EECS 583 – Class 17
Register Allocation

University of Michigan

November 18, 2019

Announcements + Reading Material

- Today's class reading
  - "Register Allocation and Spilling Via Graph Coloring," G. Chaitin, Proc. 1982 SIGPLAN Symposium on Compiler Construction, 1982.
- Signup sheet for final project presentations available next class

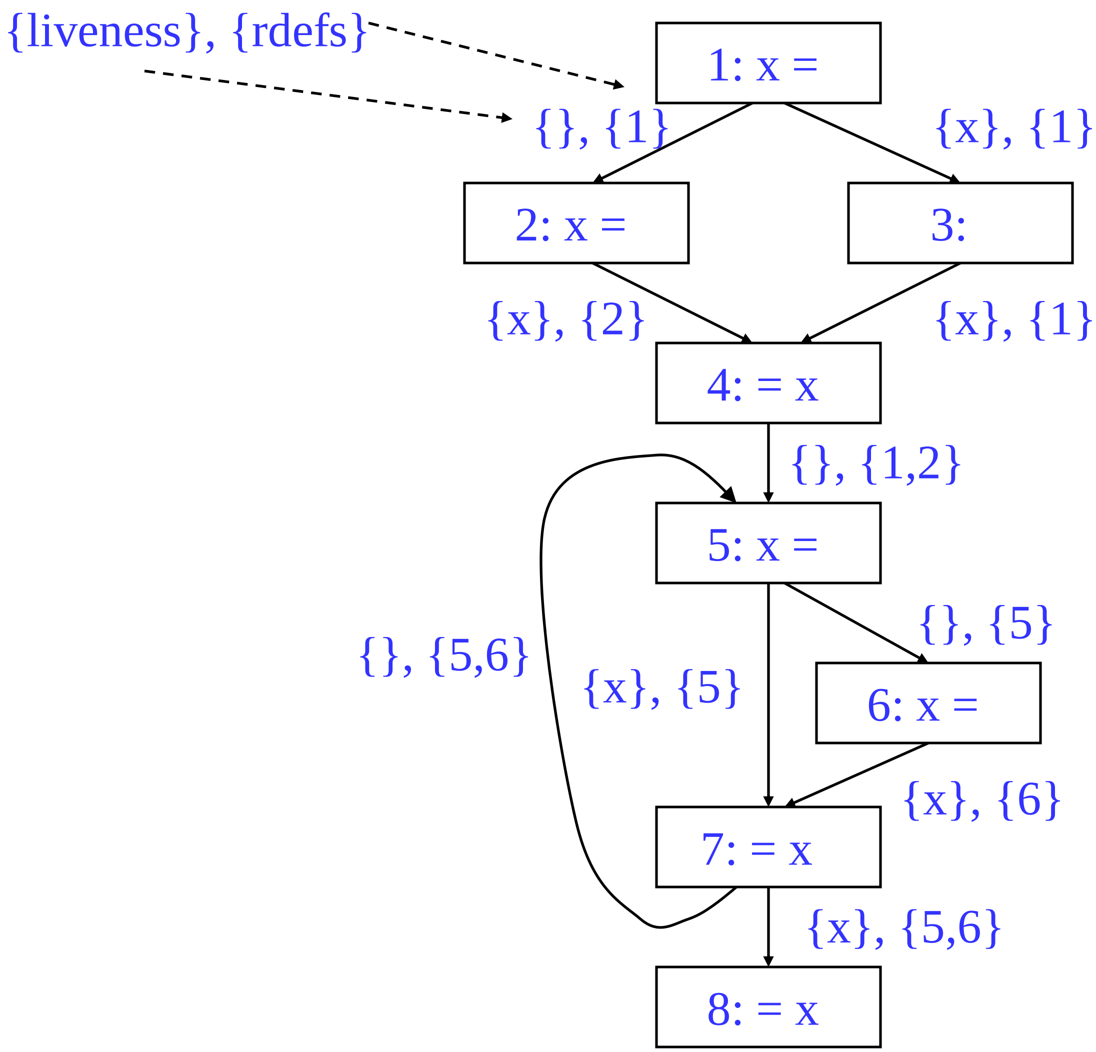
Register Allocation: Problem Definition

- Through optimization, assume an infinite number of virtual registers
  - » Now, must allocate these infinite virtual registers to a limited supply of hardware registers
  - » Want most frequently accessed variables in registers
    - Speed, registers much faster than memory
    - Direct access as an operand
  - » Any VR that cannot be mapped into a physical register is said to be <u>spilled</u>
- Questions to answer
  - » What is the minimum number of registers needed to avoid spilling?
  - » Given n registers, is spilling necessary
  - » Find an assignment of virtual registers to physical registers
  - » If there are not enough physical registers, which virtual registers get spilled?

Live Range

- Value = definition of a register
- Live range = Set of operations
  - » 1 more or values connected by common uses
  - » A single VR may have several live ranges
- Live ranges are constructed by taking the intersection of reaching defs and liveness
  - » Initially, a live range consists of a single definition and all ops in a function in which that definition is live

Example – Constructing Live Ranges

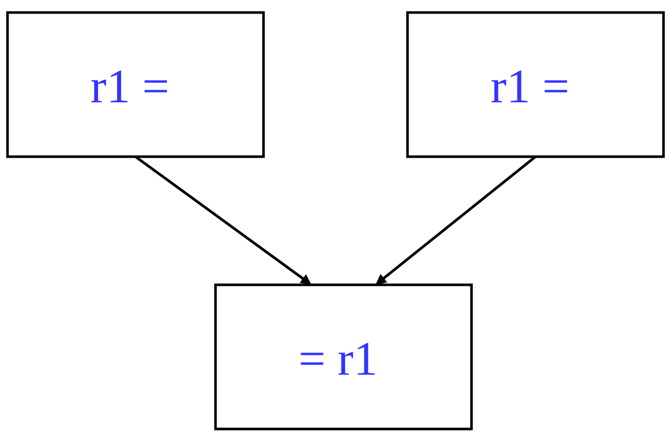


Each definition is the seed of a live range.
Ops are added to the LR where both the defn reaches and the variable is live

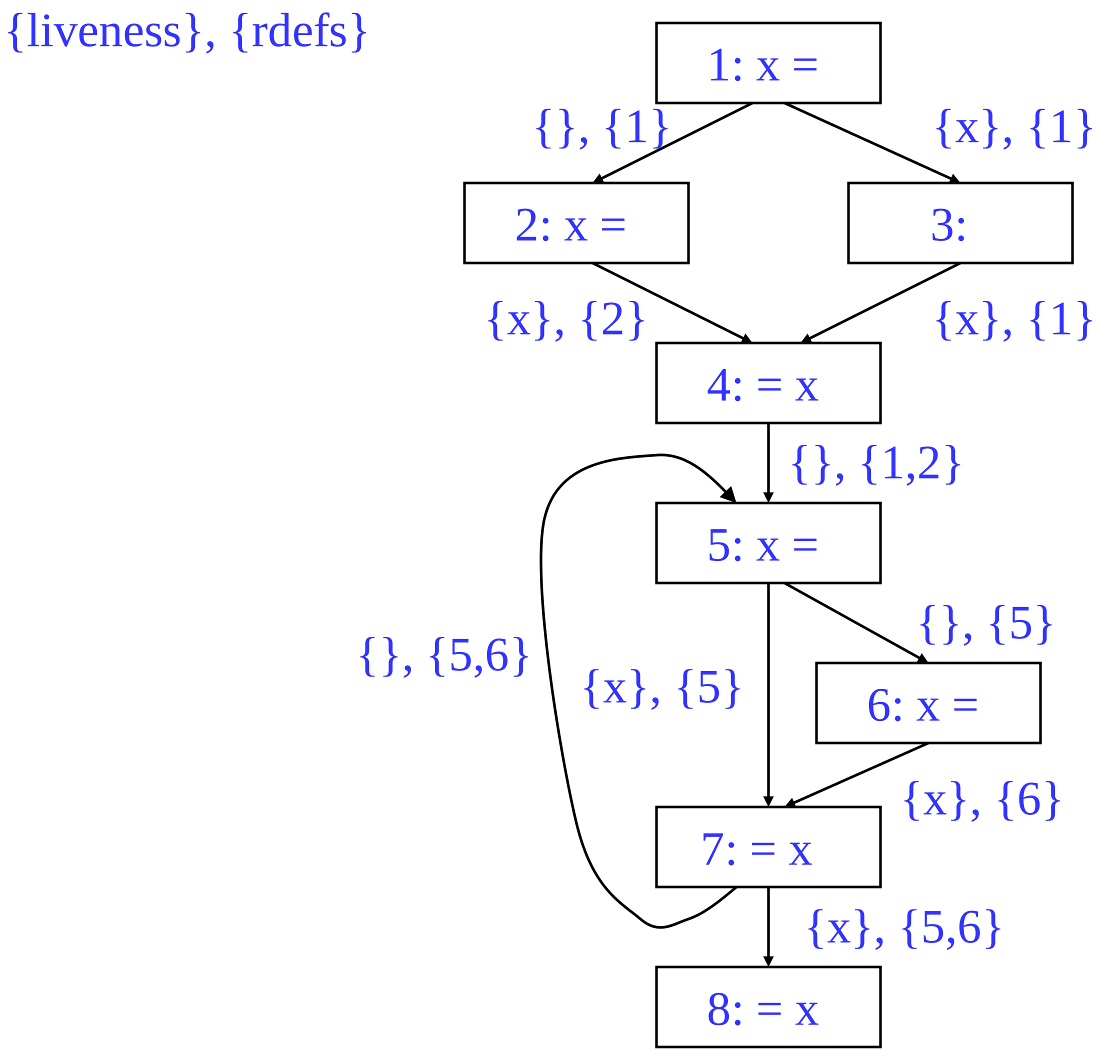
LR1 for def 1 = {1,3,4}
LR2 for def 2 = {2,4}
LR3 for def 5 = {5,7,8}
LR4 for def 6 = {6,7,8}

Merging Live Ranges

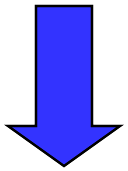
- If 2 live ranges for the same VR overlap, they must be merged to ensure correctness
  - » LRs replaced by a new LR that is the union of the LRs
  - » Multiple defs reaching a common use
  - » Conservatively, all LRs for the same VR could be merged
    - Makes LRs larger than need be, but done for simplicity
    - We will not assume this



Example – Merging Live Ranges



LR1 for def 1 = {1,3,4}
LR2 for def 2 = {2,4}
LR3 for def 5 = {5,7,8}
LR4 for def 6 = {6,7,8}



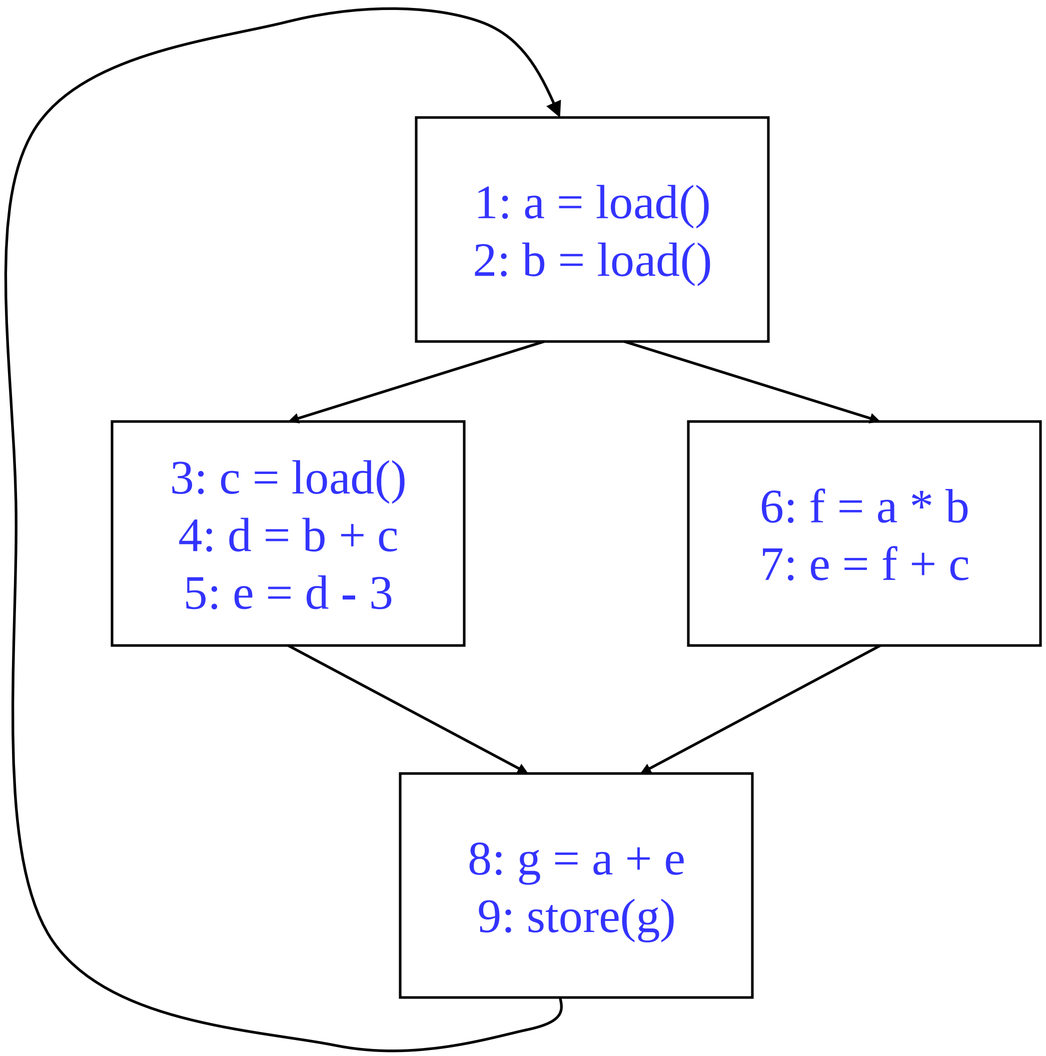
Merge LR1 and LR2, LR3 and LR4

LR5 = {1,2,3,4}
LR6 = {5,6,7,8}

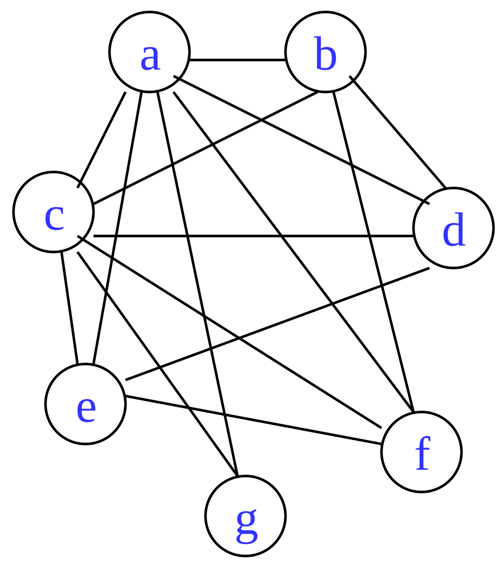
Interference

- Two live ranges interfere if they share one or more ops in common
  - Thus, they cannot occupy the same physical register
  - Or a live value would be lost
- Interference graph
  - Undirected graph where
    - Nodes are live ranges
    - There is an edge between 2 nodes if the live ranges interfere
  - What's not represented by this graph
    - Extent of interference between the LRs
    - Where in the program is the interference

Example – Interference Graph



lr(a) = {1,2,3,4,5,6,7,8}
lr(b) = {2,3,4,6}
lr(c) = {1,2,3,4,5,6,7,8,9}
lr(d) = {4,5}
lr(e) = {5,7,8}
lr(f) = {6,7}
lr{g} = {8,9}



Graph Coloring

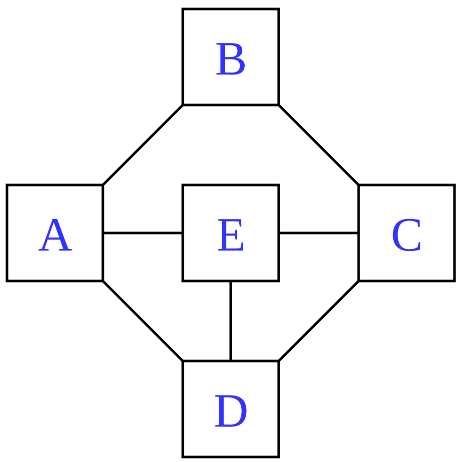
- A graph is n-colorable if every node in the graph can be colored with one of the n colors such that 2 adjacent nodes do not have the same color
  - Model register allocation as graph coloring
  - Use the fewest colors (physical registers)
  - Spilling is necessary if the graph is not n-colorable where n is the number of physical registers
- Optimal graph coloring is NP-complete for n > 2
  - Use heuristics proposed by compiler developers
    - "Register Allocation Via Coloring", G. Chaitin et al, 1981
    - "Improvement to Graph Coloring Register Allocation", P. Briggs et al, 1989
  - **Observation** – a node with degree < n in the interference can always be successfully colored given its neighbors colors

Coloring Algorithm

- 1. While any node, x, has < n neighbors
  - » Remove x and its edges from the graph
  - » Push x onto a stack
- 2. If the remaining graph is non-empty
  - » Compute cost of spilling each node (live range)
    - For each reference to the register in the live range
      - Cost += (execution frequency * spill cost)
  - » Let NB(x) = number of neighbors of x
  - » Remove node x that has the smallest cost(x) / NB(x)
    - Push x onto a stack (mark as spilled)
  - » Go back to step 1
- While stack is non-empty
  - » Pop x from the stack
  - » If x's neighbors are assigned fewer than R colors, then assign x any unsigned color, else leave x uncolored

Example – Finding Number of Needed Colors

How many colors are needed to color this graph?



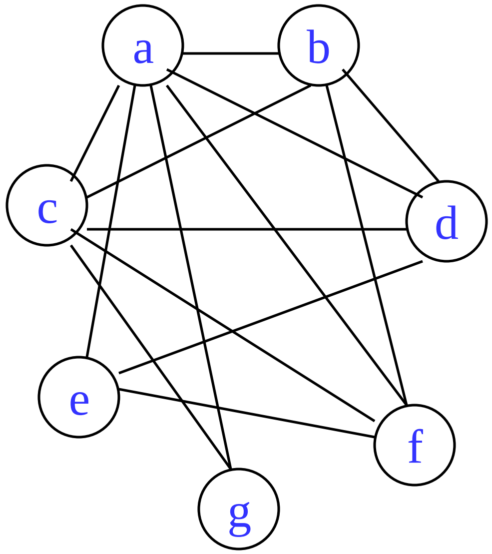
Try n=1, no, cannot remove any nodes

Try n=2, no again, cannot remove any nodes

Try n=3,

Remove B
Then can remove A, C
Then can remove D, E
Thus it is 3-colorable

Example – Do a 3-Coloring



lr(a) = {1,2,3,4,5,6,7,8}
refs(a) = {1,6,8}
lr(b) = {2,3,4,6}
refs(b) = {2,4,6}
lr(c) = {1,2,3,4,5,6,7,8,9}
refs(c) = {3,4,7}
lr(d) = {4,5}
refs(d) = {4,5}
lr(e) = {5,7,8}
refs(e) = {5,7,8}
lr(f) = {6,7}
refs(f) = {6,7}
lr{g} = {8,9}
refs(g) = {8,9}

Profile freqs
1,2 = 100
3,4,5 = 75
6,7 = 25
8,9 = 100

Assume each spill requires 1 operation

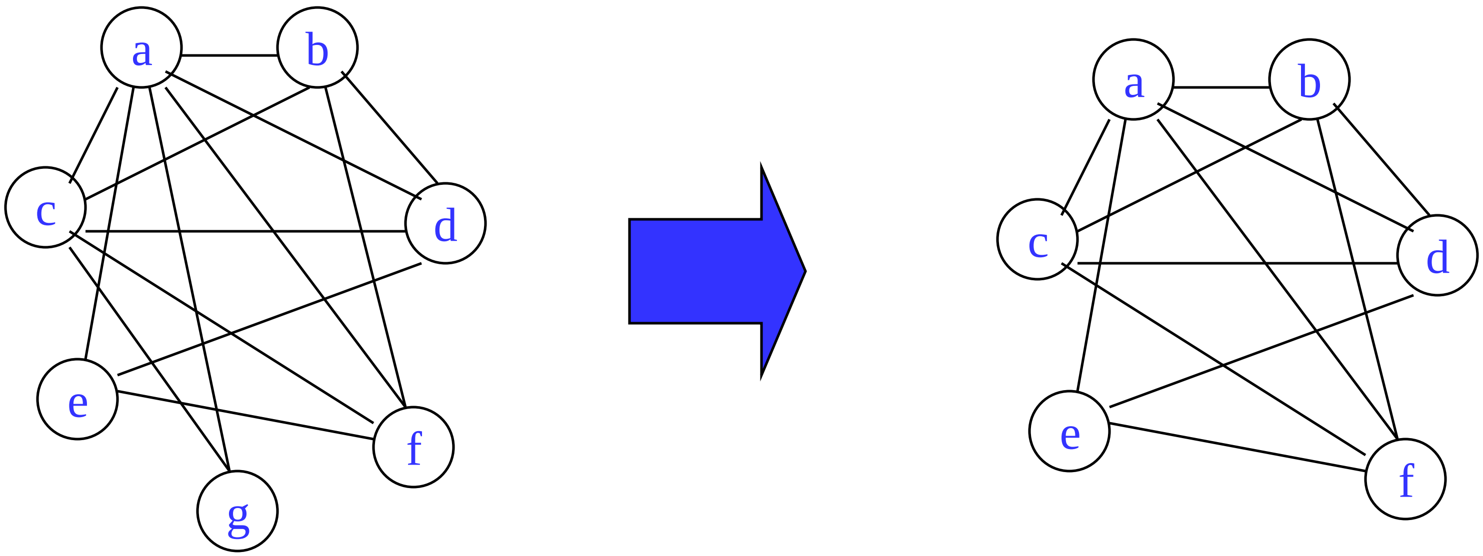
	a	b	c	d	e	f	g
cost	225	200	175	150	200	50	200
neighbors	6	4	5	4	3	4	2
cost/n	37.5	50	35	37.5	66.7	12.5	100

Example – Do a 3-Coloring (2)

Remove all nodes < 3 neighbors

So, g can be removed

Stack
g

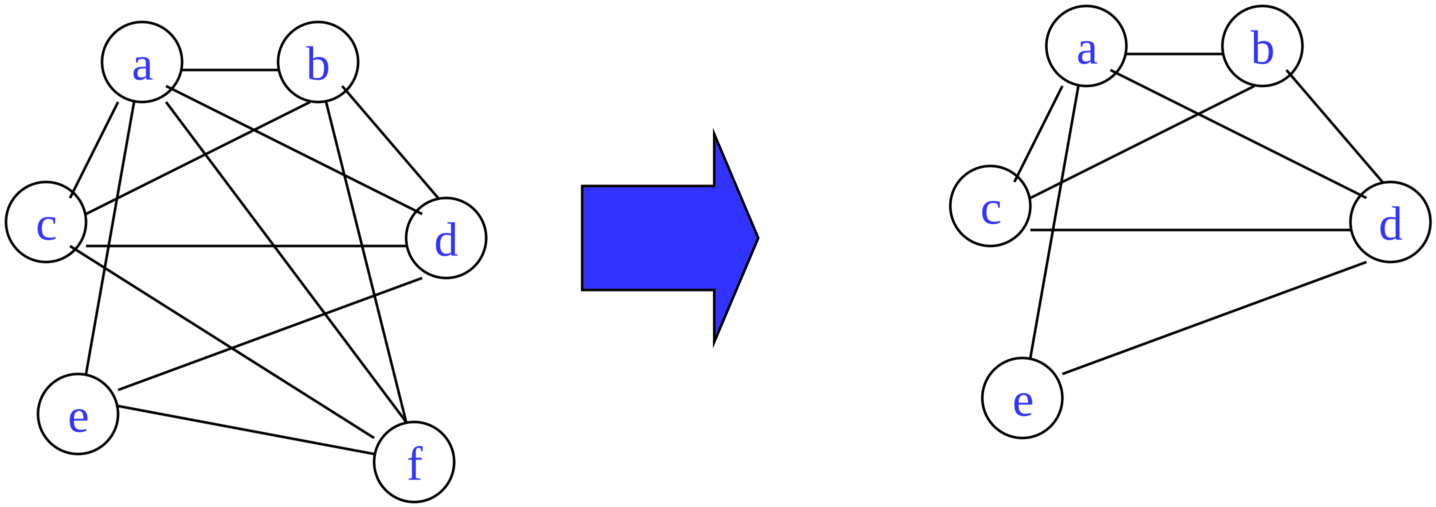


Example – Do a 3-Coloring (3)

Now must spill a node

Choose one with the smallest cost/NB → f is chosen

Stack
f (spilled)
g

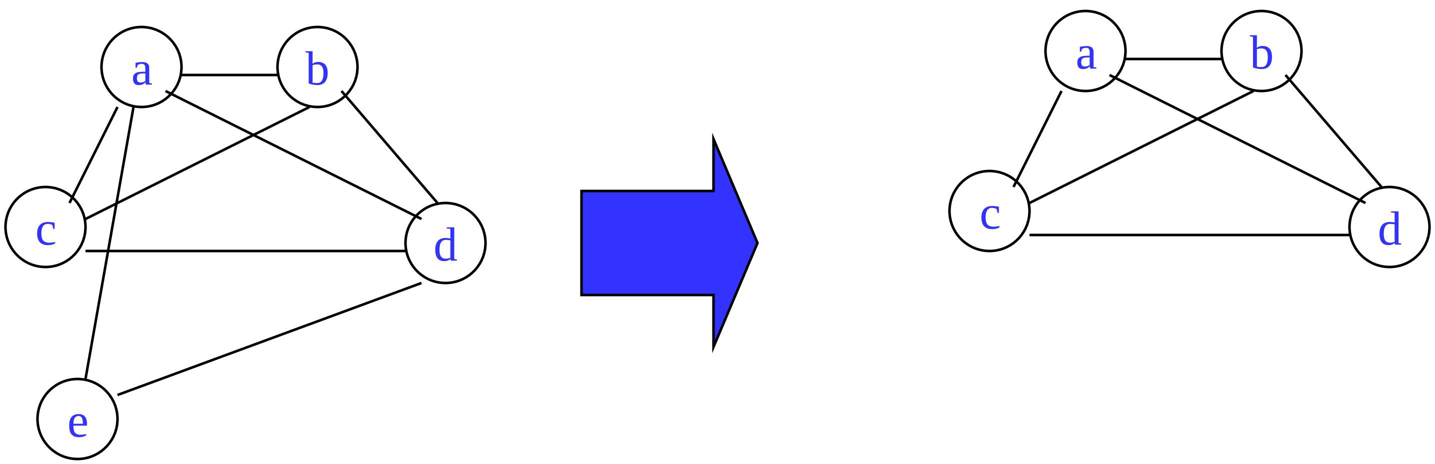


Example – Do a 3-Coloring (4)

Remove all nodes < 3 neighbors

So, e can be removed

Stack
e
f (spilled)
g

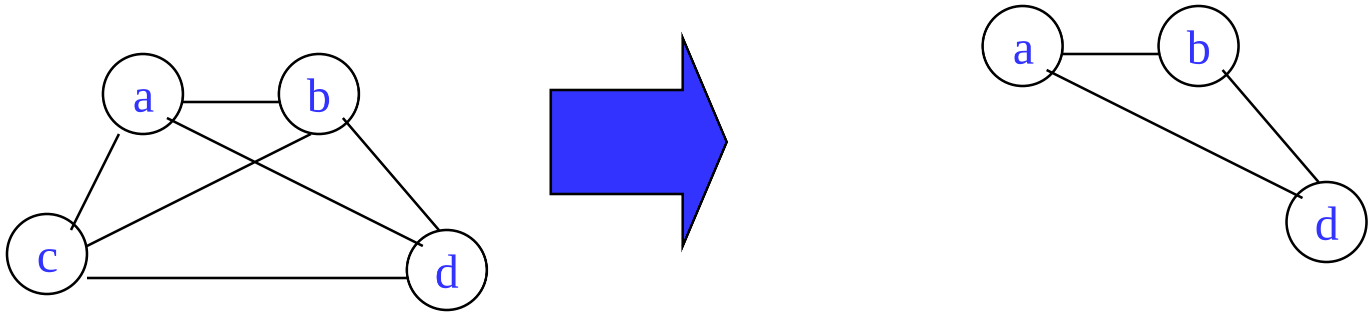


Example – Do a 3-Coloring (5)

Now must spill another node

Choose one with the smallest cost/NB → c is chosen

Stack
c (spilled)
e
f (spilled)
g

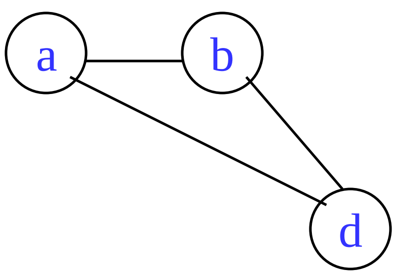


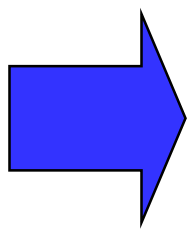
Example – Do a 3-Coloring (6)

Remove all nodes < 3 neighbors

So, a, b, d can be removed

Stack
d
b
a
c (spilled)
e
f (spilled)
g

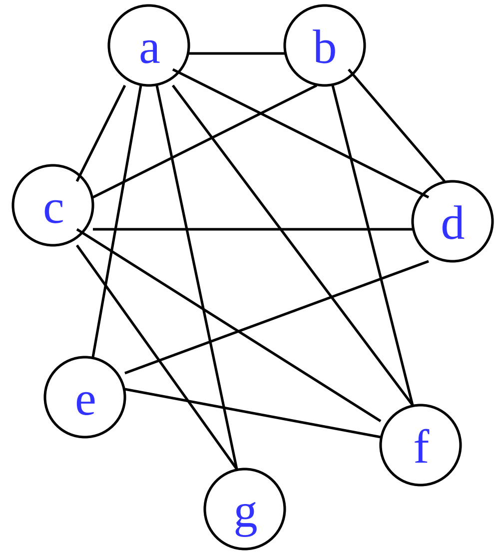




Null

Example – Do a 3-Coloring (7)

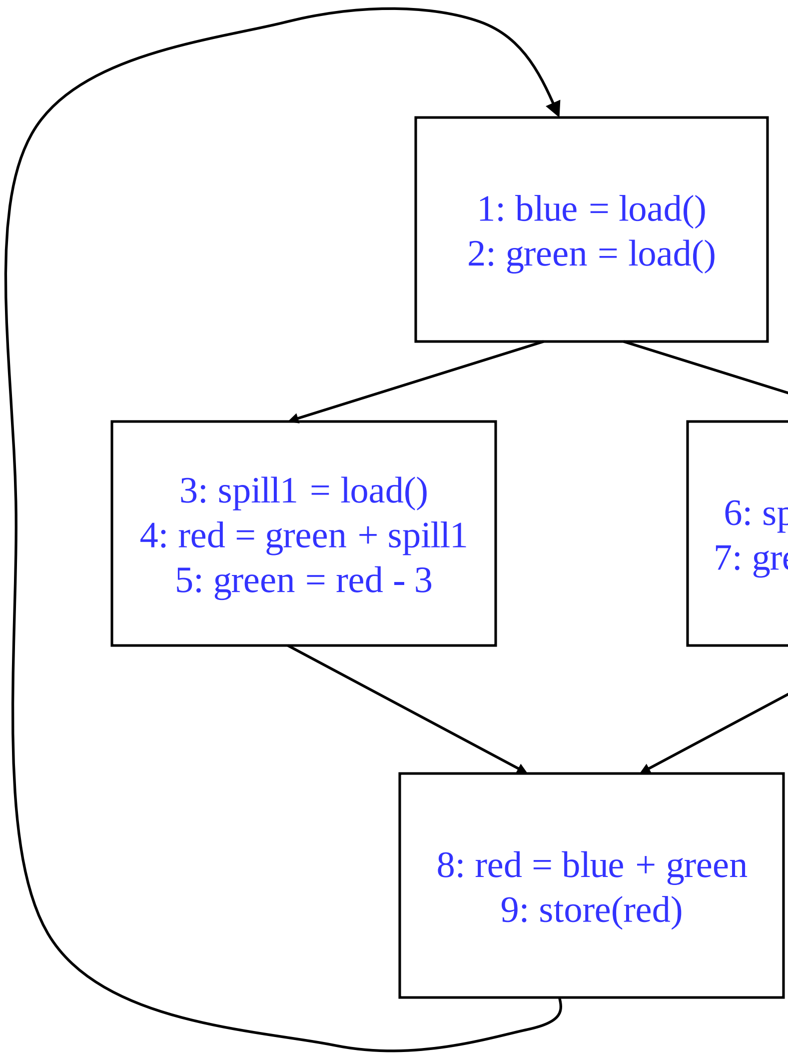
<u>Stack</u>
d
b
a
c (spilled)
e
f (spilled)
g



Have 3 colors: red, green, blue, pop off the stack assigning colors
only consider conflicts with non-spilled nodes already popped off stack

d → red
b → green (cannot choose red)
a → blue (cannot choose red or green)
c → no color (spilled)
e → green (cannot choose red or blue)
f → no color (spilled)
g → red (cannot choose blue)

Example – Do a 3-Coloring (8)



d → red
b → green
a → blue
c → no color
e → green
f → no color
g → red

Notes: no spills in the blocks executed 100 times. Most spills in the block executed 25 times. Longest lifetime (c) also spilled