

CS 453/698: Software and Systems Security

Module: Hardware & Mobile Security

Lecture: Android & ARM TrustZone

Adam Caulfield

University of Waterloo

Spring 2025

Reminders & Recap

Reminders:

- [A4 is released](#)
 - Due July 25th
- Send your research project proposals to Meng and me!

Recap – last time we covered:

Intro to Trusted Execution Environments (TEE)

- Separate, isolated, and minimal execution environment
- Enabled as a part of the CPU architecture itself (not a separate external module)

Intel SGX

- User-space TEE → *enclaves*
- Architecture details → Isolation, life cycle, address translation, attestation

Today

Continue: Hardware and Mobile Security

Different TEE architecture and real-world use case

ARM TrustZone

- System-level or “split world” architecture
- Overview
- Architectural details

Android OS

- How it uses TrustZone (particular focus on KeyStore)
- Other security features

ARM Processors

A few family of CPUs provided by ARM

ARM Cortex-A family

- Application processors
- Supports OS and high-performance apps
- This is the CPU in smart phones, smart tv

ARM Cortex-R family

- Real-time processors with high-performance and reliability
- Support real-time processing and mission-critical control

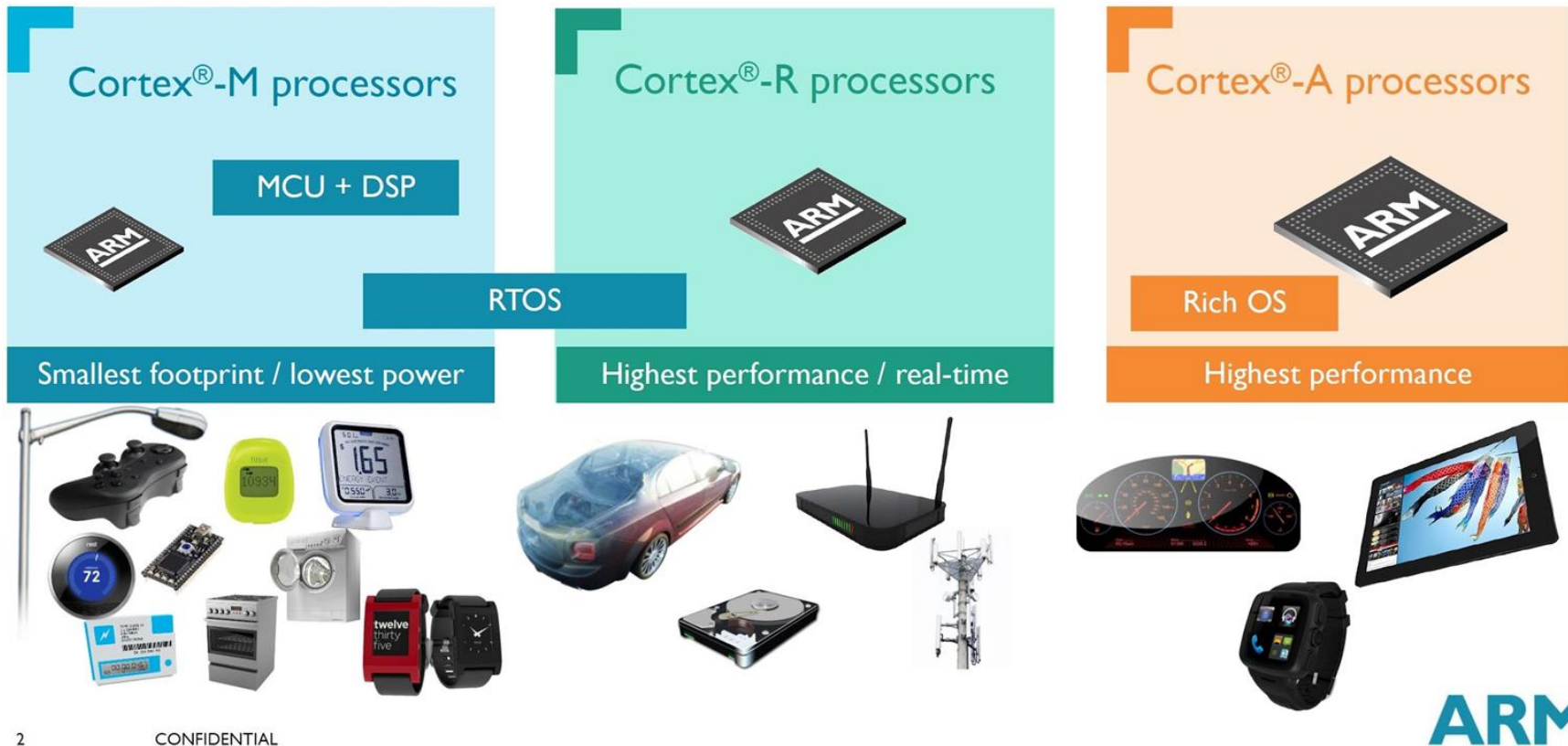
ARM Cortex-M family

- Microcontrollers
- Cost-sensitive, SoC, low-power processing

ARM Processors

A few family of CPUs provided by ARM

ARM® Cortex® Processors across the Embedded Market



ARM Processors

A few family of CPUs provided by ARM

ARM® Cortex® Processors across the Embedded Market

Cortex®-M processors

MCU + DSP

RTOS

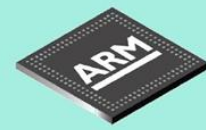
Smallest footprint / lowest power




2 CONFIDENTIAL

Later... (Research lecture)

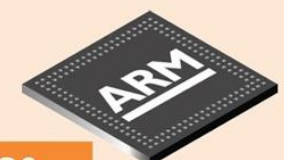
Cortex®-R processors



Highest performance / real-time




Cortex®-A processors



Rich OS

Highest performance



ARM

Covered Today!

ARM TrustZone Overview

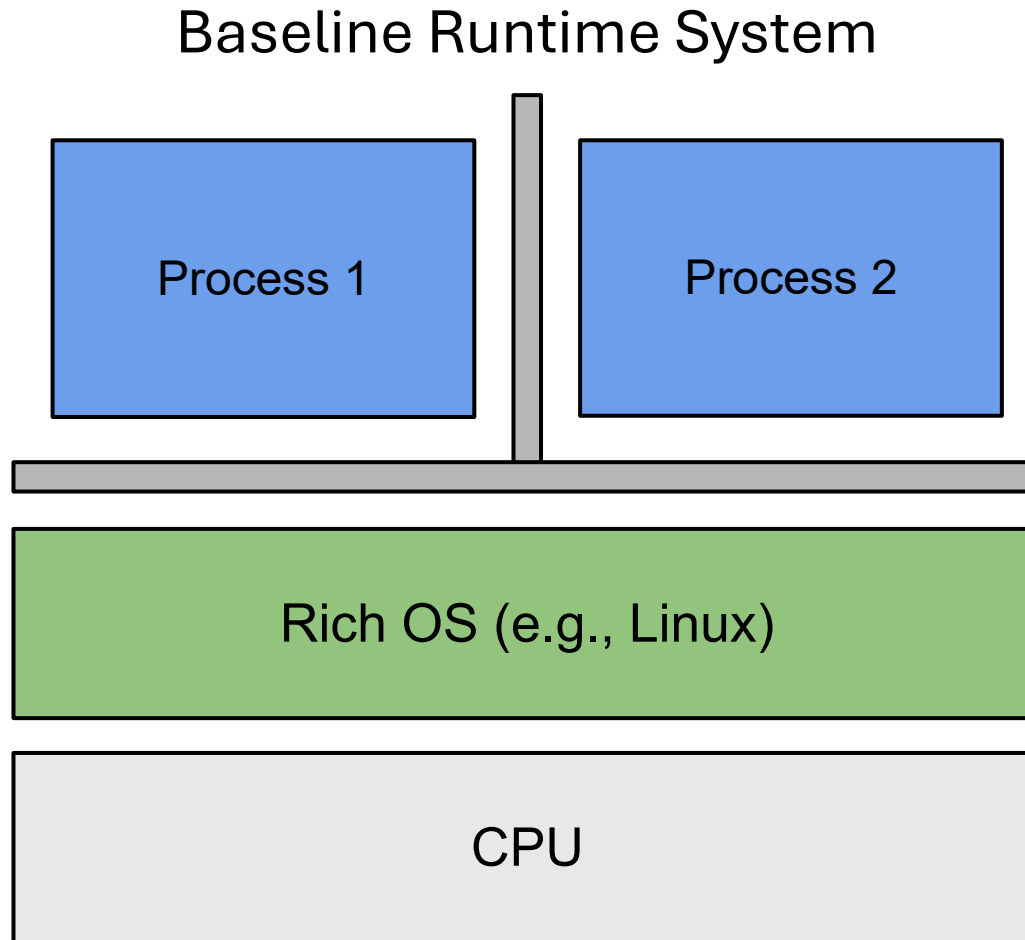
What is TrustZone?

ARM Processors' TEE

- Splits the system into two worlds
- From ARM:
 - “The security of the system is achieved by partitioning all (...) hardware and software resources so that **they exist in one of two worlds – the Secure world for the security subsystem,** and the **Normal world for everything else.**”

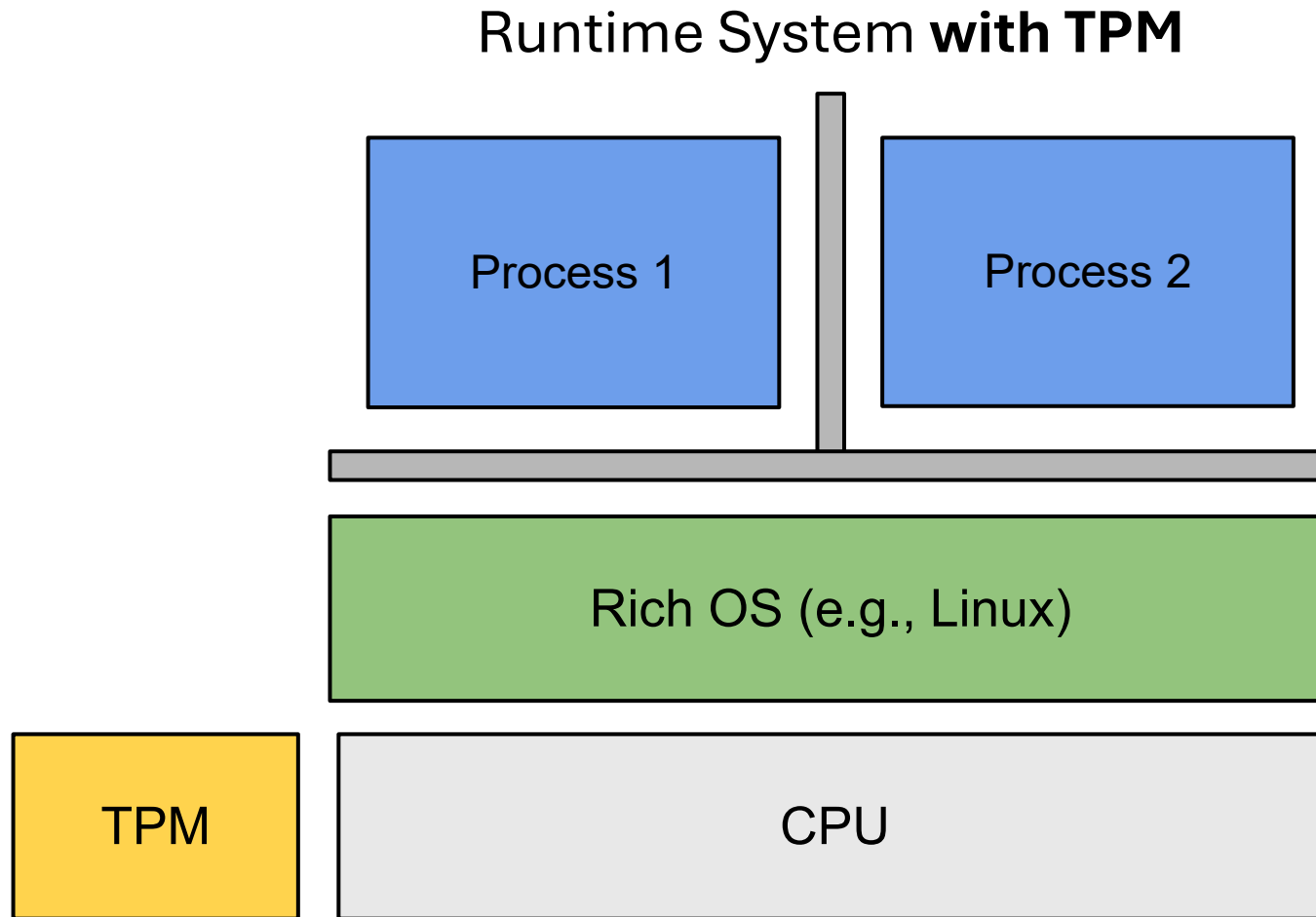
ARM TrustZone Overview

Some visualizations....



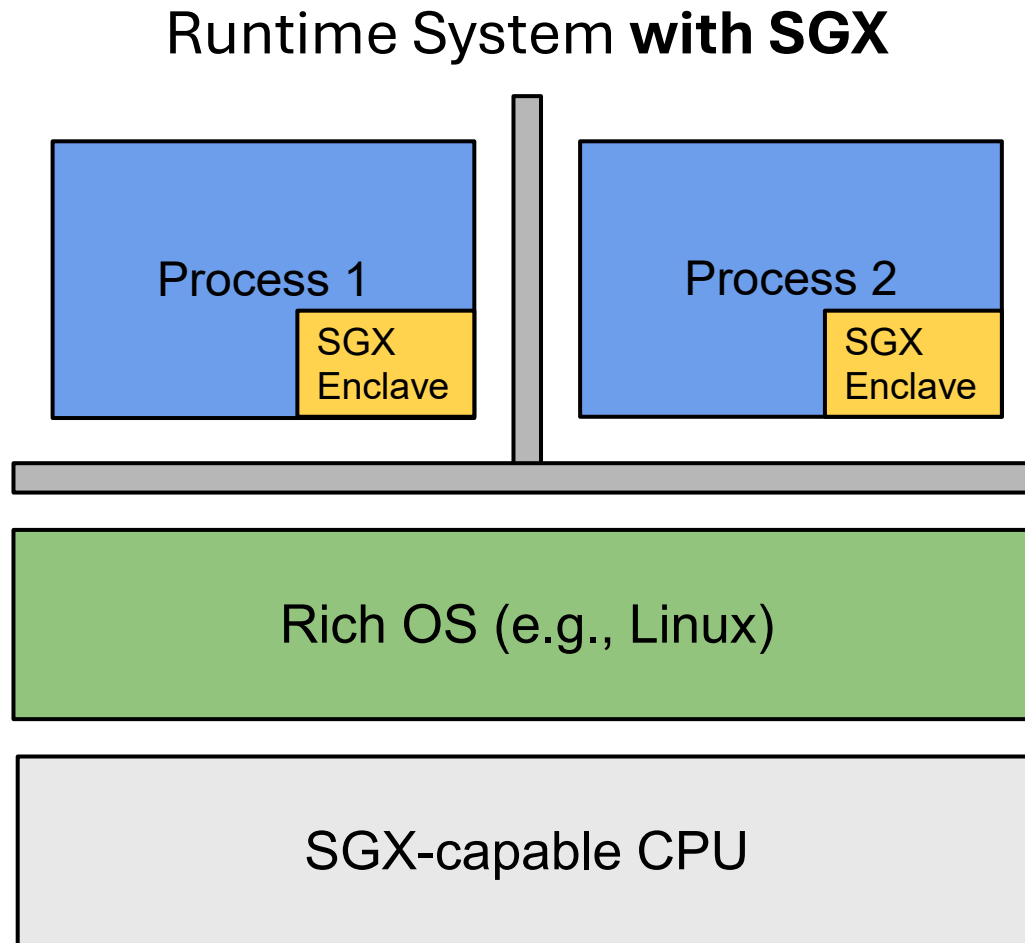
ARM TrustZone Overview

Some visualizations....



ARM TrustZone Overview

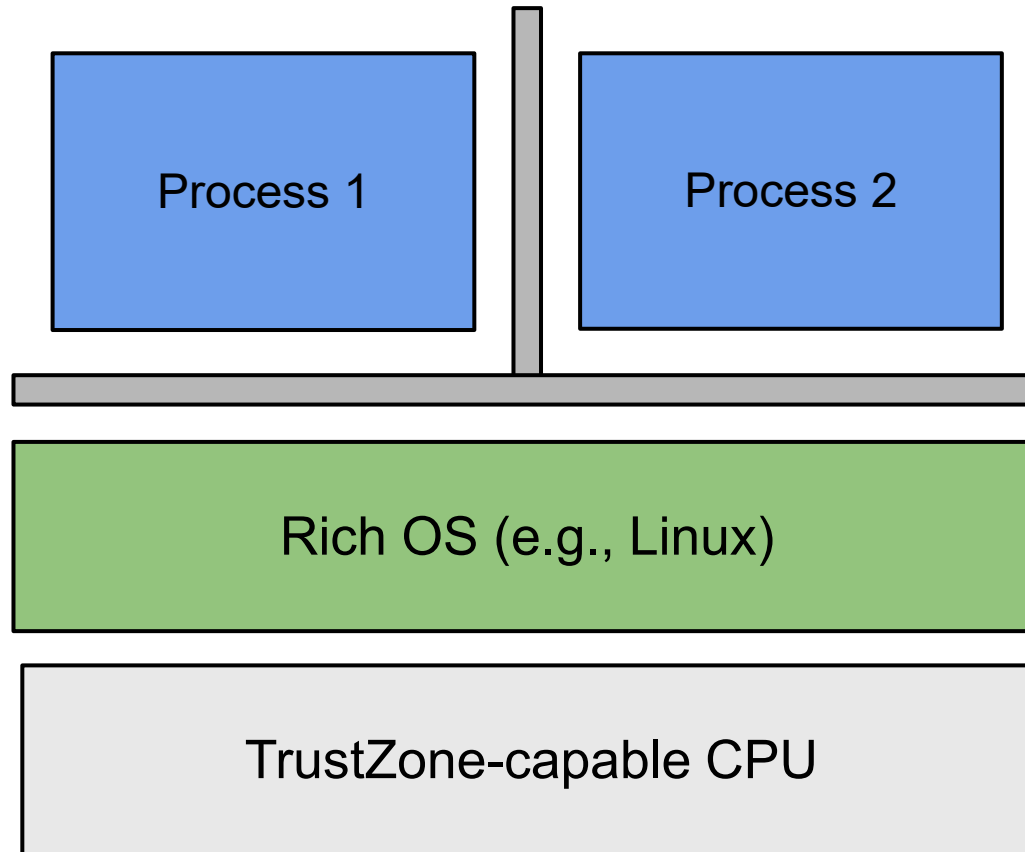
Some visualizations....



ARM TrustZone Overview

Some visualizations....

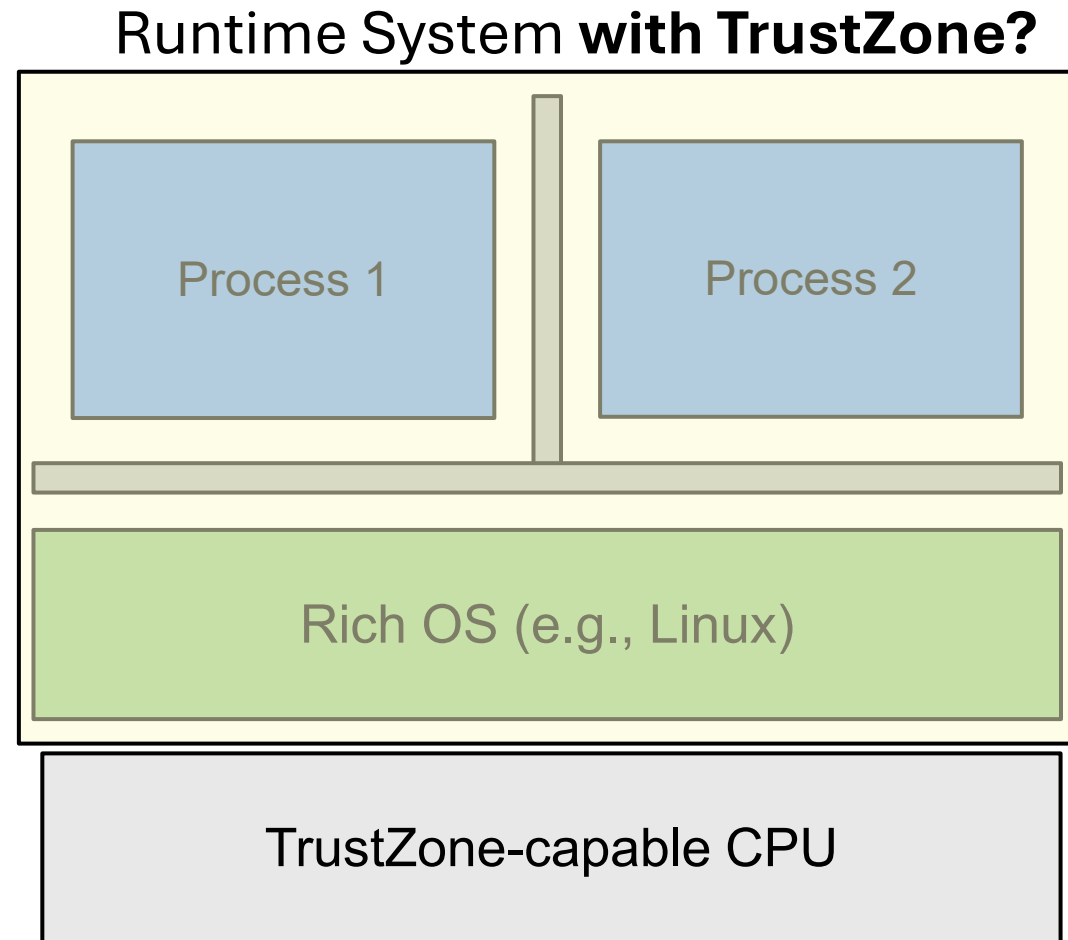
Runtime System **with TrustZone?**



ARM TrustZone Overview

Some visualizations....

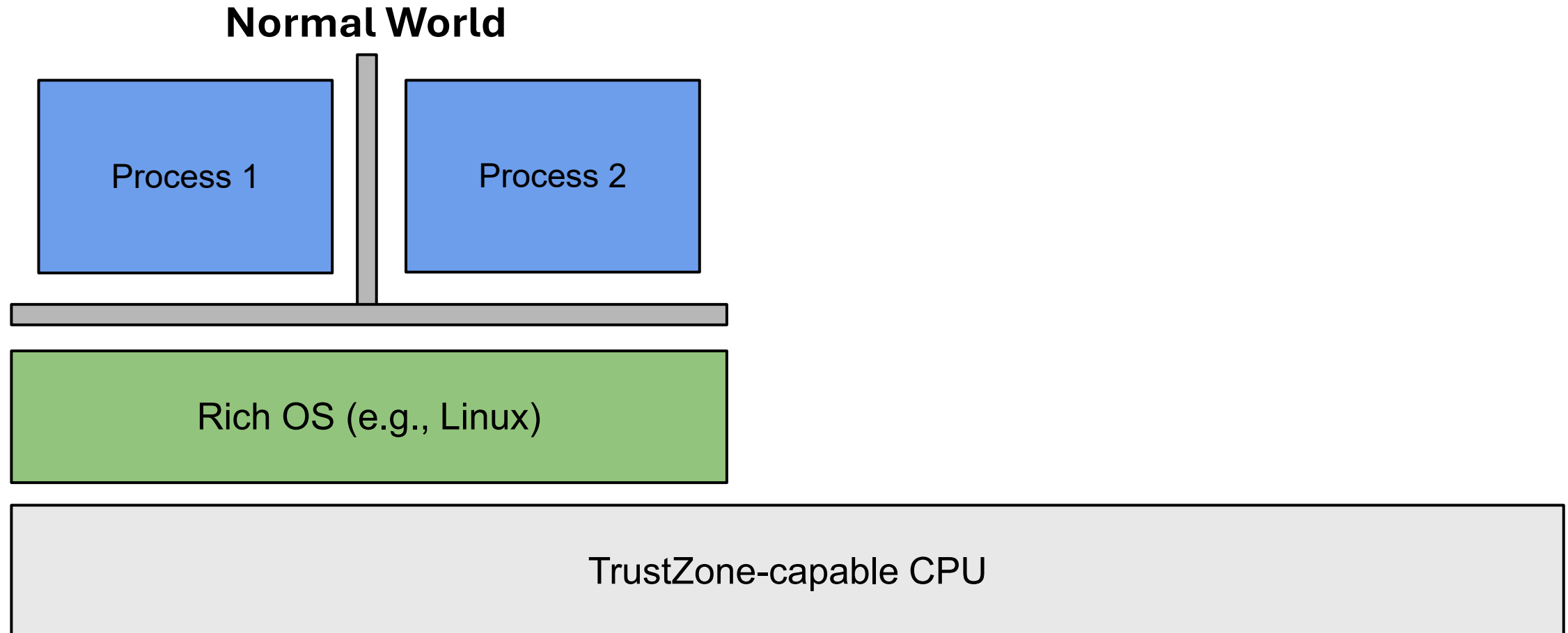
Split everything
between two
worlds



ARM TrustZone Overview

Some visualizations....

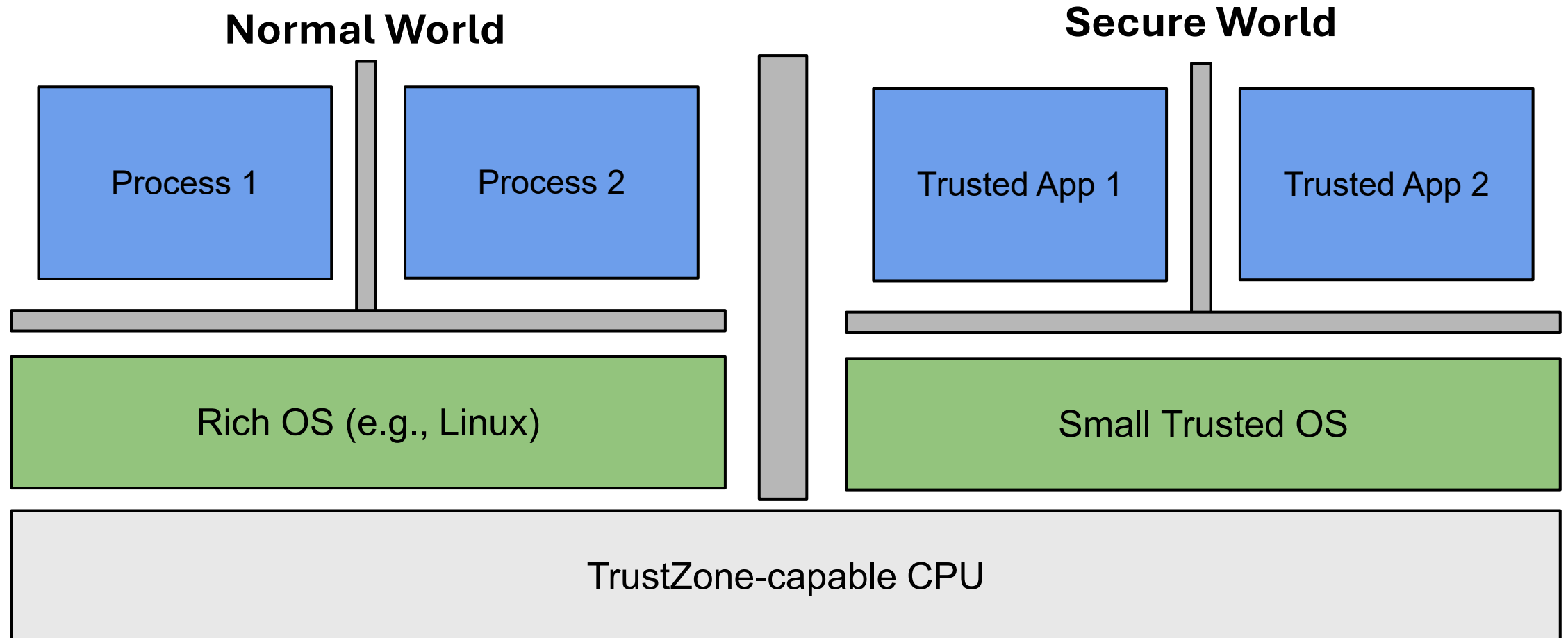
Runtime System **with TrustZone?**



ARM TrustZone Overview

Some visualizations....

Runtime System **with TrustZone?**



ARM TrustZone Overview

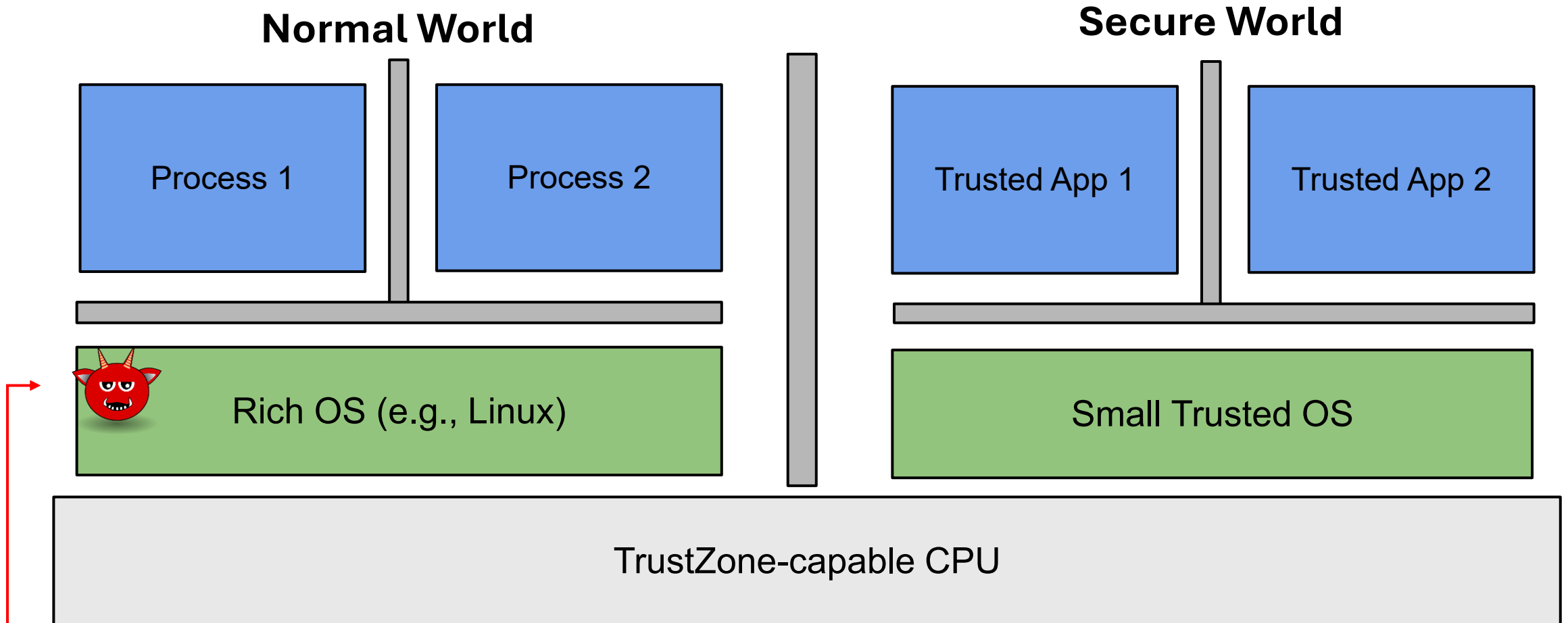
TrustZone's Guarantee:

Even if the Normal World is fully compromised, the Secure World remains safe, confidential, isolated, etc.

ARM TrustZone Overview

Some visualizations....

Runtime System **with TrustZone?**

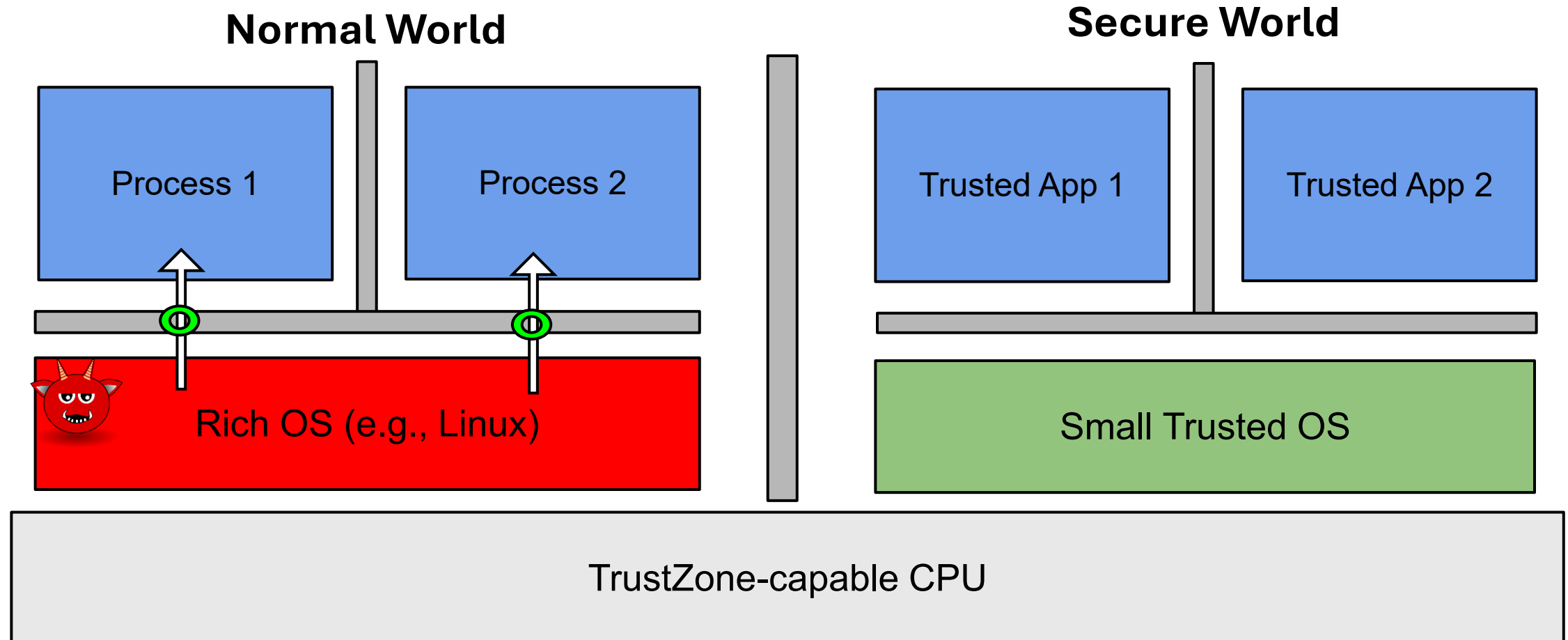


What happens now?

ARM TrustZone Overview

Some visualizations....

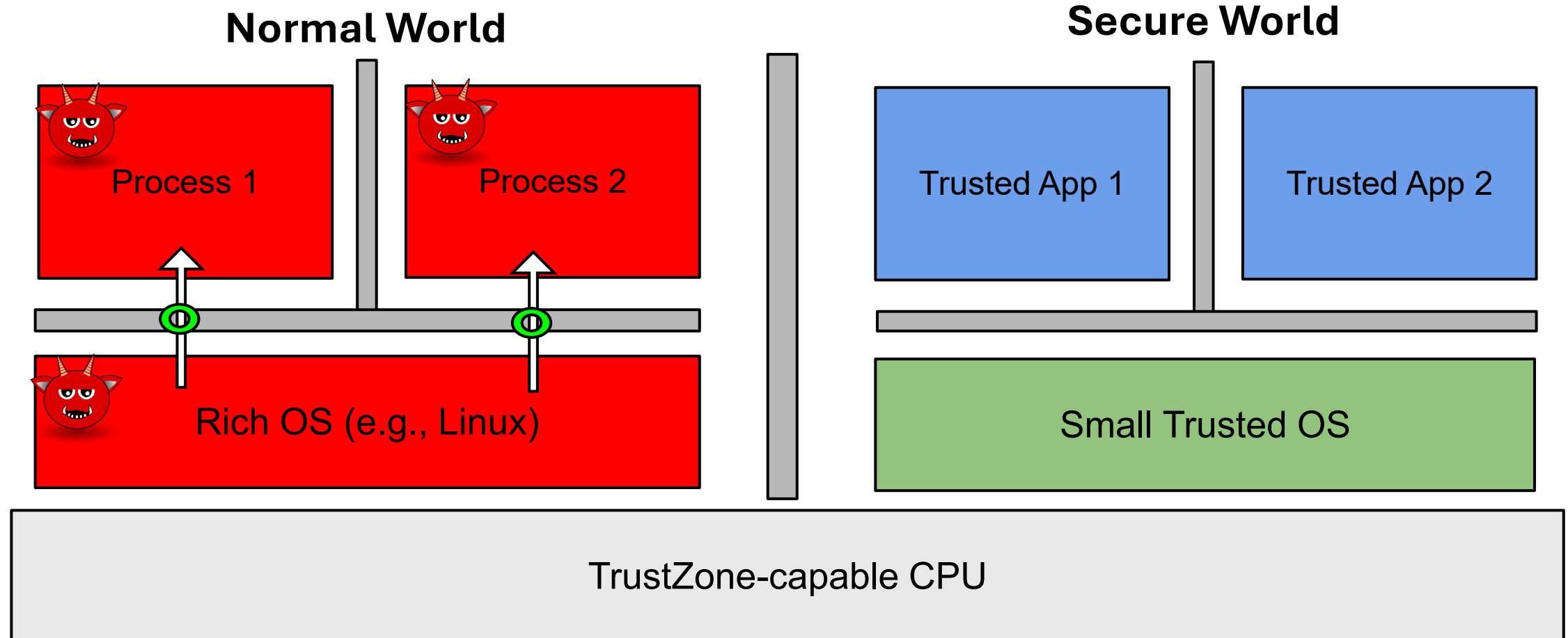
Runtime System **with TrustZone?**



ARM TrustZone Overview

Some visualizations....

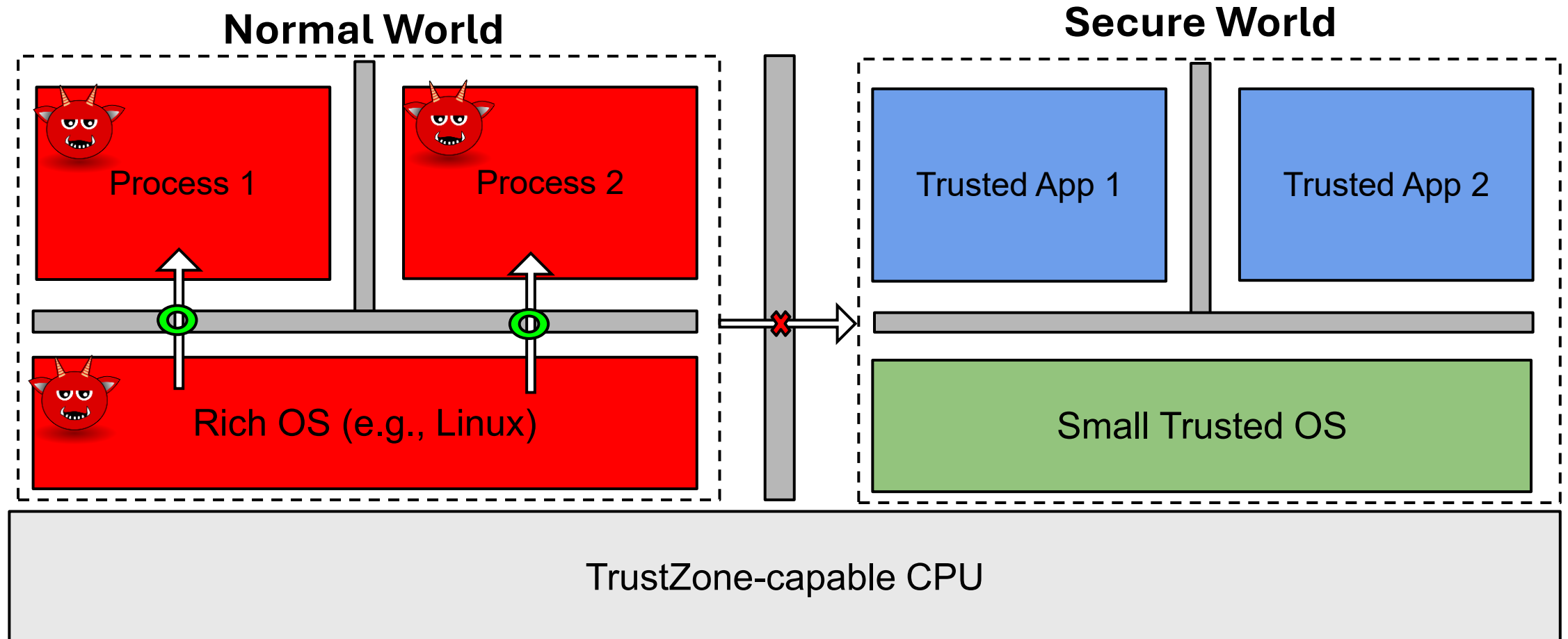
Runtime System **with TrustZone?**



ARM TrustZone Overview

Some visualizations....

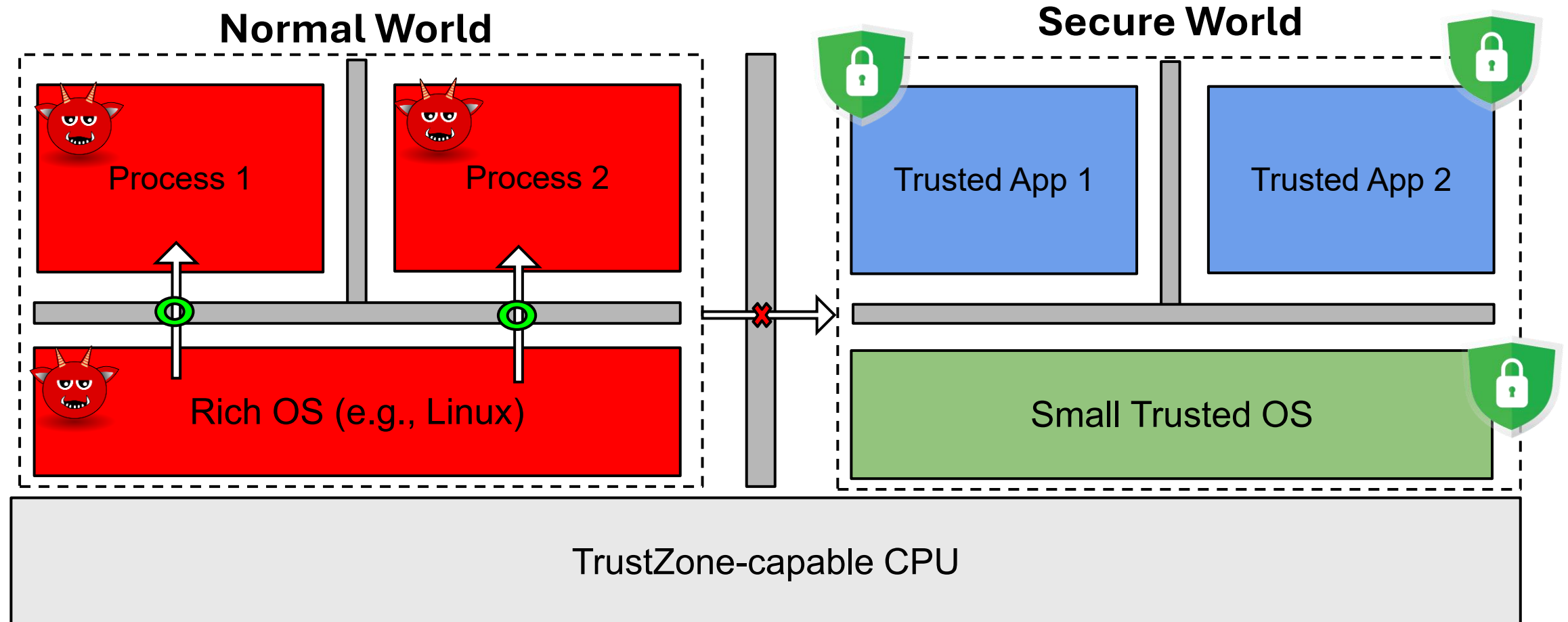
Runtime System **with TrustZone?**



ARM TrustZone Overview

Some visualizations....

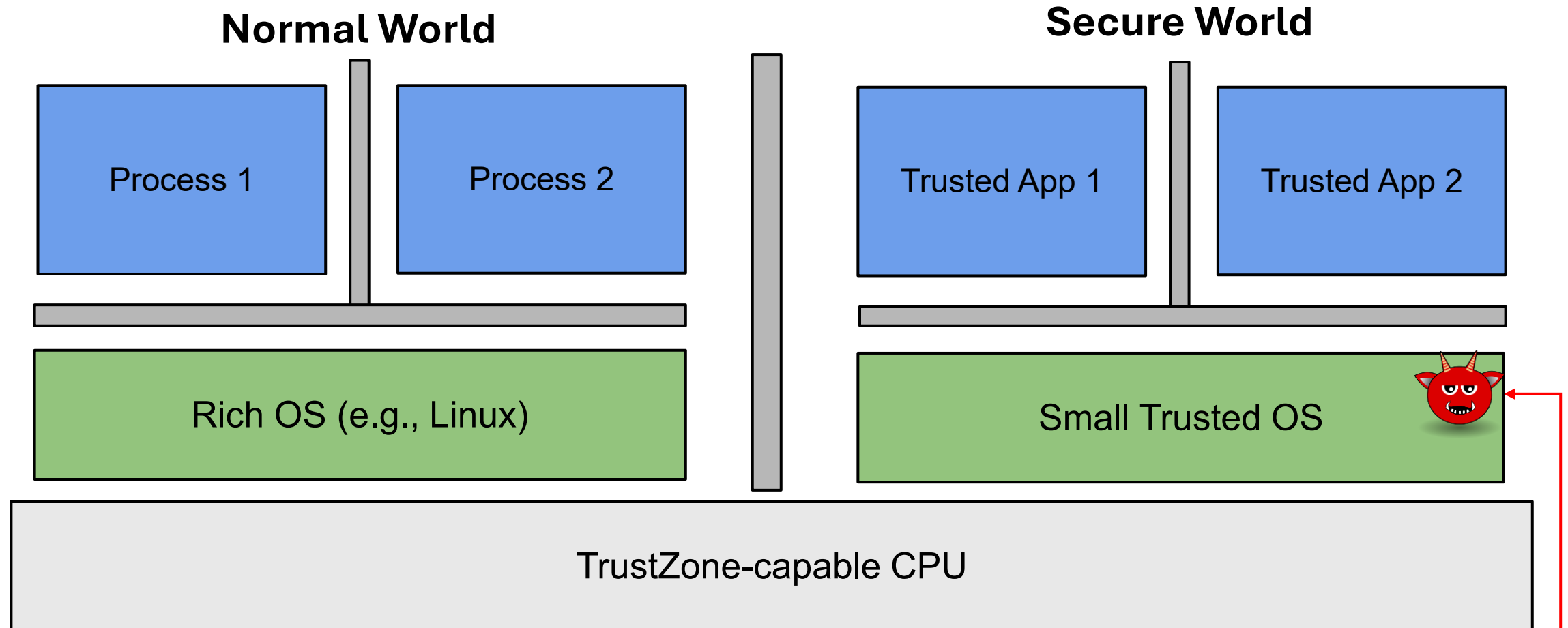
Runtime System **with TrustZone?**



ARM TrustZone Overview

Some visualizations....

Runtime System **with TrustZone?**

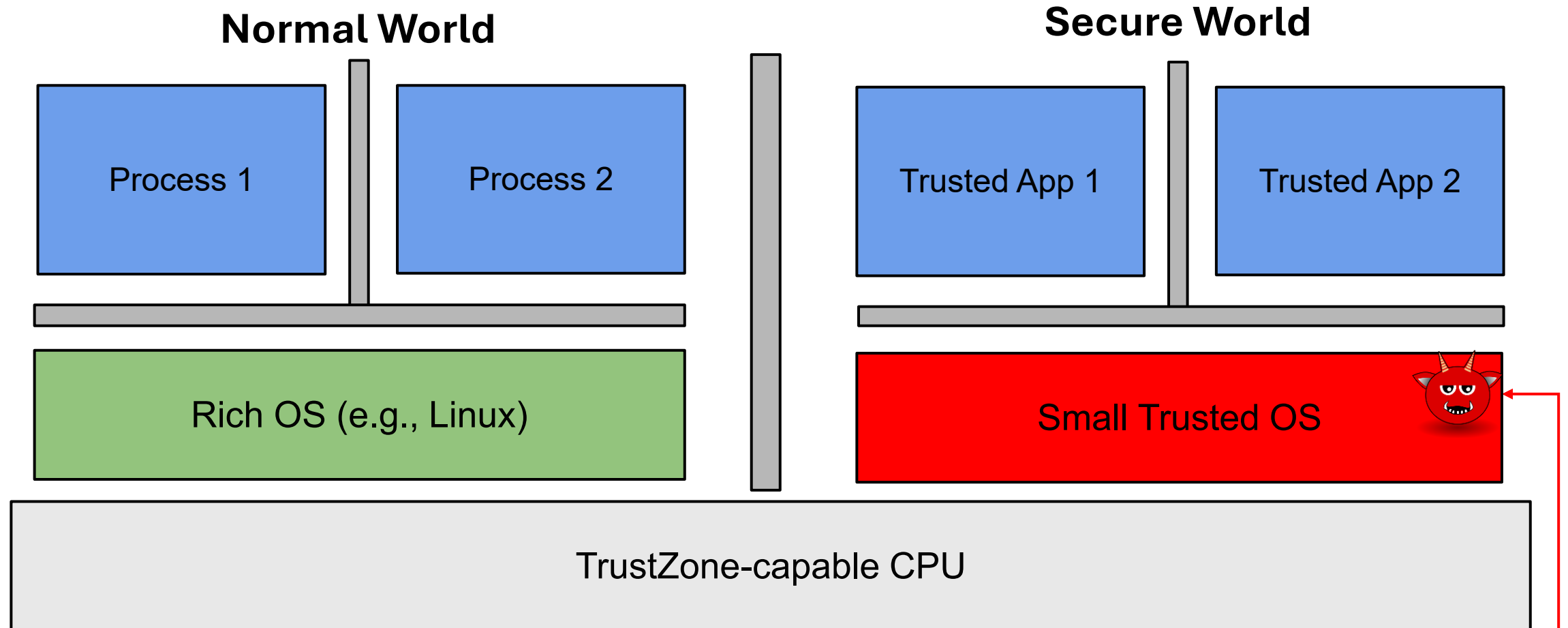


But beware!!!

ARM TrustZone Overview

Some visualizations....

Runtime System **with TrustZone?**

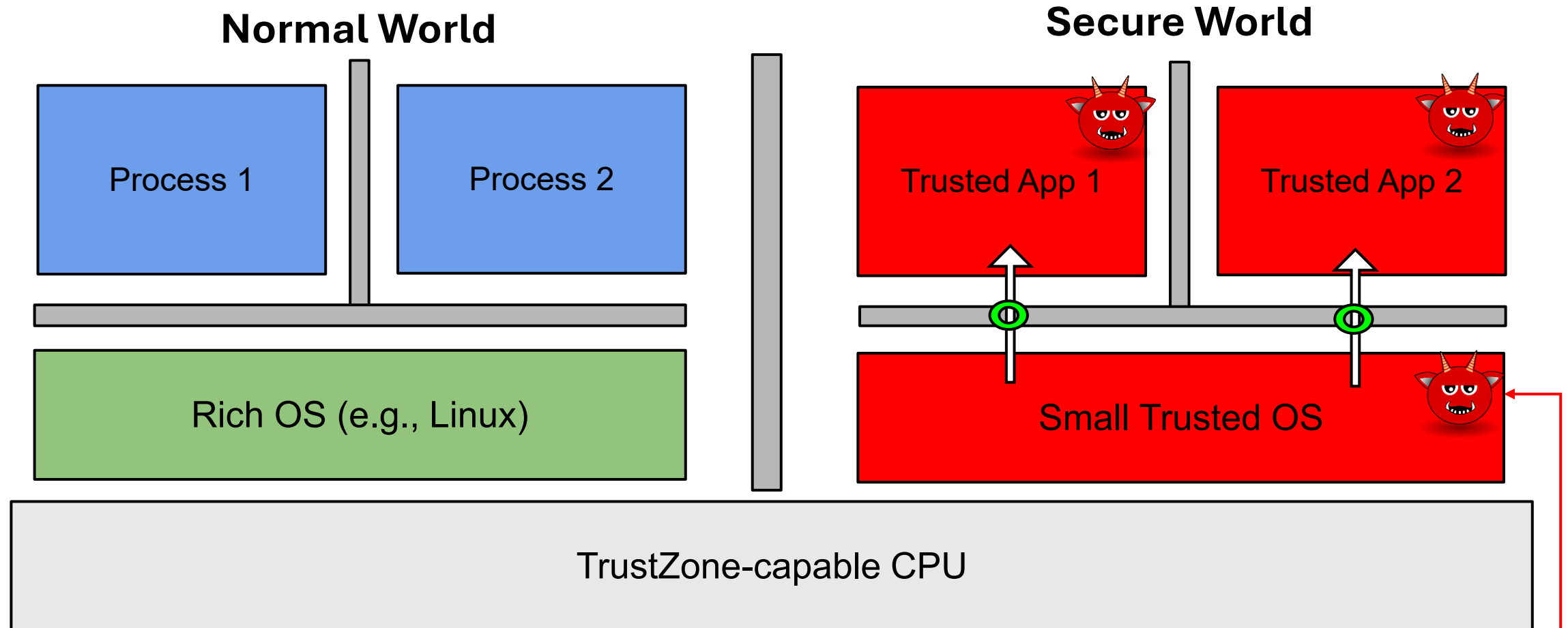


But beware!!!

ARM TrustZone Overview

Some visualizations....

Runtime System **with TrustZone?**

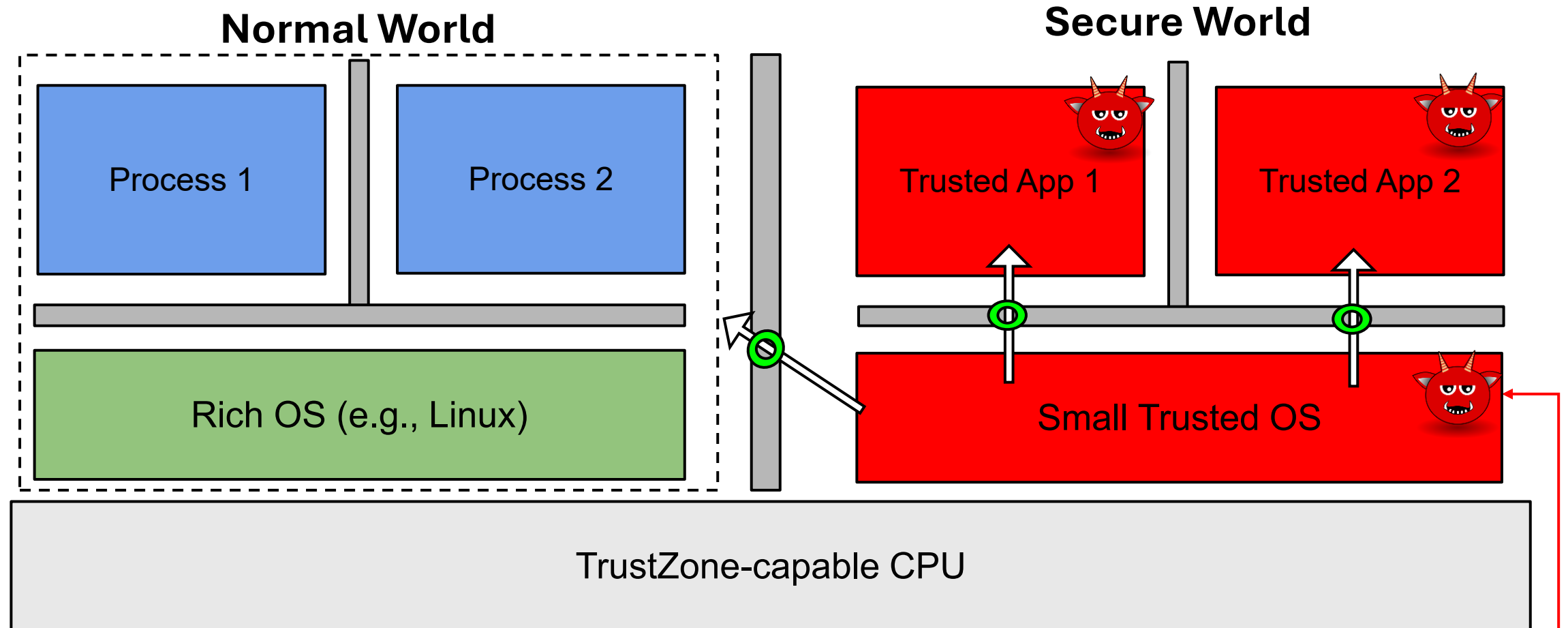


But beware!!!

ARM TrustZone Overview

Some visualizations....

Runtime System **with TrustZone?**

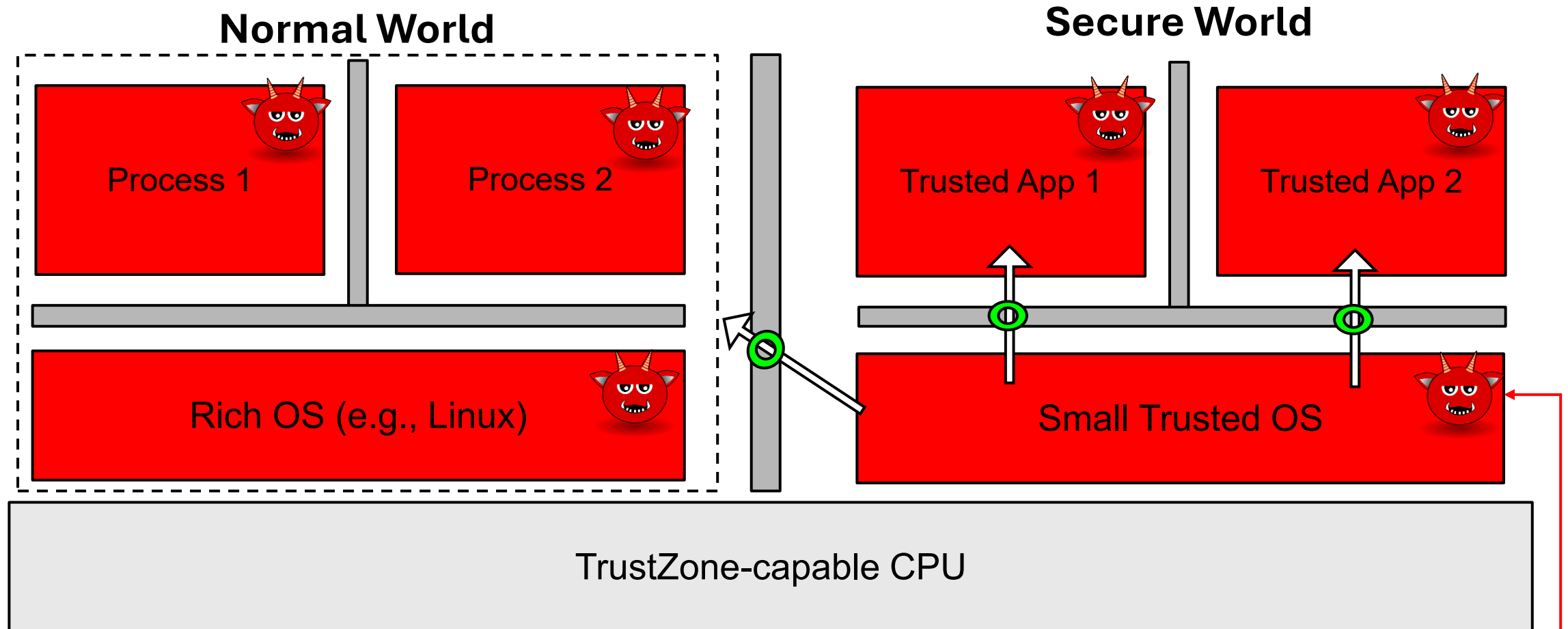


But beware!!!

ARM TrustZone Overview

Some visualizations....

Runtime System **with** TrustZone?



But beware!!!

ARM TrustZone Overview

Topics:

- Isolation in TrustZone
- Secure Monitor Calls (SMC) – Invocation of Secure World code
- Android

ARM TrustZone Overview

Topics:

- Isolation in TrustZone
- Secure Monitor Calls (SMC) – Invocation of Secure World code
- Android

ARM TrustZone – Isolation

Main Design Guidelines

- Store and manipulate security-critical info within the Secure World
 - Passwords, biometrics, private data, etc.

ARM TrustZone – Isolation

Main Design Guidelines

- Store and manipulate security-critical info within the Secure World
 - Passwords, biometrics, private data, etc.
- Keep the code inside the Secure World minimal → small TCB

ARM TrustZone – Isolation

Main Design Guidelines

- Store and manipulate security-critical info within the Secure World
 - Passwords, biometrics, private data, etc.
- Keep the code inside the Secure World minimal → small TCB
- Non-security tasks stay out in the Normal World
 - E.g., network stack, device drivers, UI implementation, etc.

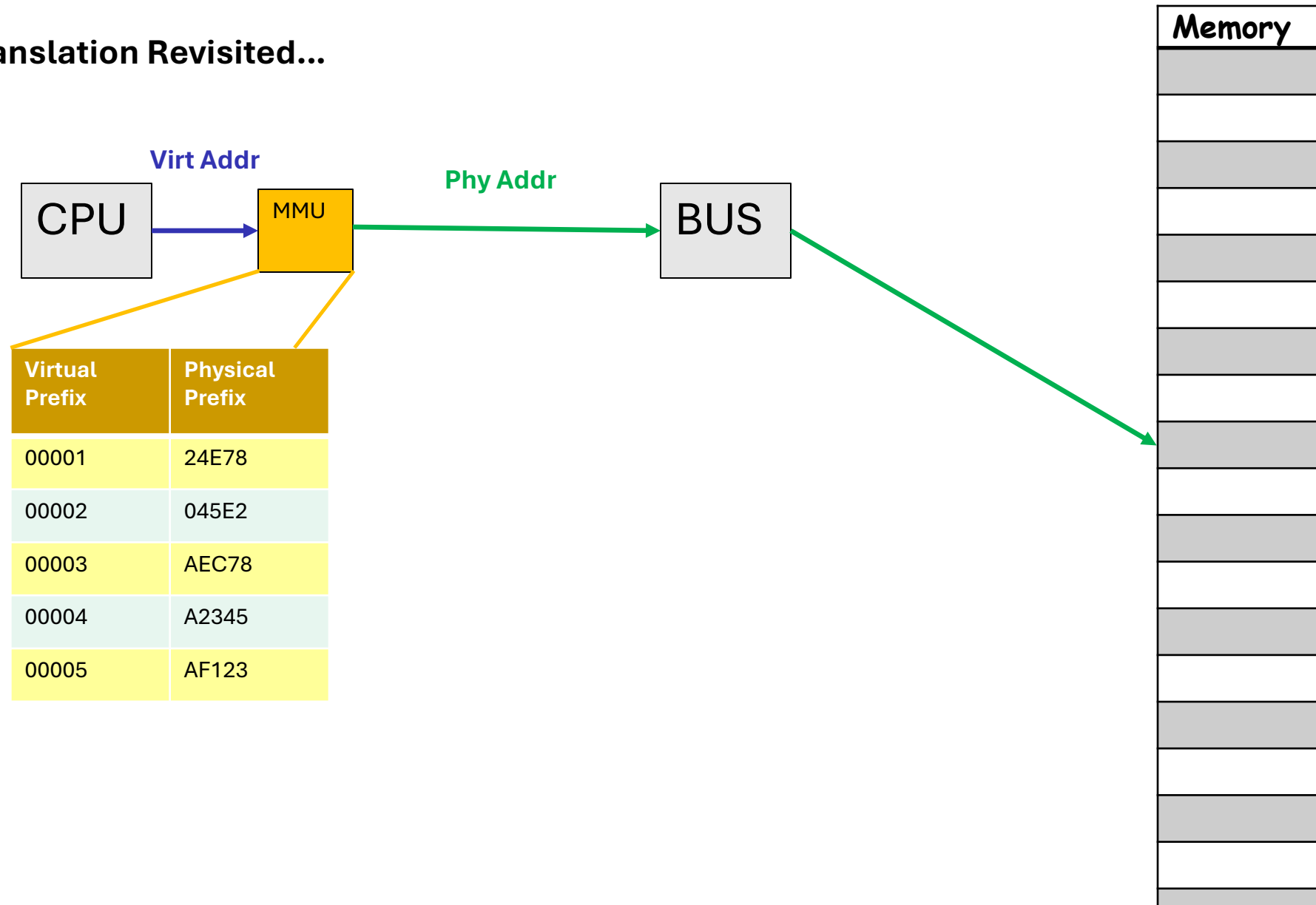
ARM TrustZone – Isolation

Main Design Guidelines

- Store and manipulate security-critical info within the Secure World
 - Passwords, biometrics, private data, etc.
- Keep the code inside the Secure World minimal → small TCB
- Non-security tasks stay out in the Normal World
 - E.g., network stack, device drivers, UI implementation, etc.
- Normal World Apps make requests to Secure World apps via well-defined APIs
 - E.g., request decryption, check this biometric input, etc...

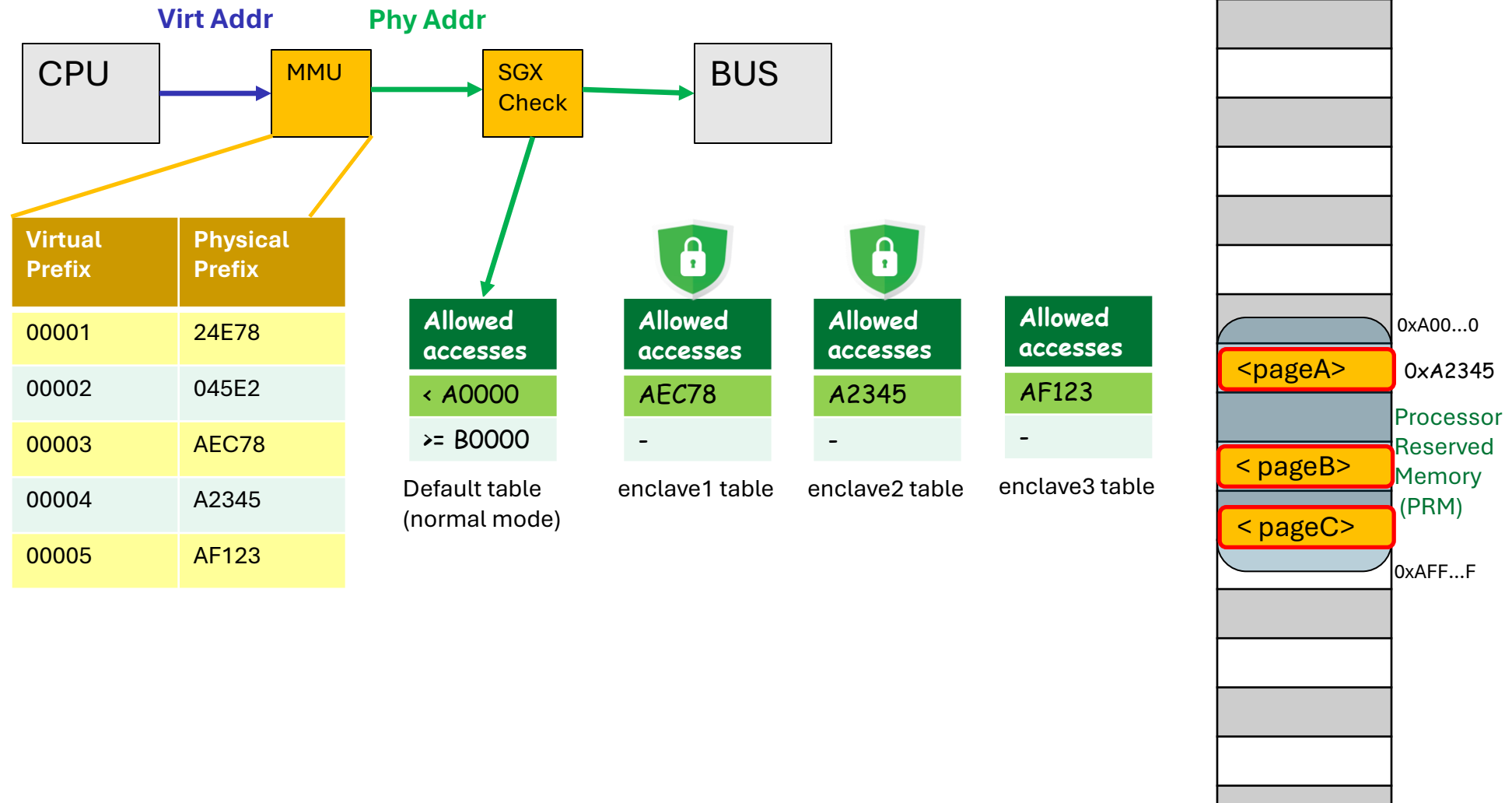
ARM TrustZone – Isolation

Memory Translation Revisited...



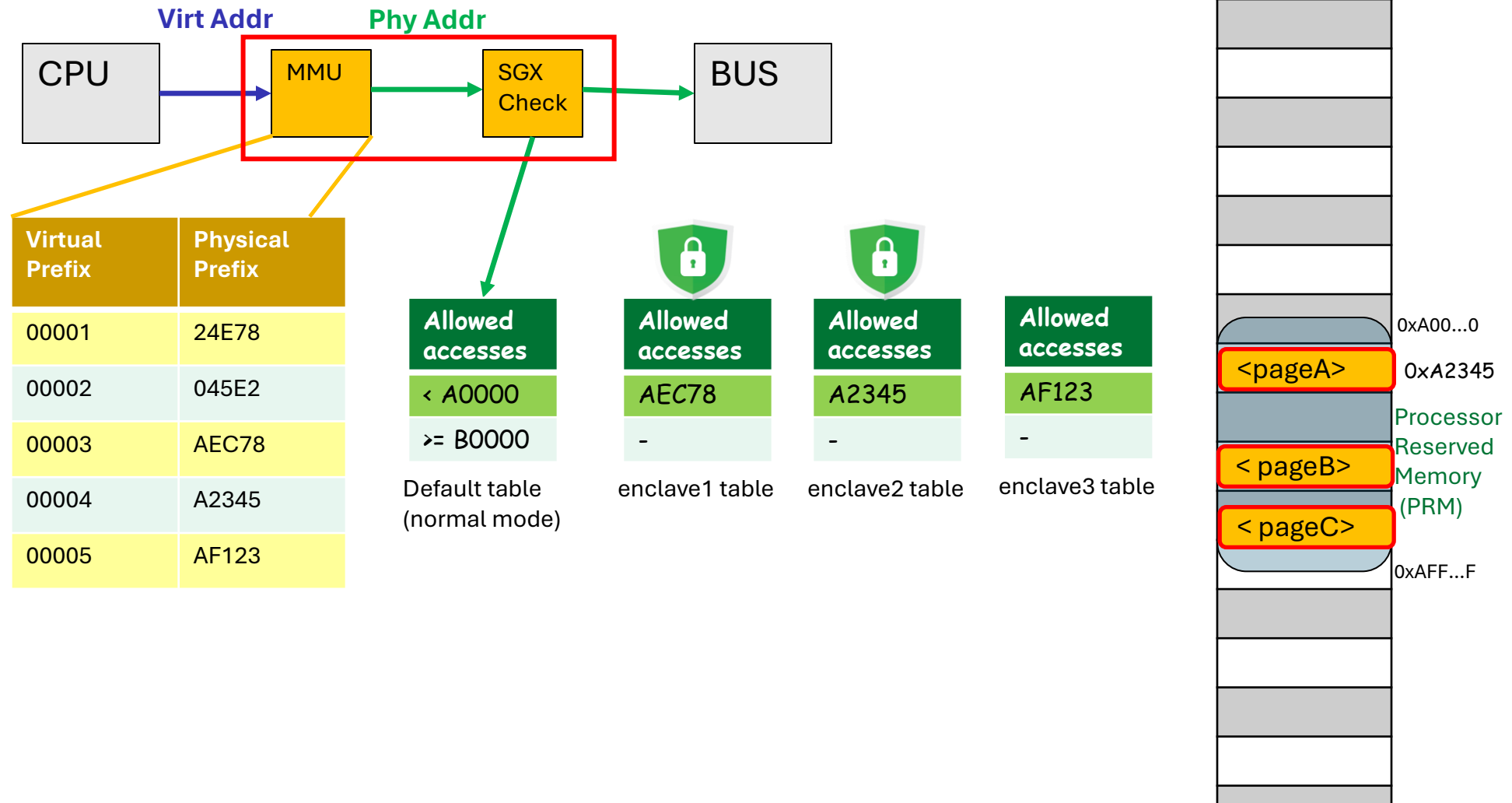
ARM TrustZone – Isolation

Memory Translation in Intel SGX Revisited....



ARM TrustZone – Isolation

Memory Translation in Intel SGX Revisited....



ARM TrustZone – Isolation

Similar idea → CPU is now involved in the memory translation

TrustZone approach:

- Two worlds → two page tables
- Both are active in an MMU at a given time

ARM TrustZone – Isolation

Similar idea → CPU is now involved in the memory translation

TrustZone approach:

- Two worlds → two page tables
- Both are active in an MMU at a given time
 - Normal world page table → managed by the Rich OS
 - Secure world page table → managed by the Trusted OS

ARM TrustZone – Isolation

Similar idea → CPU is now involved in the memory translation

TrustZone approach:

- Two worlds → two page tables
- Both are active in an MMU at a given time
 - Normal world page table → managed by the Rich OS
 - Secure world page table → managed by the Trusted OS
- One additional bit in the CPU → tells MMU which table to load

ARM TrustZone – Isolation

Similar idea → CPU is now involved in the memory translation

TrustZone approach:

- Two worlds → two page tables
- Both are active in an MMU at a given time
 - Normal world page table → managed by the Rich OS
 - Secure world page table → managed by the Trusted OS
- One additional bit in the CPU → tells MMU which table to load
 - **Non-Secure (NS) bit:**

ARM TrustZone – Isolation

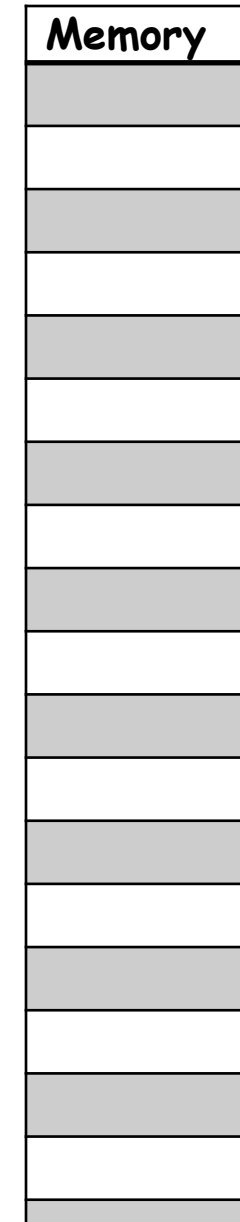
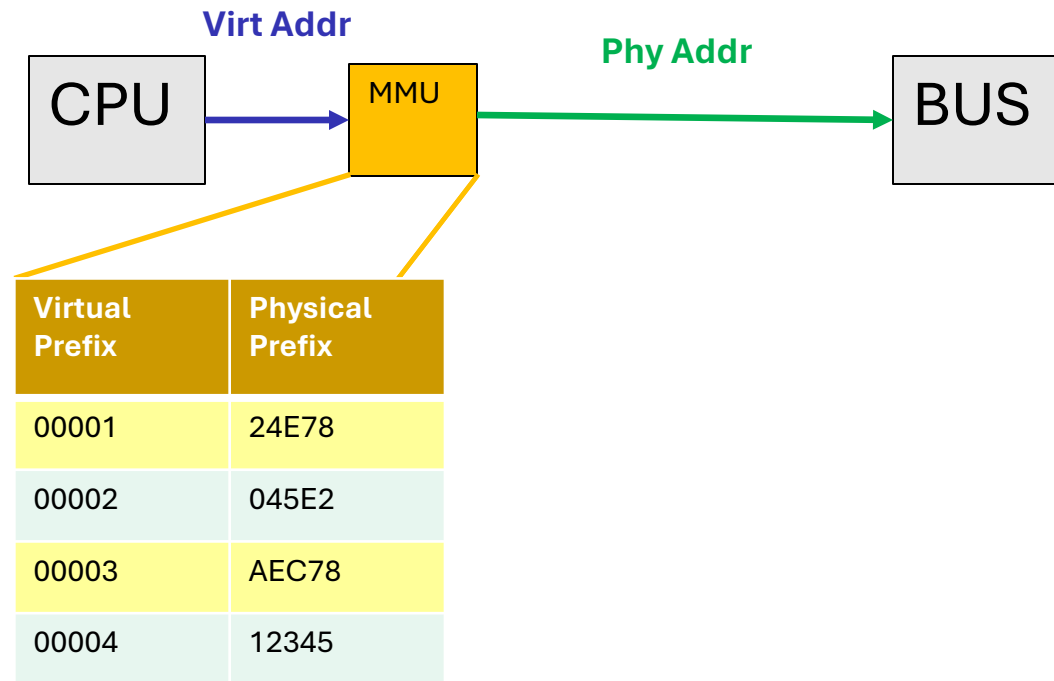
Similar idea → CPU is now involved in the memory translation

TrustZone approach (part 1):

- Two worlds → two page tables
- Both are active in an MMU at a given time
 - Normal world page table → managed by the Rich OS
 - Secure world page table → managed by the Trusted OS
- One additional bit in the CPU → tells MMU which table to load
 - **Non-Secure (NS) bit:**
 - NS = 1 → currently in Normal World, Secure World access is blocked
 - NS = 0 → currently in Secure World, Secure World access is allowed

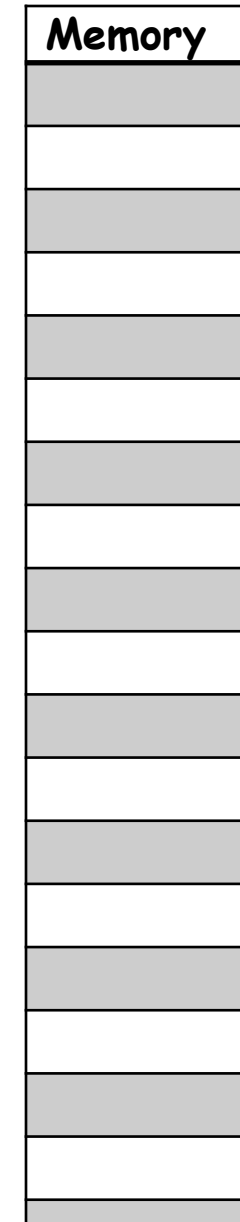
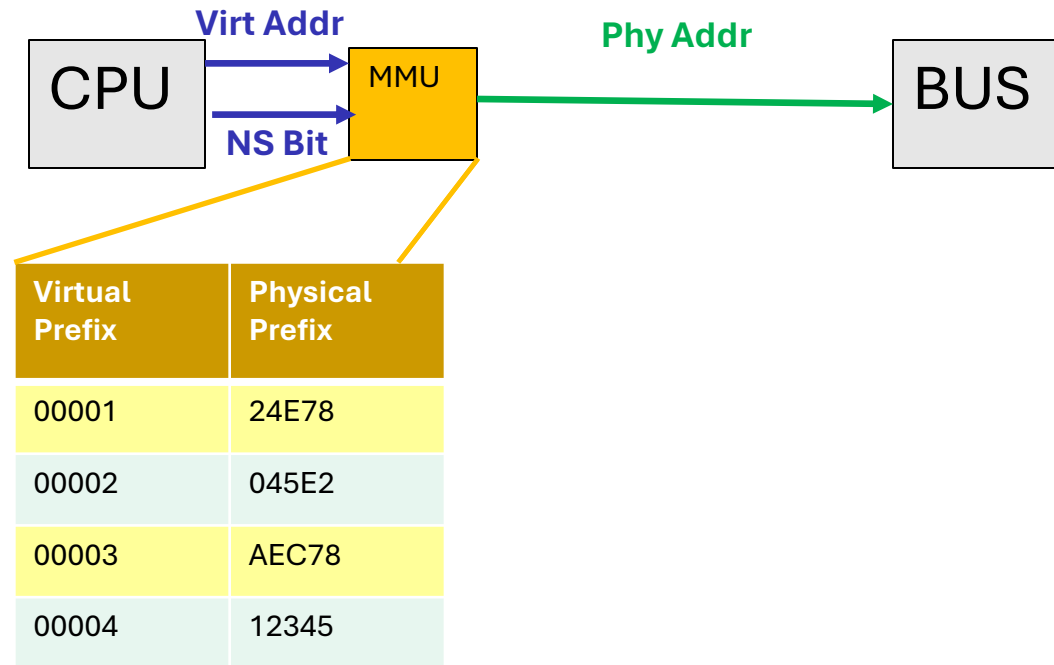
ARM TrustZone – Isolation

The details...



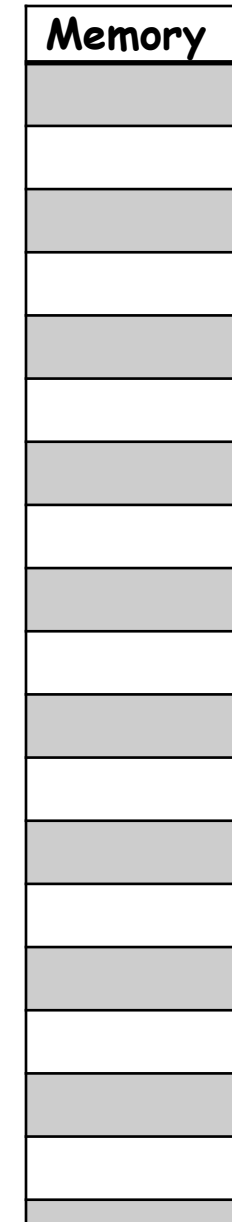
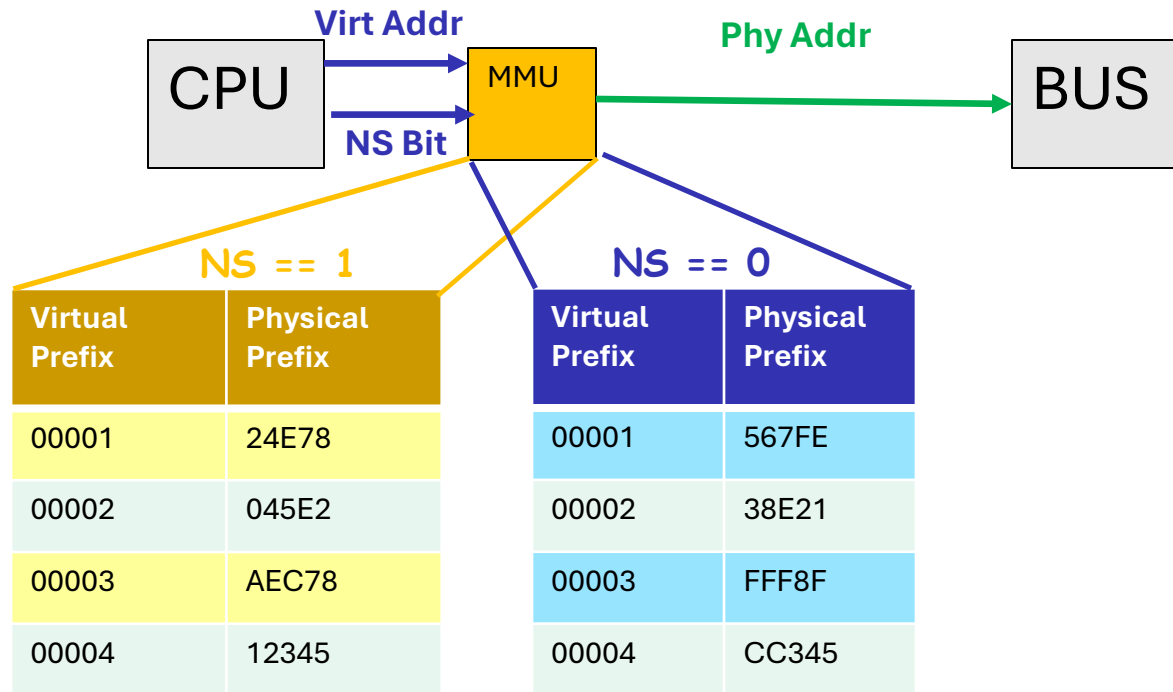
ARM TrustZone – Isolation

Now, the CPU also passes the NS bit to the MMU



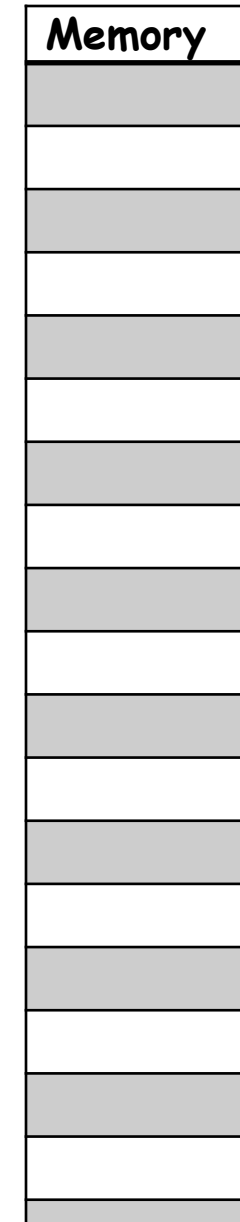
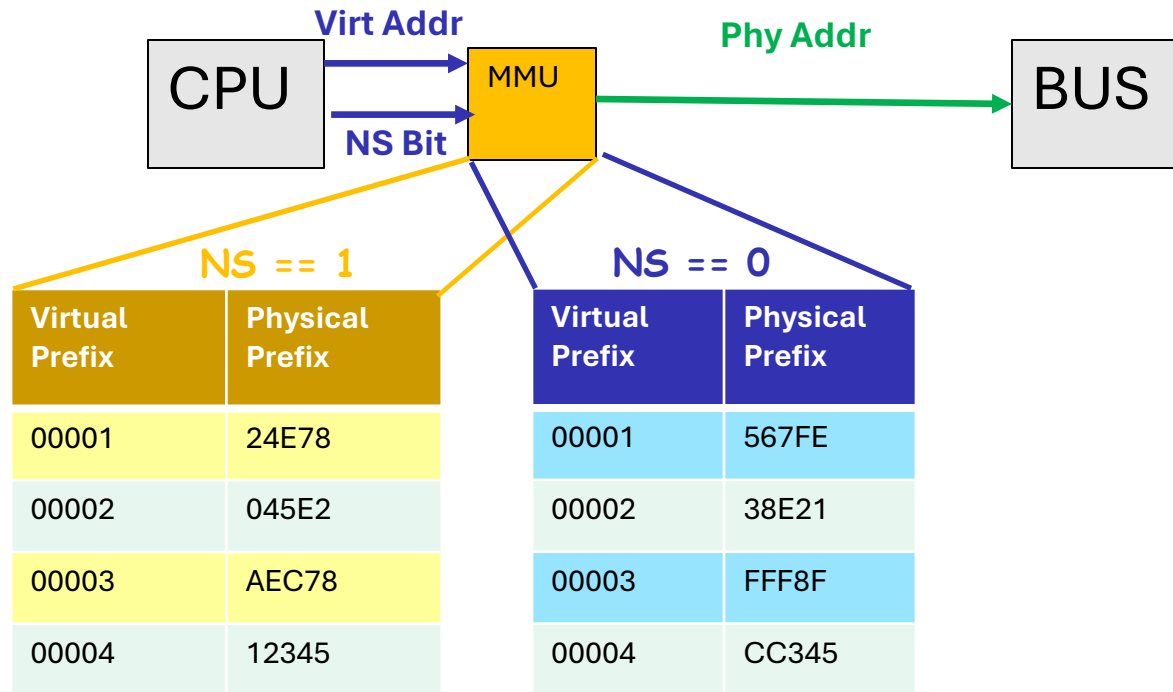
ARM TrustZone – Isolation

And the MMU has two page tables.
NS bit tells MMU which to use



ARM TrustZone – Isolation

Now, the CPU also passes the NS bit to the MMU



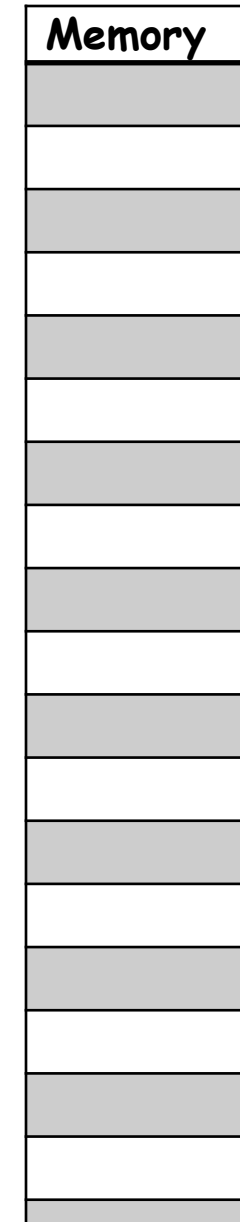
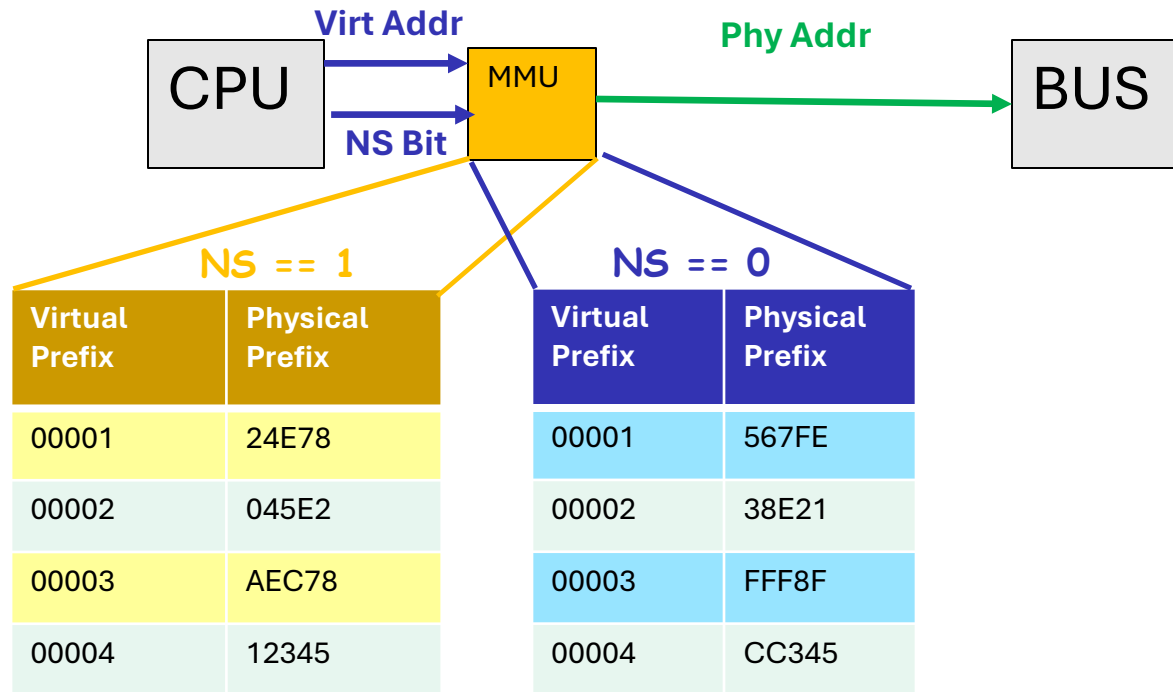
ARM TrustZone – Isolation

TrustZone approach (continued..)

- **Physical Memory Partitioning** in addition to the modified MMU!
- TrustZone enables configuration of specific **physical** memory regions as **secure** or **non-secure**, such that applications can only access memory assigned to their world
- **How?**
 - **Two hardware controllers:**
 - TrustZone Address Space Controller (TZASC) → on chip memory (SoC) and DRAM
 - TrustZone Memory Adapter (TZMA) → off-chip memory (e.g., external peripherals SRAM)
- **TZASC and TZMA have the same function applied to different resources**

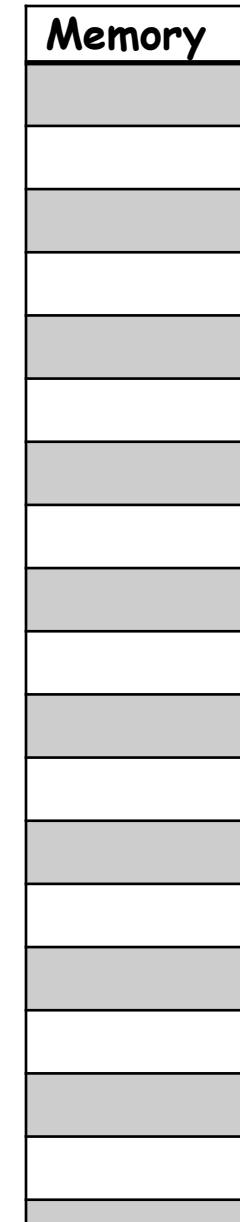
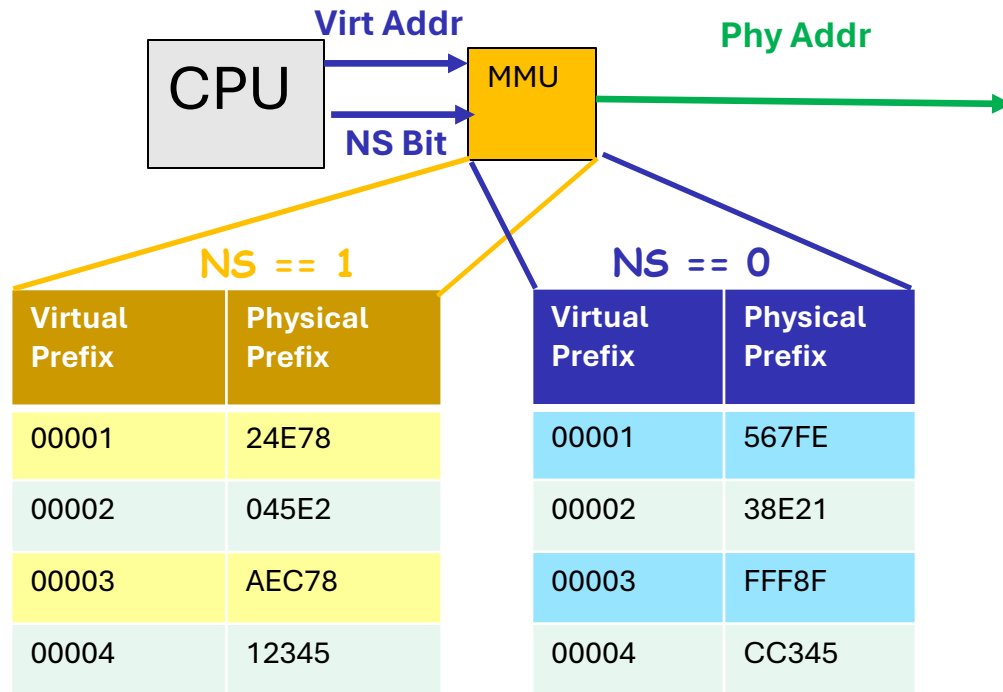
ARM TrustZone – Isolation

Where are the TZASC/TZMA?



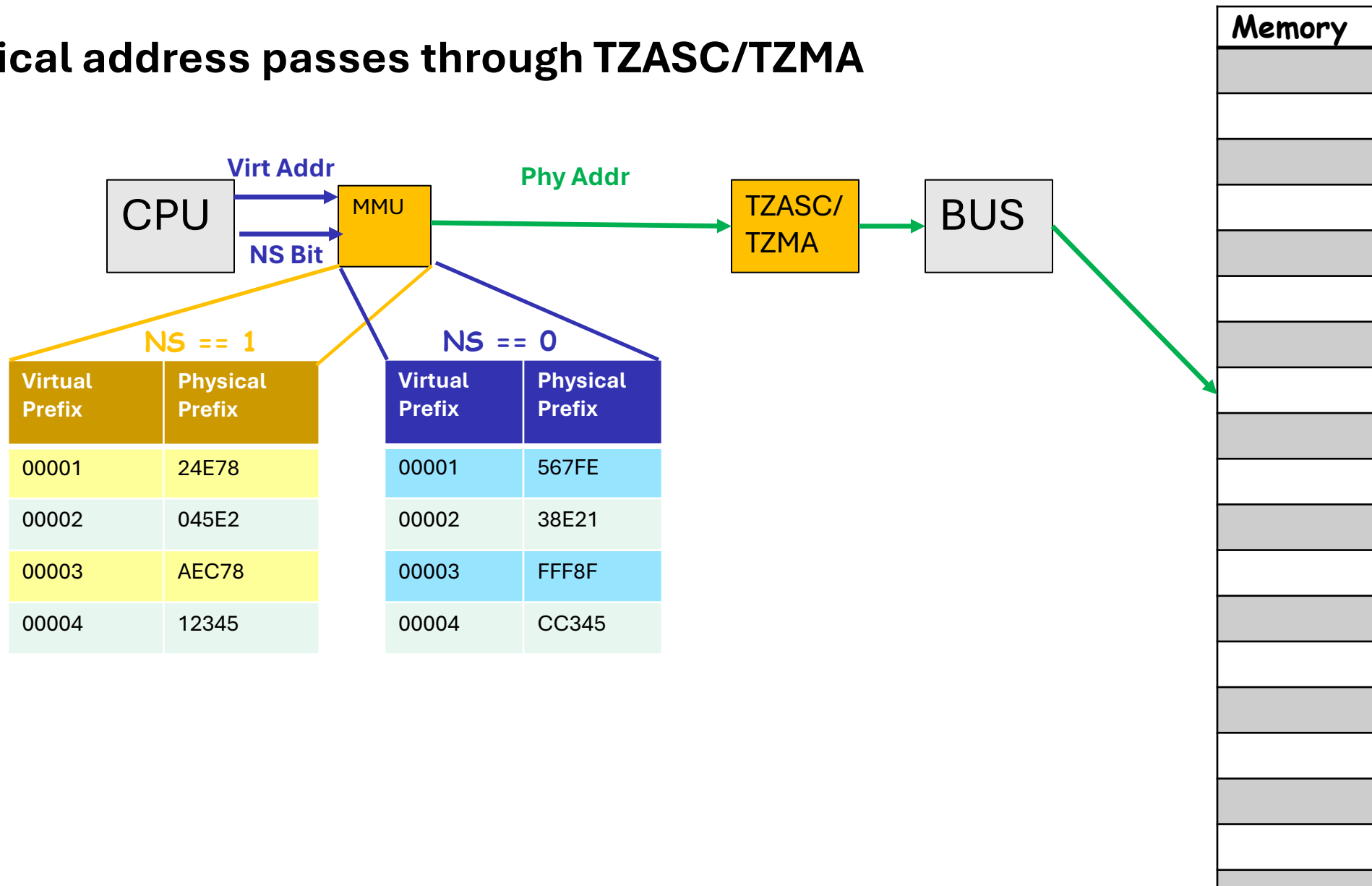
ARM TrustZone – Isolation

Instead of the MMU accessing the BUS directly...



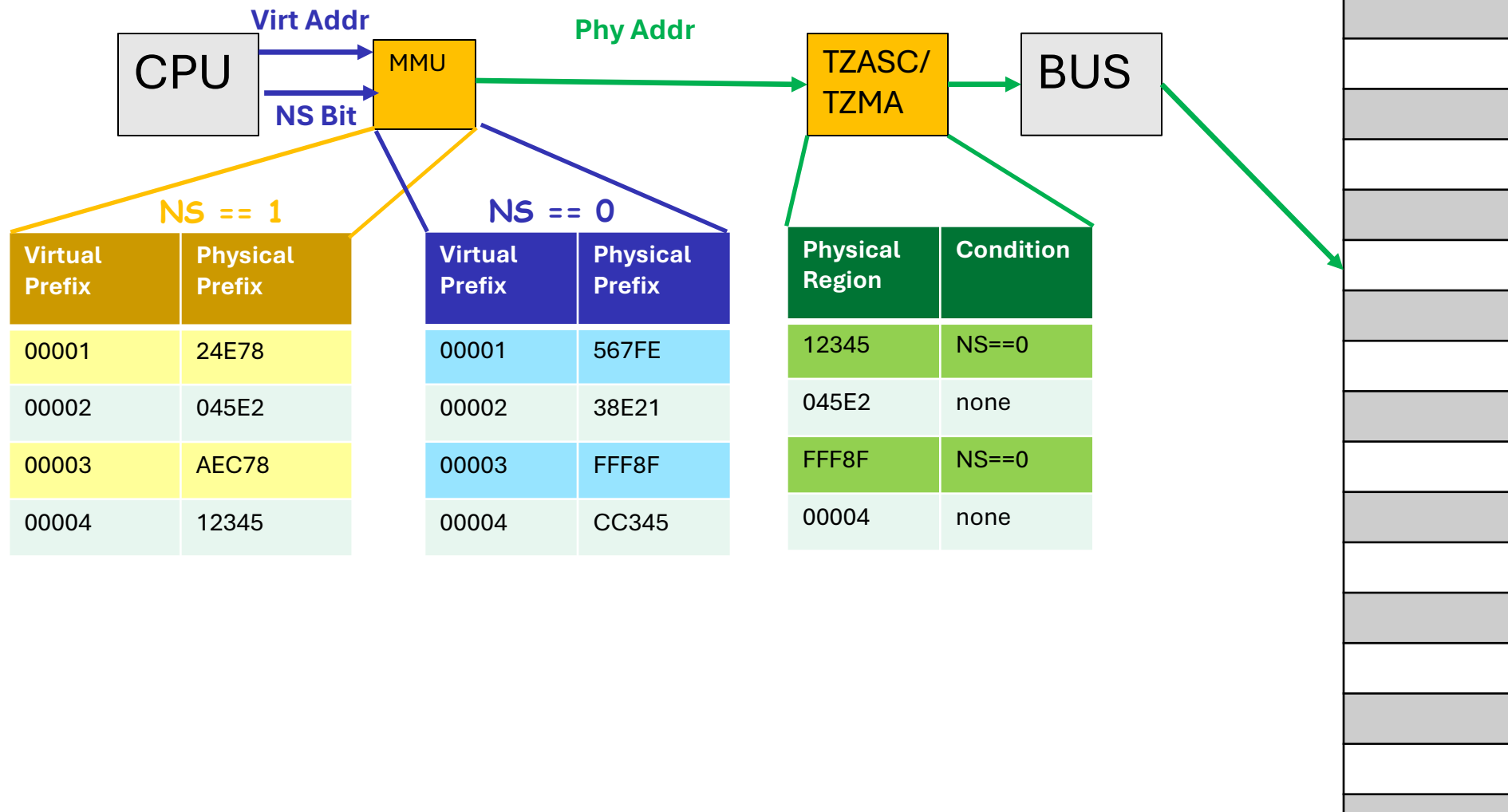
ARM TrustZone – Isolation

Physical address passes through TZASC/TZMA



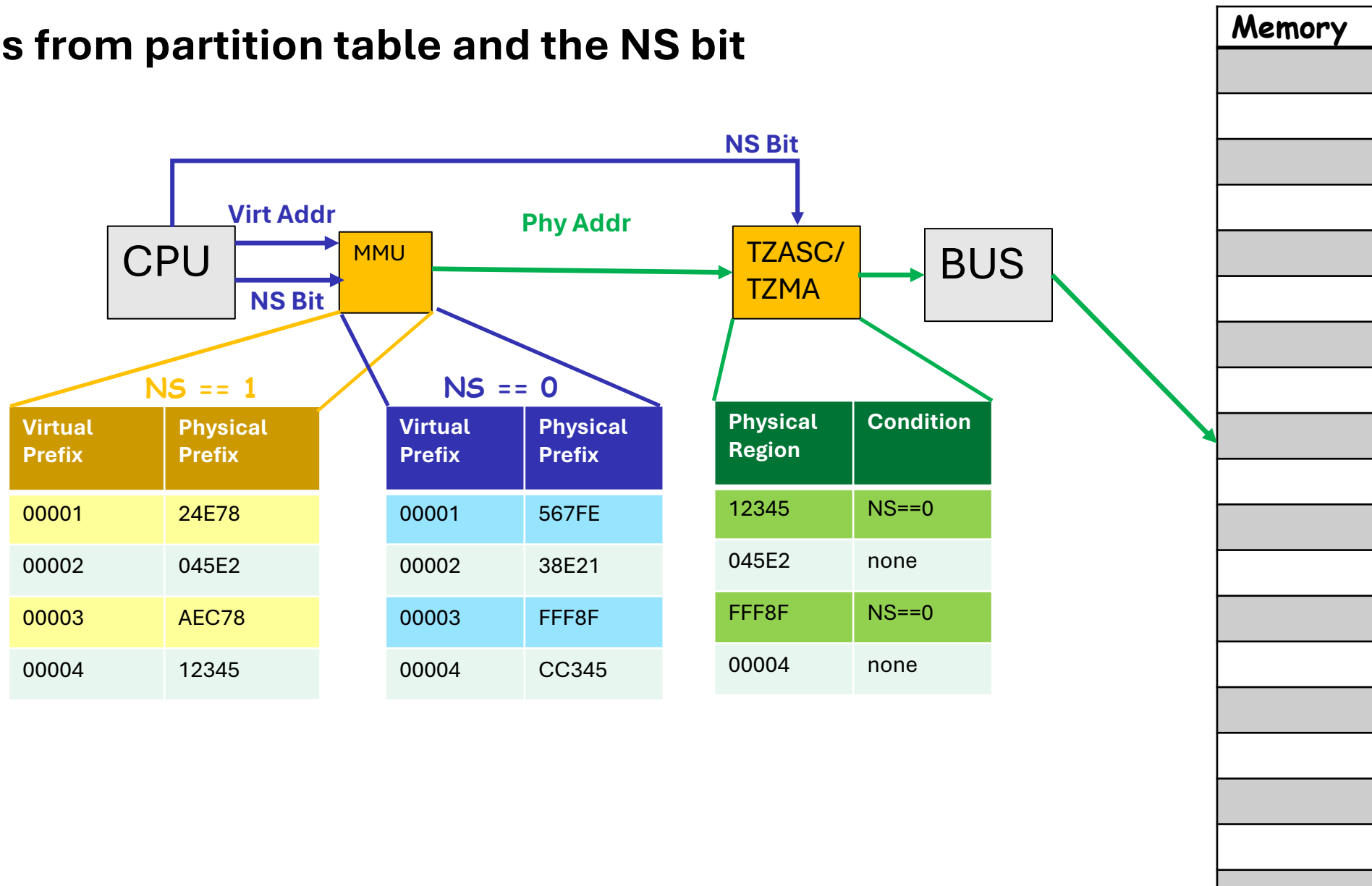
ARM TrustZone – Isolation

TZASC/TZMA check access according to the defined partition



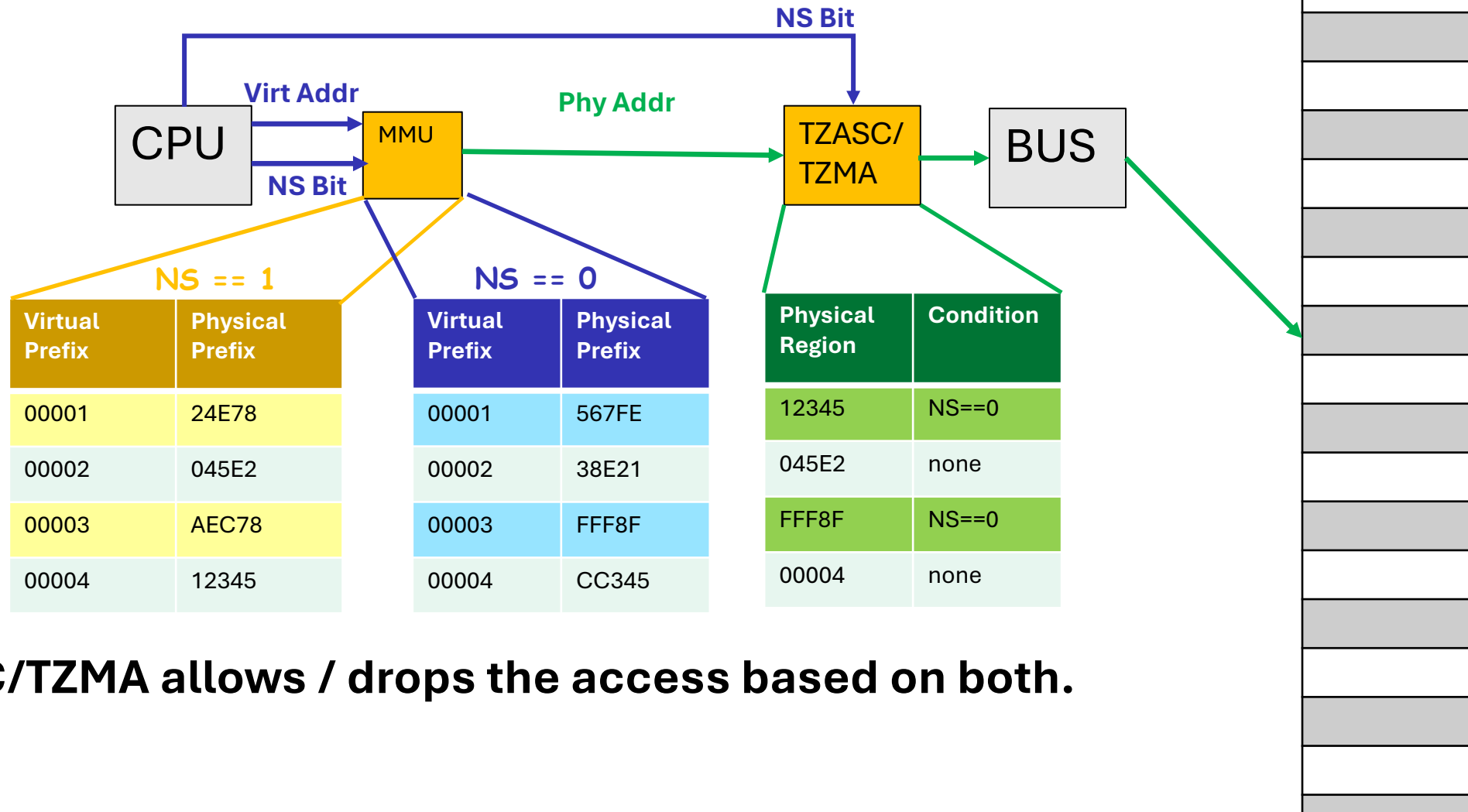
ARM TrustZone – Isolation

Reads from partition table and the NS bit



ARM TrustZone – Isolation

Reads from partition table and the NS bit



TZASC/TZMA allows / drops the access based on both.

ARM TrustZone – Isolation

TrustZone-A MMU + TZASC/TZMA

Together provide isolation in TrustZone

TZASC/TZMA:

Implement physical isolation between the worlds

- Isolate physical memory, peripherals, and hardware resources
- Provides system-level isolation

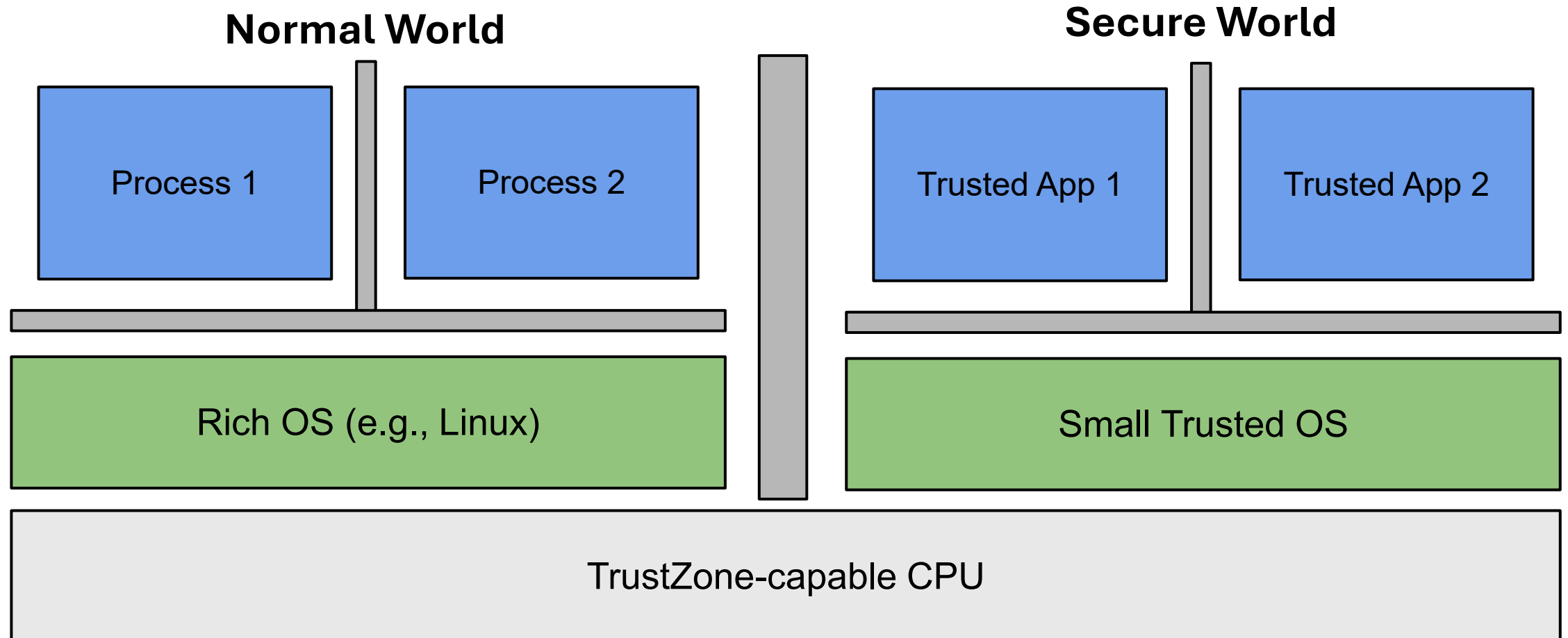
MMU:

- Virtual isolation between processes running in each world
- Provides process-level isolation

ARM TrustZone – Isolation

Some visualizations....

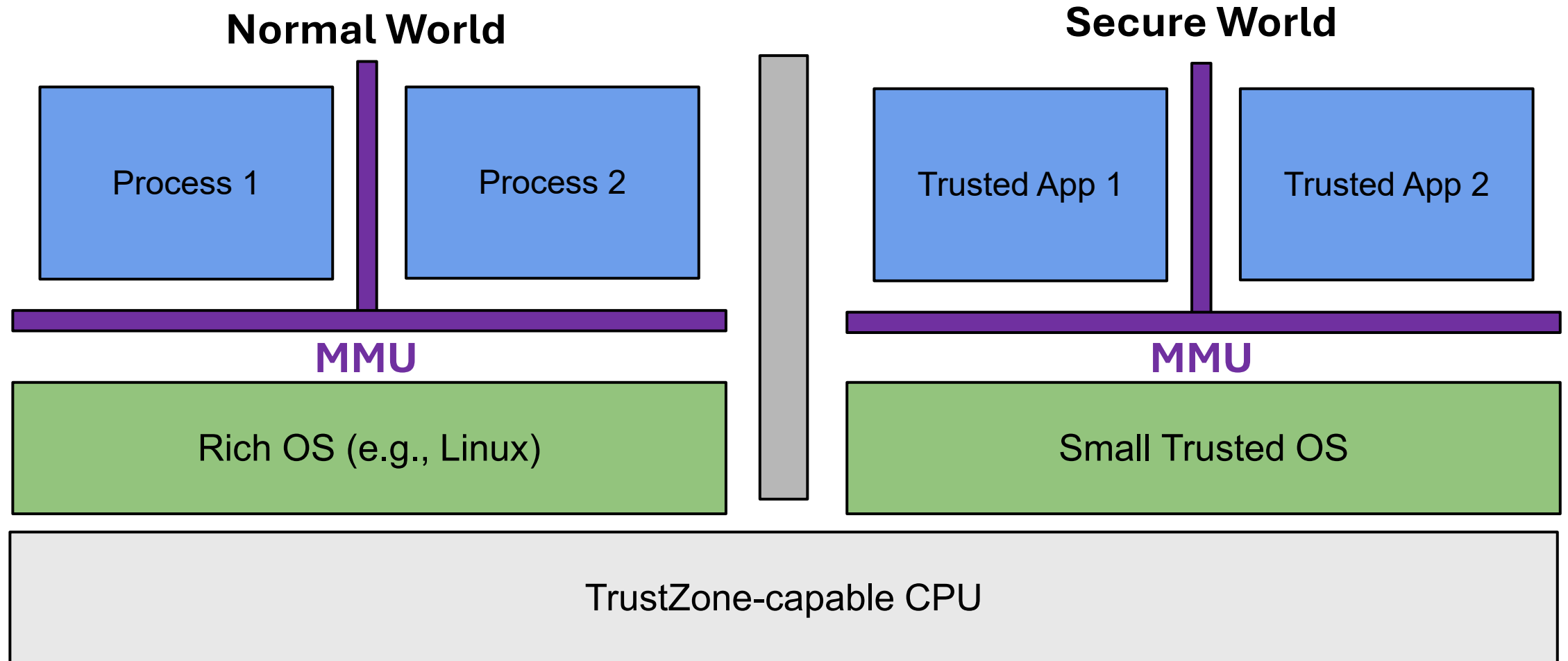
Runtime System **with TrustZone?**



ARM TrustZone – Isolation

Some visualizations....

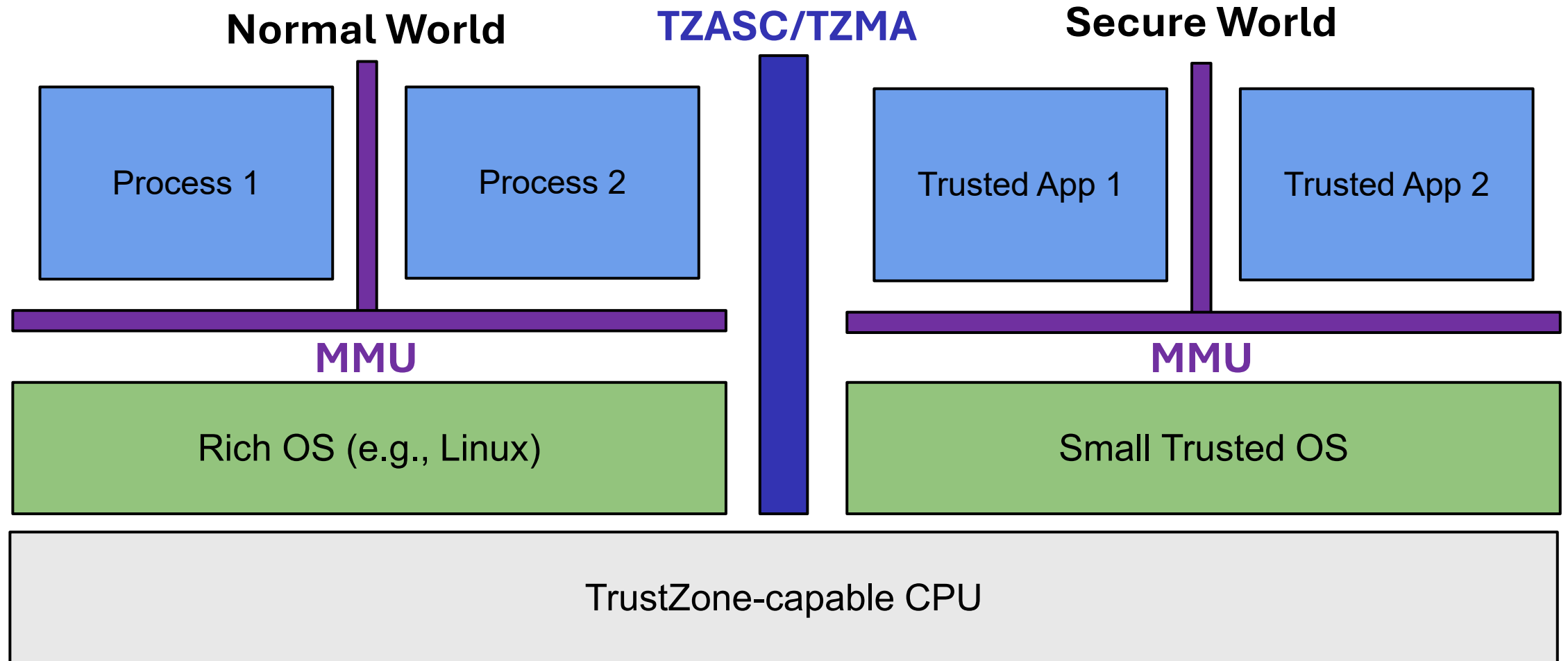
Runtime System **with TrustZone?**



ARM TrustZone – Isolation

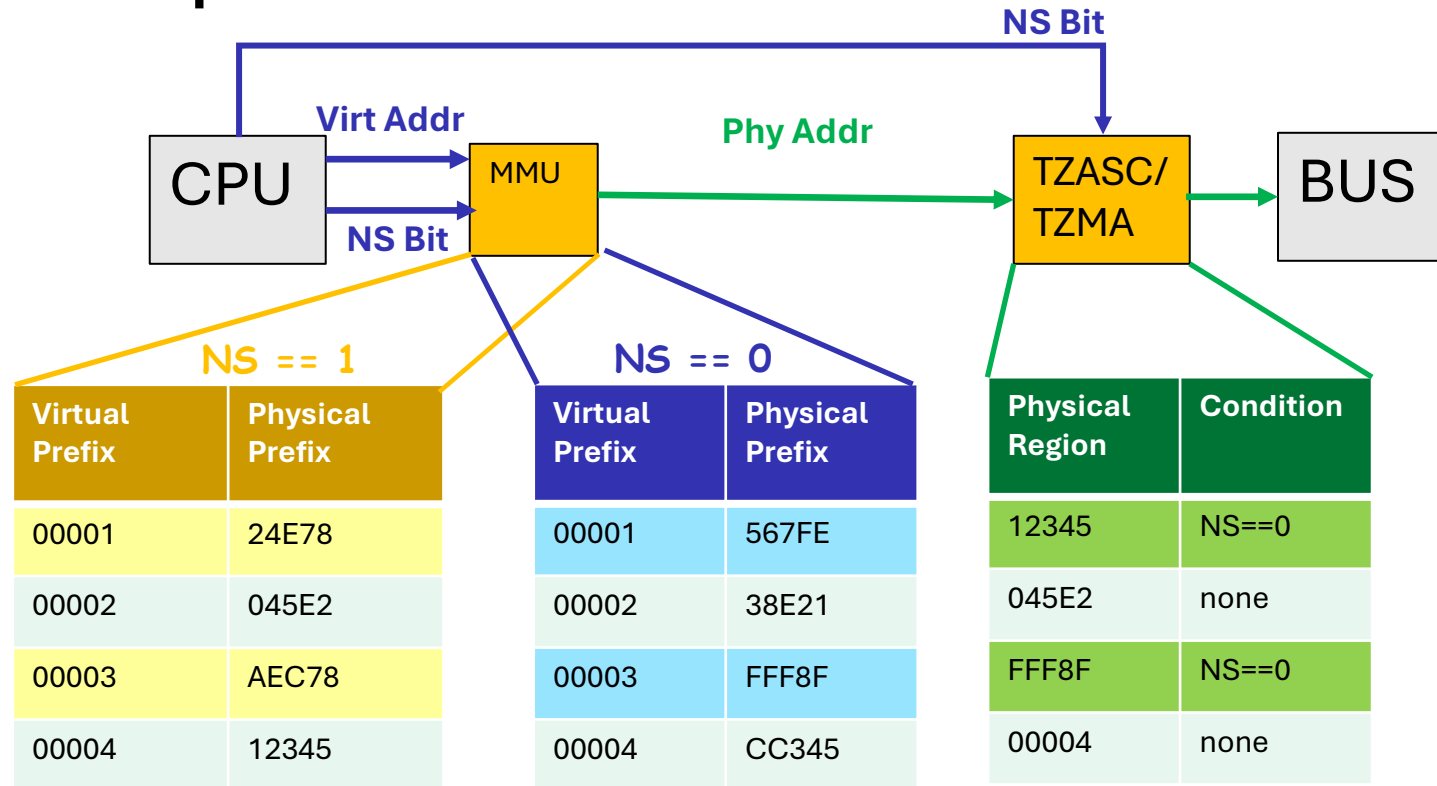
Some visualizations....

Runtime System **with TrustZone?**



ARM TrustZone – Isolation

Some examples....



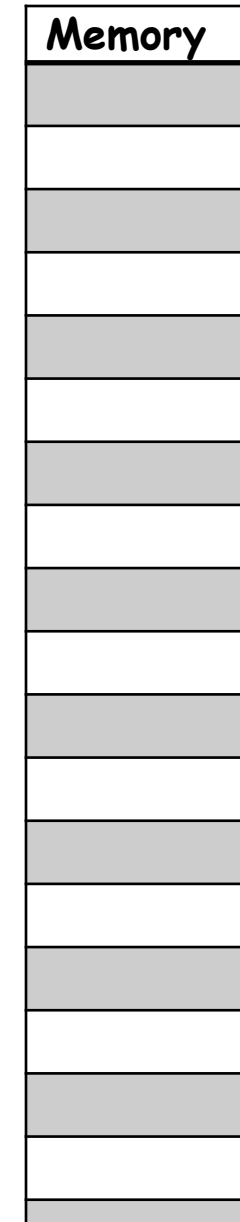
Access 1: NS = 1; Virt Addr = 0x00002123

Access 2: NS = 0; Virt Addr = 0x00003456

Access 3: NS = 1; Virt Addr = 0x00004789

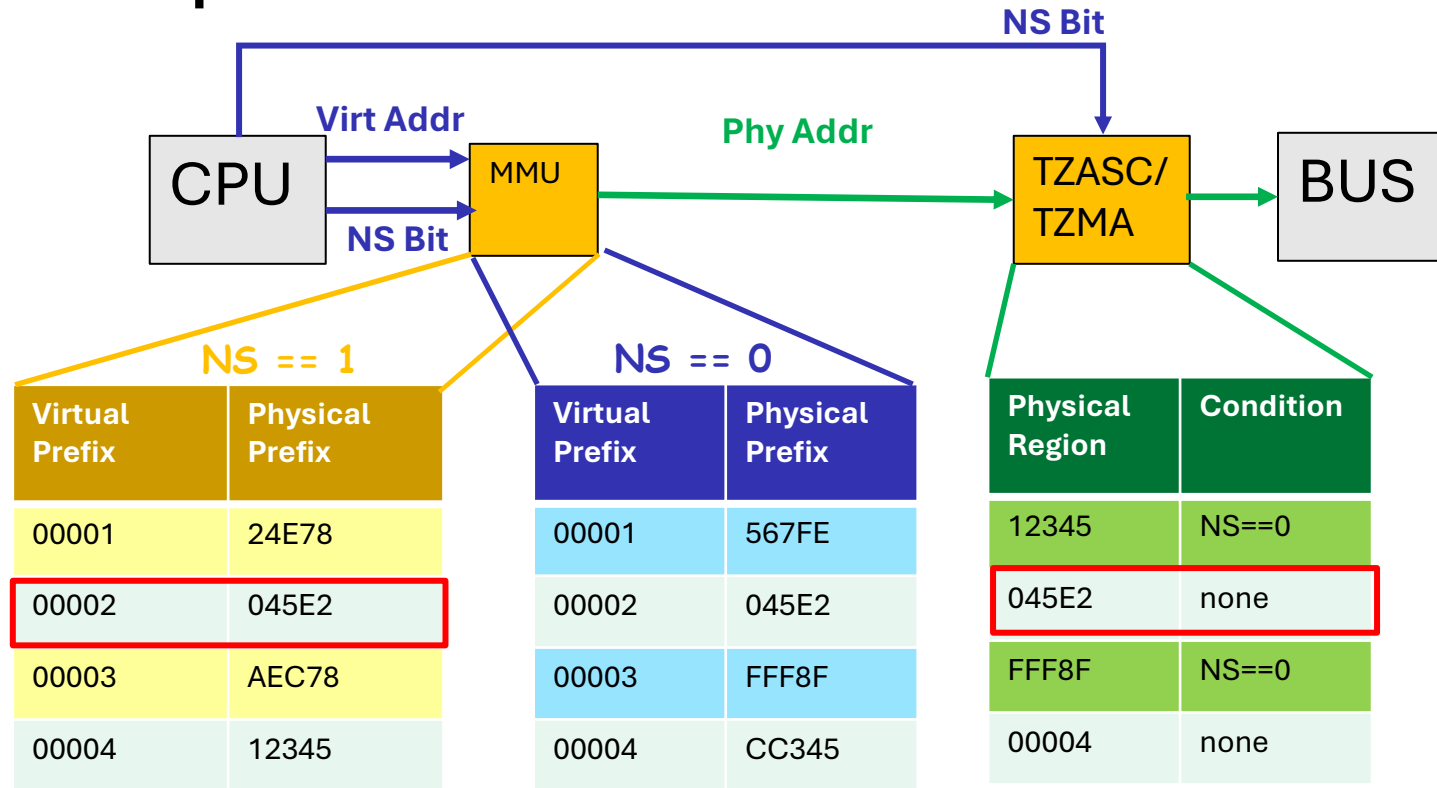
Access 4: NS = 1; Virt Addr = 0x00006333

Access 5: NS = 0; Virt Addr = 0x00002123



ARM TrustZone – Isolation

Some examples....



Access 1: NS = 1; Virt Addr = 0x00002123

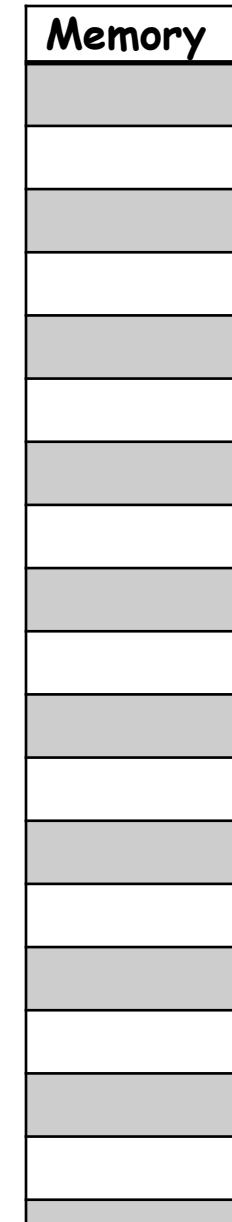
Access 2: NS = 0; Virt Addr = 0x00003456

Access 3: NS = 1; Virt Addr = 0x00004789

Access 4: NS = 1; Virt Addr = 0x00006333

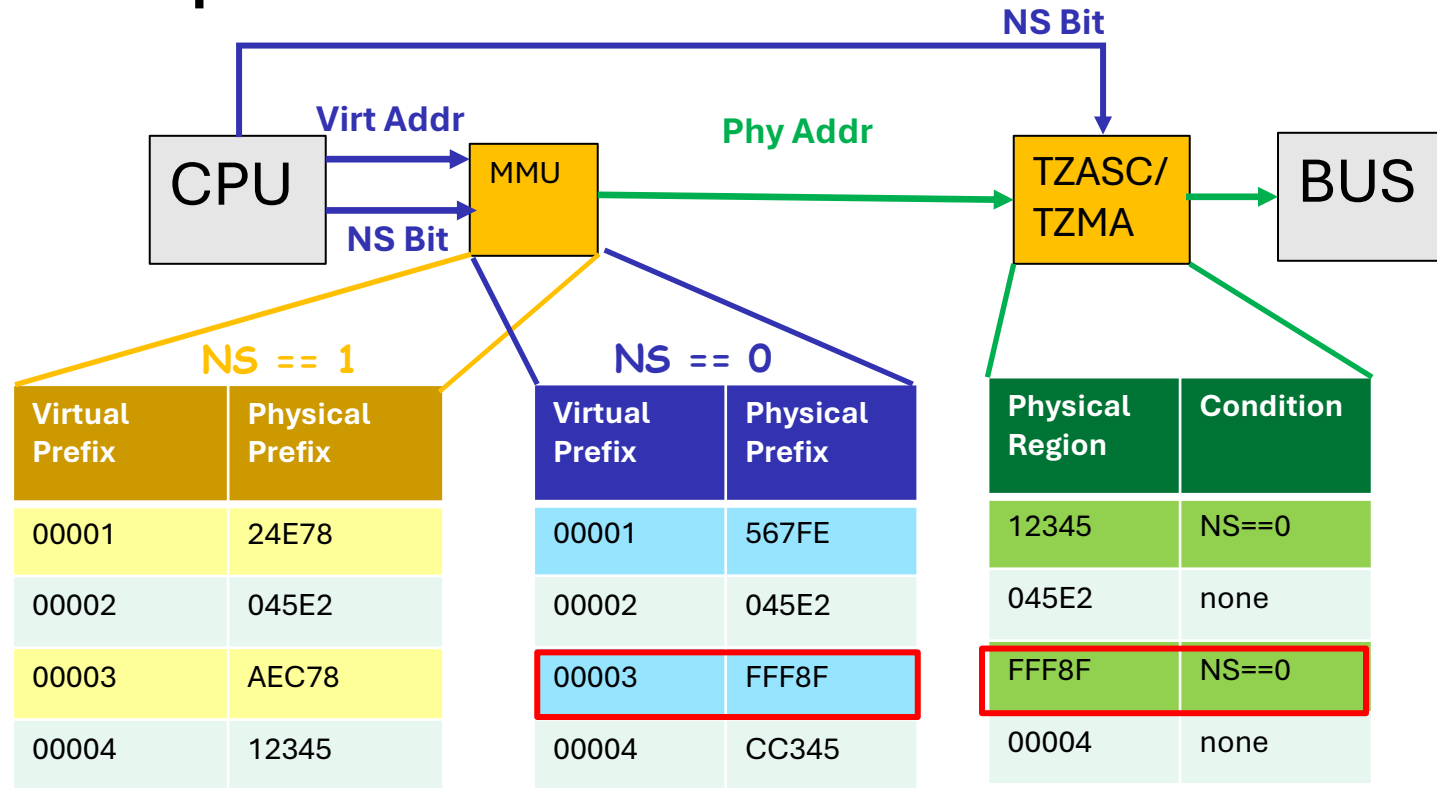
Access 5: NS = 0; Virt Addr = 0x00002123

Accesses 0x045E2123



ARM TrustZone – Memory Translation

Some examples....



Access 1: NS = 1; Virt Addr = 0x00002123

Access 2: NS = 0; Virt Addr = 0x00003456

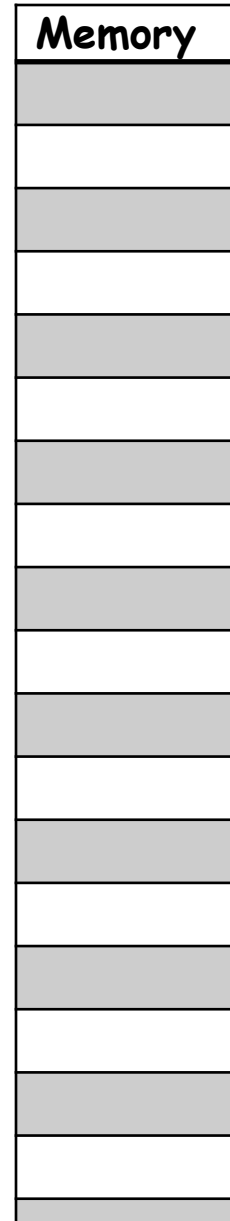
Access 3: NS = 1; Virt Addr = 0x00004789

Access 4: NS = 1; Virt Addr = 0x00006333

Access 5: NS = 0; Virt Addr = 0x00002123

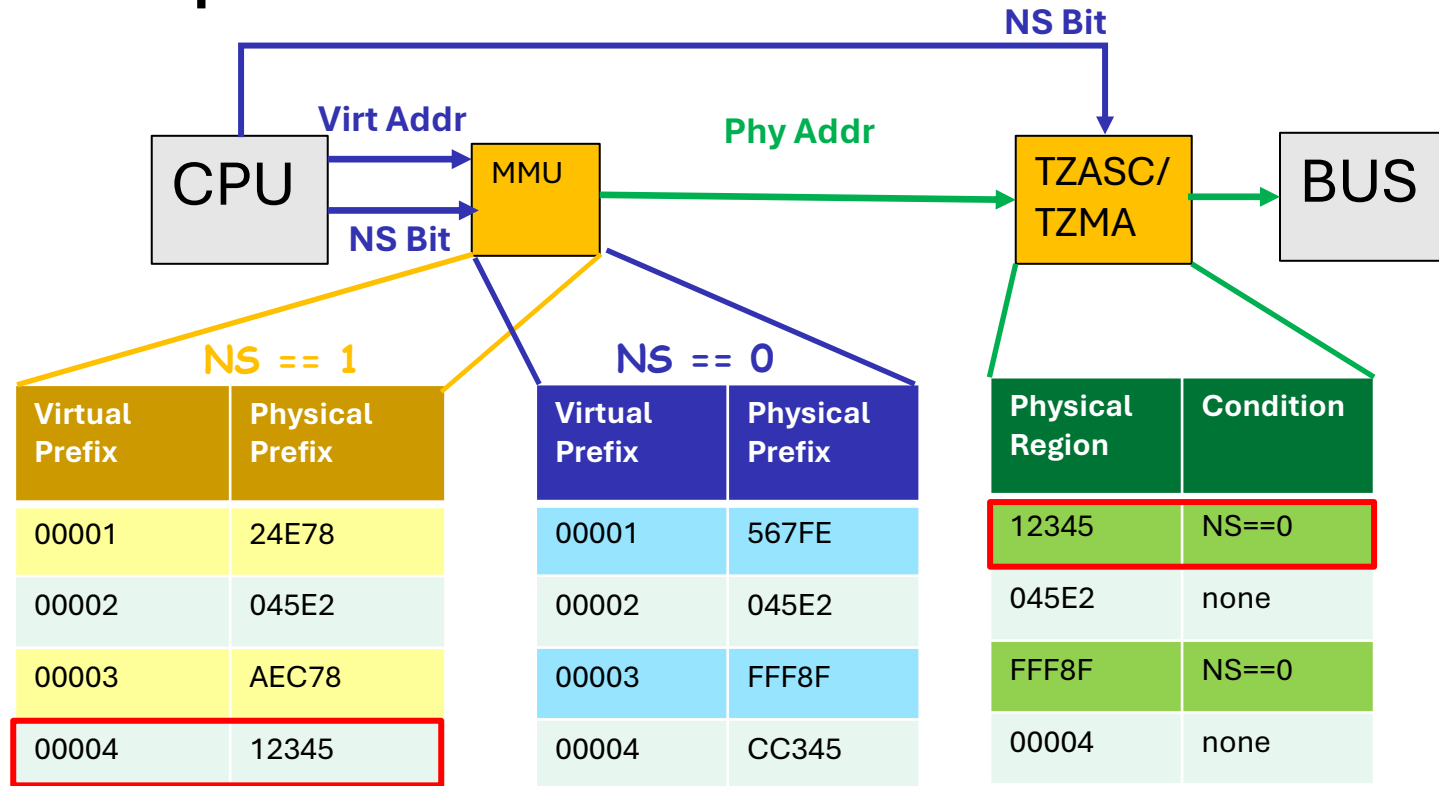
Accesses 0x045E2123

Accesses 0xFFF8F456



ARM TrustZone – Isolation

Some examples....



Access 1: NS = 1; Virt Addr = 0x00002123

Access 2: NS = 0; Virt Addr = 0x00003456

Access 3: NS = 1; Virt Addr = 0x00004789

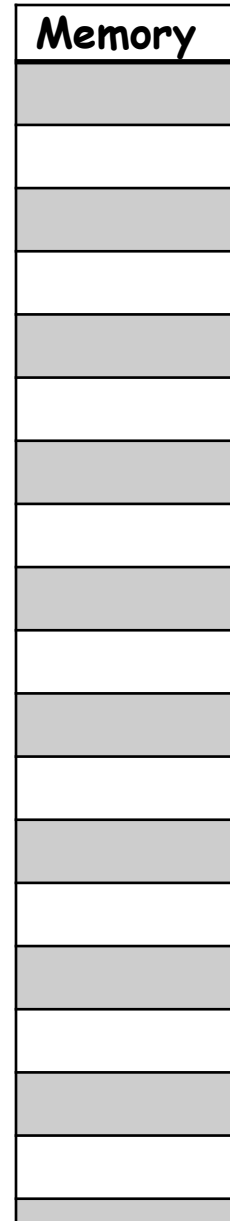
Access 4: NS = 1; Virt Addr = 0x00006333

Access 5: NS = 0; Virt Addr = 0x00002123

Accesses 0x045E2123

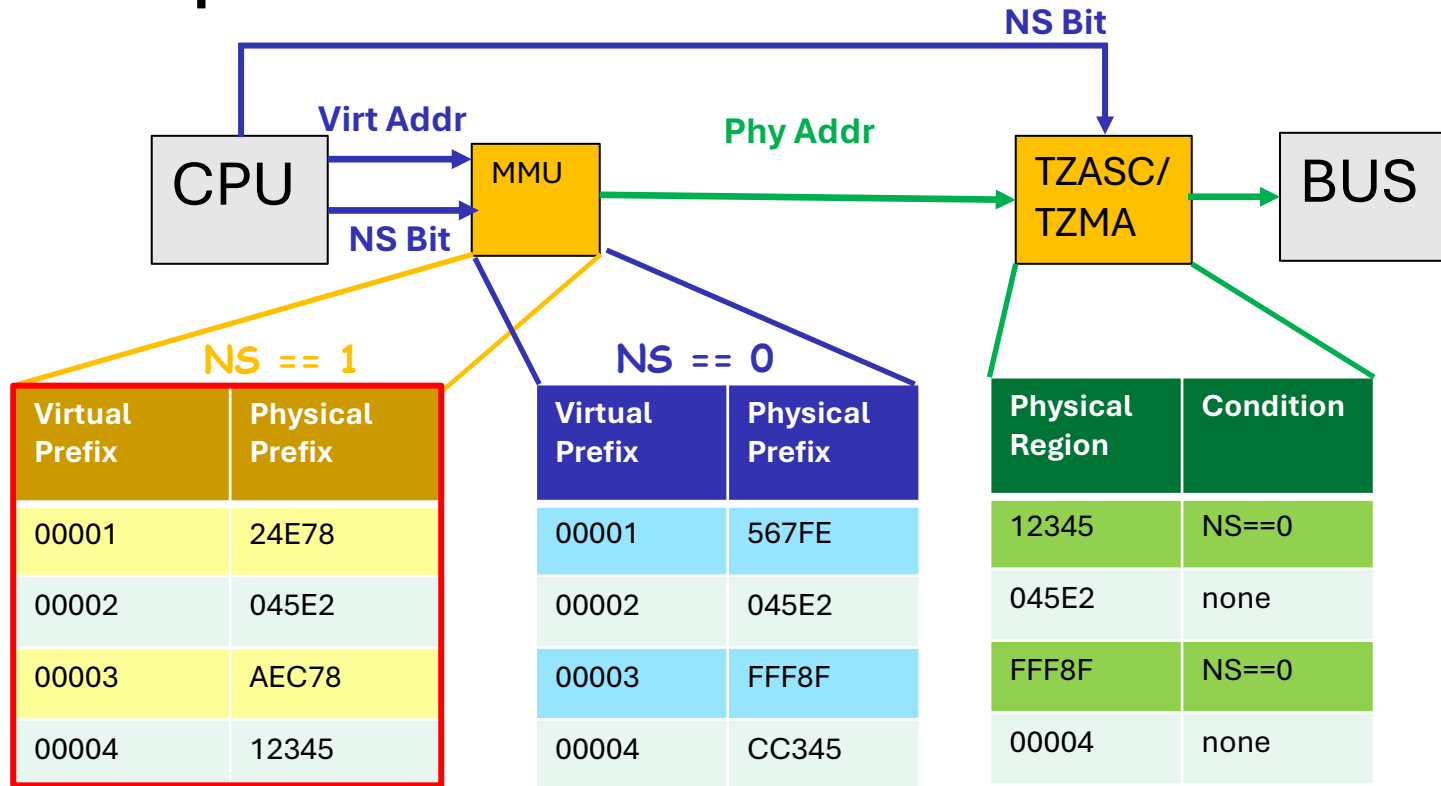
Accesses 0xFFF8F456

Discarded by TZASC/TZMA



ARM TrustZone – Isolation

Some examples....



Access 1: NS = 1; Virt Addr = 0x00002123

Access 2: NS = 0; Virt Addr = 0x00003456

Access 3: NS = 1; Virt Addr = 0x00004789

Access 4: NS = 1; Virt Addr = 0x00006333

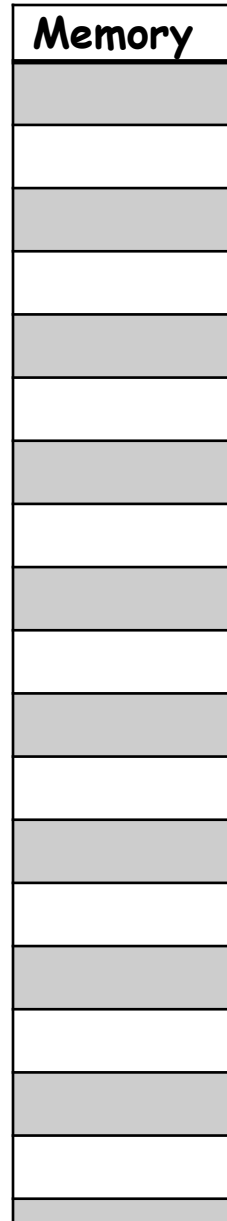
Access 5: NS = 0; Virt Addr = 0x00002123

Accesses 0x045E2123

Accesses 0xFFF8F456

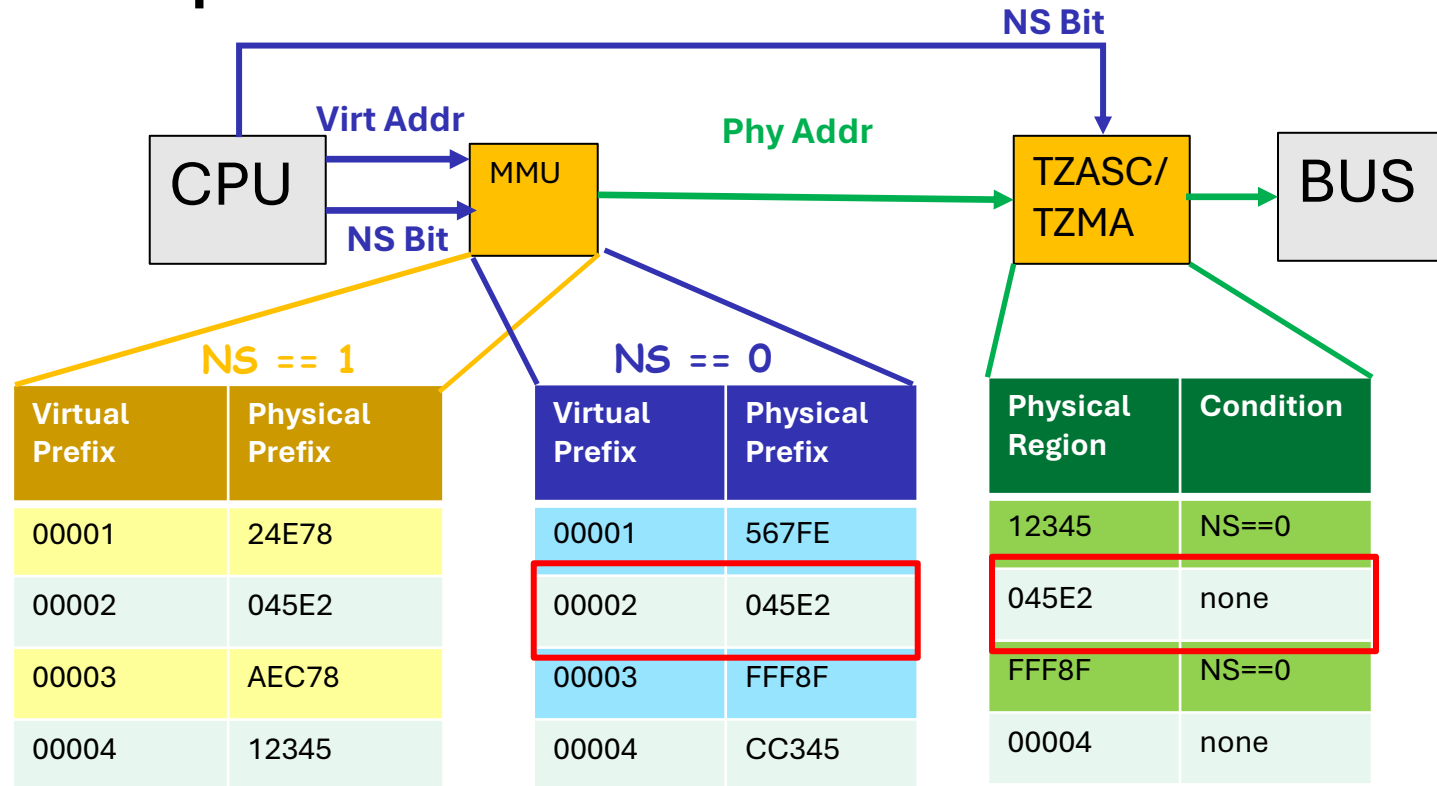
Discarded by TZASC/TZMA

Discarded by MMU



ARM TrustZone – Isolation

Some examples....



Access 1: NS = 1; Virt Addr = 0x00002123

Access 2: NS = 0; Virt Addr = 0x00003456

Access 3: NS = 1; Virt Addr = 0x00004789

Access 4: NS = 1; Virt Addr = 0x00006333

Access 5: NS = 0; Virt Addr = 0x00002123

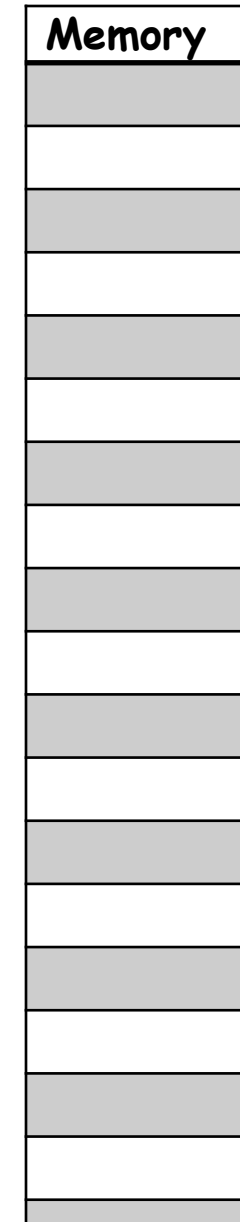
Accesses 0x045E2123

Accesses 0xFFF8F456

Discarded by TZASC/TZMA

Discarded by MMU

Accesses 0x045E2123



ARM TrustZone – Isolation

Two key questions:

- Who configures the TZASC/TMA table?
- Who controls the NS bit value?

ARM TrustZone – Isolation

Two key questions:

- Who configures the TZASC/TMA table?
 - **Secure World code:** first configuration after boot! It is more privileged than Normal World
 - Secure world executes first and configures TZMA/TZASC before launching the normal world and rich OS.
 - **Security of TrustZone requires TrustZone-aware Secure boot!**
- Who controls the NS bit value?

ARM TrustZone – Isolation

Two key questions:

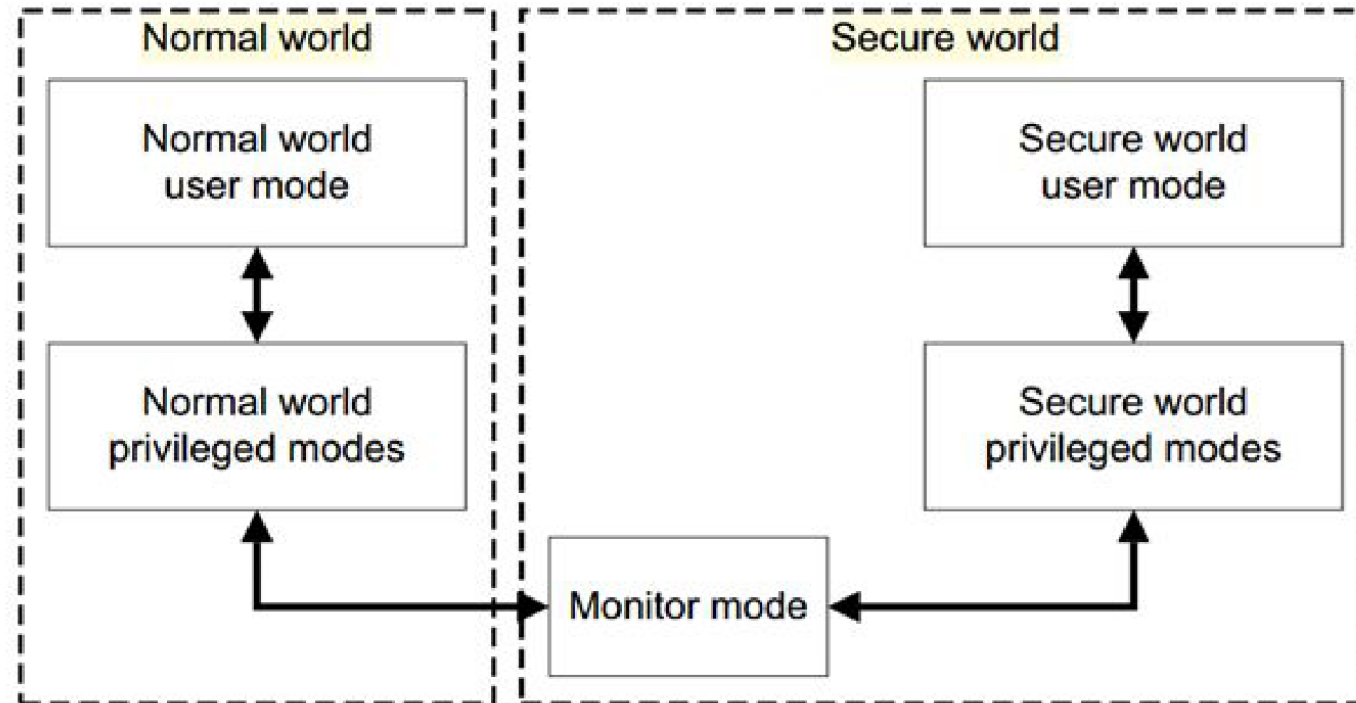
- Who configures the TZASC/TMA table?
 - **Secure World code:** first configuration after boot! It is more privileged than Normal World
 - Secure world executes first and configures TZMA/TZASC before launching the normal world and rich OS.
 - **Security of TrustZone requires TrustZone-aware Secure boot!**
- Who controls the NS bit value?
 - The **CPU in hardware**
 - From normal world, NS bit can only be changed (1 → 0) by issuing a **Security Monitor Call (SMC)**
 - **SMC atomically gives control to Secure World and sets NS=0**
 - SMC jumps to **Security Monitor** that performs context switch between the worlds
 - **The NS bit is set back to NS=1 before returning to Normal World**

ARM TrustZone – Isolation

Security Monitor and SMC:

Switching between worlds requires a security monitor call (SMC)

The Security Monitor is part of the Secure World's TCB



ARM TrustZone – Isolation

Caching in TrustZone

The problem: the CPU, and consequently the cache, must be securely shared between worlds

The TZMA/TZASC split physical memory, but not the cache

ARM TrustZone – Isolation

Caching in TrustZone

The problem: the CPU, and consequently the cache, must be securely shared between worlds

The TZMA/TZASC split physical memory, but not the cache

So without any additional measures, the following is a possibility:

1. Secure World is running
2. Secure World transfers context back to the Normal World
3. Normal World reads the same cached address used by Secure World
4. **Data leaked! → Isolation is broken!**

ARM TrustZone – Isolation

Caching in TrustZone

How to handle this problem?

Naïve solution: Remove the cache => secure, but extremely slow.

Alternative: Always flush the cache when switching worlds => secure, but still pretty slow.

ARM TrustZone – Isolation

Caching in TrustZone

How to handle this problem?

Naïve solution: Remove the cache => secure, but extremely slow.

Alternative: Always flush the cache when switching worlds => secure, but still pretty slow.

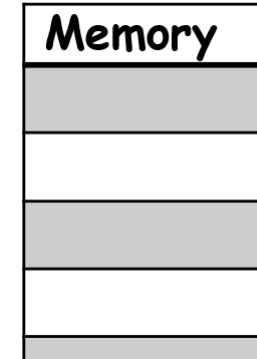
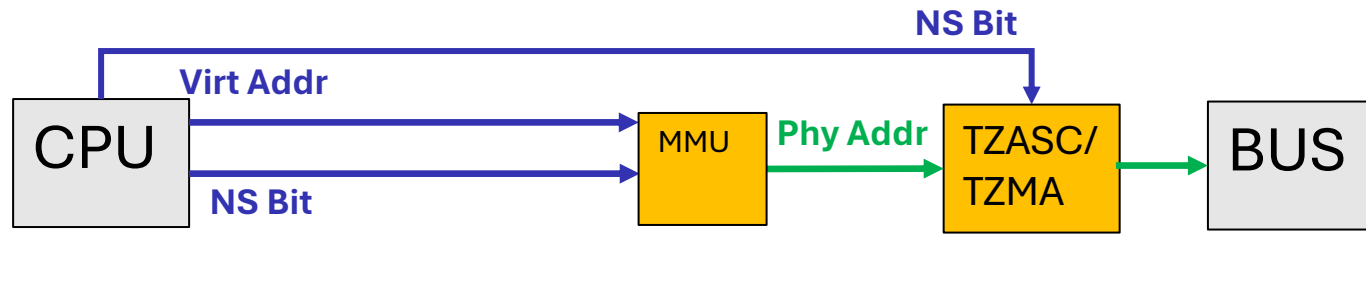
TrustZone's solution:

Include the NS bit in the cache look-up => no need to flush the cache!

- Allows for fast world switching
- Cached data may be kept across successive switches

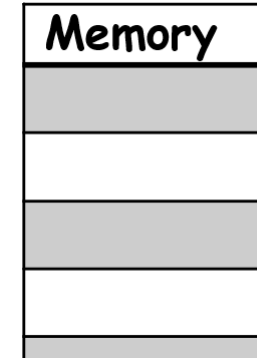
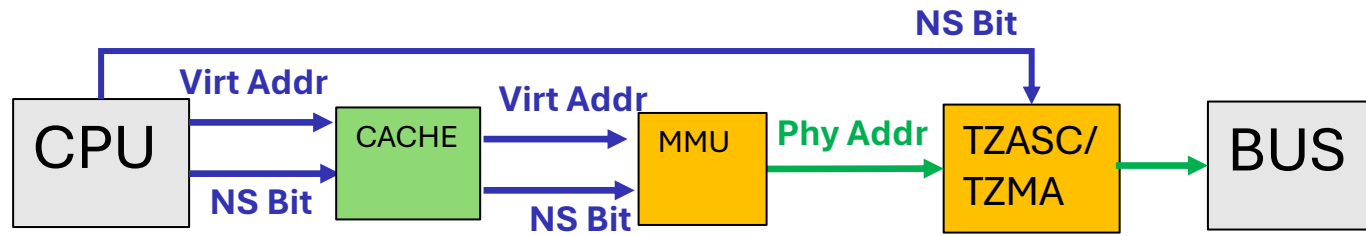
ARM TrustZone – Isolation

Caching in TrustZone



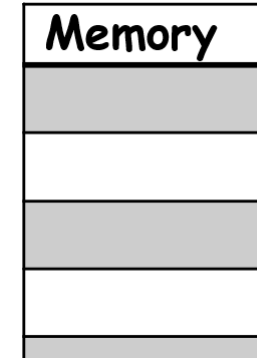
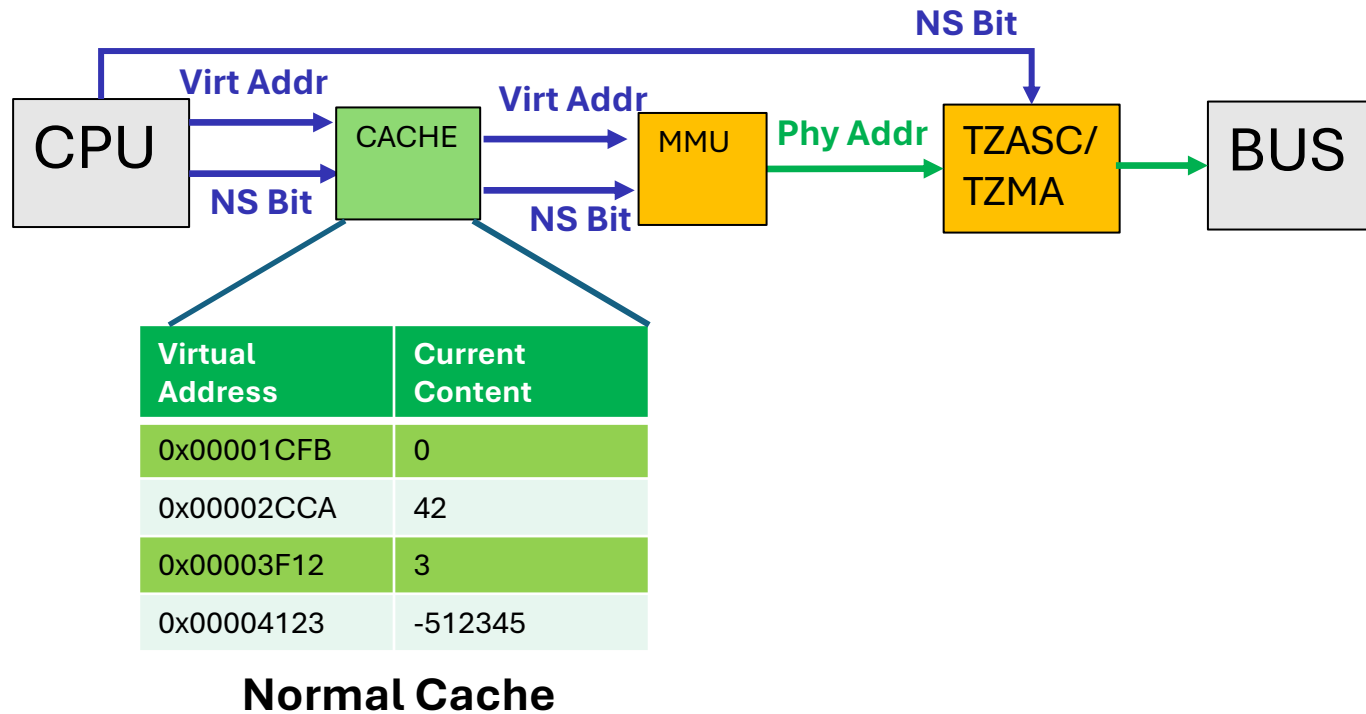
ARM TrustZone – Isolation

Caching in TrustZone



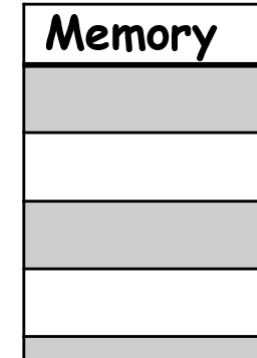
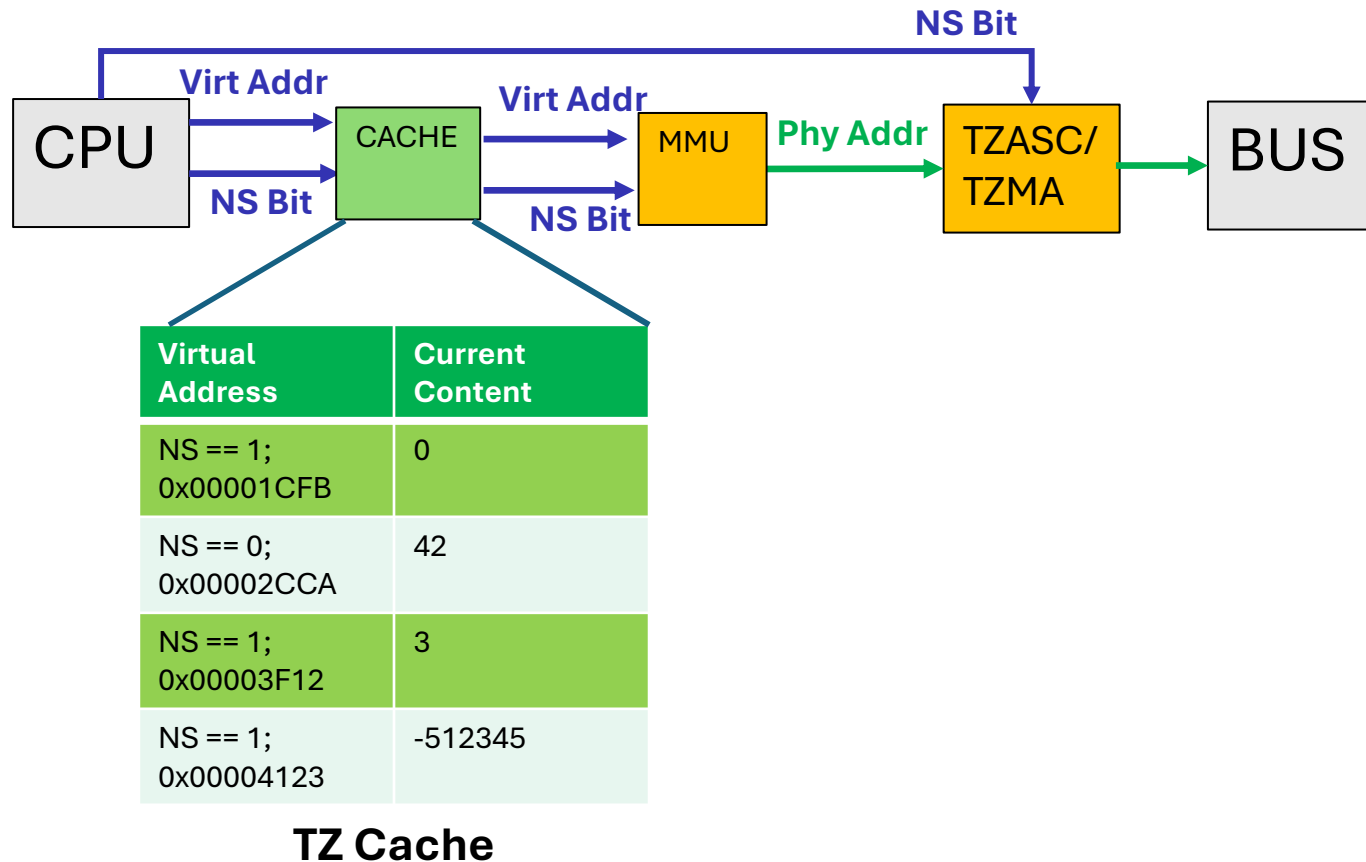
ARM TrustZone – Isolation

Caching in TrustZone



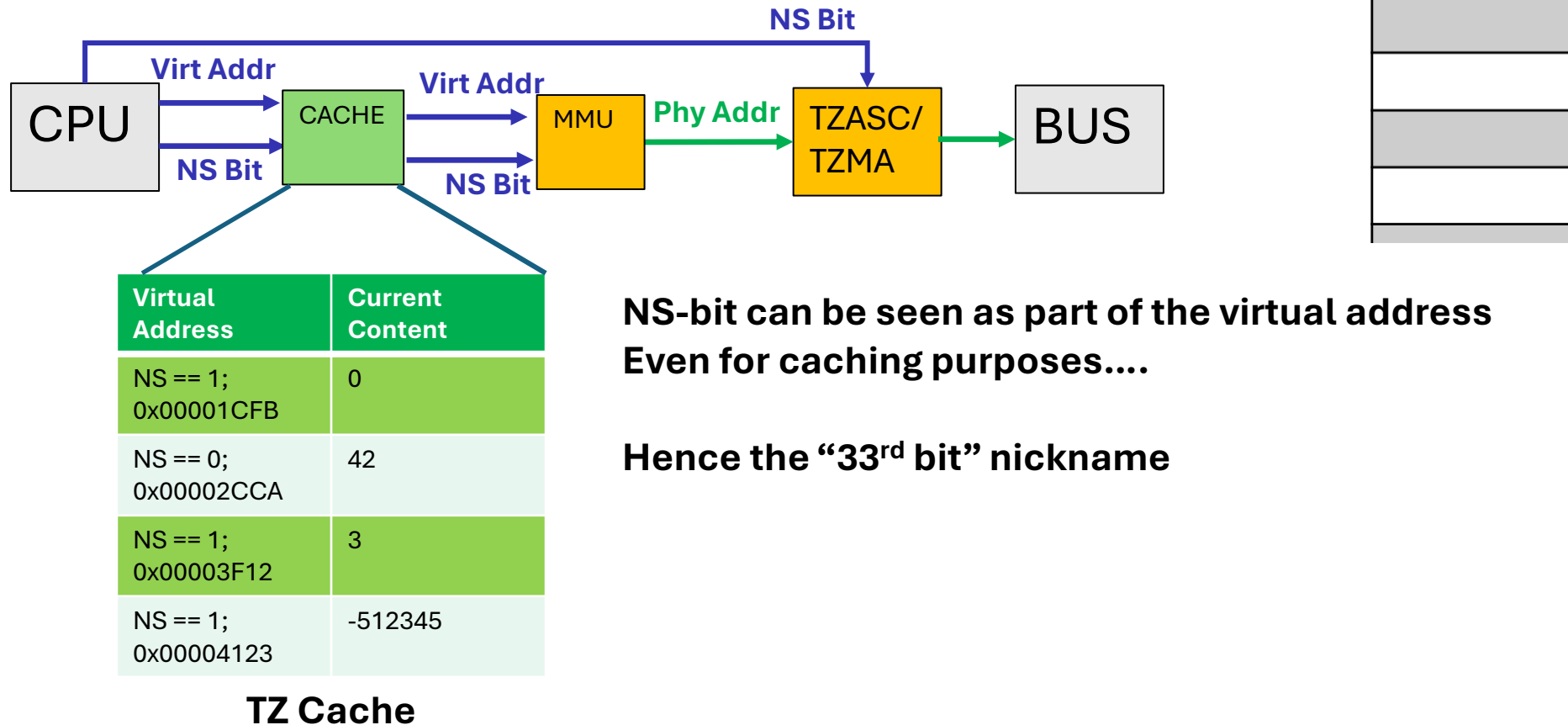
ARM TrustZone – Isolation

Caching in TrustZone



ARM TrustZone – Isolation

Caching in TrustZone

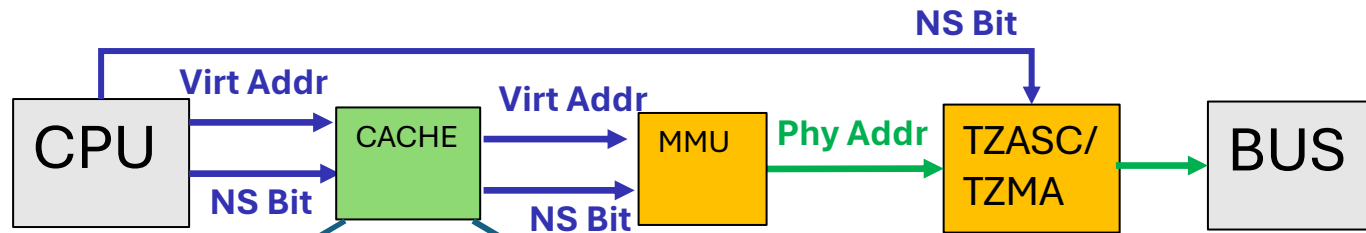


**NS-bit can be seen as part of the virtual address
Even for caching purposes....**

Hence the “33rd bit” nickname

ARM TrustZone – Isolation

Caching in TrustZone



Virtual Address	Current Content
NS == 1; 0x00001CFB	0
NS == 0; 0x00002CCA	42
NS == 1; 0x00003F12	3
NS == 1; 0x00004123	-512345

TZ Cache

**NS-bit can be seen as part of the virtual address
Even for caching purposes....**

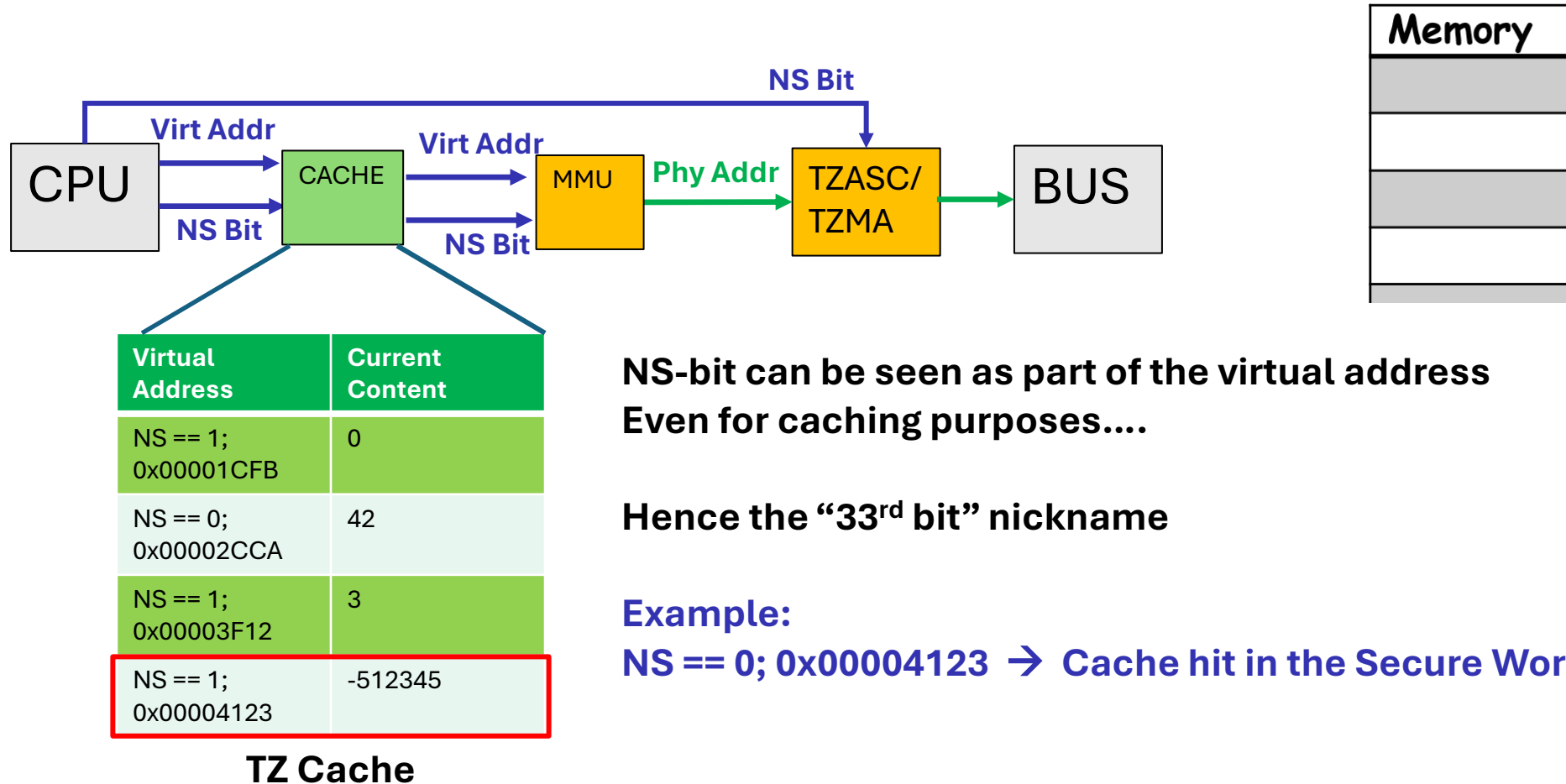
Hence the “33rd bit” nickname

Example:

NS == 1; 0x00001CFB → Cache hit in the Normal World

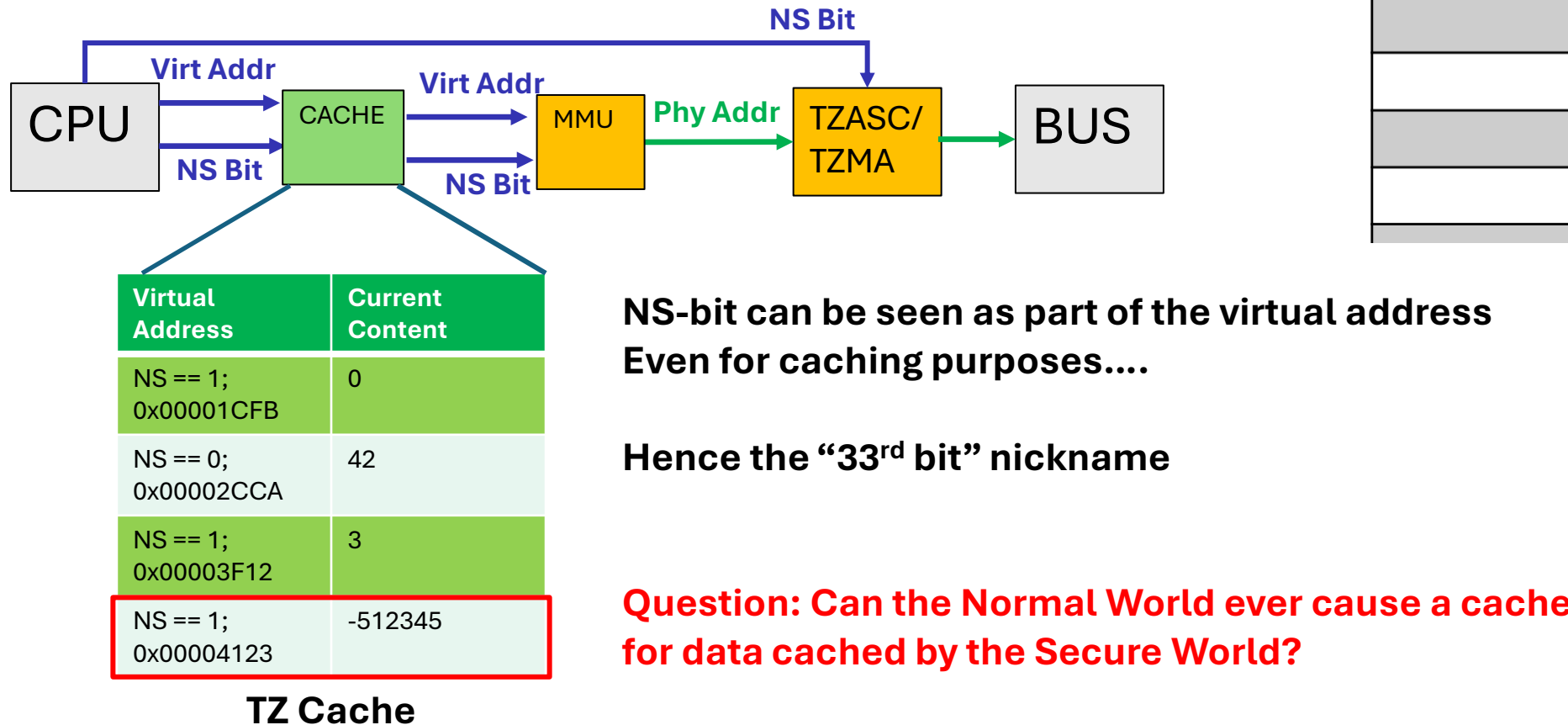
ARM TrustZone – Isolation

Caching in TrustZone



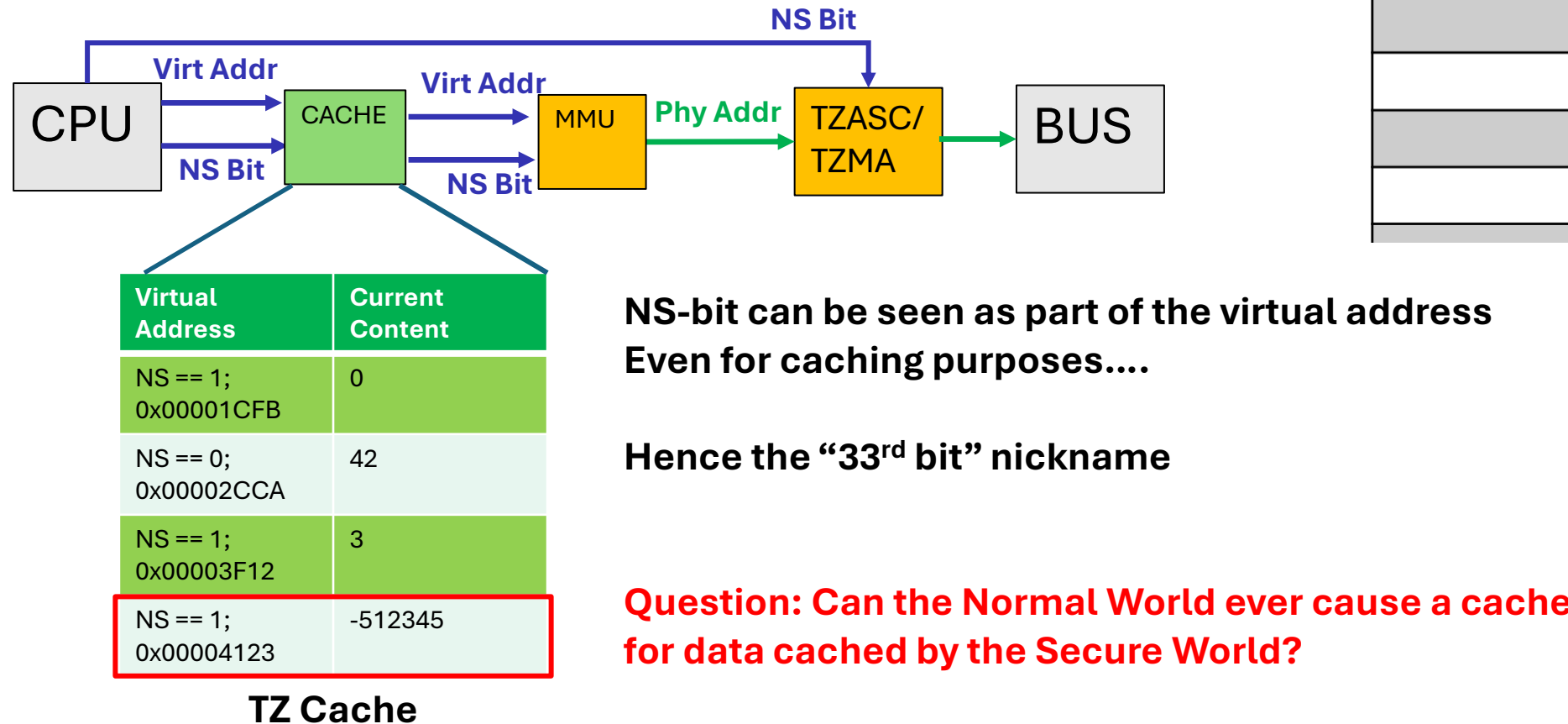
ARM TrustZone – Isolation

Caching in TrustZone



ARM TrustZone – Isolation

Caching in TrustZone



NS-bit can be seen as part of the virtual address
Even for caching purposes....

Hence the “33rd bit” nickname

Question: Can the Normal World ever cause a cache hit for data cached by the Secure World?

Normal World can never cause a cache hit for data cached by the Secure World execution.

ARM TrustZone – Isolation

Caching in TrustZone

Also, the MMU has a cache:

- Called the **TLB: Translation Lookaside Buffer**
- Same principle for TrustZone's CPU cache

ARM TrustZone – Isolation

Summary of Isolation in TrustZone:

In main memory:

- Physical isolation implemented by TZMA/TZASC based on NS-bit

Within CPU cache and TLB:

- Propagate NS-bit through every virtual address look-up
- NS-bit is “33rd bit”

Reminder:

- NS bit value is controlled by CPU hardware. Only way to set it to 0 is by calling SMC, which also gives control to TrustZone’s trusted Security Monitor

ARM TrustZone Overview

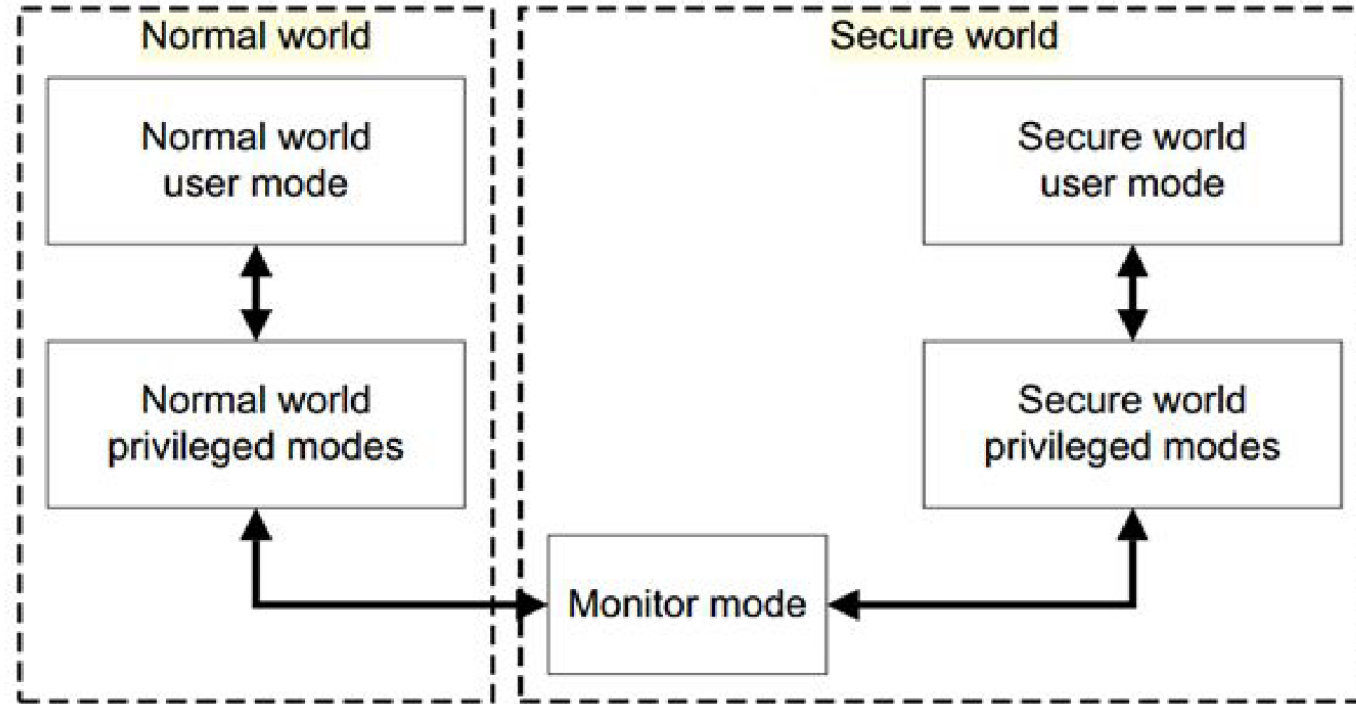
Topics:

- Isolation in TrustZone
- Secure Monitor Calls (SMC) – Invocation of Secure World code
- Android

ARM TrustZone – SMC

Revisiting the system flow:

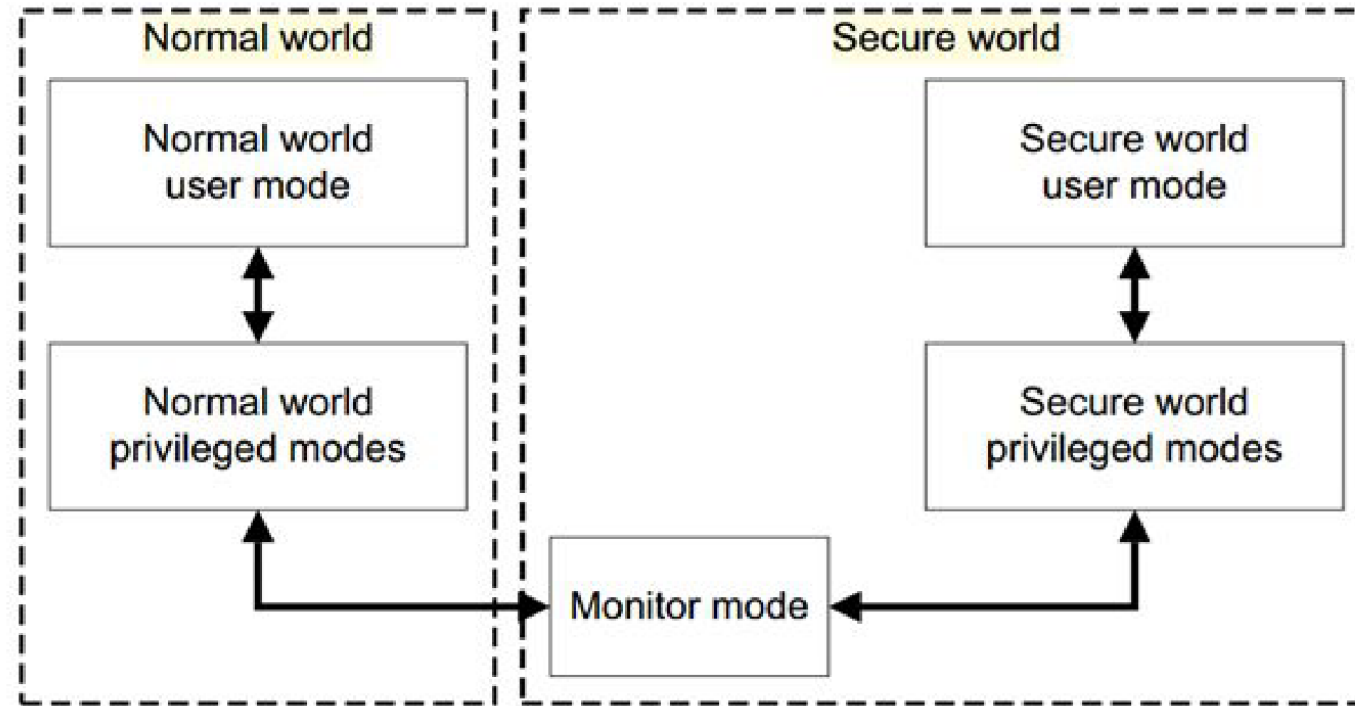
Controlled enter and exit from the Secure World



ARM TrustZone – SMC

Revisiting the system flow:

Controlled enter and exit from the Secure World



The moment the CPU Flips the NS-bit

ARM TrustZone – SMC

SMC Instruction – Calling Convention

- A function identifier (32-bits) is passed using CPU register (R0)
 - Can be used to tell the security monitor which Trusted App is the destination of this call

ARM TrustZone – SMC

SMC Instruction – Calling Convention

- A function identifier (32-bits) is passed using CPU register (R0)
 - Can be used to tell the security monitor which Trusted App is the destination of this call
- SMC Arguments are passed in registers R1-R7
 - Inputs destined to the secure world

ARM TrustZone – SMC

SMC Instruction – Calling Convention

- A function identifier (32-bits) is passed using CPU register (R0)
 - Can be used to tell the security monitor which Trusted App is the destination of this call
- SMC Arguments are passed in registers R1-R7
 - Inputs destined to the secure world
- Results are also returned to normal world using registers R0-R7

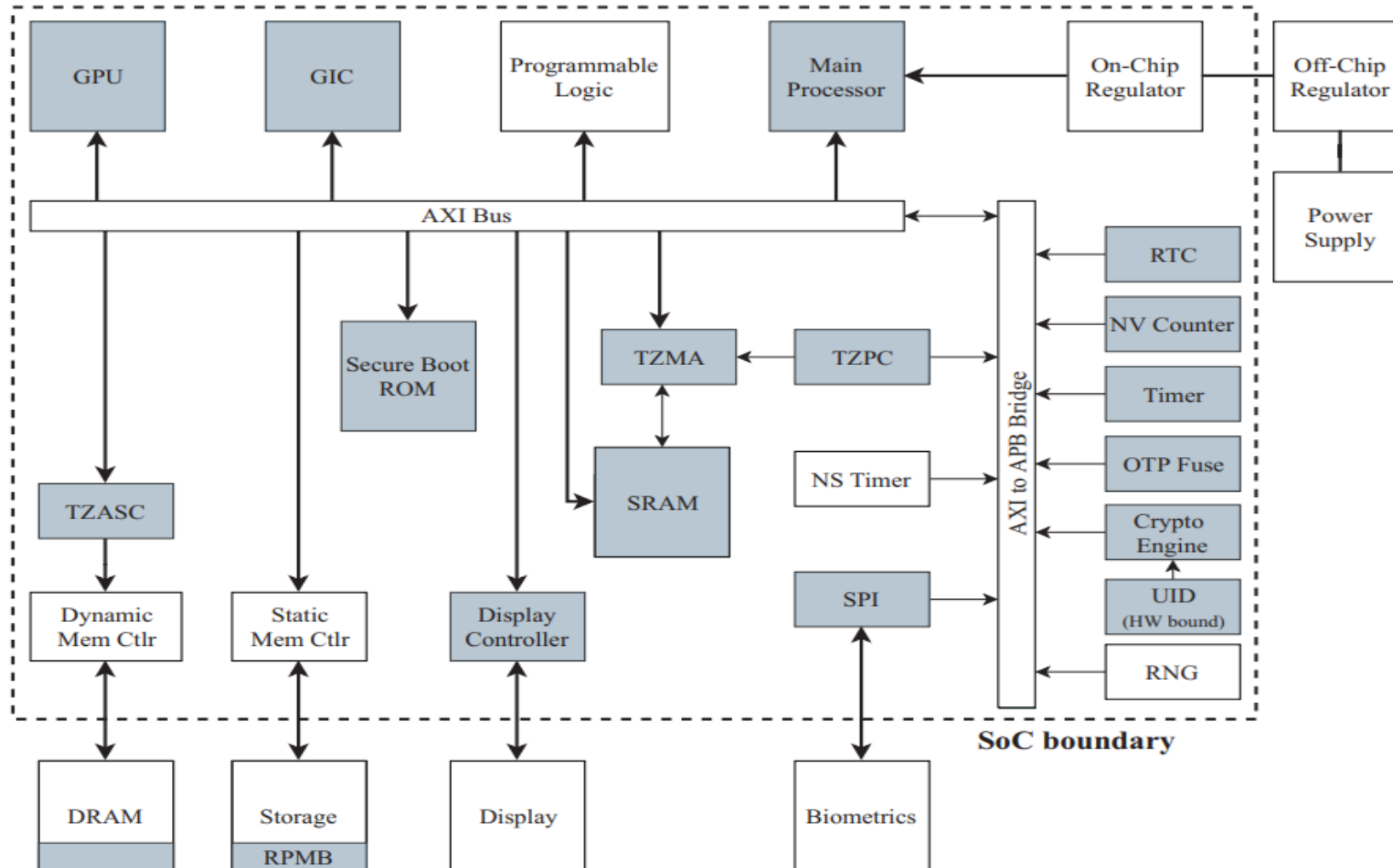
ARM TrustZone – SMC

SMC Instruction – Calling Convention

- A function identifier (32-bits) is passed using CPU register (R0)
 - Can be used to tell the security monitor which Trusted App is the destination of this call
- SMC Arguments are passed in registers R1-R7
 - Inputs destined to the secure world
- Results are also returned to normal world using registers R0-R7
- **Convention:** not enforced by hardware anywhere
 - It is up to the Security Monitor to define its own behavior
 - Must then be followed/implemented by the SMC caller

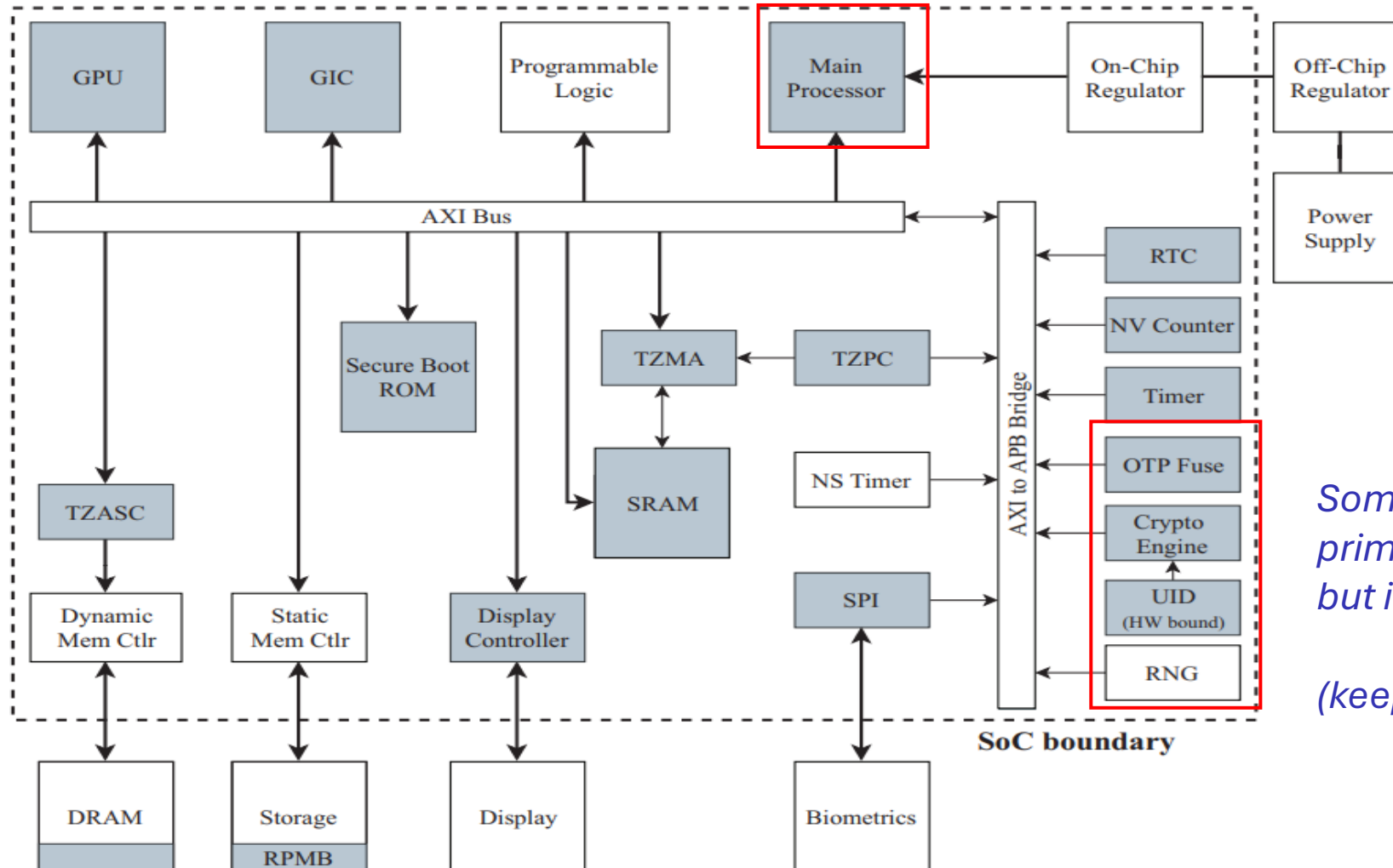
ARM TrustZone – Architecture

The whole beast:



ARM TrustZone – Architecture

The whole beast:



Some cryptographic primitives outside CPU but in SoC

(keep in mind....)

ARM TrustZone – Architecture

Important reminders about TrustZone's design:

- Secure boot must guarantee that the Secure World runs first
 - After Secure World completes secure boot → “ACTIVE”
- Availability
 - Boots first!
 - Also, resources assigned to secure world have priority (e.g., interrupts via TrustZone's GIC)
 - Different from TPM and SGX → Has an “active” characteristic

ARM TrustZone Overview

Topics:

- Isolation in TrustZone
- Secure Monitor Calls (SMC) – Invocation of Secure World code
- Android

ARM TrustZone – Android

Provides runtime environment built atop TrustZone

- Android OS & Apps → in the normal world
- Trusted OS & Trusted Apps → in the secure world

Features of interest:

- Key store

ARM TrustZone – Android

Android Key Store:

Protects key material from unauthorized use in two ways. First it .. Prevents the extraction of keys from application processes and from the Android device, and Second it makes apps specify the authorized use of their keys within the device and enforces those restrictions outside of the app's processes.

ARM TrustZone – Android

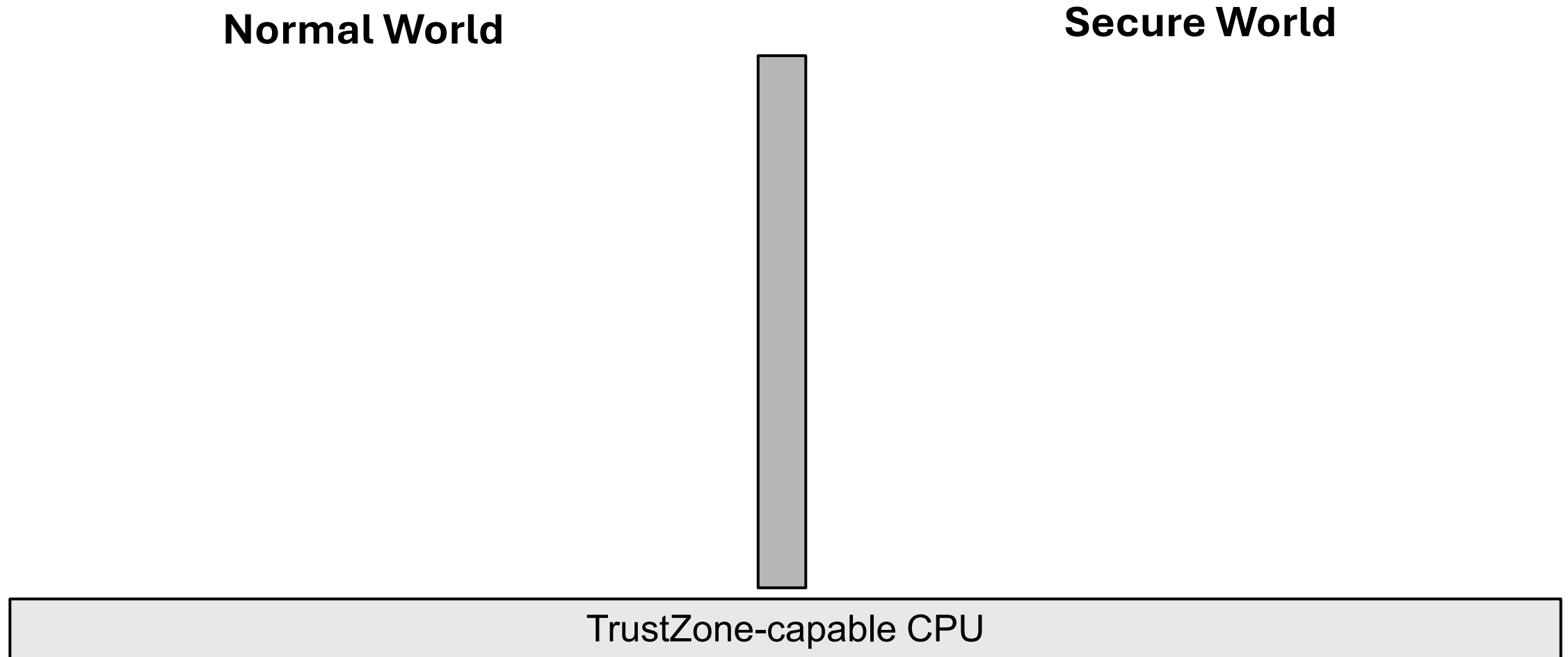
Android Key Store:

Extraction prevention is provided based on two security measures:

- **Key material never enters the application process**
 - Inputs for a operation that requires the key are fed into a “secure process”
 - Compromised App can use keys, but cannot extract the key itself
 - Confidentiality
- **Keys can be bound to the TEE**
 - Similar to “wrap key” to be used from a particular device
 - Integrity

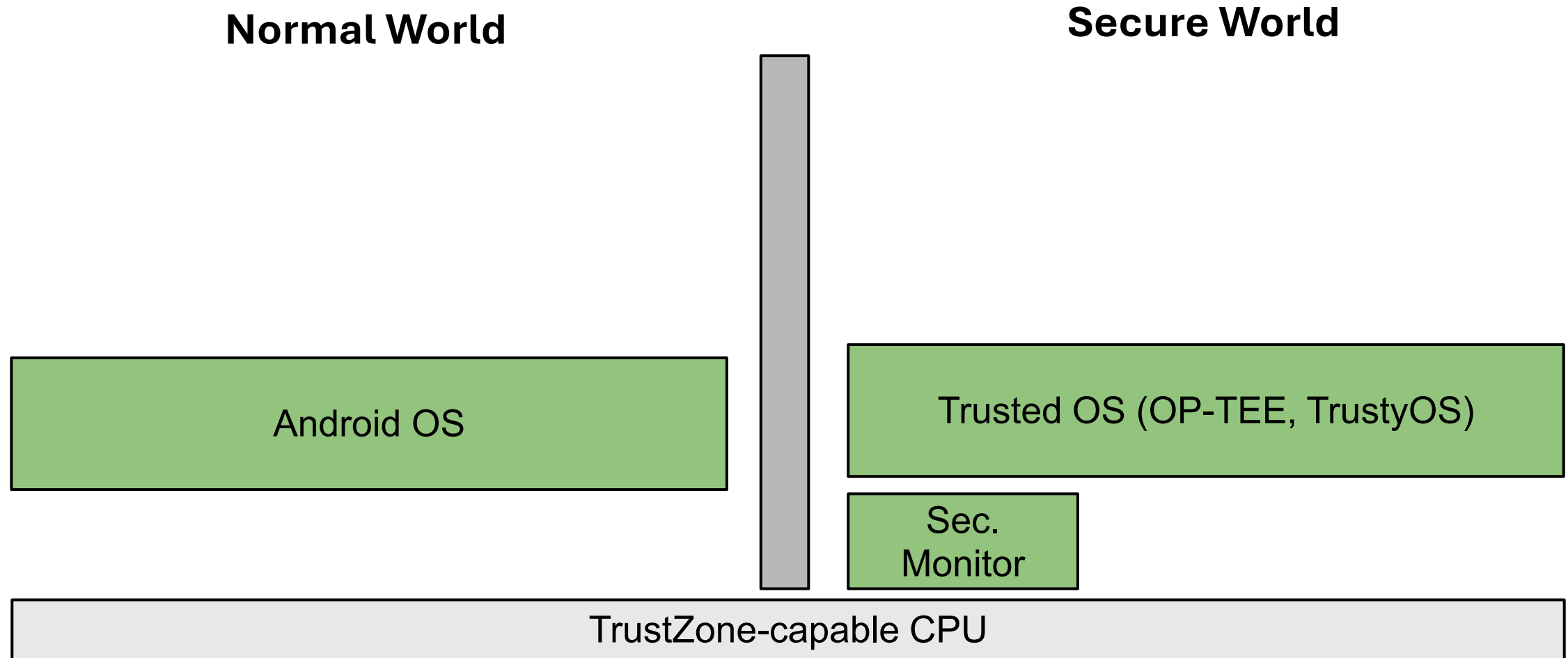
ARM TrustZone -- Android

How does Key store work? First, lets setup the key players...



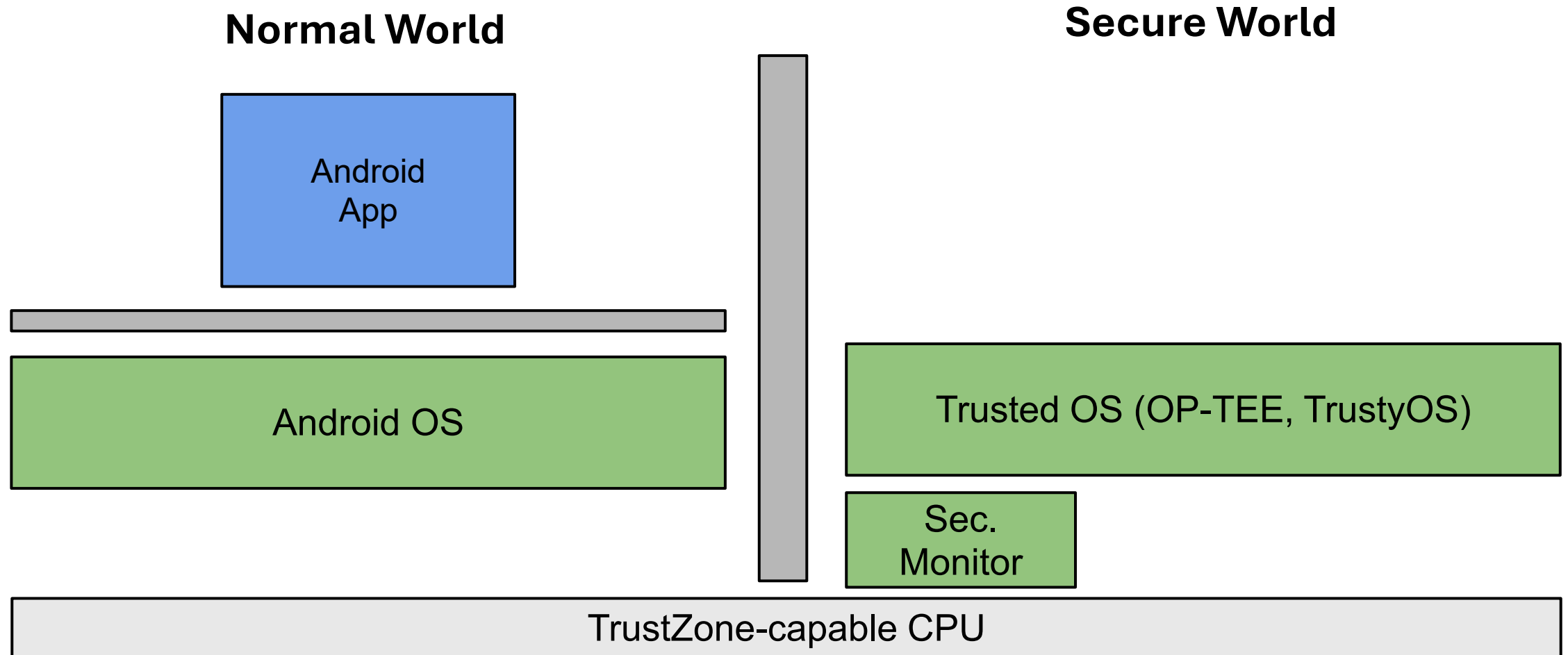
ARM TrustZone – Android

First OSes: Android OS in the Normal World, and a Trusted OS in the Secure World



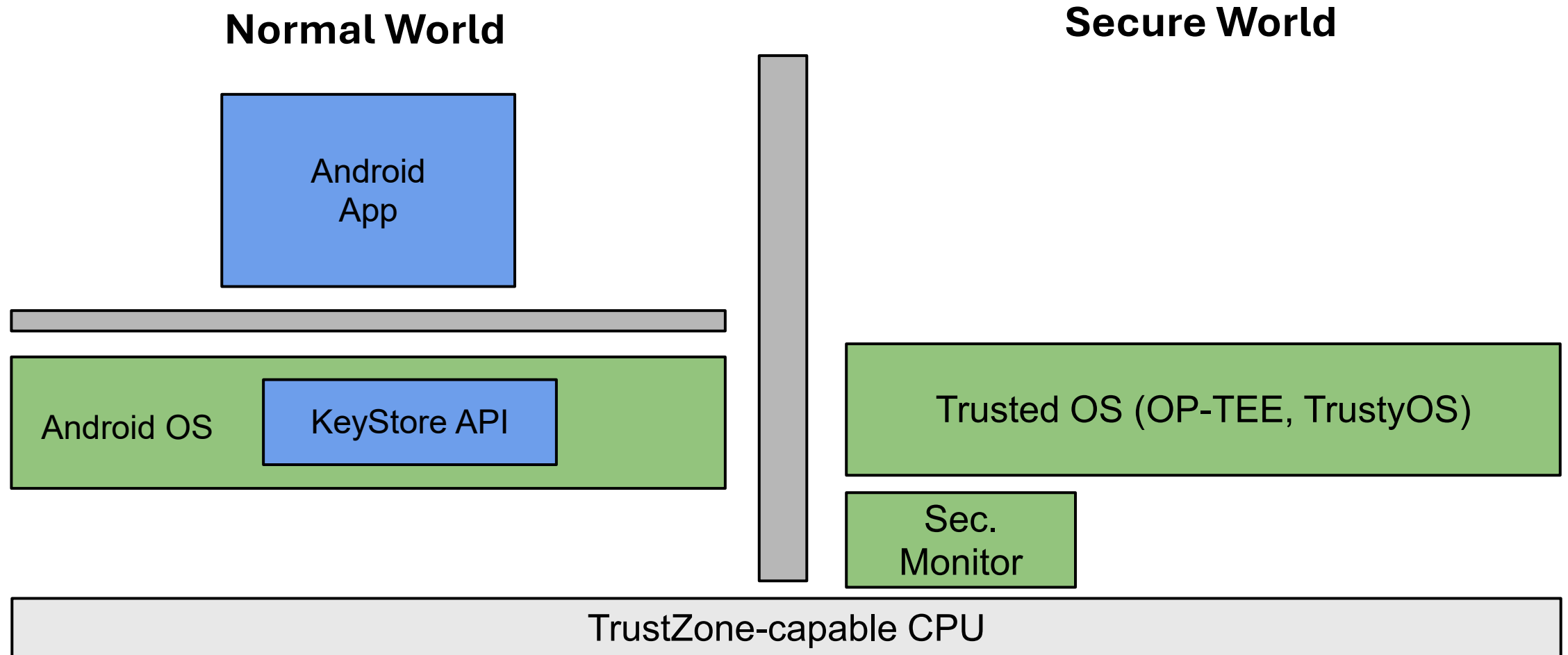
ARM TrustZone – Android

To simplify things, let's assume there is one Android app running in Normal World



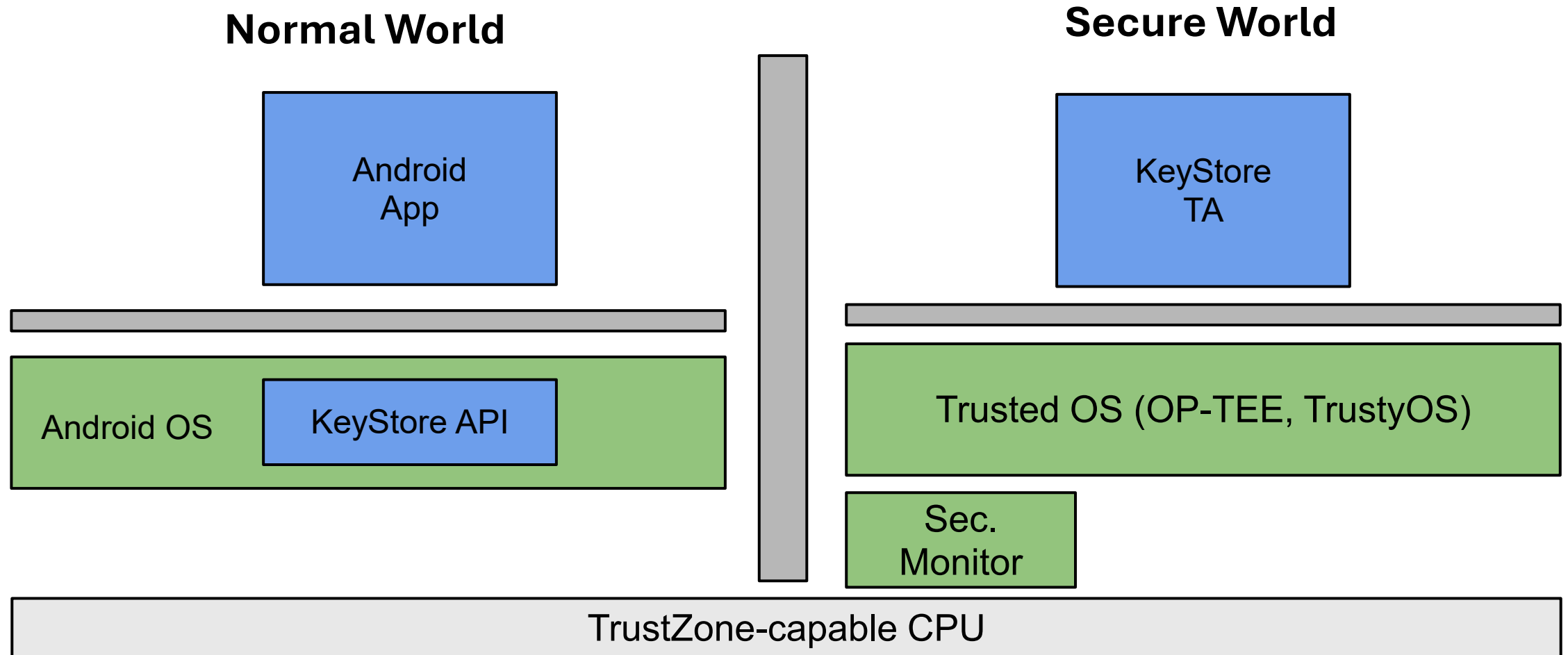
ARM TrustZone – Android

Within the Android OS is the KeyStore API



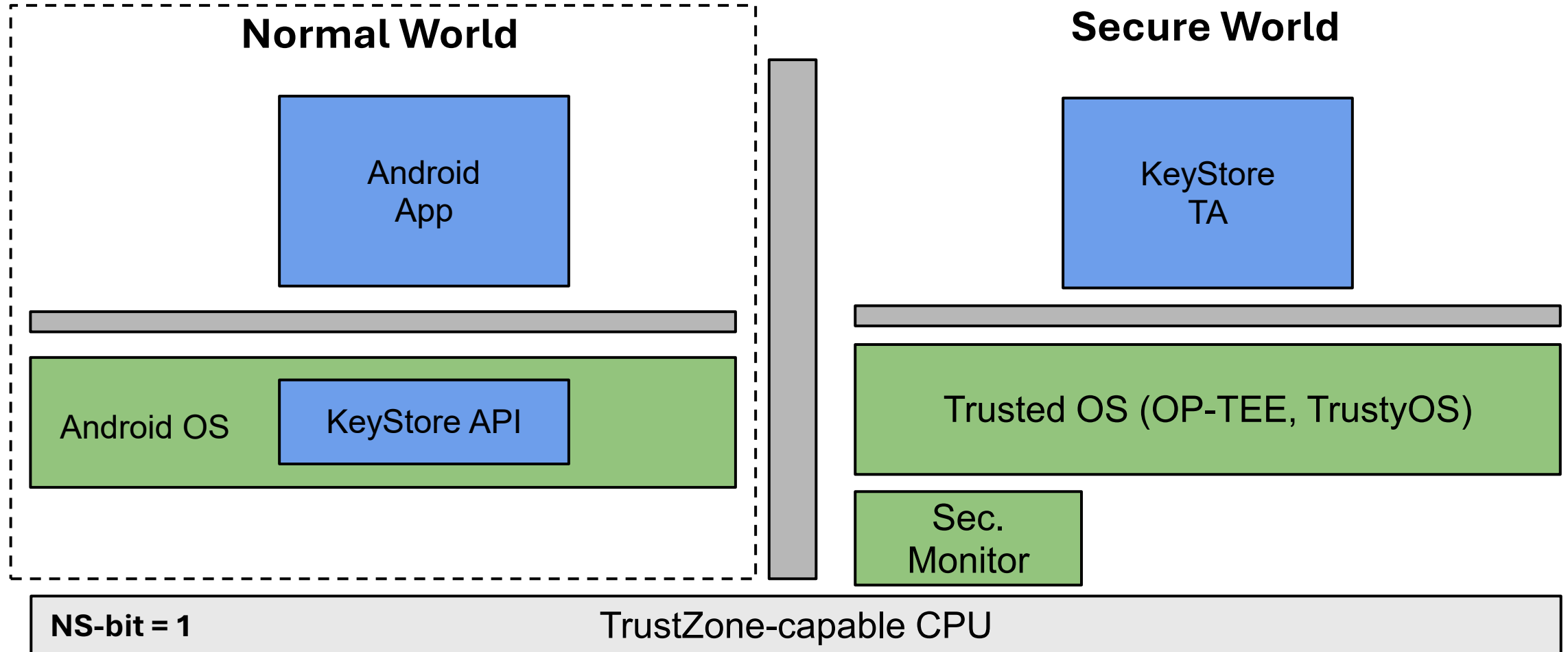
ARM TrustZone – Android

Within the Android OS is the KeyStore API, and a corresponding Keystore TA



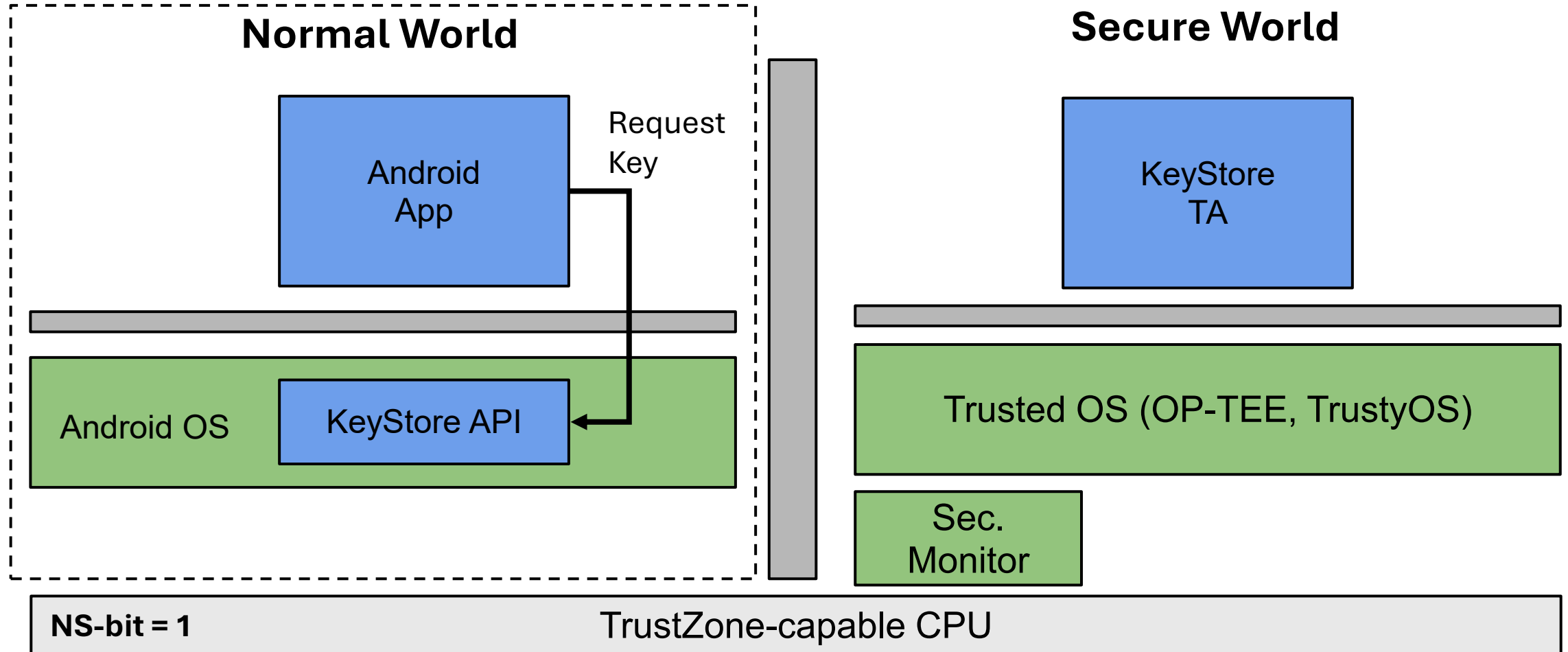
ARM TrustZone – Android

Lets assume the Android App is currently executing



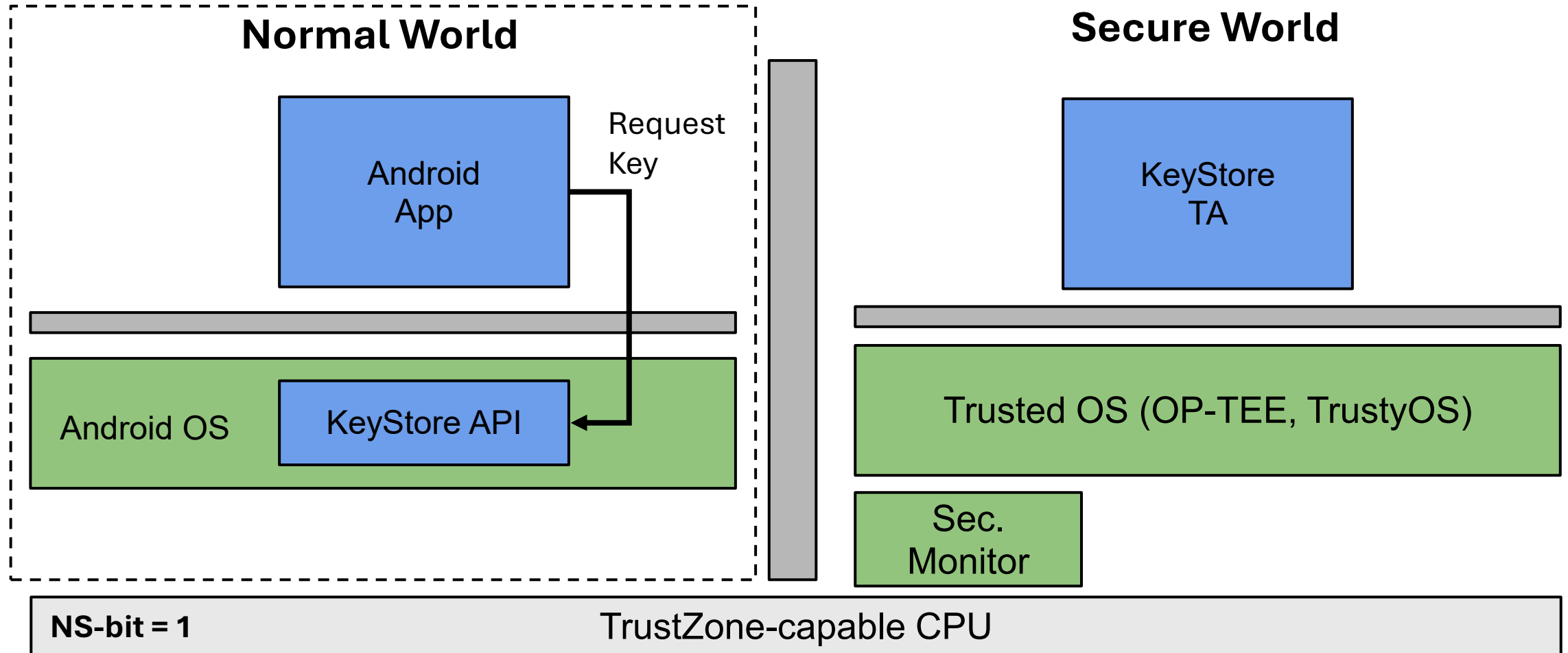
ARM TrustZone – Android

Android App will call the KeyStore API to request a key



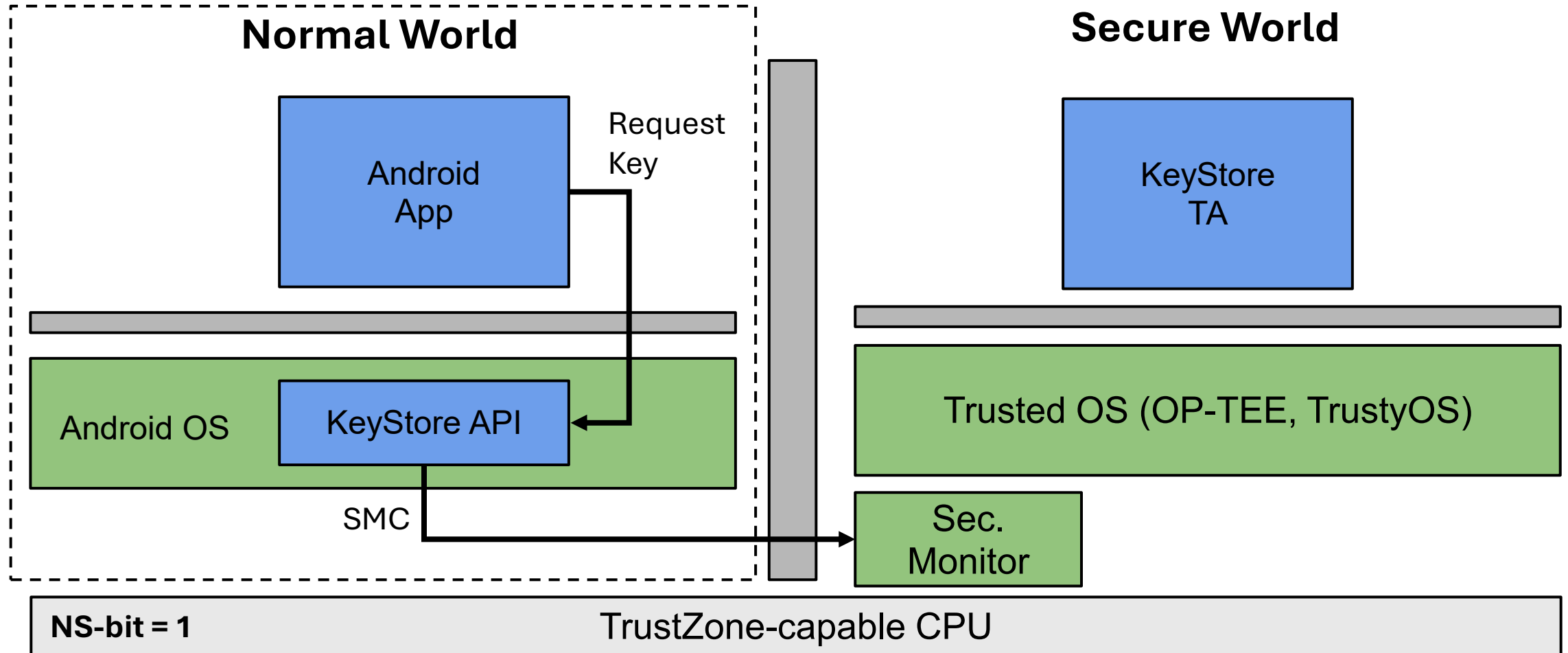
ARM TrustZone – Android

KeyStore API forwards the request to TEE How??



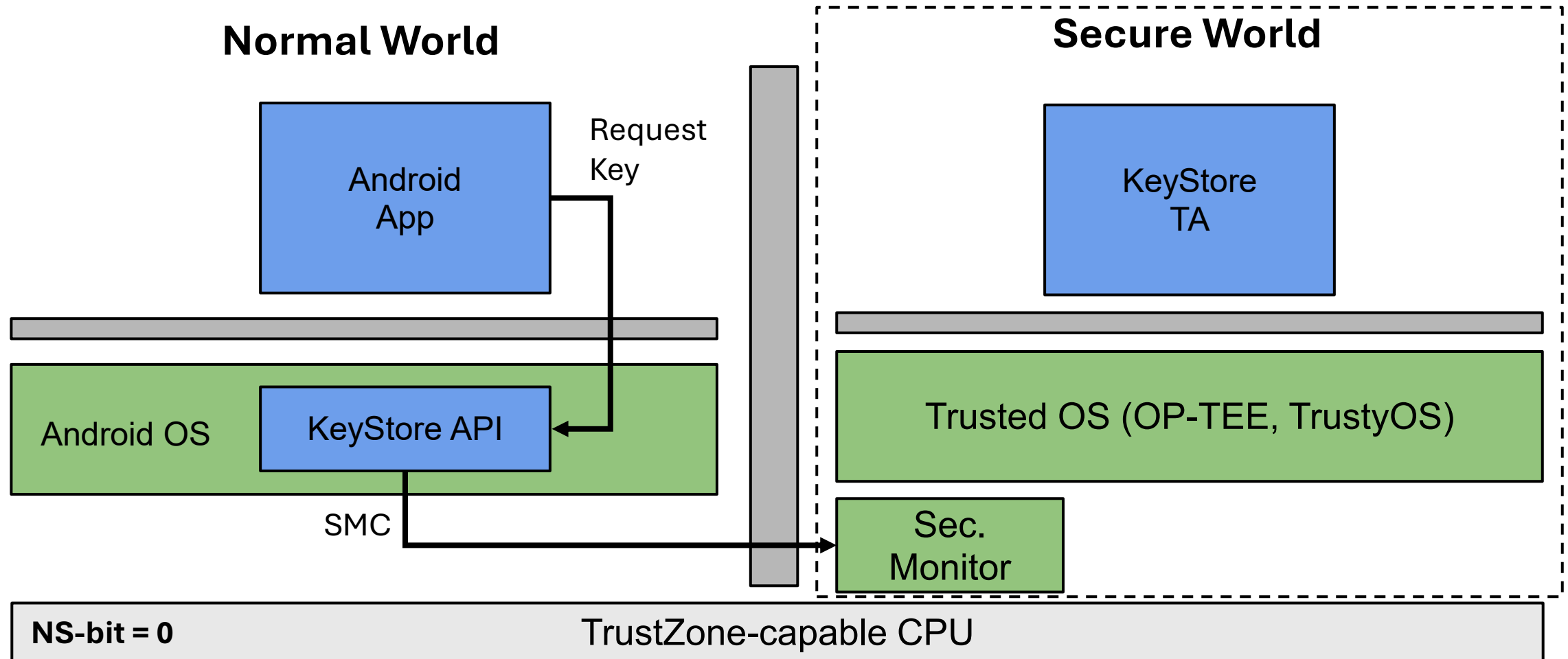
ARM TrustZone – Android

KeyStore API forwards the request to TEE How??



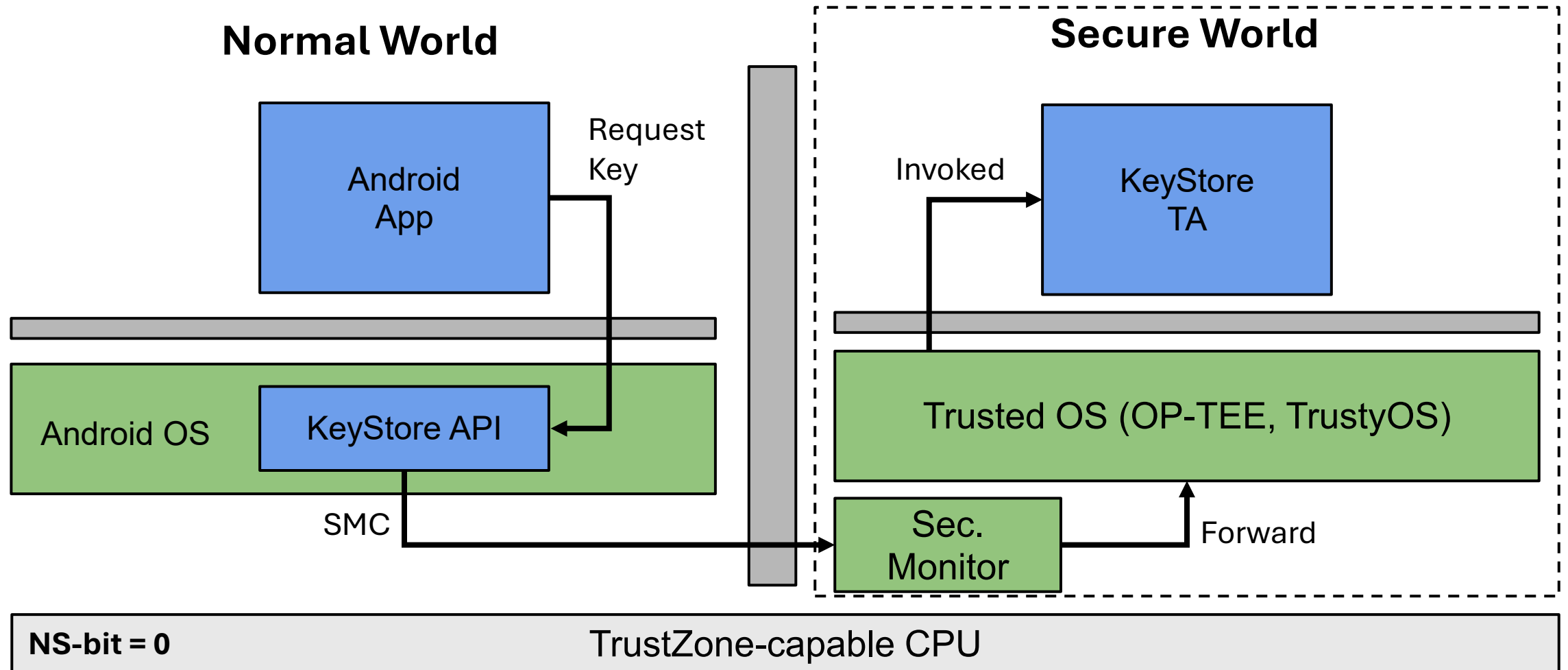
ARM TrustZone – Android

Security Monitor (atomically) switches from Normal World to Secure World



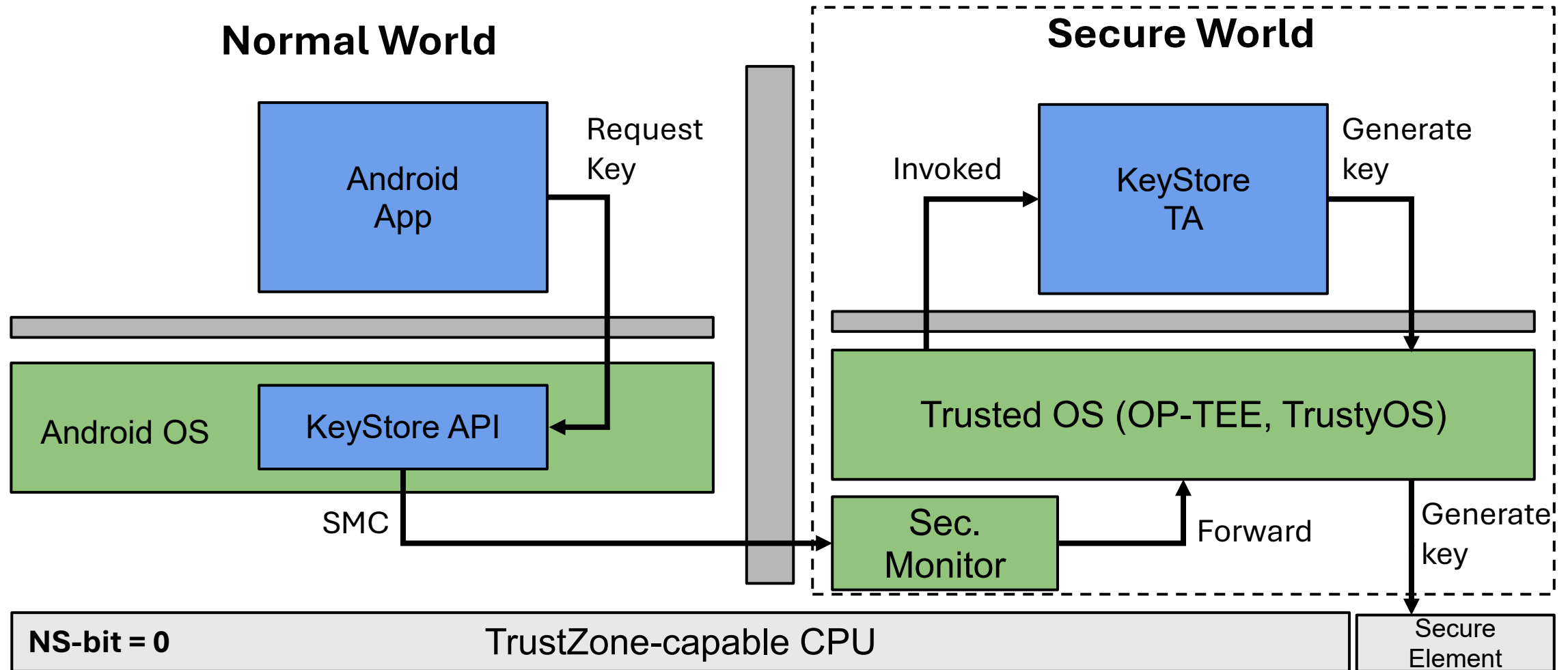
ARM TrustZone – Android

KeyStore TA is invoked. It then generates a key pair (e.g., AES, RSA) – how?



ARM TrustZone – Android

KeyStore TA is invoked. It then generates a key pair (e.g., AES, RSA) – how?



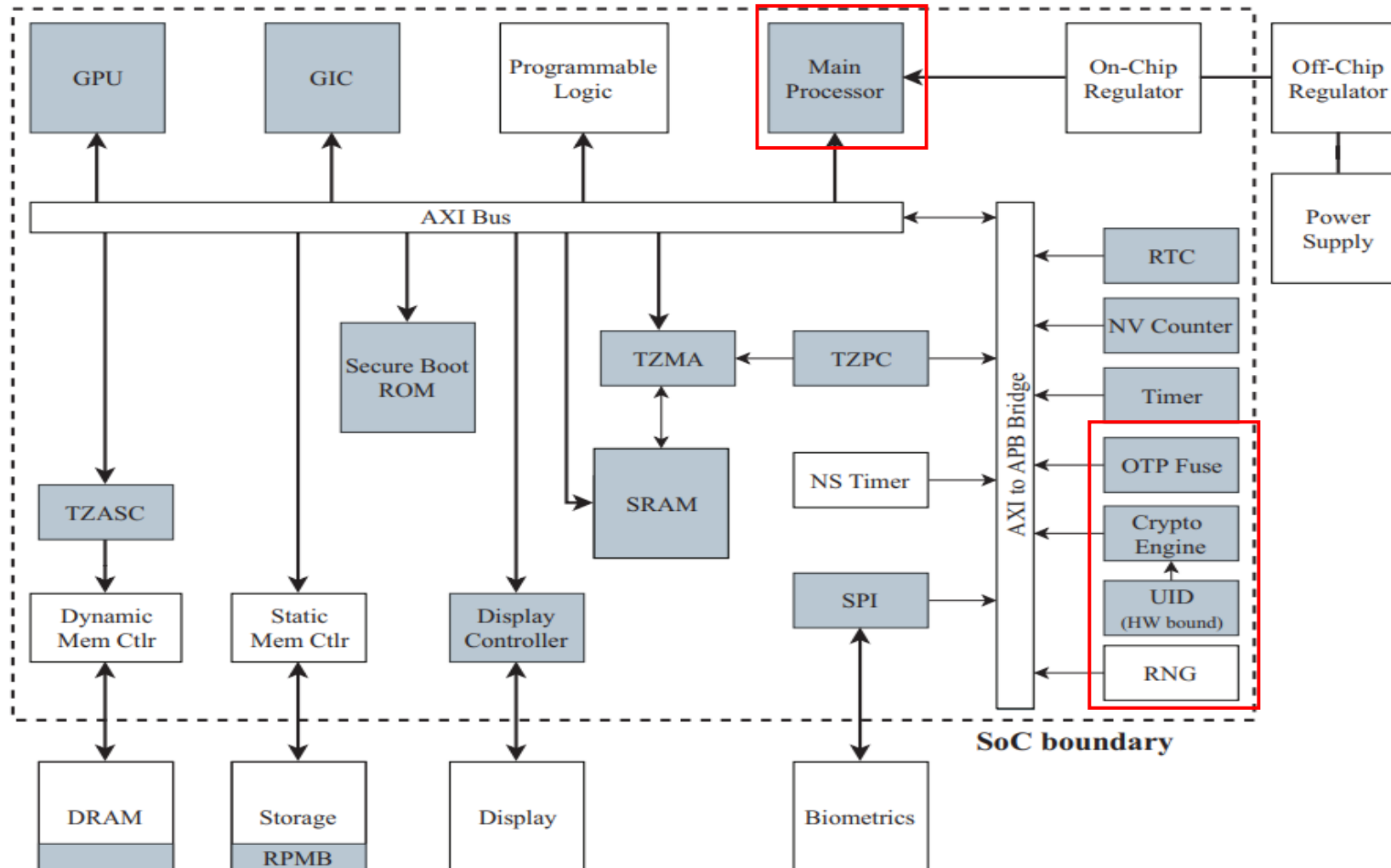
ARM TrustZone – Android

What is the Secure Element?

- Dedicated hardware module
 - Similar idea to TPM or HSM (sometimes is that)
 - Isolated component designed to handle cryptographic operations
- Very version-specific
 - Examples:
 - Separate PUF-based logic outside the CPU core but in the SoC ([ANI1271](#)):
 - [StrongBox](#)

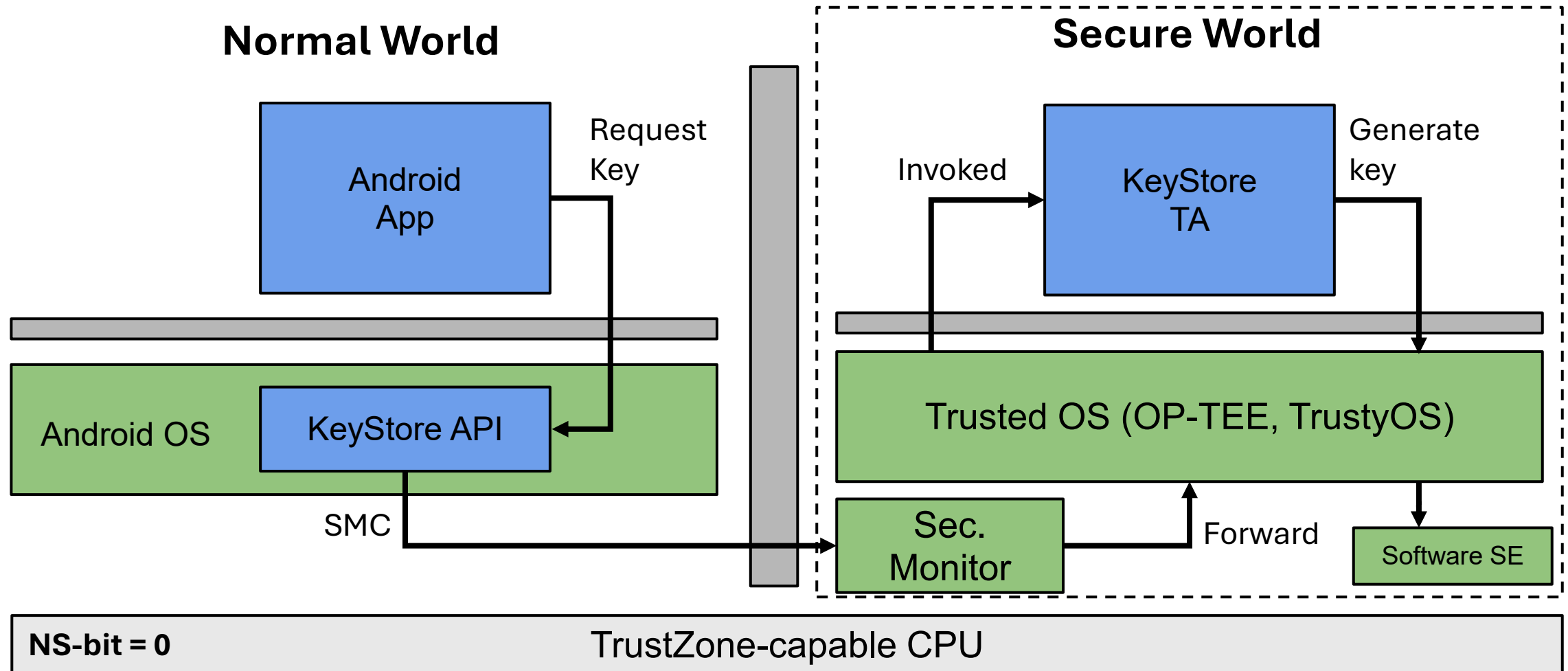
ARM TrustZone – Architecture

Recall:



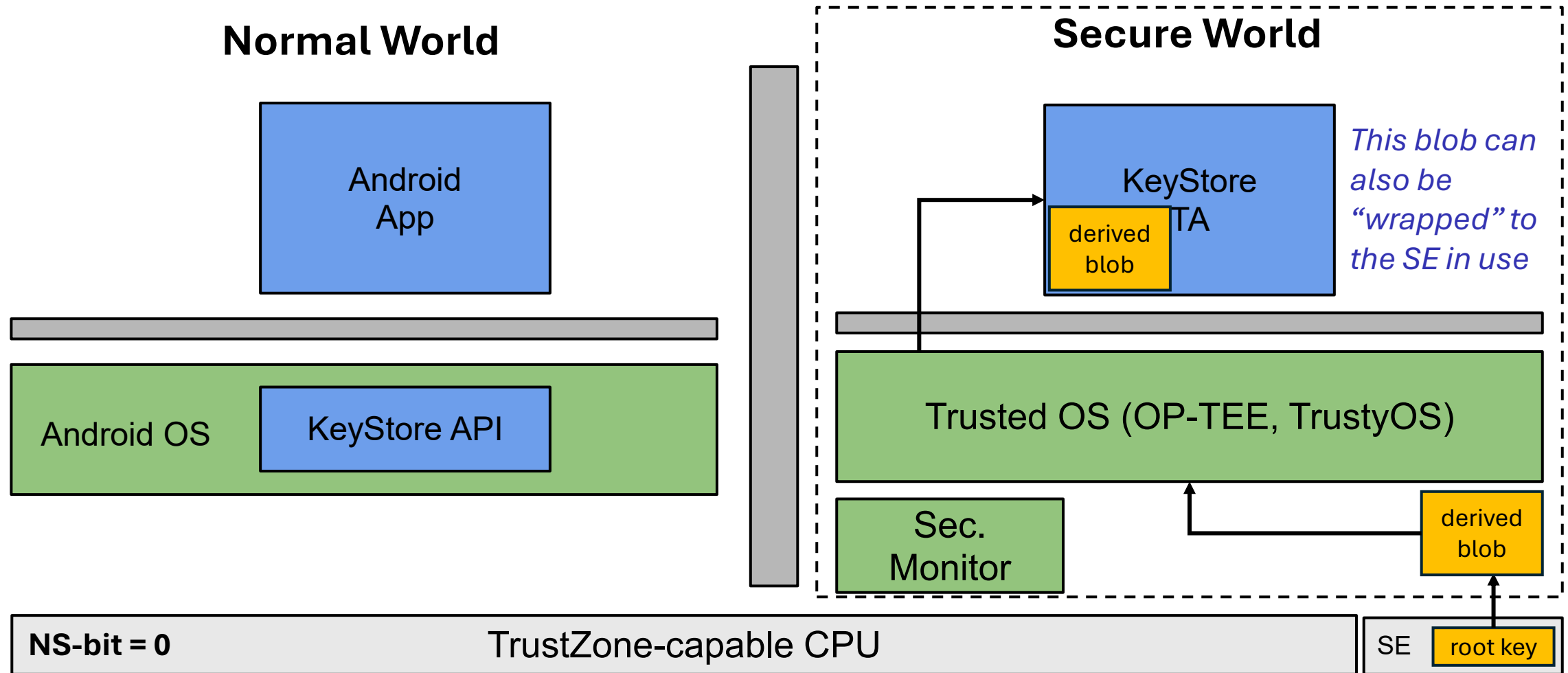
ARM TrustZone – Android

The TEE itself could be used for the SE, but less common



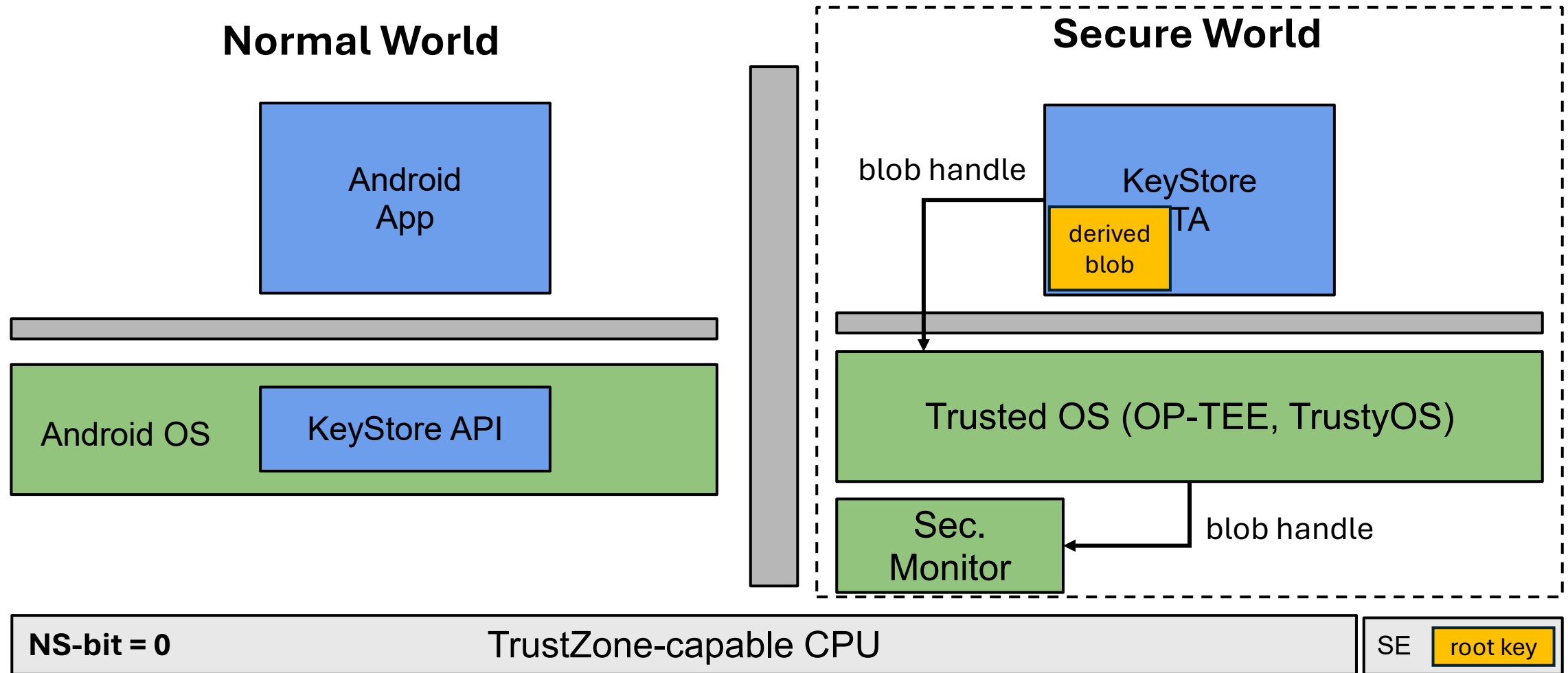
ARM TrustZone – Android

Generates key blob derived from a root key, stored in KS TA private memory



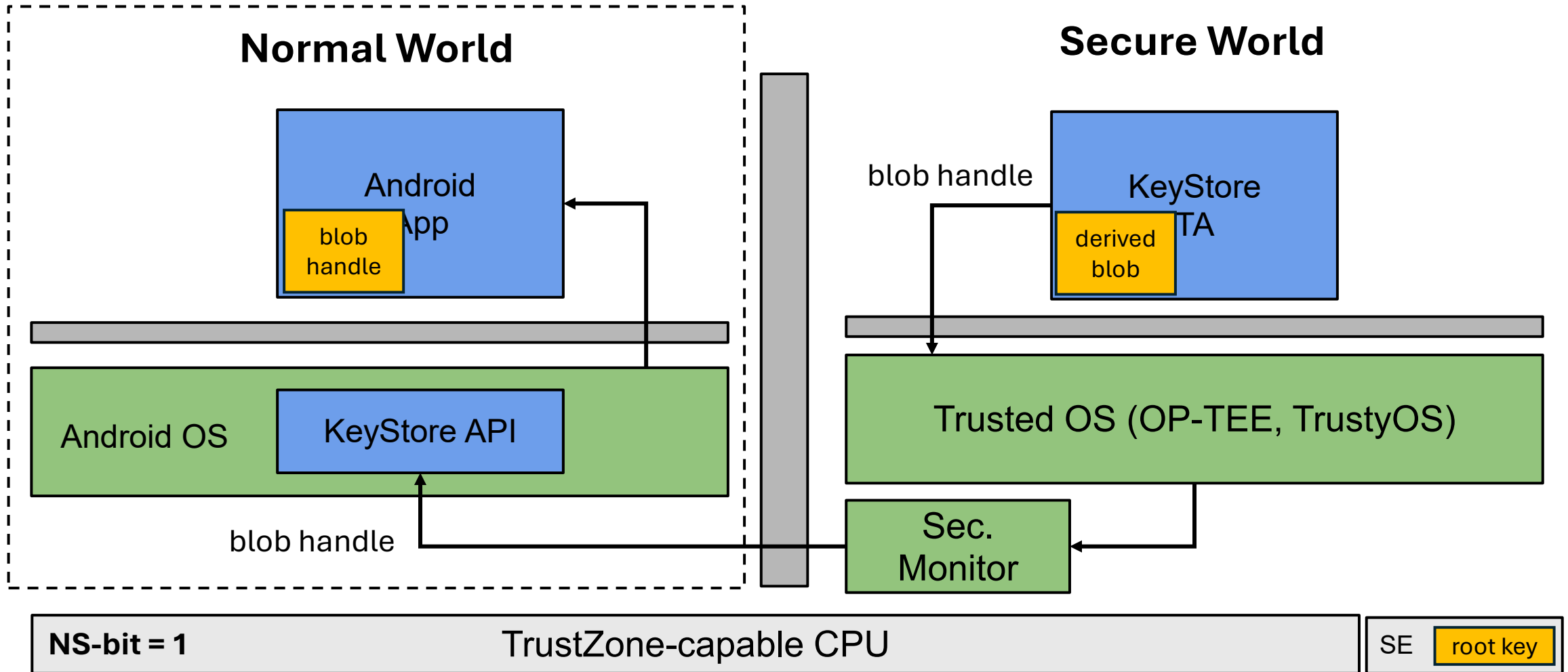
ARM TrustZone – Android

Returns a blob handle



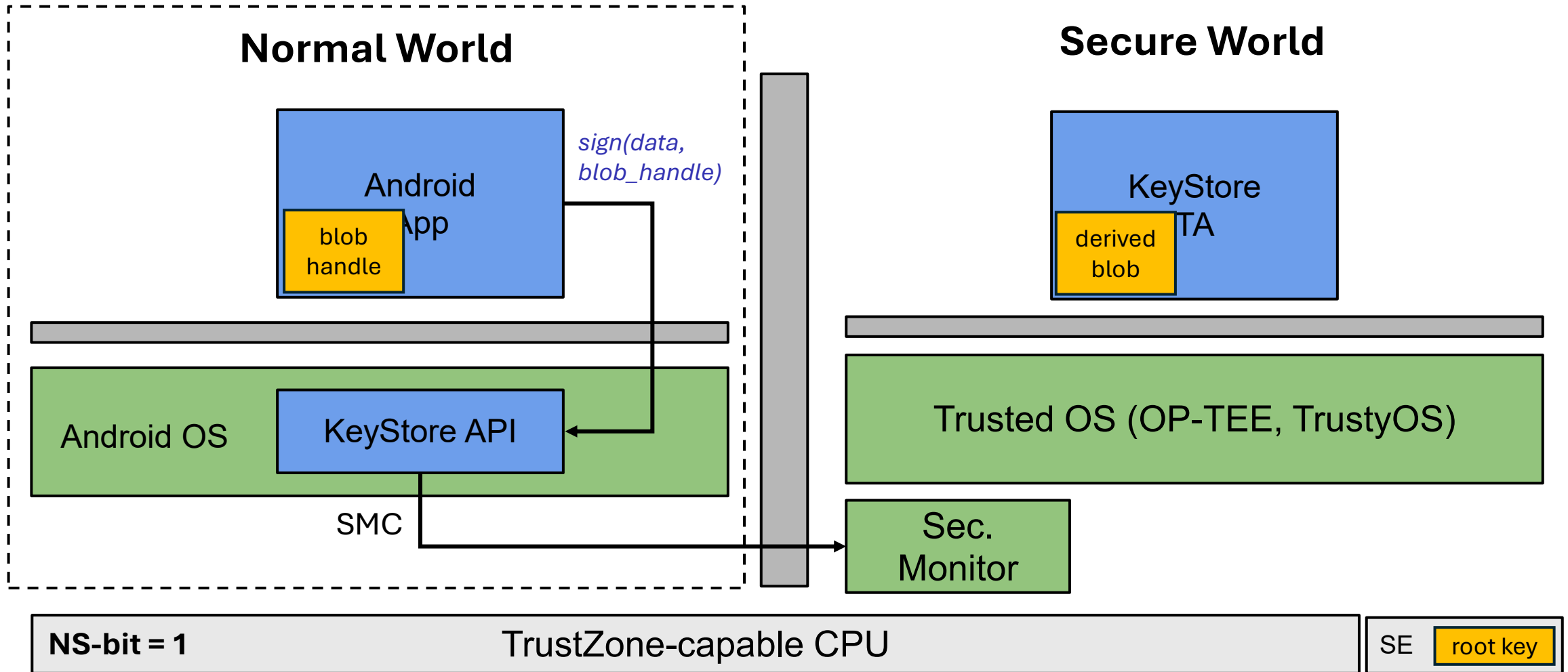
ARM TrustZone – Android

Atomic context switch, then store blob handle in Android App memory



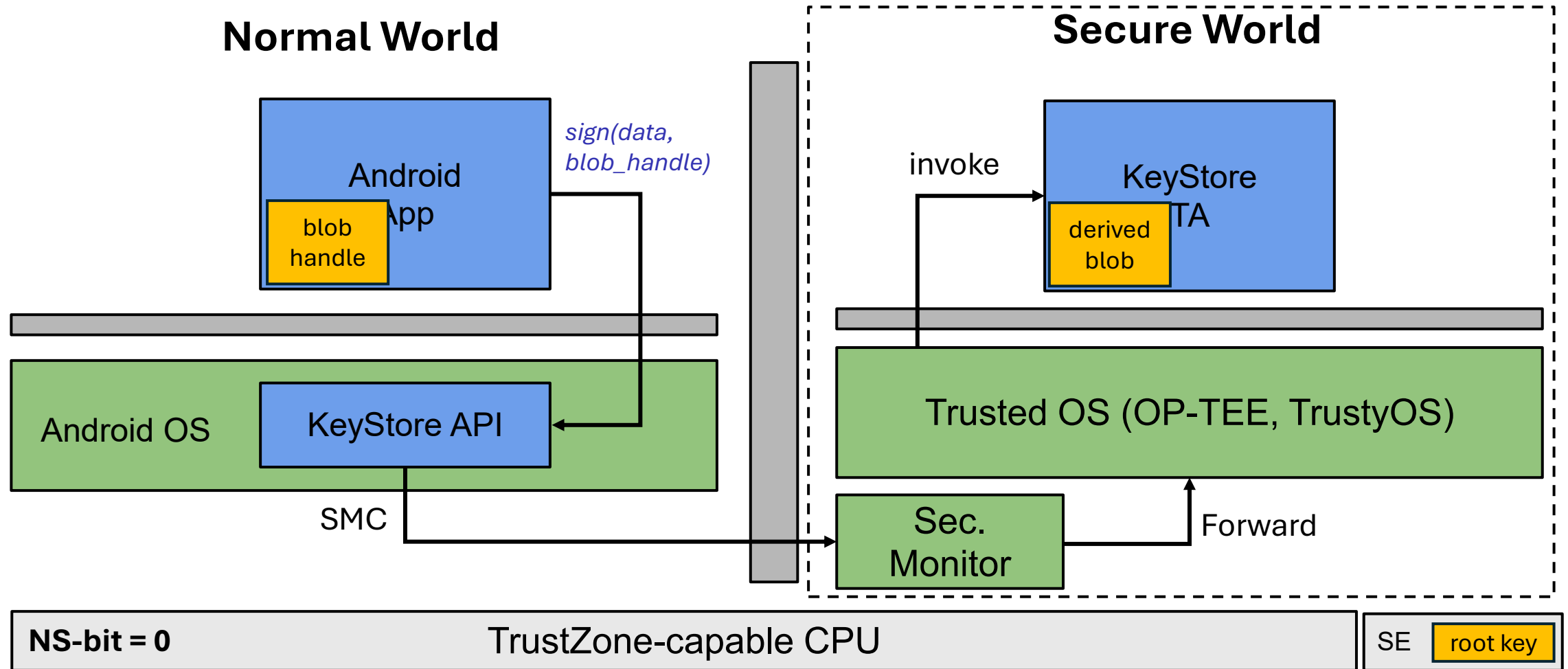
ARM TrustZone – Android

How about using the key? Invoke a KeyStore function, leading to SMC



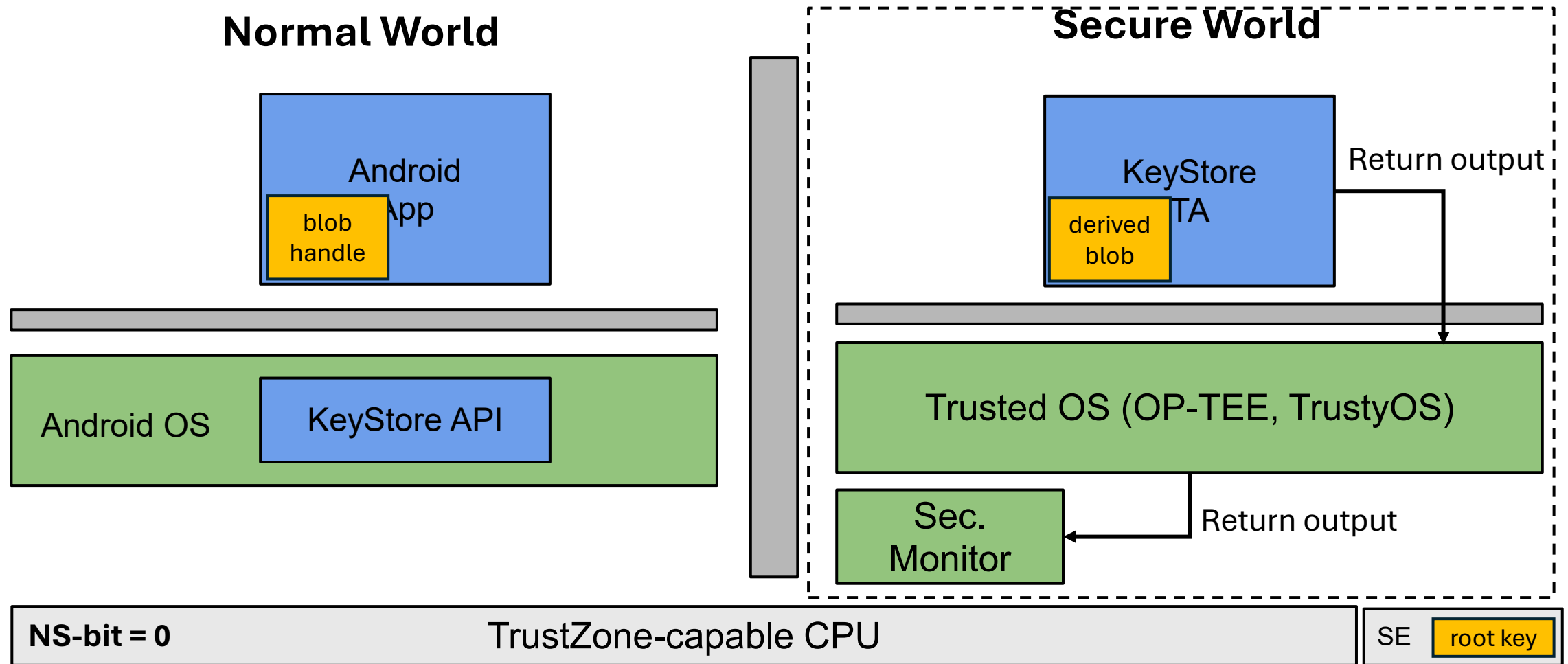
ARM TrustZone – Android

After being invoked, KeyStore TA retrieves key and performs operation



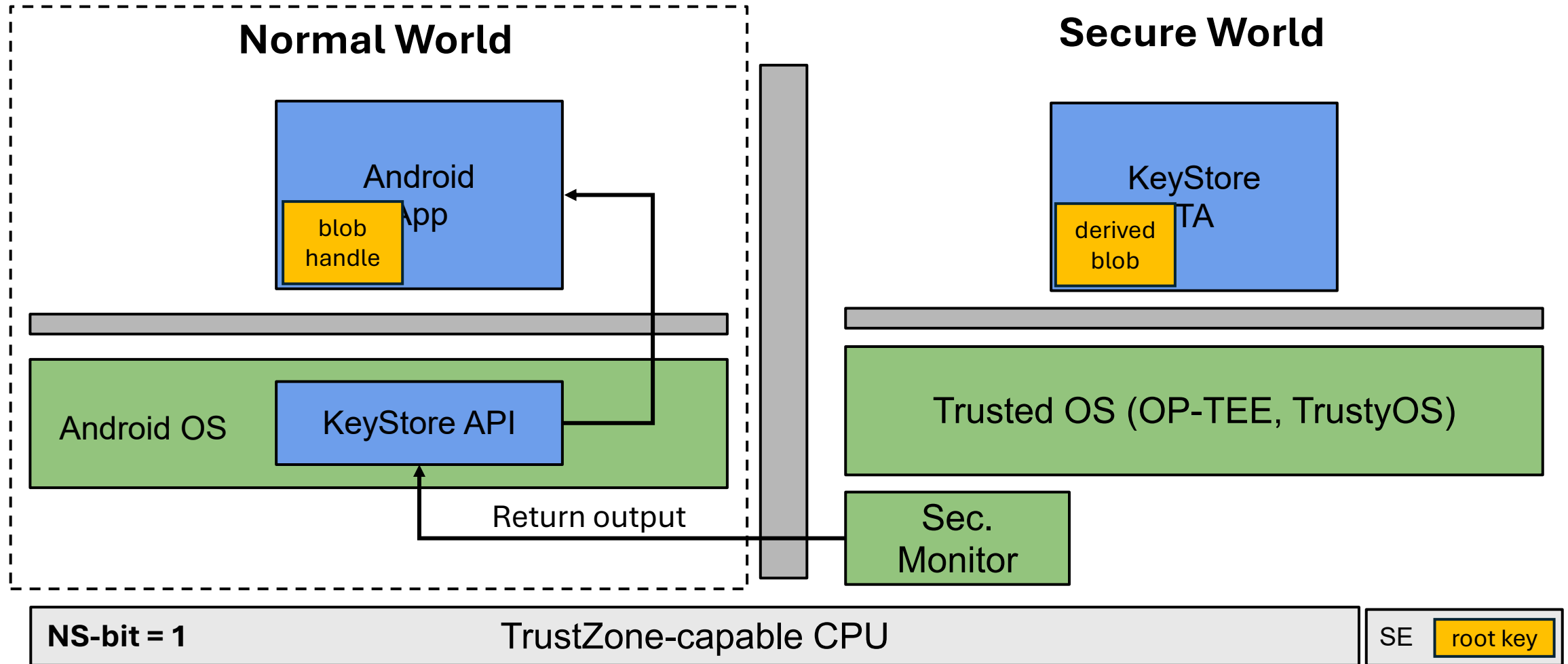
ARM TrustZone – Android

After being invoked, KeyStore TA retrieves key and performs operation



ARM TrustZone – Android

After being invoked, KeyStore TA retrieves key and performs operation



ARM TrustZone – Android

Other features of Android:

- **App signing**

- Every app must be signed by the developers
- Unsigned apps are rejected by Google play or the package installer

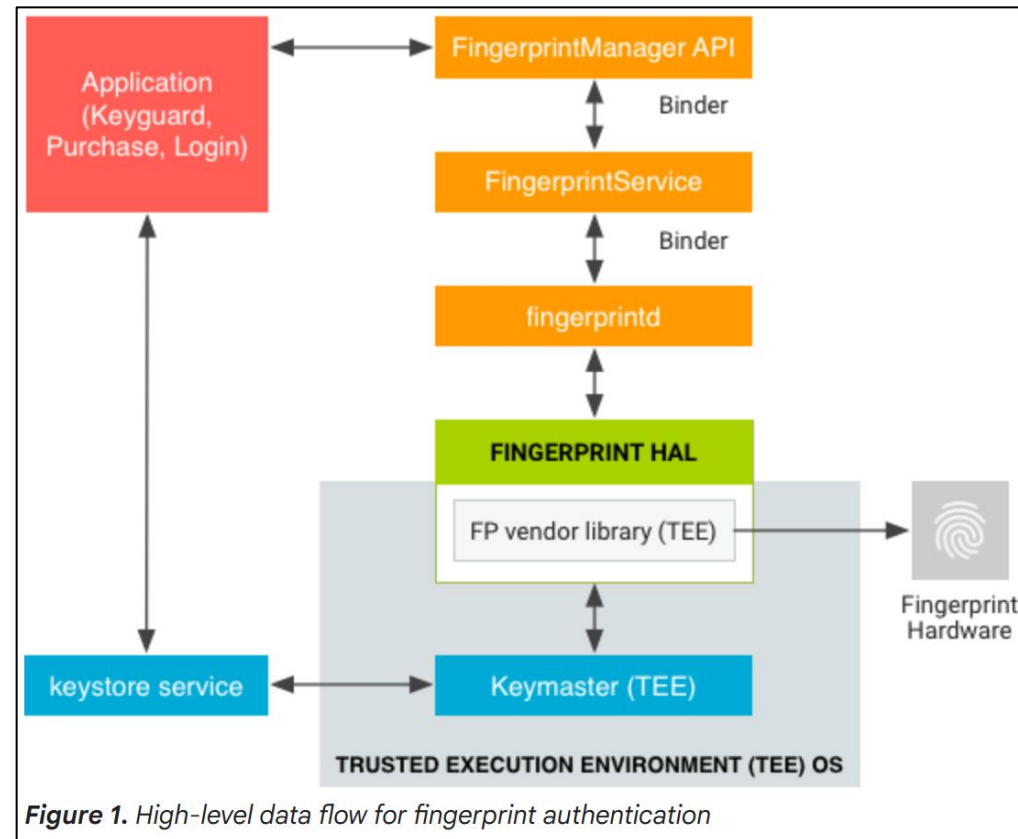
- **Biometrics**

- Part of tiered authentication model – fingerprint sensors
- Relies on the keystore for secure storage

ARM TrustZone – Android

Other features of Android:

- Biometrics (continued)



ARM TrustZone – Android

Other features of Android:

- **App signing**

- Every app must be signed by the developers
- Unsigned apps are rejected by Google play or the package installer

- **Biometrics**

- Part of tiered authentication model – fingerprint sensors
- Relies on the keystore for secure storage

- **Verified Boot**

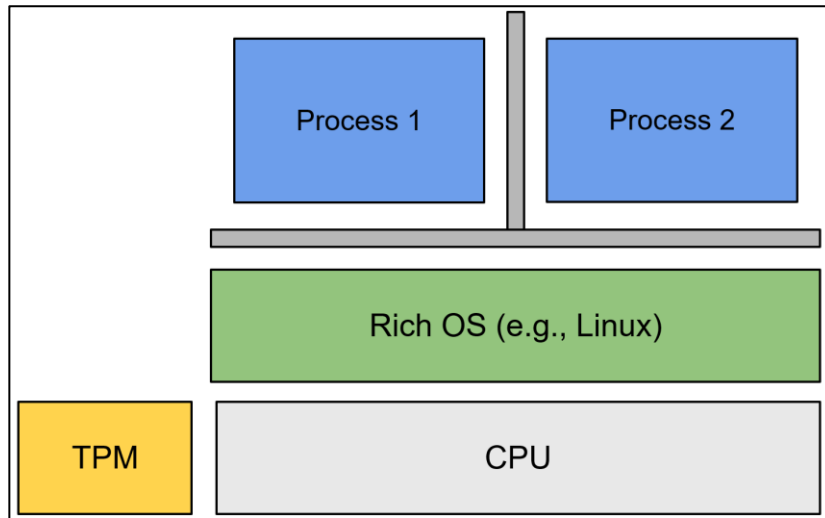
- **Rollback prevention**

- **Usable security – “Private Space”** – sandboxed space with separate install of app

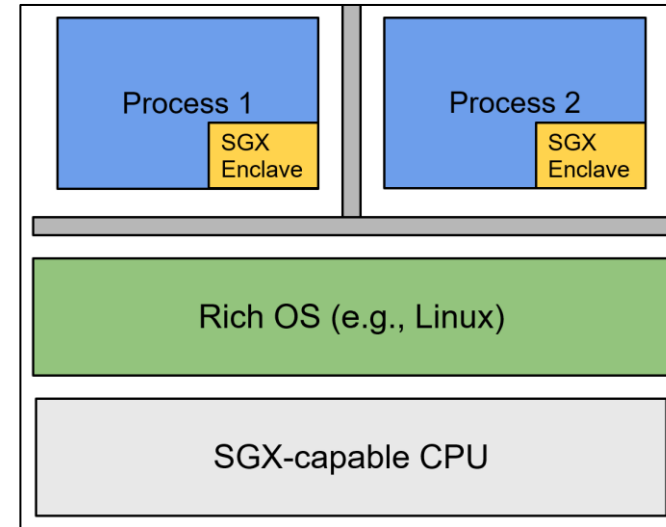
Closing thoughts

Various hardware security paradigms:

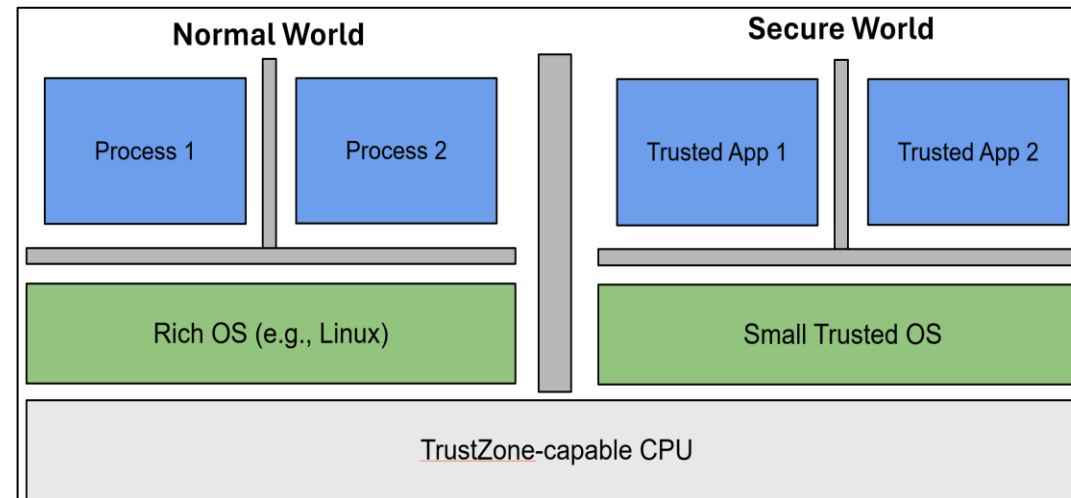
TPM



Intel SGX



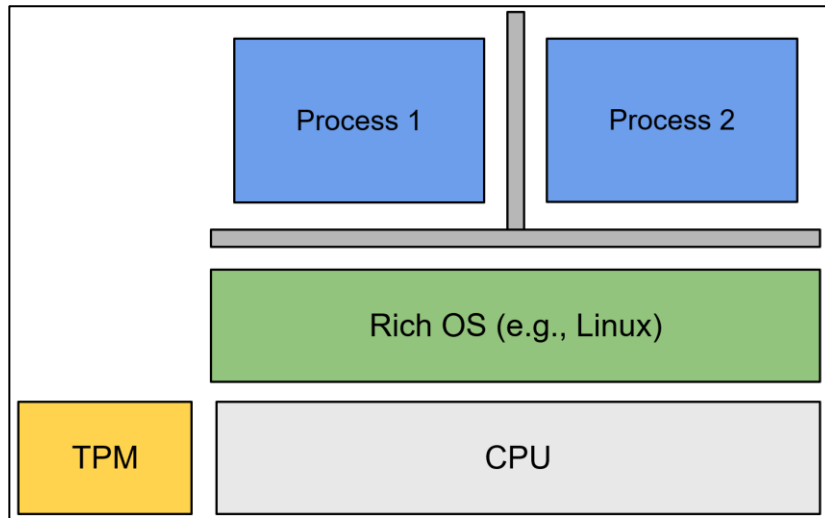
ARM TrustZone



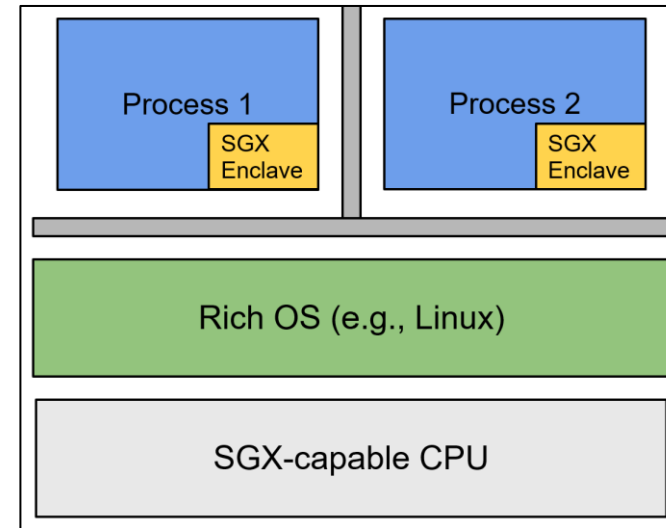
Closing thoughts

Various hardware security paradigms:

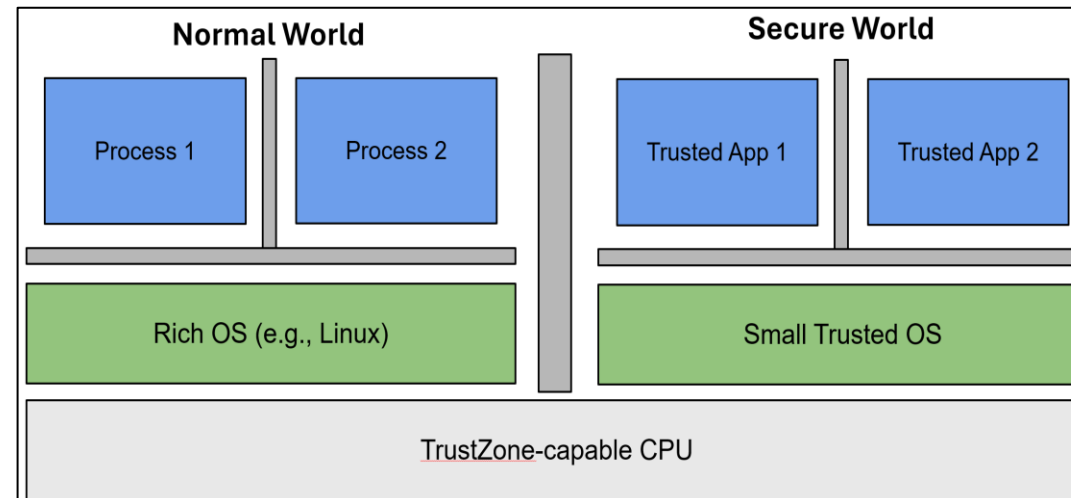
TPM



Intel SGX



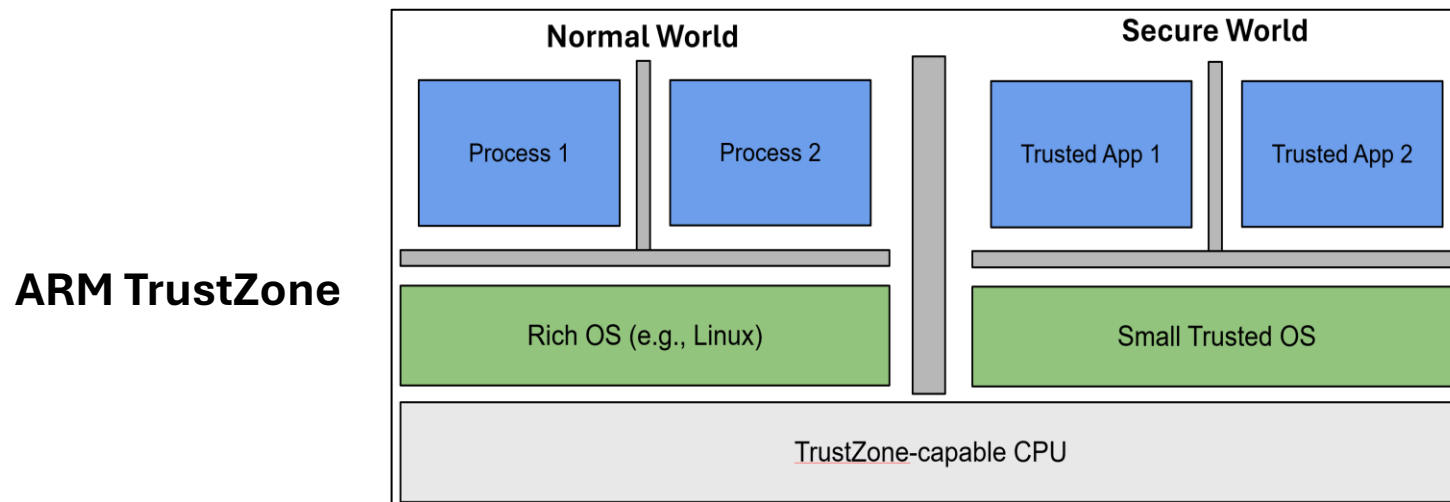
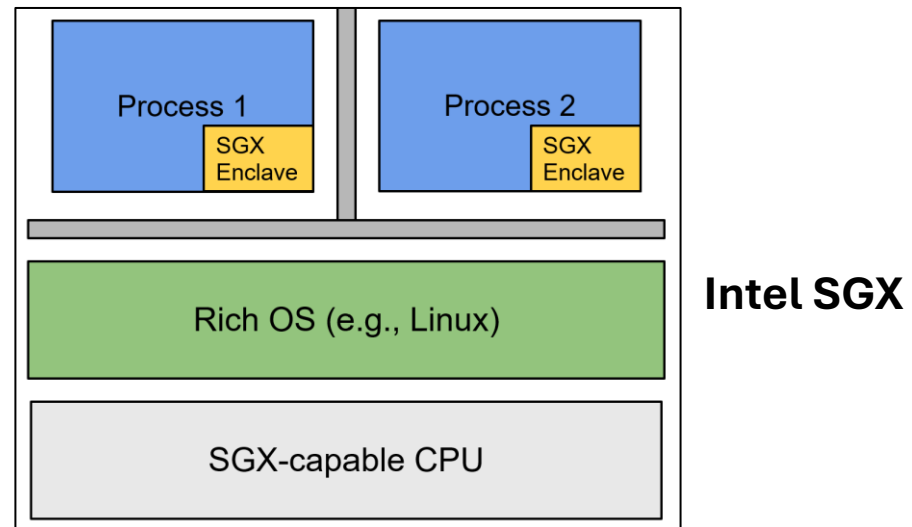
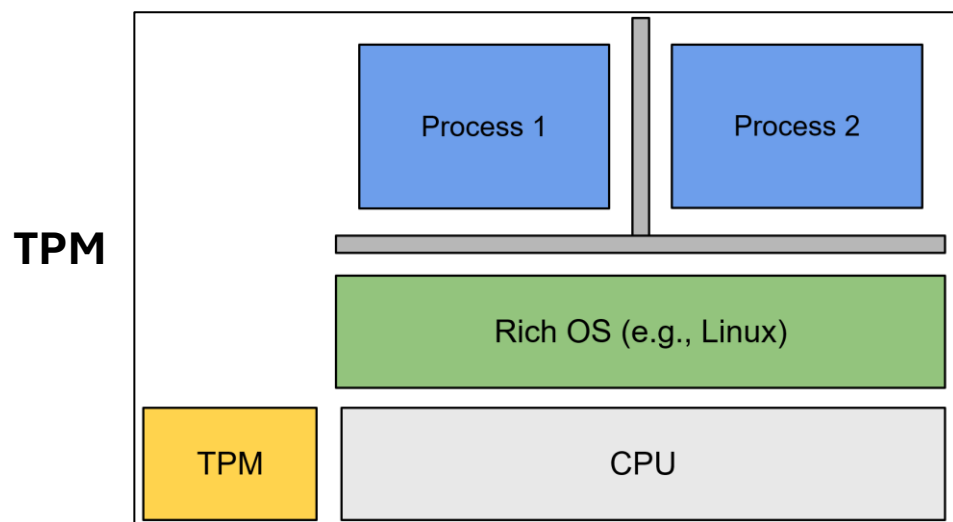
ARM TrustZone



We saved the World!

Closing thoughts

Various hardware security paradigms:



We saved the World!

Hopefully, there aren't any problems with these designs....

That's all for today!

Coming up....

- Attacks on TPMs and TEEs

Reminders:

- [A4 is due on July 25](#)
- Research project proposal

That's all for today!

Resources:

- [“Demystifying Arm TrustZone”](#) – great one!
- [“TrustZone Explained: Architectural Features and Use Cases”](#)
- [ARM Docs on TrustZone-A](#)
- [Android security resources](#)
- [Android KeyStore](#)
- [HSE & SoC as SE](#)
- [“Safeguarding Cryptographic Keys – TEE and Strongbox in Android”](#)
- [“Mobile Platform Security”](#)

