Lecture 16
Register Allocation:
Coalescing

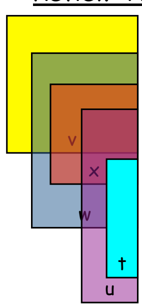
(Slides courtesy of Seth Goldstein and David Koes.)

Review: An Example, k=4

```
v <-  1
w <-  v + 3
x <-  w + v
u <-  v
t <-  u + x
  <-  w
  <-  t
  <-  u
```


Review: An Example, k=4

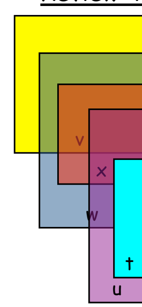
```
v <-  1
w <-  v + 3
x <-  w + v
u <-  v
t <-  u + x
  <-  w
  <-  t
  <-  u
```

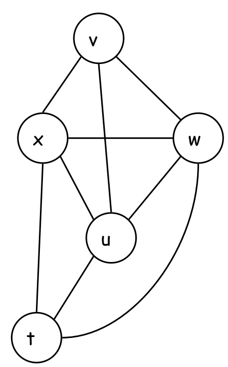


Compute live ranges

Review: An Example, k=4

```
v <-  1
w <-  v + 3
x <-  w + v
u <-  v
t <-  u + x
  <-  w
  <-  t
  <-  u
```



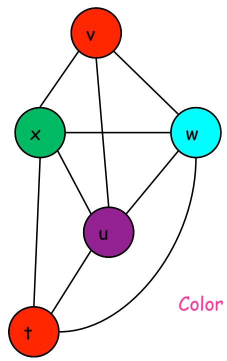


Construct the interference graph

Review: An Example, k=4

```
v <-  1
w <-  v + 3
x <-  w + v
u <-  v
t <-  u + x
  <-  w
  <-  t
  <-  u
```

Voila, registers are assigned!

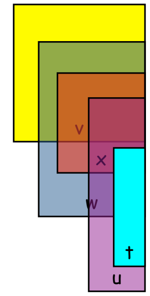


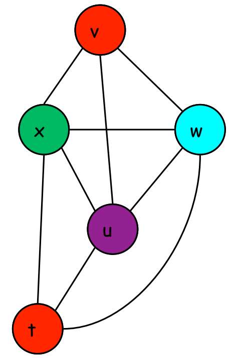
Color the graph

But, can we do better?

An Example, k=4

```
v <-  1
w <-  v + 3
x <-  w + v
u <-  v
t <-  u + x
  <-  w
  <-  t
  <-  u
```

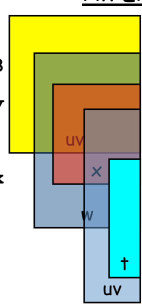


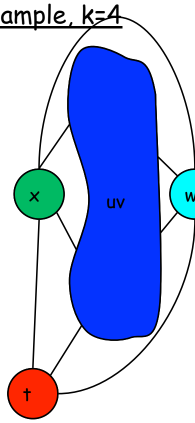


u & v are special. They interfere, but only through a move!

An Example, k=4

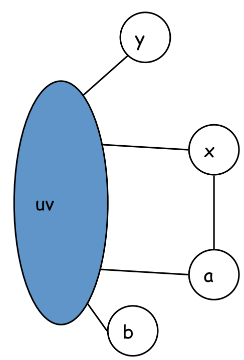
`uv <- 1`
`w <-  uv + 3`
`x <-  w + uv`
~~`u <-  v`~~
`t <-  uv + x`
`  <-  w`
`  <-  t`
`  <-  uv`





Rewrite the code to coalesce u & v

Is Coalescing Always Good?

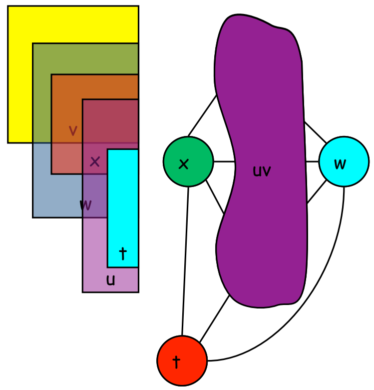


Was 2-colorable,
now it needs 3 colors

So, we treat moves specially.

An Example, k=4

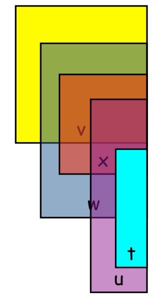
```
v <-  1
w <-  v + 3
x <-  w + v
u <-  v
t <-  u + x
  <-  w
  <-  t
  <-  u
```



Interference from moves become "move edges."

An Example, k=3

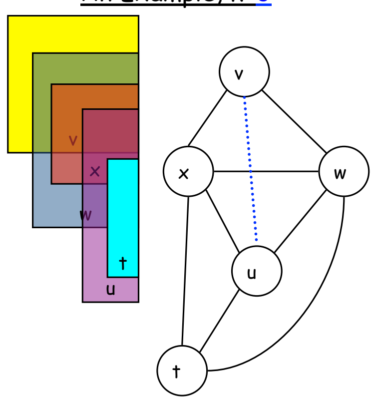
```
v <-  1
w <-  v + 3
x <-  w + v
u <-  v
t <-  u + x
  <-  w
  <-  t
  <-  u
```



Compute live ranges

An Example, k=3

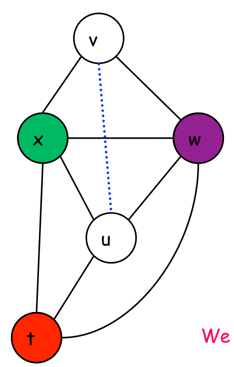
```
v <-  1
w <-  v + 3
x <-  w + v
u <-  v
t <-  u + x
  <-  w
  <-  t
  <-  u
```



Construct the interference graph

An Example, k=3

```
v <-  1
w <-  v + 3
x <-  w + v
u <-  v
t <-  u + x
  <-  w
  <-  t
  <-  u
```



We need to spill

Color the interference graph

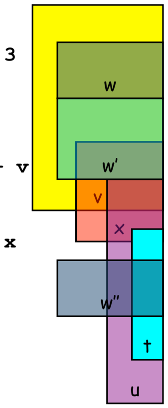
An Example, k=3

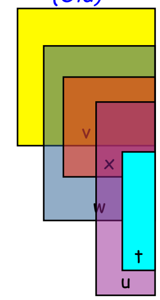
```
v   <-   1
w   <-   v + 3
M[]<-    w
w'  <-   M[]
x   <-   w' + v
u   <-   v
t   <-   u + x
w'' <-   M[]
    <-   w''
    <-   t
    <-   u
```

Rewrite program

An Example, k=3 *(Old)*

```
v   <-   1
w   <-   v + 3
M[]<-    w
w'  <-   M[]
x   <-   w' + v
u   <-   v
t   <-   u + x
w'' <-   M[]
    <-   w''
    <-   t
    <-   u
```



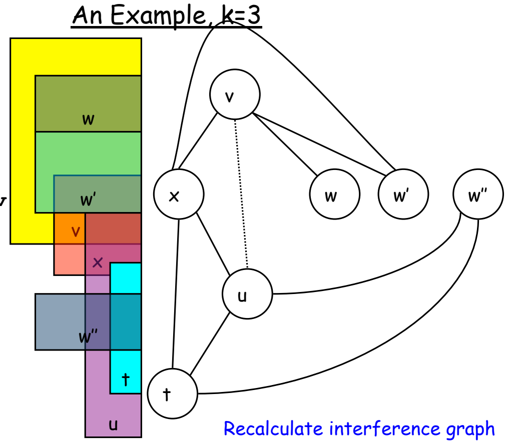


Recalculate live ranges

Spilling reduces live ranges, which decreases register pressure.

An Example, k=3

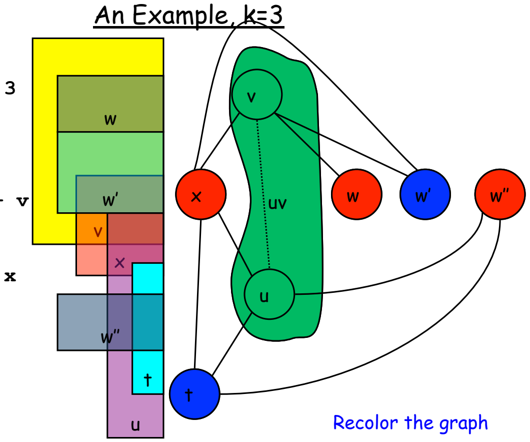
```
v   <-   1
w   <-   v + 3
M[]<-    w
w'  <-   M[]
x   <-   w' + v
u   <-   v
t   <-   u + x
w'' <-   M[]
    <-   w''
    <-   t
    <-   u
```



Recalculate interference graph

An Example, k=3

```
v   <-   1
w   <-   v + 3
M[]<-    w
w'  <-   M[]
x   <-   w' + v
u   <-   v
t   <-   u + x
w'' <-   M[]
    <-   w''
    <-   t
    <-   u
```



Recolor the graph

General Plan

- Construct an interference graph
- Respect special registers:
  - avoid reserved registers
  - use registers properly
  - respect distinction between callee/caller save registers
- Map temporaries to registers
- Generate code to save & restore
- Deal with spills

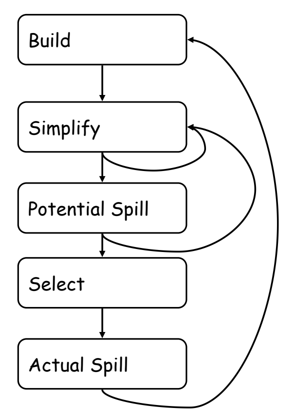
Special Registers

- Which registers can be used?
  - Some registers have special uses.
    - Register 0 or 31 is often hardwired to contain 0.
    - Special registers to hold return address, stack pointer, frame pointer, global area, etc.
    - Reserved registers for operating system.
  - Typically, leaves about 20 or so registers for other general uses.
- Impact on register allocation:
  - Temps should be assigned only to the non-reserved registers.
  - Hard registers are pre-colored in the interference graph.

Register Usage Conventions

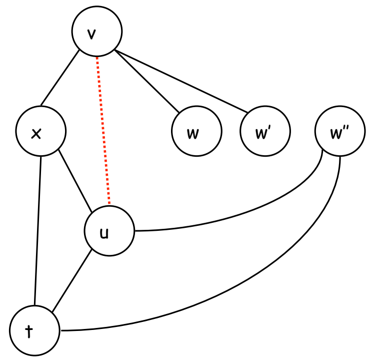
- Certain registers are used for specific purposes by standard calling convention.
  - 4-6 argument registers.
    - The first 4-6 arguments to procedures/functions are always passed in these registers.
  - ~8 callee-save registers.
    - These registers must be preserved across procedure calls. Thus, if a procedure wants to use a callee-save register, it must first save the old value and then restore it before returning.
  - The remainder are caller-save registers.
    - These are not preserved across procedure calls. Thus, a procedure is free to use them without saving first.
    - Includes the argument registers.

The Big Picture



Coalescing

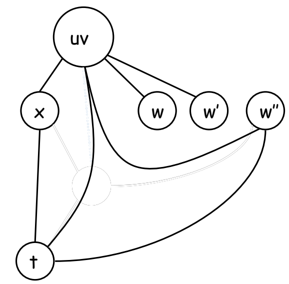
```
v    <-   1
w    <-   v + 3
M[]<-     w
w'   <-   M[]
x    <-   w' + v
u    <-   v
t    <-   u + x
w''  <-   M[]
     <-   w''
     <-   t
     <-   u
```



Can u & v be coalesced?
Should u & v be coalesced?

Briggs: Conservative Coalescing

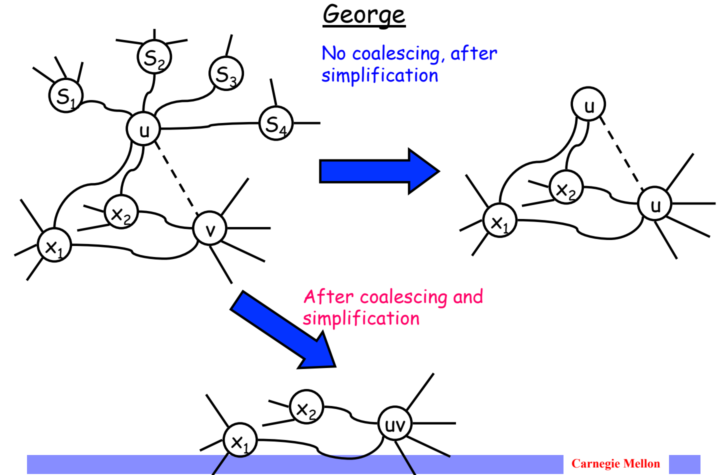
- Can coalesce u and v if:
  - (# of neighbors of uv with degree ≥ k) < k
- Why?
  - Simplify pass removes all nodes with degree < k
  - # of remaining nodes < k
  - Thus, uv can be simplified



George: Iterated Coalescing

- Can coalesce u and v if:
  - foreach neighbor t of u, either:
    - t interferes with v, or,
    - degree of t < k
- Why?
  - let S be set of neighbors of u with degree < k
  - If no coalescing, simplify removes all nodes in S; call that graph G_1
  - If we coalesce, we can still remove all nodes in S; call that graph G_2
  - G_2 is a subgraph of G_1

George



Why Two Methods?

- With Briggs, one needs to look at all neighbors of a & b
- With George, only need to look at neighbors of a.
- We need to insert hard registers in graph and they will have LARGE adjacency lists.
- Hence:
  - Precolored nodes have infinite degree
  - No other precolored nodes in adjacency list
  - Use George if one of a & b is precolored
  - Use Briggs if both are temps

Where We Are