

# Compiler Construction

~ Spilling ~

# Map coloring

A map can always be colored with 4 colors

...BUT for graph coloring, there is no reason for a solution to always exist

# Problem Statement

Not enough registers!

```
...  
t1 := t1 + t2  
...
```

# Solution

## Spilling

Put variables onto the stack!

- t1 will be stored at  $[sp + 4]$
- t2 will be stored at  $[sp + 8]$

The code can then be rewritten:

```
...  
[sp+4] := [sp+4] + [sp+8]  
...
```

# Spilling on RISC Architectures

RISC architecture does not support such operations:

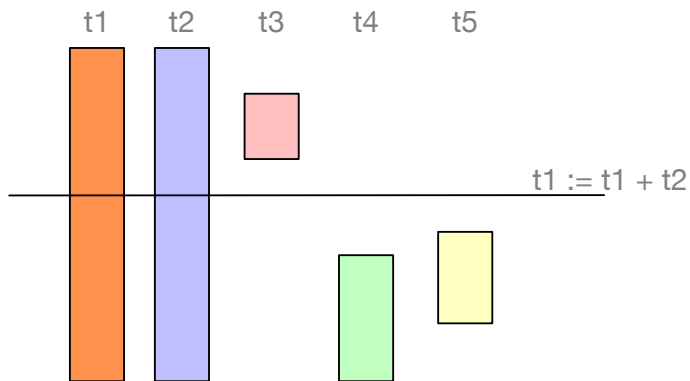
```
[sp+4] := [sp+4] + [sp+8]
```

So we need to use temporaries

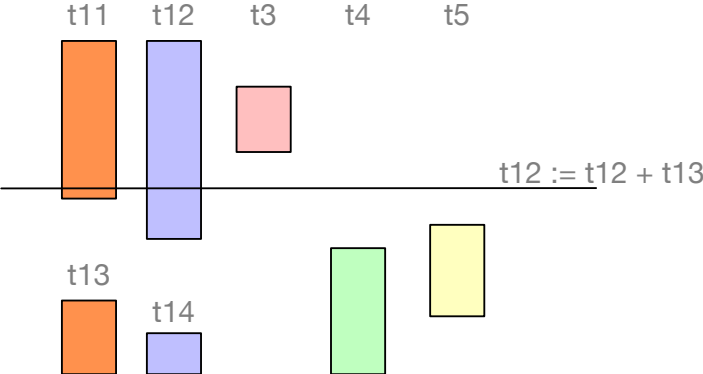
```
t12 := [sp + 4]
t13 := [sp + 8]
t12 := t12 + t13
[sp + 4] := t12
```

Why should it solve the problem?

# All is about lifetimes (1/2)



# All is about lifetimes (2/2)



# Who Should be Spilled?

- The simplification order does not matter
- The spilling order matters Spilling decreases the degree of the neighbors . . . hence it enables additional simplifications
  - ▶ so "first spilled, last served"
  - ▶ therefore: spill cheap temporaries
- few def/uses => pay attention to loops



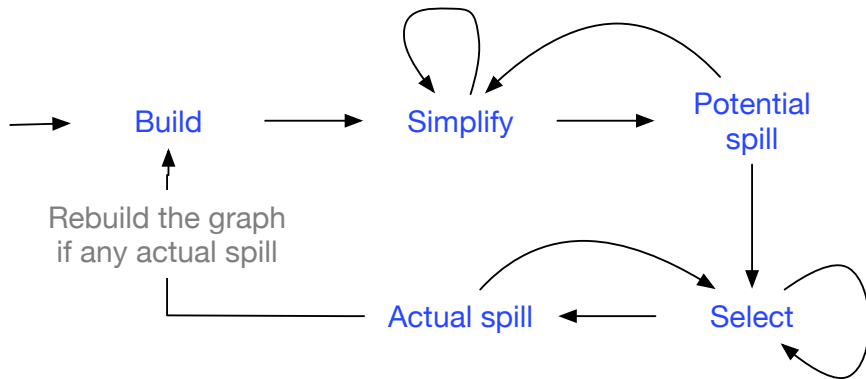
# Spill Priority

Let us consider this example:

	uses+defs outside loop			uses+defs within loop			degree		spill priority
a	(	2	+	0	* 10 ) /	4	=	0.50	
b	(	1	+	1	* 10 ) /	4	=	2.75	
c	(	2	+	0	* 10 ) /	6	=	0.33	
d	(	2	+	2	* 10 ) /	4	=	5.50	
e	(	1	+	3	* 10 ) /	3	=	10.3	

c has the lowest priority—it interferes with many other temporaries but is rarely used—so it should be spilled first.

# Complete workflow




**potential spill** nodes with significant degree

**actual spill** do assign color, wait to identify other spills.

# Summary



Potential Spill



Actual Spill



Spill Priority



Lifetime and  
Stack