CSE4051: Program Verification

CDCL Algorithm

2025 Fall

Woosuk Lee

Review: DPLL Algorithm

Called "chronological backtracking" which does not backtrack to the real reason of the conflict

- DPLL algorithm repeats the following steps:
  - DECIDE: Choose a variable to assign a value.
  - PROPAGATE: Use unit propagation to deduce further assignments.
  - CONFLICT: If a conflict arises, backtrack to the **last** decision point and make a different assignment.
  - BACKTRACK: If no decisions left, backtrack to the previous decision point.

No learning from past mistakes — may make similar mistakes again and again

CDCL Algorithm

- Conflict-Driven Clause Learning (CDCL) is an extension of the DPLL algorithm.
- It adds the ability to learn from conflicts and backtrack more intelligently.
- CDCL replaces the following steps in DPLL:
  - CONFLICT: analyze the conflict to learn a new clause that prevents similar conflicts in the future.
  - BACKTRACK: backtrack to the reason for the conflict
- CDCL is more efficient than DPLL because it can avoid repeating the same mistakes by learning from conflicts and backtracking intelligently.

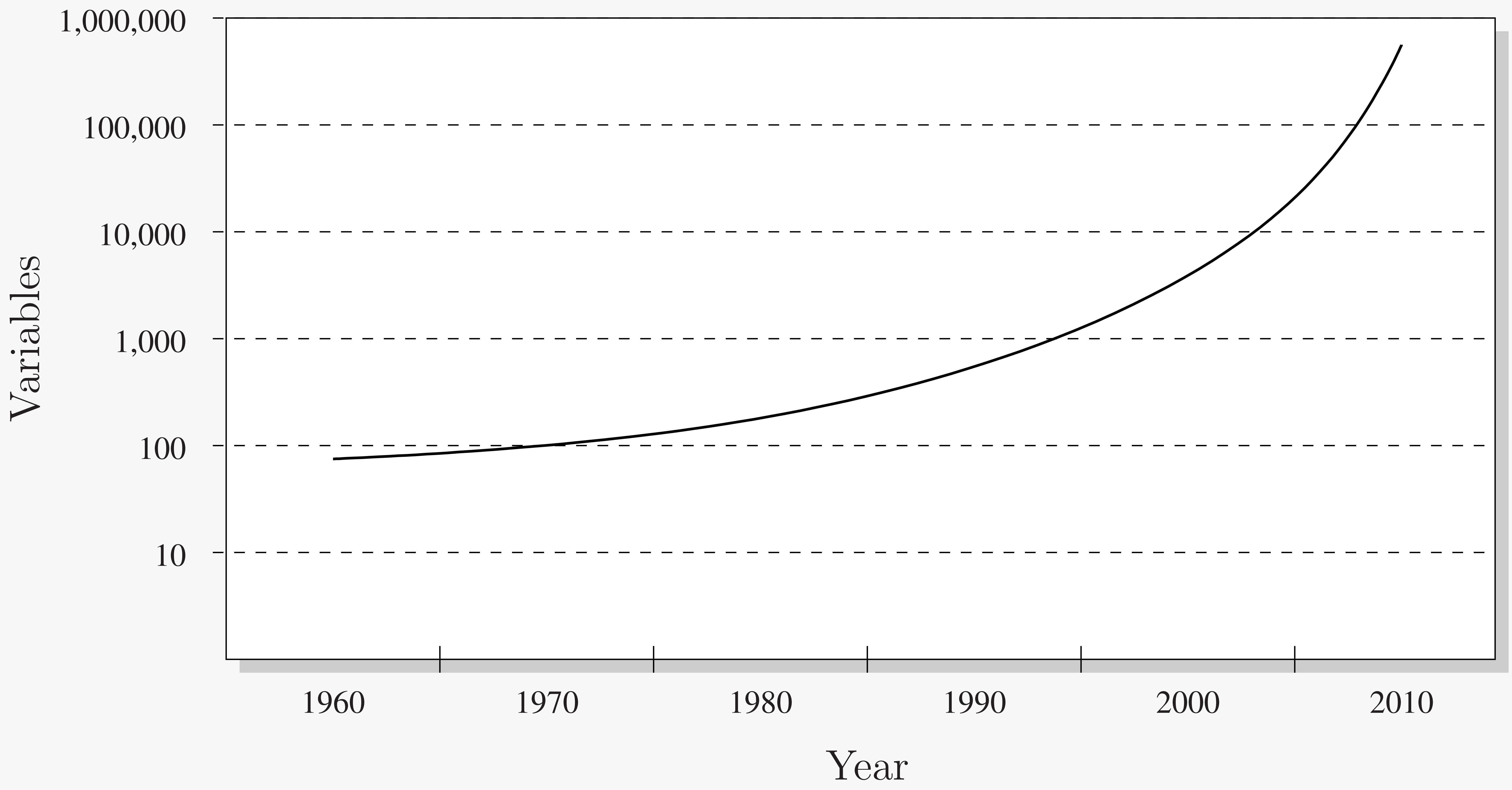


Fig. 2.3. The size of industrial CNF formulas (instances generated for solving various realistic problems such as verification of circuits and planning problems) that are regularly solved by SAT solvers in a few hours, according to year. Most of the progress in efficiency has been made in the last decade

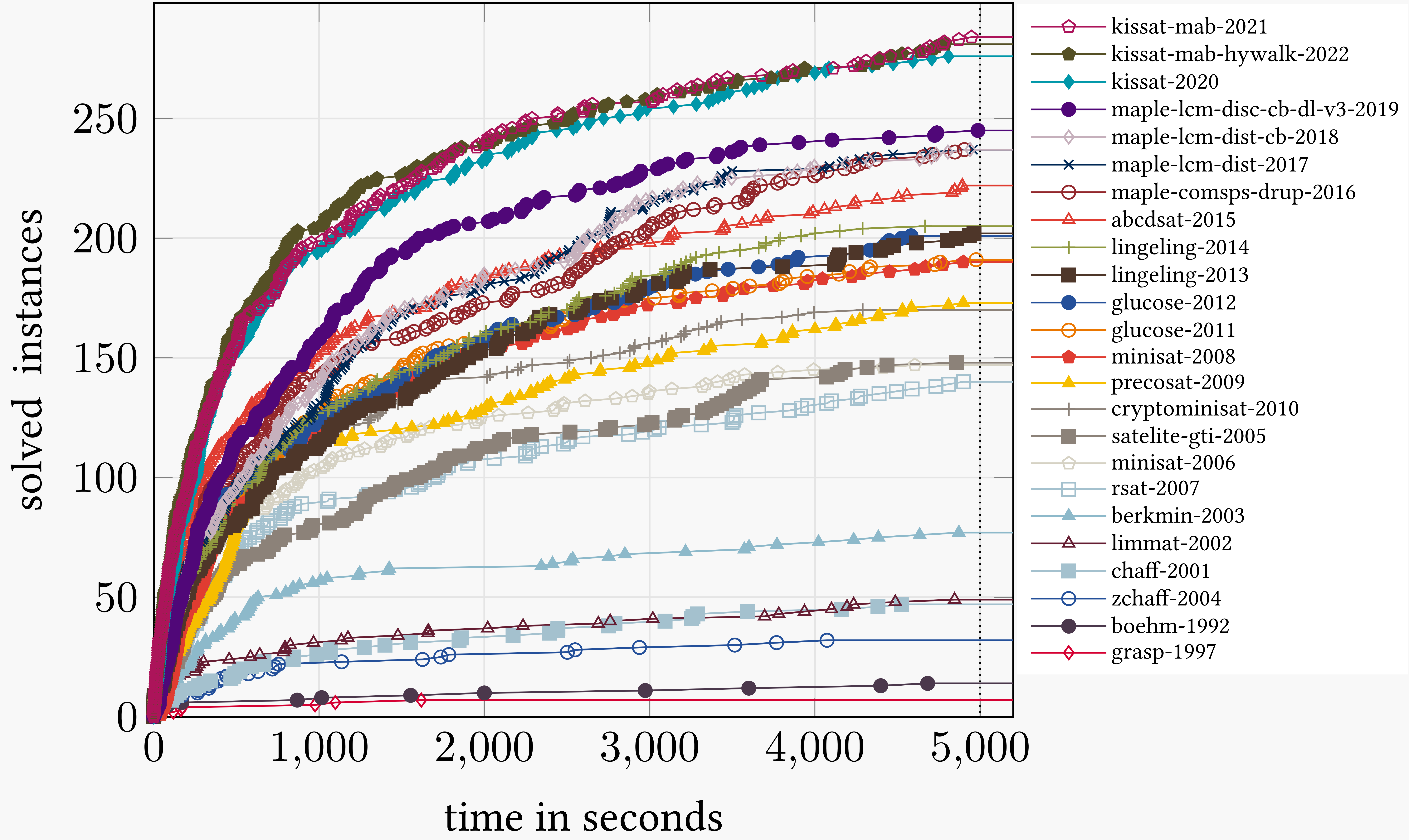


Figure 6: All time winners on the SAT Competition 2022 benchmarks (400 problems)

https://ceur-ws.org/Vol-3545/paper6.pdf

CDCL Example

- Consider $F = c1 \wedge c2 \wedge c3 \wedge c4 \wedge c5 \wedge c6$ where
  - $c1: \neg x1 \vee x2 \vee \neg x4$
  - $c2: \neg x1 \vee \neg x2 \vee x3$
  - $c3: \neg x3 \vee \neg x4$
  - $c4: x4 \vee x5 \vee x6$
  - $c5: \neg x5 \vee x7$
  - $c6: \neg x6 \vee x7 \vee \neg x8$

$c1: \neg x1 \lor x2 \lor \neg x4$

$c2: \neg x1 \lor \neg x2 \lor x3$

$c3: \neg x3 \lor \neg x4$

$c4: x4 \lor x5 \lor x6$

$c5: \neg x5 \lor x7$

$c6: \neg x6 \lor x7 \lor \neg x8$

x1 is assigned true as the first choice

x1@1

True literals in green
False literals in red

c1: ¬x1 ∨ x2 ∨¬x4

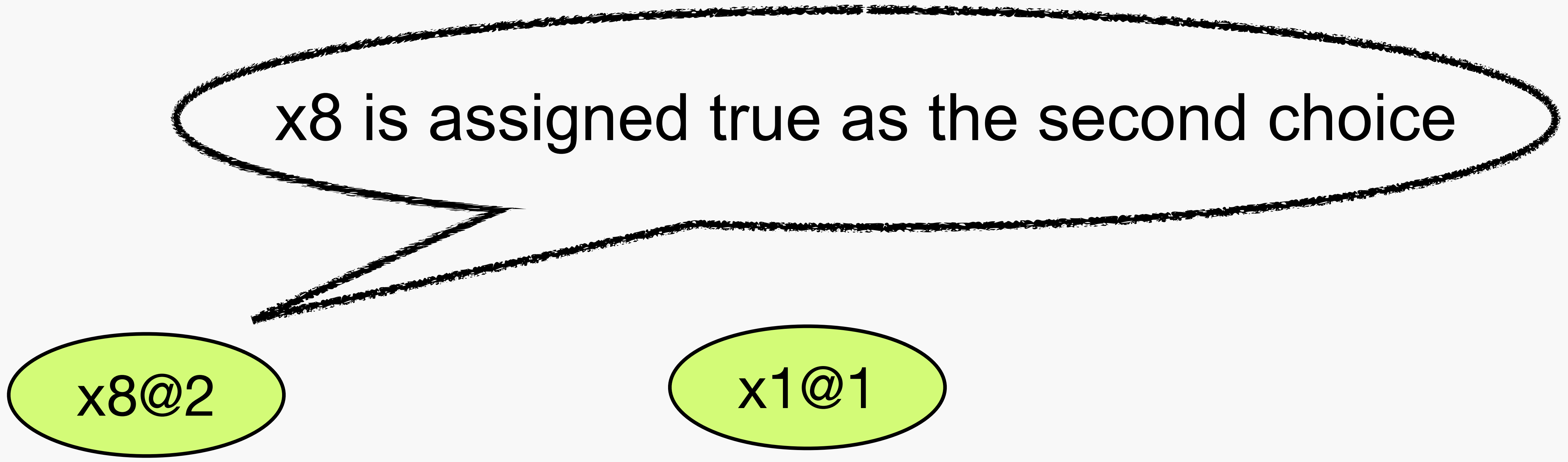
c2 : ¬x1 ∨ ¬x2 ∨ x3

c3: ¬x3 ∨¬x4

c4 : x4 ∨ x5 ∨ x6

c5 : ¬x5 ∨ x7

c6: ¬x6 ∨ x7 ∨¬x8



c1: ¬x1 ∨ x2 ∨¬x4

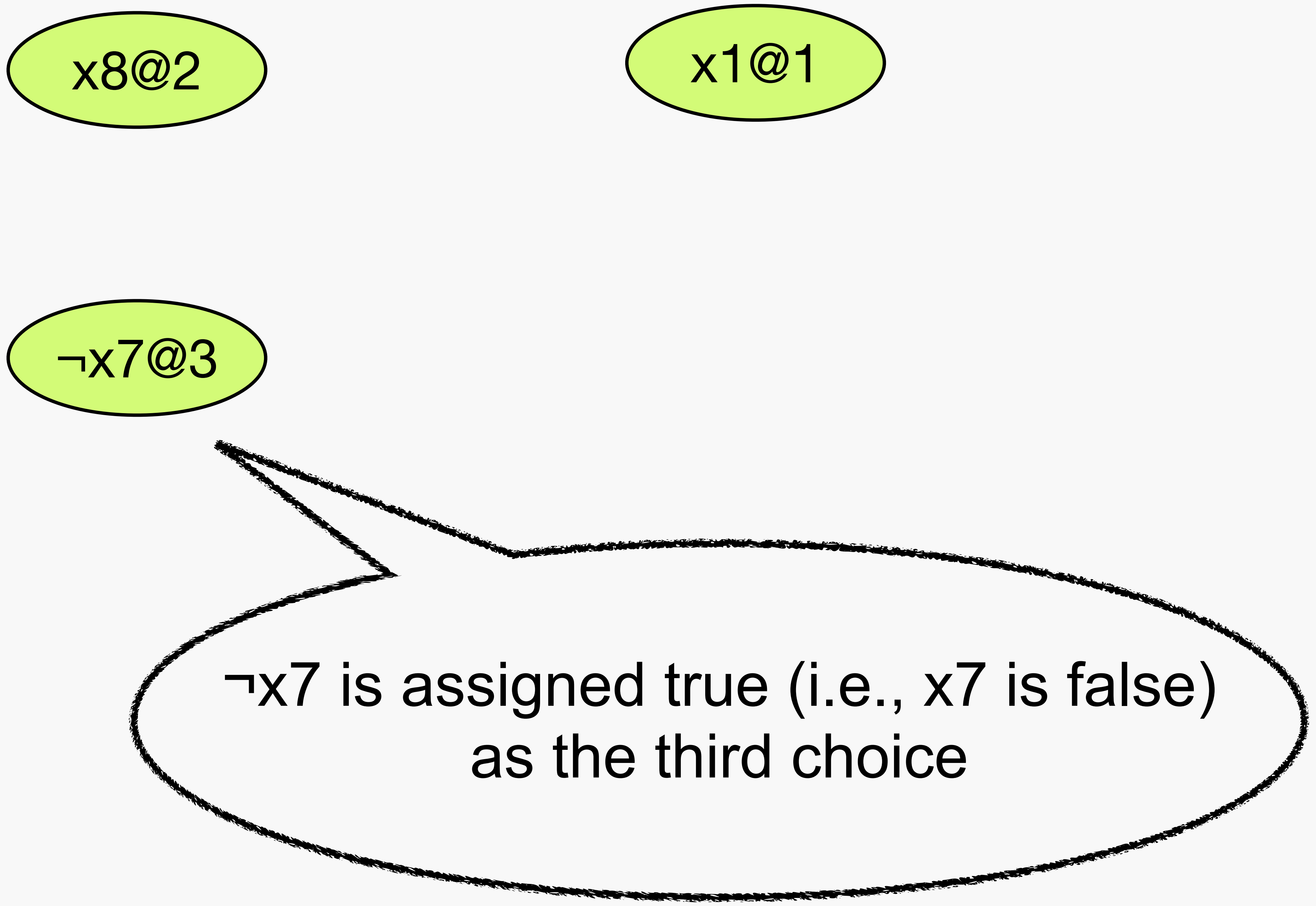
c2 : ¬x1 ∨ ¬x2 ∨ x3

c3: ¬x3 ∨¬x4

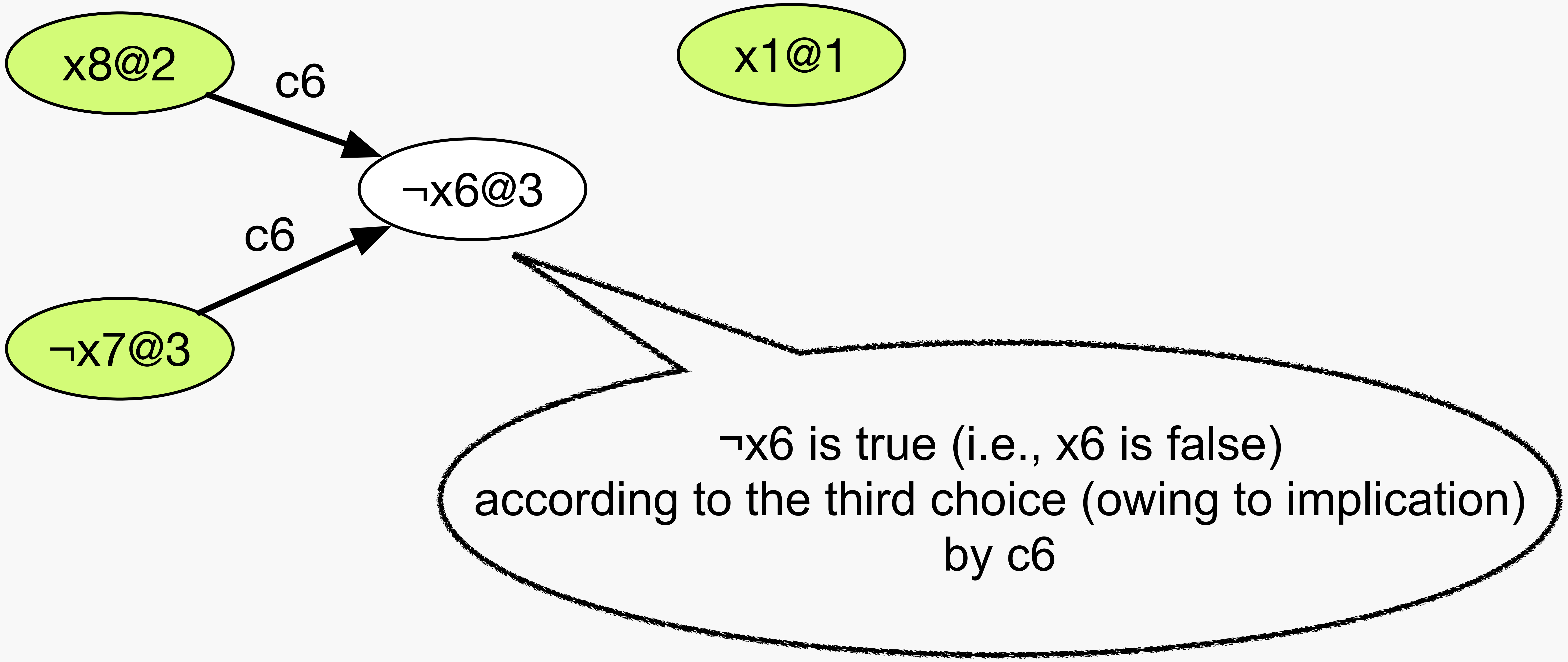
c4 : x4 ∨ x5 ∨ x6

c5 : ¬x5 ∨ x7

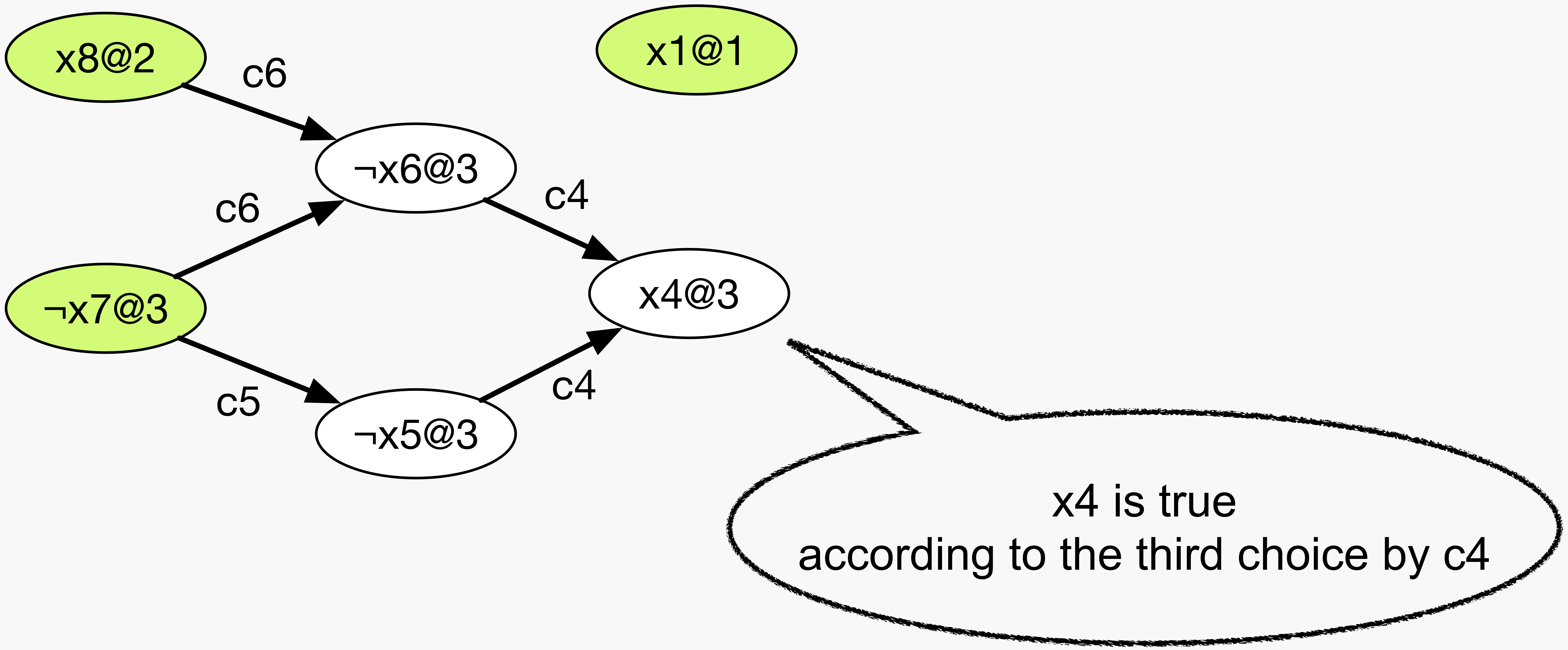
c6: ¬x6 ∨ x7 ∨¬x8



c1: ¬x1 ∨ x2 ∨¬x4
c2 : ¬x1 ∨ ¬x2 ∨ x3
c3: ¬x3 ∨¬x4
c4 : x4 ∨ x5 ∨ x6
c5 : ¬x5 ∨ x7
c6: ¬x6 ∨ x7 ∨¬x8



c1: ¬x1 ∨ x2 ∨¬x4
c2 : ¬x1 ∨ ¬x2 ∨ x3
c3: ¬x3 ∨¬x4
c4 : x4 ∨ x5 ∨ x6
c5 : ¬x5 ∨ x7
c6: ¬x6 ∨ x7 ∨¬x8



c1: ¬x1 ∨ x2 ∨¬x4

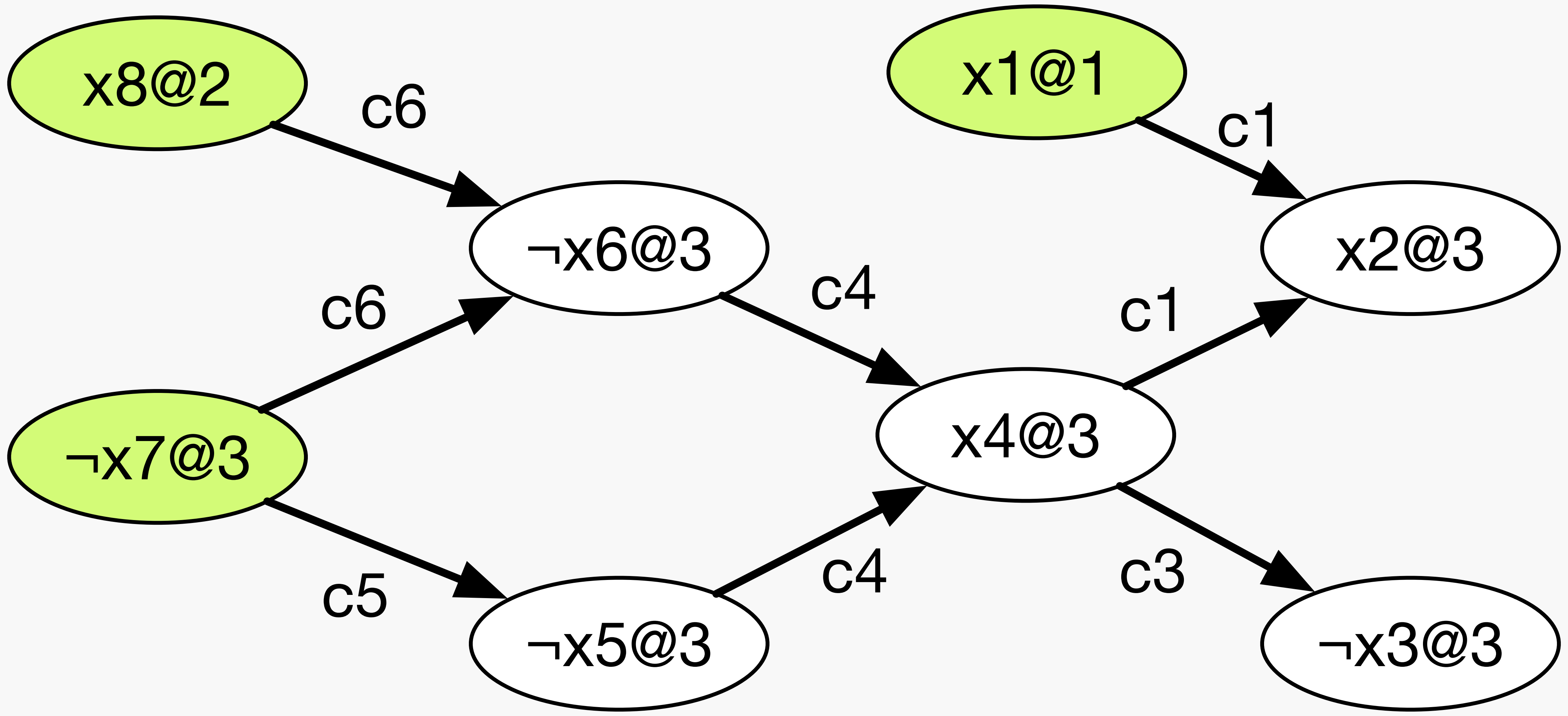
c2 : ¬x1 ∨ ¬x2 ∨ x3

c3: ¬x3 ∨¬x4

c4 : x4 ∨ x5 ∨ x6

c5 : ¬x5 ∨ x7

c6: ¬x6 ∨ x7 ∨¬x8



c1: ¬x1 ∨ x2 ∨¬x4

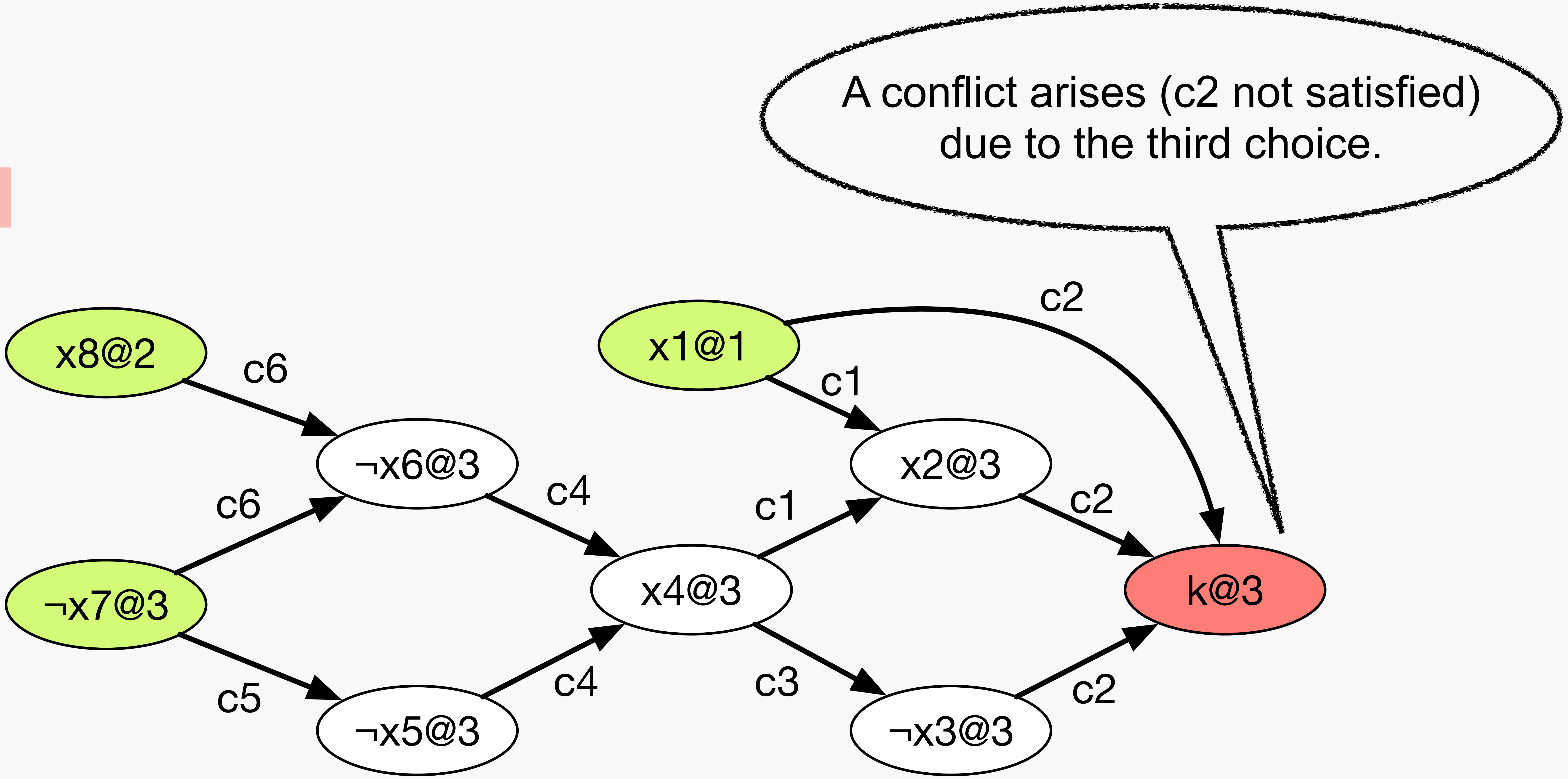
c2 : ¬x1 ∨ ¬x2 ∨ x3

c3: ¬x3 ∨¬x4

c4 : x4 ∨ x5 ∨ x6

c5 : ¬x5 ∨ x7

c6: ¬x6 ∨ x7 ∨¬x8



c1: ¬x1 ∨ x2 ∨ ¬x4

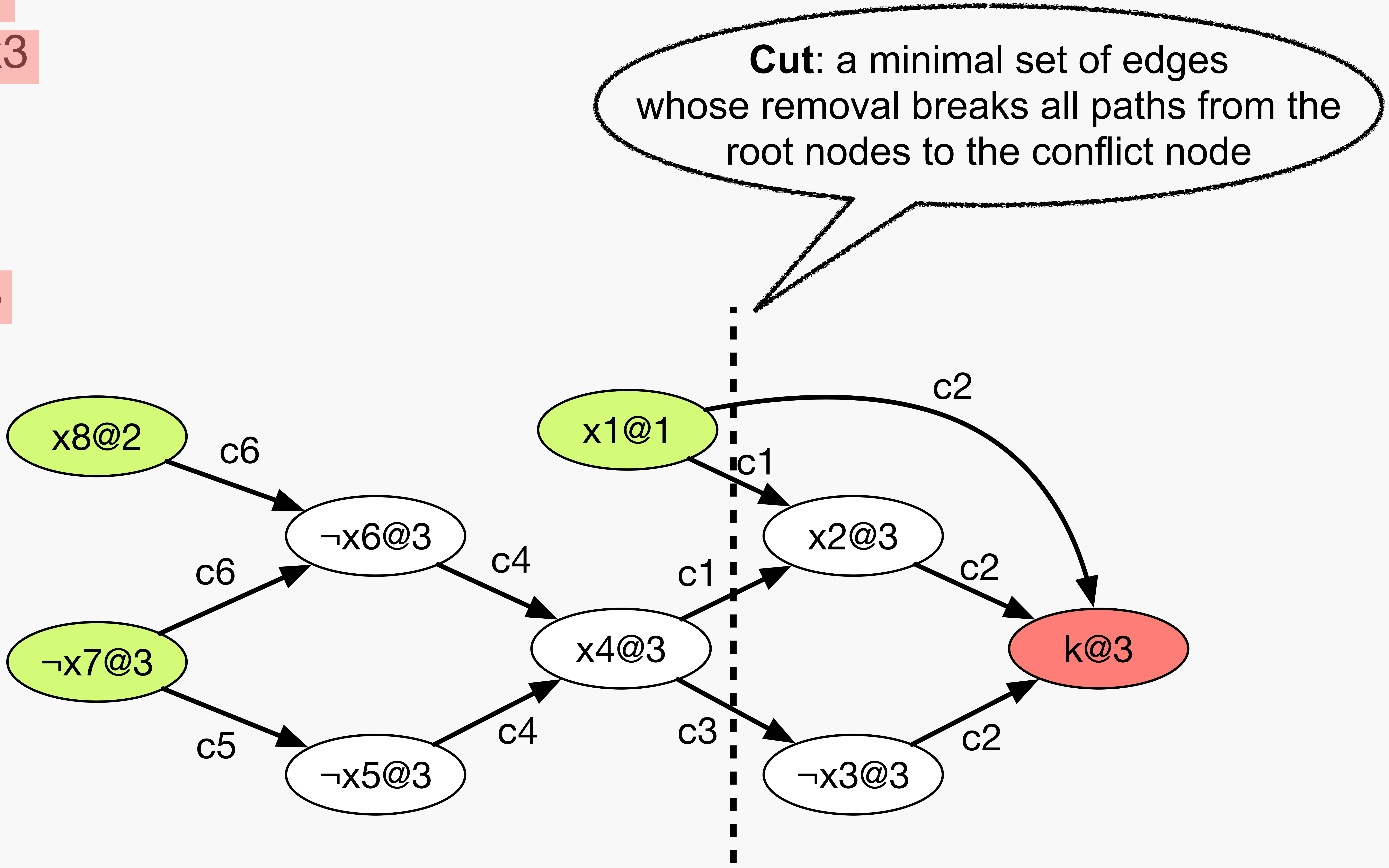
c2 : ¬x1 ∨ ¬x2 ∨ x3

c3: ¬x3 ∨ ¬x4

c4 : x4 ∨ x5 ∨ x6

c5 : ¬x5 ∨ x7

c6: ¬x6 ∨ x7 ∨ ¬x8



c1: ¬x1 ∨ x2 ∨¬x4

c2 : ¬x1 ∨ ¬x2 ∨ x3

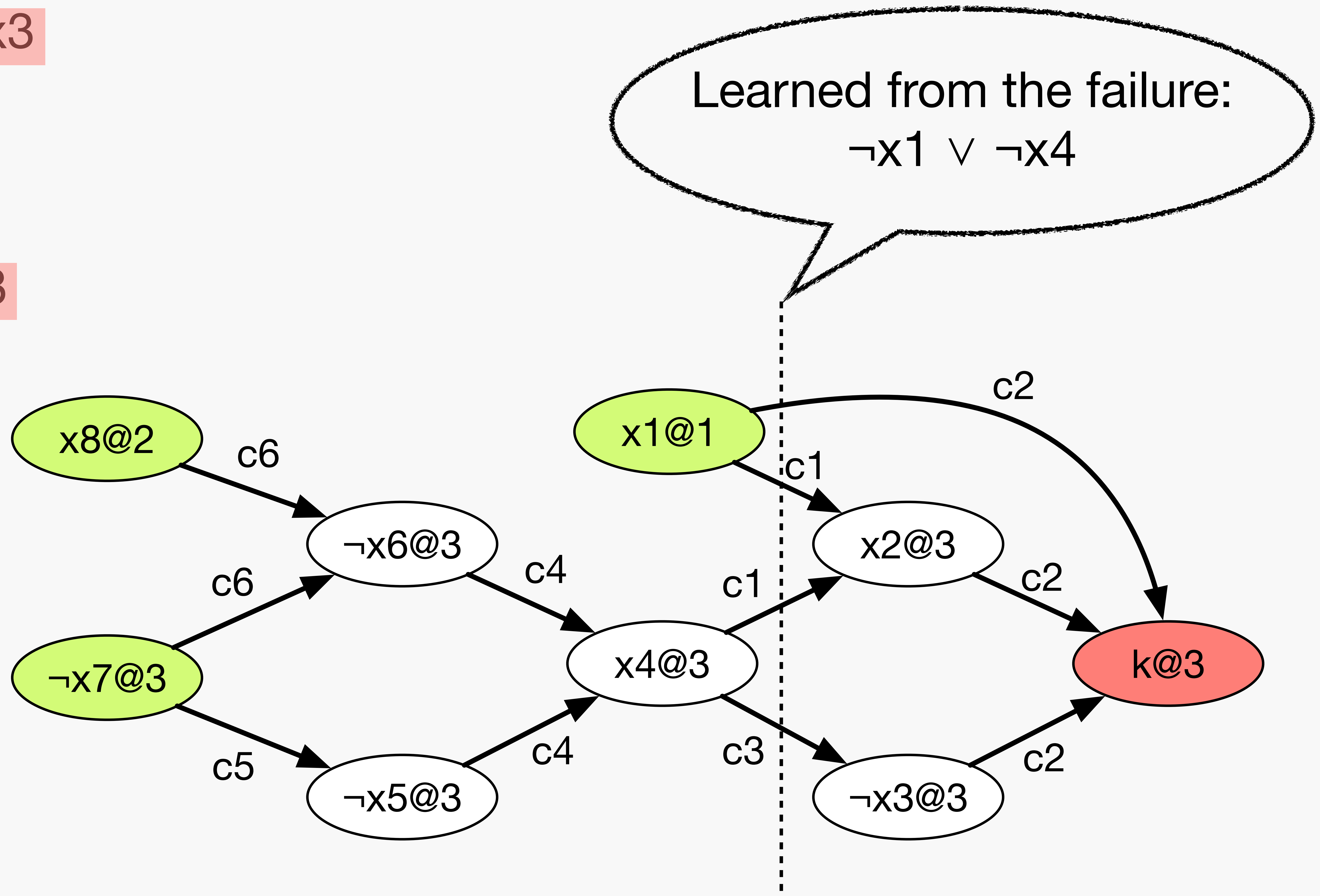
c3: ¬x3 ∨¬x4

c4 : x4 ∨ x5 ∨ x6

c5 : ¬x5 ∨ x7

c6: ¬x6 ∨ x7 ∨¬x8

c : ¬x1 ∨ ¬x4



$c1: \neg x1 \lor x2 \lor \neg x4$

$c2: \neg x1 \lor \neg x2 \lor x3$

$c3: \neg x3 \lor \neg x4$

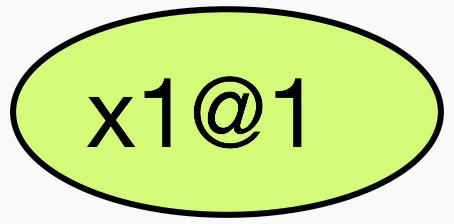
$c4: x4 \lor x5 \lor x6$

$c5: \neg x5 \lor x7$

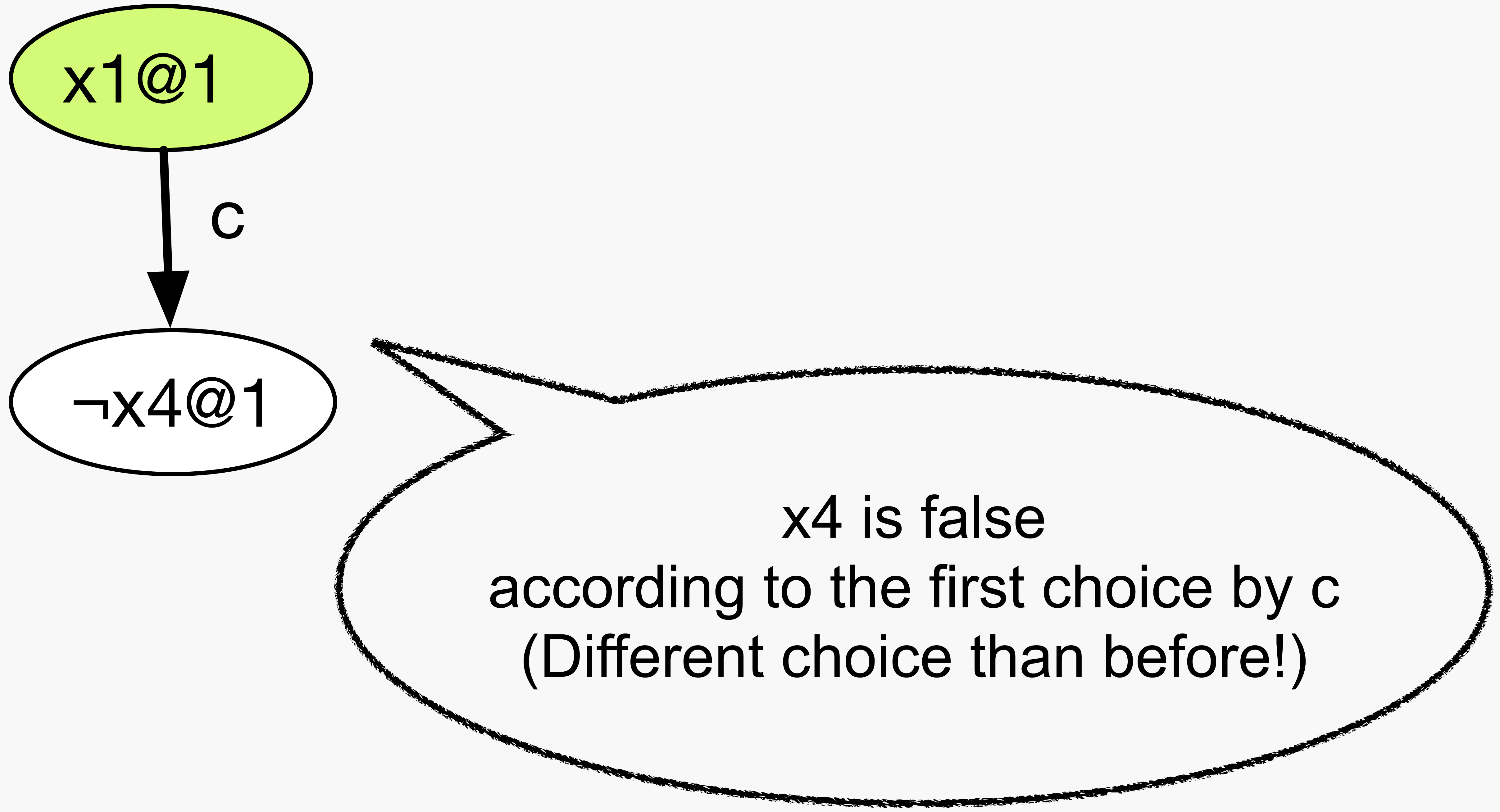
$c6: \neg x6 \lor x7 \lor \neg x8$

$c: \neg x1 \lor \neg x4$

Backtrack — remove all decisions after the first choice (but the first choice is not deleted)

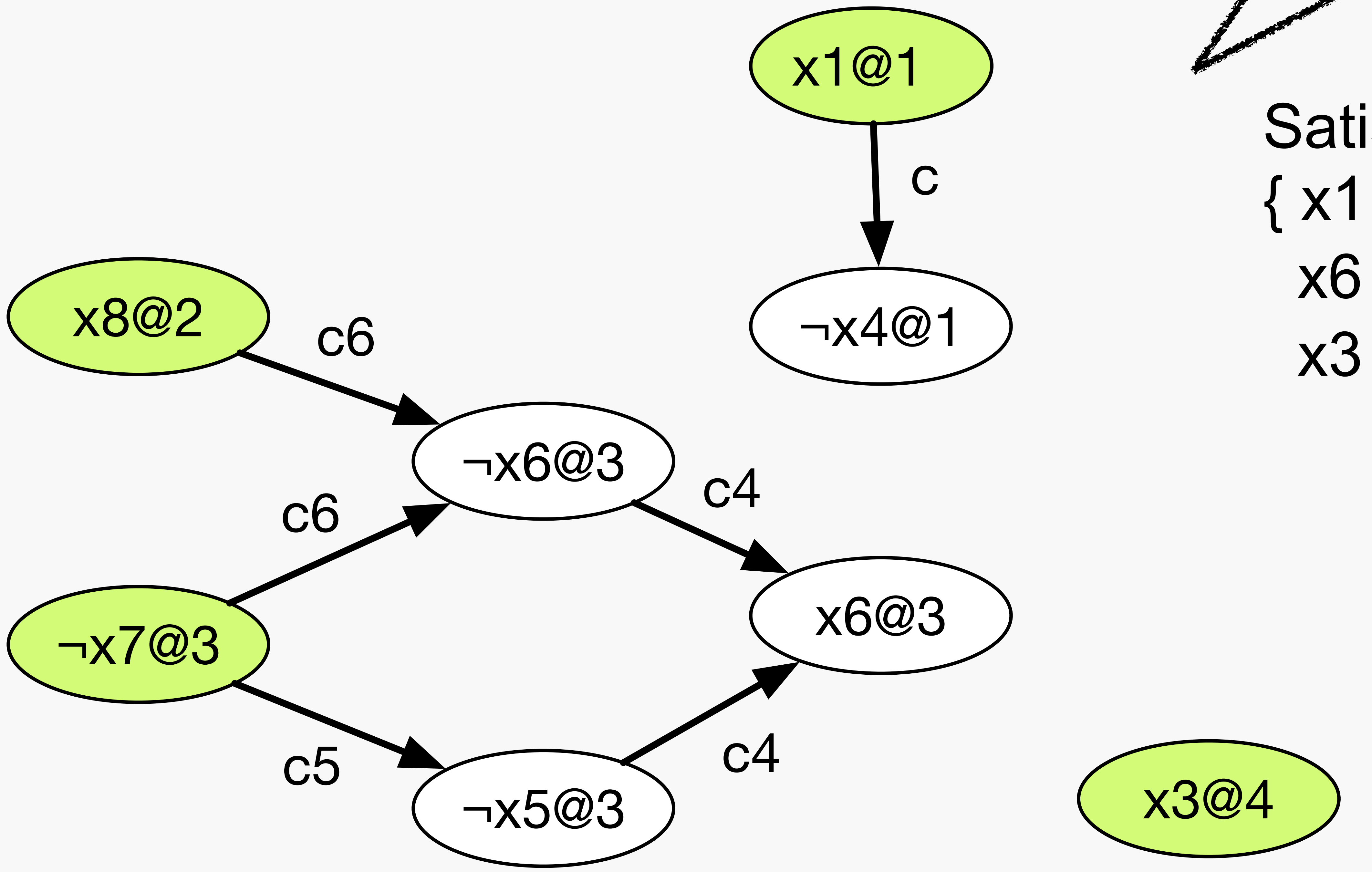


c1: ¬x1 ∨ x2 ∨¬x4
c2 : ¬x1 ∨ ¬x2 ∨ x3
c3: ¬x3 ∨¬x4
c4 : x4 ∨ x5 ∨ x6
c5 : ¬x5 ∨ x7
c6: ¬x6 ∨ x7 ∨¬x8
c : ¬x1 ∨ ¬x4



c1: ¬x1 ∨ x2 ∨ ¬x4
c2 : ¬x1 ∨ ¬x2 ∨ x3
c3: ¬x3 ∨ ¬x4
c4 : x4 ∨ x5 ∨ x6
c5 : ¬x5 ∨ x7
c6: ¬x6 ∨ x7 ∨ ¬x8
c : ¬x1 ∨ ¬x4

Make progress similarly and find a satisfying assignment



Satisfying assignment:
{ x1 ↦ ⊤, x8 ↦ ⊤, x7 ↦ ⊥,
x6 ↦ ⊥, x5 ↦ ⊥, x4 ↦ ⊥,
x3 ↦ ⊤, x2 ↦ ⊤ }

In Case of DPLL

c1: ¬x1 ∨ x2 ∨¬x4

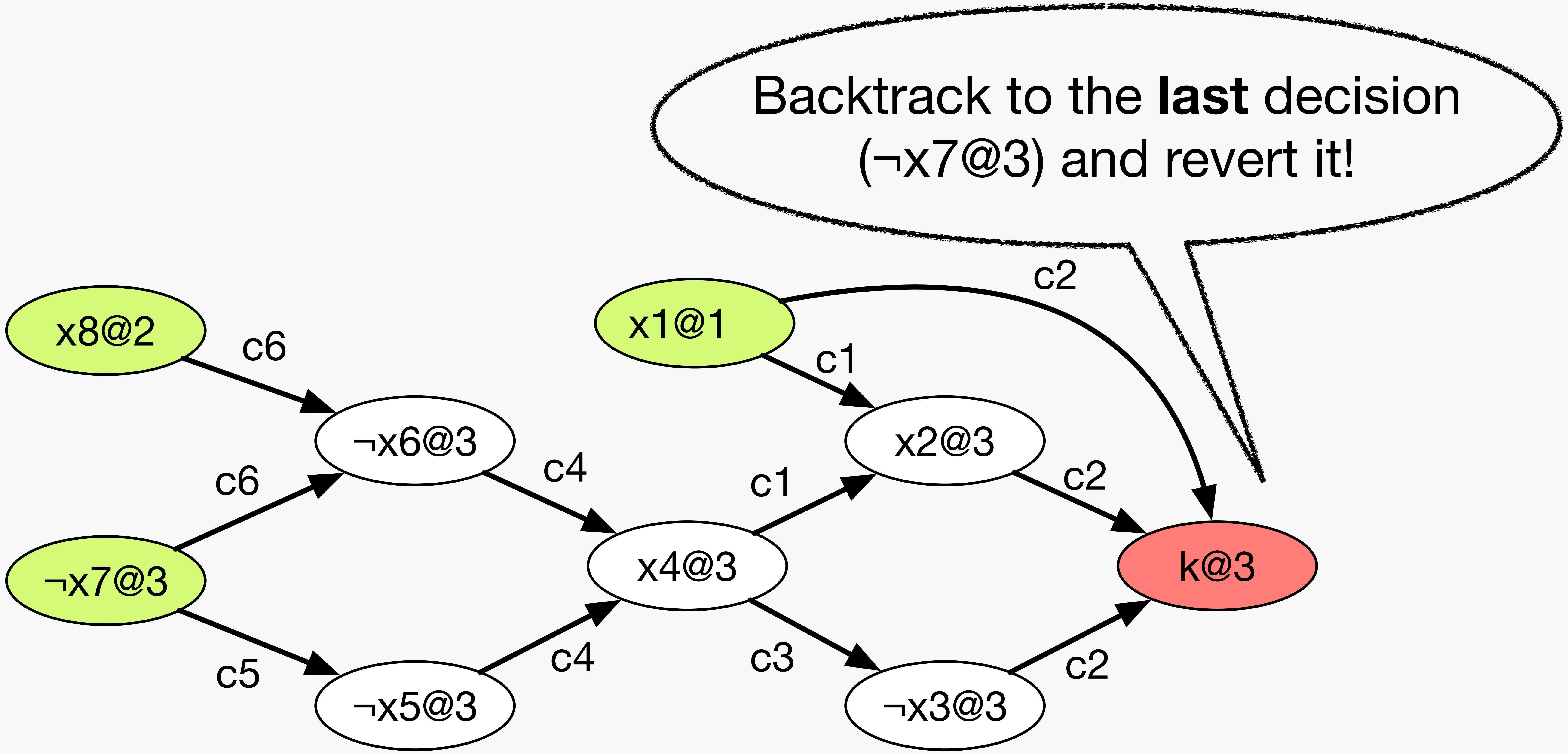
c2 : ¬x1 ∨ ¬x2 ∨ x3

c3: ¬x3 ∨¬x4

c4 : x4 ∨ x5 ∨ x6

c5 : ¬x5 ∨ x7

c6: ¬x6 ∨ x7 ∨¬x8



In Case of DPLL

c1: ¬x1 ∨ x2 ∨¬x4

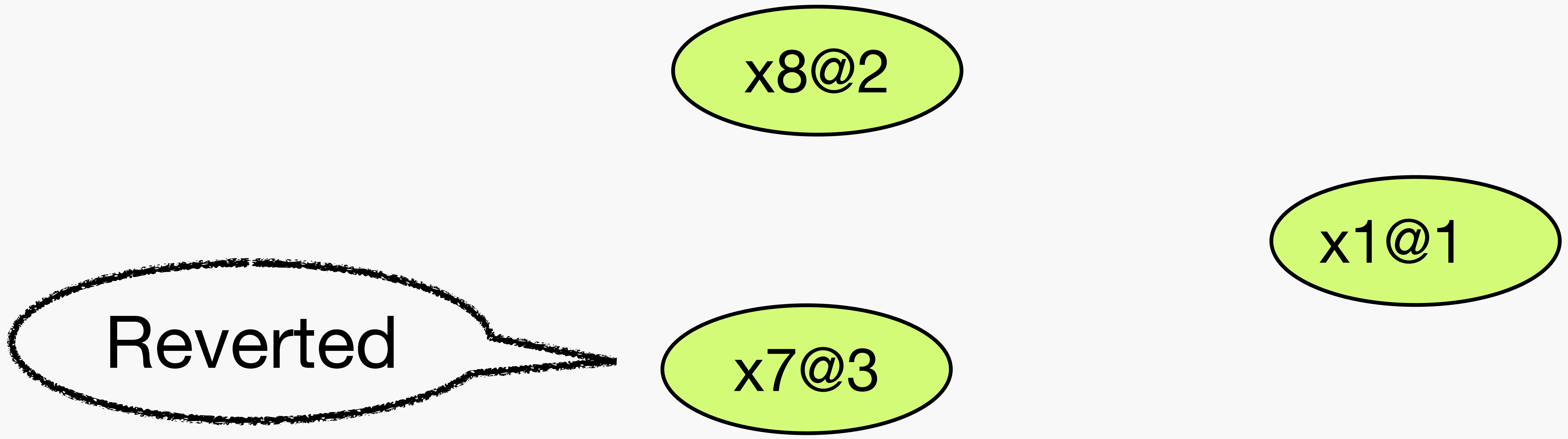
c2 : ¬x1 ∨ ¬x2 ∨ x3

c3: ¬x3 ∨¬x4

c4 : x4 ∨ x5 ∨ x6

c5 : ¬x5 ∨ x7

c6: ¬x6 ∨ x7 ∨¬x8



In Case of DPLL

c1: ¬x1 ∨ x2 ∨¬x4

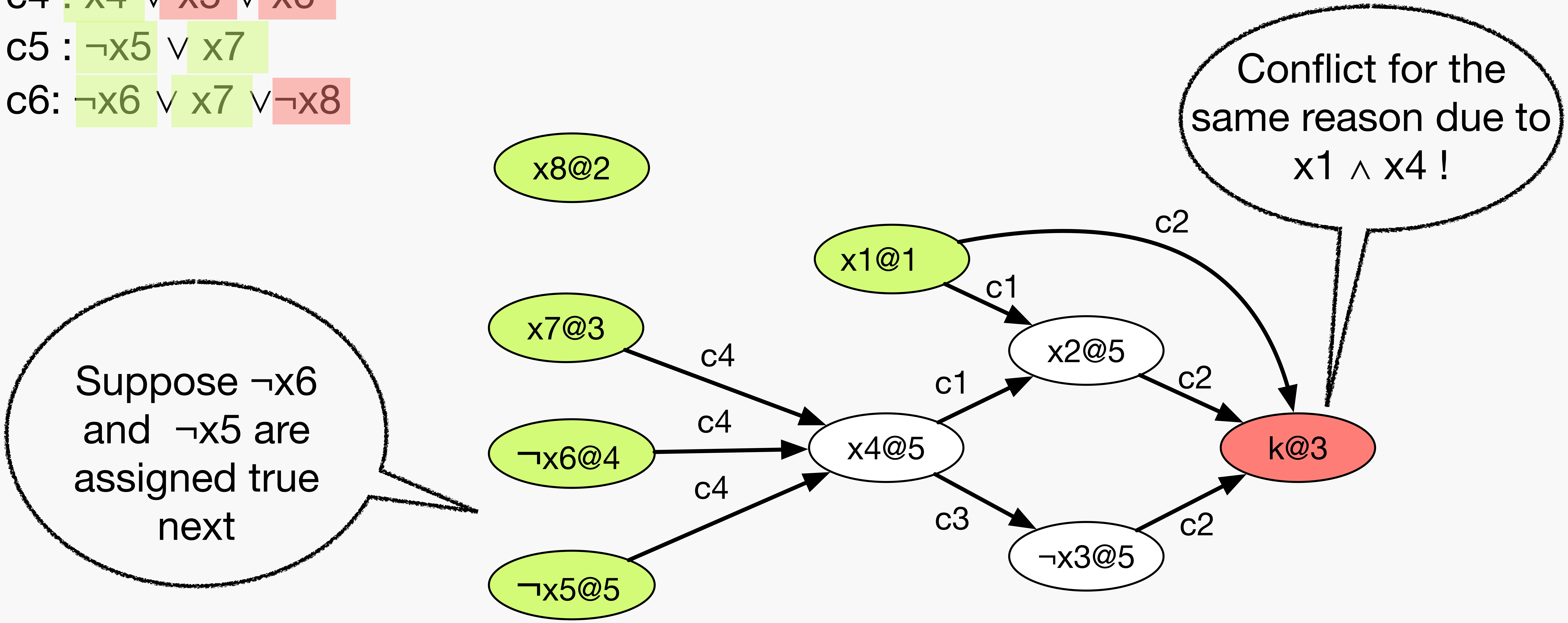
c2 : ¬x1 ∨ ¬x2 ∨ x3

c3: ¬x3 ∨¬x4

c4 : x4 ∨ x5 ∨ x6

c5 : ¬x5 ∨ x7

c6: ¬x6 ∨ x7 ∨¬x8



Formal Definition of CDCL

Decision Levels

- **Decision variable**: variable assigned in the Decide step
- **Decision level**: The level (order) in which a decision variable is assigned (starting from 1)
- Each assignment is associated with the decision level at which it occurred.
- The decision level of a variable assigned due to BCP is the decision level of the last assigned decision variable.

Quiz

- Consider a formula (¬x1 ∨ x2) ∧ (¬x3 ∨ ¬x4)
- Suppose we decide x1 = true at decision level 1
- What does BCP yield and what is the decision level of it?

x2, 1

- Suppose we decide x4 = true.
- What does BCP yield and what is the decision level of it?

¬x3, 2

Decision Levels

- If a variable x_i is assigned true (owing to either a decision or an implication) at decision level dl, we write x_i@dl.
- Assignments implied regardless of any assignments are associated with decision level 0, also called the ground level (e.g., in formula $x1 \wedge \ldots, x1 \mapsto$ true)

Status of a Clause

- Under a partial assignment (PA), a variable may be assigned (true/false) or unassigned.
- A clause can be
  - Satisfied : at least one literal is satisfied
  - Unsatisfied : all literals are assigned but non are satisfied
  - **Unit**: all but one literals are assigned but none are satisfied
  - Unresolved: all other cases
- Example : x1 ∨ x2 ∨ x3

x_1	x_2	x_3	C
1	0		Satisfied
0	0	0	Unsatisfied
0	0		Unit
	0		Unresolved

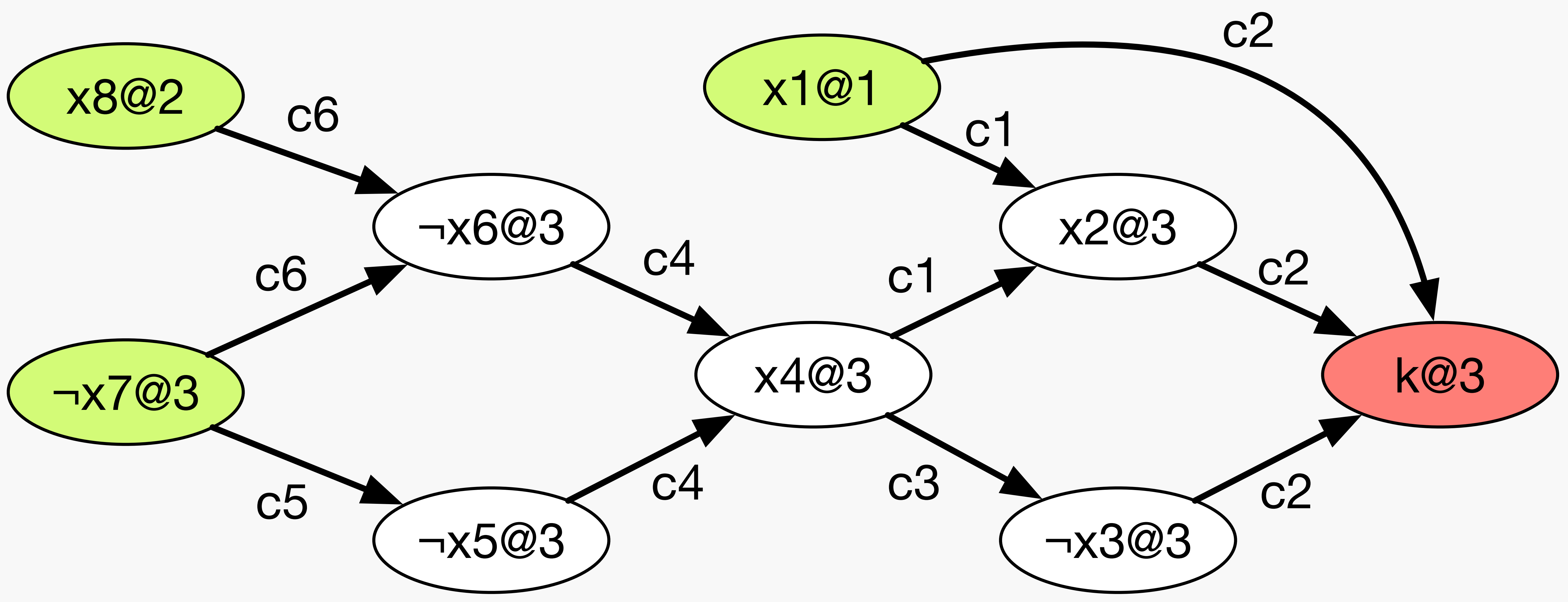
Antecedent

- For a given unit clause C with an unassigned literal l, we say that l is implied by C and that C is the antecedent clause of l, denoted by Antecedent(l)
  - Suppose we have the partial assignment $\{x1 \mapsto true, x4 \mapsto true\}$ and the clause $C := (\neg x1 \vee \neg x4 \vee x3)$, Antecedent(x3) = C

Implication Graph

- An implication graph is a labeled directed acyclic graph G(V, E), where:
  - Each $v \in V$ is a literal in the current PA and its decision level
  - $E = \{(v_i, v_j) \mid v_i, v_j \in V, \neg v_i \in Antecedent(v_j)\}$: the set of directed edges
  - G can also contain a single conflict node labeled with k and incoming edges $\{(v, \kappa) \mid \neg v \in c\}$ labeled with c for some conflicting clause c.

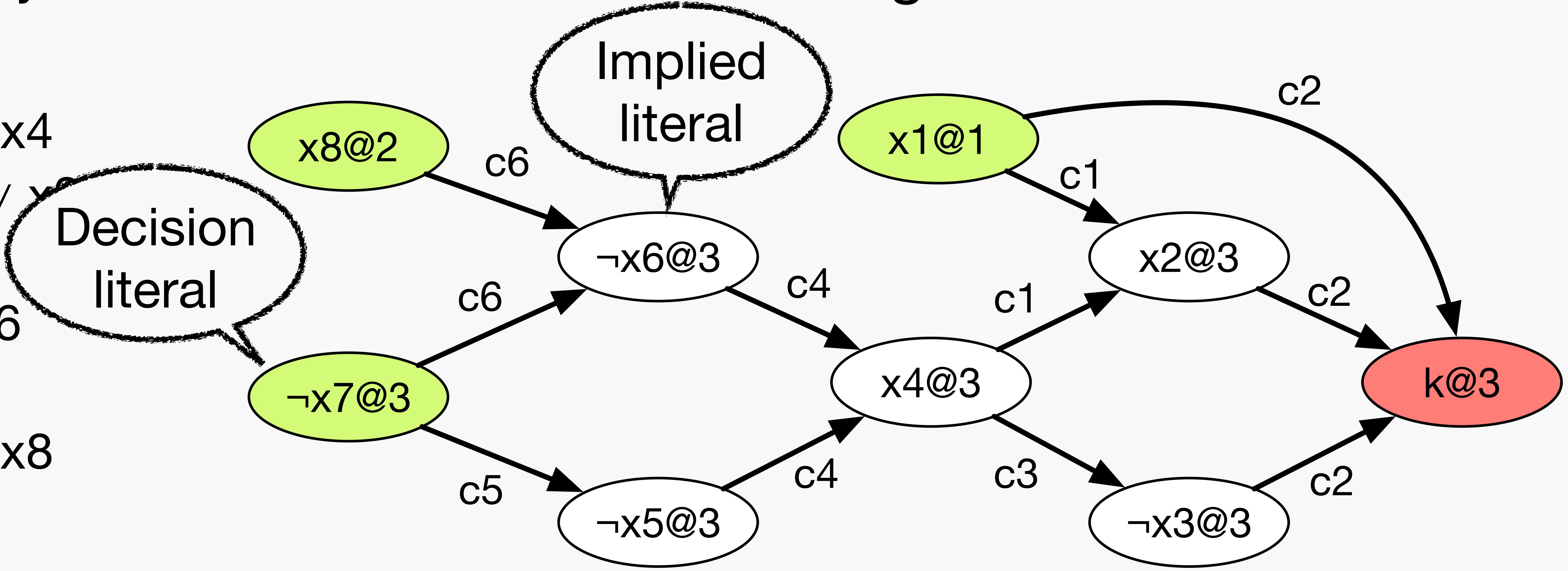
c1: ¬x1 ∨ x2 ∨¬x4
c2 : ¬x1 ∨ ¬x2 ∨ x3
c3: ¬x3 ∨¬x4
c4 : x4 ∨ x5 ∨ x6
c5 : ¬x5 ∨ x7
c6: ¬x6 ∨ x7 ∨¬x8



Implication Graph

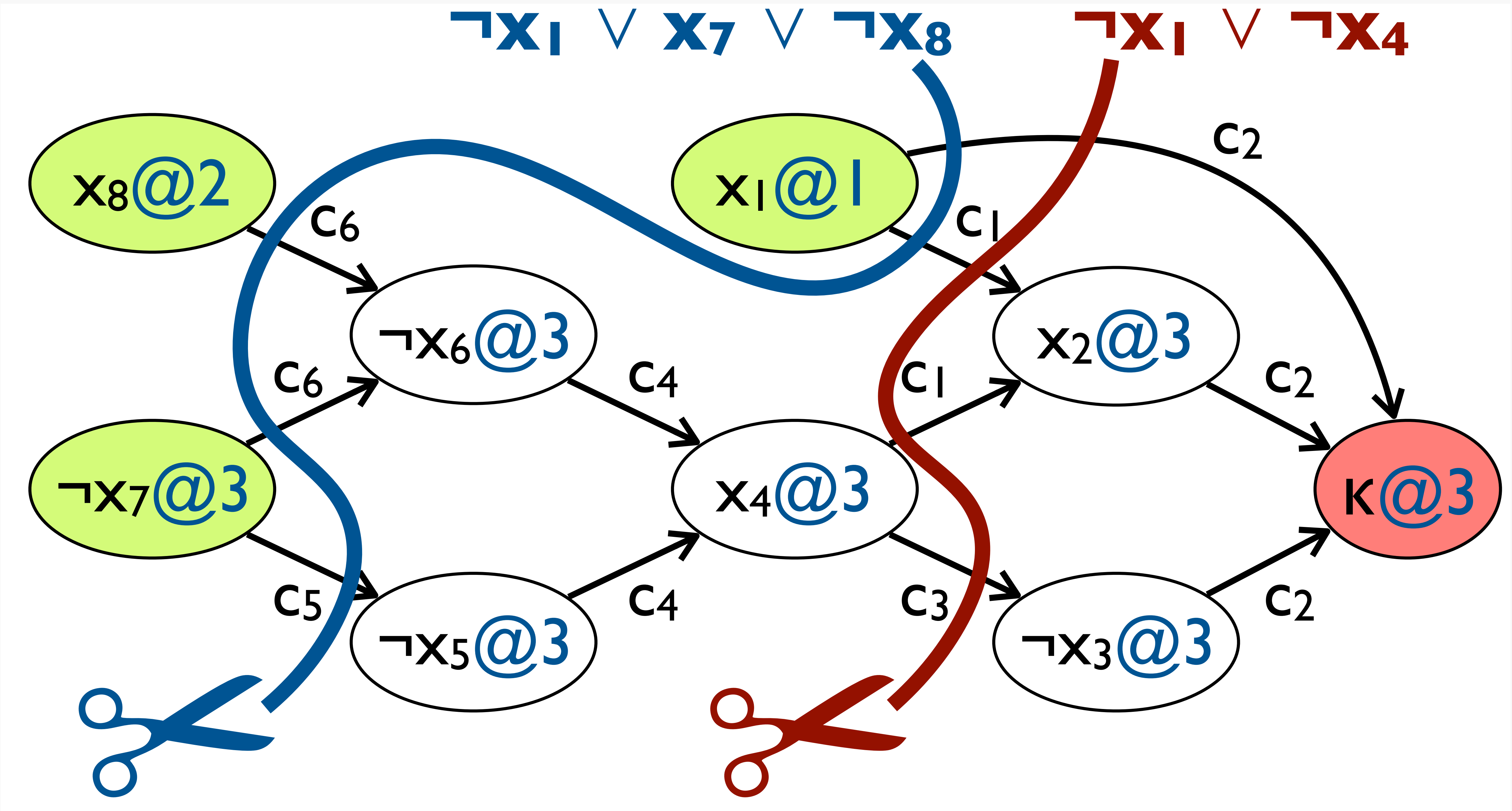
- An implication graph is a labeled directed acyclic graph G(V, E), where:
  - Each v ∈ V is a literal in the current PA and its decision level
  - $E = \{(v_i, v_j) \mid v_i, v_j \in V, \neg v_i \in Antecedent(v_j)\}$: the set of directed edges
  - G can also contain a single conflict node labeled with k and incoming edges $\{(v, \kappa) \mid \neg v \in c\}$ labeled with c for some conflicting clause c.

c1: ¬x1 ∨ x2 ∨¬x4
c2 : ¬x1 ∨ ¬x2 ∨ x
c3: ¬x3 ∨¬x4
c4 : x4 ∨ x5 ∨ x6
c5 : ¬x5 ∨ x7
c6: ¬x6 ∨ x7 ∨¬x8



Conflict Clause

- A conflict clause is a clause implied by the original formula that blocks PAs that lead to the current conflict. Multiple conflict clauses may exist.
  - Every cut that separates root nodes from the conflict node defines a valid conflict clause.
  - Which one is better?



Cut: a minimal set of edges whose removal breaks all paths from the root nodes to the conflict node
It bipartitions the nodes into the reason side (the side that includes all the roots) and the conflict side.

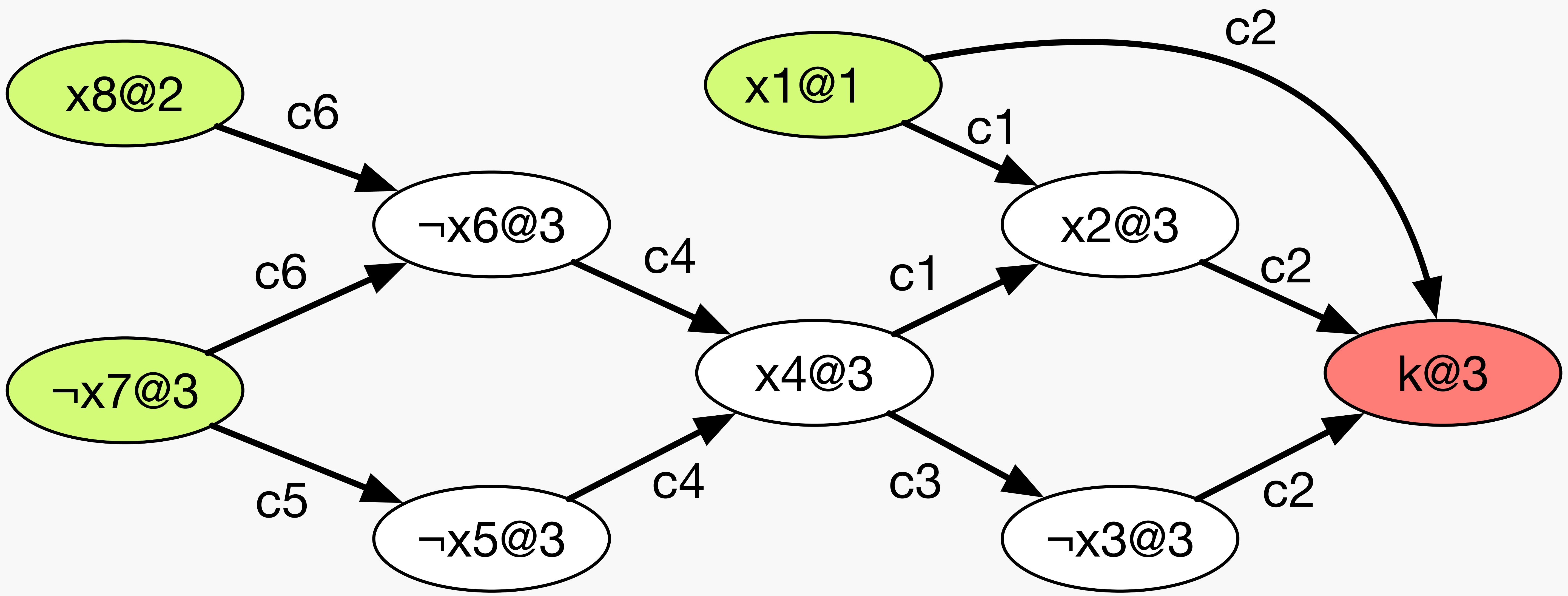
Figure from https://courses.cs.washington.edu/courses/cse507/16sp/lectures/L2.pdf

Conflict Clause

- Why are conflict clauses necessary?
  - To prevent bad partial assignments by deriving contradiction as quickly as possible
- To this end, **smaller conflict clauses are better.**
  - c1’: ¬x1 ∨ x7 ∨ ¬x8 vs. c2’ : ¬x1 ∨ ¬x4
  - Number of PAs satisfying c1’ ≥ Number of PAs satisfying c2’
  - Therefore, c2’ has better pruning power (can discard more unsatisfying assignments)

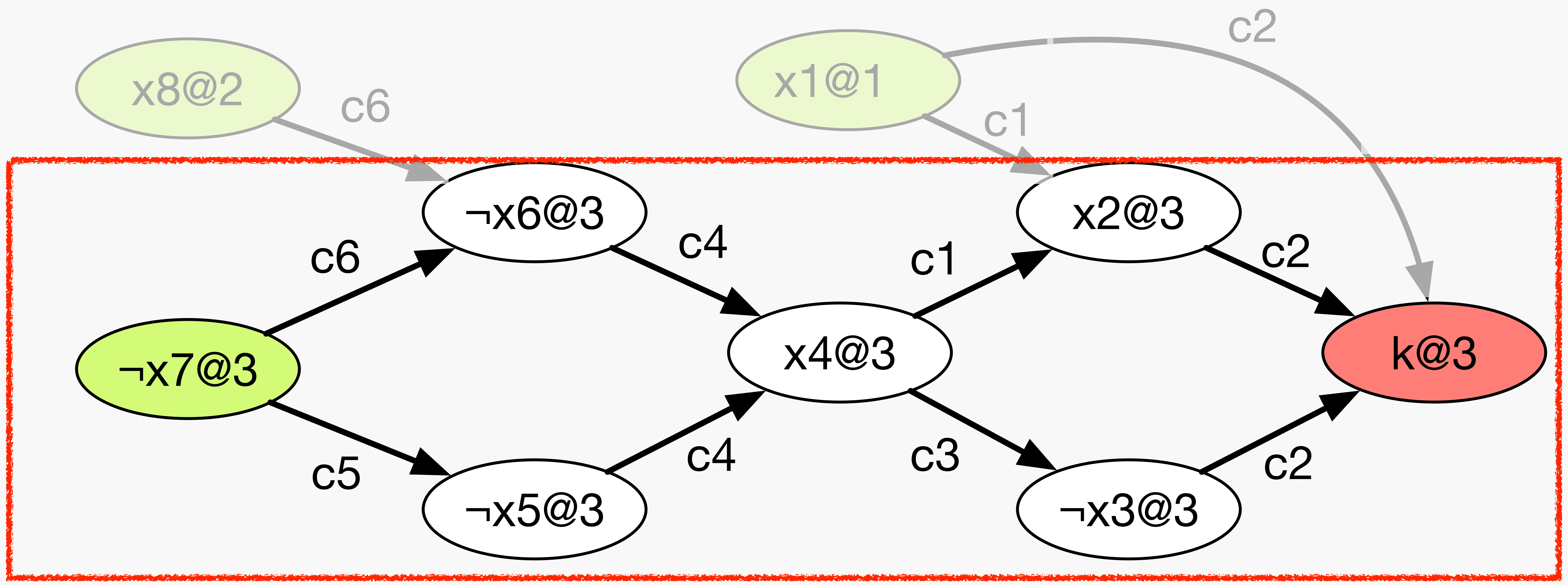
Unique Implication Point (UIP)

- Given a partial conflict graph corresponding to the decision level of the conflict



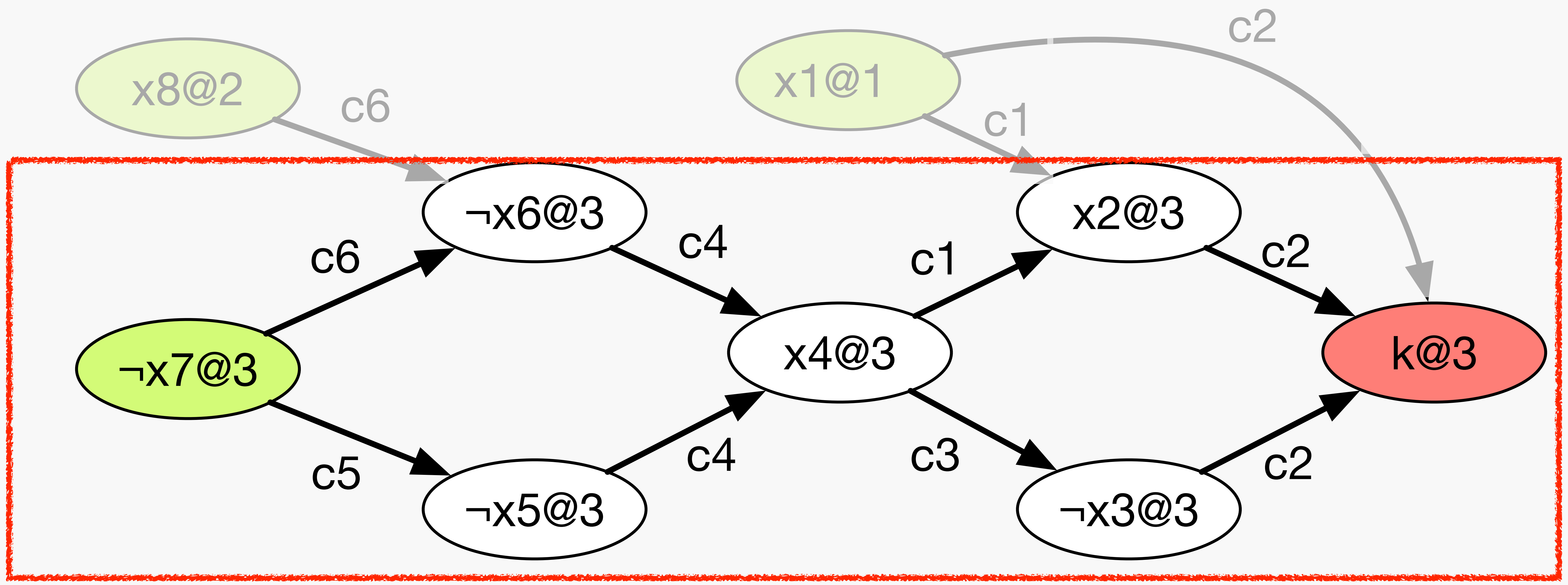
Unique Implication Point (UIP)

- Given a partial conflict graph corresponding to the decision level of the conflict



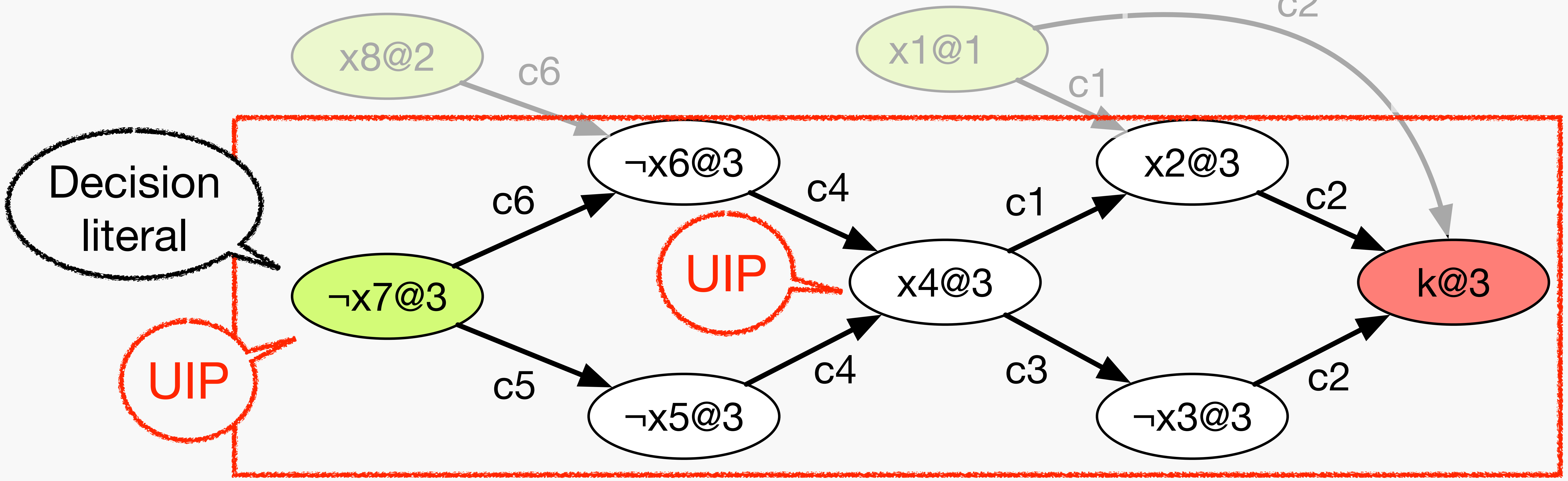
Unique Implication Point (UIP)

- Given a partial conflict graph corresponding to the decision level of the conflict, a unique implication point (UIP) is any node other than the conflict node that is on *all paths* from the decision literal to the conflict node



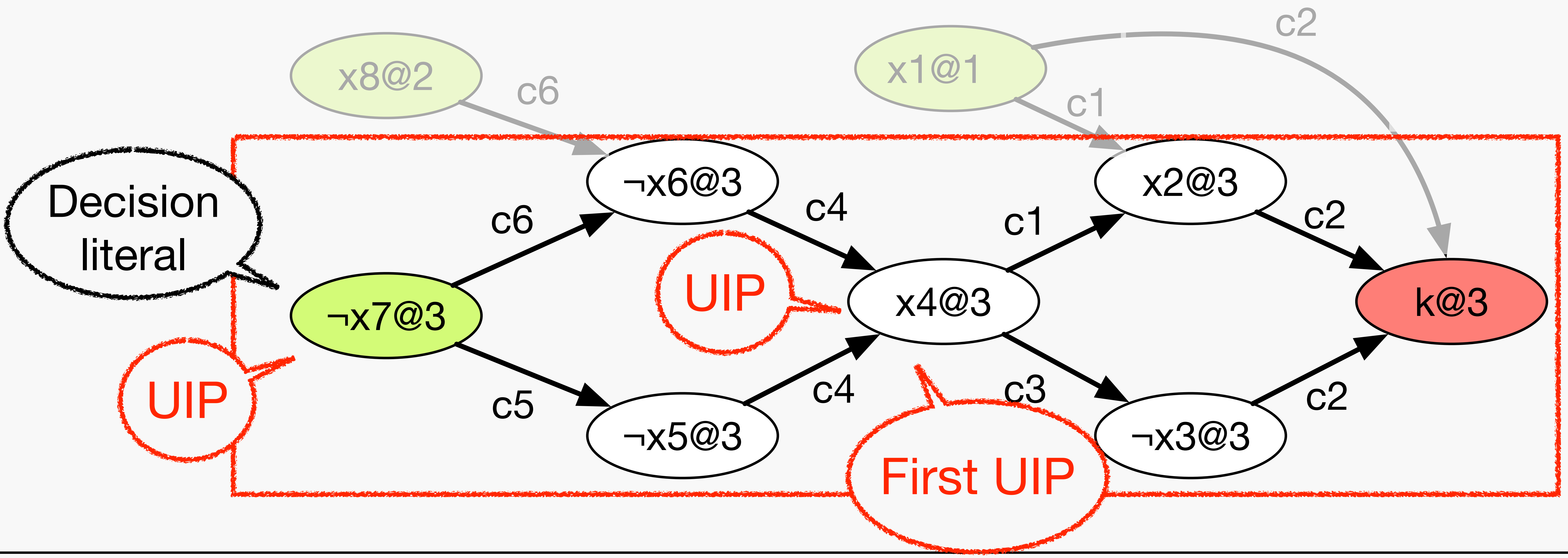
Unique Implication Point (UIP)

- Given a partial conflict graph corresponding to the decision level of the conflict, a unique implication point (UIP) is any node other than the conflict node that is on *all paths* from the decision literal to the conflict node



Unique Implication Point (UIP)

- Given a partial conflict graph corresponding to the decision level of the conflict, a unique implication point (UIP) is any node other than the conflict node that is on *all paths* from the decision literal to the conflict node
- A first UIP is a UIP that is closest to the conflict node.



Unique Implication Point (UIP)

- Any decision literal is a UIP by definition.
- Other UIPs (if exists) are implied literals at the decision level of the conflict.
- There is always a **single** UIP closest to the conflict node (why?)

Unique Implication Point (UIP)

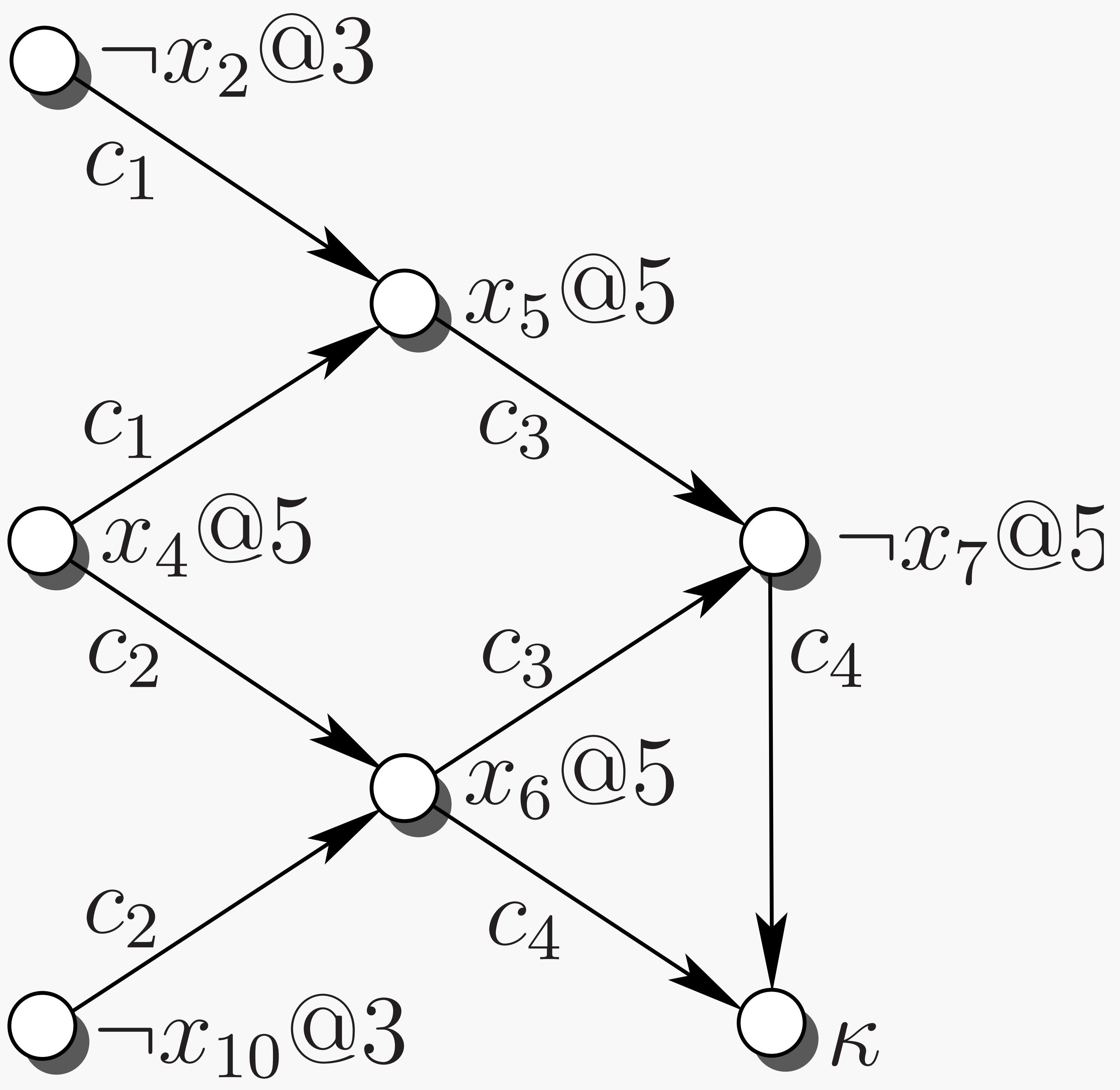
- Any decision literal is a UIP by definition.
- Other UIPs (if exists) are implied literals at the decision level of the conflict.
- There is always a **single** UIP closest to the conflict node (why?)
  - => All paths to a single conflict node should pass through the first UIP which cannot be more than two.
  - A first UIP is *a single literal* which is a common cause of the conflict in the current decision level.

Exercise

- Consider $F = c_1 \wedge c_2 \wedge c_3 \wedge c_4$ where

$$
\begin{aligned}
c_1 &= (\neg x_4 \vee x_2 \vee x_5) \\
c_2 &= (\neg x_4 \vee x_{10} \vee x_6) \\
c_3 &= (\neg x_5 \vee \neg x_6 \vee \neg x_7) \\
c_4 &= (\neg x_6 \vee x_7)
\end{aligned}
$$

  Which node is the first UIP?



CDCL Algorithm

- `A`: assignment made so far
- `BCP(F, A)`: Boolean constraint propagation over F after assigning variables using A
- `dl`: current decision level
- `Decide(F)`: choose a variable and assign a value
- `b`: level to backtrack to
- `c`: learned conflict clause
- `Backtrack(F,A,b)`: remove all variable assignments made after `b` (but assignments at level `b` not deleted)

```
function CDCL (F) =
A := {}
F′ := BCP(F, A);
if F′ = ⊤ then return SAT
else if F′ = ⊥ then return UNSAT
dl := 0
while hasUnassignedVars(F,A) do
  dl := dl + 1
  <x,v> := Decide(F)
  A := A{x ↦ v}
  F := BCP(F, A)
  while Conflict(F) do
    <b,c> := AnalyzeConflict(F,A)
    F := F ∧ c
    if b < 0 then return UNSAT
    else
      A := Backtrack(F,A,b)
      dl := b
return SAT
```

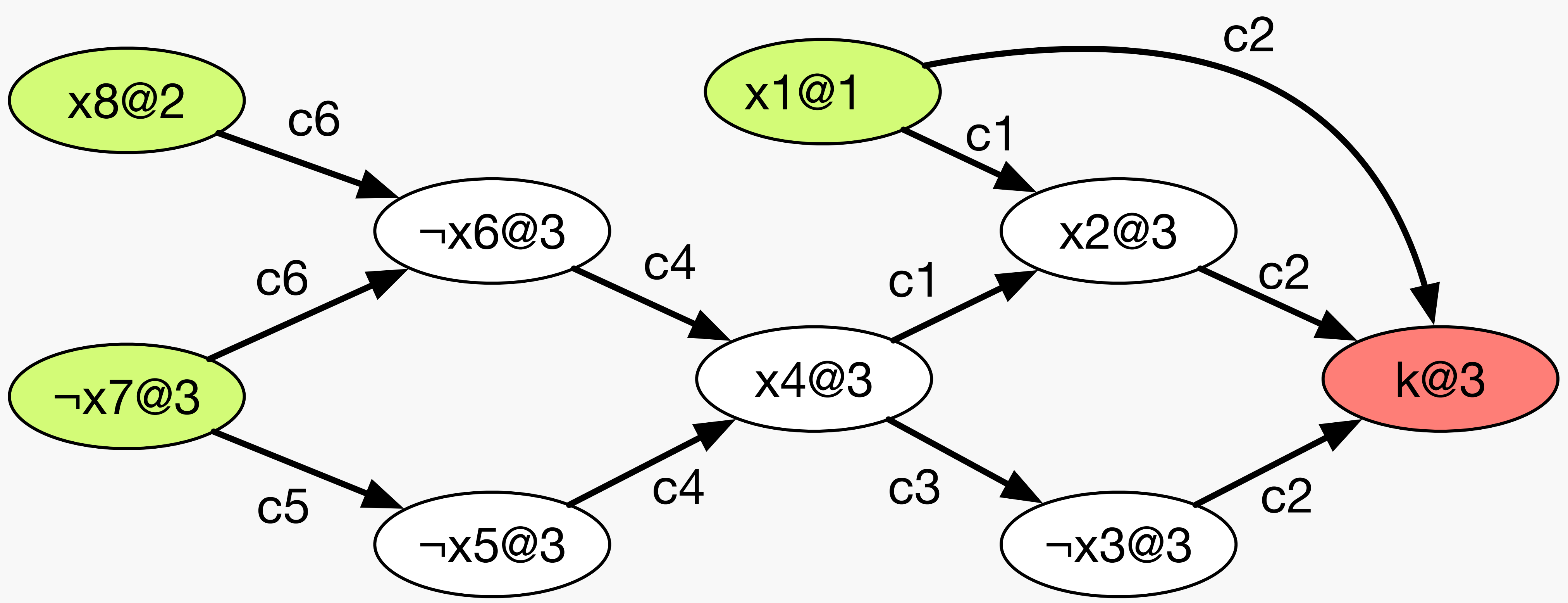
AnalyzeConflict

- Two goals:
  - Deriving conflict clauses
  - Decide what level to backtrack to
- We want to backtrack to a level that makes conflict clause `c` an asserting clause in the next step
  - Asserting clause is a clause with exactly one unassigned literal
- Hence, if we make `c` an asserting clause, BCP will force at least one assignment

```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteralAtLevel(c,d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  c := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```

AnalyzeConflict

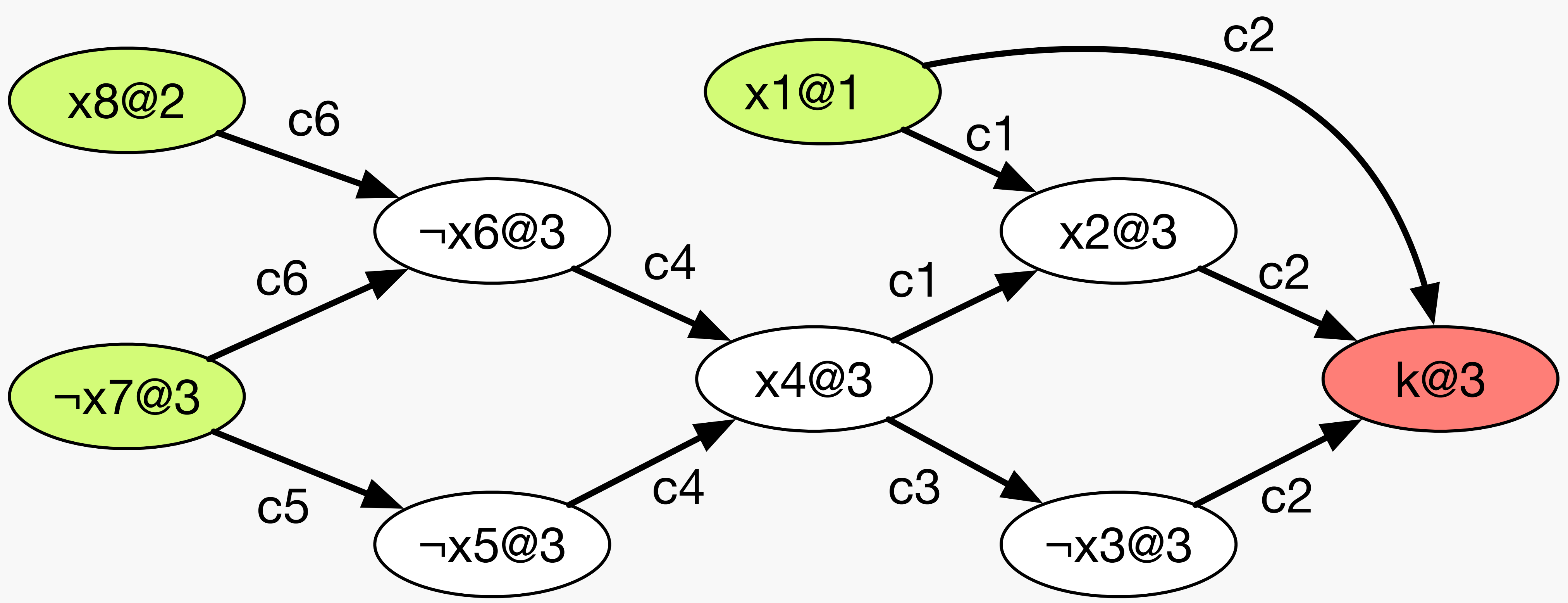
d = 3




```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteralAtLevel(c,d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  c := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```

AnalyzeConflict

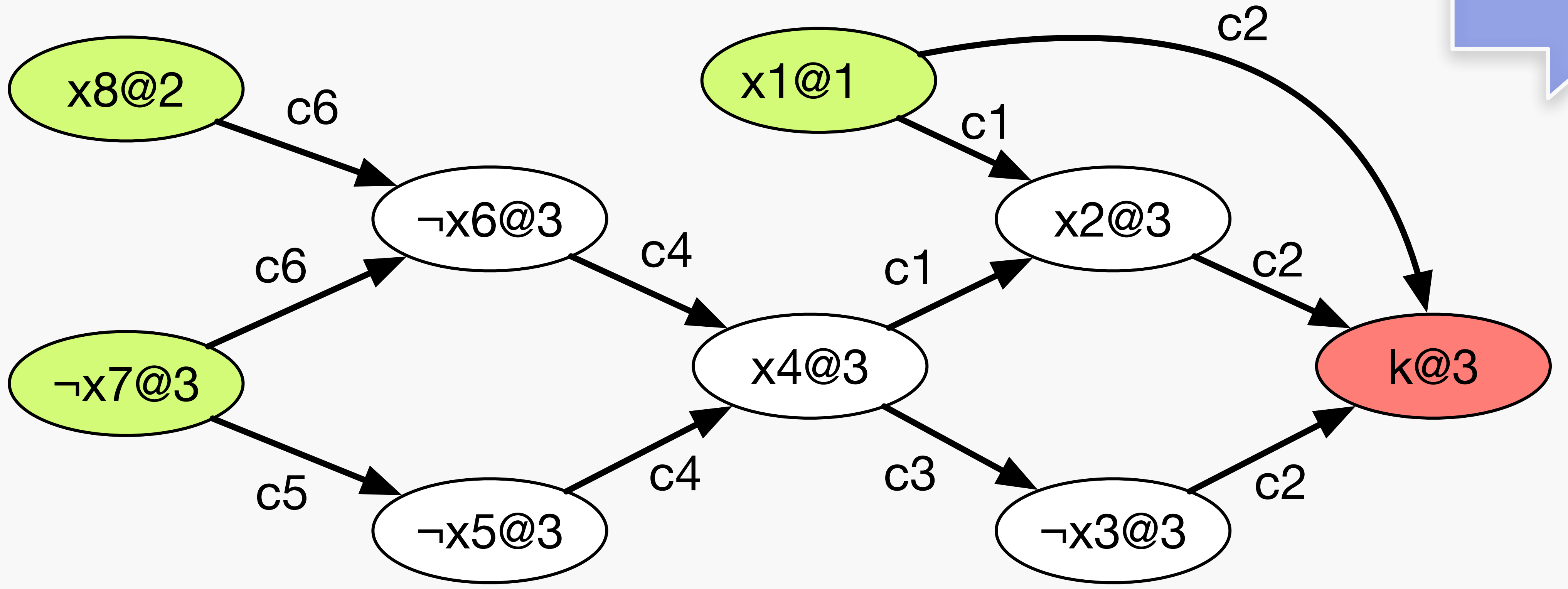
d = 3
c = c2




```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteralAtLevel(c,d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  c := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```

AnalyzeConflict

d = 3
c = c2
lit = x2
x = x2
ante = c1




```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteralAtLevel(c,d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  c := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```

AnalyzeConflict

- `Resolve(c,ante,x)`: unit resolution rule
  - Suppose `c` = $\alpha_1 \vee \cdots \vee \alpha_n \vee x$, `ante` = $\beta_1 \vee \cdots \vee \beta_m \vee \neg x$, by the rule

$$\frac{\alpha_1 \vee \cdots \vee \alpha_n \vee x \qquad \beta_1 \vee \cdots \vee \beta_m \vee \neg x}{\alpha_1 \vee \cdots \vee \alpha_n \vee \beta_1 \vee \cdots \vee \beta_m}$$

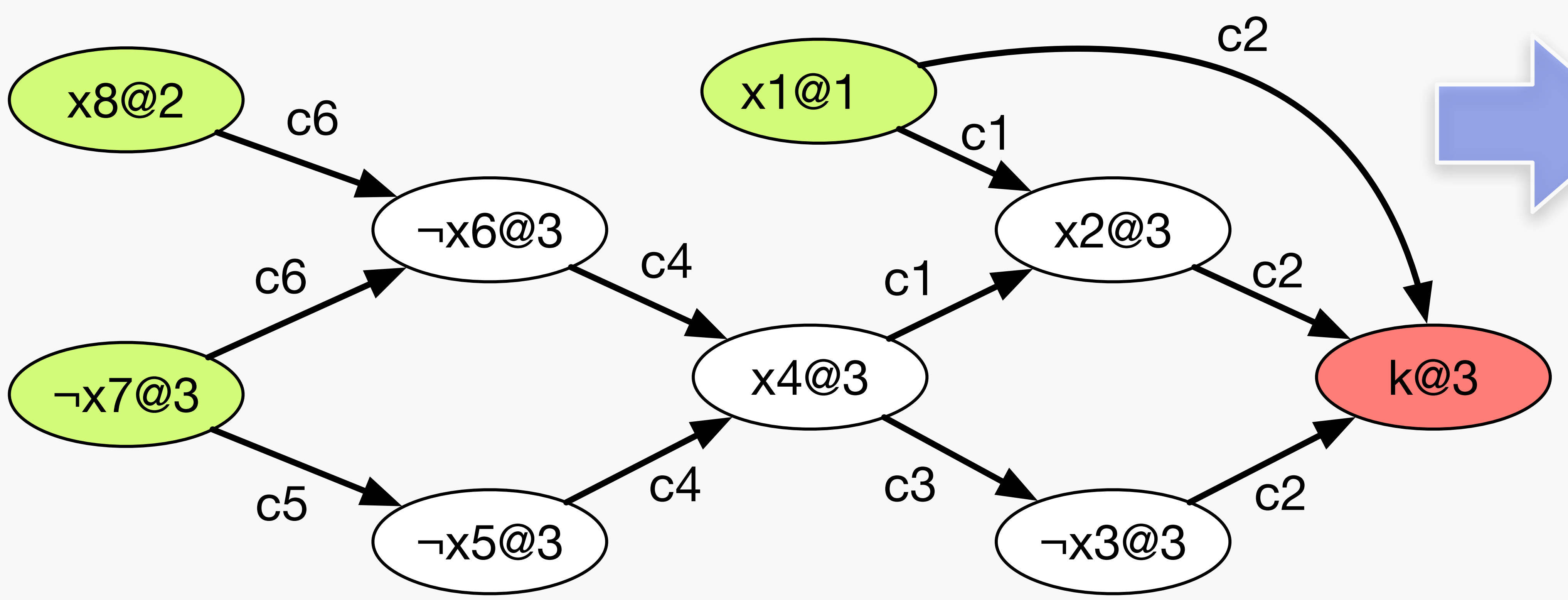
`Resolve(c,ante,x)` = $\alpha_1 \vee \cdots \vee \alpha_n \vee \beta_1 \vee \cdots \vee \beta_m$

```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteral(d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  d := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```

AnalyzeConflict

d = 3
c = c2
lit = x2
x = x2
ante = c1

Resolve(c, ante, x) =
= Resolve (¬x1 ∨ ¬x2 ∨ x3, ¬x1 ∨ x2 ∨¬x4, x2)
= ¬x1 ∨ x3 ∨ ¬x4




```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteralAtLevel(c,d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  c := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```

c1: ¬x1 ∨ x2 ∨¬x4
c2 : ¬x1 ∨ ¬x2 ∨ x3
c3: ¬x3 ∨¬x4
c4 : x4 ∨ x5 ∨ x6
c5 : ¬x5 ∨ x7
c6: ¬x6 ∨ x7 ∨¬x8

AnalyzeConflict

d = 3   c = ¬x1 ∨ x3 ∨ ¬x4   lit = x2

x = x2 ante = c1

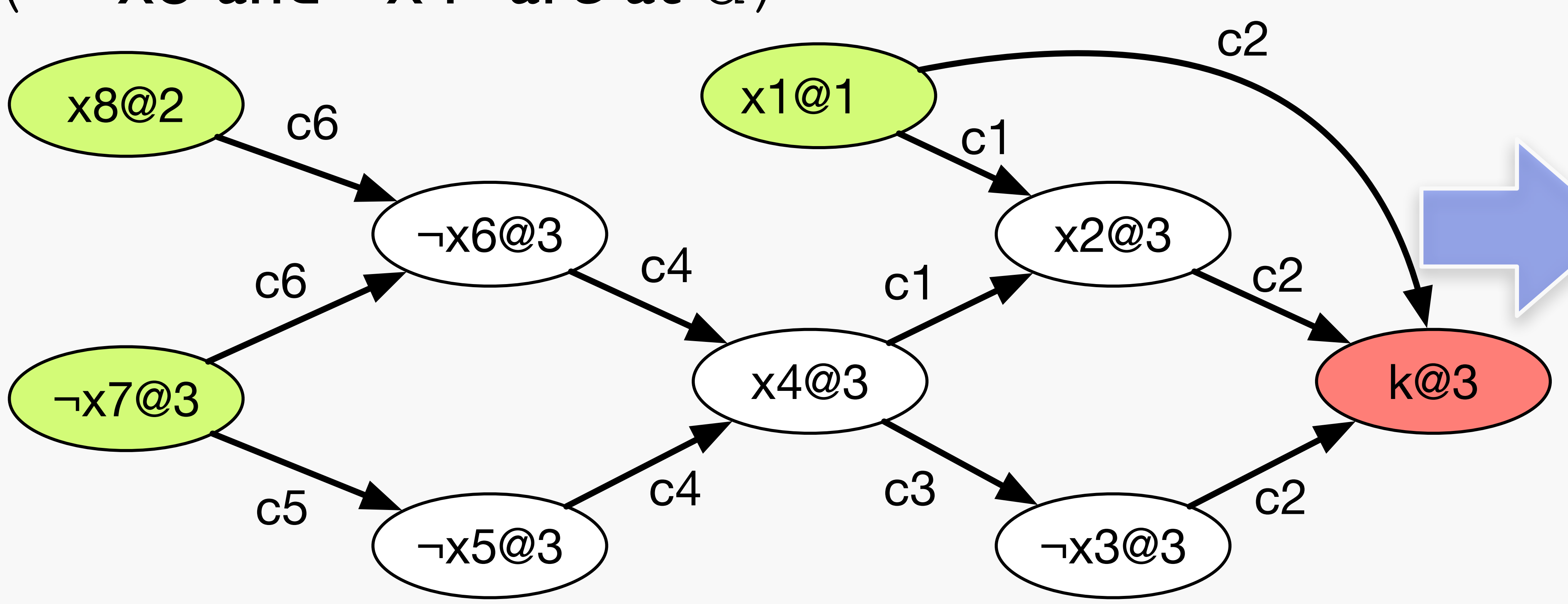
Resolve(c, ante, x) =

= Resolve(¬x1 ∨ ¬x2 ∨ x3, ¬x1 ∨ x2 ∨¬x4, x2)

= ¬x1 ∨ x3 ∨ ¬x4

oneLitAtLevel(c,d) = False

(∵ x3 and ¬x4 are at d)



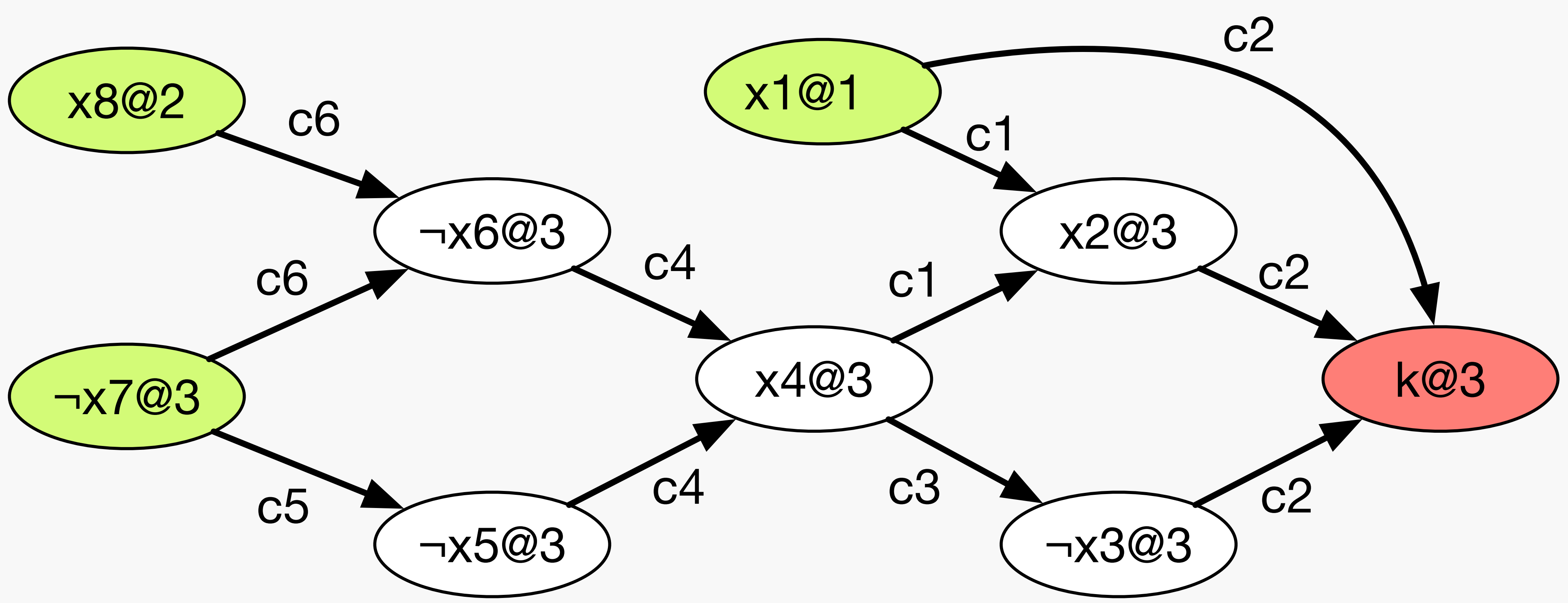

```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteralAtLevel(c,d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  c := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```

c1: ¬x1 ∨ x2 ∨¬x4
c2 : ¬x1 ∨ ¬x2 ∨ x3
c3: ¬x3 ∨¬x4
c4 : x4 ∨ x5 ∨ x6
c5 : ¬x5 ∨ x7
c6: ¬x6 ∨ x7 ∨¬x8

AnalyzeConflict

d = 3   c = ¬x1 ∨ x3 ∨ ¬x4   lit = x3

x = x2   ante = c1




```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteralAtLevel(c,d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  c := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```

c1: ¬x1 ∨ x2 ∨¬x4

c2 : ¬x1 ∨ ¬x2 ∨ x3

c3: ¬x3 ∨¬x4

c4 : x4 ∨ x5 ∨ x6

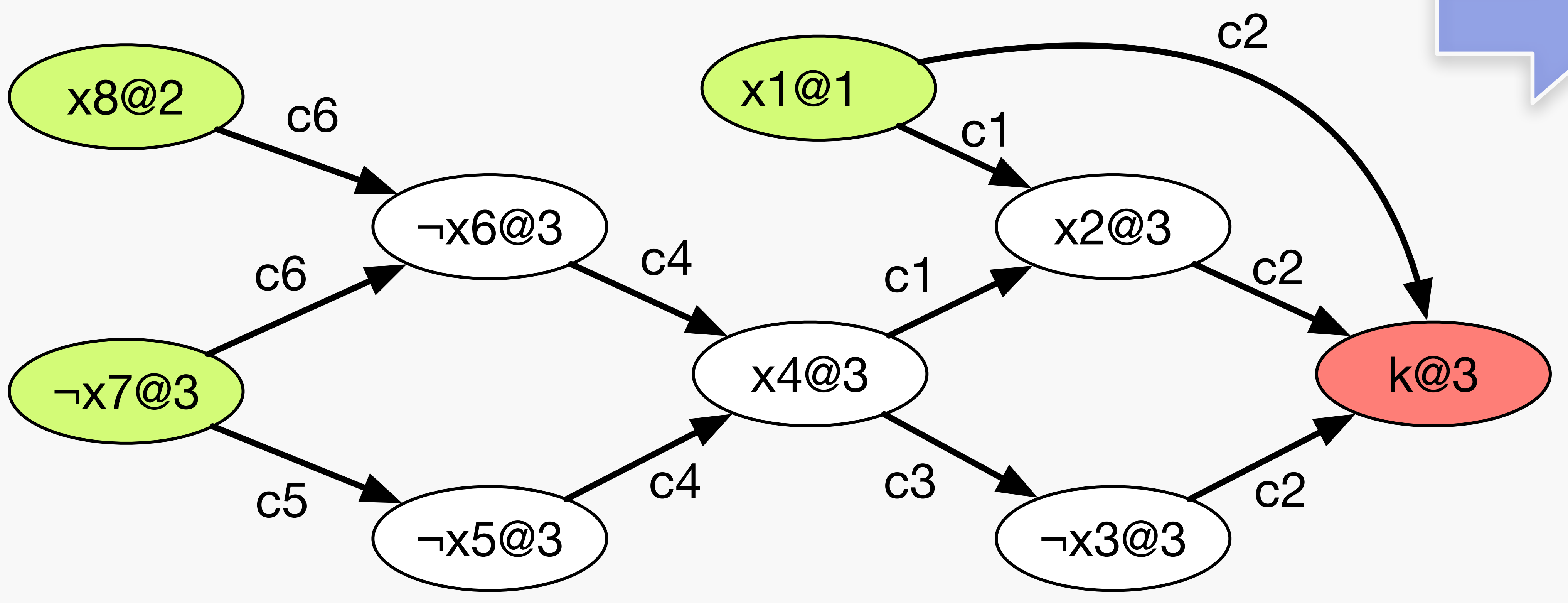
c5 : ¬x5 ∨ x7

c6: ¬x6 ∨ x7 ∨¬x8

AnalyzeConflict

d = 3   c = ¬x1 ∨ x3 ∨ ¬x4   lit = x3

x = x3   ante = c3




```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteralAtLevel(c,d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  c := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```

c1: ¬x1 ∨ x2 ∨¬x4

c2 : ¬x1 ∨ ¬x2 ∨ x3

c3: ¬x3 ∨¬x4

c4 : x4 ∨ x5 ∨ x6

c5 : ¬x5 ∨ x7

c6: ¬x6 ∨ x7 ∨¬x8

AnalyzeConflict

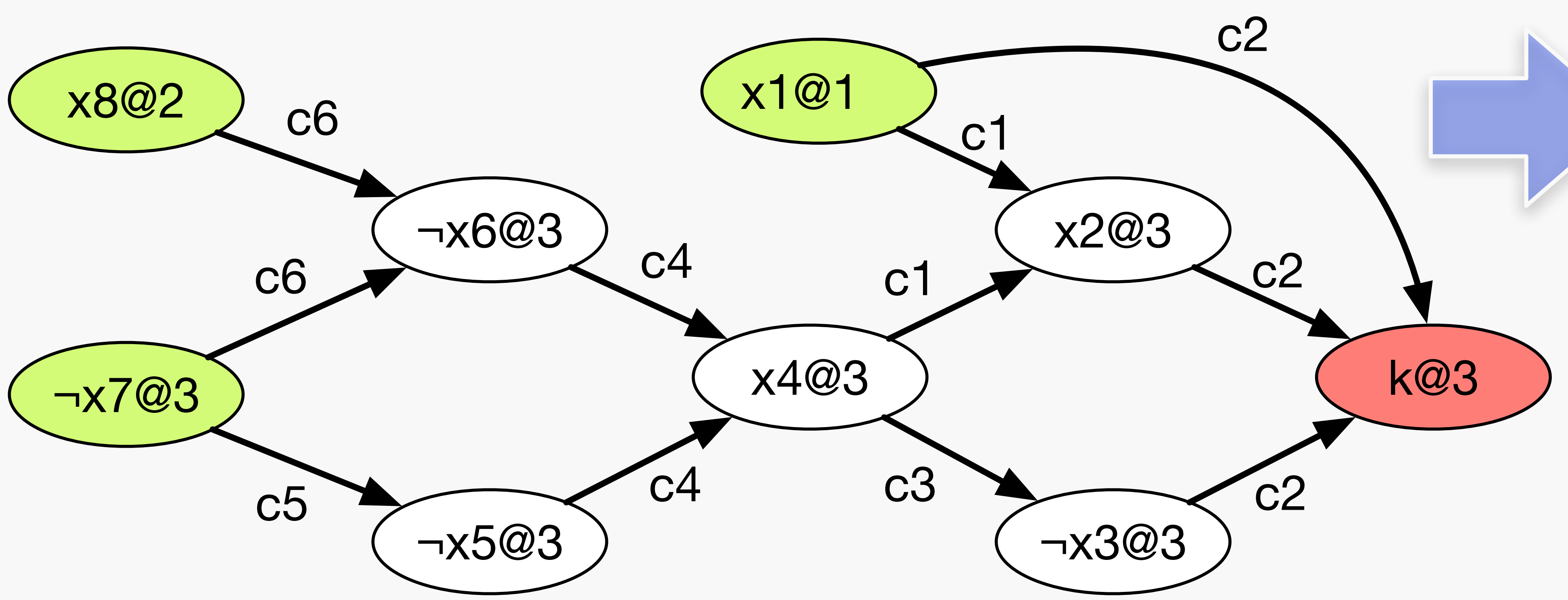
d = 3   c = ¬x1 ∨ x3 ∨ ¬x4   lit = x3

x = x3   ante = c3

Resolve(c, ante, x) =

= Resolve (¬x1 ∨ x3 ∨ ¬x4, ¬x3 ∨ ¬x4, x3)

= ¬x1 ∨ ¬x4




```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteralAtLevel(c,d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  c := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```

c1: ¬x1 ∨ x2 ∨¬x4

c2 : ¬x1 ∨ ¬x2 ∨ x3

c3: ¬x3 ∨¬x4

c4 : x4 ∨ x5 ∨ x6

c5 : ¬x5 ∨ x7

c6: ¬x6 ∨ x7 ∨¬x8

AnalyzeConflict

d = 3 c = ¬x1 ∨ x3 ∨ ¬x4 lit = x3

x = x3 ante = c3

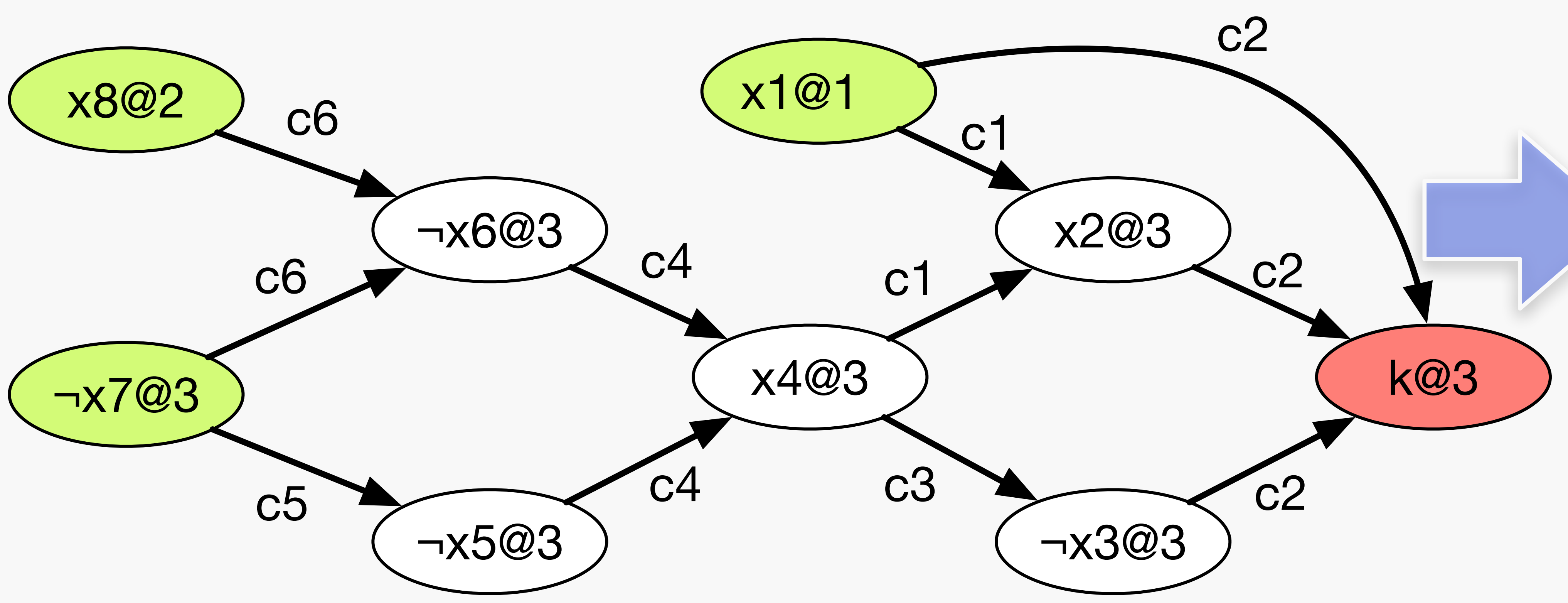
Resolve(c, ante, x) =

= Resolve(¬x1 ∨ x3 ∨ ¬x4, ¬x3 ∨ ¬x4, x3)

= ¬x1 ∨ ¬x4

oneLitAtLevel(c,d) = True

(∵ ¬x4 is at d)




```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteralAtLevel(c,d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  c := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```

c1: ¬x1 ∨ x2 ∨¬x4

c2 : ¬x1 ∨ ¬x2 ∨ x3

c3: ¬x3 ∨¬x4

c4 : x4 ∨ x5 ∨ x6

c5 : ¬x5 ∨ x7

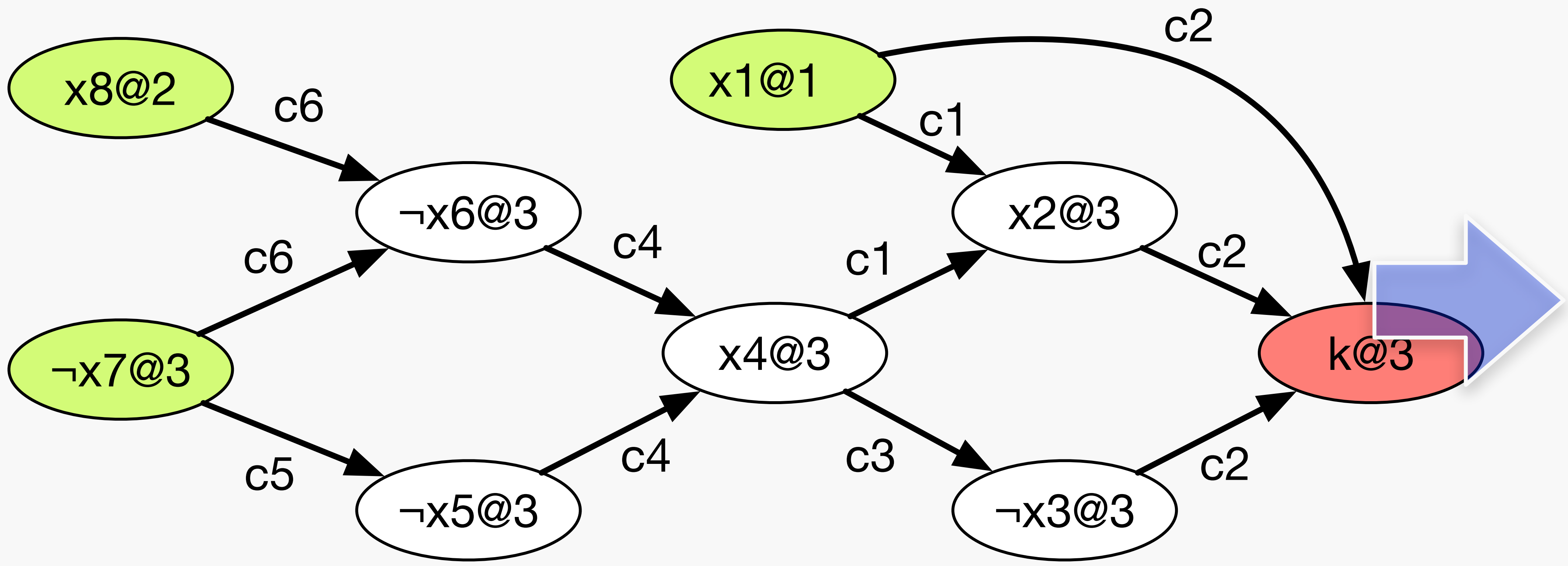
c6: ¬x6 ∨ x7 ∨¬x8

AnalyzeConflict

c = ¬x1 ∨ ¬x4

`assertingLevel(c)` returns the second highest decision level for any literal in `c`, unless `c` is unary (in that case, it returns 0).

`assertingLevel(c) = 1`
(∵ x1@1 and x4@3, 1 is the second highest)




```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteralAtLevel(c,d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  c := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```

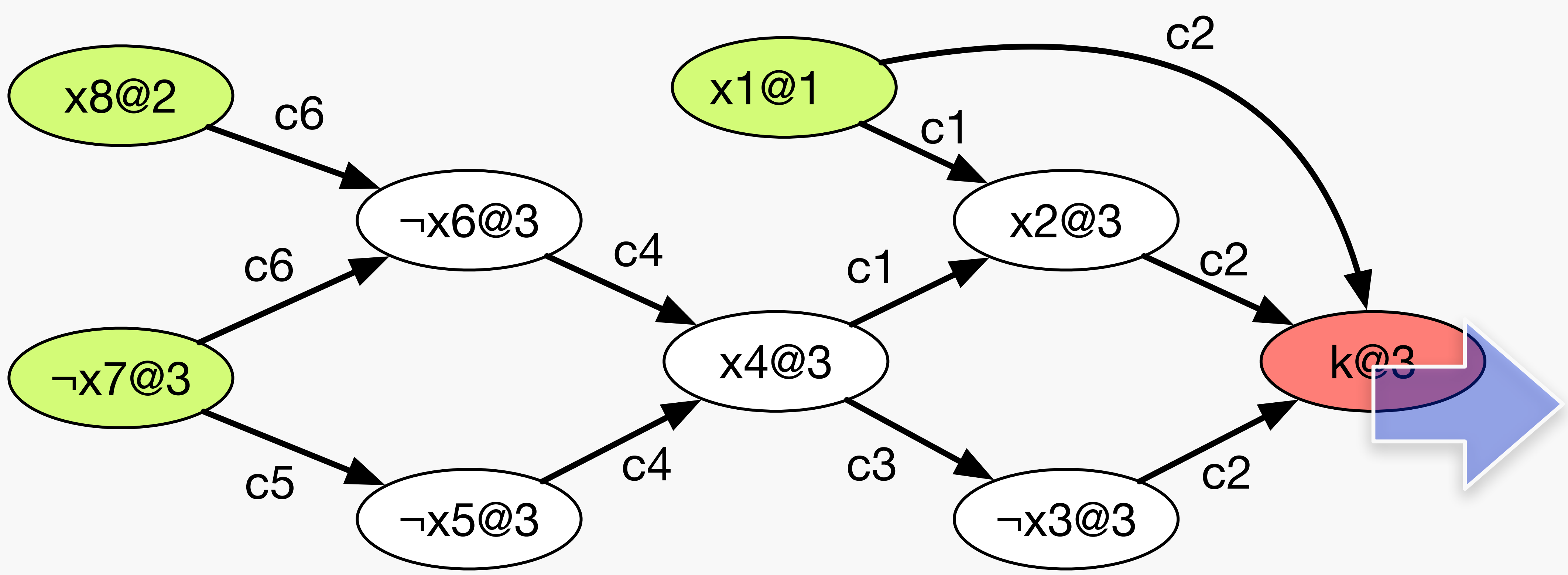
c1: ¬x1 ∨ x2 ∨¬x4
c2 : ¬x1 ∨ ¬x2 ∨ x3
c3: ¬x3 ∨¬x4
c4 : x4 ∨ x5 ∨ x6
c5 : ¬x5 ∨ x7
c6: ¬x6 ∨ x7 ∨¬x8

AnalyzeConflict

Why second highest?

The second one is the highest among the levels of the literals in the conflict clause, *excluding* the current decision level (which is the highest). It backtracks to only as far as needed to make the learned clause useful.

returns <1, ¬x1 ∨ ¬x4>



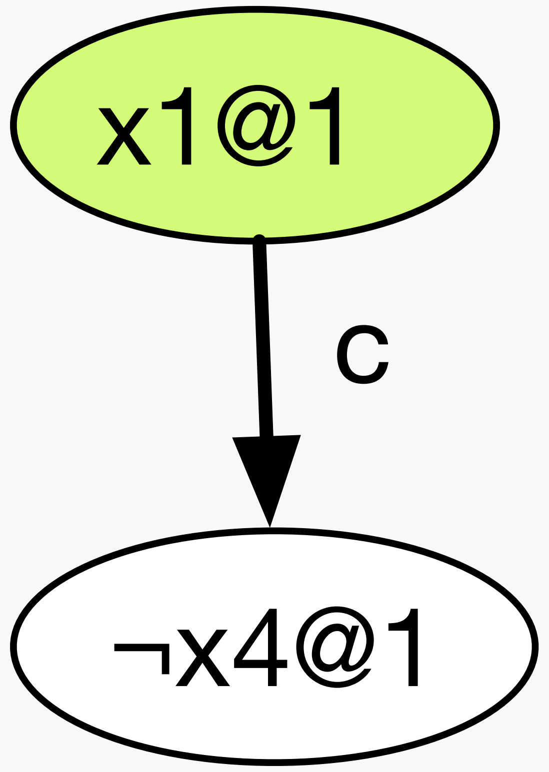

```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteralAtLevel(c,d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  c := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```

c1: ¬x1 ∨ x2 ∨¬x4
c2 : ¬x1 ∨ ¬x2 ∨ x3
c3: ¬x3 ∨¬x4
c4 : x4 ∨ x5 ∨ x6
c5 : ¬x5 ∨ x7
c6: ¬x6 ∨ x7 ∨¬x8

AnalyzeConflict

- By construction, c is always unit at b
  (It has only one literal at the current level d)
- It let the previous assignment immediately fire the learned clause c by BCP and "fix" the reason for the conflict at the current decision level.

c1: ¬x1 ∨ x2 ∨¬x4
c2 : ¬x1 ∨ ¬x2 ∨ x3
c3: ¬x3 ∨¬x4
c4 : x4 ∨ x5 ∨ x6
c5 : ¬x5 ∨ x7
c6: ¬x6 ∨ x7 ∨¬x8
c : ¬x1 ∨ ¬x4




```
function AnalyzeConflict (F,A) =
k@d := GetConflict(F,A)
if d = 0 then return -1
c := Antecedent(k)
repeat
  lit := LastAssignedLiteralAtLevel(c,d)
  x := VarOfLiteral(lit)
  ante := Antecedent(lit)
  c := Resolve(c, ante, x)
until oneLitAtLevel(c,d)
b := assertingLevel(c)
return <b,c>
```


c1: ¬x1 ∨ x2 ∨¬x4
c2 : ¬x1 ∨ ¬x2 ∨ x3
c3: ¬x3 ∨¬x4
c4 : x4 ∨ x5 ∨ x6
c5 : ¬x5 ∨ x7
c6: ¬x6 ∨ x7 ∨¬x8

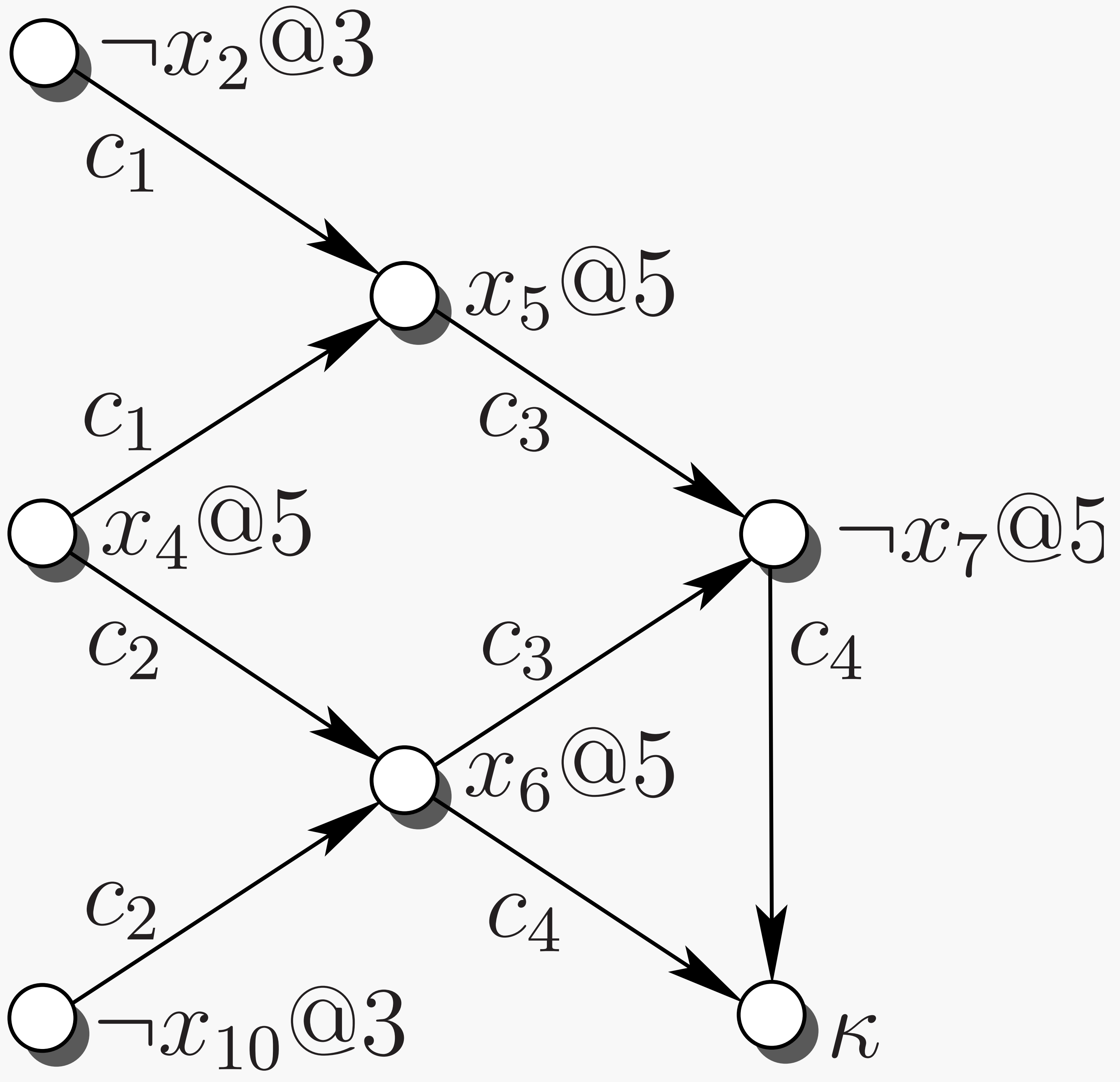
Exercise

- Consider $F = c_1 \wedge c_2 \wedge c_3 \wedge c_4$ where

$$\begin{aligned} c_1 &= (\neg x_4 \vee x_2 \vee x_5) \\ c_2 &= (\neg x_4 \vee x_{10} \vee x_6) \\ c_3 &= (\neg x_5 \vee \neg x_6 \vee \neg x_7) \\ c_4 &= (\neg x_6 \vee x_7) \end{aligned}$$

  What is the conflict clause?

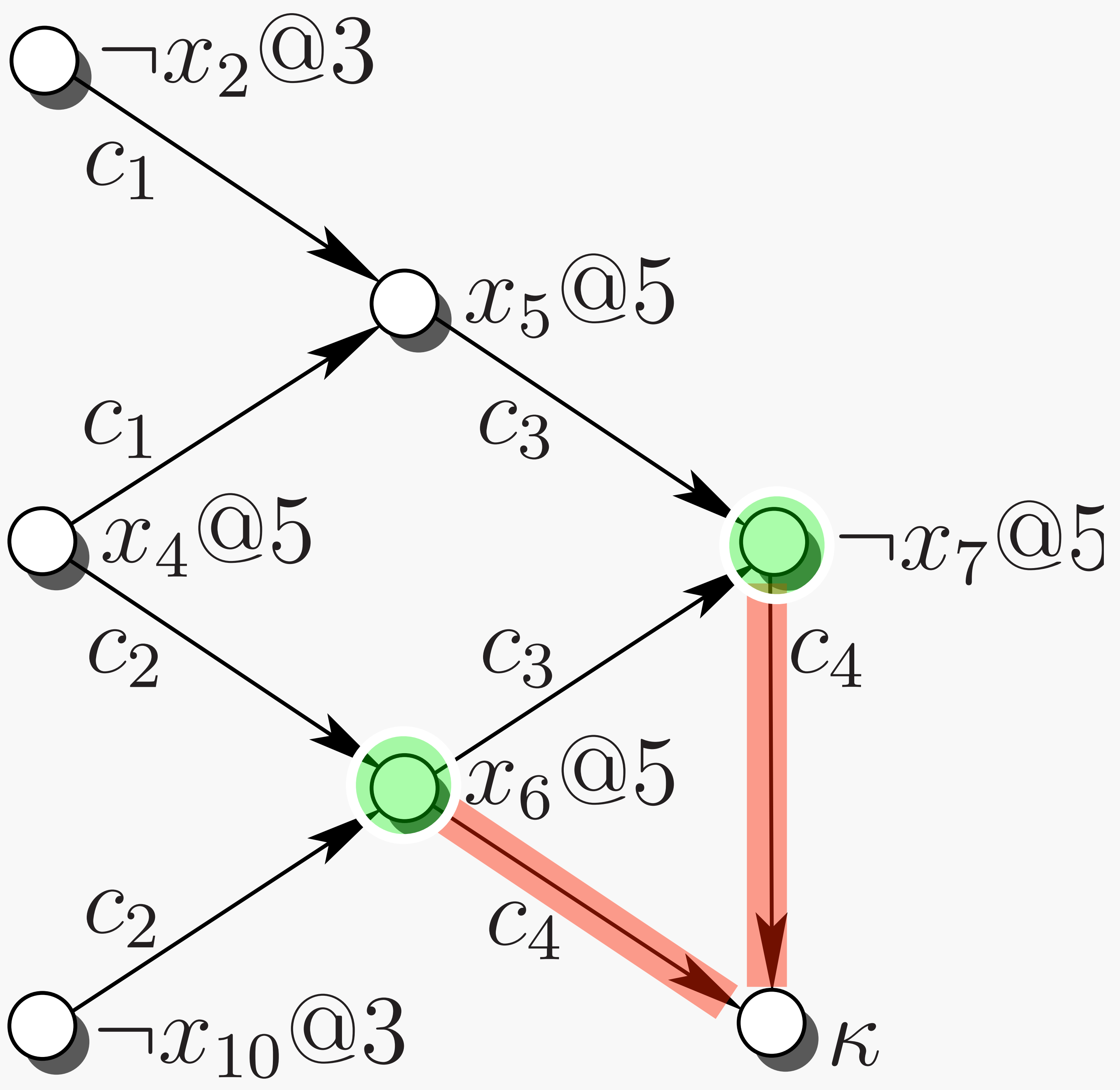
  (suppose x4, x5, x6, x7 are assigned in turn at decision level 3)



Informal, Easier Method for Clause Learning

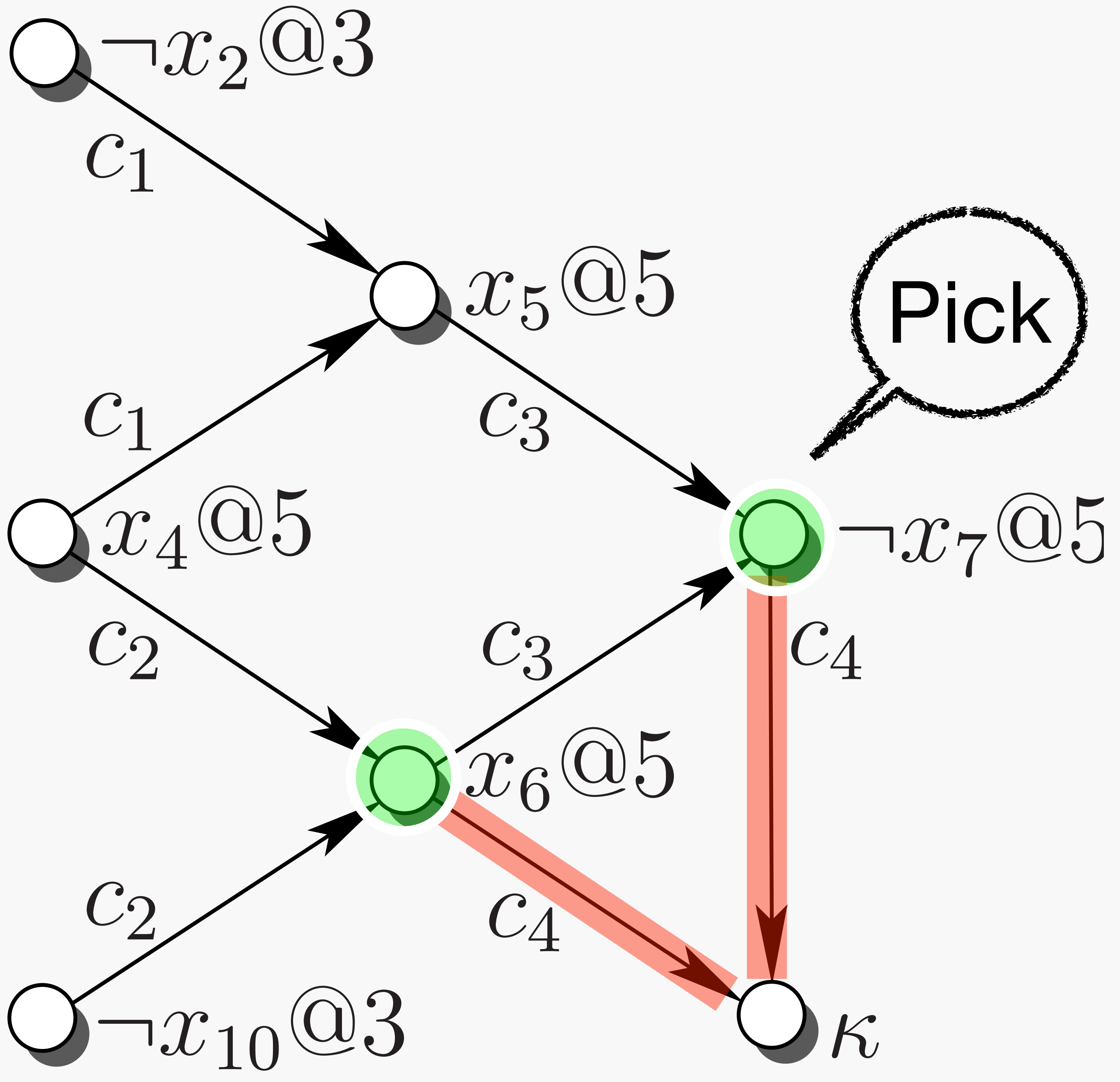
$$c_1 = (\neg x_4 \vee x_2 \vee x_5)$$
$$c_2 = (\neg x_4 \vee x_{10} \vee x_6)$$
$$c_3 = (\neg x_5 \vee \neg x_6 \vee \neg x_7)$$
$$c_4 = (\neg x_6 \vee x_7)$$

- : clauses considered so far
- : reasons for the conflict



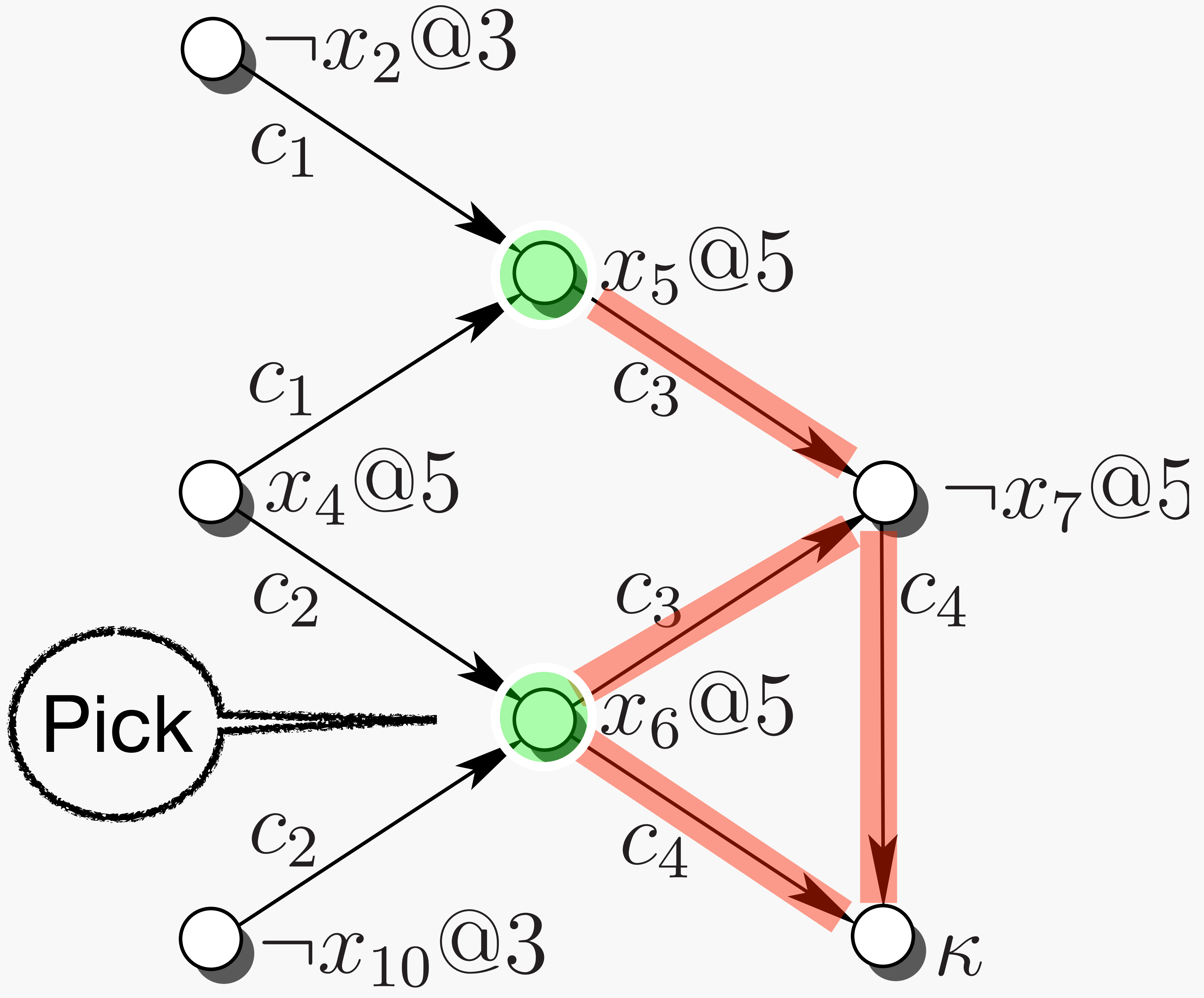
Informal, Easier Method for Clause Learning

$$
\begin{aligned}
c_1 &= (\neg x_4 \lor x_2 \lor x_5) \\
c_2 &= (\neg x_4 \lor x_{10} \lor x_6) \\
c_3 &= (\neg x_5 \lor \neg x_6 \lor \neg x_7) \\
c_4 &= (\neg x_6 \lor x_7)
\end{aligned}
$$



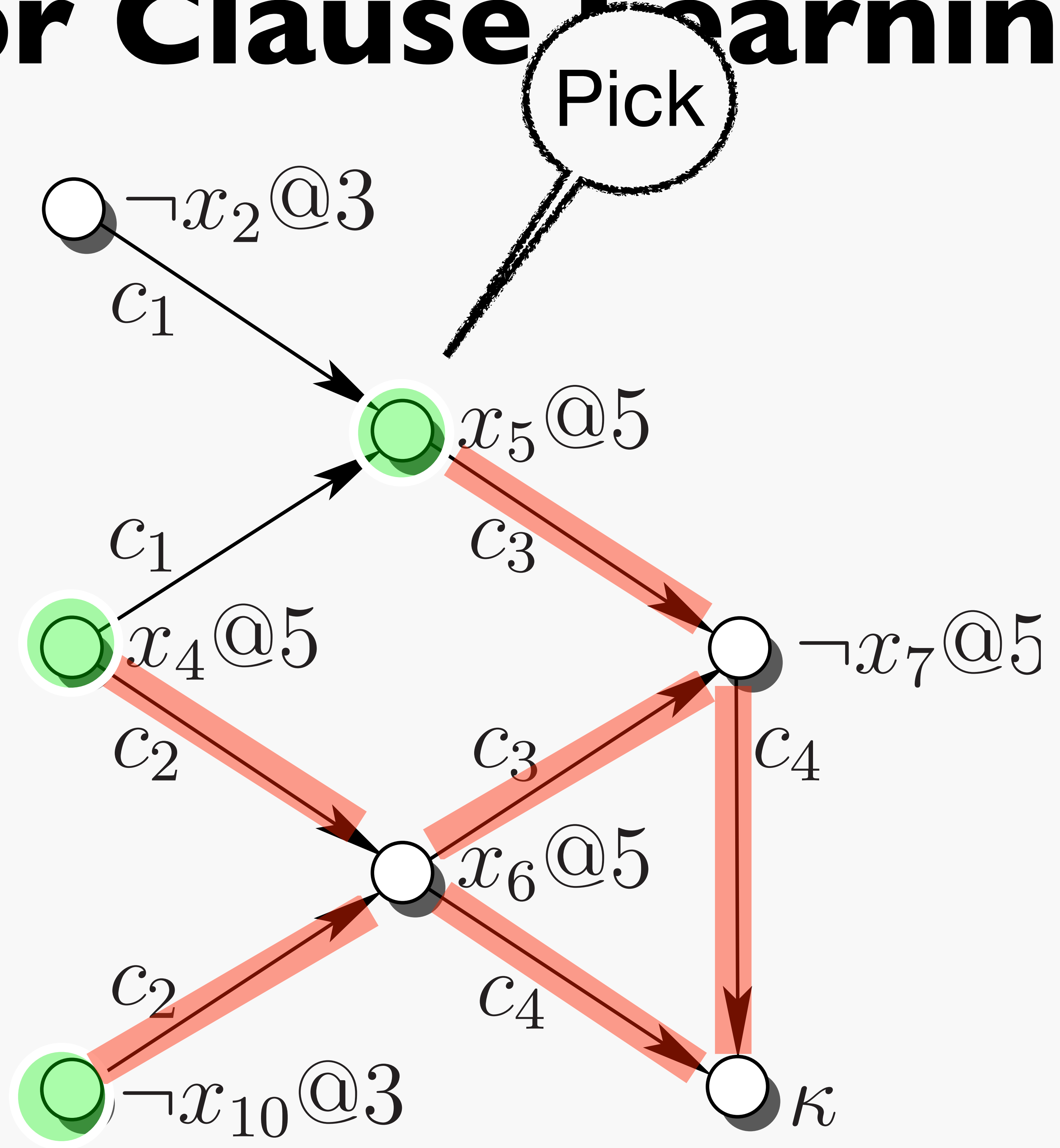
Informal, Easier Method for Clause Learning

$$c_1 = (\neg x_4 \vee x_2 \vee x_5)$$
$$c_2 = (\neg x_4 \vee x_{10} \vee x_6)$$
$$c_3 = (\neg x_5 \vee \neg x_6 \vee \neg x_7)$$
$$c_4 = (\neg x_6 \vee x_7)$$



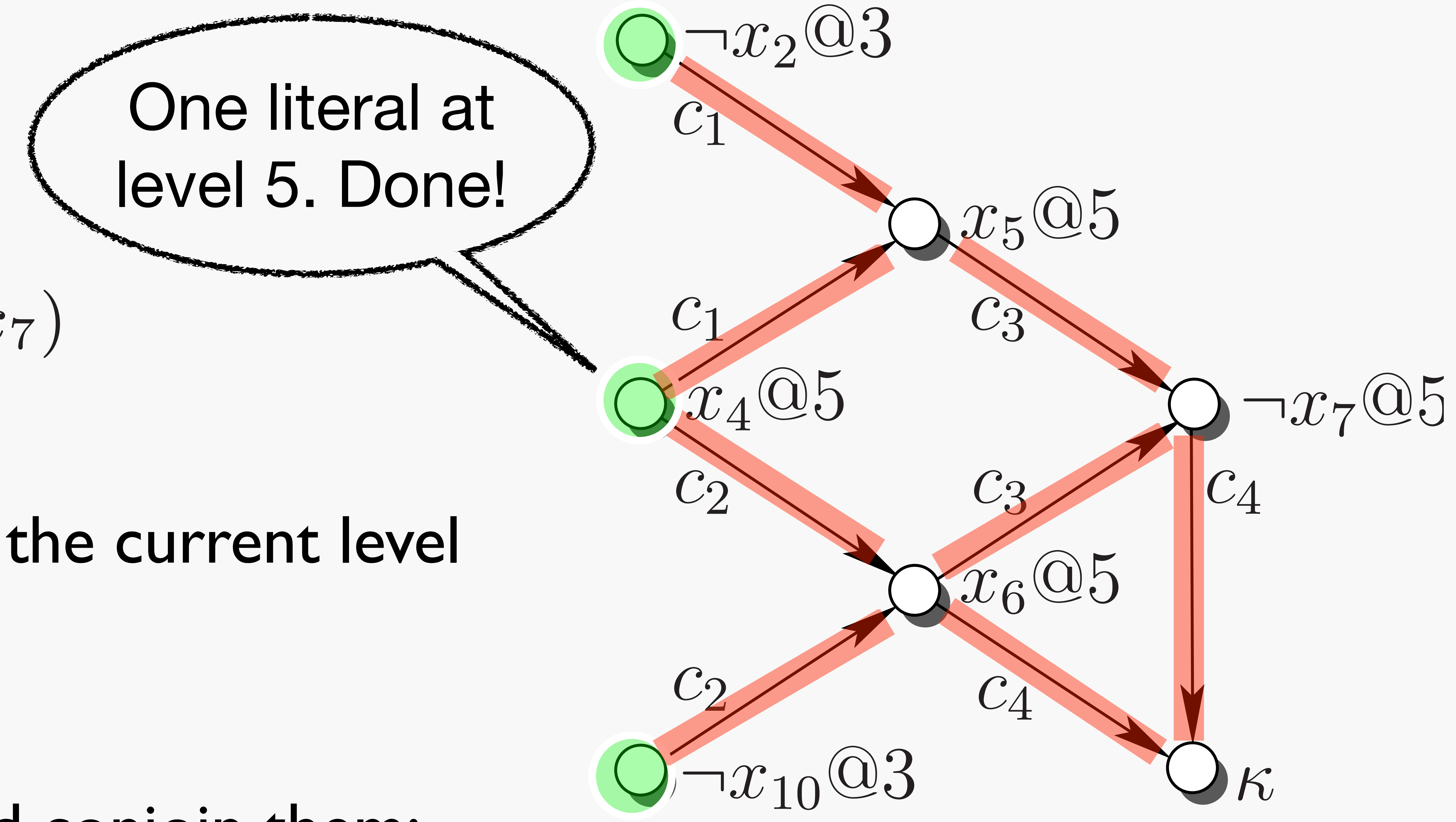
Informal, Easier Method for Clause Learning

$$c_1 = (\neg x_4 \vee x_2 \vee x_5)$$
$$c_2 = (\neg x_4 \vee x_{10} \vee x_6)$$
$$c_3 = (\neg x_5 \vee \neg x_6 \vee \neg x_7)$$
$$c_4 = (\neg x_6 \vee x_7)$$



Informal, Easier Method for Clause Learning

$$
\begin{aligned}
c_1 &= (\neg x_4 \vee x_2 \vee x_5) \\
c_2 &= (\neg x_4 \vee x_{10} \vee x_6) \\
c_3 &= (\neg x_5 \vee \neg x_6 \vee \neg x_7) \\
c_4 &= (\neg x_6 \vee x_7)
\end{aligned}
$$

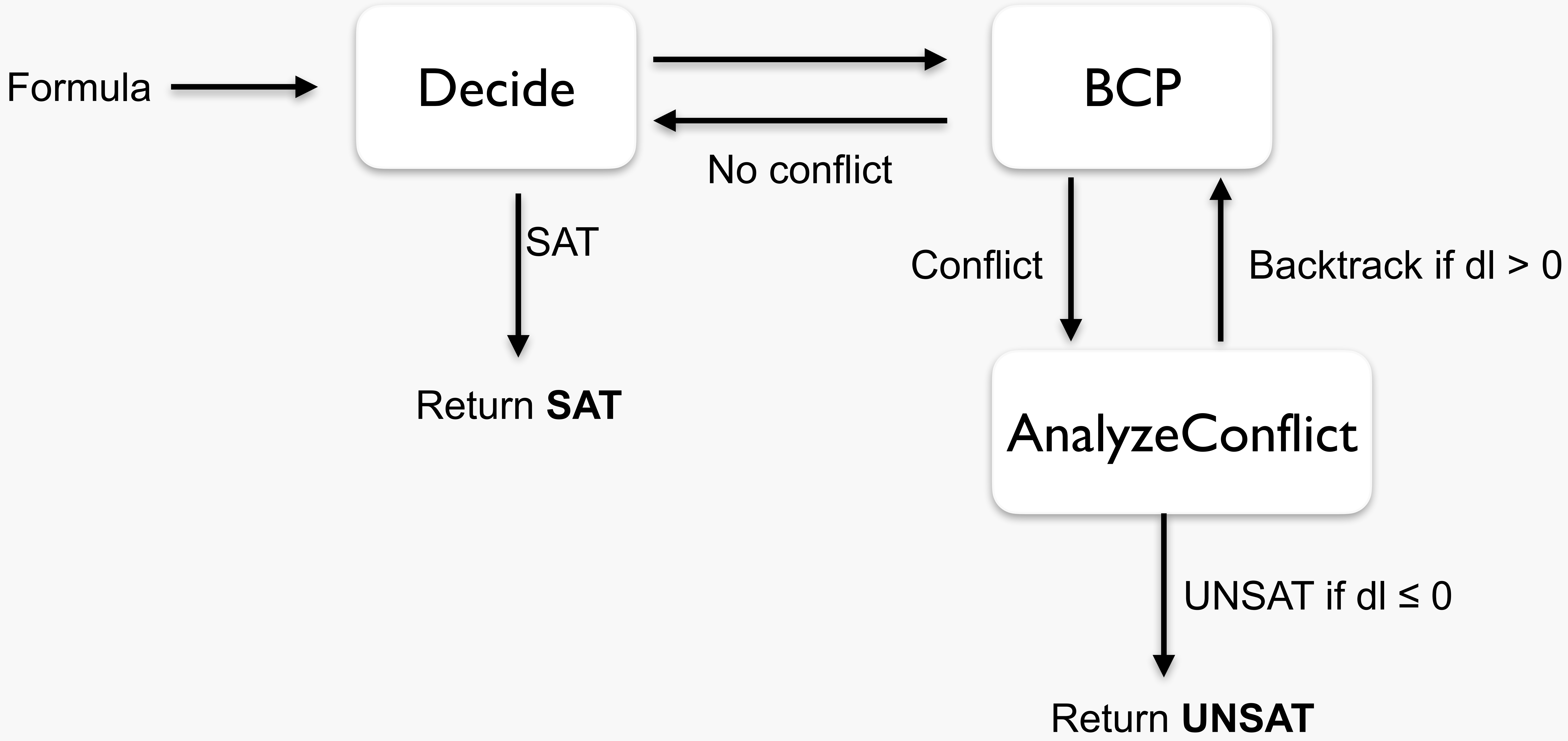


- Quit when one literal at the current level in nodes
- Negate all literals and conjoin them: $x_2 \vee \neg x_4 \vee x_{10}$

Variable Choice Heuristics in CDCL

- Various strategies by which the variables and the value given to them are chosen
- Dynamic Largest Individual Sum (DLIS): At each decision level, choose the unassigned literal that satisfies the largest number of currently unsatisfied clauses.
- Variable State Independent Decaying Sum (VSIDS): Similar to DLIS, but tries to reduce overhead and favor literals involved in conflicts

Overview of CDCL Algorithm



Summary

- CDCL
- Non-chronological backtracking
- Conflict clause learning
- Implication graph
- Unique Implication Point