arm

VLSI workshop on System Validation

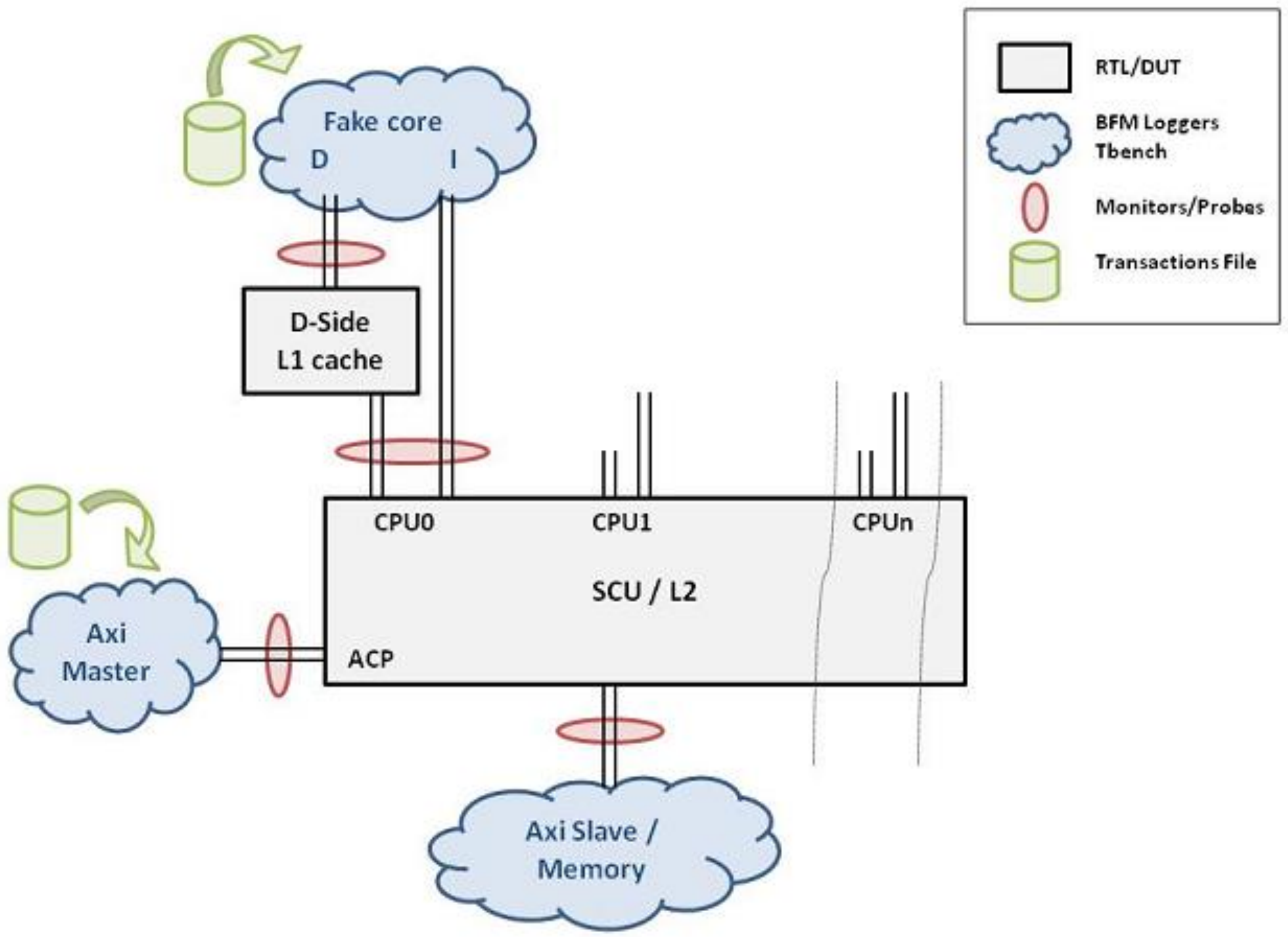
Parivesh and Vikash

Multi-layered verification strategy

Silicon verification	• At-speed testing of external interfaces • Extended OS payloads • API compliance testing
System verification	• Interoperability • Real payloads & benchmarking • OS-boot & application stress
Top level	• Basic directed testing • Random testing • Realistic payloads
Unit level	• Unit-level testbenches • Formal proofs

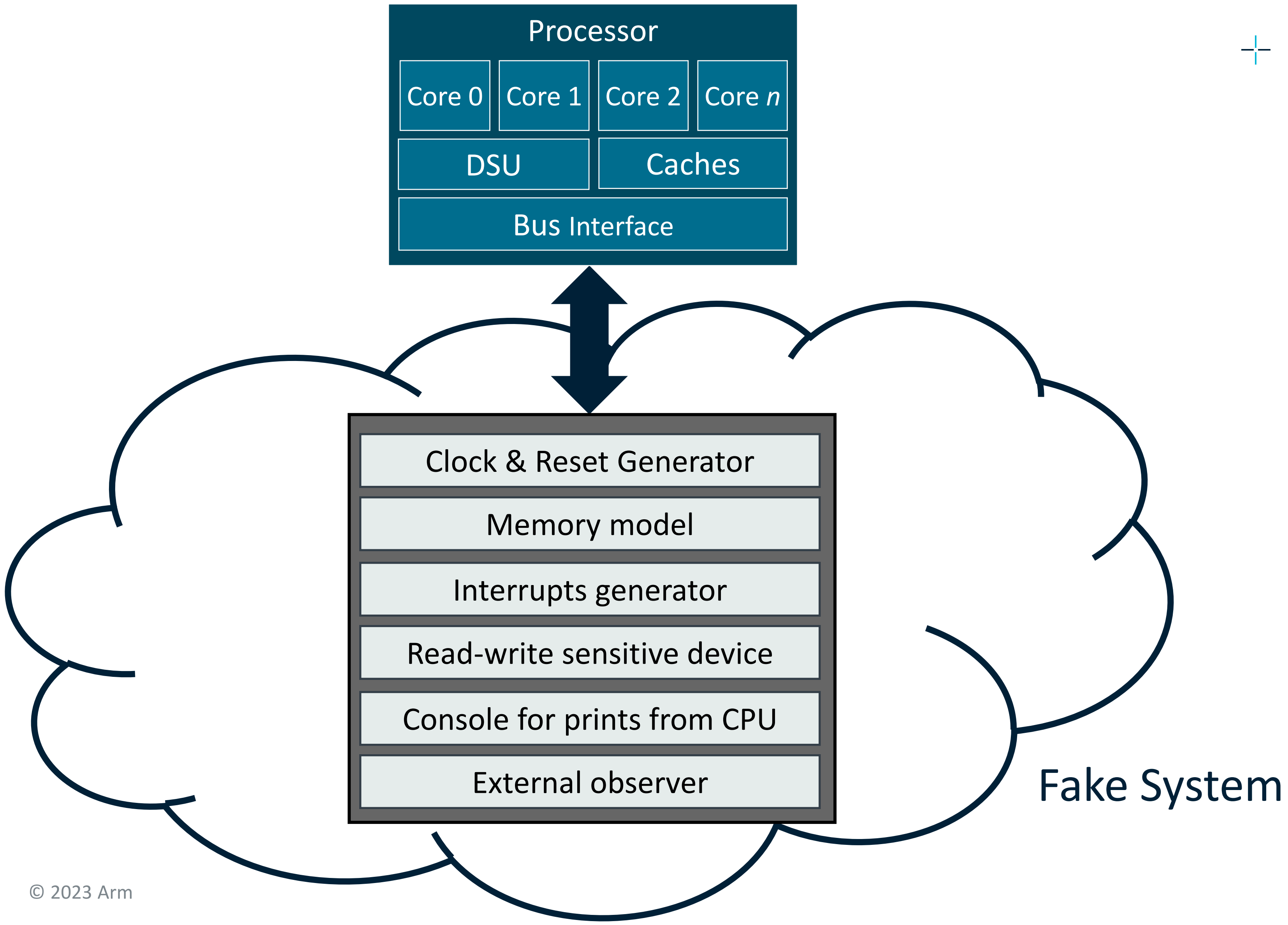
- Exhaust finding bugs at the lowest level before moving up a level of abstraction
- Bottom-up verification methodology followed for each IP in the system like Processor, GPU, Interconnect, etc.

An example of unit-level verification



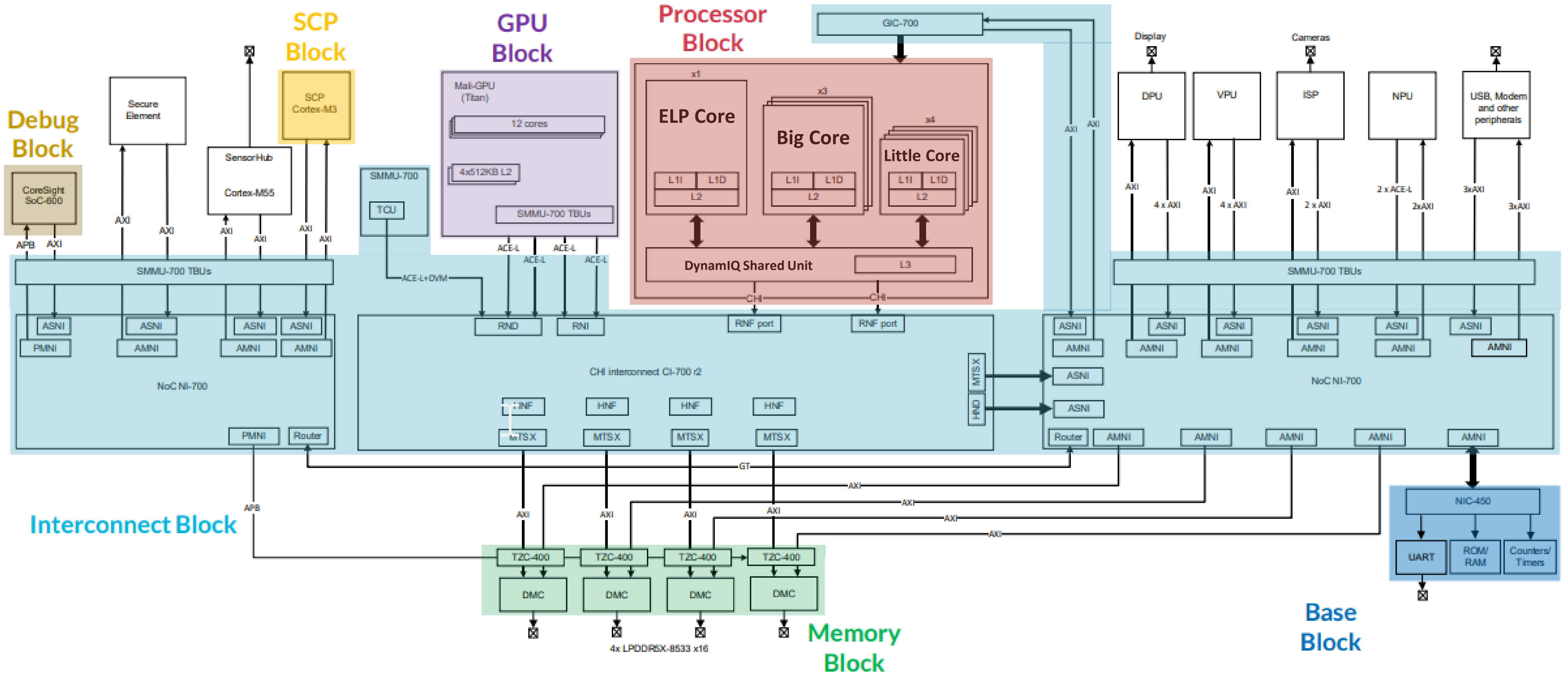
- This example shows the unit-level verification of the Snoop Control Unit (SCU) inside a processor
- Bus-Functional Units (BFMs) are used to imitate the traffic coming from other units
- Random traffic is used to thoroughly verify the unit and find all bugs

An example of top-level verification

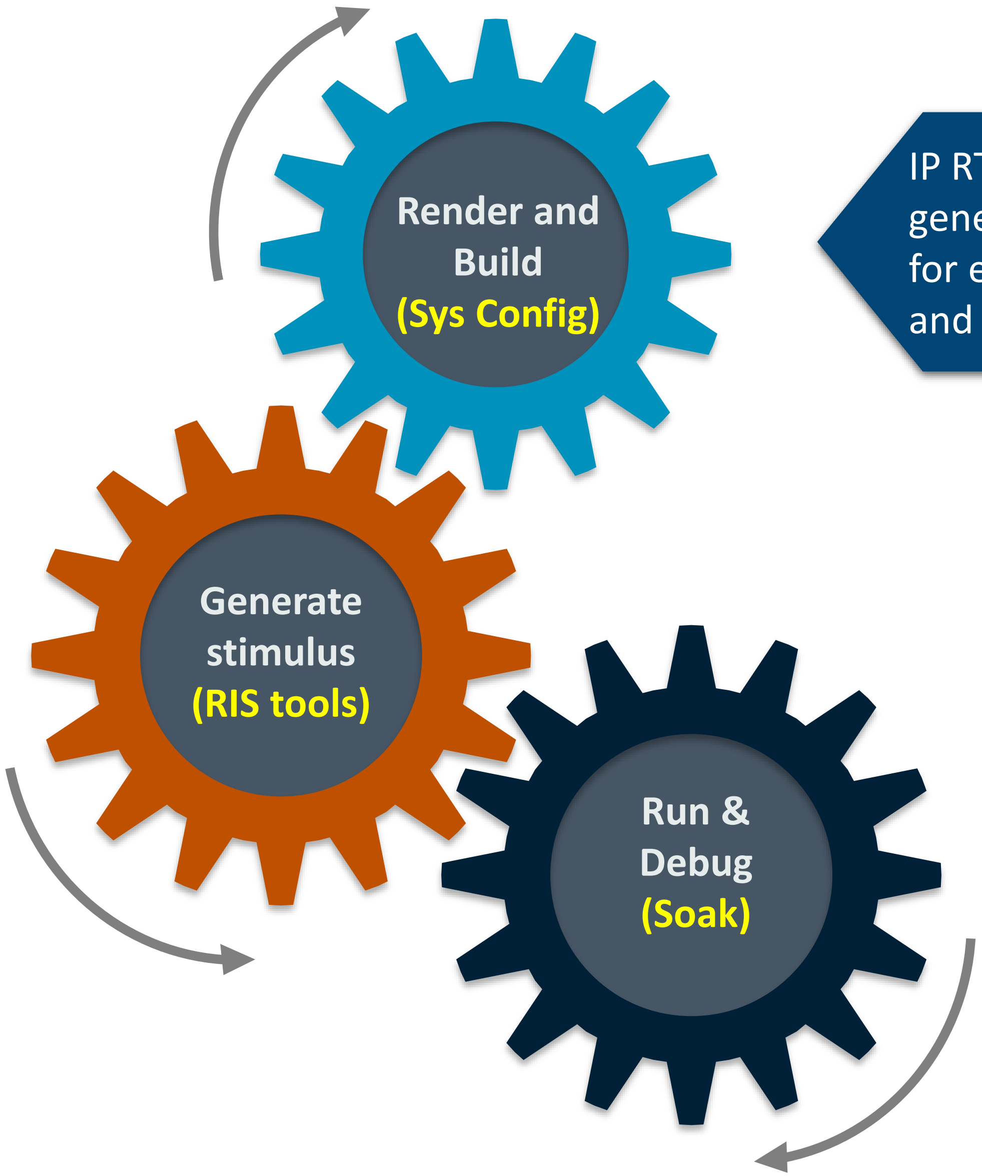


- This example shows the top-level verification setup for a multi-core processor with a fake system

An example processor subsystem



Introduction to system level verification

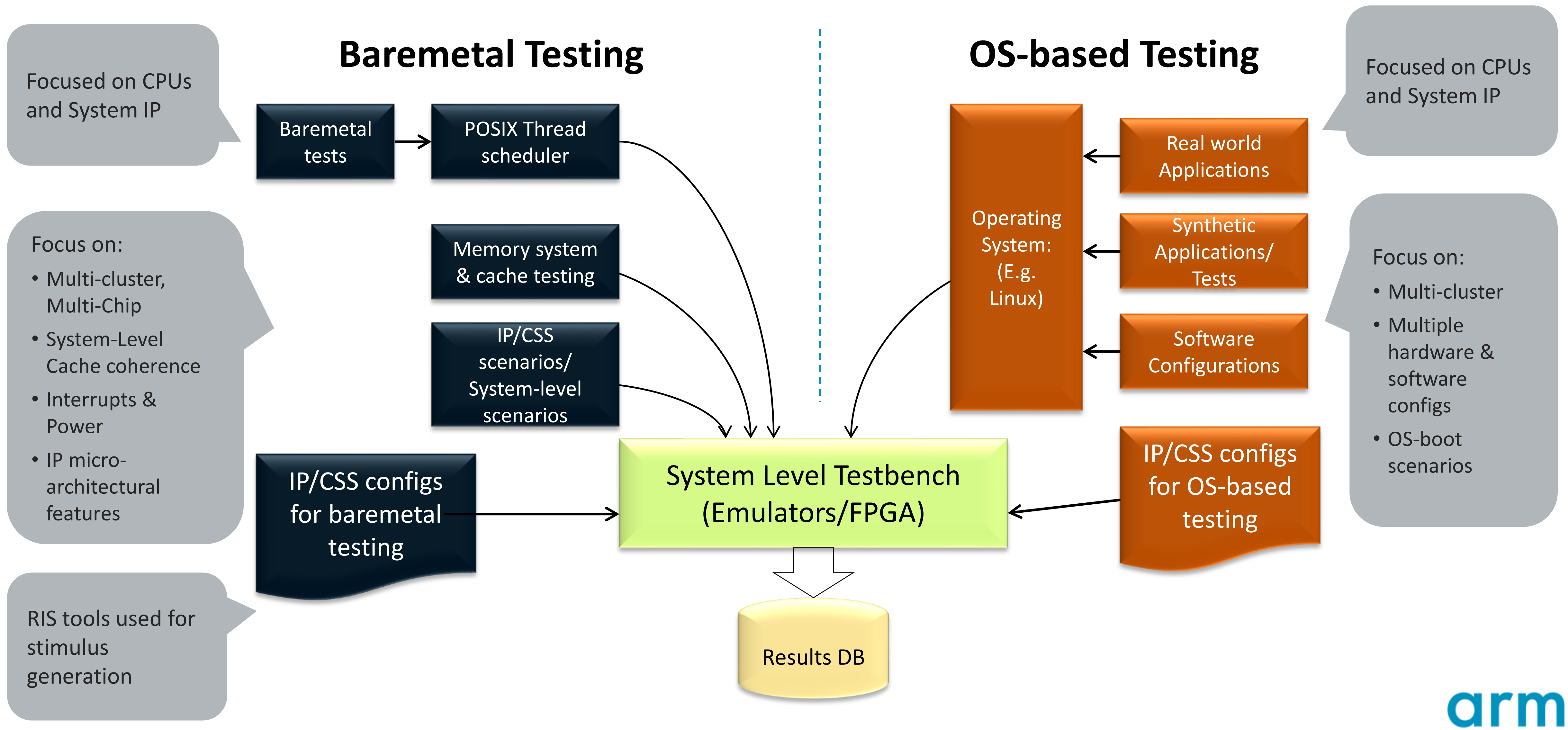


IP RTL, Test Bench, Traffic generators – fully synthesizable for ease of use on Emulators and FPGA

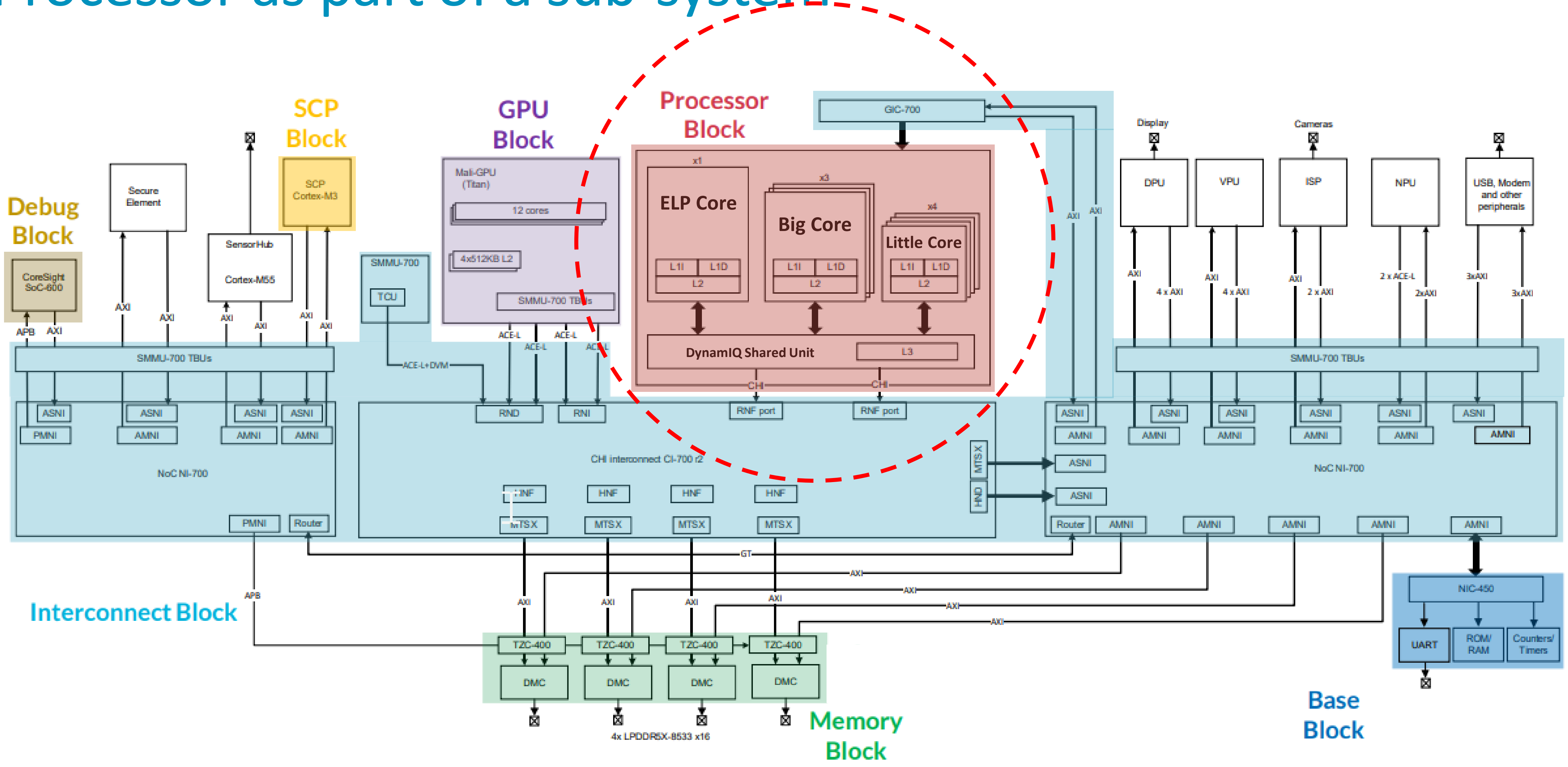
Random Instruction Sequence (RIS) generators, Instruction driven stimulus, Generators are proven on many designs

Run & debug fails, Analysis through Statistical coverage, f/b for Stimulus generators, Iterative process

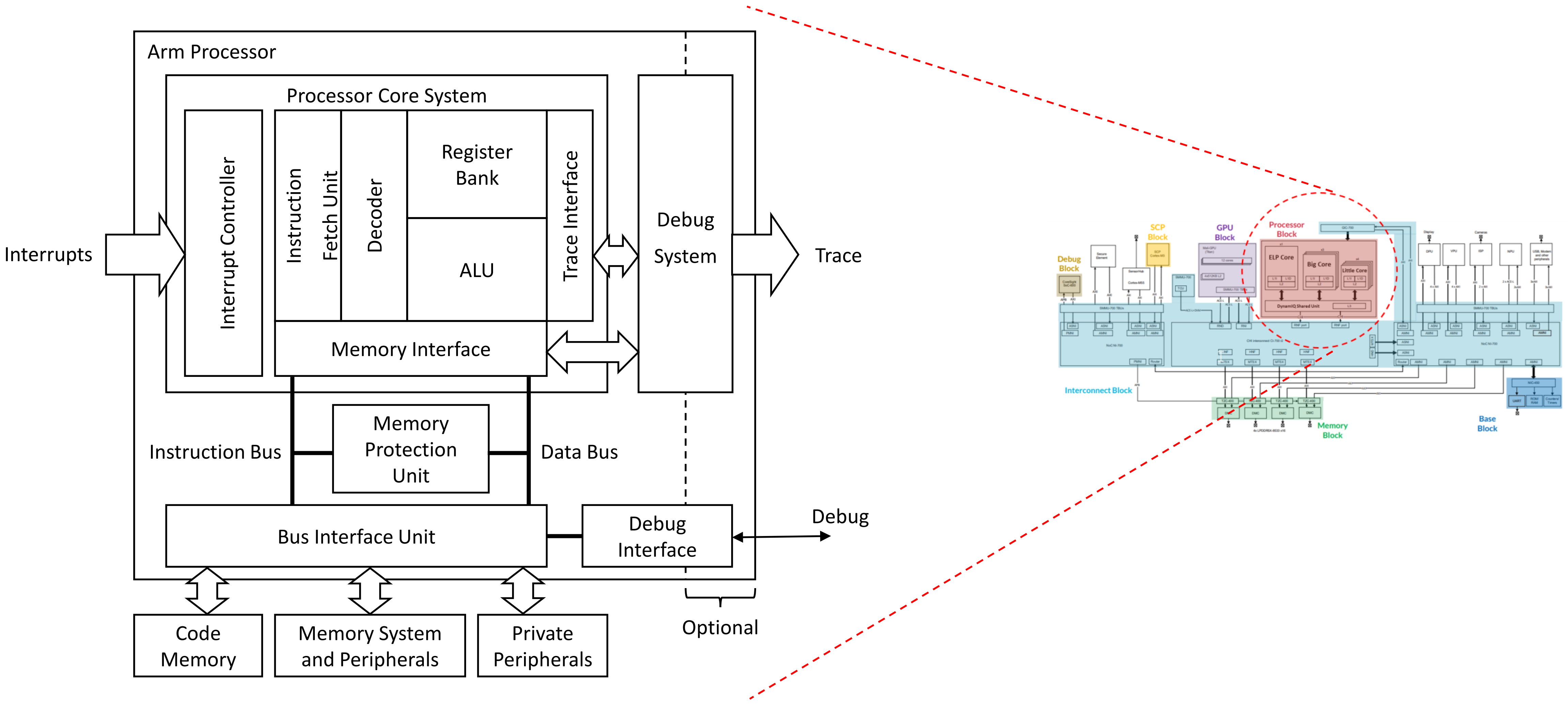
System Level Verification Overview


Focused on CPUs and System IP
Baremetal Testing
Baremetal tests
POSIX Thread scheduler
Memory system & cache testing
IP/CSS scenarios/ System-level scenarios
Focus on:
• Multi-cluster, Multi-Chip
• System-Level Cache coherence
• Interrupts & Power
• IP micro-architectural features
IP/CSS configs for baremetal testing
RIS tools used for stimulus generation
System Level Testbench (Emulators/FPGA)
Results DB
OS-based Testing
Operating System: (E.g. Linux)
Real world Applications
Synthetic Applications/ Tests
Software Configurations
IP/CSS configs for OS-based testing
Focused on CPUs and System IP
Focus on:
• Multi-cluster
• Multiple hardware & software configs
• OS-boot scenarios

Processor as part of a sub-system



An example processor made of multiple units



System Verification

Methodology

Challenge

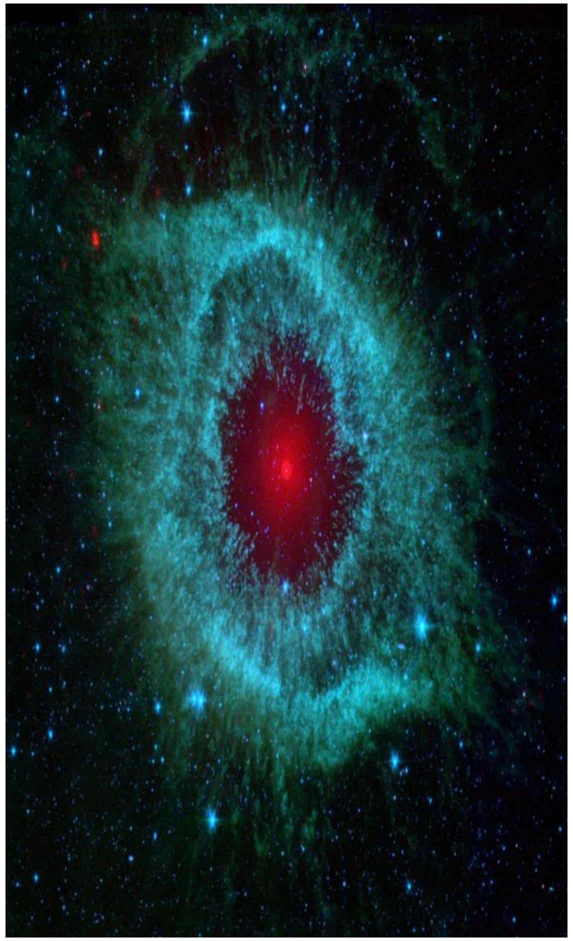
- ~450 arm64 instructions, 12 stage pipeline, 10^23 combinations
- Overlay with interrupts, memory types, levels of cache, traffic from other agents – state space explosion

Approach

- Impractical to cover the state space using directed stimulus
- Taken the random verification approach

Solution

- We created a couple of tools that focusses on different areas of the system



SVRIS Tools

SV-RIS tools

- Used within Arm for verifying CPUs and System IP for several years now
- Have found over 400 bugs
- Supports emulator, FPGA, and silicon
- Scalable with the number of CPUs – have been run on a 64-CPU system
- Use multi-pass consistency data checking
  - Tests are run twice or more (on same or different CPU variants) and results are compared across runs
  - Correctness check

Tool 1

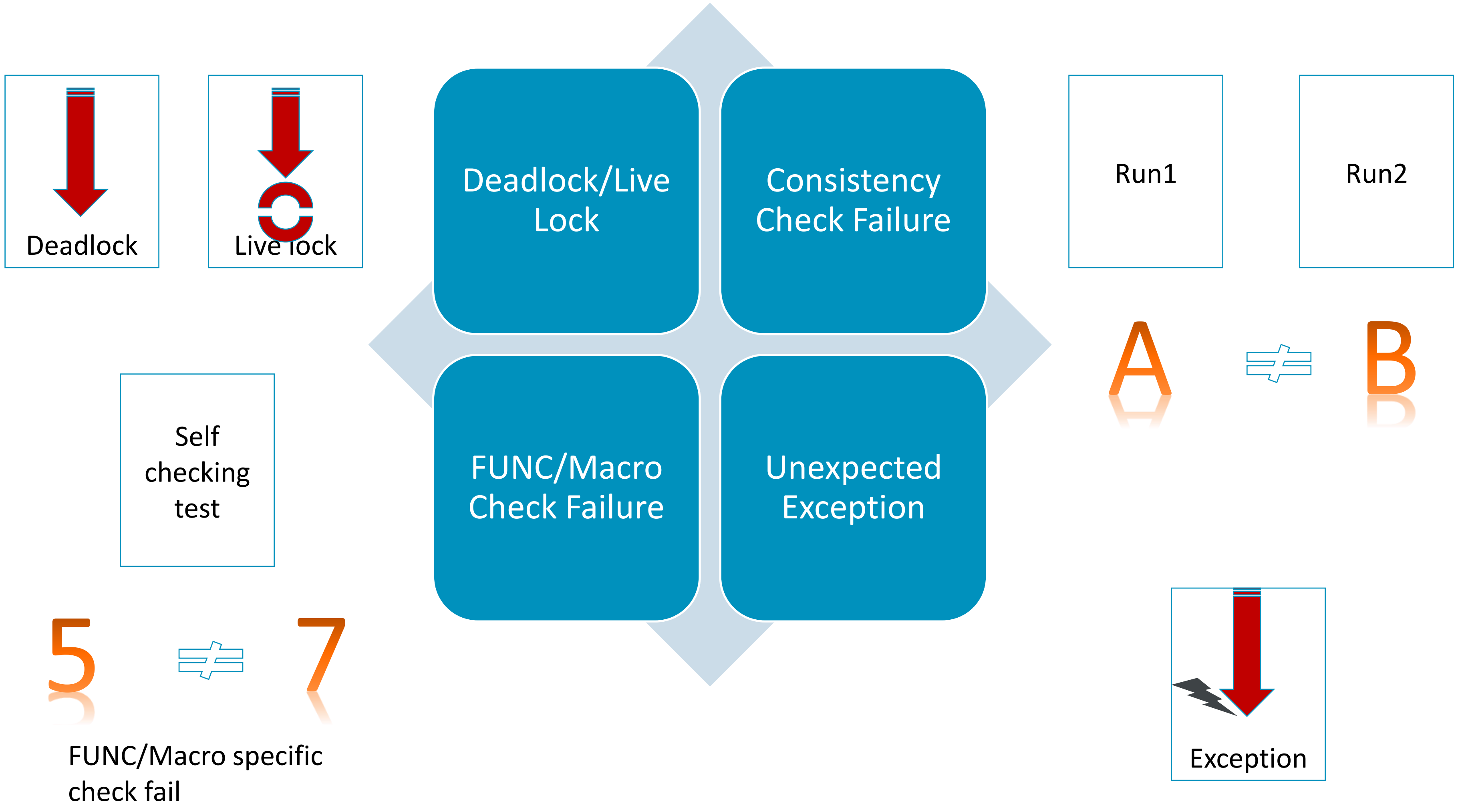
- Memory focused RIS generator
- Areas of focus: Cache coherence, atomics, ordering, translation, memsys – inter-cluster and within cluster, power, RAS

Tool 2

- For semi directed MP tests
- Core pipeline focused RIS generator, supports full ISA
- Areas of focus: Data forwarding, hazards, flushes, exceptions, branches, speculation, etc.

arm

Types of fails found using the RIS Tools

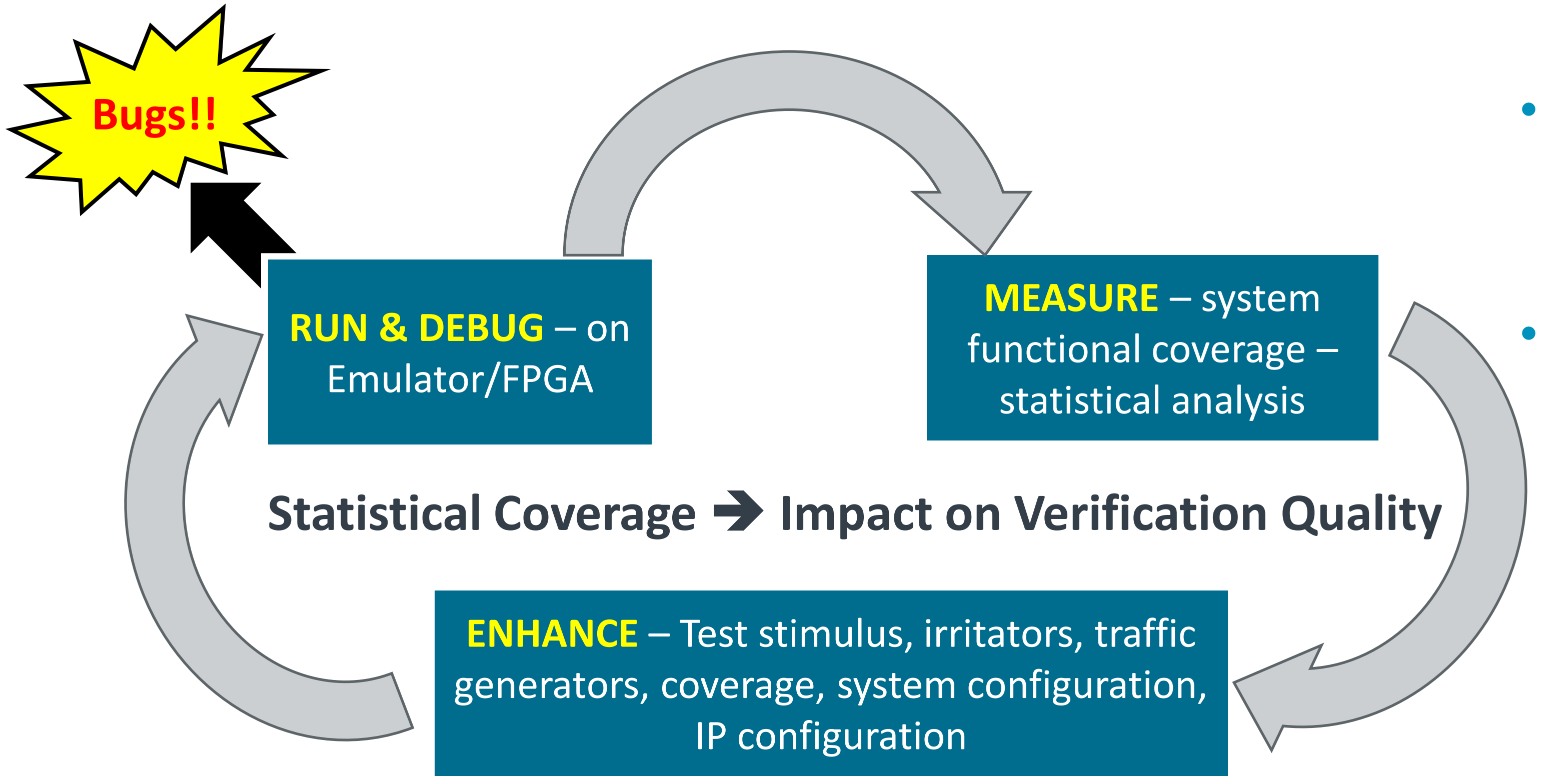


OS Based Validation

Use Case Scenarios

- Linux boot/bring-up
- Execute payloads that target real world use case scenarios on reference systems
- Run Linux along with hypervisors
  - In house Hypervisor testbench
  - KVM

System Level Statistical Coverage



Bugs!!
RUN & DEBUG – on Emulator/FPGA
MEASURE – system functional coverage – statistical analysis
Statistical Coverage ➔ Impact on Verification Quality
ENHANCE – Test stimulus, irritators, traffic generators, coverage, system configuration, IP configuration

- Event coverage is collected at system level
- Coverage is used to ensure all targeted verification scenarios are covered using the right stimulus
- These scenarios are analyzed for—
  - How many times an event was hit
  - Correlation between tests run and event count
  - Correlation between different architectural and microarchitectural events
  - Ratio between different events

Creating additional stress in the system using Accelerated VIPs

Power and DVFS cycling requests from the SCP

Random Interrupts to CPU

Deadlock & starvation detectors at different points

PCIe traffic using 3rd party IP

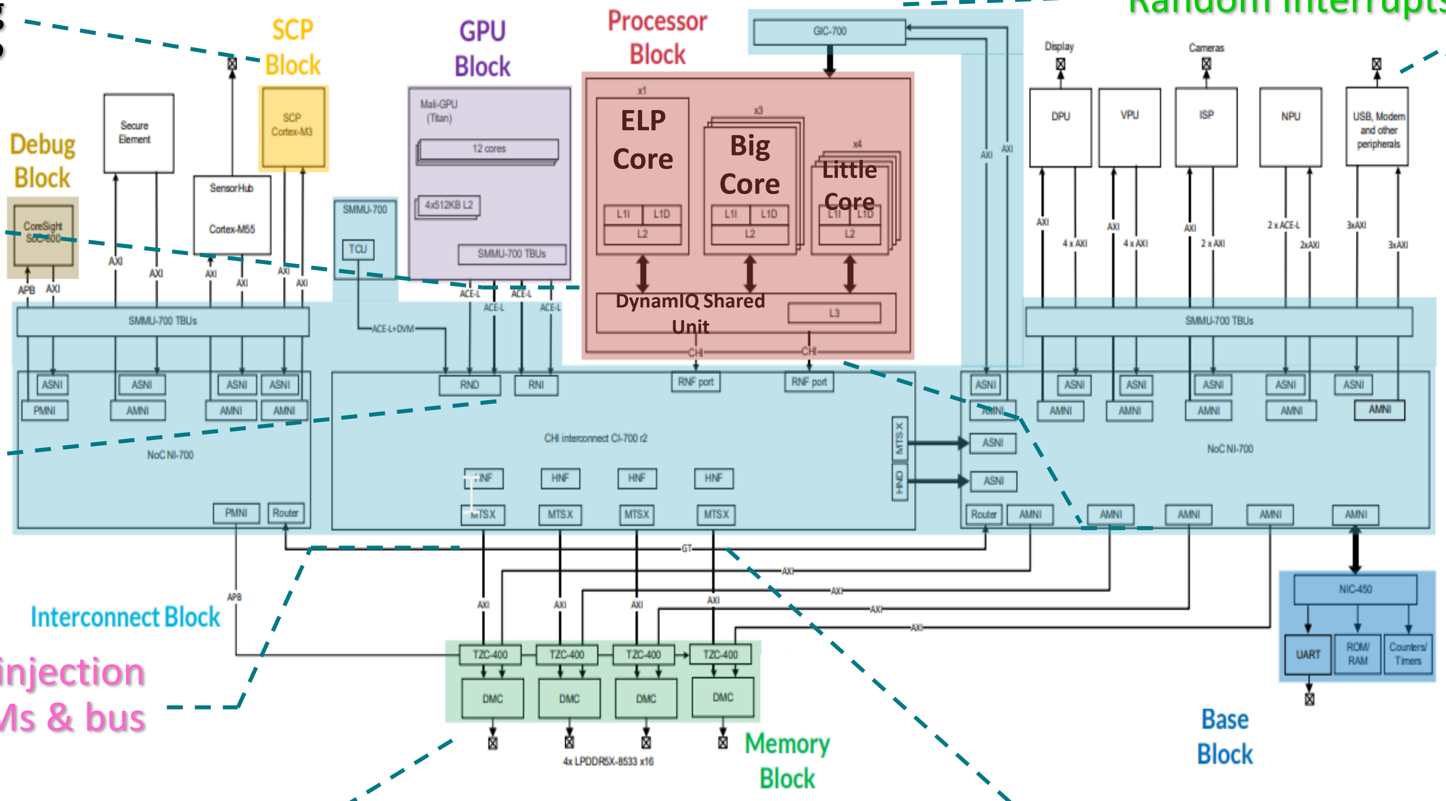
Address & response monitors on the bus to check violations

Traffic on interconnect ports using an accelerated traffic generator VIP

Single bit & double bit error injection in RAMs & bus

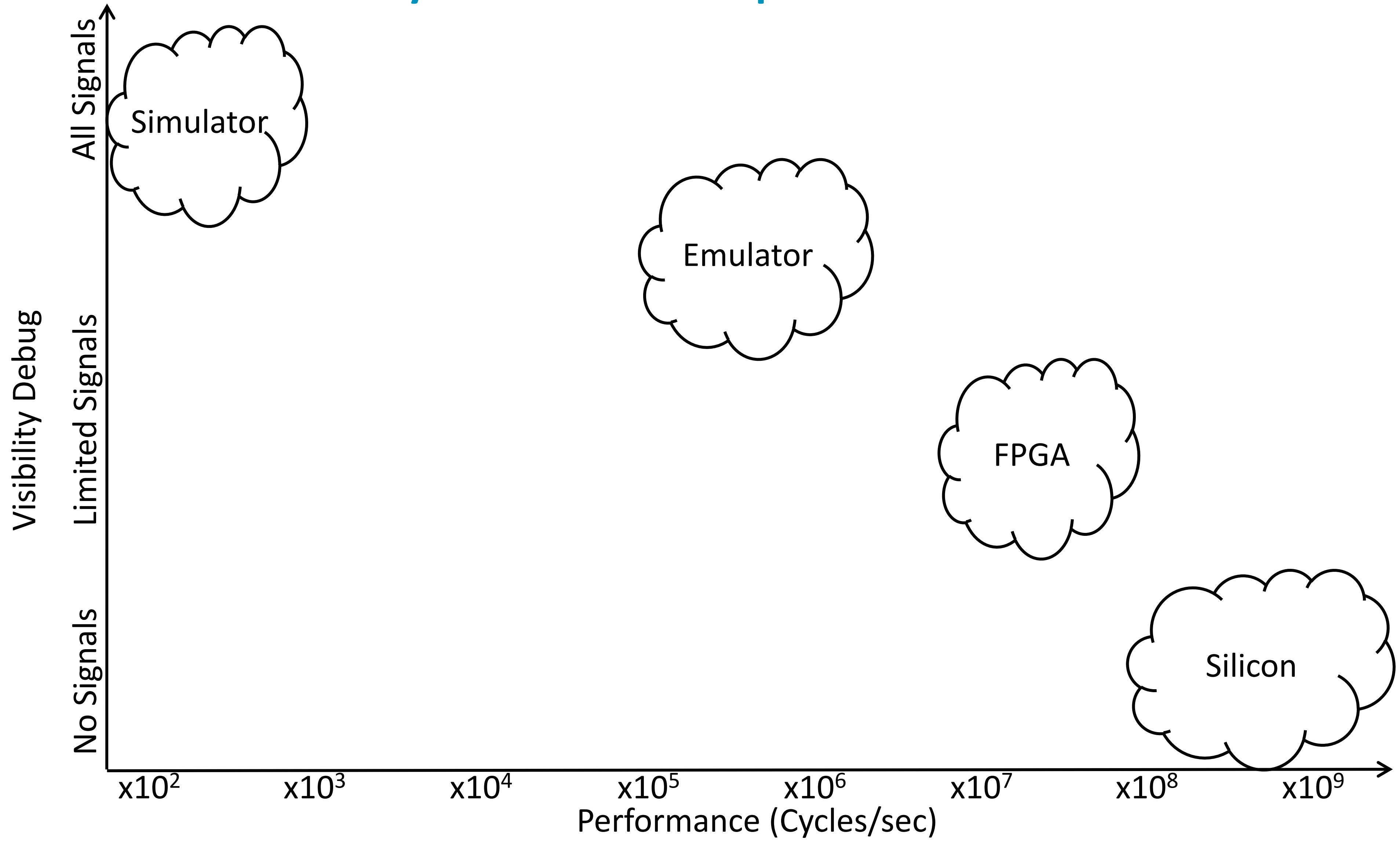
Random delays inserted in memory responses

Device model to check ordering & gathering rules



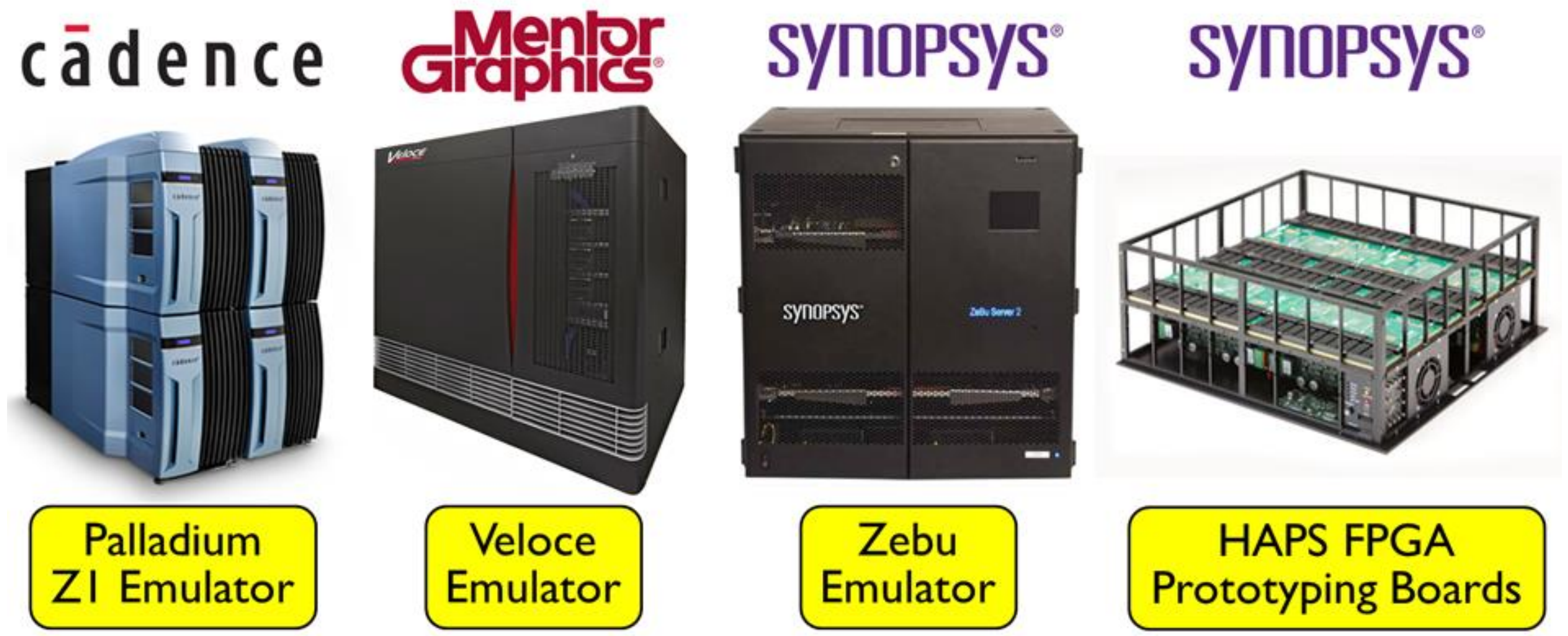
Run on Emulation and FPGA

Performance Visibility Relationship



Platform Methodology

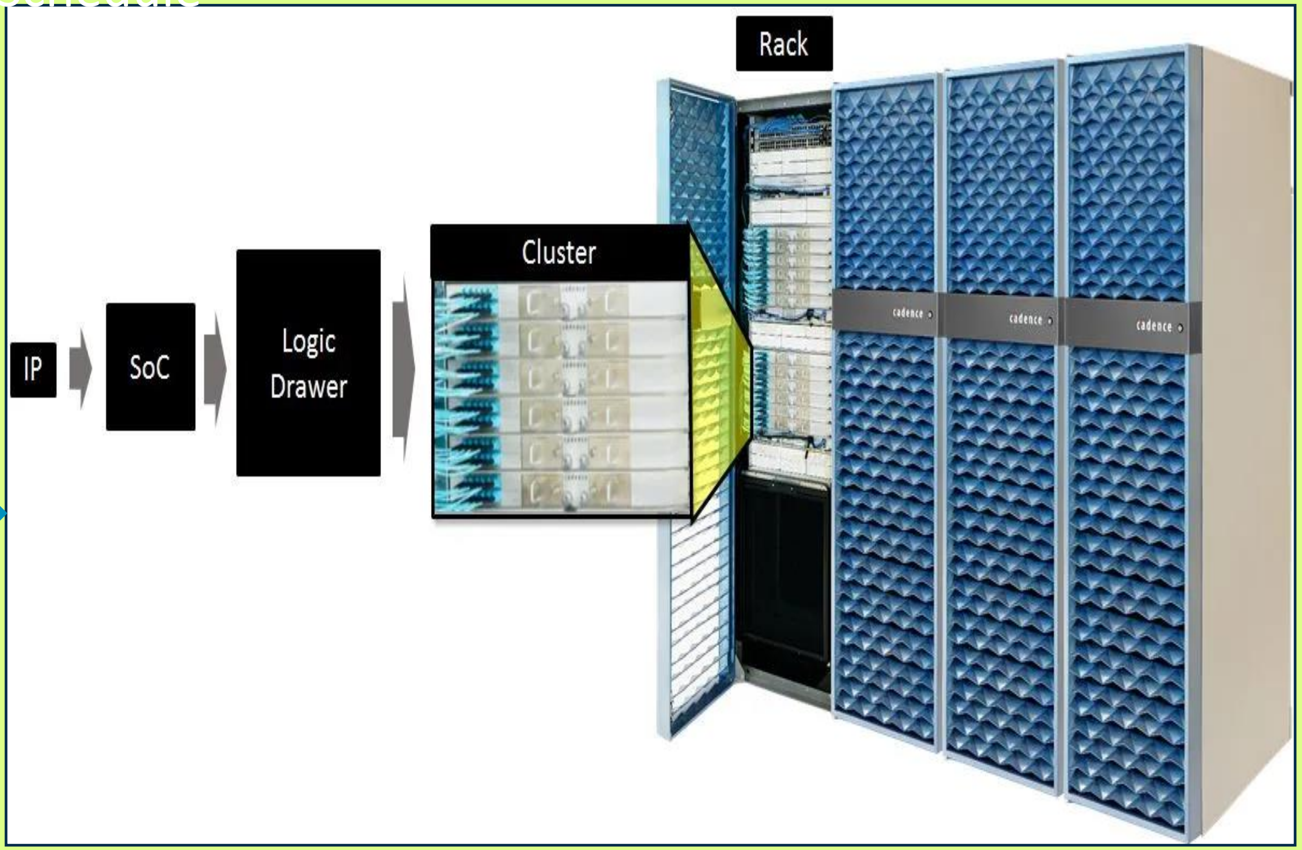
Validation on Emulators & FPGA boards for speed & capacity



- Emulation
  - Palladium (Z1/Z2) => CDNS
  - Veloce (Strato *) => Siemens
  - Zebu (ZS4/ZS5) => SNPS
- FPGA prototype
  - HAPS 100 (SNPS)
  - Primo (Siemens)
- Usage:
  - In-circuit emulation
  - Simulation Acceleration (predominant use case in Arm)

Emulator configuration

LSF &
Schedule



SoC FE – High level Overview

Verification and Validation

Integration
Boot
System Discovery
Memory backplane
Interrupt/MHU
Timer/Wake Up
Cross Chip

Feature Testing
SMMU –Virtualisation & Protection mechanism
Debug
Security
Telemetry
Media pipeline
USB/UFS/DMA

Perf/Power
Perf - CME/NGP
Perf Aanalysis IC - BW/QoS/MPAM
Power Intent
D2D - UCIE

Use case
UCA
Wifi/Modem
OS boot

Stress Validation
RIS based stress
Design mutation
Data path stress

Verification Levels Details

Abstraction	Scope	Techniques Used	Details	BB24 Verification Scope
Solution	Complete Solution: Validation & verification encompassing H/W & S/W as one whole	Emulator, FPGA; C	Multi-Die solution with full S/W stack.	
Performance & Power Analysis	Performance Benchmark test suites. Bare-Metal tests targetting Performance attributes	Emulator, FPGA; C	System tuning for optimal performance under real world traffic workloads. Perform Power Analysis on design under stress with real world workload	
SoC	Interoperability testing covering CSS Die + Companion Die	Emulator, FPGA; C; Formal; UVM; PA	Interoperability between multi-die system. Level of stree testing conducted ensuring no performance bottle-necks	SoC level Verification targets: SMS Element, Memory Subsystem, PLLs IOPAD ring DMA,Traffic Gen
CSS Top-Level	Top-Level CSS with major focus on Scenario driven tests, targeted stress testing, Power Aware tests and Performance Verification	Emulator, FPGA; C; Formal; UVM; PA	Scenario tests focussed on specific features (IO coherency, Sys$, Security etc). Stress testing with data paths excercised concurrently with disruptive DVFC actions. Top-Level connectivity checking.Power Aware (Power transition) testing. Performance verification covering Bandwidth & latency measurements for Initiators and Subordinate blocks and DMC throughput	To check the CSS integrity at Soc: CSS focused test cases targeting different blocks within the CSS will be run at Soc
Block Focussed Top-Level	Small block of a subsystem with specific functionality. Verification of functioanlity can be covered at Top-Level	Formal; UVM; C	Block can be verified at the Top-Level with sorrounding parts of the Top-Level design present. Top-Level TB/Env can be customized to BlackBox parts of design that are not required for testing scenarios OR alternatively keeping those sections in Reset. Traffic generators can be hierarchically attached to relevant interfaces within Top-Level	Block Focussed Top level Verification candidates: ISP, USB, UFS, DPU, VPU, Interrupts, CPN Flexible Hierarchy, CPN Peripheral Element, CPN/Memory Map
Element-Level	Elements integrated at SOC with specific functionality	Formal; UVM; C	The Element will be verified in its own testBench/Environment with the goal to port/integrate into SOC testbench. Example: TPIP Elements could be targeted with Element-level testing	Only new elements integrated at SOC with no legacy Element based testing targetted: USB Element Level UFS Element Level DPU Element Level DDR Element Level
Subsystem-Level	Small Subsystem integrated to DUT with specific functionality	Formal; UVM; C	The subsystem will be verified in its own testBench/Environment. Example: First stage of verification where only the Controller and PHY are verified independently	Subsystem Level verification candidates: USB and DP controller and PHY UFS controller and M-PHY MIPI DSI-2 controller, CD-PHY and pixel convertor
Unit-Level	Full functional verification of units	Formal; UVM	Standalone testbench that verifies the unit exhaustively	Only few candidates for unit level testing Bus Monitors, SMCF RAM integration testing

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה