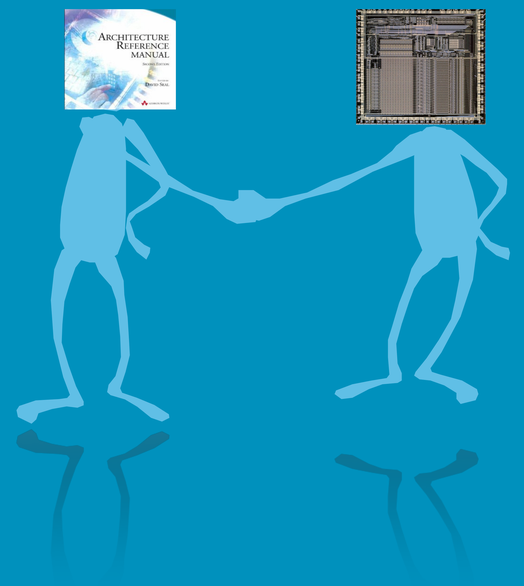


Engineering and Use of Large Formal Specifications

Alastair Reid

Arm Research

@alastair_d_reid

More

Data

Performance

Machine Learning

Internet of Things

Smart Homes

Self Driving Cars

Social Media

Less

Bugs

Crashes

Data loss

Data corruption

Data leaks / theft

DDoS attacks

Cyber-Physical attacks

arm

Better Programming Languages

Hardware Security Enforcement

Exploit Detection

Better System Design

Formal Verification

Automatic Test Generation

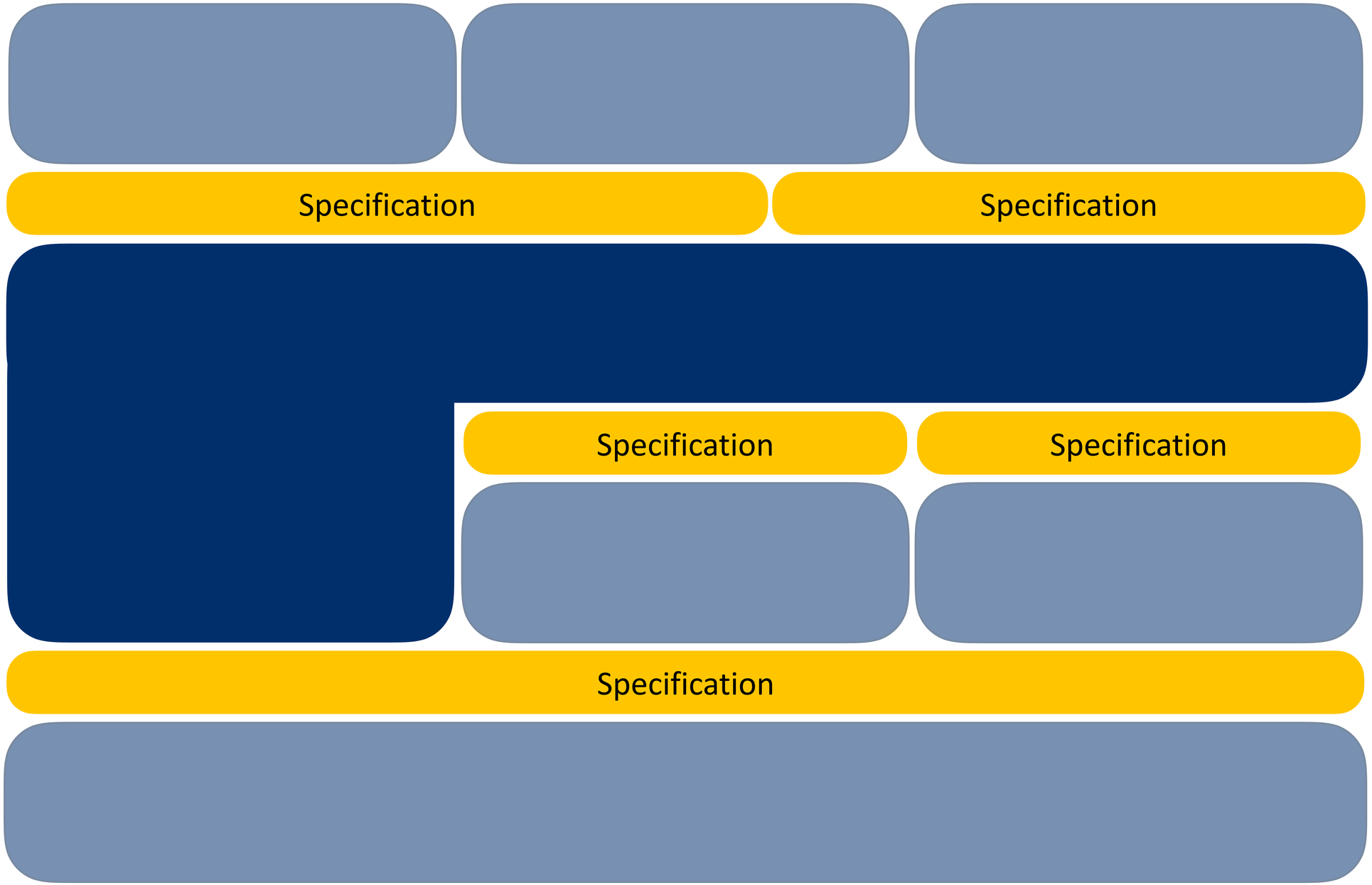
Better Bug Finding

Legal / Regulatory

Fuzz Testing

arm

arm



arm

What (formal) specifications do we need?

Libraries: stdio.h, OpenGL, …

Languages: C, C++, ML, Javascript, Verilog, …

Network: TCP/IP, OAuth, DNS, TLS, WiFi, …

Filesystems: FAT32, NTFS, ext4, …

OSes: Posix/Linux system call, Linux device driver, KVM, UEFI, …

Hardware: CPU, PCIe, AMBA, NIC, …

Critical properties of specifications

Scope

- Completeness
- Not abstracting out critical detail

Applicability

- Version agnostic
- Vendor agnostic

Trustworthiness

Overcoming the Specification Bottleneck

Creating formal specifications

Testing specifications

Getting buy in

Using specifications

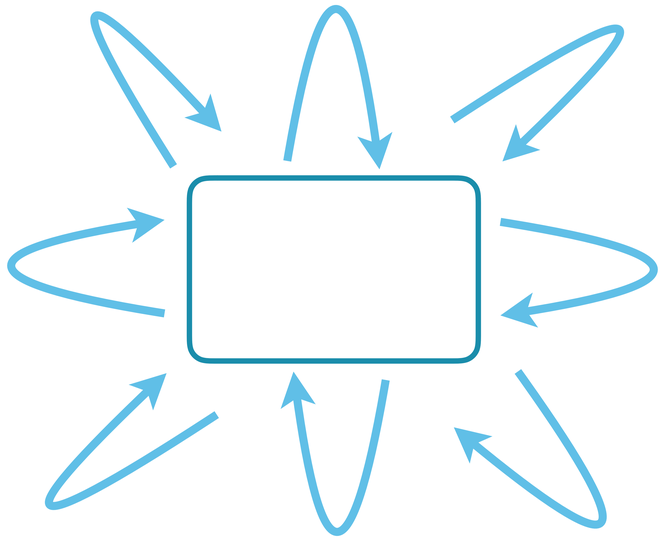
Formal validation of specifications

Making your specifications public

"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
"ISA Semantics for ARM v8-A, , RISC-V, and CHERI-MIPS," POPL 2019
https://alastairreid.github.io/papers/

arm

Creating formal specifications
Testing specifications
Getting buy in



"Trustworthy Specifications of the ARM v8-A and v8-M architecture," FMCAD 2016

Creating Specifications

Concurrent modification and execution of instructions

The ARMv8 architecture limits the set of instructions that can be executed by one thread of execution as they are being modified by another thread of execution without requiring explicit synchronization.

Concurrent modification and execution of instructions can lead to the resulting instruction performing any behavior that can be achieved by executing any sequence of instructions that can be executed from the same Exception level, except where each of the instruction before modification and the instruction after modification is one of a `B`, `BL`, `BRK`, `HVC`, `ISB`, `NOP`, `SMC`, or `SVC` instruction.

For the `B`, `BL`, `BRK`, `HVC`, `ISB`, `NOP`, `SMC`, and `SVC` instructions the architecture guarantees that, after modification of the instruction, behavior is consistent with execution of either:

- The instruction originally fetched.
- A fetch of the modified instruction.

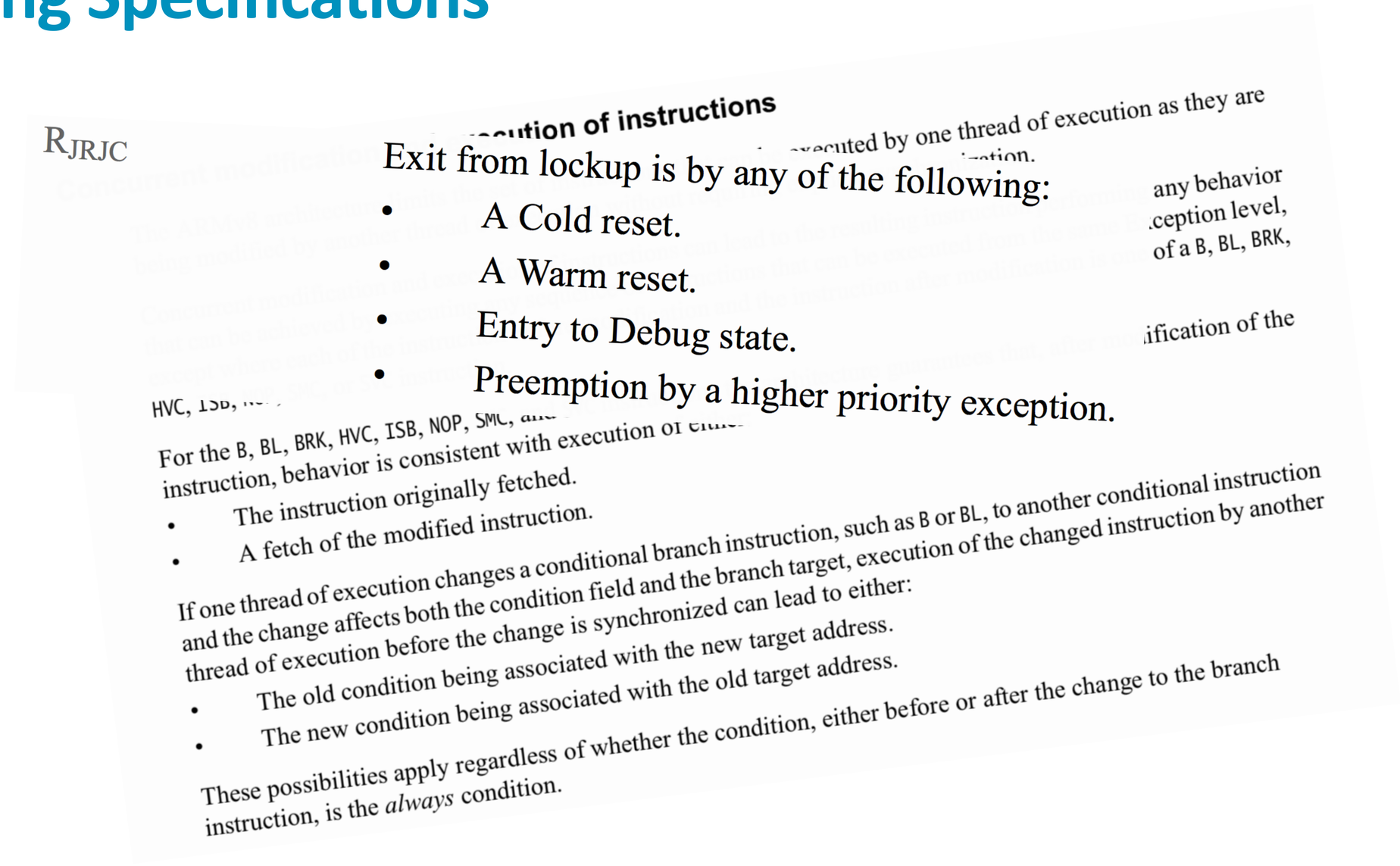
If one thread of execution changes a conditional branch instruction, such as `B` or `BL`, to another conditional instruction and the change affects both the condition field and the branch target, execution of the changed instruction by another thread of execution before the change is synchronized can lead to either:

- The old condition being associated with the new target address.
- The new condition being associated with the old target address.

These possibilities apply regardless of whether the condition, either before or after the change to the branch instruction, is the *always* condition.

arm

Creating Specifications



RJRJC

ecution of instructions

executed by one thread of execution as they are

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.

any behavior
ception level,
of a B, BL, BRK,

ification of the

HVC, ISB,

For the B, BL, BRK, HVC, ISB, NOP, SMC,
instruction, behavior is consistent with execution of

- The instruction originally fetched.
- A fetch of the modified instruction.

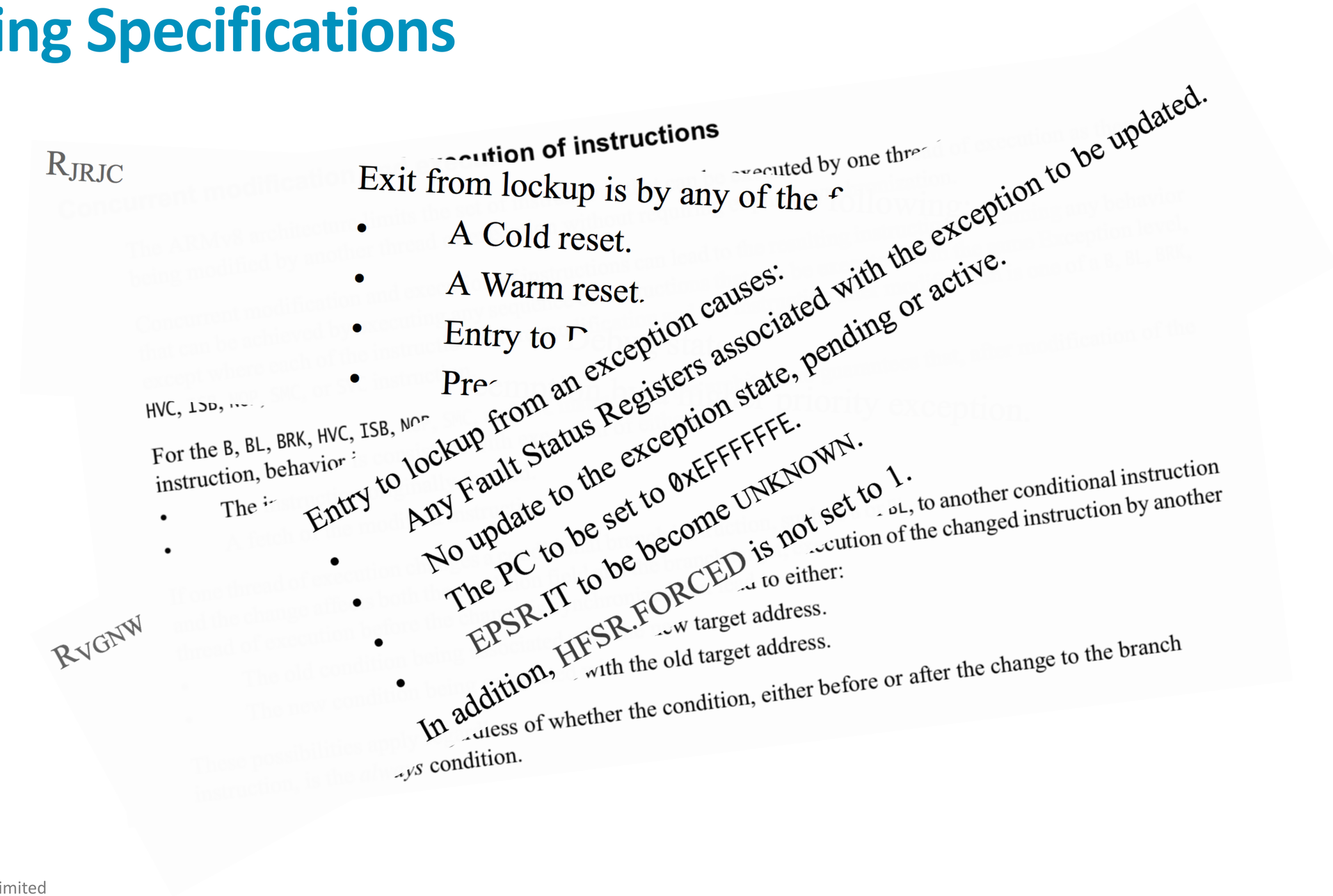
If one thread of execution changes a conditional branch instruction, such as B or BL, to another conditional instruction and the change affects both the condition field and the branch target, execution of the changed instruction by another thread of execution before the change is synchronized can lead to either:

- The old condition being associated with the new target address.
- The new condition being associated with the old target address.

These possibilities apply regardless of whether the condition, either before or after the change to the branch instruction, is the *always* condition.

arm

Creating Specifications



arm

Creating Specifications

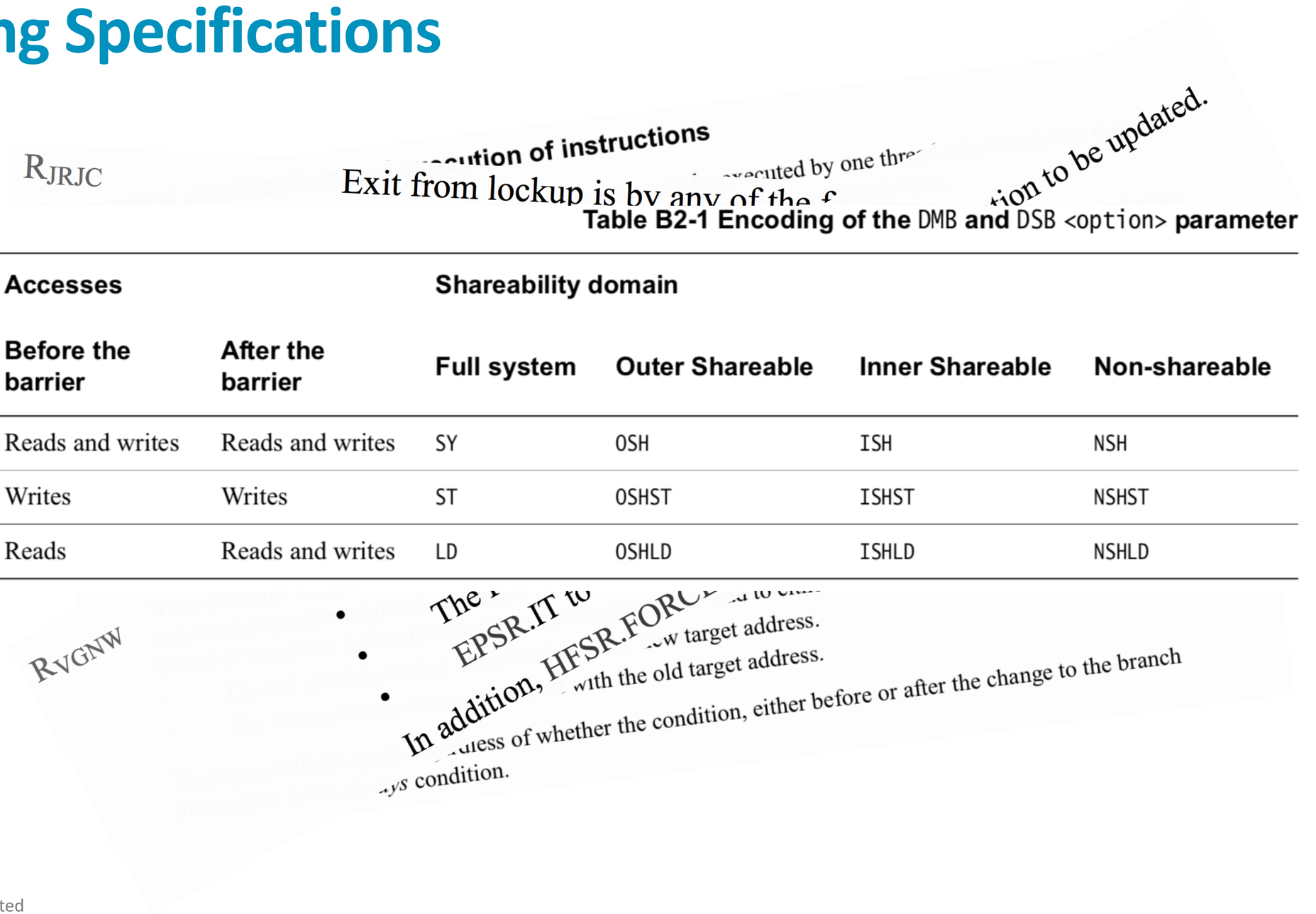
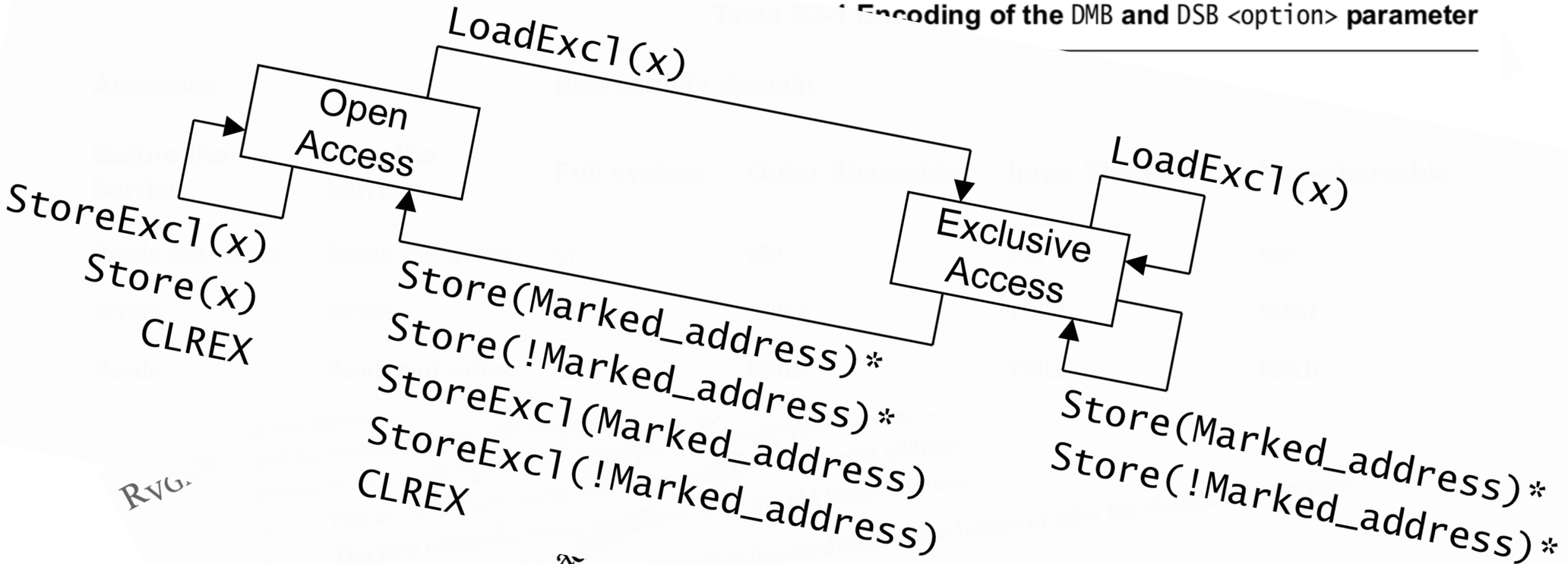


Table B2-1 Encoding of the DMB and DSB <option> parameter

Accesses		Shareability domain			
Before the barrier	**After the barrier**	**Full system**	**Outer Shareable**	**Inner Shareable**	**Non-shareable**
Reads and writes	Reads and writes	SY	OSH	ISH	NSH
Writes	Writes	ST	OSHST	ISHST	NSHST
Reads	Reads and writes	LD	OSHLD	ISHLD	NSHLD

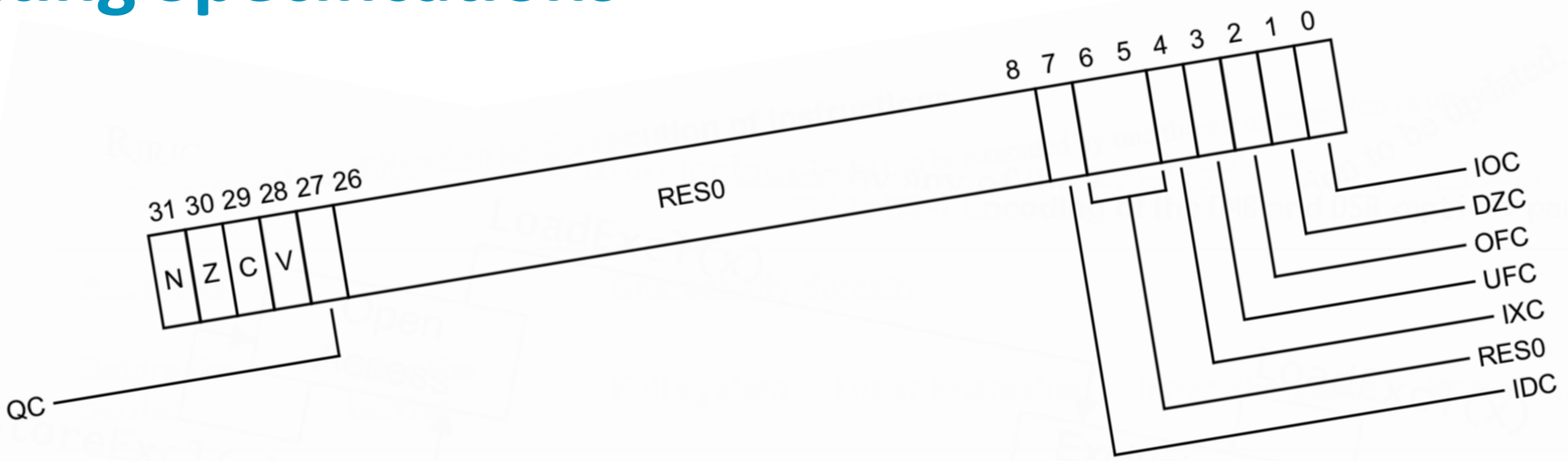
arm

Creating Specifications



arm

Creating Specifications



neter

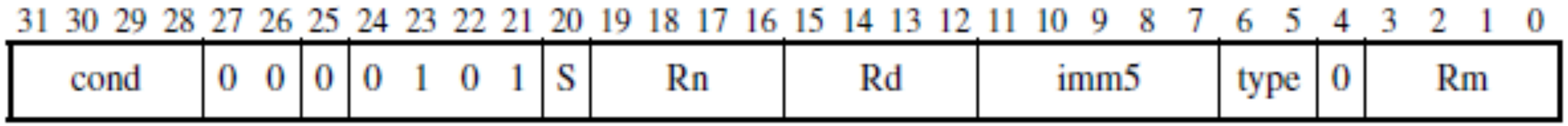
N, bit [31] Negative condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.N flag instead.

Z, bit [30] Zero condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.Z flag instead.

arm

Pseudocode

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




```
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 Pseudocode

~40,000 lines

- 32-bit and 64-bit modes
- All 4 encodings: Thumb16, Thumb32, ARM32, ARM64
- All instructions (> 1300 encodings)
- All 4 privilege levels (User, Supervisor, Hypervisor, Secure Monitor)
- Both Security modes (Secure / NonSecure)
- MMU, Exceptions, Interrupts, Privilege checks, Debug, TrustZone, …

Status at the start

- No language spec
- No tools (parser, type checker)
- Incomplete (around 15% missing)
- Unexecuted, untested
- Senior architects believed that an executable spec was
  - Impossible
  - Not useful
  - Less readable
  - Less correct

Architectural Conformance Suite

Processor architectural compliance sign-off

Large

- v8-A 32,000 test programs, billions of instructions
- v8-M 3,500 test programs, > 250 million instructions

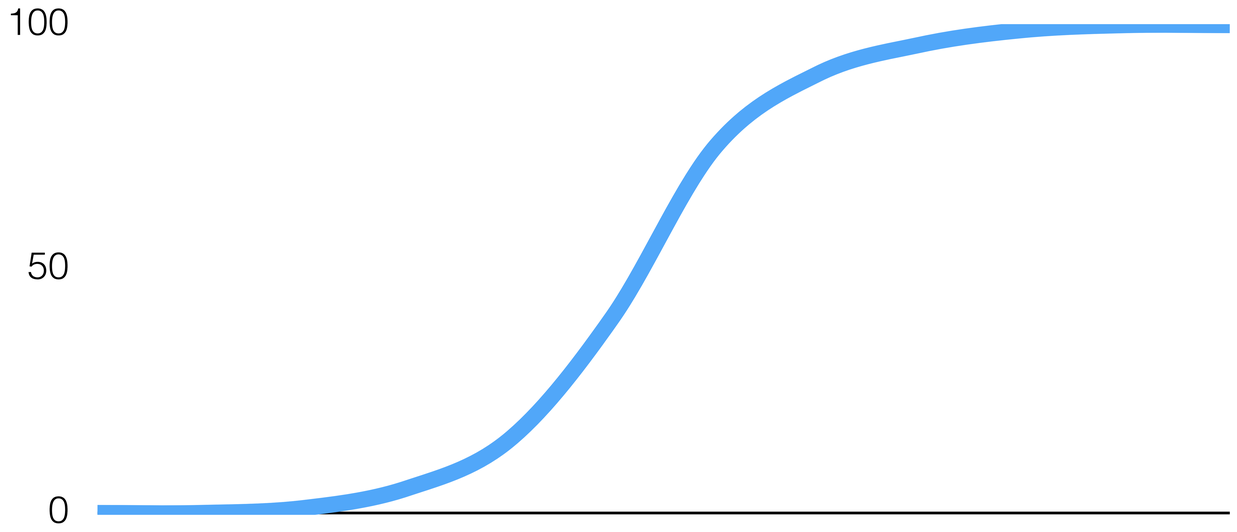
Thorough

- Tests dark corners of specification

Hard to run

- Requires additional testing infrastructure

Progress in testing Arm specification



- Does not parse, does not typecheck
- Can’t get out of reset
- Can’t execute first instruction
- Can’t execute first 100 instructions
- …
- Passes 90% of tests
- Passes 99% of tests
- …

arm

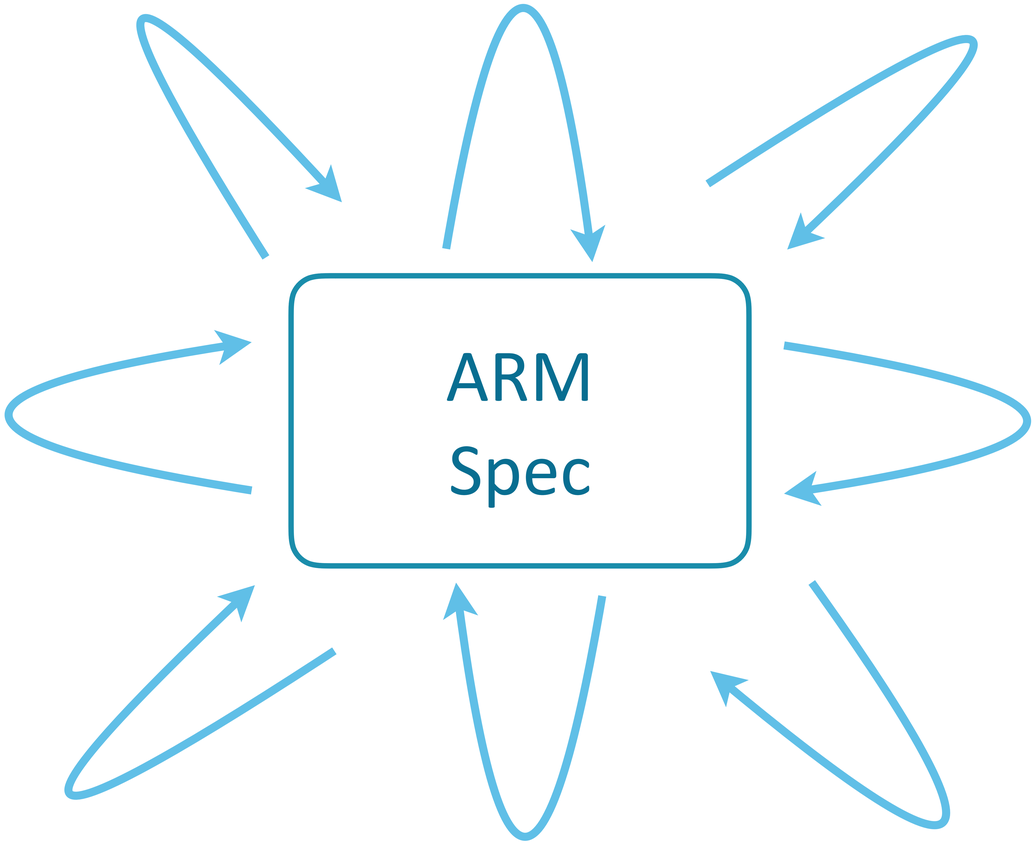
Measuring architecture coverage of tests

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

Creating a Virtuous Cycle



arm

Lessons learned about engineering a specification

Specifications contain bugs

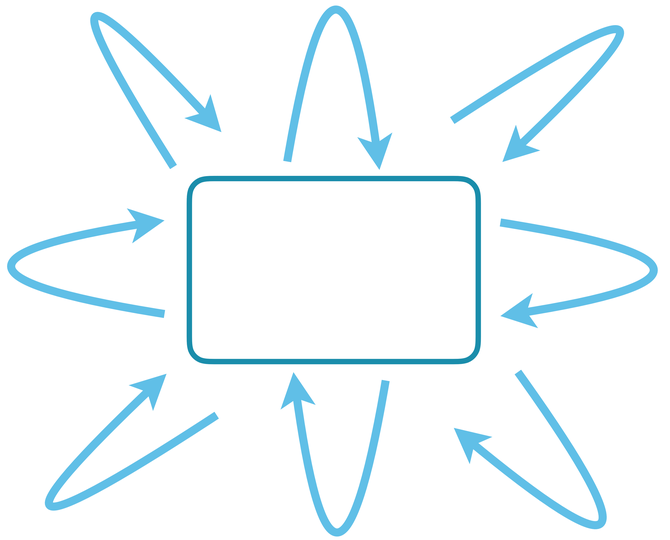
Huge value in being able to run existing test suites

- Need to balance against benefits of non-executable specs

Find ways to provide direct benefit to other users of spec

- They will do some of the testing/debugging for you
- They will support getting your changes/spec adopted as master spec
- Creates Virtuous Cycle

Using Specifications



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

Documentation

- **Generate PDF/HTML**
- Interactive specifications

Verification of Implementations

- **Bounded Model Checking**
- **Testing (Golden Reference)**
- Deductive Reasoning

Specification Extension

- **Testing / Exploration**

Instrumented Execution

- **Measure Coverage**
- **Driving Fuzz Testing**

Generation

- **Testsuites (Concolic)**
- **Simulators**
- Peephole Optimisations
- Binary Translators

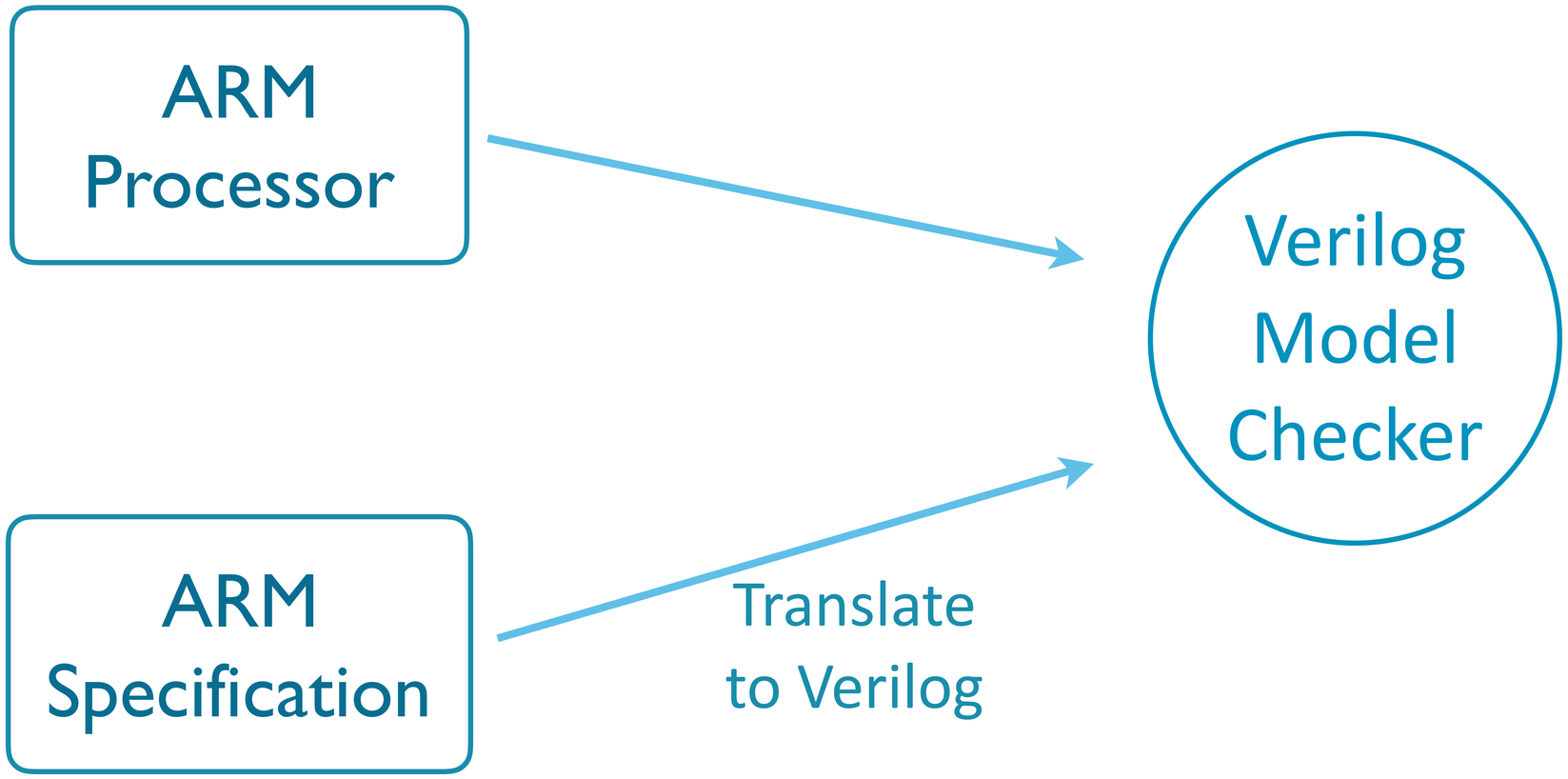
Verification of Clients

- Formally verifying OS code / etc.
- Verifying Compilers/Linkers

Static Analysis

- Abstract interpretation of binaries
- Decompilation of binaries
- Reverse engineering tools

Formally validating ARM processors - using an existing tool

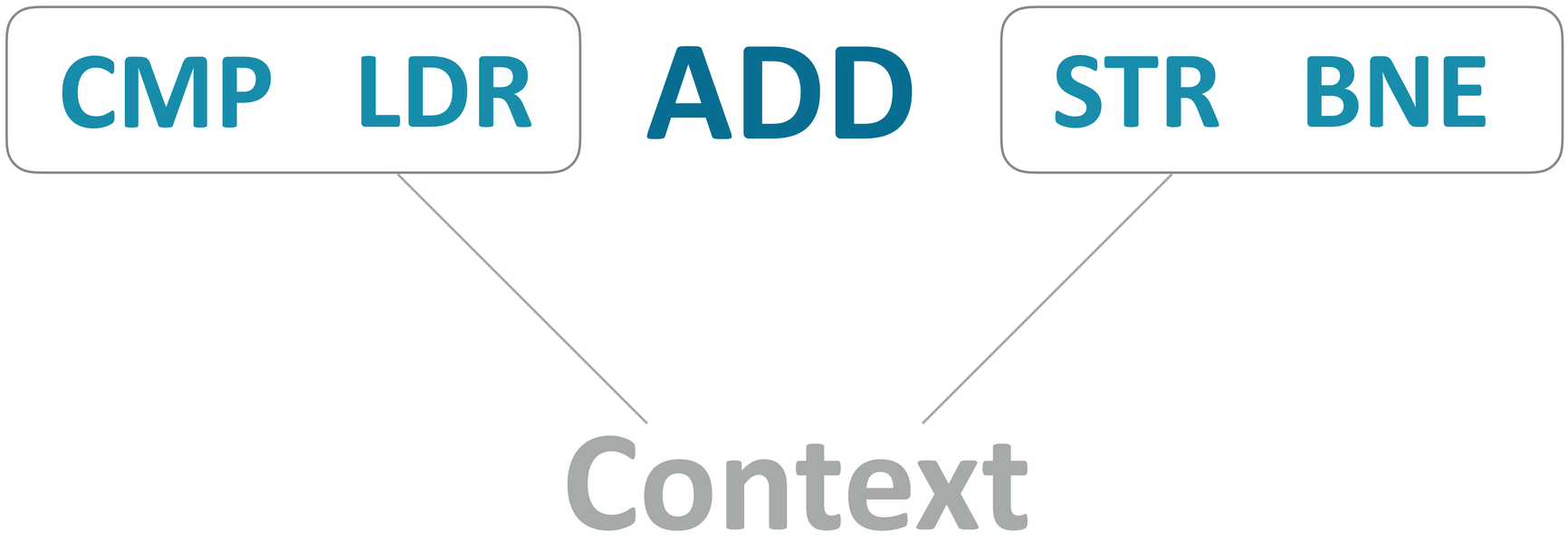


arm

Checking an instruction

ADD

Checking an instruction



arm

Lessons Learned from validating processors

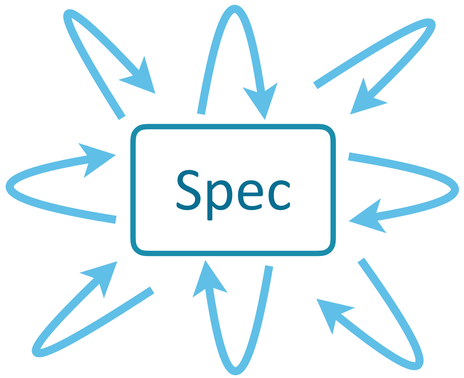
Very effective way to find bugs in implementations

Formally validating implementation is effective at finding bugs in spec

- Try to find most of the bugs in your spec before you start

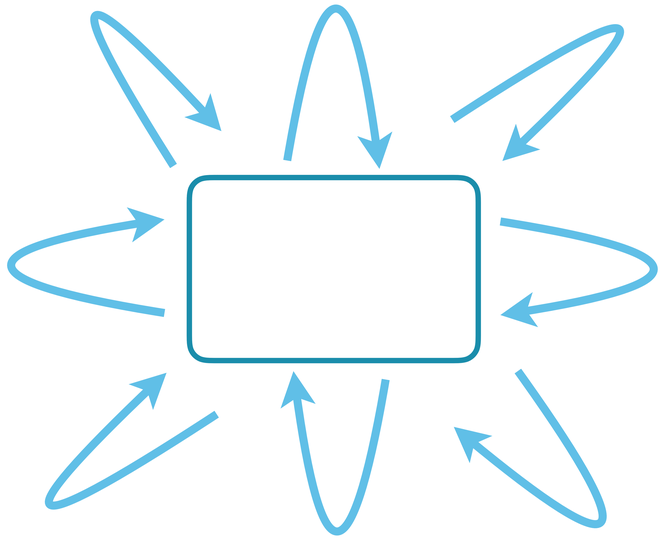
Huge value in being able to use spec to validate implementations

- Helps get formal specification adopted as part of official spec



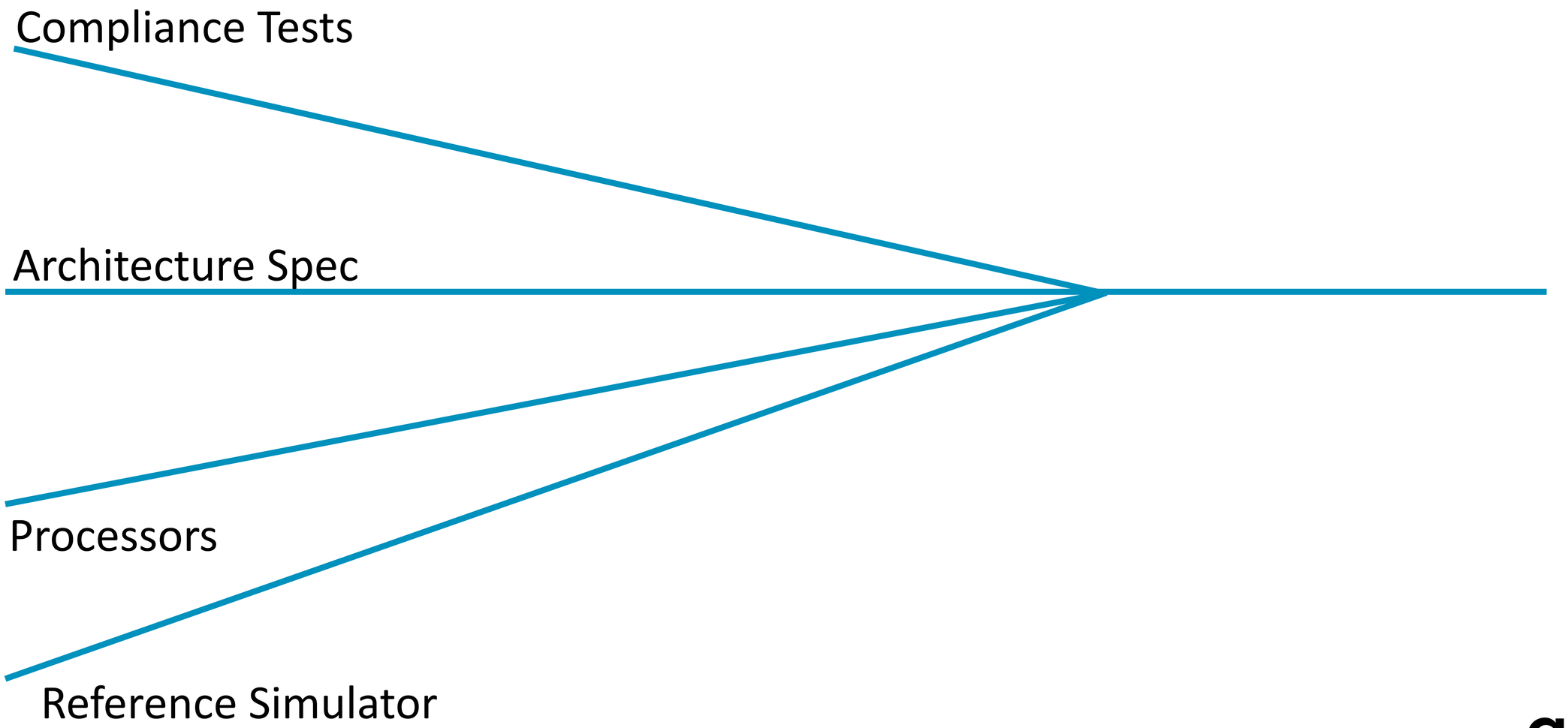
arm

Formal Validation of Specifications



“Who guards the guards? Formal Validation of ARM v8-M Specifications” OOPSLA 2017

One Specification to rule them all?



arm

Rule 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 processor exception.

Rule R

State Change X is by any of the following:

- Event A
- Event B
- State Change C
- Event D

arm

Rule R

State Change X is by any of the following:

- Event A
- Event B
- State Change C
- Event D

Rule R: $X \rightarrow A \vee B \vee C \vee D$

arm

State Change	Exit from lockup	Fell(LockedUp)
Event	A Cold reset	Called(TakeColdReset)
Event	A Warm reset	Called(TakeReset)
State Change	Entry to Debug state	Rose(Halted)
Event	Preemption by a higher priority processor exception	Called(ExceptionEntry)

arm

“Eyeball Closeness”

Rule 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 processor exception.

$$\begin{aligned} Fell(LockedUp) \rightarrow\ & Called(TakeColdReset) \\ \vee\ & Called(TakeReset) \\ \vee\ & Rose(Halted) \\ \vee\ & Called(ExceptionEntry) \end{aligned}$$

Rule VGNW

Entry to lockup from an exception causes

- Any Fault Status Registers associated with the exception to be updated.

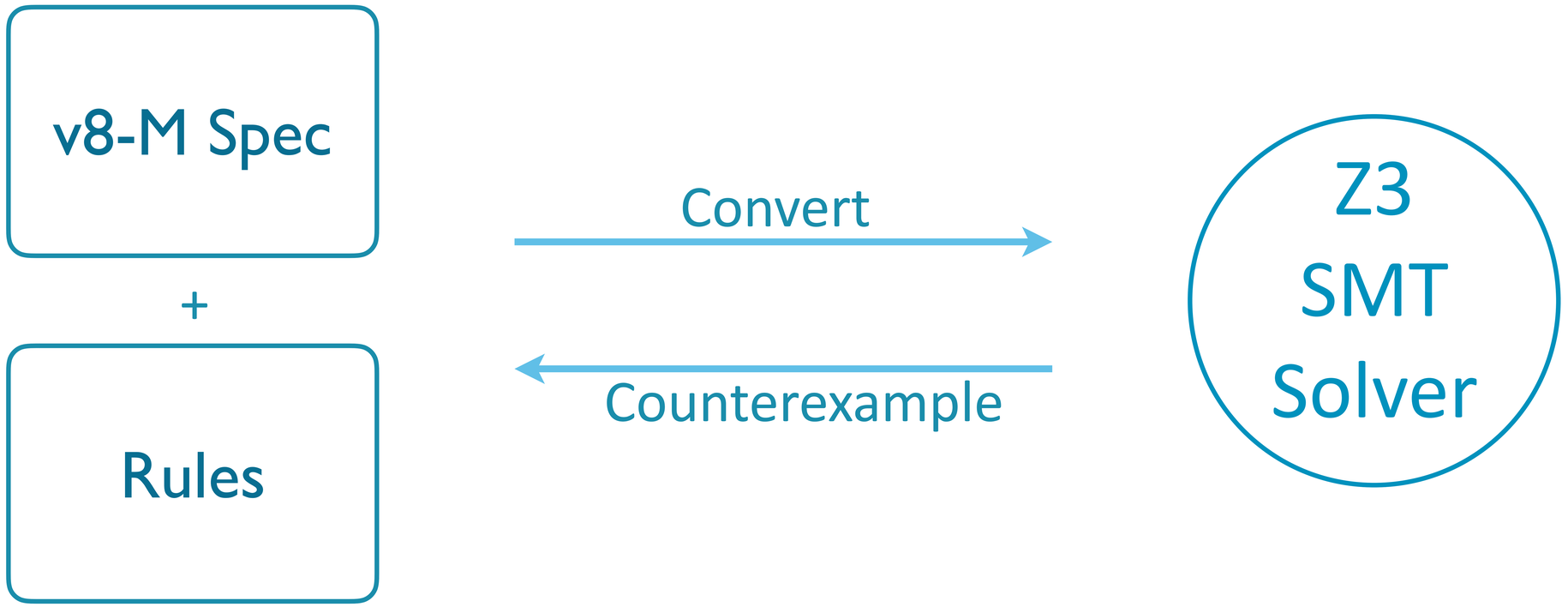
Out of date
- No update to the exception state, pending or active.

Misleading
- The PC to be set to 0xEFFFFFFE.

Untestable
- EPSR.IT to become UNKNOWN.

Ambiguous

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



arm

Lessons Learned from validating specifications

Redundancy essential for detecting errors

- Detected subtle bugs in security, exceptions, debug, …

- Found bugs in English prose

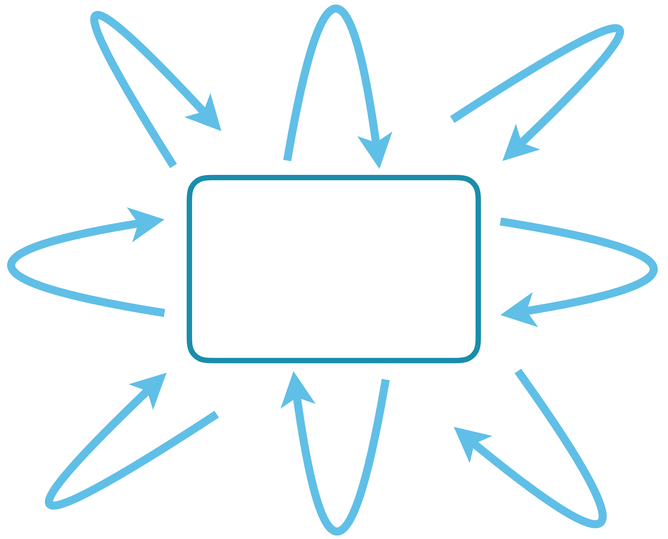
Need set of ‘orthogonal’ properties

- Invariants, Security properties, Reachability properties, etc.

Eyeball closeness

Needed to translate specification to another language to let us use other tools

Making your specification public



Public release of machine readable Arm specification

Enable formal verification of software and tools

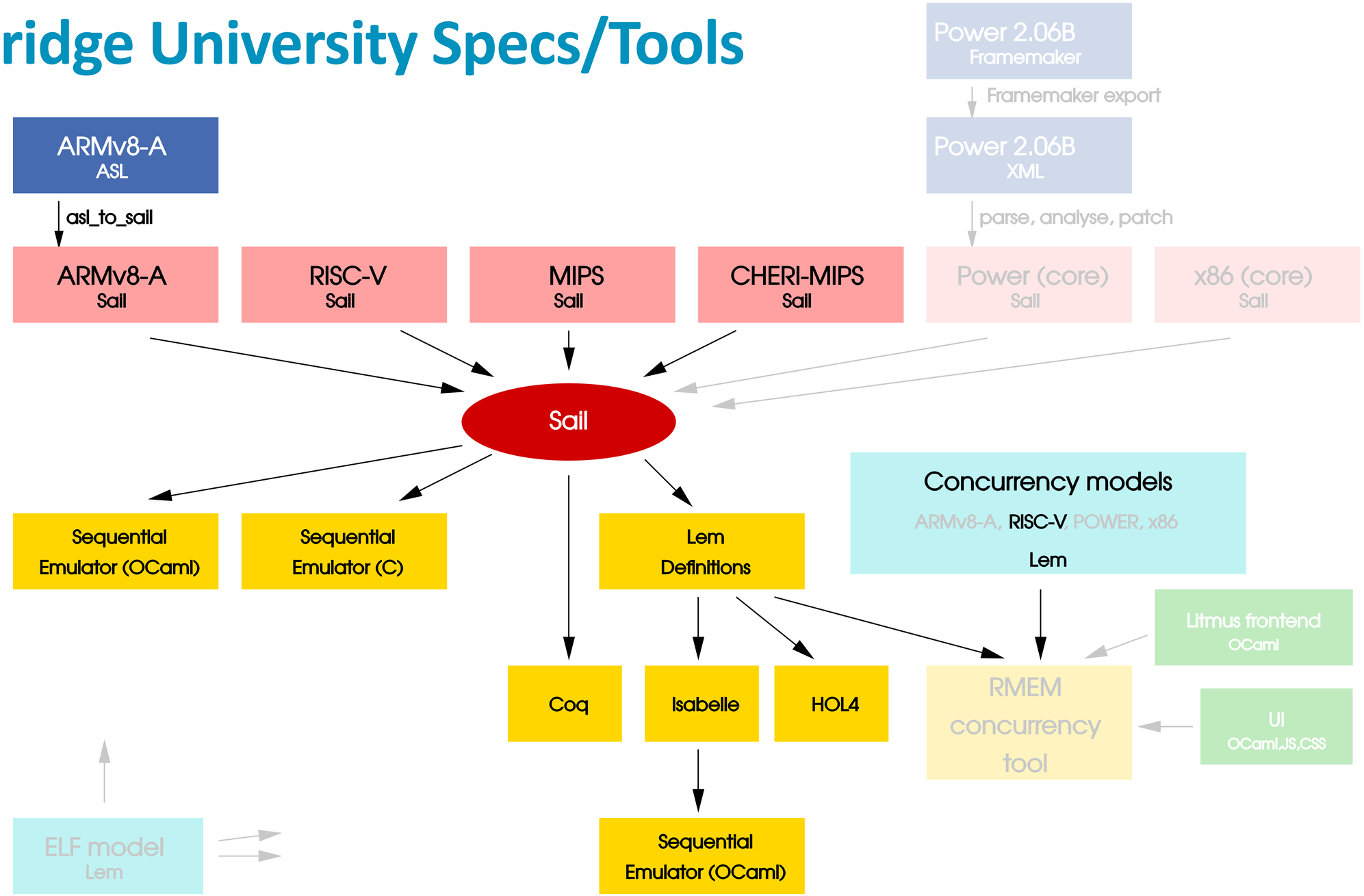
Machine readable

Releases:

- v8.2 (4/2017)
- v8.3 (10/2017)
- v8.4 (6/2018)
- v8.5 (9/2018)

https://developer.arm.com/products/architecture/a-profile/exploration-tools
https://github.com/alastairreid/mra_tools
https://github.com/herd/herdtools7/blob/master/herd/libdir/aarch64.cat

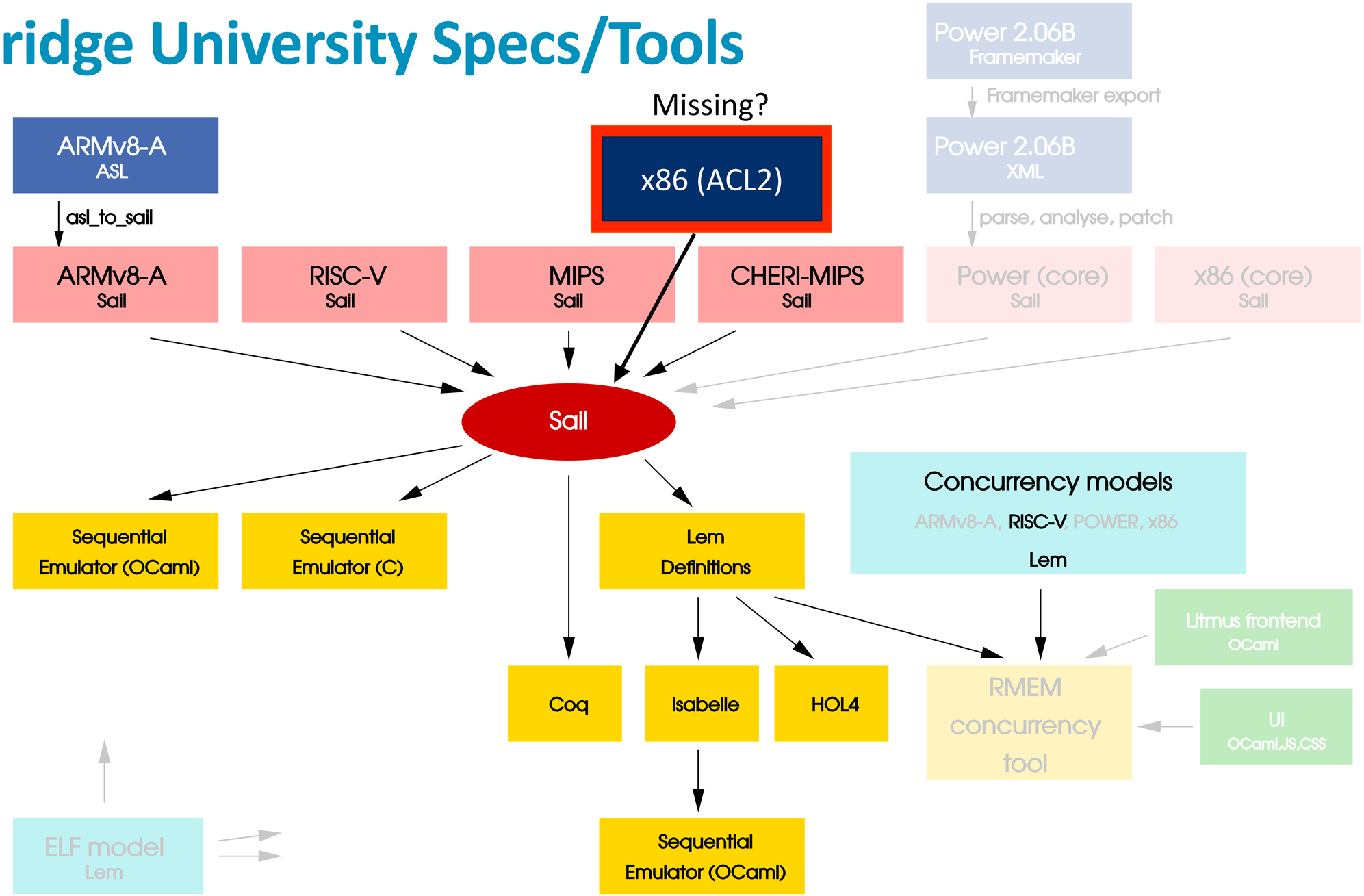
Cambridge University Specs/Tools



From “ISA Semantics for ARM v8-A, , RISC-V, and CHERI-MIPS,” POPL 2019
Used with permission of REMS Group, Cambridge University

arm

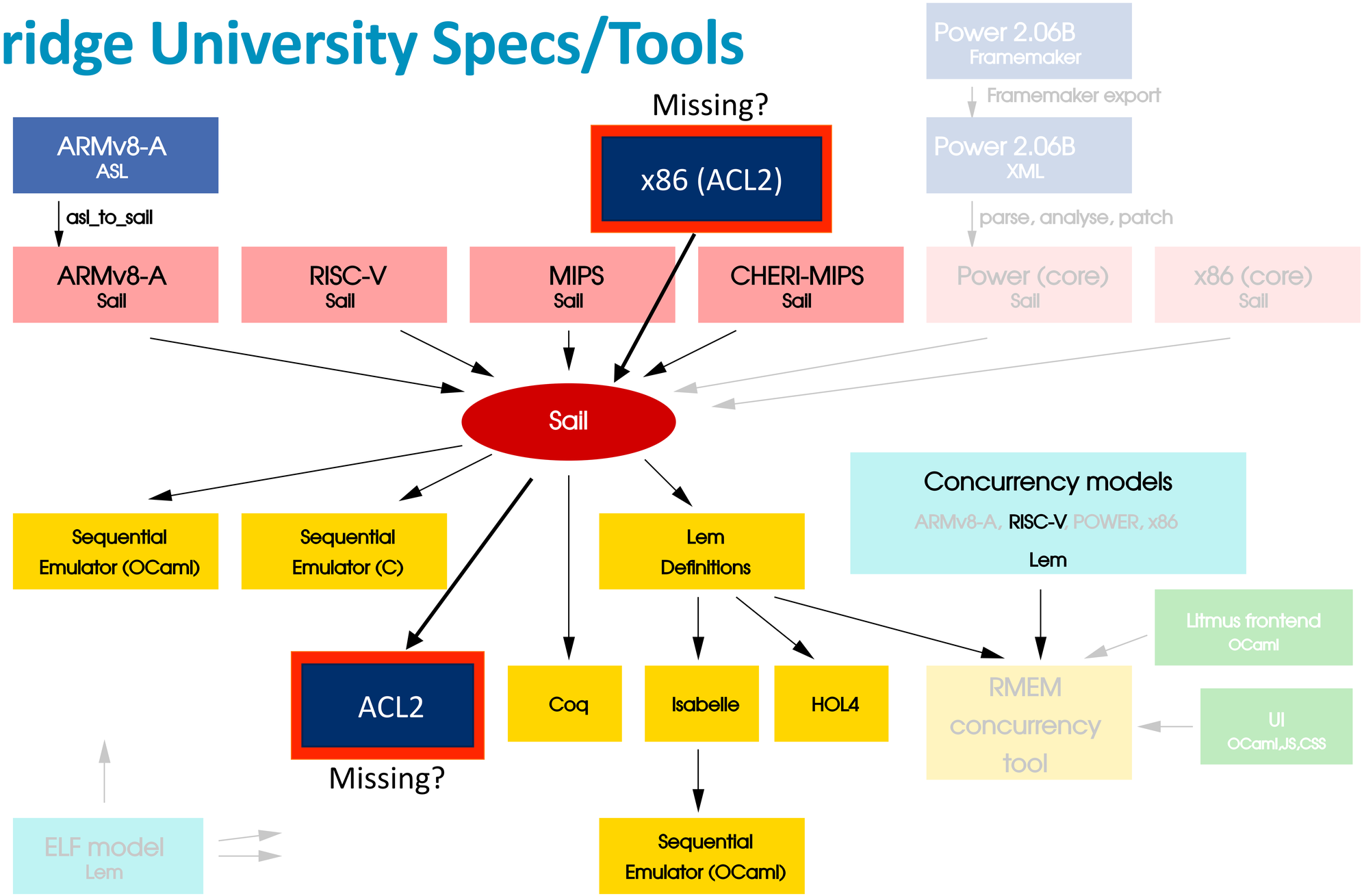
Cambridge University Specs/Tools



From "ISA Semantics for ARM v8-A, , RISC-V, and CHERI-MIPS," POPL 2019
Used with permission of REMS Group, Cambridge University

arm

Cambridge University Specs/Tools

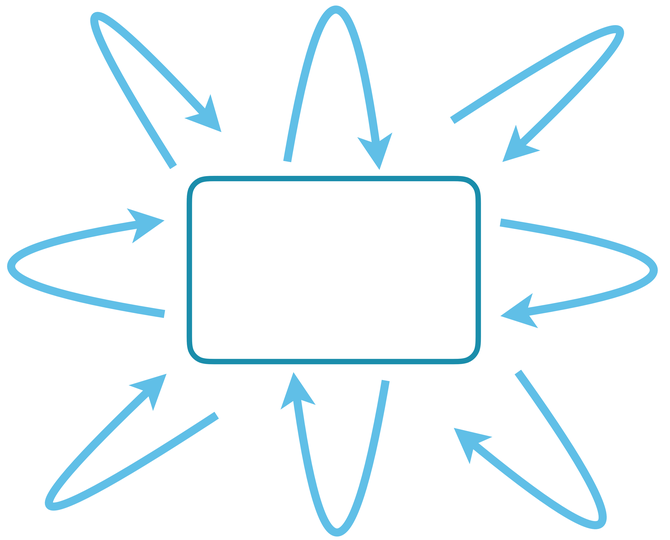


From “ISA Semantics for ARM v8-A, , RISC-V, and CHERI-MIPS,” POPL 2019
Used with permission of REMS Group, Cambridge University

arm

Work in Progress:

Security of Architecture Specifications



Validating security of processor architectures

Scope

- Hardware-based Security Enforcement (HSE) Mechanisms

- Confidentiality, Integrity, Availability

Challenges

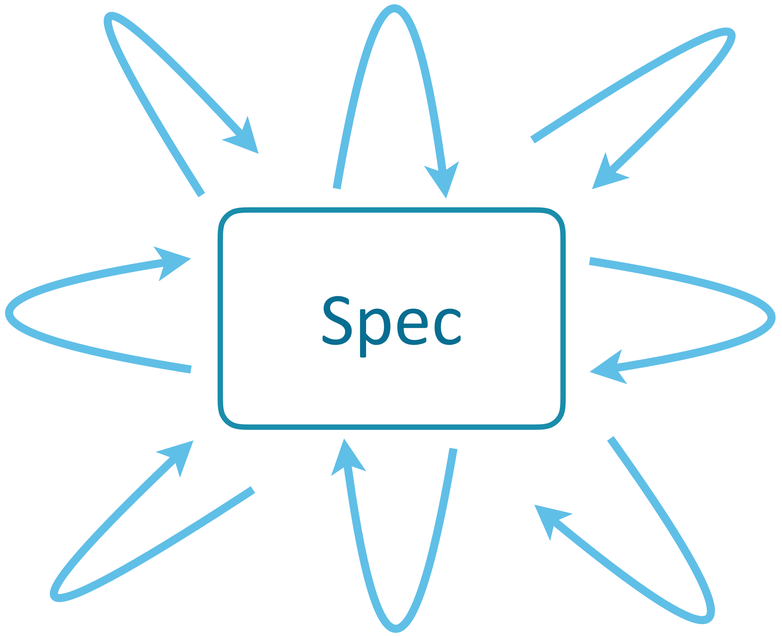
- Compositional Attacks

- Cyclic dependencies between HSEs

- Microarchitectural storage/timing channels

The Specification Bottleneck: Modelling Real World Artifacts

- Trustworthiness, Scope and Applicability
- Significant Engineering Effort
- Importance of sharing specifications across many users



arm

Thanks

Alasdair Armstrong (Cambridge U.)
Alex Chadwick (ARM)
Ali Zaidi (ARM)
Anastasios Deligiannis (ARM)
Anthony Fox (Cambridge U.)
Ashan Pathirane (ARM)
Belaji Venu (ARM)
Bradley Smith (ARM)
Brian Foley (ARM)
Curtis Dunham (ARM)
David Gilday (ARM)
David Hoyes (ARM)
David Seal (ARM)
Daniel Bailey (ARM)
Erin Shepherd (ARM)
Francois Botman (ARM)
George Hawes (ARM)
Graeme Barnes (ARM)
Isobel Hooper (ARM)
Jack Andrews (ARM)
Jacob Eapen (ARM)
Jon French (Cambridge U.)
Kathy Gray (Cambridge U.)
Krassy Gochev (ARM)
Lewis Russell (ARM)
Matthew Leach (ARM)
Meenu Gupta (ARM)
Michele Riga (ARM)
Milosch Meriac (ARM)
Nigel Stephens (ARM)
Niyas Sait (ARM)
Peng Wang (ARM)
Peter Sewell (Cambridge U.)
Peter Vrabel (ARM)
Richard Grisenthwaite (ARM)
Rick Chen (ARM)
Simon Bellew (ARM)
Thomas Grocutt (ARM)
Will Deacon (ARM)
Will Keen (ARM)
Wojciech Meyer (ARM)
(and others)

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
“ISA Semantics for ARM v8-A, , RISC-V, and CHERI-MIPS,” POPL 2019

https://alastairreid.github.io/papers/

Is it better to create specs

- Within ACL2?
- In a DSL translated to ACL2?

arm