Trusted Execution Environments

Chester Rebeiro
IIT Madras

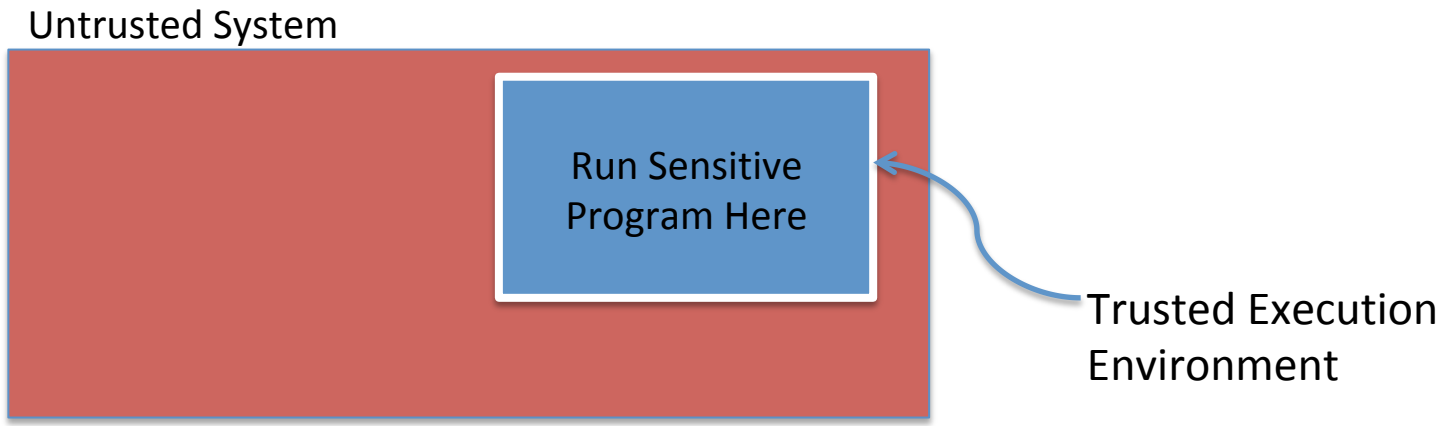
Some of the slides borrowed from Intel; CDACH; ARM

Previously in SSE…

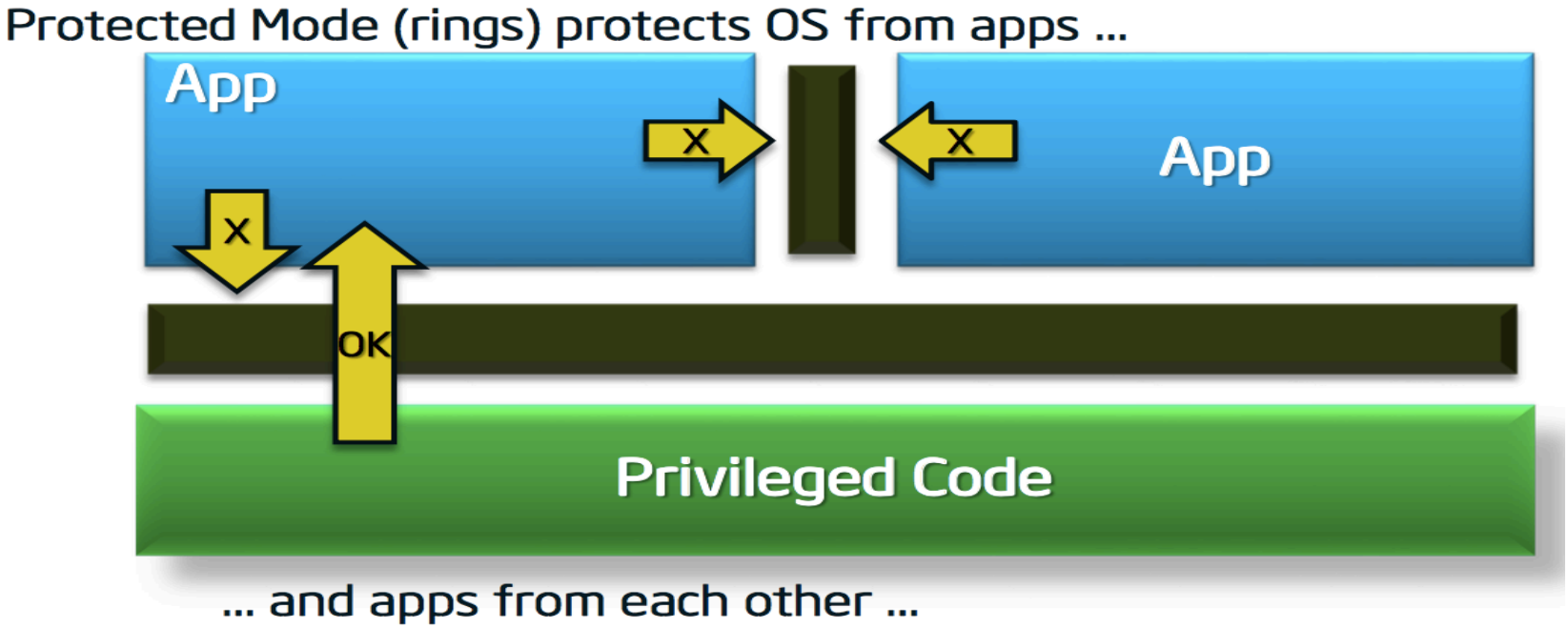
- We looked at techniques to run an untrusted code safely

Today in SSE…

- We now look at how to run sensitive code in an untrusted environment
  - Besides other applications, the OS can also be untrusted.
  - Attackers can probe hardware
- What to worry about:
  - Code / Data of the sensitive app gets read / modified by the system

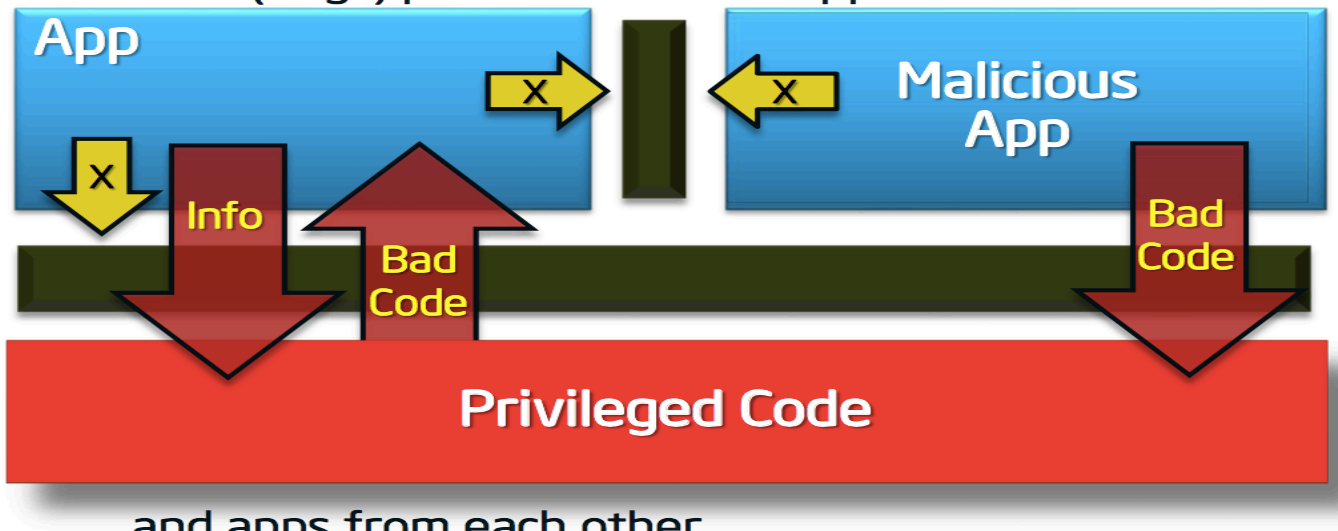


Basic Problem
(Ring Architecture)



Basic Problem (Ring Architecture)

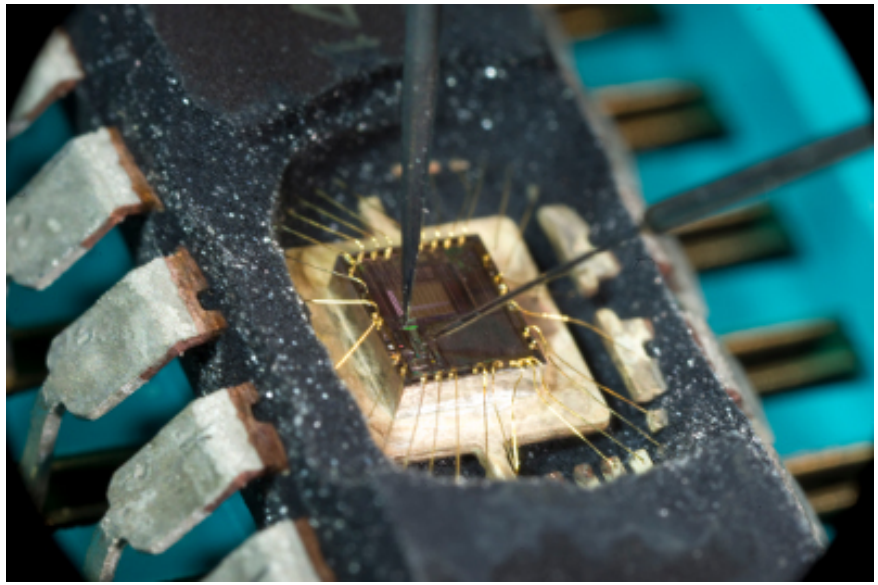
Protected Mode (rings) protects OS from apps …

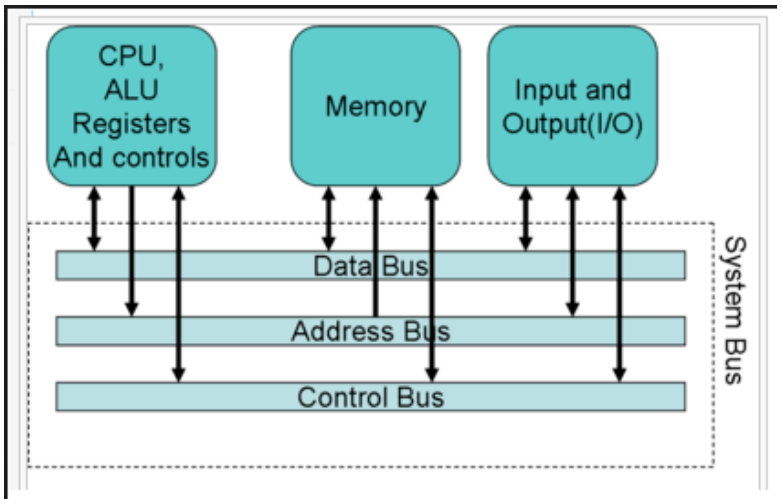


… and apps from each other …

… UNTIL a malicious app exploits a flaw to gain full privileges and then tampers with the OS or other apps

Invasive Attacks





Trusted Execution Environments

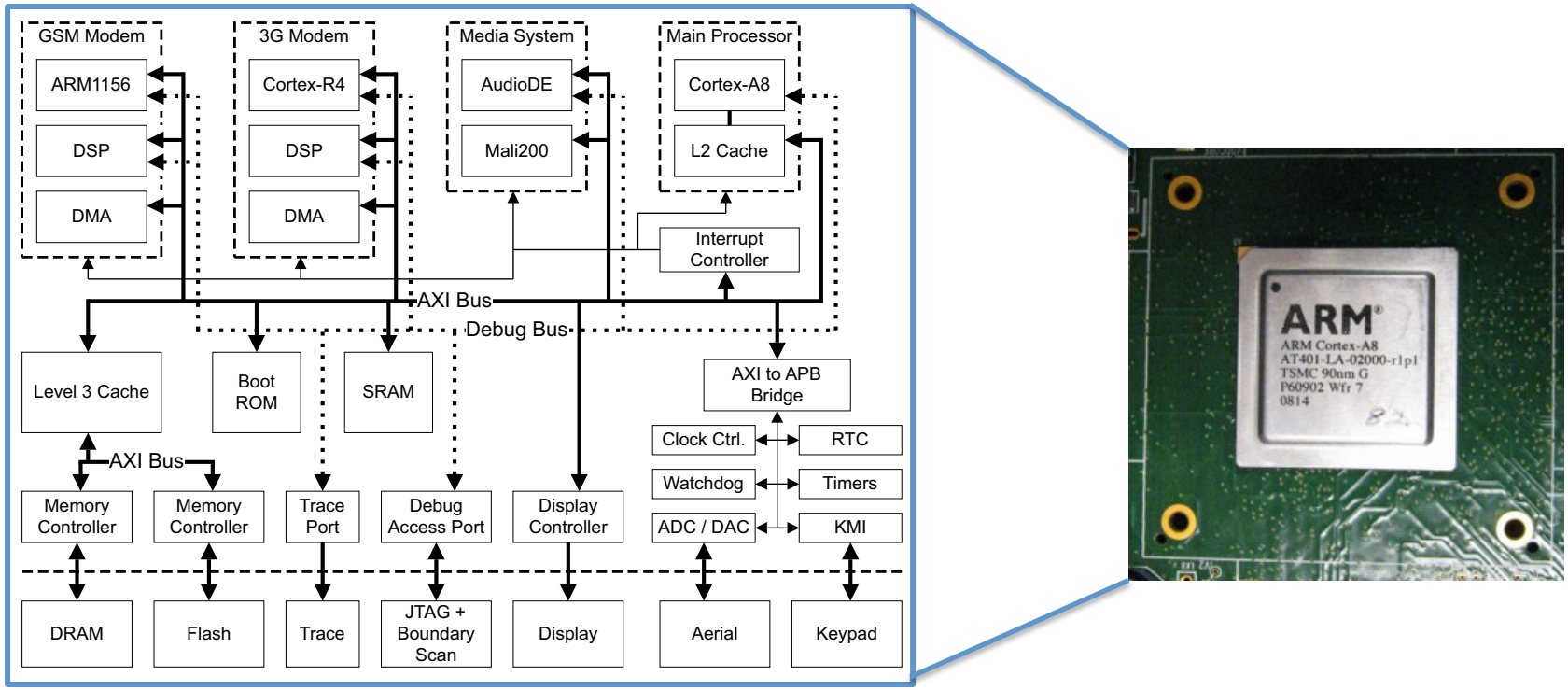
Achieve confidentiality and integrity even when the OS is compromised!

- ARM : Trustzone (trusted execution environments)
- Intel : SGX (enclaves)

ARM Trustzone

Trustzone Security Whitepaper, ARM
http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29GENC-009492C_trustzone_security_whitepaper.pdf

ARM System on Chips



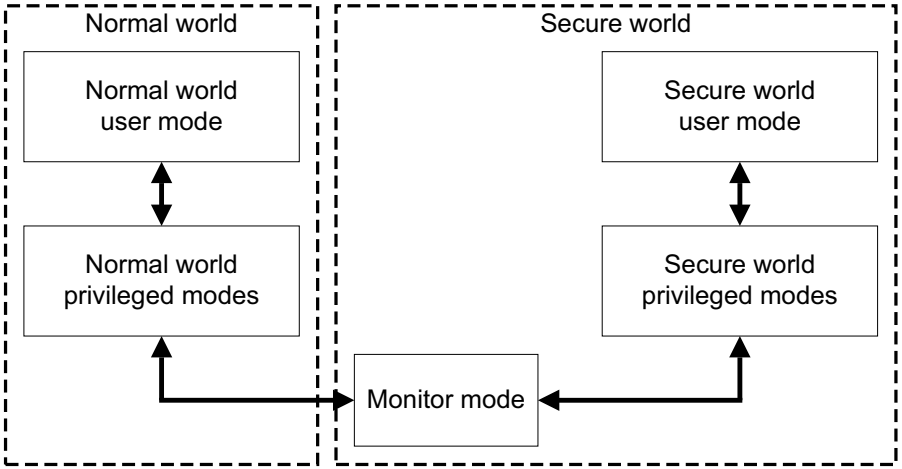
ARM Trustzone (Main Idea)

Hardware and Software partitioned into two: Normal and Secure worlds

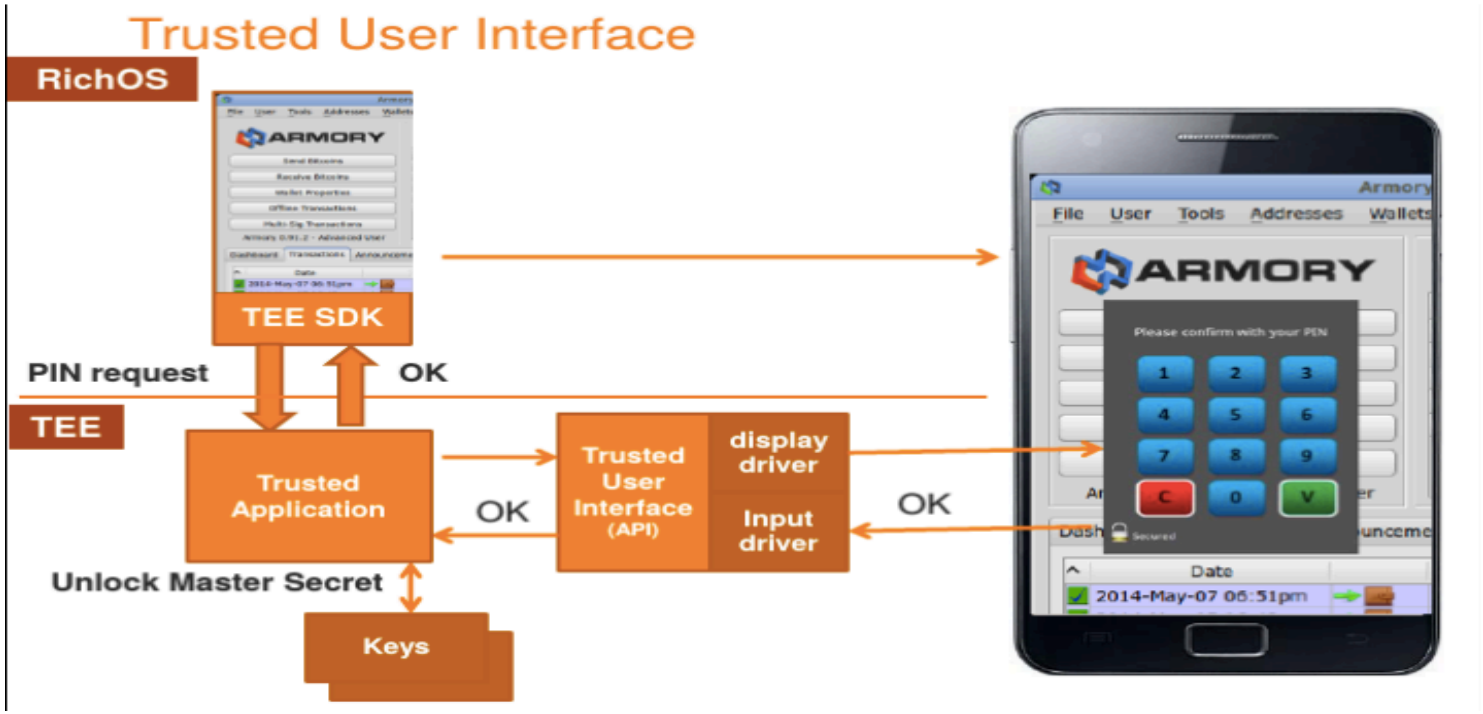
A single hardware processor timesliced between secure and normal worlds

Secure world provides an environment that supports confidentiality and integrity.

- Can prevent software attacks
- Cannot prevent invasive attacks



A Typical Trustzone Application



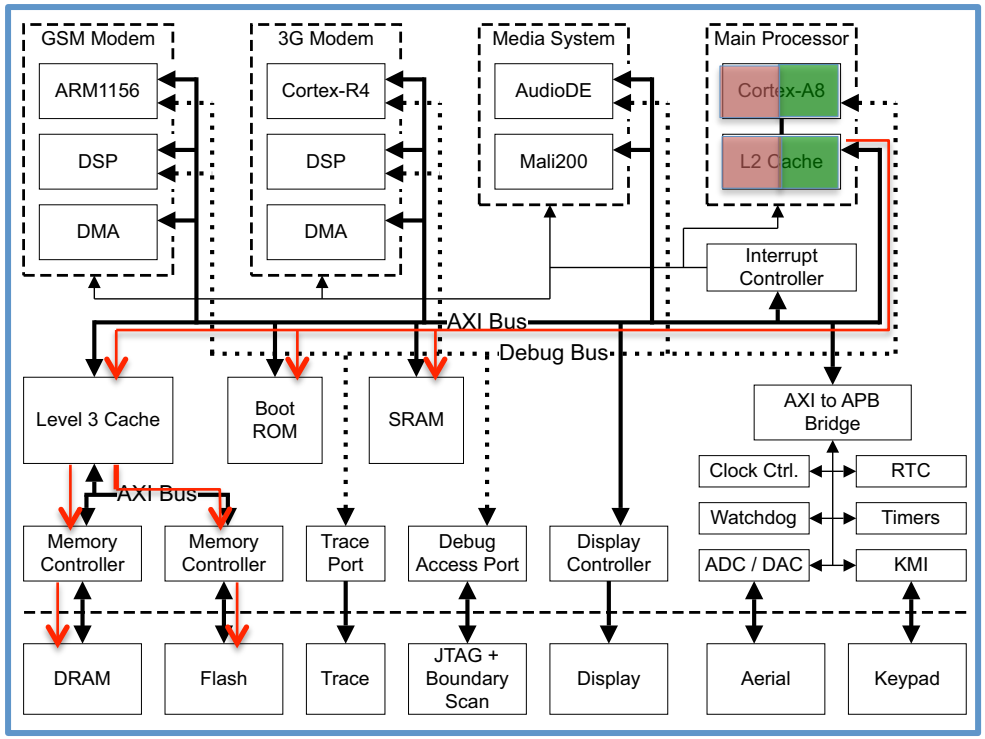
Switching Worlds

- Execution in time sliced manner (Secure <-> Normal)
- New mode (monitor mode) that is invoked during switching modes
- Mode switching
  - triggered by *secure monitoring call* (SMC) instruction
  - certain hardware exceptions (interrupts, aborts)

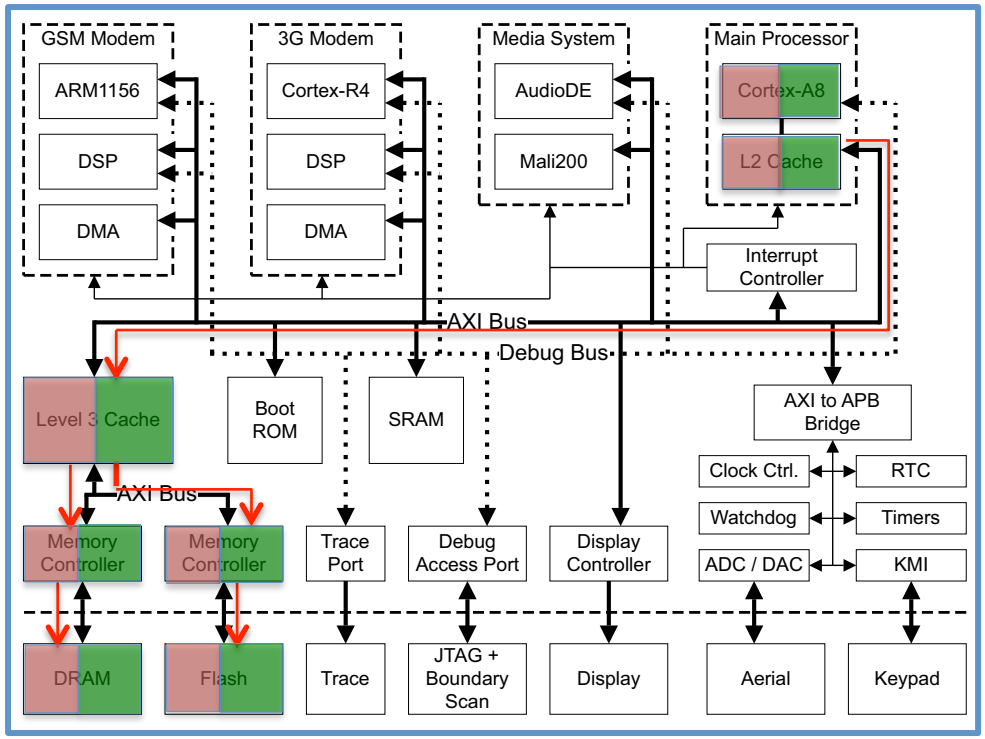
- Monitor Mode: saves state of the current world and restores the state of the world being switched to. Restoration by return-from-exception.

- NS Bit: in configuration register indicates secure / normal operating mode.
  NS = 1 -> indicates non-secure (normal) mode

NS Bit extends beyond the chip

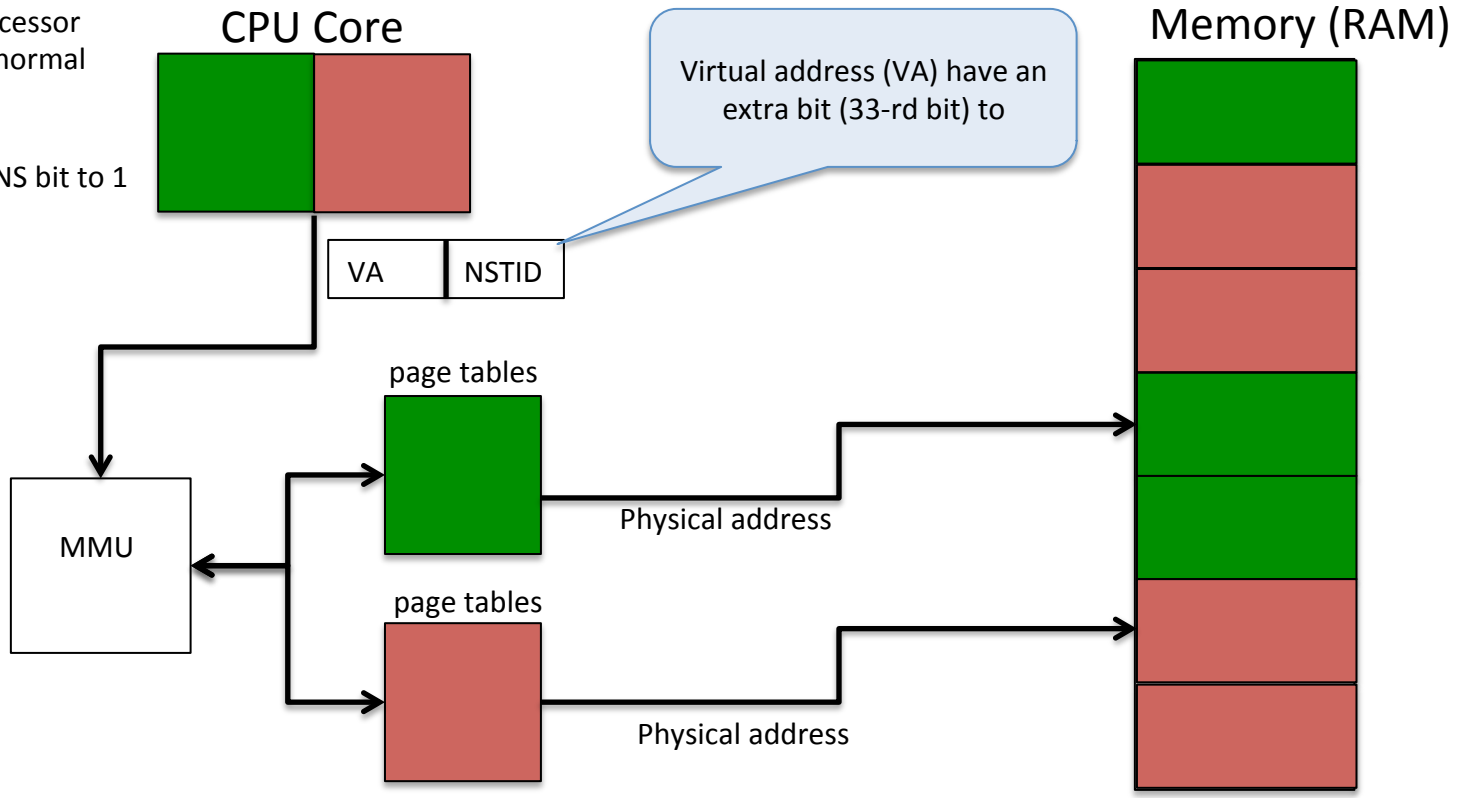


NS Bit extends beyond the chip

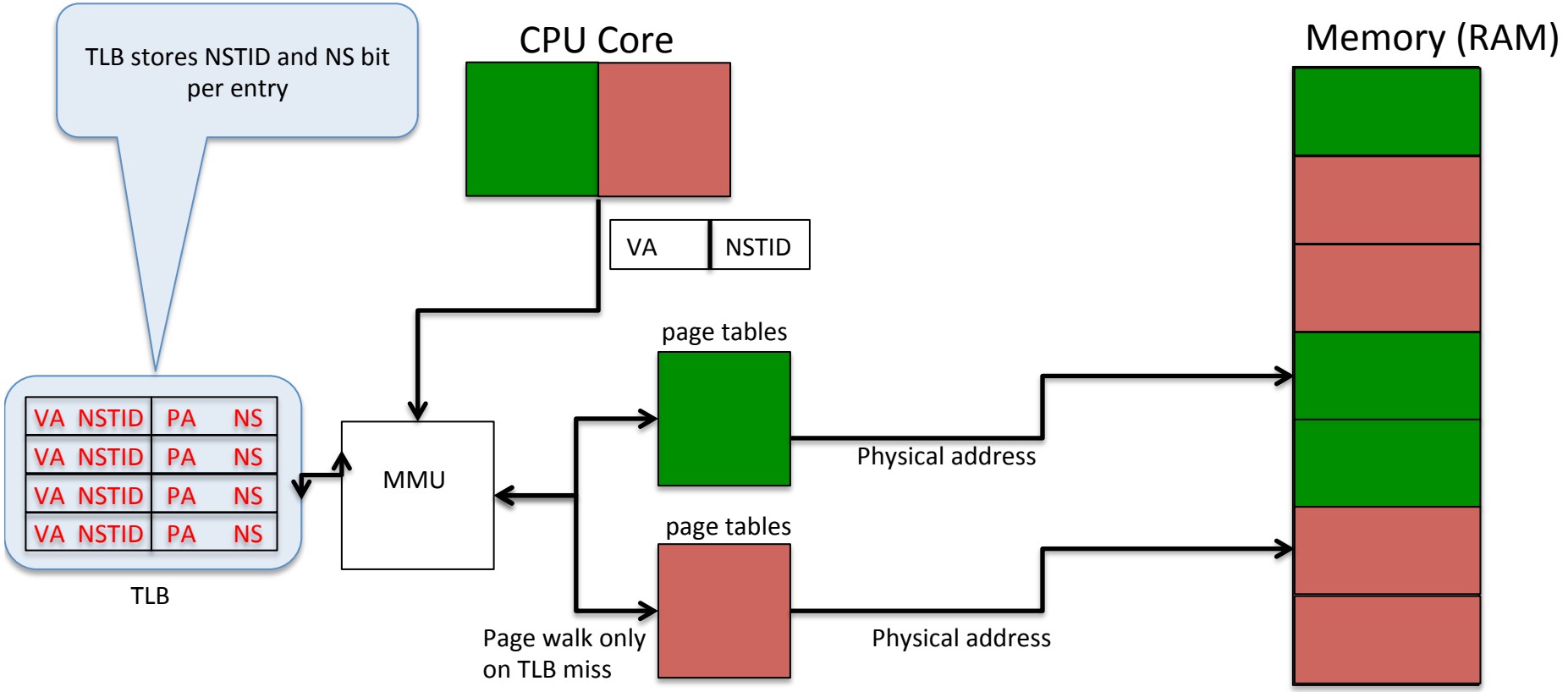


Memory Management

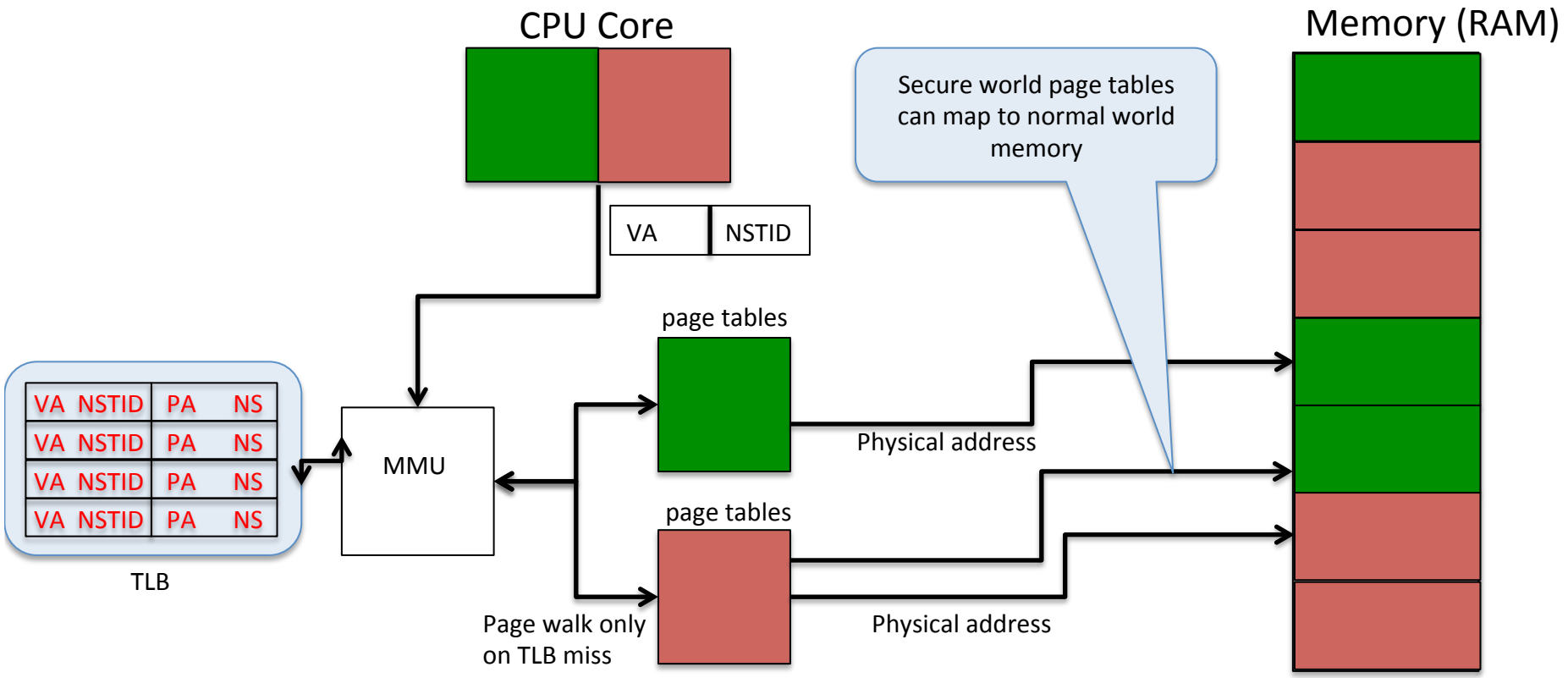
- Non Secure Table Identifier current state of the processor (0 if secure world / 1 if normal world
- If NSTID = 1 then force NS bit to 1



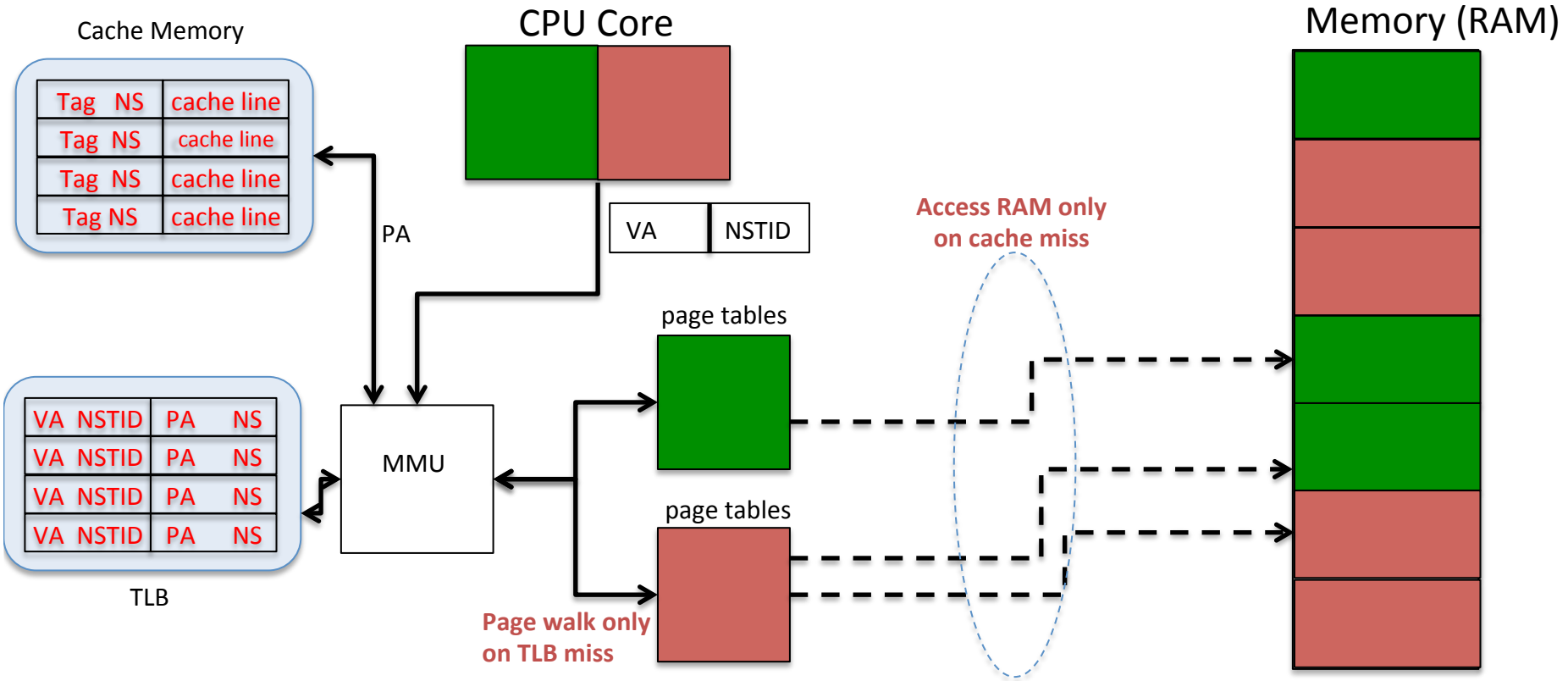
Memory Management



Memory Management



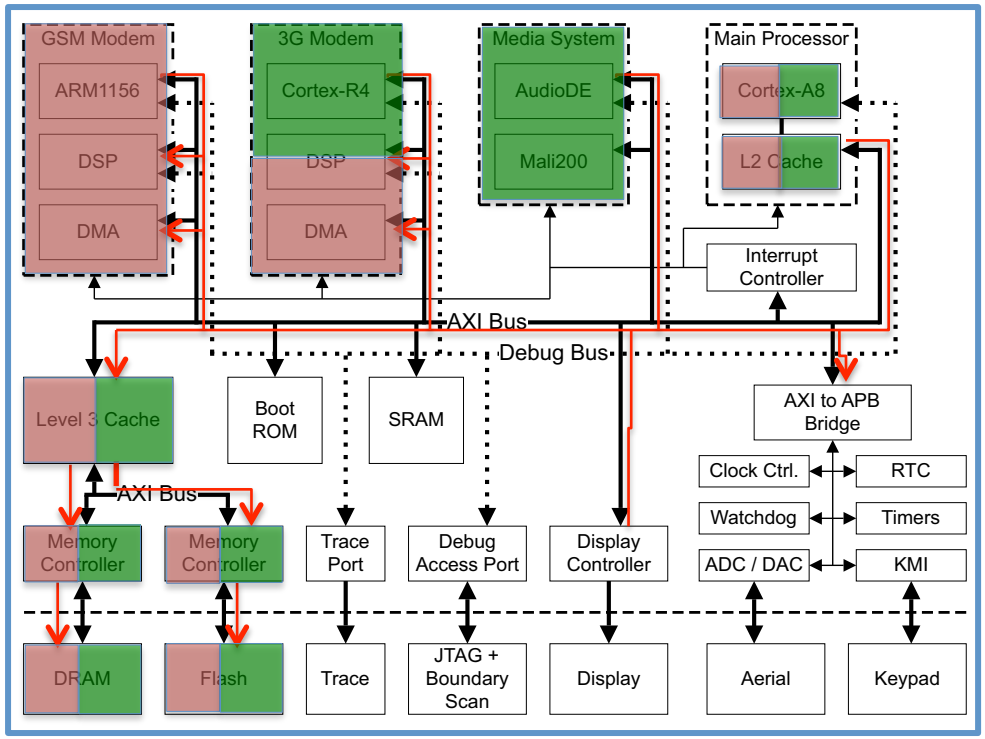
Memory Management



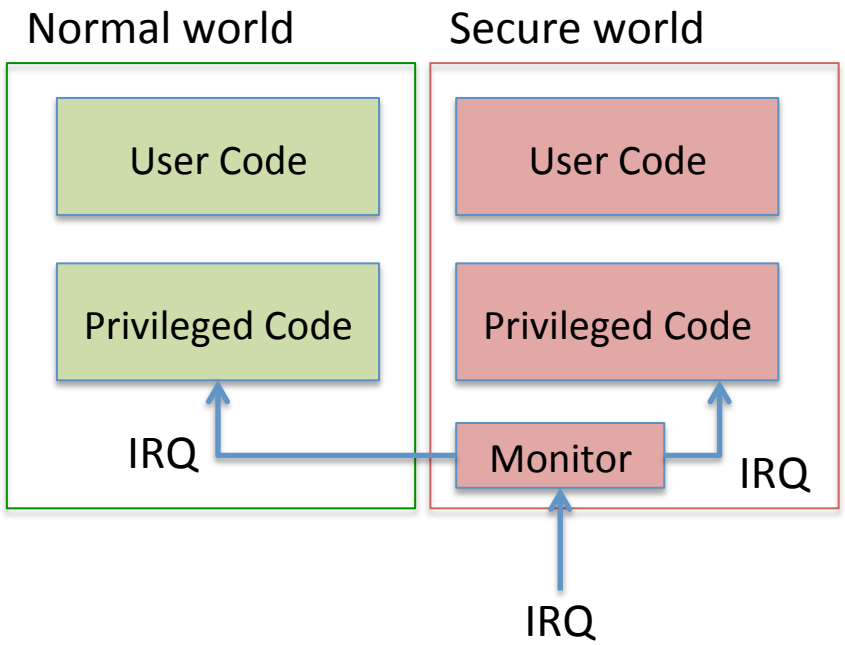
Memory Management Units

- Two virtual MMUs (one for each mode)
  - Two page-tables active simultaneously
- A single TLB present
  - A tag in each TLB entry determines the mode
    (Normal and Secure TLB entries may co-exist; this allows for quicker switching of modes)
  - alternatively the monitor may flush the TLB whenever switching mode
- A single cache is present
  - Tags (again) in each line used to store state
  - Any non-locked down cache line can be evicted to make space for new data
  - A secure line load can evict a non-secure line load (and vice-versa)

Secure and Normal Devices

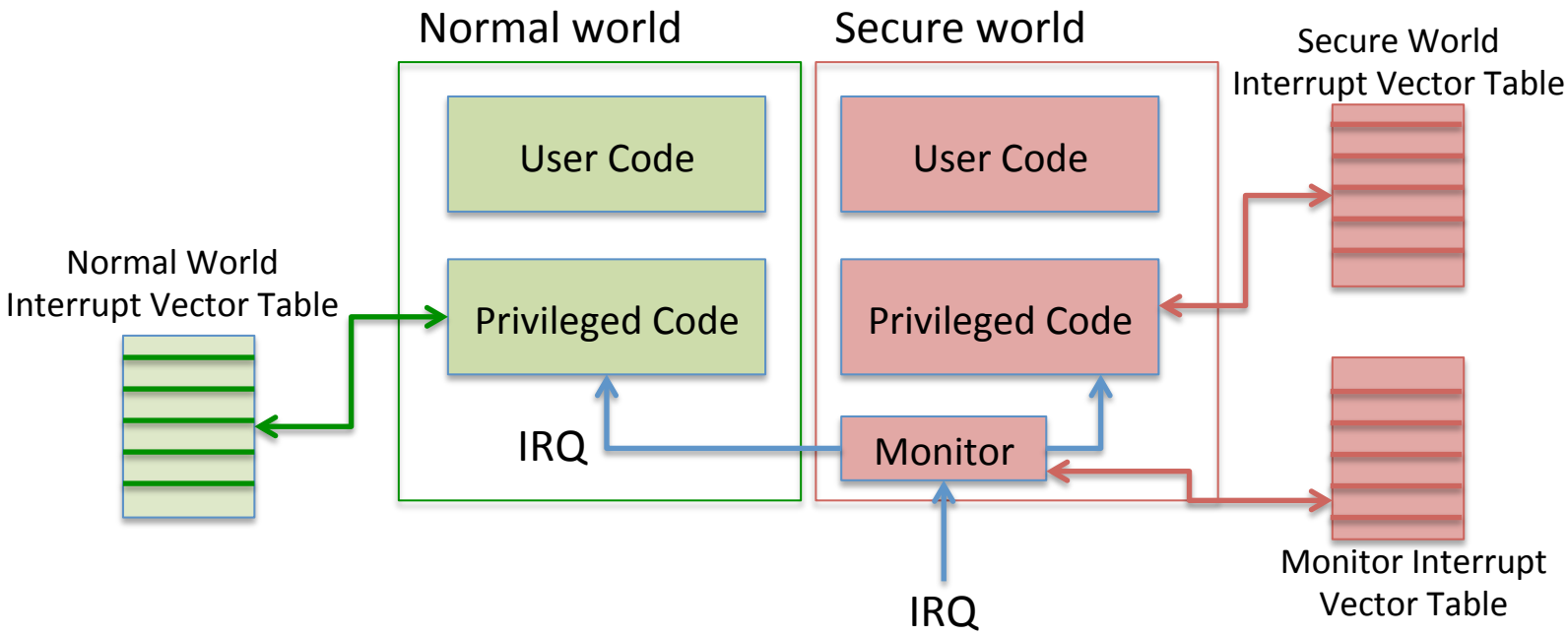


Interrupts



All interrupts routed to monitor first.
Interrupts can be configured to go either to the normal world or secure world.

Interrupts



All interrupts routed to monitor first.
Interrupts can be configured to go either to the normal world or secure world.

Software Architecture

- The minimal secure world can just have implementations of synchronous code libraries
- Typically has an entire operating system
  - Qualcomm's QSEE; Trustonics Kinibi; Samsung Knox; Genode
  - The secure OS could be tightly couples to the rich OS so that a priority of a task in the rich OS gets mapped accordingly in the secure OS
  - Advantage of having a full OS is that we will have complete MMU support
- Intermediate Options

Secure Boot

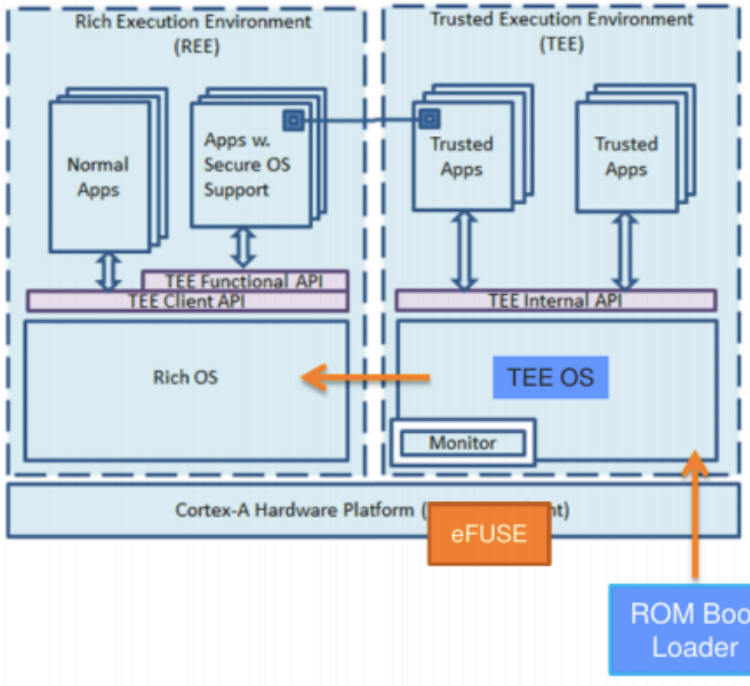
Why?

Attackers may replace the flash software with a malicious version, compromising the entire system.

How?

Secure chain of trust.
Starting from a root device (root of trust) that cannot be easily tampered

Secure Boot Sequence



On chip ROM based Bootloader

May be internal to the CPU;
Initializes critical peripherals, memory controllers

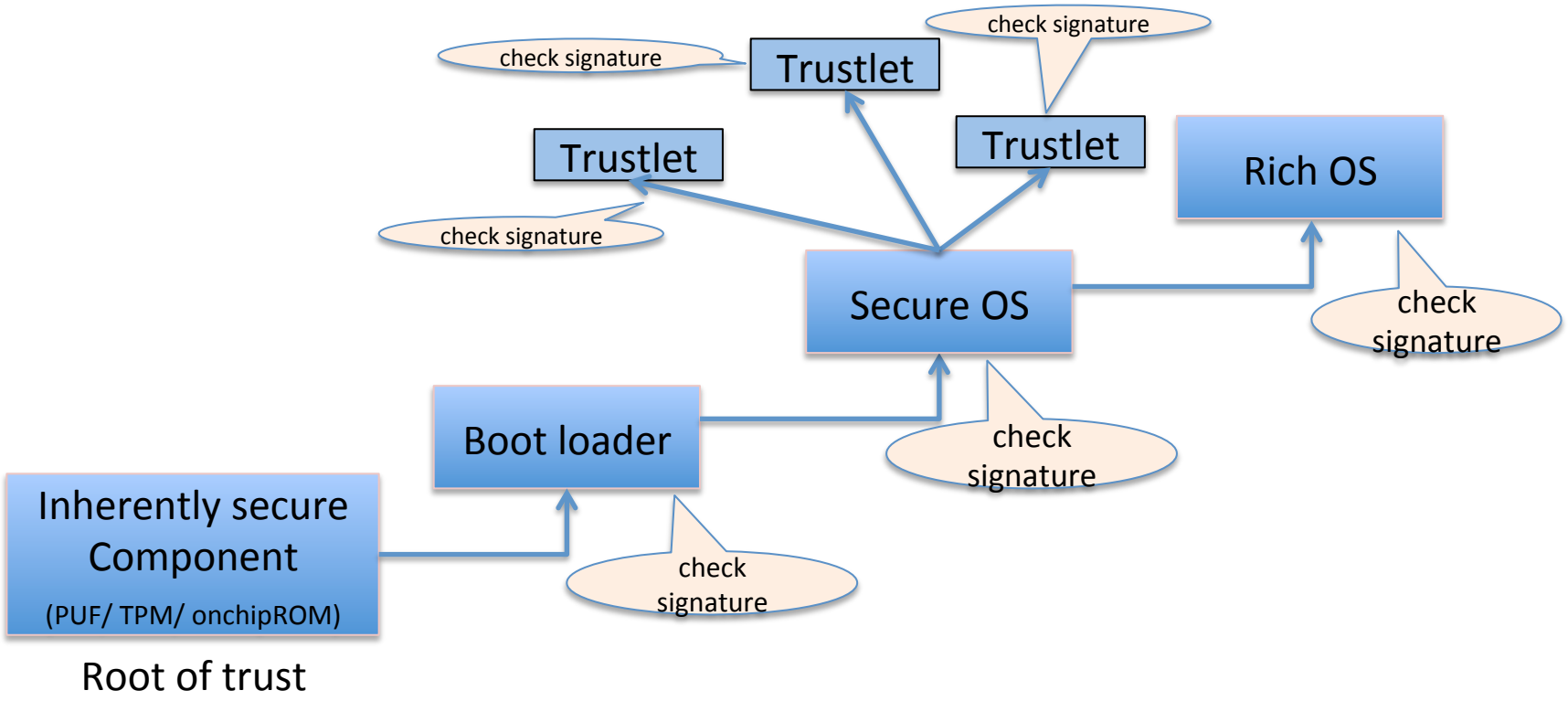
Device bootloader

Stored on Flash device (typically)
Initialize critical peripherals

Secure operating system

Rich operating system

Chain of Trust



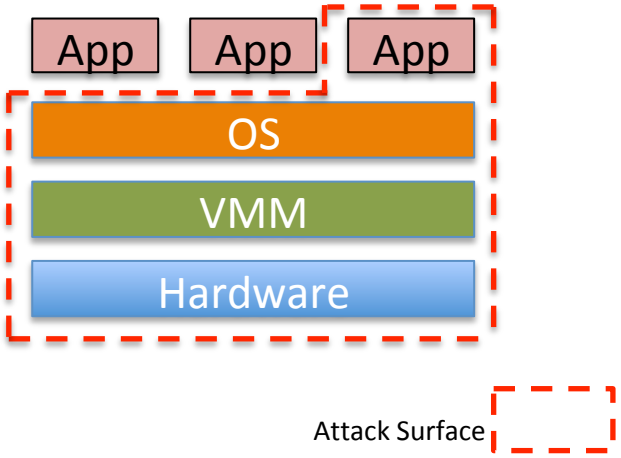
Root of trust

Intel's SGX

Innovative Instructions and Software Model for Isolated Execution, HASP 2013 (F. McKeen et. al.)

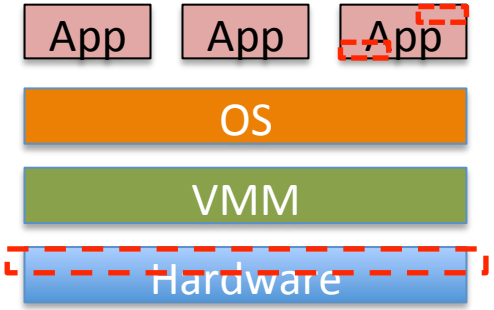
Reduced Attack Surface with SGX

Normally



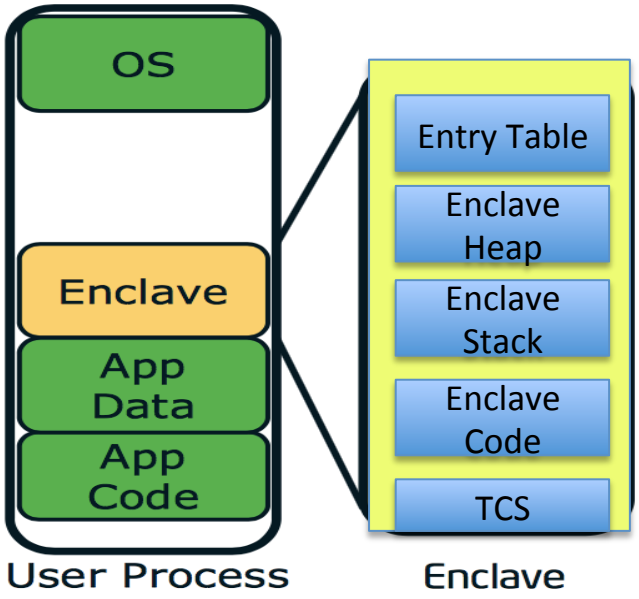
Malware that can subvert any one of app, OS, VMM, or hardware can steal secrets

With SGX enabled



Small attack surface (App + Hardware)
Malware cannot steel secrets inspite of subverting OS, BIOS, VMM, most parts of the App, etc.

Enclaves
(reverse sandbox)

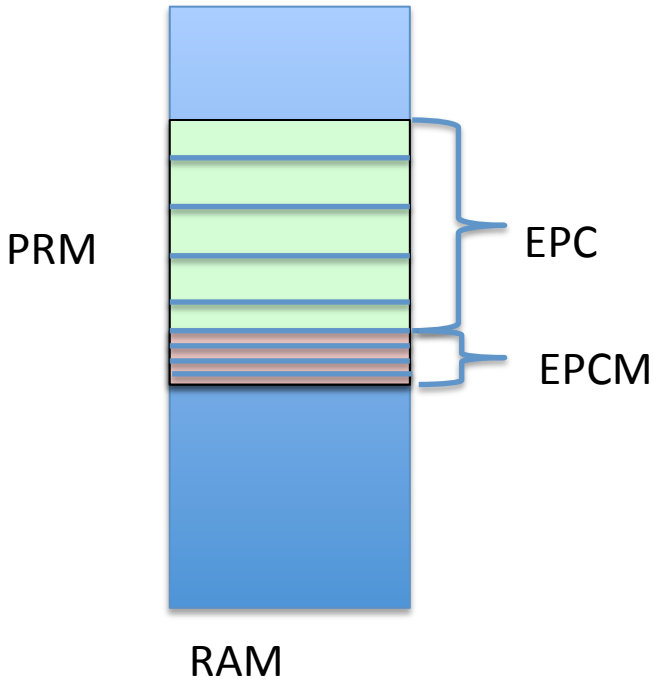


- Enclave has its own code and data areas
  Provides confidentiality and integrity
  With controlled entry points
- However, enclave code and data cannot be accessed from outside the enclave not even by the operating system.
- TCS: Thread control Structure
  (SGX supports multi-threading;
  one TCS for each thread supported)

Enclave Properties

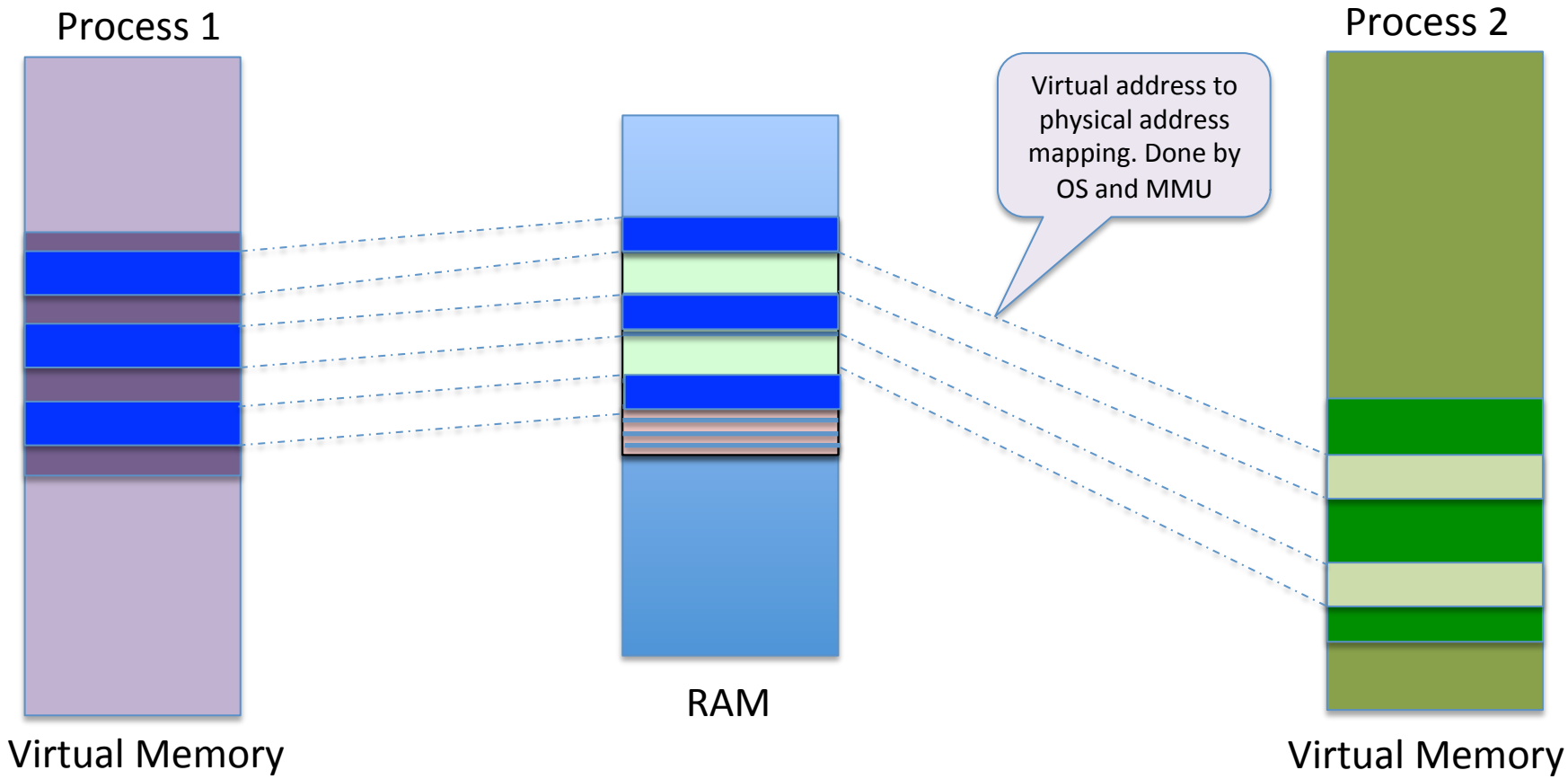
- Achieves confidentiality and integrity
  - Tampering of code / data is detected and access to tampered code / data is prevented.
- Code outside enclave cannot access code/data inside the enclave
- Even though OS is untrusted, it should still be able to manage page translation and page tables of the enclave
- Enclave code and data
  - Enclave code and data is in the clear when in the CPU package (eg. Registers / caches), but unauthorized access is prevented
  - Enclave code and data is automatically encrypted it leaves the CPU package

Physical Memory

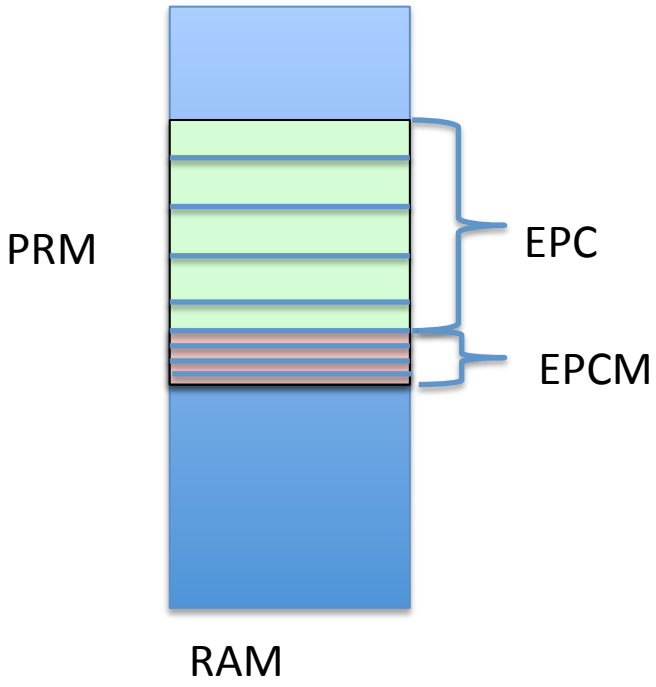


- PRM – processor related memory allocated by the BIOS. Access to PRM is blocked by external agents such as DMA, graphics engine, etc.)
  - To the other devices, this range is treated as non-existent memory
  - All SGX enclaves mapped into the PRM
- EPC Pages: Enclave page cache holds enclaves from any application.
  - Divided into 4KB pages
  - If an EPC page is valid, it either contains an SGX enclave page or EPCM (EPC micro-architecture structure)

SGX Enclaves and PRM

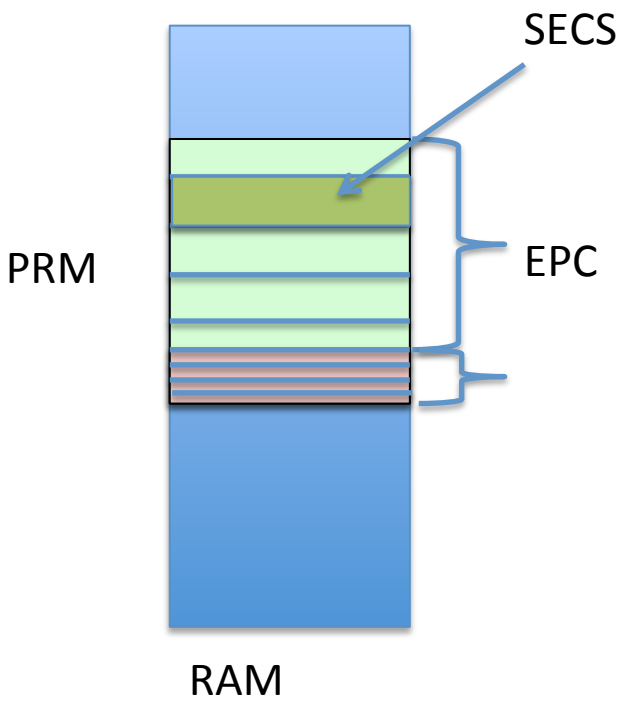


Physical Memory



- EPCM: Enclave page cache map
  - one for each EPC
  - Used by hardware for access control
  - It stores management related aspects for the corresponding EPC
    - Aspects such as valid / invalid; r/w/x permissions
    - Type of page
    - Virtual address range through which, the EPC can be accessed
    - It is an additional layer of security compared to legacy paging and segmentation since we do not trust the OS

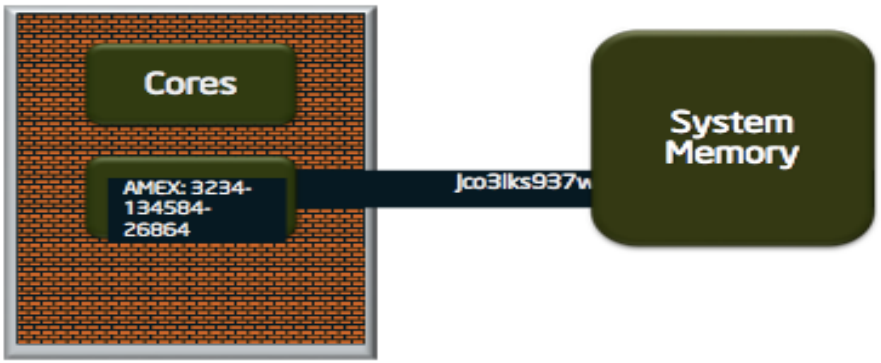
Physical Memory



- SECS: SGX Enclave Control Store
  - One for each enclave
  - 4KB (present in an EPC)
  - Contains global metadata about the enclave
    - EPC pages that are used
      - Mapping information
      - Crypto log of each used EPC page
    - Range of protected addresses used by the enclave
    - 32 / 64 bit operating mode
    - Debug access

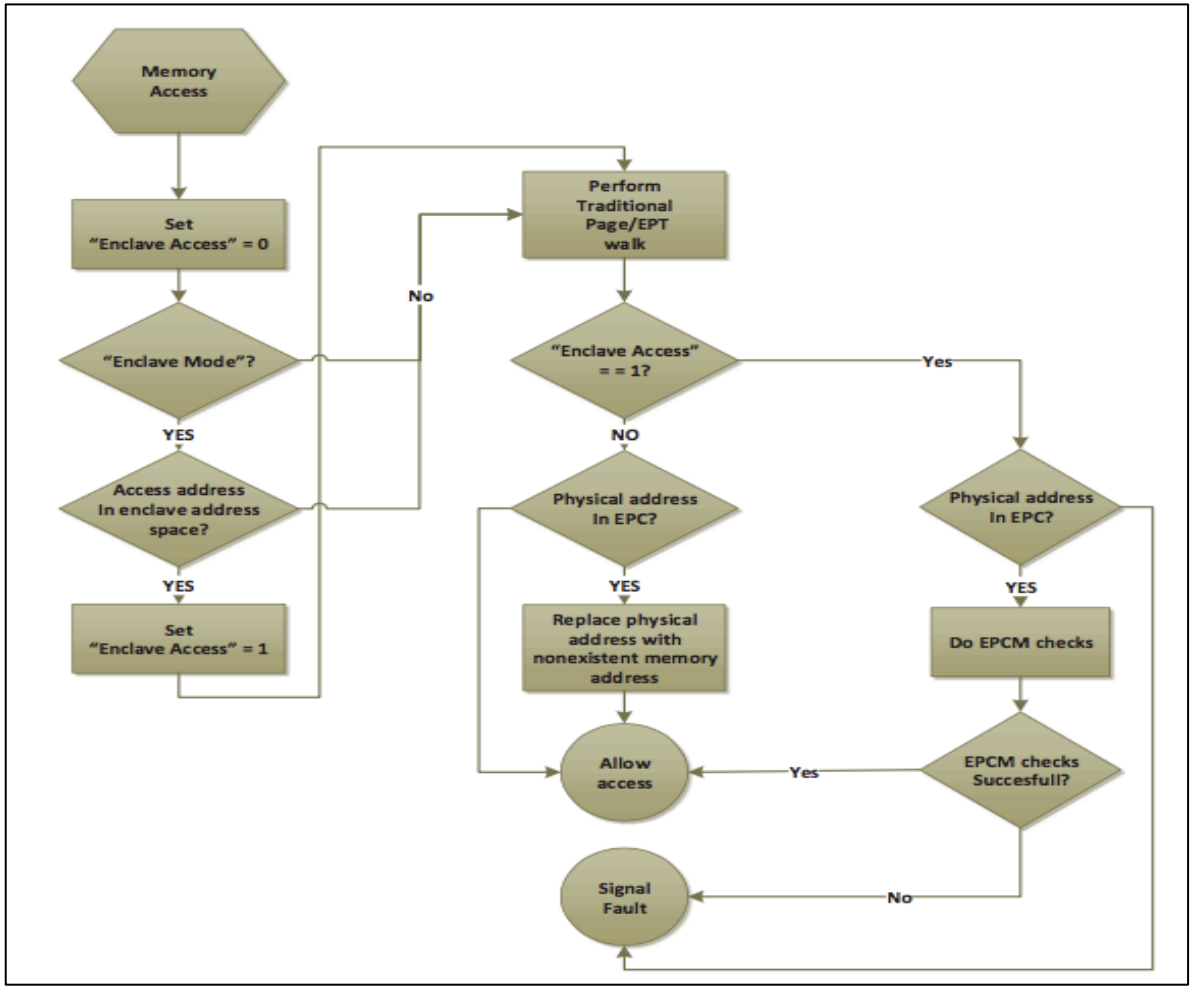
EPC Encryption

- Hardware unit that encrypts and protects integrity of each EPC



Memory Access

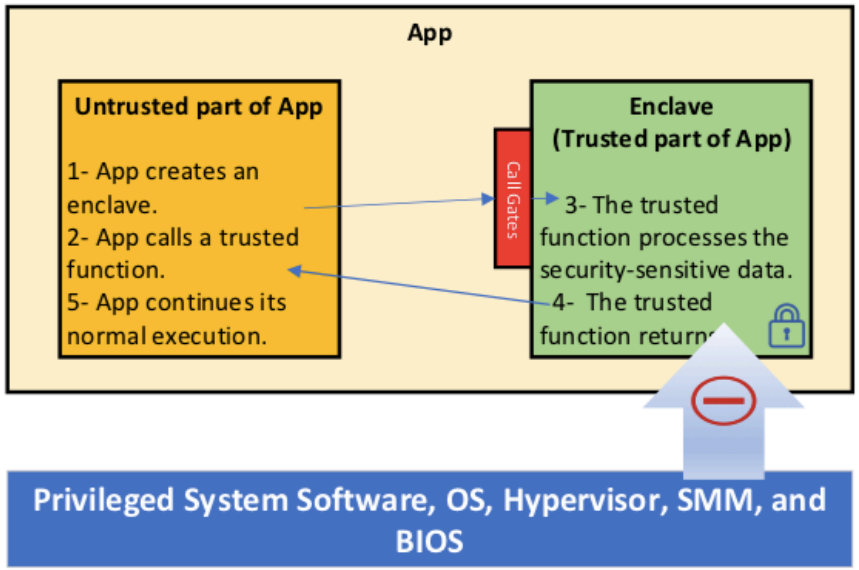
X



Application Execution Flow

App built with trusted and untrusted part

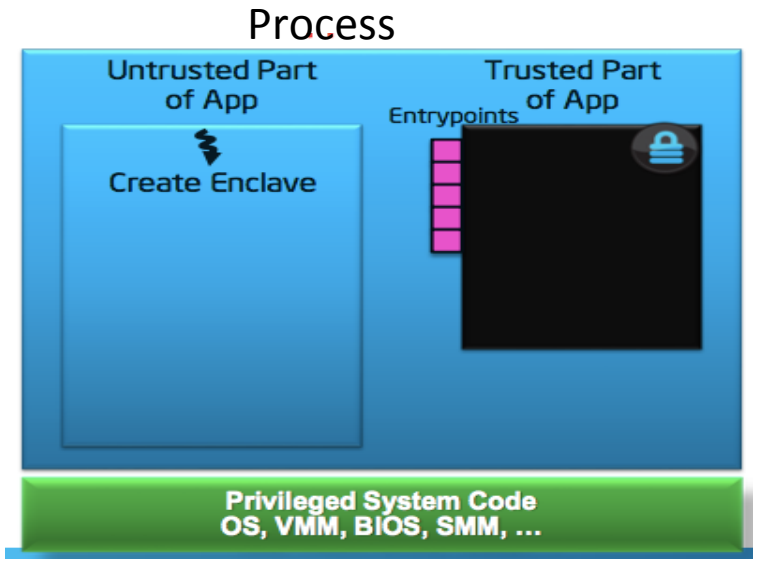
1. Untrusted part creates and executes the enclave
    1. Enclave is placed in the EPC. It is encrypted and trusted
2. Trusted function is called and execution is transferred into the enclave
3. Trusted function executes
4. Trusted function returns
5. Application continues execution



Enclave Life Cycle
(creation)

ECREATE Instruction

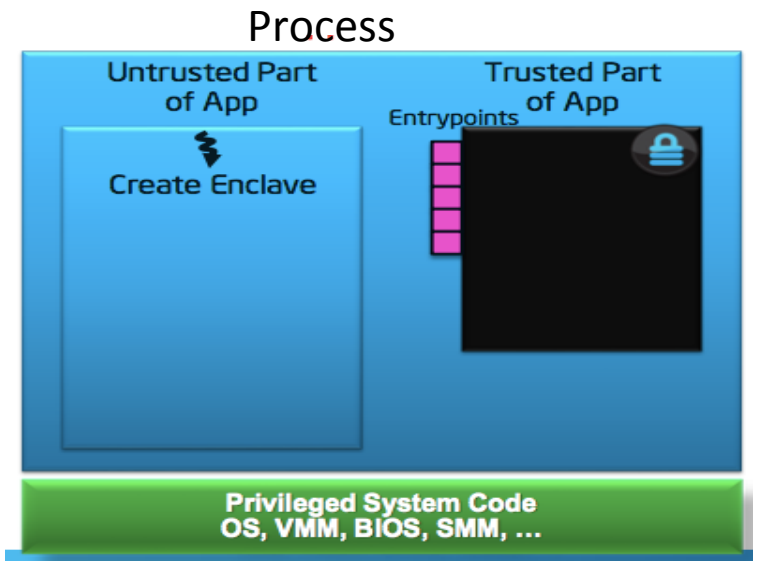
- Creates a SECS (SGX enclave control structure)
  - Contains global information about the enclave
- System software can choose where (in the process virtual space) the enclave should be present
- Also specifies
  - Operating mode (32/64 bit)
  - Processor features that is supported
  - Debug allowed



Enclave Life Cycle
(adding pages)

EADD Instruction

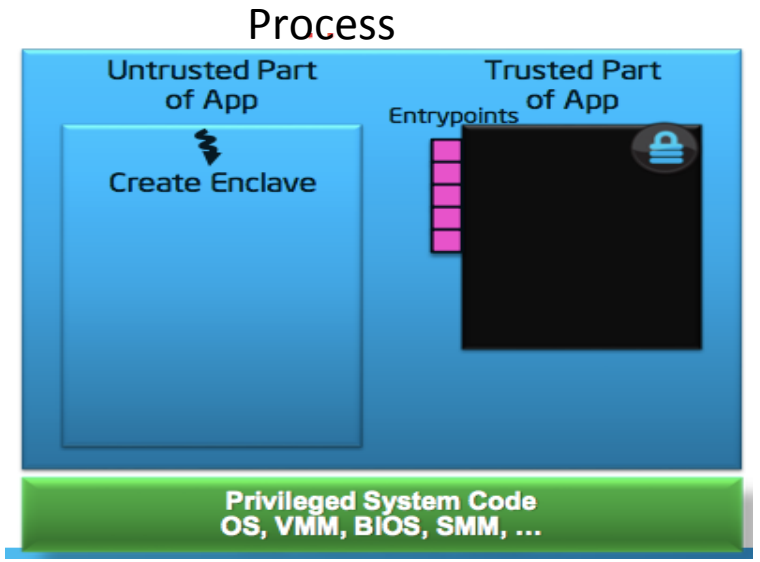
- System software should select free ECS page
- EADD will initialize EPCM with
  - Page type (TCS / REG)
  - Linear address that will access the page
  - RWX permissions
  - Associate the page in SECS structure
- EADD will then record EPCM information in a crypto log stored in the SECS
  - This is the measurement of the enclave
  - Used for gaining assurance
- Copy 4K bytes of data from unprotected memory into the enclave



Enclave Life Cycle
(measuring pages)

EEXTEND

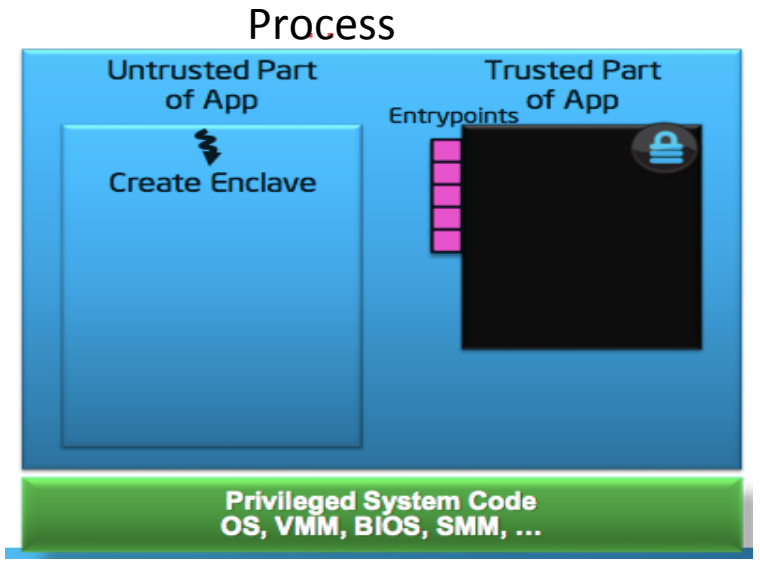
- Measure a 256 byte region in an EPC page
  - This region is specified by the developer
  - The measurement comprising of a 64 bit address and a 256 byte information in the SECS
  - 16 invocations EEXTEND needed to measure the whole page
- Correct construction of the enclave would result in a matching with the enclave owner
  - The enclave owner's signature is stored in a SIGSTRUCT structure
  - This can also be remotely verified



Enclave Life Cycle
(initializing pages)

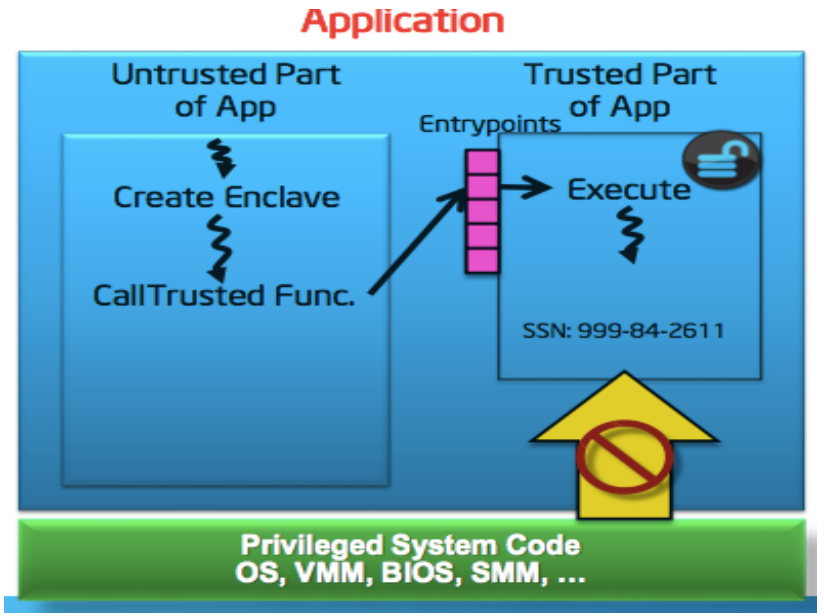
EINIT

- Should be invoked after all pages have been added
- Verify that the signature matches that of the owner's signature
- If EINIT is successful, it allows the enclave to be entered



Enclave Life Cycle
(enter/exit)

Process invokes the enclave through pre-defined entry points using EENTER instruction



EENTER

- Changes made to enclave mode
- Need to know the location to transfer control and location where to save state in case of an interrupt
- Defines an Asynch. Exit pointer, which where IRET returns to after servicing an interrupt
  - It is outside the enclave
  - And typically will have an instruction **ERESUME**

Entry into the Enclave

- Set TCS to busy
- Change mode to enclave mode
- Save state of SP, BP, etc. for return in case of async. Exit
- Save AEP
- Transfer control from outside the enclave to inside

Exit from Enclave

- **EEXIT**
  - Clear enclave mode and flush TLB entries
  - Mark TCS as free.
  - Transfer control outside the enclave

Asynchronous Exit (AEX)

- Occurs when an interrupt / exit occurs
- Processor state is securely saved inside the enclave and replaced with synthetic states
- AEP pushed onto the stack
  (AEP is a location outside the enclave where execution goes to after IRET)
- After AEX completes, the logical processor is no longer in enclave mode

- Resuming after an interrupt
  - EERESUME instruction is invoked, which restores all registers
  - Typically EERESUME is present at the AEP location
- Resuming after a fault that occurred in the enclave?
  - Eg. A divide by zero

Instruction set Extensions for SGX

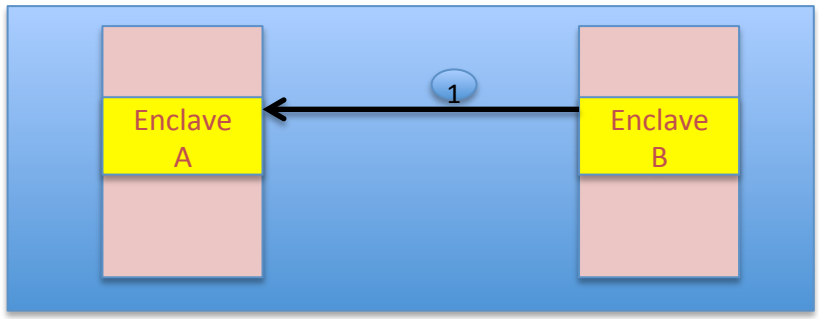
- Privileged Instructions
  - Creation related: to create, add pages, extend, initialize, remove enclave
  - Paging related: evict page, load an evicted page
- User level instructions
  - Enter enclave, leave enclave
  - Interrupt related: asynchronous exit, resume

Attestation

- system proves to somebody else that it has a particular SGX enclave
- Two attestation techniques
  - Intra machine (prove to another enclave in the same machine)
  - Inter machine (prove to a third party)
- Makes use of a register called MRENCLAVE
  - Contains the SHA-256 hash of an internal log that measures the activity done by the enclave
    - The log contains the pages (code, data, stack, heap) in the enclave
    - Relative position of the pages in the enclave
    - Security flags associated with the pages

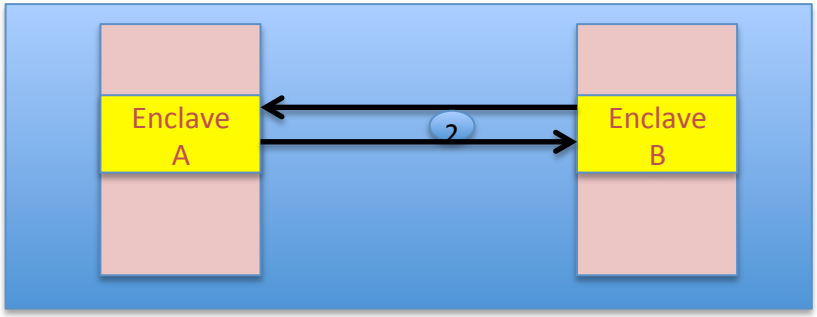
Innovative Technology for CPU Based Attestation and Sealing, HASP 2015, Ittai Anati et al

Intra-Platform Enclave Attestation



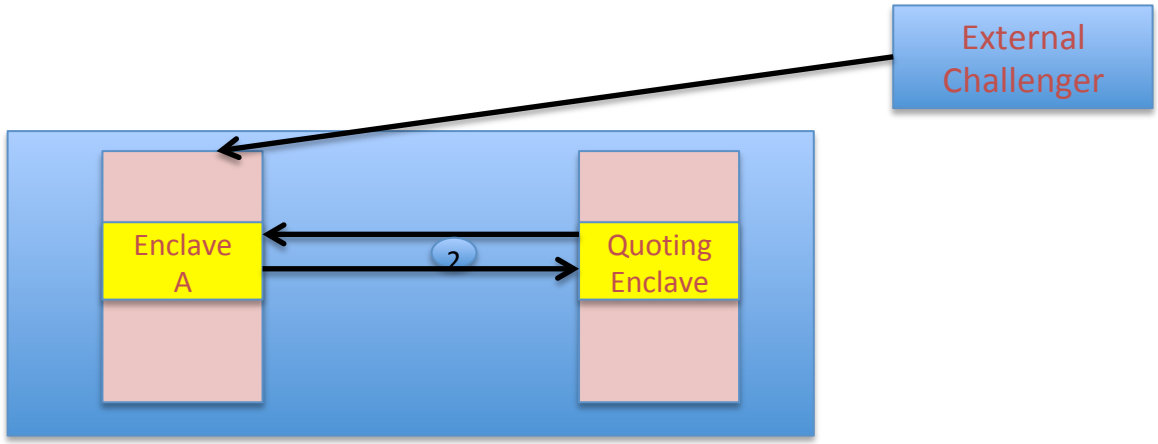
- (1) Enclave A obtains enclave B's MRENCLAVE
- Enclave A invokes EREPORT together with B's MRENCLAVE to create a signed report destined for enclave B
  - Enclave contains: attributes associated with the enclave
  - Attributes of the Trusted Control Block
  - MAC (produced by a key called report key, which is known only to the hardware and Enclave B)

Intra-Platform Enclave Attestation



- (1) Enclave A obtains enclave B's MRENCLAVE
- Enclave A invokes EREPORT together with B's MRENCLAVE to create a signed report destined for enclave B
- (2) Enclave A sends the report to B, via an untrusted channel
- Enclave B calls EGETKEY to retrieve the report key, re-computes the MAC accompanying the REPORT. If there is a match with the MAC, then the enclave is indeed running on the same machine.
- Once the MACs have been verified, Enclave B can verify Enclave A's report using the MRENCLAVE it just received

Inter-Platform Enclave Attestation



- Quoting enclave and external system uses asymmetric crypto. to transfer a quote to the external system