Compilation avancée et optimisation de programmes

Alain Darte

CNRS, Compsys
Laboratoire de l'Informatique du Parallélisme
École normale supérieure de Lyon

Back-end code optimizations

Outline

What is register allocation?

Input:

- program (intermediate representation) with scalar variables.
- fixed instruction schedule.

Goal: assign variables to storage locations in

- a pool of limited resources: *registers*.
- a pool of unlimited resources: *memory*.

Memory hierarchy constraints:

- Register accesses: faster.
- Memory accesses: slower and higher power consumption.

What is register allocation?

Input:

- program (intermediate representation) with scalar variables.
- fixed instruction schedule.

Goal: assign variables to storage locations in

- a pool of limited resources: *registers*.
- a pool of unlimited resources: *memory*.

Memory hierarchy constraints:

- Register accesses: faster.
- Memory accesses: slower and higher power consumption.

☛ Prefer register accesses and register-to-register moves to memory transfers, i.e., try to minimize loads & stores.

NP-completeness of Chaitin et al.

Problem 1 (Chaitin-like register allocation)

Instance: Program P, number k of available registers.
Question: Can each variable of P be mapped to one of the k registers so that variables with interfering live ranges are mapped to different registers?

☛ NP-complete with a reduction from graph-k-colorability.

- variable ⇔ vertex;
- interferences between variables ⇔ edge;
- variable assignment ⇔ graph coloring.

G. Chaitin, M. Auslander, A. Chandra, J. Cocke, M. Hopkins, and P. Markstein. Register allocation via coloring. *Computer Languages*, 6:47–57, January 1981.

NP-completeness of Chaitin et al.

Problem 1 (Chaitin-like register allocation)

Instance: Program P, number k of available registers.
Question: Can each variable of P be mapped to one of the k registers so that variables with interfering live ranges are mapped to different registers?

☛ NP-complete with a reduction from graph-k-colorability.

- variable ⇔ vertex;
- interferences between variables ⇔ edge;
- variable assignment ⇔ graph coloring.

G. Chaitin, M. Auslander, A. Chandra, J. Cocke, M. Hopkins, and P. Markstein. Register allocation via coloring. *Computer Languages*, 6:47–57, January 1981.

Traditional (but wrong) interpretation: “Register allocation is NP-complete because graph coloring is NP-complete.”

The real life: more constraints & more freedom

Many architectural subtleties:

- load/store architecture or not.
- specific registers (sp, fp, r0), variable affinities (auto-inc), register pairing (64 bits ops), calling conventions, etc.
- distributed register banks, register aliasing, etc.

The real life: more constraints & more freedom

Many architectural subtleties:

- load/store architecture or not.
- specific registers (sp, fp, r0), variable affinities (auto-inc), register pairing (64 bits ops), calling conventions, etc.
- distributed register banks, register aliasing, etc.

More rules of the games:

- insert loads and stores: *spilling*.
- add register-to-register moves: *live-range splitting*.
- delete moves: *coalescing*.
- don't store, recompute: *rematerialization*.

The real life: more constraints & more freedom

Many architectural subtleties:

- load/store architecture or not.
- specific registers (sp, fp, r0), variable affinities (auto-inc), register pairing (64 bits ops), calling conventions, etc.
- distributed register banks, register aliasing, etc.

More rules of the games:

- insert loads and stores: *spilling*.
- add register-to-register moves: *live-range splitting*.
- delete moves: *coalescing*.
- don't store, recompute: *rematerialization*.

NP-completeness of Chaitin et al. ☛ use of heuristics interleaving all aspects: register assignment, spilling, splitting, coalescing.

Global register allocation: related work

- **Chaitin-like graph coloring** Chaitin et al. (1981), Briggs et al. (1989-1992), George&Appel (1996), Smith et al. (2004), etc.
- **Use program structure** Chow&Hennessy (1990), Callahan&Koblenz (1991), Knobe&Zadeck (1992), Norris&Pollock (1994), etc.
- **More involved live-range splitting** Bergner et al. (1997), Cooper&Simpson (1998), Lueh et al. (2000), etc.
- **"Exact" formulation (ILP or multi-flow)** Goodwin&Wilken (1996), Appel&George (2001), Fu&Wilken (2002), Koes&Goldstein (2006), Barik et al. (2007), Colombet et al. (2011), etc.
- **JIT linear-scan** Poletto&Sarkar (1999), Wimmer&Mössenböck (2005), Sarkar&Barik (2007), etc.
- **Two-phases (SSA-based) register allocation** Bouchez et al., Brisk, Hack, Pereira&Palsberg (2005-...).

Example: iterated register coalescing

Chaitin et al. (1981), Briggs-Cooper-Torczon (1994), Appel-George (2001), ...

Simplify remove a non-move-related vertex with degree $< k$

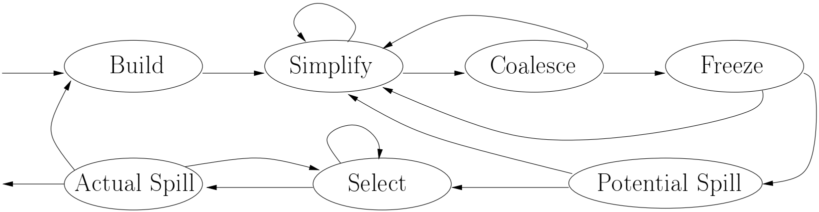
Coalesce merge (conservatively: Briggs/George rules) move-related vertices

Freeze give up coalescing some moves

Potential spill remove a vertex and push it on a stack

Select pop a vertex and assign a color

Actual spill if no color is found, really insert load/store



Example: iterated register coalescing

Chaitin et al. (1981), Briggs-Cooper-Torczon (1994), Appel-George (2001), ...

Simplify remove a non-move-related vertex with degree $< k$

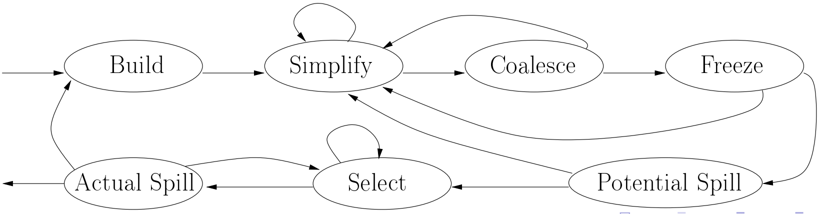
Coalesce merge (conservatively: Briggs/George rules) move-related vertices

Freeze give up coalescing some moves

Potential spill remove a vertex and push it on a stack

Select pop a vertex and assign a color — see powerpoint slides

Actual spill if no color is found, really insert load/store



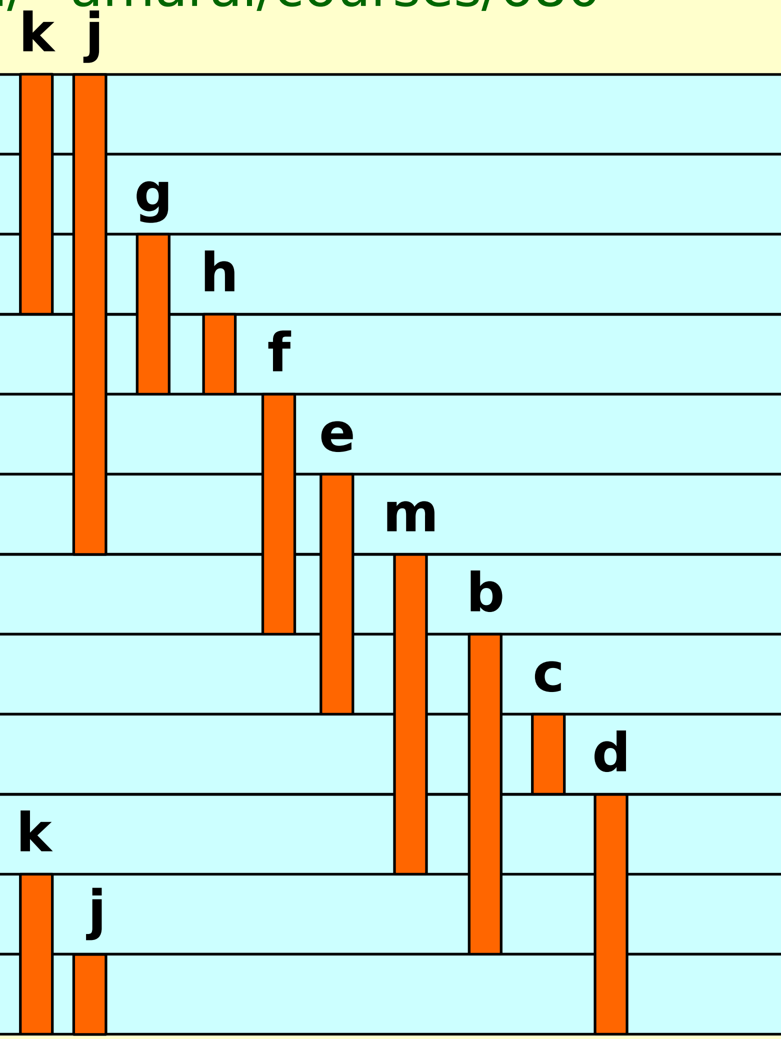
Borrowed from J. N. Amaral, slightly modified

http://www.cs.ualberta.ca/~amaral/courses/680

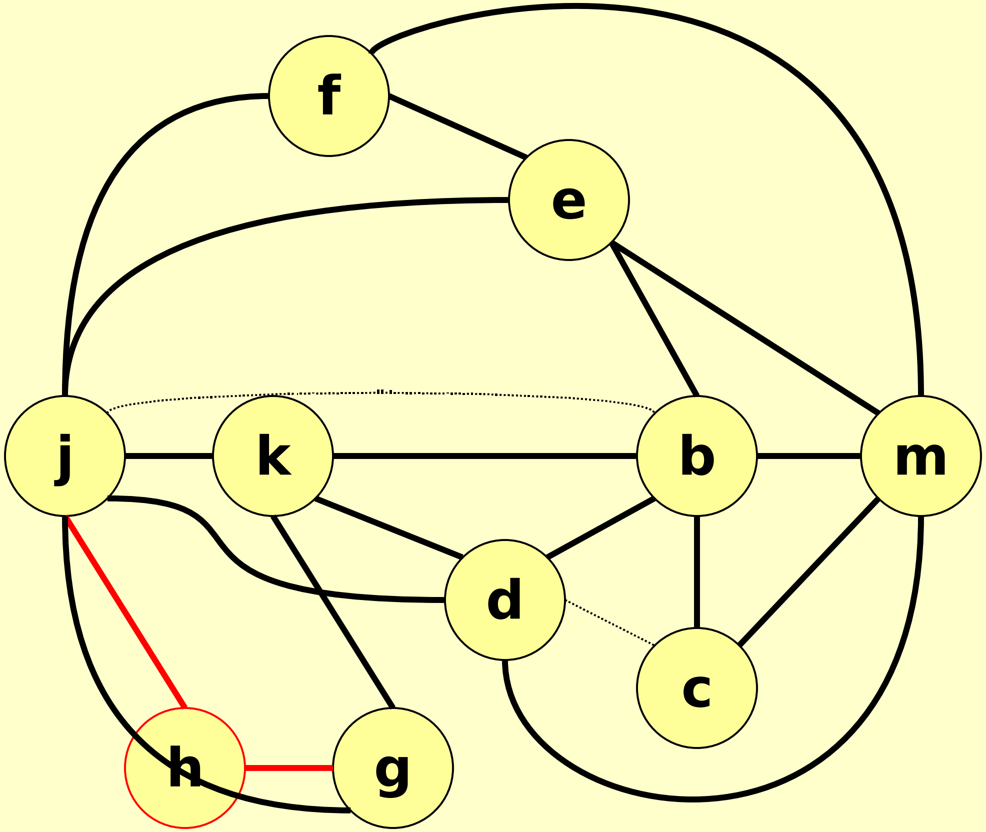
LIVE-IN: k j

```
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



Example: Simplify (K=4)

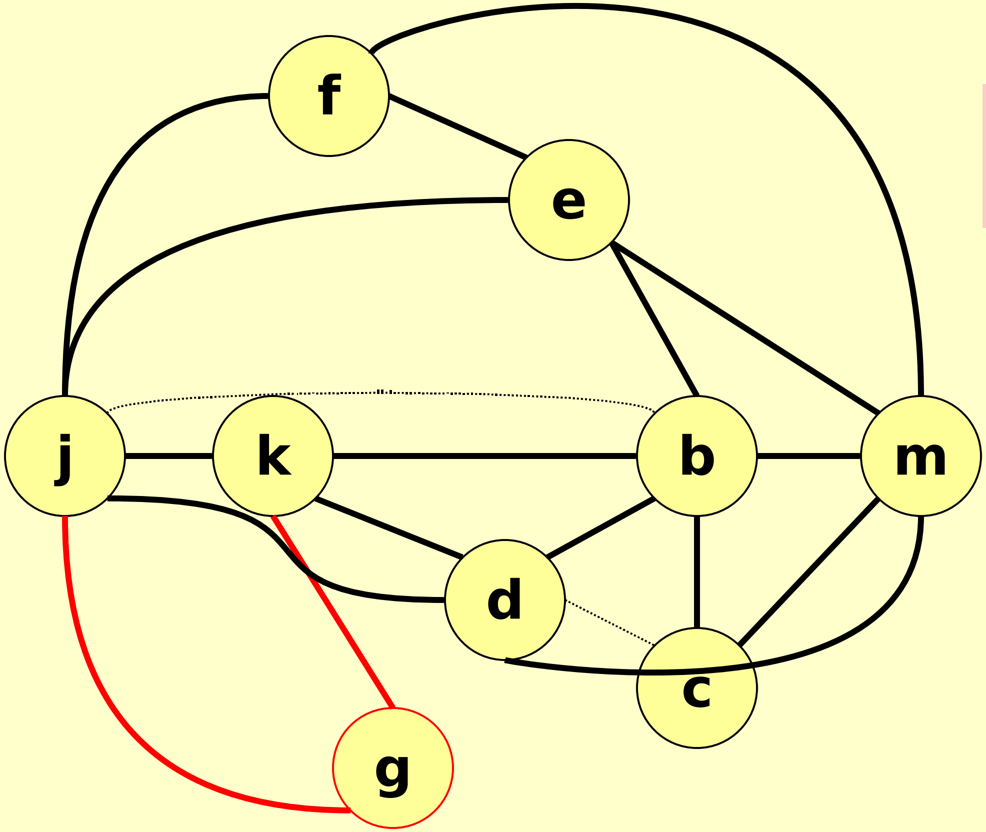


stack

(h,no-spill)

(Appel, pp. 237)

Example: Simplify (K=4)

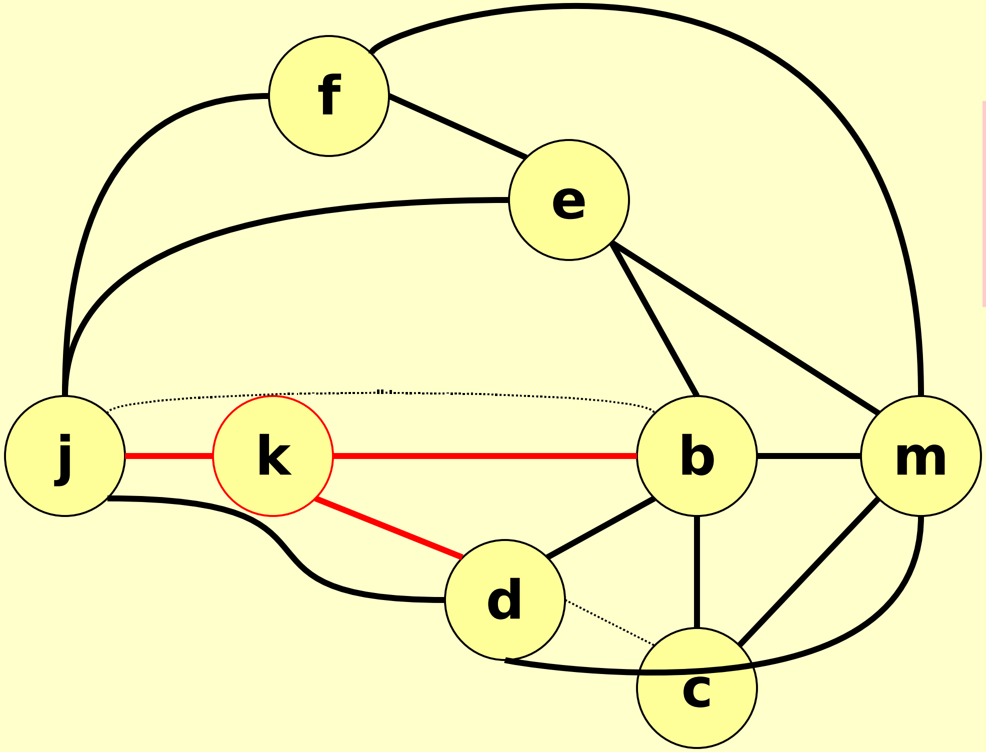


stack

(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Simplify (K=4)

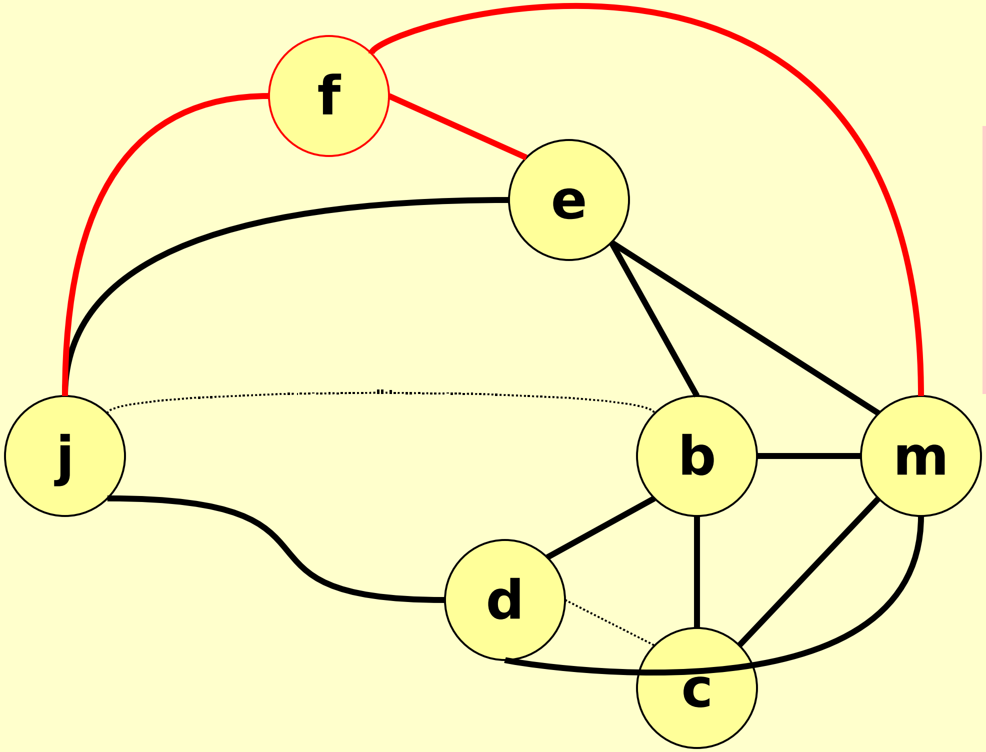


stack

(k, no-spill)
(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Simplify (K=4)

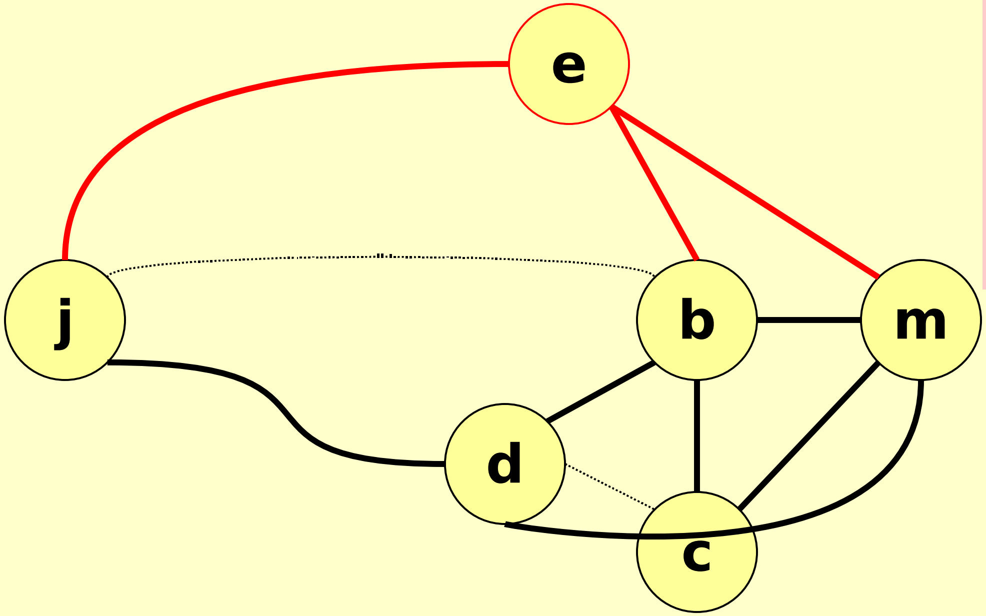


stack

(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Simplify (K=4)



stack

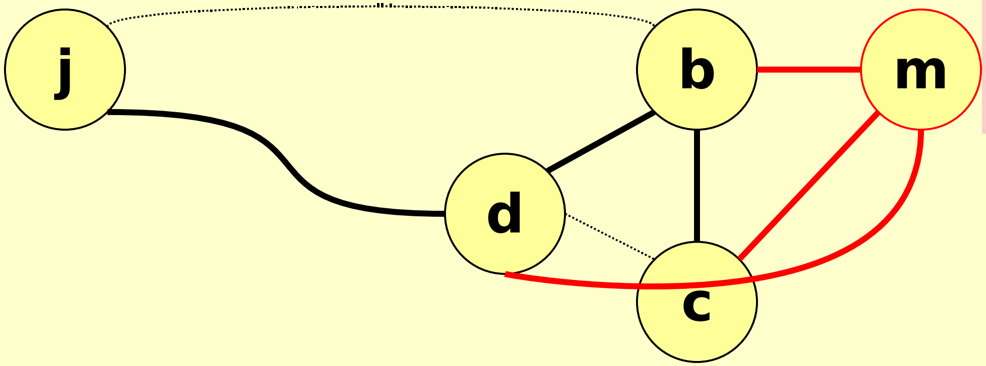
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Simplify (K=4)

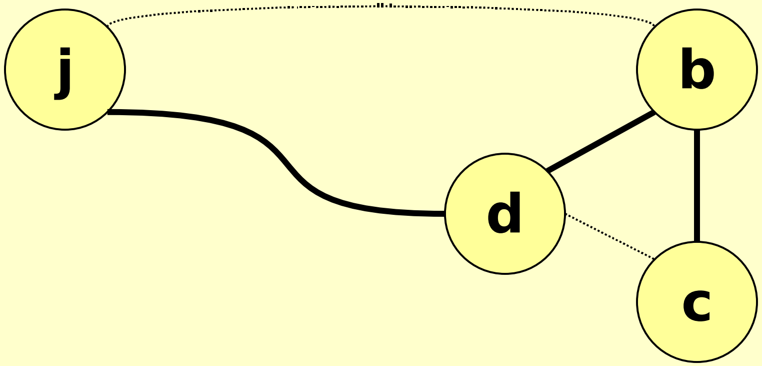
stack

(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)



(Appel, pp. 237)

Example: Coalesce (K=4)



stack

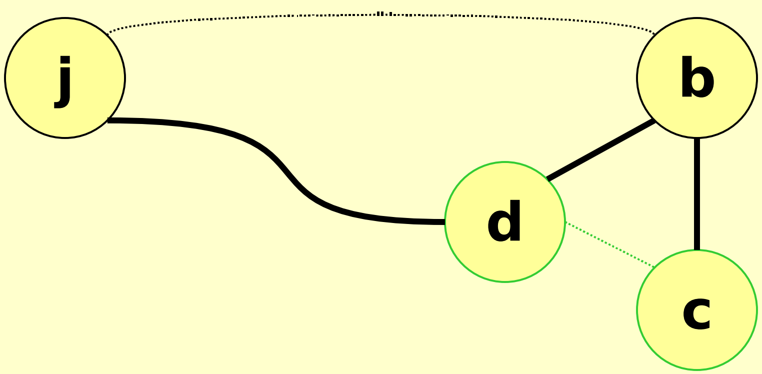
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

Why can’t we simplify?

Cannot simplify move-related nodes.

(Appel, pp. 237)

Example: Coalesce (K=4)

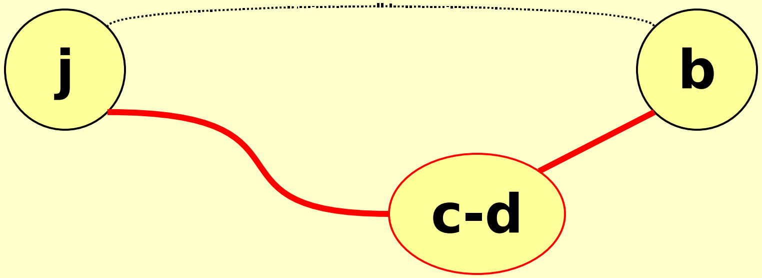


stack

(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Simplify (K=4)

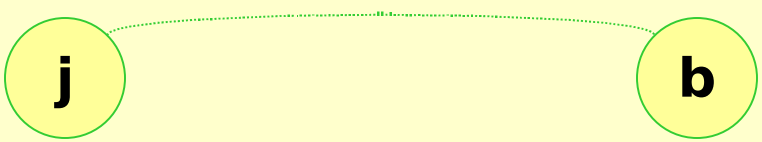


stack

(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Coalesce (K=4)



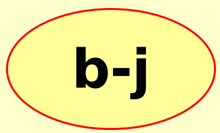
stack

(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example:
Simplify (K=4) greedy-4-colorable

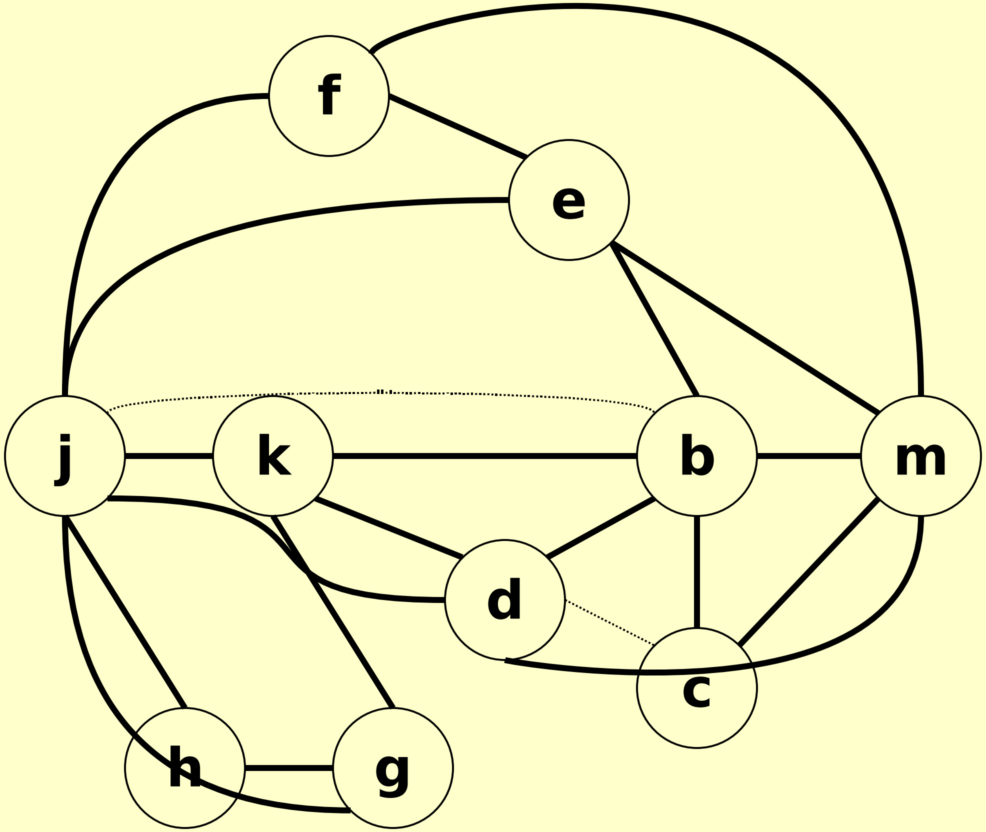
stack



(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Select (K=4)



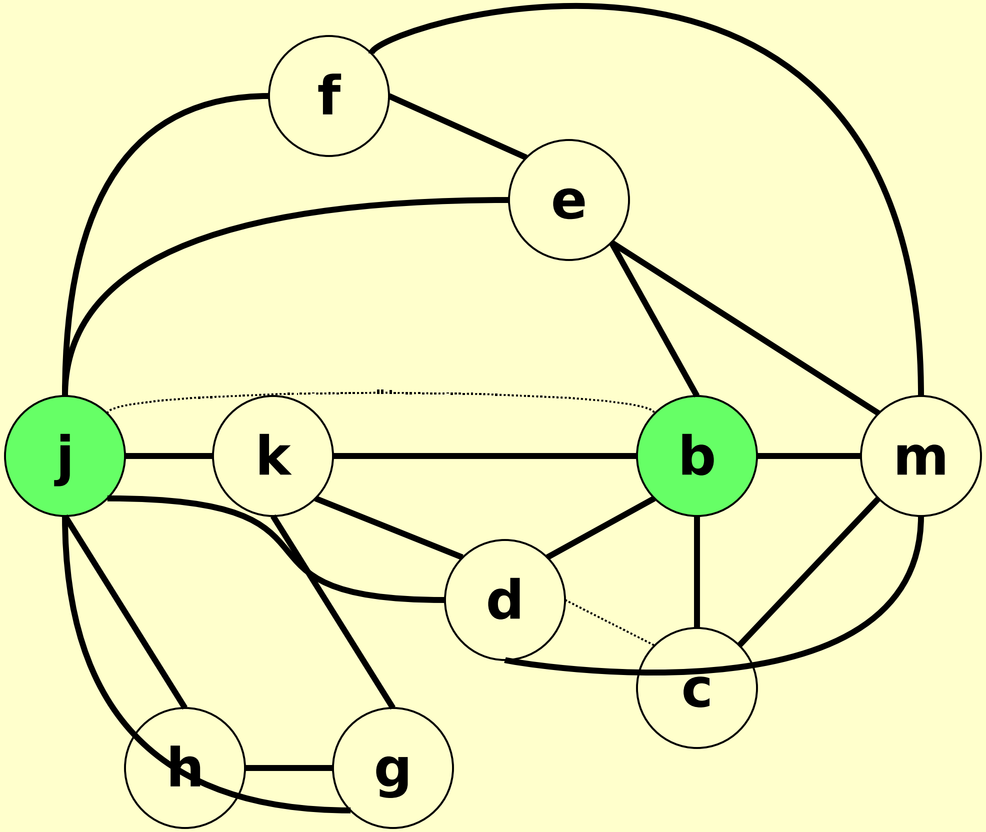
stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1
R2
R3
R4

(Appel, pp. 237)

Example: Select (K=4)



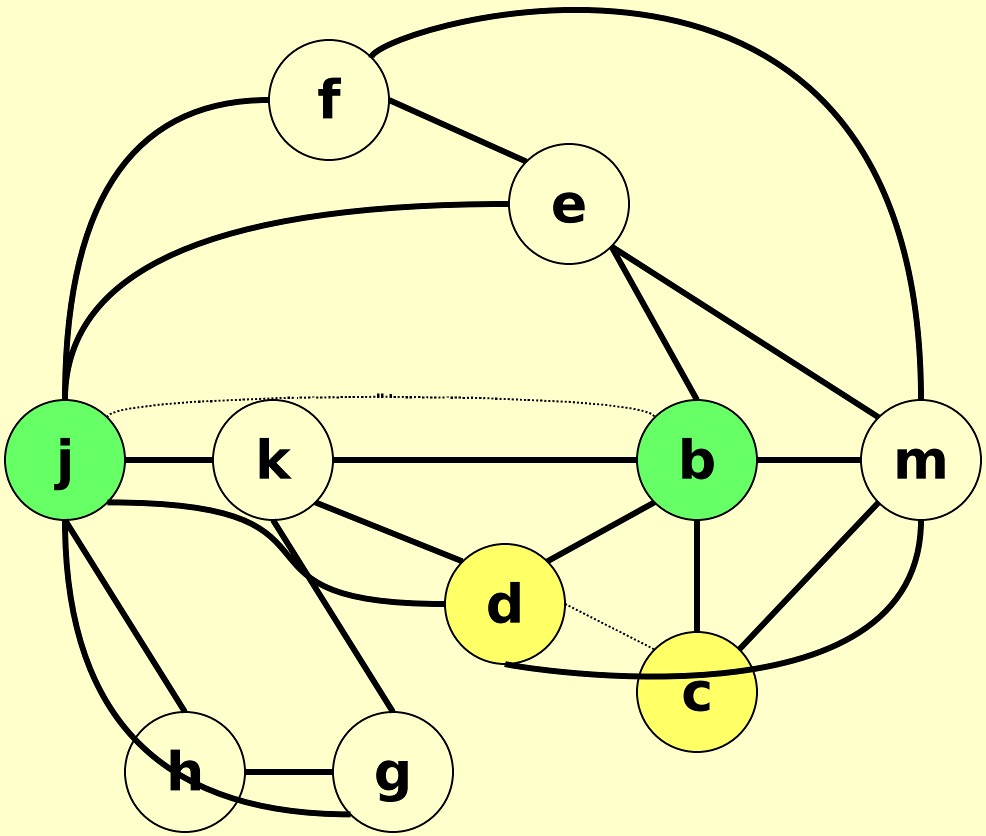
stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1
R2
R3
R4

(Appel, pp. 237)

Example: Select (K=4)



stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1

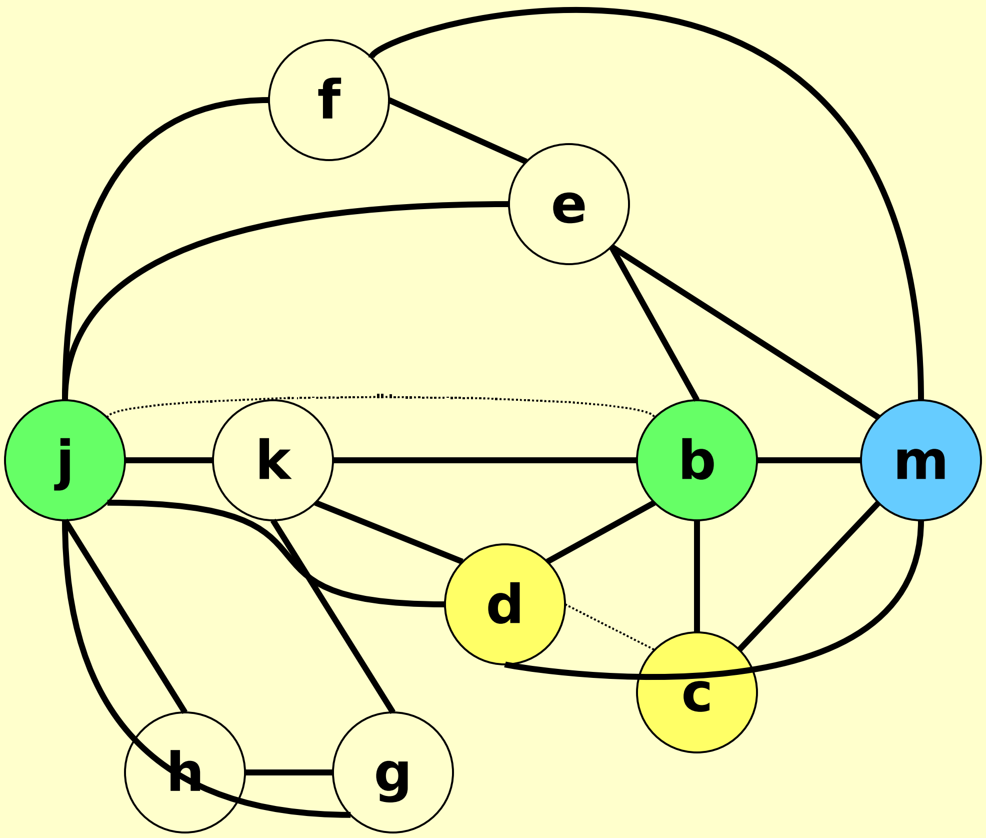
R2

R3

R4

(Appel, pp. 237)

Example: Select (K=4)



stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1

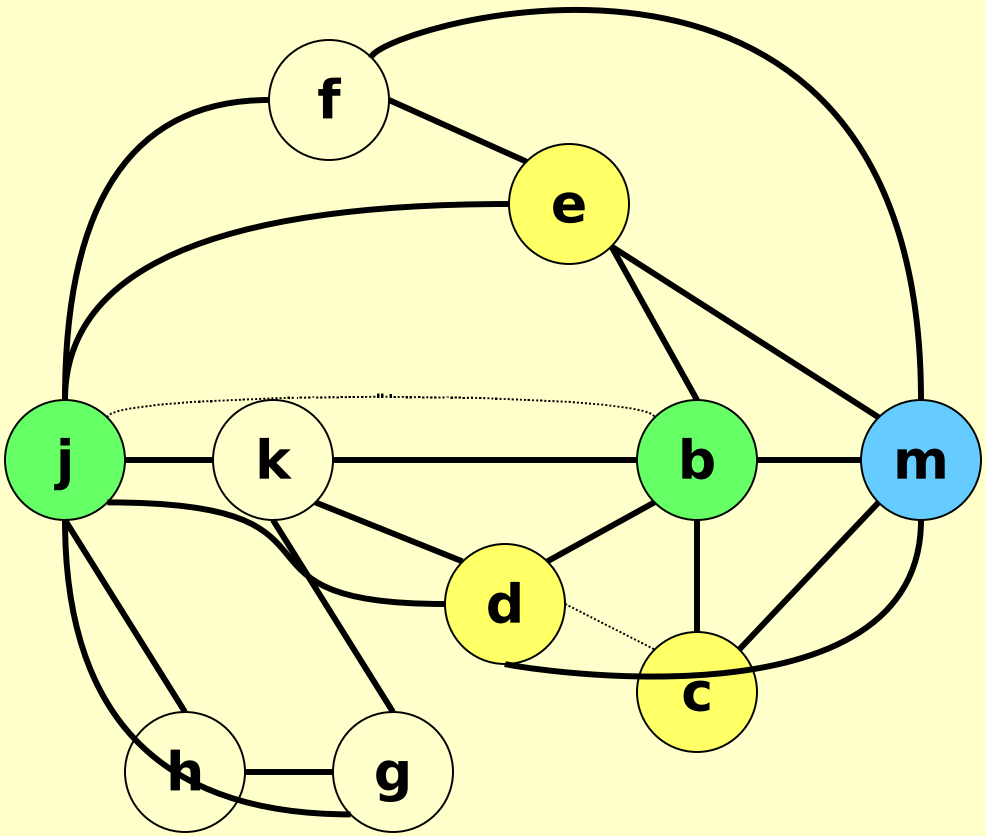
R2

R3

R4

(Appel, pp. 237)

Example: Select (K=4)



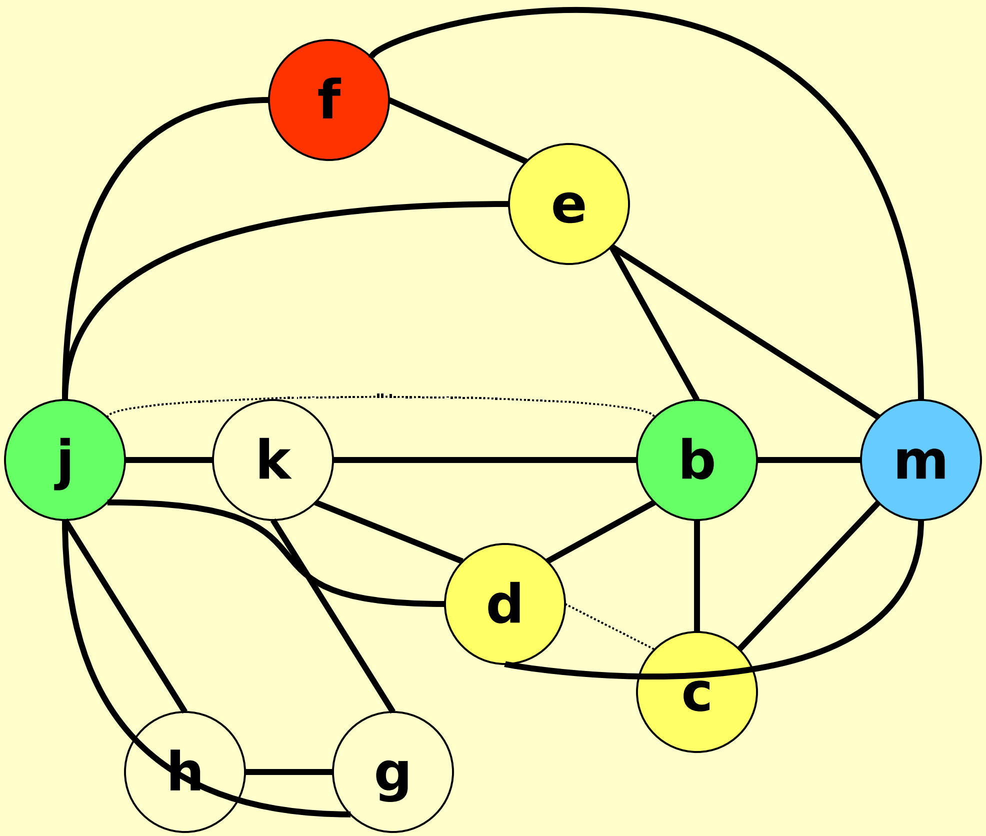
stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1
R2
R3
R4

(Appel, pp. 237)

Example:
Select (K=4)



stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1

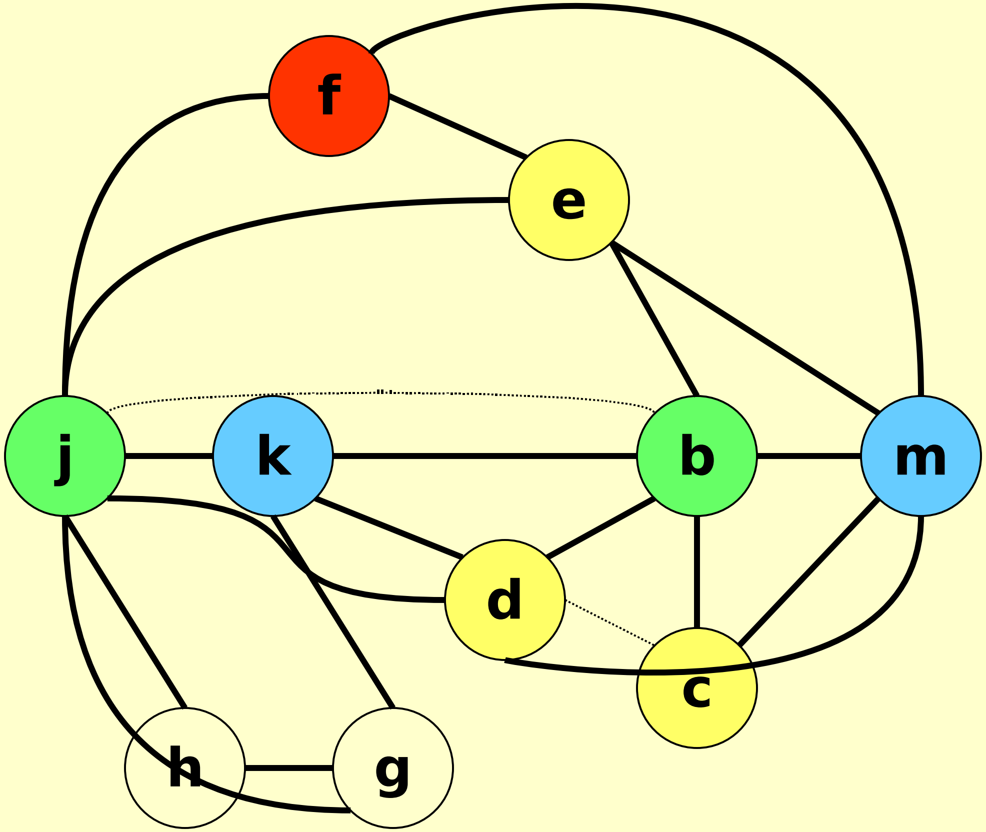
R2

R3

R4

(Appel, pp. 237)

Example: Select (K=4)



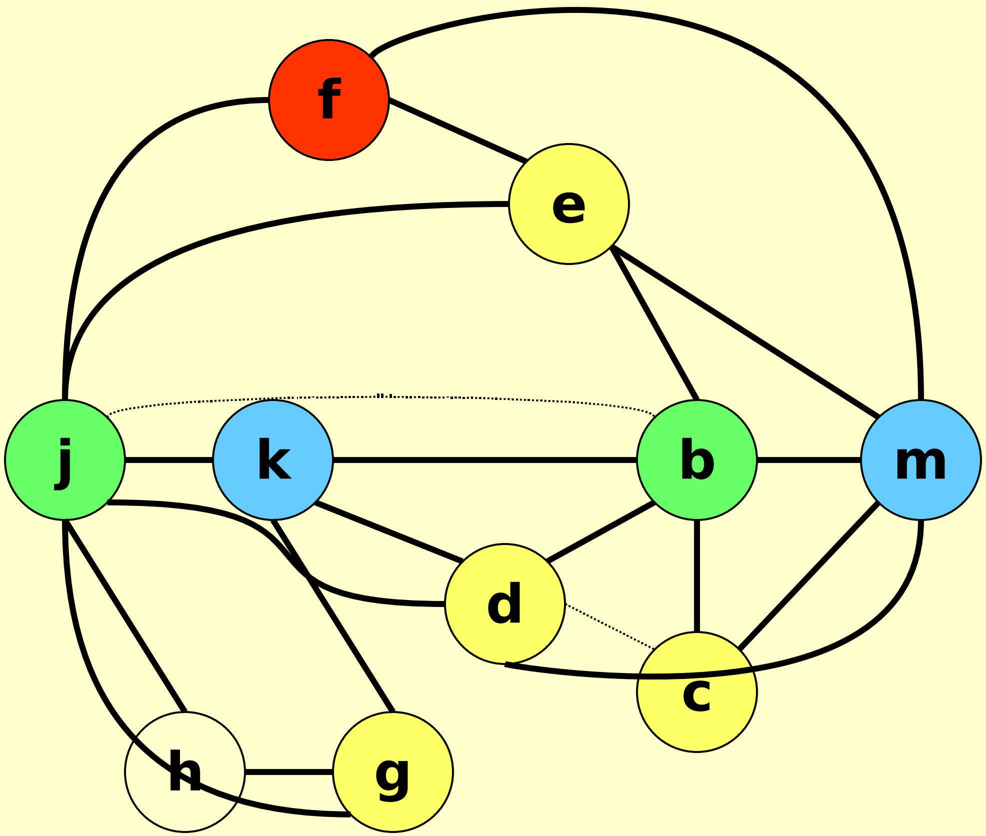
stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1
R2
R3
R4

(Appel, pp. 237)

Example: Select (K=4)



stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1

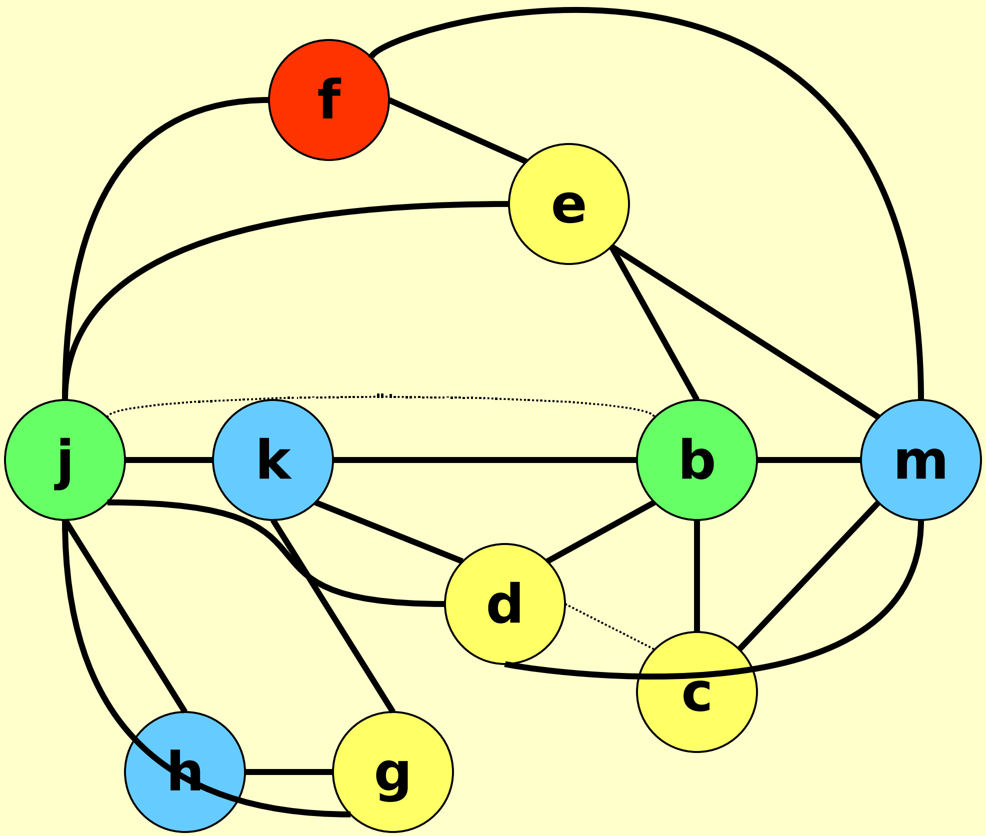
R2

R3

R4

(Appel, pp. 237)

Example:
Select (K=4)



stack

(b-j, no-spill)
(c-d, no-spill)
(m, no-spill)
(e, no-spill)
(f, no-spill)
(k, no-spill)
(g, no-spill)
(h, no-spill)

R1

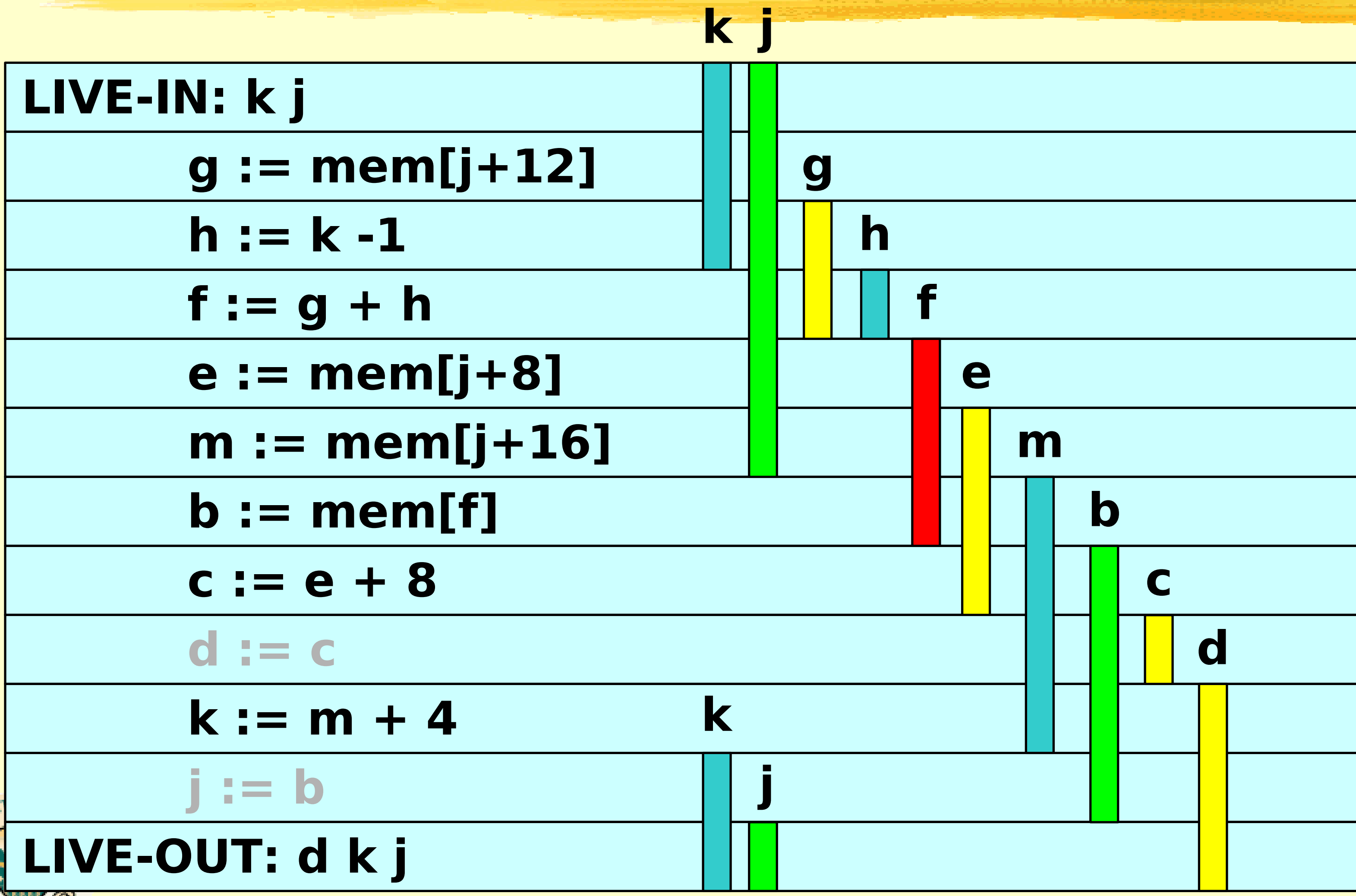
R2

R3

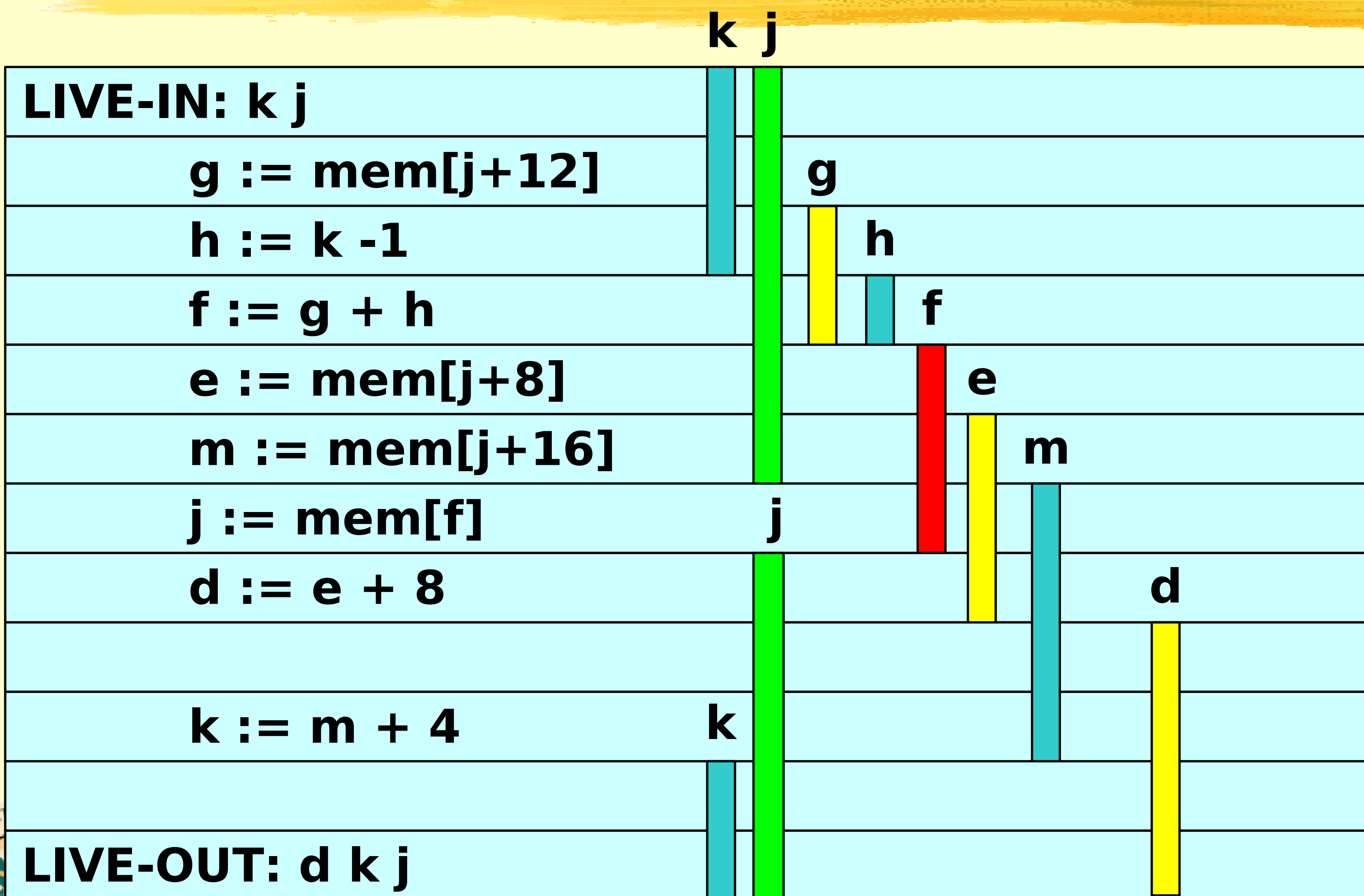
R4

(Appel, pp. 237)

Example:
Allocation with 4 registers

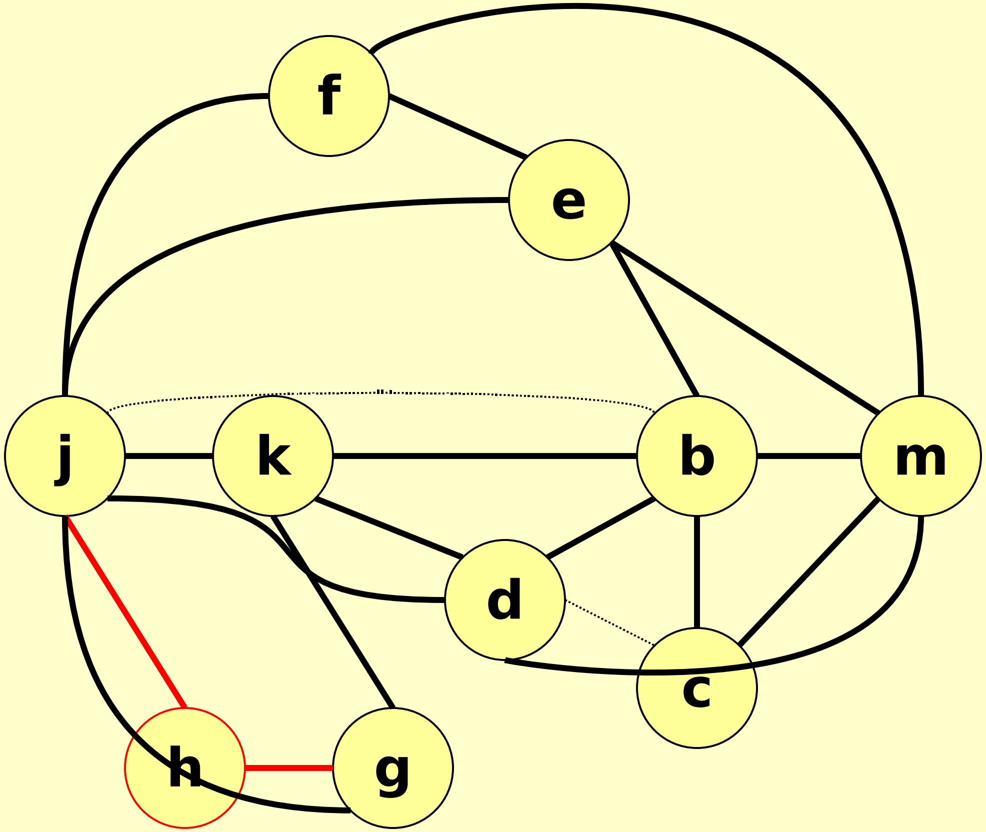


Example: Allocation with 4 registers



Could we do the allocation in the previous example with 3 registers?

Example: Simplify (K=3)

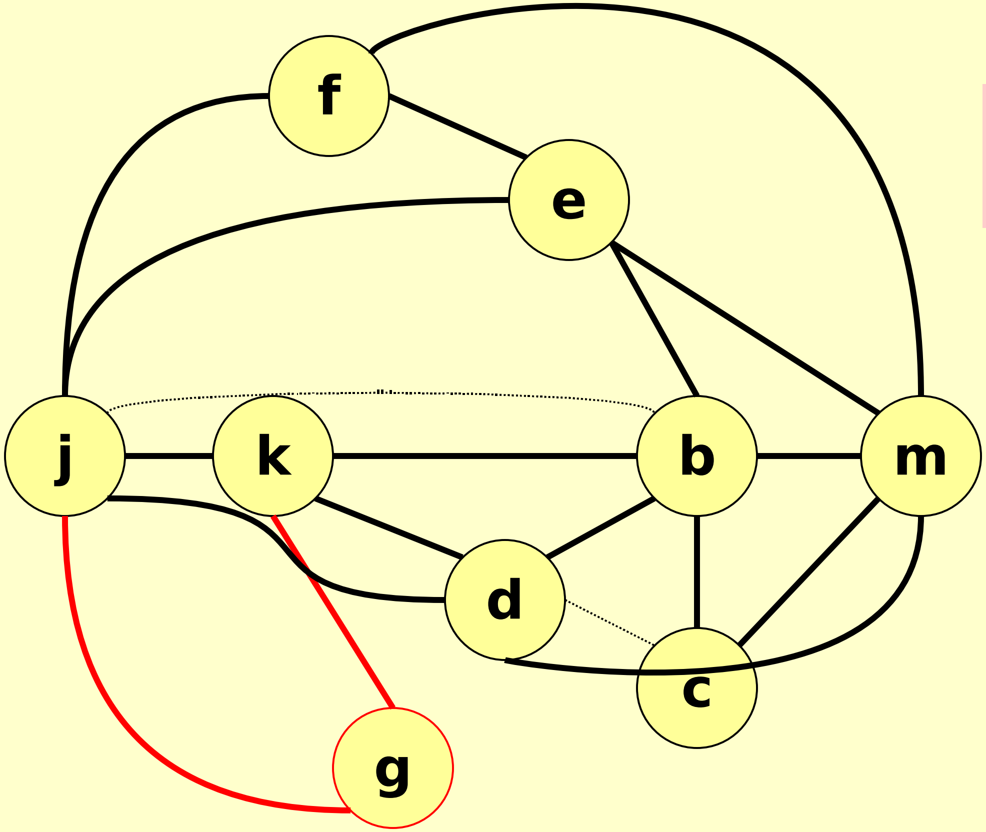


stack

(h,no-spill)

(Appel, pp. 237)

Example: Simplify (K=3)

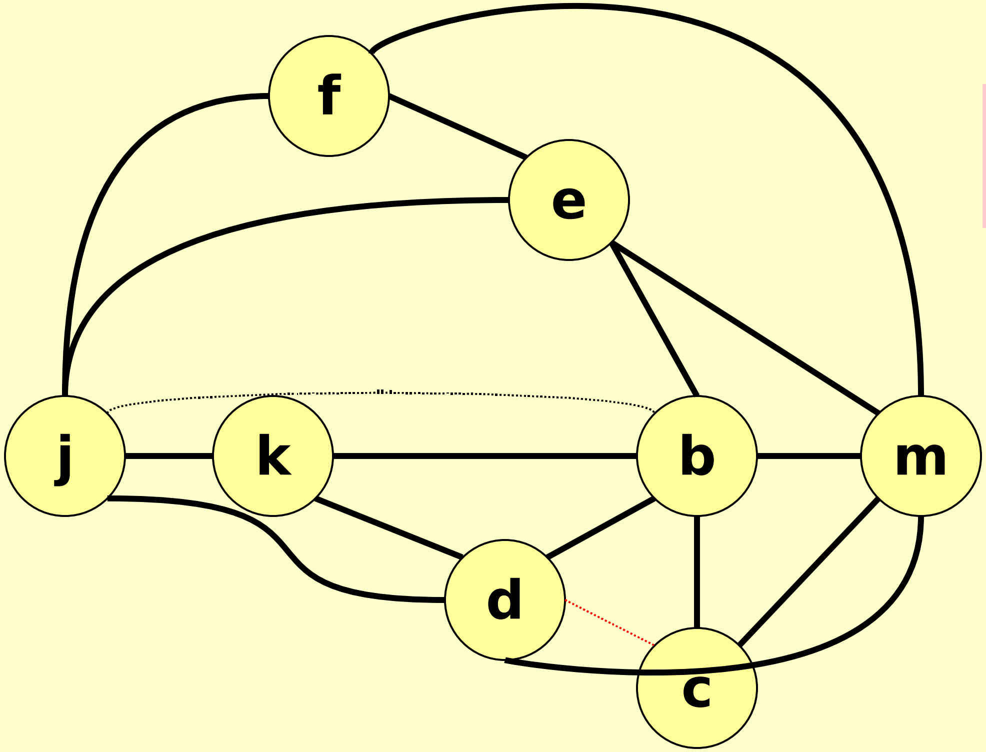


stack

(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Freeze (K=3)



stack

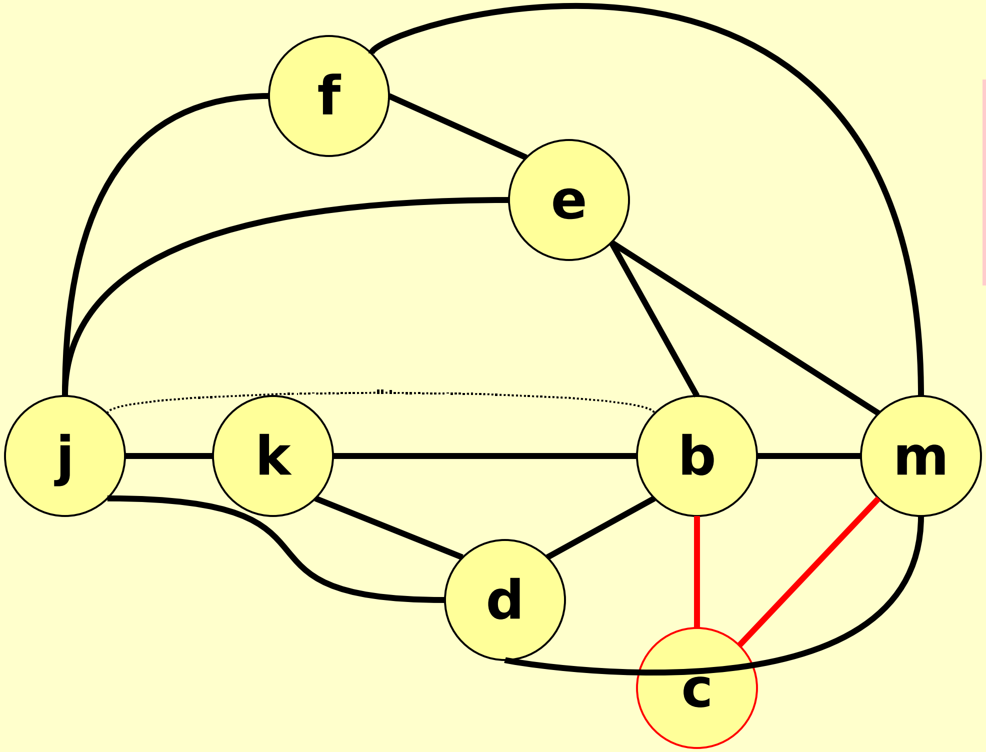
(g, no-spill)
(h, no-spill)

Coalescing may make things worse (not always).

George's rule would coalesce the move d-c, Briggs' rule would freeze.

(Appel, pp. 237)

Example: Simplify (K=3)

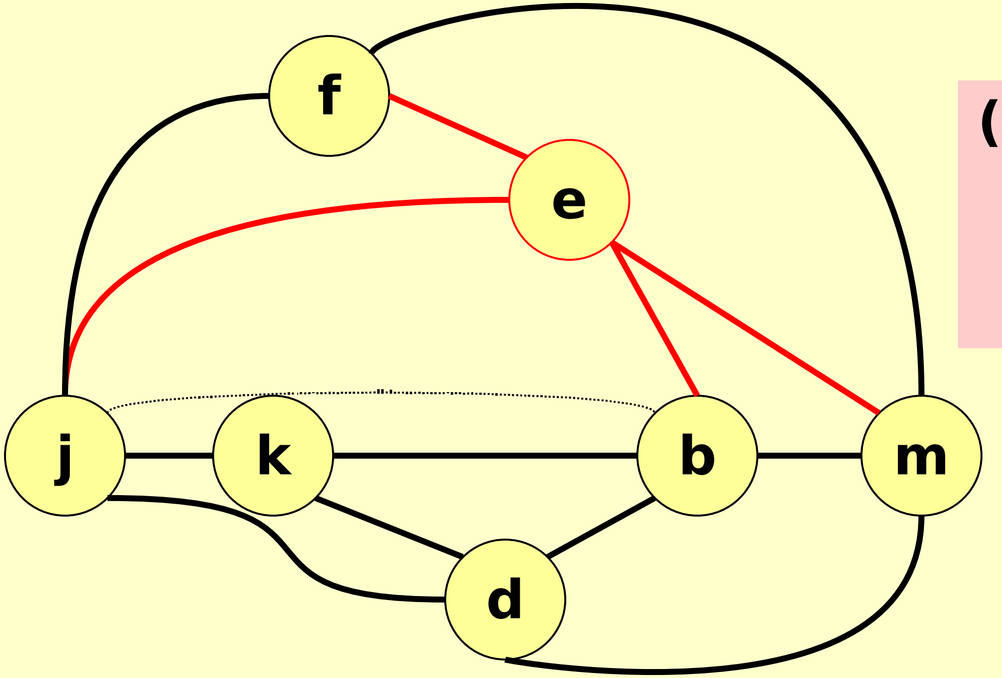


stack

(c, no-spill)
(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Potential Spill (K=3)



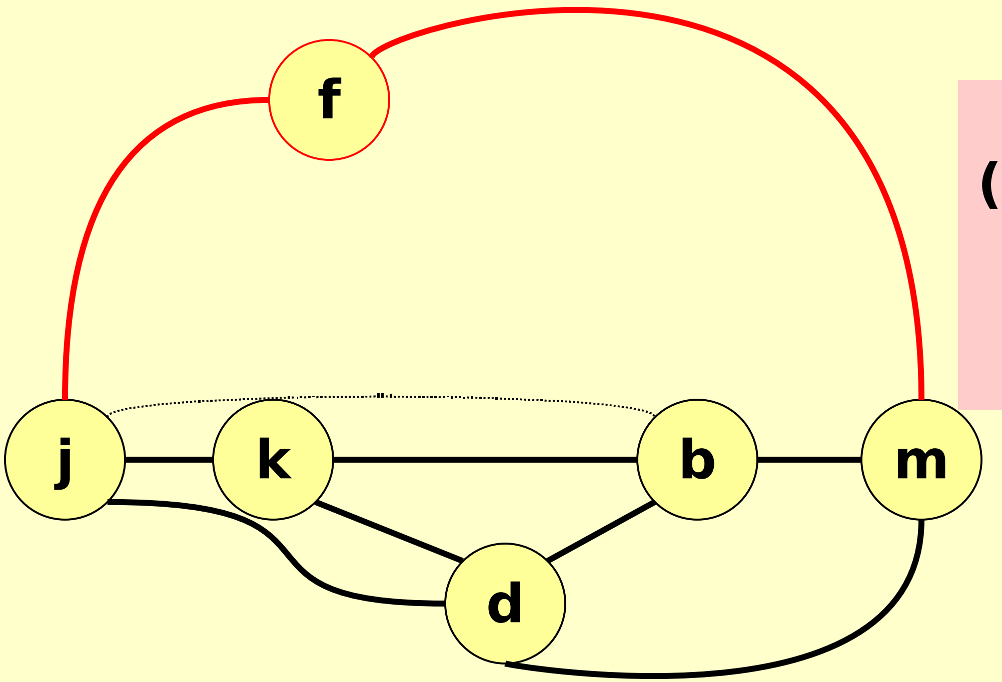
stack

(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

Neither coalescing nor freezing help us.
At this point we should use some profitability analysis to choose a node as *may-spill*.

(Appel, pp. 237)

Example: Simplify (K=3)

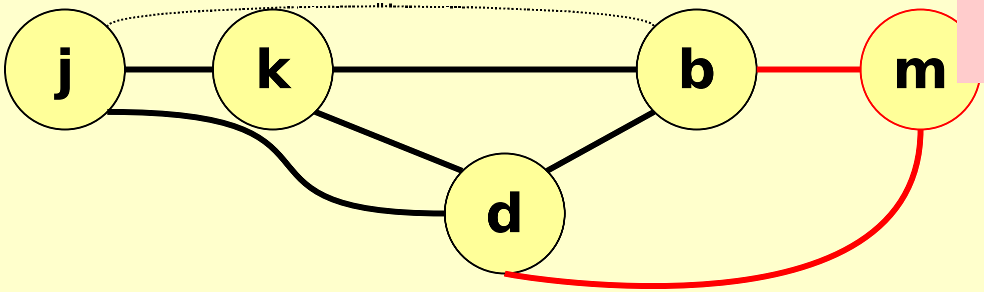


stack

(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Simplify (K=3)

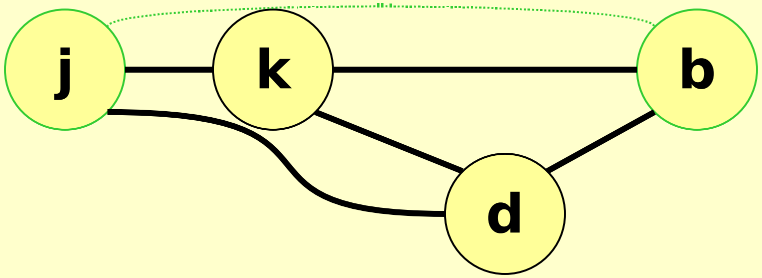


stack

(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Coalesce (K=3)



stack

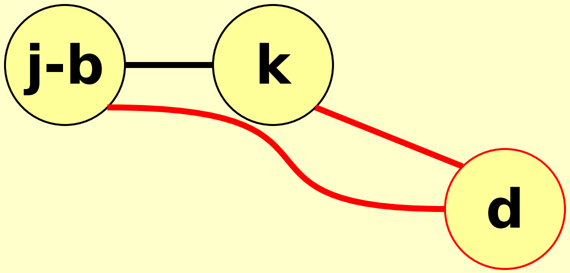
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Coalesce (K=3)

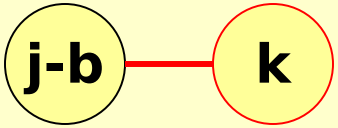
stack

(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)



(Appel, pp. 237)

Example: Coalesce (K=3)

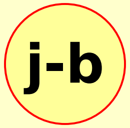


stack

(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Coalesce (K=3)

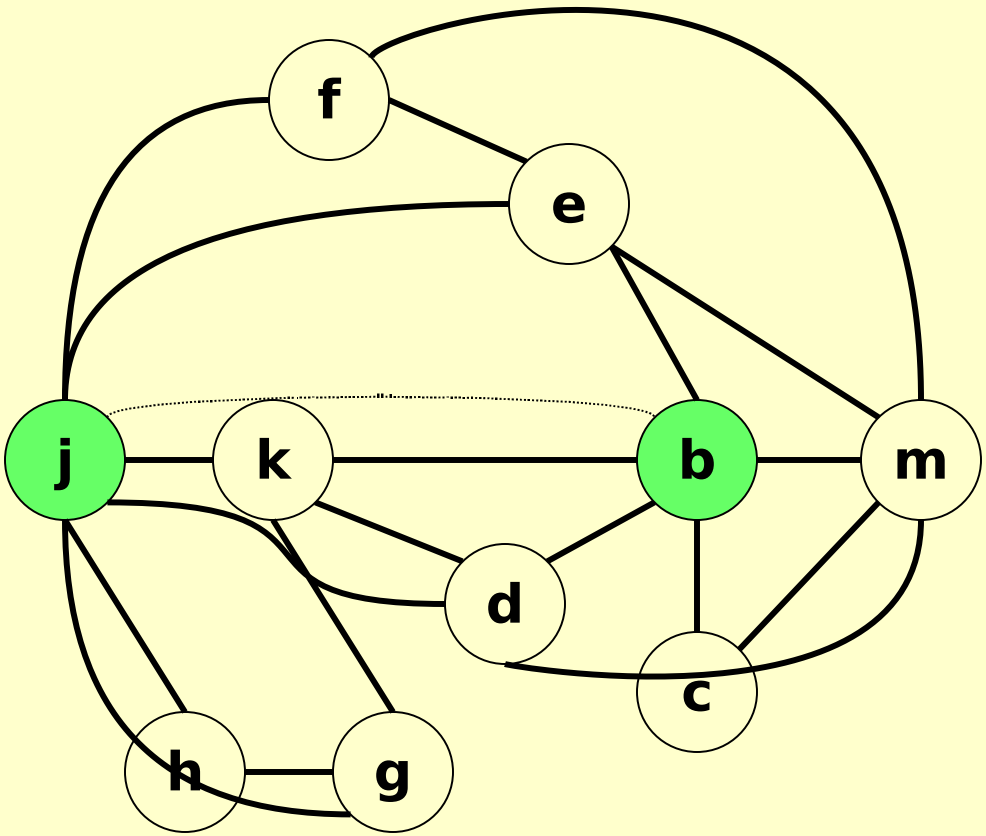


stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

(Appel, pp. 237)

Example: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

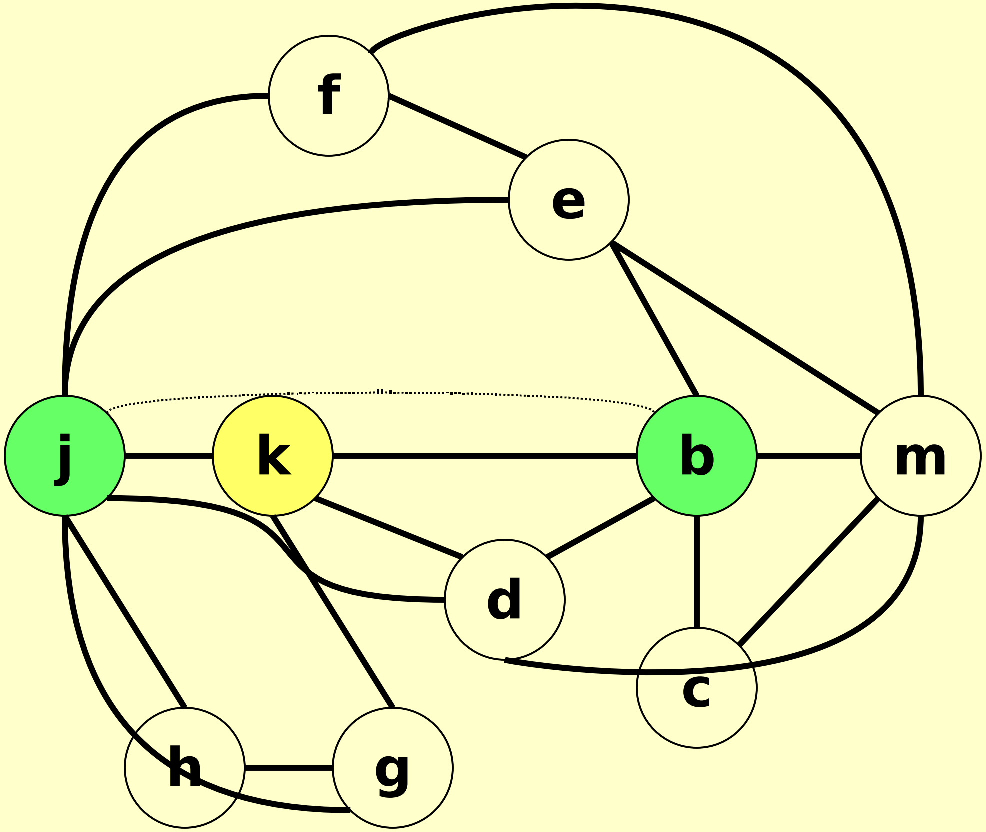
R1

R2

R3

(Appel, pp. 237)

Example: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

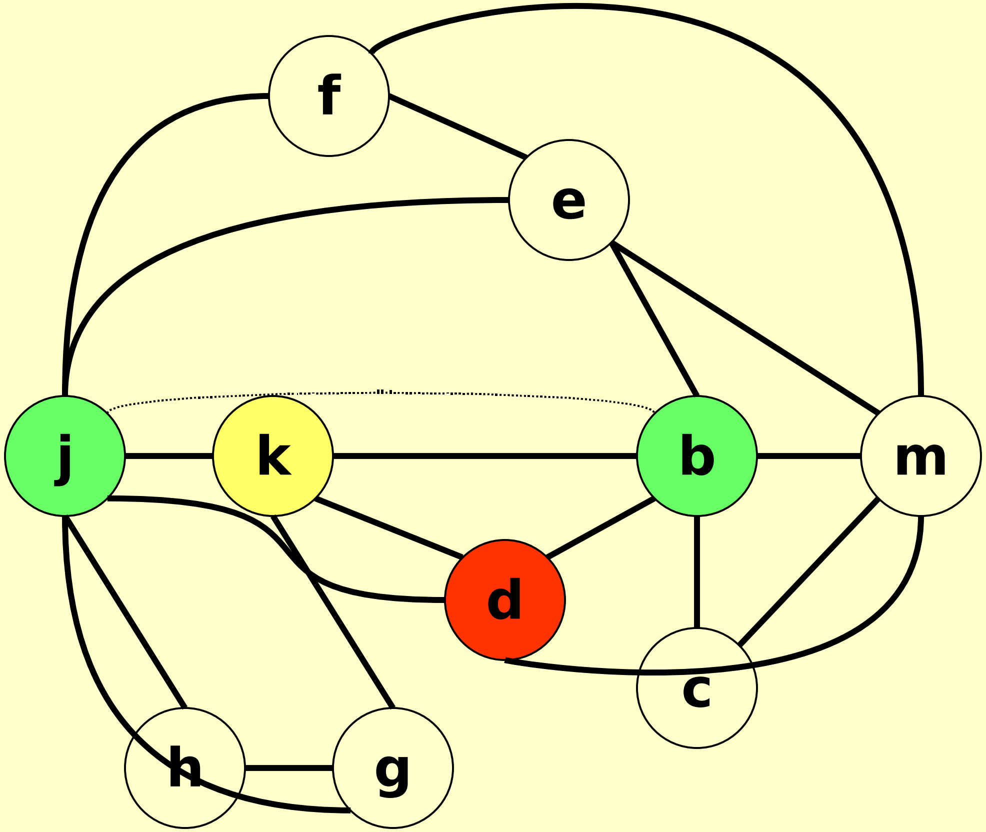
R1

R2

R3

(Appel, pp. 237)

Example: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

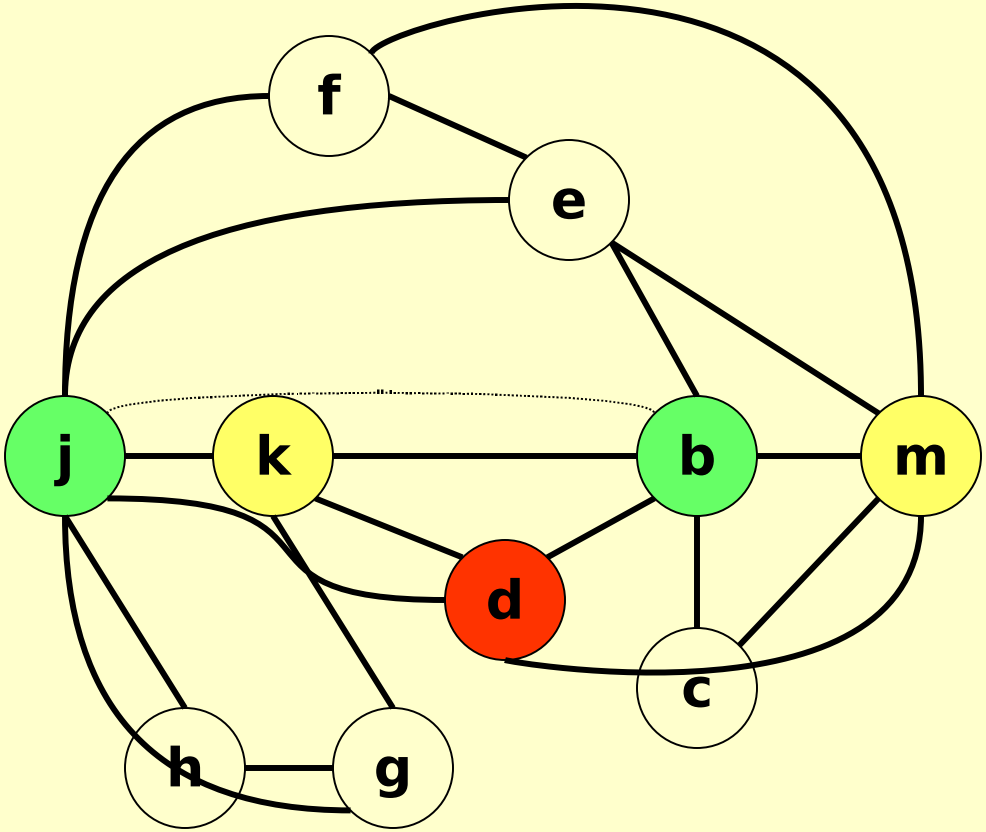
R1

R2

R3

(Appel, pp. 237)

Example: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

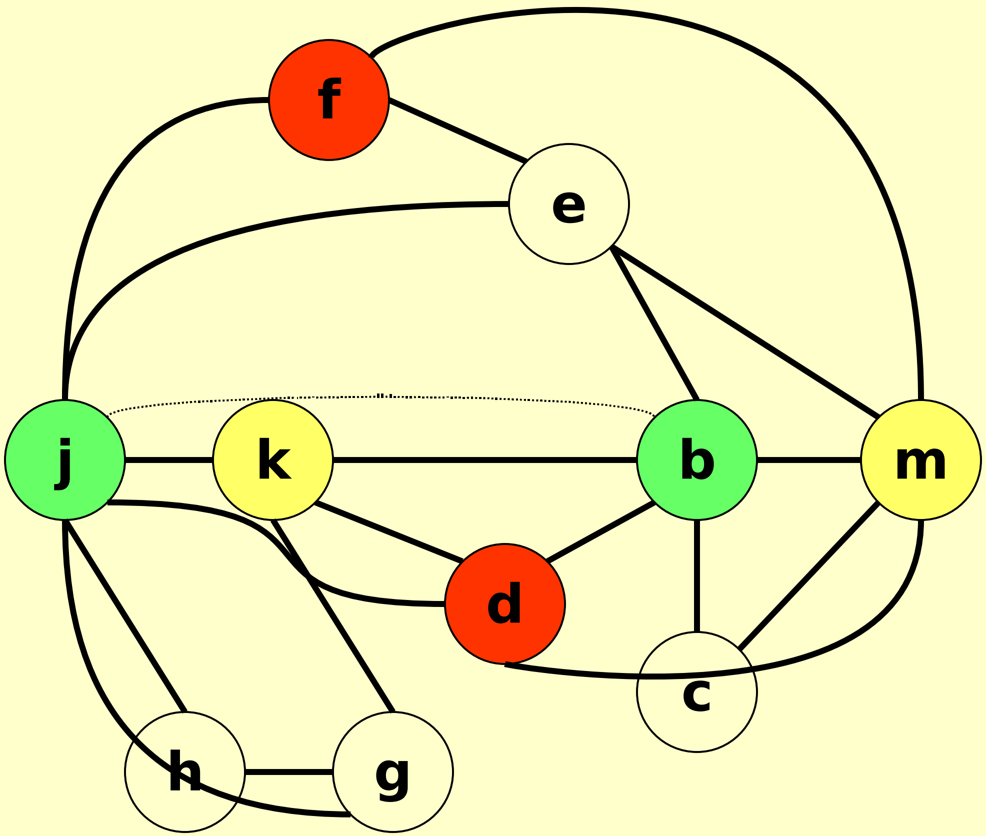
R1

R2

R3

(Appel, pp. 237)

Example: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

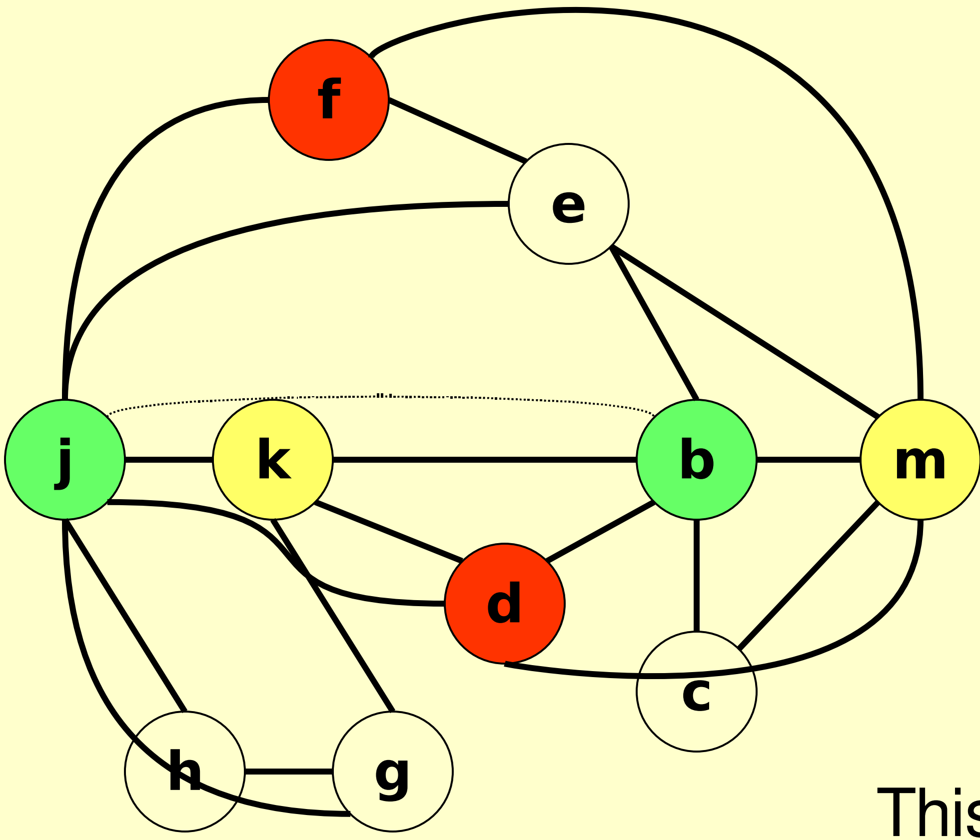
R1

R2

R3

(Appel, pp. 237)

Example: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

R1

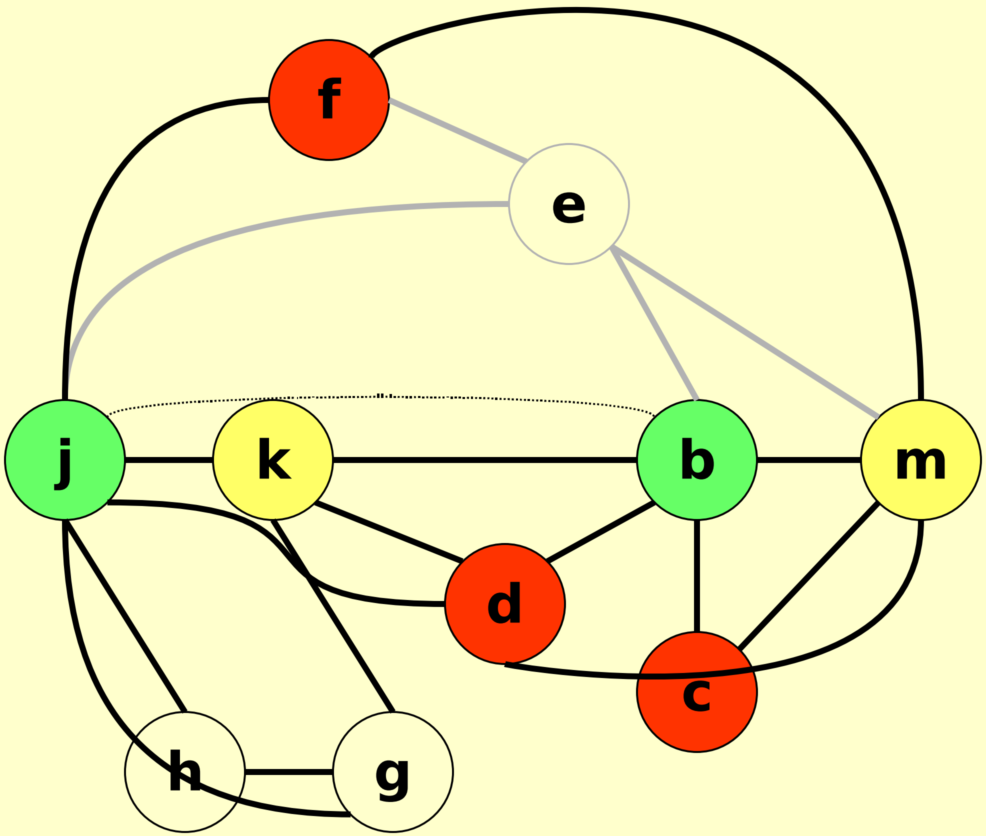
R2

R3

This is when our optimism could have paid off.

(Appel, pp. 237)

Example: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

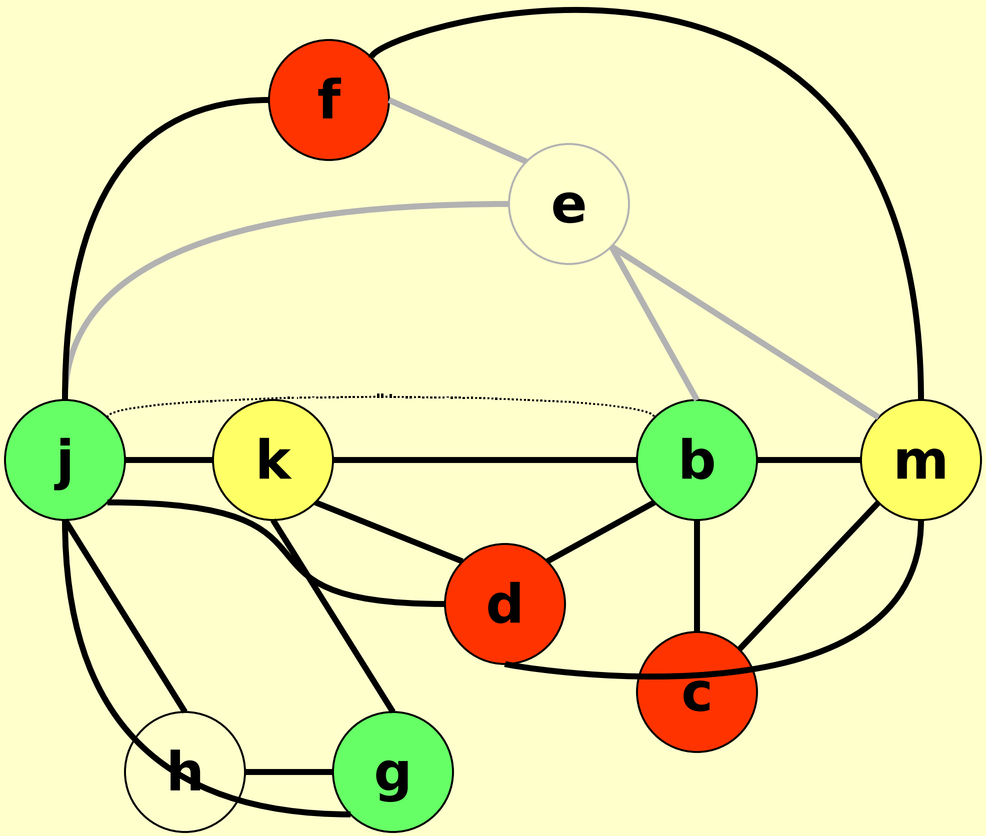
R1

R2

R3

(Appel, pp. 237)

Example: Select (K=3)



stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

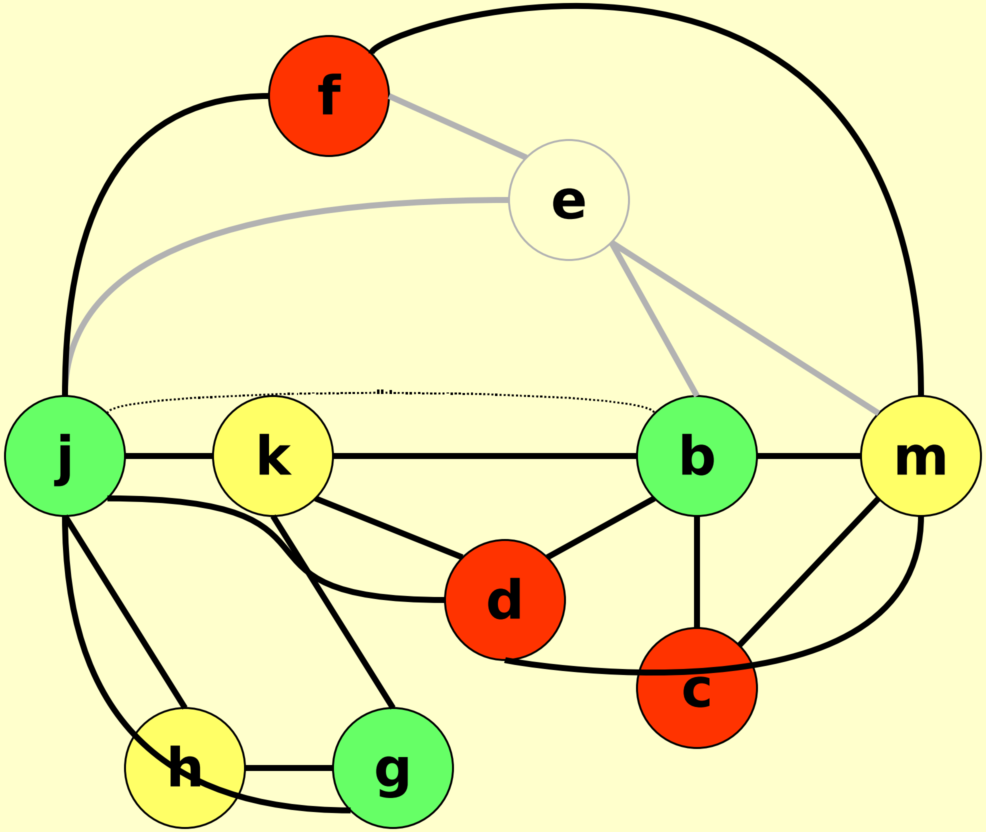
R1

R2

R3

(Appel, pp. 237)

Example: Select (K=3)



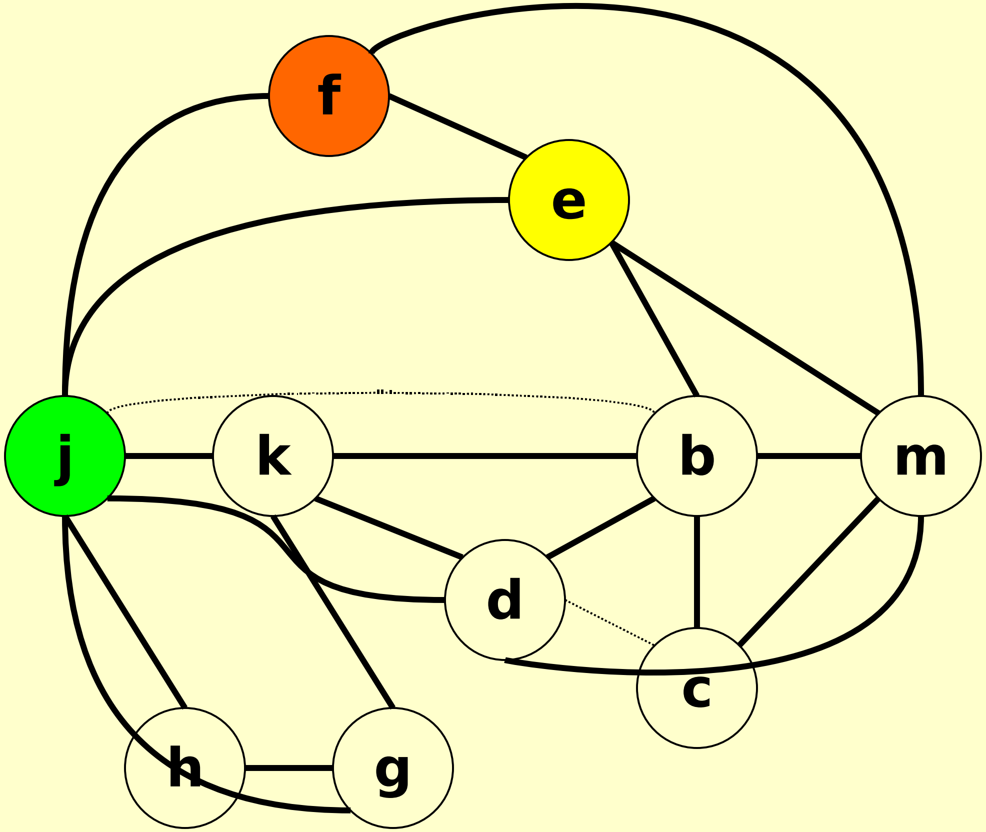
stack

(j-b, no-spill)
(k, no-spill)
(d, no-spill)
(m, no-spill)
(f, no-spill)
(e, may-spill)
(c, no-spill)
(g, no-spill)
(h, no-spill)

R1
R2
R3

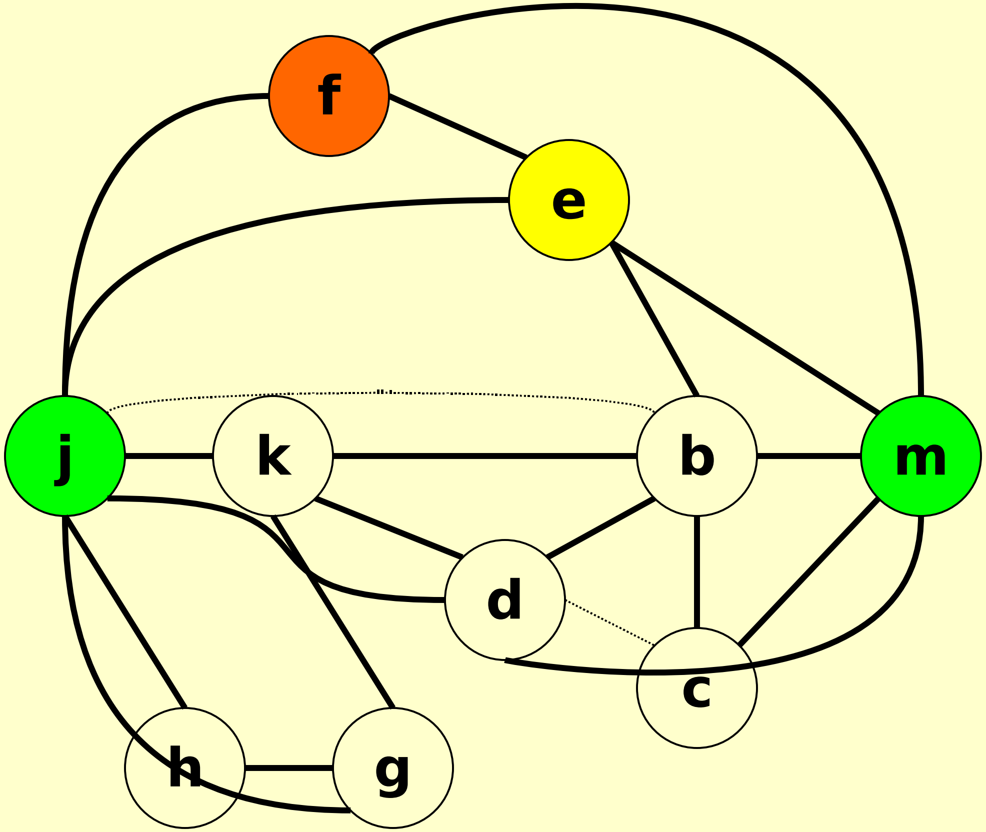
(Appel, pp. 237)

So, is it possible for K=3?



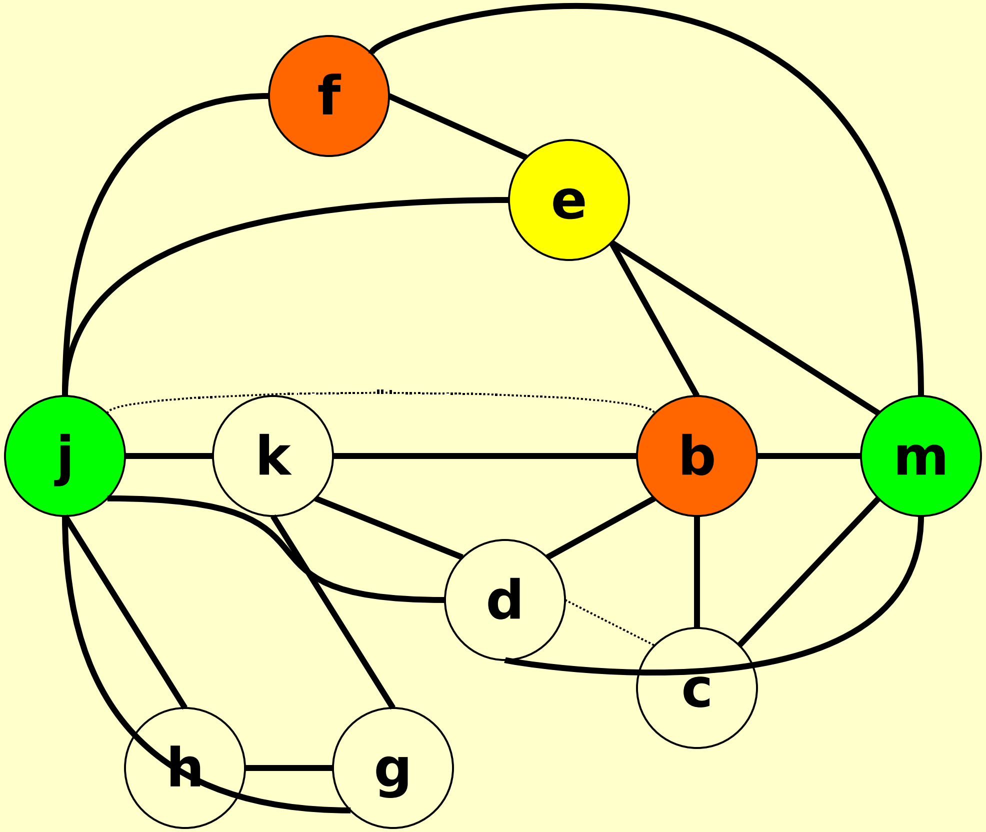
(Appel, pp. 237)

Example: Simplify (K=3)



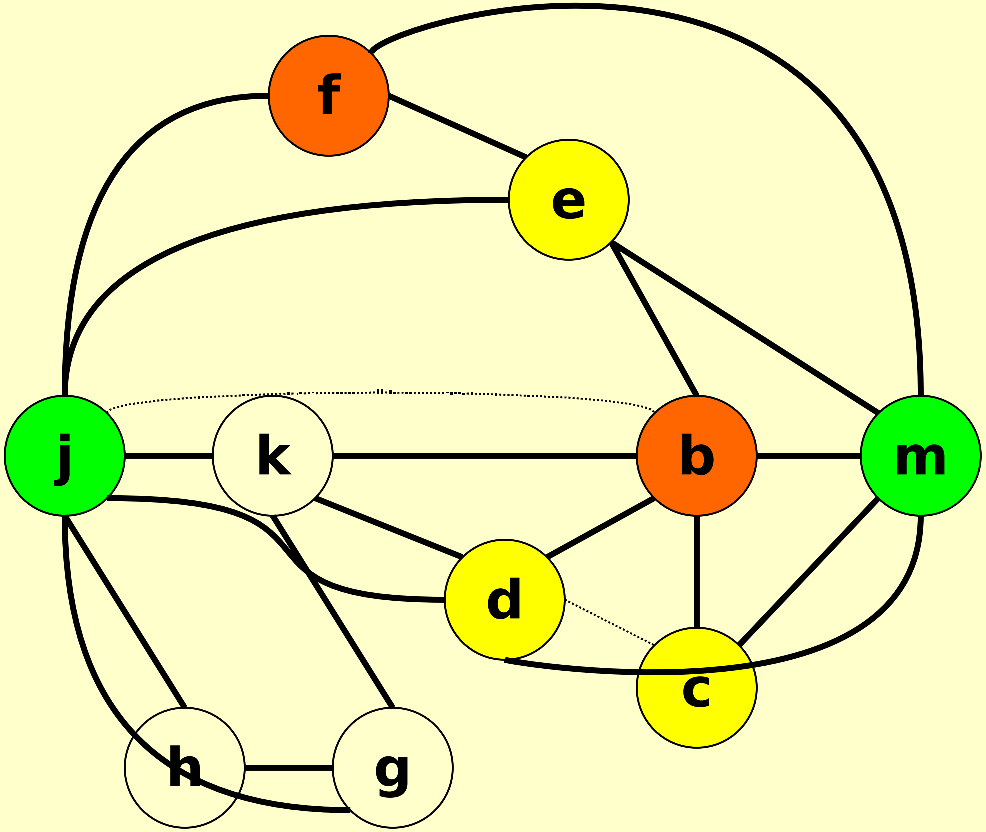
(Appel, pp. 237)

Example:
Simplify (K=3)



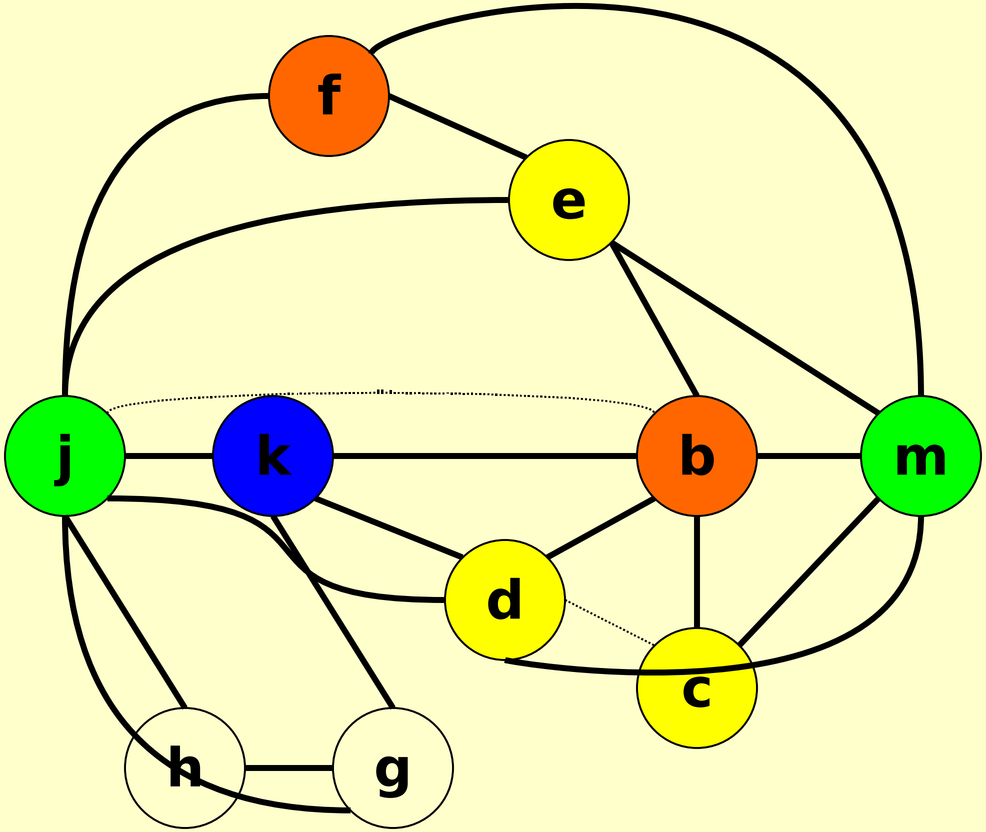
(Appel, pp. 237)

Example: Simplify (K=3)



(Appel, pp. 237)

Example: Simplify (K=3)

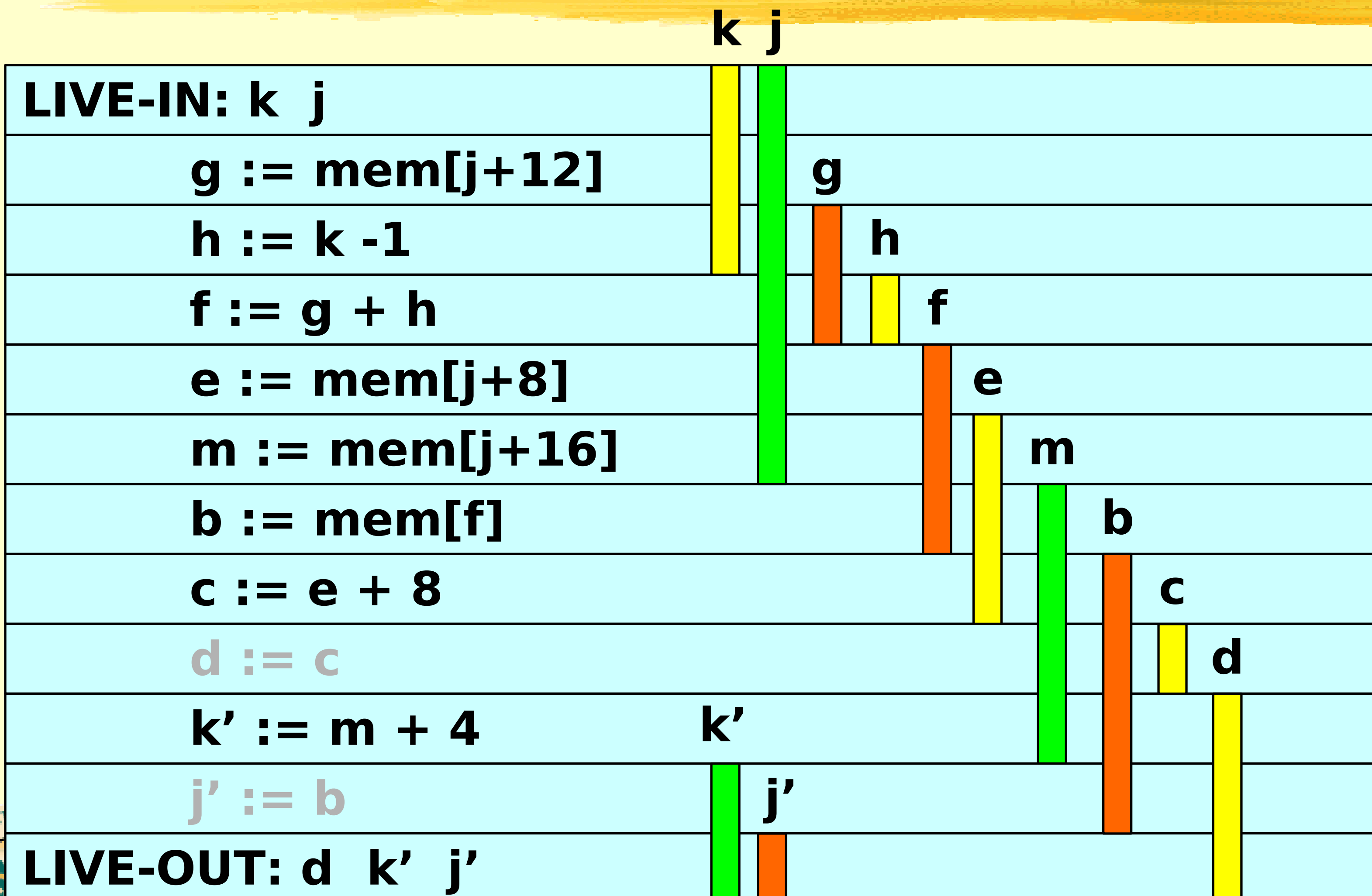


Impossible!

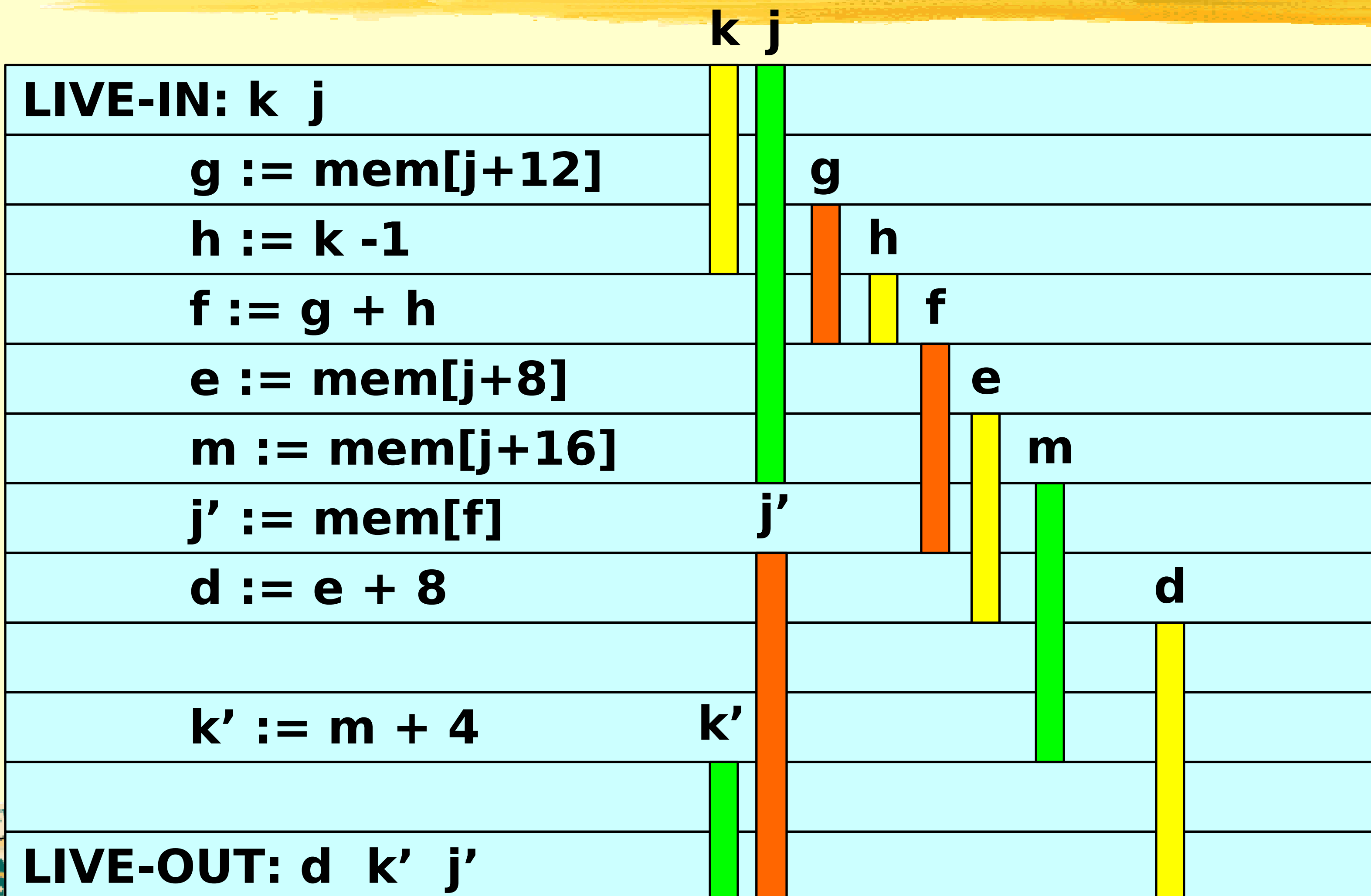
But only 3 variables are live at any time… there may be a way?

(Appel, pp. 237)

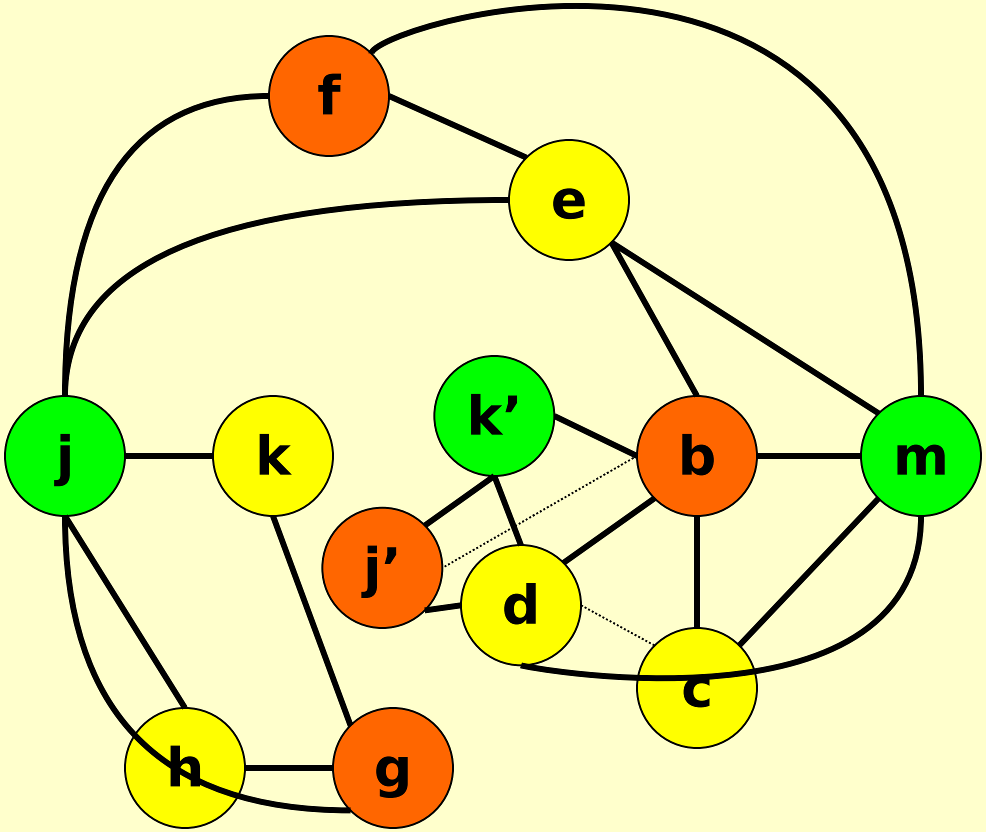
Example as basic block:
3 Registers by renaming k & j



Example as basic block: 3 Registers by renaming k & j



Example as basic block: A 3-coloring of the graph

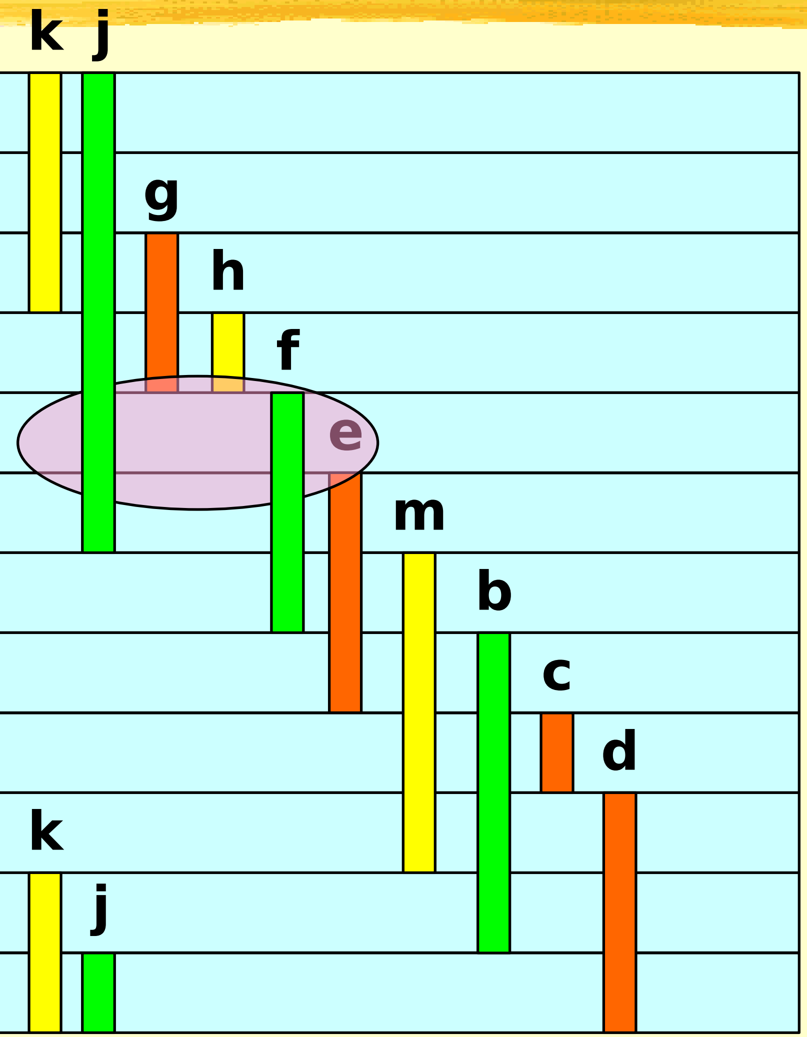


The two assignments of k (resp. j) can be placed in two different registers.

(Appel, pp. 237)

Example as a loop: 3 Registers are enough!

```
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
```



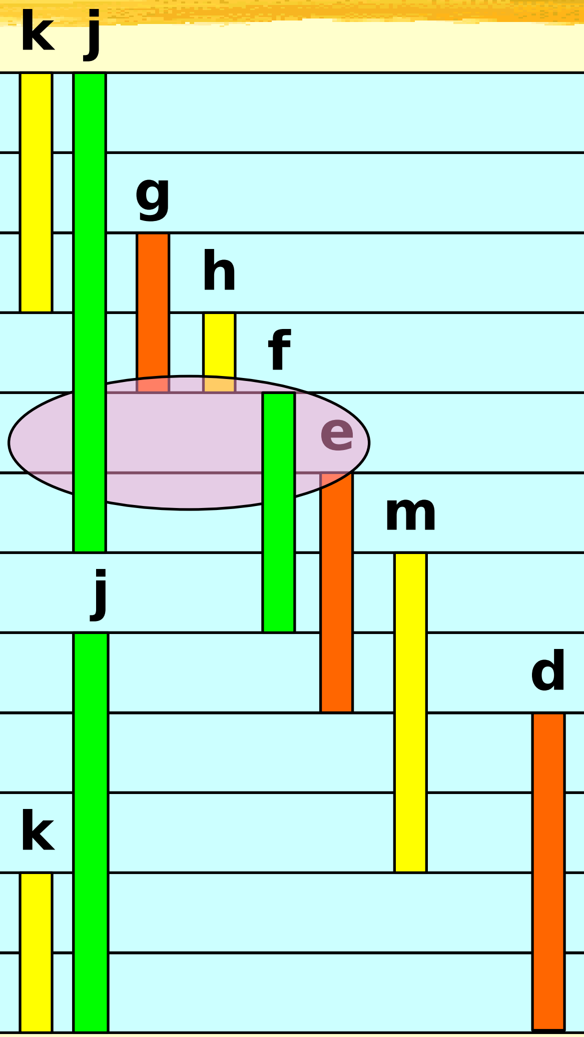
Example as a loop: 3 Registers are enough!

LIVE-IN: k j

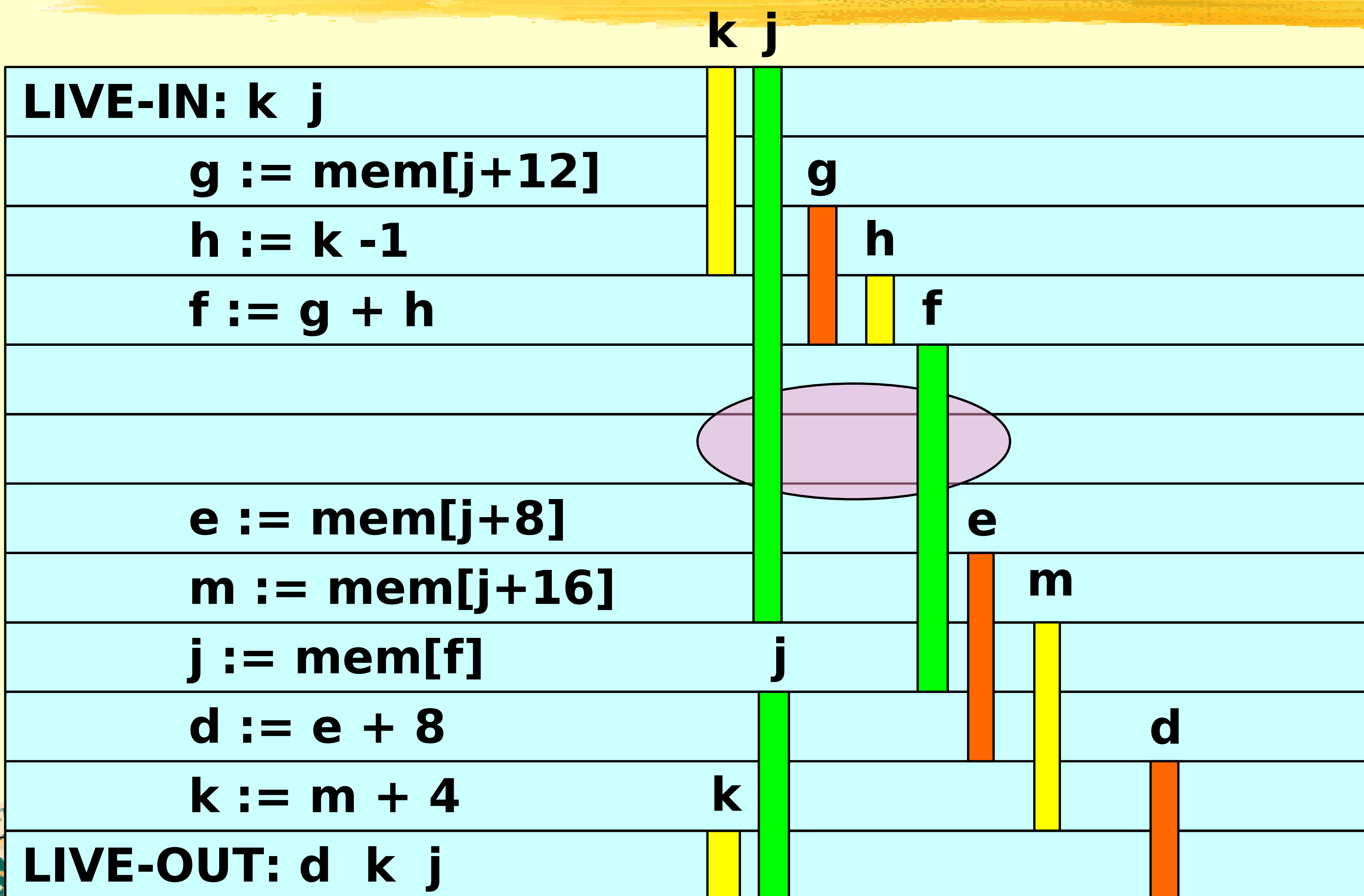
```
g := mem[j+12]
h := k -1
f := g + h
e := mem[j+8]
m := mem[j+16]
b := mem[f]
d := e + 8

k := m + 4
```

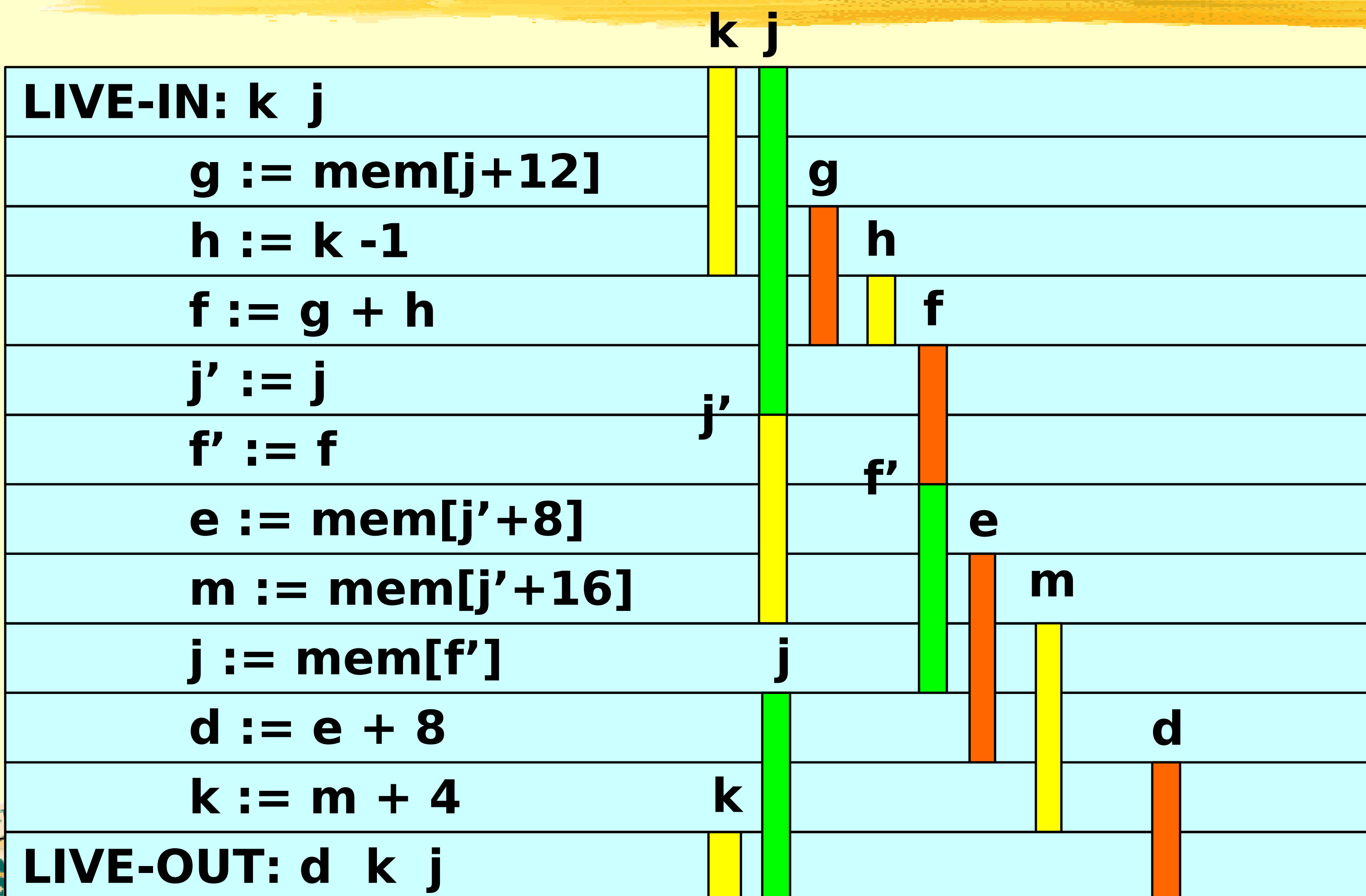
LIVE-OUT: d k j



Example as a loop:
3 Registers are enough!



Example as a loop:
3 Registers are enough!



Outline

Where did the NP-completeness disappear?

Chaitin et al.

Can each variable be mapped to one of the k registers so that simultaneously-live variables are mapped to different registers?
NP-complete to decide.

SSA-based register allocation

Can the (chordal) interference graph be colored with k colors?
Can be checked in linear time.

So a proof that $P = NP$?

Where did the NP-completeness disappear?

Chaitin et al.

Can each variable be mapped to one of the k registers so that simultaneously-live variables are mapped to different registers?

NP-complete to decide.

SSA-based register allocation

Can the (chordal) interference graph be colored with k colors?

Can be checked in linear time.

So a proof that P = NP? Of course not. But a new track to analyze register allocation subtleties, in particular the impact of:

- Strictness.
- Live-range splitting.
- Critical edges.
- Parallel copies (e.g., swap).
- Instruction types (ISA).