Register Allocation

- **Goal:** replace temporary variable accesses by register accesses
- **Why?**
- **Constraints:**
  - Fixed set of registers
  - Two simultaneously live variables cannot be allocated to the same register

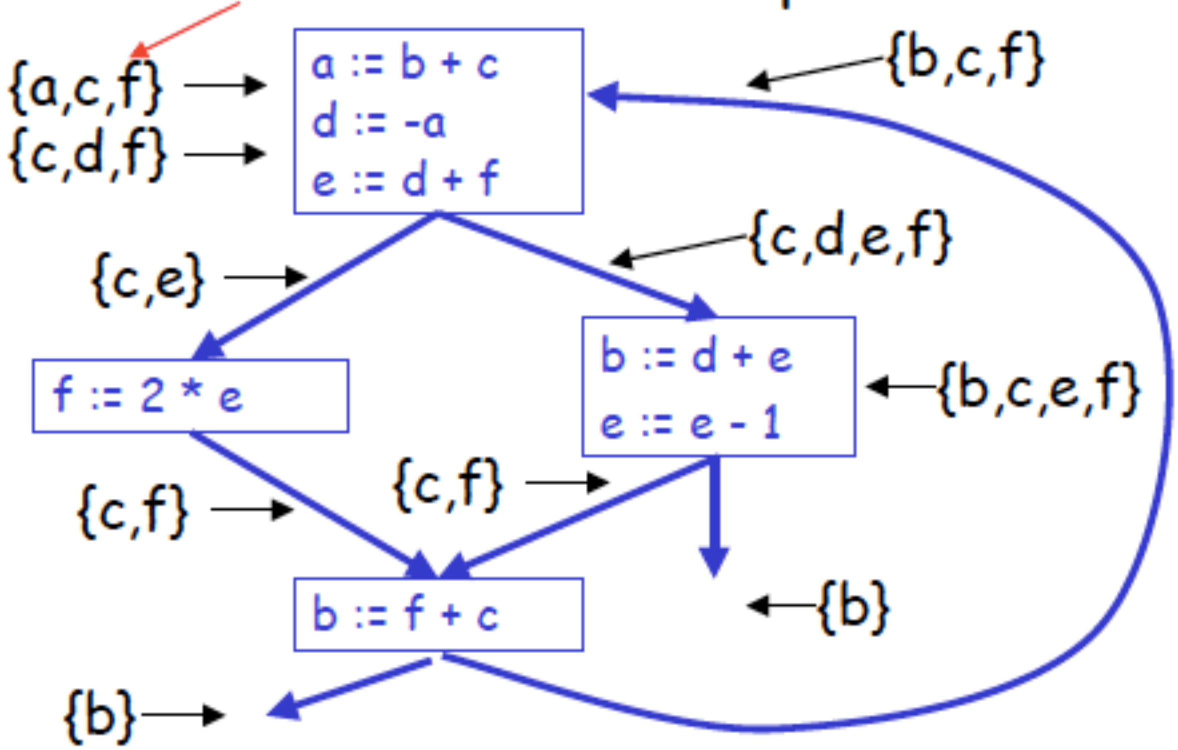
*A variable is **live** if it will be used again before being redefined.*

1. Identify Live Variable Ranges

Basic rule:

Temporaries t1 and t2 can share the same register if at any point in the program at most one of t1 or t2 is live !

- Compute live variables for each point:

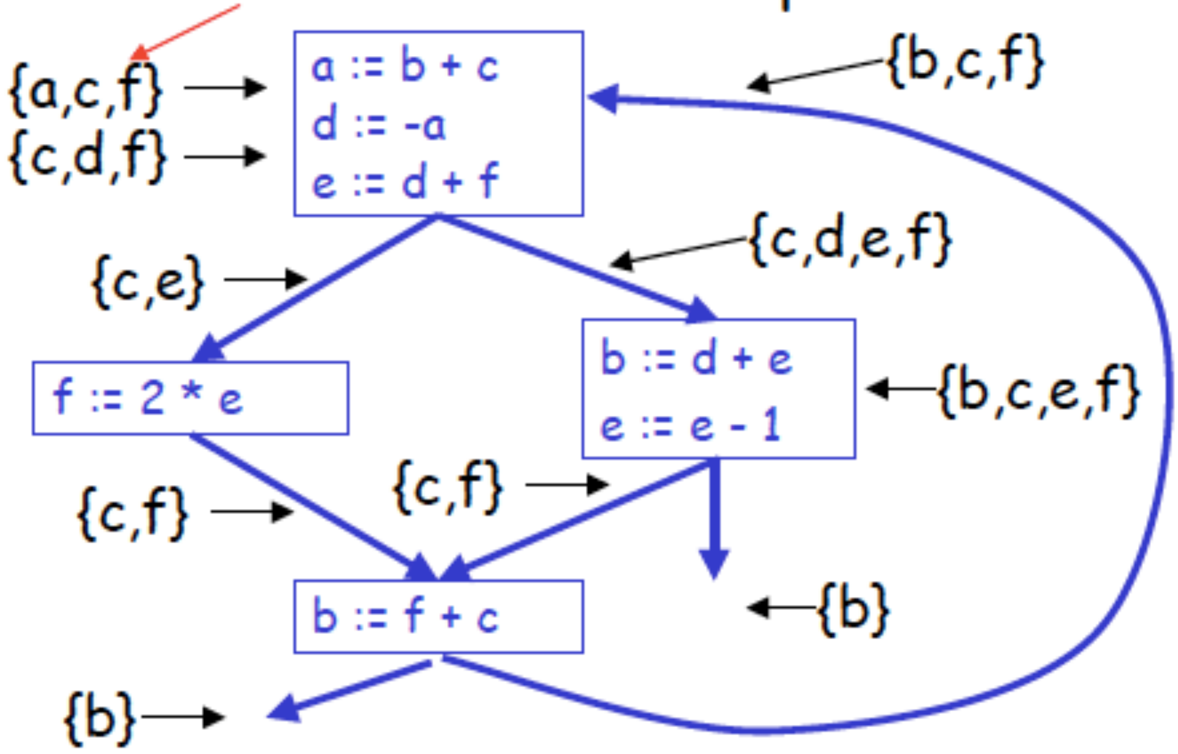


Register Interference Graph

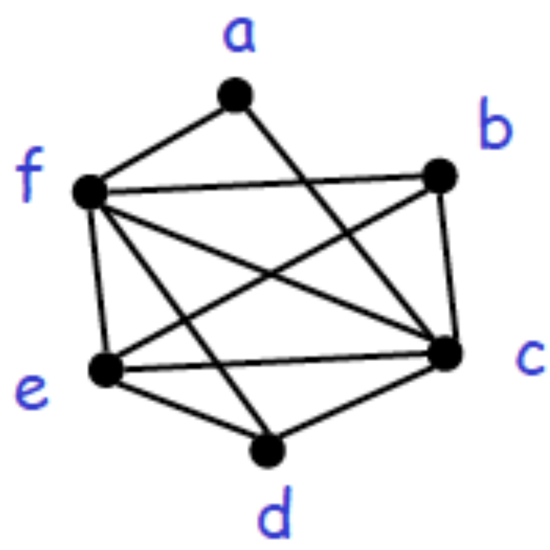
- We construct an undirected graph
  - A node for each temporary
  - An edge between t_1 and t_2 if they are live simultaneously at some point in the program
- This is the register interference graph (RIG)
  - Two temporaries can be allocated to the same register if there is no edge connecting them

What is the Register Inference Graph for this example?

- Compute live variables for each point:



Register Interference Graph



1. What **cannot** be assigned same register?
2. What **can** be assigned the same register?

Your Turn – Write down the live variables after each statement. Hint: Start at the bottom.

Instructions	Live vars
b = a + 2	
c = b * b	
b = c + 1	
return b * a	

Live Variables

Instructions	Live vars
b = a + 2	
c = b * b	
b = c + 1	
	b,a
return b * a	

Live Variables

Instructions	Live vars
b = a + 2	
c = b * b	
	a,c
b = c + 1	
	b,a
return b * a	

Live Variables

Instructions	Live vars
b = a + 2	
	b,a
c = b * b	
	a,c
b = c + 1	
	b,a
return b * a	

Live Variables

Instructions	Live vars
	a
b = a + 2	
	b,a
c = b * b	
	a,c
b = c + 1	
	b,a
return b * a	

Interference graph and Register Allocation

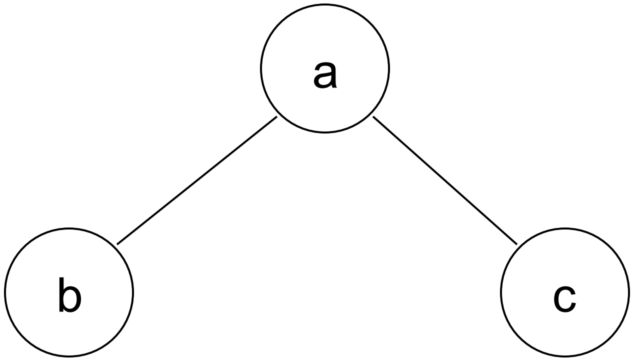
- Nodes of the graph = variables
- Edges connect variables that interfere with one another
- Nodes will be assigned a color corresponding to the register assigned to the variable
- Two colors can't be next to one another in the graph

Register Allocation = Graph Coloring

- A coloring of a graph is an assignment of colors to nodes, such that nodes connected by an edge have different colors
- A graph is k-colorable if it has a coloring with k colors

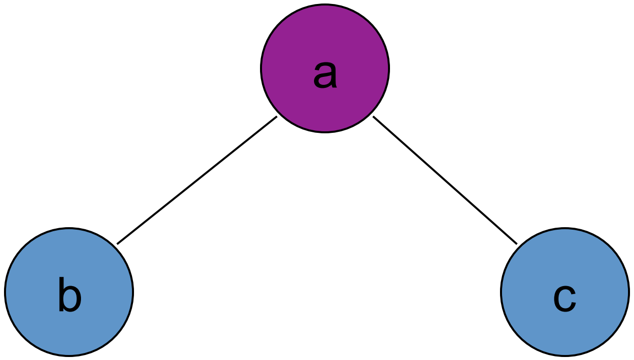
Coloring the RIG

Instructions	Live vars
	a
b = a + 2	
	a,b
c = b * b	
	a,c
b = c + 1	
	a,b
return b * a	



Coloring the RIG

Instructions	Live vars
	a
b = a + 2	
	a,b
c = b * b	
	a,c
b = c + 1	
	a,b
return b * a	



How to do the Graph coloring

- **Questions:**
  - Can we efficiently find a coloring of the graph whenever possible?
  - Can we efficiently find the **optimum coloring** of the graph?
  - How do we choose registers to avoid move instructions?
  - What do we do when there aren't enough colors (registers) to color the graph?

Coloring a graph

- Kempe's algorithm [1879] for finding a K-coloring of a graph
- Assume K=3
- Step 1 (simplify): find a node with at most K-1 edges and remove from the graph (with its edges).

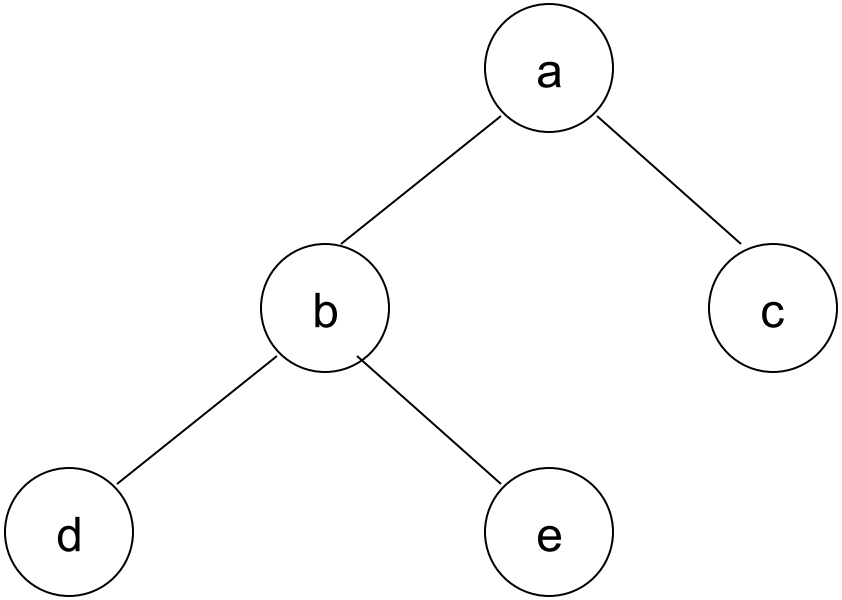
(Remember this node on a stack for later stages.)

Coloring a graph

- 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

Coloring with K=2

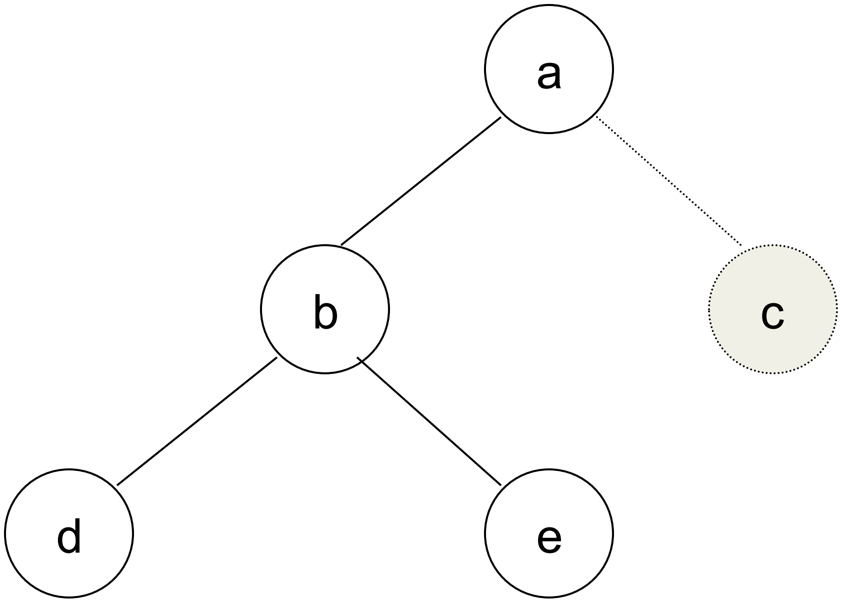
color	register
(blue)	R1
(purple)	R2



stack:

Coloring

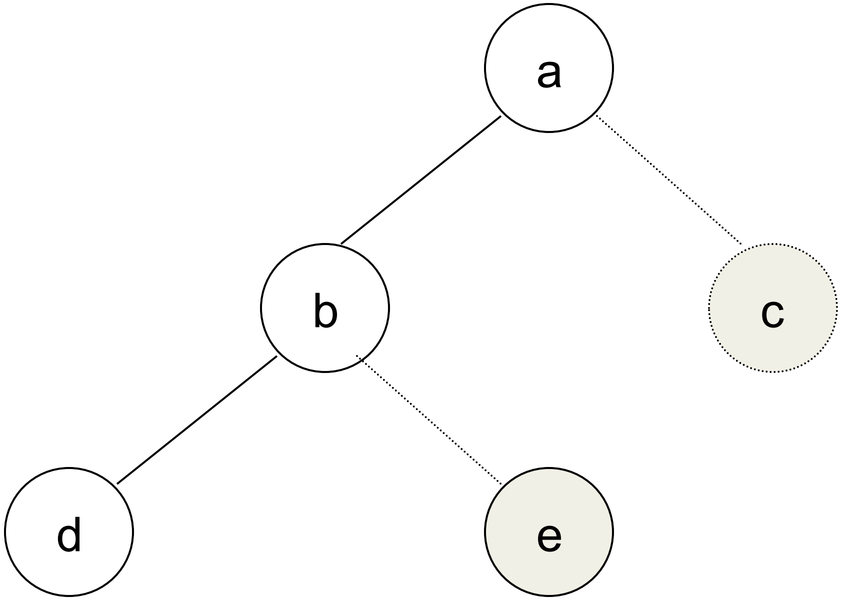
color	register
(blue)	R1
(purple)	R2



stack:

c

Coloring

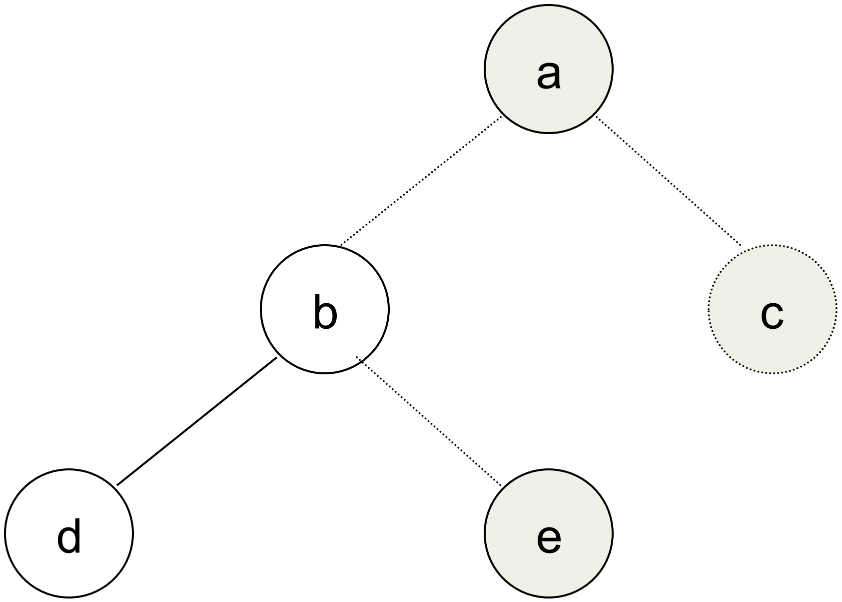


stack:

e
c

Coloring

color	register
(blue)	R1
(purple)	R2

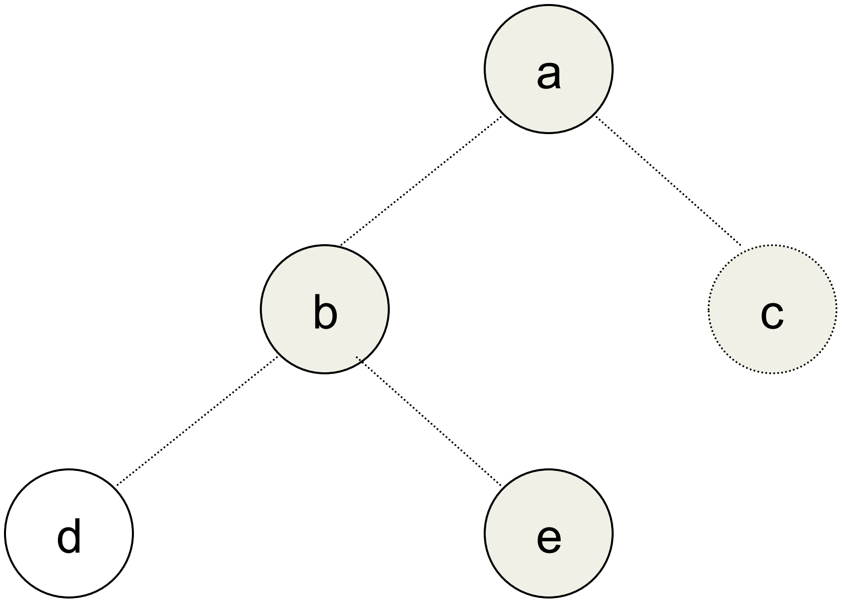


stack:

a
e
c

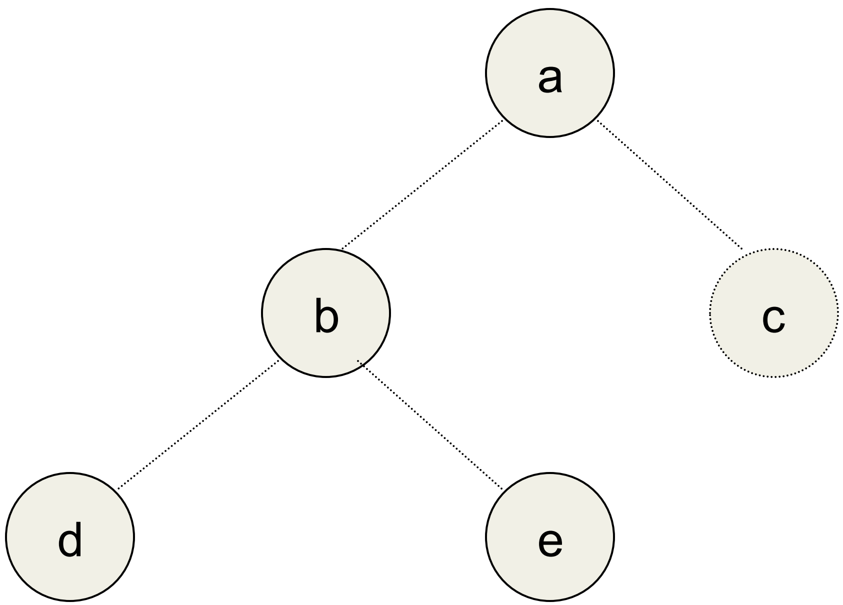
Coloring

color	register
(blue)	R1
(purple)	R2



stack:
b
a
e
c

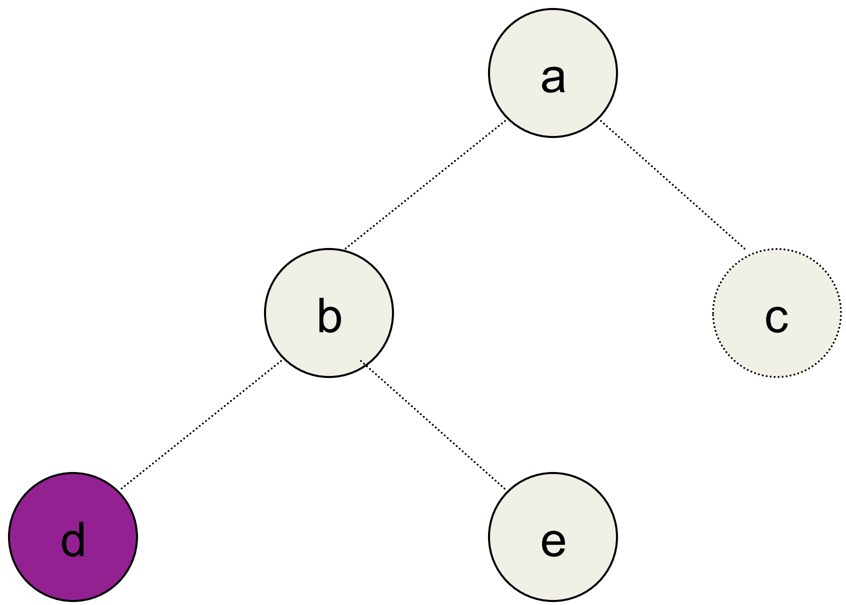
Coloring



stack:
d
b
a
e
c

Coloring

color	register
	R1
	R2

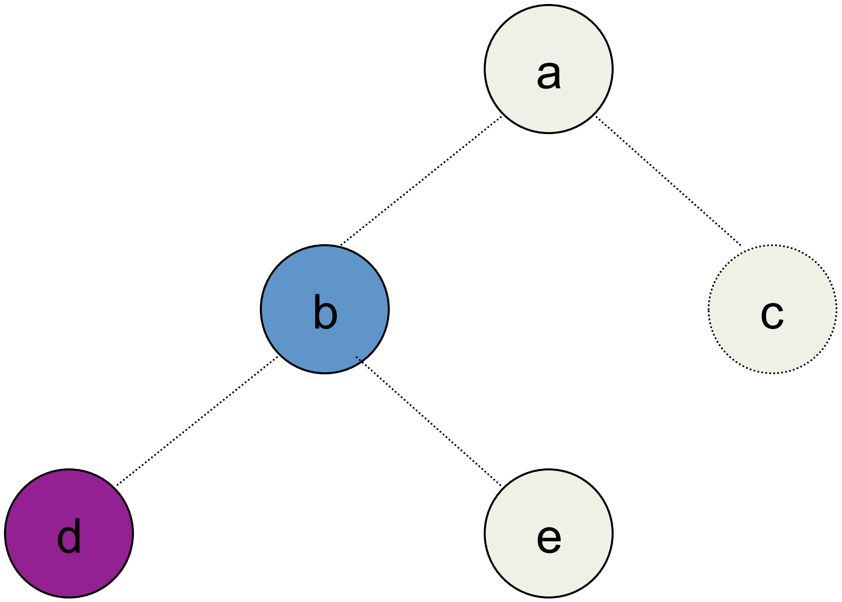


stack:

b
a
e
c

Coloring

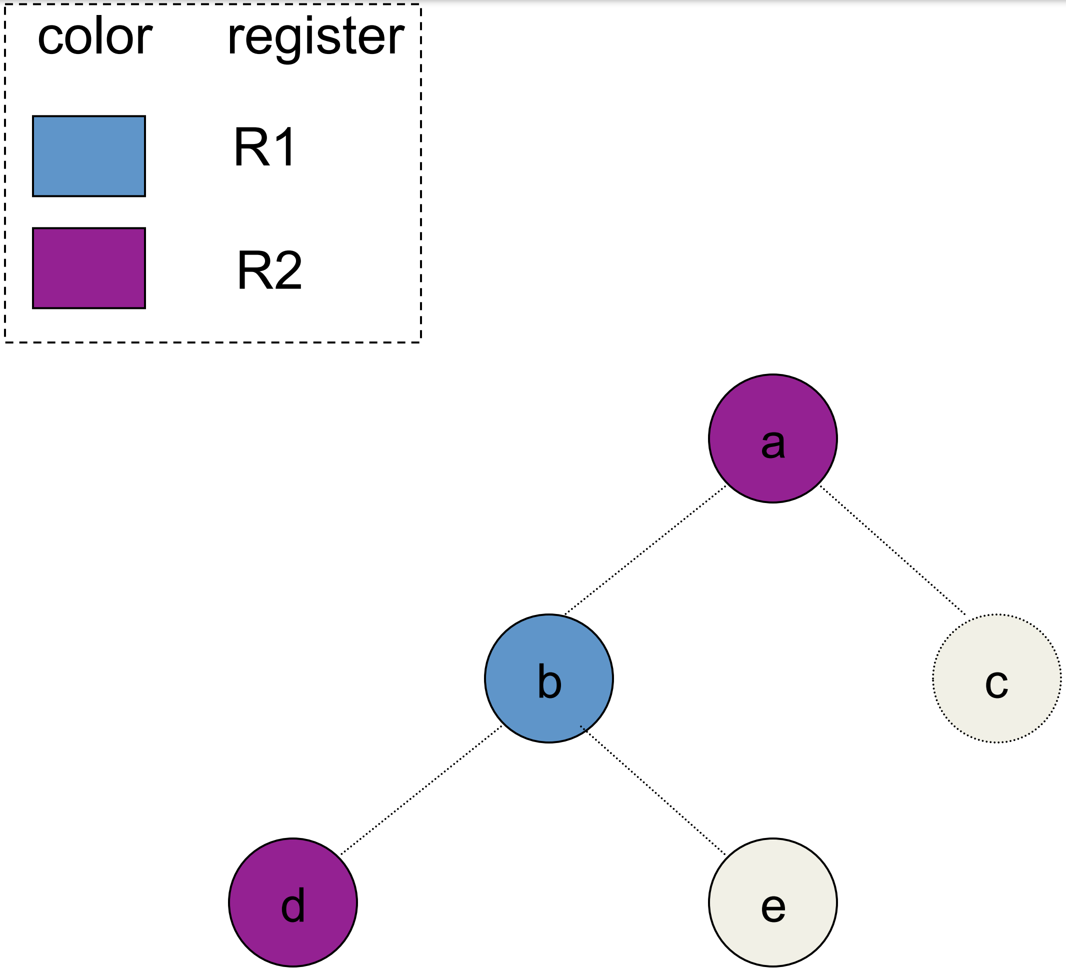
color	register
(blue)	R1
(purple)	R2



stack:

a
e
c

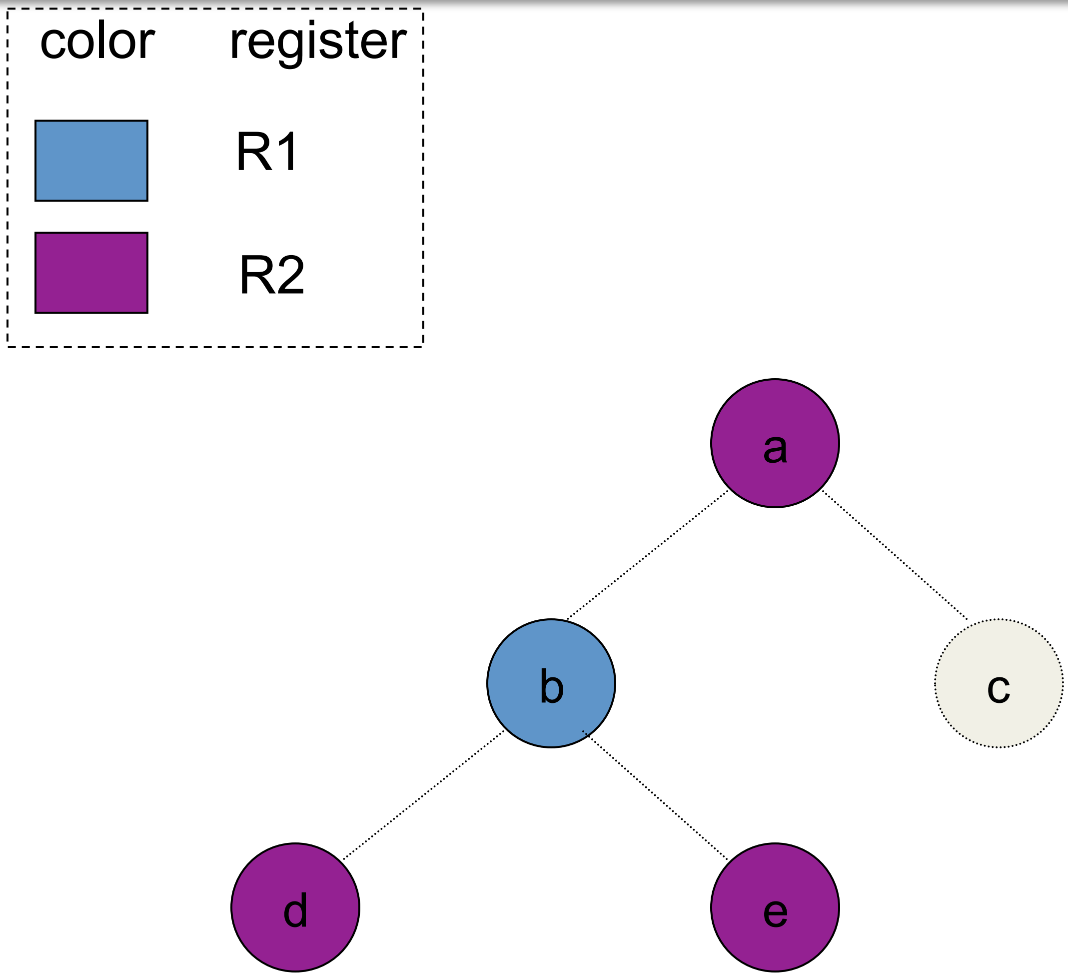
Coloring



stack:

e
c

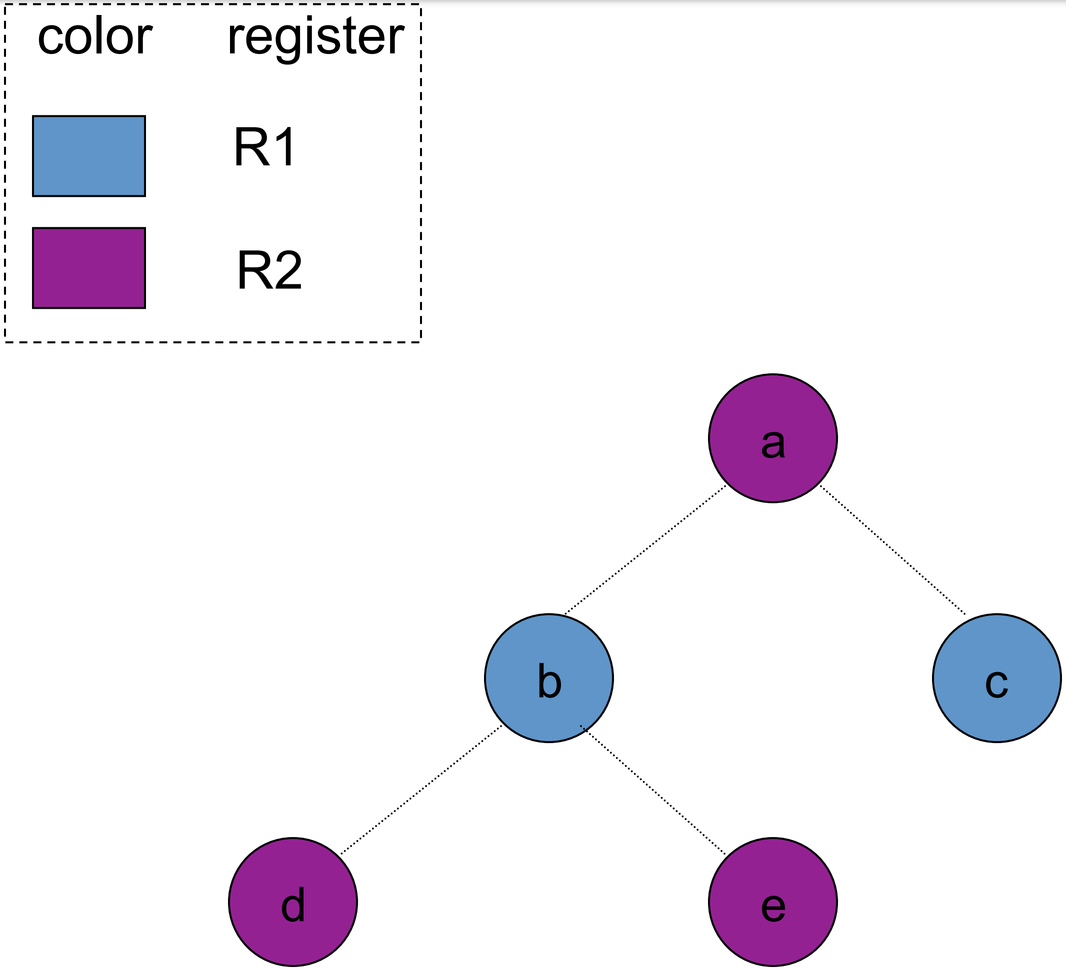
Coloring



stack:

c

Coloring



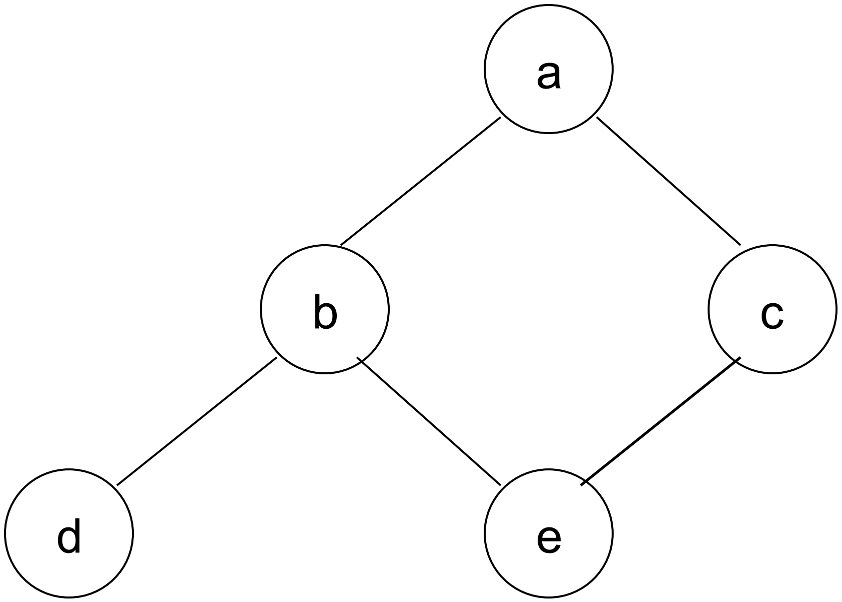
stack:

Failure

- If the graph cannot be colored, it will eventually be simplified to graph in which every node has at least K neighbors
- Sometimes, the graph is still K-colorable!
- Finding a K-coloring in all situations is an NP-complete problem
  - We will have to approximate to make register allocators fast enough

Coloring with K=2

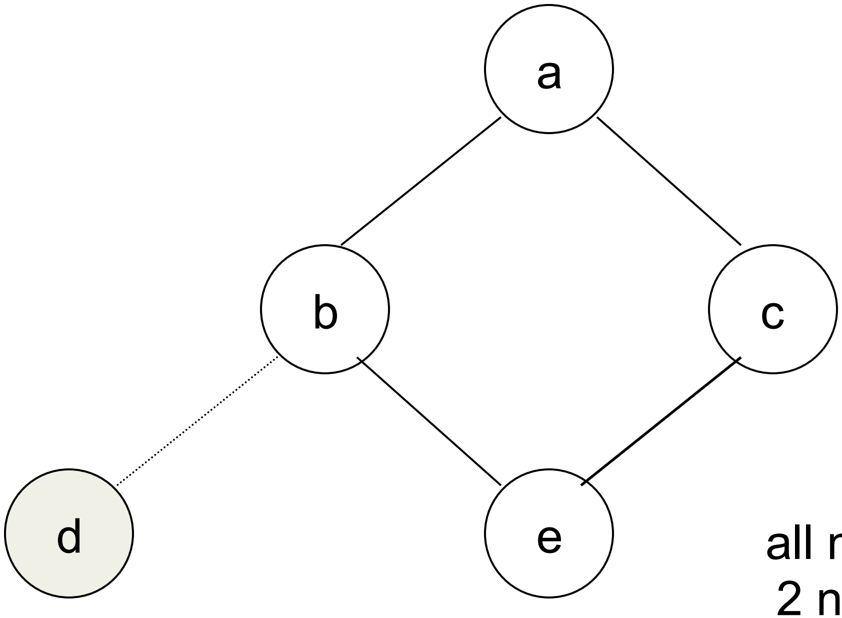
color	register
(blue)	R1
(purple)	R2



stack:

Coloring

color	register
(blue)	R1
(purple)	R2

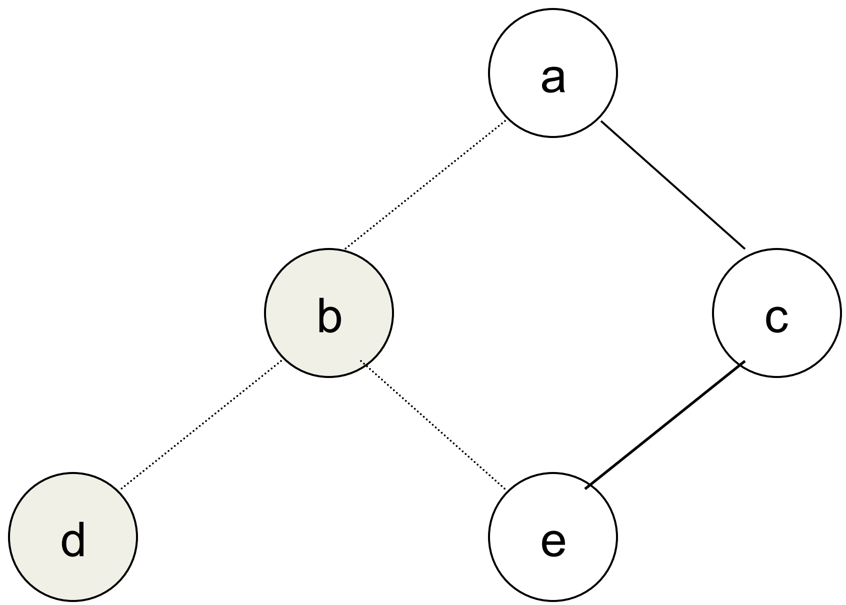


stack:
d

all nodes have 2 neighbours!

Coloring

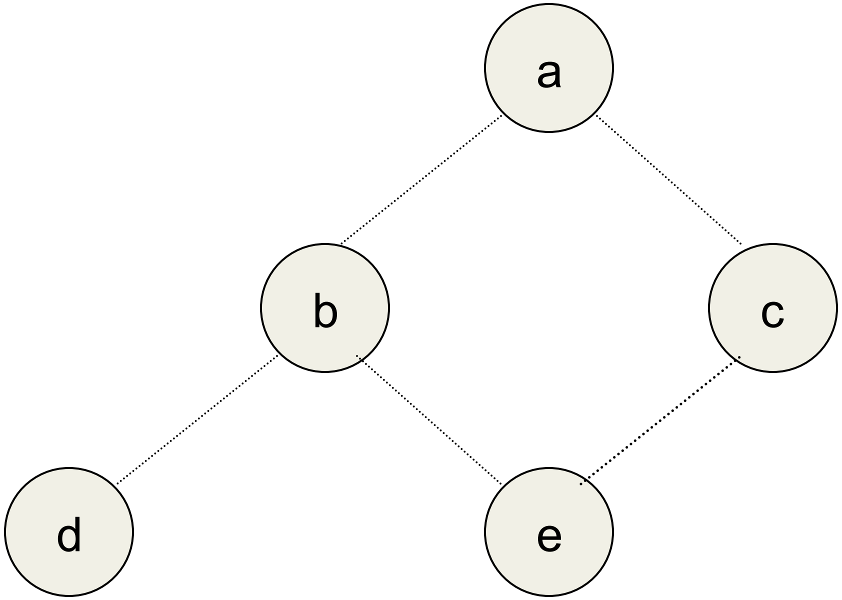
color	register
(blue)	R1
(purple)	R2



stack:

b
d

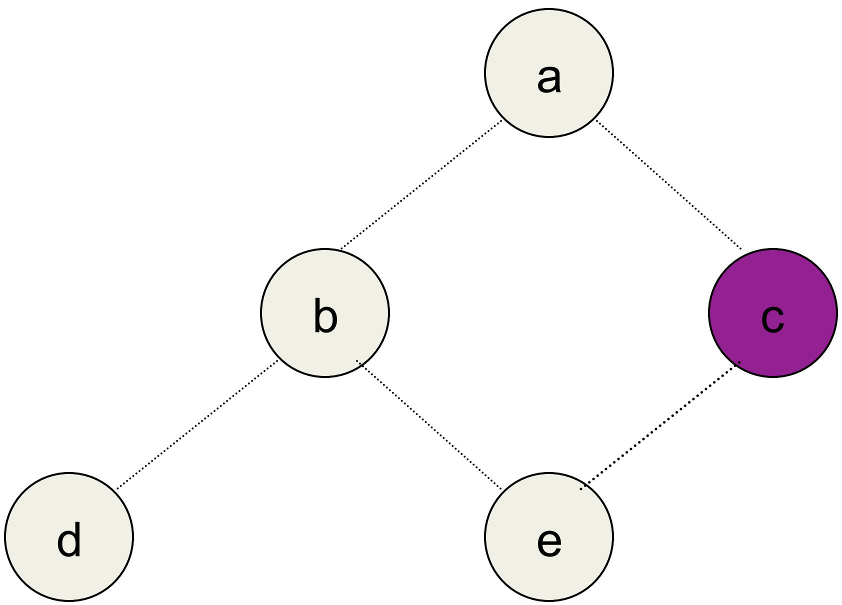
Coloring



stack:
c
e
a
b
d

Coloring

color	register
(blue)	R1
(purple)	R2

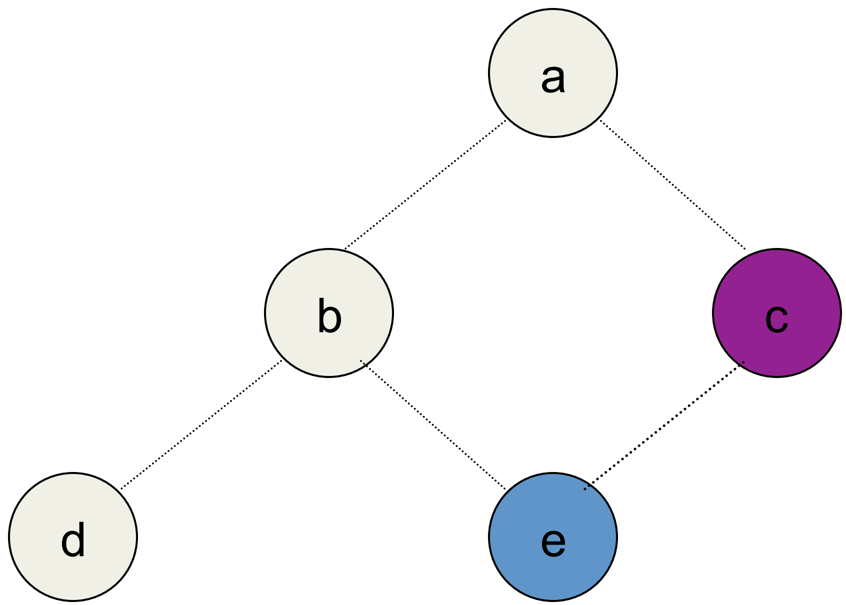


stack:

e
a
b
d

Coloring

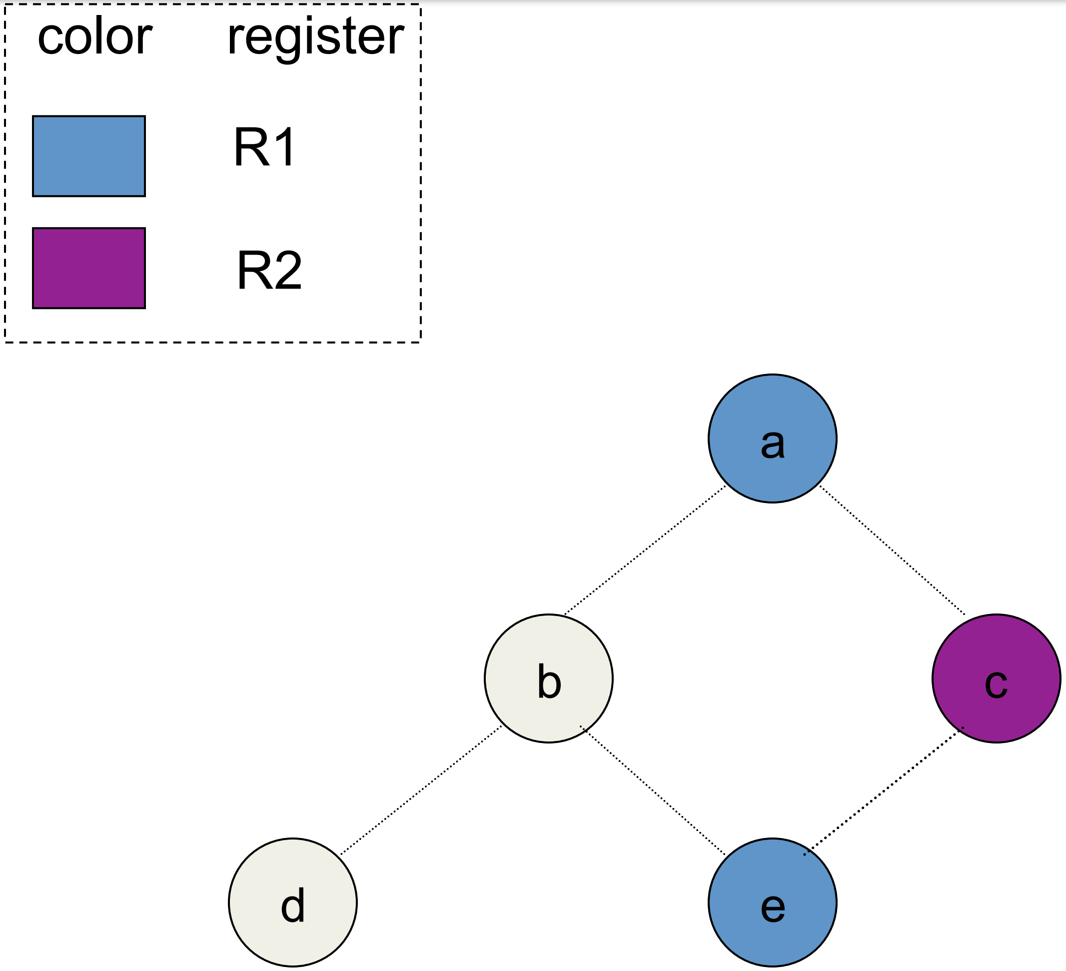
color	register
(blue)	R1
(purple)	R2



stack:

a
b
d

Coloring

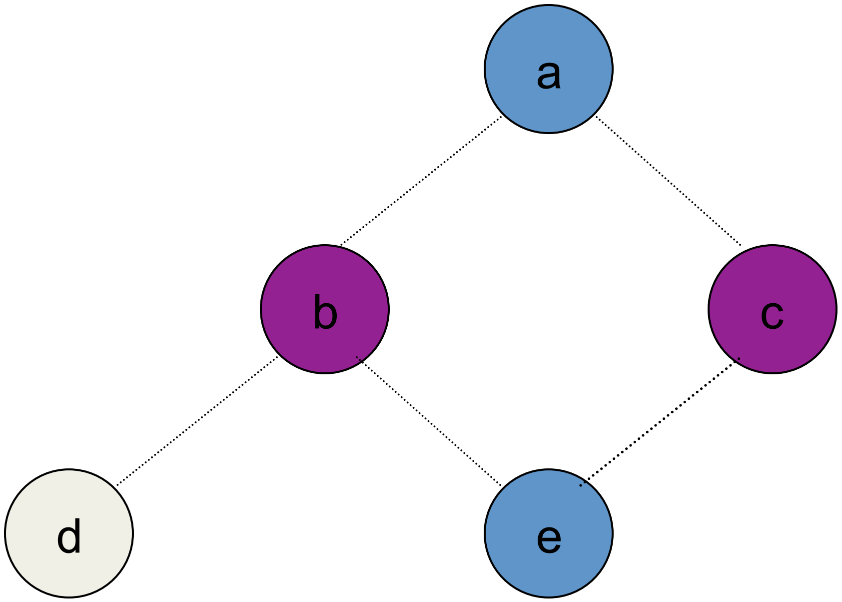


stack:

b
d

Coloring

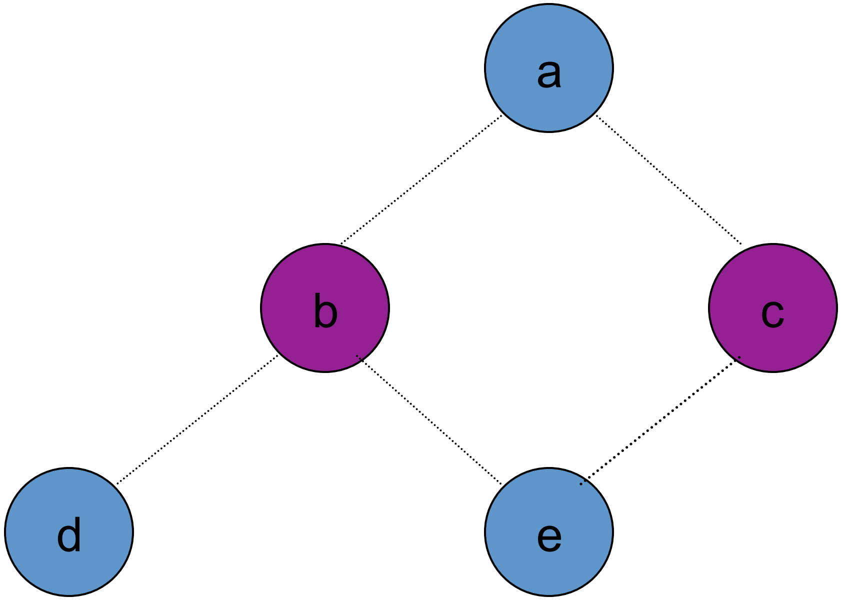
color	register
	R1
	R2



stack:

d

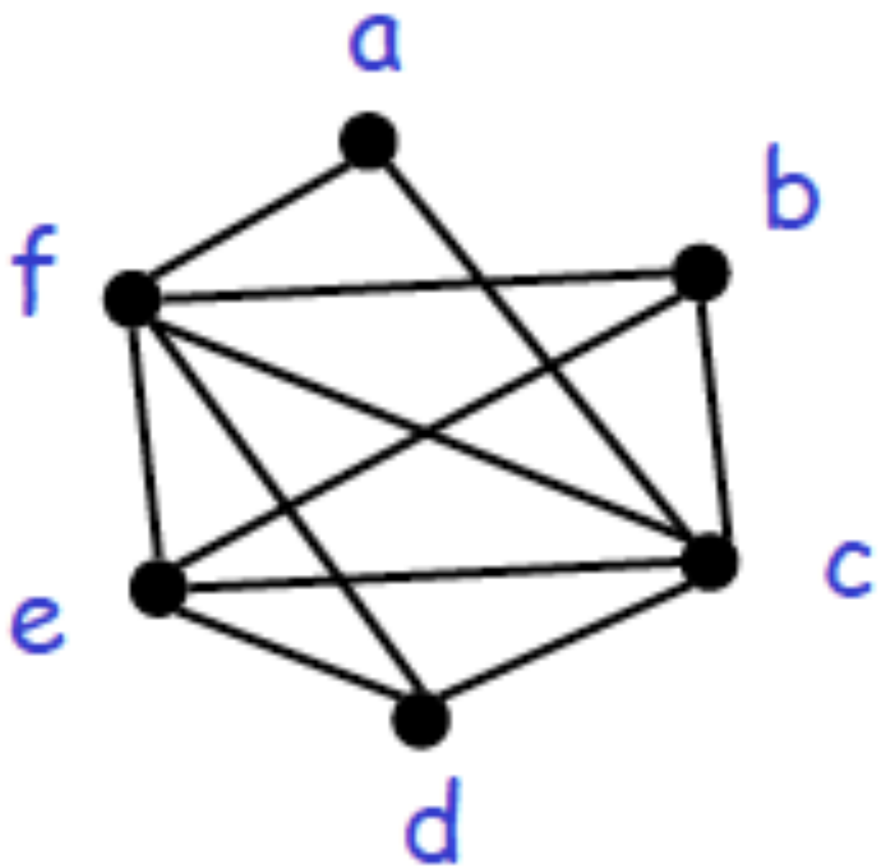
Coloring



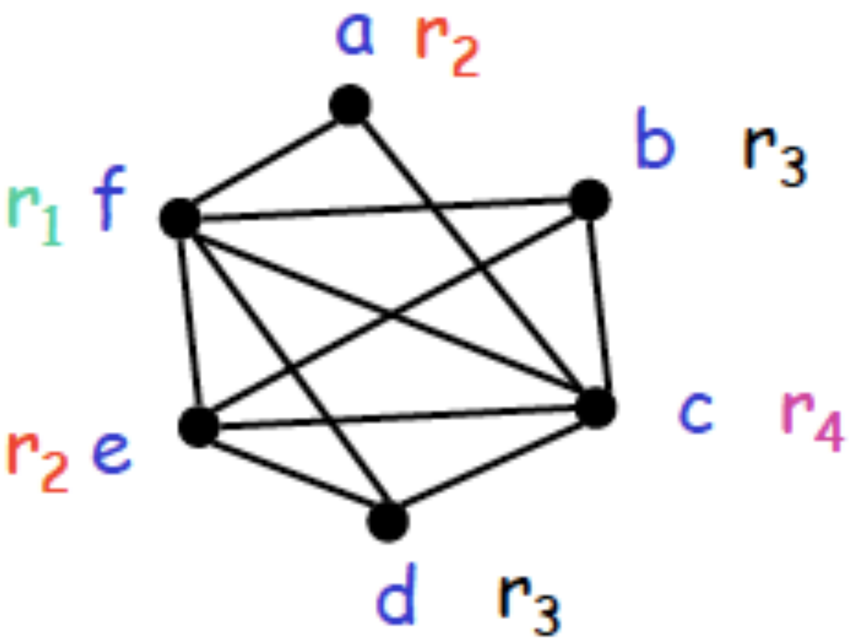
stack:

We got lucky!

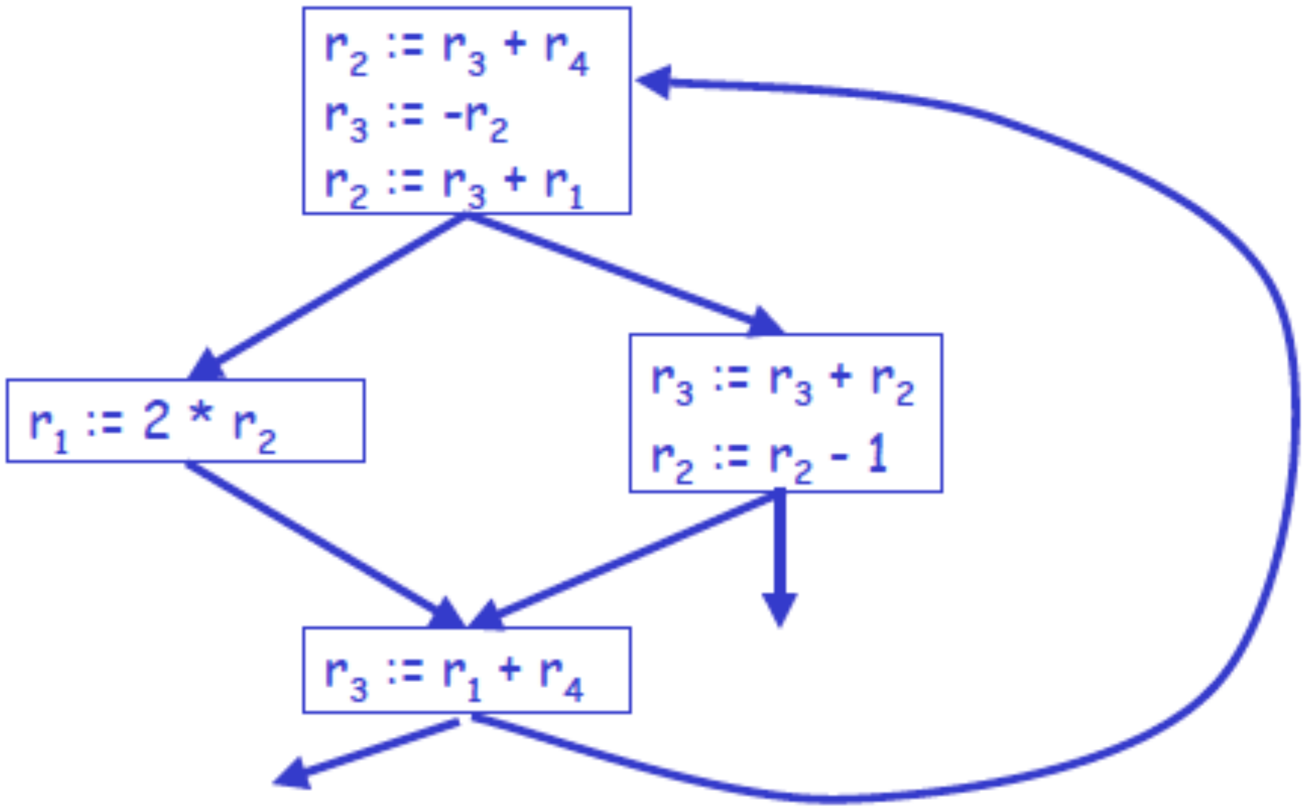
Try to Color this with 4 colors?
3 colors?



One Possible 4 coloring

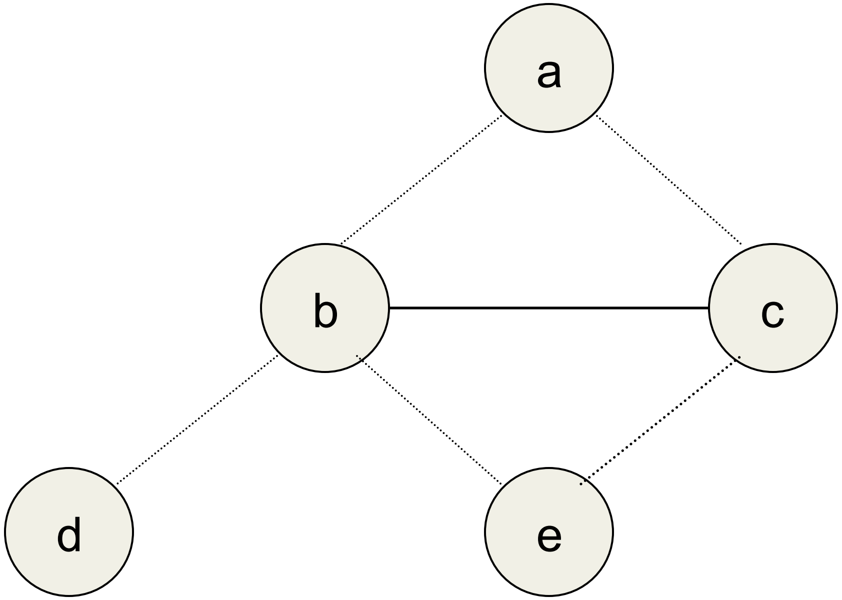


The code would look like this…



Coloring with K=2

Some graphs can't be colored in K colors:

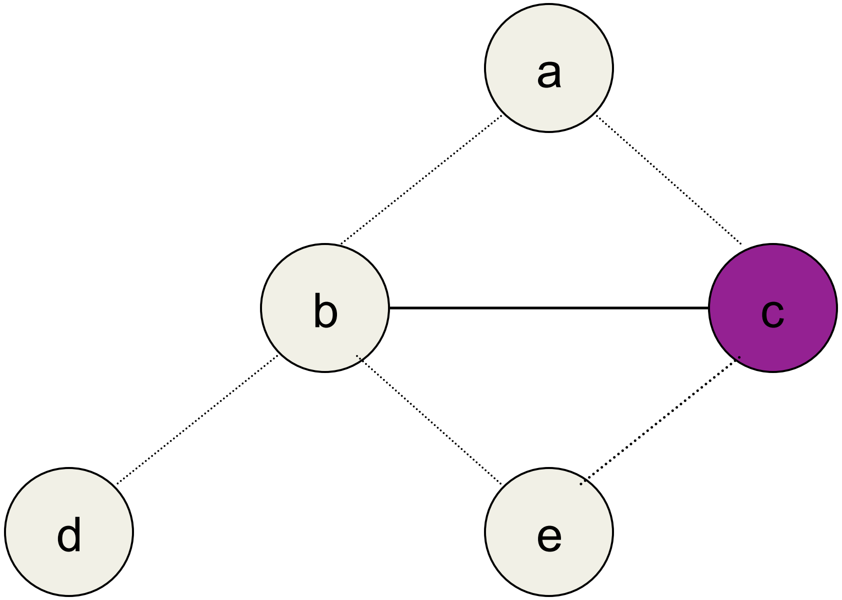


stack:

c
b
e
a
d

Coloring

Some graphs can't be colored in K colors:

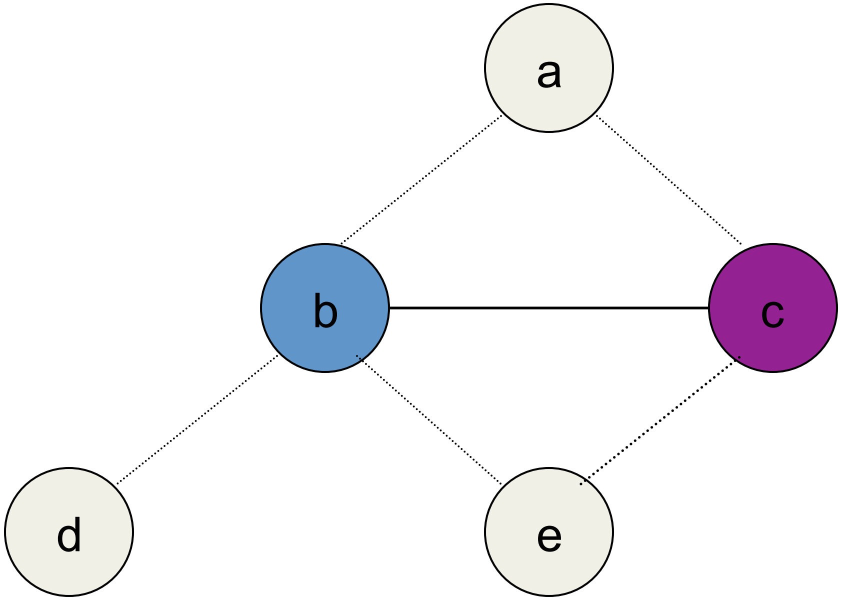


stack:

b
e
a
d

Coloring

Some graphs can't be colored in K colors:

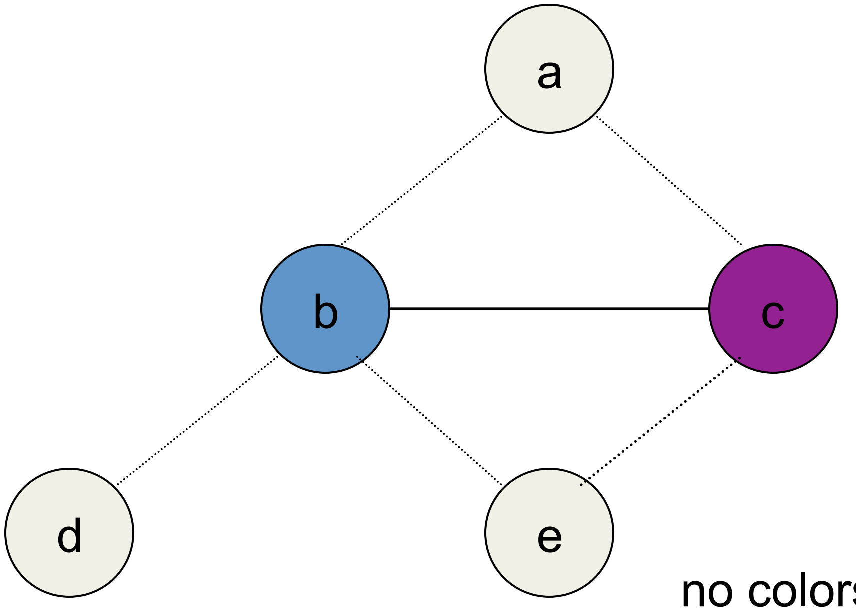


stack:

e
a
d

Coloring

Some graphs can’t be colored in K colors:



no colors left for e!

stack:

e
a
d

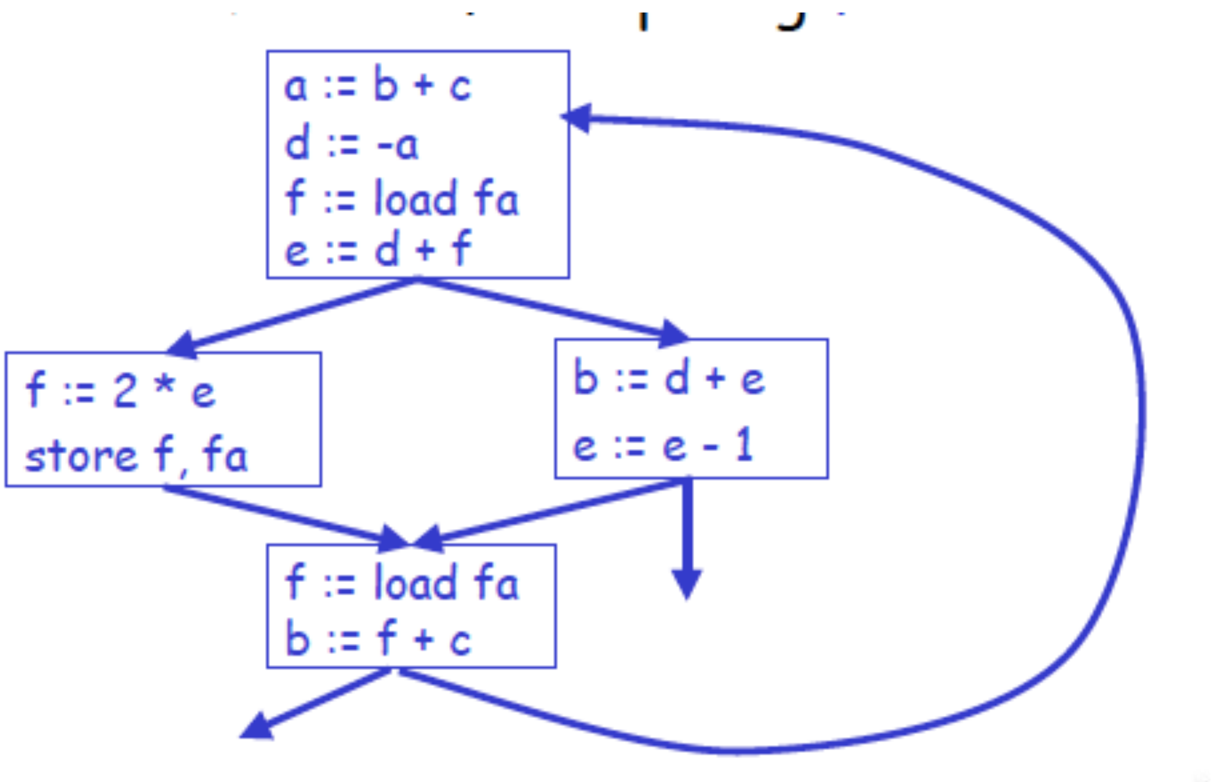
Spilling

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

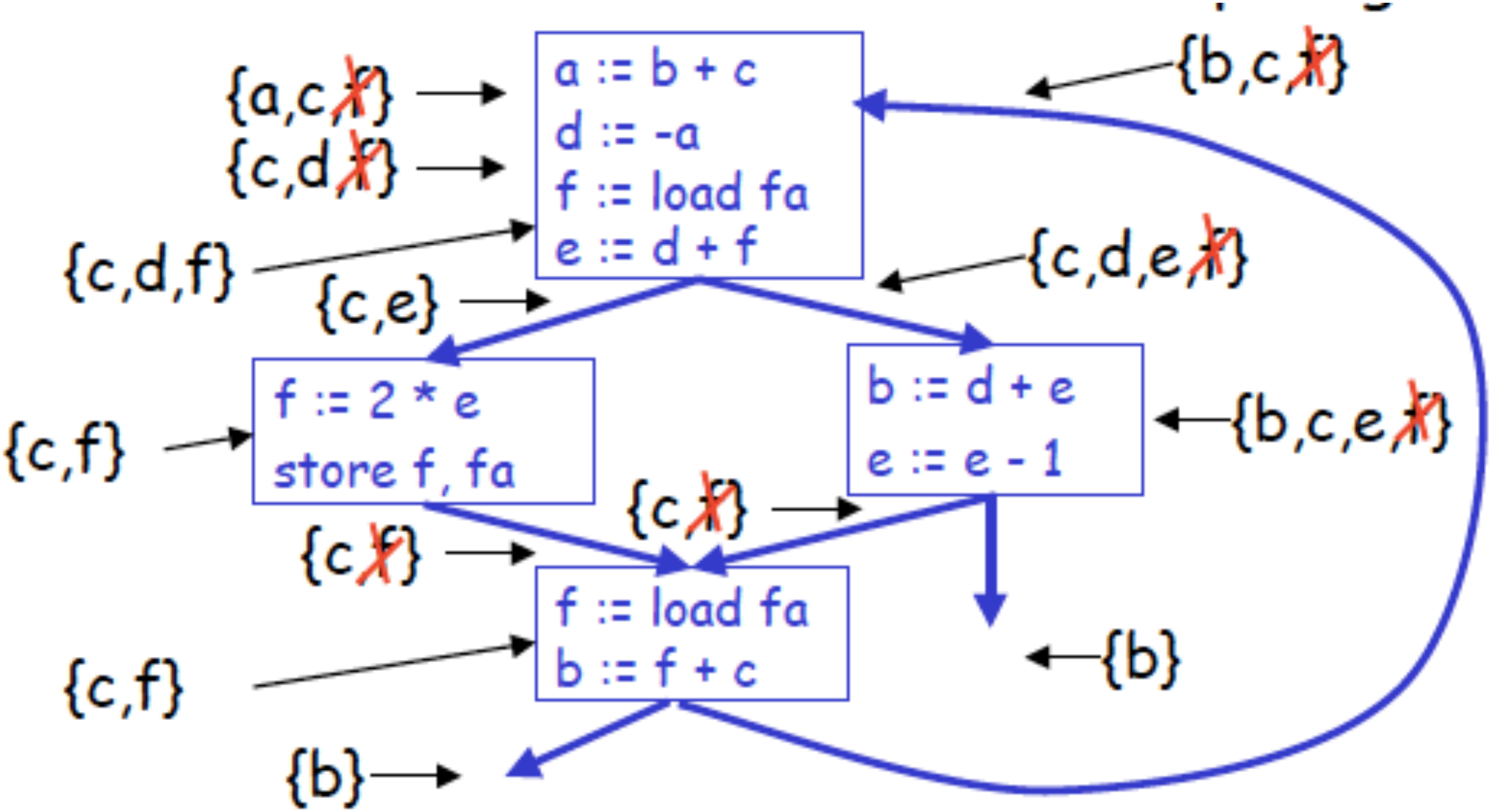
Spilling: Inserting Code

- Since optimistic coloring failed we must spill temporary f
- We must allocate a memory location as the home of f
  - Typically this is in the current stack frame
  - Call this address fa
- Before each operation that uses f, insert
  f := load fa
- After each operation that defines f, insert
  store f, fa

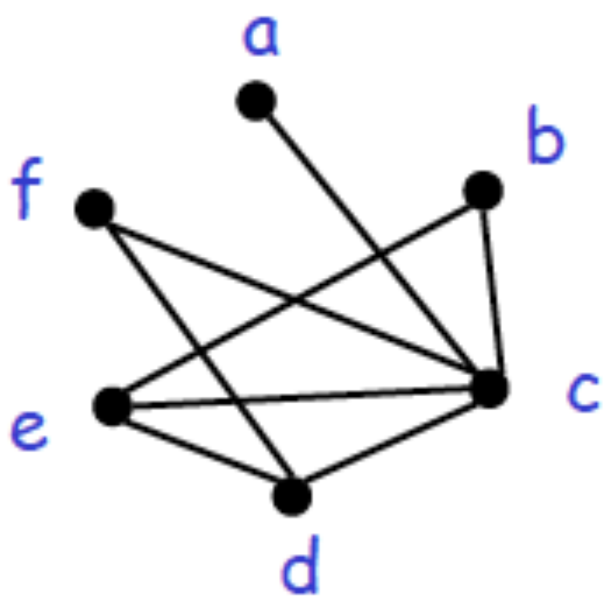
Example



Recomputing Variable Liveness

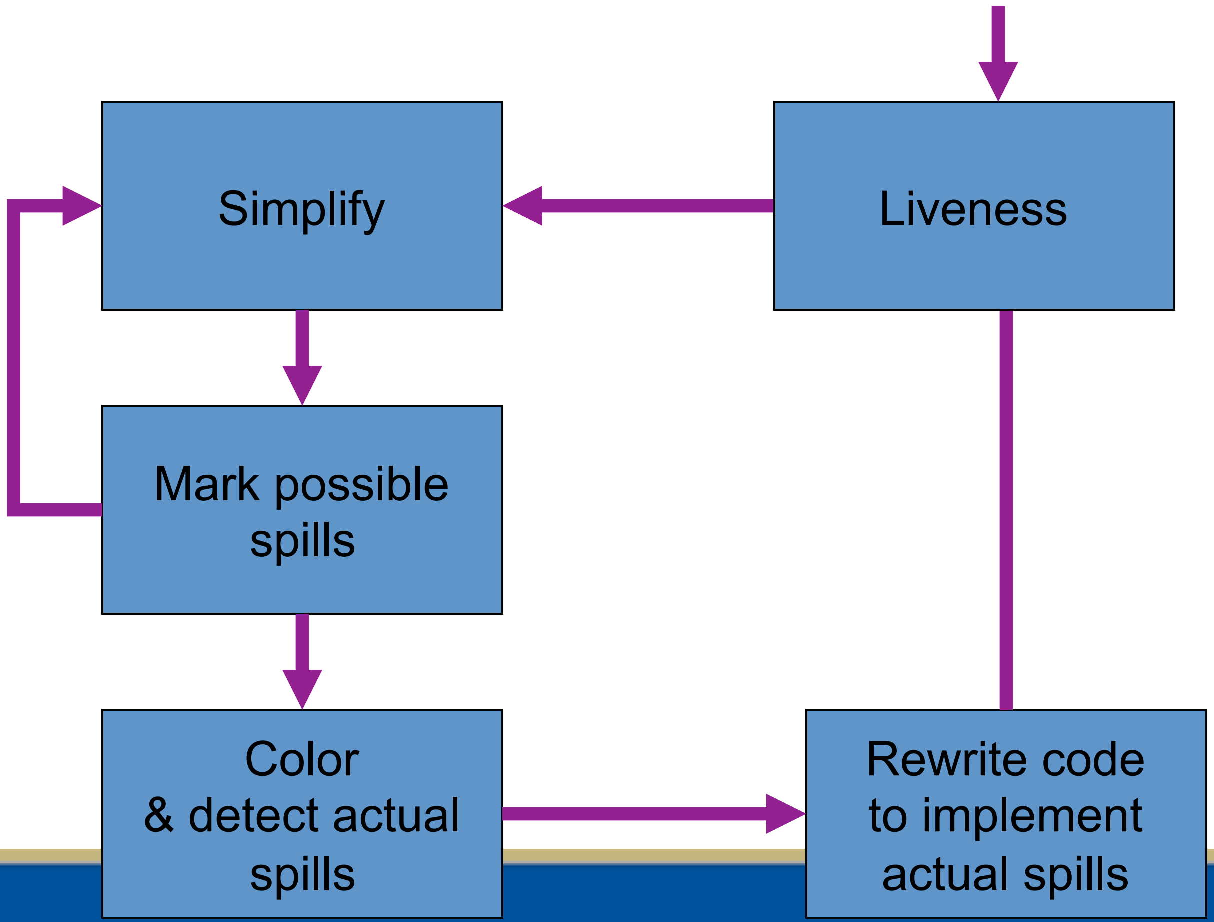


Recompute the RIG after spilling



This is 3-colorable!

Overall Algorithm



Summary

- Register allocation has three major parts
  - Liveness analysis
  - Graph coloring
  - Program transformation (spilling)
- For more information, chapter 11.1-11.3 in Appel