

# Intel SGX

Mengjia Yan

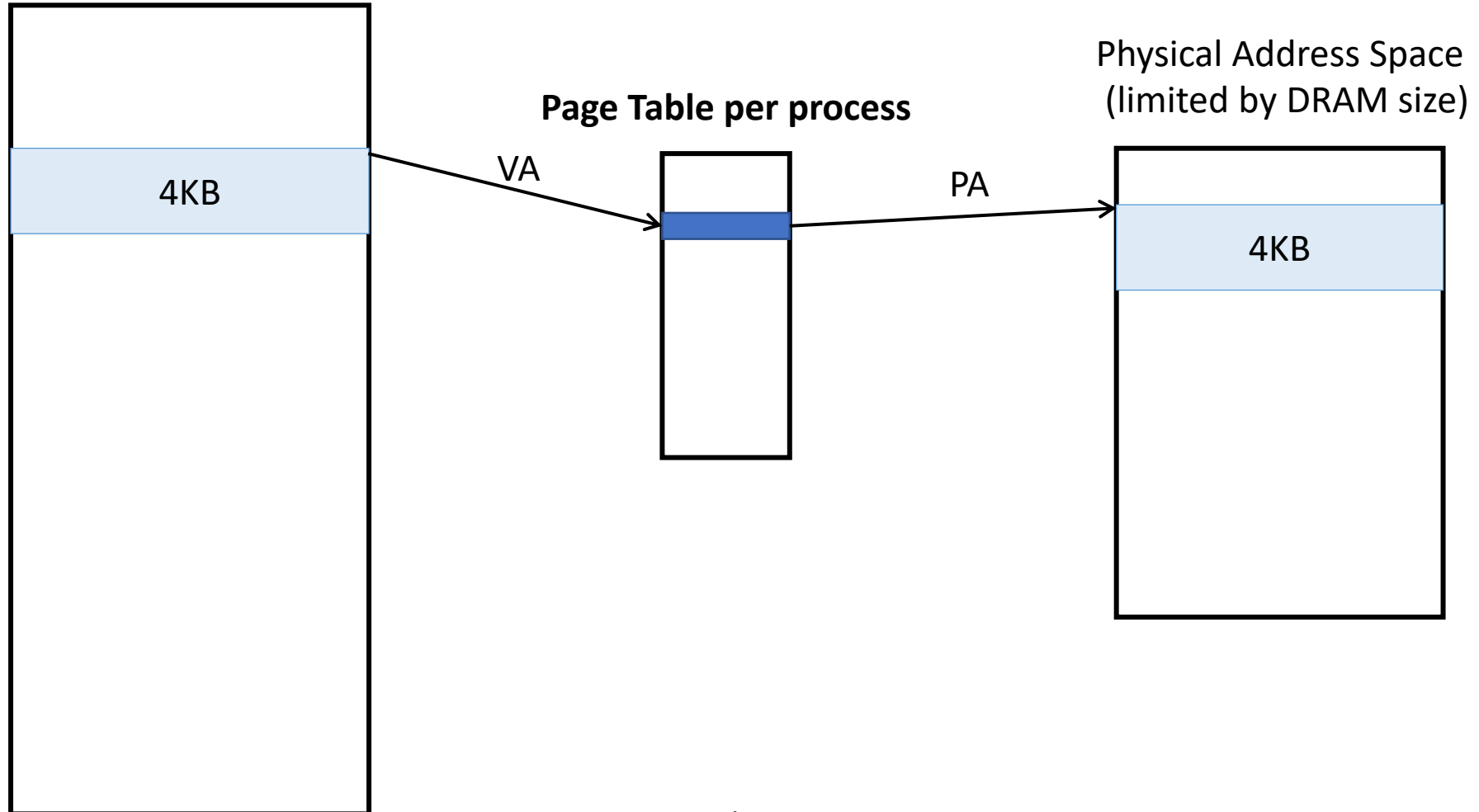
Fall 2020

Based on slides of Intel SGX Tutorial



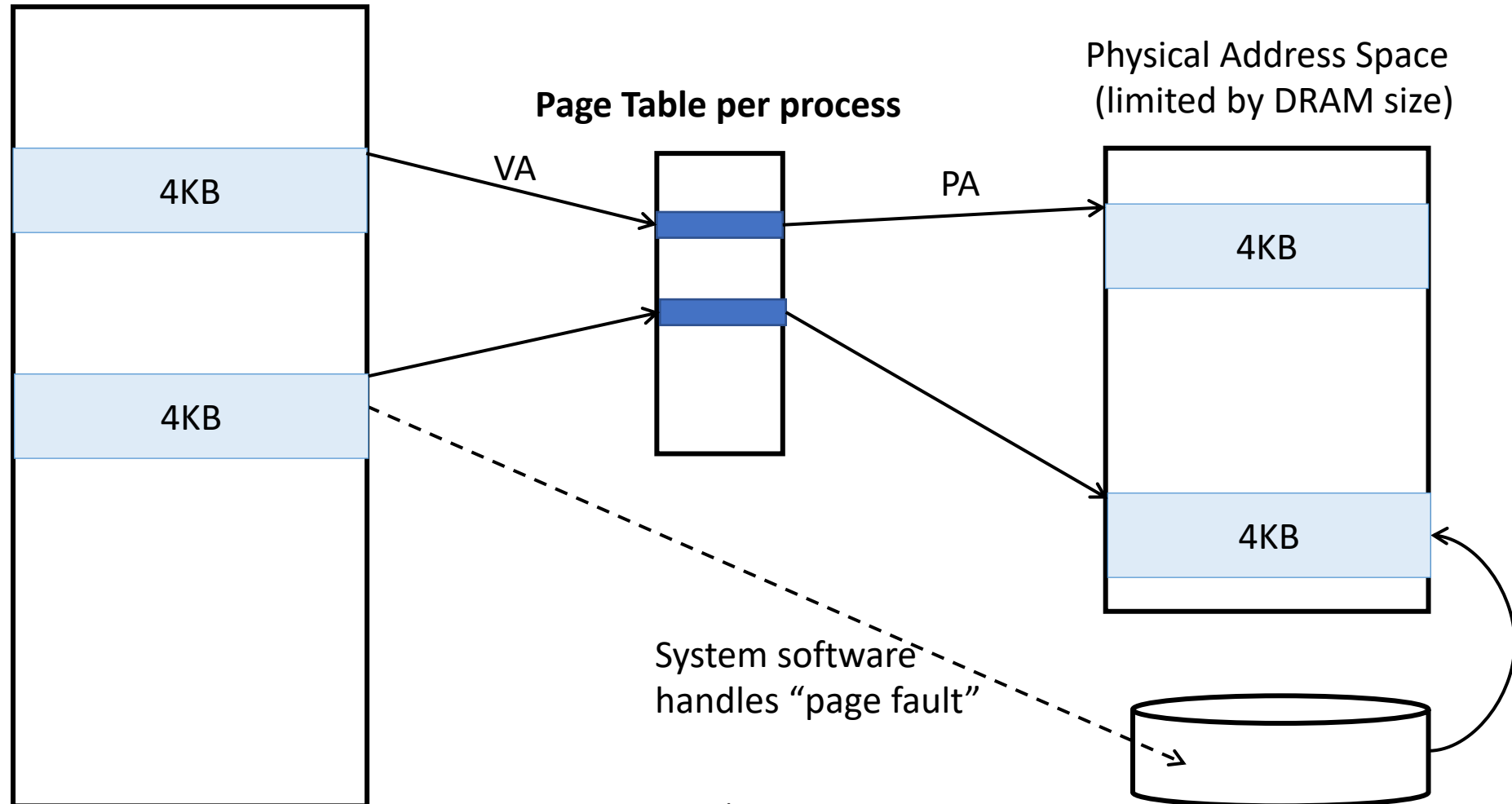
# Recap: Address Translation

Virtual Address Space (Programmer's View)

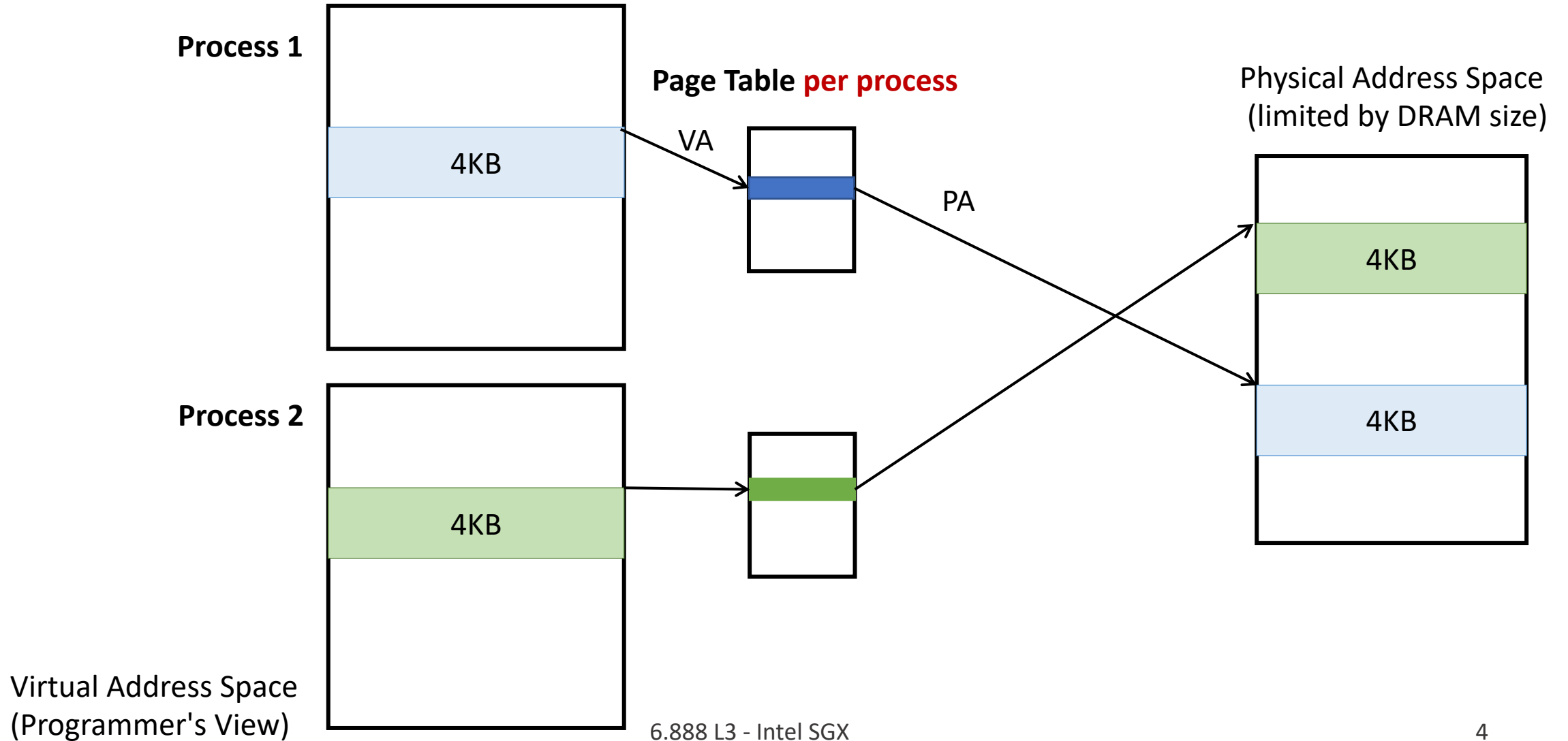


# Recap: Address Translation

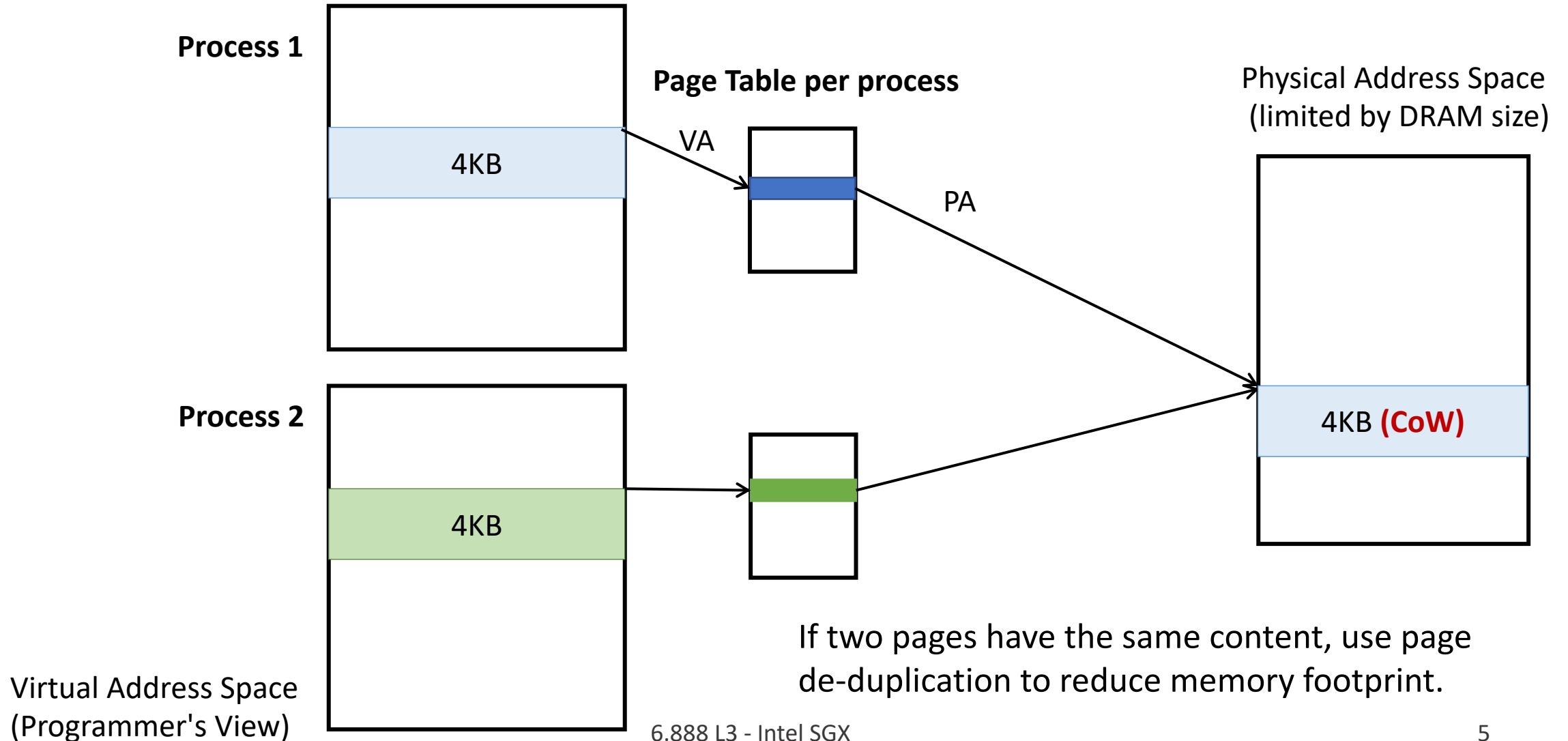
Virtual Address Space (Programmer's View)



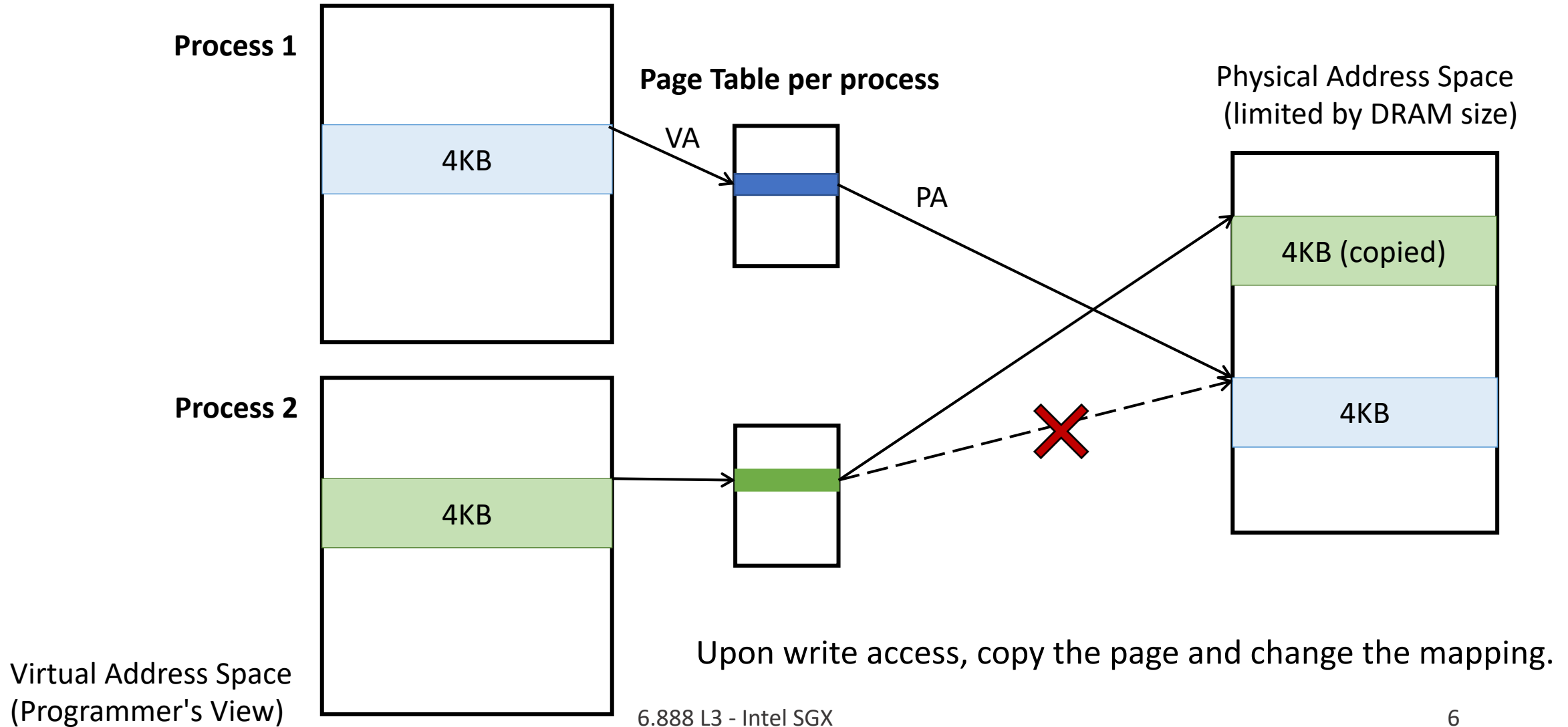
# Recap: Process Isolation



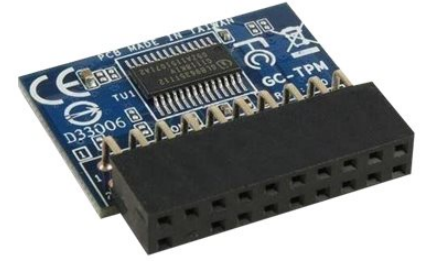
# Page De-duplication and Copy-on-write



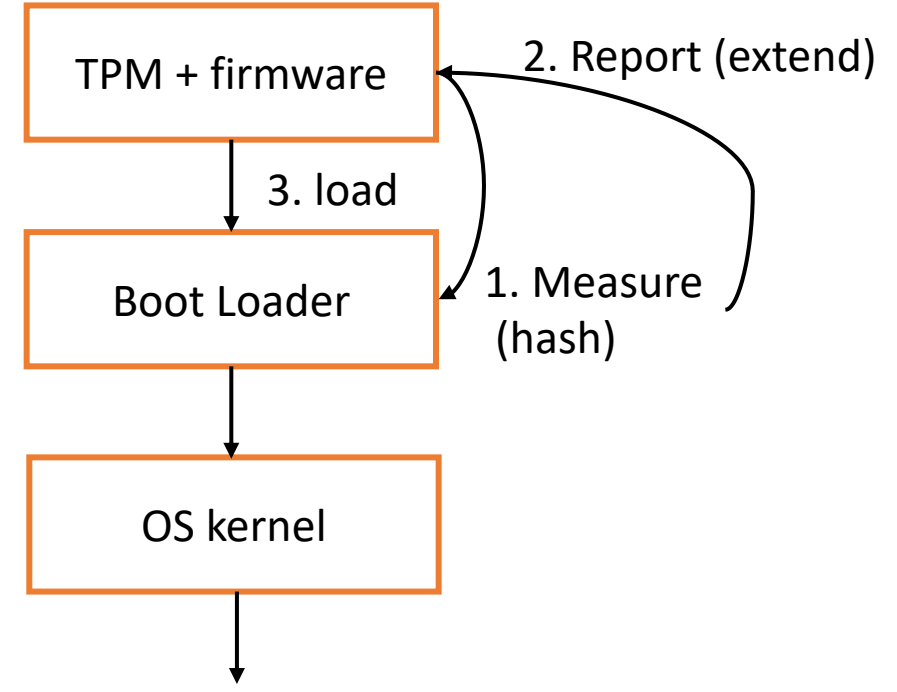
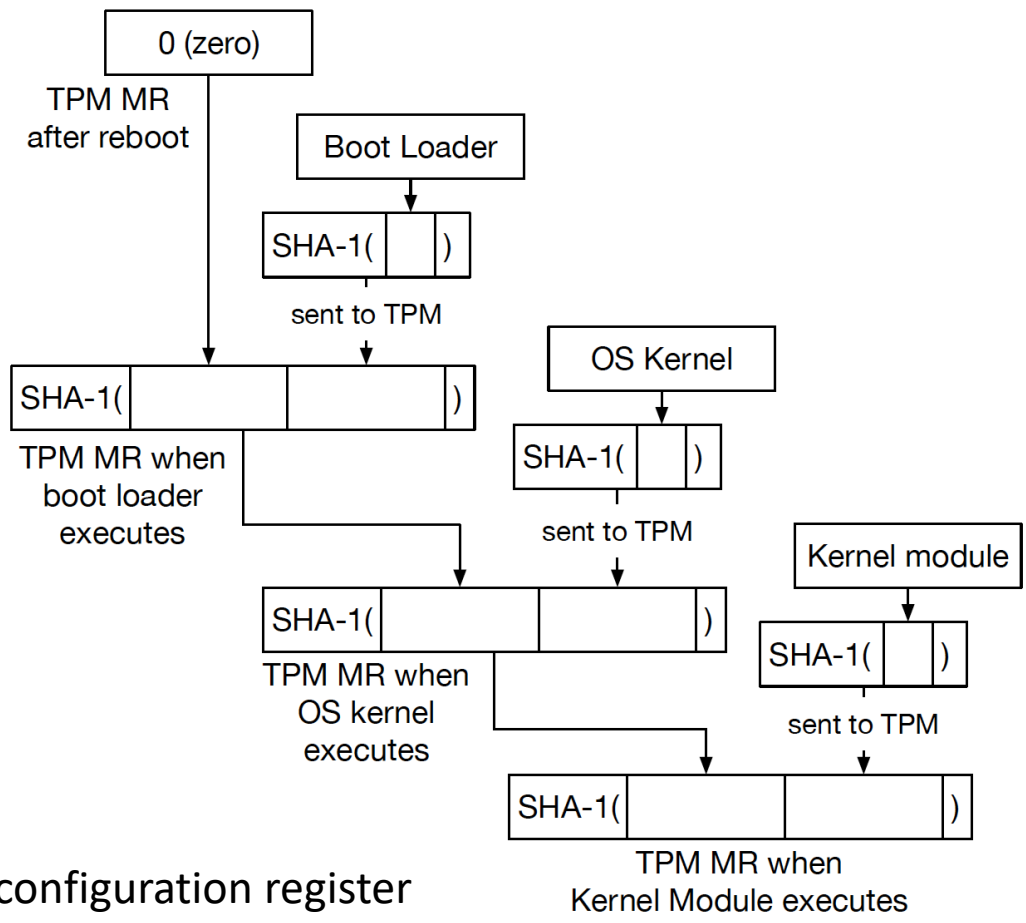
# Page De-duplication and Copy-on-write



# Recap: Secure Boot



- Static root of trust for measurement (SRTM)

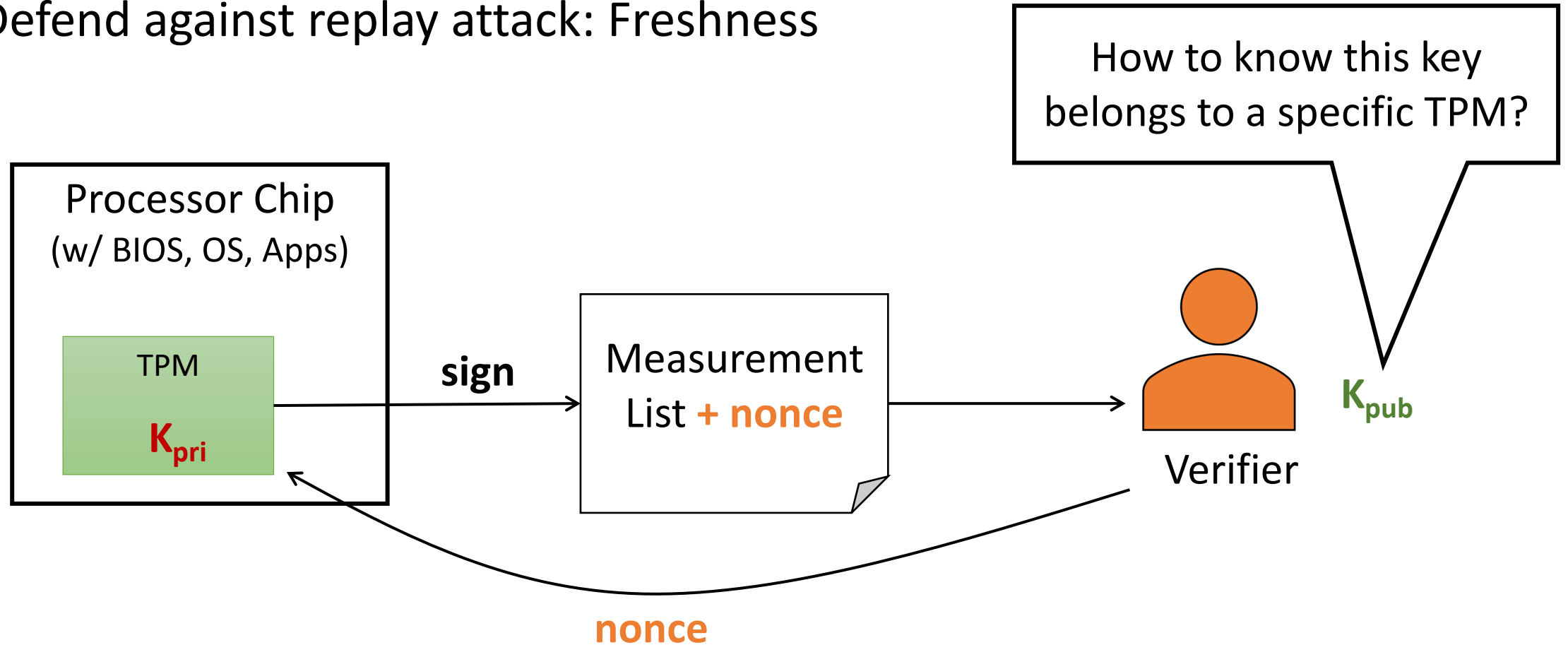


Compared to expected values locally or submitted to a remote attester.

PCR: platform configuration register

# Software Attestation

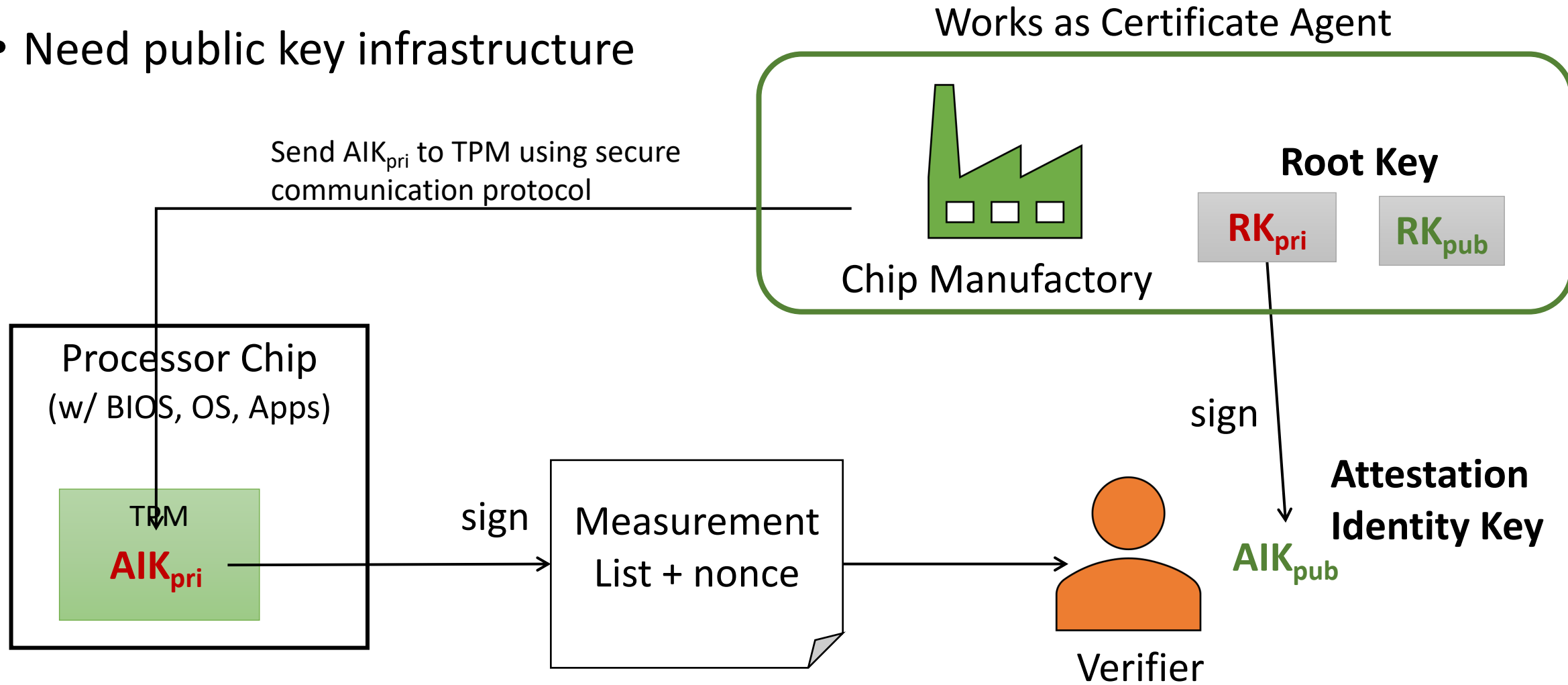
- Defend against replay attack: Freshness



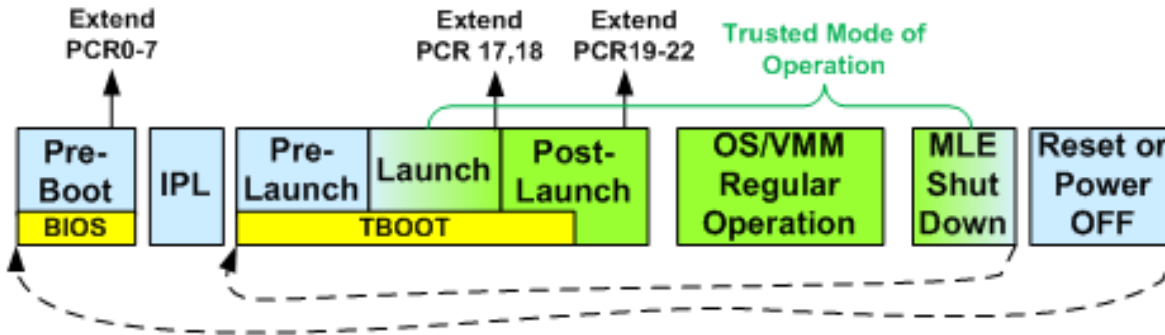


# Software Attestation

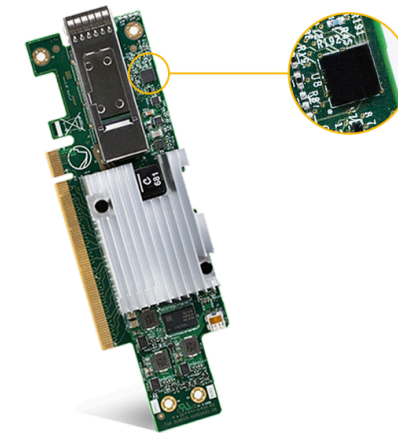
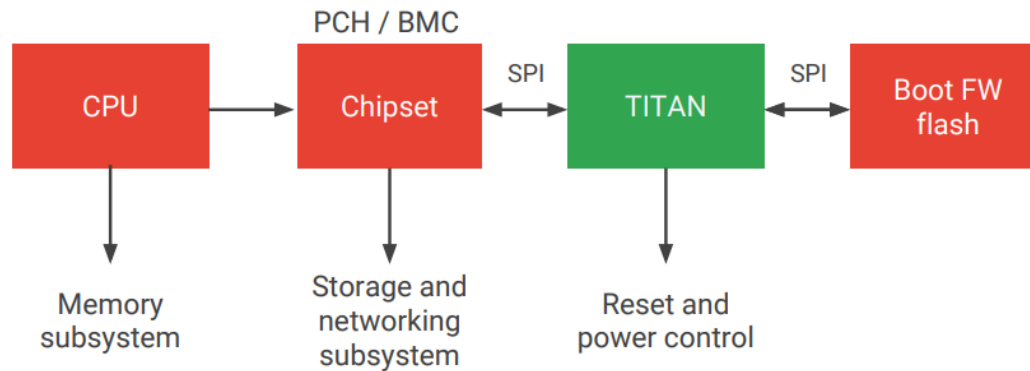
- Need public key infrastructure



# Intel TXT, AMD PSP, Google Titan



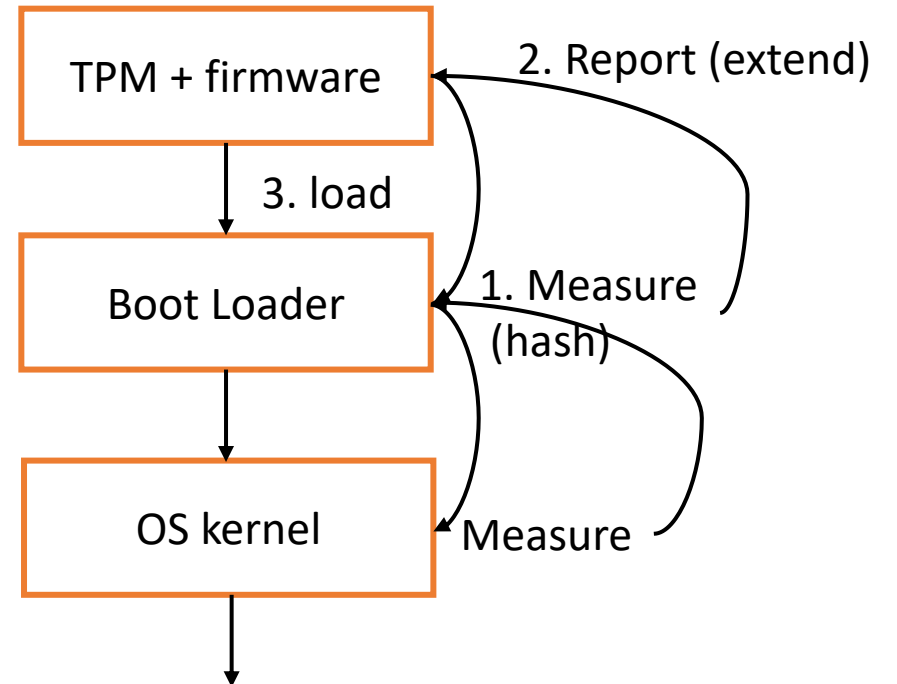
Intel TXT Dynamic trust of measurement



from [https://www.hotchips.org/hc30/1conf/1.14\\_Google\\_Titan\\_GoogleFinalTitanHotChips2018.pdf](https://www.hotchips.org/hc30/1conf/1.14_Google_Titan_GoogleFinalTitanHotChips2018.pdf)

# Security Vulnerabilities of Using TPM

- Vulnerable to bus tapping attacks
- TPM Reset attacks
  - SW reports hash values
- Bugs in the trusted software

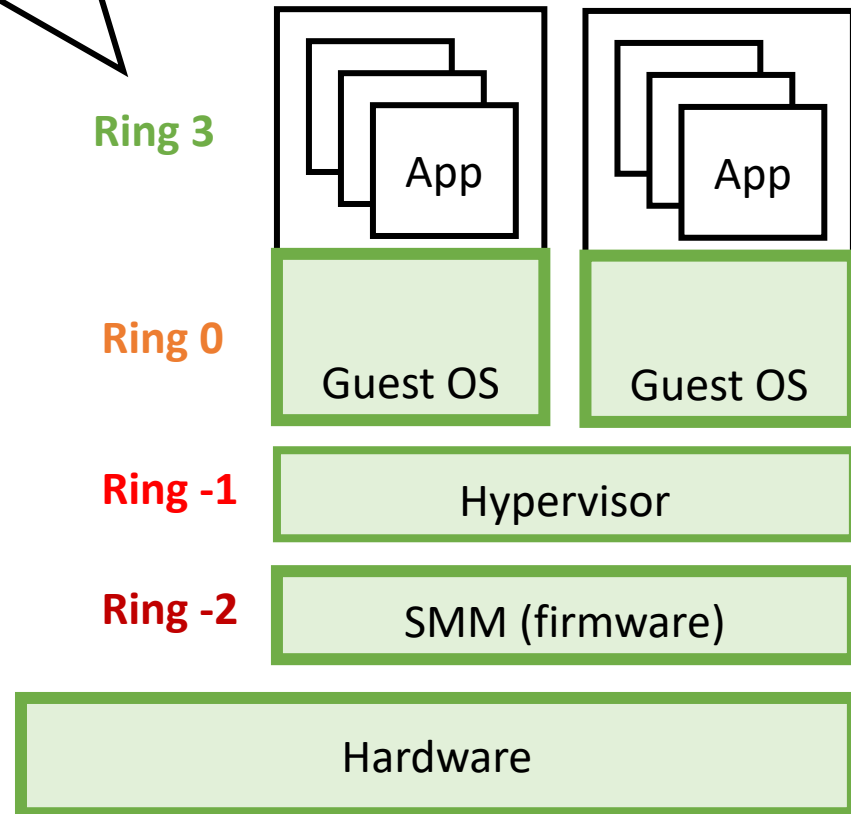
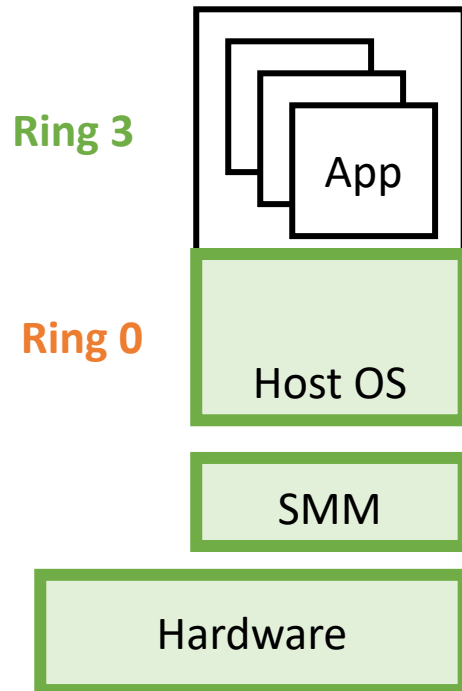


*Han et al. A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping. Usenix Security'18*  
Wojtczuk et al. Attacking Intel TXT® via SINIT code execution hijacking. 2011

# So Far .....

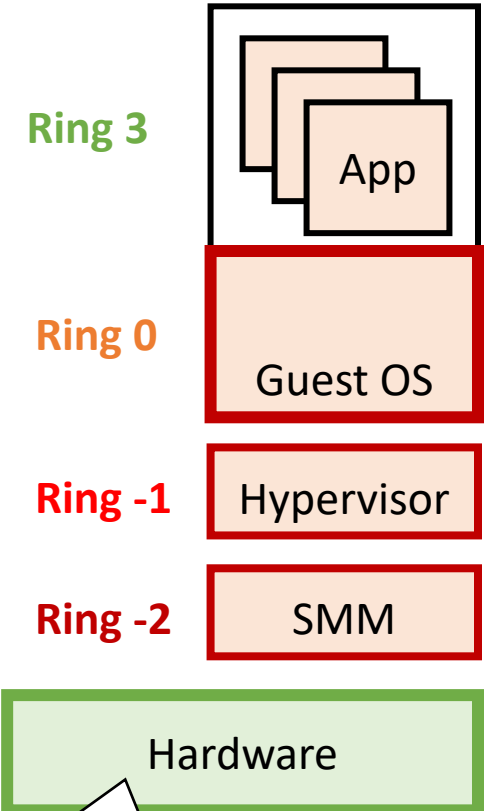
The trend: shrink TCB.  
Why?

Trusted



# Why Shrink TCB?

- Software bugs
  - SMM-based rootkits
  - Xen 150K LOC, 40+ vulnerabilities per year
  - Monolithic kernel, e.g., Linux, 17M LOC, 100+ vulnerabilities per year
- Remote Computing
  - Remote computer and software stack owned by an untrusted party
  - Examples



Shrink HW TCB?

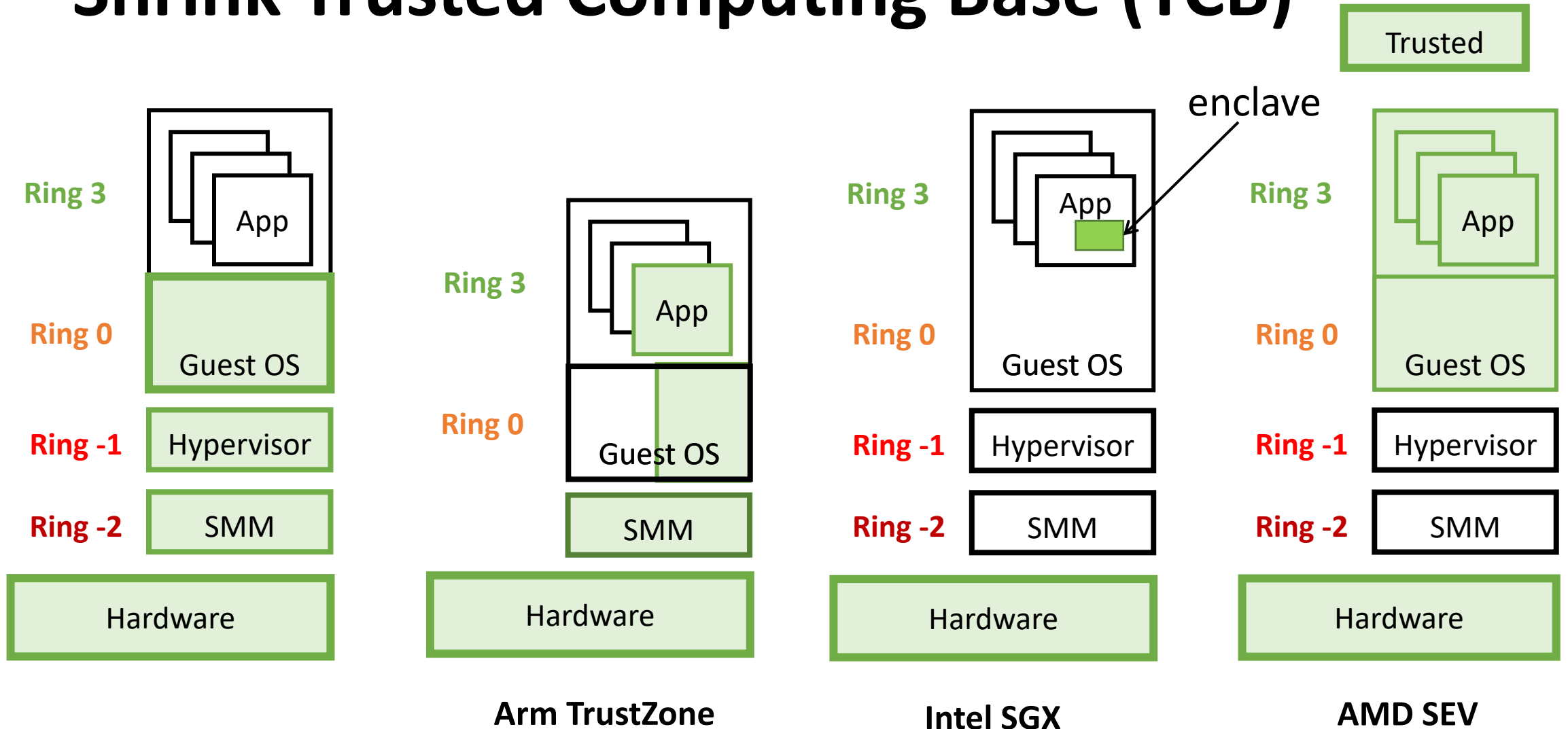
# Secure Remote Computing



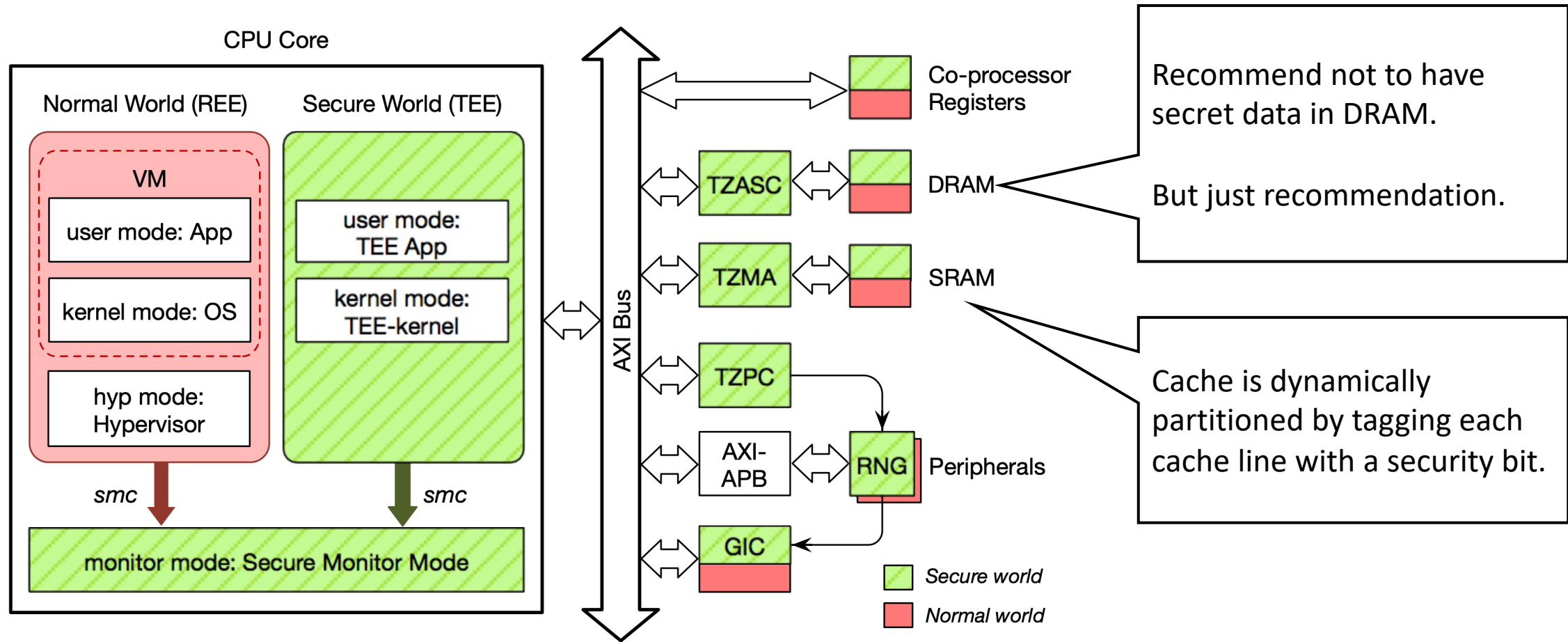
- Example: Video processing



# Shrink Trusted Computing Base (TCB)



# Arm TrustZone

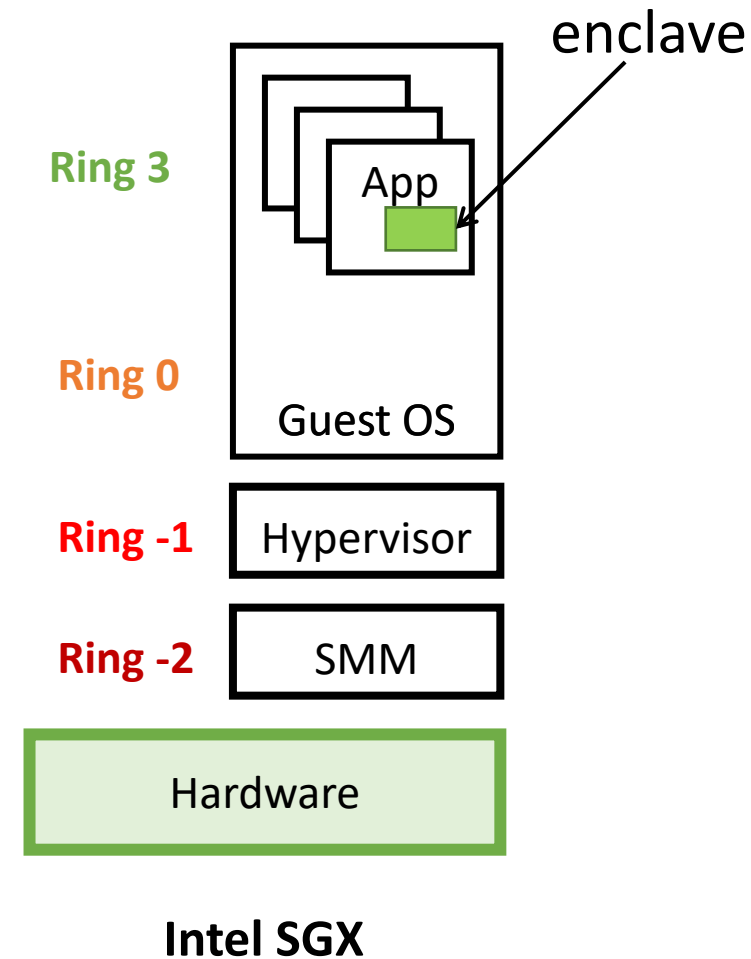


from Hua et al. vTZ: Virtualizing ARM TrustZone. Usenix'17



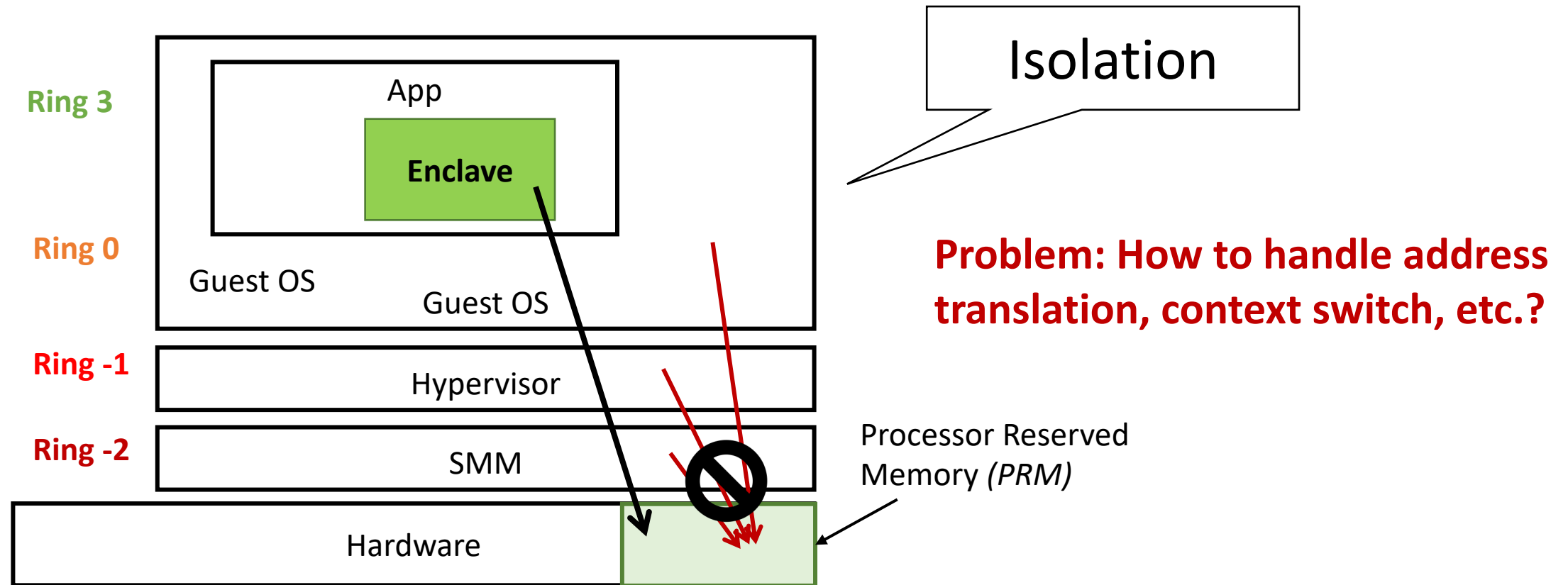
# Privileged Software Attacks

- Manipulate everything
- Directly see and modify application code and data
  - Need to encrypt secret data
  - Need to verify integrity (software attestation)
- Mess up with
  - Address translation
  - Process initialization and context switch
  - Interrupts, I/Os
  - etc.

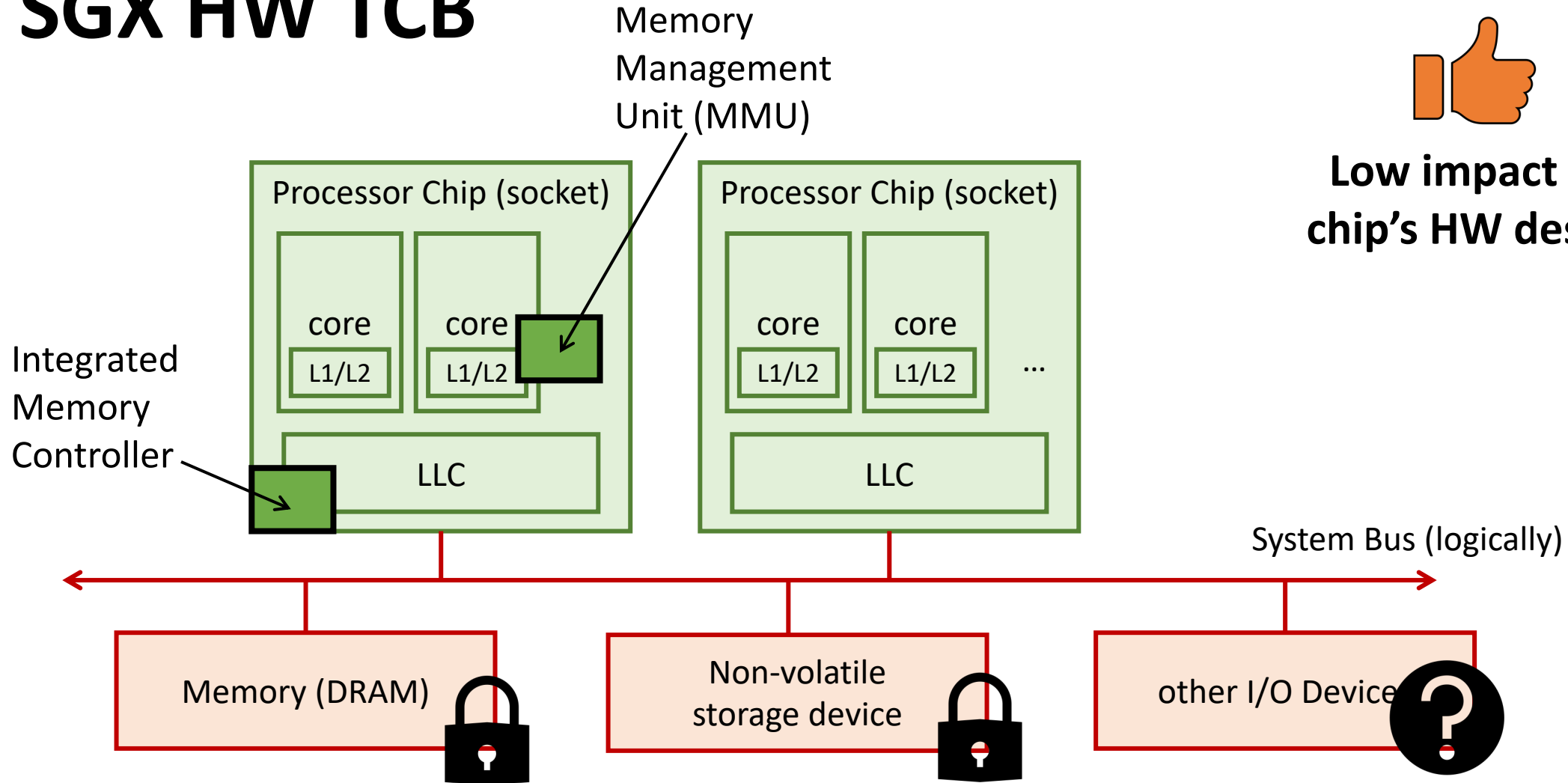


# Enclave High-level View

- Goal: A protected environment that contains the code and data of a security-sensitive computation.



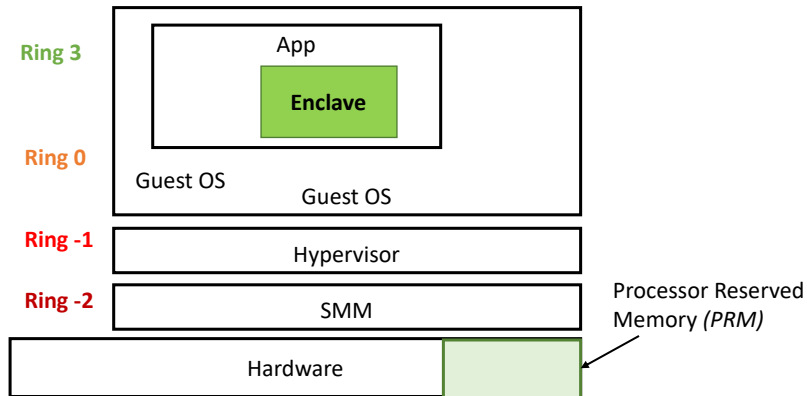
# SGX HW TCB



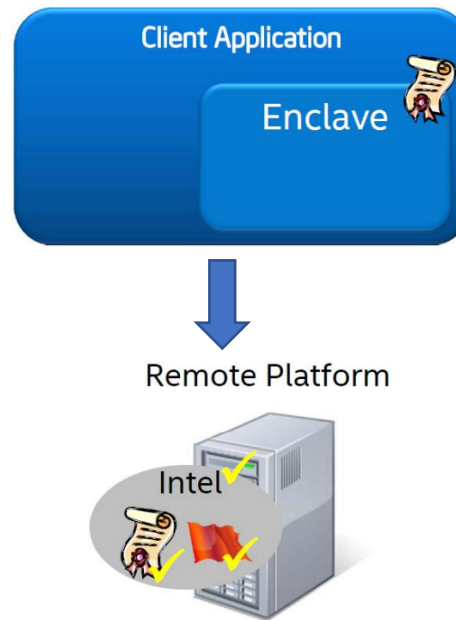
**Low impact on  
chip's HW design**

# Intel SGX Security Mechanisms

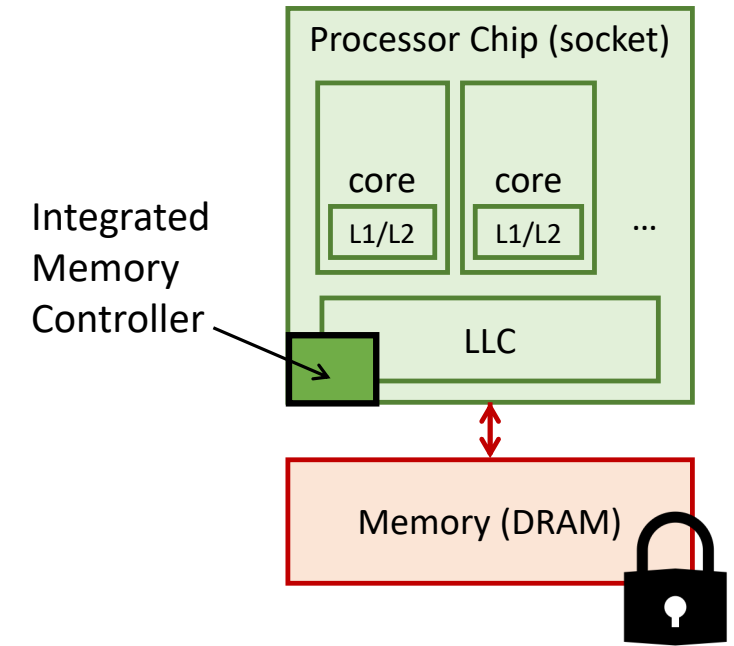
## Isolation



## Attestation

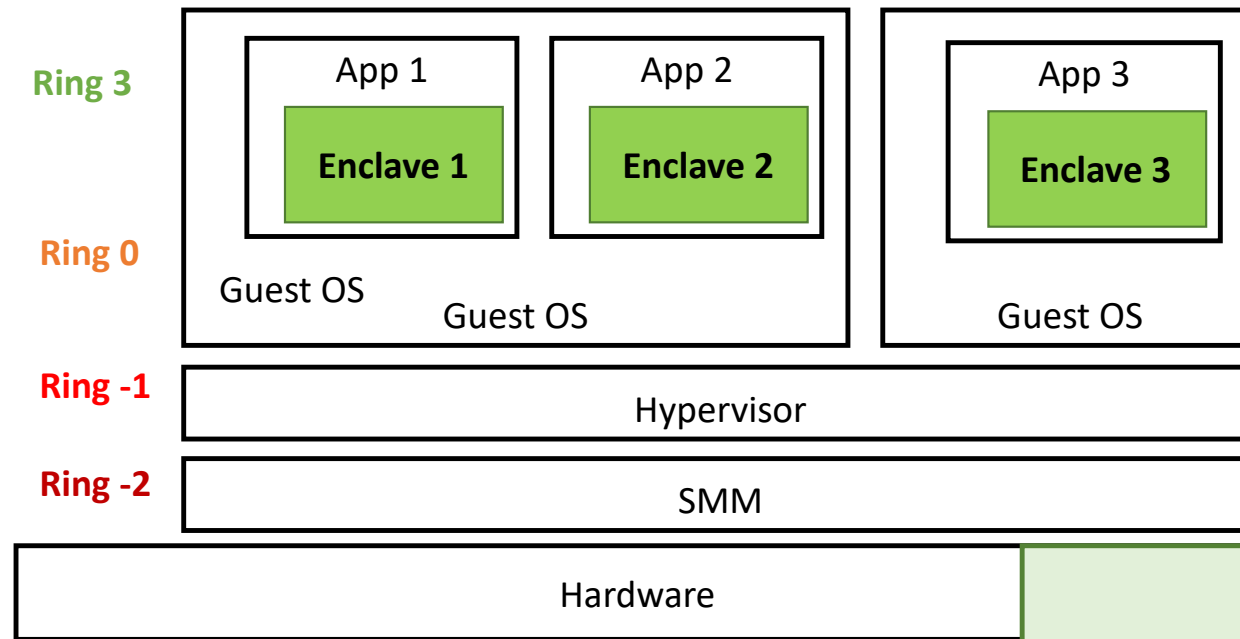


## DRAM Protection



# SGX Access Control

- Assume software attestation is done
- Can have multiple enclaves

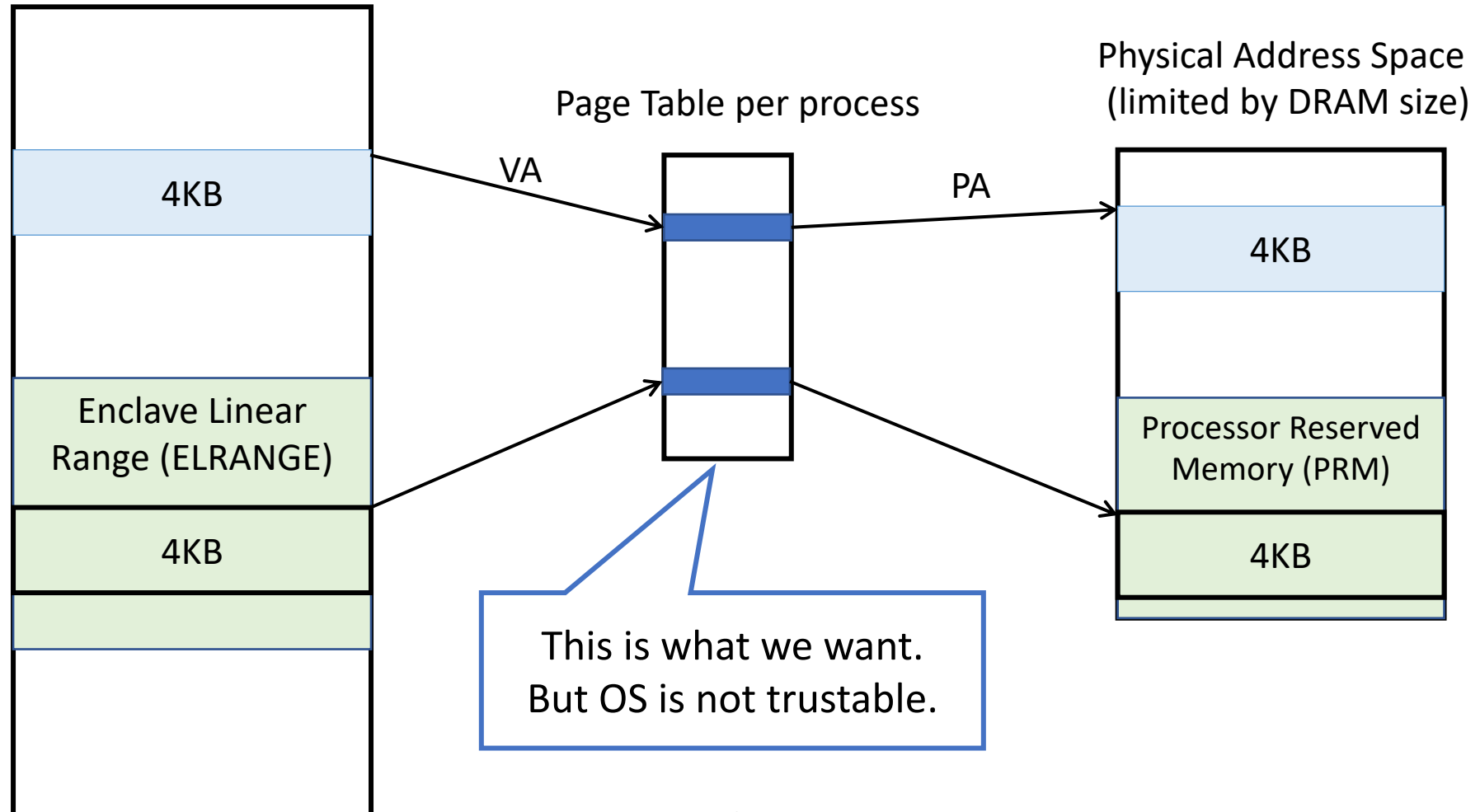


Performance issues.  
PRM size is 128MB in SGX V1.0  
***All enclaves loaded at the same time cannot exceed said ~90MB***

Processor Reserved Memory (PRM)

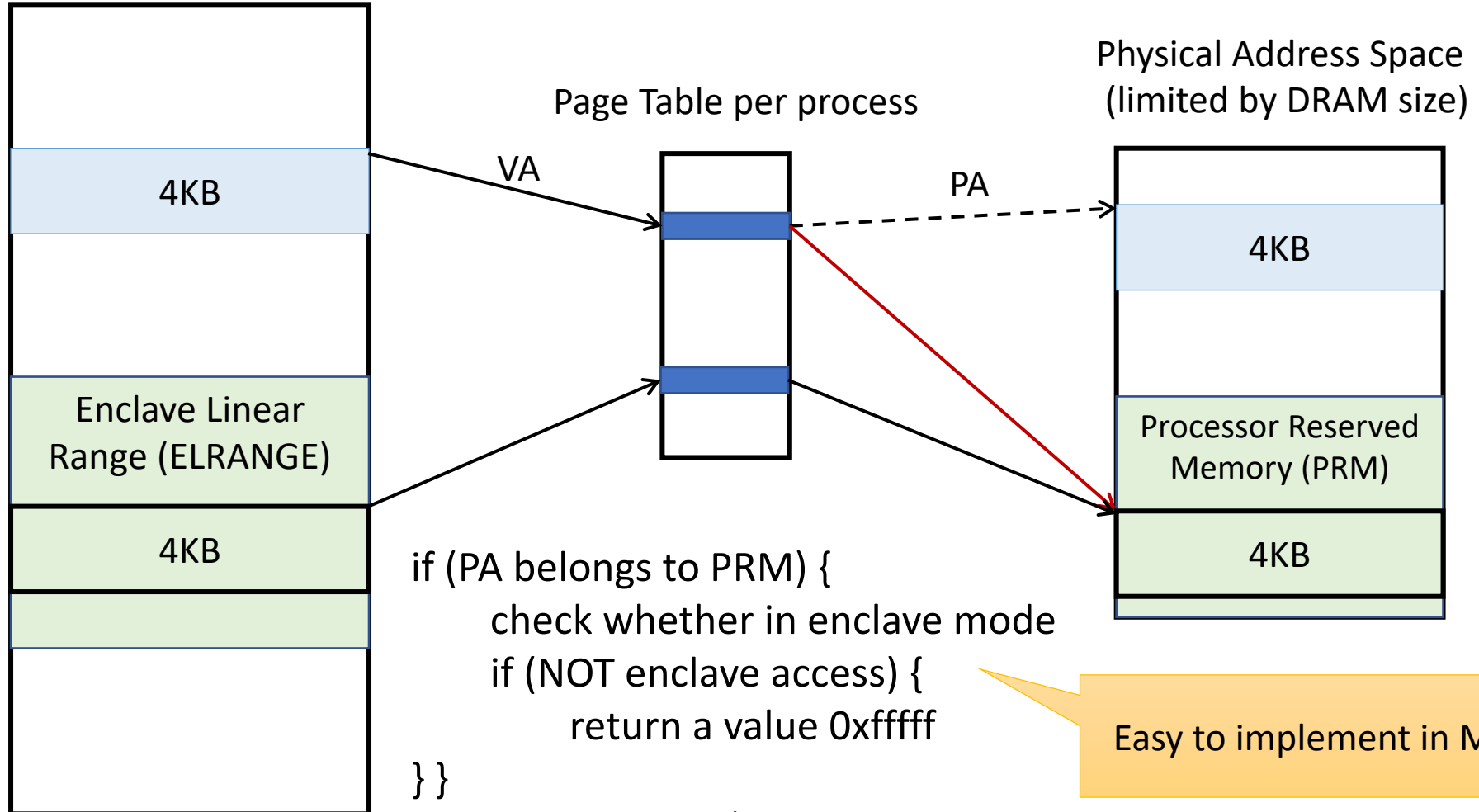
# Enclave Address Translation

Virtual Address Space (Programmer's View)



# Malicious Address Translation

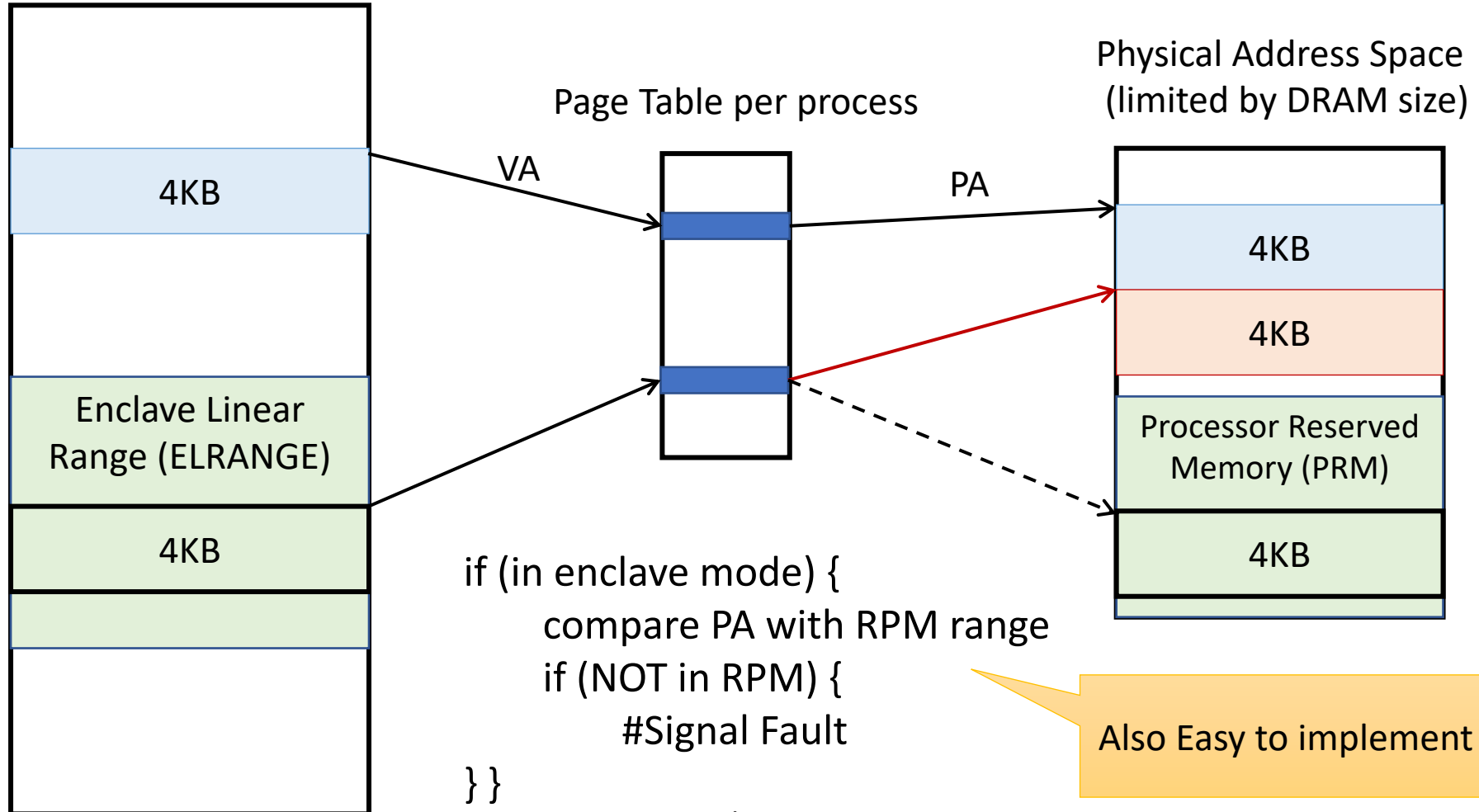
Virtual Address Space (Programmer's View)



Easy to implement in MMU

# Malicious Address Translation

Virtual Address Space (Programmer's View)

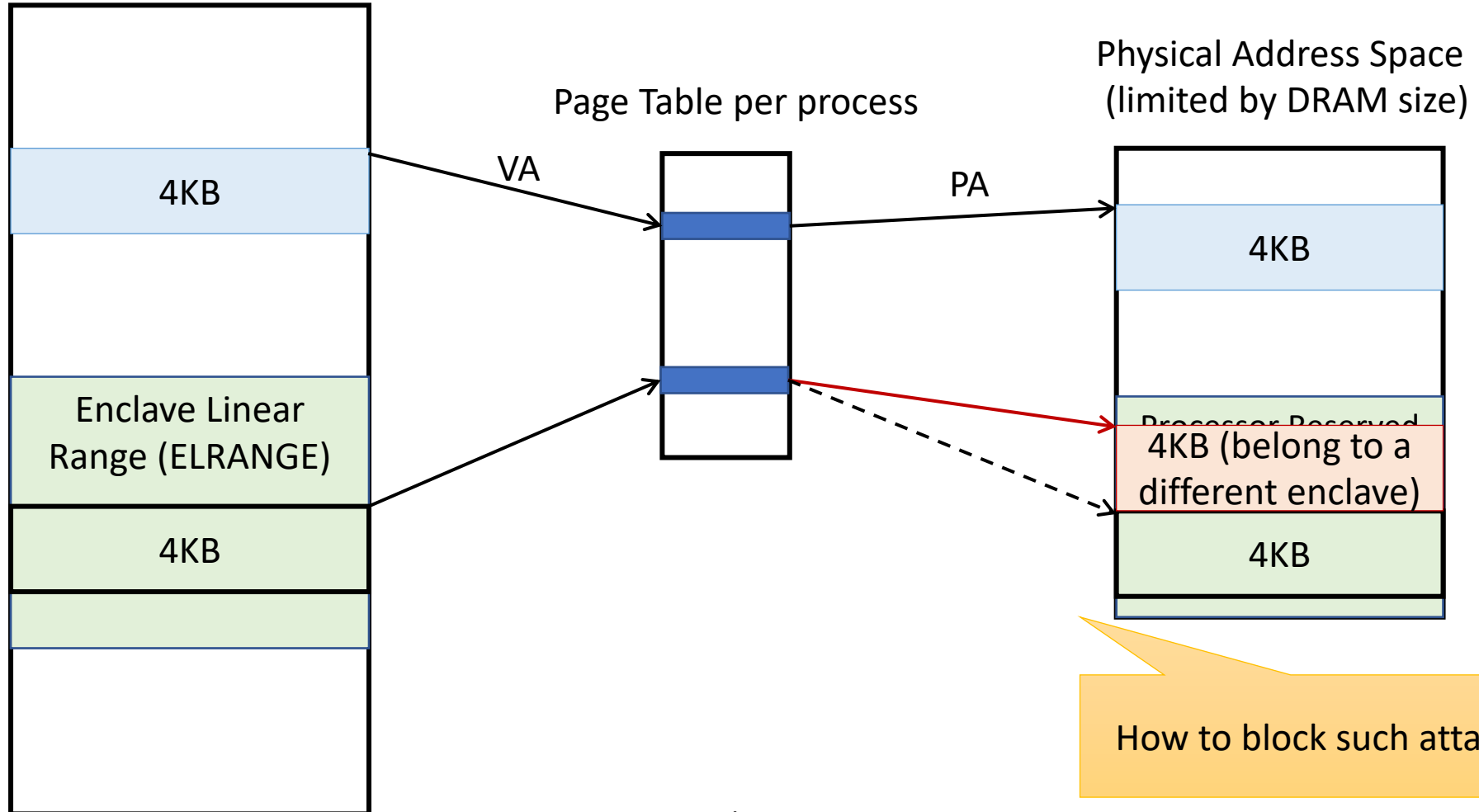


Also Easy to implement in MMU



# Malicious Address Translation

Virtual Address Space (Programmer's View)

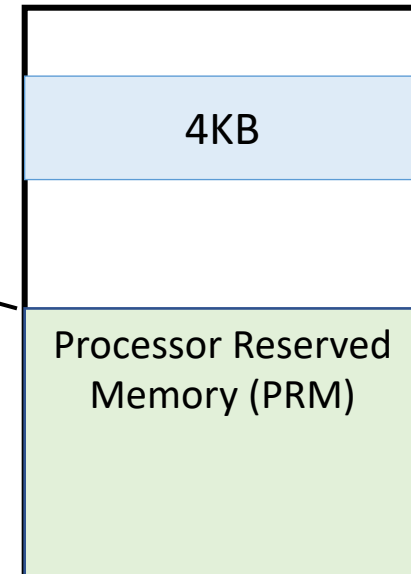


# SGX Memory Organization

- Keep page mapping metadata in PRM
- MMU performs extra checks

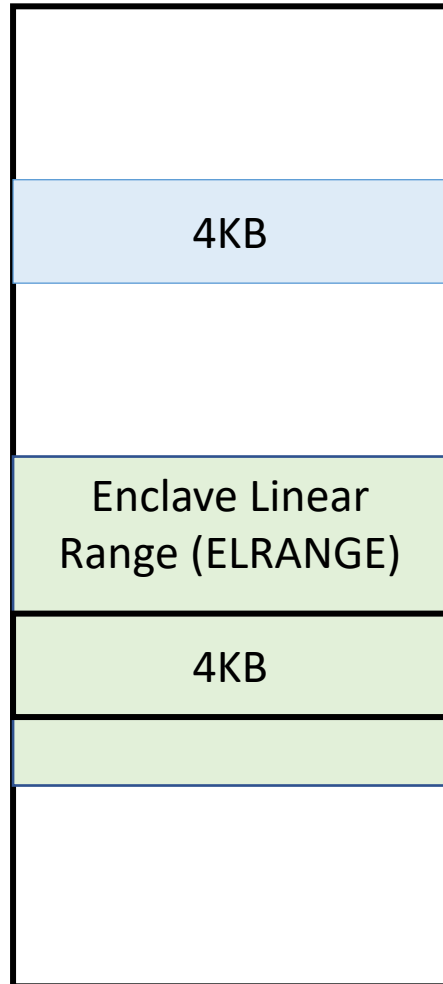
- Enclave pages (code, data)
- Meta data per enclave
  - enclave page mapping information, enclave thread context information, etc.

Physical Address Space  
(limited by DRAM size)

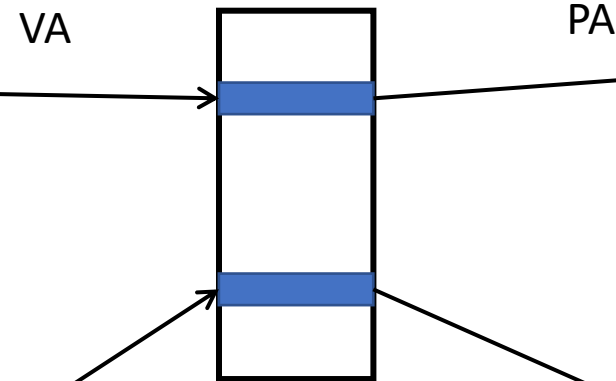


# Enclave Page Mapping Information

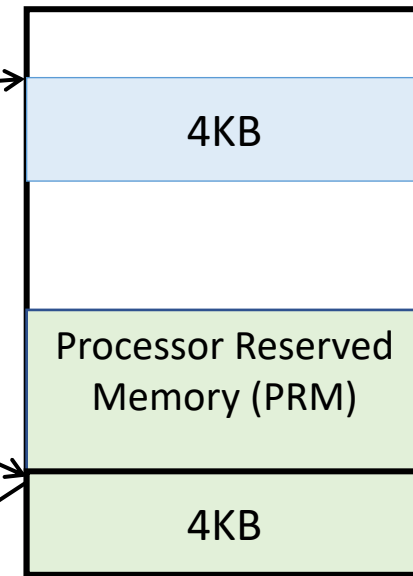
Virtual Address Space (Programmer's View)



Page Table per process



Physical Address Space (limited by DRAM size)



Enclave Page Cache Mapping (EPCM)

Stored in PRM



{PA, VA, Enclave ID}

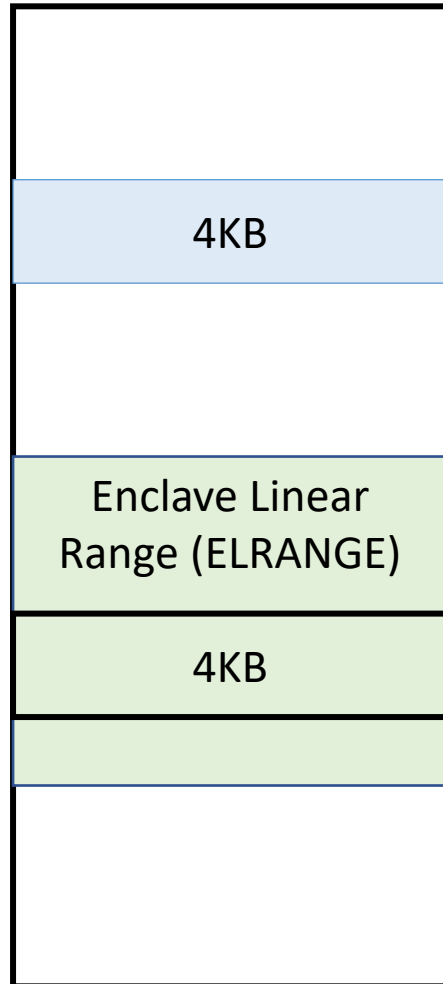
```

if (PA belongs to PRM) {
    compare VA in EPCM
    if (NOT match) {
        #Signal Fault
    }
}

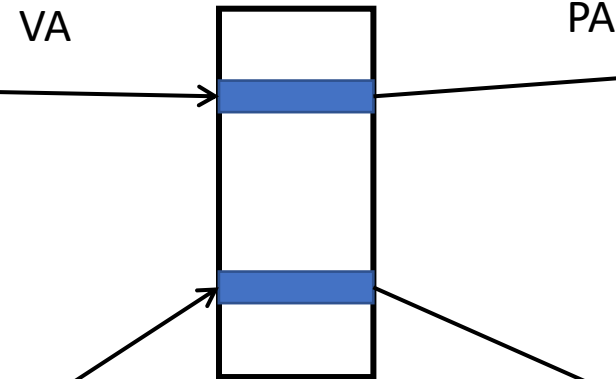
```

# Enclave Page Mapping Information

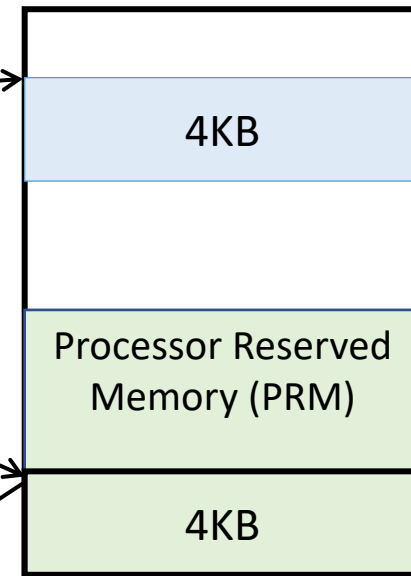
Virtual Address Space (Programmer's View)



Page Table per process



Physical Address Space (limited by DRAM size)



Enclave Page Cache Mapping (EPCM)

Stored in PRM



{PA, VA, Enclave ID}

Problem: pages are allocated and selected by system software.

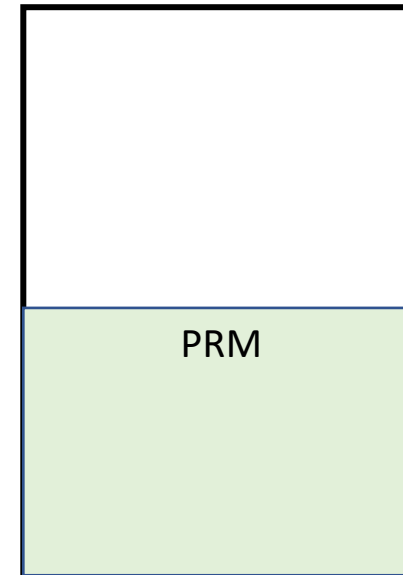
# So far .....

- Once the enclave is initialized correctly, it can be isolated from system software using
  - Hardware access control (supported by MMU)
  - Hardware support for secure context switch
- How to ensure the initialization is correct?
  - Software Attestation (similar to secure boot)

# Enclave Initialization

- BIOS setup PRM region

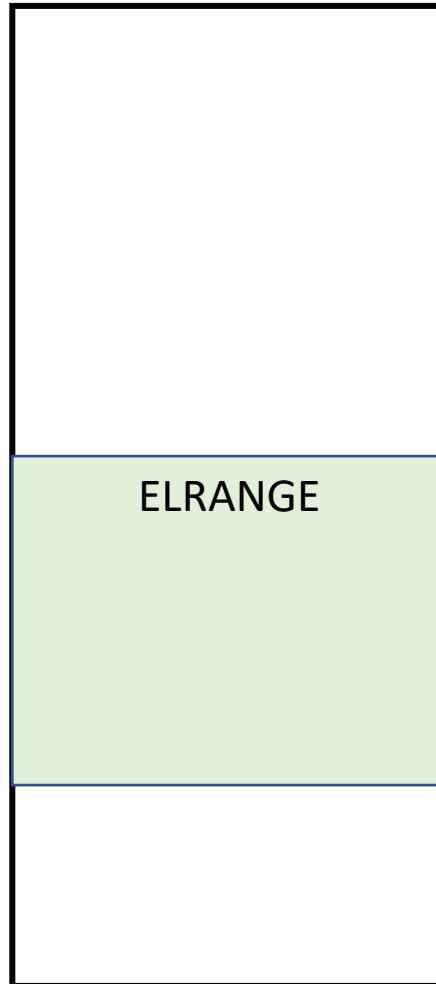
Physical Address Space



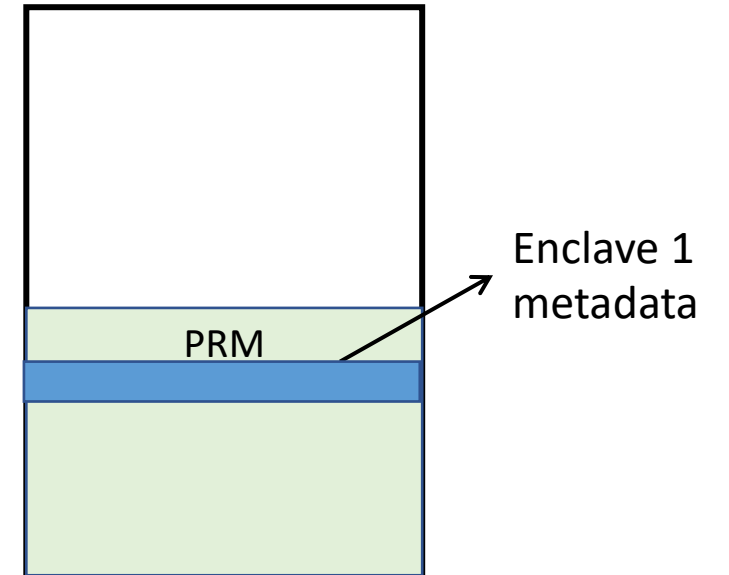
# Enclave Initialization

- Enclave creation (ECREATE)

Virtual Address Space



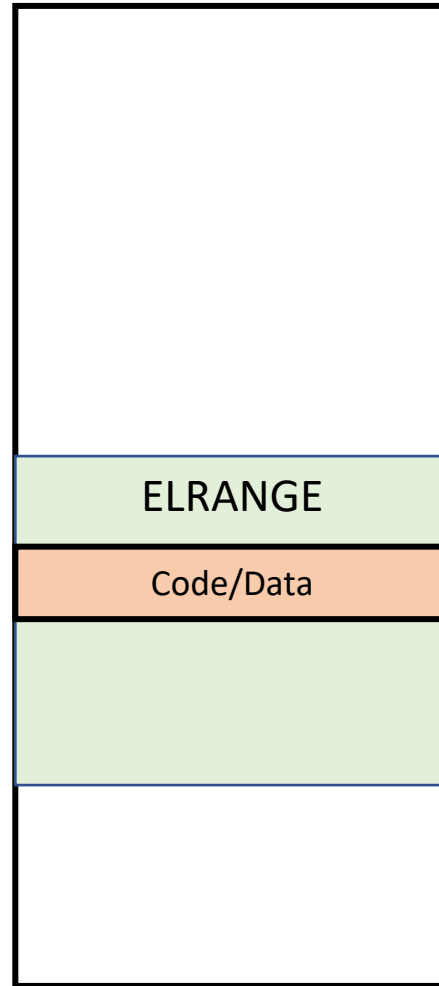
Physical Address Space



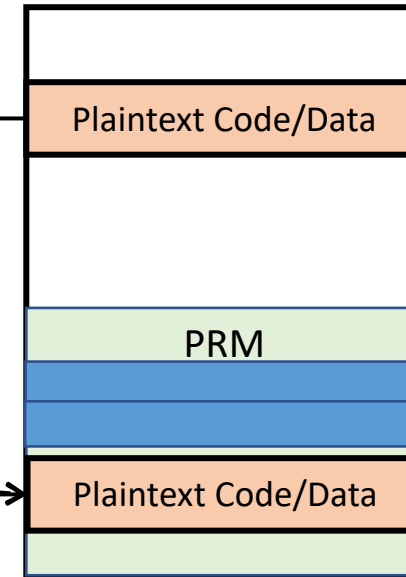
# Enclave Initialization

- Add page (EADD)
- Measure (EEXTEND)

Virtual Address Space



Physical Address Space



copy

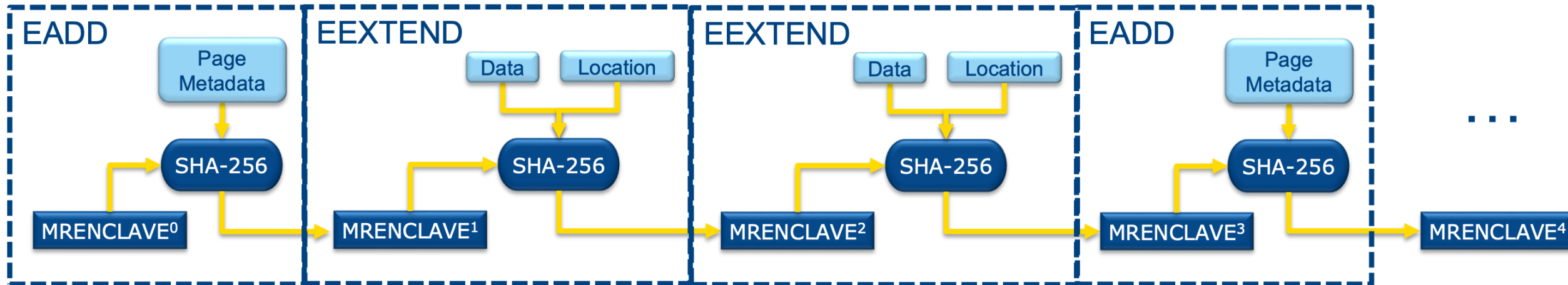
Enclave 1 metadata

Update mapping information in EPCM



# Enclave Measurement

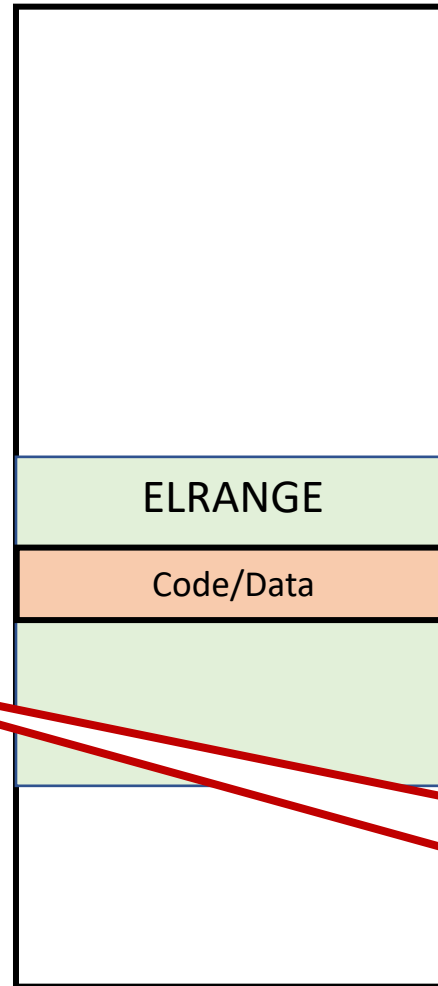
- Hardware generates a cryptographic log of the build process
  - Code, data, stack, and heap contents
  - Location of each page within the enclave
  - Security attributes (e.g., page permissions) and enclave capabilities
- Enclave identity (MRENCLAVE) is a 256-bit digest of the log that represents the enclave



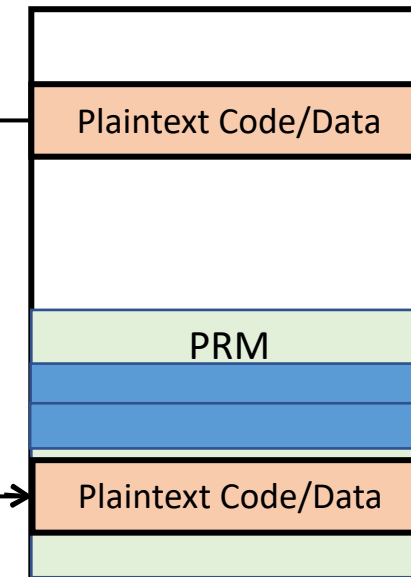
# Enclave Initialization

- Add page (EADD)
- Measure (EEXTEND)
- Init (EINIT)
  - Finalize measurement
- Active (EENTER)
  - Switch to enclave mode

Virtual Address Space



Physical Address Space



copy

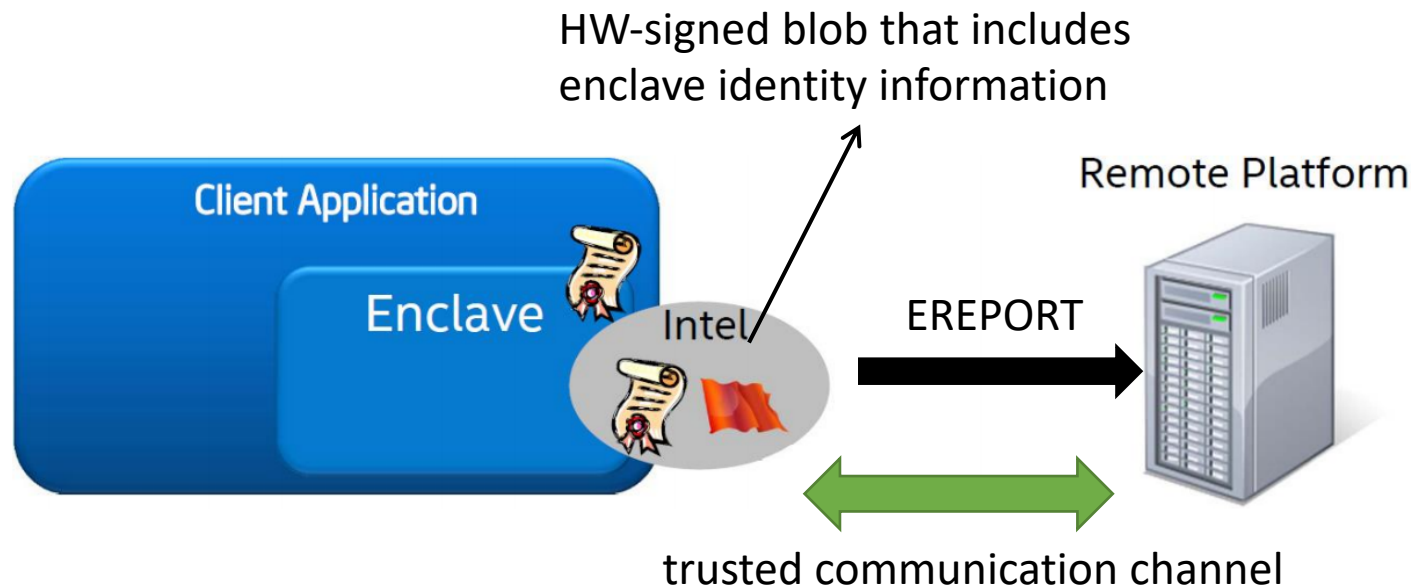
Enclave 1 metadata

Update mapping information in EPCM

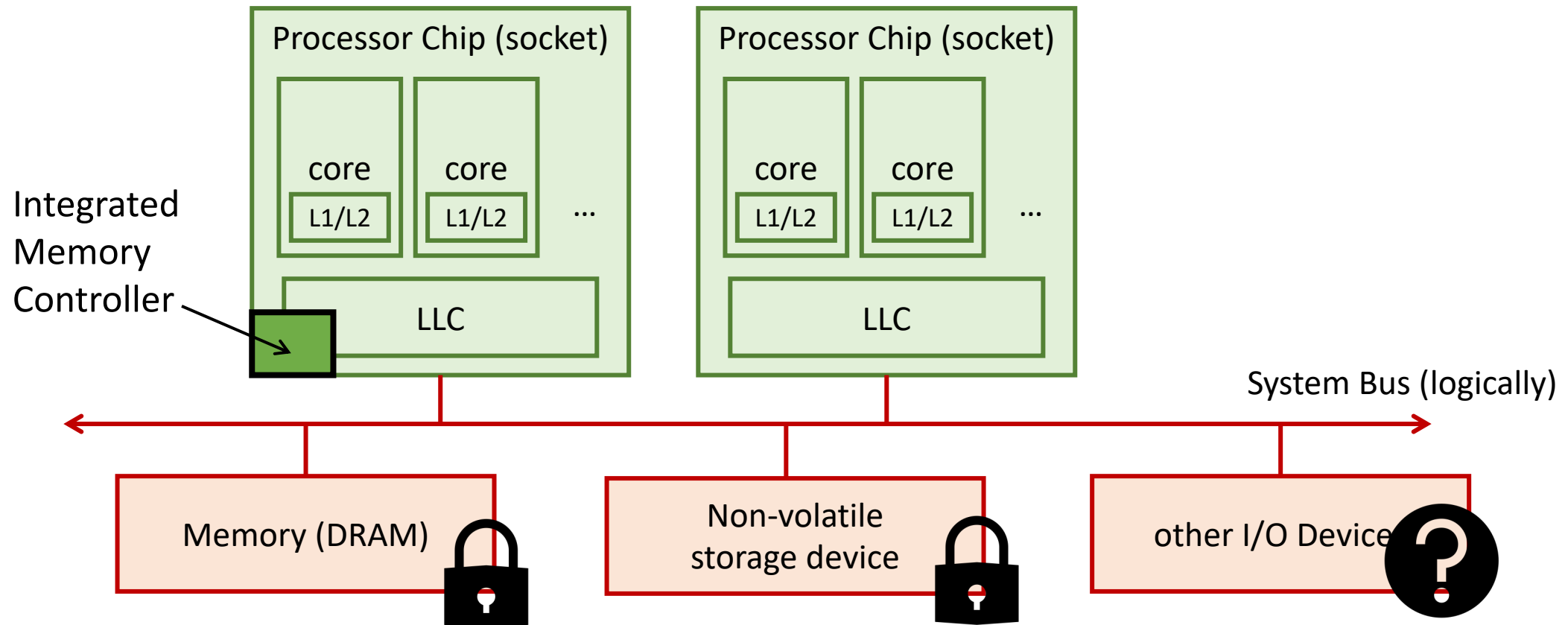
Problem:  
No measurement after EINIT

# Enclave Attestation and Sealing

- HW based attestation provides evidence that “this is the right application executing on an authentic platform” (approach similar to secure boot attestation)

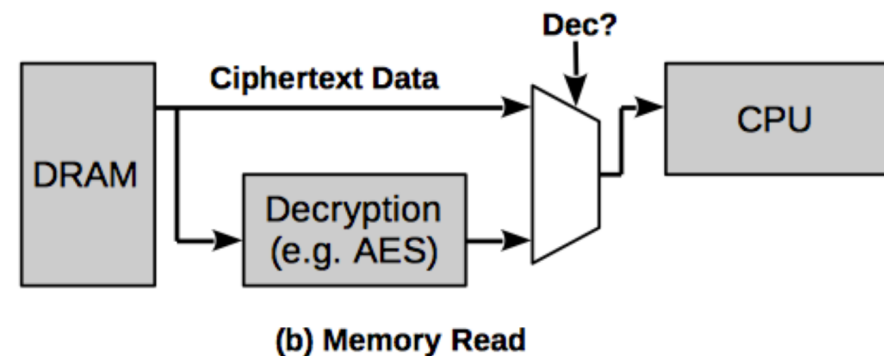
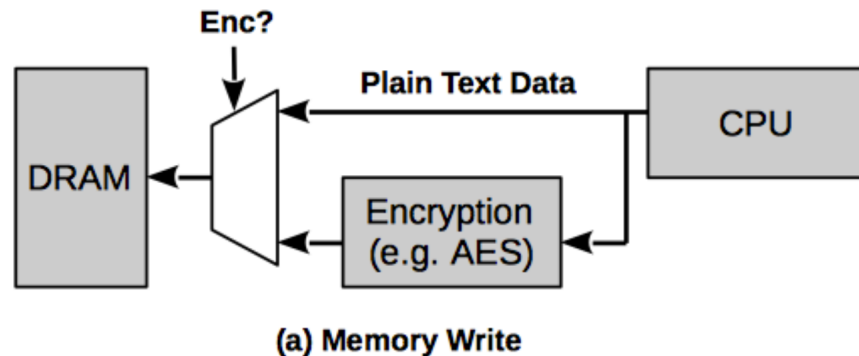


# Protect Memory



# Confidentiality Protection with Encryption

- Secret key is stored inside chip
- For freshness, encrypt with nonce (counter)
- {nonce, ciphertext} per cache block are stored externally in DRAM



# Integrity Protection with Hash

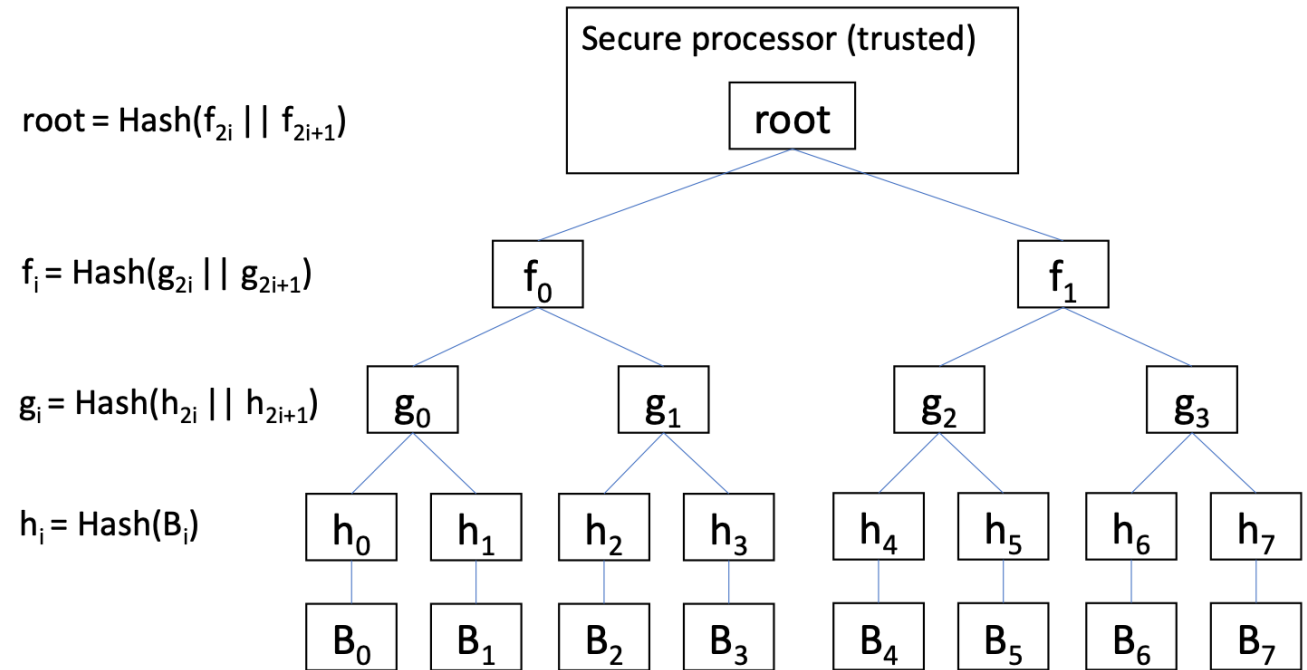
- For each cache line: {ciphertext + nonce + hash}
- Problem:
  - Need to store hashes or nonces on-chip → high on-chip storage requirement
  - Too much storage requirement (~64bits / block) → high off-chip storage requirement
- General solution:
  - Integrity Tree (Merkle tree)



Otherwise?

# Operations on Merkle Tree

- Only need to store the root node on chip
- How to verify block B1?
- Write to block B3?



# Next Lecture:

# Side Channel Introduction