Formal for Everyone

Challenges in Achievable Multicore Design and Verification

FMCAD 25 Oct 2012

Daryl Stewart

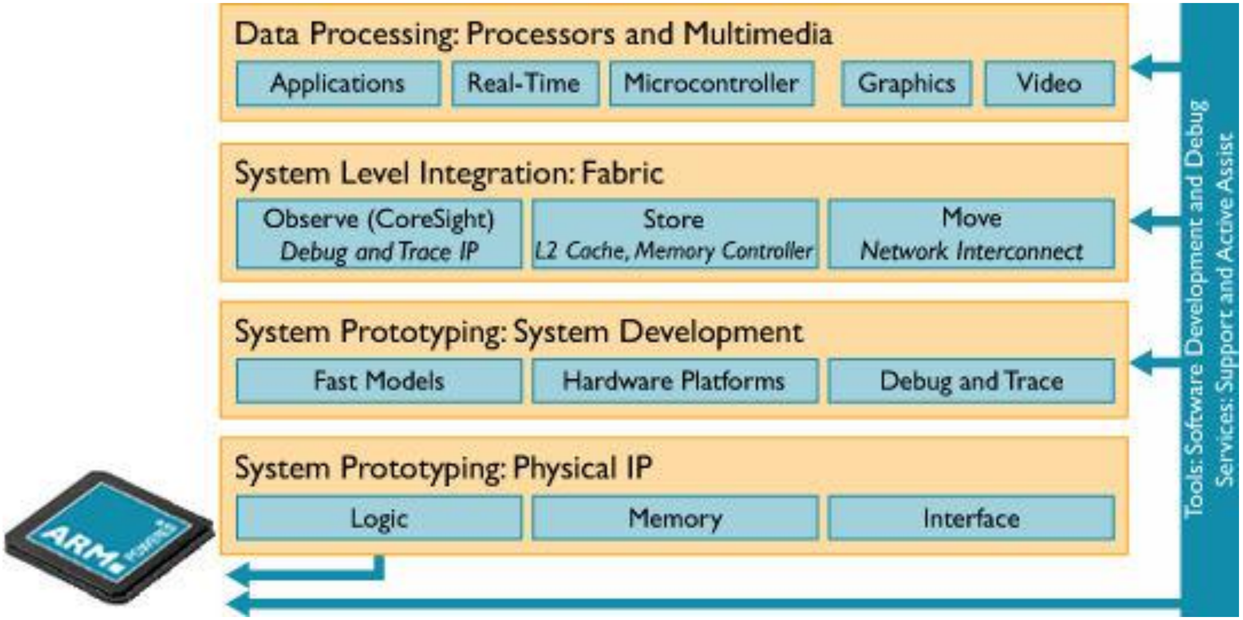
ARM is an IP company

- ARM licenses technology to a network of more than 1000 partner companies within the ARM® Connected Community®, spanning the semiconductor supply chain

- ARM provides developers with intellectual property (IP) solutions in the form of
  - CPUs/GPUs
  - Physical IP
  - Cache and SoC designs
  - Application-specific standard products (ASSPs)
  - Related software and development tools

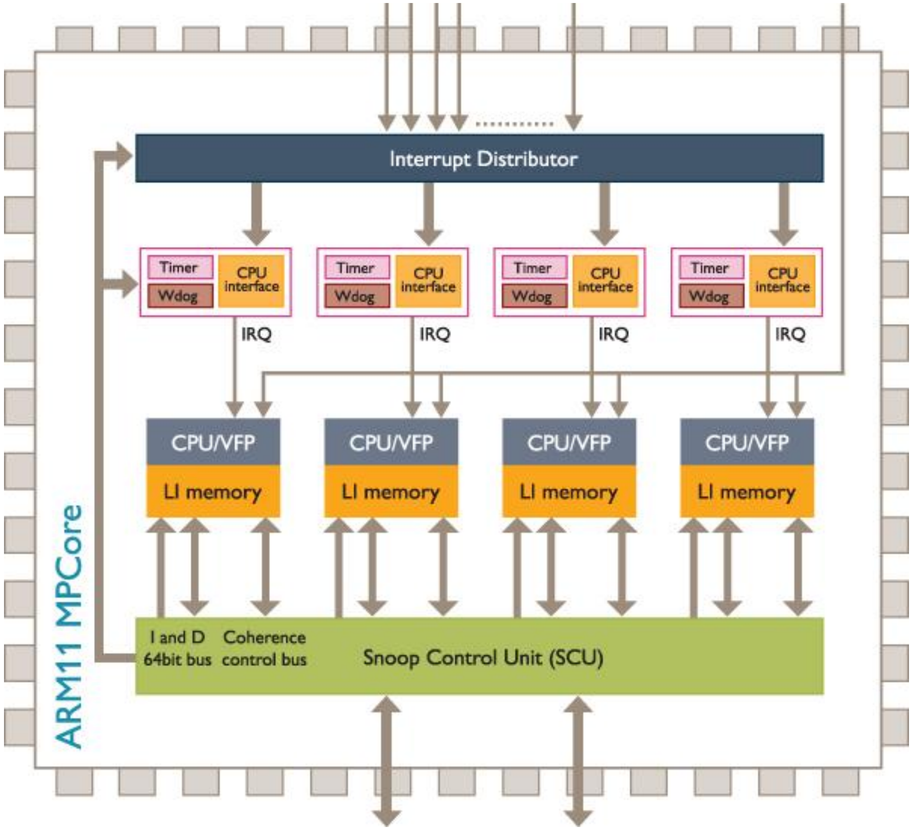
Our Partners Supply the Silicon

- ARM silicon partners supply chips into 90% of smart phones, 80% of digital cameras, and 28% of all electronic devices – over 20 billion chips to date.
- ARM technology is used in a wide variety of applications ranging from mobile handsets and digital set top boxes to car braking systems and network routers.


Data Processing: Processors and Multimedia
Applications
Real-Time
Microcontroller
Graphics
Video
System Level Integration: Fabric
Observe (CoreSight)
Debug and Trace IP
Store
L2 Cache, Memory Controller
Move
Network Interconnect
System Prototyping: System Development
Fast Models
Hardware Platforms
Debug and Trace
System Prototyping: Physical IP
Logic
Memory
Interface
Tools: Software Development and Debug
Services: Support and Active Assist
ARM

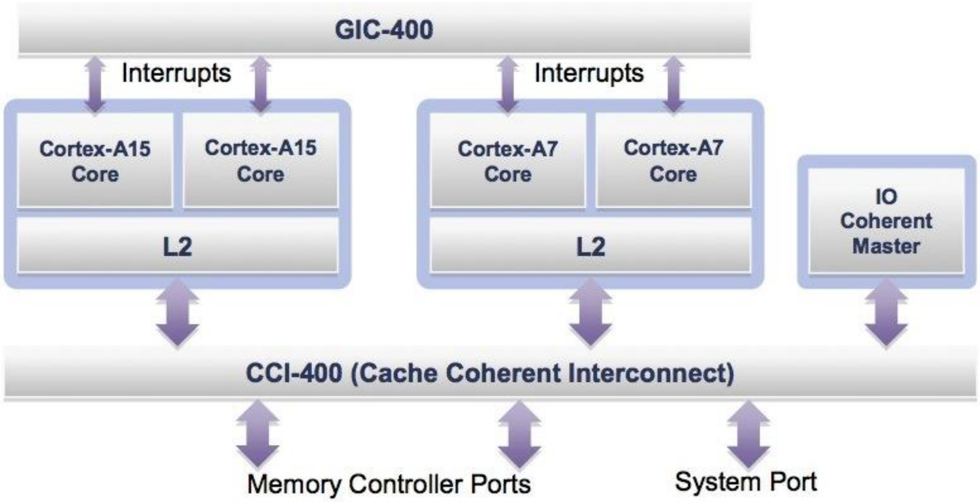
ARM11™ MPCore™ processor

- 800MHz to 1 GHz+ in 65G at under 2 mm^2
- 1 to 4 cores in an SMP cluster
- 32-bit SIMD for media processing
- Physically tagged caches
- Tightly coupled memories
- ARM TrustZone™ security



ARM Cortex™-A Series processors

- Applications processors for mobile computing
- Single to Quad core clusters
  - Fully coherent L1 cache via Snoop Control Unit
  - Accelerator Coherence Port shares cache with peripherals
- Multi cluster coherency with AMBA Coherency Extension
- Heterogeneous system with Cortex-A15/Cortex-A7 processor clusters: “ARM big.LITTLE™ processing”
  - AMBA®4 ACE™ interconnect
  - Shared interrupt controller



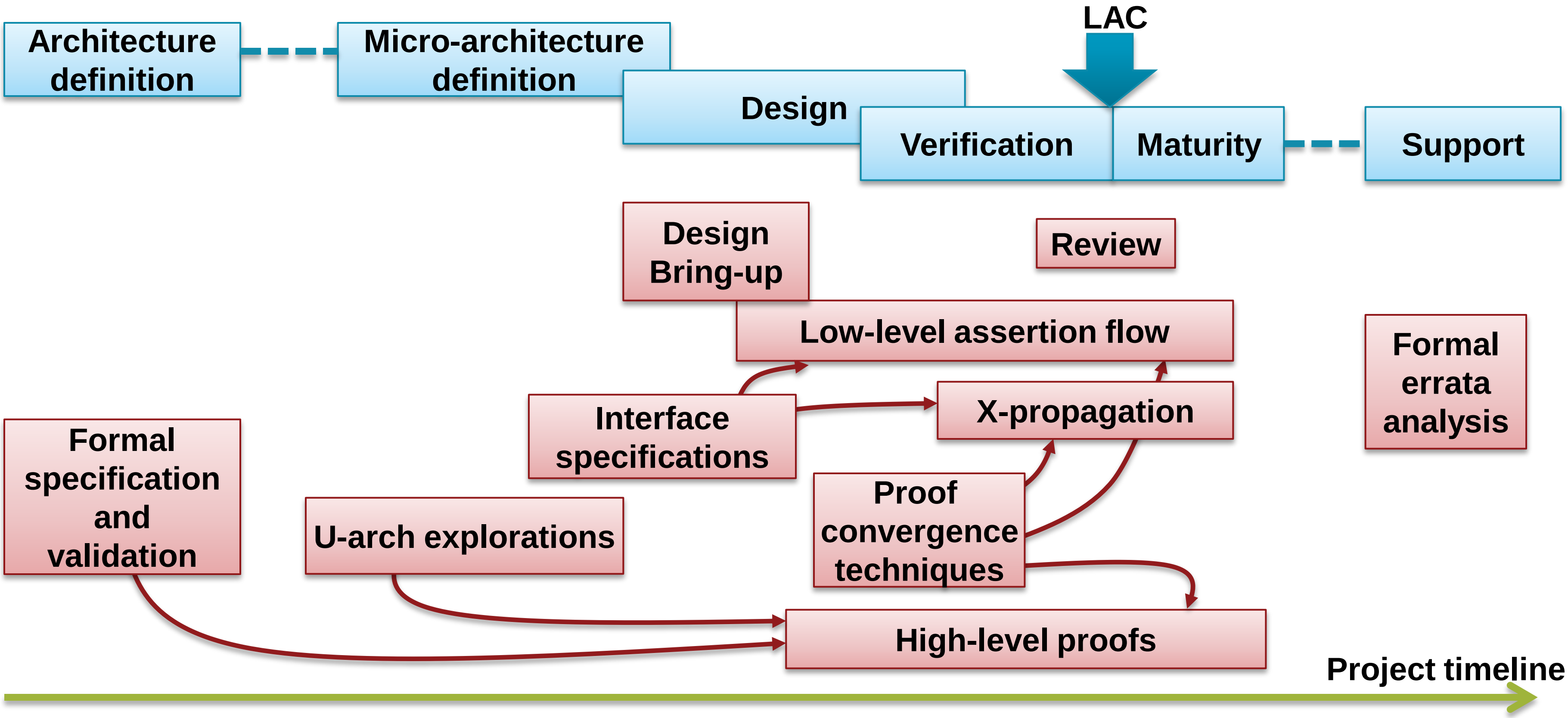
FORMAL IN ARM

Avoidance, Hunting, Absence, Analysis

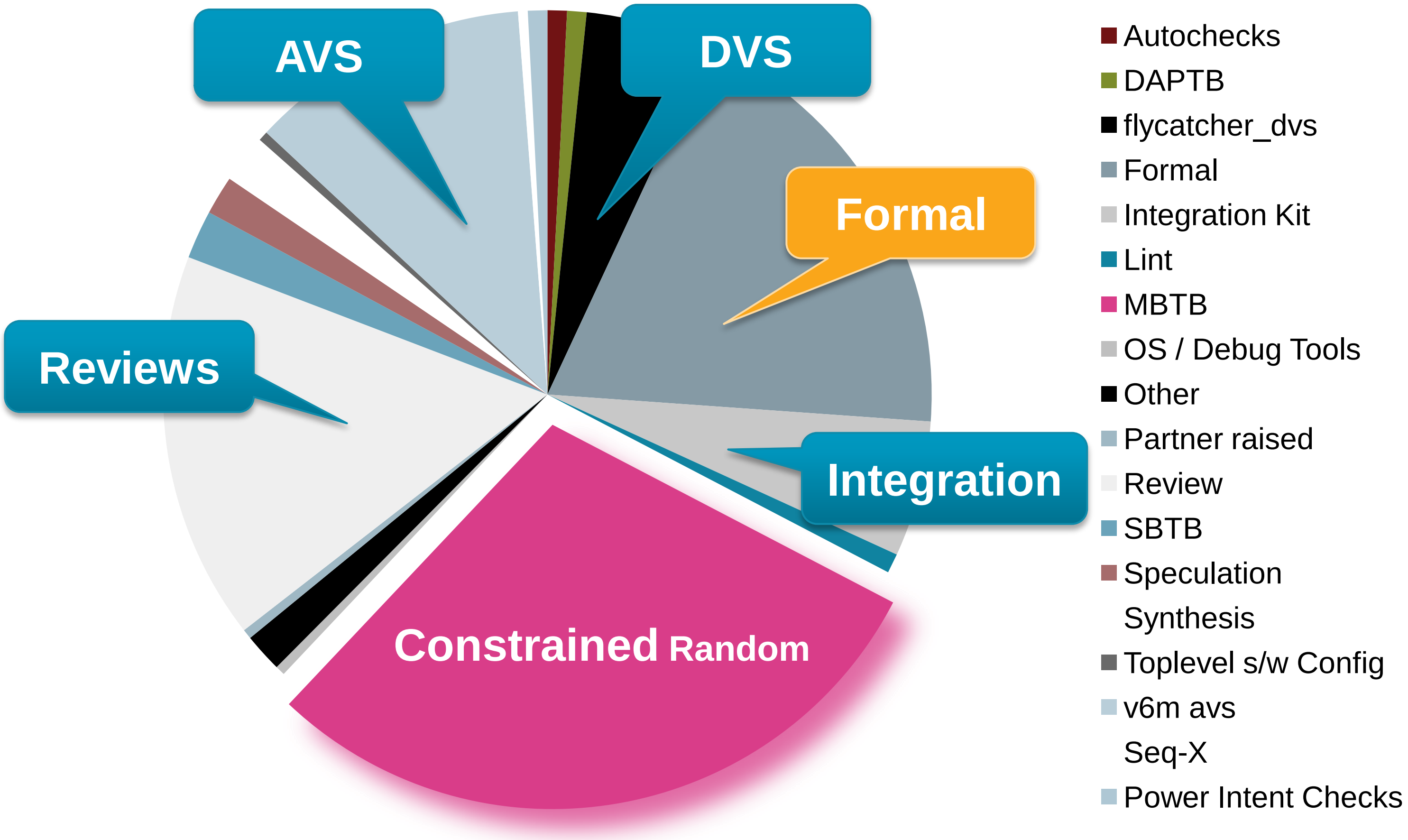
Technique	Advantages	Avoiding Drawbacks
Bug Avoidance • **Improve quality before property checks are run**	• **Improve quality during design** • **Biggest ROI**	**Usually at block level** – E.g. visualisation by designer **May not involve tooling** – E.g. formal modeling, proofs
Bug Hunting • **Looking for bugs** • **Do not worry if proofs do not complete** • **Aim for "**No failures"	• **Ease of set-up** • **Corner cases** • **Low cost, starts early in design process**	**False failures** – Run at higher structural level – Only leads to wasted debug **Non-exhaustive checks** – full proofs are welcome, but not required **Non-uniform run times** – checks are run just for the time available.
Bug Absence • **Aim to get a "complete" set of properties** • **Aim to prove properties** – under certain constraints	• **Only way to get 100% assurance** • **Cover corner cases**	**Non-uniform run times** – Use different proof engines with the tool – Use "invariants" (helper properties) (this adds non-uniform/non-predictable engineering time) – Use safe abstractions – Prove under certain condition (Add extra constraints)
Bug Analysis • **For bugs in FPGA prototypes or in Silicon** – write symptom of bug as a property, generate waveform	• **Ease of setup if constraints exist** • **Can investigate silicon bugs** • **Can confirm fix**	**Interactive generation of constraints to generate legitimate failure scenario**

Formal in the Design Flow

- Formal used at
  - Low-level by designers: design bring-up & embedded properties
  - Medium-level by validation engineers: end-to-end properties
  - High-level by architects: architectural formal specification and validation

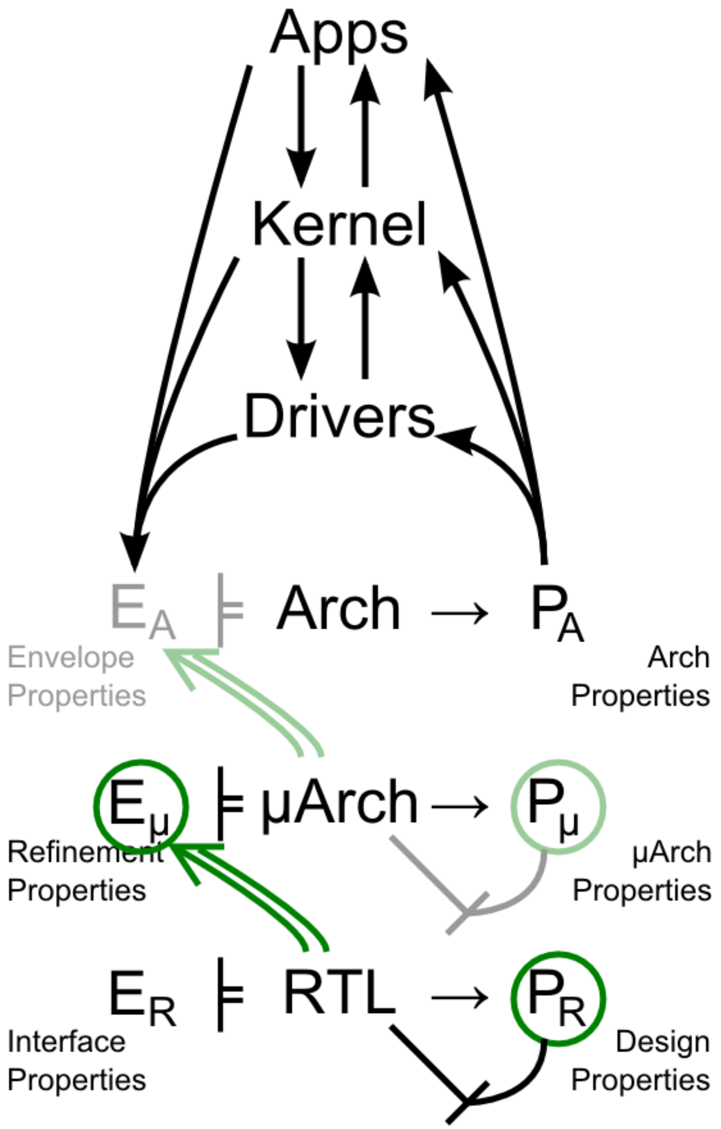


RTL Bugs Found by Method



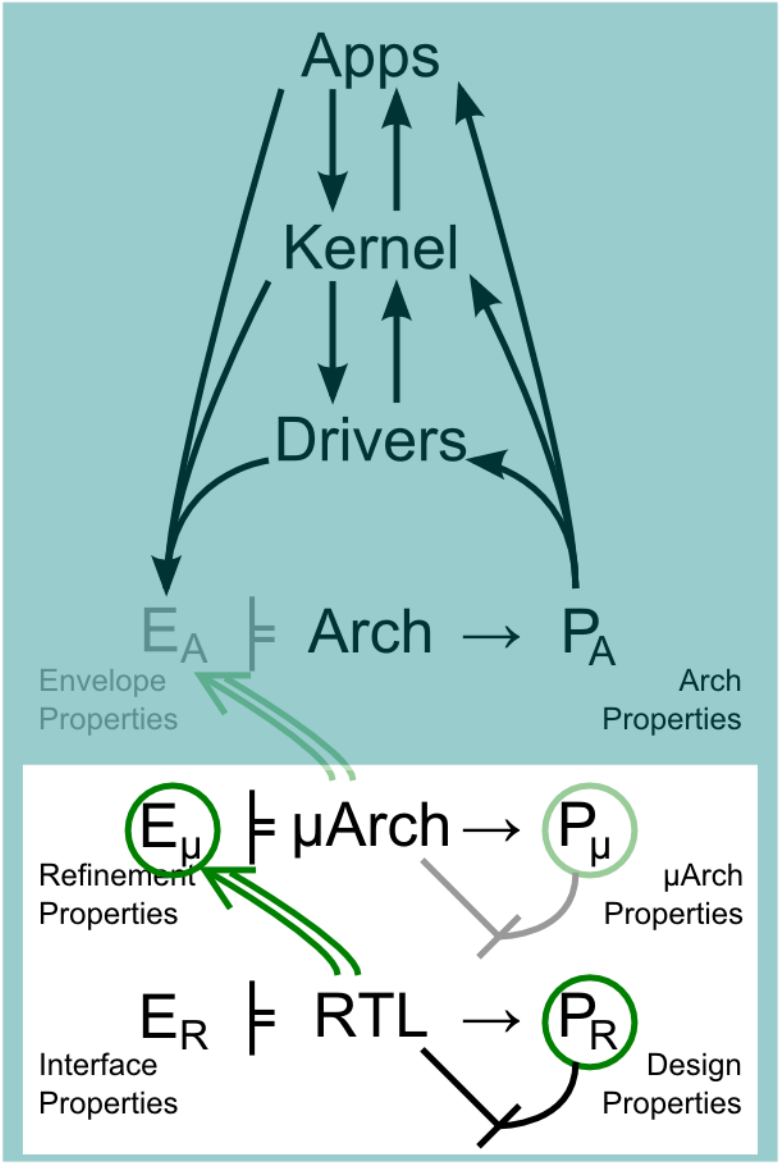
ARM1	3µ	24K T	6 My	2K Hours
Cortex-M0+	20nm	32K T	11 My	1,439K Hours

Bottom Up Formal


Apps
Kernel
Drivers
$E_A \models \text{Arch} \rightarrow P_A$
Envelope Properties
Arch Properties
$E_\mu \models \mu\text{Arch} \rightarrow P_\mu$
Refinement Properties
μArch Properties
$E_R \models \text{RTL} \rightarrow P_R$
Interface Properties
Design Properties

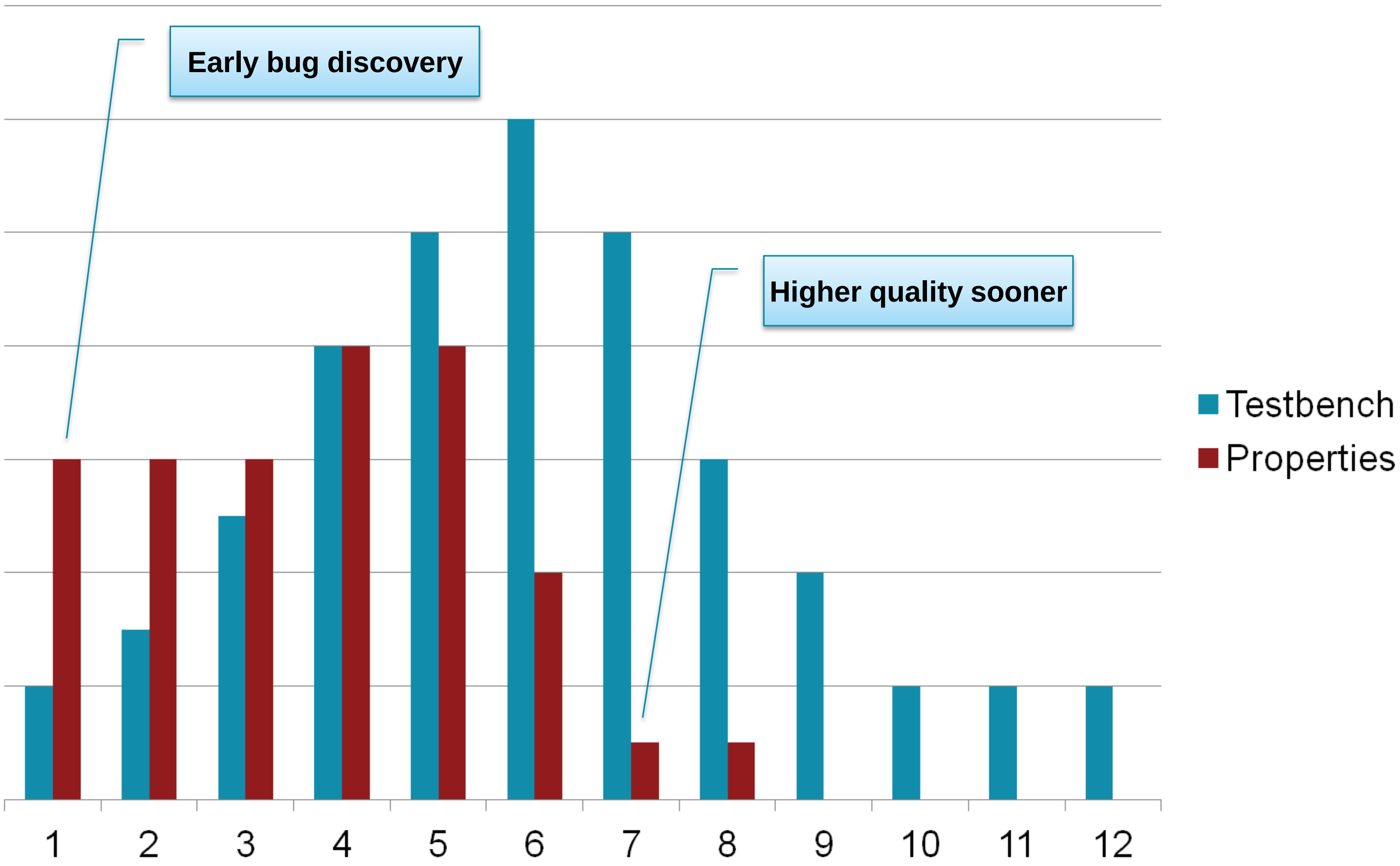
- Software Tools
  - Each level relies on levels around it AND the Architectural behaviour
  - In return the Architecture expects certain behaviour
- Architectural behaviour
  - E.g. Deadlock freedom, power modes, coherency
- Combine techniques to give chain of verification from RTL to Apps

RTL verification


Apps
Kernel
Drivers
$E_A \models \text{Arch} \rightarrow P_A$
Envelope Properties
Arch Properties
$E_\mu \models \mu\text{Arch} \rightarrow P_\mu$
Refinement Properties
μArch Properties
$E_R \models \text{RTL} \rightarrow P_R$
Interface Properties
Design Properties

- Microarchitectural specification for designers is in natural language
- RTL level assertions as standard
  - Written by designers
- Difficult to write end to end properties in terms of RTL state
  - Architectural state is smeared across time and space, or implicit
- Use of abstract models written in SystemVerilog with refinement to RTL level
  - Describing lifecycle of transactions rather than block functionality

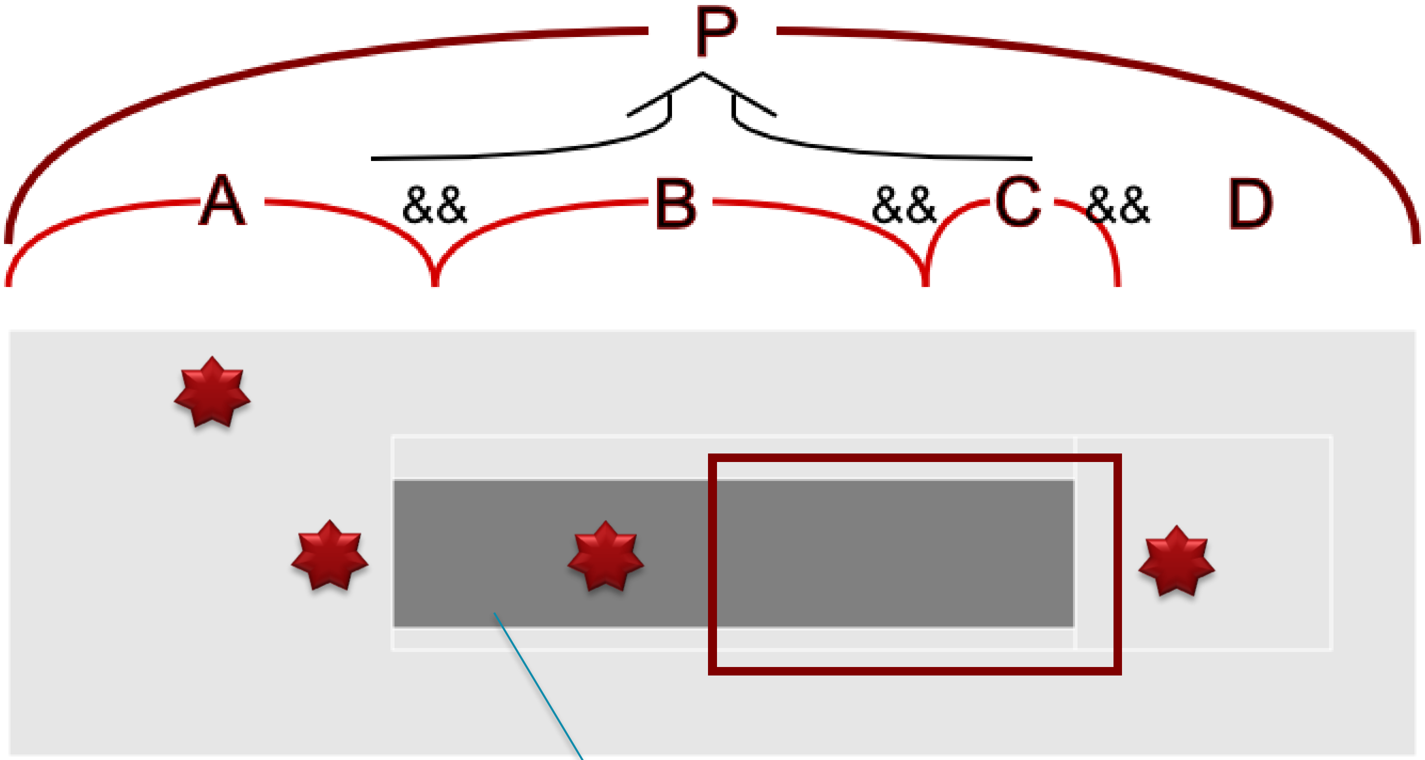
Formal for Designers



Proof Progress and Scaling

- Historically: hard to track progress of formal proof coverage
  - ARM developed progress metrics for proofs and methodology and deployed during a Bug Analysis project
  - Technique for partial proof allowing identification of bug free code
    - Enables focussed review and simulation for weakest blocks
- Historically: architectural properties involve too much RTL detail for tools to handle
  - Developed micro architectural model of SCU
  - SCU Transaction Ordering proven on this specification model
  - RTL shown to meet specification, hence RTL preserves transaction ordering
- These demonstrate proof is now measurable and scalable

Partial Proof

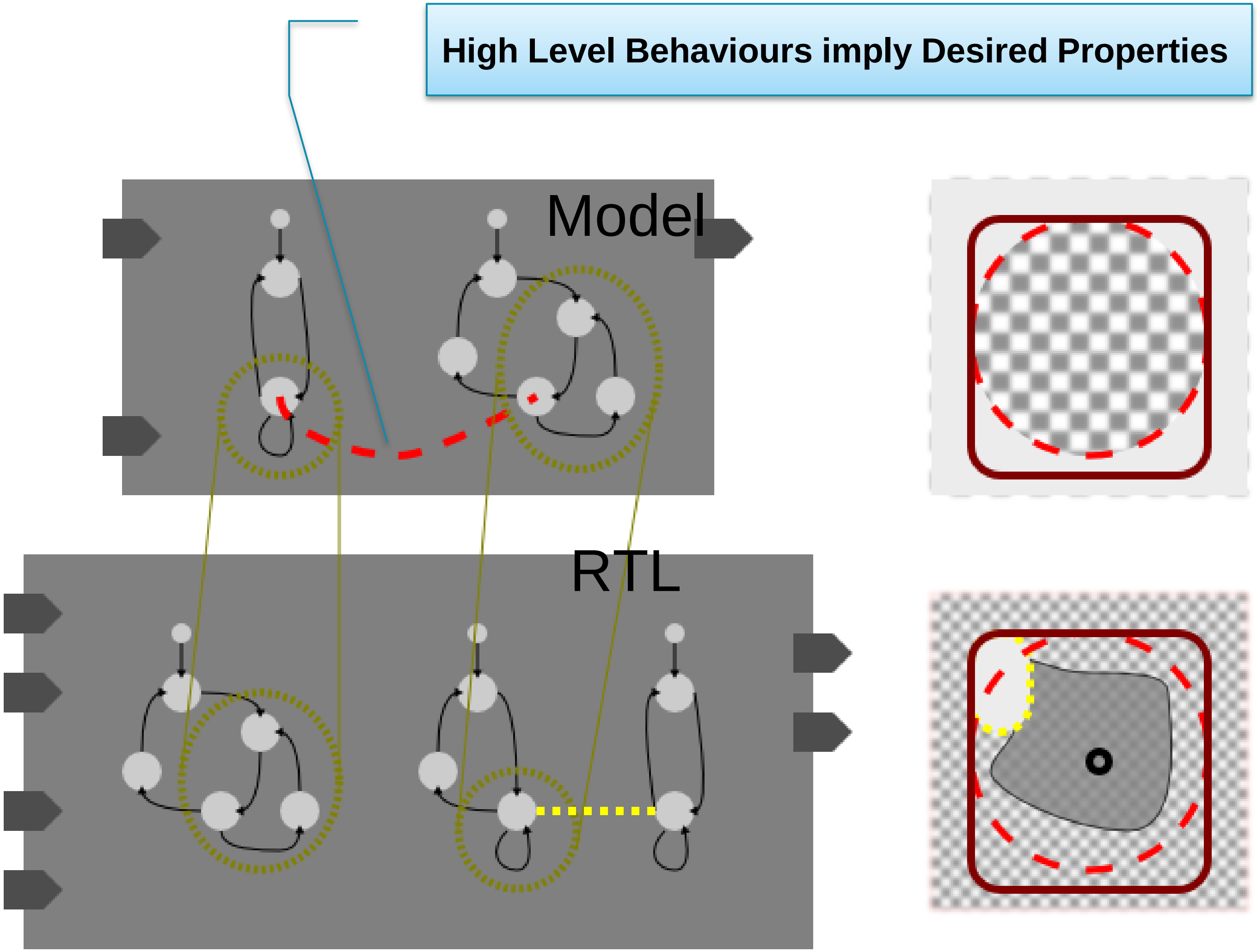


Unproven lemma D focuses Simulation and Review

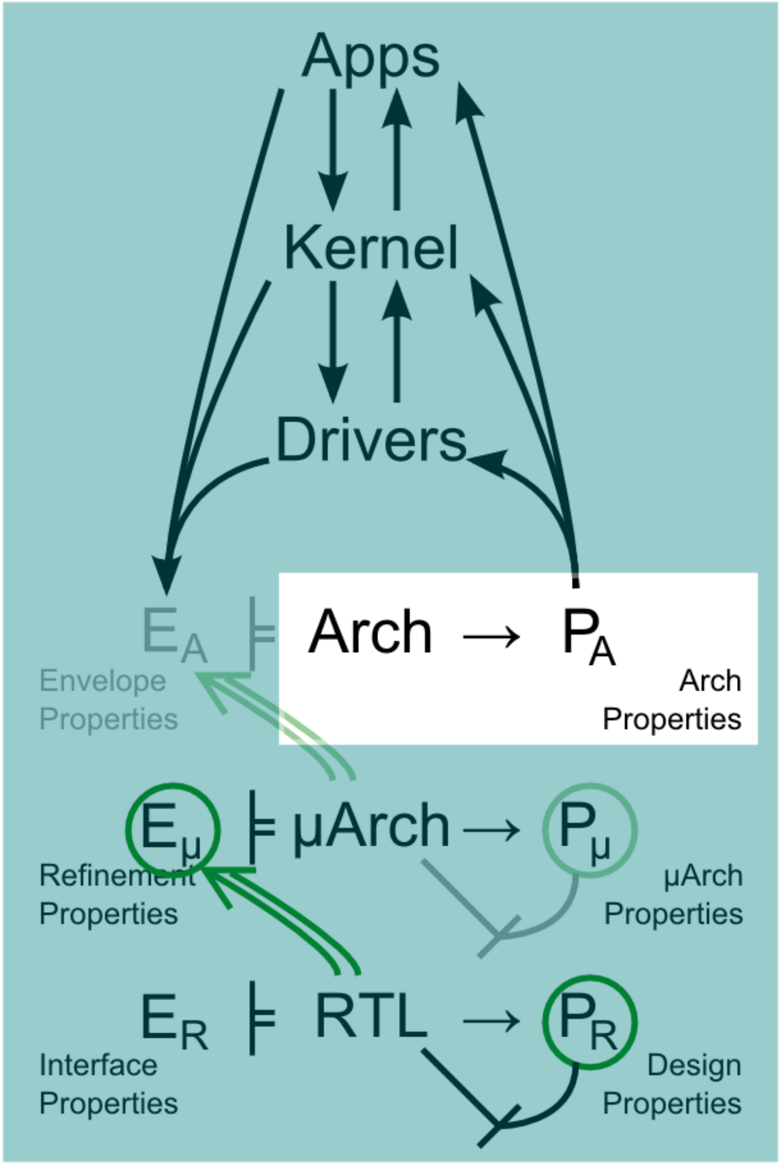
Micro Architectural Models

- Formal Model
  - An abstraction expressed as transactors, FSMs, assumptions…
  - Provides vocabulary of abstract events
- Desired Model Properties
  - Properties which should arise from a correct implementation
    - Safety or liveness assertions
- High Level Behaviour
  - What implementation is sufficient?
    - assume to prove formal model exhibits desired properties
    - assert on RTL to deduce that it satisfies specification
- Covers
  - sanity check the formal specification
  - RTL bring up

Micro Architectural Models


High Level Behaviours imply Desired Properties

Architecture



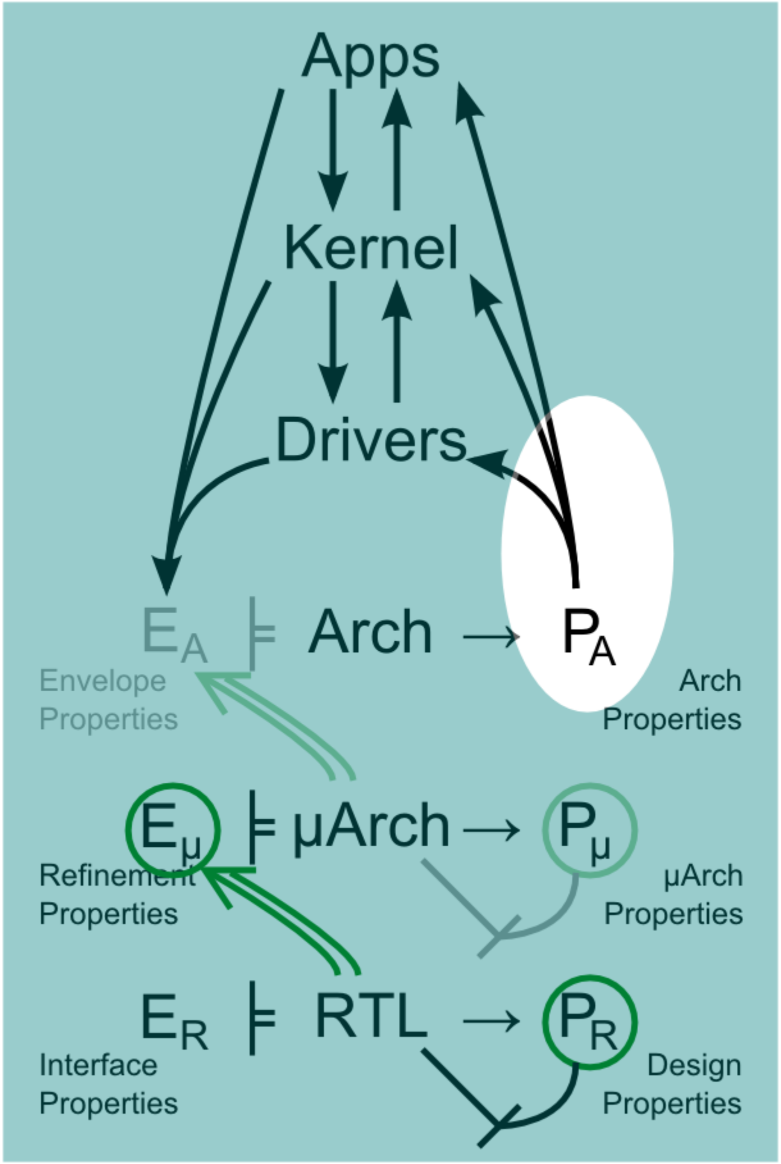
- The architecture defines several envelopes of reliable behaviour:
  - ISA – programmer's view of instruction
  - Weak Memory – implementation freedom, unintuitive behaviour
  - Coherent interconnect – AMBA4 ACE transactions
  - Power modes – domains, required functionality
  - Security – Trustzone
  - Debug and trace behaviour
- How to verify individually and interdependently?
- How to specify non-determinism?

Architecture Validation

- SystemVerilog model of AMBA4 ACE
- Deadlock discovered in draft specification using JasperGold
  - 4 master system, unlikely to find by hand
- Murphi model of AMBA4 ACE master with bridge to alternative interface for
  - Protocol deadlock
  - System coherency
- PReach Murphi
  - 25 threads, 1Tb
  - Smallest case completed
  - Several bugs found during development

Master nodes	IDs	Result
2	1	3 hours
2	2	-
3	1	-
3	2	-
4	1	-
4	2	-

Systems and Software



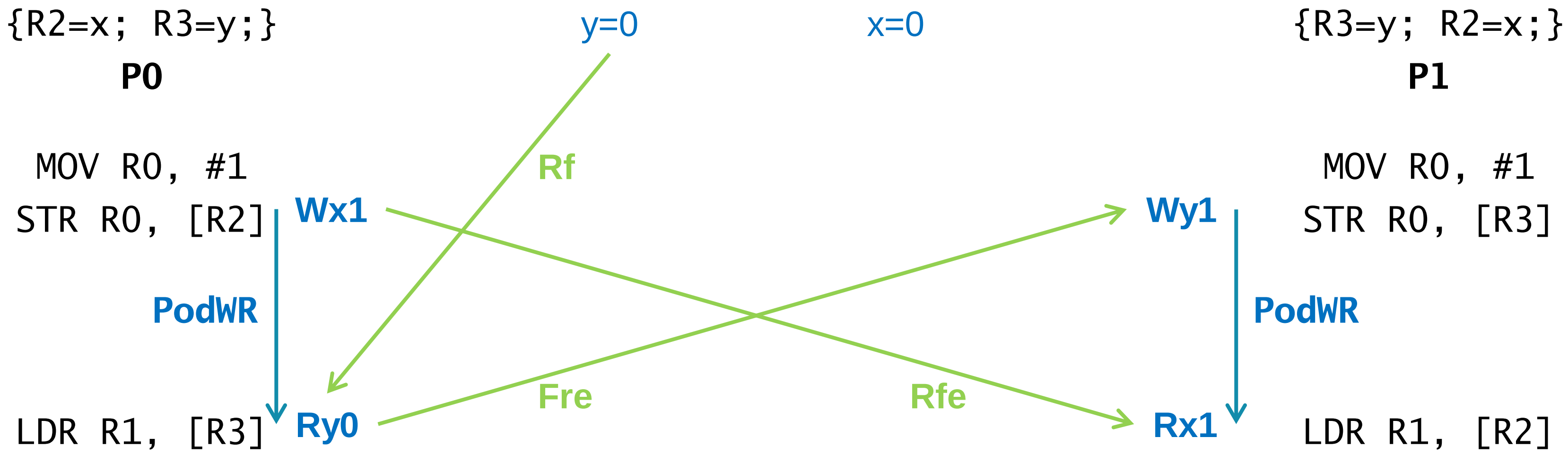
- System level testing
  - Requires accurate models of expected behaviour
  - Relate testing to coverage of specification

- What useful IP can we supply to our partners for software development?

Sequentially Consistent execution

```
ARM SB
"PodWR Fre PodWR Fre"
```

Observe **P0** end with **R1=0** and **P1** end with **R1=1**



```
{R2=x; R3=y;}
      P0

   MOV R0, #1
  STR R0, [R2]

  LDR R1, [R3]
```

```
{R3=y; R2=x;}
      P1

   MOV R0, #1
  STR R0, [R3]

  LDR R1, [R2]
```

Program order candidate relations

PodWR = Program order different address Write then Read

Coherency ordering (Communication) relations

Rfe = Target Reads its value from a source on an external processor

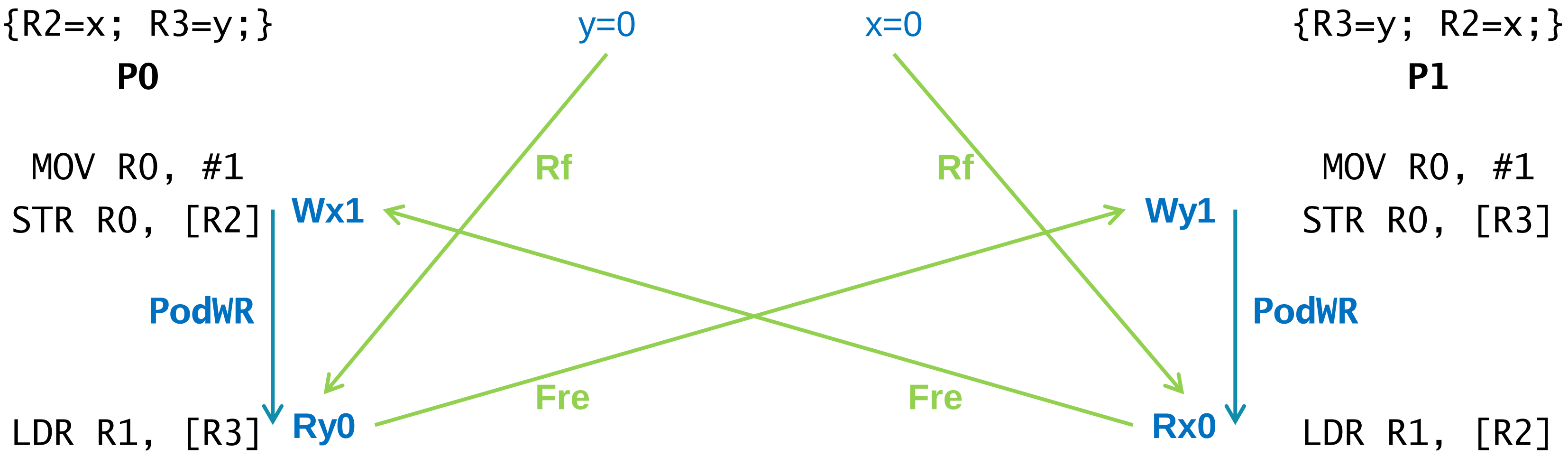
Fre = Source reads From a write that precedes target (on an external processor) in coherence order

Relaxing candidate relations

```
ARM SB
"PodWR Fre PodWR Fre"
```

Observe both threads ending with **R1=0**

Relaxing PodWR breaks the cycle



```
{R2=x; R3=y;}
      P0

  MOV R0, #1
 STR R0, [R2]

 LDR R1, [R3]
```

```
{R3=y; R2=x;}
      P1

  MOV R0, #1
 STR R0, [R3]

 LDR R1, [R2]
```

Program order candidate relations

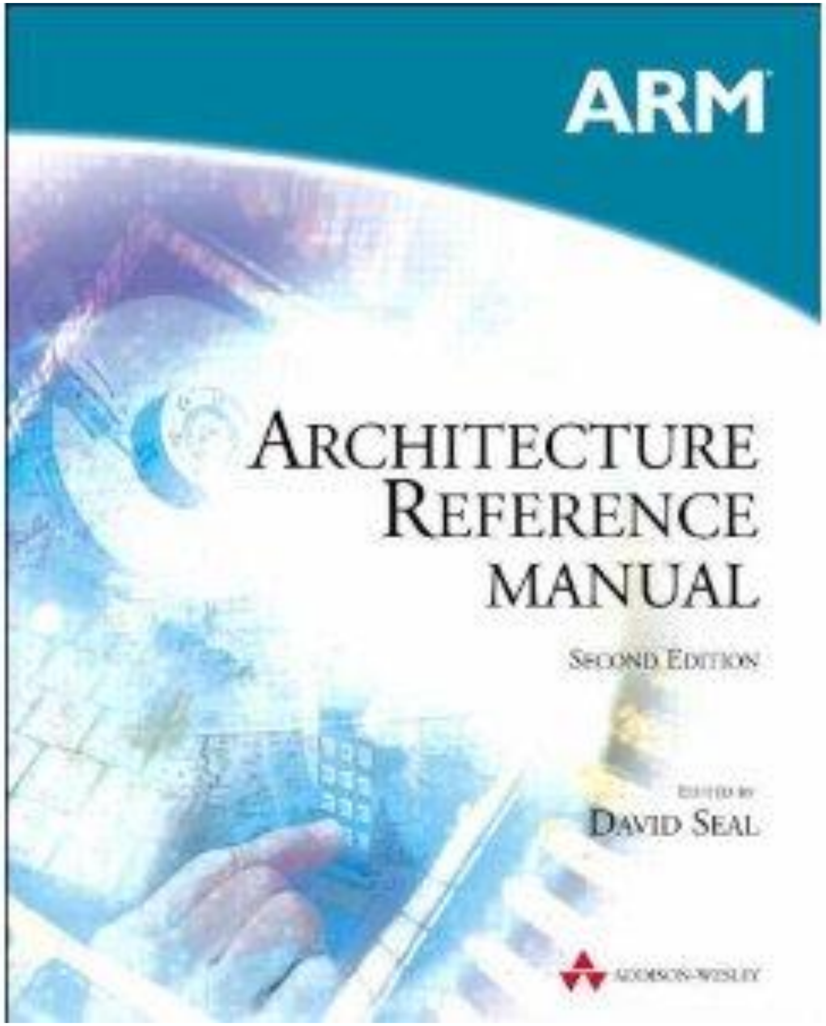
PodWR = Program order different address Write then Read

Coherency ordering (Communication) relations

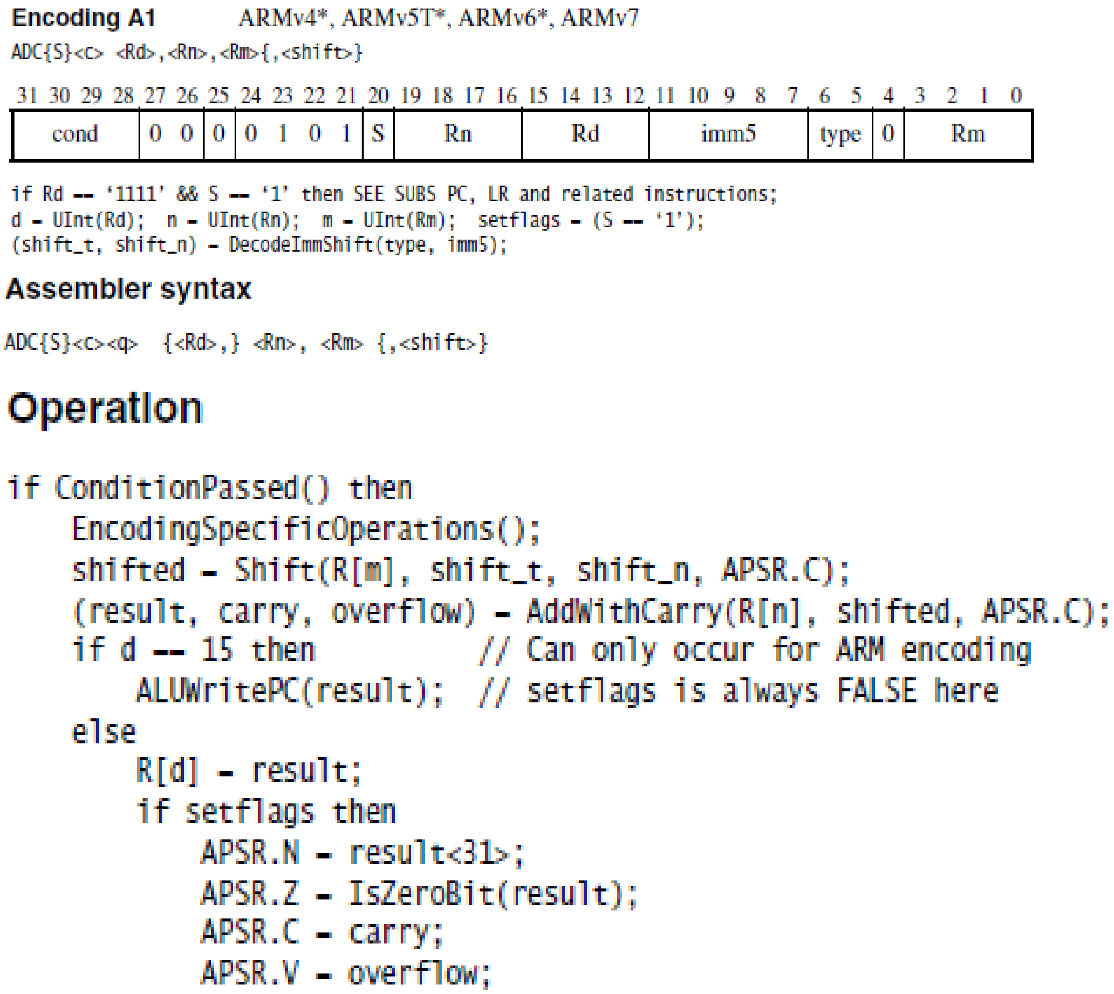
Rfe = Target Reads its value from a source on an external processor

Fre = Source reads From a write that precedes target (on an external processor) in coherence order

The ARM ARM


ARM
ARCHITECTURE
REFERENCE
MANUAL
SECOND EDITION
EDITED BY
DAVID SEAL
ADDISON-WESLEY

ARMv7 specification



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);
```

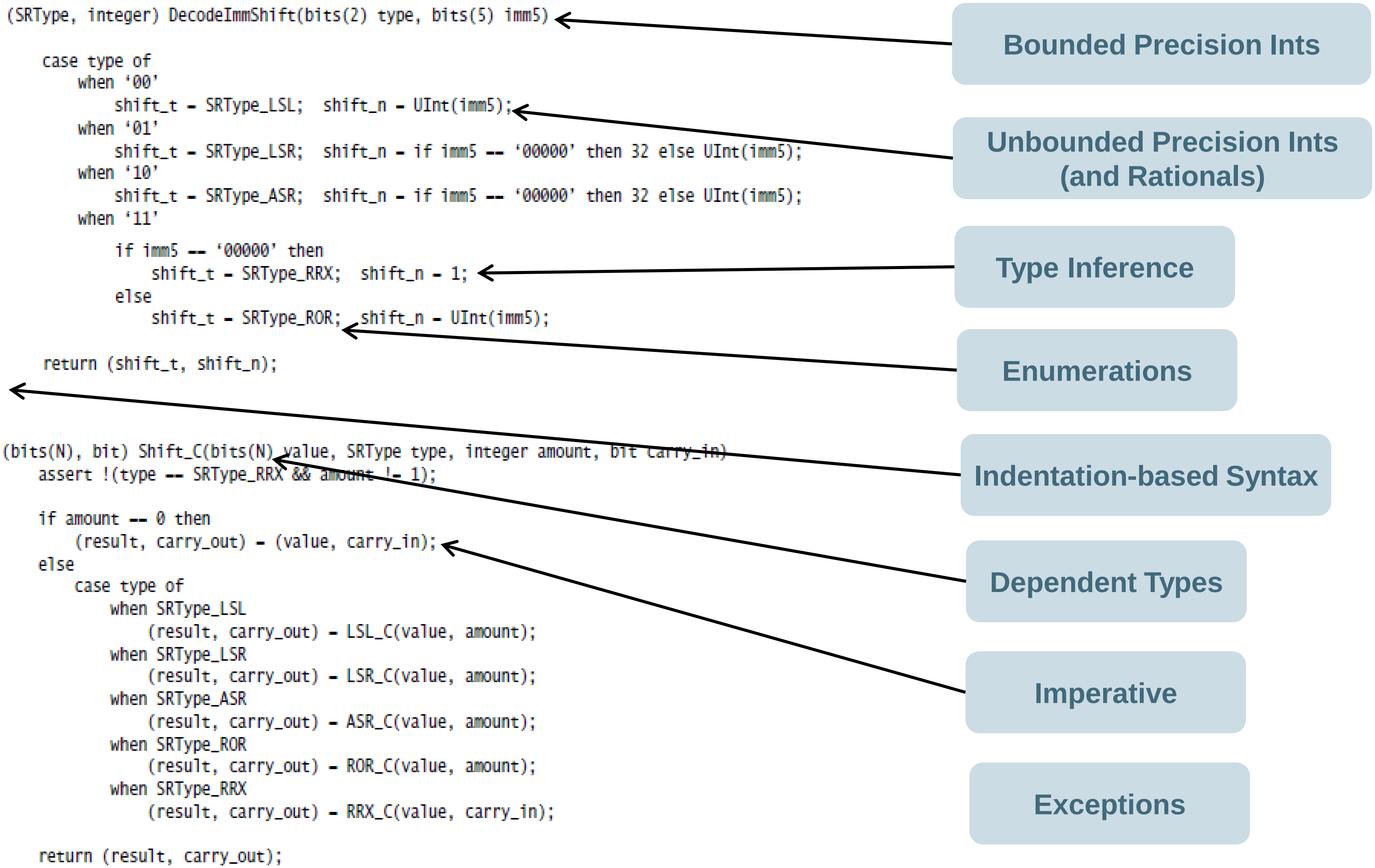
Assembler syntax

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

Operation

```
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;
```

ARMv7 support functions



```
(SRType, integer) DecodeImmShift(bits(2) type, bits(5) imm5)

    case type of
        when ‘00’
            shift_t = SRType_LSL;  shift_n = UInt(imm5);
        when ‘01’
            shift_t = SRType_LSR;  shift_n = if imm5 == ‘00000’ then 32 else UInt(imm5);
        when ‘10’
            shift_t = SRType_ASR;  shift_n = if imm5 == ‘00000’ then 32 else UInt(imm5);
        when ‘11’
            if imm5 == ‘00000’ then
                shift_t = SRType_RRX;  shift_n = 1;
            else
                shift_t = SRType_ROR;  shift_n = UInt(imm5);

    return (shift_t, shift_n);


(bits(N), bit) Shift_C(bits(N) value, SRType type, integer amount, bit carry_in)
    assert !(type == SRType_RRX && amount != 1);

    if amount == 0 then
        (result, carry_out) = (value, carry_in);
    else
        case type of
            when SRType_LSL
                (result, carry_out) = LSL_C(value, amount);
            when SRType_LSR
                (result, carry_out) = LSR_C(value, amount);
            when SRType_ASR
                (result, carry_out) = ASR_C(value, amount);
            when SRType_ROR
                (result, carry_out) = ROR_C(value, amount);
            when SRType_RRX
                (result, carry_out) = RRX_C(value, carry_in);

    return (result, carry_out);
```

Bounded Precision Ints

Unbounded Precision Ints (and Rationals)

Type Inference

Enumerations

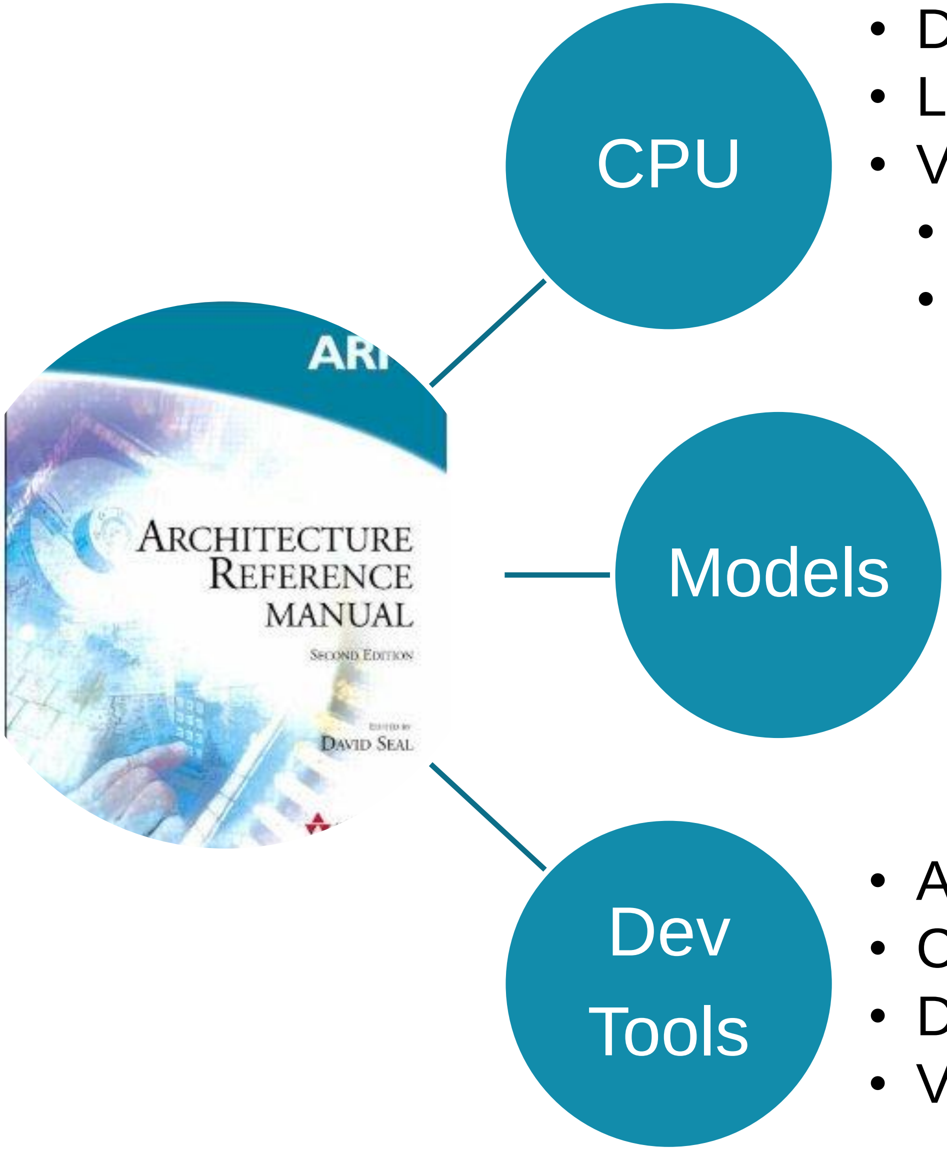
Indentation-based Syntax

Dependent Types

Imperative

Exceptions

What ARM uses ISA spec for



CPU

- Design
- Licensing
- Validation
  - Test suites
  - Test tools

Models

- Design
- Validation

Dev Tools

- Asm/dasm/ld
- Compiler
- Debugger
- Validation

Summary

- “Systems design today is on the same level of development as mechanics in the middle ages - based on experiences with no formal theory of design.” J. Sifakis FMCAD 2010
- We have made good progress on pieces of the puzzle, designers are turning to formal to relieve the pain
- A combination of tools and techniques
    - Use those best suited to each problem domain
    - Must be able to relate to each other, and simulation
    - The system design does not end with us – enable partners

THE END