based on various sources, including slides from:
José Nelson Amaral – University of Alberta,
David Walker – Princeton University

Register Allocation

Compilers course

Masters in Informatics and Computing Engineering (MIEIC), 3rd Year

João M. P. Cardoso
Email: jmpc@fe.up.pt

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Outline

- Introduction to Register Allocation
- Variables' Live Ranges
- Register Allocation by Graph Coloring
  - Heuristics
  - Spilling
- Summary

Register Allocation

- Store as many variables as possible in registers
- Use each register to store as many variables as possible (registers are limited resources)
  - use live range (also known as “lifetime interval”) of variables
- One the optimizations with highest impact (code size and performance)

Variables' Live Ranges

- Duration in the code from a definition of a variable and a use of this variable reached by that definition
- See Liveness Analysis

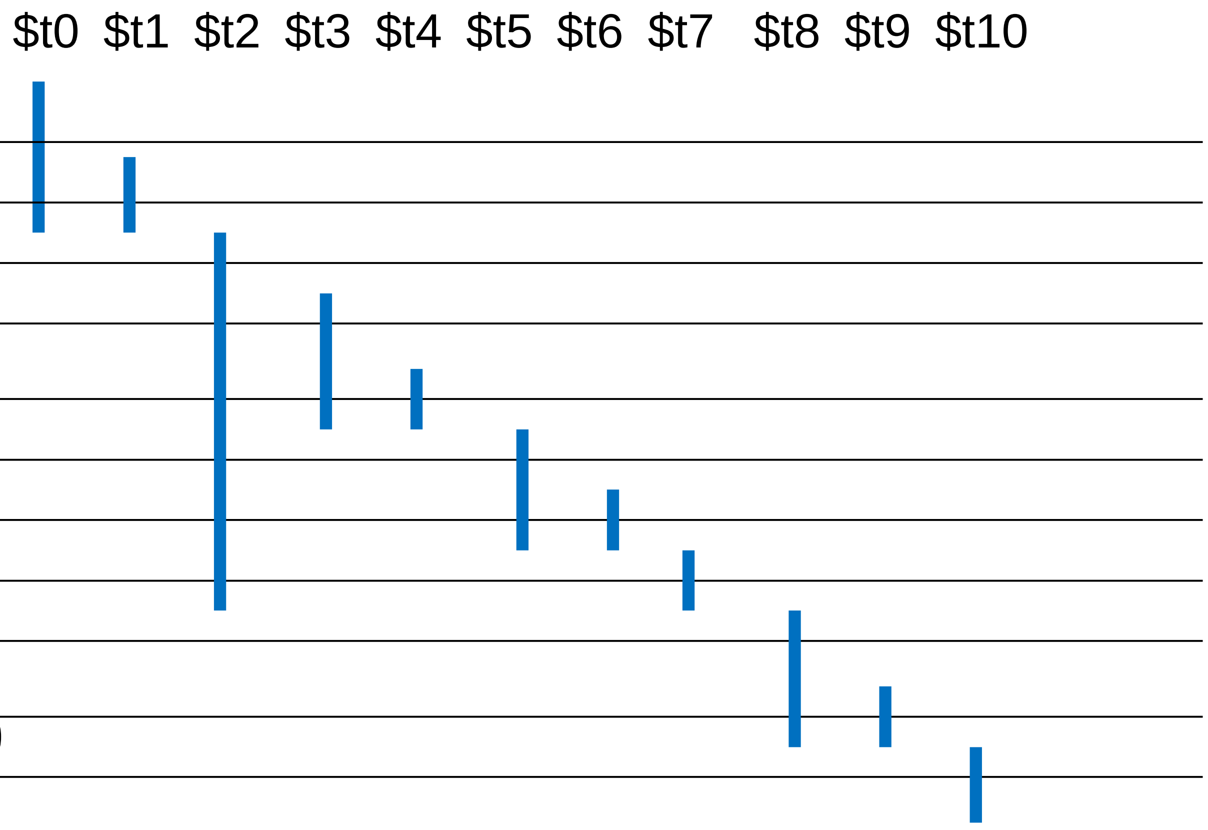
```
a = b*c;
d=b*b+e;
e=a+d;
```

lifetime of a

Variables' Live Ranges

- Variables' ($t registers) live ranges in the following MIPS code

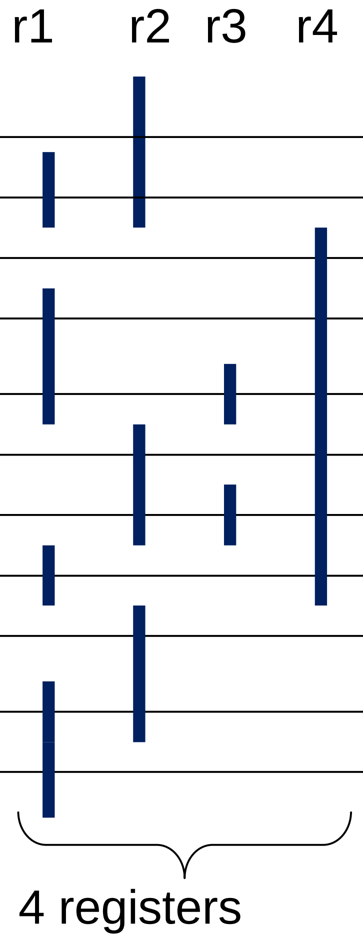
```
lw $t0, 4($sp)
lw $t1, 8($sp)
mult $t2, $t1, $t0
lw $t3, 8($sp)
lw $t4, 8($sp)
mult $t5, $t3, $t4
lw $t6, 0($sp)
mult $t7, $t5, $t6
add $t8, $t7, $t2
lw $t9, 12($sp)
add $t10, $t8, $t9
sw  $t10, 16($sp)
```



Register Allocation

- Based on the variables' ($t registers) live ranges try to use each register to store more than one variable ($t register)

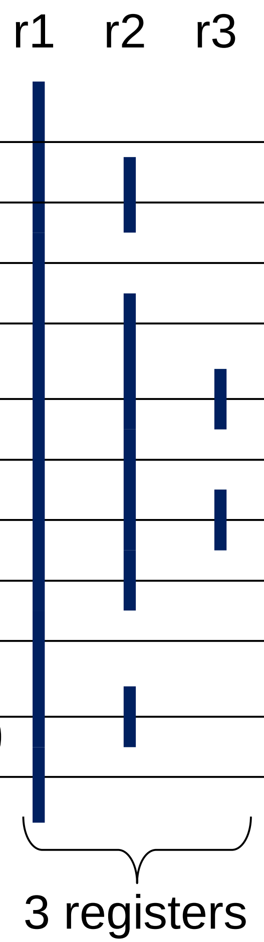
```
lw $t0, 4($sp)
lw $t1, 8($sp)
mult $t2, $t1, $t0
lw $t3, 8($sp)
lw $t4, 8($sp)
mult $t5, $t3, $t4
lw $t6, 0($sp)
mult $t7, $t5, $t6
add $t8, $t7, $t2
lw $t9, 12($sp)
add $t10, $t8, $t9
sw  $t10, 16($sp)
```



Register Allocation

- Let's try to reduce the number of registers t in the following MIPS code

```
lw $t0, 4($sp)
lw $t1, 8($sp)
mult $t2, $t1, $t0
lw $t3, 8($sp)
lw $t4, 8($sp)
mult $t5, $t3, $t4
lw $t6, 0($sp)
mult $t7, $t5, $t6
add $t8, $t7, $t2
lw $t9, 12($sp)
add $t10, $t8, $t9
sw  $t10, 16($sp)
```



Register Allocation

- Determine the live range for each variable
  - Use liveness analysis
- Allocate a register to one or more variables
- How?
  - Graph Coloring (problem NP-complete)
    - Use heuristics

Register Allocation by Graph Coloring

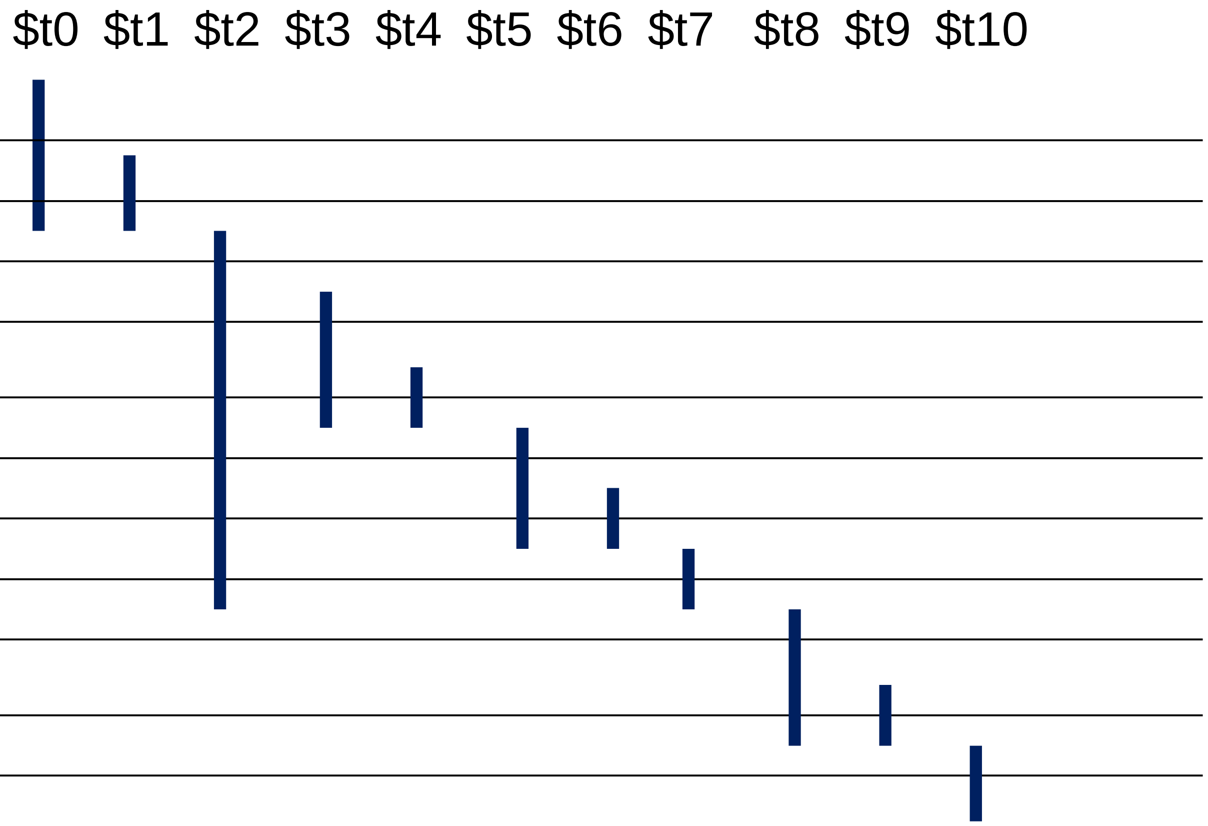
- Graph Coloring
  - Calculate the live range for each variable
  - Construct the Register-Interference Graph* (there is interference when 2 variables have lifetimes with non-null intersection)
    - Edges represent interference
    - Nodes represent variables
  - Find the minimum colors or the k colors
  - Each color corresponds to a register
    - i.e., number of registers = number of colors

* Also known as Register-Conflict Graph

Register Allocation by Graph Coloring

- Variables' ($t registers) Live Range

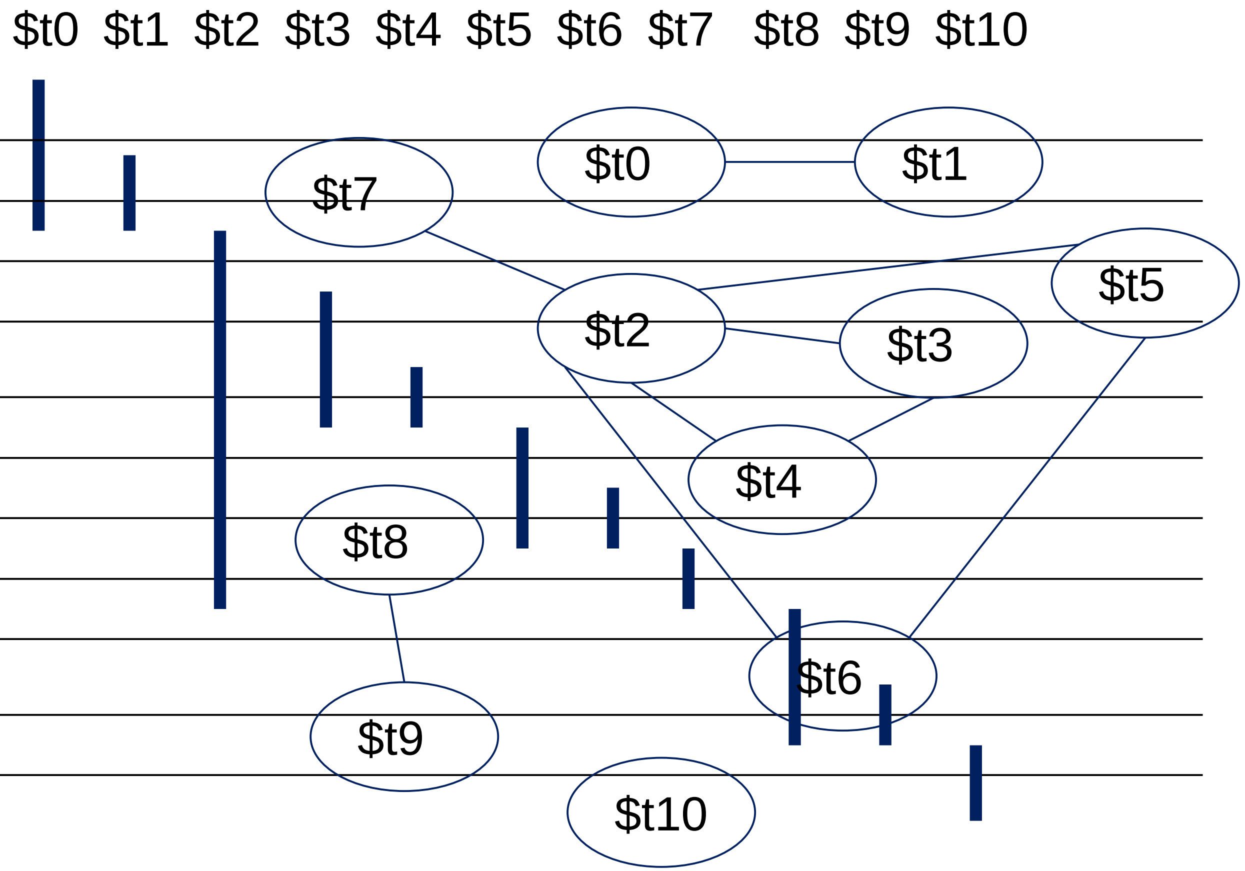
```
lw $t0, 4($sp)
lw $t1, 8($sp)
mult $t2, $t1, $t0
lw $t3, 8($sp)
lw $t4, 8($sp)
mult $t5, $t3, $t4
lw $t6, 0($sp)
mult $t7, $t5, $t6
add $t8, $t7, $t2
lw $t9, 12($sp)
add $t10, $t8, $t9
sw  $t10, 16($sp)
```



Register Allocation by Graph Coloring

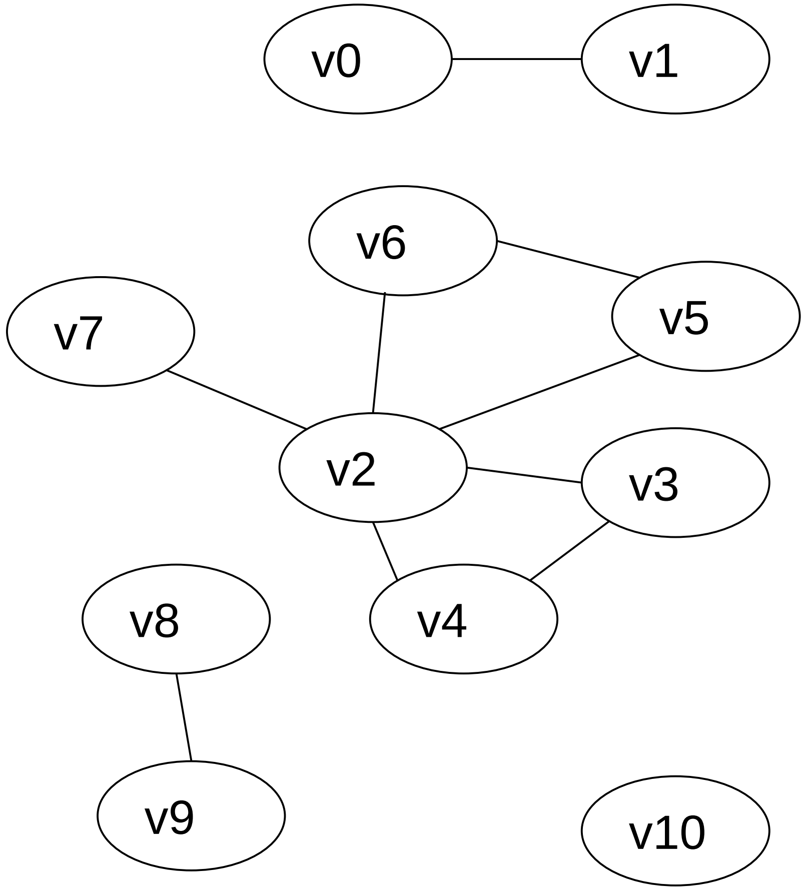
- Register-Interference Graph (IG)

```
lw $t0, 4($sp)
lw $t1, 8($sp)
mult $t2, $t1, $t0
lw $t3, 8($sp)
lw $t4, 8($sp)
mult $t5, $t3, $t4
lw $t6, 0($sp)
mult $t7, $t5, $t6
add $t8, $t7, $t2
lw $t9, 12($sp)
add $t10, $t8, $t9
sw  $t10, 16($sp)
```



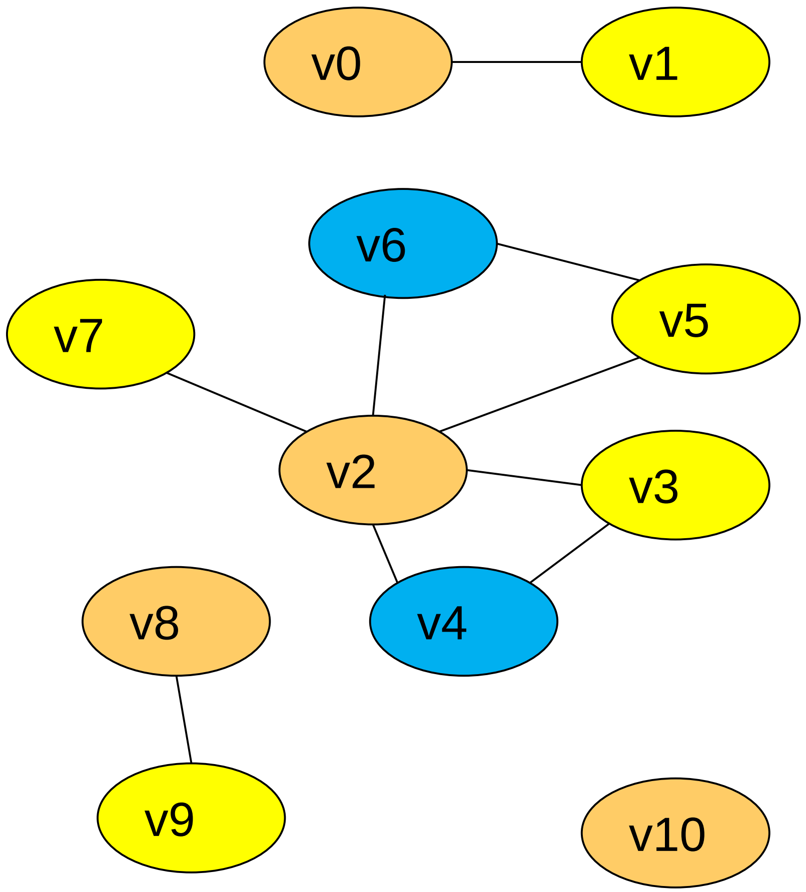
Register Allocation by Graph Coloring

- Register-Interference Graph
  - Interference (edge) between two variables (nodes) indicates that the two variables could not be stored in the same register



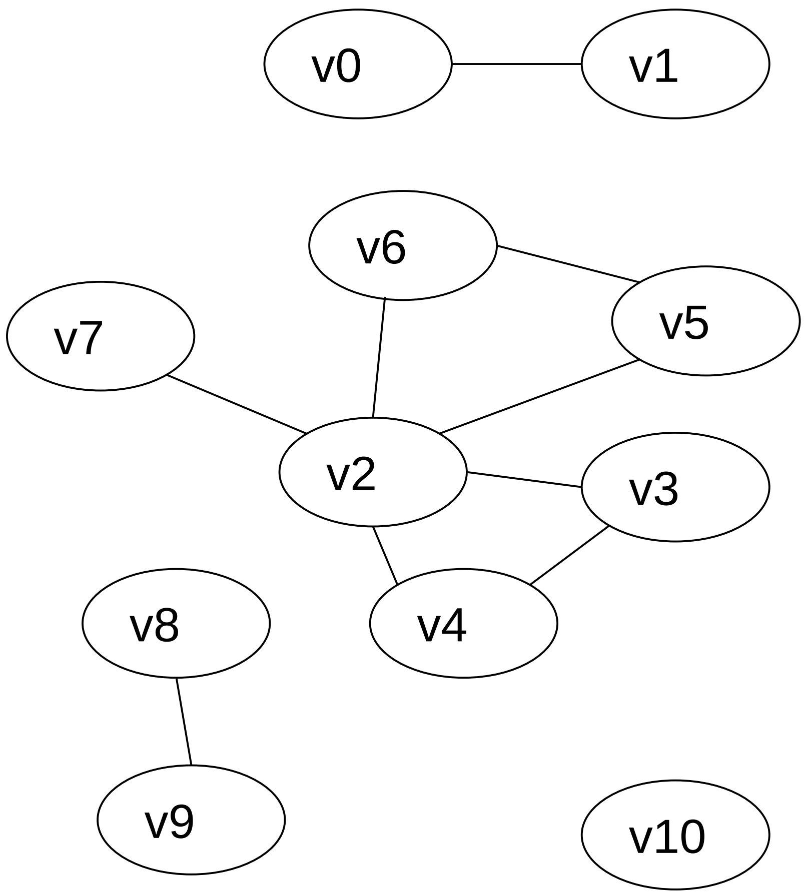
Register Allocation by Graph Coloring

- Register-Inference Graph
- After Coloring:
  - Number of colors indicate the number of necessary registers



Register Allocation by Graph Coloring

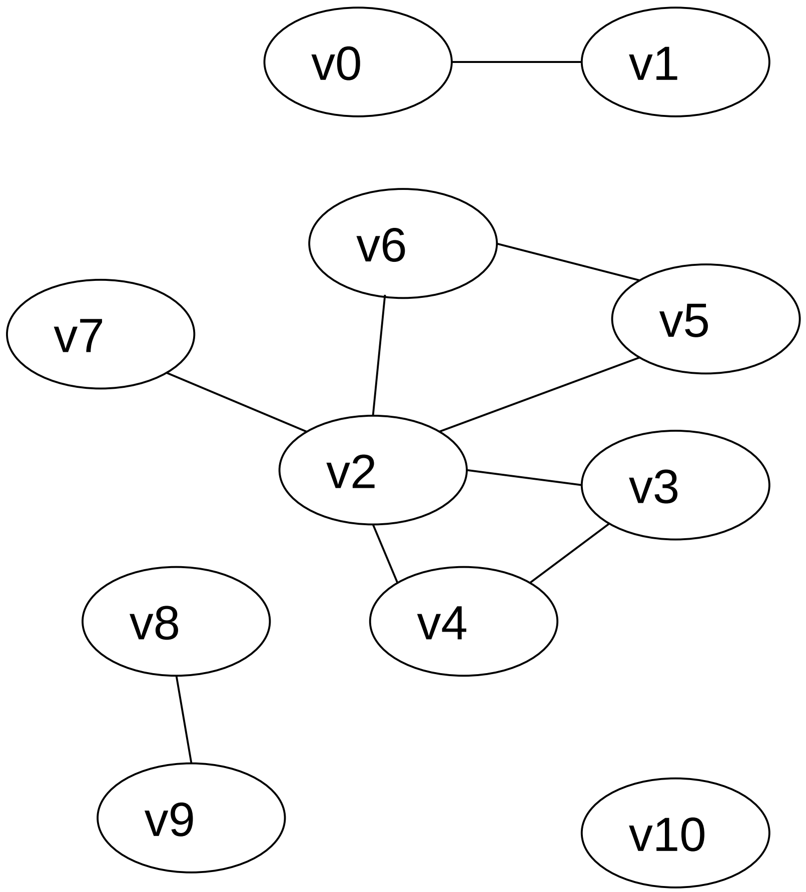
- A graph is **k**-colorable if each node can be assigned one of **k** colors in such a way that no two adjacent nodes have the same color.



Register Allocation by Graph Coloring

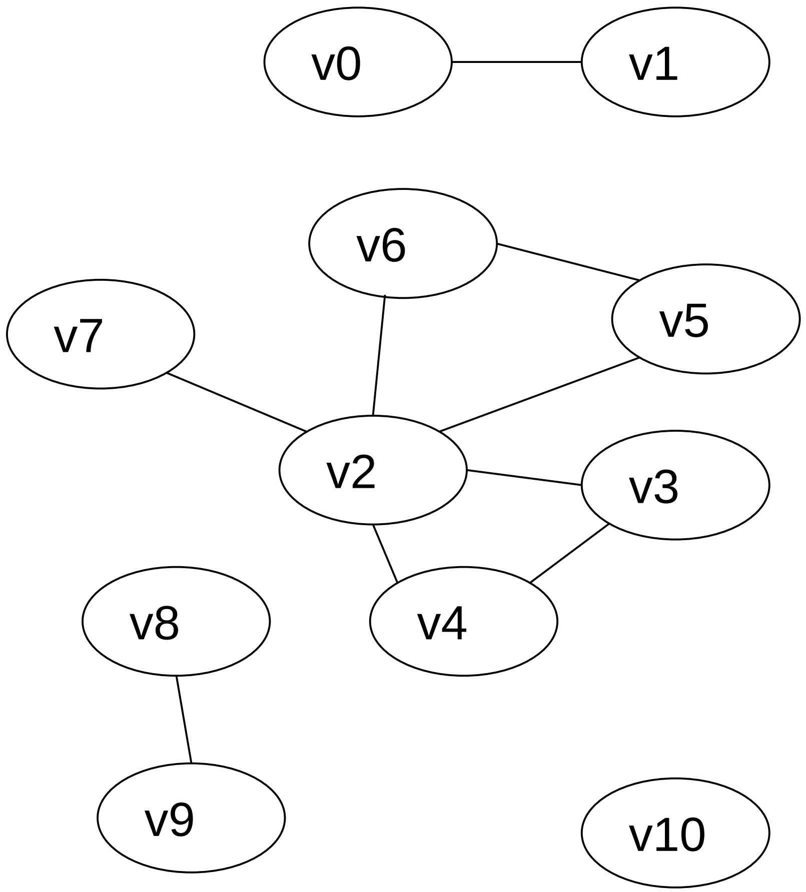
Steps:

1. Build the register interference graph,
2. Attempt to find a k-coloring for the the interference graph.



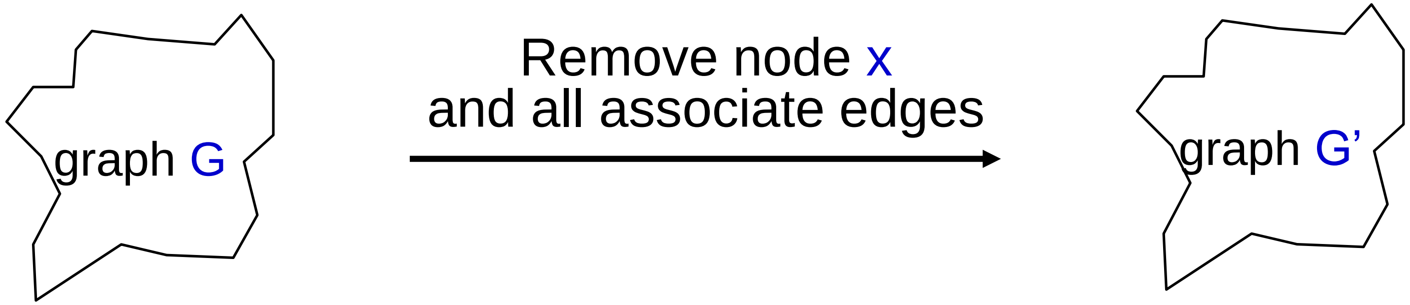
Register Allocation by Graph Coloring

- The problem of determining if an undirected graph is k-colorable is NP-hard for k ≥ 3
- It is also hard to find approximate solutions to the graph coloring problem



Heuristic Solution for Graph Coloring

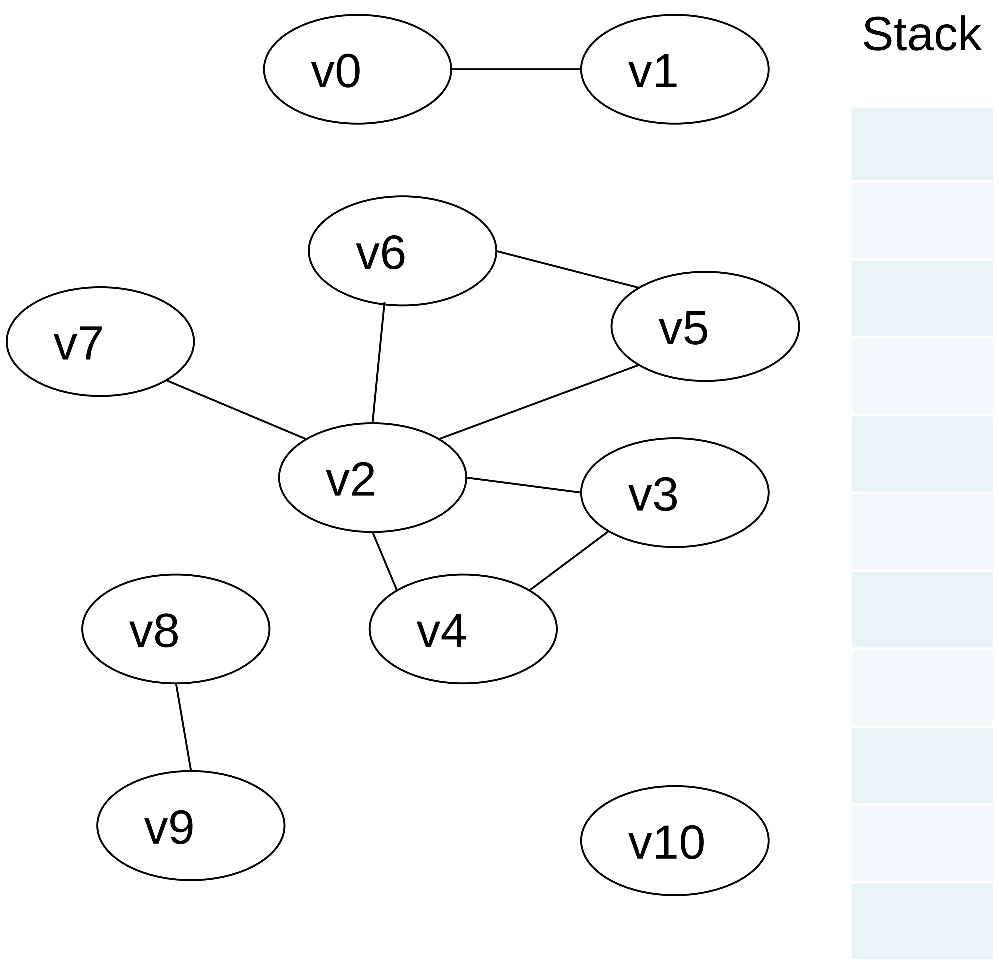
- Key observation:
  - Let G be an undirected graph
  - Let x be a node of G such that degree(x) < k



Then G is k-colorable if G' is k-colorable.

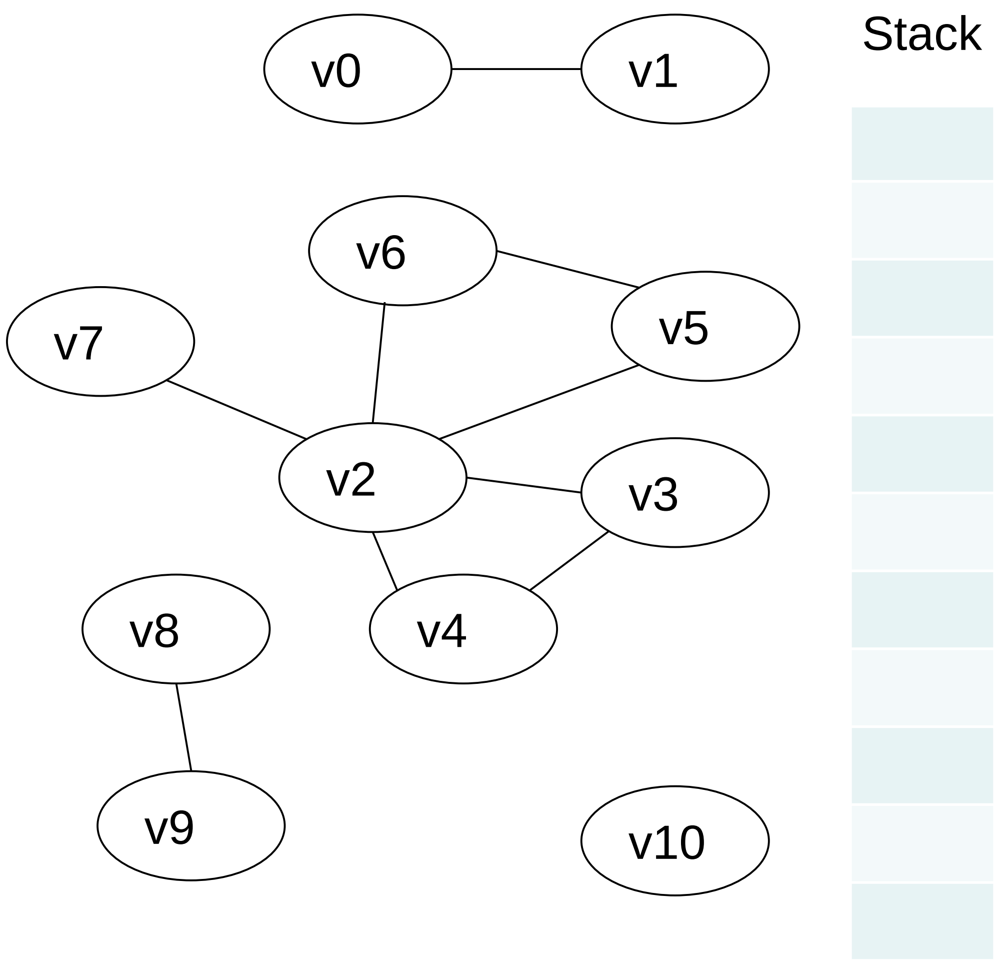
Heuristic Solution for Graph Coloring

- Kempe's algorithm [1879] for finding a K-coloring of a graph
- Step 1 (simplify): find a node n with degree(n)<k and cut it out of the graph (remember this node on a stack for later stages)



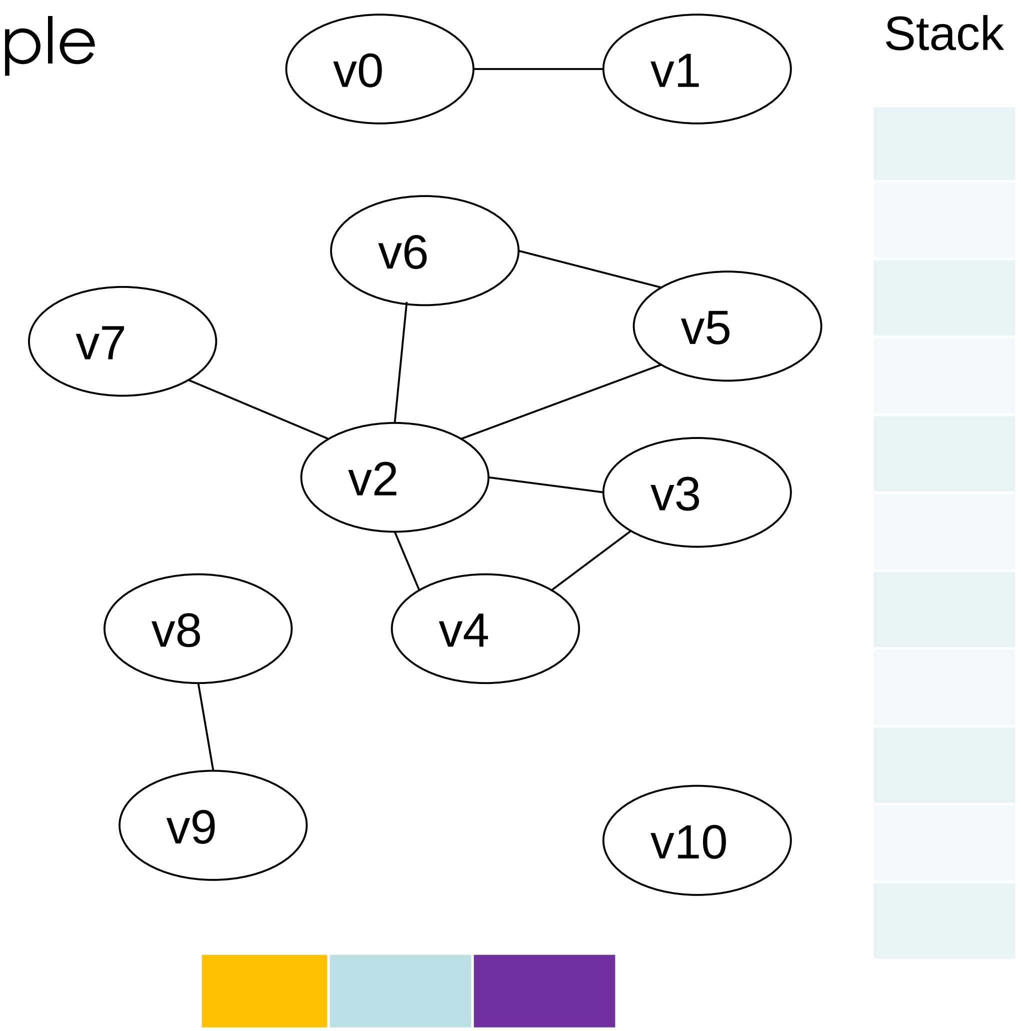
Heuristic Solution for Graph Coloring

- Once a coloring is found for the simpler graph, we can always color the node we saved on the stack
- Step 2 (color): when the simplified subgraph has been colored, add back the node on the top of the stack and assign it a color not taken by one of the adjacent nodes



Heuristic Solution for Graph Coloring

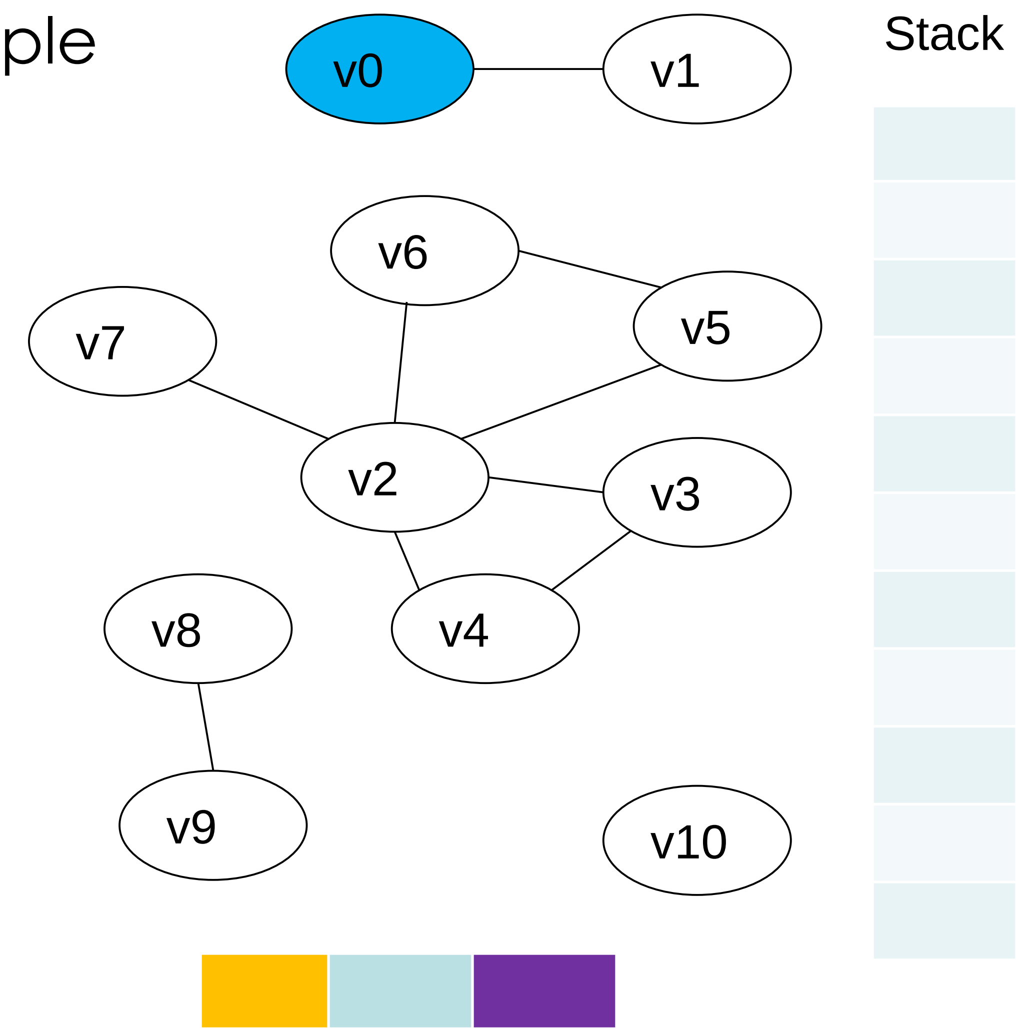
- Let's go back to the example
- Consider k=3



Heuristic Solution for Graph Coloring

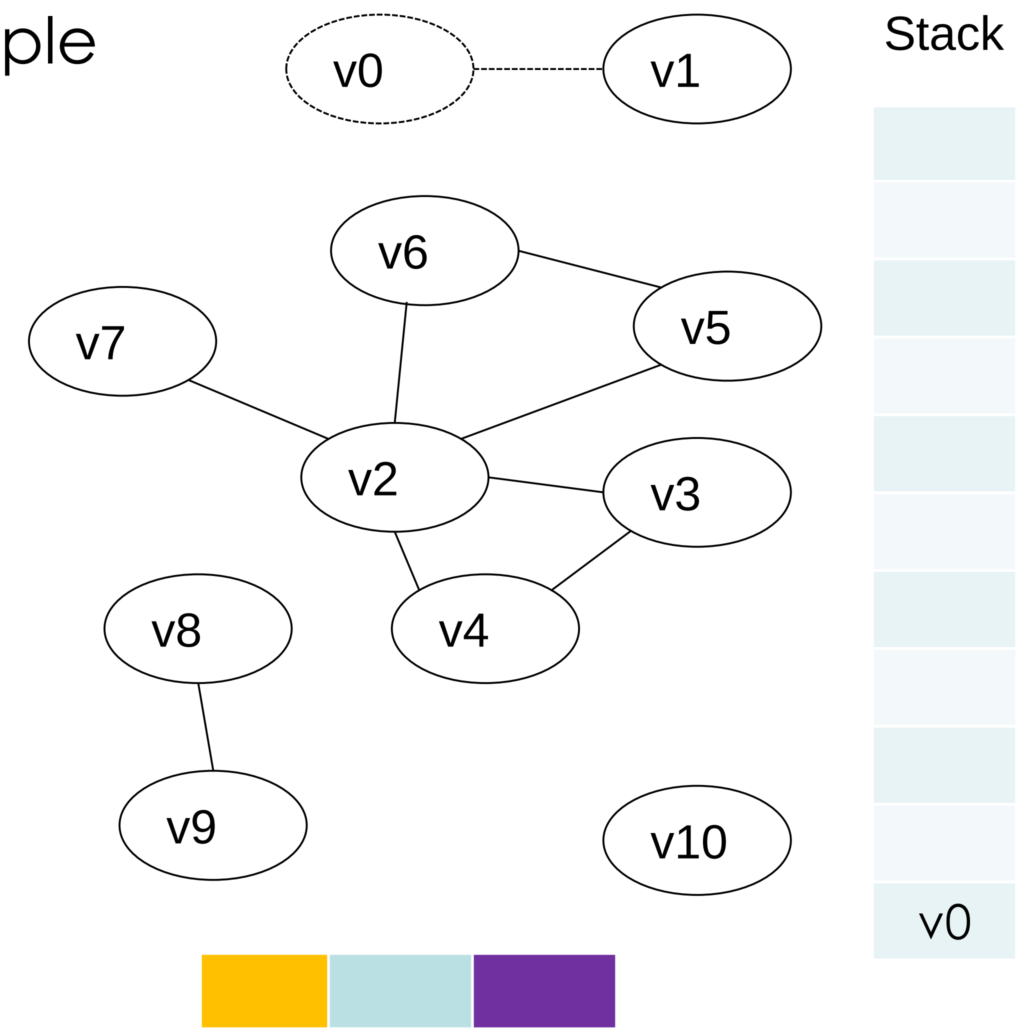
- Let's go back to the example
- Consider k=3

- Edges(v0) < 3



Heuristic Solution for Graph Coloring

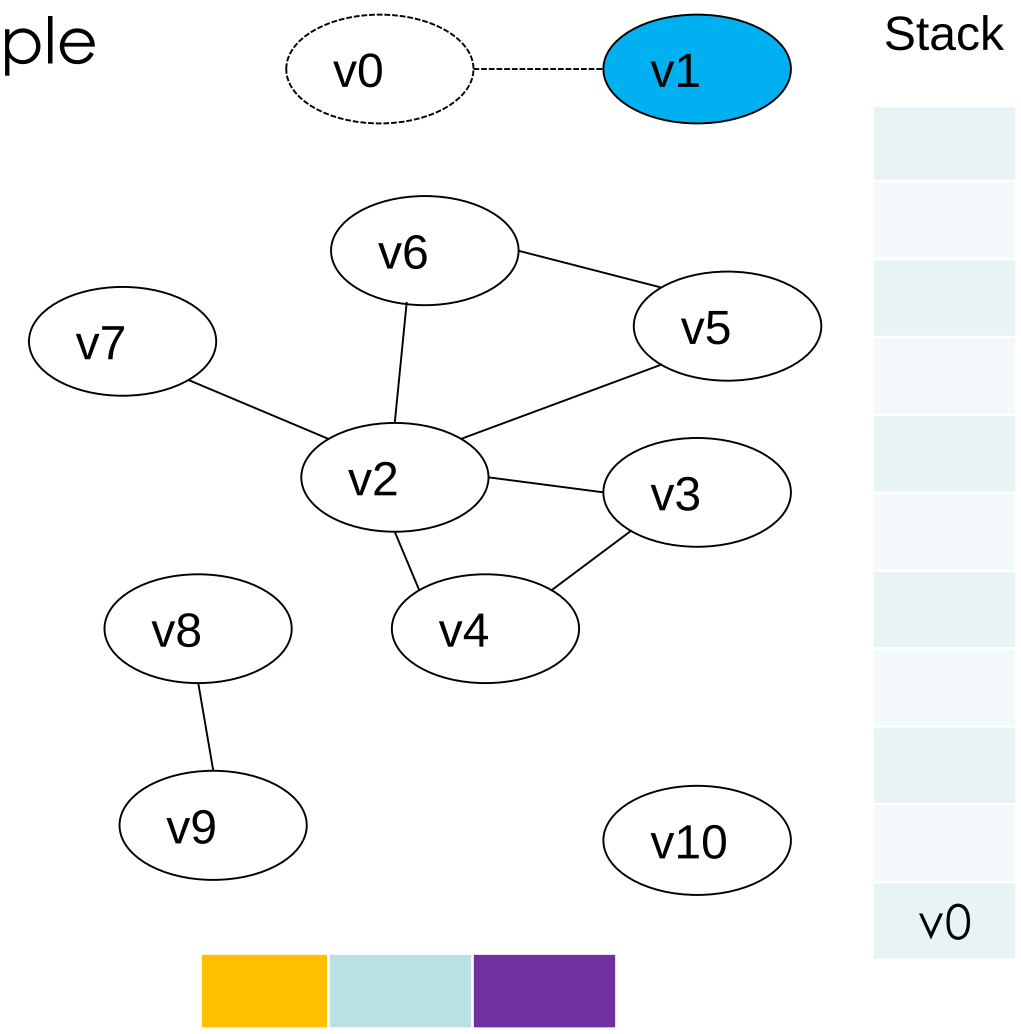
- Let's go back to the example
- Consider k=3



Heuristic Solution for Graph Coloring

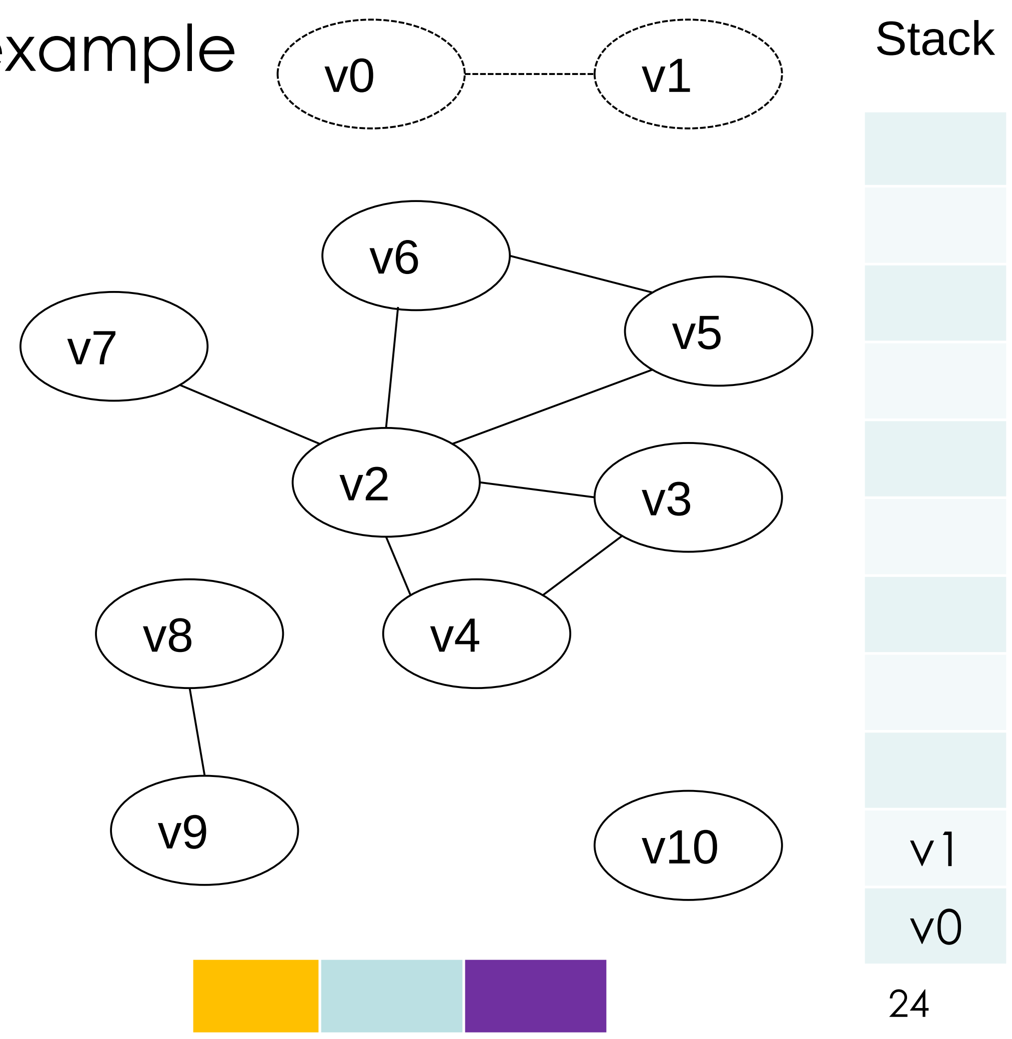
- Let's go back to the example
- Consider k=3

- Edges(v1) < 3



Heuristic Solution for Graph Coloring

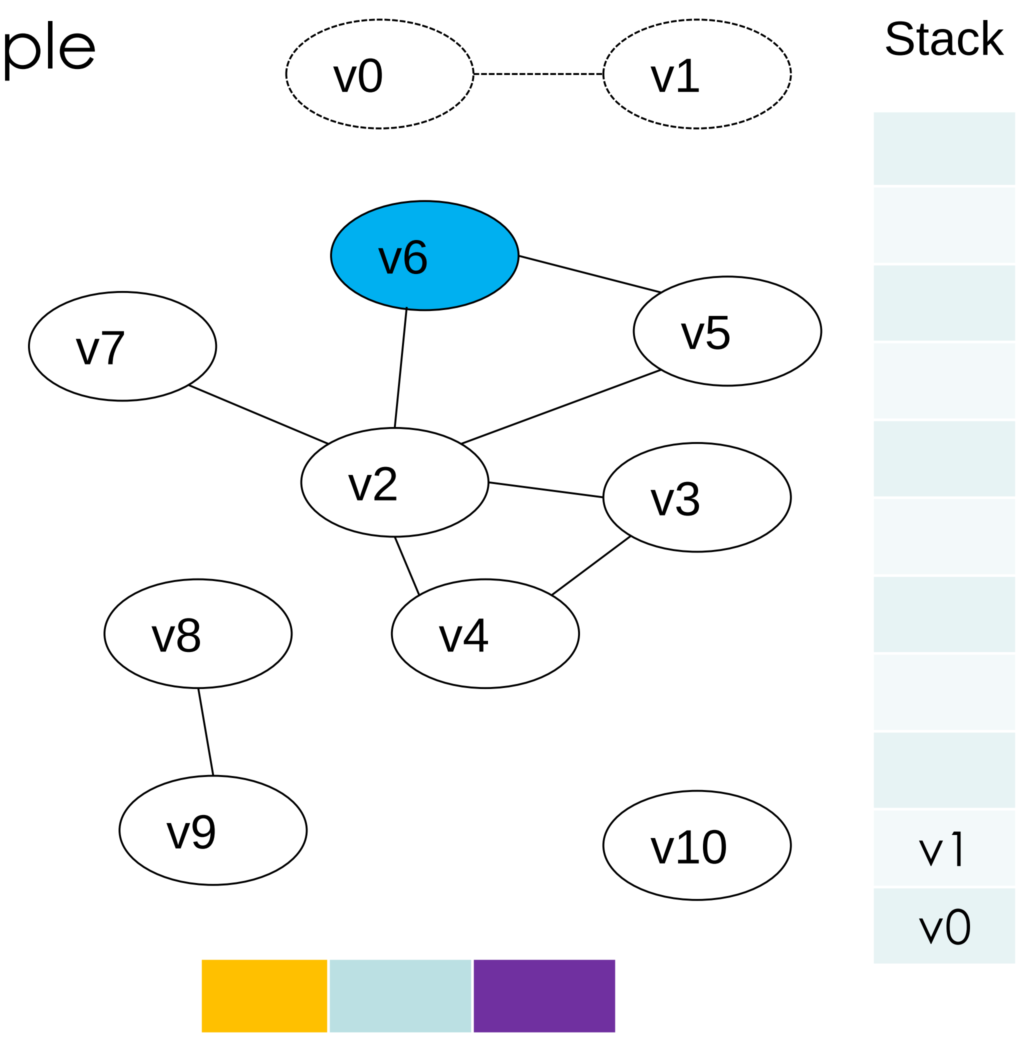
- Let's go back to the example
- Consider k=3



Heuristic Solution for Graph Coloring

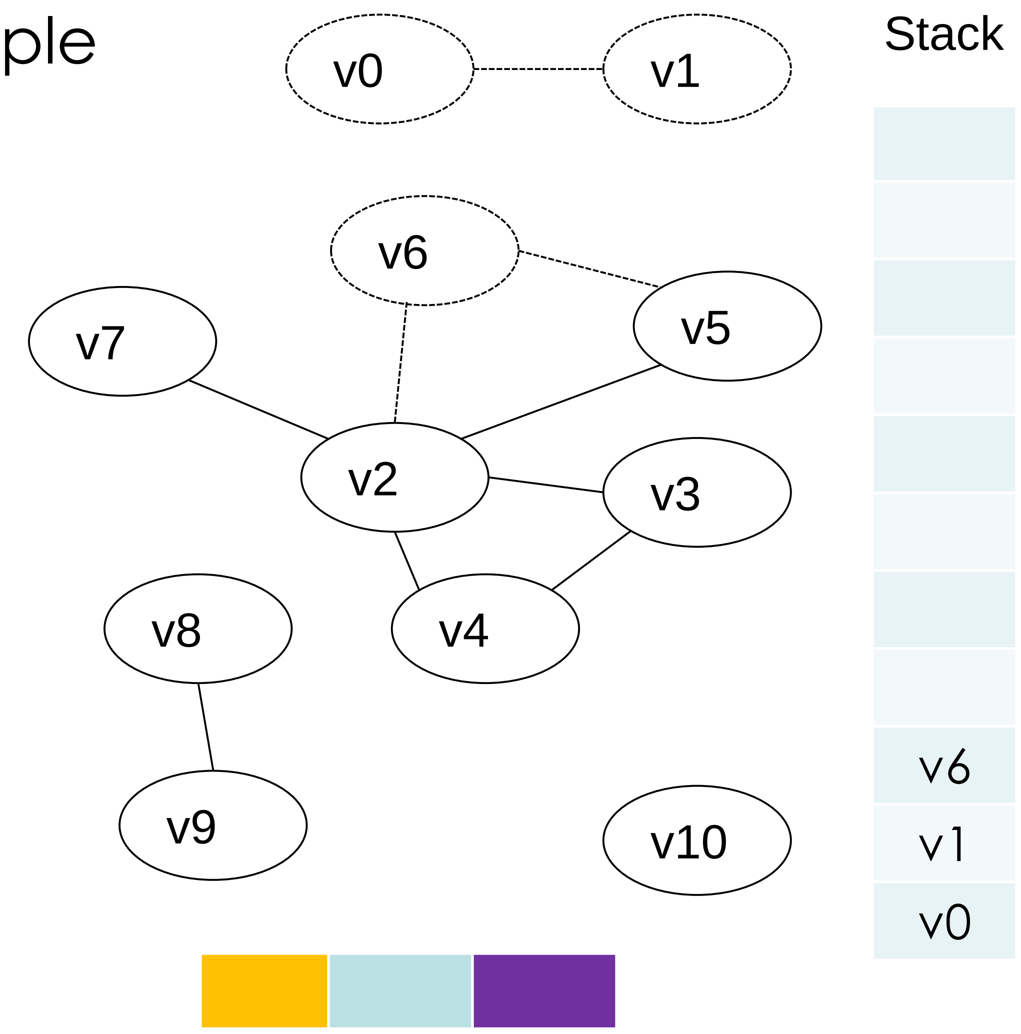
- Let's go back to the example
- Consider k=3

- Edges(v6) < 3



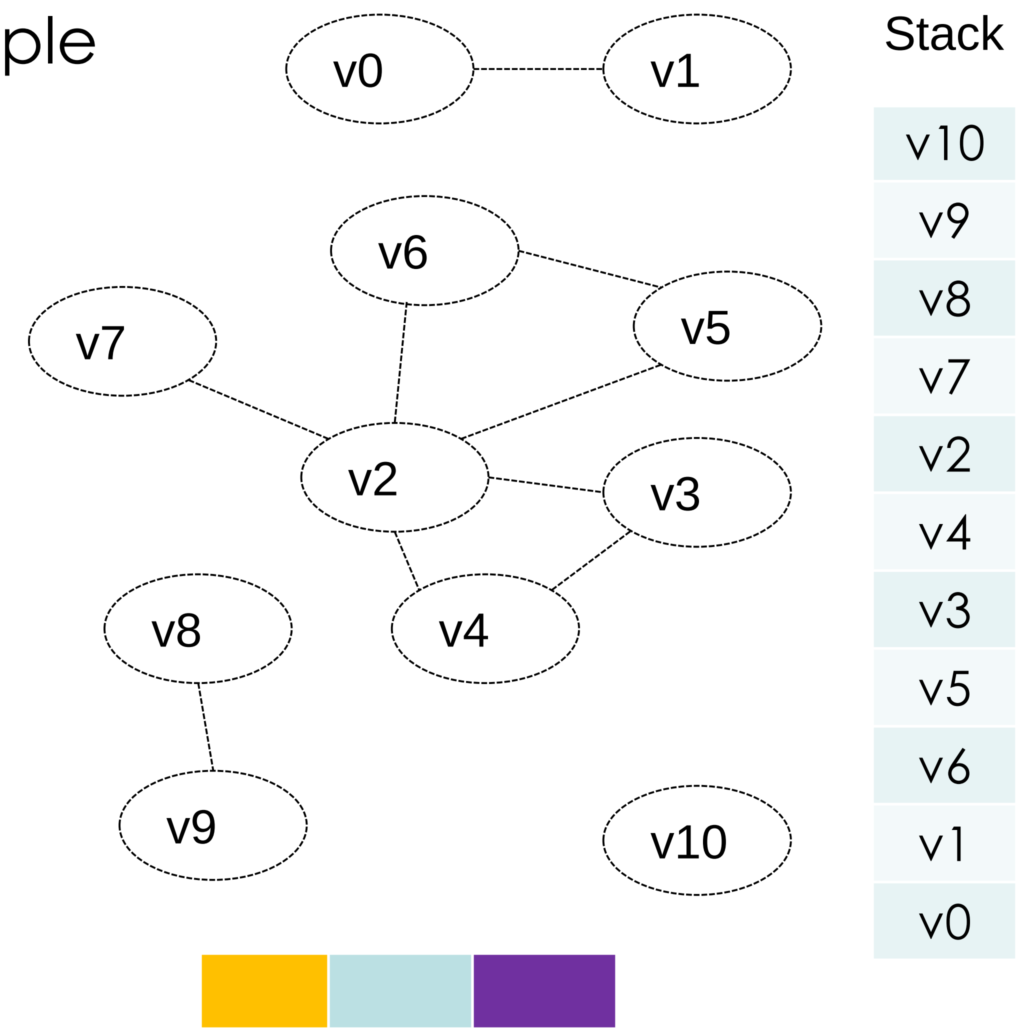
Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3



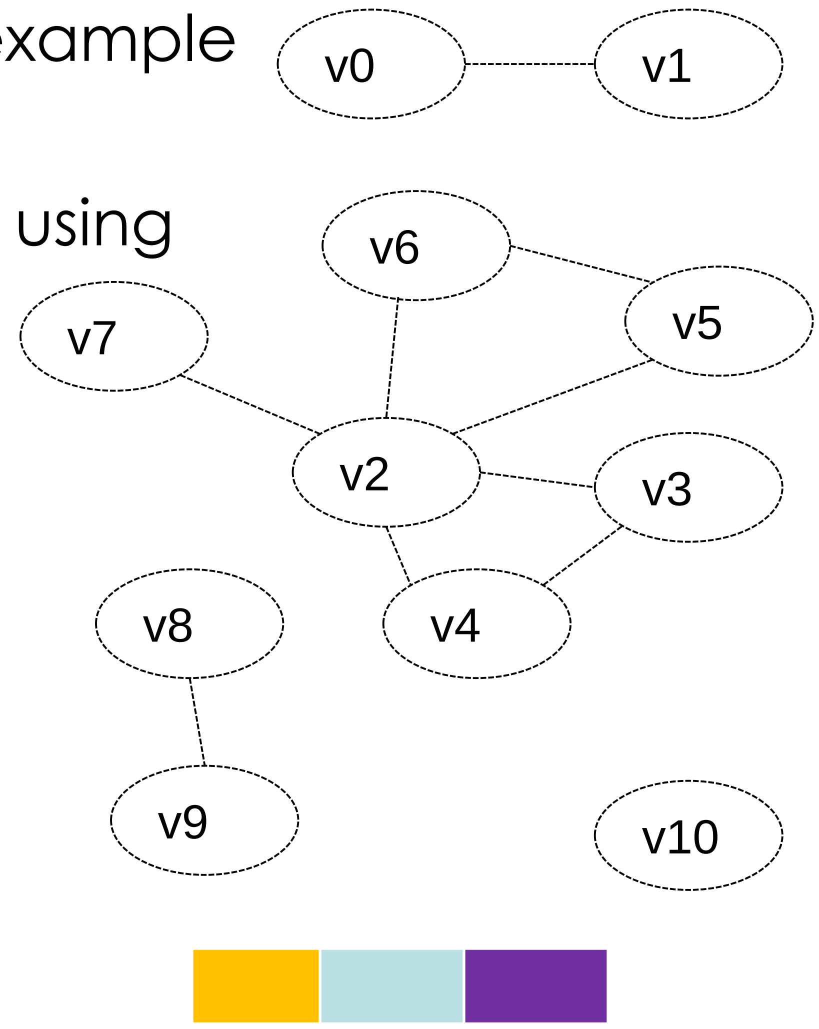
Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- After some steps…



Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- Now we start coloring using the top of the stack

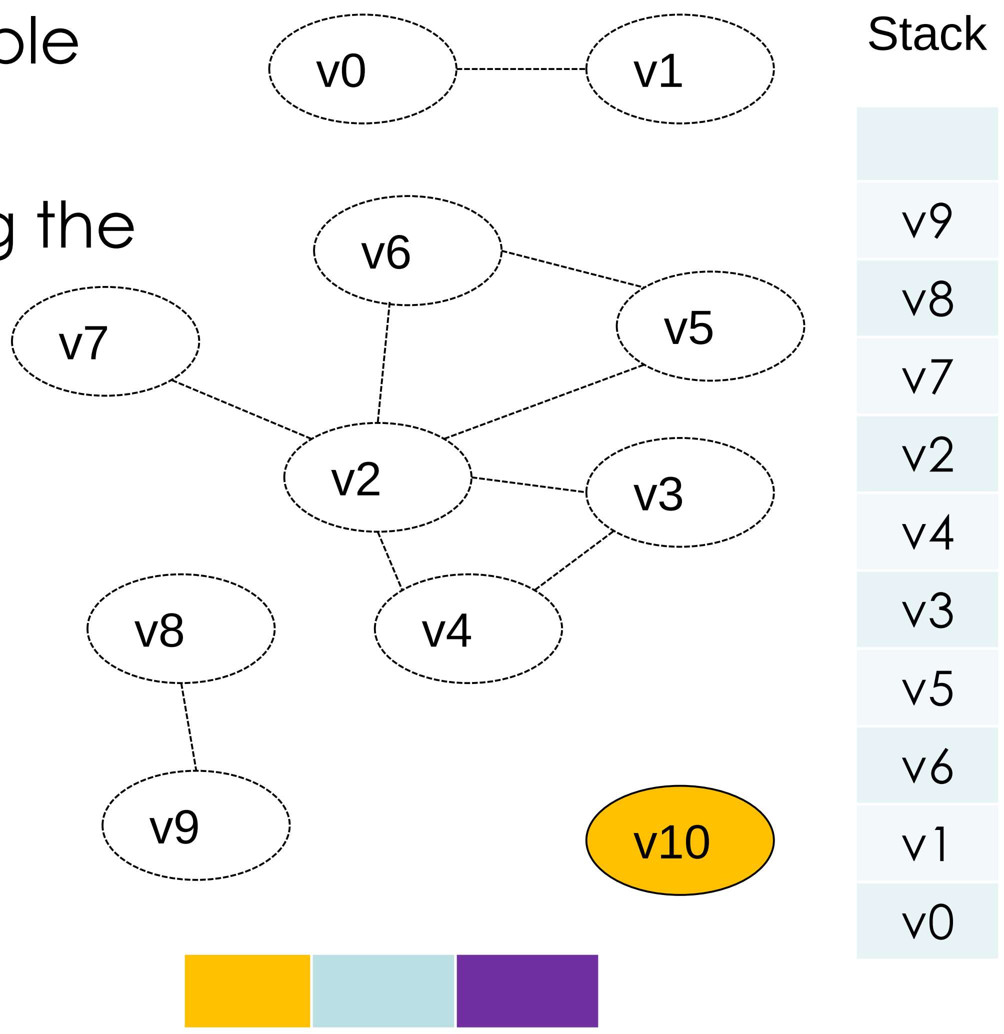


Stack

v10
v9
v8
v7
v2
v4
v3
v5
v6
v1
v0

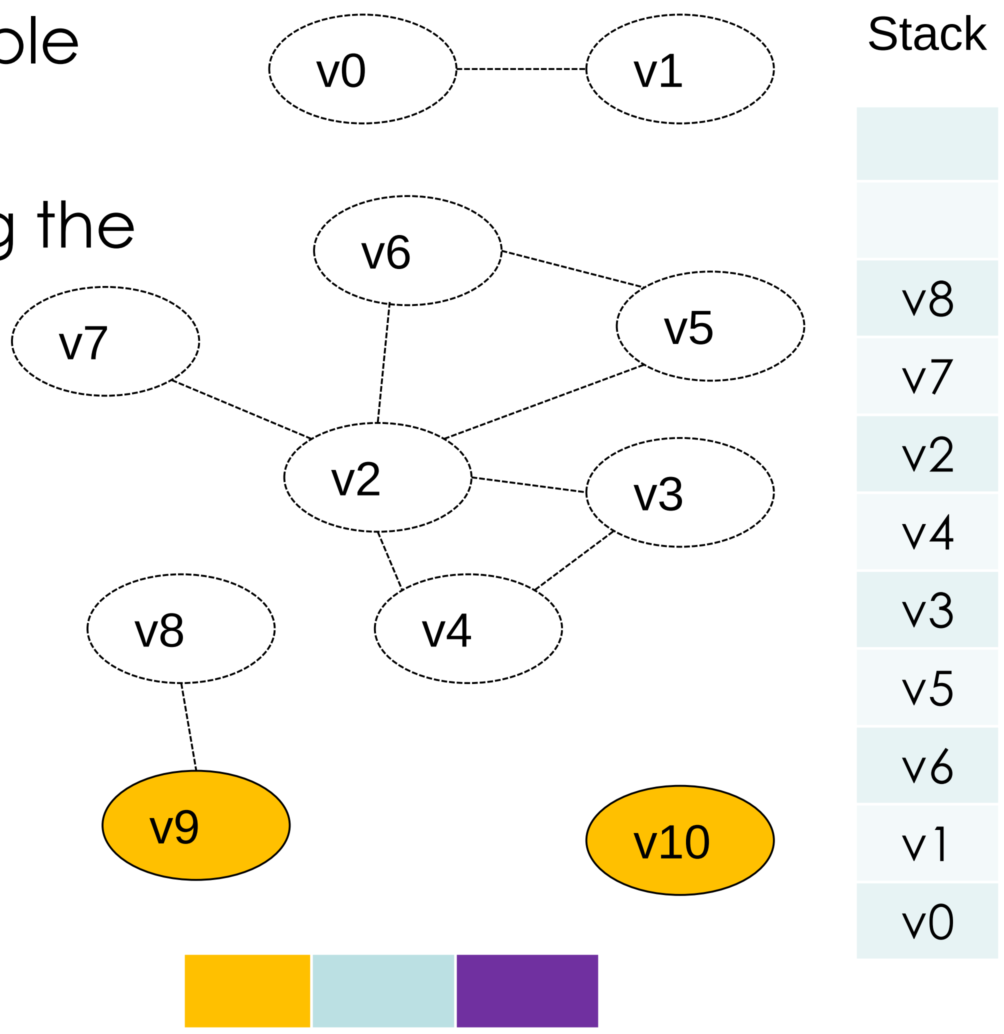
Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- Now we start coloring using the top of the stack
  - v10



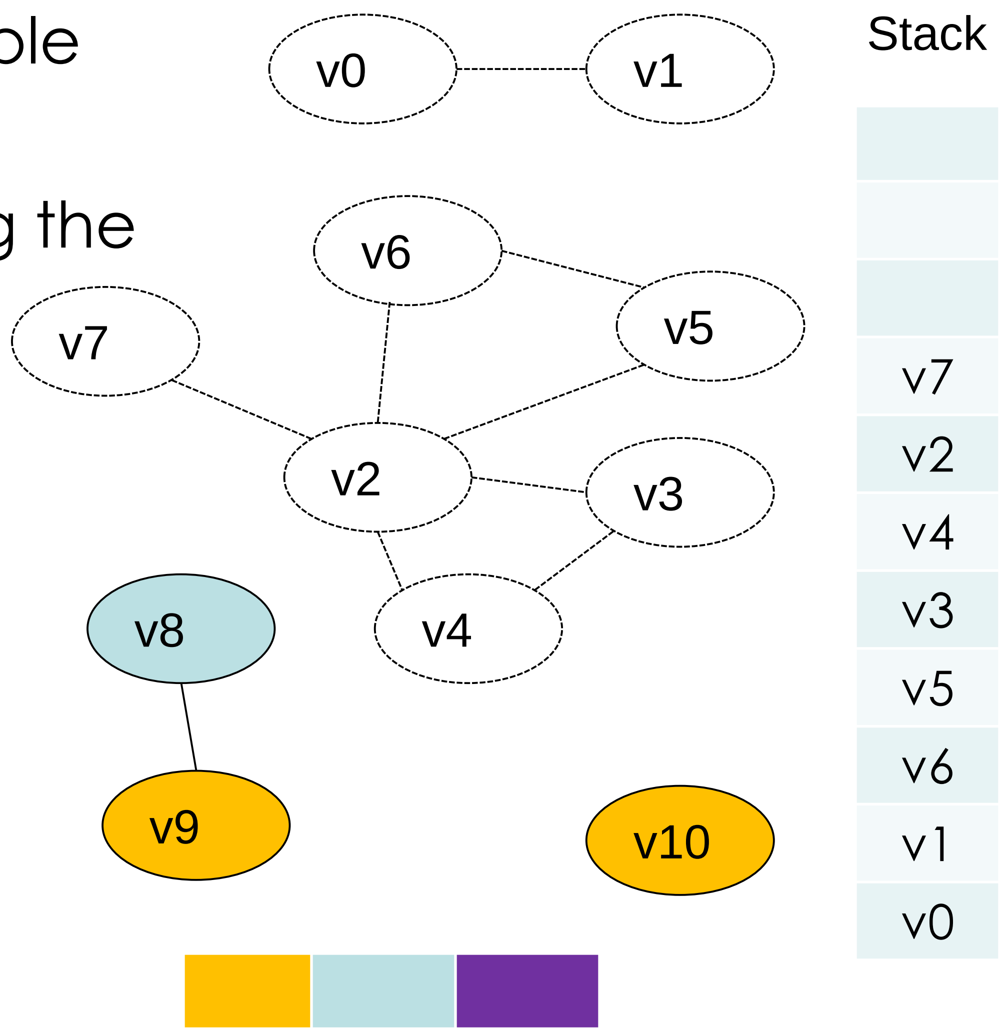
Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- Now we start coloring using the top of the stack
  - v9



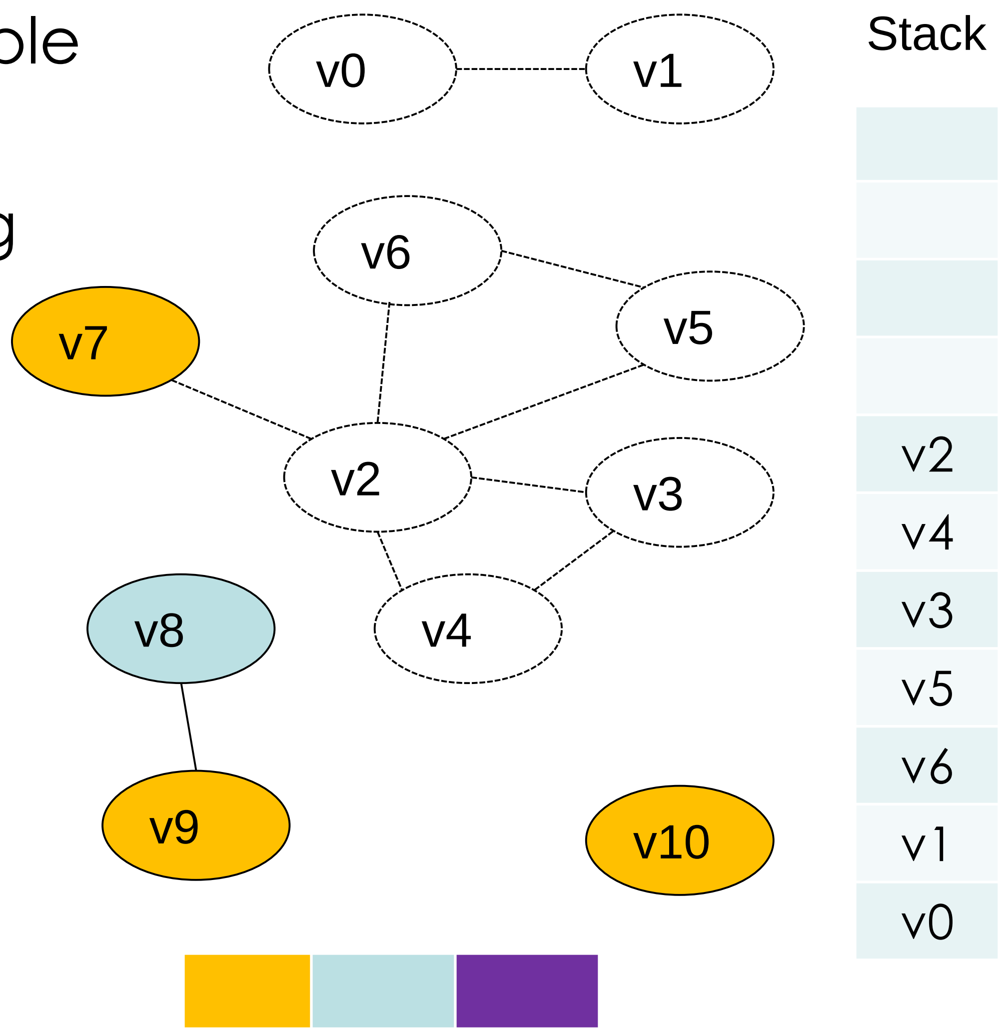
Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- Now we start coloring using the top of the stack
  - v8



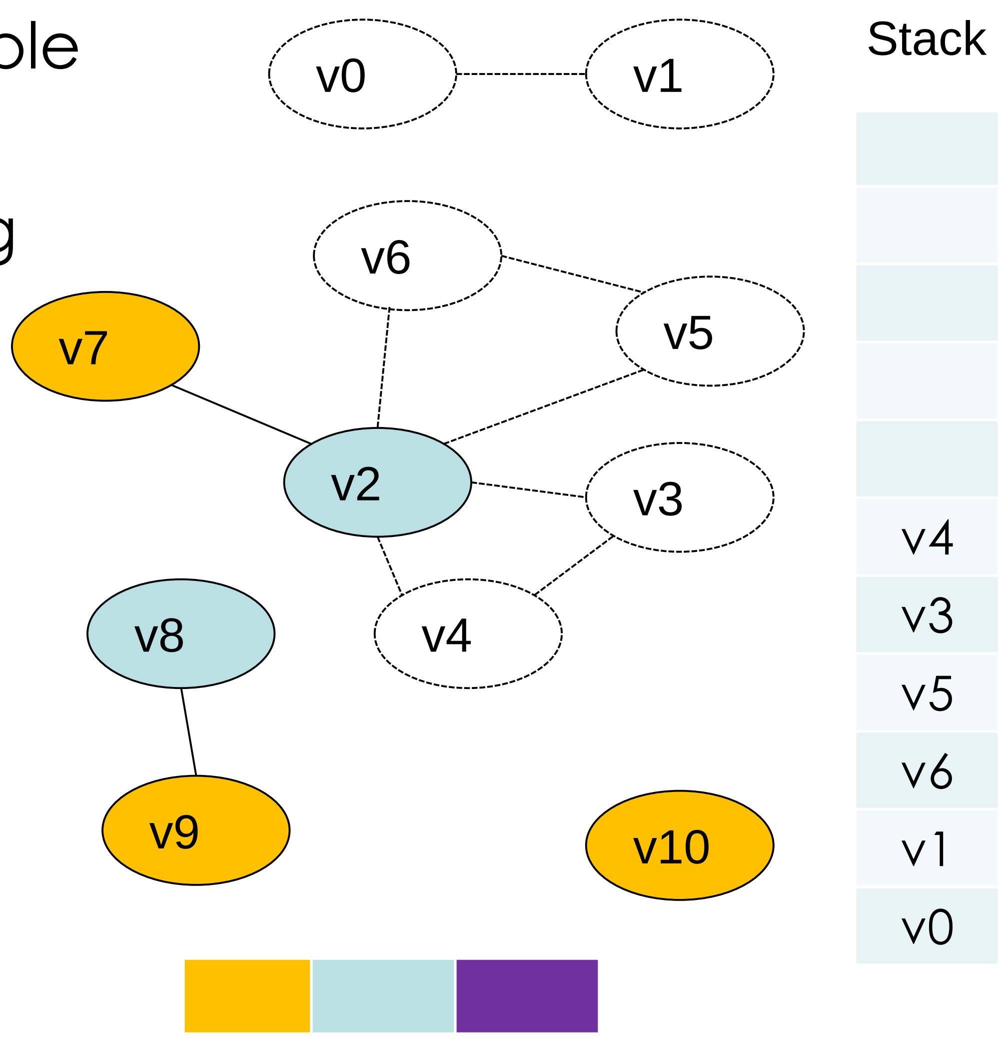
Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- Now we start coloring using the top of the stack
  - v7



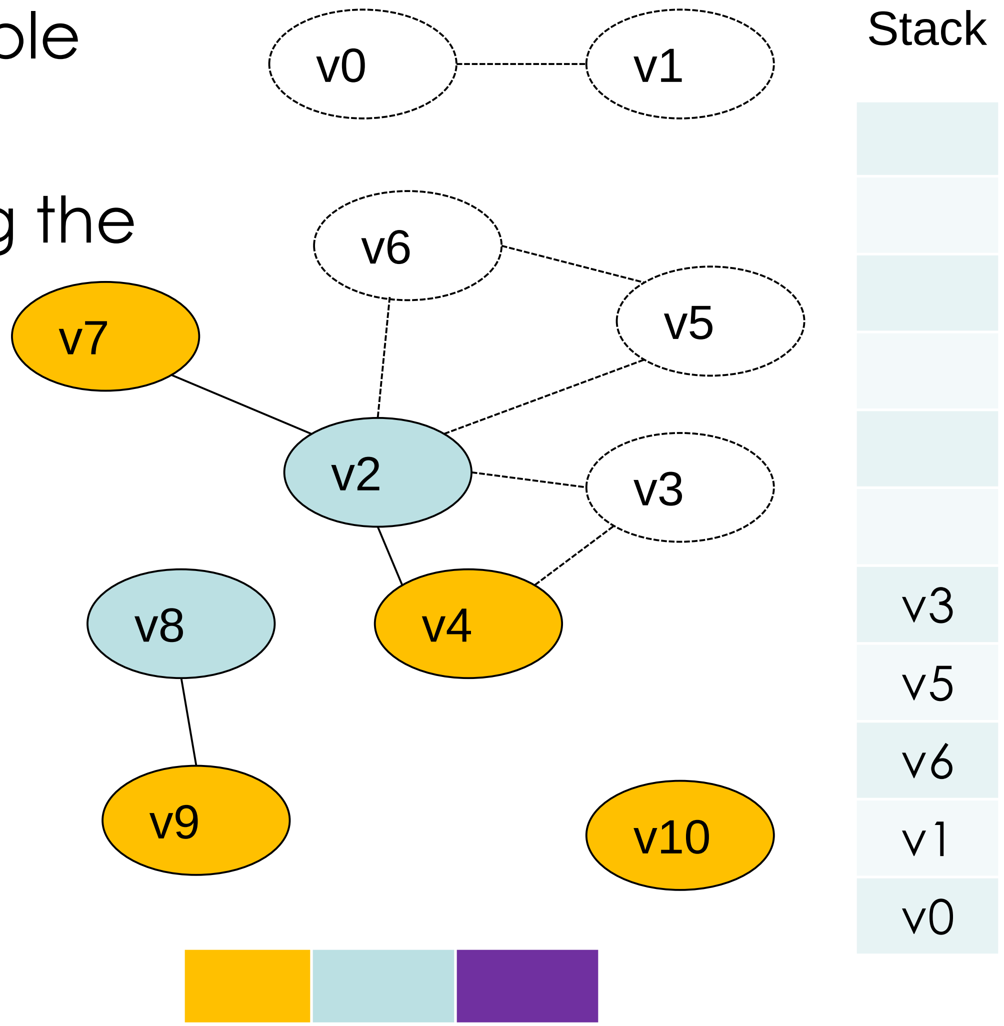
Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- Now we start coloring using the top of the stack
  - v2



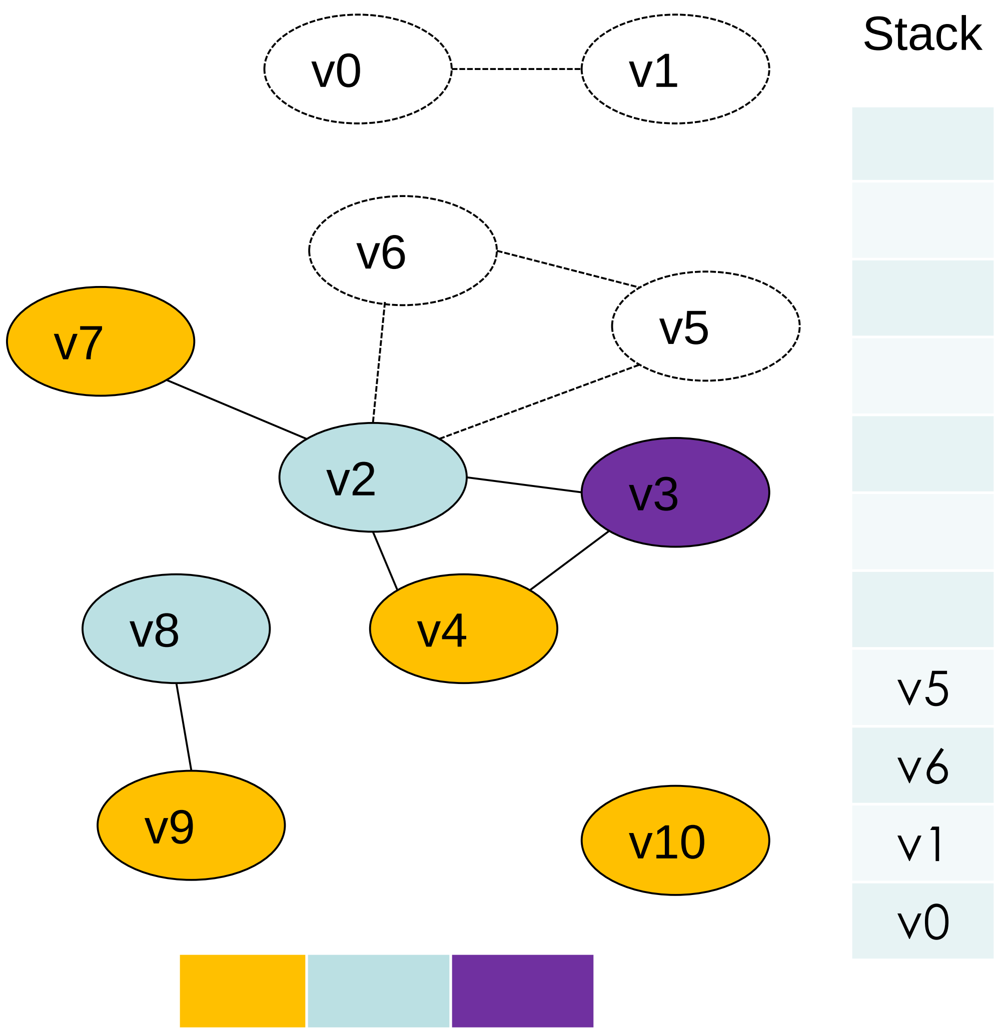
Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- Now we start coloring using the top of the stack
  - v4



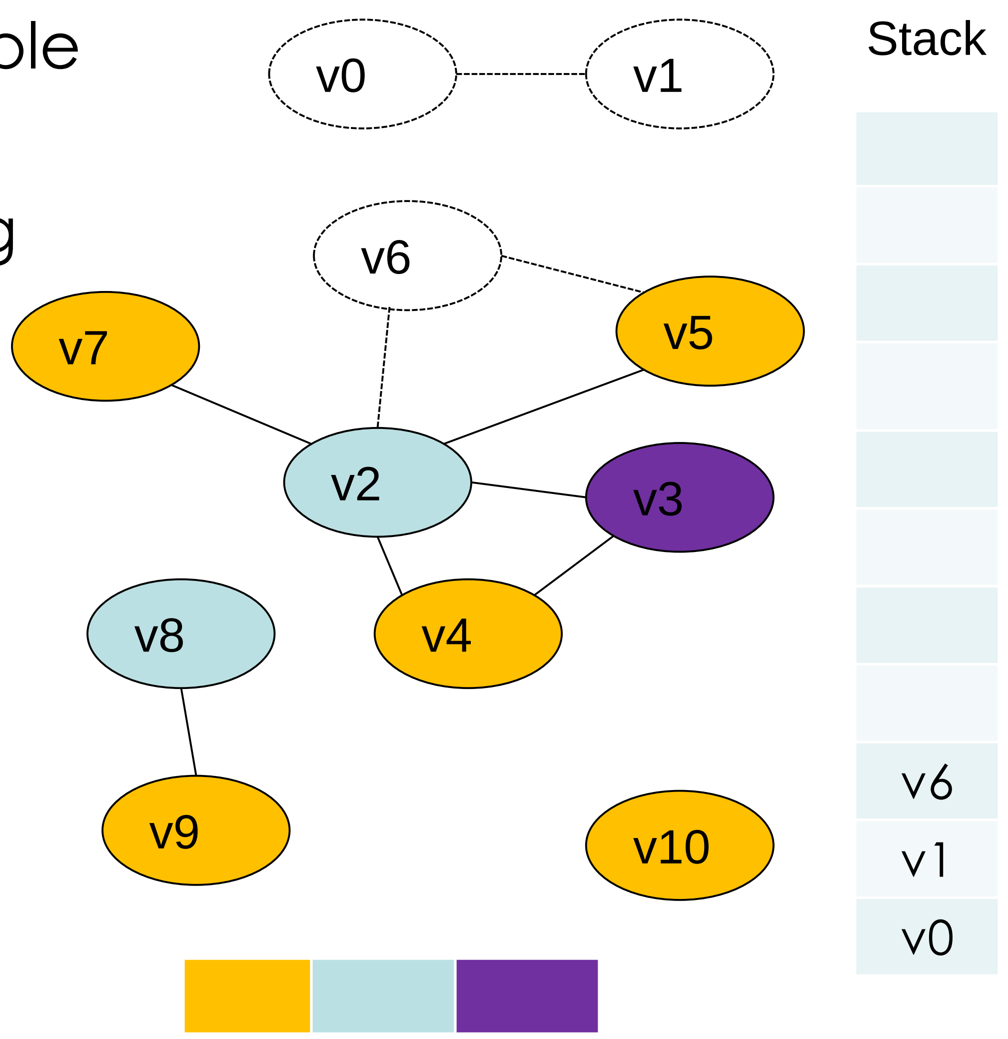
Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- Now we start coloring using the top of the stack
  - v3



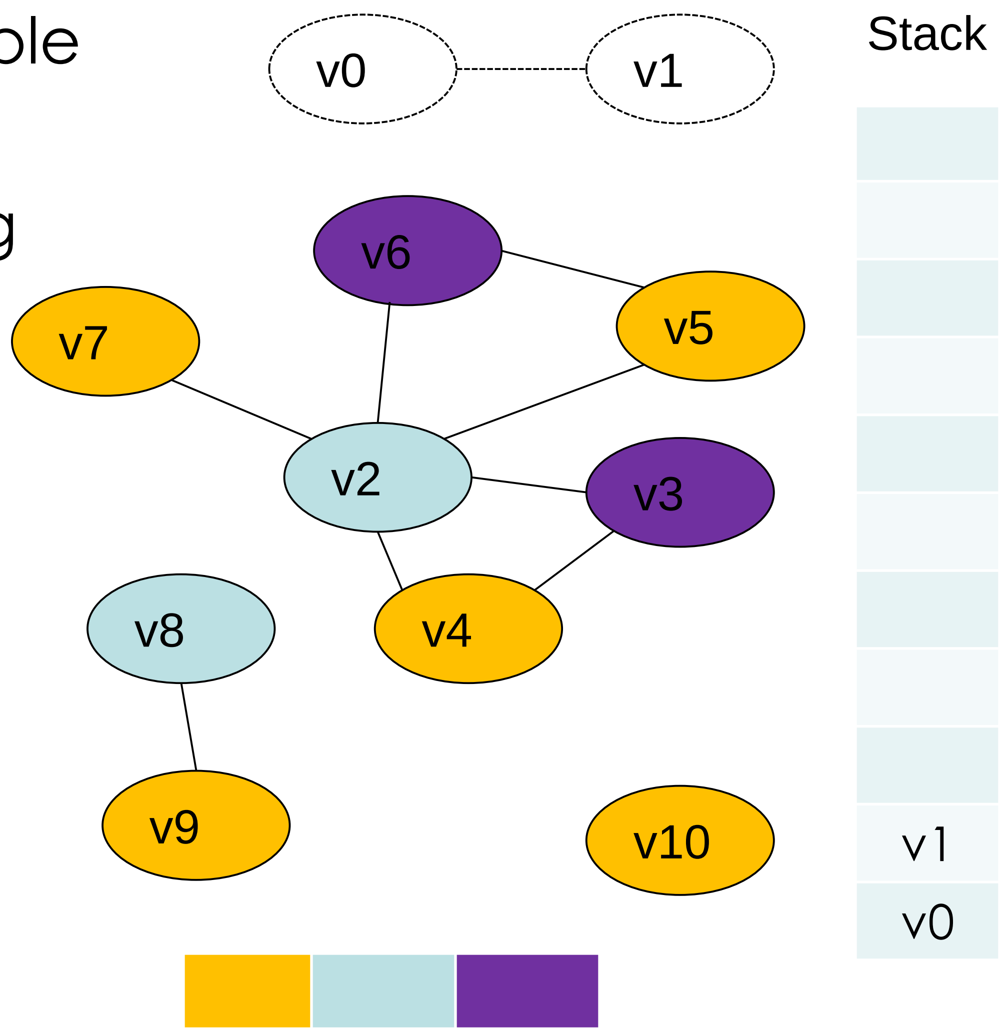
Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- Now we start coloring using the top of the stack
  - v5



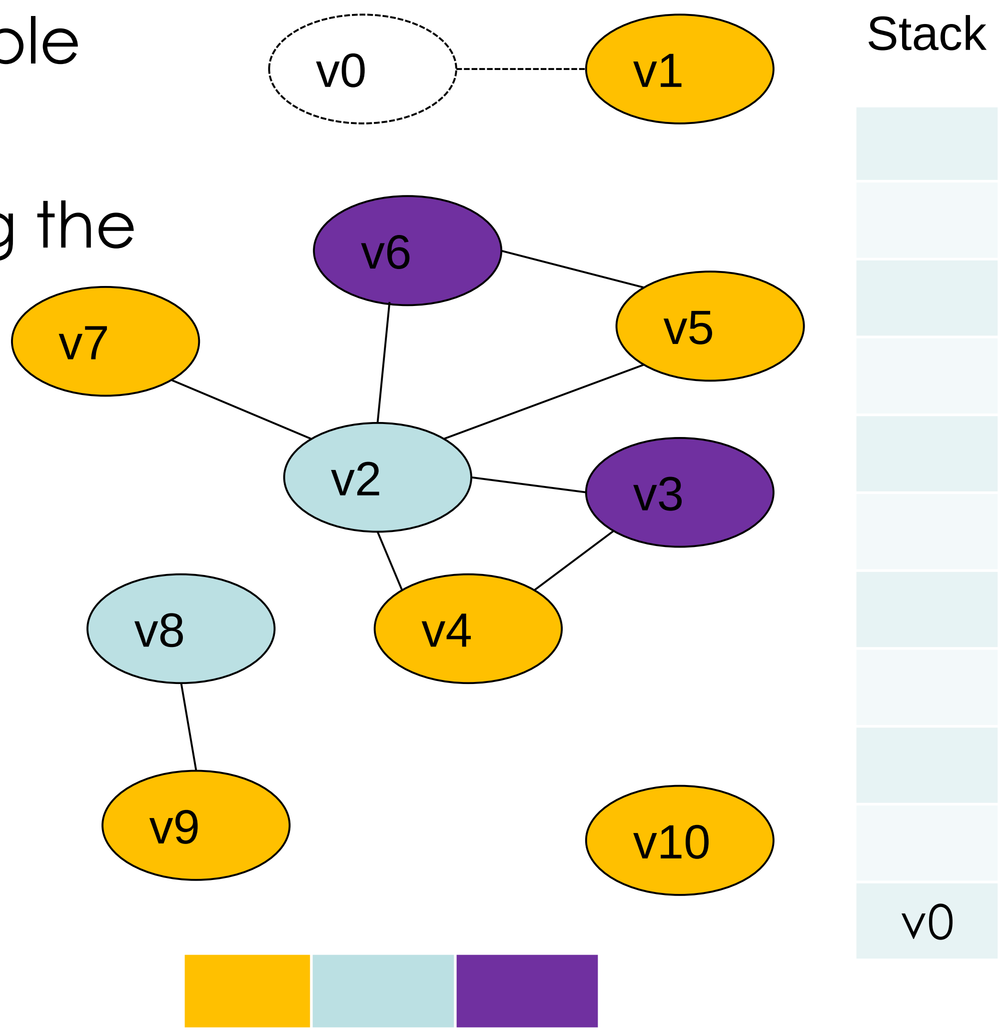
Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- Now we start coloring using the top of the stack
  - v6



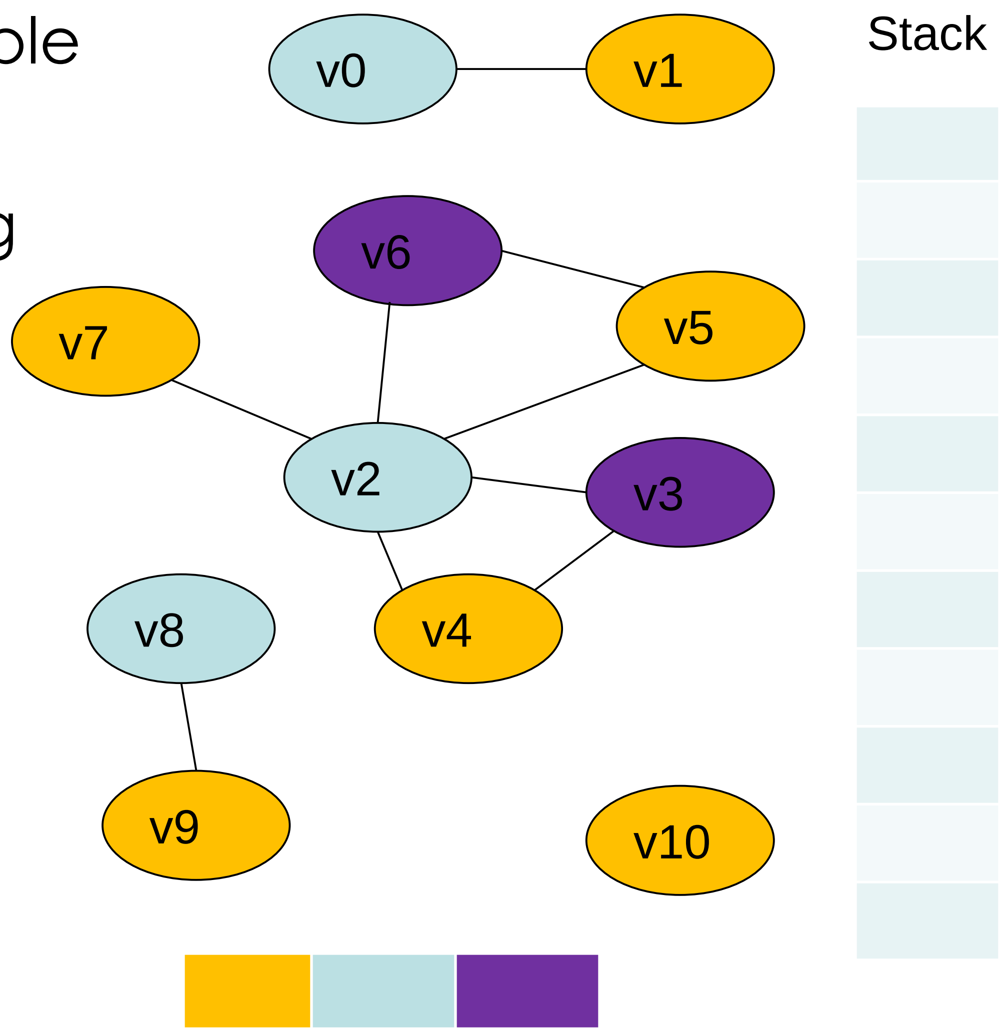
Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- Now we start coloring using the top of the stack
  - v1



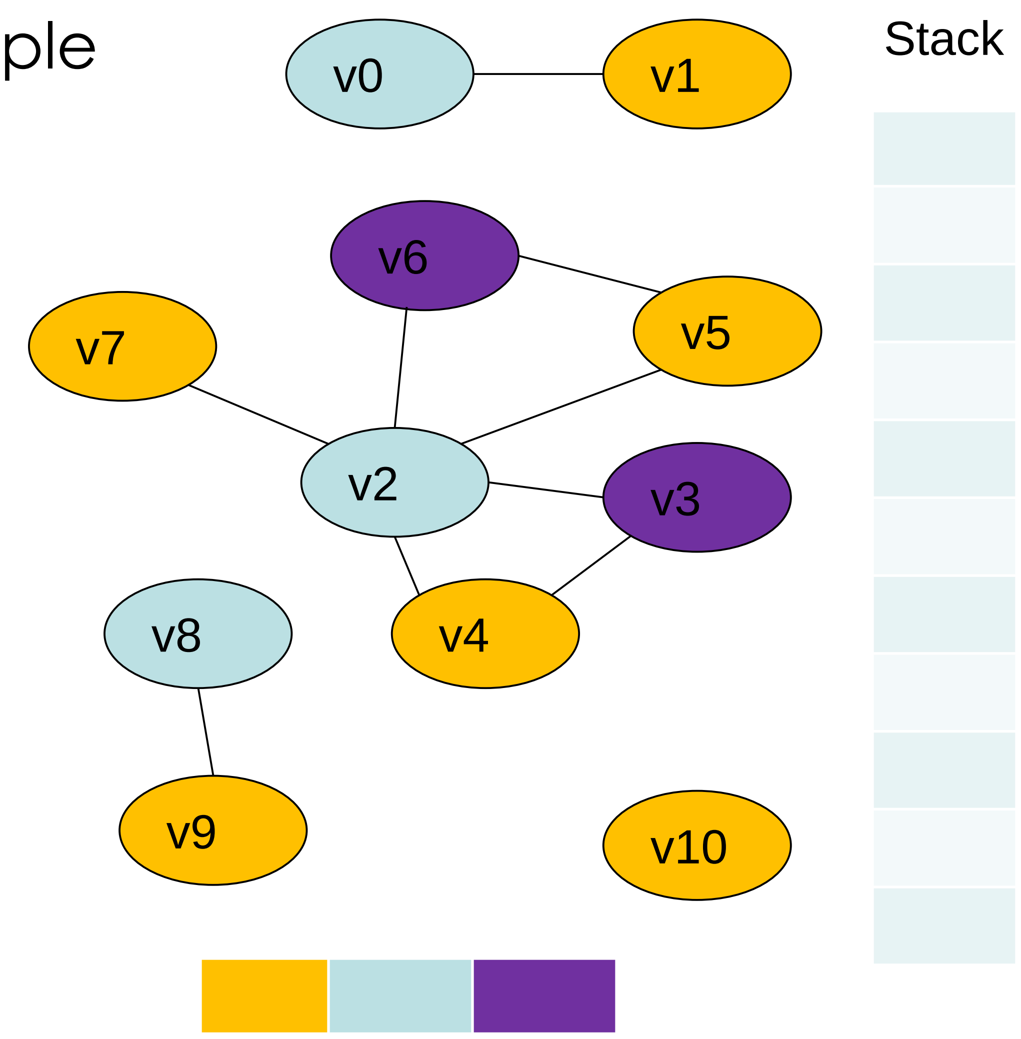
Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- Now we start coloring using the top of the stack
  - v0



Heuristic Solution for Graph Coloring

- Let's go back to the example
- Consider k=3
- Done!
- 3 colors imply 3 registers



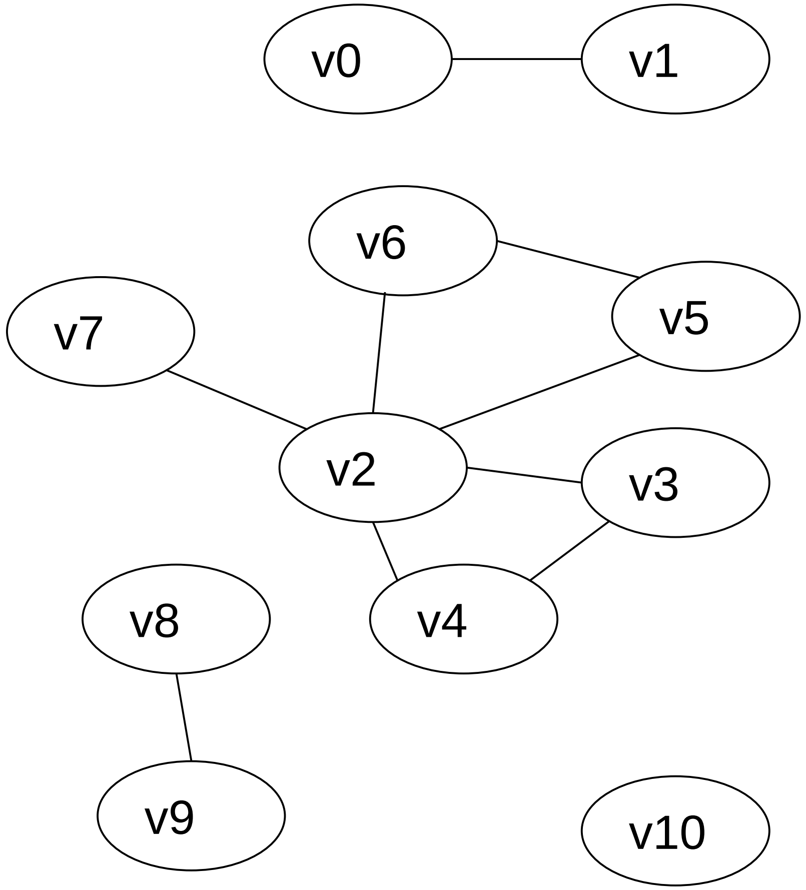
Register Allocation

Question: What to do if a register-interference graph is not k-colorable? Or if the compiler cannot efficiently find a k-coloring even if the graph is k-colorable?

Answer: Repeatedly select less profitable variables for "spilling" (i.e. not to be assigned to registers) and remove them from the interference graph until the graph becomes k-colorable.

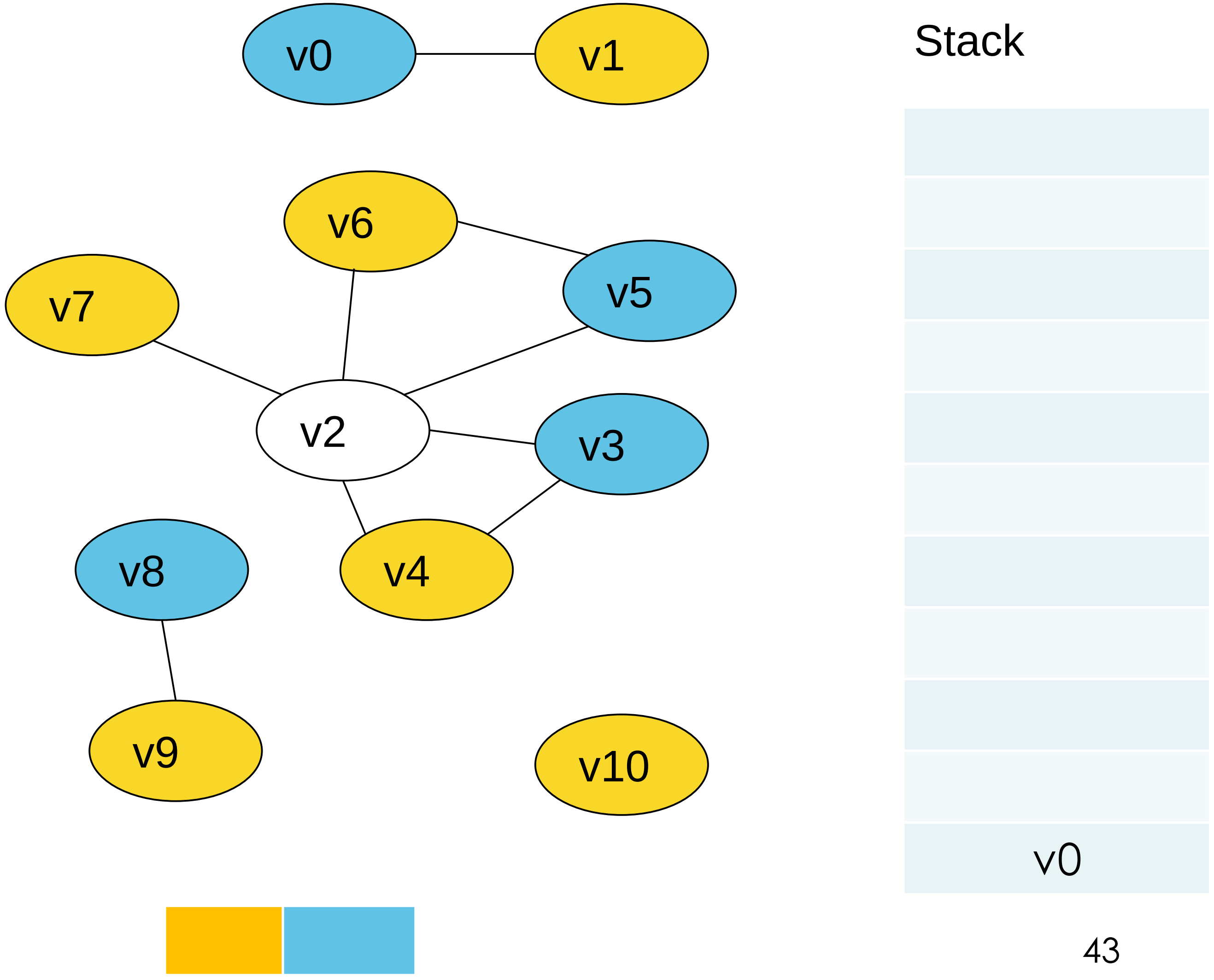
Heuristic Solution for Graph Coloring

- Example:
  - What if we only have 2 registers, i.e., k=2?



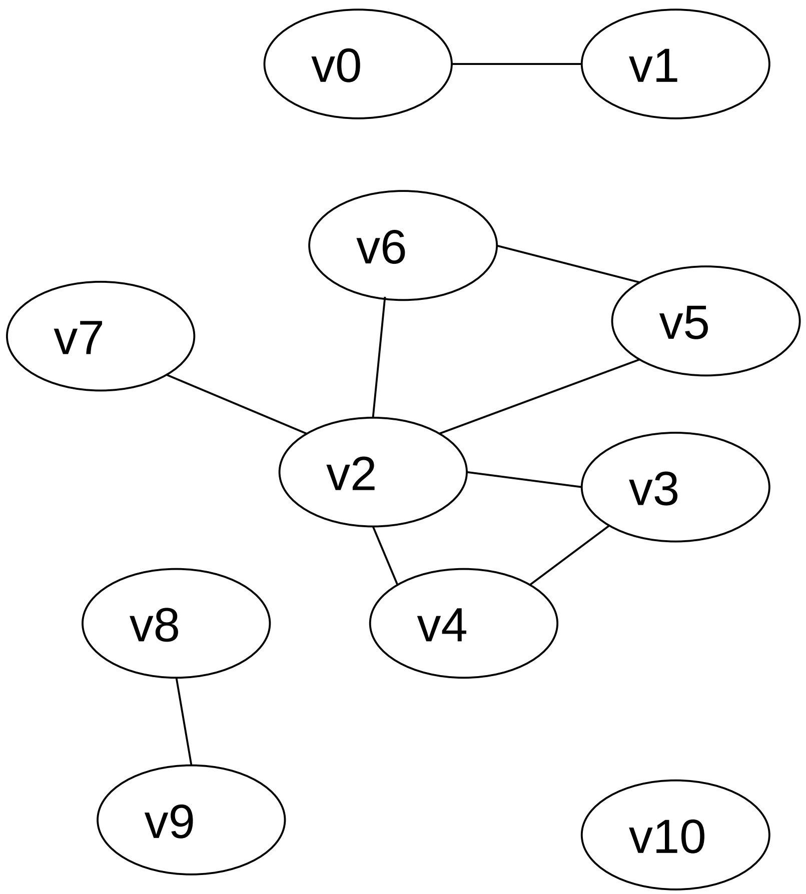
Heuristic Solution for Graph Coloring

- Example:
  - What if we only have 2 registers, i.e., k=2?



Heuristic Solution for Graph Coloring

- Step 3 (spilling): once all nodes have K or more neighbors, pick a node for spilling
  - Storage on the stack
- There are many heuristics that can be used to pick a node
  - E.g., not in an inner loop



Spilling

- We need to generate extra instructions to load variables from stack and store them
- These instructions use registers themselves. What to do?
  - Stupid approach: always keep extra registers handy for shuffling data in and out: what a waste!
  - Better approach: ?

Spilling

- We need to generate extra instructions to load variables from stack and store them
- These instructions use registers themselves. What to do?
  - Stupid approach: always keep extra registers handy for shuffling data in and out: what a waste!
  - Better approach: rewrite code introducing a new temporary; rerun liveness analysis and register allocation

Spilling

- Consider: add t1, t2, t3
  - Suppose t3 is selected for spilling and assigned to stack location [8+$sp]
  - Invented new temporary t35 for just this instruction and rewrite:
    - **lw $t35, 8($sp)**; add t1, t2, t35
  - Advantage: t35 has a very short live range and is much less likely to interfere
  - Rerun the algorithm
    - fewer variables will spill

Spilling

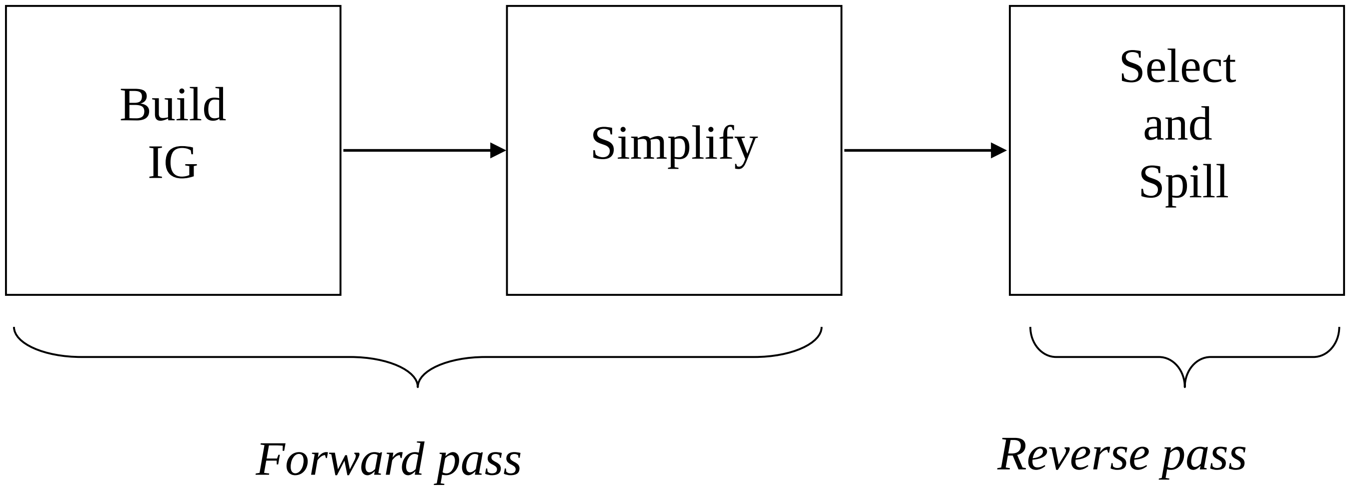
- Variables selected to Spill?
- The selection can be based on a number of properties:
  - frequencies of execution of uses/defs (based on the iteration count, profiling results)
  - number of uses/defs
  - number of adjacent nodes for the variable in the Interference Graph
  - Lifetime duration
  - etc.

Precolored Nodes

- Some variables are pre-assigned to registers
- Treat these registers as special temporaries; before beginning, add them to the graph with their colors
- Can’t simplify a graph by removing a precolored node
- Precolored nodes are the starting point of the coloring process
- Once simplified down to colored nodes start adding back the other nodes as before

Heuristic Solution for Graph Coloring

- A 2-Phase Register Allocation Algorithm



Remarks

- This register allocation algorithm, based on graph coloring, is both efficient (linear time) and effective (good assignment)

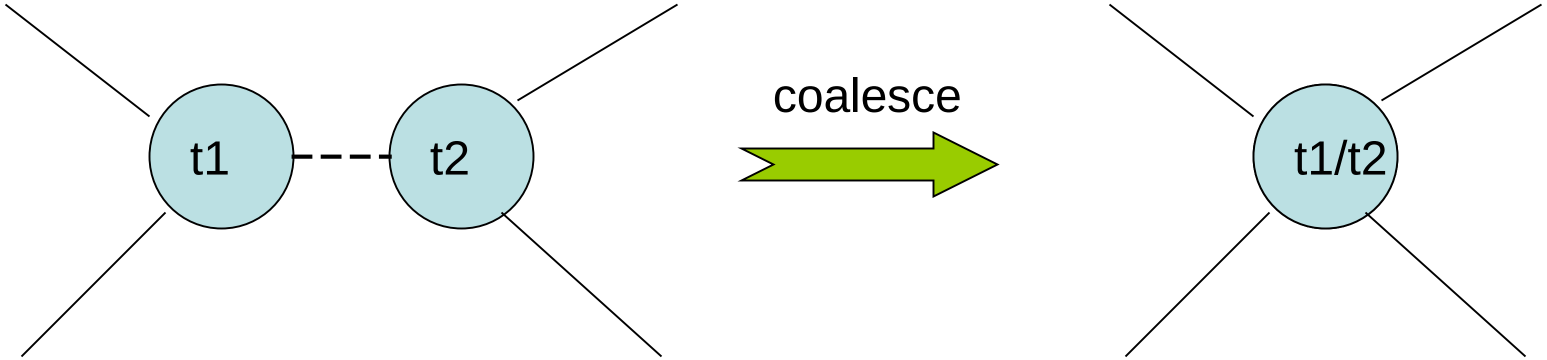
- It has been used in many industry-strength compilers to obtain significant improvements over simpler register allocation heuristics

Optimizing Moves

- Code generation produces a lot of extra move instructions
  - mov t1, t2 ($t1 \leftarrow t2$)
  - If we can assign t1 and t2 to the same register, we do not have to execute the mov
  - Idea: if t1 and t2 are not connected in the interference graph, we coalesce into a single variable
  - First: Include in the register interference graph a move-related edge between two variables used in a move instruction

Coalescing

- Problem: coalescing can increase the number of interference edges and make a graph uncolorable



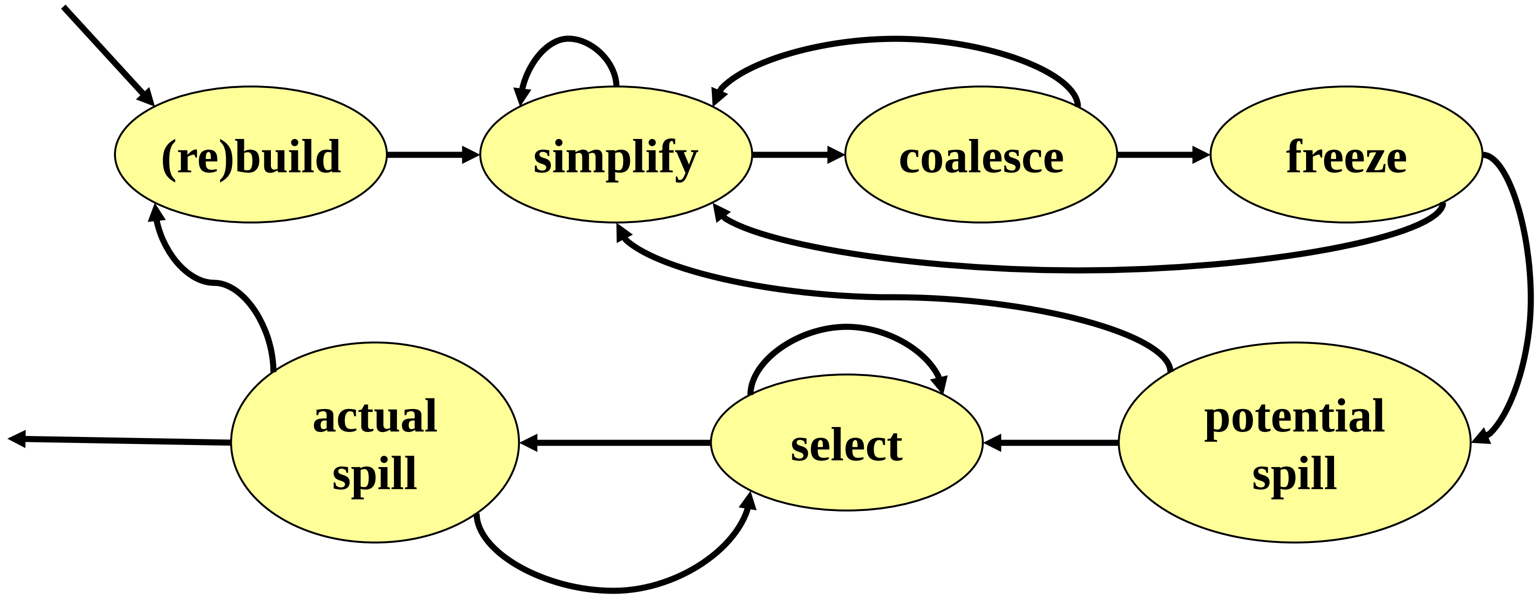
- Solution 1 (Briggs): avoid creation of high-degree (>= K) nodes
- Solution 2 (George): a can be coalesced with b if every neighbor t of a:
  - already interferes with b, or
  - has low-degree (< K)

Simplify and Coalesce

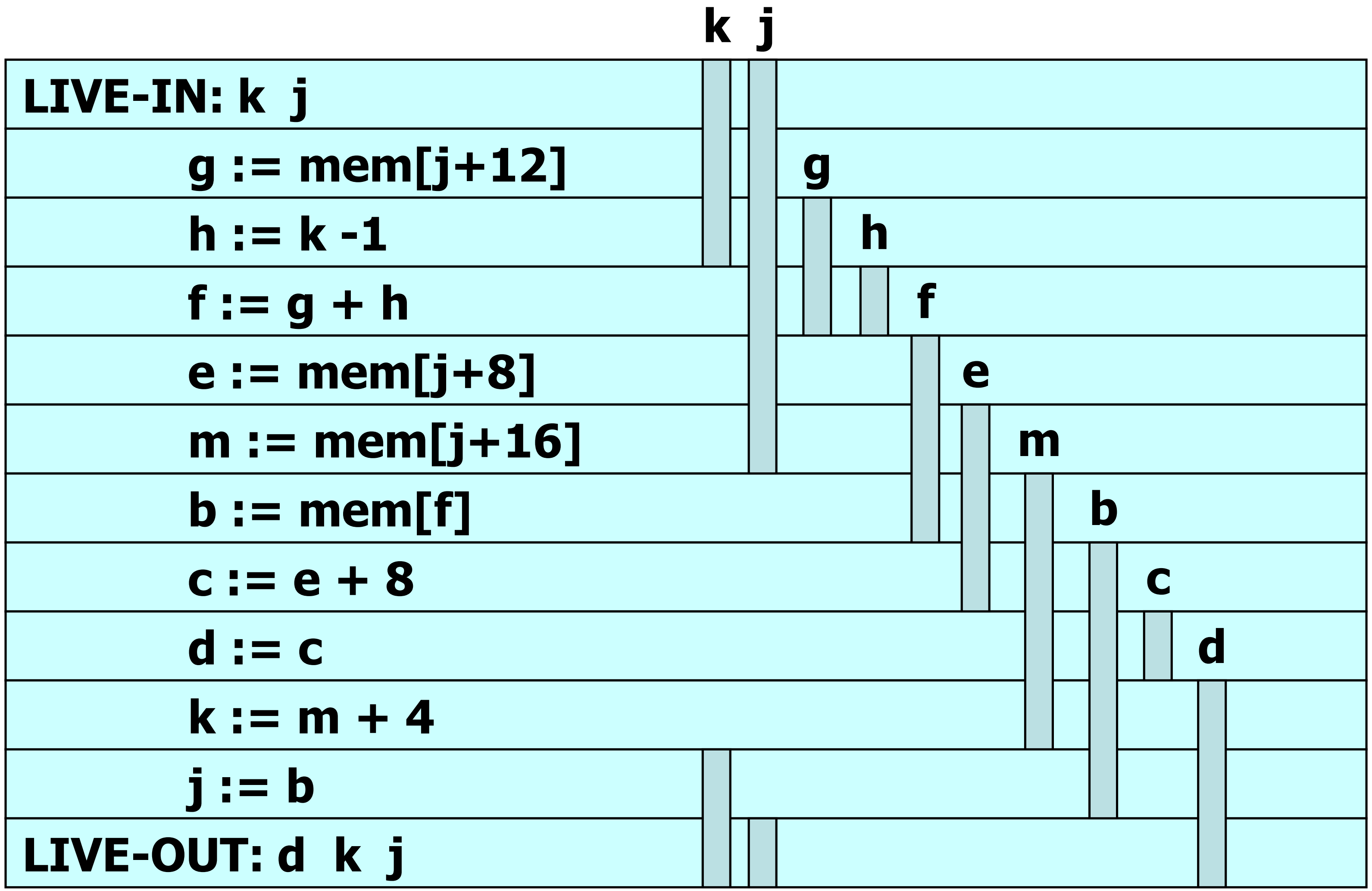
- Step 1 (simplify): simplify as much as possible without removing nodes that are the source or destination of a move (move-related nodes)
- Step 2 (coalesce): coalesce move-related nodes provided low-degree node results
- Step 3 (freeze): if neither steps 1 or 2 apply, freeze a move instruction: registers involved are marked not move-related and try step 1 again
- Step 4 (spill): if there are no low-degree nodes, select a node for potential spilling
- Step 5 (select): pop each element of the stack assigning colors and turning potential spill into actual spill if needed
- Step 6 (rewrite the program): rewrite the program based on the register allocation, remove move operations with coalesced variables, and inserting spilling code. If there is spill build a new register-inference graph and goto Step 1

Overall Algorithm

- From Tiger Book (by Appel)

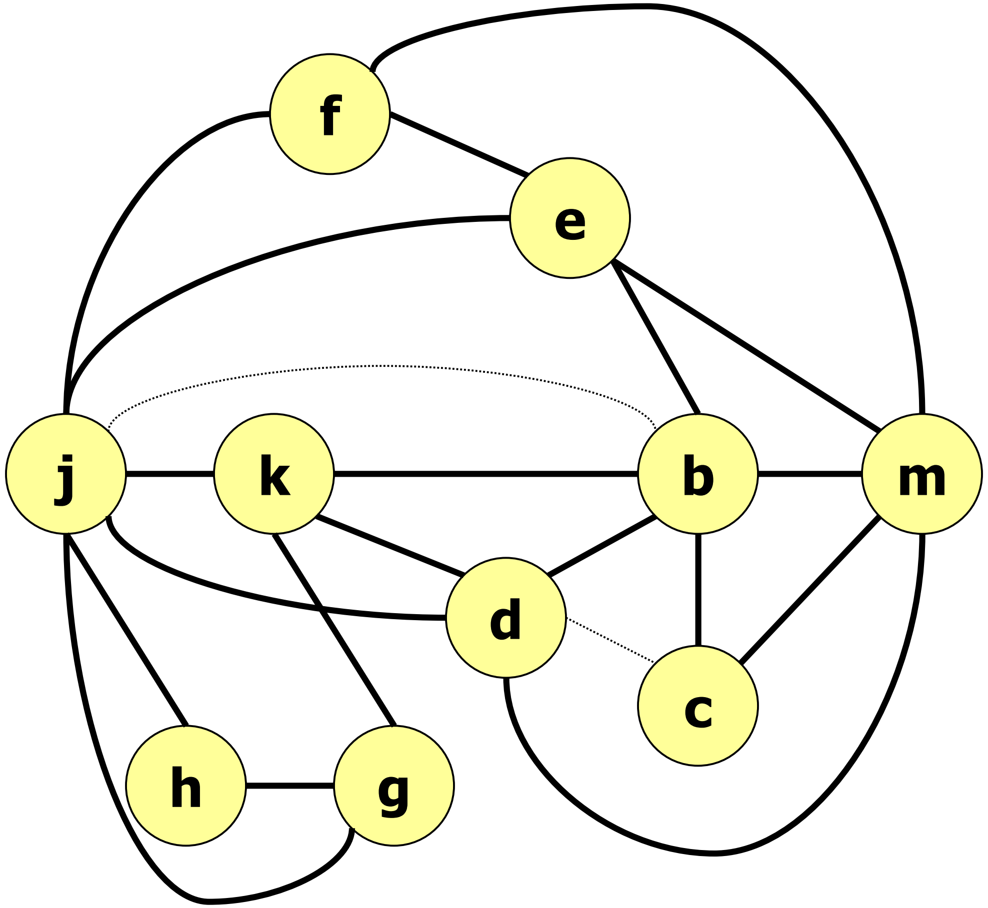


Example:
Step 1: Compute Live Ranges



(José Nelson Amaral based on Tiger Book, Appel)

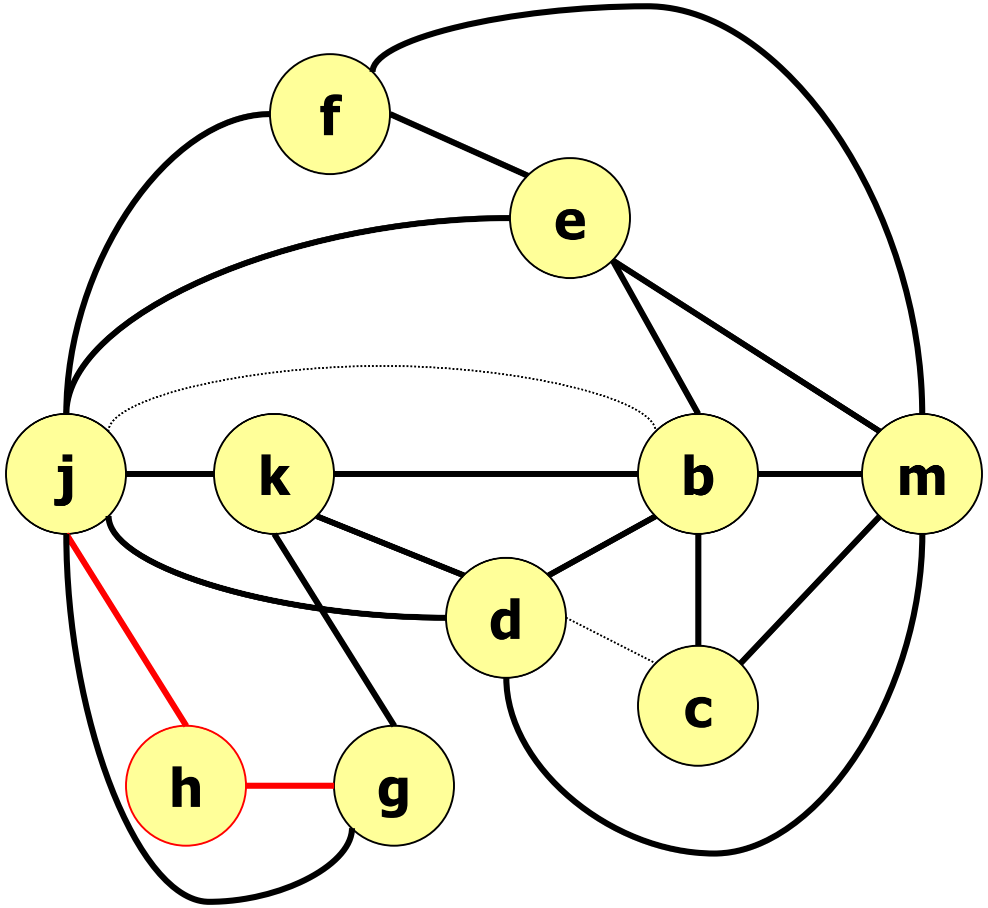
Example:
Step 3: Simplify (K=4)



stack

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Simplify (K=4)

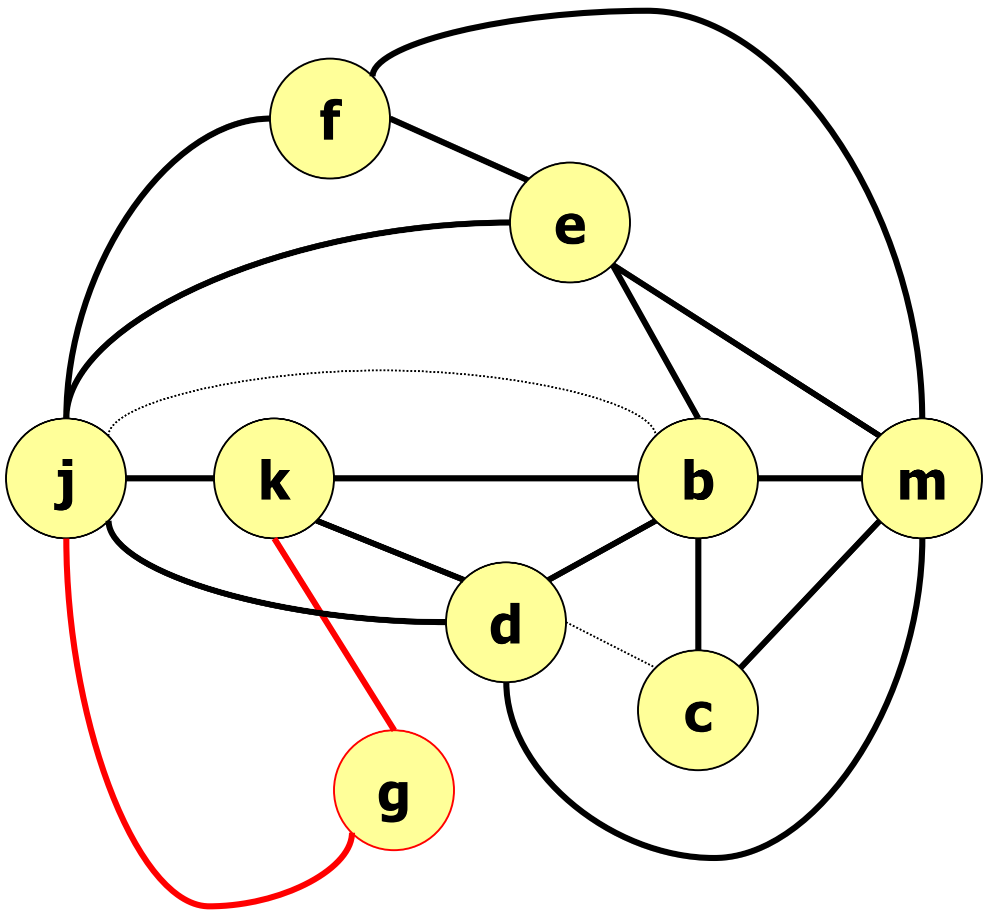


stack

(h,no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Simplify (K=4)

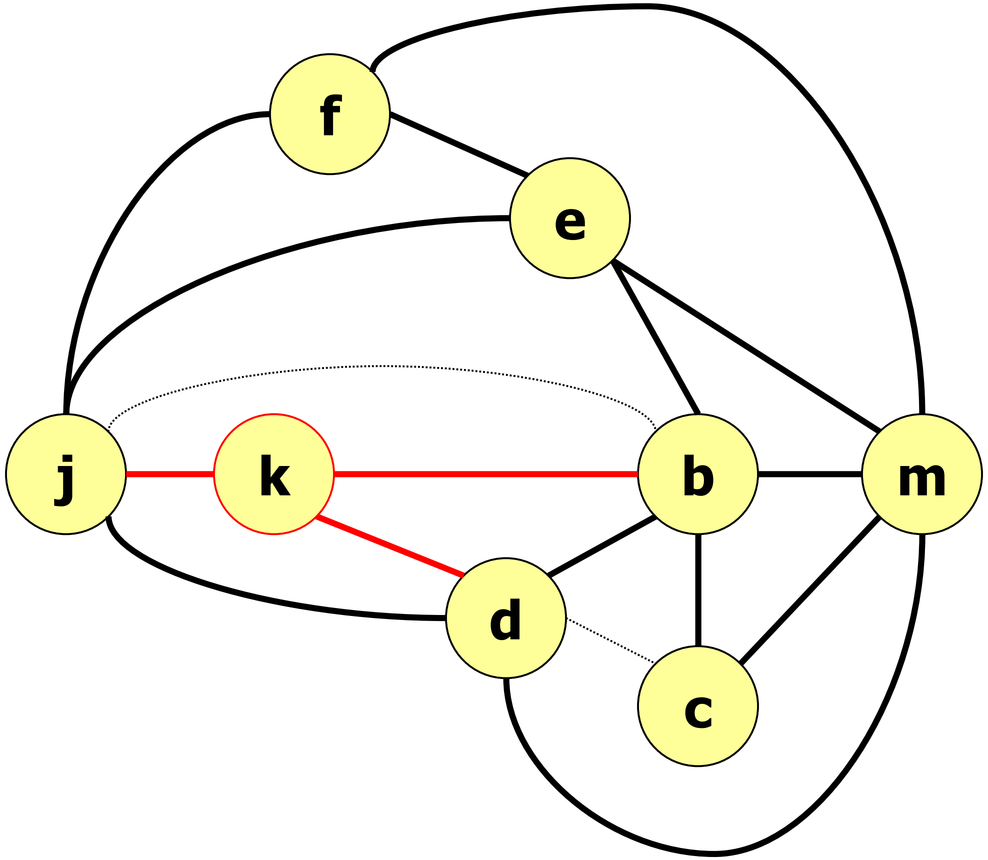


stack

(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Simplify (K=4)

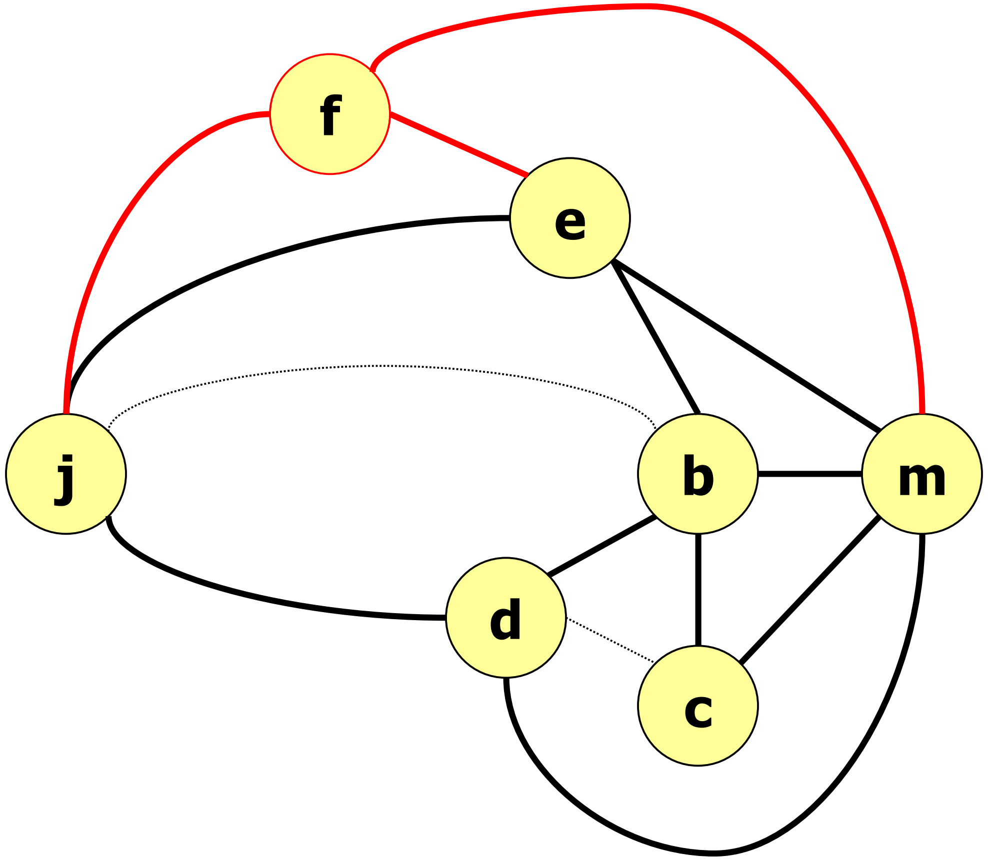


stack

(k, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Simplify (K=4)

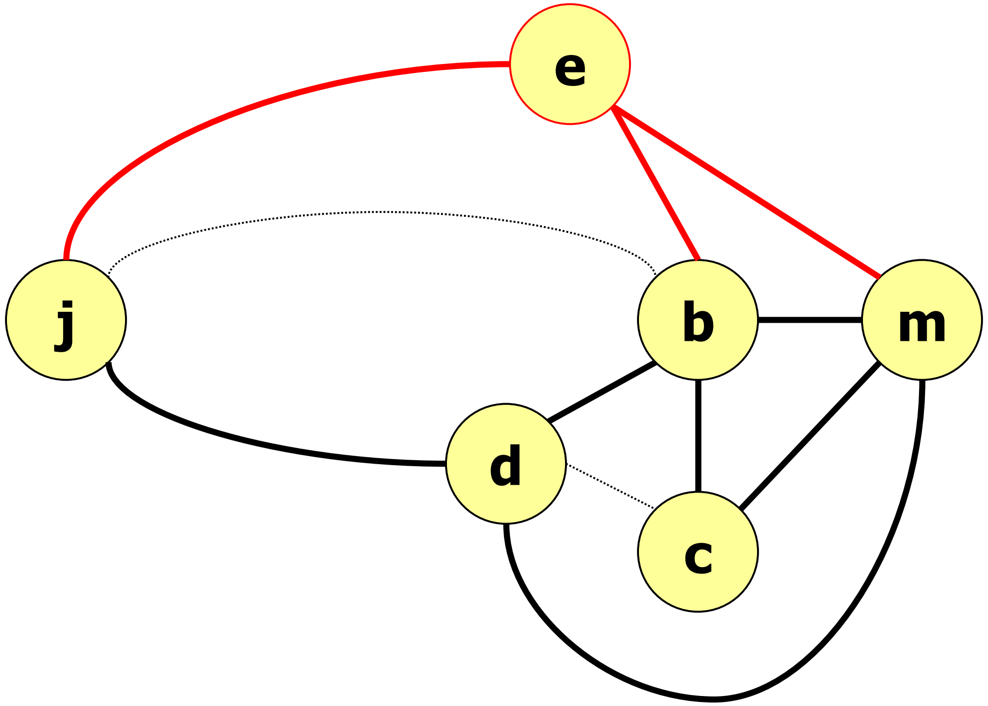


stack

(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Simplify (K=4)

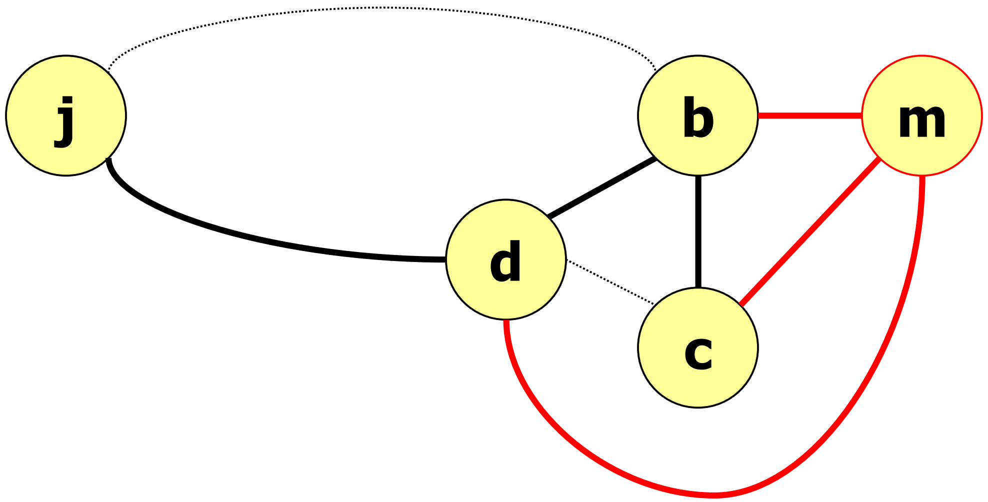


stack

(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Simplify (K=4)

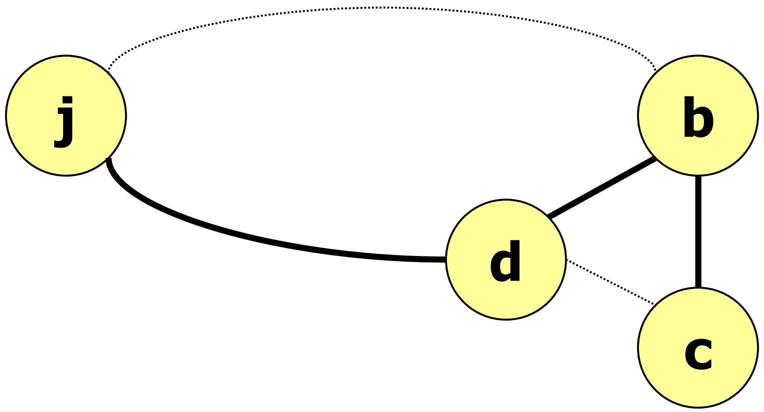


stack

(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Coalesce (K=4)



stack

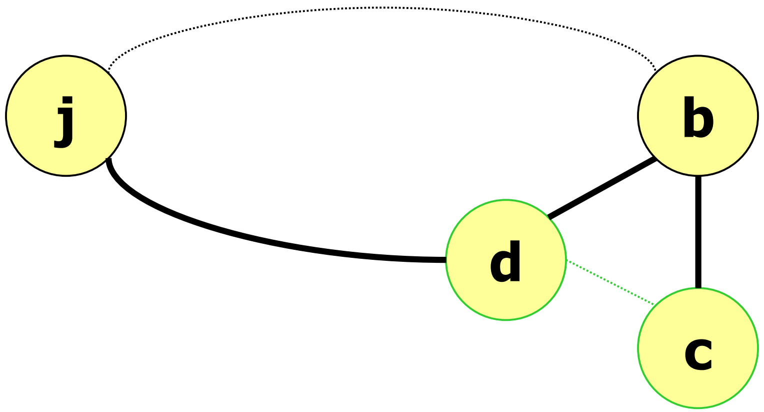
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

Why we cannot simplify?

Cannot simplify move-related nodes.

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Coalesce (K=4)

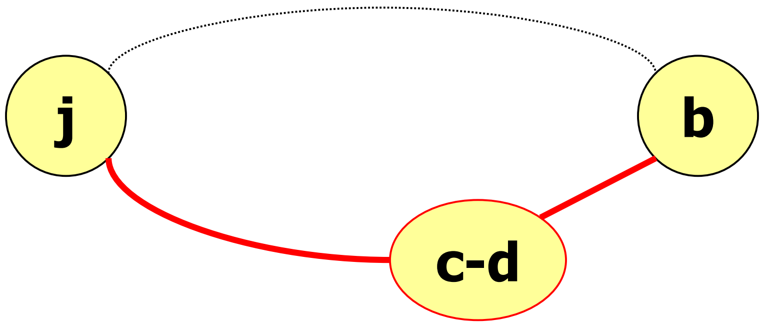


stack

(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Simplify (K=4)



stack

(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

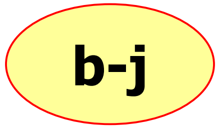
Example:
Step 3: Coalesce (K=4)

stack

(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Simplify (K=4)

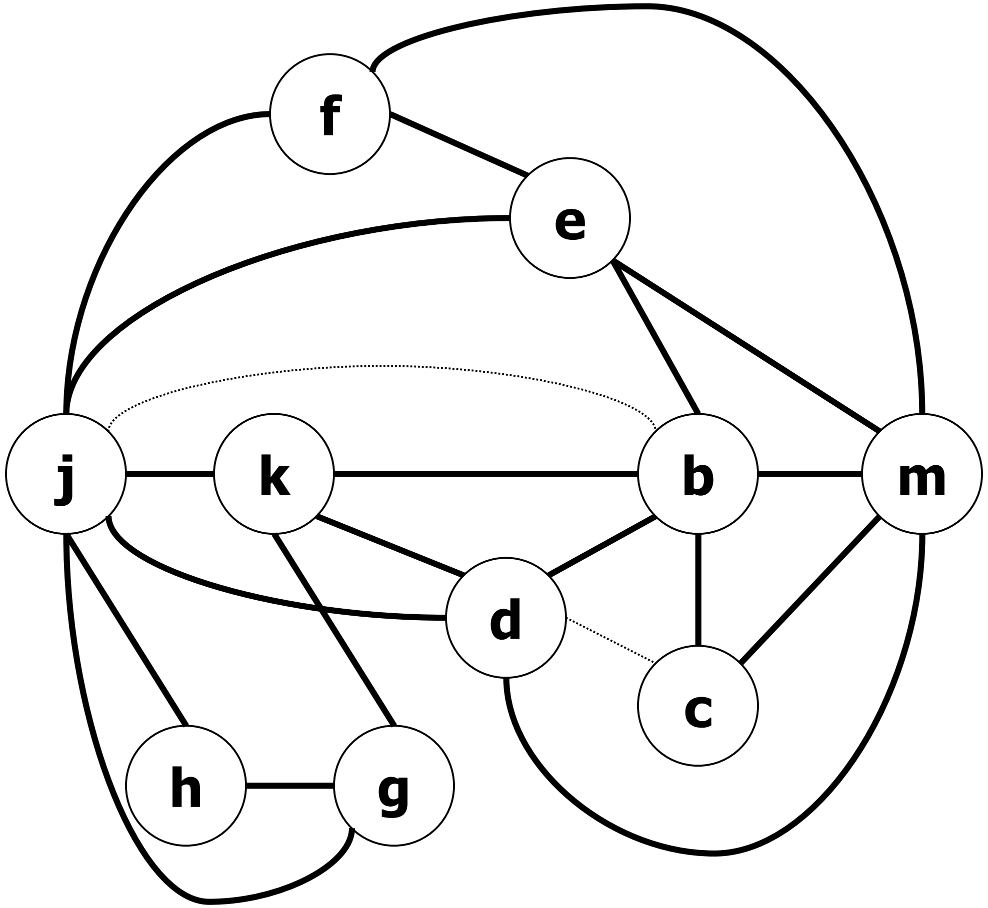


stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=4)



stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1

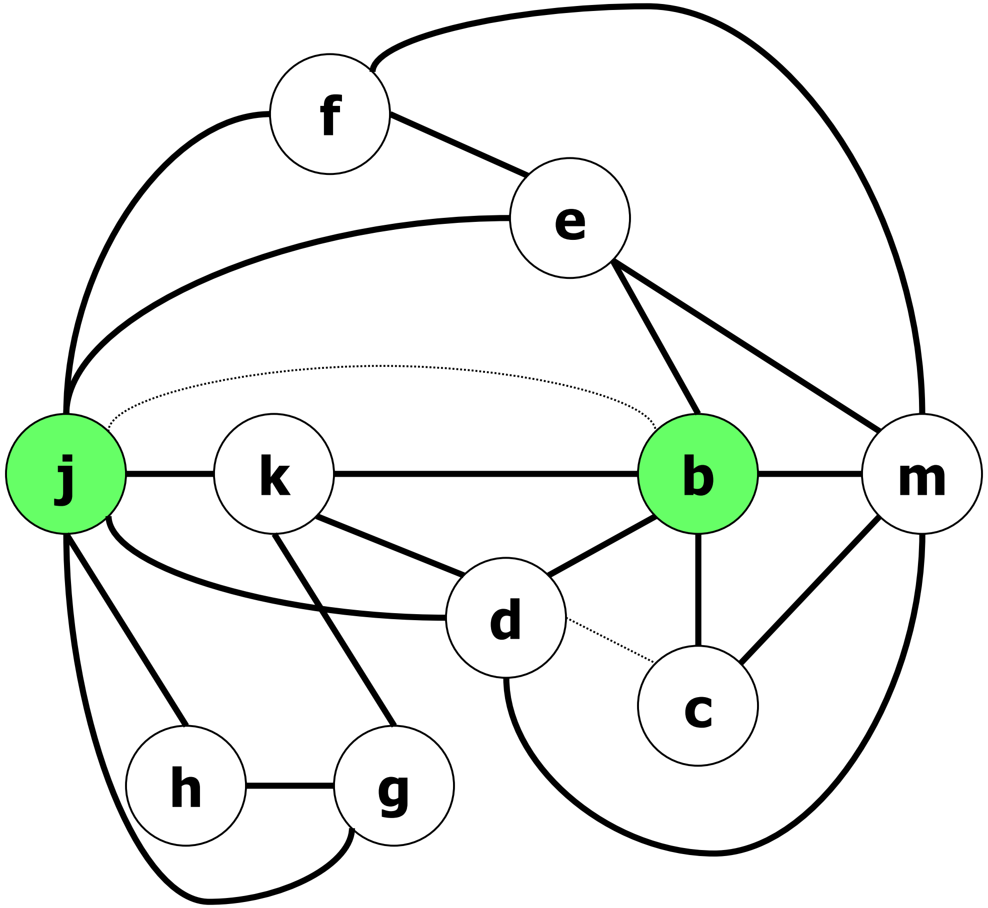
R2

R3

R4

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=4)



stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1

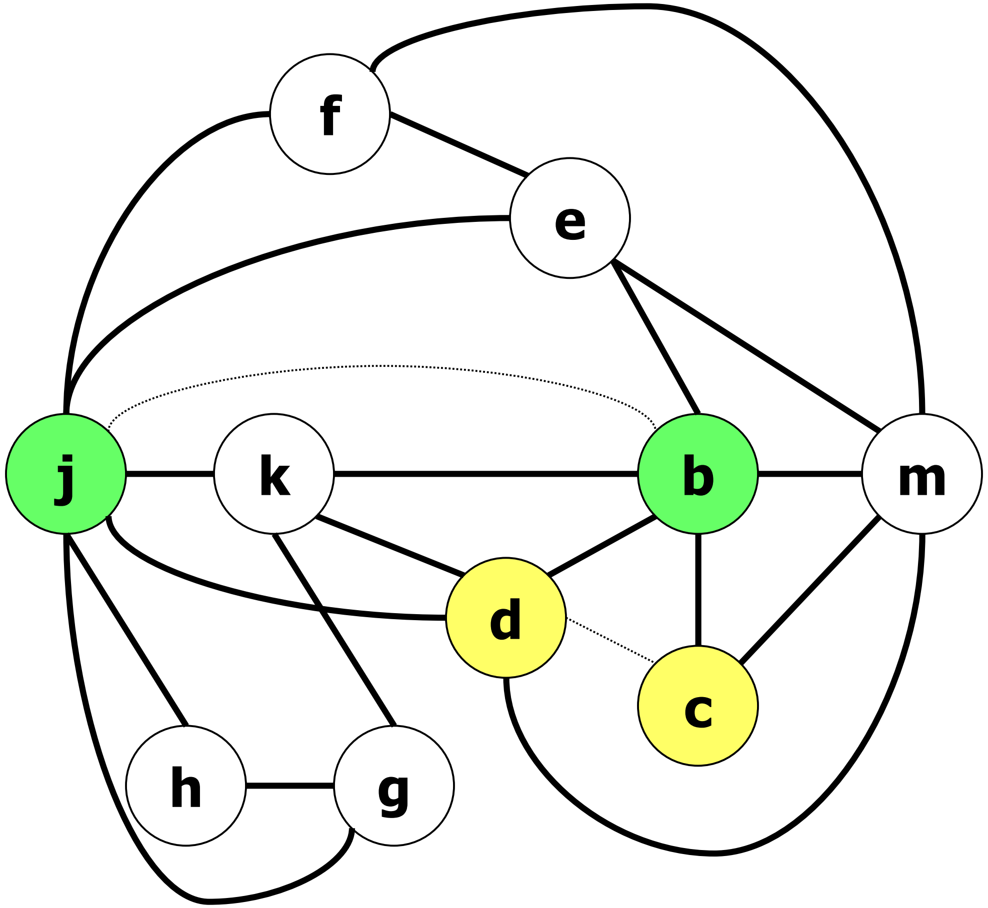
R2

R3

R4

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=4)



stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1

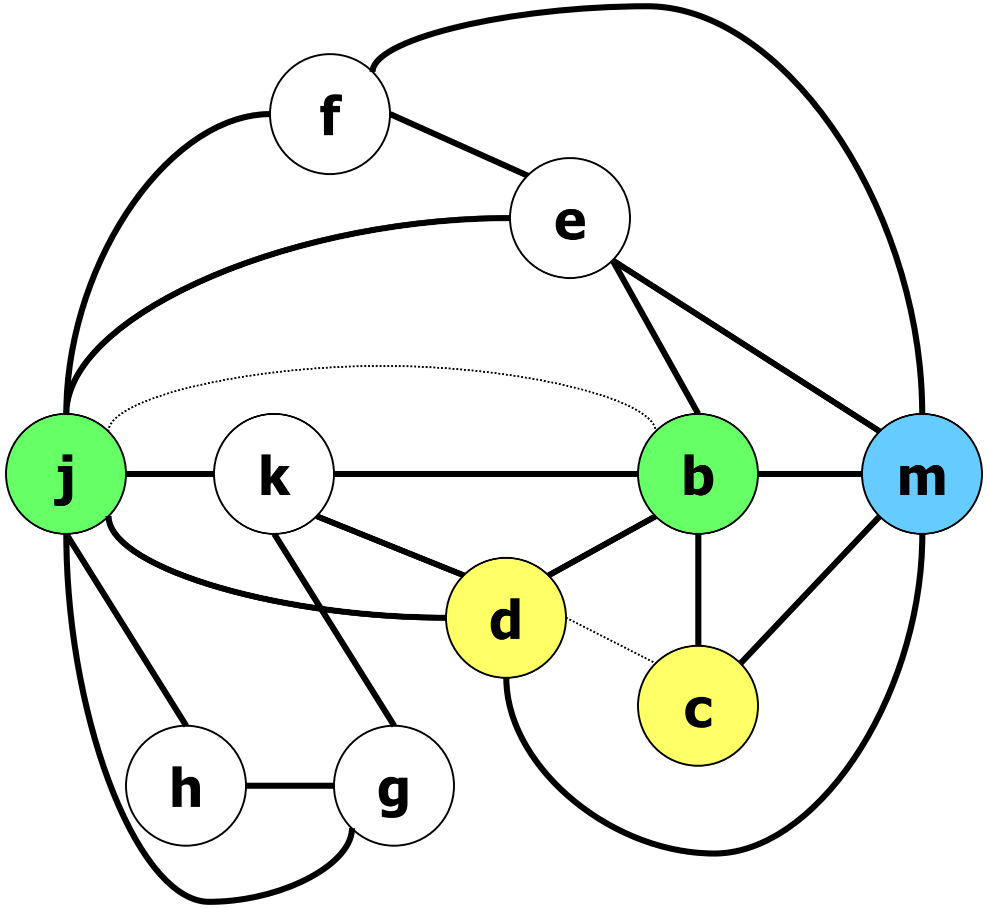
R2

R3

R4

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=4)



stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1

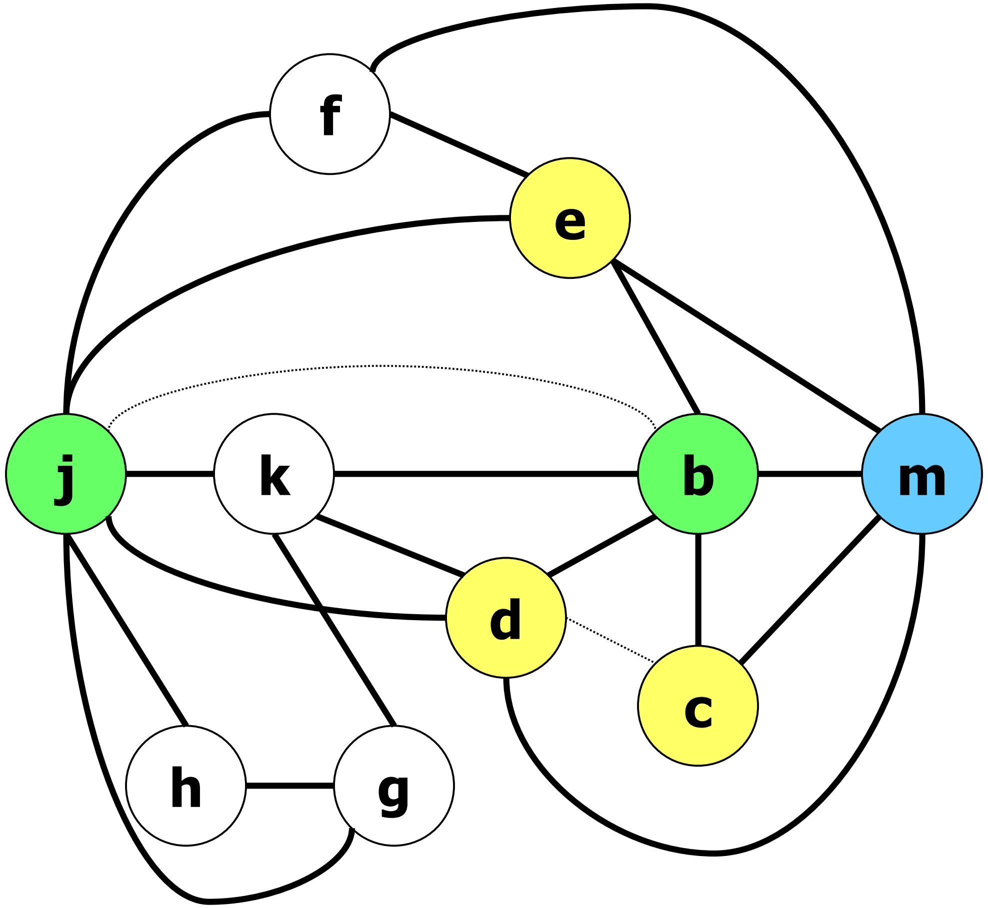
R2

R3

R4

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=4)



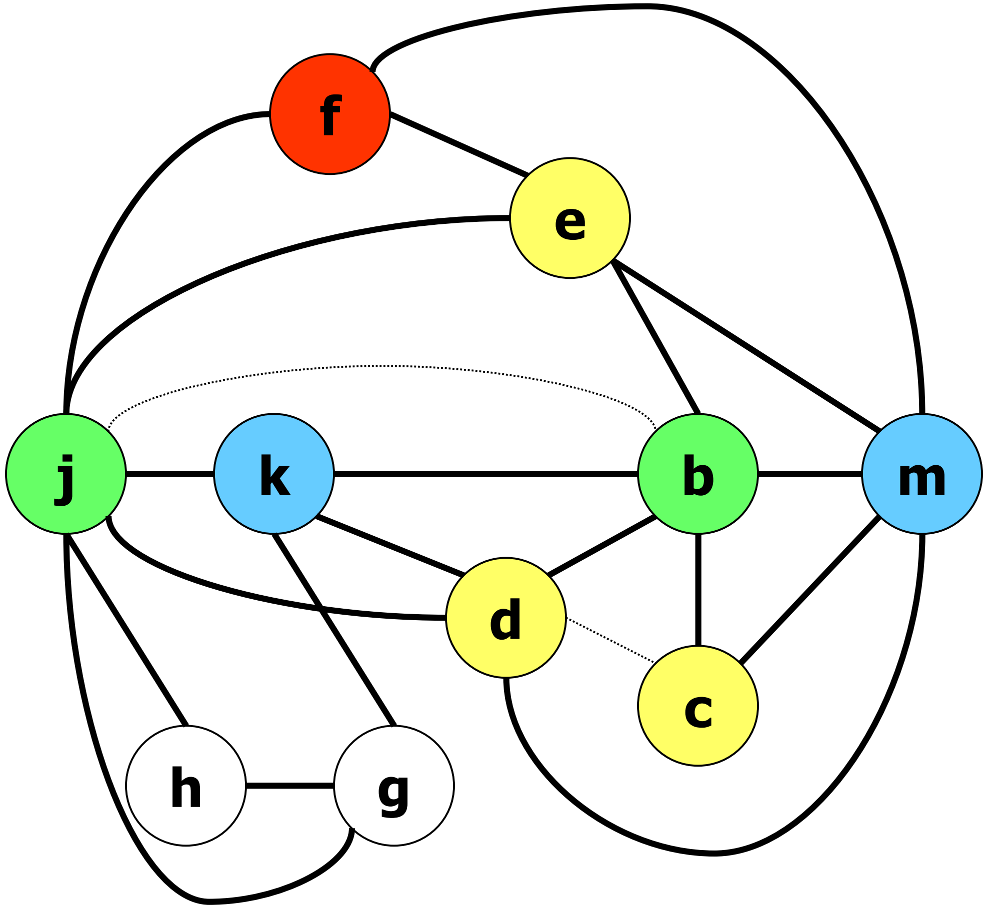
stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1
R2
R3
R4

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=4)



stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1

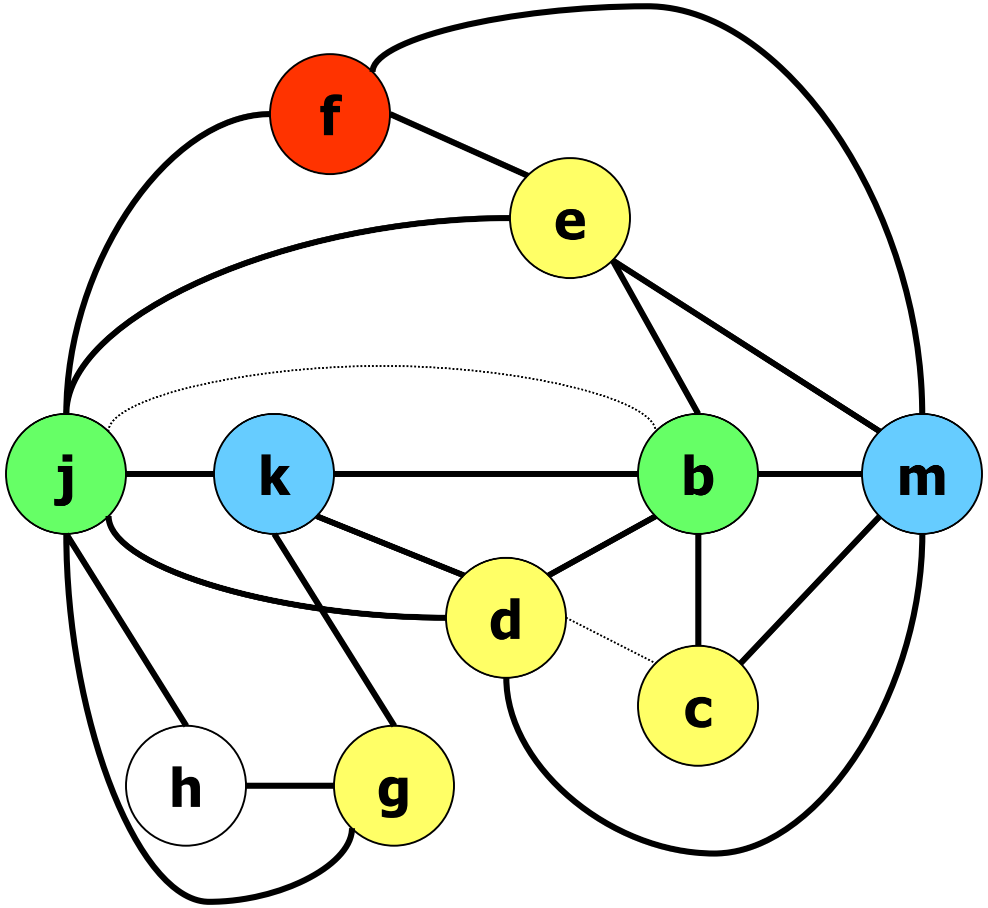
R2

R3

R4

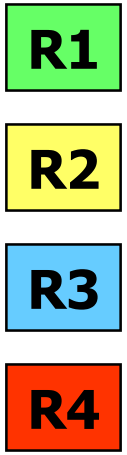
(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=4)



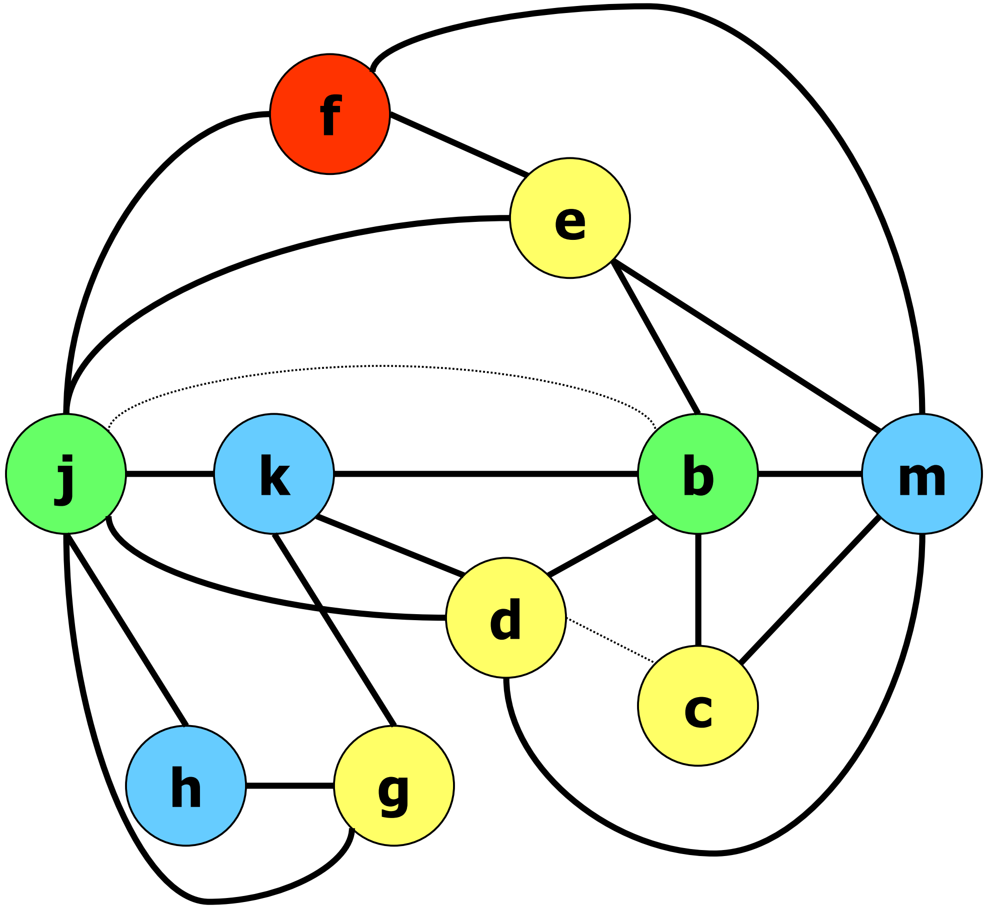
stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)



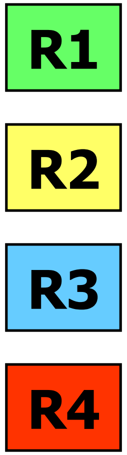
(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=4)



stack

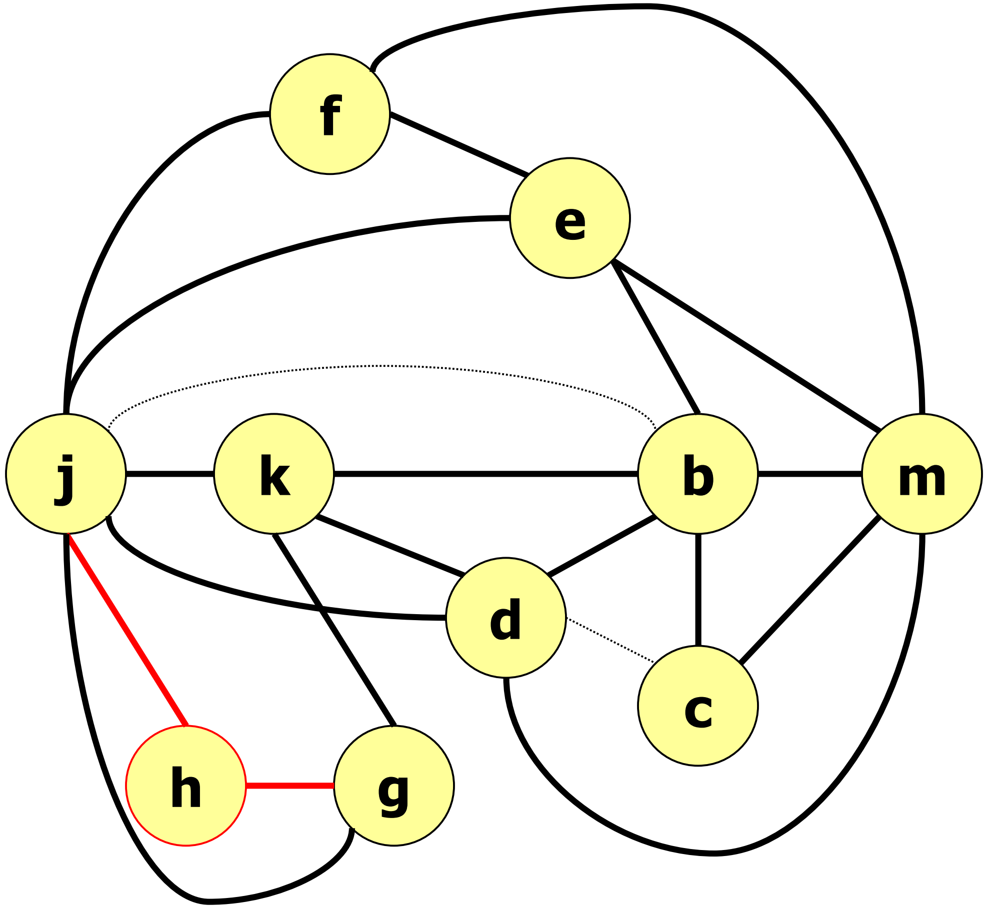
(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)



(José Nelson Amaral based on Tiger Book, Appel)

Could we do the allocation in the previous example with 3 registers?

Example:
Step 3: Simplify (K=3)

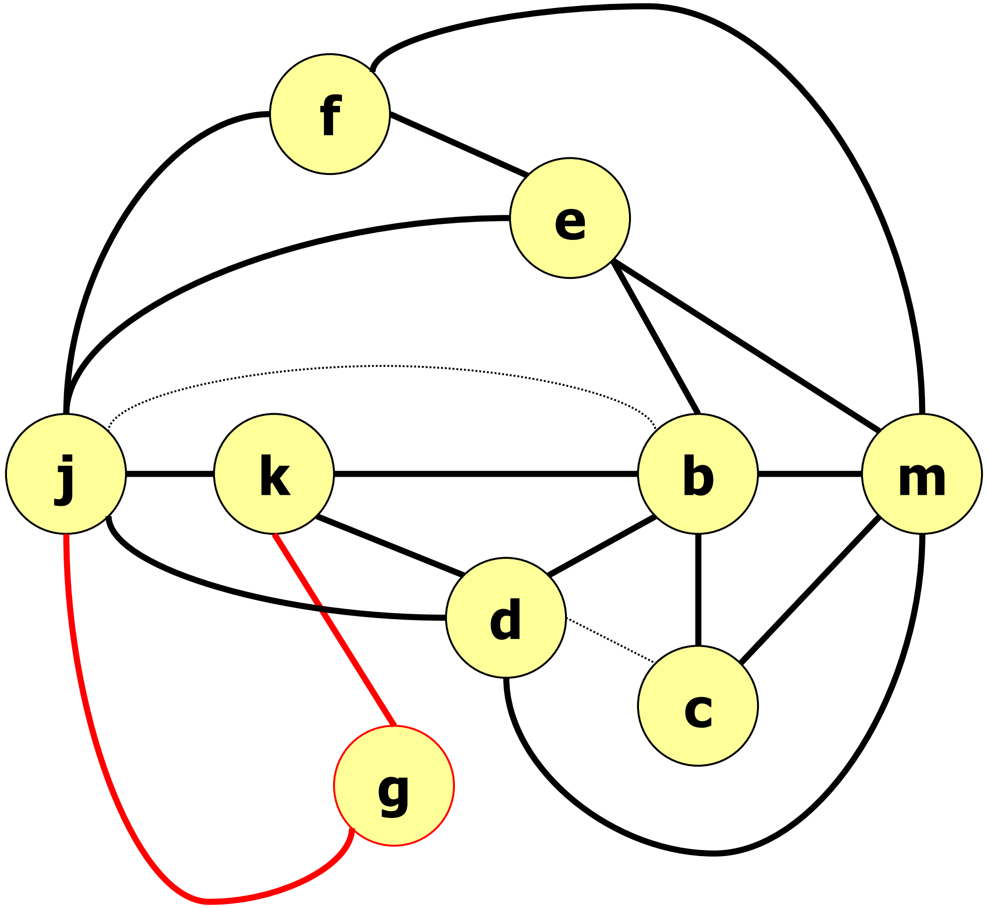


stack

(h,no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Simplify (K=3)

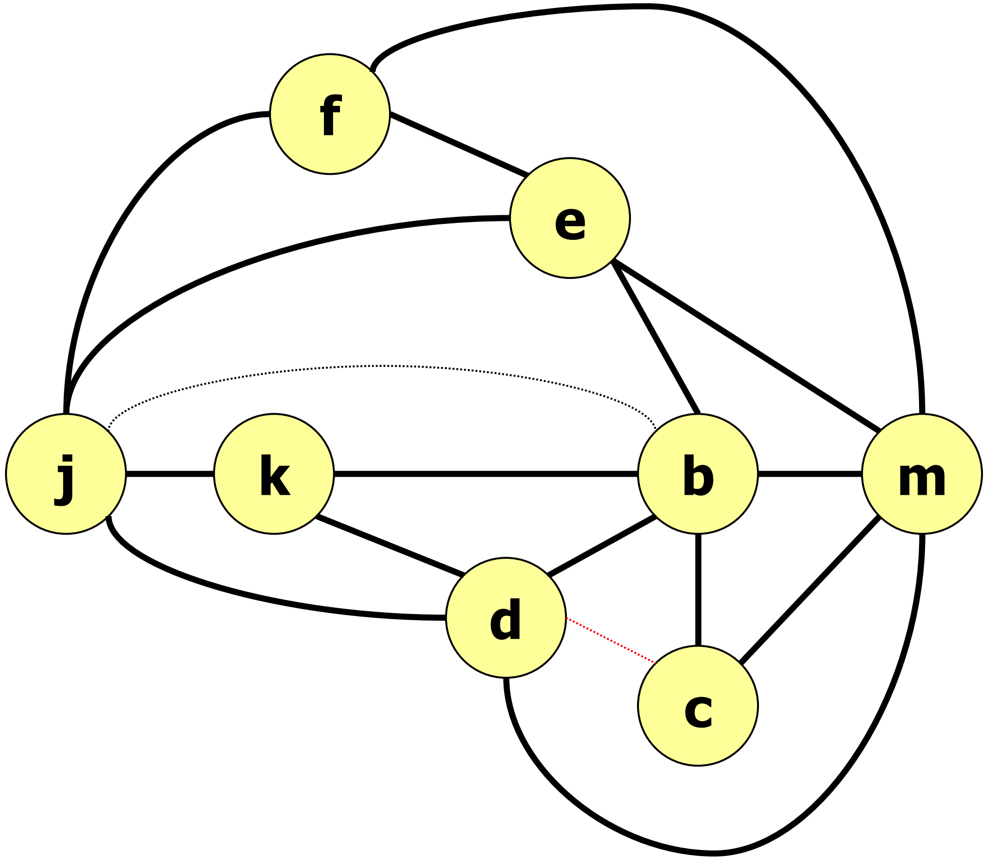


stack

(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 5: Freeze (K=3)



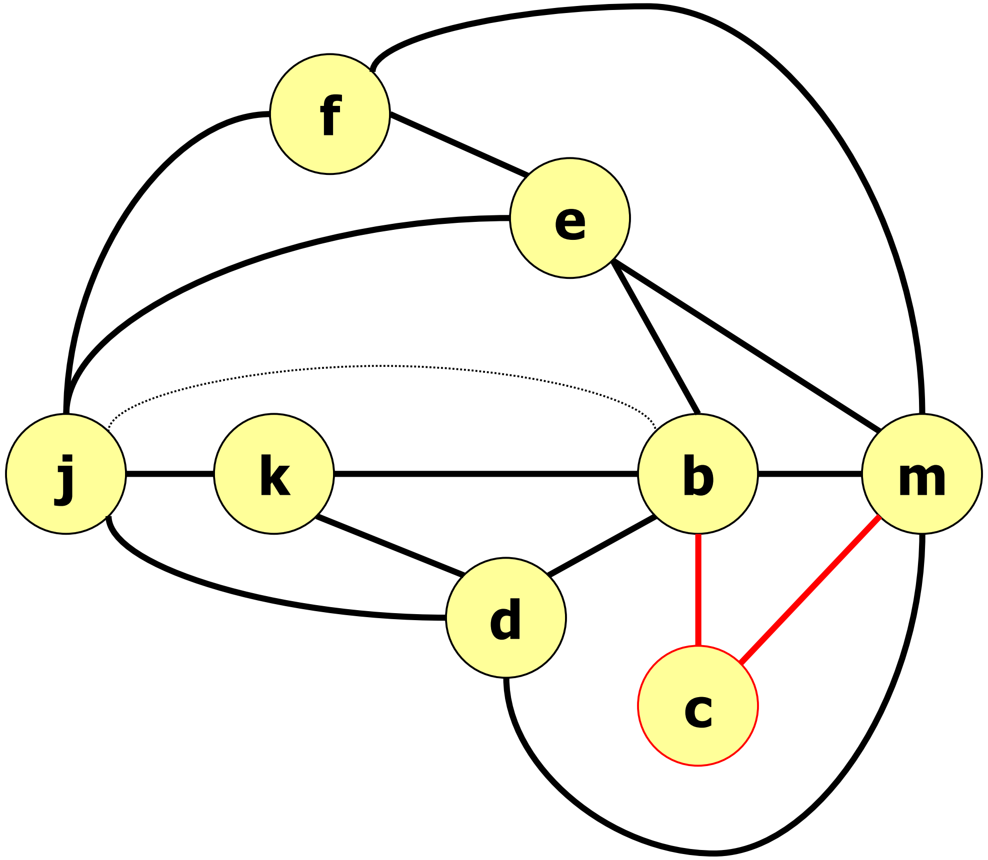
stack

(g, no-spill)
(h, no-spill)

Coalescing would make things worse.
We can freeze the move d-c.

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Simplify (K=3)

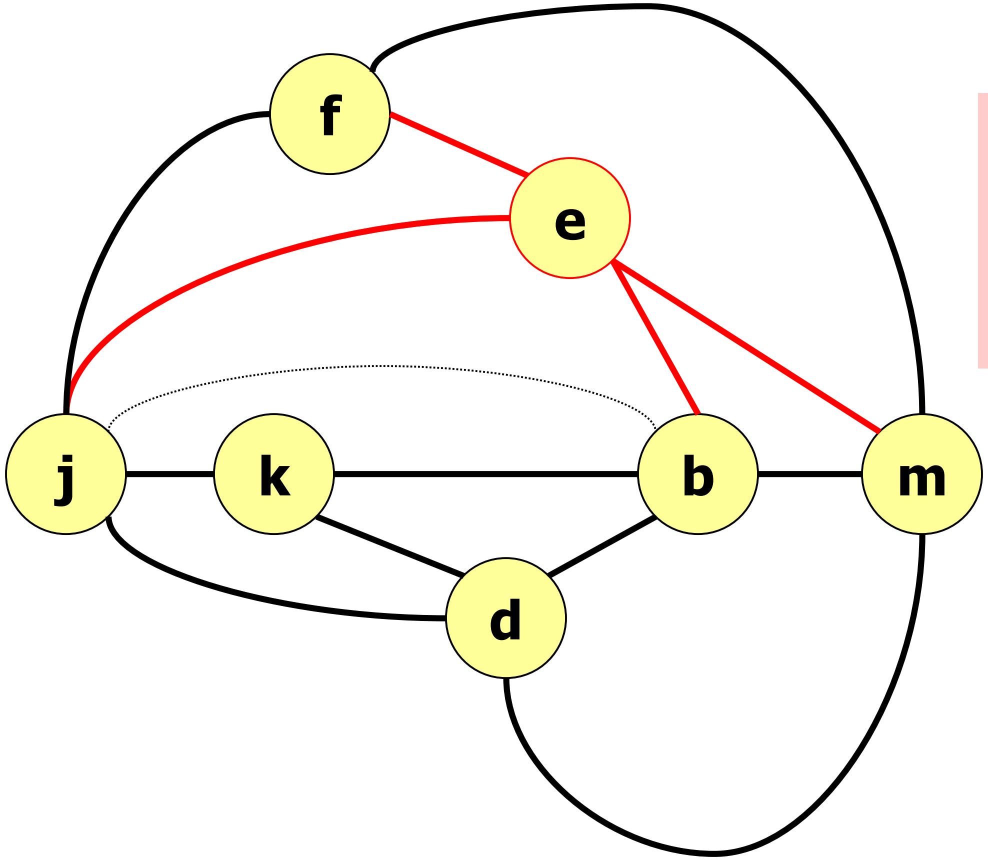


stack

(c, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 6: Spill (K=3)



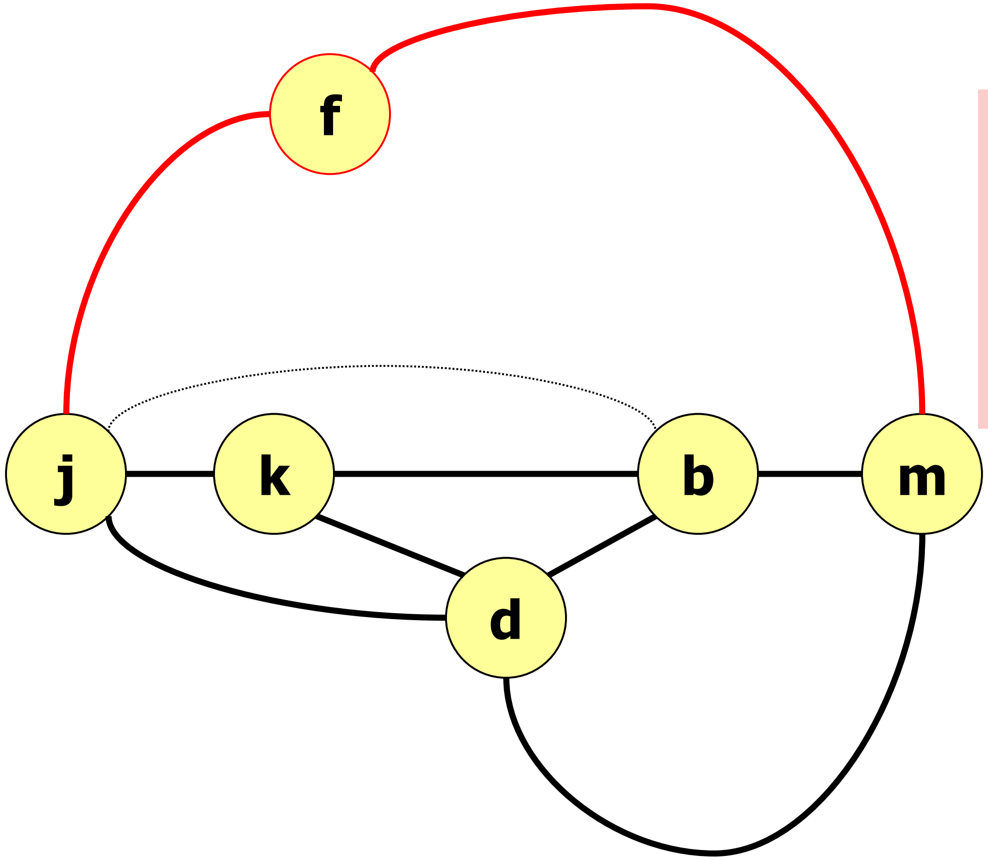
stack

(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

Neither coalescing nor freezing help us.
At this point we should use some profitability analysis to choose a node as *may-spill*.

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Simplify (K=3)

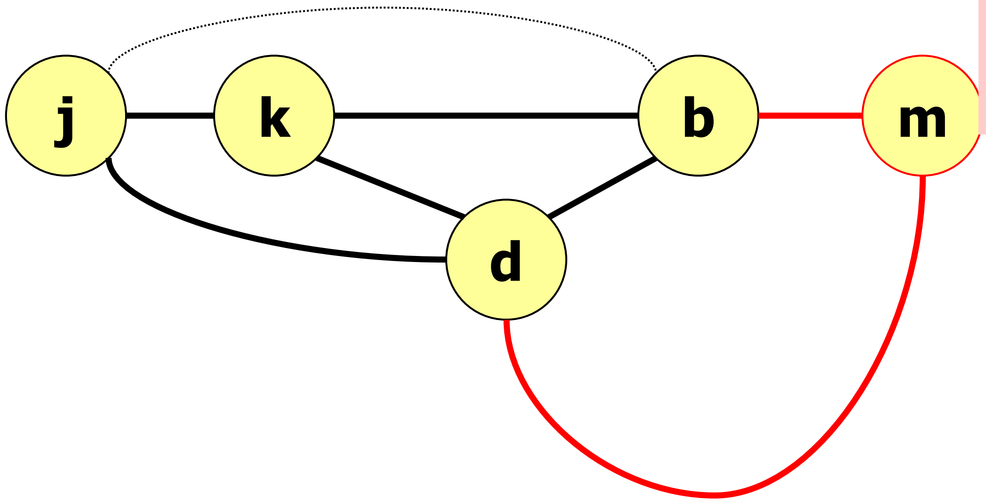


stack

(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Simplify (K=3)

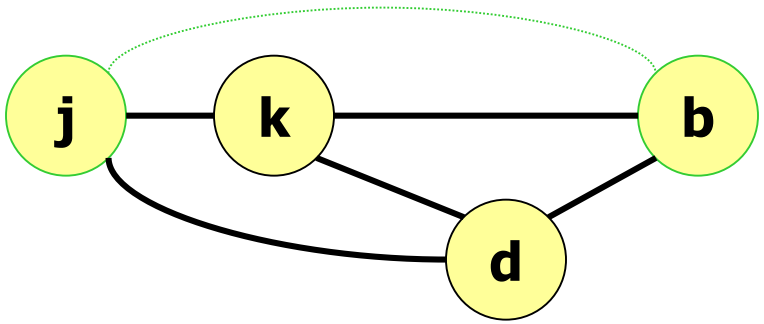


stack

(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Coalesce (K=3)

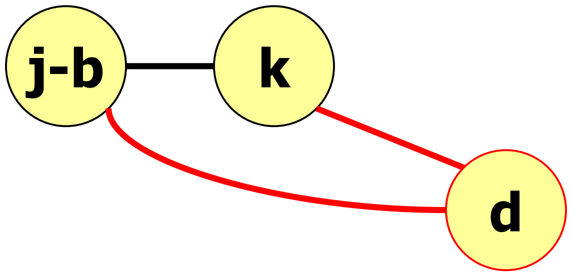


stack

(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Coalesce (K=3)

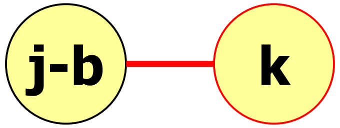


stack

(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Coalesce (K=3)

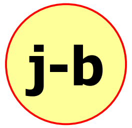


stack

(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Coalesce (K=3)

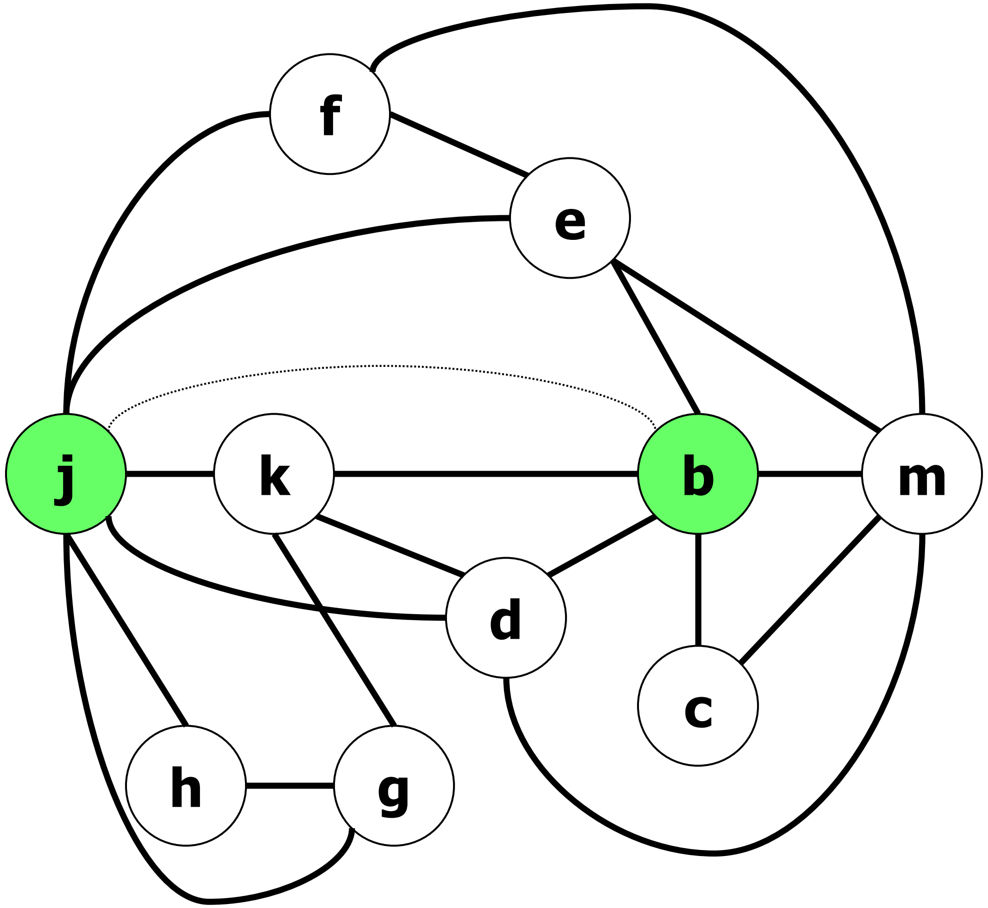


stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

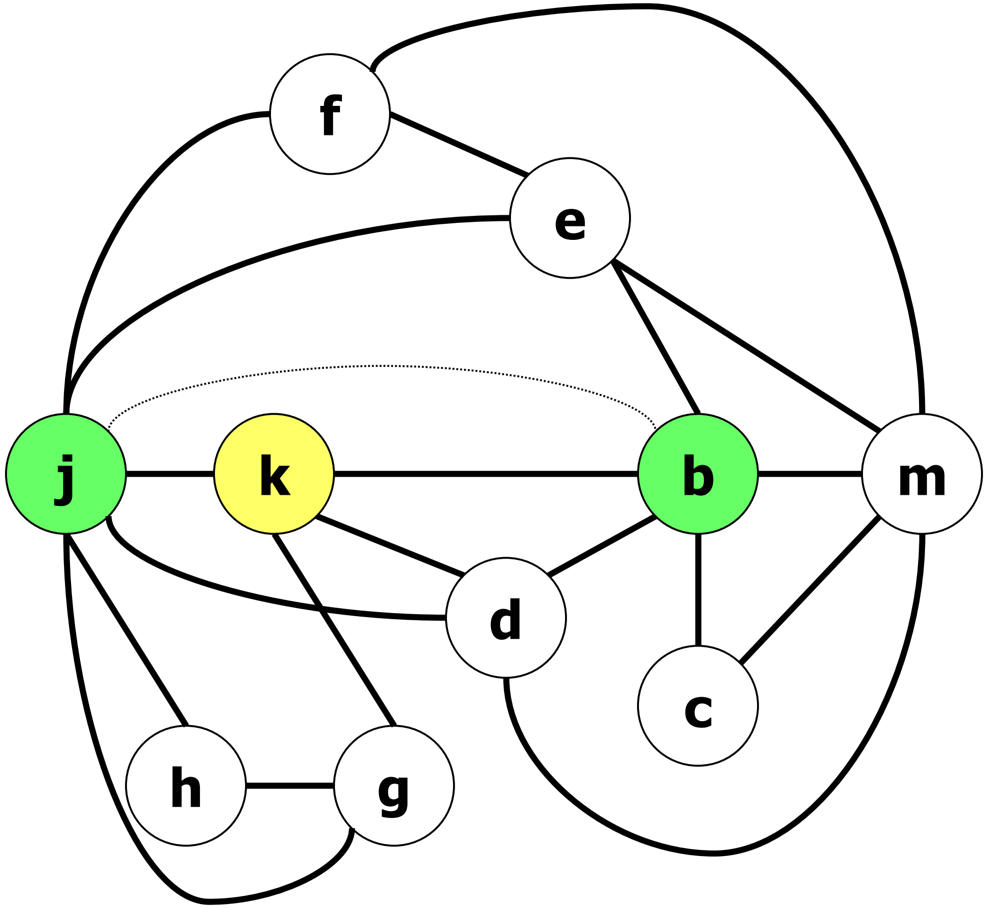
R1

R2

R3

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

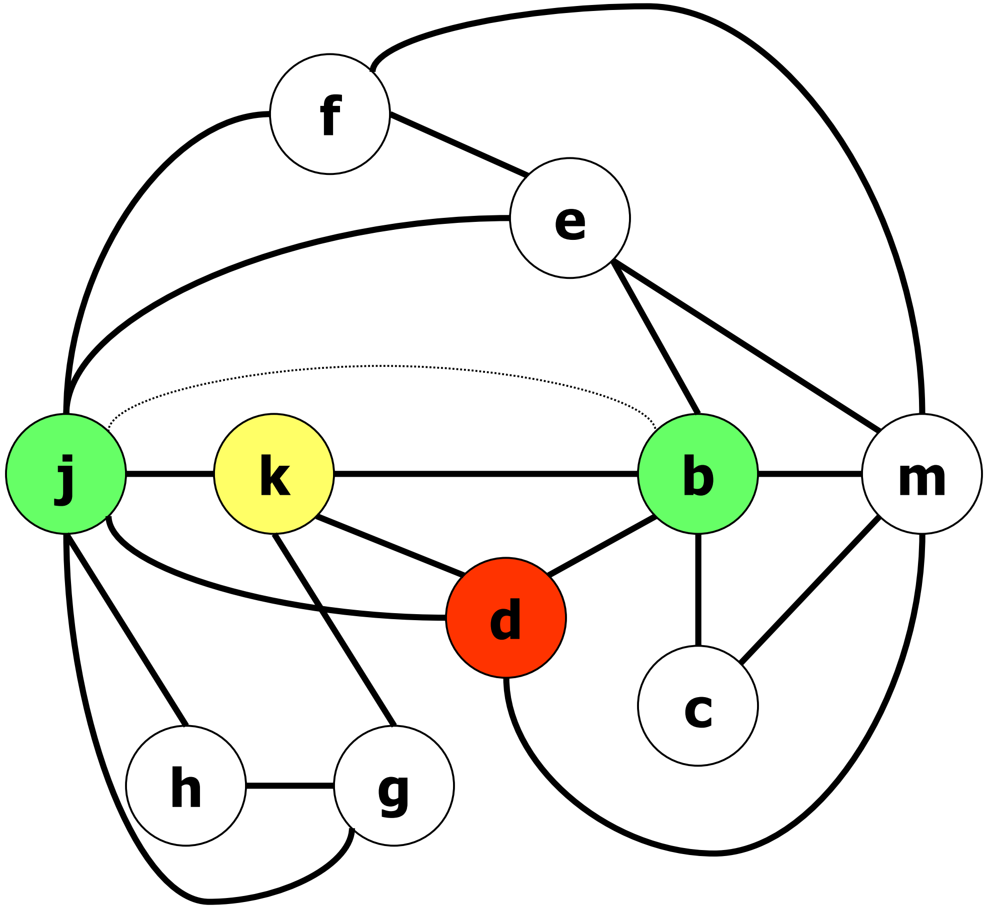
R1

R2

R3

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

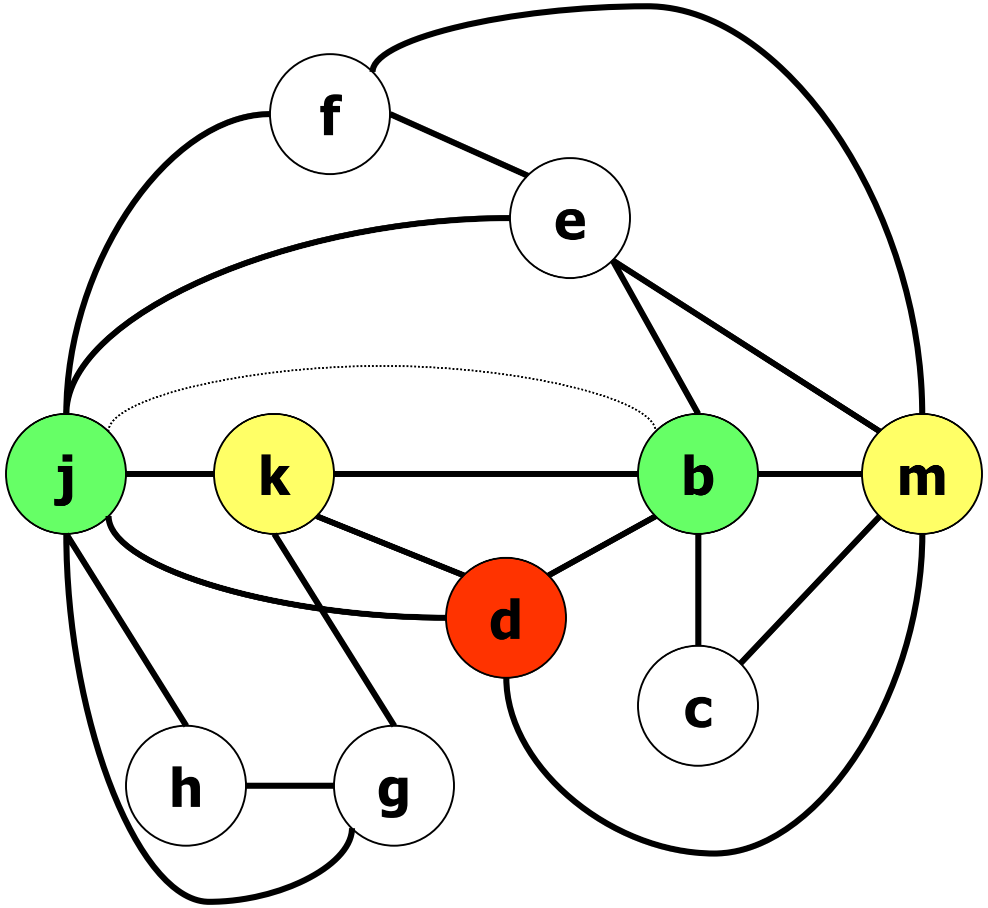
R1

R2

R3

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

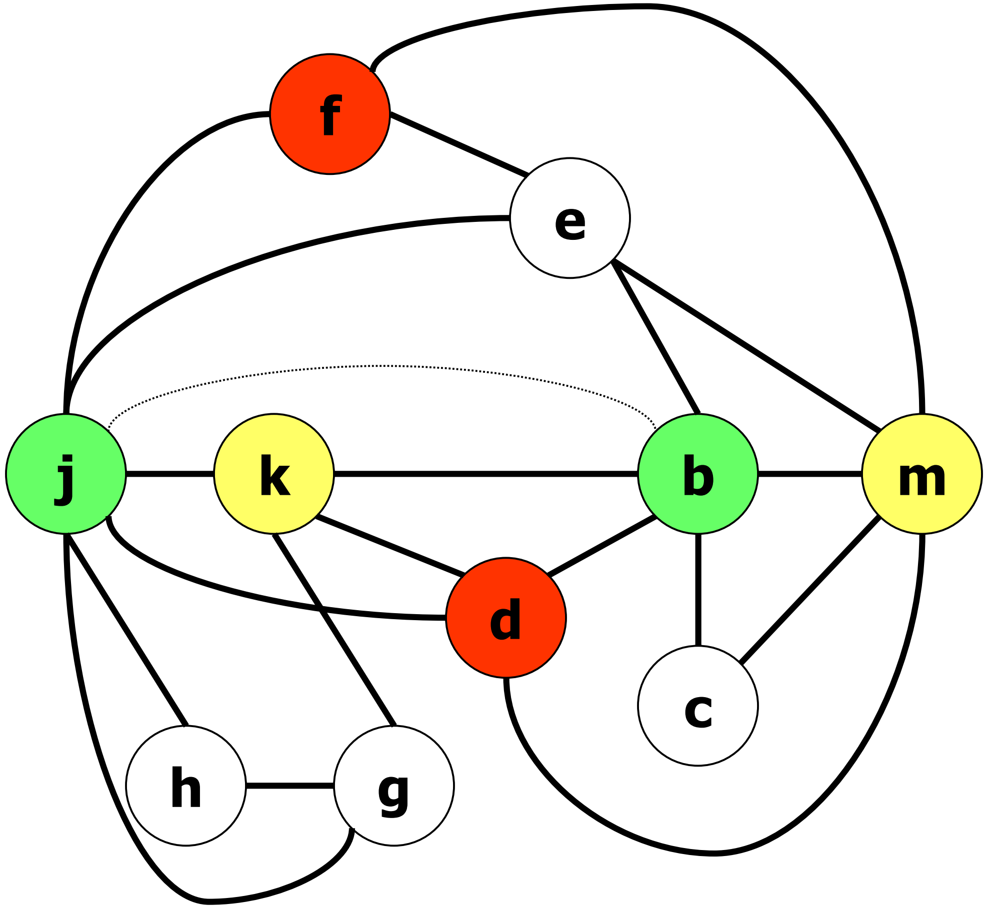
R1

R2

R3

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

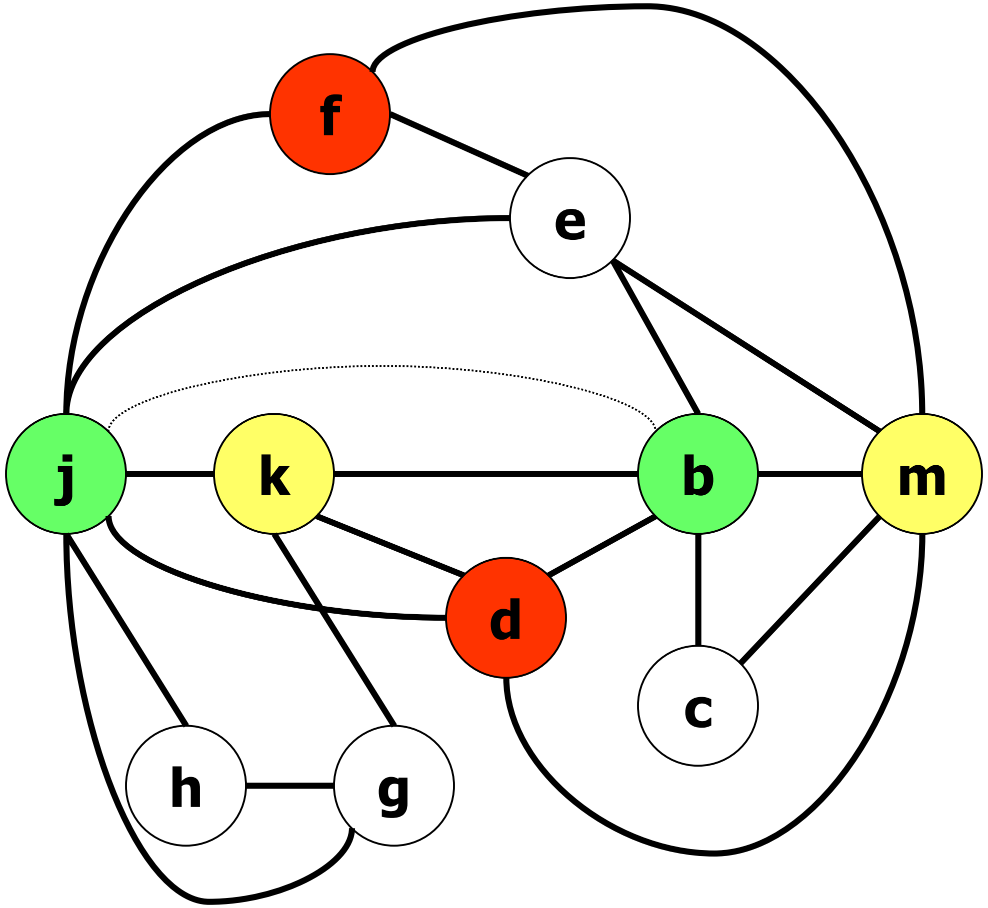
R1

R2

R3

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

R1

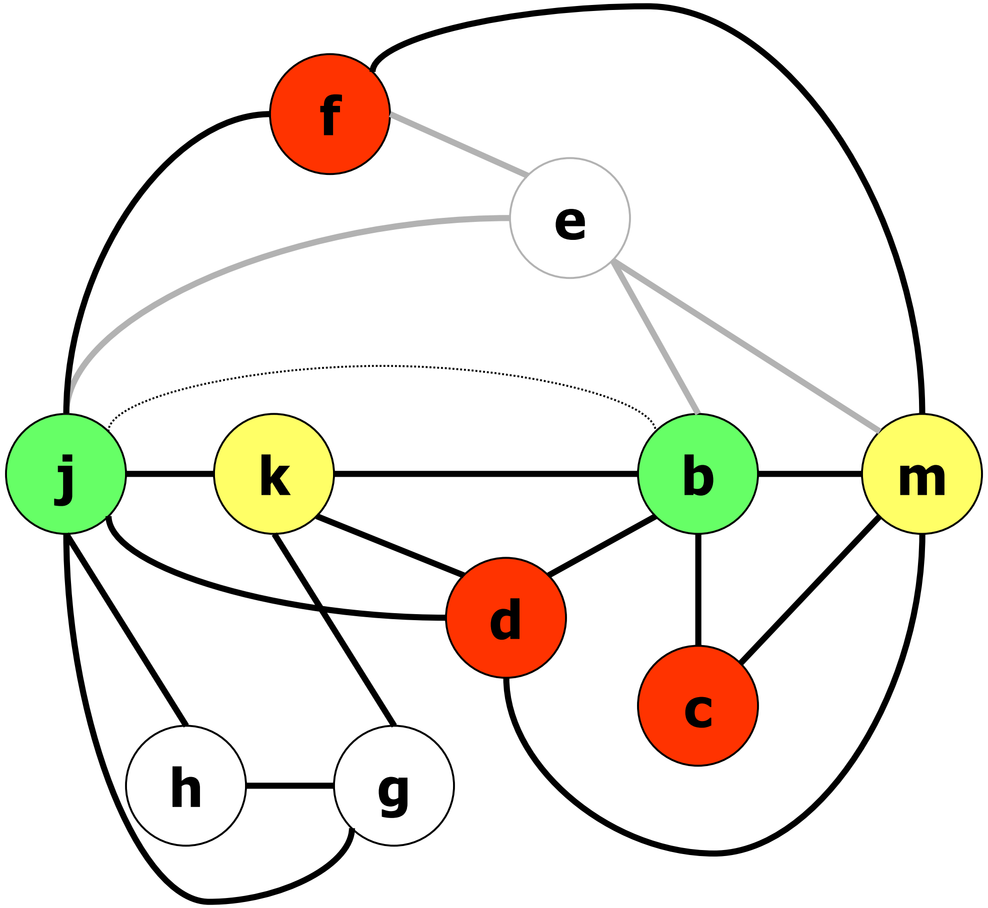
R2

R3

This is when our optimism could have paid off.

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

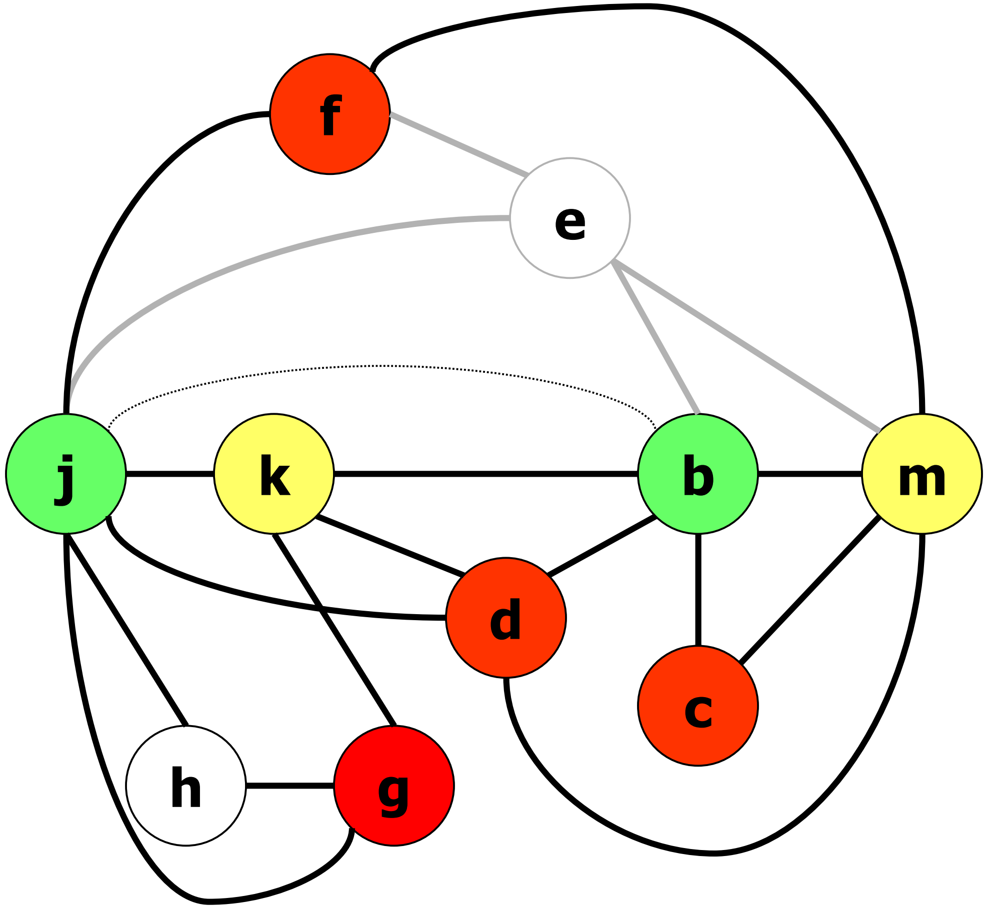
R1

R2

R3

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=3)

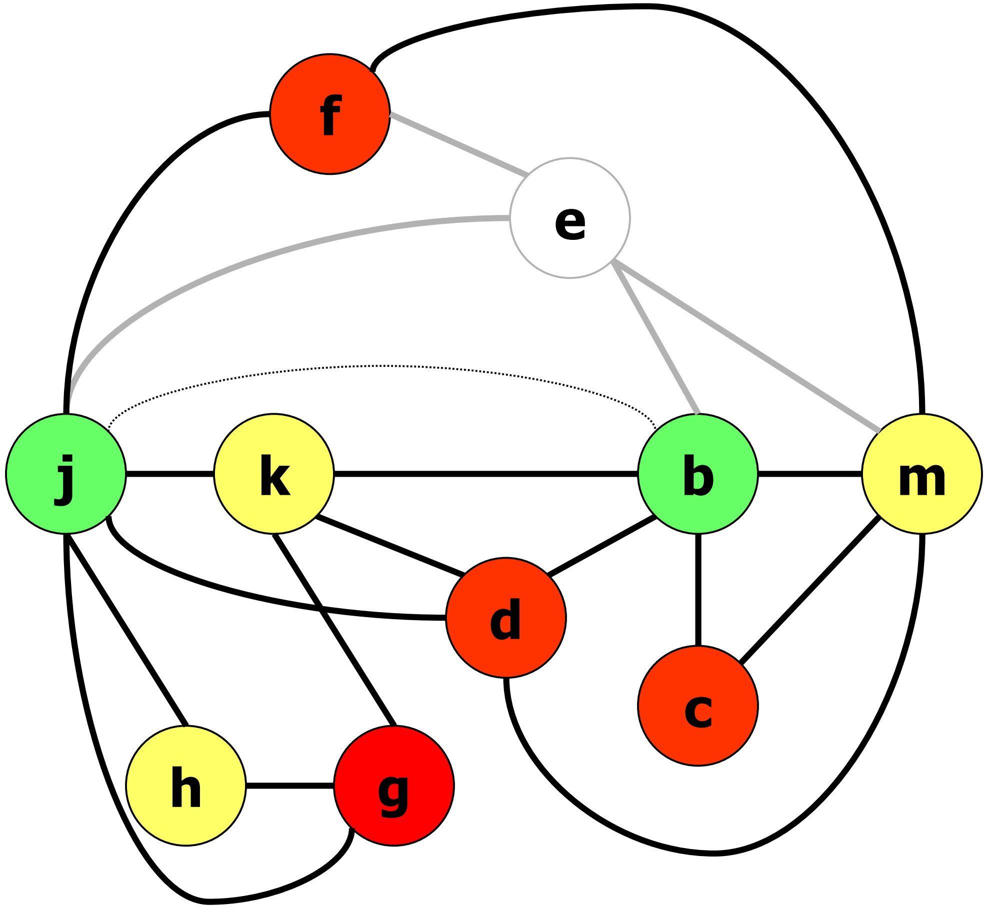


stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

(José Nelson Amaral based on Tiger Book, Appel)

Example:
Step 3: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

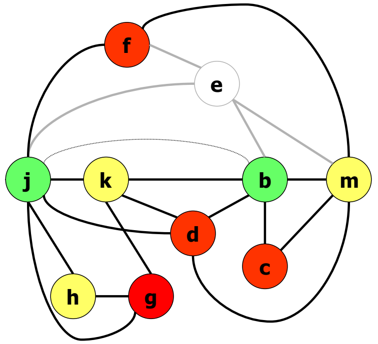
R1

R2

R3

R1={j,b}
R2={k,h,m}
R3={f,d,c,g}

(José Nelson Amaral based on Tiger Book, Appel)



Example: Step 3: Select (K=3)

R1={j,b}
R2={k,h,m}
R3={f,d,c,g}

R1

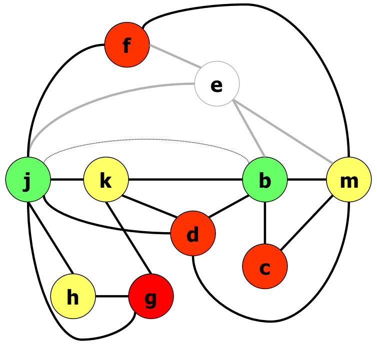
R2

R3

LIVE-IN: r2(k) r1(j)
r3 := mem[r1+12]
r2 := r2 -1
r3 := r3 + r2
e := mem[r1+8] ⇒ t4 := mem[r1+8]; mem[$sp+4] := t4
r2 := mem[r1+16]
r1 := mem[r3]
r3 := e + 8 ⇒ t5 := mem[$sp+4]; r3 := t5 + 8
r3 := r3
r2 := r2 + 4
r1 := r1
LIVE-OUT: r3(d) r2(k) r1(j)

A good optimizing compiler would recognize that the assignment to "e" can be moved to just before its use and no spilling would be needed!

(José Nelson Amaral based on Tiger Book, Appel)



Example:
Step 3: Select (K=3)

R1={j,b}
R2={k,h,m}
R3={f,d,c,g}

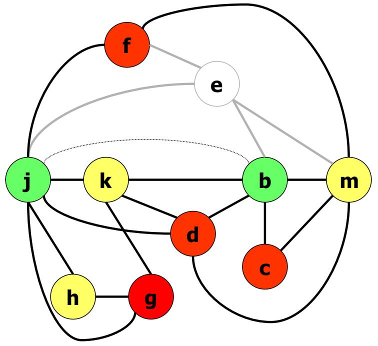
R1

R2

R3

LIVE-IN: r2(k) r1(j)
r3 := mem[r1+12]
r2 := r2 -1
r3 := r3 + r2
e := mem[r1+8] ⇒ t4 := mem[r1+8]; mem[$sp+4] := t4
r2 := mem[r1+16]
r1 := mem[r3]
r3 := e + 8 ⇒ t5 := mem[$sp+4]; r3 := t5 + 8
r2 := r2 + 4
LIVE-OUT: r3(d) r2(k) r1(j)

(José Nelson Amaral based on Tiger Book, Appel)



Example: Step 3: Select (K=3)

R1

R2

R3

LIVE-IN: r2(k) r1(j)
r3 := mem[r1+12]
r2 := r2 -1
r3 := r3 + r2
t4 := mem[r1+8]
mem[$sp+4] := t4
r2 := mem[r1+16]
r1 := mem[r3]
t5 := mem[$sp+4]
r3 := t5 + 8
r2 := r2 + 4
LIVE-OUT: r3(d) r2(k) r1(j)

Example:
Step 3: Select (K=3)

R1

R2

R3

LIVE-IN: r2(k) r1(j)

r3 := mem[r1+12]

r2 := r2 -1

r3 := r3 + r2

t4 := mem[r1+8]

mem[$sp+4] := t4

r2 := mem[r1+16]

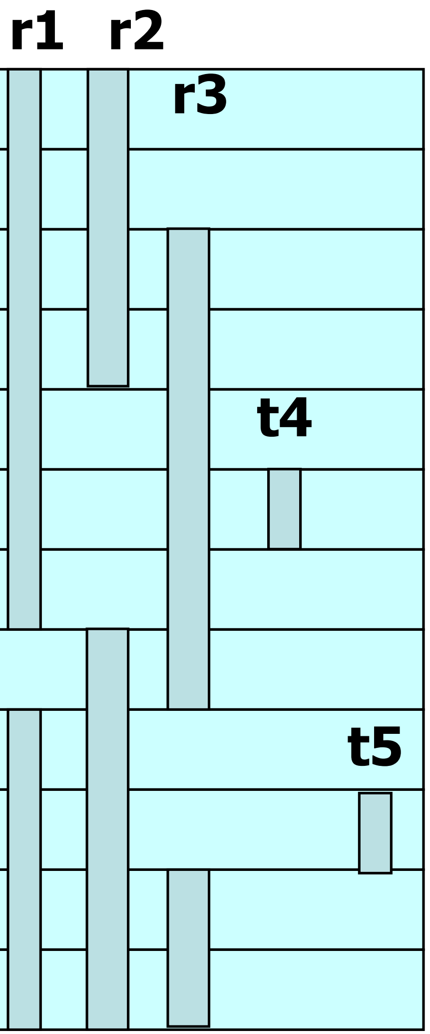
r1 := mem[r3]

t5 := mem[$sp+4]

r3 := t5 + 8

r2 := r2 + 4

LIVE-OUT: r3(d) r2(k) r1(j)



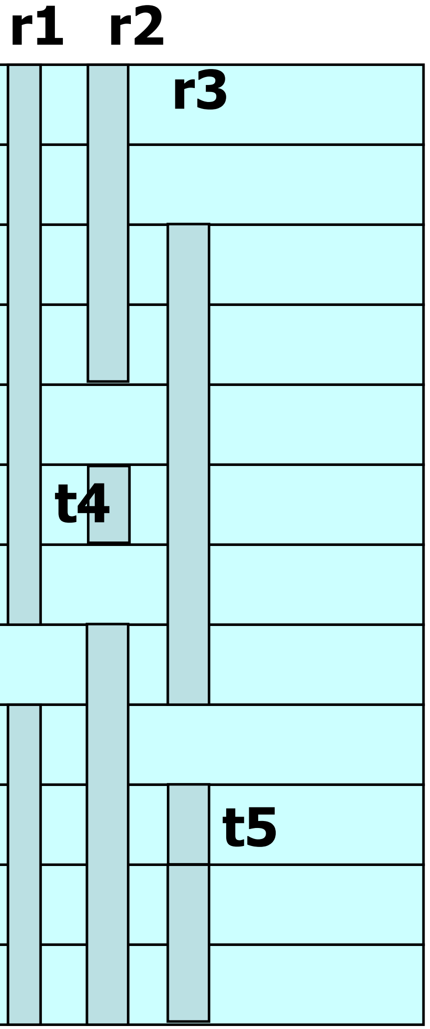
Example:
Step 3: Select (K=3)

R1

R2

R3

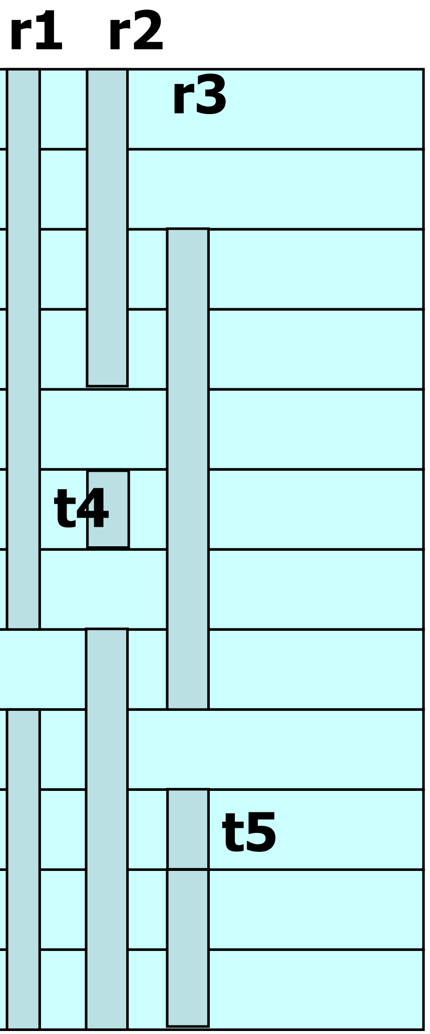
```
LIVE-IN: r2(k)  r1(j)
    r3 := mem[r1+12]
    r2 := r2 -1
    r3 := r3 + r2
    t4 := mem[r1+8]
    mem[$sp+4] := t4
    r2 := mem[r1+16]
    r1 := mem[r3]
    t5 := mem[$sp+4]
    r3 := t5 + 8
    r2 := r2 + 4
LIVE-OUT: r3(d)  r2(k)  r1(j)
```



After repeating Register Allocation …

Example:
Step 3: Select (K=3)


```
LIVE-IN: r2(k)  r1(j)
    r3 := mem[r1+12]
    r2 := r2 -1
    r3 := r3 + r2
    r2 := mem[r1+8]
    mem[$sp+4] := r2
    r2 := mem[r1+16]
    r1 := mem[r3]
    r5 := mem[$sp+4]
    r3 := r5 + 8
    r2 := r2 + 4
LIVE-OUT: r3(d)  r2(k)  r1(j)
```



After repeating Register Allocation …

Live Range Splitting

- The basic coloring algorithm does not consider cases in which a variable can be allocated to a register for part of its live range
  - Some compilers split live ranges within the iteration structure of the coloring algorithm
  - When a variable is split into two new variables, one of the new variables might be profitably assigned to a register while the other is not

(José Nelson Amaral)

Length of Live Ranges

- The interference graph does not contain information of where in the CFG variables interfere and what the length of a variable's live range is
- For example, if we only had few available registers in the following intermediate-code example, the right choice would be to spill variable w because it has the longest live range:

```
x = w + 1
c = a - 2
y = x * 3
z = w + y
```


(José Nelson Amaral)

Summary

- Register allocation has three major parts
  - Liveness analysis
  - Graph coloring
  - Program transformation (move coalescing and spilling)
- See Sections 11.1-11.3 in the Tiger Book (Appel)

References

- Kempe, A. B. 1879. On the geographical problem of the four colors. Am. J. Math. 2, 193-200.
- G.J. Chaitin, M.A. Auslander, A.K. Chandra, J. Cocke, M.E. Hopkins, and P.W. Markstein. **Register Allocation via Coloring**. Computer Languages, 6:45-57, January 1981.
- *Gregory Chaitin. 2004.* **Register allocation and spilling via graph coloring**. *SIGPLAN Not. 39, 4 (April 2004), 66–74. [1982] DOI: https://doi.org/10.1145/989393.989403*
- See also Patent US4571678A: "**Register allocation and spilling via graph coloring**," Inventor: Gregory J. Chaitin https://patents.google.com/patent/US4571678A/en
- Preston Briggs, Keith D. Cooper, and Linda Torczon. **Improvements to Graph Coloring Register Allocation**. ACM Transactions on Programming Languages and Systems, 16(3):428-455, May 1994. https://doi.org/10.1145/177492.177575
- Lal George and Andrew W. Appel. **Iterated register coalescing**. ACM Trans. Program. Lang. Syst., 18(3):300-324, 1996. https://doi.org/10.1145/229542.229546