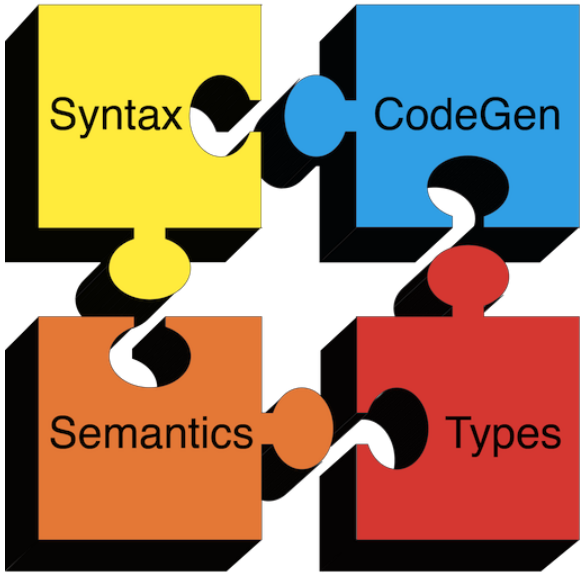


Compilers

Spilling

Spilling

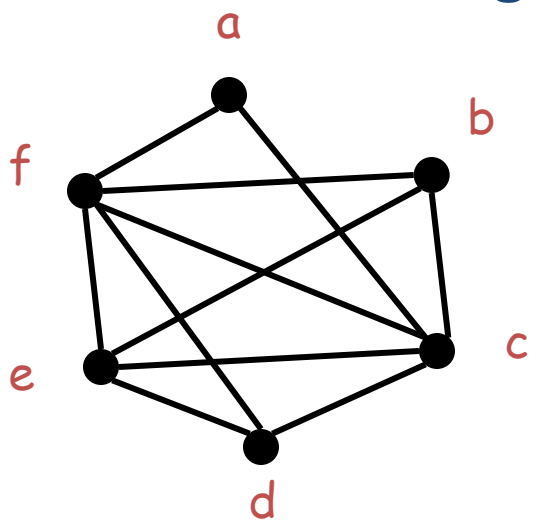
- What happens if the graph coloring heuristic fails to find a coloring?

- In this case, we can't hold all values in registers.
  - Some values are *spilled* to memory

Spilling

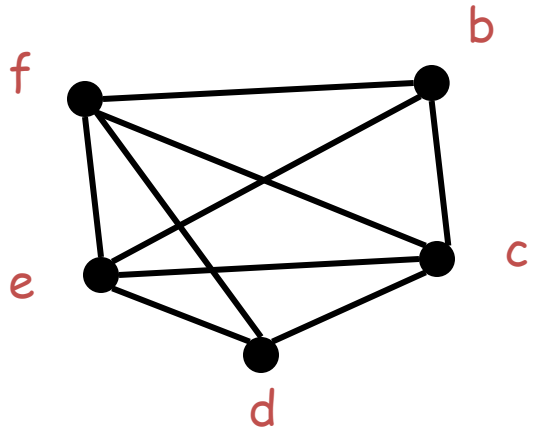
- What if all nodes have k or more neighbors?

- Example: Try to find a 3-coloring of the RIG:



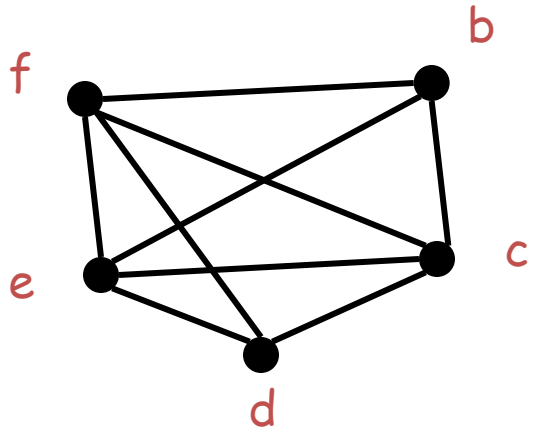
Spilling

- Remove a and get stuck



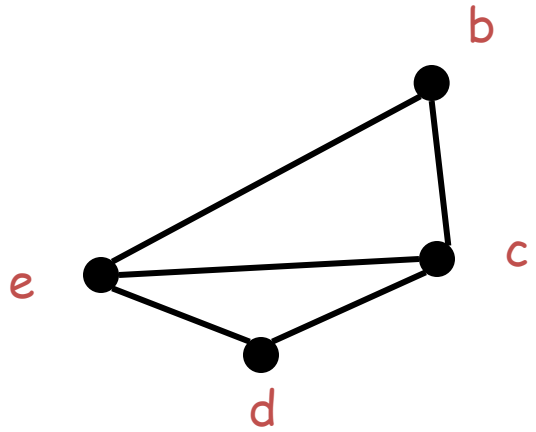
Spilling

- Pick a node as a candidate for spilling
  - A spilled value “lives” in memory
  - Assume f is chosen



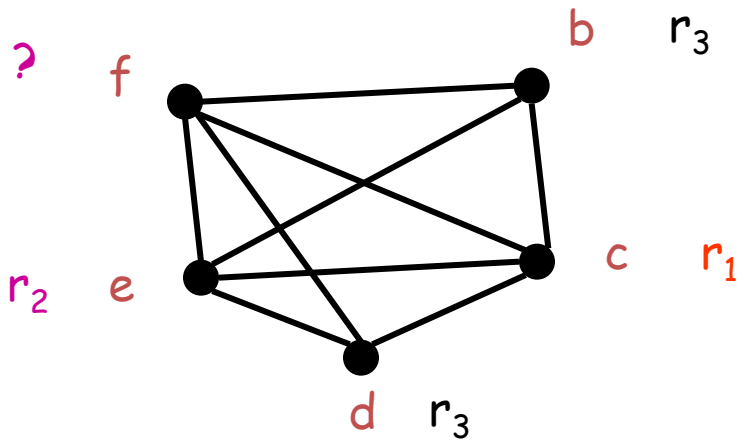
Spilling

- Remove f and continue the simplification
  - Simplification now succeeds: b, d, e, c



Spilling

- Eventually we must assign a color to f
- We hope that among the 4 neighbors of f we use less than 3 colors ⇒ optimistic coloring



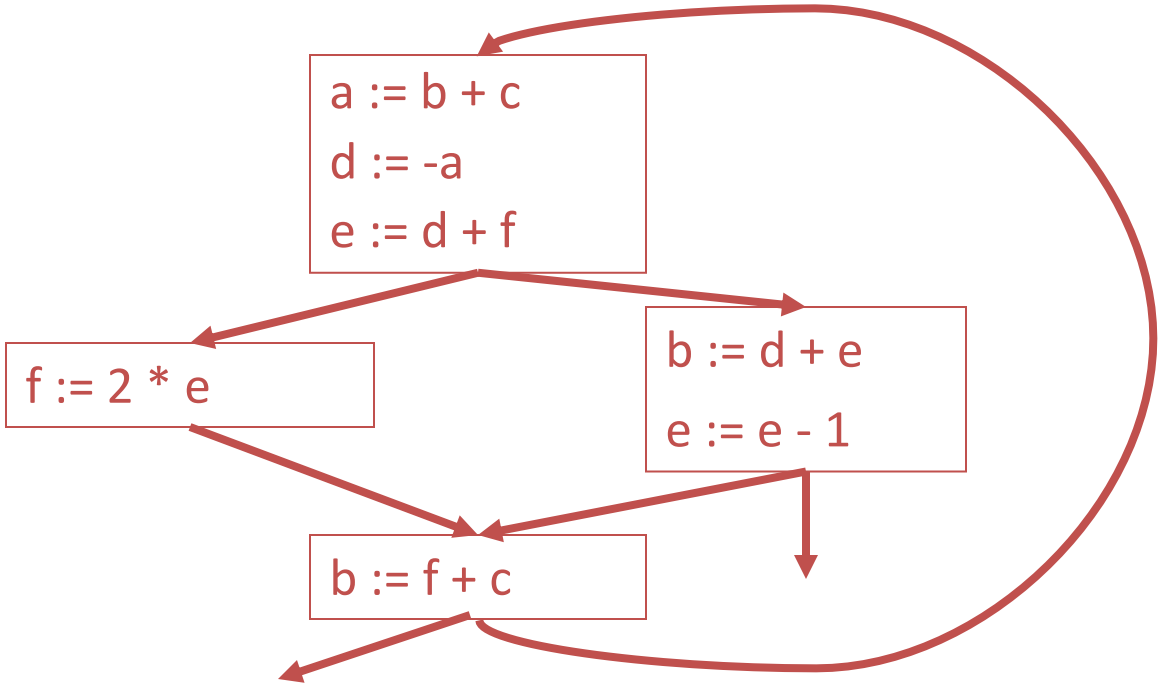
Spilling

- If optimistic coloring fails, we spill f
  - Allocate a memory location for f
    - Typically in the current stack frame
    - Call this address fa

- Before each operation that reads f, insert
  f := load fa

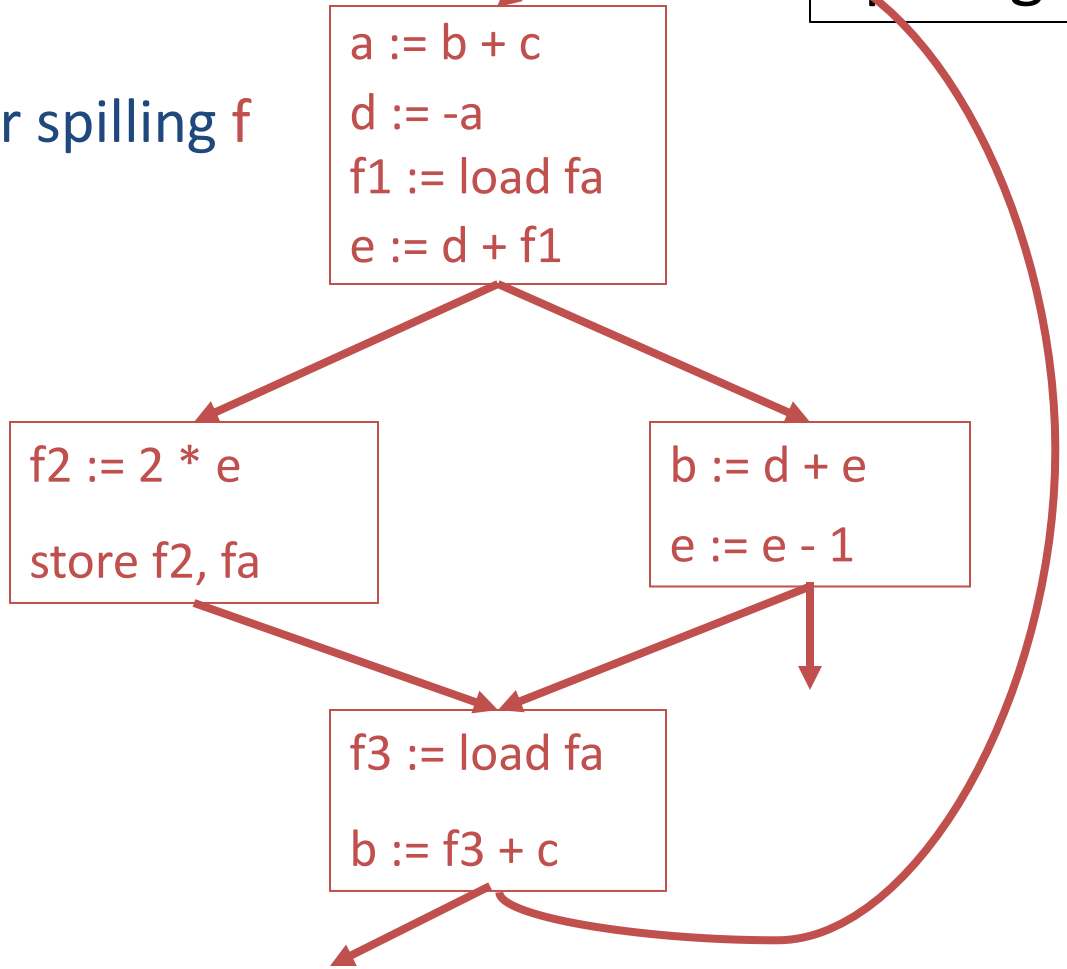
- After each operation that writes f, insert
  store f, fa

Original code

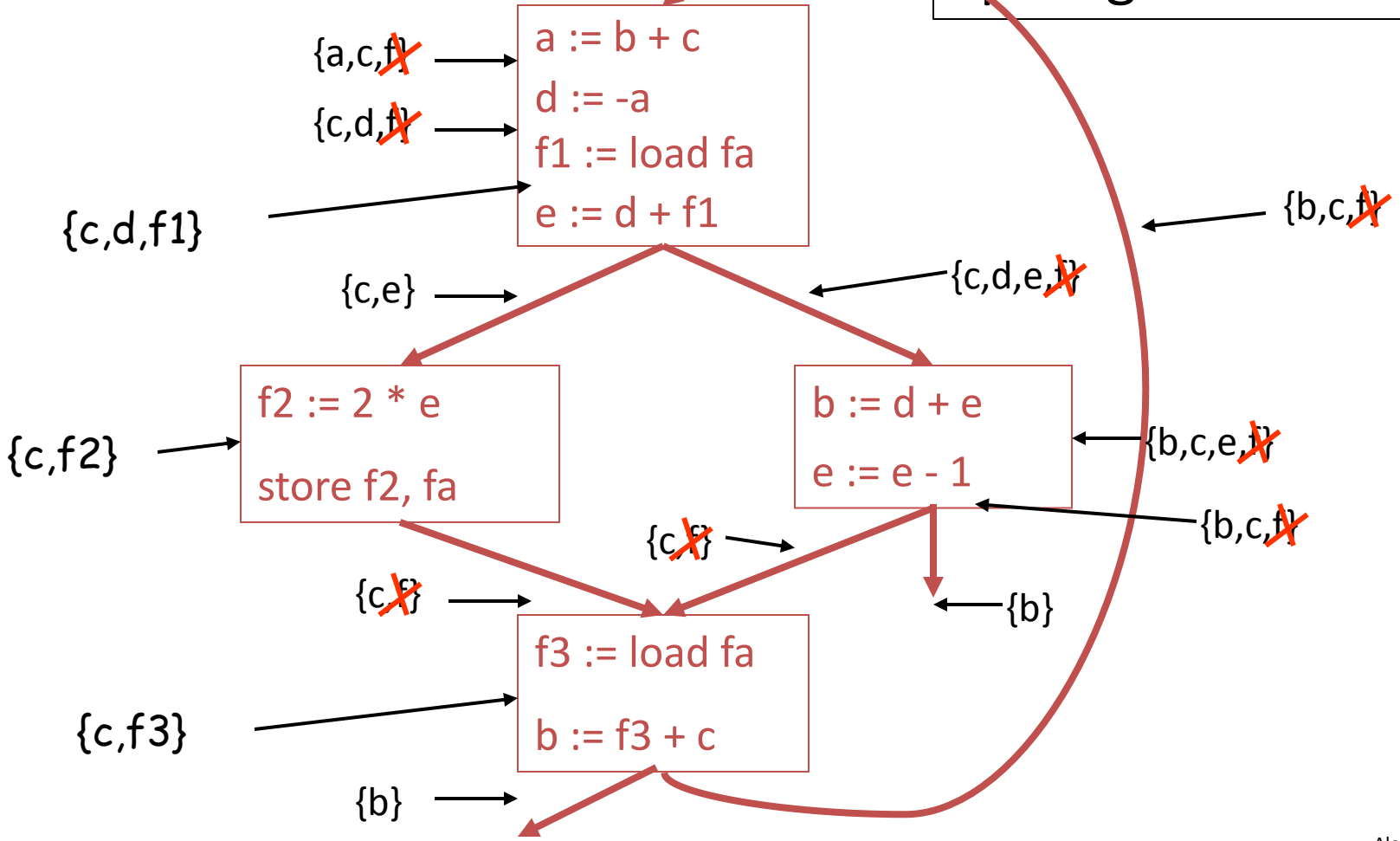


Spilling

The code after spilling f



Recompute liveness

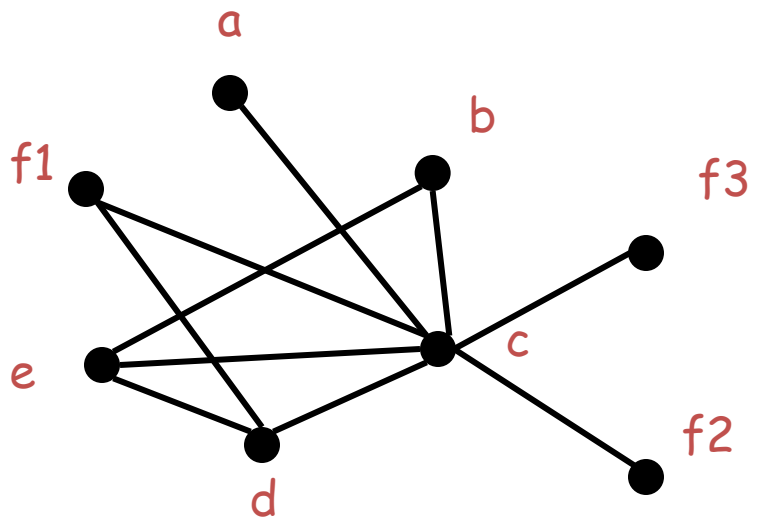


Spilling

- New liveness information is almost as before
  - Note f has been split into three temporaries
- fi is live only
  - Between a fi := load fa and the next instruction
  - Between a store fi, fa and the preceding instr.
- Spilling reduces the live range of f
  - And thus reduces its interferences
  - Which results in fewer RIG neighbors

Spilling

- Some edges of the spilled node are removed
- In our case f still interferes only with c and d
- And the new RIG is 3-colorable



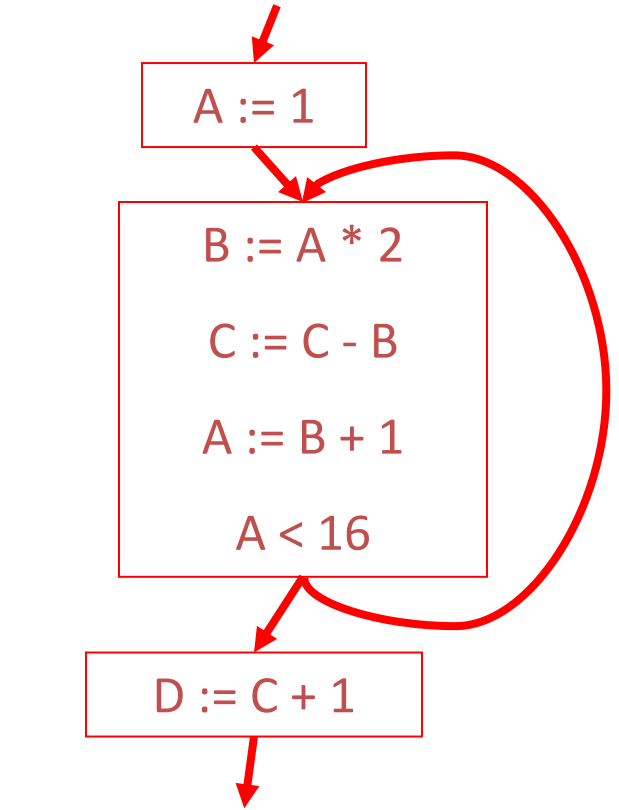
Spilling

- 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

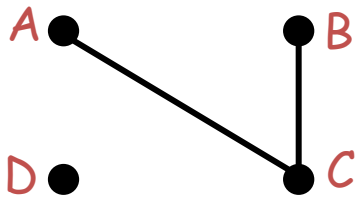
Spilling

For the given code fragment and RIG, find the minimum cost spill. In this example, the cost of spilling a node is given by:

# of occurrences (use or definition)
– # of conflicts
+ 5 if the node corresponds to a variable used in a loop



- ○ A
- ○ B
- ○ C
- ○ D



Spilling

- Register allocation is a “must have” in compilers:
  - Because intermediate code uses too many temporaries
  - Because it makes a big difference in performance

- Register allocation is more complicated for CISC machines