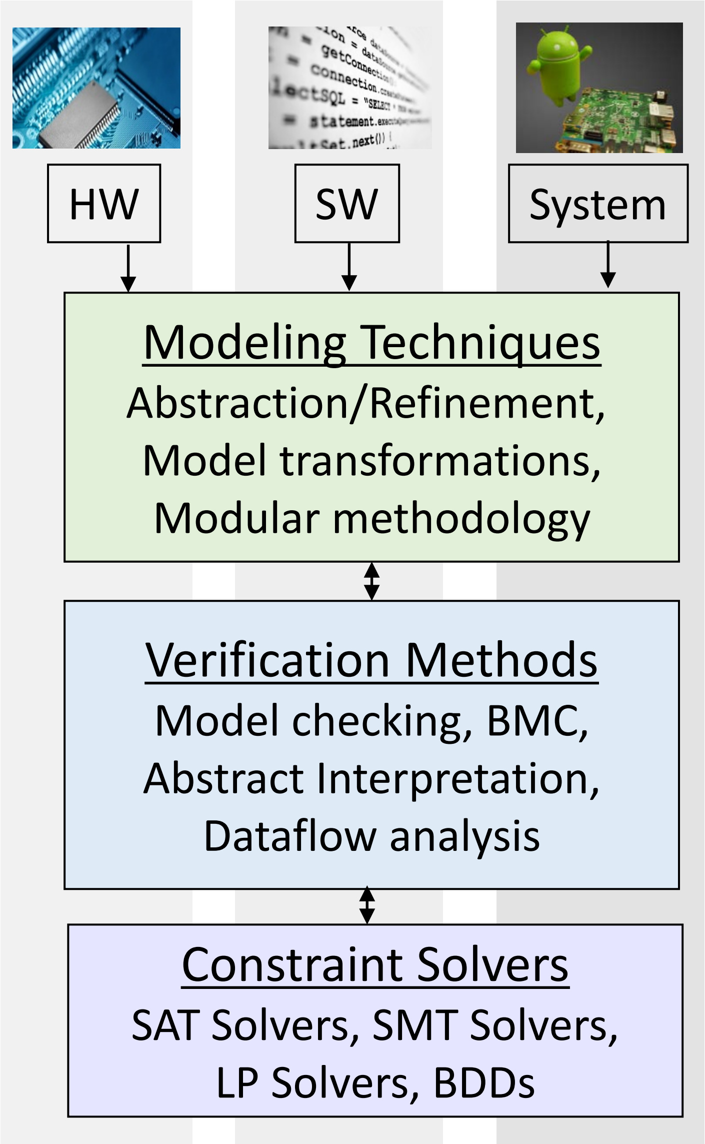
Software Model Checking: Locks and MLoCs

Aarti Gupta

Ph.D. from CMU SCS – Thanks Ed!

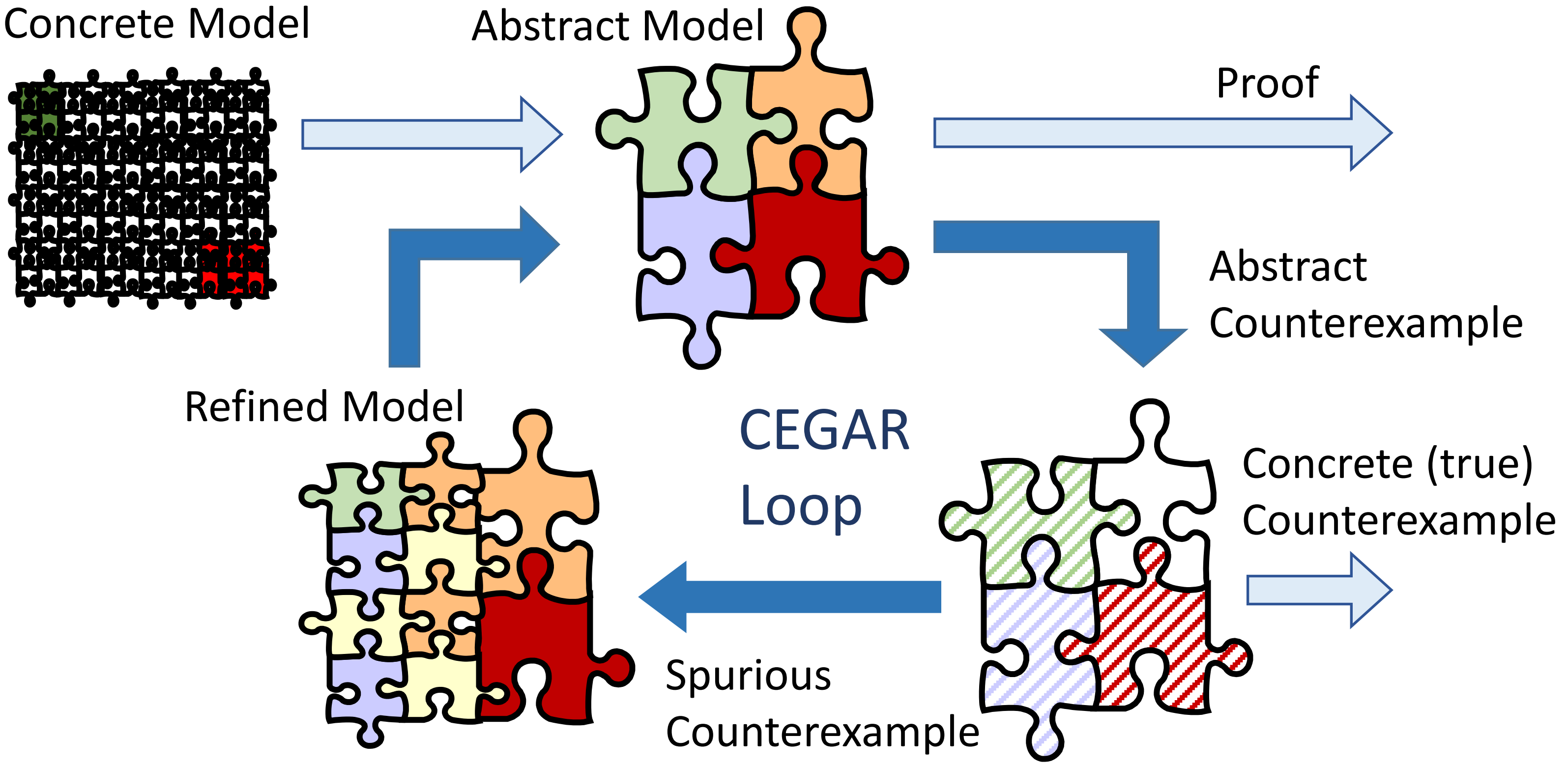
Acknowledgements: NEC Labs America
Gogul Balakrishnan, Malay Ganai, Franjo Ivančić, Vineet Kahlon, Naoto Maeda, Nadia Papakonstantinou, Sriram Sankaranarayanan, Chao Wang, Varvel group in NEC Japan

Verification Research at NEC Labs



Layered approach
- Constraint solvers
- Verification and analysis methods
- Modeling techniques

Application domains (so far)
- Software programs, multi-threaded programs
- Hardware/embedded/hybrid systems



API Usage Bugs

SLAM	[Ball & Rajamani 01]
Blast	[Henzinger *et al.* 02]
SatAbs	[Clarke *et al.* 04]

F-Soft-CEGAR [JIGG 05]

Does not scale for finding memory-safety bugs

- *null pointer derefs*
- *array buffer overflows*
- *string usage bugs*
- *uninitialized variables*

If concrete model is missing alias information
CEGAR loop makes no progress

Precision

Number of alias predicates blows up
Harder to get proof

Scalability

Precise memory & pointer models

[AGGI+ 04, ISGG+ 05]

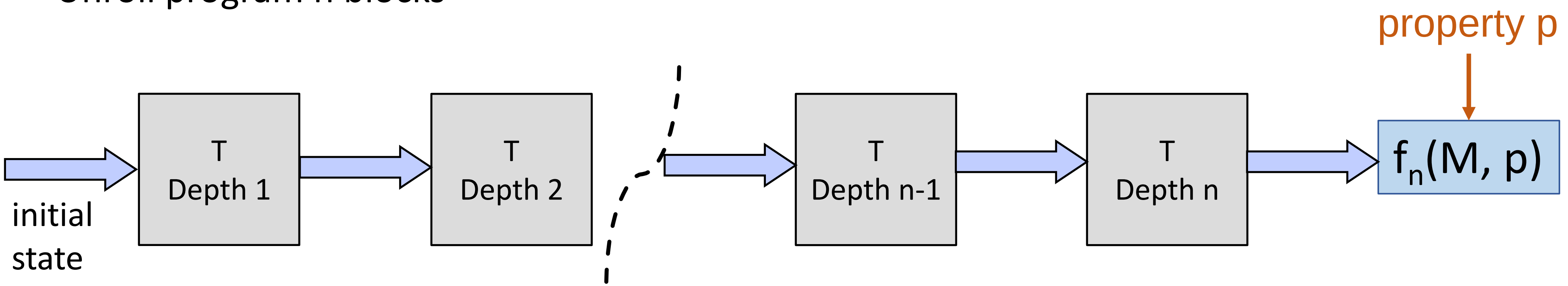
Scalability: Finding Bugs using Search

Bounded Model Checking (BMC) [Biere *et al.* 99]

Unroll transition relation T to depth n

Software Bounded Model Checking [Clarke *et al.* 04 (CBMC), AGGIY 04]

Unroll program n blocks

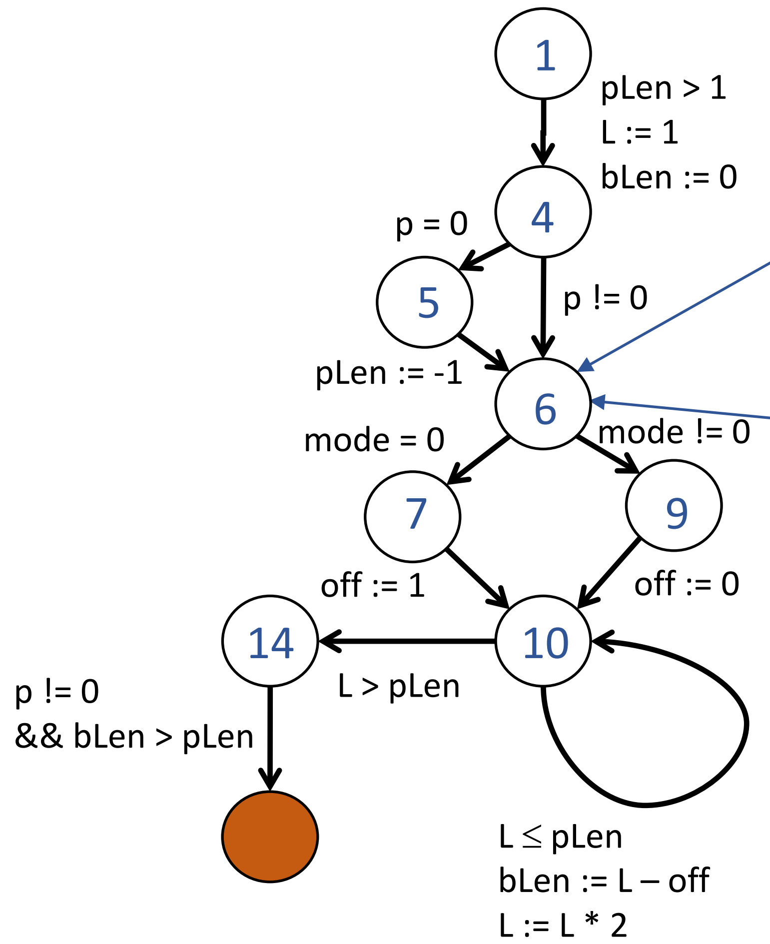


Satisfiability of $f_n(M, p) \equiv$ Property violation at depth n

SAT solver searches space *relevant* to property p
State sets are not saved

Critical for scalability

Finding Proofs: Scalability and Precision



Predicate abstractions: SLAM, Blast, SatAbs

$((p=0) \wedge (pLen = -1)) \vee$
$((p!=0) \wedge (pLen > 1))$

allows disjunctions
scalability challenge
path-sensitive – has precision

Numeric abstract domains: Astrée [Cousot&Cousot 77, Blanchet+ 03]

$(pLen \geq -1)$

no disjunctions (generally)
scales well
path-insensitive – loses precision at merge

Precision: Path-Sensitive Analysis

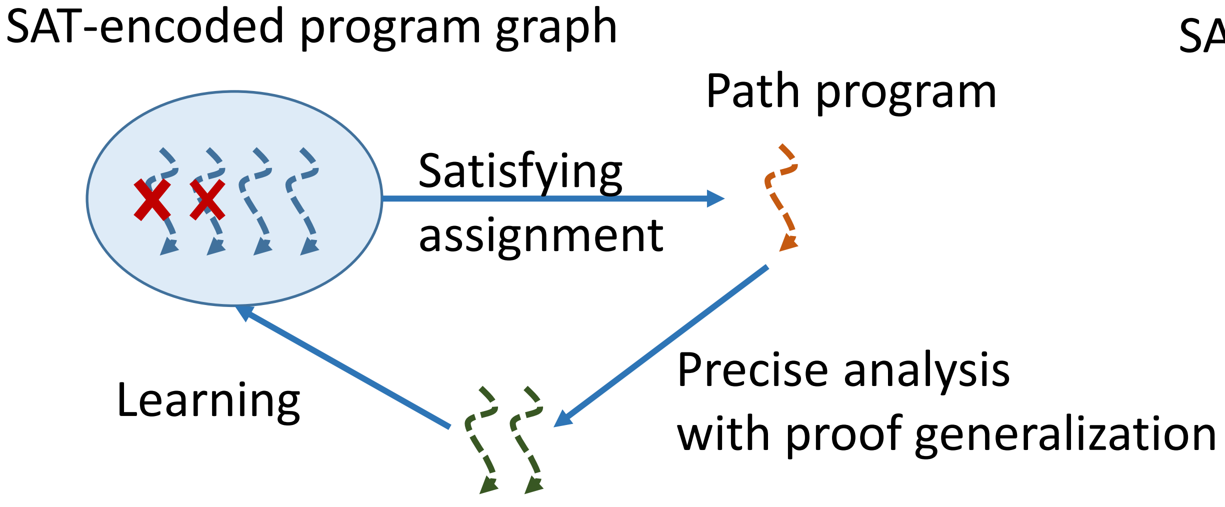
Takes branch conditions into account
May not get proof otherwise

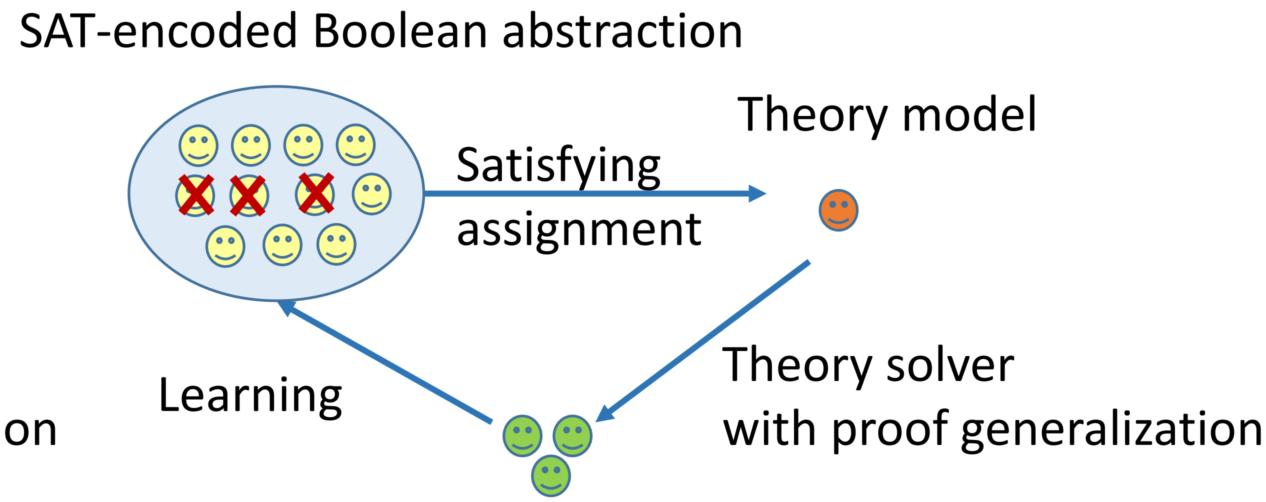
[HSIG 10]

Q: Scalability + Path-sensitivity?

A: Lazy path sensitivity

Balancing Precision and Scalability





Scalability *Precision*

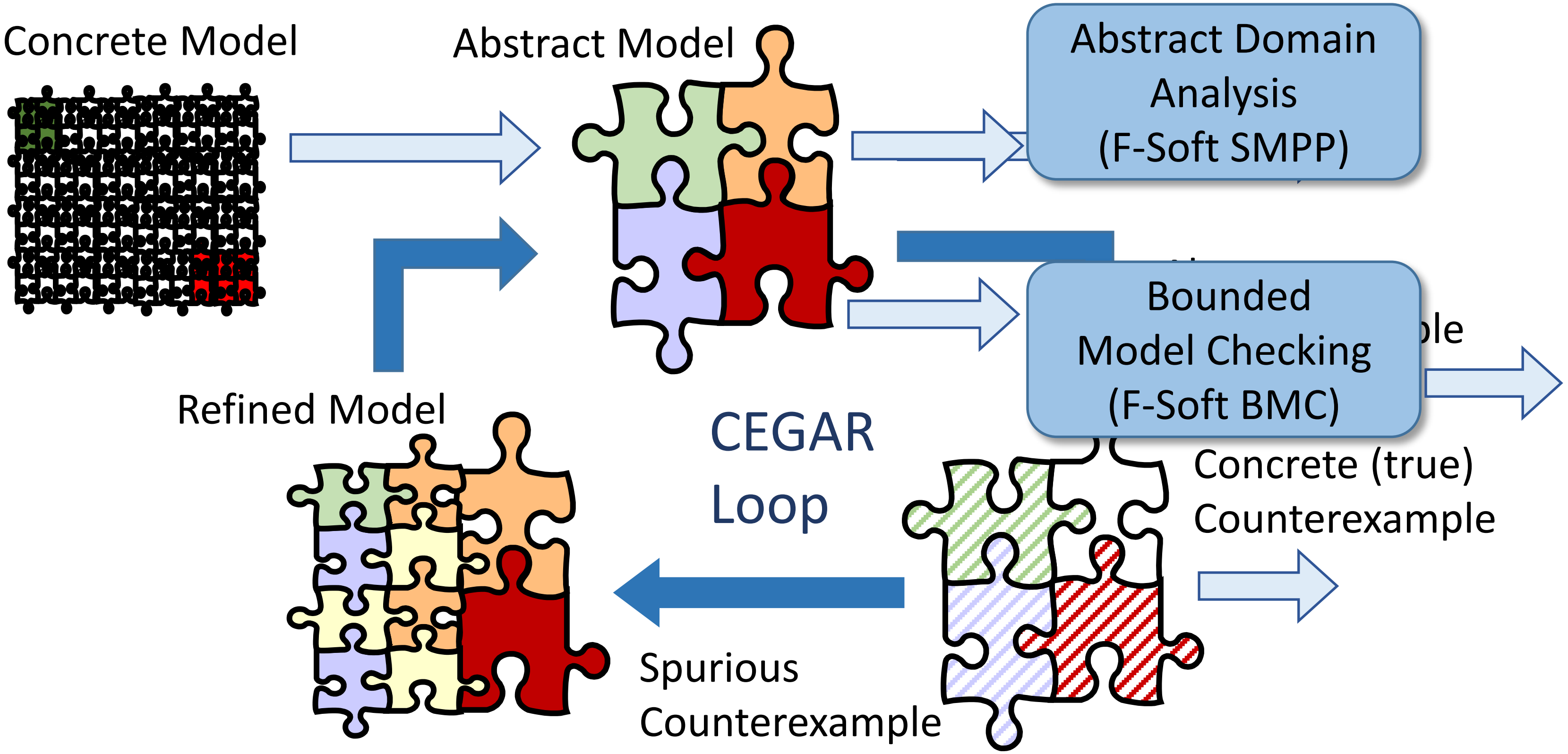
Satisfiability Modulo Path Programs (SMPP)

[HSIG 10]

Satisfiability Modulo Theories (SMT)

[Ganzinger *et al. 04*, Barrett *et al.* 09]

F-Soft Verifier



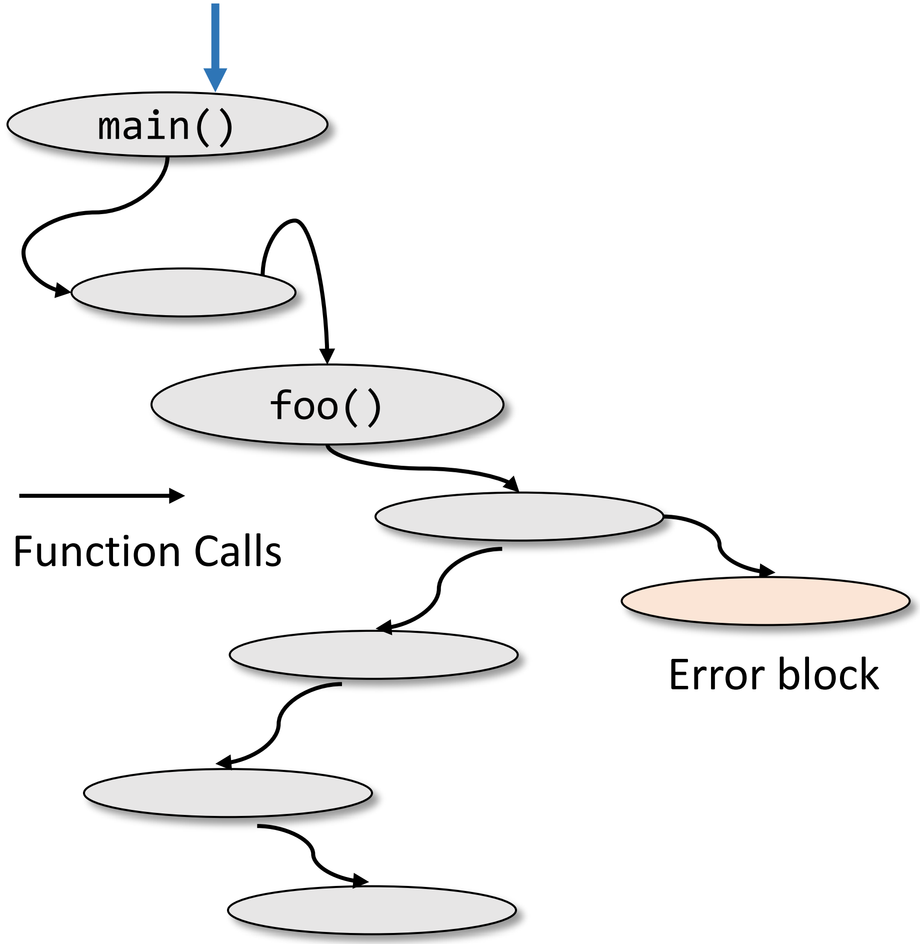
CEGAR loop makes no progress

Number of predicates blows up

Precision

Scalability

In Practice



Bugs can be deep from `main()`

Challenges

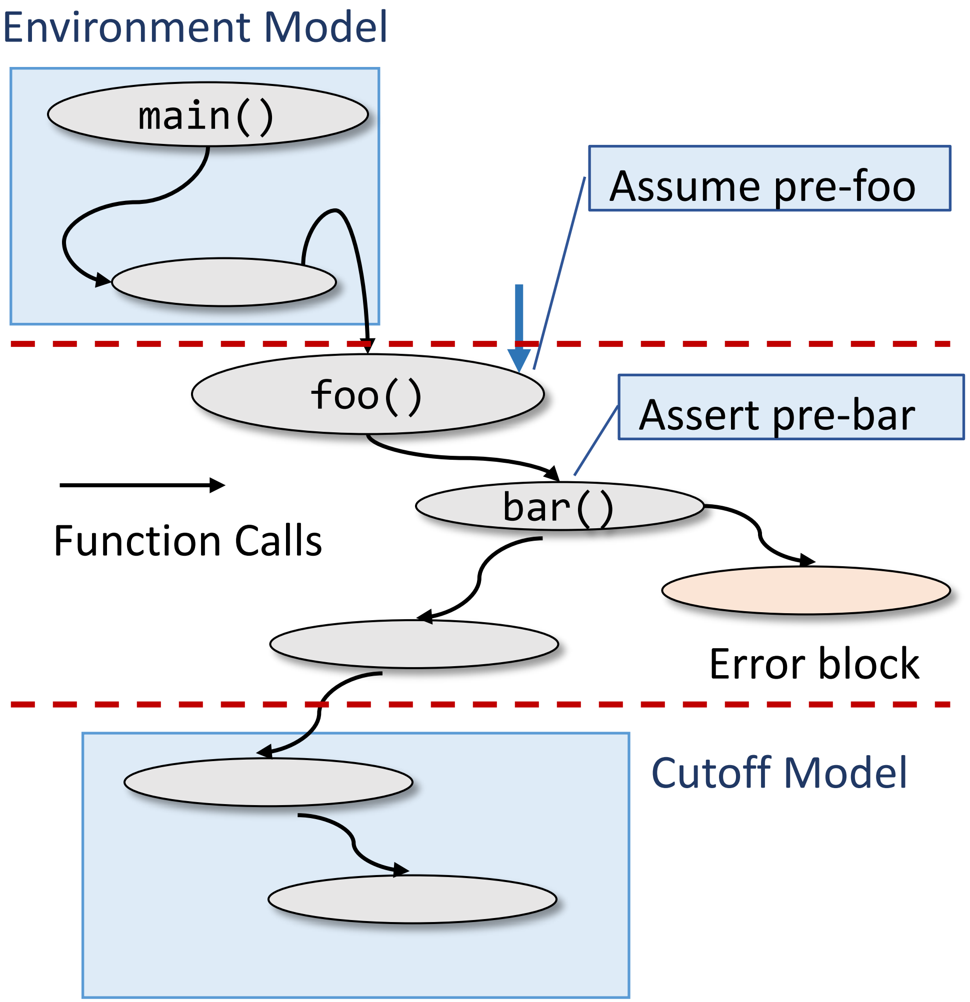
- Verifier runs out of time/memory
- Missing code for functions (libraries)
- Code with deep recursion (e.g. parsers)

Strategy

- Start from an intermediate function `foo()`

- Issue: How to supply the environment for `foo()`?

In Practice



From top

- Start from an intermediate function `foo()`
- Approximate environment model

From bottom

- Depth cutoff for bounding scope
- Approximate cutoff model

Modeling Strategy

- Light-weight static analysis
  - infers *likely* pre- and post-conditions, stubs

Depth Cutoff with Design Constraints [IBGS+ 11]

Modular assume-guarantee verification

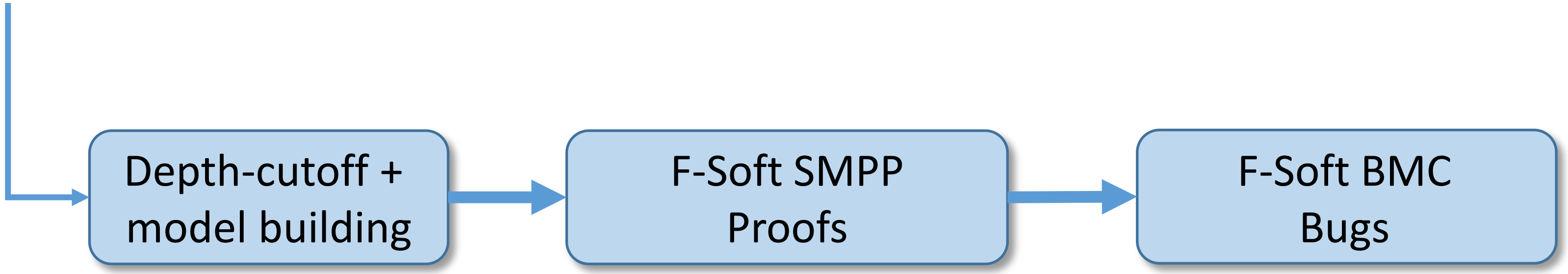
- links multiple levels in call-graph

Staging the Analyses

Build-analyzer (works on makefiles): MLoC C/C++

↳ Design constraint inference: compilable units, 100s KLoC, ~10 min (1 hr timeout)

↳ foreach-entry-function: 10s KLoC (checked in parallel)



~1 min

10 min timeout
discharges ~80% properties

2 min timeout
Bugs post-processed
~40% true bugs

false bugs mainly due to calling environment

In-house NEC Product: Varvel

Software Factory: since Nov '10

In 2013, Varvel applied on 65 projects, total: 40.5 MLoC, size: 1K to 20 MLoCs

Concurrent Programs: Additional Challenges

```
void Alloc_Page( ){
   pt_lock(&plk);
   if (pg_count  >= LIMIT) {
       pt_wait(&pg_lim, &plk);
       incr(pg_count);
       pt_unlock(&plk);
       a = sh;
   } else {
       pt_lock(&count_lock);
       pt_unlock(&plk);
       page = alloc_page();
       sh = 5;
       if (page)
           incr(pg_count);
       pt_unlock(&count_lock);
    end-if
  }
}
```

||

```
void Dealloc_Page( ){
   pt_lock(&plk);
   if (pg_count  == LIMIT) {
       sh = 2;
       decr(pg_count);
       b = sh;
       pt_notify(&pg_lim, &plk);
       pt_unlock(&plk);
   } else {
       pt_lock(&count_lock);
       pt_unlock(&plk);
       decr(pg_count);
       sh = 4;
       pt_unlock(&count_lock);
    end-if
  }
}
```

shared variables

synchronizations

interleavings

Data Race Detection: Staging the Analyses

Precise pointer analysis

Shared variable detection
Lockset computation

Lockset analysis

~1 minute
99 warnings

Lock history analysis

Invariant generation

+6 minutes
24 warnings

Bounded model checking

+11 hours
21 bugs

9 Linux device drivers
0.6 KLoC – 26.1 KLoC

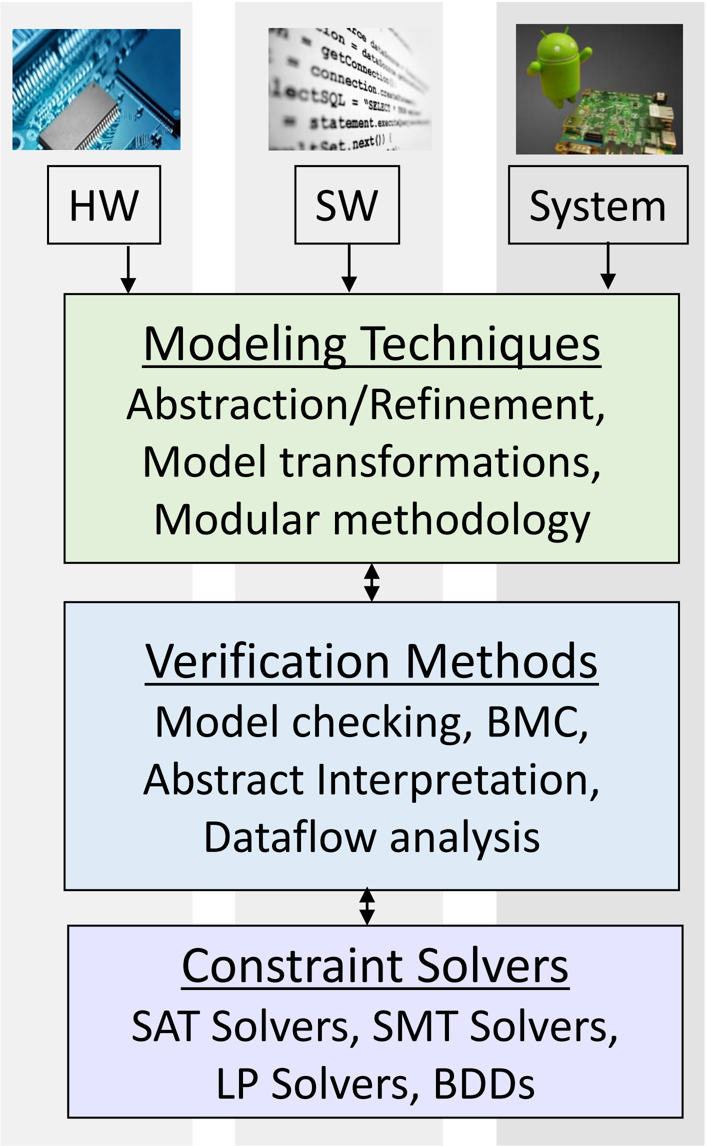
[Kahlon 08]
[KYSG 07]
[KIG 05, KG 07]
[KSG 09, KSG13]
[KGS 06]

CoBe: <u>Co</u>ncurrency <u>Be</u>nch

Found ~25 critical data race bugs in 5 industry projects, 9 – 379 KLoC

Soon to be deployed in NEC's Software Factory

Research Framework



Layered approach

- Constraint solvers
- Verification and analysis methods
- Modeling techniques

Application domains (so far)

- Software programs, multi-threaded programs
- Hardware/embedded/hybrid systems

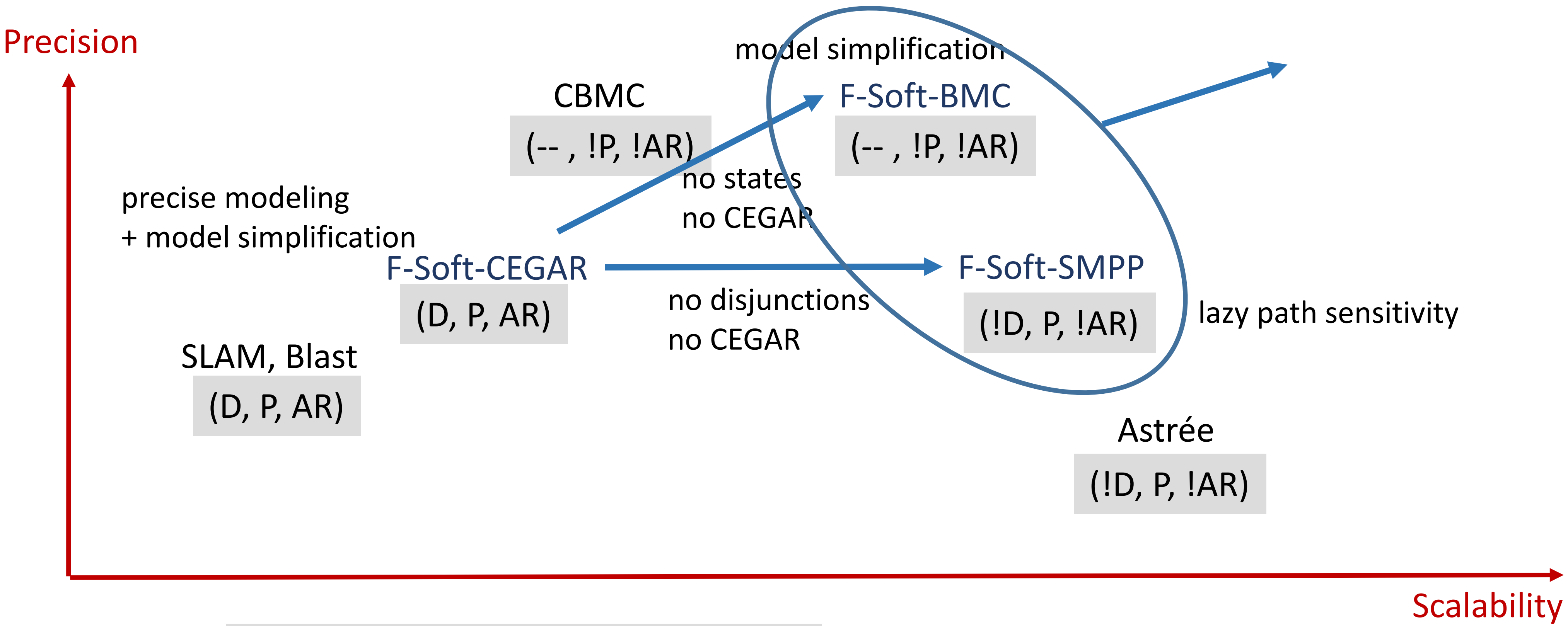
Future domains of interest

- *Distributed systems (Networks, Mobile, Cloud)*
- *Cyber-physical systems*
- *Biological systems*

Beyond verification applications

- *Synthesis, security, reliability*

Precision–Scalability Space



Verifier Design Dimensions: (D, P, AR)

D = disjunctive state sets
!D = conjunctive state sets

P = proofs
!P = bugs only

AR = abstraction-refinement
!AR = no refinement