SMT Solving

Wenxi Wang

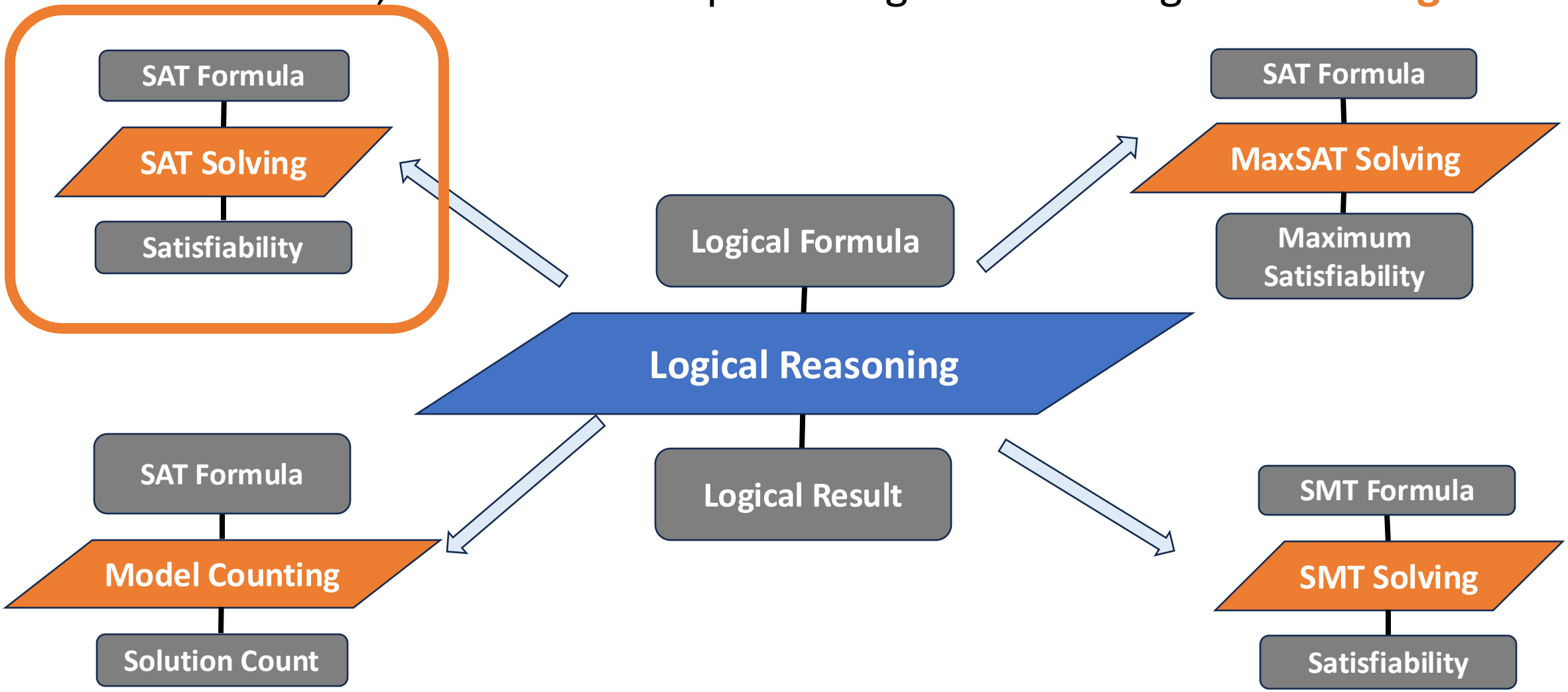
University of Virginia

wenxiw@virginia.edu

Recap

Logical Reasoning

In this lecture, we focus on a specific logical reasoning- **SAT solving**



Recap

Logical Reasoning

Propositional Logic

Variables: Boolean

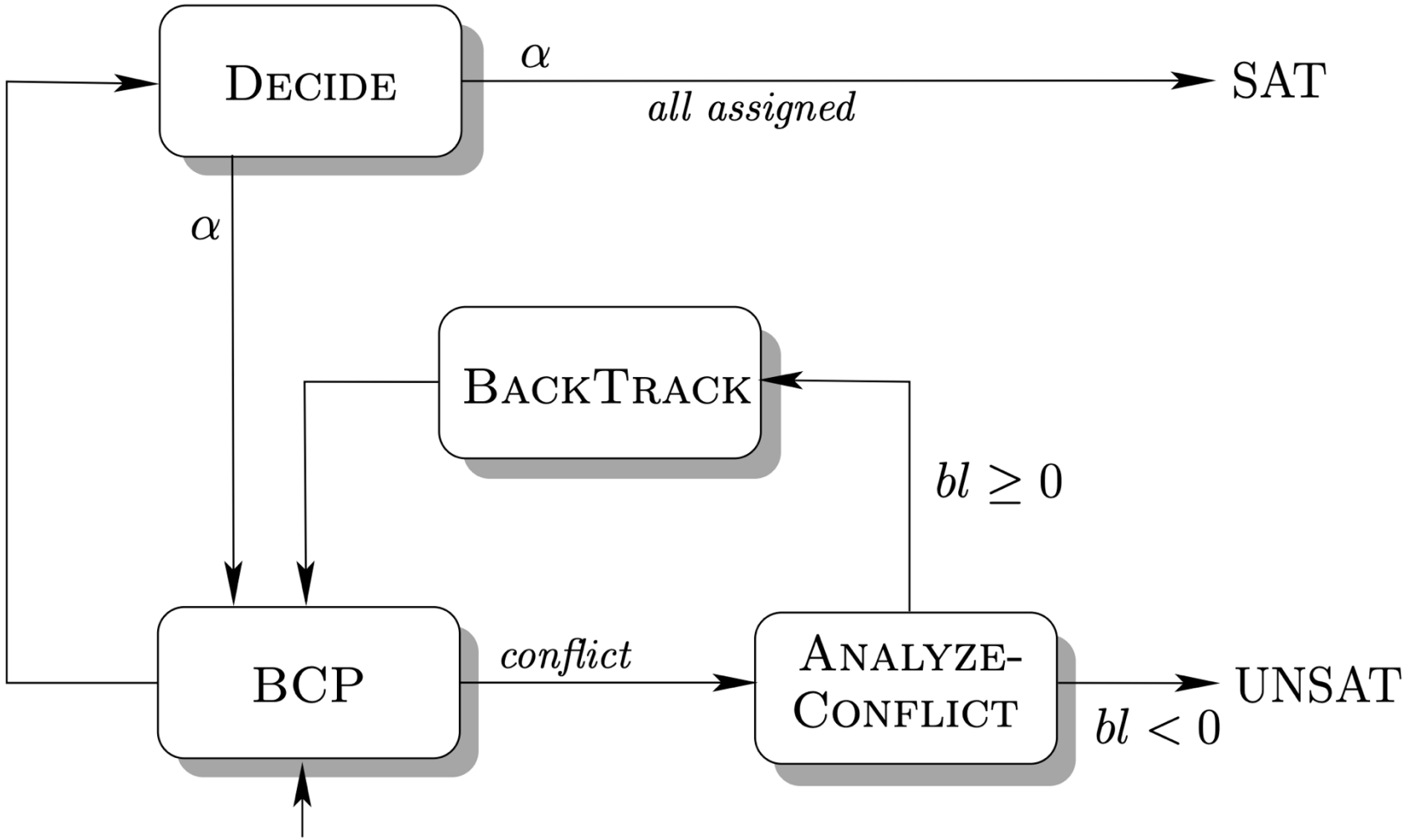
Operators sorted by precedence:

¬ >> ∧ >> ∨ >> ⇒ >> ⇔

Recap

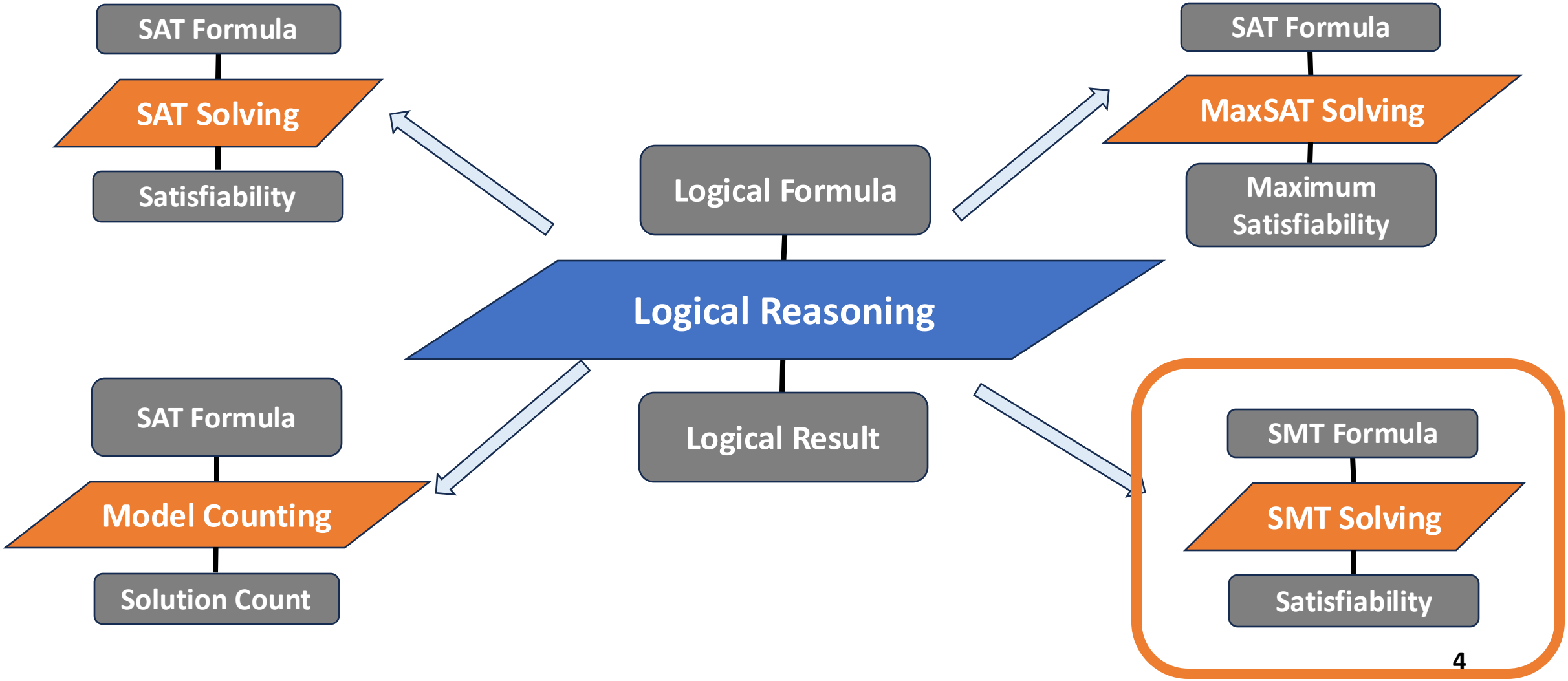
CDCL SAT solving

General Workflow



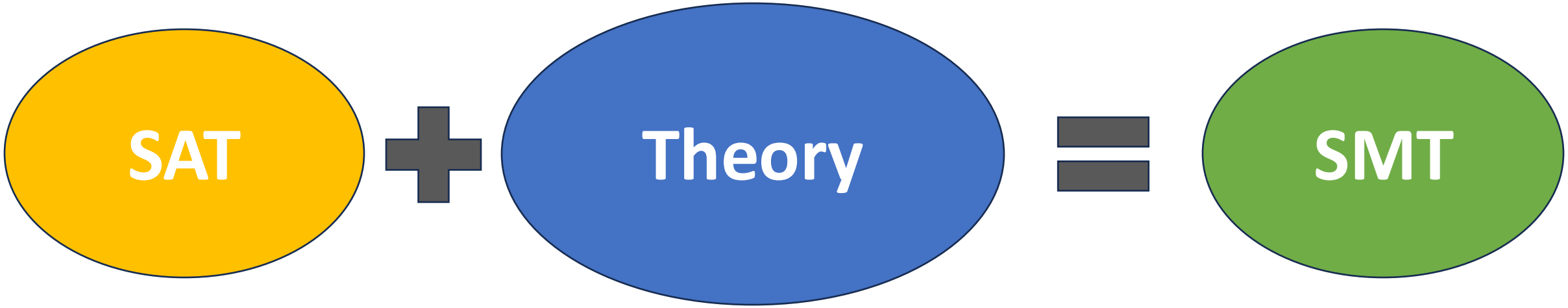
Logical Reasoning

In this lecture, we focus on **SMT solving**



SMT Solving

SMT: Satisfiability Modulo Theory



Bit vectors: ((a>> b) & c) < c

Arithmetic: $(2x + 3y \leq 5) \vee (y + 5y - 10z \geq 6)$

Equality: $y1 = y2 \wedge \neg(y1 = y3) \Rightarrow \neg(y2 = y3)$

Arrays: $(i = j \wedge a[j] = 1) \Rightarrow a[i] = 1$

… …

SMT Solving

First-order logic

First-order logic extends propositional logic with **quantifiers** and the **nonlogical symbols,** representing all kinds of **theories**.

Bit vectors: ((a>> b) & c) < c

Arithmetic: (2x + 3y ≤ 5) ∨ (y + 5y − 10z ≥ 6)

Equality: y1 = y2 ∧ ¬(y1 = y3) ⇒ ¬(y2 = y3)

Arrays: (i = j ∧ a[j] = 1) ⇒ a[i] = 1

… …

Example:

- $\exists y \in Z.\ \forall x \in Z.\ x > y$,
- $\forall n \in N.\ \exists p \in N.\ n > 1 \Rightarrow (isprime(p) \wedge n < p < 2n)$,

where “>”, “isprime” are nonlogical symbols

SMT Solving

Let's focus on a simple case:
From Propositional to Quantifier-Free Theories

Example:

$F = \underbrace{x1 + x2 > 9}_{b1} \vee x2 \neq x3 \wedge \underbrace{x2 = x3}_{b2} <=> \underbrace{x4 > x5}_{b3} \wedge A \wedge \neg B$

Propositional Skeleton PS_F = $(b1 \vee \neg b2) \wedge b2 <=> b3 \wedge A \wedge \neg B$

$b1$: $x1 + x2 > 9$ — can be denoted as **e(x1+x2>9)**

$b2$: $x2 = x3$ — can also be denoted as **e(x2 = x3)**

$b3$: $x4 > x5$ — can also be denoted as **e($x4 > x5$)**

SMT Solving

Interpretation of symbols is important!

Syntax to Semantics

Example

F : x + y > z => y > z − x

We construct a “standard” interpretation **I**

The domain is the integers, $\mathbb{Z}$: $D\_I = \mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$

α_I:{ + ⟼ +$\mathbb{Z}$, − ⟼ −$\mathbb{Z}$, > ⟼ >$\mathbb{Z}$, x⟼10, y⟼39, x⟼-2}

SMT Solving

T-satisfiability

Let T be a Σ-theory.

A **Σ-**formula φ is T-satisfiable if there exists an interpretation I such that the interpretation satisfies φ, $I \models \varphi$.

SMT Solving

Two main approaches for SMT

Lazy Approach:
Integrate a theory solver with a CDCL solver for SAT

Eager Approach
Encode the SMT formula to a equi-satisfiable SAT formula

Lazy SMT Solving

General Algorithm

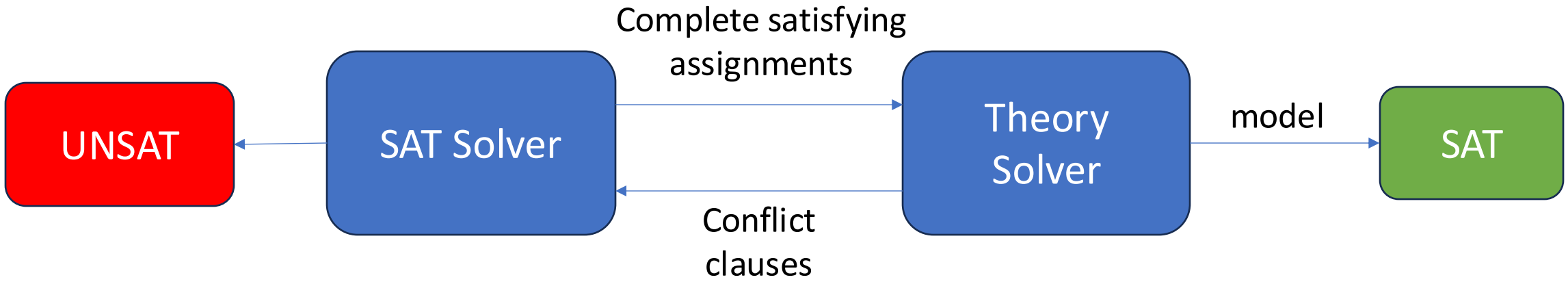
Algorithm LAZY-CDCL

Input: A formula φ
Output: "Satisfiable" if the formula is satisfiable, and "Unsatisfiable" otherwise

```
1.  function LAZY-CDCL
2.      ADDCLAUSES(cnf(e(φ)));
3.      while (TRUE) do
4.          while (BCP() = "conflict") do
5.              backtrack-level := ANALYZE-CONFLICT();
6.              if backtrack-level < 0 then return "Unsatisfiable";
7.              else BackTrack(backtrack-level);
8.          if ¬DECIDE() then                          ▷ Full assignment
9.              ⟨t, res⟩:=DEDUCTION(T̂h(α));            ▷ α is the assignment
10.             if res="Satisfiable" then return "Satisfiable";
11.             ADDCLAUSES(e(t));
```

Lazy SMT Solving

General framework

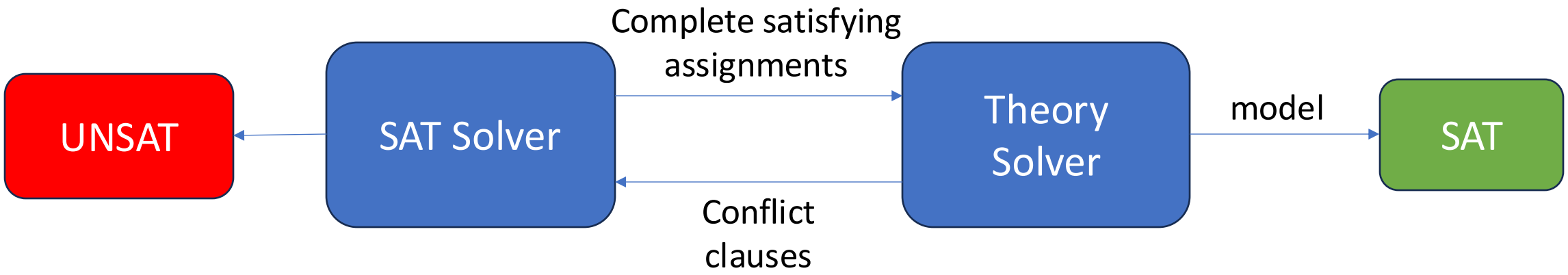


Lazy SMT Solving

Running Example

$(x = y) \wedge (y = z) \wedge (x \neq z)$

Propositional skeleton **PS:** **b1** $\wedge$ **b2** $\wedge$ **b3**

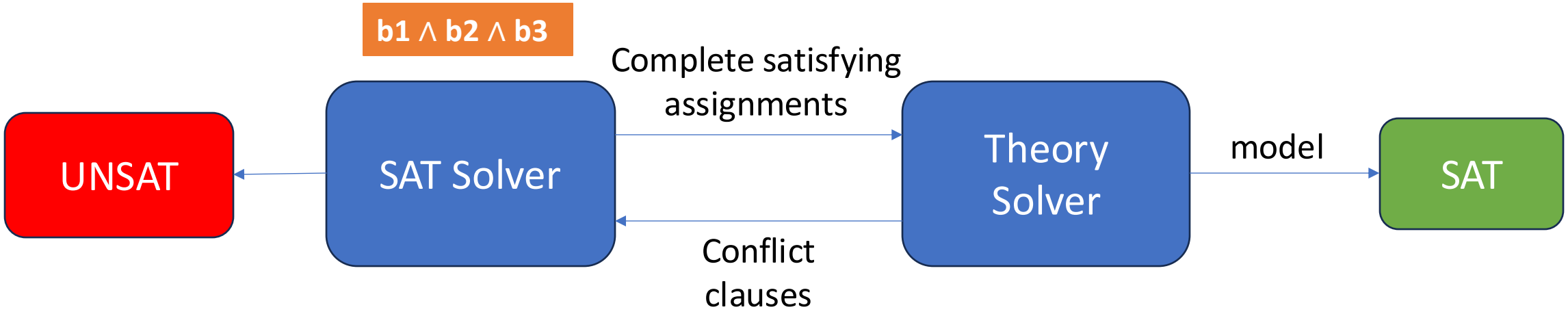


Lazy SMT Solving

Running Example

$$(x = y) \wedge (y = z) \wedge (x \neq z)$$

Propositional skeleton **PS:** $b1 \wedge b2 \wedge b3$

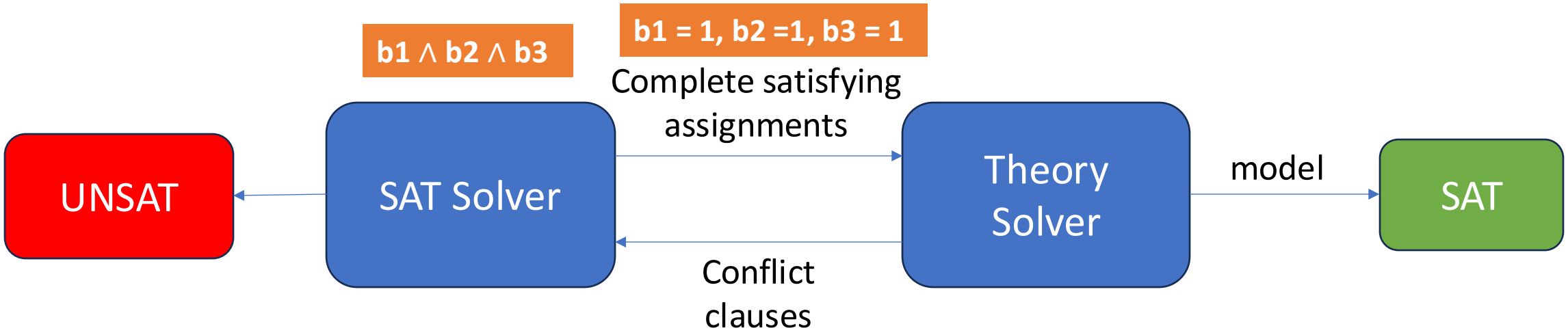


Lazy SMT Solving

Running Example

(x = y) ∧ (y = z) ∧ (x ≠ z)

Propositional skeleton **PS:** **b1** ∧ **b2** ∧ **b3**

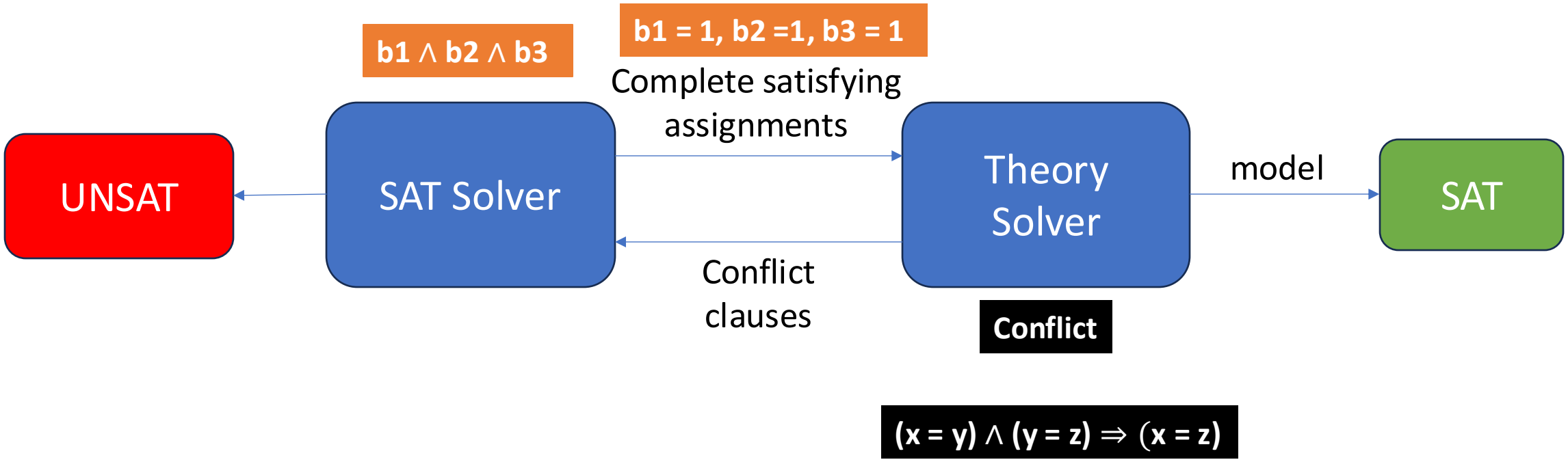


Lazy SMT Solving

Running Example

$$(x = y) \wedge (y = z) \wedge (x \neq z)$$

Propositional skeleton **PS:** **b1** $\wedge$ **b2** $\wedge$ **b3**

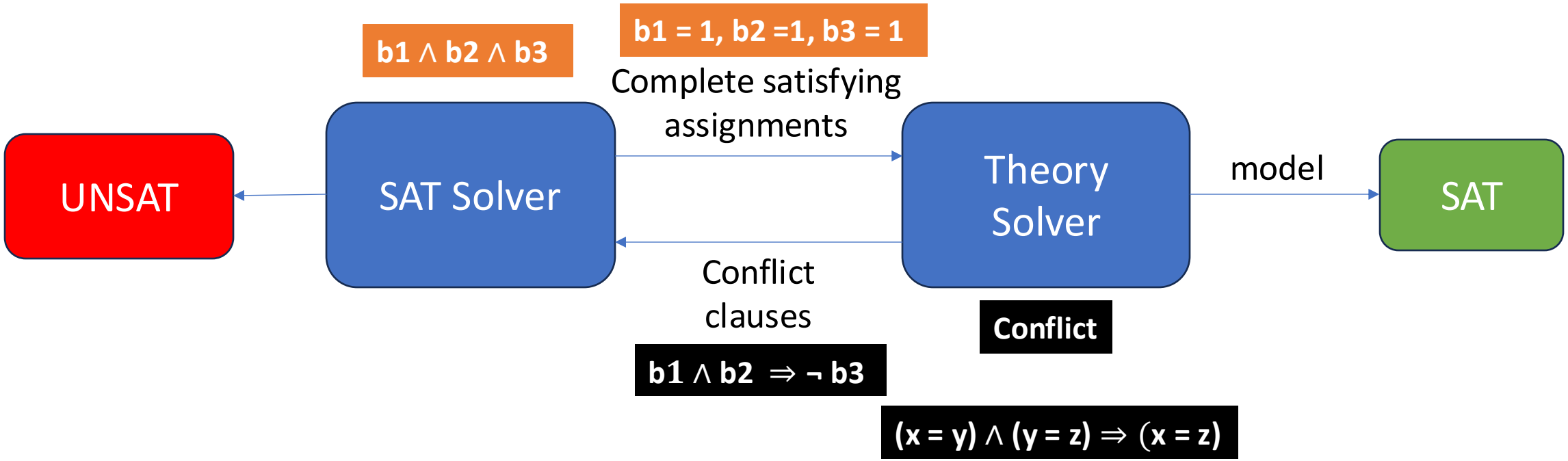


Lazy SMT Solving

Running Example

$(x = y) \wedge (y = z) \wedge (x \neq z)$

Propositional skeleton **PS:** **b1 ∧ b2 ∧ b3**

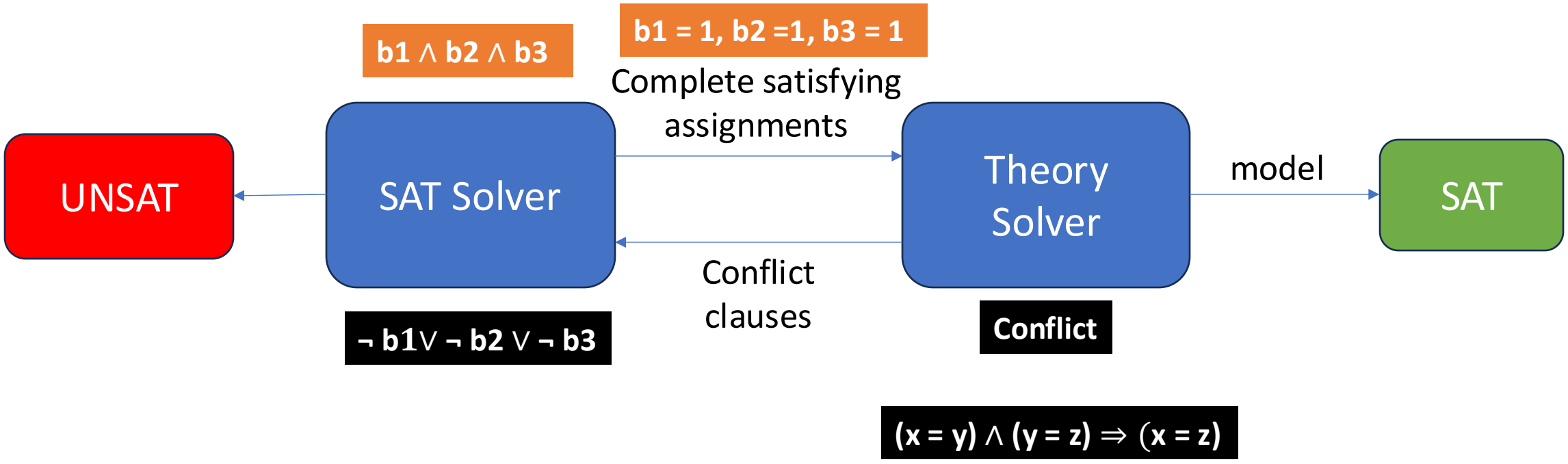


Lazy SMT Solving

Running Example

$(x = y) \wedge (y = z) \wedge (x \neq z)$

Propositional skeleton **PS:** **b1 ∧ b2 ∧ b3**

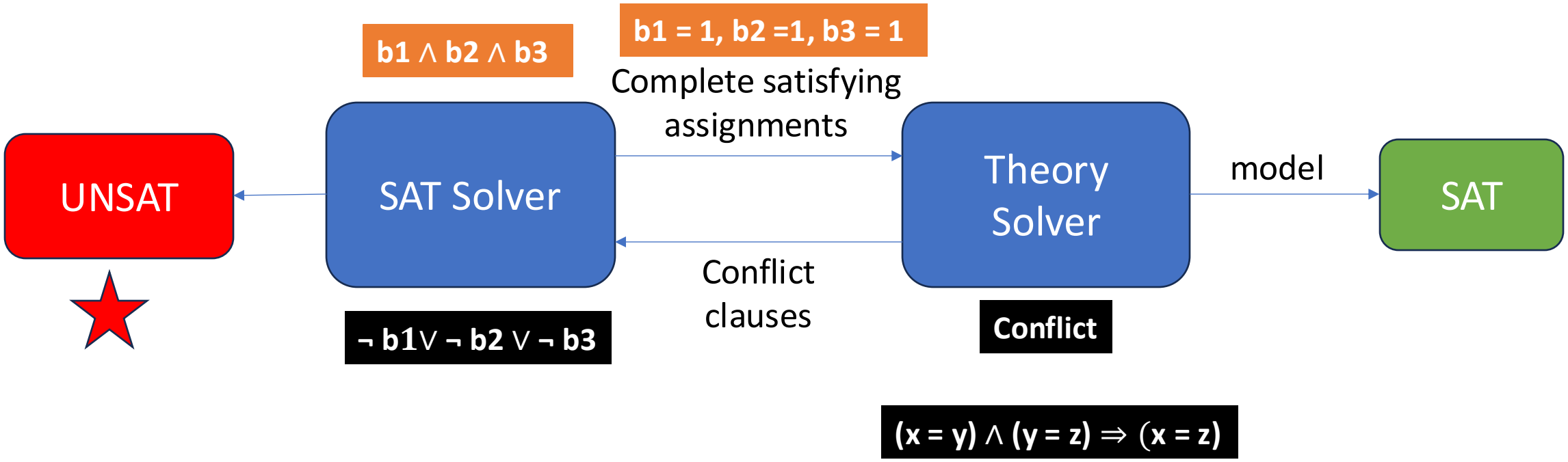


Lazy SMT Solving

Running Example

(x = y) ∧ (y = z) ∧ (x ≠ z)

Propositional skeleton **PS:** **b1 ∧ b2 ∧ b3**

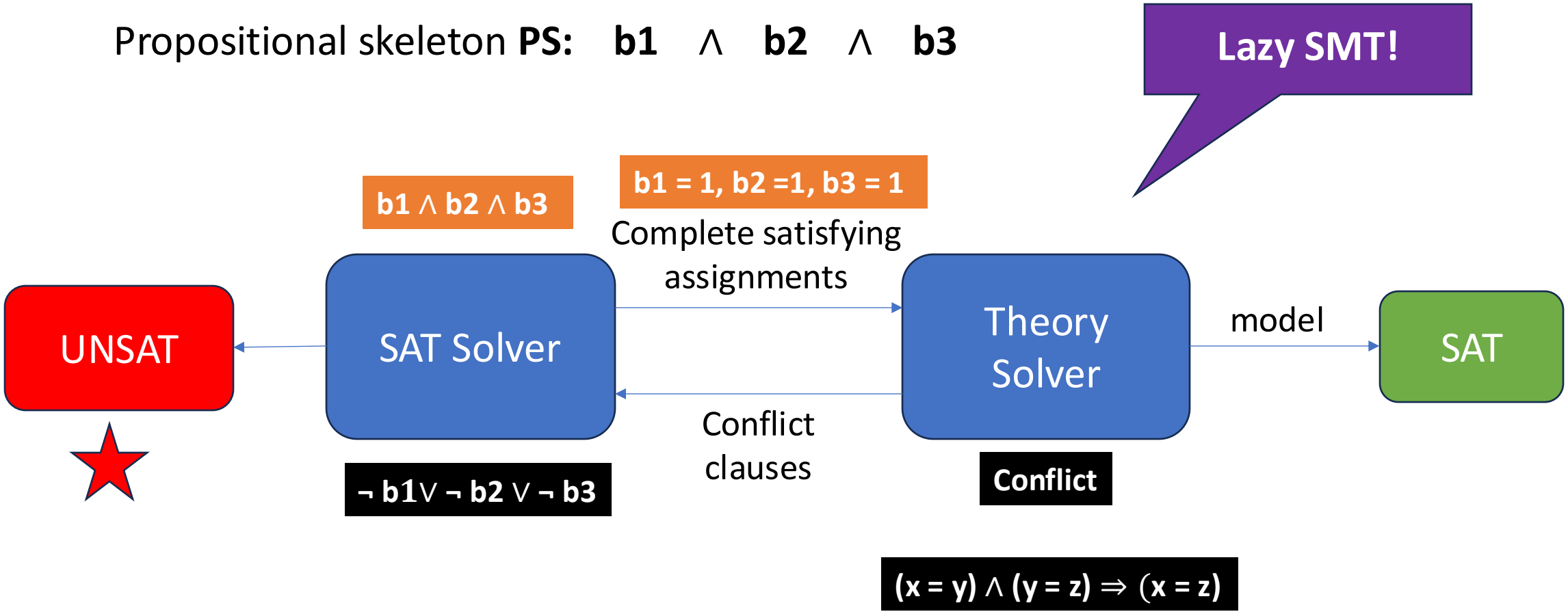


Lazy SMT Solving

Running Example

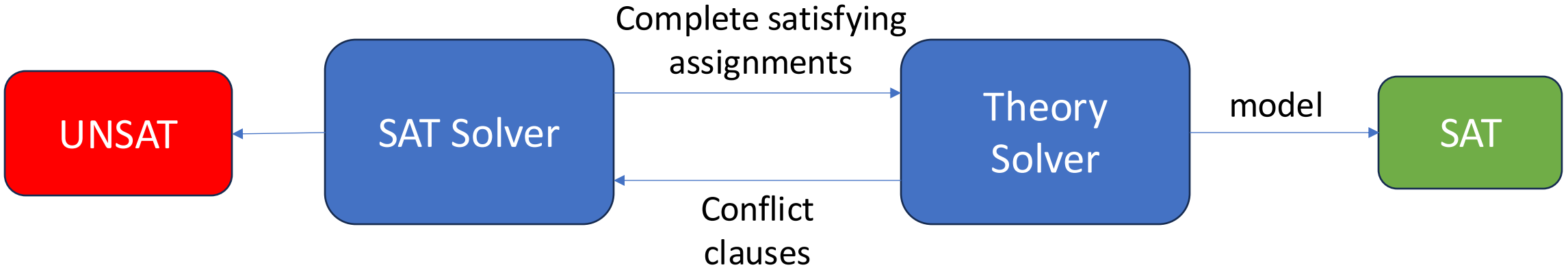
(x = y) ∧ (y = z) ∧ (x ≠ z)

Propositional skeleton **PS:** **b1 ∧ b2 ∧ b3**



Lazy SMT Solving

Still not efficient enough …



Example, a formula that contains literals

$x1 \geq 12$ and $x1 < 2$

$x1$ is integer

Assume SAT assigns $e(x1 \geq 12) \mapsto$ true and $e(x1 < 2) \mapsto$ true.

Lazy SMT solving does not call Theory solver until a full satisfying assignment is found.

Any call to Theory solver results in a contradiction between these two facts.

waste time to compute the complete SAT assignment

Improvements: DPLL(T)

Do Theory Propagation when the SAT assignment is still partial!

Example

a formula that contains literals $x1 \geq 12$ and $x1 < 2$

Assume SAT makes $e(x1 \geq 12)$ **true**,

Theory solver detects that $\neg(x1 < 2)$ is implied, that is

$e(x1 \geq 12) \Rightarrow \neg e(x1 < 2)$

Then $\neg e(x1 \geq 12) \vee \neg e(x1 < 2)$ is added to SAT solver

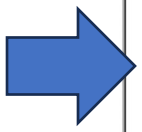
Improvements: DPLL(T)

Call theory solver to do Theory Propagation
when the SAT assignment is still partial!

Algorithm LAZY-CDCL

Input: A formula φ
Output: "Satisfiable" if the formula is satisfiable, and "Unsat
otherwise

```
1.  function LAZY-CDCL
2.      AddClauses(cnf(e(φ)));
3.      while (TRUE) do
4.          while (BCP() = "conflict") do
5.              backtrack-level := Analyze-Conflict();
6.              if backtrack-level < 0 then return "Unsatisfi
7.              else BackTrack(backtrack-level);
8.          if ¬Decide() then                              ▷
9.              ⟨t, res⟩:=Deduction(T̂h(α));                ▷ α is
10.             if res="Satisfiable" then return "Satisfiable
11.             AddClauses(e(t));
```



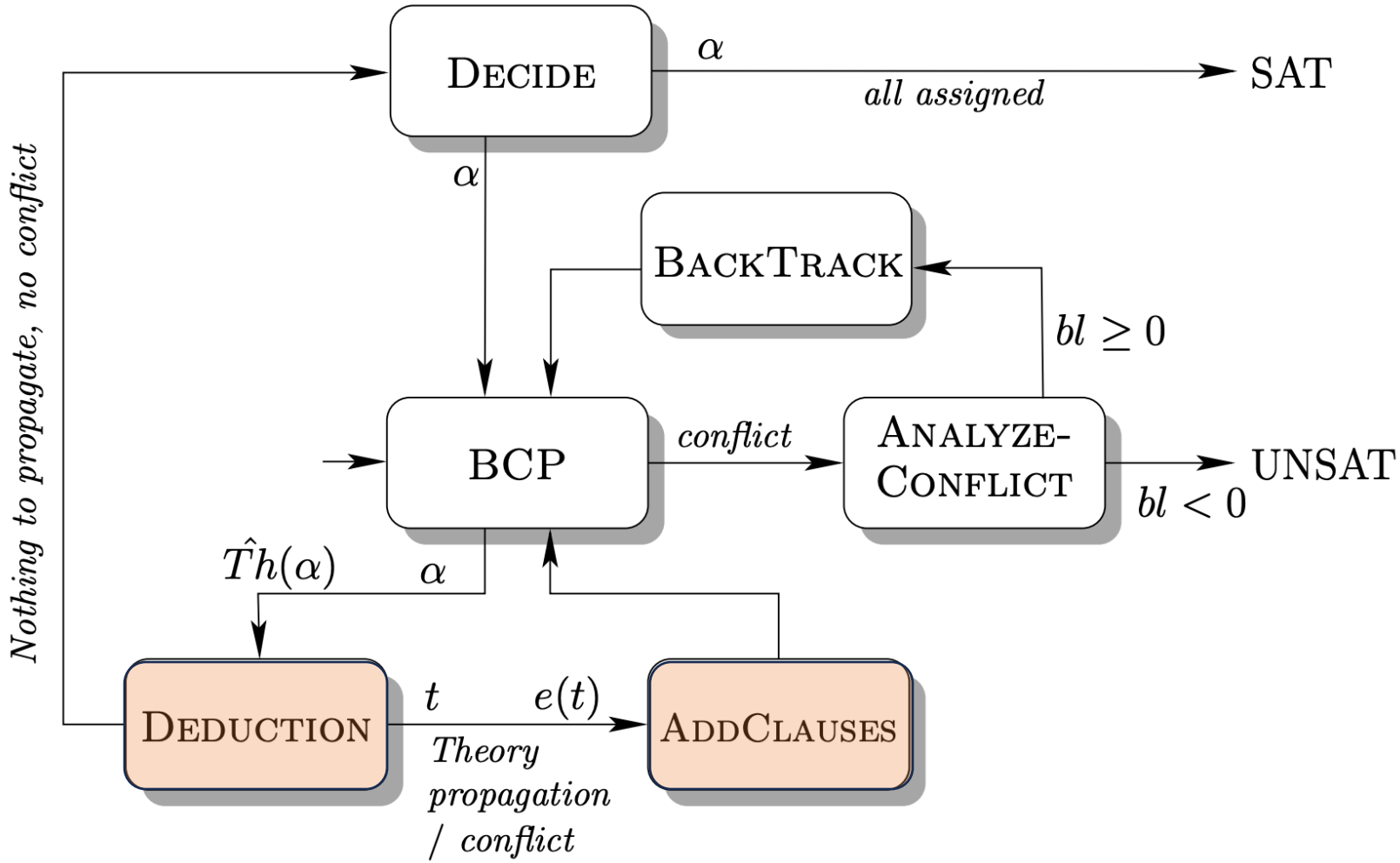
Algorithm DPLL(T)

Input: A formula φ
Output: "Satisfiable" if the formula is satisfiable, and "Unsatisfiable" otherwise

```
1.  function DPLL(T)
2.      AddClauses(cnf(e(φ)));
3.      while (TRUE) do
4.          repeat
5.              while (BCP() = "conflict") do
6.                  backtrack-level := Analyze-Conflict();
7.                  if backtrack-level < 0 then return "Unsatisfiable";
8.                  else BackTrack(backtrack-level);
9.              ⟨t, res⟩:=Deduction(T̂h(α));
10.             AddClauses(e(t));
11.         until t ≡ TRUE;
12.         if α is a full assignment then return "Satisfiable";
13.         Decide();
```

Improvements: DPLL(T)

Call theory solver to do Theory Propagation
when the SAT assignment is still partial!



Improvements: DPLL(T)

Call theory solver to do Theory Propagation
when the SAT assignment is still partial!