MICROCHIP

Survey of Notable Security-Enhancing Activities in the RISC-V Universe

G. Richard Newell

Cryptarchi '19 (June 25)

Have you heard of RISC-V?

- **A free and open ISA developed at UC Berkeley**
  - Via a permissive BSD license

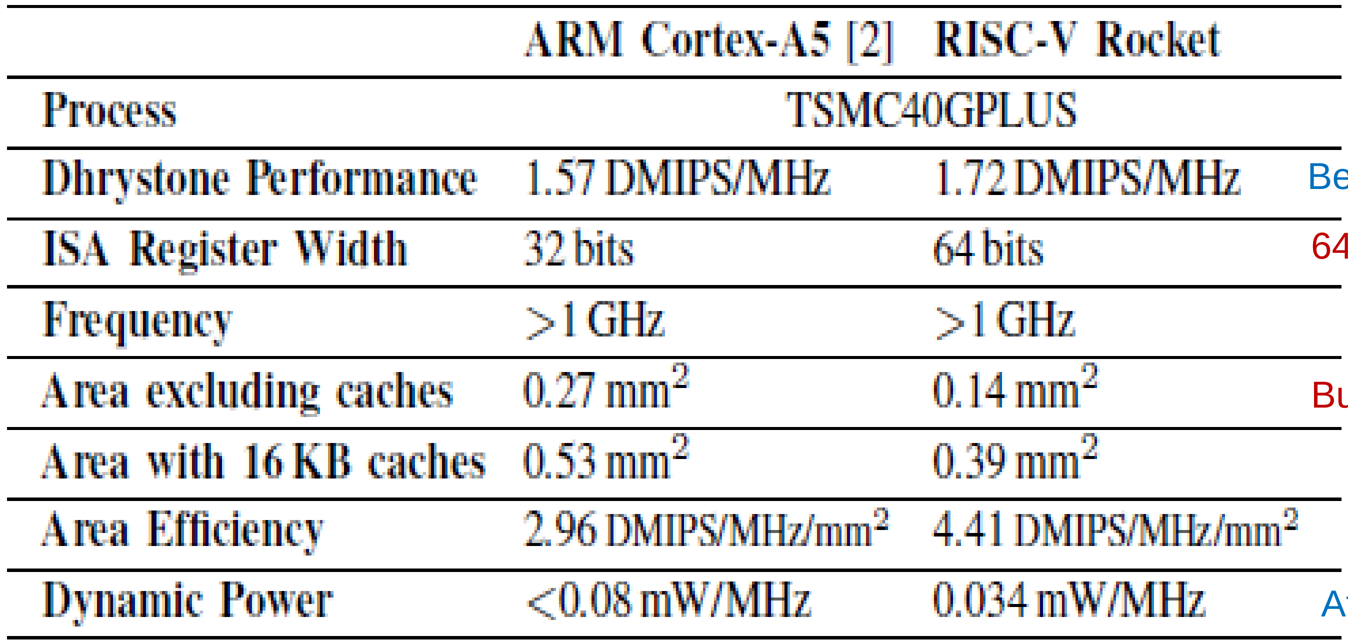
- **ISA designed for**
  - Simplicity - < 50 instructions required to run Linux
  - Longevity – standardized instructions are fixed, your code runs forever

- **RISC-V foundation setup to**
  - Protect the ISA
  - Foster adoption

- **RISC-V is not an open source processor**
  - Although open source implementations will exist
  - Provides everyone an “architectural” license to innovate

RISC-V Area and Power Advantage

- **RISC-V ISA based microarchitecture compared to a roughly comparable-in-performance ARM CPU implemented in the same silicon process:**



	ARM Cortex-A5 [2]	RISC-V Rocket
Process	TSMC40GPLUS	
Dhrystone Performance	1.57 DMIPS/MHz	1.72 DMIPS/MHz
ISA Register Width	32 bits	64 bits
Frequency	>1 GHz	>1 GHz
Area excluding caches	0.27 mm^2	0.14 mm^2
Area with 16 KB caches	0.53 mm^2	0.39 mm^2
Area Efficiency	2.96 DMIPS/MHz/mm^2	4.41 DMIPS/MHz/mm^2
Dynamic Power	<0.08 mW/MHz	0.034 mW/MHz

Better performance

64 bit machine

But only ½ the area

At ½ the power

Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, V. Stojanovic and K. Asanovic, "A45nm 1.3GHz 16.7 Double-Precision GFLOPS/W RISC-V Processor with Vector Accelerators," in *European Solid State Circuits Conference (ESSCIRC)*, 2014

UCB “Rocket” single-issue, in-order, <u>5-stage pipeline</u>, single- and double-precision floating point, 64-bit RV64G ISA microarchitecture
ARM “Cortex-A5”, single-issue, in-order, single- and double-precision floating point, <u>8-stage pipeline</u>, 32-bit ARMv7 ISA microarchitecture

RISC-V IP Providers

RISC-V

Berkeley Architecture Research
NXP
antmicro
SANECHIPS 中兴微电子
Western Digital
NVIDIA
bluespec
ORION 猎户星空
SAMSUNG
C-SKY 中天微
Google
QUALCOMM
Microsemi a Microchip company
cortus
Micron
Rambus
SiFive
THALES
MARVELL

ESPRESSIF
MEDIATEK
HIGO 海高
QuickLogic
ANDES TECHNOLOGY
galois
imperas
REM
SECURE RF
BAE SYSTEMS
Esperanto TECHNOLOGIES
NUS
inside secure
Hortonworks
सी डैक CDAC
ETH zürich
oculus
Symbiotic EDA
谨振电子科技
Hewlett Packard Enterprise
VeriSilicon
SK hynix
OpenSingularity
INTRINSIC ID
MIT CSAIL
DOLPHIN
TRINAMIC MOTION CONTROL
GOWIN
BSC Barcelona Supercomputing Center
SIEMENS
Minima
ASHLING
EMBECOSM
INTRINSIX
HITACHI Inspire the Next
NUCLEI
imt.
NSI-TEXE
SILICON LABS
优矽科技 UC TECH IP
UBILITE
TESLA
CloudBEAR
FORTH Institute of Computer Science
TANGRAM
PANGO
NORTHROP GRUMMAN
Mentor
surecore
CEVA
君正 Ingenic
DOVER MICROSYSTEMS
SILEX INSIGHT
MINRES
CRYPTAPE
Semiconductor Investment 厦门半导体
PerfXLab 澎峰科技
PEZY Computing
IBM
SEGGER
SoundAI
LAUTERBACH DEVELOPMENT TOOLS
NETRONOME
DATA 61 CSIRO
GRAY RESEARCH
cadence
GREENWAVES TECHNOLOGIES
XtremeEDA
Alibaba Group 阿里巴巴集团
MOSIS
ultraSoC
CMC MICROSYSTEMS
BITMAIN
EMDALO TECHNOLOGIES
Centipede
SH CONSULTING
Syntacore
lowRISC
VectorBlox
Canaan
Technolution
Rumble Development
Tortuga Logic
BERKELEY LAB
IAR SYSTEMS
DRAPER
Mellanox TECHNOLOGIES
tsmc
SEAGATE
codasip
FOUNDRIES.IO
ICT
HEX-Five Security
csem
Roa Logic
aselsan
LATTICE SEMICONDUCTOR
PRINCETON UNIVERSITY
Blockstream
dxcorr
TechanaLye
ALLWINNER
HUAWEI
expresslogic

MICROCHIP

RISC-V Members Through a Security Filter

- Defense Companies

THALES

NORTHROP GRUMMAN

aselsan

- Chip Organizations

Microsemi
a MICROCHIP company

Micron

ETH zürich

IA.

- Security IP

inside secure

Rambus
Cryptography Research

0x5 HEX–Five Security

DOVER
MICROSYSTEMS

INTRINSIC ID

codasip

TU Graz

es and Tools

ortuga
ogic

DATA 61

CSIRO

Blockstream

bluespec

galois

INTRINSIX

- Application Denied

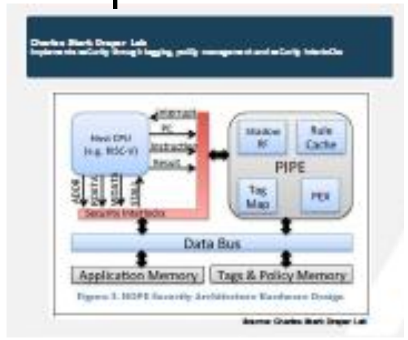
Some Notable RISC-V Security Activities in Academia

- Tagged memory, Enclaves, CFI protection, Side Channels, etc.
- Many cores incl. popular PULP cores, Timing channels mitigation
- Tagged memory (Cambridge Univ.)
- Keystone (TEE)
- Sanctum TEE

DARPA SSITH Program

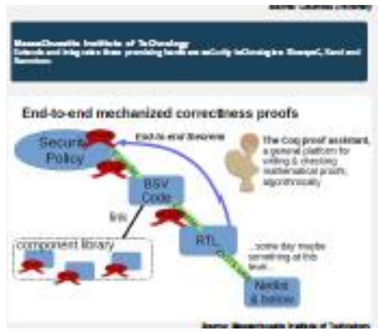
- **Addressing Cyber Threats w/ HW; ~$60M spend**
  - RISC-V mandated as demonstration vehicle
- **~Half-dozen HW performers**
- **Plus analysis and red teaming (Galois)**

Draper Labs



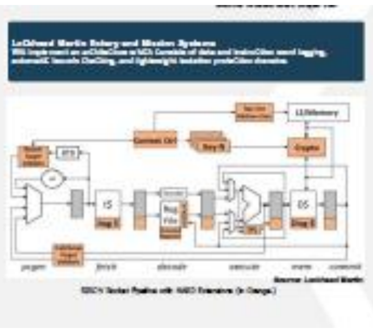
Multi-security policy enforcement w/ tags

MIT



Correctness Proofs

Lockheed Martin

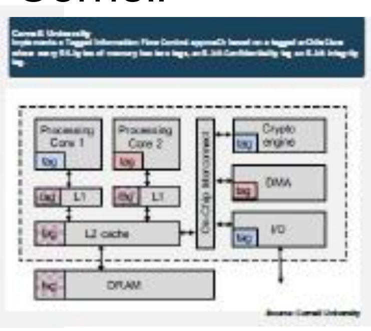


Isolation & bounds checking w/ tags

Univ. of Michigan

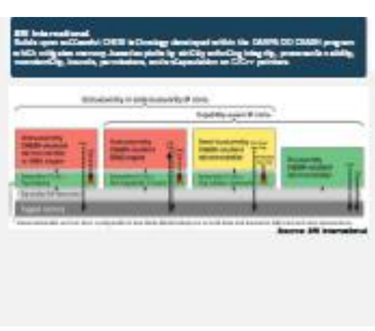
Cyber-attack countermeasure using churn

Cornell



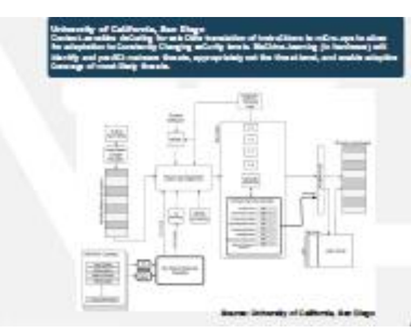
Information Flow Tracking

SRI International



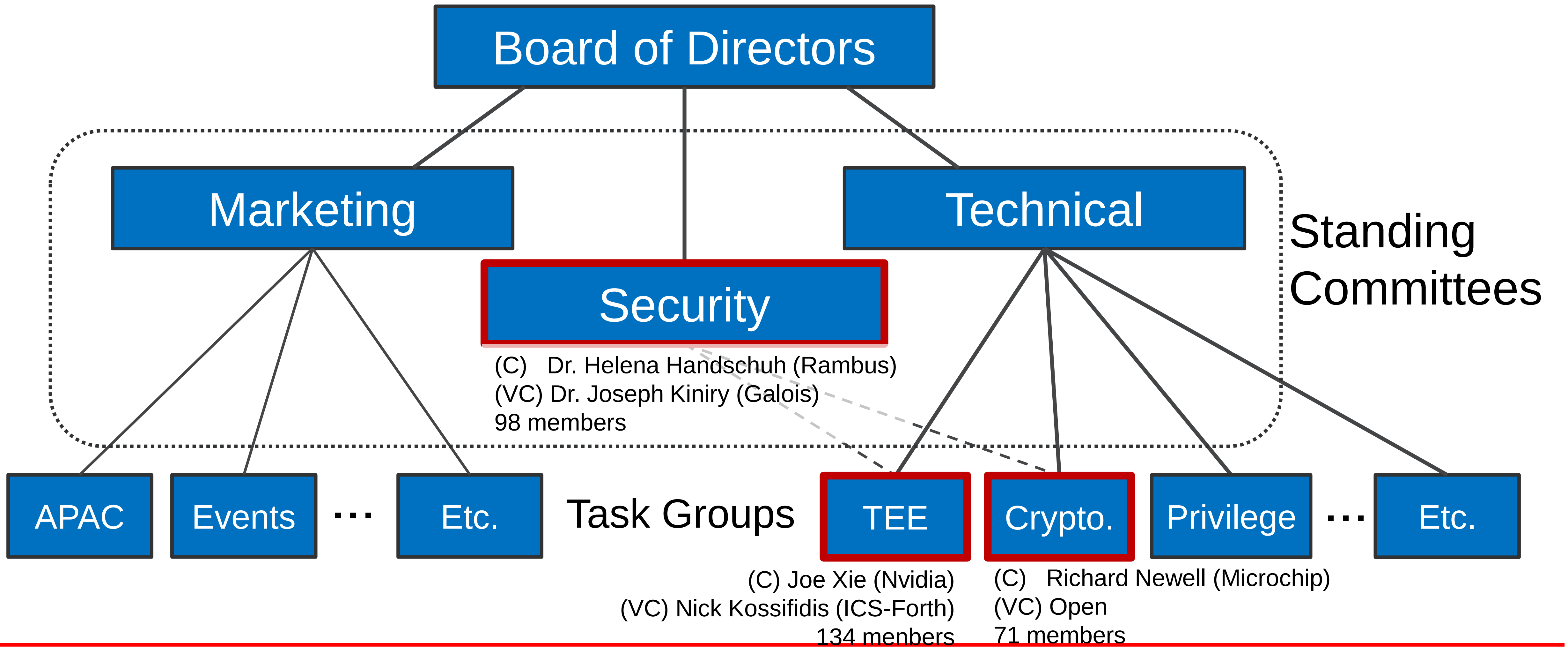
Memory protection w/ tags (CHERI)

UCSD

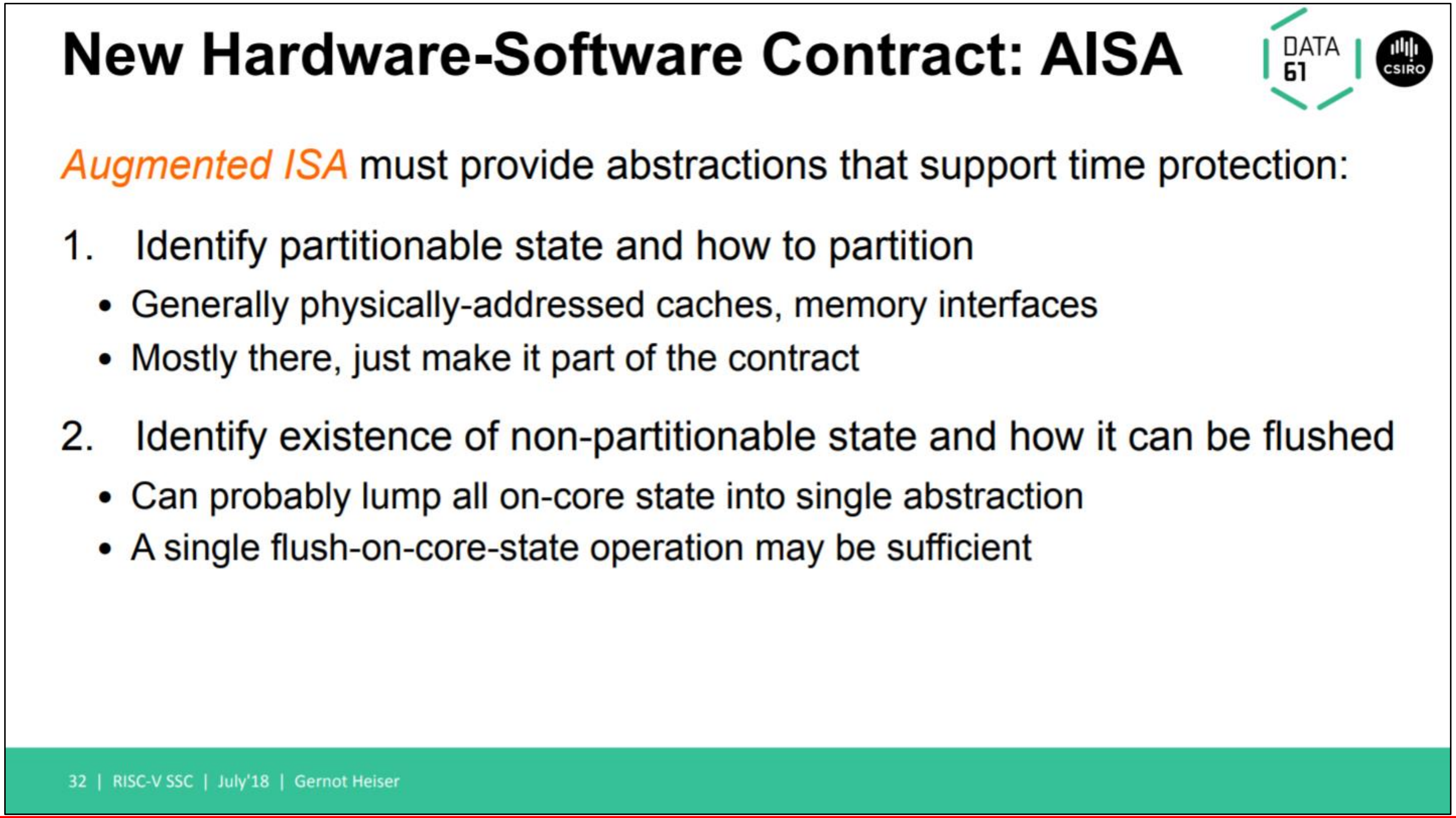


Adaptive use of encryption

RISC-V Foundation Security Groups Organization



Augmented ISA

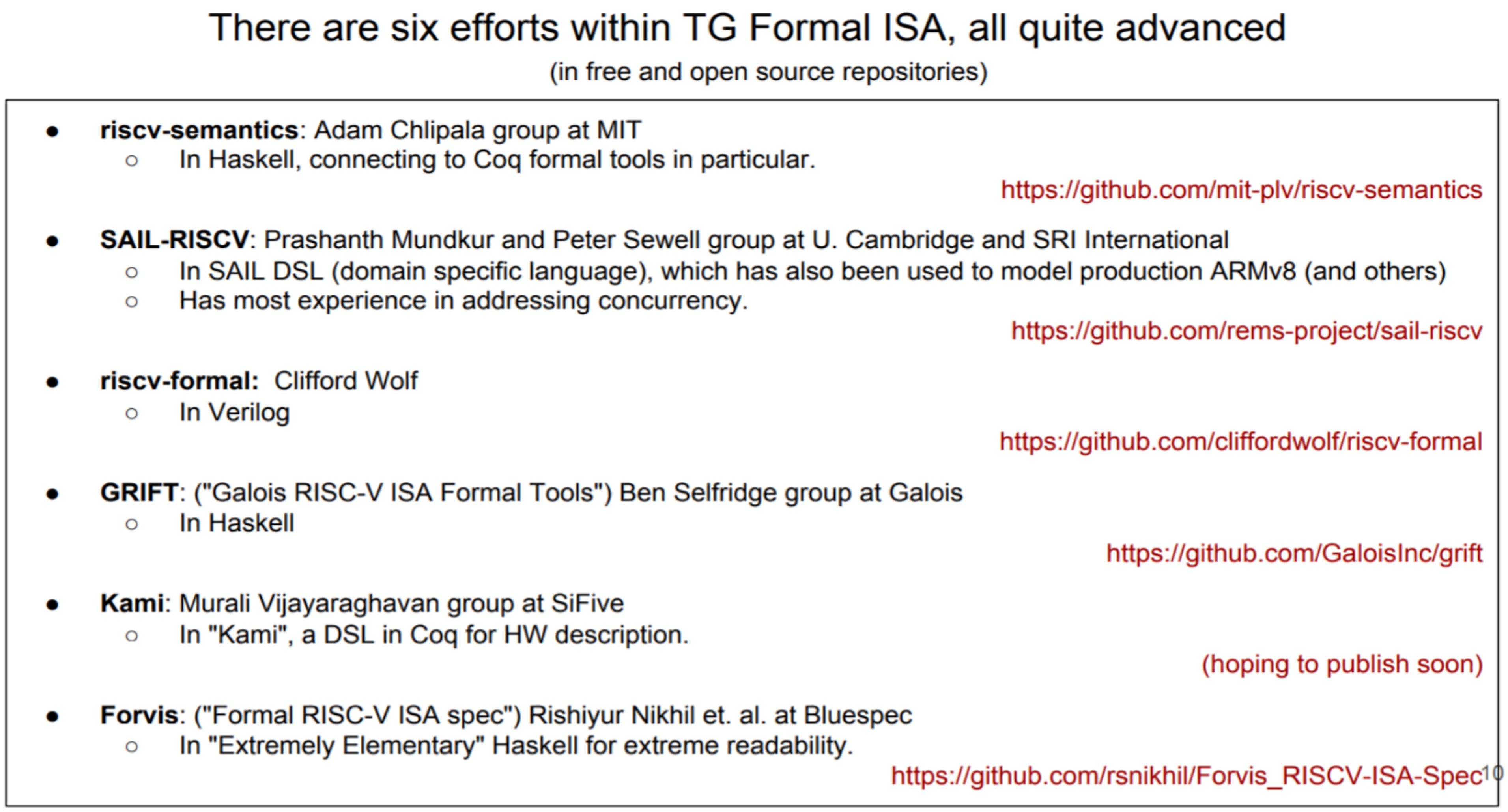


New Hardware-Software Contract: AISA

Augmented ISA must provide abstractions that support time protection:

1. Identify partitionable state and how to partition
   - Generally physically-addressed caches, memory interfaces
   - Mostly there, just make it part of the contract
2. Identify existence of non-partitionable state and how it can be flushed
   - Can probably lump all on-core state into single abstraction
   - A single flush-on-core-state operation may be sufficient

Activity of Note: Formal Spec



There are six efforts within TG Formal ISA, all quite advanced

(in free and open source repositories)

- **riscv-semantics**: Adam Chlipala group at MIT
  - In Haskell, connecting to Coq formal tools in particular.

https://github.com/mit-plv/riscv-semantics

- **SAIL-RISCV**: Prashanth Mundkur and Peter Sewell group at U. Cambridge and SRI International
  - In SAIL DSL (domain specific language), which has also been used to model production ARMv8 (and others)
  - Has most experience in addressing concurrency.

https://github.com/rems-project/sail-riscv

- **riscv-formal:** Clifford Wolf
  - In Verilog

https://github.com/cliffordwolf/riscv-formal

- **GRIFT**: ("Galois RISC-V ISA Formal Tools") Ben Selfridge group at Galois
  - In Haskell

https://github.com/GaloisInc/grift

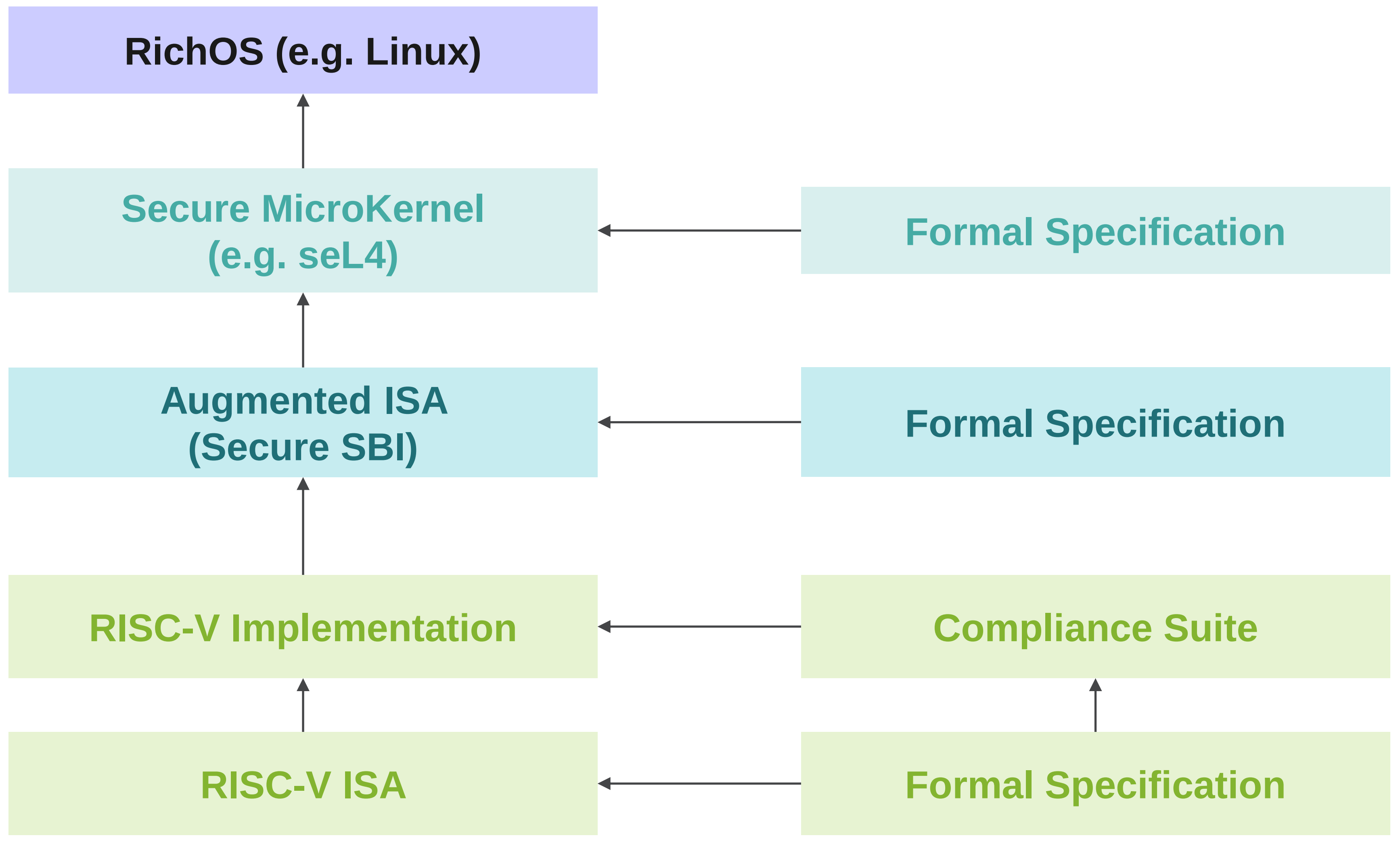
- **Kami**: Murali Vijayaraghavan group at SiFive
  - In "Kami", a DSL in Coq for HW description.

(hoping to publish soon)

- **Forvis**: ("Formal RISC-V ISA spec") Rishiyur Nikhil et. al. at Bluespec
  - In "Extremely Elementary" Haskell for extreme readability.

https://github.com/rsnikhil/Forvis_RISCV-ISA-Spec

Putting it all Together: The RISC-V Security Stack



About the TEE Task Group

- **One of the most popular groups (129 registered members)**
- **Regular conference calls / mailing list**
- **Its mission is:**
  - To define an architecture specification for supporting Trusted Execution Environments on RISC-V processors
  - To provide necessary implementation guidelines and/or recommendations in order to assist developers to realize the specification
  - To enable the development of necessary components (hardware and software) to support the specification

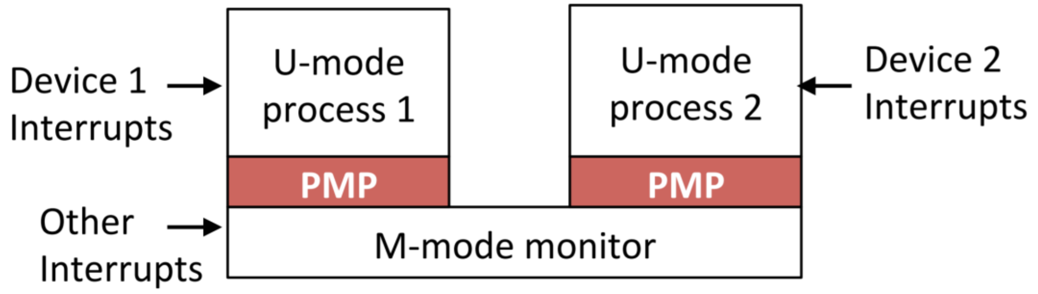
Work in progress

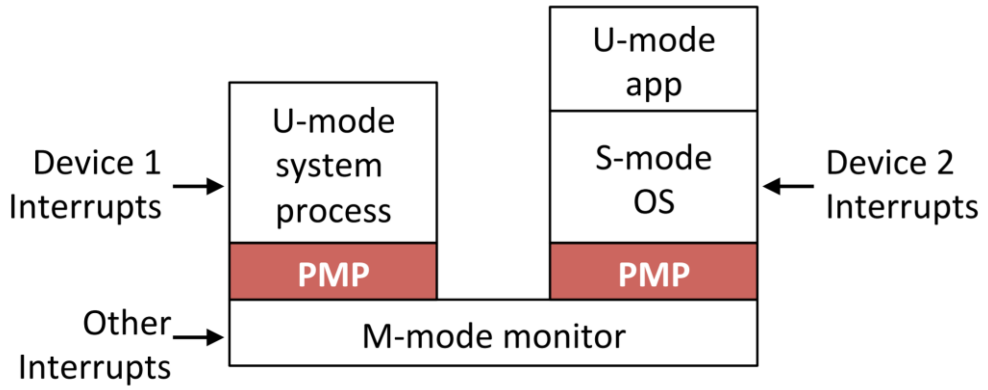
- **On the hardware side**
  - Modifications on the Physical Memory Protection (PMP) mechanism
  - Proposal for an I/O Physical Memory Protection (IOPMP) block
  - Proposal for a Control Flow Integrity (CFI) extension
- **On the software side**
  - Secure Monitor architecture
- **TODO**
  - Secure Boot
  - ...

RISCV TEE Core Arch

- **Embedded Profile**
  - M/U mode
  - Physical Memory Protection
  - (Optionally) User Mode Interrupt

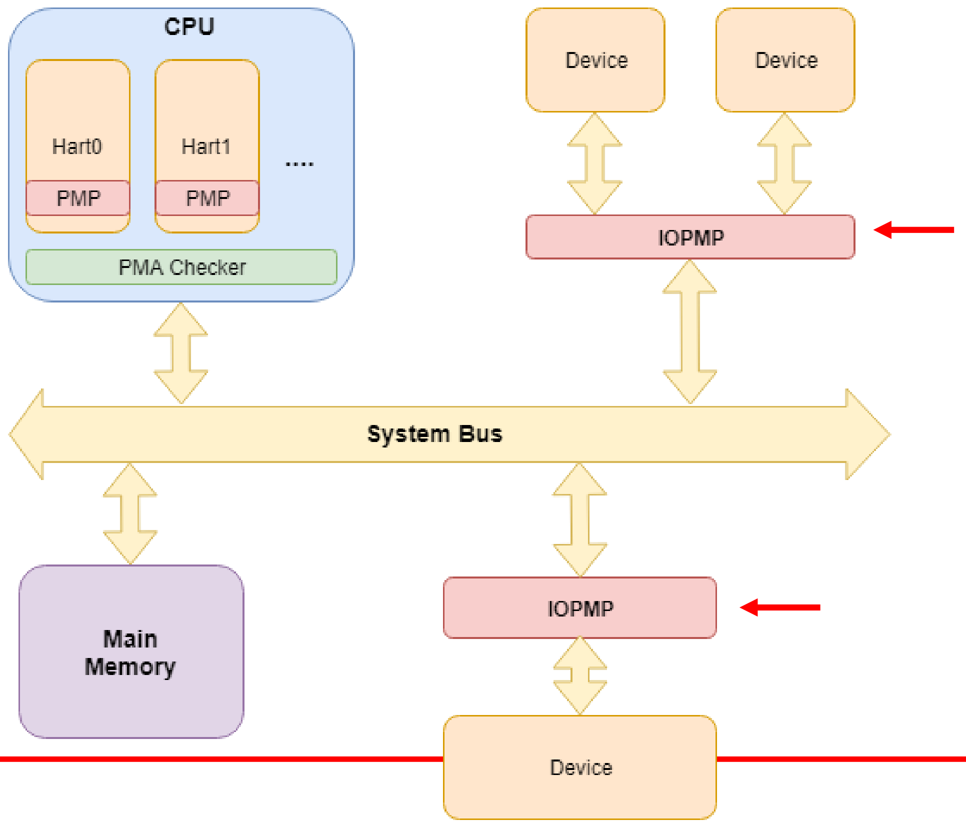
- **Application Profile**
  - M/S/U mode
  - Virtual Memory (SV32/SV39/SV48)
  - Physical Memory Protection (PMP)



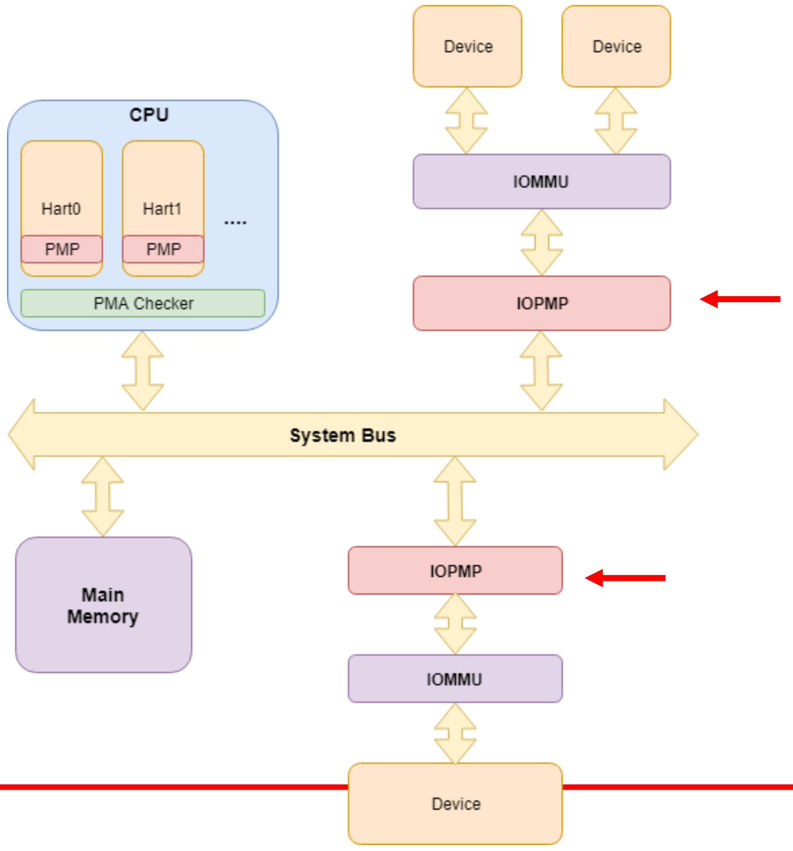


RISCV TEE SOC Arch

- **Embedded Profile**
  - IO Physical Memory Protection

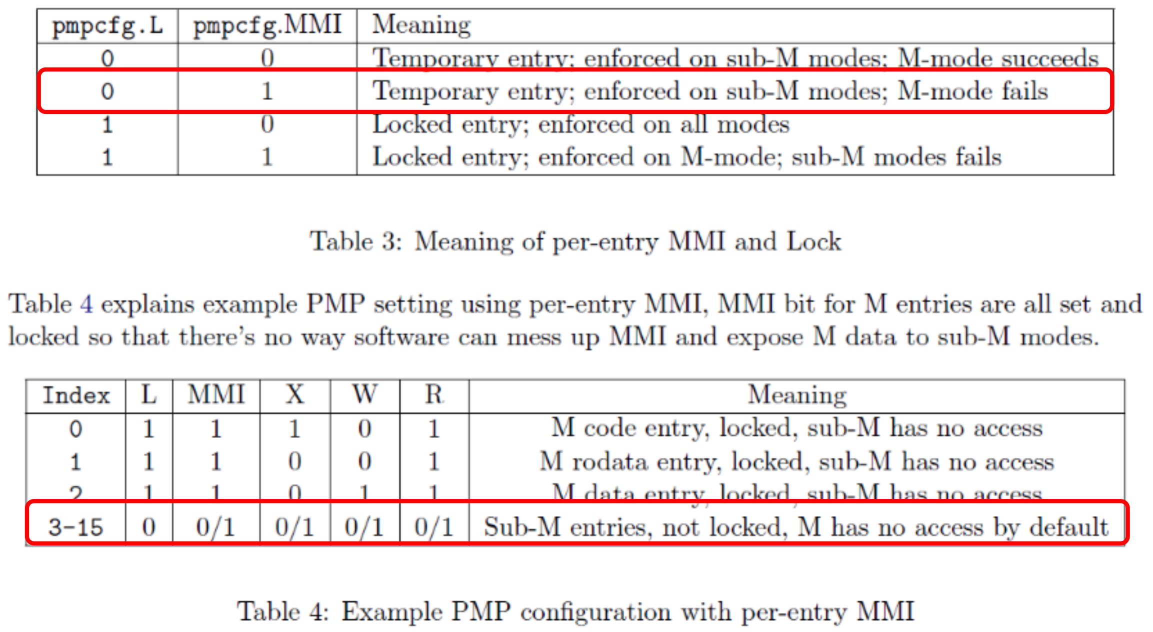


- **Application Profile**
  - SMMU/IOMMU
  - IO Physical Memory Protection



Proposed PMP modifications

- Rationale - Prevent M mode from accessing memory that belongs to S/U modes, to provide the equivalent of S mode's sstatus.SUM bit
- We want to have locked rules that are only enforced on M mode but not on S/U modes (e.g. to allow M mode to only have execute permission, without also allowing S/U to have the same privilege)
- Say hello to Machine Mode Isolation bit on mstatus (mstatus.MMI) !



pmpcfg.L	pmpcfg.MMI	Meaning
0	0	Temporary entry; enforced on sub-M modes; M-mode succeeds
0	1	Temporary entry; enforced on sub-M modes; M-mode fails
1	0	Locked entry; enforced on all modes
1	1	Locked entry; enforced on M-mode; sub-M modes fails

Table 3: Meaning of per-entry MMI and Lock

Table 4 explains example PMP setting using per-entry MMI, MMI bit for M entries are all set and locked so that there's no way software can mess up MMI and expose M data to sub-M modes.

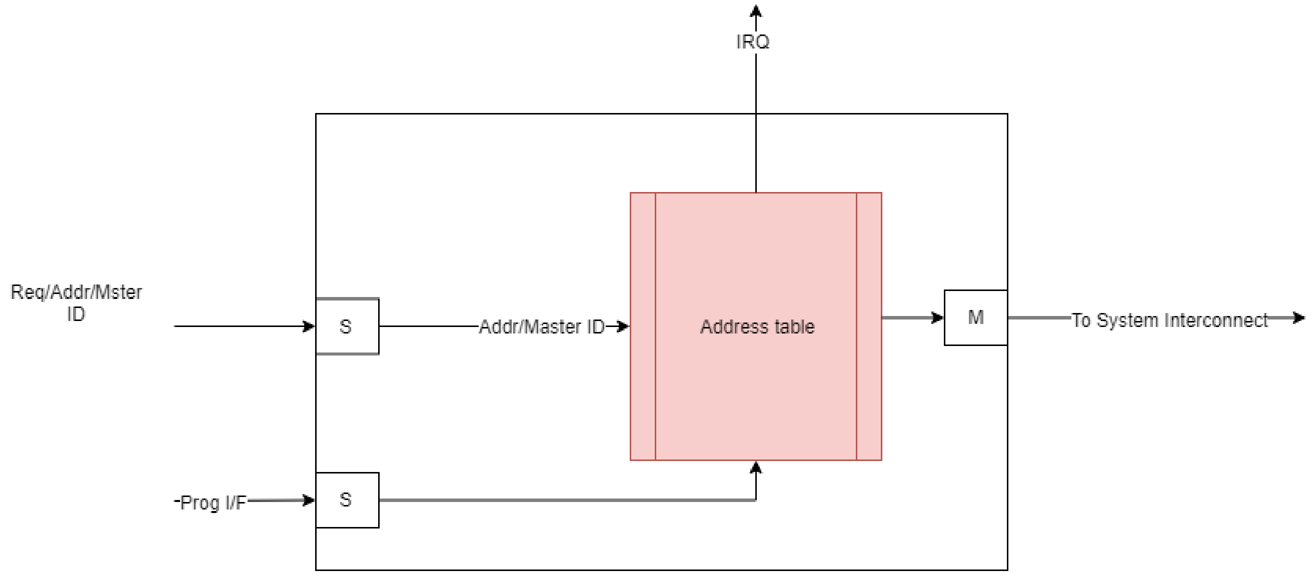
Index	L	MMI	X	W	R	Meaning
0	1	1	1	0	1	M code entry, locked, sub-M has no access
1	1	1	0	0	1	M rodata entry, locked, sub-M has no access
2	1	1	0	1	1	M data entry, locked, sub-M has no access
3-15	0	0/1	0/1	0/1	0/1	Sub-M entries, not locked, M has no access by default

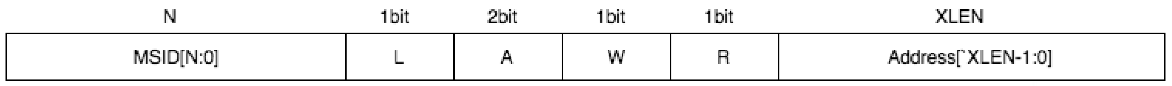
Table 4: Example PMP configuration with per-entry MMI

M-mode access to U/S Memory fails!

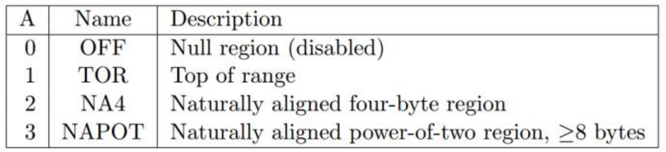
I/O Physical Memory Protection Proposal

- Protects physical memory from all memory masters in system
- Supports N memory masters sharing one IOPMP, or one IOPMP for one memory master
- Supports both 32bit and 64bit RISC-V implementations
- Scalable number of entries
- Supports error reporting



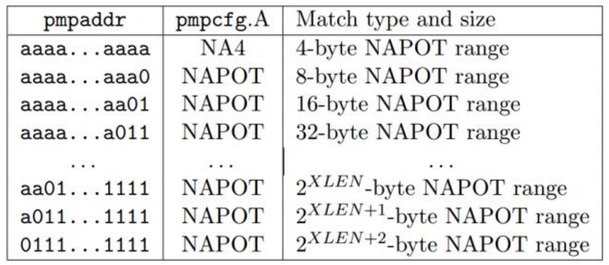


N	1bit	2bit	1bit	1bit	XLEN
MSID[N:0]	L	A	W	R	Address[`XLEN-1:0]



A	Name	Description
0	OFF	Null region (disabled)
1	TOR	Top of range
2	NA4	Naturally aligned four-byte region
3	NAPOT	Naturally aligned power-of-two region, $\geq$8 bytes

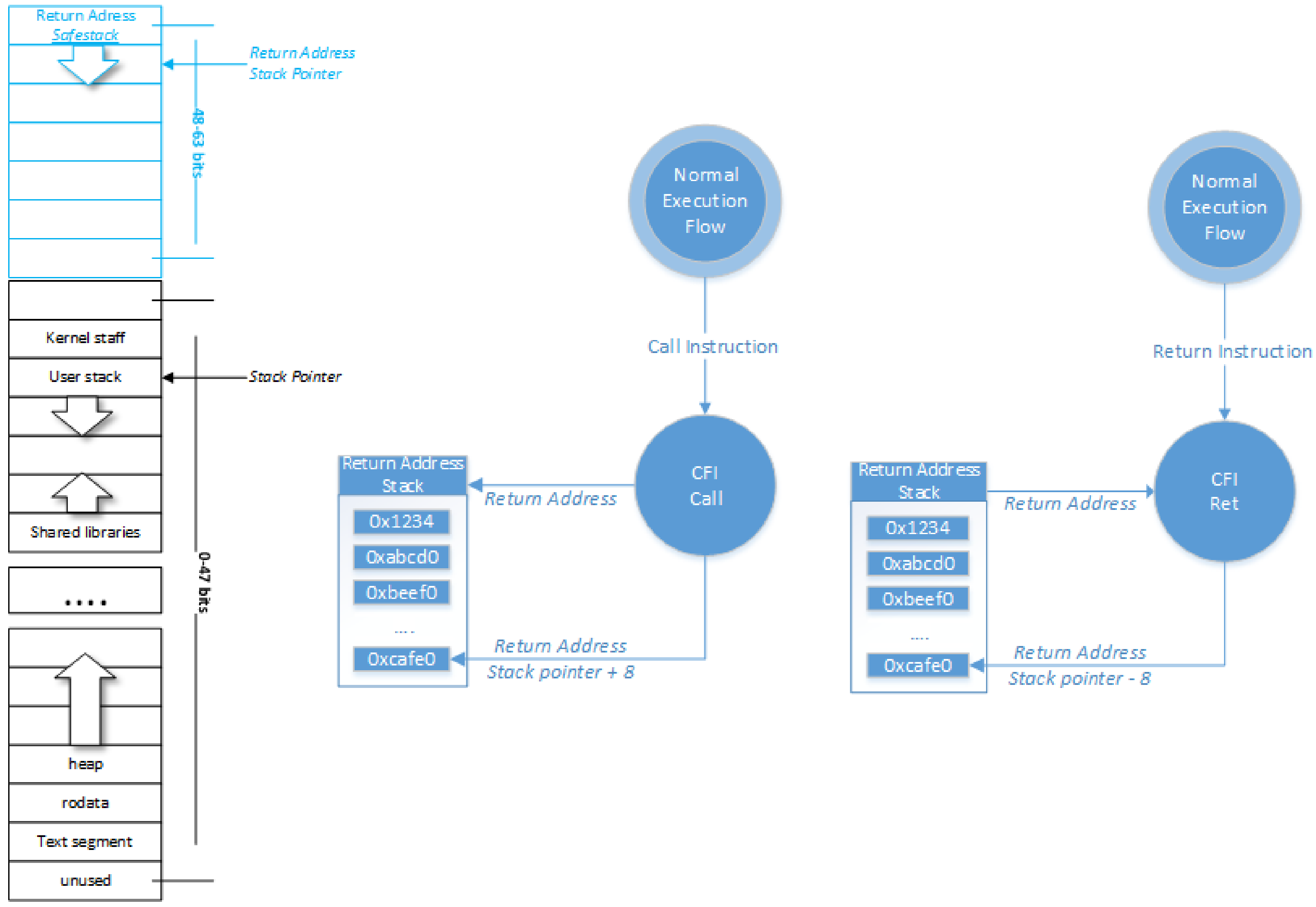
Table 3.8: Encoding of A field in PMP configuration registers.



pmpaddr	pmpcfg.A	Match type and size
aaaa...aaaa	NA4	4-byte NAPOT range
aaaa...aaa0	NAPOT	8-byte NAPOT range
aaaa...aa01	NAPOT	16-byte NAPOT range
aaaa...a011	NAPOT	32-byte NAPOT range
...	...	...
aa01...1111	NAPOT	2^{XLEN}-byte NAPOT range
a011...1111	NAPOT	2^{XLEN+1}-byte NAPOT range
0111...1111	NAPOT	2^{XLEN+2}-byte NAPOT range

Table 3.9: NAPOT range encoding in PMP address and configuration registers.

Control Flow Integrity extension proposal



Secure Monitor’s architecture

- **Current implementations from group members**
  - MultiZone™ from HexFive (https://hex-five.com/products/)
  - Keystone from UC Berkeley (https://keystone-enclave.org/)

- **A lot of work to be done !**
  - Define APIs between TEEs and between TEEs and the rest of the world (we need to work together with the upcoming platform specification task group e.g. for the SBI part)
  - Define a memory isolation scheme using PMP (there is a draft proposal on that)
  - Define a memory isolation scheme for I/O PMP
  - Define mechanisms for handling multiple harts
  - Define mechanisms for interupt handling / delegation
  - Define common format for TEE binaries (e.g. ELF with extras)
  - Write code for all of the above and test it
  - Provide an SDK
  - ...

Base Crypto Extension: AES Round-based instructions

- **These instructions perform a round of AES encryption or decryption**

```
vaese       vData,   vRndKey  # encrypt
vaeselast   vData,   vRndKey  # encrypt last round
vaesd       vData,   vRndKey  # decrypt
vaesdlast   vData,   vRndKey  # decrypt last round
```

`.vv` and `.vs` variants; maskable; SEW=128, `vrep` is ignored

Key Expansion commands not shown

- Data Input (`vData`) – Vector register with `vl` 128–bit elements
  - Input round: Input message plaintext (to be encrypted) or ciphertext (to be decrypted)
  - Other rounds: Current AES intermediate round state from previous round
- Key Input (`vRndKey`) – Vector with `vl` 128-bit round keys (**`.vv`**); or with `1` shared round key (**`.vs`**)
  - Previously computed from the AES Crypto key by key-expansion commands.
    - The round key can be pre-computed and stored or computed on-the-fly
    - Round keys are always 128 bits (AES Crypto key can be 128, 192, or 256 bits)
- Data output (`vData`) – 128-bits, overwrites Data Input (i.e., these commands are destructive)
  - Final round: Resulting final ciphertext (when encrypting) or plaintext (when decrypting)
  - Other rounds: Current AES intermediate round state

Extended Crypto Extension: AES All-Rounds Instructions

- **These instructions perform *all rounds* (10-14) of AES encryption or decryption**

```
vaese128   vData,  vKey      # encrypt (all 10 rounds), 128-bit raw AES key (w0-3)
vaese192   vData,  vKey      # encrypt (all 12 rounds), 2*SEW 192-bit raw AES key (w0-5)
vaese256   vData,  vKey      # encrypt (all 14 rounds), 2*SEW 256-bit raw AES key  (w0-7)

vaesd128   vData,  vRndKey   # decrypt (all 10 rounds), Last 128-bit round key (w40-43)
vaesd192   vData,  vRndKey   # decrypt (all 12 rounds), 2*SEW Last two round keys (w44-47, w48-51)
vaesd256   vData,  vRndKey   # decrypt (all 14 rounds), 2*SEW Last two round keys (w52-55, w56-59)
```

```
SEW = 128
For 192 and 256 the vData input/output are narrower (128 bits) than the 2*SEW (256 bit) key elements
```

- **Destructive – saves opcode space**
- **Vector-Scalar variant – key shared by all elements**
- **Key-expansion functionality built in (unlike the single-round instructions)**
  - vKey - standard AES key
  - vRndKey: last one or two standard round keys

Base Extension: SHA-2 family of secure hashes

- **Vector instructions for two underlying algorithms (polymorphic):**
  - **SHA-256:** Consumes 512 bits of message per 64 rounds (SEW=256)
  - **SHA-512**: Consumes 1024 bits of message per 80 rounds (SEW=512)
- **Four additional simple variants supported using above instructions**
  - **Based on SHA-256**: SHA-224
  - **Based on SHA-512**: SHA-512/224, SHA-512/256, SHA-384
- **4 vector registers (or groups)**
  - 2*SEW Message State - input message in 2*SEW chunks
  - Working State - intermediate state between rounds
  - Hash State - Accumulates final working state after each 60/84 rounds

SHA Vector Opcodes

- **These instructions perform 16 rounds of SHA-256 or -512:**

```
vsha2_ms   vms_dst,   vms_src          # Update message states by 16 rounds
vsha2_ws   vws,       vms,       rnd   # Update working states by 16 rounds
```

```
vms_dst: vector of vl (2*SEW) elements of the next message states
vms_src: vector of vl (2*SEW) elements of the previous message states
vms:     vector of vl (2*SEW) elements of the current message states
vws:     vector of vl (SEW) elements of the previous working states (input)
         and the next working states (after execution, i.e., destructive)
rnd:     Immediate value indicating first of next 16 rounds to work on:
         (0, 16, … 48) for SHA-256, (0,16,… 64) for SHA-512
```

- **This instruction performs all 64 (or 80) rounds:**

```
vsha2_hs   vhs, vm         # Update hash states (all rounds); may be DPA resistant
```

```
vhs:     vector of vl (SEW) elements of the current/next hash states
vm:      vector of vl (2*SEW) elements of the current input message chunks
```

Thank You!

Richard Newell
richard.newell@microchip.com