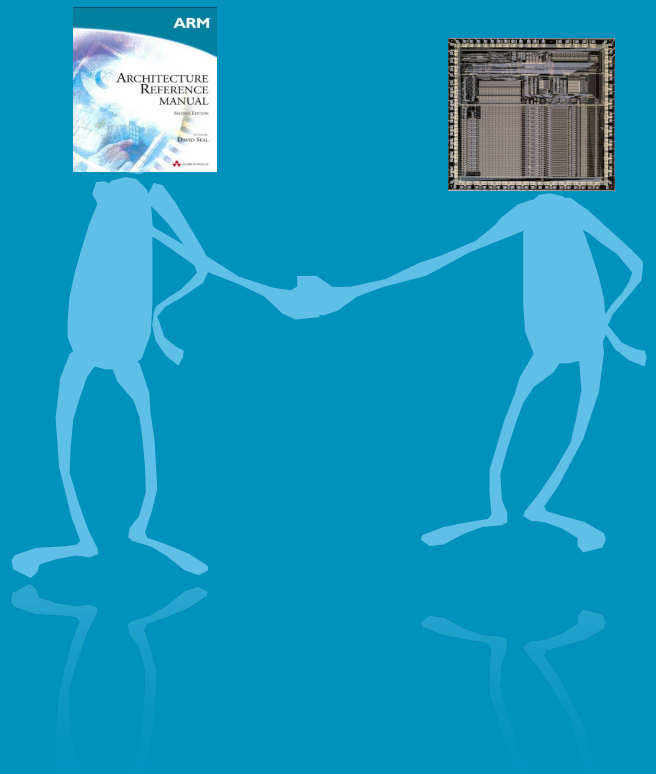
arm



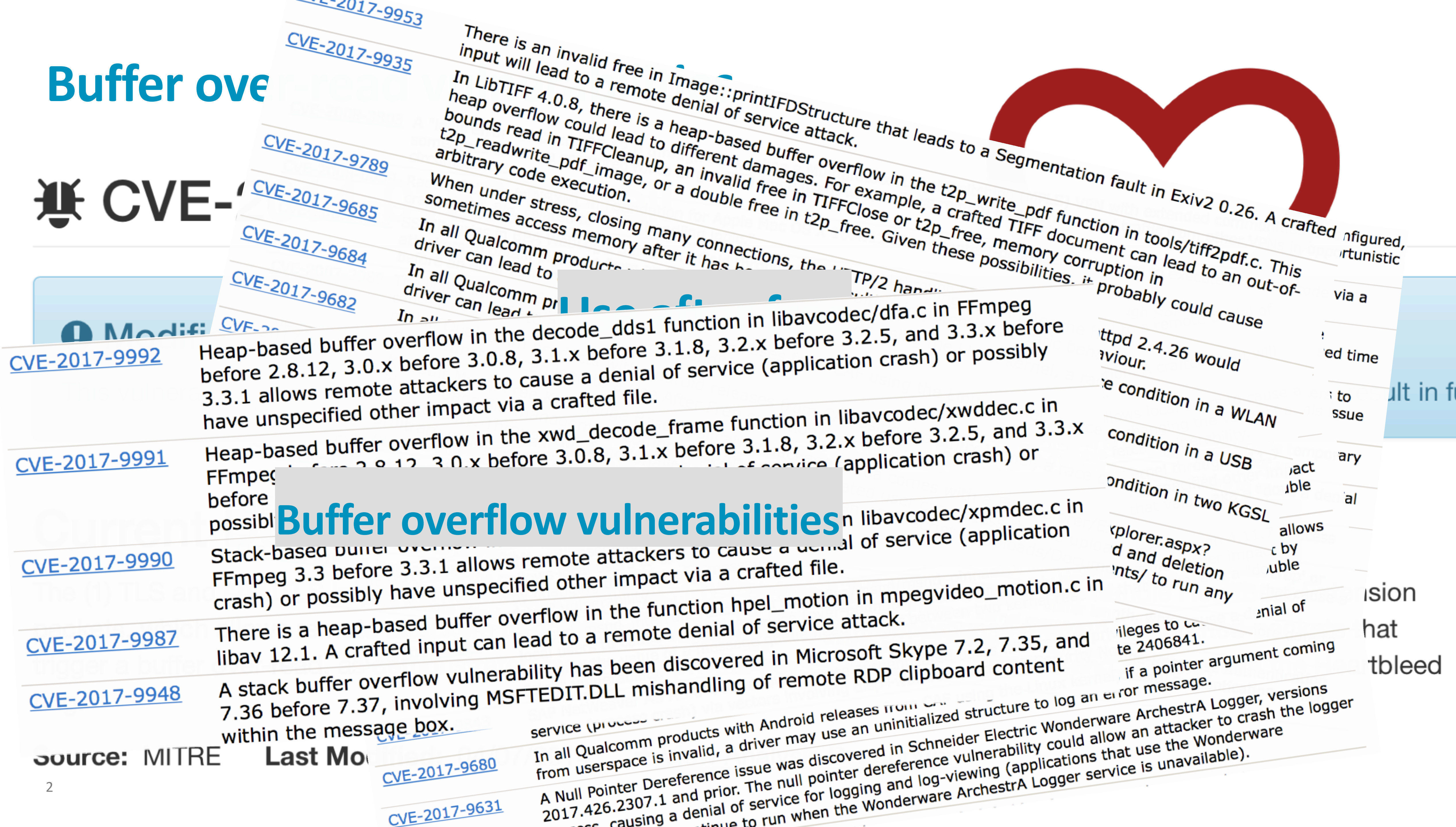
How Can You Trust Formally Verified Software?

Alastair Reid

Arm Research

@alastair_d_reid

Buffer overflow vulnerabilities



Formal verification

Of libraries and apps

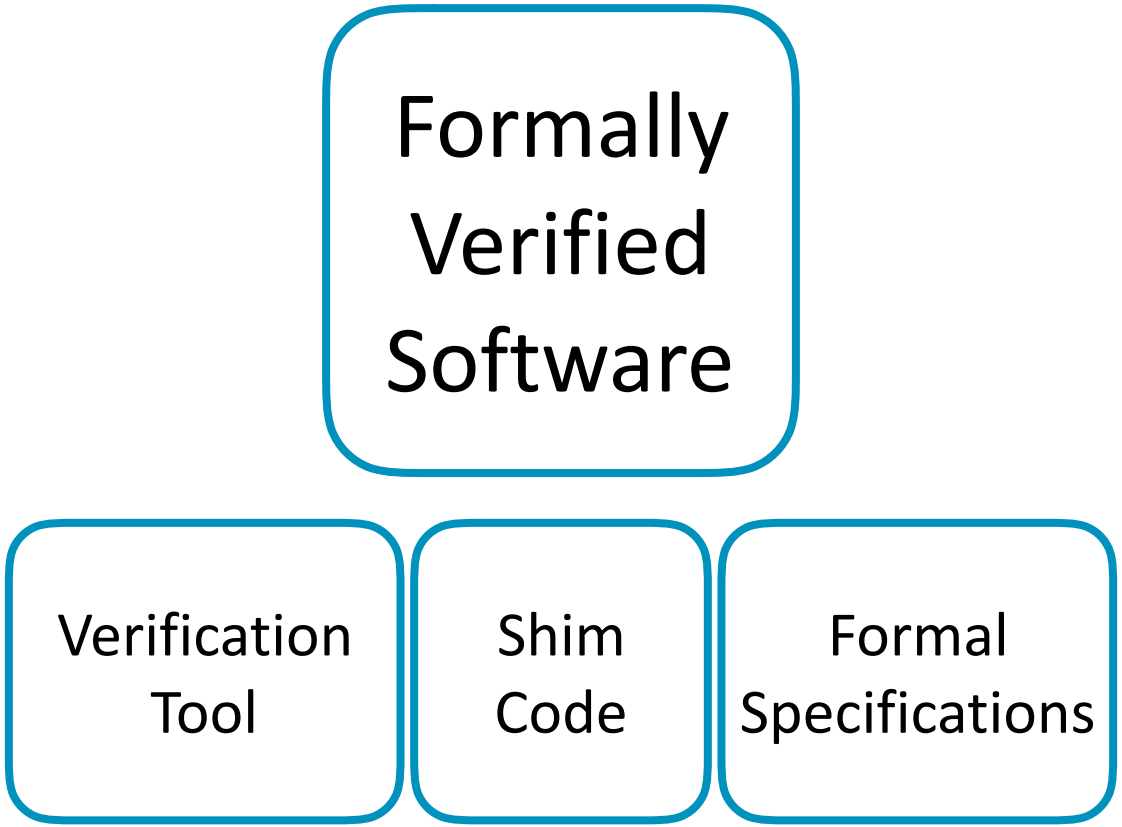
Of compilers

COMPCERT

Of operating systems

arm

Fonseca et al., An Empirical Study on the Correctness of Formally Verified Distributed Systems, Eurosys ‘17

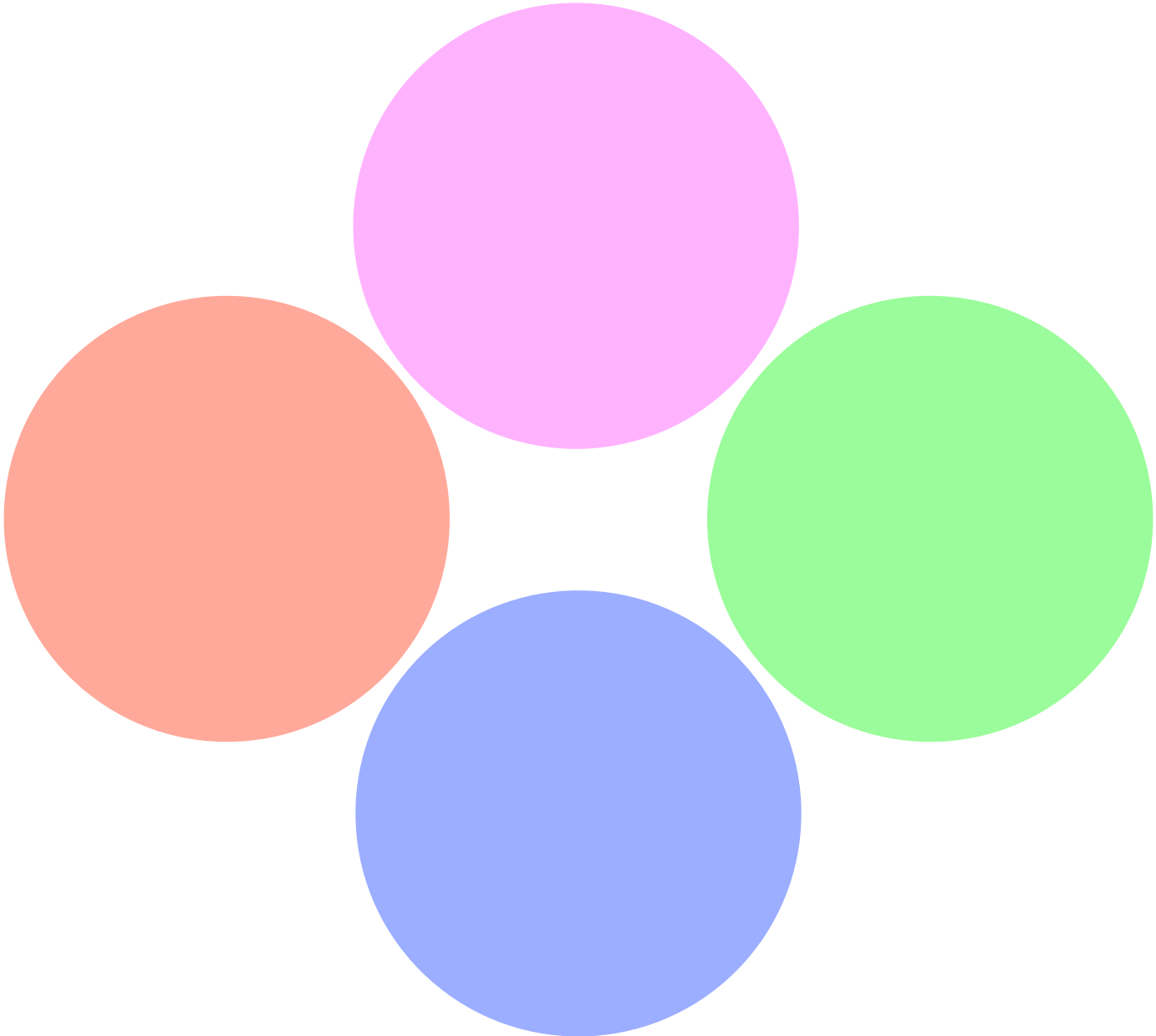


arm

Takeaway #1: 3 key questions to ask

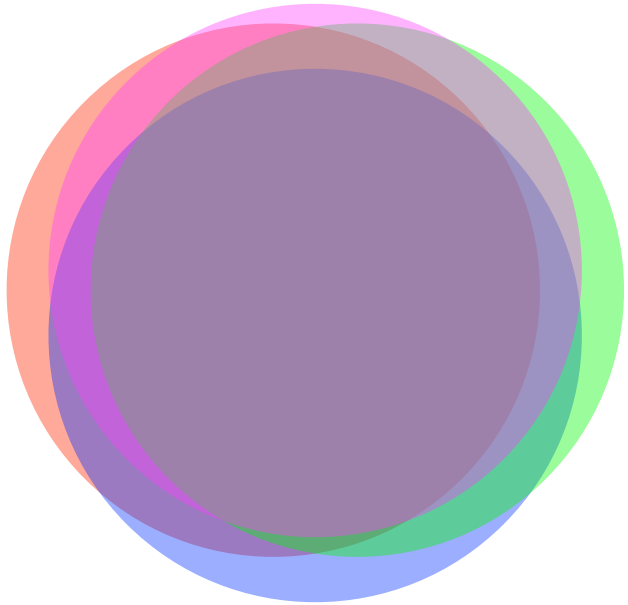
1. What specifications does your proof rely on?
2. Why do you trust those specifications?
3. Does anybody else use these specifications?

Takeaway #2: Specifications must have multiple uses



arm

Takeaway #2: Specifications must have multiple uses



arm

How can you trust formally verified software?

How can you trust formally verified software?

- Specifications are part of the TCB
- 3 key questions
- Specifications must have multiple users

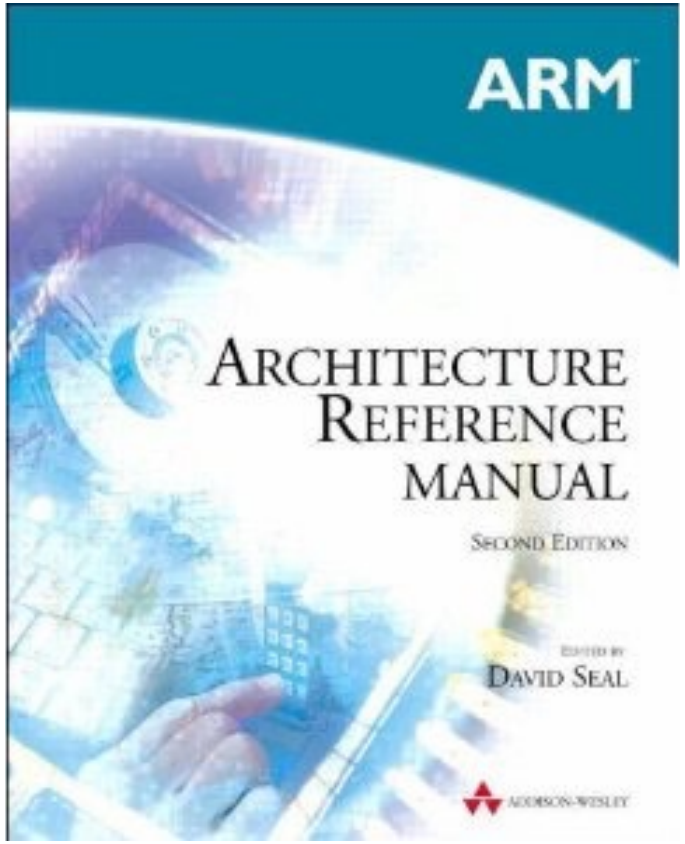
How can you trust formal specifications?

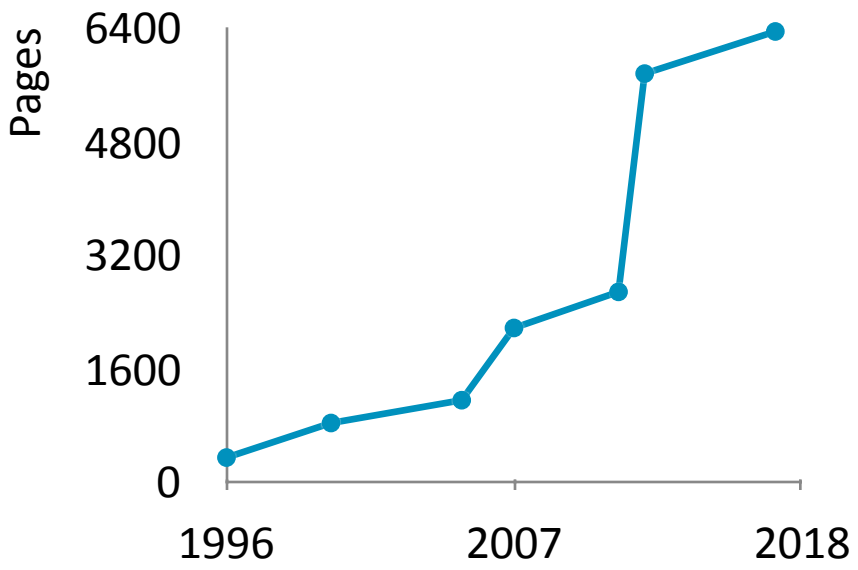
- Testing specifications
- Verifying processors
- Verifying specifications

How can you trust formally verified software?

arm

Arm Architecture Reference Manual (ARMARM)





32-bit / 64-bit Instructions
Exceptions / Interrupts
Privilege / Security
Virtual Memory
System registers
Debug / Trace
Profiling
...

arm

English prose

R_{JRJC}

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.

R_{VGNW}

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.

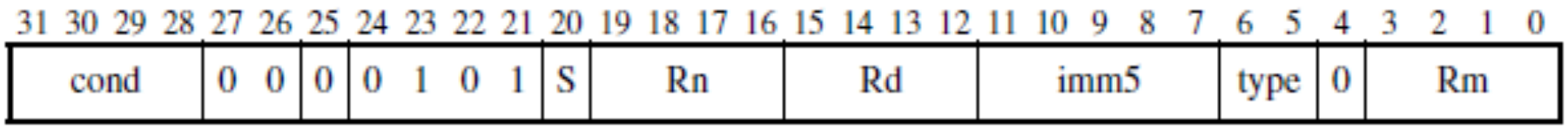
In addition, HFSR.FORCED is not set to 1.

arm

Pseudocode

Encoding A1 ARMv4*, ARMv5T*, ARMv6*, ARMv7

```
ADC{S}<c> <Rd>,<Rn>,<Rm>{,<shift>}
```



31 30 29 28	27 26	25	24 23 22 21	20	19 18 17 16	15 14 13 12	11 10 9 8 7	6 5	4	3 2 1 0
cond	0 0	0	0 1 0 1	S	Rn	Rd	imm5	type	0	Rm

```
if Rd == '1111' && S == '1' then SEE SUBS PC, LR and related instructions;
d = UInt(Rd);  n = UInt(Rn);  m = UInt(Rm);  setflags = (S == '1');
(shift_t, shift_n) = DecodeImmShift(type, imm5);

if ConditionPassed() then
    EncodingSpecificOperations();
    shifted = Shift(R[m], shift_t, shift_n, APSR.C);
    (result, carry, overflow) = AddWithCarry(R[n], shifted, APSR.C);
    if d == 15 then          // Can only occur for ARM encoding
        ALUWritePC(result);  // setflags is always FALSE here
    else
        R[d] = result;
        if setflags then
            APSR.N = result<31>;
            APSR.Z = IsZeroBit(result);
            APSR.C = carry;
            APSR.V = overflow;
```


arm

Arm Architecture Specification Language (ASL)

Indentation-based syntax

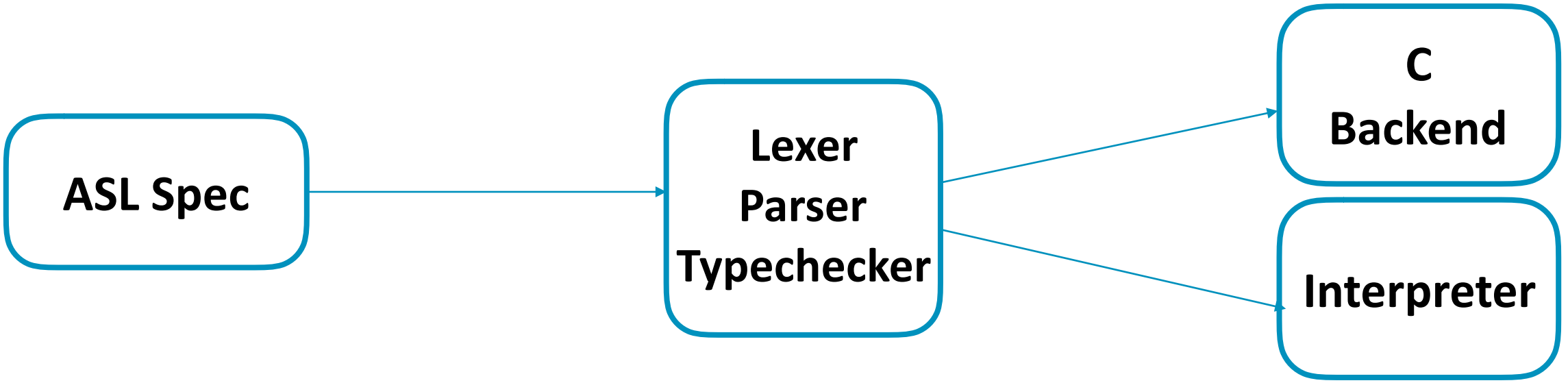
Imperative

First-order

Strongly typed (type inference, polymorphism, dependent types)

- Bit-vectors
- Unbounded integers
- Infinite precision reals
- Arrays, Records, Enumerations

Exceptions



arm

Architectural Conformance Suite

Processor architectural compliance sign-off

Large

- v8-A 11,000 test programs, > 2 billion instructions
- v8-M 3,500 test programs, > 250 million instructions

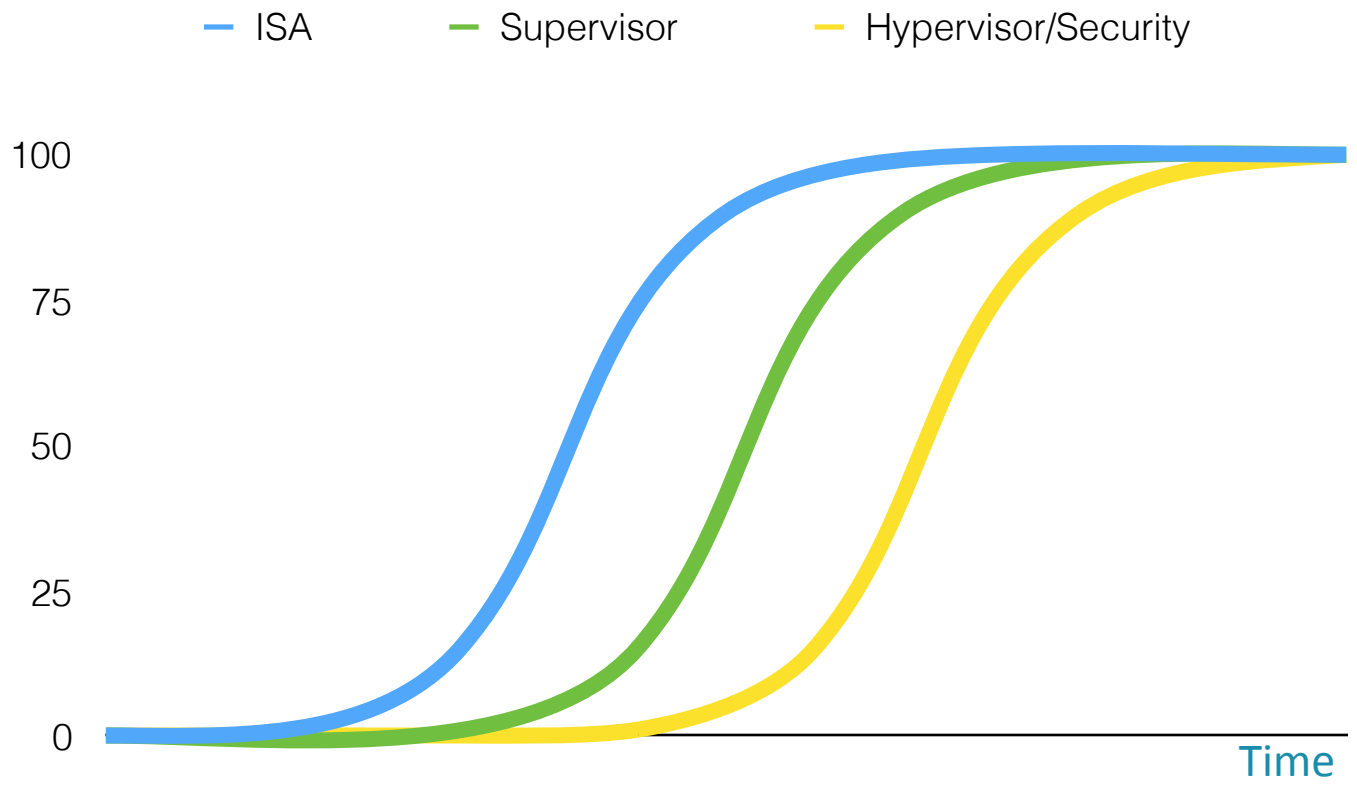
Thorough

- Tests dark corners of specification

arm

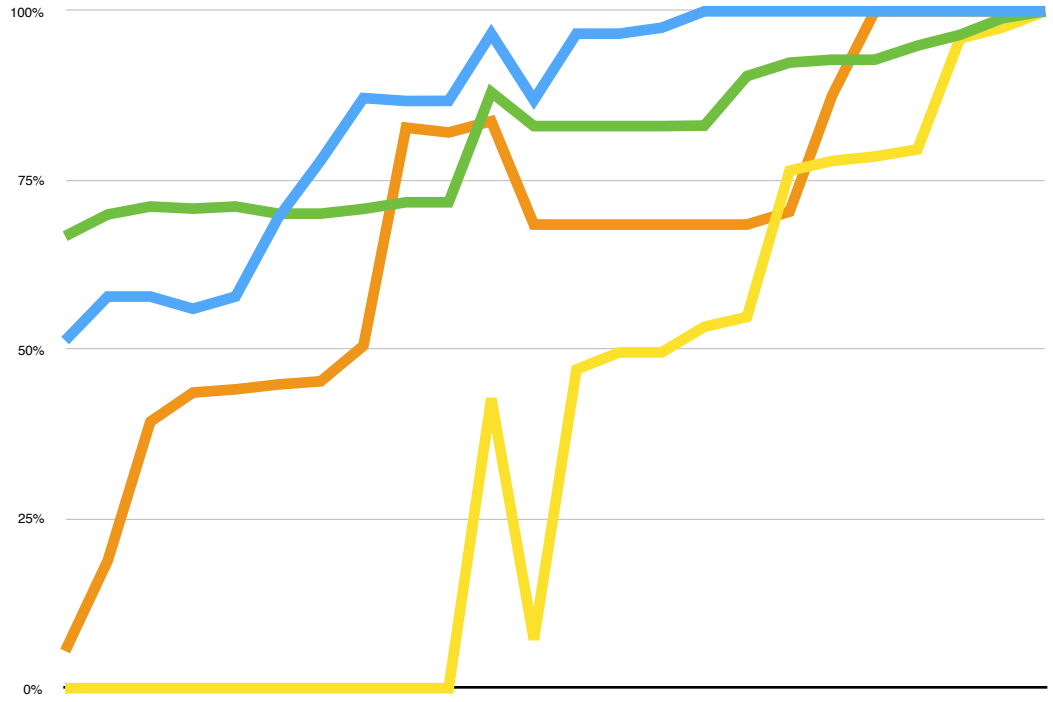
Testing Pass Rate

(Artists Impression)



arm

v8-A



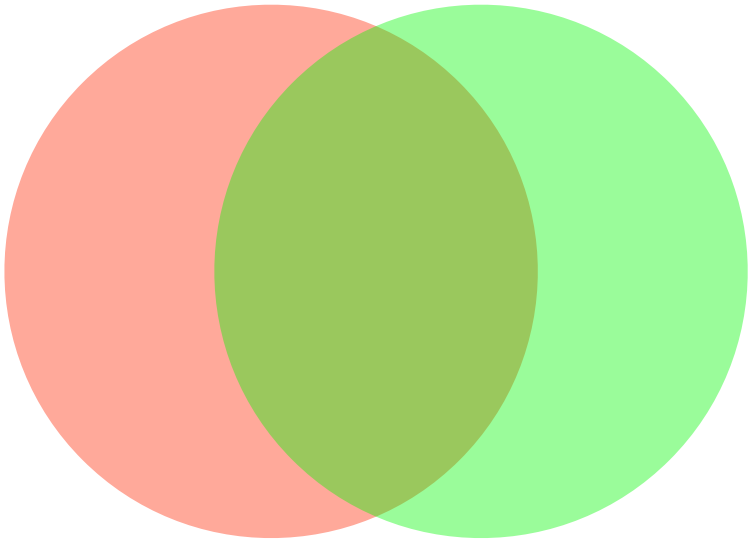
arm

Measuring architecture coverage of tests

Untested: op1*op2 == -3.0, FPCR.RND=-Inf


```
bits(N) FPRSqrtStepFused(bits(N) op1, bits(N) op2)
    assert N IN {32, 64};
    bits(N) result;
    op1 = FPNeg(op1);    // per FMSUB/FMLS
    (type1,sign1,value1) = FPUnpack(op1, FPCR);
    (type2,sign2,value2) = FPUnpack(op2, FPCR);
    (done,result) = FPProcessNaNs(type1, type2, op1, op2, FPCR);
    if !done then
        inf1 = (type1 == FPType_Infinity);
        inf2 = (type2 == FPType_Infinity);
        zero1 = (type1 == FPType_Zero);
        zero2 = (type2 == FPType_Zero);
        if (inf1 && zero2) || (zero1 && inf2) then
            result = FPOnePointFive('0');
        elsif inf1 || inf2 then
            result = FPInfinity(sign1 EOR sign2, N);
        else
            // Fully fused multiply-add and halve
            result_value = (3.0 + (value1 * value2)) / 2.0;
            if result_value == 0.0 then
                // Sign of exact zero result depends on rounding mode
                sign = if FPCRRounding() == FPRounding_NEGINF then '1' else '0';
                result = FPZero(sign, N);
            else
                result = FPRound(result_value, FPCRRounding());
    return result;
```


arm



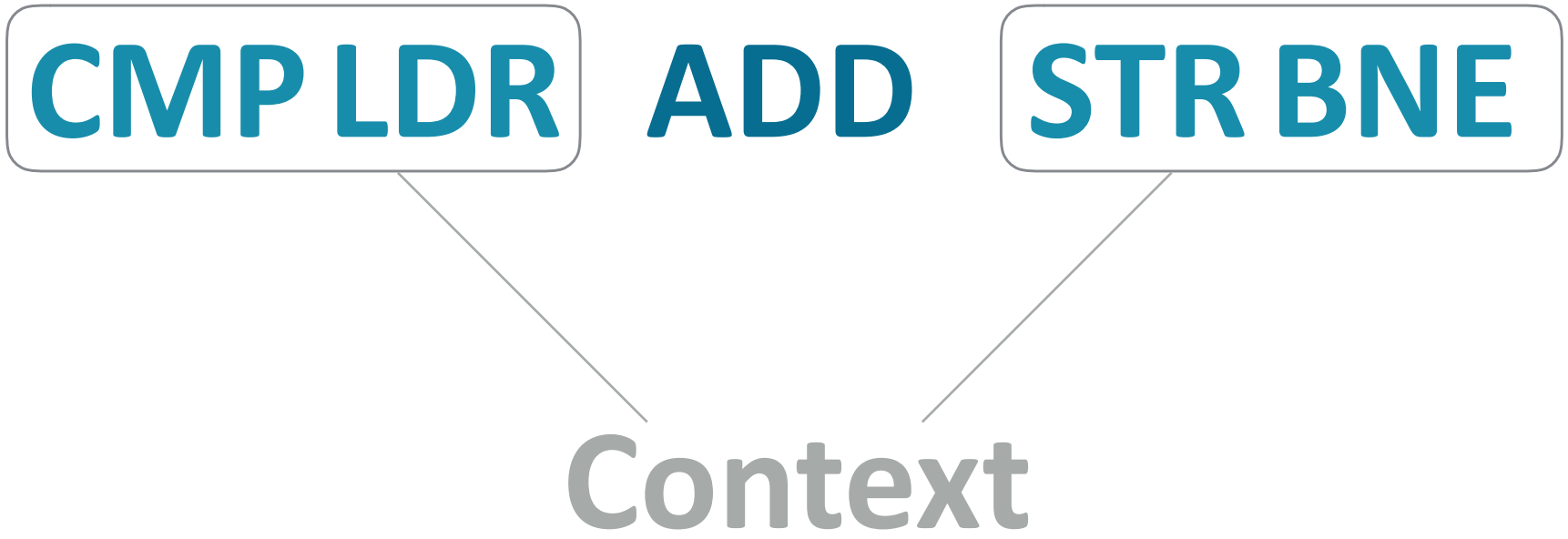
arm

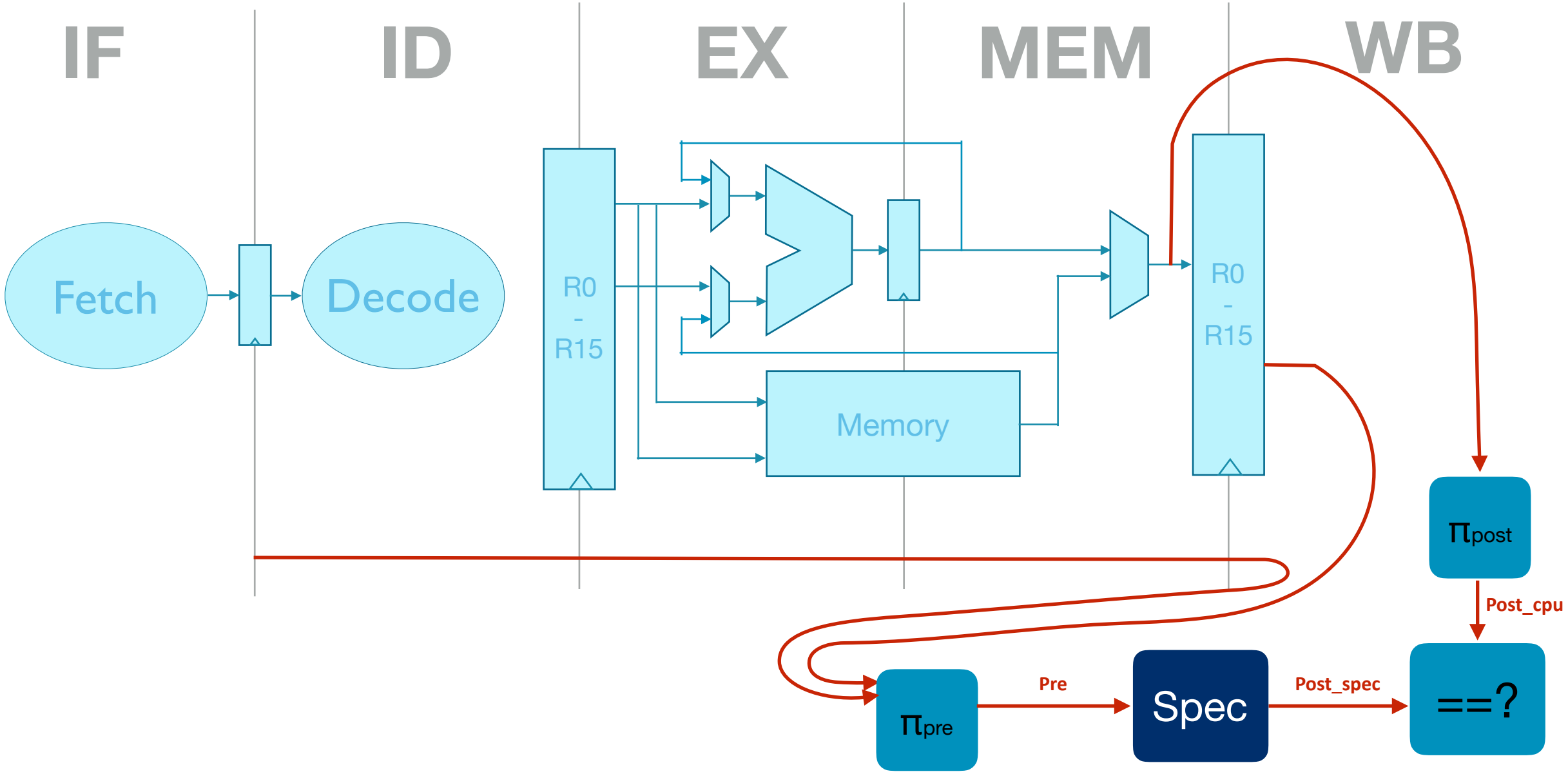
"End to End Verification of ARM processors with ISA Formal," CAV 2016

Formal verification of processors

arm

Checking an instruction





arm

Specialize
Monomorphize
Constant Propagation
Width Analysis
Exception Handling
…

Arm CPUs verified with ISA-Formal

A-class	R-class	M-class
Cortex-A53	Cortex-R52	Cortex-M4
Cortex-A32	Next generation	Cortex-M7
Cortex-A35		Cortex-M33
Cortex-A55		Next generation
Next generation		

Cambridge Projects

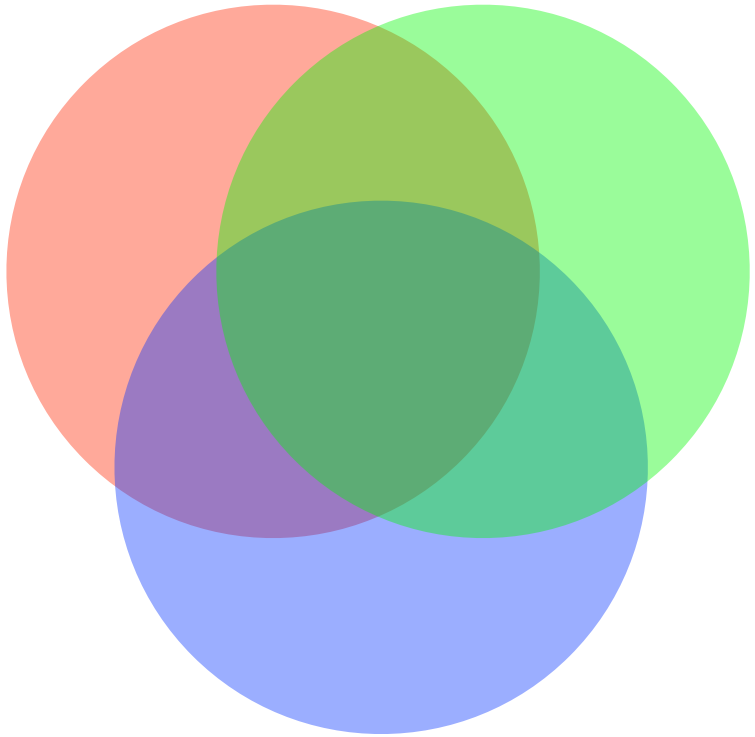
Rolling out globally to other design centres

Sophia, France - Cortex-A75 (partial)

Austin, USA - TBA

Chandler, USA - TBA

arm



arm

Formal validation of specifications

arm

Suppose…

Last year: audited all accesses to privileged registers

- Specification: Added missing privilege checks
- Testsuite: Added new tests to test every privilege check
- Formal testbench: Verify every check

This year: add new instruction but accidentally omit privilege check

How many tests in the test suite will fail on new specification?

Can we formally verify specification?

Specification of the specification

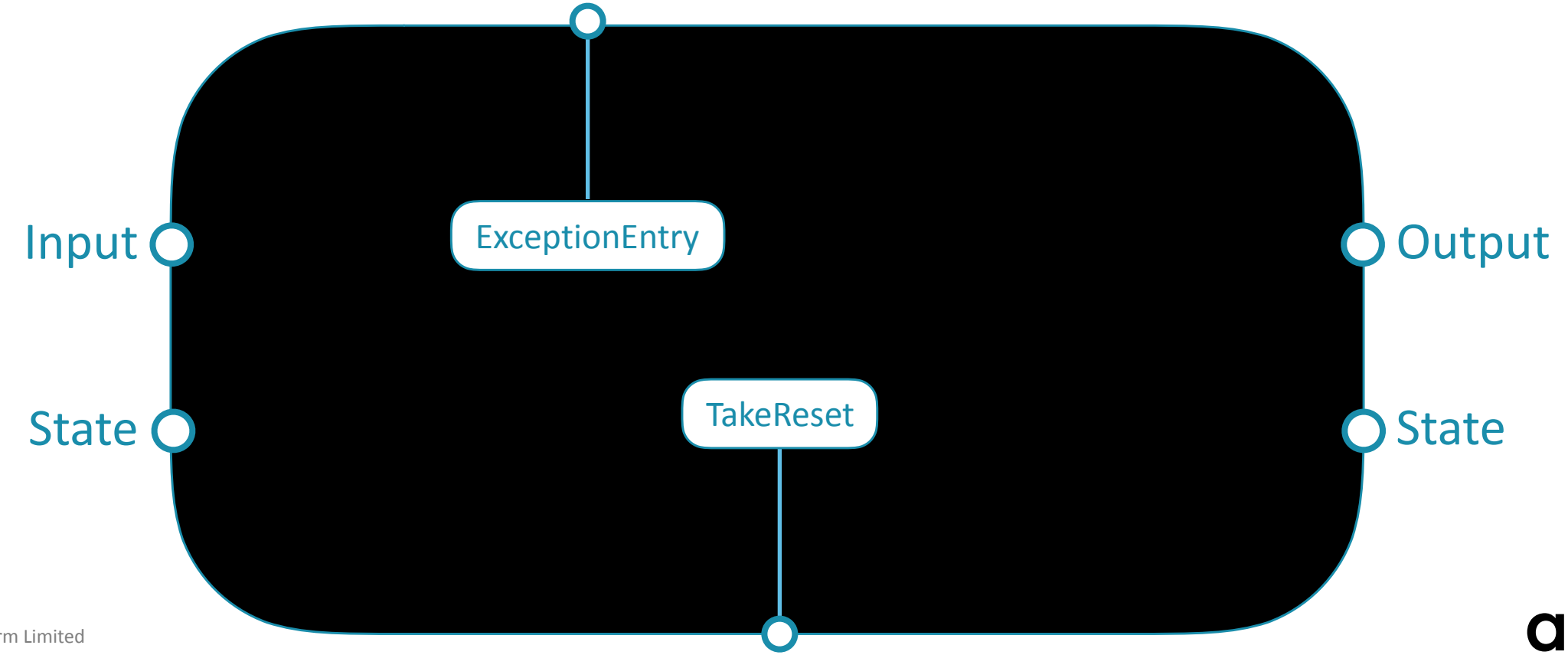
- Disallowed behaviour
- Invariants
- Cross-cutting properties

Tools that can prove properties of ASL specifications

R_{JRJC}

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.



arm

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(InDebugState())

∨ Called(ExceptionEntry);

arm

Converting ASL to SMT

Functions
Local Variables
Statements
    Assignments
    If-statements
Exceptions
Arithmetic operations
Boolean operations
Bit Vectors
Arrays

~~Functions~~
~~Local Variables~~
~~Statements~~
    ~~Assignments~~
    ~~If-statements~~
~~Exceptions~~
Arithmetic operations
Boolean operations
Bit Vectors
Arrays

arm

Formally Validating Specifications

arm

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.

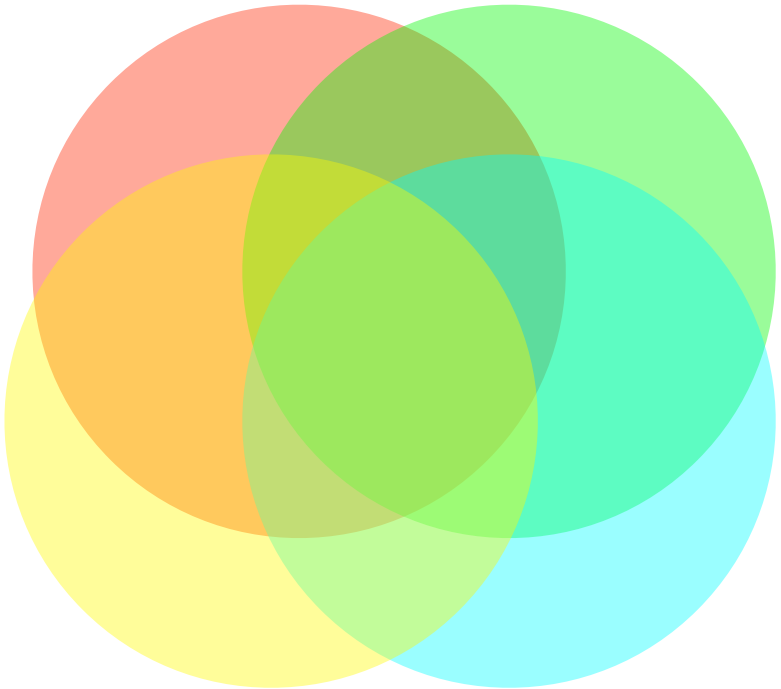
In addition, HFSR.FORCED ~~is not set to 1.~~ is not changed.

Debug view of

```
rule lockup entry
    assume Rose(LockedUp);
    assume ¬Called(TakeReset);

    property a    HaveMainExt() ⇒ CFSR != 0;
    property b1   Stable(ExnPending);
    property b2   Stable(ExnActive);
    property c    PC == 0xEFFFFFFE;
    property e    HFSR.FORCED == 0;
                  Stable(HFSR.FORCED);
```


arm



arm

Public release of machine readable Arm specification

Enable formal verification of software and tools

Releases

- April 2017: v8.2
- July 2017: v8.3

Working with Cambridge University REMS group to convert to SAIL

- Backends for HOL, OCaml, Memory model, (Coq just started)

Tools: https://github.com/alastairreid/mra_tools

Talk to me about how I can help you use it

arm

How can you trust formally verified software?

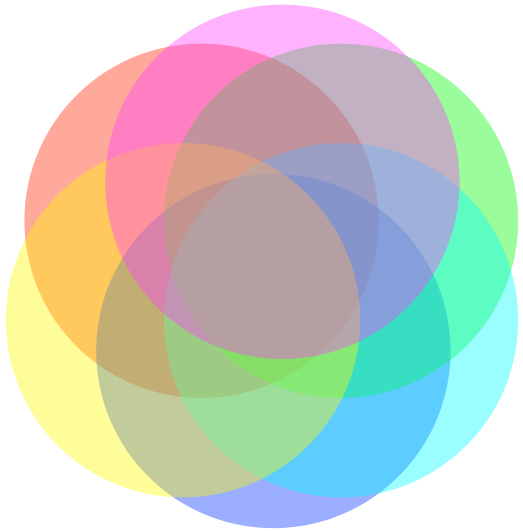
arm

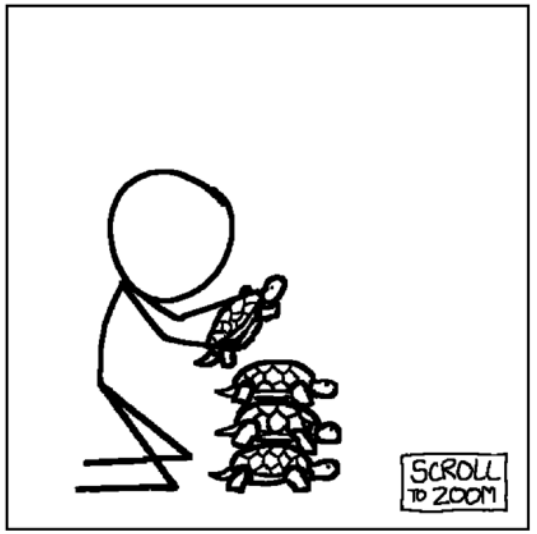
How can you trust formal specifications?

Test the specifications you depend on

Ensure specifications have multiple uses

Create meta-specifications





https://xkcd.com/1416/

arm

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!

@alastair_d_reid

arm

“Trustworthy Specifications of the ARM v8-A and v8-M architecture,” FMCAD 2016
“End to End Verification of ARM processors with ISA Formal,” CAV 2016
“Who guards the guards? Formal Validation of ARM v8-M Specifications” OOPSLA 2017