HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

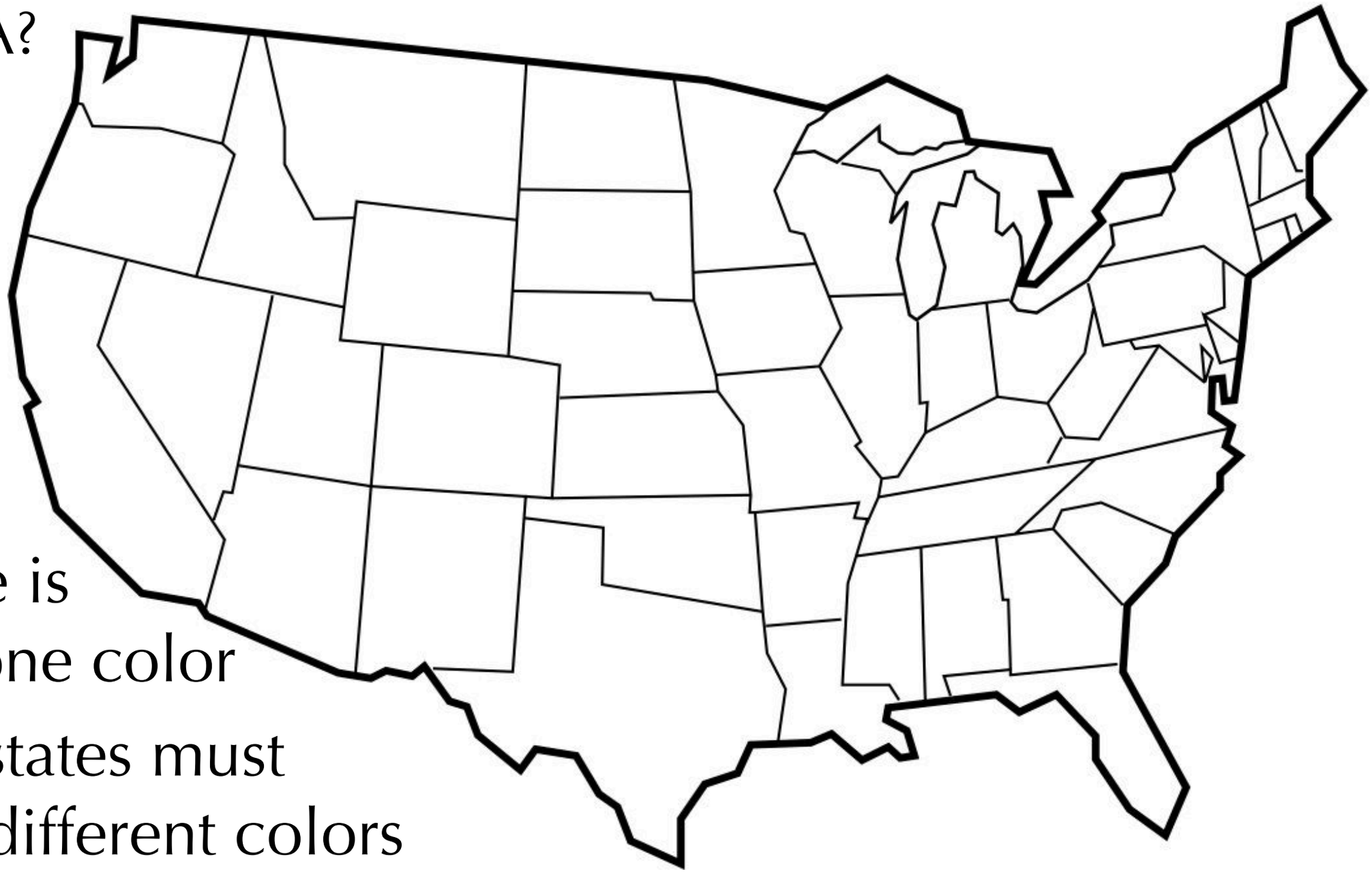
CS153: Compilers
Lecture 20: Register Allocation I

Stephen Chong

https://www.seas.harvard.edu/courses/cs153

Pre-class Puzzle

- What's the minimum number of colors needed to color a map of the USA?

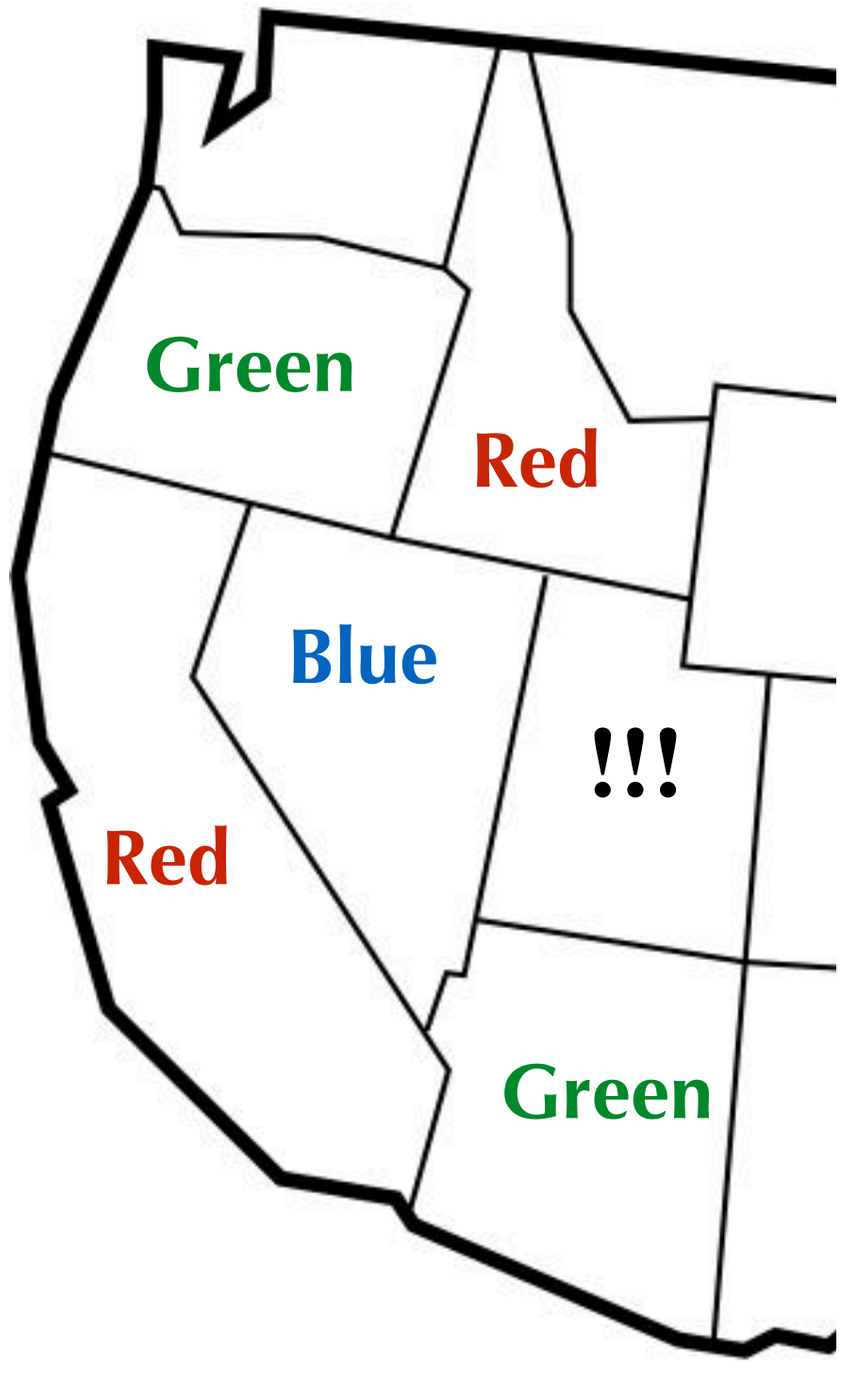


- Every state is assigned one color
- Adjacent states must be given different colors

https://printable-maps.blogspot.com/2011/12/blank-map-of-united-states.html

Pre-class Puzzle Answer

- 4
- Four-color theorem says ≤4
- Must be at least 4:
  - Suppose we had only 3 colors
  - Pick some colors for CA and OR (Red and Green)
  - NV must be Blue
  - ID must be Red
  - AZ must be Green
  - UT!!!!!!



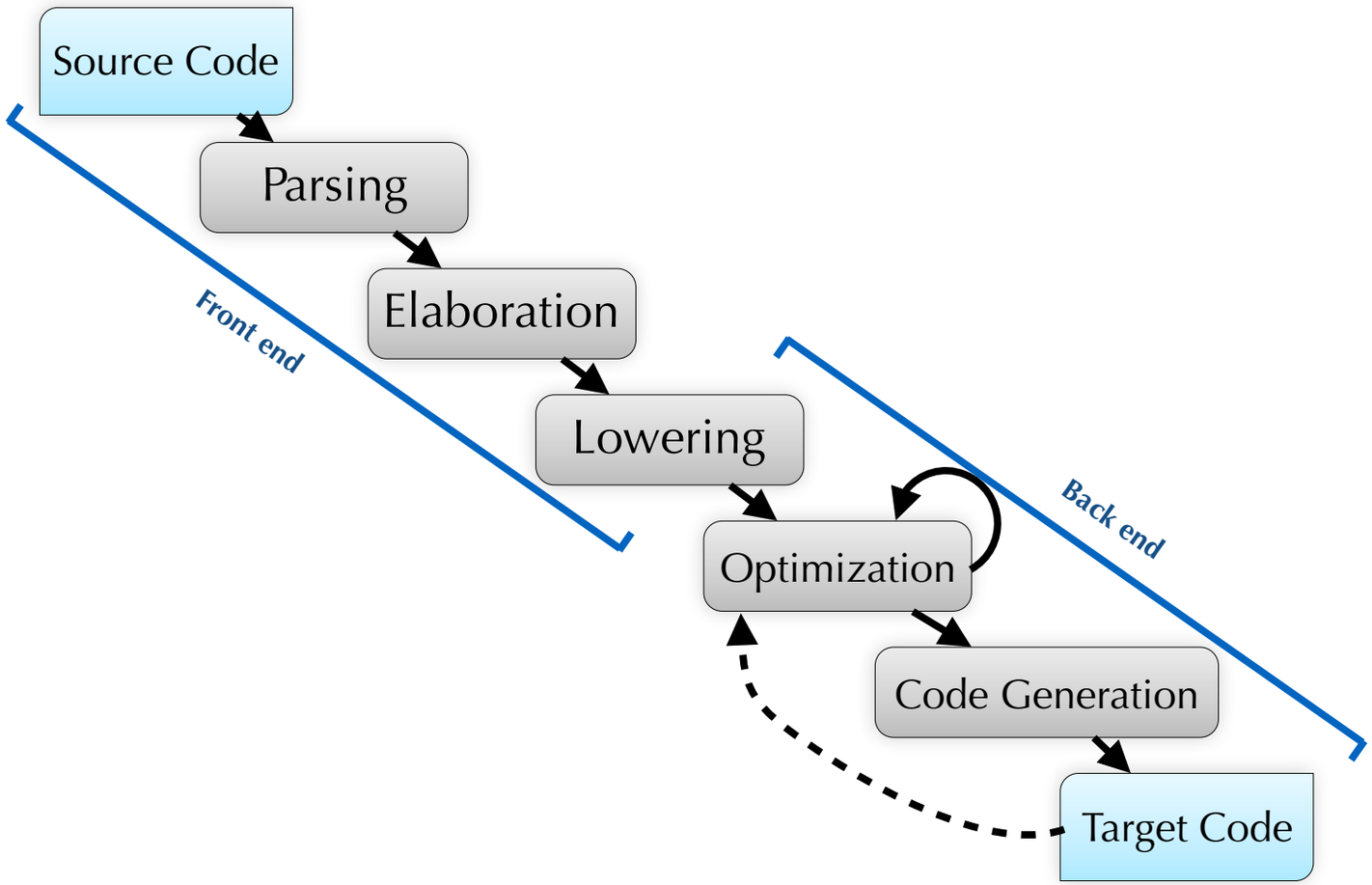
Announcements

- Project 5 out
  - Due Tuesday Nov 13 (5 days)
- Project 6 out
  - Due Tuesday Nov 20 (12 days)
- Project 7 out
  - Due Thursday Nov 29 (21 days)
- Project 8 will be released on Tuesday
  - Due Saturday Dec 8

Today

- Register allocation
  - Graph coloring by simplification
  - Coalescing

Register Allocation



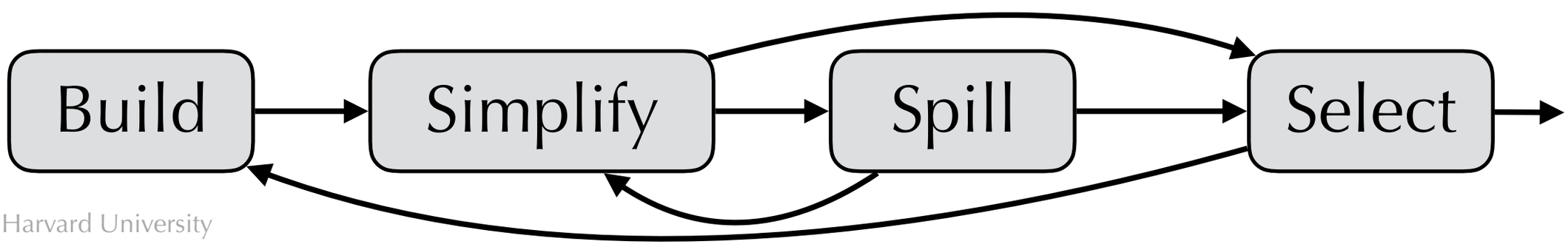
- From an intermediate representation with unlimited number of "temporary"/local variables
- Assign temporary variables to the (small) number of machine registers

Register Allocation

- Register allocation is in generally an NP-complete problem
  - Can we allocate all these *n* temporaries to *k* registers?
- But we have a heuristic that is linear in practice!
  - Based on **graph coloring**
    - Given a graph, can we assign one of *k* colors to each node such that connected nodes have different colors?
  - Here, nodes are temp variables, an edge between t1 and t2 means that `t1` and `t2` are live at the same time. Colors are registers.
- But graph coloring is also NP-complete! How does that work?

Coloring by Simplification

- Four phases
- **Build:** construct interference graph, using dataflow analysis to find for each program point vars that are live at the same time
- **Simplify:** color based on simple heuristic
  - If graph G has node *n* with *k*-1 edges, then G-{n} is *k*-colorable iff G is *k*-colorable
  - So remove nodes with degree <*k*
- **Spill:** if graph has only nodes with degree ≥*k*, choose one to **potentially spill** (i.e., that may need to be saved to stack)
  - Then continue with Simplify
- **Select:** when graph is empty, start restoring nodes in reverse order and color them
  - When we encounter a potential spill node, try coloring it. If we can't, rewrite program to store it to stack after definition and load before use. Try again!

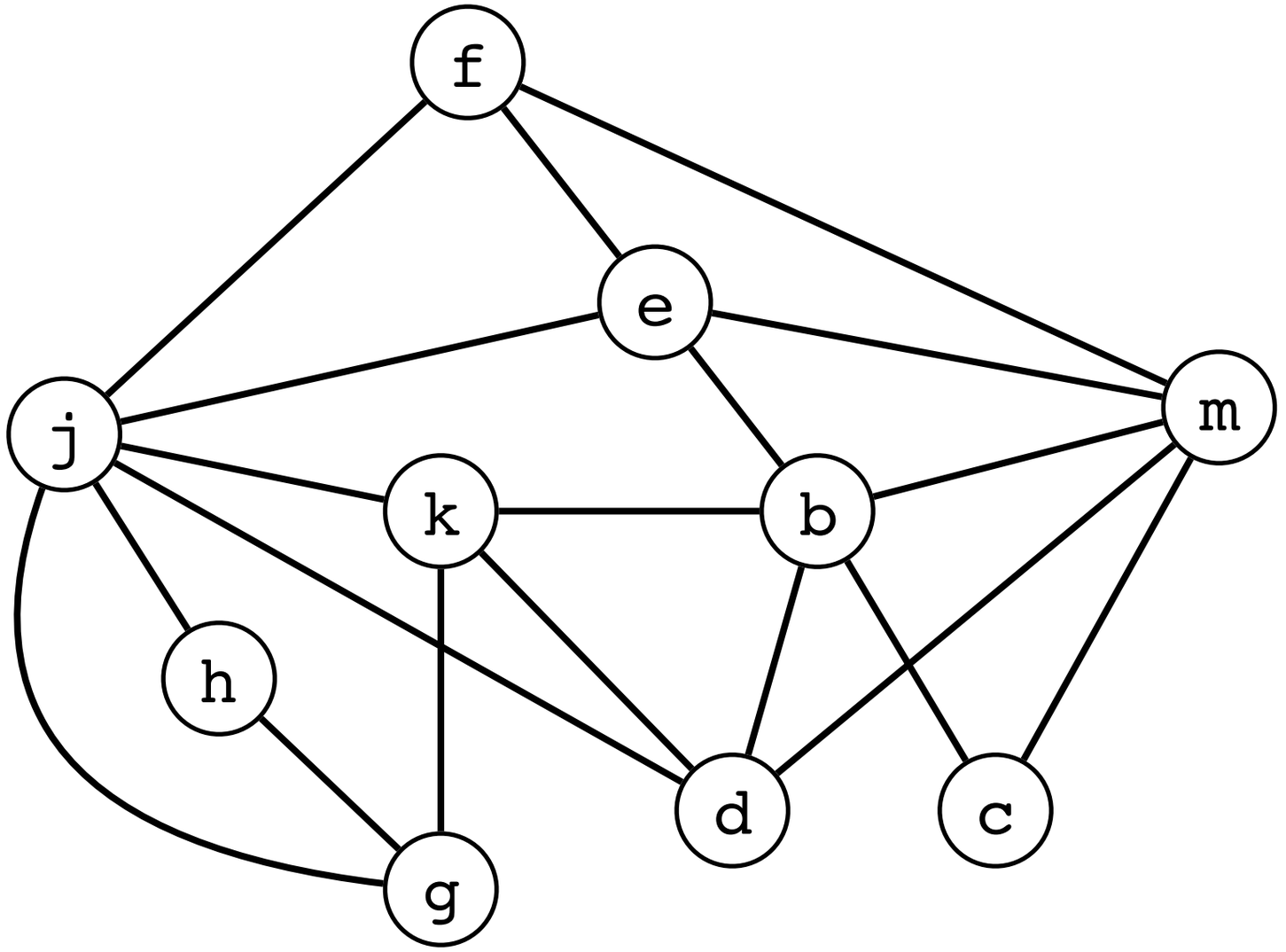


Example

From Appel

```
{live-in: j, k}
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
b := *(f+0)
c := e + 8
d := c
k := m + 4
j := b
{live-out: d,j,k}
```

Interference graph

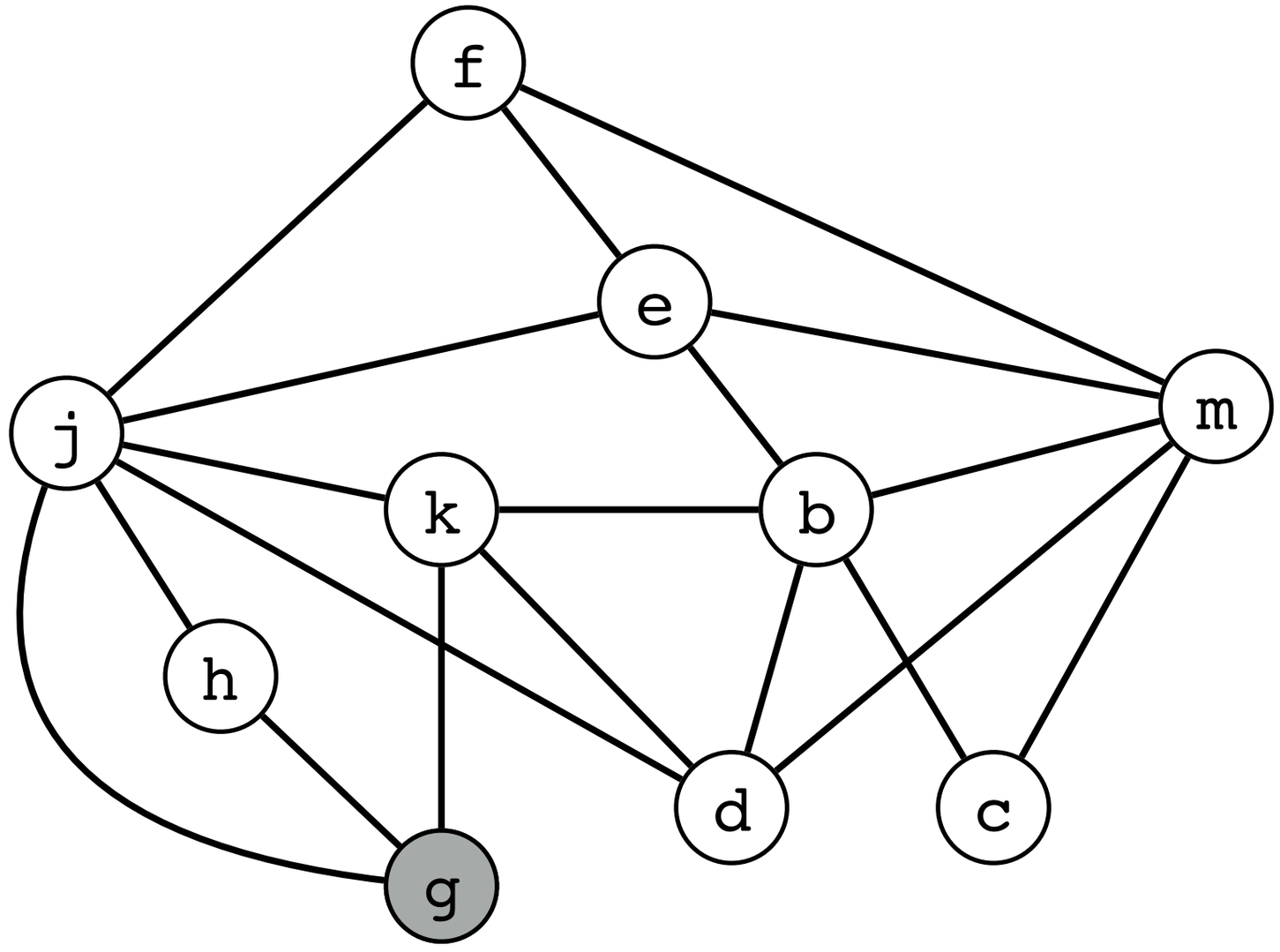


Simplification (4 registers)

Choose any node with degree <4

Stack:

g



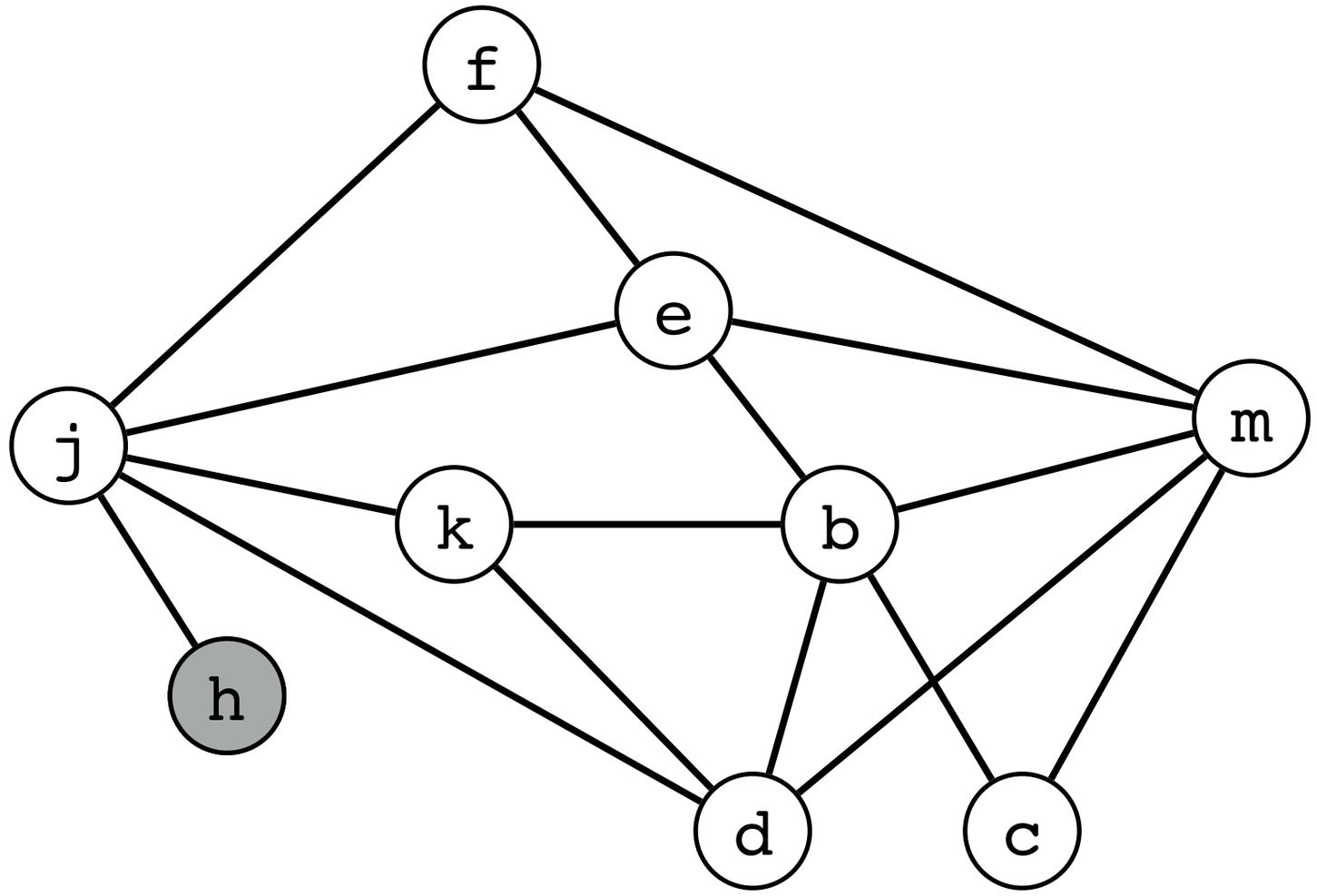
Simplification (4 registers)

Choose any node with degree <4

Stack:

g

h



Simplification (4 registers)

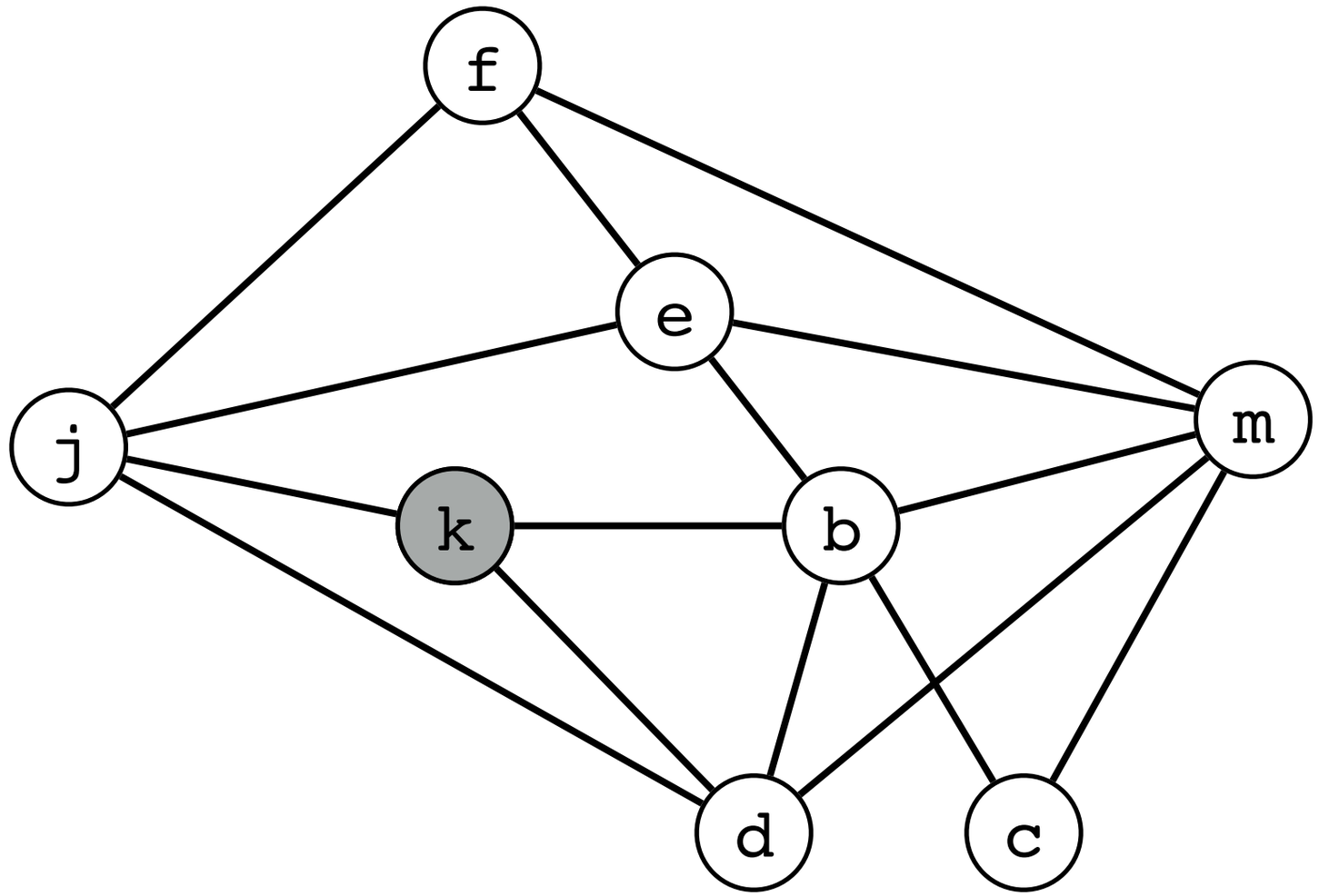
Choose any node with degree <4

Stack:

g

h

k



Simplification (4 registers)

Choose any node with degree <4

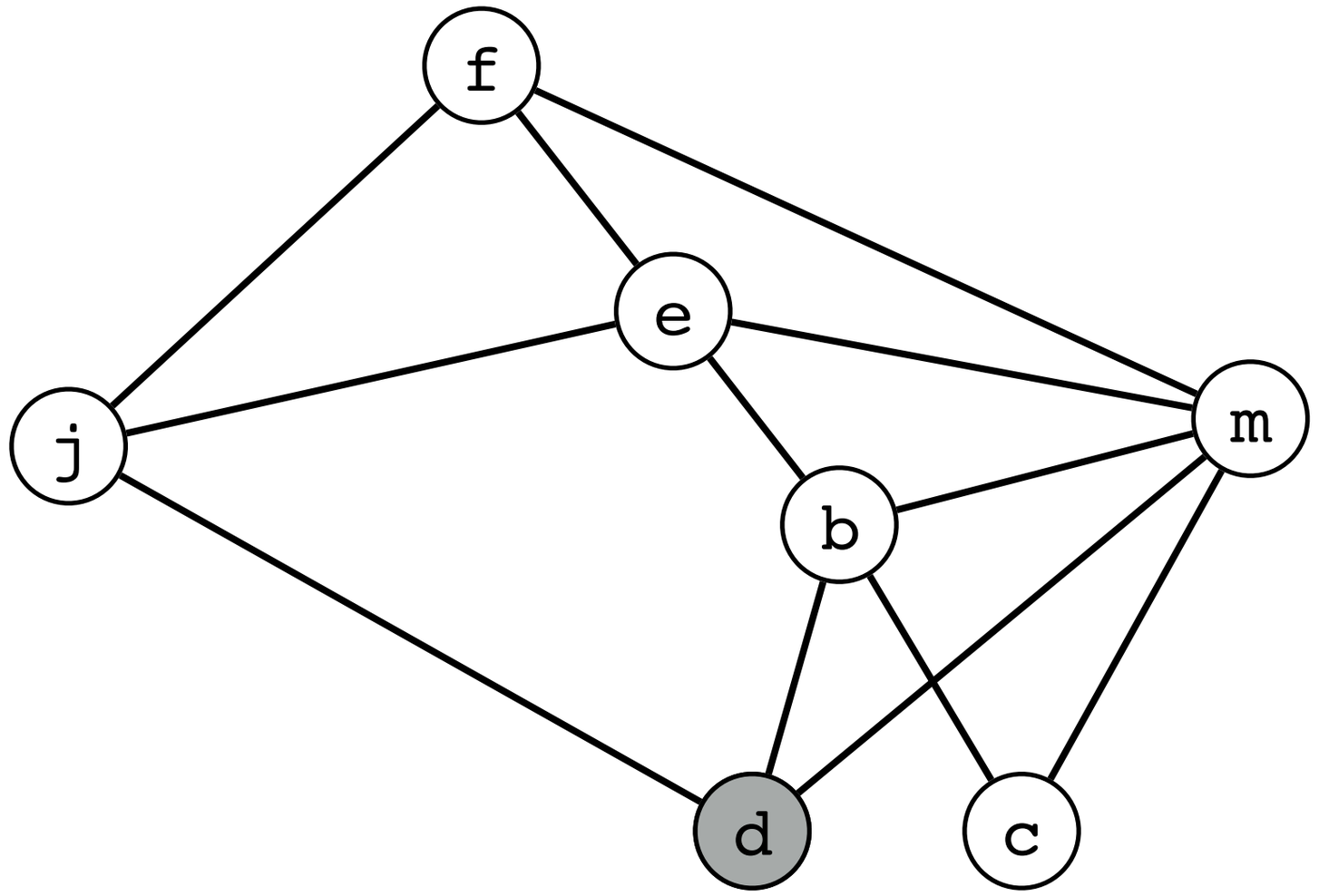
Stack:

g

h

k

d



Simplification (4 registers)

Choose any node with degree <4

Stack:

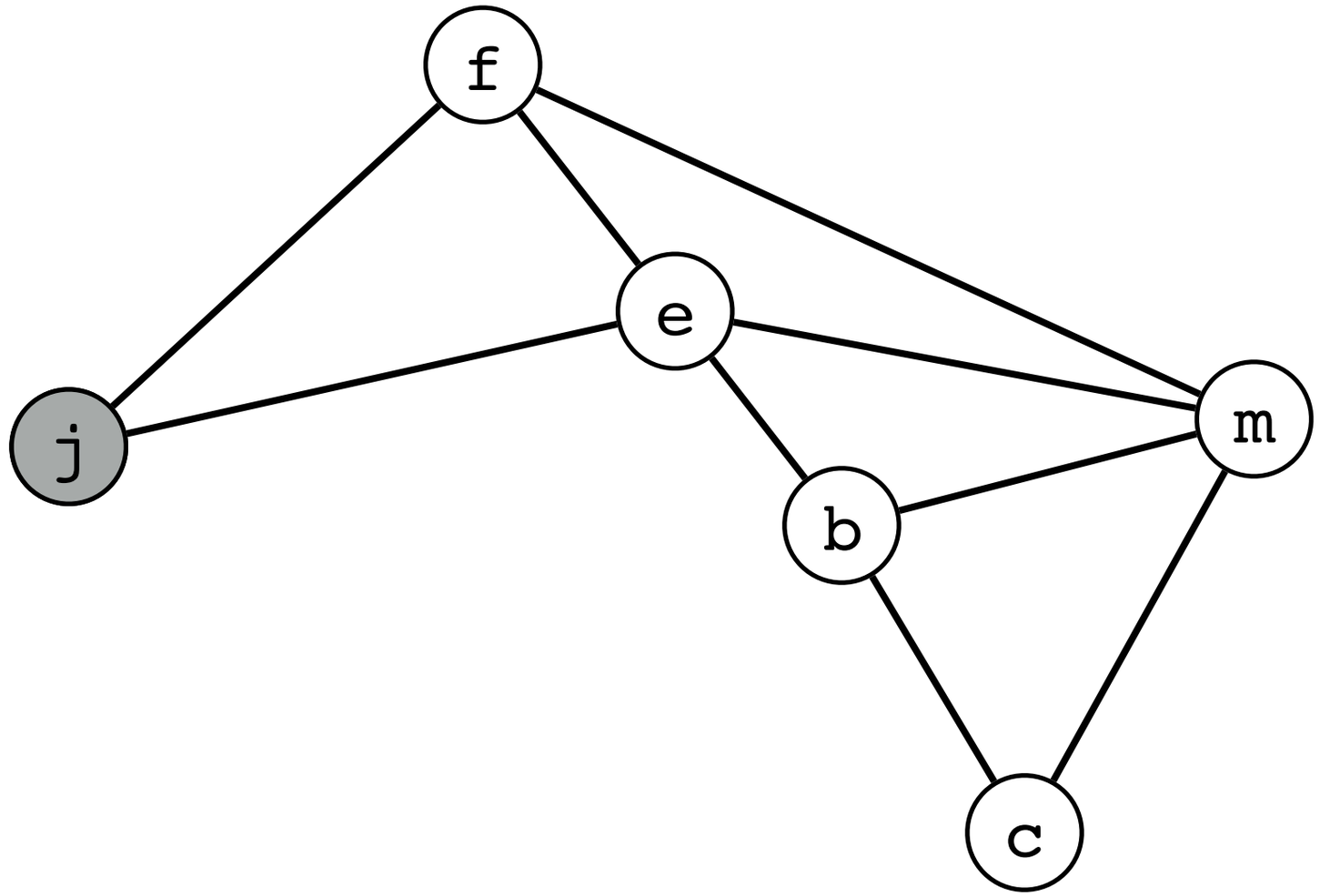
g

h

k

d

j



Simplification (4 registers)

Choose any node with degree <4

Stack:

g

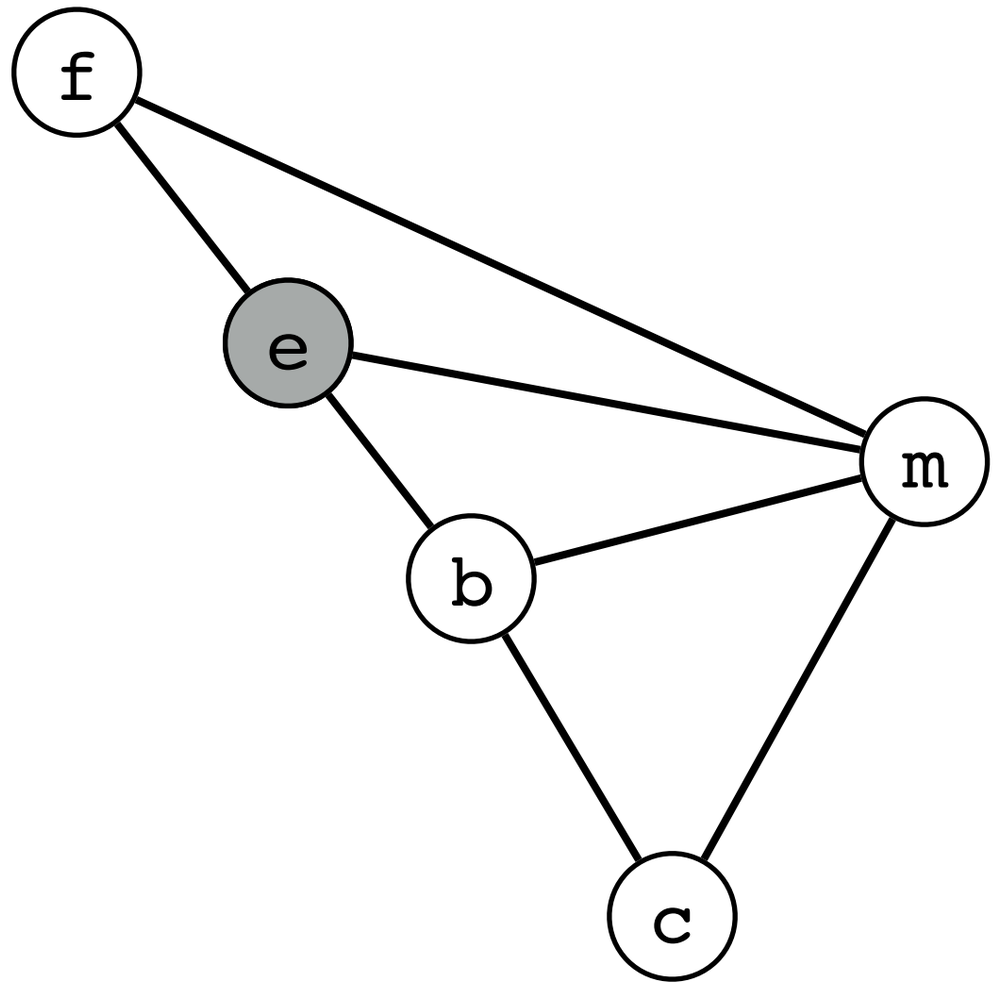
h

k

d

j

e

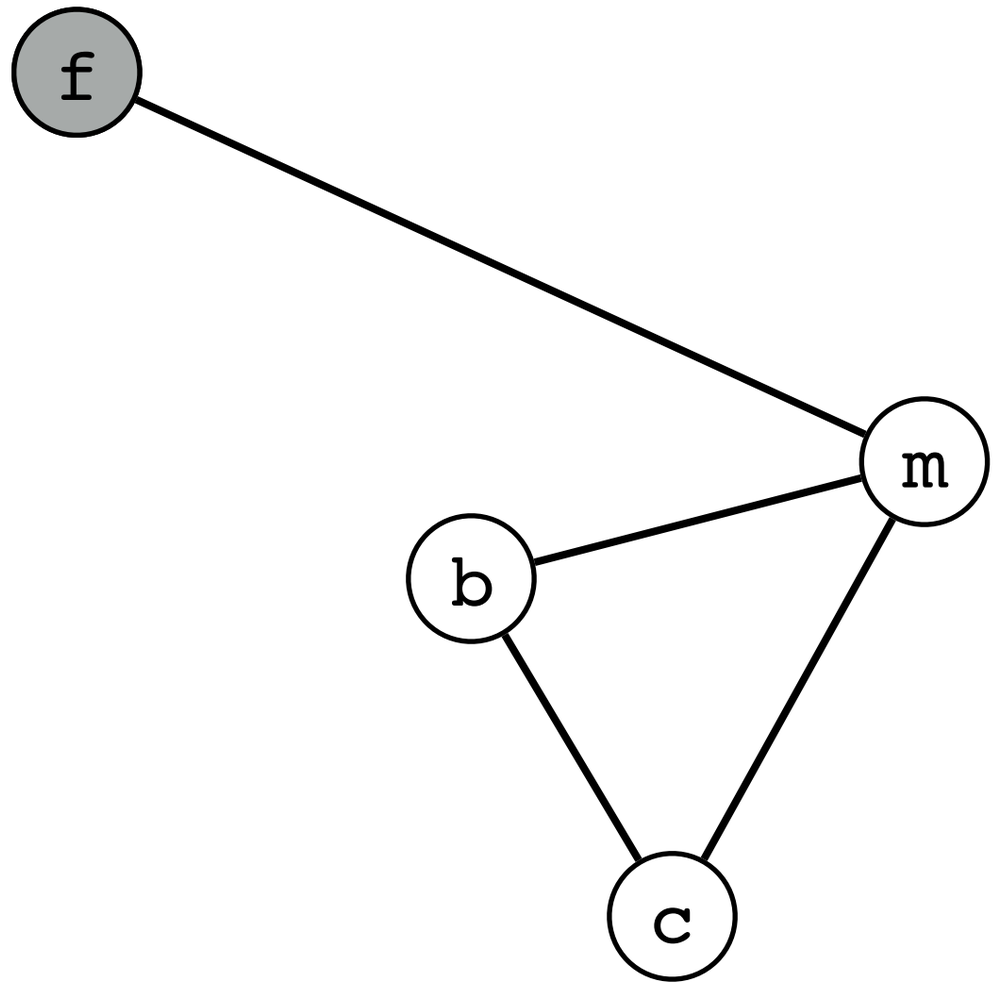


Simplification (4 registers)

Choose any node with degree <4

Stack:

g
h
k
d
j
e
f

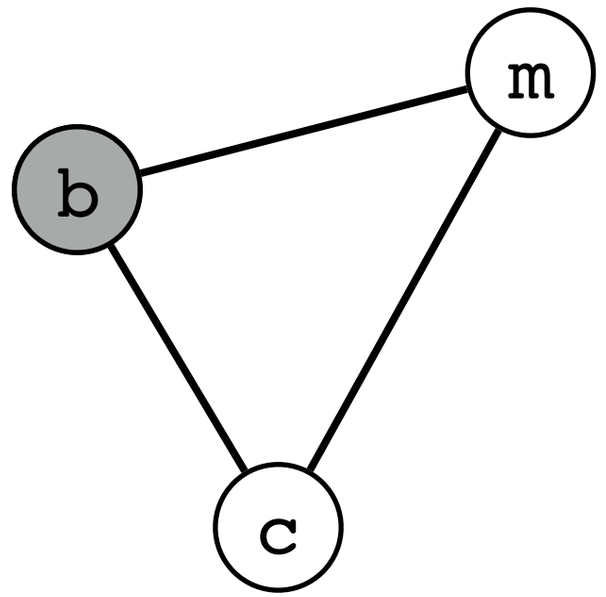


Simplification (4 registers)

Choose any node with degree <4

Stack:

g
h
k
d
j
e
f
b

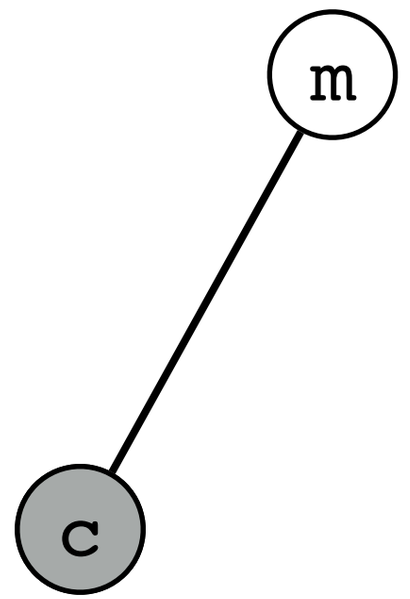


Simplification (4 registers)

Choose any node with degree <4

Stack:

g
h
k
d
j
e
f
b
c



Simplification (4 registers)

Choose any node with degree <4

Stack:

g

h

k

k

d

j

e

f

b

c

m

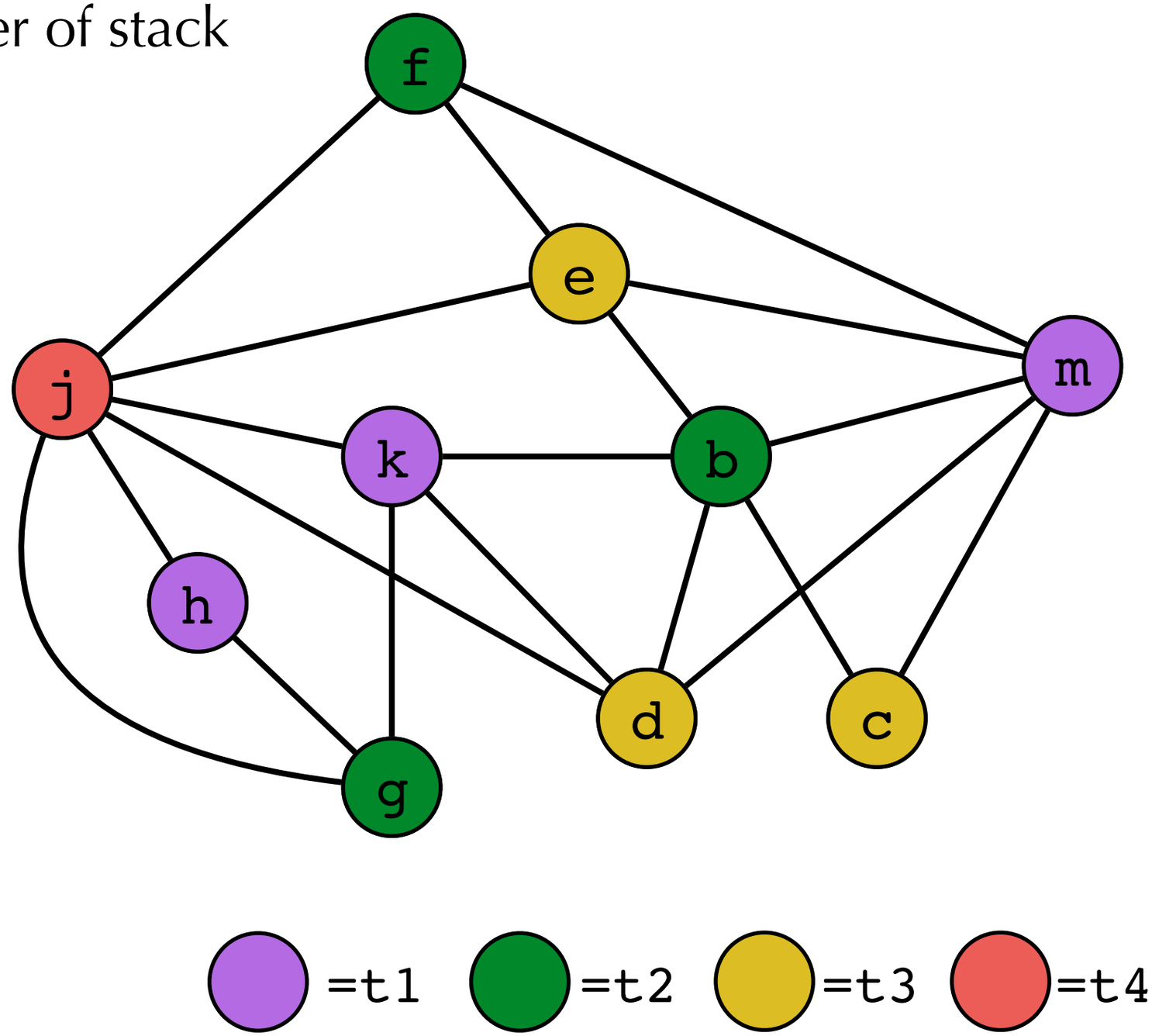
Select (4 registers)

Graph is now empty!

Stack:

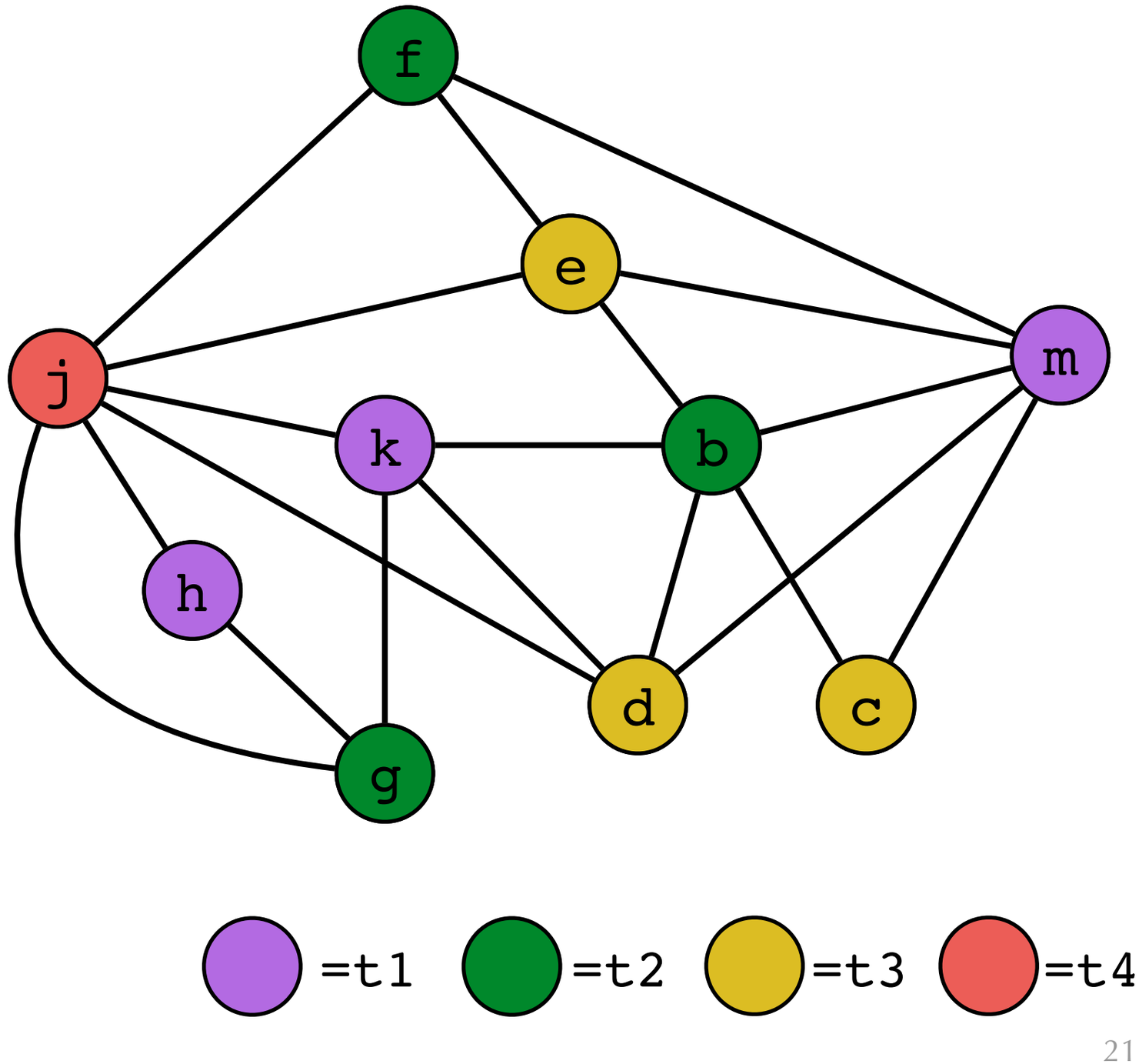
Color nodes in order of stack

g
h
k
d
j
e
f
b
c
m



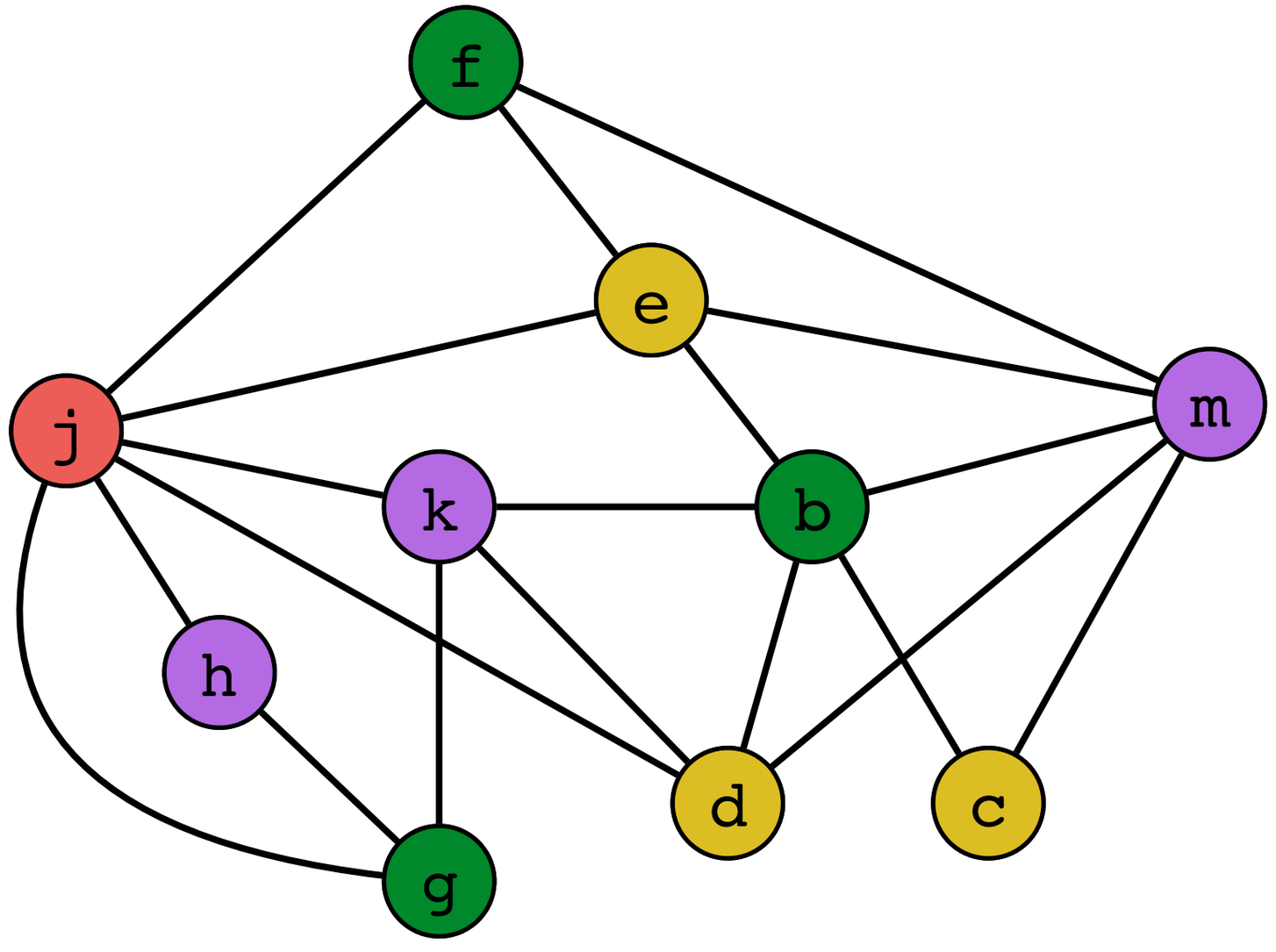
Select (4 registers)

```
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
b := *(f+0)
c := e + 8
d := c
k := m + 4
j := b
```



Select (4 registers)

```
$t2 := *(t4+12)
$t1 := $t1 - 1
$t2 := $t2 * $t1
$t3 := *($t4+8)
$t1 := *($t4+16)
$t2 := *($t2+0)
$t3 := $t3 + 8
$t3 := $t3
$t1 := $t1 + 4
$t4 := $t2
```



Some moves might subsequently be simplified...

Spilling

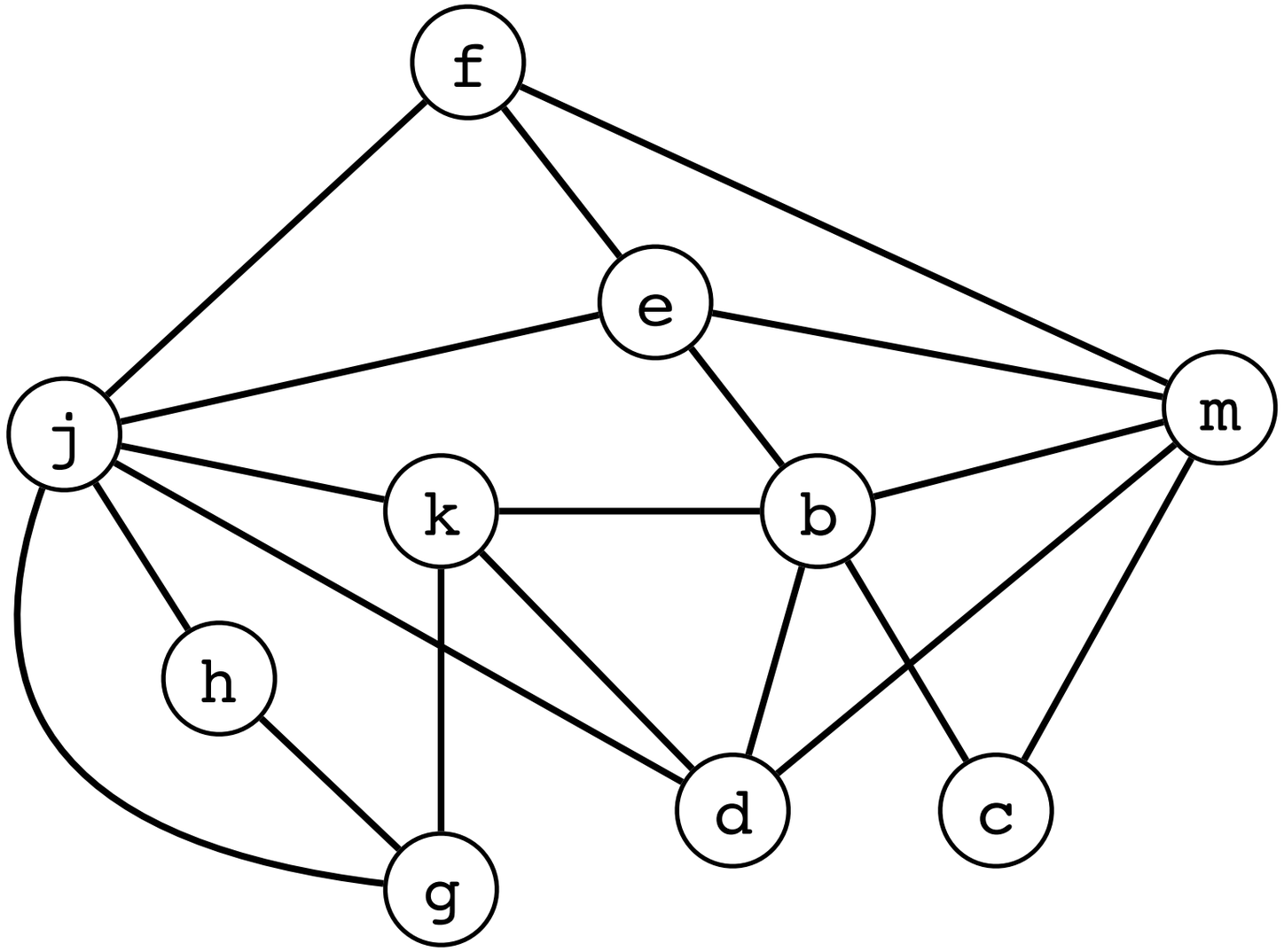
- This example worked out nicely!
- Always had nodes with degree $<k$
- Let's try again, but now with only 3 registers...

Example

From Appel

```
{live-in: j, k}
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
b := *(f+0)
c := e + 8
d := c
k := m + 4
j := b
{live-out: d,j,k}
```

Interference graph

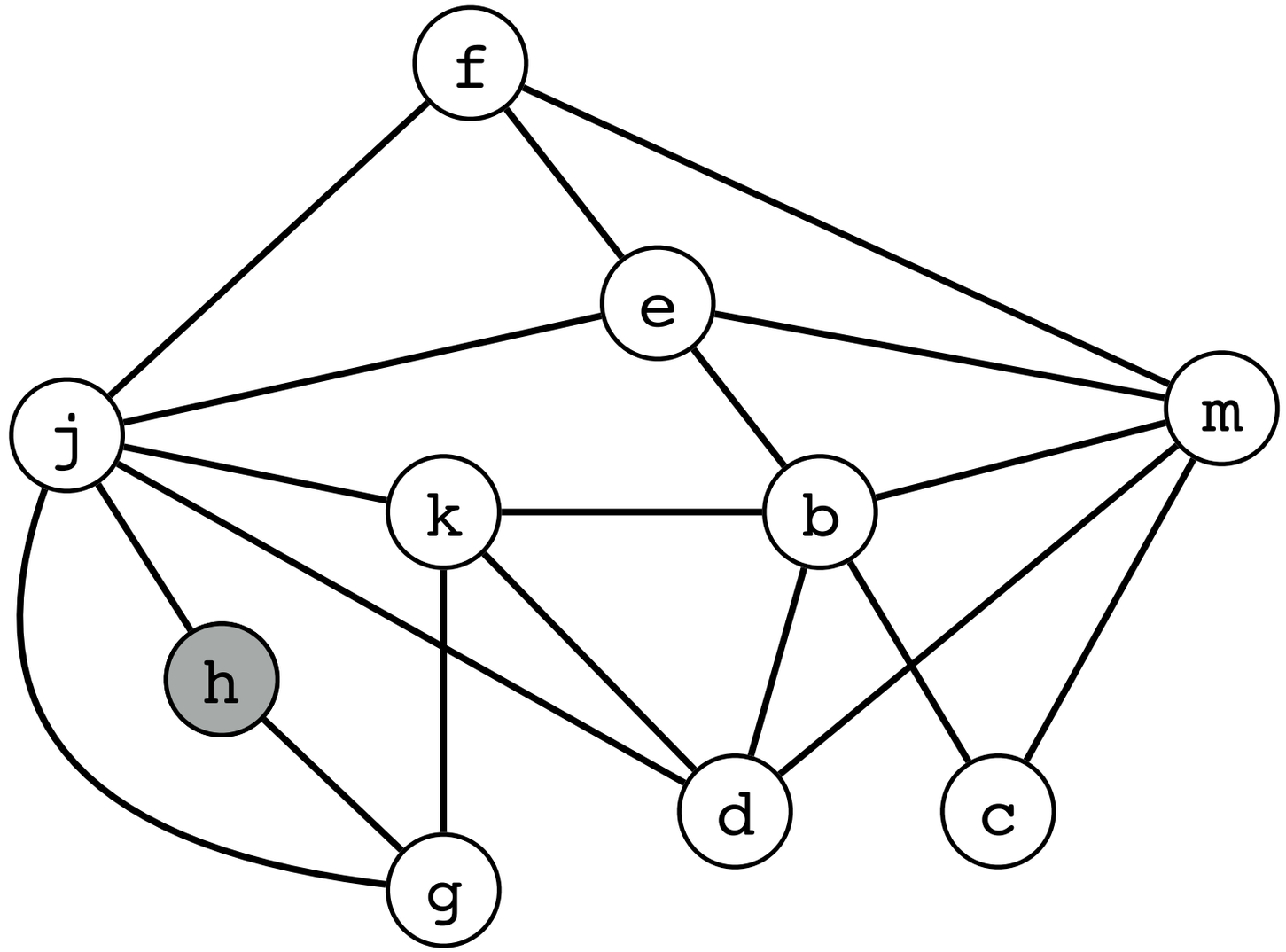


Simplification (3 registers)

Choose any node with degree <3

Stack:

h

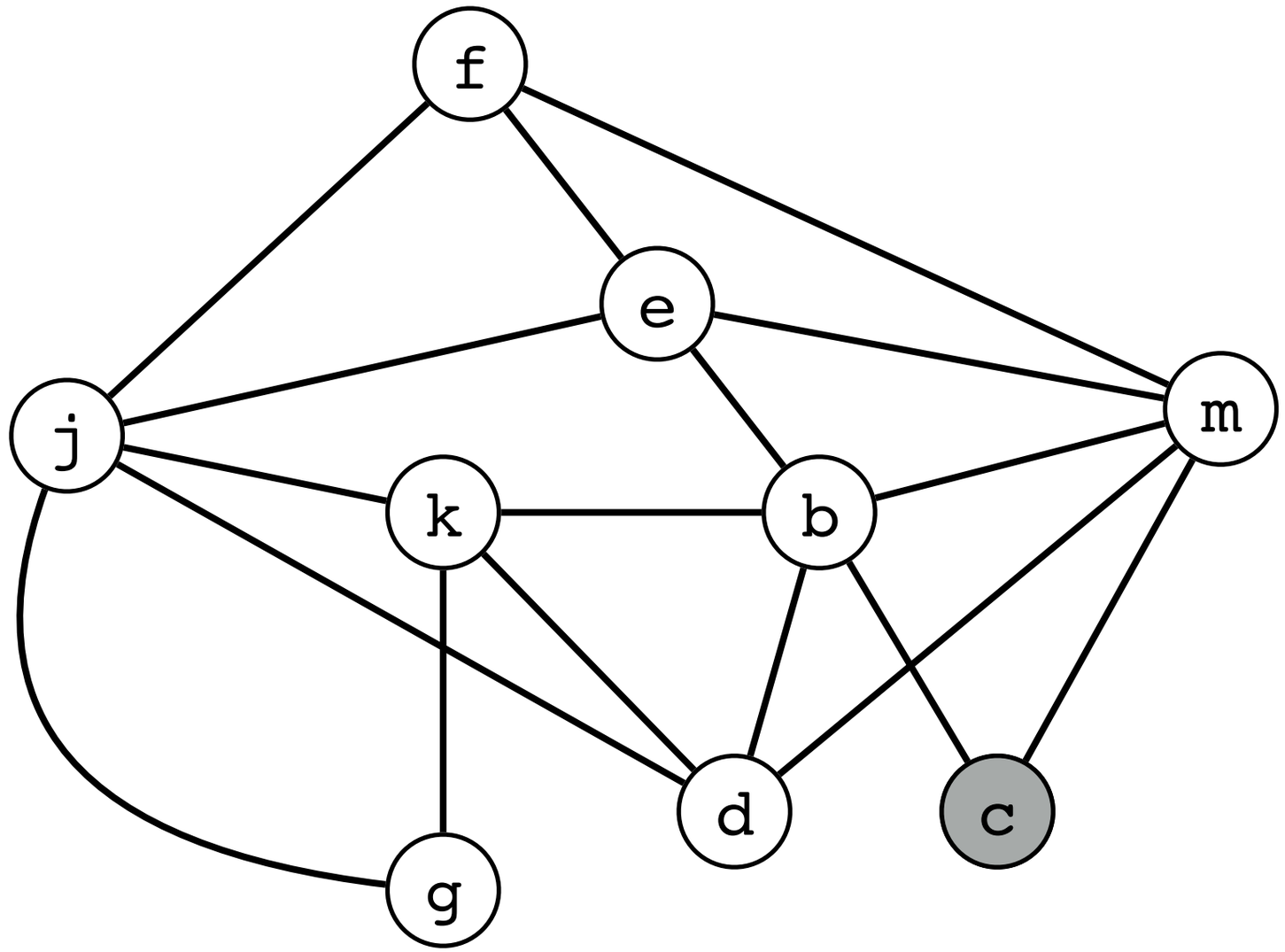


Simplification (3 registers)

Choose any node with degree <3

Stack:

h
c

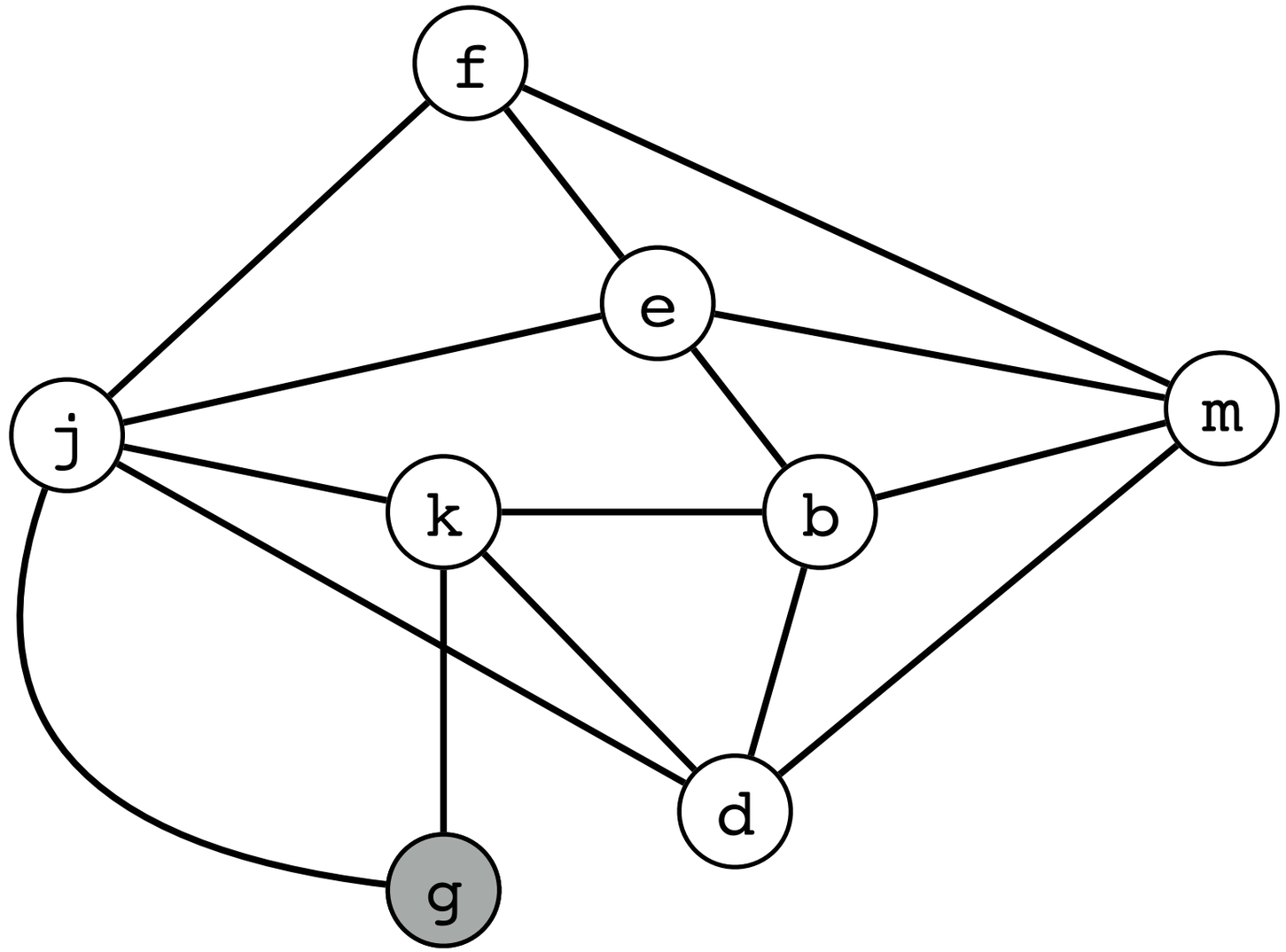


Simplification (3 registers)

Choose any node with degree <3

Stack:

h
c
g

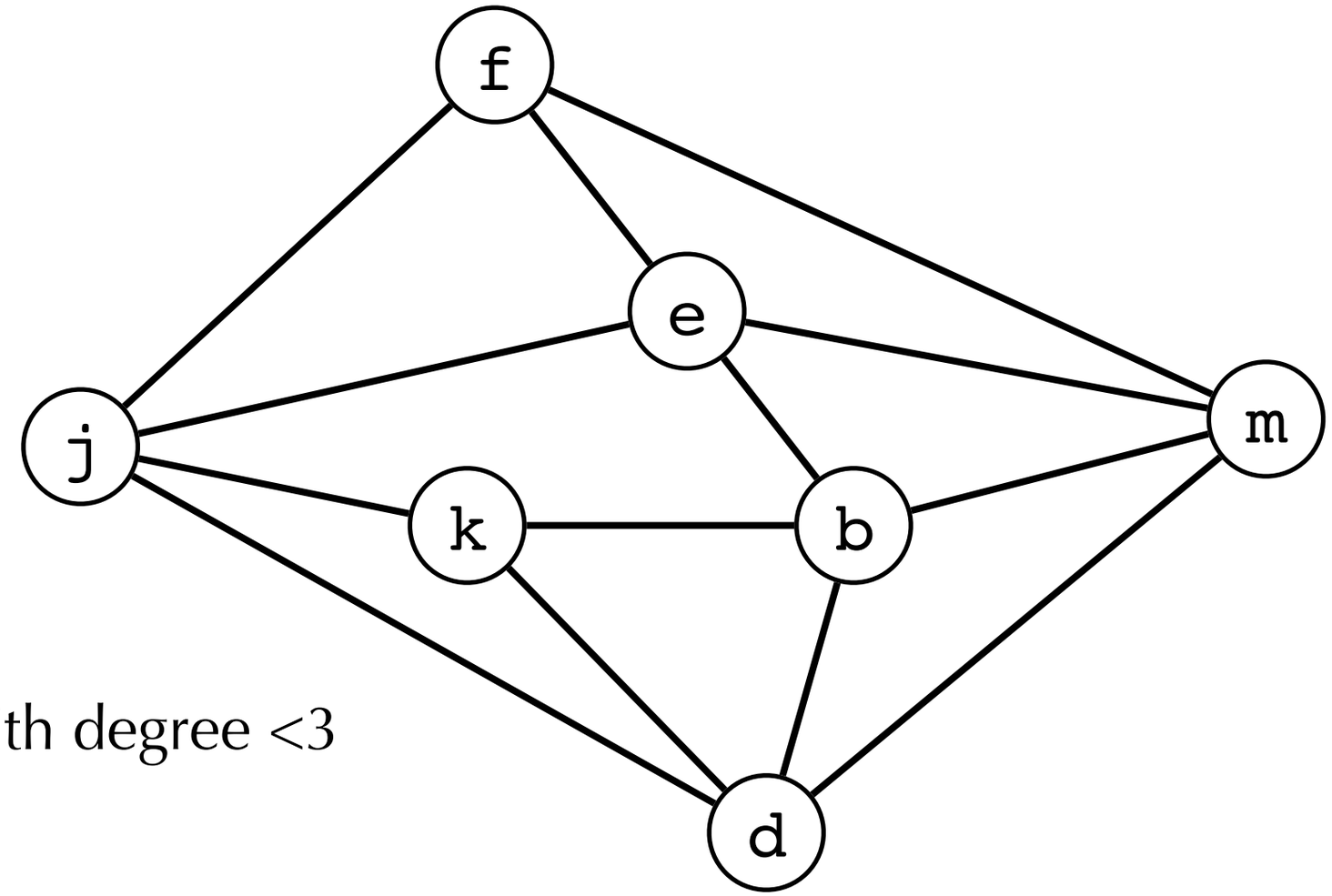


Simplification (3 registers)

Choose any node with degree <3

Stack:

h
c
g

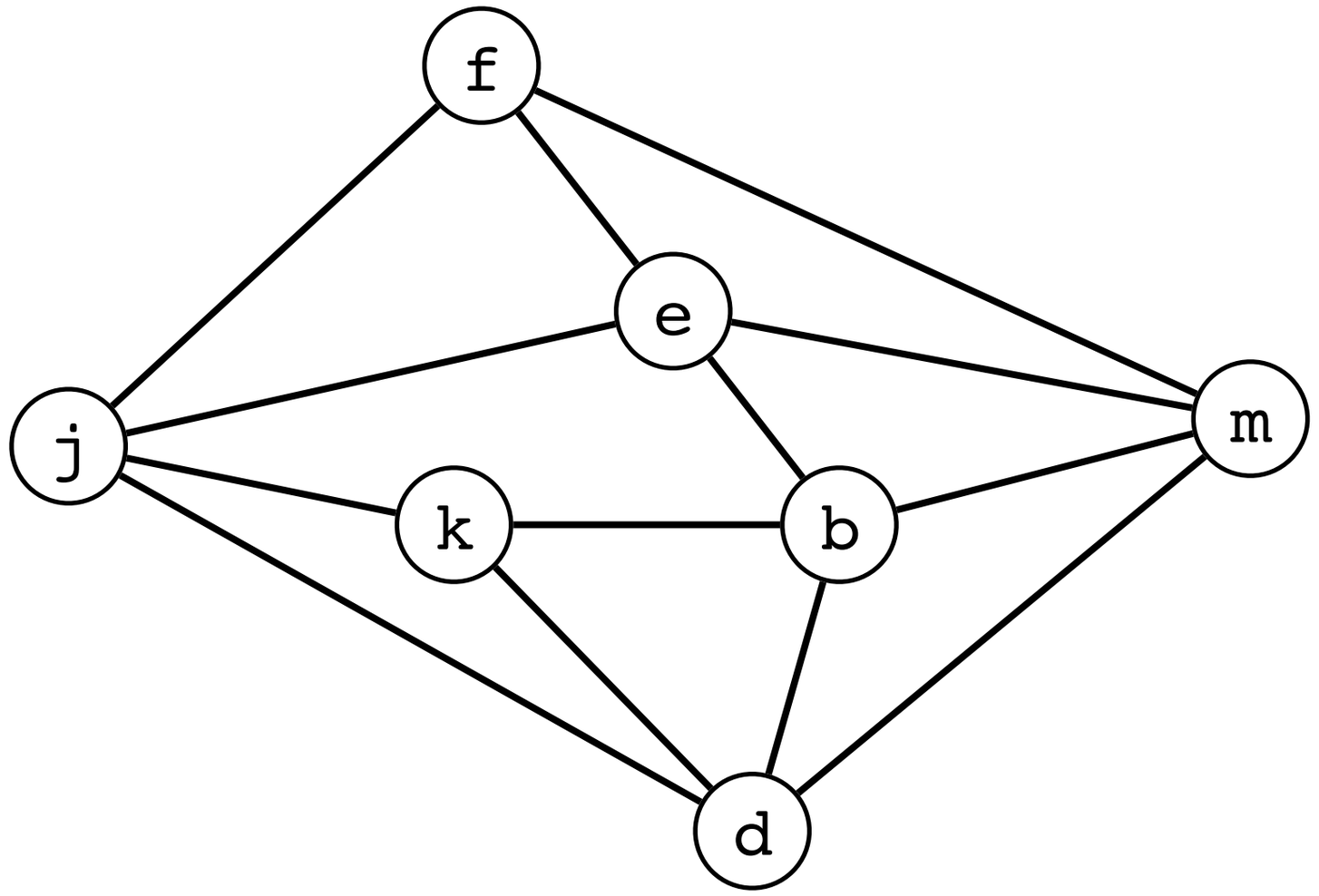


Now we are stuck! No nodes with degree <3

Pick a node to potentially spill

Which Node to Spill?

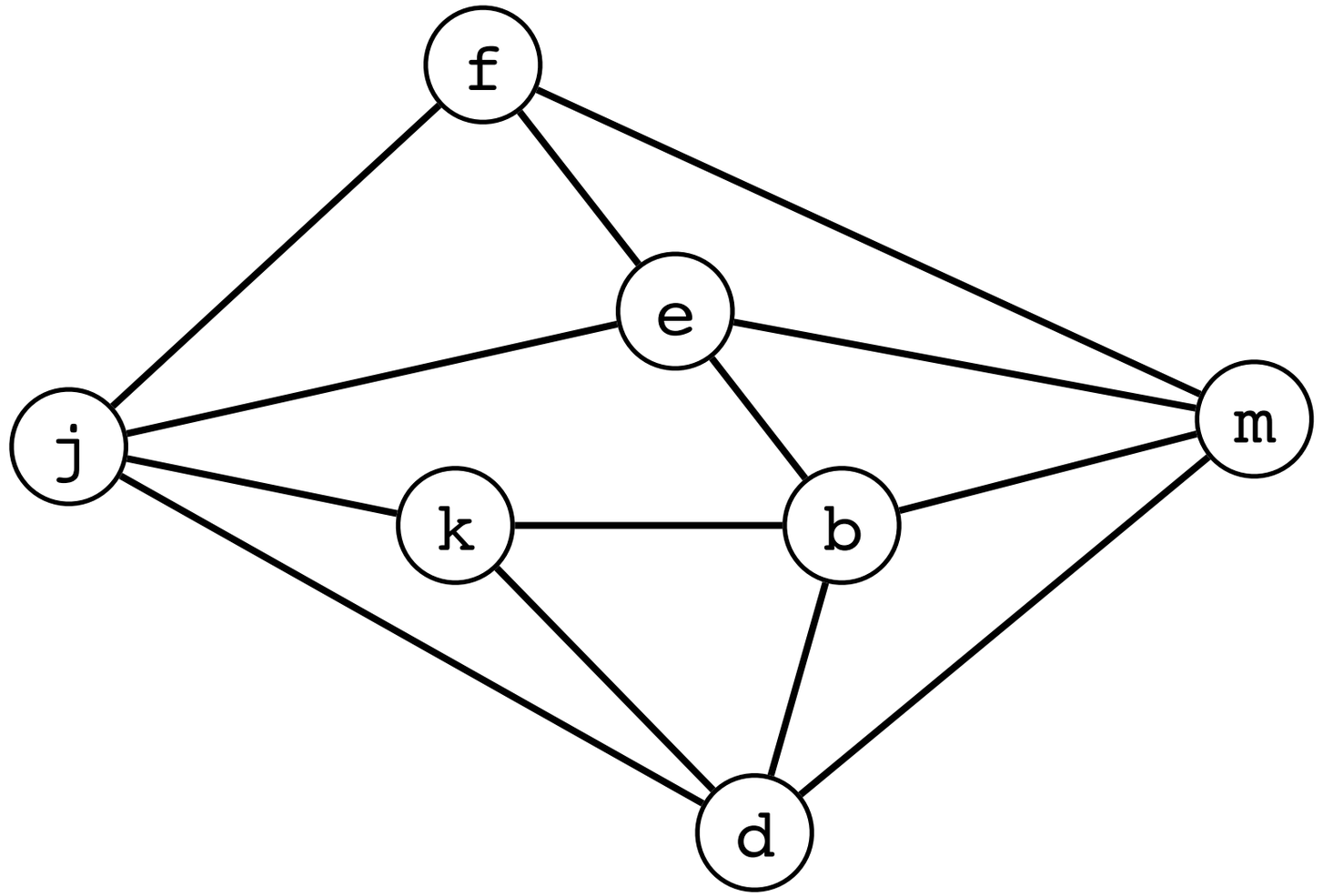
- Want to pick a node (i.e., temp variable) that will make it likely we'll be able to *k* color graph
  - High degree (≈ live at many program points)
  - Not used/defined very often (so we don't need to access stack very often)
- E.g., compute **spill priority** of node



$$\frac{\dfrac{\textit{Uses+defs}}{\textit{outside loop}} + \dfrac{\textit{Uses+defs}}{\textit{in loop}} \times 10}{\textit{degree of node}}$$

Which Node to Spill?

```
{live-in: j, k}
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
b := *(f+0)
c := e + 8
d := c
k := m + 4
j := b
{live-out: d,j,k}
```



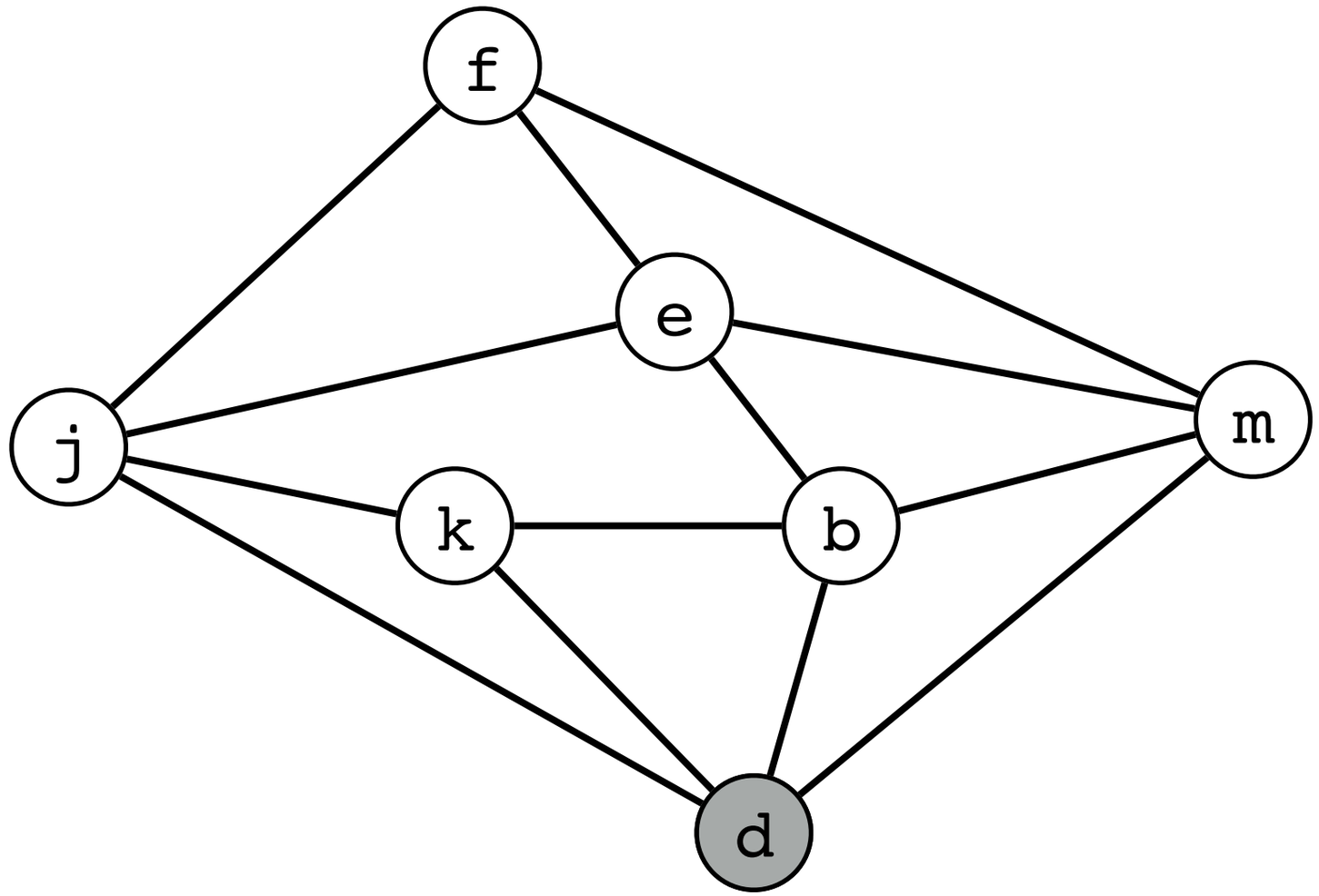
$$\textit{Spill priority} = \frac{\dfrac{\textit{Uses+defs}}{\textit{outside loop}} + \dfrac{\textit{Uses+defs}}{\textit{in loop}} \times 10}{\textit{degree of node}}$$

Simplification (3 registers)

Choose any node with degree <3

Stack:

h
c
g
d *spill?*



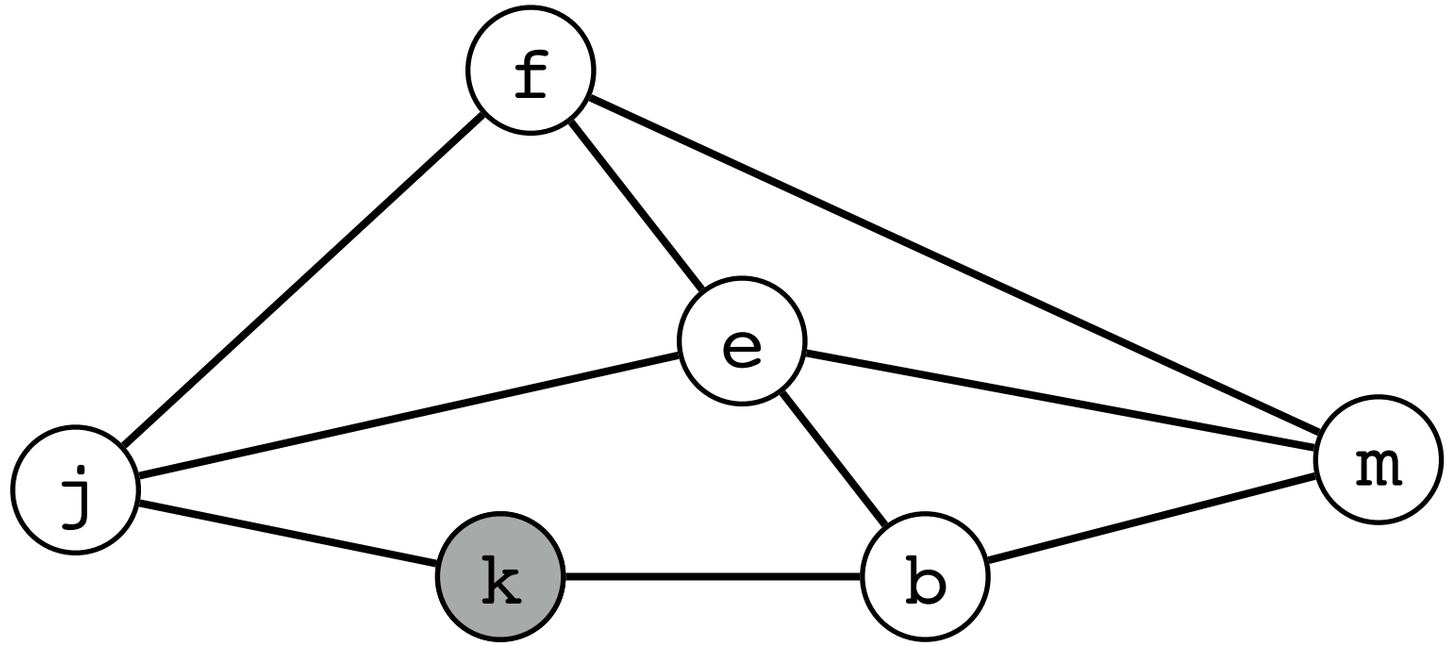
Pick a node with small spill priority degree to potentially spill

Simplification (3 registers)

Choose any node with degree <3

Stack:

h
c
g
d *spill?*
k

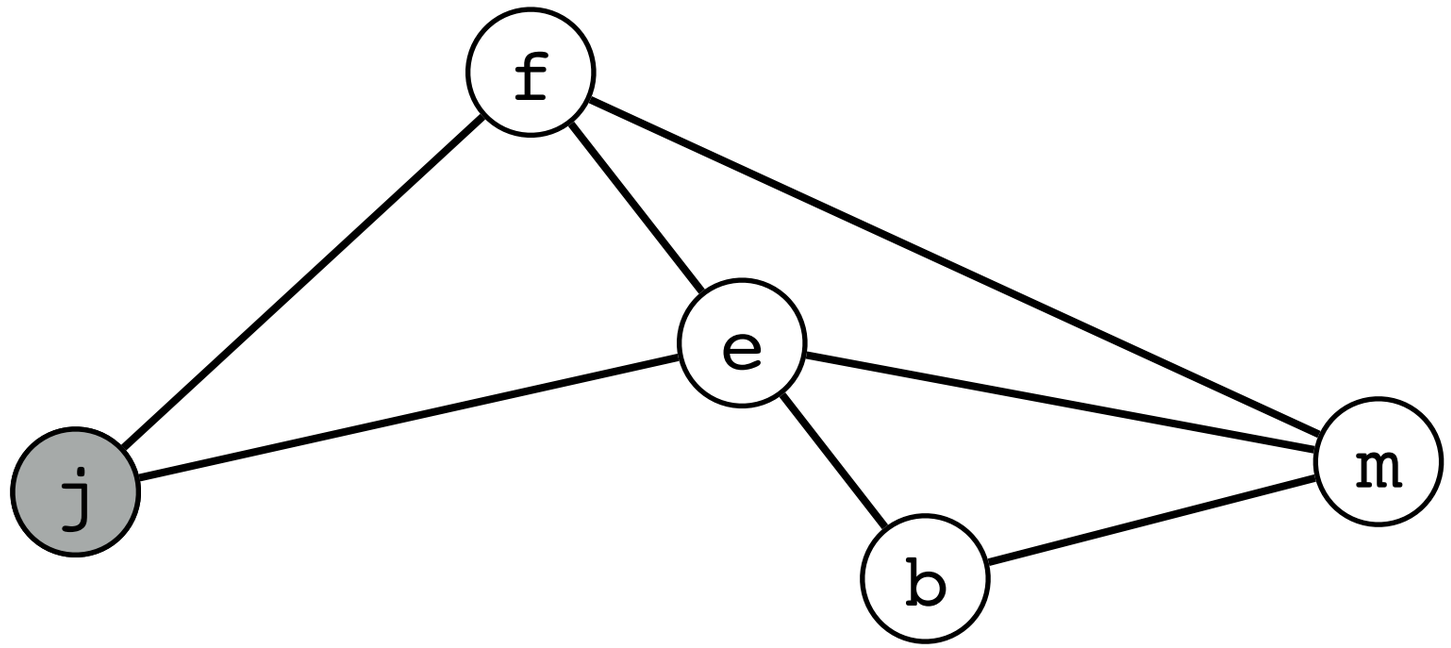


Simplification (3 registers)

Choose any node with degree <3

Stack:

h
c
g
d *spill?*
k
j

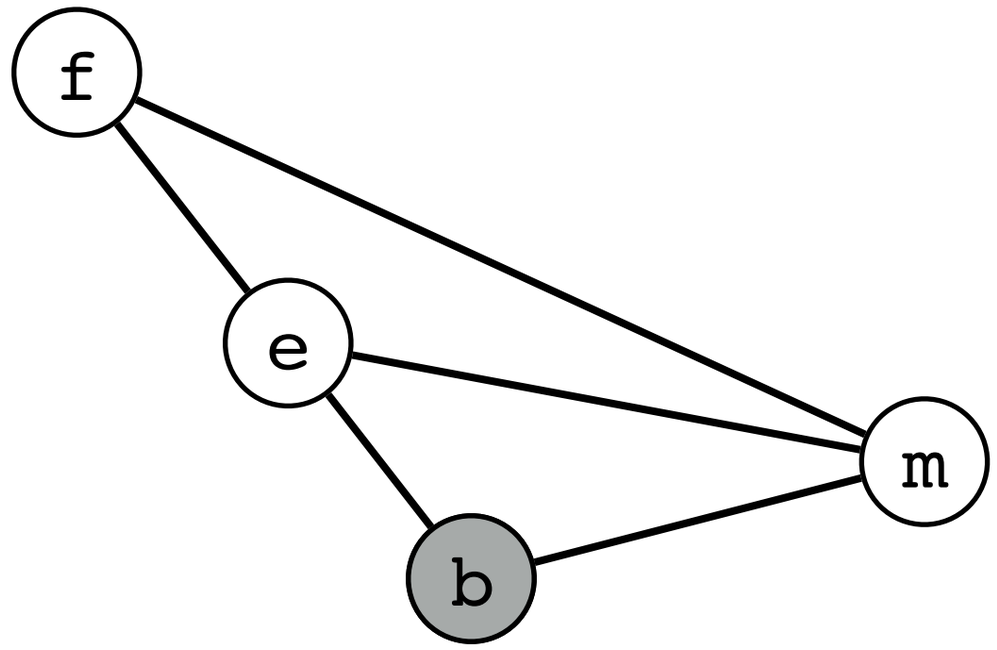


Simplification (3 registers)

Choose any node with degree <3

Stack:

h
c
g
d *spill?*
k
j
b

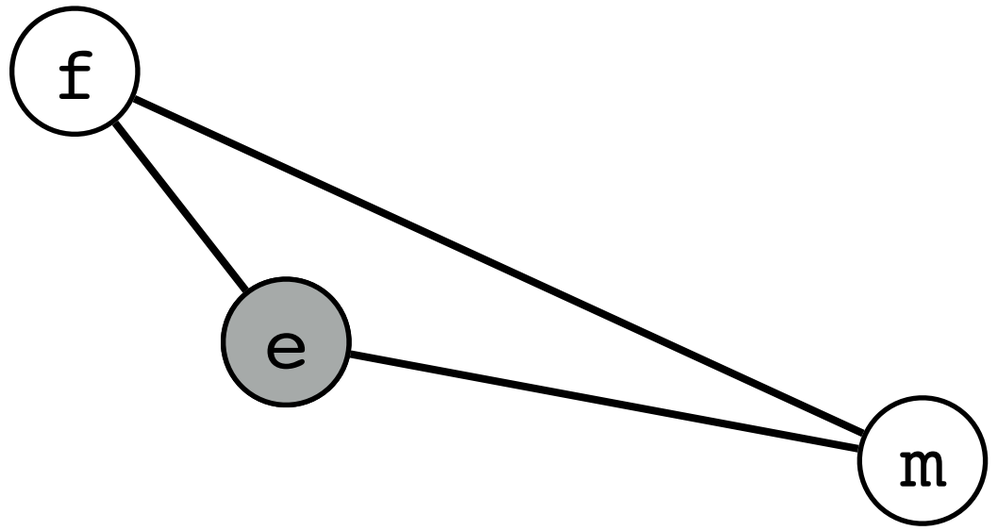


Simplification (3 registers)

Choose any node with degree <3

Stack:

h
c
g
d *spill?*
k
j
b
e

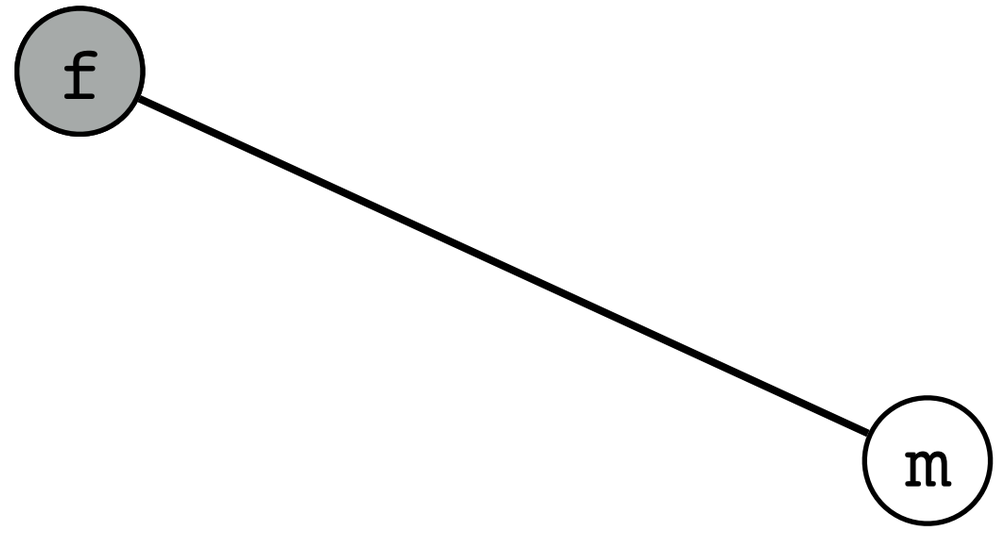


Simplification (3 registers)

Choose any node with degree <3

Stack:

h
c
g
d *spill?*
k
j
b
e
f



Simplification (3 registers)

Choose any node with degree <3

Stack:

`h`
`c`
`g`
`d` *spill?*
`k`
`j`
`b`
`e`
`f`
`m`

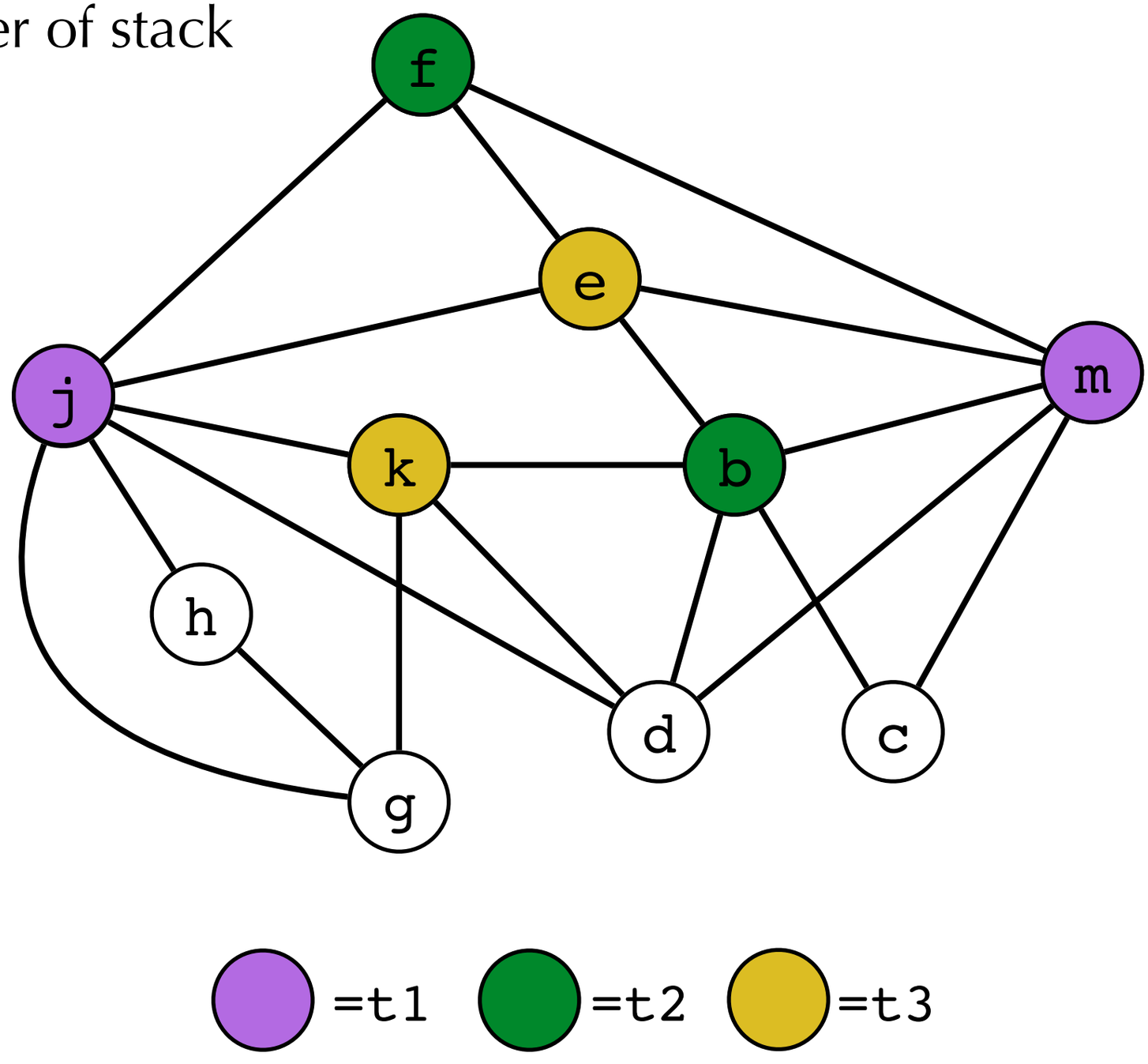
Select (3 registers)

Graph is now empty!

Stack:

Color nodes in order of stack

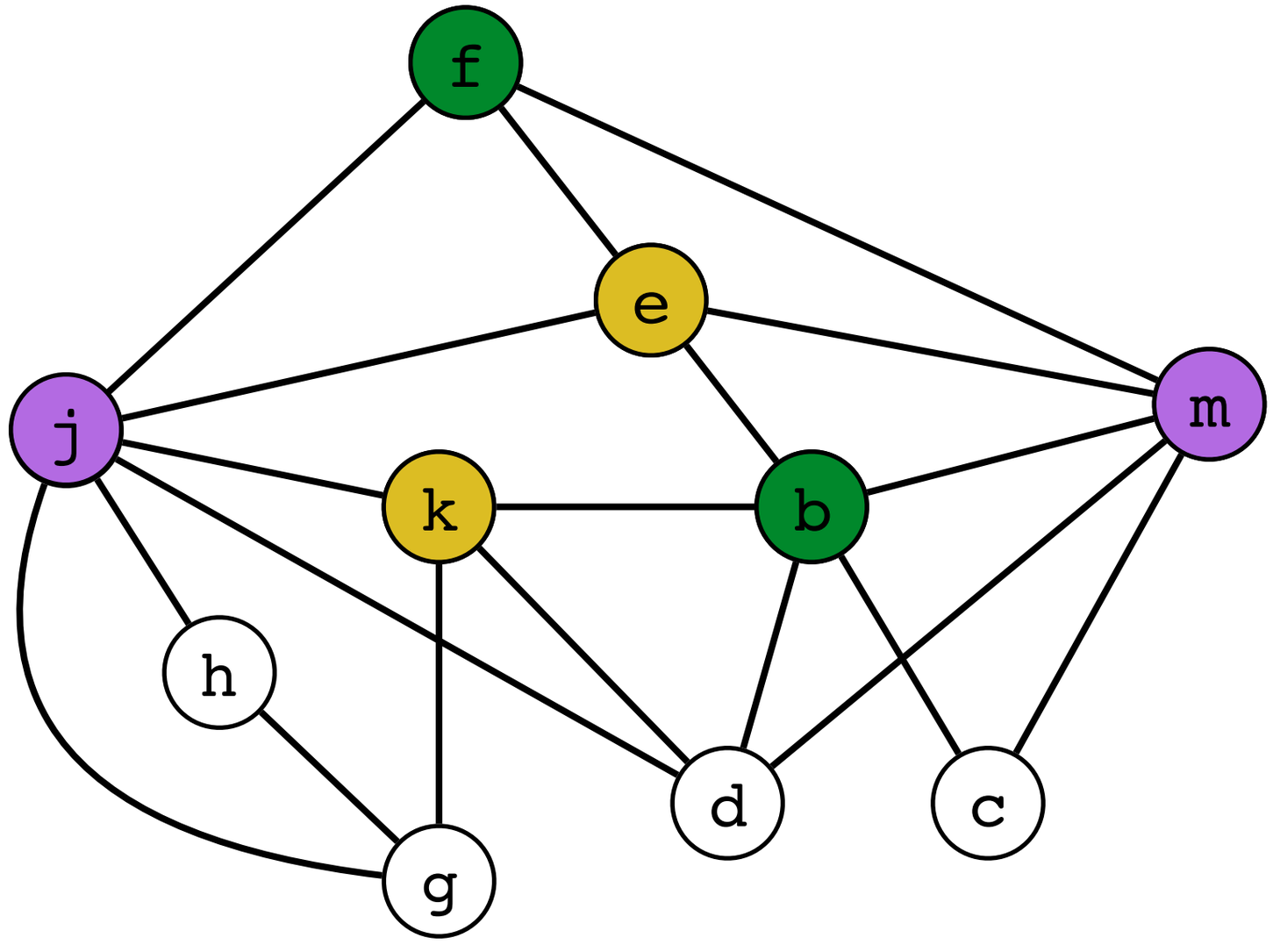
h
c
g
d *spill?*
k
j
b
e
f
m



Select (3 registers)

Stack:

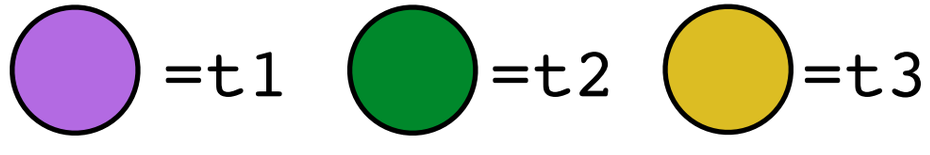
- h
- c
- g
- d *spill?*



We got unlucky!

In some cases a potential spill node is still colorable, and the Select phase can continue.

But in this case, we need to rewrite...



Select (3 registers)

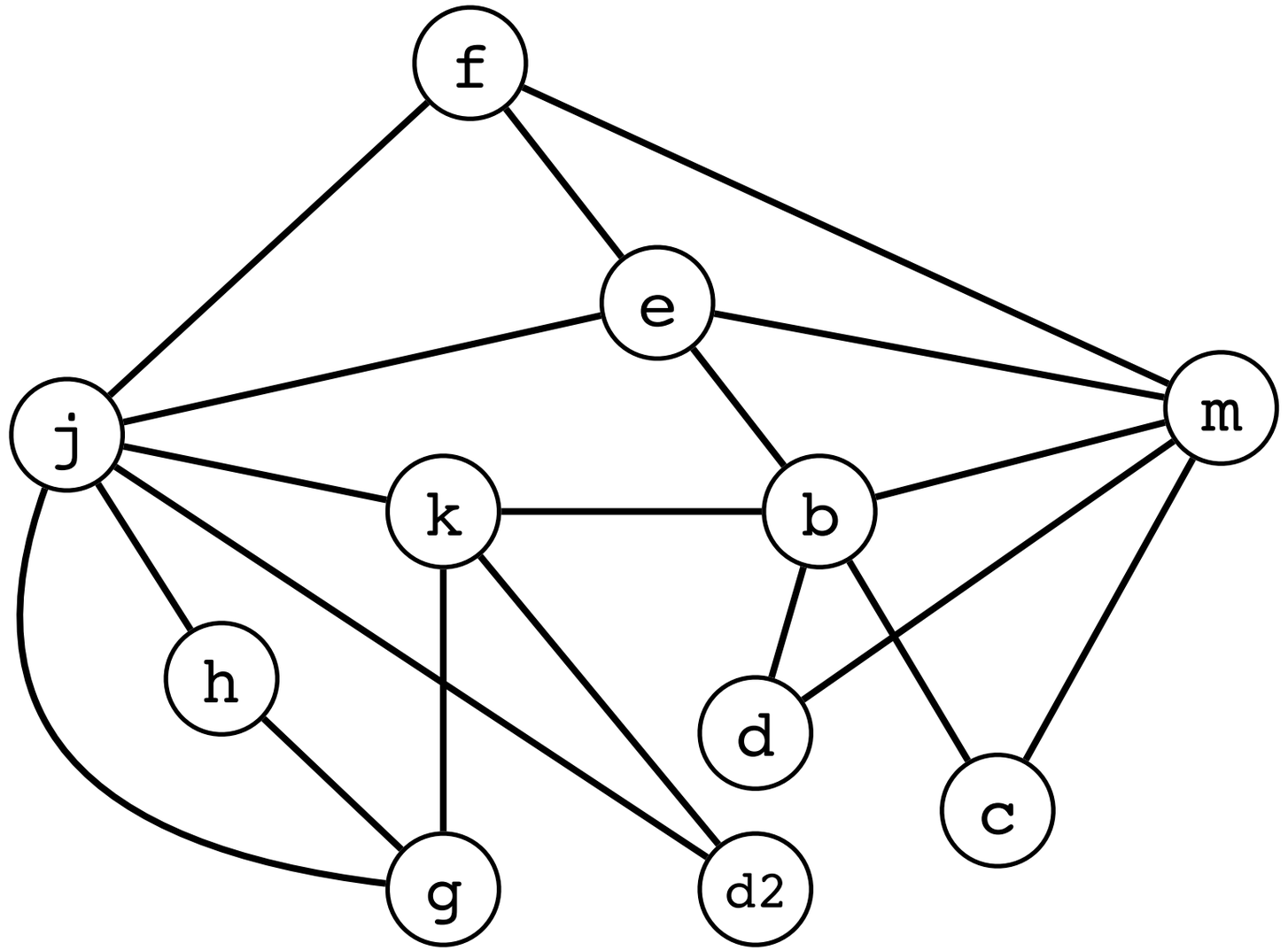
- Spill d

```
{live-in: j, k}
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
b := *(f+0)
c := e + 8
d := c
k := m + 4
j := b
{live-out: d,j,k}
```

```
{live-in: j, k}
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
b := *(f+0)
c := e + 8
d := c
*<fp+doff>:=d
k := m + 4
j := b
d2:=*<fp+doff>
{live-out: d2,j,k}
```

Build

```
{live-in: j, k}
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
b := *(f+0)
c := e + 8
d := c
*<fp+doff>:=d
k := m + 4
j := b
d2:=*<fp+doff>
{live-out: d2,j,k}
```

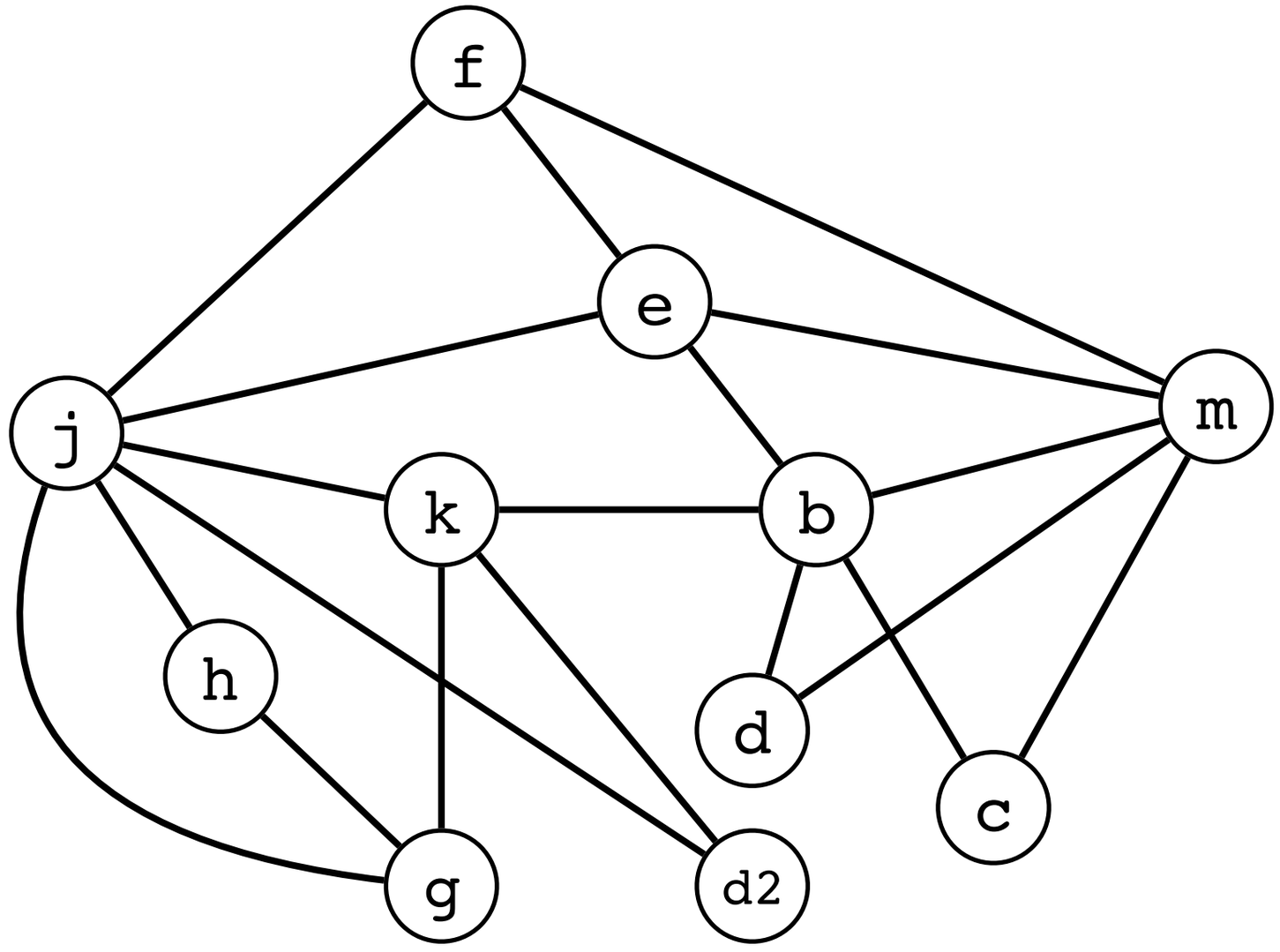


Simplification (3 registers)

Choose any node with degree <3

Stack:

h
c
g
d
d2
k
b
m
e
f
j



This time we succeed and
will be able to complete Select phase successfully!

Register Pressure

- Some optimizations increase live-ranges:
  - Copy propagation
  - Common sub-expression elimination
  - Loop invariant removal
- In turn, that can cause the allocator to spill
- Copy propagation isn't that useful anyway:
  - Let register allocator figure out if it can assign the same register to two temps!
  - Then the copy can go away.
  - And we don't have to worry about register pressure.

Coalescing Register Allocation

- If we have "`x := y`" and `x` and `y` have no edge in the interference graph, we might be able to assign them the same color.
  - This would translate to "`ri := ri`" which would then be removed
- One idea is to optimistically coalesce nodes in the interference graph
  - Just take the edges to be the union

Example

- E.g., the following nodes could be coalesced
  - `d` and `c`
  - `j` and `b`

```
{live-in: j, k}
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
b := *(f+0)
c := e + 8
d := c
k := m + 4
j := b
{live-out: d,j,k}
```