# COL:750/7250

## Foundations of Automatic Verification

Instructor: Priyanka Golia

Course Webpage

https://priyanka-golia.github.io/teaching/COL-750-COL7250/index.html

# Intro to SMT: Satisfiability Modulo Theory

FOL: grammar for a rational abstract thinking

FOL: Doesn't have a knowledge of any specific matter.

Theory = Subject Knowledge + FOL

Model M <D = set of natural numbers>

— we can consider only theory of natural numbers.

— we also consider the set of valid sentences over natural numbers.

For example: $\forall x \; x + 1 \neq 0$

# Intro to SMT: Satisfiability Modulo Theory

Theory = Subject Knowledge + FOL

Model M <D = set of natural numbers>

— we can consider only theory of natural numbers.

— we also consider the set of valid sentences over natural numbers.

For example: $\forall x \; x + 1 \neq 0$

# Intro to SMT: Satisfiability Modulo Theory

Is F = $\exists x, x > 0$ satisfiable? Valid ? In FOL?

Yes, it is satisfiable!

$M :< D = \mathbb{N}, I >$ F is satisfiable.

No, it is not valid, $M :< D = \mathbb{Z}^{-}, I >$

**A formula F is T-satisfiable if there is model M such that $M \vDash T \cup F$.**

**We write T -satisfiability as $M \vDash_T F$.**

T: set of true sentences in arithmetic over natural numbers.

Is $T \cup F$ satisfiable ?, we need to restrict our domain to set of natural numbers, and assume the knowledge of natural number arithmetic like $\forall x\ x > 0, \forall x\ x + 1 \neq 0$

Yes, it is satisfiable!

$M \vDash_T F$

If formula is satisfiable, gives an satisfying assignment

Formulas in different theories —> SMT

(Linear integer arithmetic,

Linear real arithmetic, bit vectors, strings)

Unsatisfiable

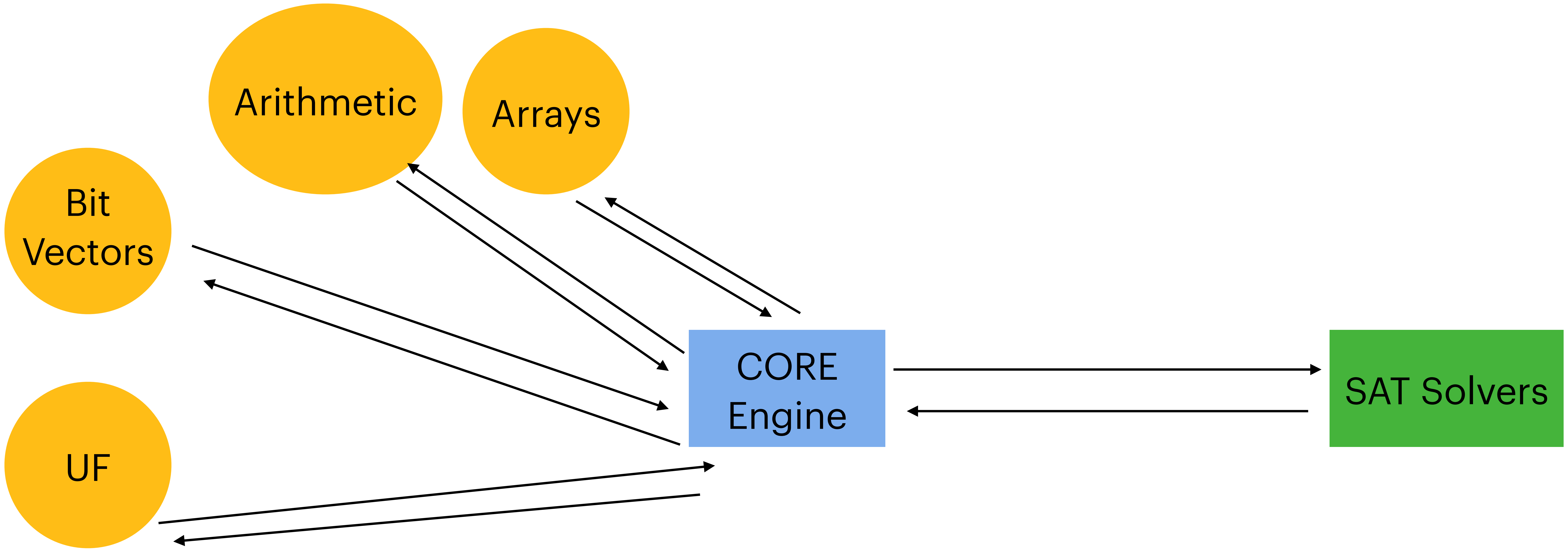
Chaff SAT Solver — 2000 (DPLL + conflict analysis, heuristics)

Order of magnitude faster than previous SAT solvers

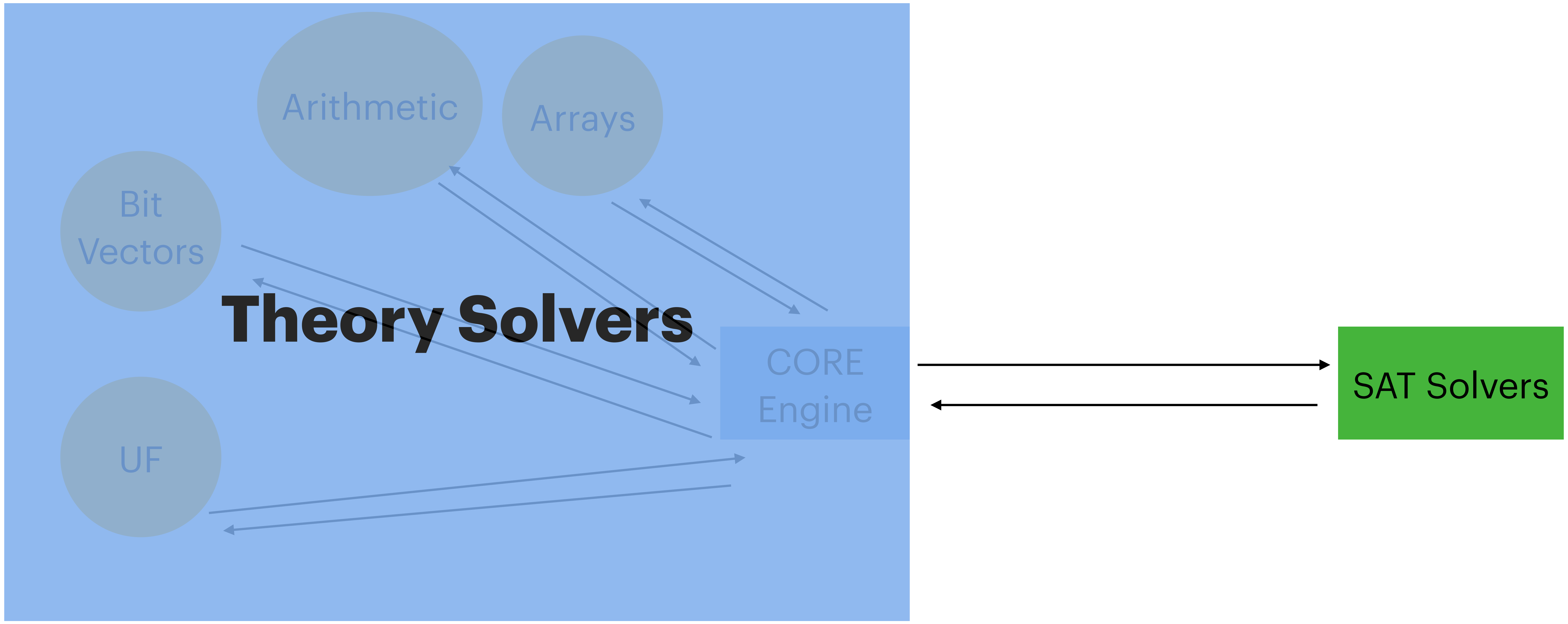
Many real-world problems don't exhibit worst case theoretical performance

Alto, 2001, came up with idea of combining SAT solvers with decision procedures for decidable first-order theories.

SVC, CVC, Yices solver came to picture — first SMT solver was born!!!




# SMT solvers


Arithmetic
Arrays
Bit Vectors
Theory Solvers
CORE Engine
UF
SAT Solvers

# SMT solvers

# Theory Solvers

Theory Solver: Difference Logic

Difference logic — the satisfiability of a conjunction of arithmetic atoms.

Each atom is of the form $x - y \oplus c$,
where x and y are variables, c is a numeric constant, and
$\oplus \in \{ <, >, \leq, \geq, = \}$

The variables can range over either the integers (QF_IDL) or the reals (QF_RDL).

# Theory Solver: Difference Logic

The first step is to rewrite everything in terms of $\leq$

$$x - y = c \equiv (x - y \leq c) \wedge (y - x \leq -c)$$

$$x - y \geq c \equiv y - x \leq -c$$

$$x - y < c \quad \equiv x - y \leq c - 1$$ For integers

$$\equiv x - y \leq c - \delta$$ For reals

$$x - y > c \quad \equiv y - x < -c$$

# Theory Solver: Difference Logic

- A conjunction of literals, all of the form $x - y \leq c$.
- From these literals, we form a weighted directed graph with a vertex for each variable.
- For each literal $x - y \leq c$, there is an edge $y \rightarrow x$, with weight c.
- The set of literals is satisfiable iff there is no cycle for which the sum of the weights on the edges is negative.
- There are a number of efficient algorithms for detecting negative cycles in graphs

$$(x - y = 5) \land (z - y \geq 2) \land (z - x > 2) \land (w - x = 2) \land (z - w < 0)$$

# Theory Solvers

### Linear Arithmetic Solver

Handles inequalities and equalities over integers or real numbers:

Techniques: Fourier-Motzkin elimination, Simplex algorithm.

Check if $(x + 2y \leq 10) \wedge (x - y \geq 3)$ ?

### Bit-Vector Solver

Deals with fixed-width integers and bitwise operations:

Techniques: Bit-blasting (reducing bit-vector problems to SAT), word-level reasoning

Check if $x > > 4 = 0x0A$

# Theory Solvers

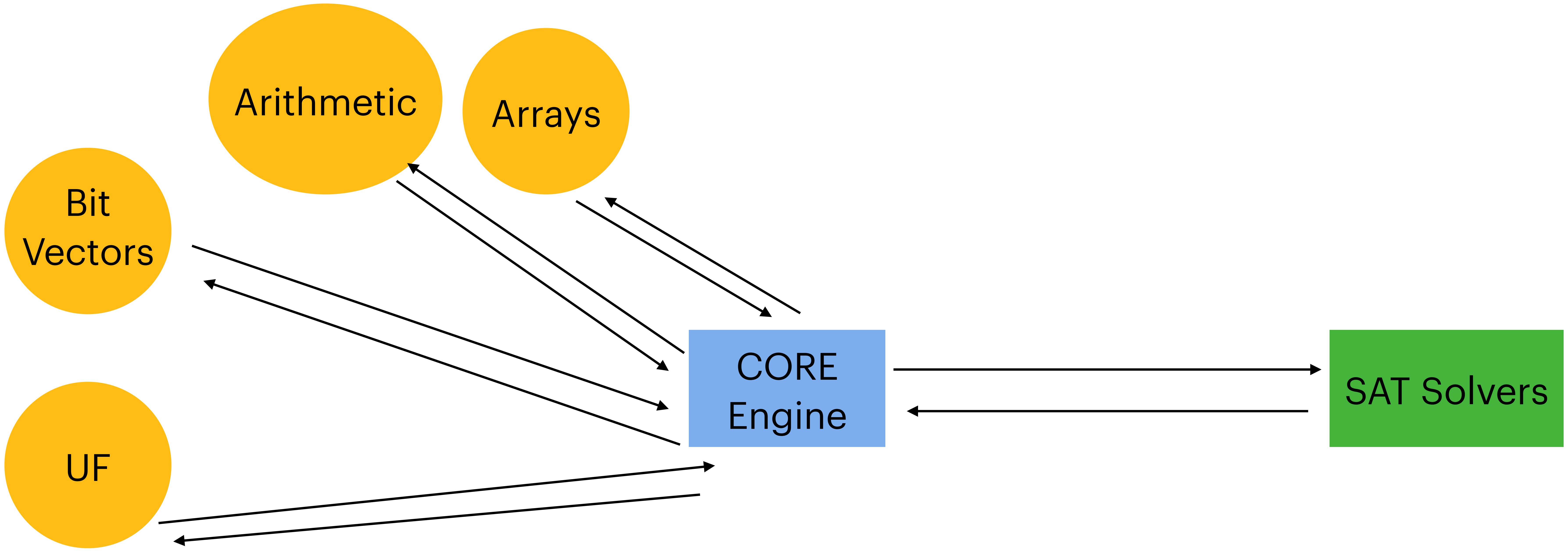
## Theory Propagation

Deducing new constraints or facts based on existing ones.
For example, in linear integer arithmetic:

given $(x \geq 5) \wedge (y = x + 2)$, we can deduce $y >= 7$

## Theory Consistency Checking

Check if a set of constraints is consistent within the theory.
If not, it provides a conflict (a minimal subset of constraints that are unsatisfiable)




# SMT solvers

# SMT Solvers

Two main approaches:

1. "Eager" approach
    1. Translate into an equisatisfiable propositional formula
    2. Feed it to any SAT solver

   UCLID

2. "Lazy" approach
    1. Abstract the input formula to a propositional formula
    2. Feed it to a SAT solver
    3. Use a theory solver to refine the formula and guide the SAT solver

   Cvc5, z3, MathSAT, OpenSMT

# SMT solving — Lazy Approach

Theory: Equality with Uninterpreted Functions

$$(g(a) = c) \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge (c \neq d)$$

$p_1$ $\neg p_2$ $p_3$ $\neg p_4$

Send $(p_1 \wedge (\neg p_2 \vee p_3) \wedge \neg p_4)$ to a SAT solver.

SAT solver returns $\sigma = \{p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 0\}$

Theory solver checks if $\sigma$ is consistent or not!!

$\sigma$ is not consistent, Theory solver returns UNSAT. Add $\neg\sigma$ as a clause.

Send $(p_1 \wedge (\neg p_2 \vee p_3) \wedge \neg p_4) \wedge (\neg p_1 \vee p_2 \vee p_3 \vee p_4)$ to a SAT solver.

# SMT solving — Lazy Approach

$$\underbrace{(g(a) = c)}_{p_1} \land (\underbrace{f(g(a)) \neq f(c)}_{\neg p_2} \lor \underbrace{g(a) = d}_{p_3}) \land \underbrace{(c \neq d)}_{\neg p_4}$$

Send $(p_1 \land (\neg p_2 \lor p_3) \land \neg p_4)$ to a SAT solver. $\sigma \models F$ $\sigma = \{p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 0\}$

$\sigma$ is not consistent, Theory solver returns UNSAT. Add $\neg\sigma$ as a clause.

Send $(p_1 \land (\neg p_2 \lor p_3) \land \neg p_4) \land (\neg p_1 \lor p_2 \lor p_3 \lor p_4)$. $\sigma = \{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

$\sigma$ is not consistent, Theory solver returns UNSAT. Add $\neg\sigma$ as a clause.

Send $(p_1 \land (\neg p_2 \lor p_3) \land \neg p_4) \land (\neg p_1 \lor p_2 \lor p_3 \lor p_4) \land (\neg p_1 \lor \neg p_2 \lor \neg p_3 \lor p_4)$

At last SAT Solver returns UNSAT, the original formula in UF is UNSAT

# SMT solving — Lazy Approach Enhancements

SAT solvers checks for satisfying assignment and returns $\sigma$

Checks for partial assignment M, and returns M.

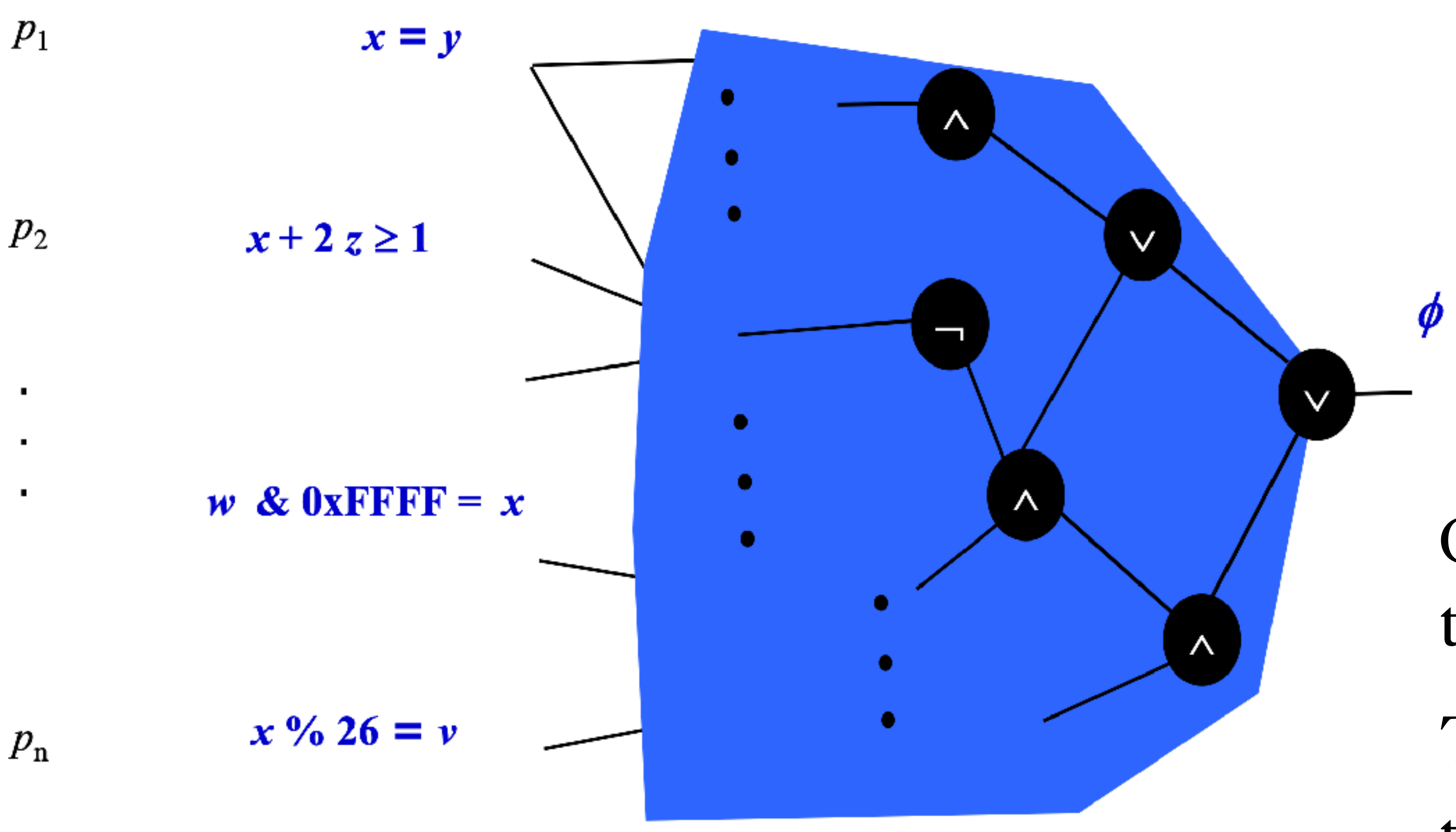
If M/($\sigma$) is T-unsatisfiable, add $\neg M$ as a clause

Identify a T-unsatisfiable subset $M_o$ of $M$, and $\neg M_o$as a clause

In our previous example, we could have added

$(\neg p_1 \vee p_2 \vee p_4)$ instead of $(\neg p_1 \vee p_2 \vee p_3 \vee p_4)$

Backtrack to a point where M was still T-Satisfiable,
use this to pass more explanation to SAT solver.




Can have combinations of theories!

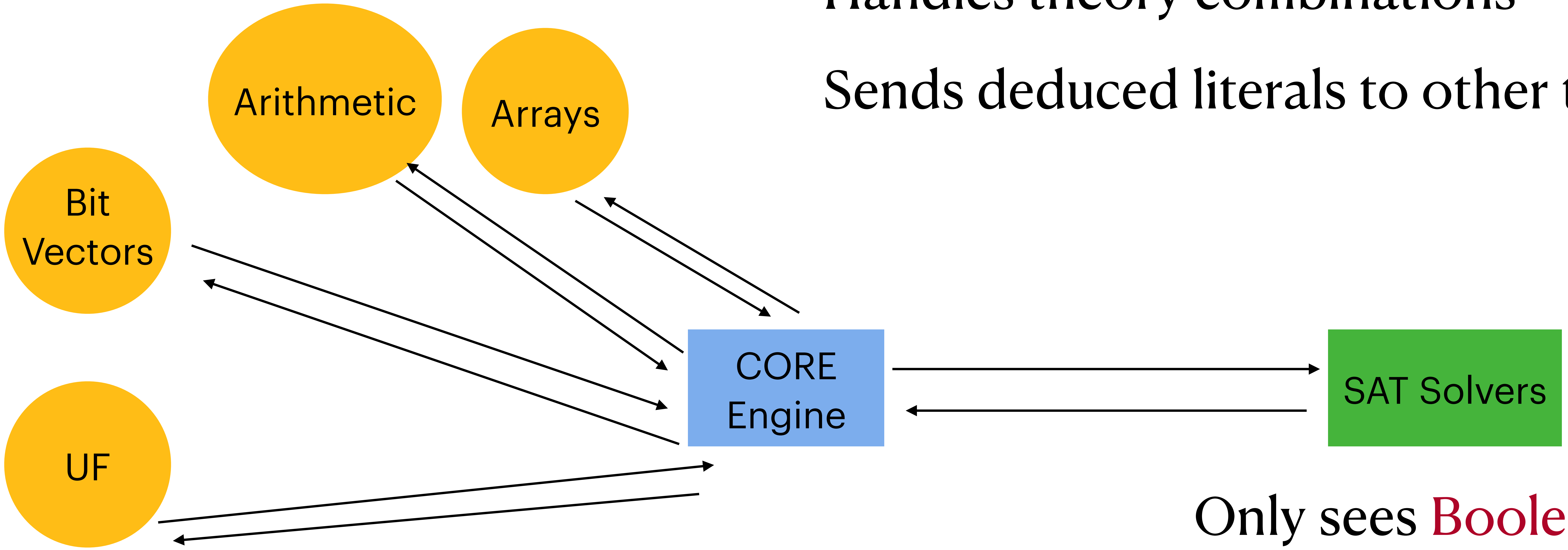
Task is to find an assignment to $Vars(\phi)$ such that $\phi$ is satisfiable!

# SMT solvers

Sends each assertions to the appropriate theory

Handles theory combinations

Sends deduced literals to other theories/SAT solver

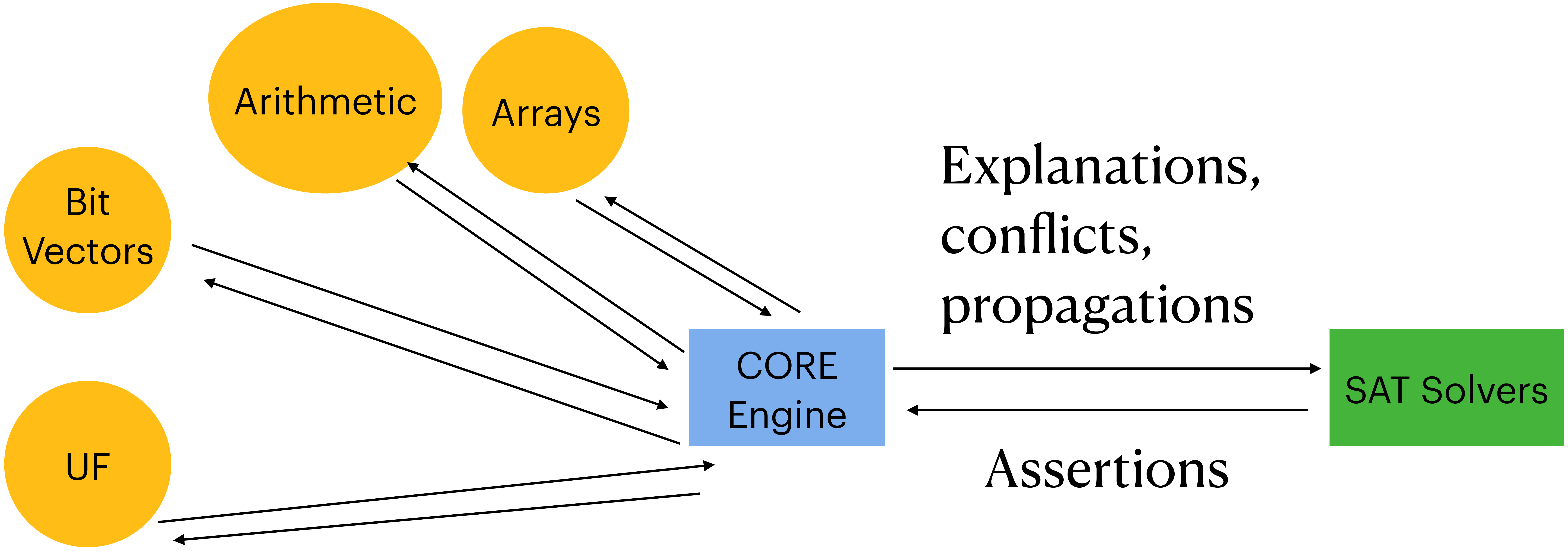



Only sees Boolean Skeleton of the problem!

Theory Solvers!
Decide T-satisfiability of a conjunction of literals.

Builds partial model by assigning truth values to literals

Sends these literals to the core as assertions


Arithmetic
Arrays
Bit Vectors
UF
CORE Engine
Explanations, conflicts, propagations
SAT Solvers
Assertions

# SMT solvers

# From SAT & SMT to Temporal Logic

SAT: Checks whether a propositional formula is satisfiable.

SMT: Extends SAT with richer theories (e.g., arithmetic, arrays).

But What About Time?

SAT/SMT/FOL verify properties in static systems.

Many real-world systems evolve over time (e.g., software, robots, protocols).

"A robot should always eventually return to its charging station."
"A user who enters a correct password will eventually get access."
"How can we verify that a system never reaches an error state?"

Can we express this in SAT or FOL?

# From SAT & SMT to Temporal Logic

Classical logic (SAT/SMT) = Static Reasoning

Temporal logic = Reasoning over time

Linear Temporal Logic (LTL)  Assumes a single timeline (one possible sequence of events).

**Next Class: Linear Temporal Logic (LTL)**

Course Webpage

Thanks!