Solvers
for
Software Reliability and Security

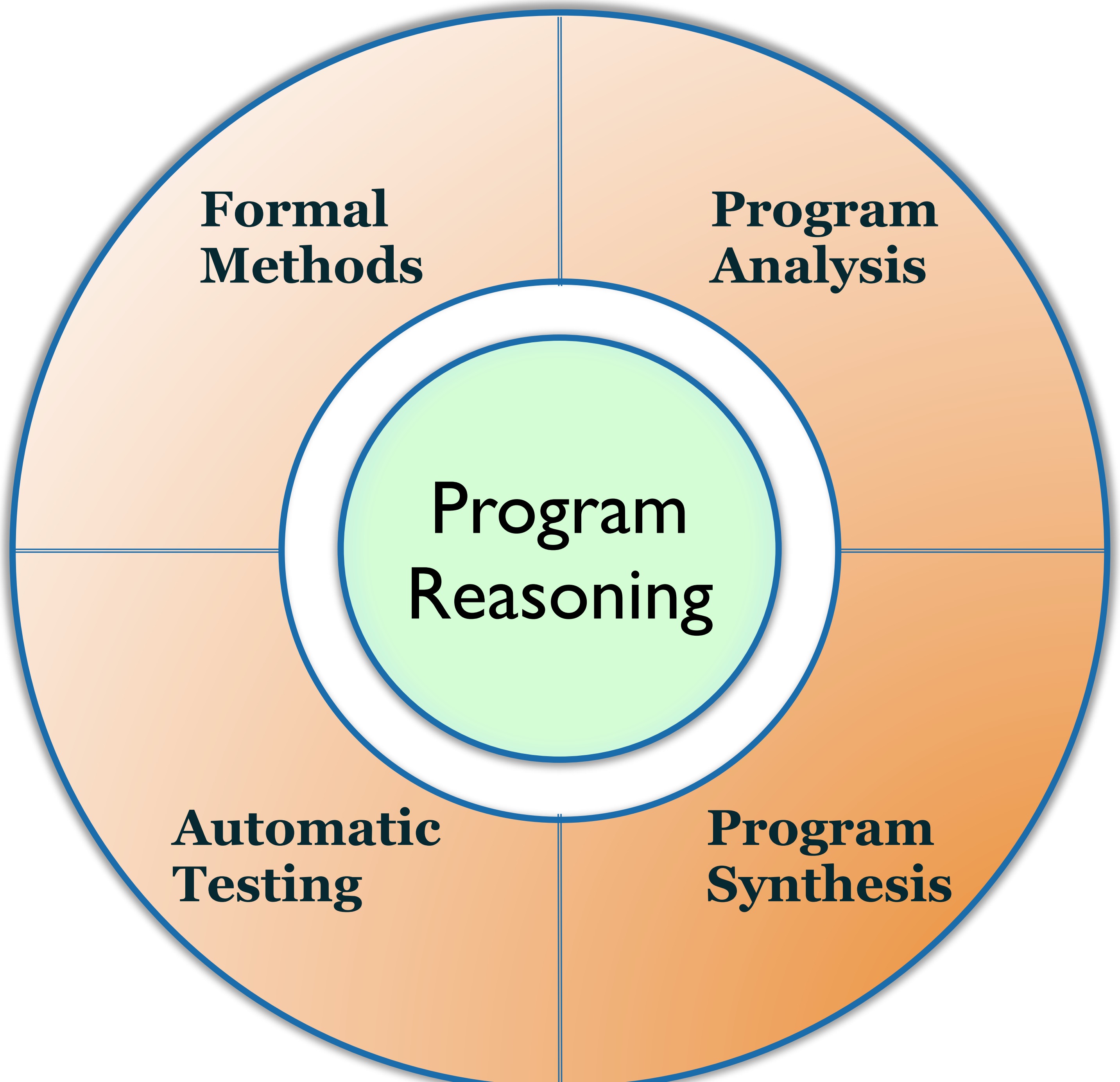
VIJAY GANESH
MIT
2011

The Software Reliability Problem

- Software is error-prone
- Significant and increasing costs
- Foundational research problem and opportunity

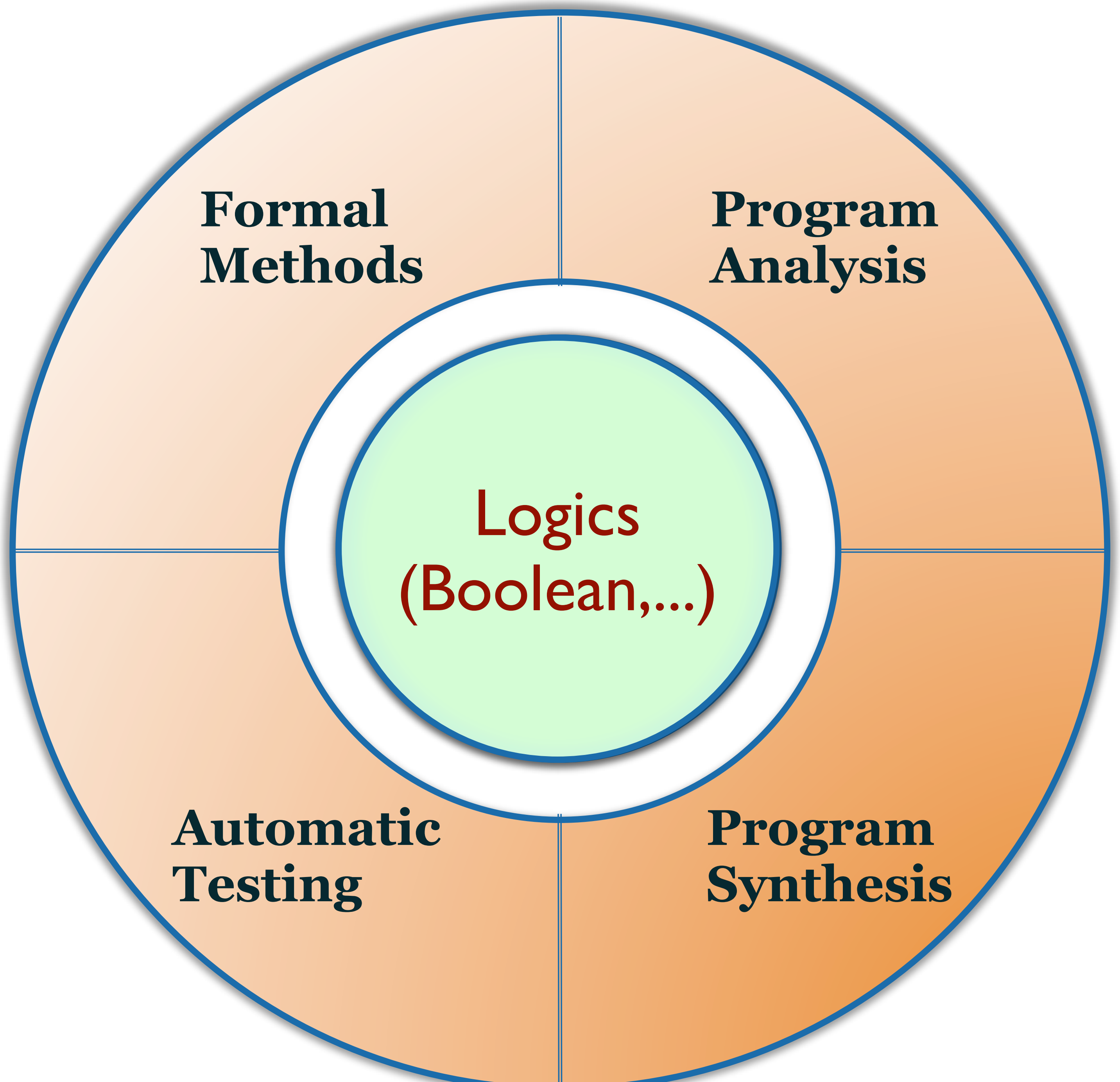
What is at the Core?

Logic Abstractions of Computation



What is at the Core?

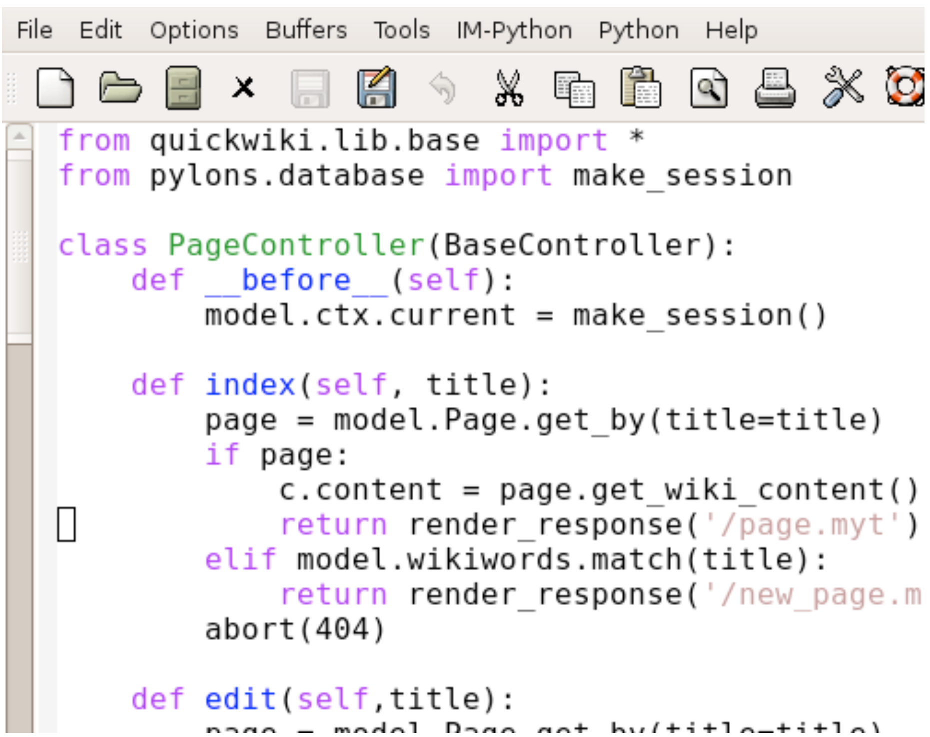
Logic Abstractions of Computation



Why Logic for Program Reasoning

Logic Abstractions of Computation

Imperative Code: Operational view



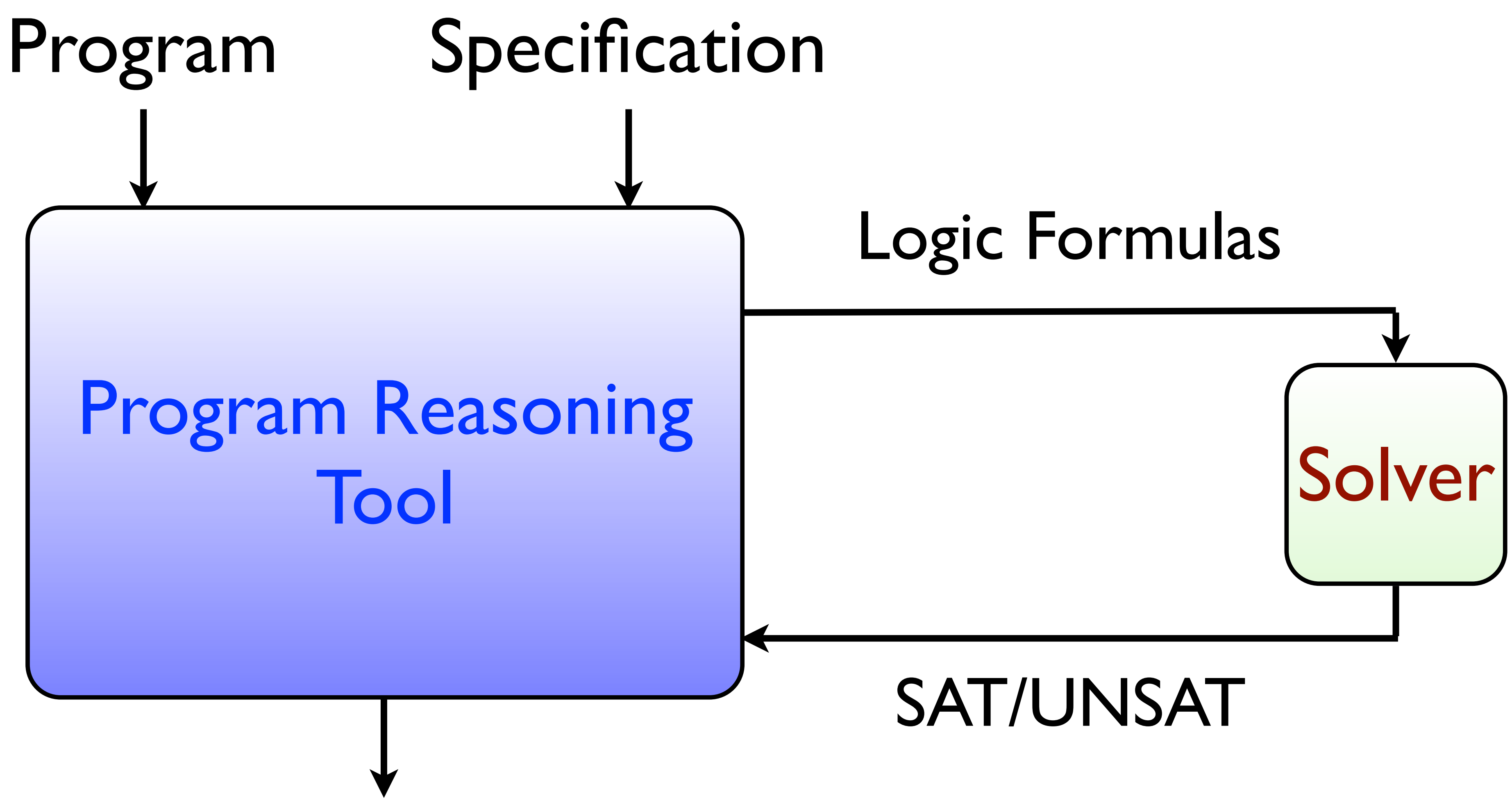
Logic Formula: Declarative View

$(\forall x.(P(x) \wedge Q(x)) \leftrightarrow ((\forall x.P(x)) \wedge (\forall x.Q(x)))$
$(\exists x.(P(x) \wedge Q(x)) \rightarrow ((\exists x.P(x)) \wedge (\exists x.Q(x)))$
$(\exists x.(P(x) \vee Q(x)) \leftrightarrow ((\exists x.P(x)) \vee (\exists x.Q(x)))$
$((\forall x.P(x)) \vee (\forall x.Q(x))) \rightarrow (\forall x.(P(x) \vee Q(x)))$
$(\exists x.\forall y.R(x,y)) \rightarrow (\forall y.\exists x.R(x,y))$
$(\neg(\exists x.P(x))) \leftrightarrow (\forall x.(\neg P(x))$
$(\neg(\forall x.P(x))) \leftrightarrow (\exists x.(\neg P(x))$
$(\neg(\exists x pt.P(x))) \leftrightarrow (\forall x pt.(\neg P(x))$
$(\neg(\forall x pt.P(x))) \leftrightarrow (\exists x pt.(\neg P(x))$
$(\forall x.(x = t \rightarrow F(x))) \leftrightarrow F(t)$
$(\exists x.(x = t \wedge F(x))) \leftrightarrow F(t)$

- Logic provides abstractions of computation
- Easy to work with abstractions
- Compact representation of desired properties

Reliability through Logical Reasoning

Engineering, Usability, Novelty



Program is Correct?
or Generate Counterexamples (Test cases)

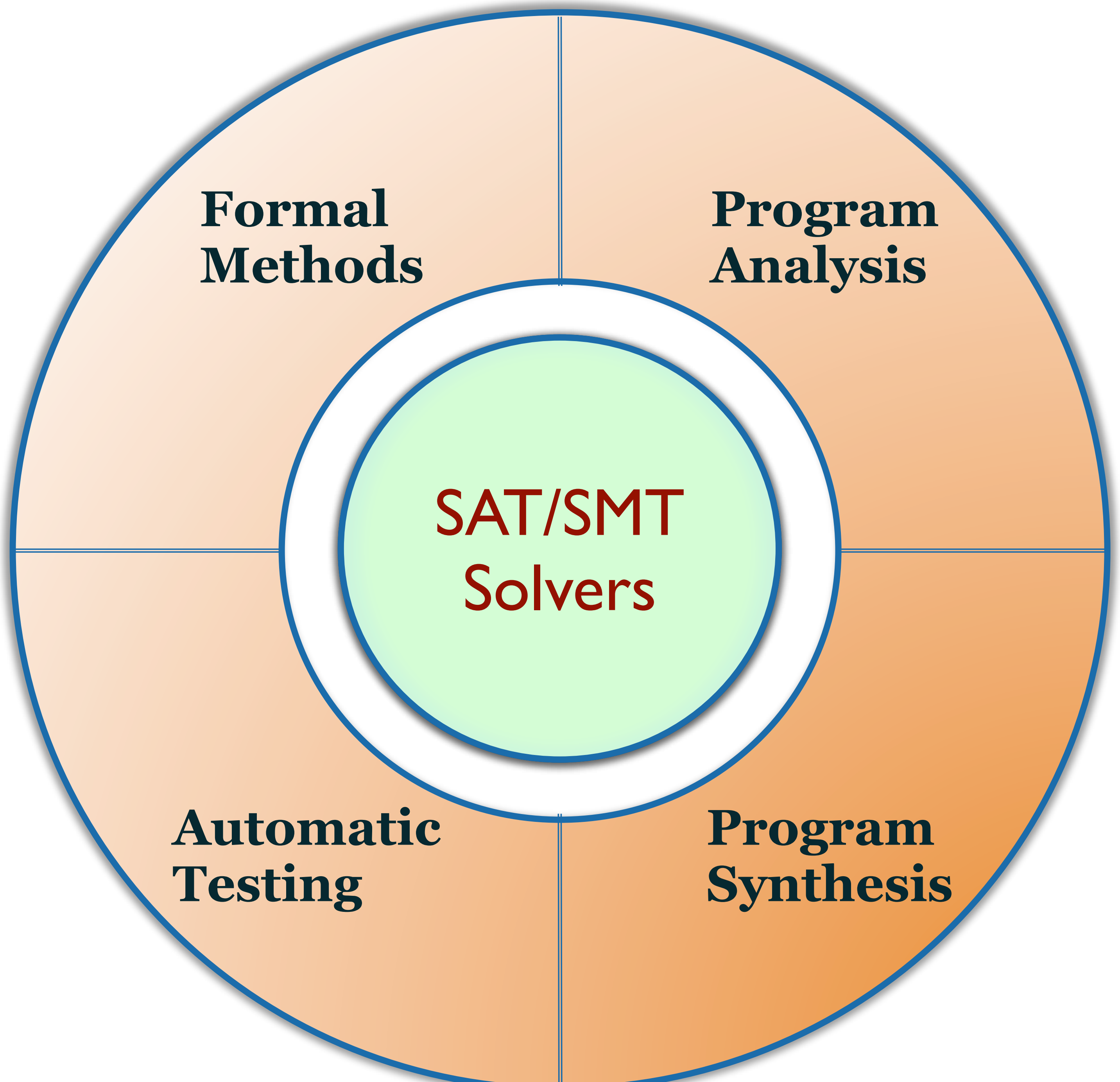
What is at the Core?

The SAT/SMT Problem

- Rich logics (Modular arithmetic, Arrays, Strings,...)
- NP-complete, PSPACE-complete,...
- Practical, scalable, usable, automatic
- Enable novel software reliability approaches

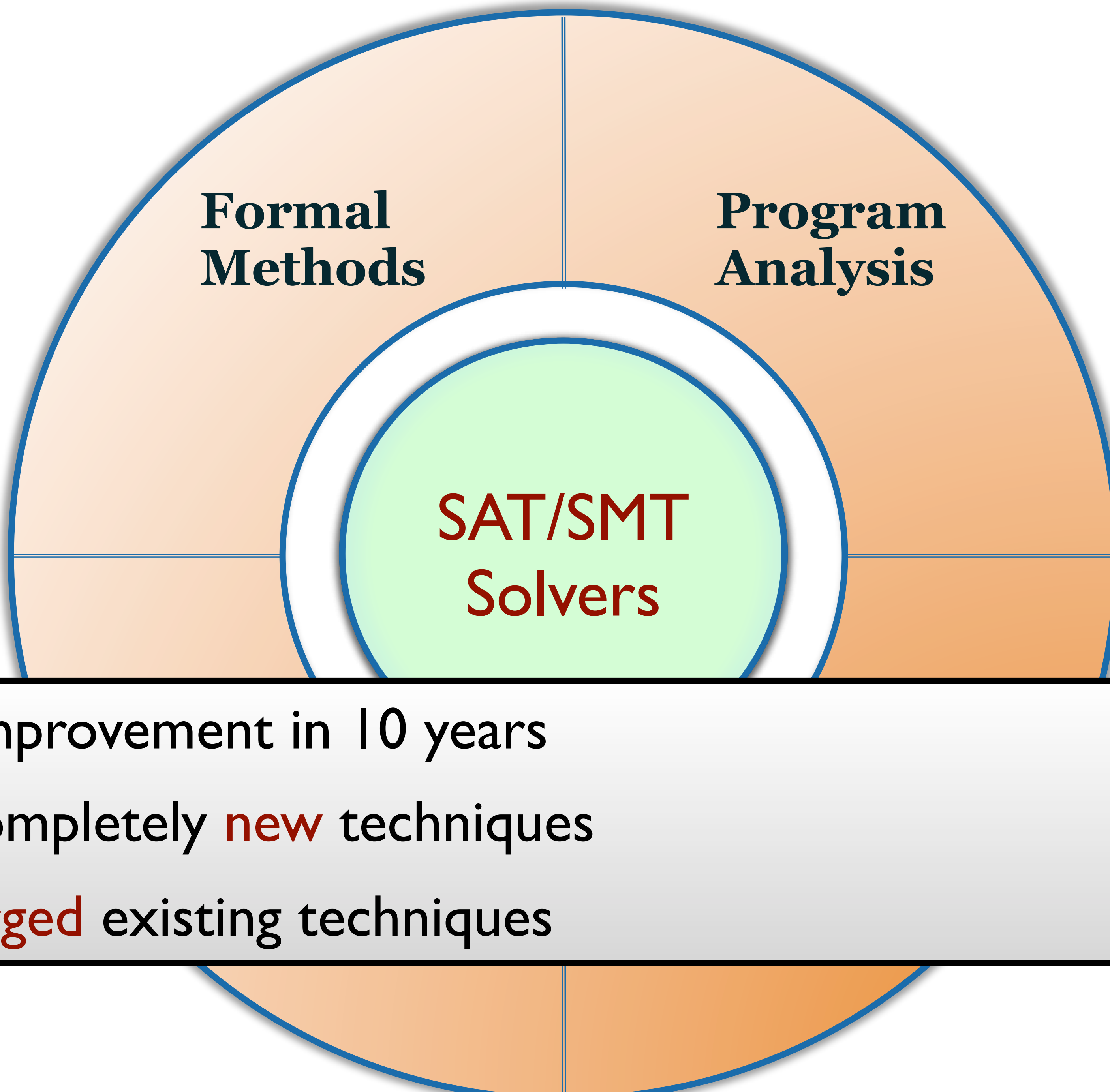
So, What's New?

From Reliability Problem to Solvers



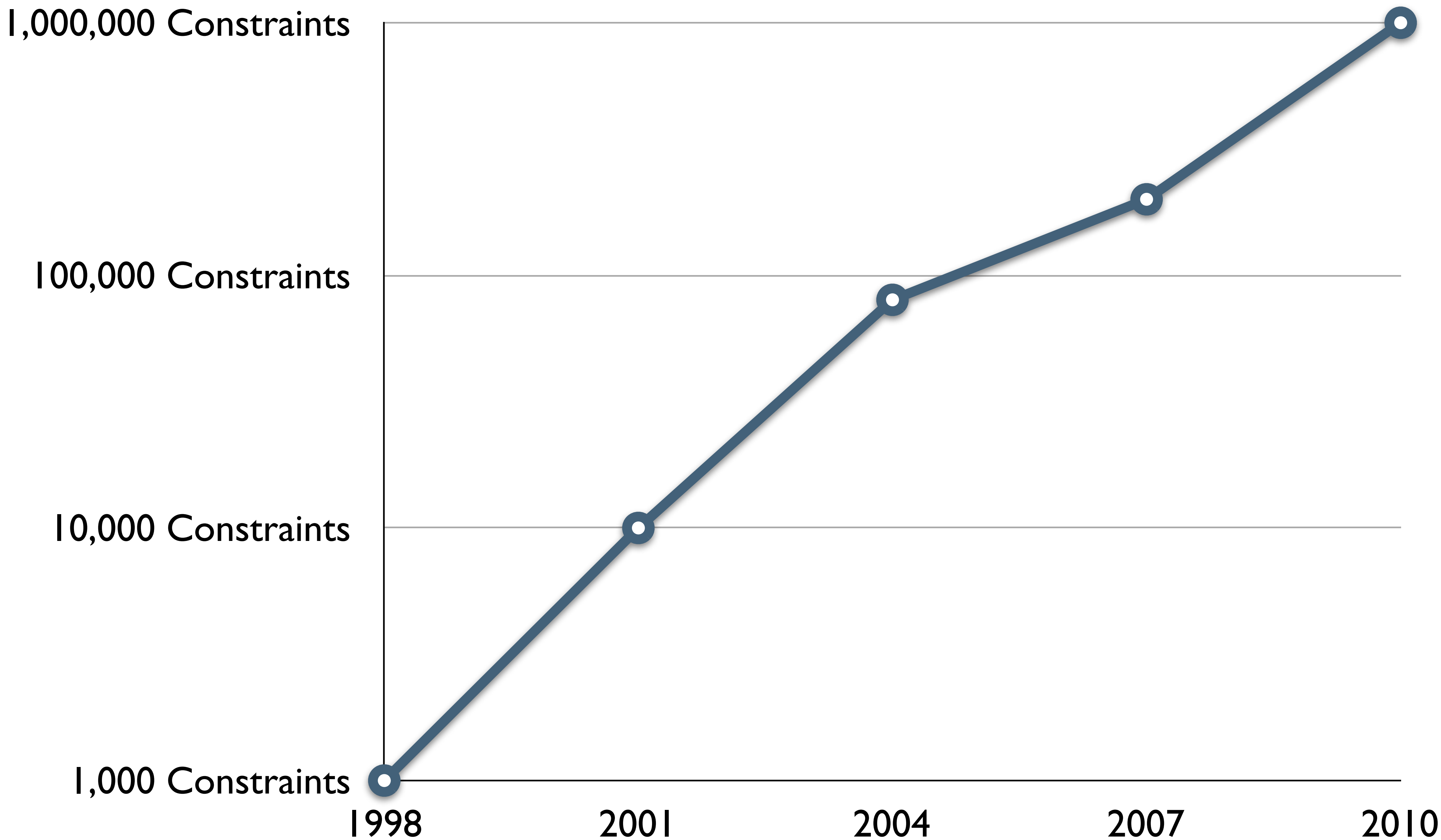
So, What's New?

From Reliability Problem to Solvers

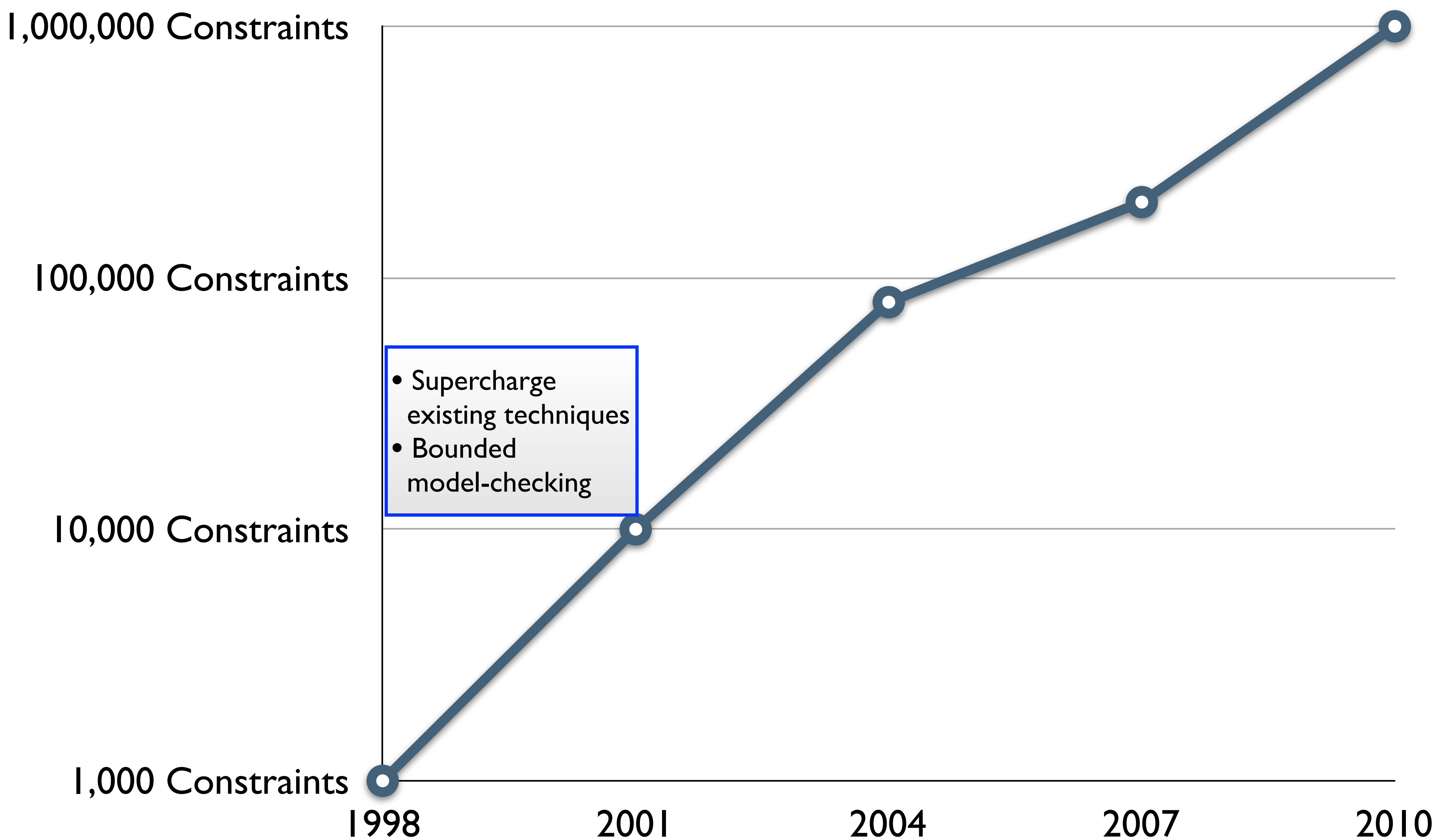


- 1000+X improvement in 10 years
- Enabled completely new techniques
- Super-charged existing techniques

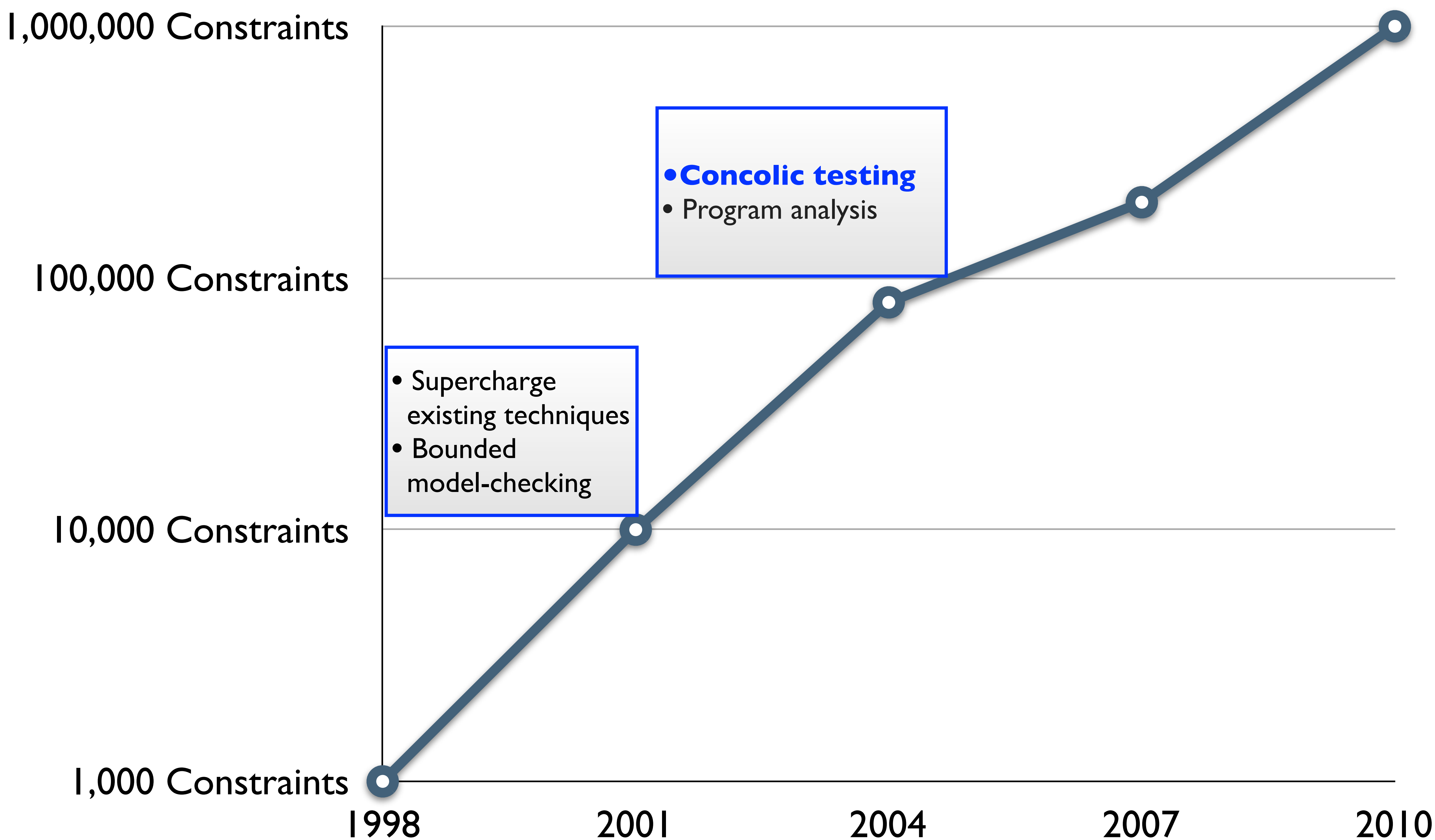
And, The Research Story is....



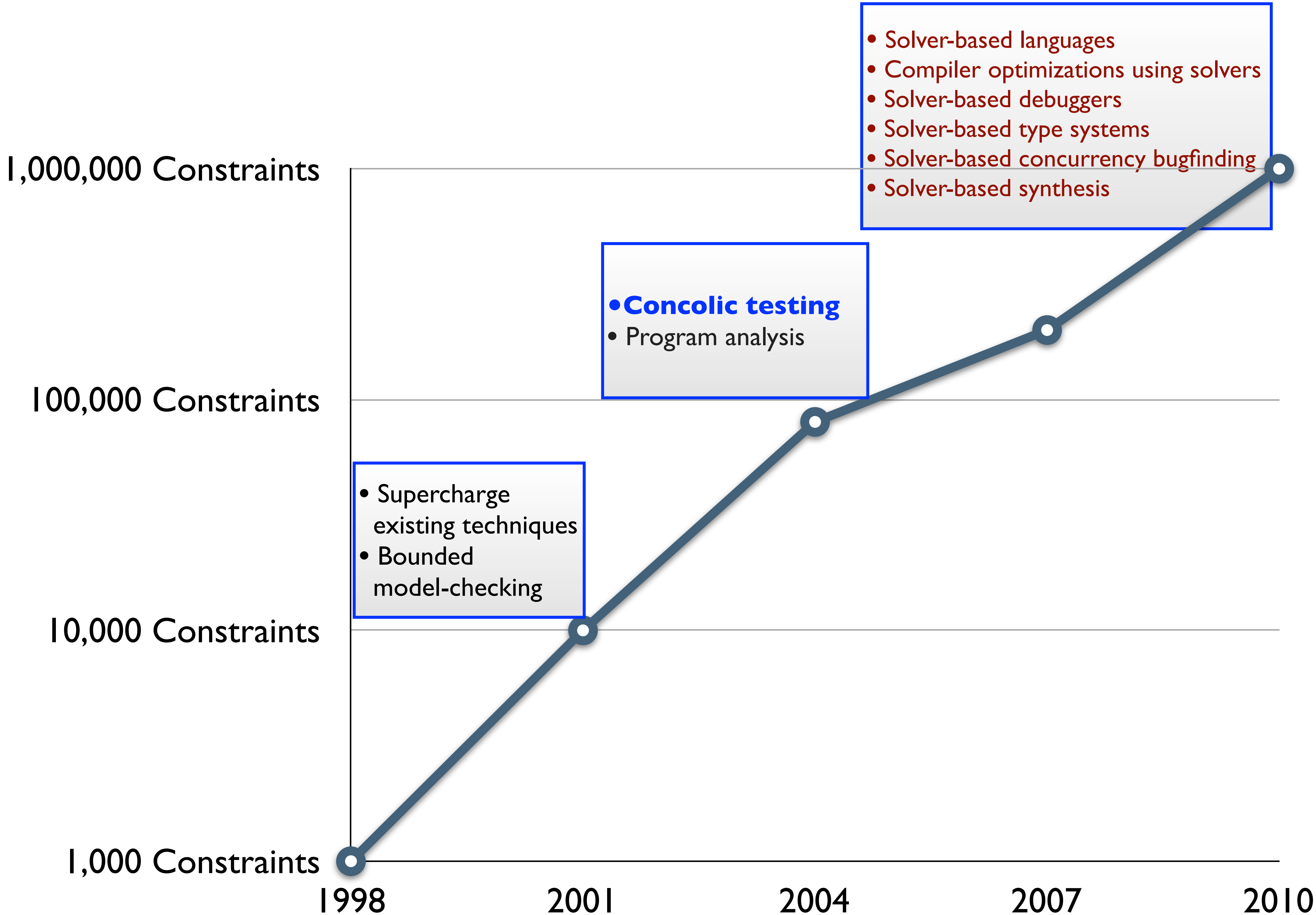
And, The Research Story is....



And, The Research Story is....

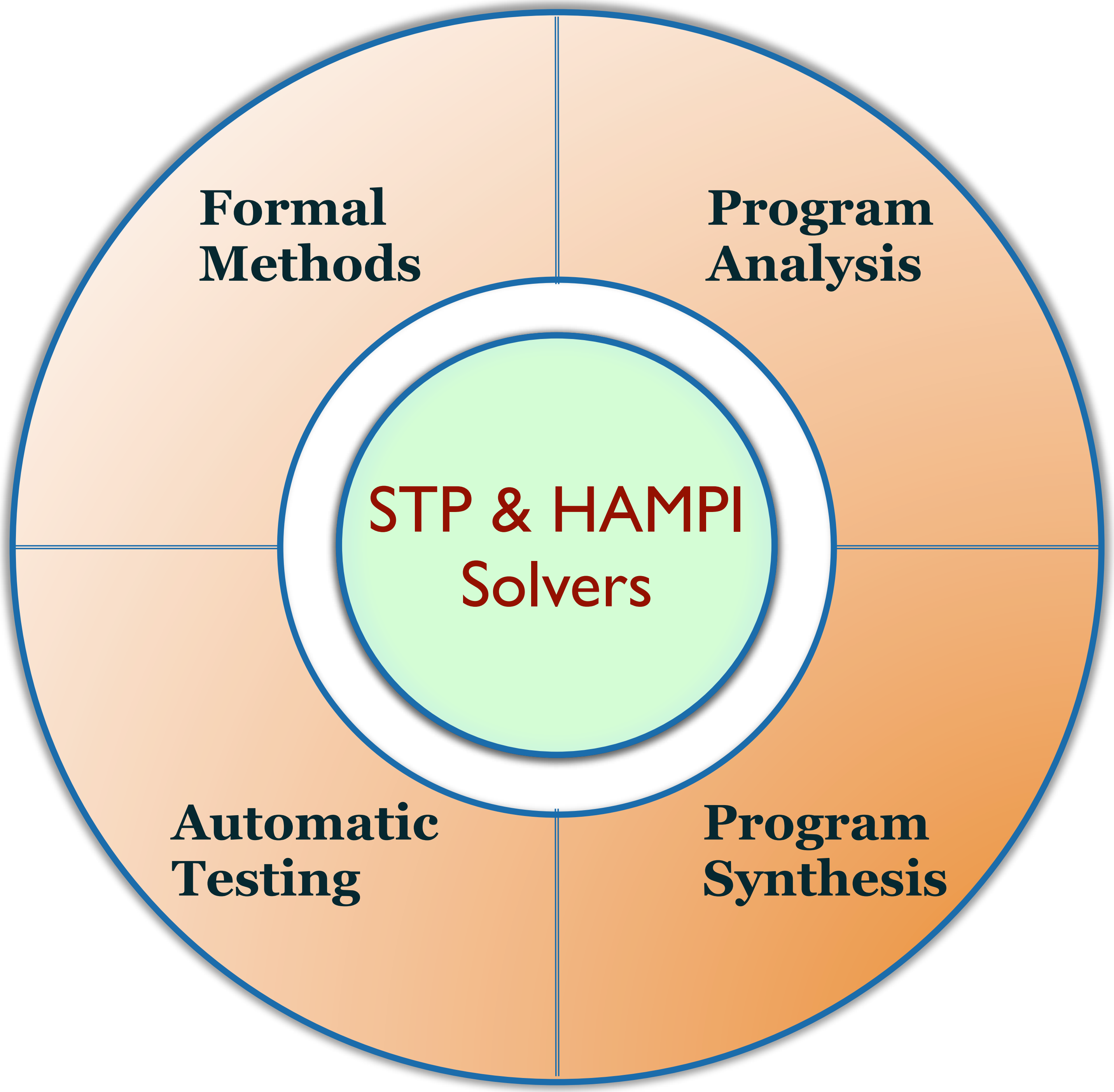


And, The Research Story is....



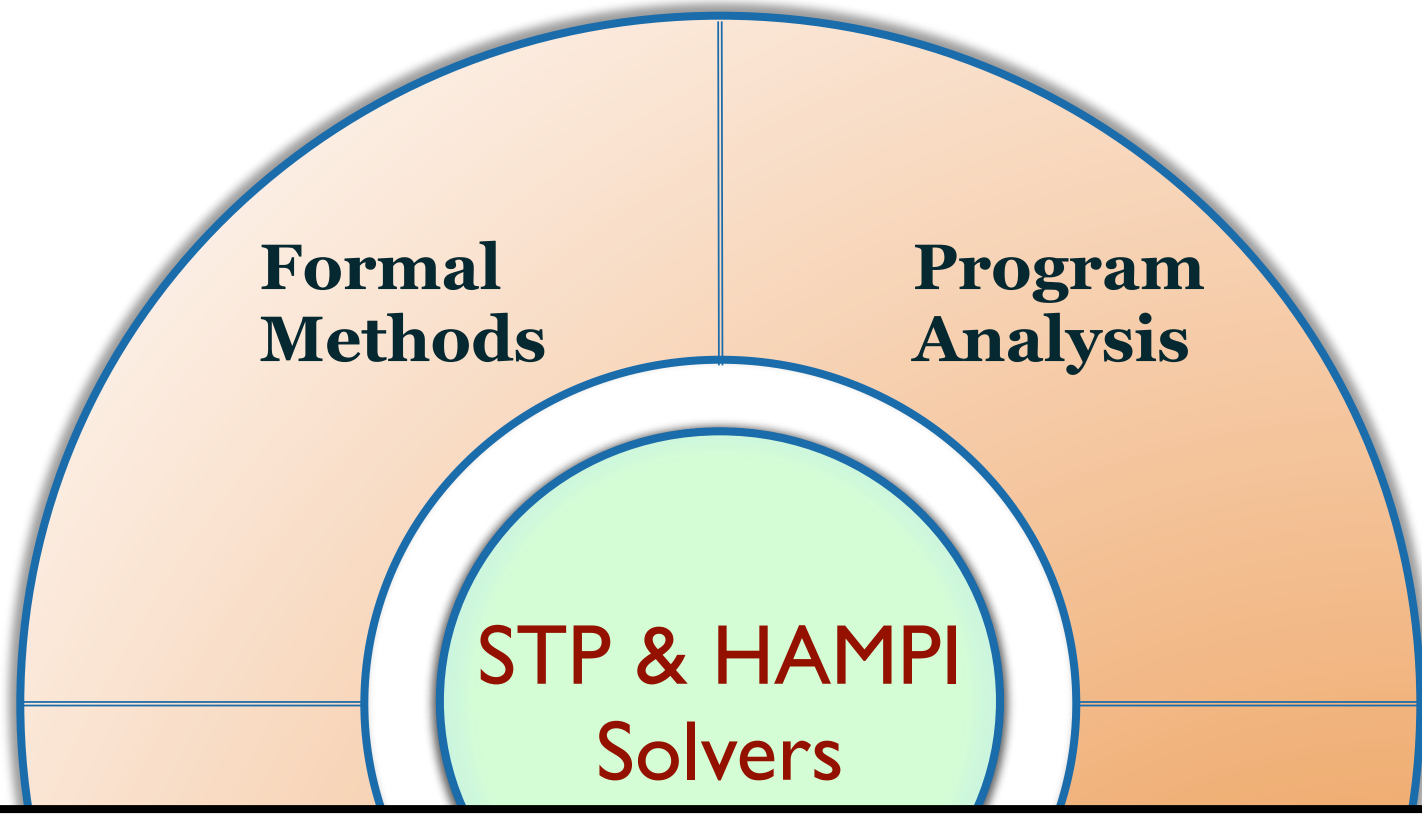
My Contributions

STP & HAMPI Solvers



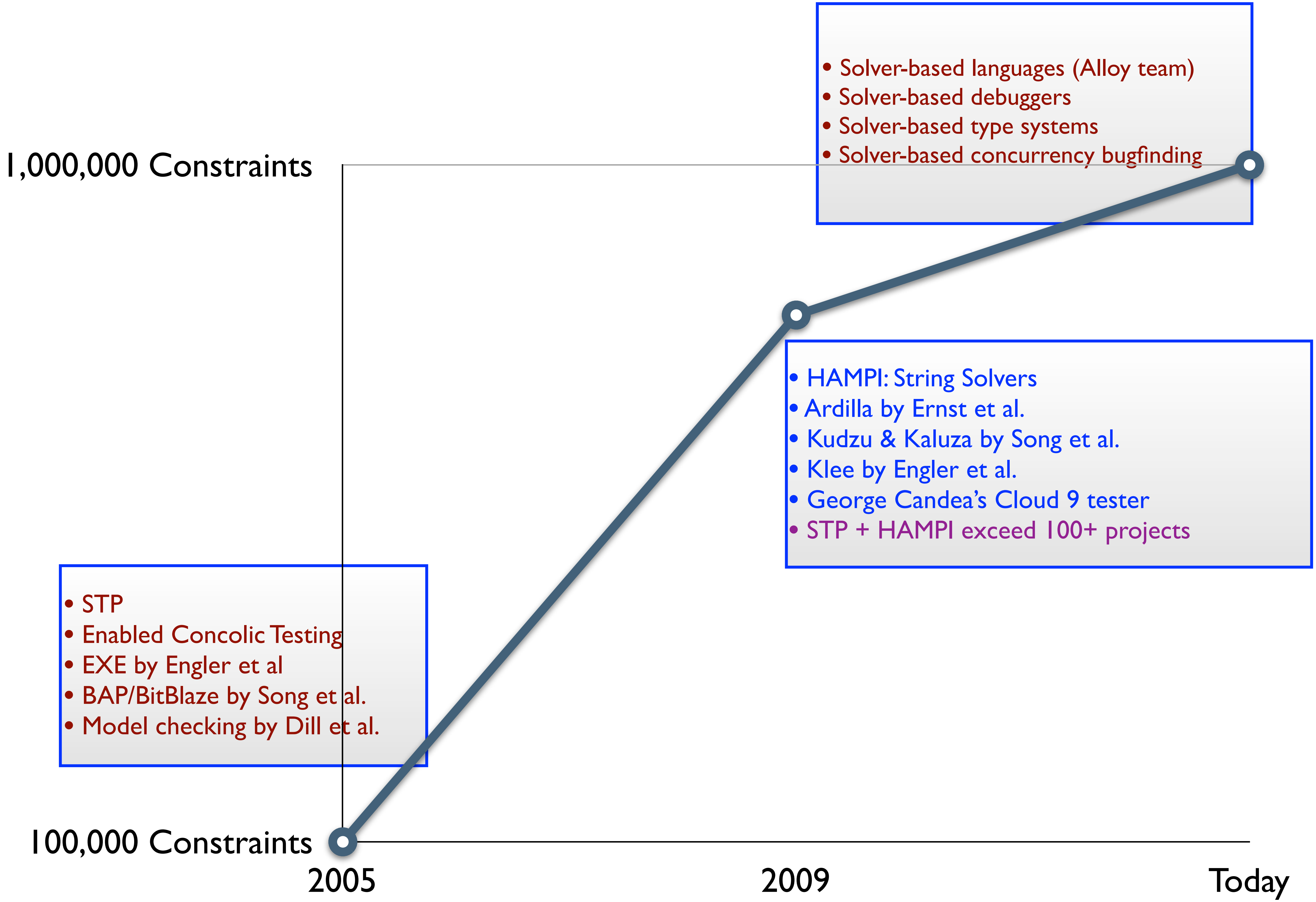
My Contributions

STP & HAMPI Solvers



- Can handle real-world formulas with millions of constraints
- Enabled completely new techniques (e.g., Concolic testing)
- Enable test million-line codes
- Super-charged existing techniques (e.g., Hardware bounded MC)
- Future is bright: Multicore, programming language, runtime systems

And, The Research Story is ...



Key Contributions

Name	**Key Concept**	**Impact**	**Pubs**
STP Bit-vector & Array Solver[1,2]	Abstraction-refinement for Solving	Concolic Testing	CAV 2007 CCS 2006 TISSEC 2008
HAMPI String Solver[1]	App-driven Bounding for Solving	Analysis of Web Apps	ISSTA 2009[3] TOSEM 2011 (Invited/in submission)
(Un)Decidability results for Strings	Insights from Practical Applications	First results for strings+length	In submission

1. 100+ research projects use STP and HAMPI
2. STP won the SMTCOMP 2006 and 2010 competitions for bit-vector solvers
3. ACM Best Paper Award 2009

Rest of the Talk

- STP Bit-vector and Array Solver
  - Why Bit-vectors and Arrays?
  - How does STP scale: Abstraction-refinement
  - Impact: Concolic testing
  - Experimental Results
- HAMPI String Solver
  - Why Strings?
  - How does HAMPI scale: Bounding
  - Impact: String-based program analysis
  - Experimental Results
- Future Work
  - Multicore SAT
  - SAT-based Languages
  - Auto-tuning Solvers
  - Advice-based Solvers

STP Bit-vector & Array Solver

- Bit-vector or machine arithmetic
- Arrays for memory
- C/C++/Java expressions
- NP-complete

Programs Reasoning & STP
Why Bit-vectors and Arrays

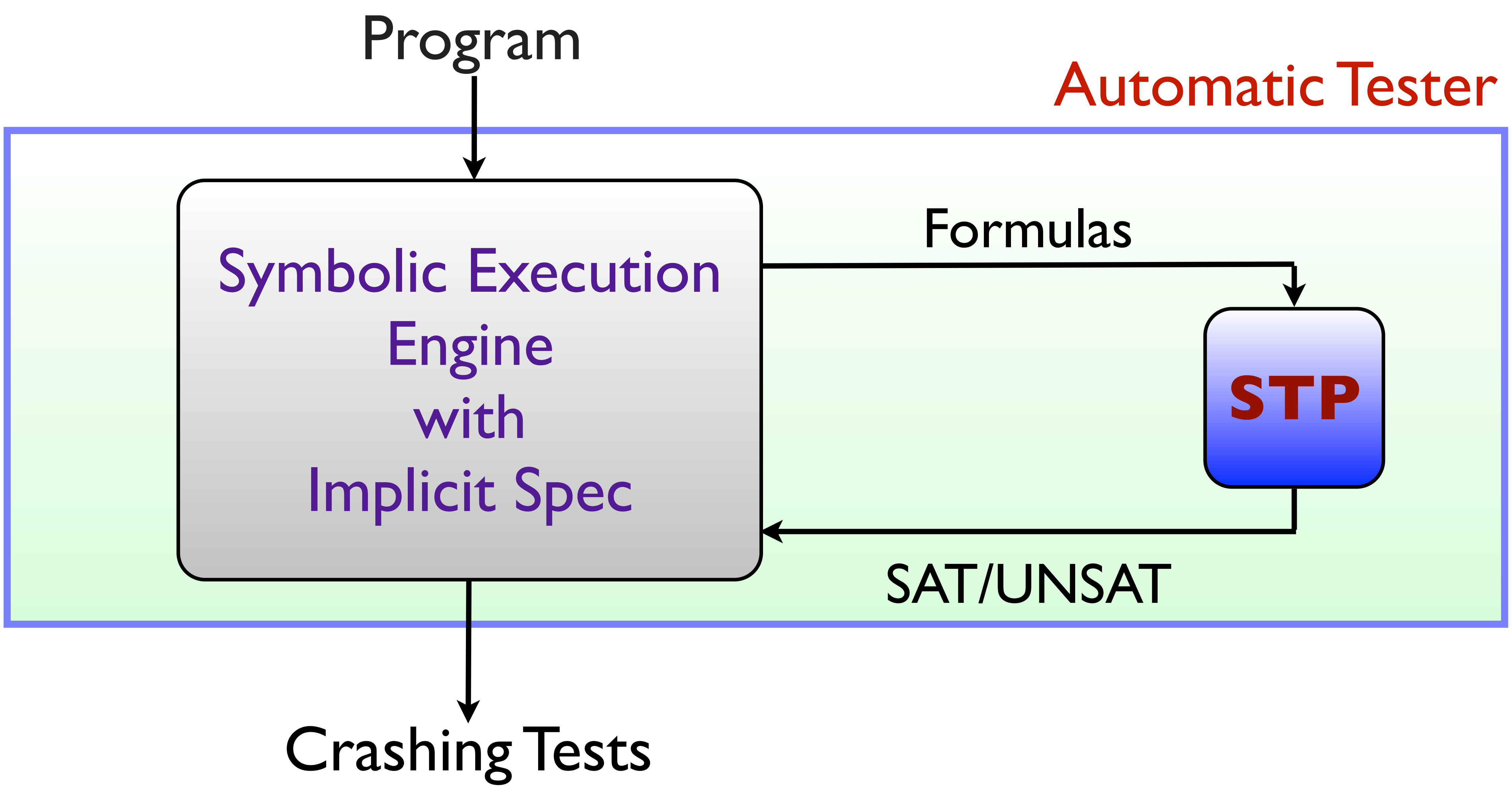
- STP logic tailored for software reliability applications
- Support symbolic execution/program analysis

C/C++/Java/...	Bit-vectors and Arrays
Int Var Char Var	32 bit variable 8 bit variable
Arithmetic operation (x+y, x-y, x*y, x/y,...)	Arithmetic function (x+y,x-y,x*y,x/y,...)
assignments x = expr;	equality x = expr;
if conditional if(cond) x = expr^1 else x = expr^2	if-then-else construct x = if(cond) expr^1 else expr^2
inequality	inequality predicate
Memory read/write x = *ptr + i;	Array read/write ptr[]; x = Read(ptr,i);
Structure/Class	Serialized bit-vector expressions
Function	Symbolic execution
Loops	Bounding

How to Automatically Crash Programs?

Concolic Execution & STP

Problem: Automatically generate crashing tests given only the code



How to Automate Testing?

Concolic Execution & STP

Structured input processing code:
PDF Reader, Movie Player,...

```
Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
      //1. Integer overflow causing NULL deref
      //2. Buffer overflow
      *(ptr+i) = process_data(*(data_field+i));
    }
}
```

- Formula captures computation
- Tester attaches formula to capture spec

How to Automate Testing?

Concolic Execution & STP

Structured input processing code:
PDF Reader, Movie Player,...

```
Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
      //1. Integer overflow causing NULL deref
      //2. Buffer overflow
      *(ptr+i) = process_data(*(data_field+i));
    }
}
```

Equivalent Logic Formula derived using
symbolic execution

```
data_field, mem_ptr : ARRAY;
len_field : BITVECTOR(32); //symbolic
i, j, ptr   : BITVECTOR(32);//symbolic
.
.
mem_ptr[ptr+i] = process_data(data_field[i]);
mem_ptr[ptr+i+1] = process_data(data_field[i+1]);
.
.
```

- Formula captures computation
- Tester attaches formula to capture spec

How to Automate Testing?

Concolic Execution & STP

Structured input processing code:
PDF Reader, Movie Player,...

```
Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
      //1. Integer overflow causing NULL deref
      //2. Buffer overflow
      *(ptr+i) = process_data(*(data_field+i));
    }
}
```

Equivalent Logic Formula derived using symbolic execution

```
data_field, mem_ptr :  ARRAY;
len_field : BITVECTOR(32); //symbolic
i,  j, ptr    : BITVECTOR(32);//symbolic
.
.
mem_ptr[ptr+i] = process_data(data_field[i]);
mem_ptr[ptr+i+1] = process_data(data_field[i+1]);
.
.
```

- Formula captures computation
- Tester attaches formula to capture spec

How to Automate Testing?

Concolic Execution & STP

Structured input processing code:
PDF Reader, Movie Player,...

```
Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
      //1. Integer overflow causing NULL deref
      //2. Buffer overflow
      *(ptr+i) = process_data(*(data_field+i));
    }
}
```

↔

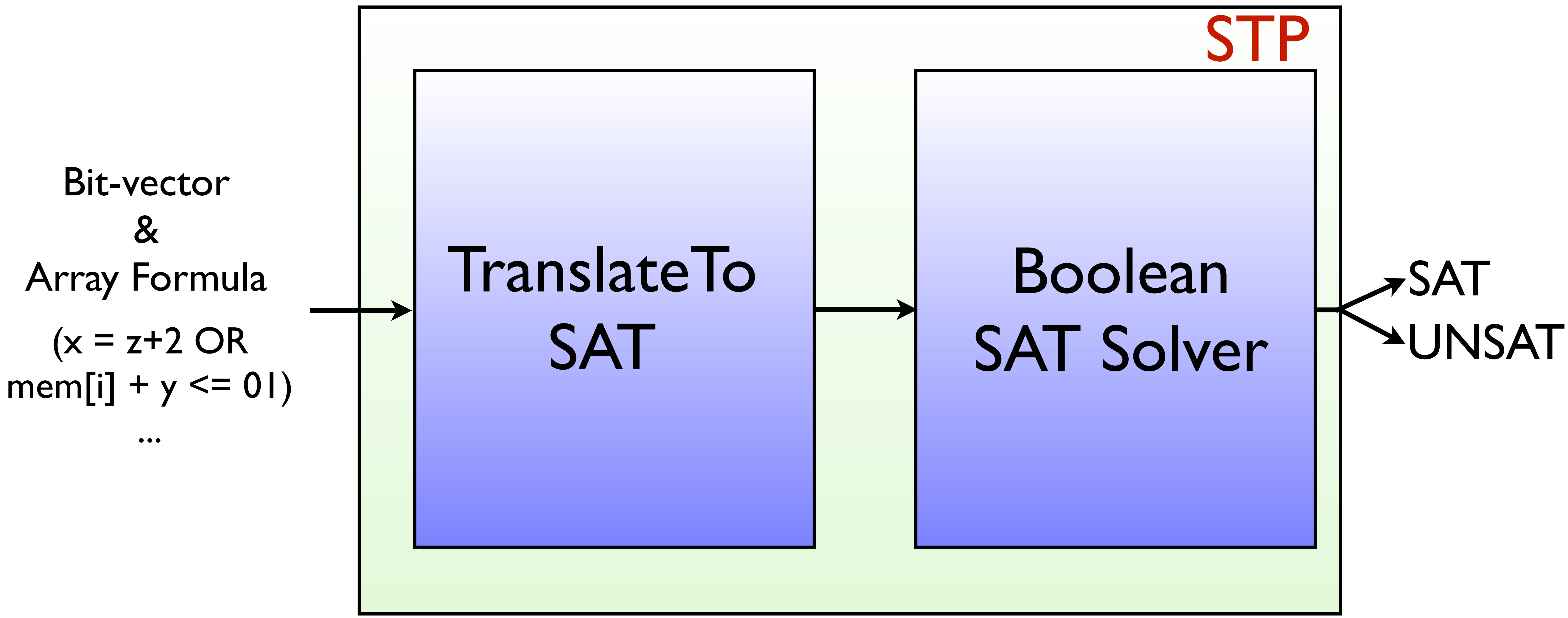
Equivalent Logic Formula derived using symbolic execution

```
data_field, mem_ptr :  ARRAY;
len_field : BITVECTOR(32); //symbolic
i,  j, ptr     : BITVECTOR(32);//symbolic
.
.
mem_ptr[ptr+i] = process_data(data_field[i]);
mem_ptr[ptr+i+1] = process_data(data_field[i+1]);
.
.
//INTEGER OVERFLOW QUERY
0 <= j <= process(len_field);
ptr + i + j = 0?
```

- Formula captures computation
- Tester attaches formula to capture spec

How STP Works

Bird's Eye View: Translate to SAT

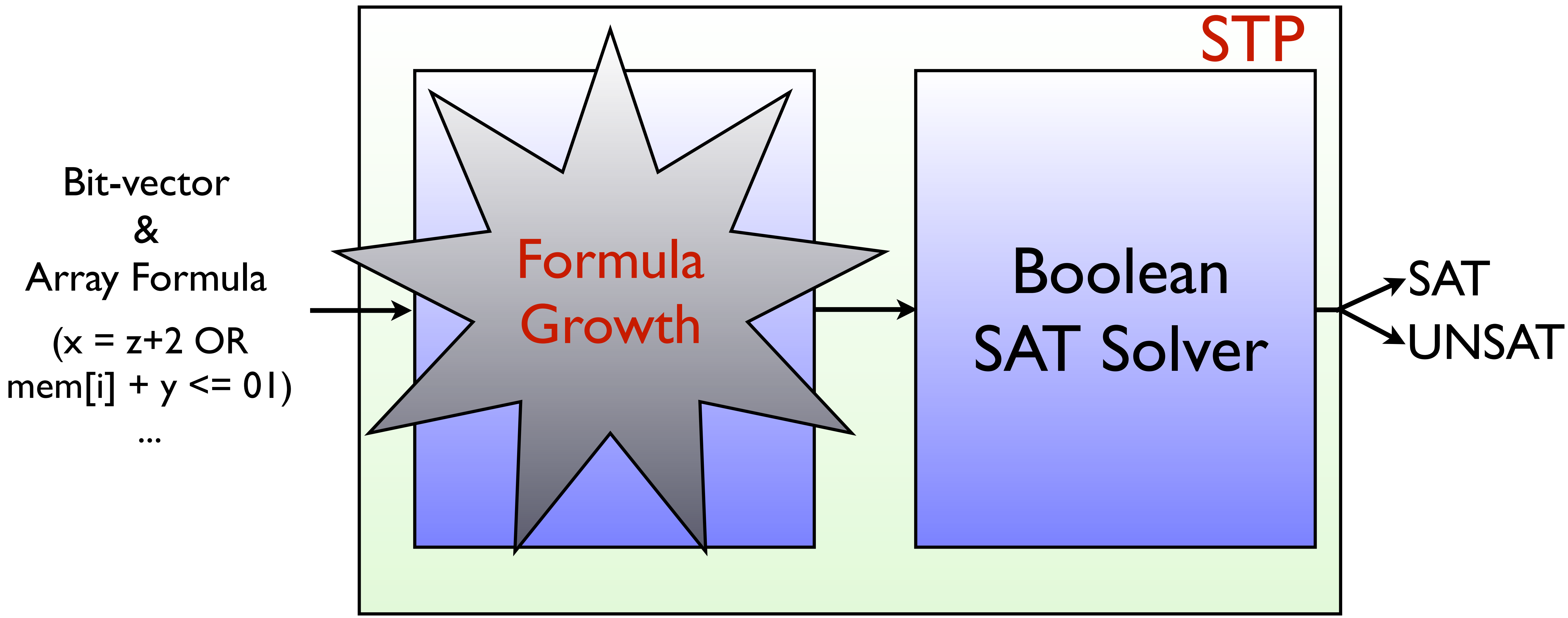


Why Translate to SAT?

- Both theories NP-complete
- Non SAT approaches didn't work
- Translation to SAT leverages solid engineering

How STP Works

Rich Theories cause MEM Blow-up



- Making information explicit
  - Space cost
  - Time cost

Explicit Information causes Blow-up

Array Memory Read Problem

Logic Formula derived using
symbolic execution

```
data_field, mem_ptr :  ARRAY;
len_field : BITVECTOR(32); //symbolic
i,  j, ptr     : BITVECTOR(32);//symbolic
.
.
mem_ptr[ptr+i] = process_data(data_field[i]);
mem_ptr[ptr+i+1] = process_data(data_field[i+1]);
.
.
if(ptr+i = ptr+i+1) then mem_ptr[ptr+i] = mem_ptr[ptr+i+1);

//INTEGER OVERFLOW QUERY
0 <= j <= process(len_field);
ptr + i + j < ptr?
```

- Array Aliasing is implicit
- Need to make information explicit during solving
- Cannot be avoided

How STP Works

Array-read MEM Blow-up Problem

- Problem: $O(n^2)$ axioms added, n is number of read indices
- Lethal, if n is large, say, n = 100,000; # of axioms is 10 Billion

Formula Growth

$Read(Mem,i_0) = expr_0$
$Read(Mem,i_1) = expr_1$
$Read(Mem,i_2) = expr_2$
.
.
.
$Read(Mem,i_n) = expr_n$

$v_0 = expr_0$
$v_1 = expr_1$
.
.
.
$v_n = expr_n$

$(i_0 = i_1) \Rightarrow (v_0 = v_1)$
$(i_0 = i_2) \Rightarrow (v_0 = v_2)$
...
$(i_1 = i_2) \Rightarrow (v_1 = v_2)$
...

How STP Works

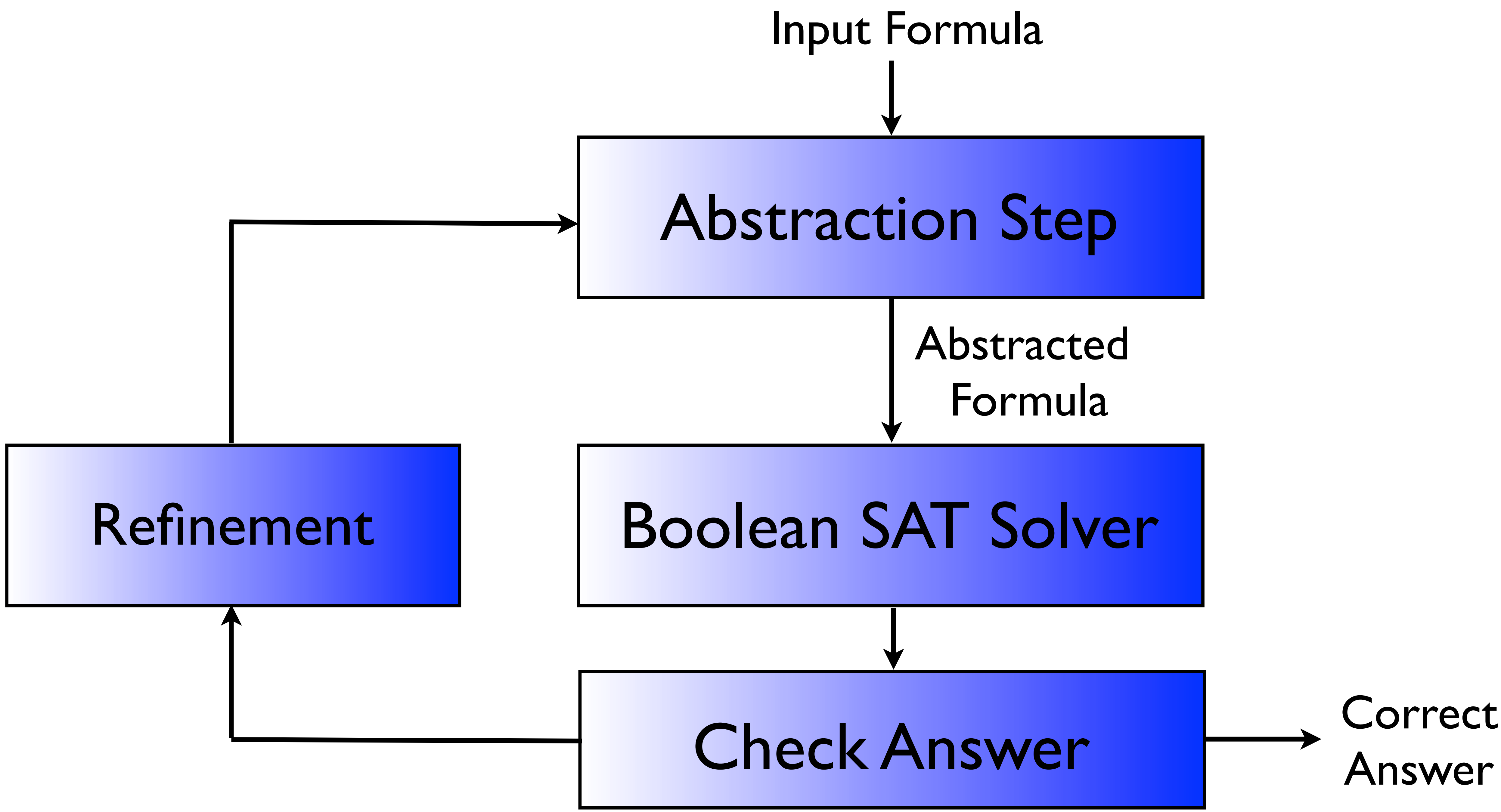
The Array-read Solution

- Key Observation
  - Most indices don't alias in practice
  - Exploit locality of memory access in typical programs
  - Need only a fraction of array axioms for equivalence

$$
\begin{array}{l}
Read(Mem,i_0) = expr_0 \\
Read(Mem,i_1) = expr_1 \\
Read(Mem,i_2) = expr_2 \\
\quad\quad\quad \vdots \\
Read(Mem,i_n) = expr_n
\end{array}
\quad \Longleftrightarrow \quad
\begin{array}{l}
v_0 = expr_0 \\
v_1 = expr_1 \\
\quad \vdots \\
v_n = expr_n \\
\\
(i_0 = i_1) \Rightarrow (v_0 = v_1)
\end{array}
$$

STP Key Conceptual Contribution

Abstraction-refinement Principle



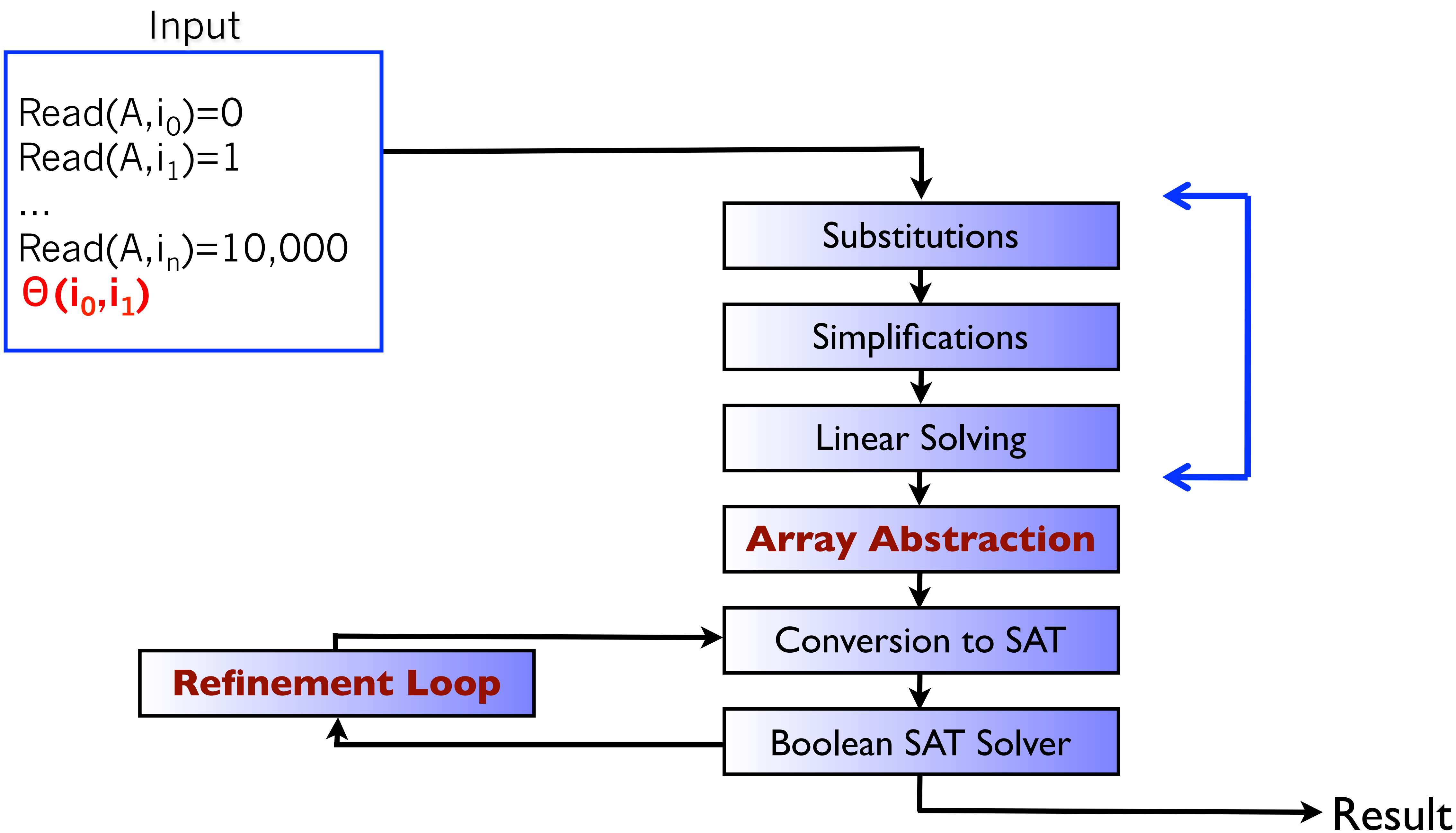
How STP Works

What to Abstract & How to Refine?

Abstraction	Refinement
1. Less essential parts 2. Causes MEM blow-up	1. Guided 2. Must remember
Abstraction manages formula growth hardness	Refinement manages search-space hardness

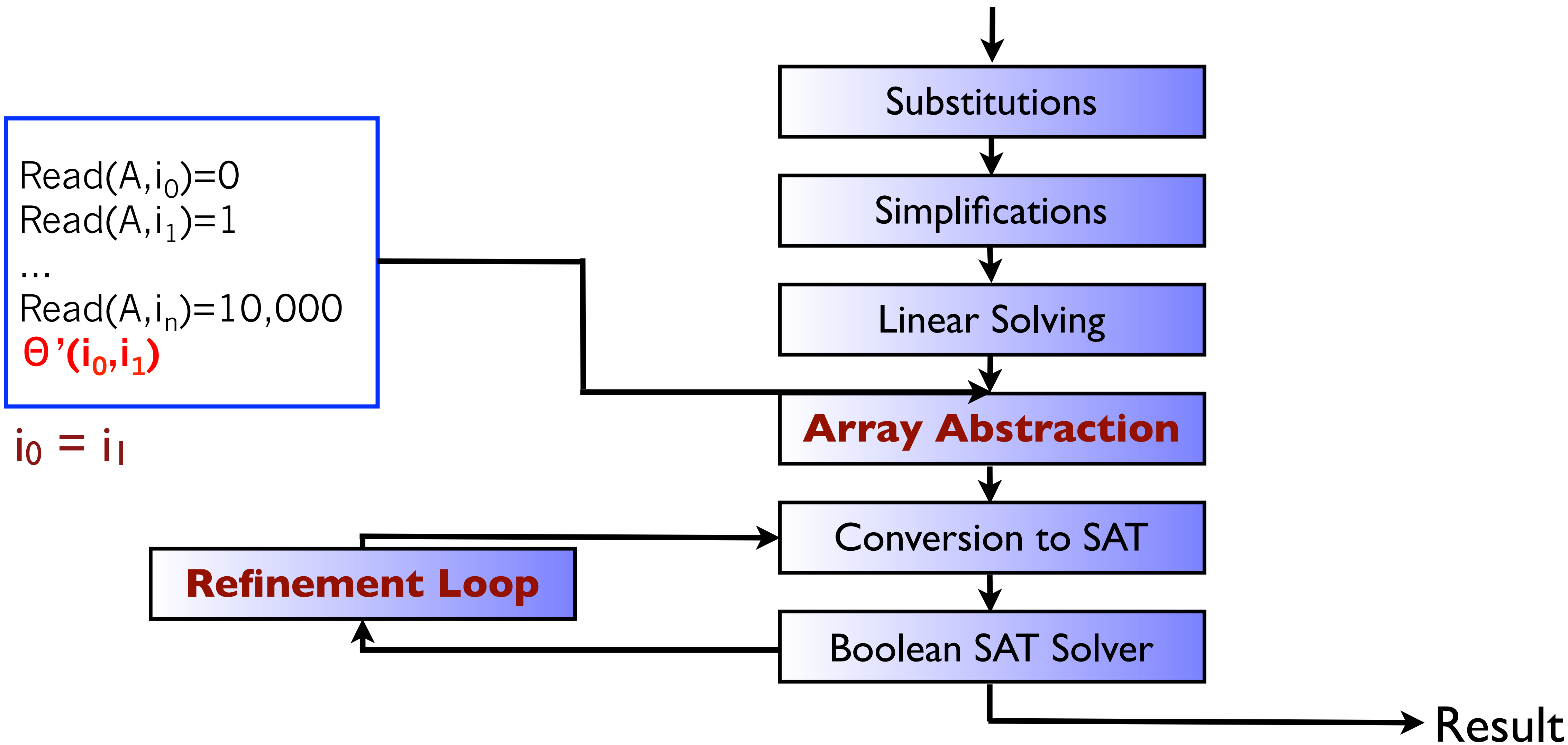
How STP Works

Abstraction-refinement for Array-reads



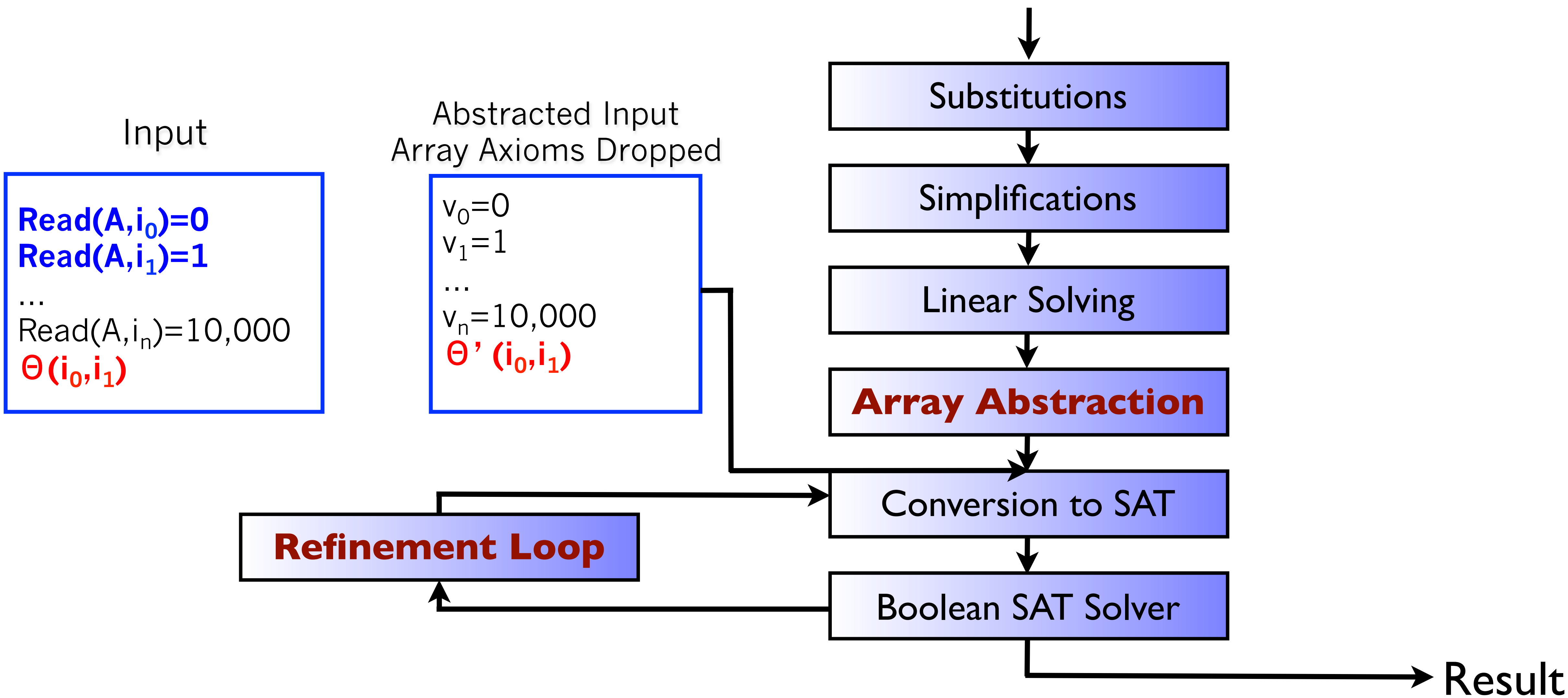
How STP Works

Abstraction-refinement for Array-reads



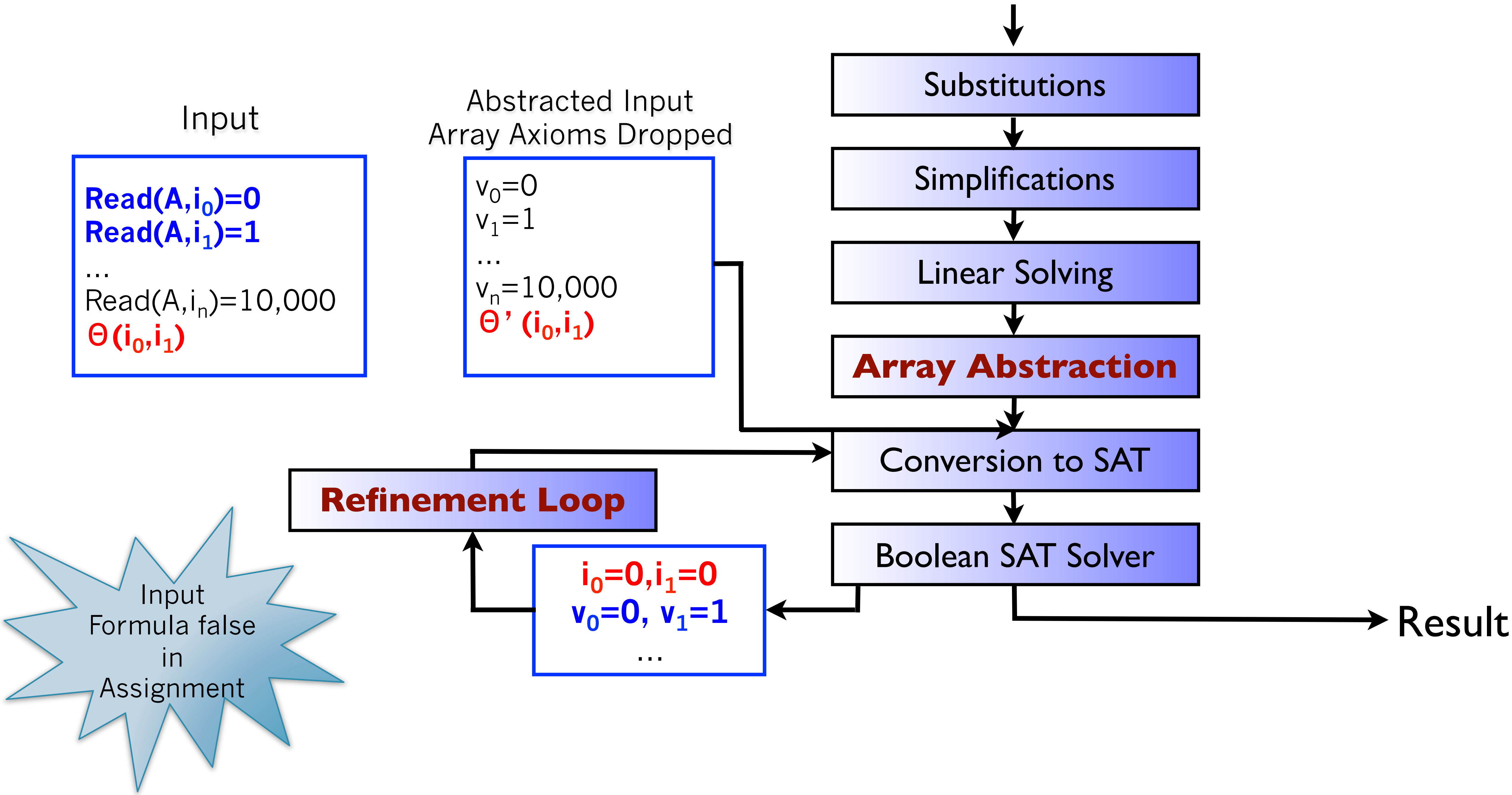
How STP Works

Abstraction-refinement for Array-reads



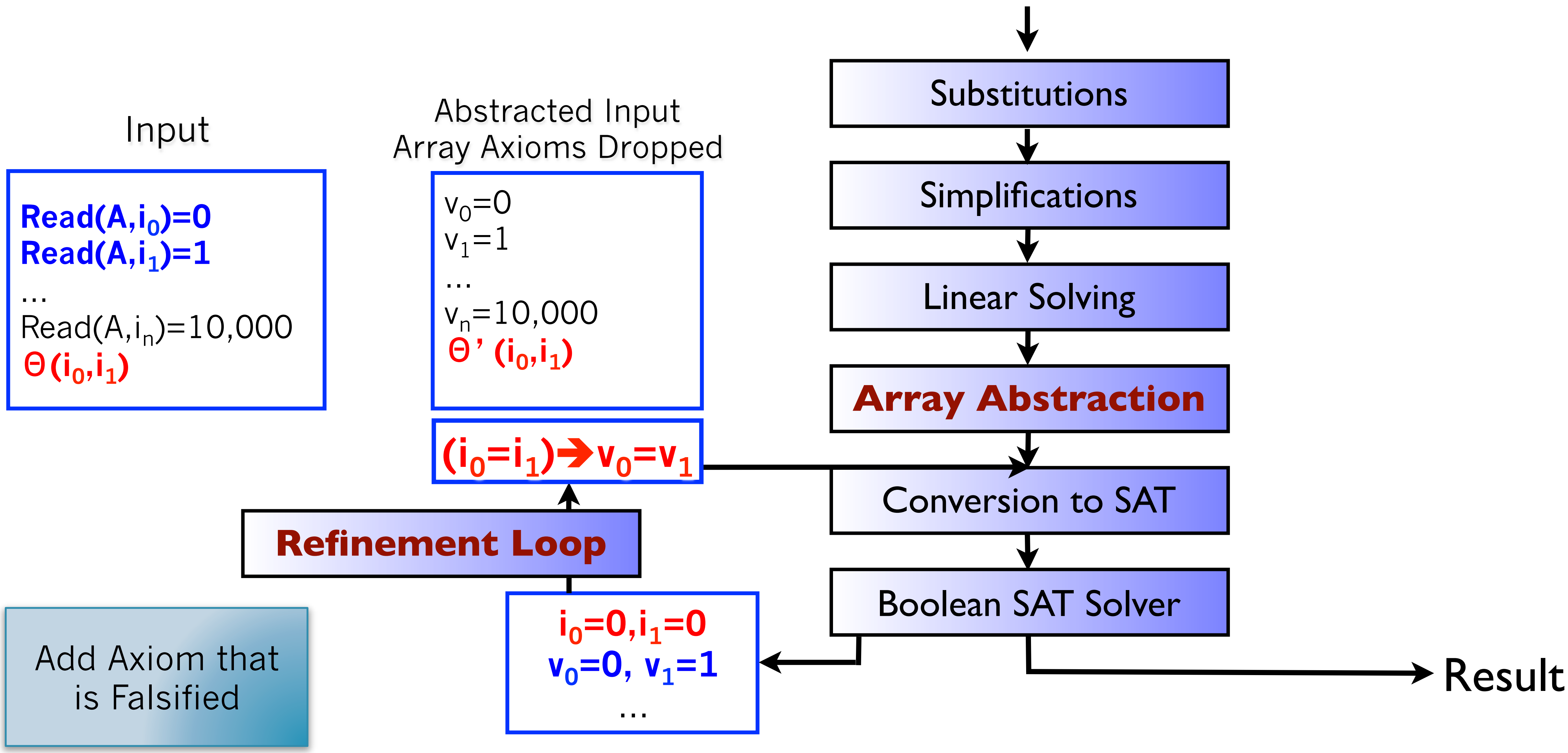
How STP Works

Abstraction-refinement for Array-reads



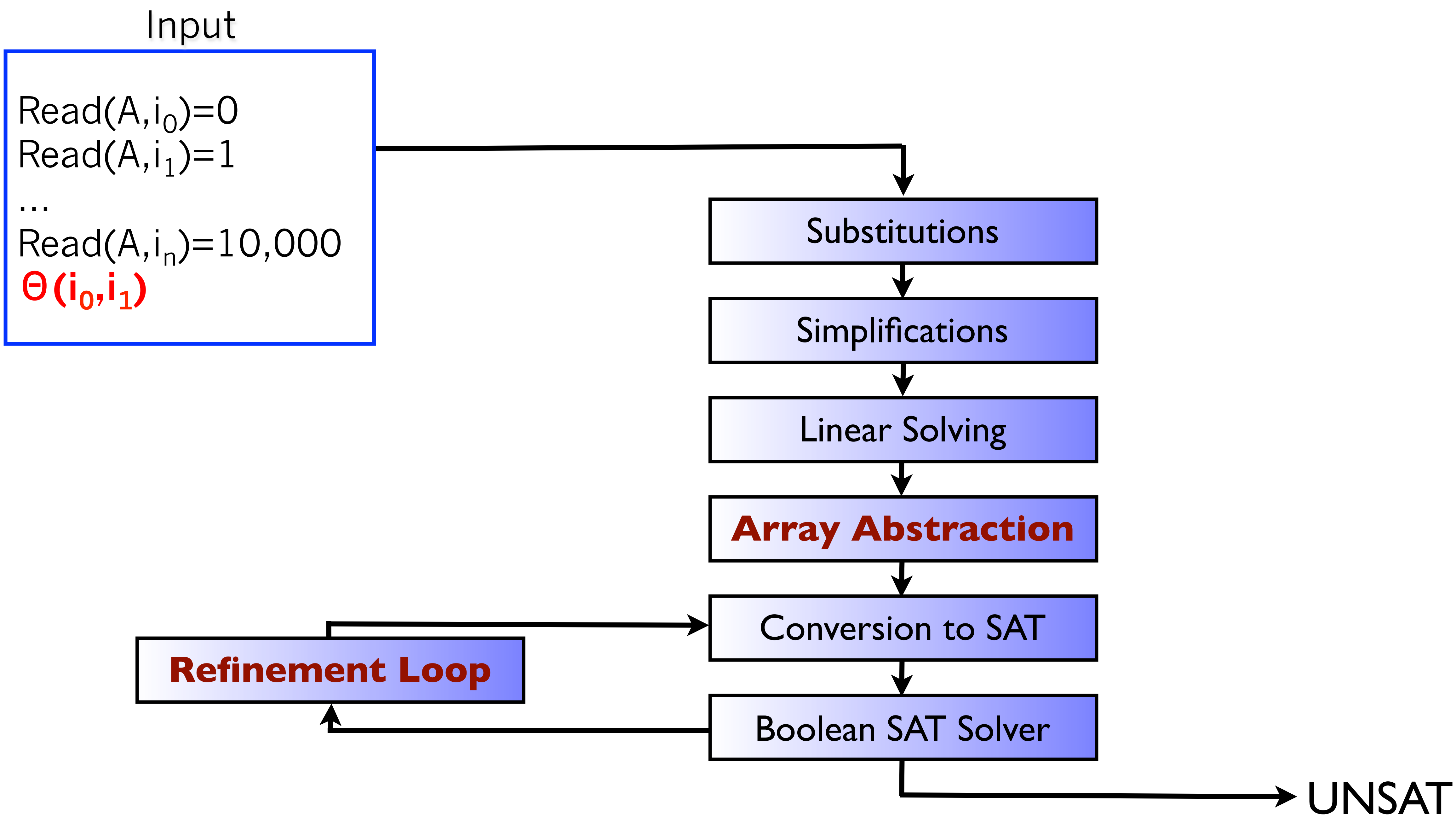
How STP Works

Abstraction-refinement for Array-reads



How STP Works

Abstraction-refinement for Array-reads

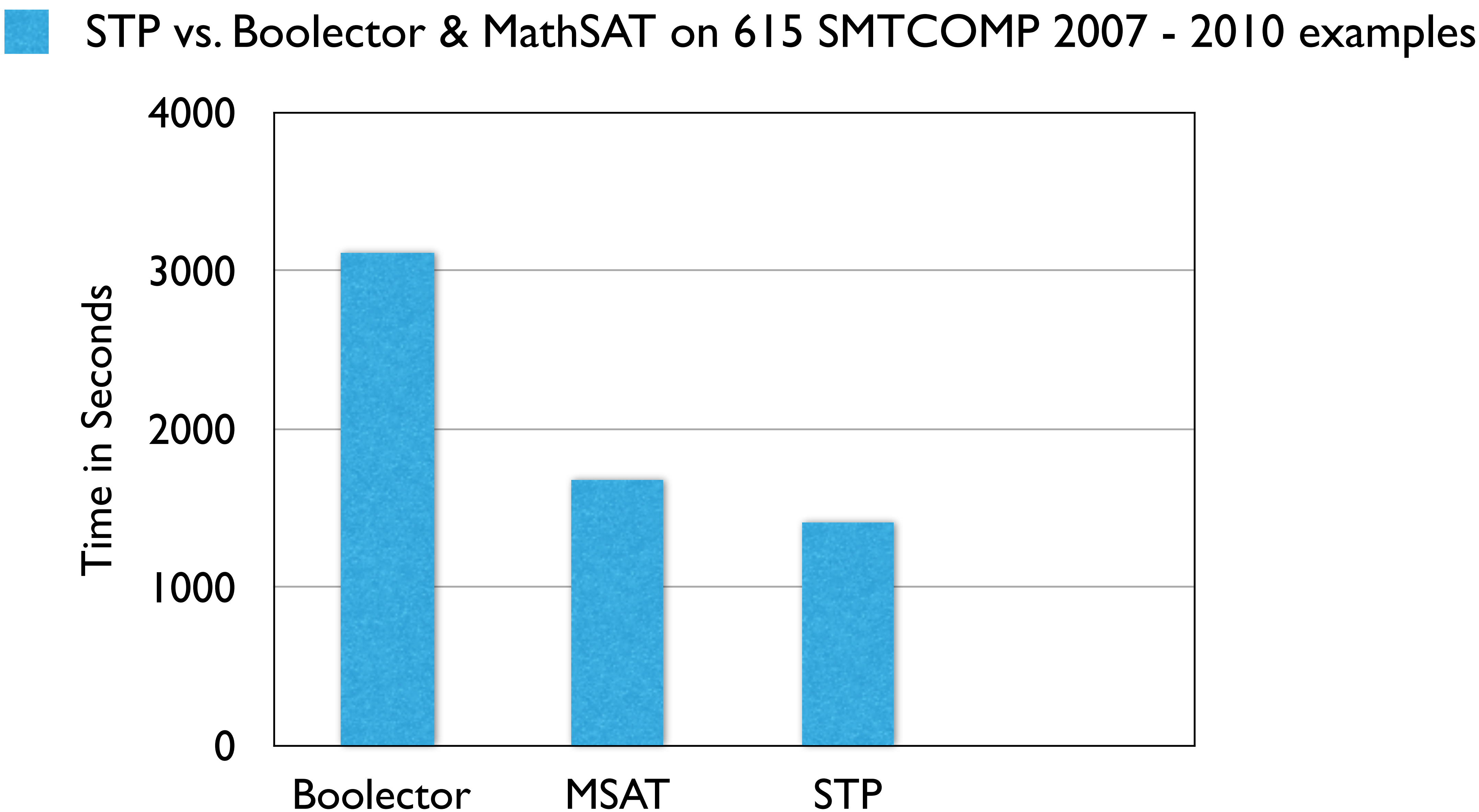


STP vs. Other Solvers

Testcase (Formula Size)	Result	Z3 (sec)	Yices (sec)	STP (sec)
610dd9c (~15K)	SAT	TimeOut	MemOut	37
Grep65 (~60K)	UNSAT	0.3	TimeOut	4
Grep84 (~69K)	SAT	176	TimeOut	18
Grep106 (~69K)	SAT	130	TimeOut	227
Blaster4 (~262K)	UNSAT	MemOut	MemOut	10
Testcase20 (~1.2M)	SAT	MemOut	MemOut	56
Testcase21 (~1.2M)	SAT	MemOut	MemOut	43

* All experiments on 3.2 GHz, 512 Kb cache
* MemOut: 3.2 GB (Memory used by STP much smaller), TimeOut: 1800 seconds
* Examples obtained from Dawn Song at Berkeley, David Molnar at Berkeley and Dawson Engler at Stanford
* Experiments conducted in 2007

STP vs. Other Leading Solvers



* All experiments on 2.4 GHz, 1 GB RAM
* Timeout: 500 seconds/example

Impact of STP

- Enabled existing SE technologies to scale
  - Bounded model checkers, e.g., Chang and Dill

- Easier to engineer SE technologies
  - Formal tools (ACL2+STP) for verifying Crypto, Smith & Dill

- Enabled new SE technologies
  - Concolic testing (EXE,Klee,...) by Engler et al., Binary Analysis by Song et al.

Impact of STP: Notable Projects

- Enabled Concolic Testing
- 100+ reliability and security projects

Category	**Research Project**	**Project Leader/Institution**
Formal Methods	ACL2 Theorem Prover + STP Verification-aware Design Checker Java PathFinder Model Checker	Eric Smith & David Dill/Stanford Jacob Chang & David Dill/Stanford Mehlitz & Pasareanu/NASA
Program Analysis	BitBlaze & WebBlaze BAP	Dawn Song et al./Berkeley David Brumley/CMU
Automatic Testing Security	Klee, EXE SmartFuzz Kudzu	Engler & Cadar/Stanford Molnar & Wagner/Berkeley Saxena & Song/Berkeley
Hardware Bounded Model-cheking (BMC)	Blue-spec BMC BMC	Katelman & Dave/MIT Haimed/NVIDIA

Impact of STP

http://www.metafuzz.com

Program Name	Lines of Code	Number of Bugs Found	Team
Mplayer	~900,000	Hundreds	David Molnar/Berkeley & Microsoft Research
Evince	~90,000	Hundreds	David Molnar/Berkeley & Microsoft Research
Unix Utilities	1000s	Dozens	Dawson Engler et al./Stanford
Crypto Hash Implementations	1000s	Verified	Eric Smith & David Dill/Stanford

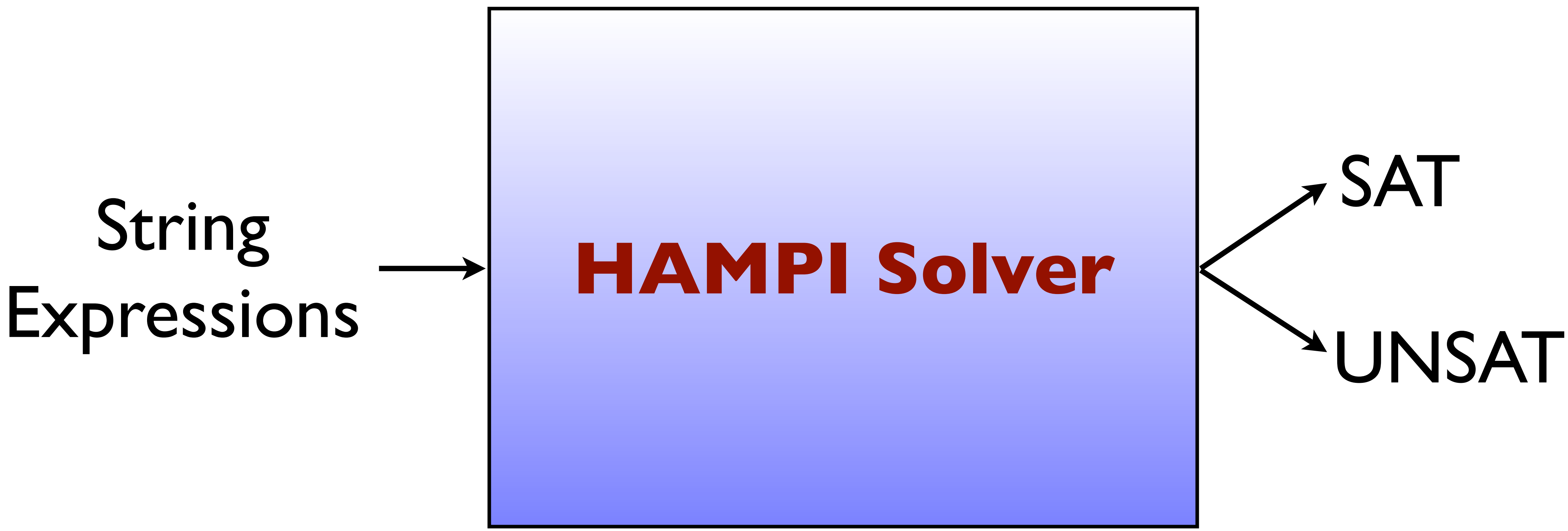
Rest of the Talk

- STP Bit-vector and Array Solver
  - Why Bit-vectors and Arrays?
  - How does STP scale: Abstraction-refinement
  - Impact: Concolic testing
  - Experimental Results

- HAMPI String Solver
  - Why Strings?
  - How does HAMPI scale: Bounding
  - Impact: String-based program analysis
  - Experimental Results

- Future Work
  - Multicore SAT
  - SAT-based Languages

HAMPI String Solver



- X = concat(“SELECT...”,v) AND (X $\in$ SQL_grammar)
- JavaScript and PHP Expressions
- Web applications, SQL queries
- NP-complete

What is the theory of Strings?

- Capture String Expressions in PHP, JavaScript, Perl, C/C++/Java
- Support symbolic execution/program analysis

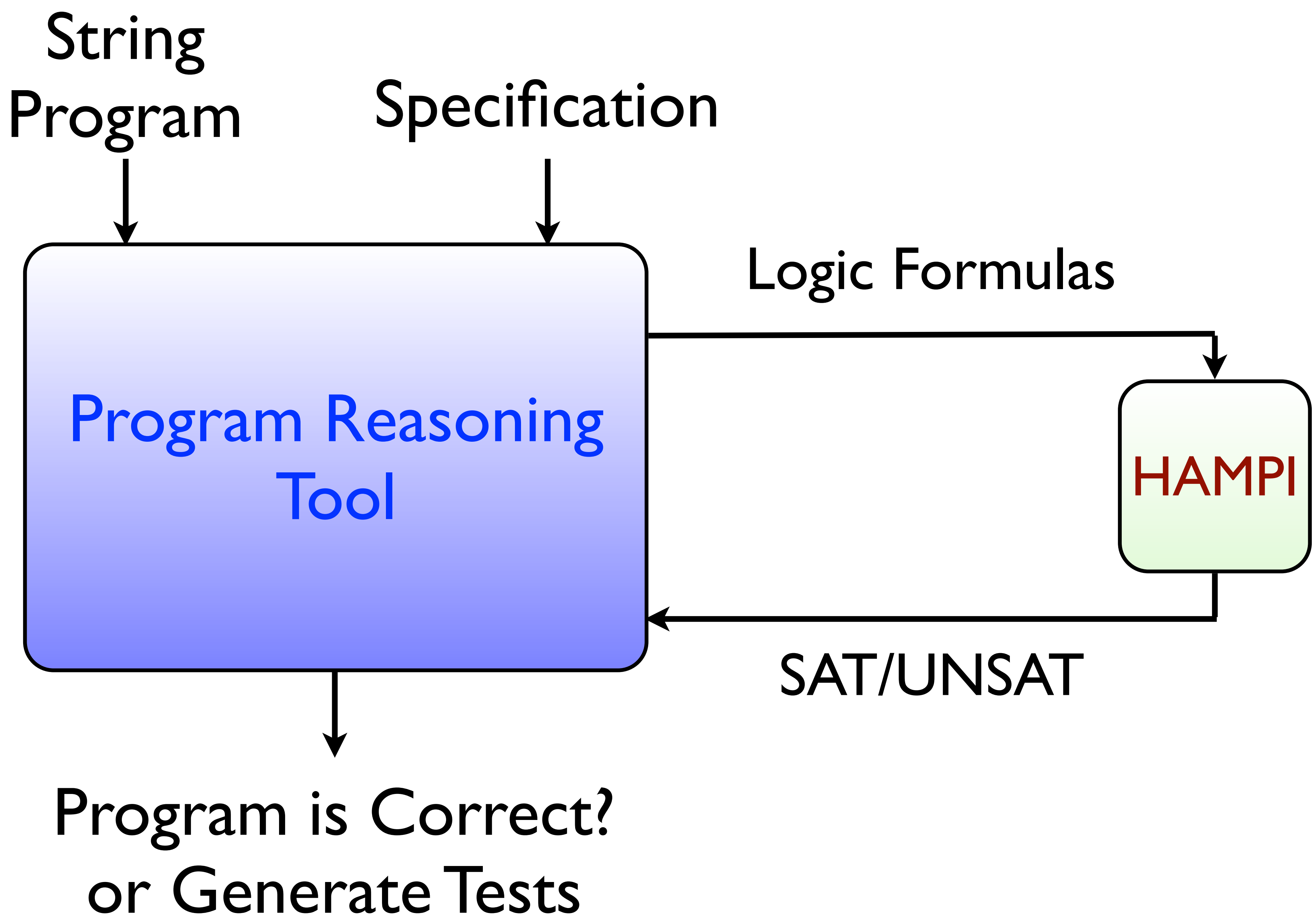
PHP/JavaScript/C++...	Theory of Strings

Var a;
$a = 'name'
	Var a : 12; //String variable of bounded-size
	a = 'name'

a." is "	Concat(a," is ");
substr(a,1,3)	sub-string extraction
assignments/strcmp a = string_expr;	equality a = string_expr;
Sanity check using regular expression RE Expression in a suitable Language (e.g., SQL)	a in RE a in SQL

Hampi Use-case

String Operations in PHP, JavaScript,...



Hampi Use-case

SQL Injection Vulnerabilities

Buggy PHP/
JavaScript

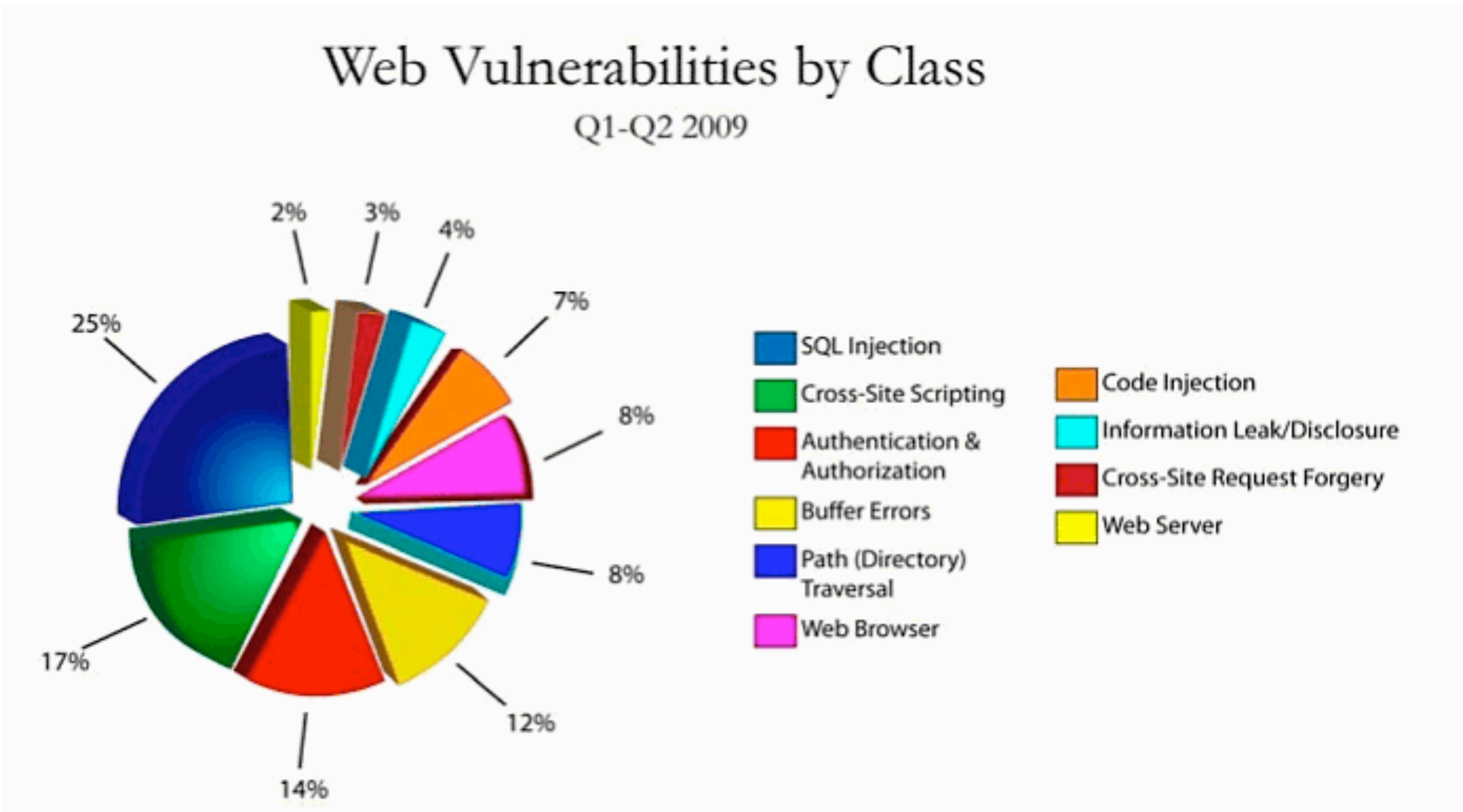
Malicious SQL Query

Unauthorized
Database Results

Backend
DataBase

Hampi Use-case

SQL Injection Vulnerabilities



Hampi Use-case
SQL Injection Vulnerabilities

Buggy Script

```
if (input in regexp("[0-9]+"))
   query := "SELECT m FROM messages WHERE id=' " + input + " ' ")
```

- input passes validation (regular expression check)
- query is syntactically-valid SQL
- query can potentially contain an attack substring (e.g., 1' OR '1' = '1)

Hampi Use-case
SQL Injection Vulnerabilities

Should be: “^[0-9]+$”

Buggy Script

```
if (input in regexp(“[0-9]+”))
   query := “SELECT m FROM messages WHERE id=‘ ” + input +  “ ’ “)
```

- input passes validation (regular expression check)
- query is syntactically-valid SQL
- query can potentially contain an attack substring (e.g., 1’ OR ‘1’ = ‘1)

Hampi Use-case
SQL Injection Vulnerabilities

Input String
```
Var v : 12;
```

SQL Grammar

cfg *SqlSmall* := "SELECT ” [a-z]+ " FROM ” [a-z]+ " WHERE " *Cond*;

cfg *Cond* := *Val* "=" *Val* | *Cond* " OR " *Cond*;

cfg *Val* := [a-z]+ | "’” [a-z0-9]* "’" | [0-9]+;

SQL Query

val q := concat("SELECT msg FROM messages WHERE topicid='", v, "'");

SQLI attack conditions

assert v in [0-9]+;

assert q in *SqlSmall*;

“q is a valid SQL query”

assert q contains "OR ‘1'=‘1'";

“q contains an attack vector”

Hampi finds an attack input: v ≔ 1’ OR ‘1’ = ‘1

SELECT msg FROM messages WHERE topicid=1’ OR ‘1’=‘1’

Hampi Key Contribution: Bounded Logics

Testing, Vulnerability Detection,...

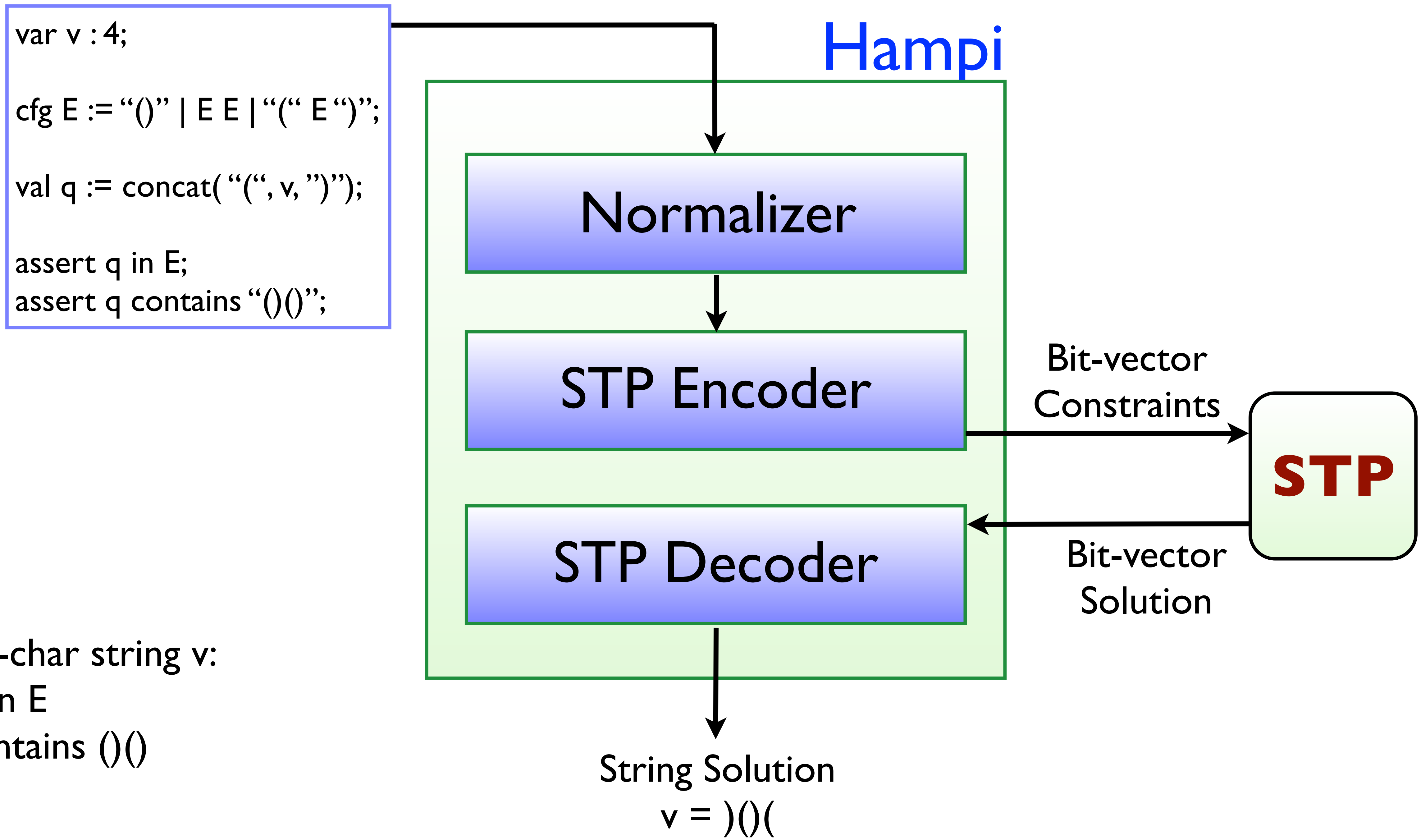
- Finding satisfying assignment is key
- Short assignments are sufficient
- Hence, bounding strings is sufficient
- Bounded logics are easier to decide

Hampi Key Conceptual Contribution

Bounding, expressiveness and efficiency

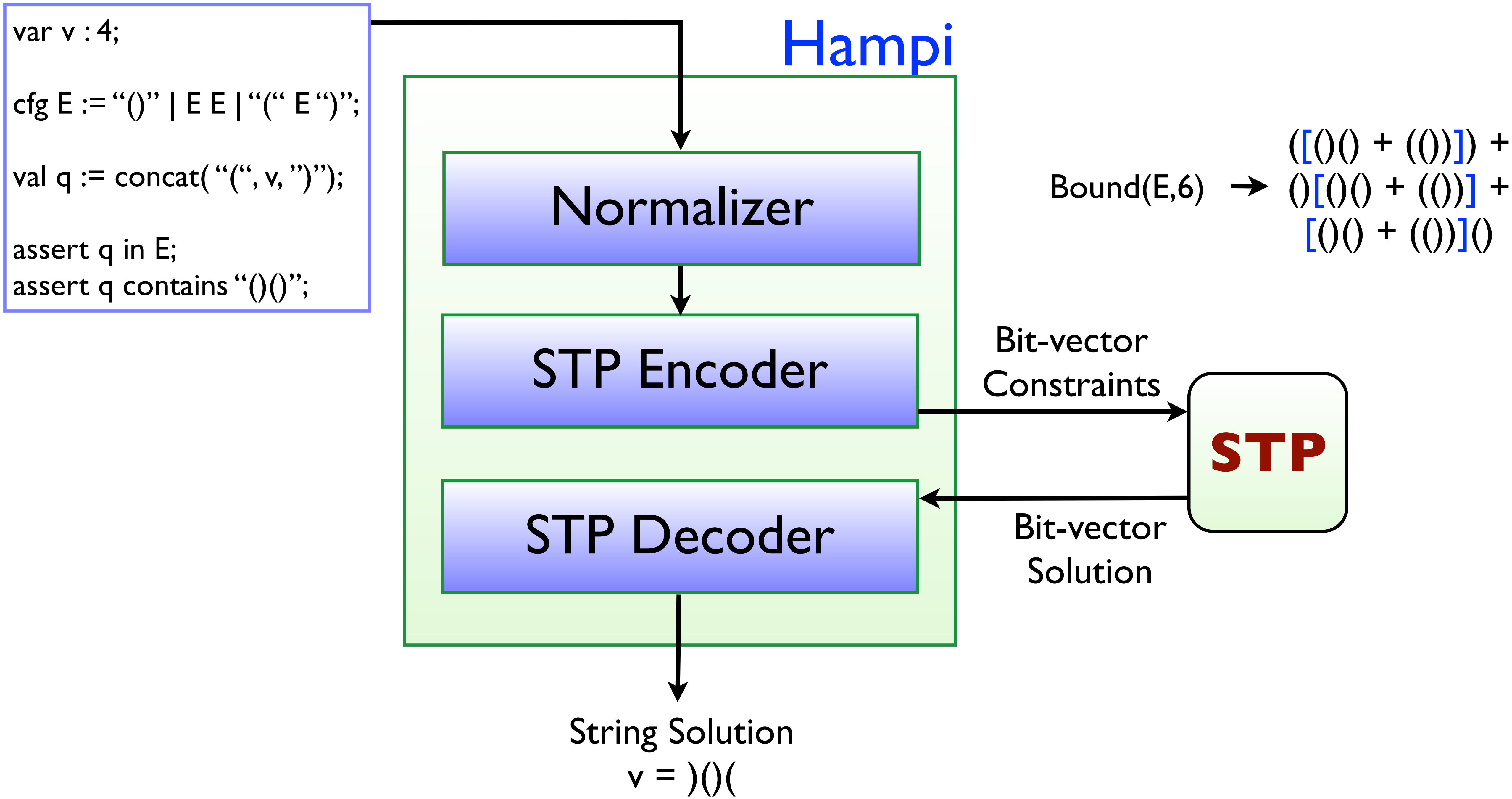
L_i	Complexity of $\emptyset = L_1 \cap \ldots \cap L_n$	Current Solvers
Context-free	Undecidable	n/a
Regular	PSPACE-complete	Quantified Boolean Logic
Bounded	NP-complete	SAT Efficient in practice

How Hampi Works
Bird’s Eye View: Strings into Bit-vectors



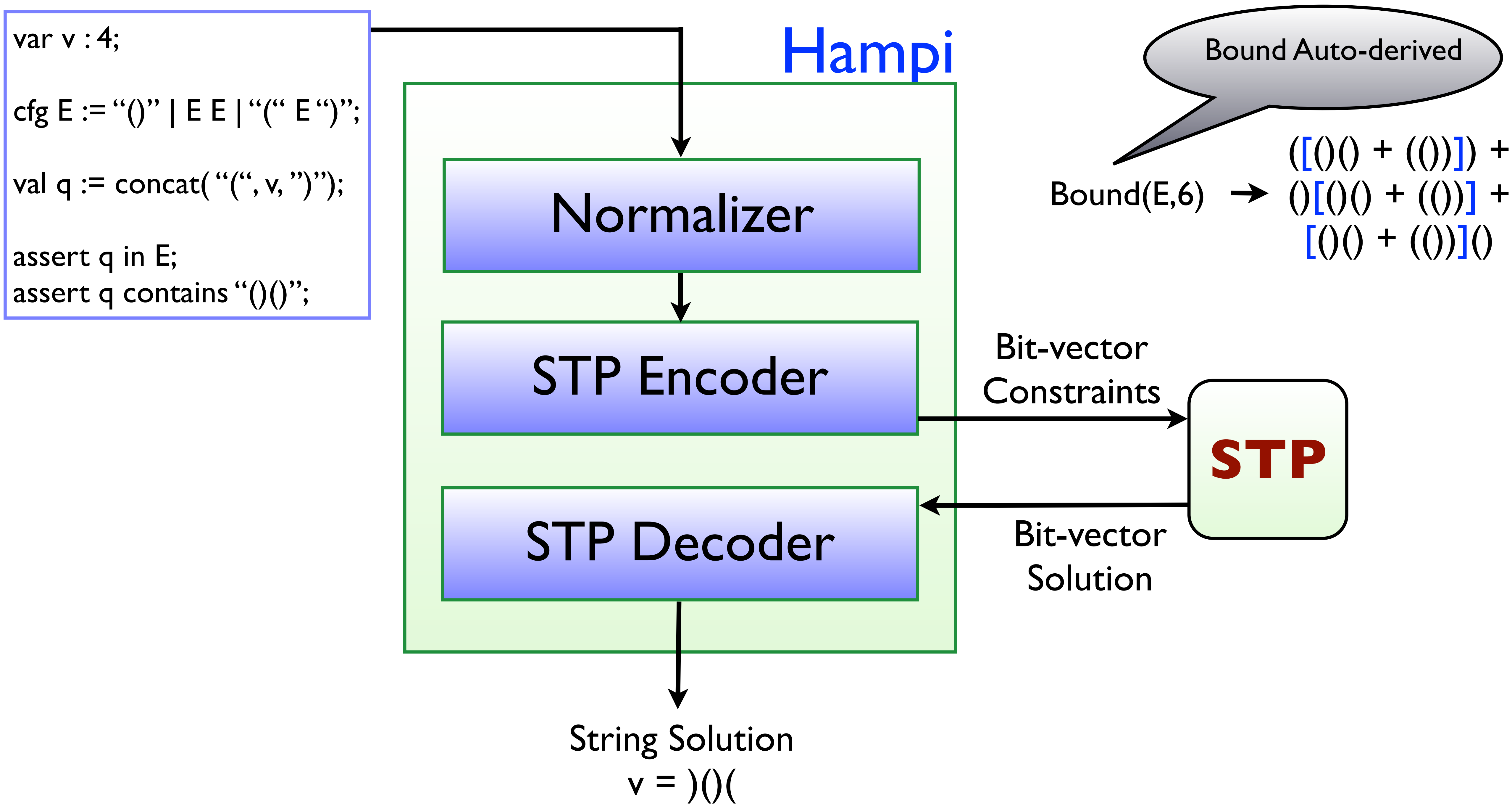
Find a 4-char string v:
- (v) is in E
- (v) contains ()()

How Hampi Works
Unroll Bounded CFGs into Regular Exp.



How Hampi Works

Unroll Bounded CFGs into Regular Exp.



How Hampi Works

Converting Regular Exp. into Bit-vectors

Encode regular expressions recursively

- Alphabet { (,) } → 0, 1
- constant → bit-vector constant
- union + → disjunction $\vee$
- concatenation → conjunction $\wedge$
- Kleene star * → conjunction $\wedge$
- Membership, equality → equality

(v) ∈ ()[()() + (())] + [()() + (())]() + ([()() + (())])

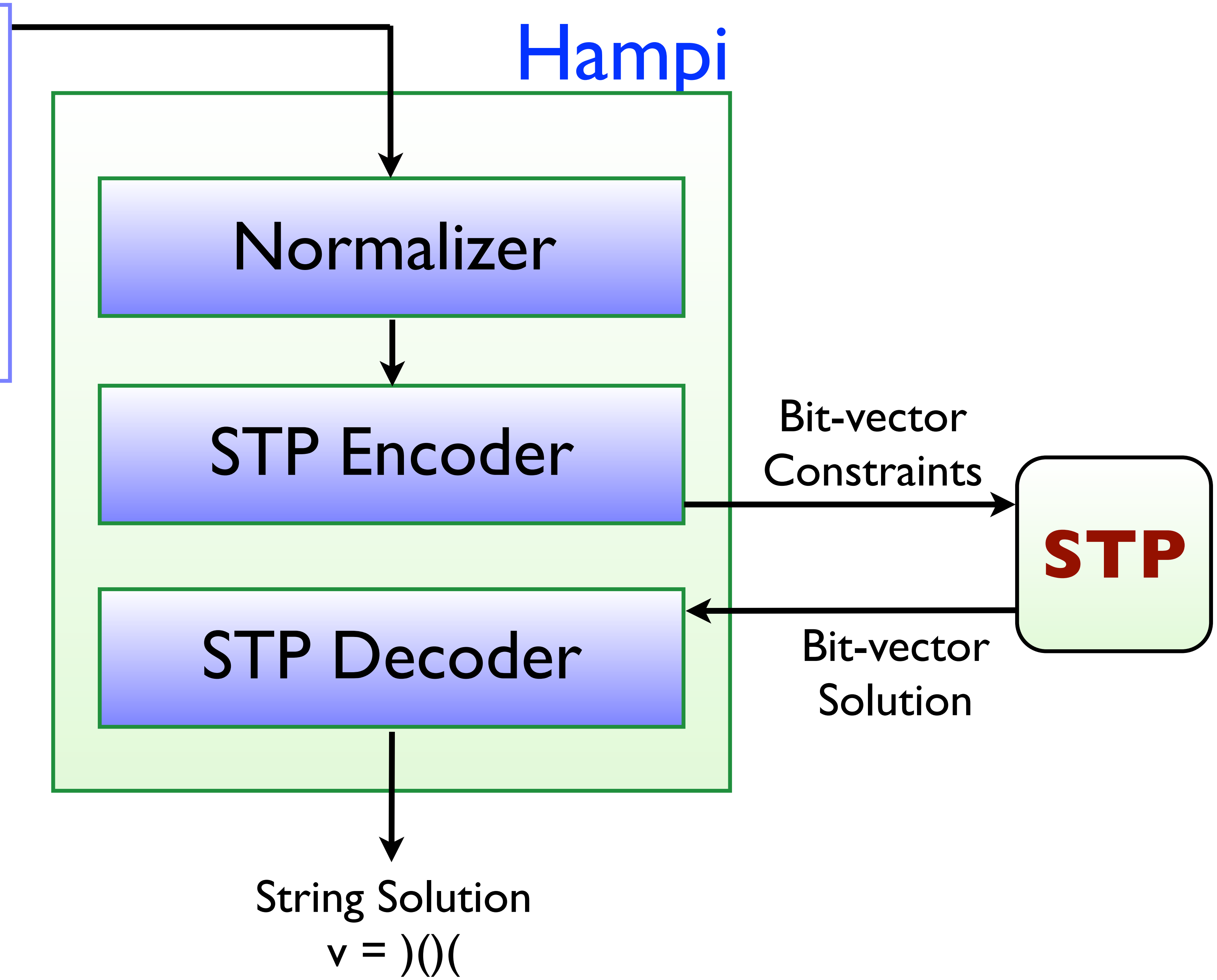
Formula Φ_1 $\vee$ Formula Φ_2 $\vee$ Formula Φ_3

$B[0]=0 \wedge B[1]=1 \wedge \{B[2]=0 \wedge B[3]=1 \wedge B[4]=0 \wedge B[5]=1 \vee \ldots$

How Hampi Works

Decoder converts Bit-vectors to Strings

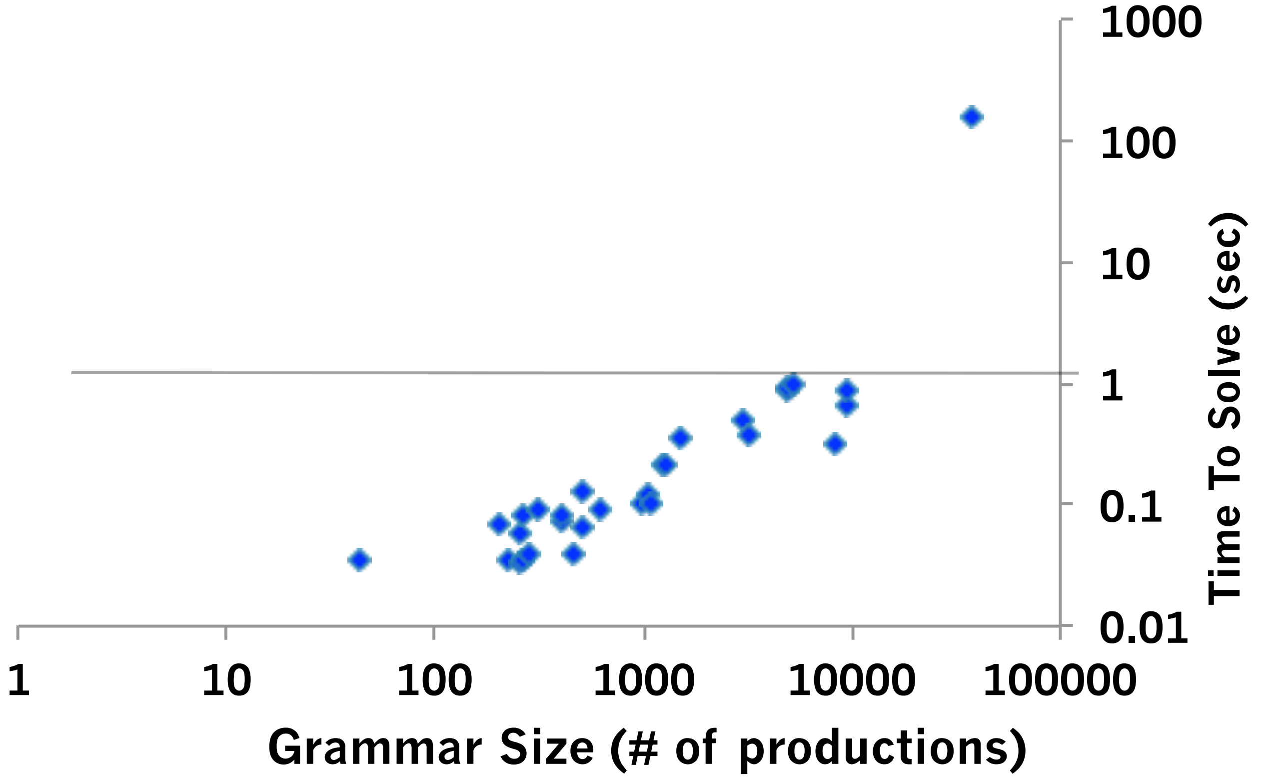
```
var v : 4;

cfg E := “()” | E E | “(“ E “)”;

val q := concat( “(“, v, ”)”);

assert q in E;
assert q contains “()()”;
```



Find a 4-char string v:
- (v) is in E
- (v) contains ()()

HAMPI: Result 1

Static SQL Injection Analysis



- 1367 string constraints from Wasserman & Su [PLDI'07]
- Hampi scales to large grammars
- Hampi solved 99.7% of constraints in < 1sec
- All solvable constraints had short solutions

Monday, May 2, 2011

HAMPI: Result 2
Security Testing

- Hampi used to build Ardilla security tester [Kiezun et al., ICSE'09]

- 60 new vulnerabilities on 5 PHP applications (300+ kLOC)
  - 23 SQL injection
  - 37 cross-site scripting (XSS)

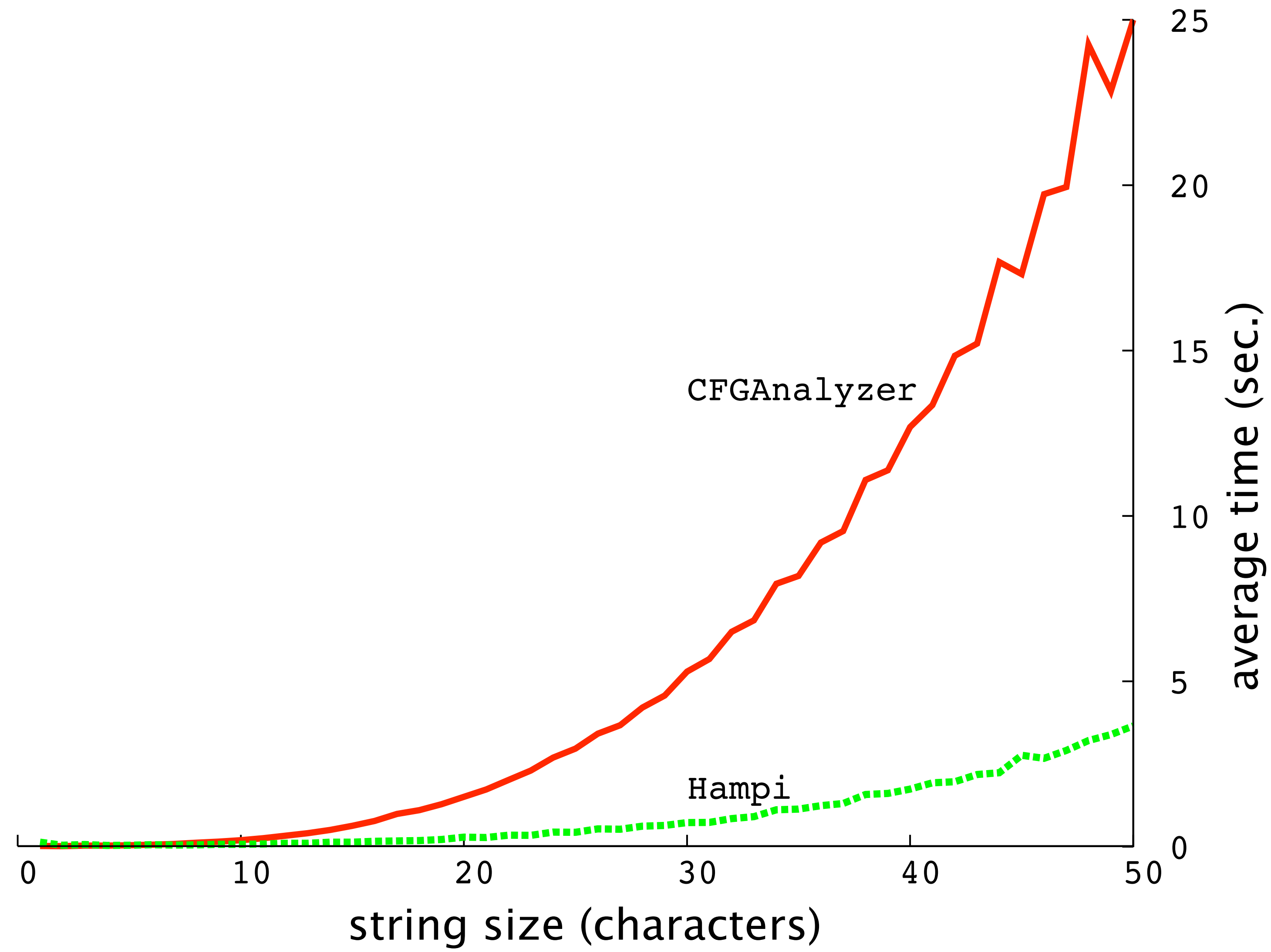
5 added to
US National Vulnerability DB

- 46% of constraints solved in < 1 second per constraint

- 100% of constraints solved in <10 seconds per constraint

HAMPI: Result 3

Comparison with Competing Tools



- HAMPI vs. CFGAnalyzer (U. Munich): HAMPI ~7x faster for strings of size 50+
- HAMPI vs. Rex (Microsoft Research): HAMPI ~100x faster for strings of size 100+
- HAMPI vs. DPRLE (U.Virginia): HAMPI ~1000x faster for strings of size 100+

Impact of Hampi: Notable Projects

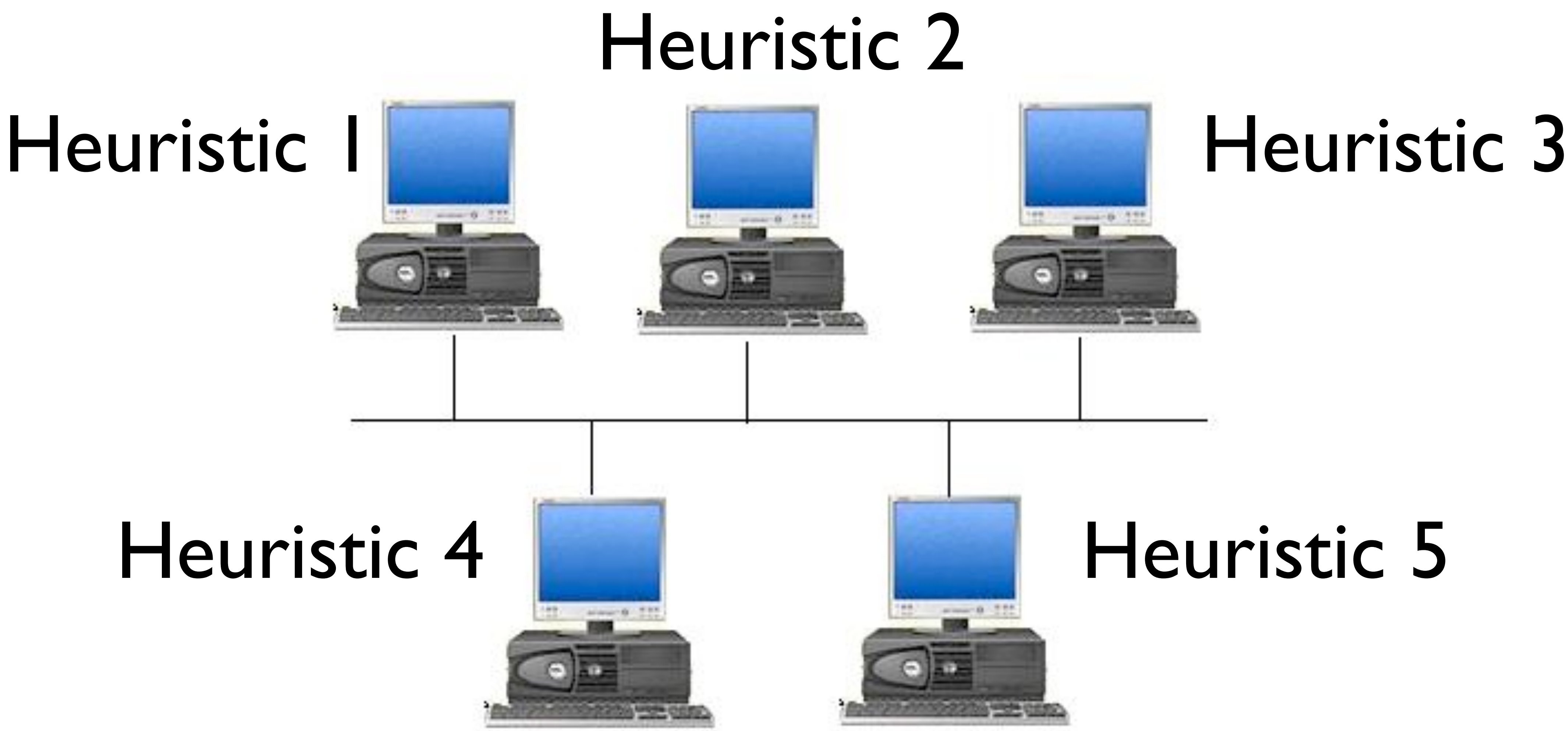
Category	Research Project	Project Leader/Institution
Static Analysis	SQL-injection vulnerabilities	Wasserman & Su/UC, Davis
Security Testing	Ardilla for PHP (SQL injections, cross-site scripting)	Kiezun & Ernst/MIT
Concolic Testing	Klee SAGE Kudzu NoTamper	Engler & Cadar/Stanford Godefroid/Microsoft Research Saxena & Song/Berkeley Bisht & Venkatakrishnan/U Chicago
New Solvers	Kaluza	Saxena & Song/Berkeley

Rest of the Talk

- STP Bit-vector and Array Solver
  - Why Bit-vectors and Arrays?
  - How does STP scale: Abstraction-refinement
  - Impact: Concolic testing
  - Experimental Results
- HAMPI String Solver
  - Why Strings?
  - How does HAMPI scale: Bounding
  - Impact: String-based program analysis
  - Experimental Results
- Future Work
  - Multicore SAT
  - SAT-based Languages

Current Parallel SAT Approaches
Won't Scale with more Nodes

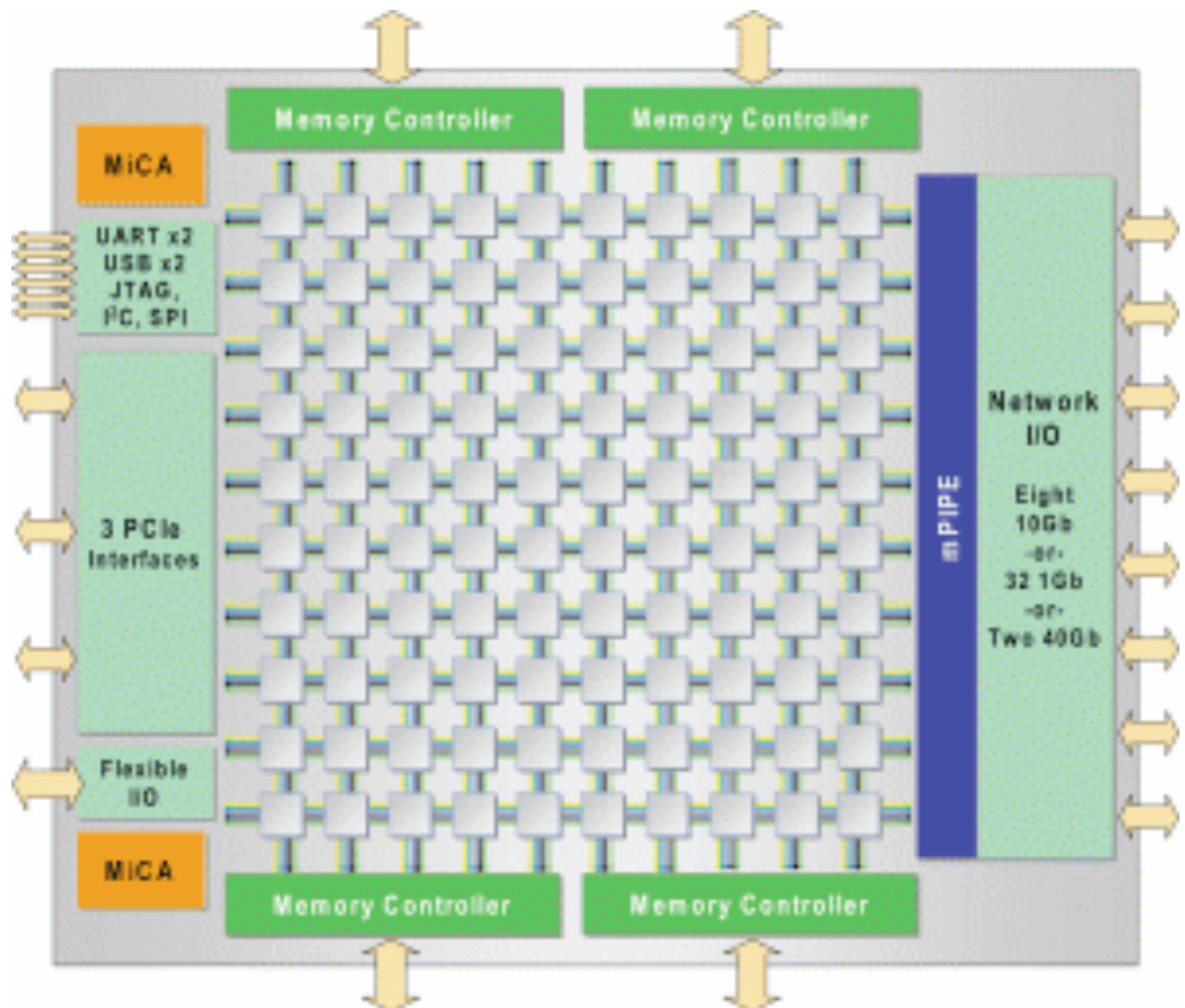
- Portfolio or search-space split approach (ManySAT, pLingeling,...)
- Works ok on clusters
- Confirmed thru' experimentation:
  - 12x speedup on a 128 node cluster
  - Not close to linear speedup



PSAT: Parallel SAT Approach

Partition SAT-Input into k Pieces

- Didn't work on clusters; much better prospects with multicore
- Latency much better on multicore than cluster
- Software engineering instances partition well
- Heuristics to minimize communication overhead



Imperative Language With SAT-based Declarative Primitives

- Motivation:
  - Declarative can be more robust
  - Delegating the "how" to runtime
- Combine imperative and SAT-based declarative language
  - Efficient solvers evaluate and search
  - Solvers leverage multicores
- Examples
  - Squander by Milicevic, Rayside and Daniel Jackson (MIT)

Related Work

- Model Checking:
  - Abstraction-refinement (Ed Clarke et al.)
  - Bounding (Ed Clarke, Daniel Jackson et al.)
- Other SMT solvers
  - Unsat core based approximations (Randy Bryant et al.)
  - Z3, CVC3, Boolector, BAT....
- DPLL(T)
  - Tinelli, Nieuwenhuis and Oliviera

Conclusions

- Logic formulas can capture meta-properties of software
  - The right logical abstraction (bit-vector and arrays, strings,...)
- Exploit meta-properties in solving formulas efficiently
  - Locality, modularity,...
- The more SMT solving, the less program analysis
  - Automation, ease-of-use,...

Questions?

Contributions at a Glance	Future Work
• STP* & HAMPI* (CAV 2007, TISSEC 2008, ISSTA 2009)	• Parallel SAT
• Decidability/Undecidability results for strings (under submission)	• SAT-based programming languages
• BuzzFuzz: Directed Whitebox Fuzzing (ICSE 2009)	• Program hardening
• Concolic testers (JFuzz: NFM 2009)	• Solvers for rich theories (attribute grammars, floating-point)
• Solvers for integer linear arithmetic (FMCAD 2002, TACAS 2003)	• Auto-tuning SAT solvers
• Retargetable compilers (DATE 1999)	• Advice-based SAT solvers
	• Unsound and incomplete solvers
	• Solver-based concurrency bug-finding

* 100+ research projects use STP and HAMPI (NSF funding $600,000.00)
* STP won the SMTCOMP 2006 and 2010 competitions for bit-vector solvers
* HAMPI paper won ACM Best Paper Award 2009