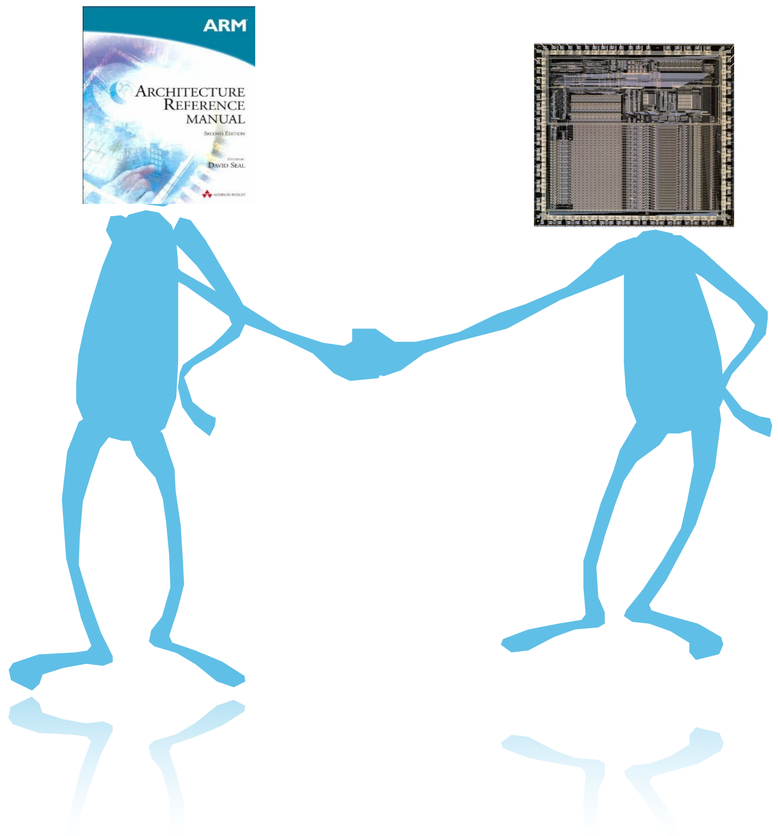
How can you trust formally verified software?



Alastair Reid

alastair.reid@arm.com

@alastair_d_reid

COMPCERT

COMPILERS YOU CAN *FORMALLY* TRUST

[…] By applying program proof techniques to the source code of the compiler, we can prove, with mathematical certainty, that the executable code produced by the compiler behaves exactly as specified by the semantics of the source~C~program, **therefore ruling out all risks of miscompilation.**

http://compcert.inria.fr/motivations.html

Microsoft research project IronFleet advances bug-free software systems

[…] recent advances have made it possible to write smaller-scale software that can be **mathematically proven not to have the type of imperfections that make a program freeze up or leave it vulnerable to a security attack.**

https://www.microsoft.com/en-us/research/blog/microsoft-researchers-explore-a-practical-way-to-build-bug-free-software/

seL4

Security. Performance. Proof.

seL4 is unique: it is the only operating system that has undergone formal verification, **proving bug-free implementation**, and enforcement of spatial isolation (data confidentiality and integrity).

https://sel4.systems/Info/Docs/seL4-brochure.pdf

```
Release 3.0, 2017-02-10
=======================

[…]

Bug fixing:

- Issue #155: on ARM, assembly errors caused by large jump tables for
  "switch" statements and overflow in accessing constant pools.
- Issue #151: large inductive definition causes a fatal error in
  32-bit versions of Coq.
- Issue #143: handle "%lf" printf() format in the reference interpreter
- Issue #138: struct declarations in K&R function parameters were ignored.
- Issues #110, #111, #113, #114, #115, #119, #120, #121, #122, #123, #124,
  #125, #126, #127, #128, #129, #130, #133, #138, #144: various cases
  of internal errors and failed assertions that should have been
  proper errors instead.
- For __builtin_memcpy_aligned, size and alignment arguments of 64-bit
  integer type were causing a fatal error on a 32-bit target.
- ARM and x86 ports: wrong register allocation for some calls to
  function pointers.
```

```
Release 2.7.1, 2016-07-18
=========================

[…]

Bug fixing:
- Fixed a compile-time assertion failure involving builtins
  taking a 64-bit integer parameter and given an unsigned 32-bit
integer
  argument.
- Updates to the Cminor parser.


Release 2.7, 2016-06-29
=======================

[…]

Bug fixing:
- Some declarations within C expressions were incorrectly ignored
  (e.g. "sizeof(enum e {A})").
- ARM in Thumb mode: incorrect "movs" instructions involving the stack
  pointer register were generated.
```

Verdi

Formally Verifying Distributed Systems

executable code and run their systems on real networks.[2] Assuming the network semantics correctly describes all possible behaviors of the system's environment, Verdi guarantees that Φ holds on all executions of the system.[3]

3. This also assumes the correctness of Verdi's trusted computing base (TCB), which includes: the soundness of Coq's logic, the correctness of Coq's proof checker, the correctness of Verdi's shim, and the correctness of OCaml's compiler and runtime, etc.↩

An Empirical Study on the Correctness of Formally Verified Distributed Systems

Pedro Fonseca Kaiyuan Zhang Xi Wang Arvind Krishnamurthy

University of Washington

Bug	Component	Trigger	Incorrect results	Crash	Impact	Reported	Fixed	PK
Specification								
I1	High-level specification	Packet duplication	-	-	Void exactly-once guarantee	✓	-	✓
C4	Test case	-	-	-	Void client guarantee	✓	✓	-
Verification tool								
I2	Verification framework	Incompatible libraries	-	-	Verify incorrect programs	✓	✓	✓
I3	Verification framework	Signal delivered	-	-	Verify incorrect programs	✓	✓	-
I4	Binary libraries	-	-	-	Prevent verification	-	✓	✓
Shim layer								
V1	Client-server communication	Partial socket read	-	✓	Crash server	✓	-	✓
V2	Client-server communication	Client input	✓	✓	Inject commands	✓	-	✓
V3	Recovery	Replica crash	-	✓	Crash server	✓	-	✓
V4	Recovery	Replica crash	✓	✓	Crash server	✓	-	✓
V5	Recovery	OS error during recovery	✓	-	Incomplete recovery	✓	-	✓
V6	Server-server communication	Lagging replica	-	✓	Crash server	-	✓	✓
V7	Server-server communication	Lagging replica	-	✓	Crash server	-	✓	✓
V8	Server-server communication	Lagging replica	-	✓	Crash server	✓	-	-
C1	Server-server communication	Packet duplication	✓	-	Violate causal consistency	✓	-	✓
C2	Server-server communication	Packet loss	-	✓	Return stale results	✓	-	✓
C3	Server-server communication	Client input	✓	✓	Hang and corrupt storage	✓	-	✓

Figure 3: Bugs that our analysis found in the high-level specification, verification tool, and shim layer of verified distributed systems. Some bugs caused servers to crash or to produce incorrect results, and most bugs are detected by our testing toolchain (PK). We reported all listed bugs to developers, except bug V6 and bug V7, which the developers had already fixed.

An Empirical Study on the Correctness of Formally Verified Distributed Systems

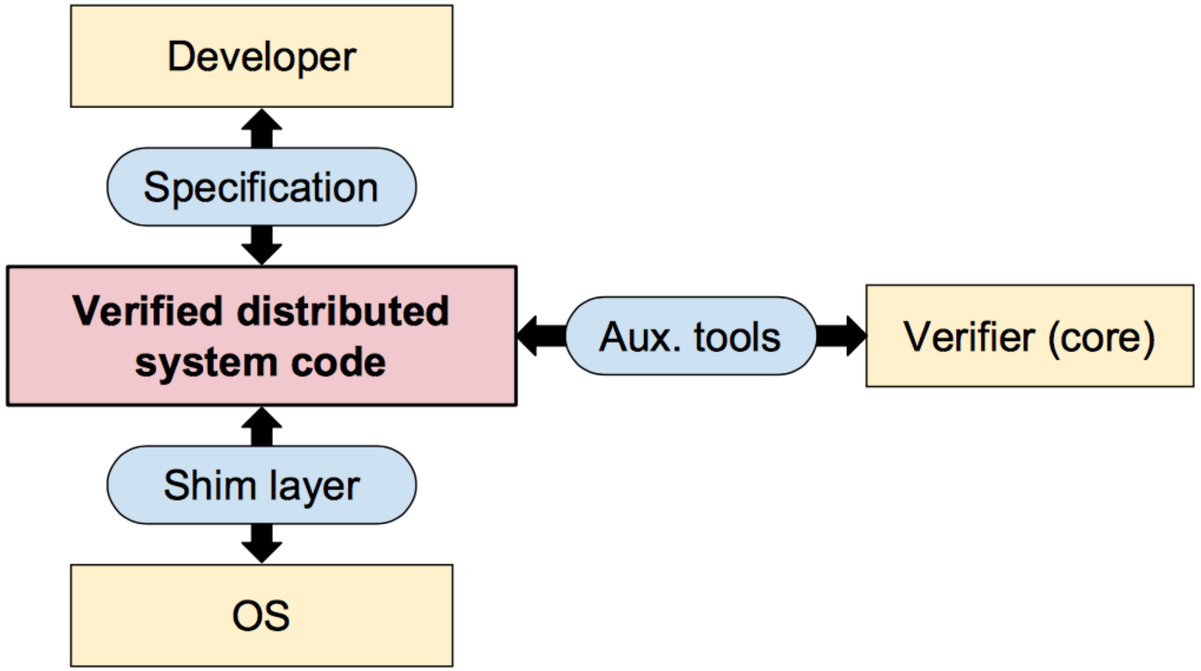
Pedro Fonseca Kaiyuan Zhang Xi Wang Arvind Krishnamurthy
University of Washington

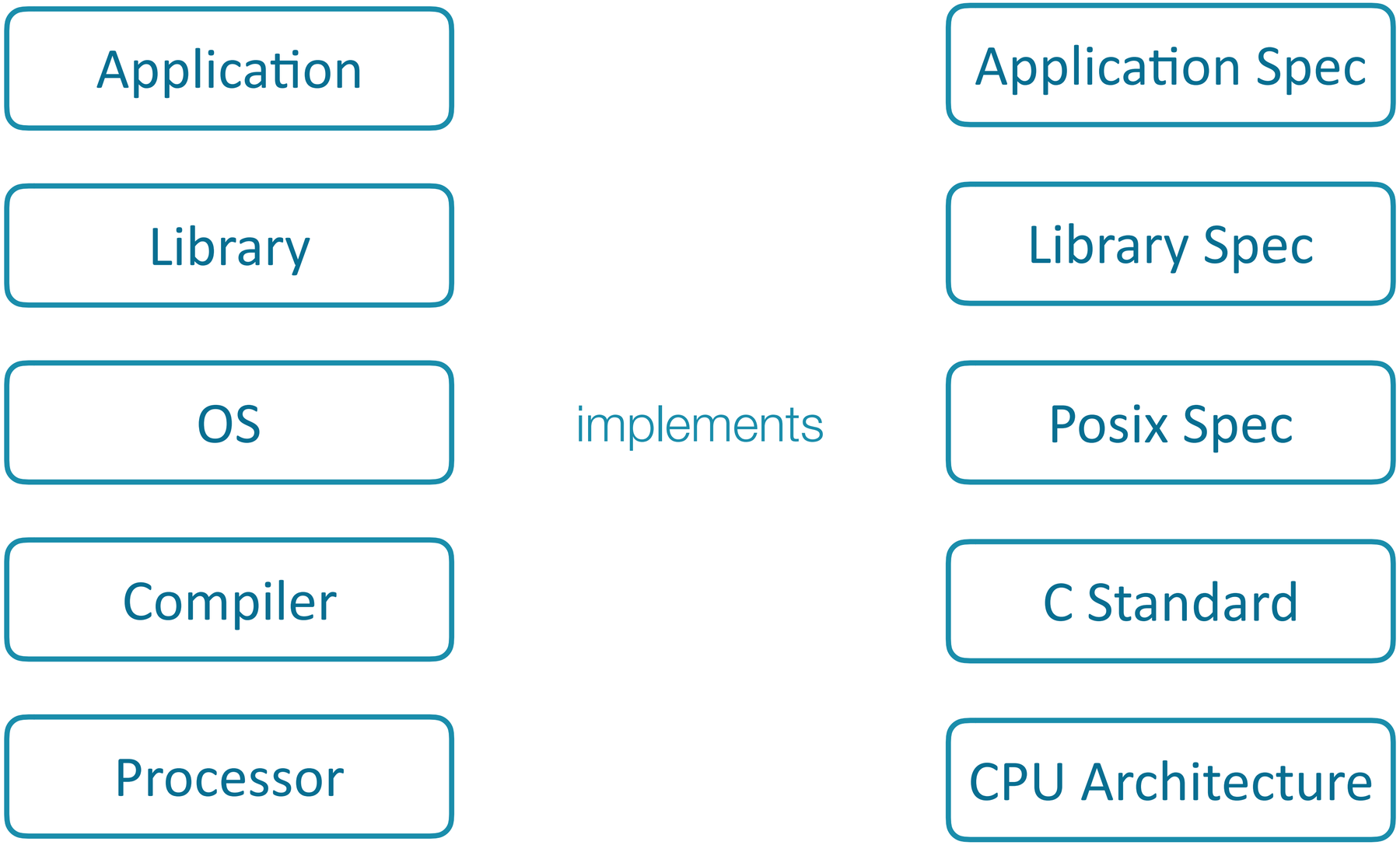
	LogCabin	ZooKeeper	Etcd	Cassandra	Total
Communication	4	1	3	9	17
Recovery	0	1	0	7	8
Logging / snapshot	5	5	6	5	21
Protocol	1	1	2	8	12
Configuration	1	2	0	0	3
Client library	1	23	11	7	42
Reconfiguration	1	6	8	17	32
Management tools	1	22	21	116	160
Single-node storage	1	18	11	200	230
Concurrency	3	1	2	18	24
Total	23	80	65	387	555

Figure 13: Sample of known bugs from the bug reports of *unverified* distributed systems.

An Empirical Study on the Correctness of Formally Verified Distributed Systems

Pedro Fonseca Kaiyuan Zhang Xi Wang Arvind Krishnamurthy
University of Washington




Application
Library
OS
Compiler
Processor
implements
Application Spec
Library Spec
Posix Spec
C Standard
CPU Architecture

Trusted Computing Base

- Application Spec
- Library Spec
- Posix Spec
- C Standard
- CPU Architecture

Specifications are part of your TCB

Testing and Formal Validation of Processor Specifications

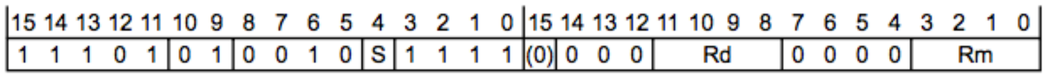
- Testing Specifications (FMCAD 2016)
- Formally Validating Processors (CAV 2016)
- Formally Validating Specifications (submitted)
  - Generating Testcases
  - Security Checking
  - Booting an OS
  - Fuzzing an OS

The Virtuous Cycle

ISA Specification

Encoding T3 ARMv7-M

MOV{S}<c>.W <Rd>,<Rm>



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	0	0	1	0	S	1	1	1	1	(0)	0	0	0	Rd				0	0	0	0	Rm			

```
d = UInt(Rd);  m = UInt(Rm);  setflags = (S == '1');
if setflags && (d IN {13,15} || m IN {13,15}) then UNPREDICTABLE;
if !setflags && (d == 15 || m == 15 || (d == 13 && m == 13)) then UNPREDICTABLE;


if ConditionPassed() then
    EncodingSpecificOperations();
    result = R[m];
    if d == 15 then
        ALUWritePC(result);  // setflags is always FALSE here
    else
        R[d] = result;
        if setflags then
            APSR.N = result<31>;
            APSR.Z = IsZeroBit(result);
            // APSR.C unchanged
            // APSR.V unchanged
```

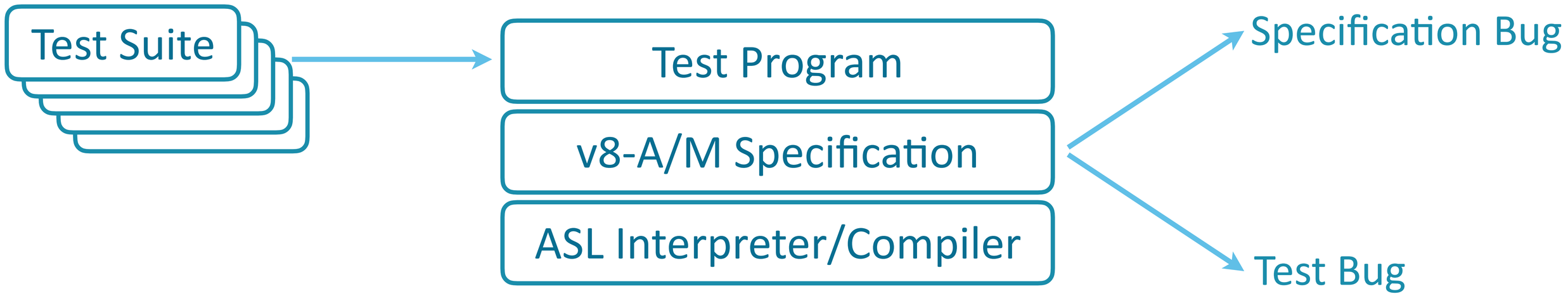
System Specification

```
AArch64.DataAbort(bits(64) vaddress, FaultRecord fault)

    route_to_el3 = HaveEL(EL3) && SCR_EL3.EA == '1' && IsExternalAbort(fault);
    route_to_el2 = (HaveEL(EL2) && !IsSecure() && PSTATE.EL IN {EL0,EL1} &&
                    (HCR_EL2.TGE == '1' || IsSecondStage(fault)));

    bits(64) preferred_exception_return = ThisInstrAddr();
    vect_offset = 0x0;

    exception = AArch64.AbortSyndrome(Exception_DataAbort, fault, vaddress);

    if PSTATE.EL == EL3 || route_to_el3 then
        AArch64.TakeException(EL3, exception, preferred_exception_return, vect_offset);
    elsif PSTATE.EL == EL2 || route_to_el2 then
        AArch64.TakeException(EL2, exception, preferred_exception_return, vect_offset);
    else
        AArch64.TakeException(EL1, exception, preferred_exception_return, vect_offset);
```

ARM Spec (lines of code)

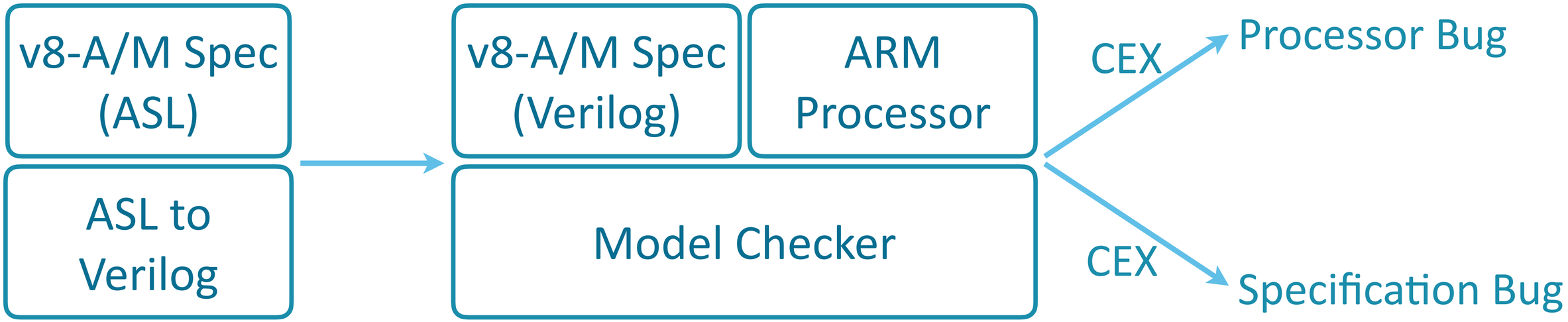
	v8-A	v8-M
Instructions Int/FP/SIMD	26,000	6,000
Exceptions	4,000	3,000
Memory	3,000	1,000
Debug	3,000	1,000
Misc	5,500	2,000
(Test support)	1,500	2,000
Total	**43,000**	**15,000**

Testing Specifications



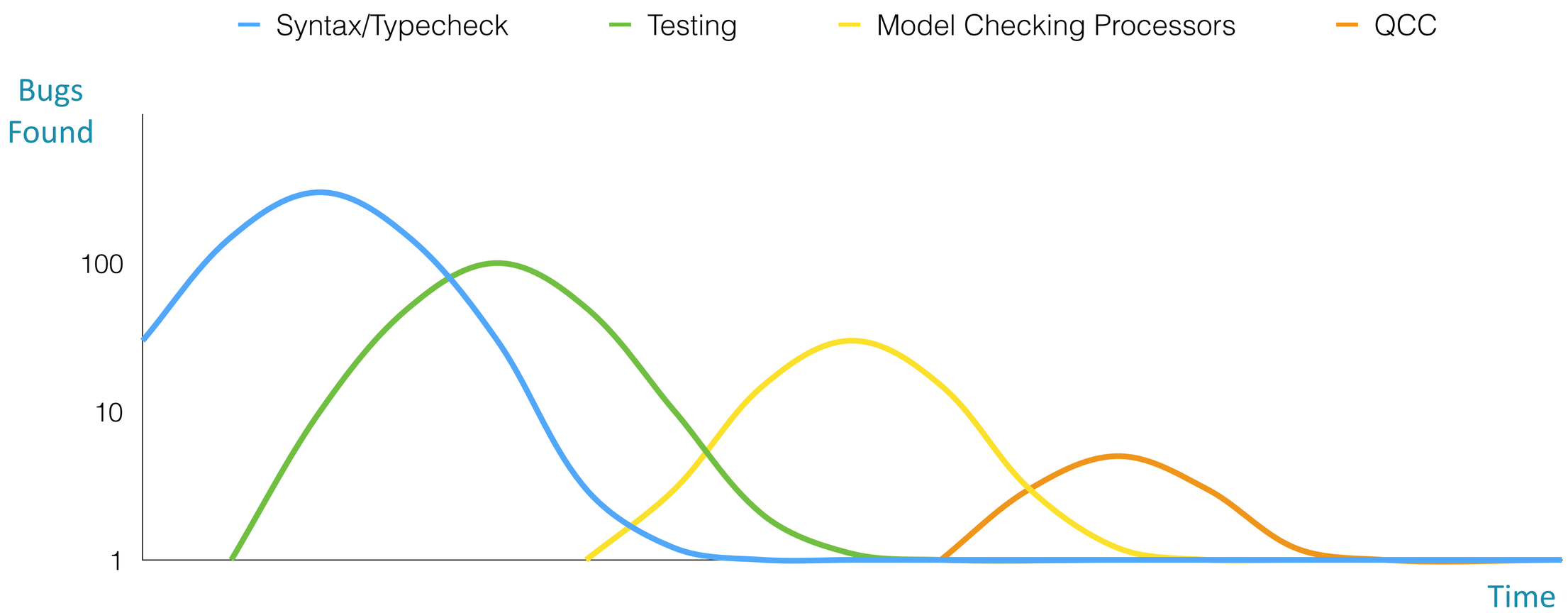
11,000 test programs
2 Billion instructions

Formally Validating Processors

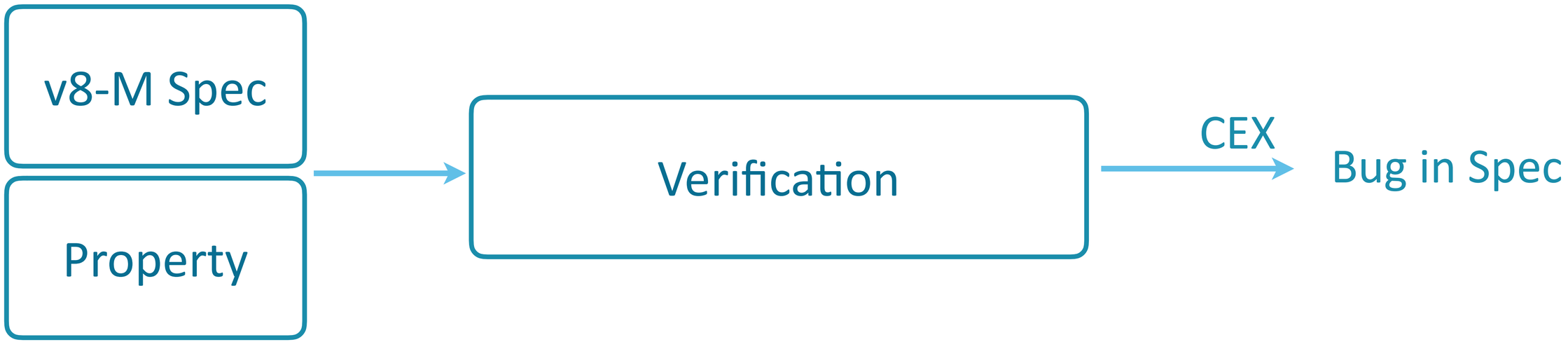


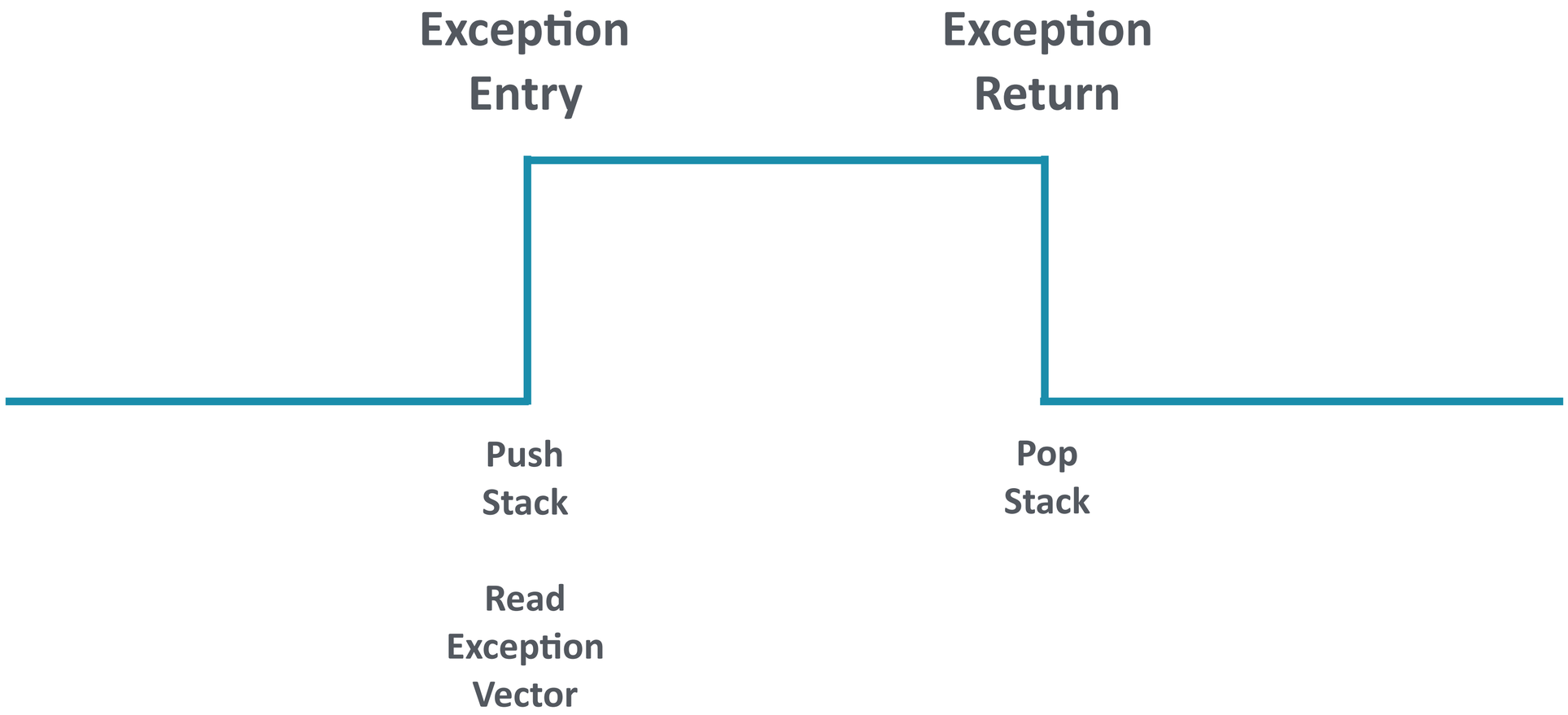
Finding Bugs in Specs

(Artists Impression)



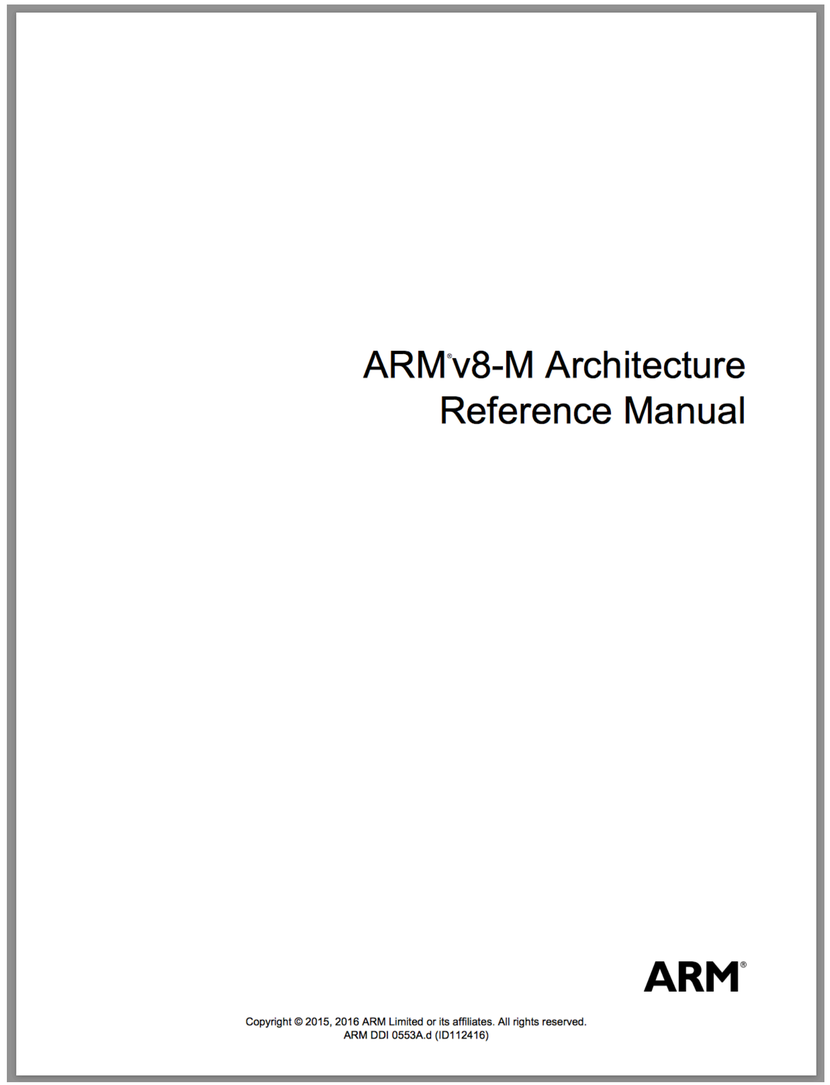
Formally Validating Specifications




Exception
Entry
Exception
Return
Push
Stack
Pop
Stack
Read
Exception
Vector

Derived Exception

Lockup



Entry to lockup from an exception causes:

- Any Fault Status Registers associated with the exception to be updated.
- No update to the exception state, pending or active.
- The PC to be set to `0xEFFFFFFE`.
- EPSR.IT to be become UNKNOWN.

In addition, HFSR.FORCED is not set to 1.

When the PE is in lockup:

- DHCSR.S_LOCKUP reads as 1.
- The PC reads as `0xEFFFFFFE`. This is an XN address.
- The PE stops fetching and executing instructions.
- If the implementation provides an external **LOCKUP** signal, **LOCKUP** is asserted HIGH.

Exit from lockup is by any of the following:

- A Cold reset.
- A Warm reset.
- Entry to Debug state.
- Preemption by a higher priority exception.

Entry to lockup from an exception causes:

- Any Fault Status Registers associated with the exception to be updated.
- No update to the exception state, pending or active.
- The PC to be set to 0xEFFFFFFE.
- EPSR.IT to be become UNKNOWN.

In addition, HFSR.FORCED is not set to 1.

rule lockup entry

assume Rose(LockedUp);

assume ¬Called(TakeReset);

property a HaveMainExt() ⇒ CFSR != 0;

property b1 Stable(ExnPending);

property b2 Stable(ExnActive);

property c RName[RNamesPC] = 0xEFFFFFFE;

property e Stable(HFSR.FORCED);

Exit from lockup is by any of the following:

- A Cold reset.
- A Warm reset.
- Entry to Debug state.
- Preemption by a higher priority exception.

rule lockup_exit

assume Fell(LockedUp);

Called(TakeColdReset)

∨ Called(TakeReset)

∨ Rose(Halted)

∨ Called(ExceptionEntry);

When the PE is in lockup:

- DHCSR.S_LOCKUP reads as 1.
- The PC reads as `0xEFFFFFFE`. This is an XN address.
- The PE stops fetching and executing instructions.
- If the implementation provides an external **LOCKUP** signal, **LOCKUP** is asserted HIGH.

rule lockup

assume LockedUp;

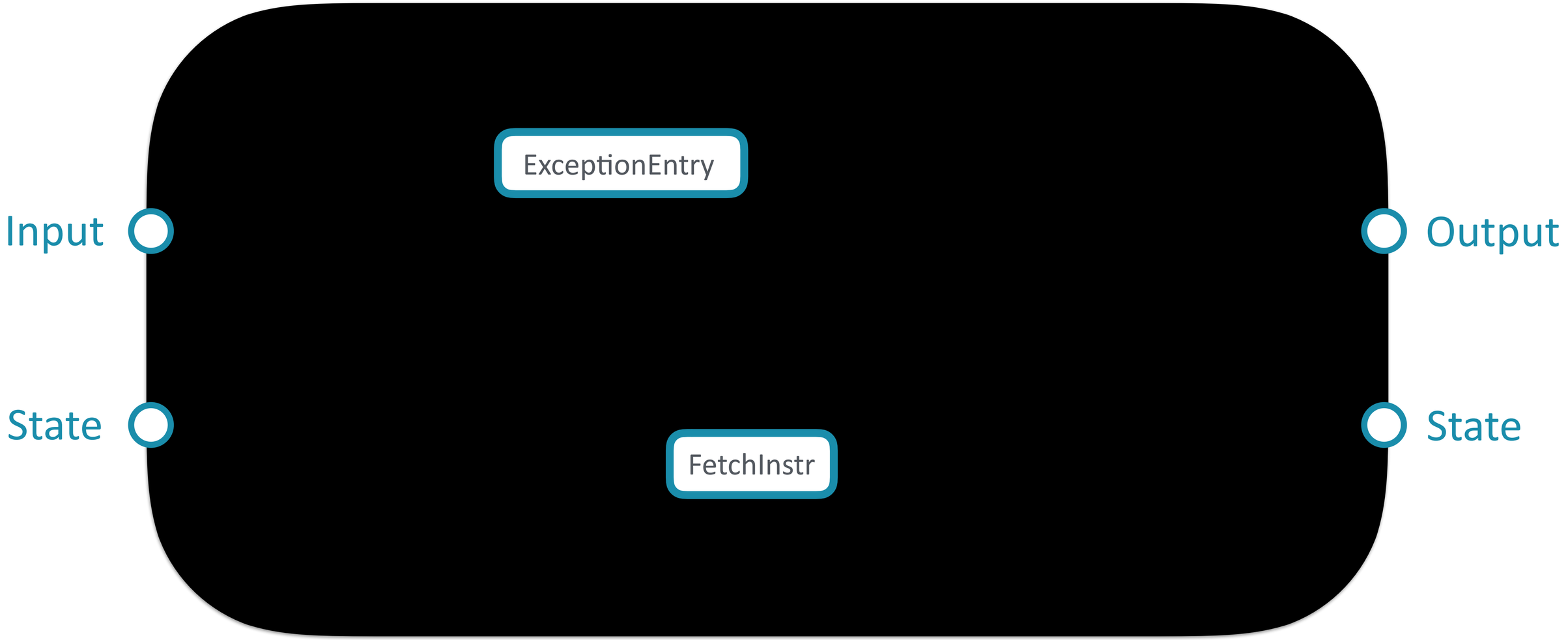
invariant a DHCSR.S_LOCKUP = 1;

invariant b PC == 0xEFFFFFFE;

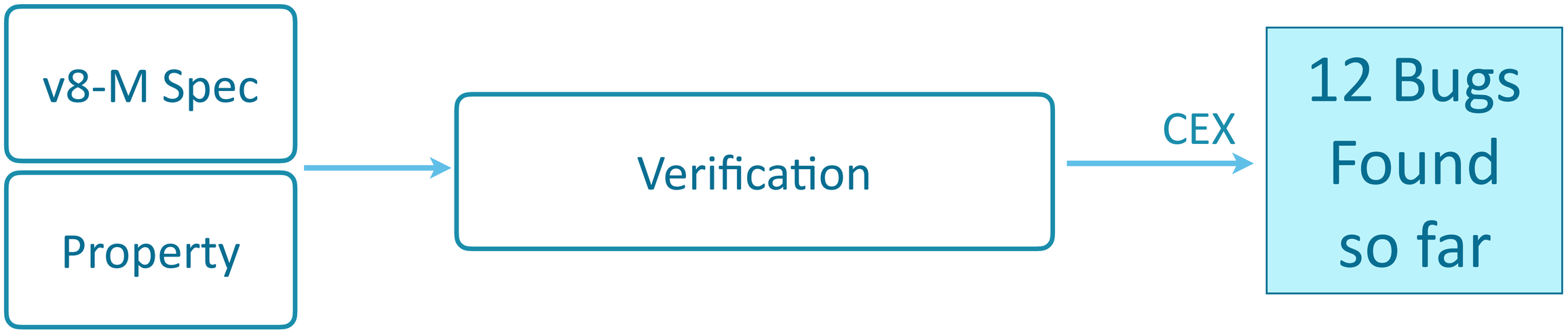
property c

assume Past(LockedUp);
¬ Called(FetchInstr) ∧ ¬Called(DecodeExecute);

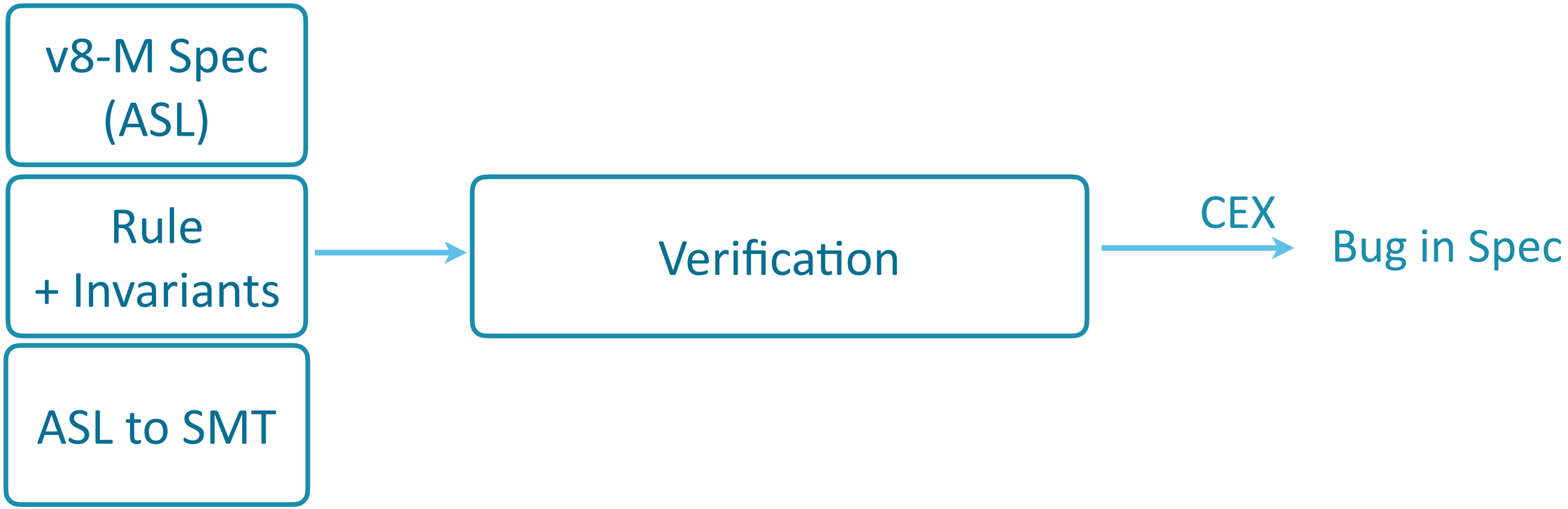

Input
State
Output
State


ExceptionEntry
Input
Output
State
State
FetchInstr

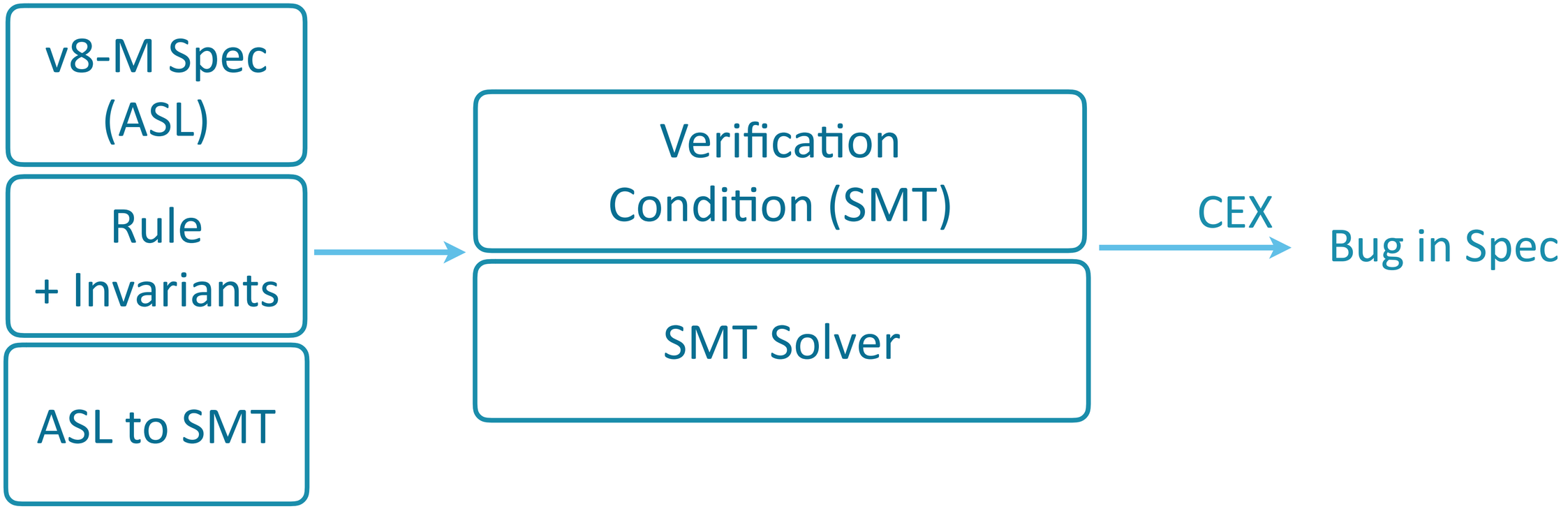
Formally Validating Specifications



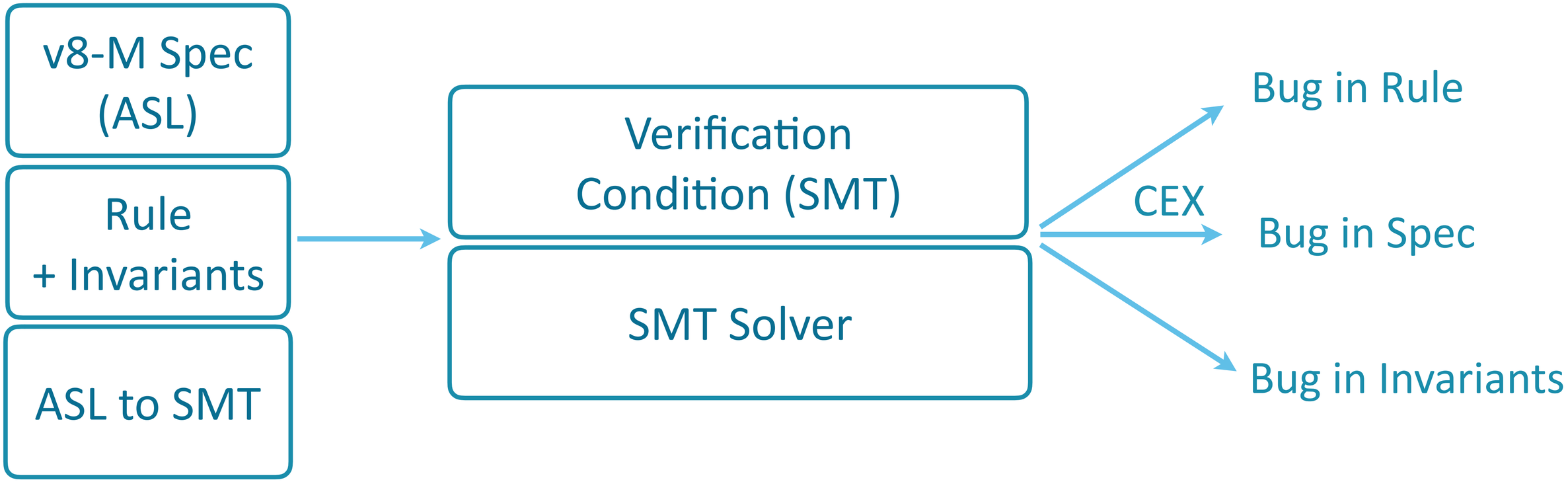
Formally Validating Specifications



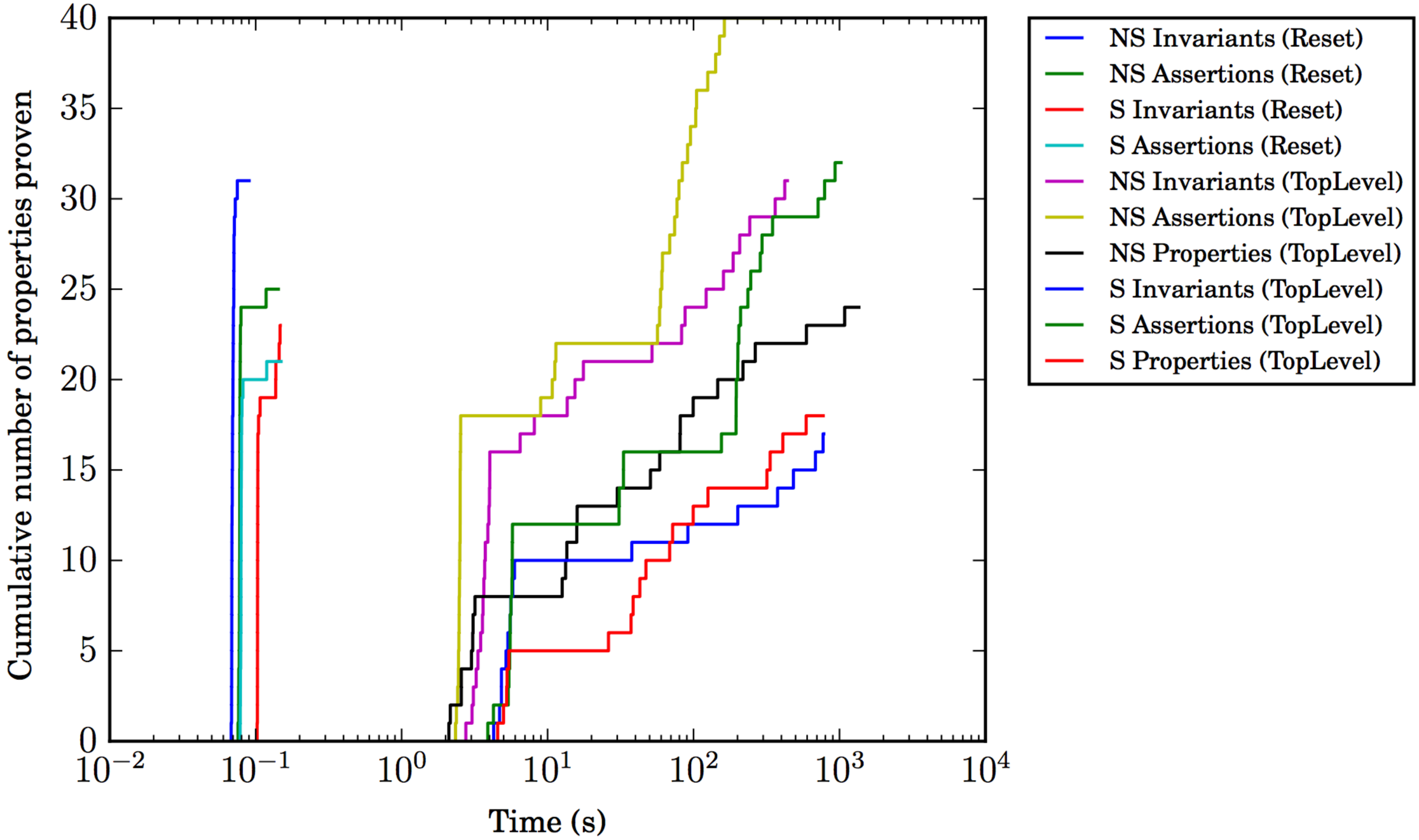
Formally Validating Specifications



Formally Validating Specifications



	TakeColdReset			TopLevel			
	Asserts	Bounds	Invariant	Asserts	Bounds	Invariant	Properties
Configuration = NS							
Total	25	2	32	41	2	32	25
Passed	25	2	32	41	2	32	21
Failed							4
Timeout							
Configuration = S							
Total	23	3	32	36	3	32	25
Passed	23	3	32	33	3	28	19
Failed							
Timeout				3		4	6


Cumulative number of properties proven
Time (s)
0
5
10
15
20
25
30
35
40
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
NS Invariants (Reset)
NS Assertions (Reset)
S Invariants (Reset)
S Assertions (Reset)
NS Invariants (TopLevel)
NS Assertions (TopLevel)
NS Properties (TopLevel)
S Invariants (TopLevel)
S Assertions (TopLevel)
S Properties (TopLevel)

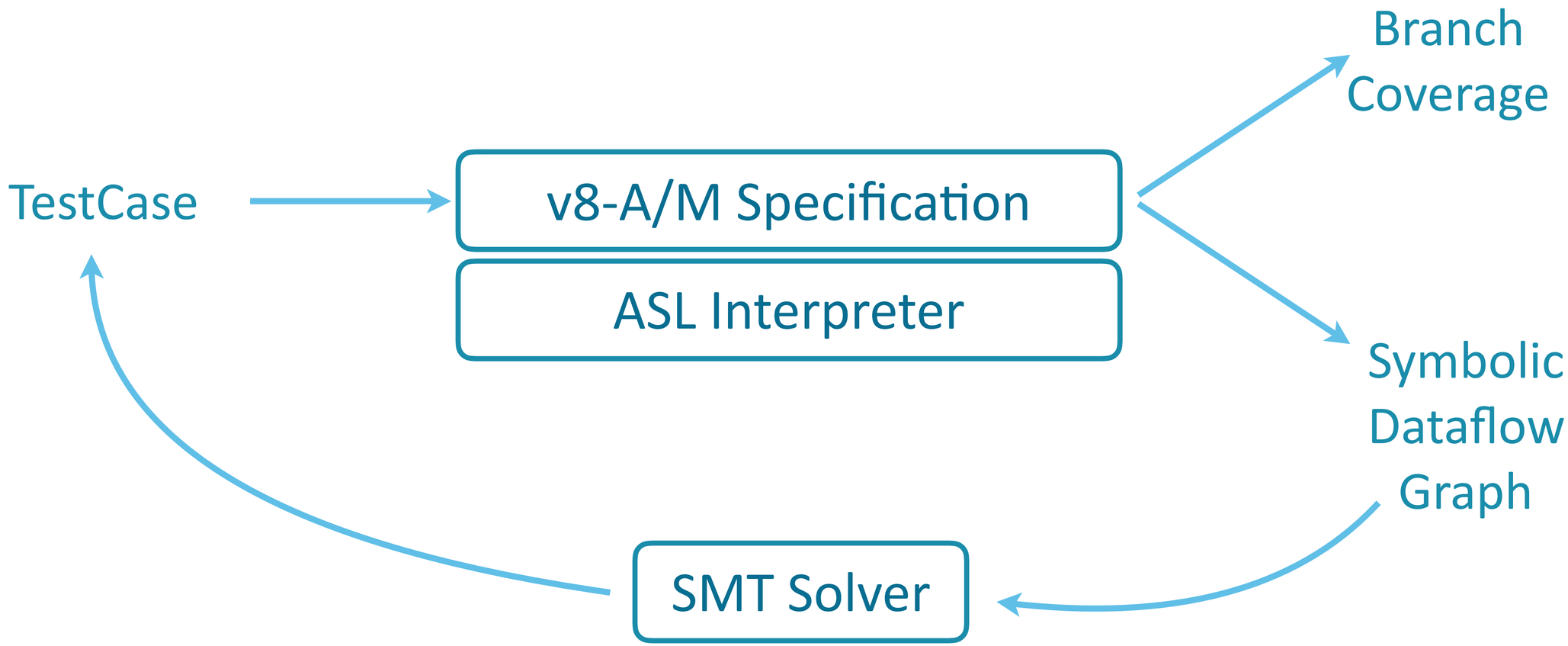
Specifications are part of your TCB

Testing and Formal Validation of Processor Specifications

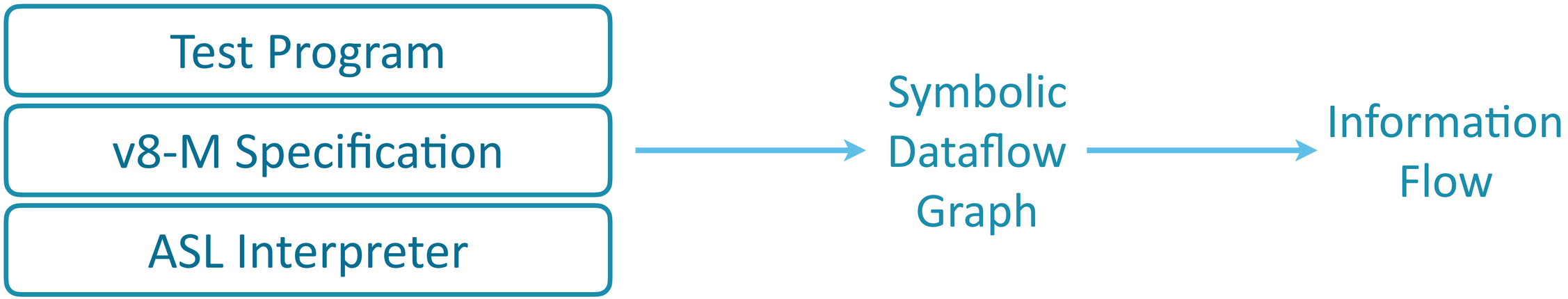
- Testing Specifications (FMCAD 2016)
- Formally Validating Processors (CAV 2016)
- Formally Validating Specifications (submitted)
- Generating Testcases
- Security Checking
- Booting an OS
- Fuzzing an OS

The Virtuous Cycle

Testcase Generation

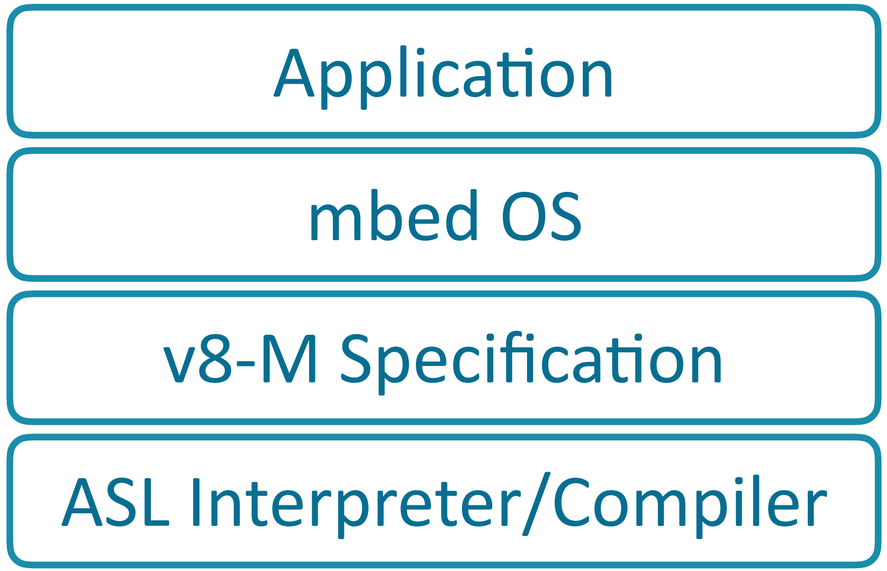


Security Checking



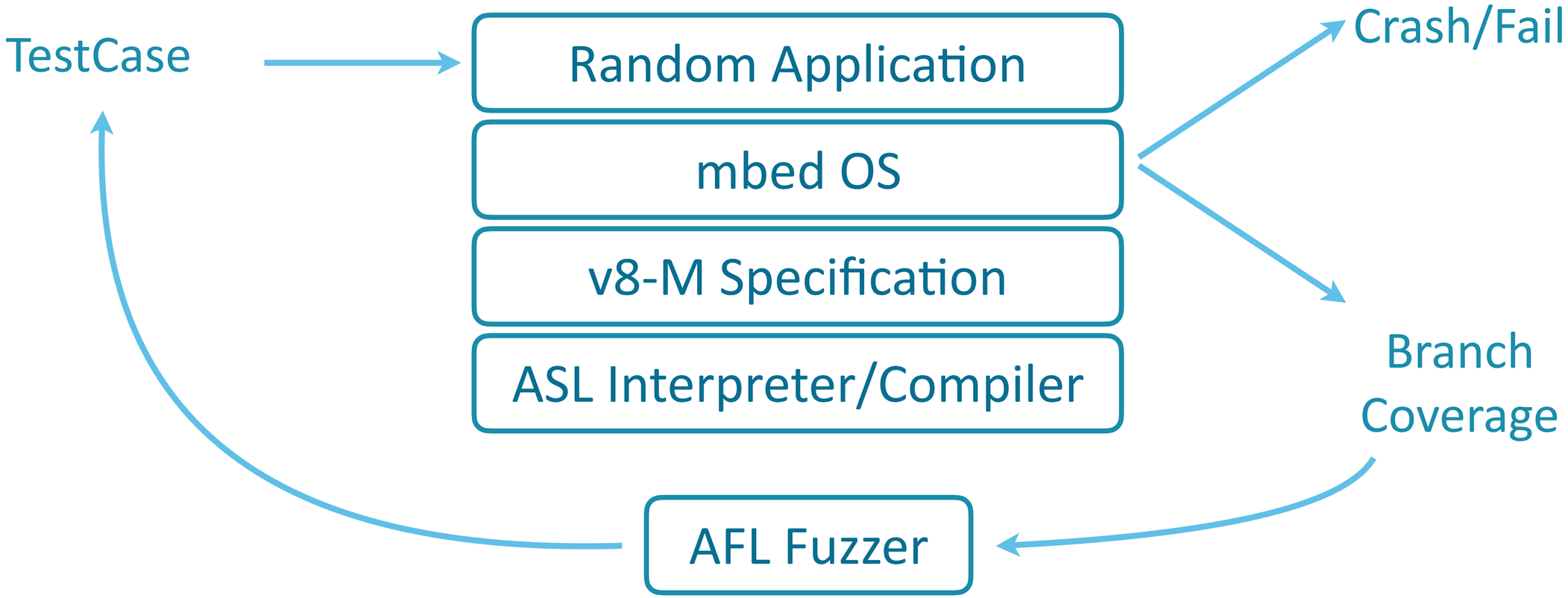
(Work by Jon French and Nathan Chong)

Booting an OS

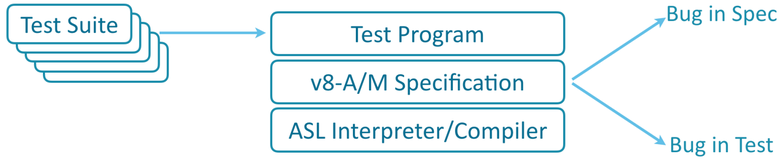


(Work by Jon French and Nathan Chong)

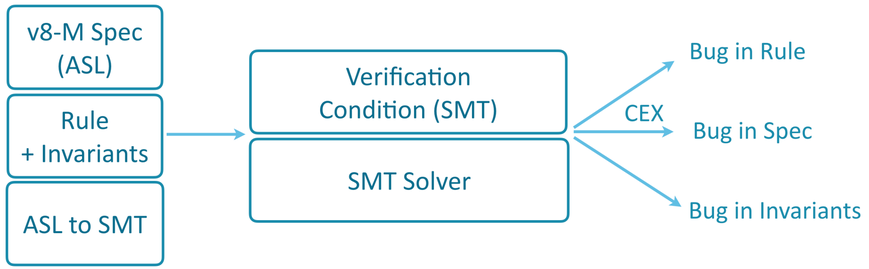
Fuzzing the mbed OS



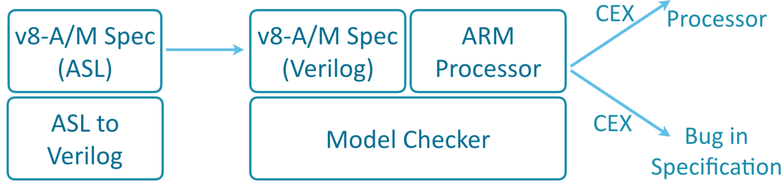
Testing Specifications



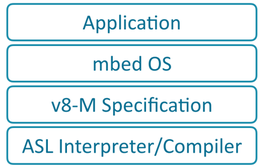
Formally Validating Specifications



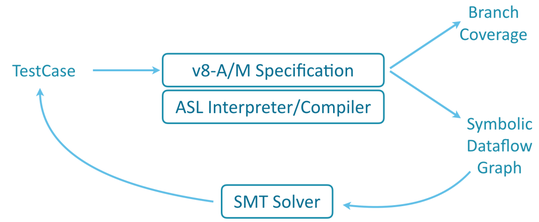
Verifying Processors



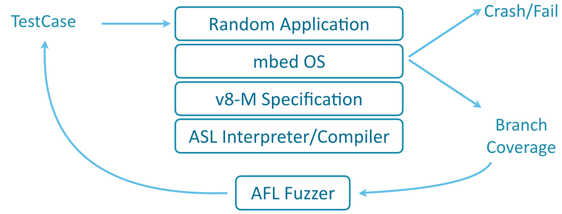
Booting an OS



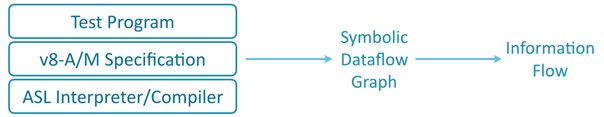
Testcase Generation



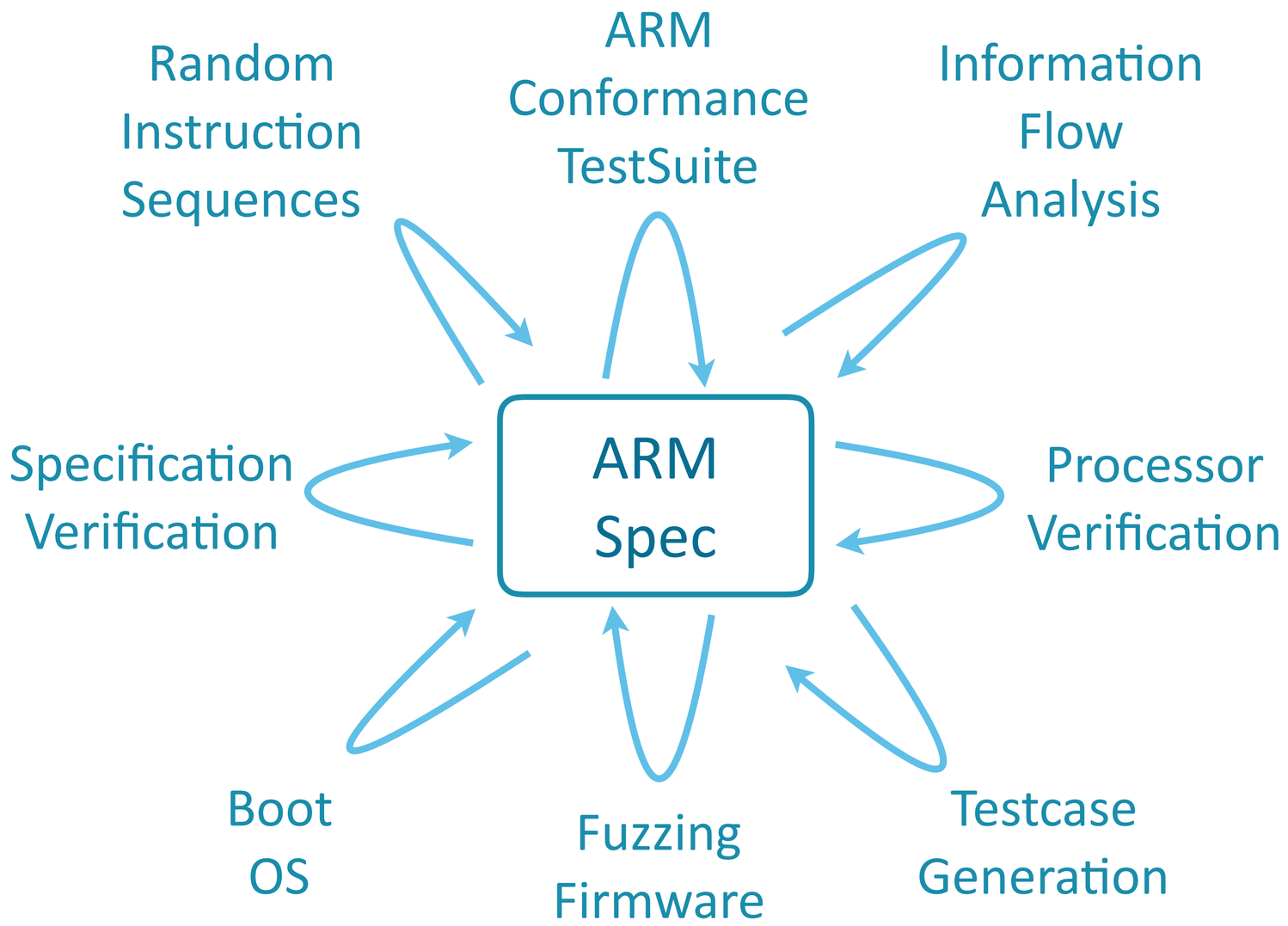
Fuzzing the mbed OS



Security Checking



Creating a Virtuous Cycle



How can you trust formally verified software?

Don’t forget the TCB

Specifications — The Next Formal Verification Bottleneck

- Too large to be “obviously correct”
- Testing
- Formally validating implementations
- Formally validating specifications

Hiring in Security and Correctness group — contact me

End

alastair.reid@arm.com
@alastair_d_reid