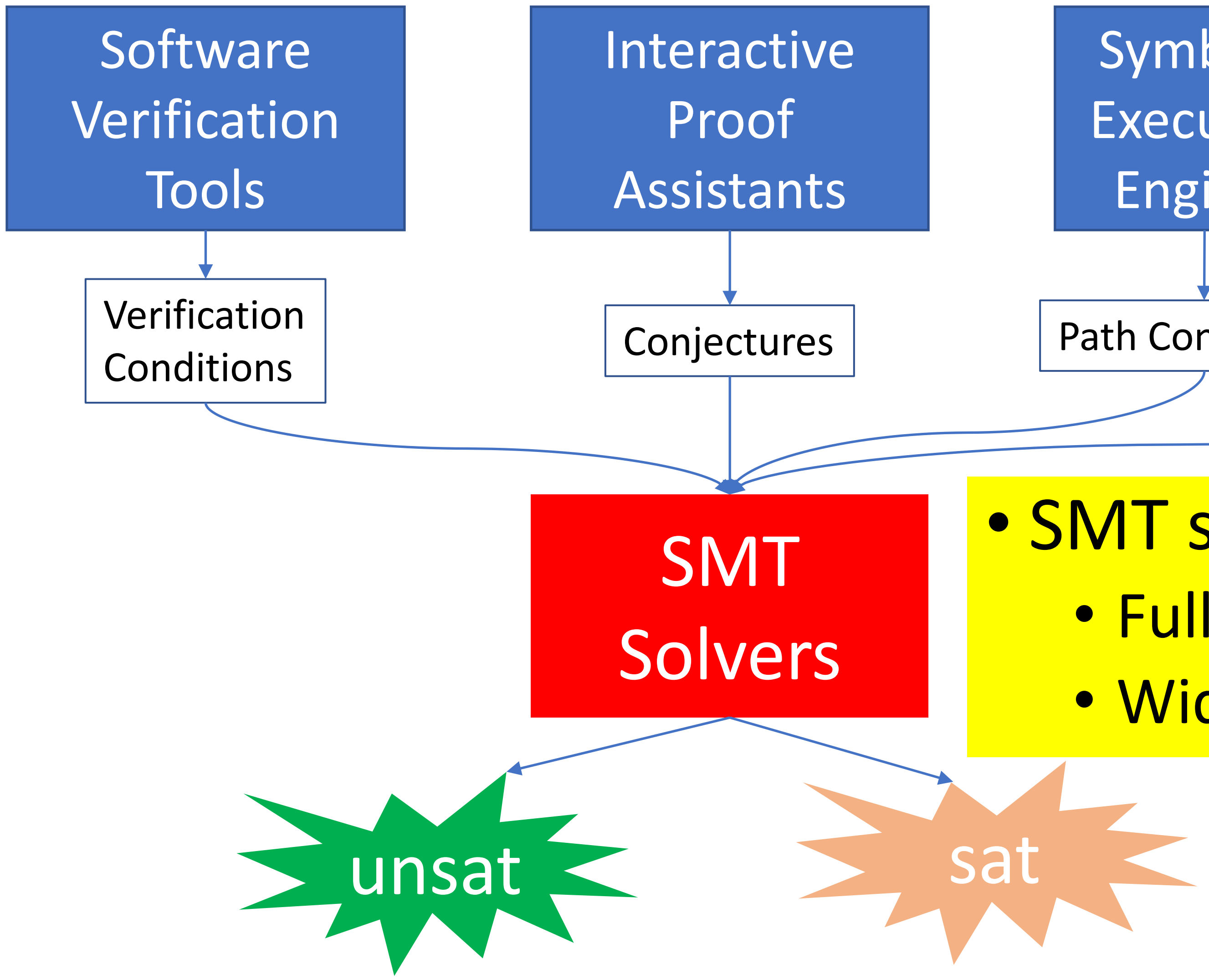
An Overview of Quantifier Instantiation in Modern SMT Solvers

Andrew Reynolds

May 28, 2021

The University of Iowa

Satisfiability Modulo Theories (SMT) Solvers



- SMT solvers are:
  - Fully automated reasoners
  - Widely used in applications

SMT Solvers

- Traditionally:
  - Efficient decision procedures for *quantifier-free* constraints over theories:
    - Arithmetic
    - Uninterpreted functions (UF)
    - Bitvectors
    - Arrays
    - Datatypes
    - More recently: strings, floating points, sets, relations, …
- In the past decade:
  - Efficient (heuristic) techniques for *quantified* formulas as well

    $\Rightarrow$ Focus of this talk

Applications of ∀ in SMT

- Are used for:
  - **Automated theorem proving**:
    - Background axioms $\{\forall x.\, g(e,x)=g(x,e)=x,\ \forall x.\, g(x,g(y,z))=g(g(x,y),x),\ \forall x.\, g(x,i(x))=e\}$
  - **Software verification**:
    - Unfolding $\forall x.\, foo(x)=bar(x+1)$, code contracts $\forall x.\, pre(x) \Rightarrow post(f(x))$
    - Frame axioms $\forall x.\, x \neq t \Rightarrow A'(x)=A(x)$
  - **Function Synthesis**:
    - Synthesis conjectures $\forall i{:}input.\, \exists o{:}output.\, R[o,i]$
  - **Planning**:
    - Specifications $\exists p{:}plan.\, \forall t{:}time.\, F[P,t]$

SMT Solvers for ∀ using Quantifier Instantiation

- Traditionally:
  - E-matching **[Detlefs et al 2005, Bjorner et al 2007, Ge et al 2007]**

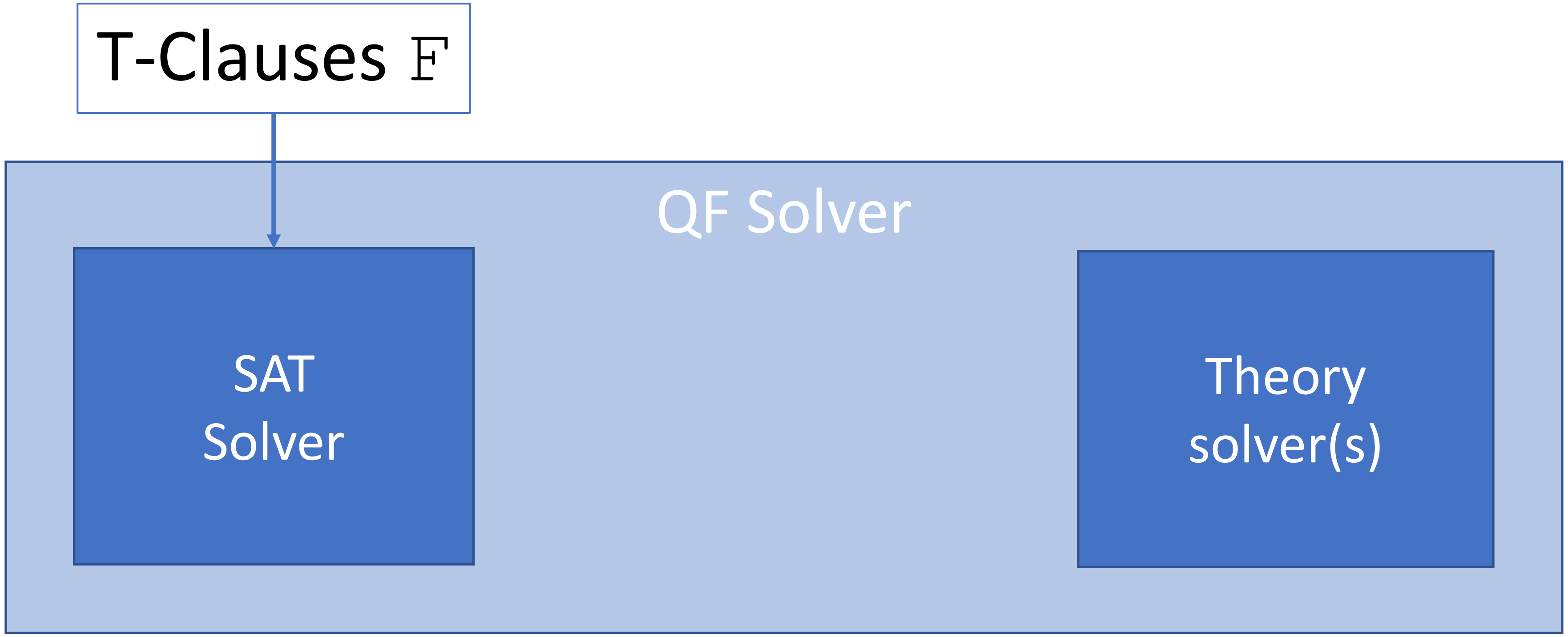
<u>Implemented in</u>

simplify, z3, FX7,
Alt-Ergo, Princess,
cvc5, veriT, SMTInterpol

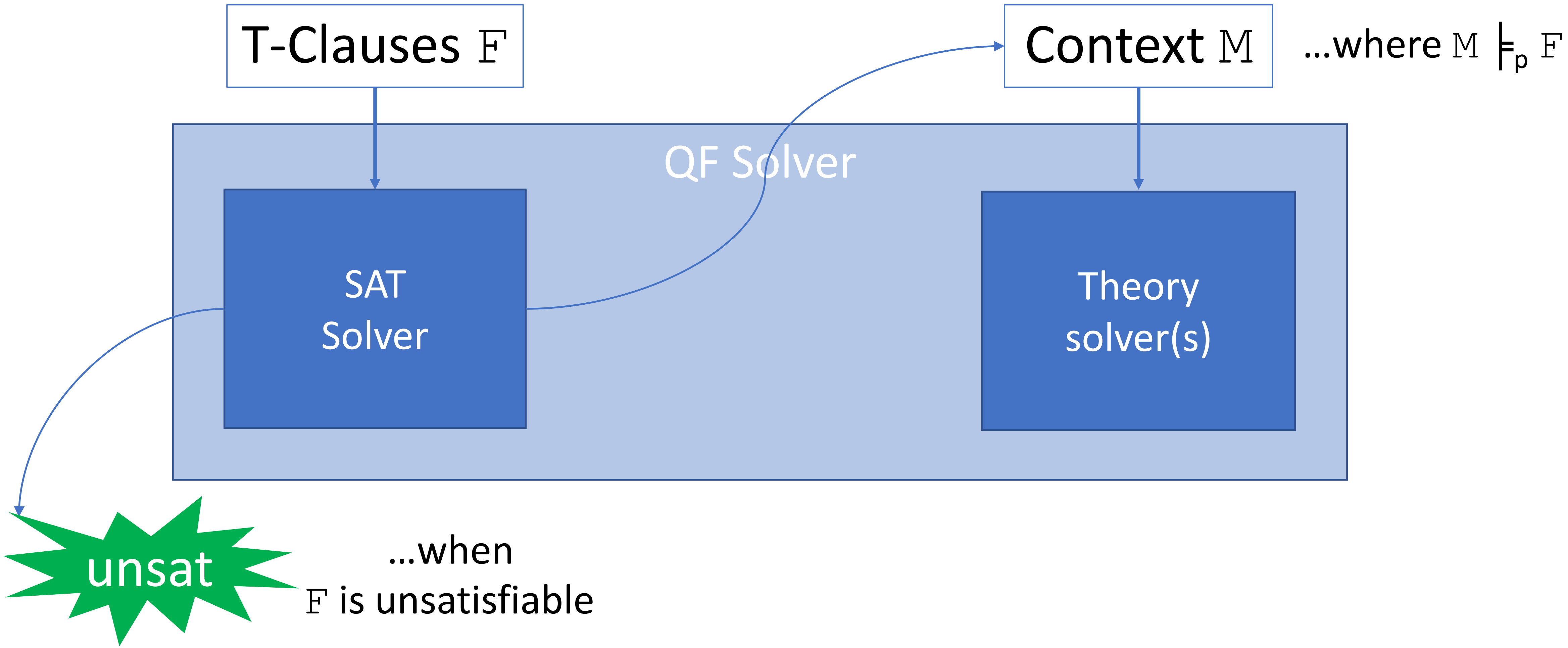
SMT Solvers for $\forall$ **using Quantifier Instantiation**

	Implemented in
• Traditionally:	
• E-matching [Detlefs et al 2005, Bjorner et al 2007, Ge et al 2007]	simplify, z3, FX7, Alt-Ergo, Princess, cvc5, veriT, SMTInterpol
• More recently:	
• Conflict-Based Instantiation [Reynolds et al 2014, Barbosa et al 2017]	cvc5, veriT, SMTInterpol
• Model-Based Instantiation [Ge et al 2009, Reynolds et al 2013]	z3, cvc5
• Enumerative Instantiation [Reynolds et al 2018]	cvc5, veriT
• Counterexample-Guided / QE [Reynolds et al 2015, Janota et al 2015]	z3, cvc5, yices
• Syntax-Guided [Preiner et al 2017, Niemetz et al 2021]	boolector, cvc5

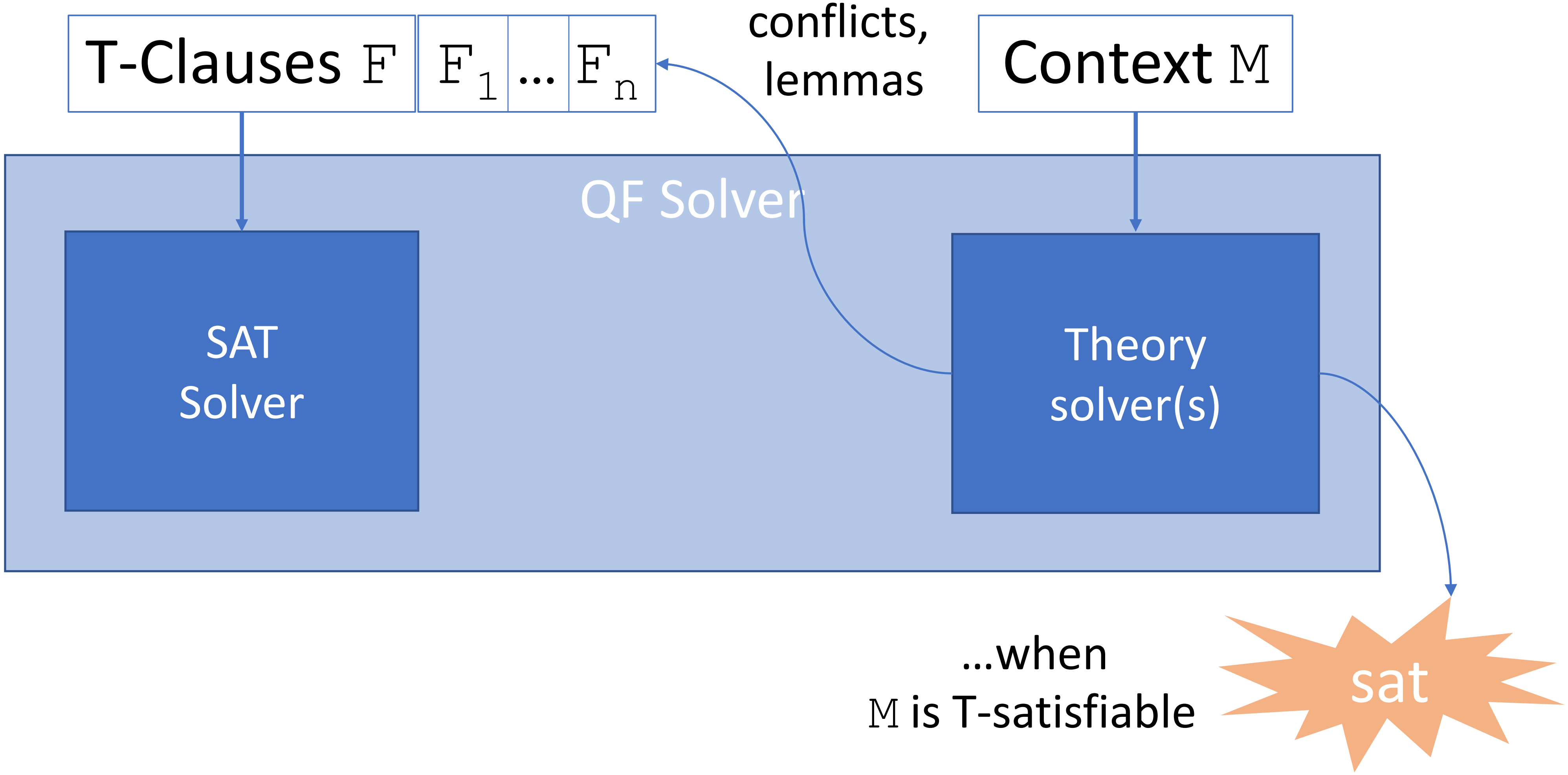
DPLL(T)-Based SMT Solvers (quantifier-free)



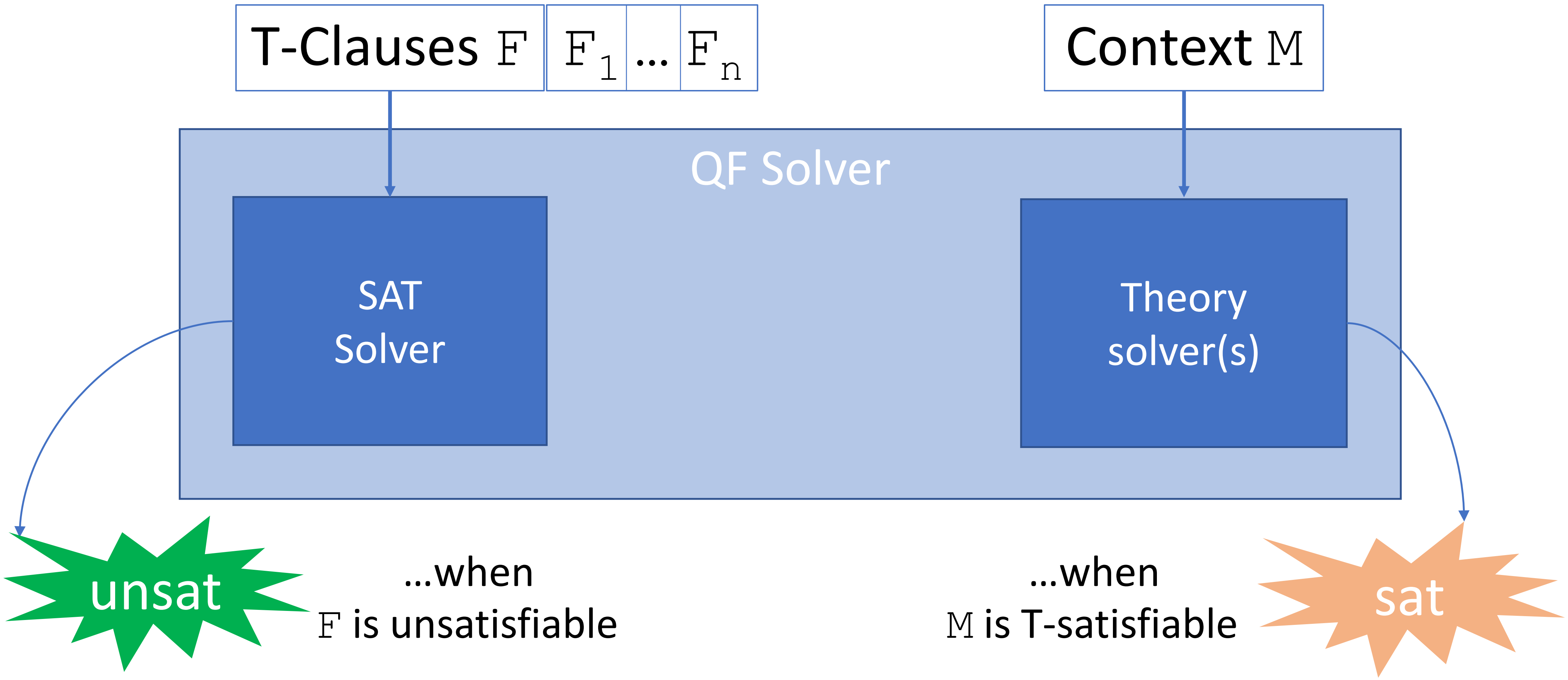
DPLL(T)-Based SMT Solvers



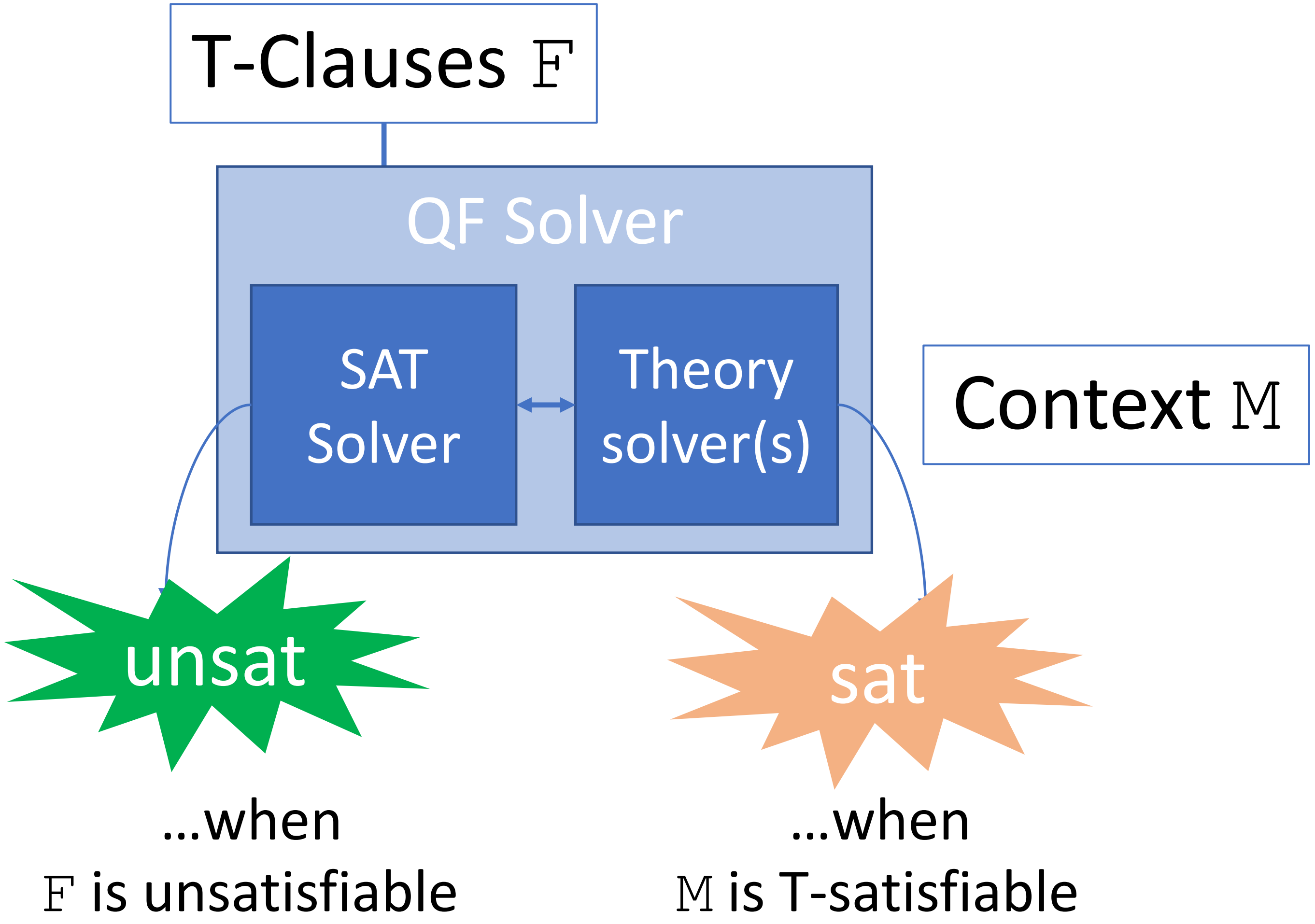
DPLL(T)-Based SMT Solvers



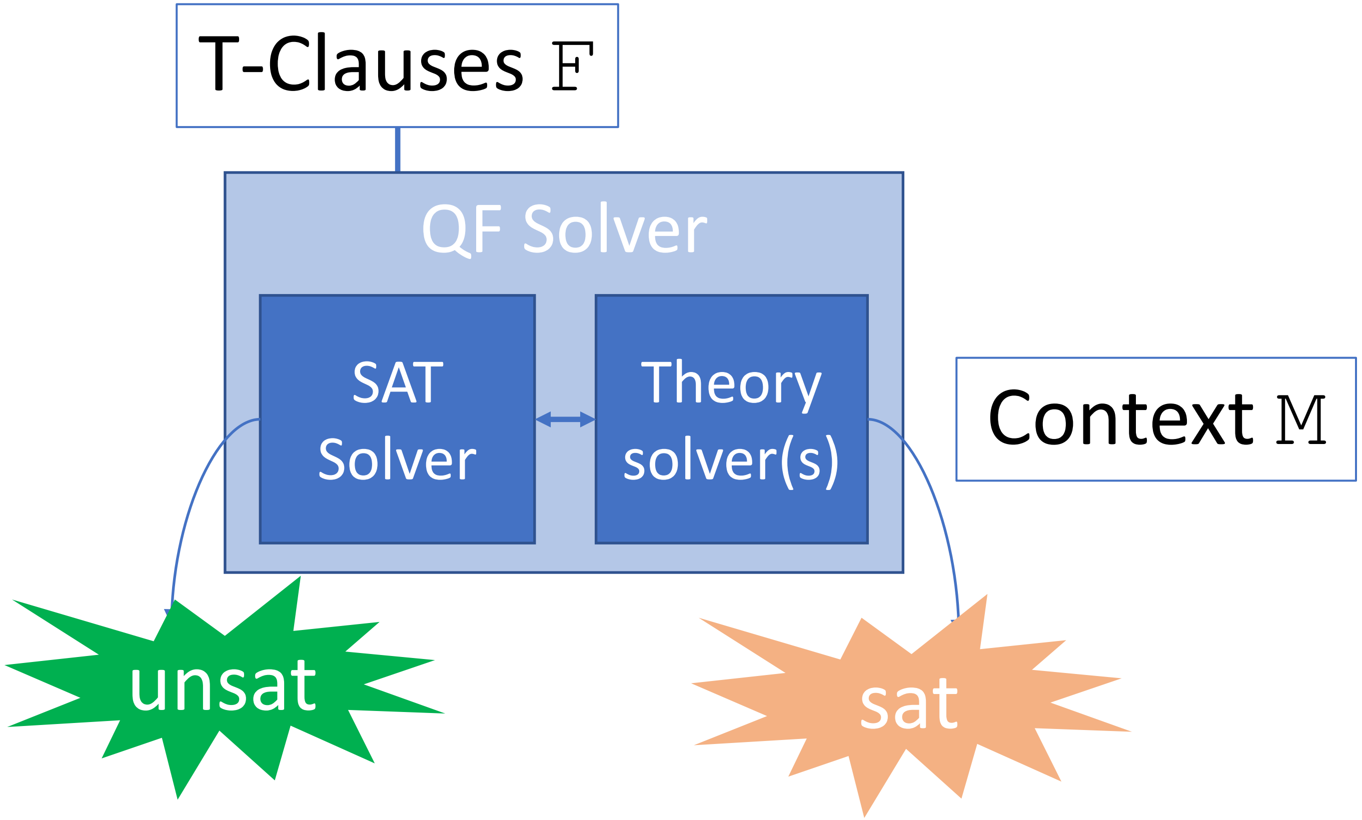
DPLL(T)-Based SMT Solvers



DPLL(T)-Based SMT Solvers + ∀ Instantiation



DPLL(T)-Based SMT Solvers + ∀ Instantiation

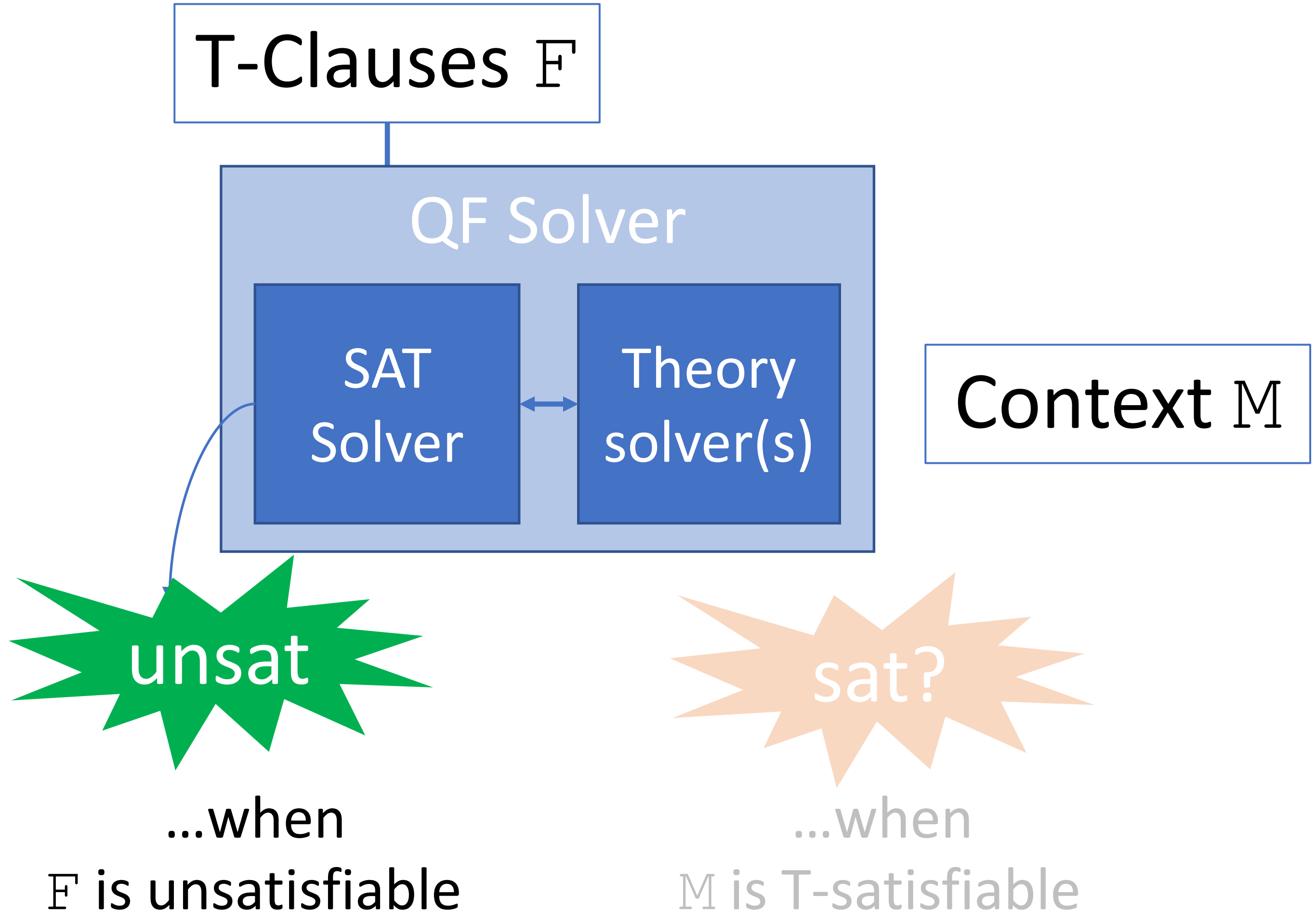


...when
$\mathtt{F}$ is unsatisfiable

...when
$\mathtt{M}$ is T-satisfiable

When $\mathtt{M}$ contains quantified formulas ∀...

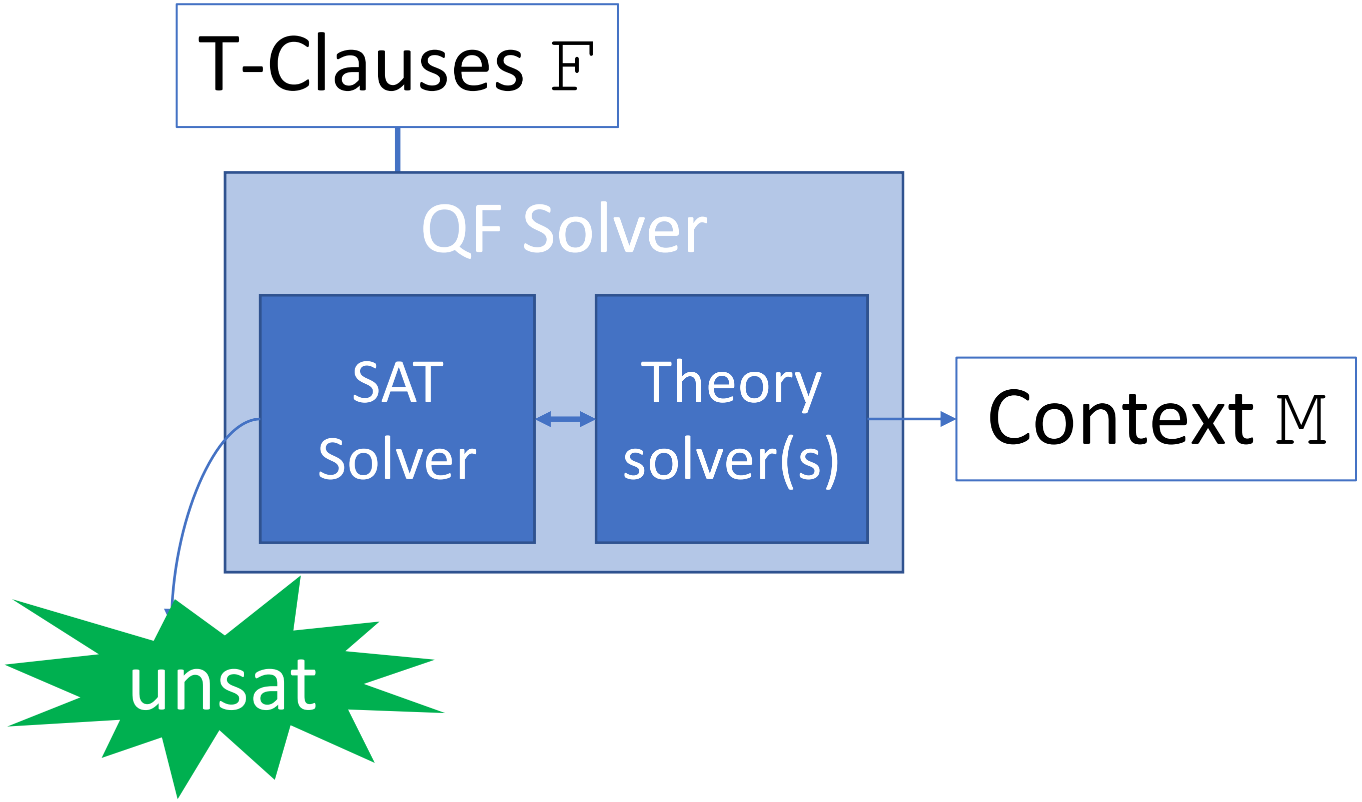
DPLL(T)-Based SMT Solvers + ∀ Instantiation



Undecidability!

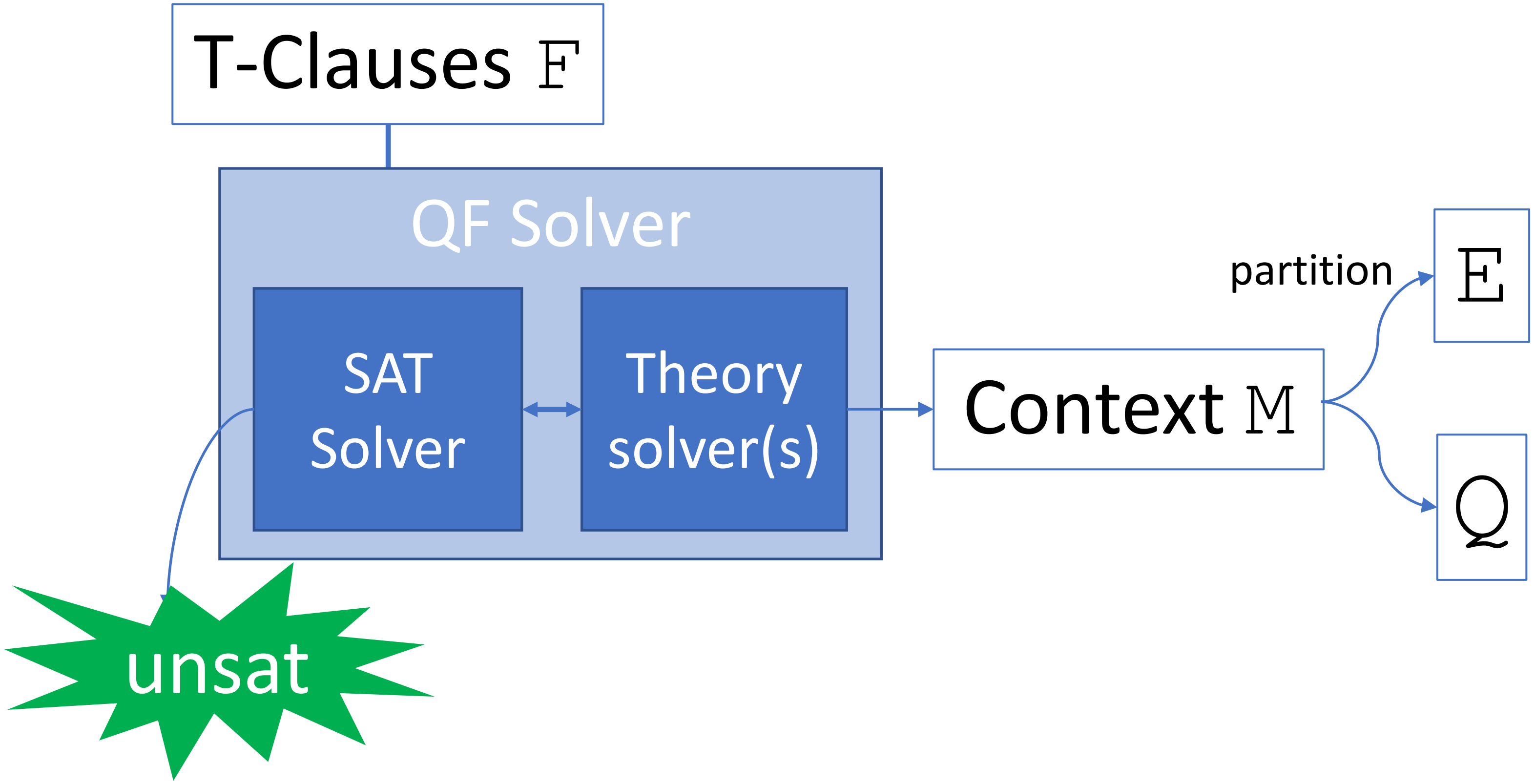
...cannot always establish $\mathbb{M}$ is sat

DPLL(T)-Based SMT Solvers + ∀ Instantiation



…when
$\mathbb{F}$ is unsatisfiable

DPLL(T)-Based SMT Solvers + ∀ Instantiation



...when
F is unsatisfiable

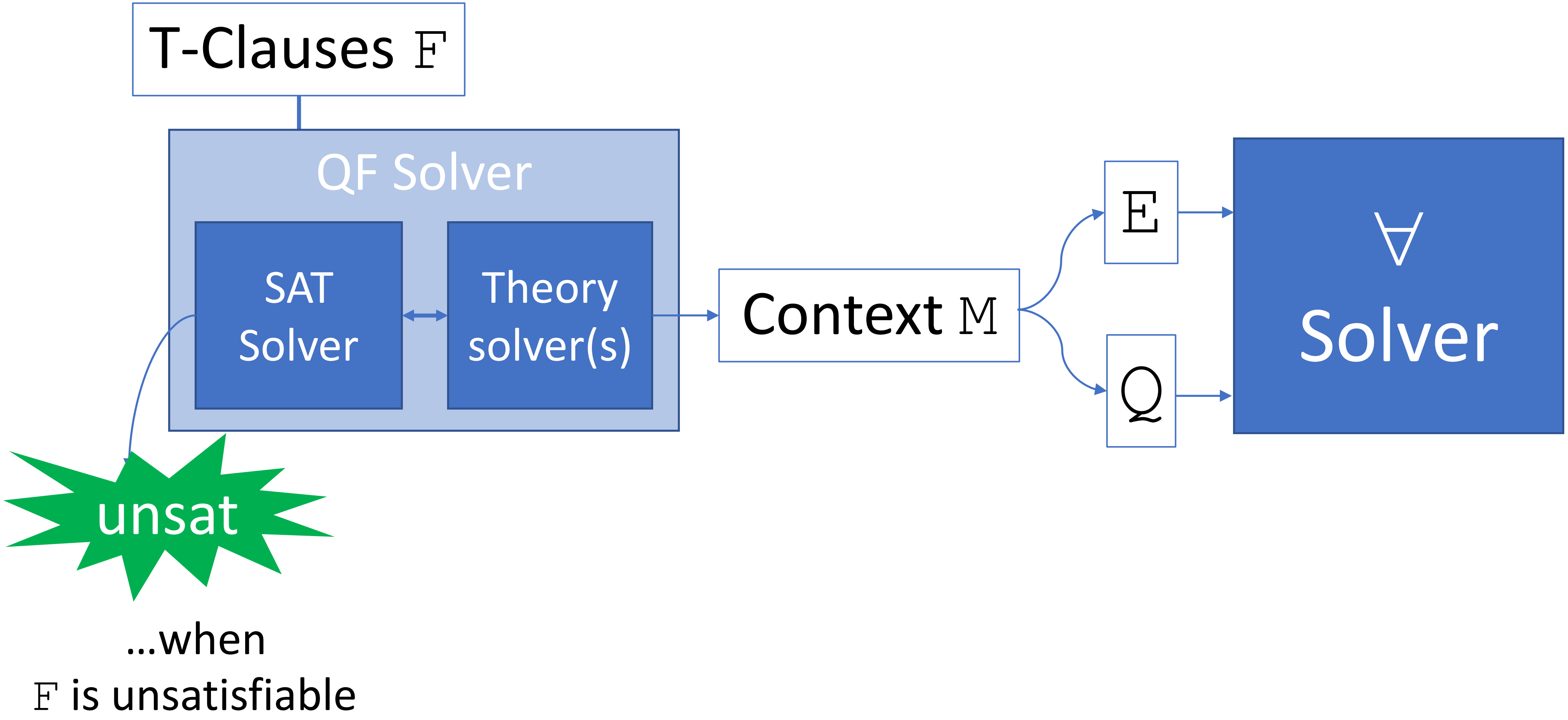
Set of ground formulas

- {f(a)=b, P(a), ...}

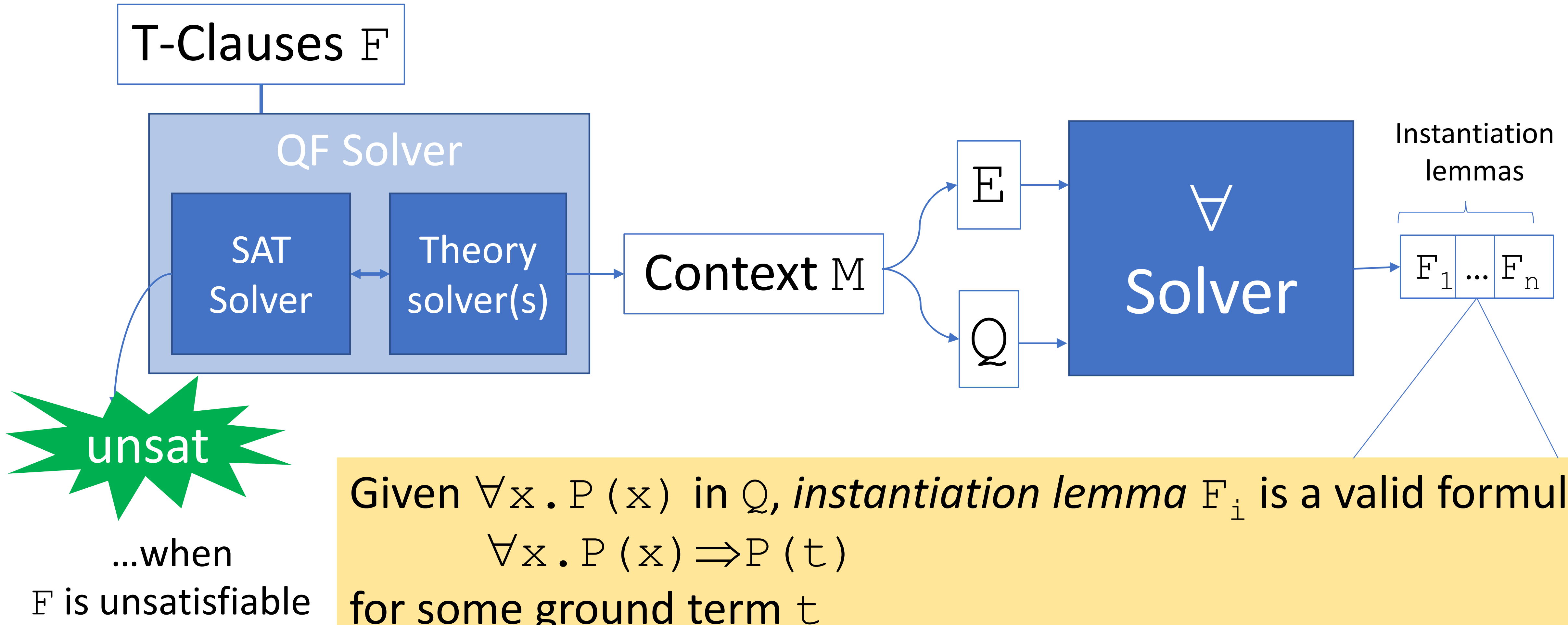
Set of quantified formulas

- {∀x.P(x), ...}

DPLL(T)-Based SMT Solvers + ∀ Instantiation



DPLL(T)-Based SMT Solvers + ∀ Instantiation

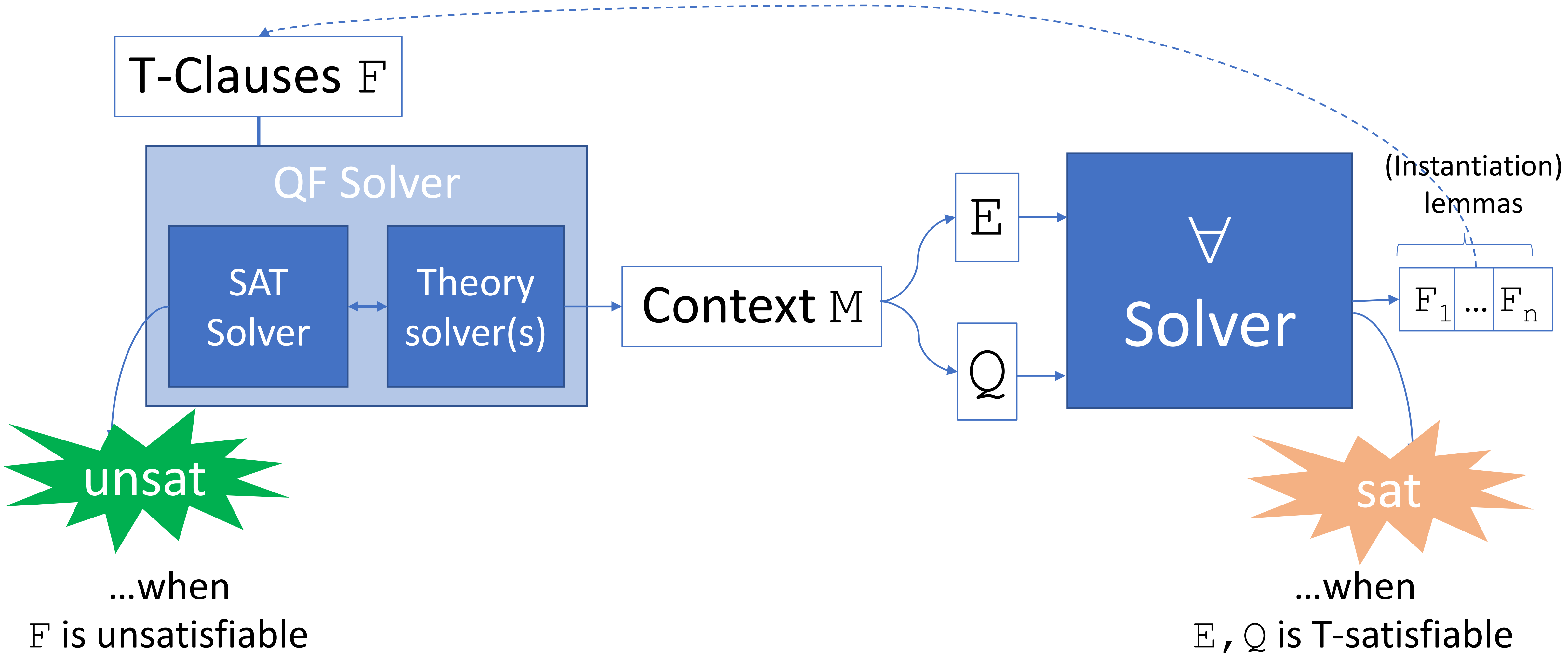


Given $\forall x.P(x)$ in Q, *instantiation lemma* F_i is a valid formula:

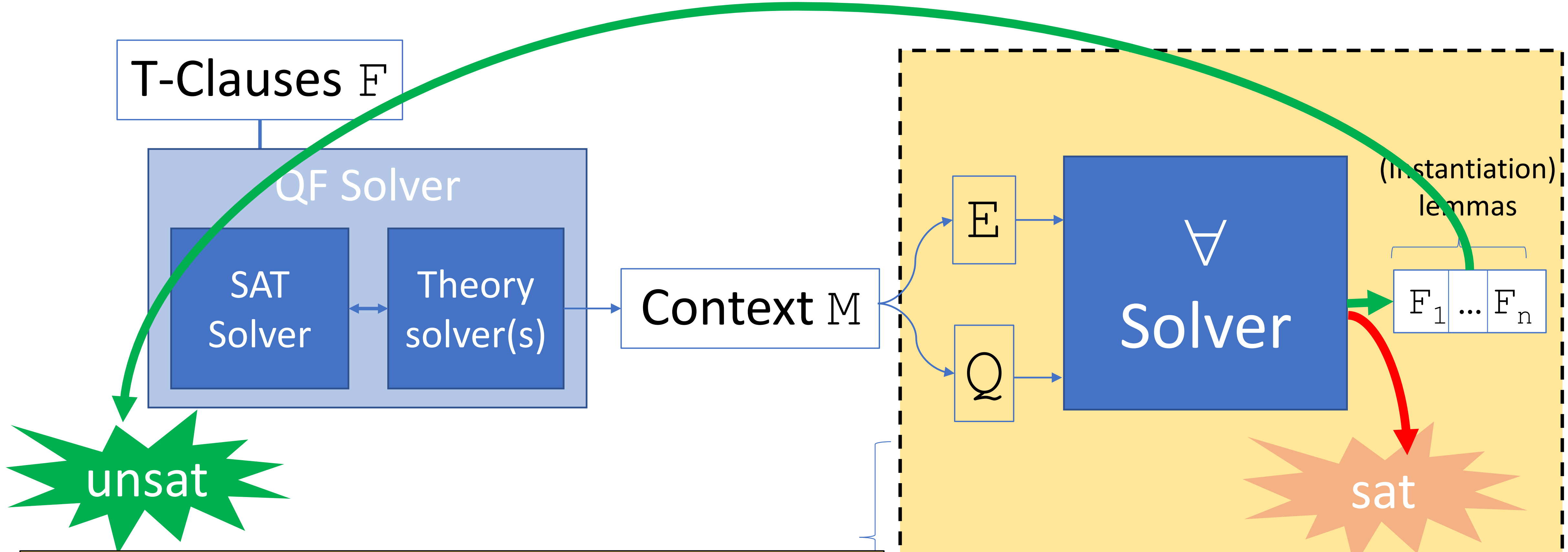
$$\forall x.P(x) \Rightarrow P(t)$$

for some ground term t

DPLL(T)-Based SMT Solvers + ∀ Instantiation

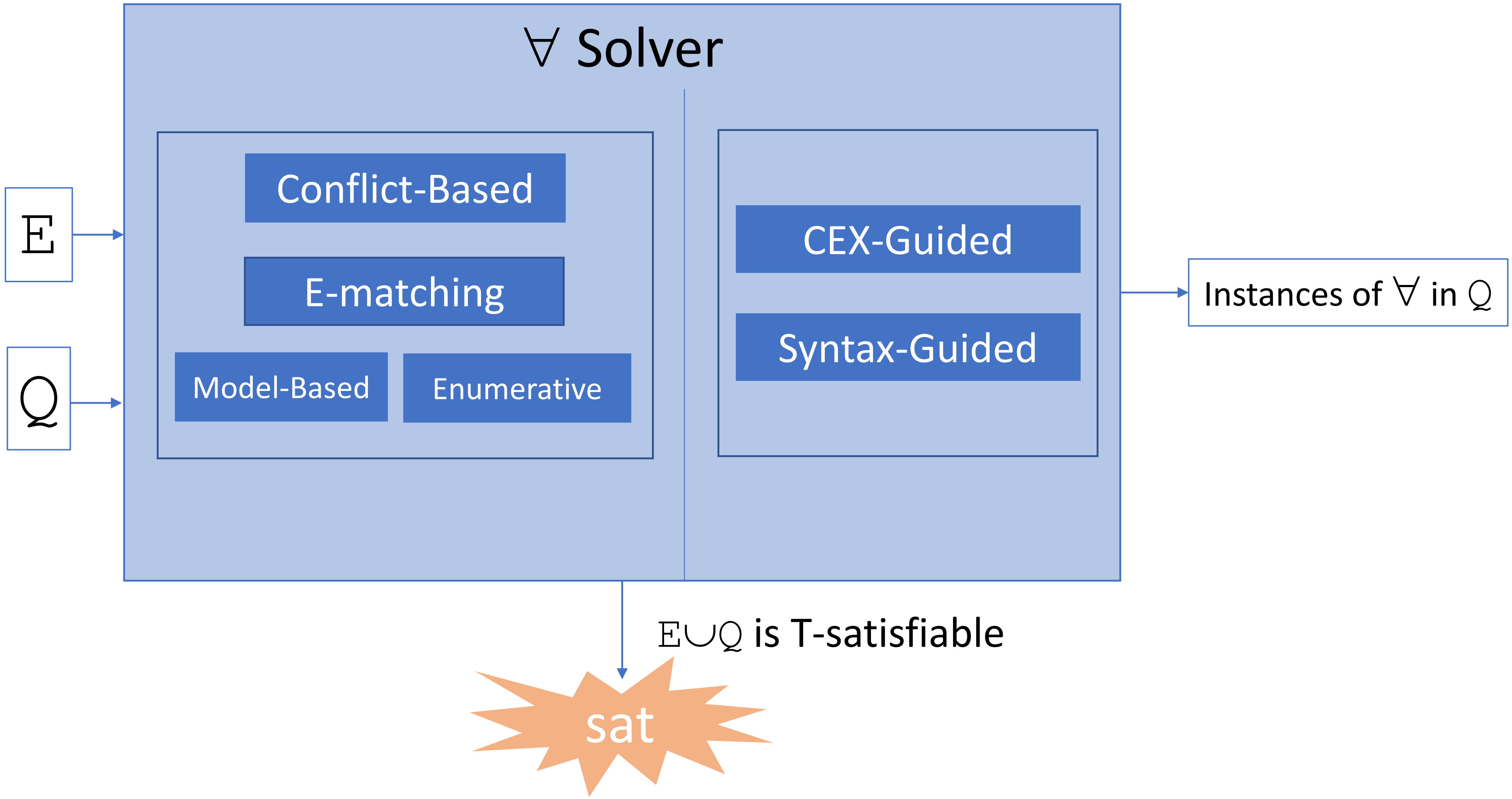


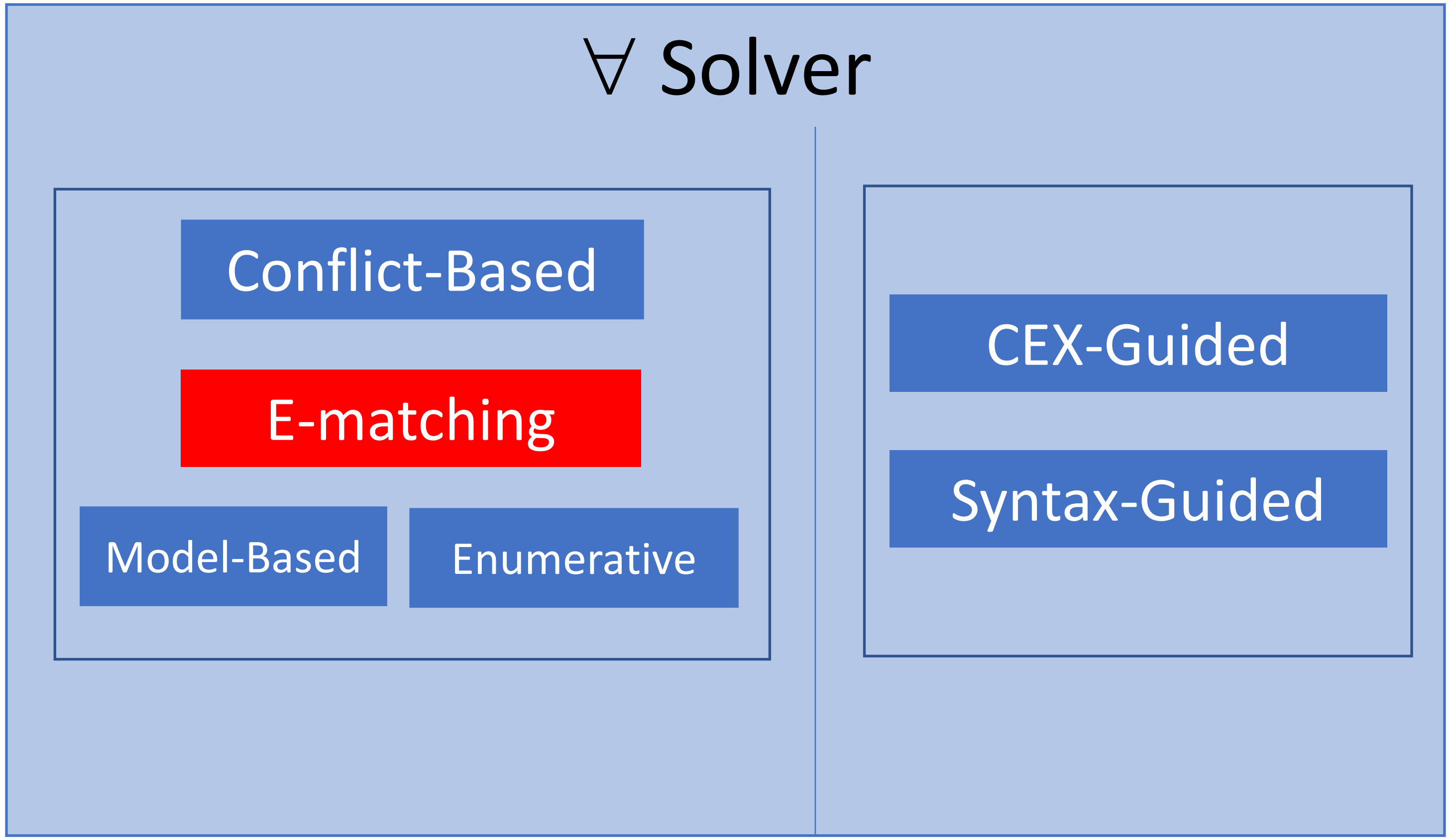
DPLL(T)-Based SMT Solvers + ∀ Instantiation



- Which lemmas are likely lead to "**unsat**"?
- When can we answer "**sat**"?

Techniques for Quantifier Instantiation

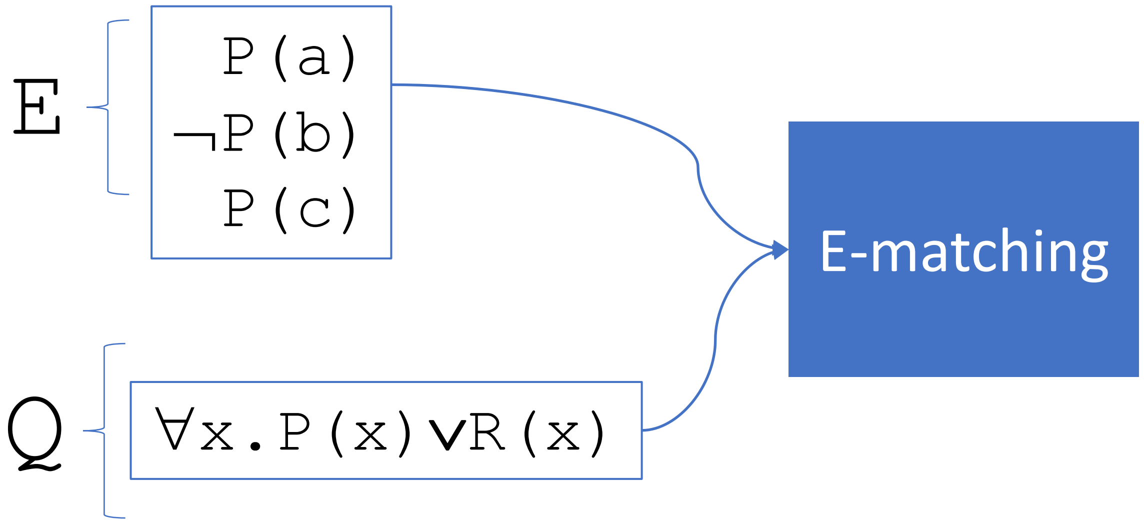



∀ Solver
Conflict-Based
E-matching
Model-Based
Enumerative
CEX-Guided
Syntax-Guided

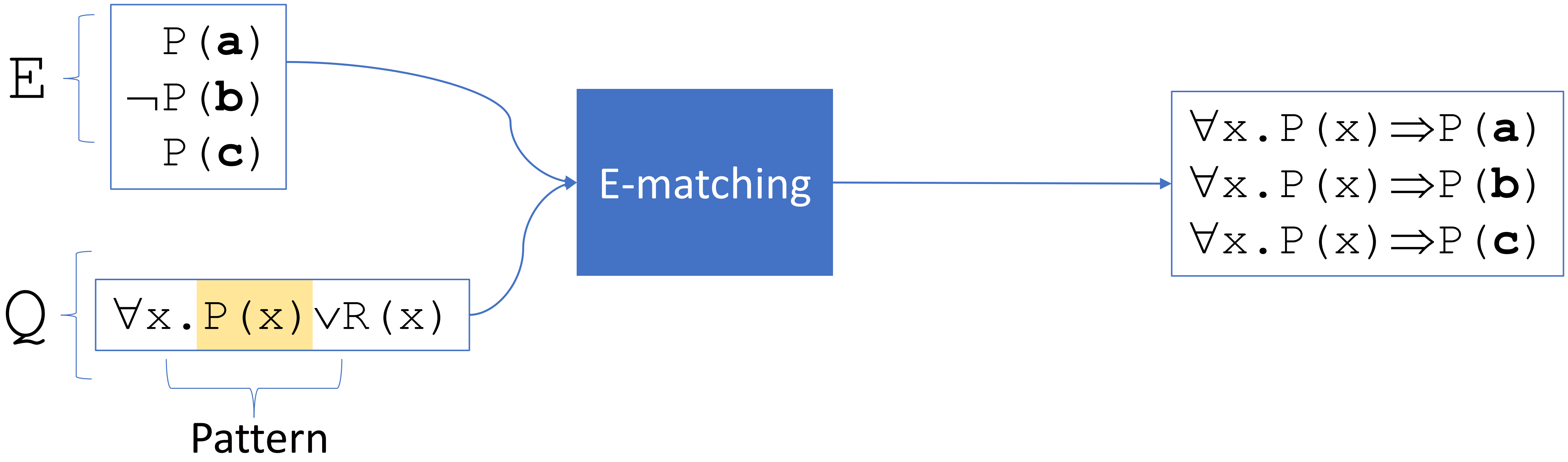
E-matching

- **Idea:** Instantiations found by pattern matching $\mathbb{Q}$ to terms from $\mathbb{E}$
- Implemented in early SMT solvers (e.g. simplify) as well as z3, cvc5
    - **[Detlefs et al 2005, Bjorner et al CADE 2007, Ge et al CAV 2007]**
- **Key applications:** Software verification
    - Example: Dafny/Boogie

E-matching

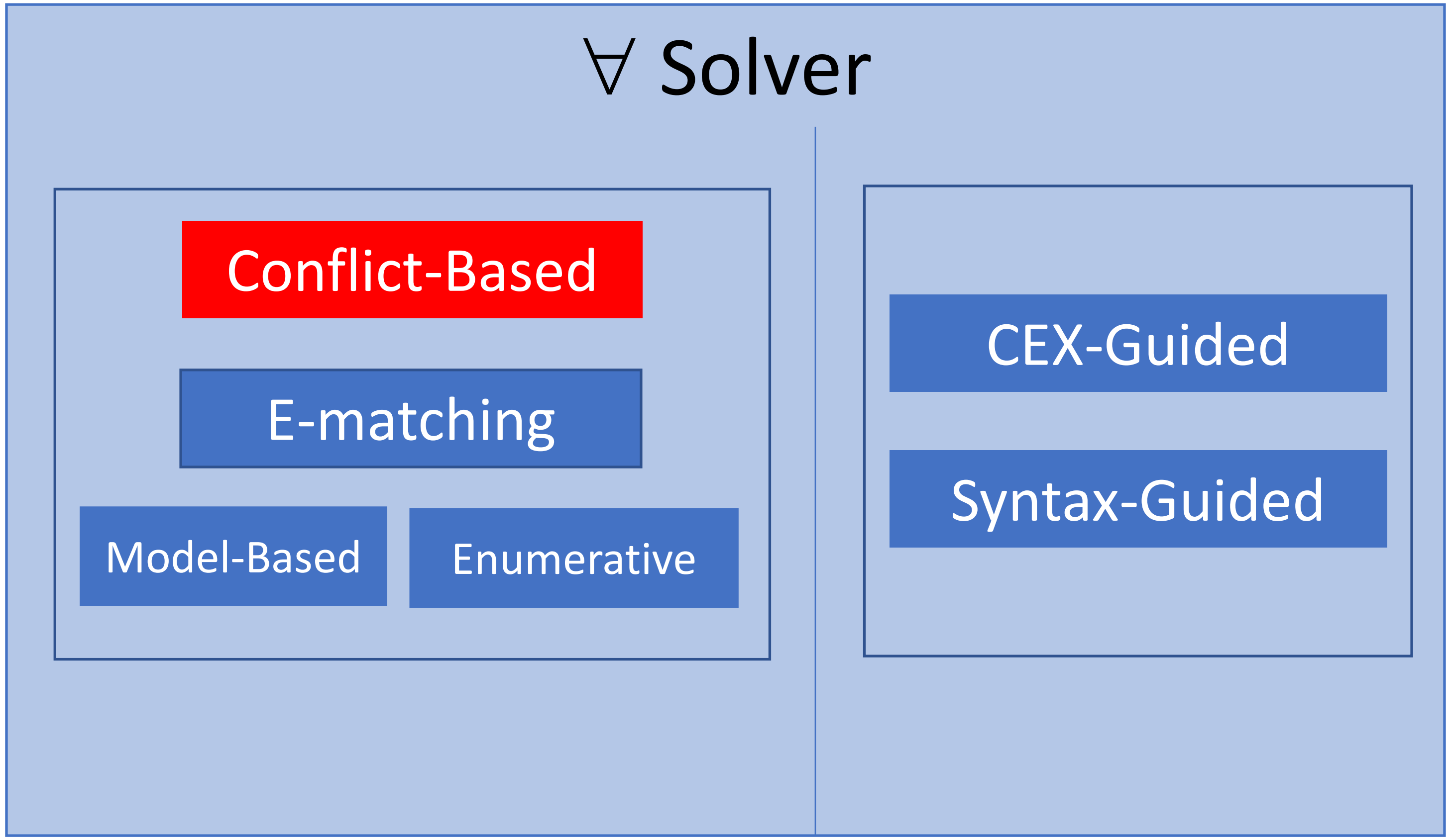


E-matching



E-matching: Impact

- Highly effective for quantifiers with UF
  - Widely used as backend for many software verification applications
- Challenges:
  - Pattern selection, multi-patterns
  - ***Too many*** instances produced, non-termination (matching loops)
    - …solver times out
  - ***Incomplete***
    - …solver answers “unknown”



∀ Solver
Conflict-Based
E-matching
Model-Based
Enumerative
CEX-Guided
Syntax-Guided

Conflict-Based Instantiation

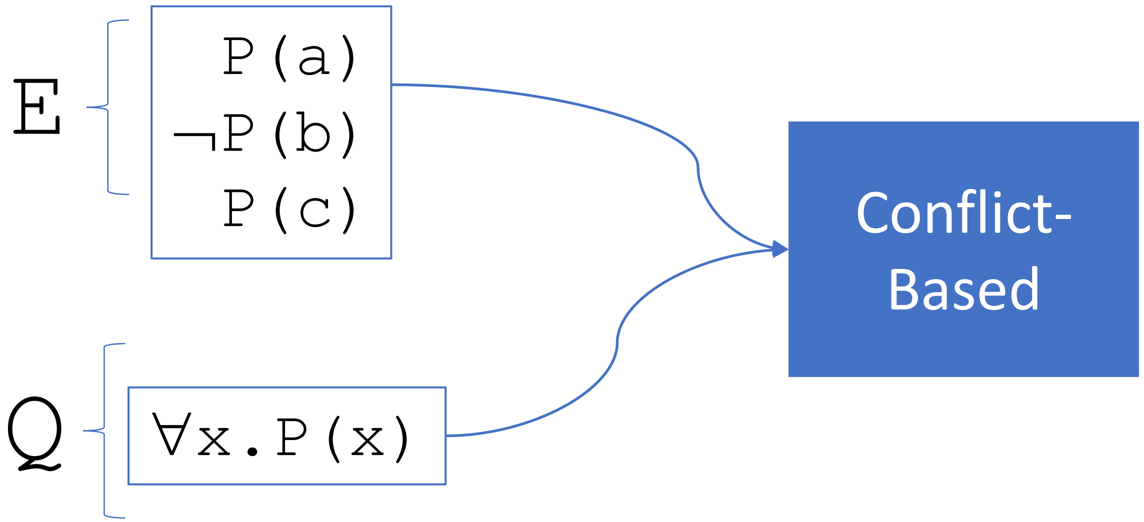
- **Idea:** Find instantiation that is in conflict with $\mathbb{E}$, if it exists

- A *conflicting instance* forces the solver to backtrack
  - Improves ability to answer "unsat"

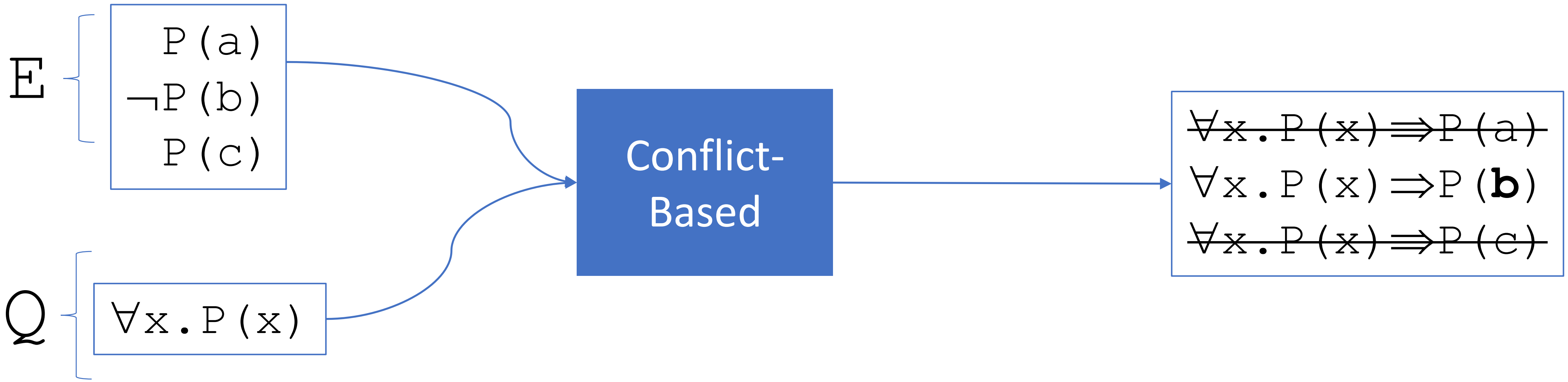
- Implemented in cvc5, veriT
  - **[Reynolds et al FMCAD 2014, Barbosa et al TACAS 2017]**

- **Key applications:** Automated Theorem Proving
  - Example: Isabelle/Sledgehammer

Conflict-Based Instantiation



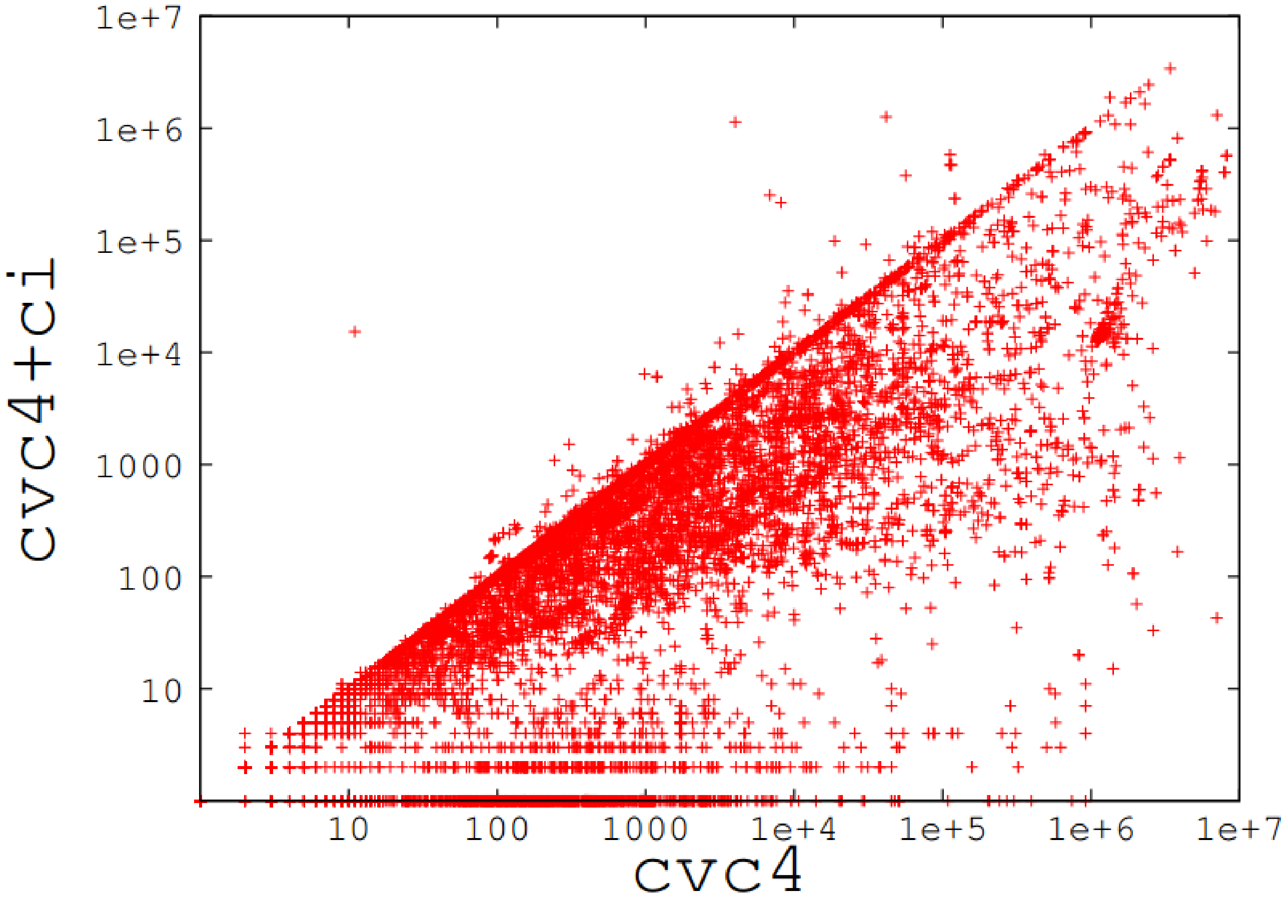
Conflict-Based Instantiation



$$\neg\, P(b),\ \underbrace{P(\mathbf{b})}_{\text{Conflicting instance}} \models \bot$$

⇒If no conflicting instance exists, resort to E-matching

Conflict-Based Instantiation: Impact



Reported number of instances.

- Using conflict-based instantiation (**`cvc4+ci`**), require an order of magnitude fewer instances for showing “UNSAT” wrt E-matching alone

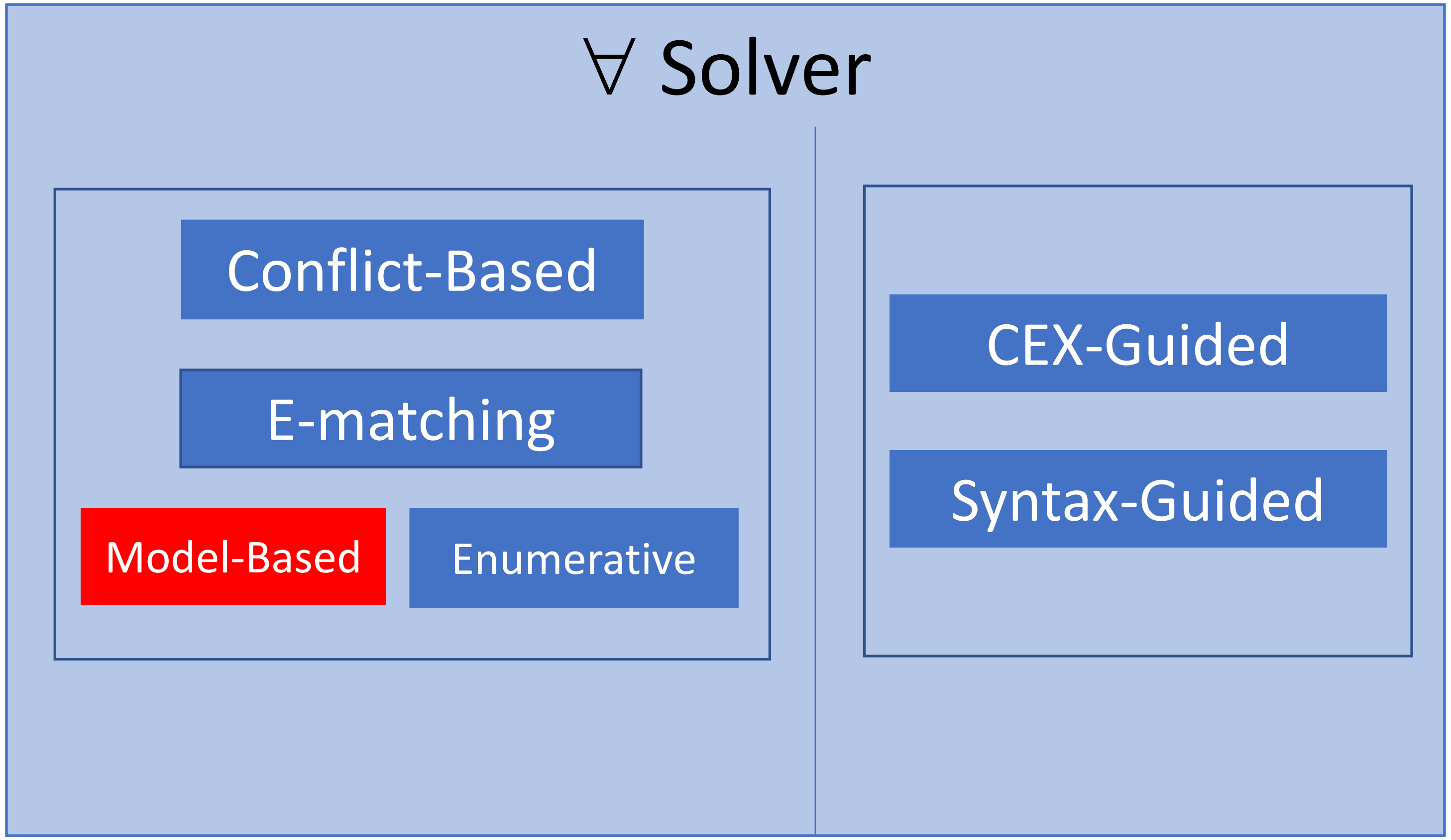
(taken from [Reynolds et al FMCAD14], evaluation On SMTLIB, TPTP, Isabelle benchmarks)

Conflict-Based Instantiation: Impact

- CVC4 with conflicting instances **`cvc4+ci`**
  - Solves the most benchmarks for TPTP and Isabelle
  - Requires almost an order of magnitude fewer instantiations

	TPTP		Isabelle		SMT-LIB	
	Solved	Inst	Solved	Inst	Solved	Inst
cvc3	5,245	627.0M	3,827	186.9M	3,407	42.3M
z3	6,269	613.5M	3,506	67.0M	**3,983**	6.4M
cvc4	6,100	879.0M	3,858	119.0M	3,680	60.7M
cvc4+ci	**6,616**	150.9M	**4,082**	28.2M	3,747	32.4M

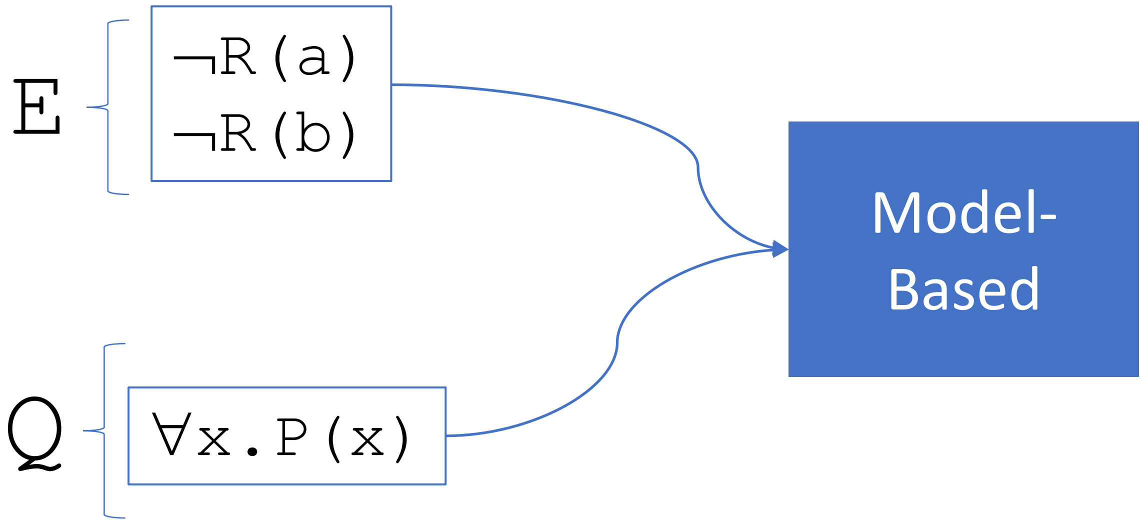
$\Rightarrow$ *A number of hard benchmarks can be solved without resorting to E-matching at all*


∀ Solver
Conflict-Based
E-matching
Model-Based
Enumerative
CEX-Guided
Syntax-Guided

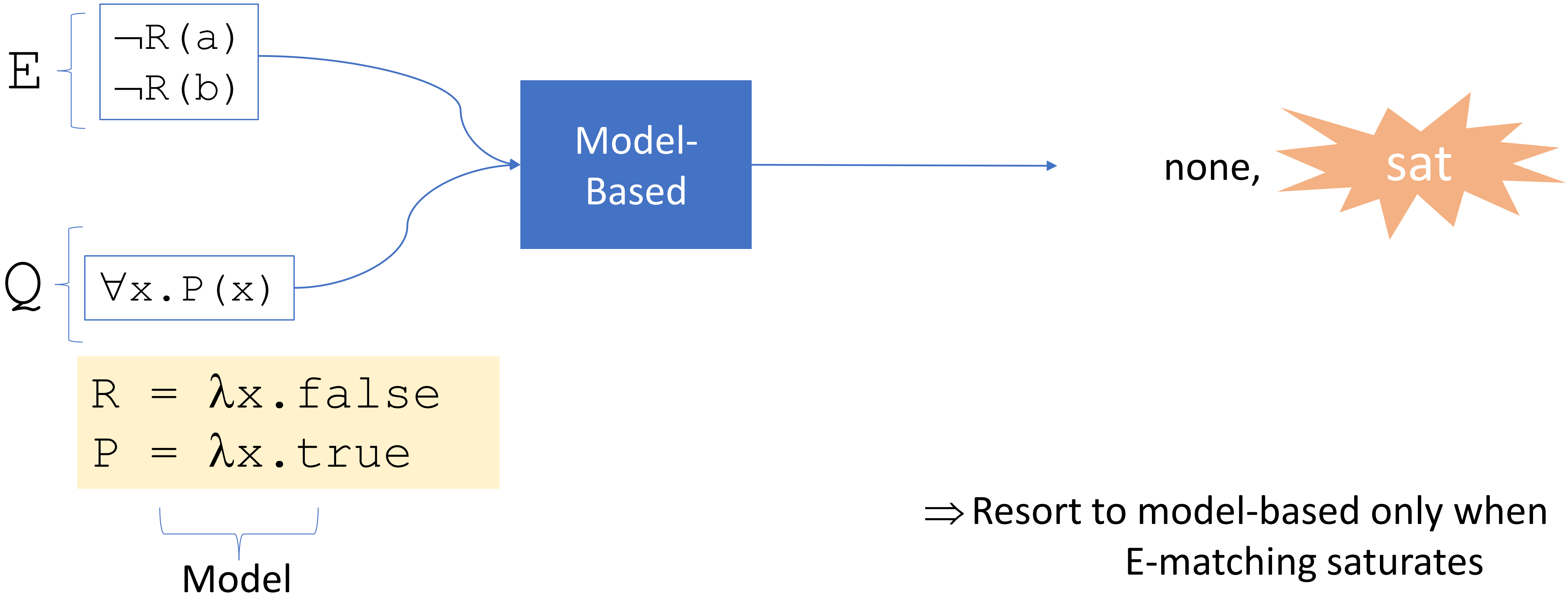
Model-Based Instantiation

- **Idea:** Instantiate quantifiers based on (complete) models for $\mathbb{E}$
- Complete for certain fragments, e.g. EPR, essentially uninterpreted
  - Can be useful for answering “sat”
- Implemented in z3, finite model finding in cvc4
  - **[Ge et al 2009, Reynolds et al 2013]**
- **Key applications:** Software Design, Planning
  - Example: Alloy Analyzer

Model-Based Instantiation

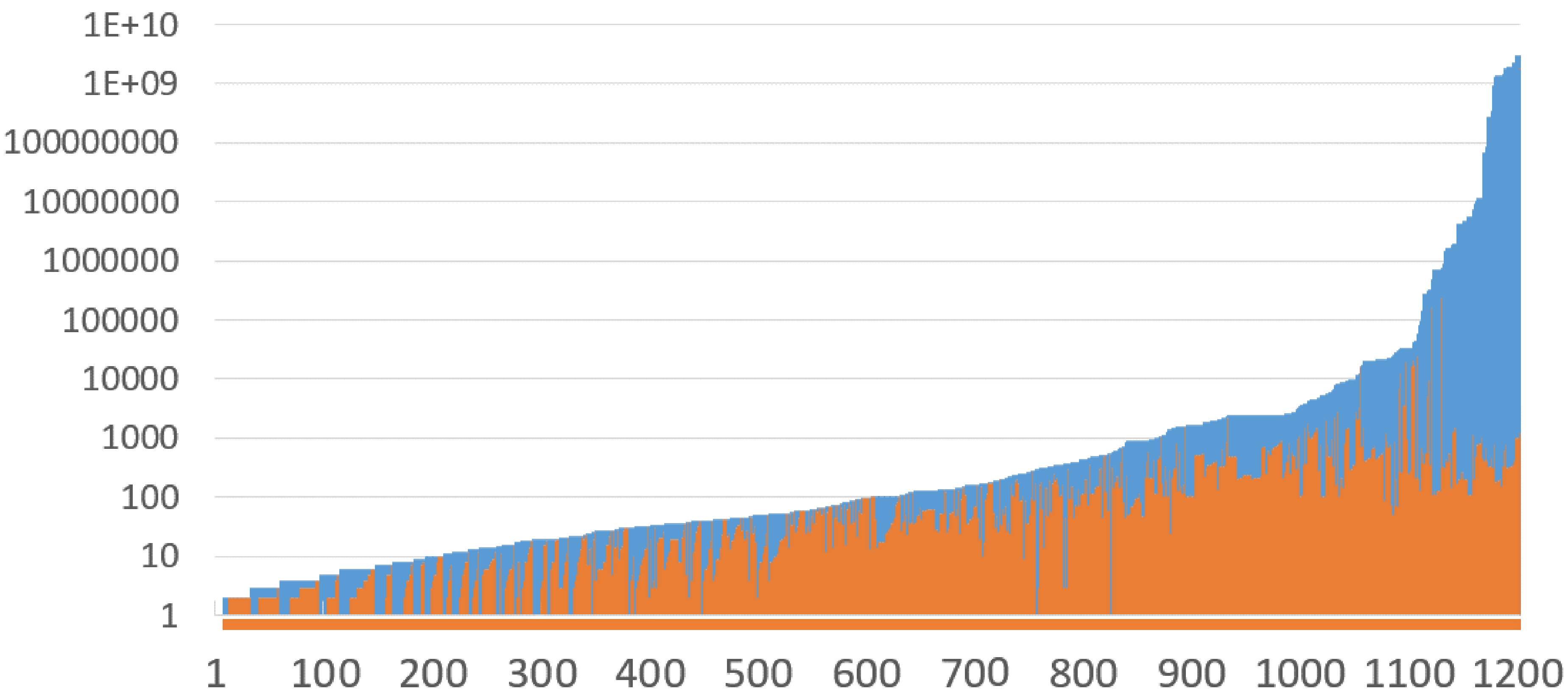


Model-Based Instantiation

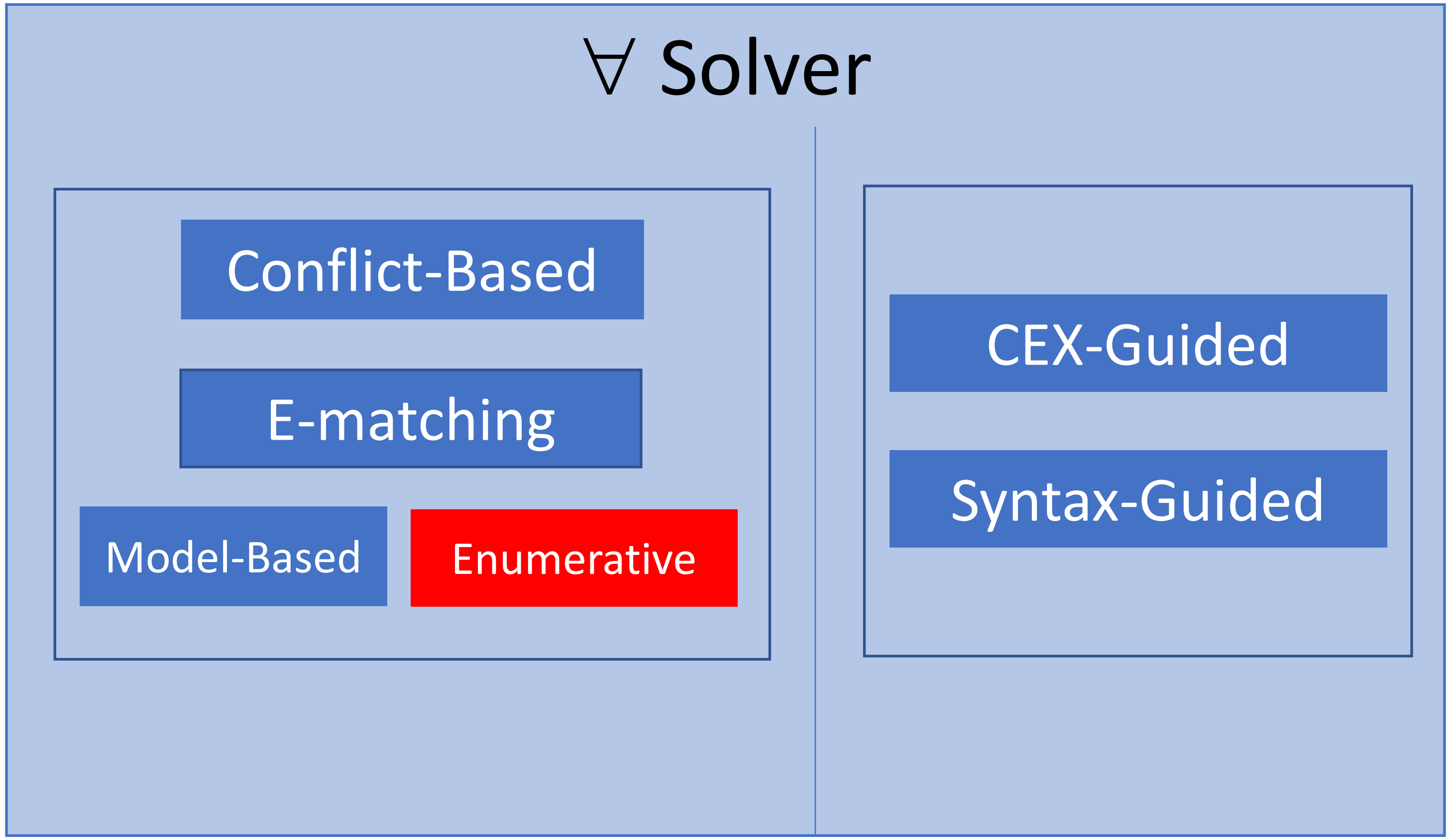


$\Rightarrow$ Resort to model-based only when E-matching saturates

Model-based Instantiation: Impact



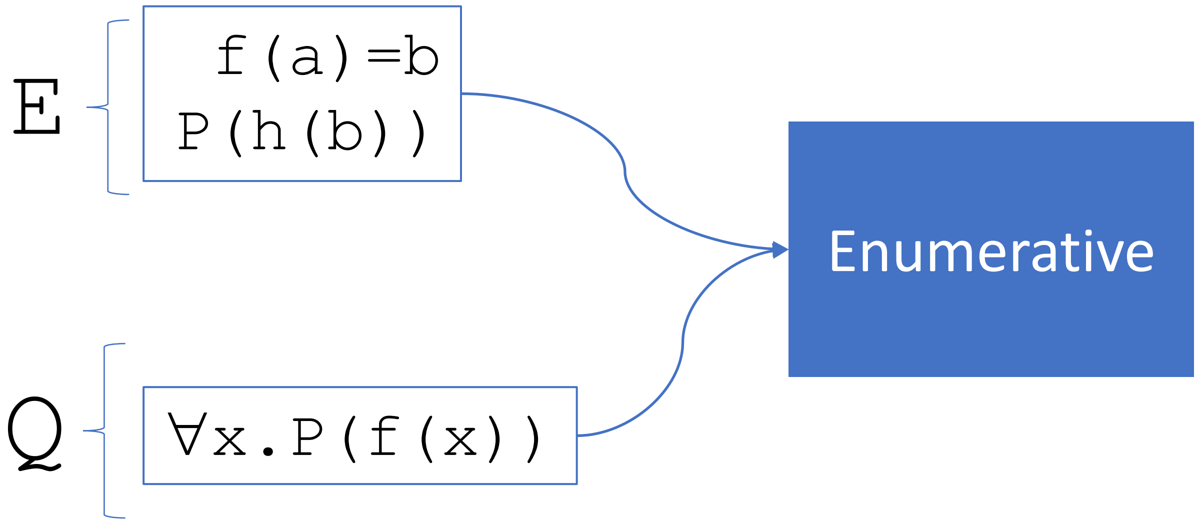
- CVC4 Finite Model Finding + Model-Based instantiation **[Reynolds et al CADE13]**
  - Approach can scale to domains of >2 billion, only adds fraction of possible instances


∀ Solver
Conflict-Based
E-matching
Model-Based
Enumerative
CEX-Guided
Syntax-Guided

Enumerative Instantiation

- **Idea:** Instantiate based on (fair) enumeration of terms from $\mathbb{E}$
- Effective alternative to model-based, better performance for “unsat”
- Complete for limited fragments
- Implemented in cvc5, veriT
  - **[Reynolds et al TACAS 2018, Janota et al 2021]**
- Key applications: Automated theorem proving
  - Example: Isabelle/Sledgehammer, TPTP

Enumerative Instantiation



Enumerative Instantiation

$\mathbb{E}$: $f(a)=b$, $P(h(b))$

$\mathbb{Q}$: $\forall x. P(f(x))$

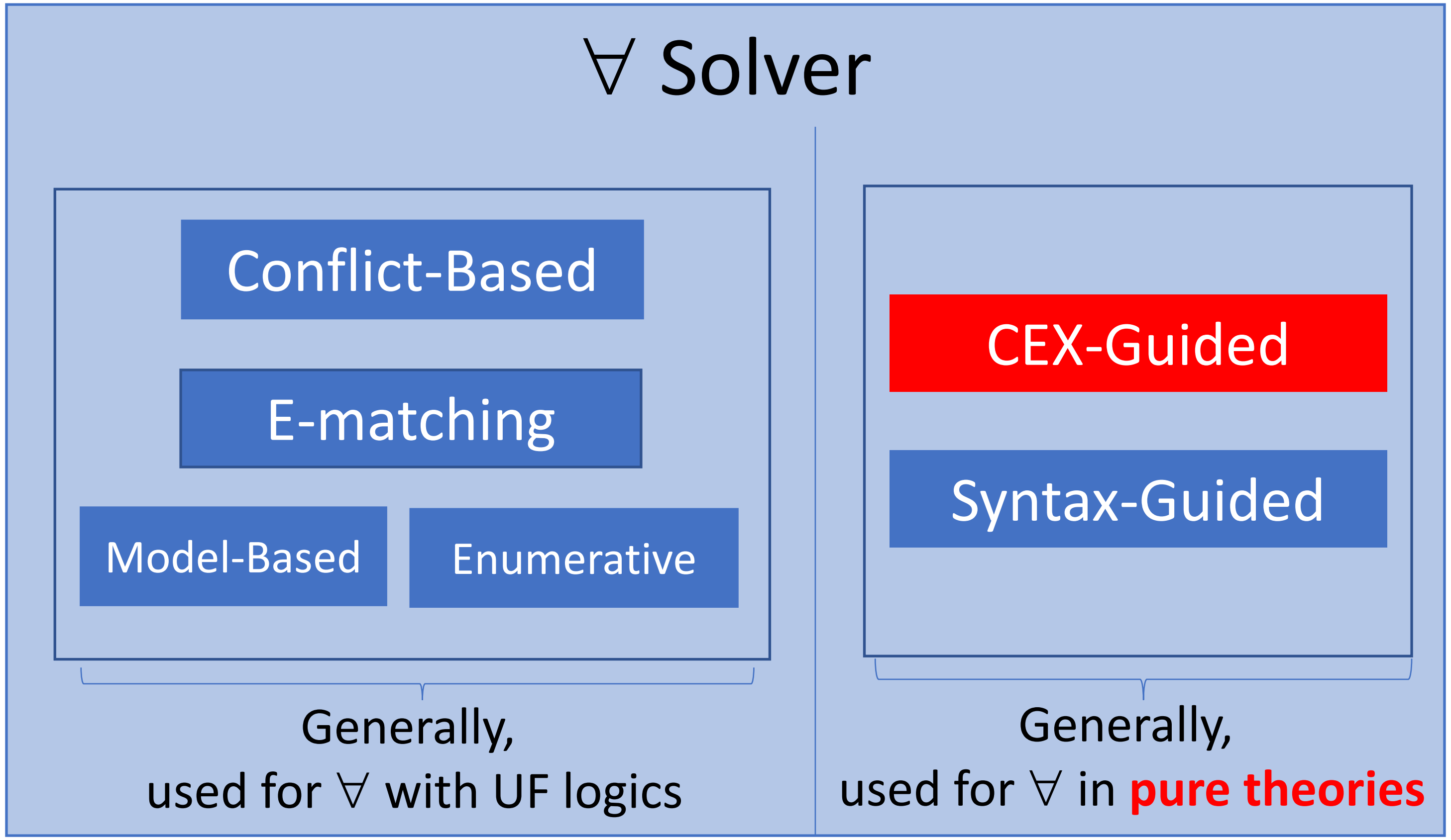
Enumerative

$\forall x. P(f(x)) \Rightarrow P(f(a))$
$\forall x. P(f(x)) \Rightarrow P(f(b))$
$\forall x. P(f(x)) \Rightarrow P(f(f(a)))$
$\forall x. P(f(x)) \Rightarrow P(f(h(b)))$
…

$a < b < f(a) < h(b) < \ldots$

Ordering over terms from $\mathbb{E}$

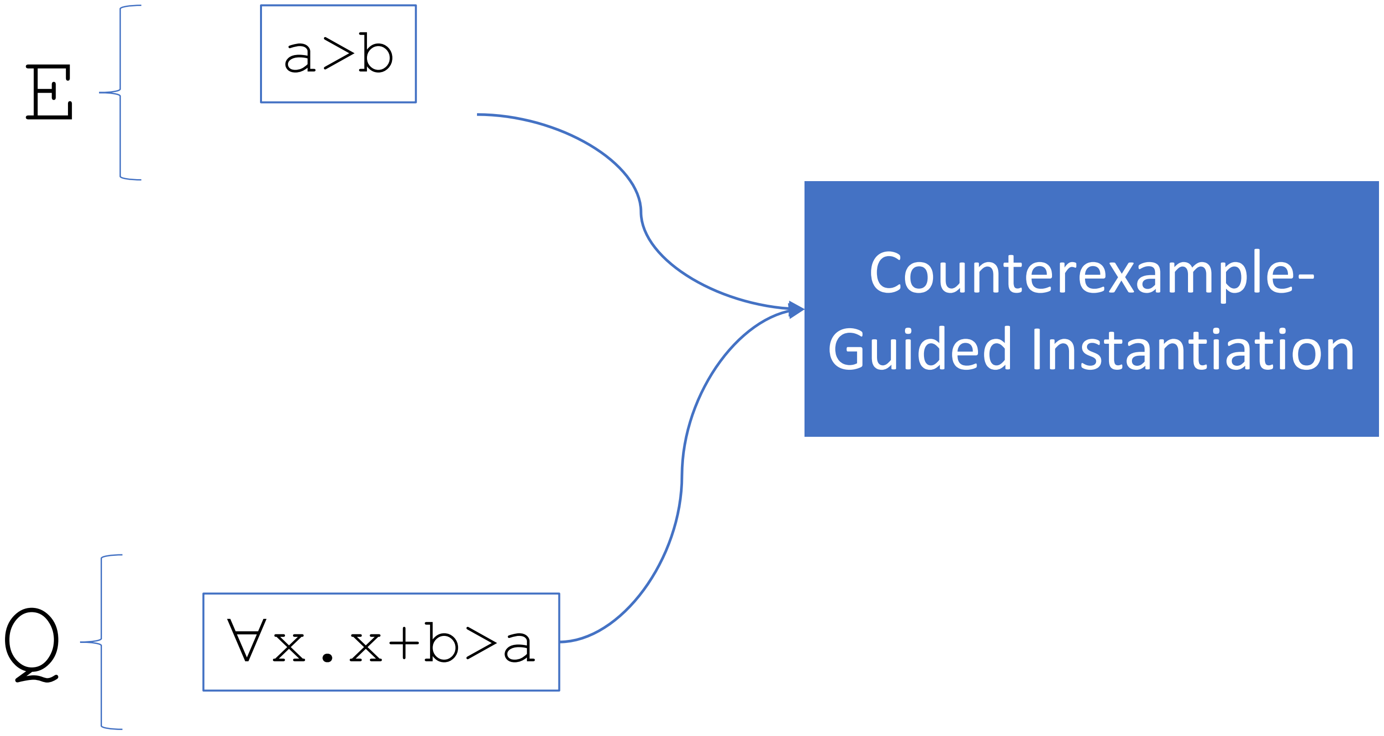
$\Rightarrow$ Finds instances that E-matching may miss, more lightweight than MBQI


∀ Solver
Conflict-Based
E-matching
Model-Based
Enumerative
Generally,
used for ∀ with UF logics
CEX-Guided
Syntax-Guided
Generally,
used for ∀ in pure theories

Counterexample-Guided Instantiation

- **Idea:** Instantiate based on T-solving for counterexamples $\neg Q \wedge E$
- Can be seen as a lazy quantifier elimination algorithm in SMT loop
- Complete for quantified linear integer/real arithmetic, finite domains
- Variants of idea implemented in cvc5, (extensions of) z3, yices
  - **[Bjorner 2012, Komuravelli et al 2014, Dutertre 2015, Reynolds 2015, Bjorner/Janota 2016, Fedyukovich et al 2016]**
- Key applications: Synthesis, Hardware Verification, Compiler Optimization

Counterexample-guided Instantiation



Counterexample-guided Instantiation

E: $a>b$, $\mathbf{x+b \leq a}$

$\underbrace{x=a-b}_{\text{Solve for } x}$

Q: $\forall x. x+b>a$

Counterexample-Guided Instantiation

$\forall x. x+b>a \Rightarrow (a-b)+b>a$

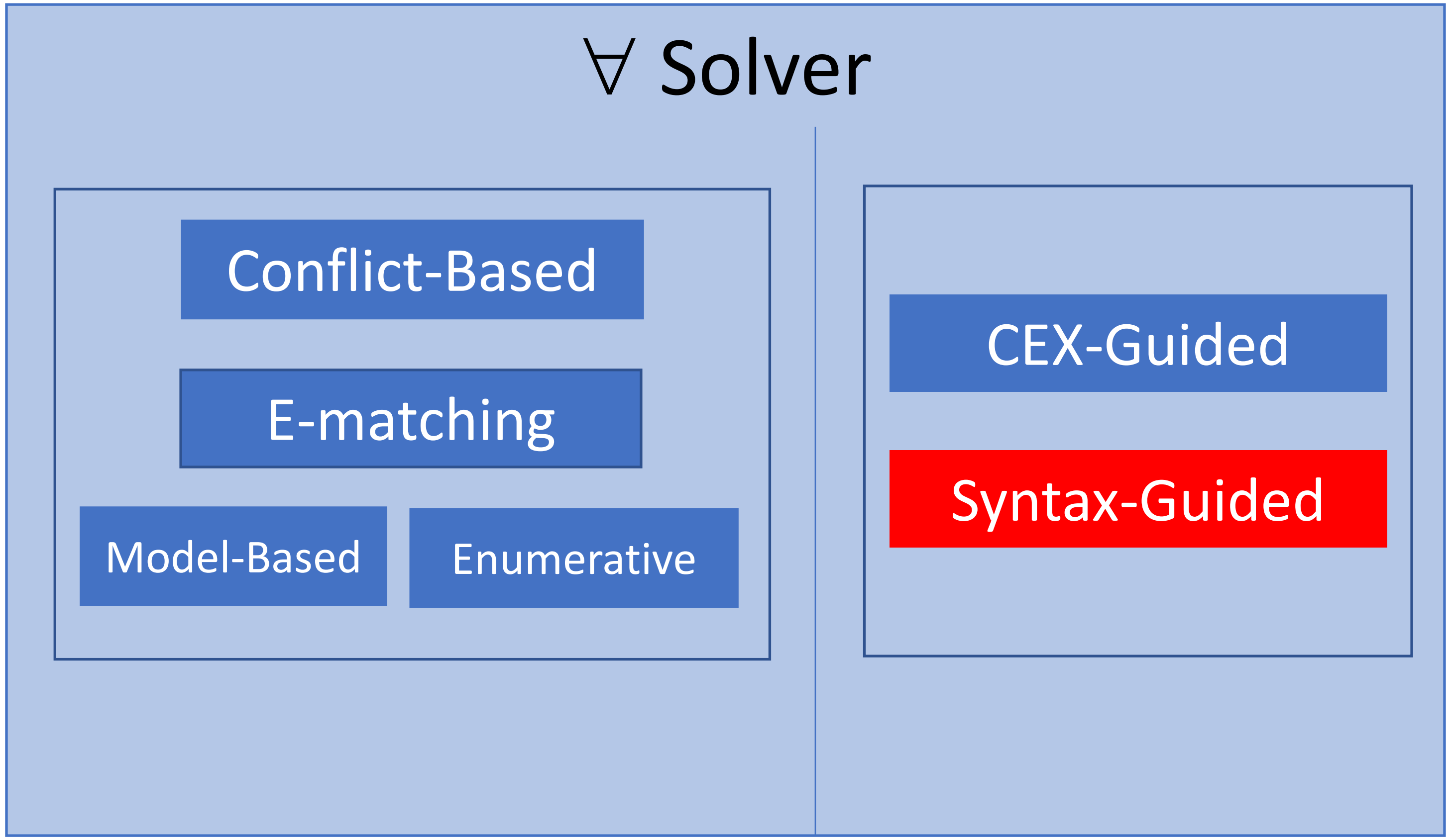
where:

$(a-b)+b>a \Leftrightarrow$

$a>a \Leftrightarrow$

$\perp$

$\Rightarrow$ Can simulate e.g. Cooper, Loos-Weispfenning, Ferrante-Rackoff algorithms for QE


∀ Solver
Conflict-Based
E-matching
Model-Based
Enumerative
CEX-Guided
Syntax-Guided

Syntax-Guided Instantiation

- **Idea:** Instantiate based on enumerating terms from T-specific grammar
- Leverages advances in syntax-guided synthesis (SyGuS) **[Alur et al 2013]**
- Implemented:
  - For bitvector theory in Boolector **[Preiner et al TACAS 2017]**
  - For all supported theories in cvc5 **[Niemetz et al TACAS 2021]**
- Key applications: Synthesis and Verification for emerging theories
  - E.g. quantifiers over floating points

Syntax-Guided Instantiation

E none

Q $\boxed{\forall x.\, x^2 \neq a^2 + b^2 + 2*a*b}$

Syntax-Guided Instantiation

Syntax-Guided Instantiation

E: none

Q: $\forall x. x^2 \neq a^2+b^2+2*a*b$

Syntax-Guided Instantiation

$Q \Rightarrow 0^2=a^2+b^2+2*a*b$
$Q \Rightarrow 1^2=a^2+b^2+2*a*b$
$Q \Rightarrow a^2=a^2+b^2+2*a*b$
…
$Q \Rightarrow (\mathbf{a+b})^2=a^2+b^2+2*a*b$

$G \rightarrow a \mid b \mid 0 \mid 1 \mid G+G \mid G*G \mid -G$

Construct grammar generating terms of integer type

$\Rightarrow$ Best known approach for theories where QE is unknown

Summary

- SMT solvers handle diverse set of inputs (with quantifiers)

- Best instantiation technique depends on the logic
  - When UF is present:

    $\Rightarrow$ *E-matching, conflict-based, model-based, enumerative*
  - For traditional theories (e.g. LIA, BV) which emit quantifier elimination:

    $\Rightarrow$ *Counterexample-guided*
  - For other theories (e.g. floating points, strings, non-linear arithmetic):

    $\Rightarrow$ *Syntax-guided*

- Techniques in this talk implemented in SMT solver cvc5
  - Open source
  - Available at : **https://github.com/cvc5**
- …Thanks for listening!