Register Allocation: Spilling

- Chaitin's Coloring and Spilling
- Spilitting Live Ranges

Spilling to Memory

- If a node is
  - Colored successfully
    - allocated a hardware register
  - Not colored
    - left in memory

- If we cannot color all nodes, we need to decide which node(s) to spill, but how?
  - Need to consider benefit-to-cost of spilling

Cost-to-Benefit of Spilling

Cost of spilling a node

- Proportional to dynamic number of uses/definitions
- Can be estimated by its loop nesting
- Need to minimize sum of costs of uncolored nodes

Benefit of spilling a node

- Increase colorability of nodes it interfere with
- Can be estimated by its degree in the graph

Greedy heuristic

- Spill the pseudo register with lowest cost-to-benefit ratio, whenever spilling is necessary

Coloring Algorithm (Without Spilling)

Build interference graph

Iterate until there are no nodes left

    If there exists a node v with less than n neighbors

        place v on stack to register allocate

    else

        return (coloring heuristics fail)

    remove v and its edges from graph

While stack is not empty

    remove v from stack

    reinsert v and its edges into the graph

    assign v a color that differs from all its neighbors

Chaitin's Coloring & Spilling Algorithm

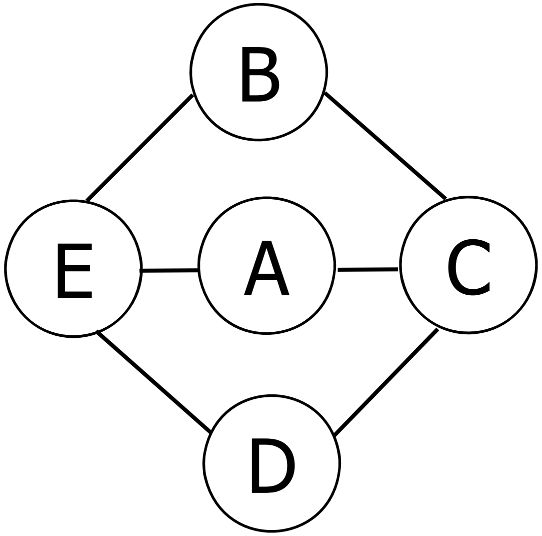
Build interference graph
Iterate until there are no nodes left
    If there exists a node v with less than n neighbors
        place v on stack to register allocate
    else
        v = node with highest degree-to-cost ratio
        spill v and mark v as spilled (or spill v after the iteration)
    remove v and its edges from graph

Spilling may require use of registers and change interference graph
While there is spilling
    rebuild interference graph and perform above step

Assign registers
While stack is not empty
    remove v from stack
    reinsert v and its edges into the graph
    assign v a color that differs from all its neighbors

Quality of Chaitin's Algorithm

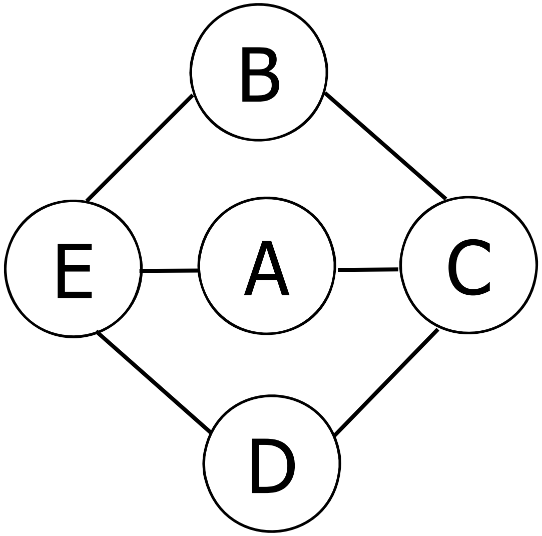
- All-or-nothing: giving up too early
- For the example below when *n=2*
  - Chaitin's will spill although we can color the graph



Optimistic Coloring [Briggs]

An optimization: "Be more optimistic"

- Still remove a spill node and its edges from graph
- But do not commit to "spilling" just yet
- Try to color it again in assignment phase and see if it must really be spilled; otherwise color it!
- More details follow later



Optimization: Splitting Live Ranges

- Split a live range into sub live ranges (by paying small costs) to create a graph that is easier to color
  1. Eliminate interference in a variable's "nearly dead" zones
     - Cost: memory loads and stores at boundaries of regions with no activity
     - # of live ranges at a program point **can be**> # registers
  2. Allocate different registers to a single live range
     - Cost: register copies at boundaries between regions of different assignments
     - # of live ranges at a program point **cannot be**> # registers

Case 1

A and B cannot be assigned to the same register

```
FOR i = 0 TO 10
    FOR j = 0 TO 10000
        A = A + ...  (does not use B)
    FOR j = 0 TO 10000
        B = B + ...  (does not use A)
```

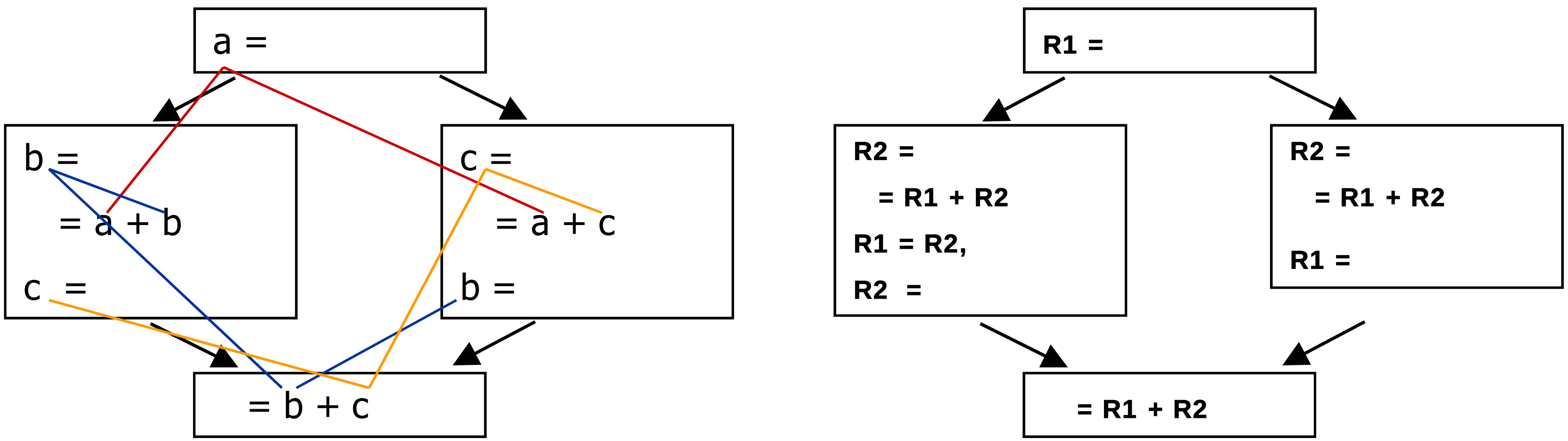
We can allocate A and B the same register by spilling at nearly dead zone

```
FOR i = 0 TO 10
    restore A
    FOR j = 0 TO 10000
        A = A + ...
    store A
    restore B
    FOR j = 0 TO 10000
        B = B + ...
    store B
```

Case 2

When *n=2*, we cannot color the interference graph

But we can avoid a spill by inserting a copy



Live Range Splitting Implementation

- When do we apply live range splitting?
- Which live range to split?
- Where should the live range be split?
- How to apply live range splitting with coloring?
  - Coloring: defers arbitrary assignment decisions until later
  - When coloring fails to proceed, may not need to split live range since degree of a node $\geq n$ does not necessarily mean that the graph is definitely not colorable
  - Interference graph does not show positions of live ranges

It is not that simple at all so there have been lots of research

One Idea for Case 1

- Observation: spilling is absolutely necessary if number of live ranges active at a program point > *n*
- Apply live range splitting before coloring

  Identify a point where # of live ranges > *n*

  For each live range active around that point,

  - Find the outermost "block construct" that does not access the variable

  Choose a live range with the largest inactive region

  Split the inactive region from the live range

Summary

- When there are not enough registers: how to best use the given number of registers?
  - Solve problem in coloring framework
    - Objective: minimize sum of cost of uncolored nodes
      heuristically spill nodes with lowest cost-to-benefit ratio
    - Change interference graph
      Split live ranges: different parts reside in different locations
- Other techniques: reorder execution order to change live ranges