

Register Allocation

Compiler Design

CSE 504

- 1 Preliminaries
- 2 Graph Coloring
- 3 Spilling

Last modified: Wed Feb 04 2015 at 12:58:26 EST
Version: 1.4 18:44:24 2015/03/06
Compiled at 21:56 on 2016/04/23

Final Code Generation

- Intermediate Code assumed a machine with few physical limitations.
- For final code generation, intermediate instructions must be mapped to machine instructions
 - This is called **instruction selection**, and generally done using a dynamic programming algorithm (tiling).
- Variables may be allocated in registers instead of memory, mapping multiple variables without overlapping lifetimes to the same register/location.
 - This is the task of **register allocation**.

Lifetime of Variables

- A variable is *live*
 - from the program point where it is assigned a value
 - to the point where its value is no longer used.
- The extent of a variable's lifetime can be found by a *static program analysis*, called **liveness analysis**.

Liveness Analysis

- $use(s)$: the set of variables **used** in statement s (e.g. rhs of an assignment statement).
- $def(s)$: the set of variables **defined** in statement s (e.g. lhs of an assignment statement).
- $succ(s)$: the set of statements that **succeed** s .
- $out(s)$: the set of variables that are **live** immediately **after** s .
- $in(s)$: the set of variables that are **live** immediately **before** s .

$$\begin{aligned} in(s) &= use(s) \cup (out(s) - def(s)) \\ out(s) &= \bigcup_{t \in succ(s)} in(t) \end{aligned}$$

Liveness Analysis: an example

$$in(s) = use(s) \cup (out(s) - def(s))$$

$$out(s) = \bigcup_{t \in succ(s)} in(t)$$

Stmt	Def	Use	In	Out
$g = M[j+12]$	{g}	{j}		
$h = k - 1$	{h}	{k}		
$f = g * h$	{f}	{g,h}		
$e = M[j+8]$	{e}	{j}		
$m = M[j+16]$	{m}	{j}		
$b = M[f]$	{b}	{f}		
$c = e + 8$	{c}	{e}		
$d = c$	{d}	{c}		
$k = m + 4$	{k}	{m}		
$j = b$	{j}	{b}		{d, j, k}

Liveness Analysis: an example

$$in(s) = use(s) \cup (out(s) - def(s))$$

$$out(s) = \bigcup_{t \in succ(s)} in(t)$$

Stmt	Def	Use	In	Out
$g = M[j+12]$	$\{g\}$	$\{j\}$		
$h = k - 1$	$\{h\}$	$\{k\}$		
$f = g * h$	$\{f\}$	$\{g, h\}$		
$e = M[j+8]$	$\{e\}$	$\{j\}$		
$m = M[j+16]$	$\{m\}$	$\{j\}$		
$b = M[f]$	$\{b\}$	$\{f\}$		
$c = e + 8$	$\{c\}$	$\{e\}$		
$d = c$	$\{d\}$	$\{c\}$		
$k = m + 4$	$\{k\}$	$\{m\}$		
$j = b$	$\{j\}$	$\{b\}$	$\{b, d, k\}$	$\{d, j, k\}$

Liveness Analysis: an example

$$in(s) = use(s) \cup (out(s) - def(s))$$

$$out(s) = \bigcup_{t \in succ(s)} in(t)$$

Stmt	Def	Use	In	Out
$g = M[j+12]$	{g}	{j}		
$h = k - 1$	{h}	{k}		
$f = g * h$	{f}	{g,h}		
$e = M[j+8]$	{e}	{j}		
$m = M[j+16]$	{m}	{j}		
$b = M[f]$	{b}	{f}		
$c = e + 8$	{c}	{e}		
$d = c$	{d}	{c}		
$k = m + 4$	{k}	{m}		{b, d, k}
$j = b$	{j}	{b}	{b, d, k}	{d, j, k}

Liveness Analysis: an example

$$in(s) = use(s) \cup (out(s) - def(s))$$

$$out(s) = \bigcup_{t \in succ(s)} in(t)$$

Stmt	Def	Use	In	Out
$g = M[j+12]$	{g}	{j}		
$h = k - 1$	{h}	{k}		
$f = g * h$	{f}	{g, h}		
$e = M[j+8]$	{e}	{j}		
$m = M[j+16]$	{m}	{j}		
$b = M[f]$	{b}	{f}		
$c = e + 8$	{c}	{e}		
$d = c$	{d}	{c}		{b, d, m}
$k = m + 4$	{k}	{m}	{b, d, m}	{b, d, k}
$j = b$	{j}	{b}	{b, d, k}	{d, j, k}

Liveness Analysis: an example

$$in(s) = use(s) \cup (out(s) - def(s))$$

$$out(s) = \bigcup_{t \in succ(s)} in(t)$$

Stmt	Def	Use	In	Out
$g = M[j+12]$	$\{g\}$	$\{j\}$		
$h = k - 1$	$\{h\}$	$\{k\}$		
$f = g * h$	$\{f\}$	$\{g, h\}$		
$e = M[j+8]$	$\{e\}$	$\{j\}$		
$m = M[j+16]$	$\{m\}$	$\{j\}$		
$b = M[f]$	$\{b\}$	$\{f\}$		
$c = e + 8$	$\{c\}$	$\{e\}$		$\{b, c, m\}$
$d = c$	$\{d\}$	$\{c\}$	$\{b, c, m\}$	$\{b, d, m\}$
$k = m + 4$	$\{k\}$	$\{m\}$	$\{b, d, m\}$	$\{b, d, k\}$
$j = b$	$\{j\}$	$\{b\}$	$\{b, d, k\}$	$\{d, j, k\}$

Liveness Analysis: an example

$$in(s) = use(s) \cup (out(s) - def(s))$$

$$out(s) = \bigcup_{t \in succ(s)} in(t)$$

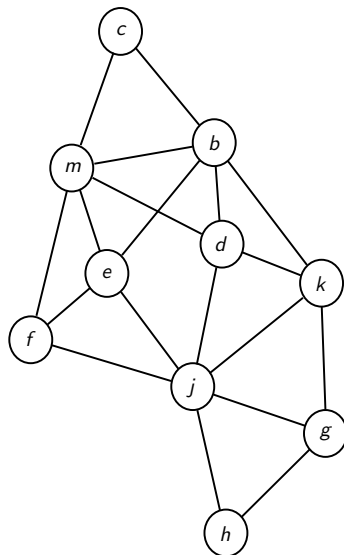
Stmt	Def	Use	In	Out
$g = M[j+12]$	{g}	{j}	{j, k}	{g, j, k}
$h = k - 1$	{h}	{k}	{g, j, k}	{g, h, j}
$f = g * h$	{f}	{g, h}	{g, h, j}	{f, j}
$e = M[j+8]$	{e}	{j}	{f, j}	{e, f, j}
$m = M[j+16]$	{m}	{j}	{e, f, j}	{e, f, m}
$b = M[f]$	{b}	{f}	{e, f, m}	{b, e, m}
$c = e + 8$	{c}	{e}	{b, e, m}	{b, c, m}
$d = c$	{d}	{c}	{b, c, m}	{b, d, m}
$k = m + 4$	{k}	{m}	{b, d, m}	{b, d, k}
$j = b$	{j}	{b}	{b, d, k}	{d, j, k}

Liveness Information

- If two variables are **live** at the same program point, then they cannot be both allocated to the same register/location.
- v_1 and v_2 **interfere** if $\{v_1, v_2\} \subseteq \text{in}(s)$ or $\text{out}(s)$ for some statement s .
- We build an **interference graph** with
 - variables as vertices, and
 - edge (v_1, v_2) if v_1 and v_2 interfere.

Interference Graph Example

Stmt	In
<code>g = M[j+12]</code>	$\{j, k\}$
<code>h = k - 1</code>	$\{g, j, k\}$
<code>f = g * h</code>	$\{g, h, j\}$
<code>e = M[j+8]</code>	$\{f, j\}$
<code>m = M[j+16]</code>	$\{e, f, j\}$
<code>b = M[f]</code>	$\{e, f, m\}$
<code>c = e + 8</code>	$\{b, e, m\}$
<code>d = c</code>	$\{b, c, m\}$
<code>k = m + 4</code>	$\{b, d, m\}$
<code>j = b</code>	$\{b, d, k\}$
<code>end</code>	$\{d, j, k\}$



From Appel's "Modern Compiler Implementation in Java" book.

Register Allocation as Graph Coloring

- **Graph Coloring:** an assignment of colors (from a finite set) to vertices such that

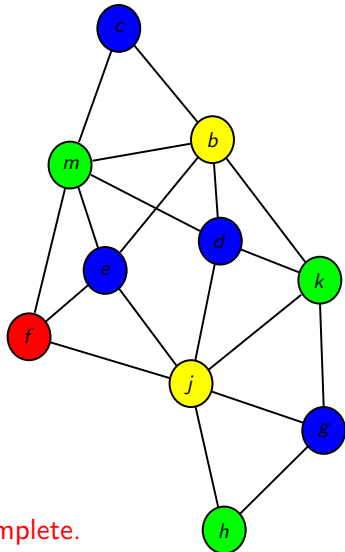
no two adjacent vertices have the same color.

- **Register Allocation:** an assignment of registers (colors) to variables (vertices)

no two adjacent vertices (i.e. variables live at same time)

have the same register (color).

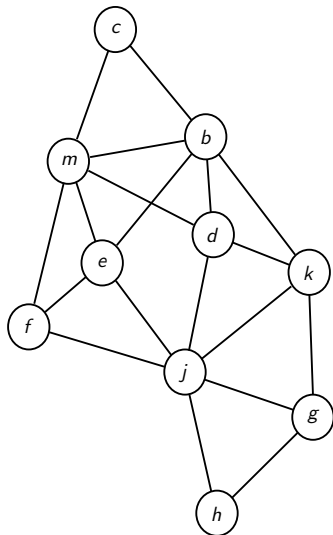
Unfortunately, graph coloring is NP-complete.



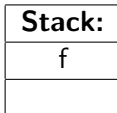
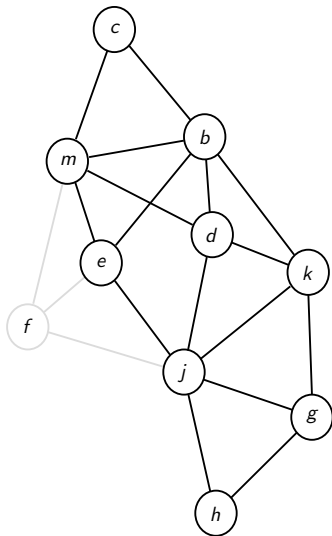
Graph Coloring Heuristic 1: Simplify & Select

```
1: procedure K-COLOR( $G, K$ )
2:   stack = empty
3:   while  $\exists v \in G$  with degree  $< K$  do                                ▷ Simplify
4:      $E$  = set of edges incident on  $v$ ;   Remove  $v$  and  $E$  from  $G$ 
5:     push( $v, E$ )
6:   end while
7:   if  $G$  is empty then                                                ▷ Select
8:     while stack is not empty do
9:        $(v, E) = \text{pop}()$ ;   Add  $v$  and  $E$  to  $G$ 
10:      color( $v$ ) =  $c$ , such that  $c \notin \{\text{color}(u) \mid (u, v) \in G\}$ 
11:                                     ▷ Note  $|E| < K$ , so there is such a  $c$ 
12:    end while
13:  else
14:    Exit with failure
15:  end if
16: end procedure
```

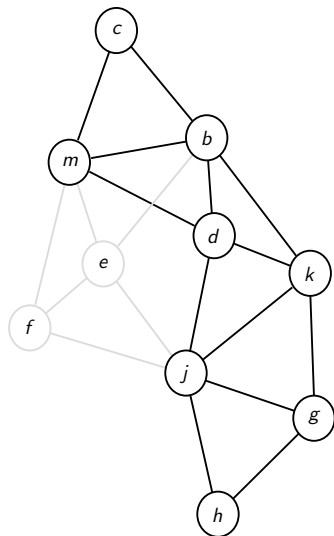
Graph Coloring Heuristic 1: Example

 $K = 4$ **Stack:**

Graph Coloring Heuristic 1: Example

 $K = 4$ 

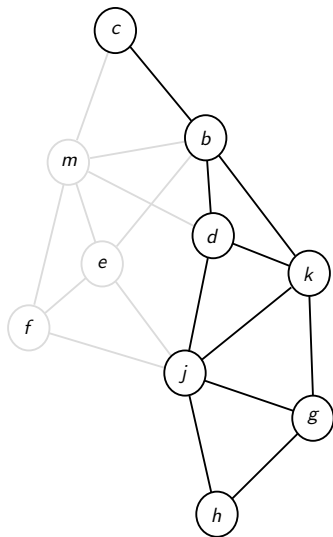
Graph Coloring Heuristic 1: Example

 $K = 4$ **Stack:**

f

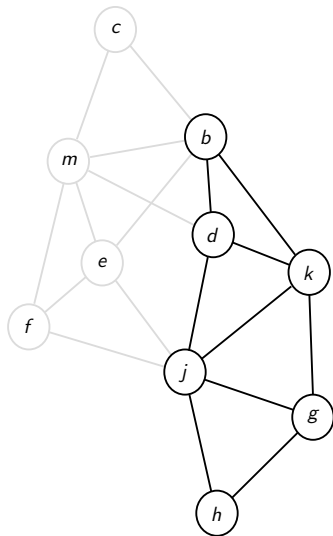
e

Graph Coloring Heuristic 1: Example

 $K = 4$ 

Stack:
f
e
m

Graph Coloring Heuristic 1: Example

 $K = 4$ **Stack:**

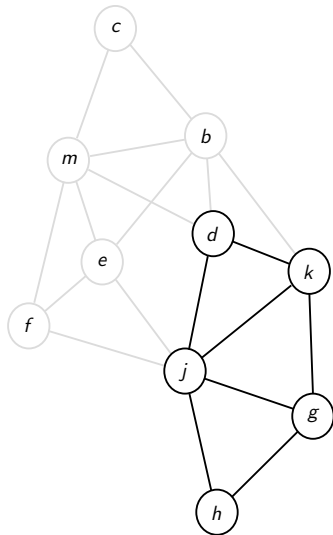
f

e

m

c

Graph Coloring Heuristic 1: Example

 $K = 4$ **Stack:**

f

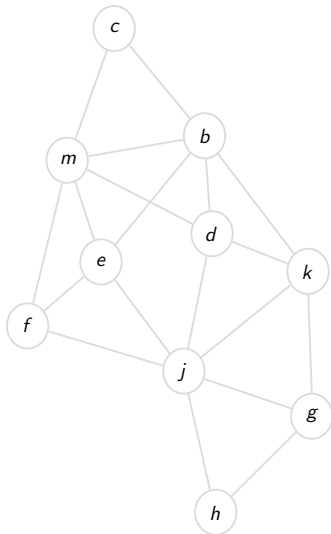
e

m

c

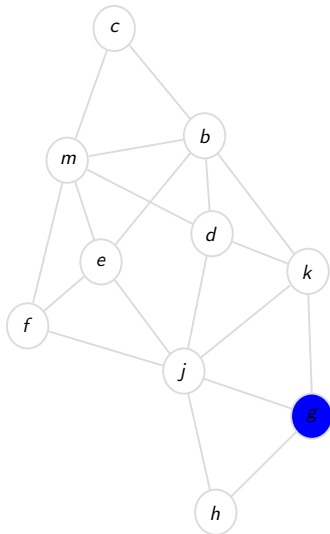
b

Graph Coloring Heuristic 1: Example

 $K = 4$ 

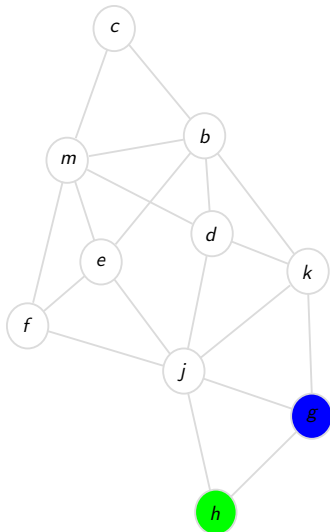
Stack:
f
e
m
c
b
d
j
k
h
g

Graph Coloring Heuristic 1: Example

 $K = 4$ 

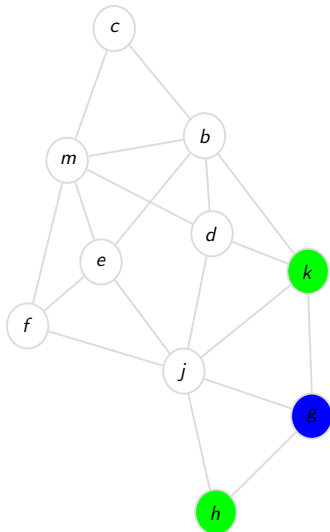
Stack:
f
e
m
c
b
d
j
k
h
g

Graph Coloring Heuristic 1: Example

 $K = 4$ 

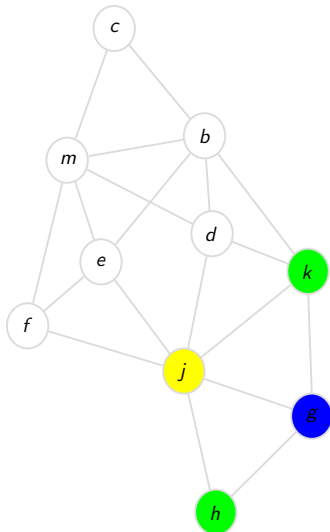
Stack:
f
e
m
c
b
d
j
k
h

Graph Coloring Heuristic 1: Example

 $K = 4$ 

Stack:
f
e
m
c
b
d
j
k

Graph Coloring Heuristic 1: Example

 $K = 4$ **Stack:**

f

e

m

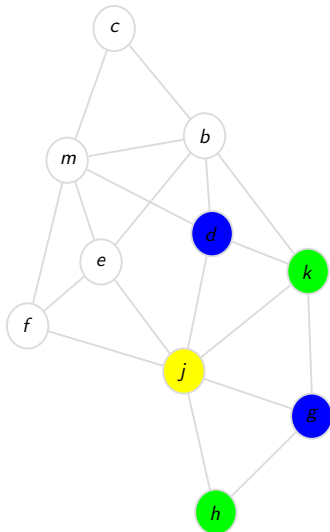
c

b

d

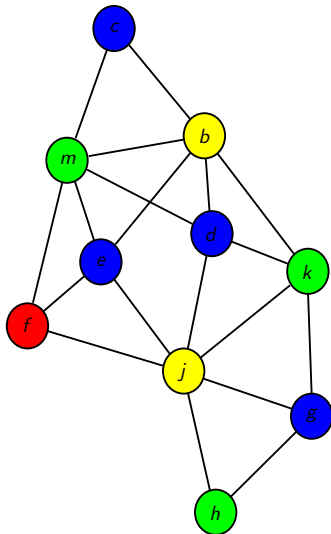
j

Graph Coloring Heuristic 1: Example

 $K = 4$ 

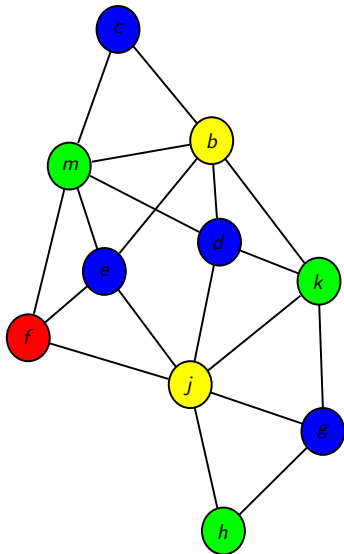
Stack:
f
e
m
c
b
d

Graph Coloring Heuristic 1: Example

 $K = 4$ 

Stack:

Code With Registers

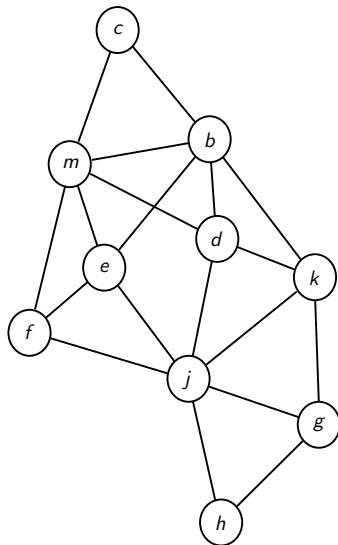


Stmt	With Regs
$g = M[j+12]$	$r0 = M[r2+12]$
$h = k - 1$	$r1 = r1 - 1$
$f = g * h$	$r3 = r0 * r1$
$e = M[j+8]$	$r0 = M[r2+8]$
$m = M[j+16]$	$r1 = M[r2+16]$
$b = M[f]$	$r2 = M[r3]$
$c = e + 8$	$r0 = r0 + 8$
$d = c$	$r0 = r0$
$k = m + 4$	$r1 = r1 + 4$
$j = b$	$r2 = r2$

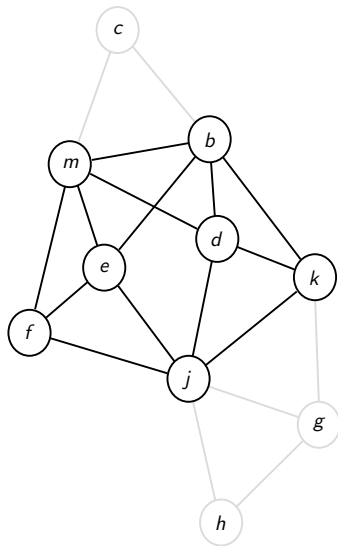
Graph Coloring Heuristic 2: Simplify, Spill & Select

```
1: procedure K-COLOR( $G, K$ )
2:    $stack = empty$ 
3:   while  $G$  is not empty do
4:     if  $\exists v \in G$  with degree  $< K$  then                                ▷ Simplify
5:       Select such a  $v$ 
6:     else                                                                ▷ Spill
7:       Select a  $v$  based on spill priority
8:     end if
9:      $E =$  set of edges incident on  $v$ ;   Remove  $v$  and  $E$  from  $G$ 
10:    push( $v, E$ )
11:  end while
12:  while  $stack$  is not empty do                                         ▷ Select
13:    ( $v, E$ ) = pop();   Add  $v$  and  $E$  to  $G$ 
14:     $C = \{color(u) | (u, v) \in G\}$ 
15:    if  $|C| < K$  then
16:       $color(v) = c$ , such that  $c \notin C$ 
17:    else
18:      Exit with failure
19:    end if
20:  end while
21: end procedure
```

Graph Coloring Heuristic 2: Example

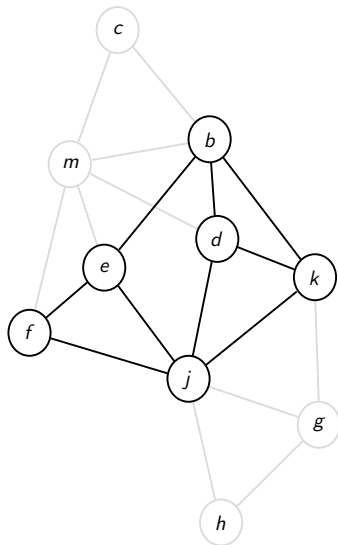
 $K = 3$ **Stack:**

Graph Coloring Heuristic 2: Example

 $K = 3$ 

Stack:
c
h
g

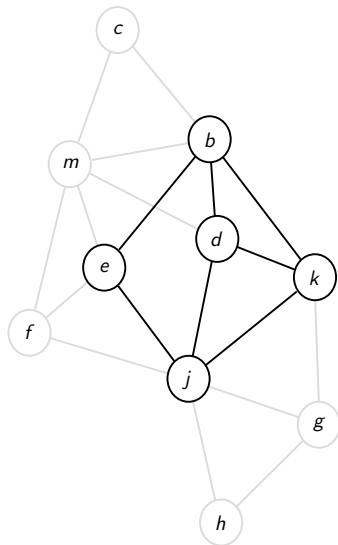
Graph Coloring Heuristic 2: Example

 $K = 3$ 

Stack:
c
h
g
m

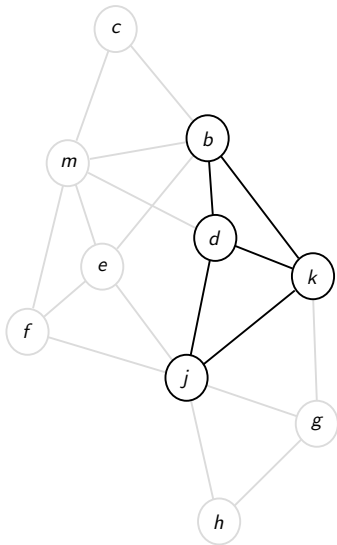
- **m** is selected for potentially spilling: it is rarely used, and has high degree.

Graph Coloring Heuristic 2: Example

 $K = 3$ 

Stack:
c
h
g
m
f

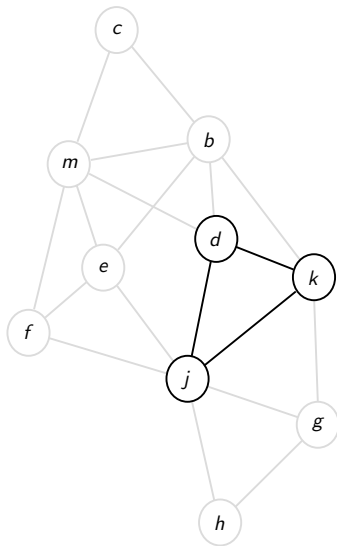
- **m** is selected for potentially spilling: it is rarely used, and has high degree.

$K = 3$ 

Stack:
c
h
g
m
f
e

- **m** is selected for potentially spilling: it is rarely used, and has high degree.

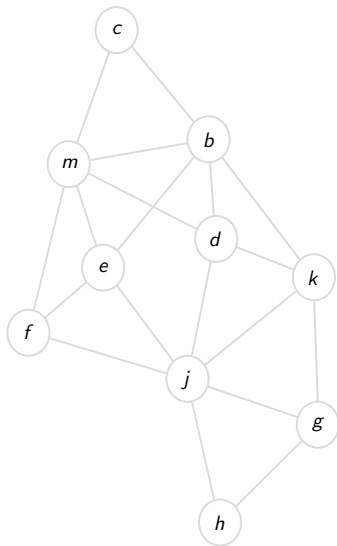
Graph Coloring Heuristic 2: Example

 $K = 3$ 

Stack:
c
h
g
m
f
e
b

- **m** is selected for potentially spilling: it is rarely used, and has high degree.

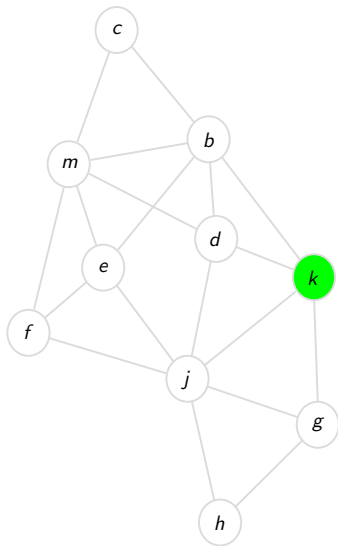
Graph Coloring Heuristic 2: Example

 $K = 3$ 

Stack:
c
h
g
m
f
e
b
d
j
k

- **m** is selected for potentially spilling: it is rarely used, and has high degree.

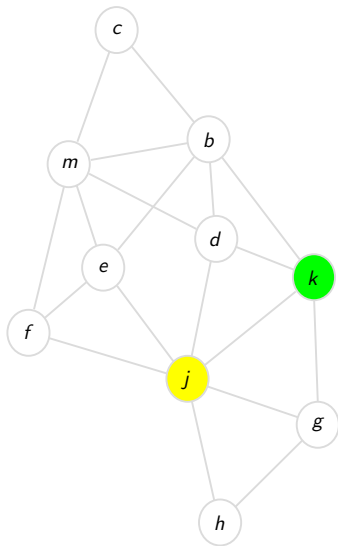
Graph Coloring Heuristic 2: Example

 $K = 3$ 

Stack:
c
h
g
m
f
e
b
d
j
k

- **m** is selected for potentially spilling: it is rarely used, and has high degree.

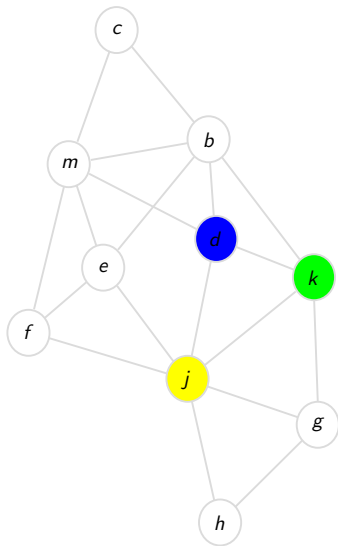
Graph Coloring Heuristic 2: Example

 $K = 3$ 

Stack:
c
h
g
m
f
e
b
d
j

- **m** is selected for potentially spilling: it is rarely used, and has high degree.

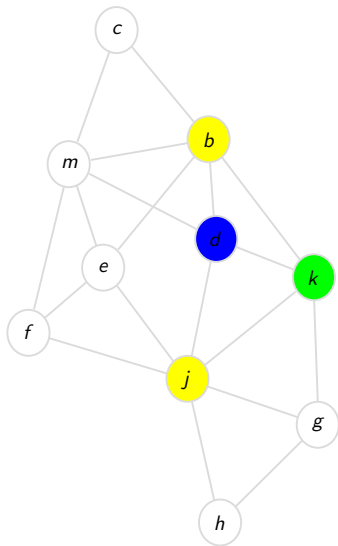
Graph Coloring Heuristic 2: Example

 $K = 3$ 

Stack:
c
h
g
m
f
e
b
d

- **m** is selected for potentially spilling: it is rarely used, and has high degree.

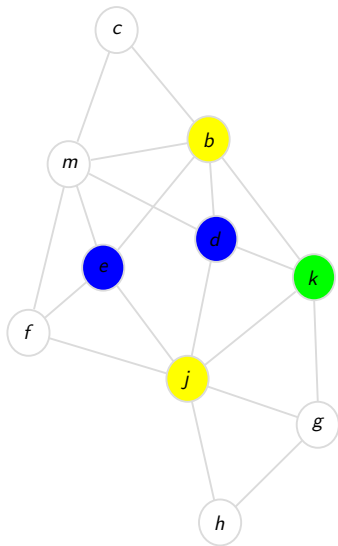
Graph Coloring Heuristic 2: Example

 $K = 3$ 

Stack:
c
h
g
m
f
e
b

- **m** is selected for potentially spilling: it is rarely used, and has high degree.

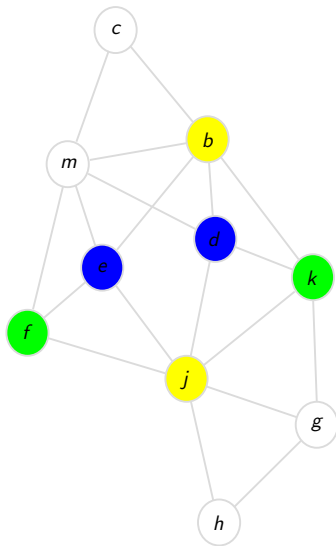
Graph Coloring Heuristic 2: Example

 $K = 3$ 

Stack:
c
h
g
m
f
e

- **m** is selected for potentially spilling: it is rarely used, and has high degree.

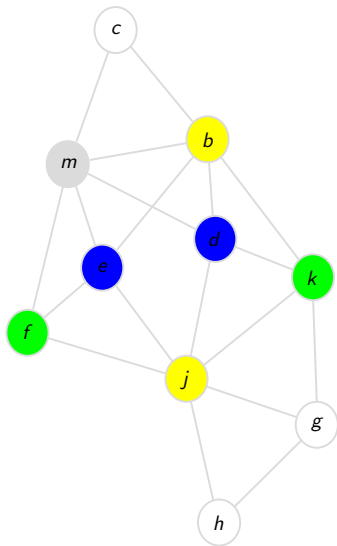
Graph Coloring Heuristic 2: Example

 $K = 3$ 

Stack:
c
h
g
m
f

- **m** is selected for potentially spilling: it is rarely used, and has high degree.

Graph Coloring Heuristic 2: Example

 $K = 3$ 

Stack:
c
h
g
m

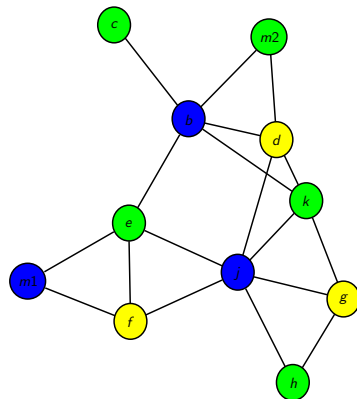
- **m** is selected for potentially spilling: it is rarely used, and has high degree.
- **m** needs to be spilled: stored and restored from memory.

Simplify, Spill, Select & Start Over

```
g = M[j+12]
h = k - 1
f = g * h
e = M[j+8]
m = M[j+16]
b = M[f]
c = e + 8
d = c
k = m + 4
j = b
```

 $\Rightarrow$

```
g = M[j+12]
h = k - 1
f = g * h
e = M[j+8]
m1 = M[j+16]
M[new] = m1
b = M[f]
c = e + 8
d = c
m2 = M[new]
k = m2 + 4
j = b
```



Register Allocation in Practice

- Spills are uncommon on machines with large, uniform, register files.
- When a temporary is spilled, new instructions to load/store it are generated.
- The new instructions cut the temporaries lifetime into smaller parts, thus reducing interference.
- Register allocation is performed again on the modified code, potentially leading to other spills and restarts.
- **Precoloring:** Certain temporaries may have a predefined register (e.g. argument register). Then we color the interference graph, but start with preallocated colors for certain vertices.
- **Coalescing:** Temporaries that move data from each other can be constrained to have the same register, thereby avoiding data movement.

SSA forms and Register Allocation

- SSA form eliminates spurious relationships in the interference graph.
- SSA may help avoid spills.

```
g = M[j+12]
h = k - 1
f = g * h
e = M[j+8]
m = M[j+16]
b = M[f]
c = e + 8
d = c
k = m + 4
j = b
```

 $\Rightarrow$

```
g = M[j1+12]
h = k1 - 1
f = g * h
e = M[j1+8]
m = M[j1+16]
b = M[f]
c = e + 8
d = c
k2 = m + 4
j2 = b
```

