Global Register Allocation - Part 3

Y N Srikant
Computer Science and Automation
Indian Institute of Science
Bangalore 560012

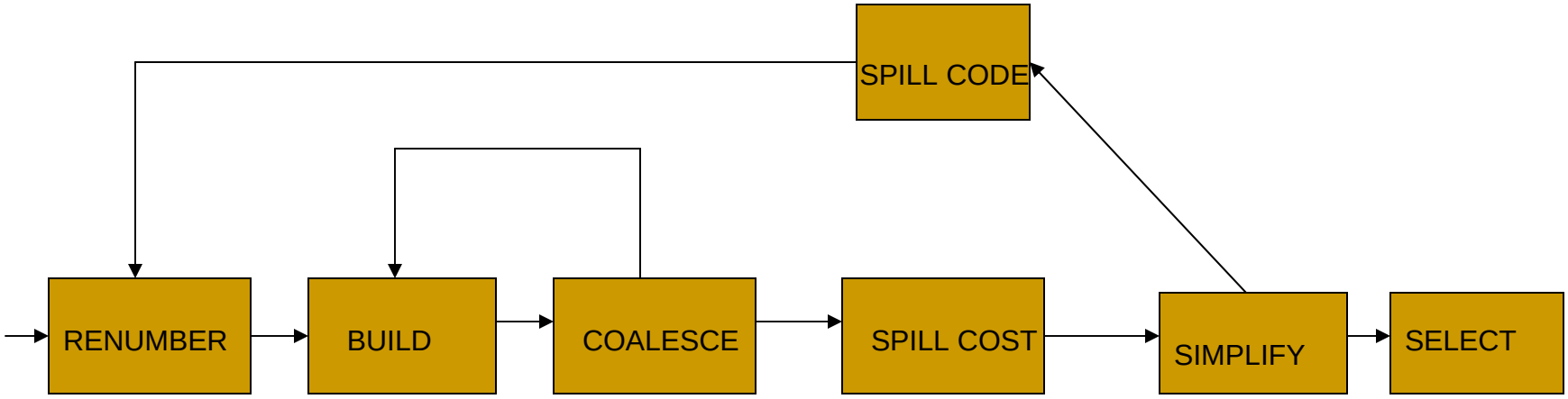
NPTEL Course on Compiler Design

Outline

- Issues in Global Register Allocation
- The Problem
- Register Allocation based in Usage Counts
- Linear Scan Register allocation
- Chaitin's graph colouring based algorithm

Topics 1,2,3,4, and part of 5 were covered in part 1 of the lecture.

The Chaitin Framework



Simplification

- If a node ***n*** in the interference graph has degree less than R, remove ***n*** and all its edges from the graph and place ***n*** on a colouring stack.
- When no more such nodes are removable then we need to spill a node.
- Spilling a variable x implies
  - loading x into a register at every use of x
  - storing x from register into memory at every definition of x

Spilling Cost

- The node to be spilled is decided on the basis of a spill cost for the live range represented by the node.
- Chaitin's estimate of spill cost of a live range v
  - cost(v) = $\sum_{\text{all load or store operations in a live range v}} c * 10^d$
  - where *c* is the cost of the op and *d*, the loop nesting depth.
  - 10 in the eqn above approximates the no. of iterations of any loop
  - The node to be spilled is the one with MIN(cost(v)/deg(v))

Spilling Heuristics

- Multiple heuristic functions are available for making spill decisions (cost(v) as before)

1. $h_0(v)$ = cost(v)/degree(v) : Chaitin's heuristic
2. $h_1(v)$ = cost(v)/[degree(v)]^2
3. $h_2(v)$ = cost(v)/[area(v)*degree(v)]
4. $h_3(v)$ = cost(v)/[area(v)*(degree(v))^2]

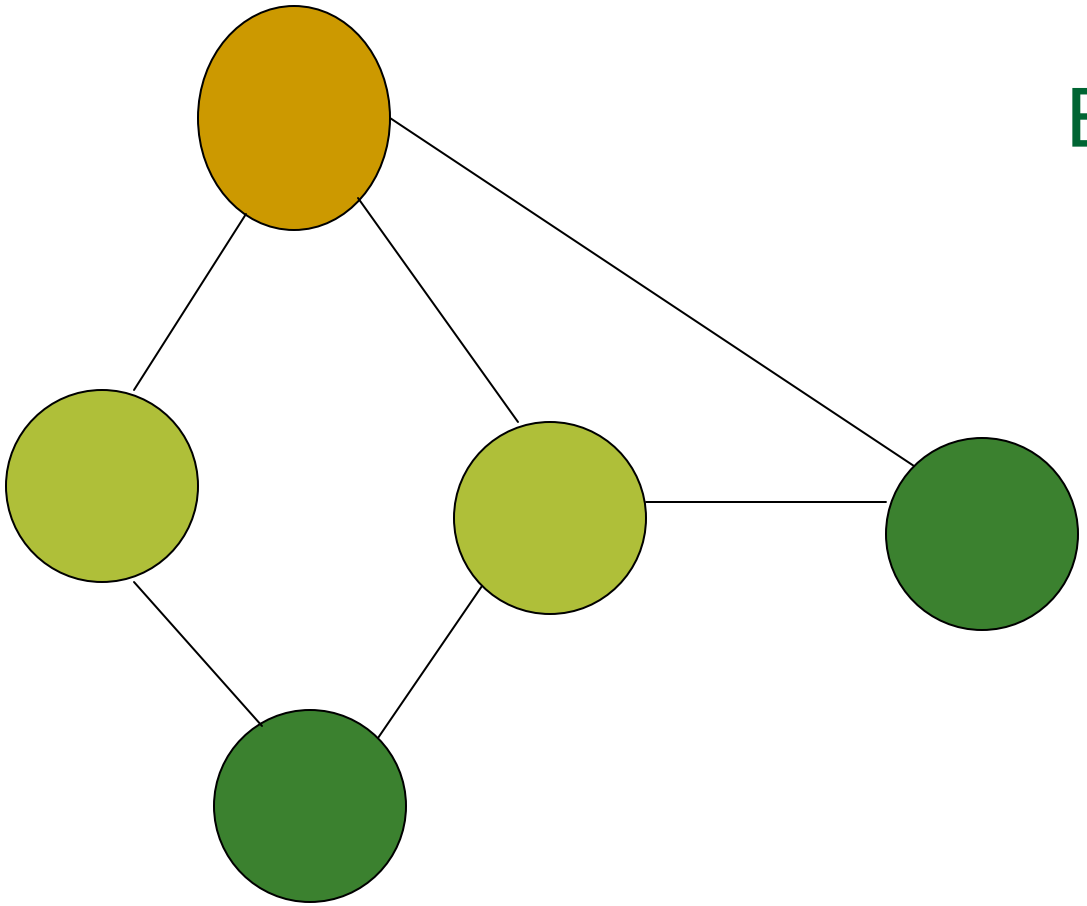
where area(v) = $\sum_{\text{all instructions I in the live range v}} width(v,I) * 5^{depth(v,I)}$

- width(v,I) is the number of live ranges overlapping with instruction I and depth(v,I) is the depth of loop nesting of I in v

Spilling Heuristics

- area(v) represents the global contribution by v to register pressure, a measure of the need for registers at a point
- Spilling a live range with high area releases register pressure; i.e., releases a register when it is most needed
- Choose v with MIN(h_i(v)), as the candidate to spill, if h_i is the heuristic chosen
- It is possible to use different heuristics at different times

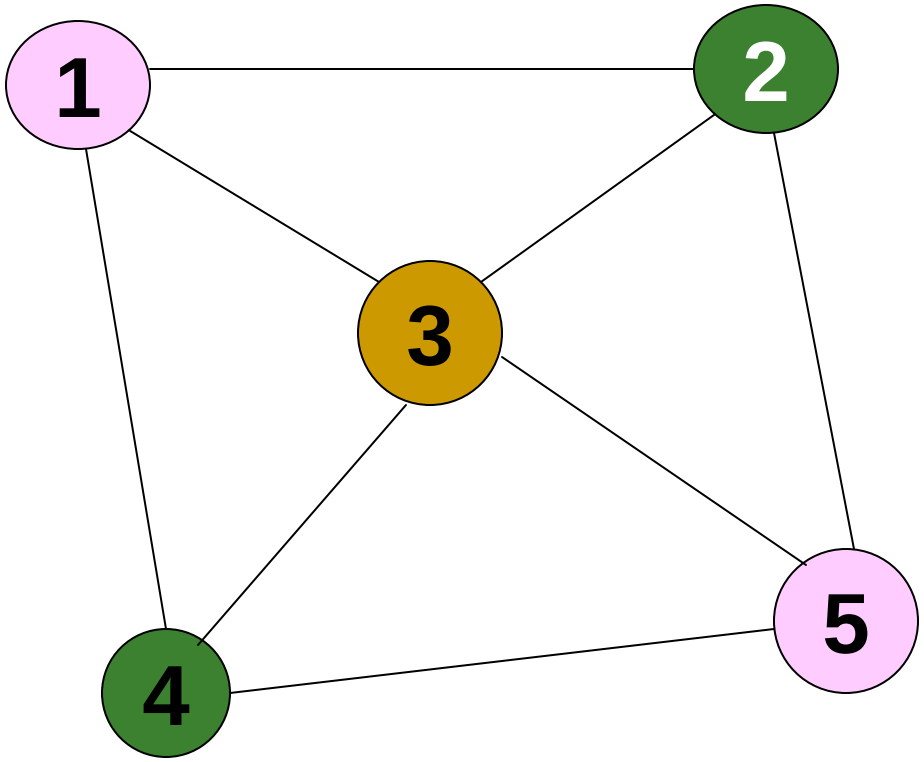
Example



Here R = 3 and the graph is 3-colourable
No spilling is necessary

Example

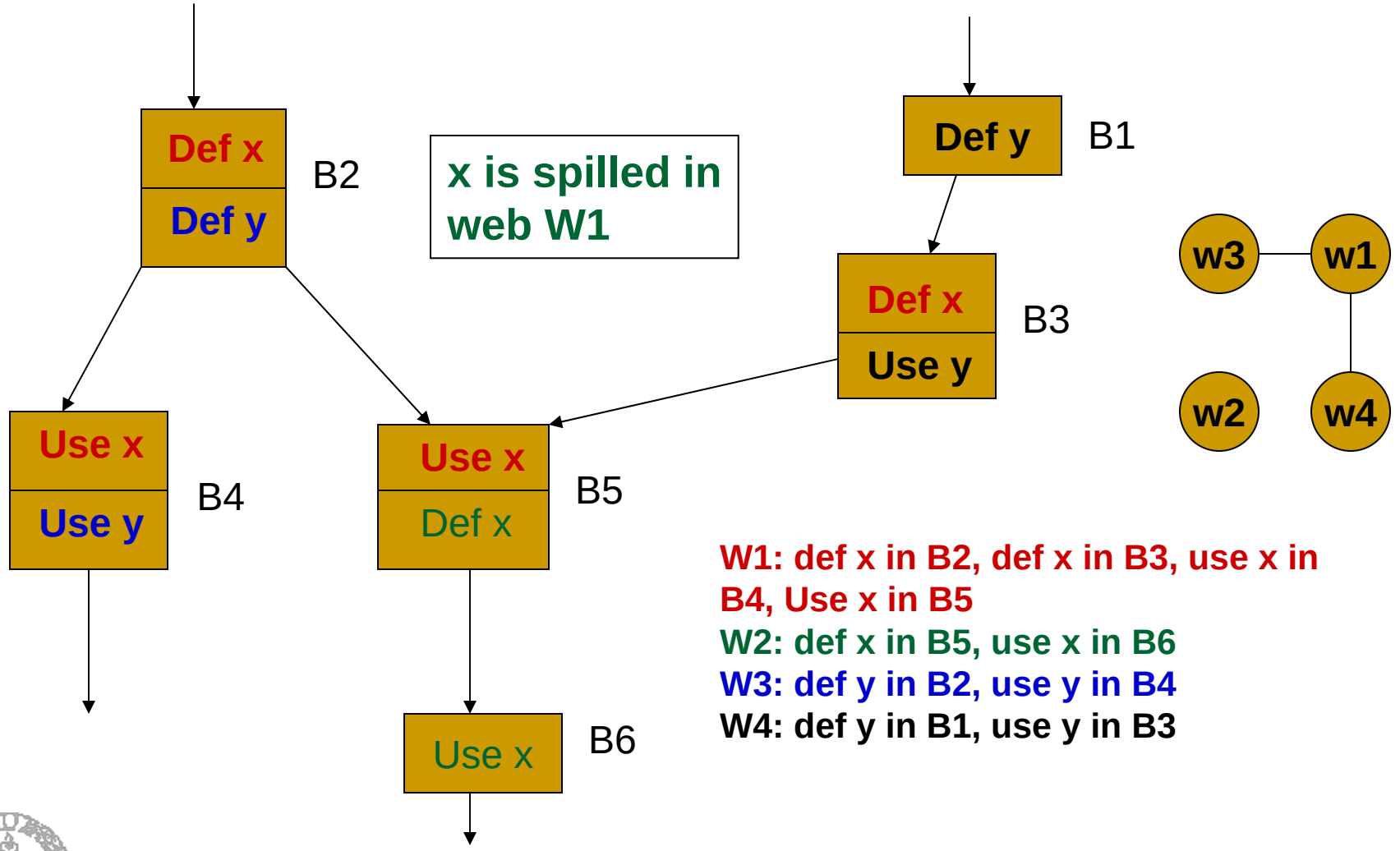
A 3-colourable graph which is not 3-coloured by colouring heuristic



Spilling a Node

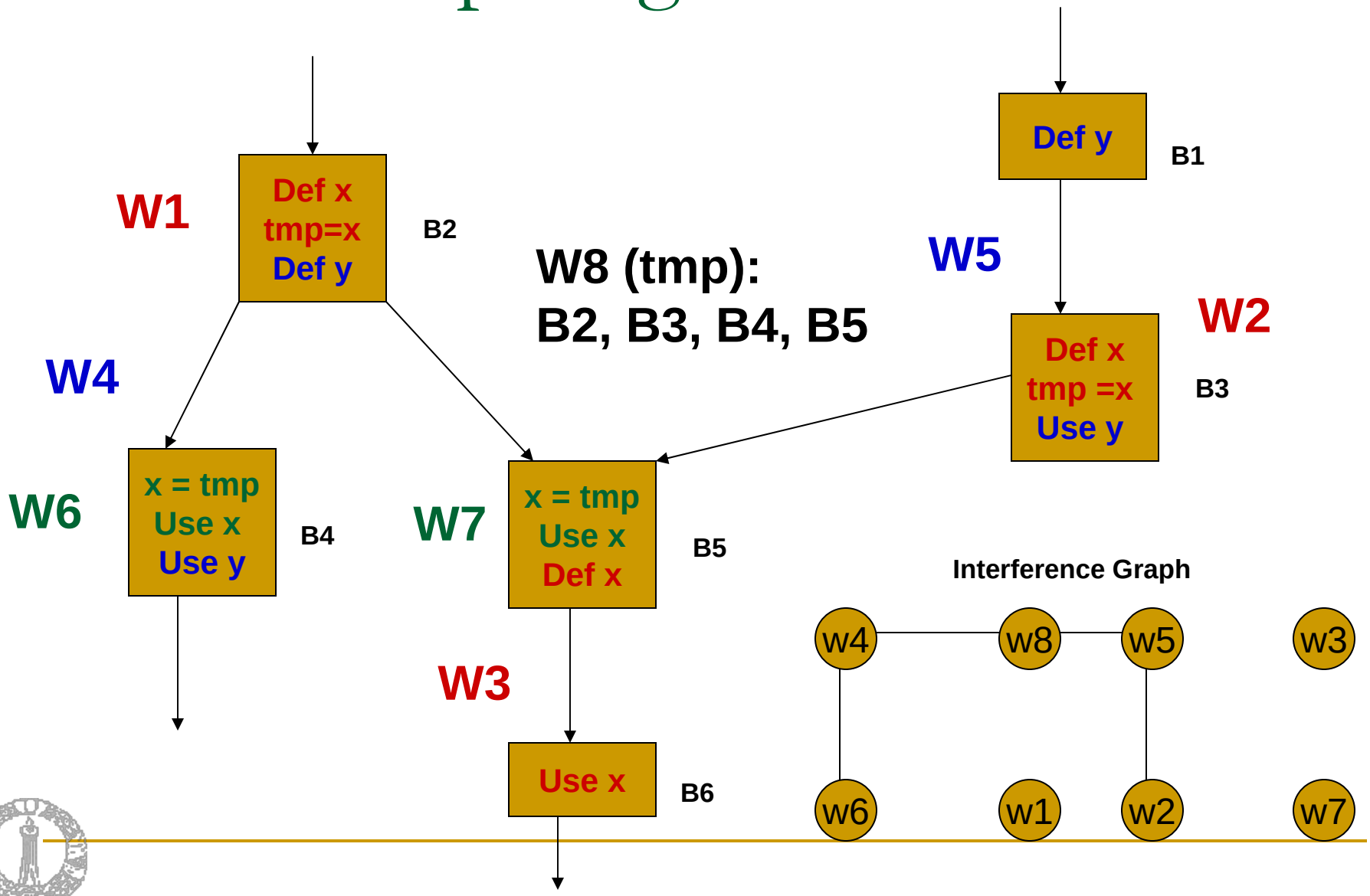
- To spill a node we remove it from the graph and represent the effect of spilling as follows (It cannot just be removed from the graph).
  - Reload the spilled object at each use and store it in memory at each definition point
  - This creates new webs with small live ranges but which will need registers.
- After all spill decisions are made, insert spill code, rebuild the interference graph and then repeat the attempt to colour.
- When simplification yields an empty graph then select colours, that is, registers

Effect of Spilling



W1: def x in B2, def x in B3, use x in B4, Use x in B5
W2: def x in B5, use x in B6
W3: def y in B2, use y in B4
W4: def y in B1, use y in B3

Effect of Spilling



Colouring the Graph(selection)

Repeat

V= pop(stack).

Colours_used(v)= colours used by neighbours of V.

Colours_free(v)=all colours - Colours_used(v).

Colour (V) = any colour in Colours_free(v).

Until stack is empty

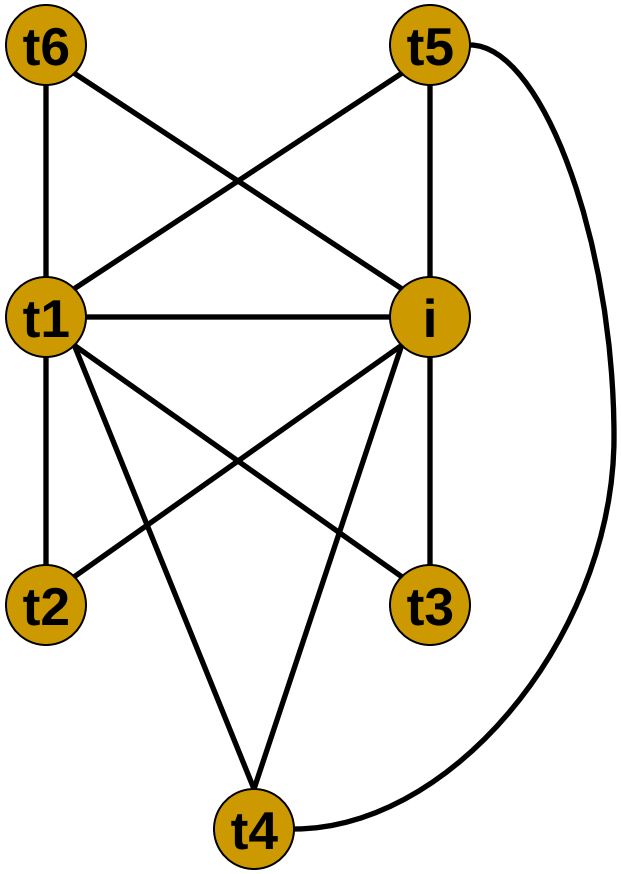
- Convert the colour assigned to a symbolic register to the corresponding real registers name in the code.

A Complete Example

```
1.          t1 = 202
2.          i = 1
3.  L1:     t2 = i>100
4.          if t2 goto L2
5.          t1 = t1-2
6.          t3 = addr(a)
7.          t4 = t3 - 4
8.          t5 = 4*i
9.          t6 = t4 + t5
10.        *t6 = t1
11.         i = i+1
12.         goto L1
13. L2:
```

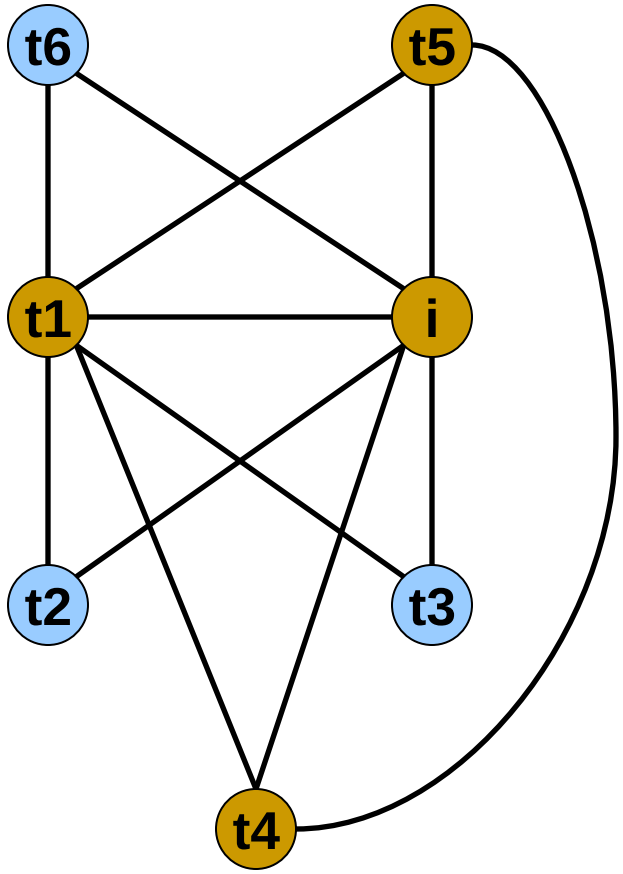
variable	live range
t1	1-10
i	2-11
t2	3-4
t3	6-7
t4	7-9
t5	8-9
t6	9-10

A Complete Example

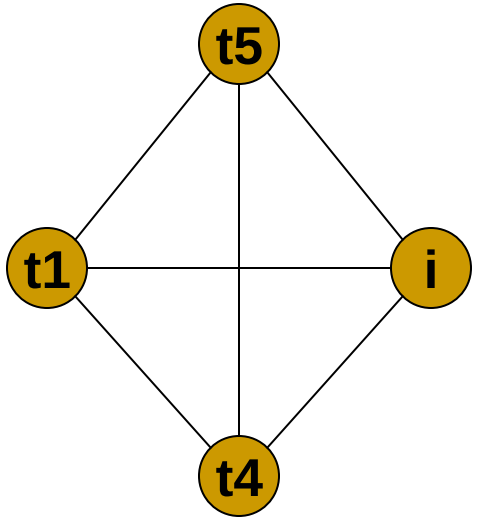


variable	live range
t1	1-10
i	2-11
t2	3-4
t3	6-7
t4	7-9
t5	8-9
t6	9-10

A Complete Example



Assume 3 registers. Nodes t6,t2, and t3 are first pushed onto a stack during reduction.



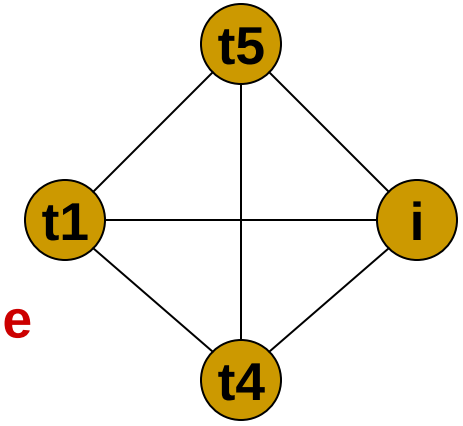
This graph cannot be reduced further. Spilling is necessary.

A Complete Example

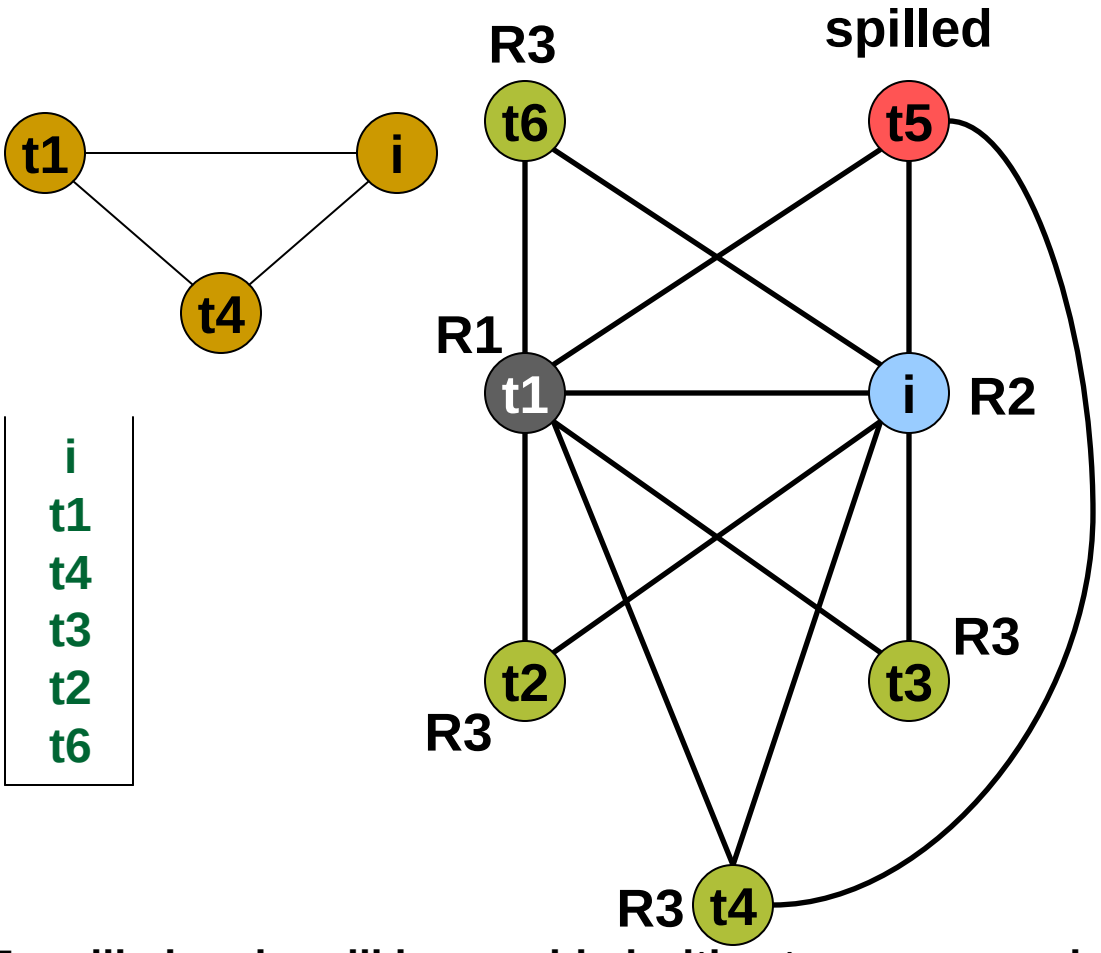
Node V	Cost(v)	deg(v)	$h_0(v)$
t1	31	3	10
i	41	3	14
t4	20	3	7
t5	20	3	7

```
1.        t1 = 202
2.        i = 1
3.  L1:   t2 = i>100
4.        if t2 goto L2
5.        t1 = t1-2
6.        t3 = addr(a)
7.        t4 = t3 - 4
8.        t5 = 4*i
9.        t6 = t4 + t5
10.      *t6 = t1
11.       i = i+1
12.       goto L1
13. L2:
```

t1: 1+(1+1+1)*10 = 31
i : 1+(1+1+1+1)*10 = 41
t4: (1+1)*10 = 20
t5: (1+1)*10 = 20
t5 will be spilled. Then the graph can be coloured.



A Complete Example



i
t1
t4
t3
t2
t6

```
1.          R1 = 202
2.          R2 = 1
3.   L1:    R3 = i>100
4.          if R3 goto L2
5.          R1 = R1 - 2
6.          R3 = addr(a)
7.          R3 = R3 - 4
8.          t5 = 4*R2
9.          R3 = R3 + t5
10.        *R3 = R1
11.         R2 = R2+1
12.         goto L1
13. L2:
```

t5: spilled node, will be provided with a temporary register during code generation

Drawbacks of the Algorithm

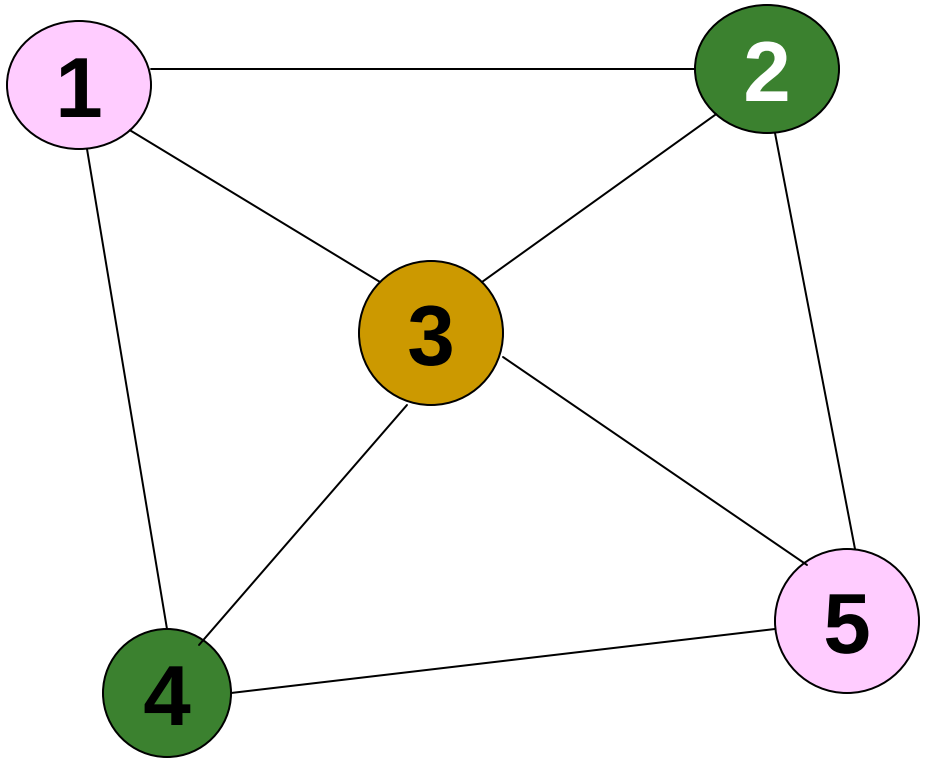
- Constructing and modifying interference graphs is very costly as interference graphs are typically huge.
- For example, the combined interference graphs of procedures and functions of gcc in mid-90's have approximately 4.6 million edges.

Some modifications

- Careful coalescing: Do not coalesce if coalescing increases the degree of a node to more than the number of registers
- Optimistic colouring: When a node needs to be spilled, put it into the colouring stack instead of spilling it right away
  - spill it only when it is popped and if there is no colour available for it
  - this could result in colouring graphs that need spills using Chaitin's technique.

A 3-colourable graph which is not 3-coloured by colouring heuristic, but coloured by optimistic colouring

Example



Say, 1 is chosen for spilling. Push it onto the stack, and remove it from the graph. The remaining graph (2,3,4,5) is 3-colourable. Now, when 1 is popped from the colouring stack, there is a colour with which 1 can be coloured. It need not be spilled.