OPT2: Spilling

Spilling in Register Allocation

CMPT 379: Compilers

Instructor: Anoop Sarkar

anoopsarkar.github.io/compilers-class

Register Allocation as Graph Coloring

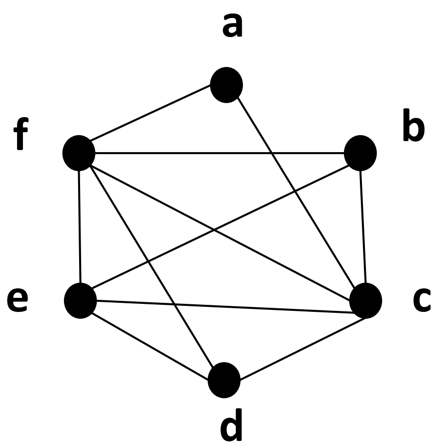
- What happens if the graph coloring heuristic fails to find a coloring?

- In this case we cannot hold all values in the registers
    - Some values should be *spilled* to memory

K-coloring fails

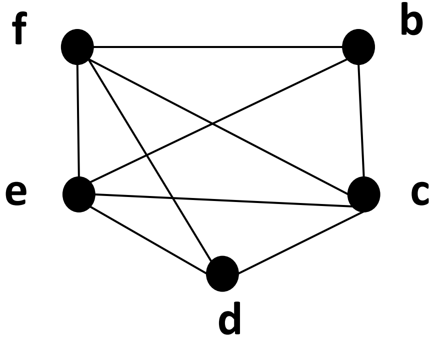
- What if all nodes have k or more neighbors?
- Try to find a 3 coloring of this graph

Remove a



Example of 3-coloring

- There is no node such that if we remove it then 3-coloring for the graph is available



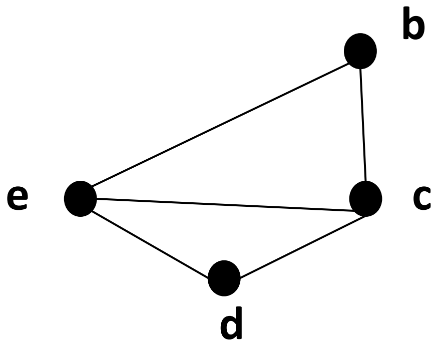
Optimistic Coloring

- If every node in G has more than *k* neighbors,   *k*-coloring of G might not be possible

- Pick a node as candidate for spilling, remove it from the graph and continue *k*-coloring

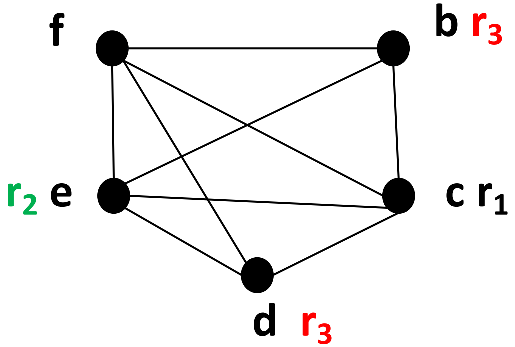
Optimistic Coloring

- Remove f and continue:
  - The ordering: {c,e,d,b,f,a}



Optimistic Coloring

- Color the nodes {c,e,d,b,f,a}
- Try to assign a color to f
- We hope that among 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

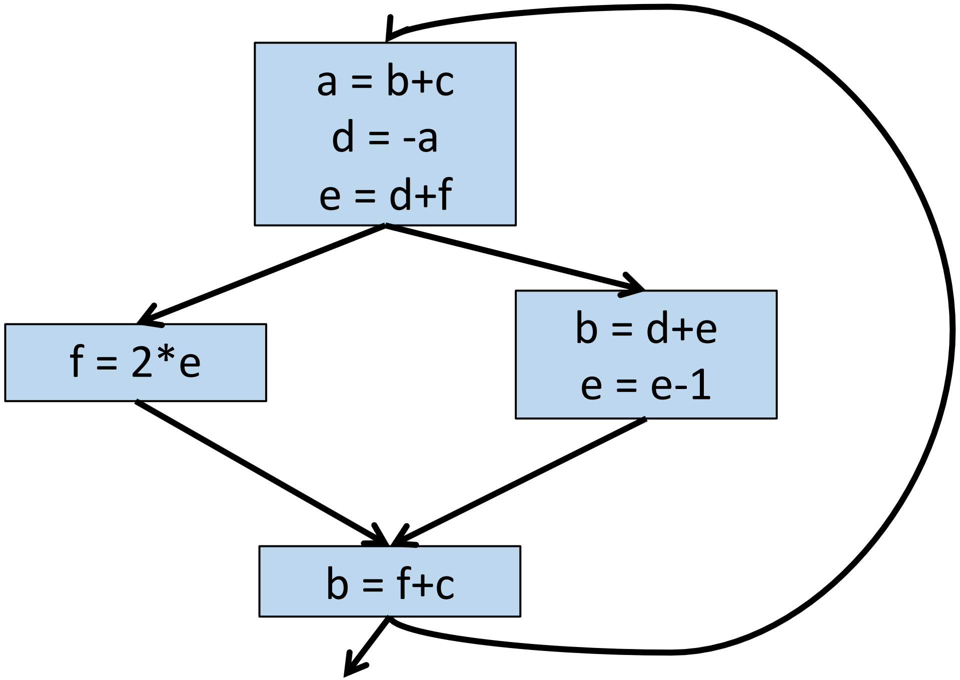
  f = load fa

- After each operation that writes f, insert

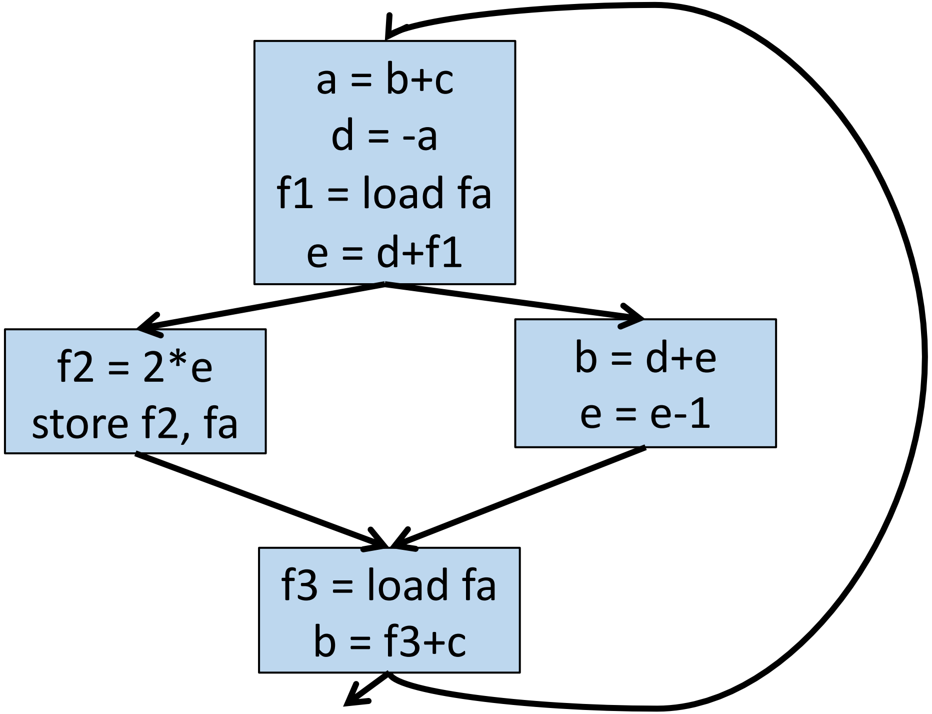
  store f, fa

- Spilling is expensive (wrt time) but sometimes necessary.

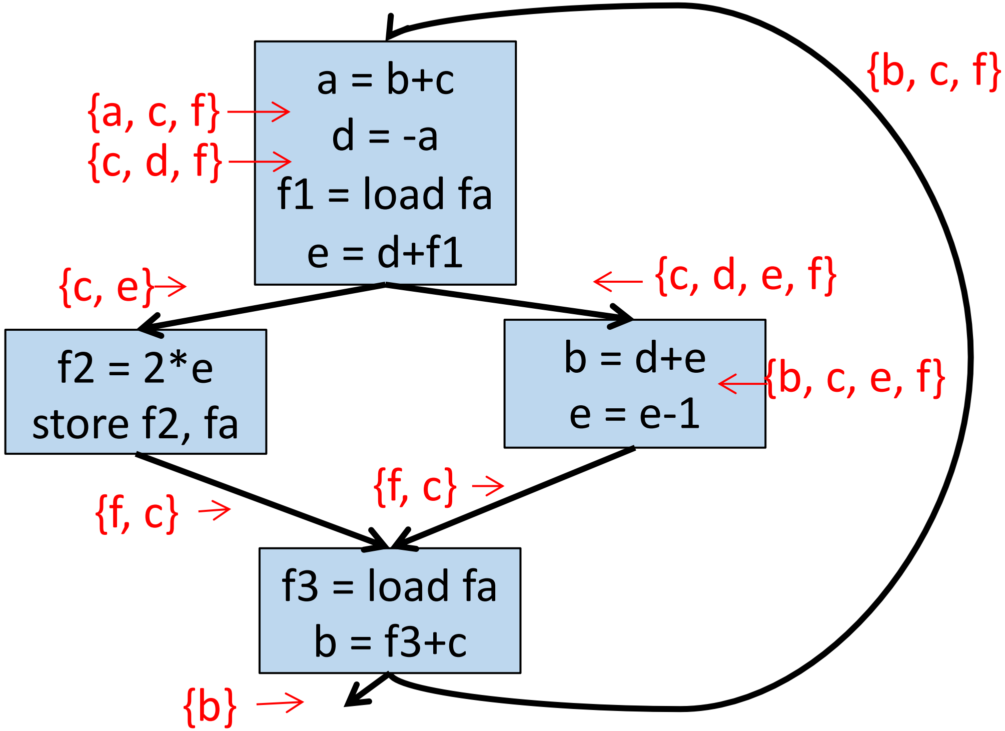
Original Code



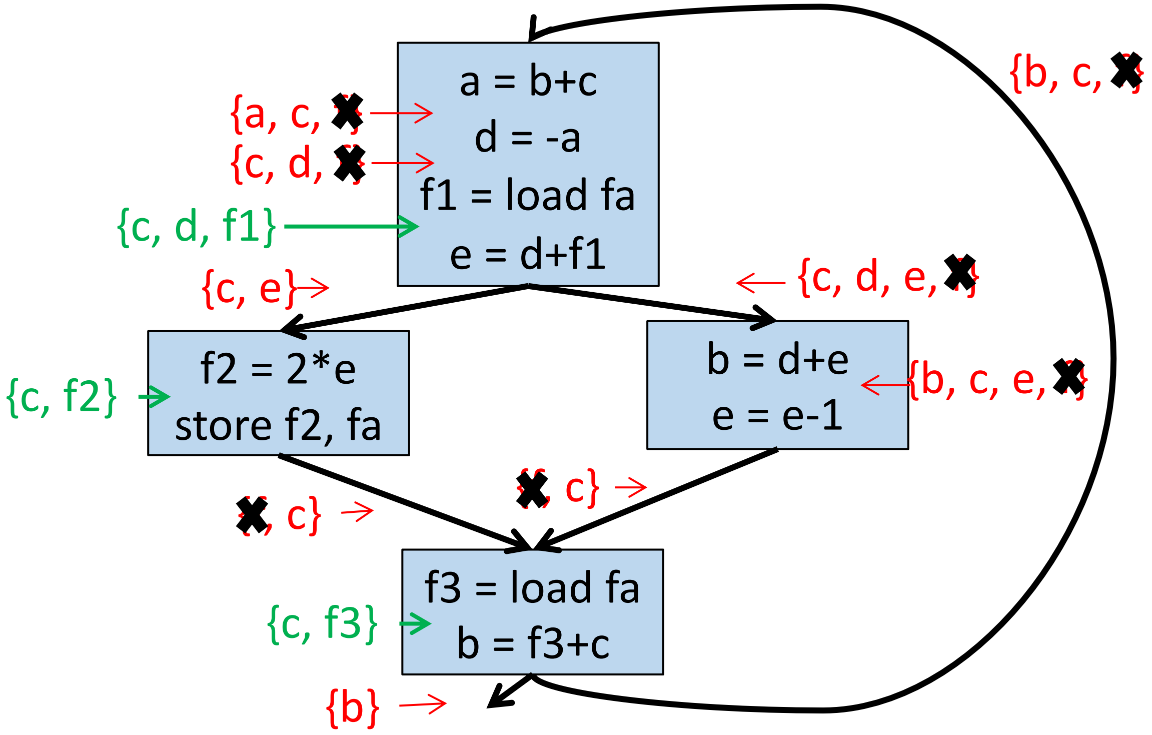
Code after Spilling f



Recompute the Liveness



Recompute the Liveness



a = b+c
d = -a
f1 = load fa
e = d+f1

{b, c, ✖}

{a, c, ✖}

{c, d, ✖}

{c, d, f1}

{c, e}

{c, d, e, ✖}

f2 = 2*e
store f2, fa

{c, f2}

b = d+e
e = e-1

{b, c, e, ✖}

✖, c}

✖, c}

f3 = load fa
b = f3+c

{c, f3}

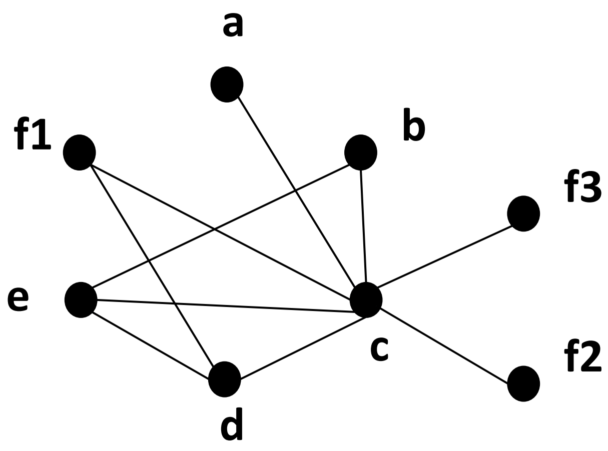
{b}

Rebuild the Interference Graph

- 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

Rebuild the Interference Graph

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



Spilling

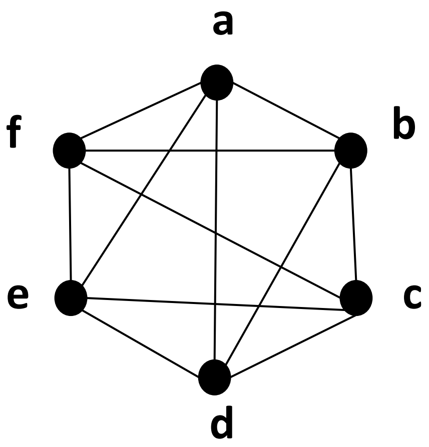
- Additional spilling might be required before a coloring is found

Example

K=3

remove **a**

Stack: {}

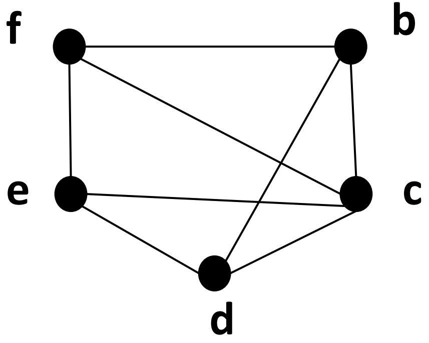


Example

K=3

remove **c**

Stack: {**a**}

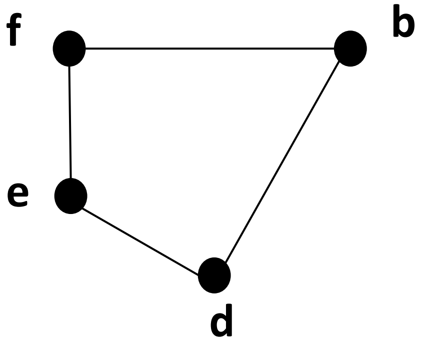


Example

K=3

remove **b**

Stack: {**c,a**}

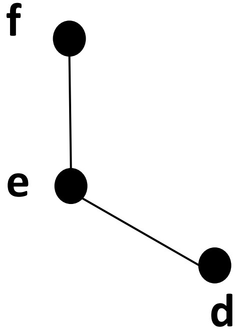


Example

K=3

remove **e**

Stack: {**b**,**c**,**a**}

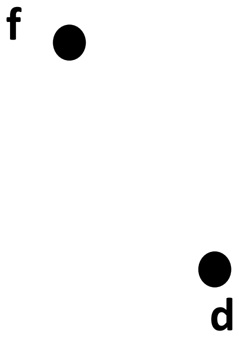


Example

K=3

remove **f**

Stack: {**e**,**b**,**c**,**a**}



Example

K=3

remove **d**

Stack: {**f**,**e**,**b**,**c**,**a**}

Example

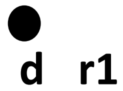
K=3

Stack: {**d**,**f**,**e**,**b**,**c**,**a**}

Example

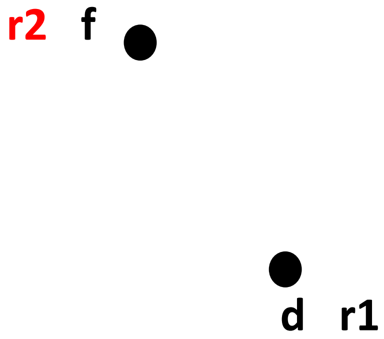
K=3

Stack: {**f**,**e**,**b**,**c**,**a**}



Example

K=3

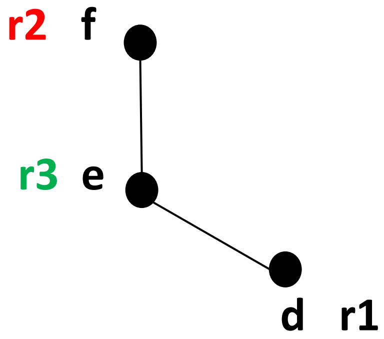


Stack: {**e**,**b**,**c**,**a**}

Example

K=3

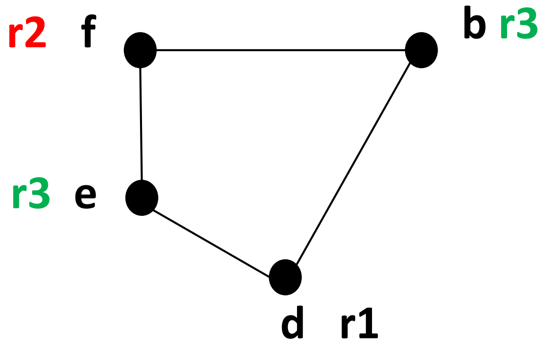
Stack: {**b**,**c**,**a**}



Example

K=3

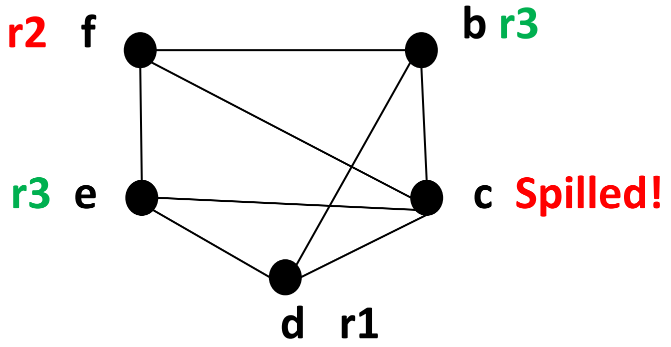
Stack: {c,a}



Example

K=3

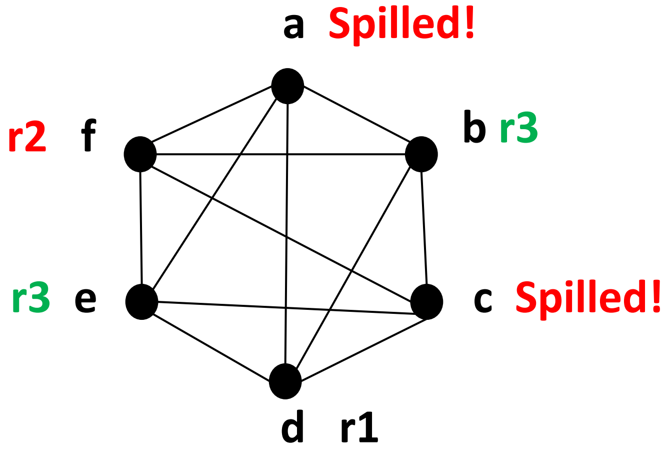
Stack: {a}



Example

K=3

Stack: {}



Spilling

- Many different heuristics for picking a node to spill
  - Spill temporaries with most conflicts
  - Spill temporaries with few definitions and uses
  - Avoid spilling in inner loops (heavily visited regions of the code)
- C allows a *register* keyword to direct the compiler whether a variable contains a value that is heavily used.