

CS 453/698: Software and Systems Security

Module: Research Lecture

Lecture: Research in Software and Systems Security

Adam Caulfield

University of Waterloo

Spring 2025

Reminders & Recap

Reminders:

- [Mini Research Project is due tomorrow!](#)
- Course Evaluations → “Student Course Perceptions”

Reminders & Recap

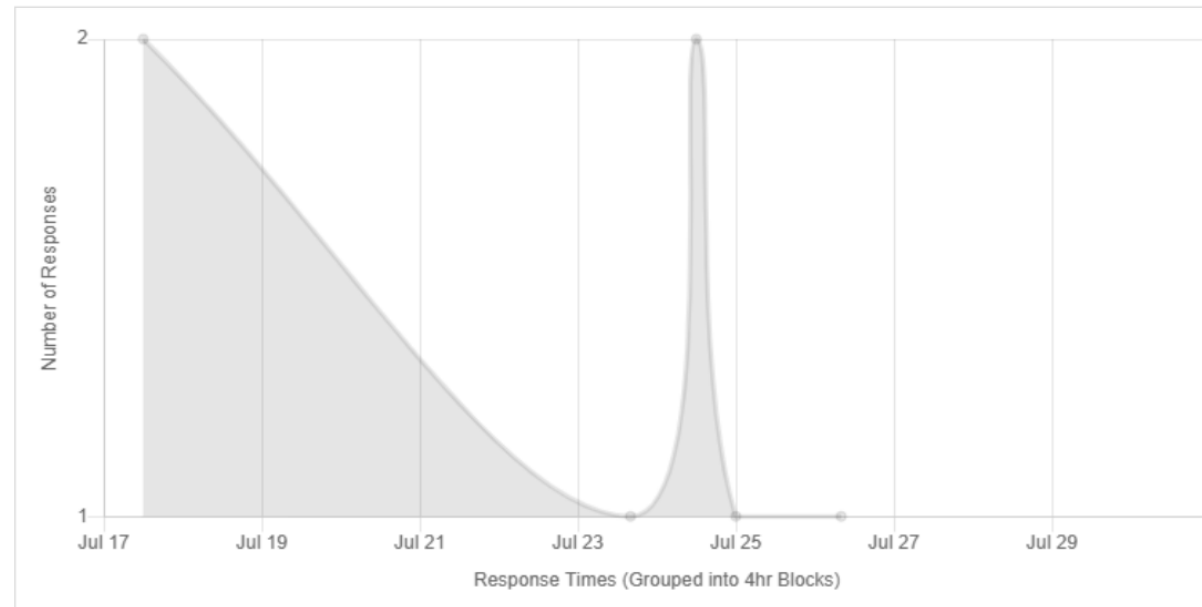
Student Course Perceptions status: make your voices heard!

Progress

Your survey is currently live. Below is the current number of registered students for this course vs. the number of surveys currently registered as completed. If your survey end-date is drawing near and your response-rate is lower than expected, it may be useful to remind your students that these online evaluations are official and important to you.

Total Students: 52 Surveys Completed: 7

If the dates listed above are wrong, or there is an issue with this evaluation, please contact your administrator. This course was configured by m2martin@uwaterloo.ca.



Note: There are two for this course (one per half), so make sure to do both!

Reminders & Recap

Reminders:

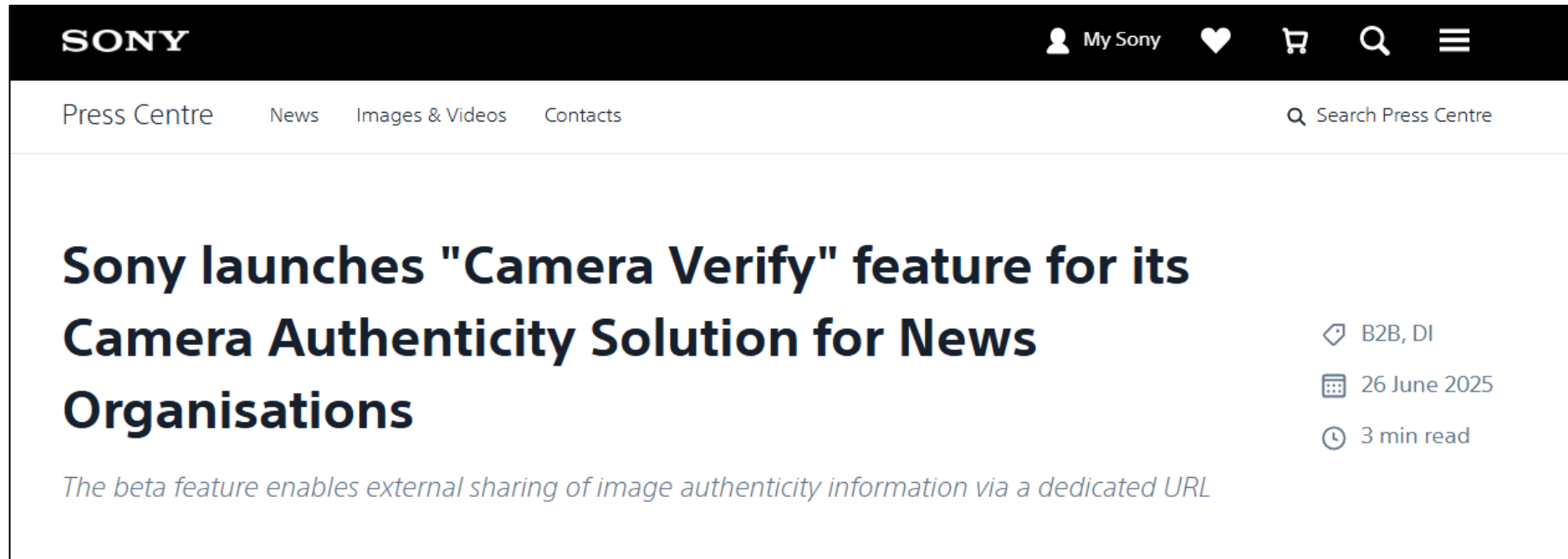
- [Mini Research Project is due tomorrow!](#)
- Course Evaluations → “Student Course Perceptions”

Recap – last time we covered:

Ethics, legal issues, laws, compliance

Follow up from last time...

SONY Press release from over the weekend



[SONY Press Centre \(UK\)](#)

Follow up from last time...

SONY Press release from over the weekend

Weybridge, June 26, 2025 – Today, Sony announced the beta release of Camera Verify, a new feature of its Camera Authenticity Solution¹, that enables external sharing of image authenticity information via a dedicated URL. This has been developed to help news organisations address the growing challenge of verifying the authenticity of digital images in the age of generative AI.

As AI-generated and manipulated content becomes increasingly sophisticated, the need for trusted, verifiable imagery has never been greater, especially for media professionals. Sony's Camera Authenticity Solution is designed to meet this need by embedding C2PA (Coalition for Content Provenance and Authenticity)² digital signatures and Sony's proprietary 3D depth information directly into the image at the moment of capture.

This solution records C2PA digital signatures and Sony's proprietary 3D depth information in the camera at the moment of capture, allowing the image's authenticity information to be verified on the Image Validation site³. With the newly added "Camera Verify" (beta), news organisations can now issue external sharing URLs for images with embedded digital signatures allowing third parties to view verification results through reliable URLs directly issued by the verification site⁴. With this feature, organisations can select specific authenticity items to share during the content publication and distribution process, enabling faster dissemination of credible, verifiable content.

Follow up from last time...

SONY Press release from over the weekend

Weybridge, June 26, 2025 – Today, Sony announced the beta release of Camera Verify, a new feature of its Camera Authenticity Solution¹, that enables external sharing of image authenticity information via a dedicated URL. This has been developed to help news organisations address the growing challenge of verifying the authenticity of digital images in the age of generative AI.

As AI-generated and manipulated content becomes increasingly sophisticated, the need for trusted, verifiable imagery has never been greater, especially for media professionals. Sony's Camera Authenticity Solution is designed to meet this need by embedding C2PA (Coalition for Content Provenance and Authenticity)² digital signatures and Sony's proprietary 3D depth information directly into the image at the moment of capture.

This solution records C2PA digital signatures and Sony's proprietary 3D depth information in the camera at the moment of capture, allowing the image's authenticity information to be verified on the Image Validation site³. With the newly added "Camera Verify" (beta), news organisations can now issue external sharing URLs for images with embedded digital signatures allowing third parties to view verification results through reliable URLs directly issued by the verification site⁴. With this feature, organisations can select specific authenticity items to share during the content publication and distribution process, enabling faster dissemination of credible, verifiable content.

Follow up from last time...

SONY Press release from over the weekend

Weybridge, June 26, 2025 – Today, Sony announced the beta release of Camera Verify, a new feature of its Camera Authenticity Solution¹, that enables external sharing of image authenticity information via a dedicated URL. This has been developed to help news organisations address the growing challenge of verifying the authenticity of digital images in the age of generative AI.

As AI-generated and manipulated content becomes increasingly sophisticated, the need for trusted, verifiable imagery has never been greater, especially for media professionals. Sony's Camera Authenticity Solution is designed to meet this need by embedding C2PA (Coalition for Content Provenance and Authenticity)² digital signatures and Sony's proprietary 3D depth information directly into the image at the moment of capture.

This solution records C2PA digital signatures and Sony's proprietary 3D depth information in the camera at the moment of capture, allowing the image's authenticity information to be verified on the Image Validation site³. With the newly added "Camera Verify" (beta), news organisations can now issue external sharing URLs for images with embedded digital signatures allowing third parties to view verification results through reliable URLs directly issued by the verification site⁴. With this feature, organisations can select specific authenticity items to share during the content publication and distribution process, enabling faster dissemination of credible, verifiable content.

Other Research in Systems and Software Security

Embedded Systems

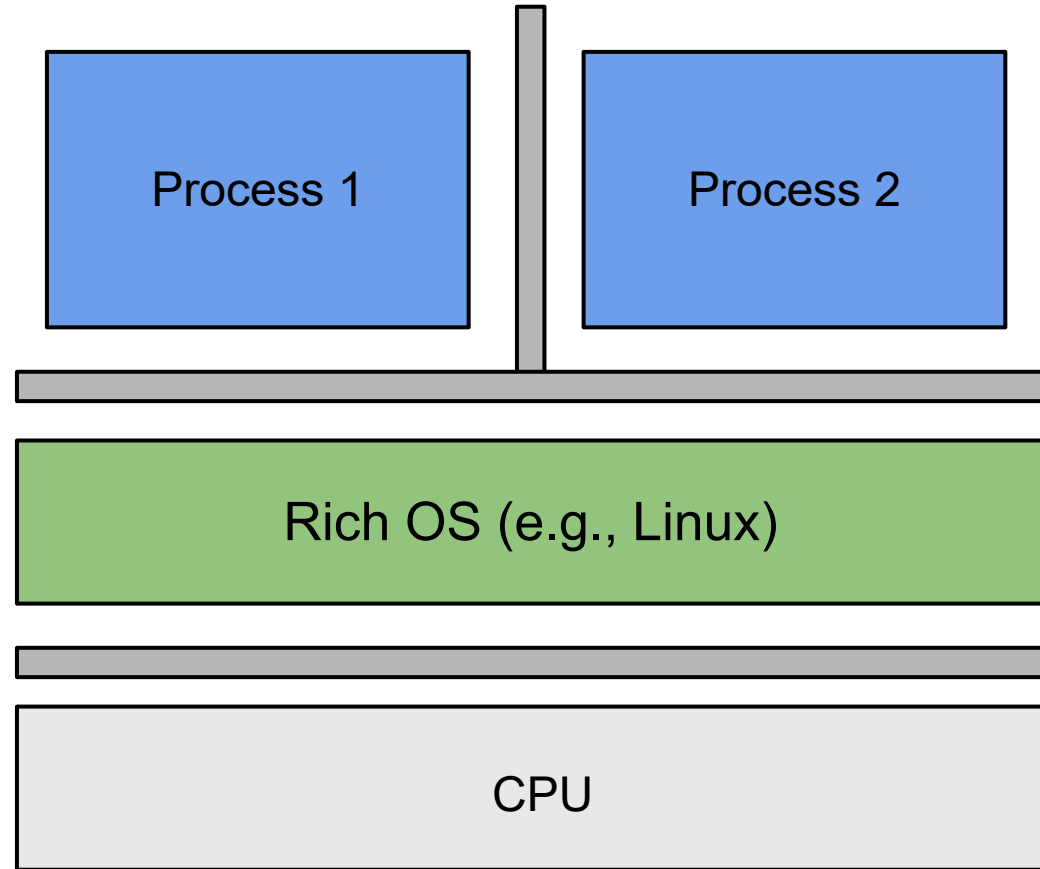
- How does the system model change?
- What type of system-level support is available in today's devices?
- Availability mechanisms
 - How to build into a system?
- Advancing attestation protocols
 - “Run-time” attestation
 - From *attestation* to *auditing*

Other Research in Systems and Software Security

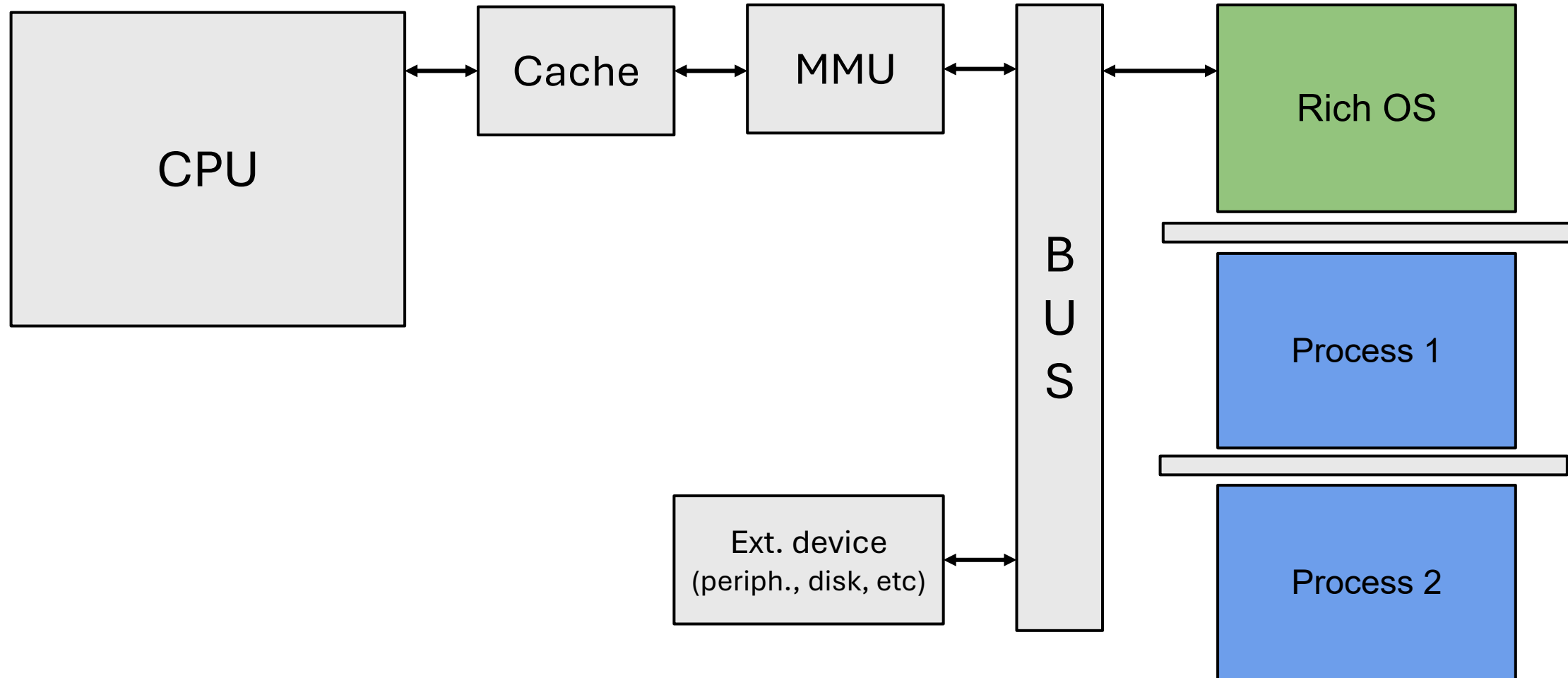
Embedded Systems

- How does the system model change?
 - **Custom Hardware Extensions in Research**
- What type of system-level support is available in today's devices?
 - **TrustZone in Cortex-M**
- Availability mechanisms
 - How to build into a system? → **GAROTA**
- Advancing attestation protocols
 - “Run-time” attestation → **C-FLAT**

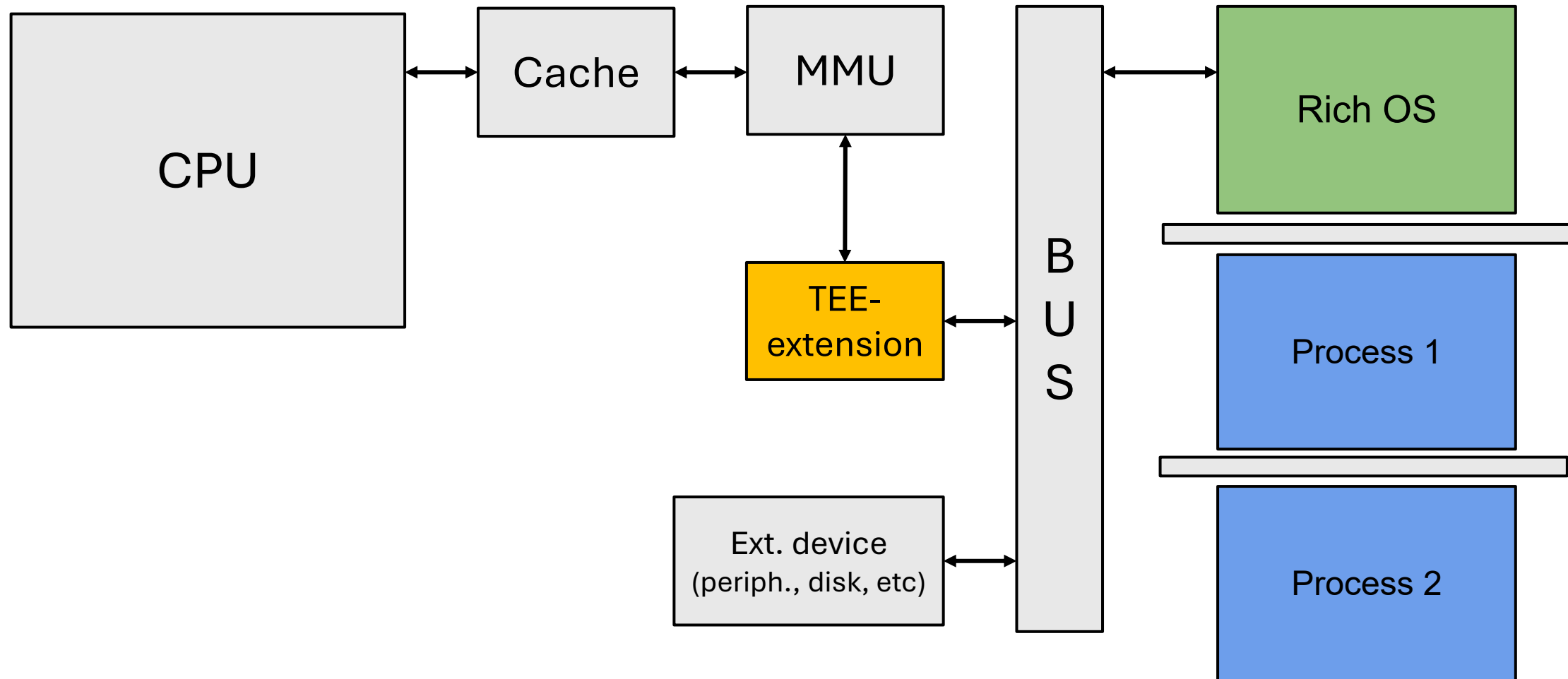
System models revisited...



System models revisited...

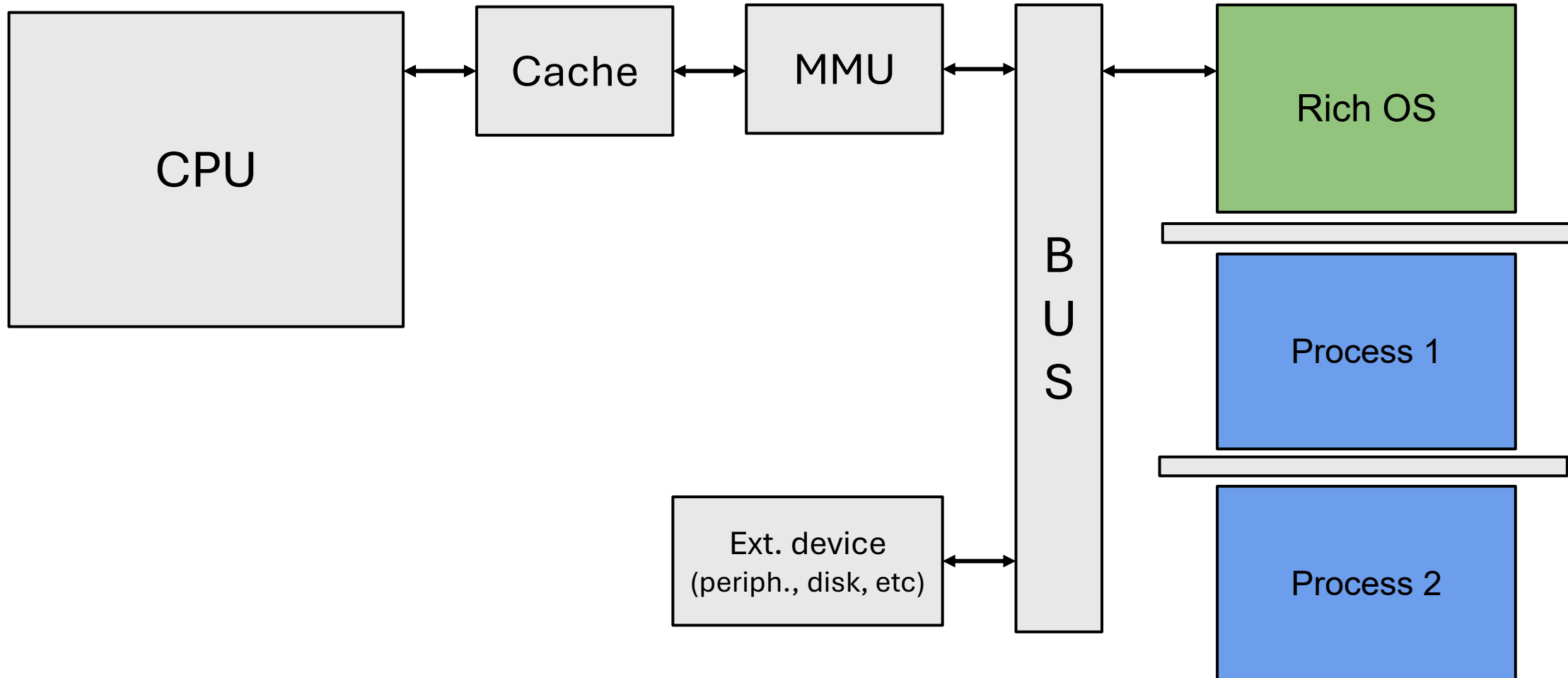


System models revisited...



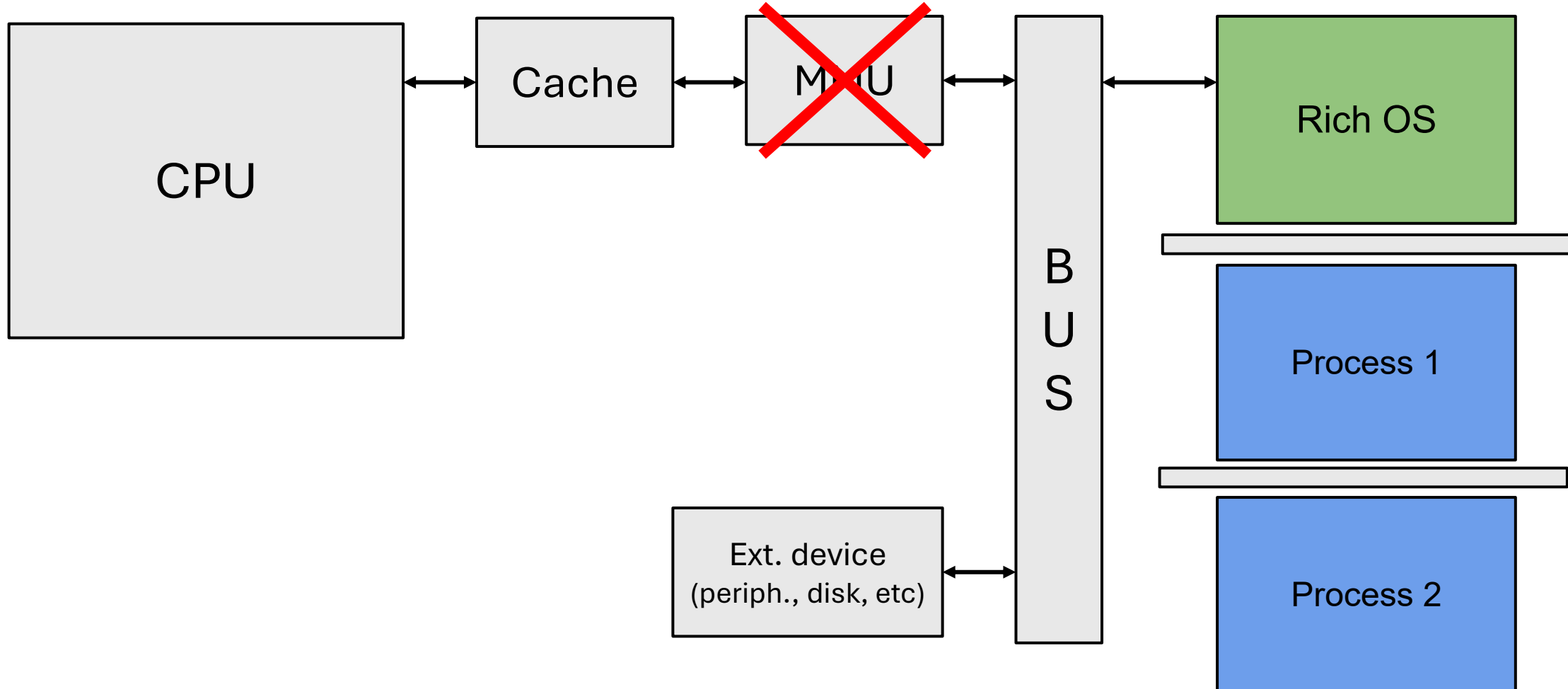
System models revisited...

What changes in the microcontroller model?



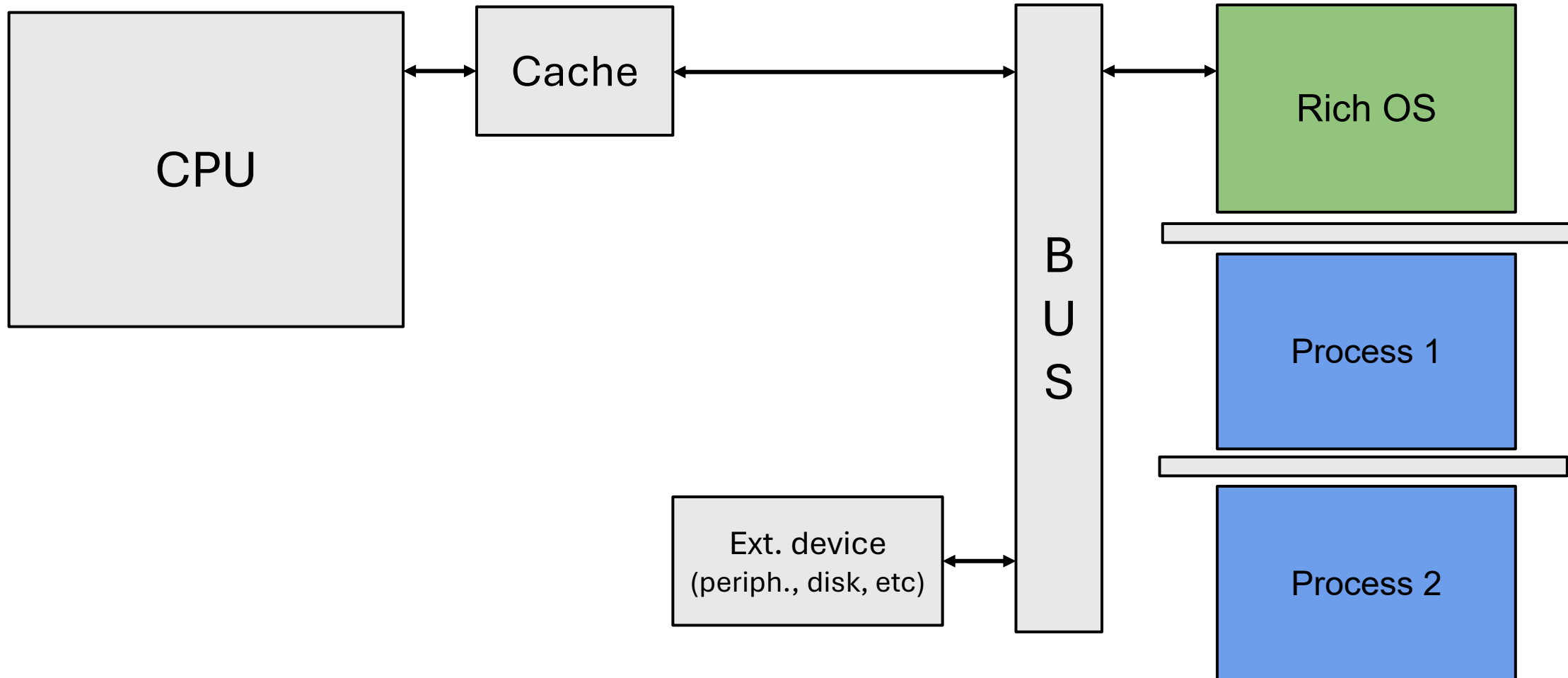
System models revisited...

What changes in the microcontroller model? **No MMUs**



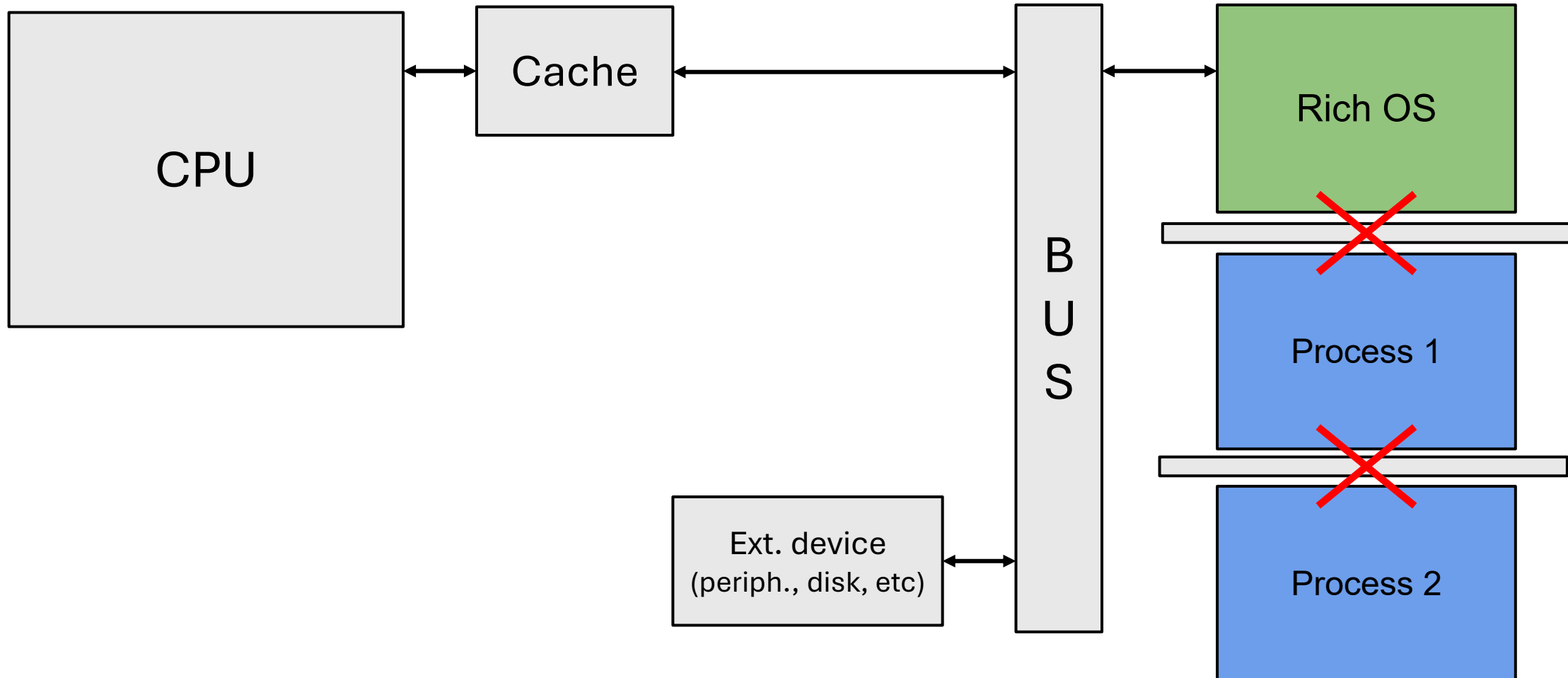
System models revisited...

What changes in the microcontroller model?



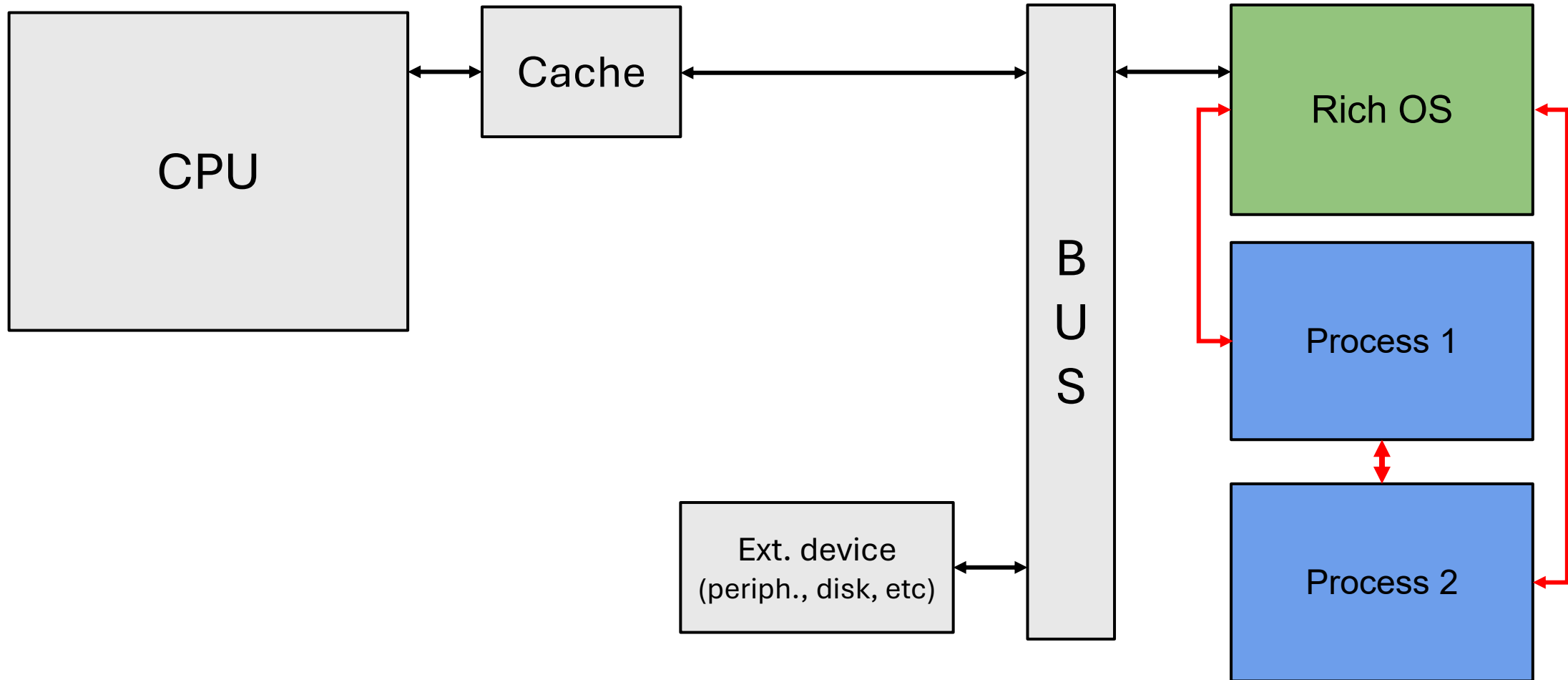
System models revisited...

What changes in the microcontroller model? **No Inter-process isolation**



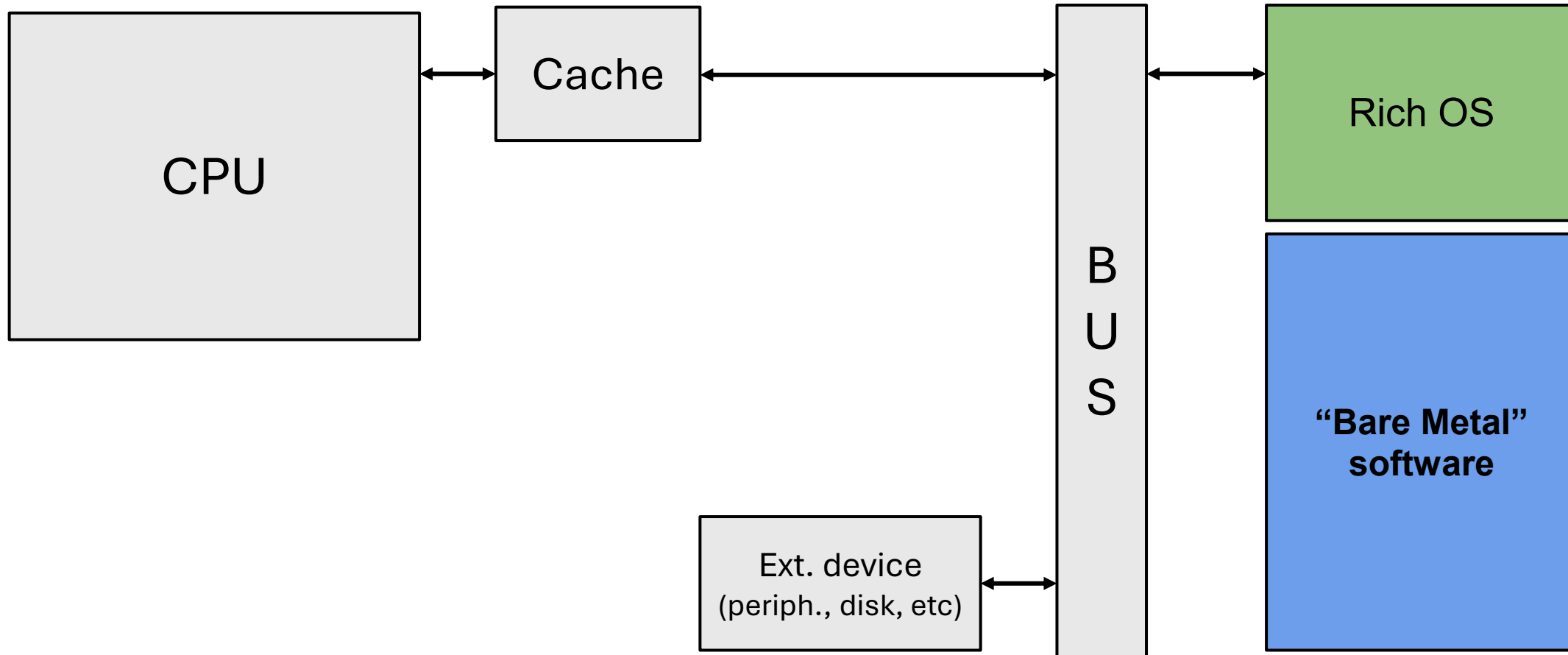
System models revisited...

What changes in the microcontroller model? **No Inter-process isolation**



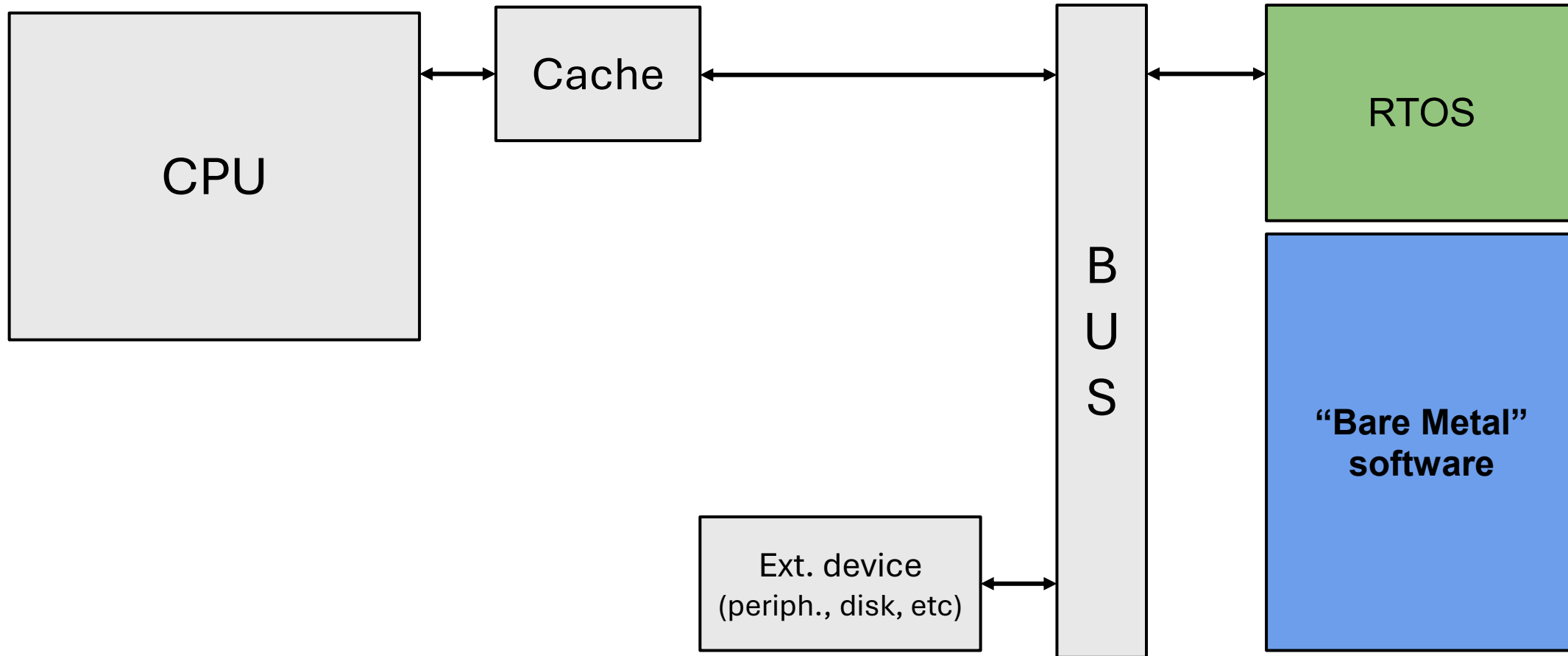
System models revisited...

What changes in the microcontroller model? **No Inter-process isolation**



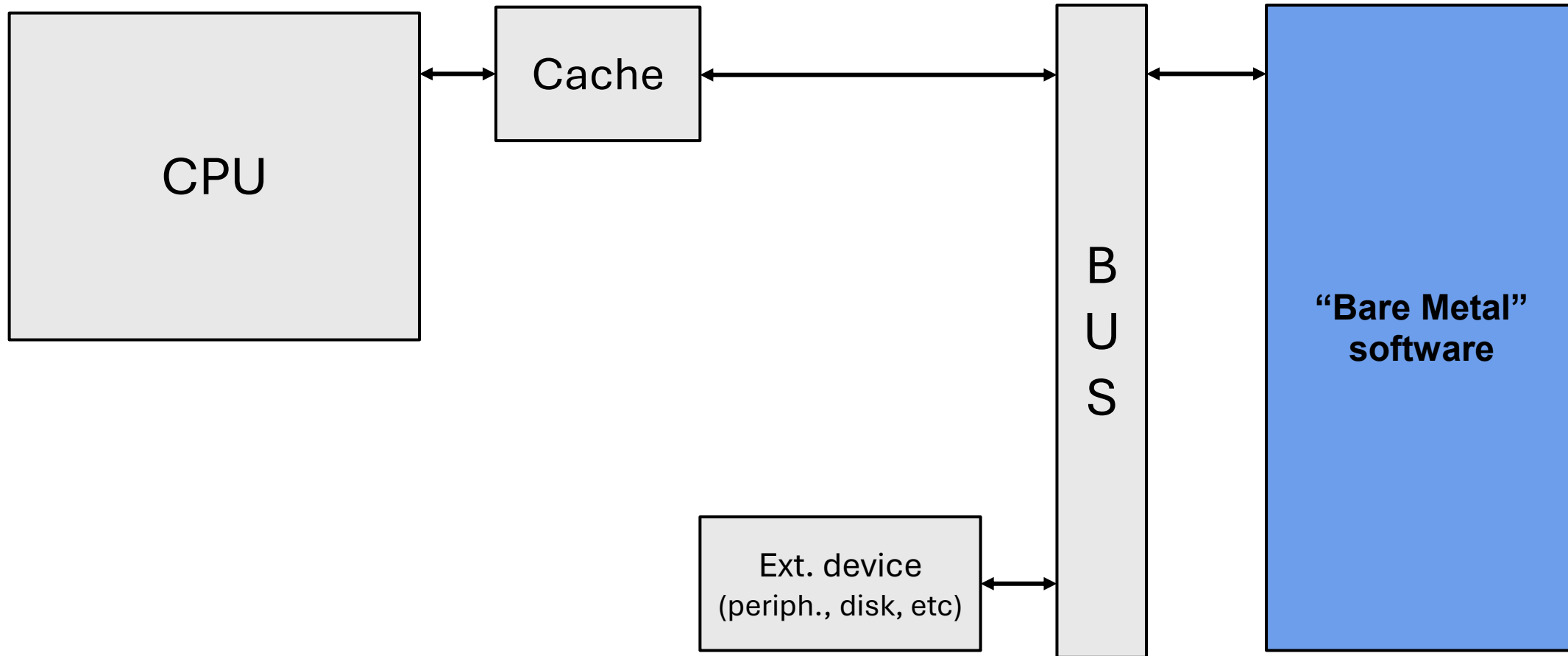
System models revisited...

What changes in the microcontroller model? **No Inter-process isolation**



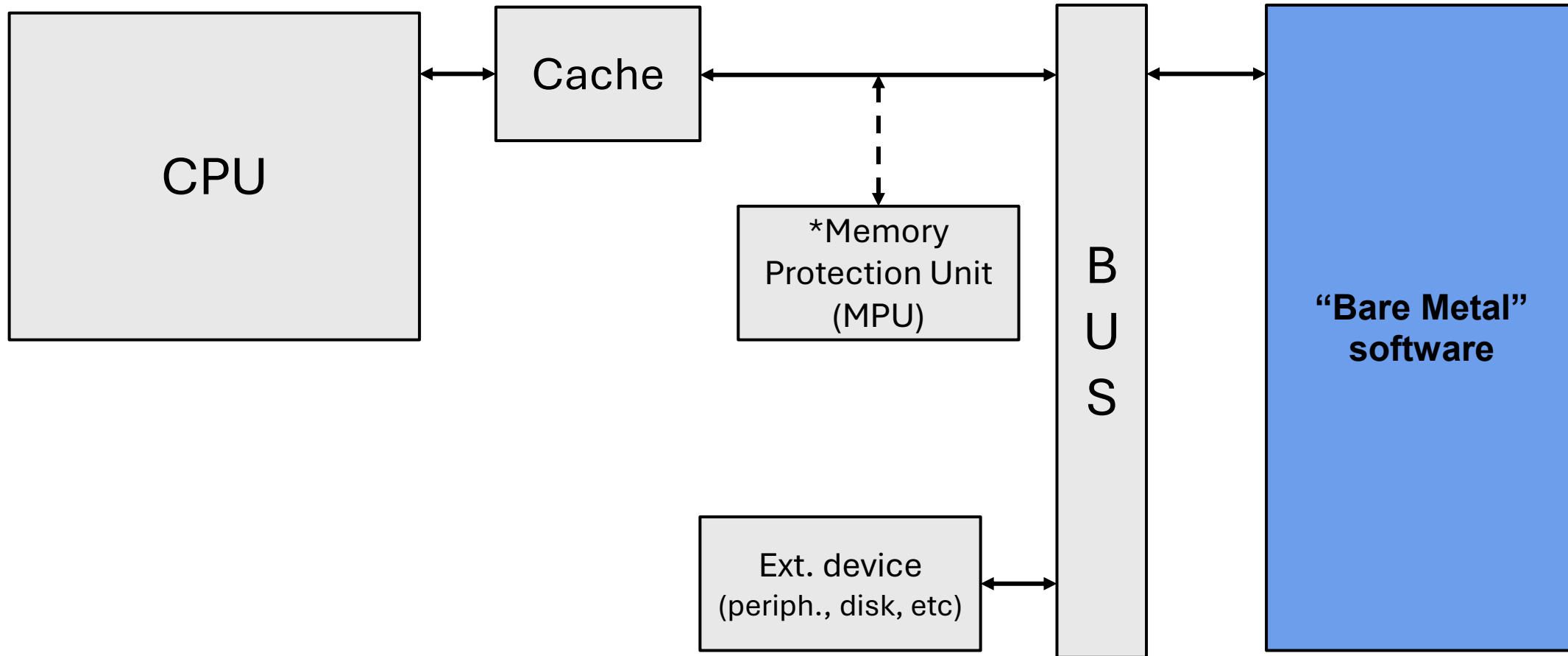
System models revisited...

What changes in the microcontroller model? **No Inter-process isolation**



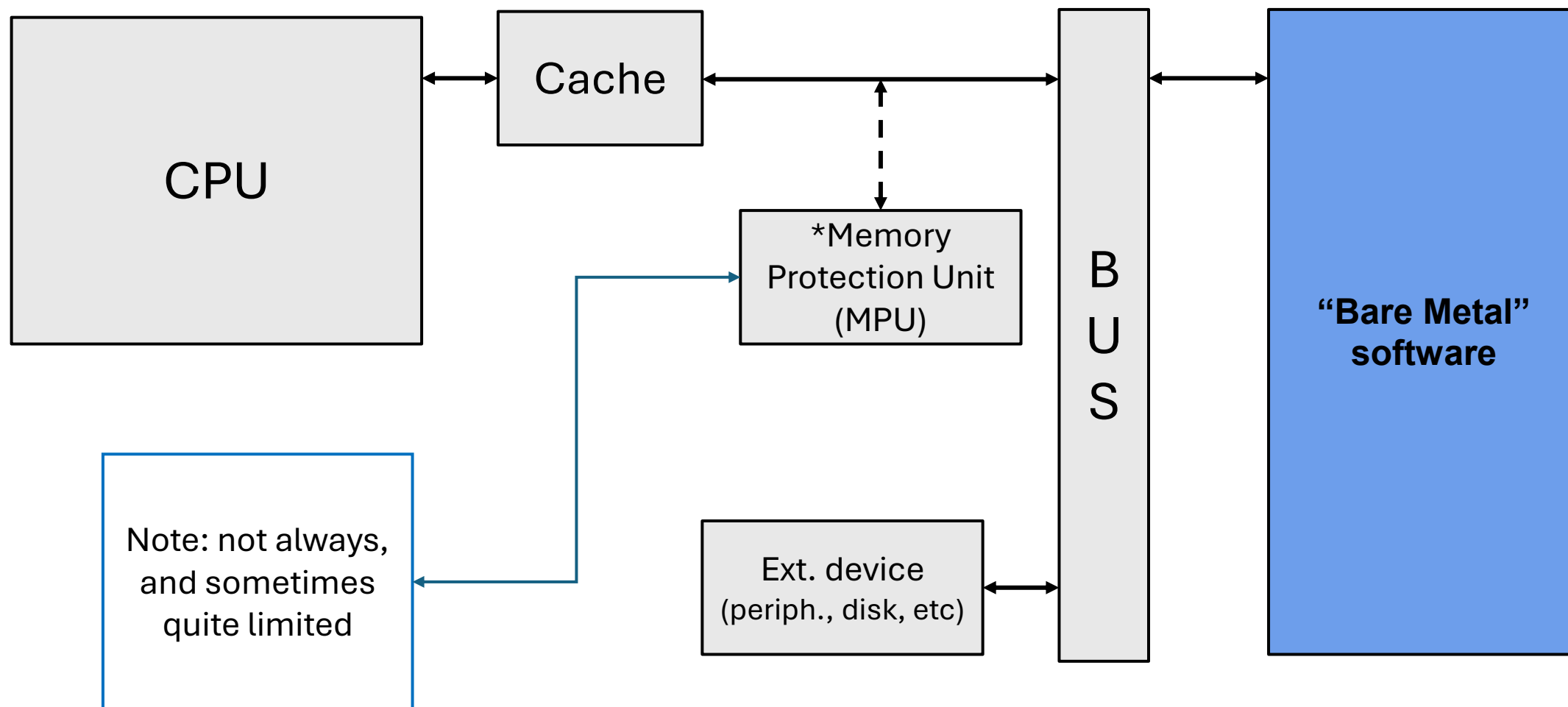
System models revisited...

What changes in the microcontroller model? **No Inter-process isolation**



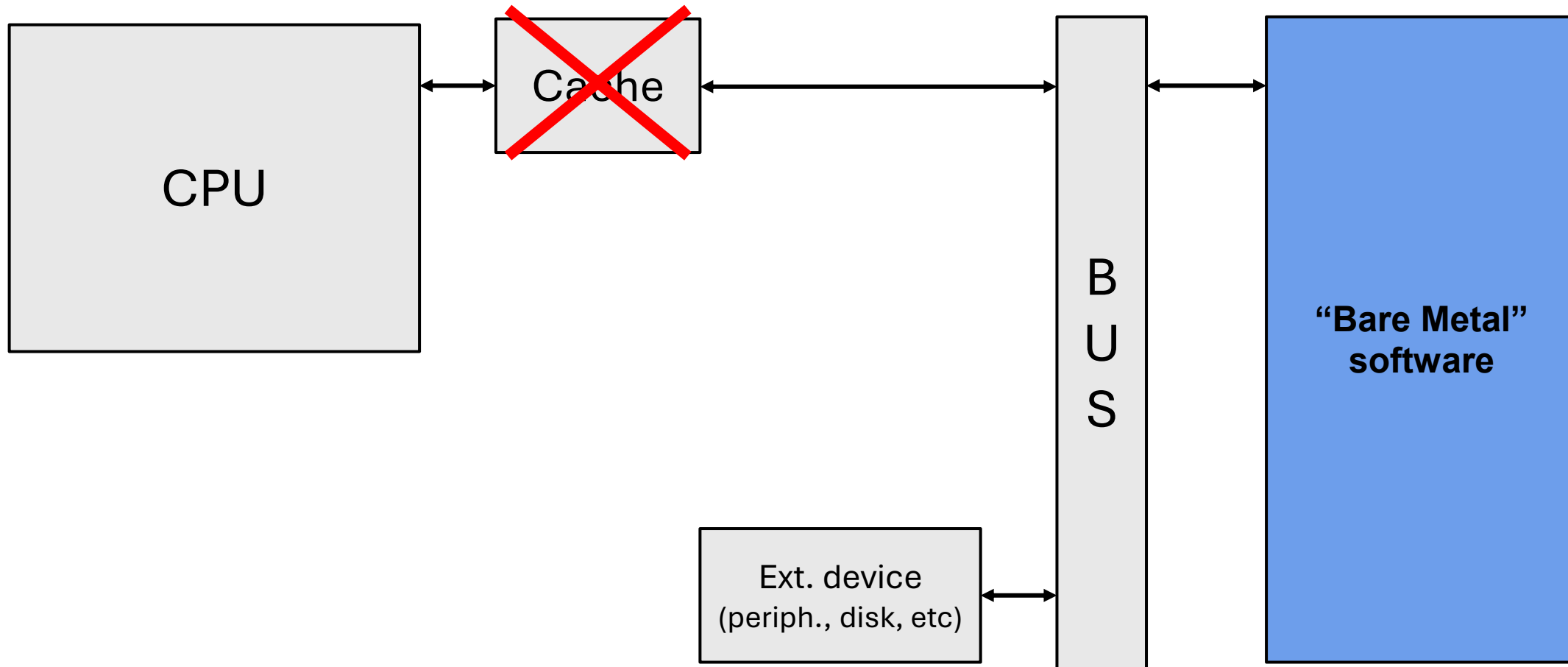
System models revisited...

What changes in the microcontroller model? **No Inter-process isolation**



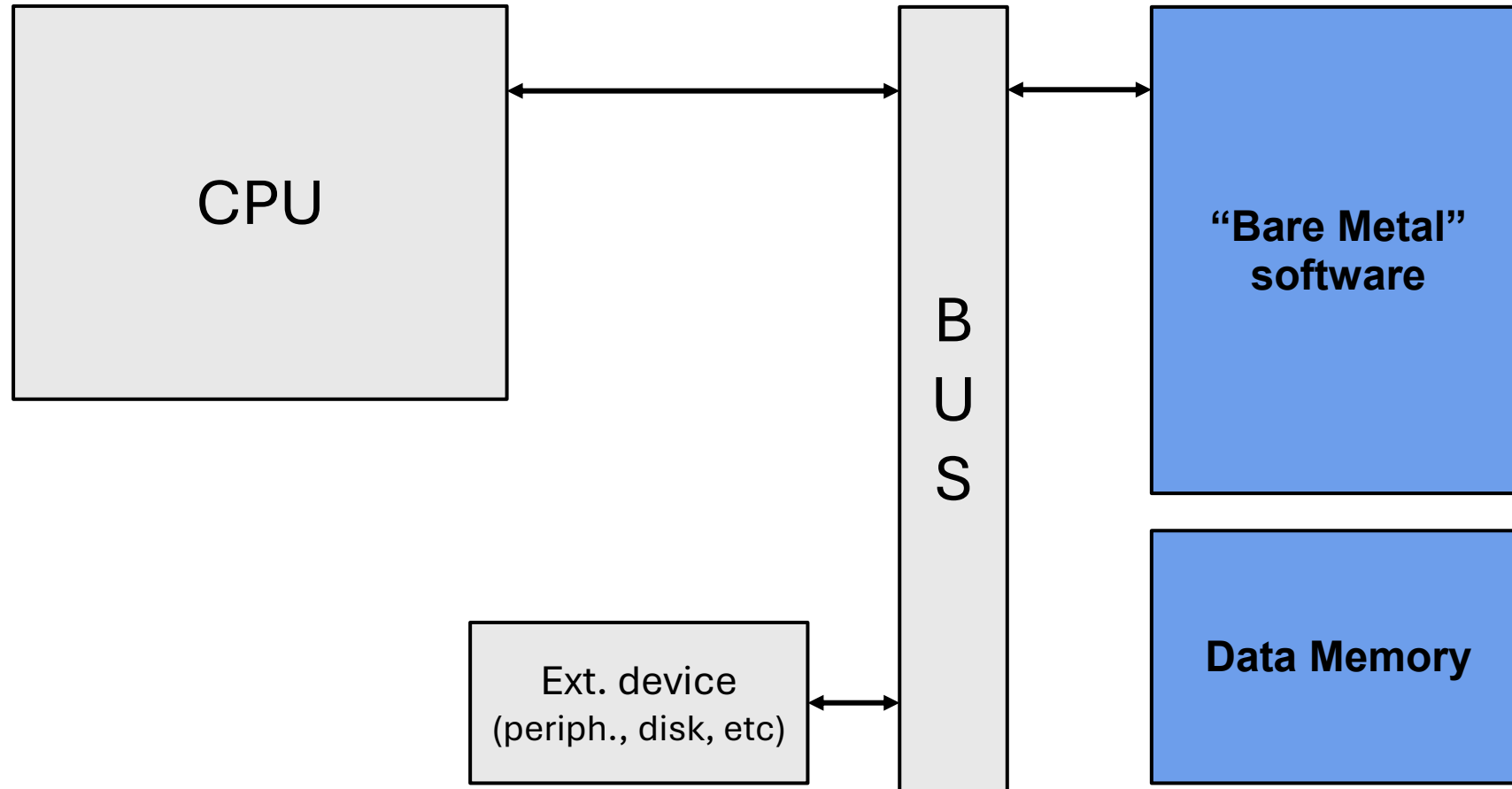
System models revisited...

What changes in the microcontroller model? **Not always having cache**



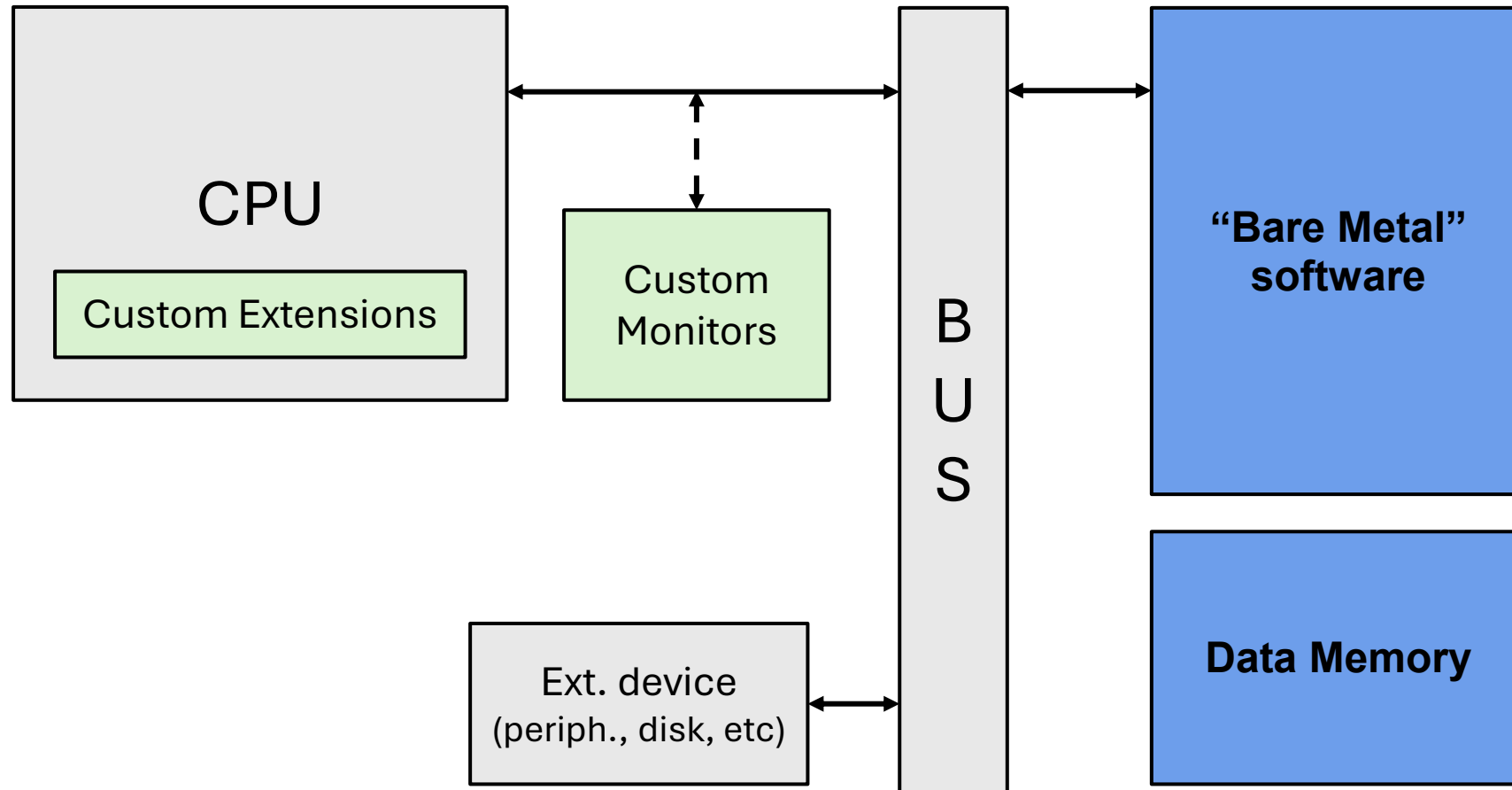
System models revisited...

Software adversary → **all memory could be accessible**



System models revisited...

Some research takes the form of developing custom hardware extensions or monitors (or classified as both depending on the abstraction)




RISC-V

RISC-V

- RISC : reduced instruction set computing
- V : fifth generation from UC Berkeley
- Open ISA → no licensing fees, full specification access
- Modular designs → ISA can be easily extended
- Built in support for custom extensions → (sometimes)
- Minimal cores

Examples: **PULPino** → 32-bit 4 pipeline MCU model



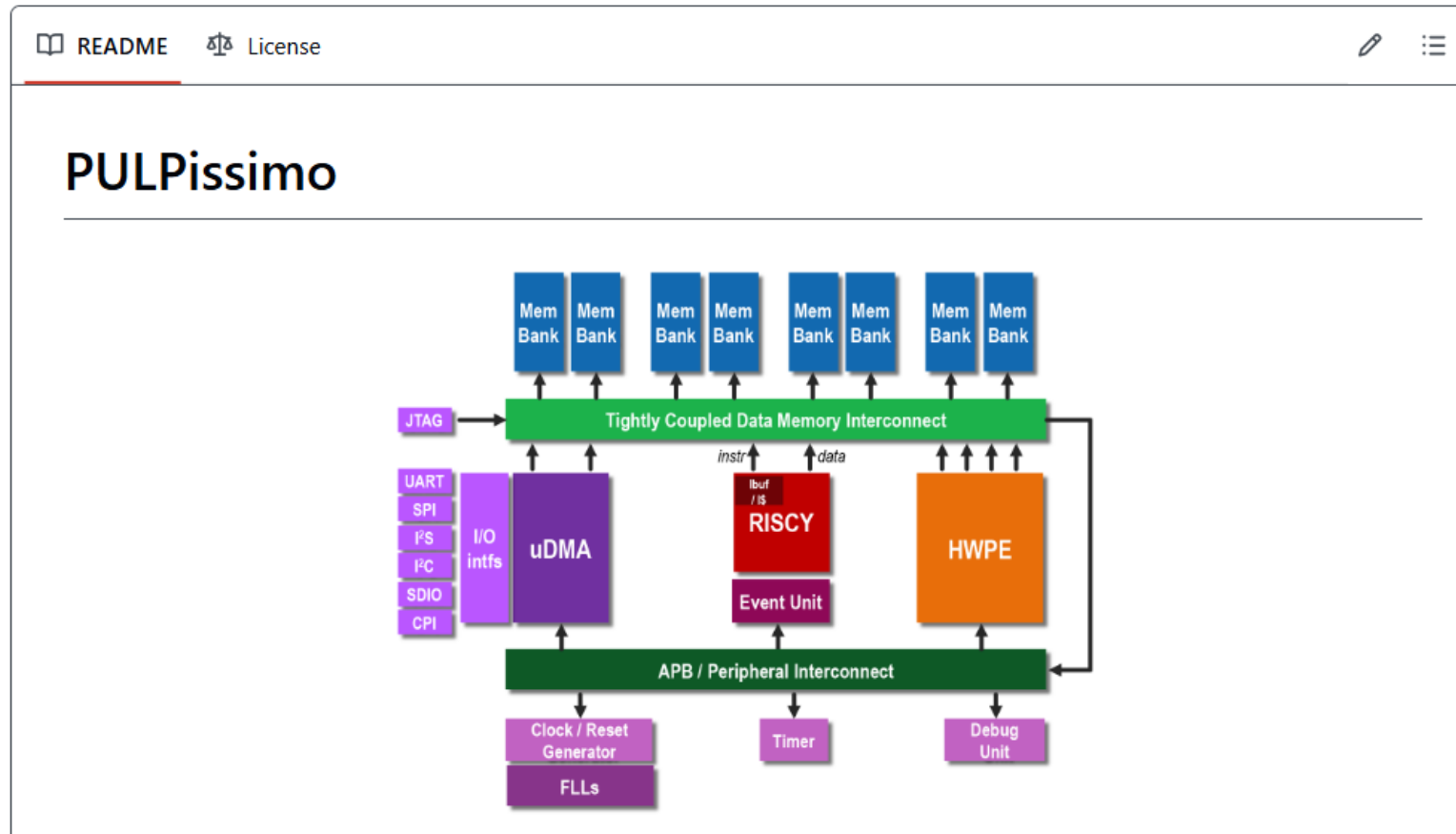
Introduction

PULPino is an open-source single-core microcontroller system, based on 32-bit RISC-V cores developed at ETH Zurich. PULPino is configurable to use either the RISCY or the zero-riscy core.

RISCY is an in-order, single-issue core with 4 pipeline stages and it has an IPC close to 1, full support for the base integer instruction set (RV32I), compressed instructions (RV32C) and multiplication instruction set extension (RV32M). It can be configured to have single-precision floating-point instruction set extension (RV32F). It implements several ISA extensions such as: hardware loops, post-incrementing load and store instructions, bit-manipulation instructions, MAC operations, support fixed-point operations, packed-SIMD instructions and the dot product. It has been designed to increase the energy efficiency of in ultra-low-power signal processing applications. RISCY implements a subset of the 1.9 privileged specification. Further informations can be found in <http://ieeexplore.ieee.org/abstract/document/7864441/>.

RISC-V

Examples: **PULPissimo** → further support for external hardware engines



Examples: Ibex Core

[README](#) [Apache-2.0 license](#) [Security](#)

[Ibex OpenTitan configuration](#) [Nightly Regression](#)

Total Tests	1530	Tests Passing	97.1%	Functional Coverage	89.8%	Code Coverage	94.8%
-------------	------	---------------	-------	---------------------	-------	---------------	-------

Ibex RISC-V Core

Ibex is a production-quality open source 32-bit RISC-V CPU core written in SystemVerilog. The CPU core is heavily parametrizable and well suited for embedded control applications. Ibex is being extensively verified and has seen multiple tape-outs. Ibex supports the Integer (I) or Embedded (E), Integer Multiplication and Division (M), Compressed (C), and B (Bit Manipulation) extensions.

Ibex Core

debug_req_i

Instruction Memory Interface

Register File

Instruction Fetch

- PC
- ICache
- Prefetch Buffer
- Compressed Instruction Decoder

Decode and Execute

- Controller
- Decoder
- CSRs
- Execute
- ALU
- Mul/Div

Writeback

- Writeback
- PMP Check

Data Memory Interface

lowRISC

Optional feature

Ibex was initially developed as part of the [PULP platform](#) under the name "[Zero-riscy](#)", and has been contributed to [lowRISC](#) who maintains it and develops it further. It is under active development.

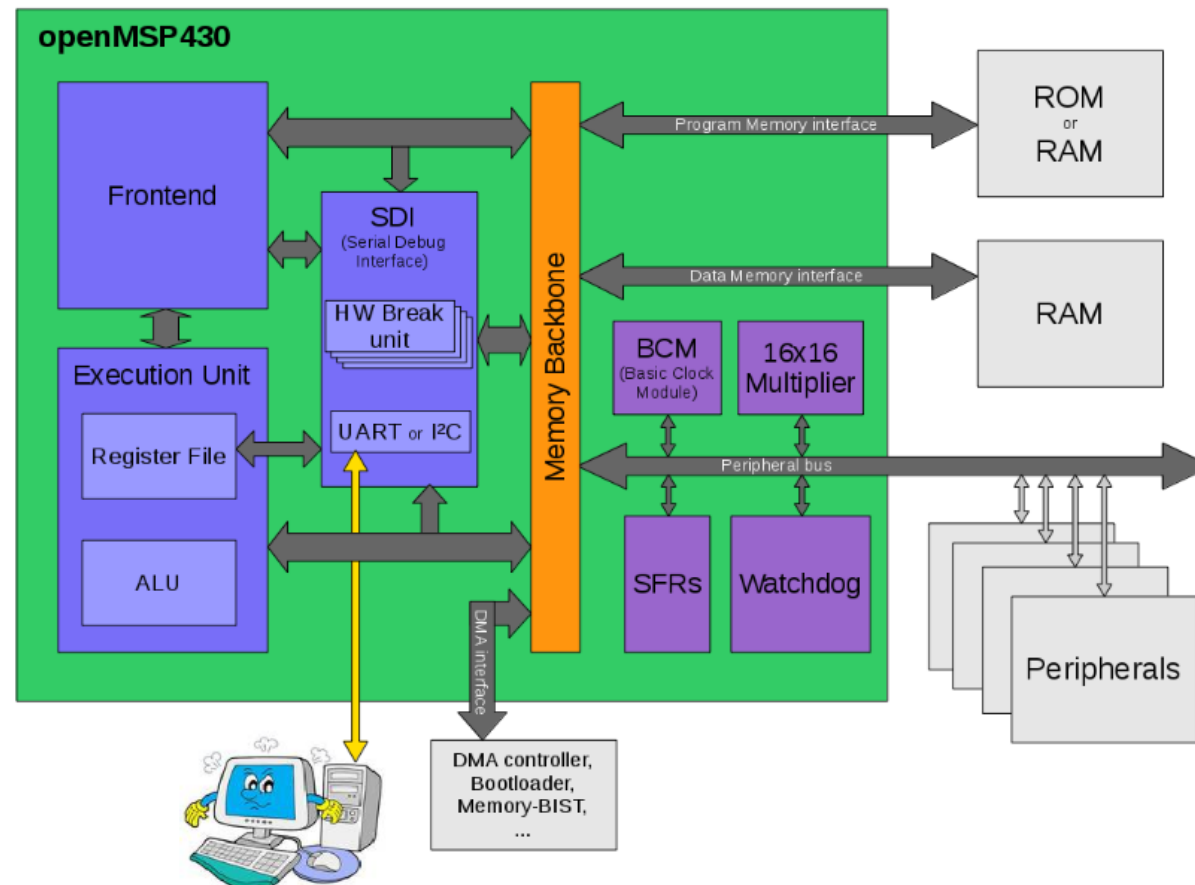
Other open cores:

Examples: openMSP430: 16-bit, 2-stage microcontroller

2. Core

2.1 Design structure

The following diagram shows the openMSP430 design structure:



Commercial extensions for MCUs

General purpose hardware controllers

Memory Protection Units

- Provide configuration for “privilege” and “unprivileged” mode
- Also r-w-x permissions on address ranges
- Some limitations on implementations

Company-specific features

- Intellectual Property Encapsulation (TI MSP430)
- **ARM TrustZone-M**

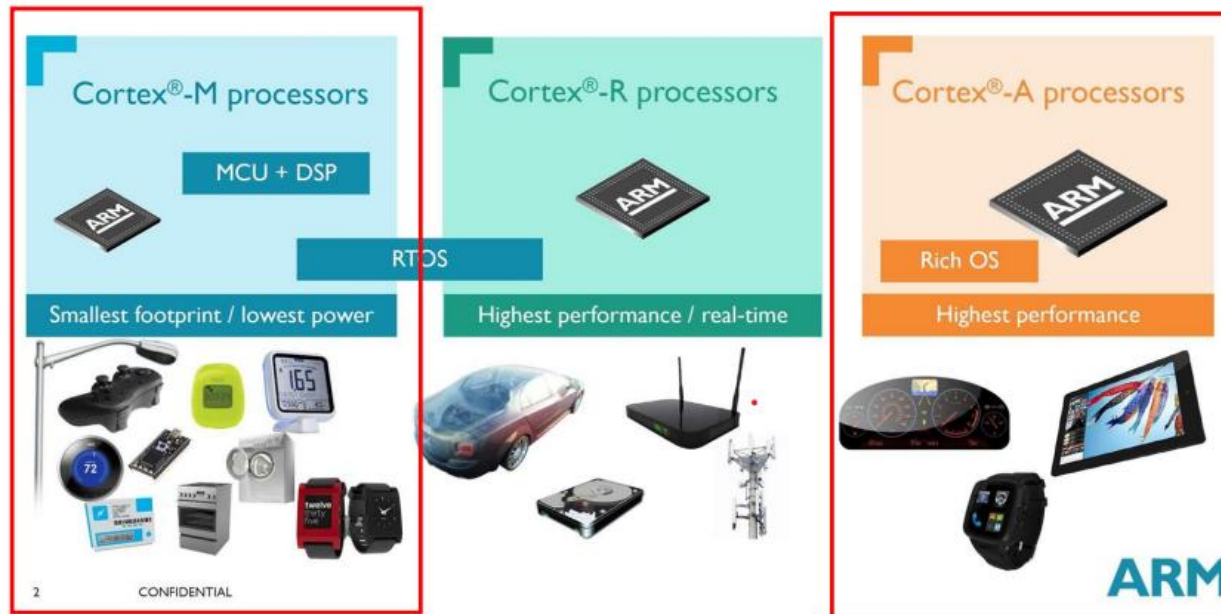
Commercial extensions for MCUs

Recall from the previous lecture...

ARM Processors

A few family of CPUs provided by ARM

ARM® Cortex® Processors across the Embedded Market



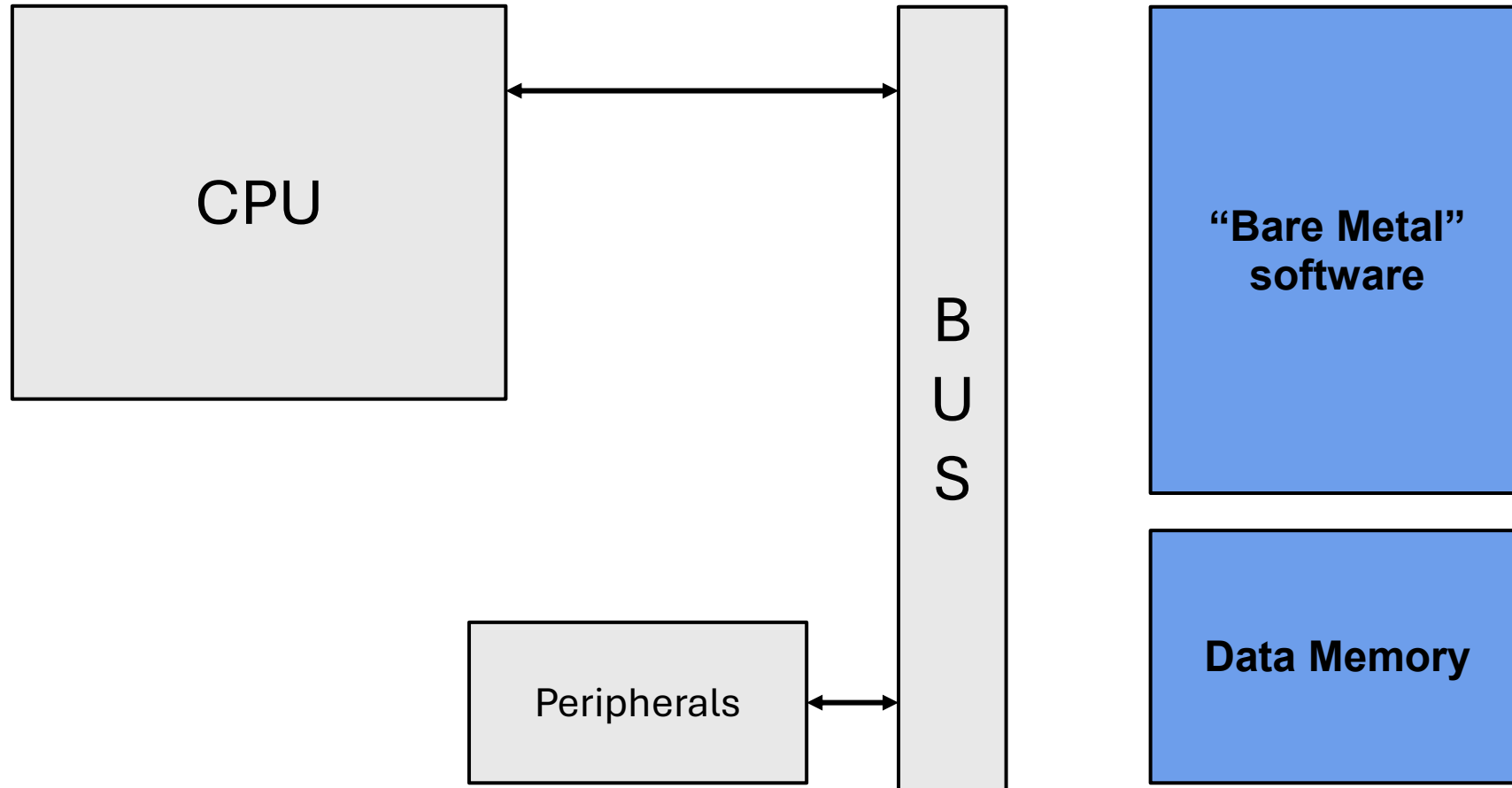
Later... (Research lecture)

Covered Today!

6

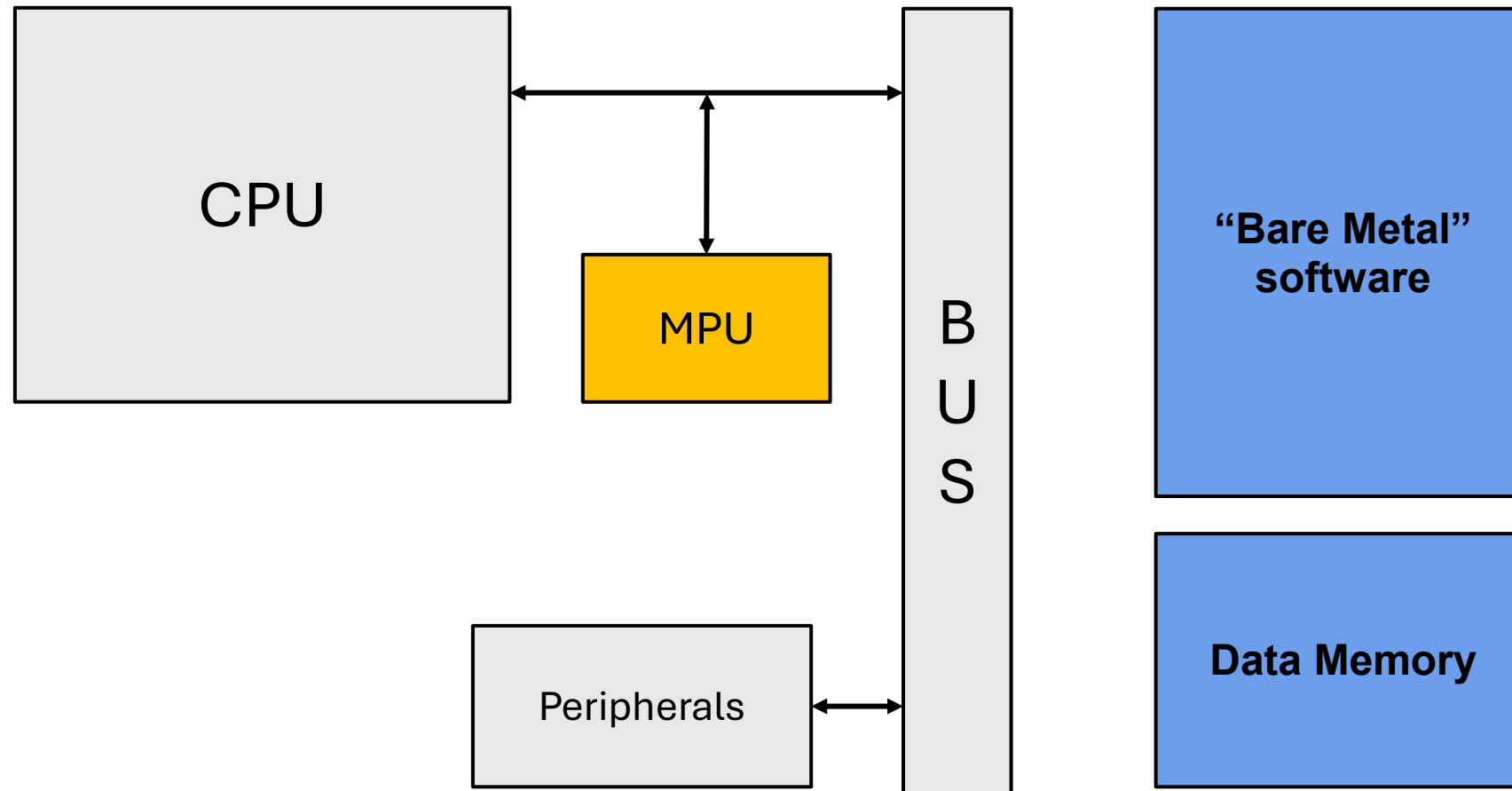
ARM Cortex-M Processors

Follow the same simple computer model



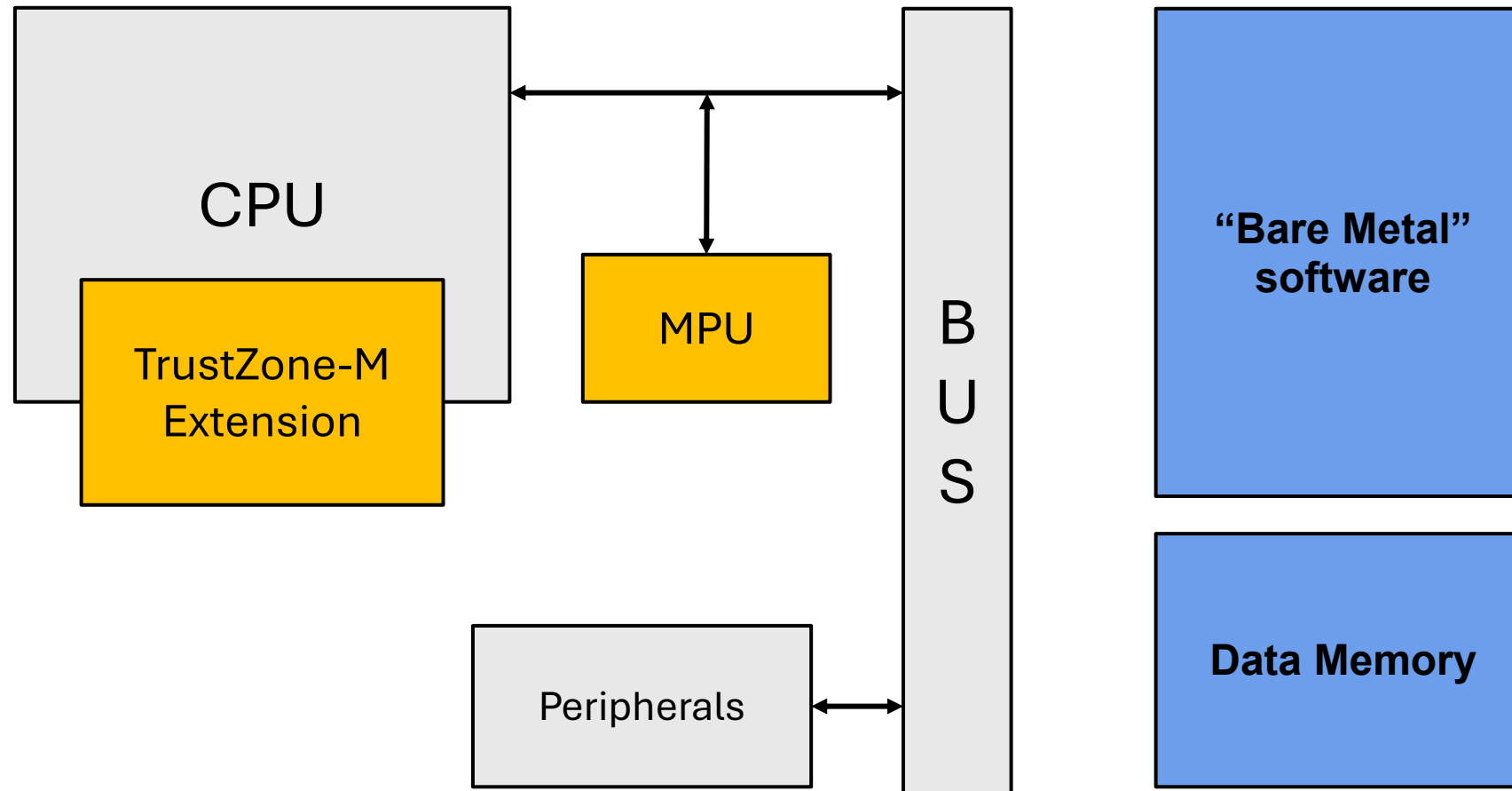
ARM Cortex-M Processors

Follow the same simple computer model



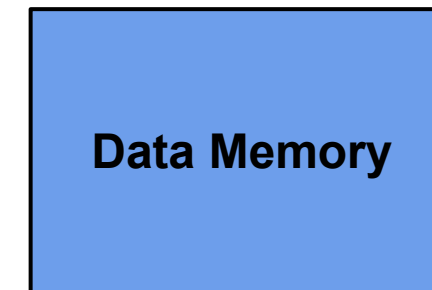
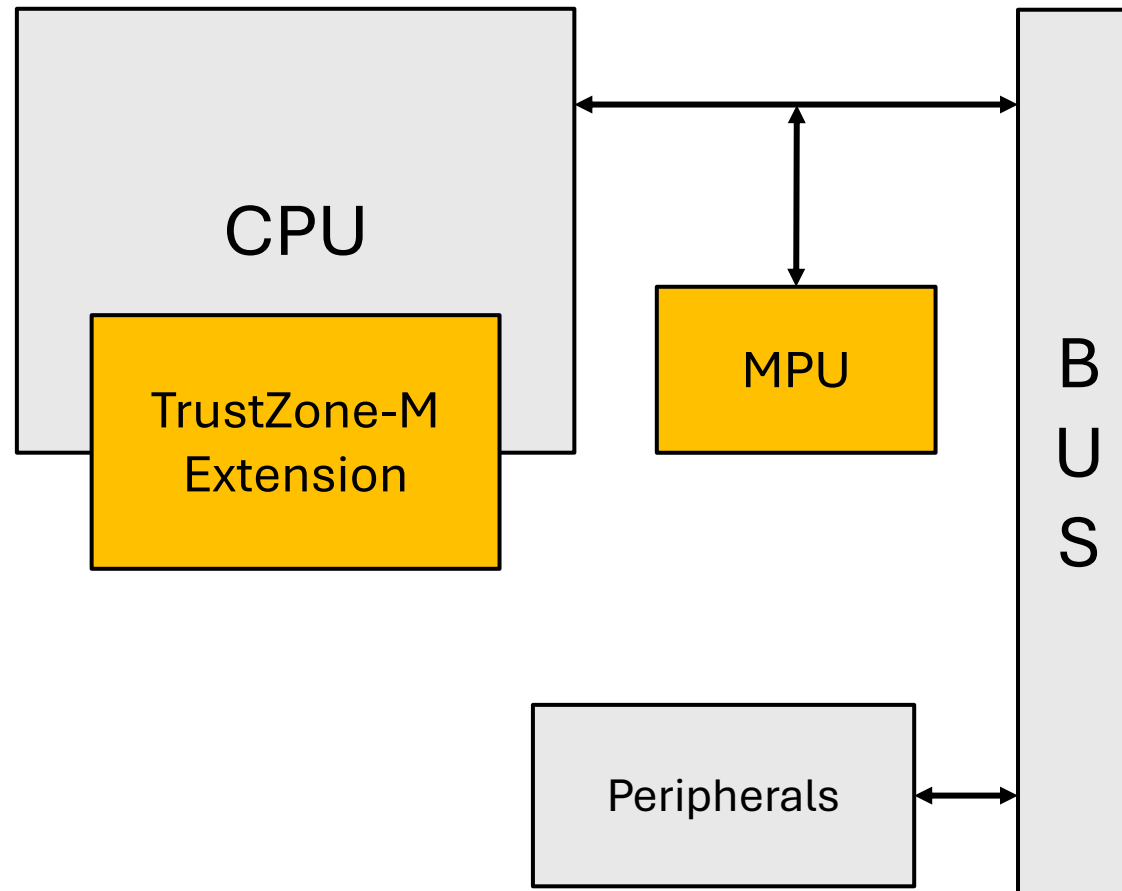
ARM Cortex-M Processors

Follow the same simple computer model



ARM Cortex-M Processors

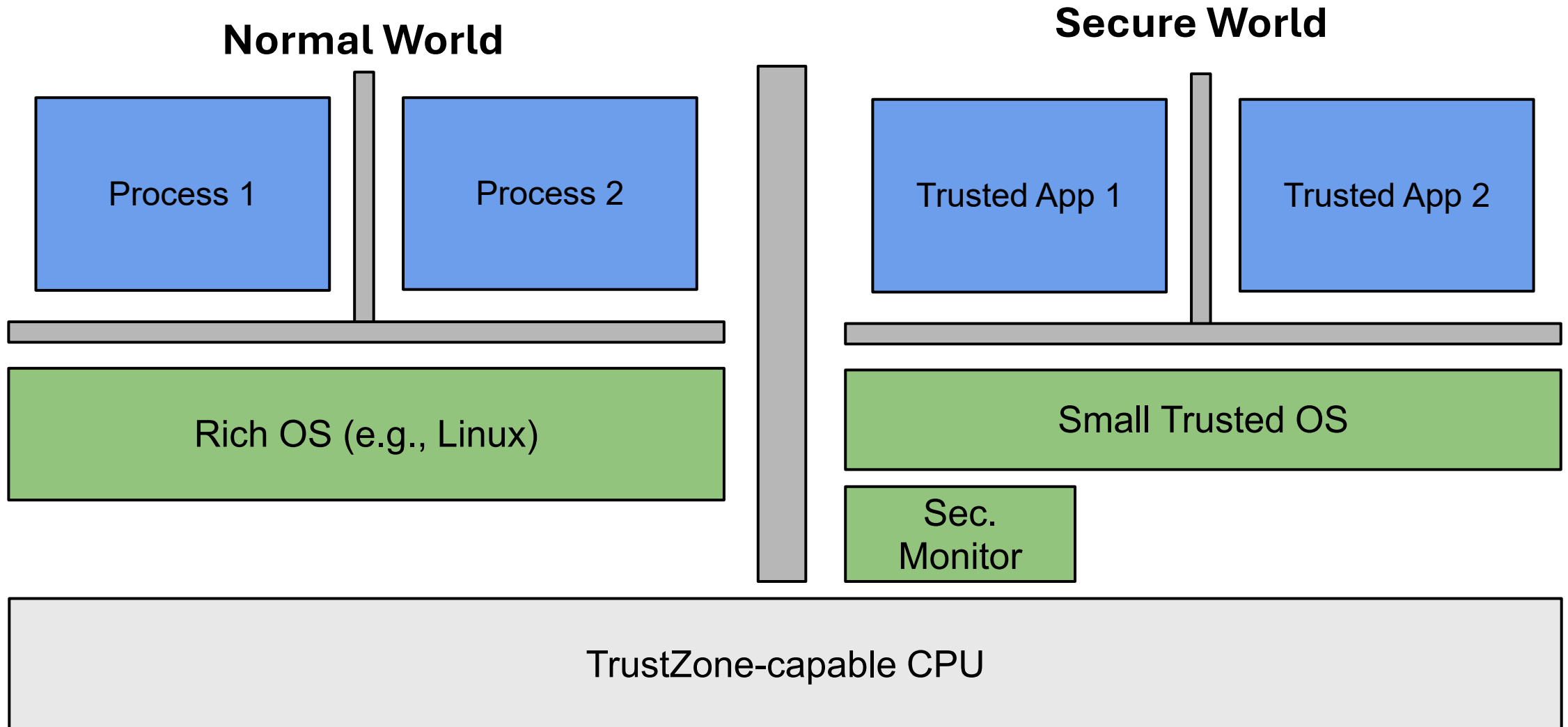
Follow the same simple computer model



**So how do we get
any of the same
guarantees?**

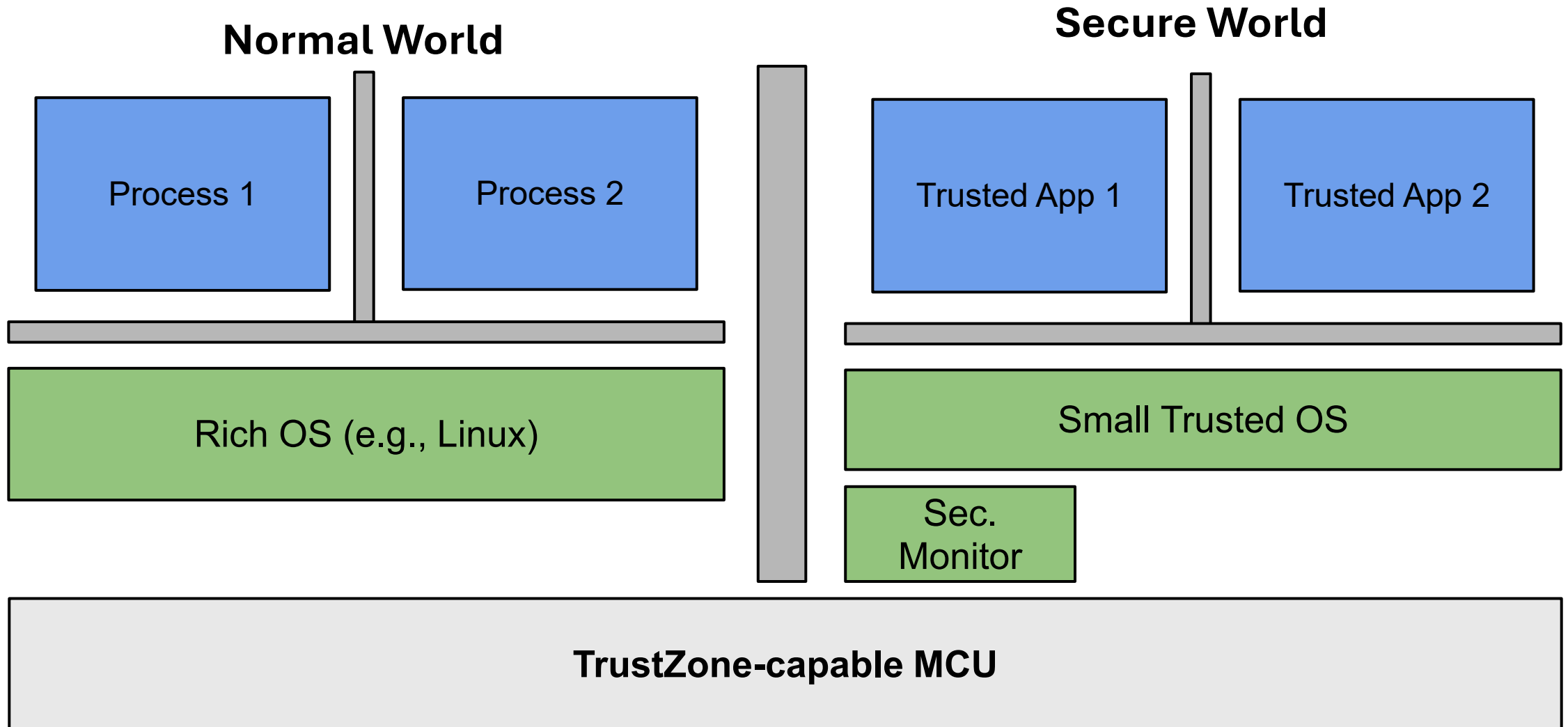
TrustZone-M

First, let's look from the software point of view.



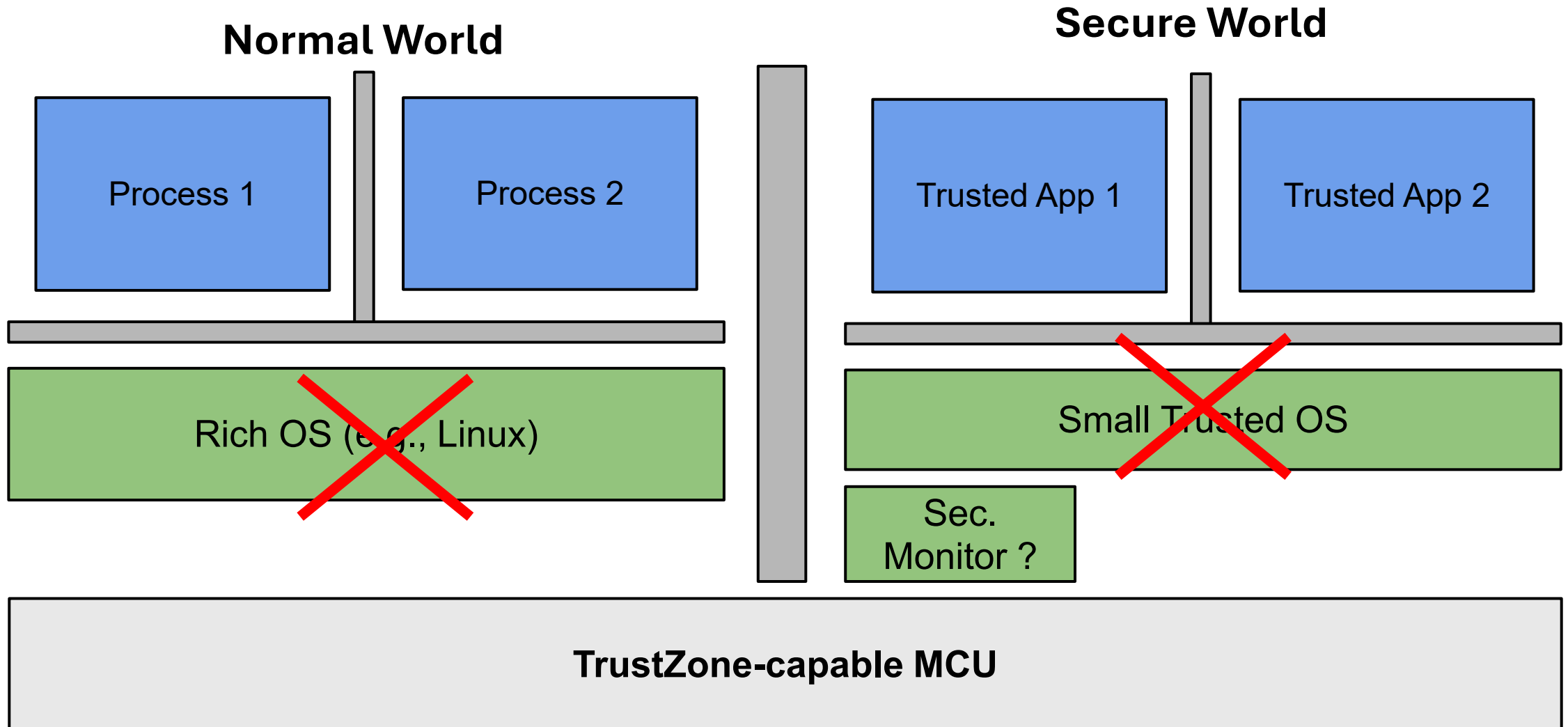
TrustZone-M

First, let's look from the software point of view.



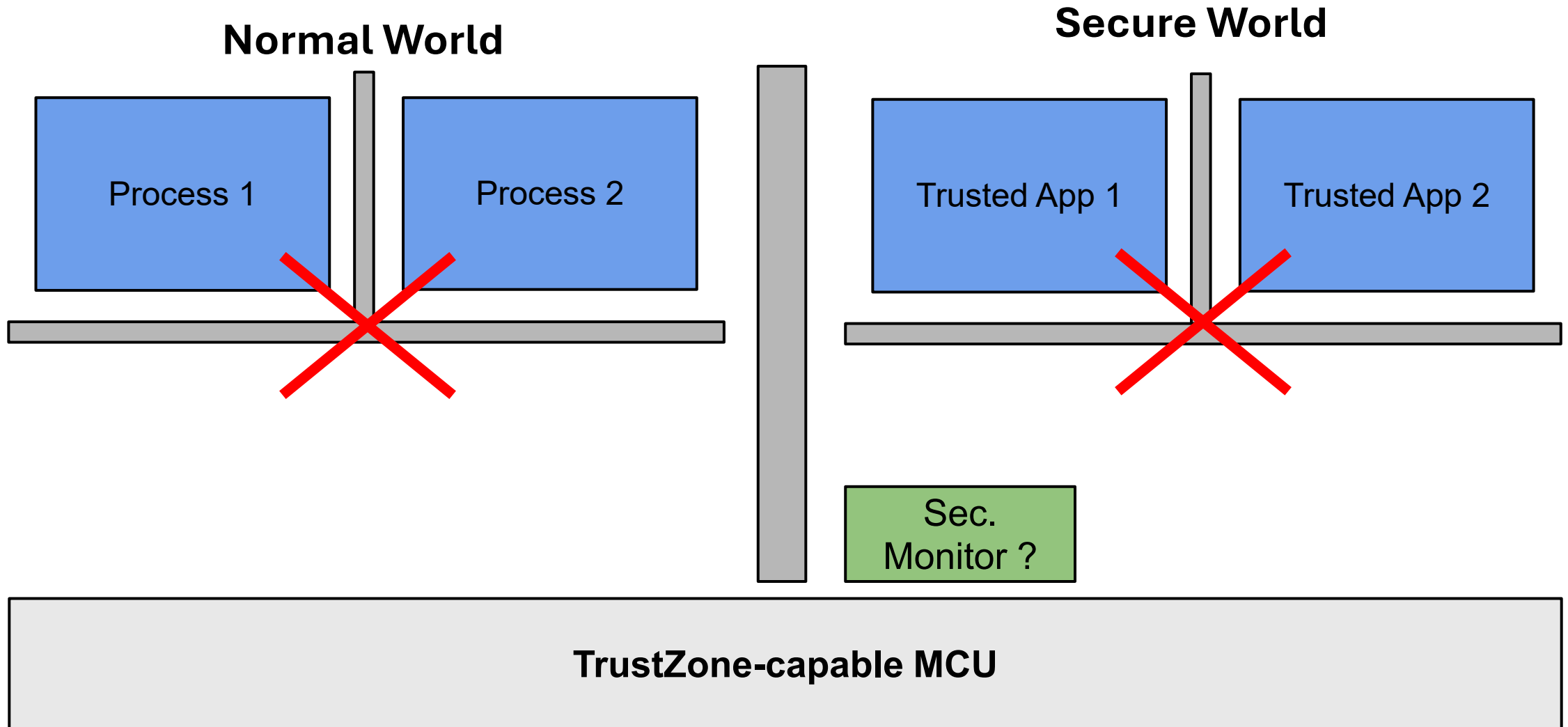
TrustZone-M

First, no MMU



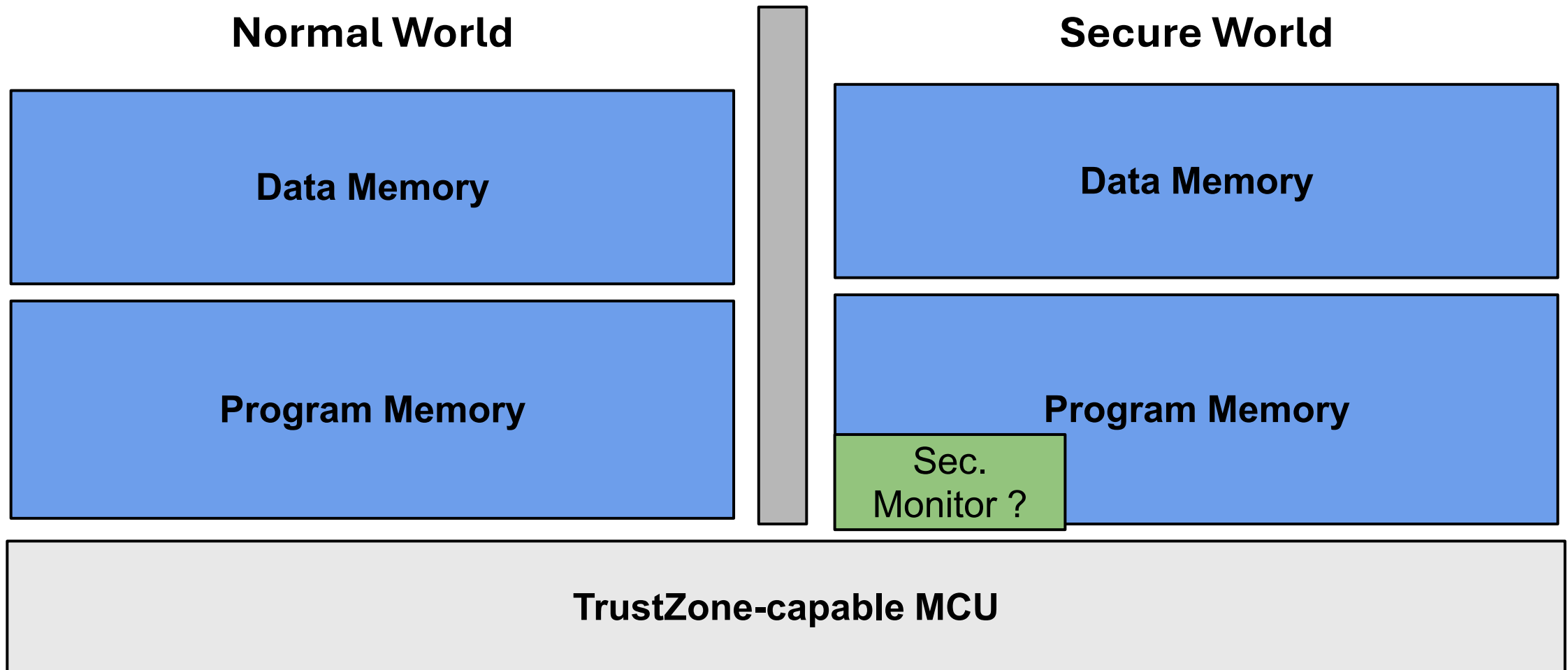
TrustZone-M

First, no MMU



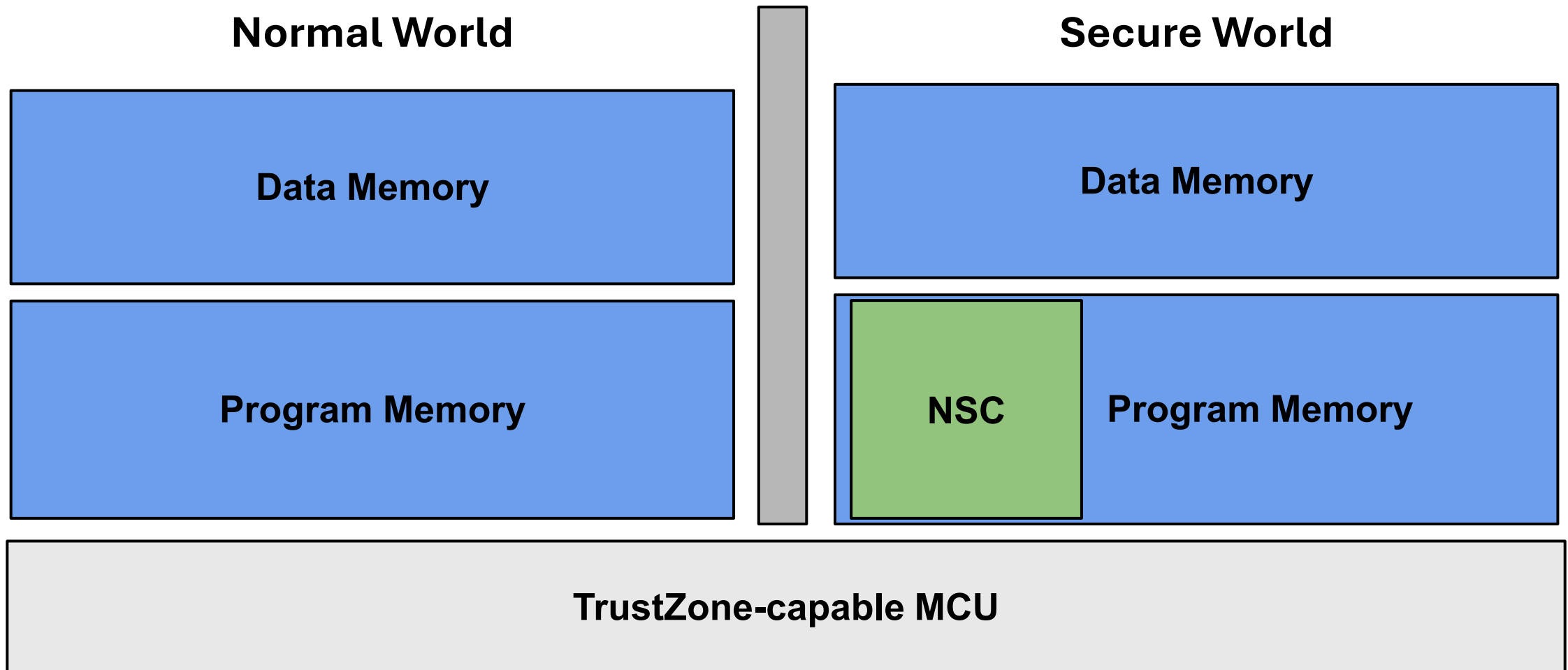
TrustZone-M

What about safe invocation of the Secure World?



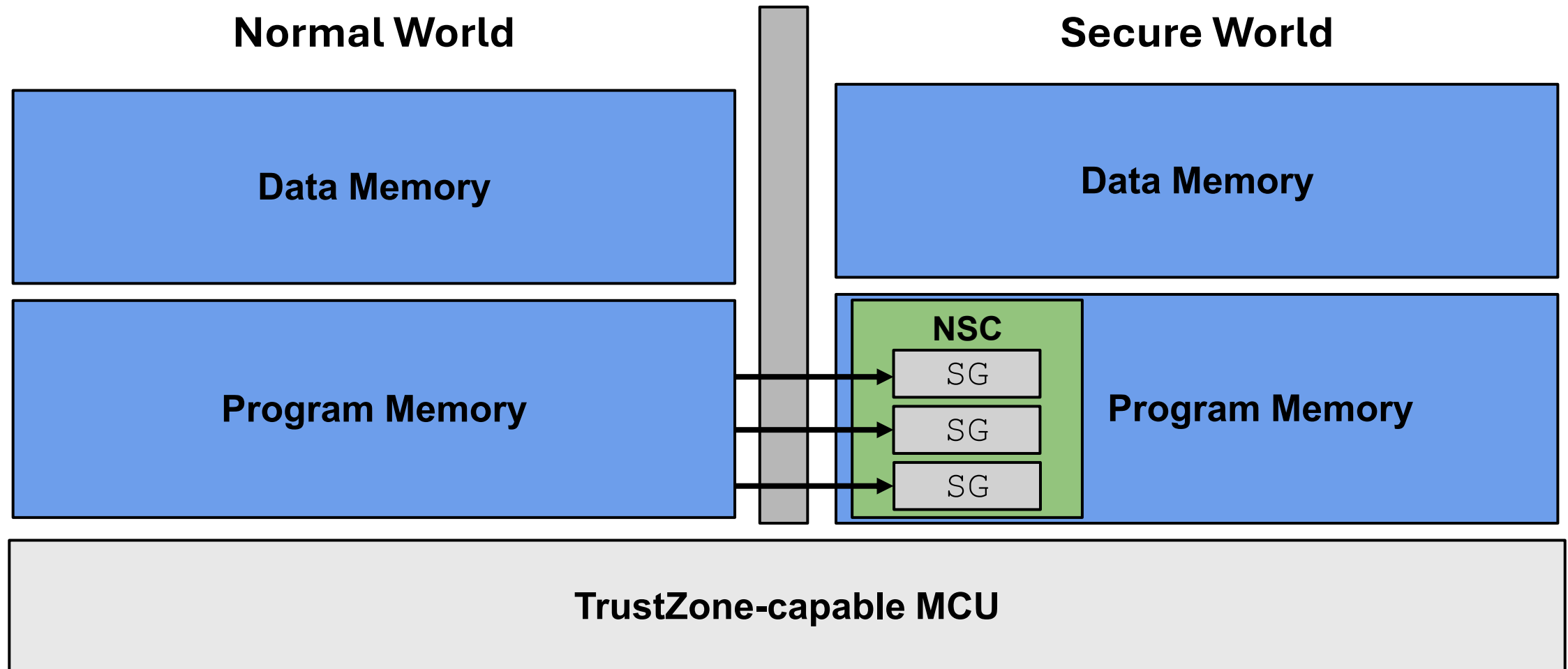
TrustZone-M

Non-Secure Callable (NCS) Region



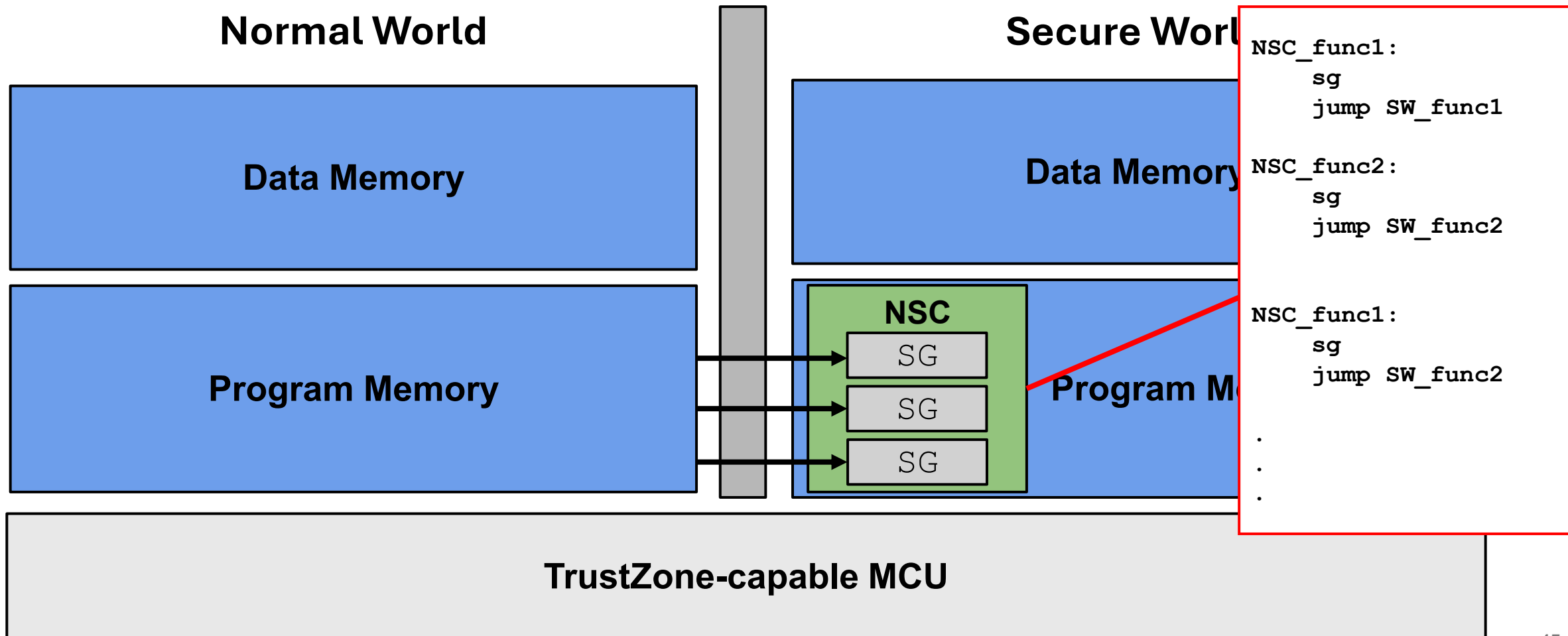
TrustZone-M

Contains “secure gateway” instructions → launch point into SW



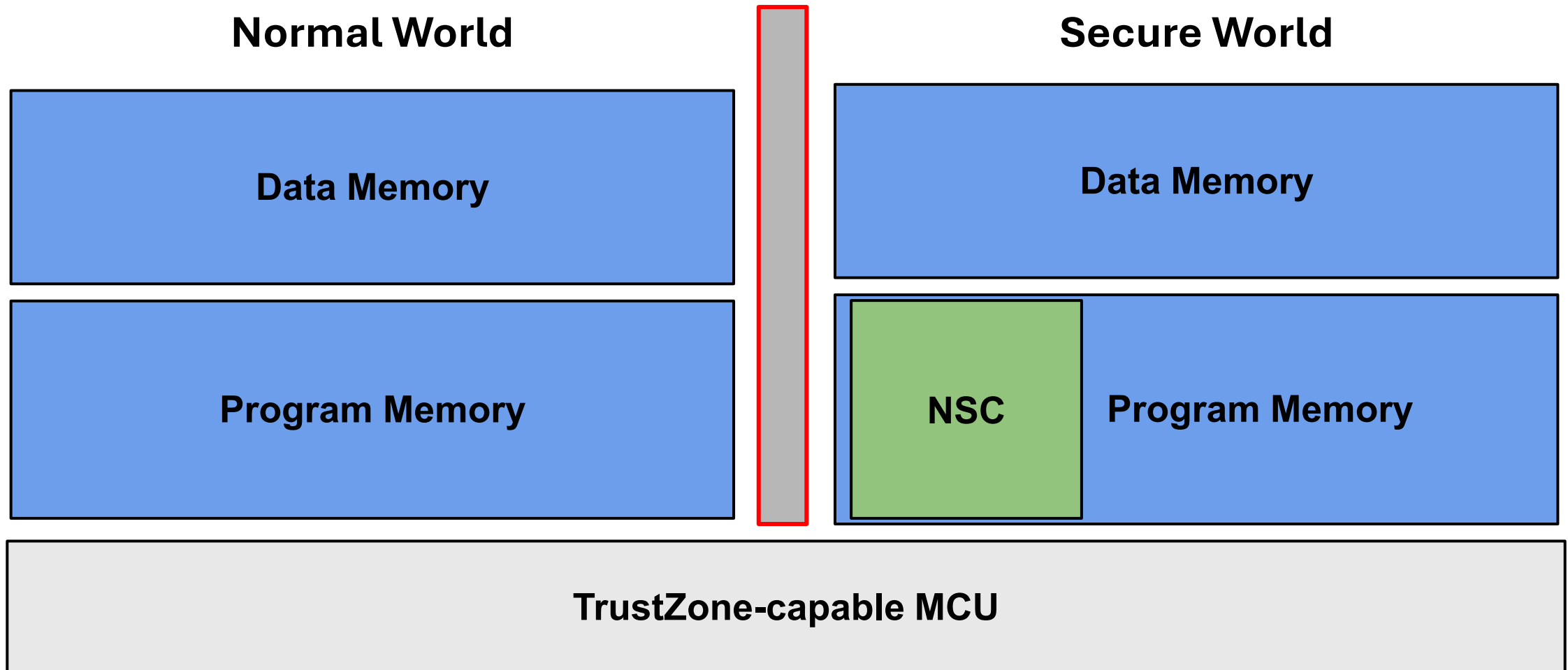
TrustZone-M

Contains “secure gateway” instructions → launch point into SW



TrustZone-M

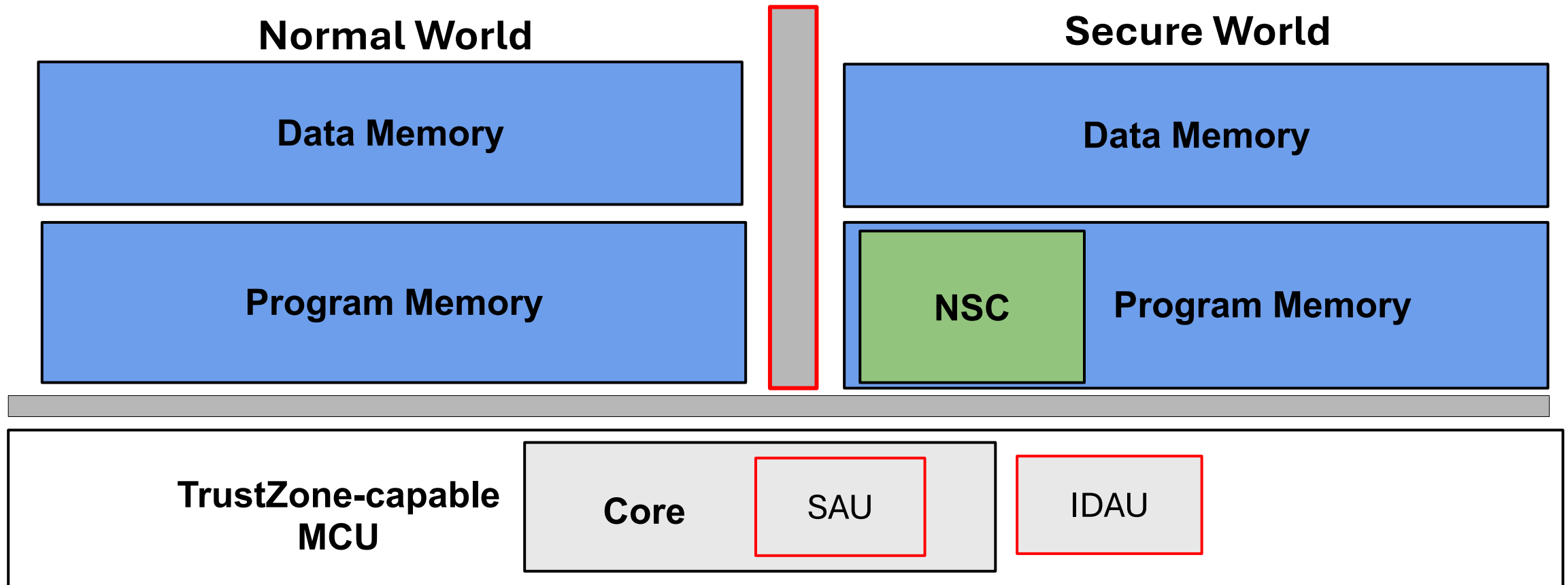
What about isolation?



TrustZone-M

Hardware Controllers:

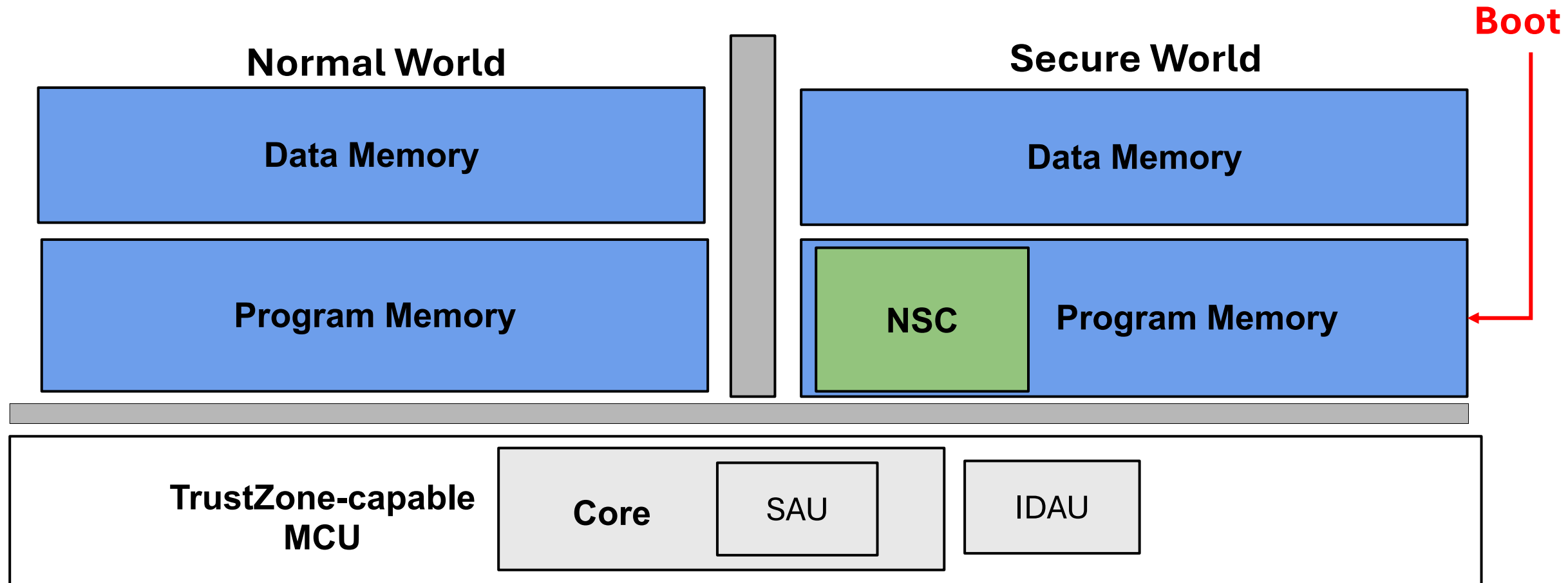
- Implementation-Defined Attribution Unit (IDAU) → enforces fixed SW definition
- Secure Attribution Unit (SAU) → extends SW definition, enforces isolation
- Assign an “attribution bit” (i.e., NS bit) to each address. Allow access if addresses match



TrustZone-M

Secure World boots first!

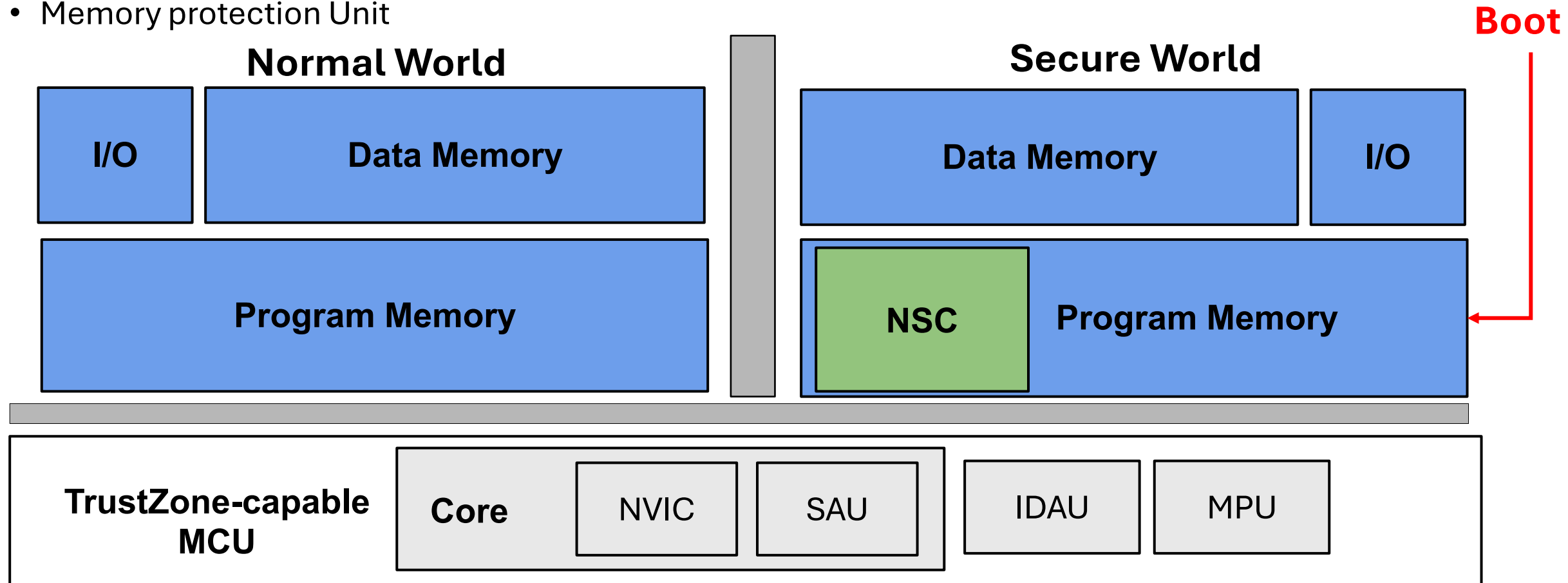
- Can configure the SAU to setup Secure and Normal Worlds



TrustZone-M

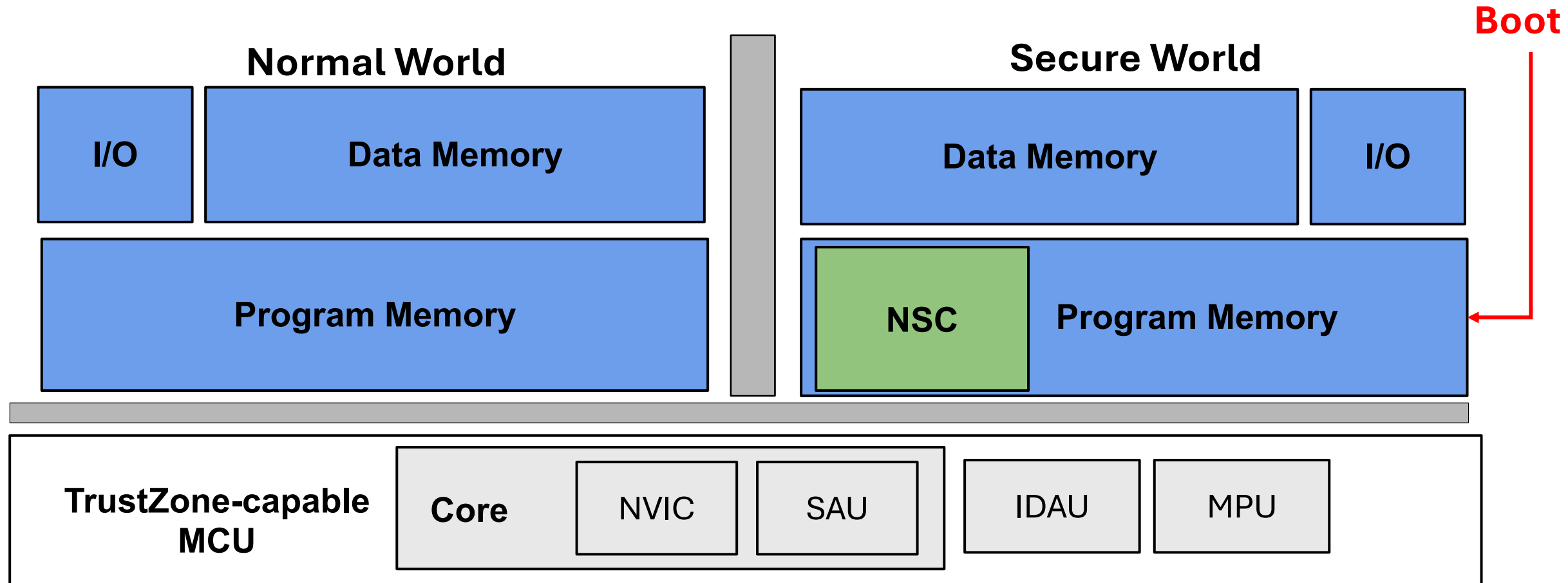
Final notes: Other components that are “split”

- Peripherals (I/O)
- Dedicated interrupt controller (NVIC)
- Memory protection Unit



TrustZone-M

Done TrustZone-M!



Other Research in Systems and Software Security

Embedded Systems

- How does the system model change?
 - **Custom Hardware Extensions in Research** **Done!**
- What type of system-level support is available in today's devices?
 - **TrustZone in Cortex-M** **Done!**
- Availability mechanisms
 - How to build into a system? → **GAROTA**
- Advancing attestation protocols
 - “Run-time” attestation → **C-FLAT**

Other Research in Systems and Software Security

Embedded Systems

- How does the system model change?
 - Custom Hardware Extensions in Research **Done!**
- What type of system-level support is available in today's devices?
 - TrustZone in Cortex-M **Done!**
- Availability mechanisms
 - How to build into a system? → **GAROTA** → Assuming no system support (custom hardware ext.)
- Advancing attestation protocols
 - “Run-time” attestation → **C-FLAT** → TrustZone-M

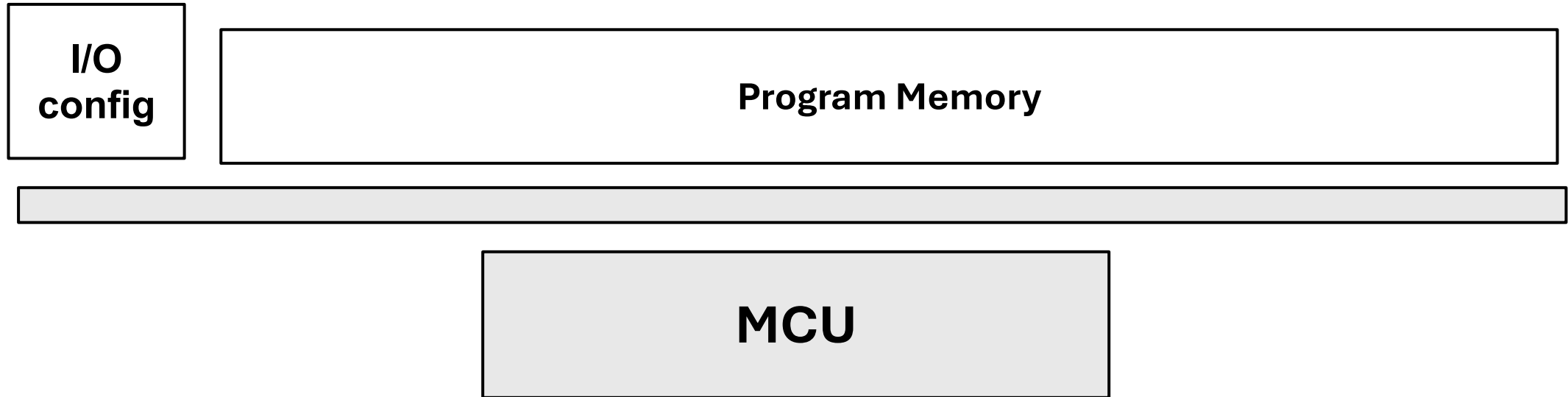
Availability in MCUs

GAROTA → Generalized Active Root of Trust

- Goal:
 - Provide a mechanisms to ensure some critical action ***always executes***
 - Make it generalizable
 - Any general-purpose peripheral on the device can be used
 - E.g., GPIO-triggered active root of trust
 - Make low-cost for MCUs
 - Formally verified

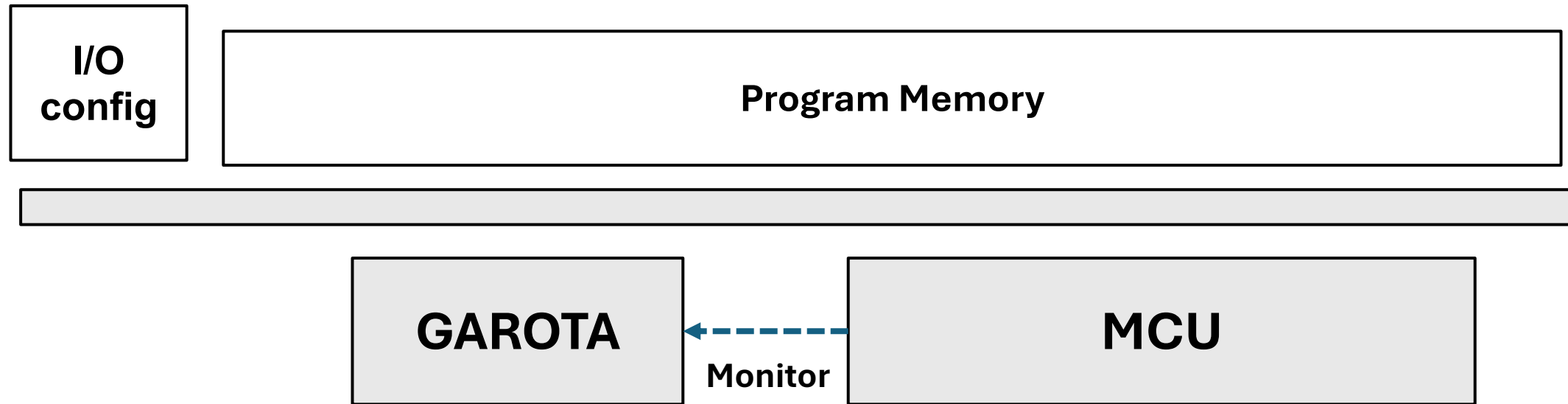
Availability in MCUs

Start with the following address space



Availability in MCUs

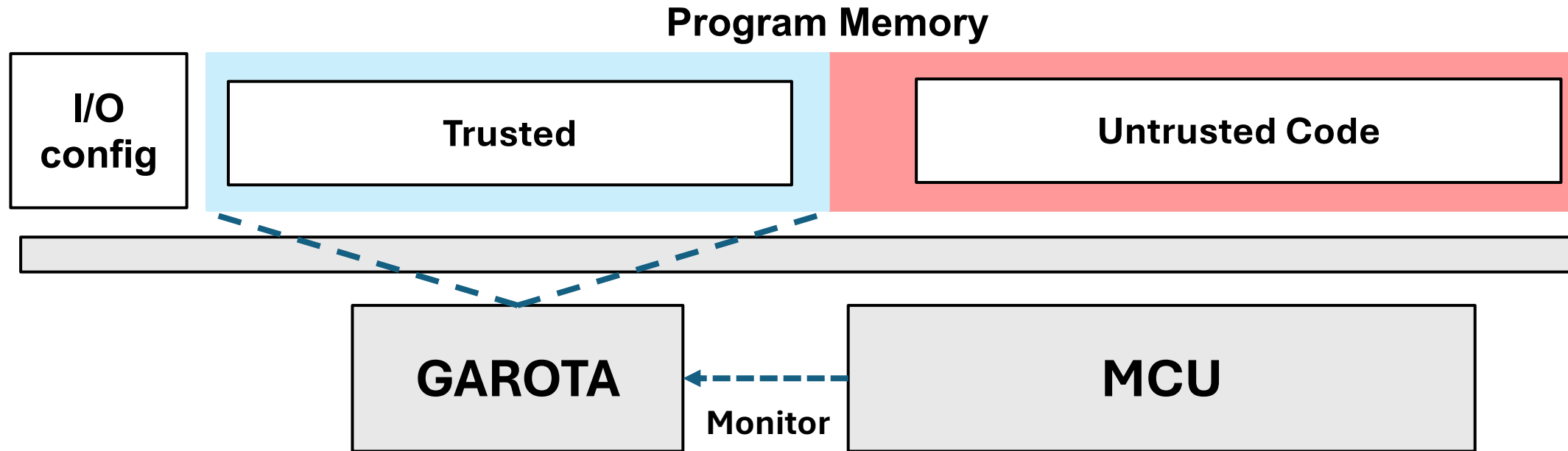
Also have **GAROTA** hardware monitoring MCU signals



Availability in MCUs

GAROTA Splits Program Memory into two regions:

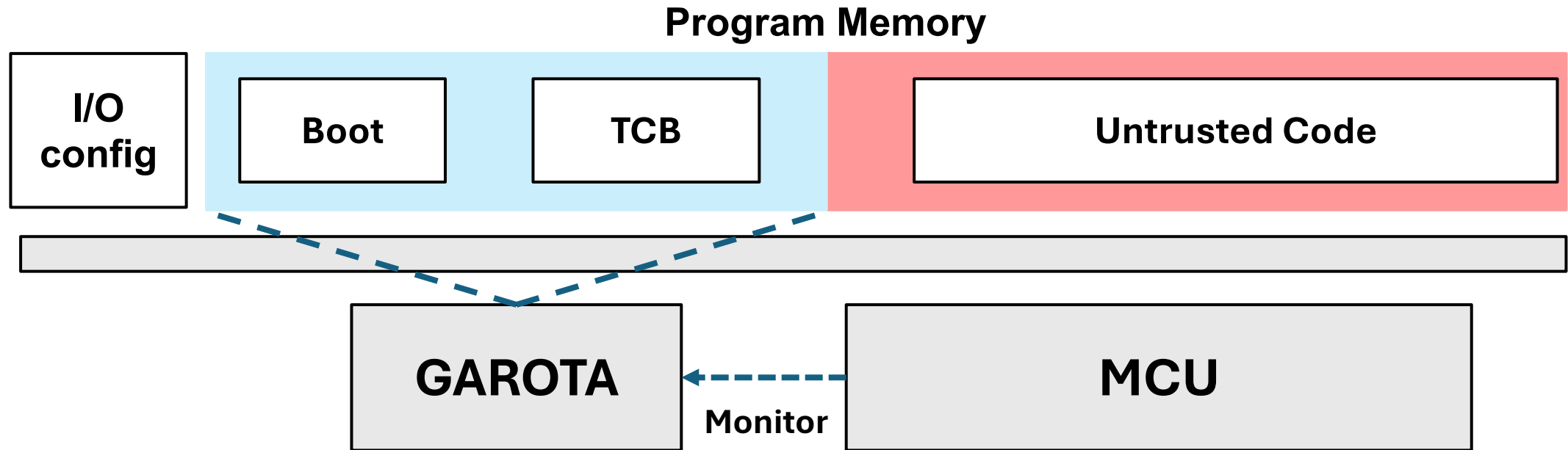
- Trusted (and protected) code
- Untrusted (and unprotected) code



Availability in MCUs

The trusted code has:

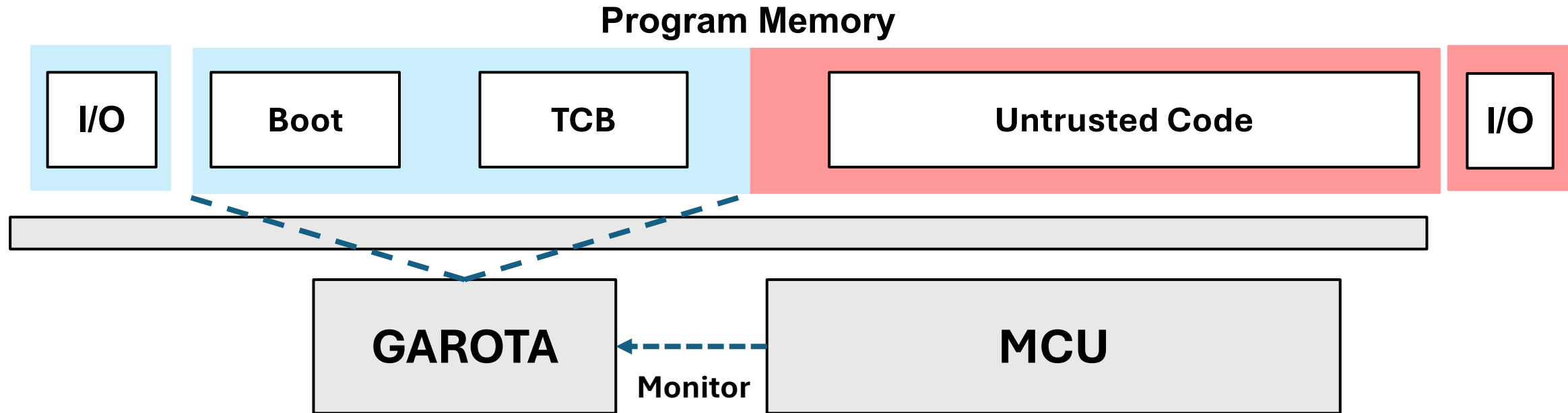
- Boot → sequence to initialize the system
- TCB → GAROTA Trusted Computing Base → the action whose availability is protected



Availability in MCUs

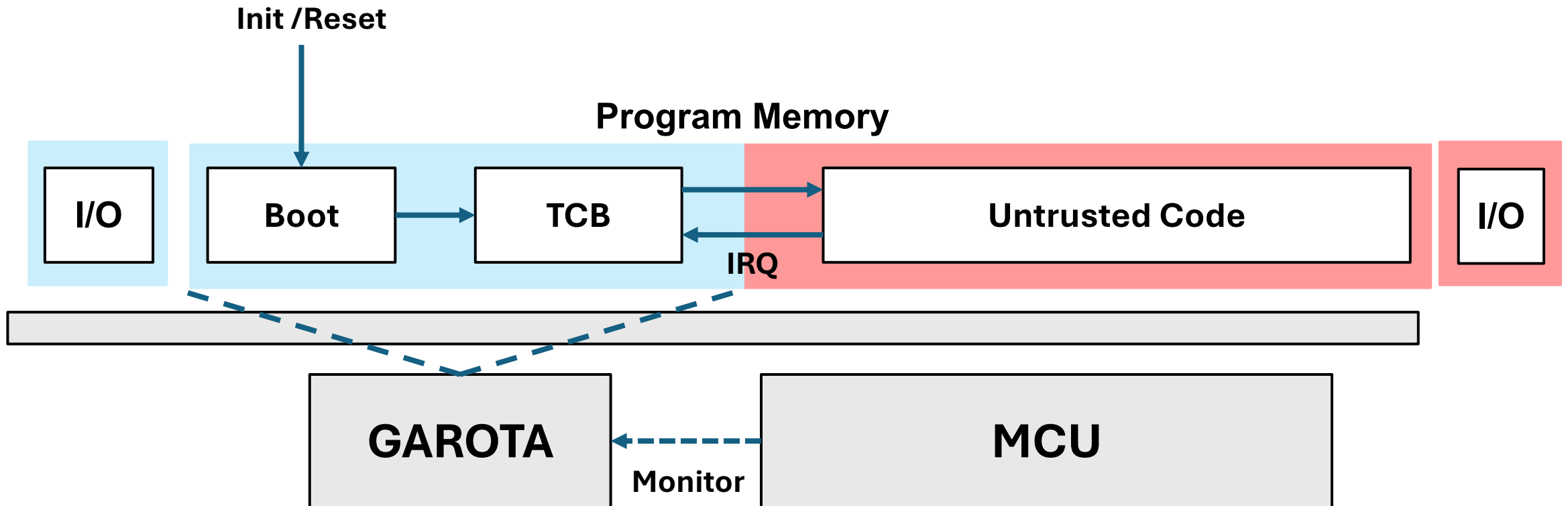
TCB is paired with a particular general-purpose IO device

- It's configs are also monitored by GAROTA



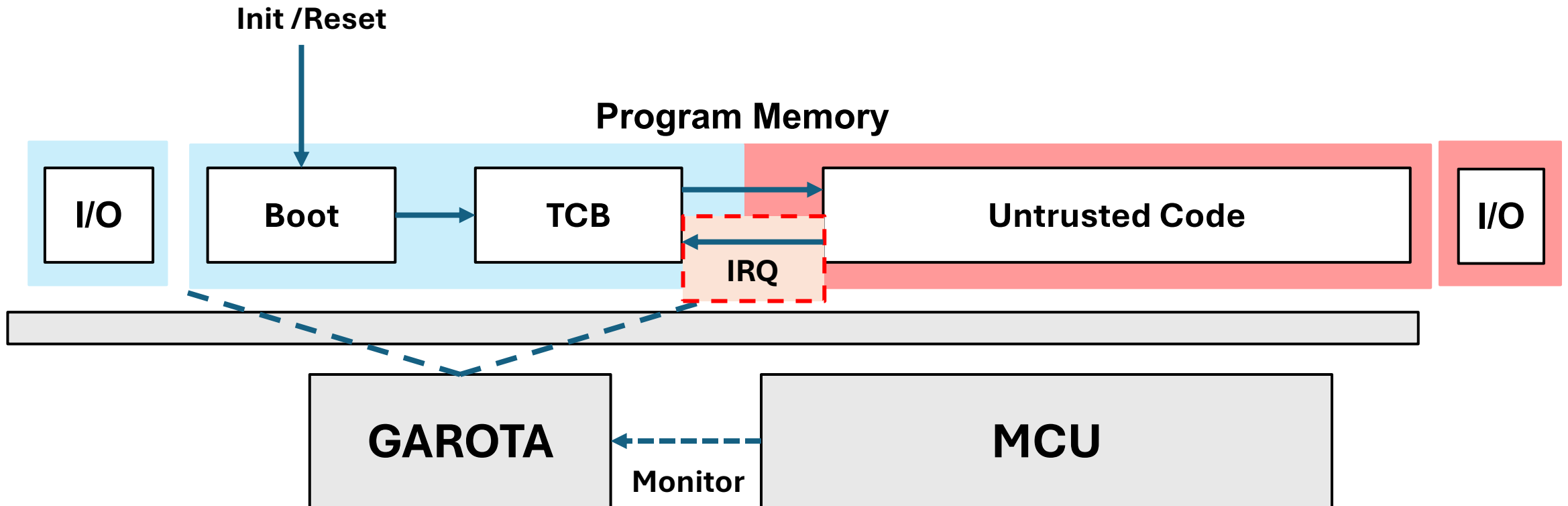
Availability in MCUs

Execution has the following flow:



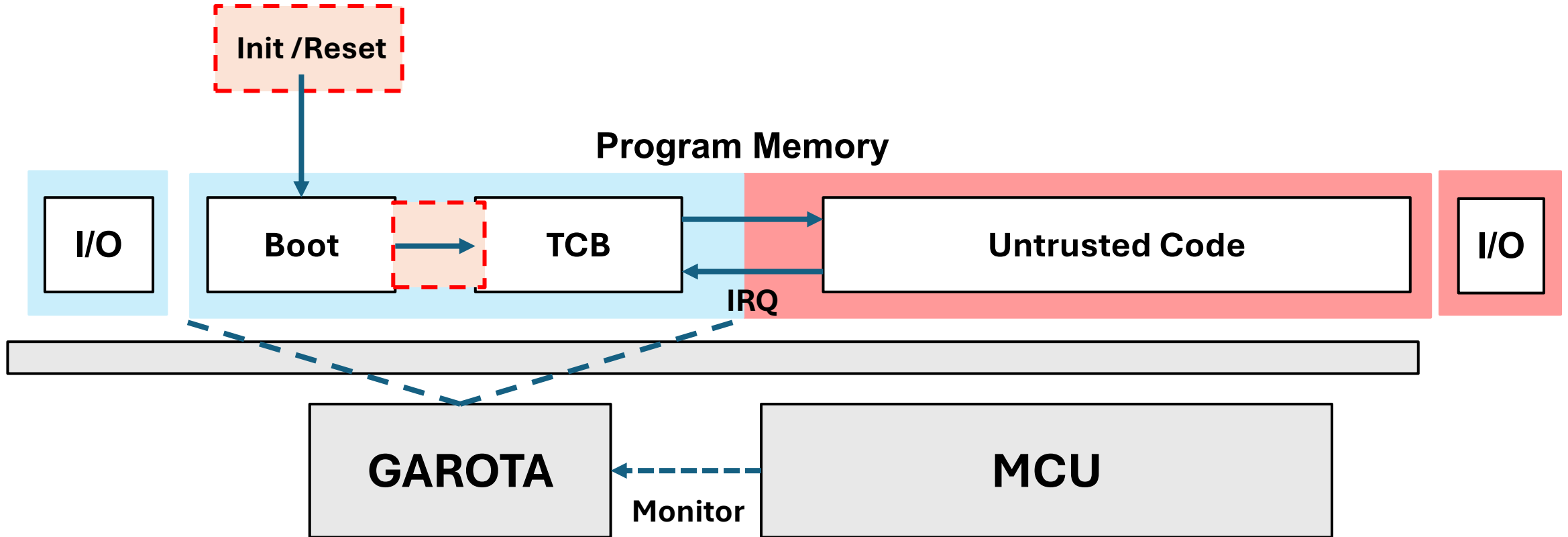
Availability in MCUs

GAROTA guarantees: (1) IRQ from TCB-based I/O will always trigger TCB



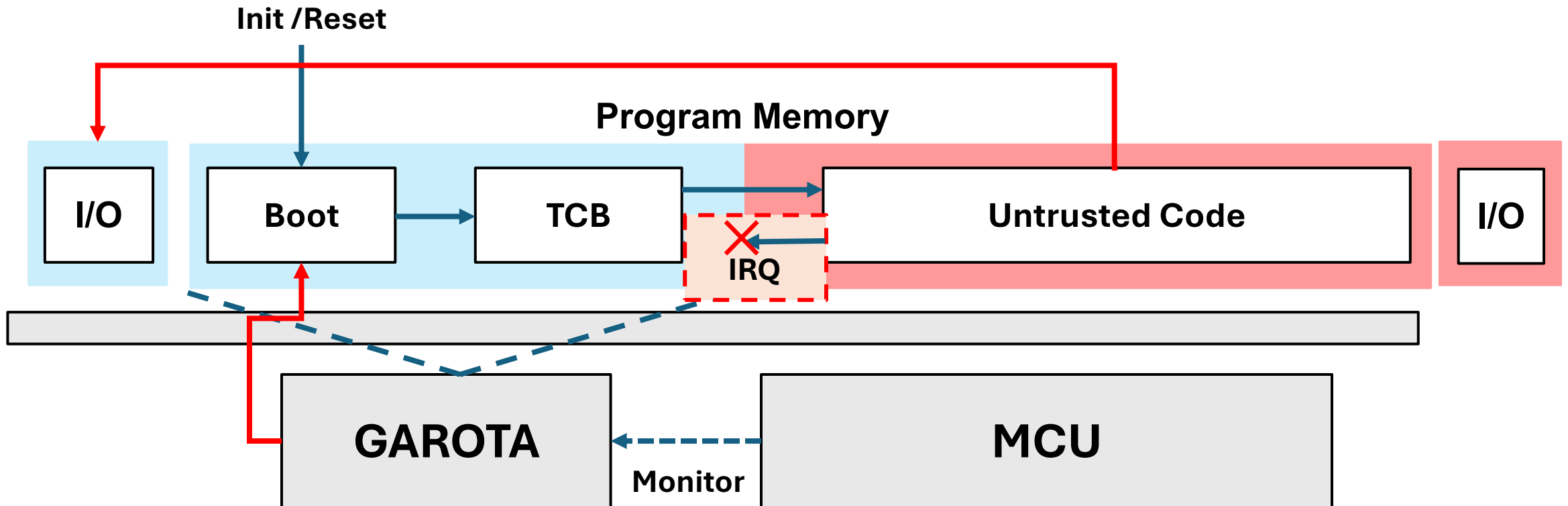
Availability in MCUs

GAROTA guarantees: (2) TCB will always execute after boot/reset



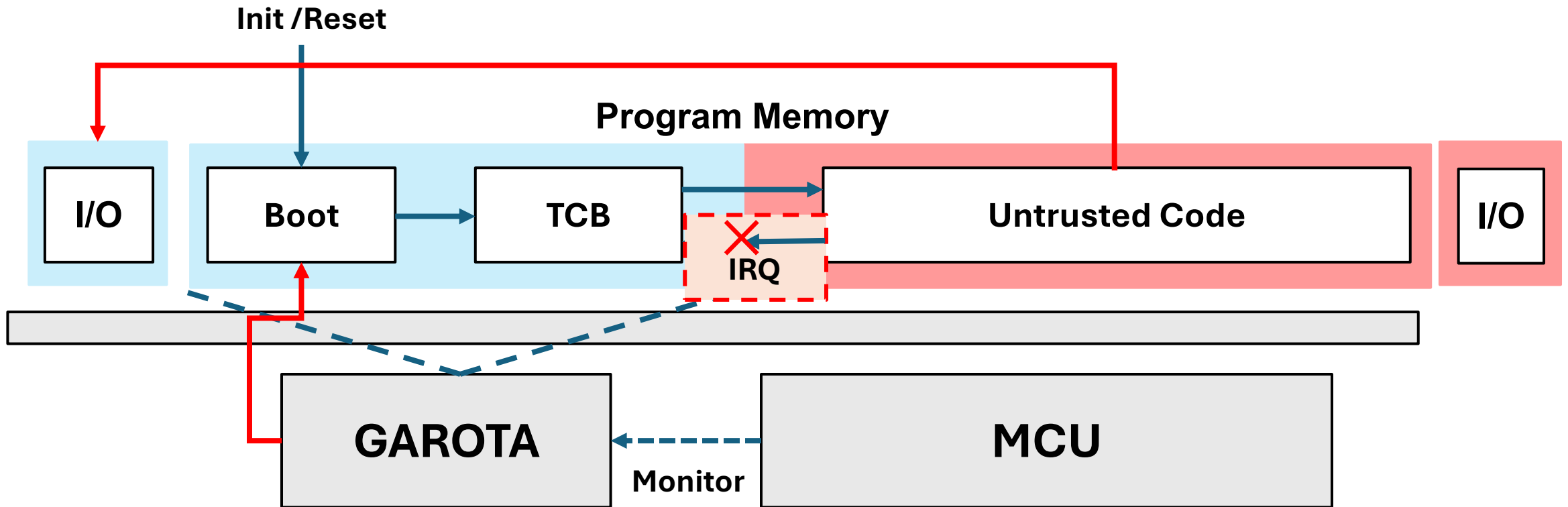
Availability in MCUs

GAROTA guarantees: (3) Attempts to disable IRQ will cause HW-reset



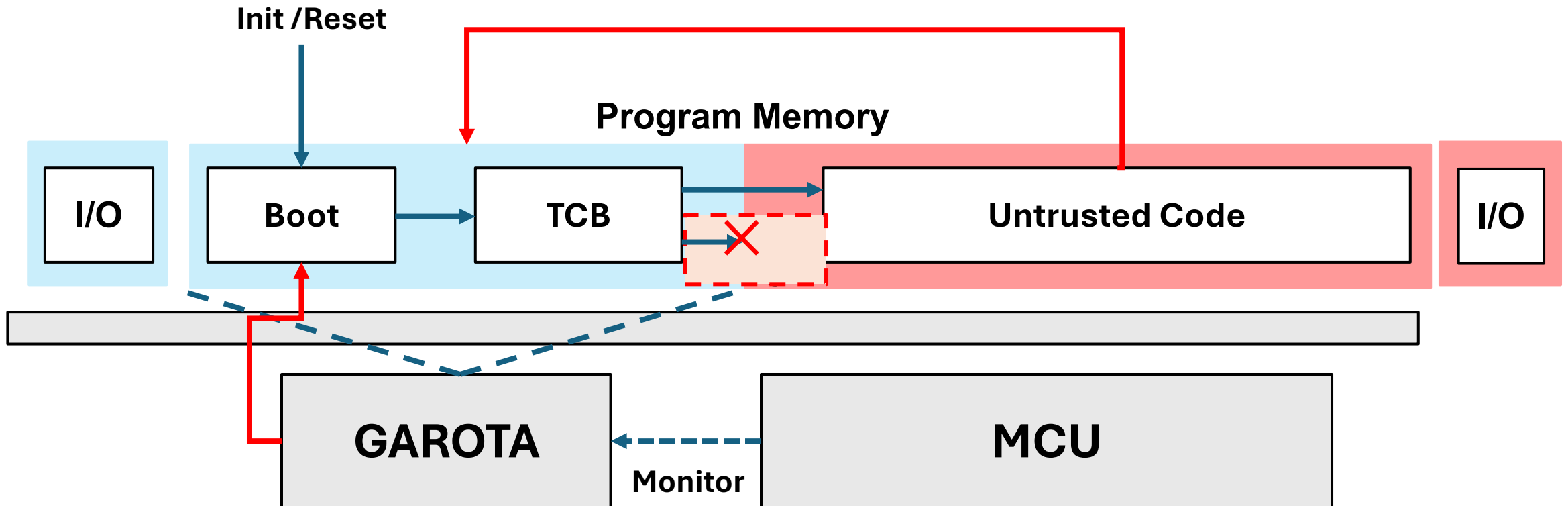
Availability in MCUs

GAROTA guarantees: (3) Attempts to disable IRQ will cause HW-reset



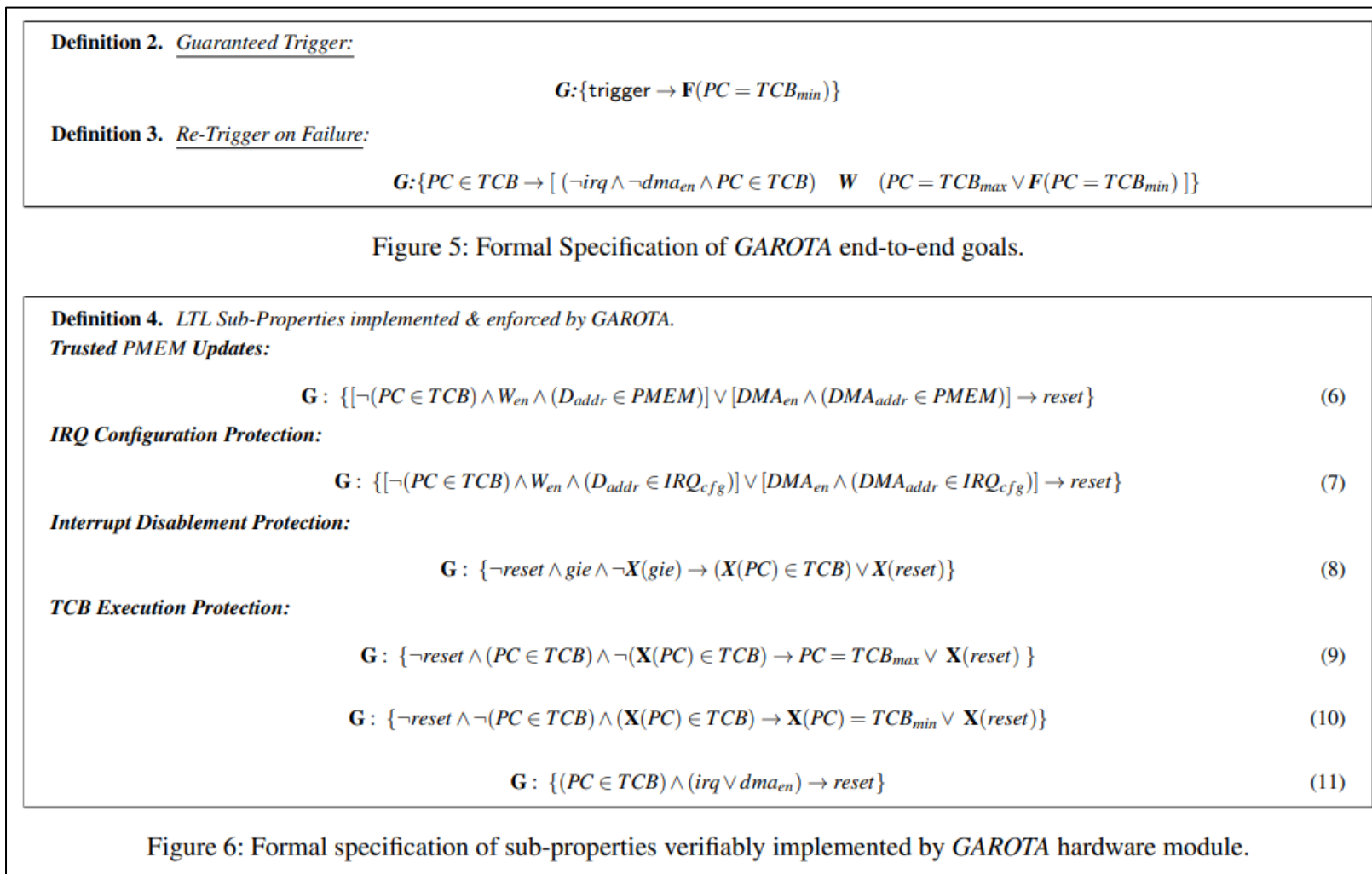
Availability in MCUs

GAROTA guarantees: (4) Tampering or interrupting TCB results in HW-reset




Availability in MCUs

GAROTA specifications:



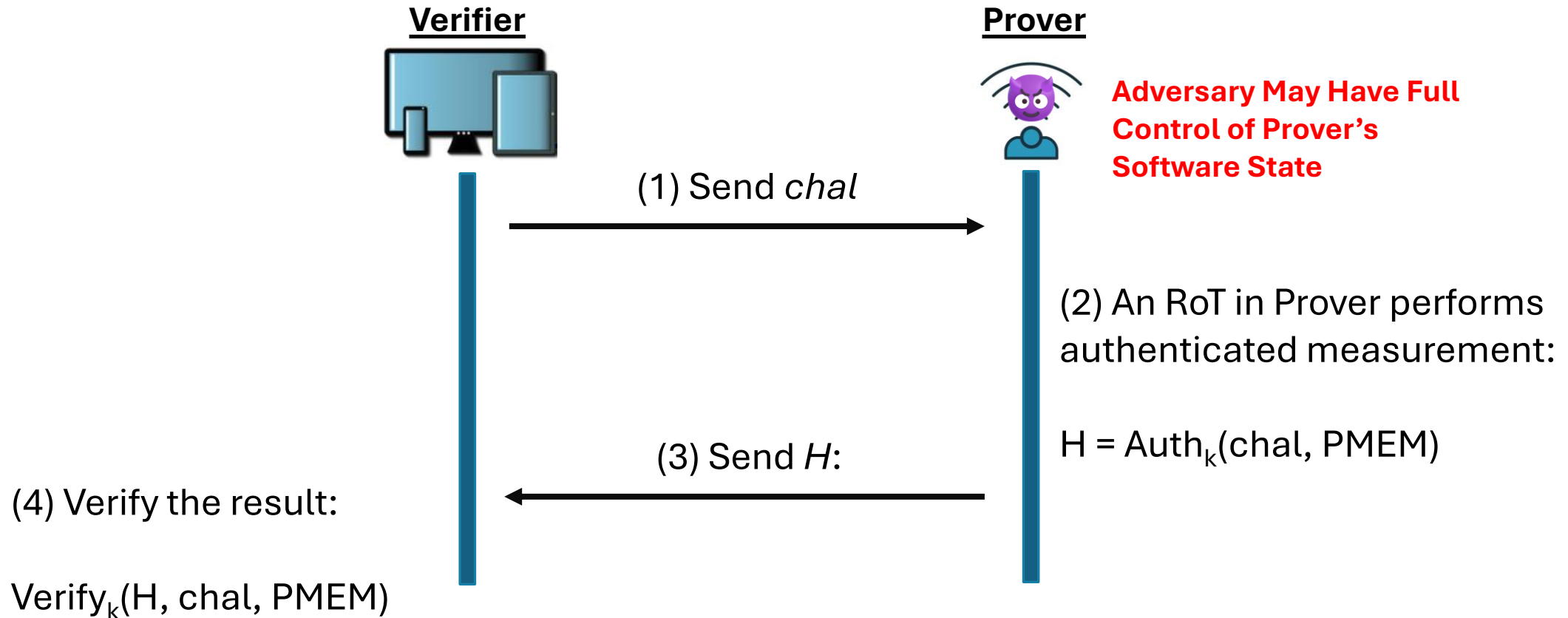
Other Research in Systems and Software Security

Embedded Systems

- How does the system model change?
 - **Custom Hardware Extensions in Research** **Done!**
- What type of system-level support is available in today's devices?
 - **TrustZone in Cortex-M** **Done!**
- Availability mechanisms
 - How to build into a system? → **GAROTA** **Done!**
- Advancing attestation protocols
 - “Run-time” attestation → **C-FLAT**  **TrustZone-M**

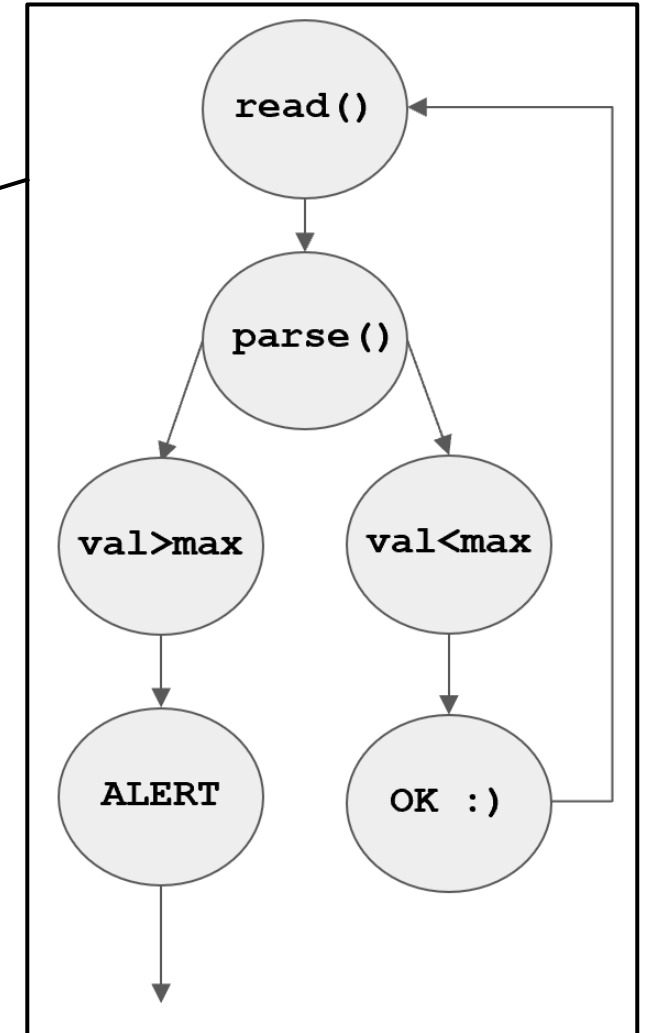
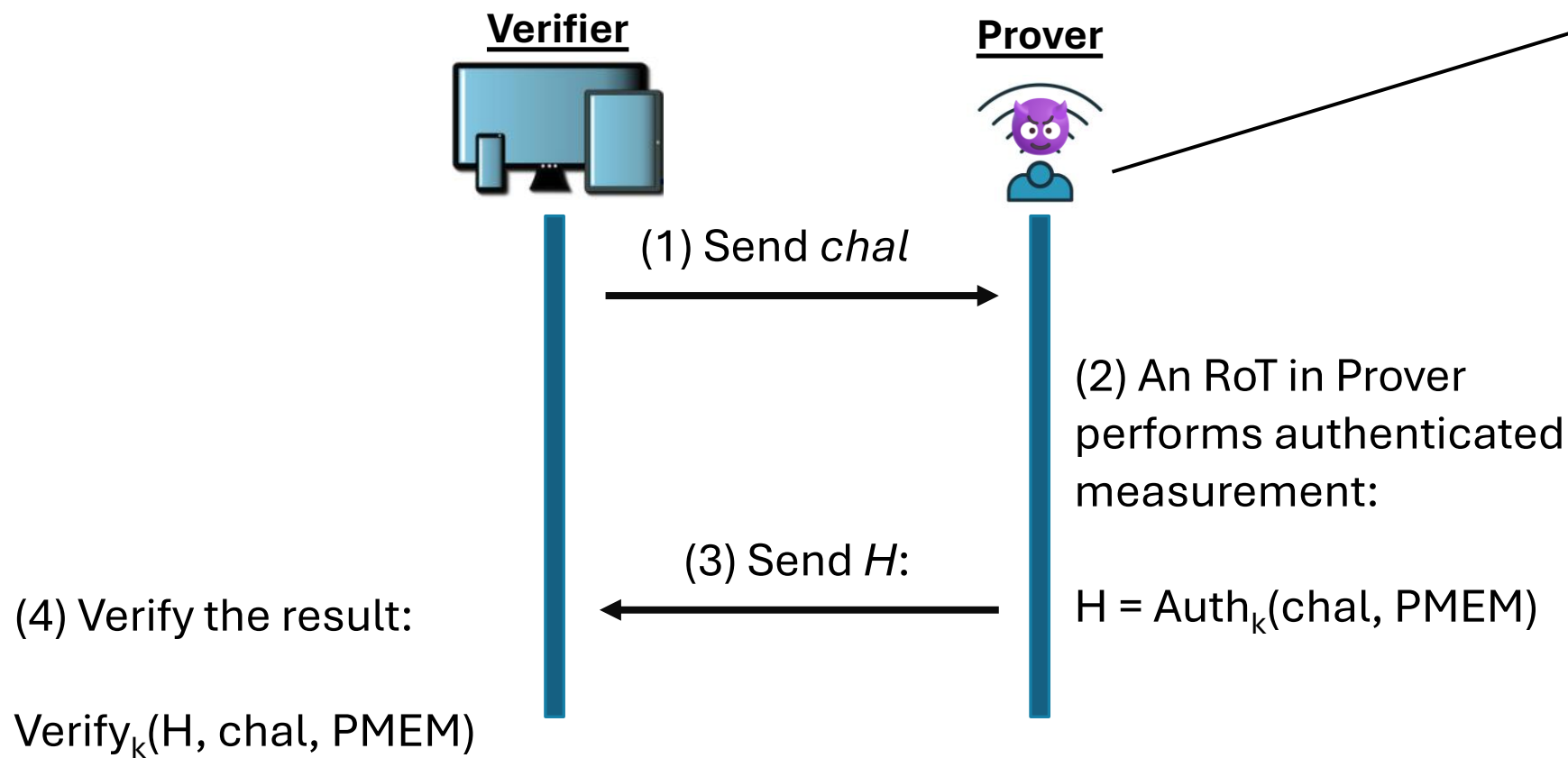
Run-time Attestation

Recall this Attestation Protocol



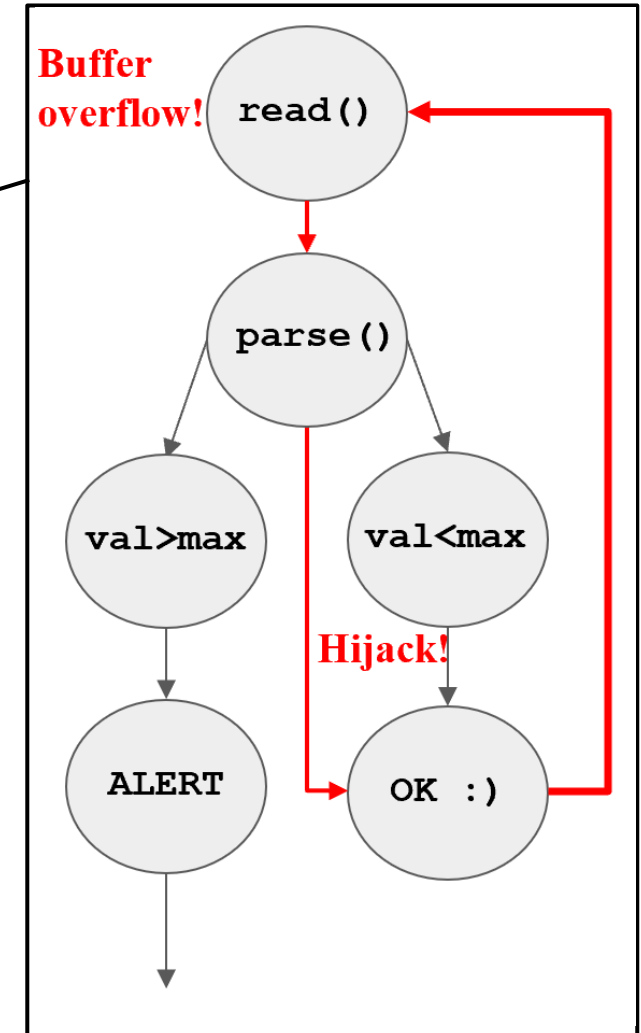
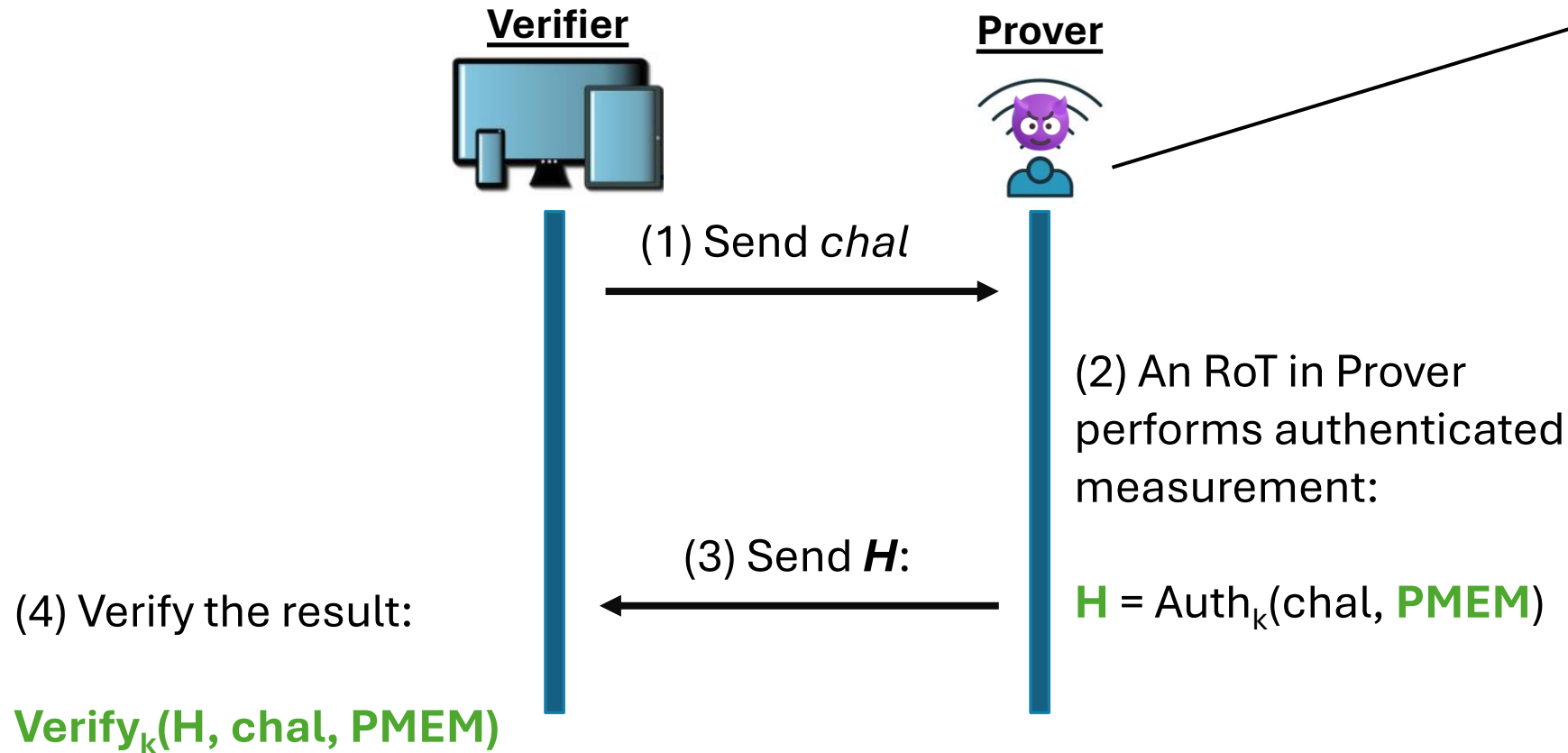
Run-time Attestation

Recall this Attestation Protocol



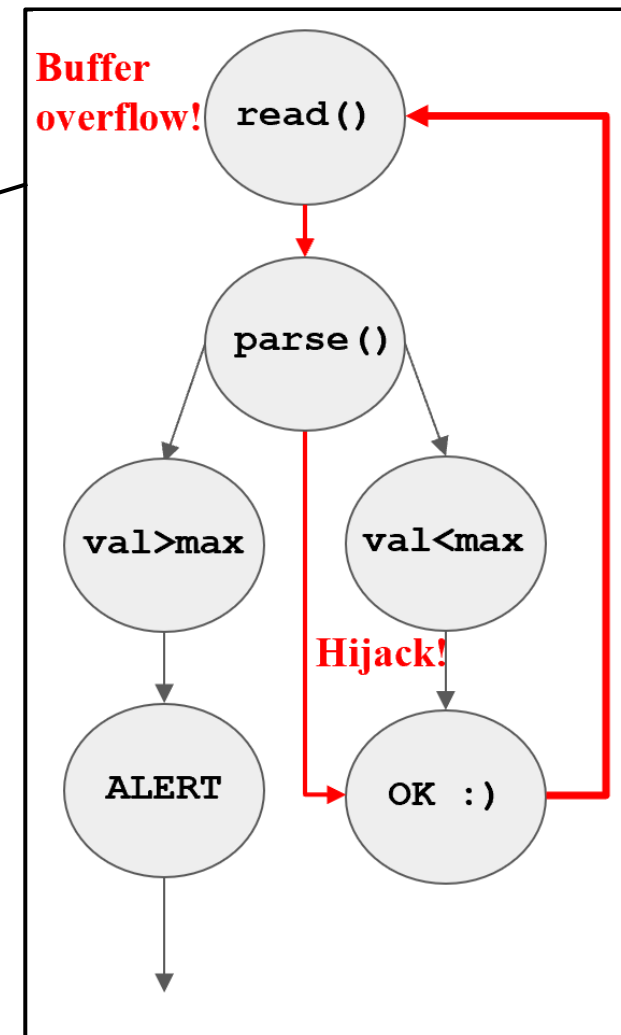
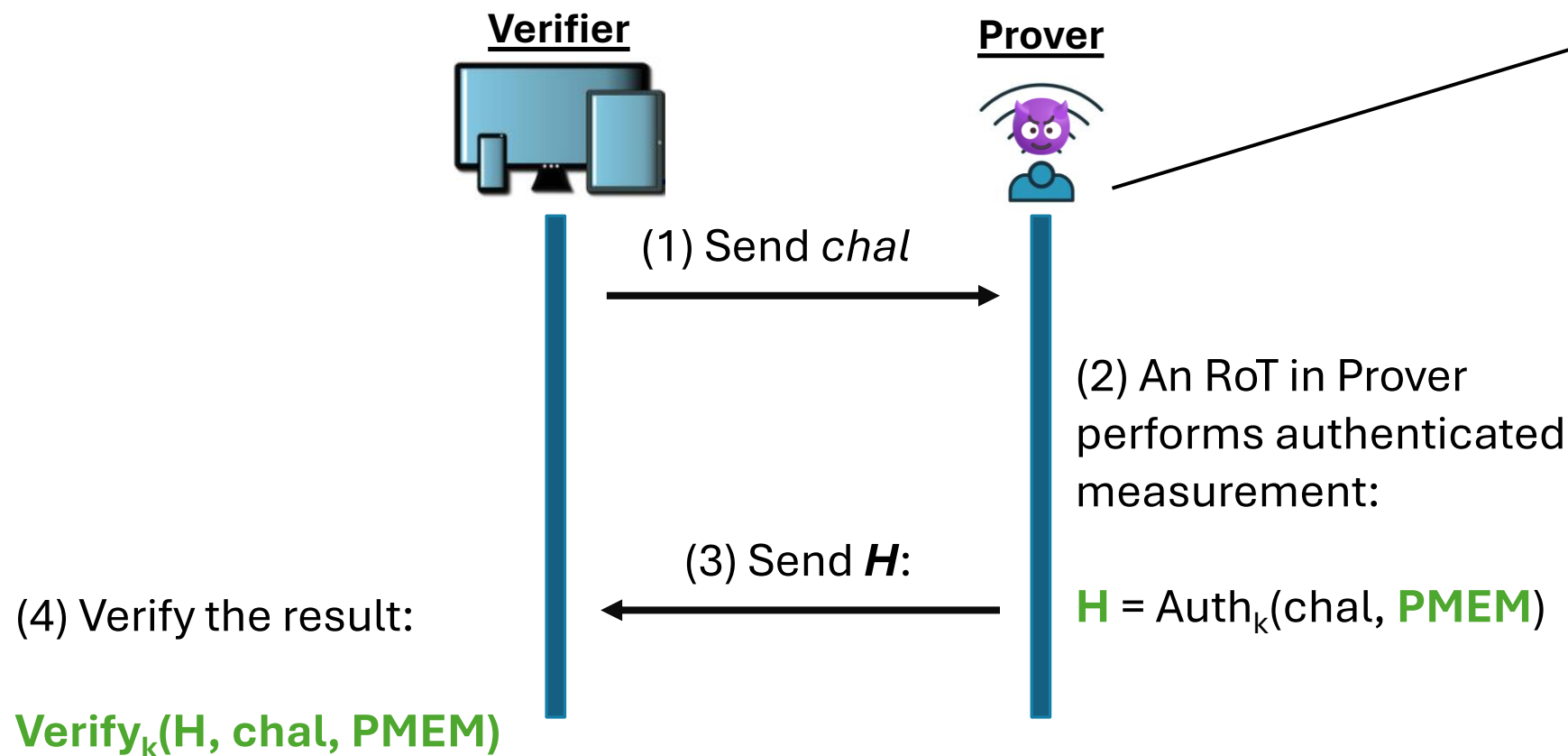
Run-time Attestation

Recall this Attestation Protocol



Run-time Attestation

Recall this Attestation Protocol



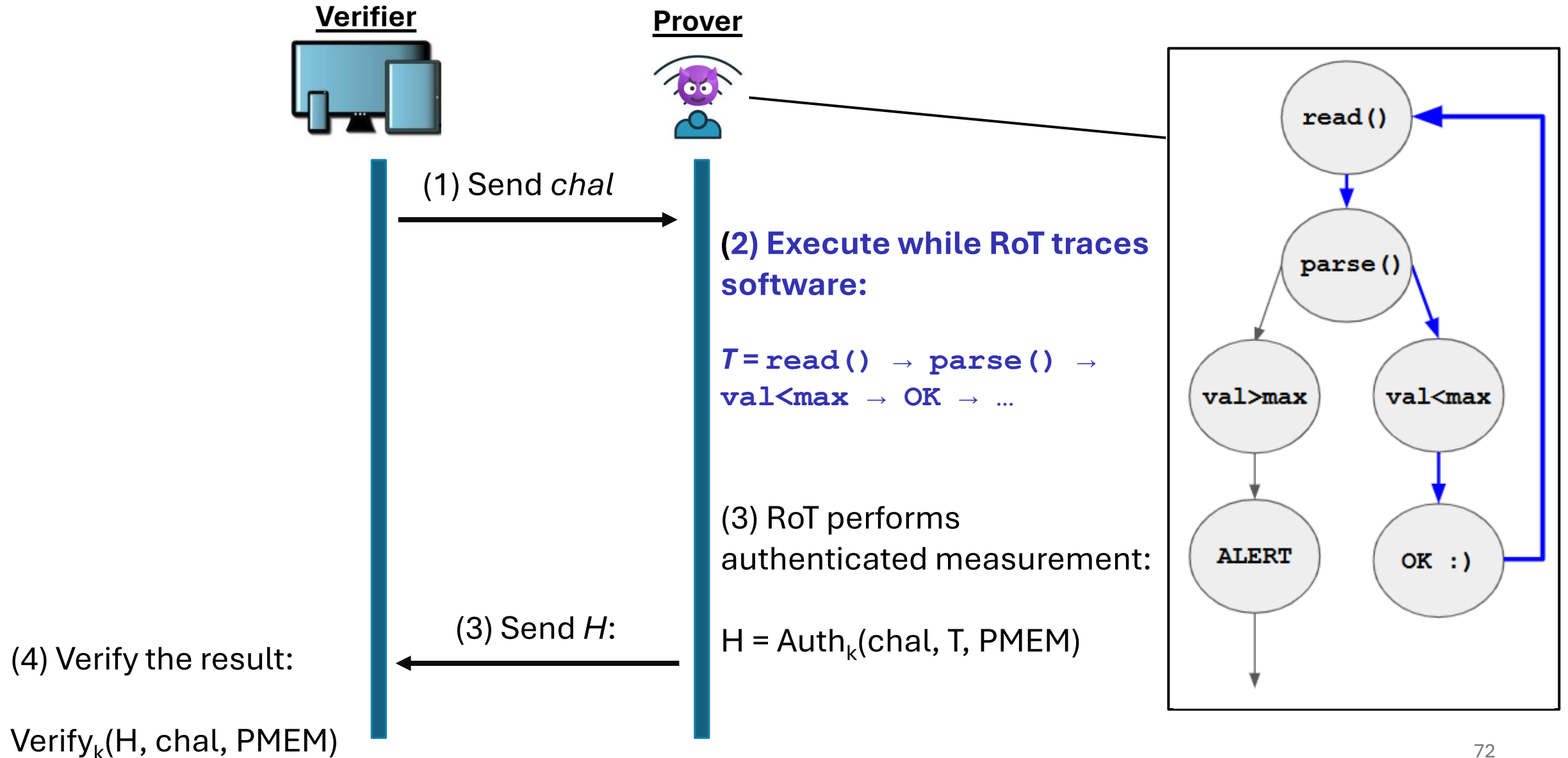
Run-time attacks (like control flow hijack) do not require modifying memory → bypass RA!!

Run-time Attestation

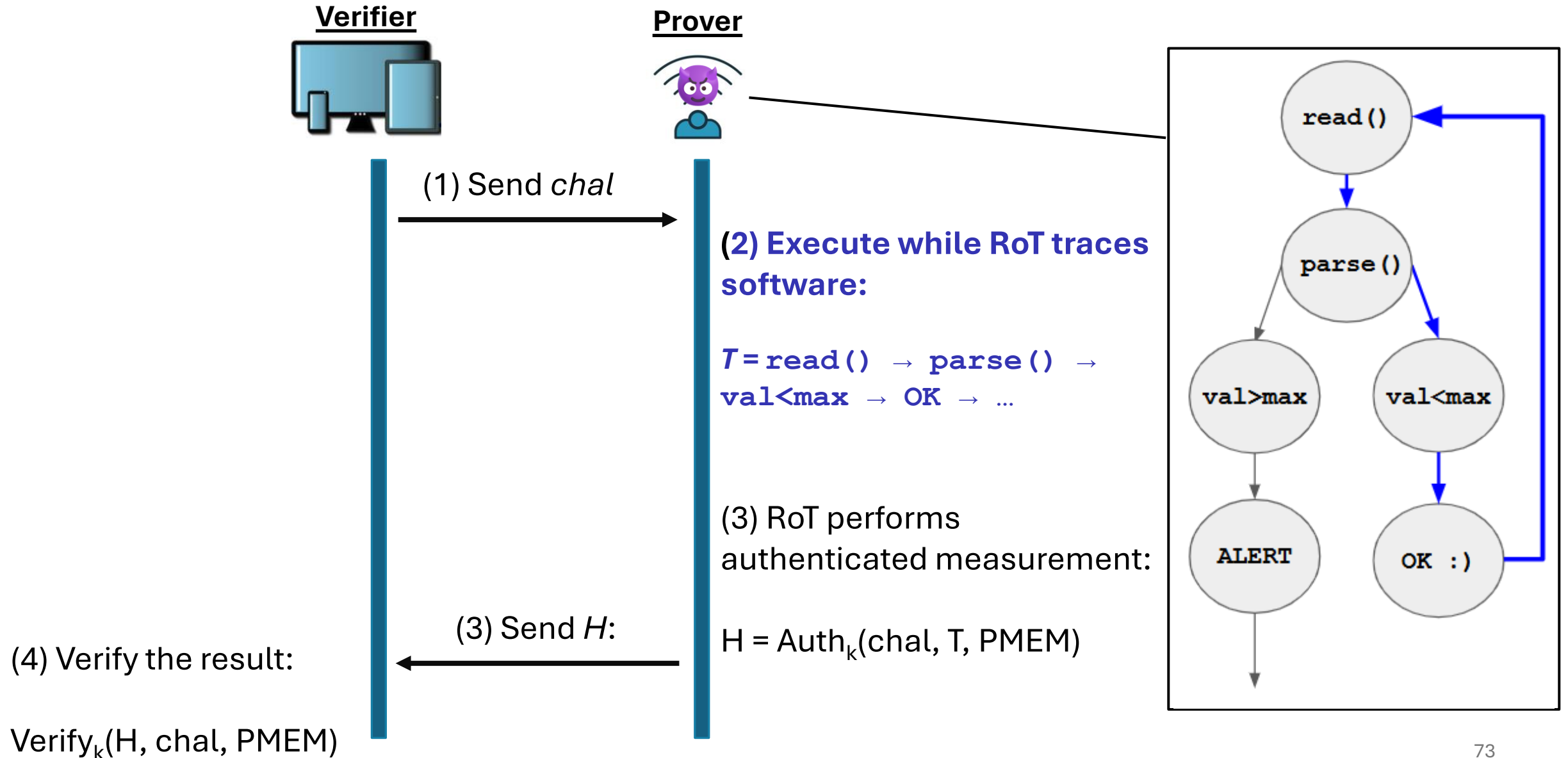
Run-time Attestation

- Require Prover to attest to
 - The correct system state (e.g., program is installed)
 - The system behaved at run-time in a valid way
- First proposed in **C-FLAT**
 - **Control Flow Attestation**
- C-FLAT: Requires an MCU Prover to attest to:
 - It is executing the correct software
 - It executed it following valid control flow paths

Control Flow Attestation

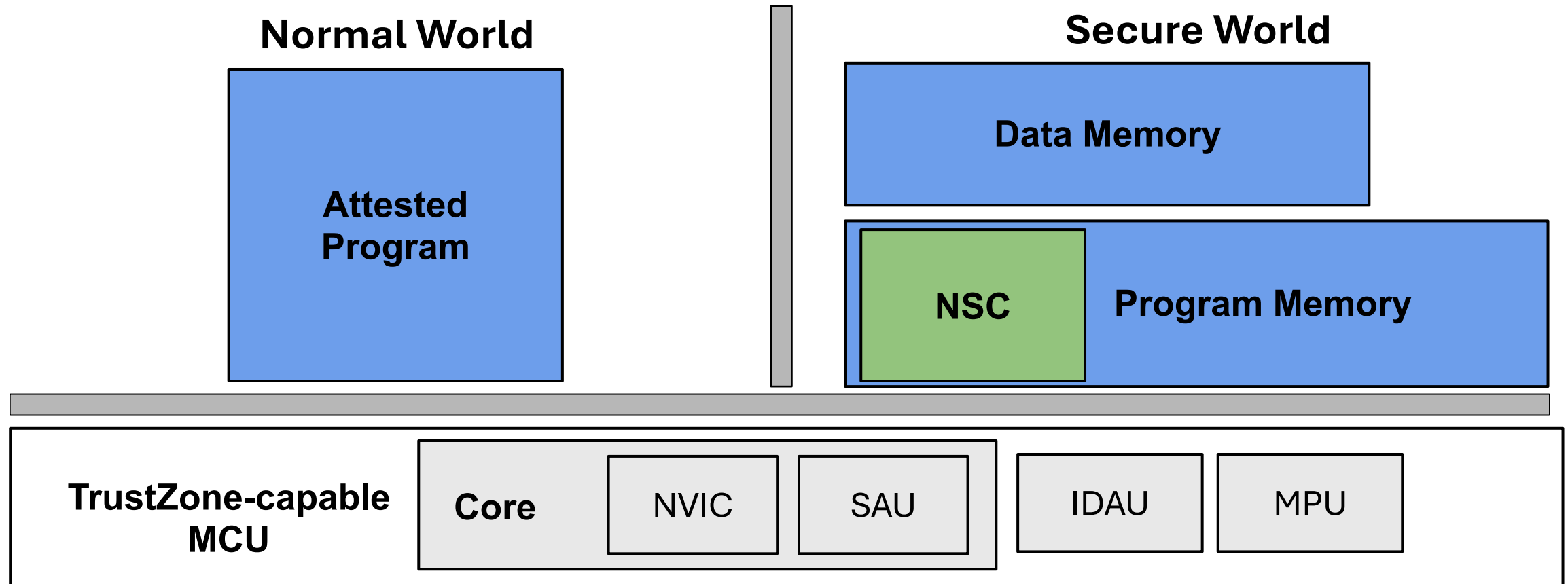


Control Flow Attestation



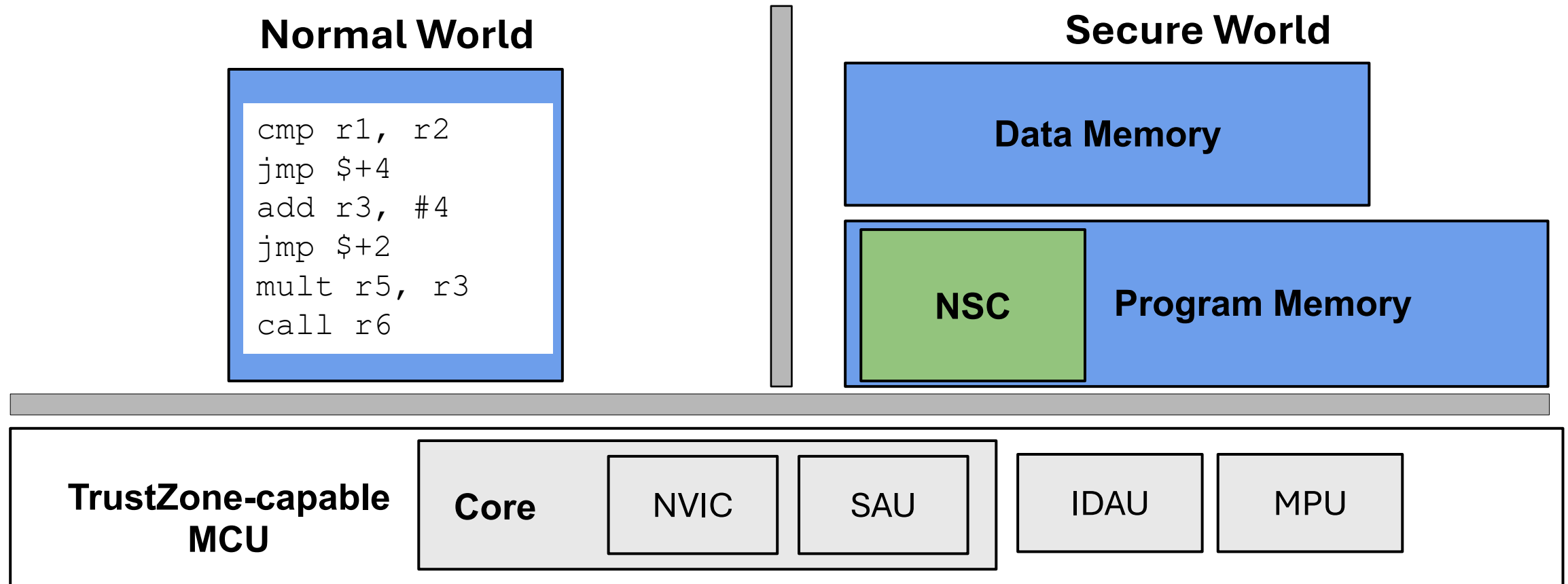
C-FLAT Approach

C-FLAT proposes a TrustZone-M based approach:



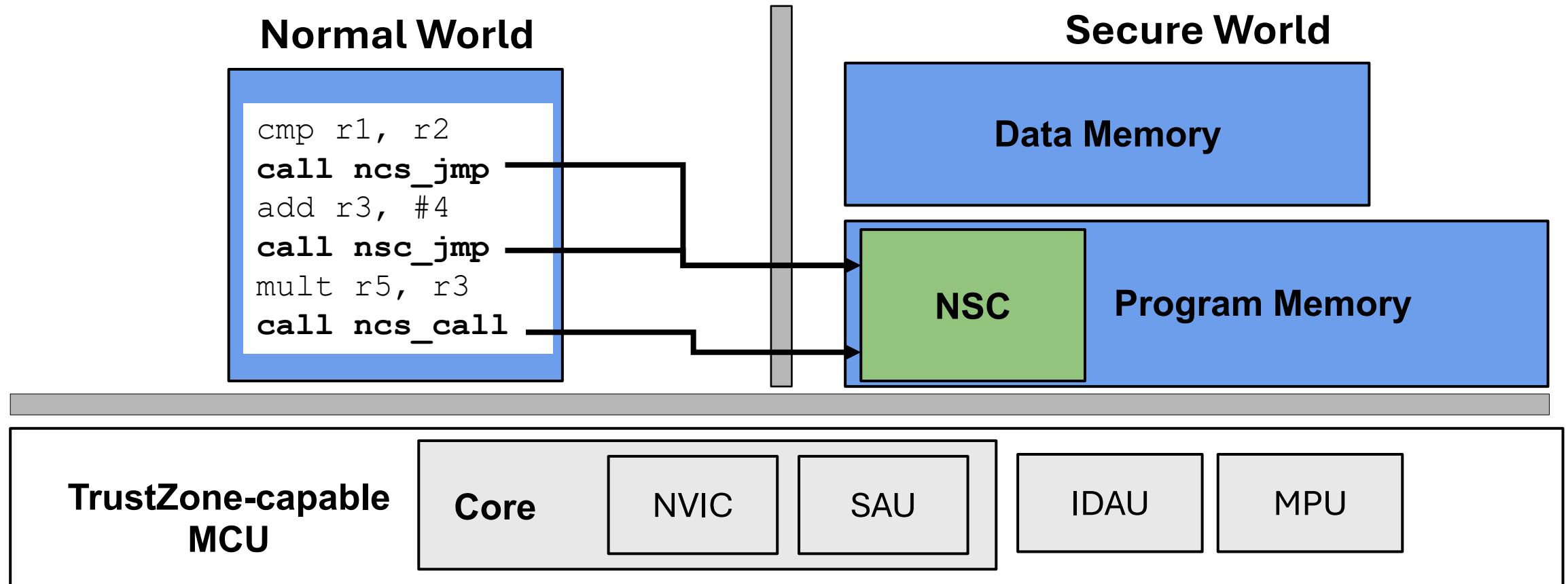
C-FLAT Approach

Before installing the program, first static analysis and instrumentation



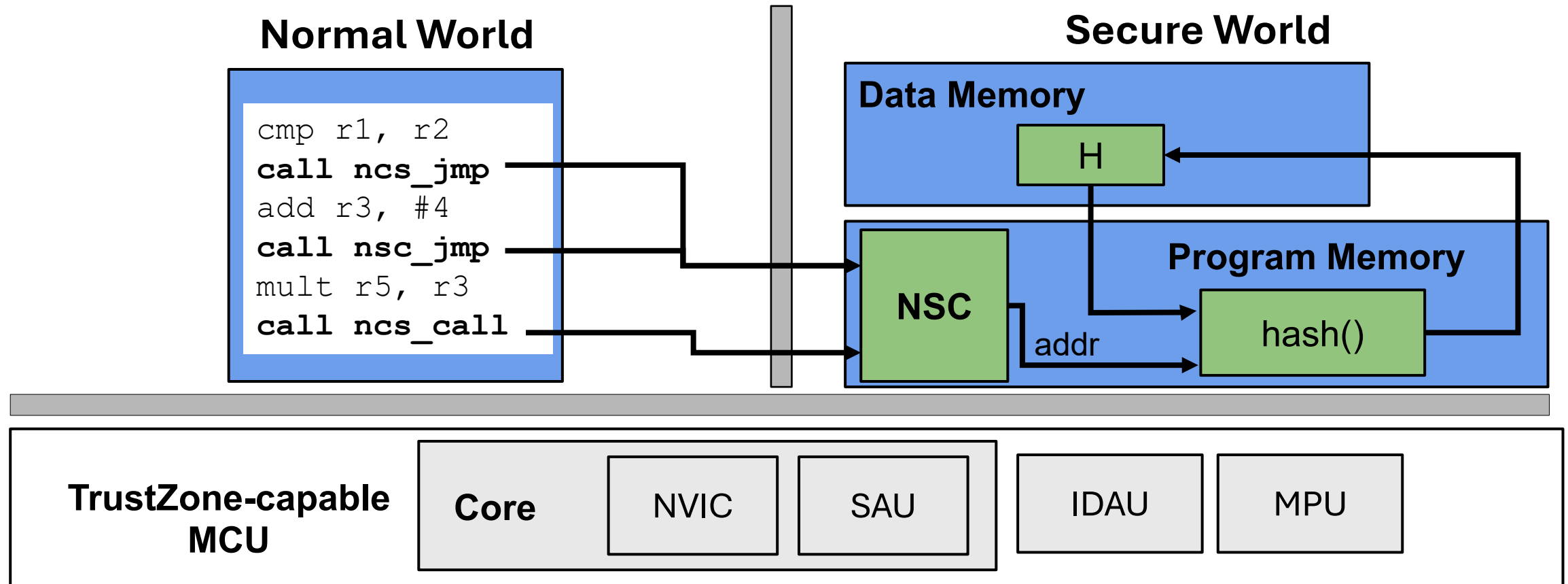
C-FLAT Approach

Every branch instruction is redirected to a NSC



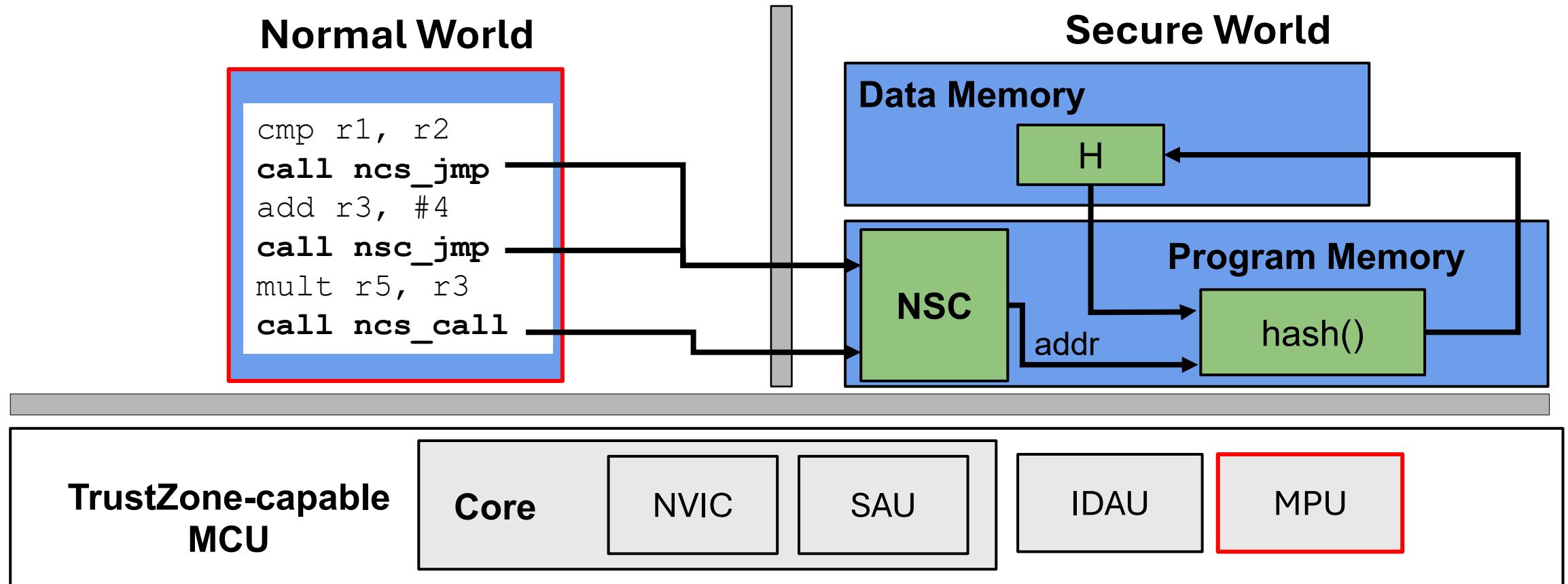
C-FLAT Approach

Once arrived inside the SW, compute running hash $H_i = \text{hash}(\text{addr}, H_{i-1})$



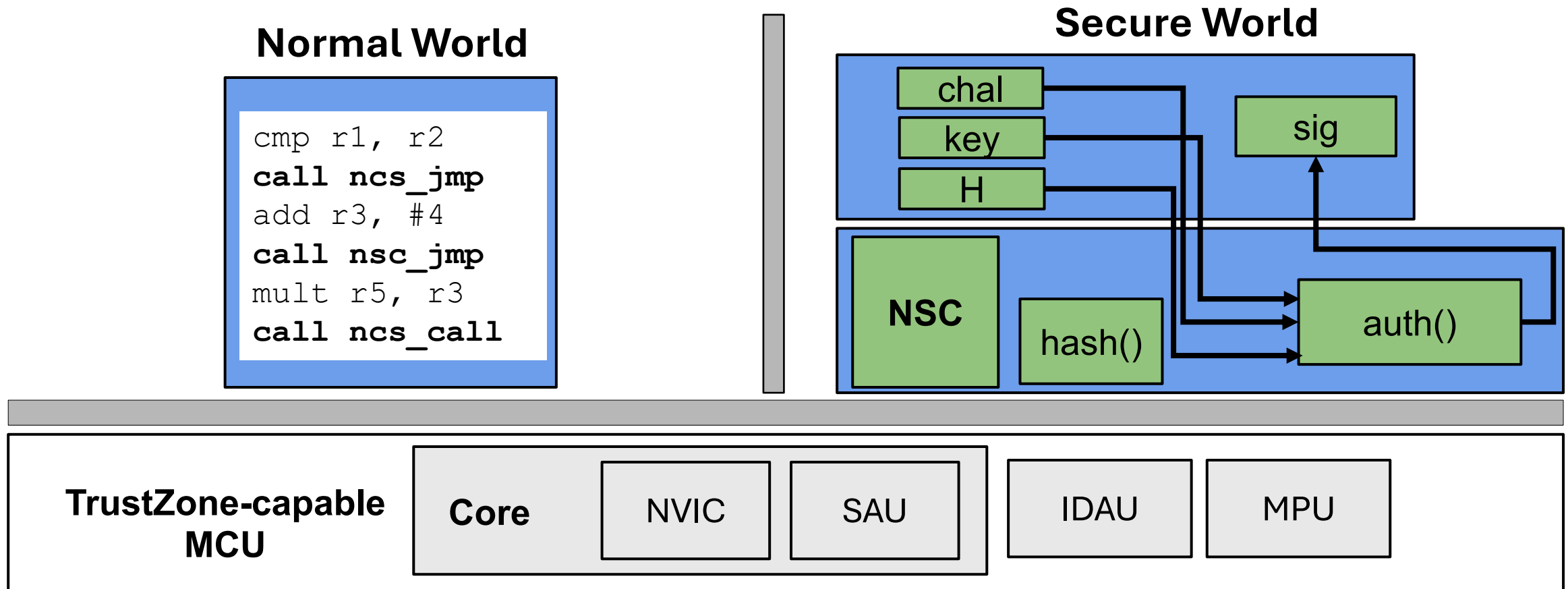
C-FLAT Approach

One note! This requires protecting the normal world app with the MPU



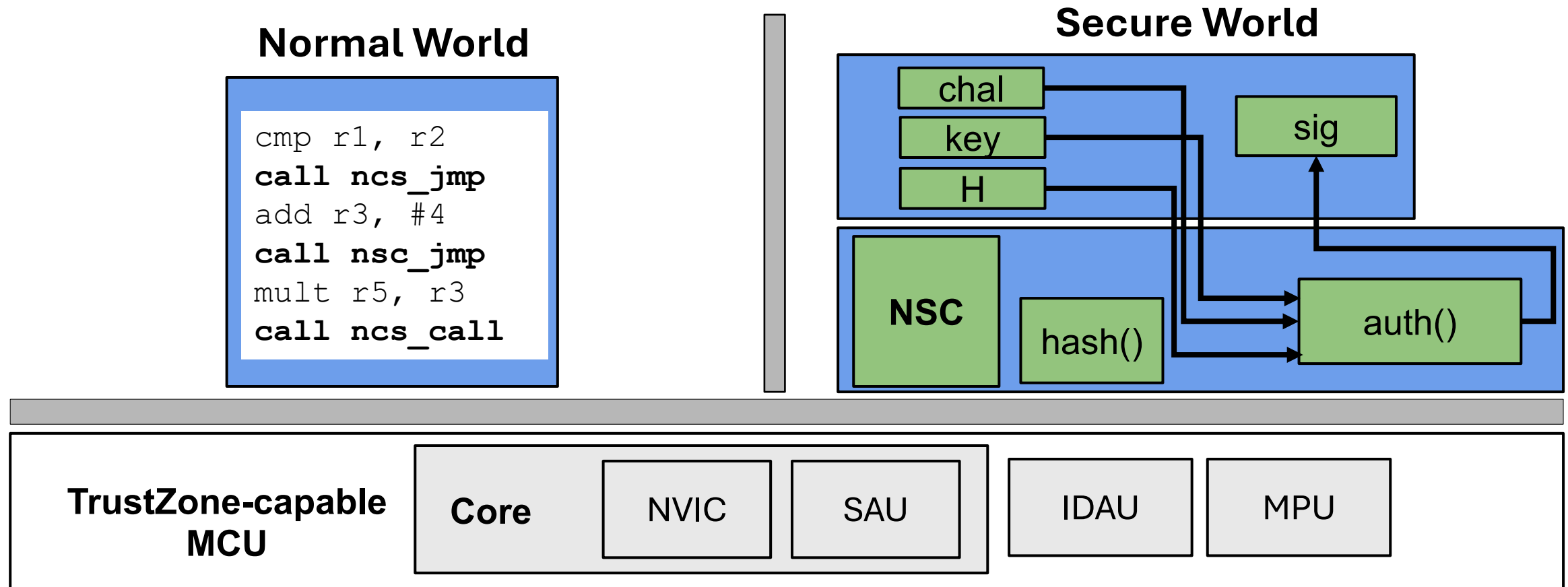
C-FLAT Approach

After execution has ended, attest by producing $\text{sig} = \text{auth}_k(\text{chal}, H)$

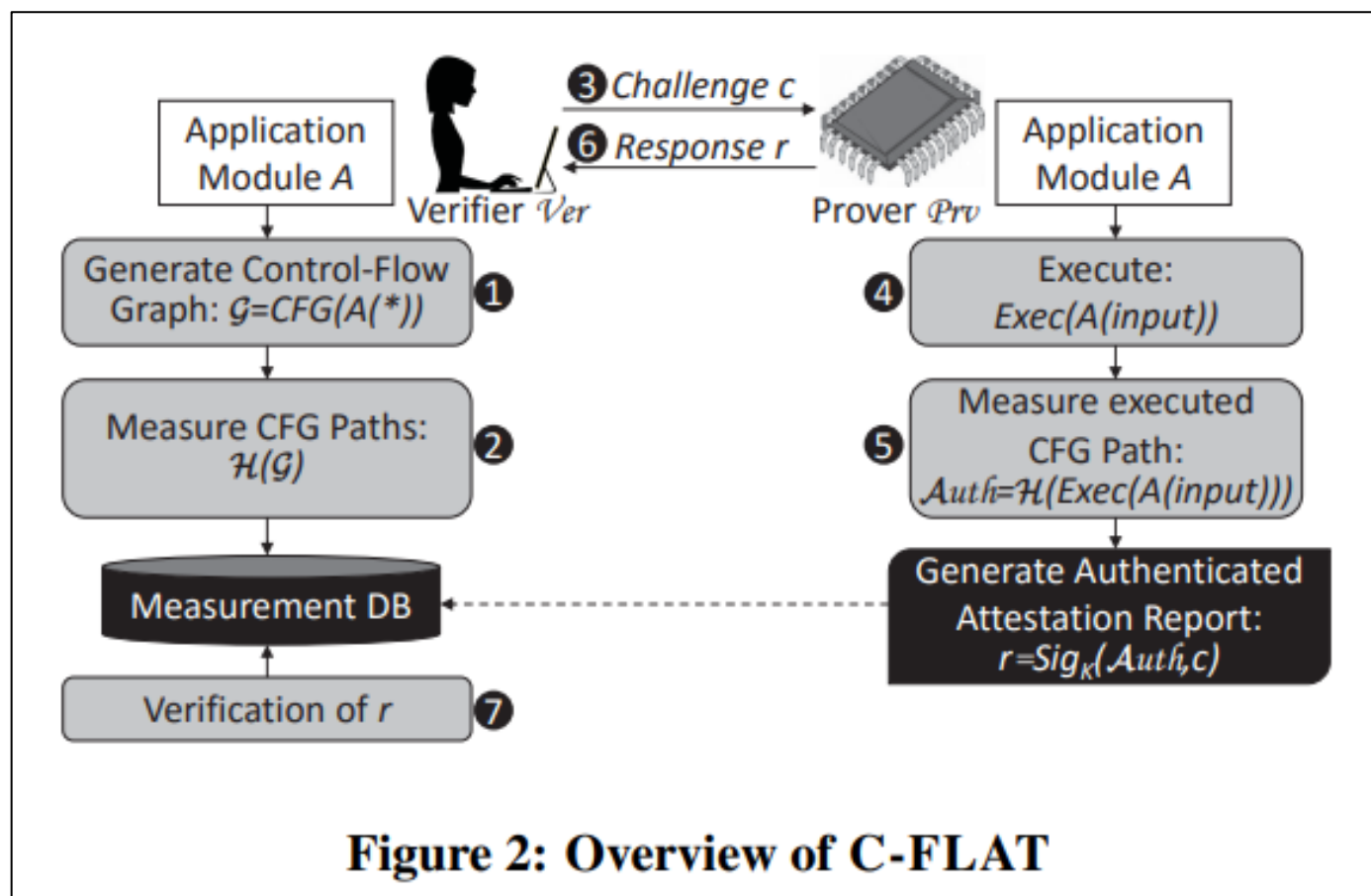


C-FLAT Approach

NOTE: Assumes Prover was installed with a key (MCU assumption)



More details on C-FLAT in the paper



Other Research in Systems and Software Security

Embedded Systems

- How does the system model change?
 - **Custom Hardware Extensions in Research** **Done!**
- What type of system-level support is available in today's devices?
 - **TrustZone in Cortex-M** **Done!**
- Availability mechanisms
 - How to build into a system? → **GAROTA** **Done!**
- Advancing attestation protocols
 - “Run-time” attestation → **C-FLAT** **Done!**

Concluding remarks...

Concluding remarks

1 Memory
Vulnerability

2 Integrity
Violation

3 Exploit
Payload

4 Exploit
Dispatch

5 Exploit
Execution

6 Attack

Concluding remarks

1

Memory
Vulnerability

2

Integrity
Violation

3

Exploit
Payload

4

Exploit
Dispatch

5

Exploit
Execution

6

Attack

Information leak

Malicious execution

Concluding remarks

1 Memory Vulnerability

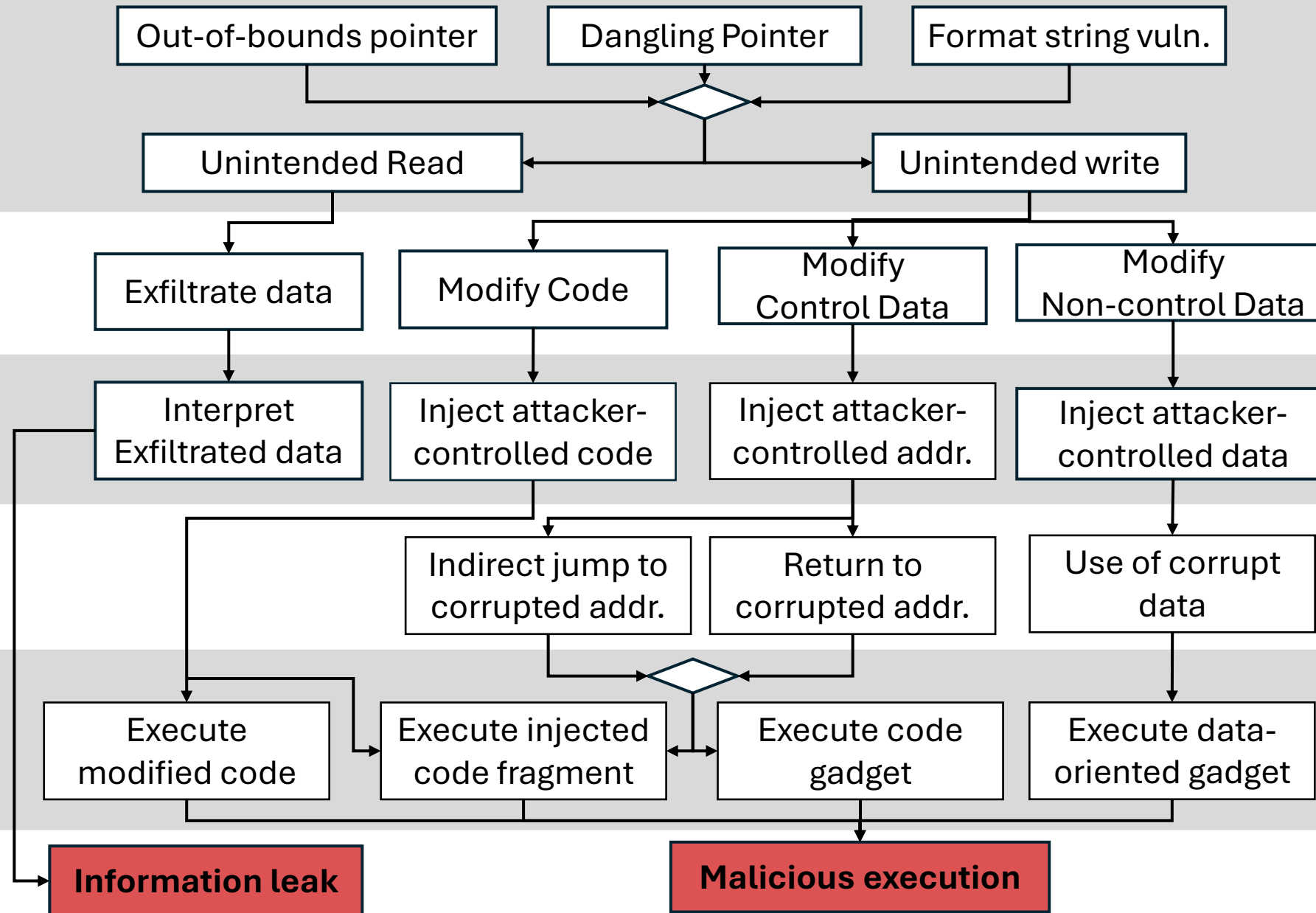
2 Integrity Violation

3 Exploit Payload

4 Exploit Dispatch

5 Exploit Execution

6 Attack



Concluding remarks

1 Memory Vulnerability

2 Integrity Violation

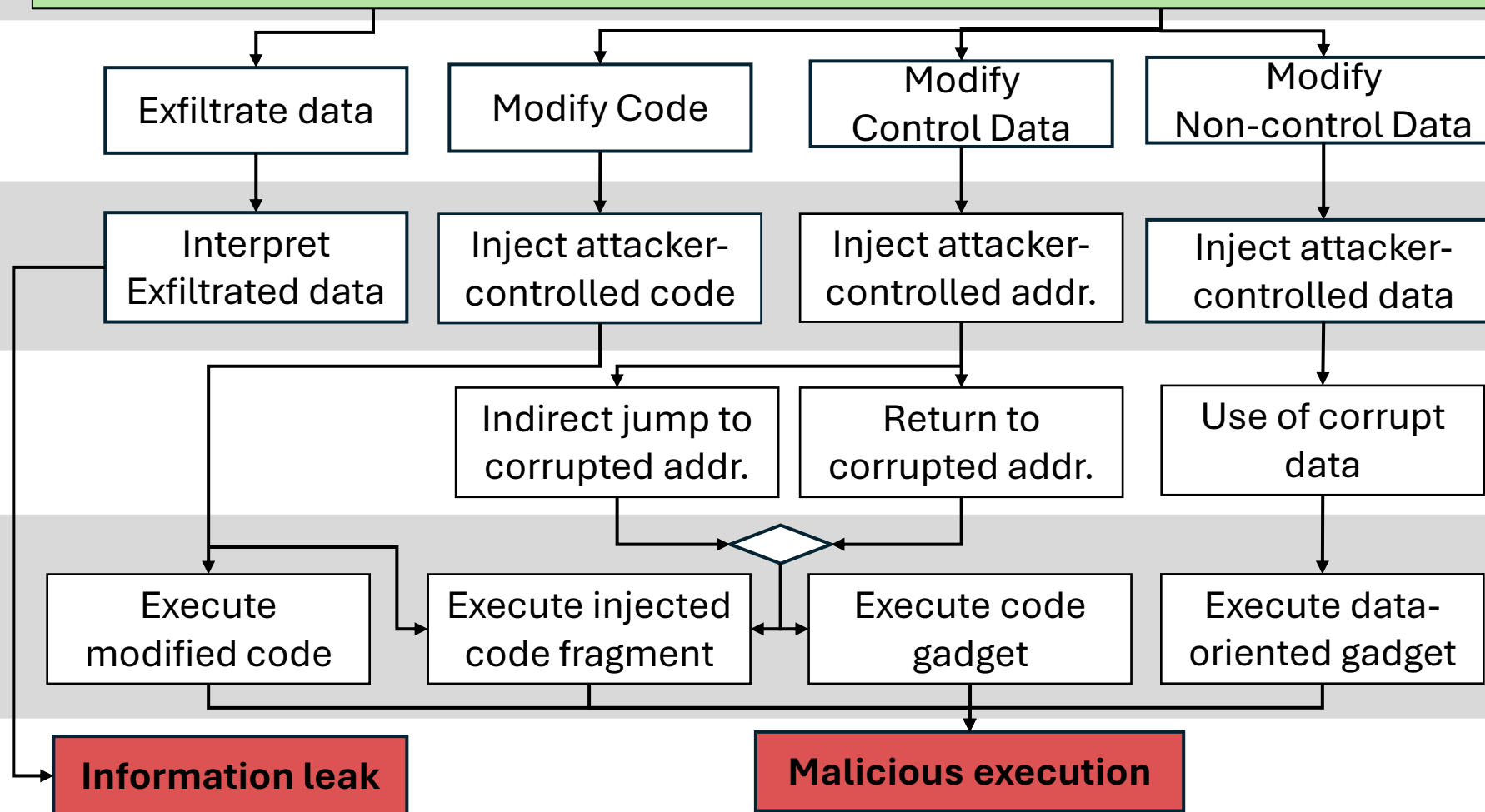
3 Exploit Payload

4 Exploit Dispatch

5 Exploit Execution

6 Attack

Software Testing: Fuzzing, symbolic exec., sanitizers
Memory safety: Static analysis, safe languages



Concluding remarks

1 Memory Vulnerability

Software Testing: Fuzzing, symbolic exec., sanitizers
Memory safety: Static analysis, safe languages

2 Integrity Violation

Software Compartmentalization: Code Integrity, Pointer integrity, Memory Management

3 Exploit Payload

Interpret Exfiltrated data

Inject attacker-controlled code

Inject attacker-controlled addr.

Inject attacker-controlled data

4 Exploit Dispatch

Indirect jump to corrupted addr.

Return to corrupted addr.

Use of corrupt data

5 Exploit Execution

Execute modified code

Execute injected code fragment

Execute code gadget

Execute data-oriented gadget

6 Attack

Information leak

Malicious execution

Concluding remarks

1 Memory Vulnerability

Software Testing: Fuzzing, symbolic exec., sanitizers
Memory safety: Static analysis, safe languages

2 Integrity Violation

Software Compartmentalization: Code Integrity, Pointer integrity, Memory Management

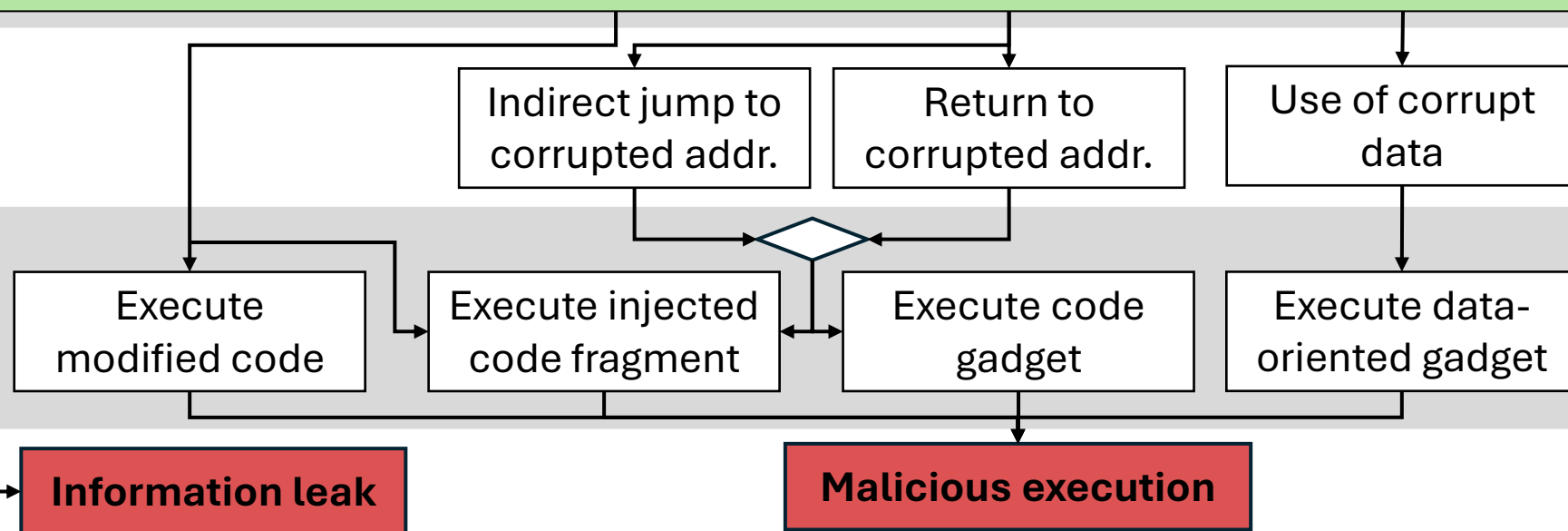
3 Exploit Payload

Software Diversification: ASLR, ISR, DSR

4 Exploit Dispatch

5 Exploit Execution

6 Attack



Concluding remarks

1 Memory Vulnerability

Software Testing: Fuzzing, symbolic exec., sanitizers
Memory safety: Static analysis, safe languages

2 Integrity Violation

Software Compartmentalization: Code Integrity, Pointer integrity, Memory Management

3 Exploit Payload

Software Diversification: ASLR, ISR, DSR

4 Exploit Dispatch

Run-time Integrity: Control Flow Integrity, Data flow integrity

5 Exploit Execution

Execute modified code

Execute injected code fragment

Execute code gadget

Execute data-oriented gadget

6 Attack

Information leak

Malicious execution

Concluding remarks

1 Memory Vulnerability

Software Testing: Fuzzing, symbolic exec., sanitizers
Memory safety: Static analysis, safe languages

2 Integrity Violation

Software Compartmentalization: Code Integrity, Pointer integrity, Memory Management

3 Exploit Payload

Software Diversification: ASLR, ISR, DSR

4 Exploit Dispatch

Run-time Integrity: Control Flow Integrity, Data flow integrity

5 Exploit Execution

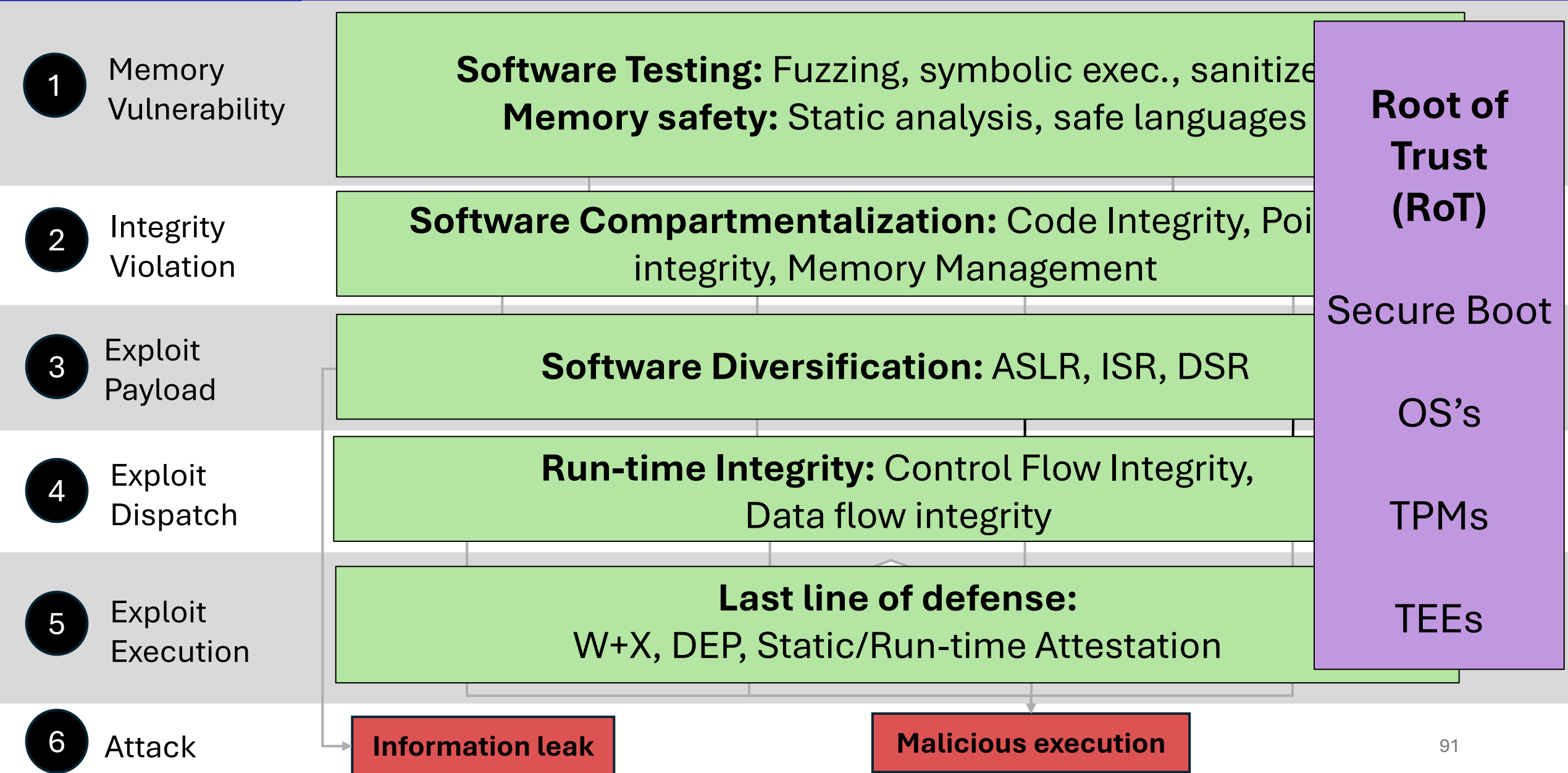
Last line of defense:
W+X, DEP, Static/Run-time Attestation

6 Attack

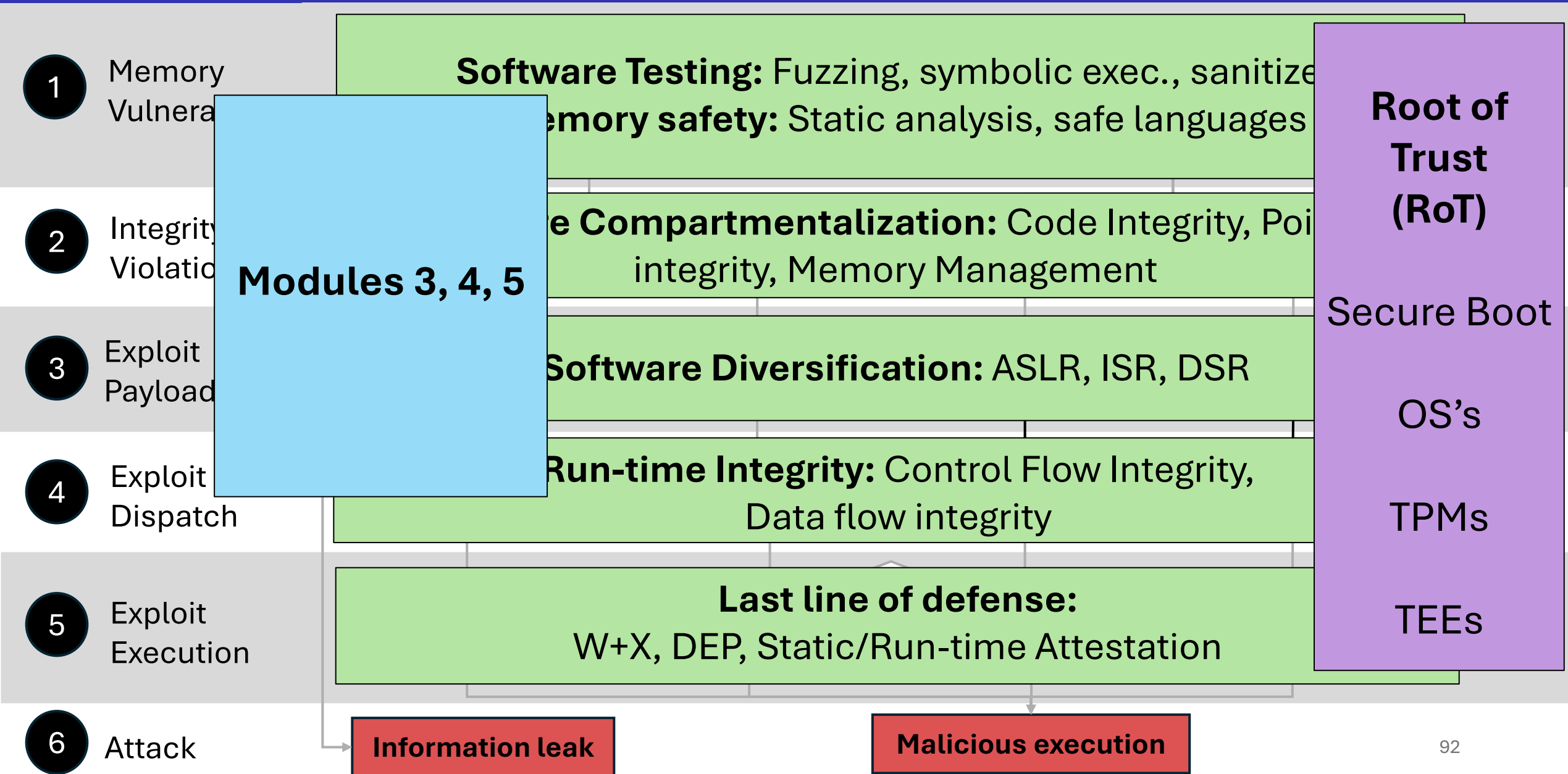
Information leak

Malicious execution

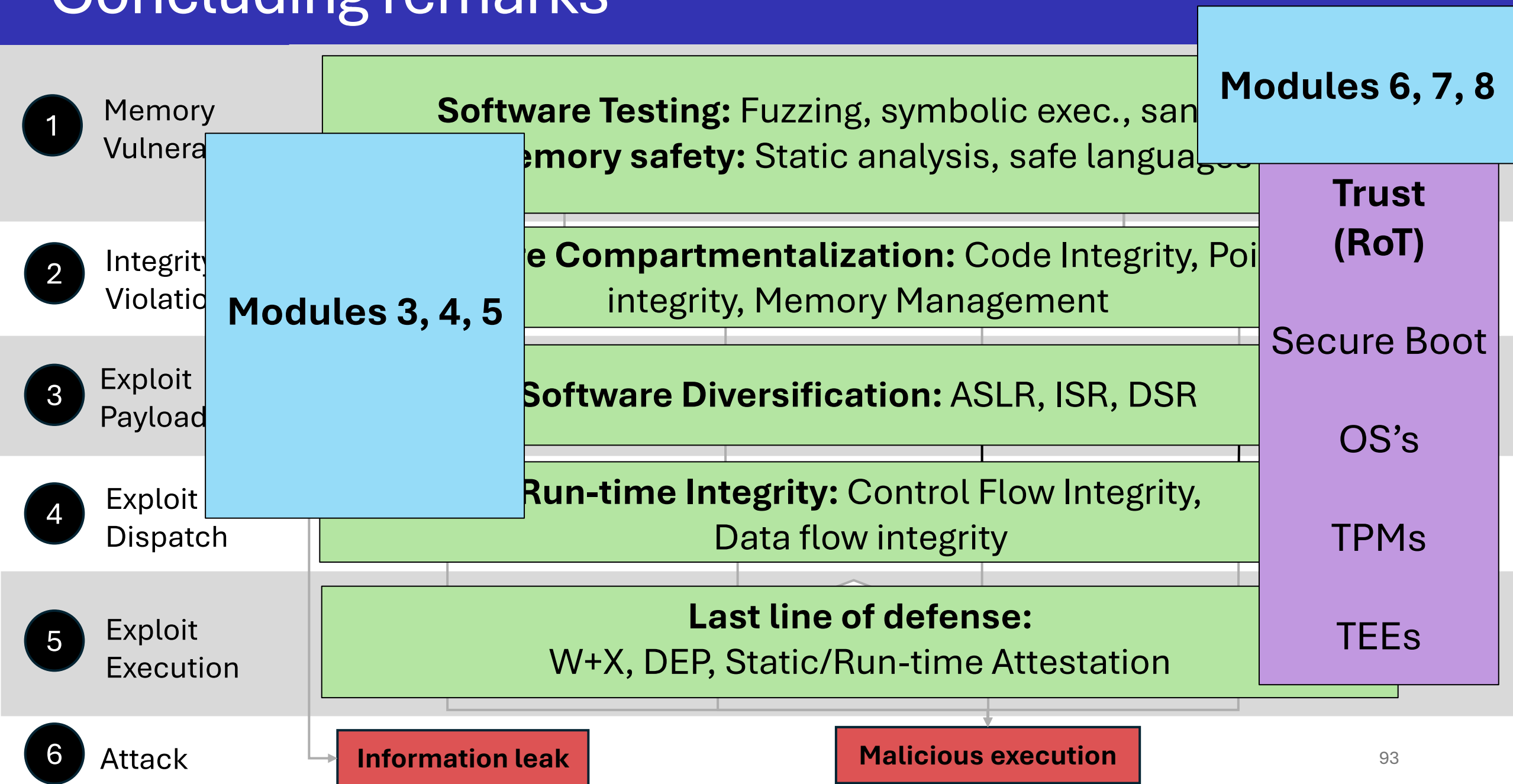
Concluding remarks



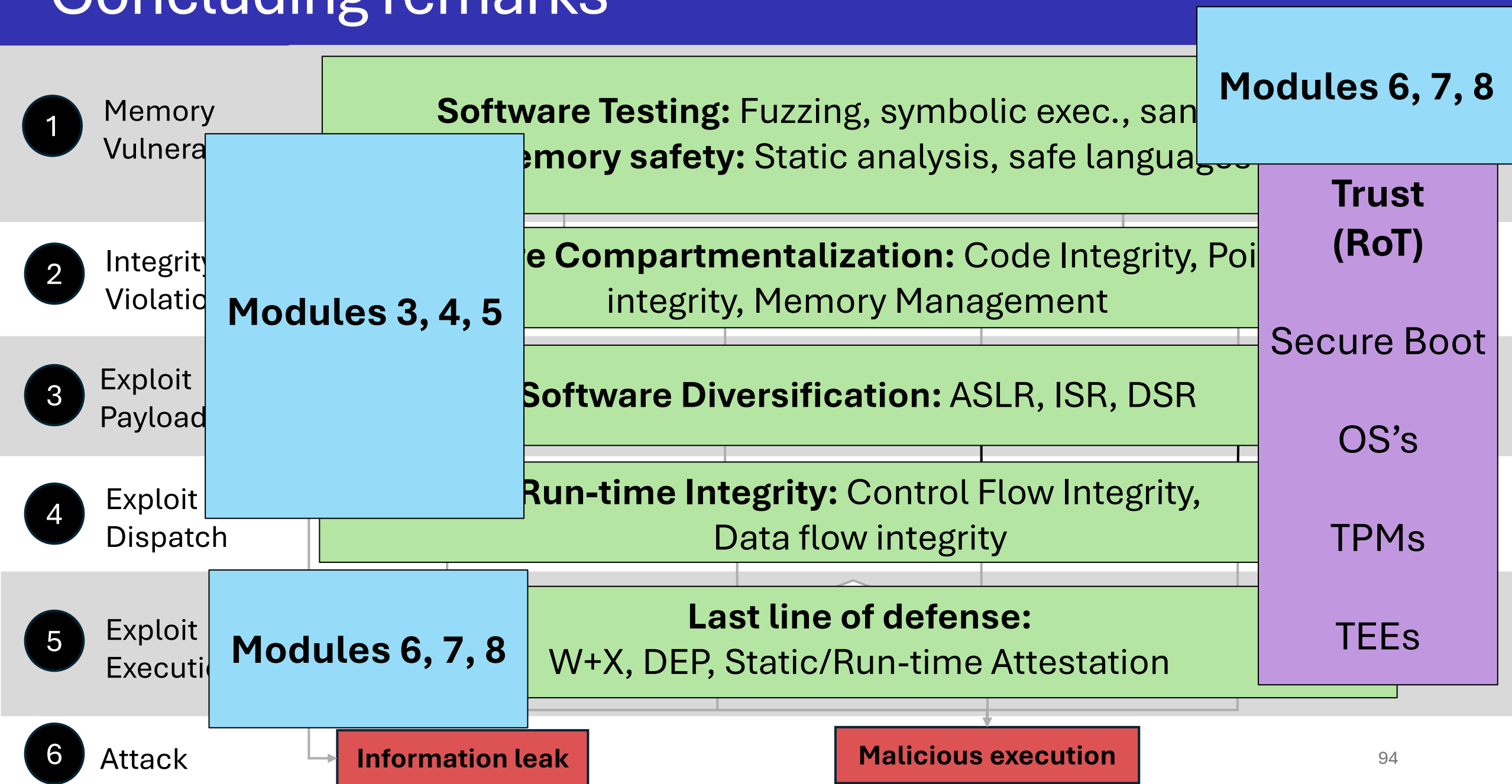
Concluding remarks



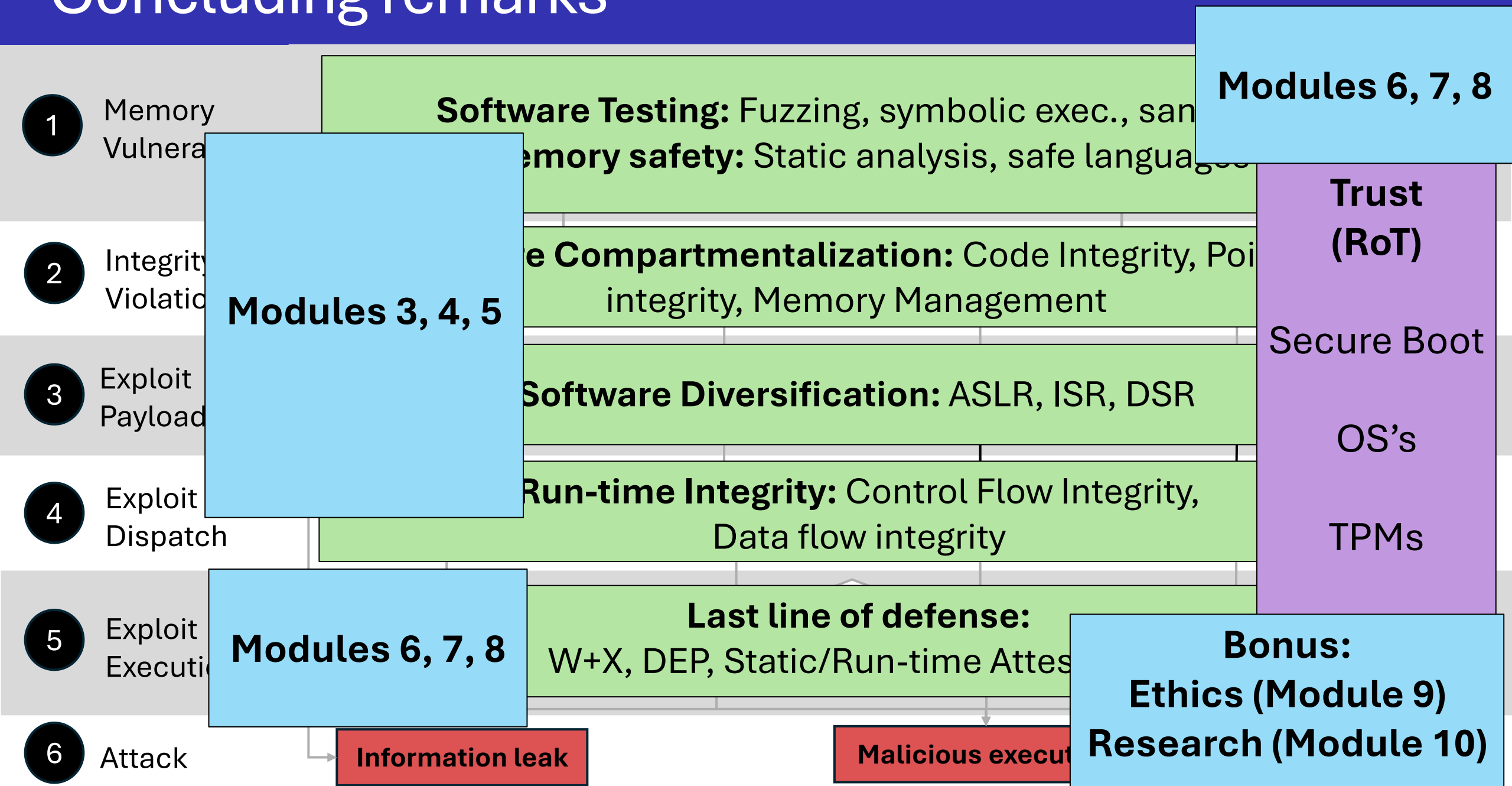
Concluding remarks



Concluding remarks



Concluding remarks



Concluding thoughts...

What is Software and System Security?

Mechanisms combining software AND roots of trust to:

- Detect memory vulnerabilities via software testing and memory safety
- Prevent integrity violations via compartmentalization, access control, memory management
- Prevent exploiting vulnerabilities with software diversification
- Prevent dispatching of payloads via run-time defenses
- Prove/ensure execution itself is valid

Concluding thoughts...

What is Software and System Security?

Mechanisms combining software AND roots of trust to:

- Detect memory vulnerabilities via software testing and memory safety
- Prevent integrity violations via compartmentalization, access control, memory management
- Prevent exploiting vulnerabilities with software diversification
- Prevent dispatching of payloads via run-time defenses
- Prove/ensure execution itself is valid

Thank you for a great term! Wish you all the best!

That's all for today!

Resources:

- [SONY Press Centre \(UK\)](#)
- [PULPino](#)
- [PULPissimo](#)
- [Ibex](#)
- [openMSP430](#)
- [MSP430 IPE](#)
- [TrustZone-M Basics](#)
- [GAROTA](#)
- [C-FLAT](#)

