



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH



**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación

Poster/Demo Sessions Slides

From RISC-V Workshop in Barcelona

7-10 May, 2018

Table of Contents for Poster/Demo Sessions Slides

- Derek Atkins, Slide 3
- Mary Bennett, Slides 4 – 5
- Ekaterina Berezina and Andrey Smolyarov, Slides 6 – 7
- Alex Bradbury, Slide 8
- Luca Carloni and Christian Palmiero, Slides 9 – 10
- Jie Chen, Slides 11 – 13
- Matt Cockrell, Slides 14-26
- Alberto Dassatti, Slides 27 – 28
- Christian Fabre, Slides 29 – 30
- Juan Fumero, Slides 31 – 32
- Nicolas Gaudé and Hai Yu, Slides 33 – 36
- Chris Jones and Zdenek Prikryl, Slides 37 – 38
- Felix Kaiser, Slides 39 – 40
- Alexander Kamkin and Andrei Tatarnikov, Slides 41 – 42
- Luke Leighton, Slide 43
- Heng Lin, Slides 44 – 45
- Maja Malenko, Slide 46
- Eric Matthews and Lesley Shannon, Slides 47 – 48
- Paulo Matos, Slides 49 – 50
- Lucas Moraes, Slides 51 – 52
- Mauro Olivieri, Slides 53 – 54
- Aleksandar Pajkanovic, Slides 55 – 56
- Matheus Ogleari, Slides 57 – 75
- Shubhdeep Roy, Slides 76 – 78
- Boris Shingarov, Slides 79 – 80
- Christoph Schulz, Slides 81 – 82
- Wei Song, Rui Hou and Dan Meng, Slides 83 – 84
- Greg Sullivan, Slides 85 – 86
- Robert Trout, Slide 87
- Vasily Varaksin and Ekaterina Berezina, Slides 88 – 89
- Danny Ybarra, Slides 90 – 97

Fast, Quantum-Resistant Secure Boot Solution

For RISC-V MCUs with off-chip program stores

Demo at Poster Session: *Mynewt* RTOS on *HiFive1* development board

Run-time metrics

Method	Security Level	Verification Time
ECDSA P256	128-bit	179 ms
ECDSA P521	256-bit	(not supported)
WalnutDSA	128-bit	6.76 ms
WalnutDSA	256-bit	33.6 ms

CPU Clock: 256 MHz

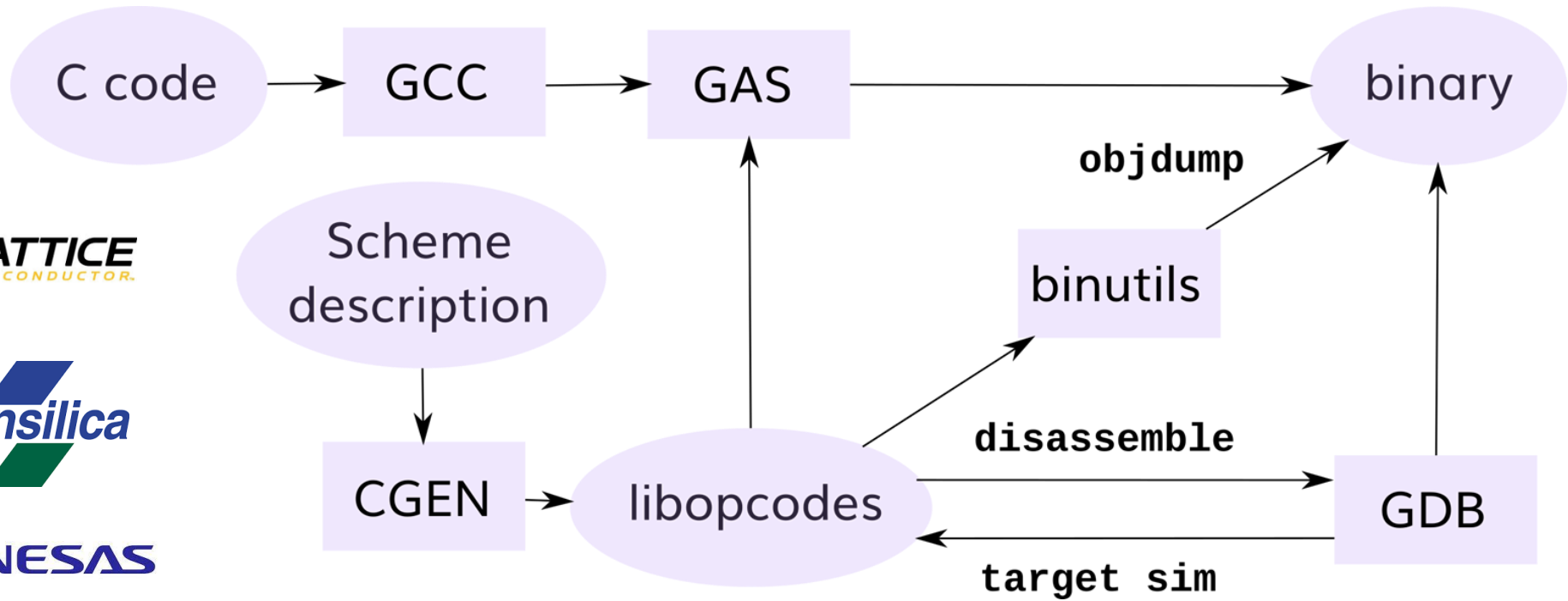
RTOS: Mynewt 1.3.0

Powered by Walnut Digital Signature Algorithm™

- Up to 40X faster than ECDSA at 128-bit security
- Small footprint: easily fits in FE310's 8K byte OTP ROM
- Based on Group Theoretic Cryptography (GTC)
- GTC Leverages
 - Structured groups
 - Matrices and permutations
 - Arithmetic over finite fields



What is CGEN



How does it work?

```
(define-insn
  (name "add")
  (comment "register add")
  (attrs base-isas)
  (syntax "add ${rd},${rs1},${rs2}")
  (format + (f-funct7 funct7) rs2 rs1
    (f-funct3 funct3) rd (f-opcode opcode))

  (semantics (set DI rd (add DI rs1 rs2))))
)
```


About Syntacore

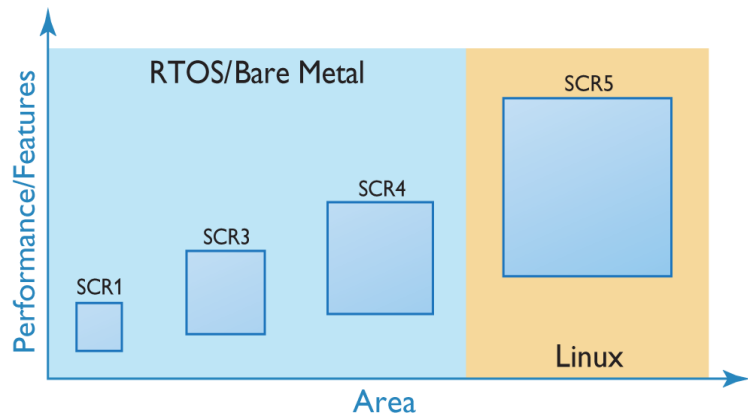
IP company, founding member of RISC-V foundation

Develops and licenses state-of-the-art RISC-V cores

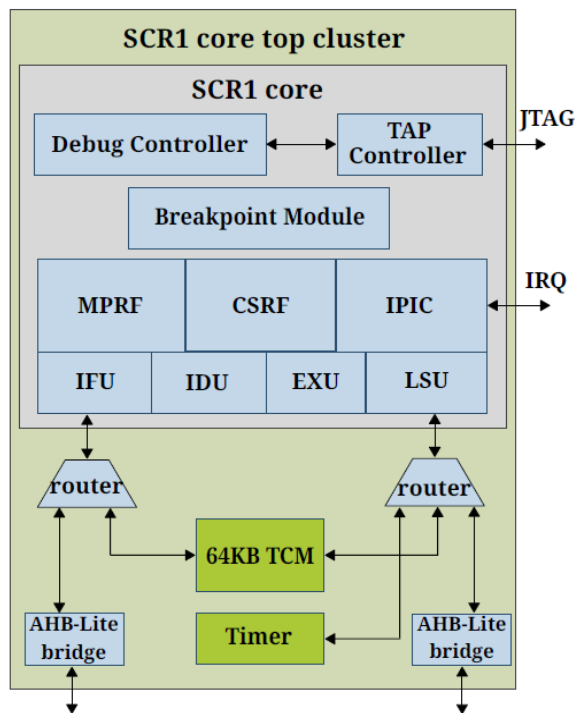
- Initial line is available and shipping to customers
- SDKs, samples in silicon, full collateral
- Core team comes from 10+ years of highly-relevant background
- 2.5+ years of *focused* RISC-V development

Full service to specialize CPU IP for customer needs

- One-stop workload-specific customization for 10x improvements
 - with tools/compiler support
- IP hardening at the required library node
- SoC integration and SW migration



SCR1 overview



Compact MCU core for deeply embedded applications

- Open source under SHL-license since May 2017 (Apache 2.0 derivative with HW specific)
 - Unrestricted commercial use allowed
- RV32I|E[MC] ISA
- <15 kGates in basic RV32EC configuration
- 2 to 4 stages pipeline
- M-mode only
- Optional configurable IPIC: 8..32 IRQs
- Optional integrated Debug Controller
 - OpenOCD based
- Verification suite
- Documentation
- Best-effort support provided
- Commercial support available

Performance*, per MHz	DMIPS	-O2	1.28
		-best**	1.72
	Coremark	-O2	2.60
		-best**	2.78

* Dhrystone 2.1, Coremark 1.0, GCC 7.1 BM from TCM
 ** -O3 -funroll-loops -fpeel-loops -fcommon -fdata-sections -ffunction-sections -fno-common -fno-lto

Diving into RISC-V LLVM

- Why RISC-V LLVM
- Current status
- Approach
- Next steps
- Supporting your own extension
- Get involved



Alex Bradbury
asb@lowrisc.org @asbradbury @lowrisc

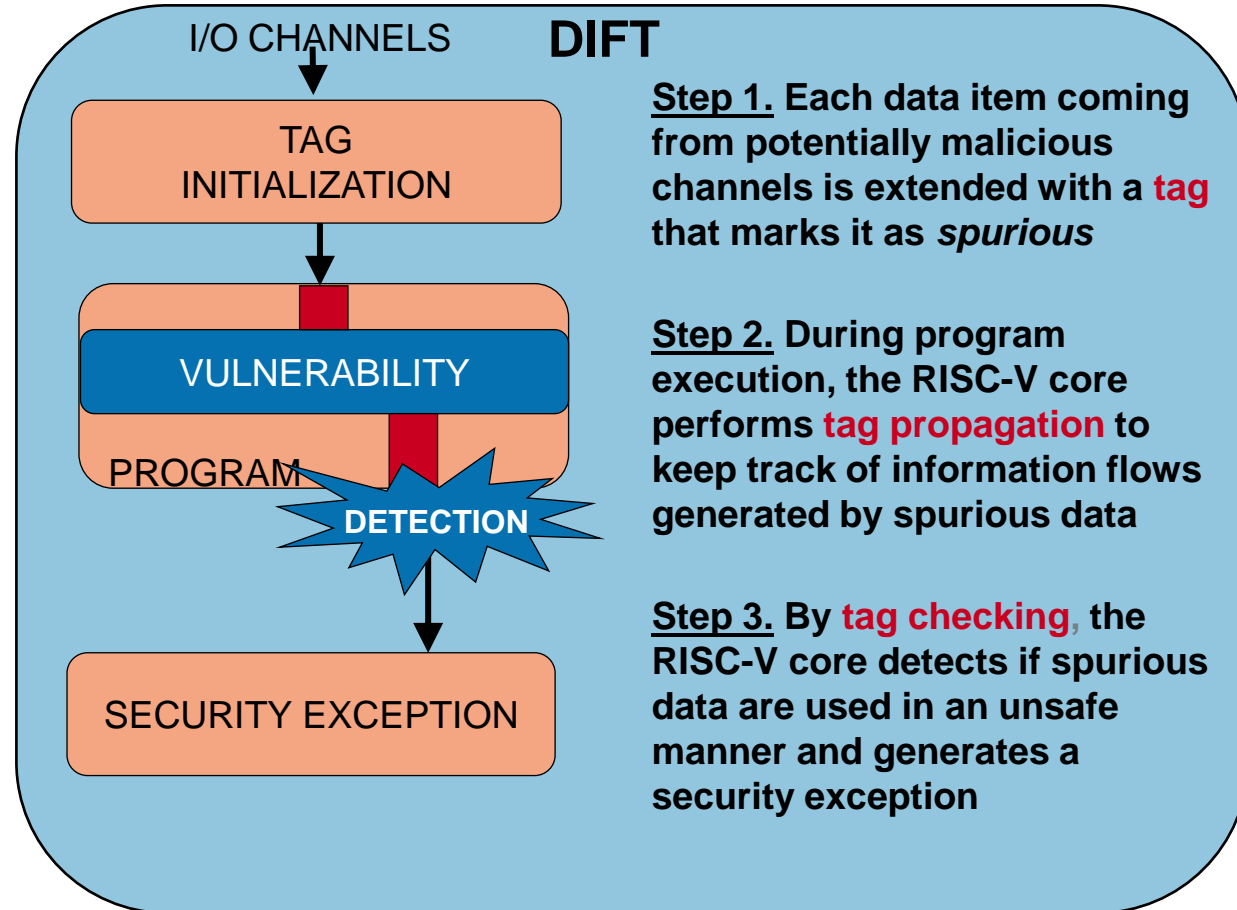
Securing a RISC-V Core for IoT Applications with Dynamic Information Flow Tracking (DIFT)

Motivation

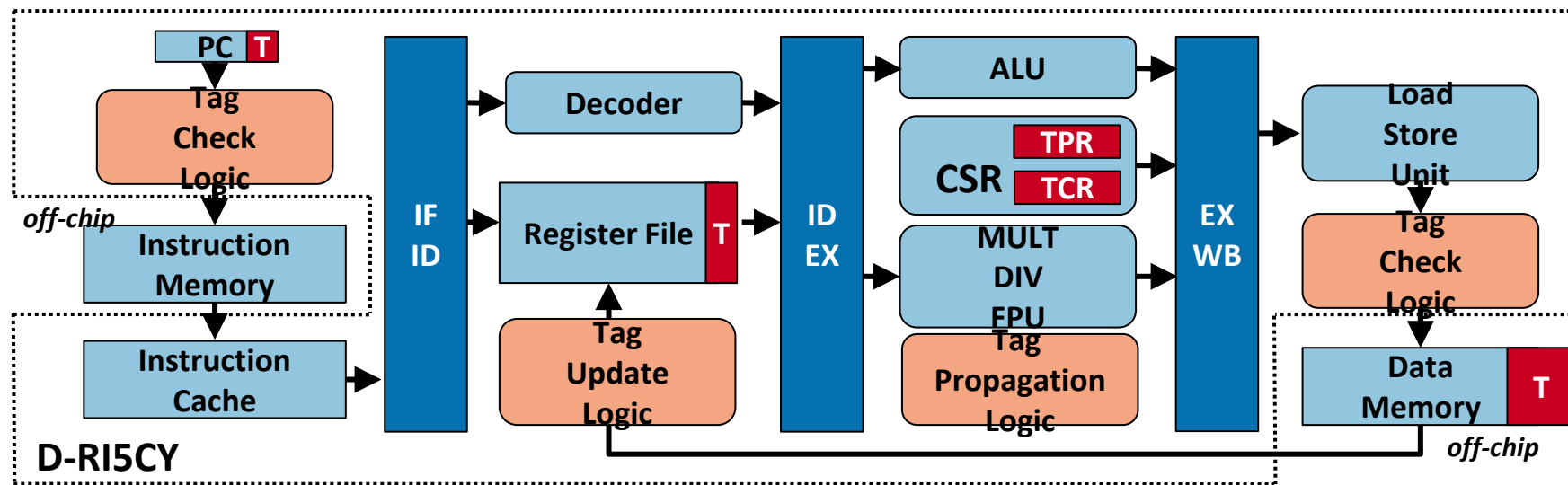
- Many IoT devices are prone to attacks due to low-level programming errors (e.g. buffer overflow and format string attacks)

Contributions

- Design and implementation of a low-overhead protection scheme based on DIFT to secure RISC-V cores for IoT applications
- Demonstration with an FPGA-based prototype

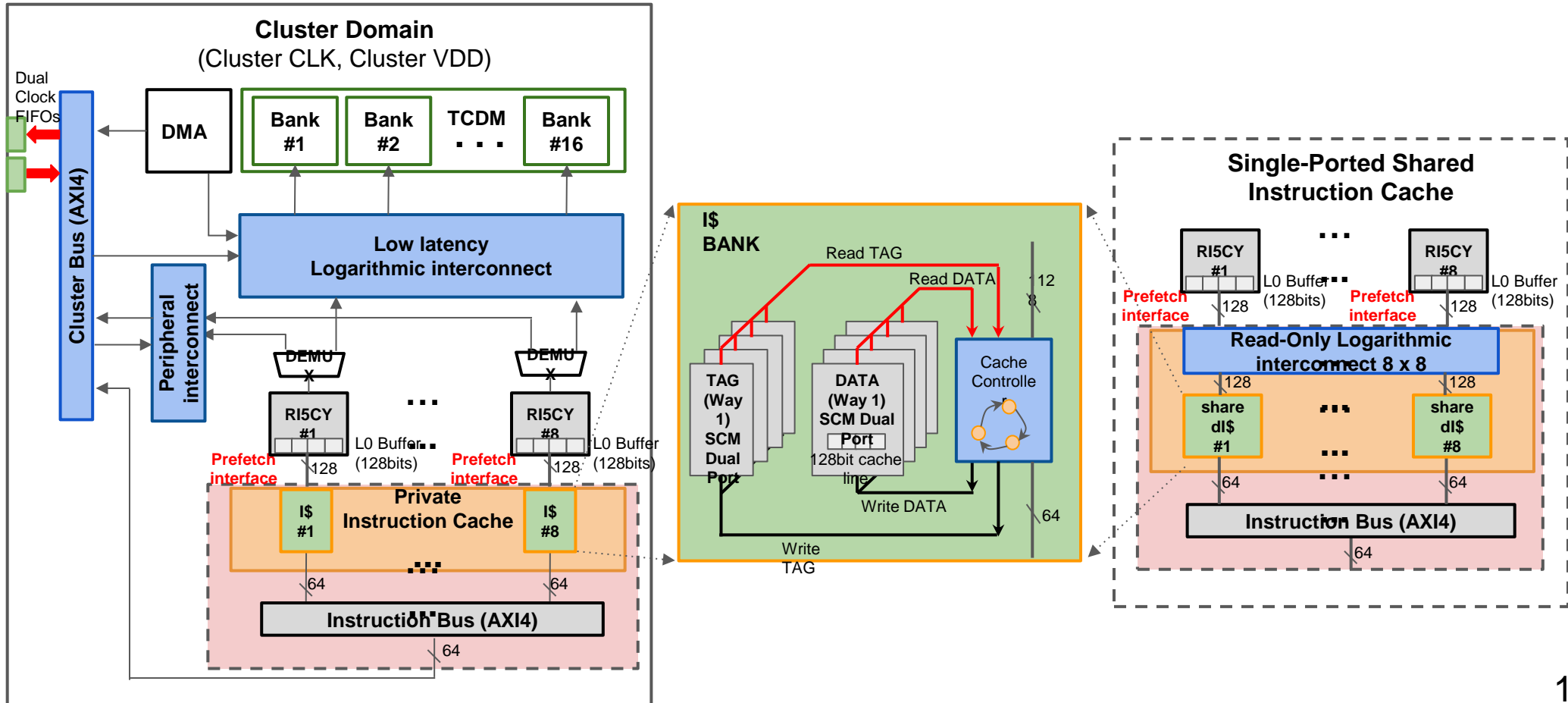


Design, Implementation, and Evaluation of the DIFT-Enhanced RISC-V Processor Core



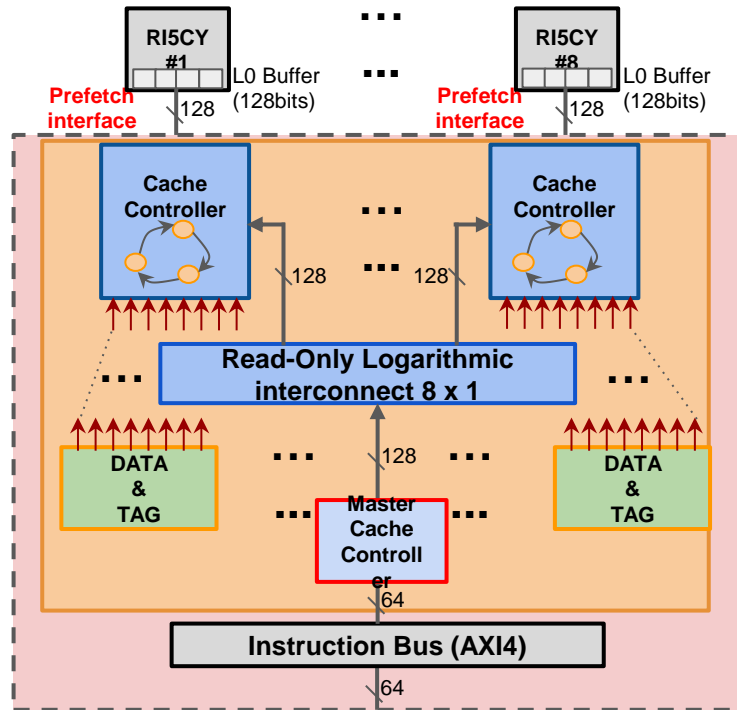
- Design and implementation of a DIFT architecture for an optimized 4-stage, in-order, 32-bit RISC-V core based on the PULPino platform
- The D-RISC-V architecture supports various software-programmable security policies; it was evaluated with policies for memory protection
- The experimental results demonstrate that securing a RISC-V core with DIFT is feasible, does not incur in any run-time overhead, and requires negligible resources

Ultra Low Power Cluster I\$ exploration with RI5CY

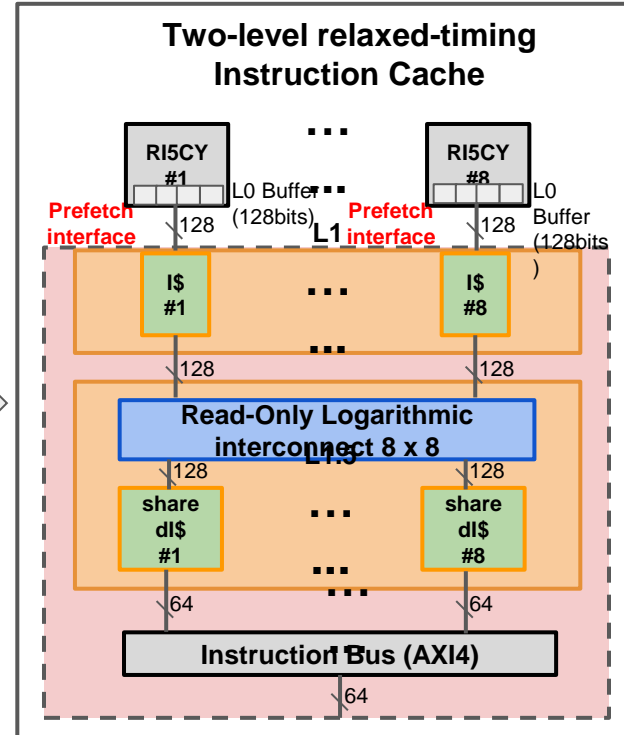


Two-Level Icache & Results

Multi-Port Shared Instruction Cache



Two-level relaxed-timing Instruction Cache



IS type	SP	MP	Two-Level
Characteristics			
Throughput	-5%	1.00	-15% ^[1]
Area	1.00	+50%	+40%
Comparison between throughput, area and power efficiency among SP, MP and two-level icache. 1.00 means the best among the three types of icache.			1.00

[1] For throughput, the value of two-level icache is the worst case.

All value in the table are the average value except [1]. For more details and data, please see the poster.

REFERENCES

- [1] I. Loi, A. Capotondi, D. Rossi, A. Marongiu and L. Benini, "The Quest for Energy-Efficient I\$ Design in Ultra-Low-Power Clustered Many-Cores," in IEEE Transactions on Multi-Scale Computing Systems, vol. PP, no. 99, pp. 1-1.
- [2] L. Benini, E. Flamand, D. Fuin, and D. Melpignano, "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," in 2012 Design, Automation Test in Europe Conference Exhibition (DATE), March 2012, pp. 983–987.
- [3] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Power, Area, and Performance Optimization of Standard Cell Memory Arrays Through Controlled Placement," ACM Trans. Des. Autom. Electron. Syst., vol. 21, no. 4, pp. 59:1–59:25, May 2016. [Online]. Available: <http://doi.acm.org/10.1145/2890498>

Evaluation of RISC-V for Pixel Visual Core

Matt Cockrell (mcockrell@google.com)

(May 9th, 2018)



Overview

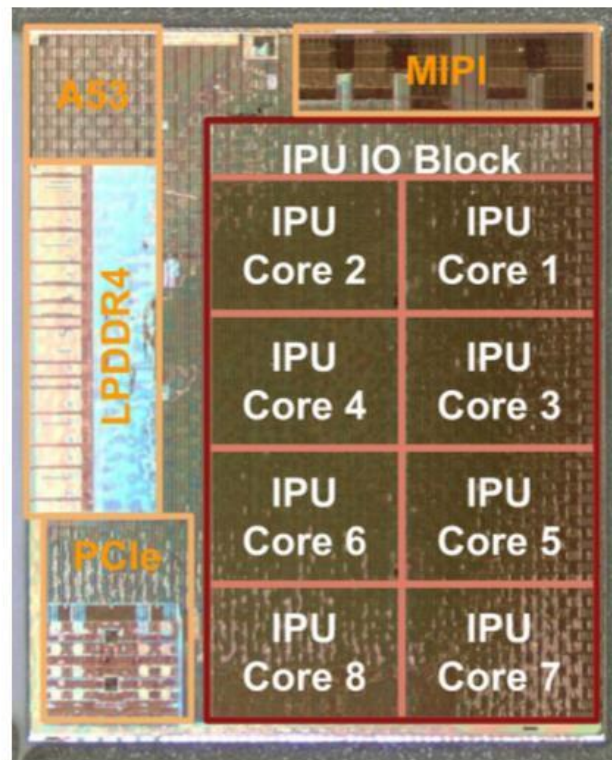
- Use case
- Core selection
- Integration

Background: Pixel Visual Core

Proprietary + Confidential

“Pixel Visual Core is the Google-designed Image Processing Unit (IPU)—a fully programmable, domain-specific processor designed from scratch to deliver maximum performance at low power.”[1]

Critical point: A dedicated A53 (top left) aggregates application layer IPU resource requests and configures appropriately.

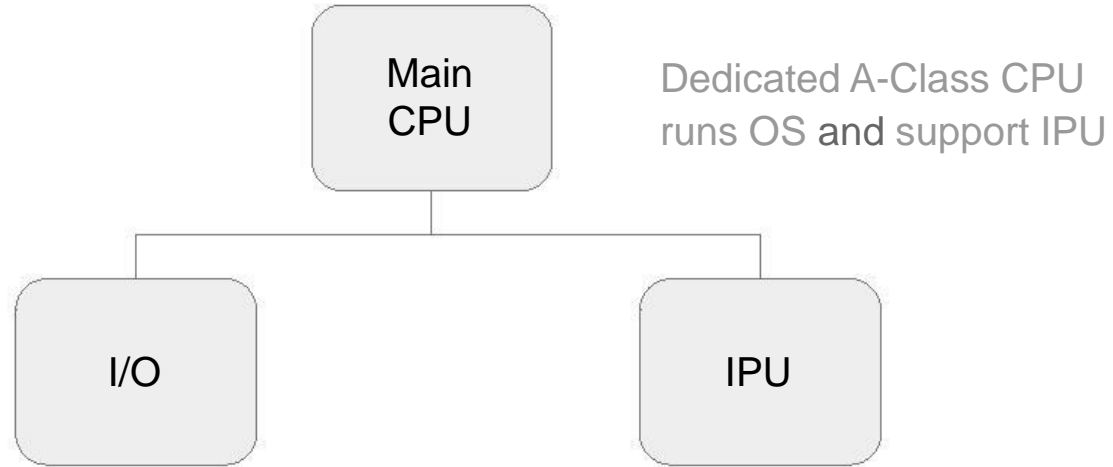


Magnified Image of Pixel Visual Core

[1] “Pixel Visual Core: image processing and machine learning on Pixel 2”, Oct 17, 2017, Ofer Shacham, Google Inc.

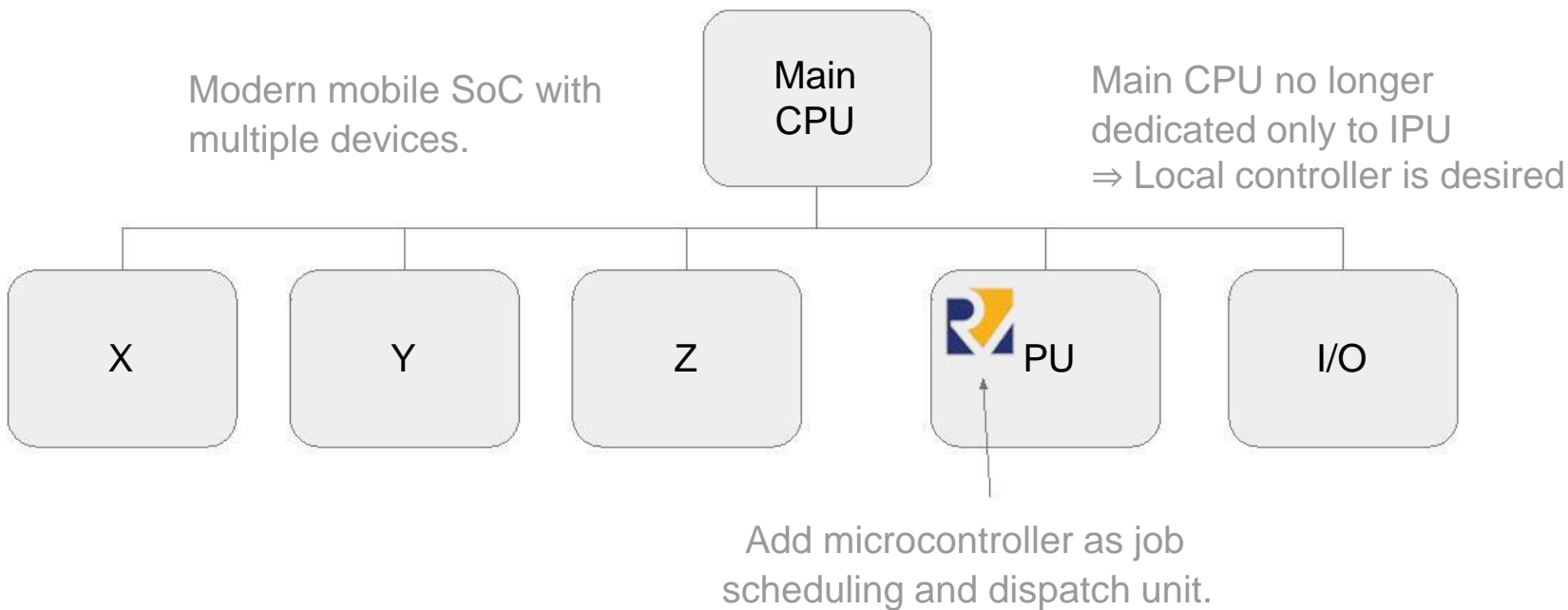
Pixel Visual Core today

Proprietary + Confidential



Modern SoC - multiple accelerators

Proprietary + Confidential



Open source IP is an interesting option for this microcontroller.

Level of Effort

How difficult would it be to work with and integrate.

Risk

Stability and reliability of support.

License

Flexibility of use.

Candidate 1: Bottle Rocket [\(<https://github.com/google/bottlerocket>\)](https://github.com/google/bottlerocket)

- Internal Project to demonstrate ability to easily develop custom RISC-V implementation by leveraging Rocket Chip.
- Implements RV32IMC
- Represents evaluating Rocket Chip as an option.



Candidate 2: Merlin (<https://github.com/origintfj/riscv>)

- Core provided from a hobbyists developed compatible with “QFlow”
- Implements RV32IC
- The hobbyist was a team member, use of this core would become a “build from scratch” candidate.





shutterstock.com · 219171082

Candidate 3: RI5CY (RISK-EE) <https://github.com/pulp-platform/riscv>

onfidential

- Provided from the PULP team
- Implements RV32IMC with added extensions
- This candidate comes from and is maintained by academia

Open core comparison

Core	Level of Effort	Risk	License
Bottle Rocket 	High	Med	✓
Merlin	Low	High	✓
RI5CY 	Low	Med	✓

Recommended Candidate

Proprietary + Confidential

Selected RI5CY from PULP:

- Had been taped-out
- Provided infrastructure
- Solderpad license
- It was implemented in SystemVerilog (instead of Chisel):
 - SystemVerilog builds on established physical design and verification flows
 - Chisel generated Verilog loses designer's intent making it difficult to read and debug
 - Chisel generated code makes certain physical design items difficult such as sync/async clocks, power domains, clock domains, etc.

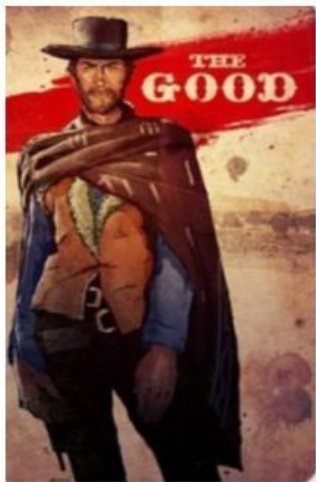


PULP
Parallel Ultra Low Power

Integration of RI5CY

The Good:

- RTL provided in SystemVerilog
- ETH/PULP Team
- Debug capability
- Able to work with Valtrix to verify
- Documentation



The Bad:

- Numerous lint errors
- Ad hoc verified
- Bugs found:
 - PULPino Compiler
 - Documentation
 - Extensions
- Debug setup requires PULPino specific utilities.



The Ugly (Scary):

- Version control
- Bugs found in:
 - Multiplier
 - LSU



Recap and next steps

Where we have been:

- Describe possible PVC configuration mechanism
- Continued evaluation of RI5CY
- Shared experience of integrating open source IP

Where we are going:

- Add full compliance for privilege/debug specification.
- Evaluate performance impact after adding RI5CY to PVC

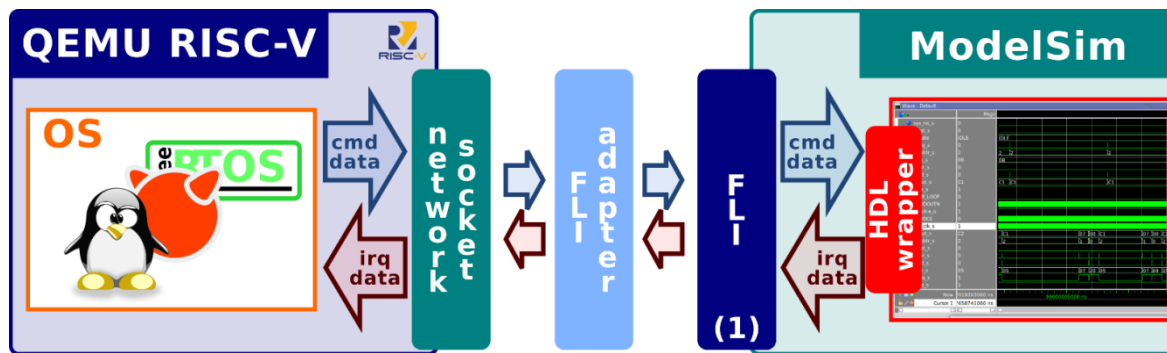
Heterogeneous systems incorporating **custom HDL designs** usually rely on **mock systems**

- additional effort required
- simulated data/scenarios
- **no visibility** on HW-SW interactions



TCCF: QEMU/Modelsim-based co-simulation framework

- ✓ **full visibility** on internals during **real interactions**
- ✓ host software and HDL **unmodified**



(1) Foreign Language Interface

TCCF: Tightly-Coupled Co-simulation Framework for RISC-V Based Systems

X. Ruppen, R. Rigamonti, A. Dassatti

Adding support for a new HDL design:

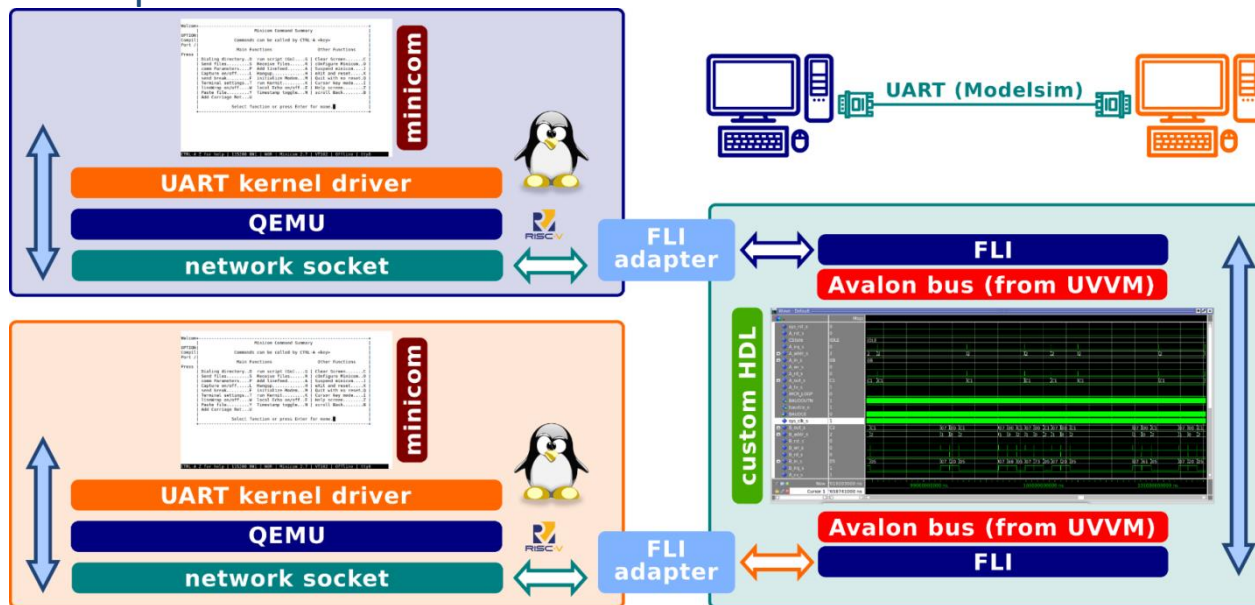
≈ 15 minutes of work!



Bus Functional Models from UVVM

- AXI4-Lite
- AXI Stream
- Avalon MM
- I2C
- ...

Example:



Freely available on GitLab!



Free as in Freedom

<https://gitlab.com/reds-public/tccf>

MOTIVATION AND PROPOSAL

- **Motivation**

- Approximate Computing (AC) exploits **error resiliency** of applications.
- There is currently **no support for AC** in RISC-V processors

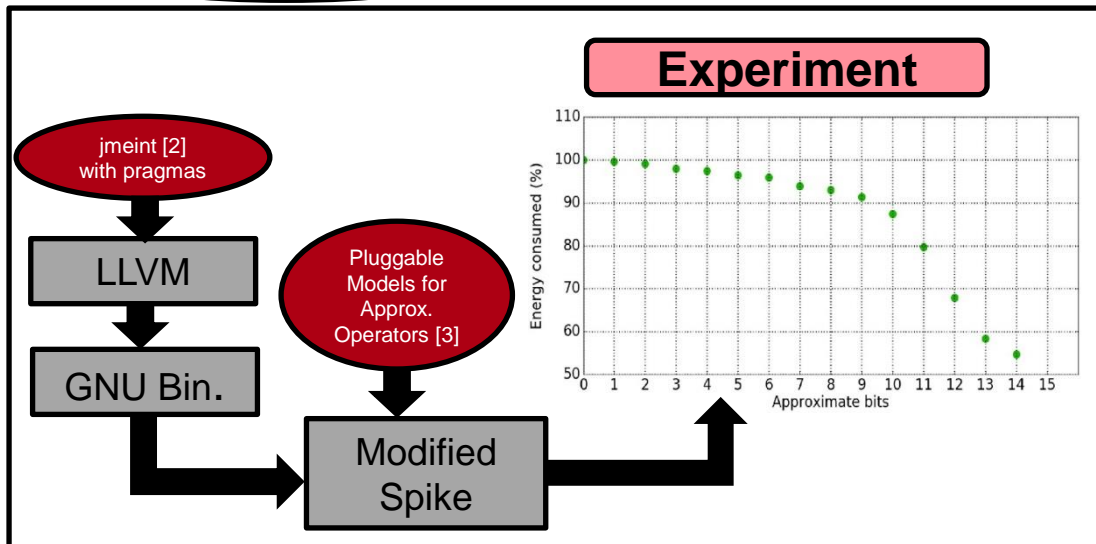
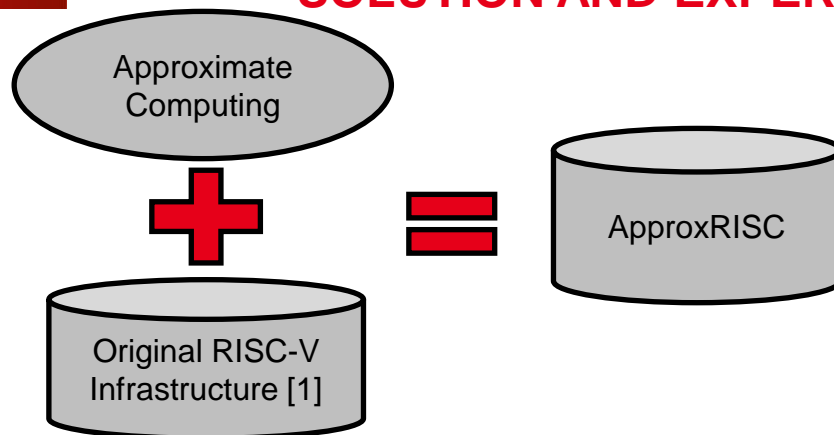
- **Proposal**

- ApproxRISC, an infrastructure for AC in RISC processors, made of
 - ISA Extension: 12 instructions, global state for approximate bit width
 - Simulator with pluggable models for approximate operators
 - LLVM version 3.9 and GNU Binutils

- **ISA Extension**

- Set of integer-type **instructions for approximate operations**.
 - Register/register, register/immediate
- Change accuracy bit width
- Operations supported: Addition, subtraction, multiplication and division.

SOLUTION AND EXPERIMENTS



LLVM

Pragmas for AC

```
#define ABW 7
#define WIDTH 32
#define FRAC 10
float example_full_approx() {
    float a = 44.23, b = 2300.12, d, e = 1000;
    #pragma full_approx(a, b) ABW (WIDTH, FRAC) True
    {
        d = a + b + e + 1;
    }
    return e;
}
```

Future Works

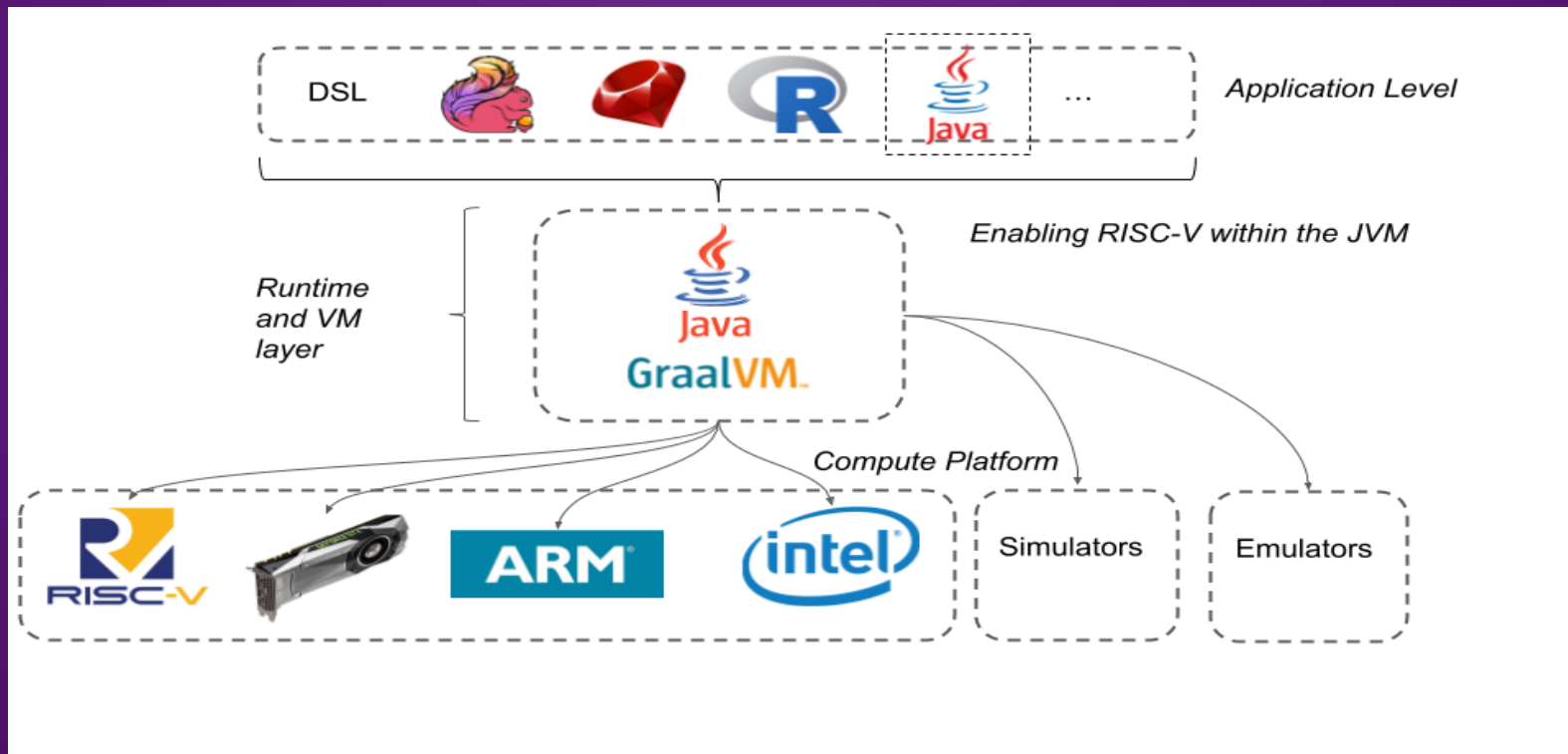
- Thorough experimentation to show the benefit of our infra
- Exploration of new instructions

References

- [1] RISC-V Foundation — Instruction Set Architecture (ISA). URL: <https://riscv.org>.
- [2] A. Yazdanbakhsh et al. "AxBench: A Multiplatform Benchmark Suite for Approximate Computing". In: IEEE Design Test
- [3] D. J. Pagliari et al. "A methodology for the design of dynamic accuracy operators by runtime back bias". In: DATE, 2017.

Enabling RISC-V support on MaxineVM

F. Zakkak, J. Fumero and C. Kotselidis



Enabling RISC-V support on MaxineVM

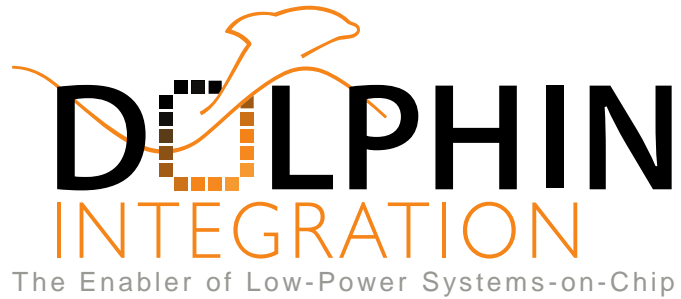
F. Zakkak, J. Fumero and C. Kotselidis

Maxine-VM:

- Meta-circular research VM for Java
- Multiple JIT-compilers (JMCI compatibility)
- Multiple GC algorithms (MMTk compativility)
- Multiple ISAs (x86_64, ARMv7, Aarch64)
- Cross-ISA testing framework

Maxine-VM RISC-V status:

- Cross-ISA testing framework ported to RISC-V
- Created RISC-V assembler skeleton
- Active RISC-V assembler development



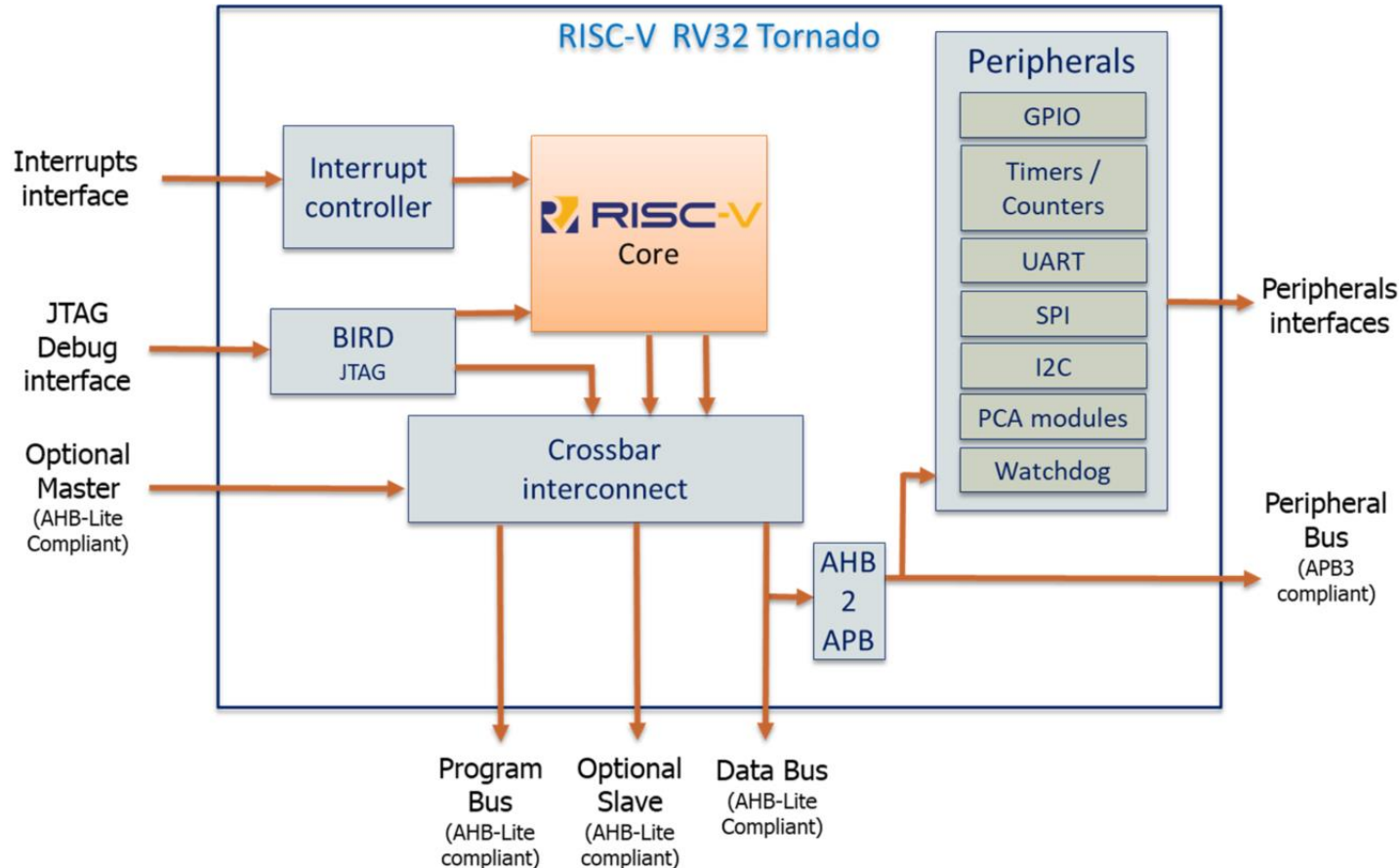
Tornado: an open platform for energy-efficient SoCs based on RISC-V



BARCELONA, 8TH OF MAY 2018

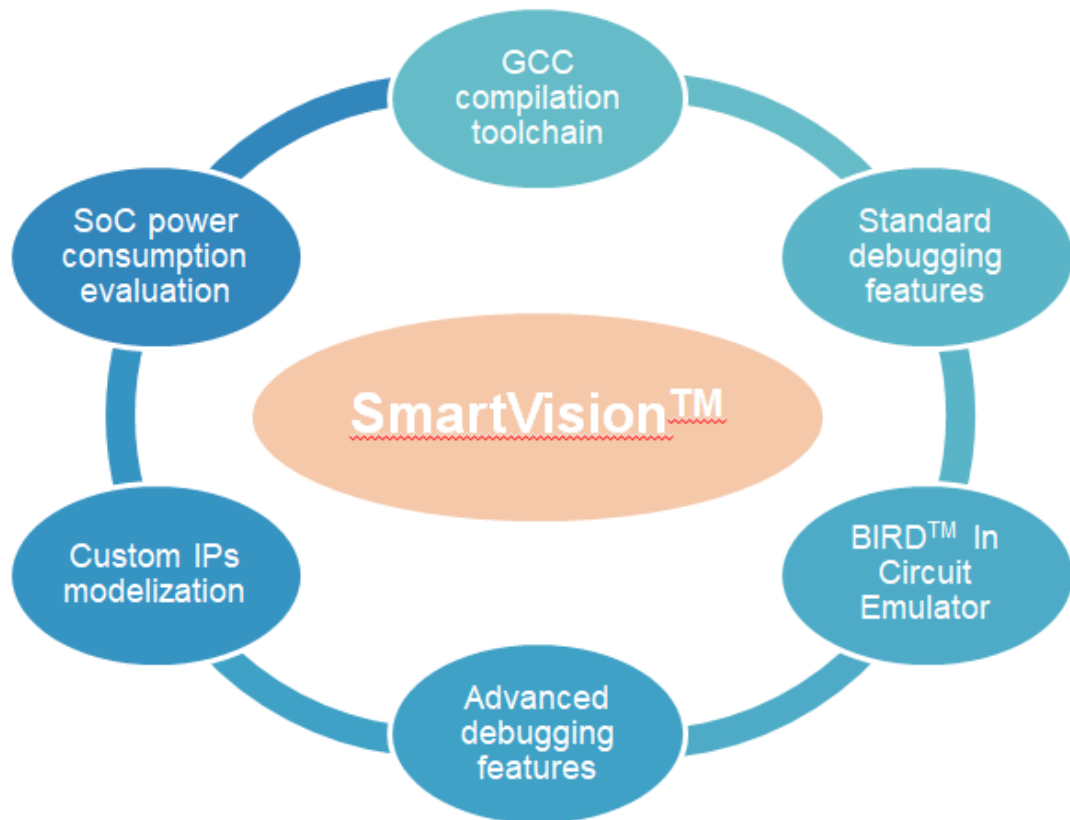
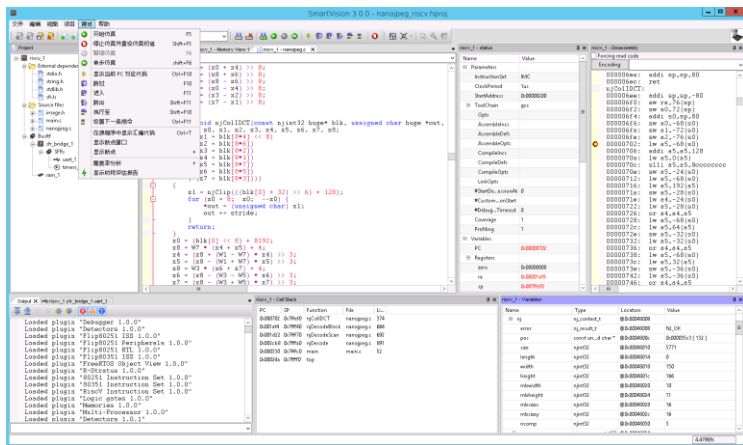


Not just a supplier of Technology, but provider of the Dolphin Integration **know-how!**



- Standard AMBA interfaces
- Fully configurable
- Ready to boot SoC
- Software library
- Low level drivers
- RTOS
- Virtual platform

- Integrated and graphical solution for RISC-V SoC
- Complete RISC-V SoC modelling
- Power consumption modelling, estimation and optimization
- Custom IPs virtual models
- Open APIs



Thank you for your attention,

Let's meet and discuss around our poster

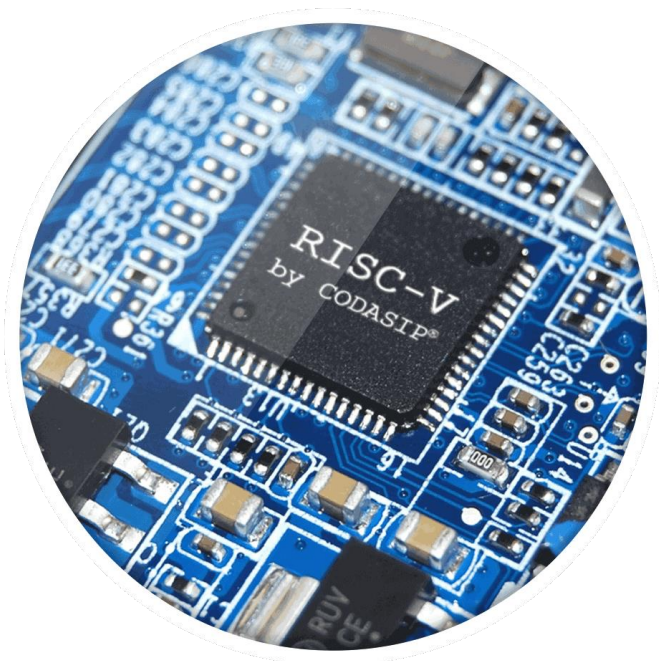




THE LEADING PROVIDER OF RISC-V PROCESSOR IP



THE FREE AND OPEN RISC
INSTRUCTION SET
ARCHITECTURE



Ahead of Game: Codalip introduced
its first RISC-V processor in
November 2015

Codalip Bk: A portfolio of RISC-V processors

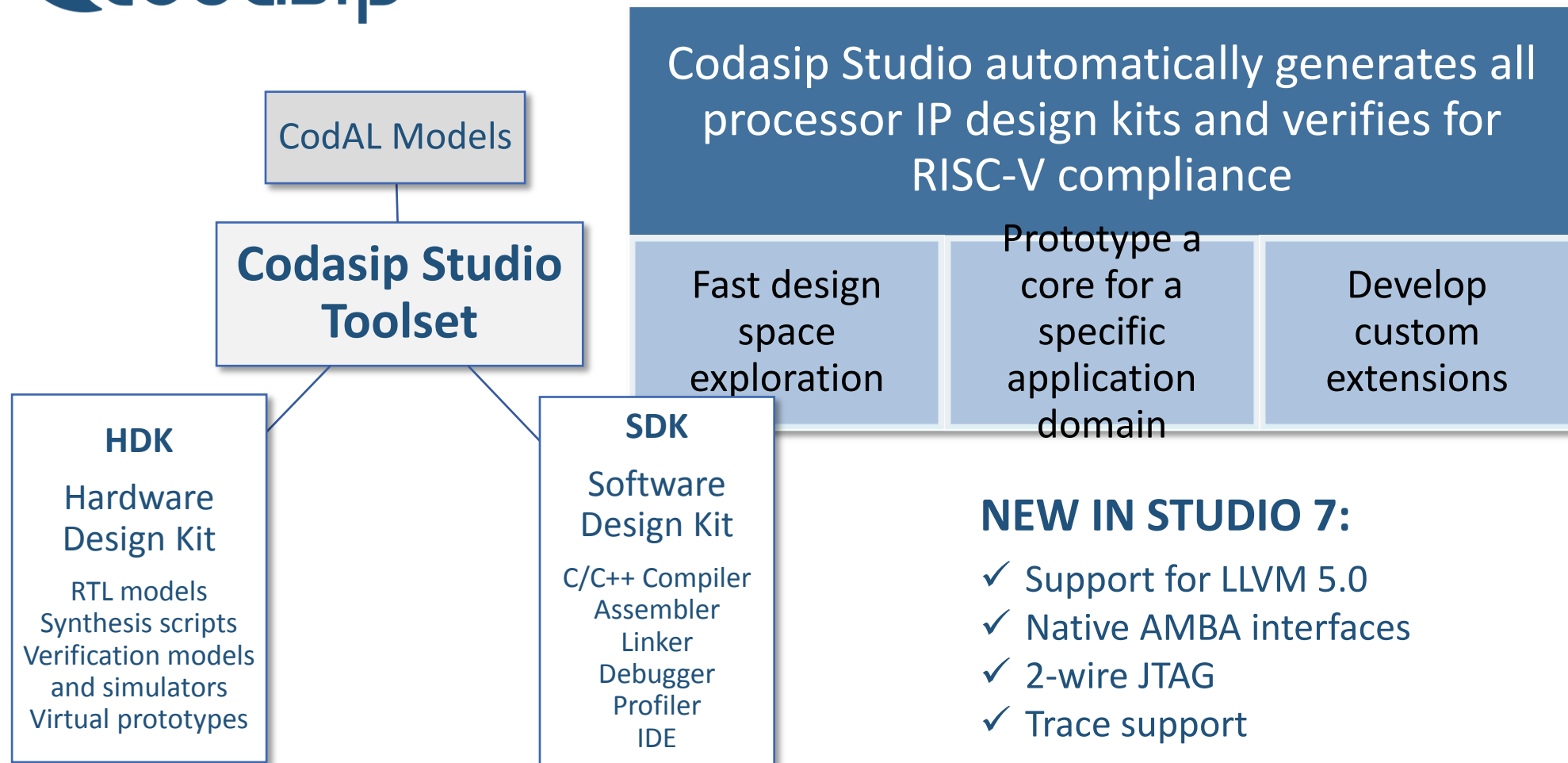
Unique design automation tools that allow
users to easily modify RISC-V processors

Performance,
power
efficiency, low-
cost

Algorithm
accelerators
(DSP, security,
audio, video...)

Profiling of
embedded SW
for IP
customization

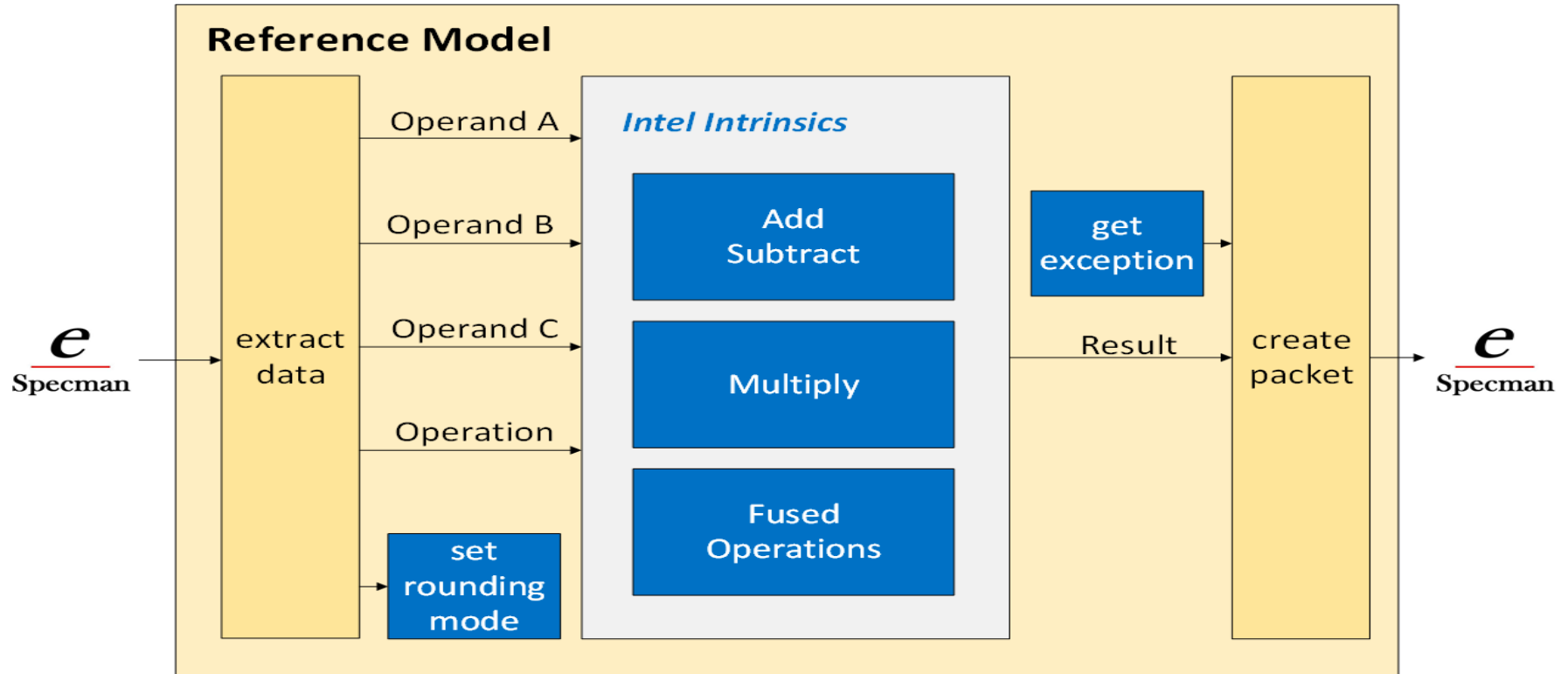
NEW: CODASIP STUDIO 7



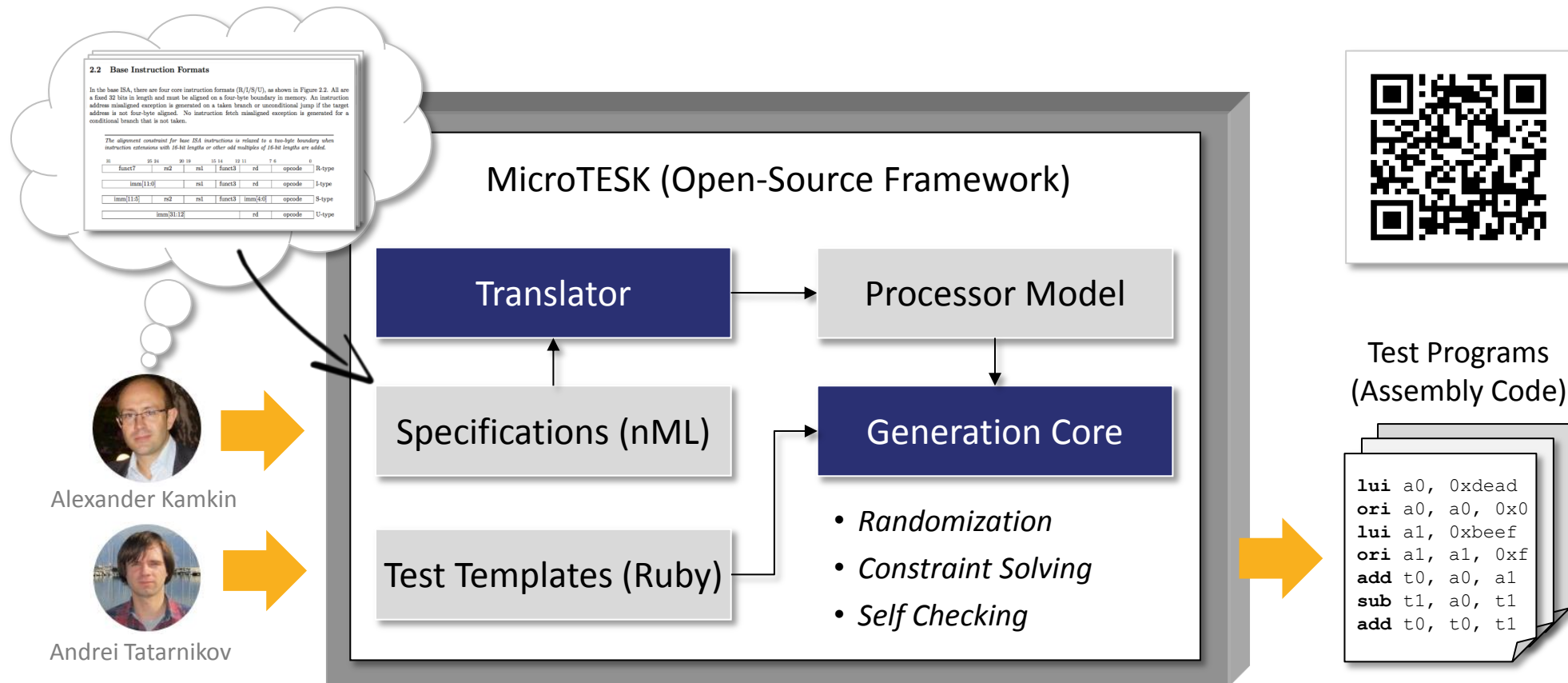
- **Task:** Development of a simulation-based Verification Environment for our RISC-V-conform arithmetic Floating-Point Unit
- **Implementation** in Specman e using the Universal Verification Methodology
- **Issue:** Finding a "known good" and error-free referencemodel
- **Solution:** Using the Intel processor which hosts the Simulation runs



Automated Verification of RISC-V-conform Floating-Point Modules



Test Generator **MicroTESK** for **RISC-V**



Alexander Kamkin

Andrei Tatarnikov

RISC-V Specifications and Test Templates

Instruction Set Architecture

RISC-V

RISC-V Instruction Set Manual
Volume I: User-Level ISA (v. 2.2)

145 pages

Specified Instructions

226 instructions

ISA Specifications

3300 LOC

// ISA Specification in nML

```
op add(rd: X, rs1: X, rs2: X)
```

```
  syntax = format("add %s, %s, %s",  
    rd.syntax, rs1.syntax, rs2.syntax)
```

```
  image = format("0000000%s%s000%s0110011",  
    rs2.image, rs1.image, rd.image)
```

```
  action = {  
    rd = rs1 + rs2;  
  }
```

Test Program Template in Ruby

```
class MyTemplate < RiscVBaseTemplate  
  def run  
    block(:combinator => 'product') {  
      iterate {  
        xor x(_), x(_), x(_)  
        lui x(_), _  
      }  
      iterate {  
        and x(_), ...  
        or x(_), ...  
      }  
      iterate {  
        auipc x(_), _  
      }  
    }.run      2 × 2 × 1 = 4  
  end          test cases  
end
```

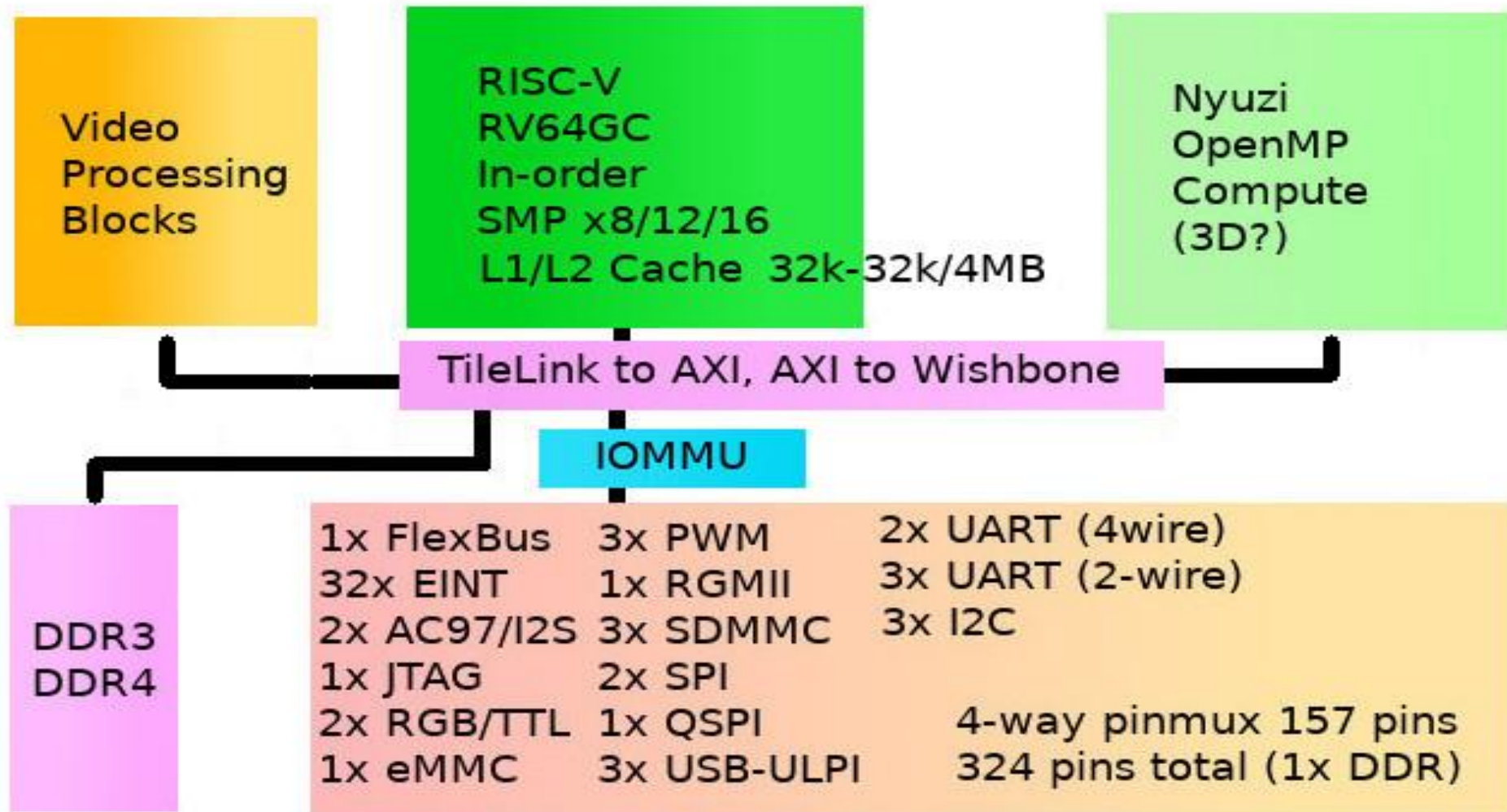
Test Program

Initialization

```
ori a7, a7, 0x2d7  
slli a7, a7, 11  
ori a7, a7, 0x1  
slli a7, a7, 11  
ori a7, a7, 0x3d2  
ori t3, t3, 0x164  
slli t3, t3, 11  
ori t3, t3, 0x52b  
slli t3, t3, 11  
ori t3, t3, 0x24e
```

Stimulus

```
and s4, a7, a7  
xor s8, s4, t3  
auipc t2, 0xafc37
```

Enabling Rust Flow and Framework for RISC-V Architectures

■ What is Rust?

- Rust is an open-source systems programming language that focuses on speed, memory safety and parallelism.

■ Famous Rust Projects

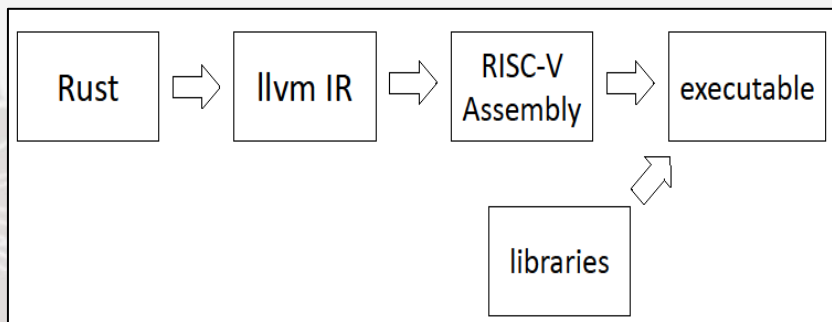
- **Servo**, the new browser engine being developed by Mozilla
- **Redox**, an operating system
- **Maidsafe**, a company that tries to create an encrypted, completely decentralized "successor" to the internet

■ Rust Support on RISC-V Platform

- Rust Support on Linux
- Rust on Bare Metal without standard library
- Rust on Bare Metal with standard library

Enabling Rust Flow and Framework for RISC-V Architectures

- Rust on Linux and Bare Metal



- Enable Thread on Bare Metal
 - To enable thread library in bare metal platform, we are designing our thread library to replace Linux pthreads library.

Ongoing – Vector Instructions

- To support Rust SIMD with RISC-V vector instructions.

```
fn main() {  
    let x1 = i32x4(1, 2, 3, 4);  
    let x2 = i32x4(2, 3, 4, 5);  
    unsafe {  
        let i32x4(a, b, c, d) = simd_add(x1, x2);  
        println!("{}", (a, b, c, d));  
    }  
}
```

Rust code

```
%8 = load <4 x i32>, <4 x i32>* %x1  
%9 = load <4 x i32>, <4 x i32>* %x2  
%10 = add <4 x i32> %8, %9
```

IR code

```
movabsq $8589934593, %rax # imm = 0x200000001  
movq %rax, (%rsp)  
movabsq $17179869187, %rax # imm = 0x400000003  
movq %rax, 8(%rsp)  
movabsq $12884901890, %rax # imm = 0x300000002  
movq %rax, 16(%rsp)  
movabsq $21474836484, %rax # imm = 0x500000004  
movq %rax, 24(%rsp)  
movdqa (%rsp), %xmm0  
padd 16(%rsp), %xmm0
```

X86

RISC-

```
sw ra, 124(sp)  
sw s1, 120(sp)  
sw s2, 116(sp)  
sw s3, 112(sp)  
addi a0, zero, 4  
sw a0, 24(sp)  
sw a0, 12(sp)  
addi a0, zero, 3  
sw a0, 32(sp)  
sw a0, 20(sp)  
sw a0, 8(sp)  
sw a0, 88(sp)  
addi a0, zero, 5  
sw a0, 36(sp)  
sw a0, 28(sp)  
sw a0, 92(sp)  
addi a0, zero, 7  
sw a0, 40(sp)  
sw a0, 96(sp)  
addi a0, zero, 9  
sw a0, 44(sp)  
sw a0, 100(sp)  
addi s1, zero, 2  
sw s1, 16(sp)  
sw s1, 4(sp)  
addi s2, zero, 1  
sw s2, 0(sp)
```

To-Do

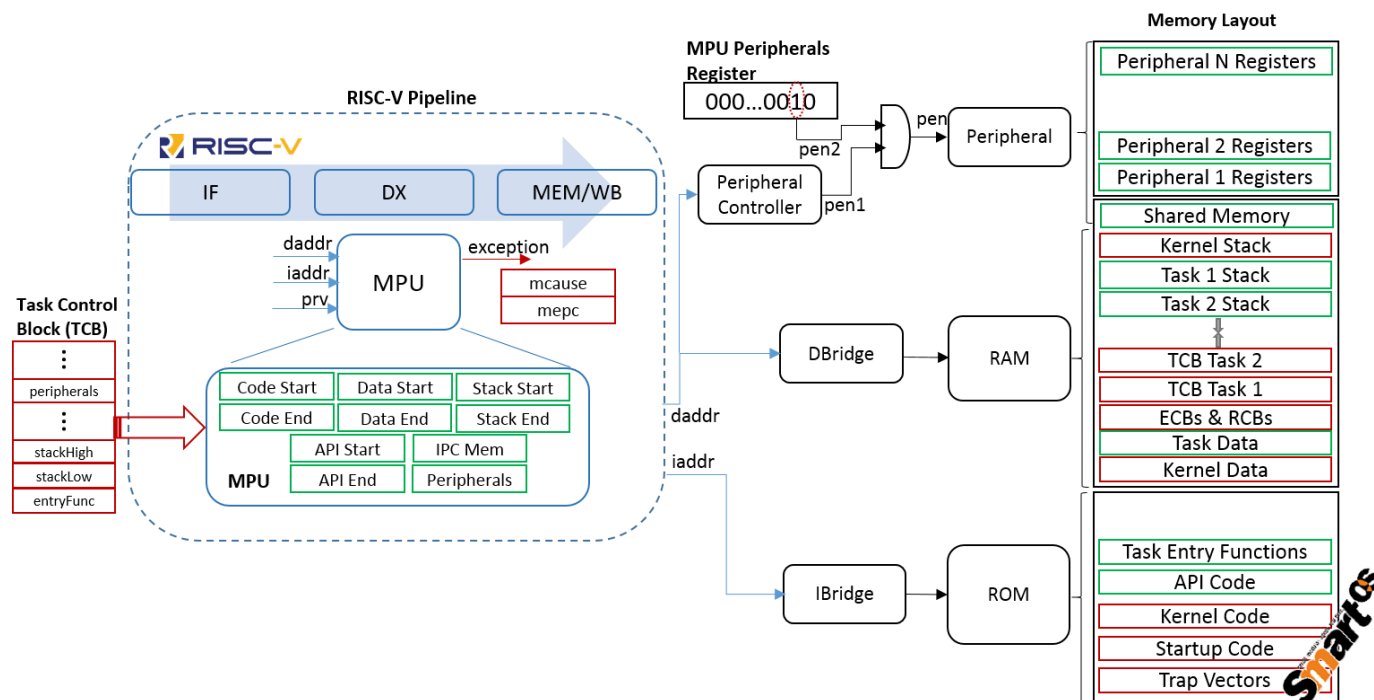
```
...  
vld v0, a1  
vld v1, a2  
vadd v0, v1  
...
```

RISC-V Vect
Instruction

Hardware-Software Co-designed Security Extensions

Maja Malenko, TU Graz, Austria

- Goal: memory isolation in small embedded devices
- SmartOS + vscale-based MCU
- MPU with **lightweight** extensions for protecting shared resources
 - Task isolation
 - Protecting **peripherals** (extended kernel resource manager)
 - Protecting **IPC**
- Configured by kernel/checked by hardware protection
 - **Low** hardware and memory footprint
 - **Insignificant** context switch overhead



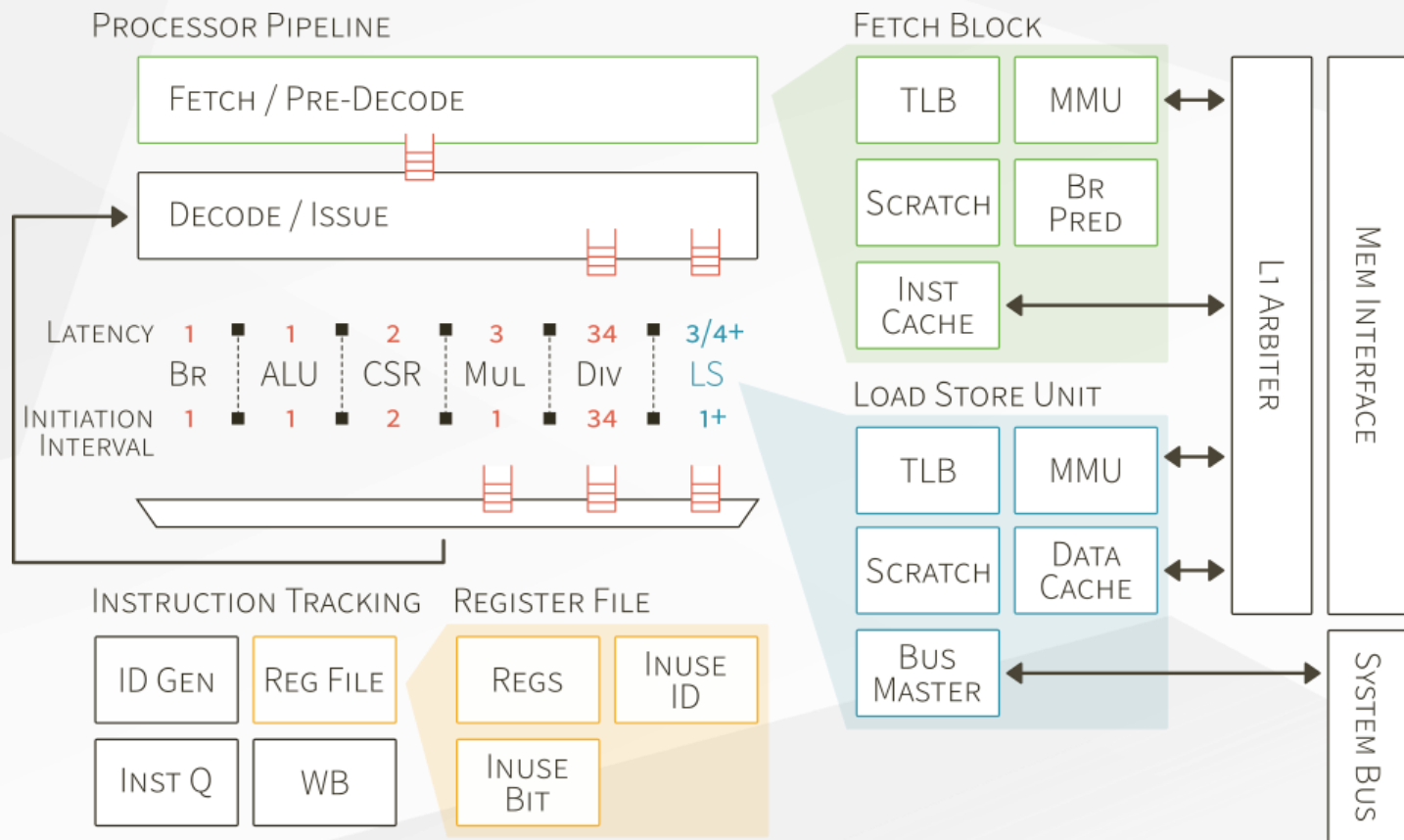
FPGA-BASED
HIGHLY CONFIGURABLE

TAIGA

RV32IMA
VENDOR AGNOSTIC

FOR HETEROGENEOUS SYSTEMS RESEARCH

Parallel-
execution units
provide first-class
support for
custom
instructions



PLATFORM COMPARISONS

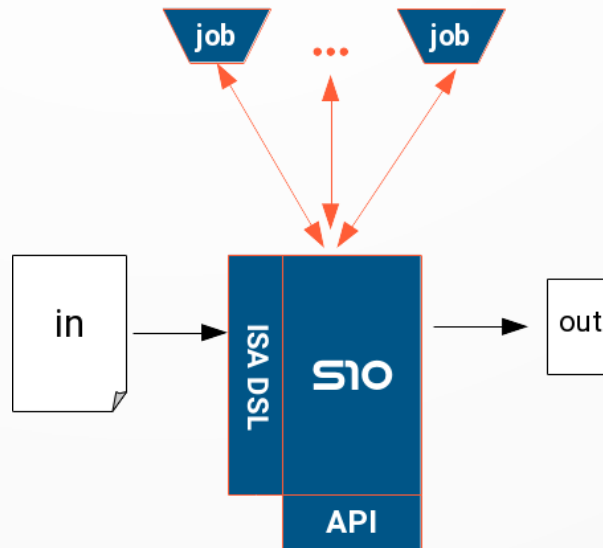
- 8KB 2-way Instruction/Data Caches
- Multiply/Divide Support

	Slices	LUTs	FFs	DSPs	BRAMs	Freq (MHz)
Rocket	5,536	17,144	9,058	0	10	54
LEON3	2,042	6,704	3,640	4	12	75
Taiga	1,371	3,998	2,942	4	10	104

AVAILABLE AT:
<https://gitlab.com/sfu-rcl/Taiga>

S10 RISC-V: revisiting the smallest program

- **Goal:** bring leading-edge superoptimization research to industry.
- S10 is a superoptimization framework which is:
- Retargetable
- Distributed
- Extensible



RISC-V is the first ISA targeted by S10, with ARM and x86 as wip!

- RV32I and RV64I support is complete, RV32E is coming up
- Extension M (Int. Mult./Div.) support complete
 - with C (Compressed Insns),
 - F (single Floats),
 - D (double Floats),
 - Q (quad Floats),
 - V (vectors) coming next;

For more information, come talk to us in the Poster session!

<https://linki.tools/s10>

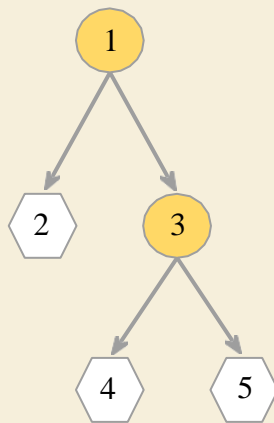
HW-Assisted Task Scheduling on Linux-enabled multicore Rocket Chip

Lucas Morais[†], Alfredo Goldman[†], Xavier Martorell[‡], Daniel Jiménez[‡], Carlos Alvarez[‡], Guido Araujo^{*}

[†]University of São Paulo, Brazil. [‡]Barcelona Supercomputing Center, Spain. ^{*}University of Campinas, Brazil.

Who needs Task Scheduling?

```
for (int i=1; j=1; i<N; i++) {  
    # (...) depend(in:v[i-1]) depend(out:v[i])  
    •fun1(&v[i-1], &v[i]);  
  
    for (int k=0; k<i; k++; j++) {  
        # (...) depend(in:v[i]) depend(out:u[i])  
        •fun2(&v[i], &u[j]);  
    }  
  
    fun3(3 * i);  
}
```



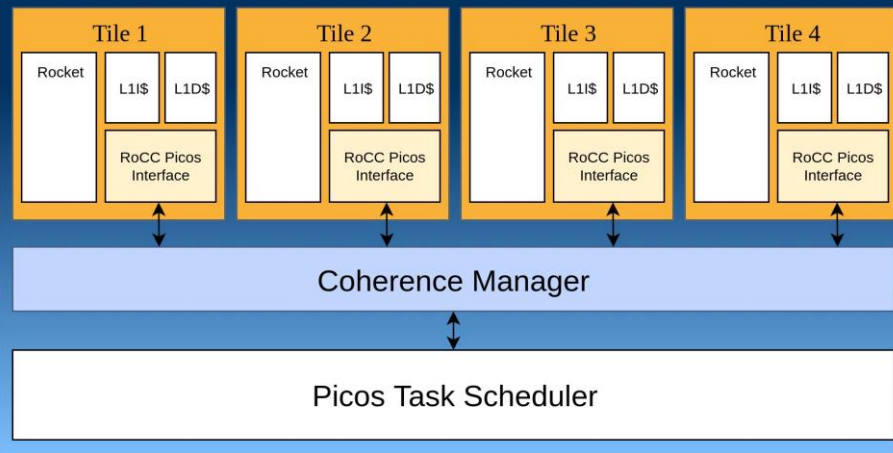
- Lets procedural programs be executed in dataflow manner: **semi-automatic parallelization**.
- Minimal code refactoring → productivity + clarity.
- Supported by OpenMP 4.0+, StarSs, etc.

Fundamental Problem

Dependence
Inference takes
non-negligible
amount of time.

- It has been successfully accelerated with FPGAs (see Picos).
- The problem remains for the most taxing workloads, though.
- That is due to the **high CPU-FPGA communication latencies**.

Our RISC-V-based Solution



- **Minimize overheads by bringing native support for Task Scheduling to the Processor.**
- Access to Picos provided by RoCC interface.
- Ongoing project - first prototype by end of semester.
- Now using ZC706 board, allowing for 6 RC cores. We'll port it to AXIOM board afterwards, allowing for up to 12 cores.
- Picos has been tested as a FPGA accelerator serving ARM cores before, so we have a proper performance baseline.

Ever tried booting SMP Linux on Rocket Chip?

- System boots SMP Linux (Kernel 4.15)
- Entirely open-source project
 - Deliverables to be released as Docker images **setting relevant git repos at the right version combination.**
 - Base release has tools for building SMP Linux, running simulations and generating compatible Rocket Chip bitstreams for Zynq boards (based on freedom-u-sdk and fpga-zynq forks).

<https://bit.ly/fpga-env>



Target implementation: FPGA
pursuing HW reconfigurability



The diagram illustrates a 3-core Klessydra SoC. A red dashed box highlights the top section, which contains three columns of text: 'PULPino feat.', 'PULPino feat.', and 'PULPino feat.'. Below this, a red bar represents the SoC, with three white circles indicating the cores. The first core is labeled 'Klessydra S0' and the second 'Klessydra T0'. A blue circle highlights the first core and its associated PULPino features. A blue line connects the first core to the text 'Klessydra S0'.

- “baseline” core
- RV32I user ISA
- MicroBinary1.10
- single hart

- family of cores
- RV32I user ISA
- RV64I mode V1.10
- Atomic ext (partial)
- multiple PC & CSR
- interleaved harts

- PULP
feat. **ARIANE**

Linux Compatible Multi-core

PULP



ETH zürich

Present design facts & figures

	Klessydra S0	Klessydra T01x	Klessydra T02x	Klessydra T03x
exec. mode	M	M	M	M
ISA	RV32I, priv 1.10	RV32I, priv 1.10	RV32I, priv 1.10	RV32I, priv 1.10
Atomic op.	no	AMOSWAP	AMOSWAP	AMOSWAP
pipe stages	2	2	3	4
Reg. file	Single 32x32b	Multiple 32x32b	Multiple 32x32b	Multiple 32x32b
harts	1	from 1 to x	from 2 to x	from 3 to x
irq sources	external	ext. + inter-hart	ext. + inter-hart	ext. + inter-hart
WFI	core	per-hart	per-hart	per-hart
Halt/wakeup	core	core	core	core
Throughput (Xilinx ser. 7)	up to 71 MIPS	up to 78 MIPS	up to 106 MIPS	up to 135 MIPS

- Passed all RISC-V RV32I tests and all Pulpino tests compatible with RV32I
- Basic debug hardware support
- Basic runtime system with software primitives for hart synchronization / mutex
- Equipped with dedicated test suite for hart synchronization / mutex

Observations and Ideas

- The first programming language learned shapes how one thinks
- Rocket Chip adoption rests upon Chisel adoption
- Collecting best practices for teaching Chisel will accelerate the adoption of Rocket Chip
- Rocket Chip uptake will also increase if the code style become easier to learn
- Main idea here is adding extra hand holding on the exact issues that people have experienced
- Excellent documentation in the Chisel wiki, `chisel-tutorial` and `generator-bootcamp`

Concrete Steps

- Add extra hand holding on the **exact issues** that people have experienced
- Create a **bridge** from where the current teaching materials leave off to the more advanced coding techniques, such as Cake Pattern and Diplomacy
- **Learning Journey** – `learningjourney.intensivate.com`
- Developed in collaboration with **Intensivate**
- First undergraduate lecture was held last week back in Banja Luka
- Chisel Bootcamp – May 10th-11th in Belgrade – 30 faculty members



Hardware Undo+Redo Logging

Matheus Ogleari

Ethan Miller

Jishen Zhao

<https://users.soe.ucsc.edu/~mogleari/>

CRSS Retreat 2018
May 16, 2018

Typical Memory and Storage Hierarchy:

Memory Fast access to working data



Storage Data persistence



Persistent M

Fast memory interface + persistence



Here! Persistent Memory is Coming!

Hardware – NonVolatile Random Access Memories (NVRAMs)

Battery-backed DRAM



NV-DIMM



3D XPoint



DDR3 Compatible MRAM



DRAM w/ Ultra-capacitor



Software – Persistent-memory-aware system software

Persistent Memory Support in OS

Using DAX in Windows

DAX Volume Creation

→ Format n: /dax /q

→ Format-Volume -DriveLetter n -

DAX Volume Identification

Is it a DAX volume?

→ call GetVolumeInformation("c:\", ...)

→ check lpFileSystemFlags for FILE_DAX_VOLUME (0x20000000)



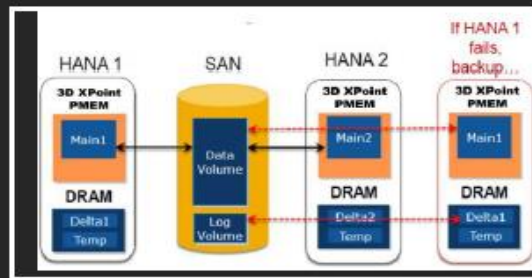
NVM.PM.FILE ACTIONS

- Implemented by kernel
- Implemented in userspace
- Optimized_flush
- Optimized_flush_and_verify



Persistent Memory File Systems

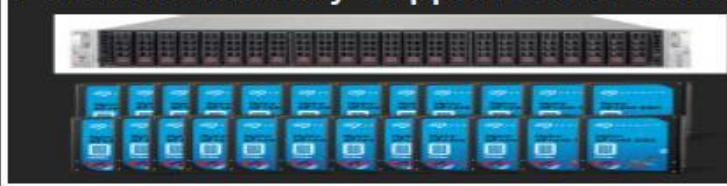
File system	Metadata atomicity	Data atomicity	Mmap Atomicity [1]
BPFS	Yes	Yes [2]	No
PMFS	Yes	No	No
Ext4-DAX	Yes	No	No
SCMFS	No	No	No
Aerie	Yes	No	No
NOVA	Yes	Yes	Yes



Persistent Memory Aware Database



Persistent Memory Support Over Fabric



Persistent Memory is ~~Coming!~~ Here!

...but unlocking its full potential isn't easy



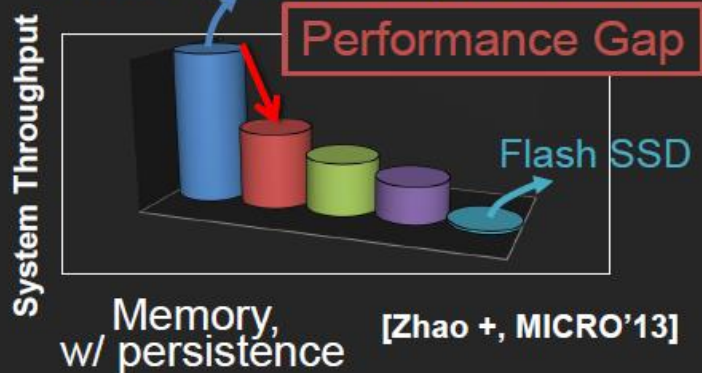
Logging, checkpointing,
copy-on-write, etc



- Persistence

- Traditionally property of **storage** systems
- Now must maintain in the **memory** system

Native system, no persistence



Opportunity Hardware Undo + Redo Logging

Contribution 1:

Relax write order control with
undo+redo logging



Contribution 2:

Leverage cache policies to
efficiently enable undo+redo
logging in hardware

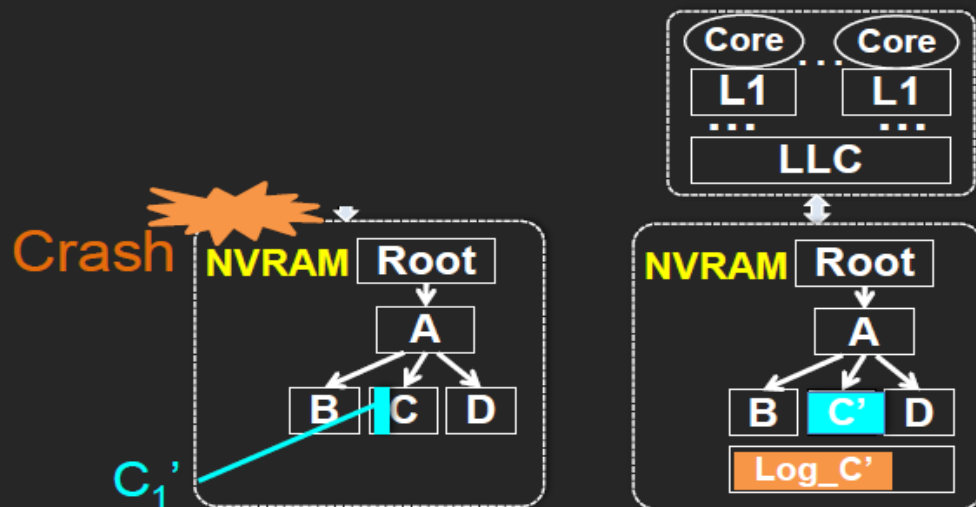


Persistence Requirement in Cache-Memory Hierarchy

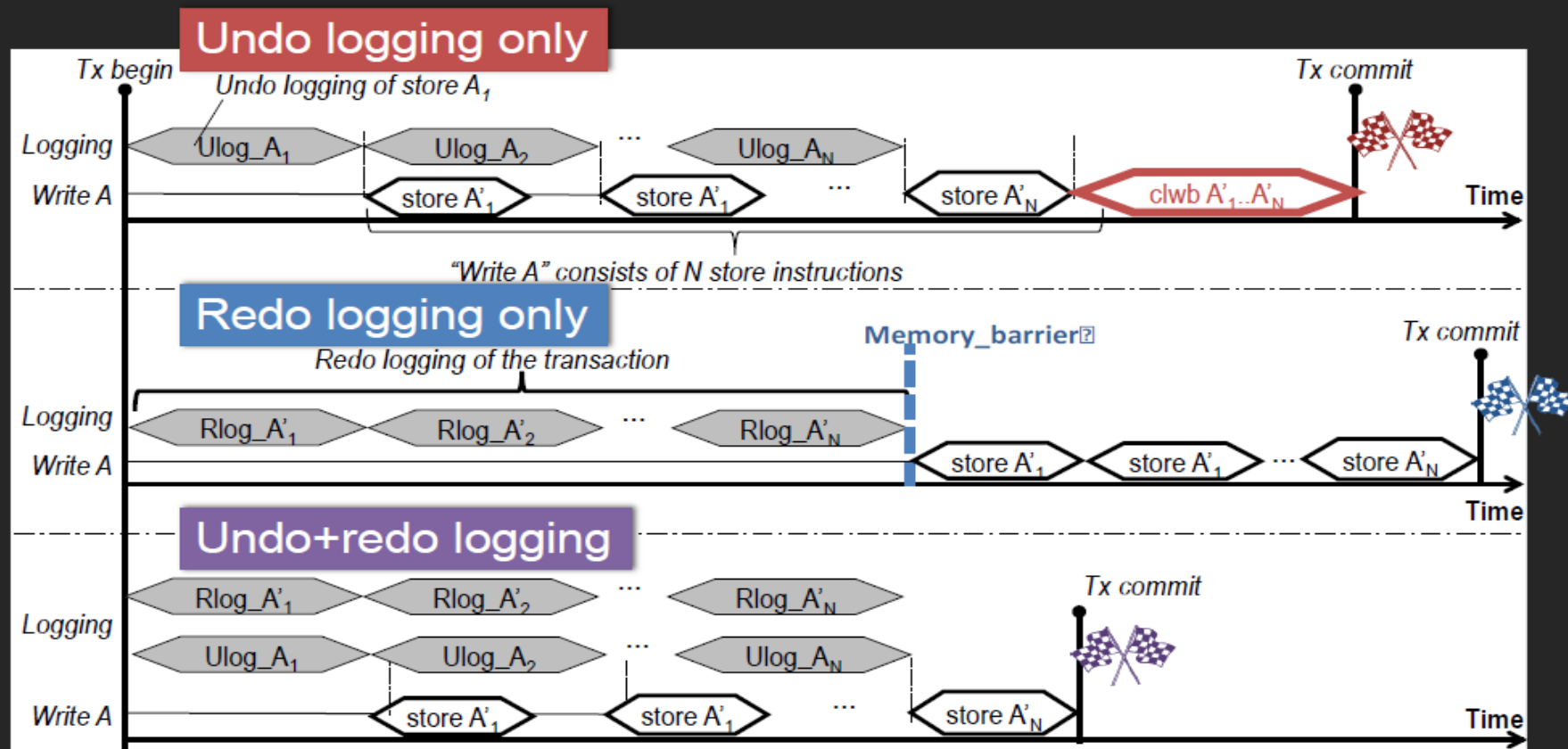
Update persistent memory
with a transaction

```
Tx_begin  
do some reads  
do some computation  
  
write C  
Tx_commit
```

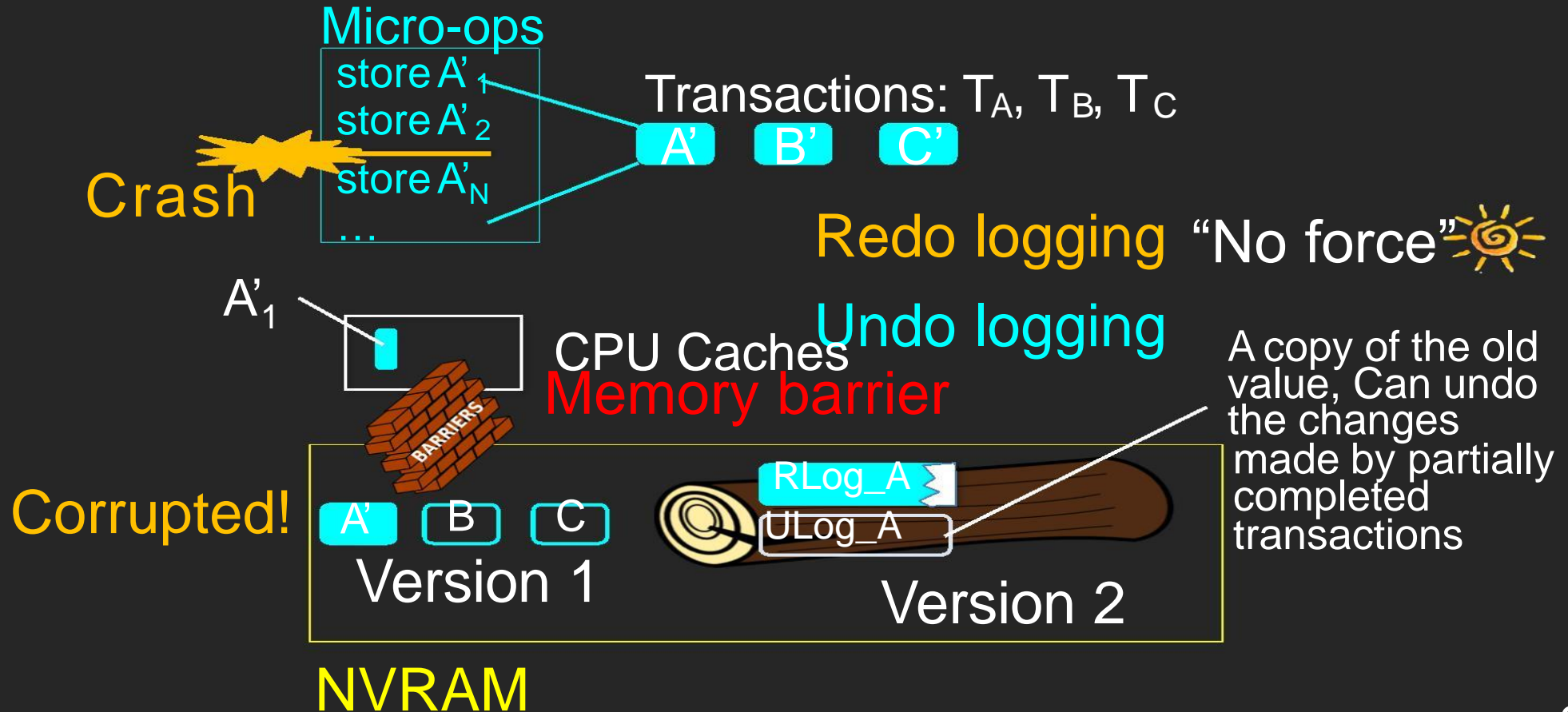
Micro-ops:
store C'_1
store C'_2
...



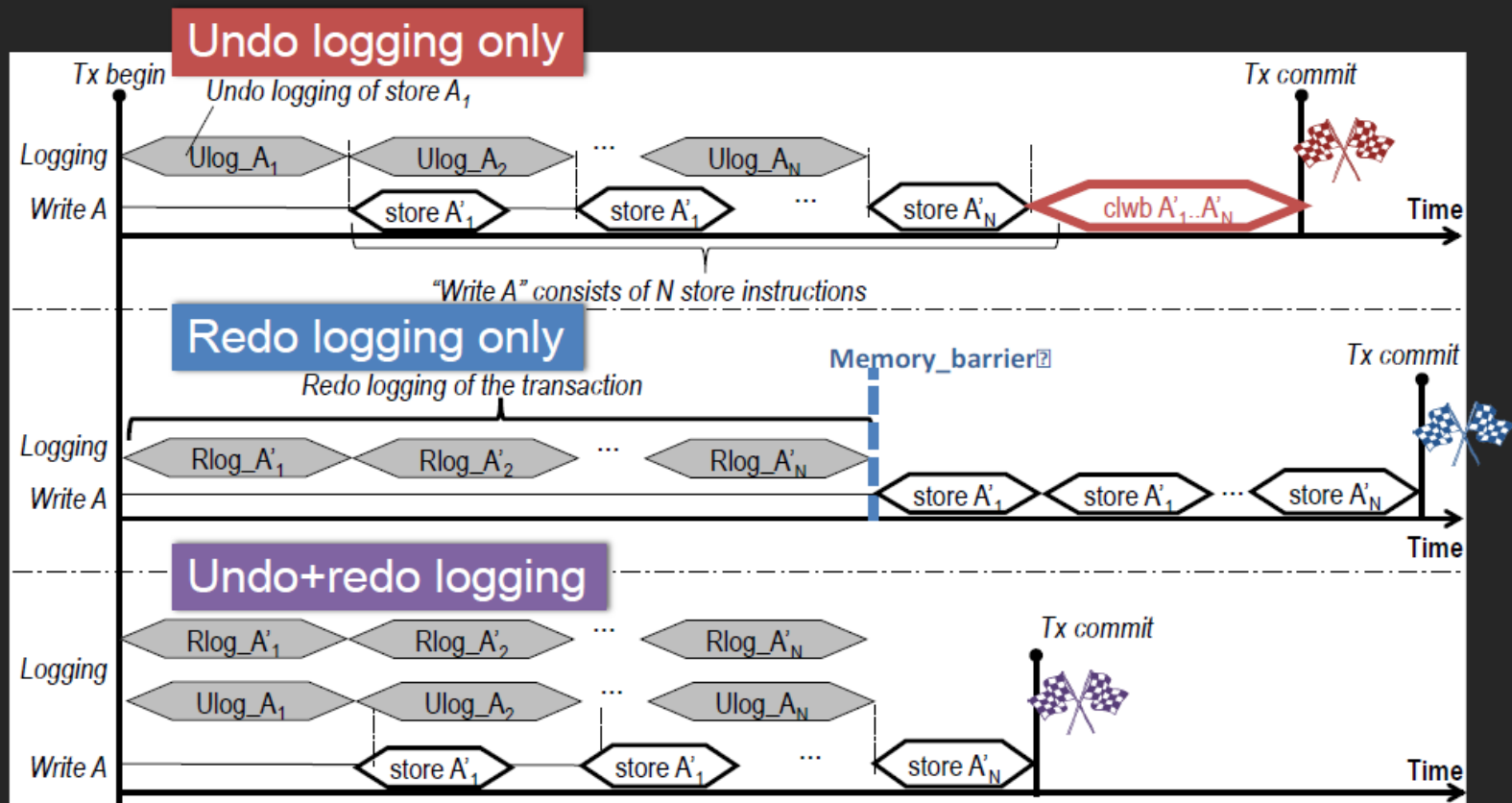
Preview of Undo+Redo Logging Benefits



Benefits of Undo + Redo Logging



Undo+Redo Logging Benefits



Inefficiency of Software Logging in Persistent Memory



Increased
memory traffic

Extra instructions
in the CPU
pipeline

Conservative
cache flushes

Risks with
multithreading?

Transaction T_A :

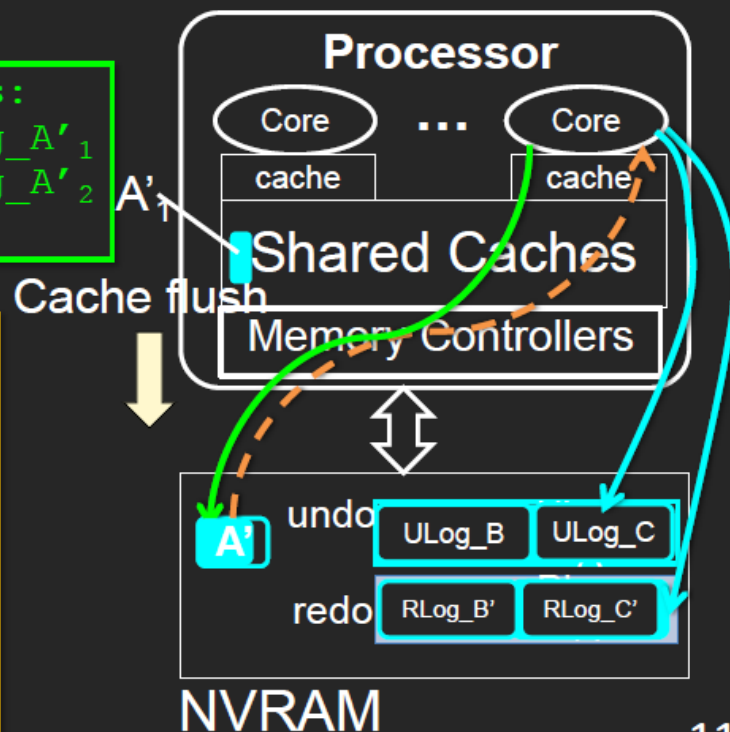
```
Tx_begin
do some reads
do some computation
Uncacheable_log(
  addr(A),
  new_val(A),
  old_val(A) )
write A
  clwb // can be
        // delayed?
Tx_commit
```

Micro-ops:

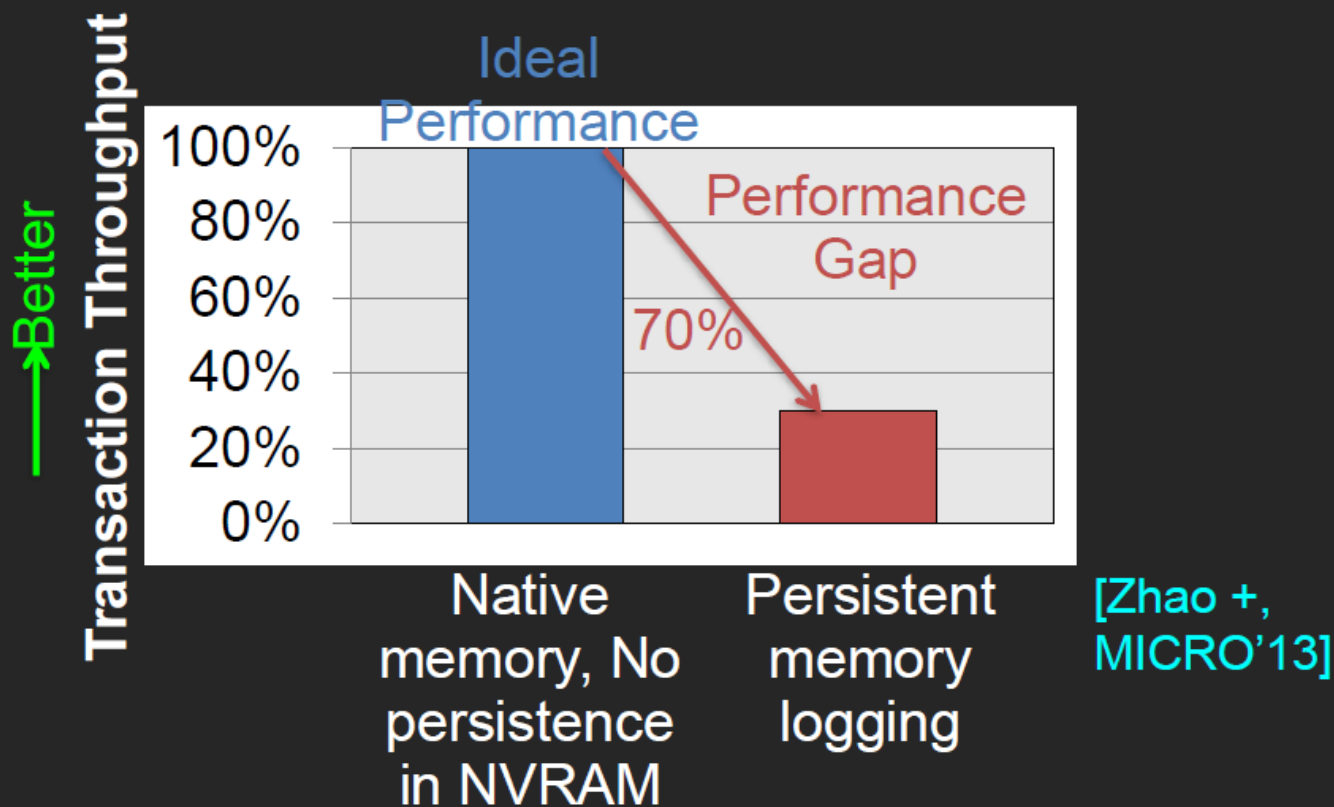
```
store log_A'_1
store log_A'_2
...
```

Micro-ops:

```
load A_1
load A_2
...
store
log_A_1
store
log_A_2
...
```

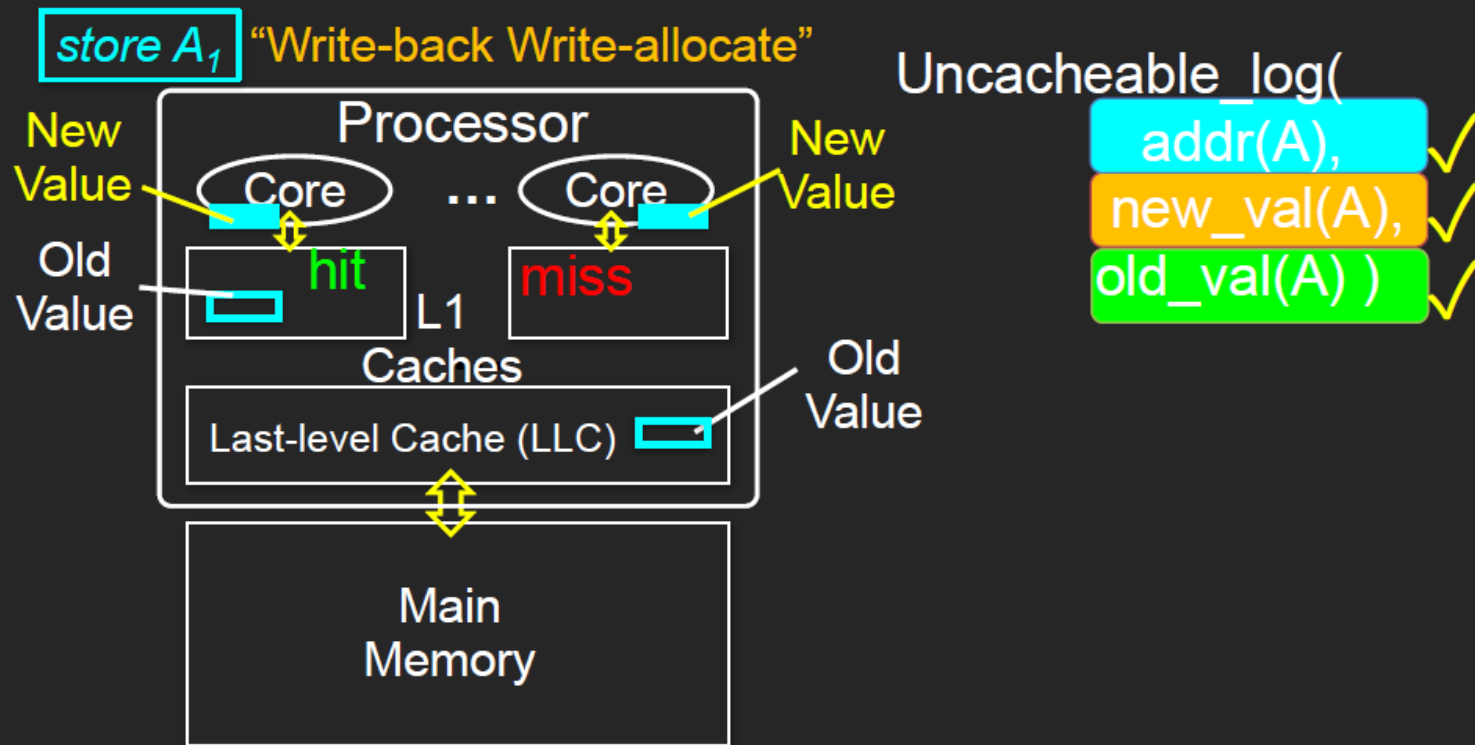


Performance Cost of Increased Memory Traffic



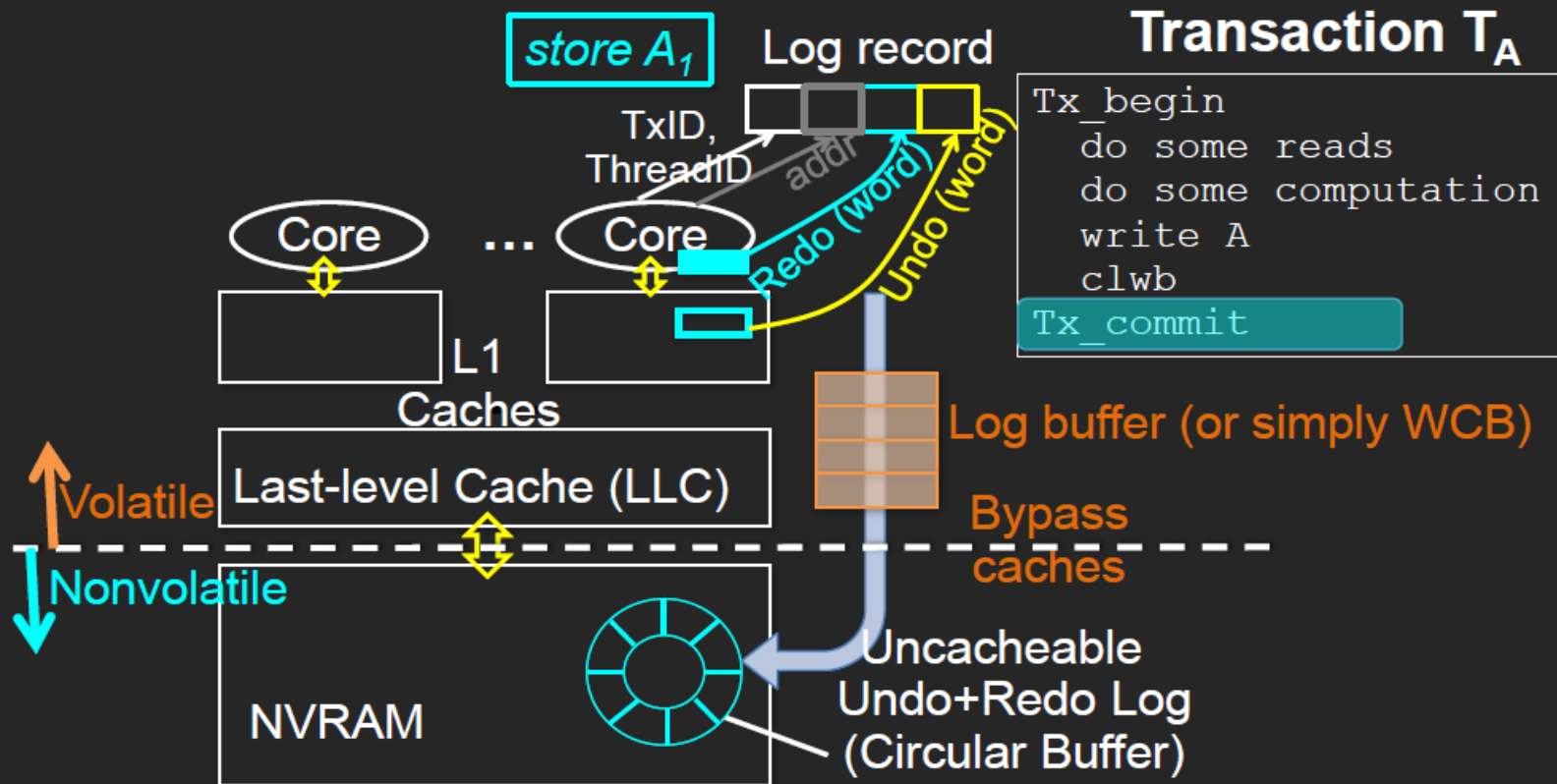
What Can We Leverage from Hardware?

Mechanism: Hardware Logging (HWL)



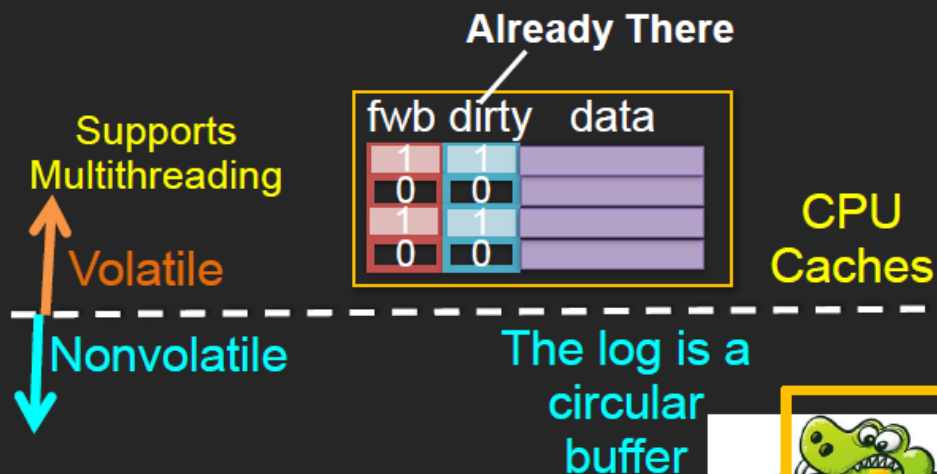
Undo+Redo Logging: Rides Along with CPU Caching

Naturally maintains the order between log and data



How About Cache Flushes?

Mechanism: Force Write Back (FWB)

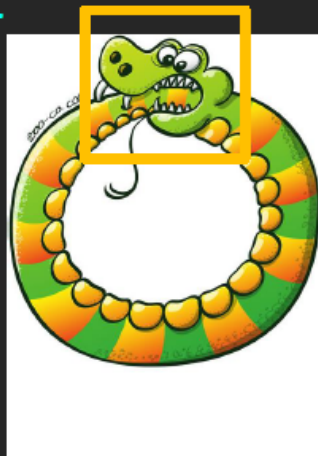


Transaction T_A

```
Tx_begin
do some reads
do some computation
write A
flush
Tx_commit
```

When is the best time for cache flushes?

Cache flushes frequency depends on log size and log update speed!



Commit the Transaction

Design principles

- **Hardware Logging (HWL)** – implements undo+redo in hardware
- **Force Write Back (FWB)** – decoupled from transaction execution

Transaction T_A

```
Tx_begin  
  do some reads  
  do some computation  
  write A  
  commit  
Tx_commit
```


Software and Hardware Cost

- Software support

Transaction interface

```
Tx_begin
  do some reads
  do some computation
  Write A
Tx_commit
```

Log_create()
Log_truncate()

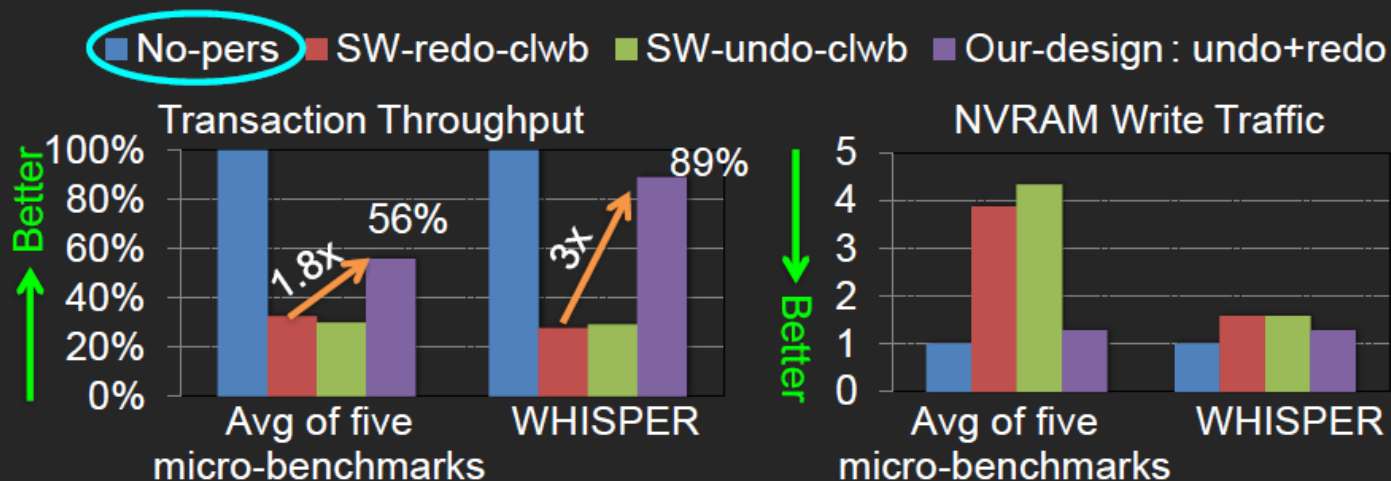


- Hardware overhead

Major Components	Logic Type	Size
Transaction ID register	Flip-flop	1 byte per HW thread
Log head and tail registers	Flip-flop	16 bytes
Fwb cache tag bit	SRAM	1 bit per cache line

Key Performance Results

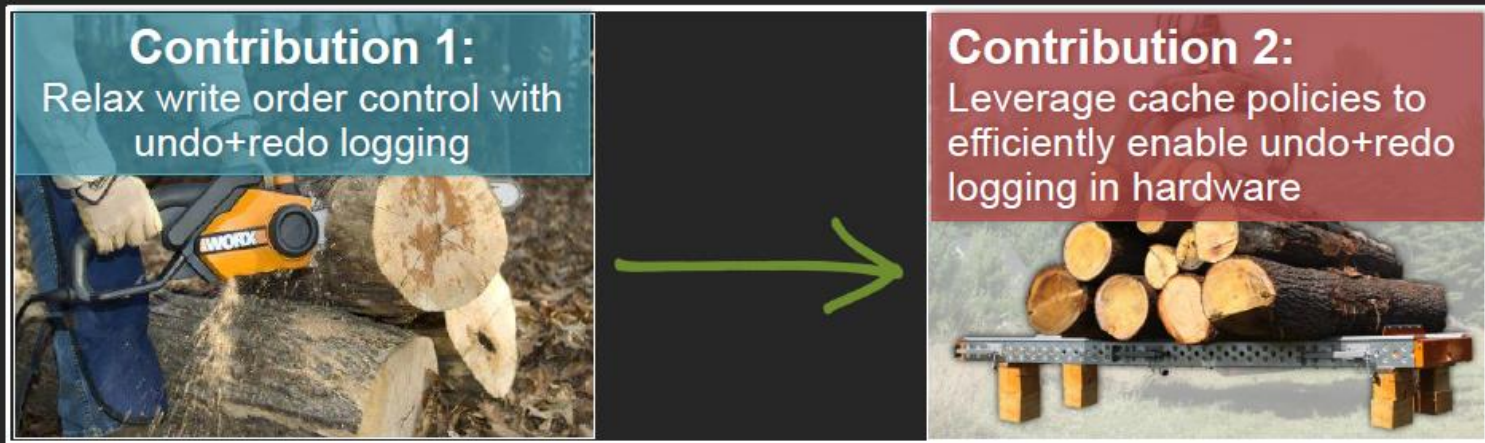
Ideal performance



Processor configuration: Core i7, 22nm, 4-core, 2.5GHz, 2 threads/core

Other results: energy consumption, instruction increase, IPC, sensitivity studies, etc.

Summary



- **Key points**
 - Rethink the way traditional software logging is done
 - Exploit opportunities in existing hardware – can naturally support data persistence

Ongoing Work and Collaboration

- Western Digital Internship



STING – A Complete RISC-V Functional Verification Solution

Presenter: Shubhodeep Roy Choudhury
Co-founder & CEO, Valtrix

Co-authors: Shajid Thiruvathodi

Introduction to STING

Software stack of test generators, checkers, device drivers, API library and micro kernel; Can be flexibly configured into a portable bare-metal program

Custom DSL (configuration file based mechanism) and programming frameworks allows development of constrained random, directed, use-case or graph- based tests

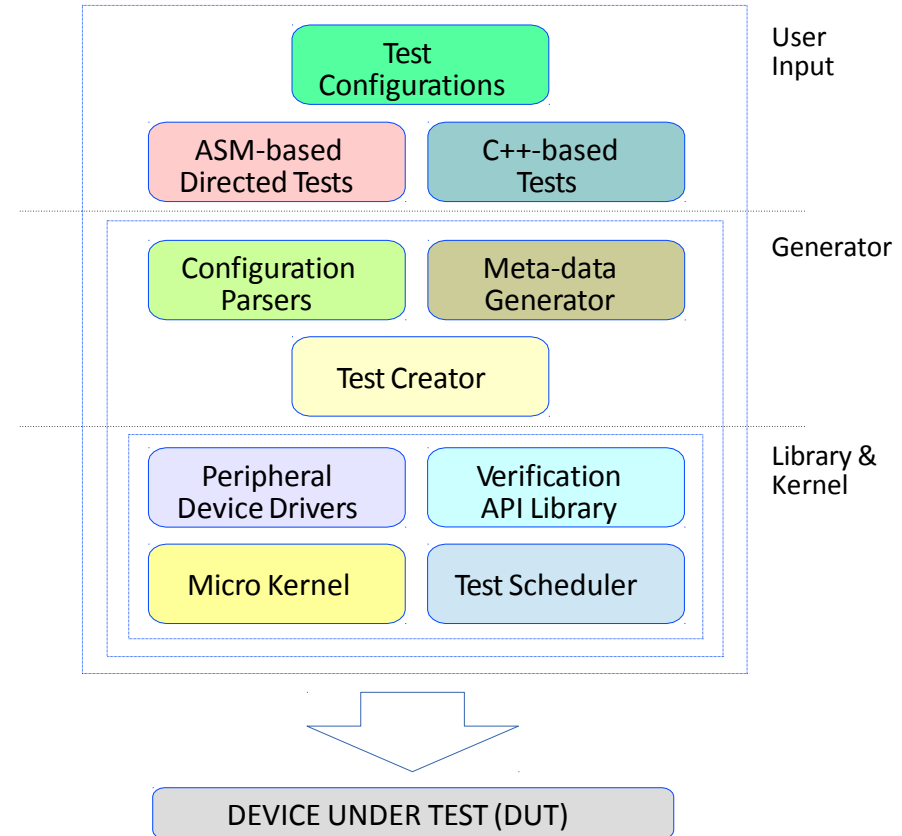
Lightweight and deterministic kernel with a very small instruction and memory footprint

Modes of execution to make the most efficient use of cycles in any verification environment

Supports all the IPs present in the SoC

Can be used out-of-box for supported IP/SoC implementations or ported with minimal efforts for new ones

Generalized architecture agnostic solution for software-driven functional verification methodology



Thank you for using www.freepdfconvert.com service!

Only two pages are converted. Please Sign Up to convert all pages.

<https://www.freepdfconvert.com/membership>

Dynamic Language Runtimes on RISC-V

I: Short-term view

- Dynamic languages are key
- OTI/IBM J9 open source
- Universal VM for Smalltalk, Java, Ruby, ...
- RISC-V port: Preliminary results

Boris Shingarov

LabWare

Dynamic Language Runtimes on RISC-V

II: Long-term view

- OMR runtime abstractions
- Target-agnostic backend synthesis from formal spec
- Formal verification of JIT

SEGGER – The Embedded Experts



SEGGER Microcontroller provides professional development and production solutions for the embedded market. All SEGGER products are highly optimized, "simply work" and benefit from more than 25 years of experience in the industry.

Founded: 1992
Employees: 50+
Founder: Rolf Segger
Headquarter: Hilden, Germany



Your one-stop shop from development to production

Proud **Sponsor** and Partner of RISC-V

SEGGER Microcontroller provides the most comprehensive and professional ecosystem for the RISC-V architecture.



Real-time Operating System

embOS is a priority-controlled real-time operating system, designed to be used as foundation for the development of embedded applications.



IDE - Embedded Studio

SEGGER's Embedded Studio supports RISC-V architecture and offers a comprehensive solution to develop and debug your application.



J-Link Debug Probe

The J-Link debug probes with their outstanding performance, robustness and ease of use are the market leading debug probes.



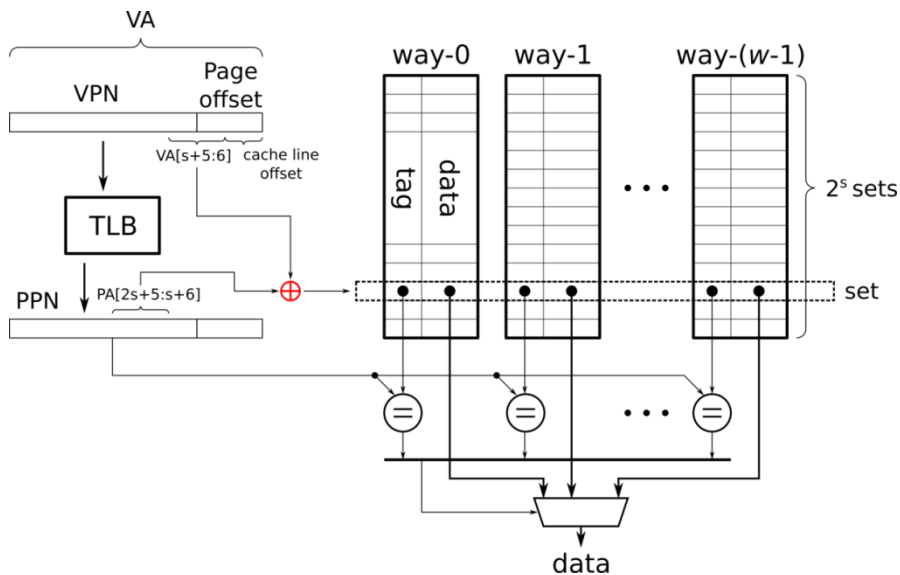
Defeating the Recent AnC Attack by Simply Hashing the Cache Indexes

— Implemented in a BOOM SoC

Wei Song, Rui Hou, Dan Meng

Institute of Information Engineering, Chinese Academy of Sciences

- Cache side-channel attacks relies on the deterministic mapping between virtual address to cache indexes.
- AnC is a smart cache side-channel attack that utilizes this mapping to break the ASLR protection in most browsers.
- Our defense is to remap the cache layout using part of the physical address.



Using PA as a secret key as it is normally unknown to user mode programs.

CI: cache index
VA: virtual address
PA: physical address
2^s: number of sets

$$CI = VA[s + 5: 6]$$

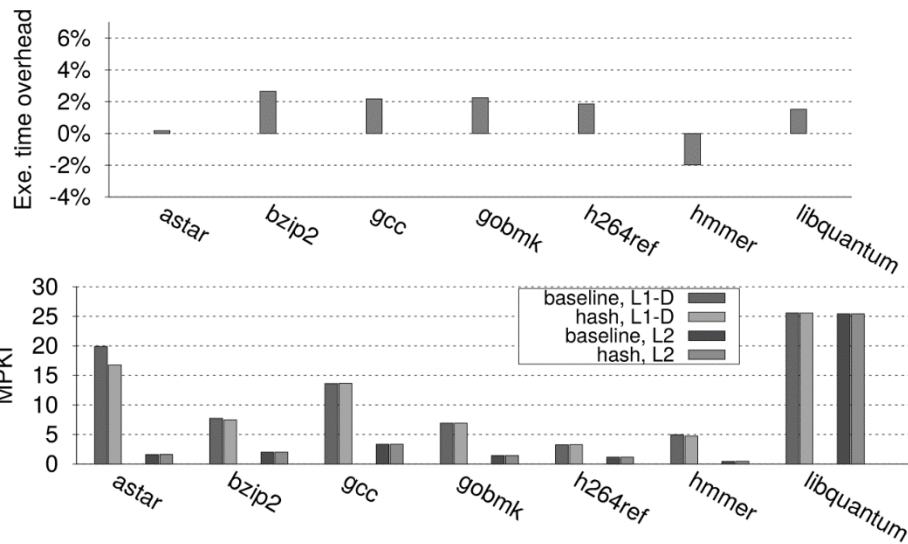
$$CI = VA[s + 5: 6] \oplus PA[2s + 5: s + 6]$$

IF:

An OS constantly allocates random physical pages to consecutive virtual pages and disable the huge page support.

THEN:

The proposed change can stop most PRIME+PROBE, EVICT+TIME and AnC attacks with marginal performance overhead.



Cybersecurity software increases vulnerability and ruins performance



- Processors blindly run vulnerable software
- Cybersecurity companies respond with more software
- But adding more layers of software—even security software—just adds more bugs
- Plus each layer of software substantially degrades system performance

CoreGuard: Silicon IP that integrates with RISC-V processors



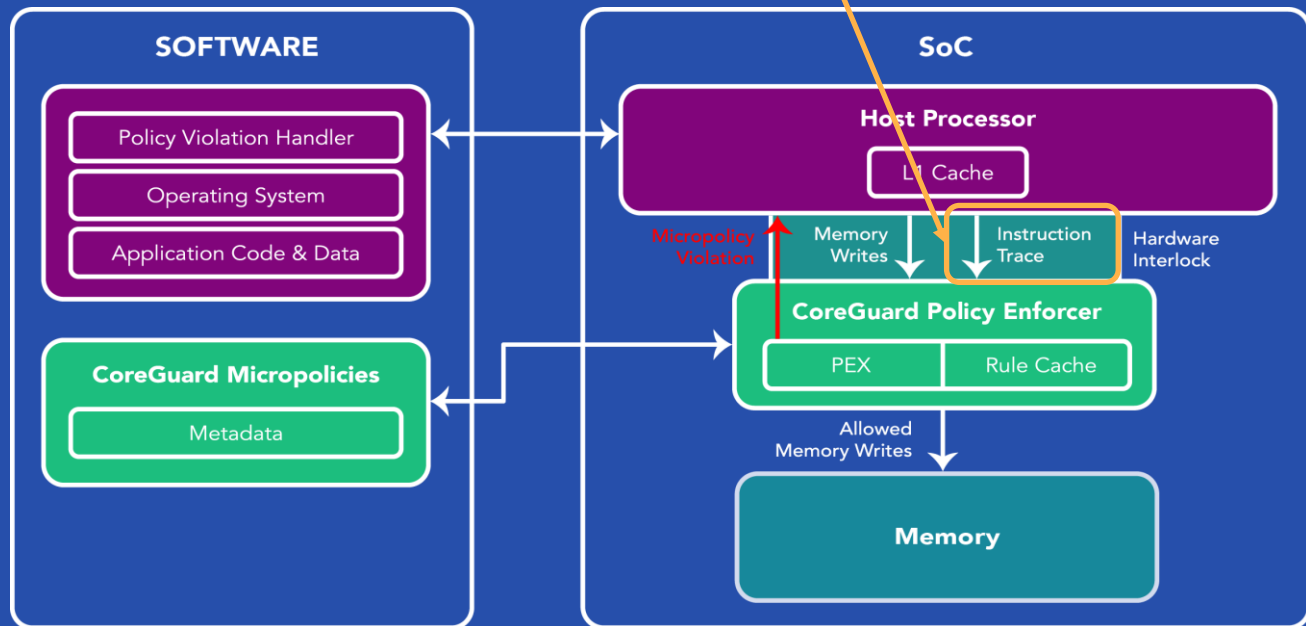
- CoreGuard empowers modern RISC processors to defend themselves in real time from network-based attacks
- Blocks entire classes of attacks from MITRE's Common Weakness Enumeration (CWE) of 705 vulnerabilities
- With optimized rule cache: little to no impact on performance. Power and area impact are design dependent

CoreGuard: Block diagram

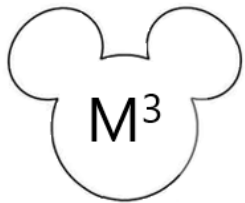
Architecture agnostic IP licensed and delivered as hardware design files

- Extract a set of trace signals from host RISC-V processor
- Provide mechanism for CoreGuard to affect a stall on the host
- Connect host RISC-V processor's data bus to CoreGuard instead of the memory bus fabric
- Instantiate CoreGuard on SoC and connect host RISC-V processor and bus fabric
- Increase system memory as needed to account for metadata's needs
- Handle exceptions thrown from CoreGuard when policy violated

RISC-V Trace Interface proposed by UltraSoC, SiFive

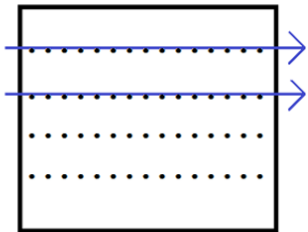


See our poster to start integrating security on your SoC!



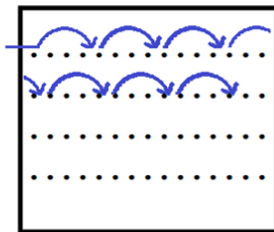
AXIS == Pre-declared access pattern

Index == Record-by-record address



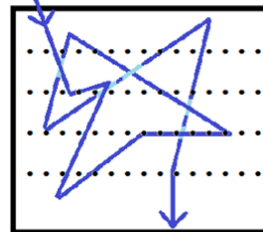
Linear (row)

- HPC algorithms > 90% efficient
- Exploits ROW buffer



Stepper (col)

- Striding access
e.g. column access,
select small field
- Cannot Exploit ROW buffer
- Mathematically predictable



Indirect

- 'Random' access
- Defined by an index
- Scatter-Gather operation
- HPCG algo < 10% efficiency
- Program code access



Key Concepts:

Forward Caching
Index Tables
Fast Forward
Caching

Key Capabilities:

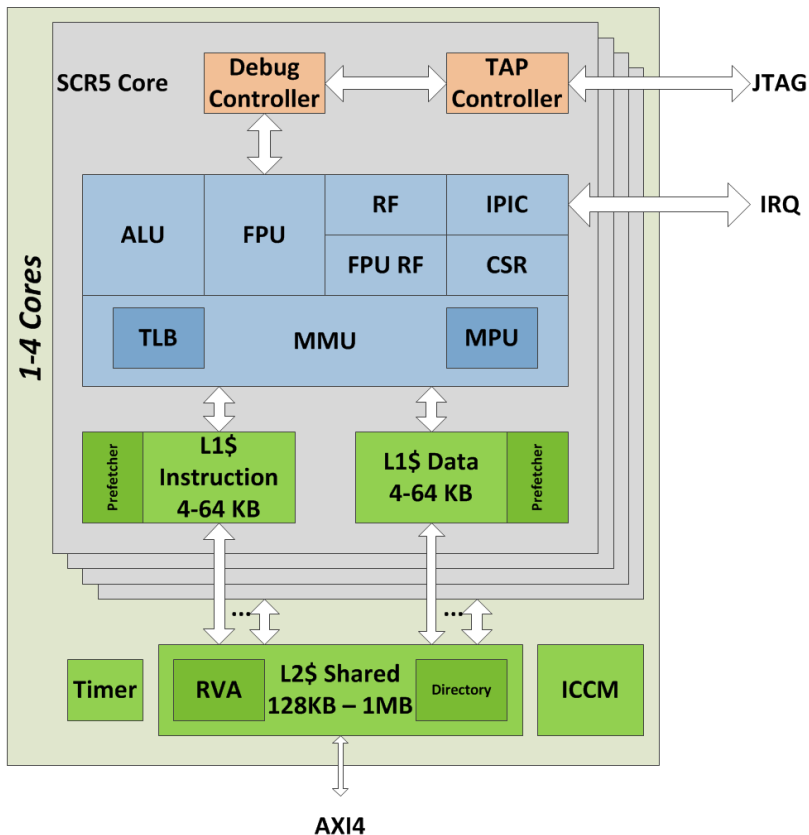
Sorting
Name/Value
Lookup
Scatter/Gather

Key Advantages:

Lower Latency
Efficient Bandwidth
Usage
Better Security

Micron Technology: M³

SCR5: efficient RISC-V core with Linux and SMP support



- RV32IMC[AFD]
- 1-4 Cores SMP
- 7-9 stages in-order pipeline
- Full MMU, virtual memory
- U/S/M modes
- Static BP, BTB, BHT RAS
- 4-64KB L1\$ with hardware prefetcher
- 128KB-1MB shared L2\$, fully coherent + Directory
- All RAMs with ECC (SEC/DED)
- Memory Protection Unit
- 32/64/128 bit AXI4

1GHz+ @28nm

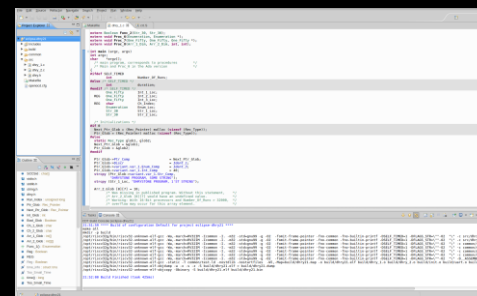
From 250 kGates (basic single-core config w/o caches)

Licensed and taped-out at the customer

Performance*, per MHz	DMIPS	-02	1.60
		-best**	2.48
		Coremark -best**	2.83

* Dhrystone 2.1, Coremark 1.0, GCC 7.1.0 BM from TCM

** O3-funroll-loops -fpeel-loops -fgcse-sm -fgcse-las -flto





CoT, RoT, RISC-V & High Volume Application

*Danny Ybarra – CTO organization Storage Device Security Architect
April 25, 2018*

Security: CoT, RoT, and RISC-V (More Than a Single Feature)

Topics

- System Security Challenge
- System Chain of Trust (CoT) and Root of Trust (RoT)
- WD Security Deployment: a High Volume Production application
- WD RoT basics
- RoT And RISC-V Opportunities

Security: CoT, RoT, and RISC-V (More Than a Single Feature)

System Security Challenge

- Today's Security Solution are created with independent security components
Each component defines its contribution to the security solution and is measured against different Security Evaluation standards or best-practices
User Data passes between the components with varying degrees of data protection

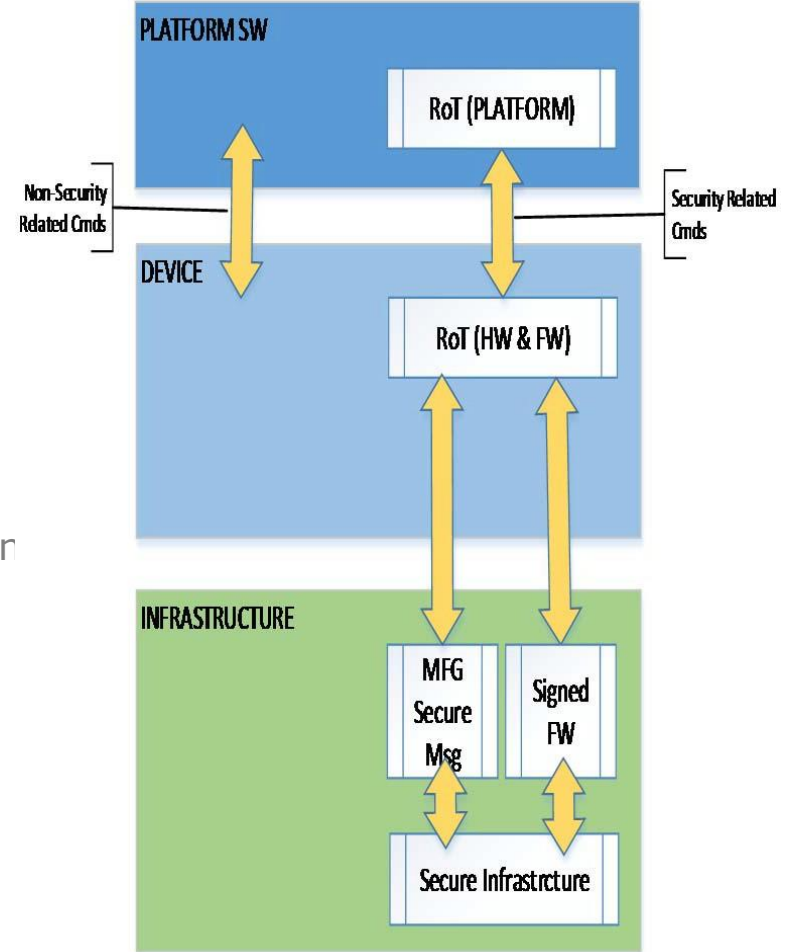


- NEXT STEPS: System Security should bind the system's security components
 - System Guidelines are being established to define relationships between components. (e.g. SP800-193: CoT & RoTs that provide Protection, Detection, & Recovery)

Security: CoT, RoT, and RISC-V (More Than a Single Feature)

System's Security Integration

- Device's RoT also serves as a Chain of Trust (CoT) keystone
- Platform & Device CoT Functions need:
 - Device Attestation
 - Device Roles Authentication
 - Device Feature Authorization
 - General Security Protocol support
- RoT (w/RISC-V) considerations
 - Balancing Security vs. Performance vs. Cost
 - Agility to adapt to multiple internal and external application

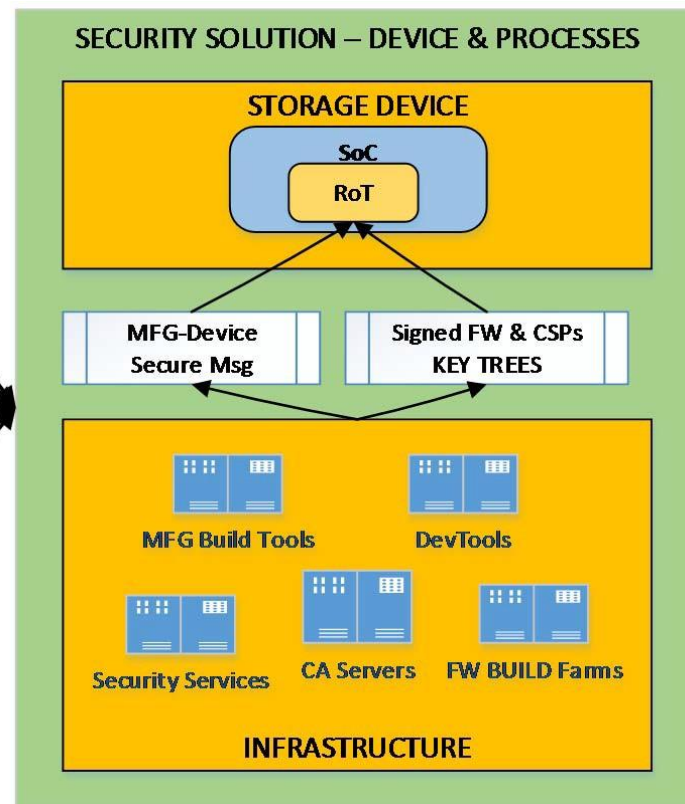
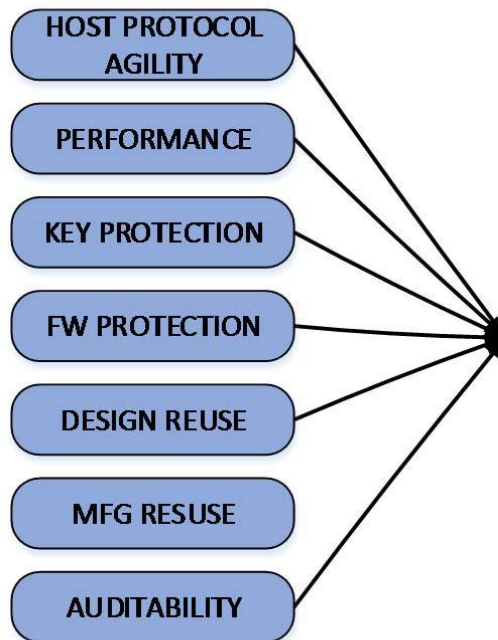


Security: CoT, RoT, and RISC-V (More Than a Single Feature)

WD Storage Security Deployment: High Volume Application

- Security Requirements must:
 - Satisfy customer and WD needs
 - Protect against remote and physical attacks
 - Protect against external and internal attacks
- Device and Processes RoTs protect/detect/manage:
 - Secure Objects (FW & CSPs)
 - Secure Messages/Protocols
 - Shared and Device Unique RSA Key Injection
- Device RoT designs balance:
 - Internal & External Protection
 - Performance
 - Cost
 - Apps Agility

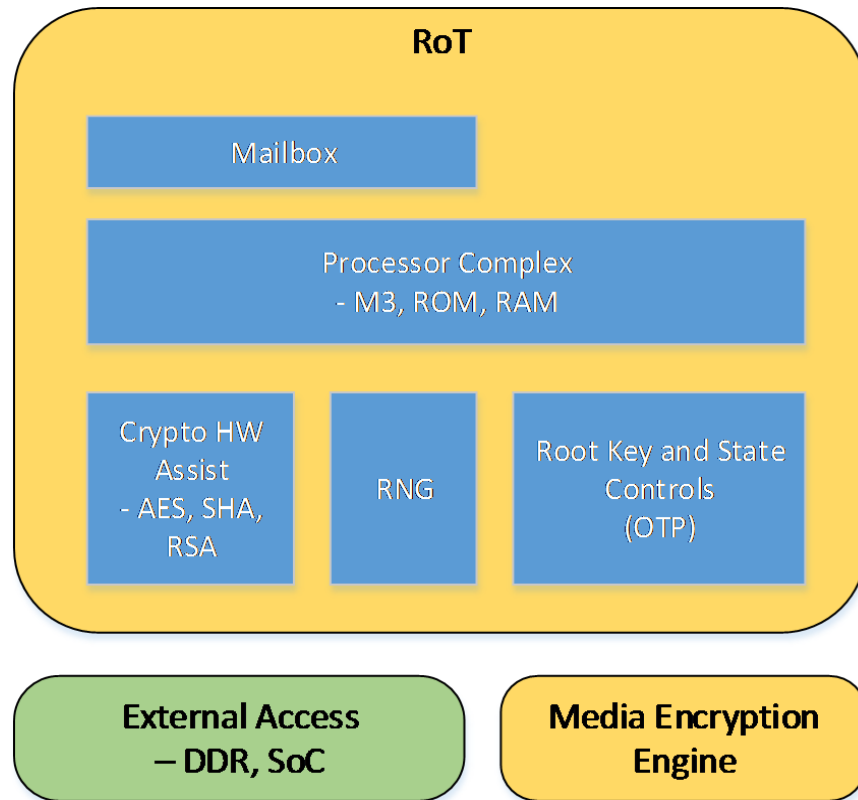
REQUIREMENTS



Security: CoT, RoT, and RISC-V (More Than a Single Feature)

WD Storage Device RoT Basics

- RoT Basics
 - Physical Isolation
 - FW, CSPs, & Message Authentication entering RoT
 - Cryptographically Binds FW/CSPs to Device
 - Provide ClearText Key/CSP isolation
 - Provide “Atomic” function protection
- RoT HW & FW Operation provide:
 - Device Access Control management
 - Security Services
 - Immutable Key Management
- Media Encryption Key (MEK) Management RoT Example
 - Has sole Media Encryptor Key cache access
 - Generates and protects Cleartext MEKs
 - Binds security protocol operations to Cleartext MEK usage
 - Binds & protects MEKs/Objects to device unique symmetric Keys

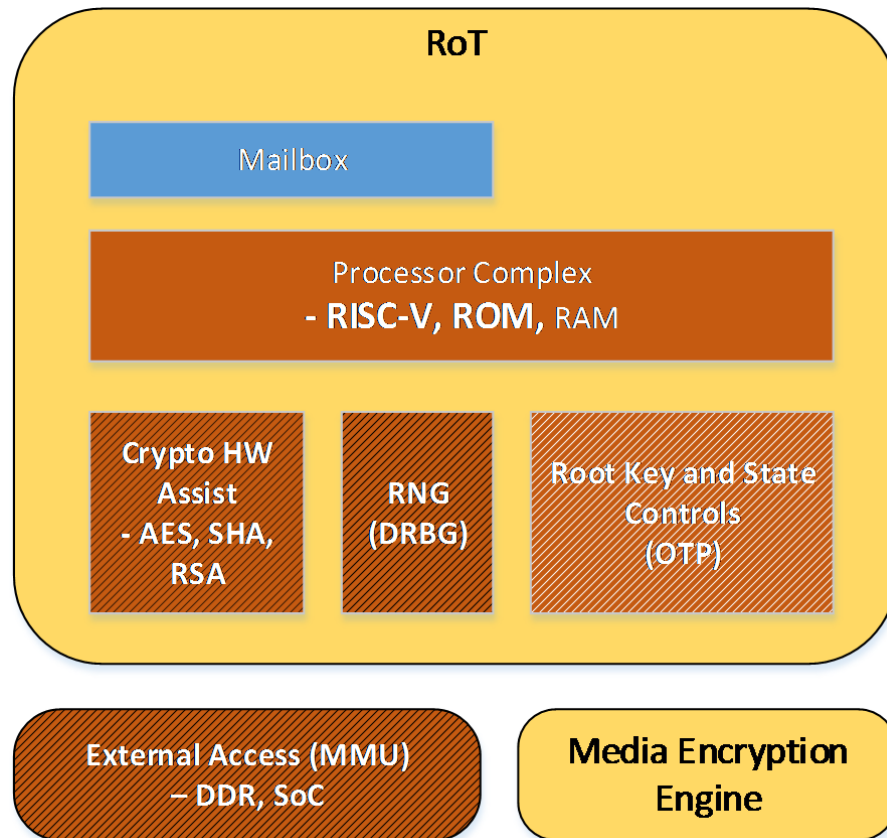


Security: CoT, RoT, and RISC-V (More Than a Single Feature)

SoC RoT And RISC-V Customization Opportunities

- Each RoT Deployment Must balance:
 - Protection, Performance, Cost, TimeToMarket
- Cryptographic Performance Range
 - Full Algorithm Assist
 - Partial Algorithm Assist
 - **FW Instruction Assist**
 - FW Only
 - **ROM or FW cryptographic functions**
 - **Random Number Generation**
- **RoT Virtual or Physical isolation & Memory Mgmt**
 - Privilege vs. User Mode Isolation
 - Resource Sharing Concerns (MMU: SRAM, DDR, Ports, SoC Modules)
- Other Opportunities
 - **Other Side Channel Attack types (simple power, diff power)**
 - **OTP Management**
- Security Evaluation standardization
- CoT Deployment

Note: RISC-V opportunity in RED



Abstract, flowing lines in shades of orange, red, and blue, resembling a stylized flame or smoke, set against a black background.

Western Digital®