# SAT Solving

**Noah Fleming**
**University of California, San Diego**

# SAT

→ Canonical NP-Complete language

→ Believed to be intractable in the worst case

# SAT

→ Canonical NP-Complete language

→ Believed to be intractable in the worst case

**Surprisingly…**

Highly efficient algorithms — SAT solvers — have been developed that routinely solve instances of SAT that occur in practice.

# SAT

→ Canonical NP-Complete language

→ Believed to be intractable in the worst case

**Surprisingly…**

Highly efficient algorithms — SAT solvers — have been developed that routinely solve instances of SAT that occur in practice.

- Solve practical SAT instances involving millions of constraints and variables

# SAT

→ Canonical NP-Complete language

→ Believed to be intractable in the worst case

**Surprisingly…**

Highly efficient algorithms — SAT solvers — have been developed that routinely solve instances of SAT that occur in practice.

- Solve practical SAT instances involving millions of constraints and variables
- Routinely used in practice

# SAT

→ Canonical NP-Complete language

→ Believed to be intractable in the worst case

**Surprisingly…**

Highly efficient algorithms — SAT solvers — have been developed that routinely solve instances of SAT that occur in practice.

- Solve practical SAT instances involving millions of constraints and variables
- Routinely used in practice
- Can be more efficient to reduce to SAT and use a SAT solver than to solve directly

# SAT Solvers

Used in a wide variety of practical applications

- Verifying correctness of hardware and software
- Planning (e.g., air-traffic control)
- Bioinformatics
- Verifying conjectures in mathematics and physics
- Security
- Program synthesis

# This Seminar

**We will explore…**

- What are SAT solvers? How do they work?

# This Seminar

**We will explore…**

- What are SAT solvers? How do they work?
- How can we analyze SAT solvers?

  → Proof complexity as a tool for algorithm analysis

# This Seminar

**We will explore…**

- What are SAT solvers? How do they work?
- How can we analyze SAT solvers?

  → Proof complexity as a tool for algorithm analysis
- Why do SAT solvers work so well?

# This Seminar

**We will explore…**

- What are SAT solvers? How do they work?
- How can we analyze SAT solvers?

  → Proof complexity as a tool for algorithm analysis
- Why do SAT solvers work so well?
- Beyond SAT (pseudo-boolean solvers, integer programming solvers)

# This Seminar

**We will explore…**

- What are SAT solvers? How do they work?
- How can we analyze SAT solvers?

  → Proof complexity as a tool for algorithm analysis
- Why do SAT solvers work so well?
- Beyond SAT (pseudo-boolean solvers, integer programming solvers)
- … and more!

# Outline for Today

1. Propositional Logic Syntax & SAT
2. DPLL
3. Analyzing DPLL by tree Resolution
4. Overview of CDCL
5. Unit Propagation
6. Clause Learning
7. Restarting

# Syntax of Propositional Logic

**Variables:** $x_1, \ldots, x_n$ taking value in $\{0,1\}$

# Syntax of Propositional Logic

**Variables:** $x_1, \ldots, x_n$ taking value in $\{0,1\}$

**Literals:** $\ell = \; x_i$ or $\bar{x}_i$

# Syntax of Propositional Logic

**Variables:** $x_1, \ldots, x_n$ taking value in $\{0,1\}$

**Literals:** $\ell = \; x_i$ or $\bar{x}_i$

**Connectives:** $\wedge$ (AND), $\vee$ (OR)

# Syntax of Propositional Logic

**Variables:** $x_1, \ldots, x_n$ taking value in $\{0,1\}$

**Literals:** $\ell = x_i$ or $\bar{x}_i$

**Connectives:** $\wedge$ (AND), $\vee$ (OR)

**Propositional Logic Formula:** built up from literals and connectives

# Syntax of Propositional Logic

**Variables:** $x_1, \ldots, x_n$ taking value in $\{0,1\}$

**Literals:** $\ell = x_i$ or $\bar{x}_i$

**Connectives:** $\wedge$ (AND), $\vee$ (OR)

**Propositional Logic Formula:** built up from literals and connectives

e.g. $F = x_1 \wedge (x_3 \vee (\bar{x}_2 \wedge \bar{x}_3)) \wedge x_2$

# Syntax of Propositional Logic

**Variables:** $x_1, \ldots, x_n$ taking value in $\{0,1\}$

**Literals:** $\ell = \ x_i$ or $\bar{x}_i$

**Connectives:** $\wedge$ (AND), $\vee$ (OR)

**Propositional Logic Formula:** built up from literals and connectives

e.g. $F = x_1 \wedge (x_3 \vee (\bar{x}_2 \wedge \bar{x}_3)) \wedge x_2$

**Satisfiable:** If there is $x \in \{0,1\}^n$ such that $F(x) = 1$

# Syntax of Propositional Logic

**Variables:** $x_1, \ldots, x_n$ taking value in $\{0,1\}$

**Literals:** $\ell = x_i$ or $\bar{x}_i$

**Connectives:** $\wedge$ (AND), $\vee$ (OR)

**Propositional Logic Formula:** built up from literals and connectives

e.g. $F = x_1 \wedge (x_3 \vee (\bar{x}_2 \wedge \bar{x}_3)) \wedge x_2$ Satisfied by $x = (1,1,1)$

**Satisfiable:** If there is $x \in \{0,1\}^n$ such that $F(x) = 1$

# Syntax of Propositional Logic

**Variables:** $x_1, \ldots, x_n$ taking value in $\{0,1\}$

**Literals:** $\ell = \; x_i$ or $\bar{x}_i$

**Connectives:** $\wedge$ (AND), $\vee$ (OR)

**Propositional Logic Formula:** built up from literals and connectives

e.g. $F = x_1 \wedge (x_3 \vee (\bar{x}_2 \wedge \bar{x}_3)) \wedge x_2$ Satisfied by $x = (1,1,1)$

**Satisfiable:** If there is $x \in \{0,1\}^n$ such that $F(x) = 1$

# Syntax of Propositional Logic

**Variables:** $x_1, \ldots, x_n$ taking value in $\{0,1\}$

**Literals:** $\ell = x_i$ or $\bar{x}_i$

**Connectives:** $\wedge$ (AND), $\vee$ (OR)

**Propositional Logic Formula:** built up from literals and connectives

e.g. $F = x_1 \wedge (x_3 \vee (\bar{x}_2 \wedge \bar{x}_3)) \wedge x_2$ Satisfied by $x = (1,1,1)$

**Satisfiable:** If there is $x \in \{0,1\}^n$ such that $F(x) = 1$

**Unsatisfiable:** Otherwise

# Syntax of Propositional Logic

**Clause:** Disjunction of literals $C = \ell_1 \vee \ldots \vee \ell_k$

e.g. $(x_1 \vee \bar{x}_2 \vee x_4)$

# Syntax of Propositional Logic

**Clause:** Disjunction of literals $C = \ell_1 \vee \ldots \vee \ell_k$

e.g. $(x_1 \vee \bar{x}_2 \vee x_4)$

**CNF Formula:** Conjunction of clauses $F = C_1 \wedge \ldots \wedge C_m$

e.g. $(x_1 \vee \bar{x}_2 \vee x_4) \wedge (x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_4) \wedge (\bar{x}_4)$

# The SAT Problem

**SAT:** Given a CNF formula $F$, does there exist $x \in \{0,1\}^n$ such that $F(x) = 1$

# The SAT Problem

**SAT:** Given a CNF formula $F$, does there exist $x \in \{0,1\}^n$ such that $F(x) = 1$

- Canonical NP-complete problem
- Nonetheless, huge success in designing efficient algorithms for solving SAT in **practice**

# The SAT Problem

**SAT:** Given a CNF formula $F$, does there exist $x \in \{0,1\}^n$ such that $F(x) = 1$

- Canonical NP-complete problem
- Nonetheless, huge success in designing efficient algorithms for solving SAT in **practice**

e.g. $(x_1 \vee \bar{x}_2 \vee x_4) \wedge (x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_4) \wedge (\bar{x}_4)$

Satisfiable?

# The SAT Problem

**SAT:** Given a CNF formula $F$, does there exist $x \in \{0,1\}^n$ such that $F(x) = 1$

- Canonical NP-complete problem
- Nonetheless, huge success in designing efficient algorithms for solving SAT in **practice**

e.g. $(x_1 \vee \bar{x}_2 \vee x_4) \wedge (x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_4) \wedge (\bar{x}_4)$

Satisfiable? Yes! $x = (0,0,0,0)$

# The SAT Problem

**SAT:** Given a CNF formula $F$, does there exist $x \in \{0,1\}^n$ such that $F(x) = 1$

- Canonical NP-complete problem
- Nonetheless, huge success in designing efficient algorithms for solving SAT in **practice**

e.g. $(x_1 \vee \bar{x}_2 \vee x_4) \wedge (x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_4) \wedge (\bar{x}_4)$

Satisfiable? Yes! $x = (0,0,0,0)$

$Q$**:** How would **you** determine whether a formula is satisfiable?

# DPLL — The Heart of SAT Solvers

**DPLL:** A brute-force approach to solving SAT

# DPLL — The Heart of SAT Solvers

**DPLL:** A brute-force approach to solving SAT

**Input:** A CNF formula $F$

**Output:** Whether $F$ is satisfiable

# DPLL — The Heart of SAT Solvers

**DPLL:** A brute-force approach to solving SAT

**Input:** A CNF formula $F$

**Output:** Whether $F$ is satisfiable

**DPLL**$(F)$**:**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \upharpoonright x_i = 0)$
3. **DPLL**$(F \upharpoonright x_i = 1)$

# DPLL — The Heart of SAT Solvers

**DPLL:** A brute-force approach to solving SAT

$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2)$

**Input:** A CNF formula $F$

**Output:** Whether $F$ is satisfiable

**DPLL$(F)$:**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \upharpoonright x_i = 0)$
3. **DPLL**$(F \upharpoonright x_i = 1)$

# DPLL — The Heart of SAT Solvers

**DPLL:** A brute-force approach to solving SAT

$$F = (x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2)$$

$x_2$

**Input:** A CNF formula $F$

**Output:** Whether $F$ is satisfiable

**DPLL$(F)$:**

**If** $F = 1$, output SAT

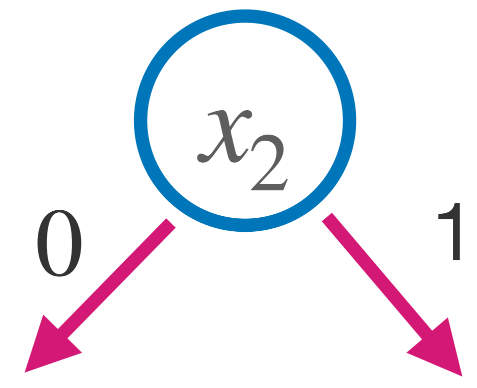
**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \upharpoonright x_i = 0)$
3. **DPLL**$(F \upharpoonright x_i = 1)$

# DPLL — The Heart of SAT Solvers

**DPLL:** A brute-force approach to solving SAT

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2)$$




**Input:** A CNF formula $F$

**Output:** Whether $F$ is satisfiable

**DPLL**$(F)$**:**

**If** $F = 1$, output SAT

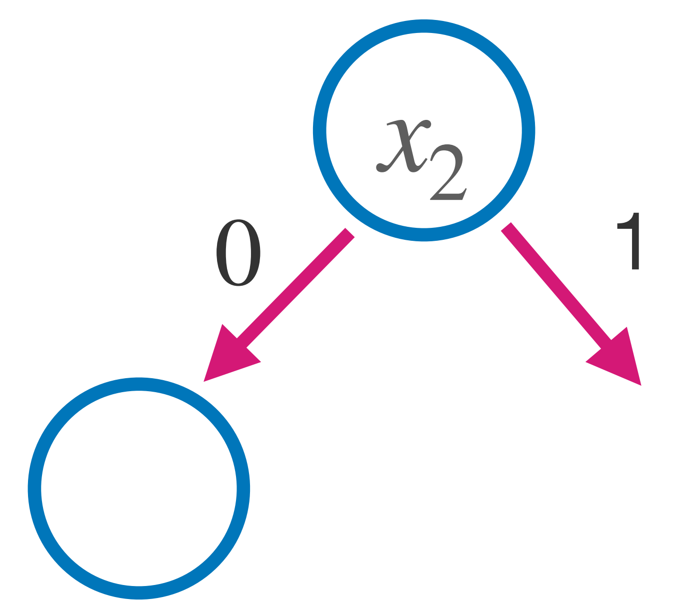
**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \restriction x_i = 0)$
3. **DPLL**$(F \restriction x_i = 1)$

# DPLL — The Heart of SAT Solvers

**DPLL:** A brute-force approach to solving SAT

$$F = (x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2)$$




**Input:** A CNF formula $F$

**Output:** Whether $F$ is satisfiable

**DPLL**$(F)$**:**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \restriction x_i = 0)$
3. **DPLL**$(F \restriction x_i = 1)$

# DPLL — The Heart of SAT Solvers

**DPLL:** A brute-force approach to solving SAT

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2)$$

**Input:** A CNF formula $F$

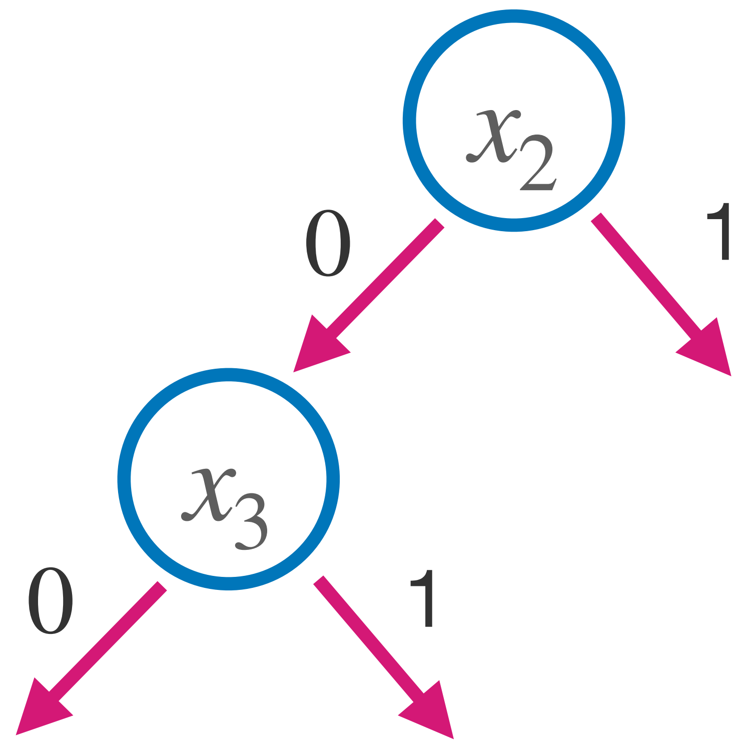
**Output:** Whether $F$ is satisfiable

**DPLL**$(F)$**:**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \restriction x_i = 0)$
3. **DPLL**$(F \restriction x_i = 1)$



# DPLL — The Heart of SAT Solvers

**DPLL:** A brute-force approach to solving SAT

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2)$$

**Input:** A CNF formula $F$

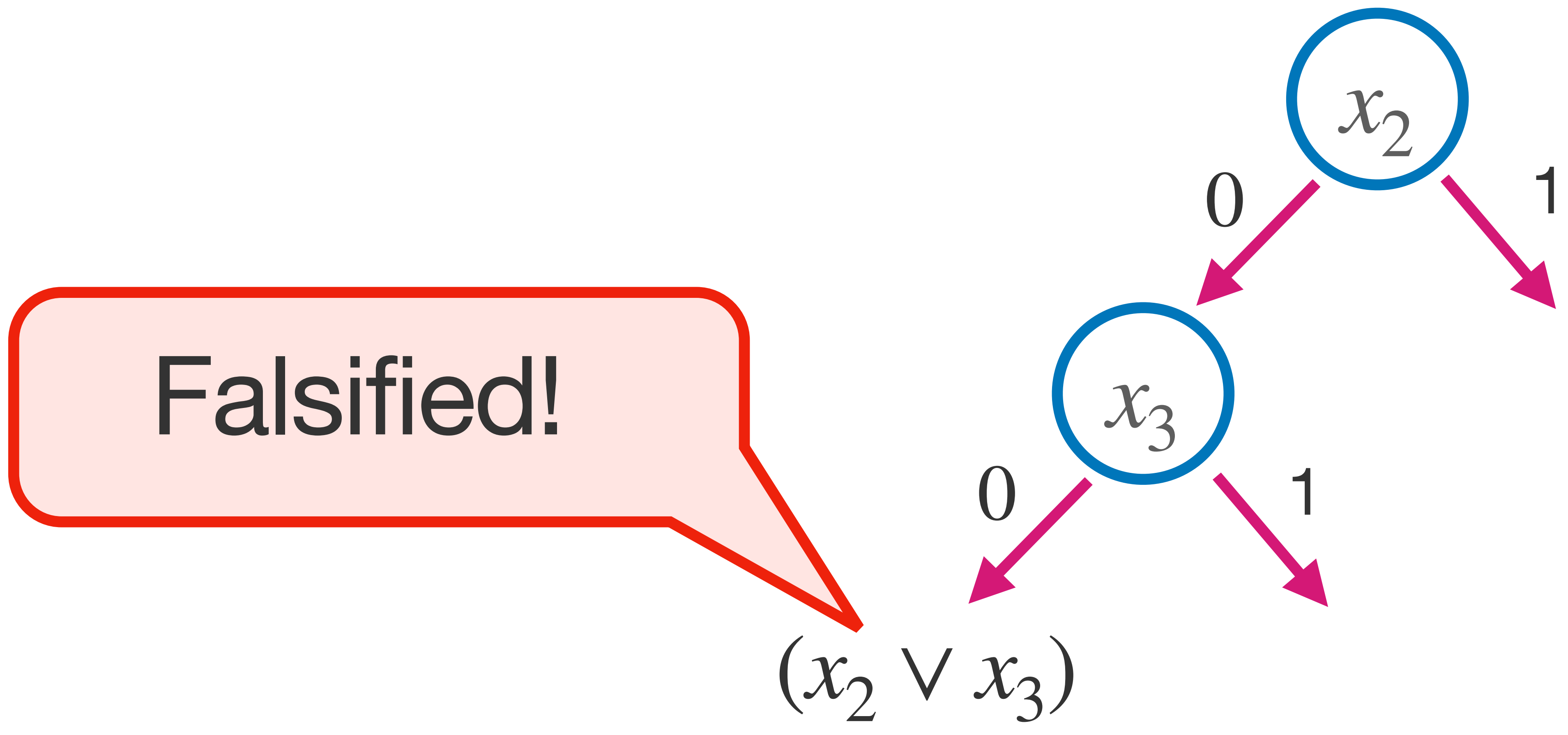
**Output:** Whether $F$ is satisfiable

**DPLL**$(F)$**:**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \restriction x_i = 0)$
3. **DPLL**$(F \restriction x_i = 1)$



# DPLL — The Heart of SAT Solvers

**DPLL:** A brute-force approach to solving SAT

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2)$$

**Input:** A CNF formula $F$

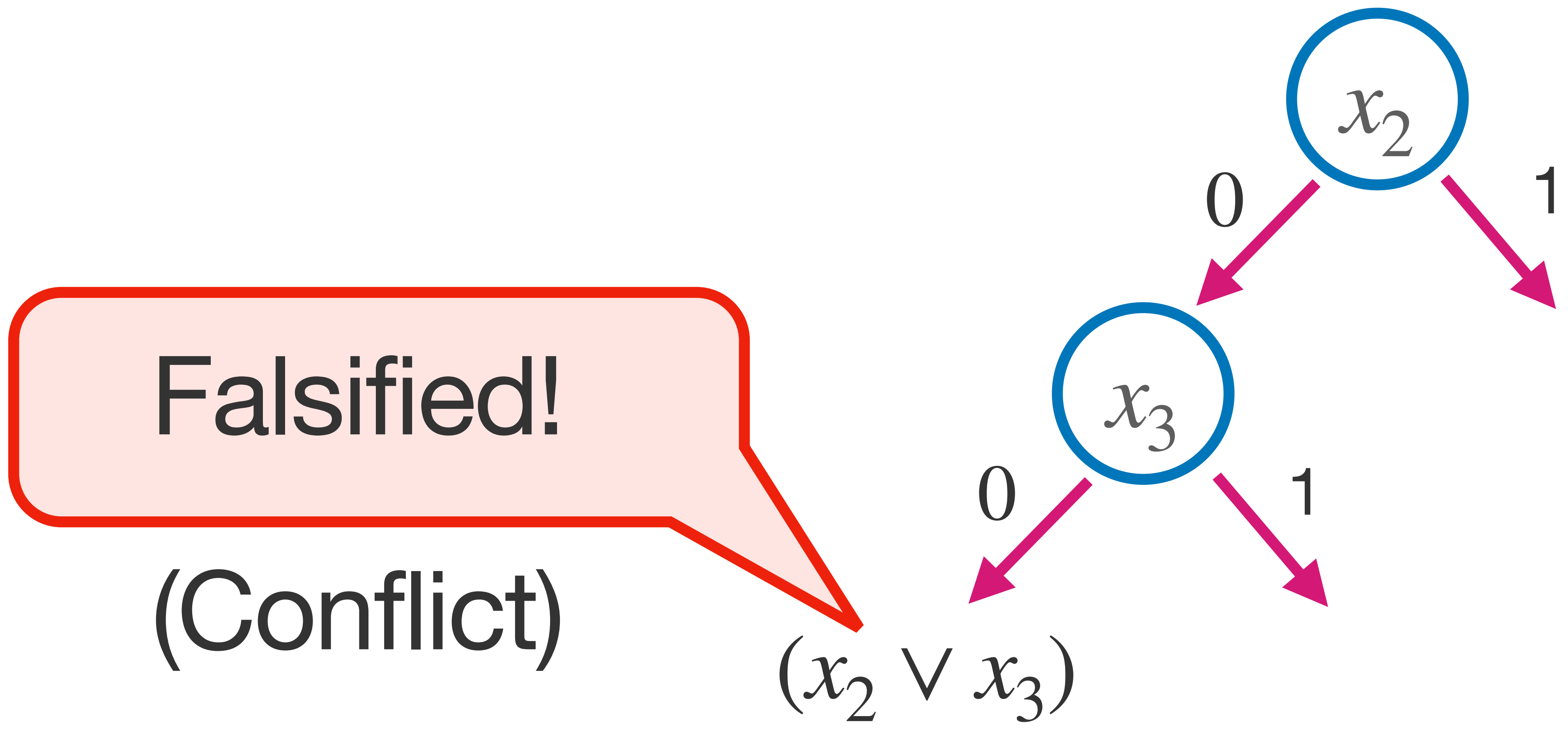
**Output:** Whether $F$ is satisfiable

**DPLL$(F)$:**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \restriction x_i = 0)$
3. **DPLL**$(F \restriction x_i = 1)$



# DPLL — The Heart of SAT Solvers

**DPLL:** A brute-force approach to solving SAT

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2)$$

**Input:** A CNF formula $F$

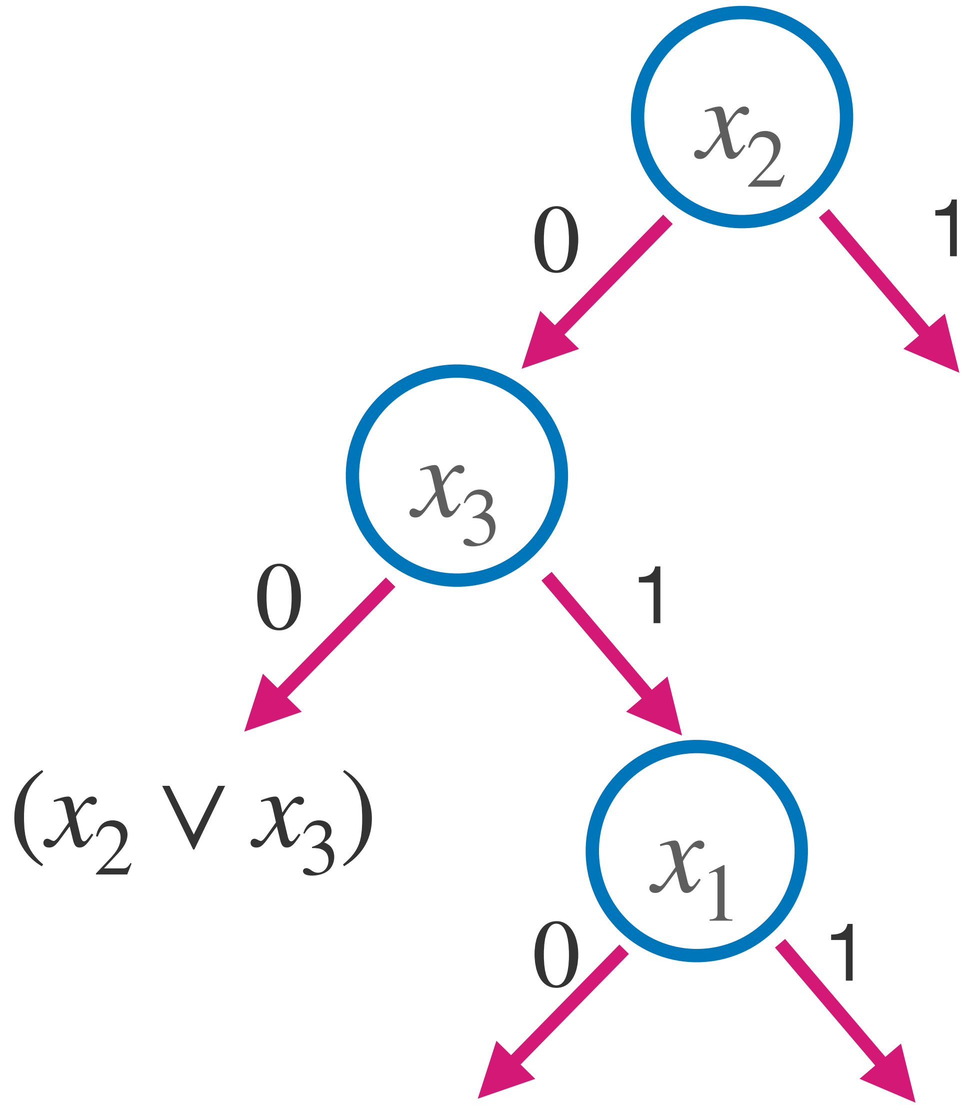
**Output:** Whether $F$ is satisfiable

**DPLL($F$):**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL($F \upharpoonright x_i = 0$)**
3. **DPLL($F \upharpoonright x_i = 1$)**



# DPLL — The Heart of SAT Solvers

**DPLL:** A brute-force approach to solving SAT

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2)$$

**Input:** A CNF formula $F$

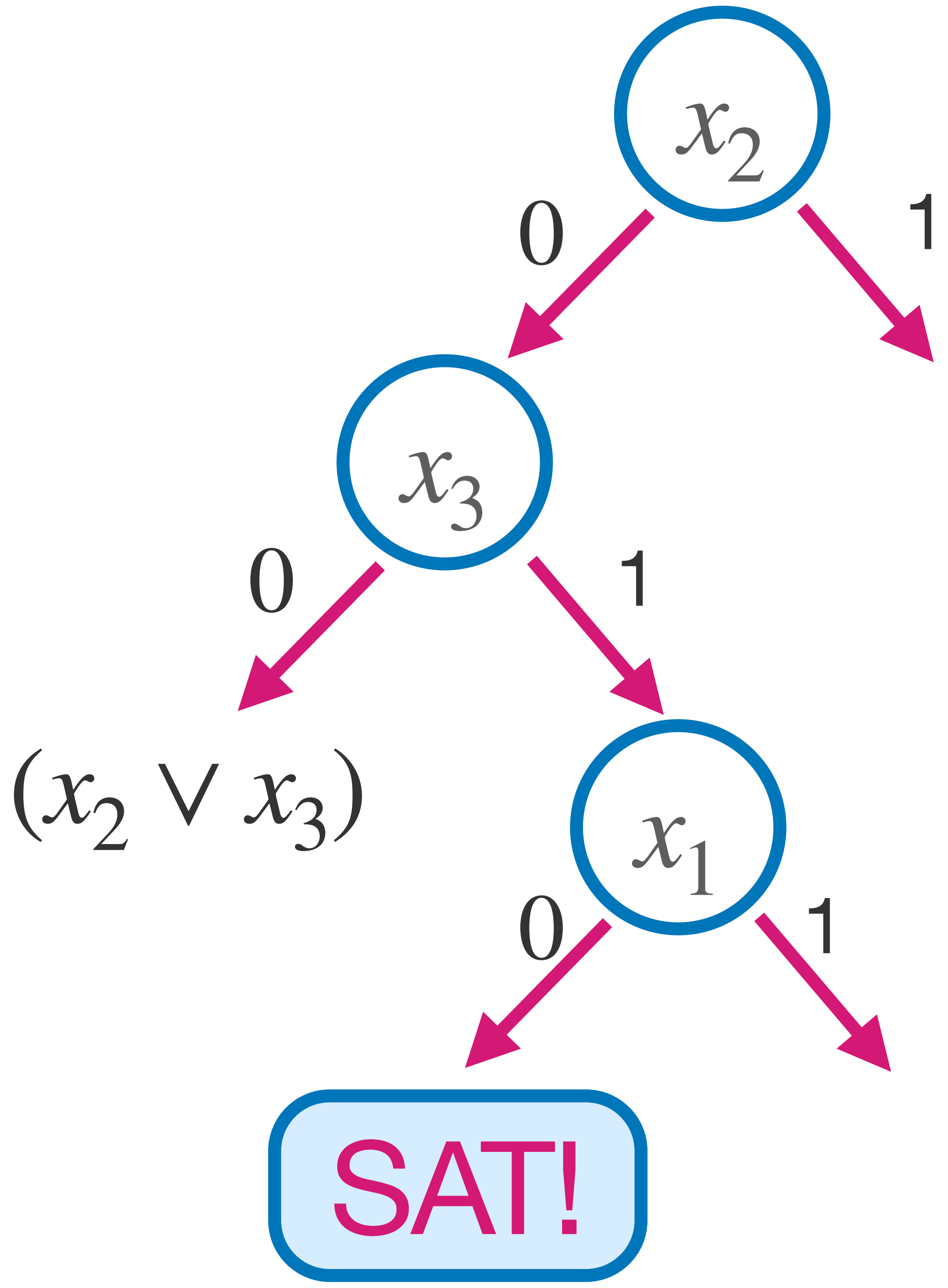
**Output:** Whether $F$ is satisfiable

**DPLL$(F)$:**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \restriction x_i = 0)$
3. **DPLL**$(F \restriction x_i = 1)$



# Analyzing DPLL

**DPLL:** A brute-force approach to solving SAT

→ Modern SAT Solvers build on DPLL

# Analyzing DPLL

**DPLL:** A brute-force approach to solving SAT

$\rightarrow$ Modern SAT Solvers build on DPLL

$Q.$ Can we show that DPLL alone is sufficient to solve SAT?

# Analyzing DPLL

**DPLL:** A brute-force approach to solving SAT

$\rightarrow$ Modern SAT Solvers build on DPLL

$Q$. Can we show that DPLL alone is sufficient to solve SAT?

**Proof Complexity** provides a convenient tool for algorithm analysis

$\rightarrow$ Studies the size of **proofs** of unsatisfiability of CNF formulas

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

$$(x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

$$(x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \ \wedge (x_1 \vee \neg x_3)$$

"Set of clauses"

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

$$(x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

"Set of clauses"

Derive new clauses from old using:

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

Derive new clauses from old using:

→ **Resolution rule:**

$$\frac{C_1 \vee x, \quad C_2 \vee \neg x}{C_1 \vee C_2}$$

$$(x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \ \wedge (x_1 \vee \neg x_3)$$

"Set of clauses"

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

$$(x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

"Set of clauses"

Derive new clauses from old using:

→ **Resolution rule:**

$$\frac{C_1 \vee x, \quad C_2 \vee \neg x}{C_1 \vee C_2}$$

**Goal:** derive empty clause $\Lambda$

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

$$(x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

"Set of clauses"

Derive new clauses from old using:

→ **Resolution rule:**

$$\frac{C_1 \vee x, \quad C_2 \vee \neg x}{C_1 \vee C_2}$$

**Goal:** derive empty clause $\Lambda$

Resolution rule is **sound**

$\Longrightarrow$ Derivation of $\Lambda$ certifies unsatisfiability

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

$$(x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

Derive new clauses from old using:

$\rightarrow$ **Resolution rule:**

$$\frac{C_1 \vee x, \quad C_2 \vee \neg x}{C_1 \vee C_2}$$

**Goal:** derive empty clause $\Lambda$

Resolution rule is **sound**

$\Longrightarrow$ Derivation of $\Lambda$ certifies unsatisfiability

$(x_1 \vee \neg x_3) \quad (\neg x_1 \vee \neg x_3)$

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

$$(x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

Derive new clauses from old using:

$\rightarrow$ **Resolution rule:**

$$\frac{C_1 \vee x, \quad C_2 \vee \neg x}{C_1 \vee C_2}$$

**Goal:** derive empty clause $\Lambda$

$$\begin{tikzcd}
& \boxed{\neg x_3} & \\
(x_1 \vee \neg x_3) \arrow[ur] & & (\neg x_1 \vee \neg x_3) \arrow[ul]
\end{tikzcd}$$

Resolution rule is **sound**

$\Longrightarrow$ Derivation of $\Lambda$ certifies unsatisfiability

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

$$(x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

Derive new clauses from old using:

→ **Resolution rule:**

$$\frac{C_1 \vee x, \quad C_2 \vee \neg x}{C_1 \vee C_2}$$

**Goal:** derive empty clause $\Lambda$

$(x_2 \vee x_3)$ $\quad$ $\boxed{\neg x_3}$

$(x_1 \vee \neg x_3)$ $\quad$ $(\neg x_1 \vee \neg x_3)$

Resolution rule is **sound**

$\Longrightarrow$ Derivation of $\Lambda$ certifies unsatisfiability

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

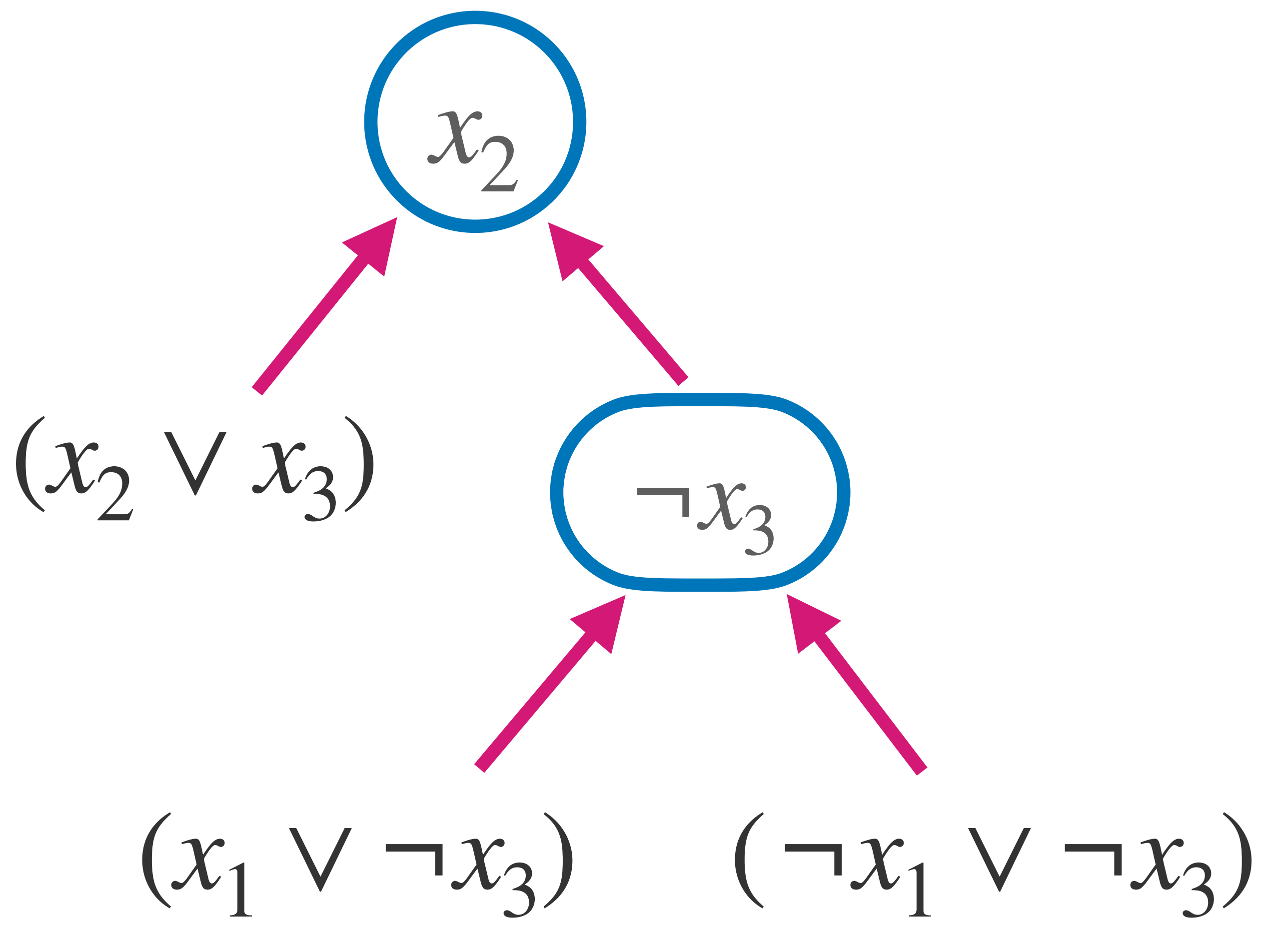
$$(x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

Derive new clauses from old using:

$\rightarrow$ **Resolution rule:**

$$\frac{C_1 \vee x, \quad C_2 \vee \neg x}{C_1 \vee C_2}$$

**Goal:** derive empty clause $\Lambda$




Resolution rule is **sound**

$\Longrightarrow$ Derivation of $\Lambda$ certifies unsatisfiability

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

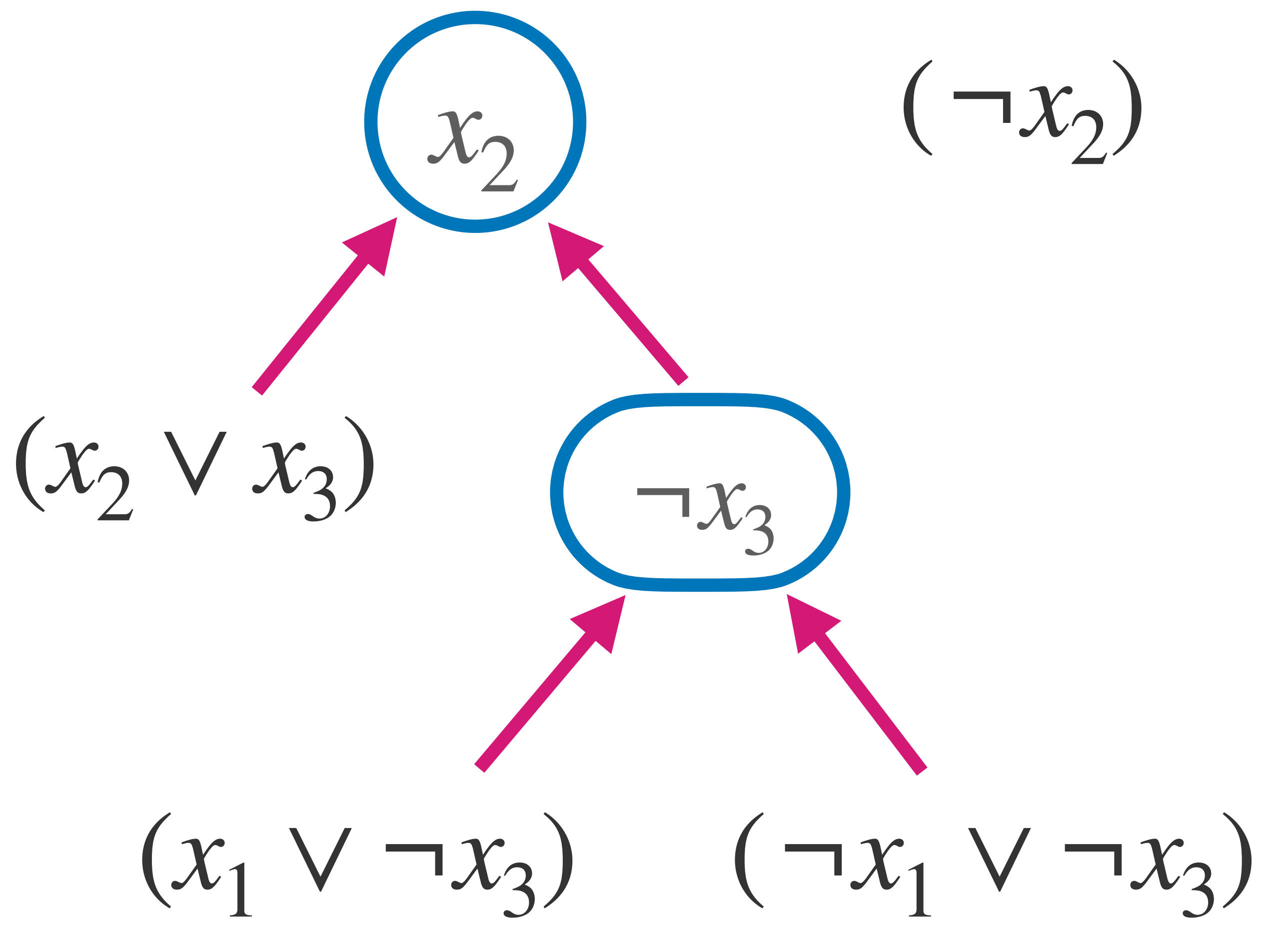
$$(x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2) \land (x_1 \lor \neg x_3)$$

Derive new clauses from old using:

→ **Resolution rule:**

$$\frac{C_1 \lor x, \quad C_2 \lor \neg x}{C_1 \lor C_2}$$

**Goal:** derive empty clause $\Lambda$




Resolution rule is **sound**

$\Longrightarrow$ Derivation of $\Lambda$ certifies unsatisfiability

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

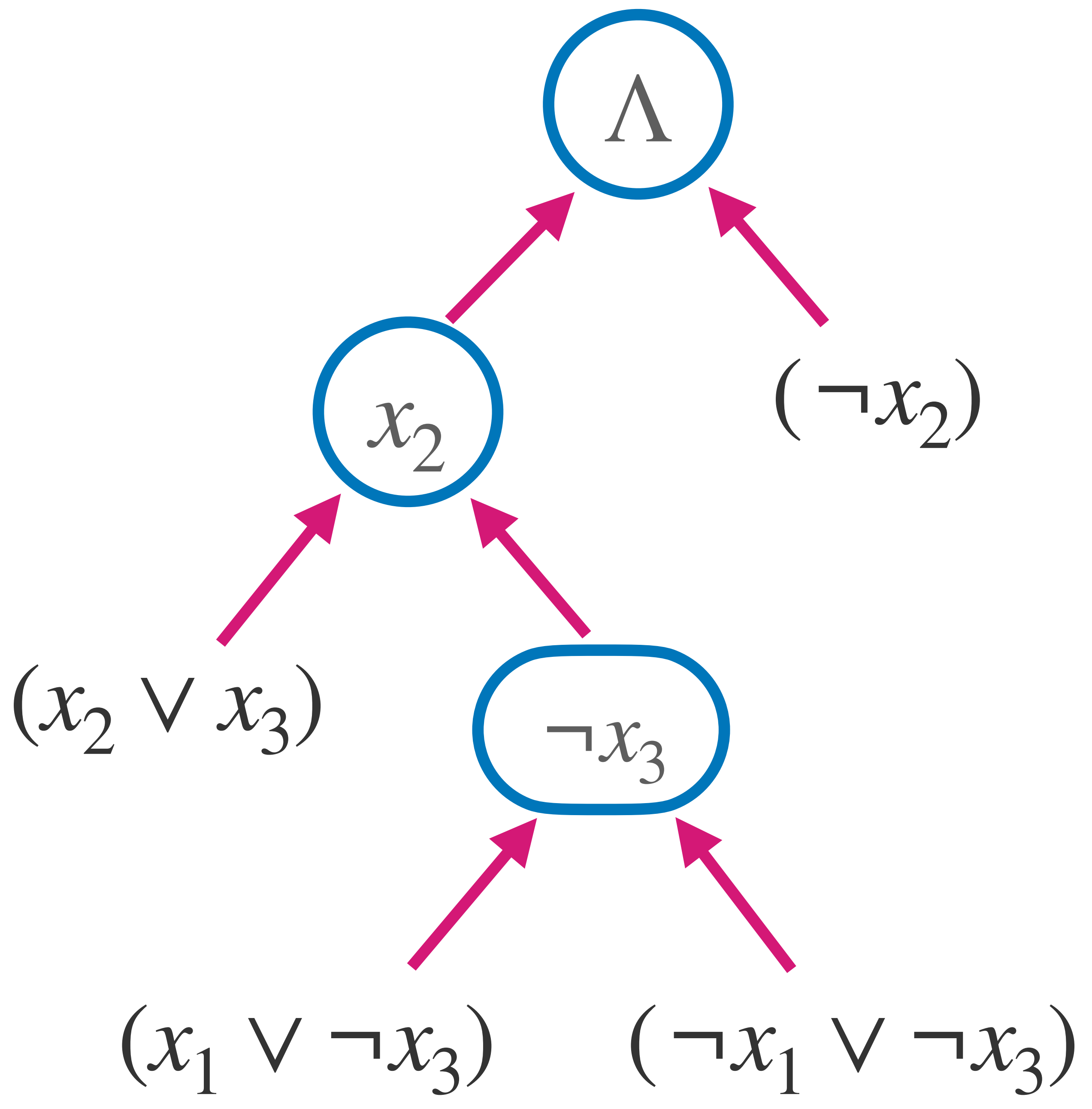
$$(x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2) \land (x_1 \lor \neg x_3)$$

Derive new clauses from old using:

$\rightarrow$ **Resolution rule:**

$$\frac{C_1 \lor x, \quad C_2 \lor \neg x}{C_1 \lor C_2}$$

**Goal:** derive empty clause $\Lambda$




Resolution rule is **sound**

$\Longrightarrow$ Derivation of $\Lambda$ certifies unsatisfiability

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

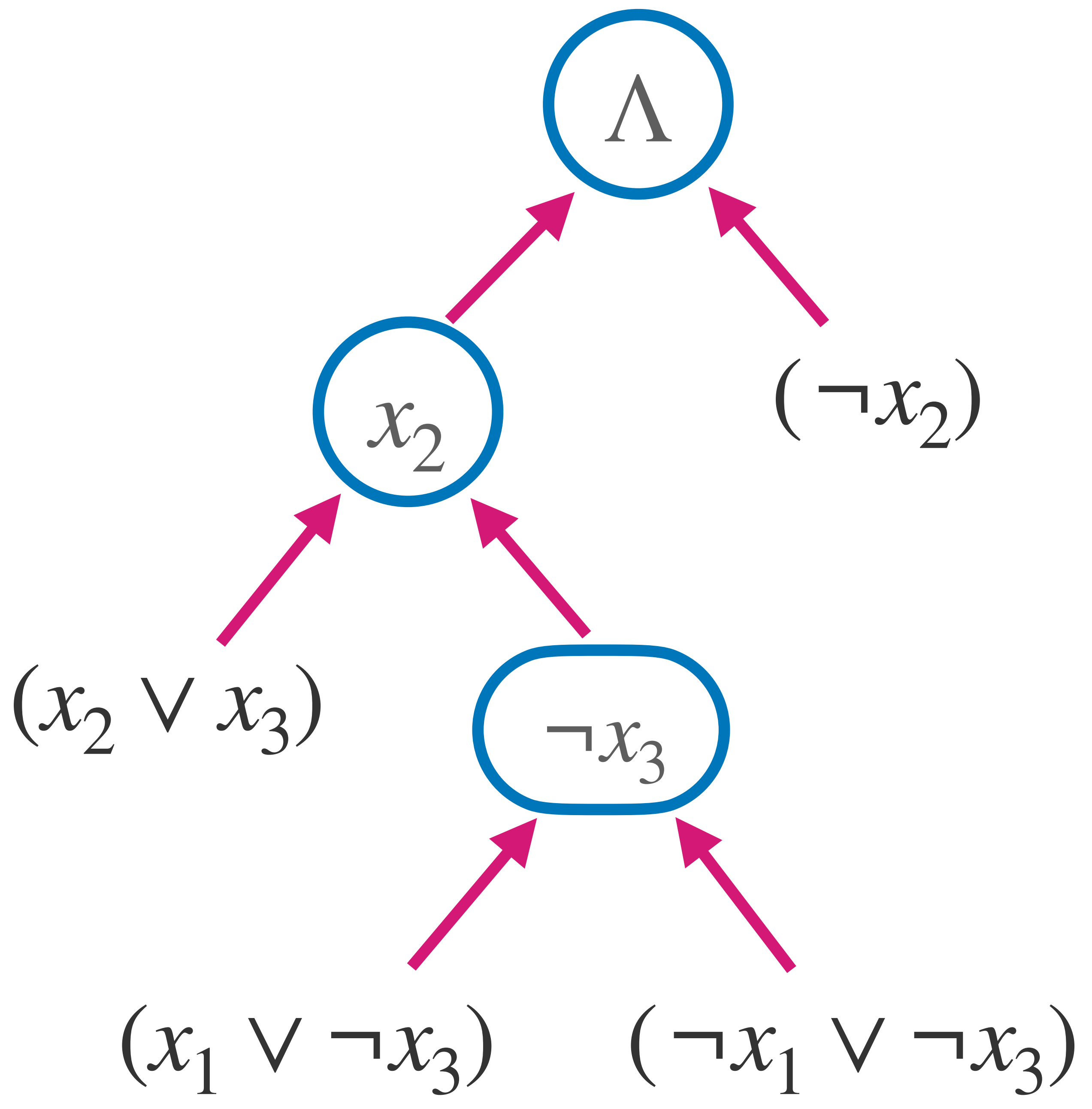
$$(x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

Derive new clauses from old using:

→ **Resolution rule:**

$$\frac{C_1 \vee x, \quad C_2 \vee \neg x}{C_1 \vee C_2}$$

**Goal:** derive empty clause $\Lambda$




**Size:** # of clauses

Resolution rule is **sound**

$\Longrightarrow$ Derivation of $\Lambda$ certifies unsatisfiability

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

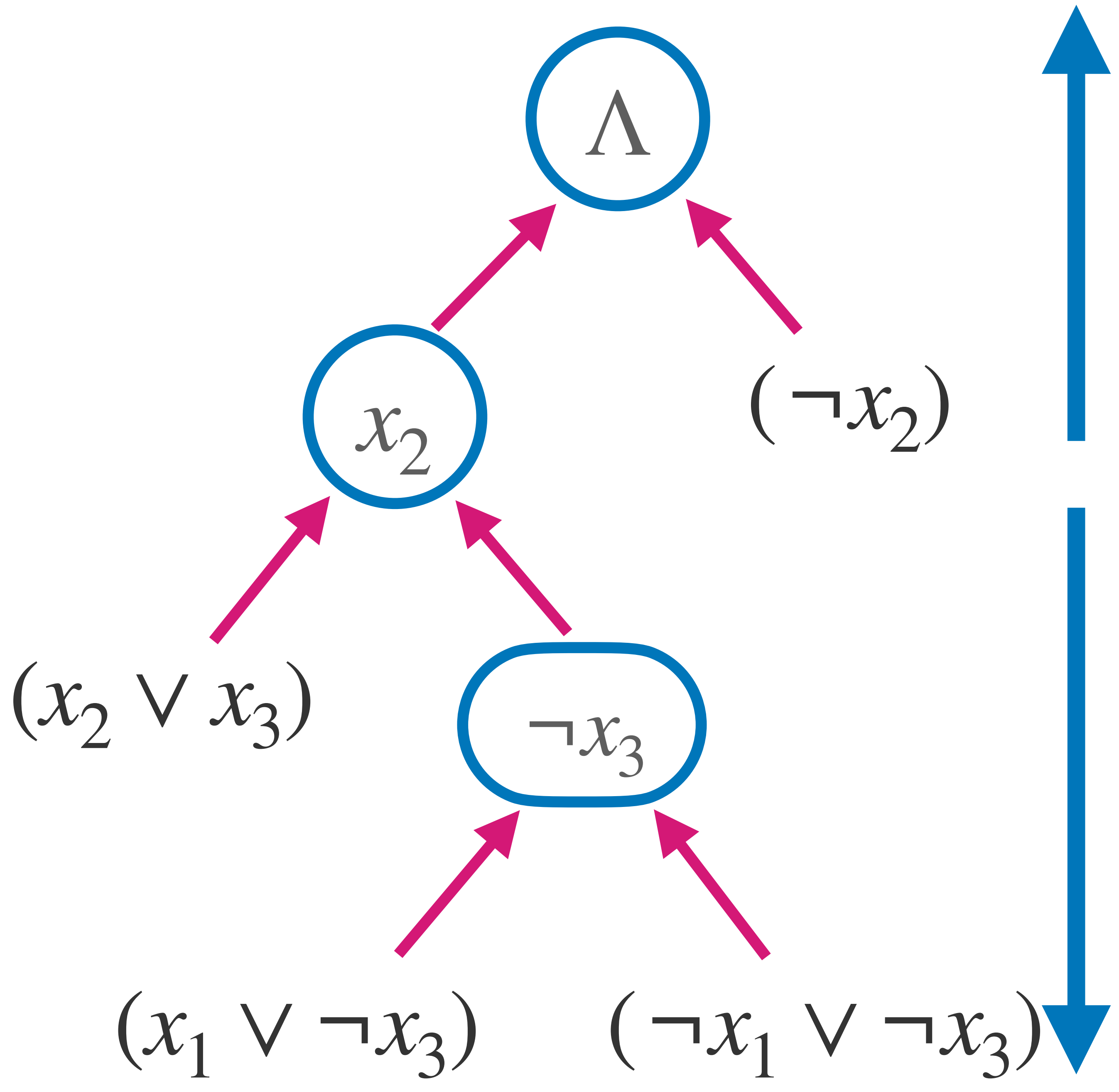
Derive new clauses from old using:

$\rightarrow$ **Resolution rule:**

$$\frac{C_1 \vee x, \quad C_2 \vee \neg x}{C_1 \vee C_2}$$

**Goal:** derive empty clause $\Lambda$

$$(x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$




**Size:** # of clauses

**Depth:** longest root-to-leaf path

Resolution rule is **sound**

$\Longrightarrow$ Derivation of $\Lambda$ certifies unsatisfiability

# Resolution

**Resolution:** A method for proving that a CNF formula is **unsatisfiable**

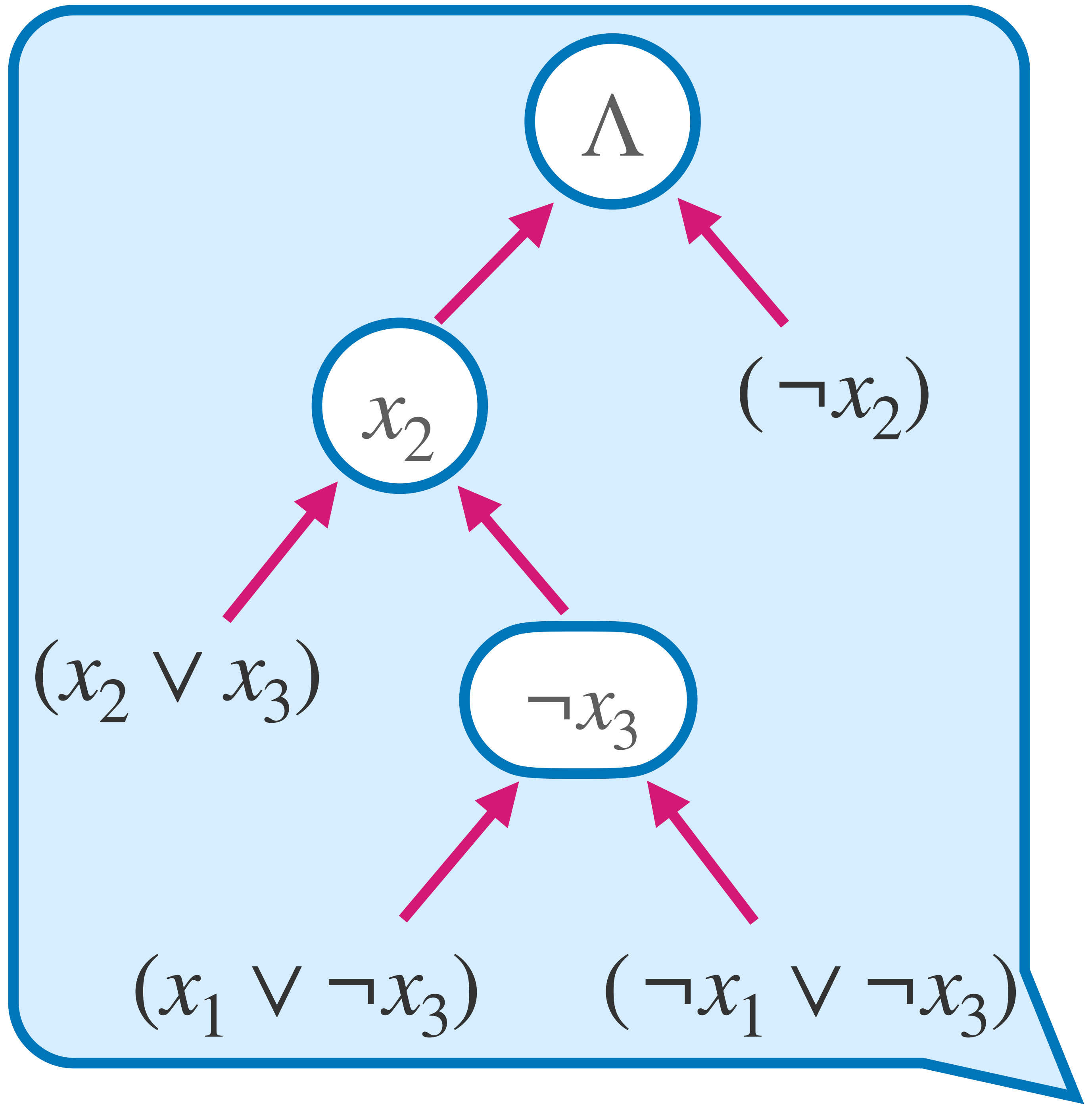
$$(x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

Derive new clauses from old using:

→ **Resolution rule:**

$$\frac{C_1 \vee x, \quad C_2 \vee \neg x}{C_1 \vee C_2}$$

**Goal:** derive empty clause $\Lambda$




Tree proof!

**Size:** # of clauses

**Depth:** longest root-to-leaf path

Resolution rule is **sound**

$\Longrightarrow$ Derivation of $\Lambda$ certifies unsatisfiability

# Analyzing DPLL

We can **use** (tree) Resolution to study DPLL!

# Analyzing DPLL

We can **use** (tree) Resolution to study DPLL!

$Q$ . What happens if we run DPLL on an unsatisfiable formula?

# DPLL

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2)$$

**Input:** A CNF formula $F$

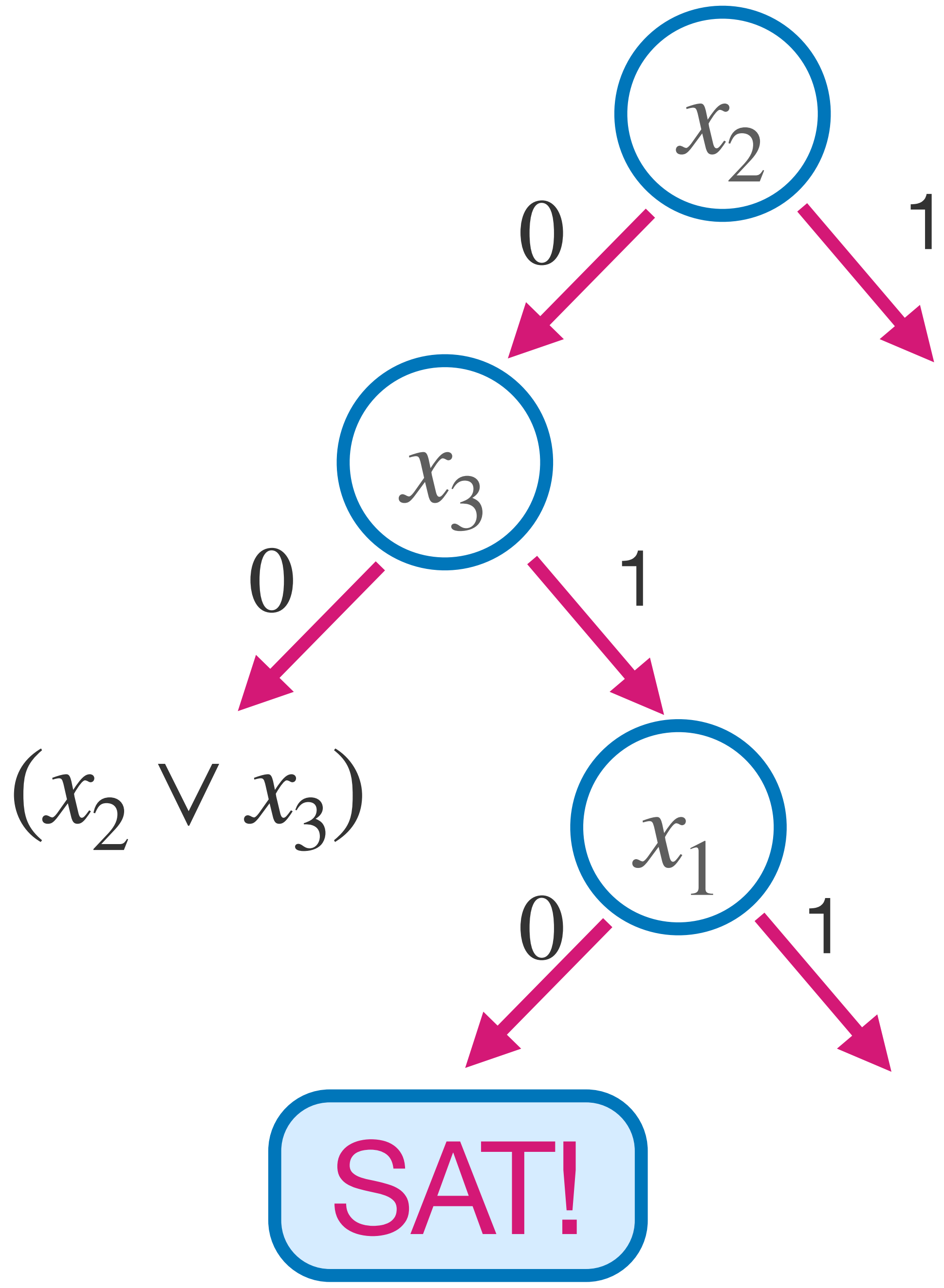
**Output:** A satisfying assignment

**DPLL$(F)$:**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \restriction x_i = 0)$
3. **DPLL**$(F \restriction x_i = 1)$



# DPLL

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

**Input:** A CNF formula $F$

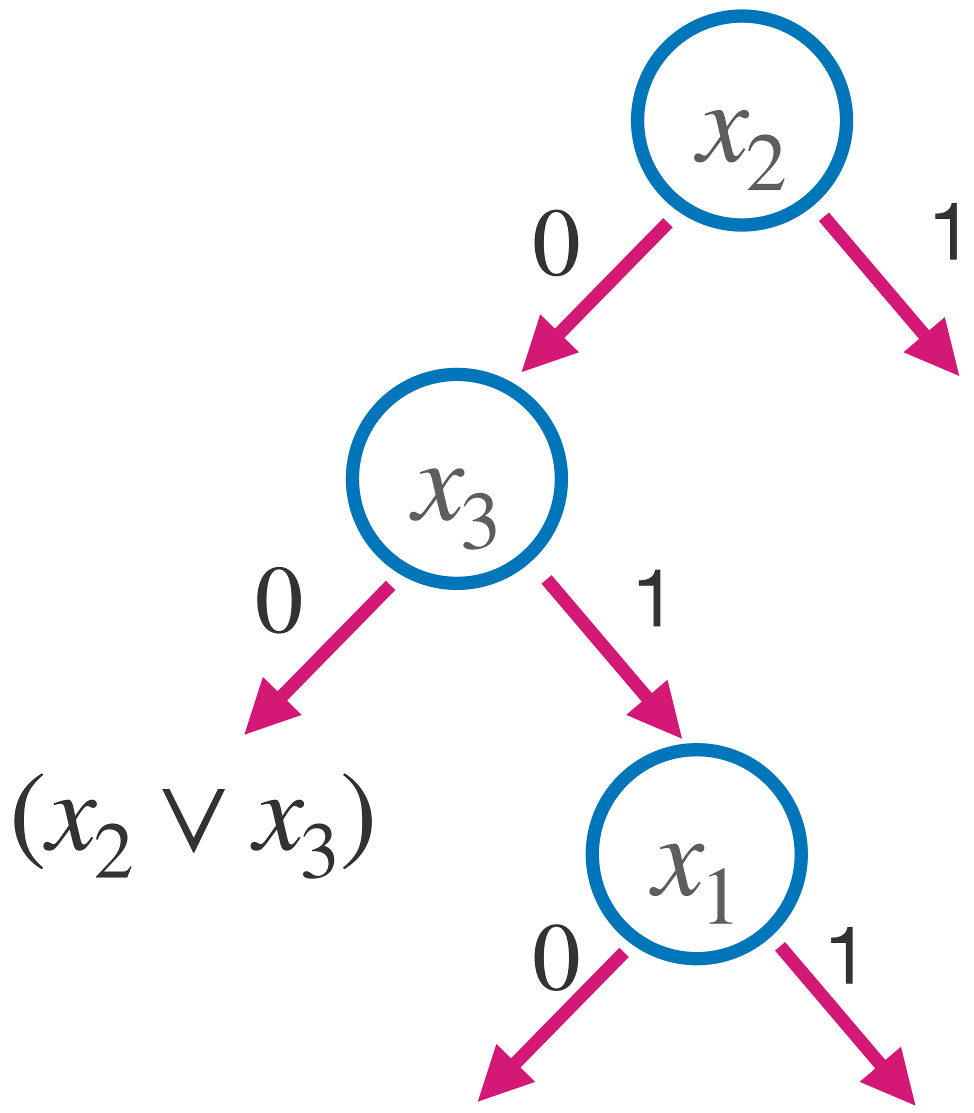
**Output:** A satisfying assignment

**DPLL**$(F)$:

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \restriction x_i = 0)$
3. **DPLL**$(F \restriction x_i = 1)$



# DPLL

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

**Input:** A CNF formula $F$

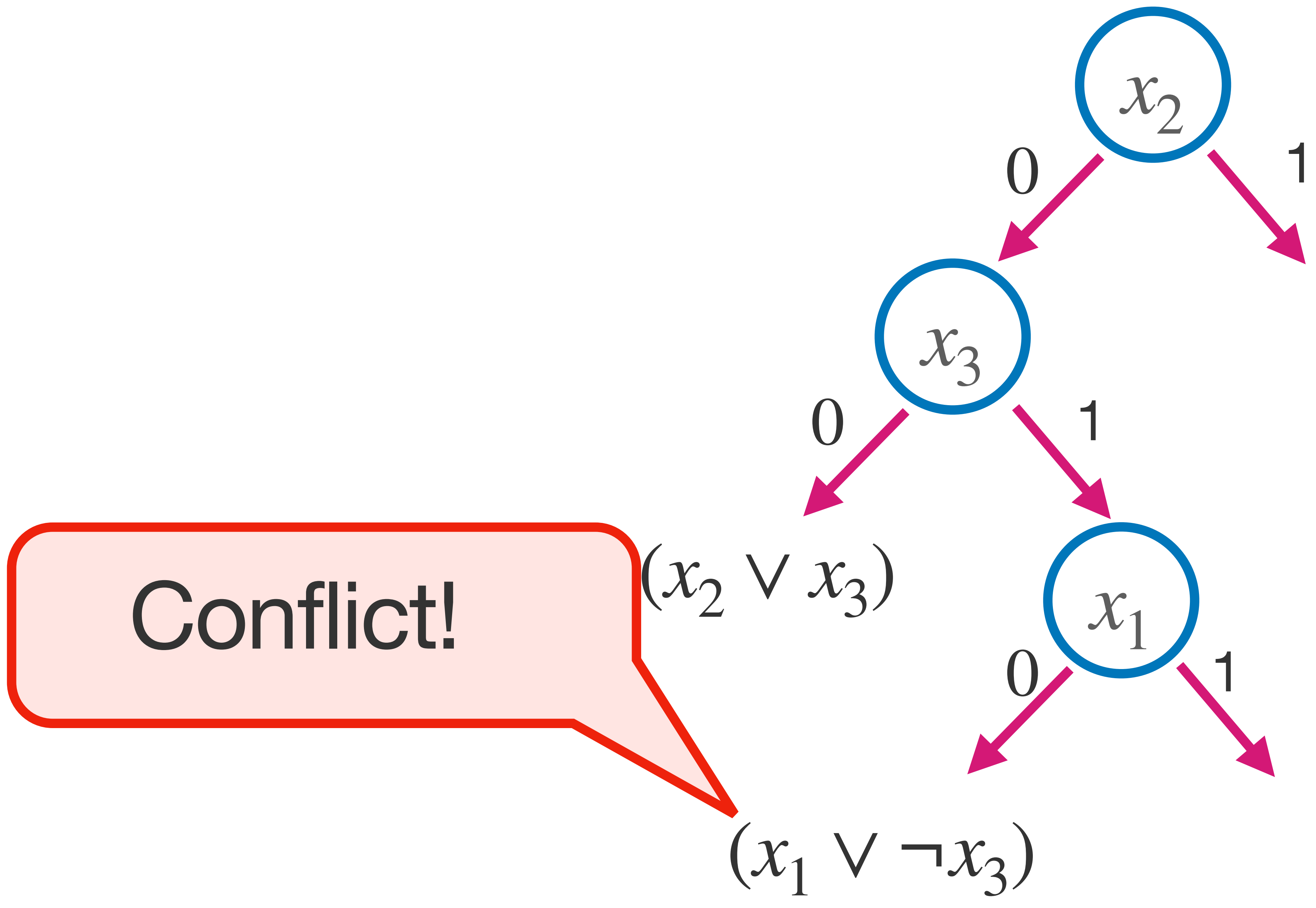
**Output:** A satisfying assignment

**DPLL($F$):**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL($F \restriction x_i = 0$)**
3. **DPLL($F \restriction x_i = 1$)**



# DPLL

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

**Input:** A CNF formula $F$

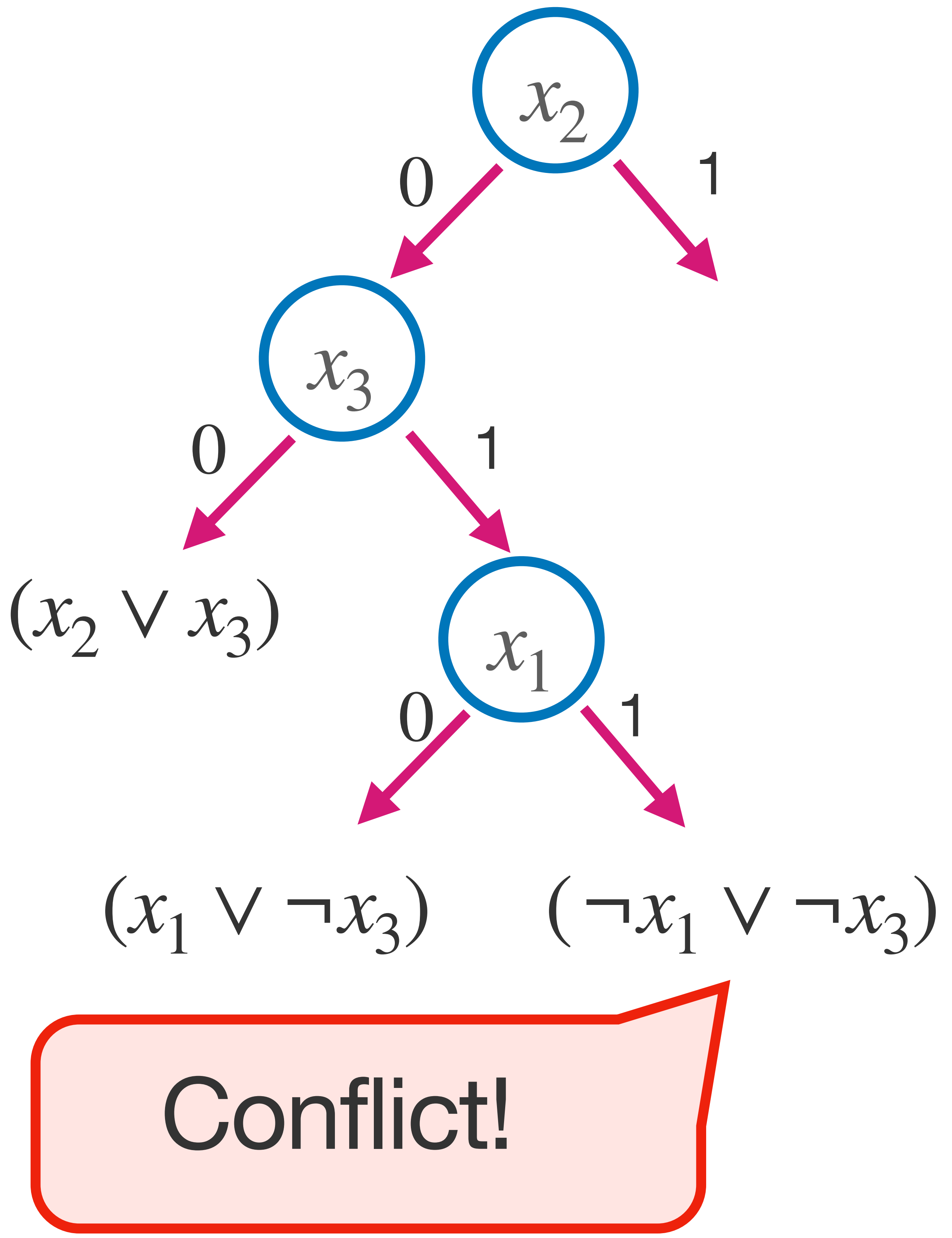
**Output:** A satisfying assignment

**DPLL**$(F)$**:**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \restriction x_i = 0)$
3. **DPLL**$(F \restriction x_i = 1)$



# DPLL

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

**Input:** A CNF formula $F$

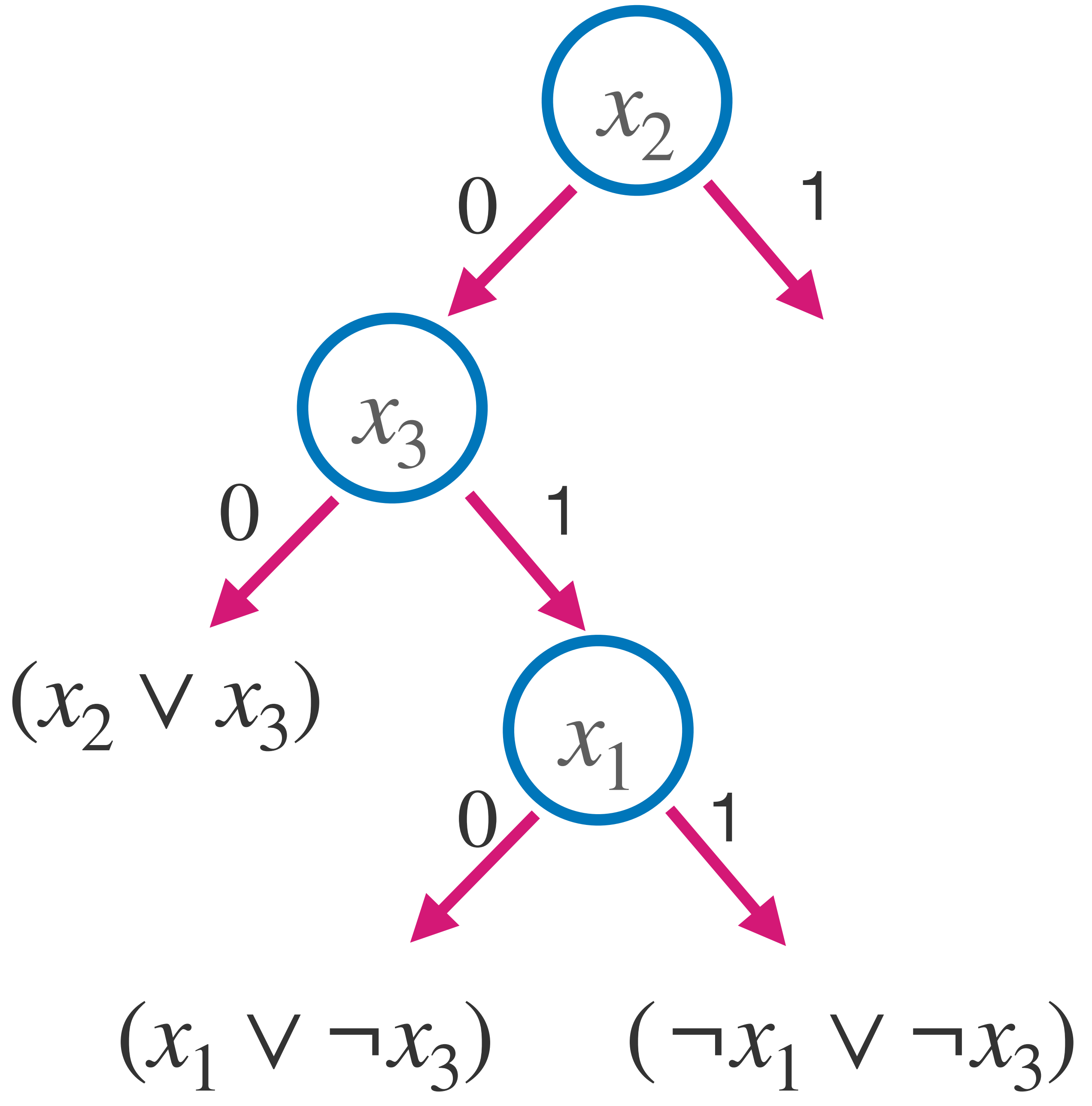
**Output:** A satisfying assignment

**DPLL$(F)$:**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL$(F \restriction x_i = 0)$**
3. **DPLL$(F \restriction x_i = 1)$**



# DPLL

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

**Input:** A CNF formula $F$

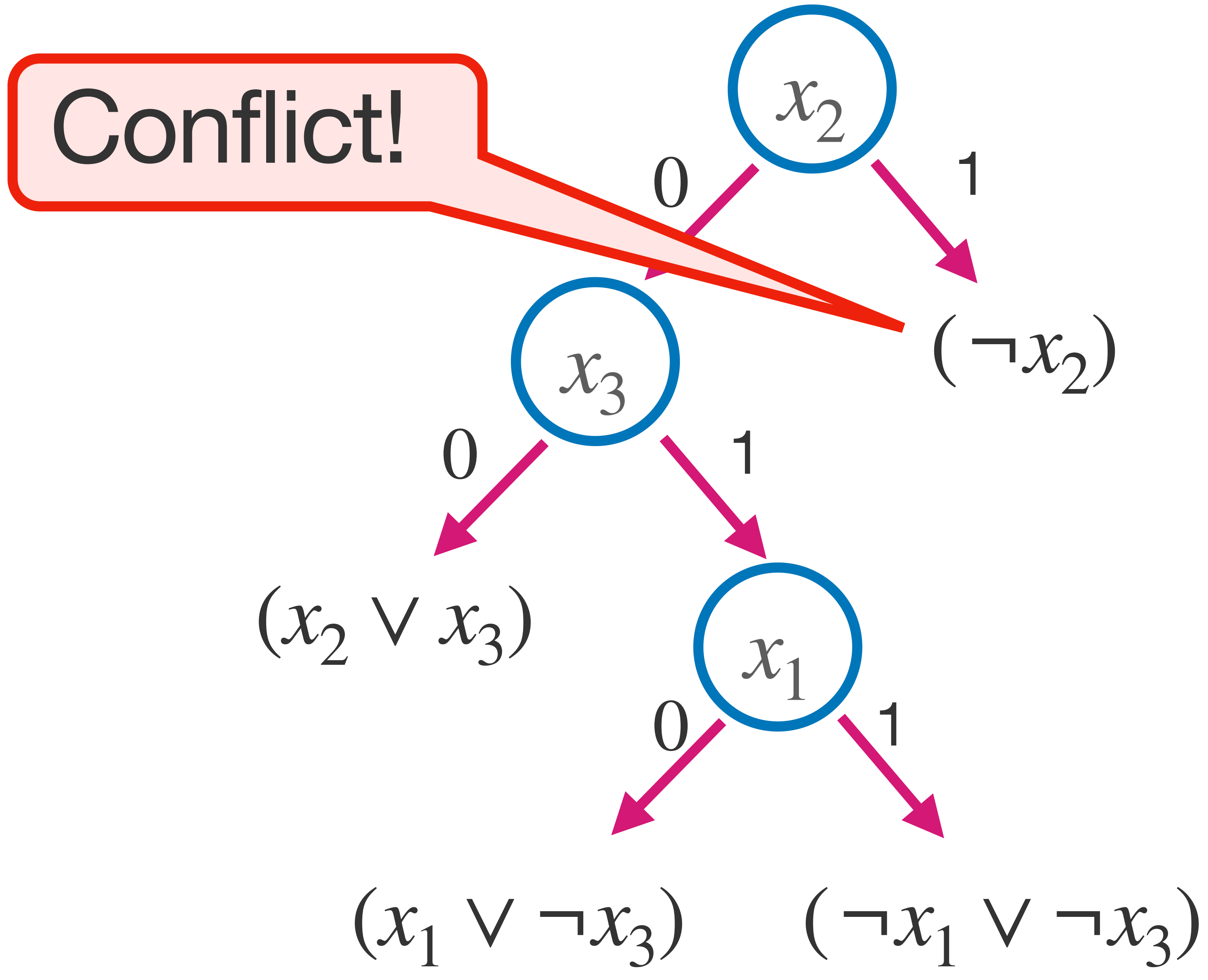
**Output:** A satisfying assignment

**DPLL**$(F)$**:**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \restriction x_i = 0)$
3. **DPLL**$(F \restriction x_i = 1)$



# DPLL

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

**Input:** A CNF formula $F$

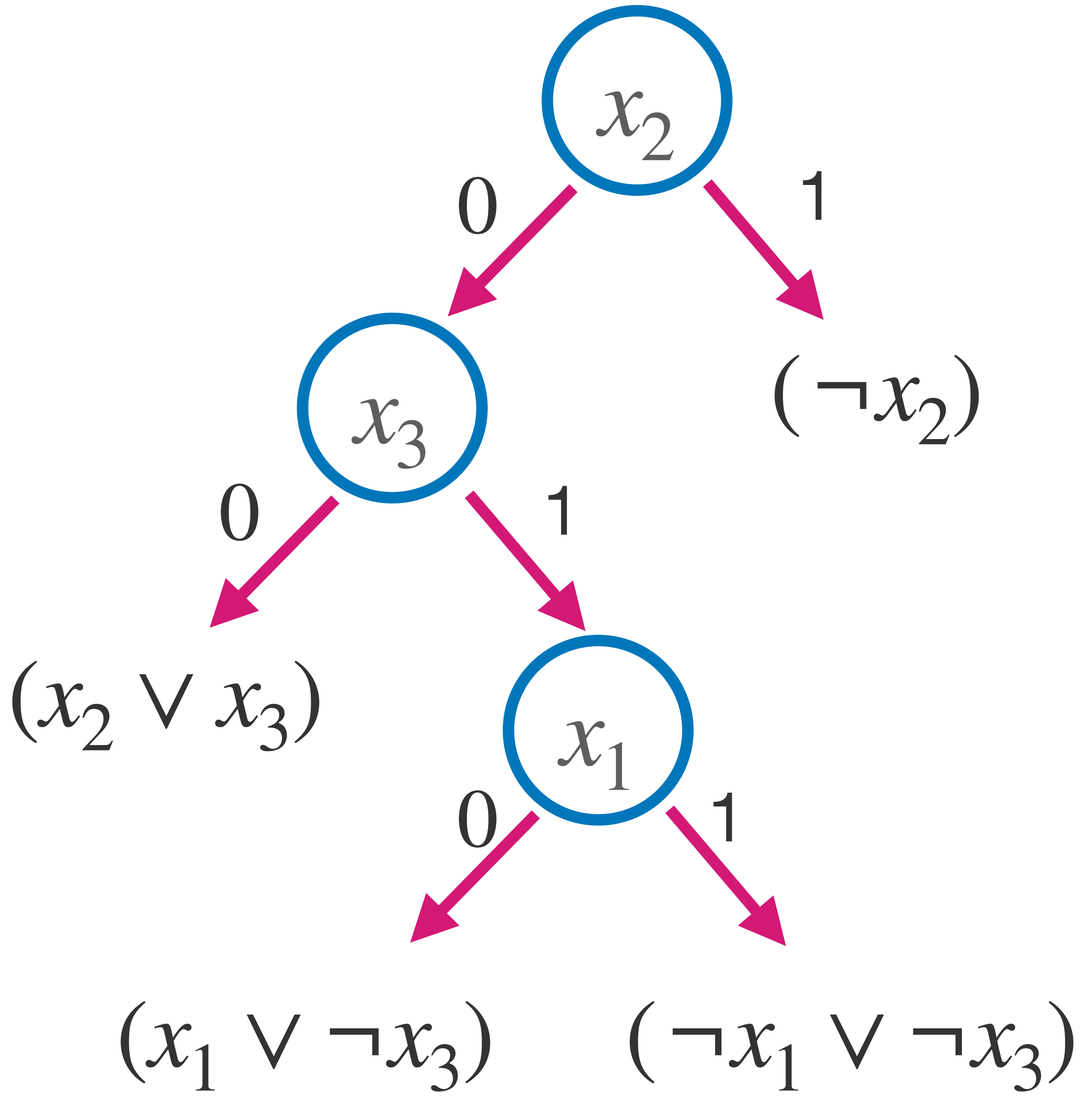
**Output:** A satisfying assignment

**DPLL**$(F)$**:**

**If** $F = 1$, output SAT

**If** $F \neq 0$, do:

1. **Choose** a variable $x_i$ (heuristically)
2. **DPLL**$(F \restriction x_i = 0)$
3. **DPLL**$(F \restriction x_i = 1)$

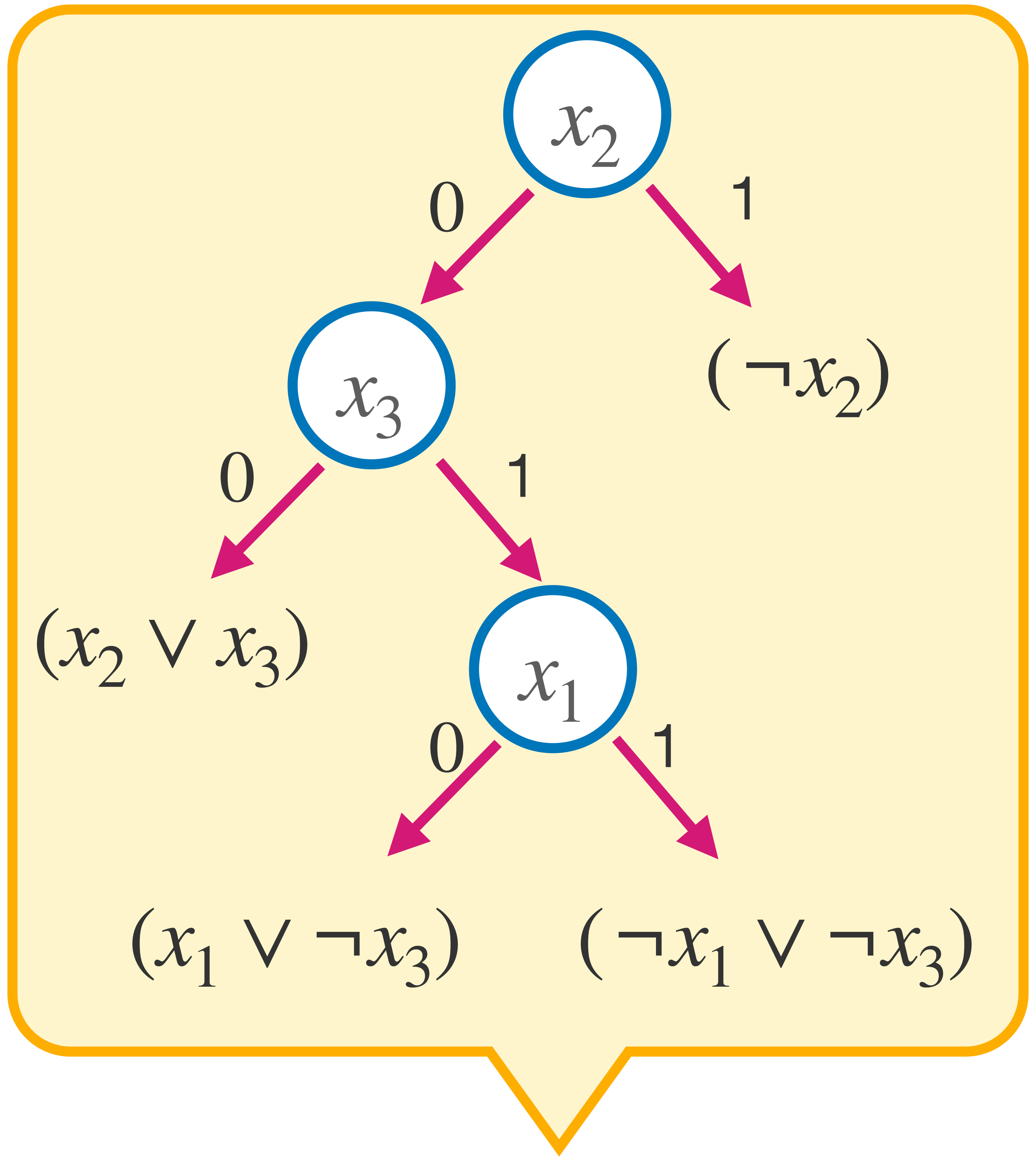



No satisfying assignment!

# DPLL

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

Execution of DPLL is a **proof** that $F$ is unsatisfiable!

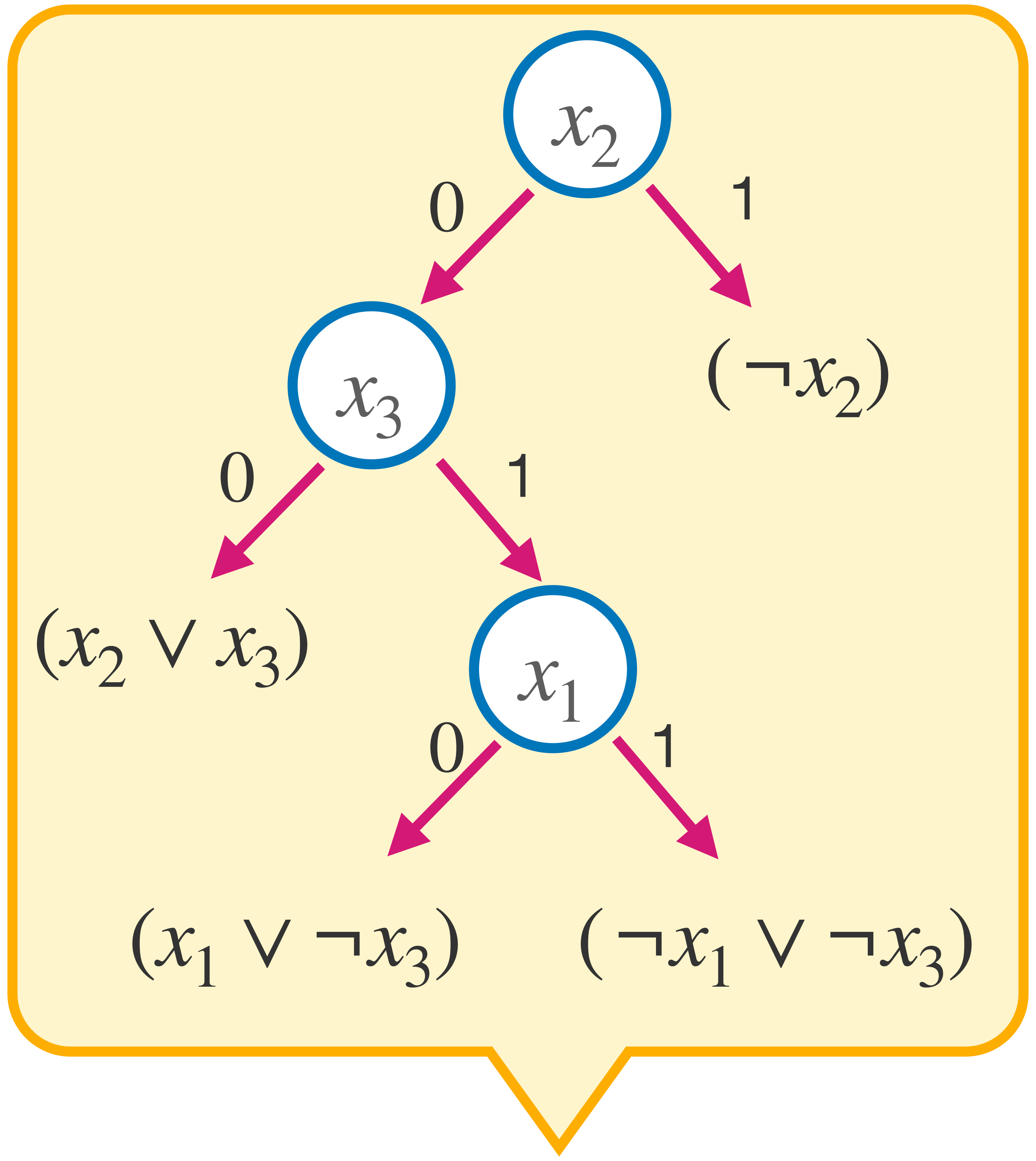



Proof of unsatisfiability!

# DPLL

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

Execution of DPLL is a ~~proof~~ that $F$ is unsatisfiable!



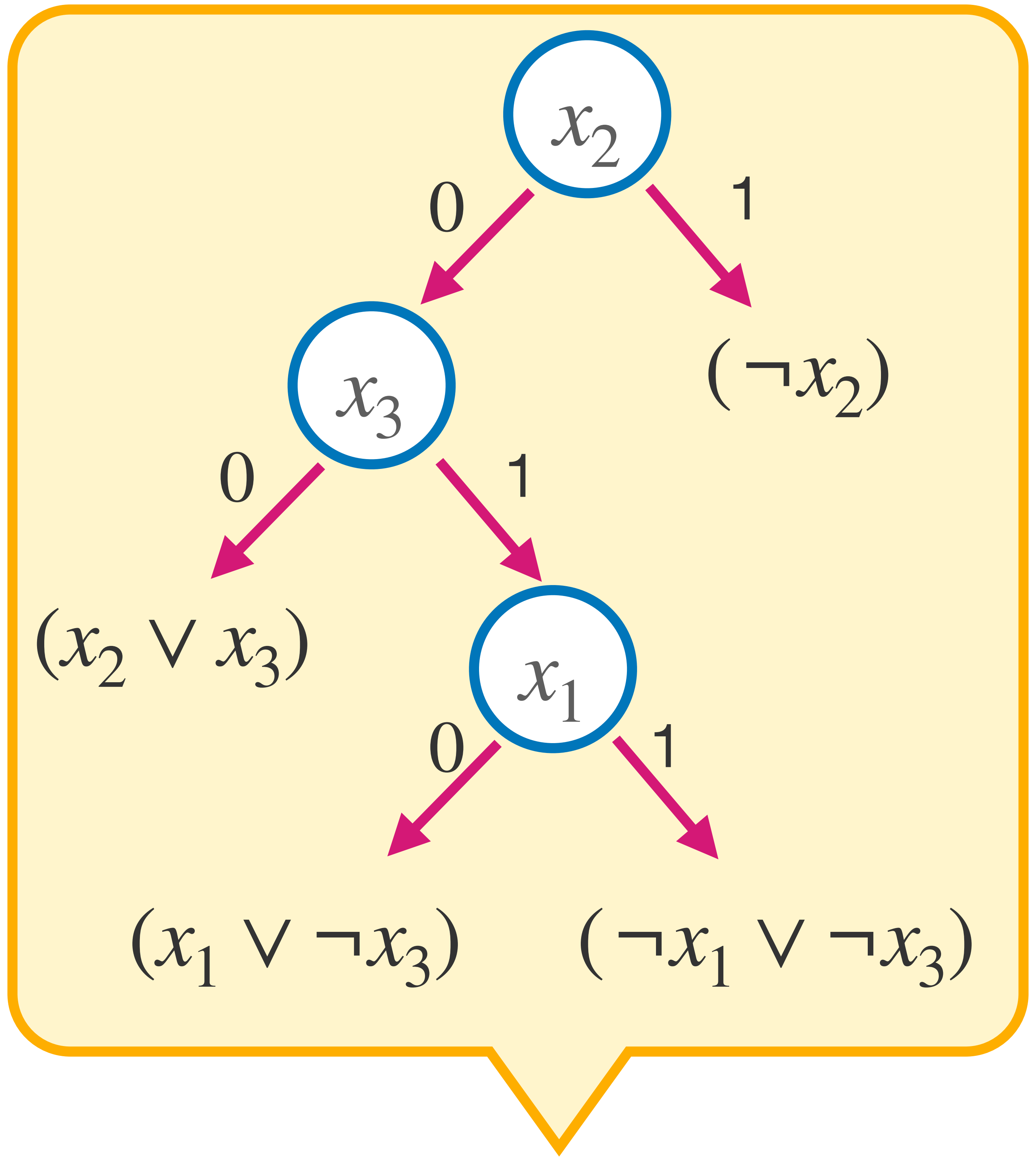

Proof of unsatisfiability!

# DPLL

$$F = (x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2) \land (x_1 \lor \neg x_3)$$

Execution of DPLL is a **tree**

**Resolution proof** of unsatisfiability




Proof of unsatisfiability!

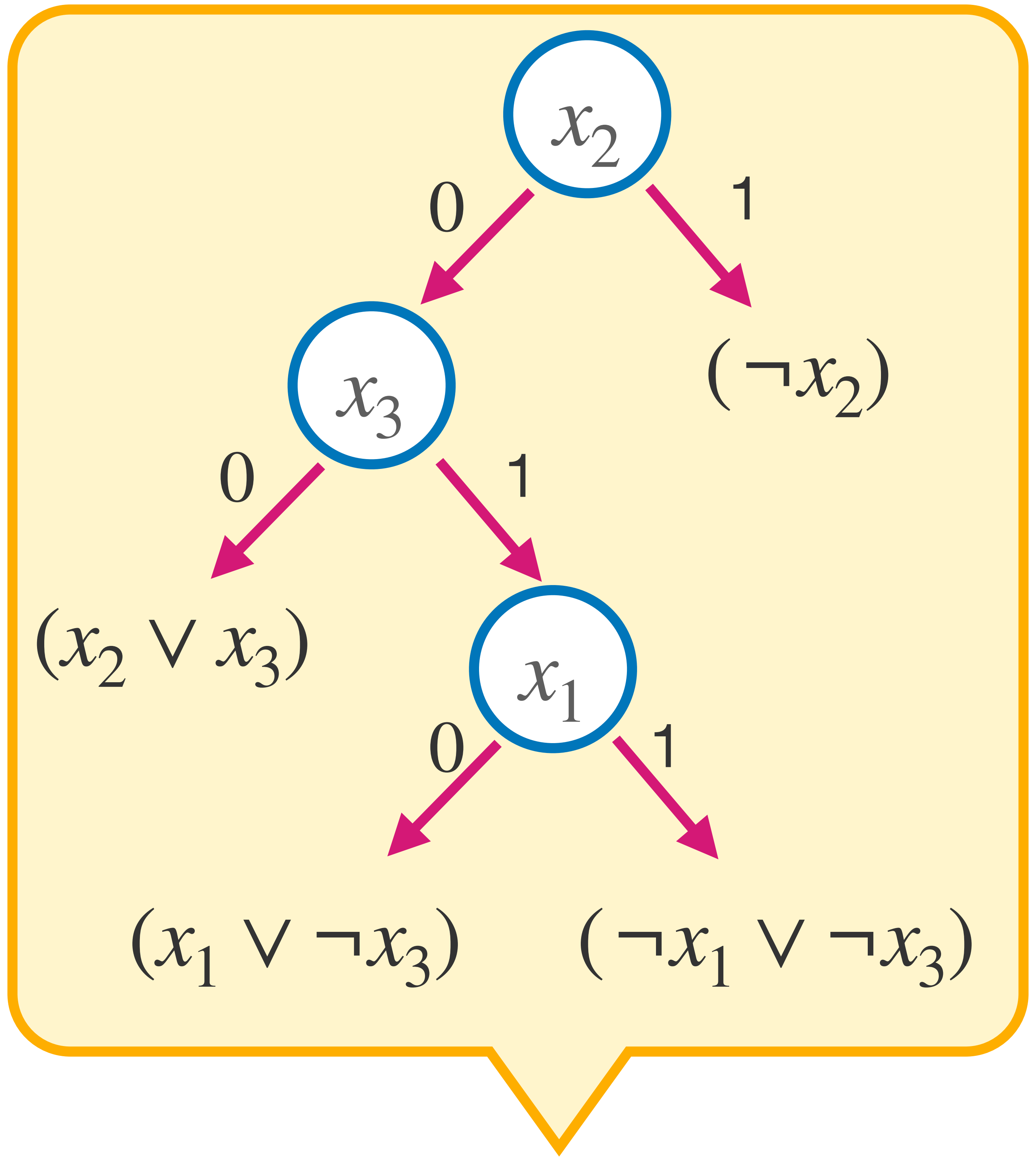
# DPLL

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

Execution of DPLL is a **tree**

**Resolution proof** of unsatisfiability

→ Every time we query a variable, resolve on it!



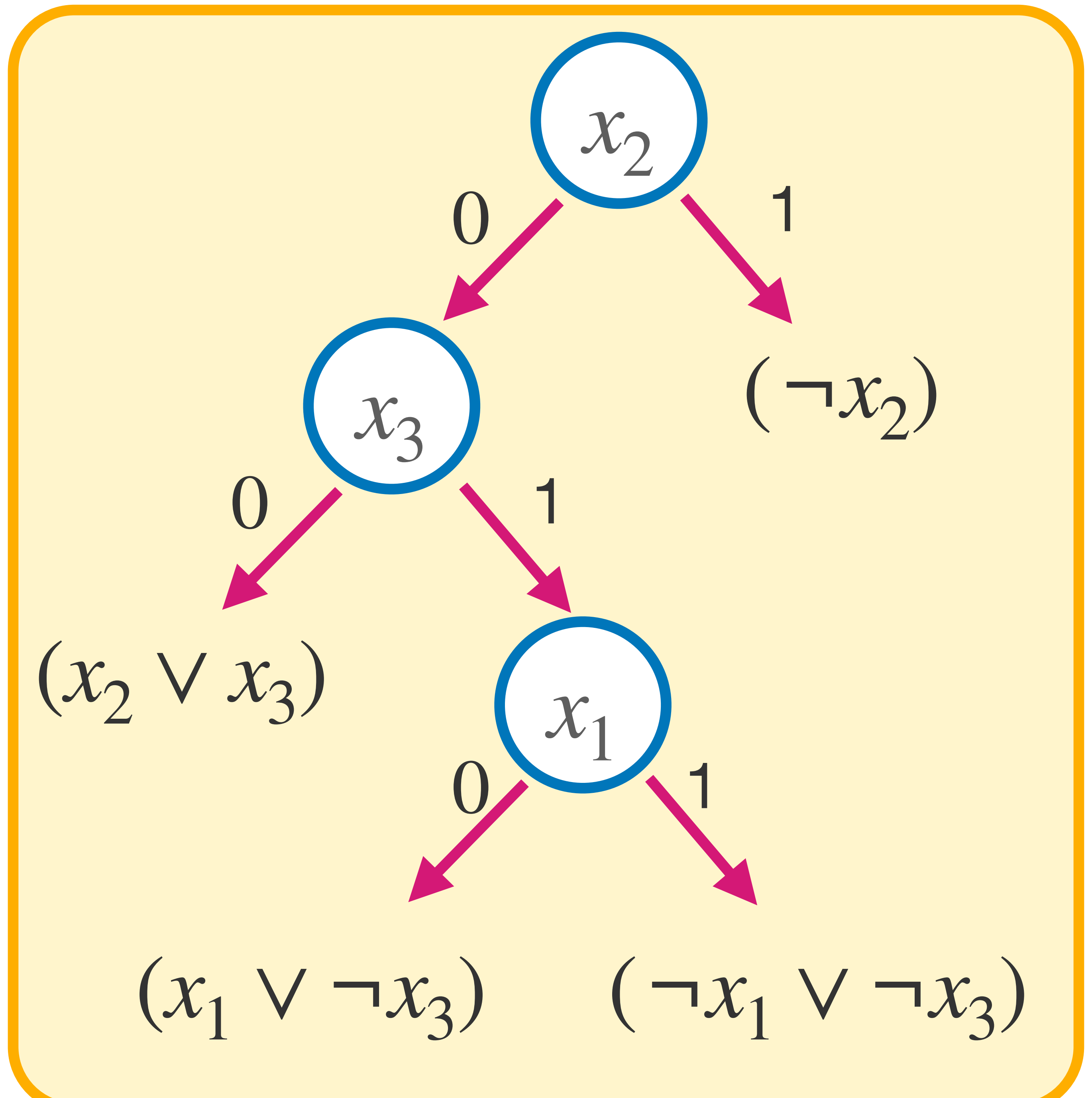

DPLL

# DPLL

$$F = (x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2) \land (x_1 \lor \neg x_3)$$

Execution of DPLL is a **tree Resolution proof** of unsatisfiability

$\rightarrow$ Every time we query a variable, resolve on it!




**Tree Resolution**

**DPLL**

# DPLL

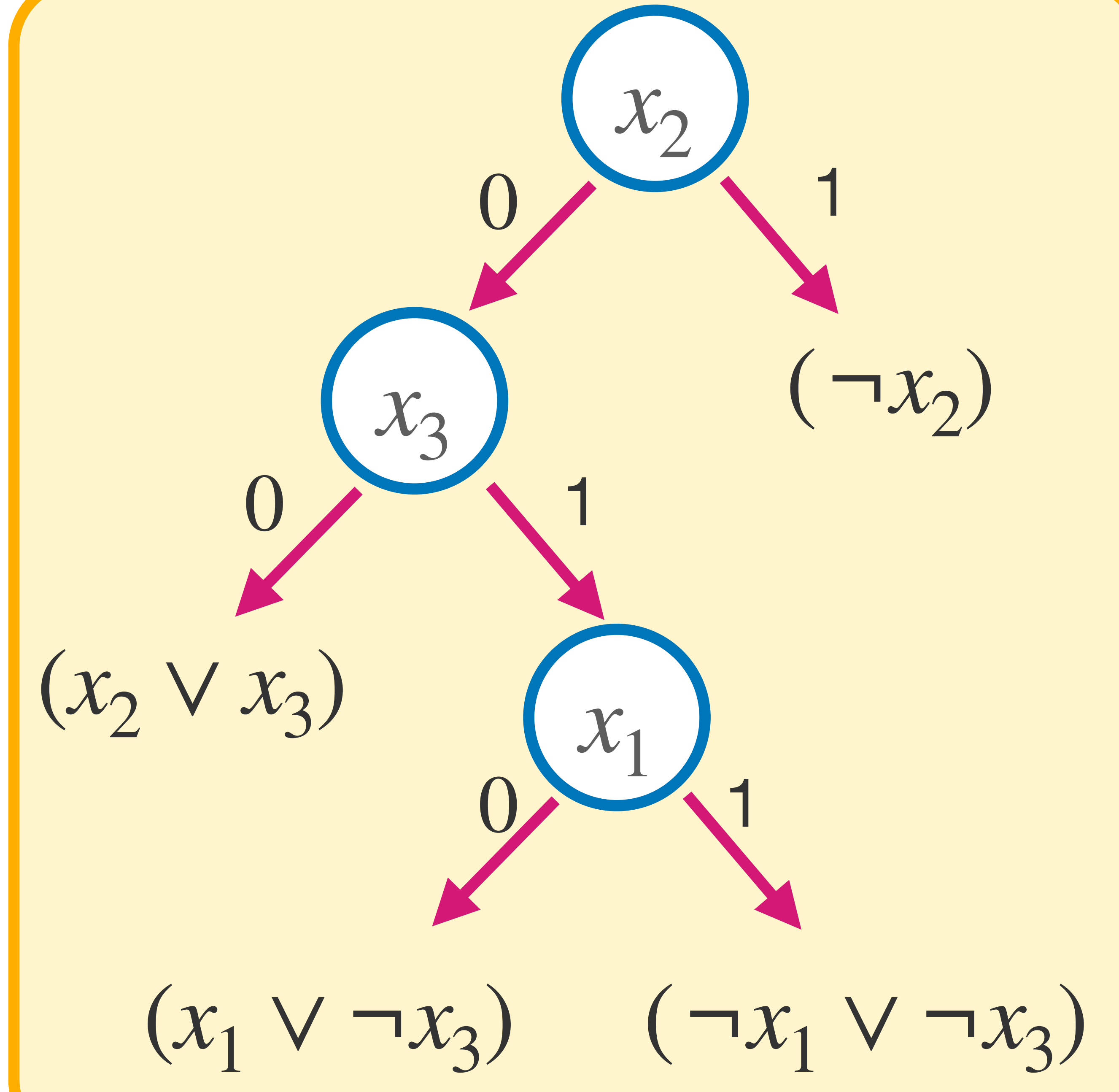
$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

Execution of DPLL is a **tree Resolution proof** of unsatisfiability

→ Every time we query a variable, resolve on it!

$(x_1 \vee \neg x_3)$ $(\neg x_1 \vee \neg x_3)$

**Tree Resolution**



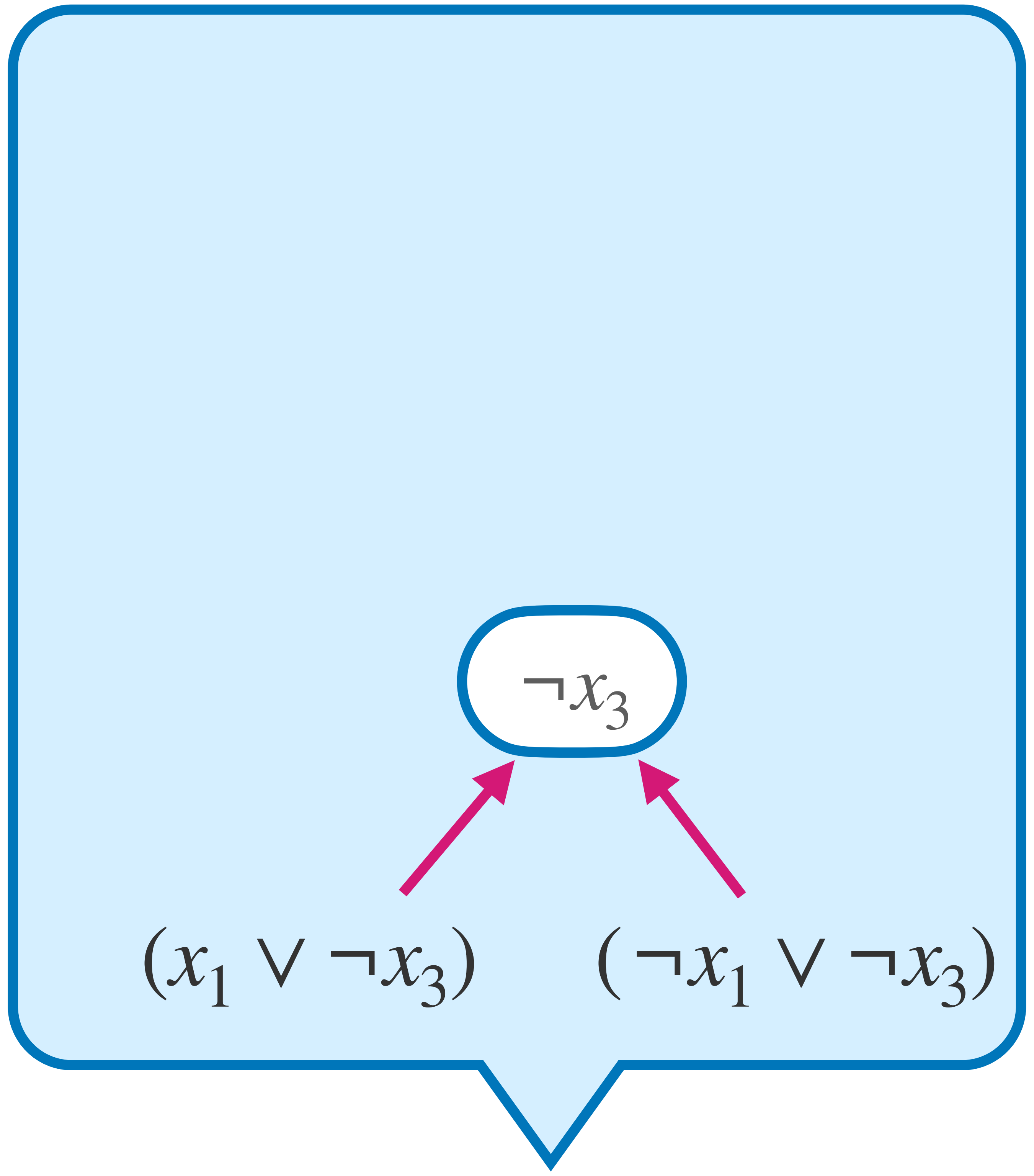

**DPLL**

# DPLL

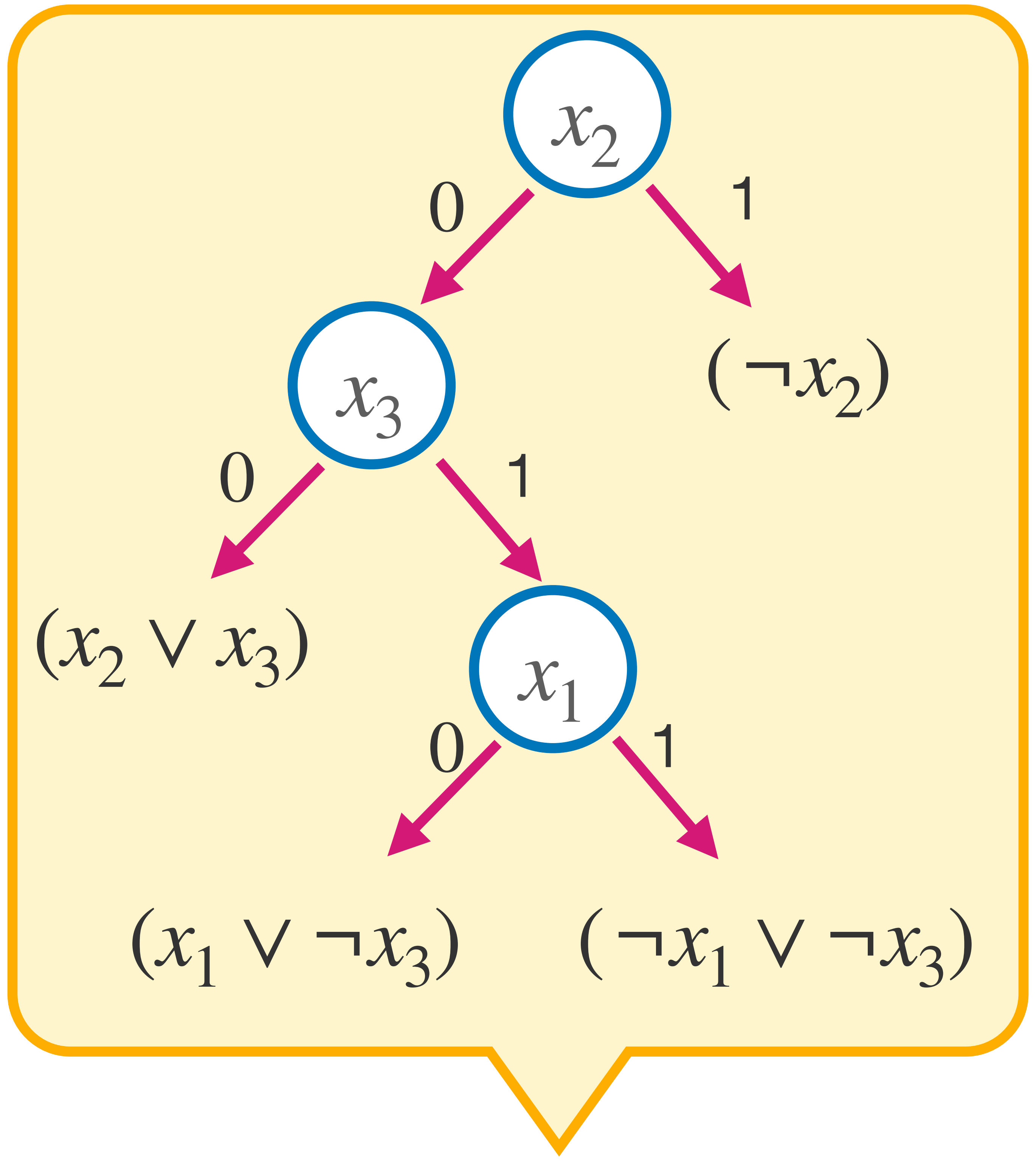
$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

Execution of DPLL is a **tree Resolution proof** of unsatisfiability

→ Every time we query a variable, resolve on it!




**Tree Resolution**



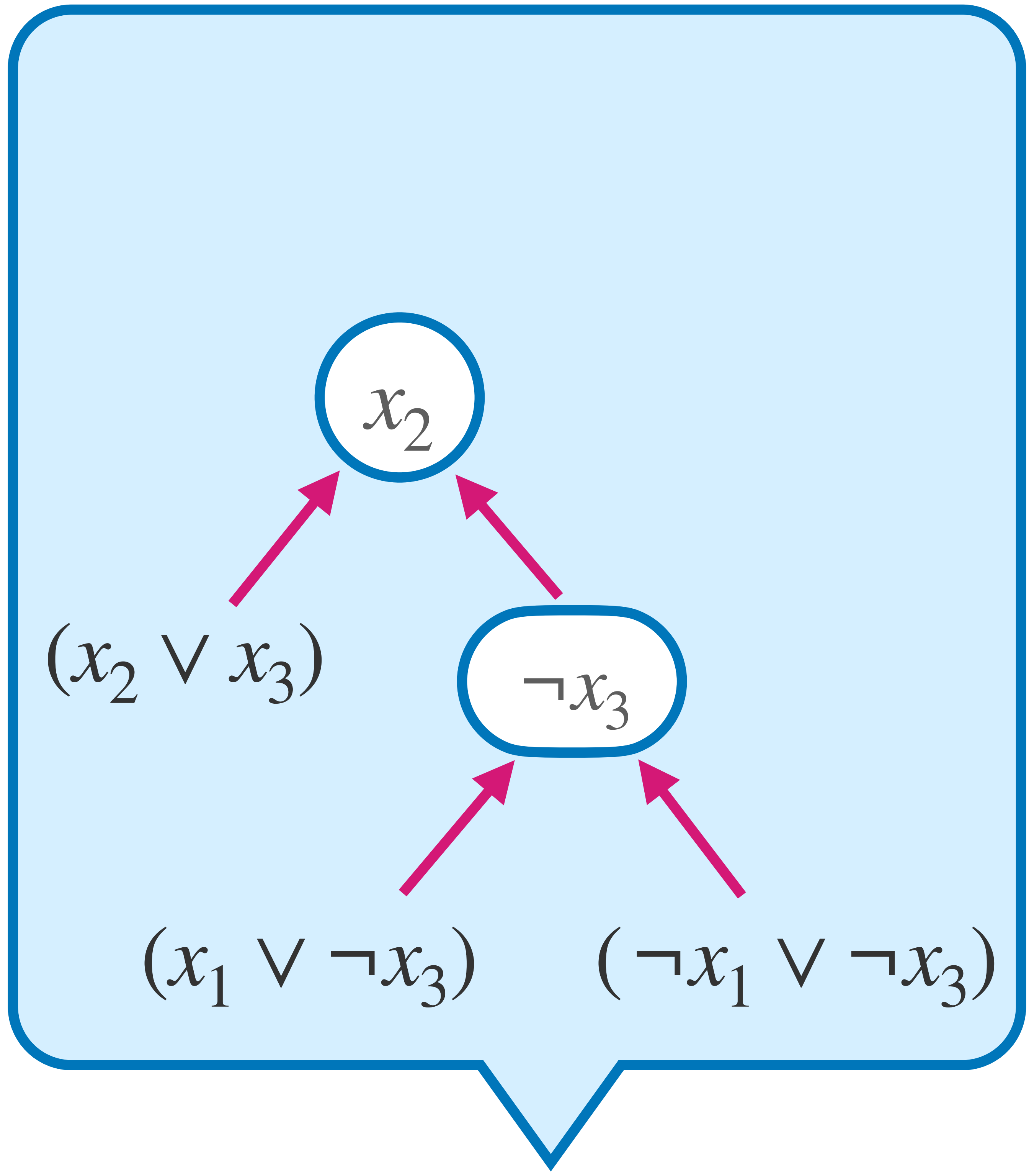

**DPLL**

# DPLL

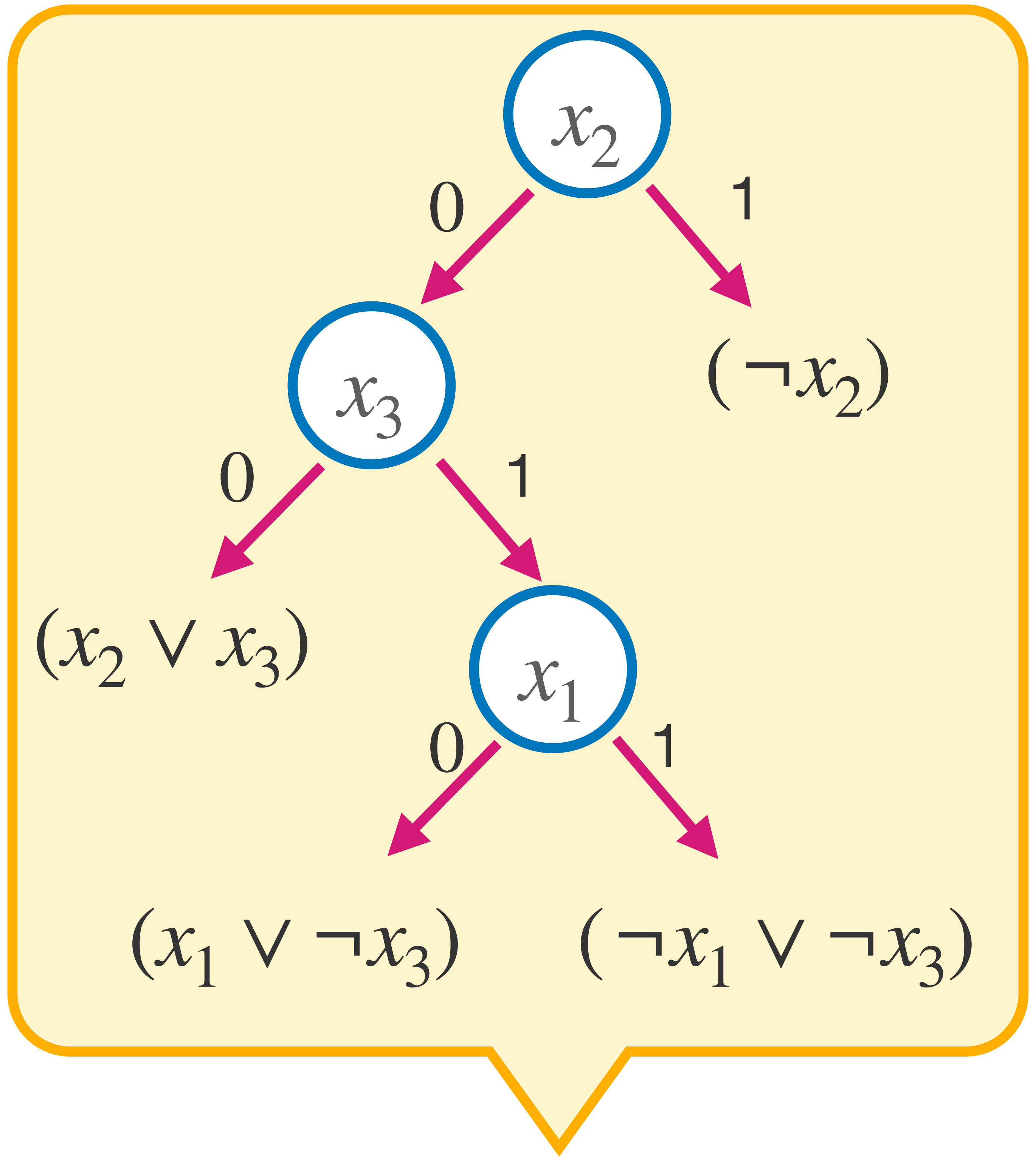
$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

Execution of DPLL is a **tree Resolution proof** of unsatisfiability

$\rightarrow$ Every time we query a variable, resolve on it!




**Tree Resolution**



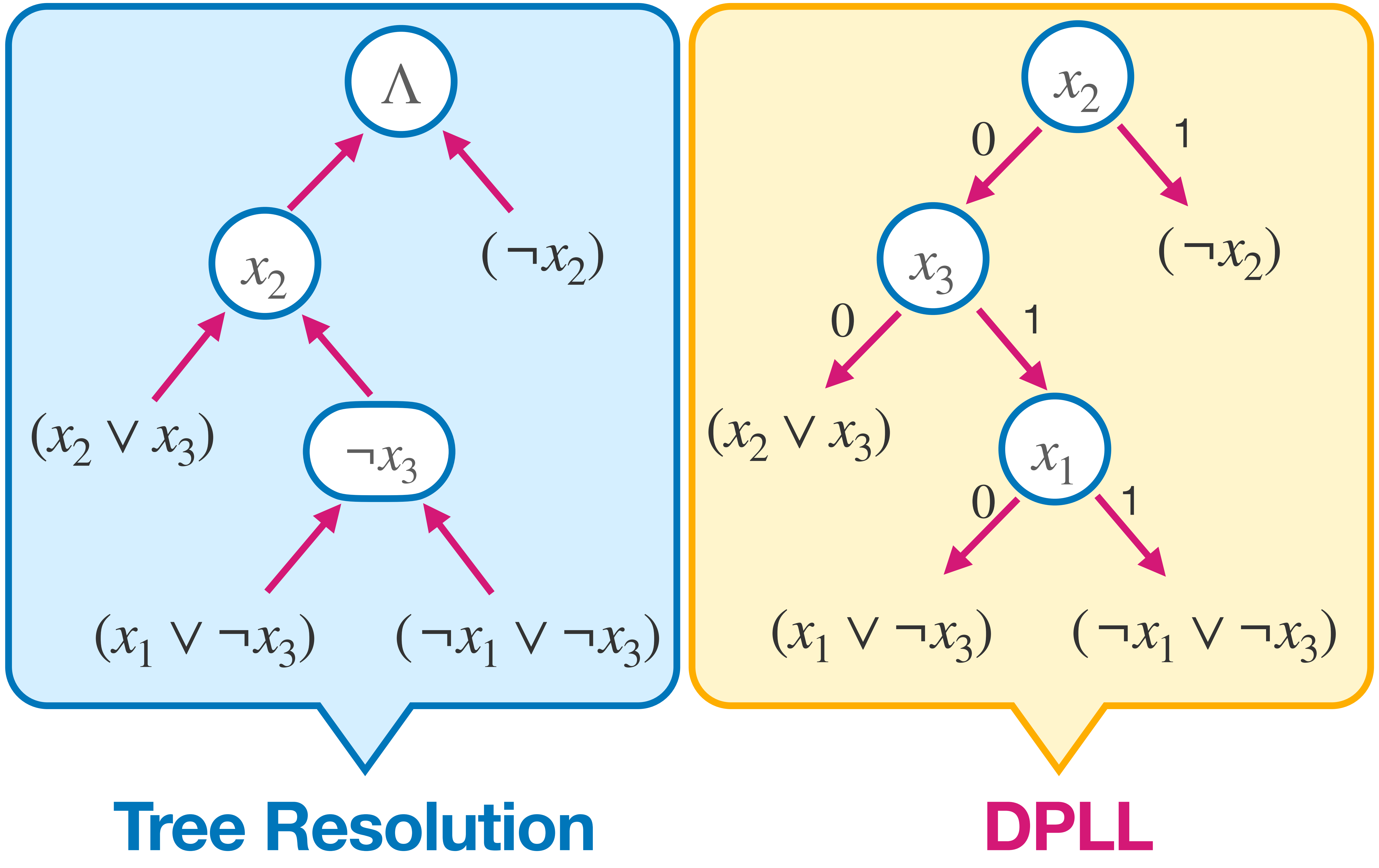

**DPLL**

# DPLL

$$F = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_3)$$

Execution of DPLL is a **tree Resolution proof** of unsatisfiability

→ Every time we query a variable, resolve on it!



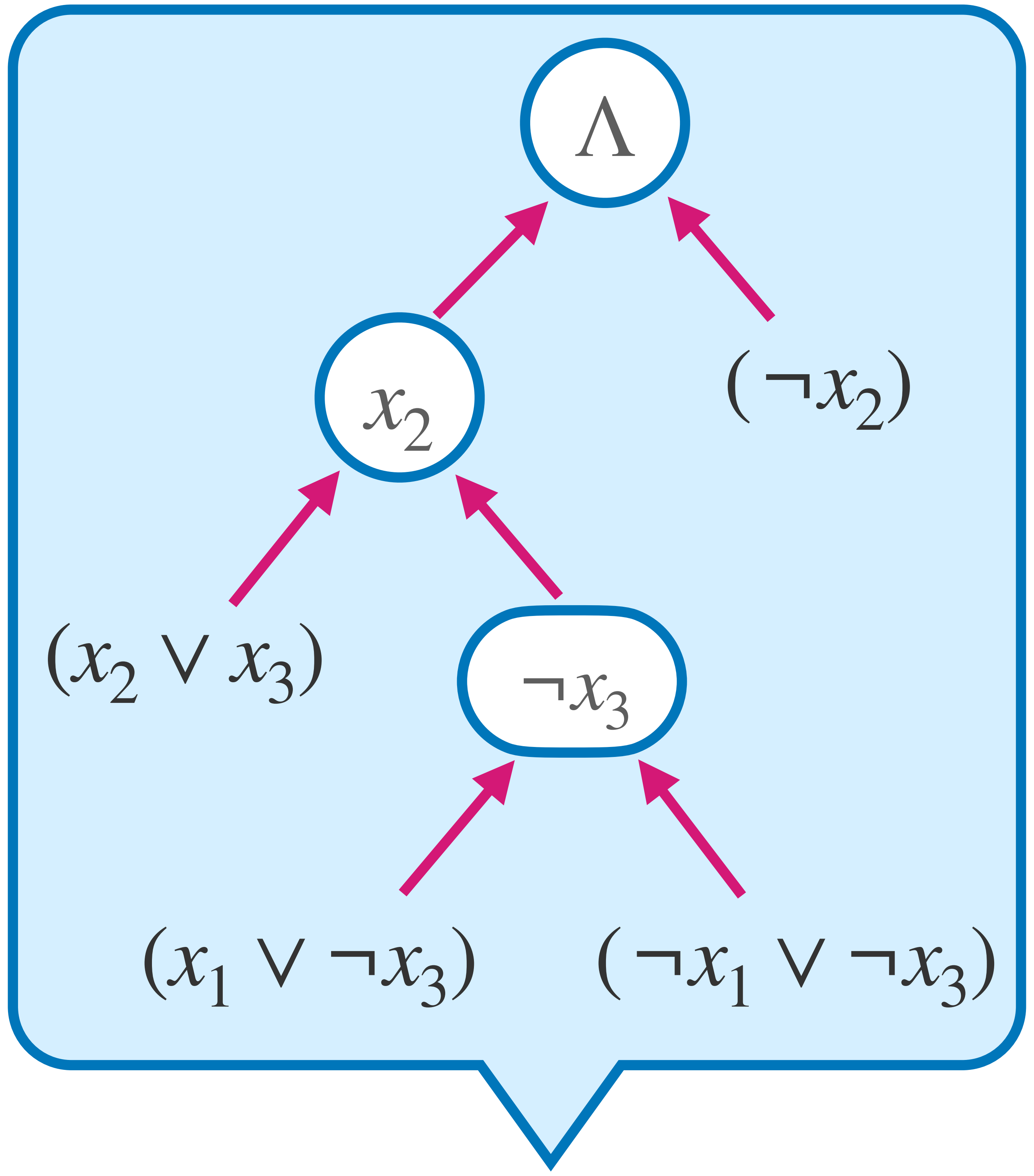
# DPLL

Execution of DPLL is a **tree Resolution proof** of unsatisfiability

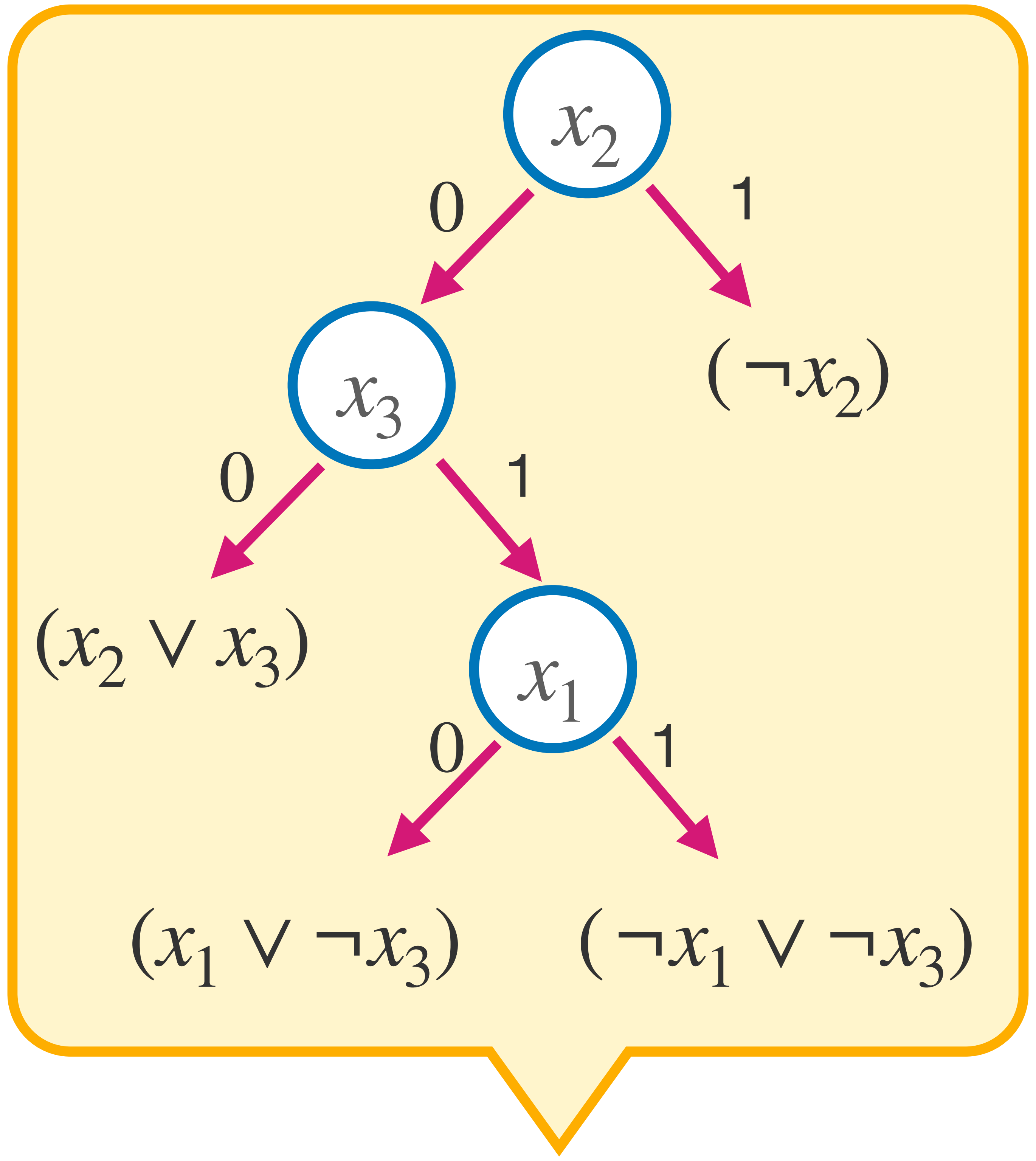
→ Every time we query a variable, resolve on it!

**Upshot**

tree Resolution proofs = DPLL trees




**Tree Resolution**




**DPLL**

# DPLL

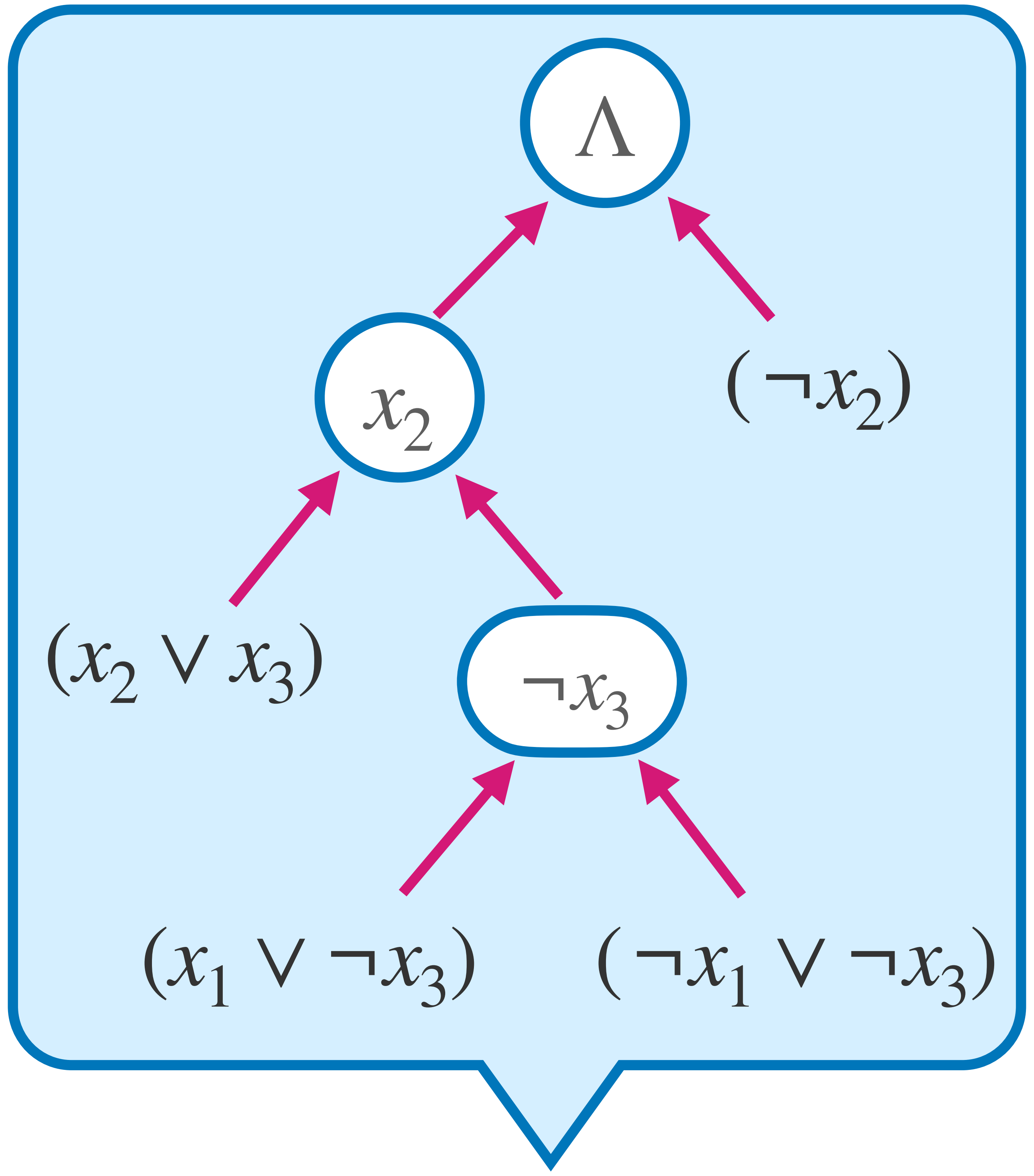
Lower bounds on **size** of tree **Resolution** proofs $\implies$ bounds on **runtime** of **DPLL**!

Execution of DPLL is a **tree Resolution proof** of unsatisfiability

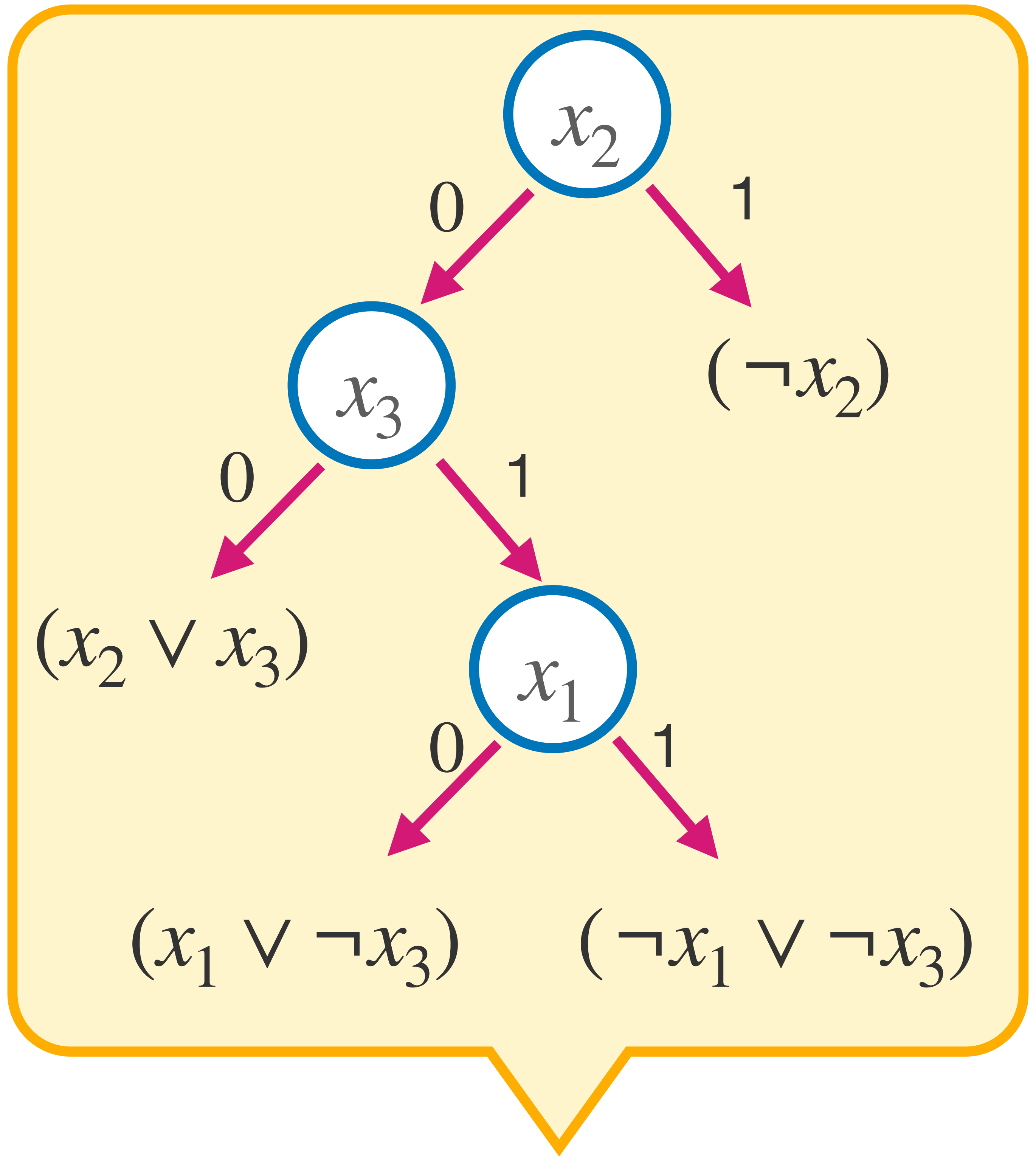
→ Every time we query a variable, resolve on it!

**Upshot**

tree Resolution proofs = DPLL trees




**Tree Resolution**




**DPLL**

# DPLL

Lower bounds on **size** of tree **Resolution** proofs $\implies$ bounds on **runtime** of **DPLL**!

$\to$ **Tons** of lower bounds on tree Resolution known!

$\to$ One of the weakest proof systems!

# DPLL

Lower bounds on **size** of tree **Resolution** proofs $\Longrightarrow$ bounds on **runtime** of **DPLL**!

$\rightarrow$ **Tons** of lower bounds on tree Resolution known!

$\rightarrow$ One of the weakest proof systems!

**Simple Lower bound idea:**

Exploit: Tree resolution cannot recognize redundant parts of the search space

# DPLL

Lower bounds on **size** of tree **Resolution** proofs $\Longrightarrow$ bounds on **runtime** of **DPLL**!

$\rightarrow$ **Tons** of lower bounds on tree Resolution known!

$\rightarrow$ One of the weakest proof systems!

**Simple Lower bound idea:**

Exploit: Tree resolution cannot recognize redundant parts of the search space

1. Find a $F$ such that any proof of $F$ has **a** long path

# DPLL

Lower bounds on **size** of tree **Resolution** proofs $\Longrightarrow$ bounds on **runtime** of **DPLL**!

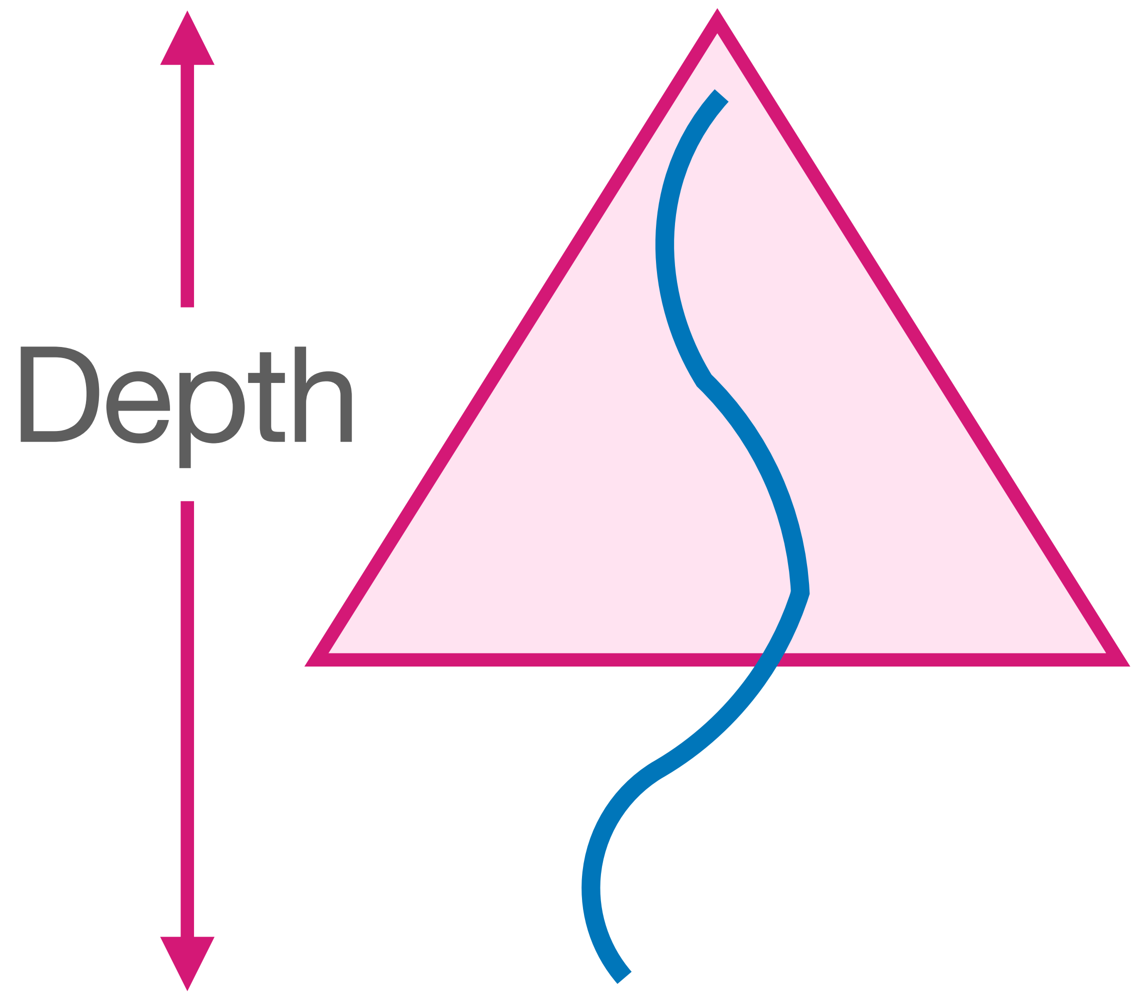
$\rightarrow$ **Tons** of lower bounds on tree Resolution known!

$\rightarrow$ One of the weakest proof systems!

**Simple Lower bound idea:**

Exploit: Tree resolution cannot recognize redundant parts of the search space

1. Find a $F$ such that any proof of $F$ has **a** long path



# DPLL

Lower bounds on **size** of tree **Resolution** proofs $\Longrightarrow$ bounds on **runtime** of **DPLL**!

$\rightarrow$ **Tons** of lower bounds on tree Resolution known!

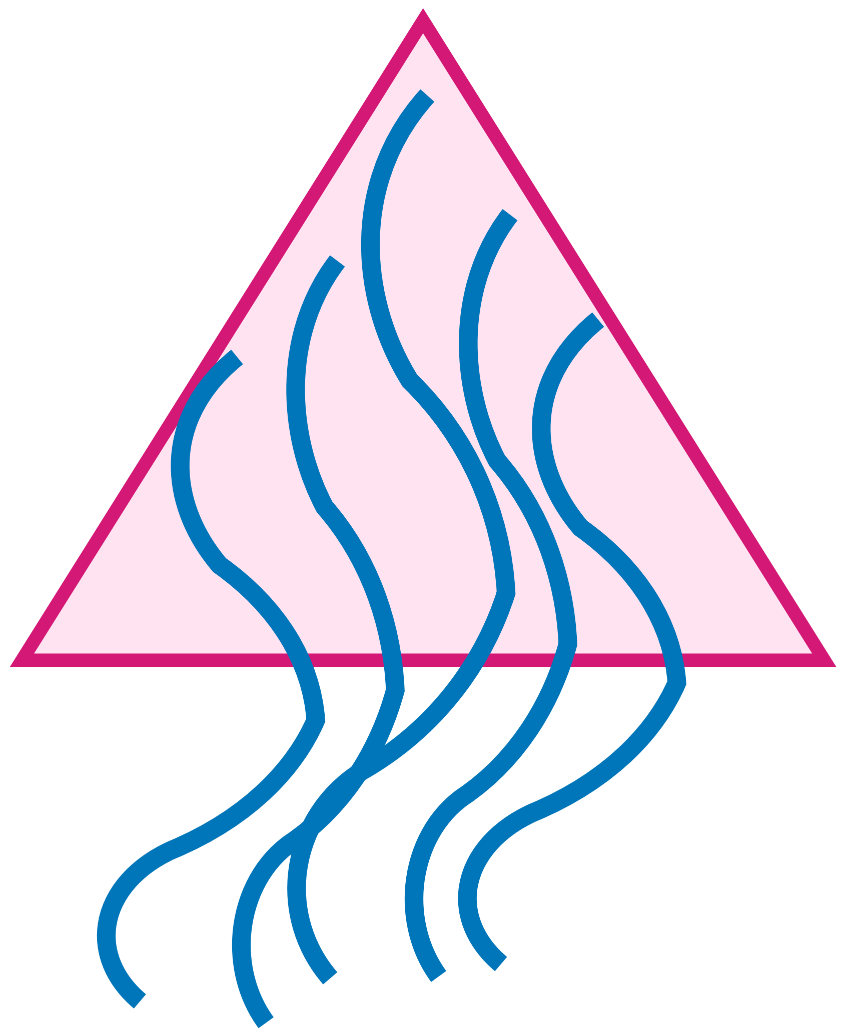
$\rightarrow$ One of the weakest proof systems!

**Simple Lower bound idea:**

Exploit: Tree resolution cannot recognize redundant parts of the search space

1. Find a $F$ such that any proof of $F$ has **a** long path

2. Then $F \circ XOR_2$ must have many long paths



# DPLL

Lower bounds on **size** of tree **Resolution** proofs $\Longrightarrow$ bounds on **runtime** of **DPLL**!

$\rightarrow$ **Tons** of lower bounds on tree Resolution known!

$\rightarrow$ One of the weakest proof systems!

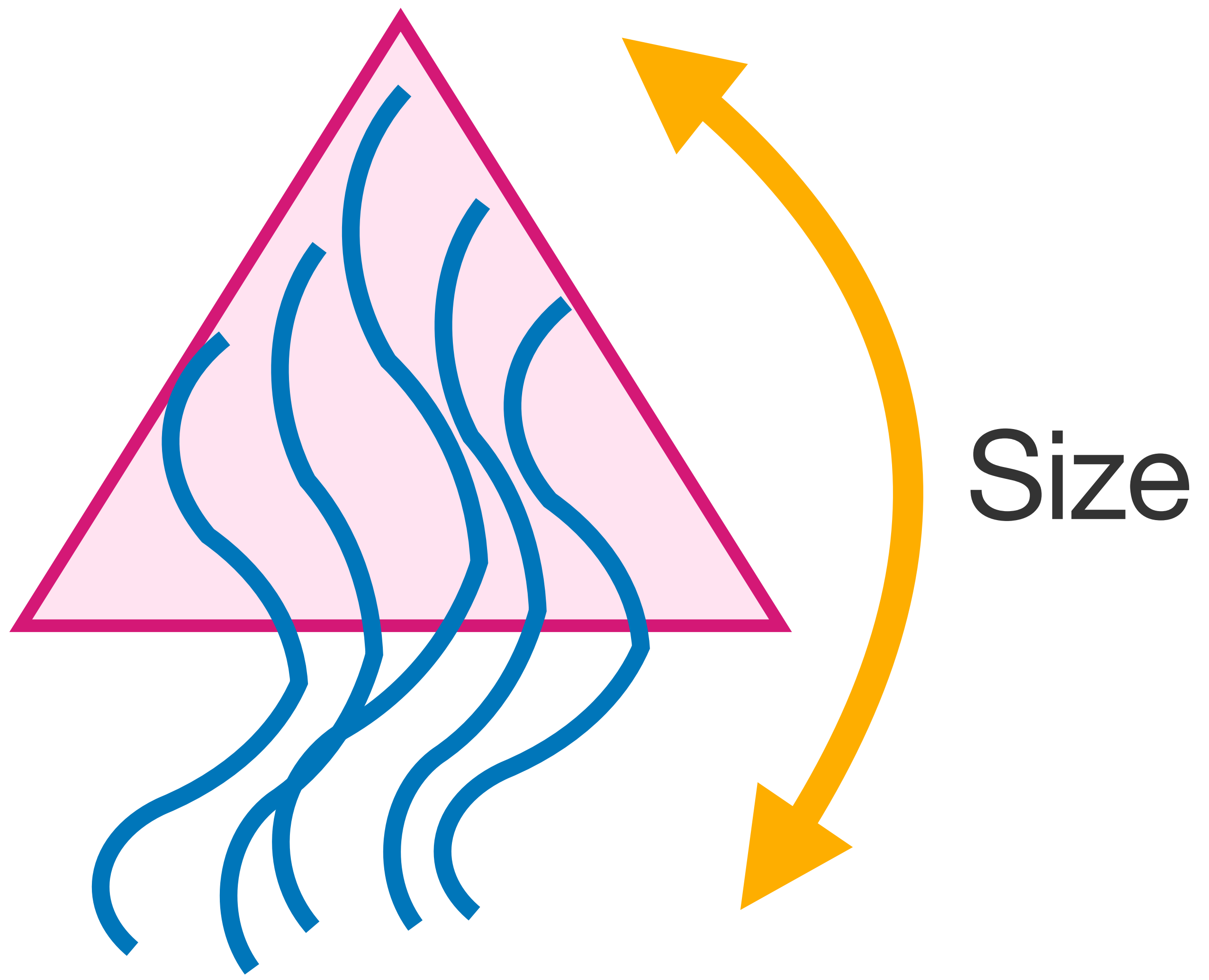
**Simple Lower bound idea:**

Exploit: Tree resolution cannot recognize redundant parts of the search space

1. Find a $F$ such that any proof of $F$ has **a** long path

2. Then $F \circ XOR_2$ must have many long paths

   Theorem: $size_{tRes}(F \circ XOR_2) \geq 2^{depth_{tRes}(F)/2}$



# Conflict-Driven Clause Learning

Modern (CDCL) SAT Solvers build on DPLL

# Conflict-Driven Clause Learning

Modern (CDCL) SAT Solvers build on DPLL

→ Multiple subroutines built to avoid getting stuck in bad areas of the search space

# Conflict-Driven Clause Learning

Modern (CDCL) SAT Solvers build on DPLL

→ Multiple subroutines built to avoid getting stuck in bad areas of the search space

We will develop CDCL in stages by extending DPLL with the following:

- Unit Propagation
- Clause Learning
- Restarts

# Unit Propagation

Speeds up search

**Unit clause:** a clause containing a **single** literal $\ell$

# Unit Propagation

Speeds up search

**Unit clause:** a clause containing a **single** literal $\ell$

**Unit Propagation:** if $F$ contains a unit clause

(under the current assignment), set $\ell = 1$

# Unit Propagation

**DPLL with unit prop**

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z})$$

Speeds up search

**Unit clause:** a clause containing a **single** literal $\ell$

**Unit Propagation:** if $F$ contains a unit clause (under the current assignment), set $\ell = 1$

# Unit Propagation

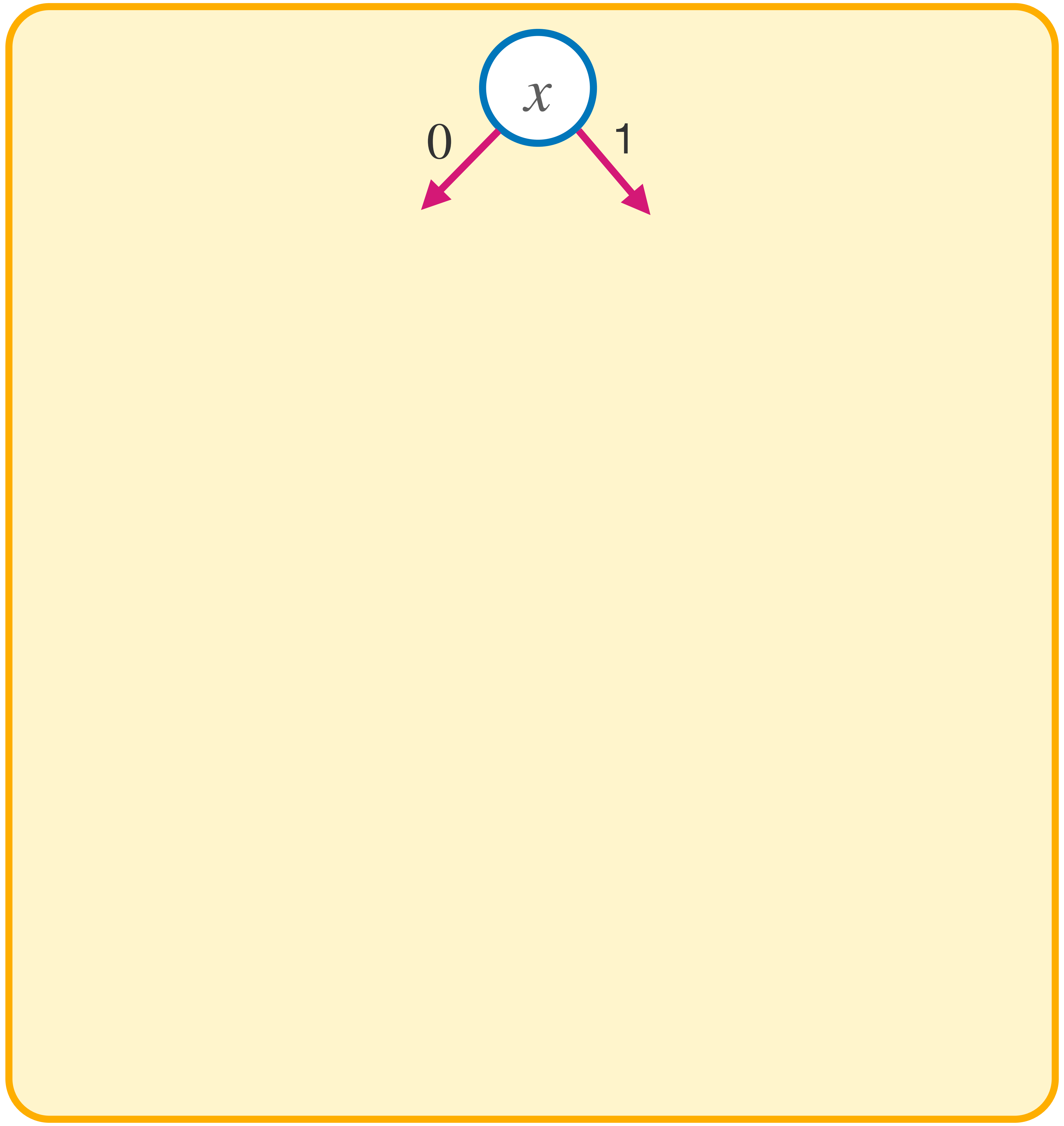
## DPLL with unit prop

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z})$$

Speeds up search

**Unit clause:** a clause containing a **single** literal $\ell$

**Unit Propagation:** if $F$ contains a unit clause (under the current assignment), set $\ell = 1$



# Unit Propagation

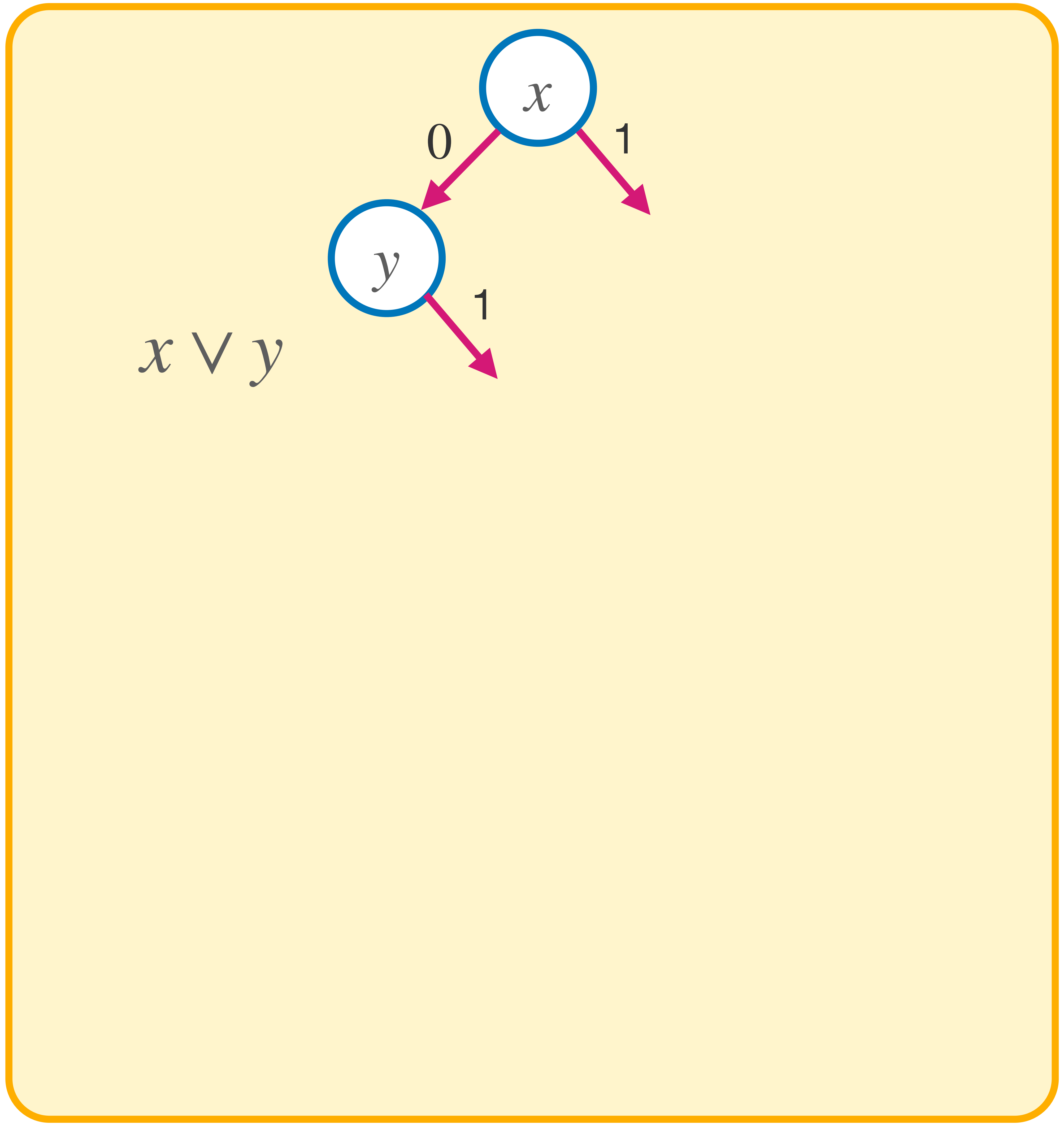
**DPLL with unit prop**

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z})$$

Speeds up search

**Unit clause:** a clause containing a **single** literal $\ell$

**Unit Propagation:** if $F$ contains a unit clause (under the current assignment), set $\ell = 1$



# Unit Propagation

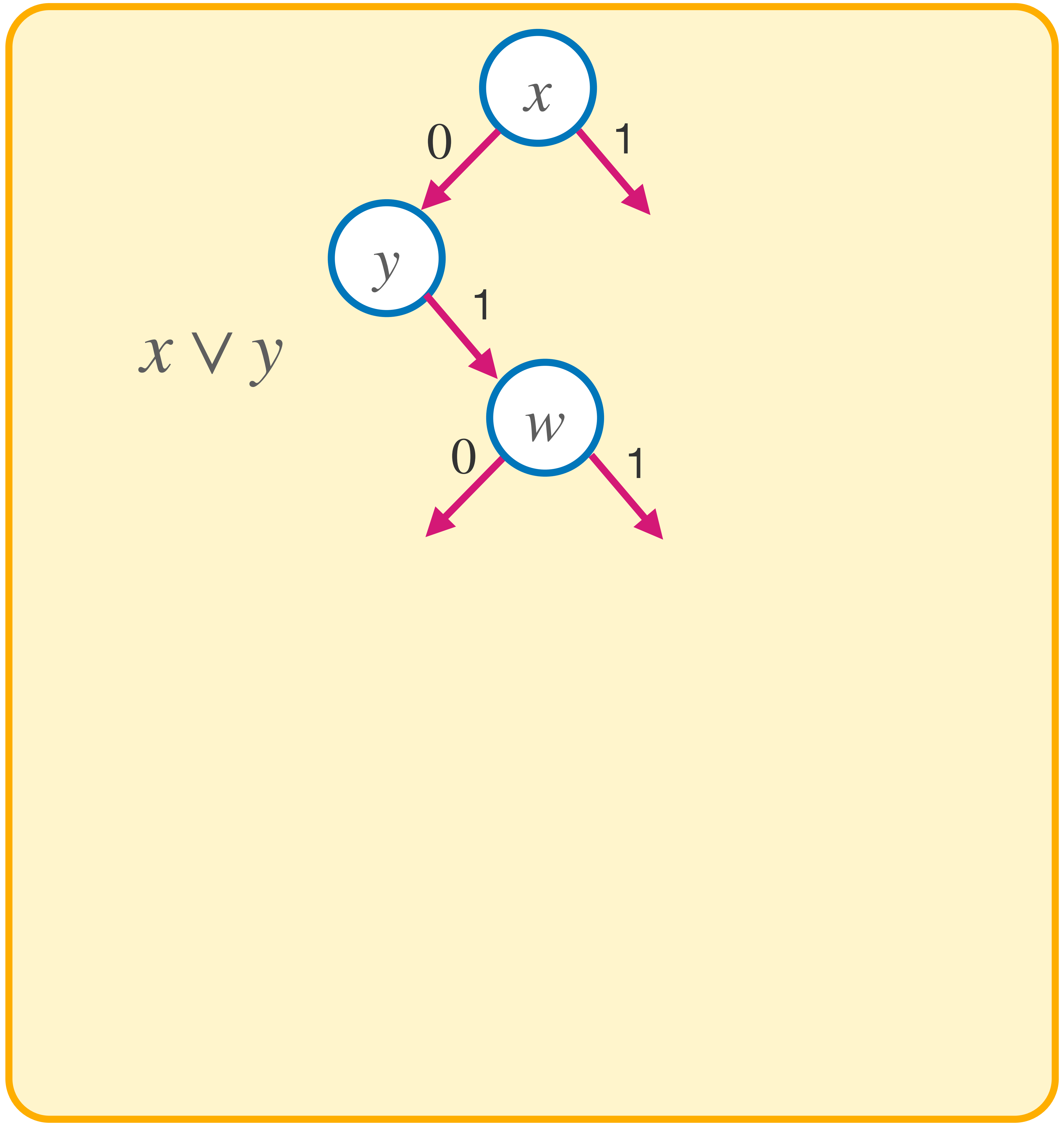
Speeds up search

**Unit clause:** a clause containing a **single** literal $\ell$

**Unit Propagation:** if $F$ contains a unit clause (under the current assignment), set $\ell = 1$

**DPLL with unit prop**

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z})$$



# Unit Propagation

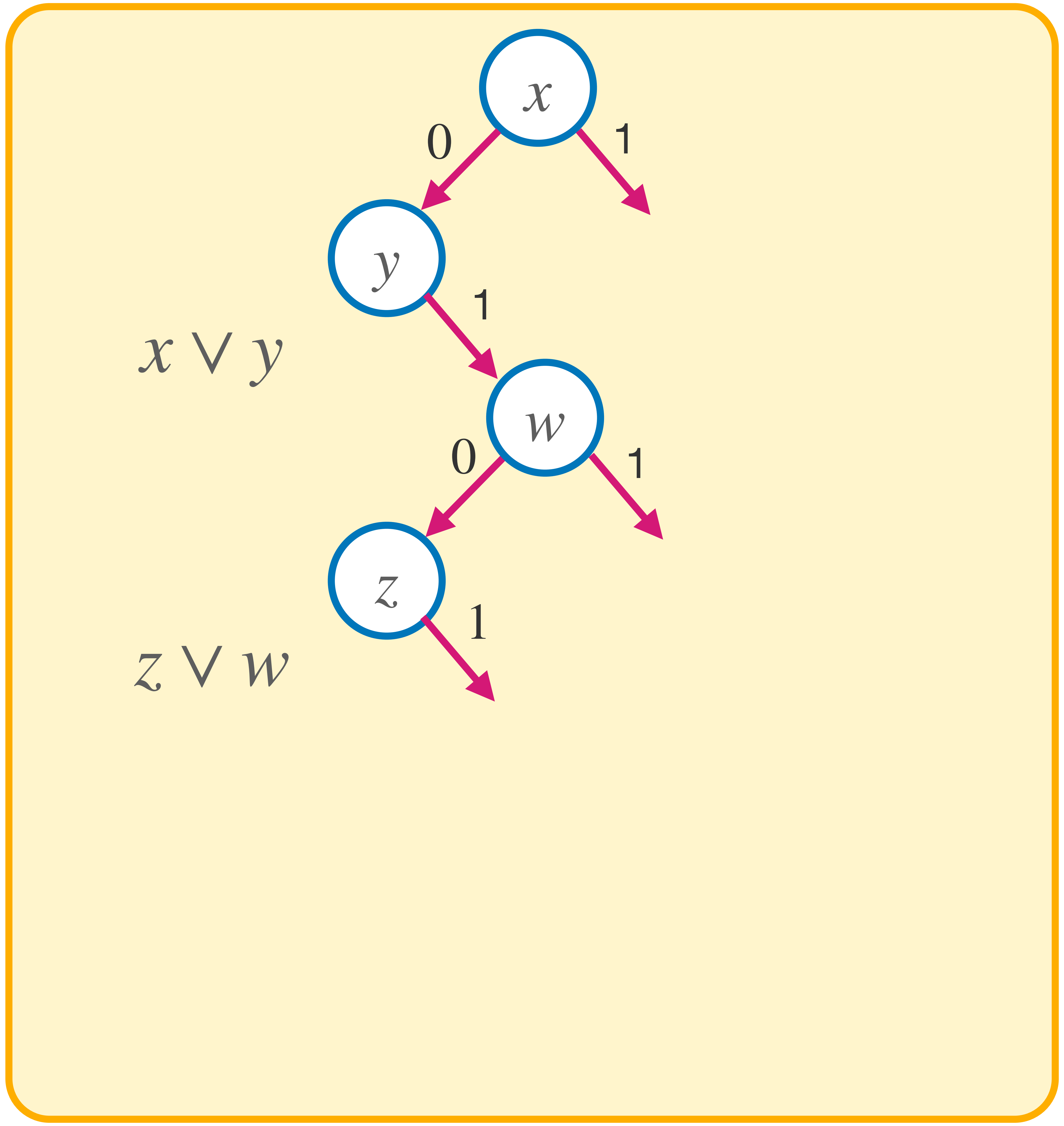
Speeds up search

**Unit clause:** a clause containing a **single** literal $\ell$

**Unit Propagation:** if $F$ contains a unit clause (under the current assignment), set $\ell = 1$

**DPLL with unit prop**

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z})$$



# Unit Propagation

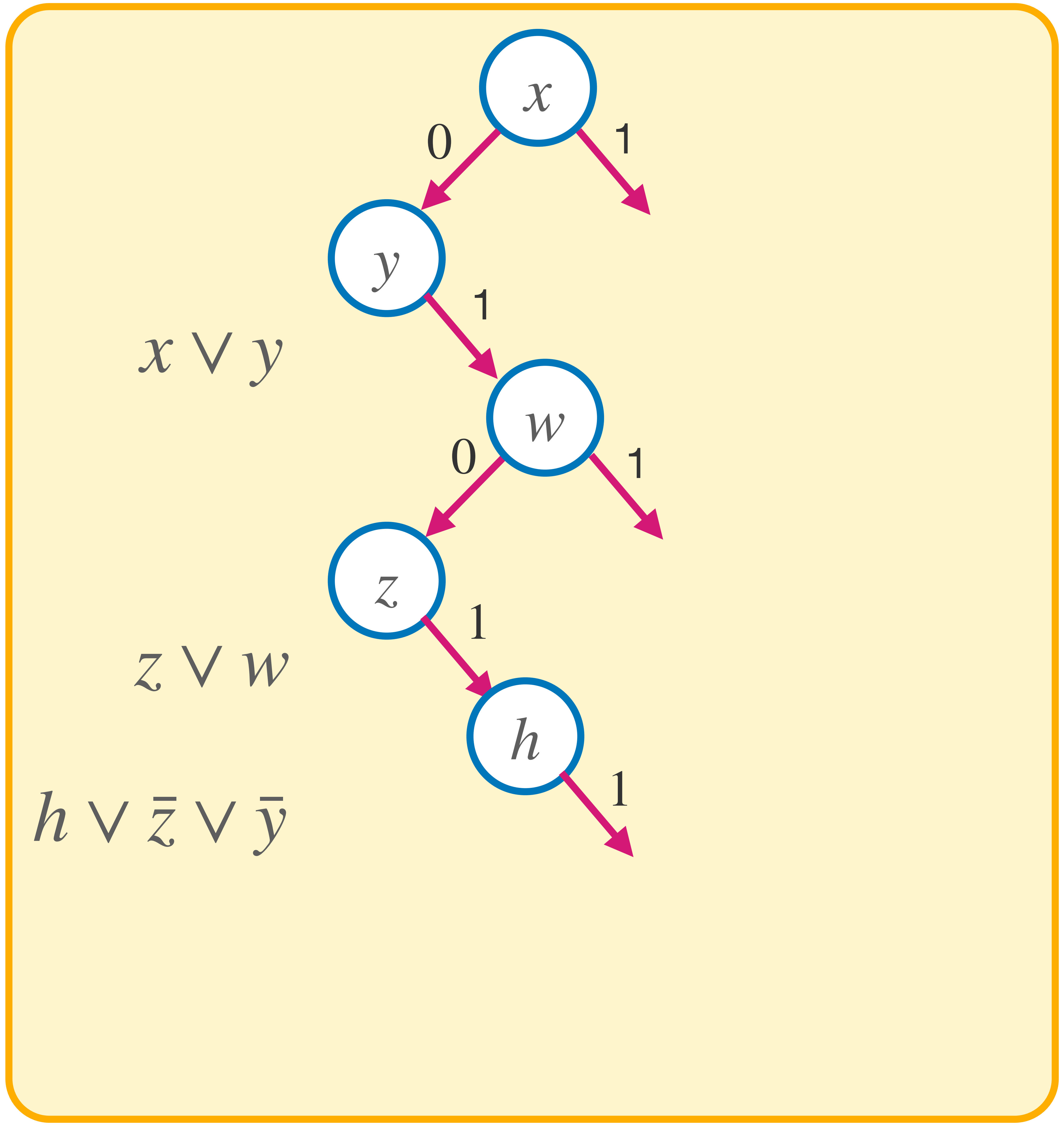
**DPLL with unit prop**

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z})$$

Speeds up search

**Unit clause:** a clause containing a **single** literal $\ell$

**Unit Propagation:** if $F$ contains a unit clause (under the current assignment), set $\ell = 1$



# Unit Propagation

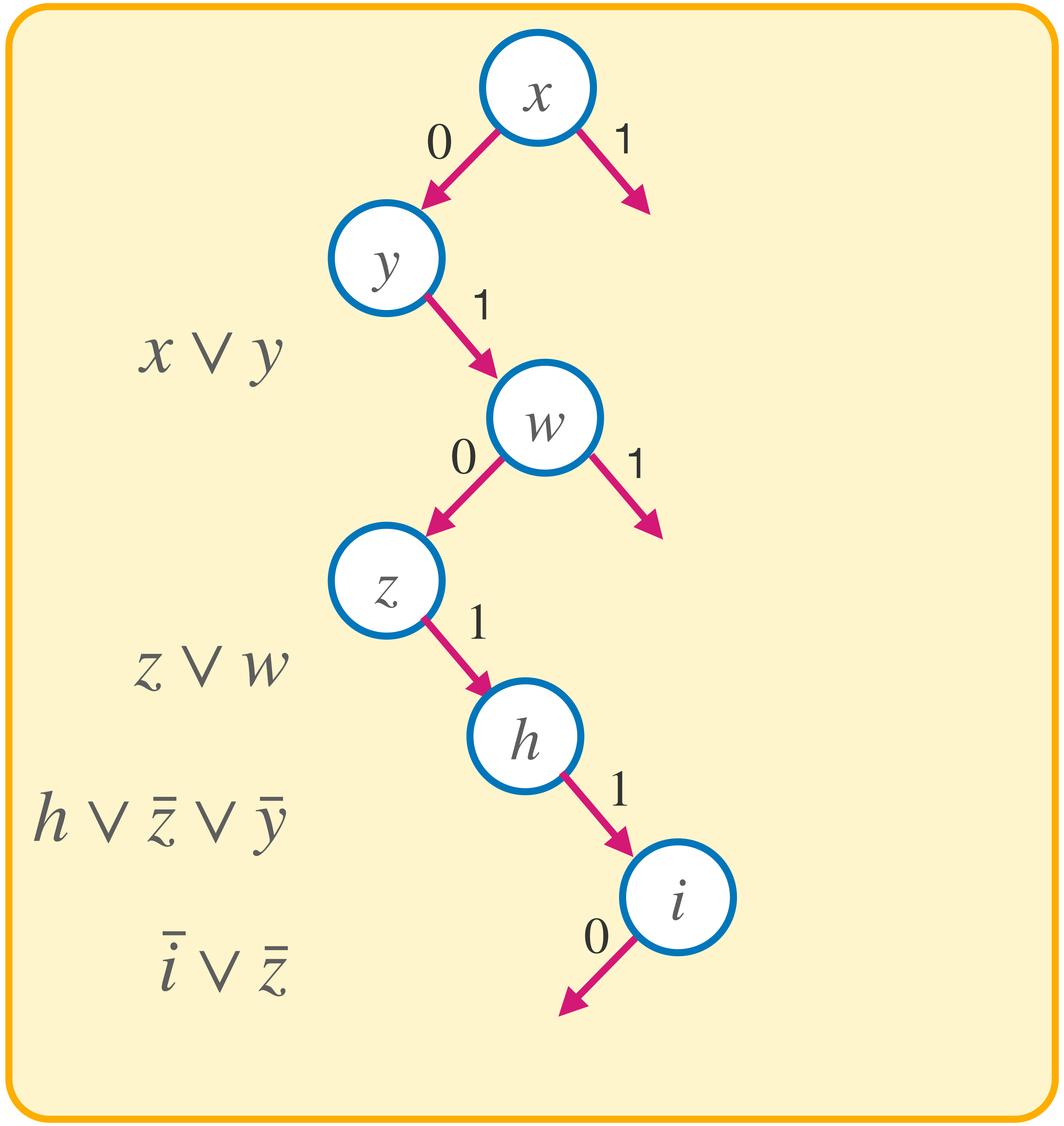
Speeds up search

**Unit clause:** a clause containing a **single** literal $\ell$

**Unit Propagation:** if $F$ contains a unit clause (under the current assignment), set $\ell = 1$

**DPLL with unit prop**

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z})$$



# Unit Propagation

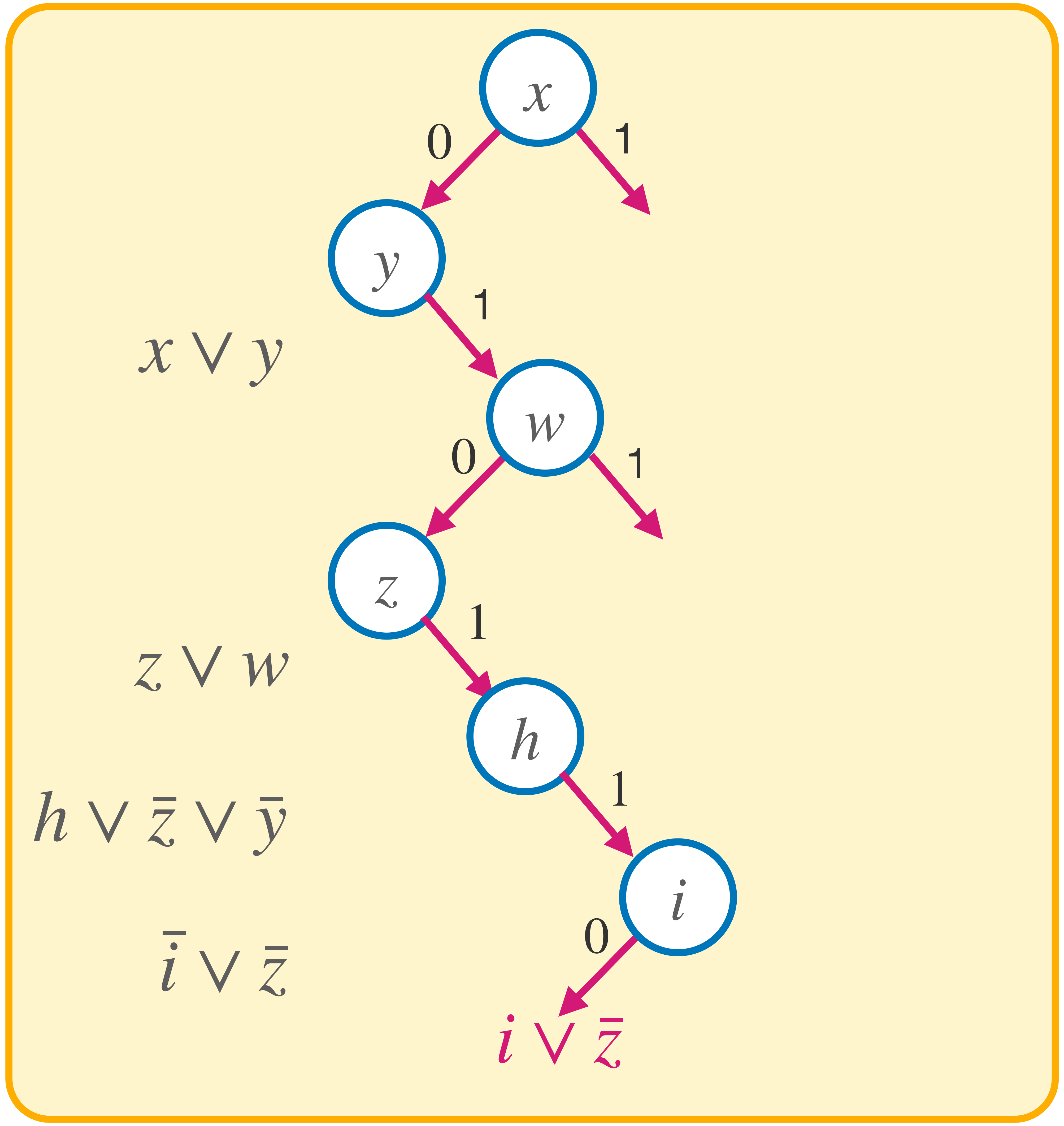
**DPLL with unit prop**

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z})$$

Speeds up search

**Unit clause:** a clause containing a **single** literal $\ell$

**Unit Propagation:** if $F$ contains a unit clause (under the current assignment), set $\ell = 1$



# Unit Propagation

**DPLL with unit prop**

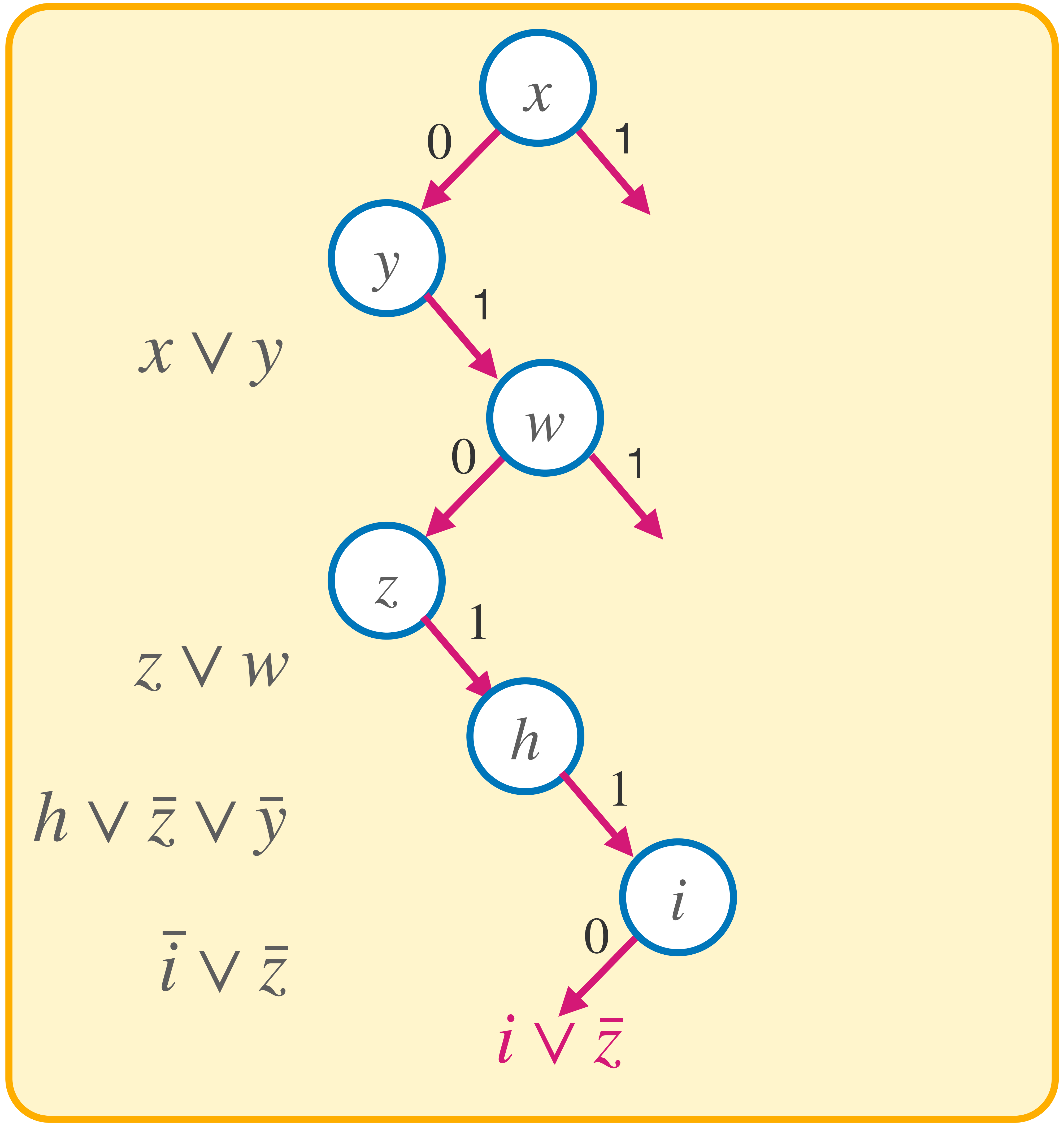
$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z})$$

Speeds up search

**Unit clause:** a clause containing a **single** literal $\ell$

**Unit Propagation:** if $F$ contains a unit clause (under the current assignment), set $\ell = 1$

**Decision Level:** A literal set by a decision together with all unit propagated literals constitutes a decision level.



# Unit Propagation

Speeds up search

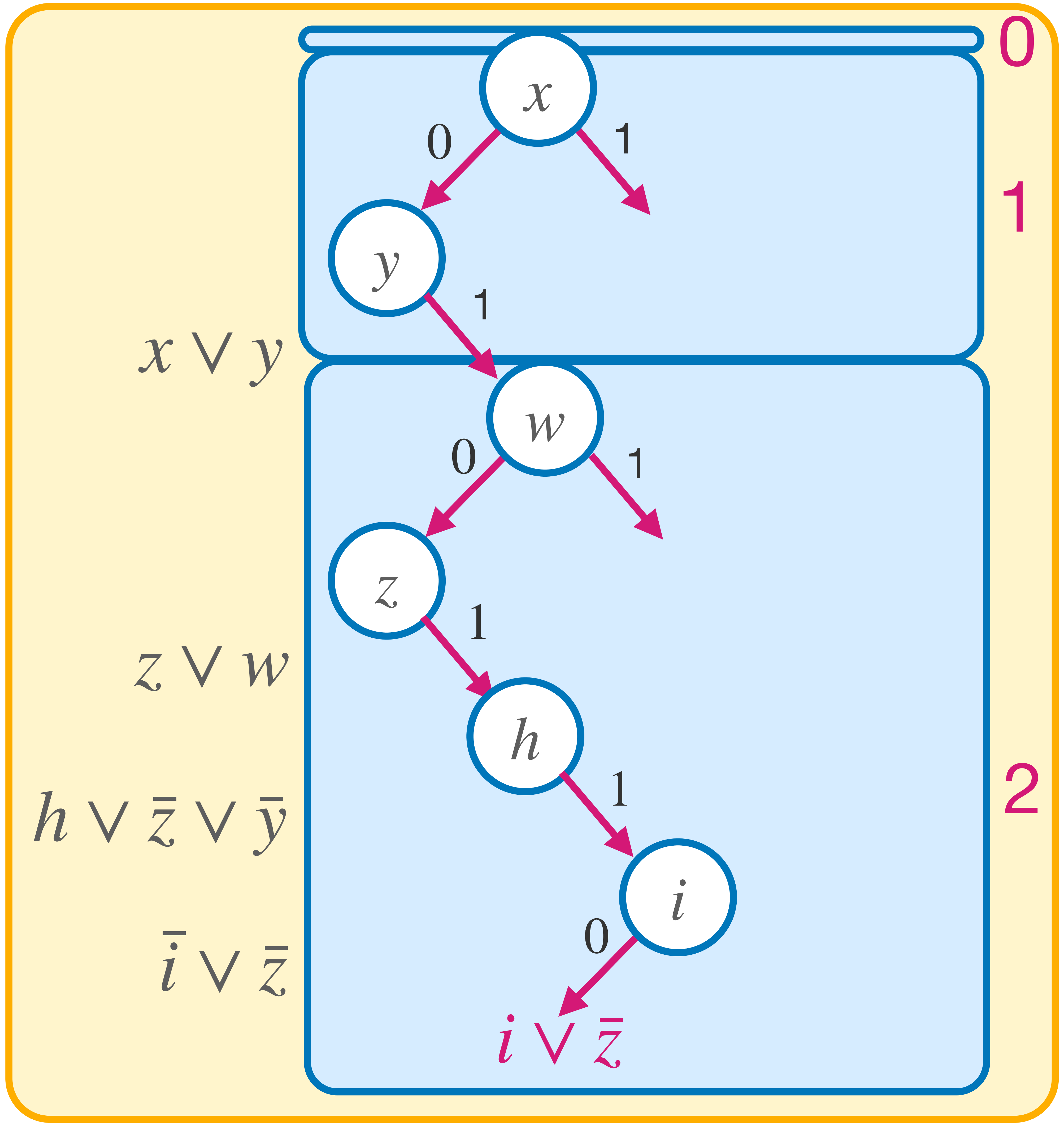
**Unit clause:** a clause containing a **single** literal $\ell$

**Unit Propagation:** if $F$ contains a unit clause (under the current assignment), set $\ell = 1$

**Decision Level:** A literal set by a decision together with all unit propagated literals constitutes a decision level.

**DPLL with unit prop**

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z})$$



# Conflict-Driven Clause Learning

Modern (CDCL) SAT Solvers build on DPLL

→ Multiple subroutines built to avoid getting stuck in bad areas of the search space

We will develop CDCL in stages by extending DPLL with the following:

- Unit Propagation ✓
- Clause Learning
- Restarts

# Clause Learning

The main improvement over DPLL

# Clause Learning

The main improvement over DPLL

When a conflict occurs learn a new clause (add it to $F$) which helps to avoid similar conflicts in the future

# Clause Learning

The main improvement over DPLL

When a conflict occurs learn a new clause (add it to $F$) which helps to avoid similar conflicts in the future

**Want:**

# Clause Learning

The main improvement over DPLL

When a conflict occurs learn a new clause (add it to $F$) which helps to avoid similar conflicts in the future

**Want:**

- The learned clause is a sound inference from $F$

# Clause Learning

The main improvement over DPLL

When a conflict occurs learn a new clause (add it to $F$ ) which helps to avoid similar conflicts in the future

**Want:**

- The learned clause is a sound inference from $F$
- The learned clause causes many unit propagations

# Clause Learning

The main improvement over DPLL

When a conflict occurs learn a new clause (add it to $F$) which helps to avoid similar conflicts in the future

**Want:**

- The learned clause is a sound inference from $F$
- The learned clause causes many unit propagations

$Q$. How can we achieve this?

# Clause Learning

The main improvement over DPLL

When a conflict occurs learn a new clause (add it to $F$ ) which helps to avoid similar conflicts in the future

**Want:**

- The learned clause is a sound inference from $F$
- The learned clause causes many unit propagations

$Q$ . How can we achieve this? **Resolution!**

# Clause Learning

Use Resolution to learn new clauses

Any variable in the conflict clause that was unit propagated along the path can be resolved with the clause that caused that unit propagation!

# Clause Learning

Use Resolution to learn new clauses

Any variable in the conflict clause that was unit propagated along the path can be resolved with the clause that caused that unit propagation!

$\Longrightarrow$ Generates a new sound clause for $F$!

# Clause Learning

Use Resolution to learn new clauses

Any variable in the conflict clause that was unit propagated along the path can be resolved with the clause that caused that unit propagation!

$\Longrightarrow$ Generates a new sound clause for $F$!

Can derive new clauses by resolving up the path

# Clause Learning

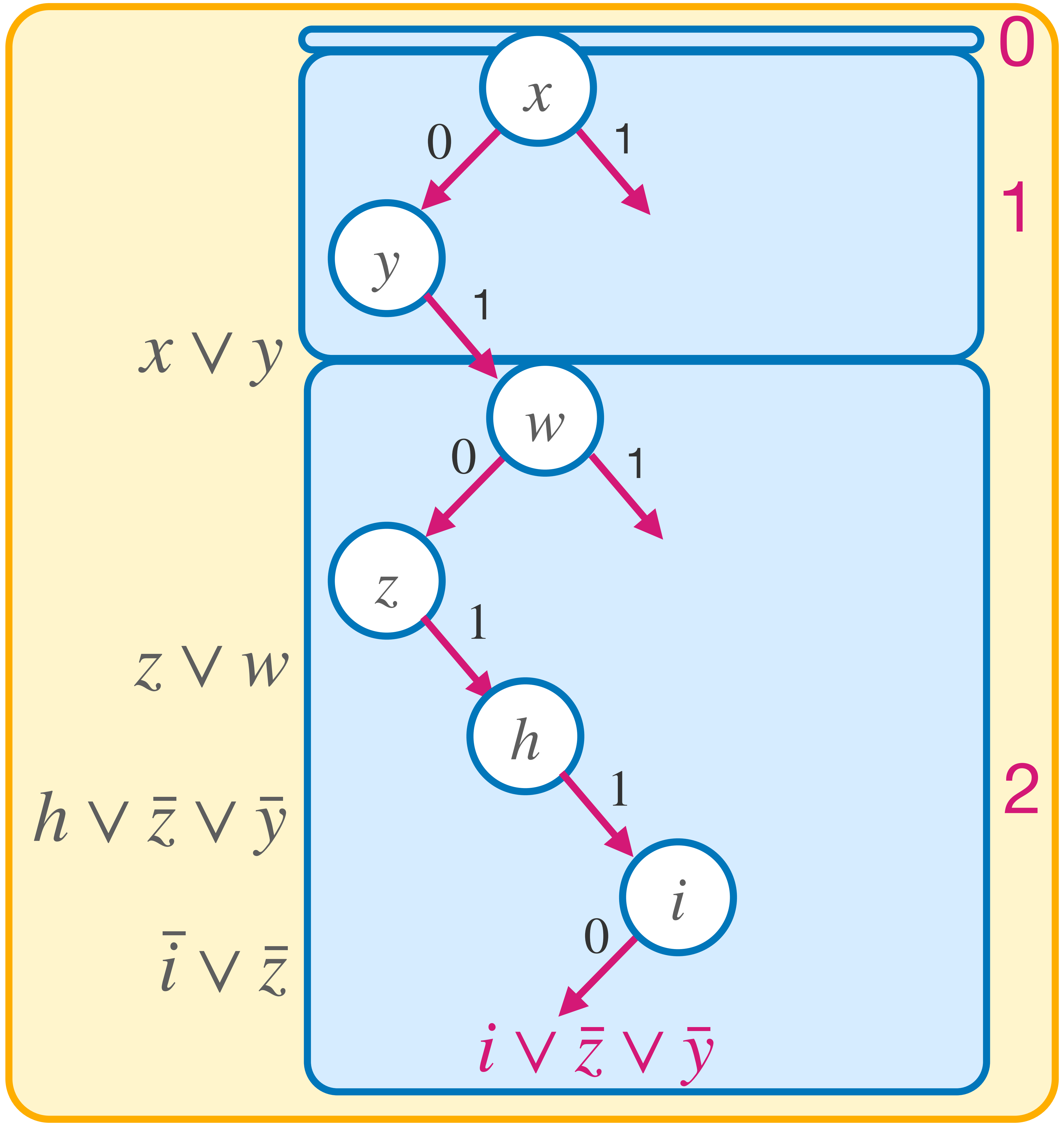
$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$$

Use Resolution to learn new clauses

Any variable in the conflict clause that was unit propagated along the path can be resolved with the clause that caused that unit propagation!

$\Longrightarrow$ Generates a new sound clause for $F$!

Can derive new clauses by resolving up the path



# Clause Learning

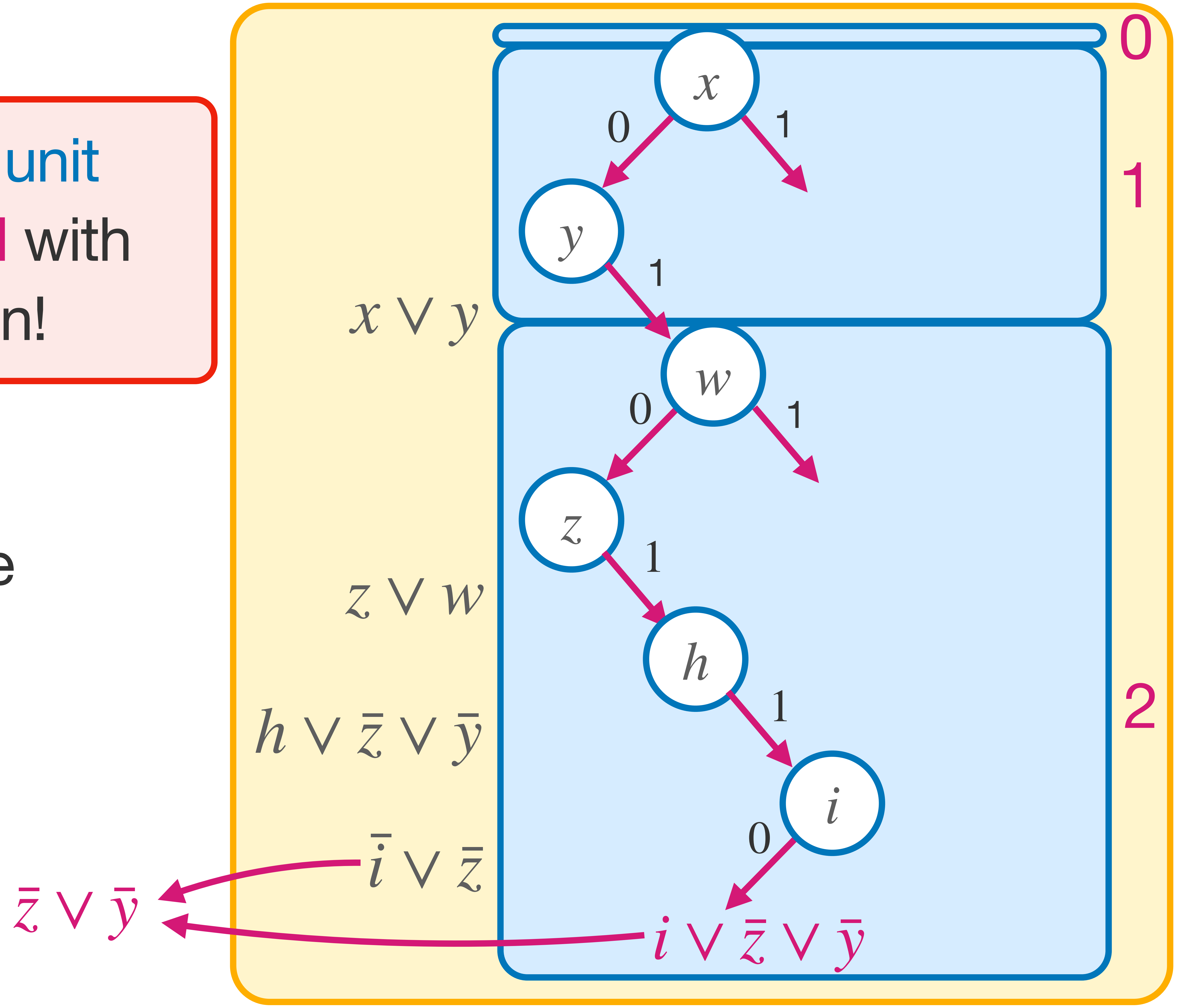
$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$$

Use Resolution to learn new clauses

Any variable in the conflict clause that was unit propagated along the path can be resolved with the clause that caused that unit propagation!

$\Longrightarrow$ Generates a new sound clause for $F$!

Can derive new clauses by resolving up the path



# Clause Learning

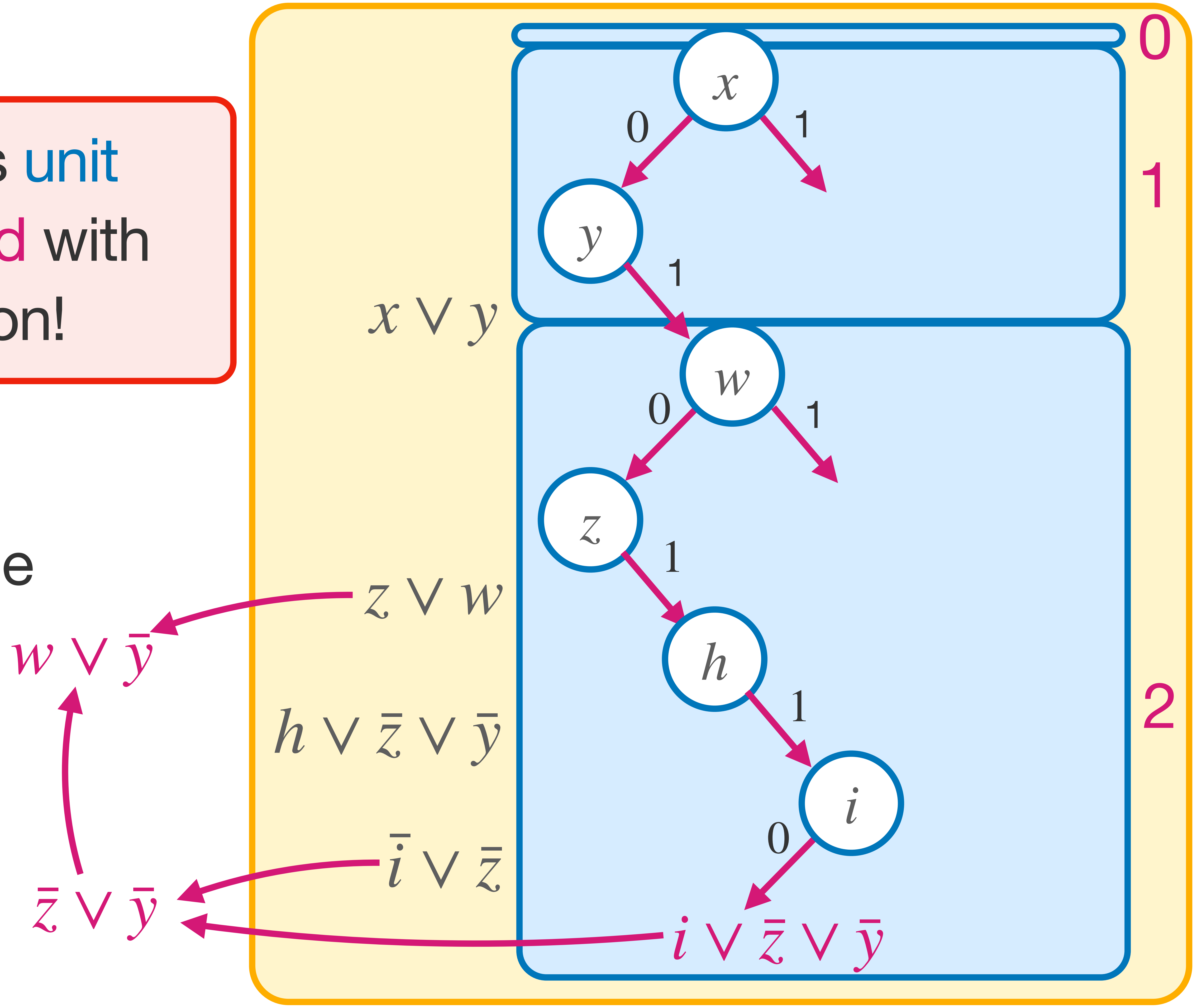
$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$$

Use Resolution to learn new clauses

Any variable in the conflict clause that was unit propagated along the path can be resolved with the clause that caused that unit propagation!

$\Longrightarrow$ Generates a new sound clause for $F$!

Can derive new clauses by resolving up the path



# Clause Learning

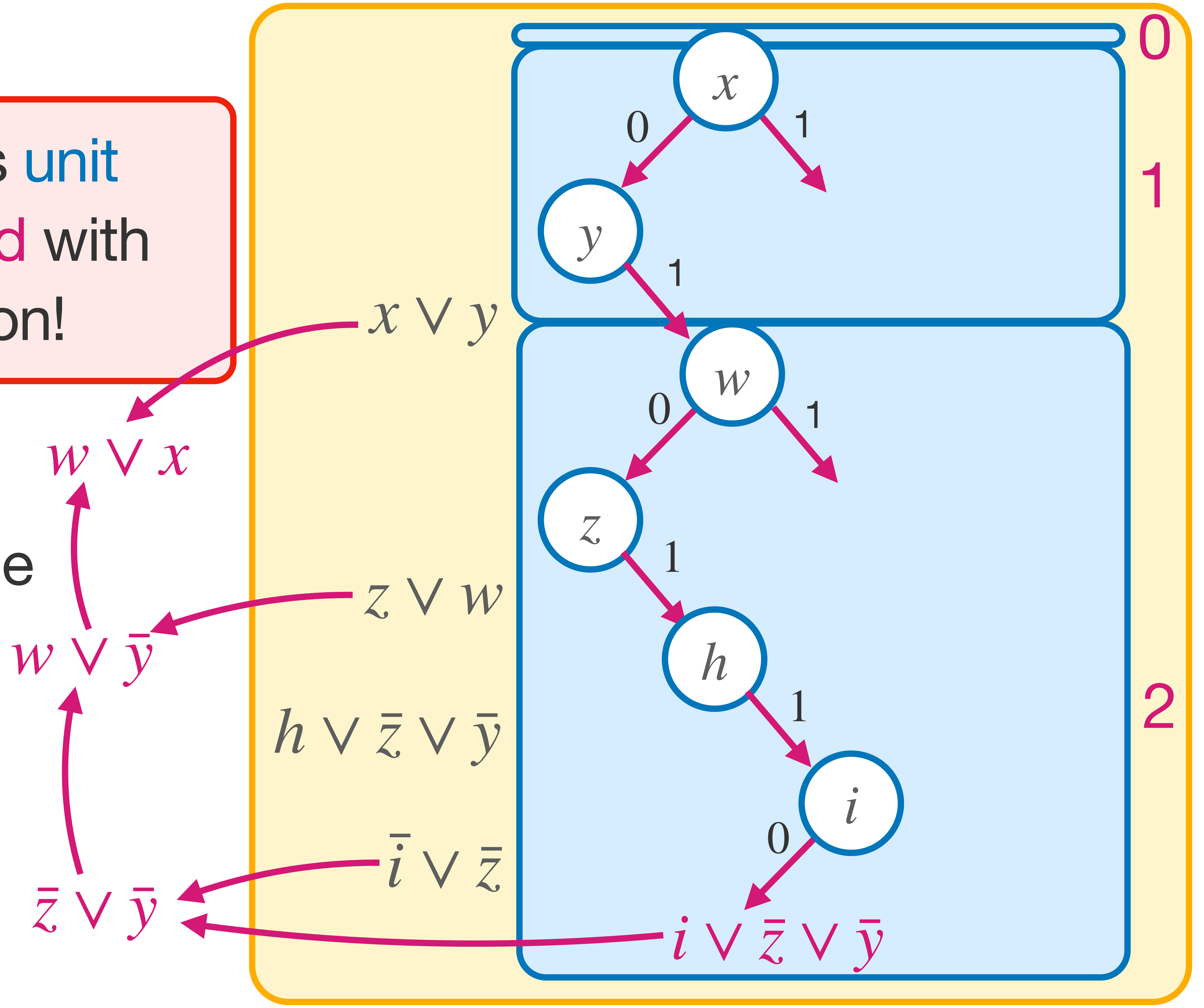
$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$$

Use Resolution to learn new clauses

Any variable in the conflict clause that was unit propagated along the path can be resolved with the clause that caused that unit propagation!

$\Longrightarrow$ Generates a new sound clause for $F$!

Can derive new clauses by resolving up the path



# Clause Learning

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$$

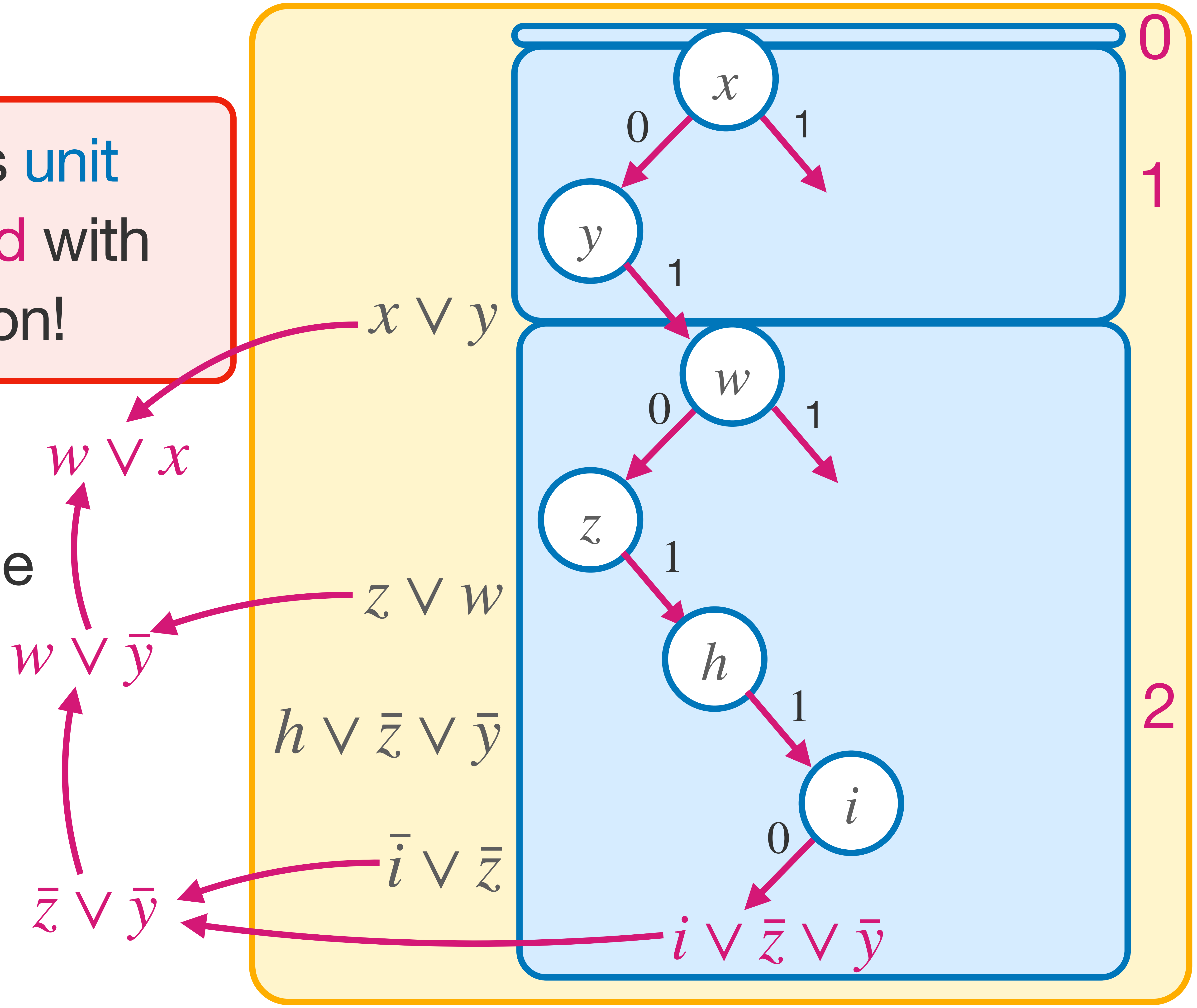
Use Resolution to learn new clauses

Any variable in the conflict clause that was unit propagated along the path can be resolved with the clause that caused that unit propagation!

$\Longrightarrow$ Generates a new sound clause for $F$!

Can derive new clauses by resolving up the path

$Q$. When should we stop?

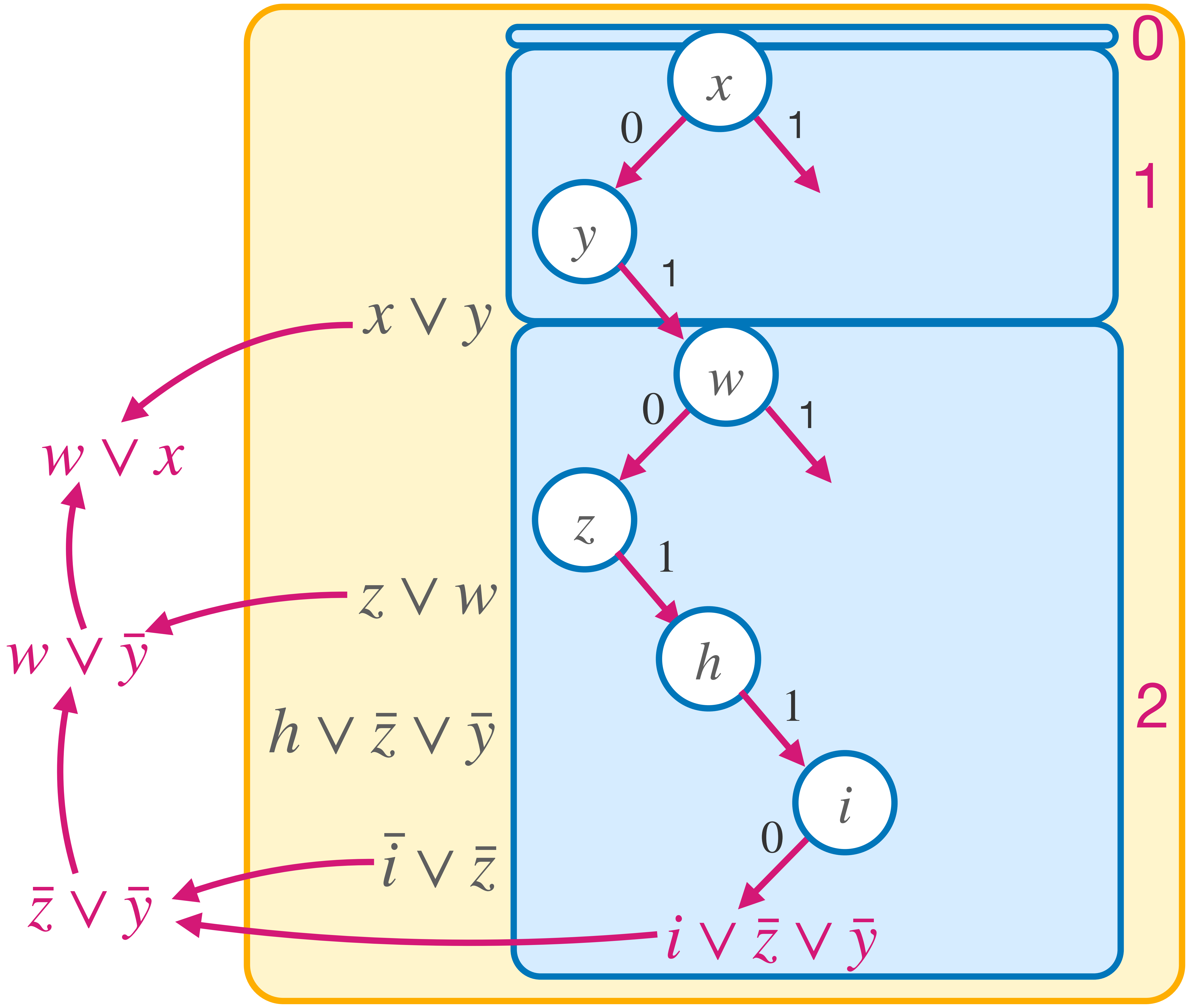


# Clause Learning

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$$

Use Resolution to learn new clauses

$Q$. When should we stop?



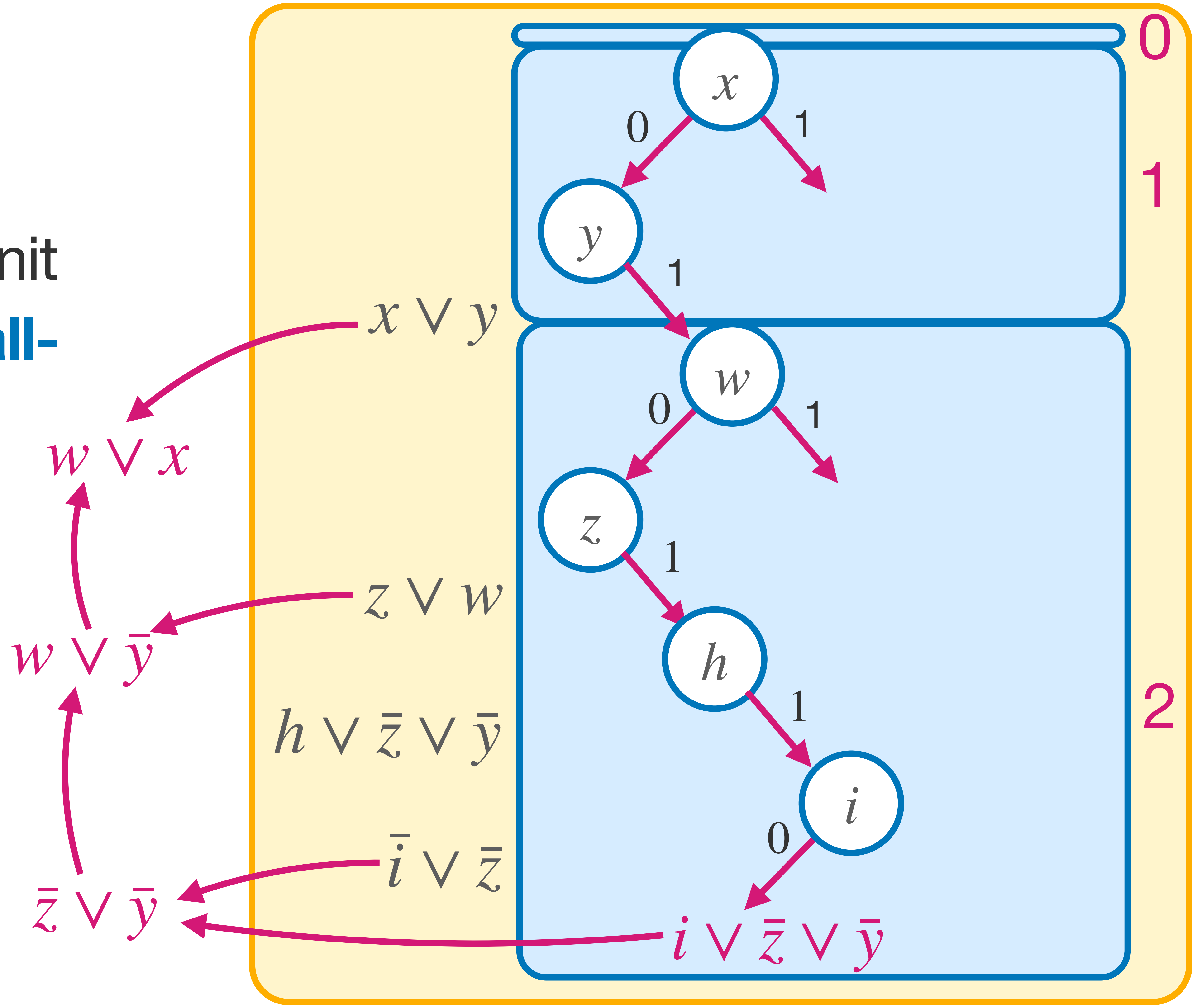
# Clause Learning

$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$

Use Resolution to learn new clauses

$Q$. When should we stop?

If we resolved until all literals which were unit propagated are resolved away we get an **all-decision** clause



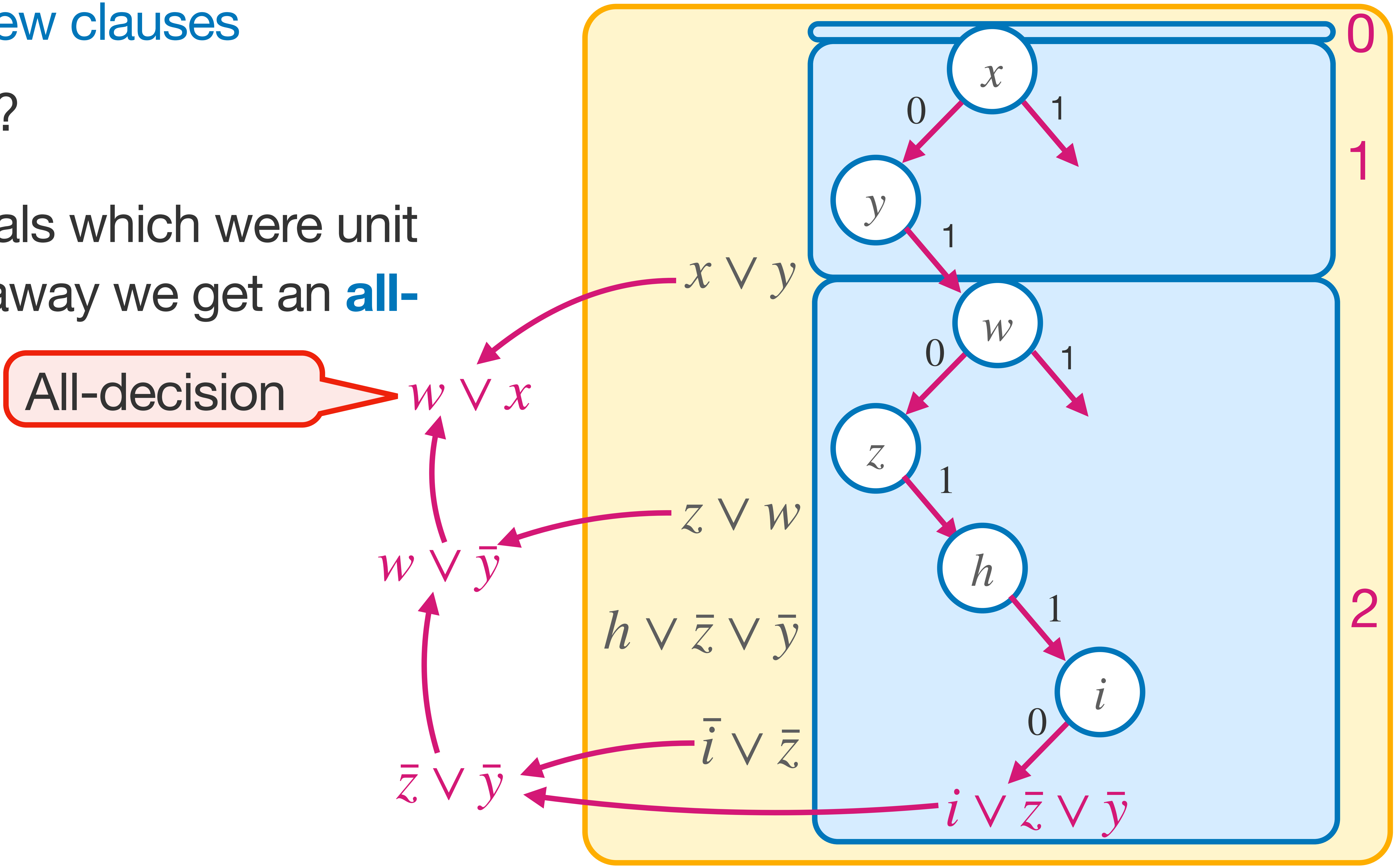
# Clause Learning

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$$

Use Resolution to learn new clauses

$Q$. When should we stop?

If we resolved until all literals which were unit propagated are resolved away we get an **all-decision** clause



# Clause Learning

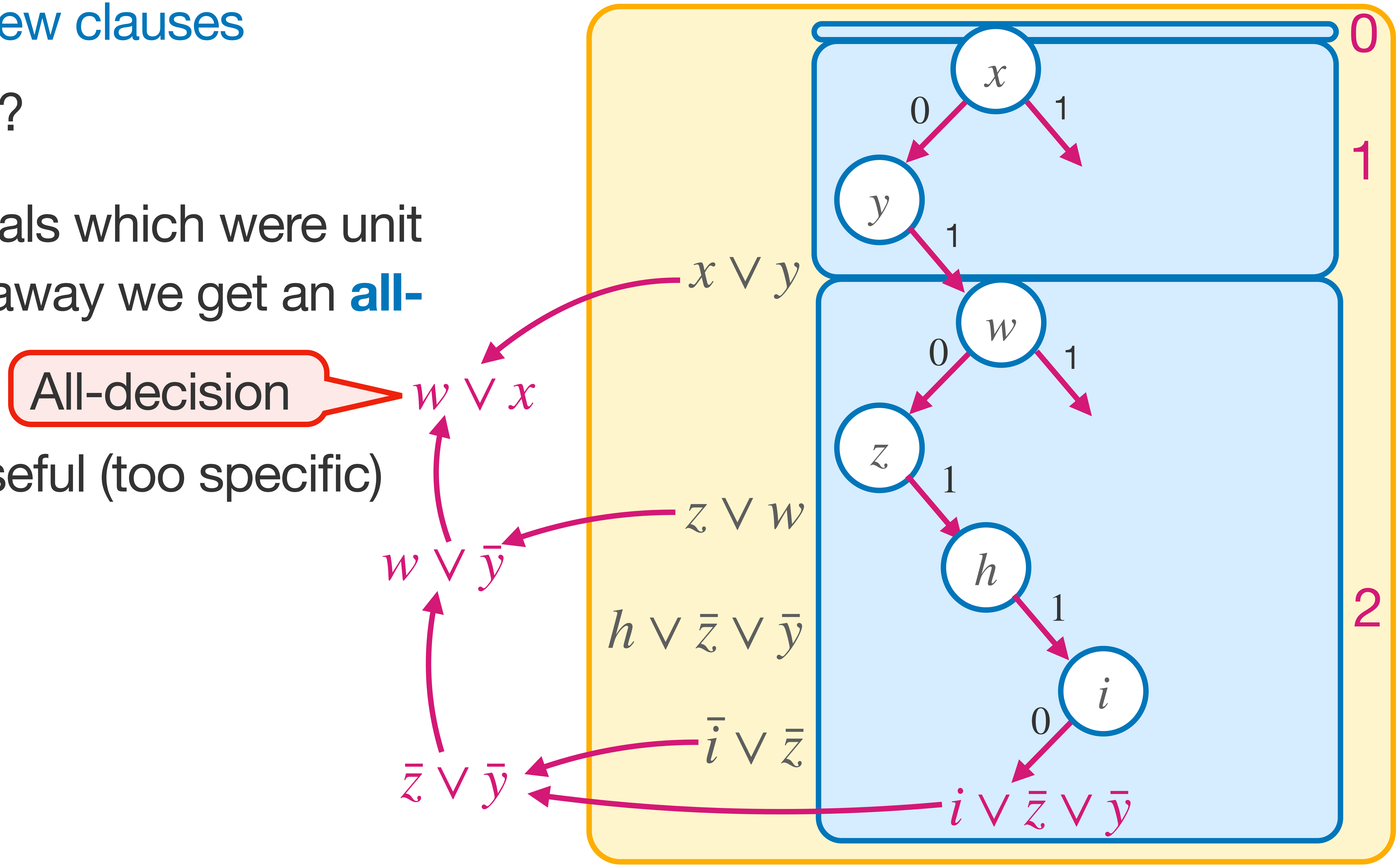
$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$$

Use Resolution to learn new clauses

$Q$. When should we stop?

If we resolved until all literals which were unit propagated are resolved away we get an **all-decision** clause

→ Empirically not very useful (too specific)



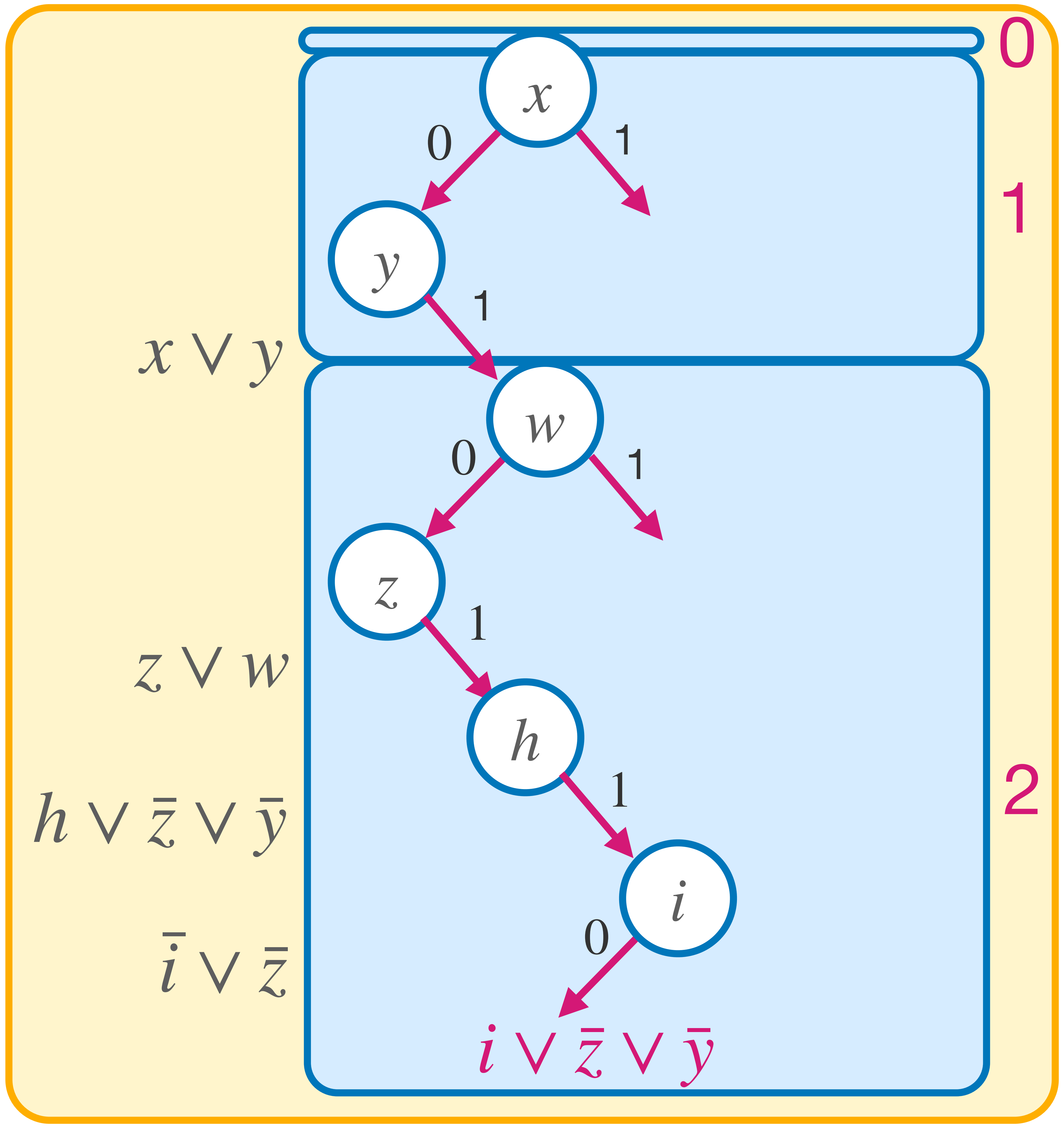
# Clause Learning

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$$

Use Resolution to learn new clauses

$Q$. When should we stop?

Standard clause to learn is a **1-UIP clause**



# Clause Learning

$$(x \lor y) \land (z \lor w) \land (h \lor \bar{z} \lor \bar{y}) \land (\bar{i} \lor \bar{z}) \land (i \lor \bar{z} \lor \bar{y})$$

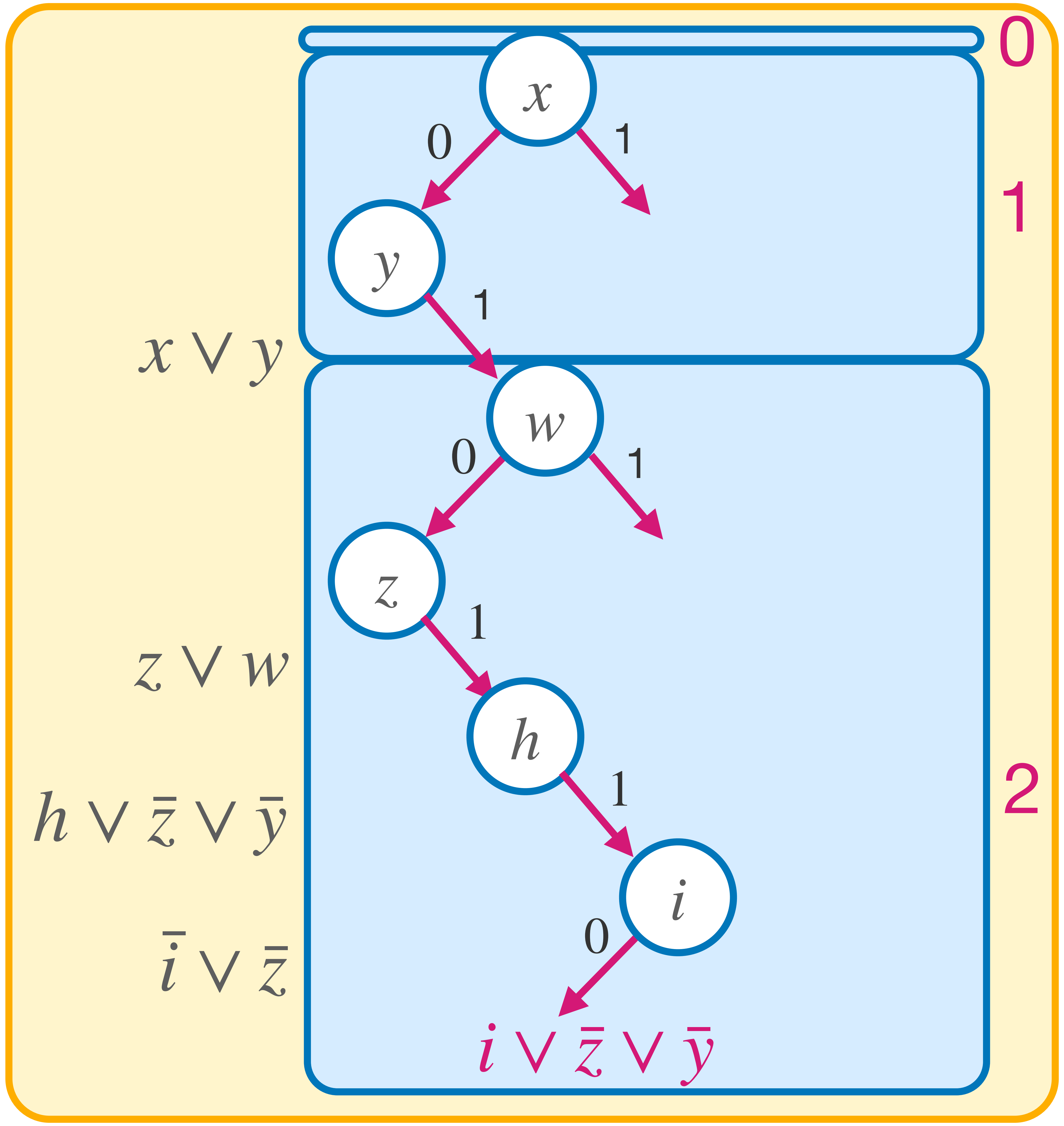
Use Resolution to learn new clauses

$Q$. When should we stop?

Standard clause to learn is a **1-UIP clause**

**1-UIP Clause**

Obtained by resolving the conflict clause along the path until there is only one literal in the clause at the largest decision level



# Clause Learning

$$(x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$$

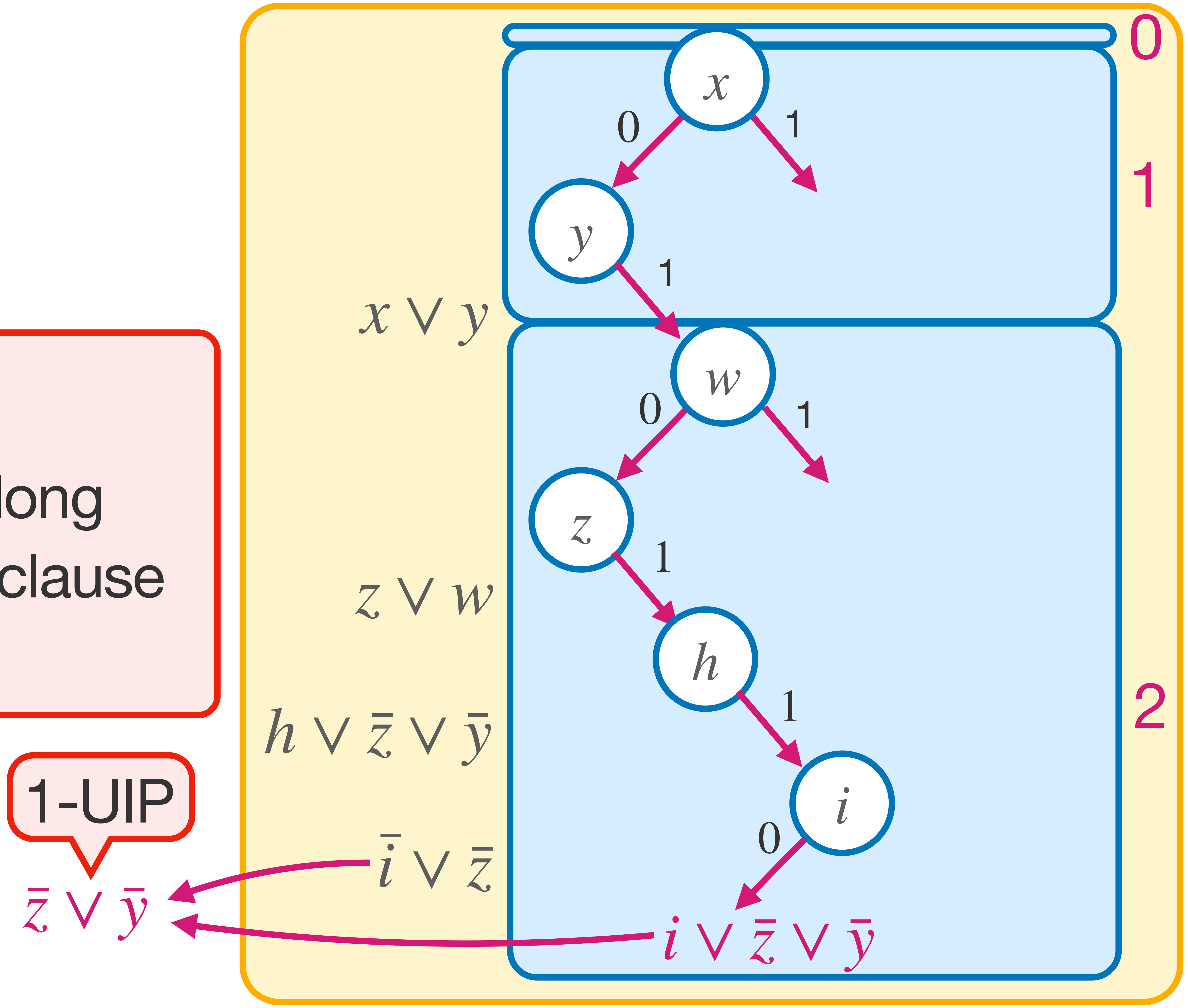
Use Resolution to learn new clauses

$Q$. When should we stop?

Standard clause to learn is a **1-UIP clause**

**1-UIP Clause**

Obtained by resolving the conflict clause along the path until there is only one literal in the clause at the largest decision level



# Clause Learning

$$(\bar{z} \vee \bar{y}) \wedge (x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$$

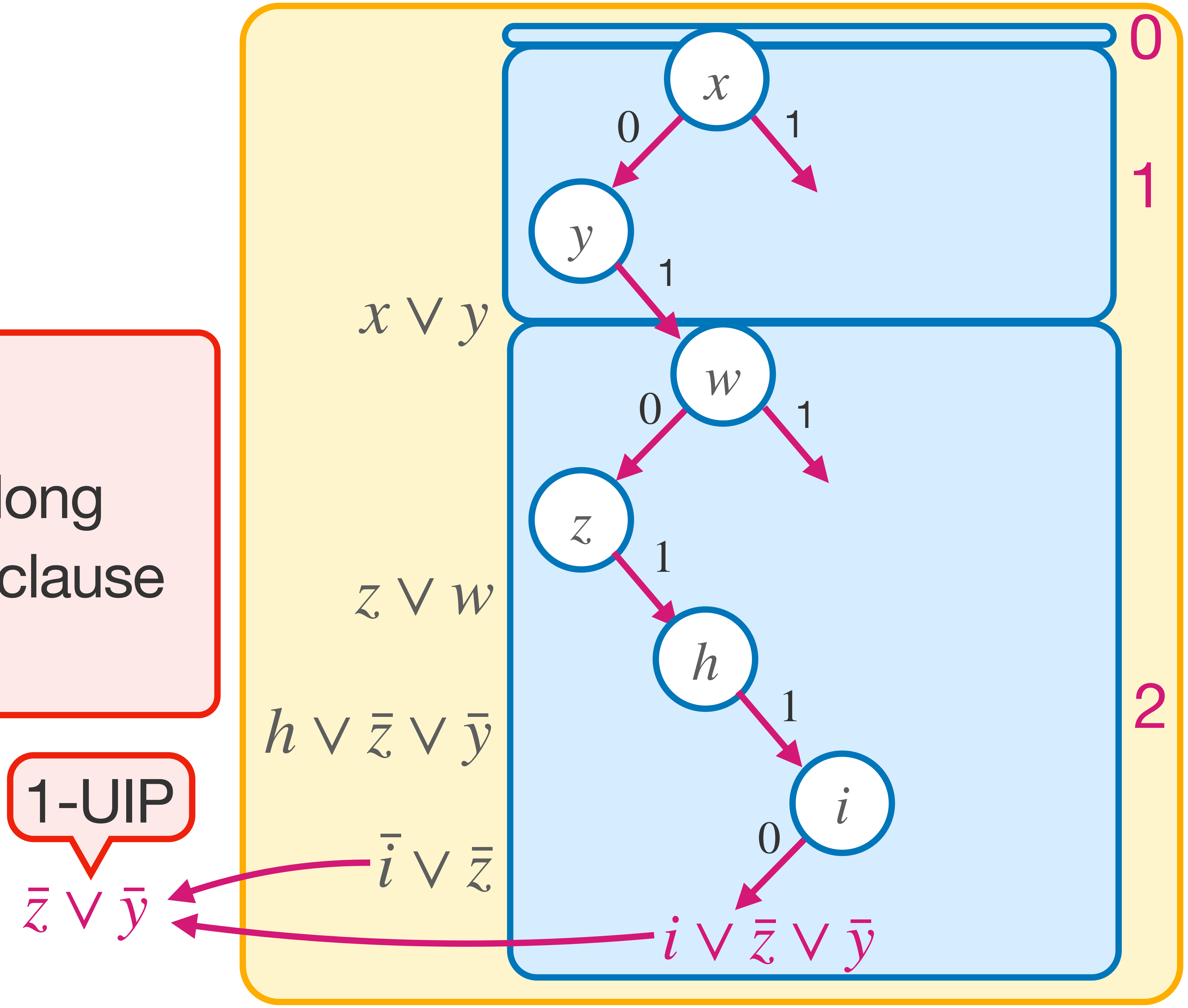
Use Resolution to learn new clauses

$Q$. When should we stop?

Standard clause to learn is a **1-UIP clause**

**1-UIP Clause**

Obtained by resolving the conflict clause along the path until there is only one literal in the clause at the largest decision level



# Clause Learning

$(\bar{z} \vee \bar{y}) \wedge (x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$

Use Resolution to learn new clauses

$Q$. When should we stop?

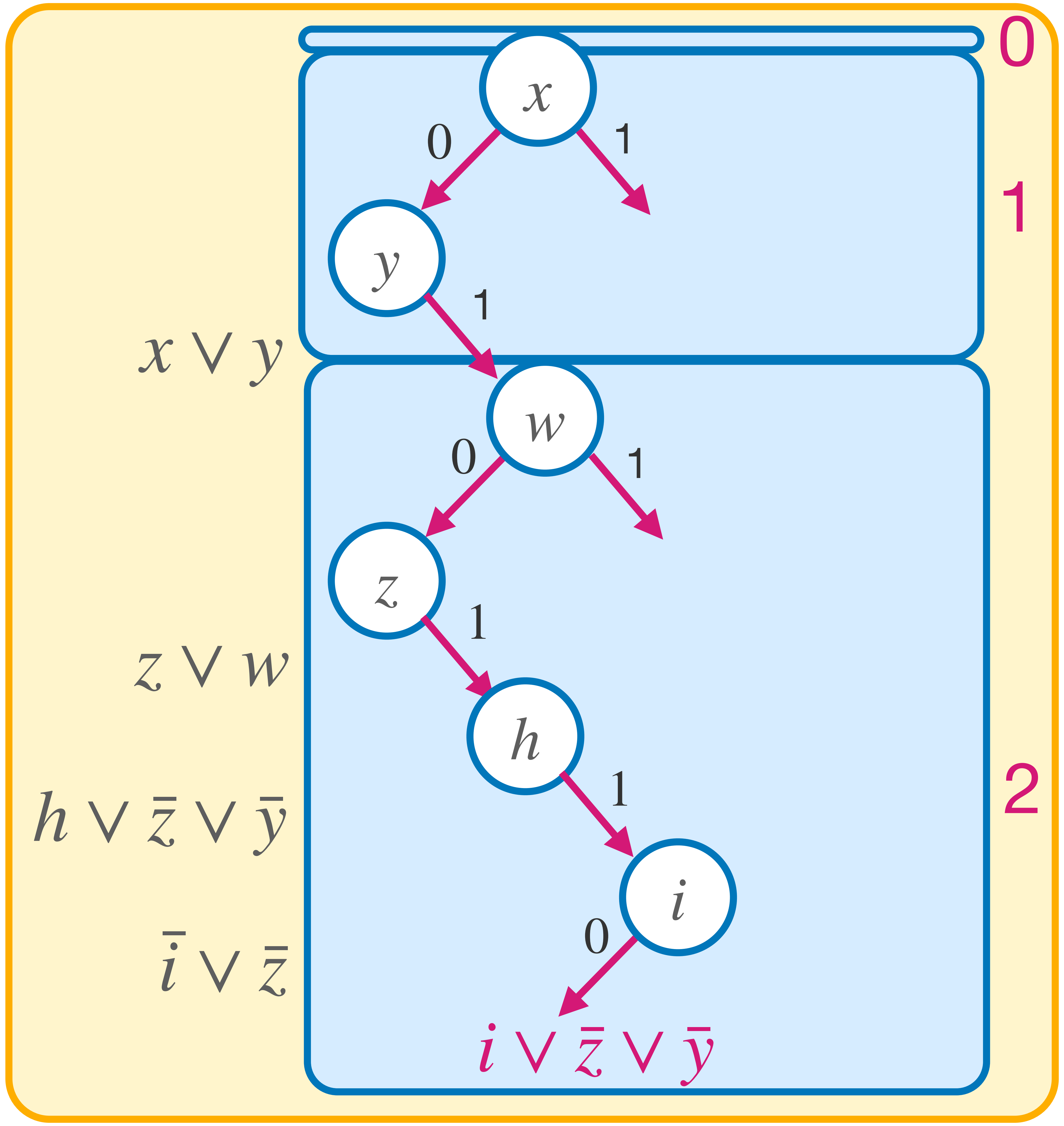
Standard clause to learn is a **1-UIP clause**

**1-UIP Clause**

Obtained by resolving the conflict clause along the path until there is only one literal in the clause at the largest decision level

**Backtracking with 1-UIP:**

Remove everything up to the second largest decision level in the learned clause



# Clause Learning

2 1

$(\bar{z} \vee \bar{y}) \wedge (x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$

Use Resolution to learn new clauses

$Q$. When should we stop?

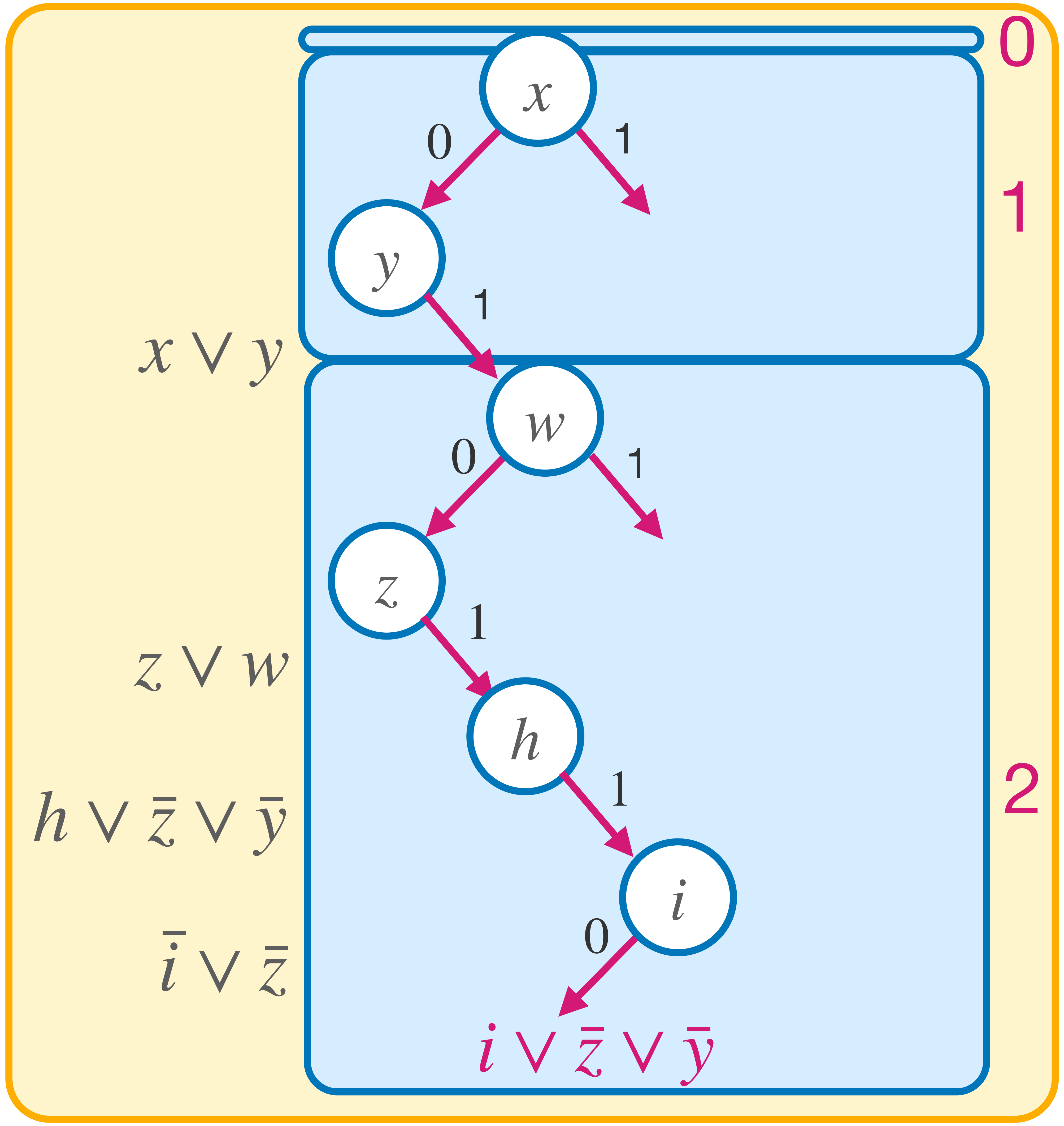
Standard clause to learn is a **1-UIP clause**

**1-UIP Clause**

Obtained by resolving the conflict clause along the path until there is only one literal in the clause at the largest decision level

**Backtracking with 1-UIP:**

Remove everything up to the second largest decision level in the learned clause



# Clause Learning

2 1 Backtrack to level 1

$(\bar{z} \vee \bar{y}) \wedge (x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$

Use Resolution to learn new clauses

$Q$. When should we stop?

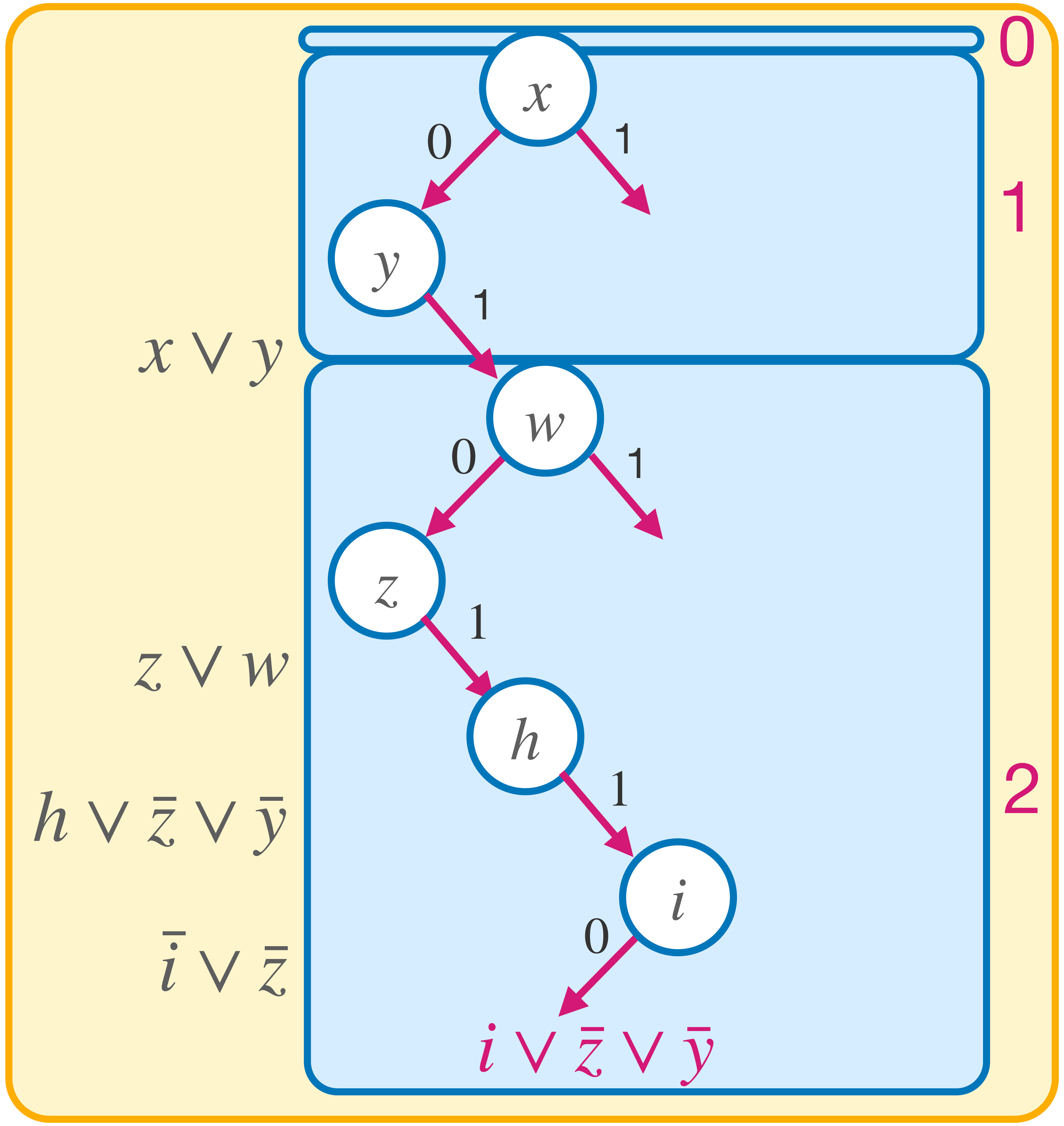
Standard clause to learn is a **1-UIP clause**

**1-UIP Clause**

Obtained by resolving the conflict clause along the path until there is only one literal in the clause at the largest decision level

**Backtracking with 1-UIP:**

Remove everything up to the second largest decision level in the learned clause



# Clause Learning

Backtrack to level 1

$$(\overset{2}{\bar{z}} \vee \overset{1}{\bar{y}}) \wedge (x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$$

Use Resolution to learn new clauses

$Q$. When should we stop?

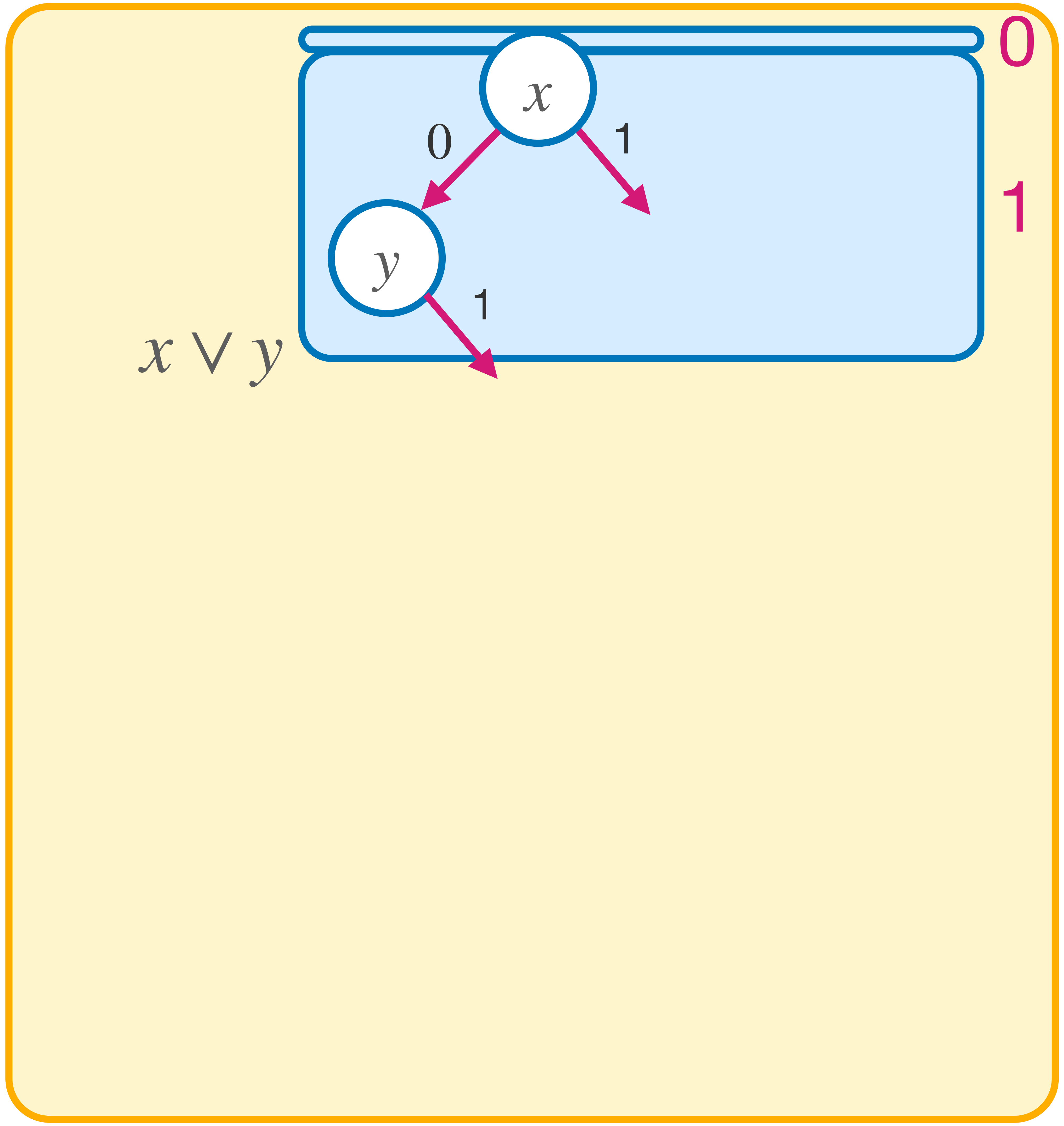
Standard clause to learn is a **1-UIP clause**

**1-UIP Clause**

Obtained by resolving the conflict clause along the path until there is only one literal in the clause at the largest decision level

**Backtracking with 1-UIP:**

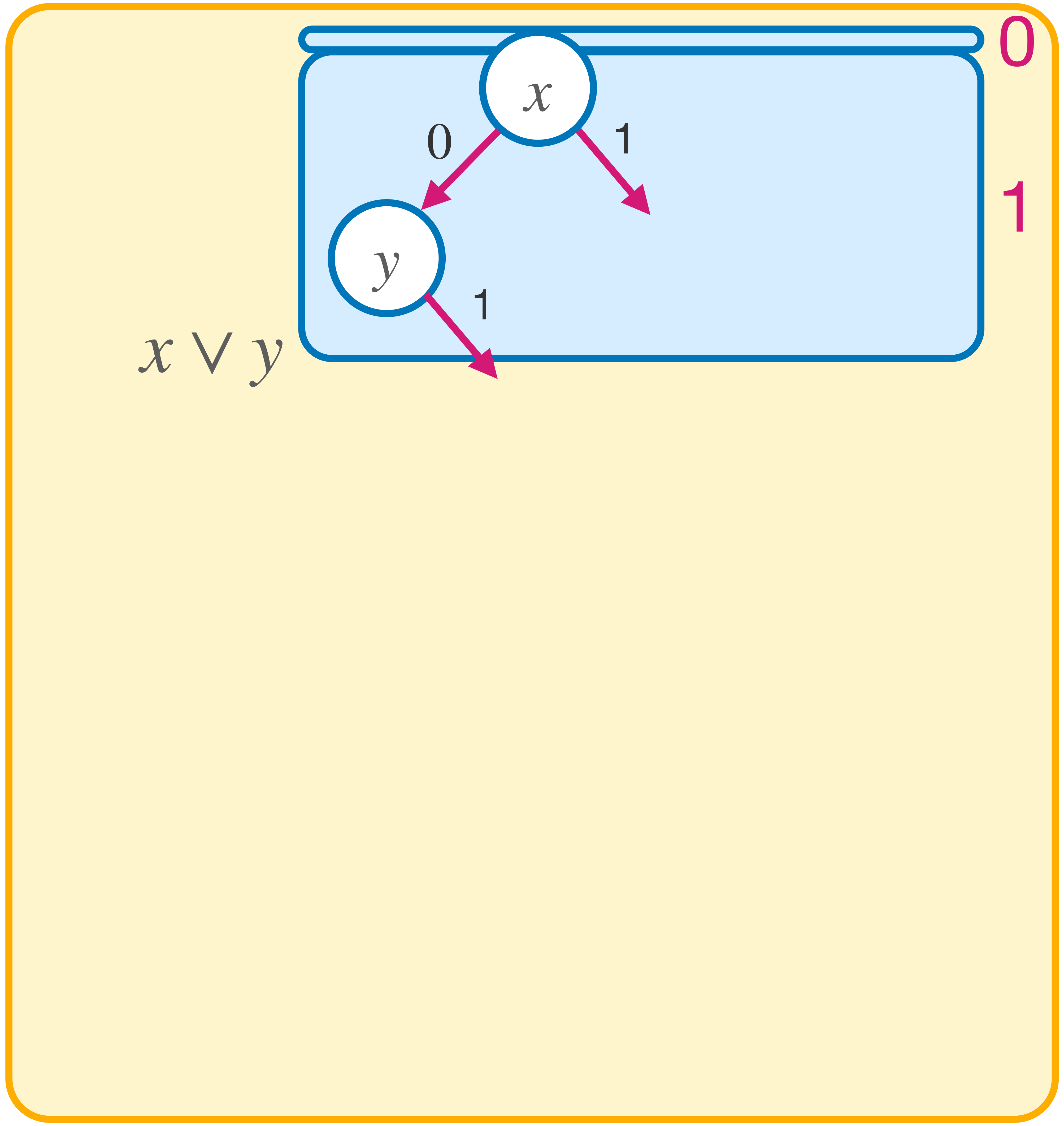
Remove everything up to the second largest decision level in the learned clause



# Clause Learning

$$(\bar{z} \lor \bar{y}) \land (x \lor y) \land (z \lor w) \land (h \lor \bar{z} \lor \bar{y}) \land (\bar{i} \lor \bar{z}) \land (i \lor \bar{z} \lor \bar{y})$$

$Q$. What happens when we backtrack?

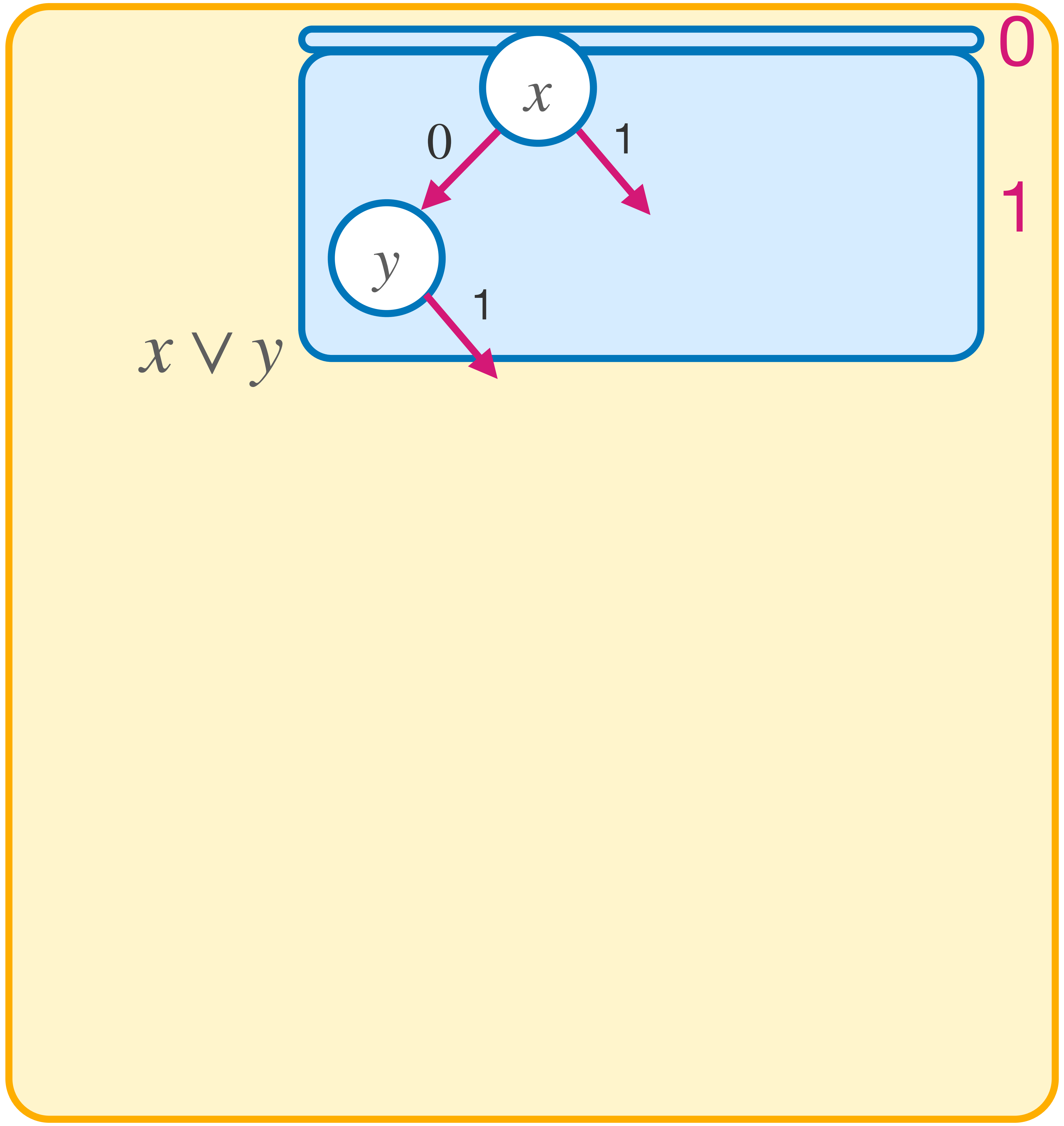


# Clause Learning

$(\bar{z} \vee \bar{y}) \wedge (x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$

$Q$. What happens when we backtrack?

New 1-UIP clause causes unit propagations!

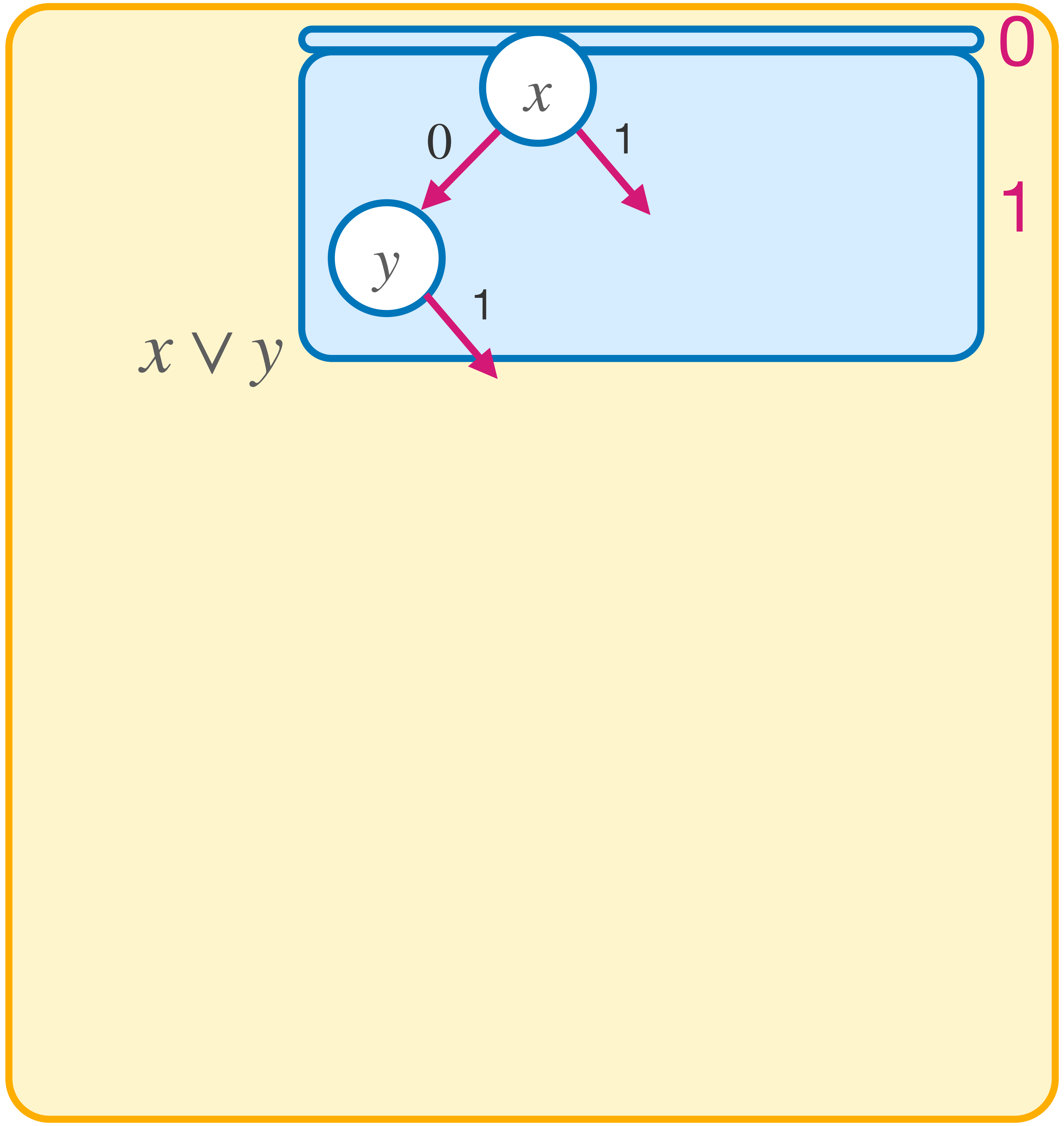


# Clause Learning

$(\bar{z} \vee \bar{y}) \wedge (x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$

$Q$ . What happens when we backtrack?

New 1-UIP clause causes unit propagations!
$\rightarrow$ This always happens because we backtracked to the second largest decision level in the learned clause!



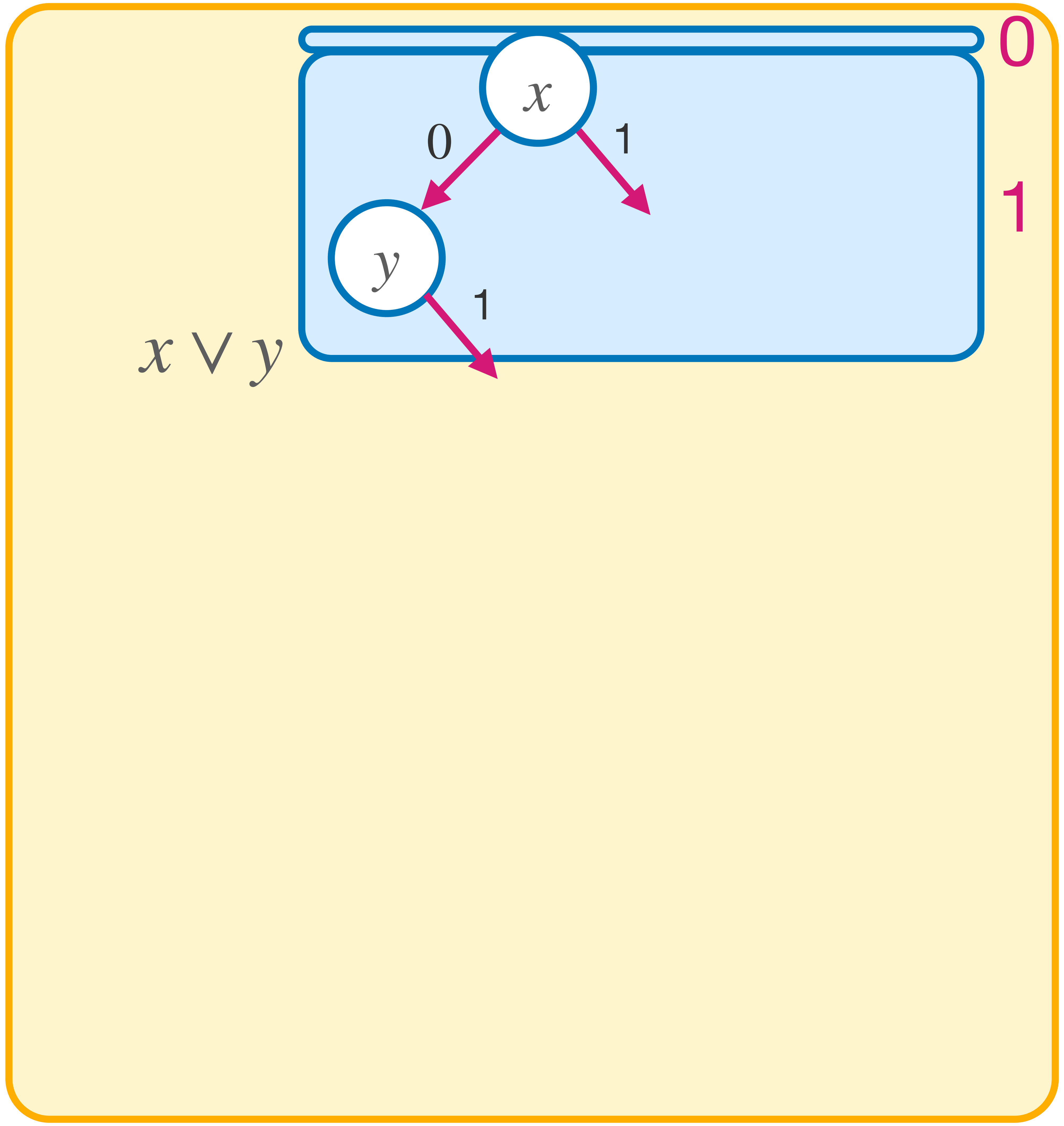
# Clause Learning

$(\bar{z} \vee \bar{y}) \wedge (x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$

$Q$. What happens when we backtrack?

New 1-UIP clause causes unit propagations!
→ This always happens because we backtracked to the second largest decision level in the learned clause!

⟹ It is a unit clause at this decision level!



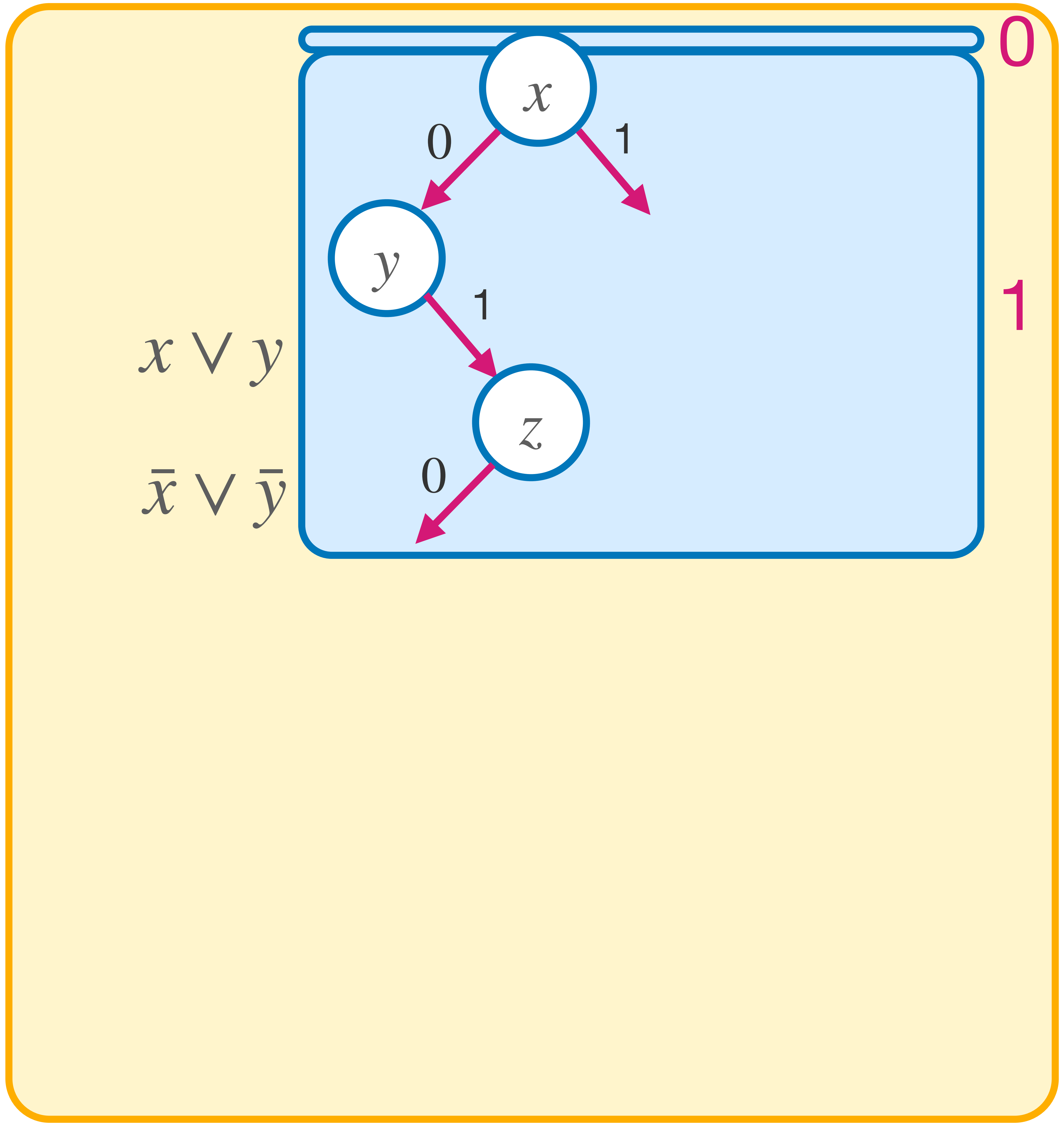
# Clause Learning

$(\bar{z} \vee \bar{y}) \wedge (x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$

$Q$. What happens when we backtrack?

New 1-UIP clause causes unit propagations!
→ This always happens because we backtracked to the second largest decision level in the learned clause!

$\Longrightarrow$ It is a unit clause at this decision level!



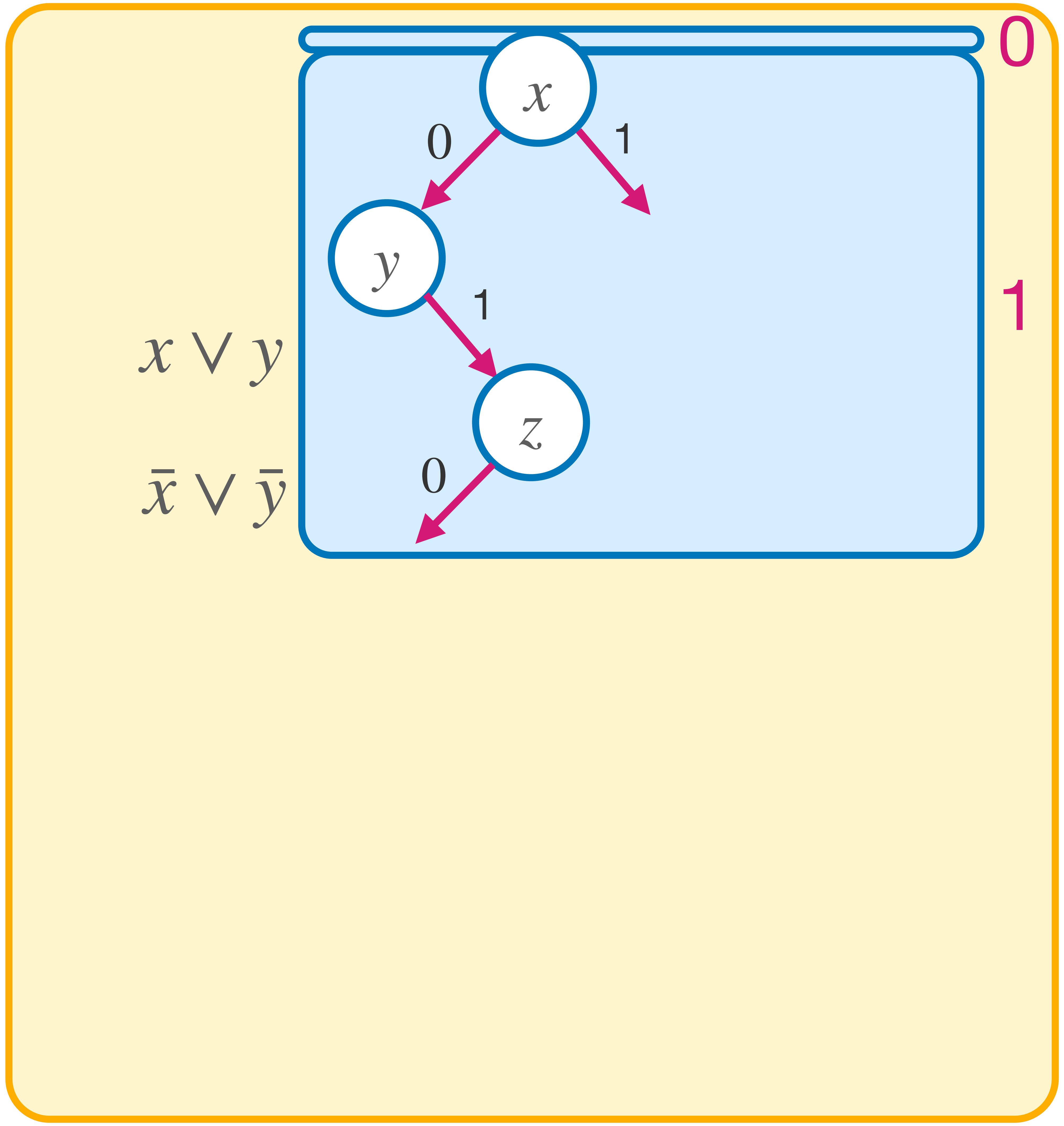
# Clause Learning

$(\bar{z} \vee \bar{y}) \wedge (x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$

$Q$. What happens when we backtrack?

New 1-UIP clause causes unit propagations!
→ This always happens because we backtracked to the second largest decision level in the learned clause!

$\Longrightarrow$ It is a unit clause at this decision level!



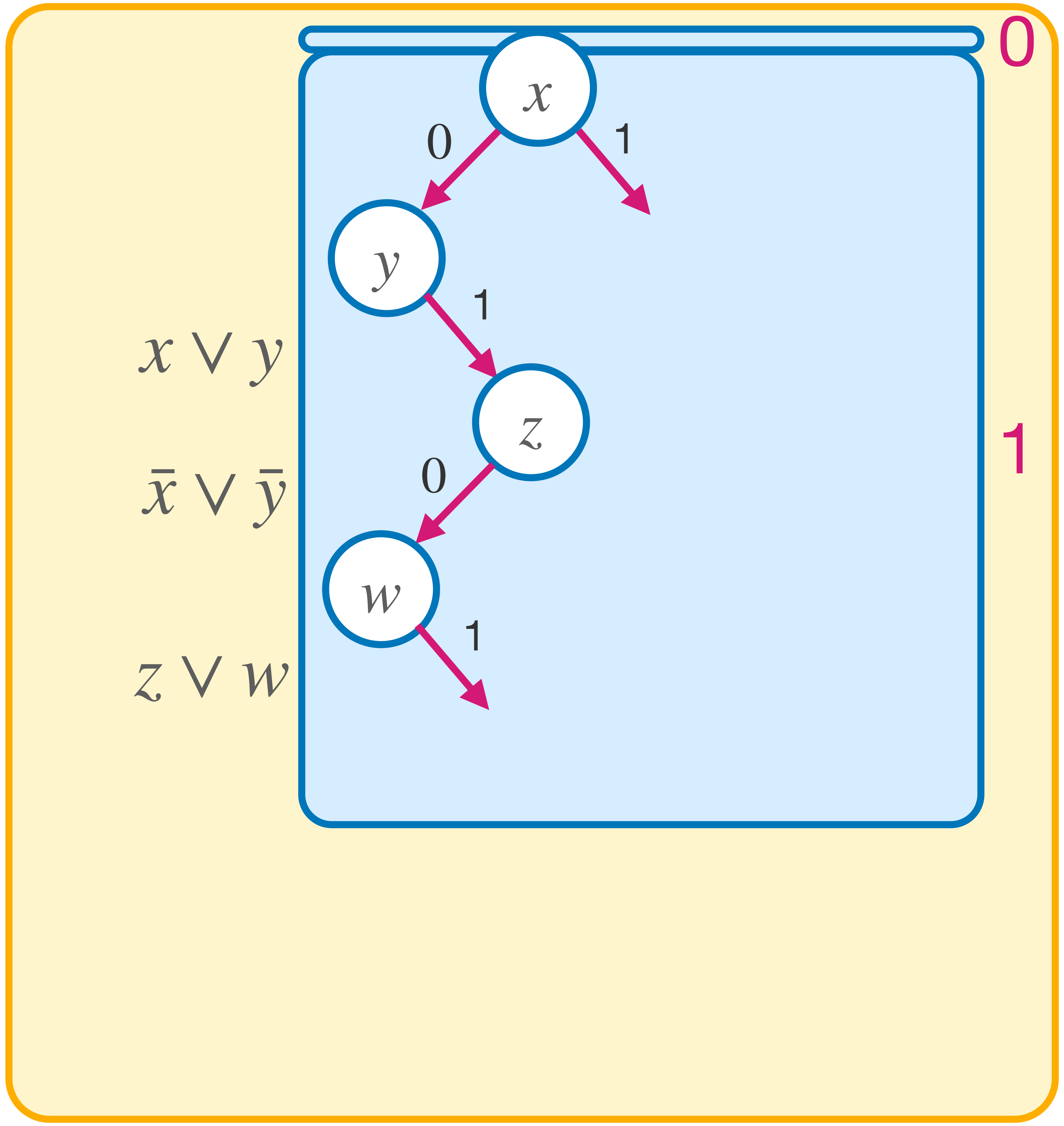
# Clause Learning

$(\bar{z} \vee \bar{y}) \wedge (x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$

$Q$. What happens when we backtrack?

New 1-UIP clause causes unit propagations!
$\rightarrow$ This always happens because we backtracked to the second largest decision level in the learned clause!

$\Longrightarrow$ It is a unit clause at this decision level!



# Clause Learning

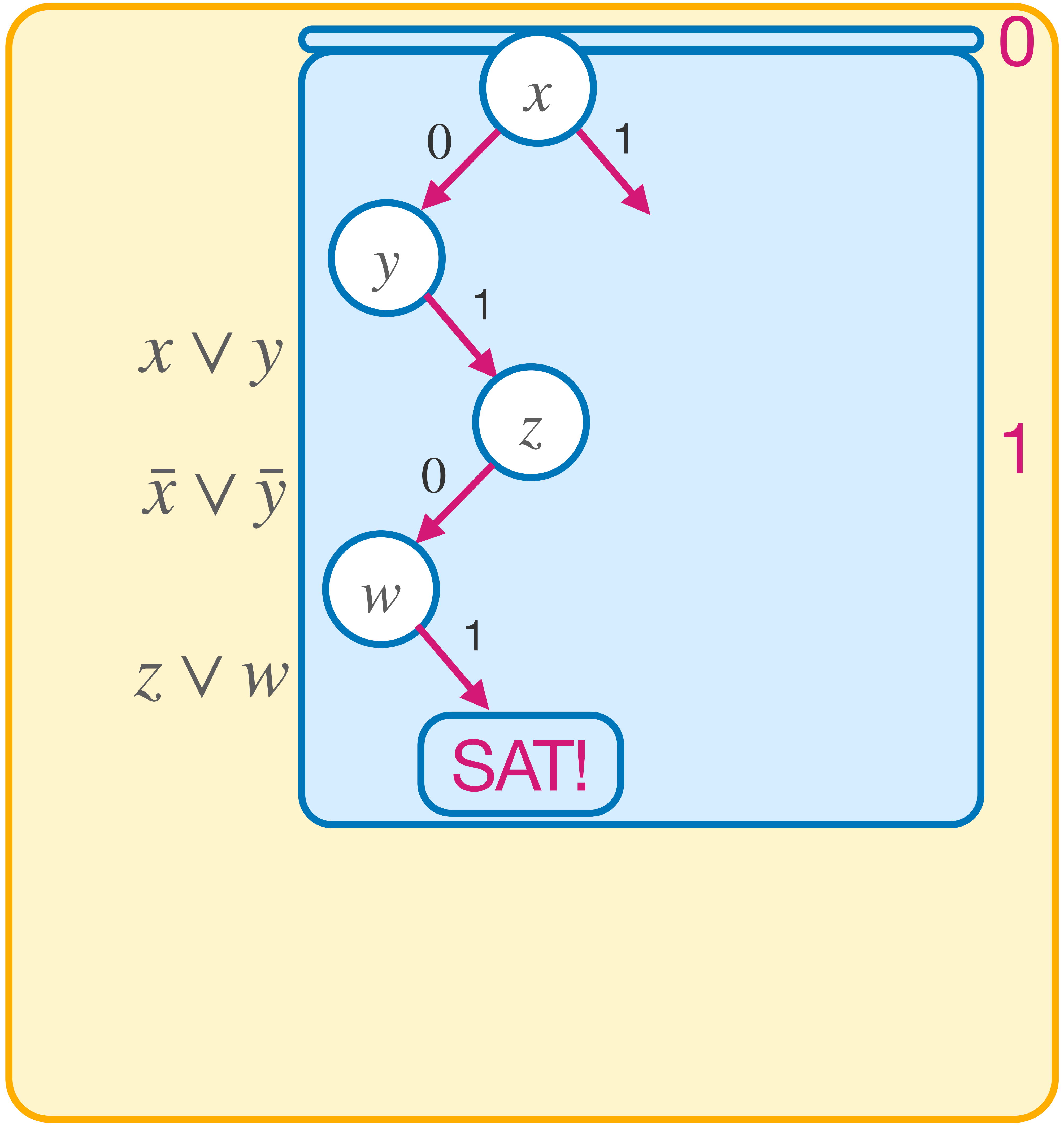
$(\bar{z} \vee \bar{y}) \wedge (x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$

$Q$. What happens when we backtrack?

New 1-UIP clause causes unit propagations!

$\rightarrow$ This always happens because we backtracked to the second largest decision level in the learned clause!

$\Longrightarrow$ It is a unit clause at this decision level!



# Clause Learning

$$\color{magenta}{(\bar{z} \lor \bar{y}) \land} (x \lor y) \land (z \lor w) \land (h \lor \bar{z} \lor \bar{y}) \land (\bar{i} \lor \bar{z}) \land (i \lor \bar{z} \lor \bar{y})$$

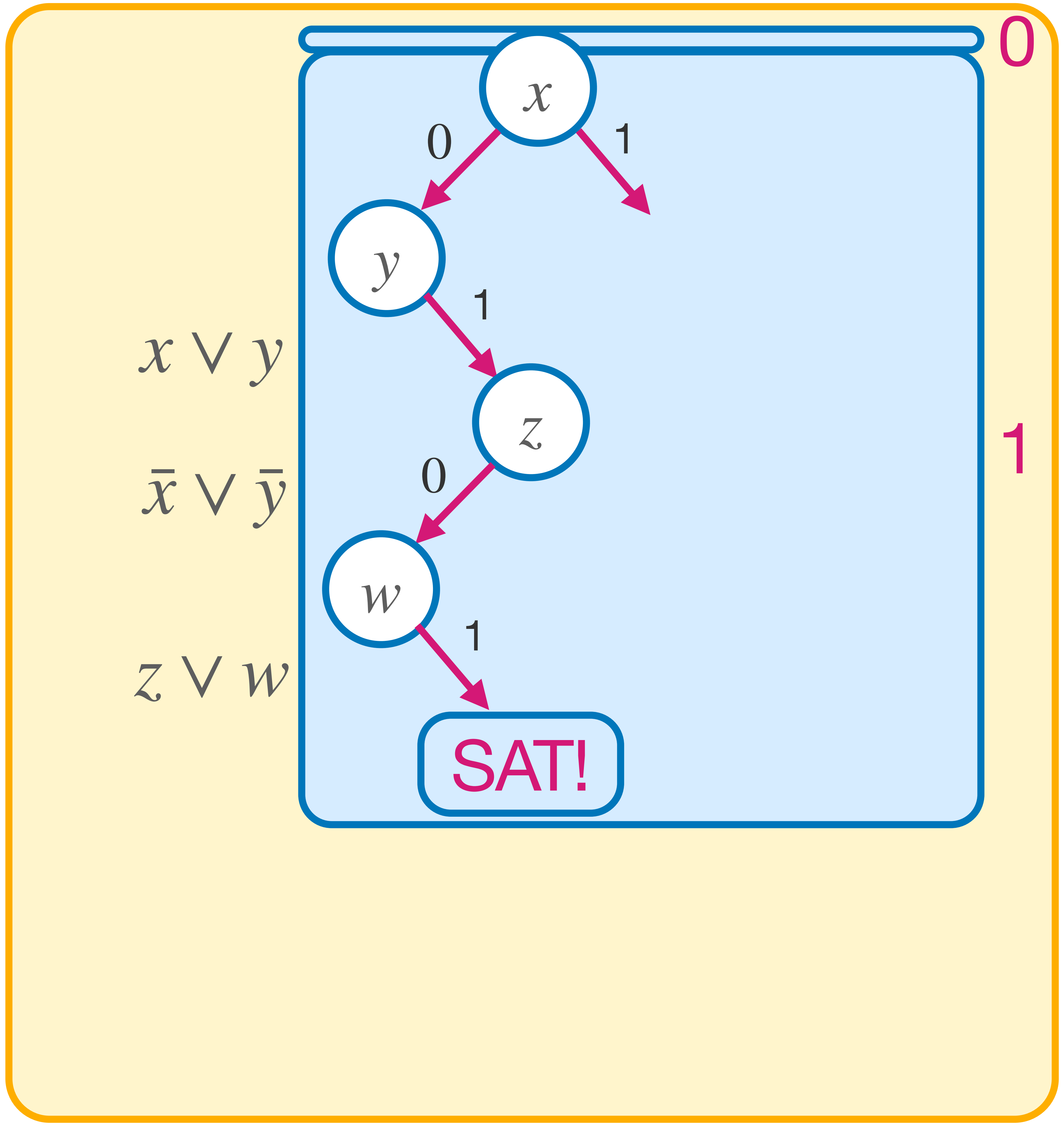
$Q$. What happens when we backtrack?

New 1-UIP clause causes unit propagations!

→ This always happens because we backtracked to the second largest decision level in the learned clause!

⟹ It is a unit clause at this decision level!

→ Known as an asserting clause



# Conflict-Driven Clause Learning

Modern (CDCL) SAT Solvers build on DPLL

→ Multiple subroutines built to avoid getting stuck in bad areas of the search space

We will develop CDCL in stages by extending DPLL with the following:

- Unit Propagation ✓
- Clause Learning ✓
- Restarts

# Restarting

**Restarting:**
After learning so many clauses, restart the search

Helps to escape bad areas of the search space

# Restarting

**Restarting:**

After learning so many clauses, restart the search

$\rightarrow$ Return to decision level $0$, discarding all queries made so far

# Restarting

**Restarting:**
After learning so many clauses, restart the search

$\rightarrow$ Return to decision level $0$, discarding all queries made so far

$\rightarrow$ Retain all learned clauses

# Restarting

**Restarting:**
After learning so many clauses, restart the search

$\rightarrow$ Return to decision level $0$, discarding all queries made so far

$\rightarrow$ Retain all learned clauses

Helps to escape bad areas of the search space

# Restarting

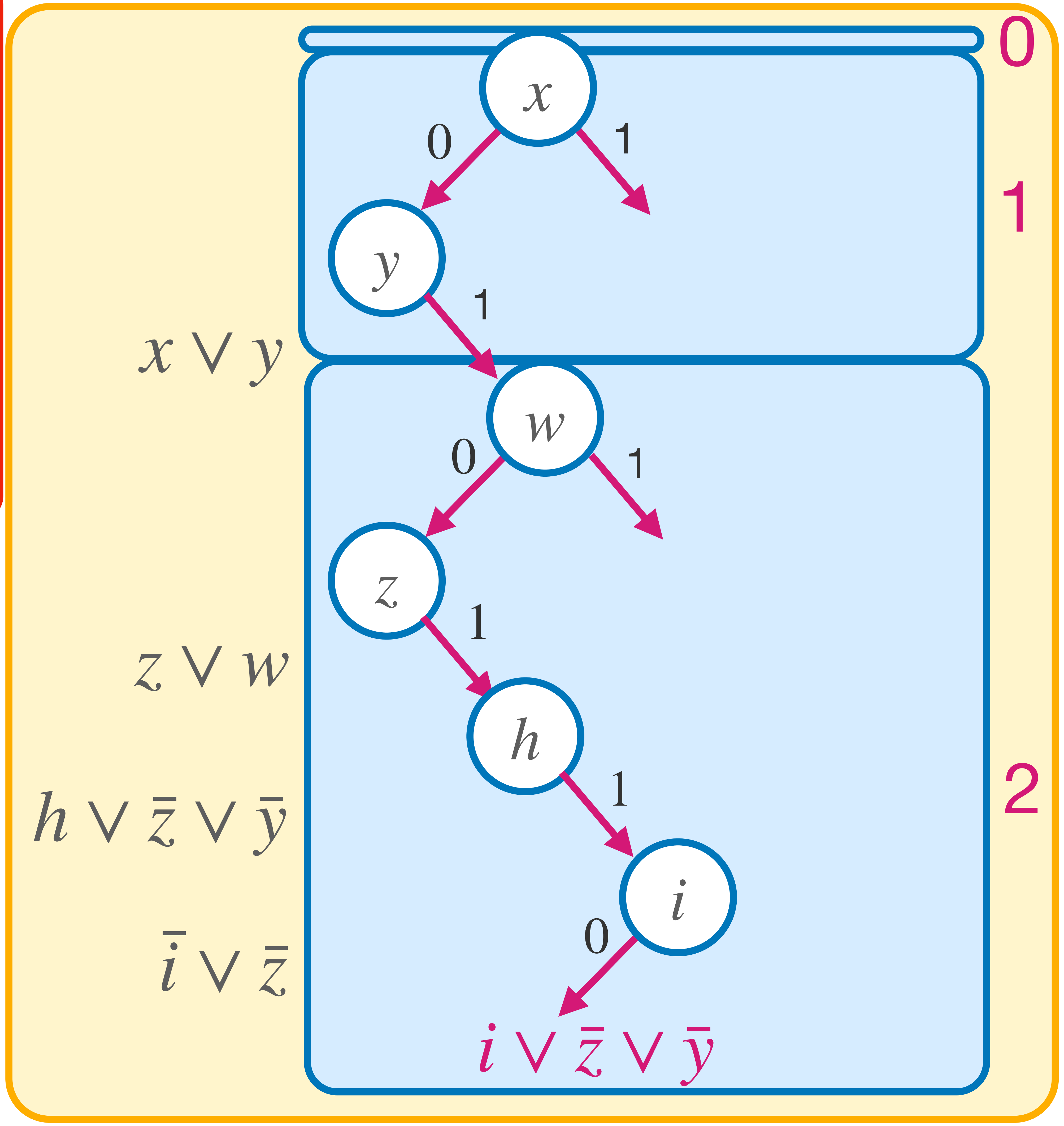
$(\bar{z} \vee \bar{y}) \wedge (x \vee y) \wedge (z \vee w) \wedge (h \vee \bar{z} \vee \bar{y}) \wedge (\bar{i} \vee \bar{z}) \wedge (i \vee \bar{z} \vee \bar{y})$

**Restarting:**

After learning so many clauses, restart the search

$\rightarrow$ Return to decision level $0$, discarding all queries made so far

$\rightarrow$ Retain all learned clauses




Helps to escape bad areas of the search space

# Restarting

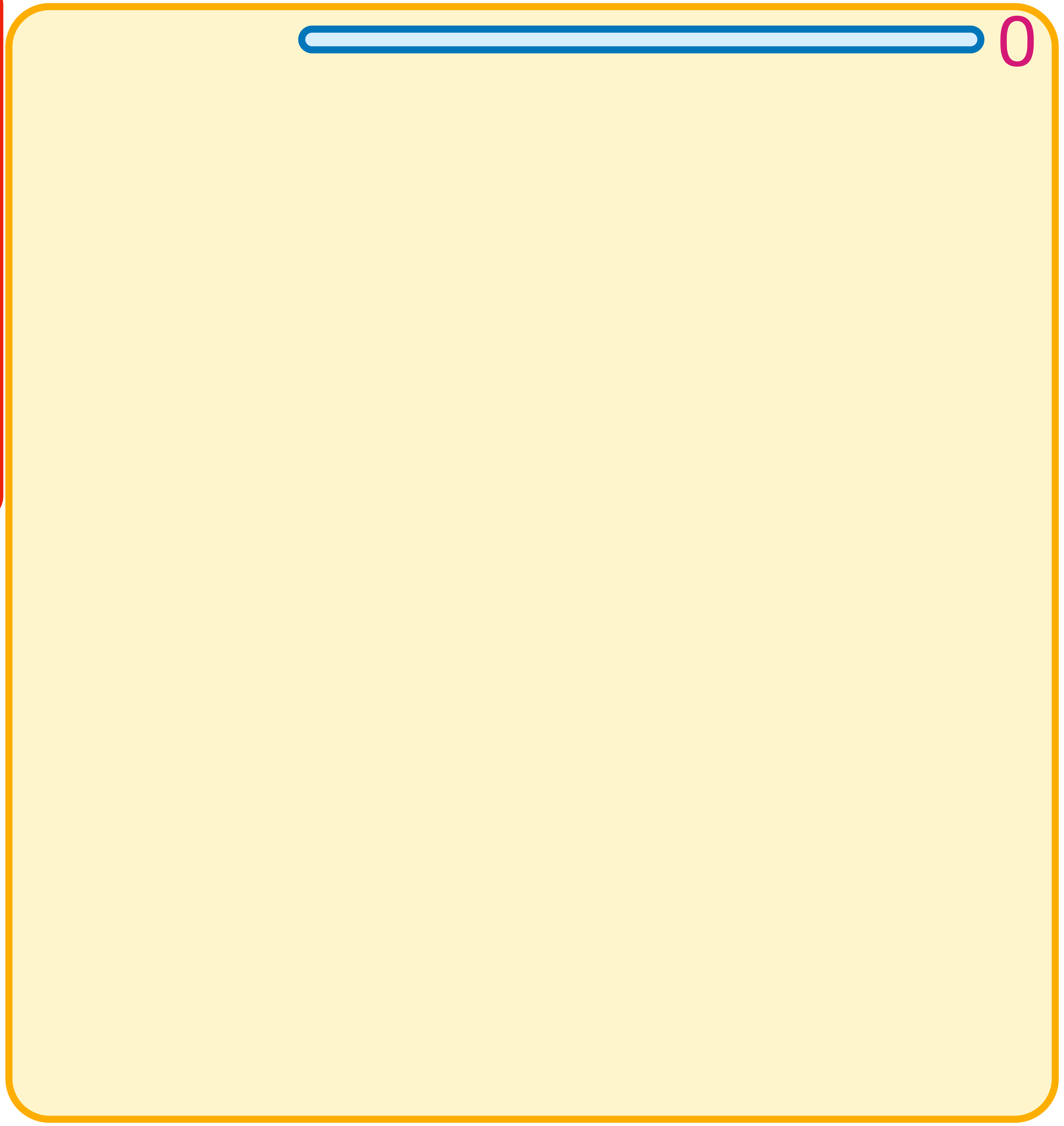
$$(\bar{z} \lor \bar{y}) \land (x \lor y) \land (z \lor w) \land (h \lor \bar{z} \lor \bar{y}) \land (\bar{i} \lor \bar{z}) \land (i \lor \bar{z} \lor \bar{y})$$

**Restarting:**

After learning so many clauses, restart the search

$\rightarrow$ Return to decision level $0$, discarding all queries made so far

$\rightarrow$ Retain all learned clauses




Helps to escape bad areas of the search space