A Decision Procedure for Program Analysis and Bug Finding

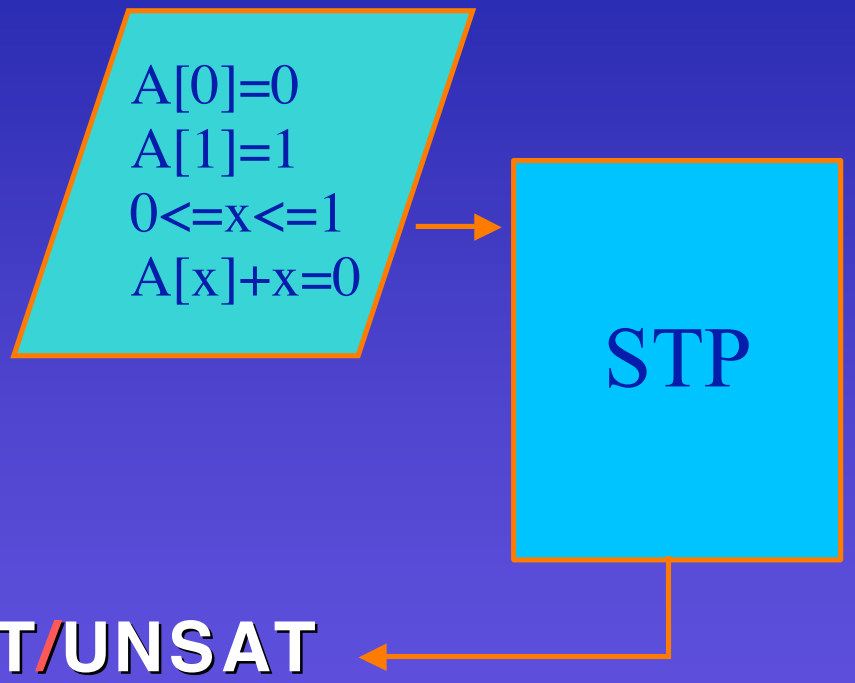
Vijay Ganesh
Affiliation: CSAIL, MIT
Supported by Lincoln Labs
February 7th, 2008

Motivating Example

```
Foo(int x){
  int A[2];
  int t;

  A[0] = 0;
  A[1] = 1;

  if(0 <= x <= 1) {
      t = 2/(A[x] + x);
  }
}
```



In Theory, Symbolic Execution + DP + Verification Conditions - Unbounded Loops, Gives Verification

Decision Procedures

- Examples: Boolean SAT, Real Arithmetic, Bit-vectors
- Reduction easy for many problems
- Approach better than coming up with special purpose algorithms:
  - More efficient and saves work
- AI, program analysis, bug finding, verification,...

STP

1. Design and Architecture of STP (CAV '07, CCS '06)
2. Abstraction-Refinement based heuristics for Deciding Arrays
3. Solver Algorithm for deciding Linear Bit-vector Arithmetic $O(n^3)$
4. Experimental Results

Projects using STP

- Bug Finders
  - EXE by Dawson Engler, Cristian Cadar and others (Stanford)
  - MINESWEEPER by Dawn Song and her group (CMU)
  - CATCHCONV by David Molnar and David Wagner (Berkeley)
  - Backward Path Sensitive Analysis by Tim Leek (MIT Lincoln)
- Security Tools
  - REPLAYER: Security analysis thru protocol replay (CMU)
  - Smart Fuzzer by Roberto Paleari (University of Milan, Italy)
- Program Analysis
  - by Rupak Majumdar (UCLA)
- Hardware verification
  - Cache coherence protocols by Dill group (Stanford)
  - By a chip company
- Software verification of crypto algorithms by Dill group (Stanford)

Projects using STP: Smart Fuzzing thru' Path Selection

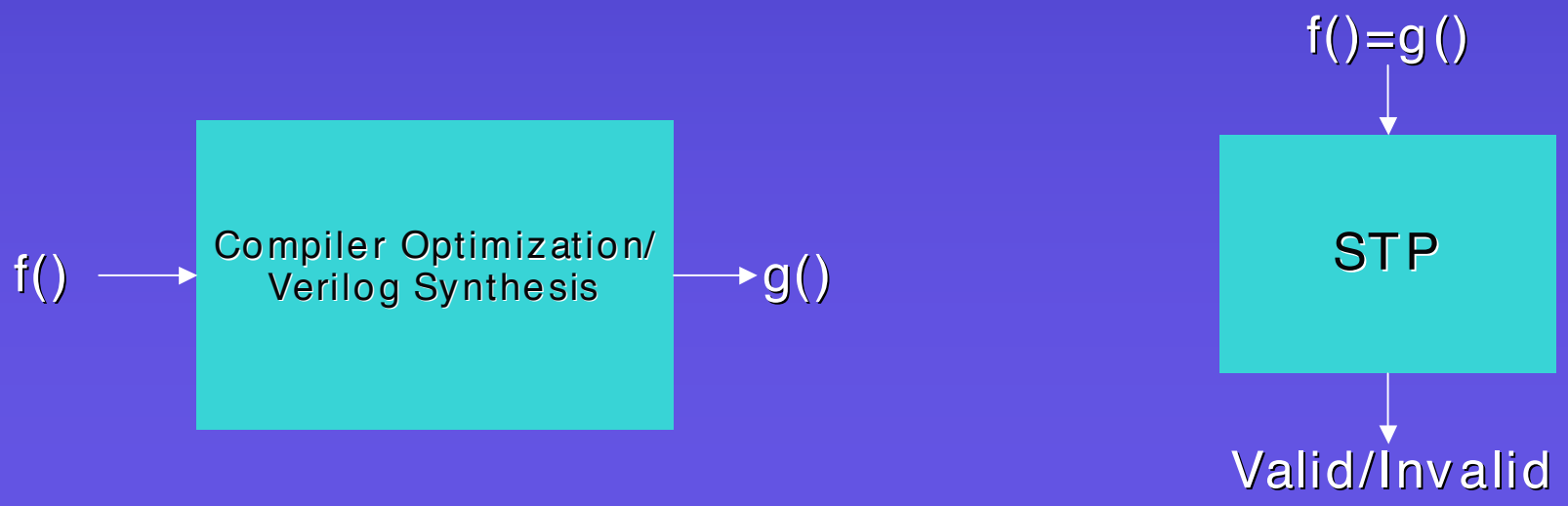
- Smart Fuzzer by Roberto Paleari (University of Milan, Italy)
  - Do dynamic analysis to determine dependency between input and control transfer (if conditional)
  - Collect path conditions
  - Feed to STP to find values that drive a path
  - Feed to STP to find values that drive the 'other' path

Projects using STP: Formal Verification of Crypto Algorithms

- Eric Smith and David Dill
  - Technique
    - Annotate code with Invariants
    - Symbolically execute the Java implementation of the Crypto Algo
    - Plug the symbolically executed terms into the invariants
    - Feed invariants into ACL2 + STP
    - ACL2 handles any induction + integer related stuff, and STP handles (in)equalities over bit-vector terms

Projects using STP:
Cross Checking, Model Checking, Equivalence Checking(?)

- Cross Checking: EXE : Dawson Engler, Cristian Cadar,...
  - Different implementations of grep... Do they match?
  - Symbolic-simulate Grep1
  - Symbolic-simulate Grep2
  - Equate the two and feed to STP
- Model Checking Cache Coherence Protocols: Chang and Dill
  - Does model satisfy property P?
  - Convert to decision problem and feed to STP
  - If you are using BDDs, try SAT or STP



Projects using STP:
Work by Dawn Song and her group

- Automatic discovery of deviations in binary implementations : error detection and fingerprint generation
- Protocol Replay: Try to reproduce a dialog between an initiator and a network host
  - Auto Generation of modules for honeypots so that they can correctly respond to connection attempts by worms
- Automatic patch based exploit generation: Using STP to reveal exploit information from a windows patch

Quantifier-free Theory of Bit-vectors and Arrays

(x + mem[i] + 0b10 = 0) OR (q[3:1]*0b01 < 0b00)

- Expressions in STP correspond to
  - C/Java… programming language expressions
  - Microprocessor instruction set
  - Arrays represent program memory or array data structure in C/Java…
- Except
  - Our bit-vectors are of any fixed length
  - No floating point
  - No loops
- SAT problem for this theory is NP-complete

Quantifier-free Theory of Bit-vectors and Arrays

(x + mem[i] + 0b10 = 0) OR (q[3:1]*0b01 < 0b00)

- Bit-vector Terms
  - Constants: 0b0011
  - Variables
  - +, -, *, (signed) div, (signed) mod
  - Concatenation, Extraction
  - Left/Right Shift, Sign-extend, bitwise-Booleans

Quantifier-free Theory of Bit-vectors and Arrays

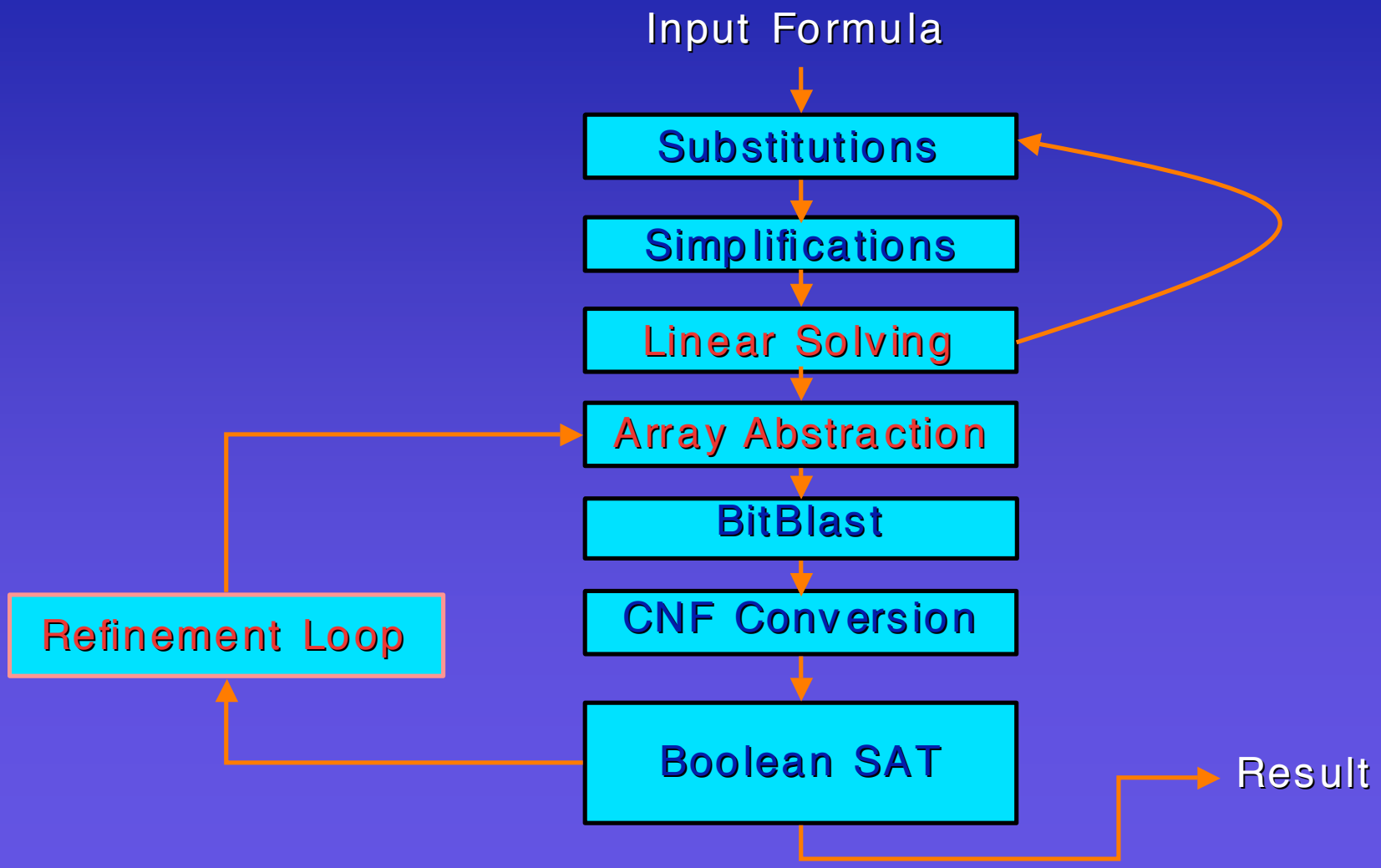
(x + mem[i] + 0b10 = 0) OR (q[3:1]*0b01 < 0b00)

- Array Terms
  - Read (Array, index)
  - Write (Array, index, val)
  - Example : R(W(A, i, 0b00), i) = 0b00
- Conditional in programming/multiplexors in hardware
  - ite (c, t1, t2) = if (c) then t1 else t2 endif
- Predicates: =, <=, <=s

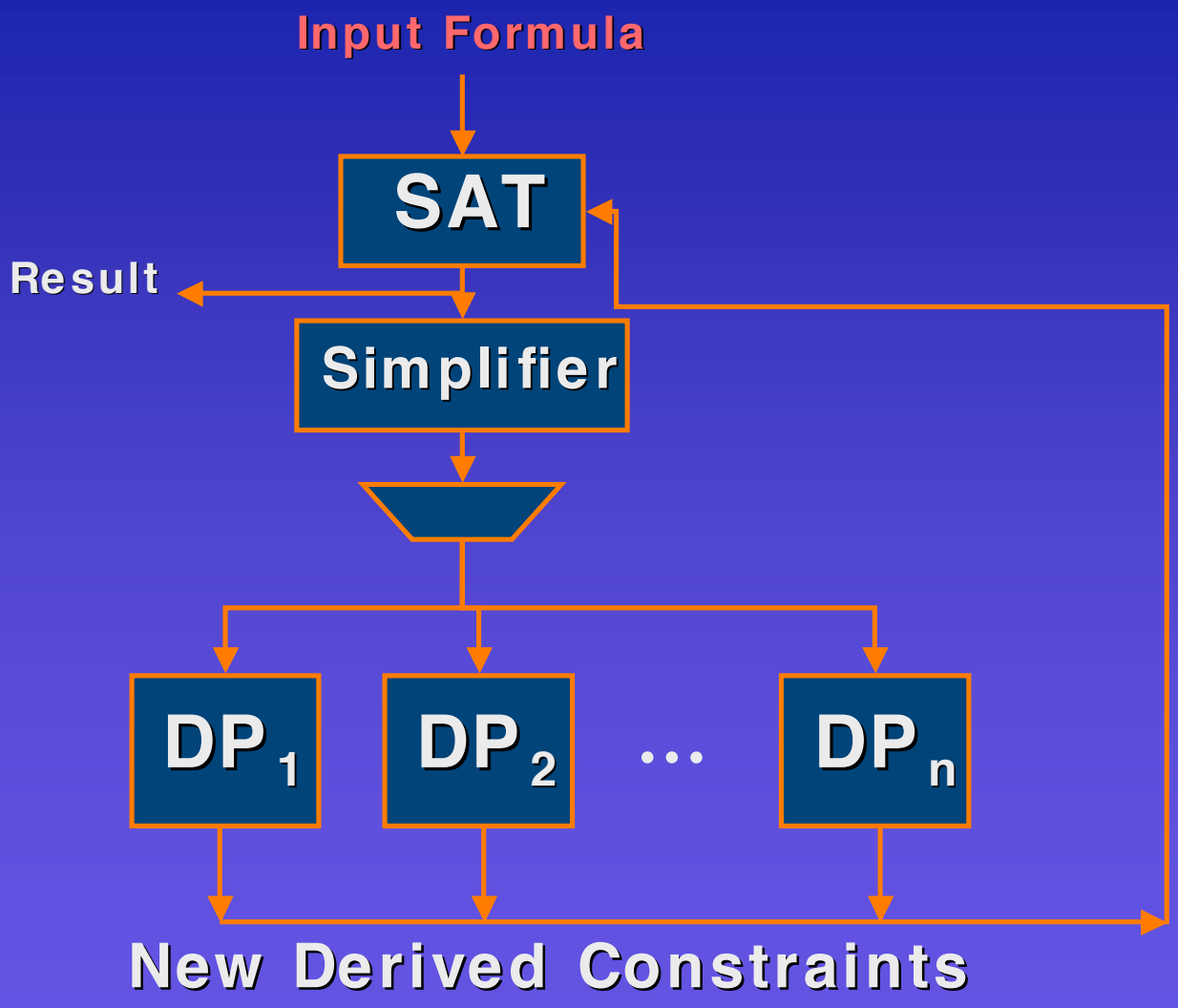
Features of STP

- Can handle very large formulas efficiently
  - Large number of array reads (10^5)
  - Deeply nested array writes (10^4 deep)
  - Very large number of linear equations (10^6)
  - Very large number of variables (10^6)
- **Enabled** several software and hardware applications
- Won the SMTCOMP 2006 competition in bit-vector category

STP Architecture



Alternative Architectures

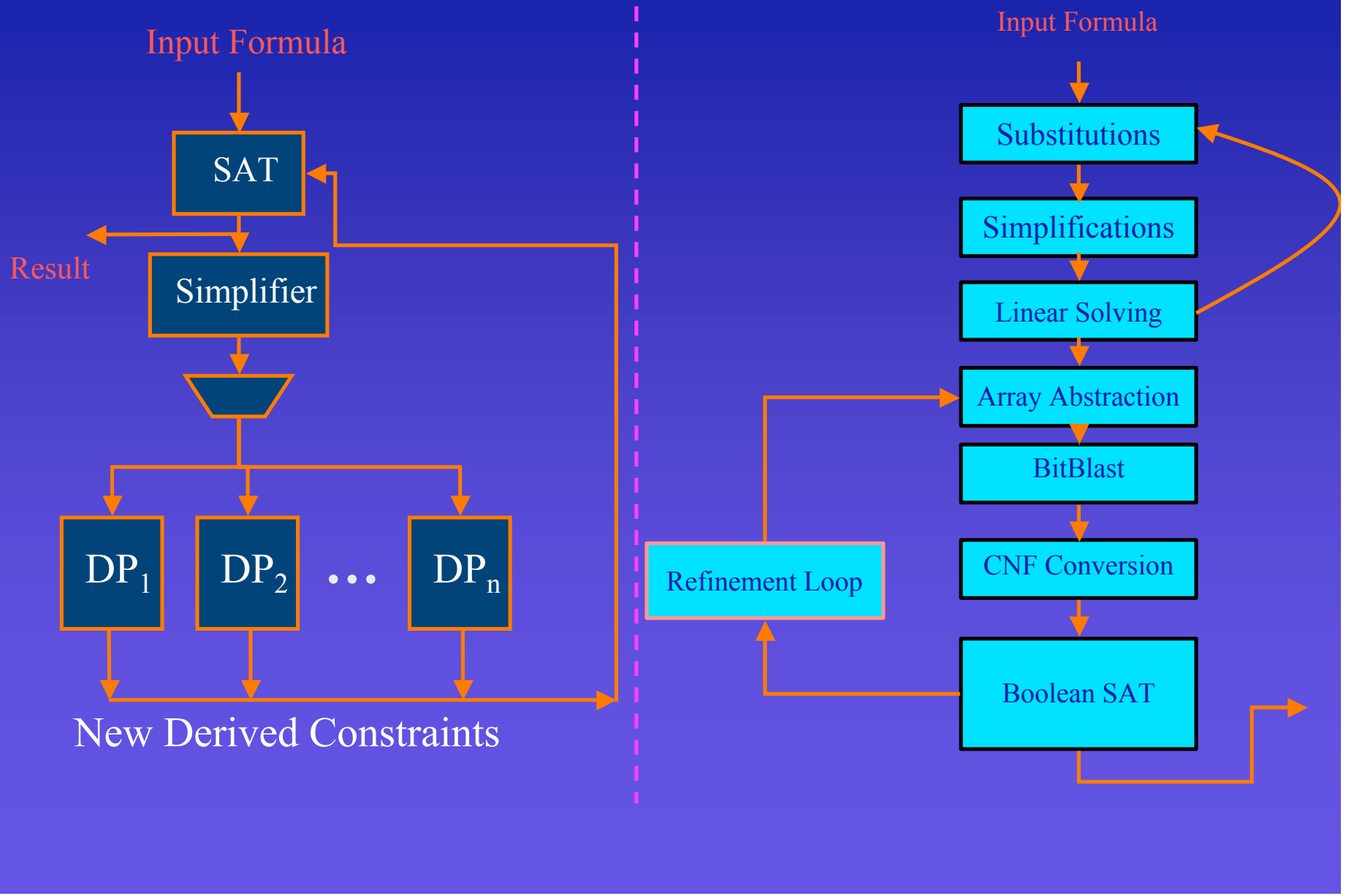


Combination:
NO79,Sho84,
RS02

CVC3 (BB04)
CVC (SBD02)
z3 (DeMB07)
Yices(DeMB05)

Others:
STP(GD06,GD07)
UCLID(BS05)
BAT(M06)
Cogent(BK05)...

Alternative Architectures



STP

1. Design and Architecture of STP
2. Abstraction-Refinement based heuristics for Deciding Arrays
3. Solver Algorithm for deciding Linear Bit-vector Arithmetic $O(n^3)$
4. Experimental Results

Standard Handling of Array reads

$Read(A,i_0) = t_0$
$Read(A,i_1) = t_1$
.
.
.
$Read(A,i_n) = t_n$

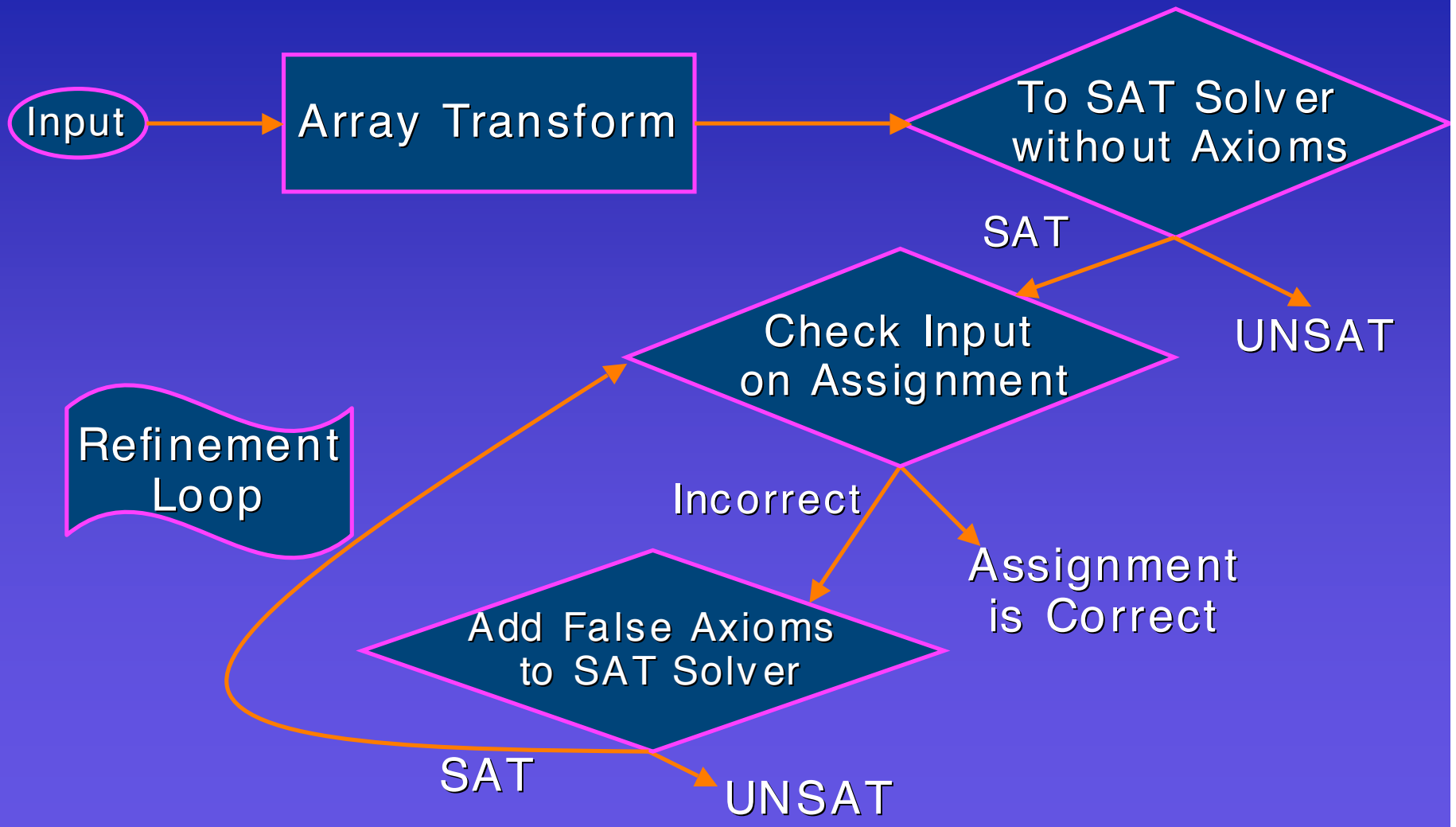
Replace array reads with fresh variables and axioms

$v_0 = t_0$
$v_1 = t_1$
.
.
$v_n = t_n$

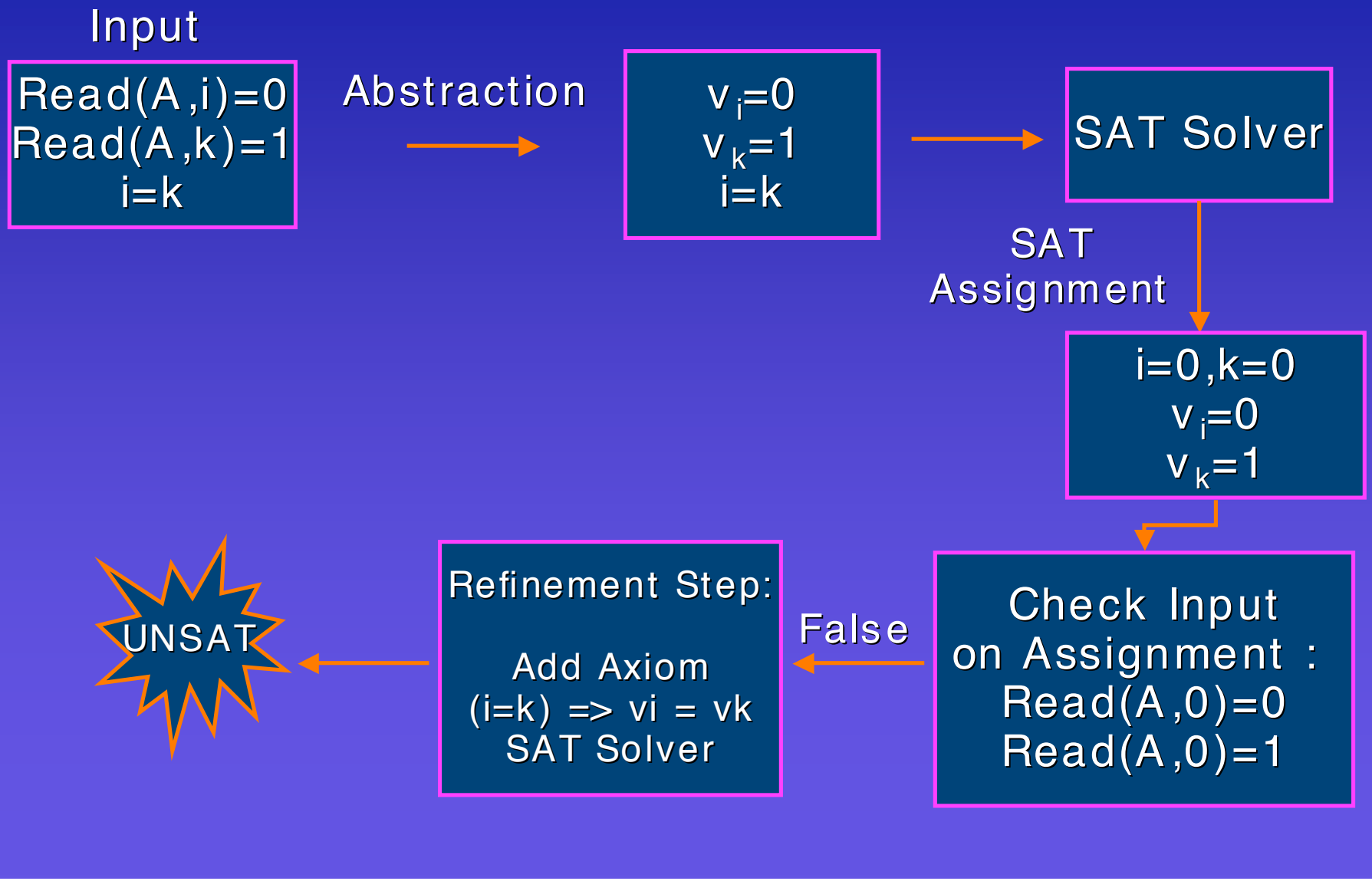
$(i_1=i_0) \Rightarrow v_1=v_0$
$(i_2=i_0) \Rightarrow v_2=v_0$
$(i_2=i_1) \Rightarrow v_2=v_1$
...

- Problem : $O(n^2)$ axioms added, n is number of read indices
  - Lethal, if n is large: n = 10000, # of axioms: ~ 100 million
  - Blowup seems hard to avoid (e.g. UCLID)
- This is "aliasing" from another perspective
- Key Observation: Most indices don't alias

Abstraction-Refinement for Array Reads



Abstraction-Refinement for Array Reads



Experience with Read Abstraction-Refinement

- Heuristic is Robust
  - In Real SAT assignment very few indices aliased
  - Few axioms need to be added during refinement
  - ~10X speed-up
  - Important for software analysis

# of Tests:8495	Only Read Refinement (sec)	No Read Refinement (sec)
Time for all tests	624	3378
# of timeouts (60 sec)	1	36

3.2 GHz Pentium, 512Kb Cache, 32 bit machine

Examples courtesy Dawson Engler

Standard Handling of Array Writes

$R(W(W(A,i_0,v_0),i_1,v_1),j)$
$=$
$R(W(W(A,i_0,v_0),i_1,v_1),k)$

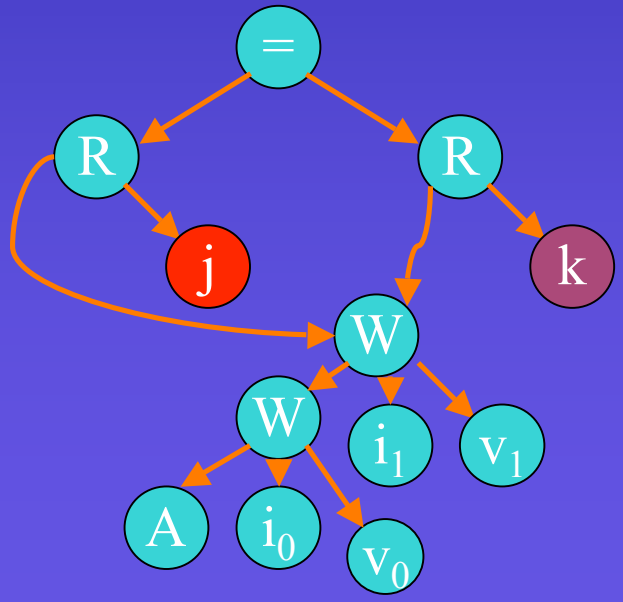
$If(i_1=j)\ v_1\ elsif\ (i_0=j)\ v_0\ else\ R(A,j)$
$=$
$If(i_1=k)\ v_1\ elsif\ (i_0=k)\ v_0\ else\ R(A,k)$

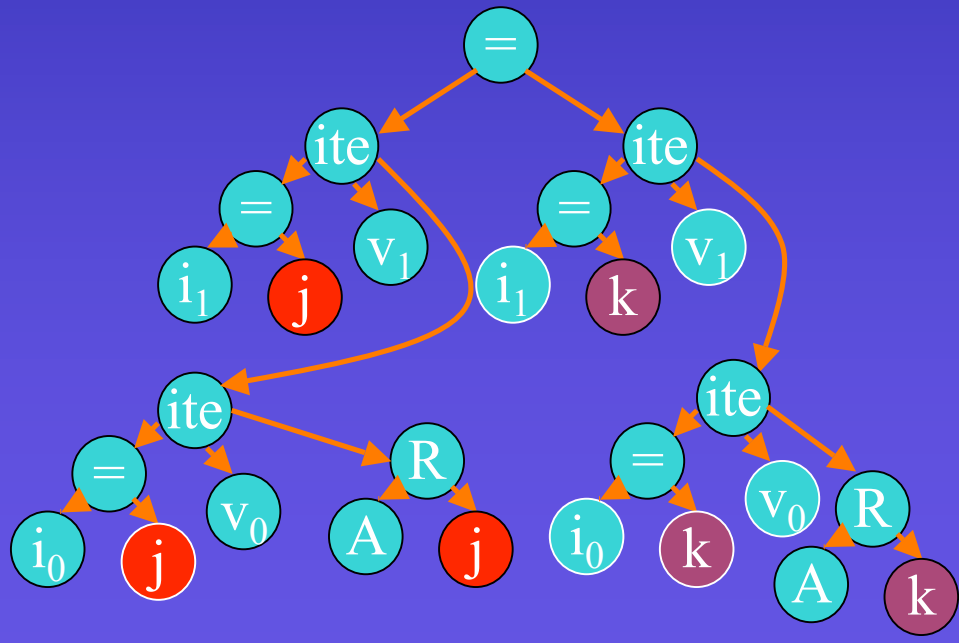
- Sharing of sub-expression in DAG
- Array Writes are deeply nested, shared over many reads
- Problem: Standard translation breaks sharing & blowup
  - O(n*m) blowup, n = # of levels of writes, m = # of reads
  - n = 10,000, m = 1000 : blow-up ~ 10 million new nodes
- Key Observation: Not all read indices read from write term

The Problem with Array Writes

$R(W(W(A,i_0,v_0),i_1,v_1),j)$
=
$R(W(W(A,i_0,v_0),i_1,v_1),k)$

$If(i_1=j)\ v_1\ elsif\ (i_0=j)\ v_0\ else\ R(A,j)$
=
$If(i_1=k)\ v_1\ elsif\ (i_0=k)\ v_0\ else\ R(A,k)$





Handling of Array Writes in STP

$R(W(W(A,i_0,v_0),i_1,v_1),j)$
$=$
$R(W(W(A,i_0,v_0),i_1,v_1),k)$

$t_1 = t_2$

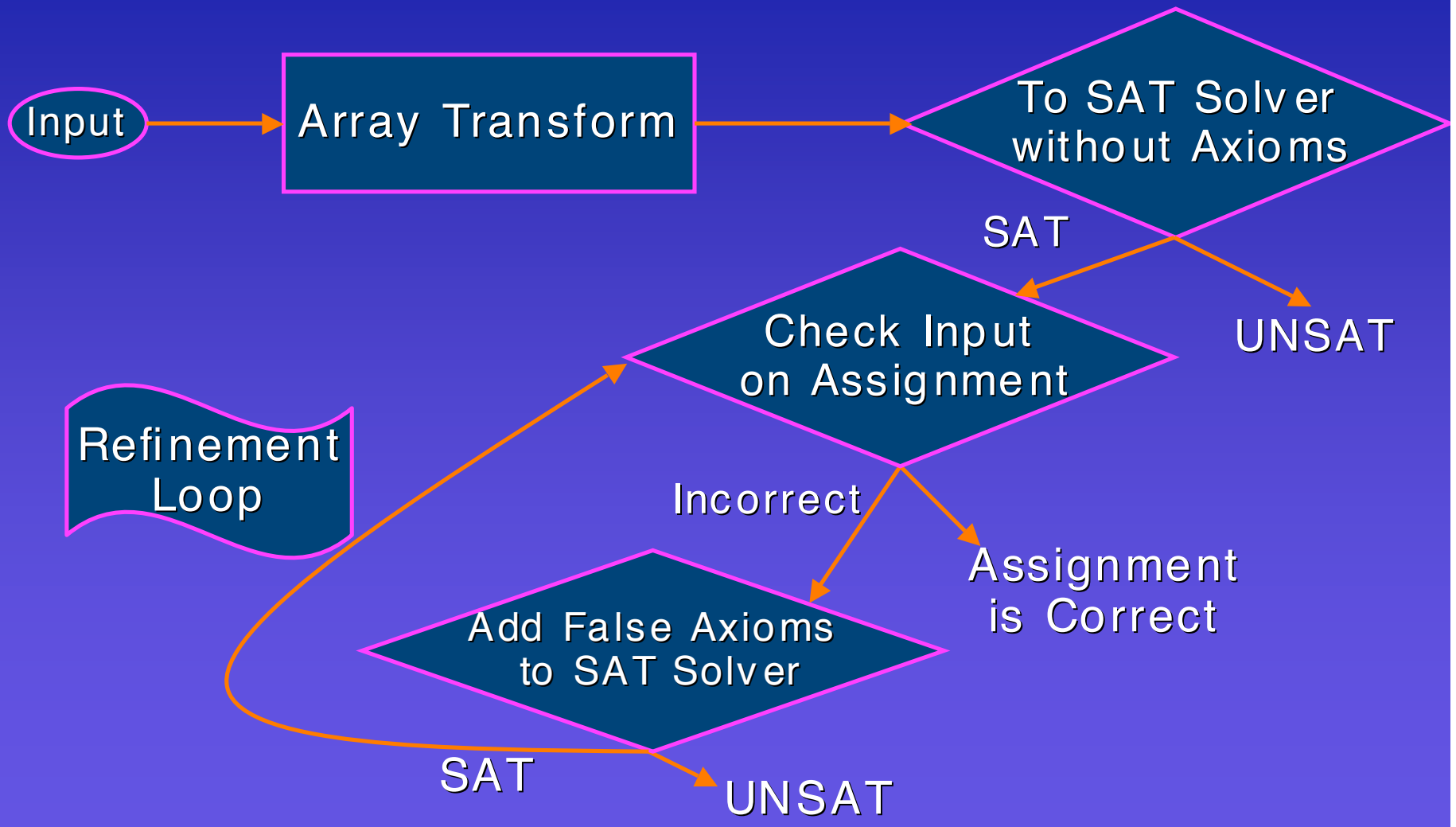
Axioms:

$t_1 = ite(i_1=j,v_1,ite(i_0=j,v_0,R(A,j))$

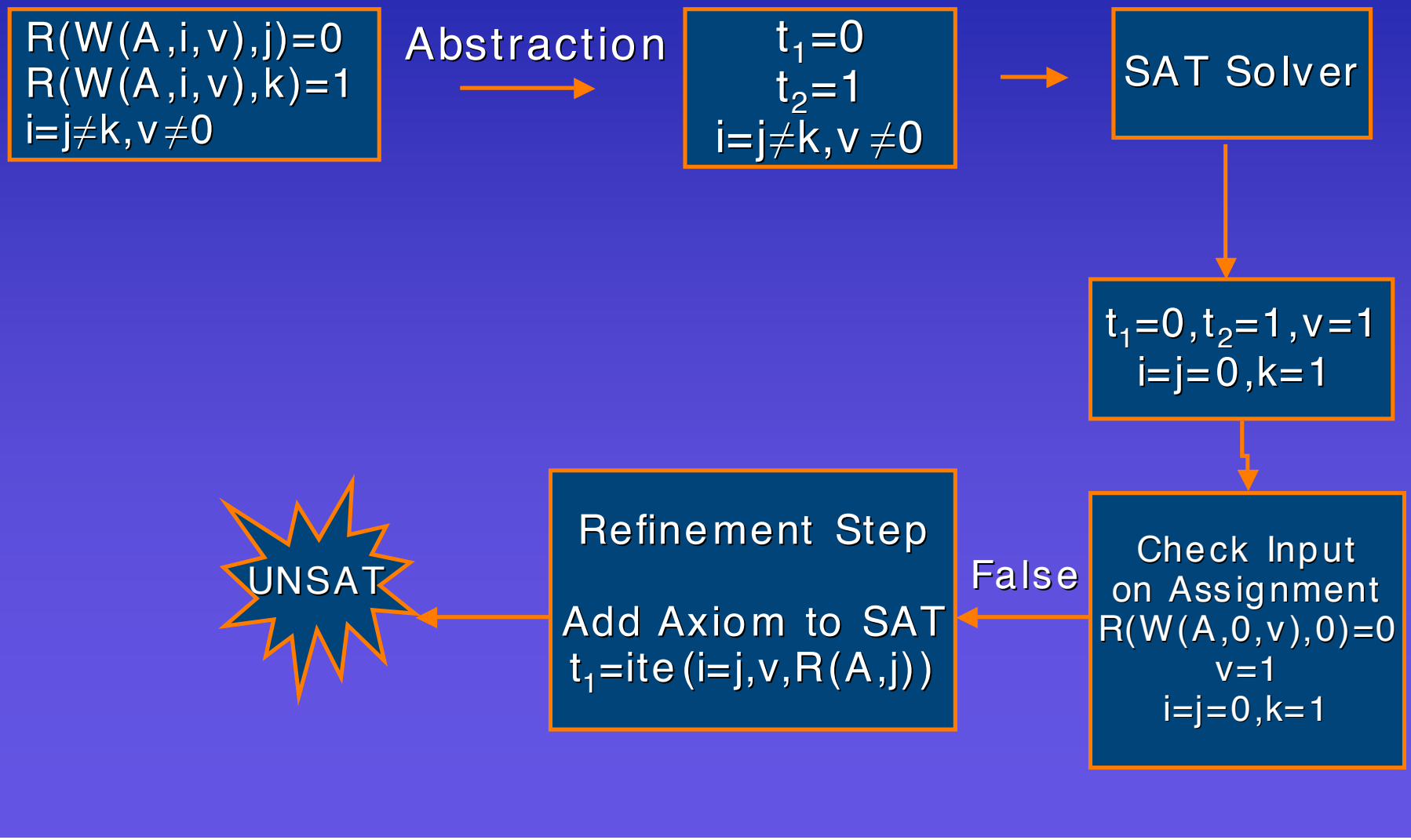
$t_2 = ite(i_1=k,v_1,ite(i_0=k,v_0,R(A,k))$

- Avoids $O(n^2)$ DAG blow-up
- Axioms are added only on a need basis
- Unfortunately, worst-case all axioms added

Abstraction-Refinement for Array Writes



Abstraction-Refinement for Array Writes



Experimental Results
Array Writes

Testcase (# of unique nodes)	Result	Write Abstraction (sec)	NO Write Abstraction (sec)
610dd9dc (15k)	Sat	37	101
Grep0084 (69K)	Sat	18	506
Grep0106 (69K)	Sat	227	TO
Grep0117 (70K)	Sat	258	TO
Testcase20 (1.2M)	Sat	56	MO

3.2 GHz Pentium, 512 Kb cache, 32 bit machine, MO @ 3.2 GB, TO @ 30 minutes

Examples courtesy Dawn Song (CMU) and David Molnar (Berkeley)

STP

1. Design and Architecture of STP
2. Abstraction-Refinement based heuristics for Deciding Arrays
3. Solver Algorithm for deciding Linear Bit-vector Arithmetic $O(n^3)$
4. Experimental Results

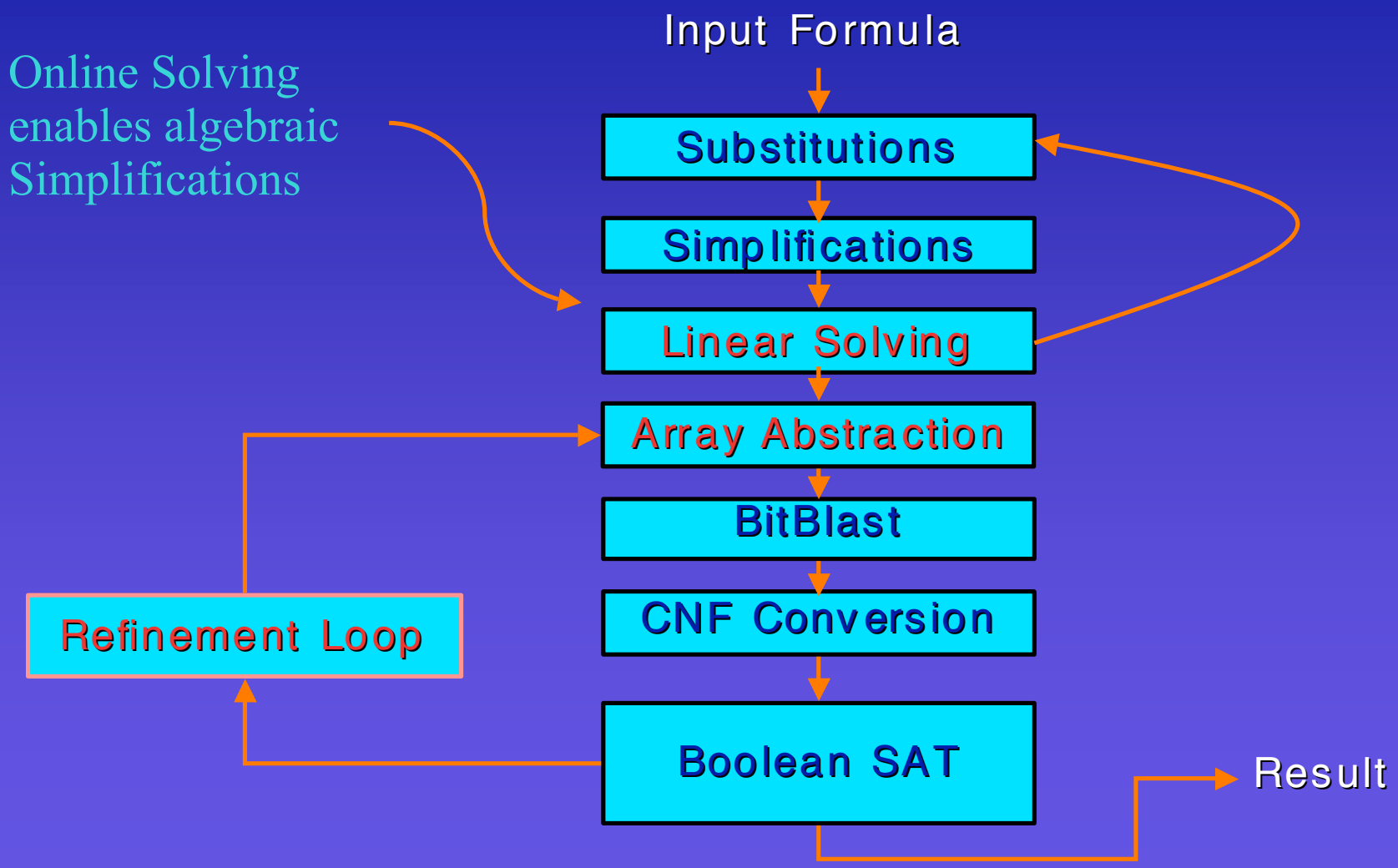
Algorithm for Solving Linear Bit-vector Equations

- Previous Work
  - Mostly Variants of Gaussian Elimination
    - Unsuitable for Online Decision Procedures
- Basic Idea in STP
  - Solve for a variable and substitute it away
  - Online Algorithm
    - Enables other algebraic simplifications
  - If cannot isolate a whole variable,
    - Then isolate part of bit-vector variable,
    - Solve, and substitute it away

Purpose of Linear Solver

- Helps eliminate lots of redundant variables
- Makes problem much easier for SAT
- Essential for many real-word large examples

Importance of Online Linear Solver



Algorithm for Solving Linear Bit-vector Equations

(mod 8)

$3x + 4y + 2z = 0$
$2x + 2y + 2 = 0$
$4y + 2x + 2z = 0$

→

Isolate 3x in first equation:

Multiplicative Inverse exists, Solve for x

↓

$x = 4y + 2z$

← Substitute x

(mod 8)

$2y + 4z + 2 = 0$
$4y + 6z = 0$

Algorithm for Solving Linear Bit-vector Equations

(mod 8)

$$2y + 4z + 2 = 0$$
$$4y + 6z = 0$$

→ All Coeffs Even

No Inverse

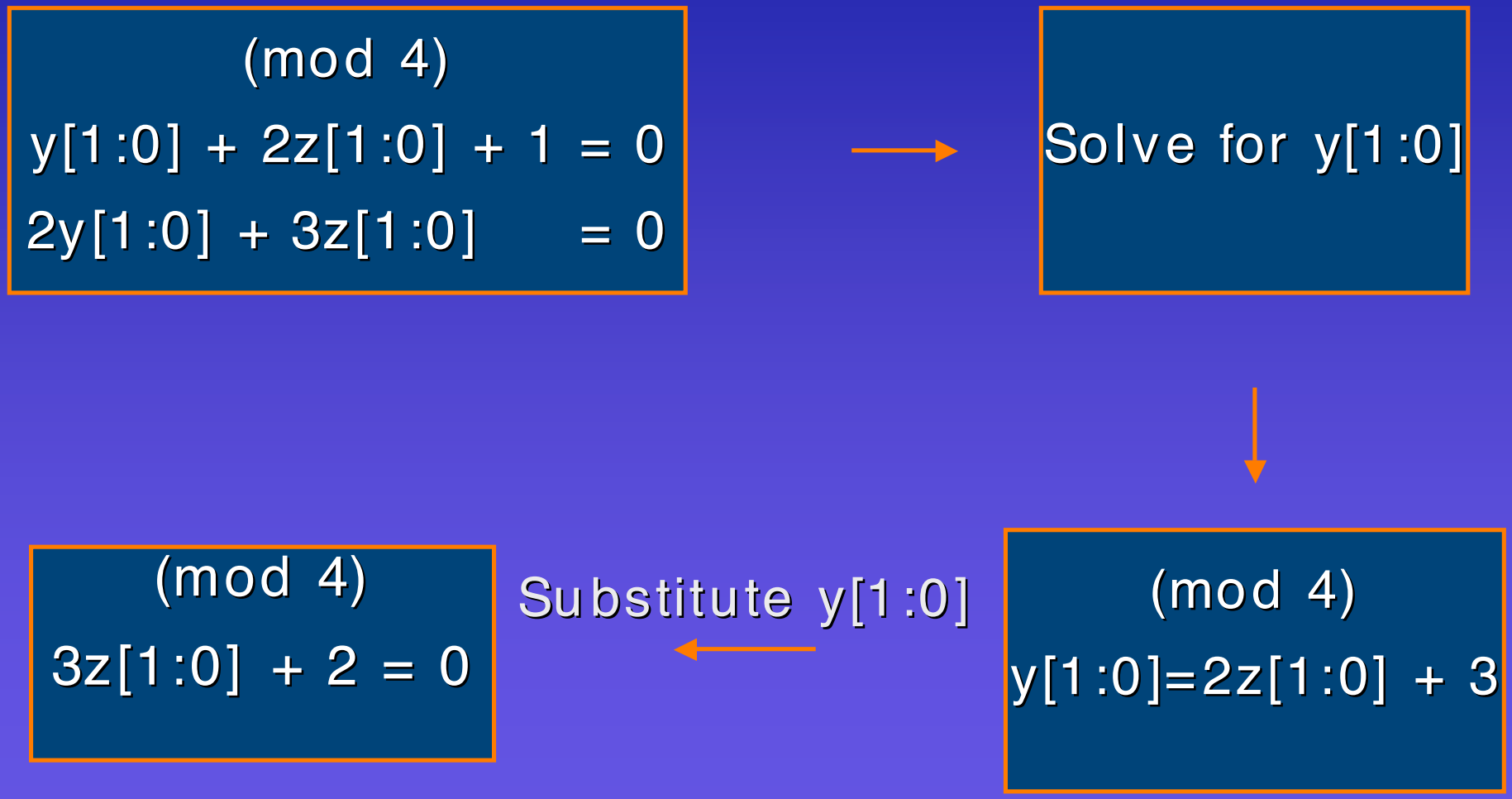
↓ Key Idea: Solve for bits of variables

Divide by 2 ←

(mod 4)

$$y[1:0] + 2z[1:0] + 1 = 0$$
$$2y[1:0] + 3z[1:0] = 0$$

Algorithm for Solving Linear Bit-vector Equations



Algorithm for Solving Linear Bit-vector Equations

(mod 4)

3z[1:0] + 2 = 0

Solve for z[1:0]

(mod 4)

z[1:0]=2

Solution (mod8, 3 bits)

x = 4(y'@3) + 2(z'@2)

y = y' @ 3

y[1:0] = 3

z = z' @ 2

z[1:0] = 2

Experimental Results: Solver for Linear Equations

Testcase (# of Unique Nodes)	Result	Solver On (sec)	Solver Off (sec)
Test15 (0.9M)	Sat	66	MO
Test16 (0.9M)	Sat	67	MO
Thumb1 (2.7M)	Sat	840	MO
Thumb2 (3.2M)	Sat	115	MO
Thumb3 (4.3M)	Sat	1920	MO

3.2 GHz Pentium, 512 Kb cache, 32 bit machine, MO @ 3.2 GB, TO @ 35 minutes

Examples courtesy David Molnar (Berkeley)

STP

1. Design and Architecture of STP
2. Abstraction-Refinement based heuristics for Deciding Arrays
3. Solver Algorithm for deciding Linear Bit-vector Arithmetic $O(n^3)$
4. Experimental Results

STP v. Existing Tools (Hardest Examples: SMT Comp, 2007)

Testcase (# of Unique Nodes)	Result	STP (sec)	Z3 (sec)	Yices (sec)
610dd9c (15k)	Sat	37	TO	MO
Grep65 (60k)	UnSat	4	0.3	TO
Grep84 (69k)	Sat	18	176	TO
Grep106 (69k)	Sat	227	130	TO
Blaster4 (262k)	Unsat	10	MO	MO
Testcase20 (1.2M)	Sat	56	MO	MO
Testcase21 (1.2M)	Sat	43	MO	MO

3.2 GHz Pentium, 512 Kb cache, 32 bit machine, MO @ 3.2 GB, TO @ 35 minutes

Examples courtesy Dawn Song (CMU) and David Molnar (Berkeley)

Lessons Learnt

- Abstraction Refinement will remain important for DPs for many applications
- Reduction to Boolean SAT
- Identify polynomial pieces and nail them
- Successful DPs highly application driven

Future Work

- Make STP more efficient for
  - Disjunctions
  - Non-linear Arithmetic (*, /, %)
- Quantifiers
- Boolean SAT tuning for structured input
- More theories
  - Uninterpreted Functions, Datatypes, Reals, Integers, ...

Other Projects at Stanford

- Software
  - CVC
    - Decision Procedure for Mixed Real and Integer Linear Arithmetic
  - CVC Lite
    - Decision Procedure for Bit-vectors
  - Collaborated on EXE
    - STP, Capturing C semantics in STP
- Theory
  - Lifted Ghilardi's Combination Result to Many-Sorted Logic

Acknowledgements

- Prof. David L. Dill, Stanford CS Department (Ph.D. Advisor)
- STP users and Stanford community
- Prof. Martin Rinard (Host)
- Lincoln Labs and Tim Leek (Support)

QUESTIONS

http://people.csail.mit.edu/vganesh/stp.html