Alocação de Registradores

Guido Araújo
guido@ic.unicamp.br

Coalescing

- Eliminar MOVES redundantes usando o IG
  - Se não existirem arestas entre os nós de uma instrução MOVE ela pode ser eliminada
- Os nós fonte e destino do MOVE são unidos (*coalesced*) em um só
- A aresta do novo nó é a união das arestas dos dois anteriores

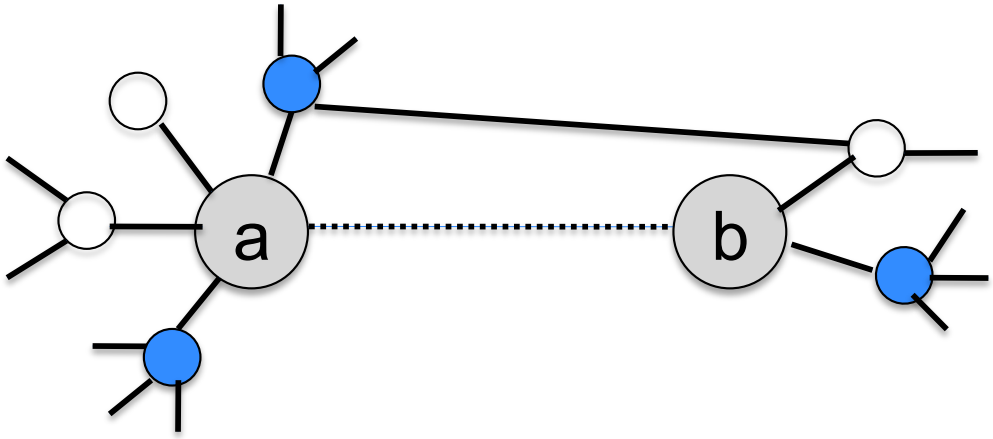
Coalescing

- O efeito é sempre benéfico?
  - Qualquer instrução MOVE sem arestas no IG poderia ser eliminada
  - Pode tornar o processo de alocação mais complicado
    - Por que?
- O nó resultante é mais restritivo que os anteriores
  - Seu grau aumenta
  - Pode ser tornar >= K
- Um grafo k-colorível antes do coalescing pode ser tornar não k-colorível após uma operação de coalescing

Coalescing

- Devemos tomar cuidados
  - Executar coalescing somente quando for seguro
  - Temos duas estratégias:
- Briggs:
  - *a* e *b* podem ser unidos se o nó resultante *ab* tiver menos do que K vizinhos com grau significativo (>=K)
  - Garante que o grafo continua k-colorível. Por que?
  - Após a simplificação remover todos os nós não-significativos, sobram menos do que K vizinhos para o nó *ab*
  - Logo, ele pode ser removido

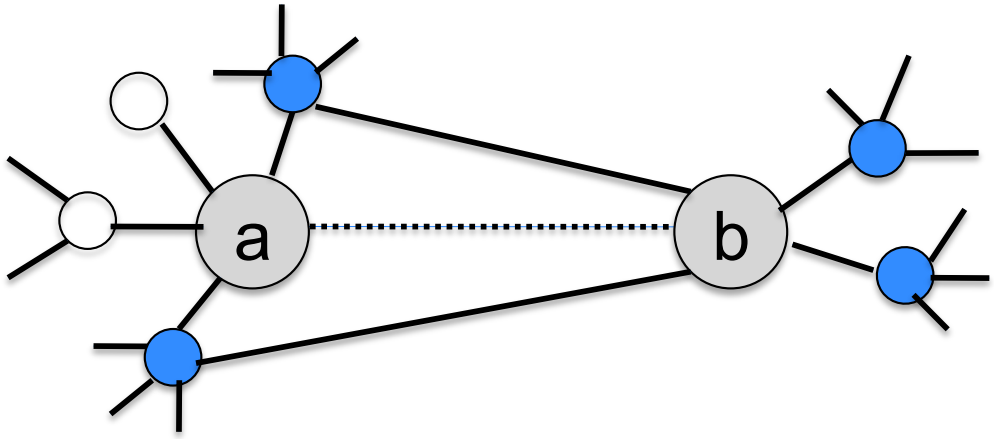
Spill em Potencial



	# non-sig.	# sig.
a	2	2
b	1	1
a&b	3	3 < K

Pode coalesce

- K = 4



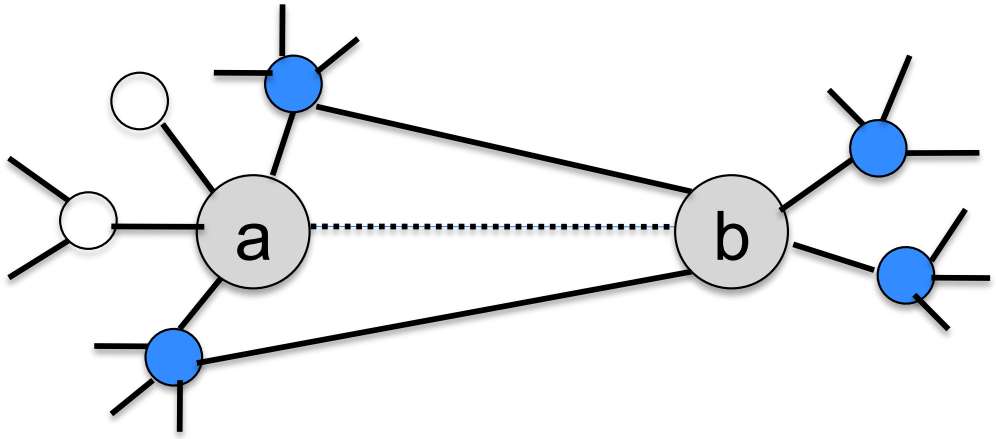
	# non-sig.	# sig.
a	2	2
b	0	2
a&b	2	4 >= K

Não pode coalesce

Coalescing

- George:
  - *a* e *b* podem ser unidos se para cada vizinho *t* de *a:*
    - *t* interfere com *b*
    - ou *t* tem grau insignificante (<K)
  - Por que é segura?
  - Seja S o conjunto de vizinhos insignificantes de a em G
  - Sem o coalescing, todos poderiam ser removidos, gerando um grafo G1
  - Fazendo o coalescing, todos os nós de S também poderão ser removidos, criando G2
  - G2 é um subgrafo de G1, onde o nó *ab* corresponde ao *b*
  - *G2* é no mímino tão fácil para colorir quanto G1

Spill em Potencial

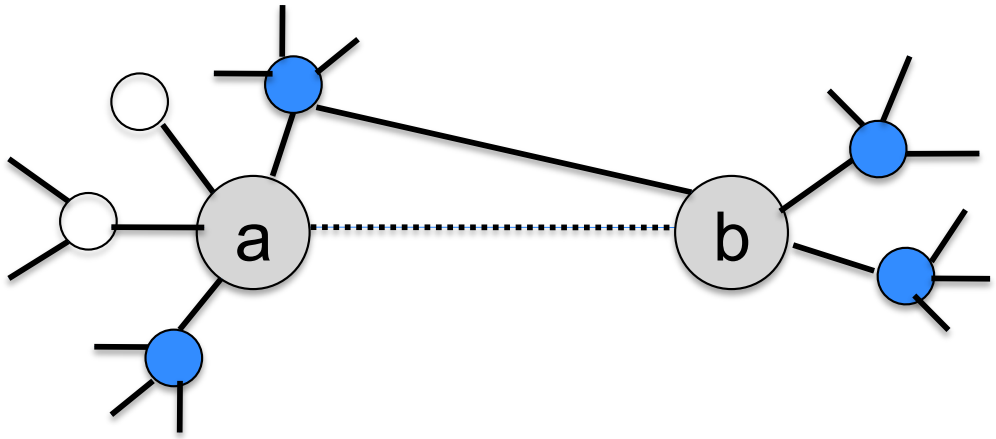


Olha somente os sigs.

1. interfere com b ou
2. é não significativo

Pode coalesce

- K = 4



Olha somente os sigs.

1. interfere com b ou
2. é não significativo

Não pode coalesce

Coalescing

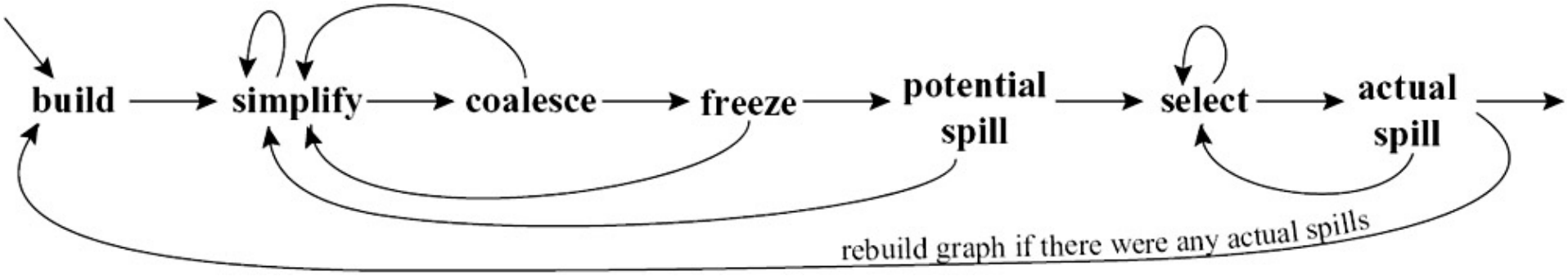
- São estratégias conservativas
- Podem sobrar MOVES que poderiam ser removidos
- Ainda assim, é melhor do que fazer spill!

Fases da Alocação com Coalescing

- Build:
  - Construir o IG
  - Categorizar os nós em move-related e move-unrelated
- Simplify:
  - Remover os nós não significativos (grau < K), um de cada vez
- Coalesce:
  - Faça o coalesce conservativo no grafo resultante do passo anterior
  - Com a redução dos graus, é provável que apareça mais oportunidades para o coalescing
  - Quando um nó resultante não é mais move-related ele fica disponível para a próxima simplificação

Fluxo com Coalescing

- Simplify, coalesce e spill são intercalados até que o grafo esteja vazio.

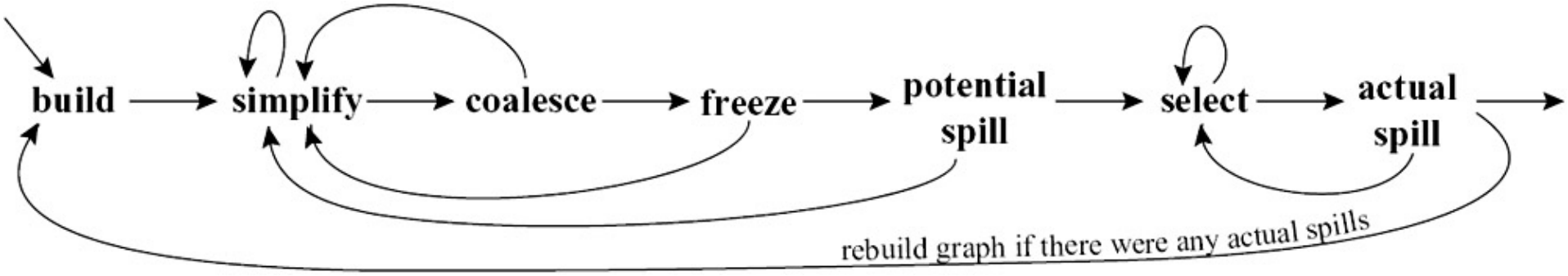


Fases da Alocação com Coalescing

- Freeze:
  - Executado quando nem o simplify nem o coalescing podem ser aplicados
  - Procura nós move-related de grau baixo.
  - Congela os moves desses nós. Eles passam a ser candidatos para simplificação
- Spill:
  - Se não houver nós de grau baixo, selecionamos um nó com grau significativo para spill
  - Coloca-se esse nó na pilha
- Select:
  - Desempilhar todos os nós e atribuir cores

Fluxo com Coalescing

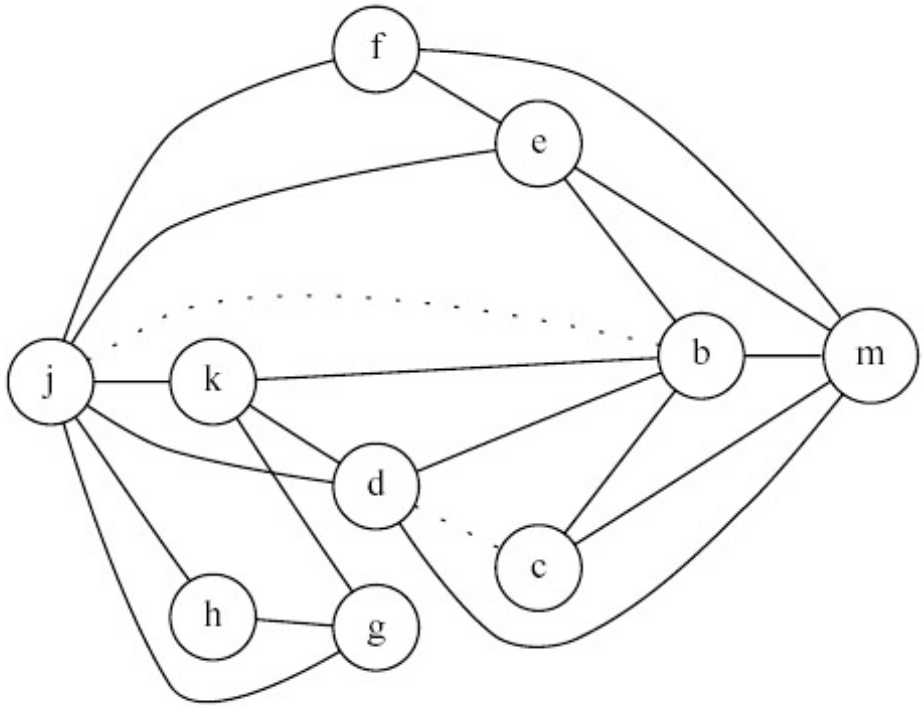
- Simplify, coalesce e spill são intercalados até que o grafo esteja vazio.



Retomando o Exemplo

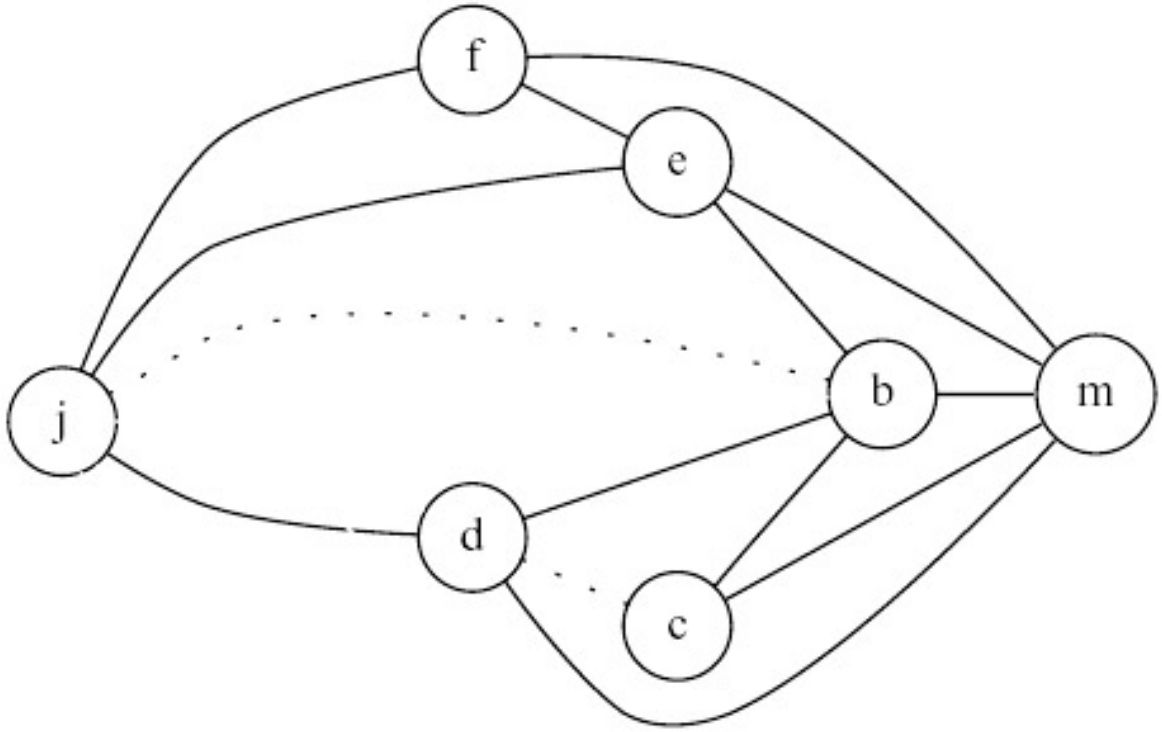
- Suponha que temos 4 registradores
- Agora somente nós não relacionados a MOVE podem ser candidatos no simplify

```
live-in: k j
        g := mem[j+12]
        h := k - 1
        f := g * h
        e := mem[j+8]
        m := mem[j+16]
        b := mem[f]
        c := e + 8
        d := c
        k := m + 4
        j := b
live-out: d k j
```



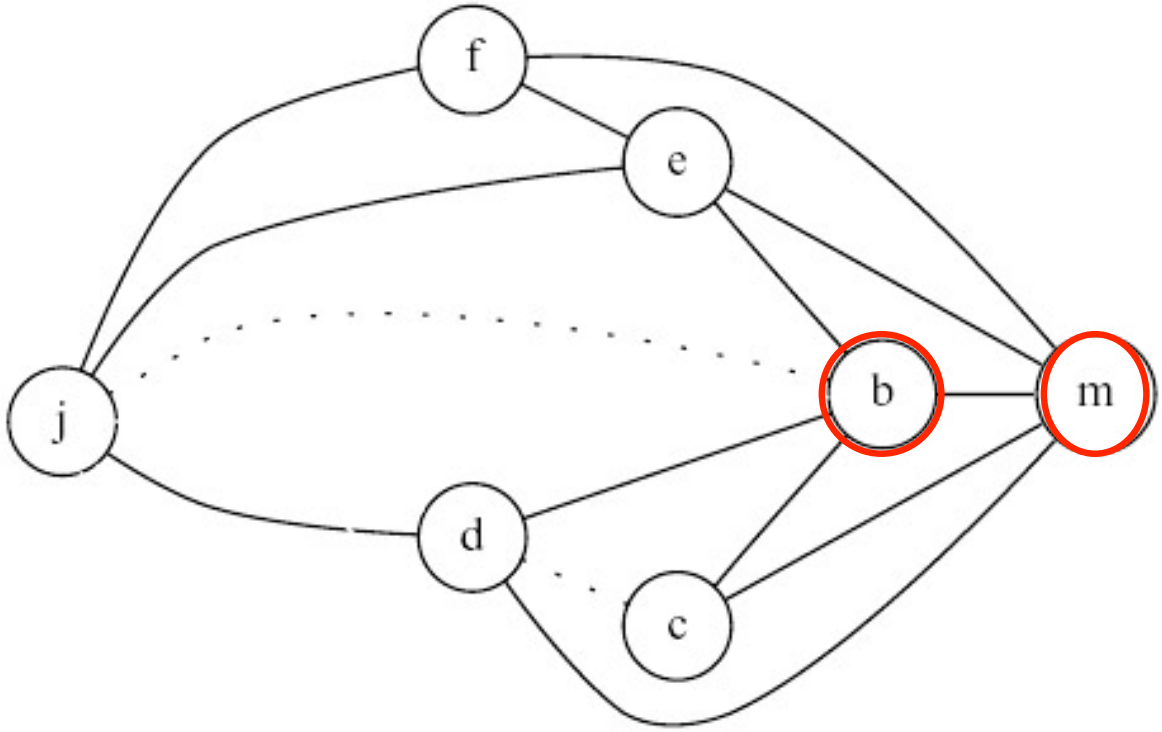
Exemplo de Coalescing

- Removendo h, g , k



Exemplo de Coalescing

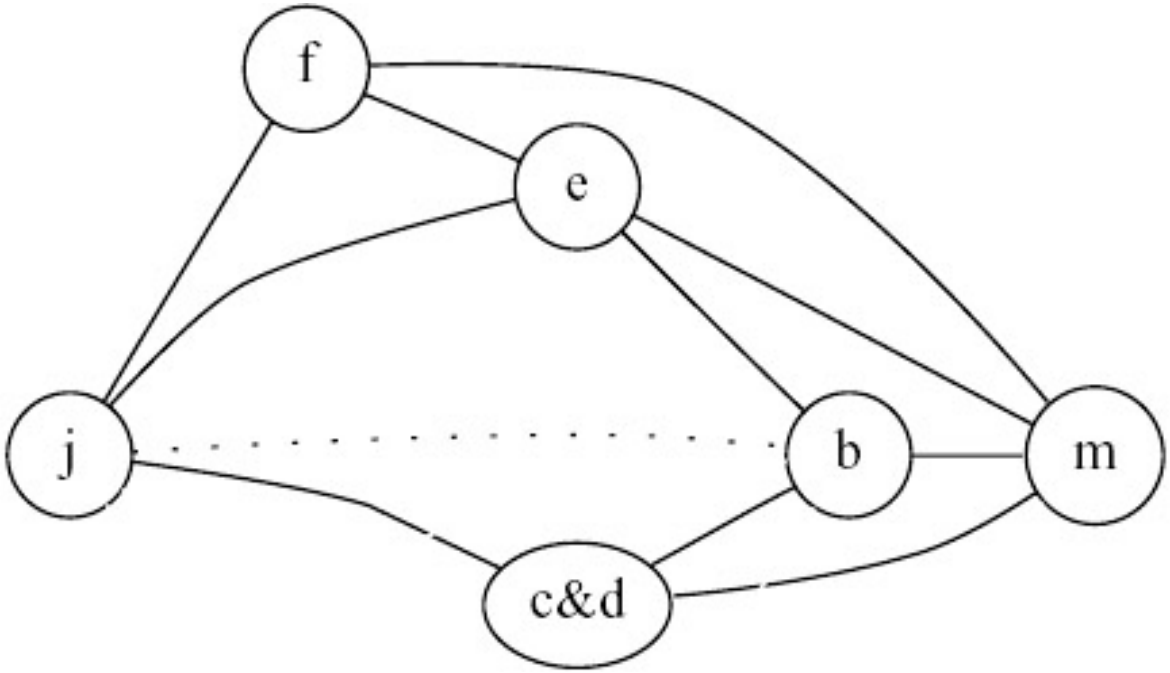
- Invocando coalescing para c e d?



Exemplo de Coalescing

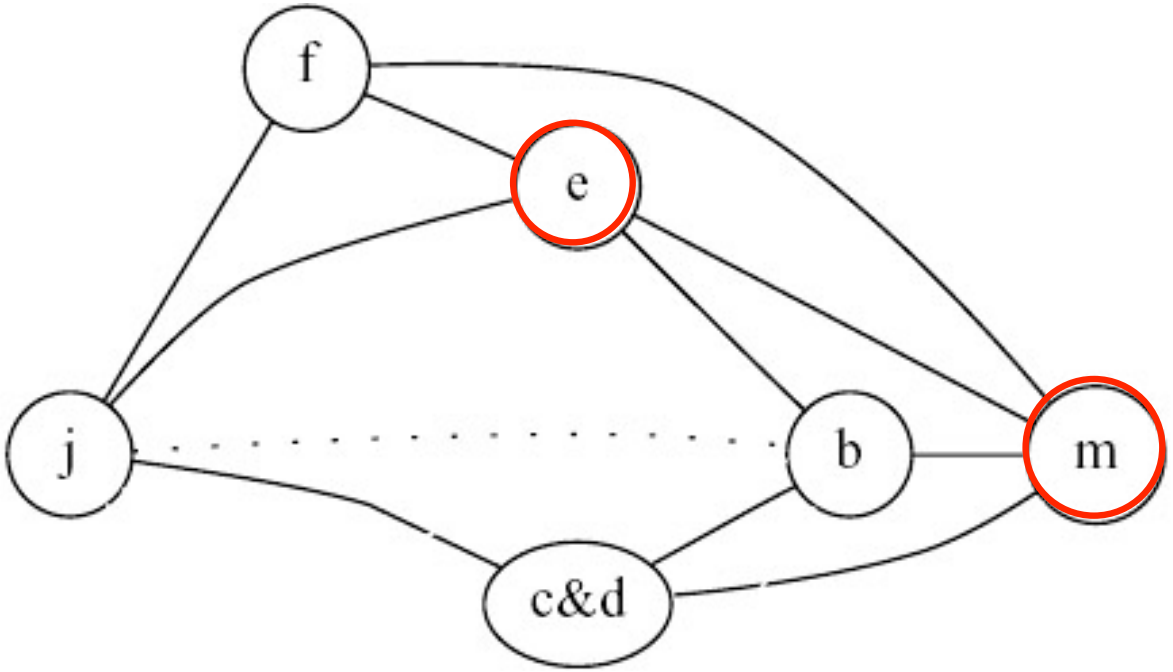
- c e d podem ser unidos

  c&d tem 2 vizinhos (b e m) com grau significativo (>=K)



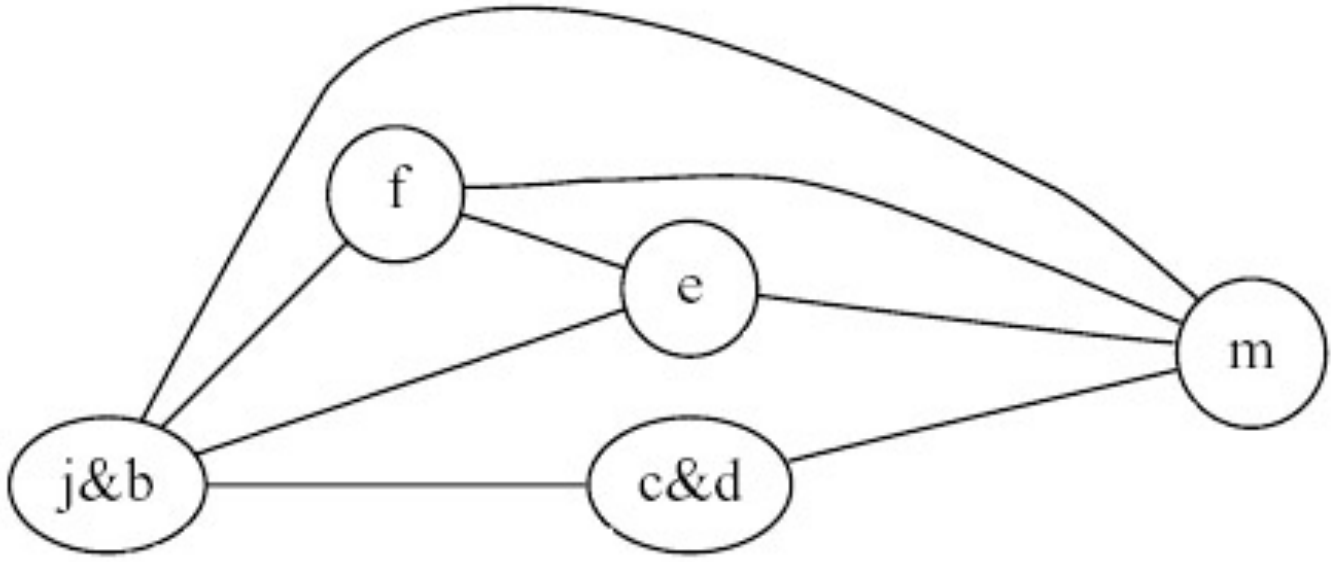
Exemplo de Coalescing

- b e j também podem ser unidos
  - j&b tem 2 vizinhos (m e e) com grau significativo (>=K)



Exemplo de Coalescing

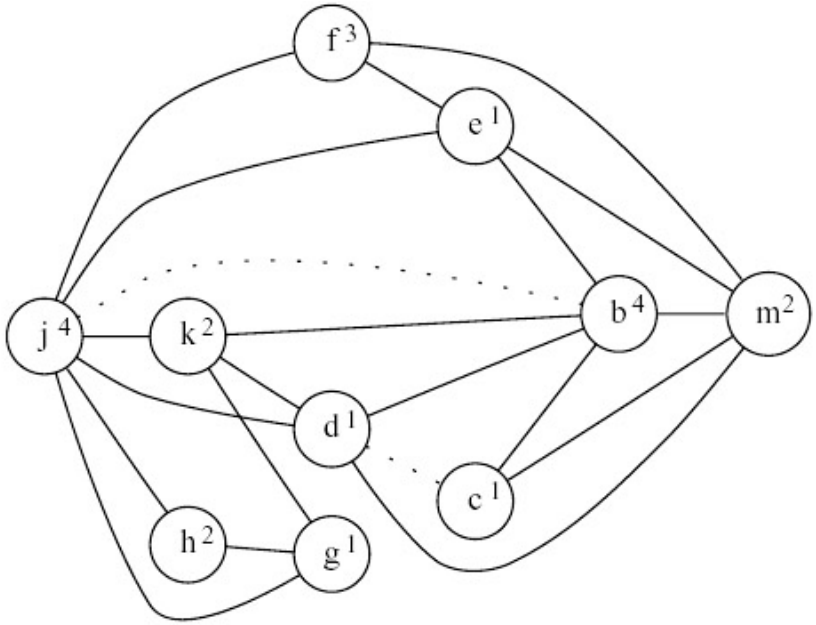
- b e j também podem ser unidos
  - j&b tem 1 vizinho com grau significativo (>=K)
- Agora simplify termina o trabalho …



Exemplo de Coalescing

- Alocação final:

stack	coloring
e	1
m	2
f	3
j&b	4
c&d	1
k	2
h	2
g	1



Spilling com Coalescing

- Solução simples:
  - Descartar todos os coalescing feitos quando recomeçar o Build
- Mais eficiente:
  - Conservar os coalescing feitos antes do primeiro *potencial spill*
  - Descarta os subsequentes

Coalescing de Spills

- Muitos registradores => poucos spills
- Poucos registradores => vários spills
  - Aumenta o tamanho dos registros de ativação (AR)
  - Ex. Pentium: 6 registradores
- Transformações/Otimizações
  - Podem gerar mais temporários
- O frame da função pode ficar grande

Coalescing de Spills

- Instruções MOVE envolvendo valores que sofreram spill
  - a ← b implica em:
    - t ← M[b]
    - M[a] ← t
  - Caro e ainda cria mais um valor temporário
- Muitos dos valores que sofrem spill não estão vivos simultaneamente
- Podemos usar a mesma técnica que para registradores!

Coalescing de Spills

- Coloração com coalescing para os *spills*

1. Use o liveness para construir um IG para os spills
2. Enquanto houver spills sem interferência e com MOVE
   - Una esses nós (Coalescing)
3. Use simplify e select para colorir o grafo

Coalescing de Spills

3. Use simplify e select para colorir o grafo
   - Não existe spill nesta coloração
   - Simplify vai retirando o nó de menor grau até o fim
   - Select vai escolhendo a menor cor possível
     - Sem limite, pois não temos limite para o tamanho do frame
4. As cores correspondem a posições do frame da função

- Fazer **antes** da reescrita do código

Pré-coloração

- Alguns nós do IG podem ser pré-coloridos
  - Temporários associados ao FP, SP, registradores de passagem de argumentos
  - Permanentemente associados aos registradores físicos
  - Cores pré-definidas e únicas
  - Podem ser reaproveitados no *select* e *coalesce*
    - Desde que não interfiram com o outro valor
    - Ex. Um registrador de passagem de parâmetro pode servir como temporário no corpo da função

Pré-coloração

- Podem ser unidos no coalescing com outros nós não pré-coloridos
- Simplify os trata como tendo grau "infinito"
  - Não devem ir para a pilha
  - Não devem sofrer spill
- O algoritmo executa *simplify*, *select* e *spill* até sobrarem somente nós pré-coloridos

Pré-coloração

- Podem ser copiados para temporários
  - Suponha que r7 seja um callee-save register

(a)

enter: $\text{def}(r_7)$

$\vdots$

exit: $\text{use}(r_7)$

(b)

enter: $\text{def}(r_7)$

$t_{231} \leftarrow r_7$

$\vdots$

$r_7 \leftarrow t_{231}$

exit: $\text{use}(r_7)$

Exemplo

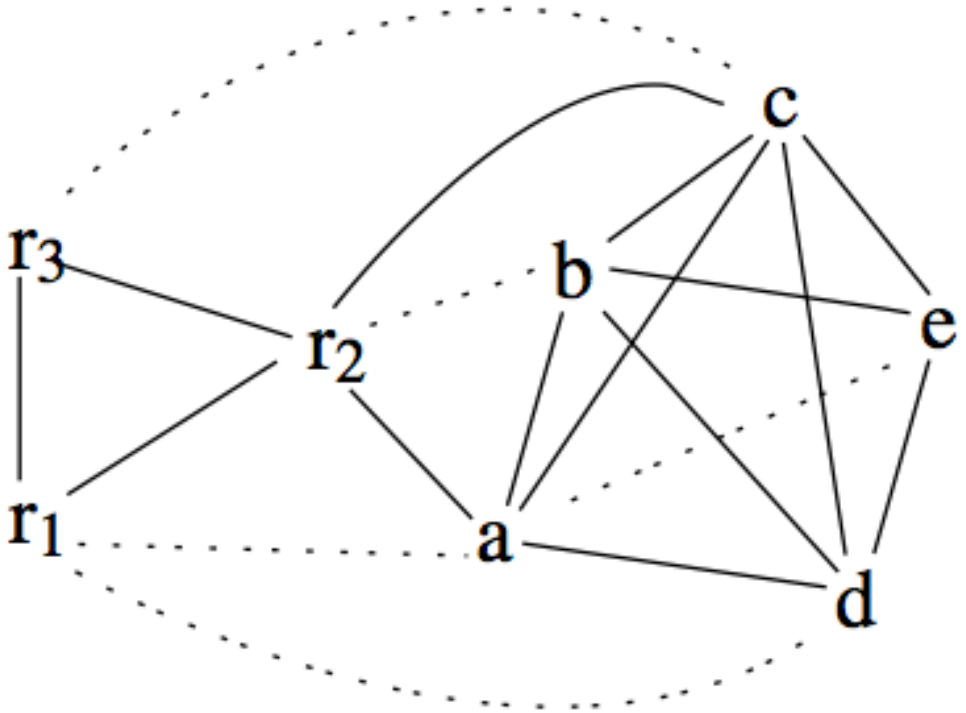
- Três registradores:
  - r1 e r2 caller-save
  - r3 callee-save

```
int f(int a, int b) {
   int d=0;
   int e=a;
   do {d = d+b;
       e = e-1;
   } while (e>0);
   return d;
}
```

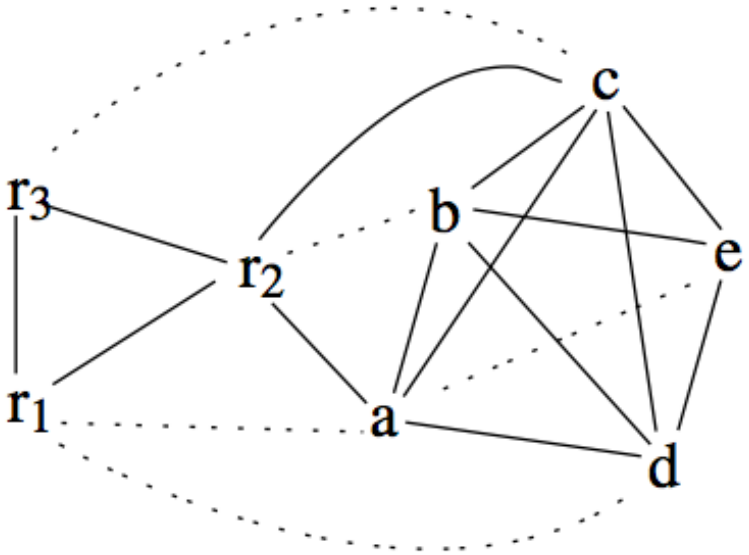
```
enter:  c ← r3
        a ← r1
        b ← r2
        d ← 0
        e ← a
loop:   d ← d + b
        e ← e − 1
        if e > 0 goto loop
        r1 ← d
        r3 ← c
        return          (r1, r3 live out)
```

Exemplo

- IG para o programa em (b)
  - K = 3
  - Tem oportunidades para simplify e spill?



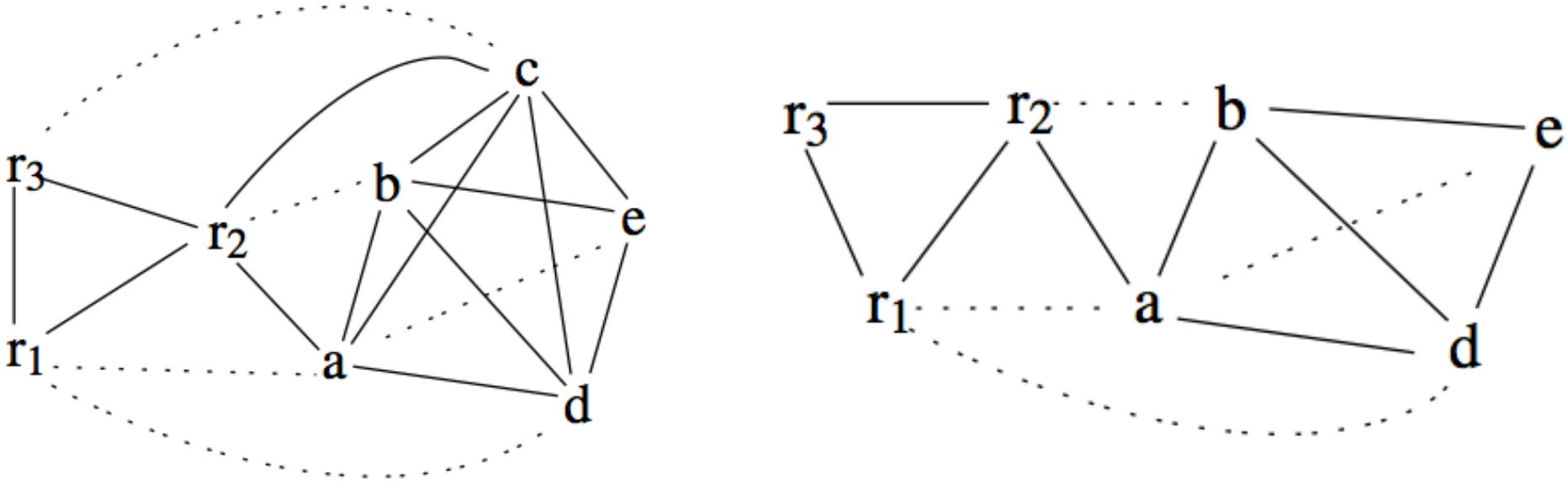
Exemplo



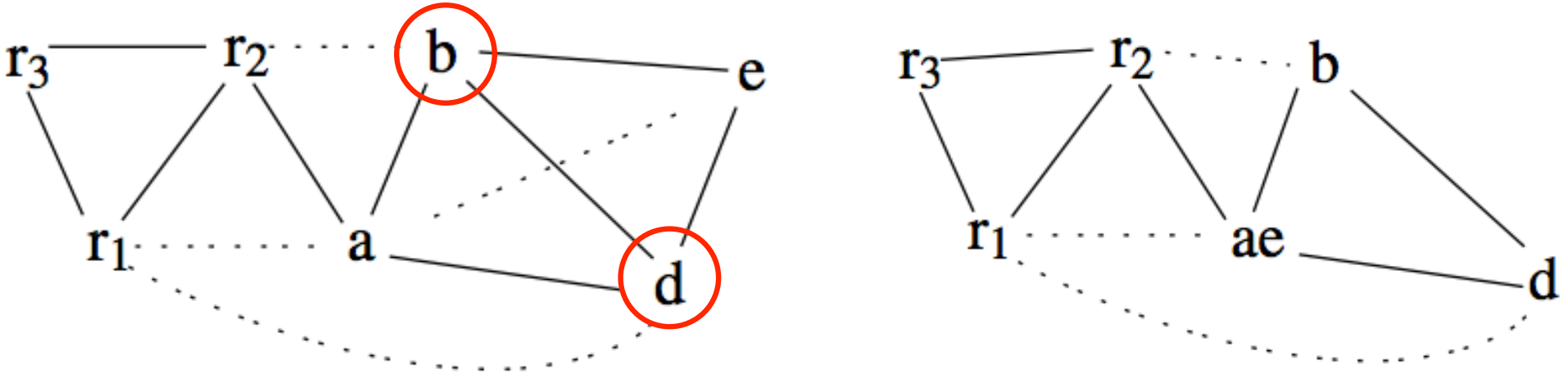

```
enter:  c ← r3
        a ← r1
        b ← r2
        d ← 0
        e ← a
loop:   d ← d + b
        e ← e − 1
        if e > 0 goto loop
        r1 ← d
        r3 ← c
return          (r1, r3 live out)
```

Node	Uses+Defs outside loop		Uses+Defs within loop		Degree		Spill priority
a	(2	+ 10 ×	0)	/	4	=	0.50
b	(1	+ 10 ×	1)	/	4	=	2.75
c	(2	+ 10 ×	0)	/	6	=	0.33
d	(2	+ 10 ×	2)	/	4	=	5.50
e	(1	+ 10 ×	3)	/	3	=	10.33

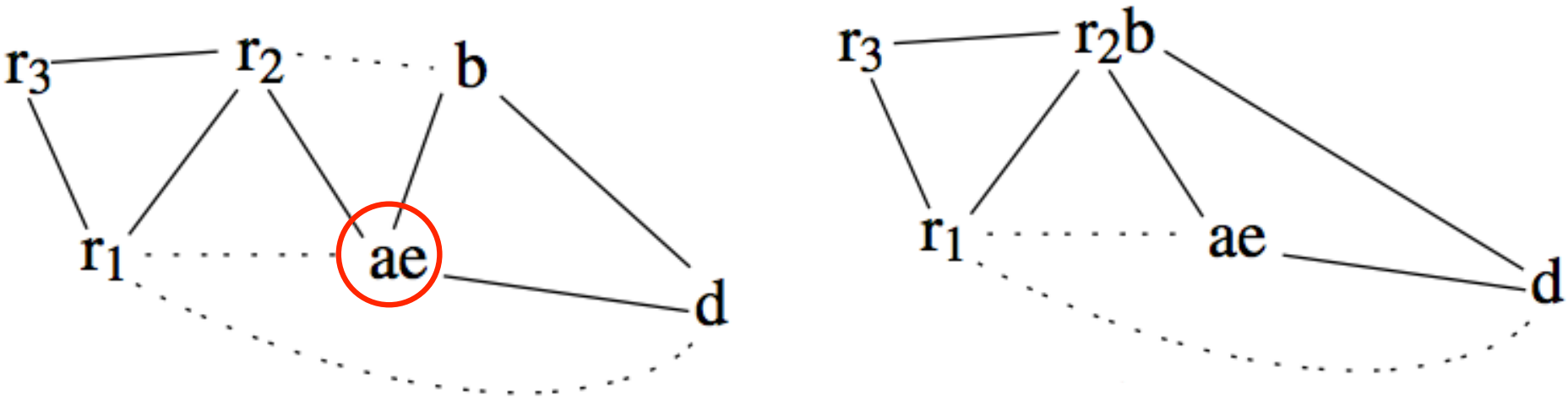
Após spill de c



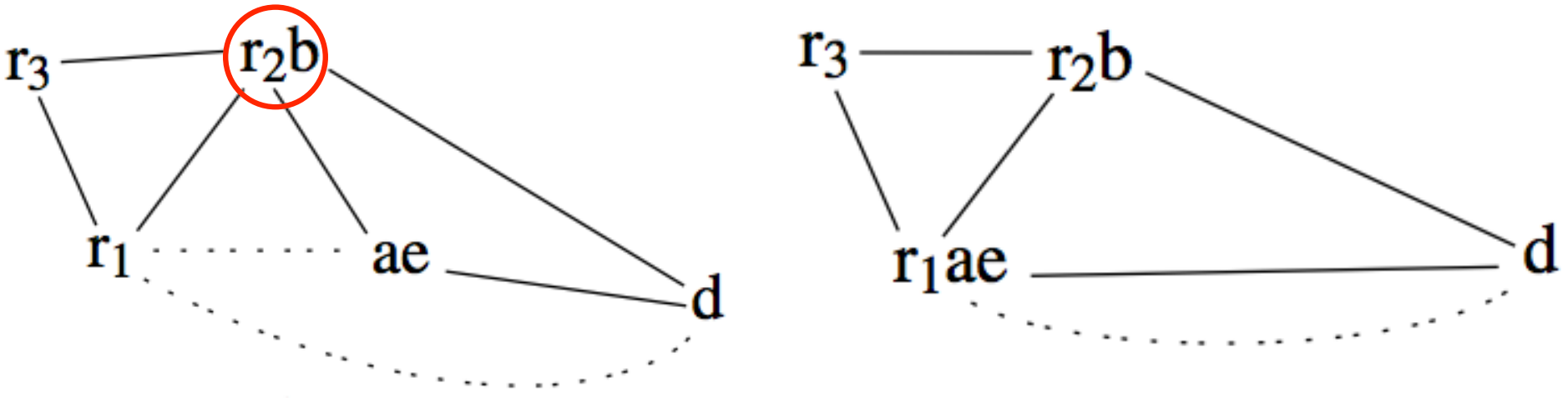
Coalesce de a&e



Coalesce de b&r2

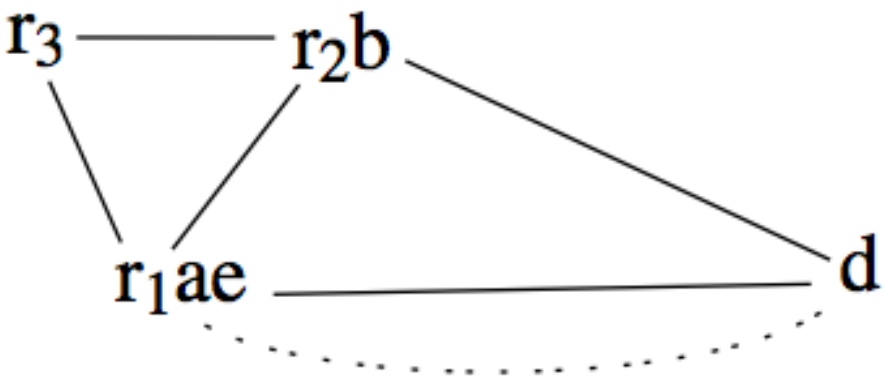


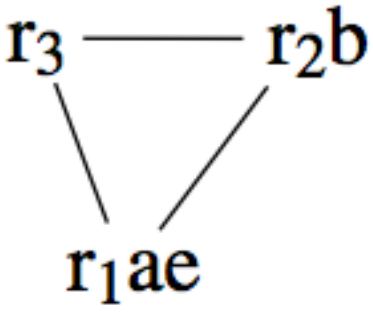
Coalesce ae&r1



Não pode coalesce r1ae&d

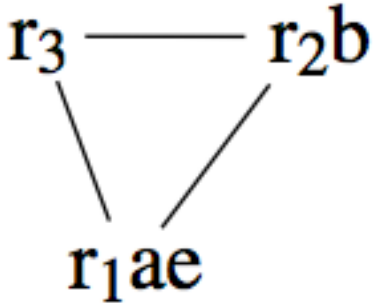
- Existe interferência entre eles
- Simplify d





Only pre-colored nodes

- Starting popping from stack
- Somente d e c na pilha



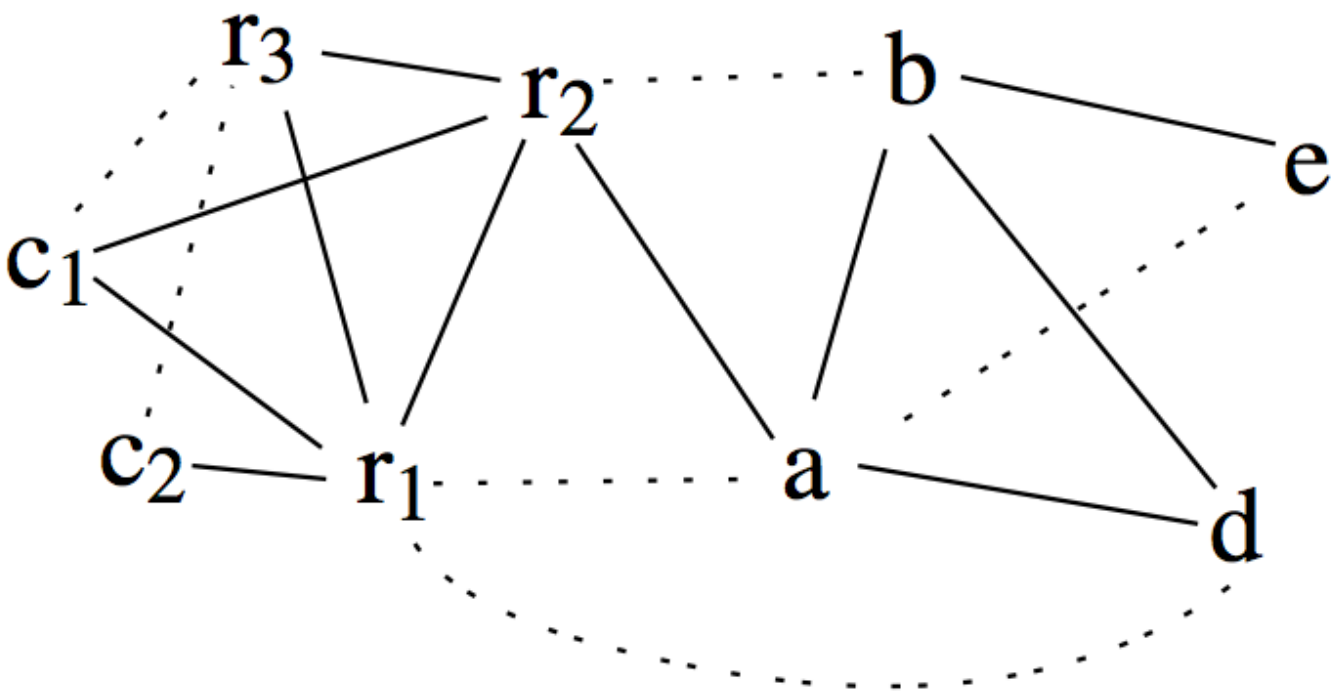
Rebuild

- Código após reescrita do spill de c

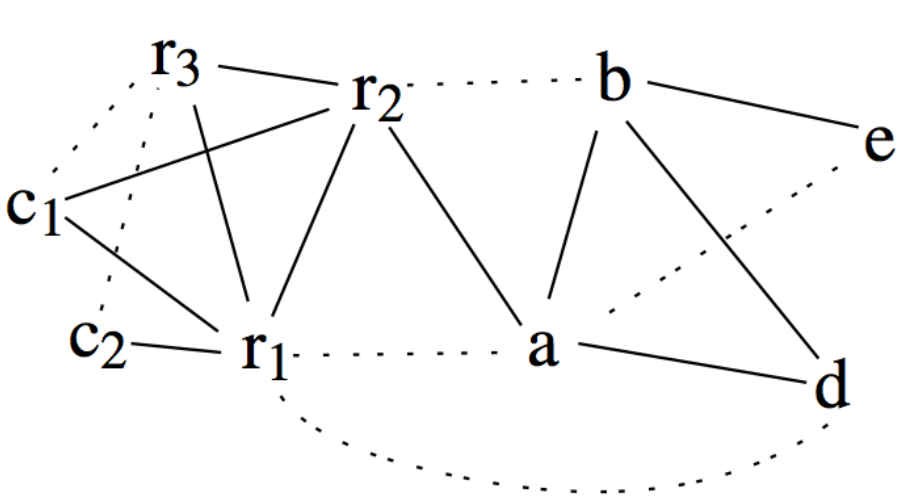
```
enter:  c1 ← r3
        M[cloc] ← c1
        a  ← r1
        b  ← r2
        d  ← 0
        e  ← a
loop:   d  ← d + b
        e  ← e − 1
        if e > 0 goto loop
        r1 ← d
        c2 ← M[cloc]
        r3 ← c2
        return
```

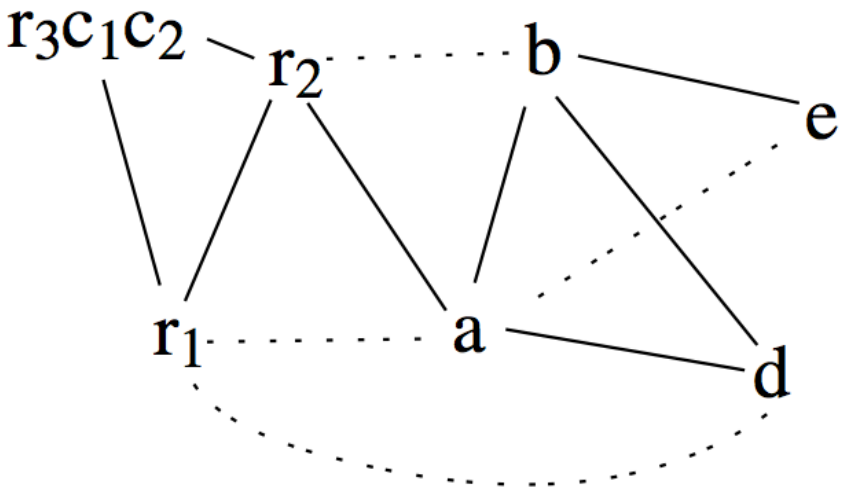
Rebuild IG

- Novo IG

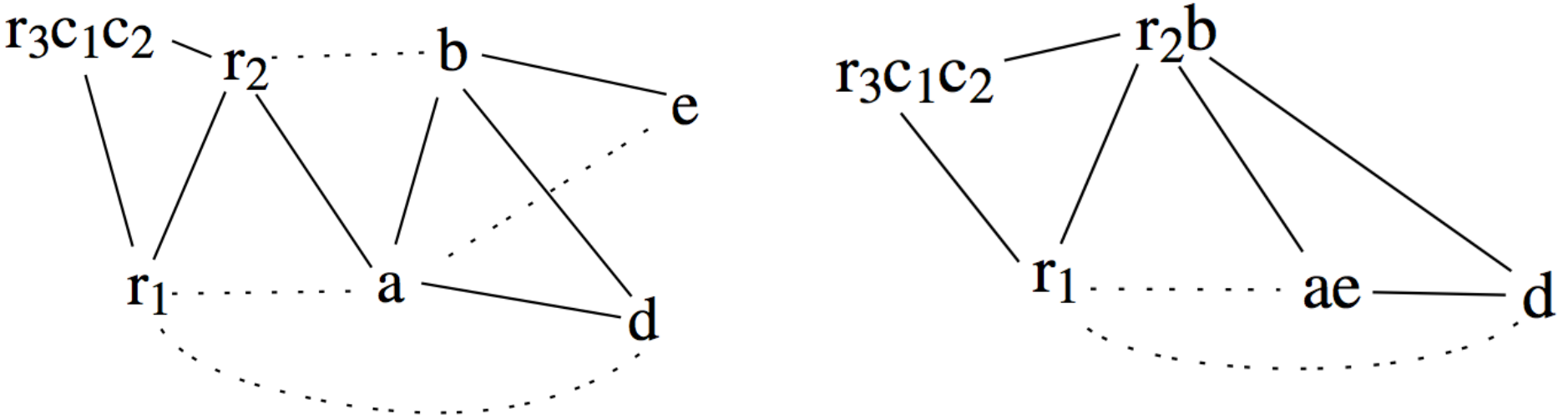


Coalesce c1&r3 e c2&c1r3

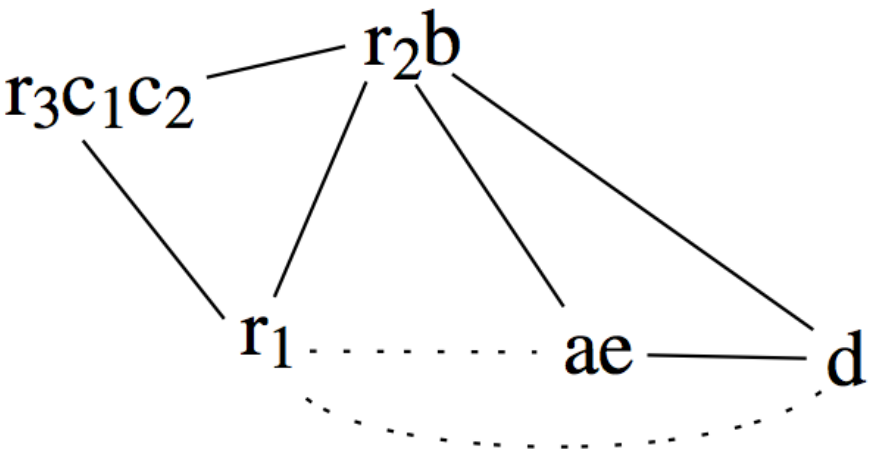


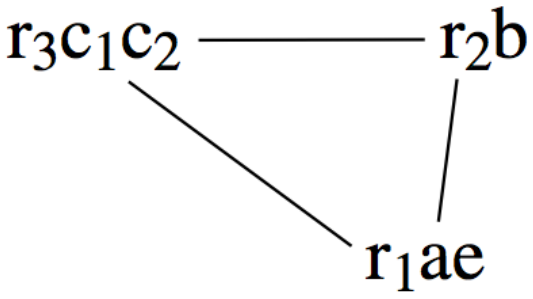


Coalesce a&e e b&r2



Coalesce ae&r1 and simplify d





Stack popping and select color

- Somente d foi para pilha

Node	Color
a	r_1
b	r_2
c	r_3
d	r_3
e	r_1

Alocando

- Reescreve o programando alocando registradores

```
enter:  r3 ← r3
        M[cloc] ← r3
        r1 ← r1
        r2 ← r2
        r3 ← 0
        r1 ← r1
loop:   r3 ← r3 + r2
        r1 ← r1 − 1
        if r1 > 0 goto loop
        r1 ← r3
        r3 ← M[cloc]
        r3 ← r3
        return
```

Limpando

- Remove todos os MOVES com fonte e destino iguais

```
enter:  r3 ← r3
        M[cloc] ← r3
        r1 ← r1
        r2 ← r2
        r3 ← 0
        r1 ← r1
loop:   r3 ← r3 + r2
        r1 ← r1 − 1
        if r1 > 0 goto loop
        r1 ← r3
        r3 ← M[cloc]
        r3 ← r3
        return
```

```
enter:  M[cloc] ← r3
        r3 ← 0
loop:   r3 ← r3 + r2
        r1 ← r1 − 1
        if r1 > 0 goto loop
        r1 ← r3
        r3 ← M[cloc]
        return
```