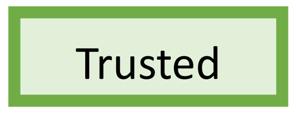
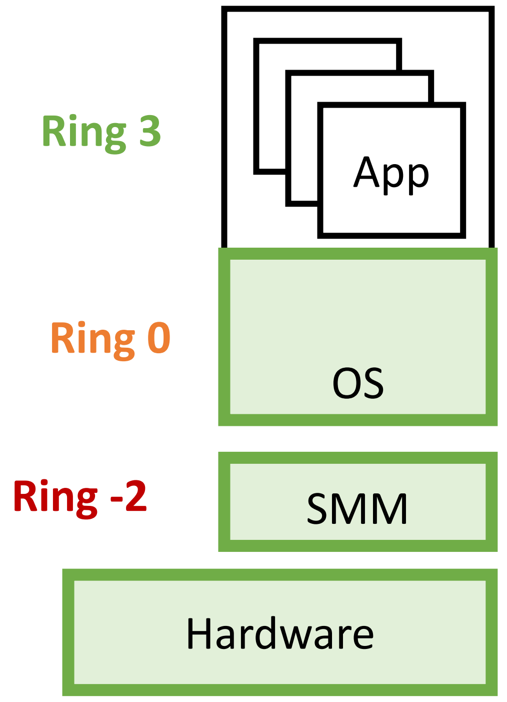
Hardware-supported Trusted Execution Environment (TEE)

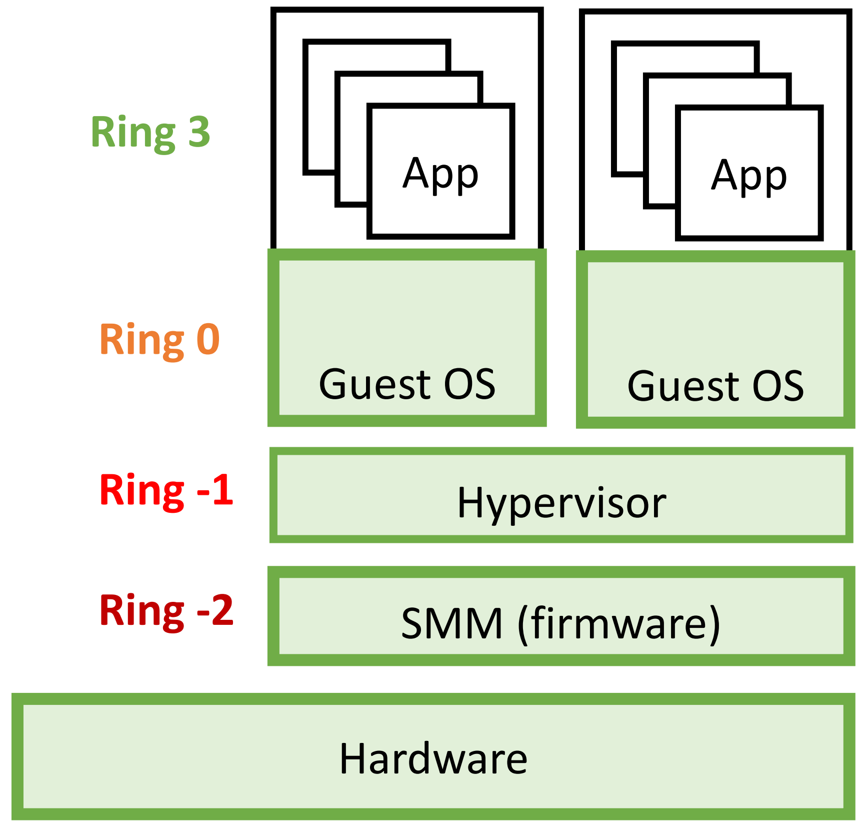
Mengjia Yan

Spring 2023

Trusted Computing Base (TCB)



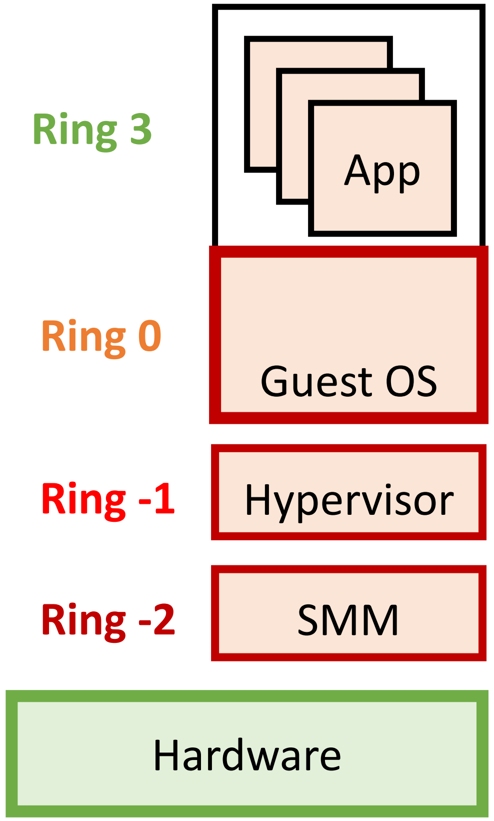




Shrink TCB. Why?

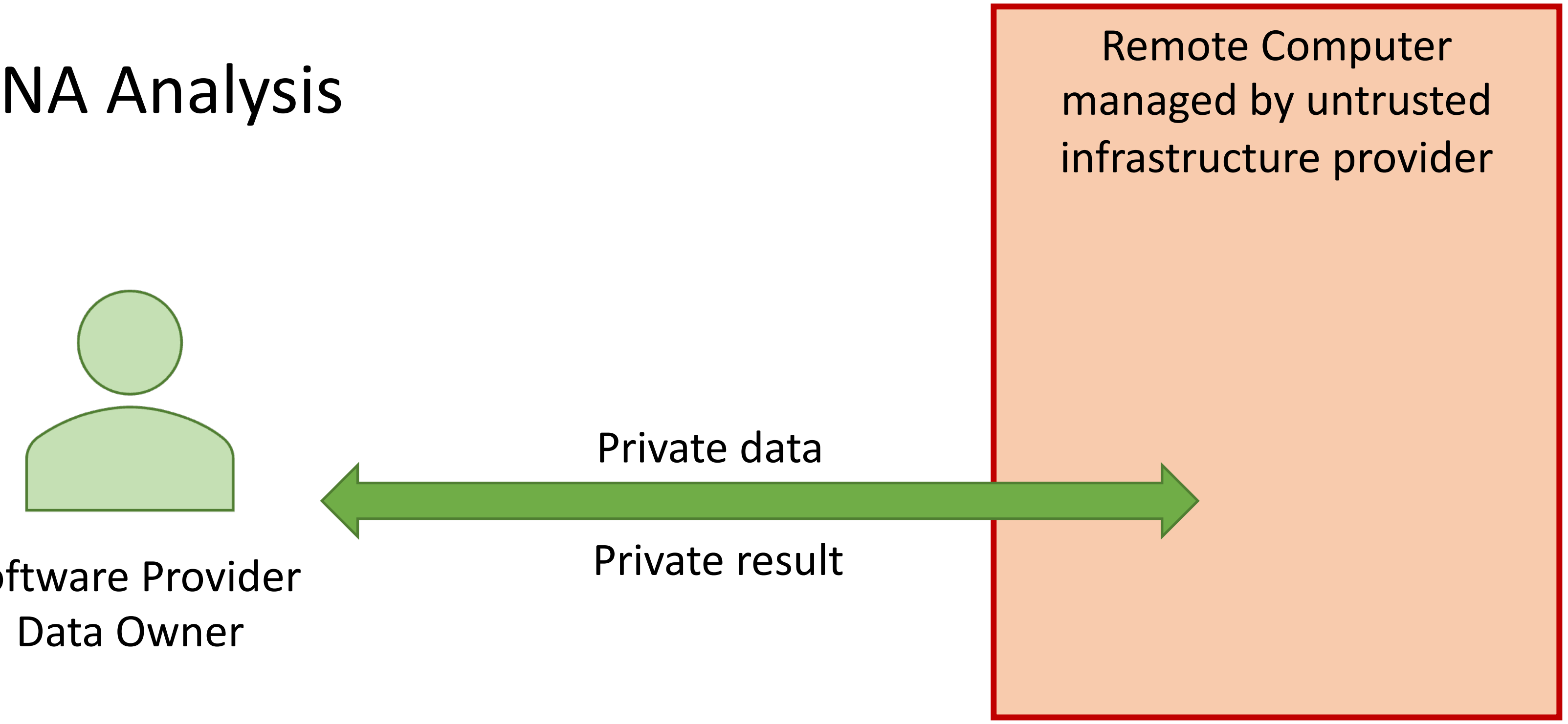
- Software bugs
  - SMM-based rootkits
  - Xen 150K LOC, 40+ vulnerabilities per year
  - Monolithic kernel, e.g., Linux, 17M LOC, 100+ vulnerabilities per year

- Remote Computing
  - Remote computer and software stack owned by an untrusted party



Secure Remote Computing

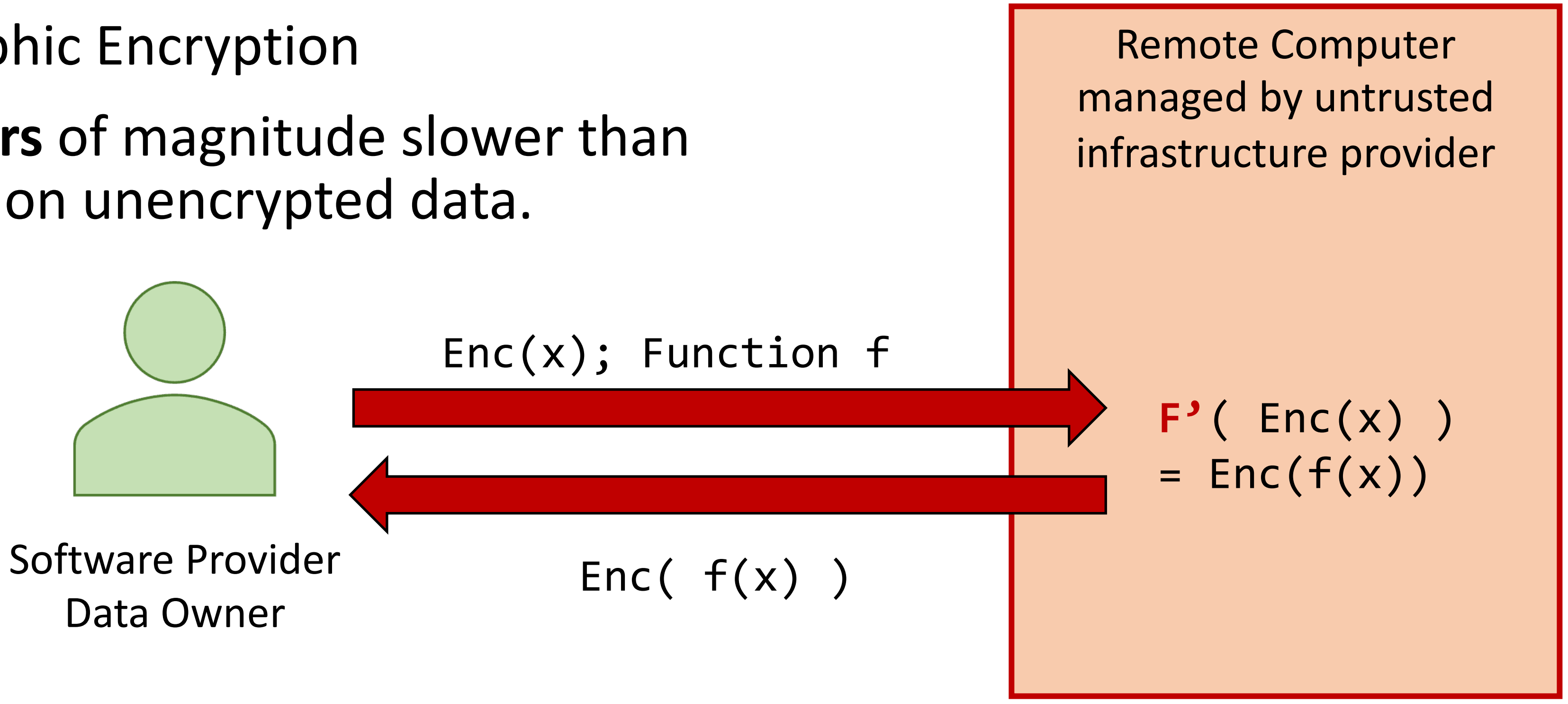
- Example: DNA Analysis


Remote Computer managed by untrusted infrastructure provider
Private data
Private result
Software Provider
Data Owner

How to keep my data private without trusting the host OS/hypervisor/SMM?

Potential Solutions

- Homomorphic Encryption
- **4 to 5 orders** of magnitude slower than computing on unencrypted data.

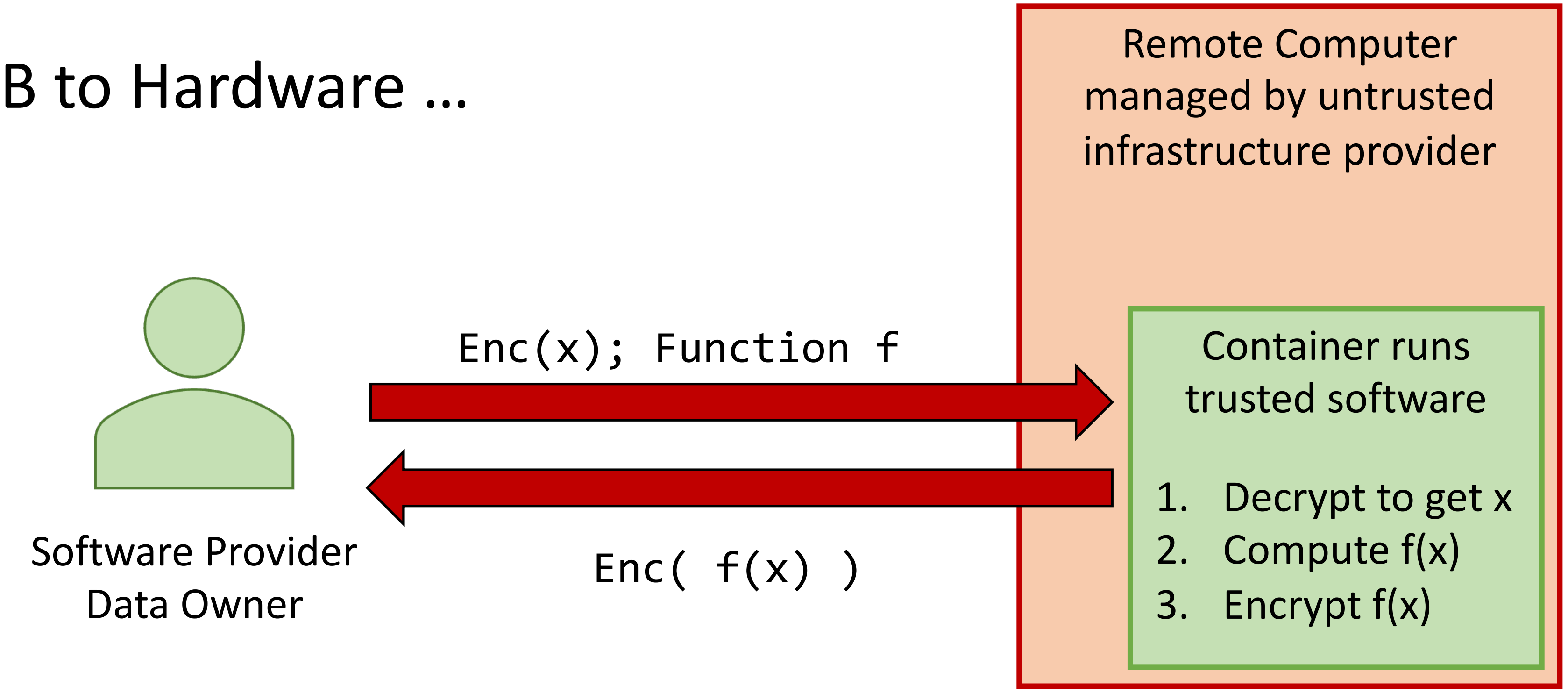


- Performance? Accelerators?

e.g., F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption; Axel Feldmann, Nikola Samardzic et al. MICRO'21

Potential Solutions

- Move TCB to Hardware …



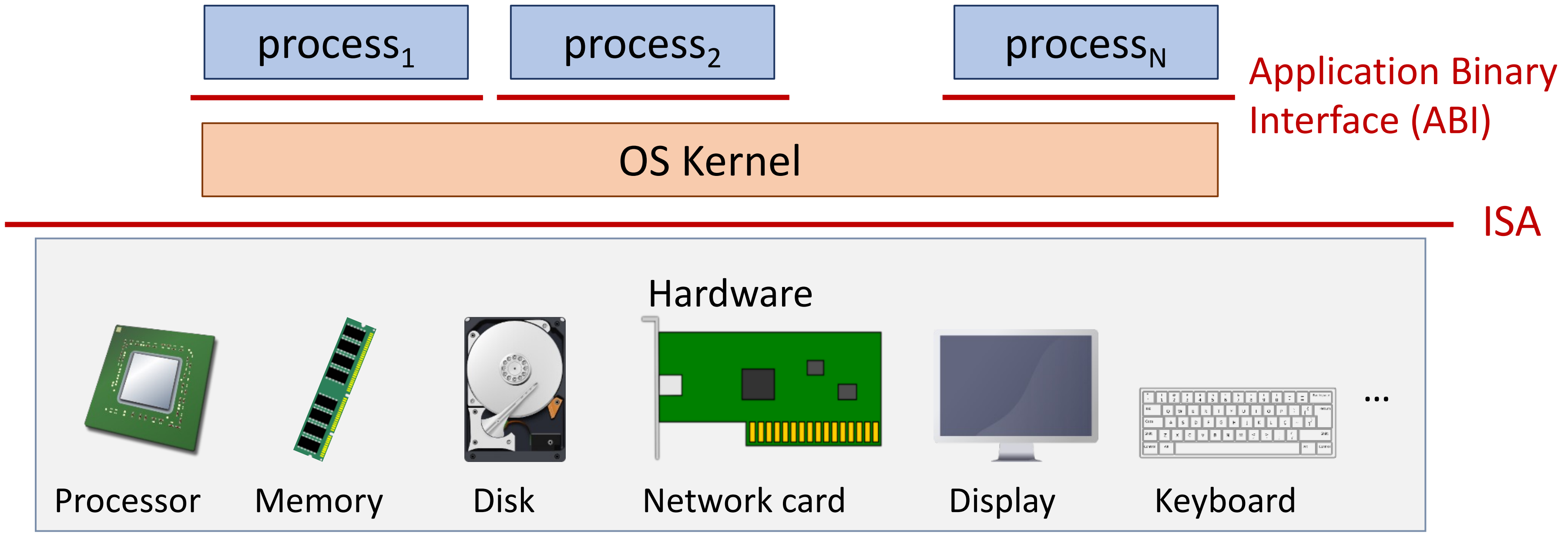
Outline

- Understand the threat model: privilege SW attacks

- Understand how to mitigate these threats

Privilege Software Attacks

Operating Systems



Launch Time

```
./helloworld
```

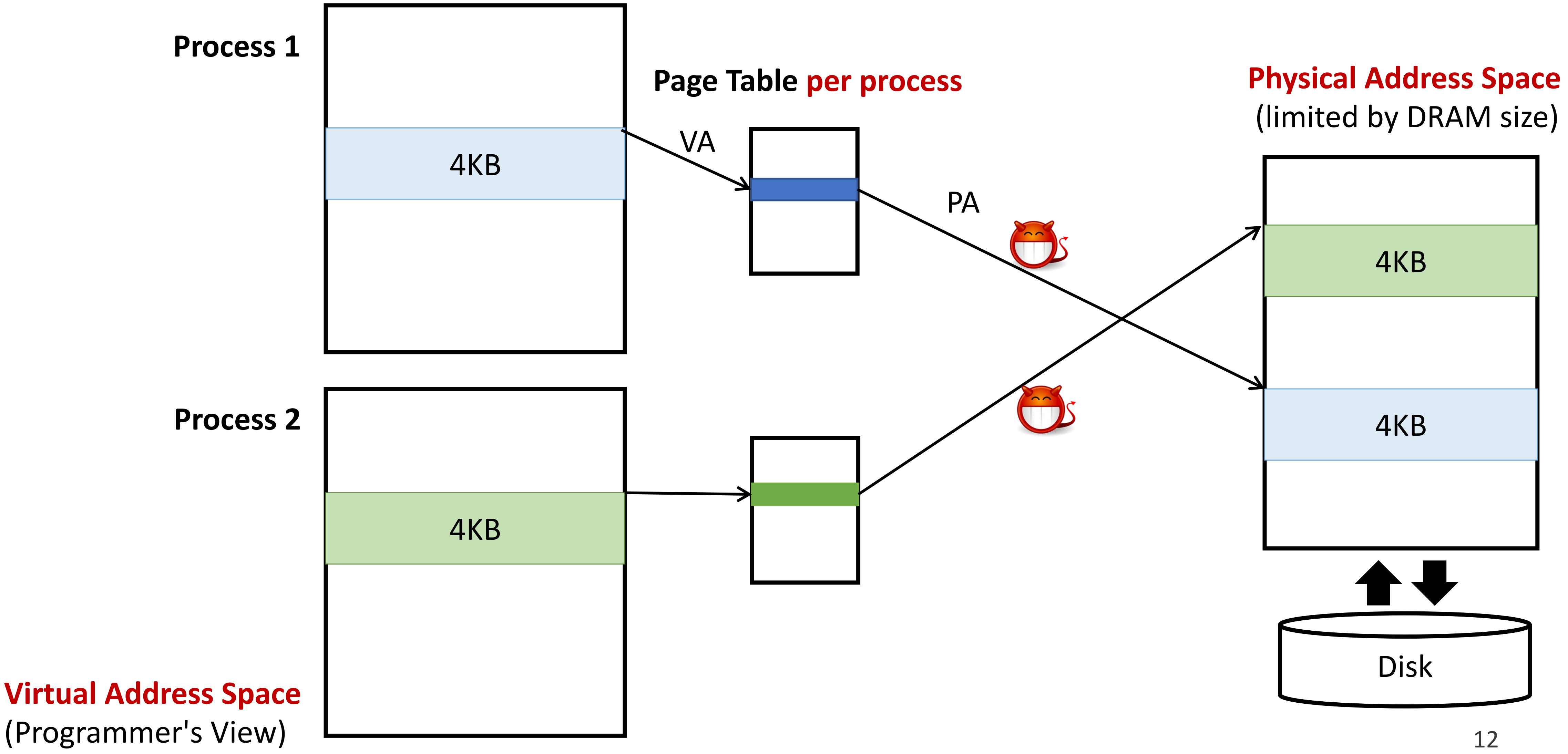
- Operations at launch time:
  - Create a process (PID, status, etc.)
  - Create a virtual address space: allocate memory for stack, heap, code region, set up the page tables
  - Setup file descriptor for input and output
  - Load the binary into the code region, and linked library if needed
  - Transfer the control to user space

CPU Abstraction

- Expose to users thread, rather than physical cores
- Achieve via context switch and interrupt handling

- Switch from user space to kernel space
  - Remember the current PC
  - Jump to kernel code: perform a sequence of save operations
    - Save general purpose registers content into an object associated with the current thread
    - Save system registers, including page table root address (CR3 in X86)
  - Based on the interrupt type, decide what to do
- Switch back to user space
  - Restore all the registers: general-purpose + system registers
  - Jump back to the saved PC

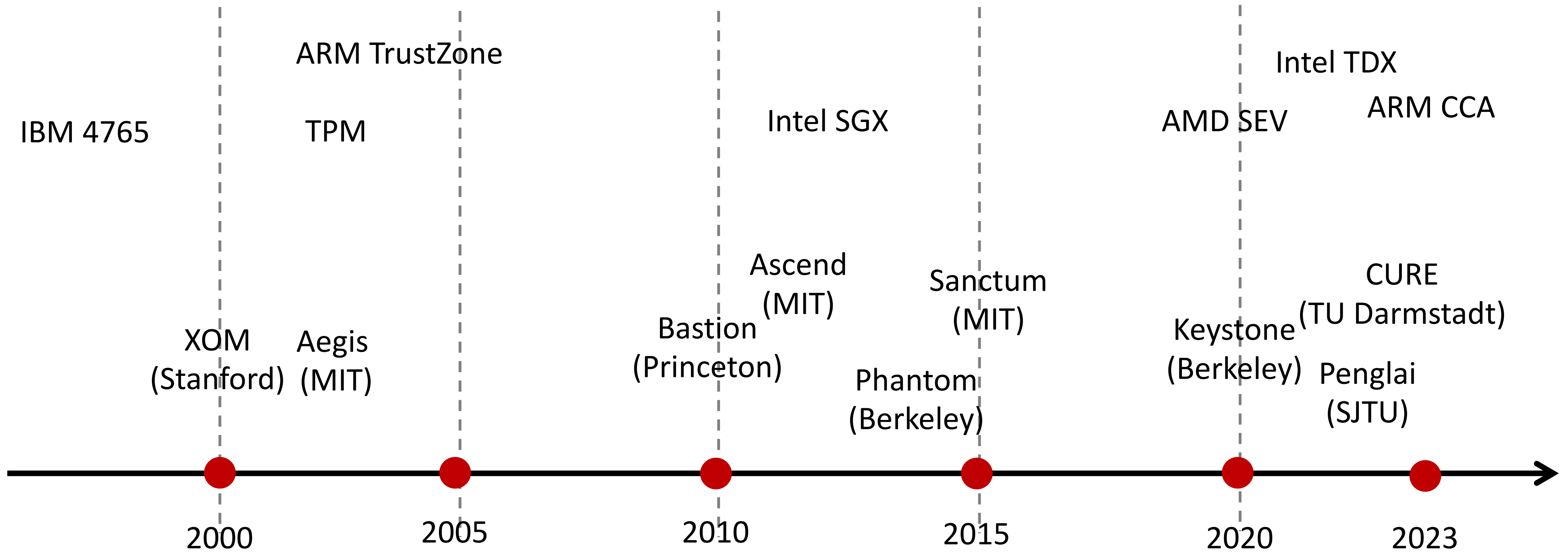
Virtual Memory Abstraction



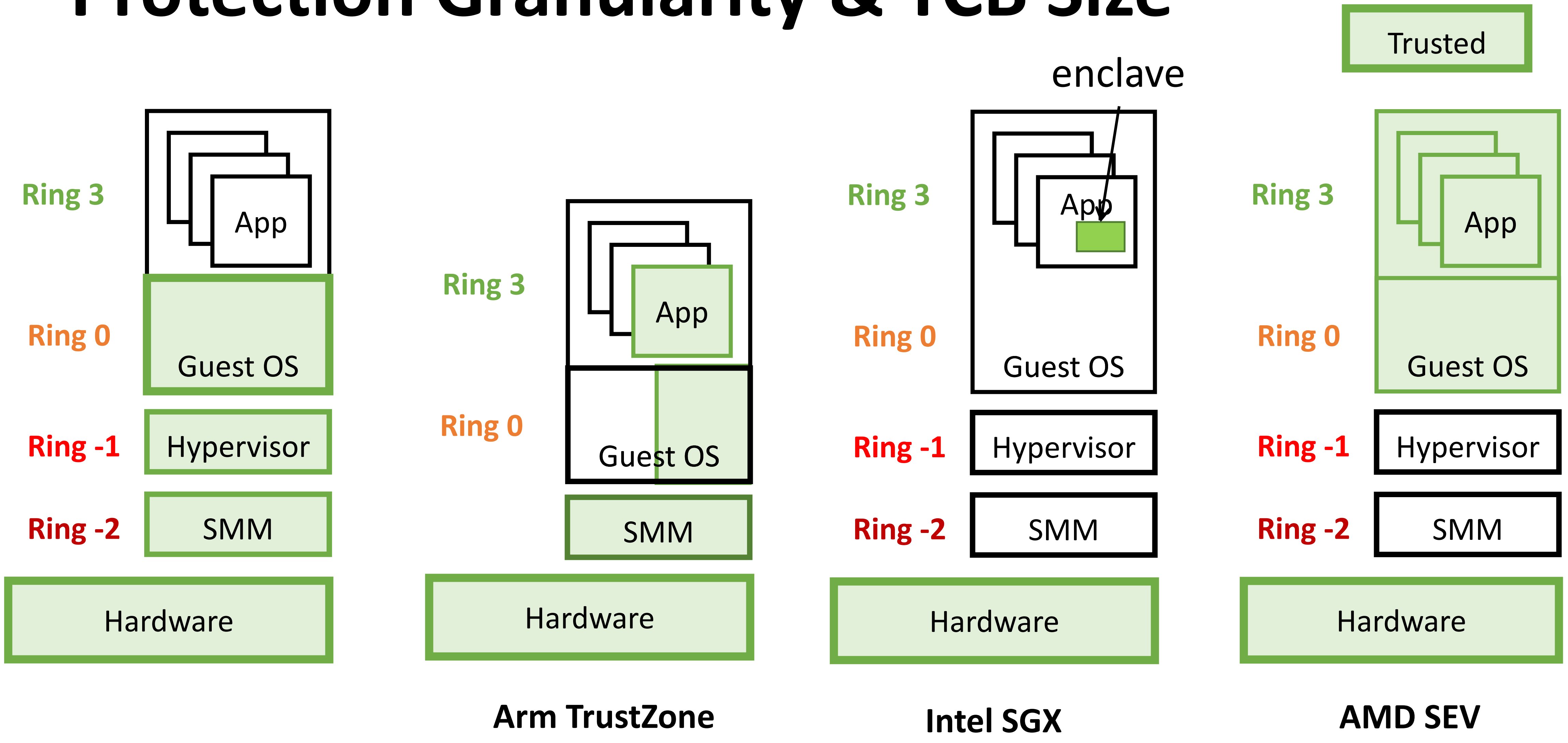
What can a privilege software attacker do?

- A non-comprehensive list
  - Modify the code to be executed
  - Monitor the whole execution process and data in register and in memory
  - Modify data in register and memory
  - Intercept IO, eavesdrop and tamper with the communication
  - ……

TEE Examples

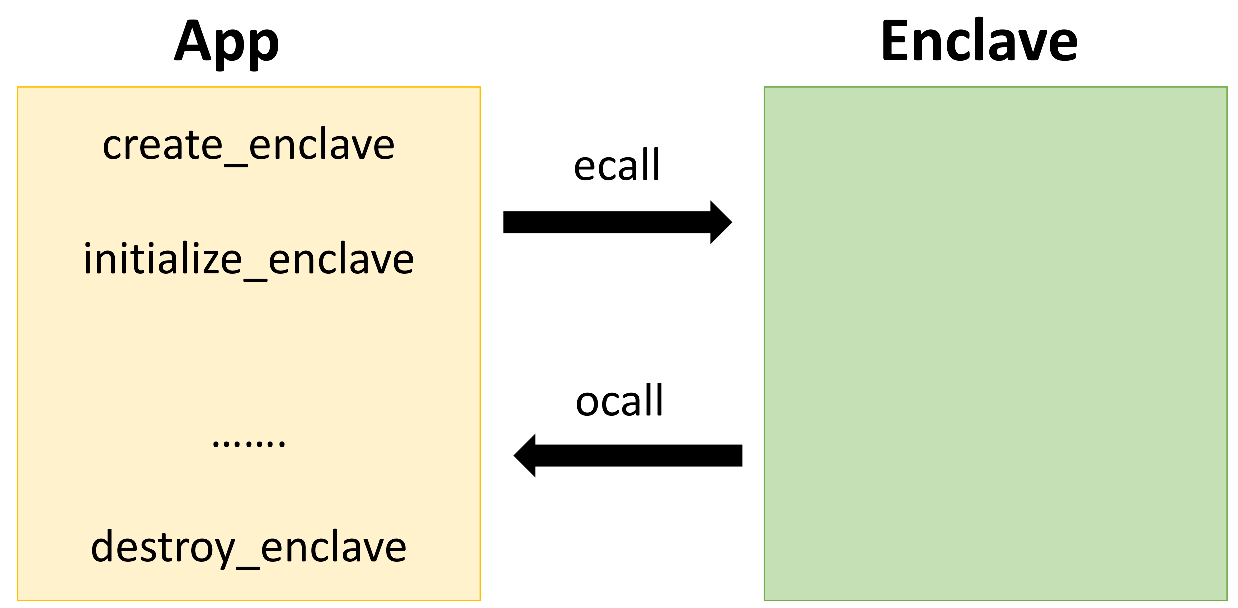


Protection Granularity & TCB Size



SGX Enclave Programming Model

- Examples from: https://github.com/intel/linux-sgx



Security Tasks

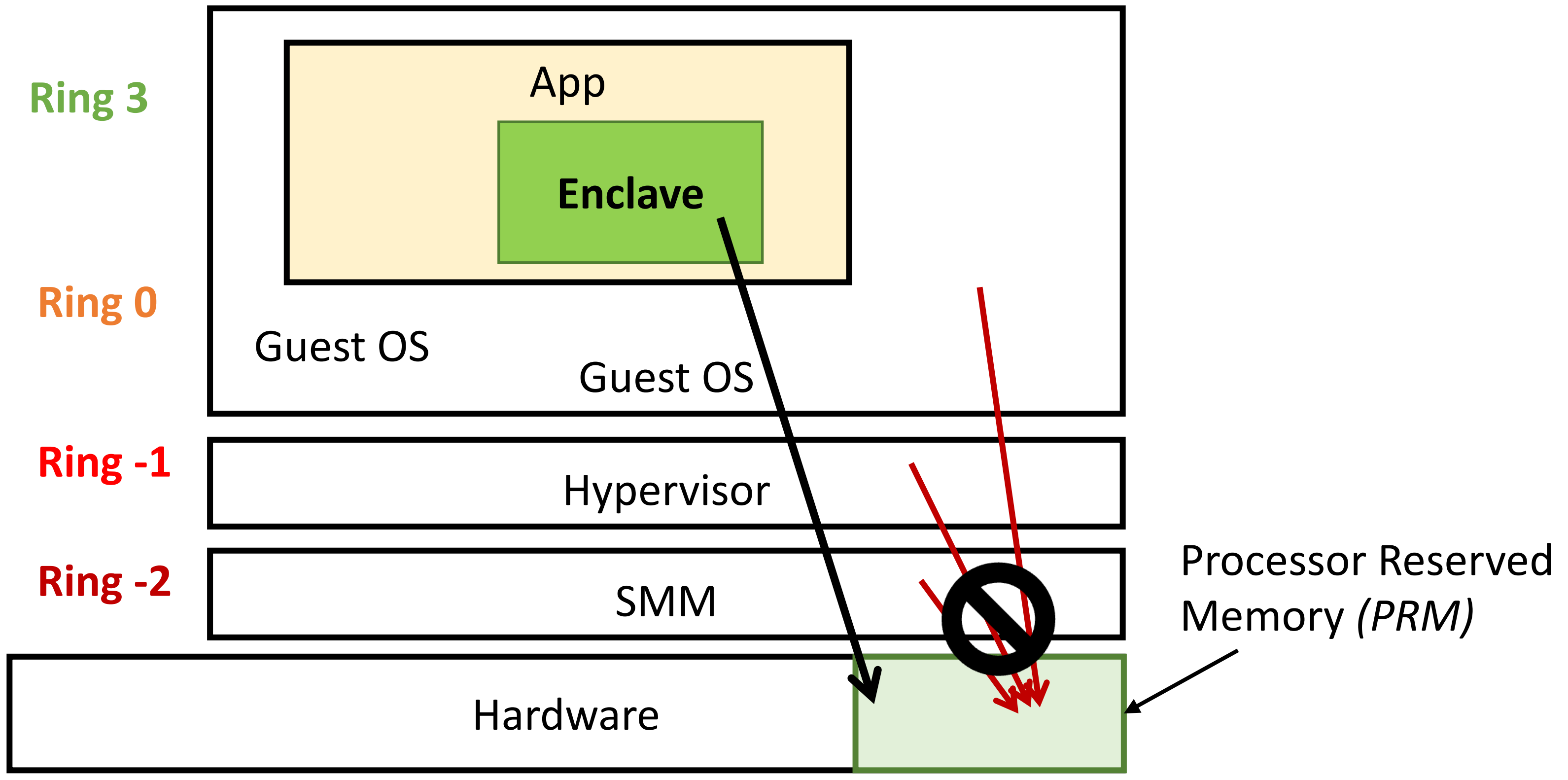
- How do we ensure the runtime execution follows our expectation (confidentiality and integrity of the execution)?

- How do we ensure the enclave code is the code that we want to execute? (code integrity during initialization)

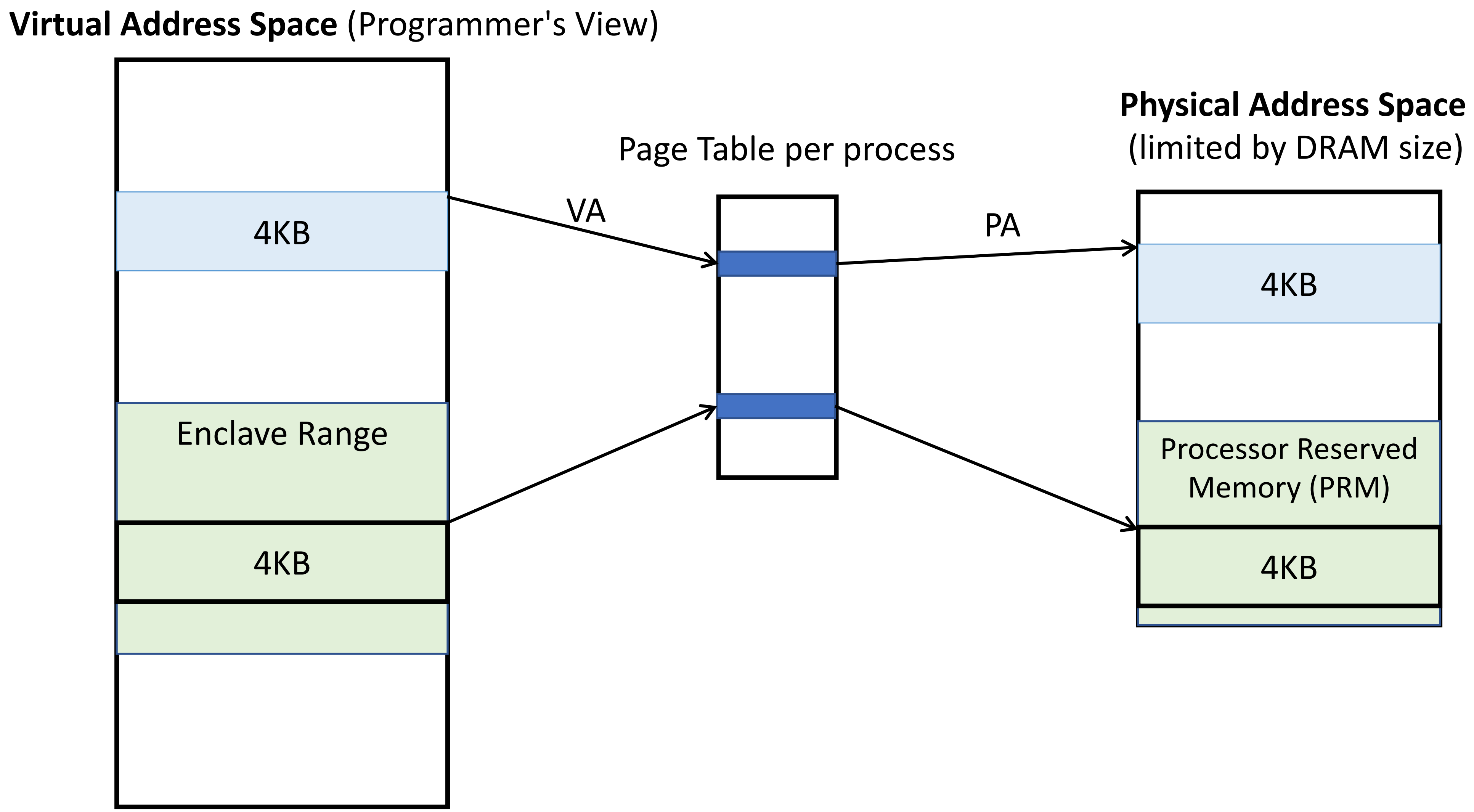
- DRAM security? How to deal with Rowhammer and Coldboot attacks? (physical attacks. Will cover if time permitted)

Intel SGX Overview

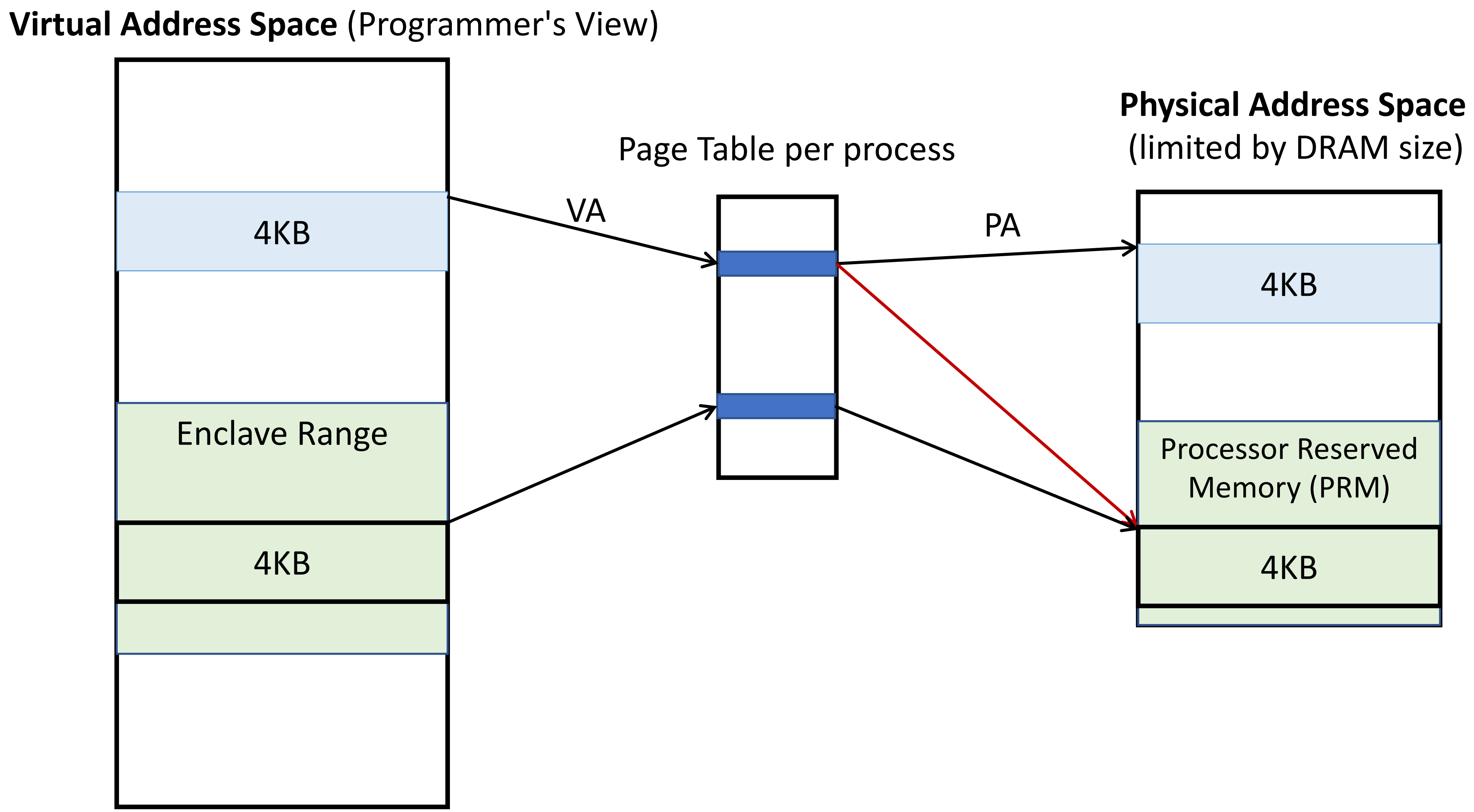
- Enclave code/data map to PRM; Different enclaves access their own memory region



Intel SGX Address Translation Overview

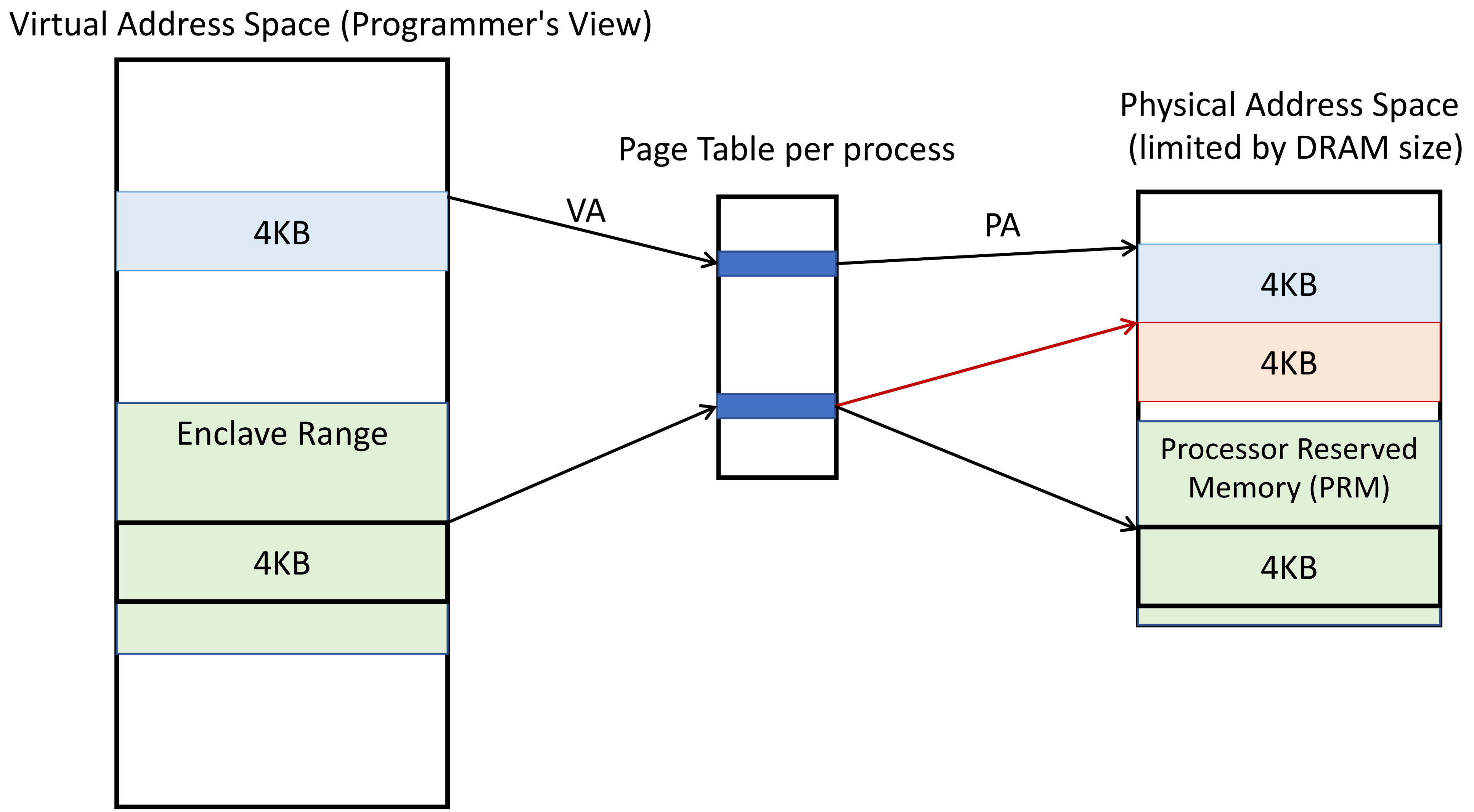

Virtual Address Space (Programmer's View)
4KB
Enclave Range
4KB
VA
Page Table per process
PA
Physical Address Space
(limited by DRAM size)
4KB
Processor Reserved Memory (PRM)
4KB

Malicious Address Translation #1

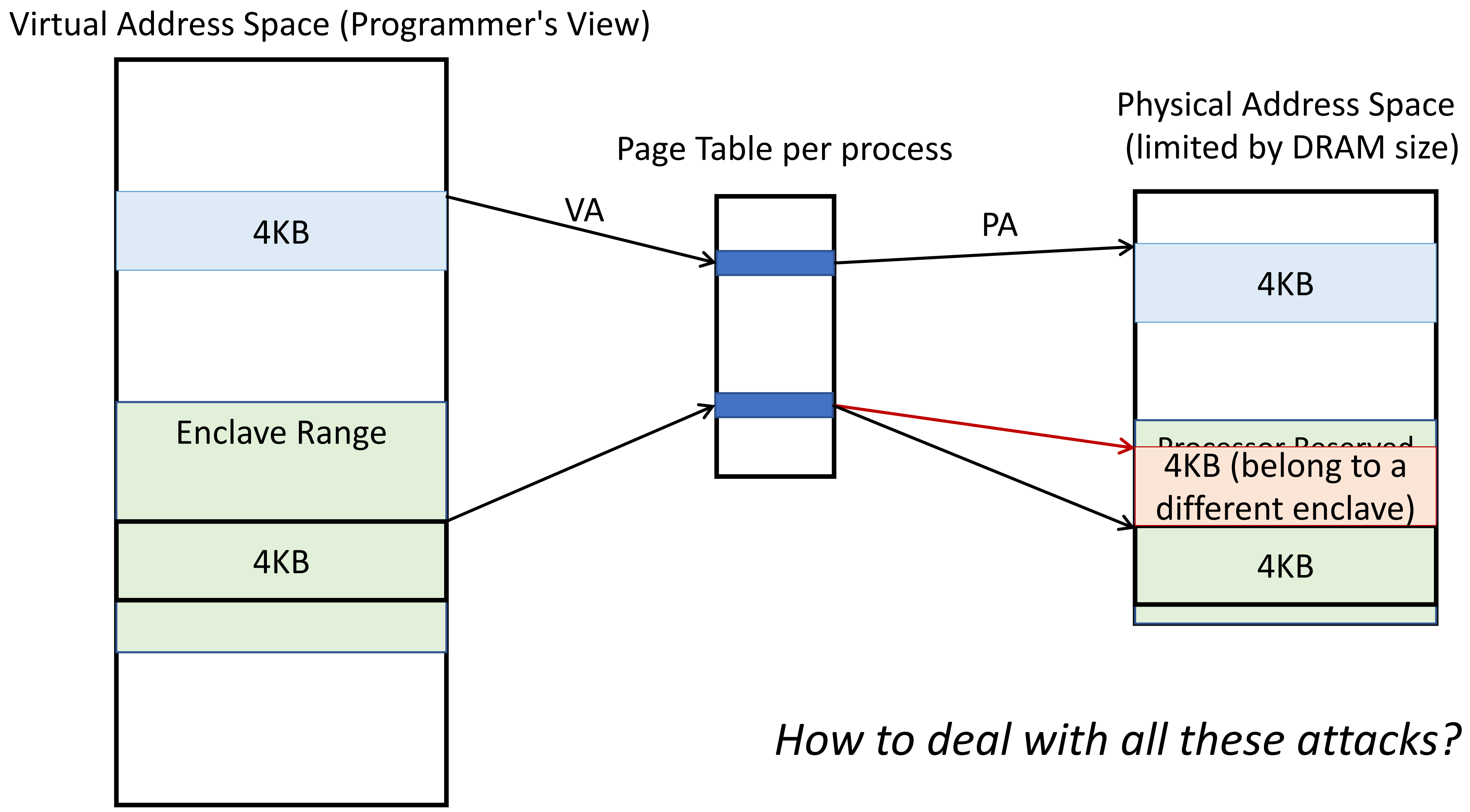


Virtual Address Space (Programmer's View)
4KB
Enclave Range
4KB
VA
Page Table per process
PA
Physical Address Space
(limited by DRAM size)
4KB
Processor Reserved Memory (PRM)
4KB

Malicious Address Translation #2

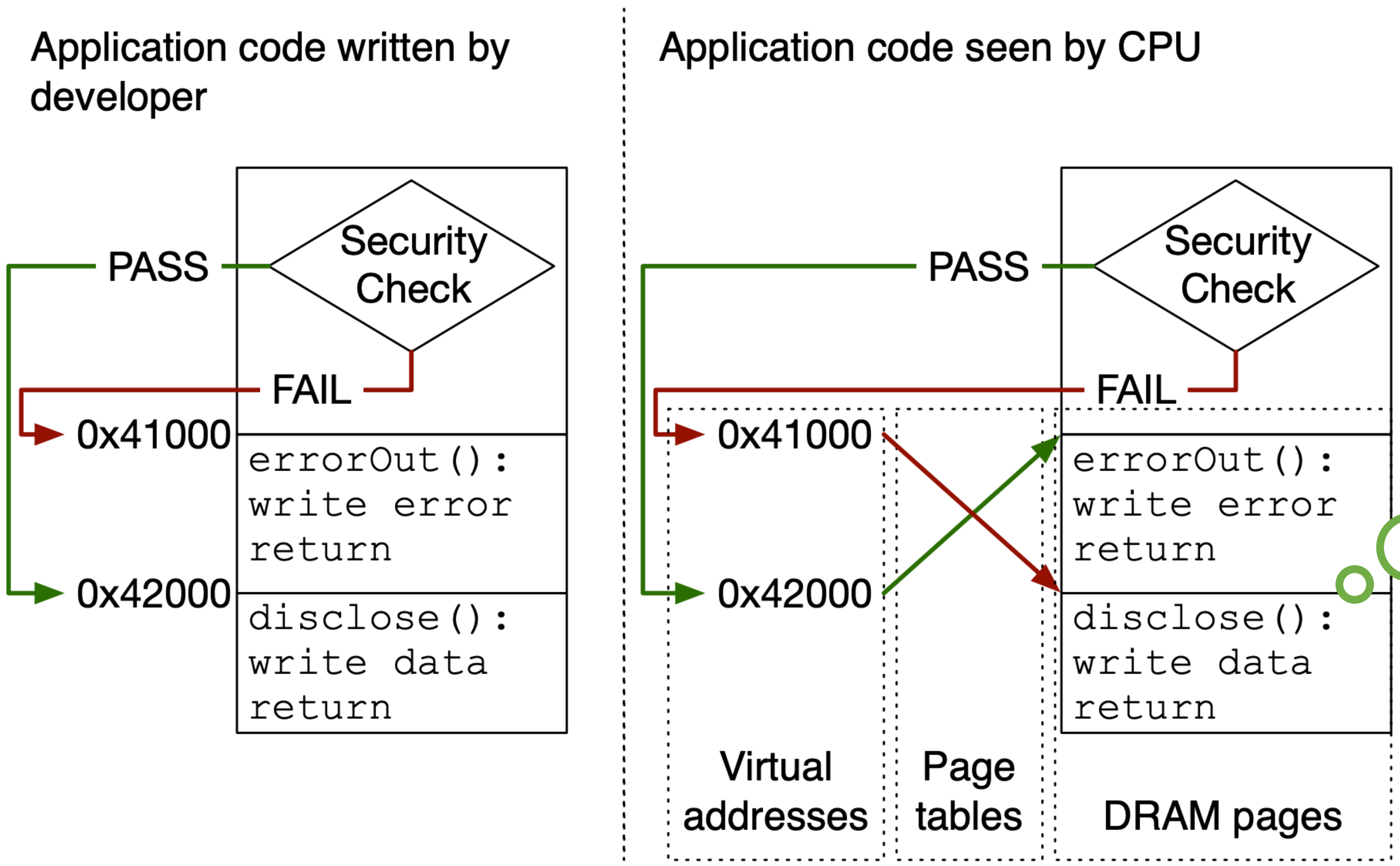

Virtual Address Space (Programmer's View)
4KB
Enclave Range
4KB
VA
Page Table per process
PA
Physical Address Space (limited by DRAM size)
4KB
4KB
Processor Reserved Memory (PRM)
4KB

Malicious Address Translation #3


Virtual Address Space (Programmer's View)
4KB
Enclave Range
4KB
VA
Page Table per process
PA
Physical Address Space (limited by DRAM size)
4KB
Processor Reserved
4KB (belong to a different enclave)
4KB

How to deal with all these attacks?

Malicious Address Translation #4



```
errorOut():
write error
return
disclose():
write data
return
```

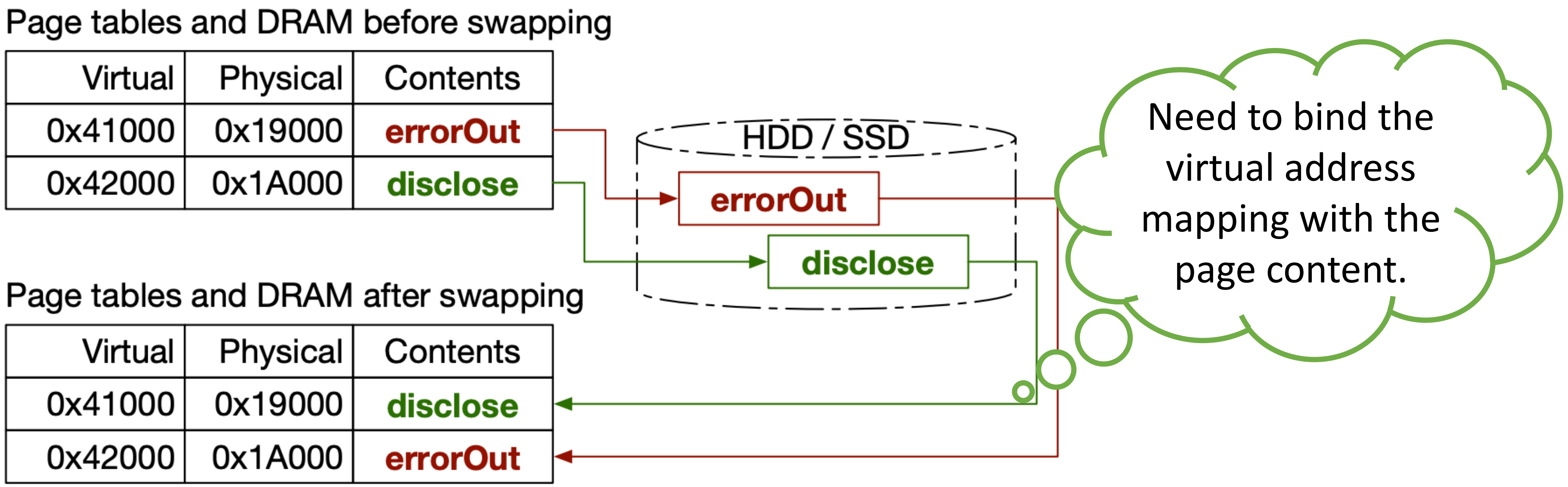
Need to keep track of the page table for enclaves by trusted hardware/software.

Solution: Inverted Page Table

- Check for security invariant:
  - Enclave VA, enclave mode → PRM
  - Non-enclave mode is not allowed access PRM using whitherever address
- For each page in the PRM, remember the mapping from
  <PPN> → <VPN, Enclave ID>
  Keep the reversed page table in PRM, so privilege software cannot modify
- When to perform the check? (Review address translation process)
  - After each address translation

Malicious Address Translation #5

A memory mapping attack that does not require modifying the page tables.



Page tables and DRAM before swapping

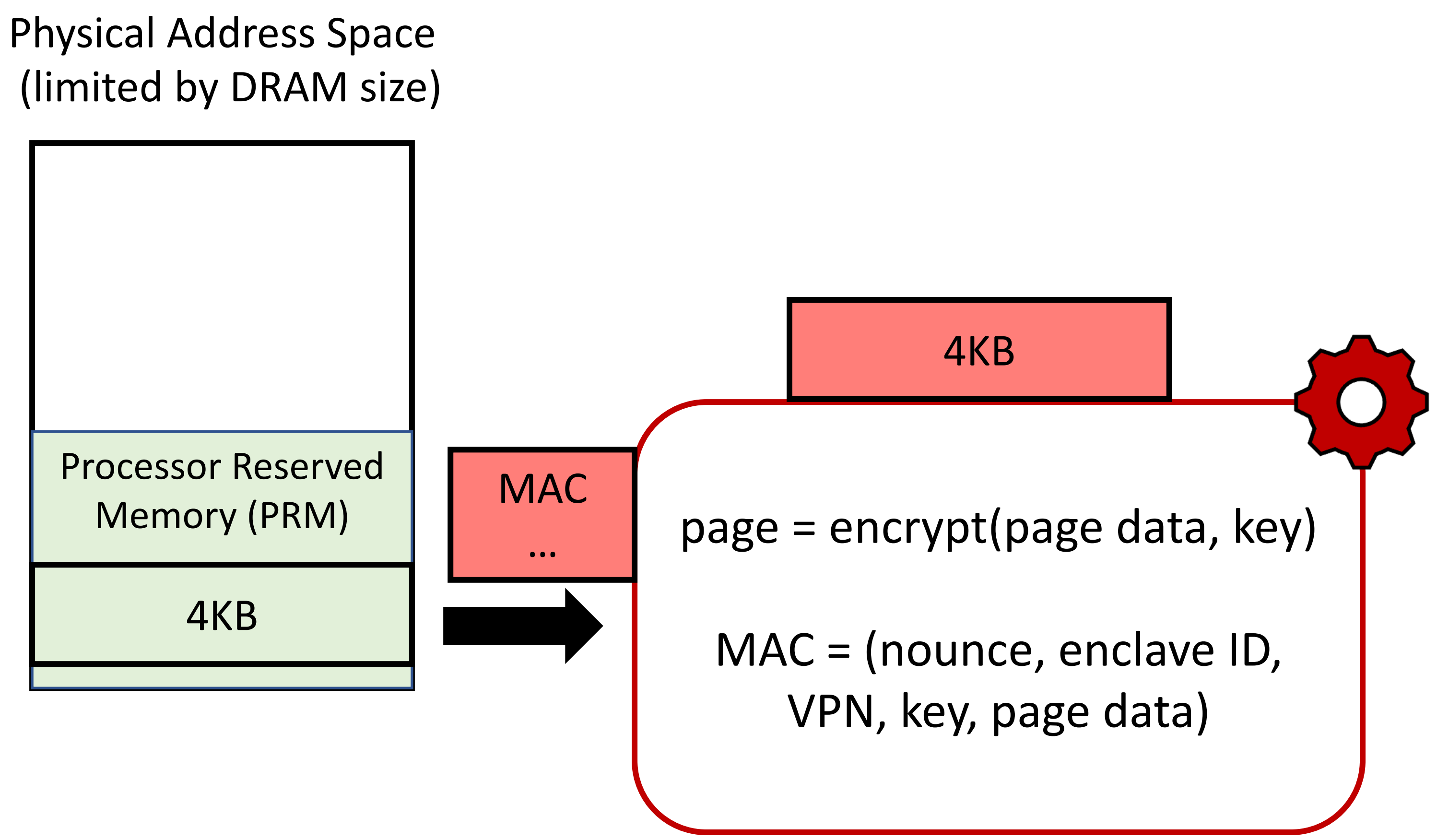
Virtual	Physical	Contents
0x41000	0x19000	errorOut
0x42000	0x1A000	disclose

Page tables and DRAM after swapping

Virtual	Physical	Contents
0x41000	0x19000	disclose
0x42000	0x1A000	errorOut

Need to bind the virtual address mapping with the page content.

Solution: Page Encryption and Authentication

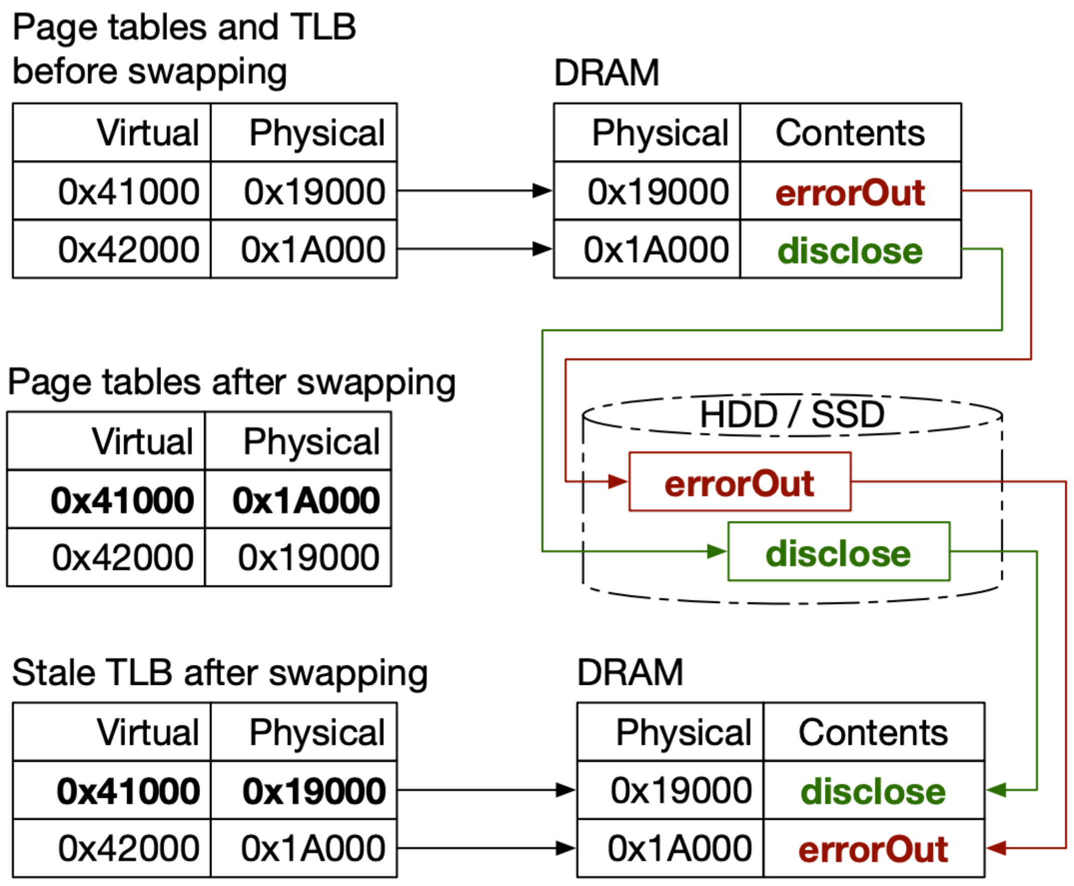


page = encrypt(page data, key)

MAC = (nounce, enclave ID, VPN, key, page data)

Malicious Address Translation #6

A memory mapping attack that exploits stable TLB entries.



Need to make sure TLB is never obsolete.

Solution: Keep TLB up-to-date

- Keep an extra state in the inverted page table
  - <PPN> → <VPN, Enclave ID>
  - <PPN, state> → <VPN, Enclave ID>
  - Mark “blocked”
  - Unset only until all the VPNs (can mapped by multiple enclaves) exist and flush TLBs

- If the TLB has stale data, post address translation check will see the physical address is “blocked”

Summary: SGX Memory Management

- #1: Maintain a inverted page table and check after every address translation

  Physical page in PRM -> (enclave ID, virtual page number)

- #2: Encrypt/decrypt upon page swap to non-PRM region

  (nounce, enclave ID, virtual page number, key, page content) → MAC

- #3: Keep TLB state up-to-date

  Upon page swap, block the page in the inverted page table and unblock only until all the corresponding TLB entries are flushed

Alternative Solutions

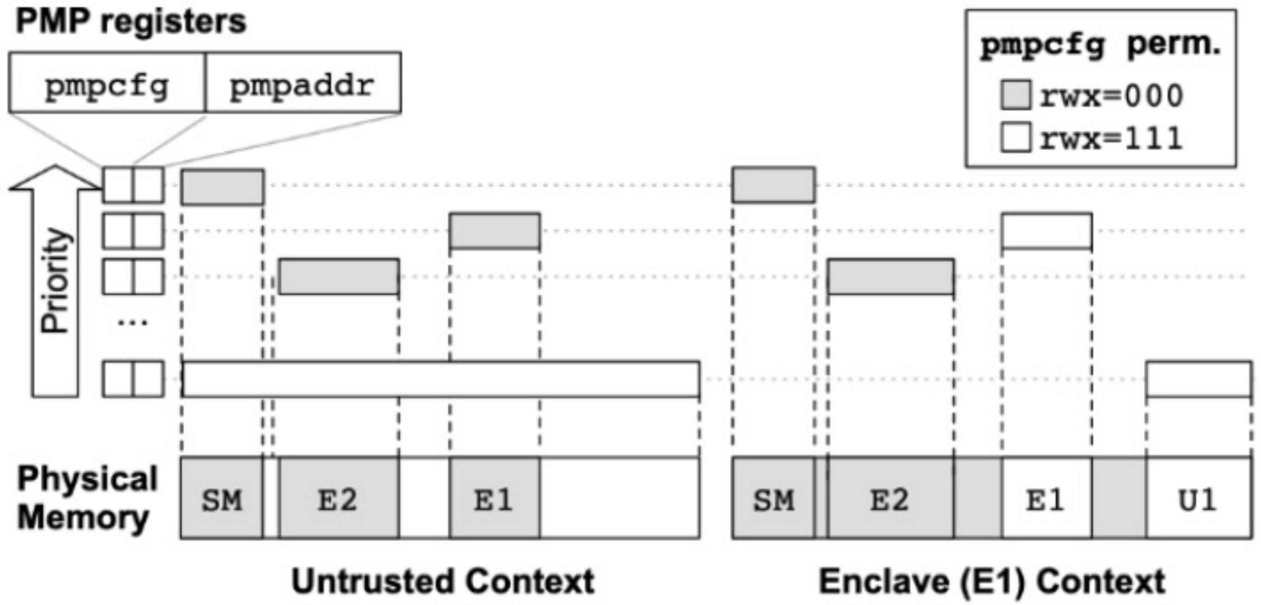
- Naïve idea:
  - Let the trusted component handle page management
  - Problem: Large TLB

- Keystone’s approach: leverage RISC-V’s PMP registers
  - Enforce coarse-grained isolation and let the application to manage their page mappings

- AMD SEV:
  - Rely on encryption. But symmetric key vulnerability
  - Recent version introduces reverse page table

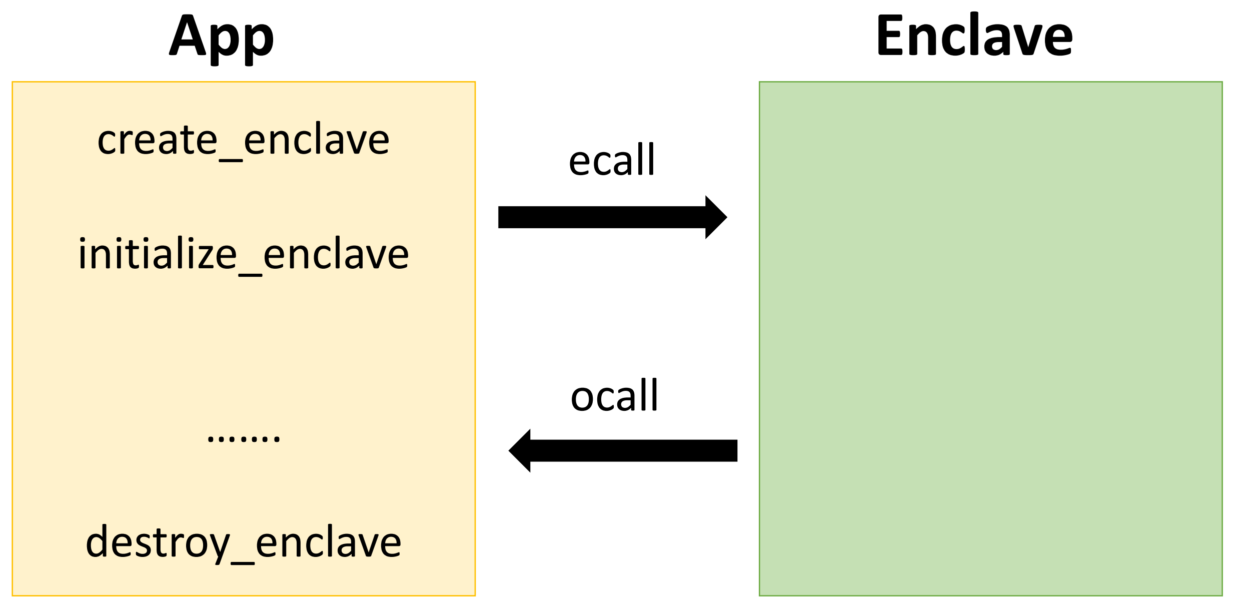
Keystone's Solution

- PMP check after every page translation to avoid cross-domain access
- Enclave application uses a runtime to support self-managed mapping.



Review: SGX Enclave Programming Model

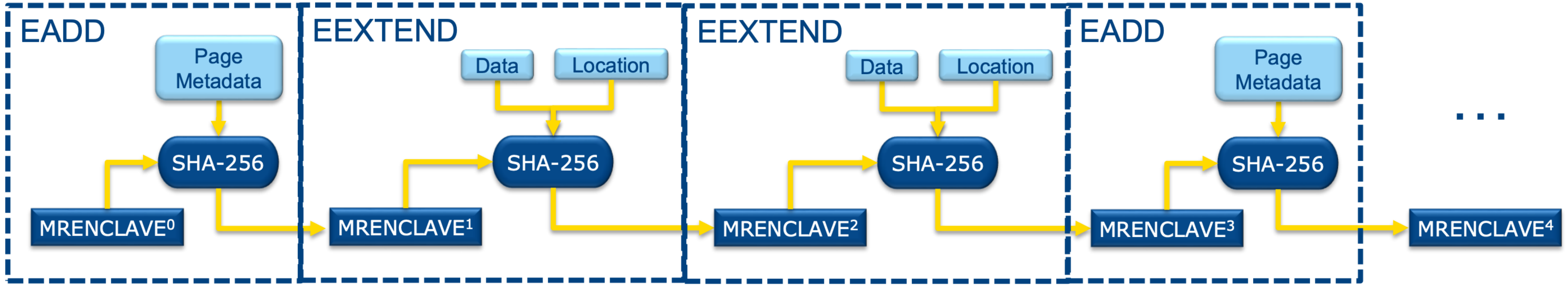
- How to ensure the enclave is initialized correctly?



Enclave Measurement

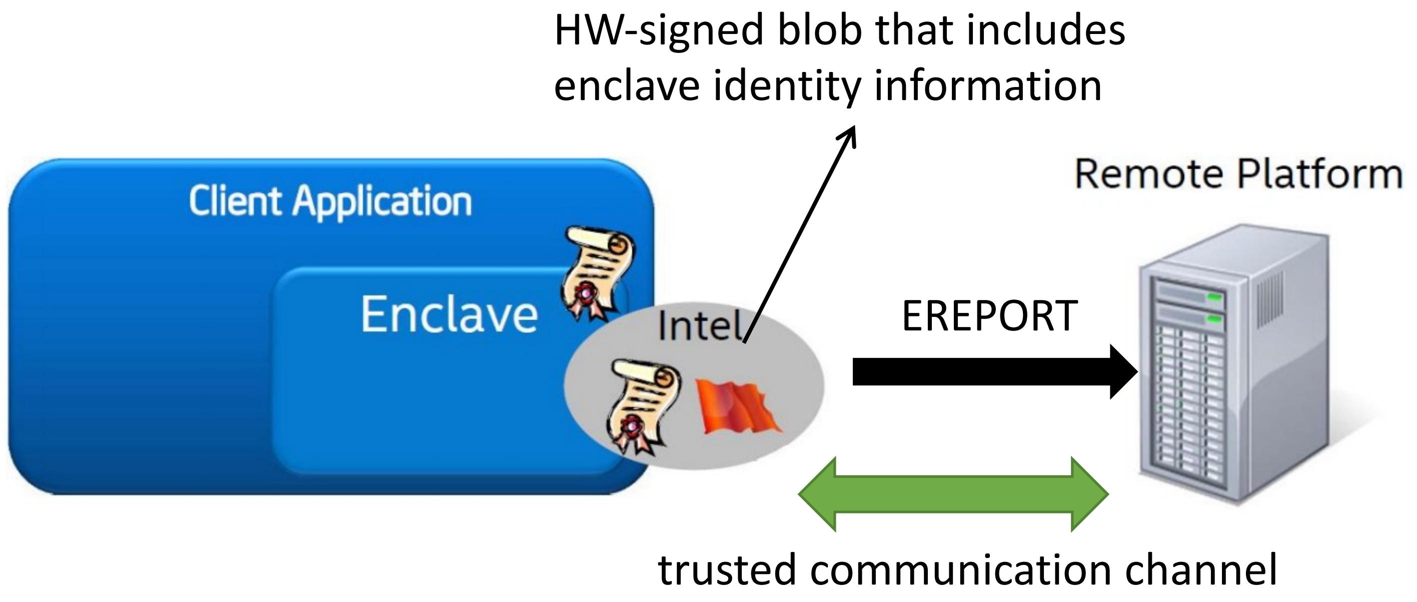
- Hardware generates a cryptographic log of the build process
  - Code, data, stack, and heap contents
  - Location of each page within the enclave
  - Security attributes (e.g., page permissions) and enclave capabilities

- Enclave identity (MRENCLAVE) is a 256-bit digest of the log that represents the enclave



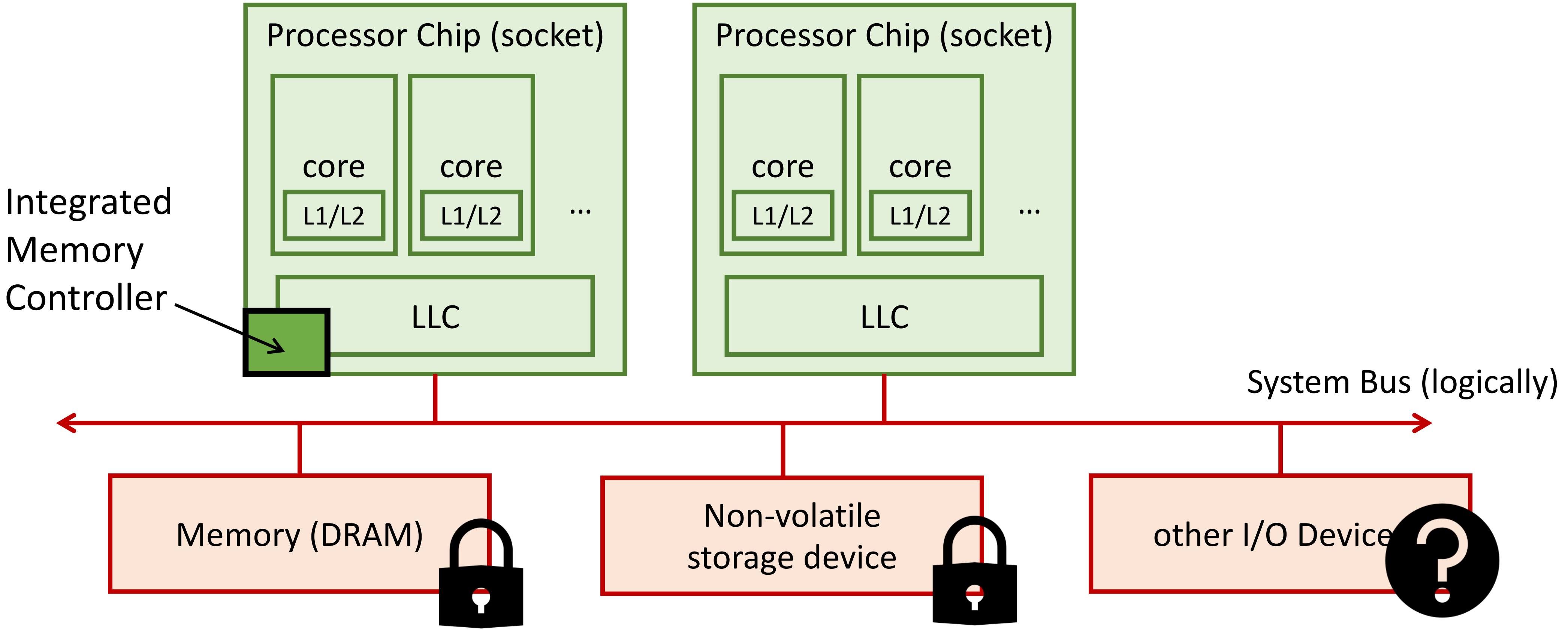
Enclave Attestation and Sealing

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


HW-signed blob that includes enclave identity information
Client Application
Enclave
Intel
Remote Platform
EREPORT
trusted communication channel

Additional Security Threats

- DRAM attacks: Rowhammer, Coldboot attacks



Memory Encryption Engine (MEE)

- Confidentiality:
  - DATA written to the DRAM cannot be distinguished from random data.

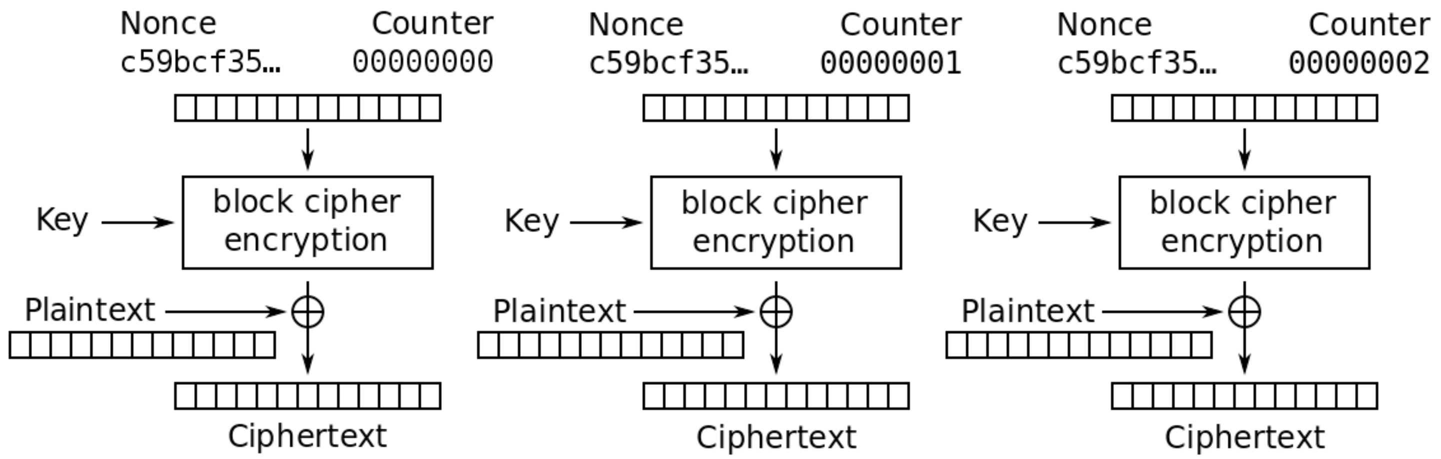
- Integrity + freshness:
  - DATA read back from DRAM to LLC is the same DATA that was most recently written from LLC to DRAM.

What attacks can be mitigated?

Rowhammer? Bus tapping? Side channels on address access?

Confidentiality

- AES 128-CTR mode



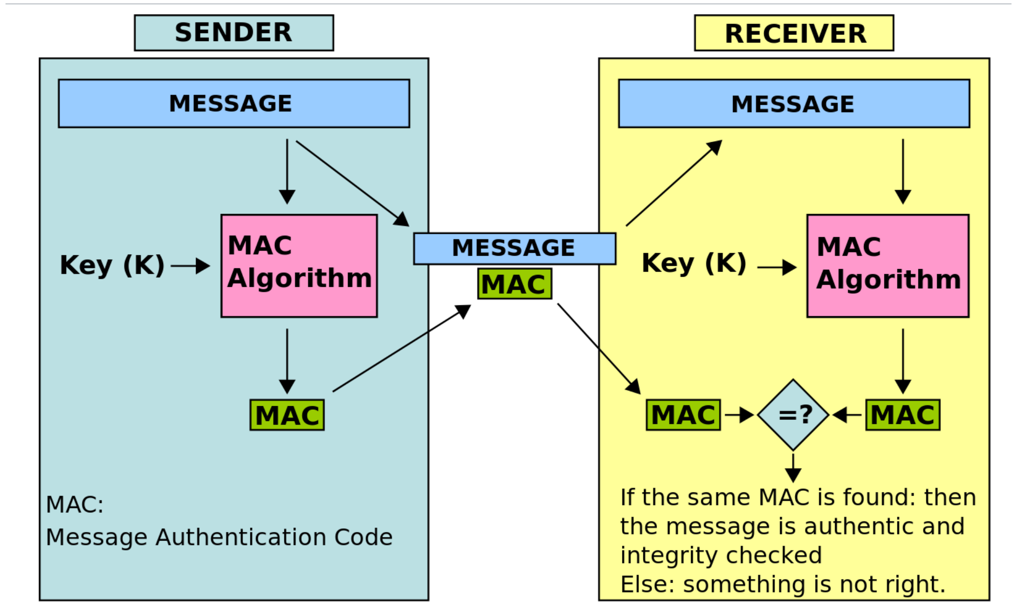
Counter (CTR) mode encryption

Message Authentication Code (MAC)

- Hash(plaintext)

- Keyed Hash
  - `hash = SHA(message)`
  - `HMAC = enc(hash, key)`

- Freshness
  - `hash = SHA(message || nounce)`
  - `HMAC = enc(hash, key)`

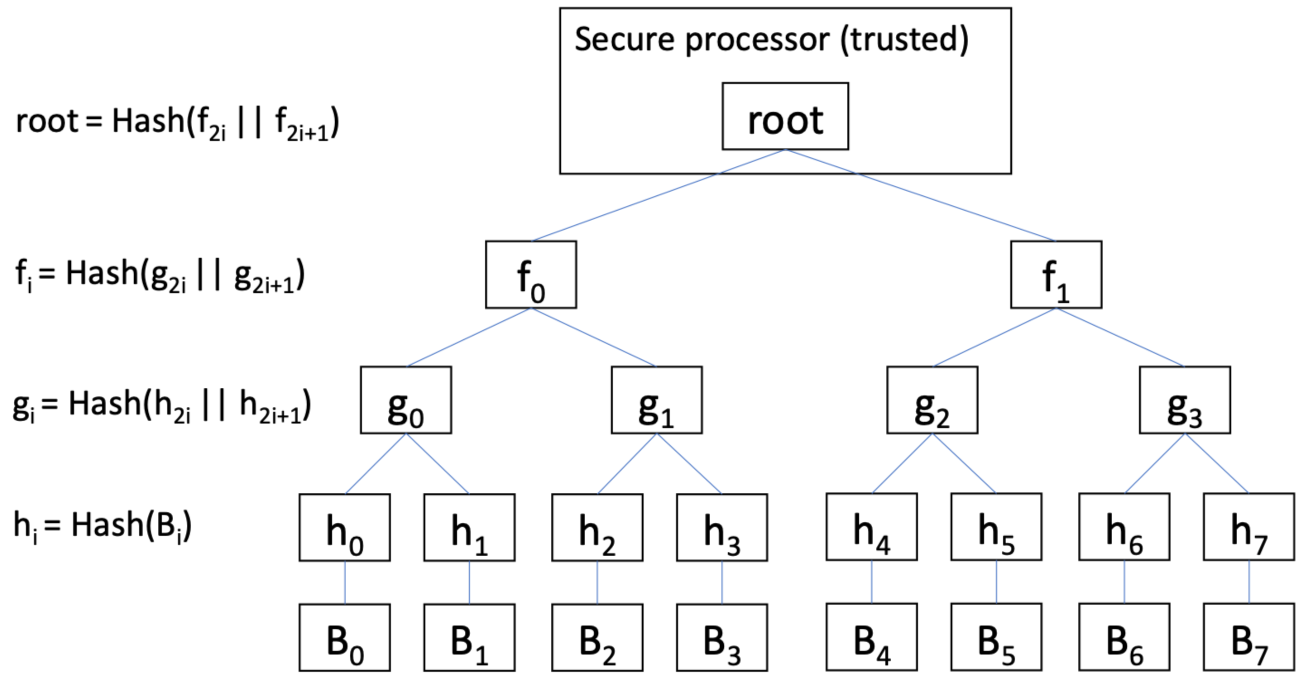

SENDER
MESSAGE
Key (K)
MAC Algorithm
MAC
MAC:
Message Authentication Code
MESSAGE
MAC
RECEIVER
MESSAGE
Key (K)
MAC Algorithm
MAC
=?
MAC
If the same MAC is found: then the message is authentic and integrity checked
Else: something is not right.

Integrity Storage Problem

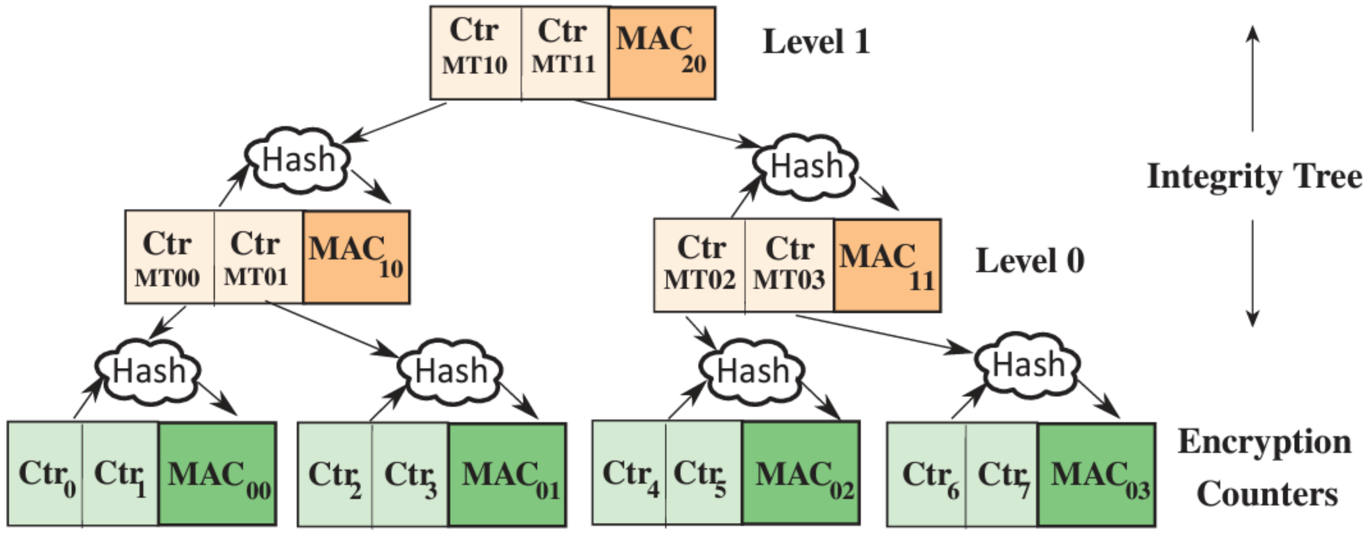
- For each cache line: {ciphertext + CTR + MAC}
  - MAC 56 bits
  - CTR 56 bits
- Can we store all the three components off-chip?
- Problem: if store CTR on-chip → high on-chip storage requirement

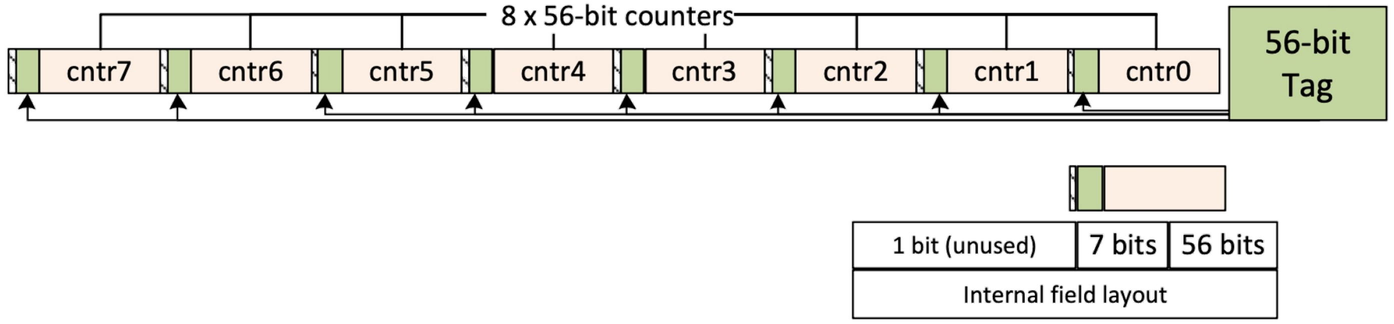
Operations on Merkle Tree

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


Secure processor (trusted)
root
root = Hash(f_{2i} || f_{2i+1})
f_i = Hash(g_{2i} || g_{2i+1})
g_i = Hash(h_{2i} || h_{2i+1})
h_i = Hash(B_i)
f_0
f_1
g_0
g_1
g_2
g_3
h_0
h_1
h_2
h_3
h_4
h_5
h_6
h_7
B_0
B_1
B_2
B_3
B_4
B_5
B_6
B_7

Bonsai-style Counter Integrity Tree





Summary

- What can privilege software attackers do?
- Design tradeoffs between TCB size, flexibility, perf overhead, cost, etc.
  - Intel SGX, AMD SEV, ARM CCA
  - Keystone, Sanctum, Penglai, etc