Lecture #34

Register Allocation & Spilling

Register Allocation via Graph Colouring

- **Graph Colouring: A**n assignment of colours to nodes, such that nodes connected by an edge have different colors
- ***k*-coloring :** A coloring using at most *k* colours.
- **Chromatic number:** The smallest number of colours needed to colour a graph
- **Independent set:** A subset of vertices assigned to the same colour

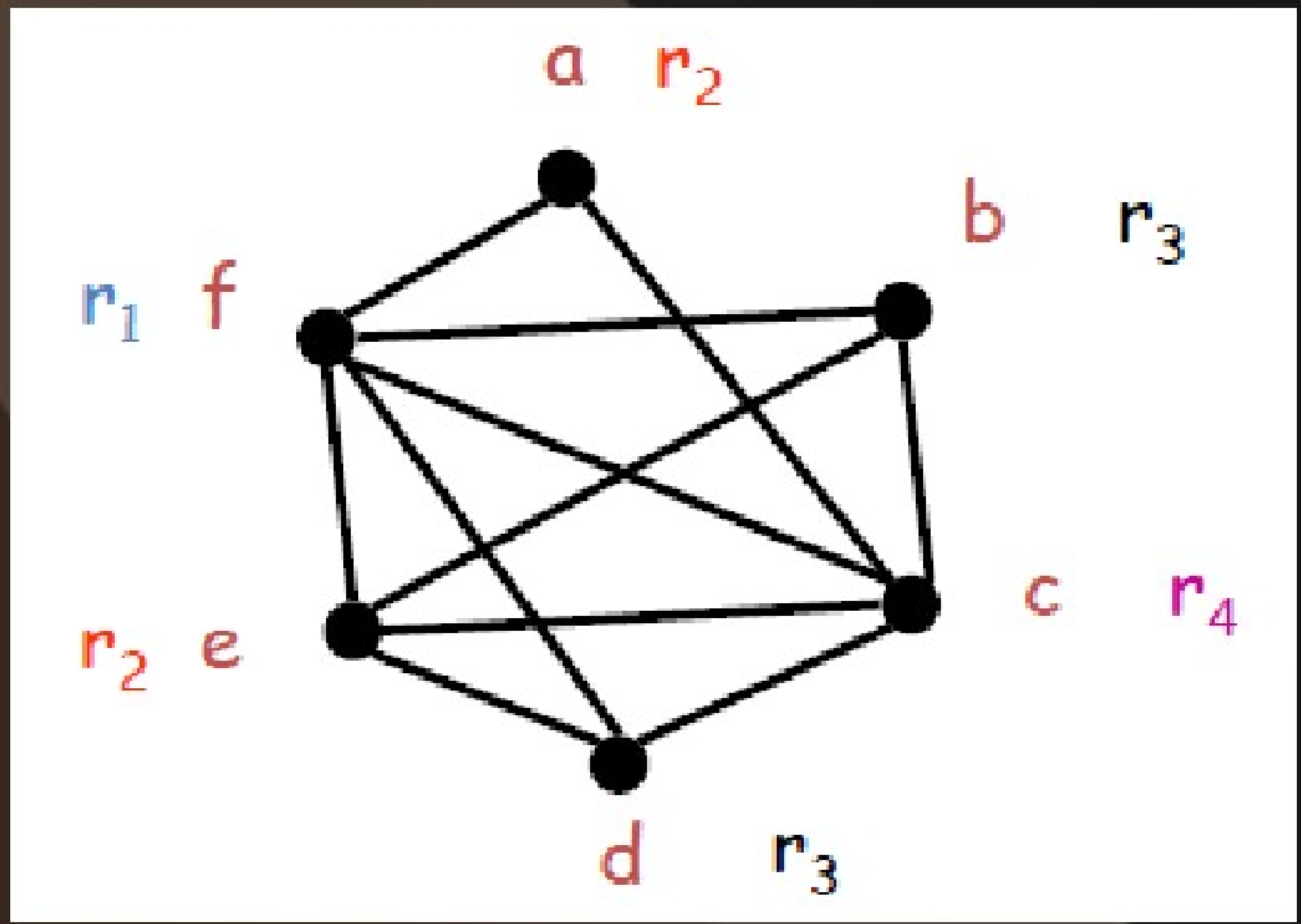
- *k*-coloring is the same as a partition of the vertex set into *k* independent sets
    - The terms *k-partite* and *k-colourable* are equivalent

- *The graph colouring problem is* ***NP-Hard***
    - Heuristics needed to solve it

Register Allocation via Graph Colouring

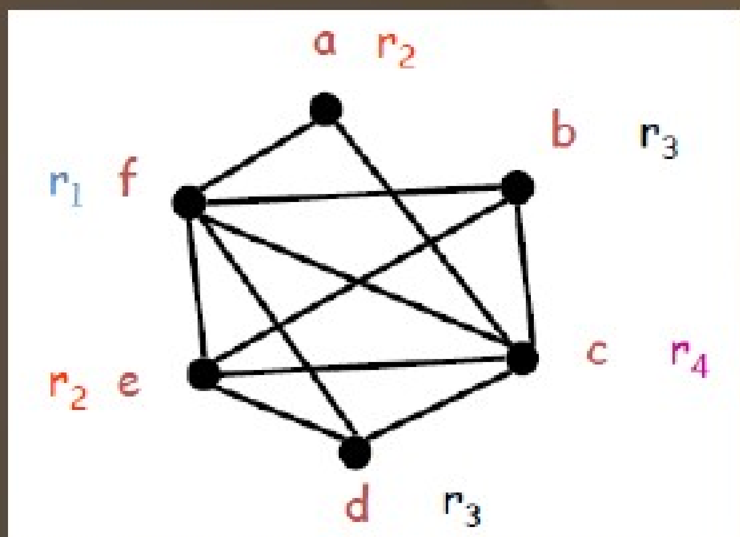
- In the register allocation problem, colours = registers
- We need to assign colours (registers) to graph nodes (temporaries)
- Let *k* = number of machine registers
- If the RIG is *k-colourable* then there is a register assignment that uses no more than *k* registers

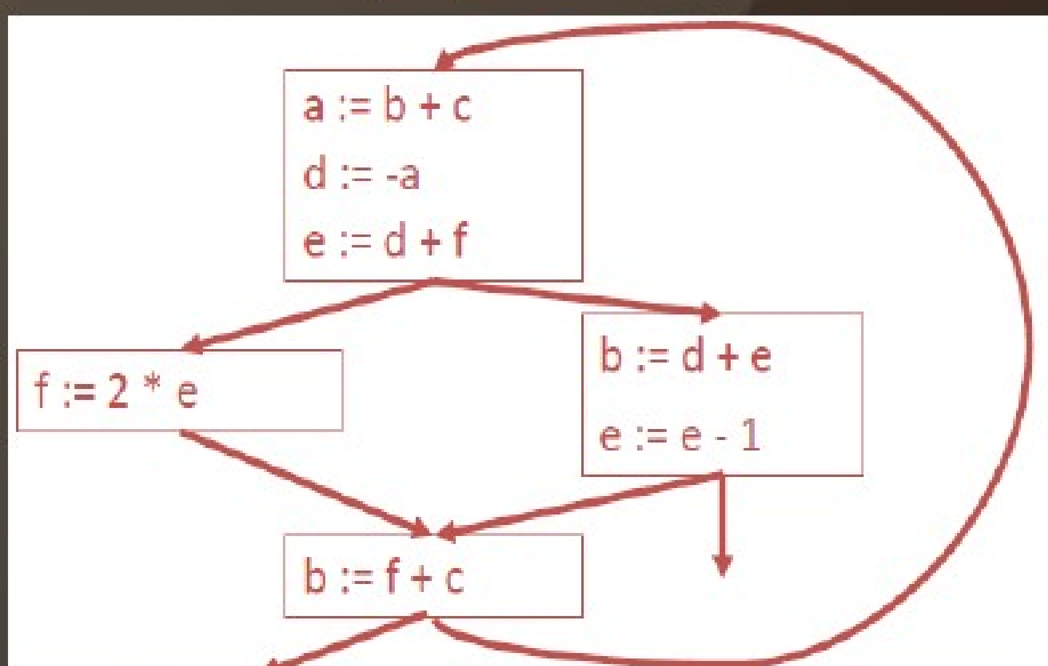
In our example RIG there is no coloring with less than 4 colours

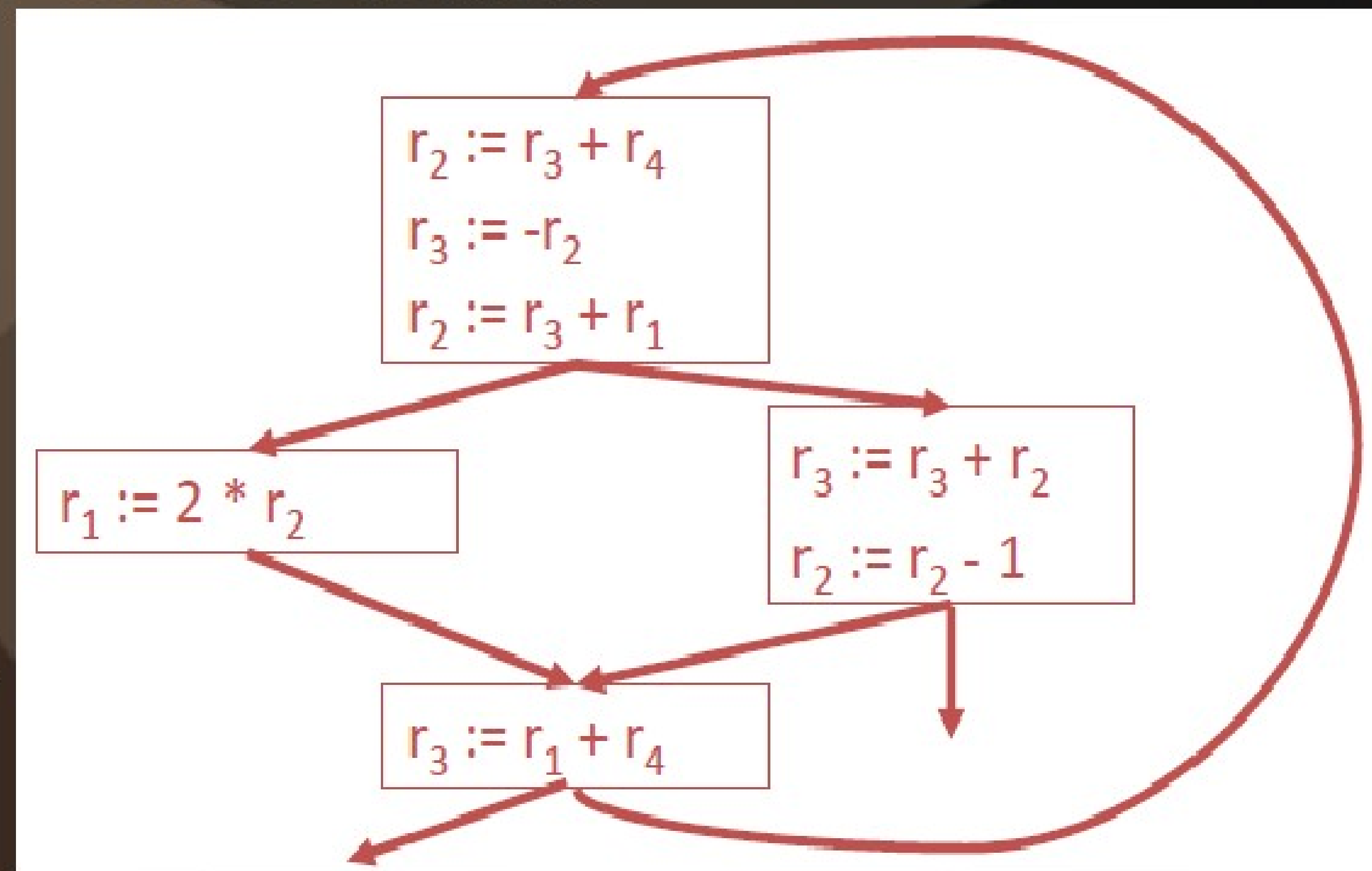


Register Allocation via Graph Colouring

- Under the colouring, the code becomes:







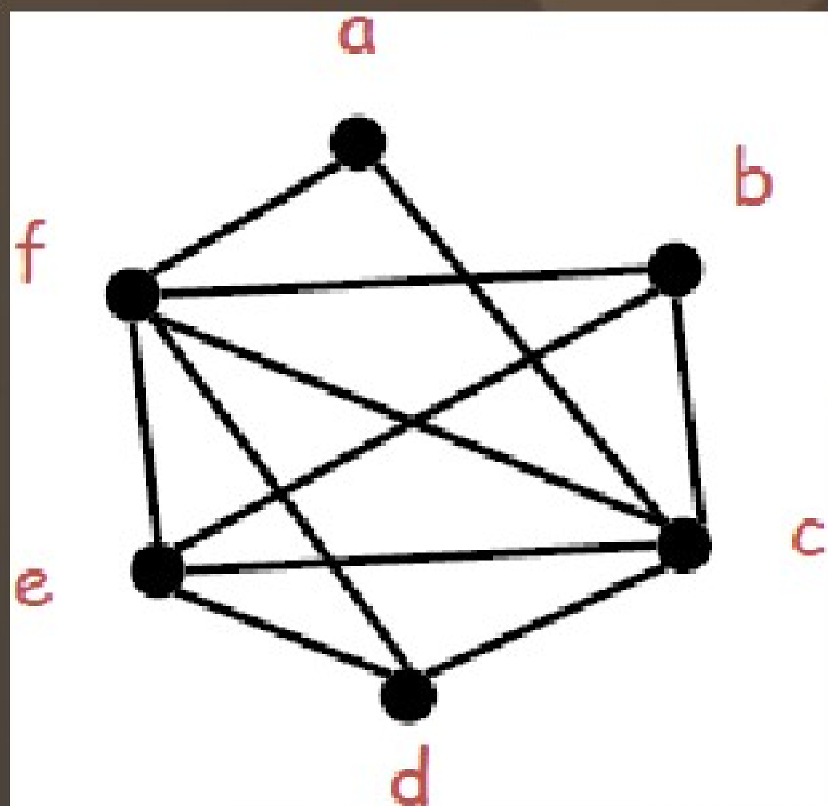
Register Allocation via Graph Colouring

- A heuristic algorithm:
  - Pick a node t with fewer than k neighbors
  - Put t on a stack and remove it from the RIG
  - Repeat until the graph is empty
- Assign colors to nodes on the stack:
  - Start with the last node added
  - At each step pick a color different from those assigned to already colored neighbors

Register Allocation via Graph Colouring

- Example: Let $k = 4$
- Initial RIG:

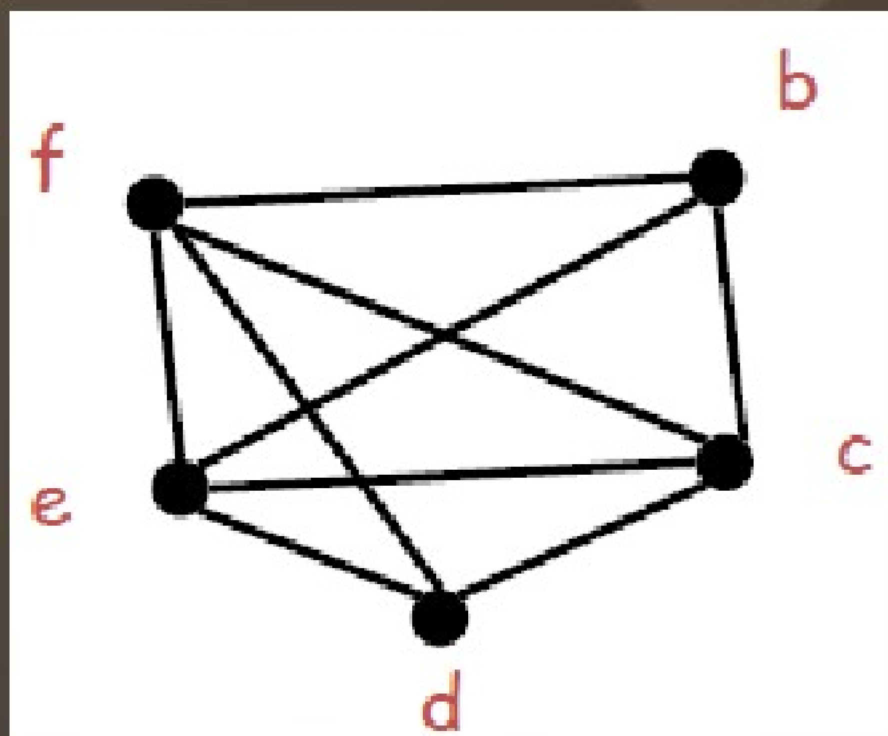
Stack: { }



Remove *a*

Register Allocation via Graph Colouring

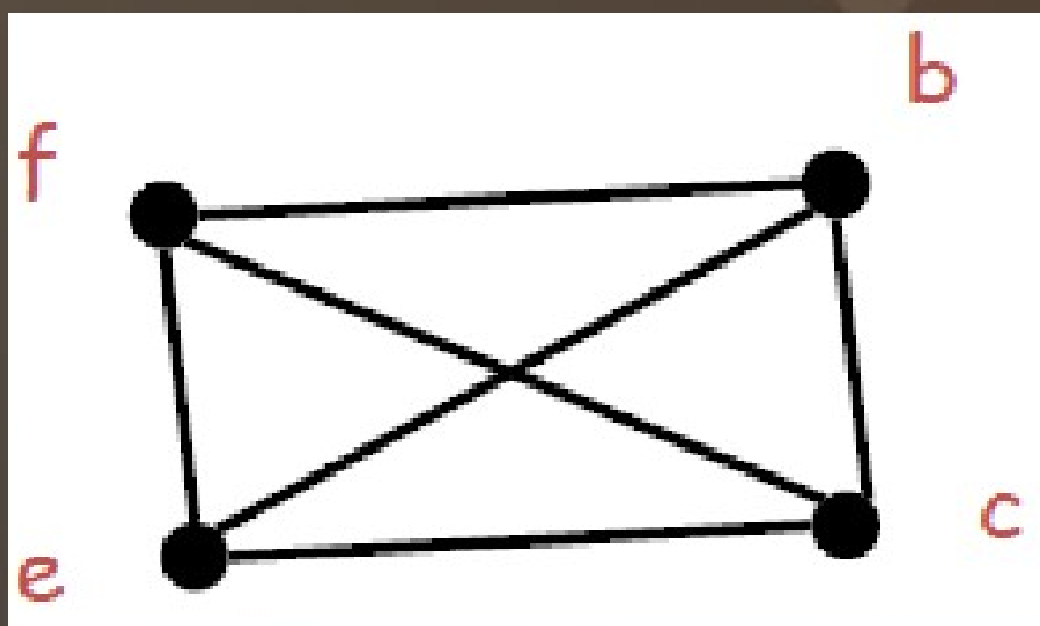
- Step 2: Stack: {a}



Remove *d*

Register Allocation via Graph Colouring

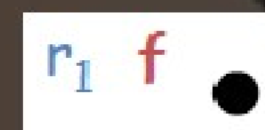
- All nodes now have fewer than 4 nodes
- Step 3: Stack: $\{d, a\}$



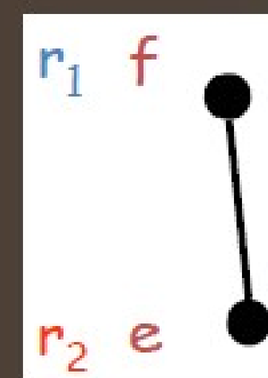
- Remove *any node*
- Continue removing nodes until the graph is *empty*
- Let the stack be: $\{f, e, b, c, d, a\}$ after removal of all nodes

Register Allocation via Graph Colouring

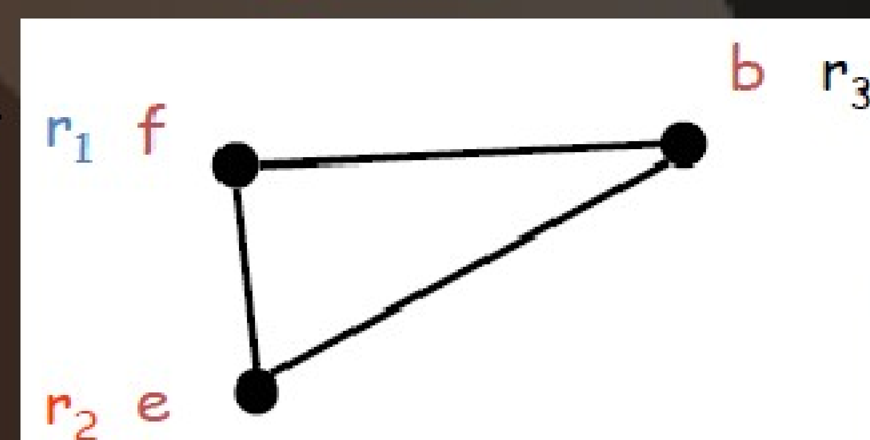
- Now start assigning colours to the nodes, starting from the top of the stack
- Stack: $\{f, e, b, c, d, a\}$
- Stack: $\{e, b, c, d, a\}$



- Stack: $\{b, c, d, a\}$

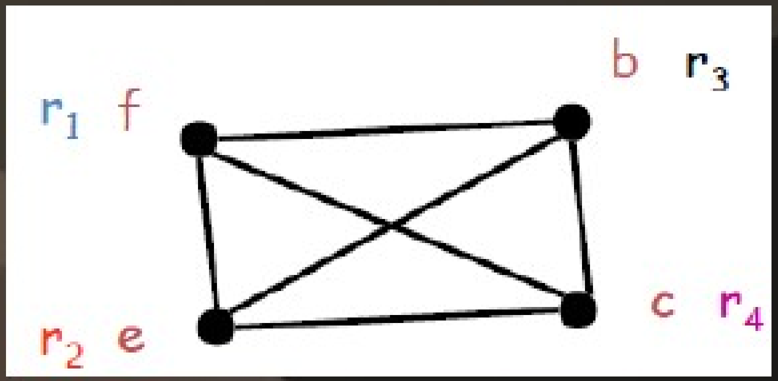


- Stack: $\{c, d, a\}$

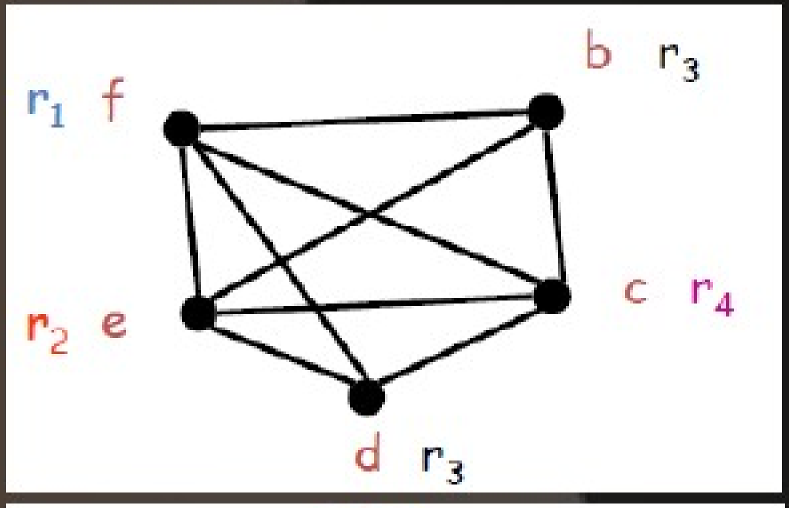


Register Allocation via Graph Colouring

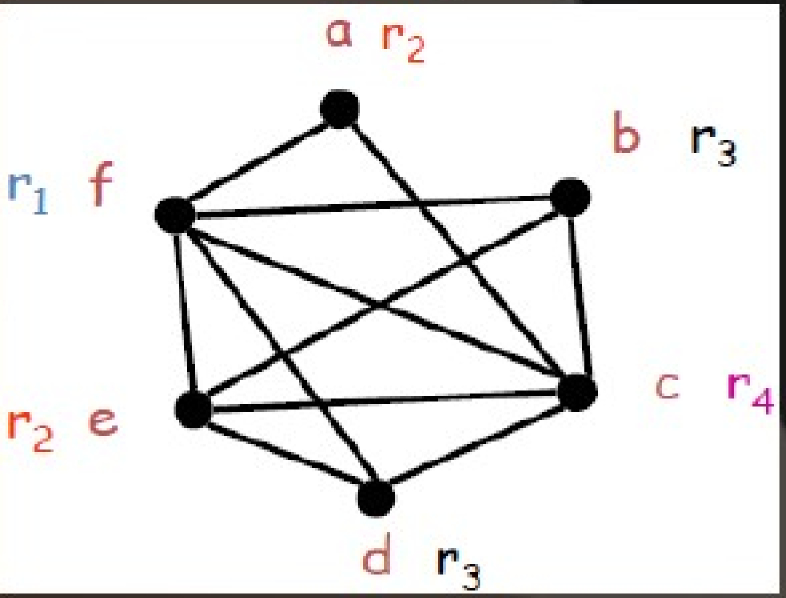
- Stack: {d, a}



- Stack: {a}(d and b can have the same register)

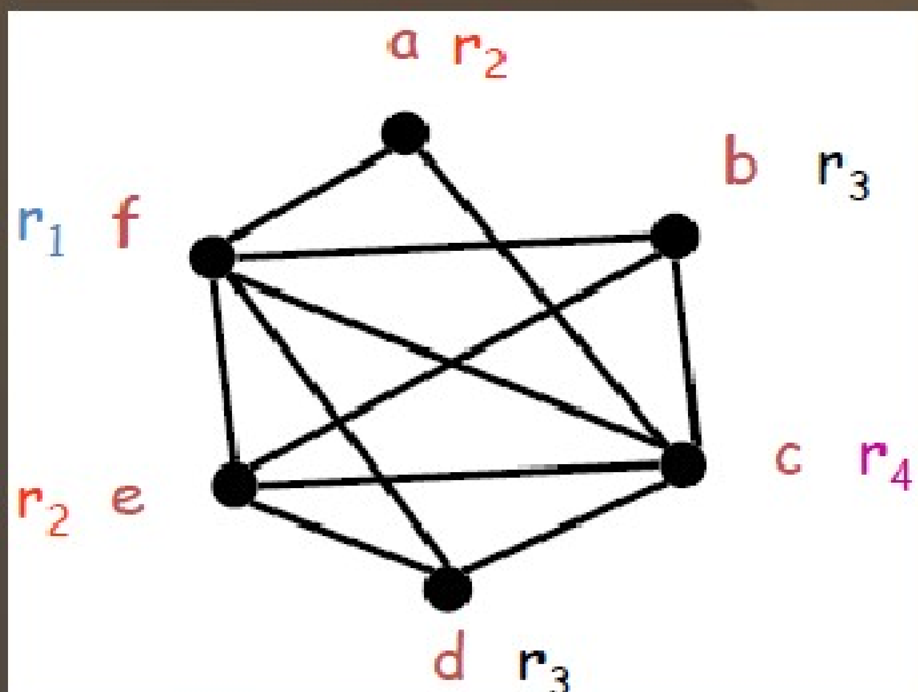


- Stack: { }(a and e can have the same register)

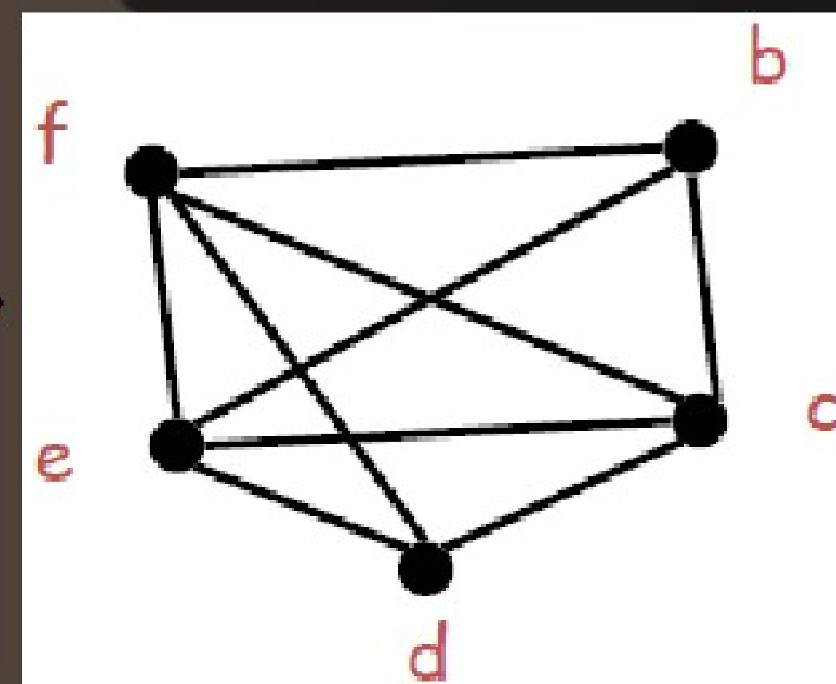


What if the heuristic fails?

- Example: Try to do a 3-colouring of the graph:

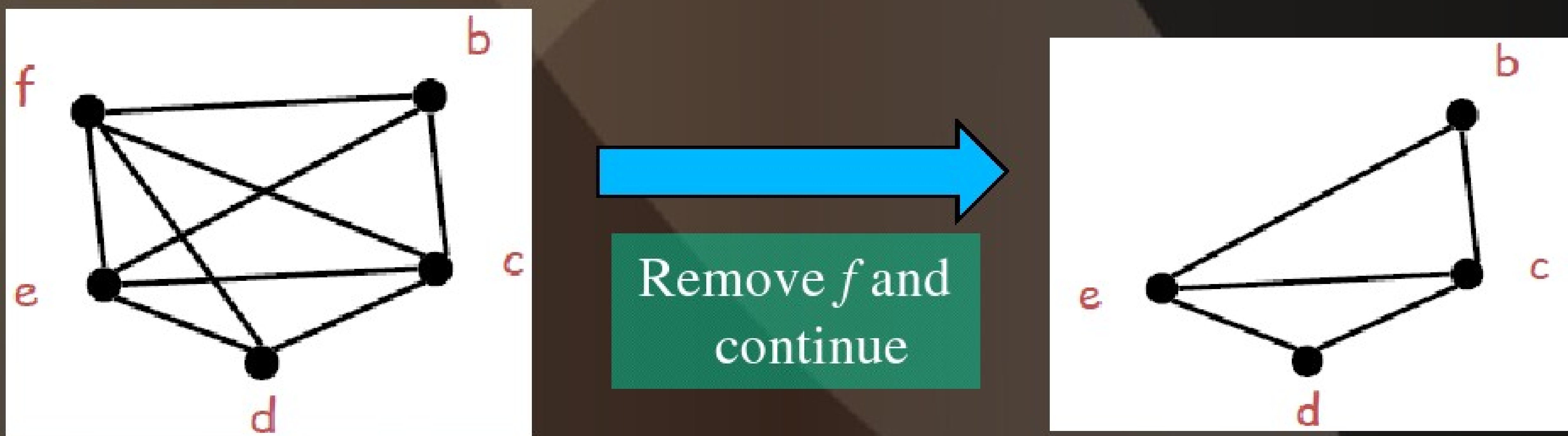


Remove *a* and get stuck



What if the heuristic fails?

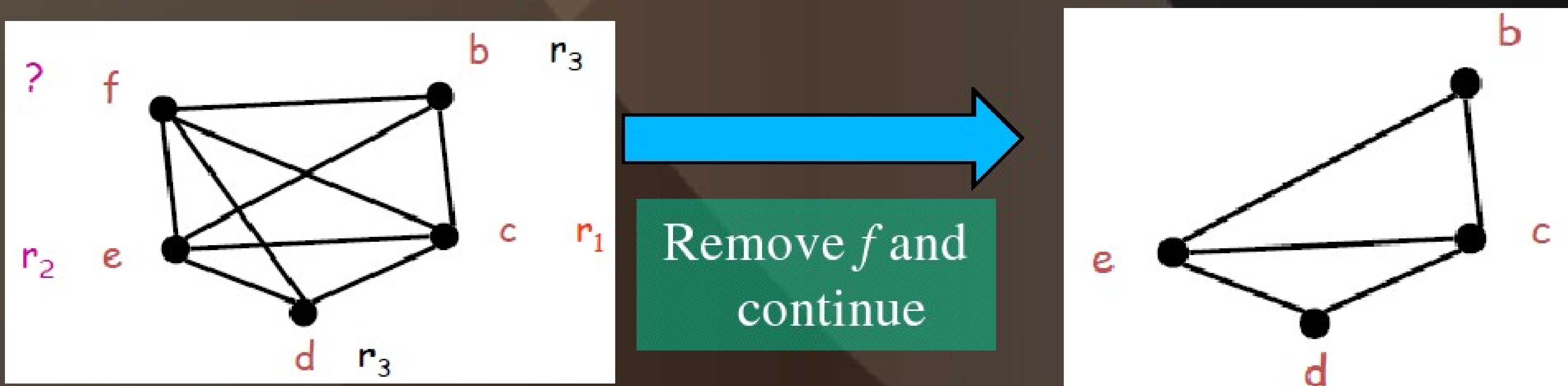
- Pick a node as a candidate for *spilling*
  - *A spilled temporary lives in memory*
- Assume we choose *f* as a candidate for spilling



- The algorithm now succeeds: *b*, *d*, *e*, *c*

What if the heuristic fails?

- On the assignment phase we get to the point when we have to assign a color to *f*
- We hope that among the 4 neighbors of *f* we use less than 3 colors $\Rightarrow$ <u>optimistic coloring</u>



- The algorithm now succeeds: *b*, *d*, *e*, *c*

Spilling

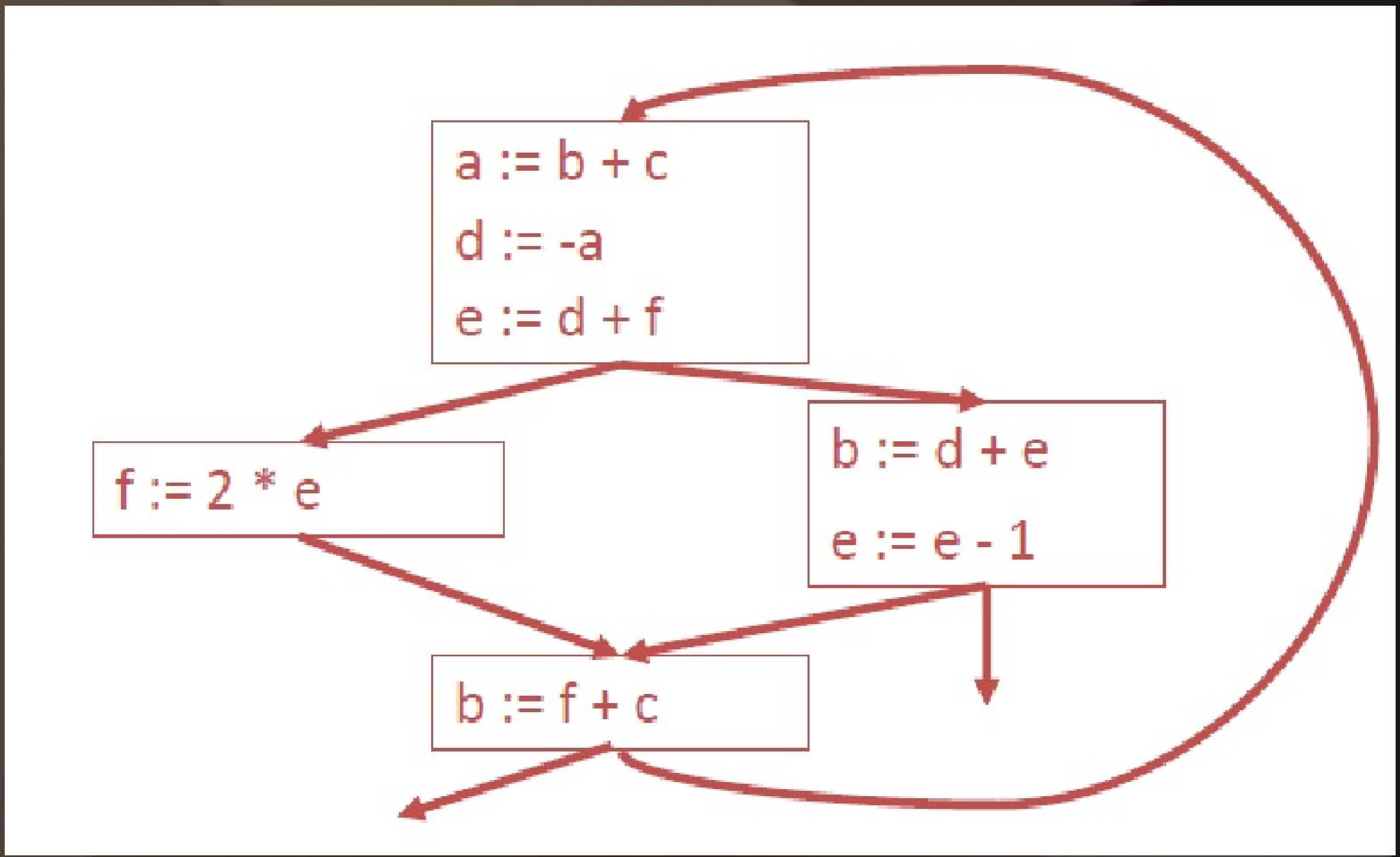
- Since optimistic coloring failed we must spill temporary *f*
- We must allocate a memory location as the home of *f*
  - Typically this is in the current stack frame
  - Call this address *fa*
- Before each operation that uses *f*, insert

  $f :=$ load fa
- After each operation that defines *f*, insert

  store f, fa

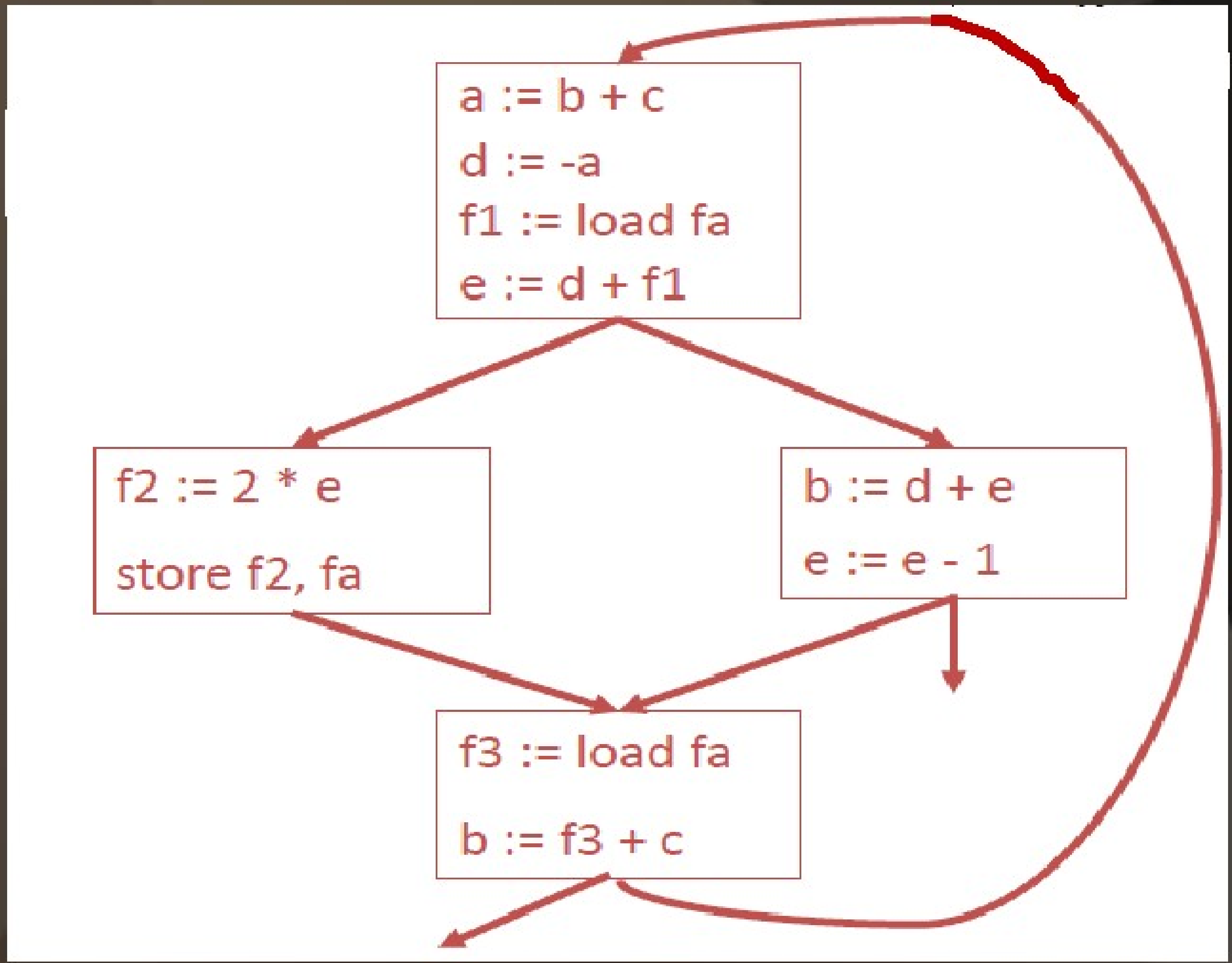
Spilling

- Original code:



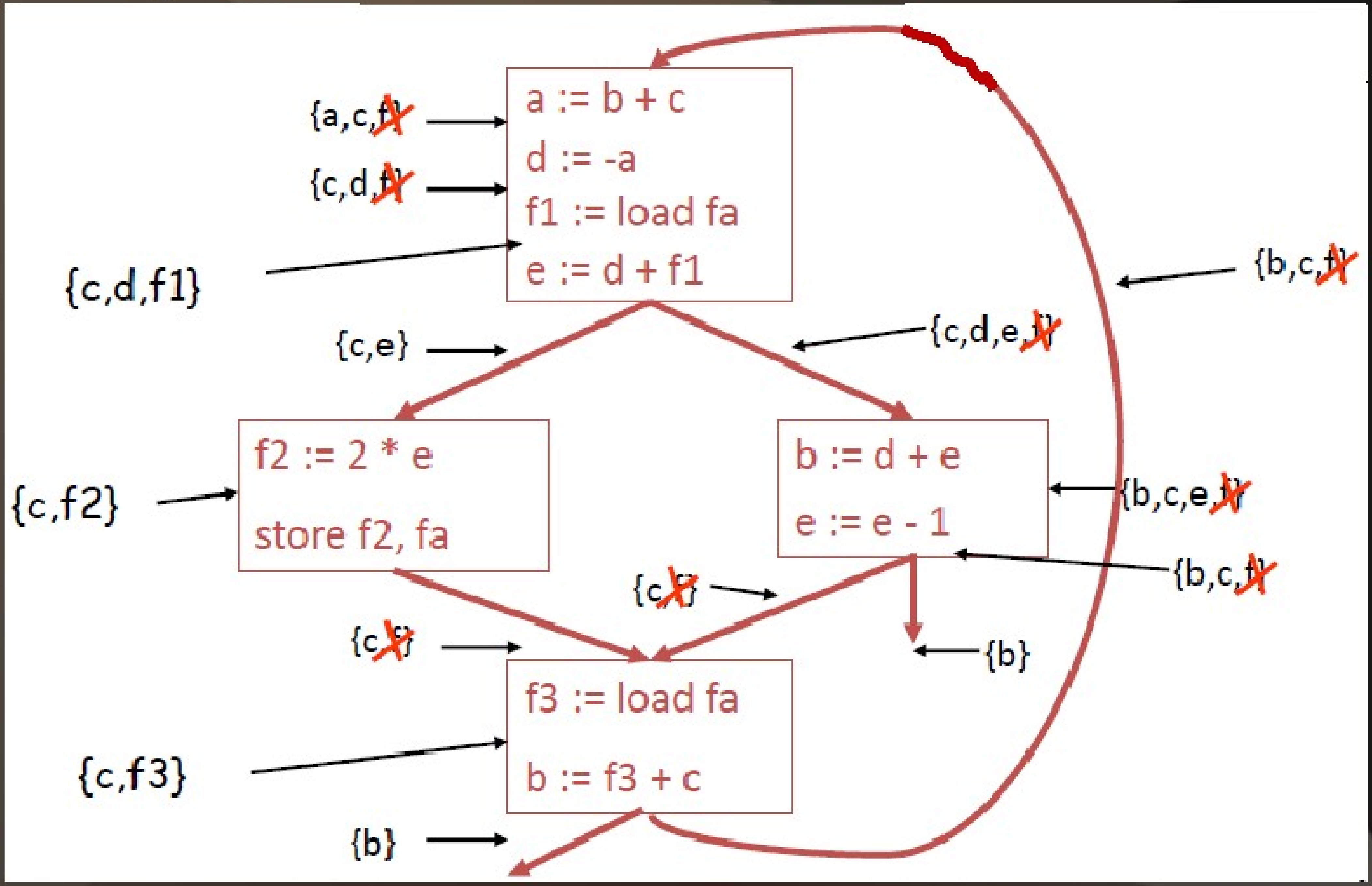
Spilling

- Code after spilling:



Spilling

- Re-compute Liveness:



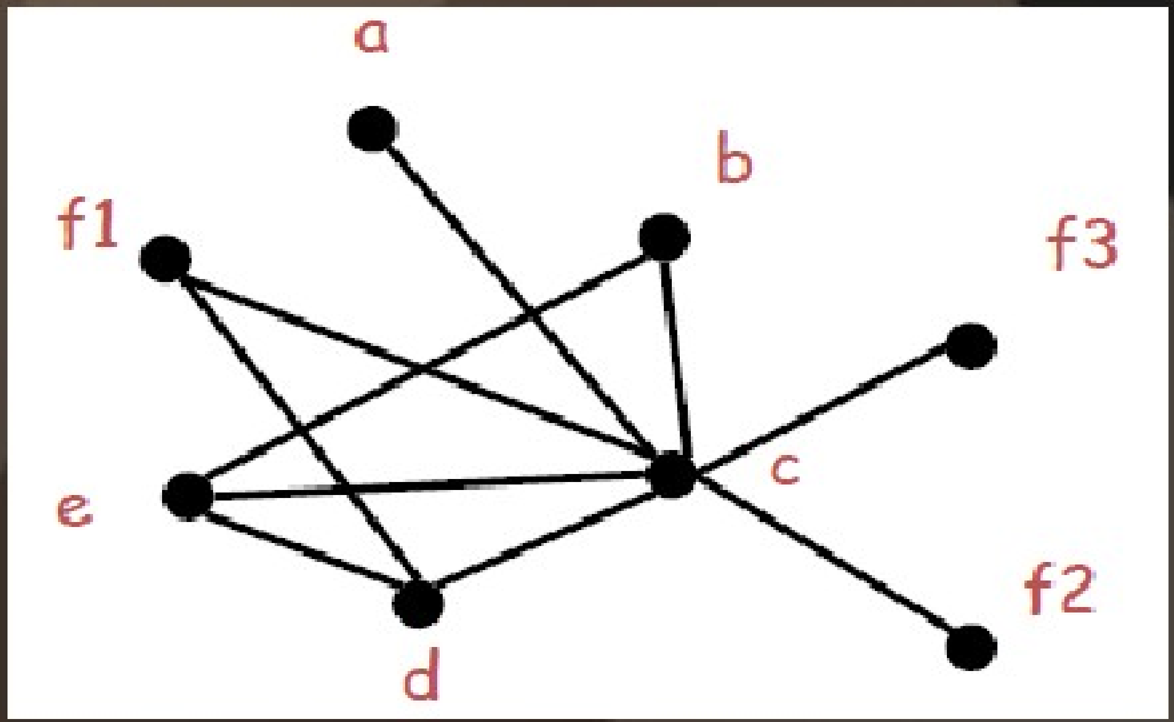
Spilling

- The new liveness information is almost as before
- f_i is live only
  - Between a f_i:= load *fa* and the next instruction
  - Between a store f_i, *fa* and the preceding instruction

- Spilling reduces the live range of f and thus reduces its interferences
  - Which result in fewer neighbors in RIG for f

Spilling

- With the new liveness information, we need to rebuild the RIG
  - And try to colour the resulting graph again



- Now *f* only interfaces with *c* and *d*
- The new RIG is 3-colourable

Spilling

- Additional spills might be required before a coloring is found
- The tricky part is deciding what to spill
    - But any choice is correct

- Possible heuristics:
    - Spill temporaries with most conflicts
    - Spill temporaries with few definitions and uses
    - Avoid spilling in inner loops