2nd TASER workshop

18th September 2022 in Leuven

MITSUBISHI ELECTRIC
Changes for the Better

PoC TEE:
Proof-of-Concept Implementation of RISC-V Trusted Execution Environment for Embedded Devices

S. Nashimoto*, R. Ueno†, N. Homma†

*Mitsubishi Electric Corporation

†Tohoku University

MITSUBISHI ELECTRIC
Changes for the Better

Motivation

- Security evaluation of RISC-V TEEs against physical attacks

Problem

- No RISC-V TEEs that 1) ran on actual devices, 2) was open-source, and 3) had permissive free license

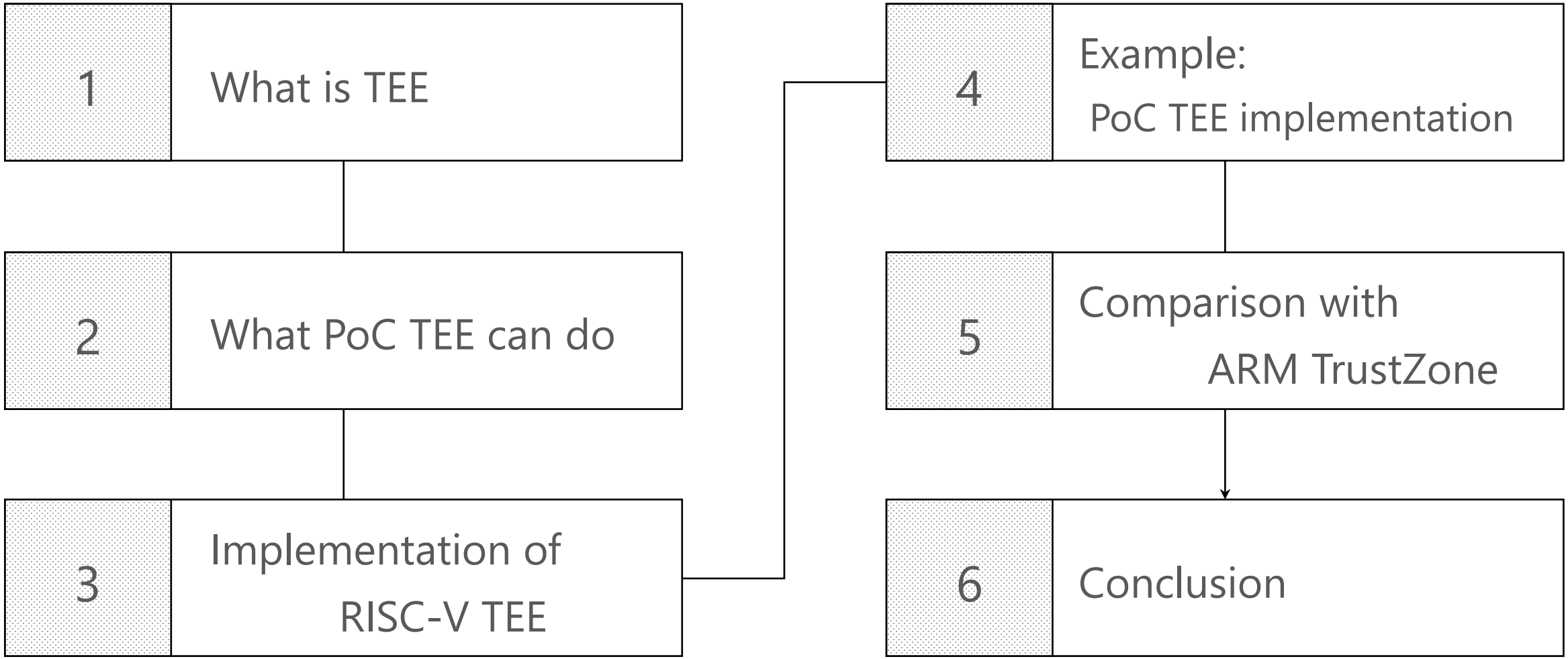
Solution

- Development and open-sourcing of bare metal TEE on RISC-V for embedded devices
- It's **RISC-V**, so you can modify both hardware and software

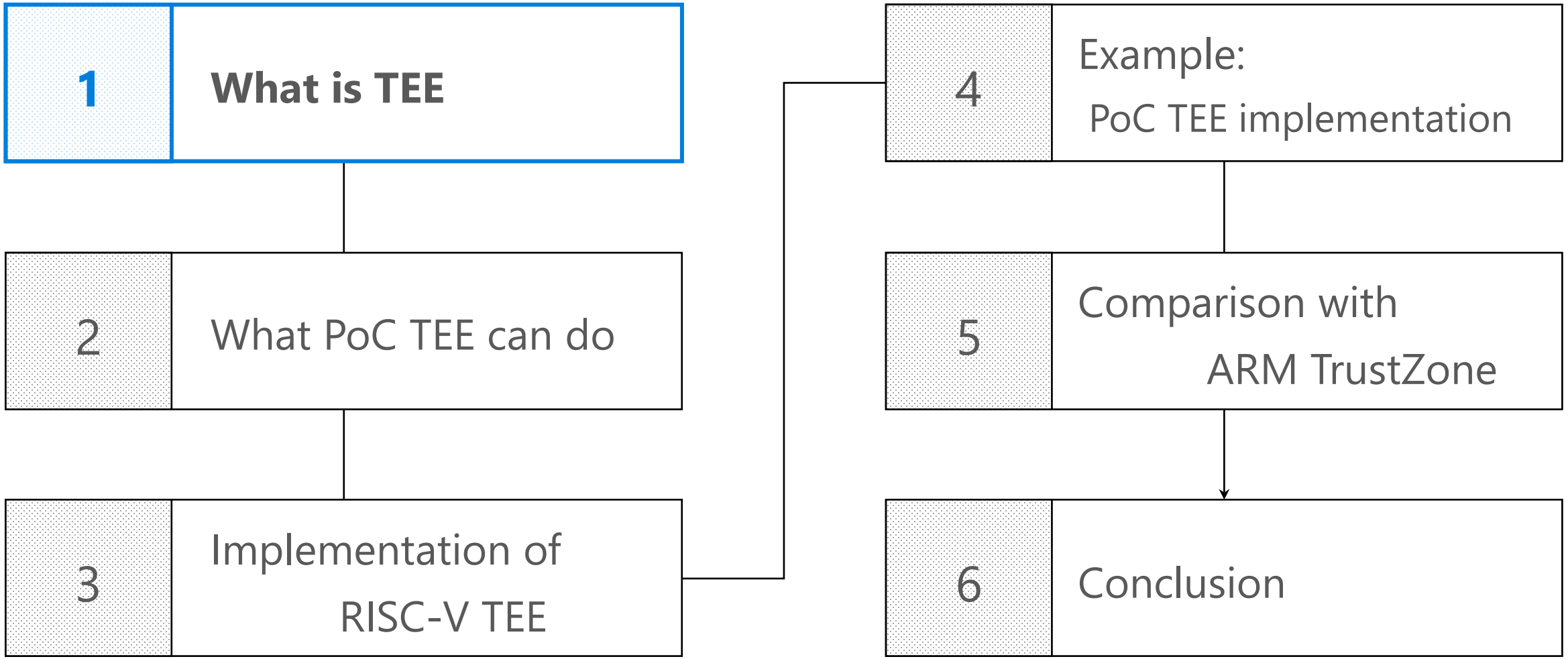
Comparison of famous RISC-V TEEs

Item	PoC TEE	MultiZone	Keystone Enclave	Penglai Enclave
OS	Bare metal	Bare metal	Linux	Linux
Board	Arty A7	Arty A7	HiFive Unleashed	X
Source code	✓	X	✓	✓
License	MIT/Apache2	Hex Five	UCB	Mulan PSL v1

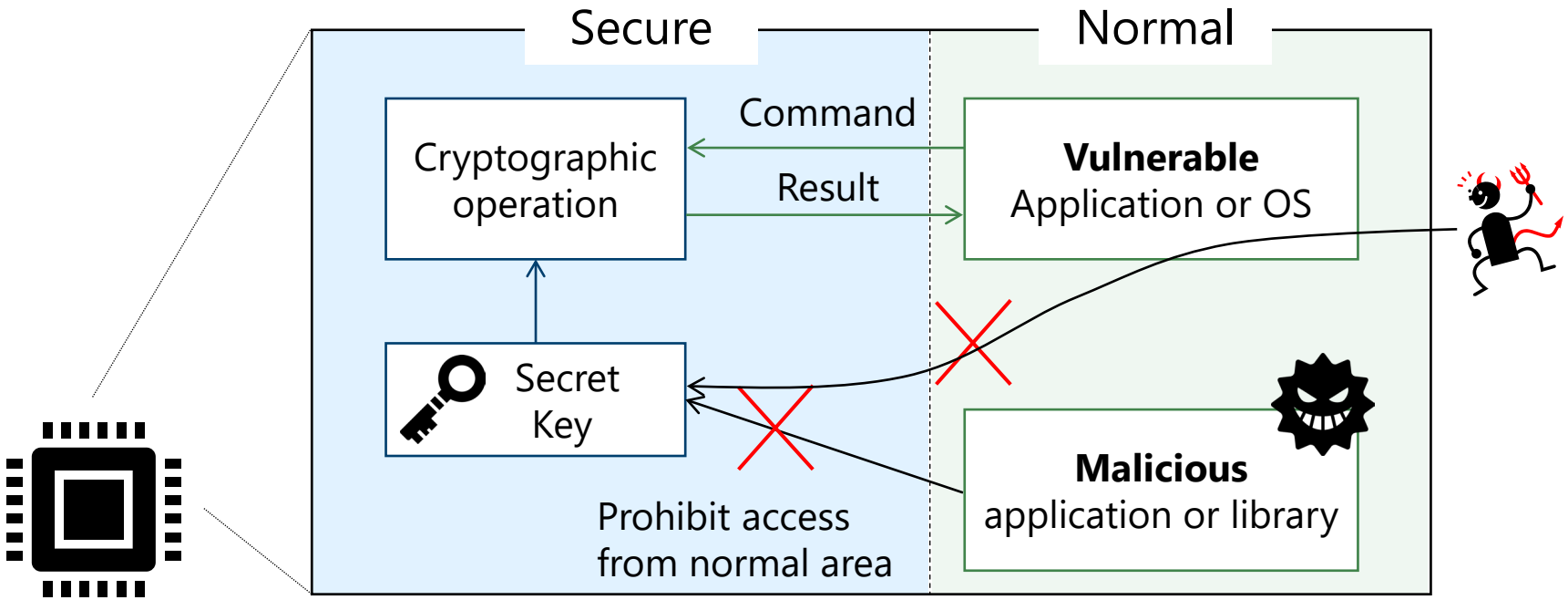
Outline



Outline

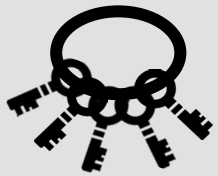


- **Secure area** of processor, isolated by hardware support
  - Promising countermeasure to protect secure apps from malicious and vulnerable apps
- Privilege alone is not sufficient
  - Continual reports of vulnerabilities in complex OSes
  - Protecting secure apps even in such critical situation



- TEEs for embedded devices have killer apps and are especially popular

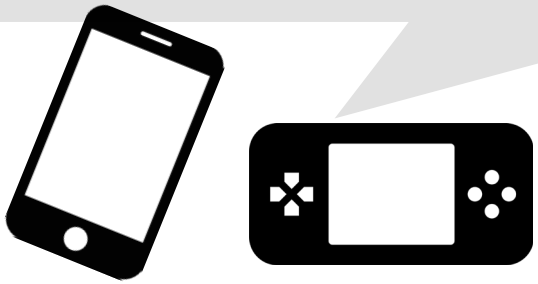
ARM TrustZone
(Android smart phone & game console)



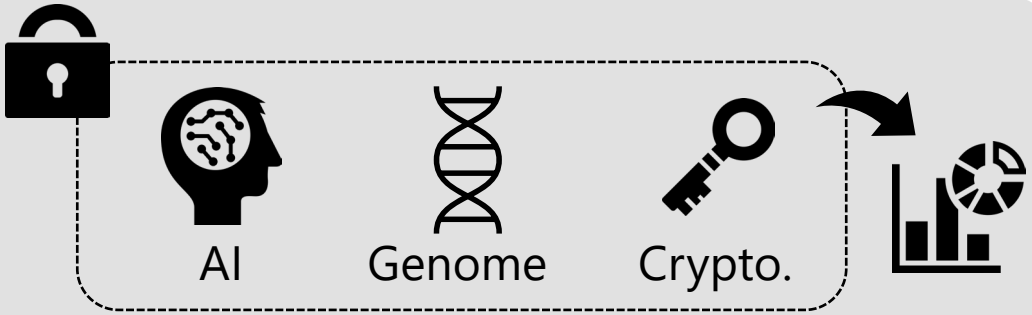
Key management

DRM

Biometrics



Intel SGX
(PC & server)



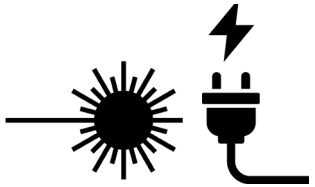
Confidentiality of data, code, and calculation

Attacks not covered by TEE

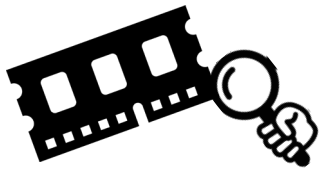
- TEE ensures **runtime memory protection**
  - i.e., attack from normal area to secure area
- Physical attacks, program analysis, compromising TEE deployment are out-of-focus

Physical attack

Side channel attack

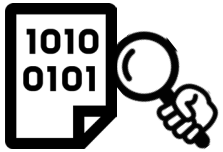


Fault injection attack



Bus probing

Program analysis



Static analysis of dumped binary

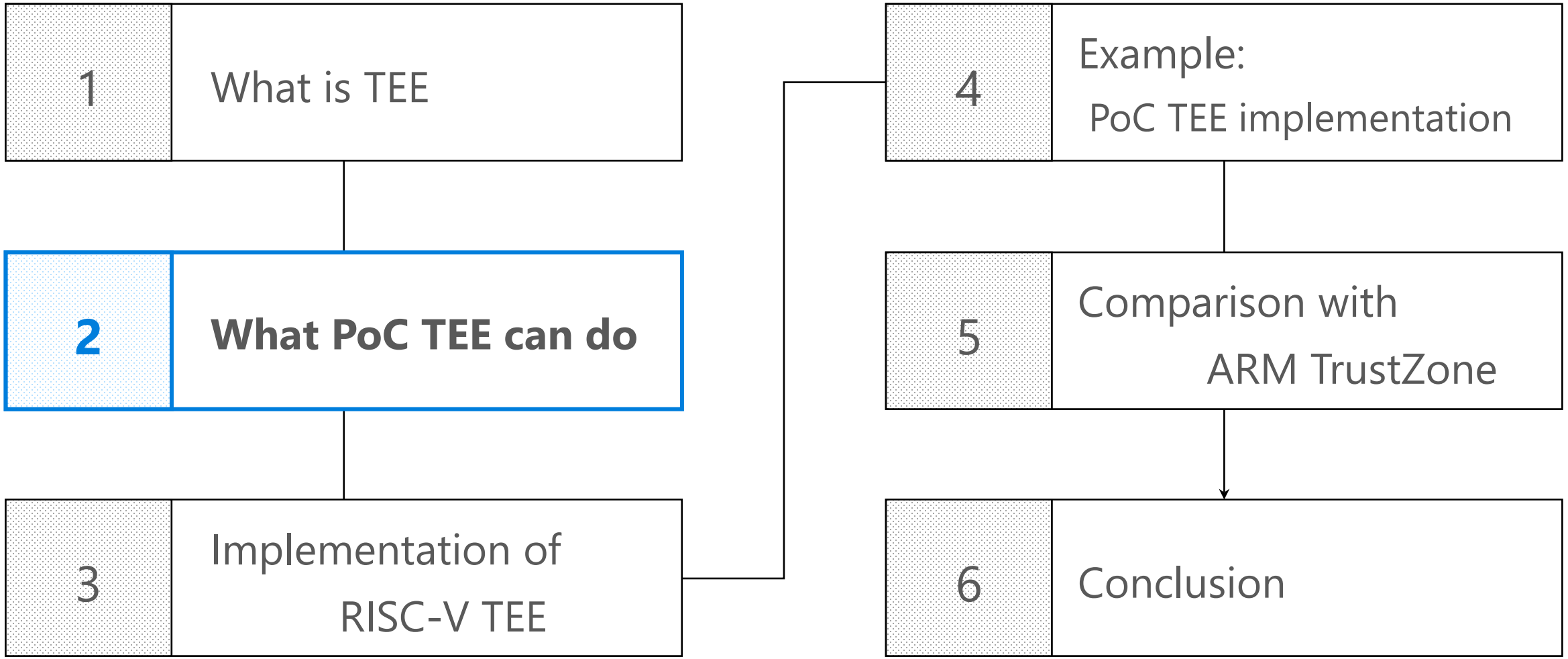
Dynamic analysis with debugger

Compromising TEE deployment

Vulnerability of secure app or TEE

Tampering with boot code

Outline



Three examples

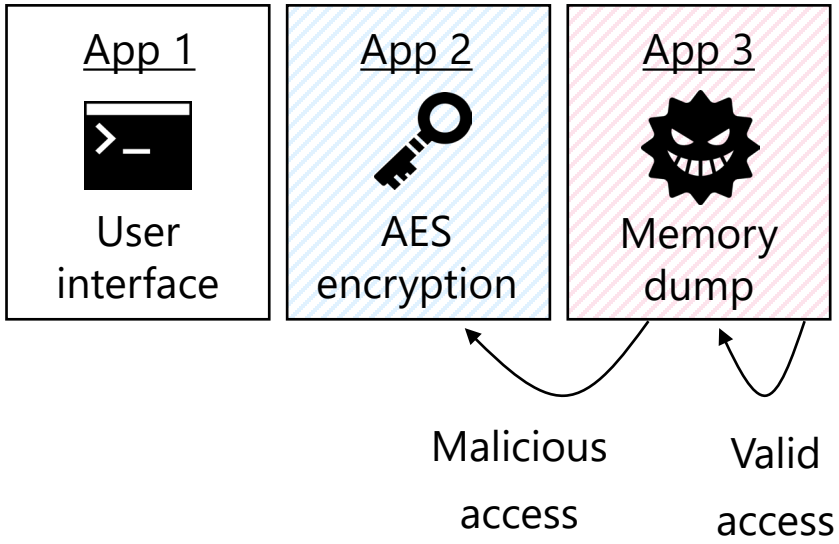
- Example 1: Confirmation of isolated execution (demo@GitHub)
- Example 2: Fault Attack Resistance Evaluation of RISC-V TEE (CHES'22)
- Example 3: Design and evaluation of TEE with additional attack resilience

Limitations

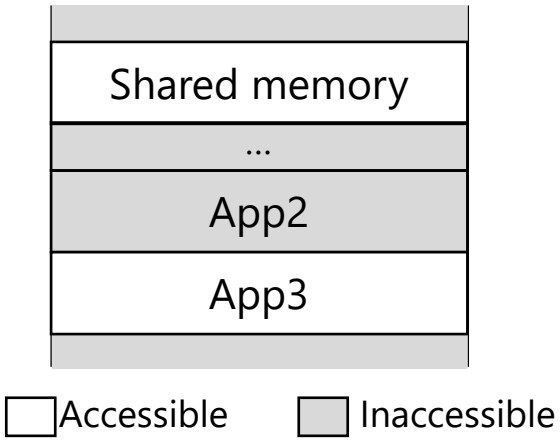
- Comparing to requirements from GlobalPlatform

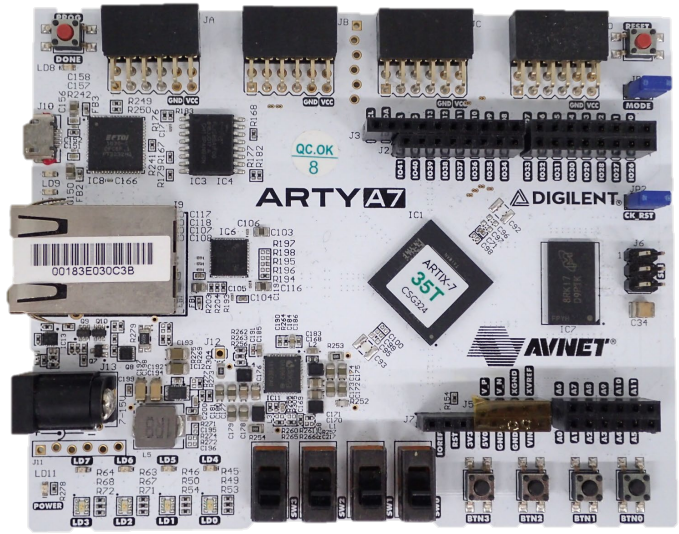
Ex. 1: Confirmation of isolation (setting)

- GitHub demo: PoC TEE denies malicious memory access
  - 3 apps are isolated each
  - App 3 tries to dump
    1) app 3's memory
    2) app 2's memory



TEE setting at App3 context (data memory)

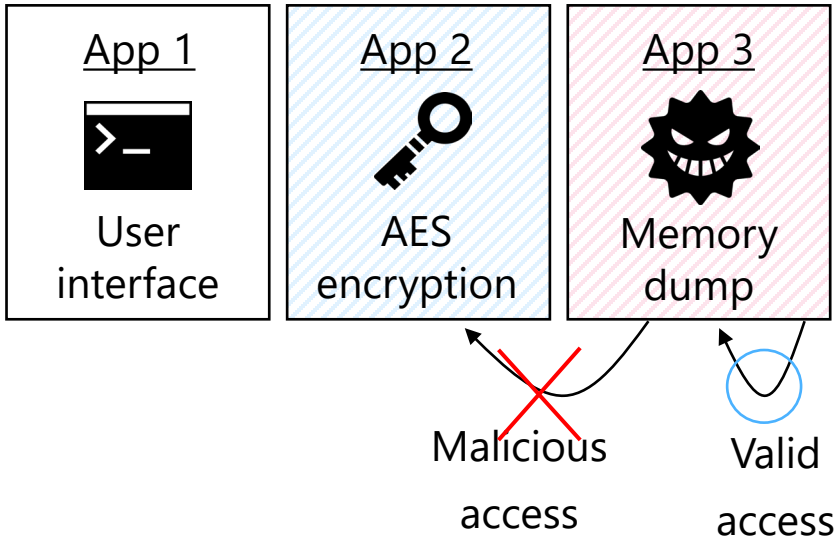




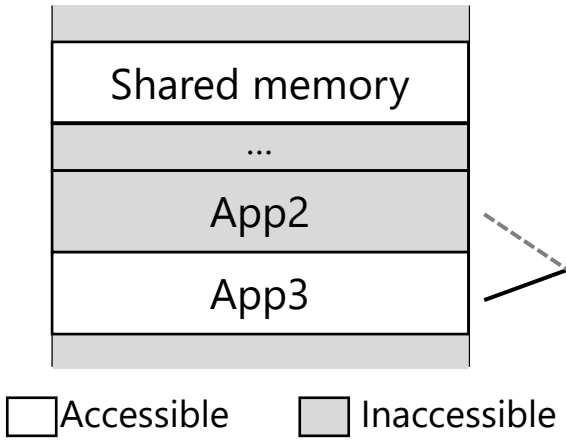
PoC TEE runs on RISC-V core on FPGA

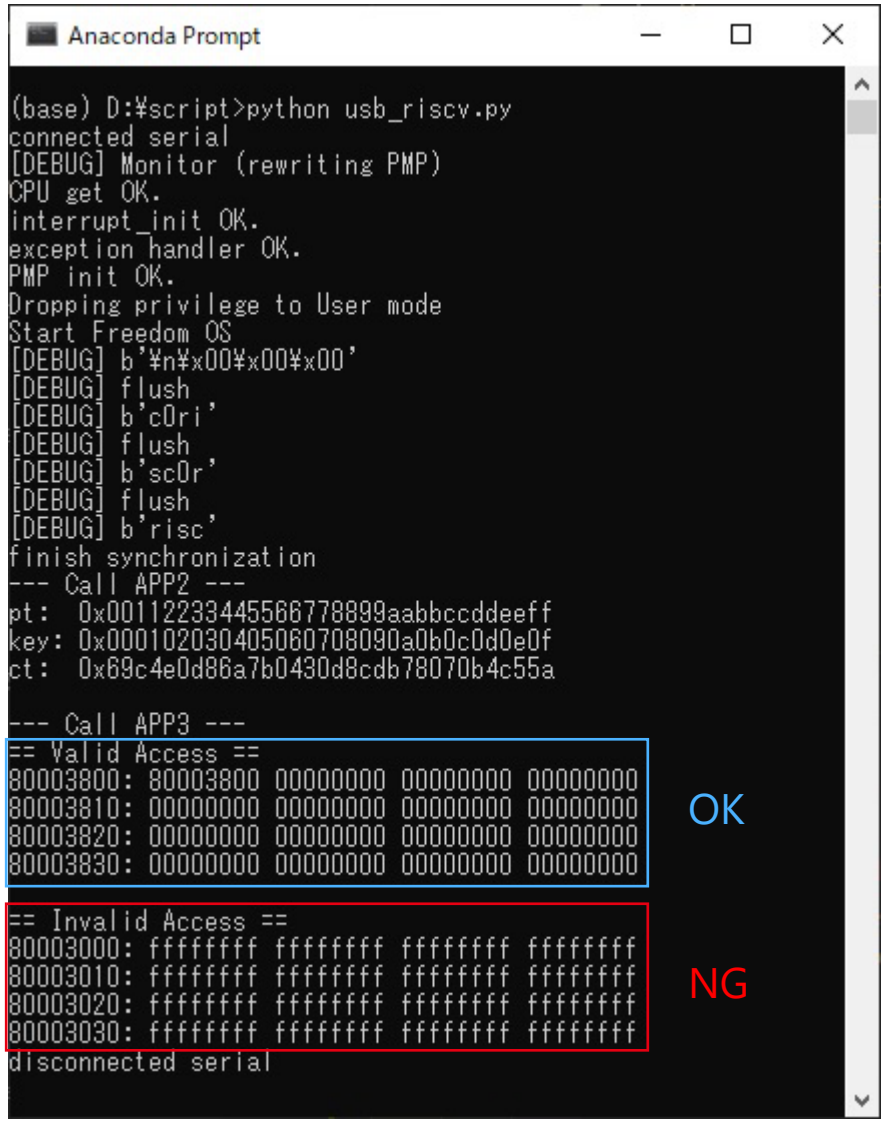
Ex. 1: Confirmation of isolation (result)

- GitHub demo: PoC TEE denies malicious memory access
  - 3 apps are isolated each
  - App 3 tries to dump
    1) app 3's memory → can be dumped
    2) app 2's memory → denied and 0xFF was returned



TEE setting at App3 context (data memory)

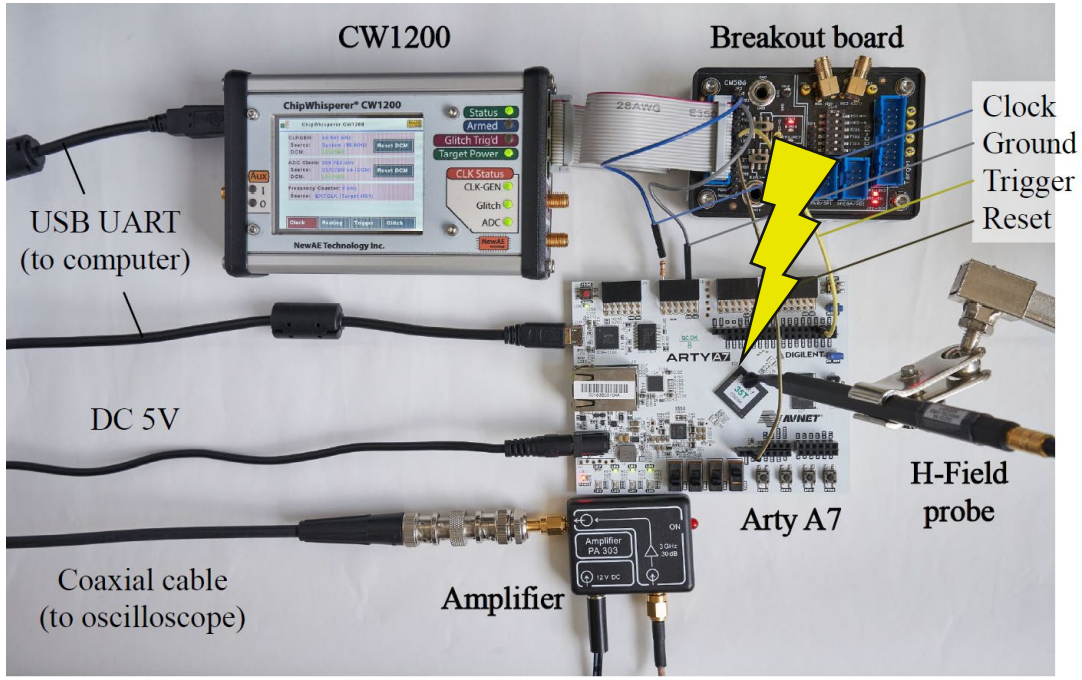




Ex. 2: Fault Attack Evaluation of RISC-V TEE

Q: If critical instructions are skipped by fault injection, could RISC-V TEE be compromised?

A: Yes. Please see CHES'22 (Wed, Sep 21)



15:50-16:50

RISC-V

Location TBA

Bypassing Isolated Execution on RISC-V using Side-Channel-Assisted Fault-Injection and Its Countermeasure
Shoei Nashimoto, Daisuke Suzuki, Rei Ueno, Naofumi Homma
Show abstract ›

Masked Accelerators and Instruction Set Extensions for Post-Quantum Cryptography
Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, Georg Sigl
Show abstract ›

Composite Enclaves: Towards Disaggregated Trusted Execution
Moritz Schneider, Aritra Dhar, Ivan Puddu, Kari Kostiainen, Srdjan Capkun
Show abstract ›

Sensors, Sensors, Sensors

Location TBA

VITI: A Tiny Self-Calibrating Sensor for Power-Variation Measurement in FPGAs
Brian Udugama, Darshana Jayasinghe, Hassaan Saadat, Aleksandar Ignjatovic, Sri Parameswaran
Show abstract ›

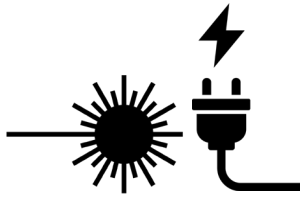
PreMSat: Preventing Magnetic Saturation Attack on Hall Sensors
Anomadarshi Barua, Mohammad Abdullah Al Faruque
Show abstract ›

A Power to Pulse Width Modulation Sensor for Remote Power Analysis Attacks
Brian Udugama, Darshana Jayasinghe, Hassaan Saadat, Aleksandar Ignjatovic, Sri Parameswaran
Show abstract ›

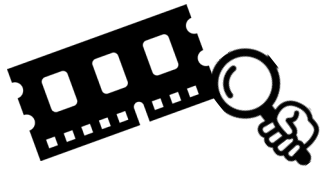
- CPU and ISA level countermeasure verification with RISC-V
  - e.g., resistance to attacks not covered by TEE

*ISA: Instruction Set Architecture

Tampering with boot code → TEE with secure boot using root-of-trust



Fault injection attack → TEE with control flow integrity using CFI extension



Bus probing → TEE with memory encryption using crypto accelerator

What you *cannot* do with PoC TEE

MITSUBISHI ELECTRIC
Changes for the Better

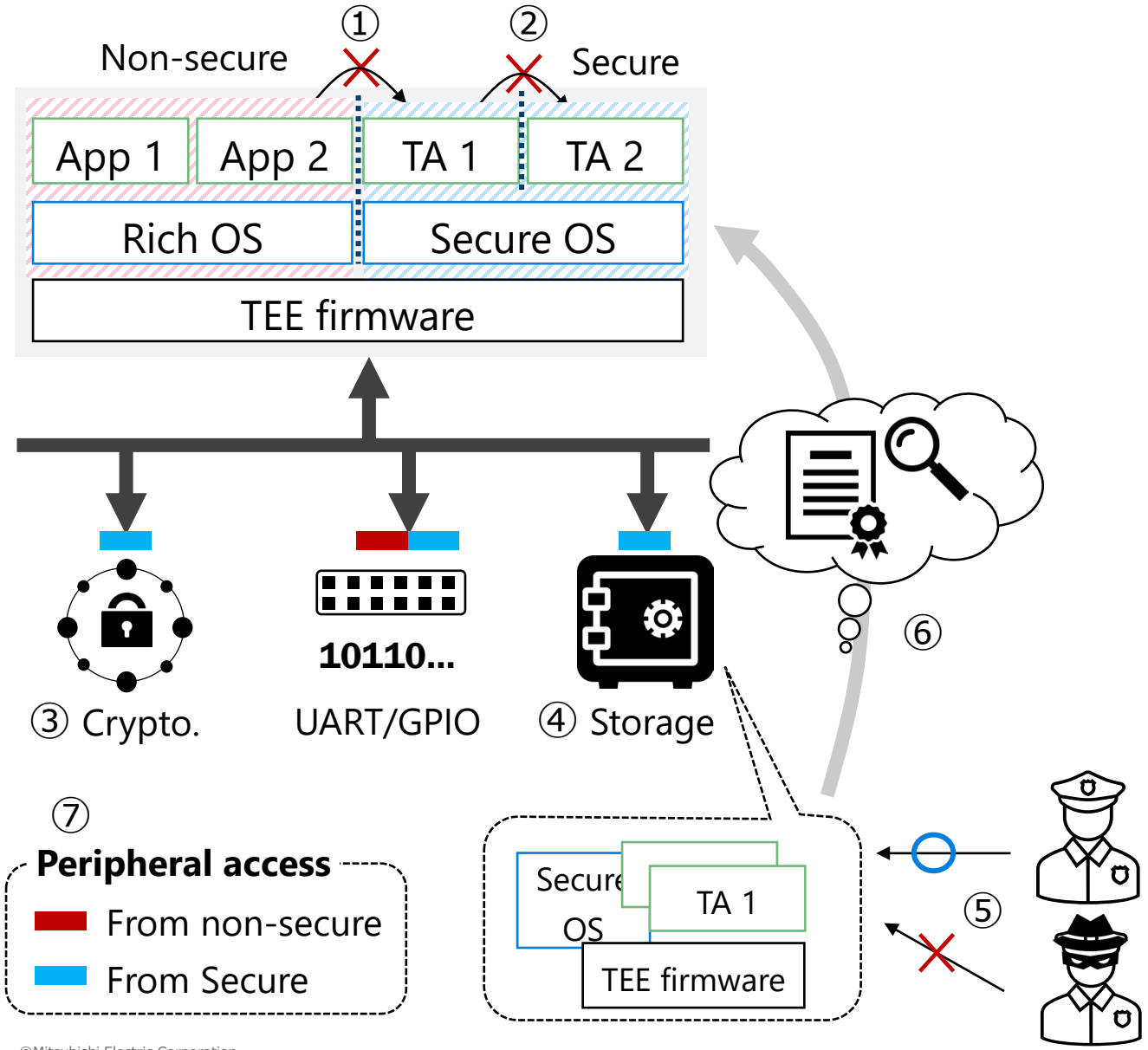
Three examples

- Example 1: Confirmation of isolated execution (demo@GitHub)
- Example 2: Fault Attack Resistance Evaluation of RISC-V TEE (CHES'22)
- Example 3: Design and evaluation of TEE with additional attack resilience

Limitations

- Comparing to requirements from GlobalPlatform

TEE requirements from GlobalPlatform*



① Isolation from the Rich OS

② Isolation from other TAs

③ State of the art cryptography

④ Trusted storage

⑤ Application management control

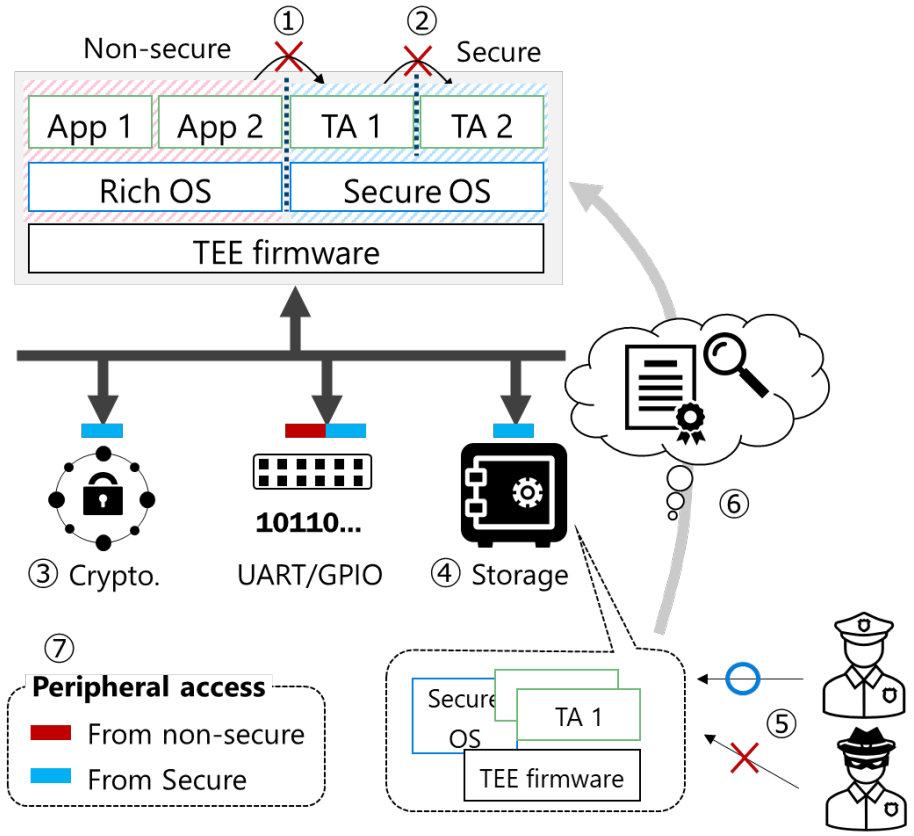
⑥ Identification and binding

⑦ Trusted access to peripherals

* GlobalPlatform, "Introduction to Trusted Execution Environments.", May, 2018.

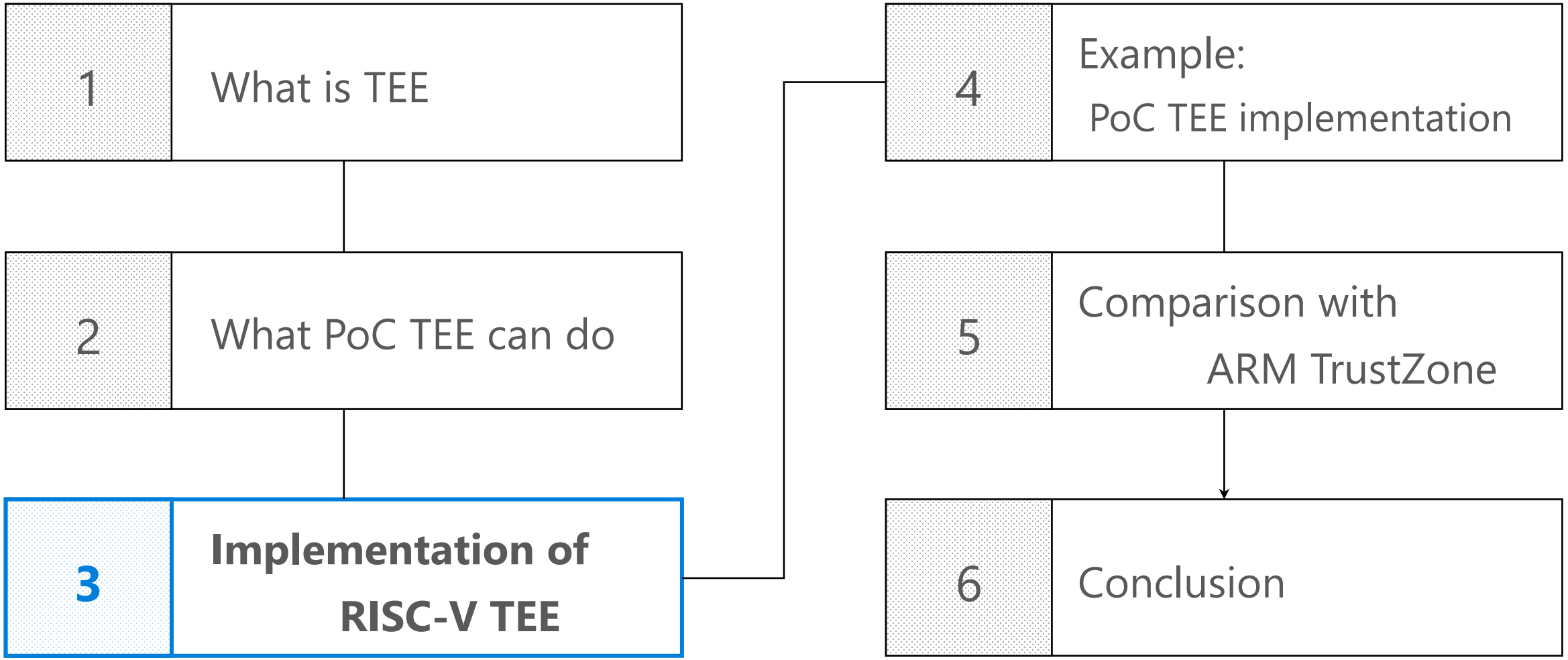
Limitations of PoC TEE

- PoC TEE only ensures **isolation of apps and peripherals**
- Lack of confidentiality, integrity, and authenticity of boot process and code
  - Hardware modification required



#	Requirements	PoC TEE	Remarks
①	Isolation from the Rich OS	N/A	Bare metal
②	Isolation from other TAs	✓	-
③	State of the art cryptography	-	No Crypto. core, but can implement
④	Trusted storage	-	No dedicated storage
⑤	Application management control	-	No code encryption or signature
⑥	Identification and binding	-	No secure boot
⑦	Trusted access to peripherals	✓*	*Rewriting method (described later) only

Outline



Goal

Control access to ROM, RAM, and peripherals for 1) each app or 2) apps between secure and non-secure

① Memory Access Monitoring
Requires to check memory access settings and decide whether it is allowed.

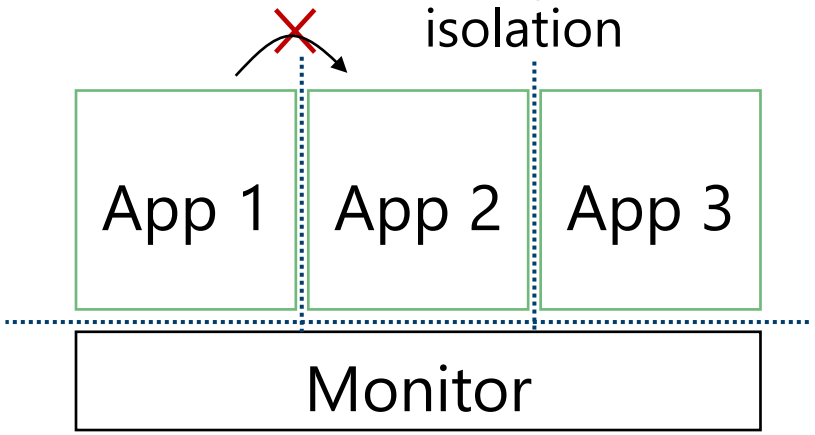
→ **Physical Memory Protection (PMP)**

② Privilege Level
Requires separation of access rights to security settings in ISA level.

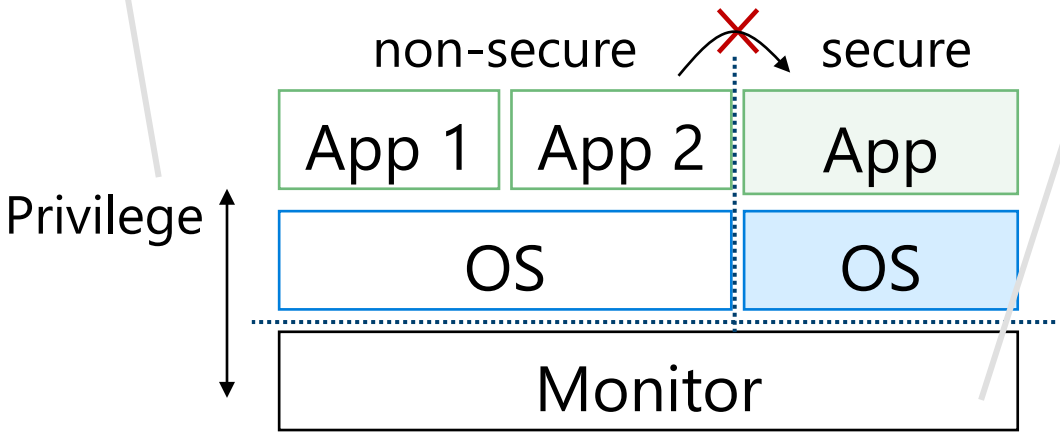
→ **Privilege mode**

③ Context Switch Management
Requires management of memory access settings per context.

→ **Monitor**

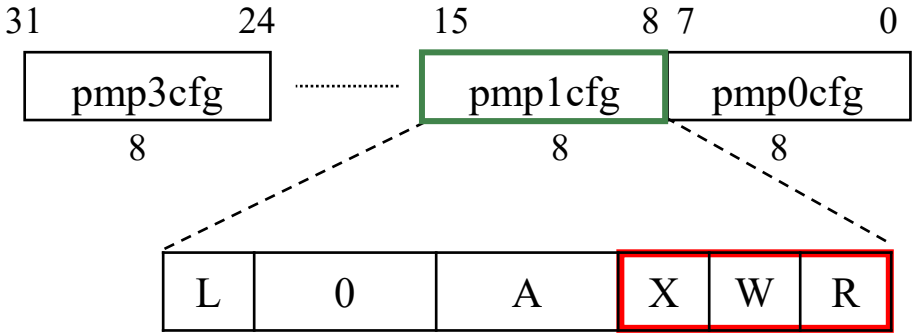


1) Isolate each application



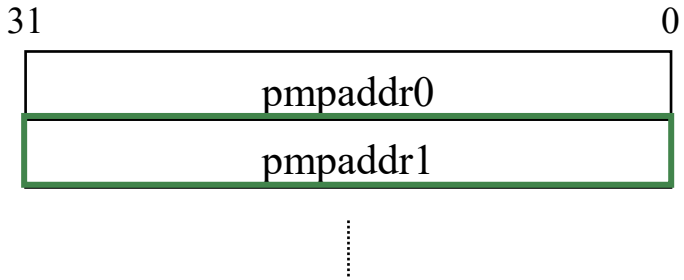
2) Isolate two worlds

① Physical Memory Protection (PMP)



Config register

(*pmpcfg*)



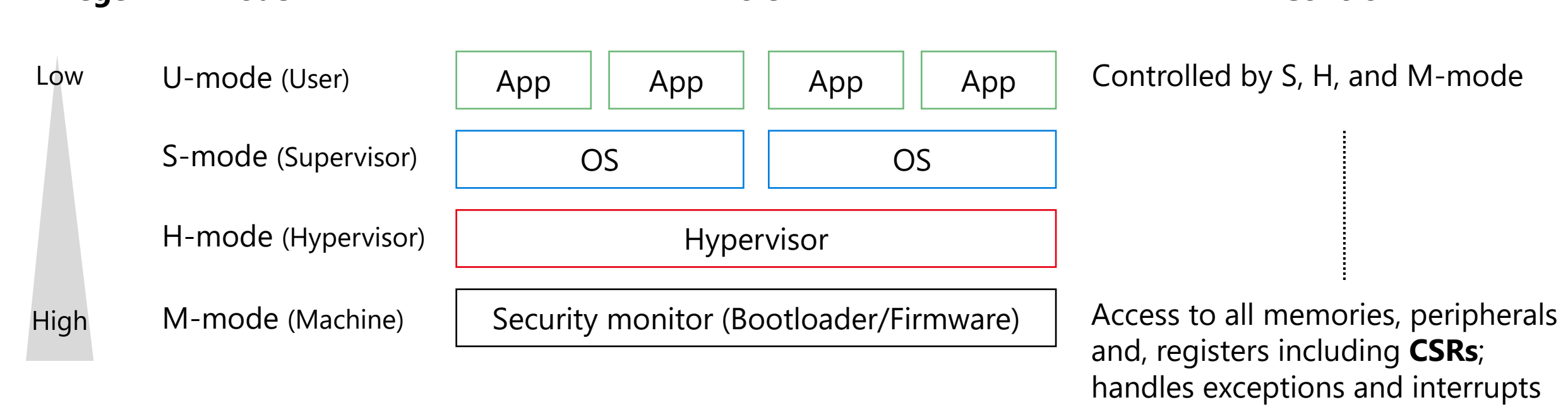
Address register

(*pmpaddr*)

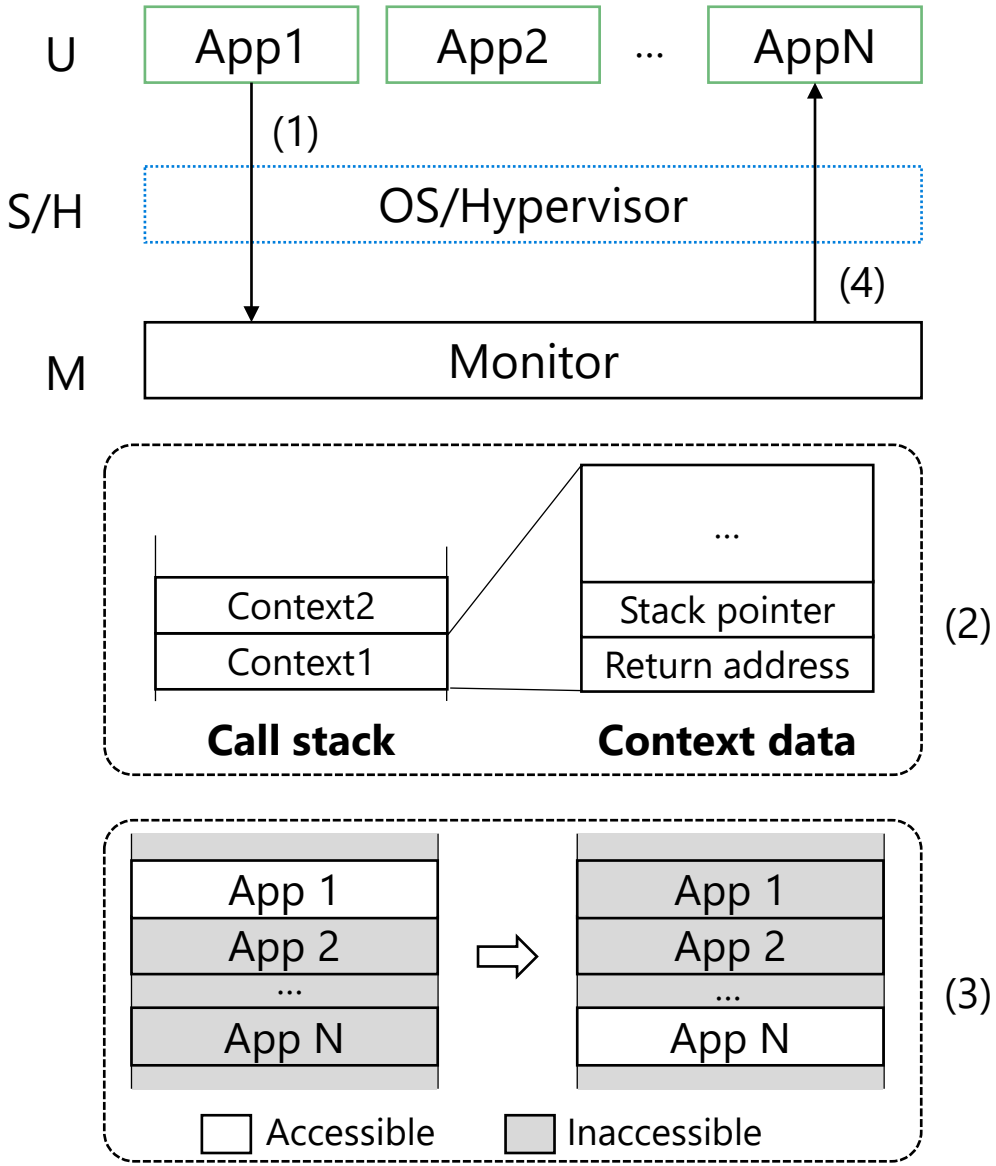
- Assign "write, read, and execute" permissions to specific memory ranges
  - *pmpcfg* defines permission
  - *pmpaddr* defines covered address
  - **PMP entry** (pair of *pmpicfg* and *pmpaddri*) manages one memory range
- All memory accesses are monitored
  - Assert **access fault exception** in case of invalid access

② Privilege mode

- Hierarchy with high/low privilege levels for memory access and exception handling
- **Only m-mode** can handle control and status registers (CSRs) including PMP



③ Monitor: context switch management



Step 1:

Call monitor by causing “environmental call exception” with *ecall*

Step 2:

Save and restore context

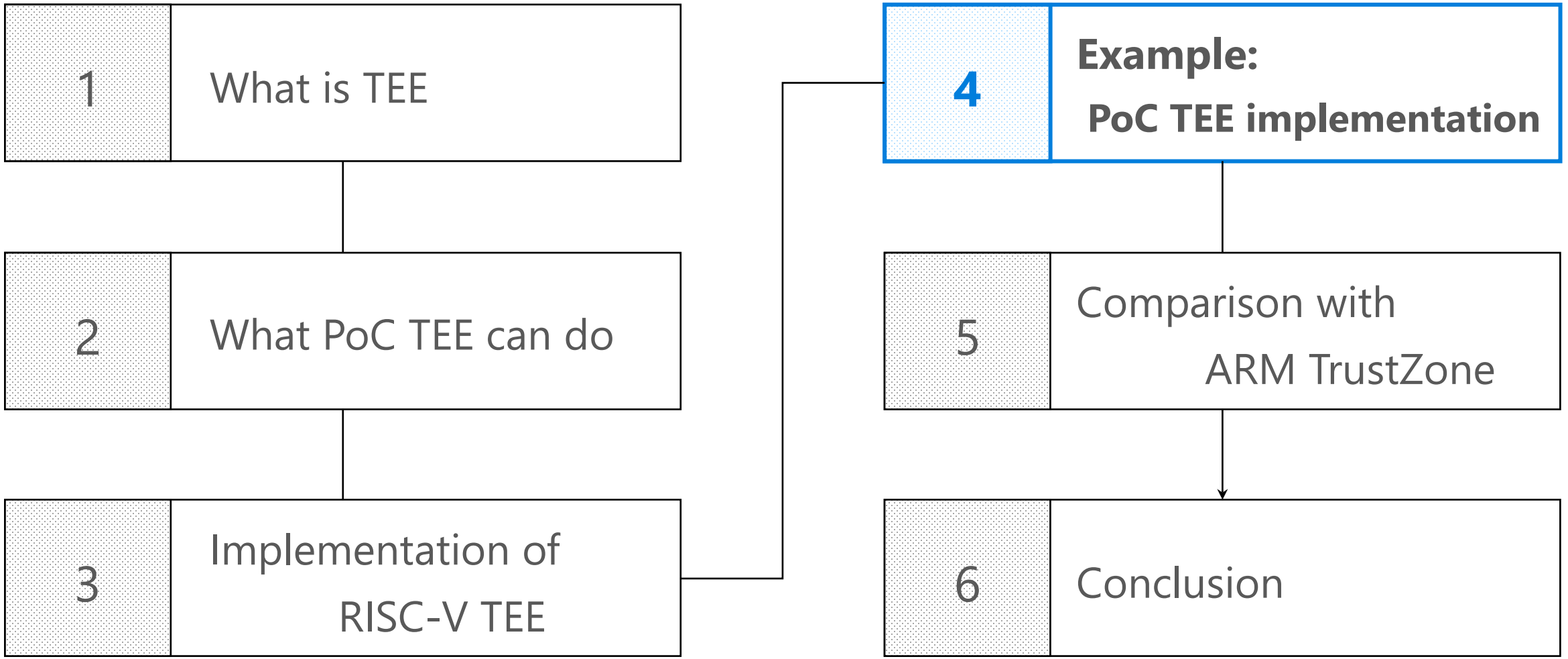
Step 3:

PMP Reconfiguration to change access permission

Step 4:

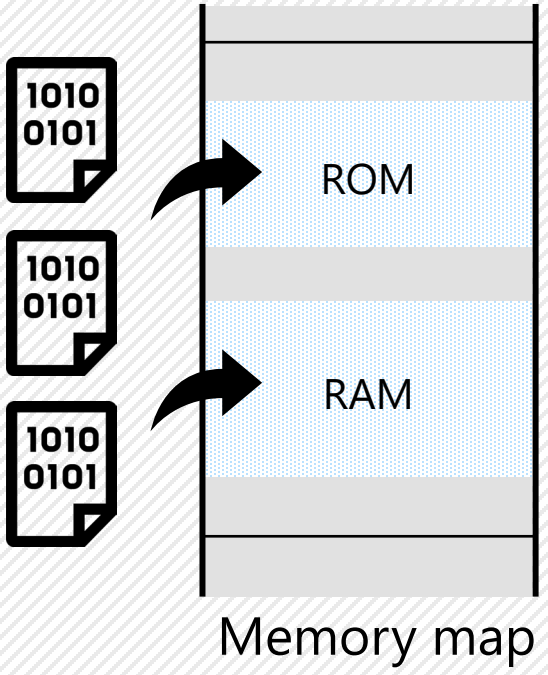
Call app by dropping privilege with *mret*

Outline



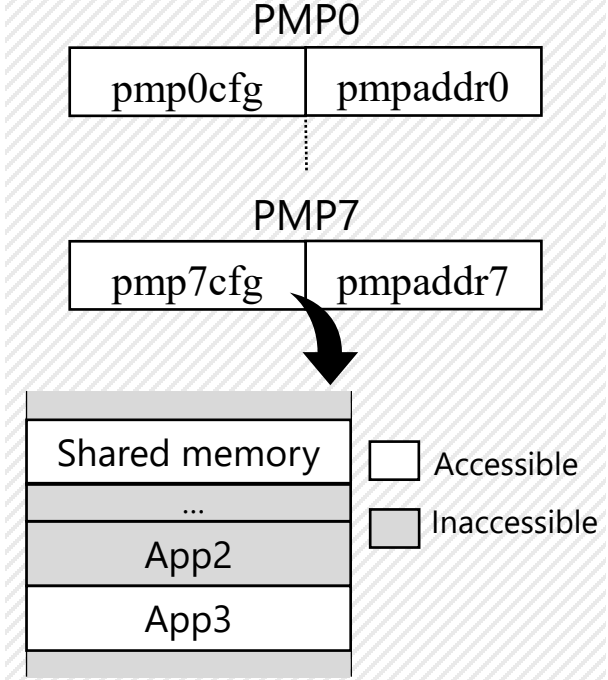
① Memory assignment

Fix app allocation area and protect it with PMP



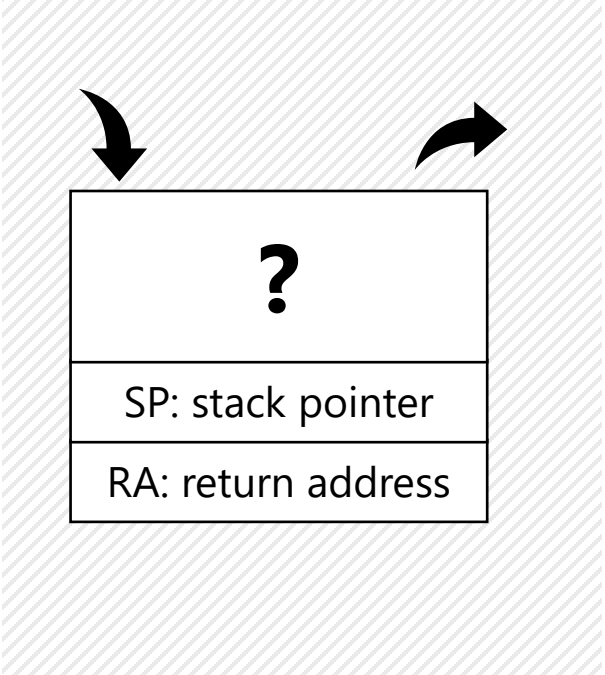
② PMP usage

PMP register settings to achieve a certain isolation



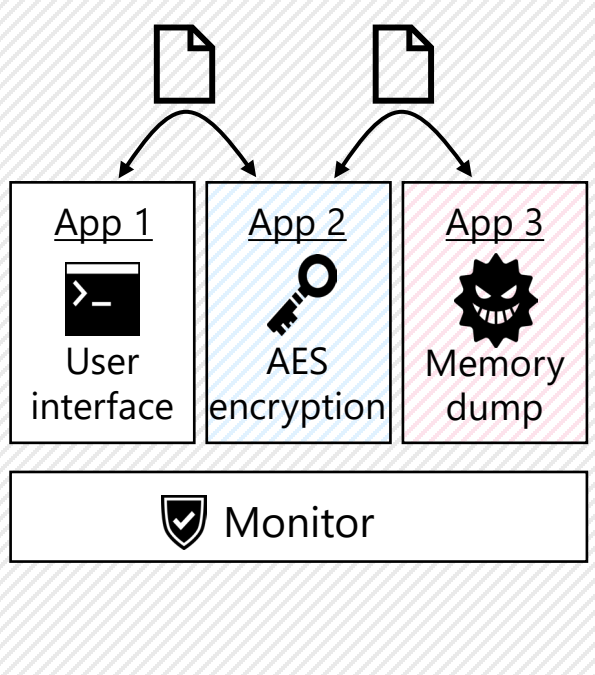
③ Context switch

Context data to include and how to transfer it

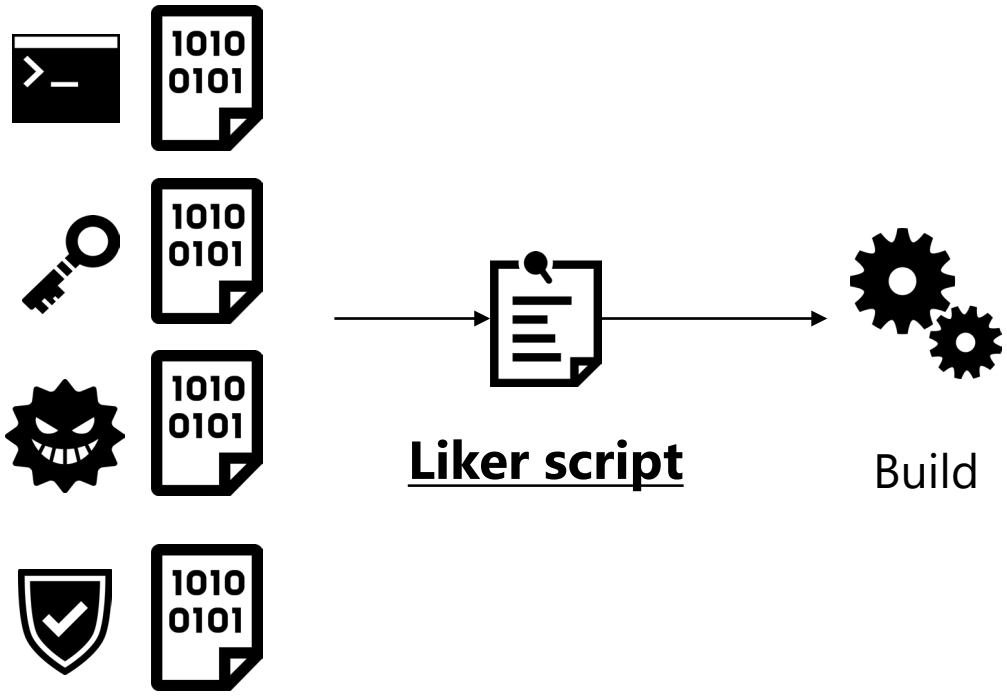


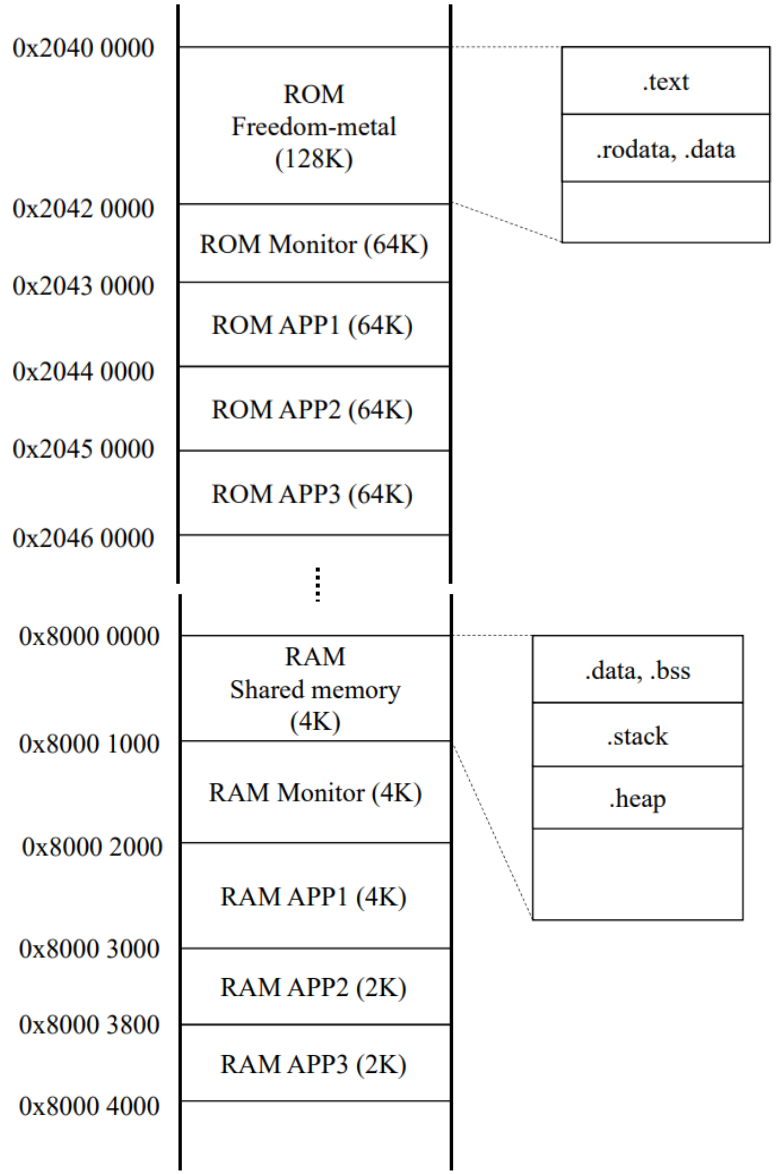
④ Interaction between *isolated* apps

Shared memory or data transfer by monitor



- Linker script specifies assigned memory





PMP	APP1	APP2	APP3
PMP0	Shared library		
PMP1	Shared memory		
PMP2	ROM APP1	ROM APP2	ROM APP3
PMP3	RAM APP1	RAM APP2	RAM APP3
PMP4	UART	N/A	N/A
PMP5	N/A		
PMP6	N/A		
PMP7	N/A		

Rewriting method

- Rewrite all PMP entries
- Specify only accessible areas like arrowlist
- Peripheral access permission is added if required
- From naïve idea of PMP usage

	normal	secure	
PMP	APP1	APP2	APP3
PMP0	ROM monitor		
PMP1	RAM monitor		
PMP2	ROM APP2	ROM APP2	ROM APP2
PMP3	RAM APP2	RAM APP2	RAM APP2
PMP4	ROM APP3	ROM APP3	ROM APP3
PMP5	RAM APP3	RAM APP3	RAM APP3
PMP6	All region	Shared library	
PMP7		Shared memory	

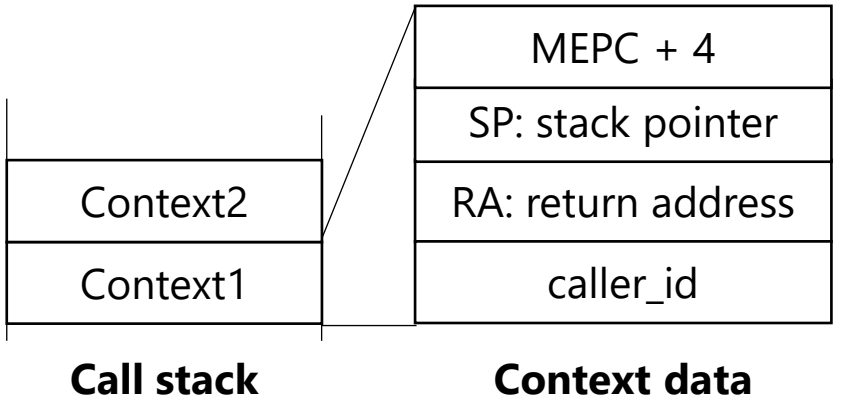
Accessible Inaccessible

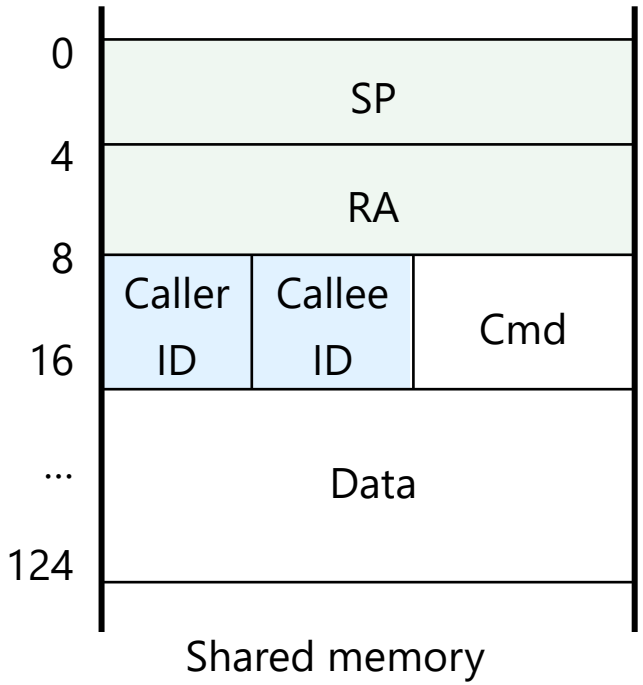
Switching method

- Switch permissions of PMP entries
- Specify inaccessible (and accessible) areas like denylist
- Only ROM and RAM
- Inspired by RISC-V Keystone

③ Context switch management

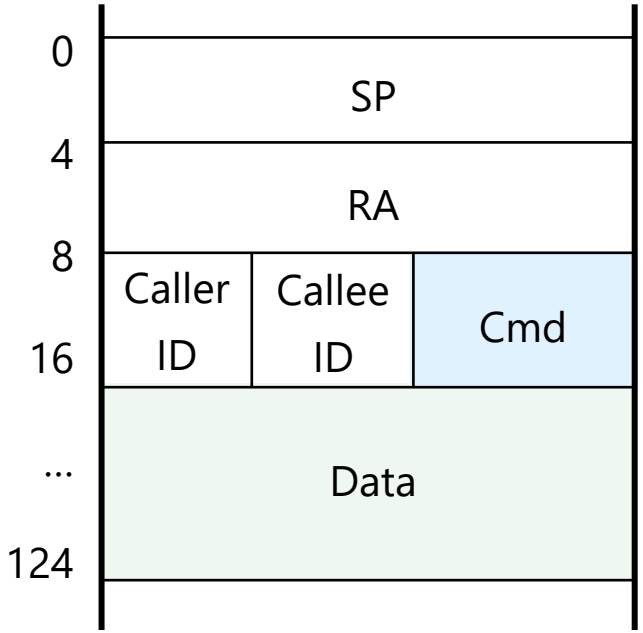
- Call monitor via `call_app()`
- Monitor receives context data from app via shared memory
  - MEPC: PC at *ecall* execution
  - Caller ID and Callee ID: Management of call stack and app running status
- Access fault exception resets saved context






```
void call_app (uint8_t caller_id, uint8_t callee_id) {
  uintptr_t sp, ra;
  uintptr_t *t;
  __asm__ volatile ("mv %0, sp" : "=r" (sp));
  __asm__ volatile ("mv %0, ra" : "=r" (ra));

  t = (uintptr_t) &shared_buffer [SHARED_SP];
  *t = sp; // [0:3]
  t = (uintptr_t) &shared_buffer [SHARED_RA];
  *t = ra; // [4:7]

  shared_buffer[SHARED_CALLER] = caller_id;
  shared_buffer[SHARED_CALLEE] = callee_id;
  __asm__ volatile ("ecall");
}
```

- Share data via **shared memory**
  - When access fault happens, monitor fills ‘Data’ area with 0xFF
- Specify application behavior with ‘Cmd’
  - e.g., set dump address or get memory dump in app 3

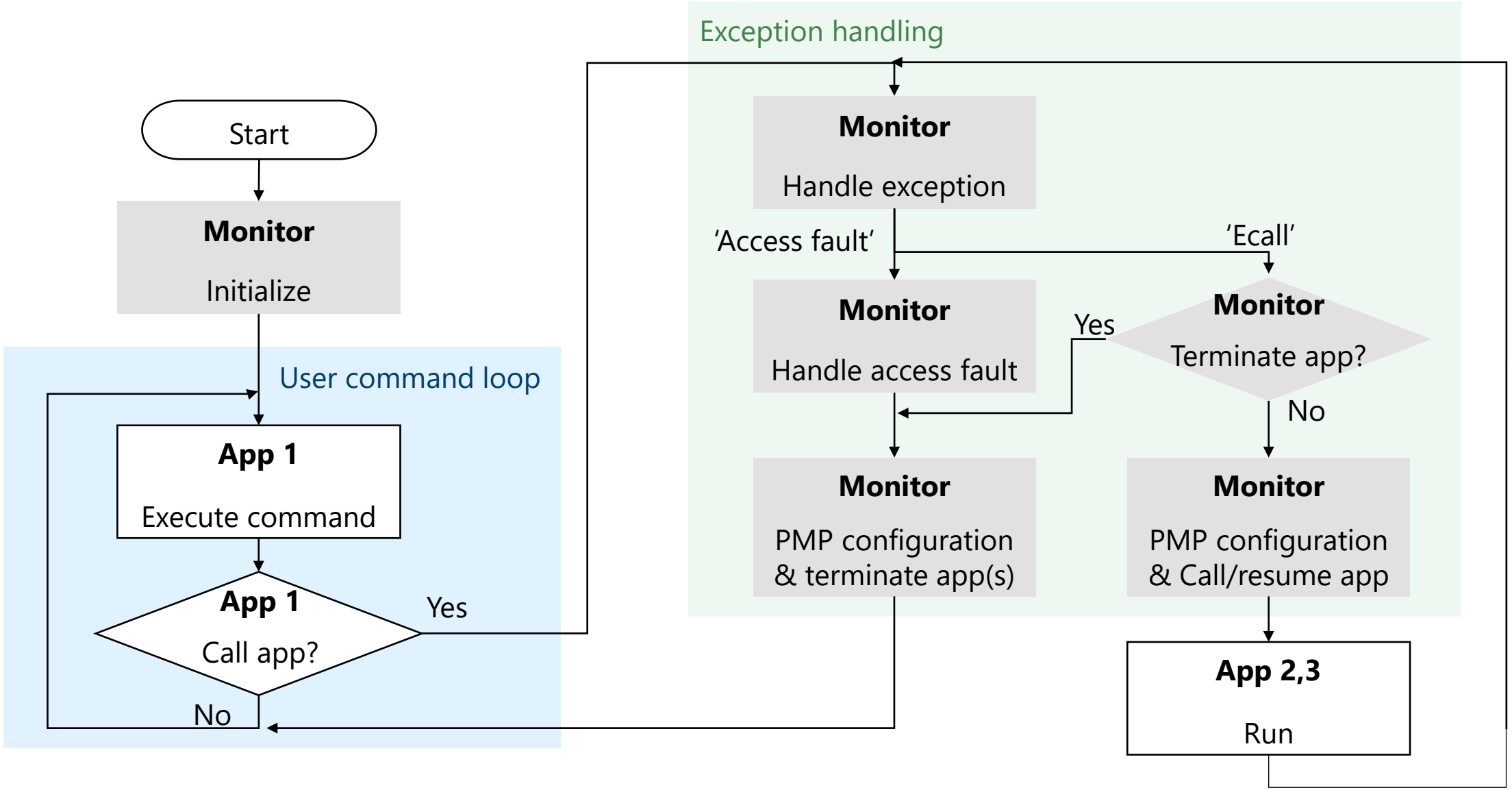


Shared memory

```
*((uint16_t *)cmd) = *((uint16_t *)&shared_buffer[SHARED_CMD]);

 switch(cmd[0]) {
 case 0x33:

   switch(cmd[1]) {
   case 0x10:
     [Set dump address]

   case 0x20:
     [Get memory dump]
   …
```

code snippet of app 3

Operational flow ("super loop" for bare metal embedded devices)



Development environment

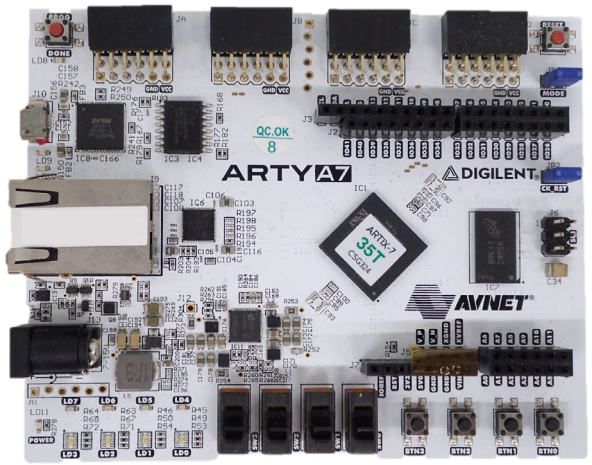
We provide in GitHub

OSS

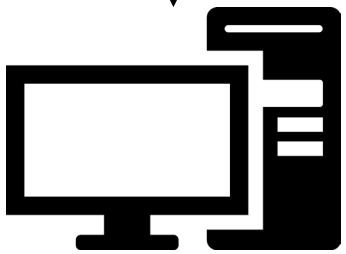
RISC-V Core

X300

(Rocket chip base)



UART



IDE: Freedom Studio v2019.08.2

Toolchain

Compiler, linker,
assembler, ...: GCC

Samples
(freedom-e-sdk)

PMP demo.

Privilege demo.

Library

Freedom-Metal

Config

Build: makefile

Link: linker script

PoC TEE

Monitor

3 apps: user I/F, encryption, mem. dump

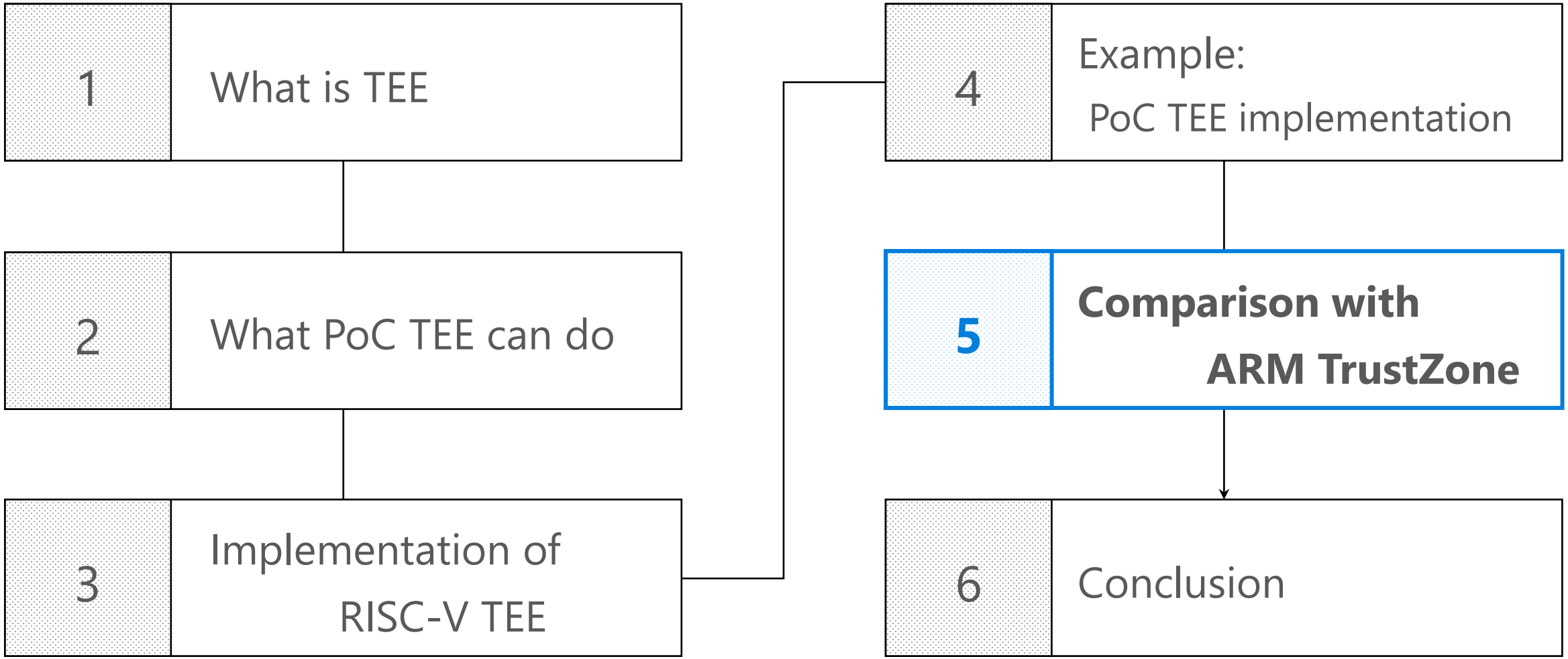
Python script

Execute 3 apps via UART

Library

micro-AES

Outline



Overview of comparison

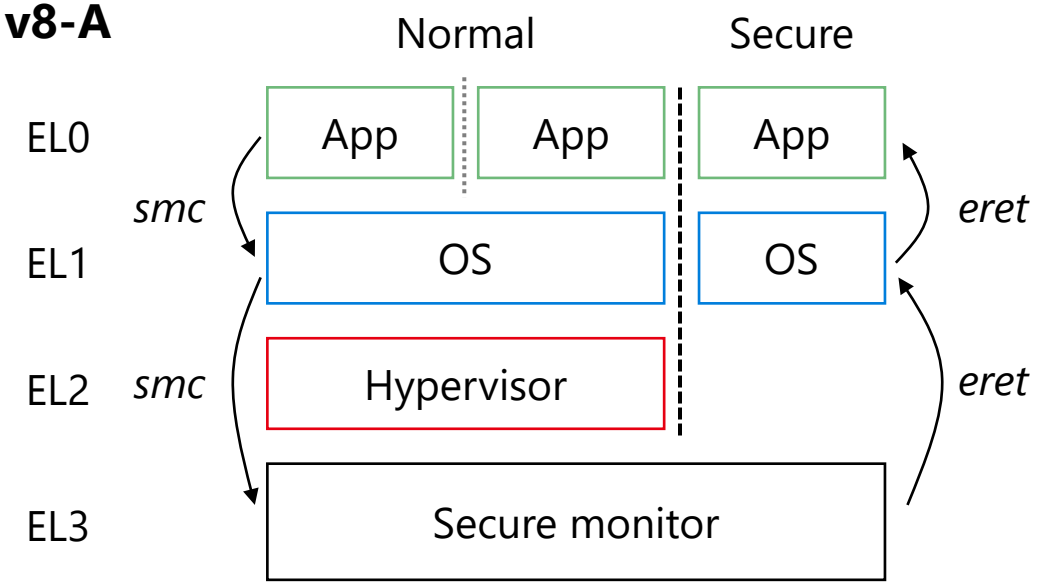
The basic structure and the isolation mechanism are similar, but there are major differences

Content	RISC-V	TrustZone v8-M	TrustZone v8-A
Hardware unit	PMP, PMA	IDAU, SAU, MPU	MMU
Privilege	M, H, S, U	Handler mode (priv.) Thread mode (priv./unpriv.)	EL3, EL2, EL1, EL0
# of world	Any (PMP entry: ～16)	2 (secure / non-secure)	2 (secure / normal)
# of application		Any (MPU: ～16)	Any
How to configure world isolation	Conf. PMP in M-mode	Conf. SAU in secure	Conf. MMU in EL3
How to configure application isolation		Conf. MPU in privileged	Conf. MMU in EL1 or EL2
World switch	*ecall:* U/S → M *mret:* M → U/S	Non-secure callable: NS → S Callback functions: S → NS	*smc*: normal → secure *eret: secure → normal*
Application switch		SVC: unpriv. → priv. Set CONTROL reg.: priv. → unpriv.	IRQ/FIQ: EL0 → EL1/2 *eret*: EL1/2 → EL0

RISC-V vs. TZ: ① Privilege

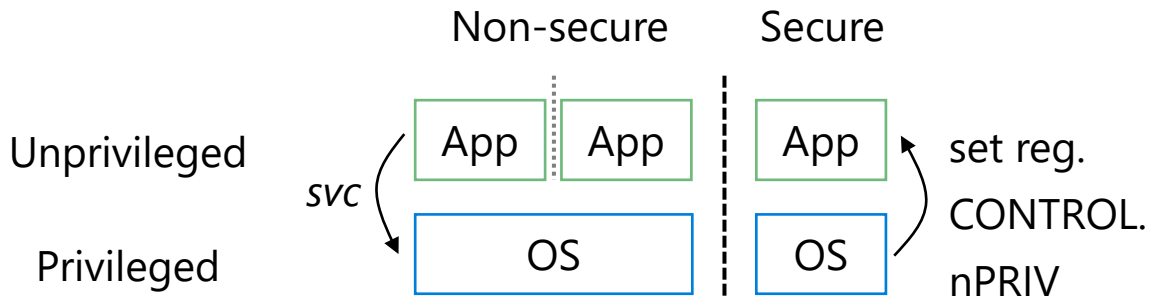
- RISC-V and ARMv8-A are similar
  - U, S, H, and M-mode & EL0, EL1, EL2, and EL3
  - *ecall* & *smc*, *mret* & *eret*
- ARMv8-M is unique
  - *ecall* & *svc*, direct register control

v8-A



**smc: secure monitor call*

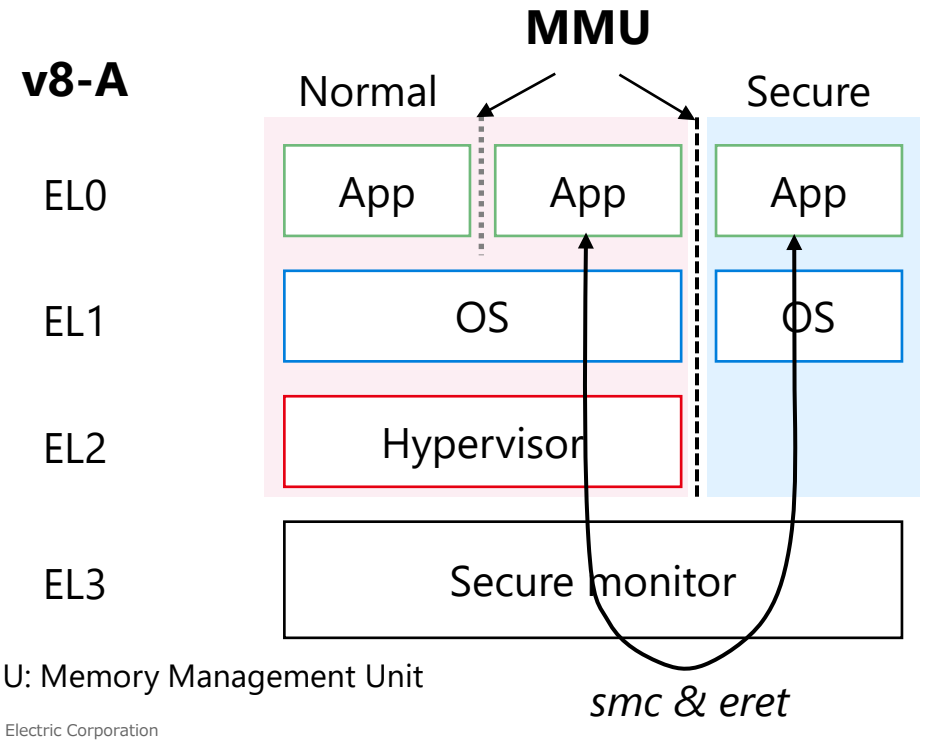
v8-M



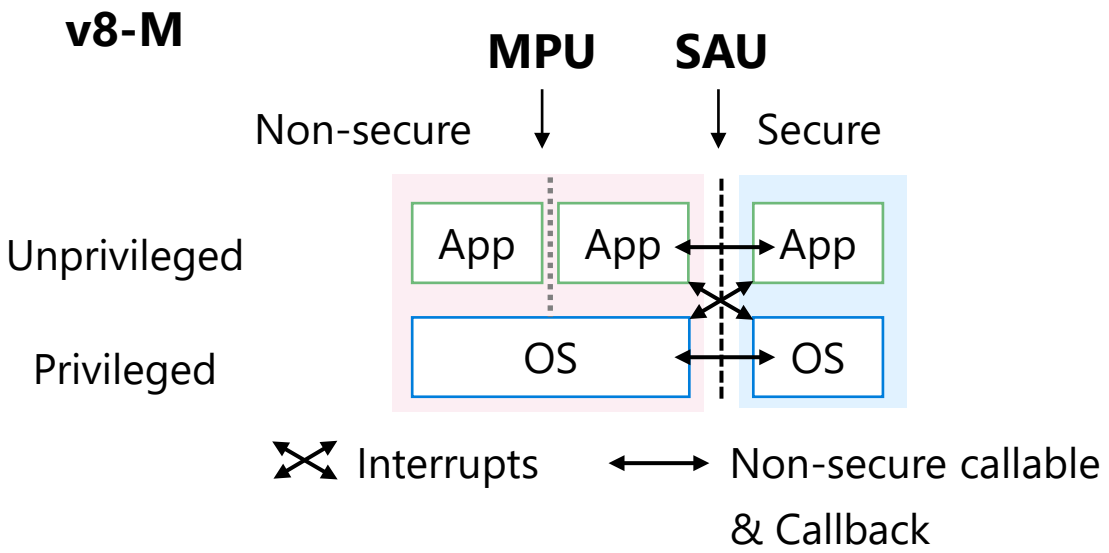
**svc: supervisor call*

RISC-V vs. TZ: ② Hardware unit and isolation

- RISC-V and ARMv8-A are similar
  - MMU & PMP, but PMP has no address translation function (RISC-V has also MMU)
- ARMv8-M is unique but somewhat like RISC-V
  - Focus on **low latency** for embedded device orientation w/o monitor
  - PMP & SAU + MPU



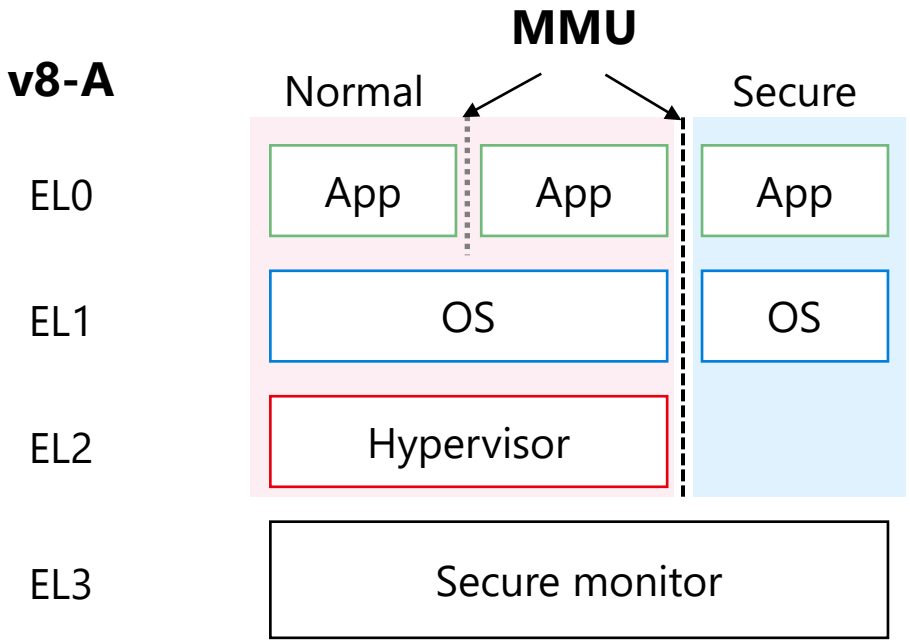
*MMU: Memory Management Unit

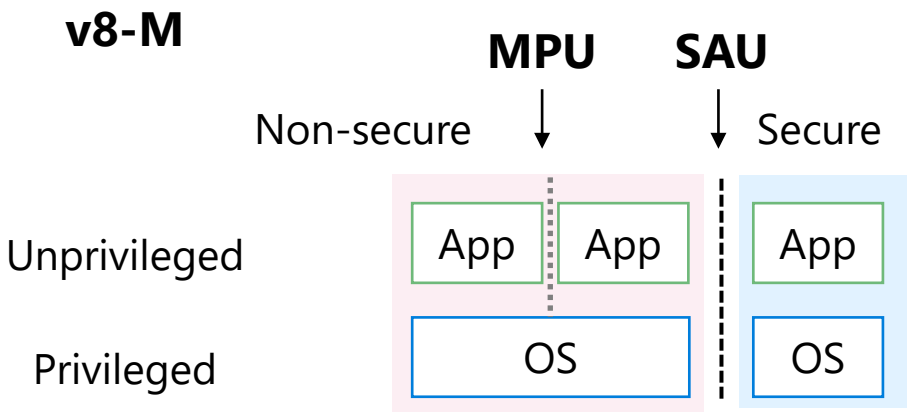


*MPU: Memory Protection Unit, SAU: Software Attribution Unit

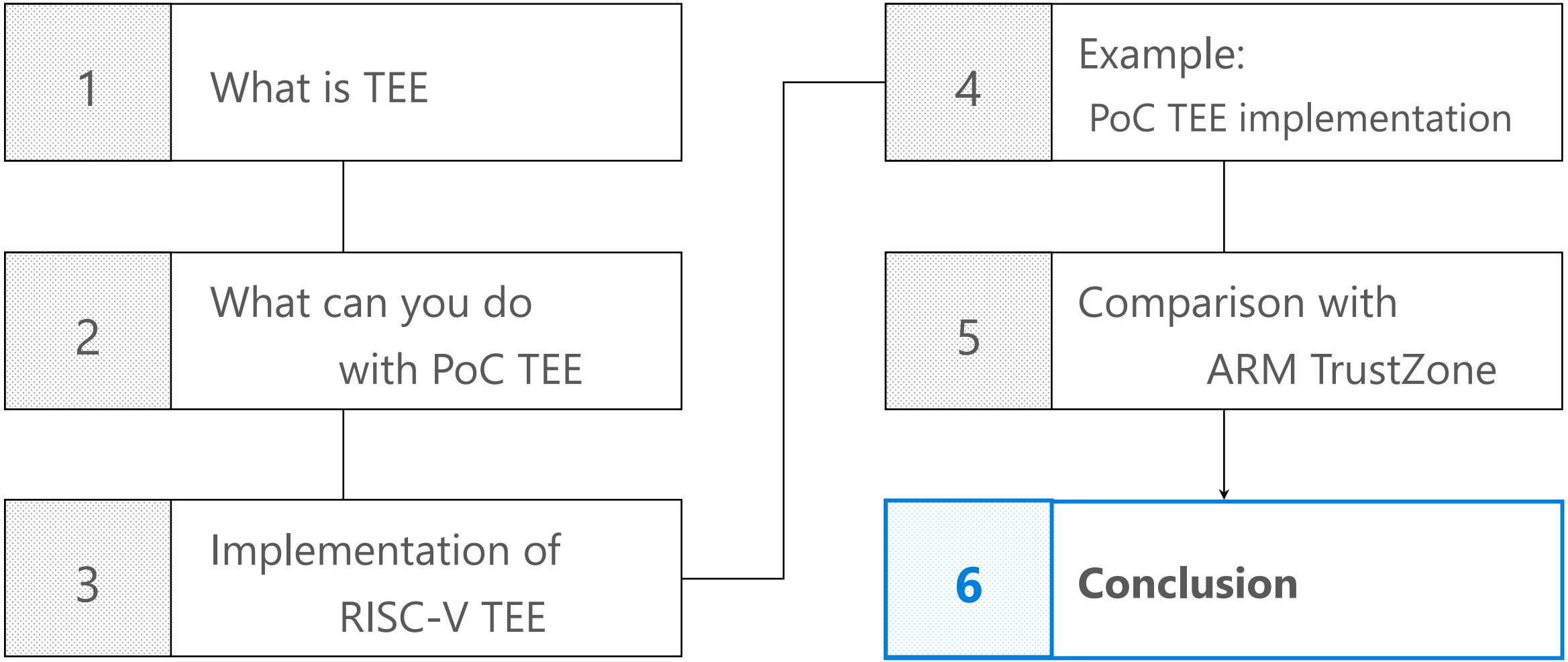
RISC-V vs. TZ: ③ Concept of world isolation

- RISC-V controls both app isolation and world isolation with PMP
- TrustZone world is dedicated area
  - Each world in ARMv8-A has its own translation table and memory access is distinguished by NS-bit
  - ARMv8-M does not change S/NS areas after initialization of SAU settings
- TrustZone app isolation is changed at context switch like RISC-V





Outline



Conclusion

- PoC TEE release
  - We developed bare metal TEE on RISC-V for embedded devices
  - Open source, running on actual device, and permissive free license
- RISC-V TEE
  - PMP and privilege mode play important roles
  - Monitor is key component to realize TEE
  - PoC TEE is an example of implementation
- Comparison with TrustZone
  - Basic structure and isolation mechanism are similar, but world concept is different

PoC TEE release

MITSUBISHI ELECTRIC
Changes for the Better

RISC-V® About RISC-V Membership RISC-V Exchange Technical News & Events Community

RISC-V Exchange

The RISC-V Exchange hosts the hardware, software, services, and learning offerings in the RISC-V community. Browse the list or search for an offering below.

SUBMIT AN ITEM
CONTACT US

Search Exchange... Hardware Cores Software Services Learning

Software

Software Type

- Accelerated Libraries
- Accelerated Libraries, Linux, macOS
- Application Infrastructure, Simulators
- Bootloaders
- BSD Distro
- C Compilers and Libraries
- C compilers and libraries, Compilers and runtimes for other languages
- Configuration
- Debugging
- Hypervisors
- IDEs and SDKs
- Kernel
- Linux distributions
- Machine Learning and AI
- Non-C Compilers and Runtimes
- Object Toolchain
- OS
- RTOS
- Security
- Simulators
- Toolchains, Simulators
- Verification Tools

PoC TEE

Organization: N/A

Proof of concept trusted execution environment implementation for embedded devices (Arty A7).

License Type: Open custom

Software Type: Security

SOFTWARE LEARN MORE

https://riscv.org/exchange/

pyth0n14n / PoC-TEE Public

Code Issues Pull requests Actions Projects Security Insights

main 1 branch 0 tags Go to file Code

S.Nashimoto and S.Nashimoto Update README 869cf24 on 16 Mar 2 commits

FreedomStudio_project	1st Commit	6 months ago
image4readme	1st Commit	6 months ago
script	1st Commit	6 months ago
.gitmodules	1st Commit	6 months ago
LICENSE	1st Commit	6 months ago
LICENSE.Apache2	1st Commit	6 months ago
LICENSE.MIT	1st Commit	6 months ago
README-ja.md	Update README	6 months ago
README.md	Update README	6 months ago
monitor_rewriting.c	1st Commit	6 months ago
monitor_switching.c	1st Commit	6 months ago

README.md

RISC-V PoC TEE: Proof-of-Concept Trusted Execution Environment

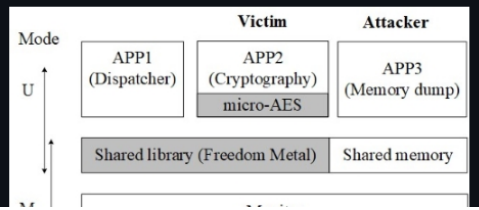
日本語版READMEはこちら

PoC TEE (Proof-ofConcept Trusted Execution Environment) is a proof-of-concept TEE implementation for RISC-V embedded devices. It runs on an FPGA board Arty A7-35t implementing Hex Five's X300 RISC-V core (URL). It is available as a Freedom Studio project.

PoC TEE was created to evaluate the resilience of RISC-V-based TEEs to fault injection attacks. See my paper at Publications & Citation for details. Please note that there are some issues with functions other than application isolation.

PoC TEE consists of the following.

- Monitor: Management PMP control and context switch management.
- APP1～3: Applications isolated by the monitor.



Thank you

Any question?

Reference (1/2)

RISC-V

- The RISC-V Instruction Set Manual Volume II: Privileged Architecture.
  https://riscv.org/specifications/privileged-isa
- David Patterson and Andrew Waterman. The RISC-V Reader: An Open Architecture Atlas. Strawberry Canyon, 2017

Use case

- DRM
  https://www.penguinsolutions.com/about-us/newsroom/protecting-premium-hd-content-widevine-digital-rights-management-drm-inforce-platforms
  https://www.bleepingcomputer.com/news/security/new-intel-chips-wont-play-blu-ray-disks-due-to-sgx-deprecation/amp/
- Nintendo Switch
  https://arstechnica.com/gaming/2018/06/inside-nintendos-perfect-method-for-detecting-online-switch-piracy/

ARM TrustZone

- ARM. ARM Cortex-A Series Programmer's Guide for ARMv8-A.
  https://developer.arm.com/documentation/den0024/a/
- ARM. Memory Protection Unit (MPU) Version 1.0.
  https://developer.arm.com/documentation/100699/0100/
- ARM. Learn the architecture: AArch64 Virtualization.
  https://developer.arm.com/documentation/102142/0100

Keystone

- Dayeol Lee. keystone-sdk.
  https://github.com/keystone-enclave/keystone-sdk/tree/master
- Dayeol Lee and David Kohlbrenner. Welcome to Keystone Enclave's Documentation!
  http://docs.keystone-enclave.org/en/latest/index.html
- Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. Keystone: An Open Framework for Architecting Trusted Execution Environments. In Proceedings of the Fifteenth European Conference on Computer Systems, pages 1–16, 2020.

Penglai Enclave

- Penglai Enclave
  https://penglai-enclave.systems/

MultiZone/HexFive

- Hex Five Security. MultiZone API.
  https://github.com/hex-five/multizone-api
- Hex Five Security. multizone-sdk.
  https://github.com/hex-five/multizone-sdk
- Hex Five Security. X300.
  https://github.com/hex-five/multizone-fpga

SiFive

- SiFive. Freedom Metal Machine Compatibility Library.
  https://github.com/sifive/freedom-metal
- SiFive. SiFive FU540-C000 Manual v1p4.
  https://sifive.cdn.prismic.io/sifive/d3ed5cd0-6e74-46b2-a12d-72b06706513e_fu540-c000-manual-v1p4.pdf

Others

- SmarterDM. micro-aes.
  https://github.com/SmarterDM/micro-aes