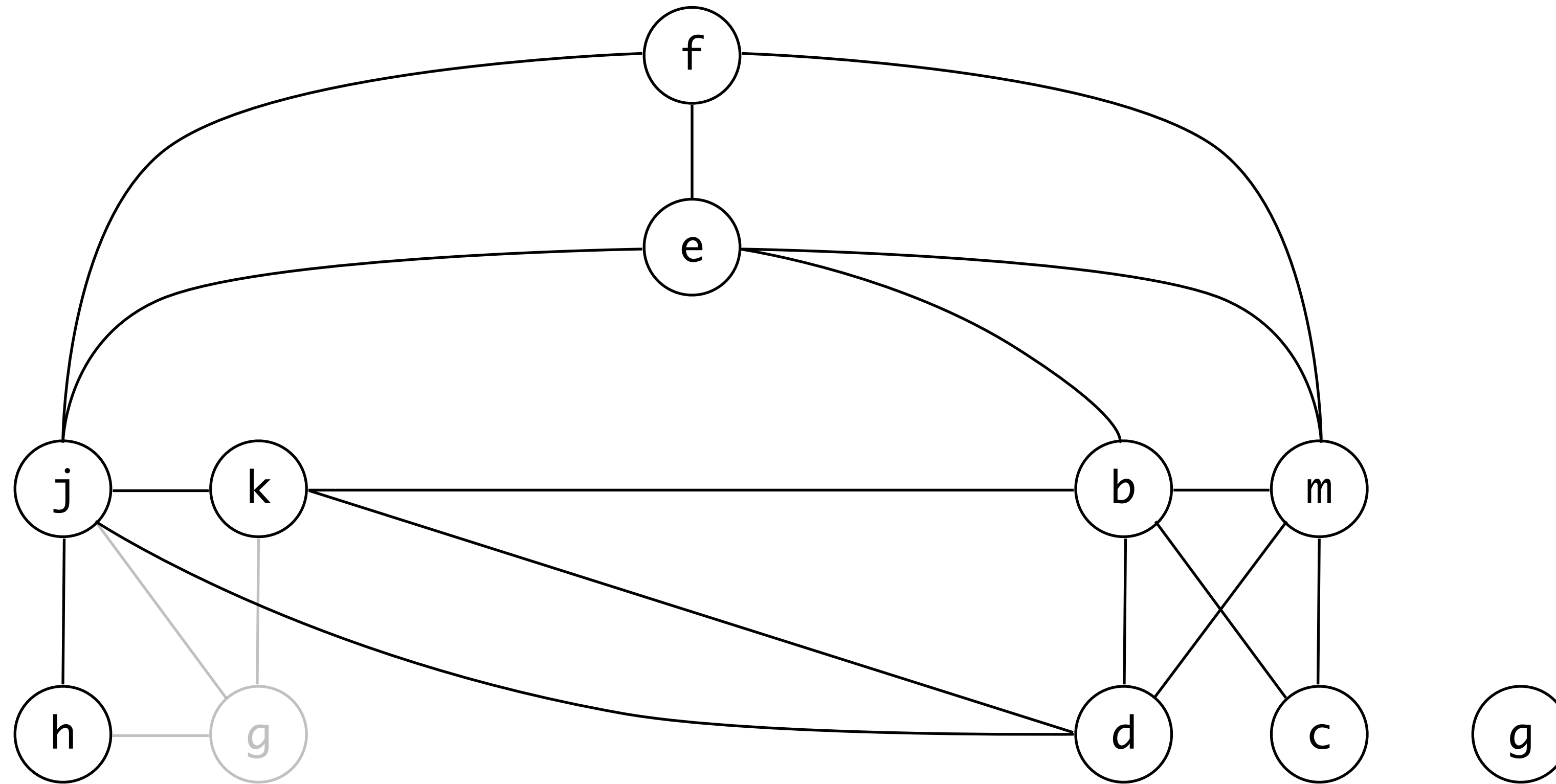
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

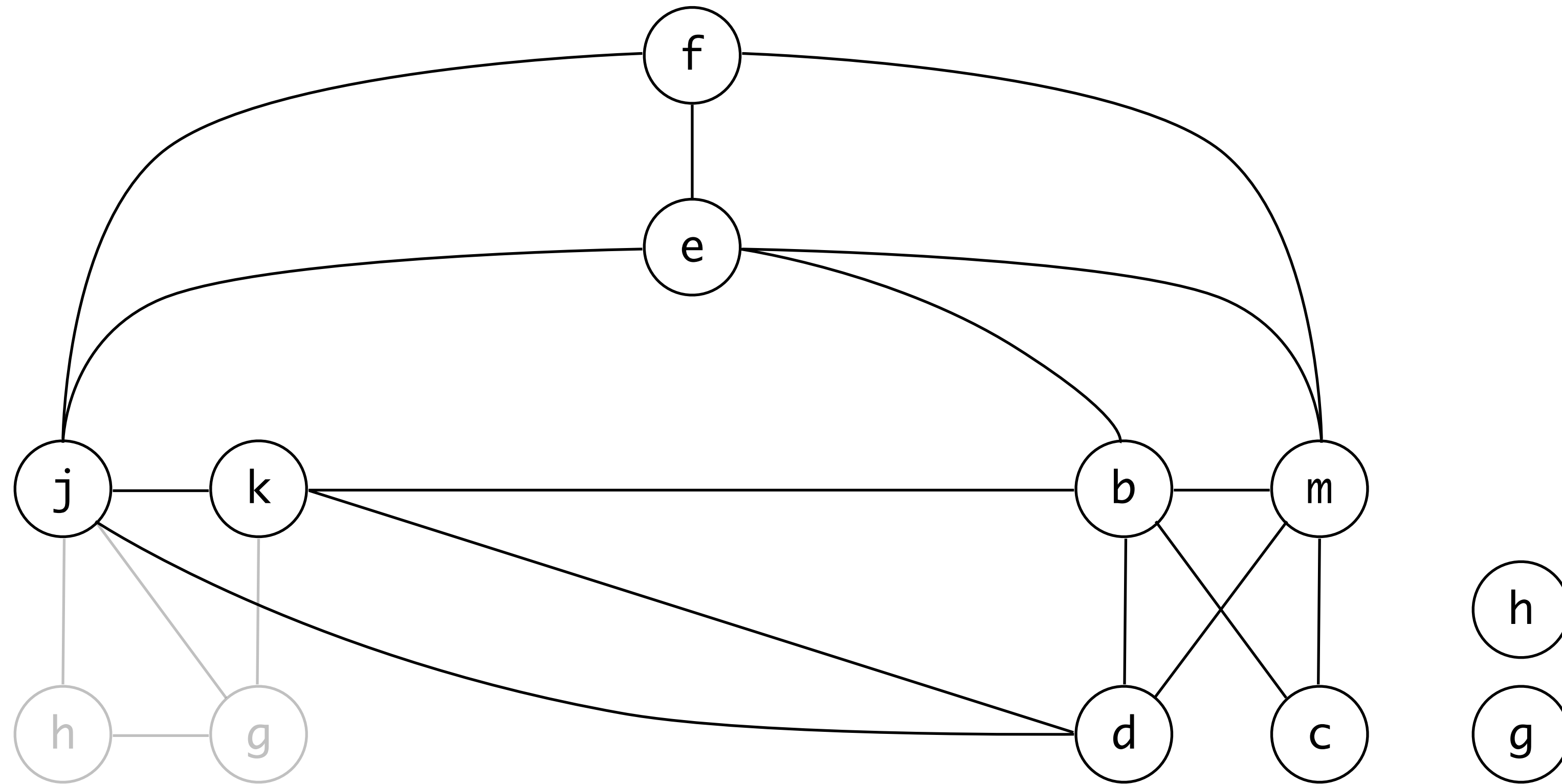
Graph Coloring: Example with 4 Colors

r_1
 r_2
 r_3
 r_4



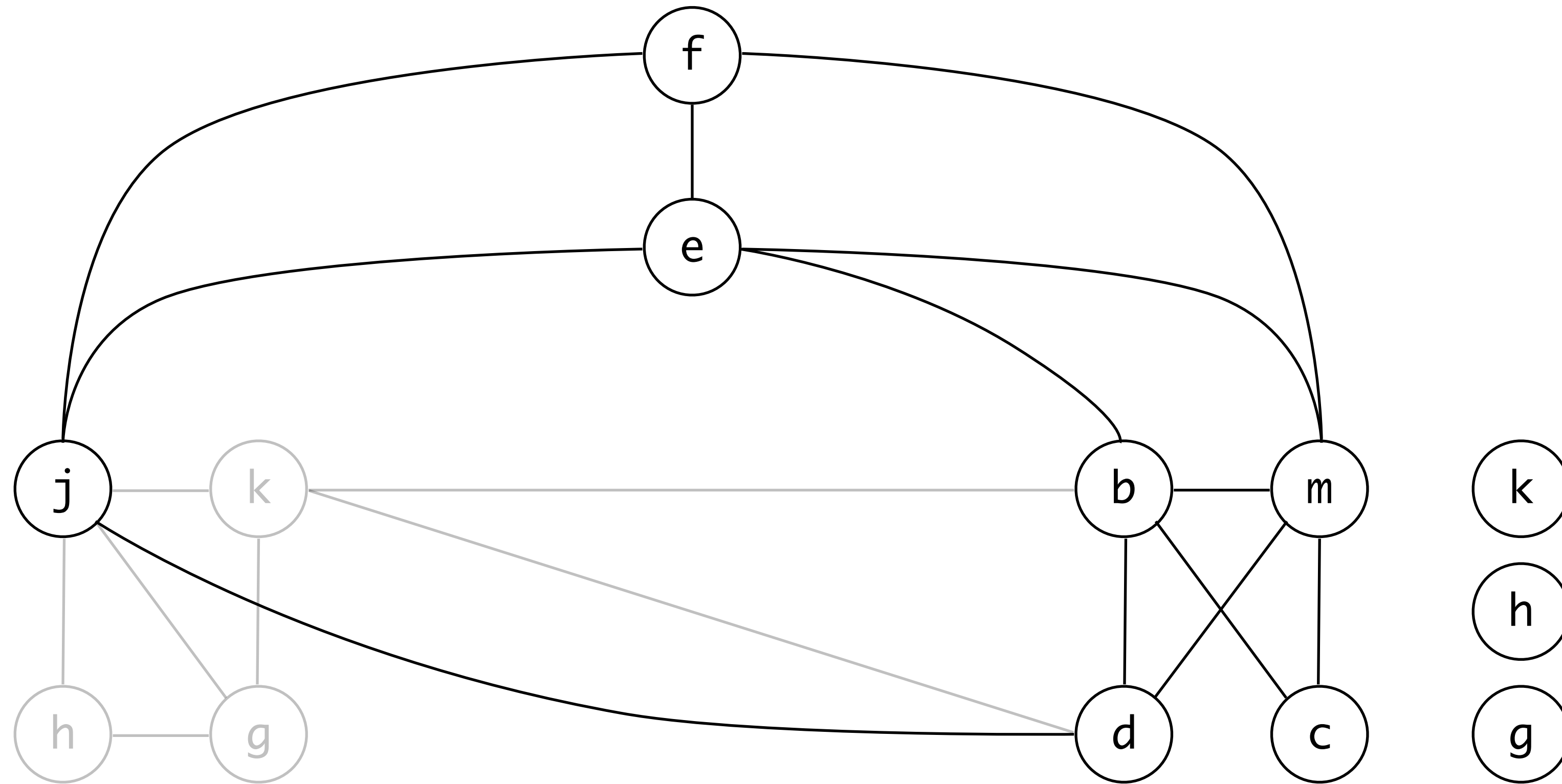
Graph Coloring: Example with 4 Colors

r_1
 r_2
 r_3
 r_4



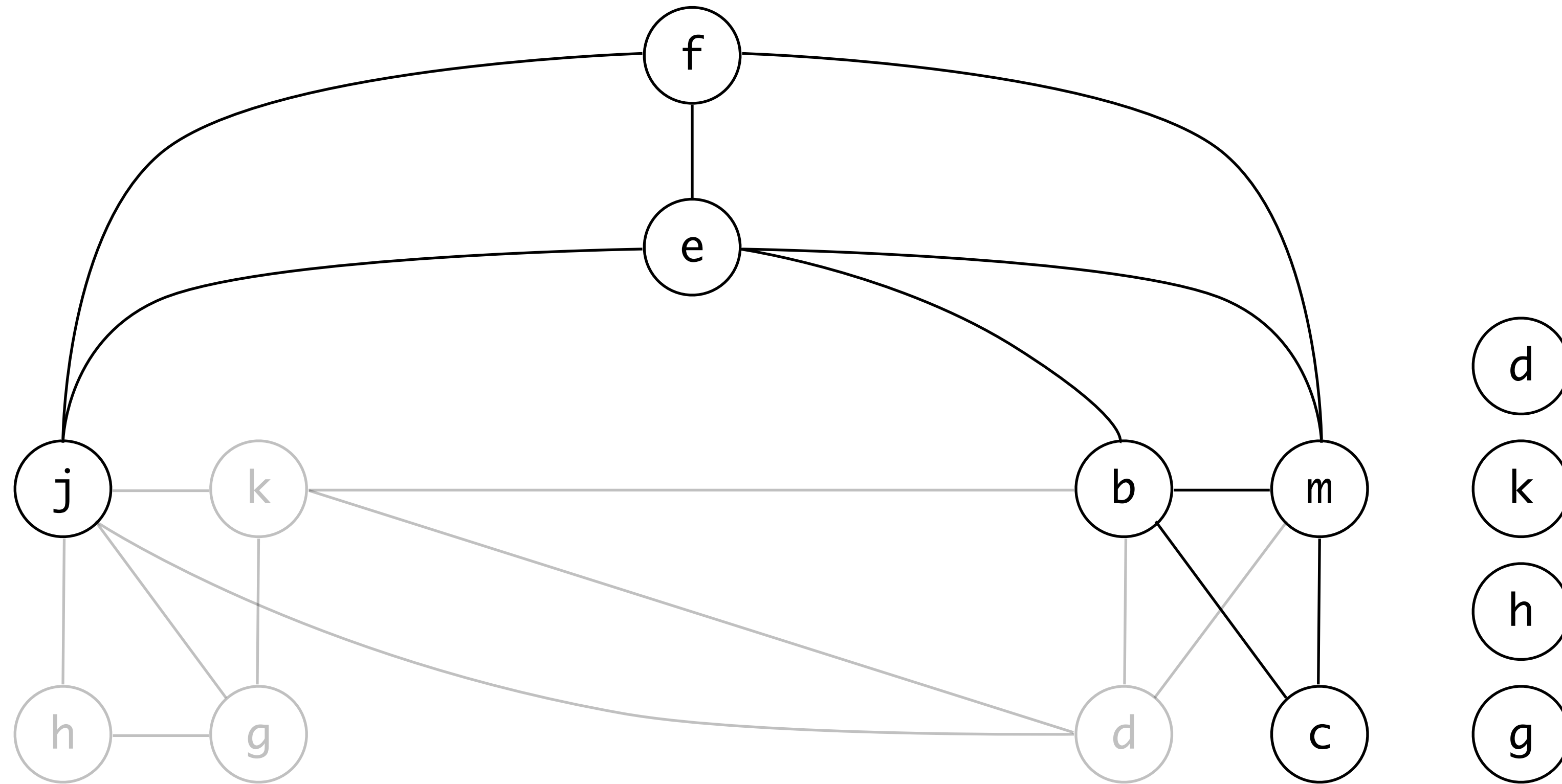
Graph Coloring: Example with 4 Colors

r_1
 r_2
 r_3
 r_4



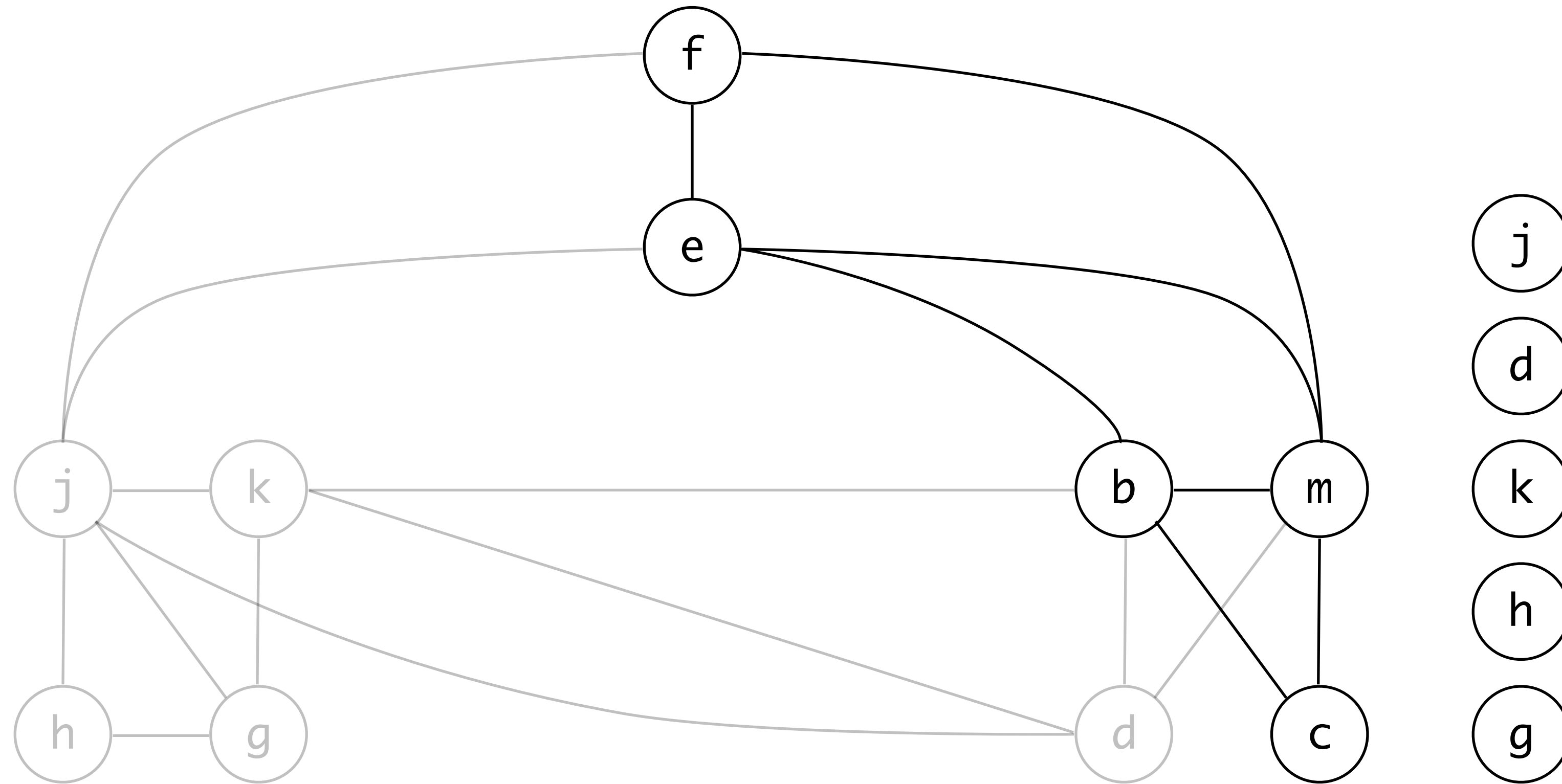
Graph Coloring: Example with 4 Colors

r_1
 r_2
 r_3
 r_4



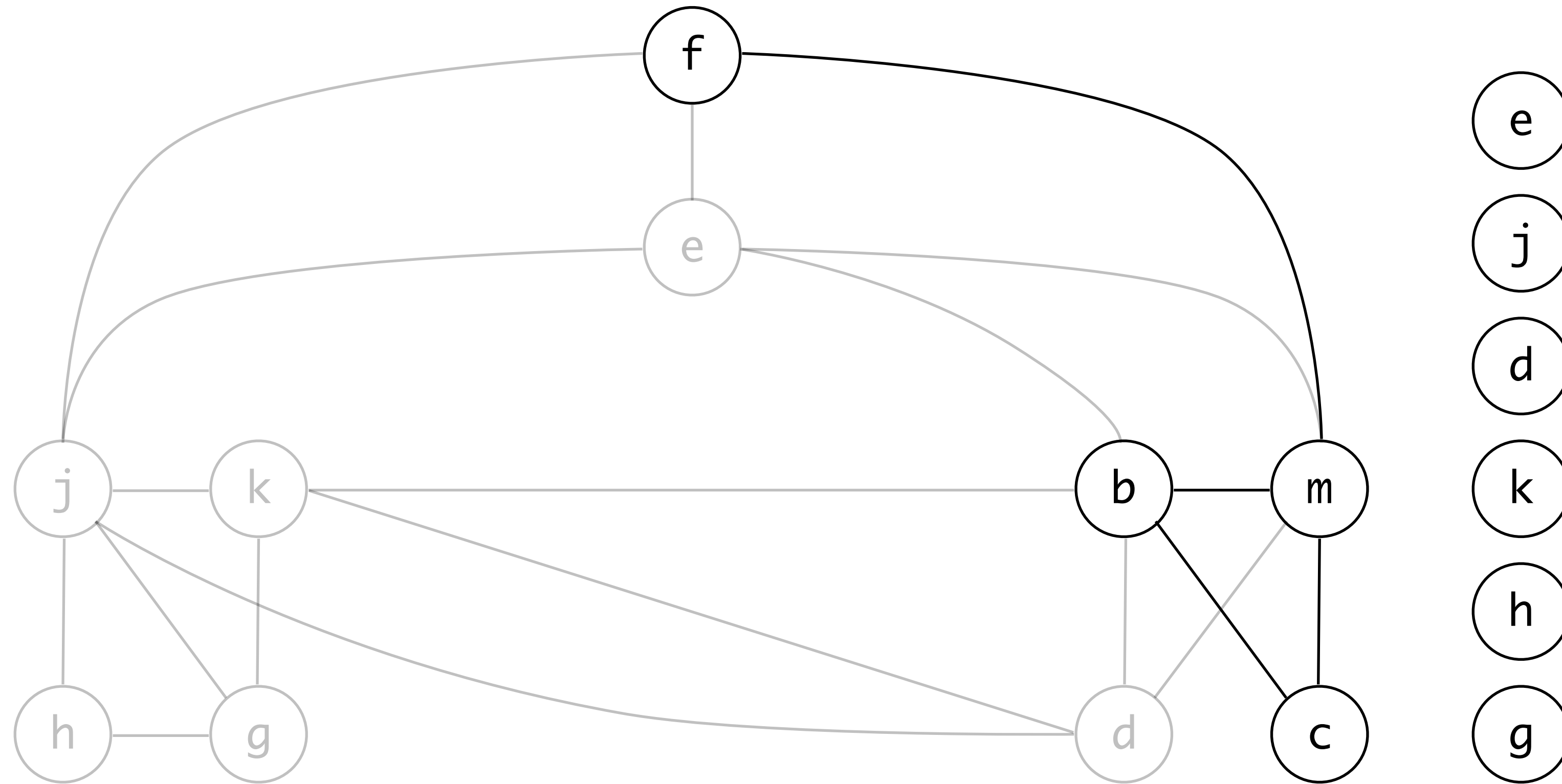
Graph Coloring: Example with 4 Colors

r_1
 r_2
 r_3
 r_4



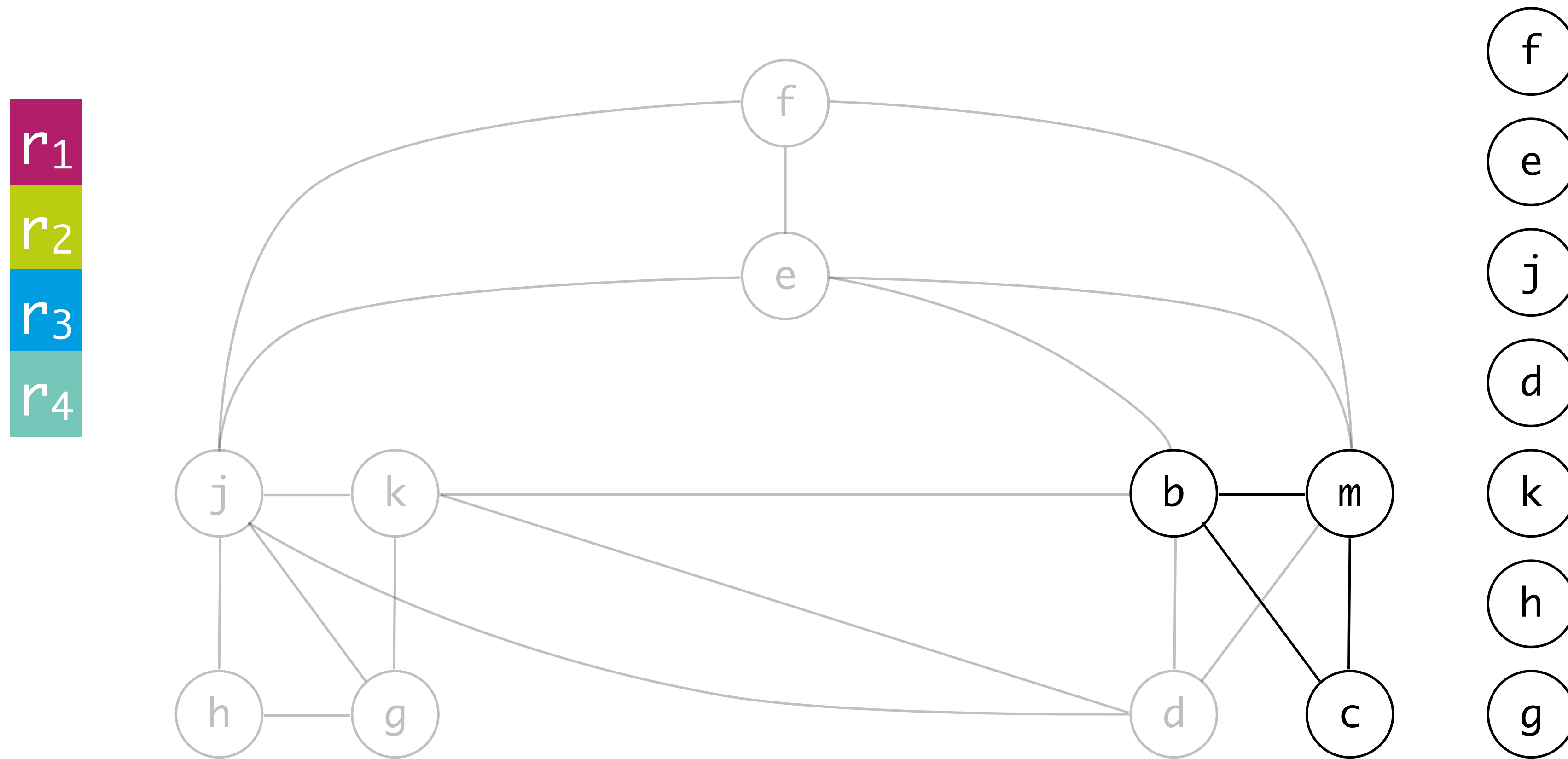
Graph Coloring: Example with 4 Colors

r_1
 r_2
 r_3
 r_4

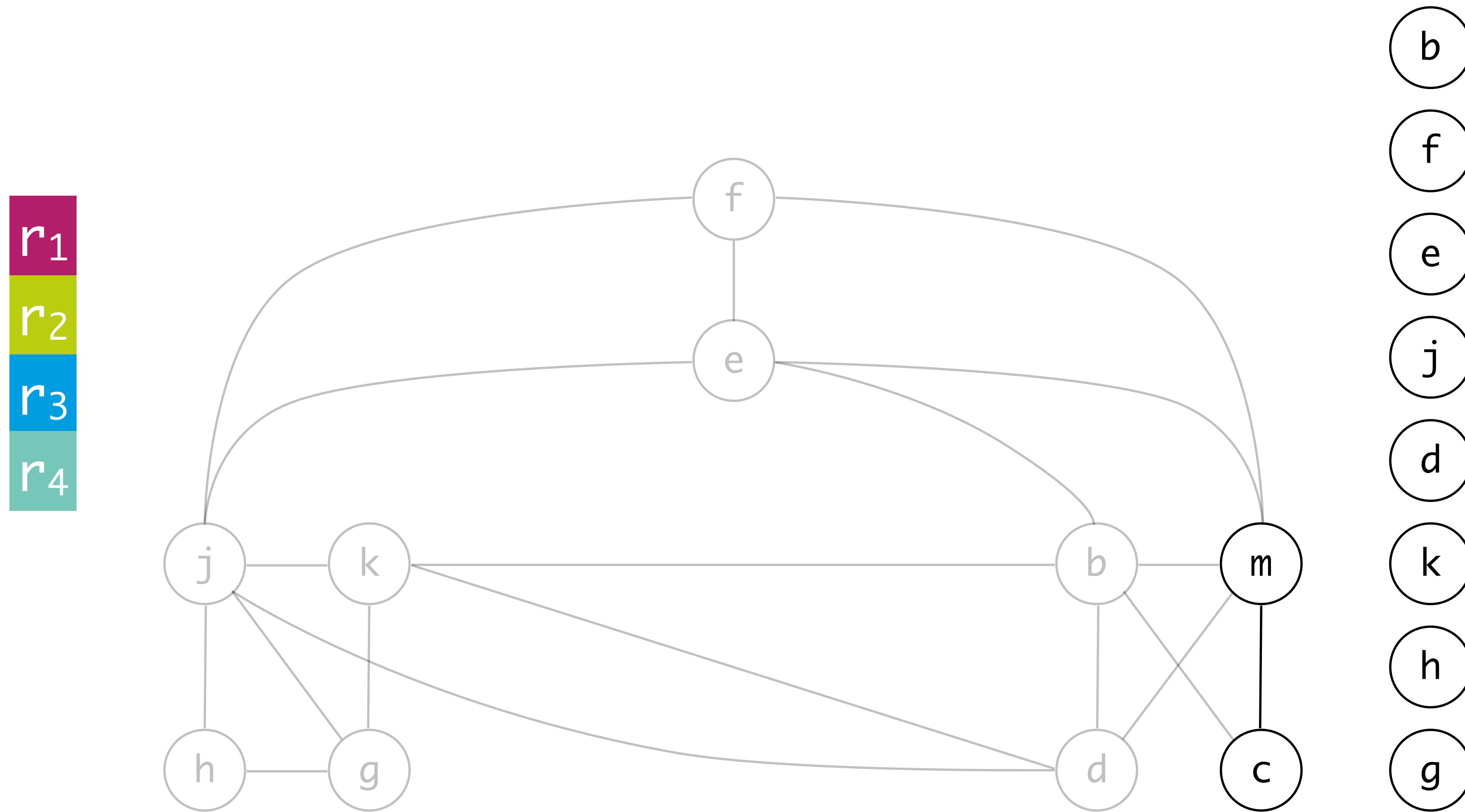


e
j
d
k
h
g

Graph Coloring: Example with 4 Colors

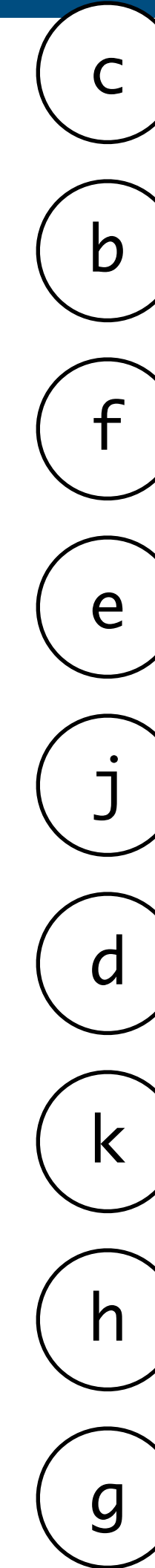
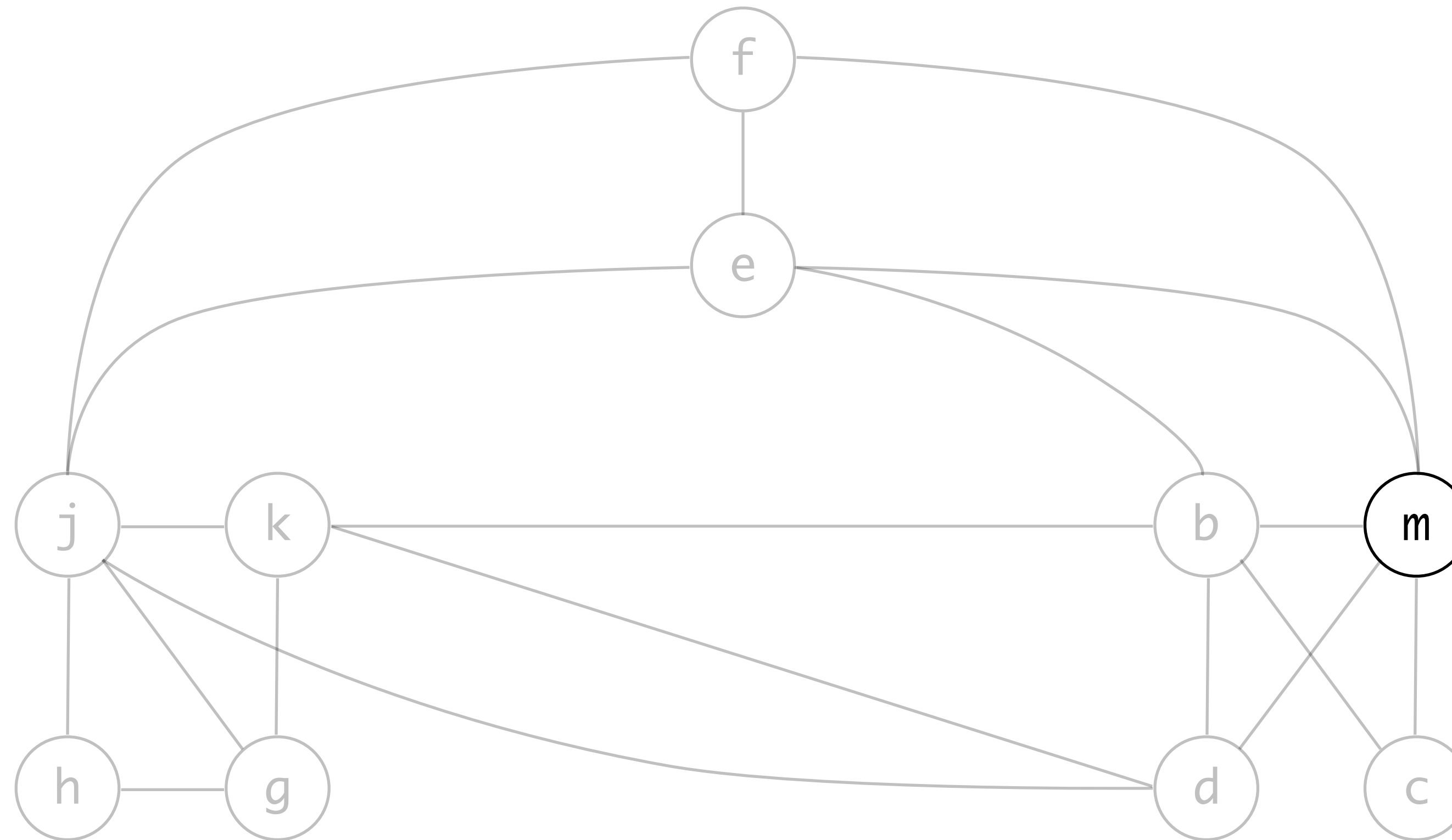


Graph Coloring: Example with 4 Colors



Graph Coloring: Example with 4 Colors

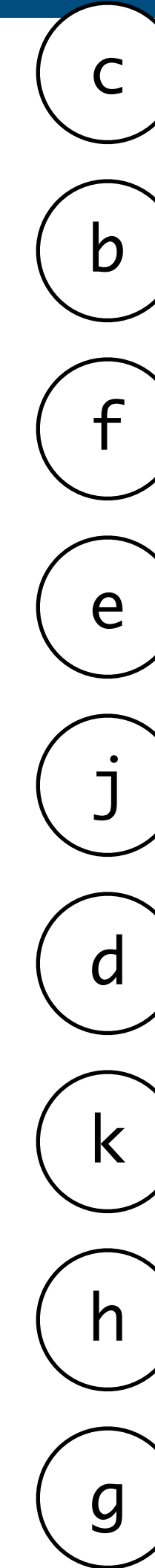
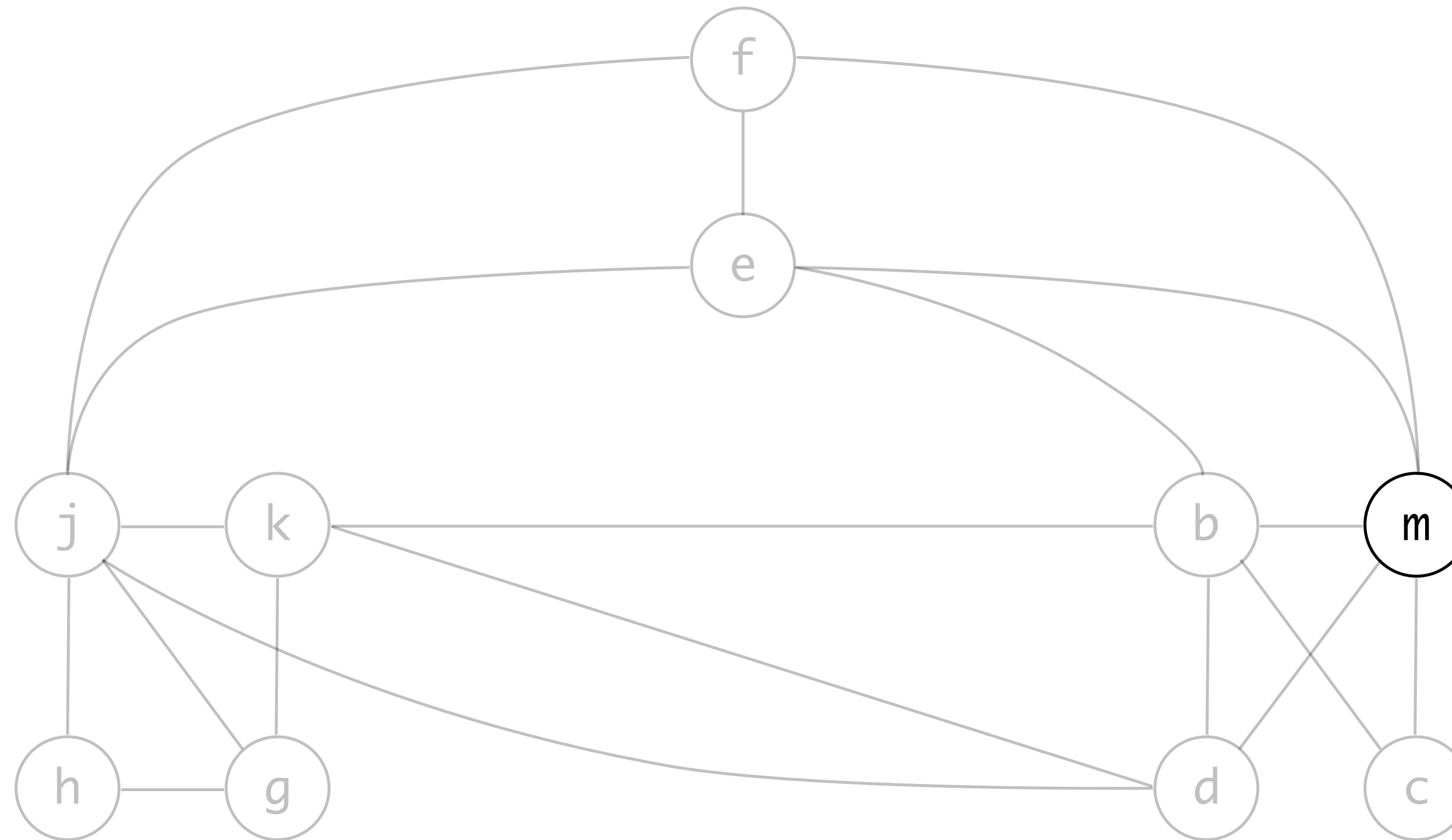
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```


Graph Coloring: Example with 4 Colors

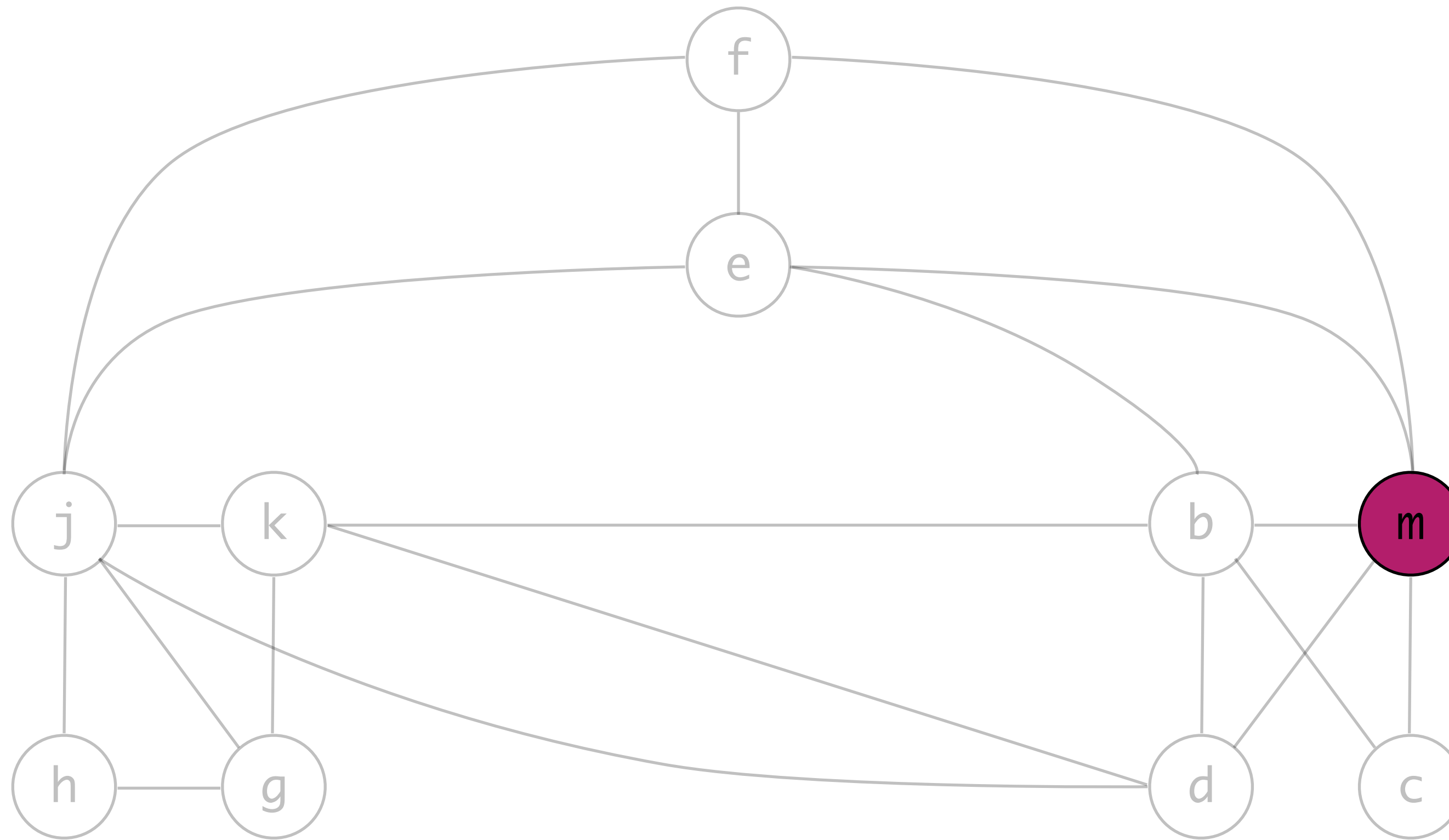
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Graph Coloring

r_1
 r_2
 r_3
 r_4

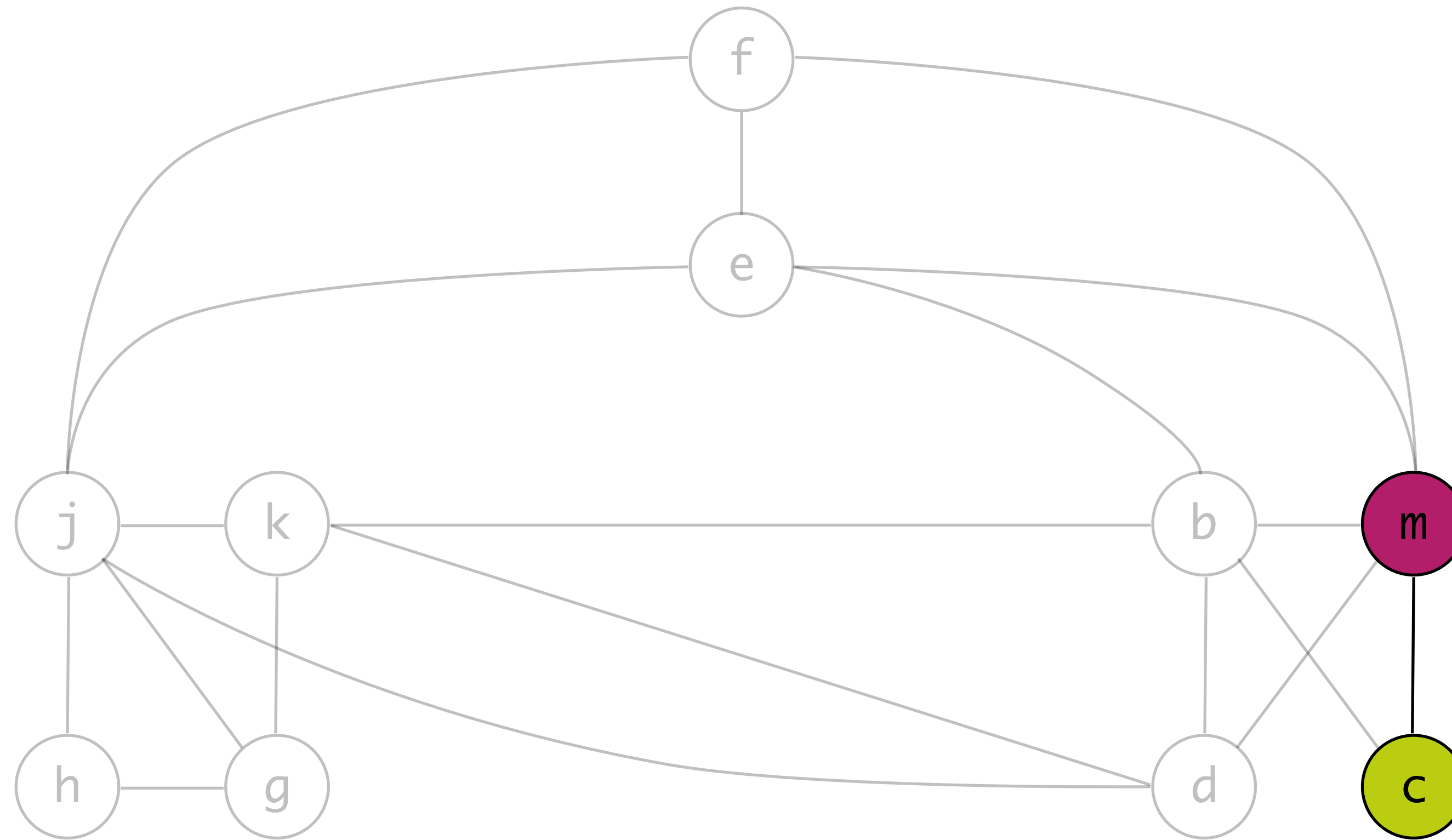


c
b
f
e
j
d
k
h
g

```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
r1 := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := r1 + 4
j := b
live out: d k j
```

Graph Coloring

r_1
 r_2
 r_3
 r_4

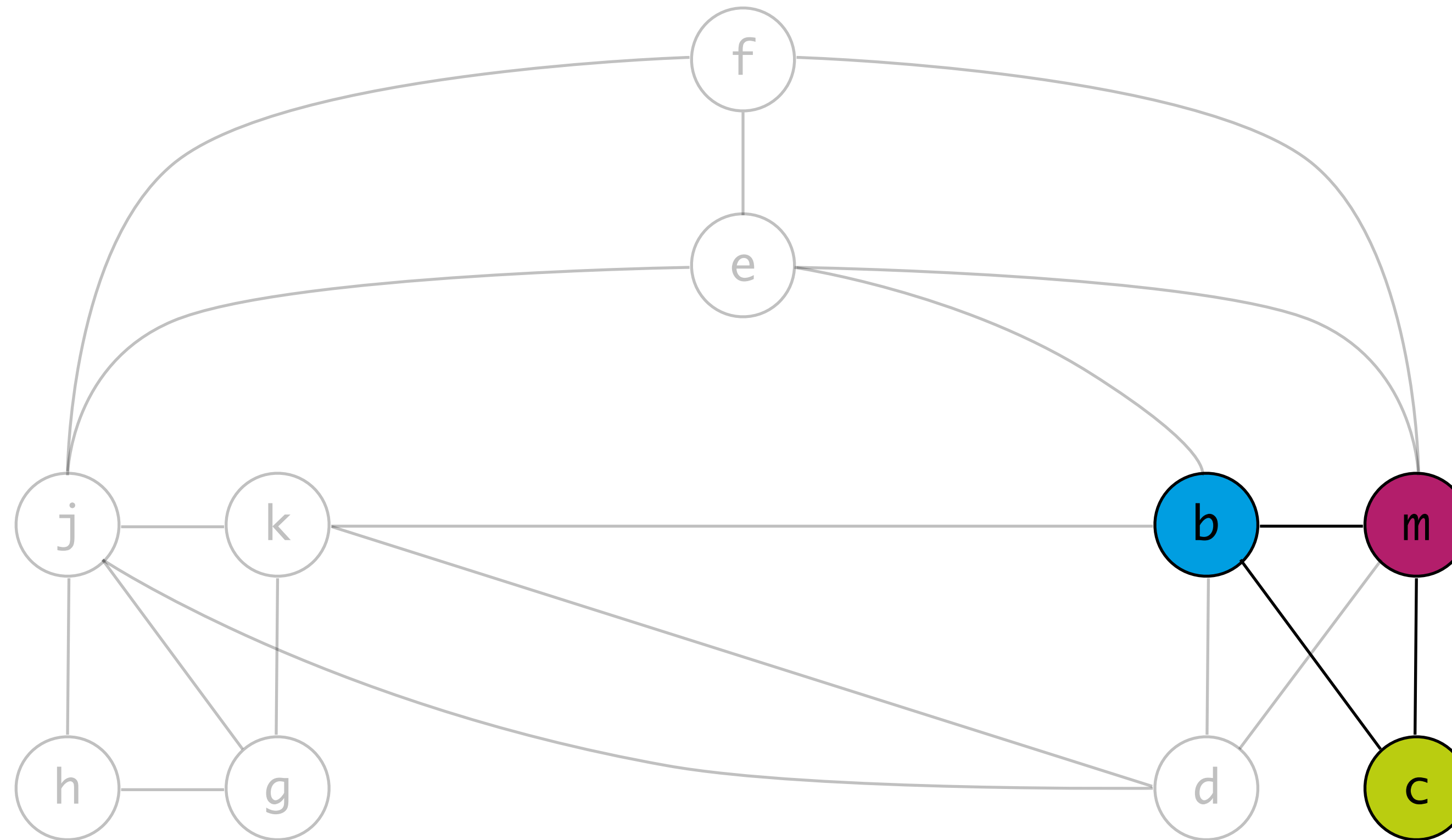


b
f
e
j
d
k
h
g

```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
r1 := mem[j + 16]
b := mem[f]
r2 := e + 8
d := r2
k := r1 + 4
j := b
live out: d k j
```

Graph Coloring

r_1
 r_2
 r_3
 r_4

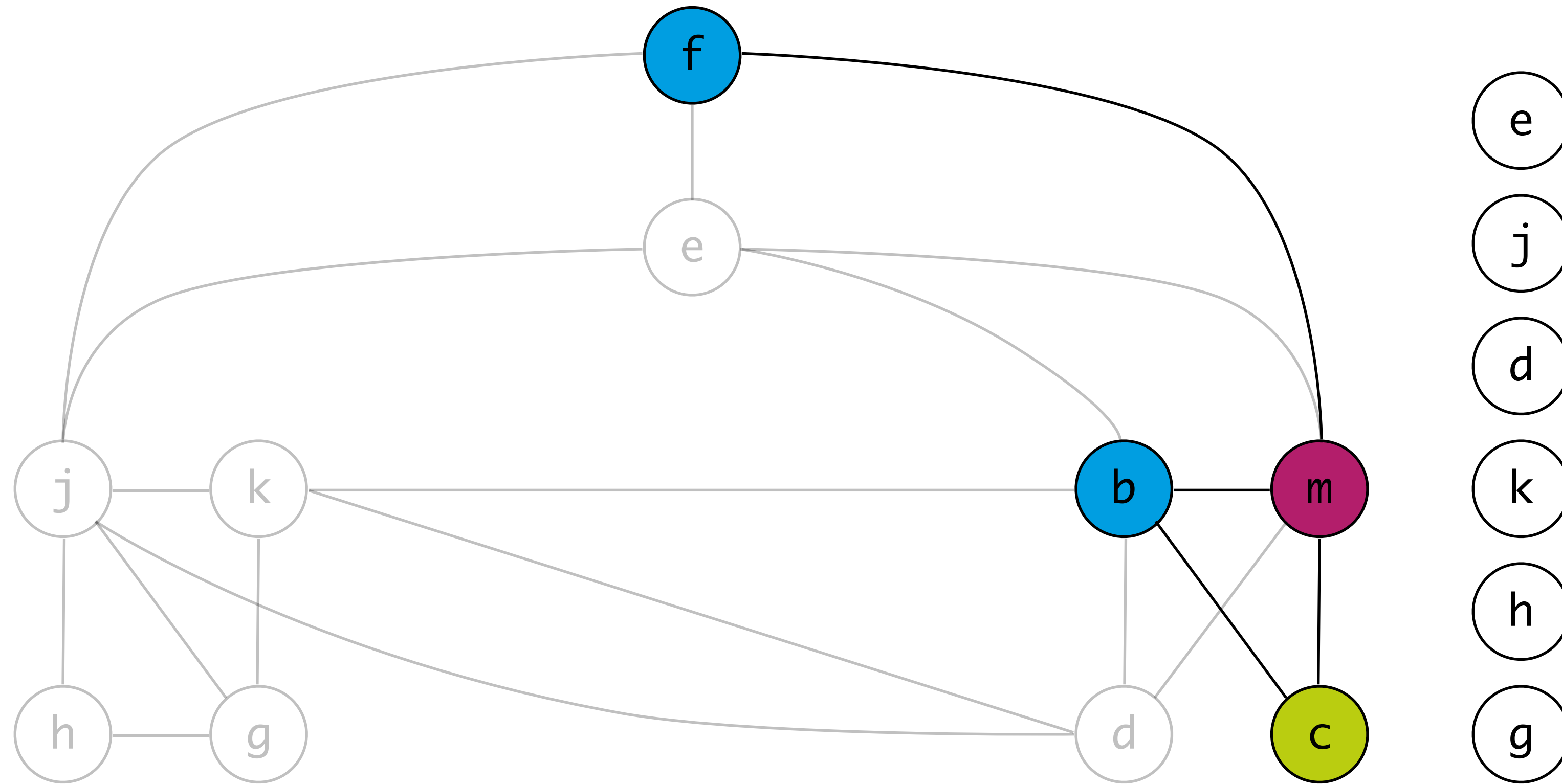


f
e
j
d
k
h
g

```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
r1 := mem[j + 16]
r3 := mem[f]
r2 := e + 8
d := r2
k := r1 + 4
j := r3
live out: d k j
```

Graph Coloring

r_1
 r_2
 r_3
 r_4

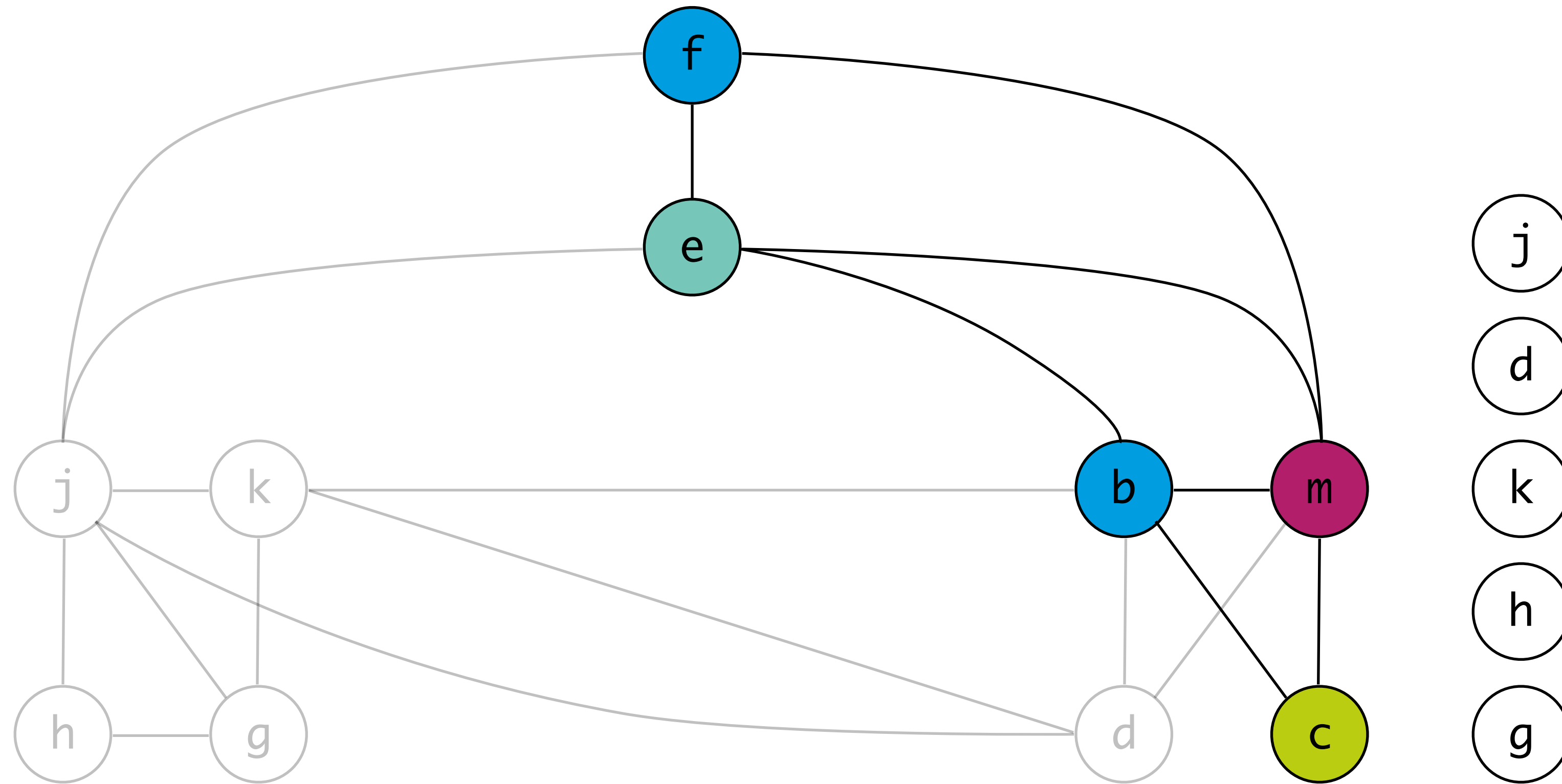


```
live-in: k j
g := mem[j + 12]
h := k - 1
r3 := g * h
e := mem[j + 8]
r1 := mem[j + 16]
r3 := mem[r3]
r2 := e + 8
d := r2
k := r1 + 4
j := r3
live out: d k j
```

e
j
d
k
h
g

Graph Coloring

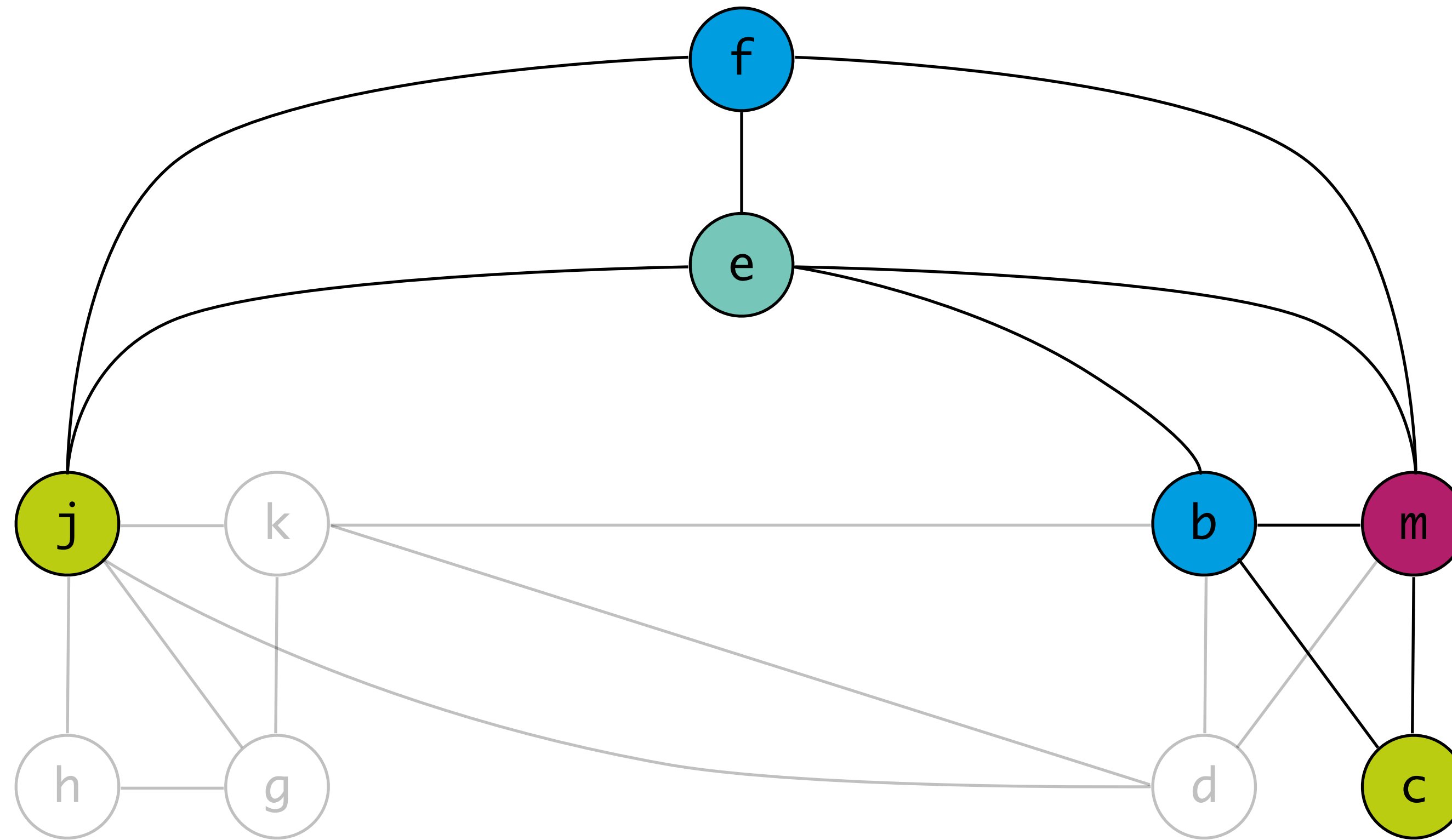
r_1
 r_2
 r_3
 r_4



```
live-in: k j  
g := mem[j + 12]  
h := k - 1  
r3 := g * h  
r4 := mem[j + 8]  
r1 := mem[j + 16]  
r3 := mem[r3]  
r2 := r4 + 8  
d := r2  
k := r1 + 4  
j := r3  
live out: d k j
```

Graph Coloring

r_1
 r_2
 r_3
 r_4

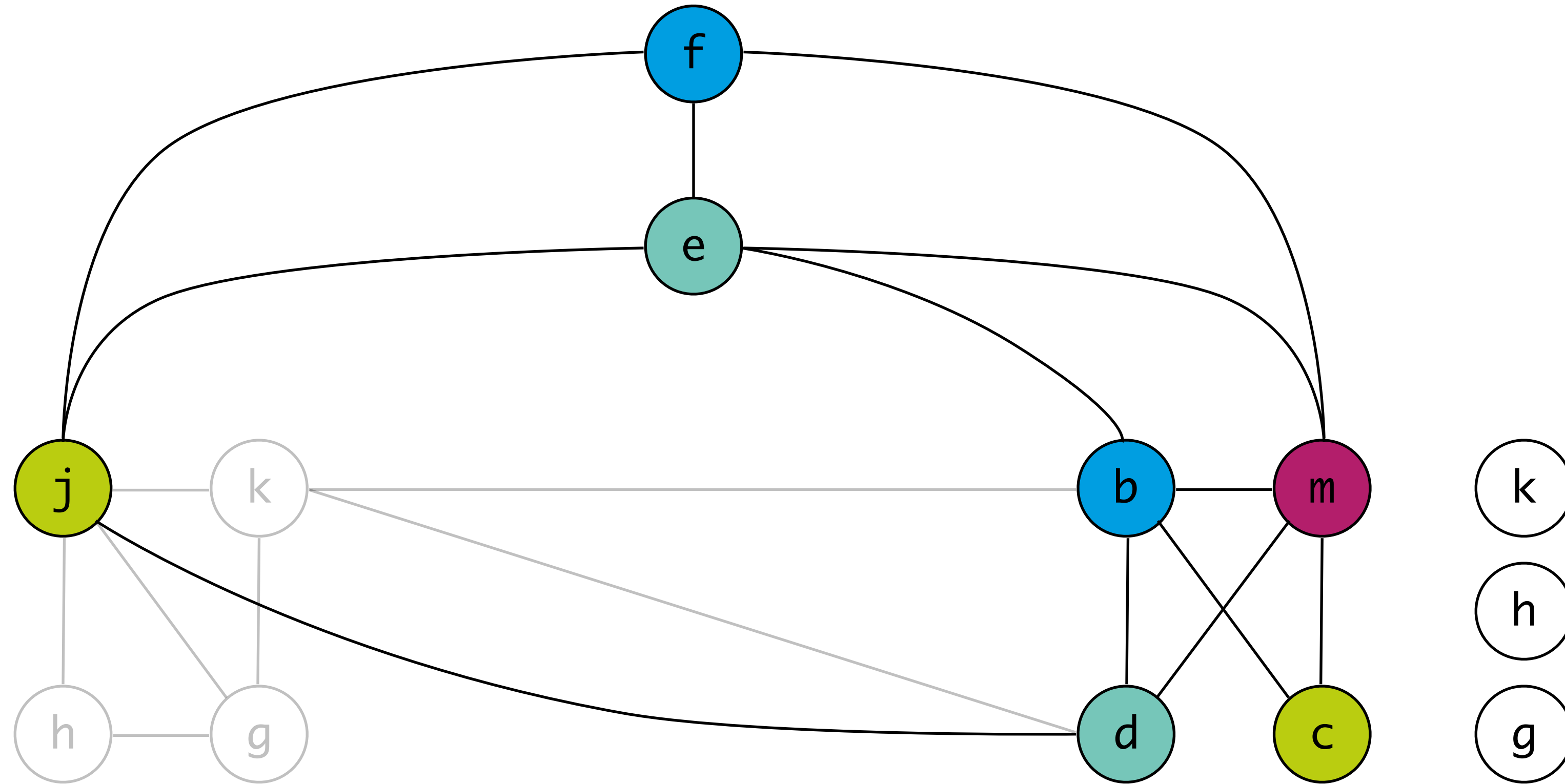


d
k
h
g

```
live-in: k  $r_2$ 
g := mem[ $r_2$  + 12]
h := k - 1
 $r_3$  := g * h
 $r_4$  := mem[ $r_2$  + 8]
 $r_1$  := mem[ $r_2$  + 16]
 $r_3$  := mem[ $r_3$ ]
 $r_2$  :=  $r_4$  + 8
d :=  $r_2$ 
k :=  $r_1$  + 4
 $r_2$  :=  $r_3$ 
live out: d k  $r_2$ 
```

Graph Coloring

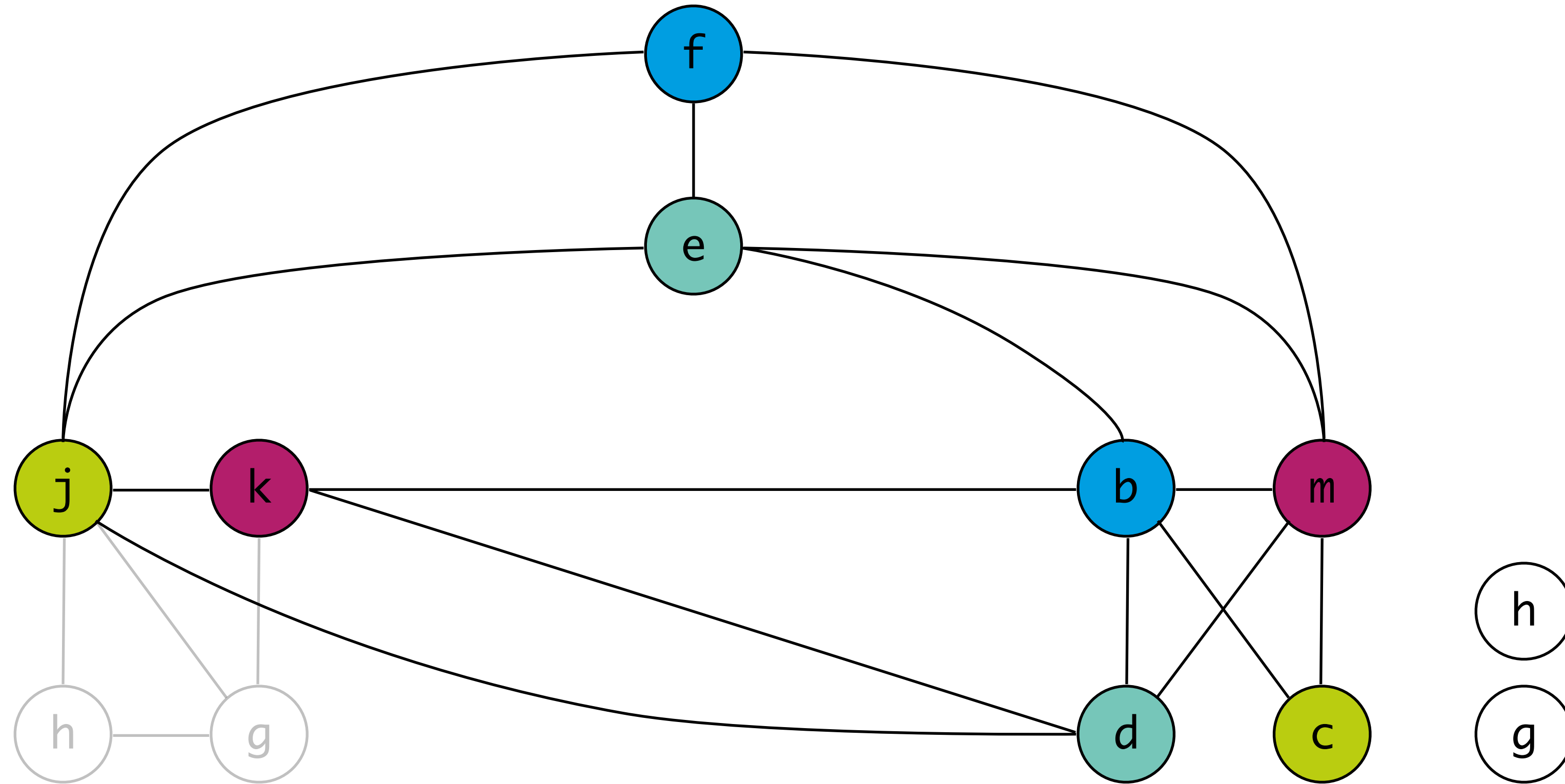
r_1
 r_2
 r_3
 r_4



```
live-in:  $k$   $r_2$   
 $g := \text{mem}[r_2 + 12]$   
 $h := k - 1$   
 $r_3 := g * h$   
 $r_4 := \text{mem}[r_2 + 8]$   
 $r_1 := \text{mem}[r_2 + 16]$   
 $r_3 := \text{mem}[r_3]$   
 $r_2 := r_4 + 8$   
 $r_4 := r_2$   
 $k := r_1 + 4$   
 $r_2 := r_3$   
live out:  $r_4$   $k$   $r_2$ 
```


Graph Coloring

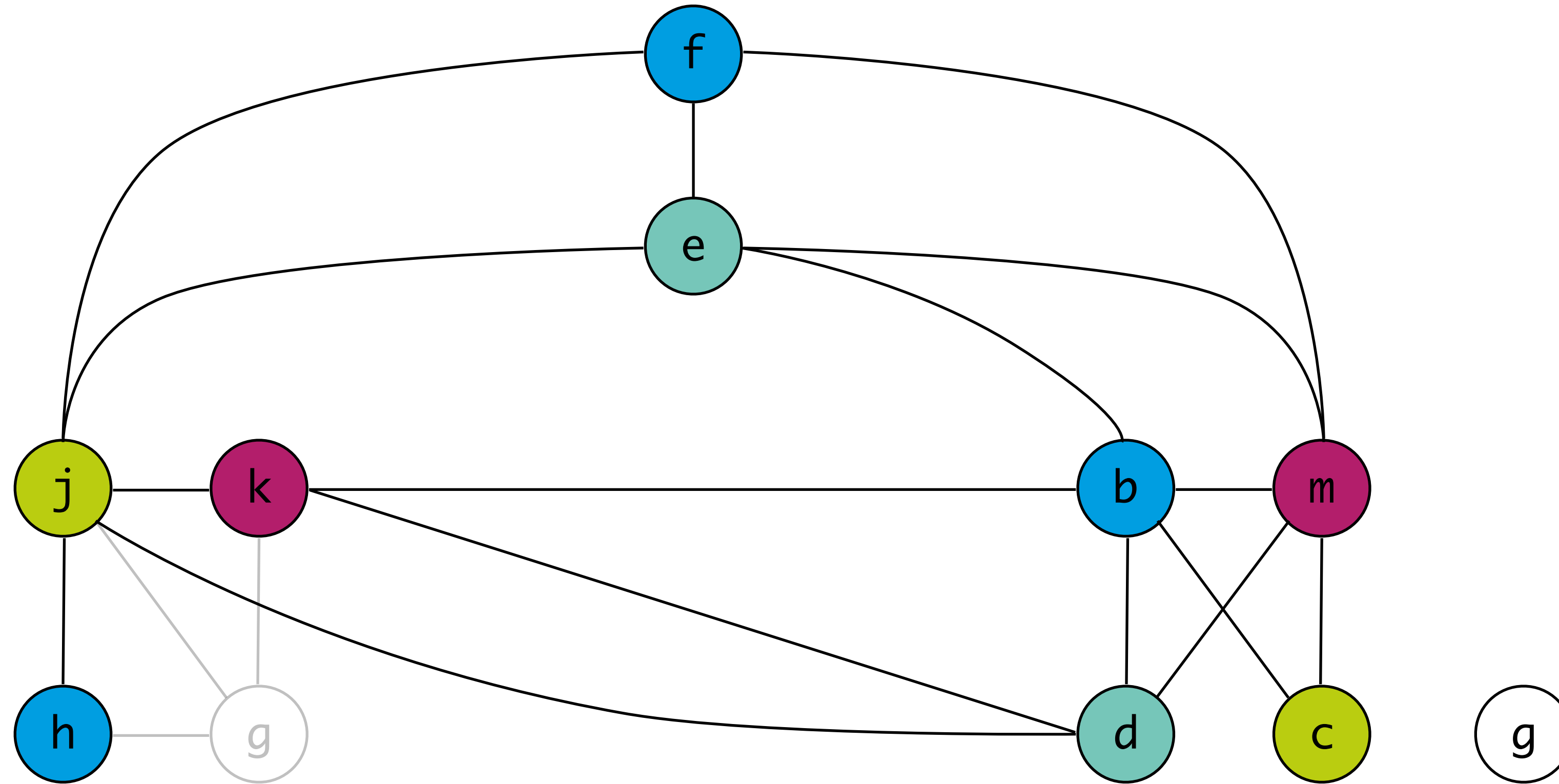
r_1
 r_2
 r_3
 r_4



```
live-in:  $r_1$   $r_2$   
 $g := \text{mem}[r_2 + 12]$   
 $h := r_1 - 1$   
 $r_3 := g * h$   
 $r_4 := \text{mem}[r_2 + 8]$   
 $r_1 := \text{mem}[r_2 + 16]$   
 $r_3 := \text{mem}[r_3]$   
 $r_2 := r_4 + 8$   
 $r_4 := r_2$   
 $r_1 := r_1 + 4$   
 $r_2 := r_3$   
live out:  $r_4$   $r_1$   $r_2$ 
```

Graph Coloring

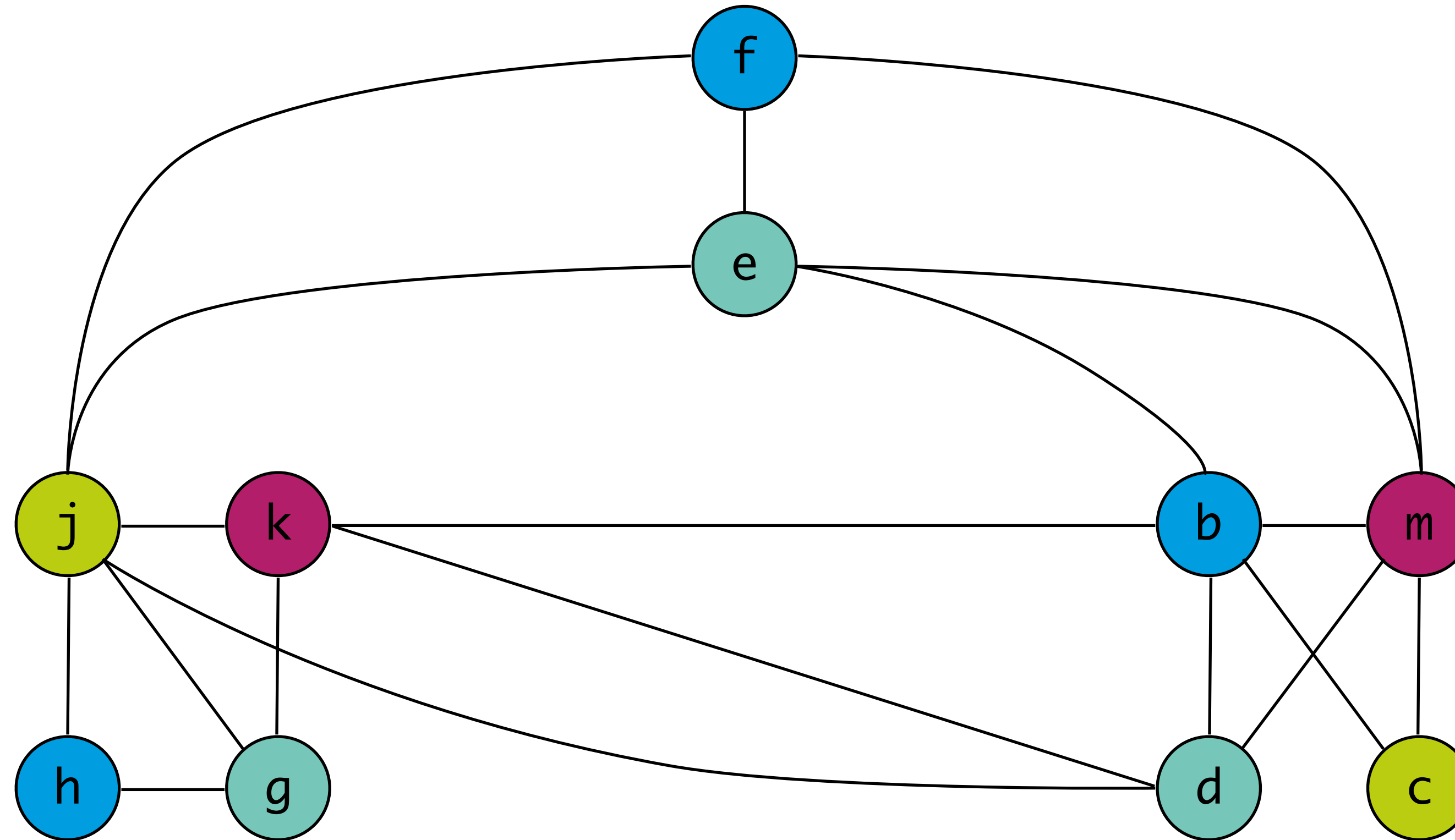
r_1
 r_2
 r_3
 r_4



```
live-in:  $r_1$   $r_2$   
 $g$  := mem[ $r_2$  + 12]  
 $r_3$  :=  $r_1$  - 1  
 $r_3$  :=  $g$  *  $r_3$   
 $r_4$  := mem[ $r_2$  + 8]  
 $r_1$  := mem[ $r_2$  + 16]  
 $r_3$  := mem[ $r_3$ ]  
 $r_2$  :=  $r_4$  + 8  
 $r_4$  :=  $r_2$   
 $r_1$  :=  $r_1$  + 4  
 $r_2$  :=  $r_3$   
live out:  $r_4$   $r_1$   $r_2$ 
```

Graph Coloring

r_1
 r_2
 r_3
 r_4



```
live-in:  $r_1$   $r_2$   
 $r_4 := \text{mem}[r_2 + 12]$   
 $r_3 := r_1 - 1$   
 $r_3 := r_4 * r_3$   
 $r_4 := \text{mem}[r_2 + 8]$   
 $r_1 := \text{mem}[r_2 + 16]$   
 $r_3 := \text{mem}[r_3]$   
 $r_2 := r_4 + 8$   
 $r_4 := r_2$   
 $r_1 := r_1 + 4$   
 $r_2 := r_3$   
live out:  $r_4$   $r_1$   $r_2$ 
```

Spilling

Optimistic Coloring

Simplify

- remove node of insignificant degree (fewer than k edges)

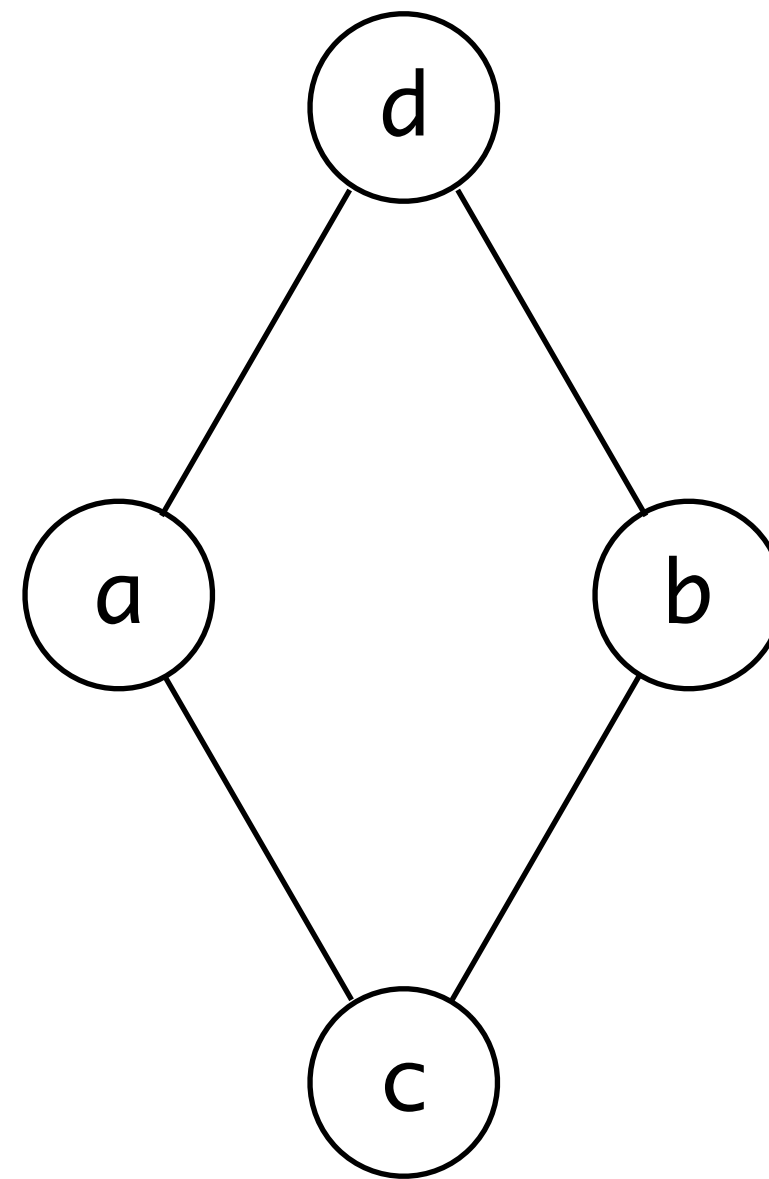
Spill

- remove node of significant degree (k or more edges)

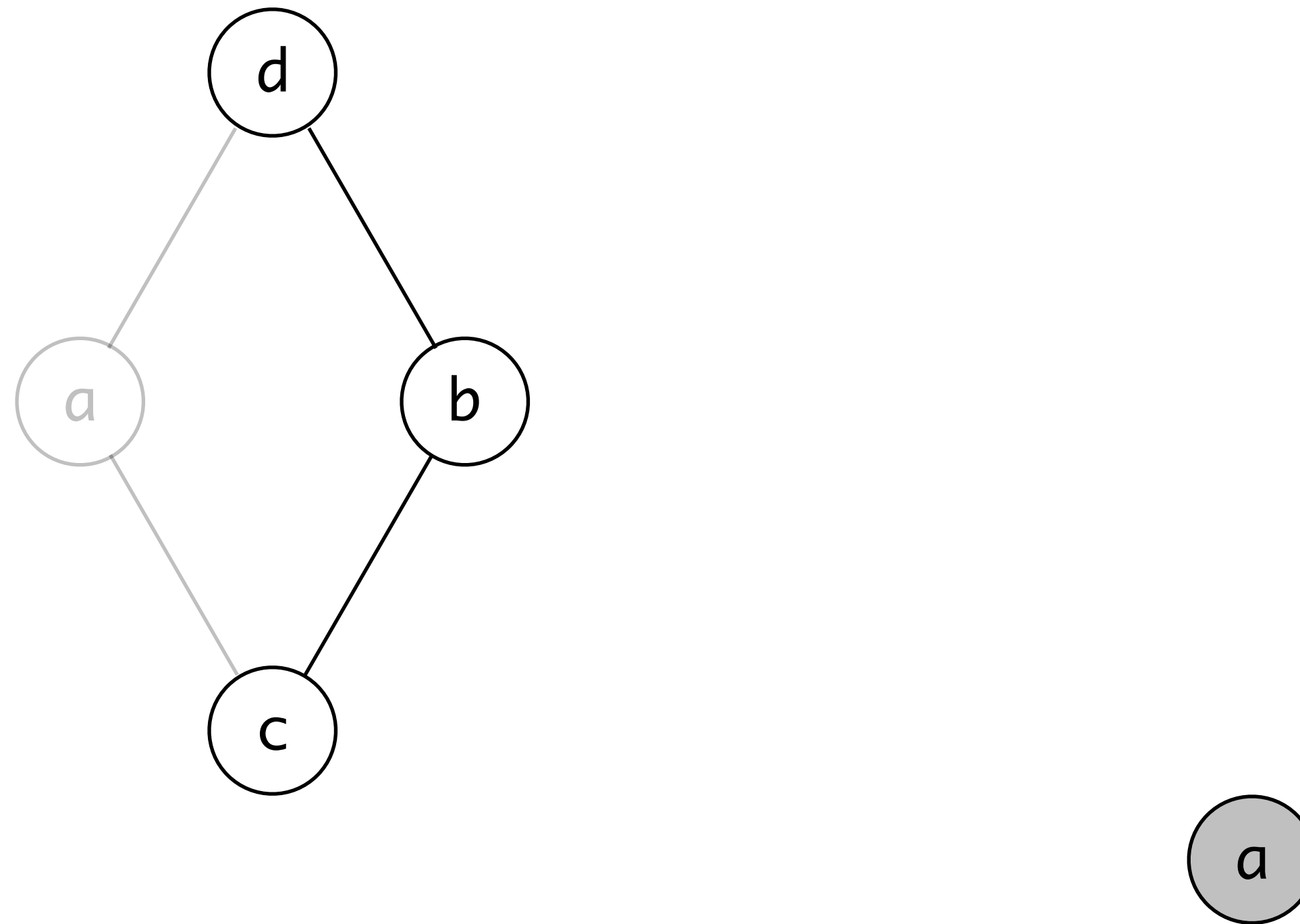
Select

- add node, select color

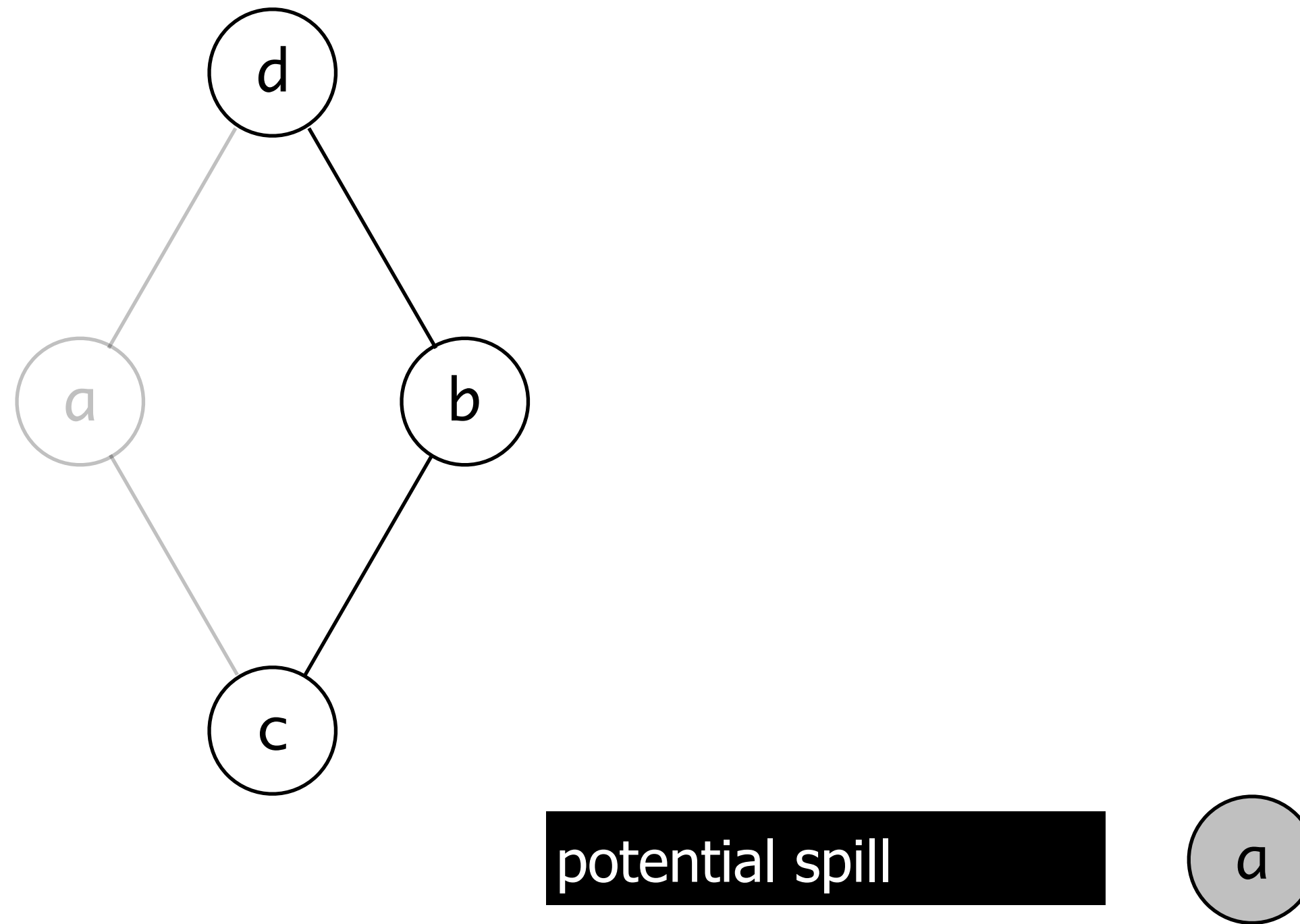
Optimistic Coloring: Example with 2 Colors



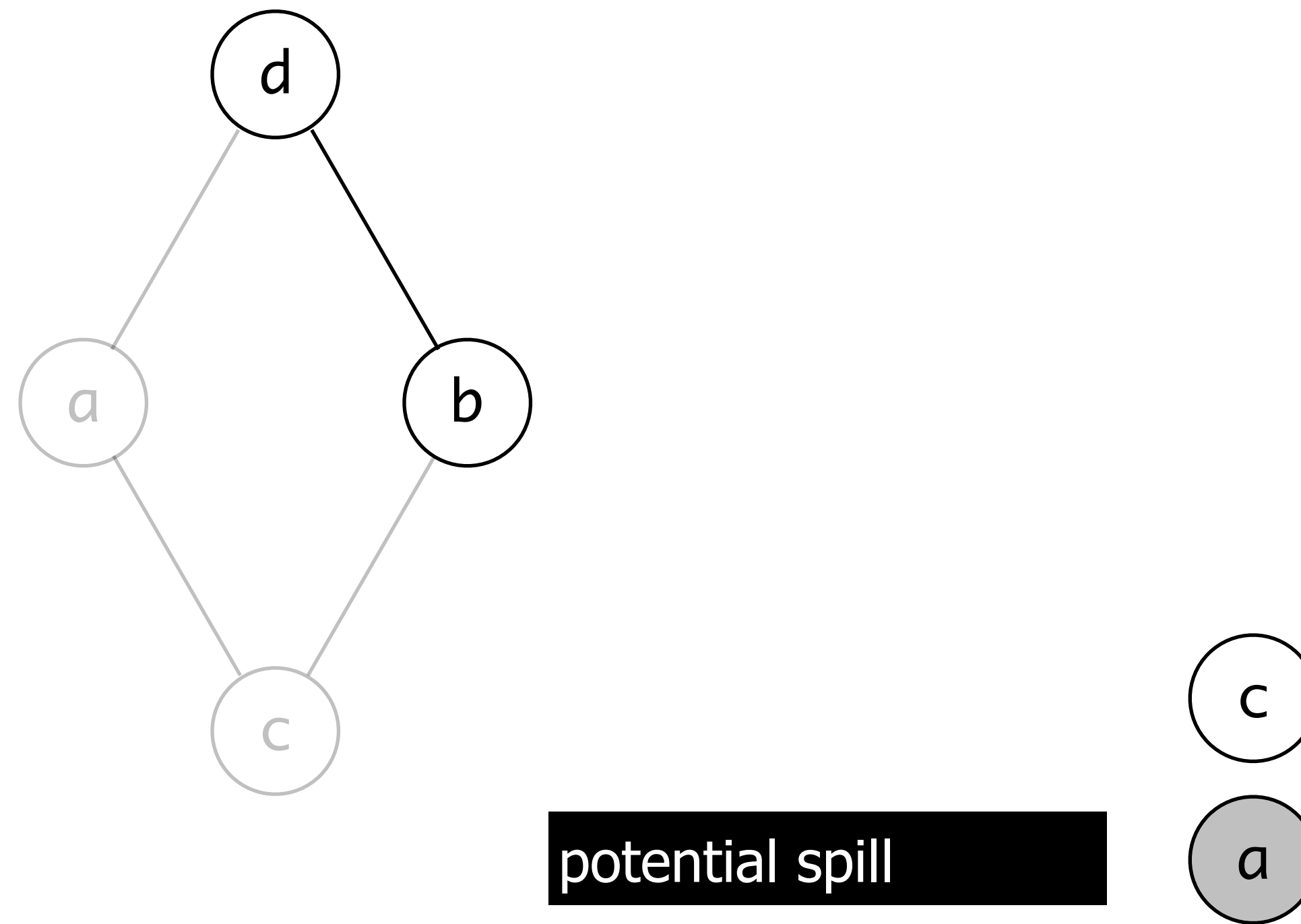
Optimistic Coloring: Example with 2 Colors



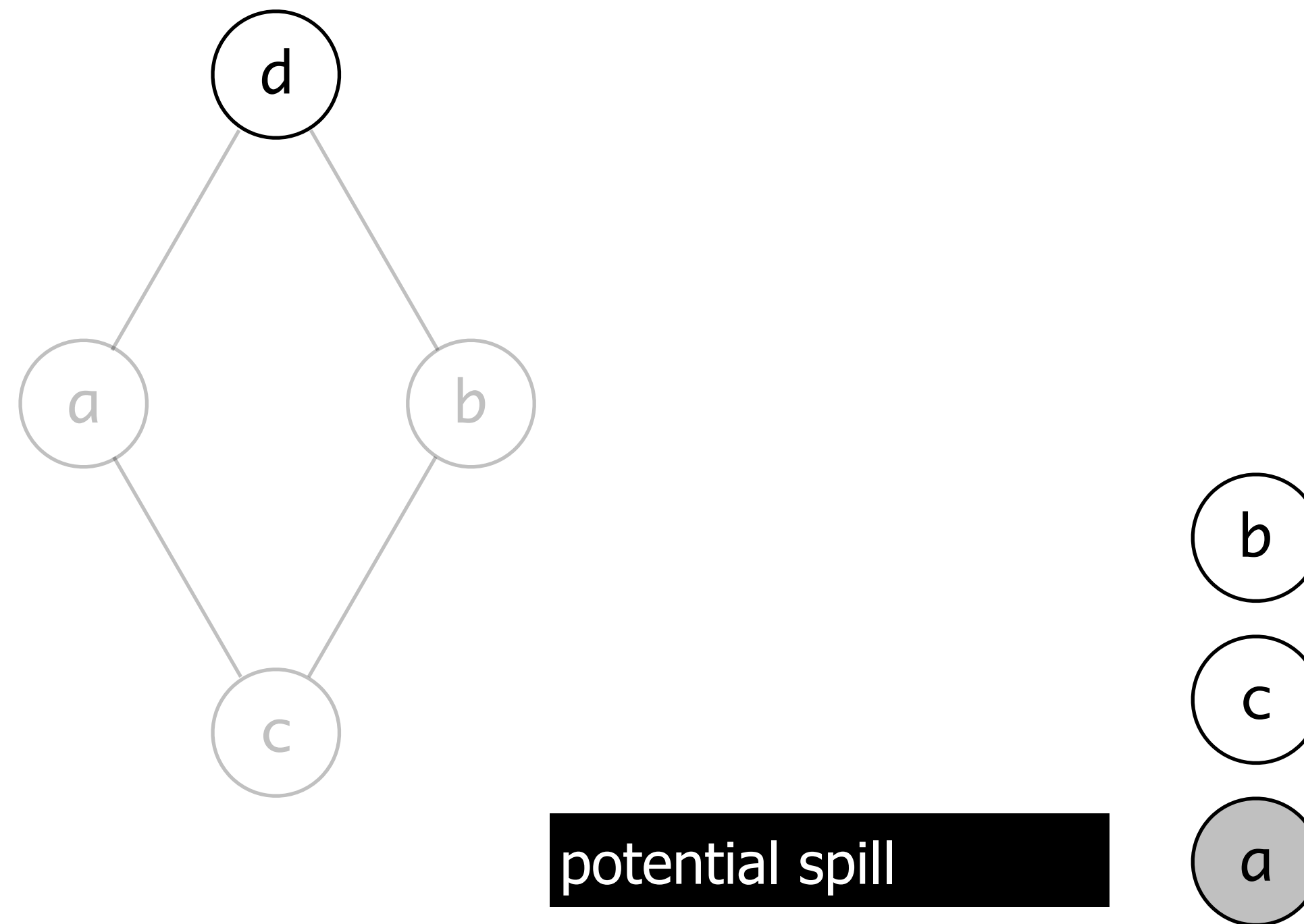
Optimistic Coloring: Example with 2 Colors



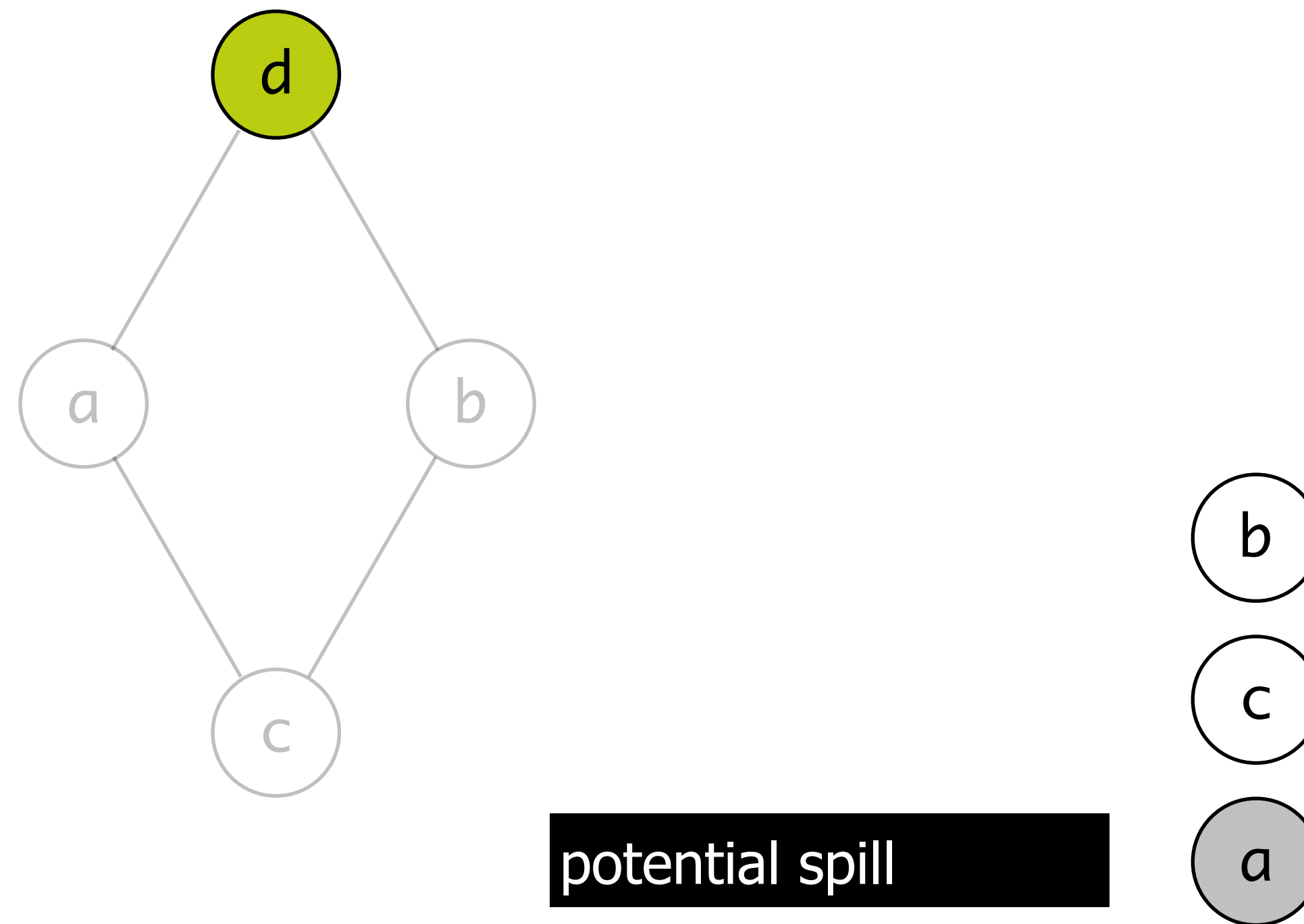
Optimistic Coloring: Example with 2 Colors



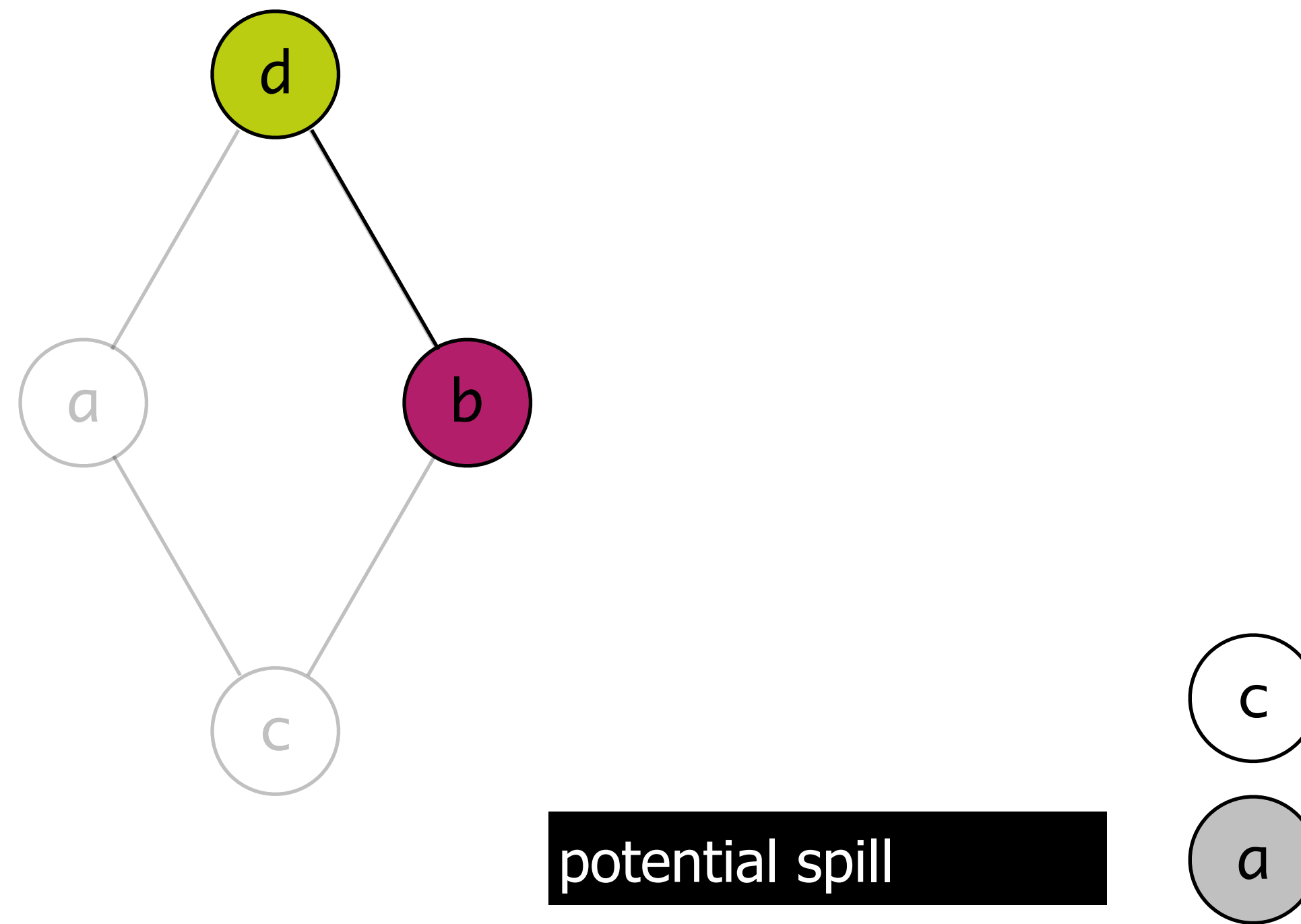
Optimistic Coloring: Example with 2 Colors



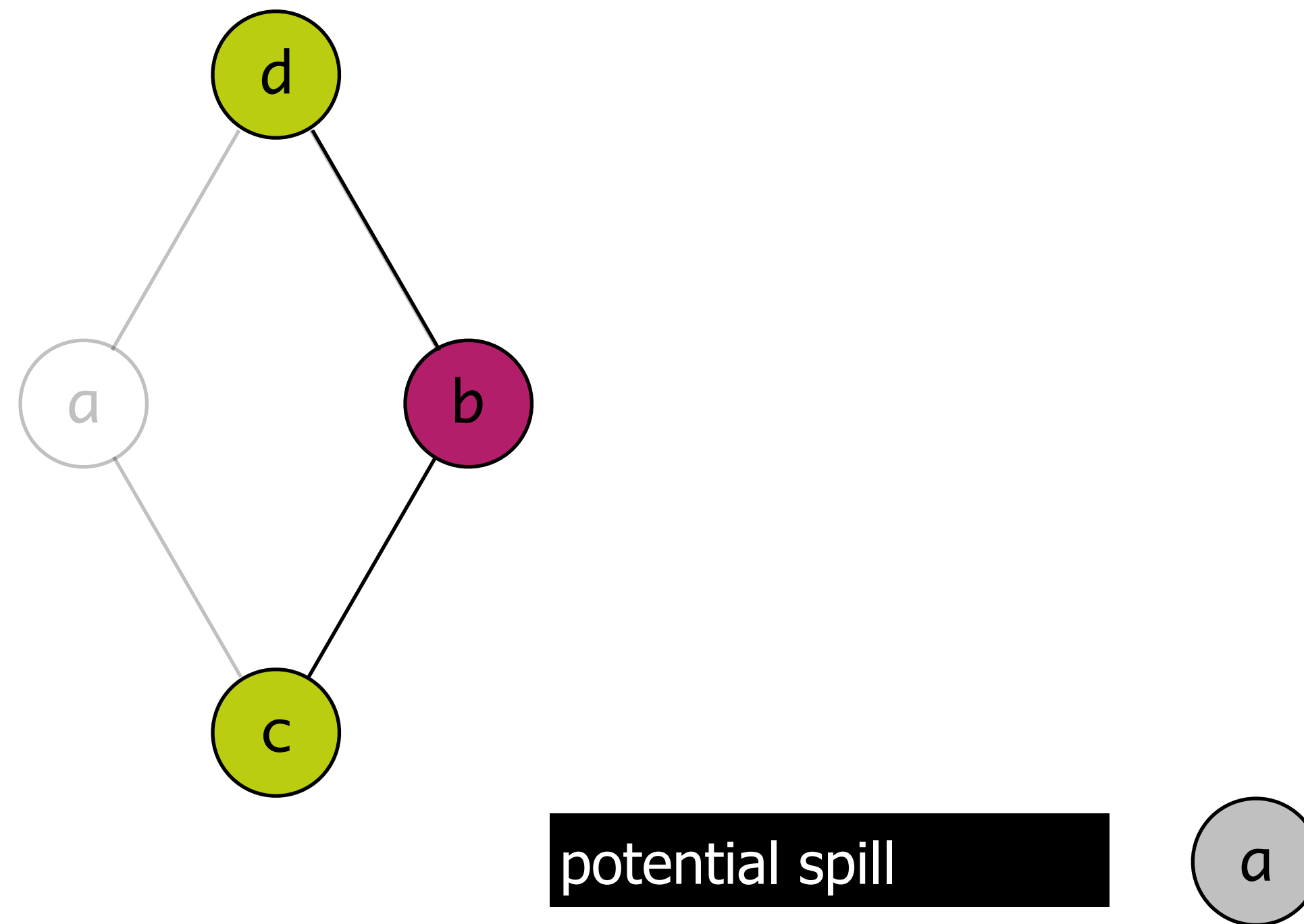
Optimistic Coloring: Example with 2 Colors



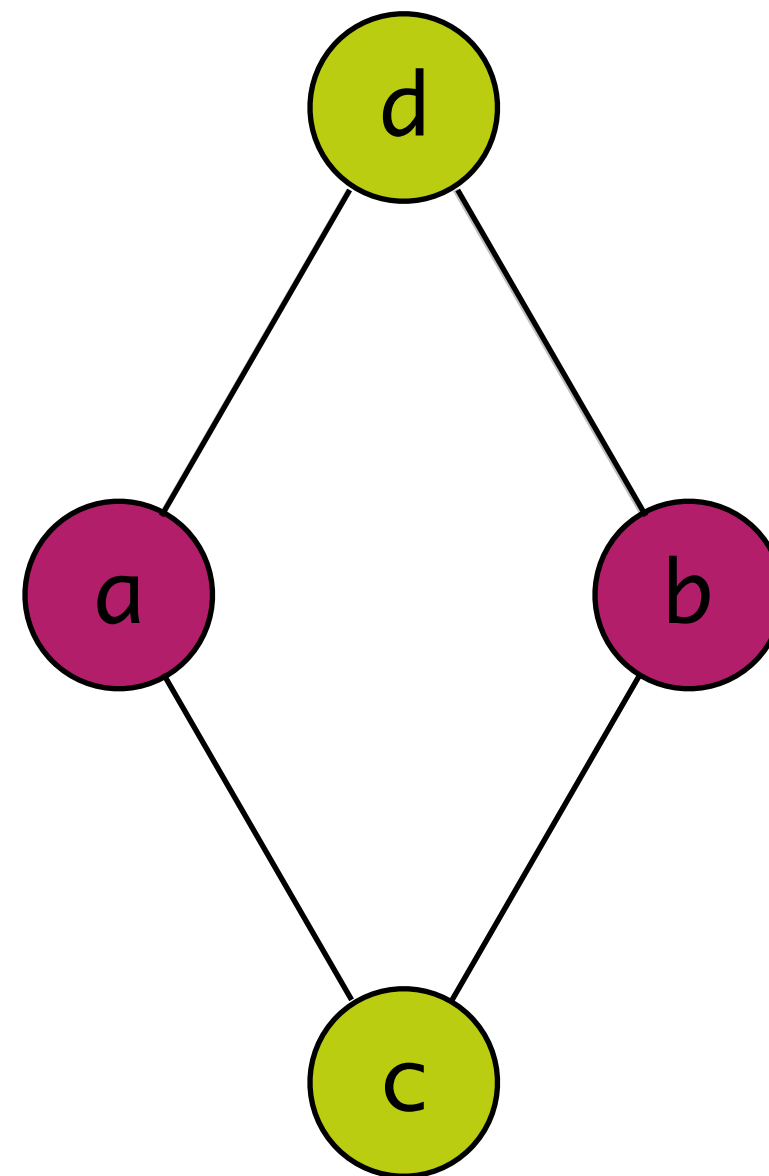
Optimistic Coloring: Example with 2 Colors



Optimistic Coloring: Example with 2 Colors



Optimistic Coloring: Example with 2 Colors



Spilling

Simplify

- remove node of insignificant degree (less than k edges)

Spill

- remove node of significant degree (k or more edges)

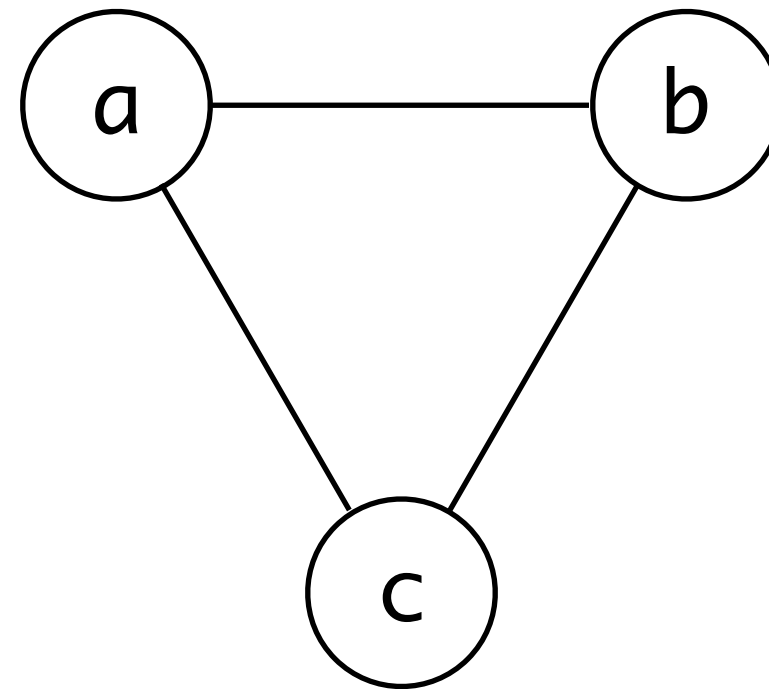
Select

- add node, select color

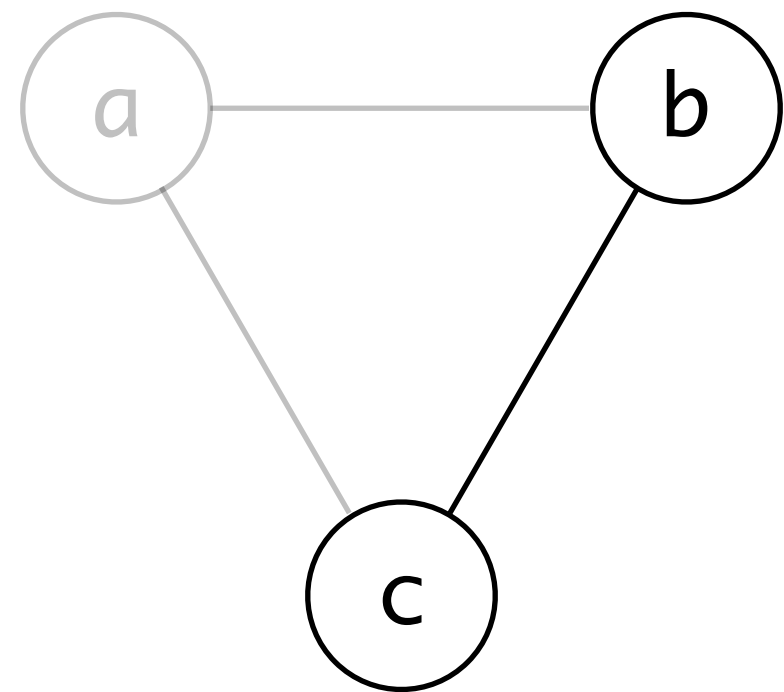
Actual spill

Start over

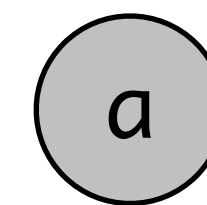
Spilling: example with 2 colors



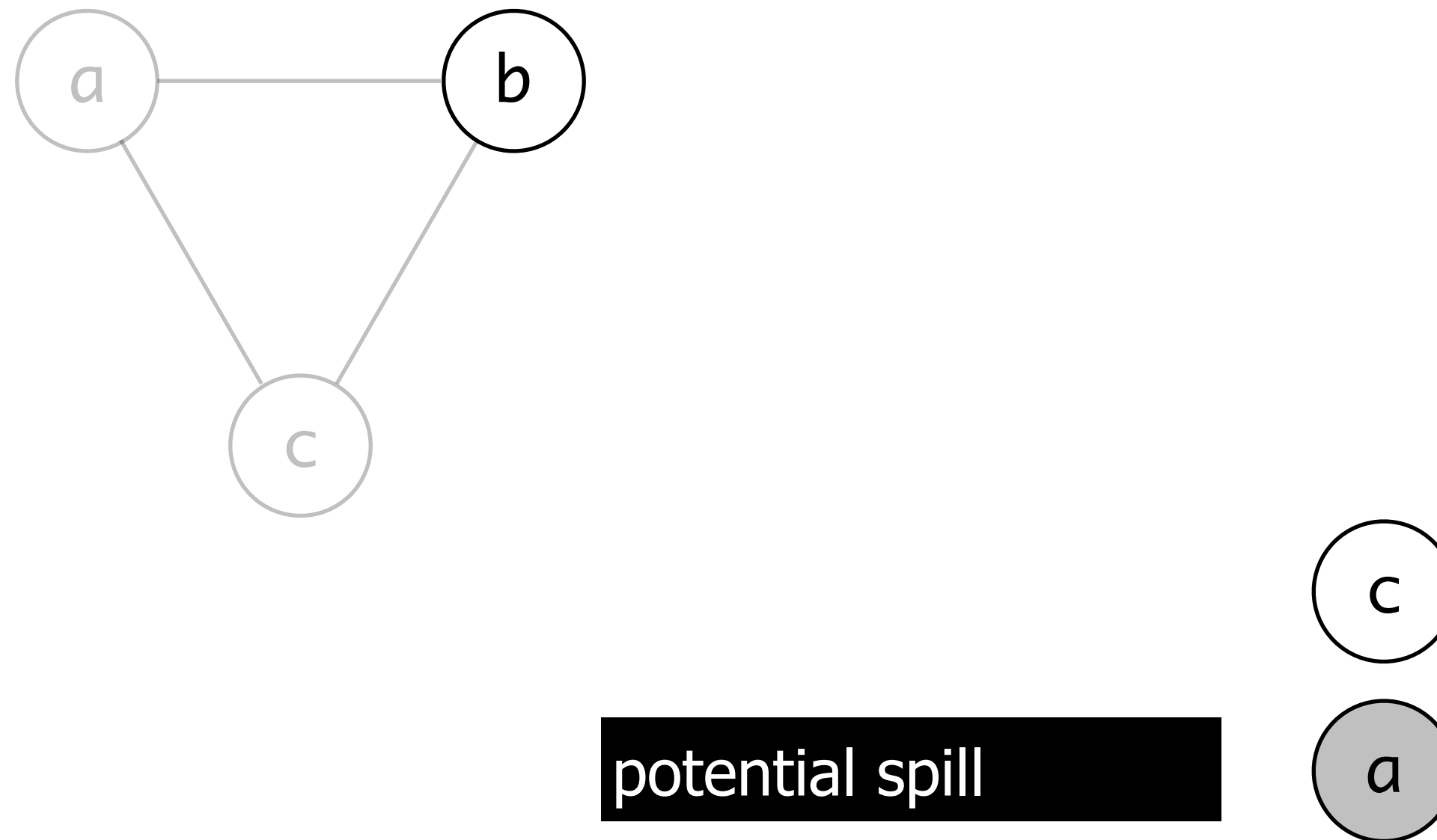
Spilling: example with 2 colors



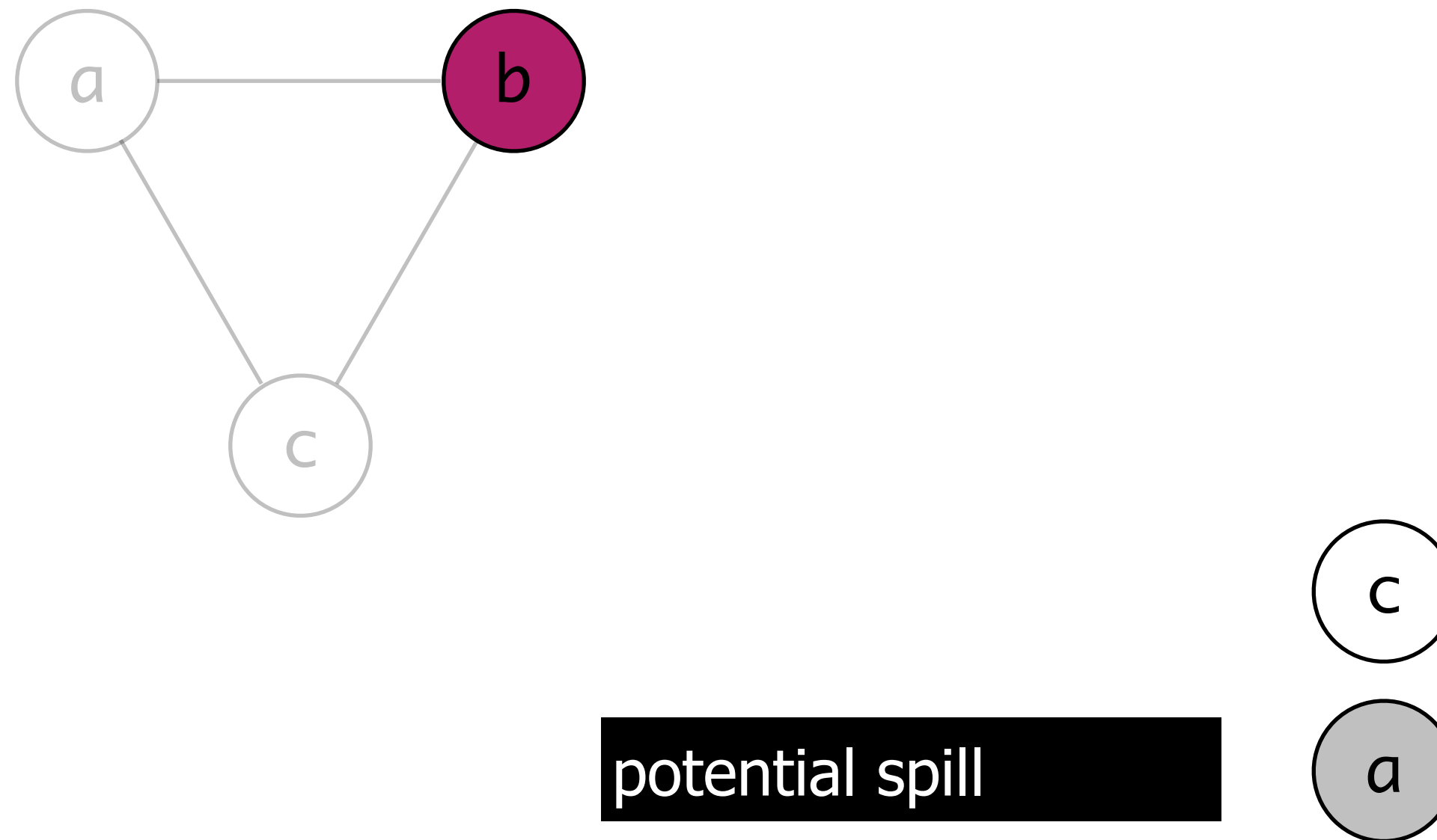
potential spill



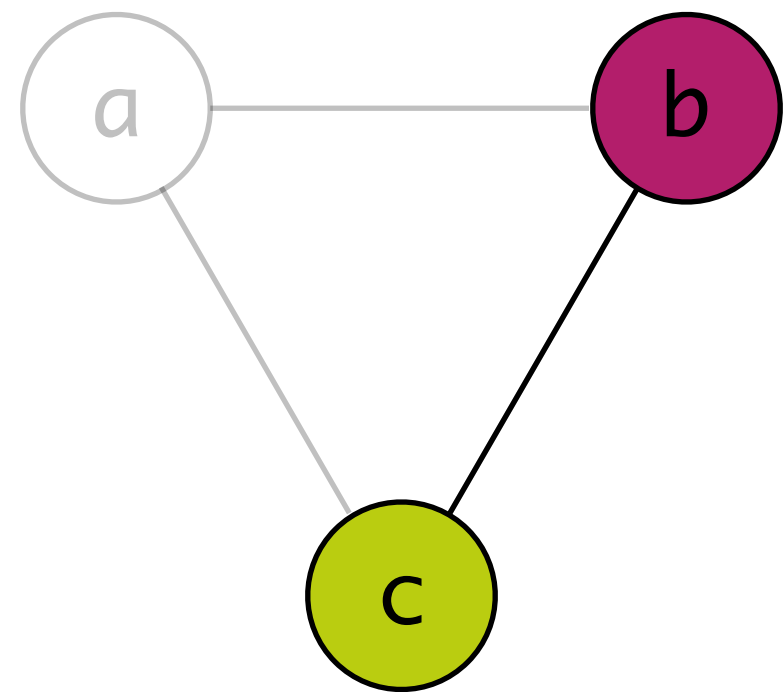
Spilling: example with 2 colors



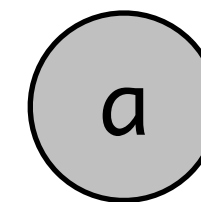
Spilling: example with 2 colors



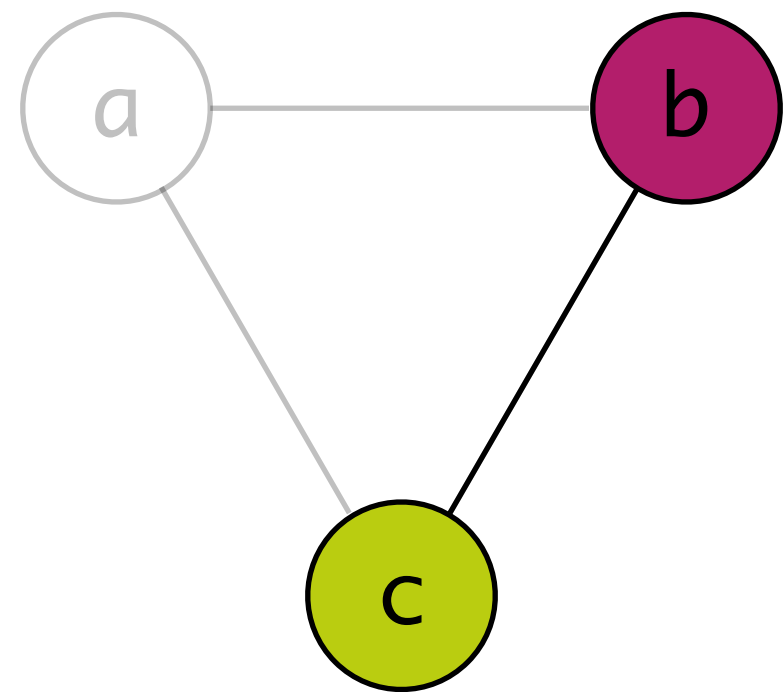
Spilling: example with 2 colors



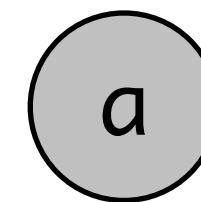
potential spill



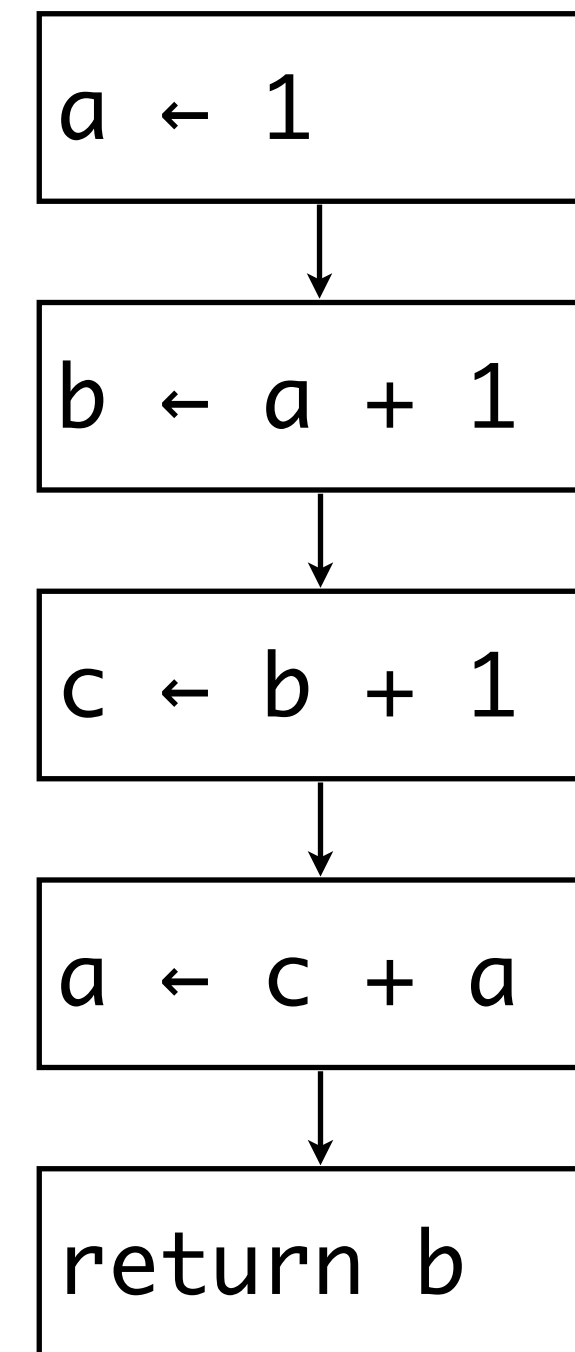
Spilling: example with 2 colors



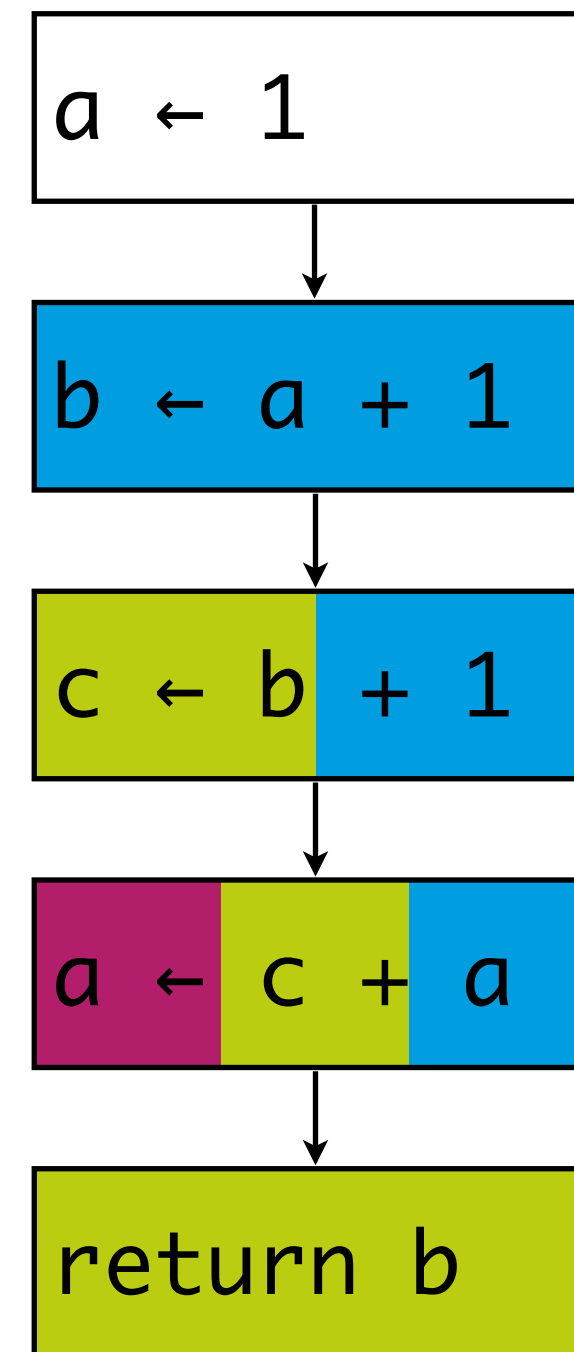
actual spill



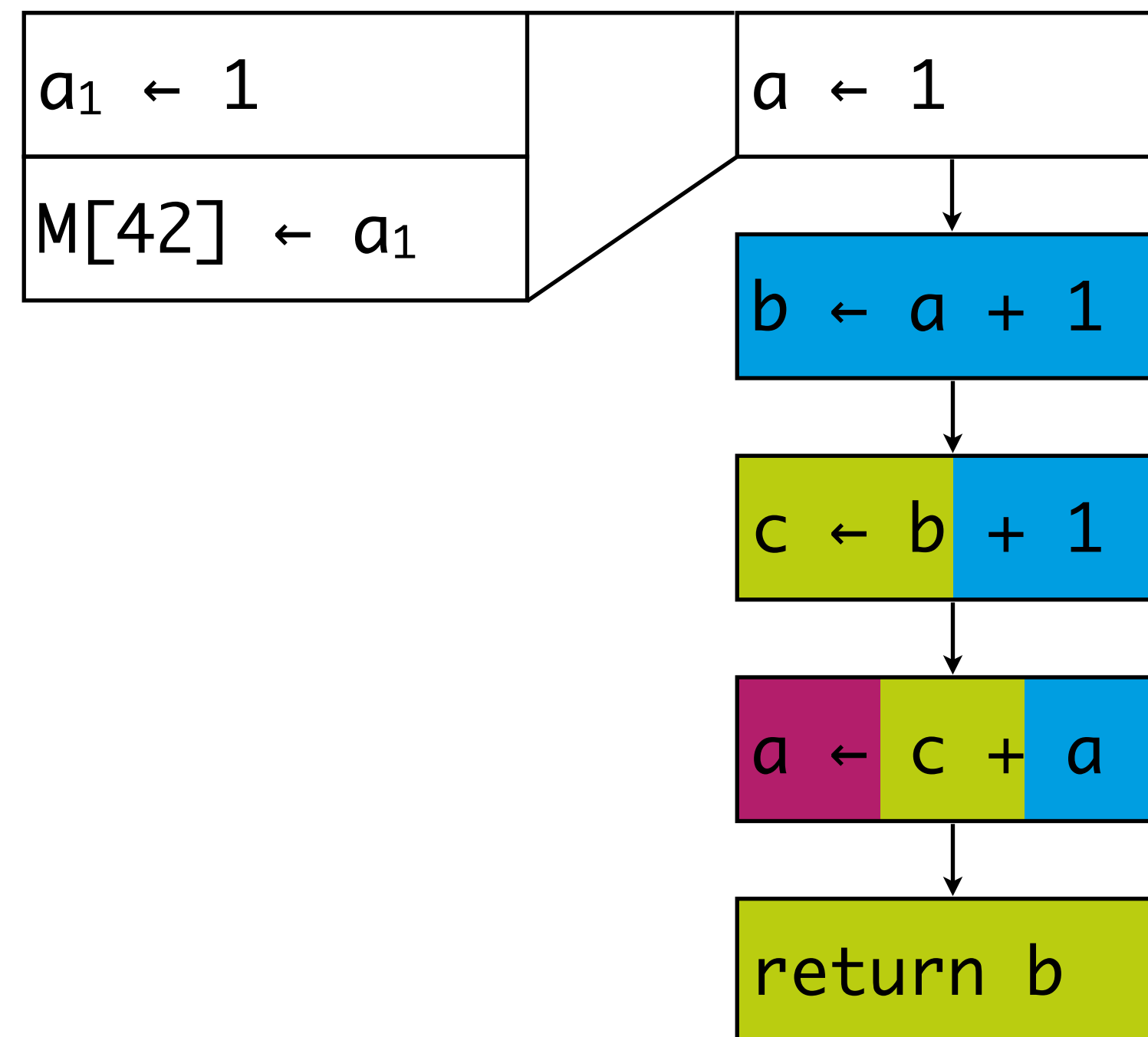
Spilling: Example



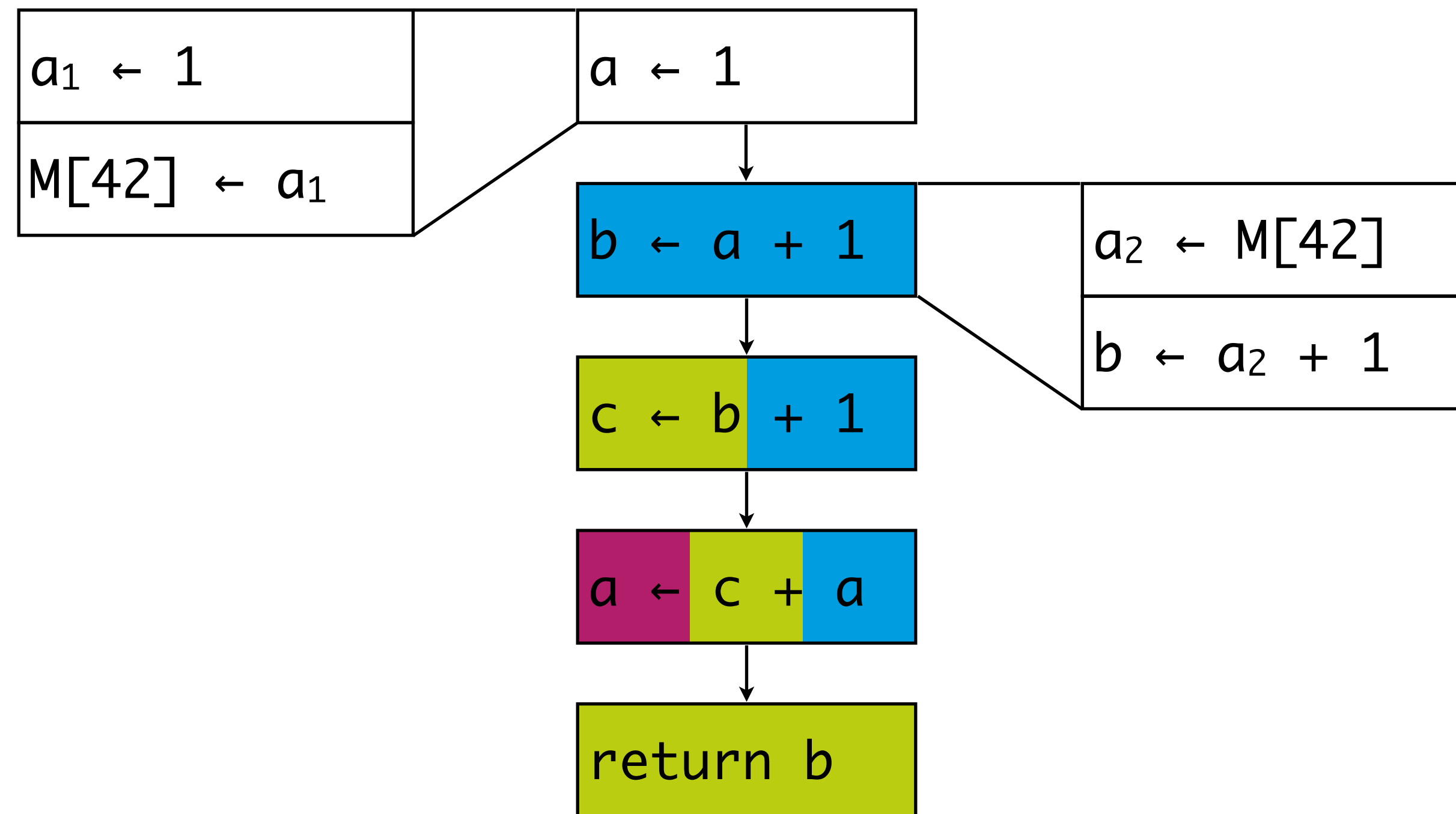
Spilling: Example



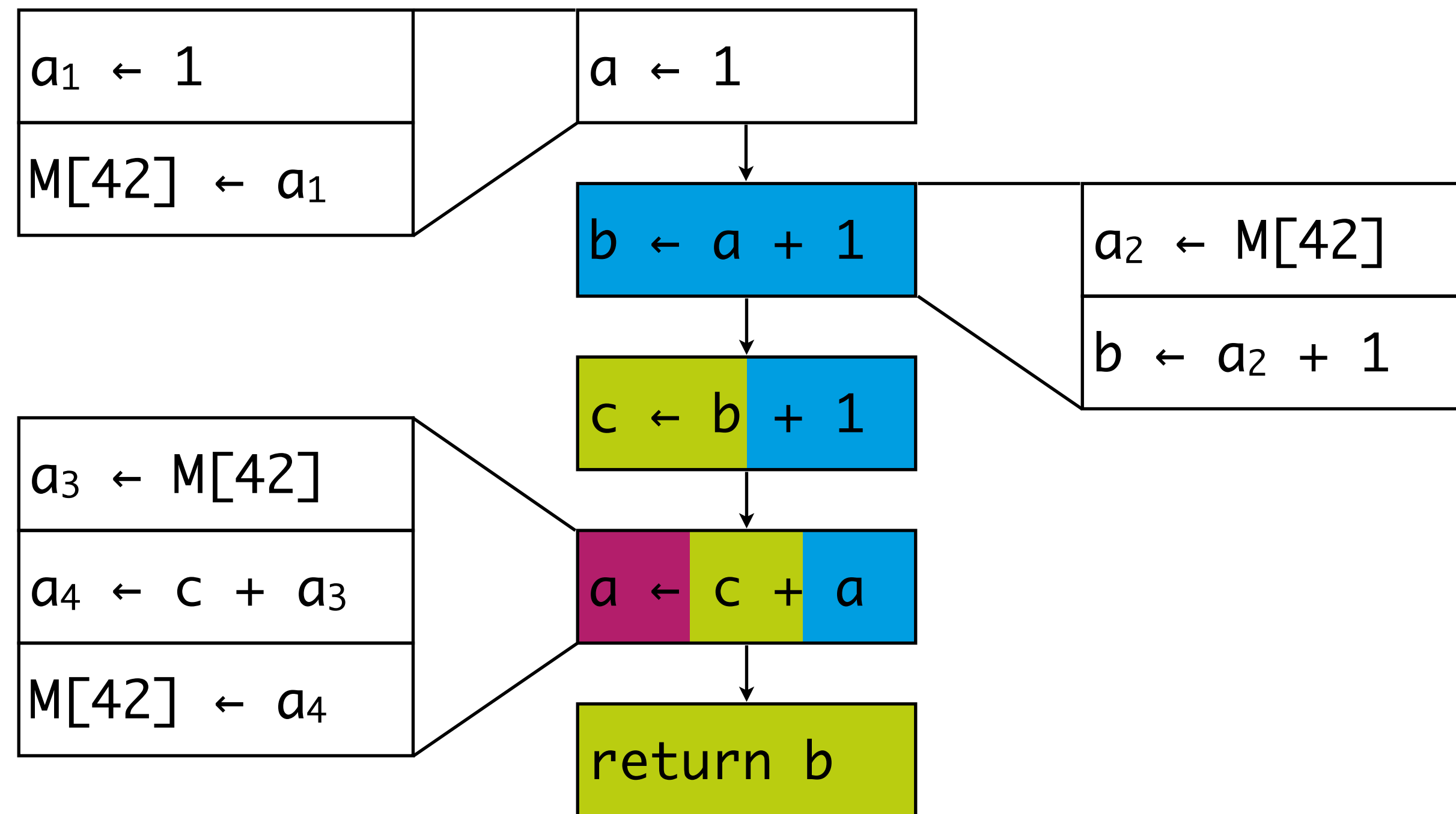
Spilling: Example



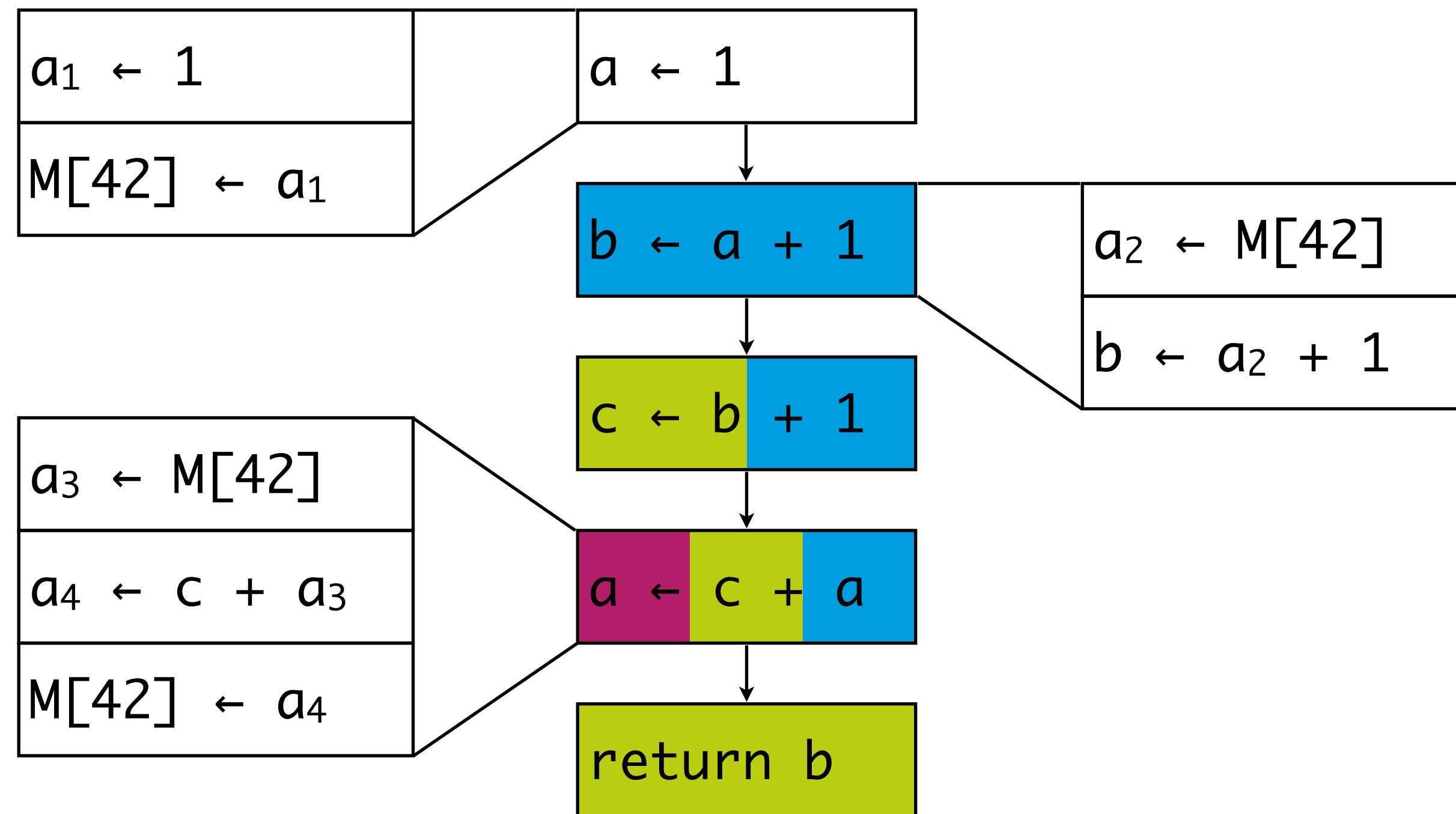
Spilling: Example



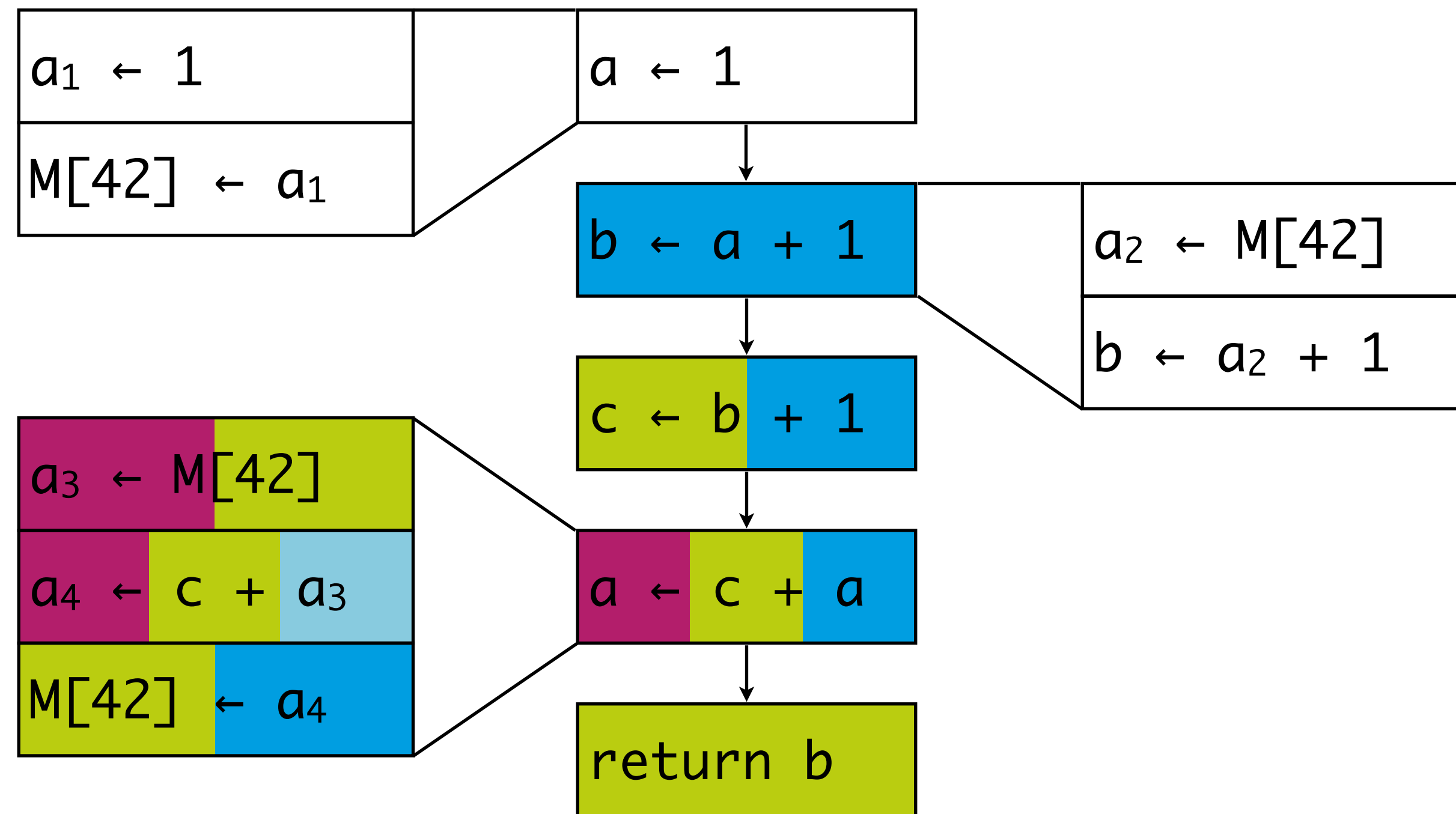
Spilling: Example



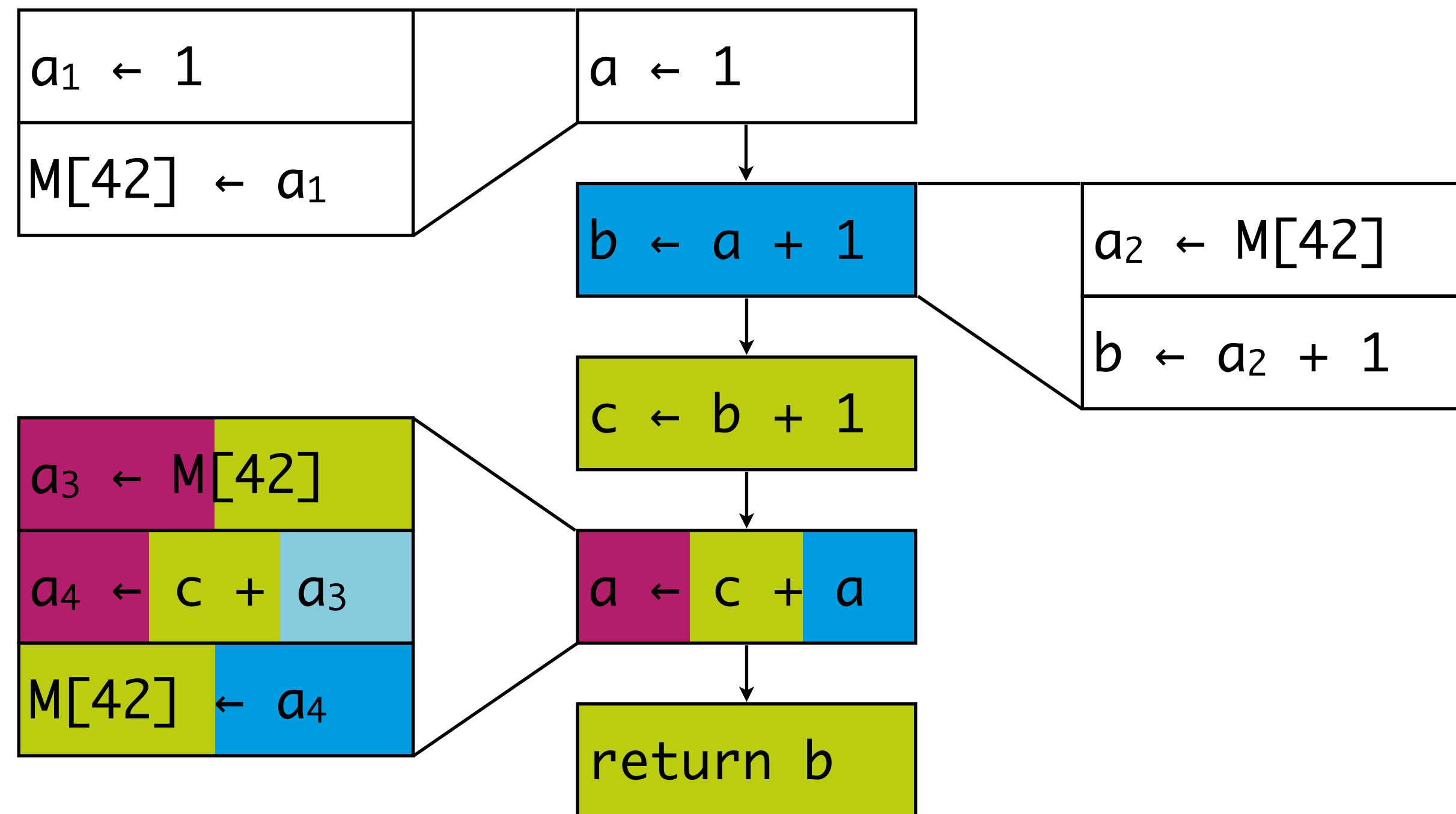
Spilling: Example



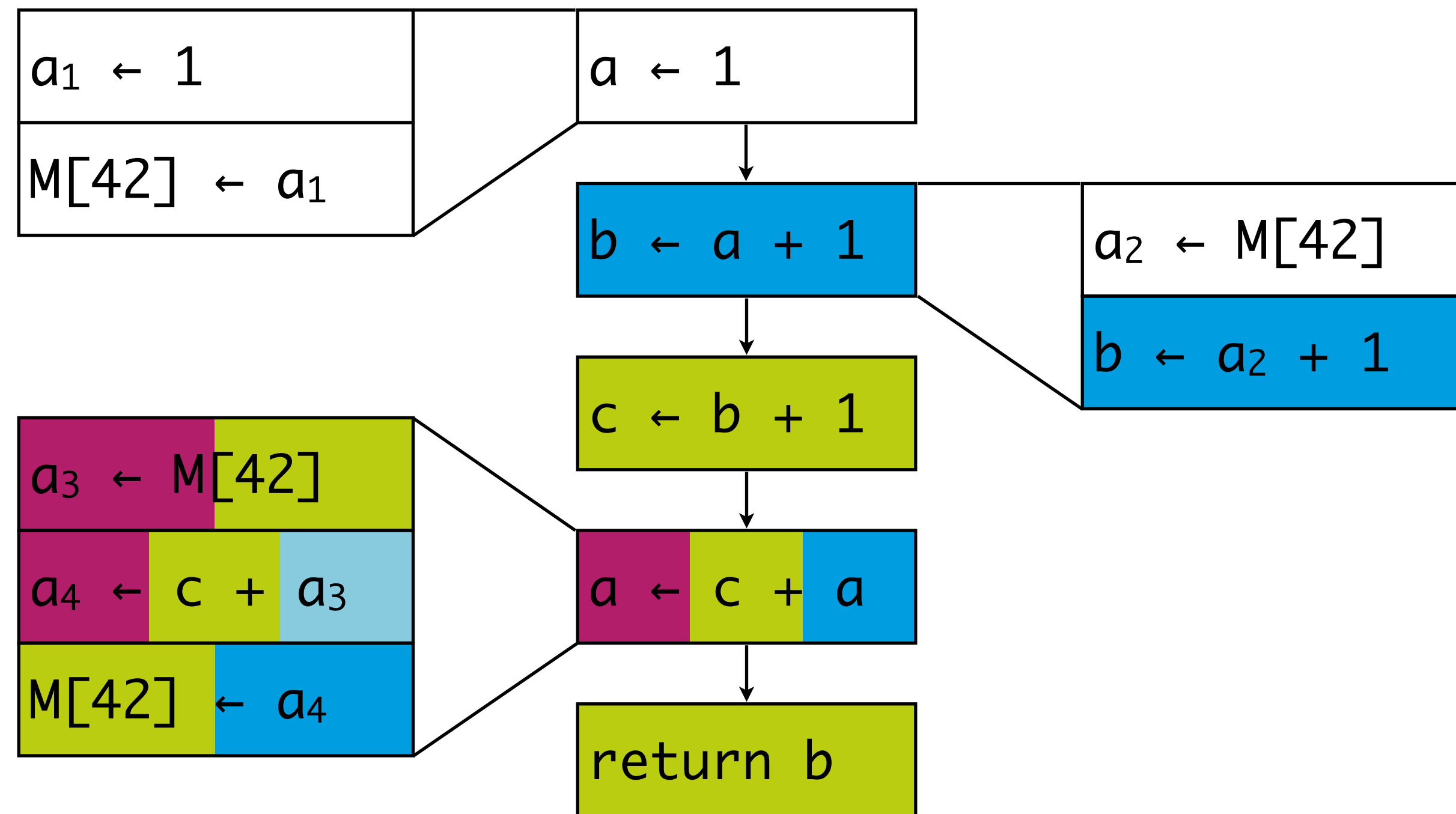
Spilling: Example



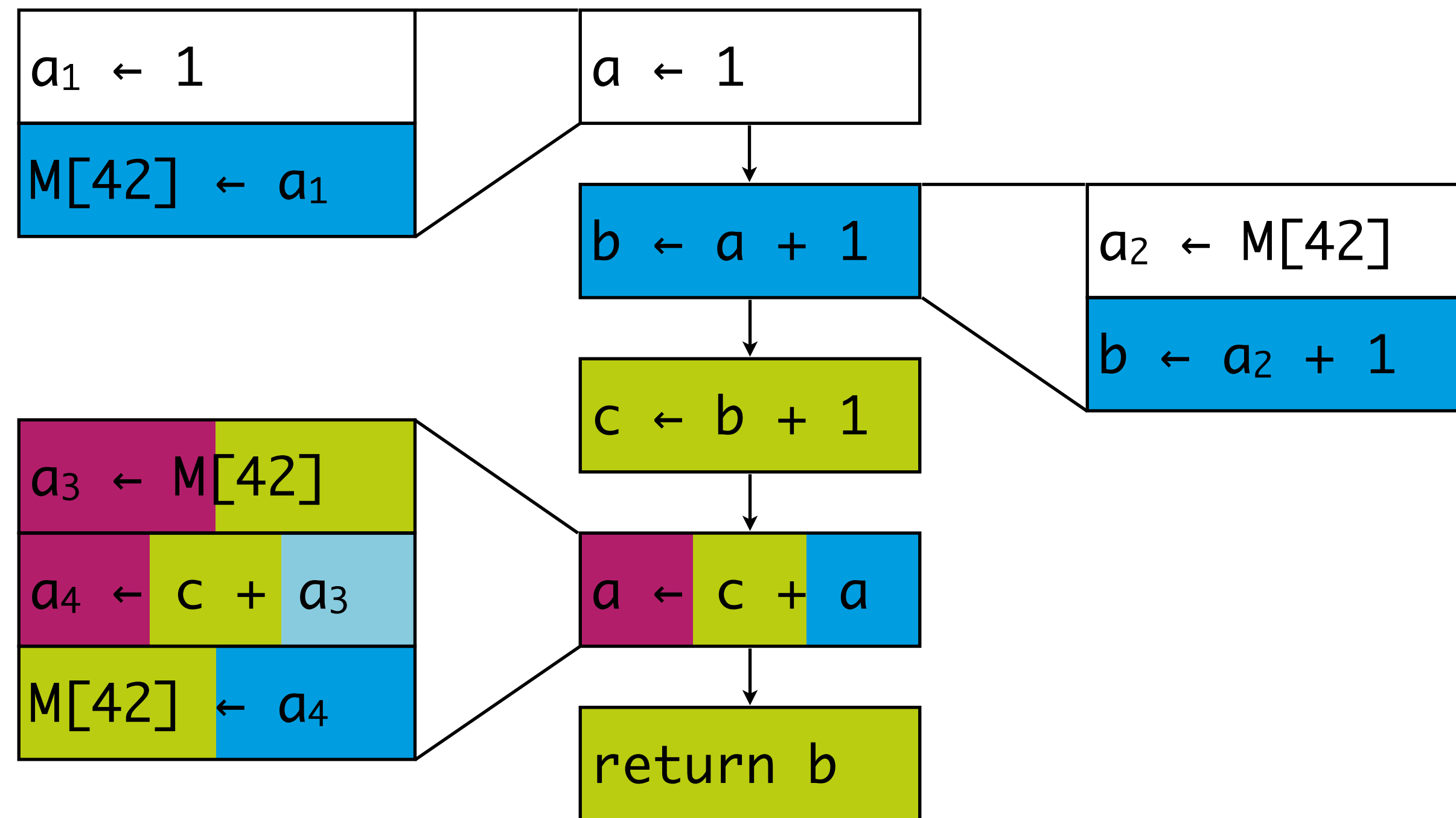
Spilling: Example



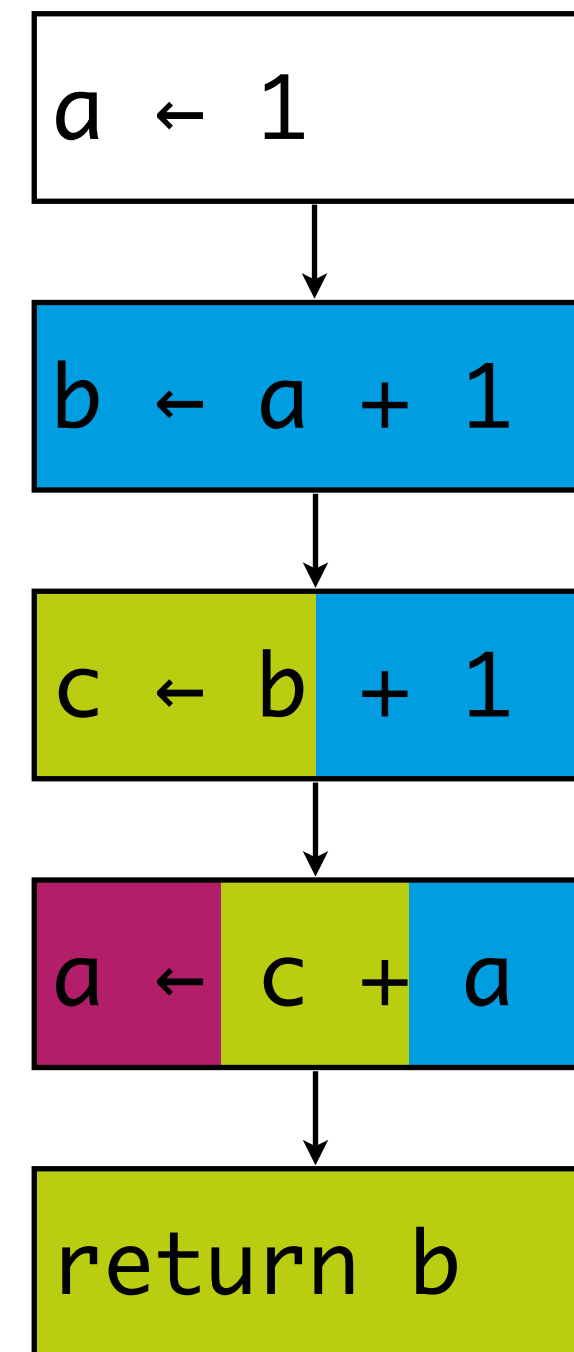
Spilling: Example



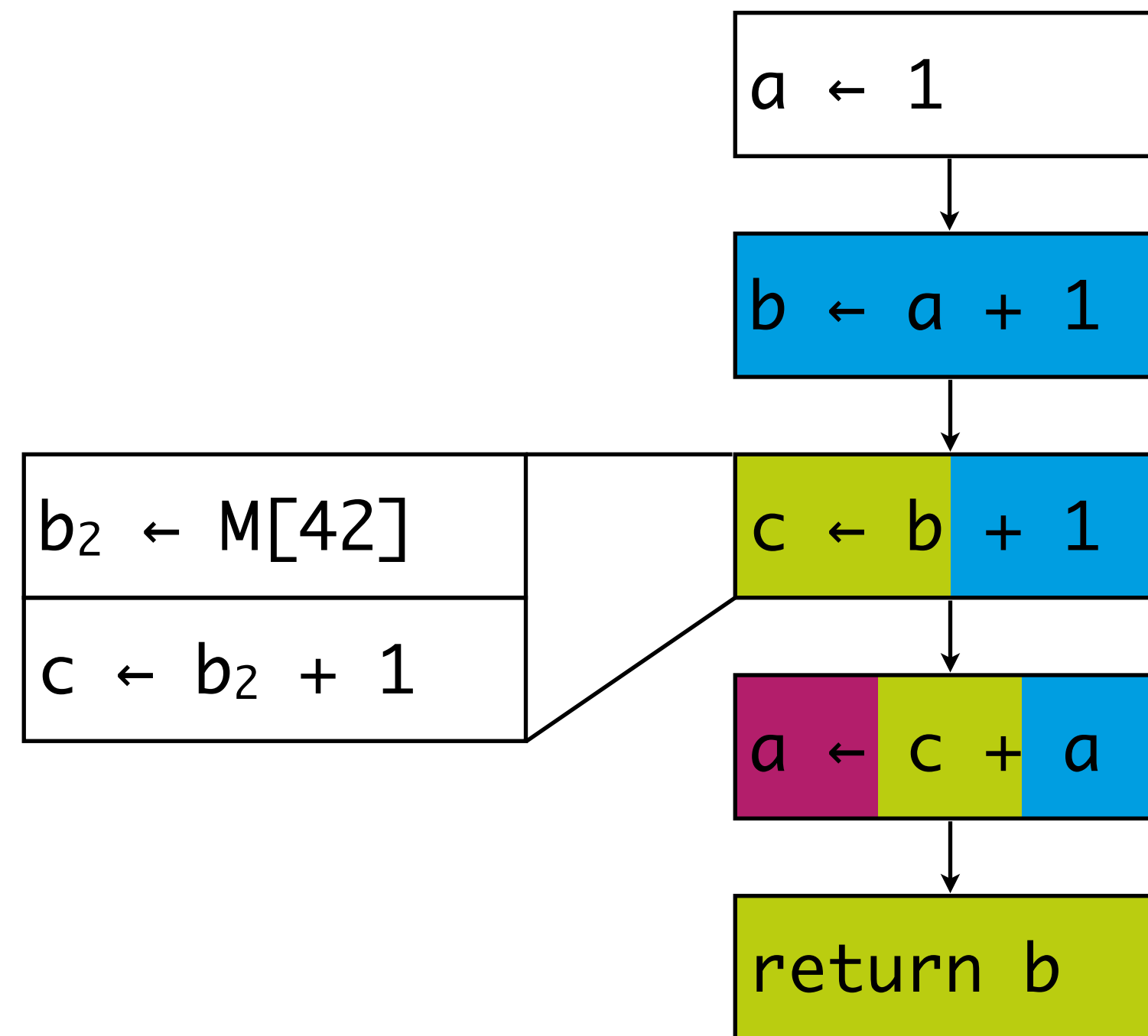
Spilling: Example



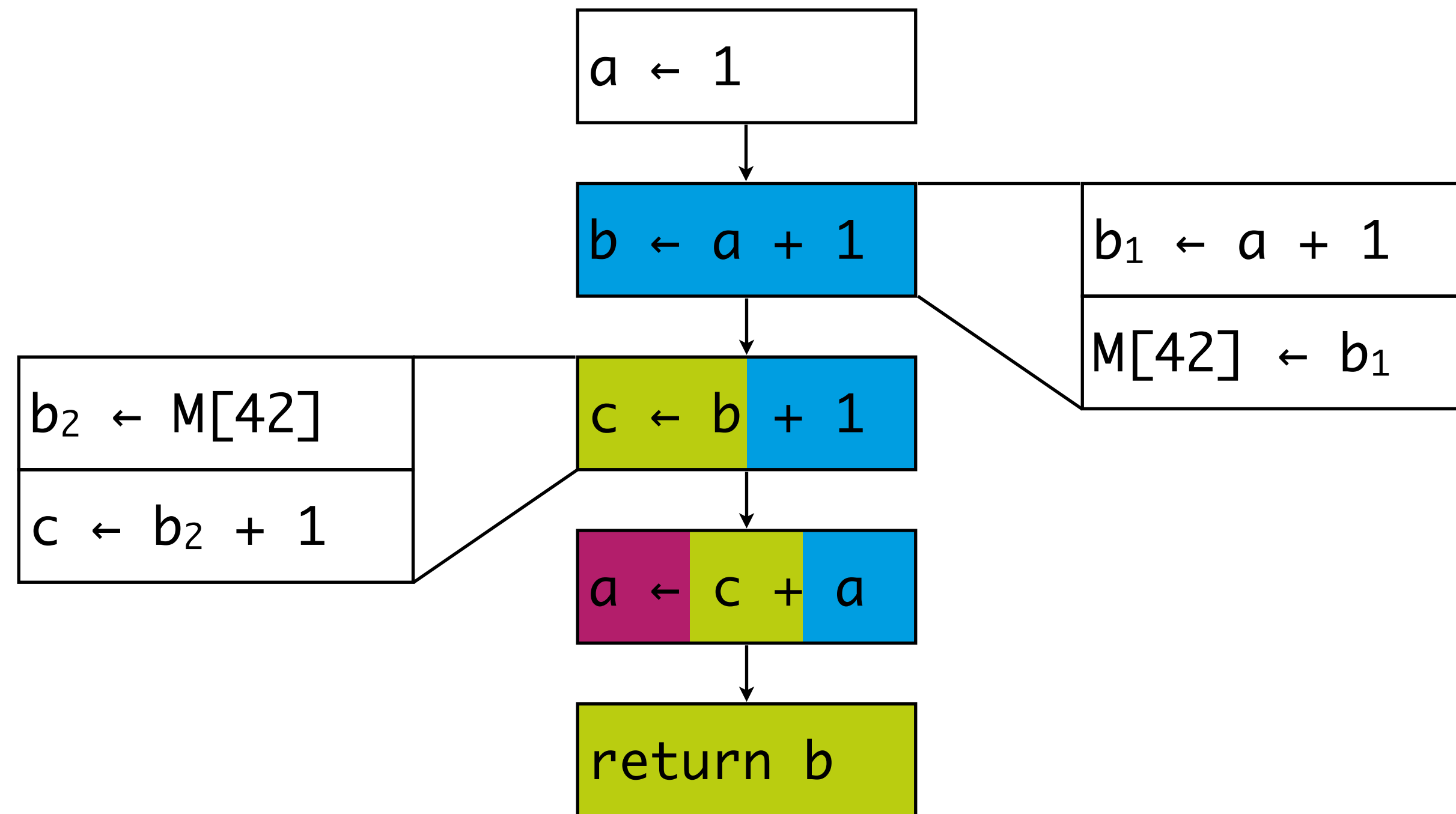
Spilling: Example



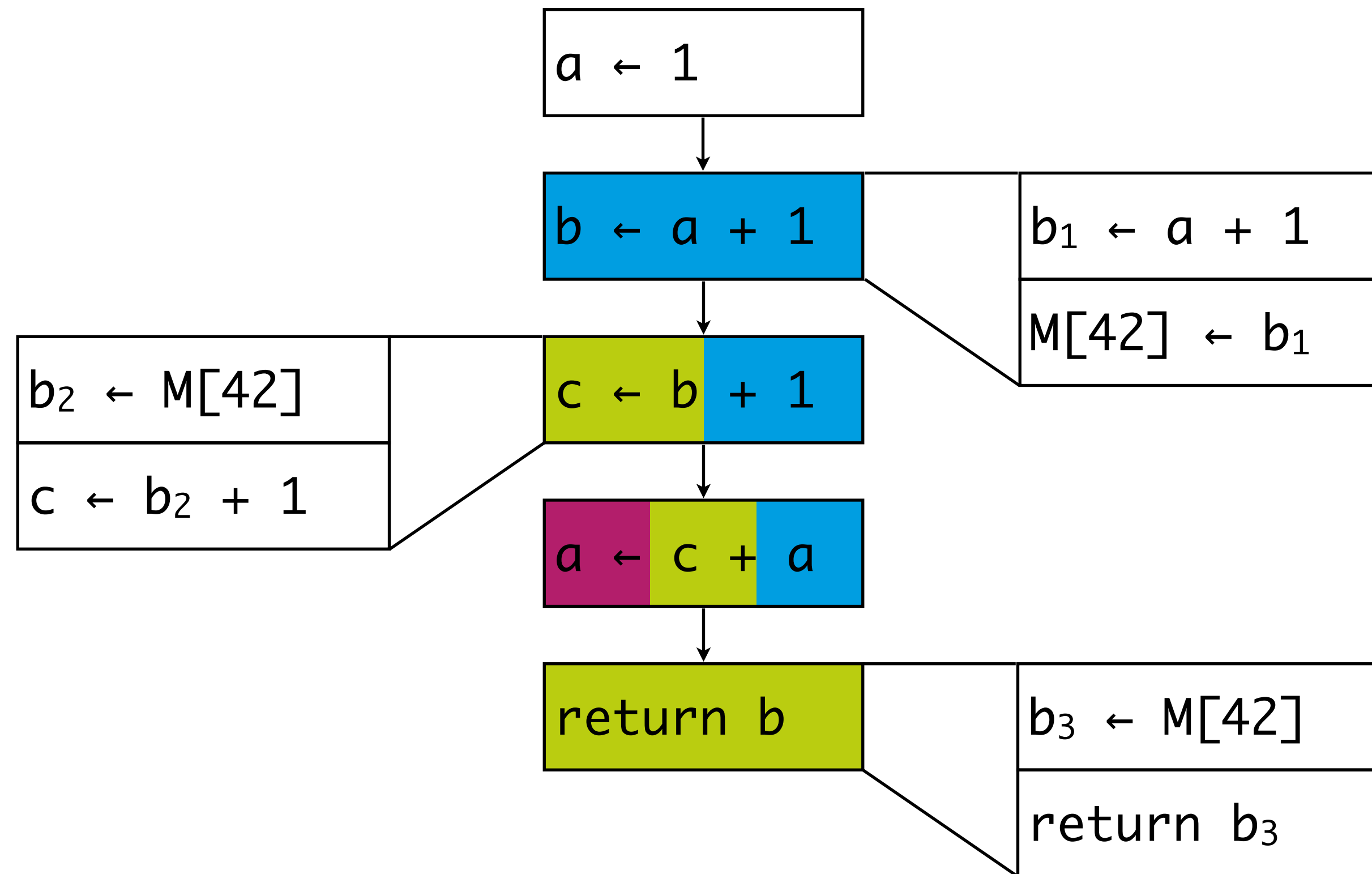
Spilling: Example



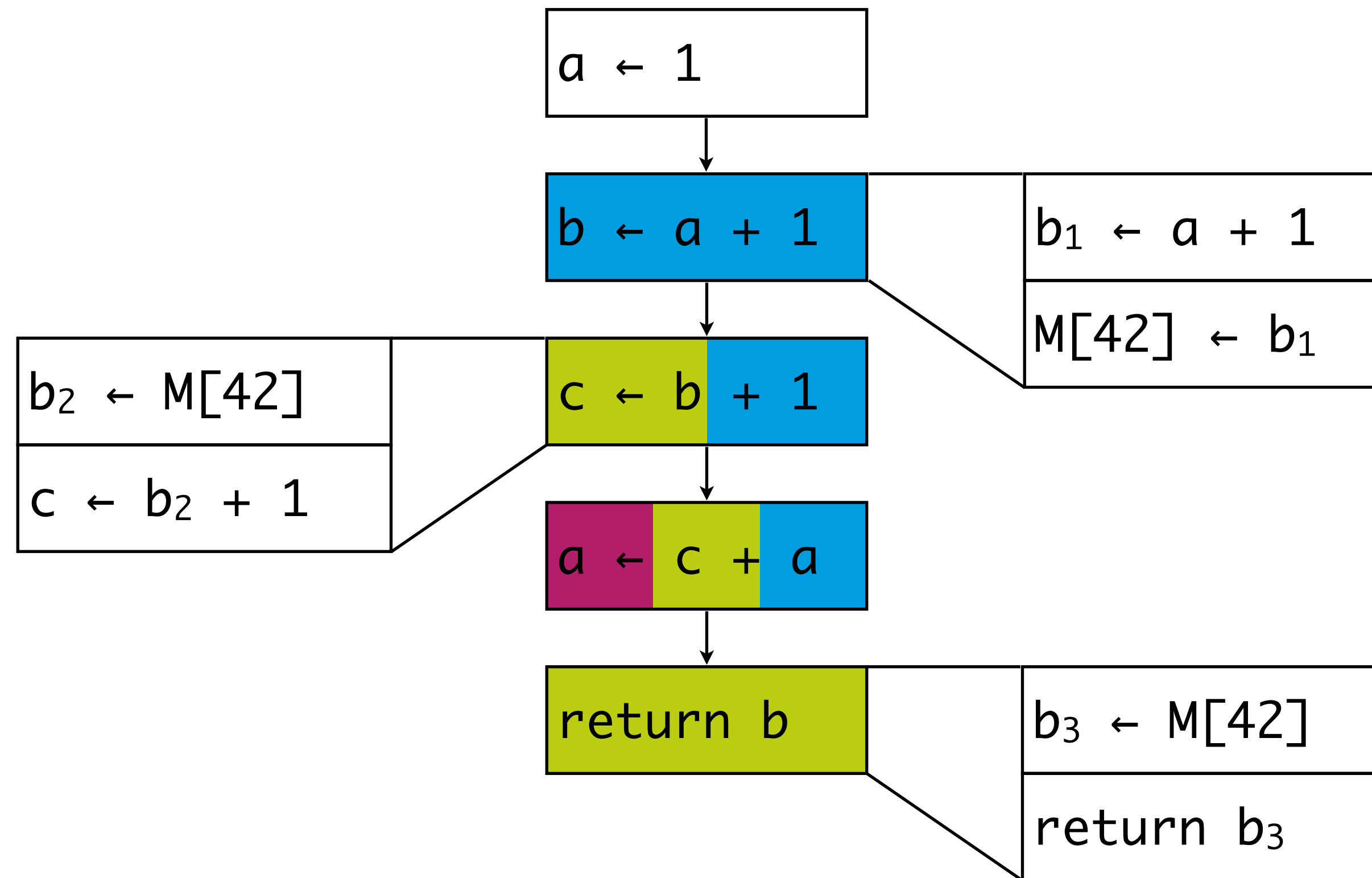
Spilling: Example



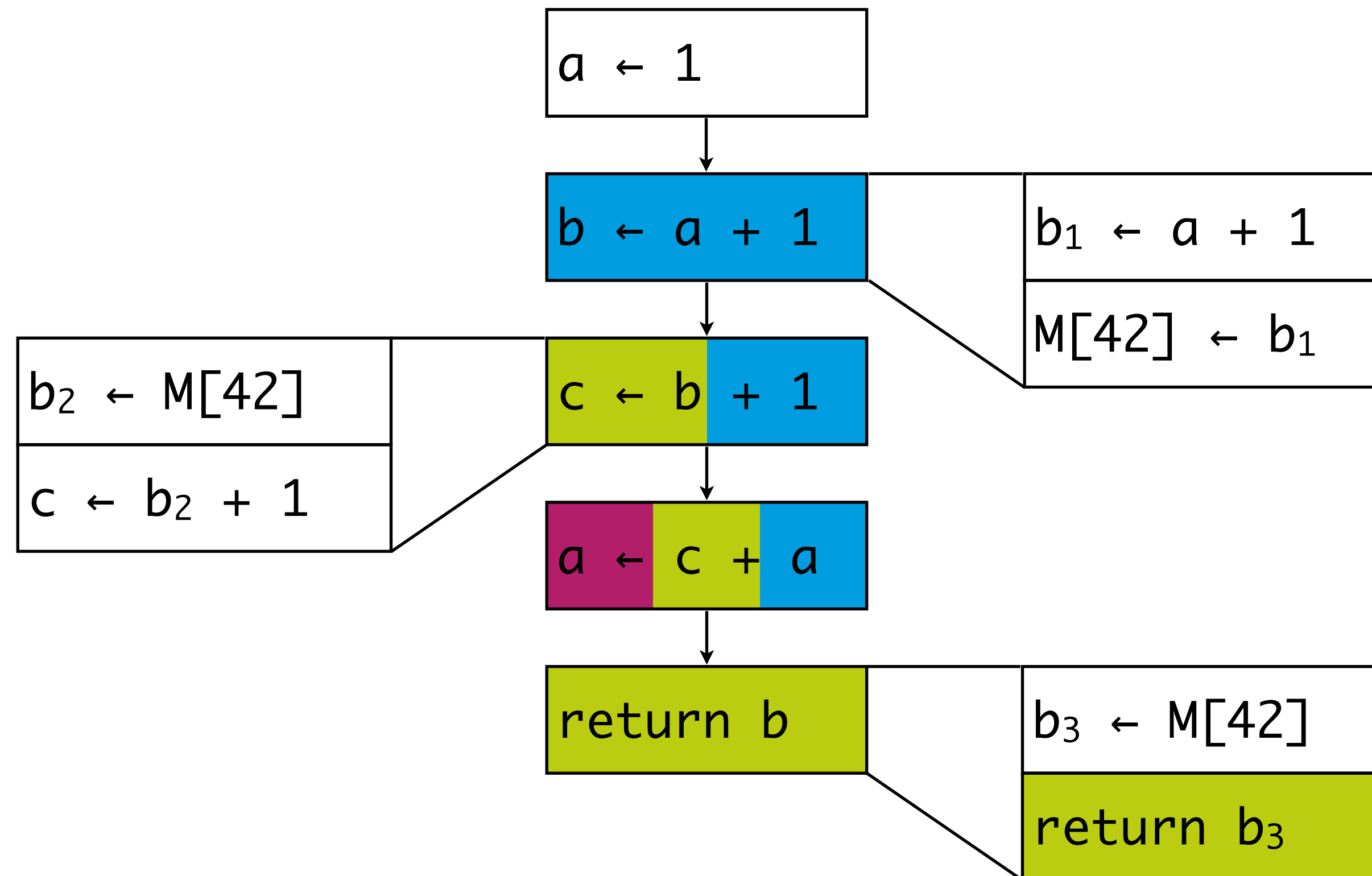
Spilling: Example



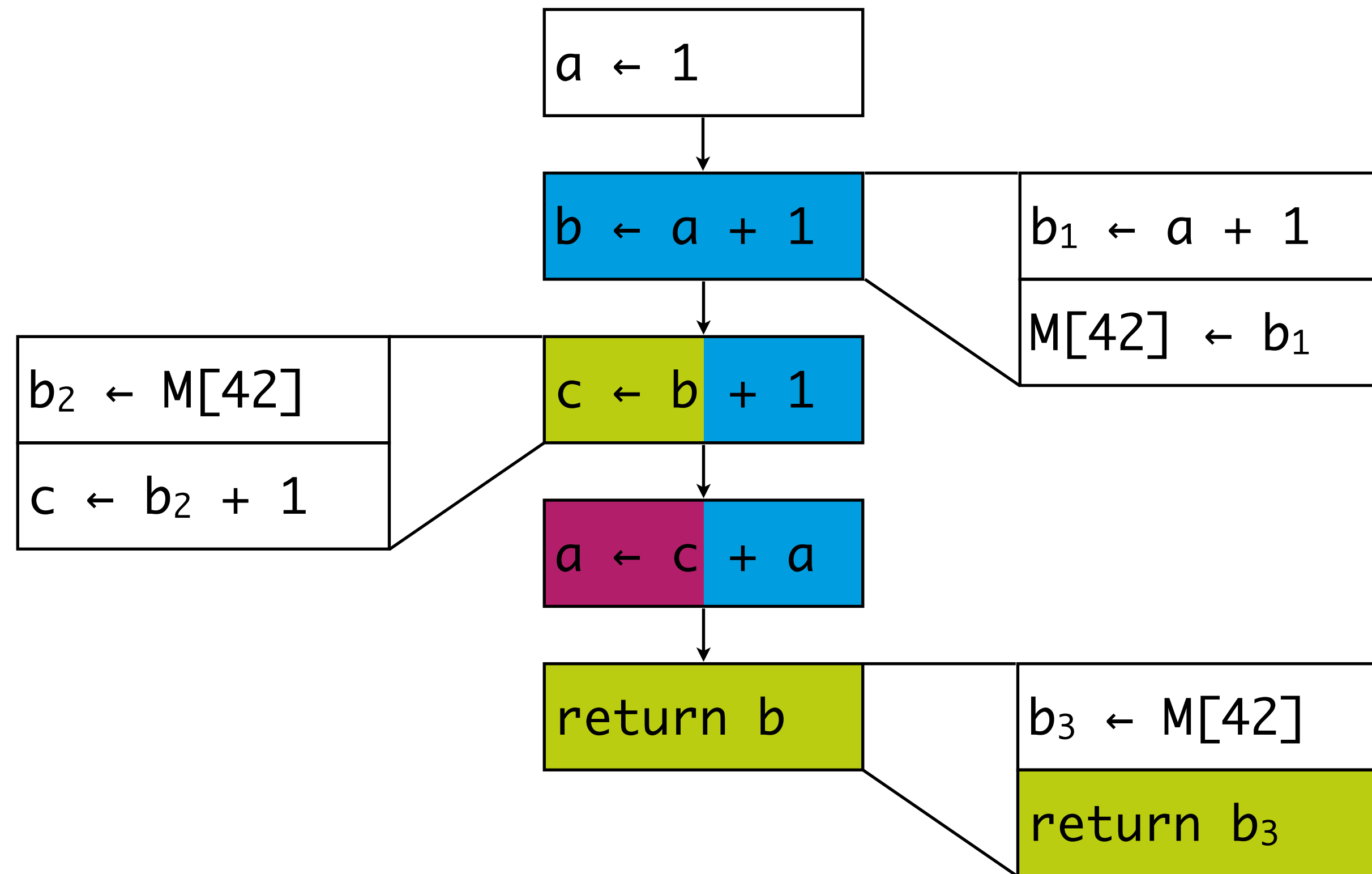
Spilling: Example



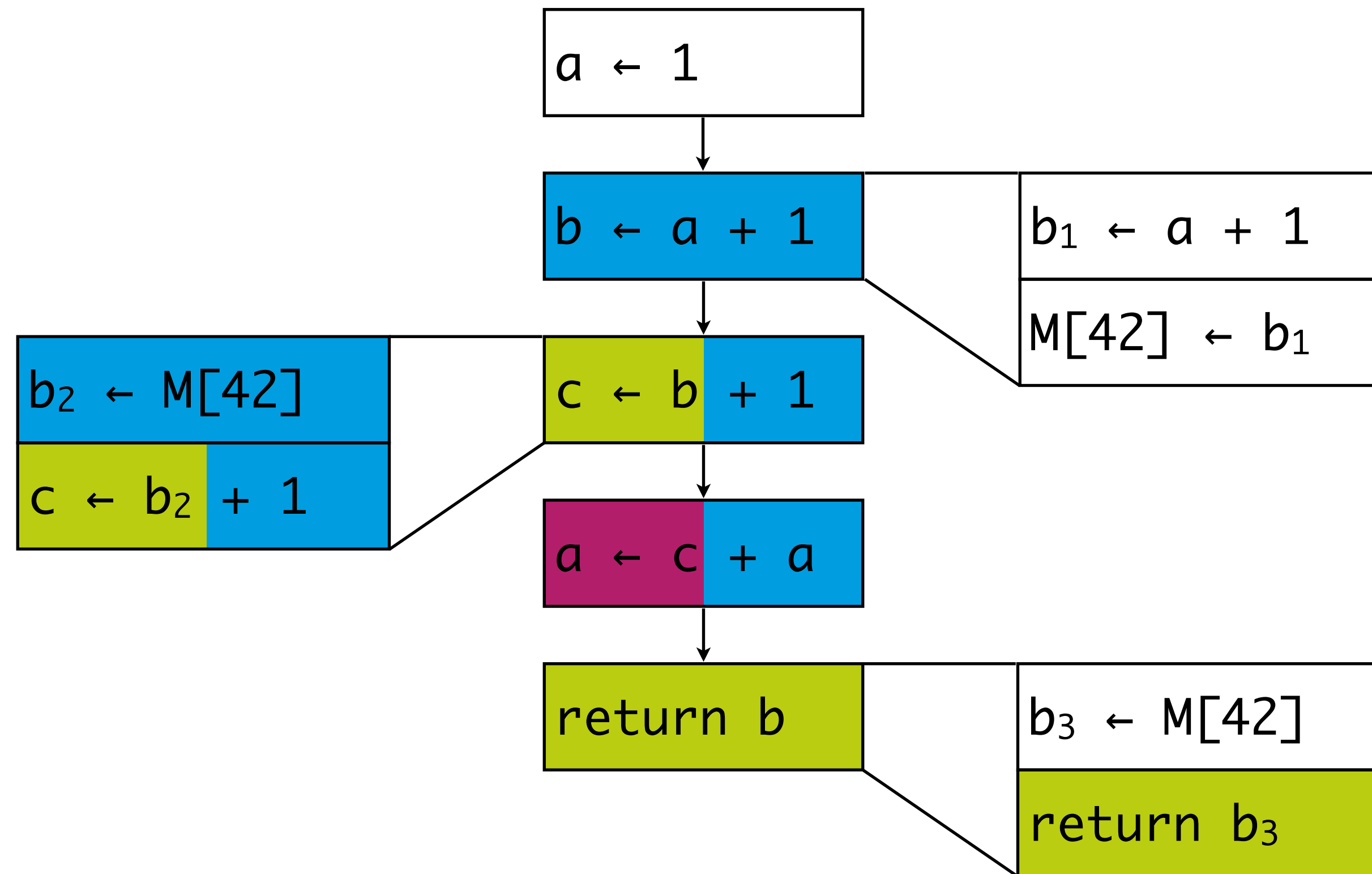
Spilling: Example



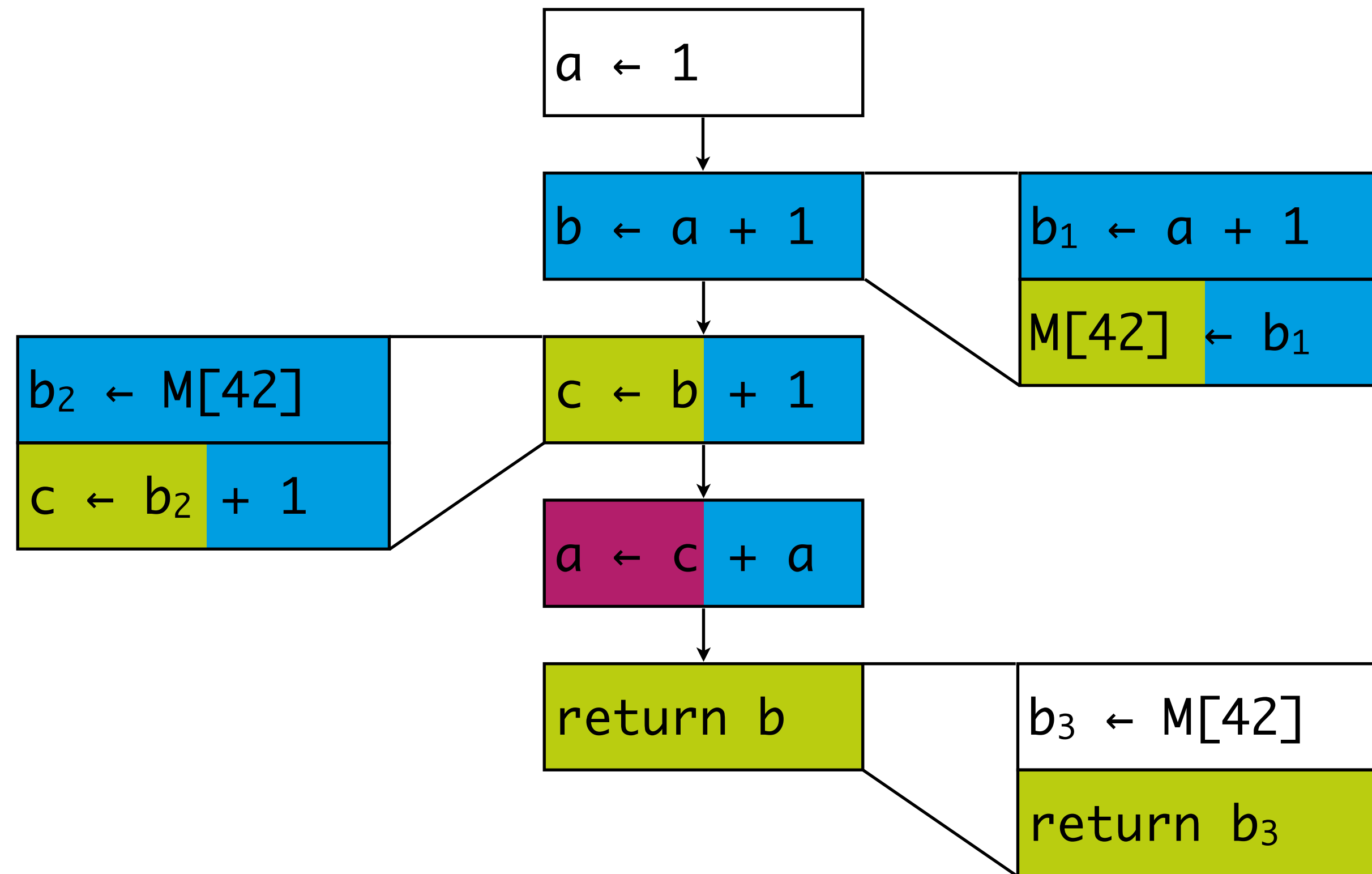
Spilling: Example



Spilling: Example



Spilling: Example



Coalescing

Eliminating Move Instructions

```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Eliminating Move Instructions

```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Coalescing

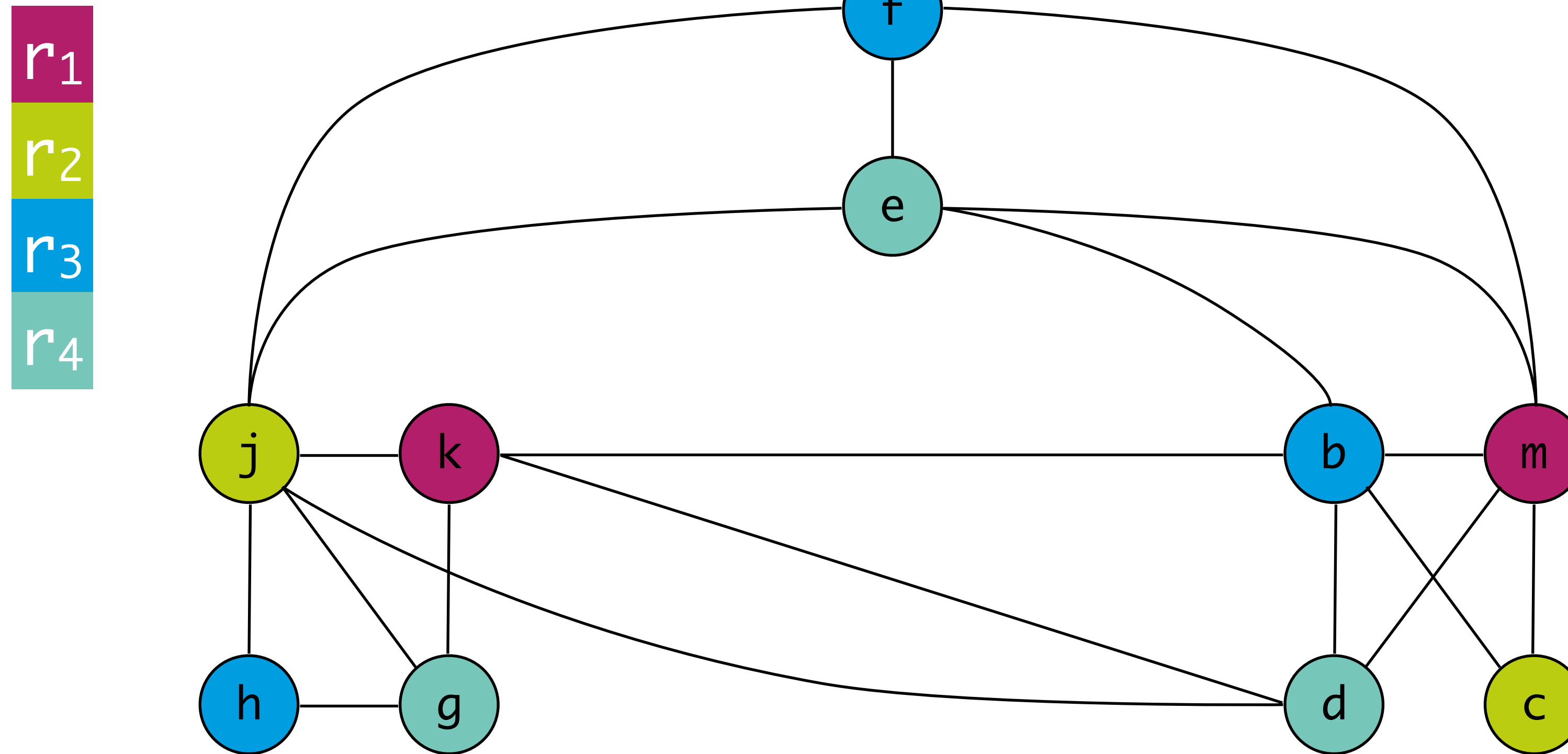
coalesce |ˌkəʊəˈleɪs|

verb *[no object]*

come together to form one mass or whole: *the puddles had **coalesced into** shallow streams.*

• *[with object]* combine (elements) in a mass or whole: *his idea served to **coalesce** all that happened **into** one connected whole.*

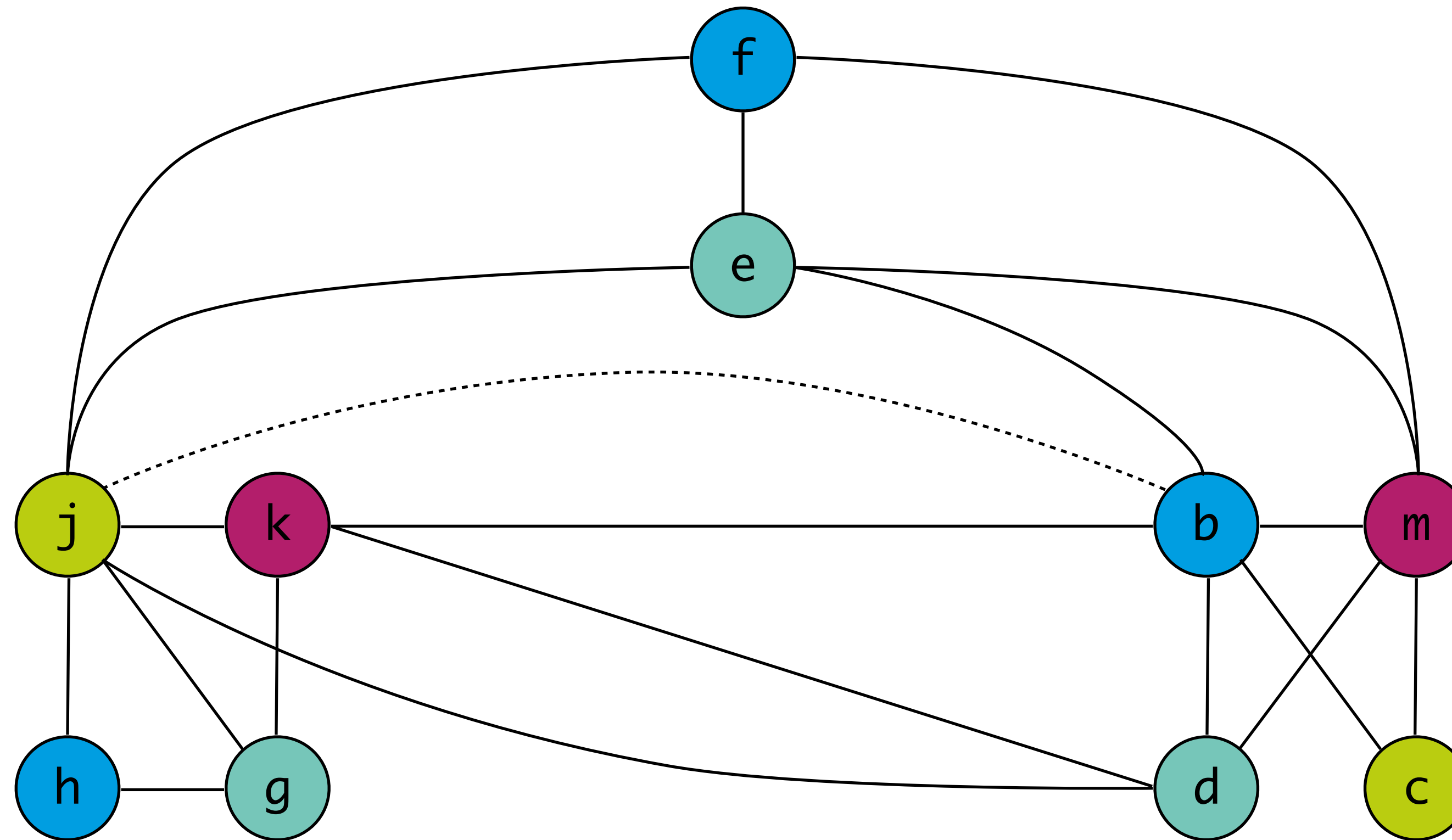
Recap: Graph Coloring



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Recap: Graph Coloring

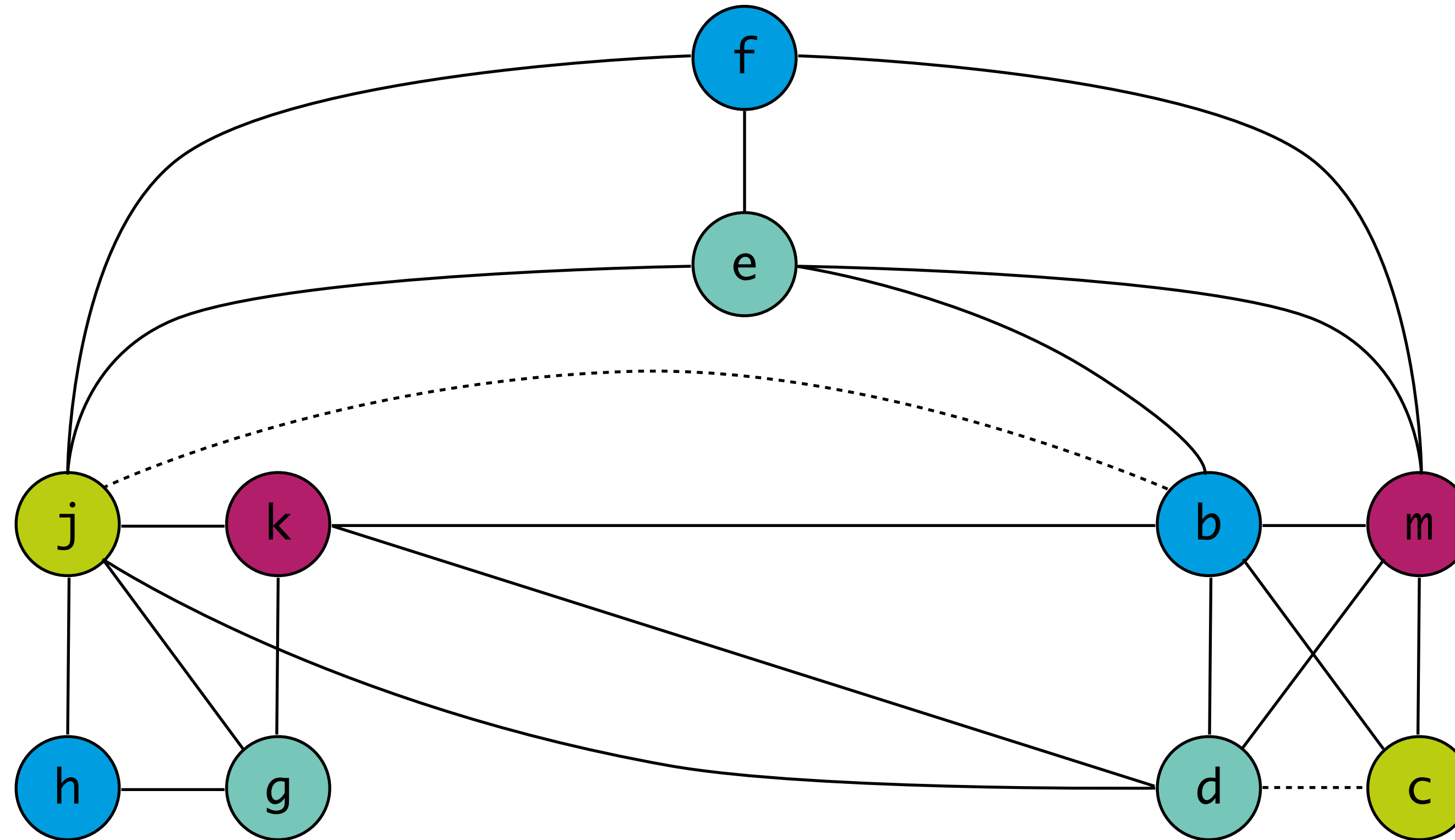
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Recap: Graph Coloring

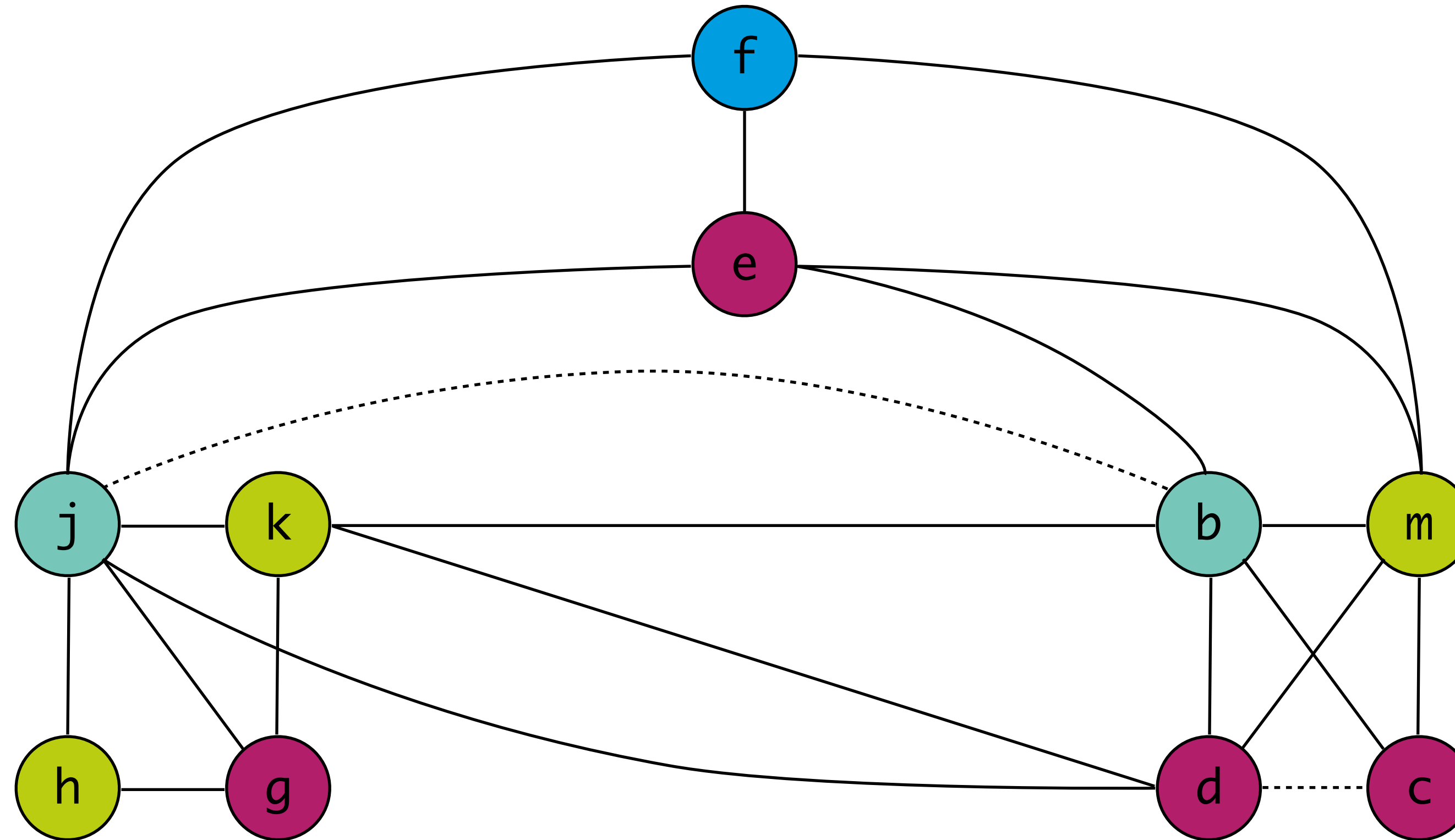
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Coalescing: better solution

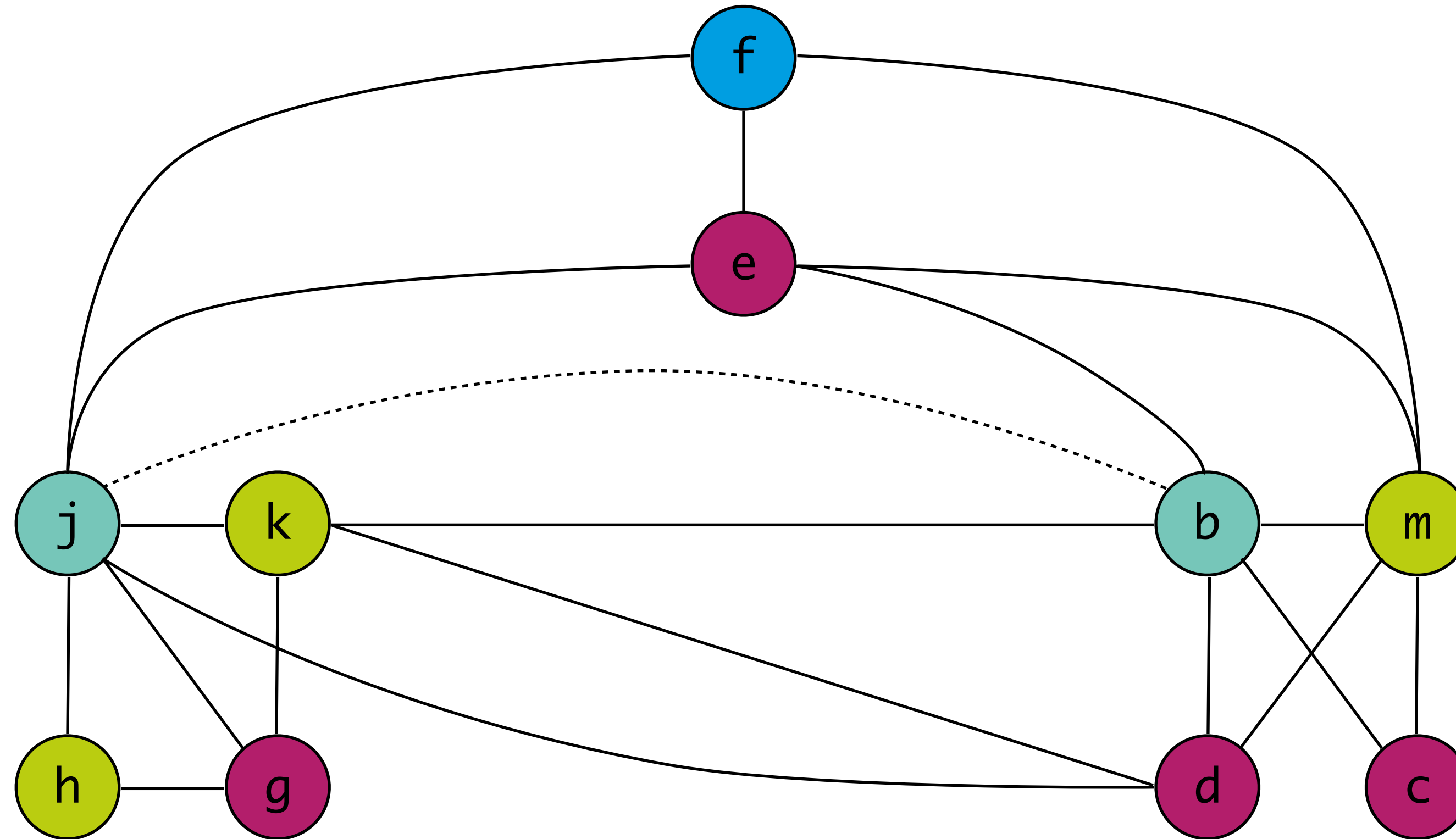
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```


Coalescing: better solution

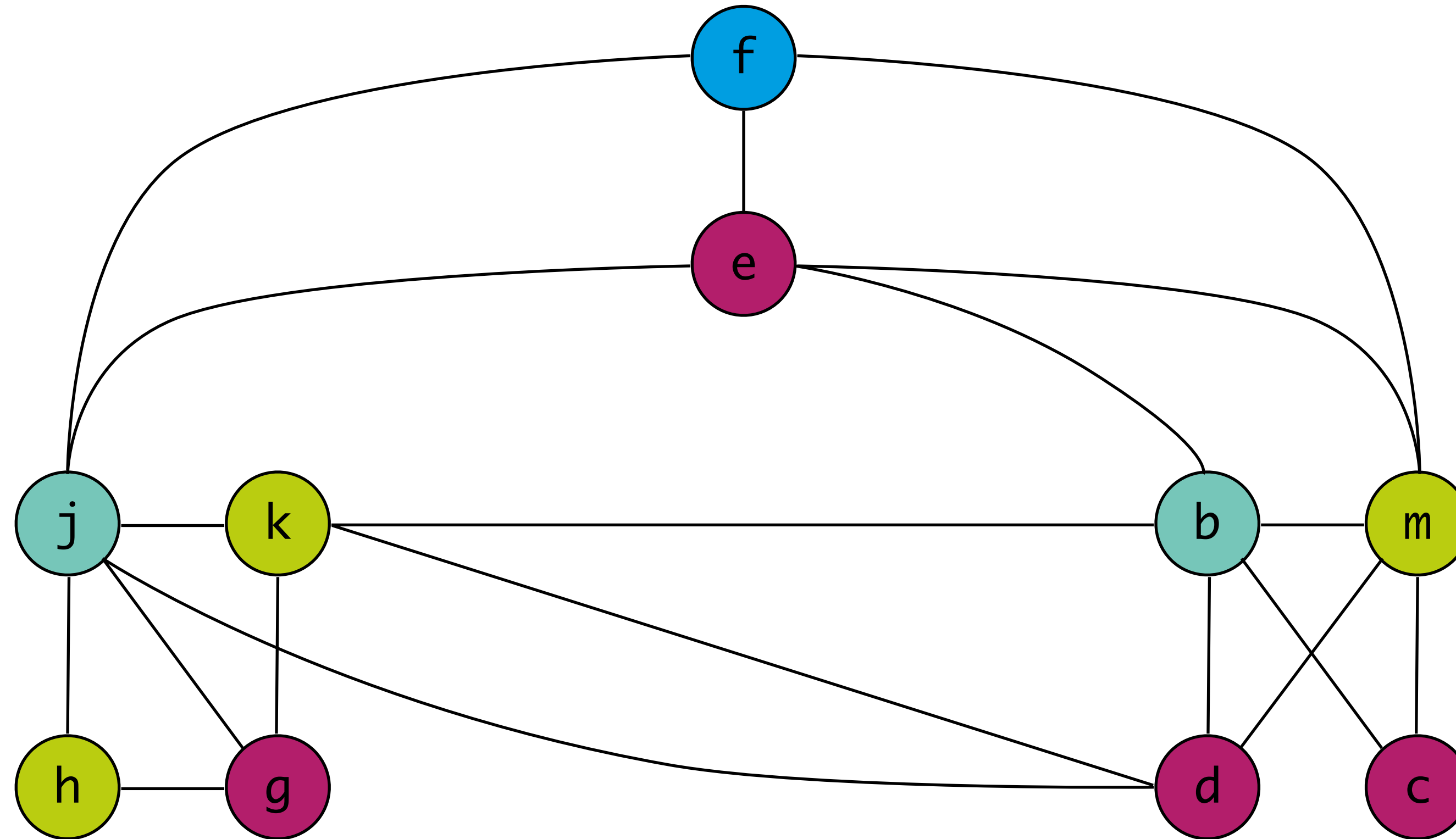
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Coalescing: better solution

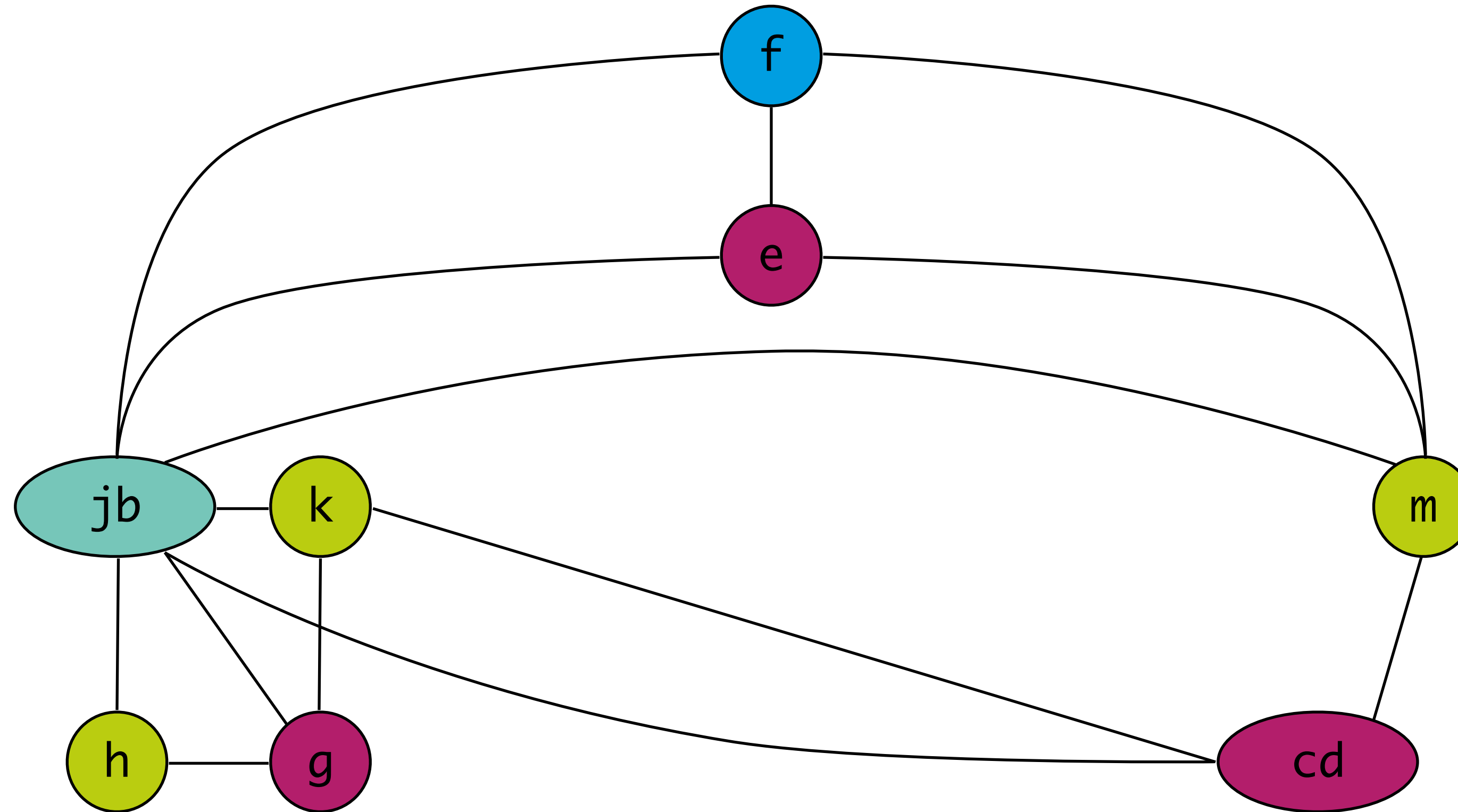
r_1
 r_2
 r_3
 r_4



```
live-in: k j  
g := mem[j + 12]  
h := k - 1  
f := g * h  
e := mem[j + 8]  
m := mem[j + 16]  
b := mem[f]  
c := e + 8  
d := c  
k := m + 4  
j := b  
live out: d k j
```

Coalescing: coalescing nodes

r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Coalescing: conservative strategies

Briggs

- a/b has fewer than k neighbours of significant degree
- nodes of insignificant degree and a/b can be simplified
- remaining graph is colorable

George

- all neighbours of a of significant degree interfere also with b
- neighbours of a of insignificant degree can be simplified
- subgraph of original graph is colorable

Graph Coloring: Steps

Simplify

- remove non-move-related node of insignificant degree

Coalesce

Freeze

- turn move-related node of insignificant degree into non-move-related

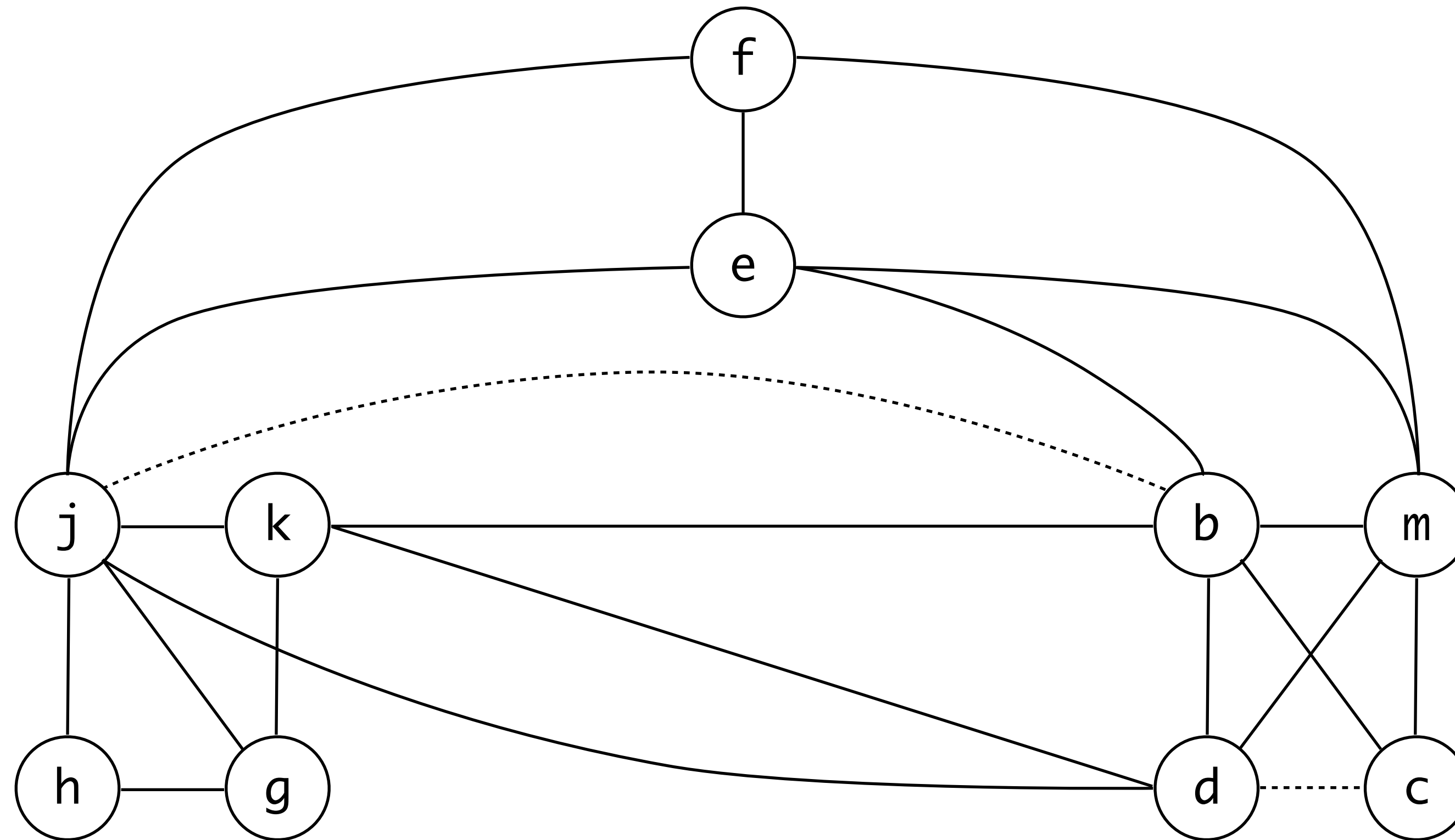
Spill

Select

Start over

Coalescing: example

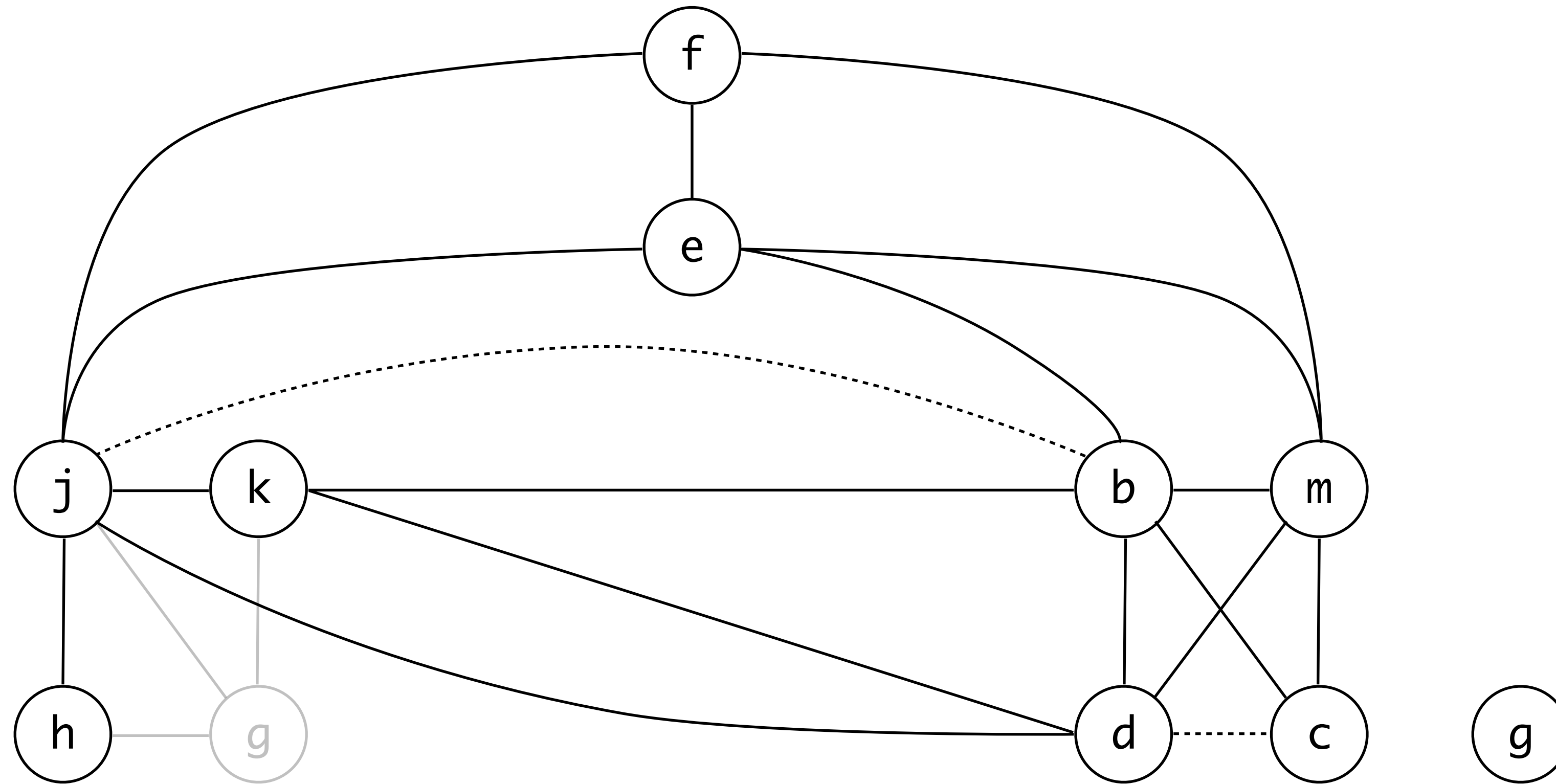
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Coalescing

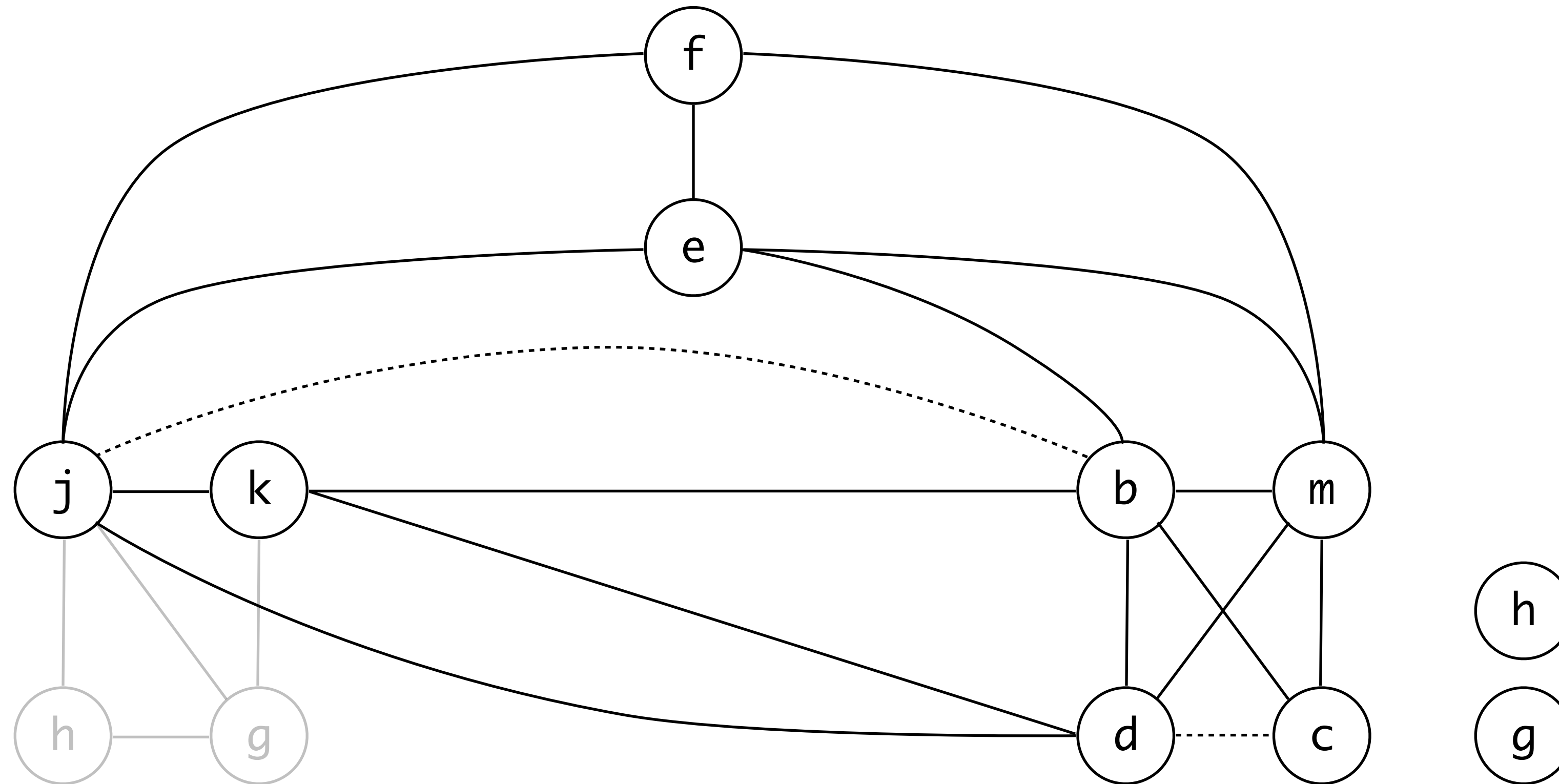
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Coalescing

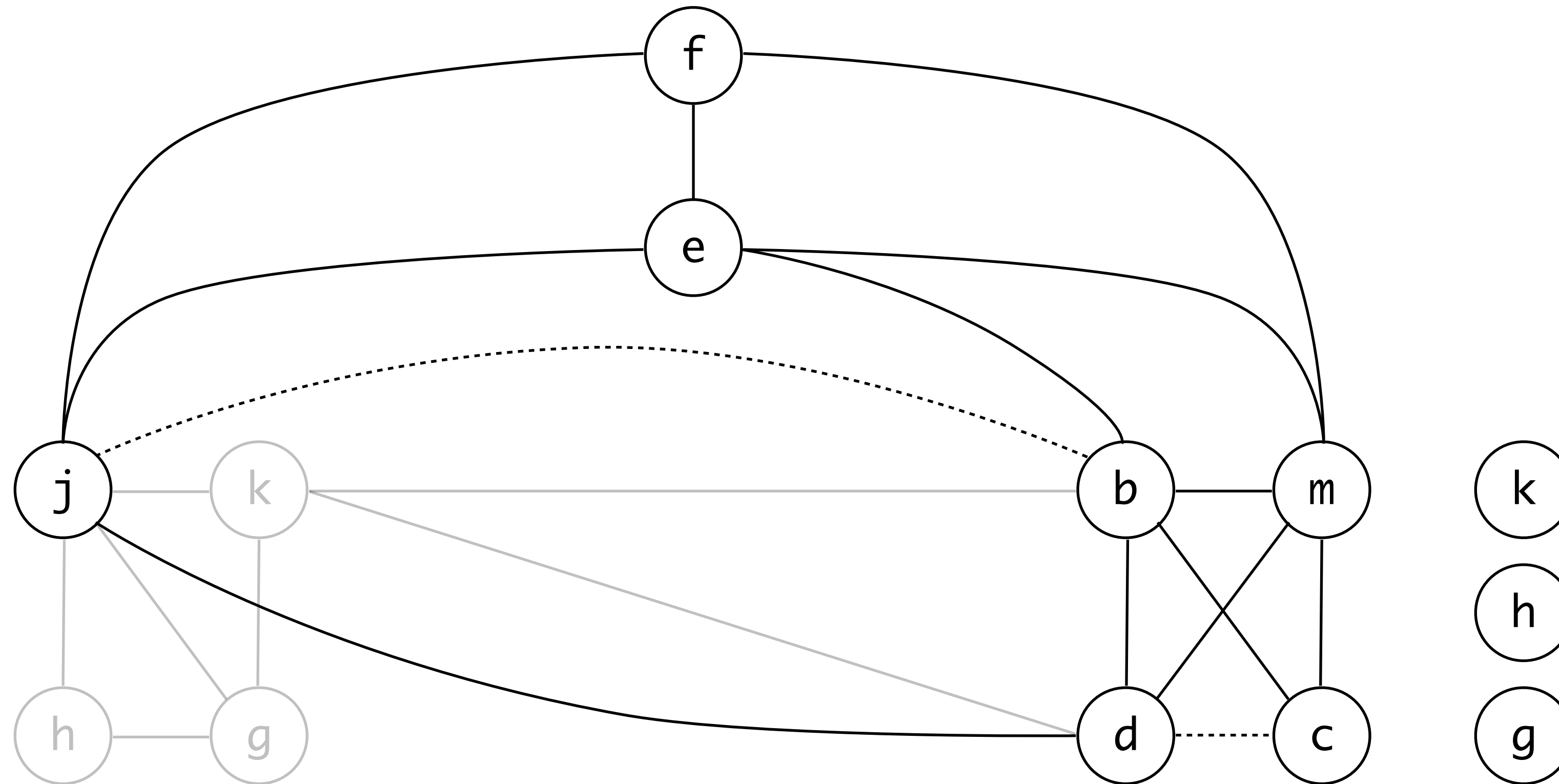
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```


Coalescing

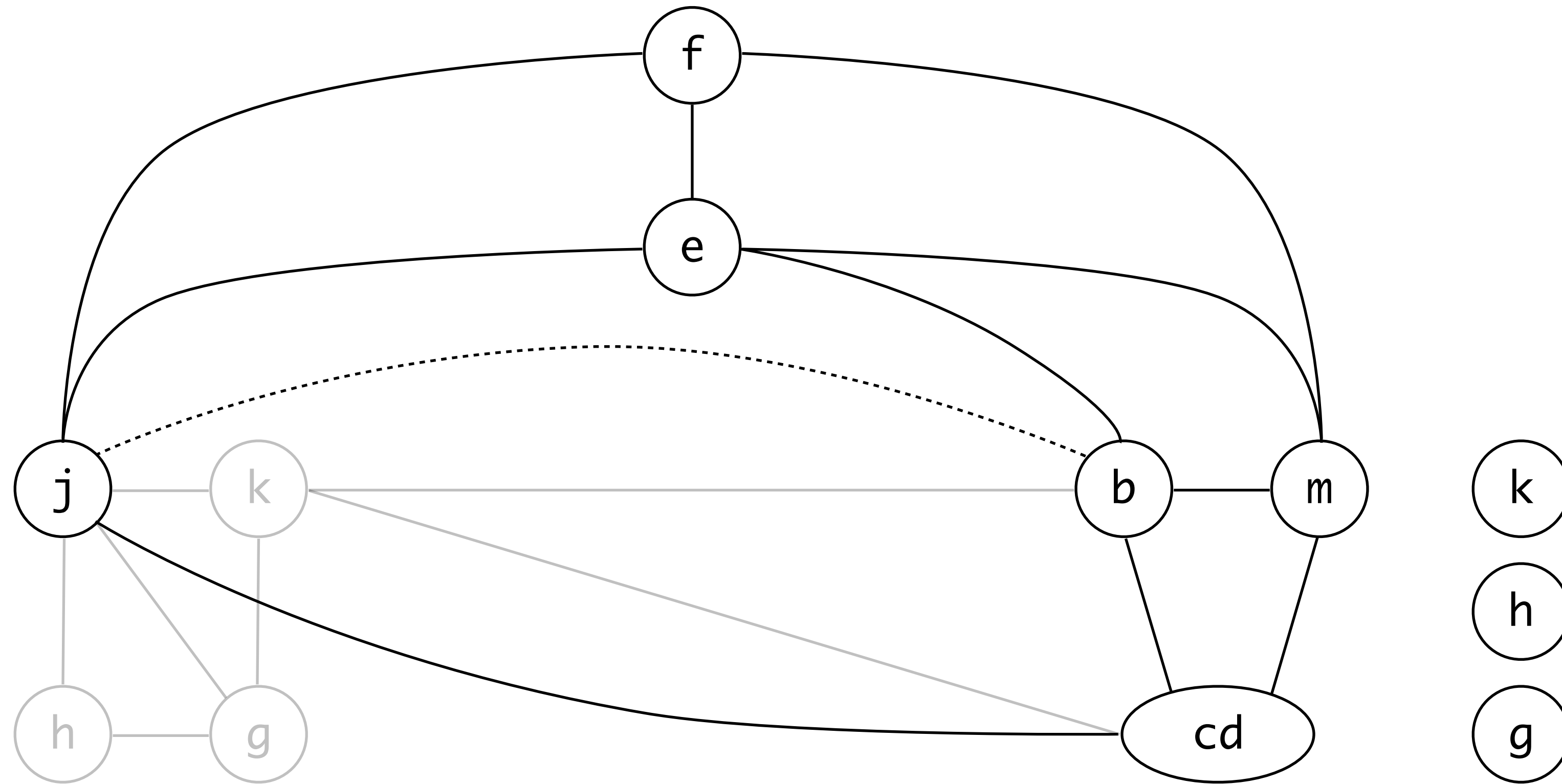
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Coalescing

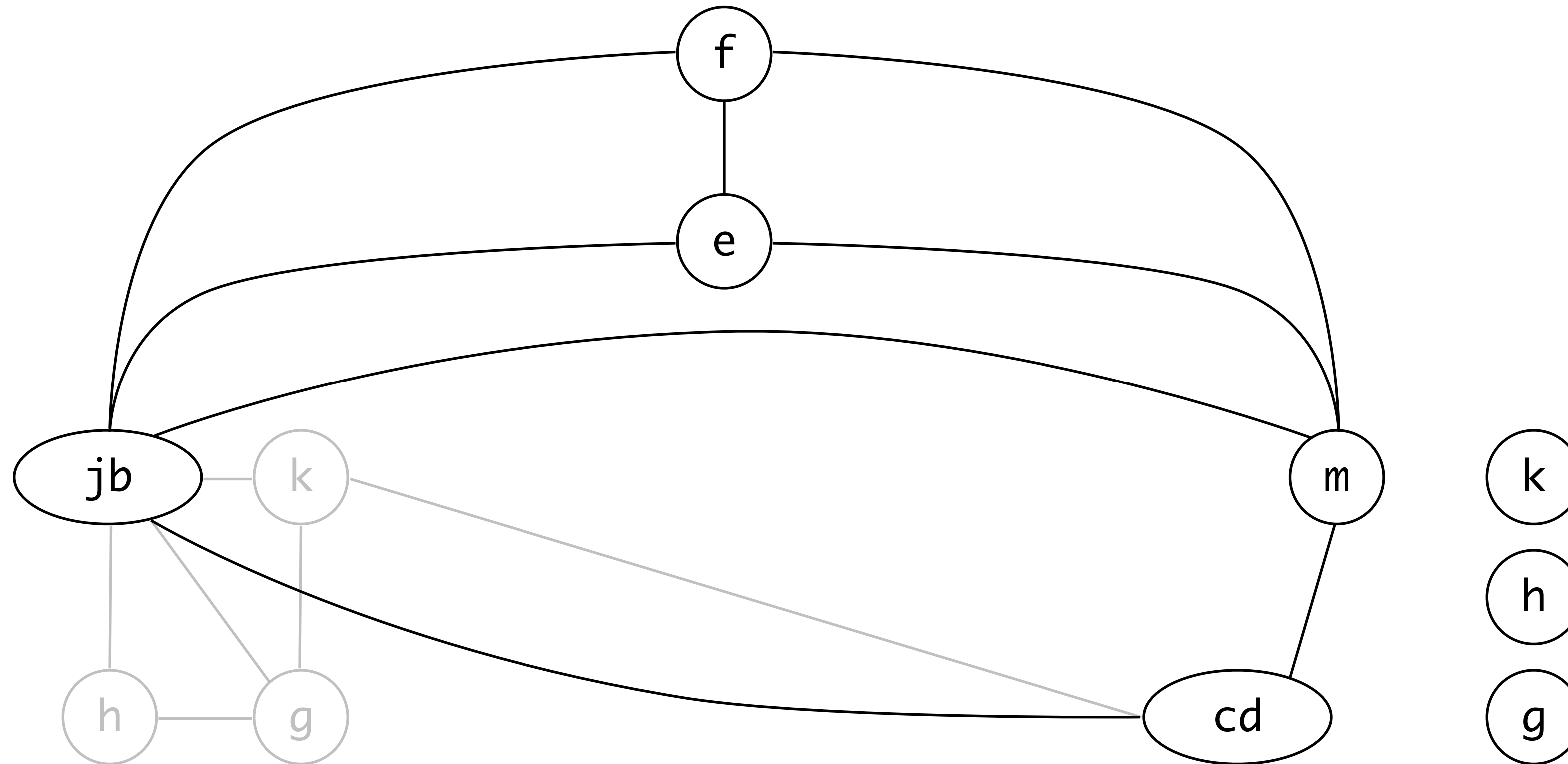
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Coalescing

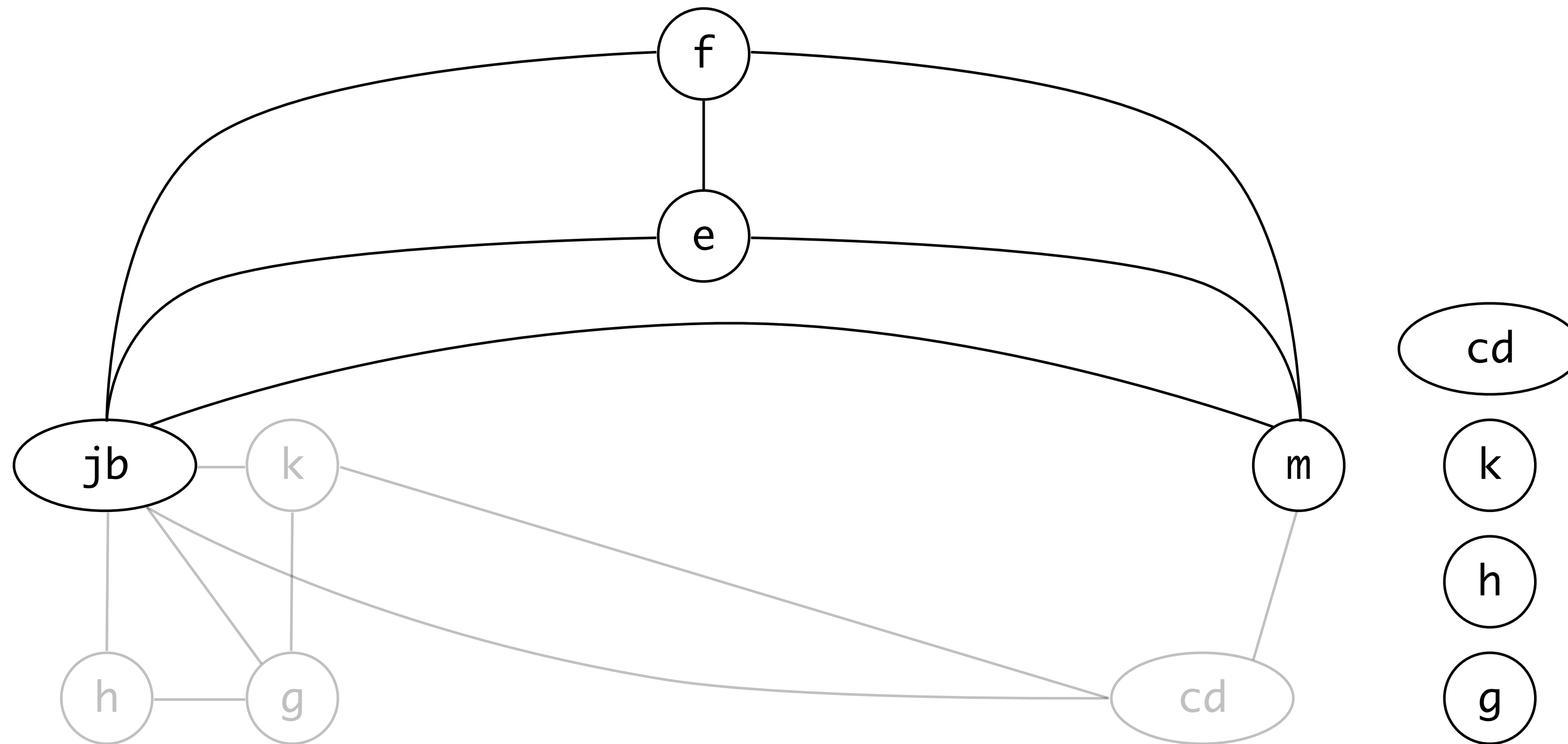
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Coalescing

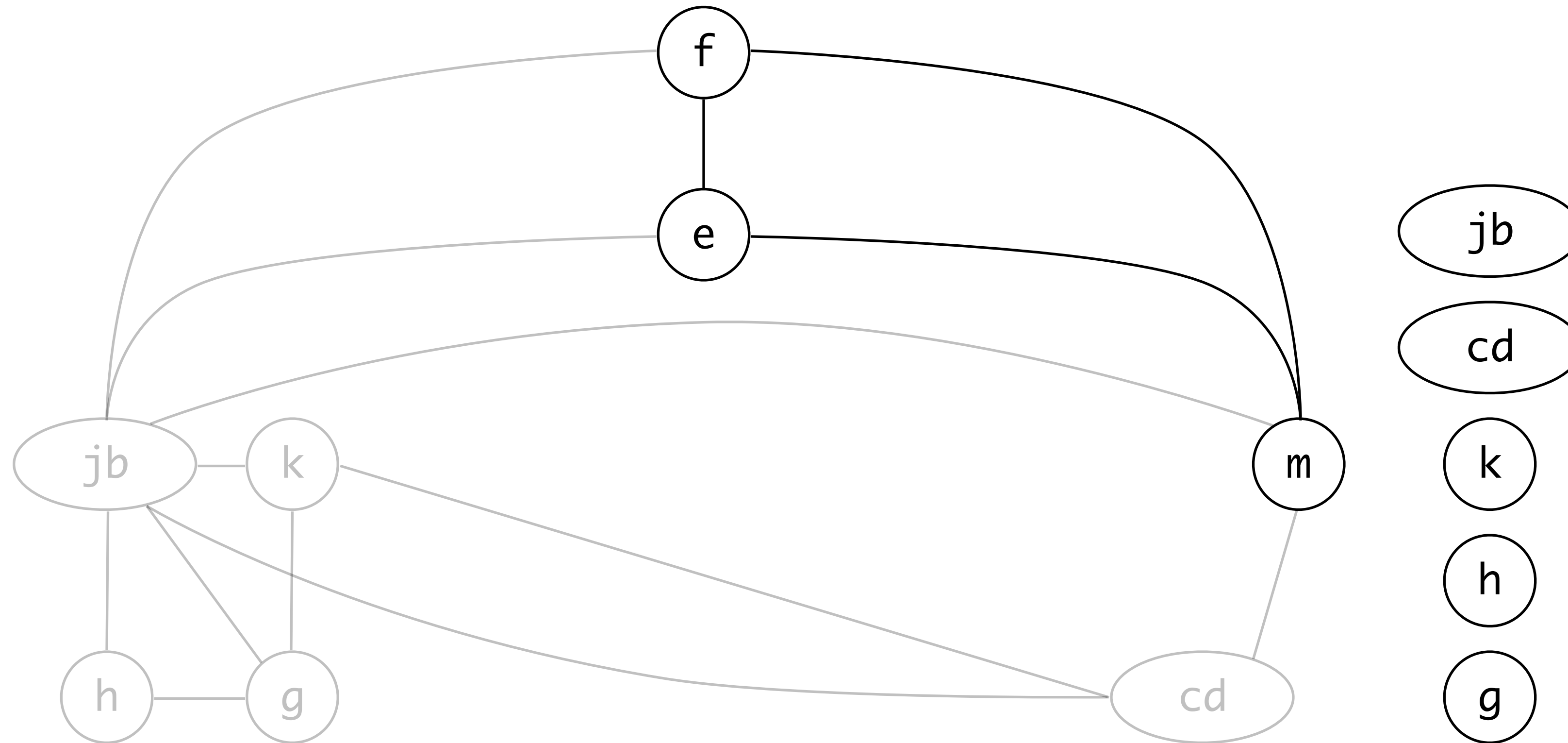
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Coalescing

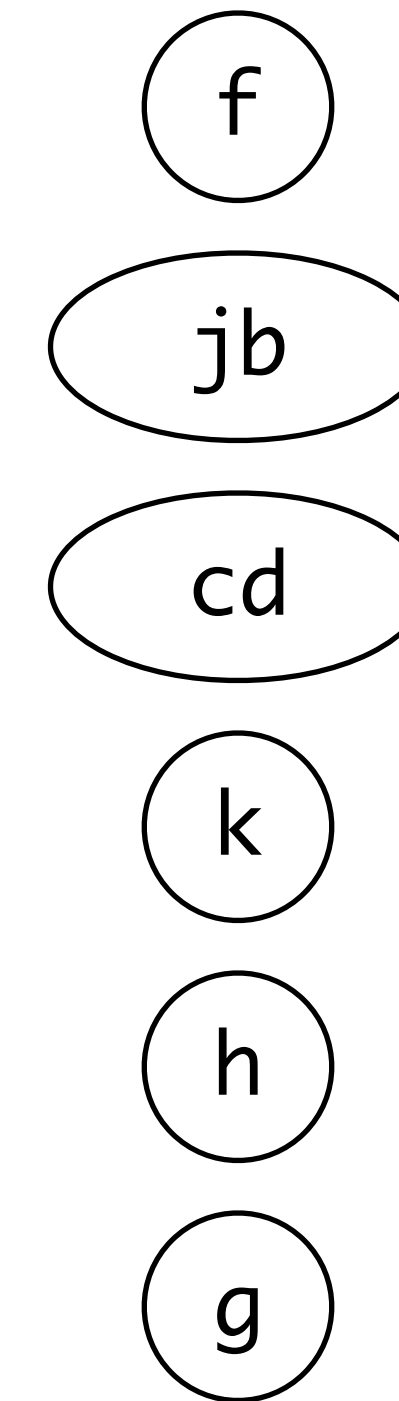
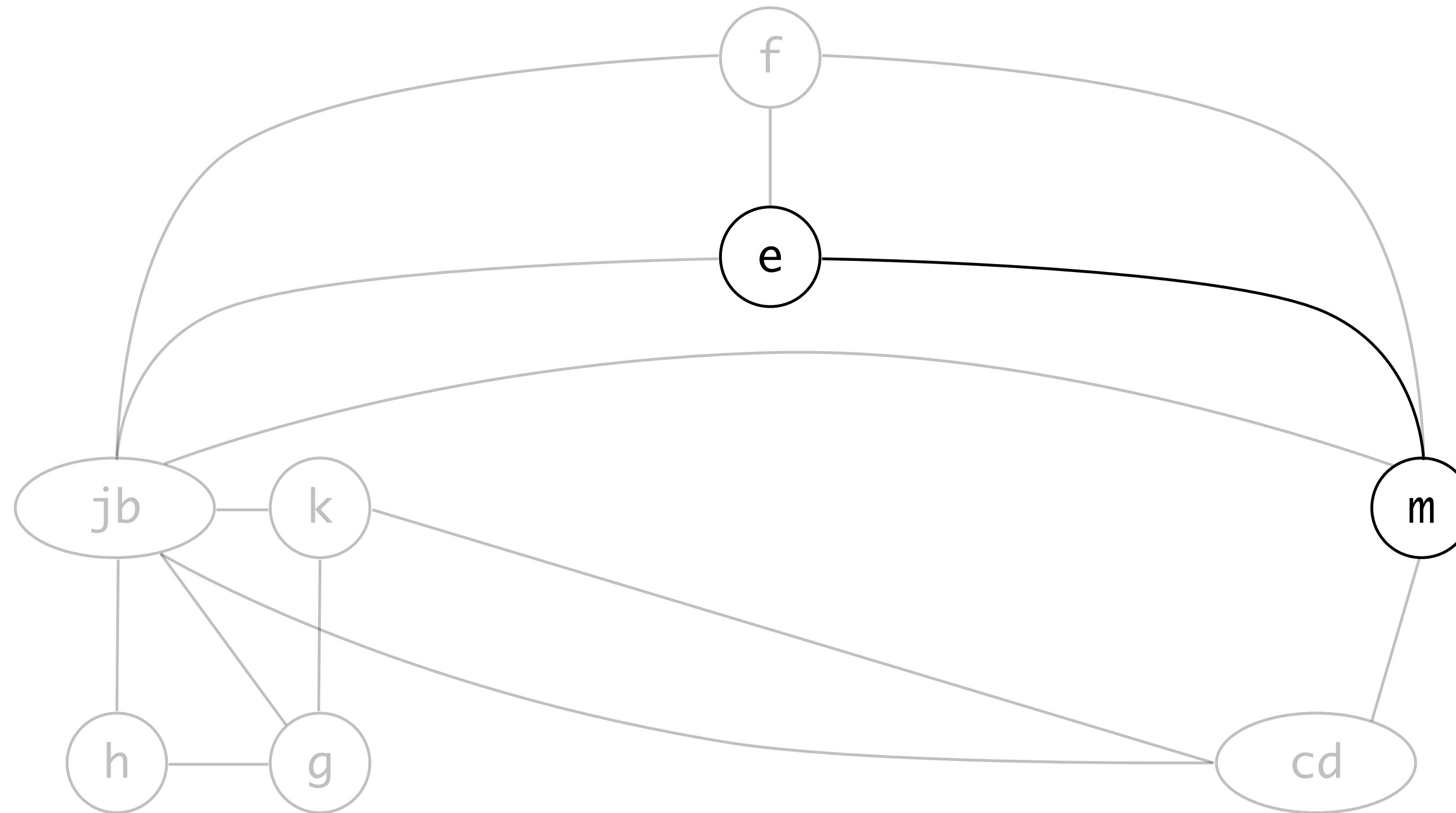
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Coalescing

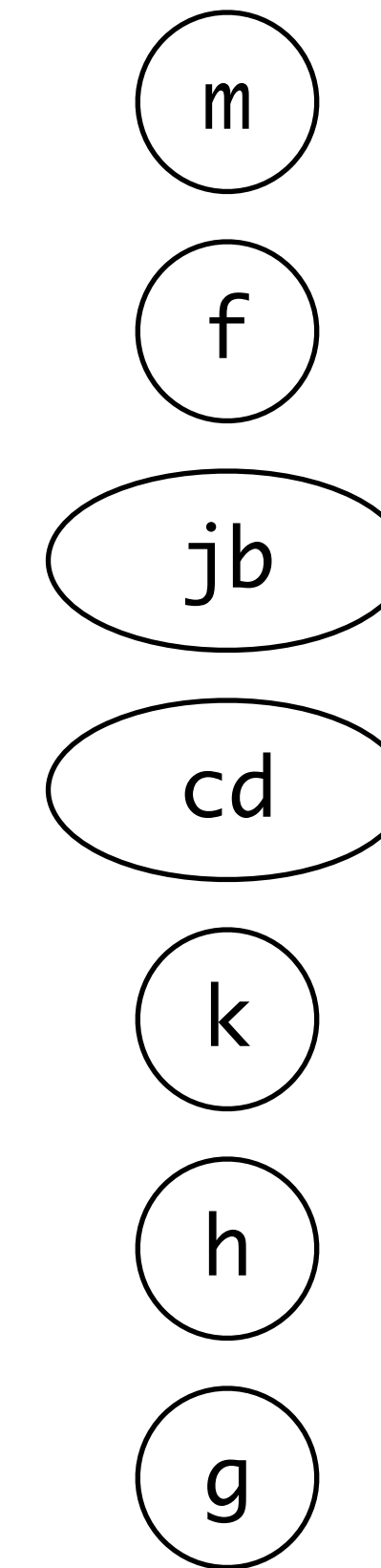
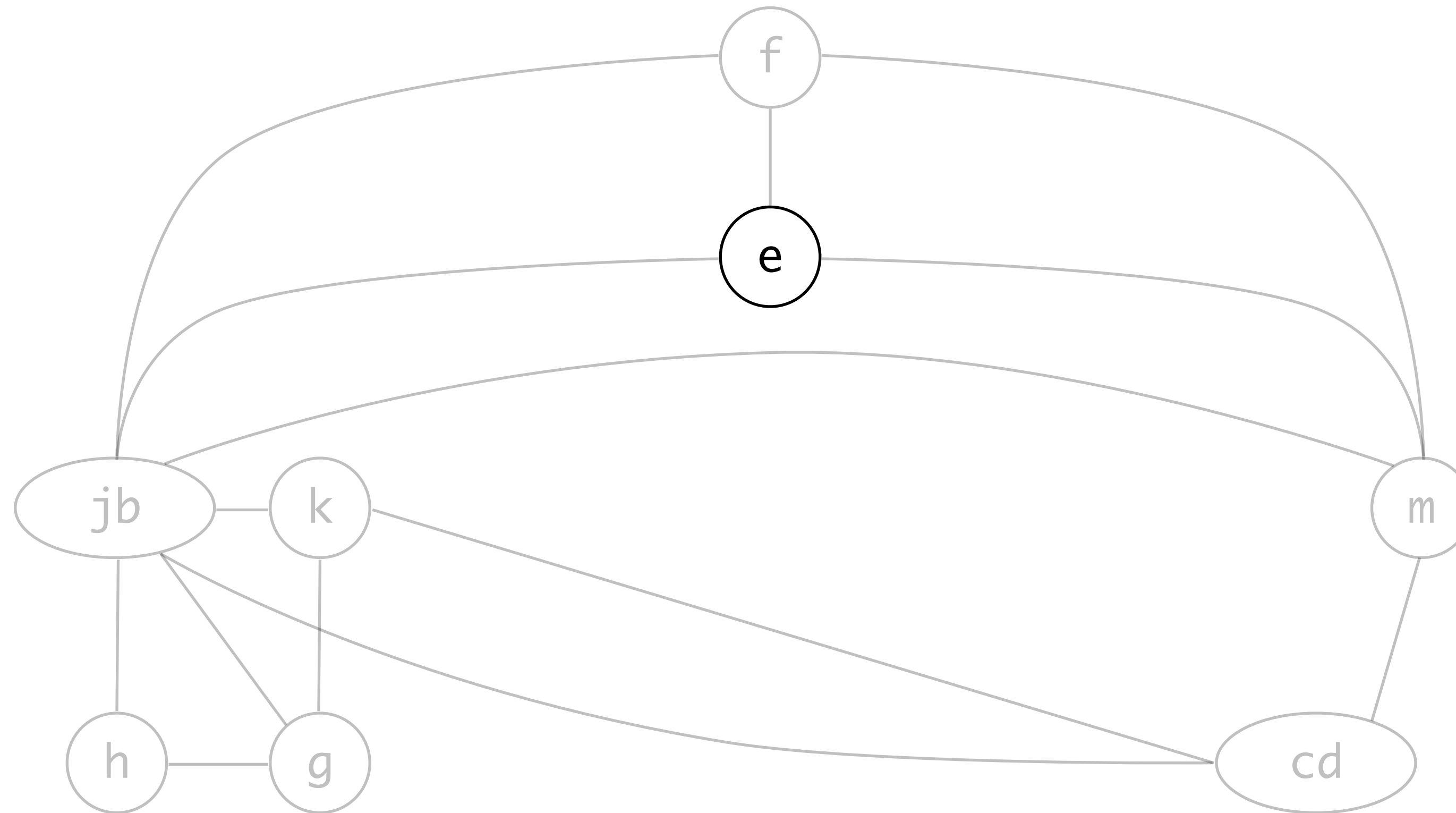
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
e := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live out: d k j
```

Coalescing

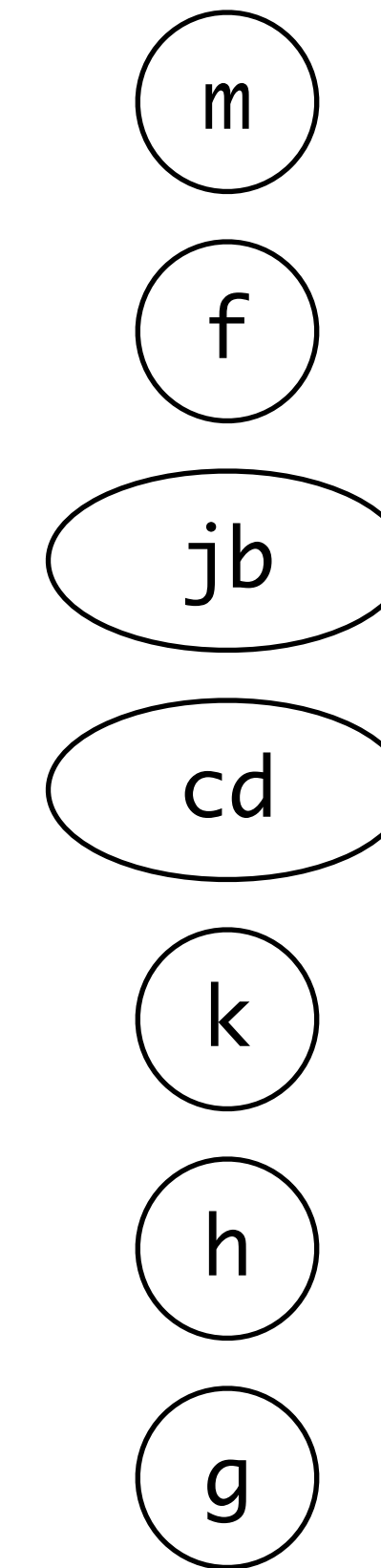
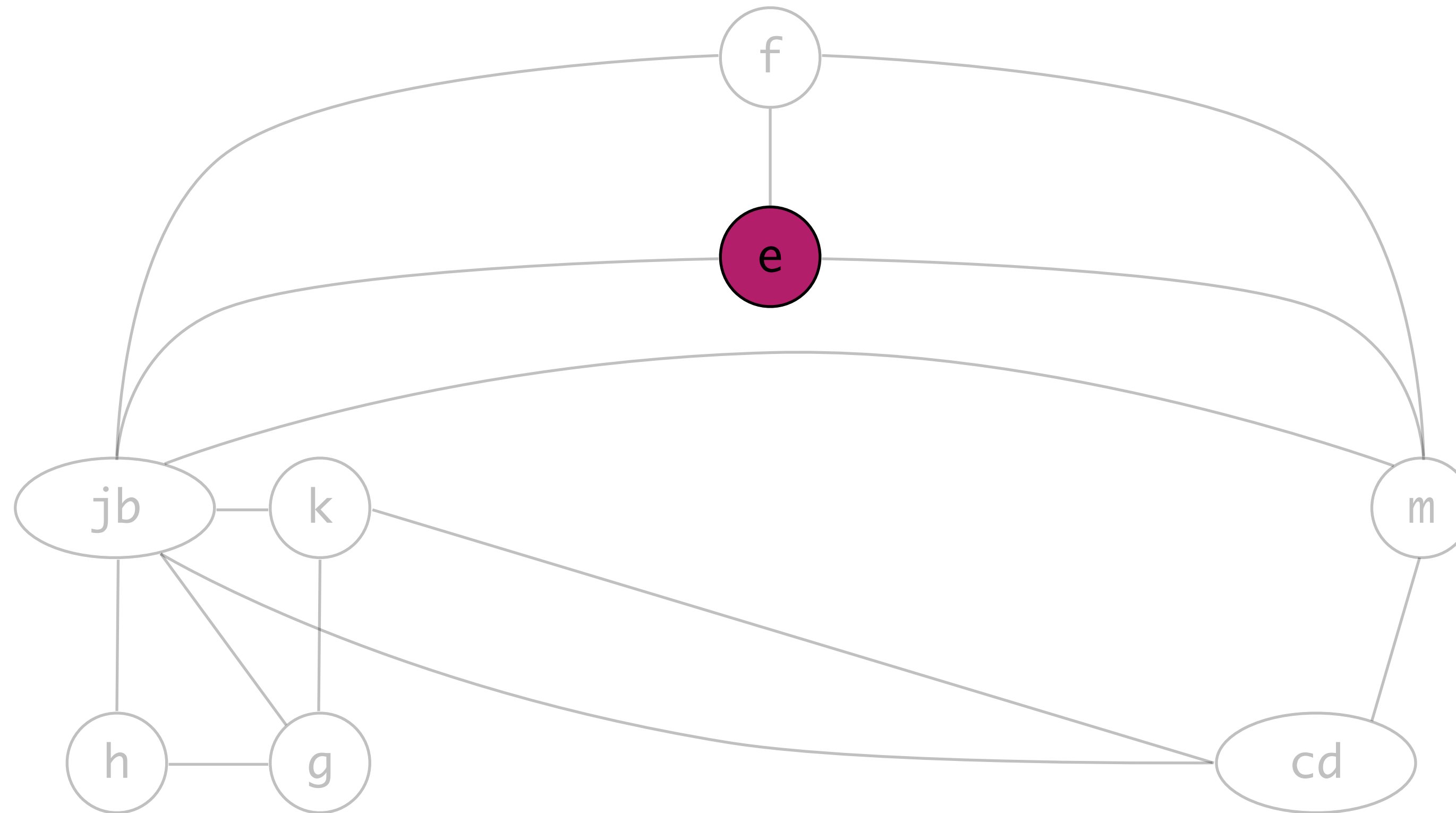
r_1
 r_2
 r_3
 r_4



```
live-in: k j  
g := mem[j + 12]  
h := k - 1  
f := g * h  
e := mem[j + 8]  
m := mem[j + 16]  
b := mem[f]  
c := e + 8  
d := c  
k := m + 4  
j := b  
live out: d k j
```

Coalescing

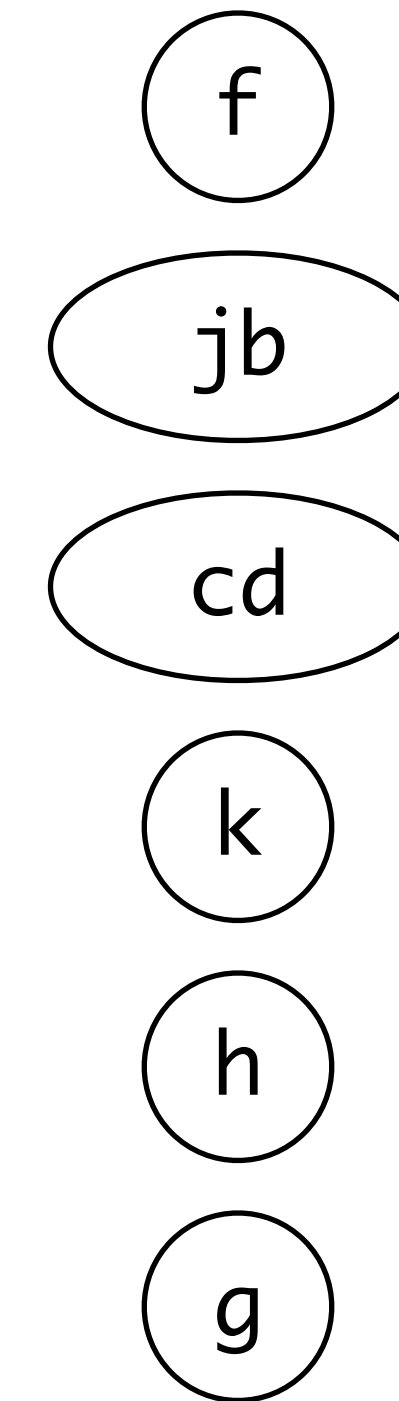
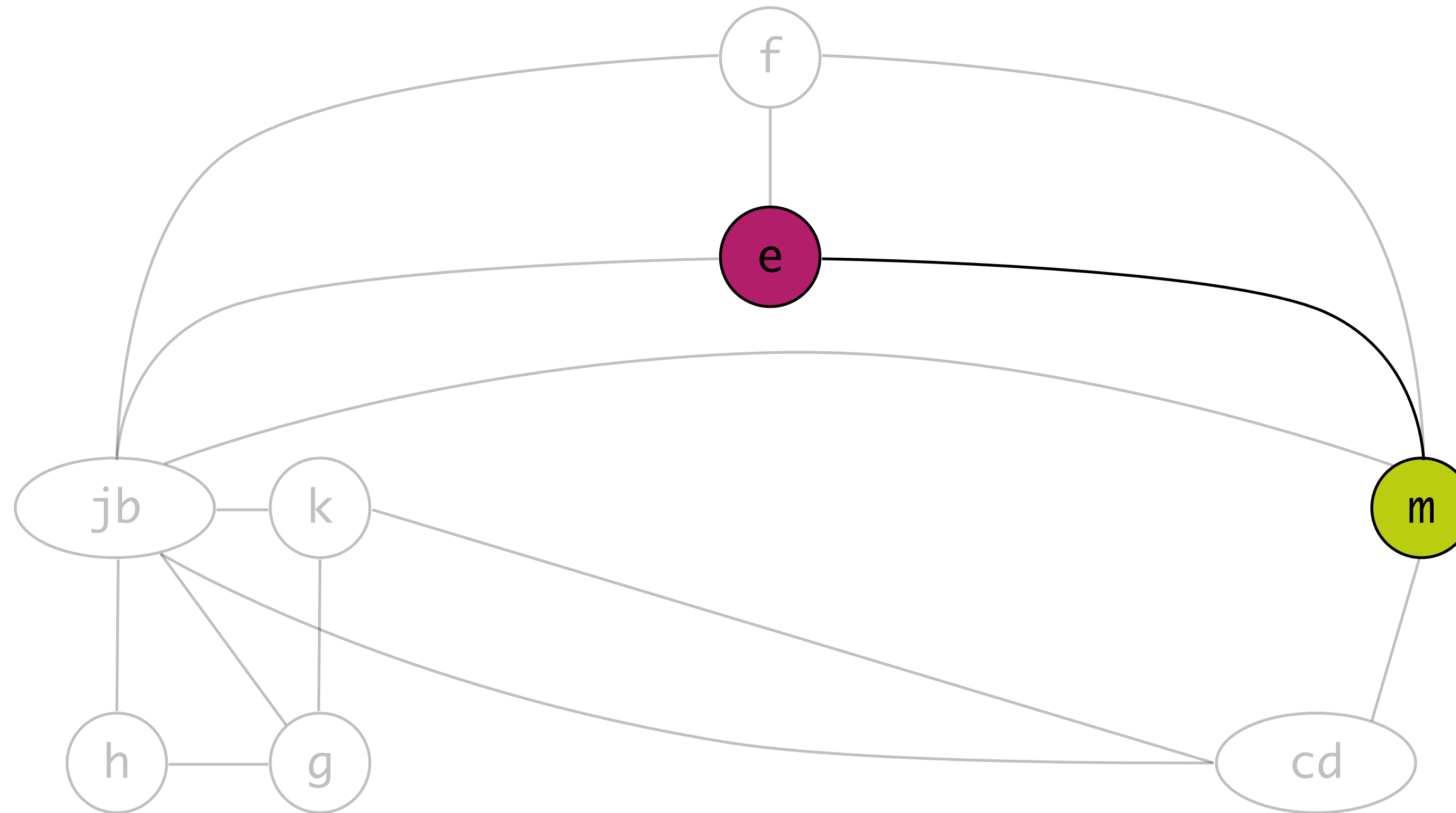
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
r1 := mem[j + 8]
m := mem[j + 16]
b := mem[f]
c := r1 + 8
d := c
k := m + 4
j := b
live out: d k j
```


Coalescing

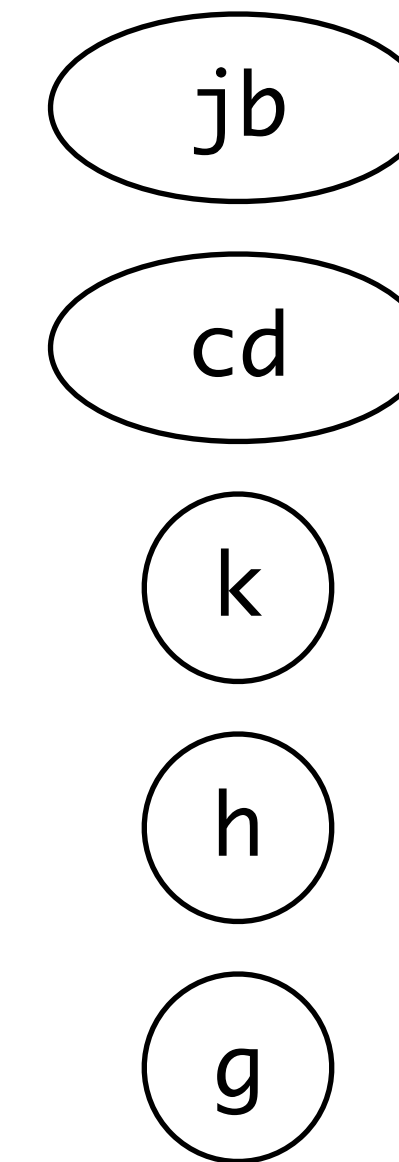
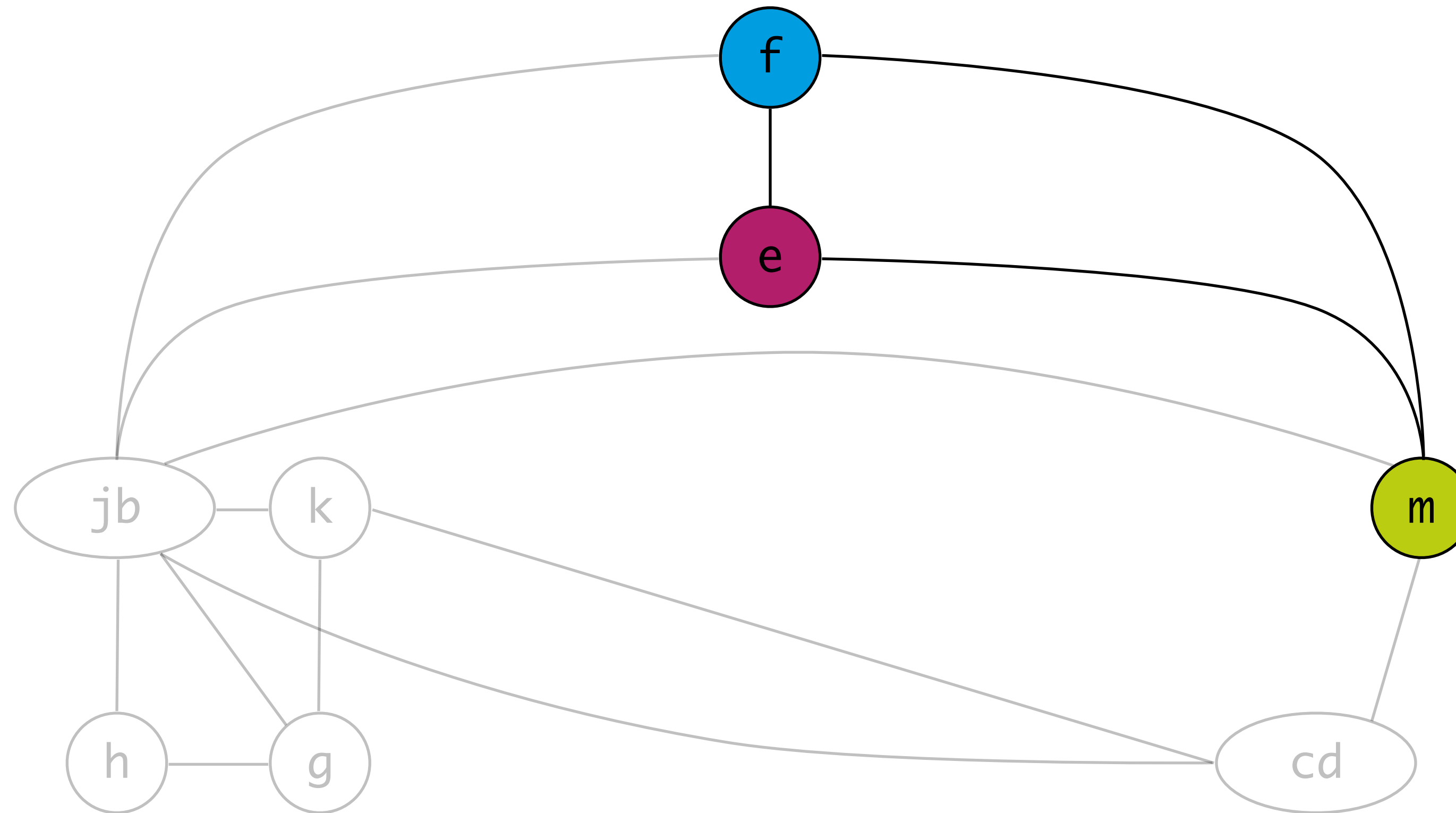
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
f := g * h
r1 := mem[j + 8]
r2 := mem[j + 16]
b := mem[f]
c := r1 + 8
d := c
k := r2 + 4
j := b
live out: d k j
```

Coalescing

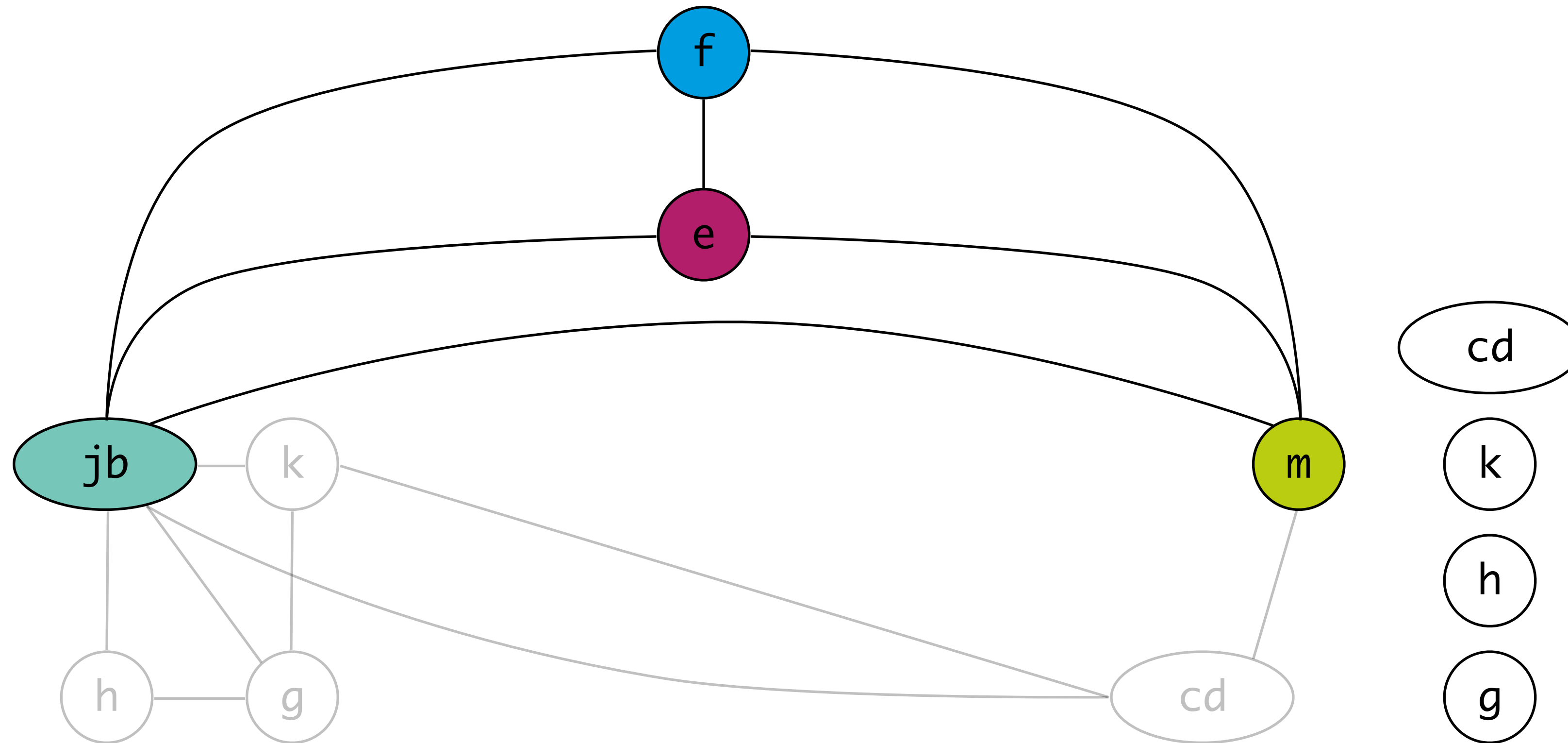
r_1
 r_2
 r_3
 r_4



```
live-in: k j
g := mem[j + 12]
h := k - 1
r3 := g * h
r1 := mem[j + 8]
r2 := mem[j + 16]
b := mem[r3]
c := r1 + 8
d := c
k := r2 + 4
j := b
live out: d k j
```

Coalescing

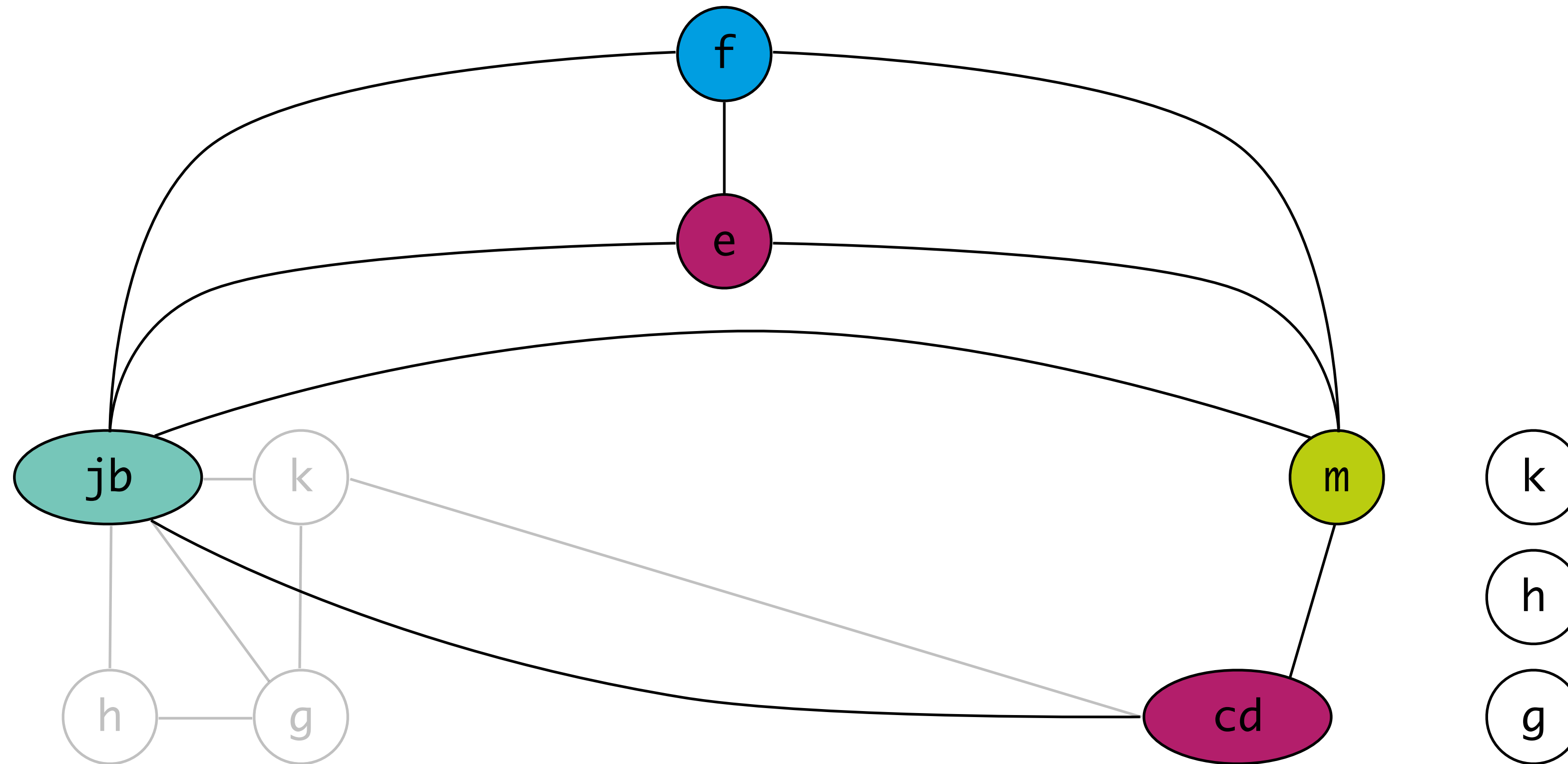
r_1
 r_2
 r_3
 r_4



```
live-in: k r4  
g := mem[r4 + 12]  
h := k - 1  
r3 := g * h  
r1 := mem[r4 + 8]  
r2 := mem[r4 + 16]  
b := mem[r3]  
c := r1 + 8  
d := c  
k := r2 + 4  
r4 := r4  
live out: d k r4
```

Coalescing

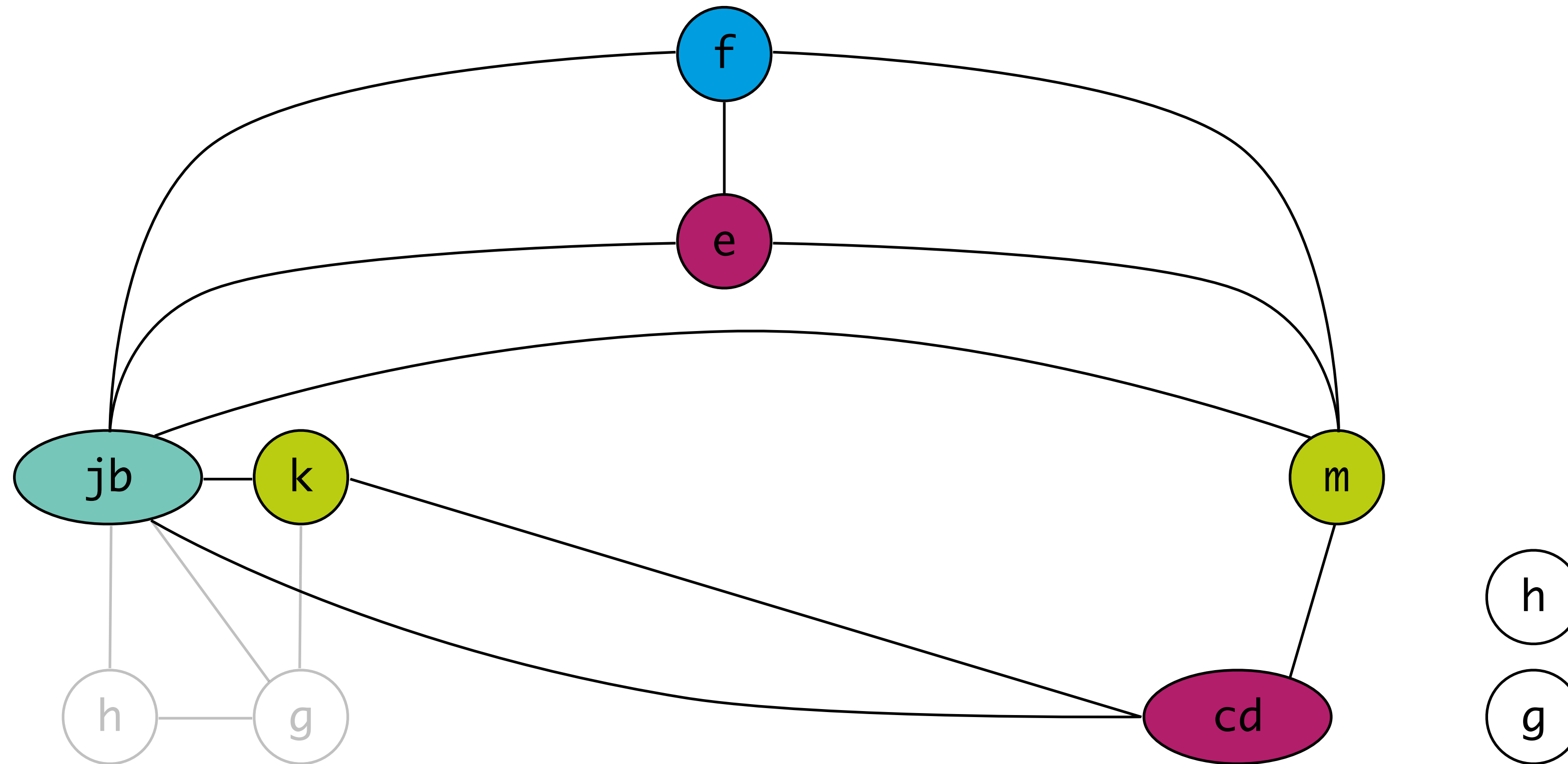
r_1
 r_2
 r_3
 r_4



```
live-in: k r4
g := mem[r4 + 12]
h := k - 1
r3 := g * h
r1 := mem[r4 + 8]
r2 := mem[r4 + 16]
b := mem[r3]
r1 := r1 + 8
r1 := r1
k := r2 + 4
r4 := r4
live out: r1 k r4
```

Coalescing

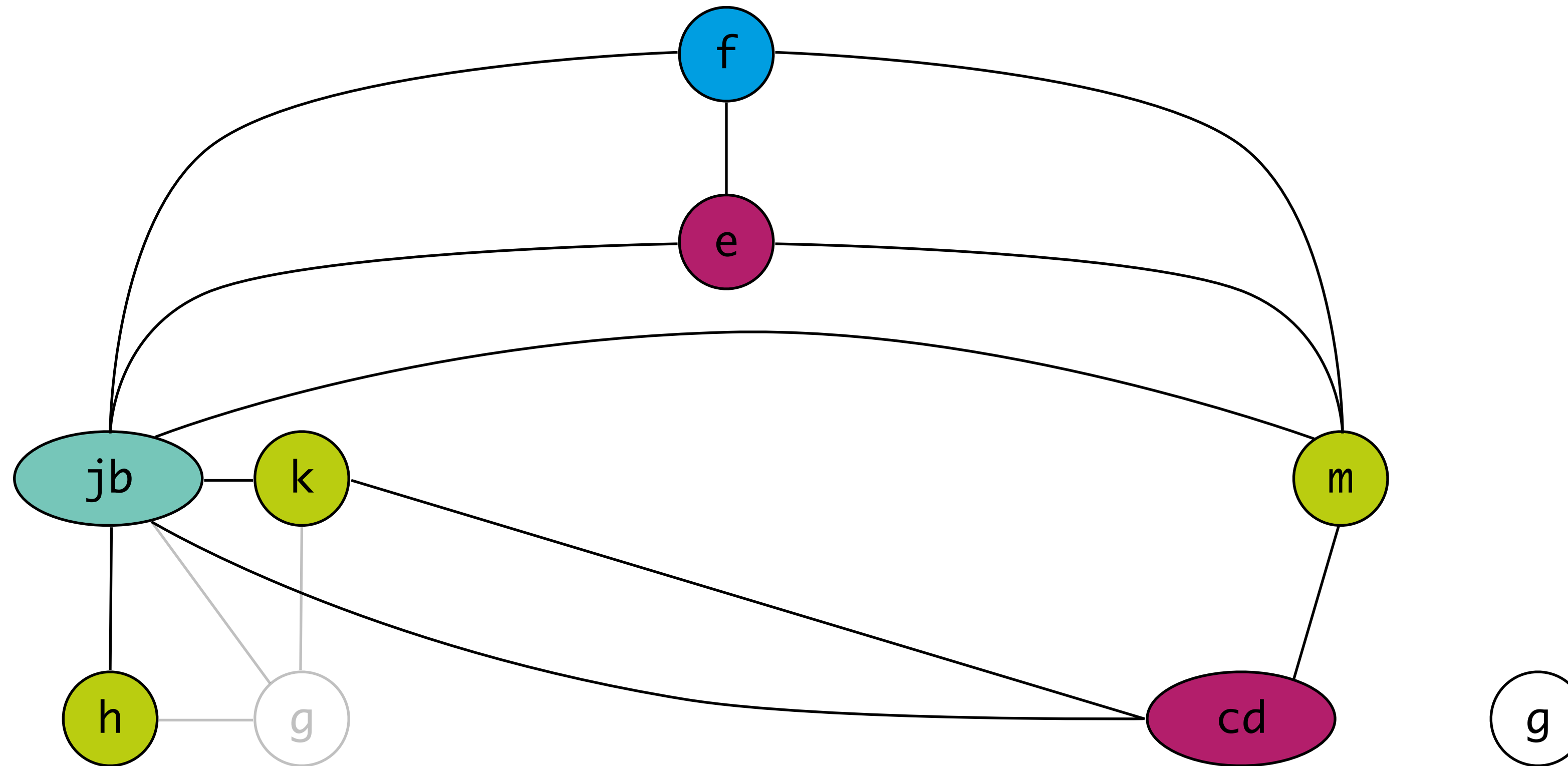
r_1
 r_2
 r_3
 r_4



```
live-in:  $r_2$   $r_4$   
 $g := \text{mem}[\mathbf{r_4} + 12]$   
 $h := \mathbf{r_2} - 1$   
 $r_3 := g * h$   
 $r_1 := \text{mem}[\mathbf{r_4} + 8]$   
 $r_2 := \text{mem}[\mathbf{r_4} + 16]$   
 $b := \text{mem}[\mathbf{r_3}]$   
 $r_1 := r_1 + 8$   
 $r_1 := r_1$   
 $k := r_2 + 4$   
 $r_4 := r_4$   
live out:  $r_1$   $r_2$   $r_4$ 
```

Coalescing

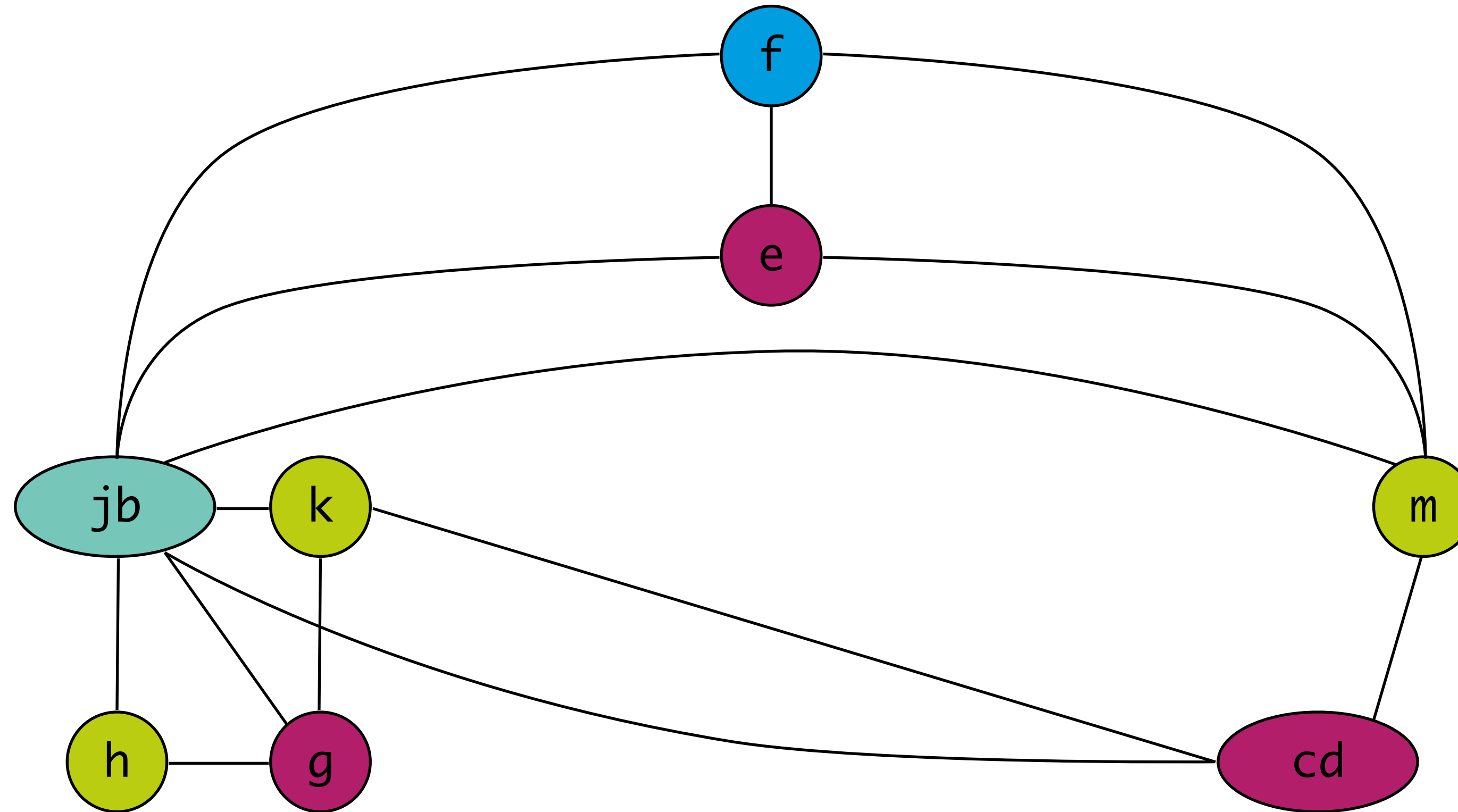
r_1
 r_2
 r_3
 r_4



```
live-in: r2 r4
g := mem[r4 + 12]
r2 := r2 - 1
r3 := g * r2
r1 := mem[r4 + 8]
r2 := mem[r4 + 16]
b := mem[r3]
r1 := r1 + 8
r1 := r1
k := r2 + 4
r4 := r4
live out: r1 r2 r4
```

Coalescing

r_1
 r_2
 r_3
 r_4



```
live-in: r2 r4  
r1 := mem[r4 + 12]  
r2 := r2 - 1  
r3 := r1 * r2  
r1 := mem[r4 + 8]  
r2 := mem[r4 + 16]  
b := mem[r3]  
r1 := r1 + 8  
r1 := r1  
k := r2 + 4  
r4 := r4  
live out: r1 r2 r4
```

Pre-Colored Nodes

Recap: Calling Conventions: CDECL

Caller

- push parameters right-to-left on the stack
- clean-up stack after call

```
push 21
push 42
call _f
add ESP 8
```

Callee

- save old BP
- initialise new BP
- save registers
- return result in AX
- restore registers
- restore BP

```
push EBP
mov EBP ESP
mov EAX [EBP + 8]
mov EDX [EBP + 12]
add EAX EDX
pop EBP
ret
```

Recap: Calling Conventions: STDCALL

Caller

- push parameters right-to-left on the stack

```
push 21  
push 42  
call _f@8
```

Callee

- save old BP
- initialise new BP
- save registers
- return result in AX
- restore registers
- restore BP

```
push EBP  
mov EBP ESP  
mov EAX [EBP + 8]  
mov EDX [EBP + 12]  
add EAX EDX  
pop EBP  
ret 8
```

Recap: Calling Conventions: FASTCALL

Caller

- passes parameters in registers
- pushes additional parameters right-to-left on the stack
- cleans up the stack

```
mov ECX 21  
mov EDX 42  
call @f@8
```

Callee

- save old BP, initialise new BP
- save registers
- return result in AX
- restore registers
- restore BP

```
push EBP  
mov EBP ESP  
mov EAX ECX  
add EAX EDX  
pop EBP  
ret
```

Recap: Calling Conventions: Saving Registers

Not enough registers for all local variables across life time

- save register to memory to free for other use

Caller-save registers

- Caller is responsible for saving and restoring register

Callee-save registers

- Callee is responsible for saving and restoring register

Use callee-save registers to pass parameters

Pre-Colored Nodes: representing registers

Nodes

- register = pre-colored node
- no simplify, no spill
- coalesce possible

Edges

- all registers interfere with each other
- explicit usage of registers
- call and return instructions influence liveness

Callee-Save Register in Temporary

```
enter: def(r7)
      ...
exit: use(r7)
```

```
enter: def(r7)
      t ← r7
      ...
      r7 ← t
exit: use(r7)
```

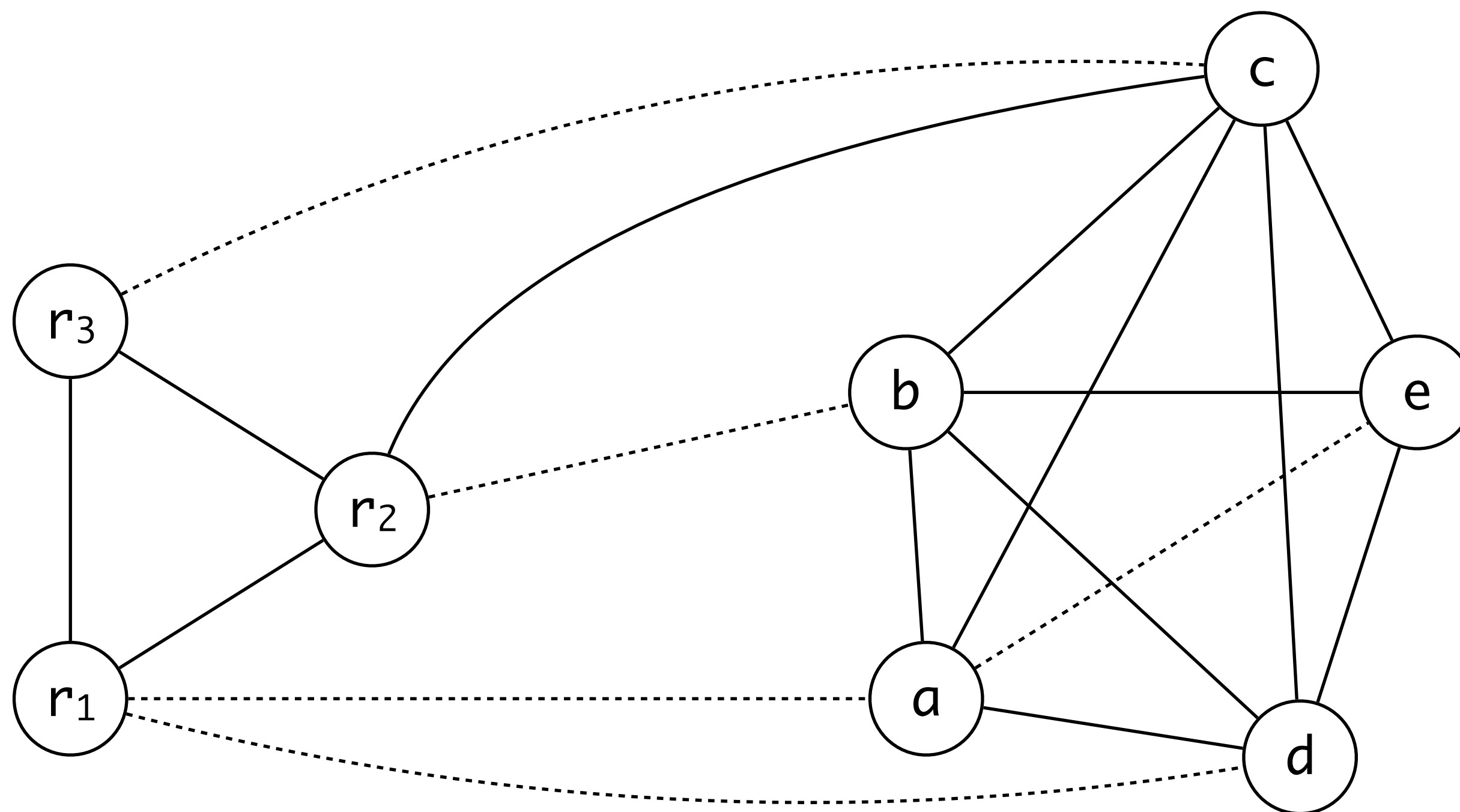
Pre-Colored Nodes

```
int f(int a, int b) {  
    int d = 0;  
    int e = a;  
    do {  
        d = d + b;  
        e = e - 1;  
    } while (e > 0);  
    return d;  
}
```

```
enter : c ← r3    // callee-save  
        a ← r1    // caller-save  
        b ← r2    // caller-save  
        d ← 0  
        e ← a  
loop :  d ← d + b  
        e ← e - 1  
        if e > 0 goto loop  
        r1 ← d  
        r3 ← c  
        return (r1, r3 live out)
```

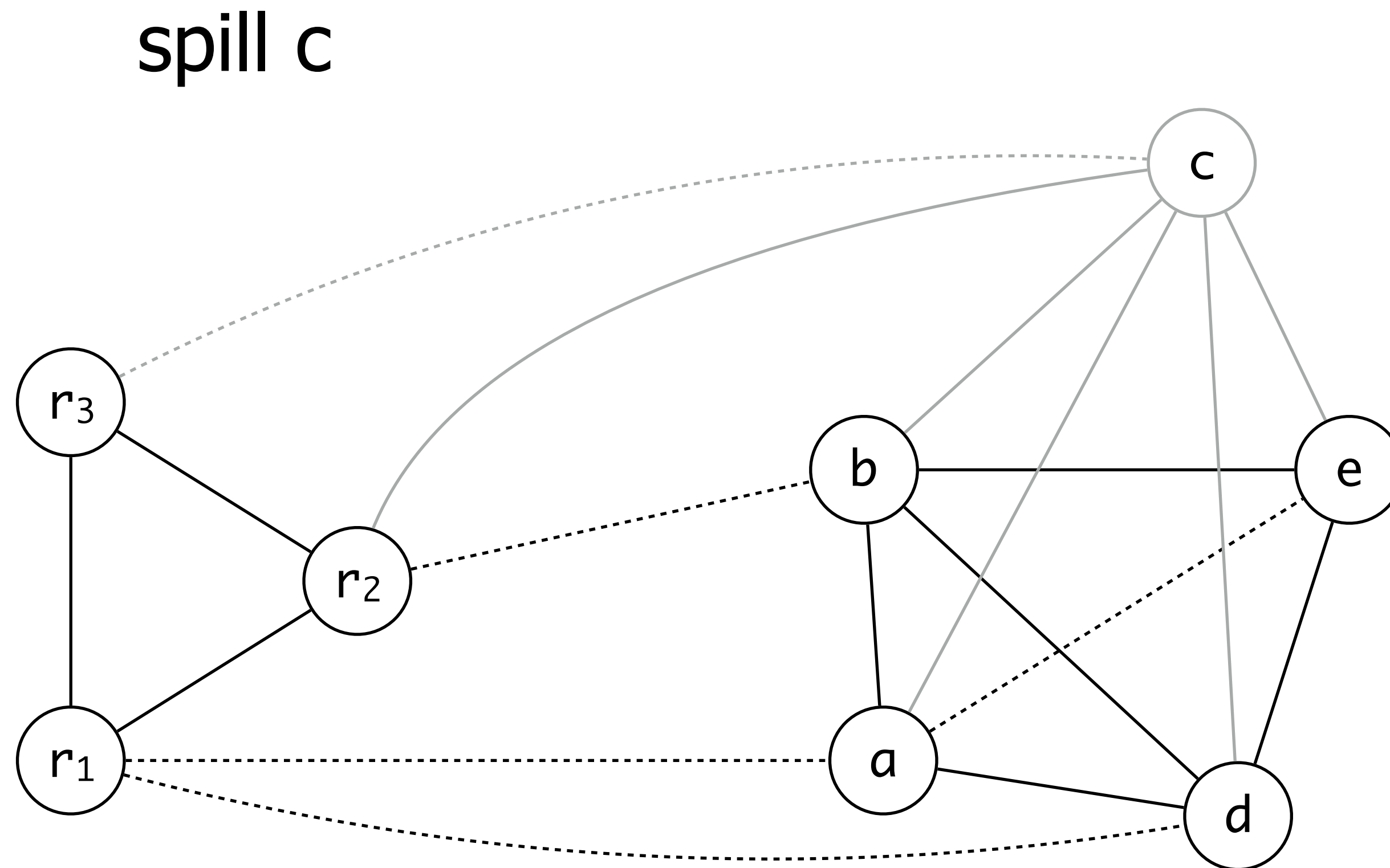
machine has 3 registers

Pre-Colored Nodes

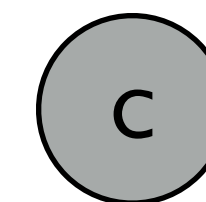


```
enter : c  $\leftarrow$  r3  
        a  $\leftarrow$  r1  
        b  $\leftarrow$  r2  
        d  $\leftarrow$   $\emptyset$   
        e  $\leftarrow$  a  
loop  : d  $\leftarrow$  d + b  
        e  $\leftarrow$  e - 1  
        if e > 0 goto loop  
        r1  $\leftarrow$  d  
        r3  $\leftarrow$  c  
        return (r1, r3)
```


Pre-Colored Nodes

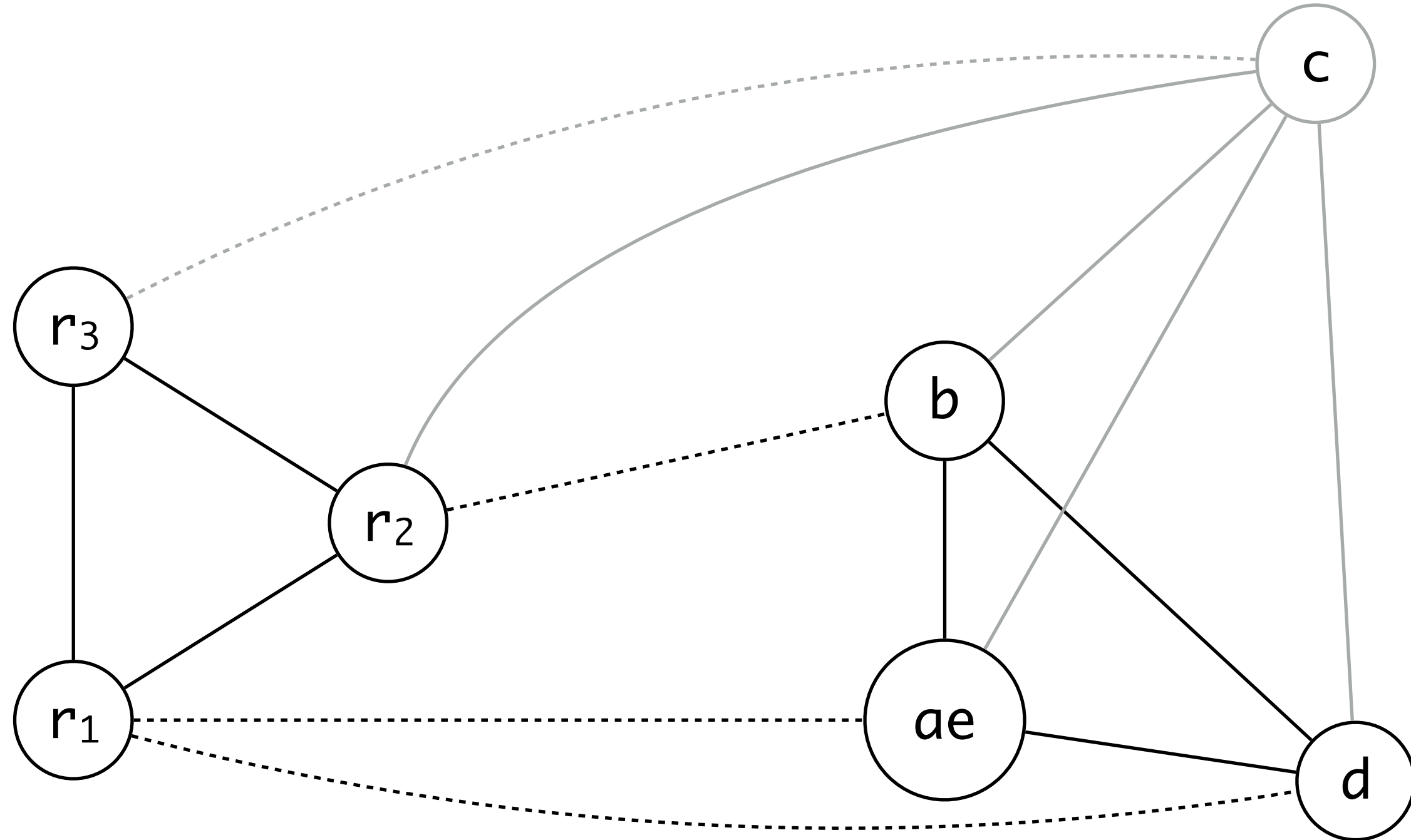


```
enter : c ← r3
        a ← r1
        b ← r2
        d ← 0
        e ← a
loop  : d ← d + b
        e ← e - 1
        if e > 0 goto loop
        r1 ← d
        r3 ← c
        return (r1, r3)
```

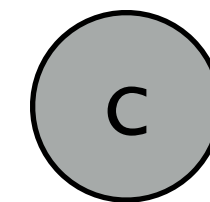


Pre-Colored Nodes

coalesce a and e

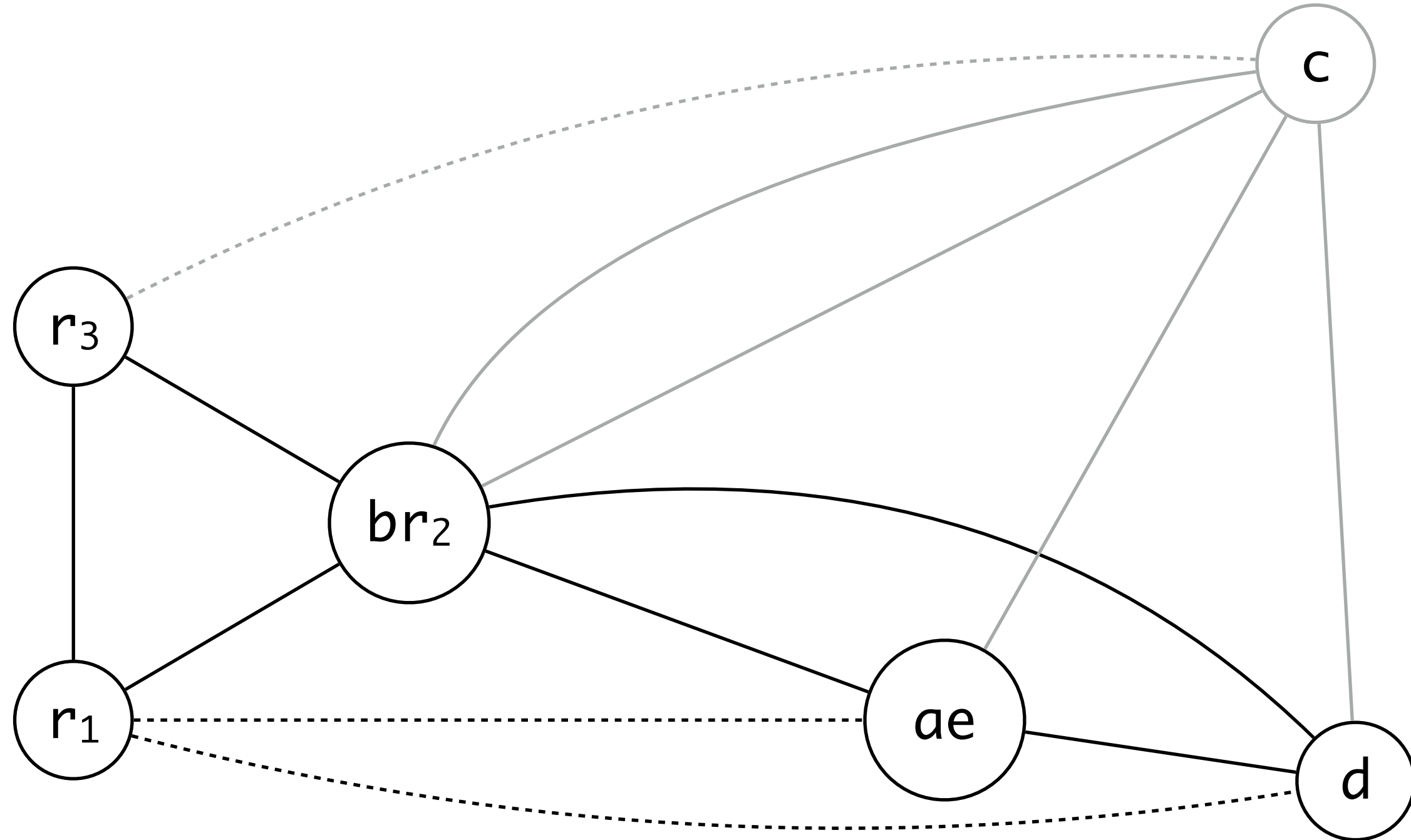


```
enter : c  $\leftarrow$  r3
        a  $\leftarrow$  r1
        b  $\leftarrow$  r2
        d  $\leftarrow$   $\emptyset$ 
        e  $\leftarrow$  a
loop  : d  $\leftarrow$  d + b
        e  $\leftarrow$  e - 1
        if e > 0 goto loop
        r1  $\leftarrow$  d
        r3  $\leftarrow$  c
        return (r1, r3)
```

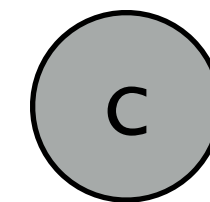


Pre-Colored Nodes

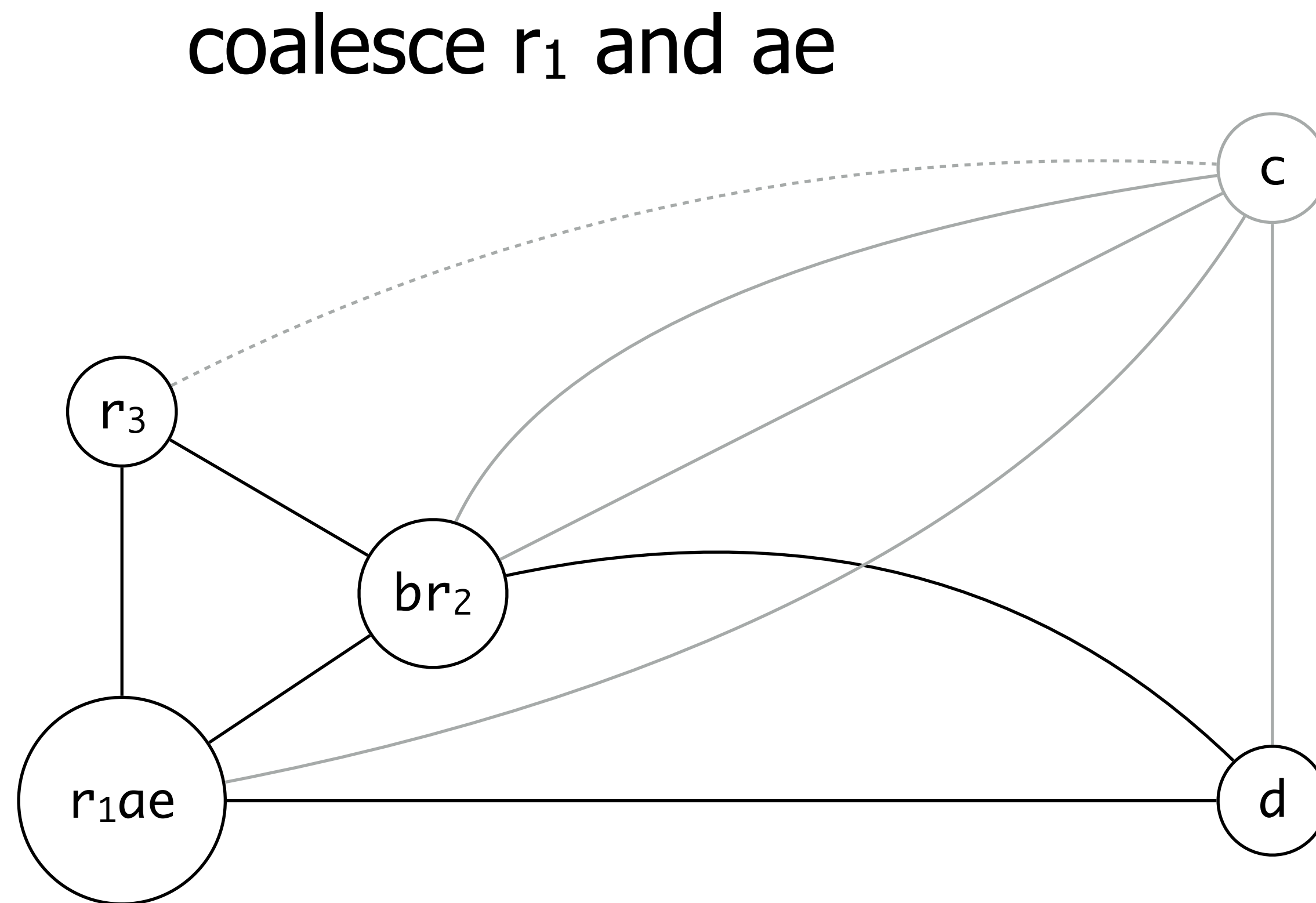
coalesce r_2 and b



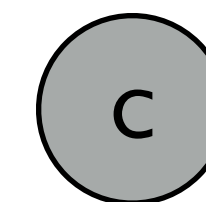
```
enter : c ← r3  
       a ← r1  
       b ← r2  
       d ← ∅  
       e ← a  
loop  : d ← d + b  
       e ← e - 1  
       if e > 0 goto loop  
       r1 ← d  
       r3 ← c  
       return (r1, r3)
```



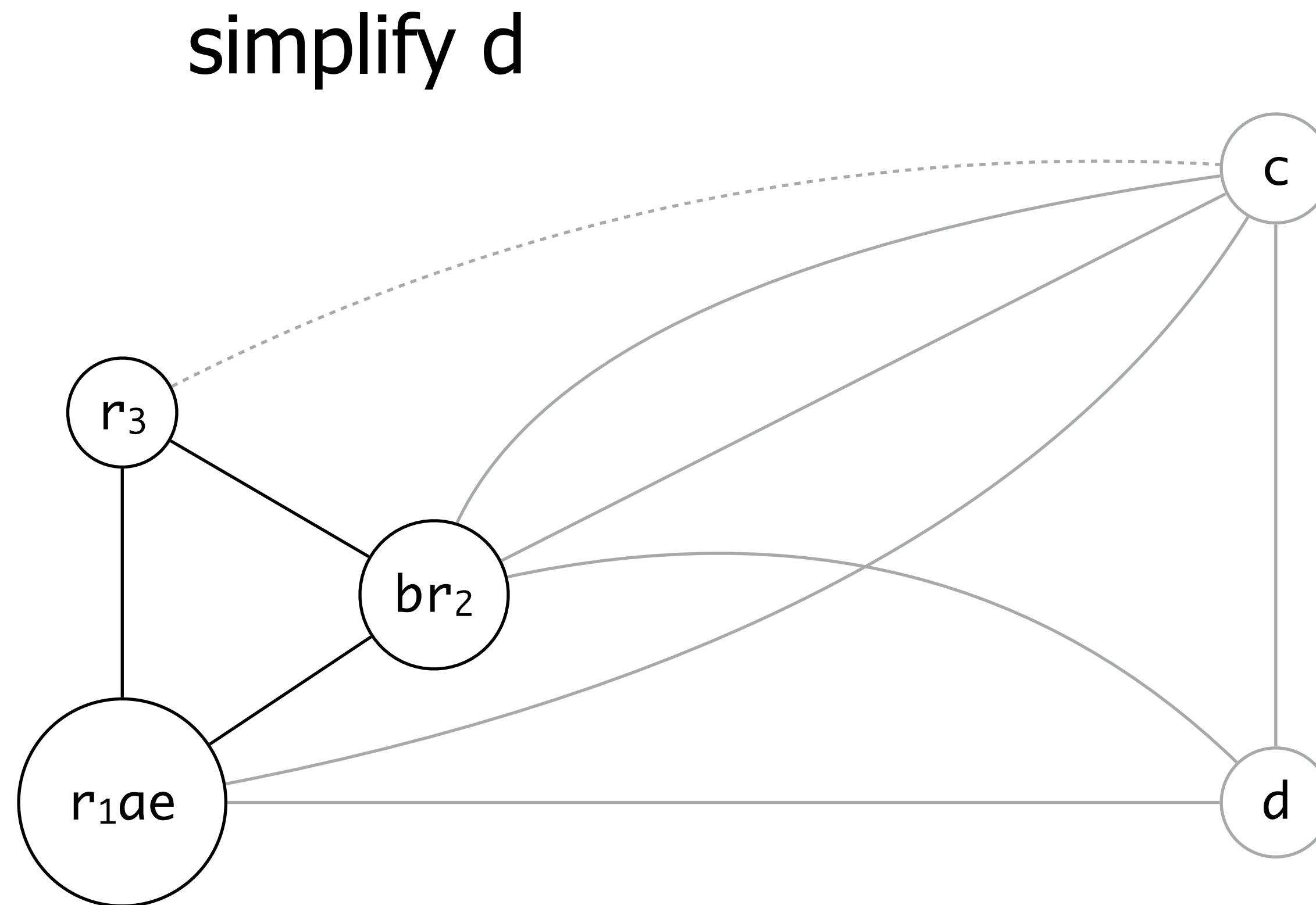
Pre-Colored Nodes



```
enter : c ← r3  
        a ← r1  
        b ← r2  
        d ← 0  
        e ← a  
loop  : d ← d + b  
        e ← e - 1  
        if e > 0 goto loop  
        r1 ← d  
        r3 ← c  
        return (r1, r3)
```



Pre-Colored Nodes

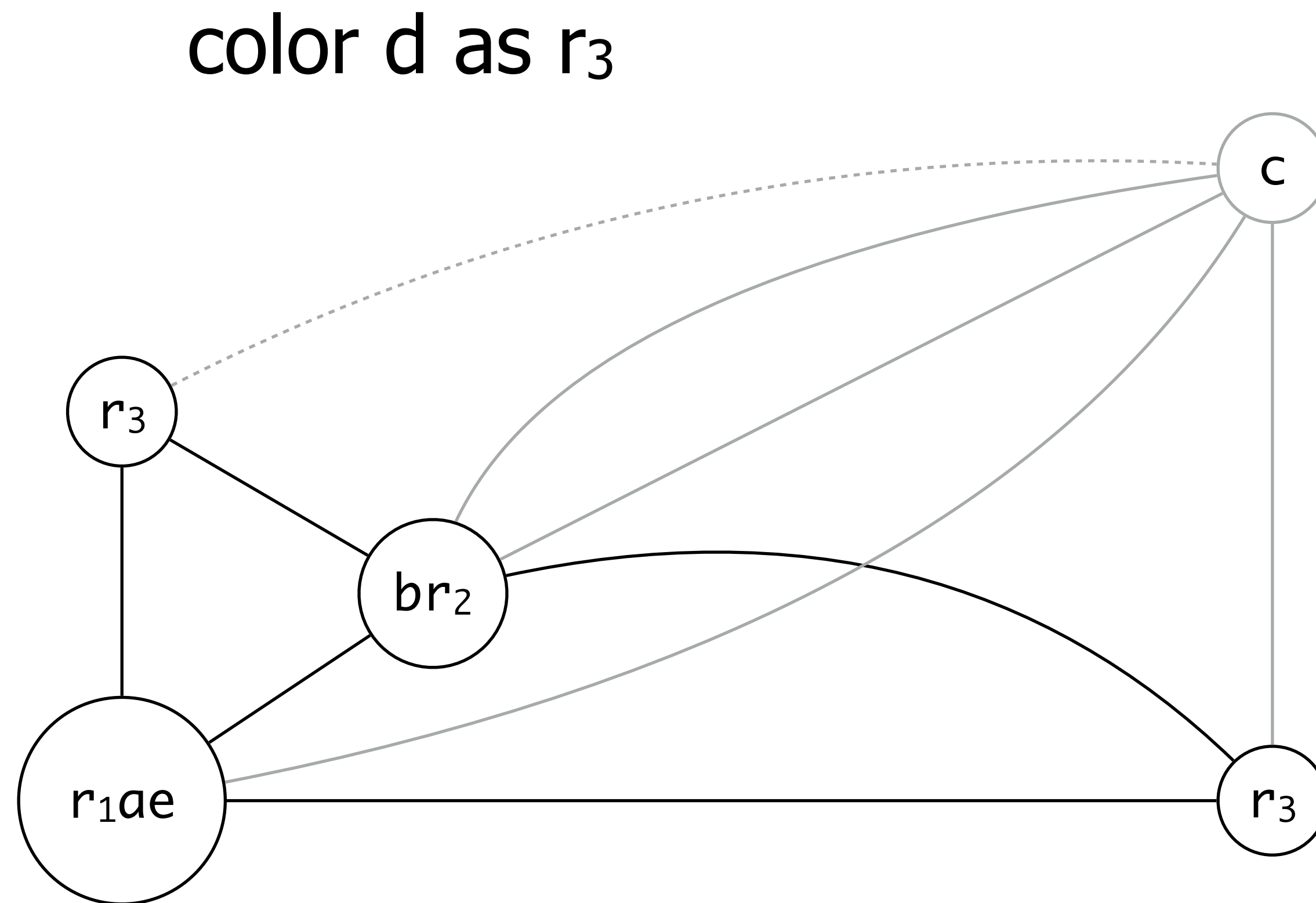


```
enter : c ← r3  
        a ← r1  
        b ← r2  
        d ← 0  
        e ← a  
loop  : d ← d + b  
        e ← e - 1  
        if e > 0 goto loop  
        r1 ← d  
        r3 ← c  
        return (r1, r3)
```

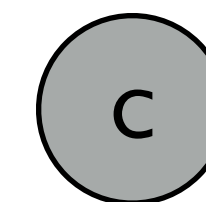
d

c

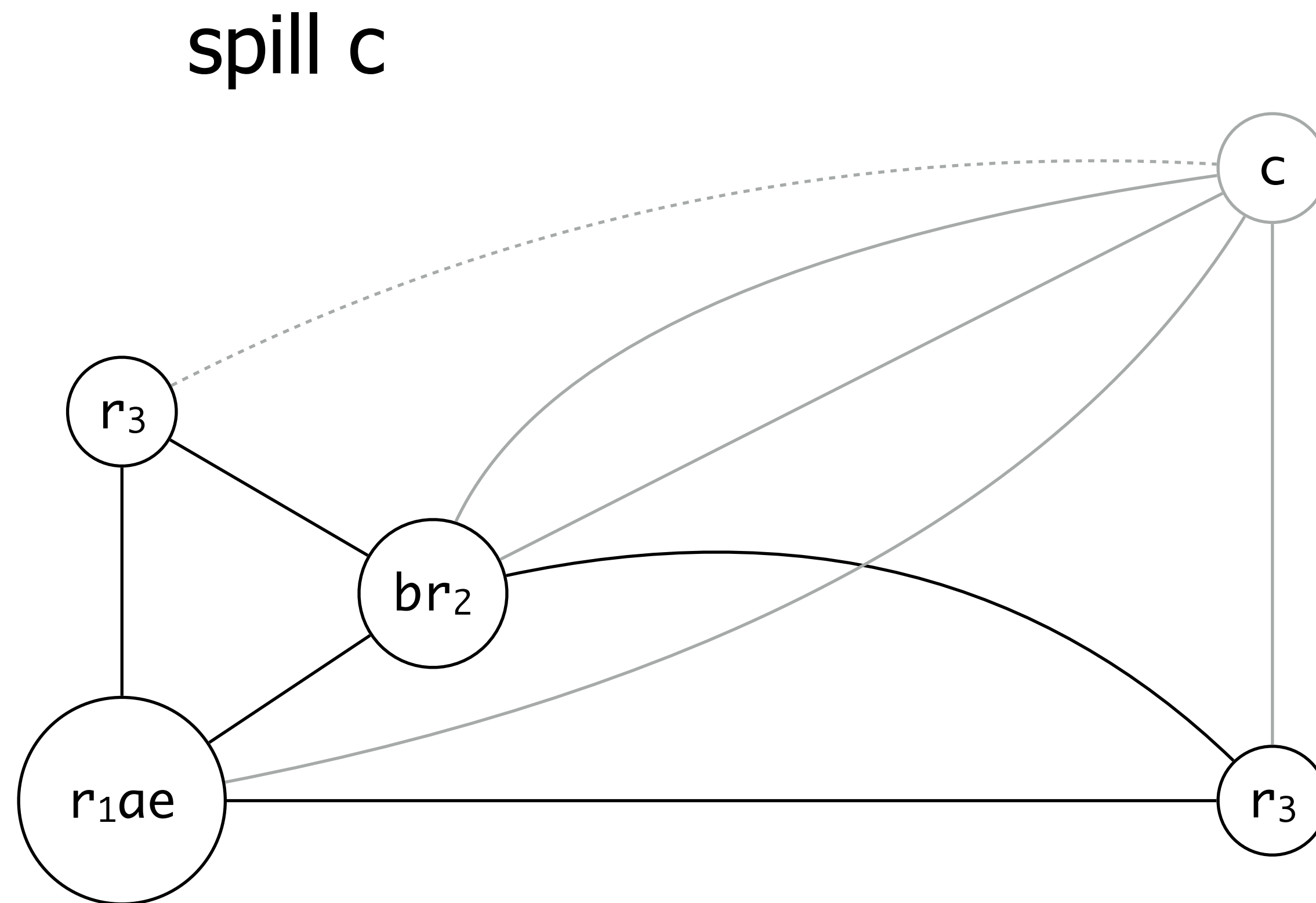
Pre-Colored Nodes



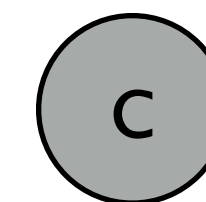
```
enter : c ← r3  
        a ← r1  
        b ← r2  
        d ← ∅  
        e ← a  
loop  : d ← d + b  
        e ← e - 1  
        if e > 0 goto loop  
        r1 ← d  
        r3 ← c  
        return (r1, r3)
```



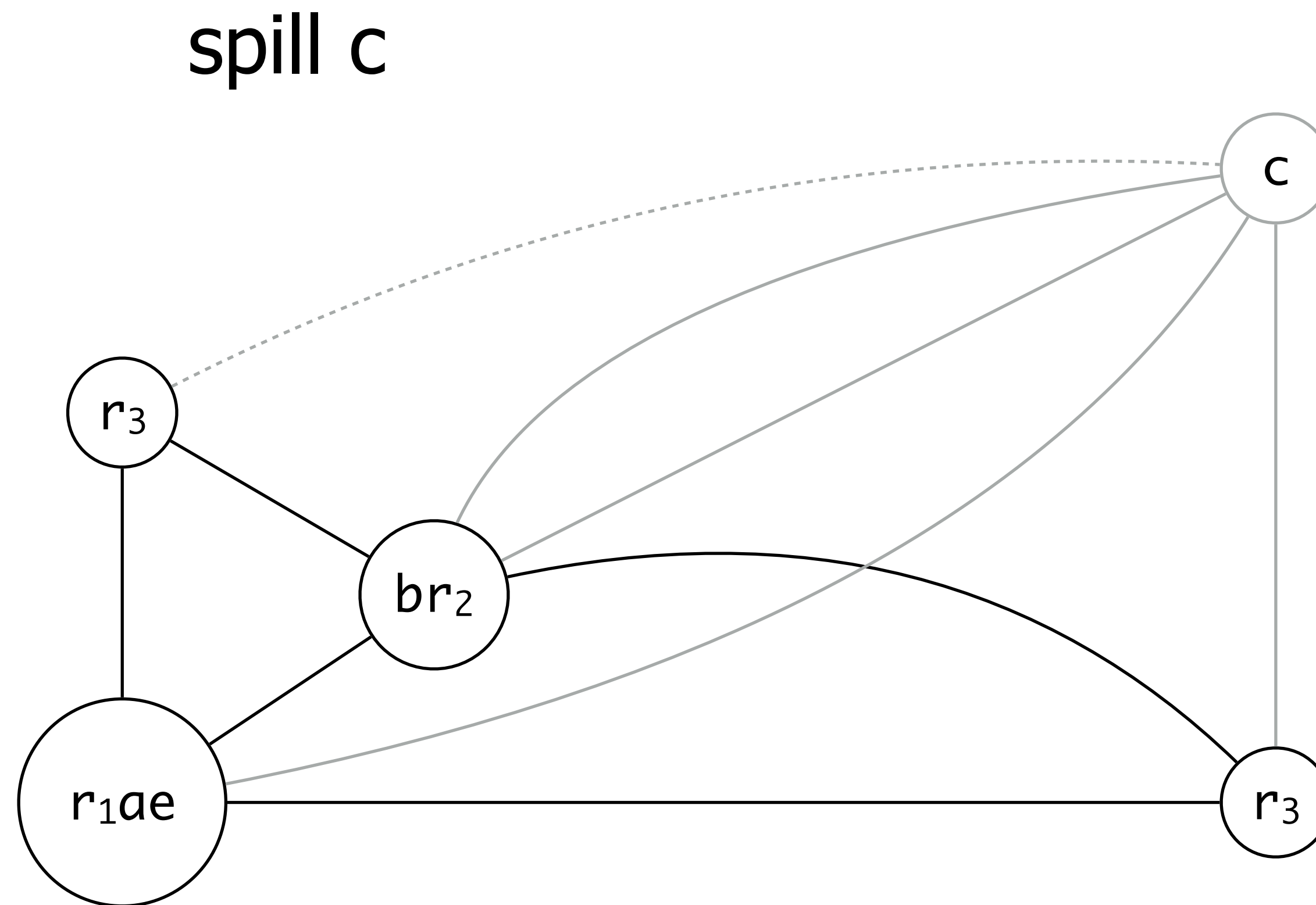
Pre-Colored Nodes



```
enter :  $c_1 \leftarrow r_3$   
         $M[c_{loc}] \leftarrow c_1$   
         $a \leftarrow r_1$   
         $b \leftarrow r_2$   
         $d \leftarrow 0$   
         $e \leftarrow a$   
loop :   $d \leftarrow d + b$   
         $e \leftarrow e - 1$   
        if  $e > 0$  goto loop  
         $r_1 \leftarrow d$   
         $r_3 \leftarrow c_2$   
         $c_2 \leftarrow M[c_{loc}]$   
        return ( $r_1, r_3$ )
```



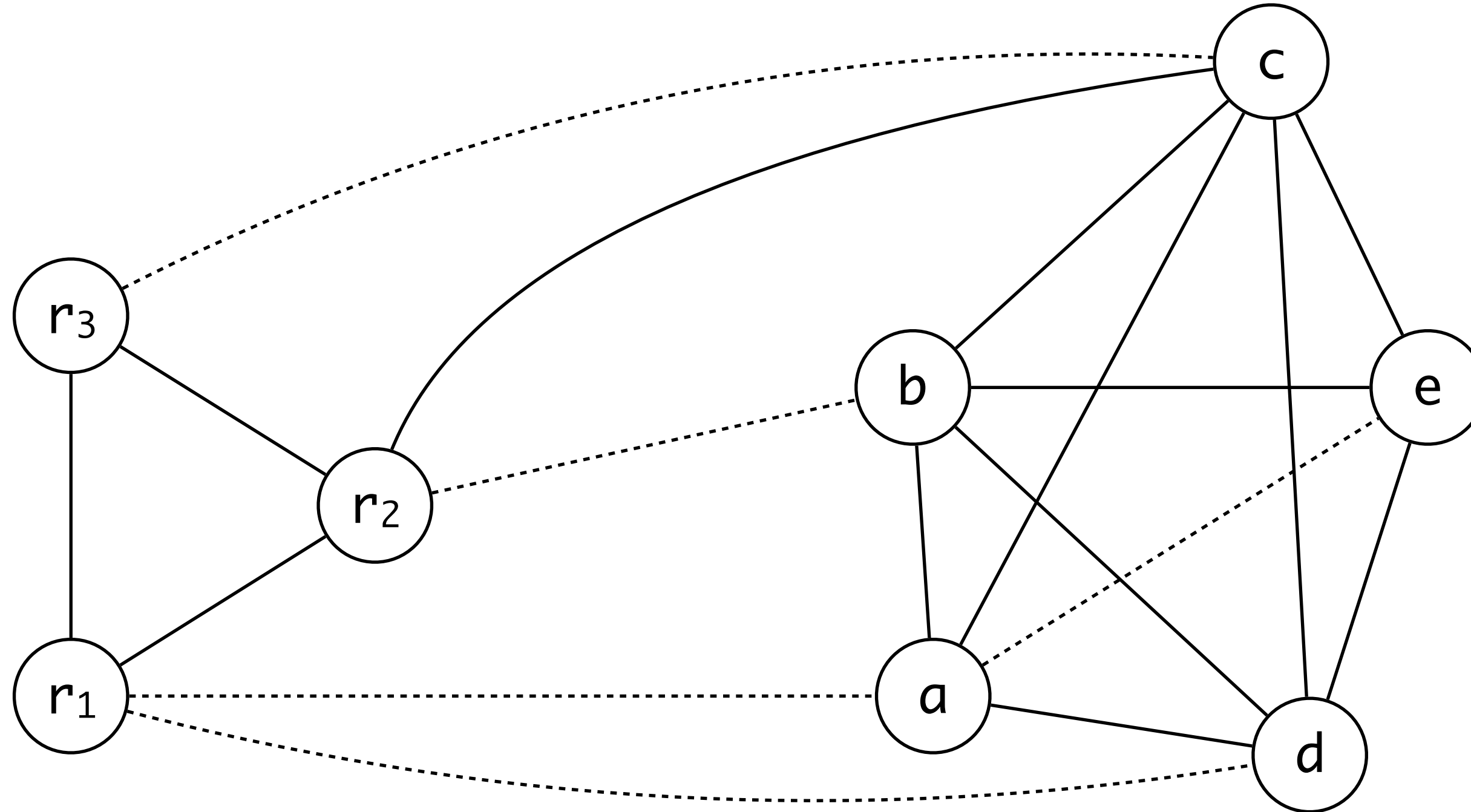
Pre-Colored Nodes



```
enter :  $c_1 \leftarrow r_3$   
         $M[c_{loc}] \leftarrow c_1$   
         $a \leftarrow r_1$   
         $b \leftarrow r_2$   
         $d \leftarrow 0$   
         $e \leftarrow a$   
loop :   $d \leftarrow d + b$   
         $e \leftarrow e - 1$   
        if  $e > 0$  goto loop  
         $r_1 \leftarrow d$   
         $r_3 \leftarrow c_2$   
         $c_2 \leftarrow M[c_{loc}]$   
        return ( $r_1, r_3$ )
```

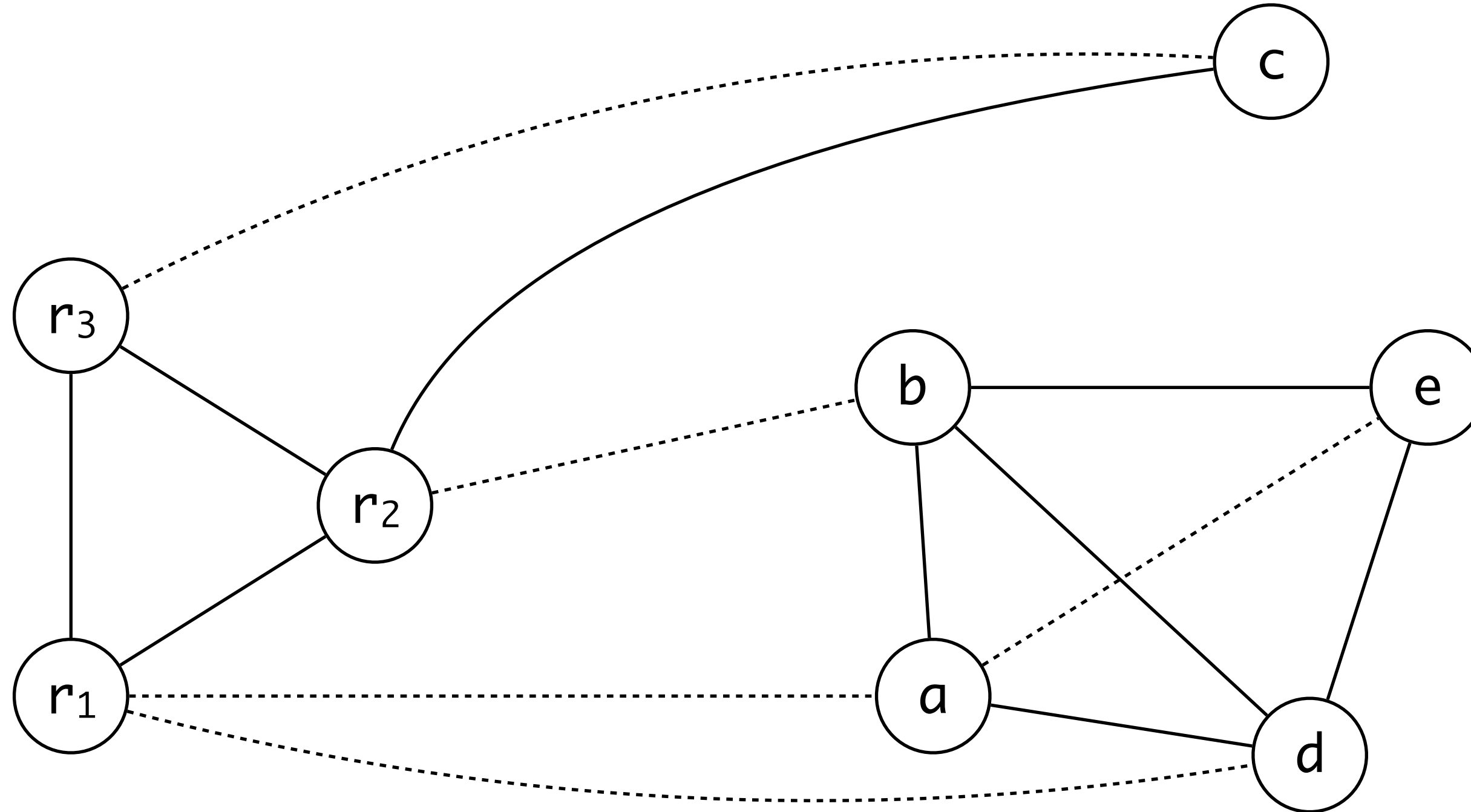

Pre-Colored Nodes

start over



```
enter : c1 ← r3
        M[cloc] ← c1
        a ← r1
        b ← r2
        d ← 0
        e ← a
loop :  d ← d + b
        e ← e - 1
        if e > 0 goto loop
        r1 ← d
        r3 ← c2
        c2 ← M[cloc]
        return (r1, r3)
```

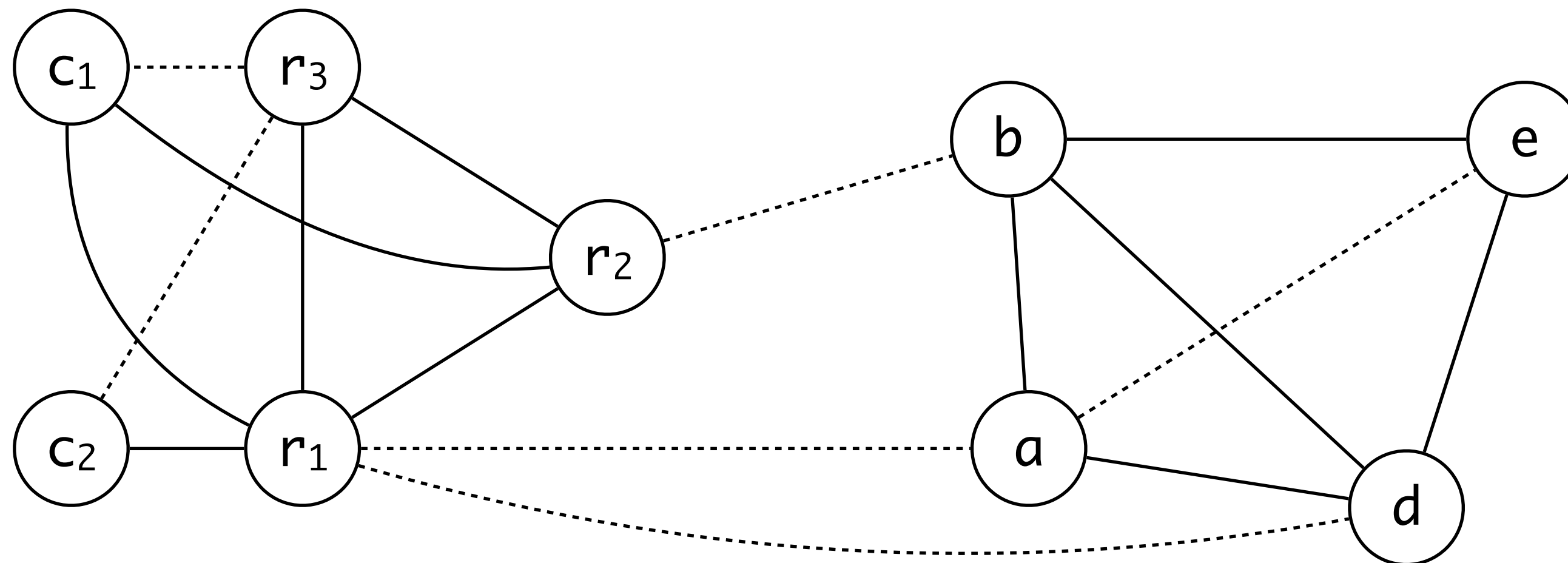
Pre-Colored Nodes



```
enter : c1 ← r3
        M[cloc] ← c1
        a ← r1
        b ← r2
        d ← 0
        e ← a
loop  : d ← d + b
        e ← e - 1
        if e > 0 goto loop
        r1 ← d
        r3 ← c2
        c2 ← M[cloc]
        return (r1, r3)
```

Pre-Colored Nodes

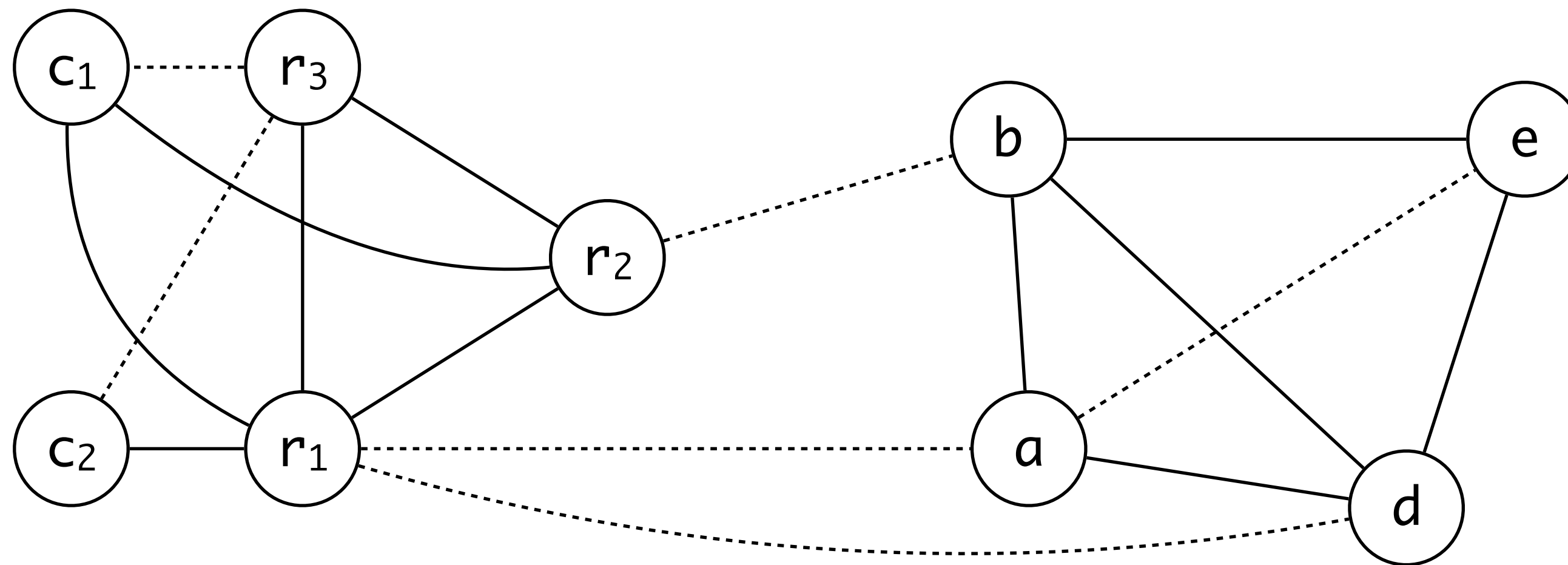
new graph



```
enter : c1 ← r3
        M[cloc] ← c1
        a ← r1
        b ← r2
        d ← 0
        e ← a
loop  : d ← d + b
        e ← e - 1
        if e > 0 goto loop
        r1 ← d
        r3 ← c2
        c2 ← M[cloc]
        return (r1, r3)
```

Pre-Colored Nodes

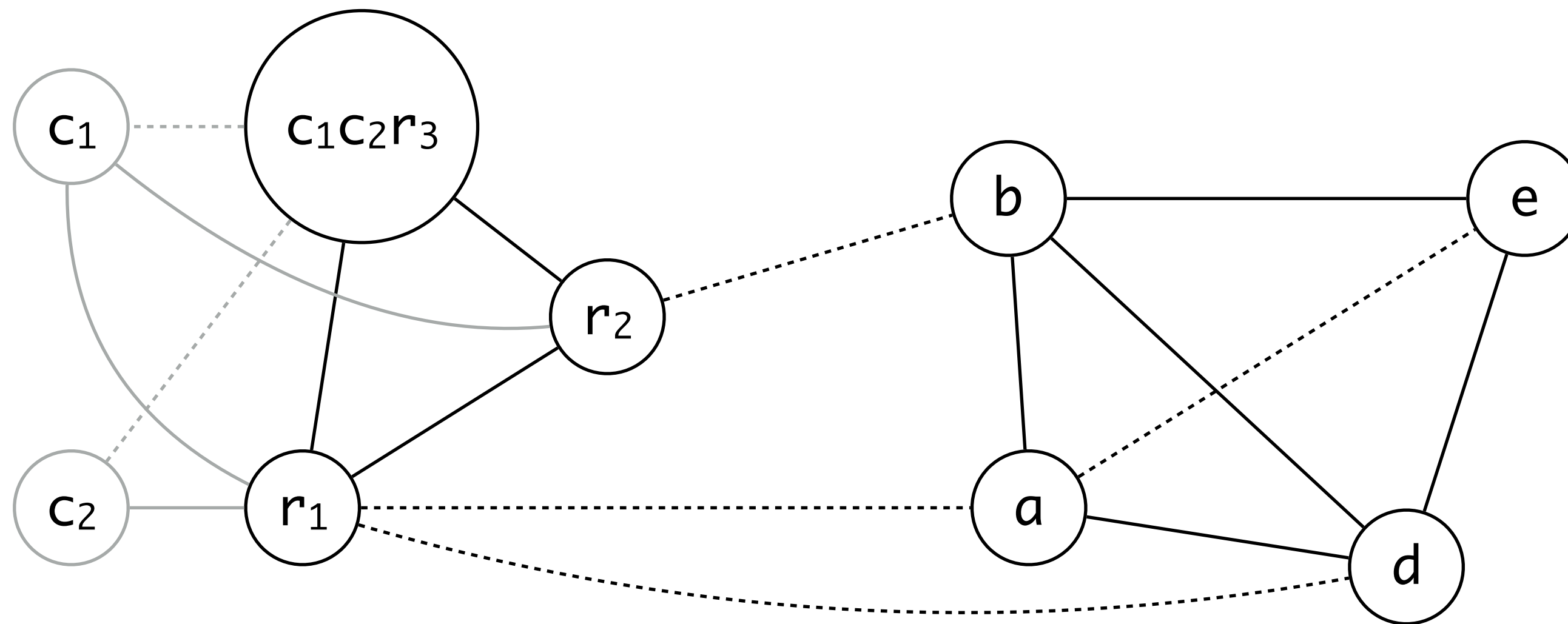
coalesce c_1, c_2, r_3



```
enter :  $c_1 \leftarrow r_3$   
         $M[c_{loc}] \leftarrow c_1$   
         $a \leftarrow r_1$   
         $b \leftarrow r_2$   
         $d \leftarrow 0$   
         $e \leftarrow a$   
loop :   $d \leftarrow d + b$   
         $e \leftarrow e - 1$   
        if  $e > 0$  goto loop  
         $r_1 \leftarrow d$   
         $r_3 \leftarrow c_2$   
         $c_2 \leftarrow M[c_{loc}]$   
        return ( $r_1, r_3$ )
```

Pre-Colored Nodes

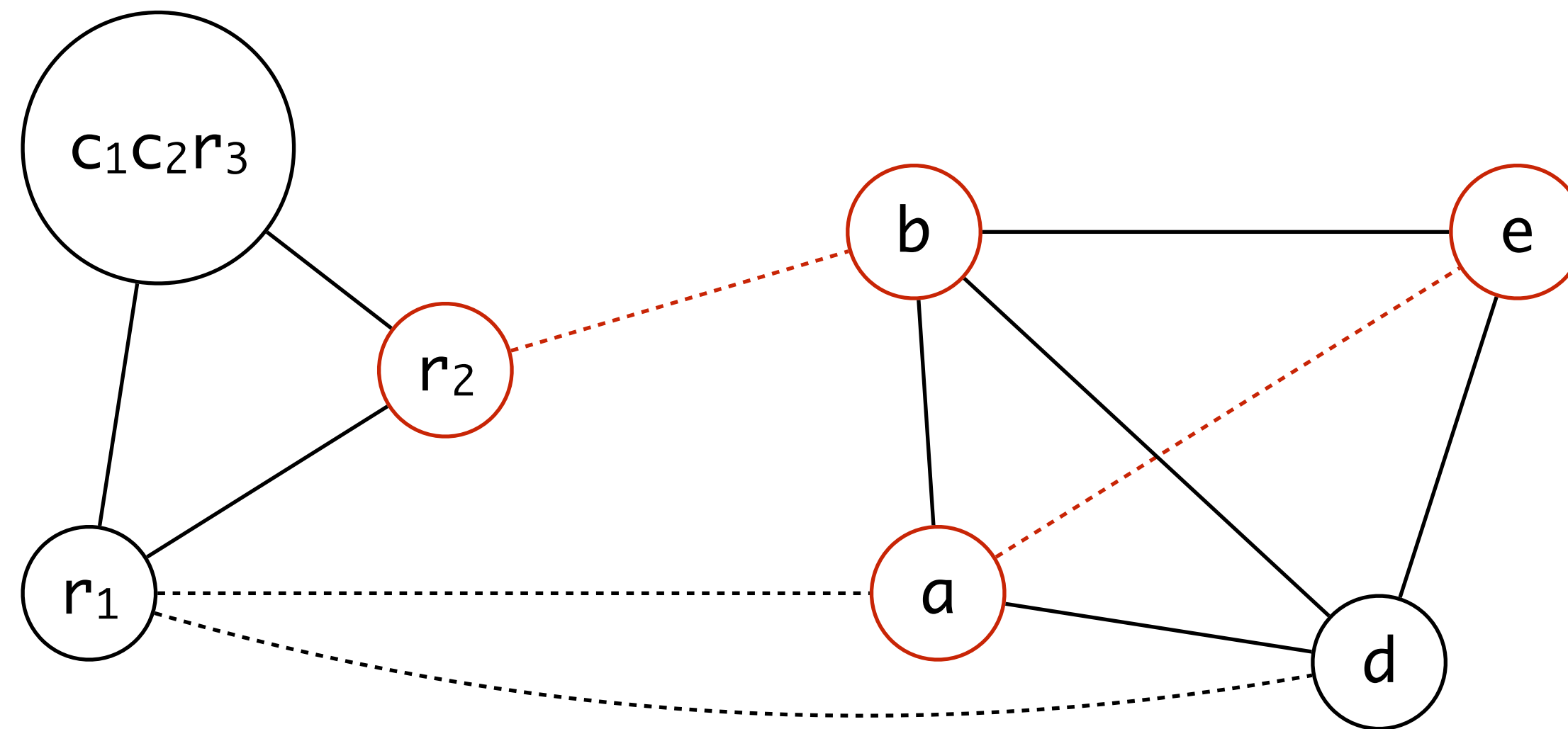
coalesce c_1, c_2, r_3



```
enter :  $c_1 \leftarrow r_3$   
         $M[c_{loc}] \leftarrow c_1$   
         $a \leftarrow r_1$   
         $b \leftarrow r_2$   
         $d \leftarrow \emptyset$   
         $e \leftarrow a$   
loop :   $d \leftarrow d + b$   
         $e \leftarrow e - 1$   
        if  $e > 0$  goto loop  
         $r_1 \leftarrow d$   
         $r_3 \leftarrow c_2$   
         $c_2 \leftarrow M[c_{loc}]$   
        return ( $r_1, r_3$ )
```

Pre-Colored Nodes

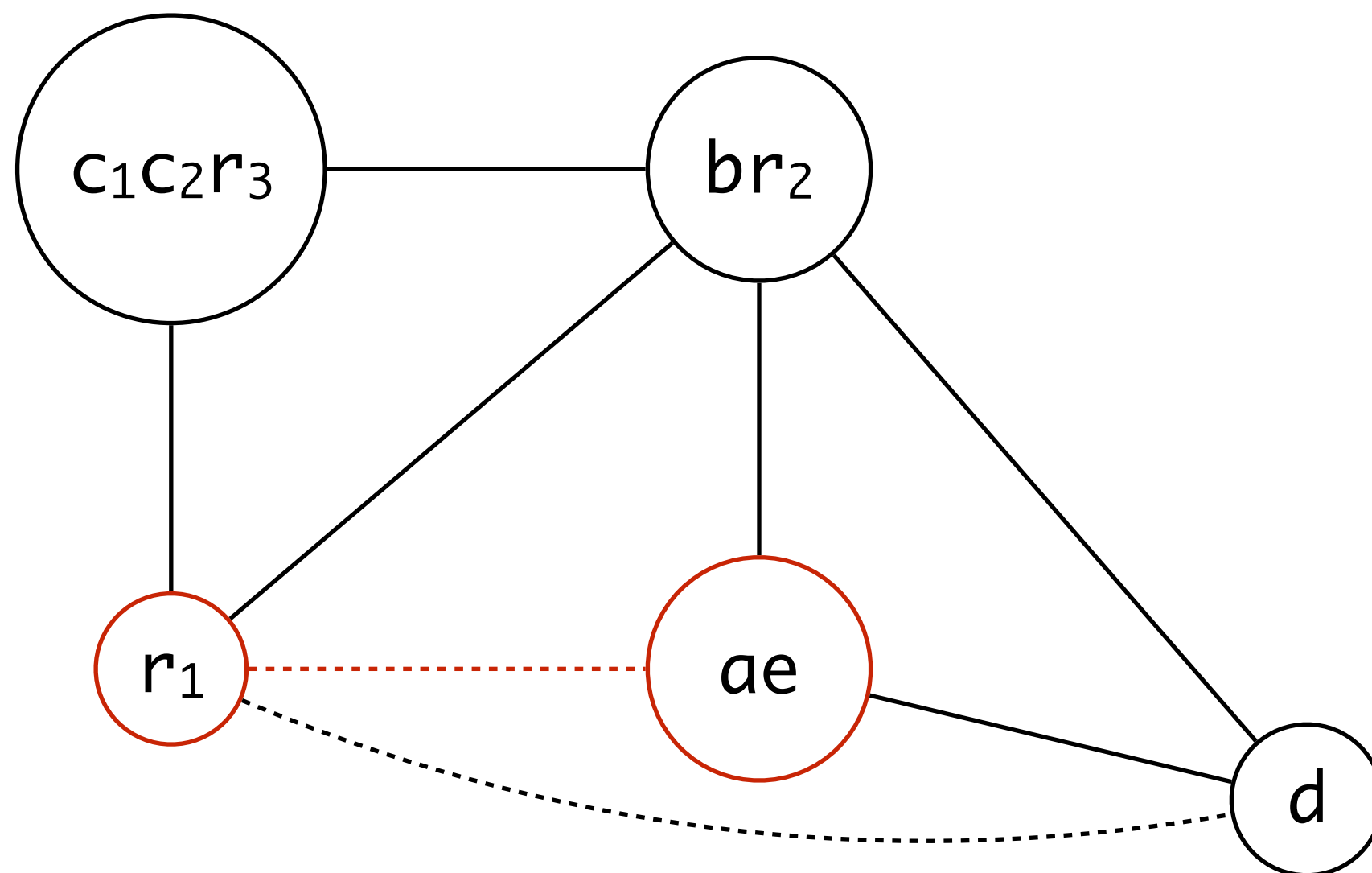
coalesce (b, r₂) and (a, e)



```
enter :  $c_1 \leftarrow r_3$   
         $M[c_{loc}] \leftarrow c_1$   
         $a \leftarrow r_1$   
         $b \leftarrow r_2$   
         $d \leftarrow 0$   
         $e \leftarrow a$   
loop :   $d \leftarrow d + b$   
         $e \leftarrow e - 1$   
        if  $e > 0$  goto loop  
         $r_1 \leftarrow d$   
         $r_3 \leftarrow c_2$   
         $c_2 \leftarrow M[c_{loc}]$   
        return ( $r_1, r_3$ )
```

Pre-Colored Nodes

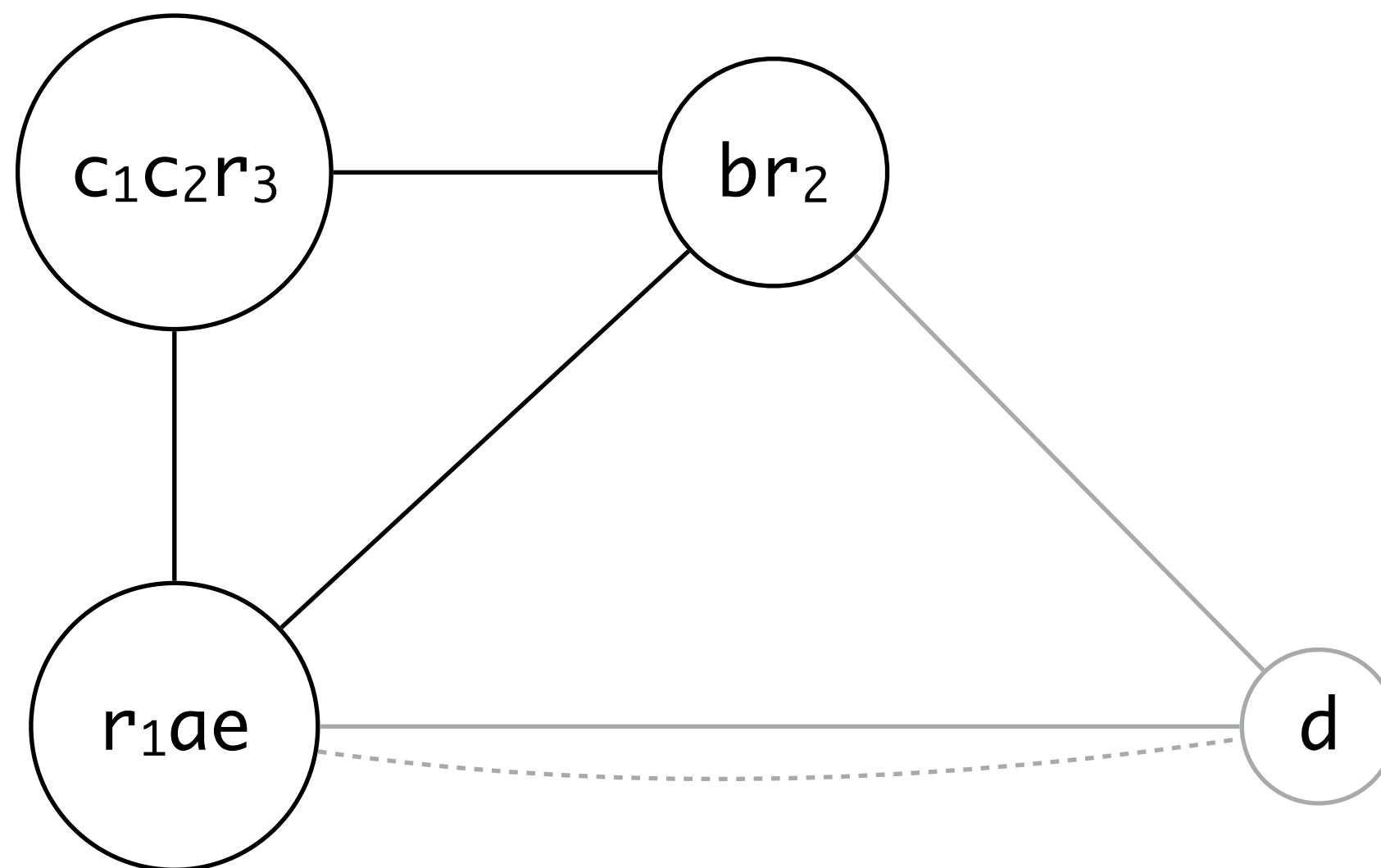
coalesce (ae, r₁)



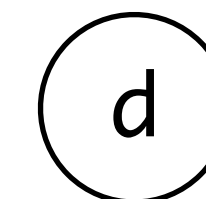
```
enter : c1 ← r3
        M[cloc] ← c1
        a ← r1
        b ← r2
        d ← 0
        e ← a
loop  : d ← d + b
        e ← e - 1
        if e > 0 goto loop
        r1 ← d
        r3 ← c2
        c2 ← M[cloc]
        return (r1, r3)
```

Pre-Colored Nodes

simplify d

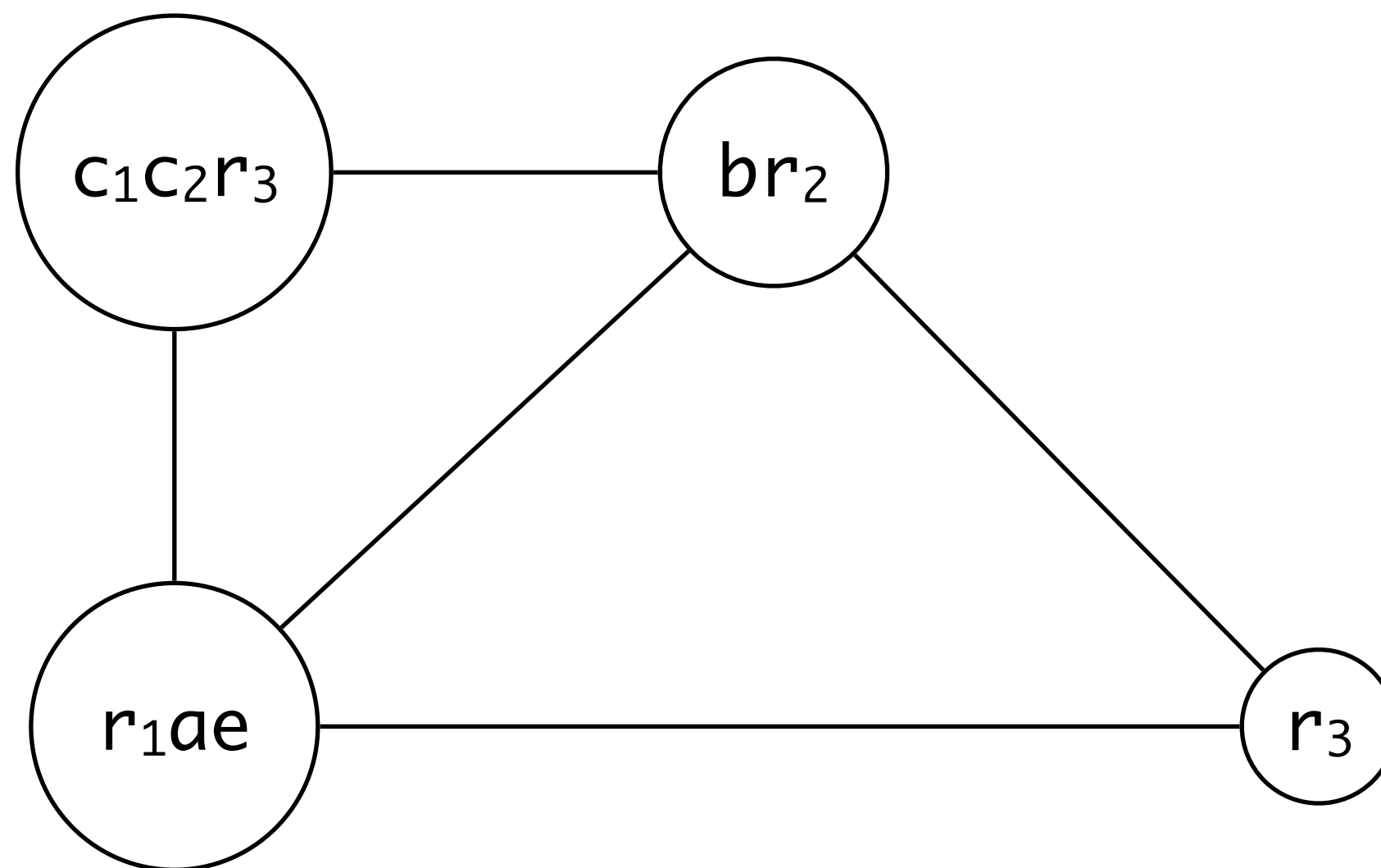


```
enter : c1 ← r3  
        M[cloc] ← c1  
        a ← r1  
        b ← r2  
        d ← 0  
        e ← a  
loop  : d ← d + b  
        e ← e - 1  
        if e > 0 goto loop  
        r1 ← d  
        r3 ← c2  
        c2 ← M[cloc]  
        return (r1, r3)
```

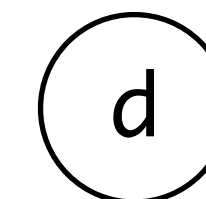


Pre-Colored Nodes

color d as r_3

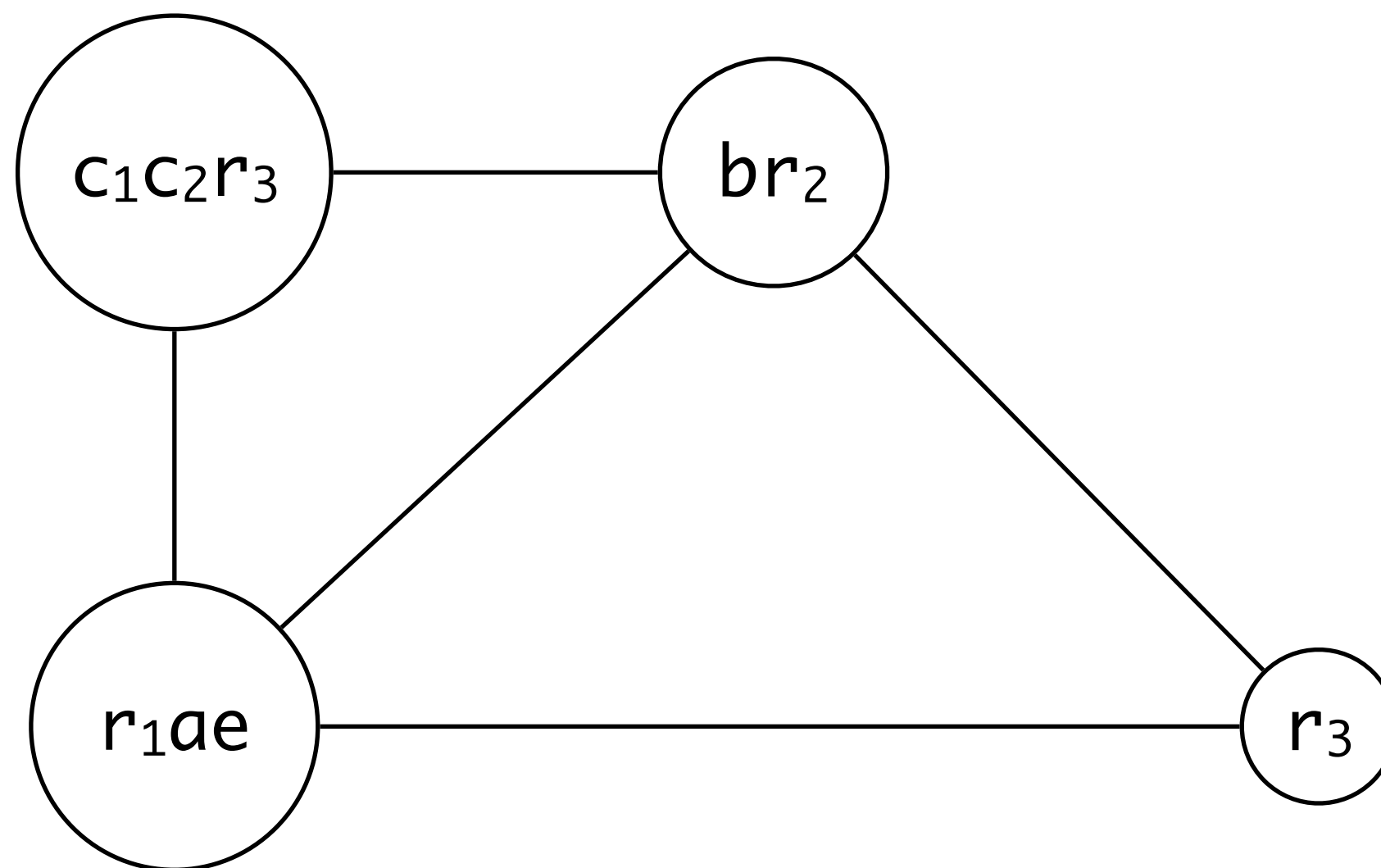


```
enter : c1 ← r3  
        M[cloc] ← c1  
        a ← r1  
        b ← r2  
        d ← 0  
        e ← a  
loop  : d ← d + b  
        e ← e - 1  
        if e > 0 goto loop  
        r1 ← d  
        r3 ← c2  
        c2 ← M[cloc]  
        return (r1, r3)
```



Pre-Colored Nodes

apply register assignment



```
enter : r3 ← r3  
        M[cloc] ← r3  
        r1 ← r1  
        r2 ← r2  
        r3 ← 0  
        r1 ← r1  
loop  : r3 ← r3 + r2  
        r1 ← r1 - 1  
        if r1 > 0 goto loop  
        r1 ← r3  
        r3 ← r3  
        r3 ← M[cloc]  
        return (r1, r3)
```

Pre-Colored Nodes

```
enter : c ← r3
        a ← r1
        b ← r2
        d ← 0
        e ← a
loop :  d ← d + b
        e ← e - 1
        if e > 0 goto loop
        r1 ← d
        r3 ← c
        return (r1, r3)
```

```
enter : r3 ← r3
        M[cloc] ← r3
        r1 ← r1
        r2 ← r2
        r3 ← 0
        r1 ← r1
loop :  r3 ← r3 + r2
        r1 ← r1 - 1
        if r1 > 0 goto loop
        r1 ← r3
        r3 ← r3
        r3 ← M[cloc]
        return (r1, r3)
```

Pre-Colored Nodes

```
enter : c ← r3
        a ← r1
        b ← r2
        d ← 0
        e ← a
loop :  d ← d + b
        e ← e - 1
        if e > 0 goto loop
        r1 ← d
        r3 ← c
        return (r1, r3)
```

```
enter : r3 ← r3
        M[cloc] ← r3
        r1 ← r1
        r2 ← r2
        r3 ← 0
        r1 ← r1
loop :  r3 ← r3 + r2
        r1 ← r1 - 1
        if r1 > 0 goto loop
        r1 ← r3
        r3 ← r3
        r3 ← M[cloc]
        return (r1, r3)
```

Pre-Colored Nodes

```
enter : c ← r3
        a ← r1
        b ← r2
        d ← 0
        e ← a
loop :  d ← d + b
        e ← e - 1
        if e > 0 goto loop
        r1 ← d
        r3 ← c
        return (r1, r3)
```

```
enter : M[cloc] ← r3
        r3 ← 0
loop :  r3 ← r3 + r2
        r1 ← r1 - 1
        if r1 > 0 goto loop
        r1 ← r3
        r3 ← M[cloc]
        return (r1, r3)
```

Pre-Colored Nodes

```
int f(int a, int b) {  
    int d = 0;  
    int e = a;  
    do {  
        d = d + b;  
        e = e - 1;  
    } while (e > 0);  
    return d;  
}
```

```
enter : M[cloc] ← r3  
        r3 ← 0  
loop :  r3 ← r3 + r2  
        r1 ← r1 - 1  
        if r1 > 0 goto loop  
        r3 ← M[cloc]  
        return (r1, r3)
```

Summary

Summary

How can we assign registers to local variables and temporaries?

- perform liveness analysis
- build interference graph
- color interference graph

What to do if the graph is not colorable?

- keep local variables in memory

How to handle move instructions efficiently?

- coalesce nodes safely

Literature

Andrew W. Appel, Jens Palsberg: Modern Compiler Implementation in Java, 2nd edition. 2002

Lal George, Andrew W. Appel: Iterative Register Coalescing. POPL 1996

Lal George, Andrew W. Appel: Iterative Register Coalescing. TOPLAS 18(3), 1996

Except where otherwise noted, this work is licensed under

