CS 453/698: Software and Systems Security

Module: Hardware & Mobile Security

Lecture: Trusted Execution Environments

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:

Trusted Platform Modules

- Root of trust for storage → storage root key
- Root of trust for reporting → endorsement key
- Architecture & Operations
- TPM-based attestation protocols
  - Quote-based
  - Seal-based

Today

Continue: Hardware and Mobile Security

What are some techniques that address limitations of TPMs?

Some limitations of TPMs…

- Not programmable
  - Dedicated cryptographic co-processor
  - Fixed functionality
- Do not provide a runtime environment:
  - Protects secrets from compromised host
  - Provides integrity of reports despite compromised host
- Passive module
  - No availability guarantees if the host is compromised

System Security (so far)

- Boot process loads the runtime environment (e.g., the OS)

- Further runtime security typically provided by the OS
  - OS provides inter-process isolation
  - Virtualization via memory management units (MMUs)
  - A core component for many other runtime security features
  - Control Flow Integrity
  - Compartmentalization
  - Etc…

System Security (so far)

- Boot process loads the runtime environment (e.g., the OS)

- Further runtime security typically provided by the OS
  - OS provides inter-process isolation
  - Virtualization via memory management units (MMUs)
  - A core component for many other runtime security features
  - Control Flow Integrity
  - Compartmentalization
  - Etc…

- OS also does a lot of other things for our system…
  - Memory allocation, configuring MMU, context switches
  - Task scheduling
  - Interfacing with hardware peripherals
  - Disk, network stack, GPU
  - **TPM & MMU**

System Security (so far)

- Boot process loads the runtime environment (e.g., the OS)

- Further runtime security typically provided by the OS
  - OS provides inter-process isolation
  - Virtualization via memory management units (MMUs)
  - A core component for many other runtime security features
  - Control Flow Integrity
  - Compartmentalization
  - Etc…

- OS also does a lot of other things for our system…
  - Memory allocation, configuring MMU, context switches
  - Task scheduling
  - Interfacing with hardware peripherals
  - Disk, network stack, GPU
  - **TPM & MMU**

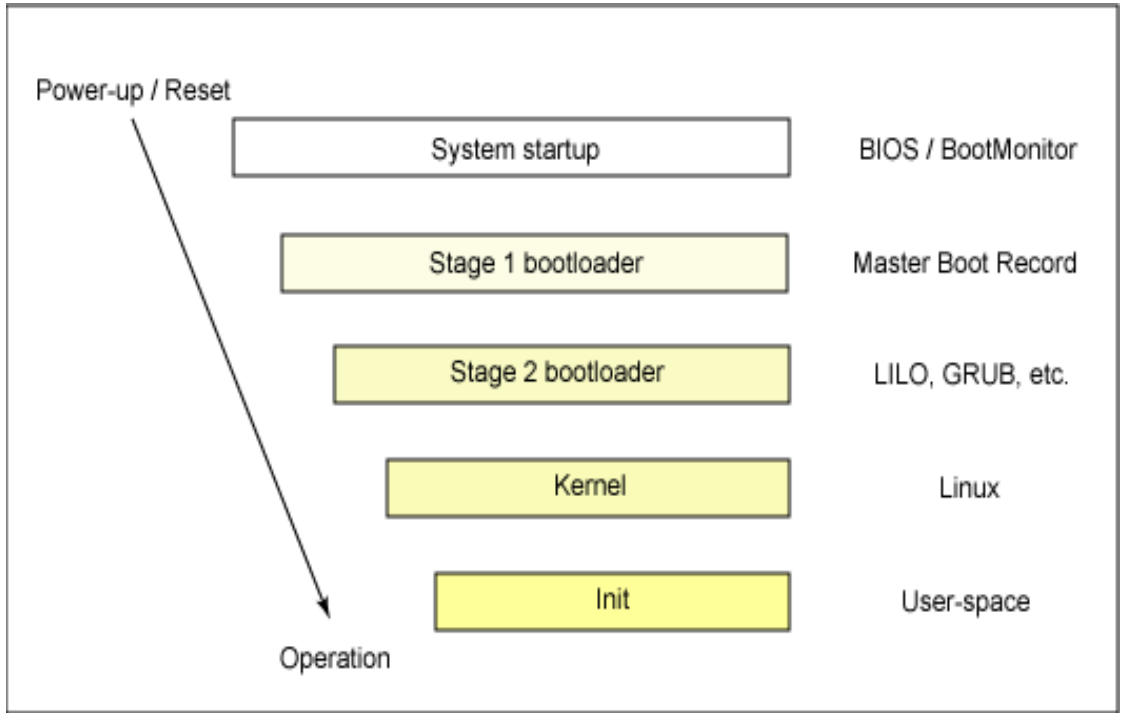
Millions of lines of low-level & complex code running as privileged!

Huge TCB!!!

A single vulnerability in the privileged code can undermine all guarantees.

OS-based runtime (in)security

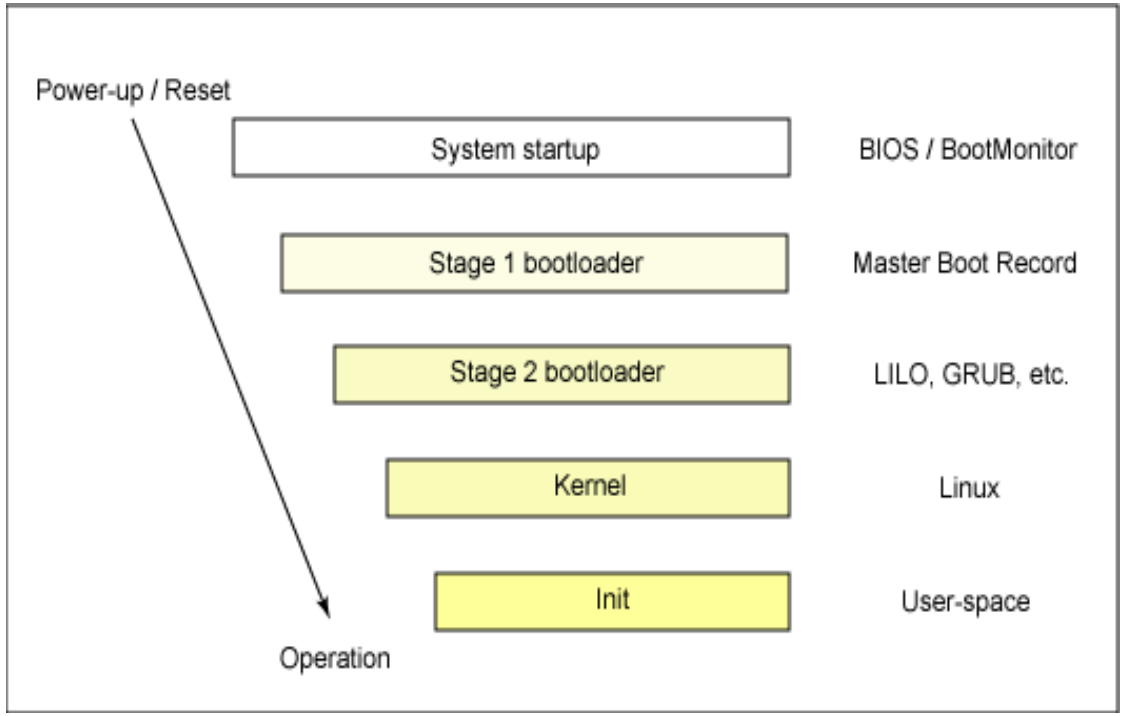
Boot



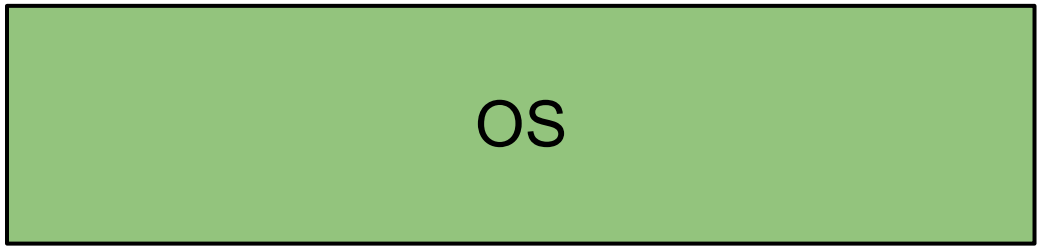
Loads some trusted
Operating System (OS)

OS-based runtime (in)security

Boot



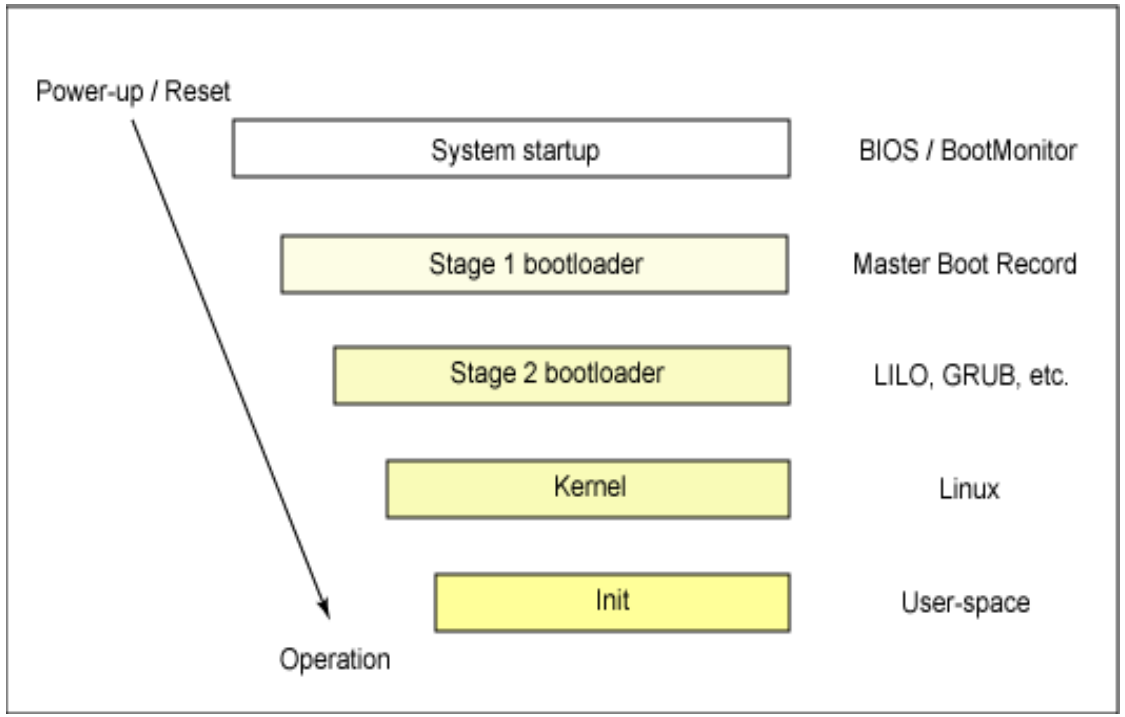
Loads some trusted Operating System (OS)



OS: launch and manage the system and applications

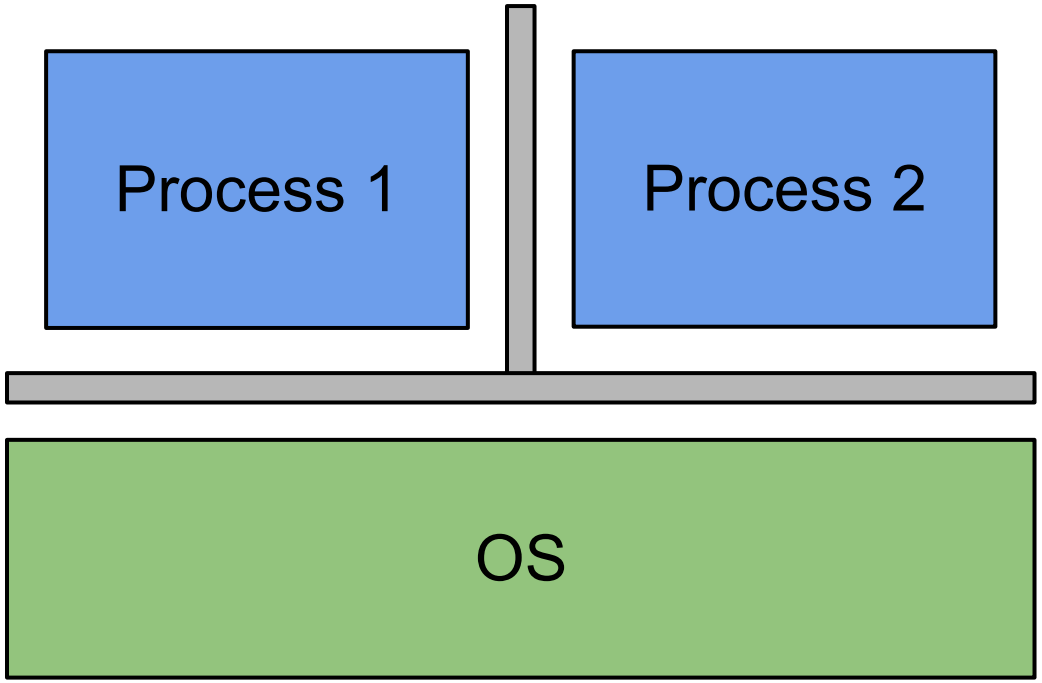
OS-based runtime (in)security

Boot



Loads some trusted Operating System (OS)

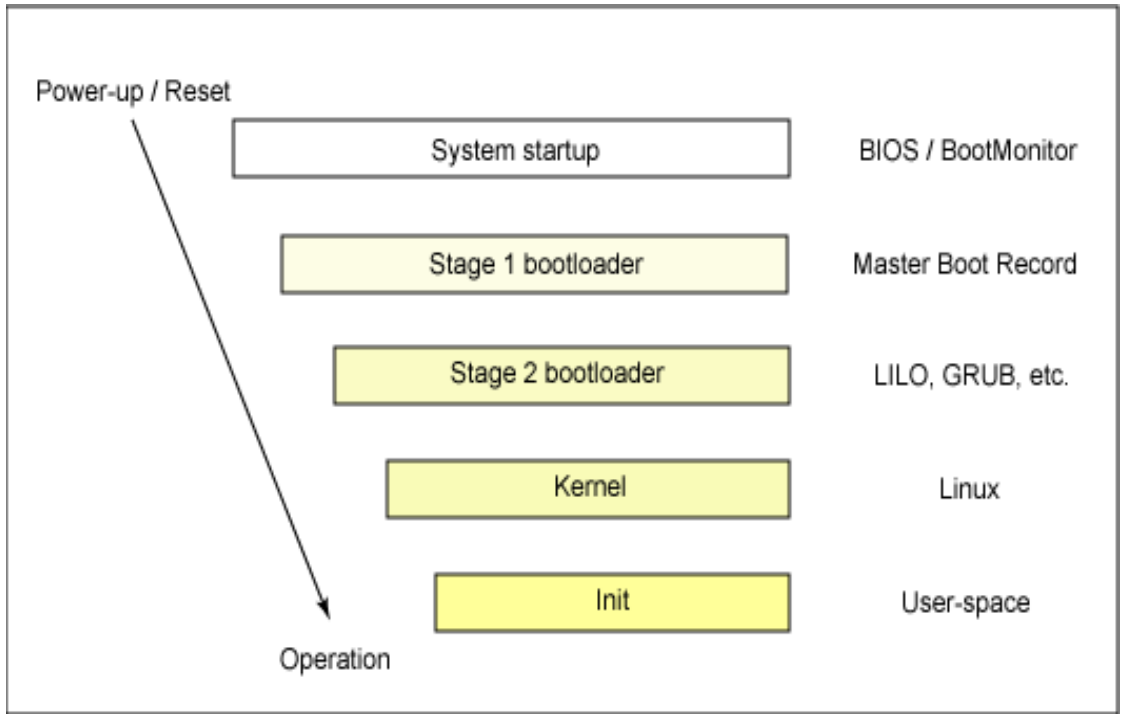
Runtime System Isolation



OS: launch and manage the system and applications

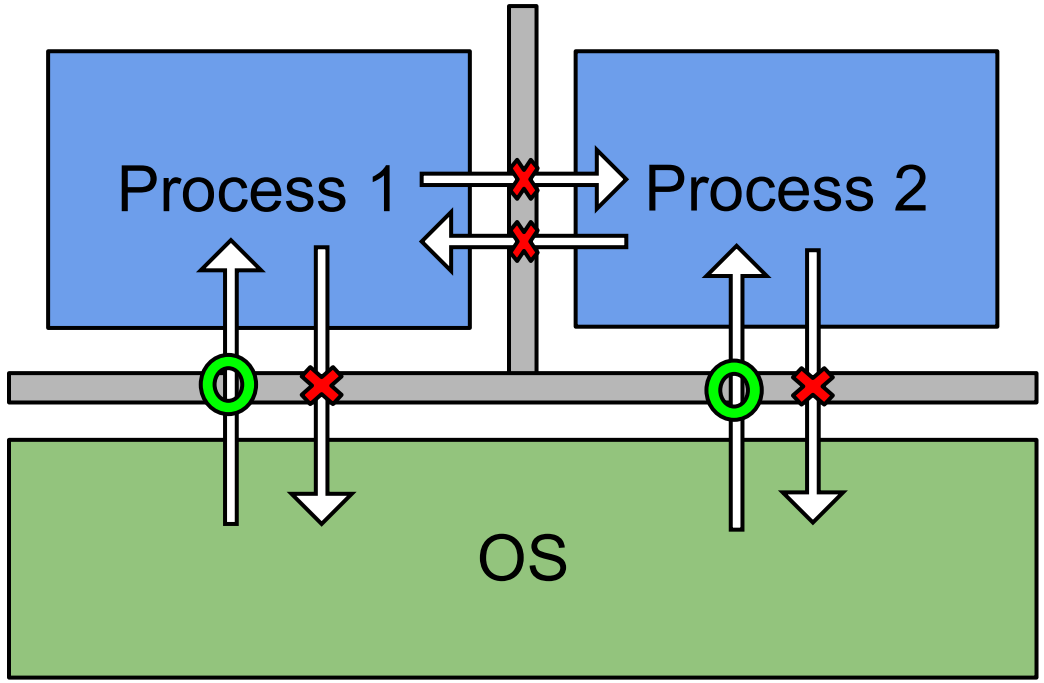
OS-based runtime (in)security

Boot



Loads some trusted Operating System (OS)

Runtime System Isolation

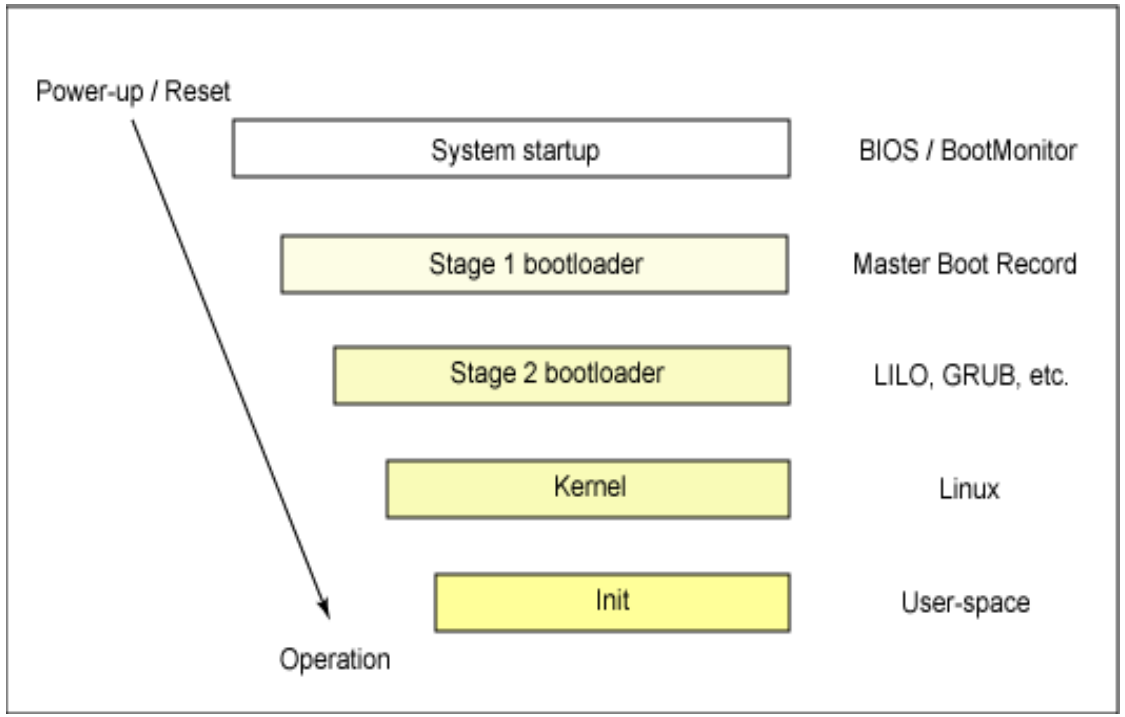


OS: launch and manage the system and applications

OS-based runtime (in)security

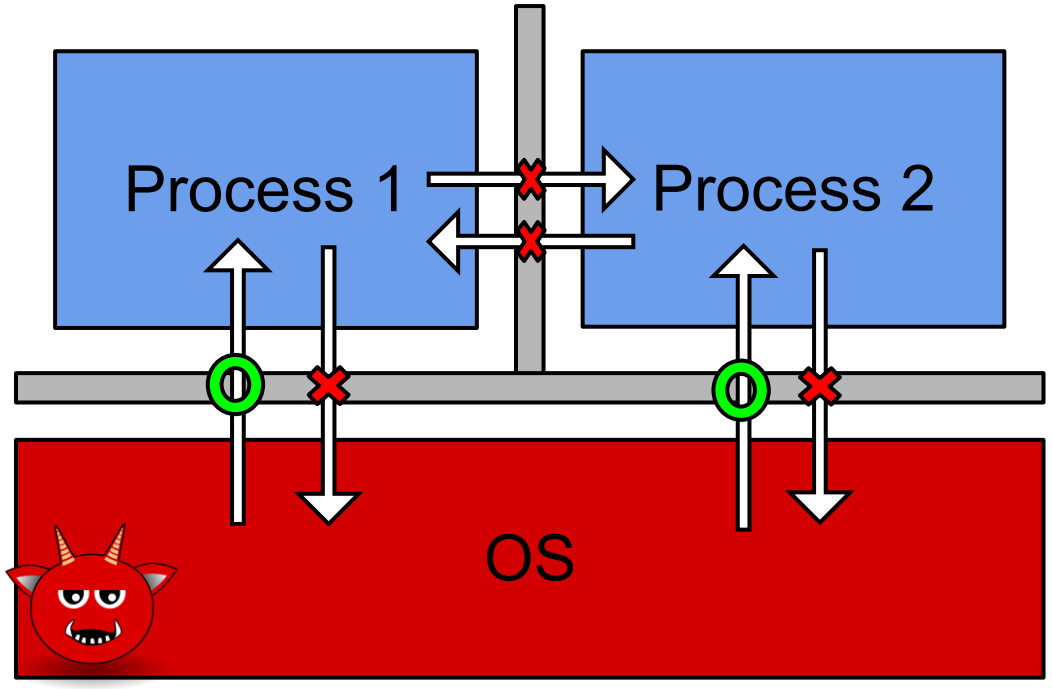
1 – Malware exploits some OS vulnerability at runtime

Boot



Loads some trusted Operating System (OS)

Runtime System Isolation

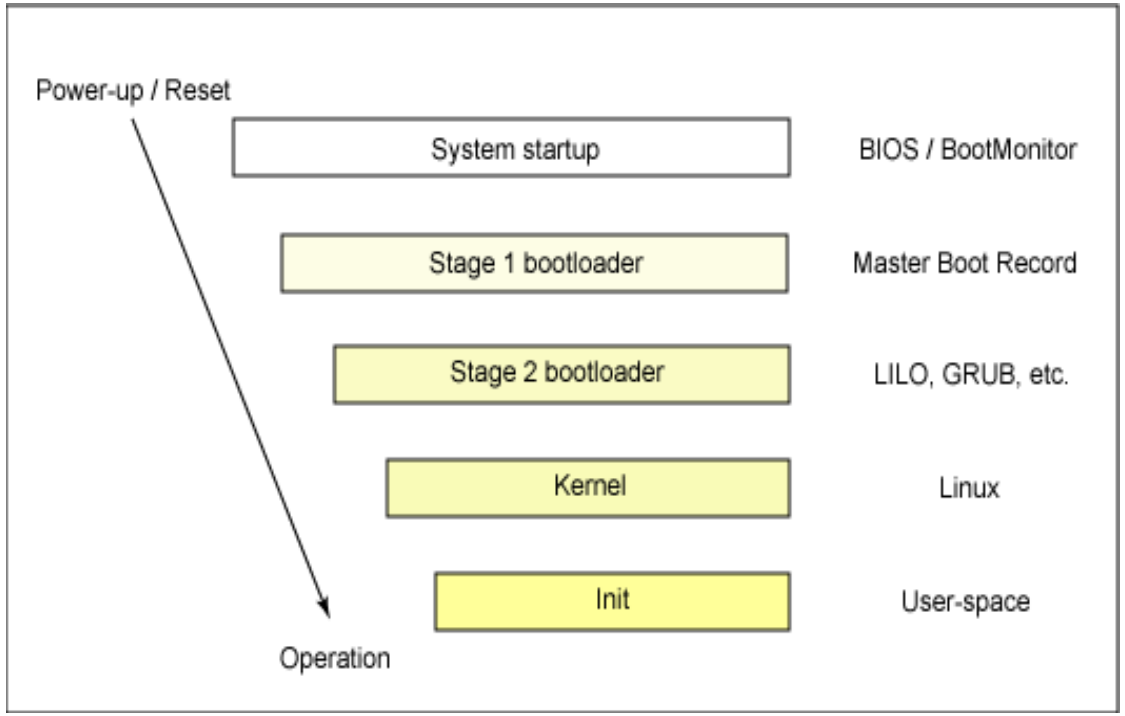


OS: launch and manage the system and applications

OS-based runtime (in)security

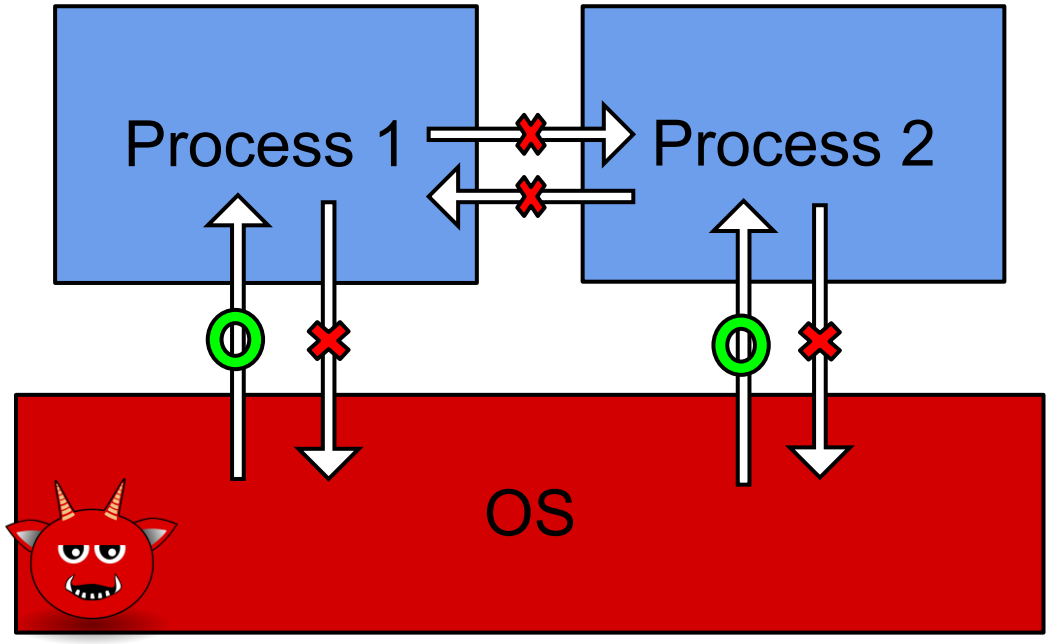
2 – Modifies page tables/MMU at will

Boot



Loads some trusted Operating System (OS)

Runtime System Isolation

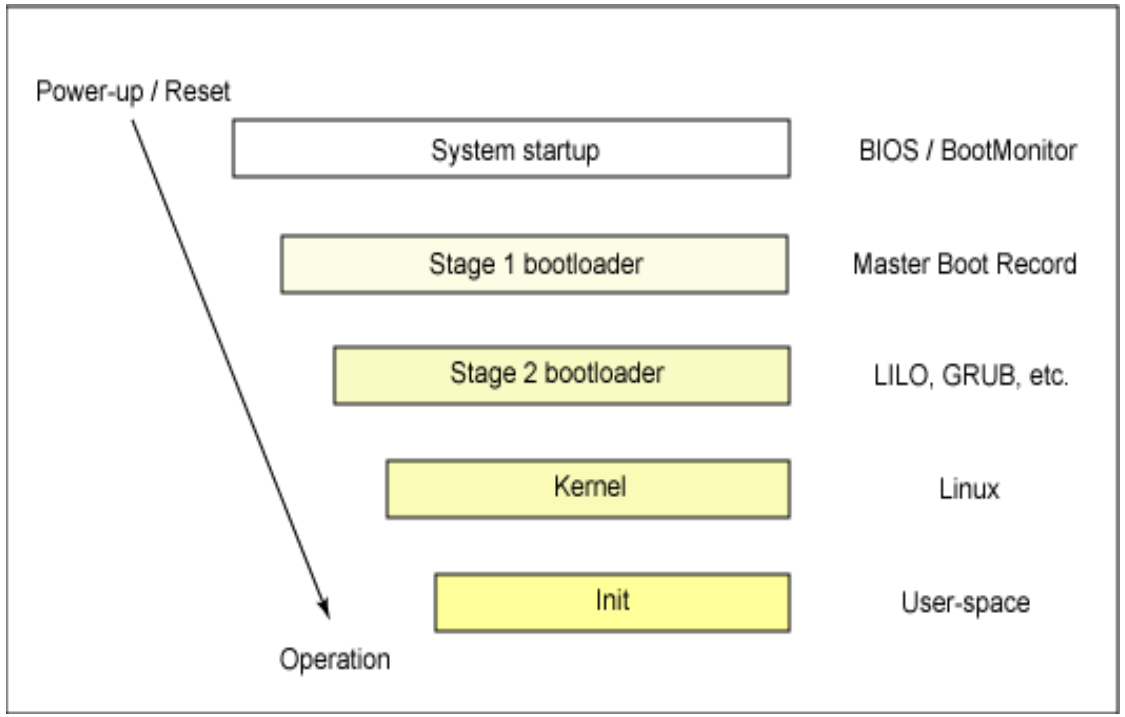


OS: launch and manage the system and applications

OS-based runtime (in)security

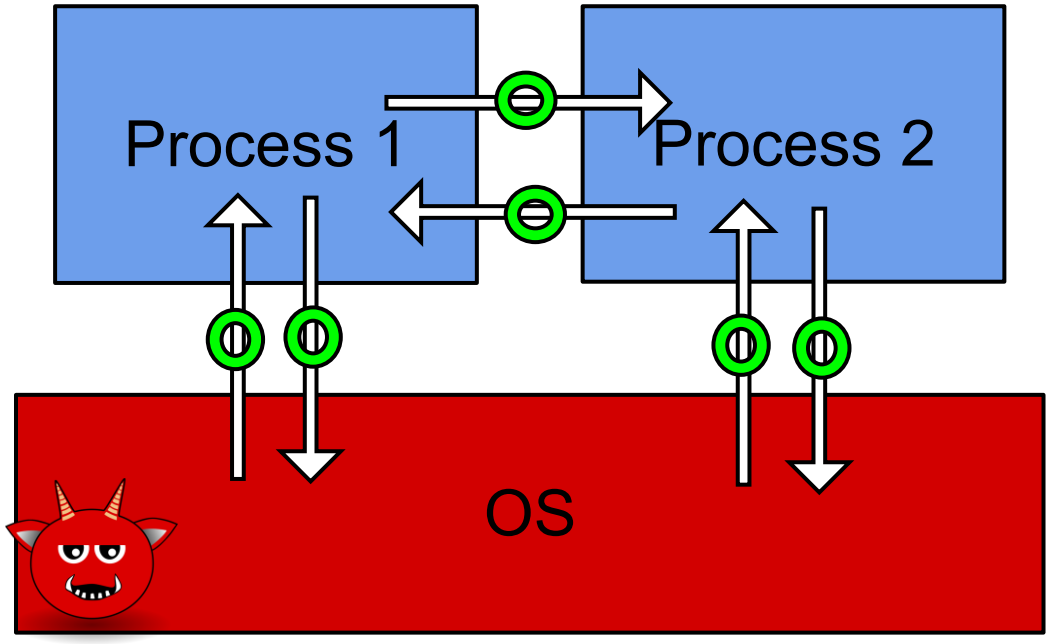
2 – Modifies page tables/MMU at will

Boot



Loads some trusted Operating System (OS)

Runtime System Isolation

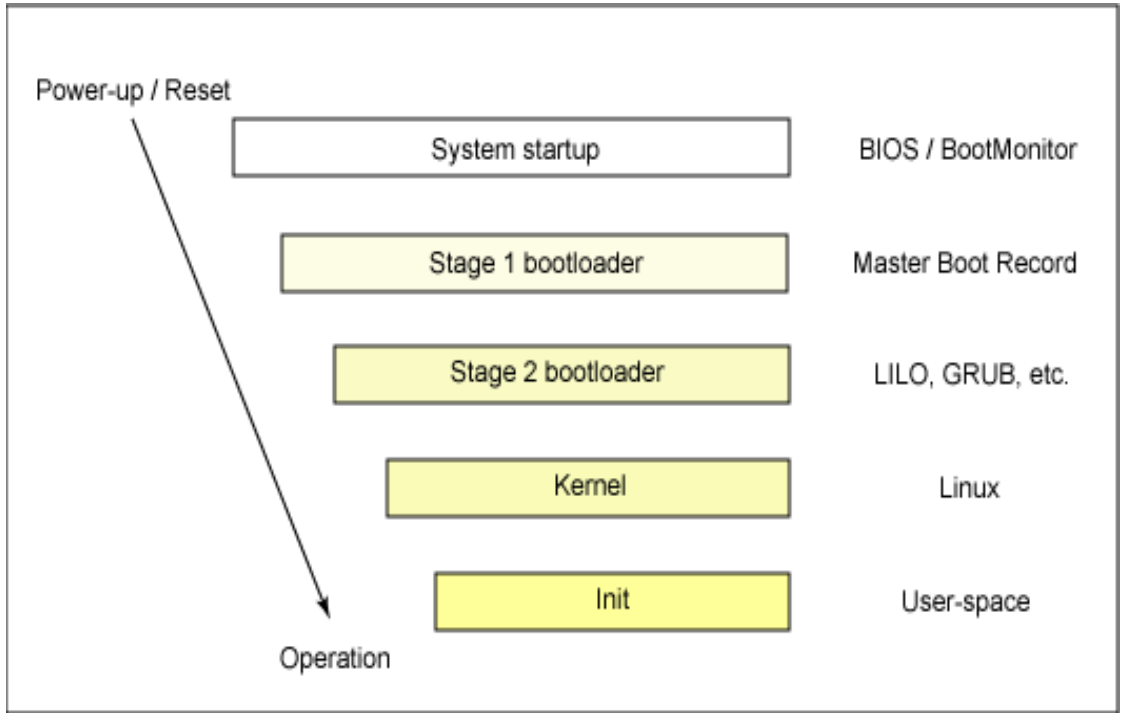


OS: launch and manage the system and applications

OS-based runtime (in)security

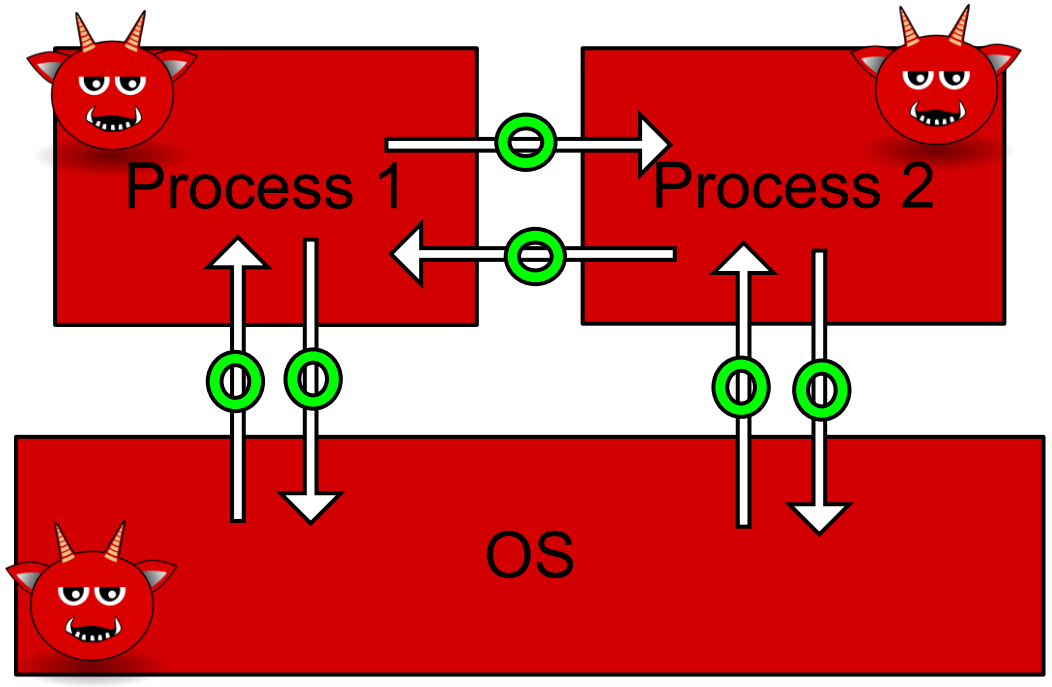
3 – Complete Host Compromise

Boot



Loads some trusted Operating System (OS)

Runtime System Isolation



OS: launch and manage the system and applications

Trusted Execution Environments (TEEs)

- Set of techniques and architectures aimed to reduce the size of the TCB implementing the runtime security guarantees

Trusted Execution Environments (TEEs)

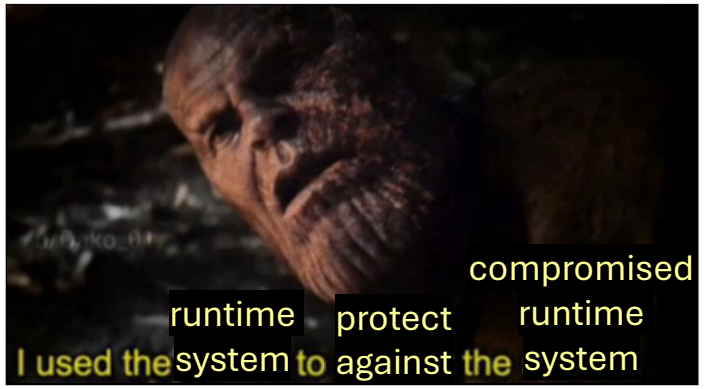
- Set of techniques and architectures aimed to reduce the size of the TCB implementing the runtime security guarantees
- Designed to withstand full compromise of the “feature-rich operating system” (e.g., Linux, Android, etc)
- How?

Trusted Execution Environments (TEEs)

- Set of techniques and architectures aimed to reduce the size of the TCB implementing the runtime security guarantees
- Designed to withstand full compromise of the “feature-rich operating system” (e.g., Linux, Android, etc)
- How?

  → Creates ***another runtime system isolated from the main system*** dedicated for security critical tasks

  - Manipulate cryptographic secrets
  - Compute on privacy sensitive data



Trusted Execution Environments (TEEs)

TEEs are applied in various system models.

- User-space
- System-wide
- Virtual machines
- Servers, laptops, desktops, mobile devices, embedded systems
- Various names depending on the model
  - “Trusted Part”, “trusted world”, “secure world”, “enclave”

TEE vs. TPMs

TPMs

- TPM secrets are not visible outside of the TPM

TEEs

TEE vs. TPMs

TPMs

- TPM secrets are not visible outside of the TPM

TEEs

- TEE secrets are not visible outside of the TEE

TEE vs. TPMs

TPMs

- TPM secrets are not visible outside of the TPM
- Small, well-defined API to communicated with trusted TPM

TEEs

- TEE secrets are not visible outside of the TEE

TEE vs. TPMs

TPMs

- TPM secrets are not visible outside of the TPM
- Small, well-defined API to communicated with trusted TPM

TEEs

- TEE secrets are not visible outside of the TEE
- Small, well-defined API to communicated with trusted “world”

TEE vs. TPMs

TPMs

- TPM secrets are not visible outside of the TPM
- Small, well-defined API to communicated with trusted TPM
- Fixed functionality (specified by TCG, implemented by TPM manufacturer)

TEEs

- TEE secrets are not visible outside of the TEE
- Small, well-defined API to communicated with trusted “world”

TEE vs. TPMs

TPMs

- TPM secrets are not visible outside of the TPM
- Small, well-defined API to communicated with trusted TPM
- Fixed functionality (specified by TCG, implemented by TPM manufacturer)

TEEs

- TEE secrets are not visible outside of the TEE
- Small, well-defined API to communicated with trusted “world”
- Programmable!

TEE vs. TPMs

Key difference: TEEs are programmable!

- Programmer can decide the behavior of the “trusted part”

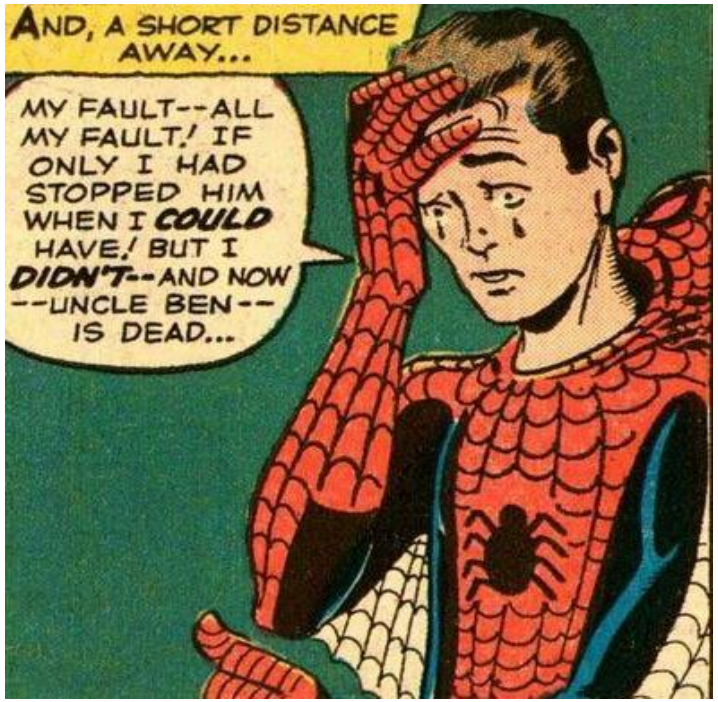
TEE vs. TPMs

Key difference: TEEs are programmable!

- Programmer can decide the behavior of the “trusted part”
- That comes with a great responsibility….

TEE Manufacturer

TEE Programmer



Trusted Execution Environments

No magic:

- **Isolation:** A bug in the rich OS will not compromise the TEE

BUT …

- A bug in the TEE's trusted part can still compromise the TEE so…

Trusted Execution Environments

No magic:

- **Isolation:** A bug in the rich OS will not compromise the TEE

BUT …

- A bug in the TEE's trusted part can still compromise the TEE so…

A TEE's trusted part (TCB) must be kept small and simple for verification.

Whenever possible, non-security critical functions should remain outside of the trusted part

TEE vs. TPMs (continued)

TPMs

- Independent peripheral device
- Passive in nature

TEEs

- Usually implemented as a part of the main CPU itself
- Sometimes can be “active” (see more next lecture…)

Trusted Execution Environments (TEE)

In this course, we will cover TEEs from two perspectives:

User-space TEEs

- Focused on high-end systems
- Intel SGX – today!

System-level TEE (“split-world”)

- Applicable to high-end, mobile, embedded systems
- ARM TrustZone in Android – next class

Intel SGX Overview

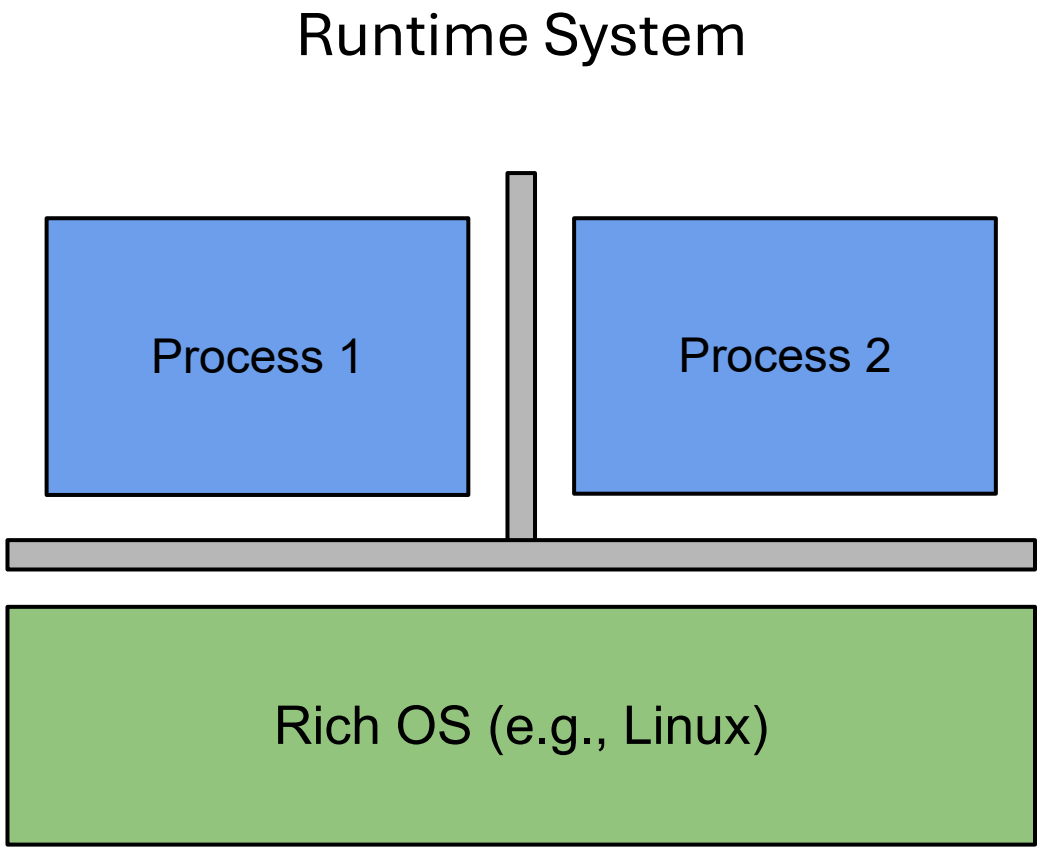
What is Intel SGX's approach for creating a TEE?

Intel SGX Approach:

- Add CPU hardware support to create one (or multiple) "mini-secure worlds" for each user-space process

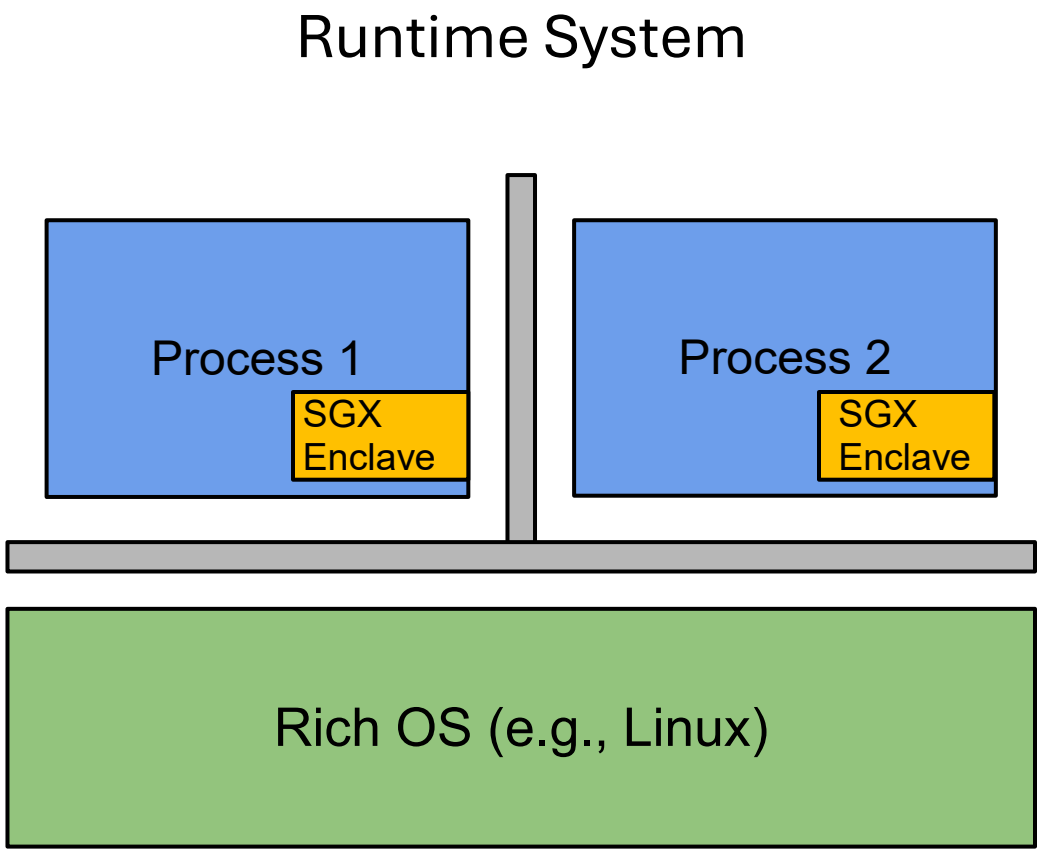
- A "mini-secure world" is called **enclave**

Intel SGX Overview

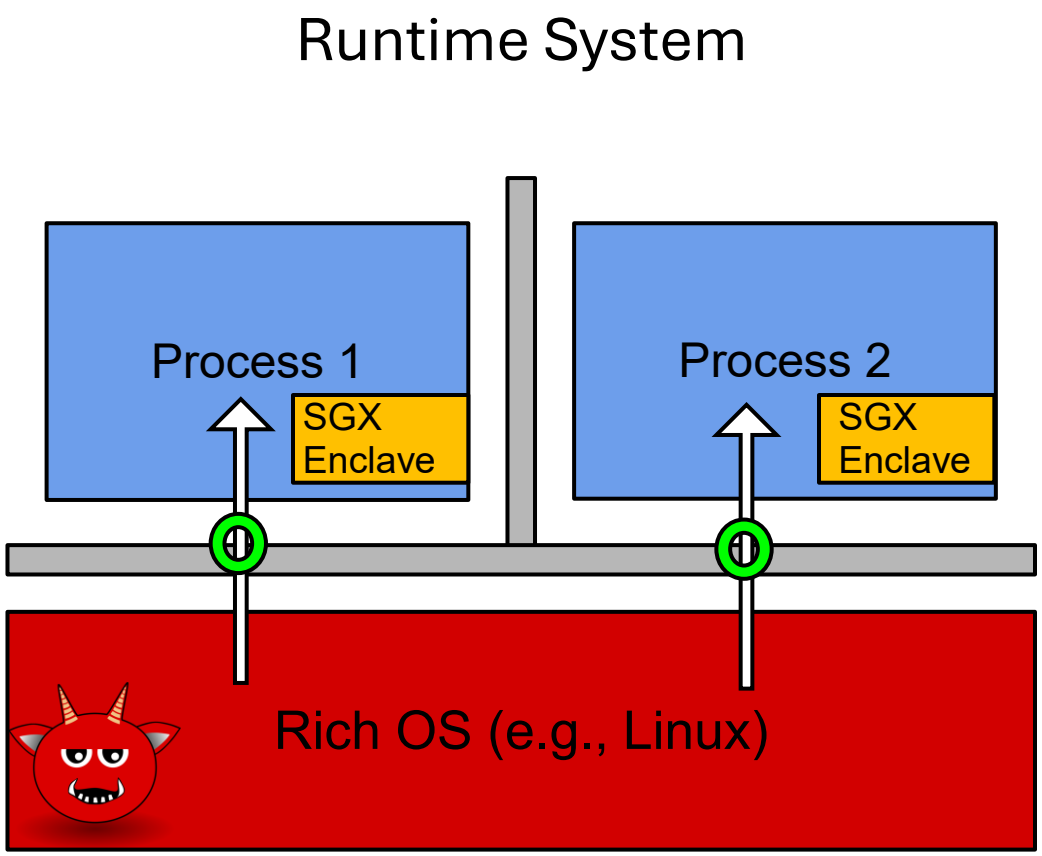


Intel SGX Overview

Enclave is a protected part of the user-space process



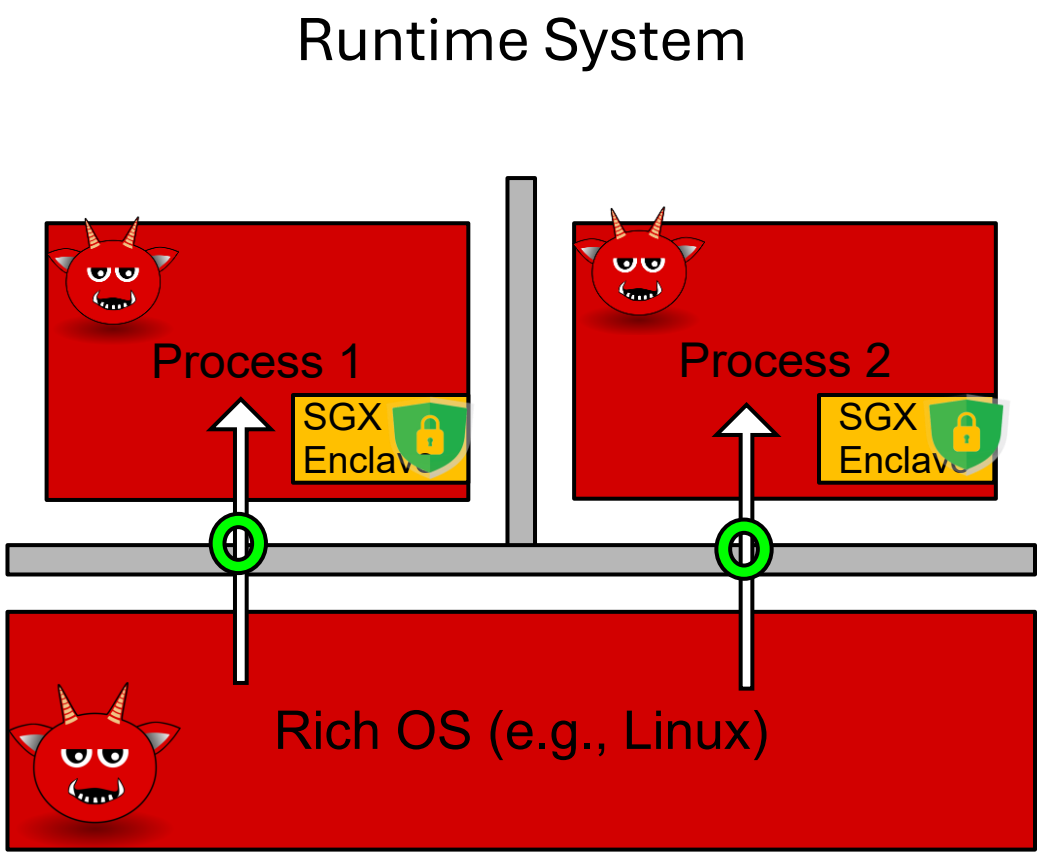
Intel SGX Overview



Intel SGX Overview

Full software compromise

BUT, the hardware guarantees that the enclave memory remains inaccessible



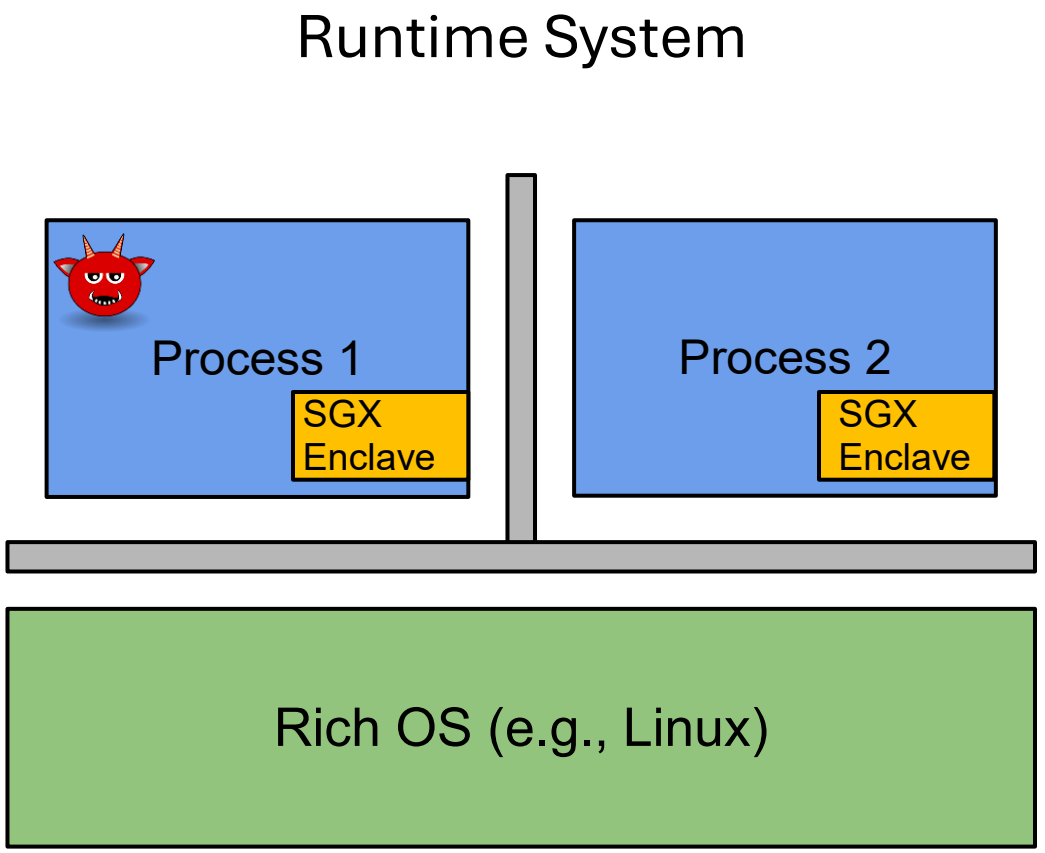
Intel SGX Overview

Each enclave lives within the virtual space of its process

- Unprivileged → cannot access OS resources directly
- But, OS is also **“unprivileged with respect to enclave contents”**
  - Only the enclave can access its own code and data
- Enclave:
  - Can only access data code/data within the same process
- A compromised enclave can not escalate to the OS or other processes

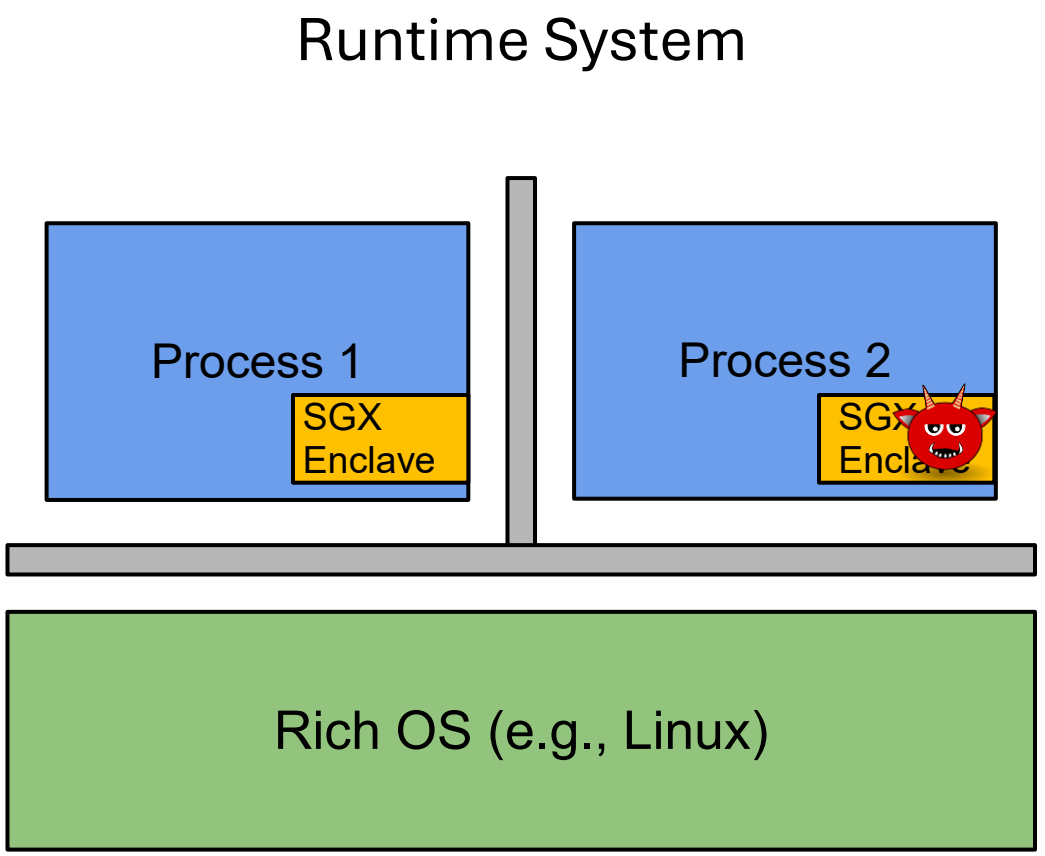
Intel SGX Overview

What happens?



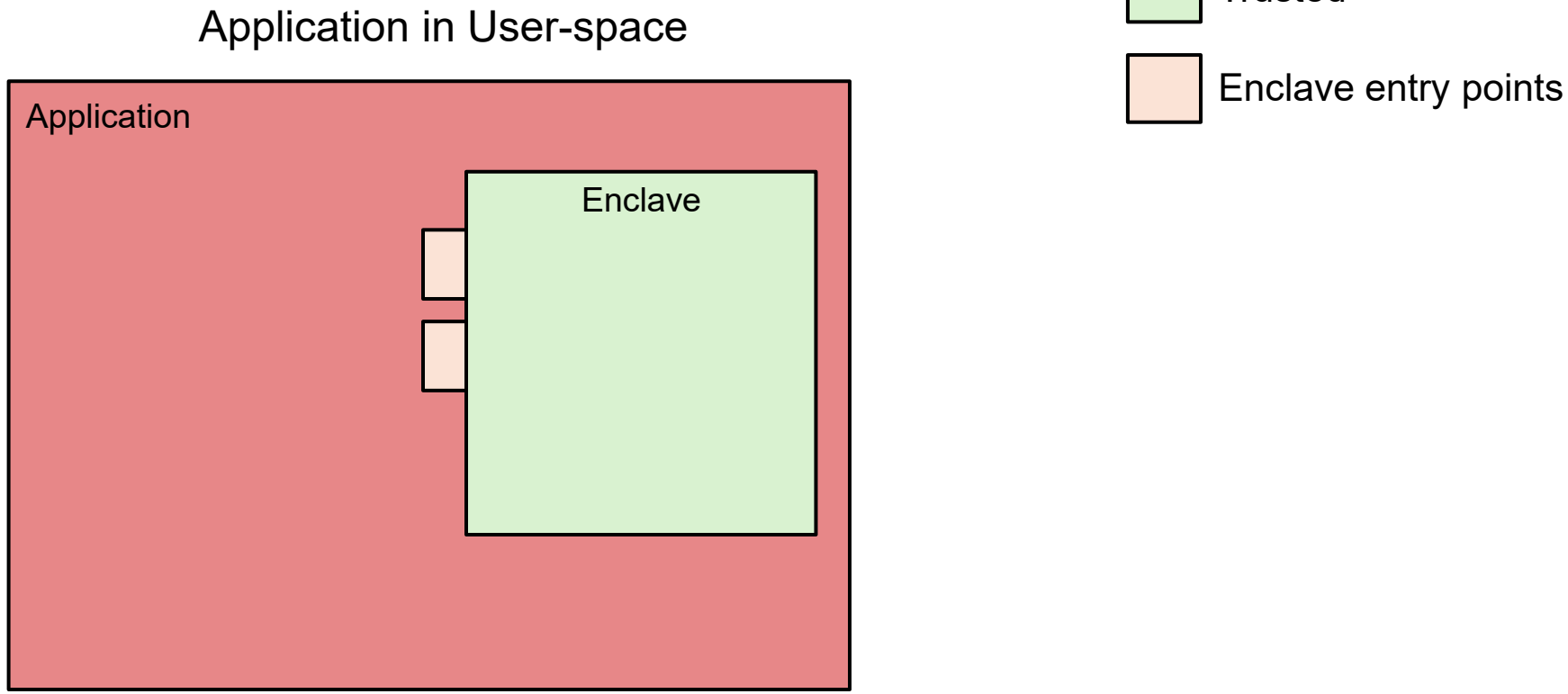
Intel SGX Overview

What happens?



Intel SGX Overview

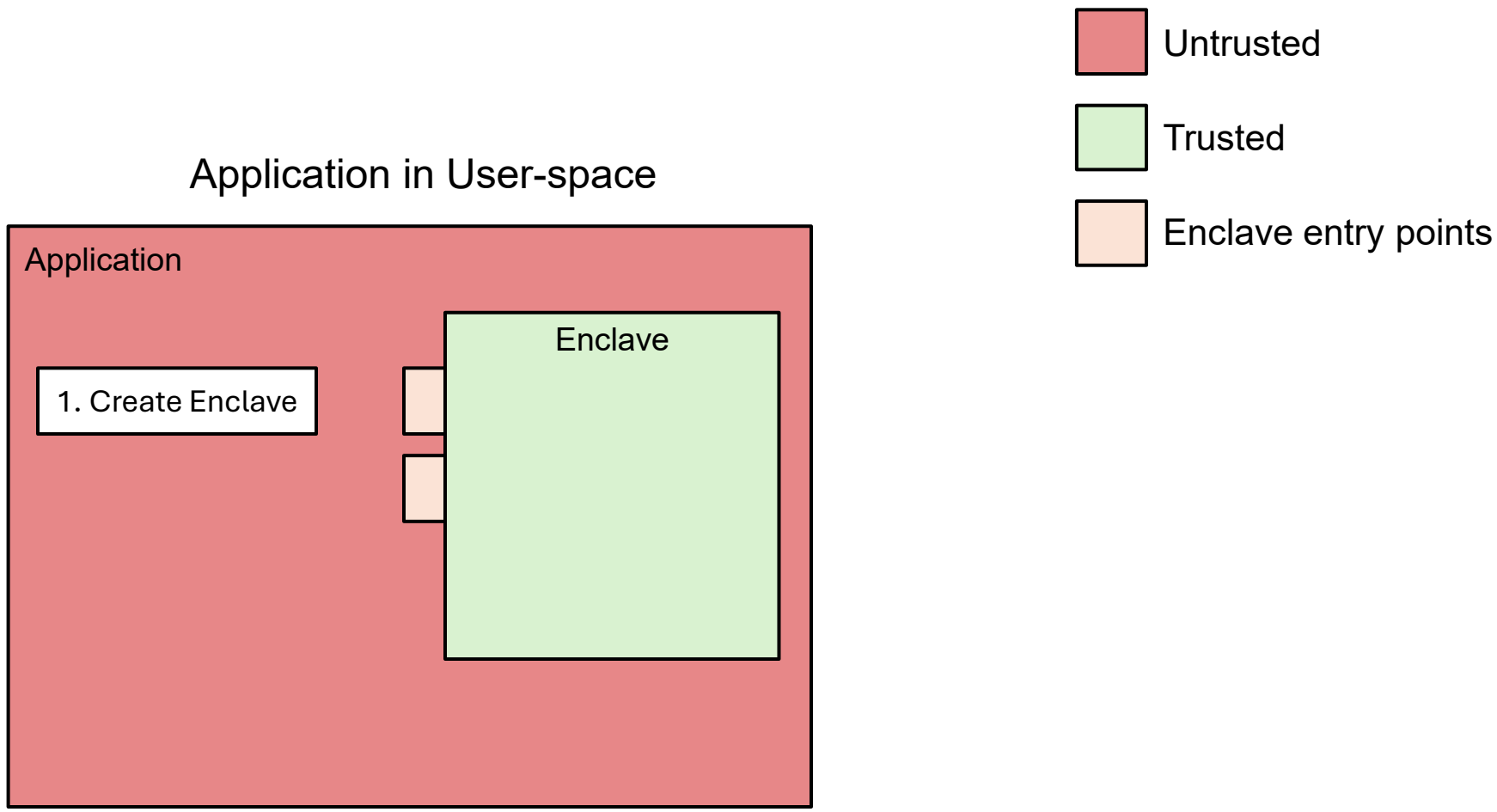
Flow within a process…



Applications are split into two parts: the secure and non-secure

Intel SGX Overview

Flow within a process…



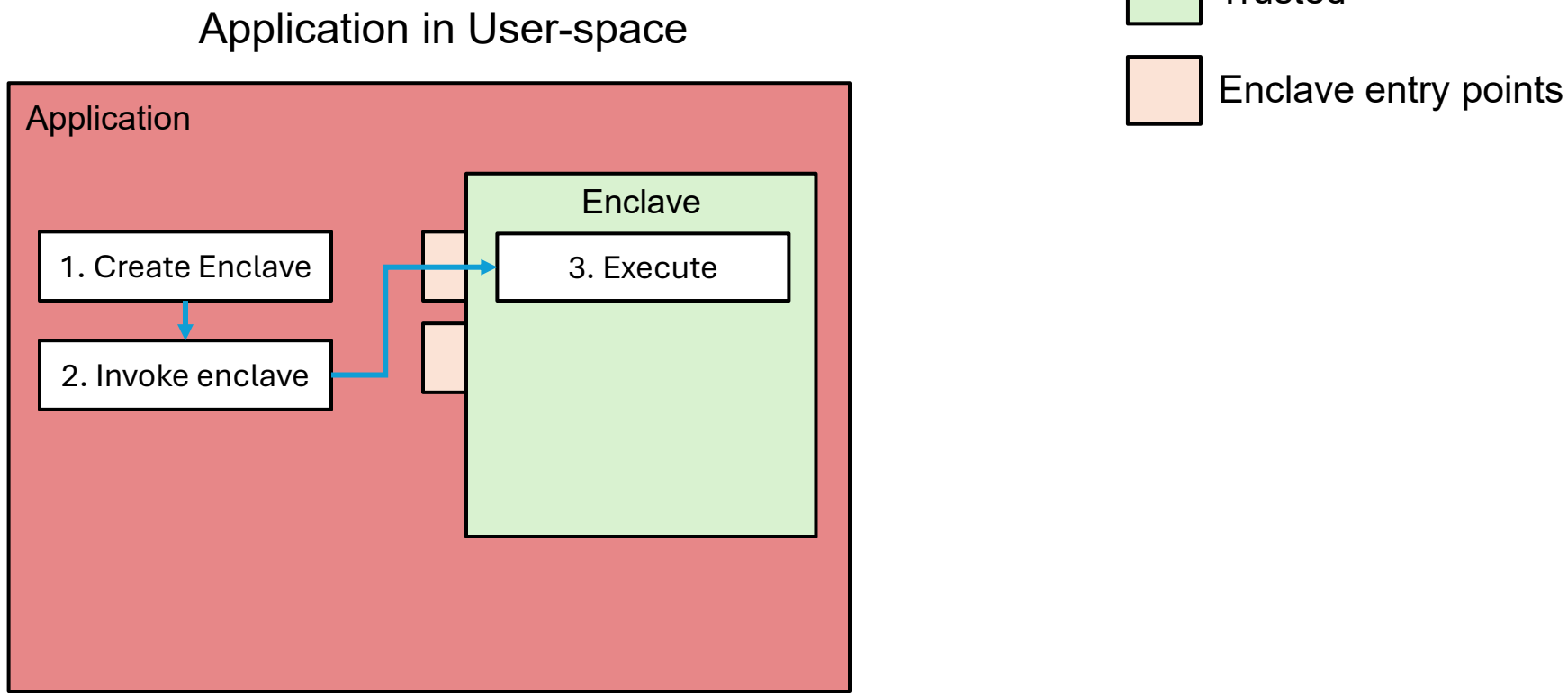
The Non-secure part defines and launches the secure enclave

The enclave is placed in SGX reserved & protected part of memory

How? CPU architectural features (coming up…)

Intel SGX Overview

Flow within a process…

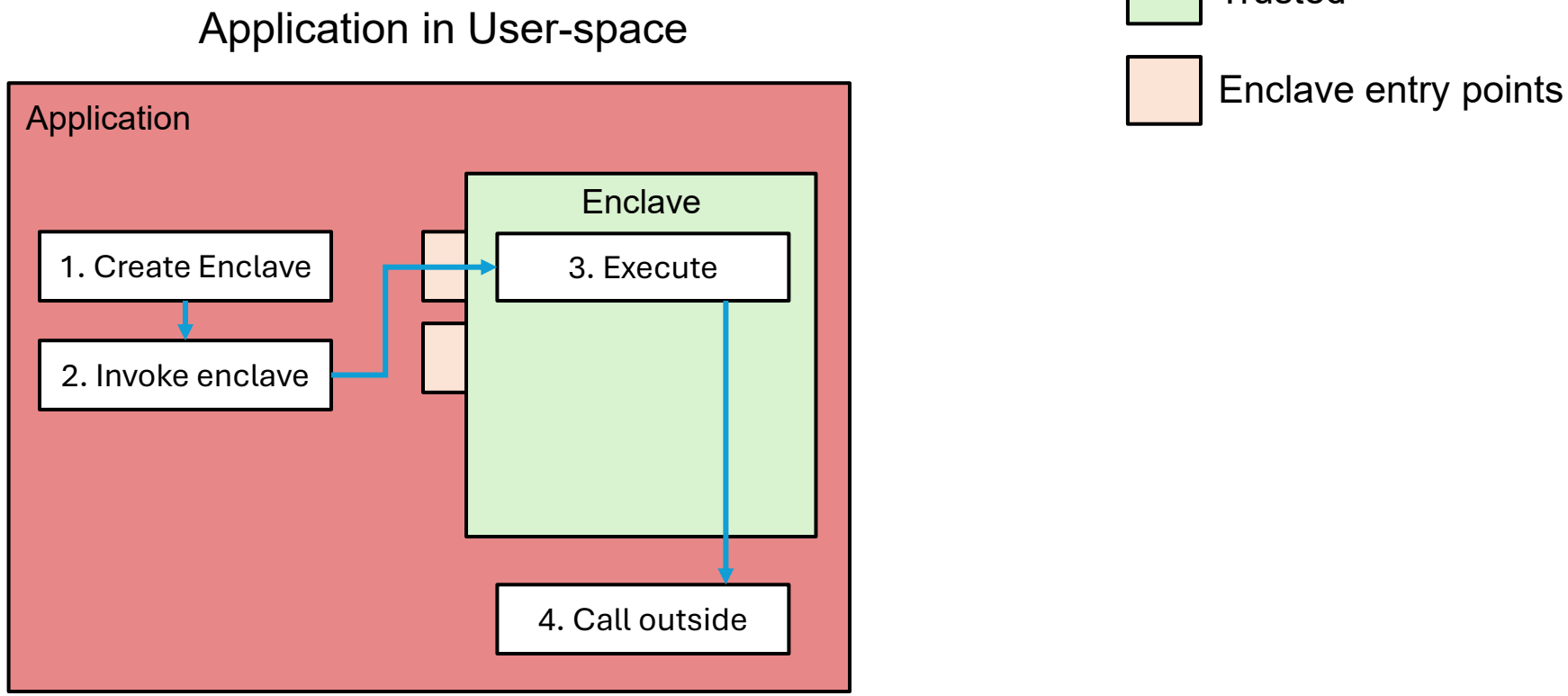


When an enclave function is called, it must go through a dedicated entry point. This is in the form of an **ecall**

Once inside, only enclave code can access its data.

Intel SGX Overview

Flow within a process…

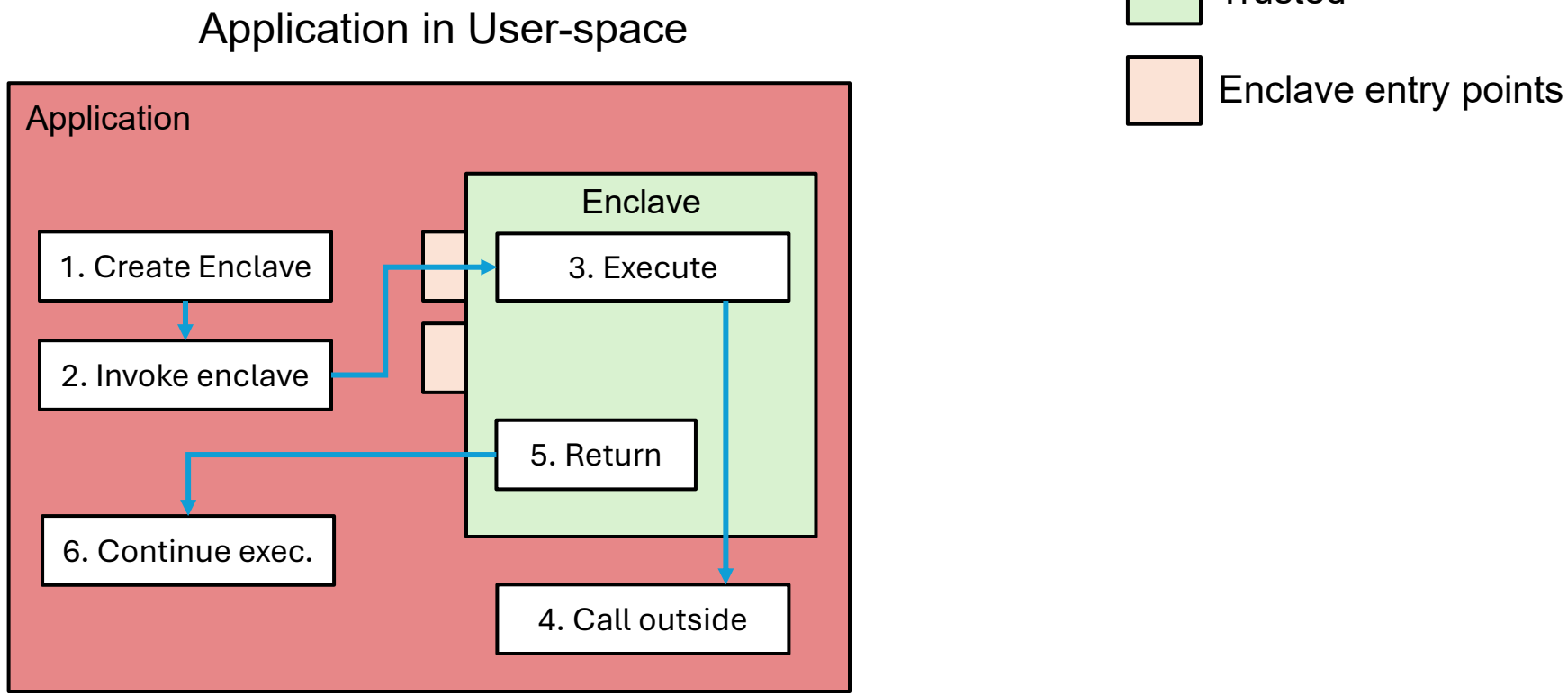


While executing, it might need to call a function outside of the enclave

(e.g., a syscall). This is in the form of an **ocall**

Intel SGX Overview

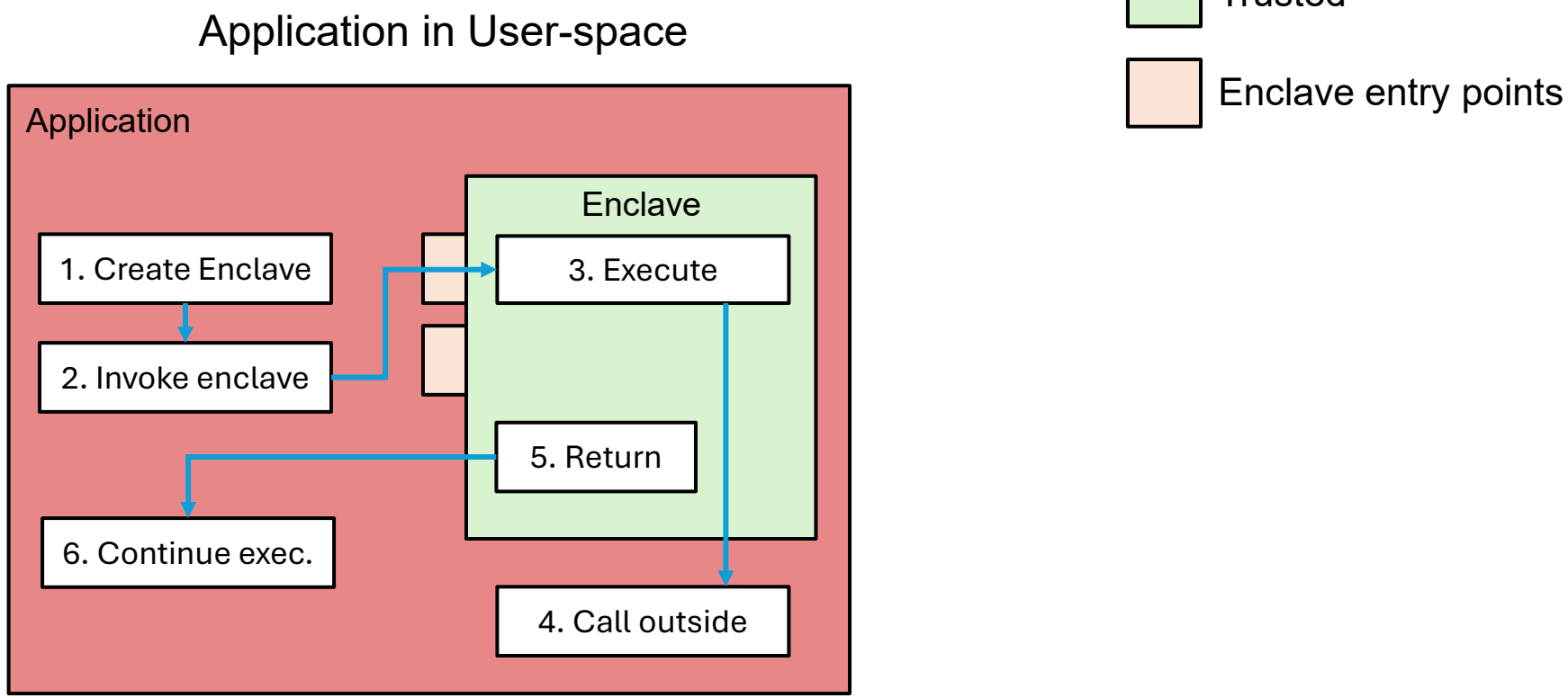
Flow within a process…



When the enclave function finishes, it safely returns to the application

Intel SGX Overview

Flow within a process…



Important:

- Untrusted code can create enclaves
- Including malicious ones
- Enclaves do not trust each other
- They should not share memory

When the enclave function finishes, it safely returns to the application

Intel SGX Architecture

Isolation in Intel SGX

Enclave life cycle

Memory Translation in SGX

Remote Attestation

Intel SGX Architecture

Isolation in Intel SGX

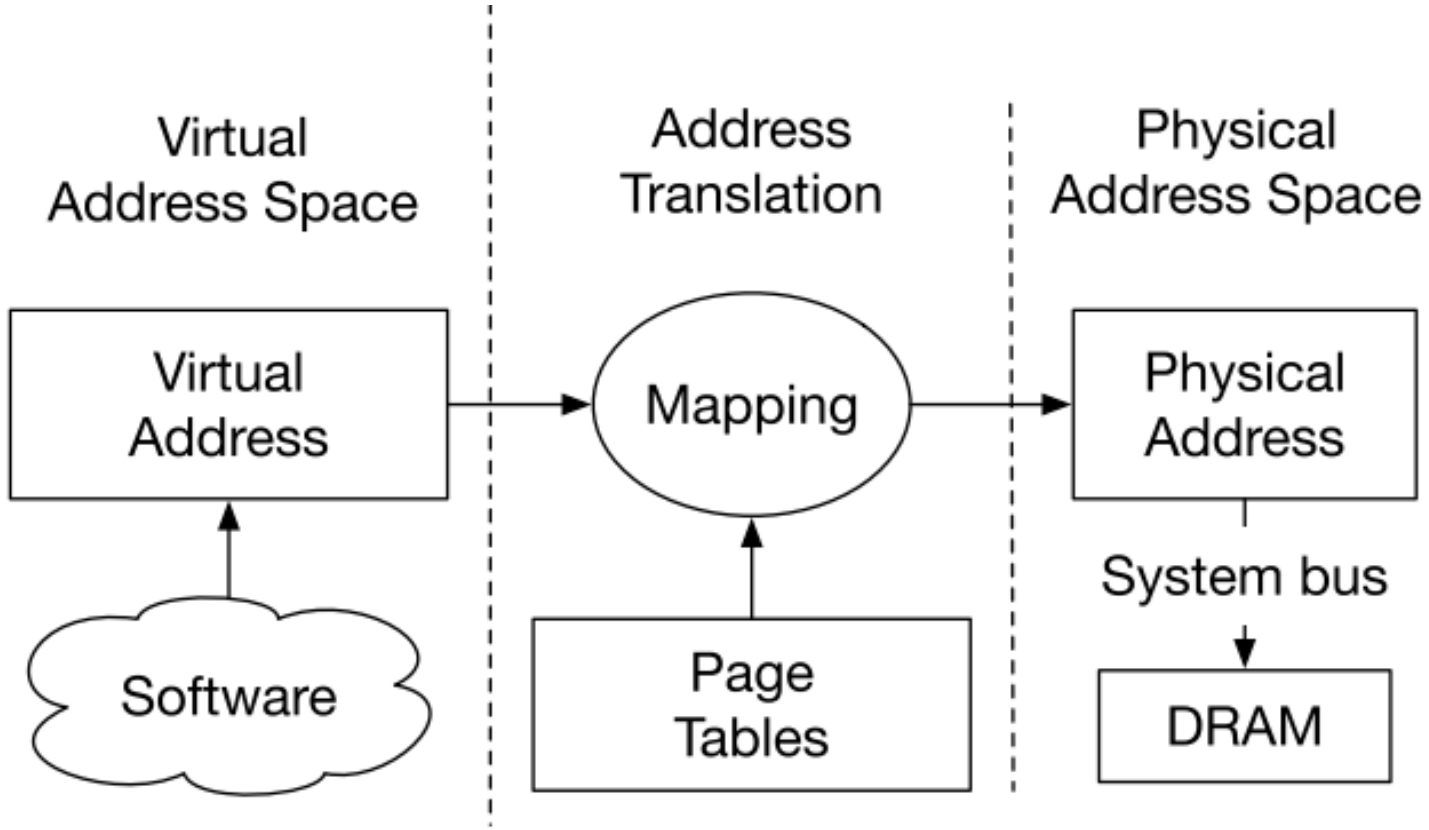
Enclave life cycle

Memory Translation in SGX

Remote Attestation

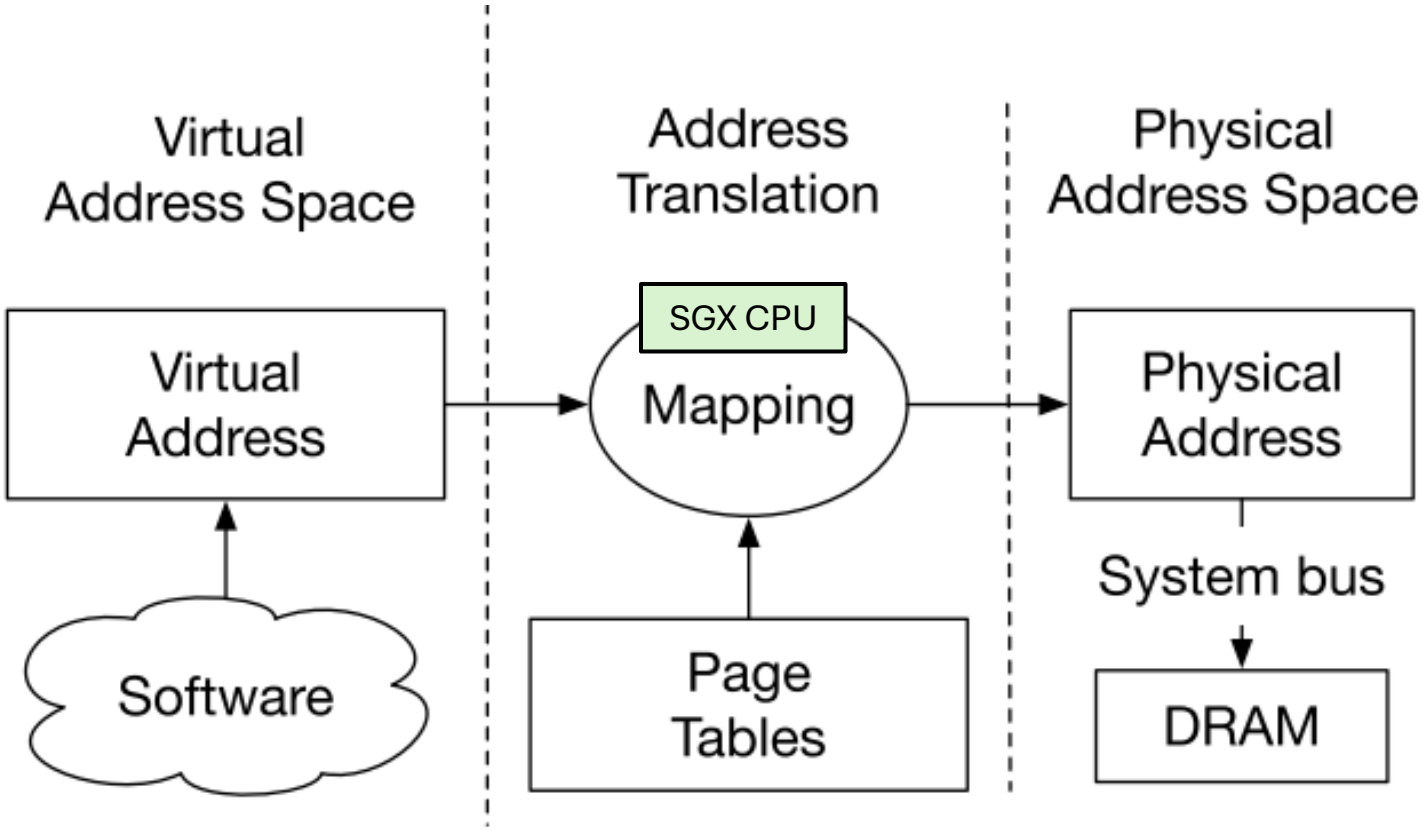
Intel SGX Architecture – Isolation

Key Architectural enablers:



Intel SGX Architecture – Isolation

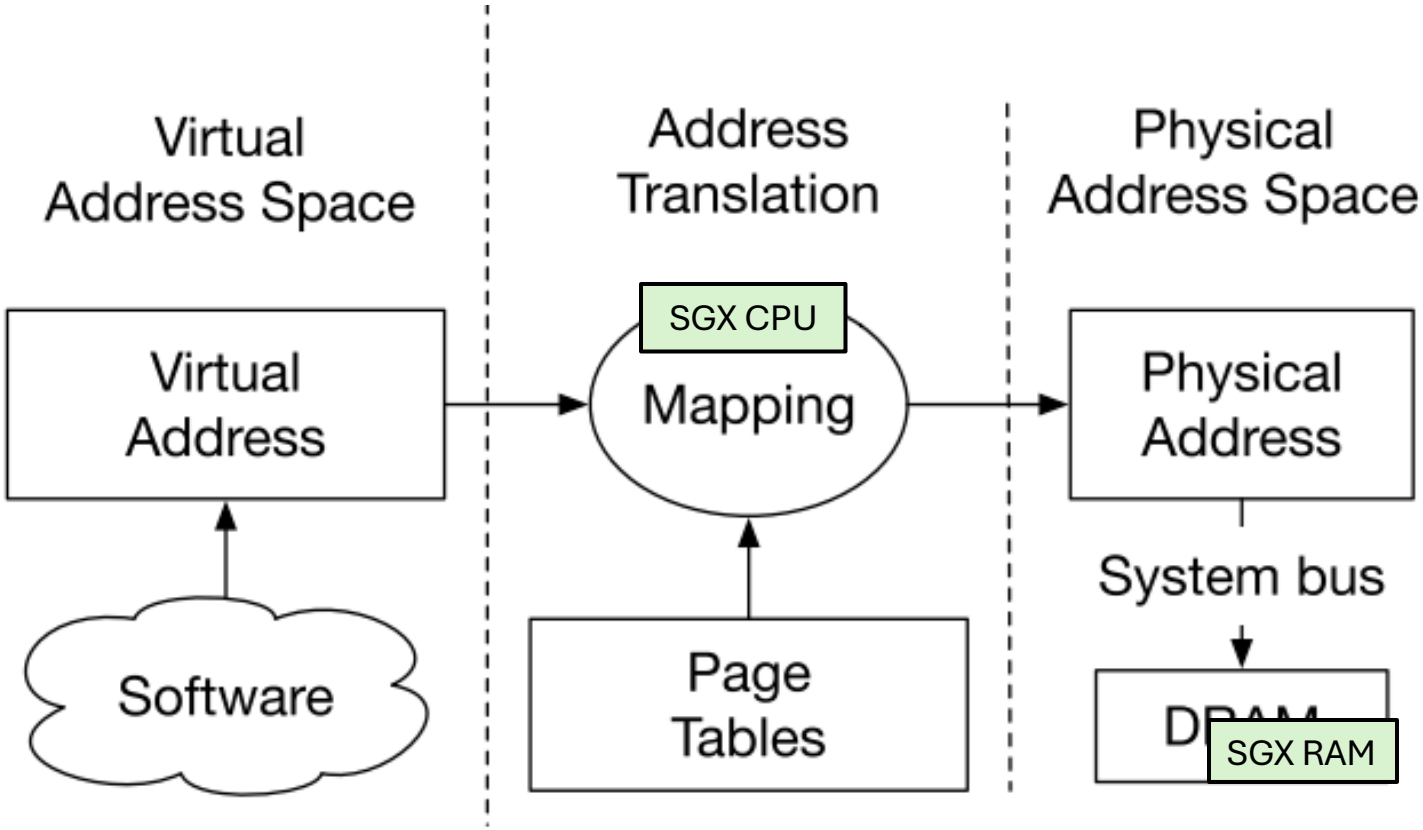
Key Architectural enablers:



When an enclave is running, memory checks are configured by the CPU hardware itself, (not the Rich OS)

Intel SGX Architecture – Isolation

Key Architectural enablers:



Reserved physical memory region for SGX enclaves (128 MB)

Intel SGX Architecture – Isolation

Processor reserved memory (PRM)

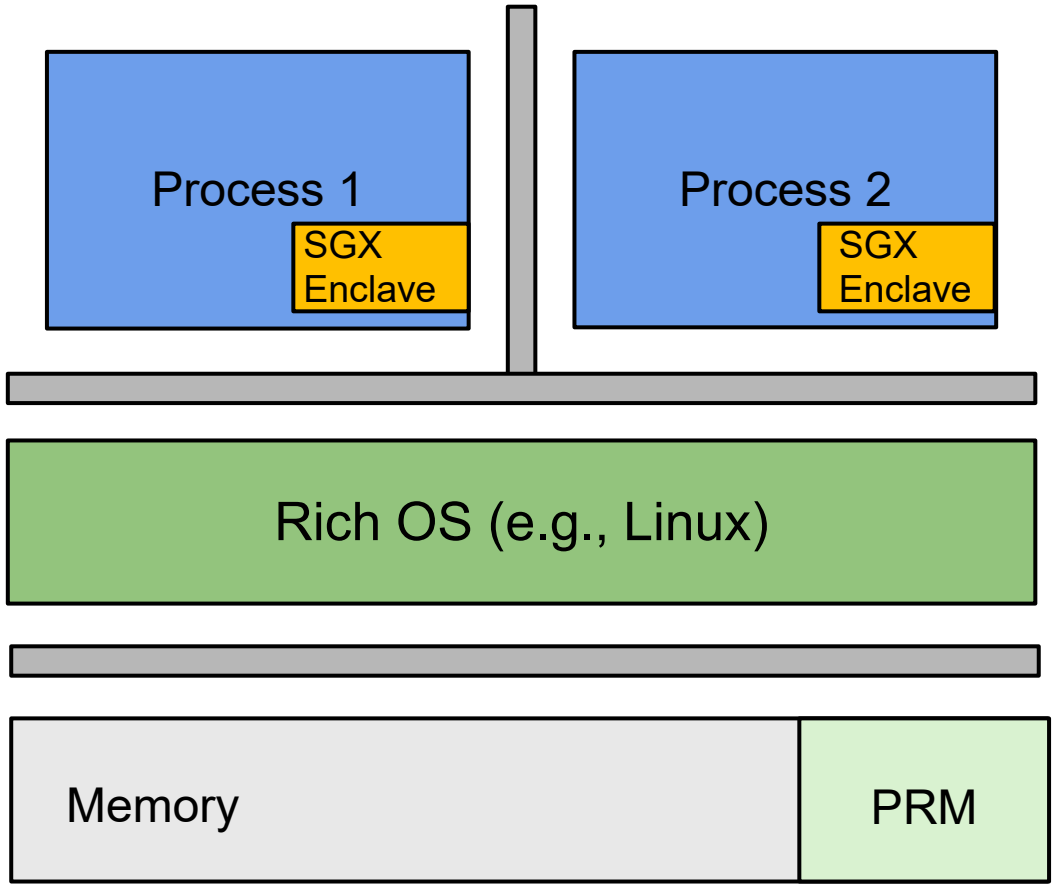
- 128 MB (SGX V1.0)
- Protected from accesses by CPU
- Divided into two parts

Enclave page cache (EPC):

- Stores enclave pages (code & data)

EPC Map (EPCM):

- Used to store enclave metadata
- Used for security checks



Intel SGX Architecture – Isolation

Processor reserved memory (PRM)

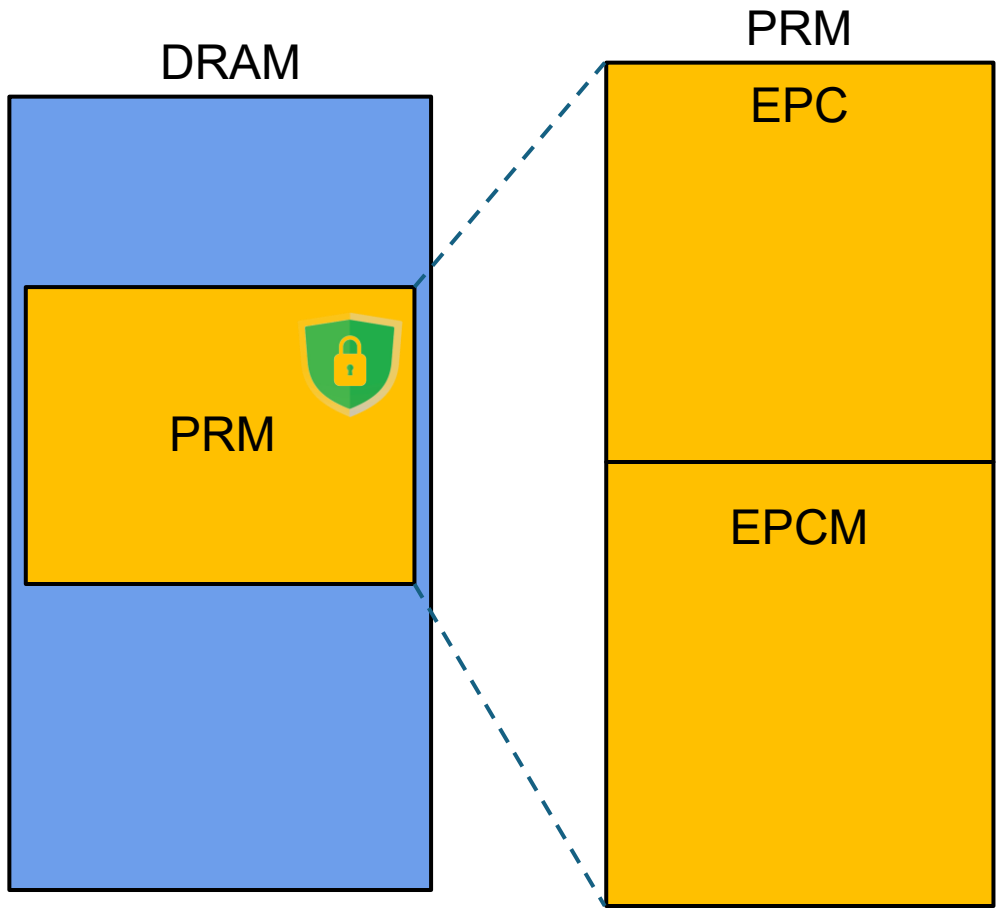
- 128 MB (SGX V1.0)
- Protected from accesses by CPU
- Divided into two parts

Enclave page cache (EPC):

- Stores enclave pages (code & data)

EPC Map (EPCM):

- Used to store enclave metadata
- Used for security checks



Intel SGX Architecture – Isolation

Processor reserved memory (PRM)

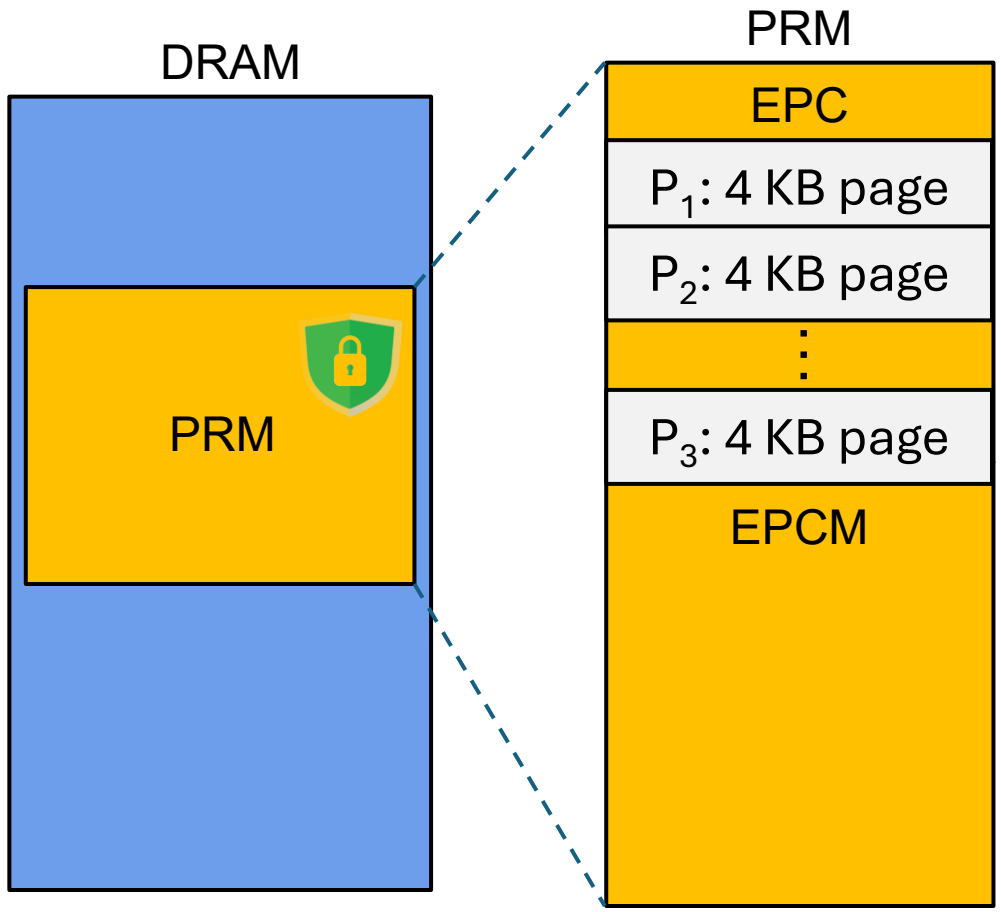
- 128 MB (SGX V1.0)
- Protected from accesses by CPU
- Divided into two parts

Enclave page cache (EPC):

- Stores enclave pages (code & data)

EPC Map (EPCM):

- Used to store enclave metadata
- Used for security checks



Intel SGX Architecture – Isolation

Processor reserved memory (PRM)

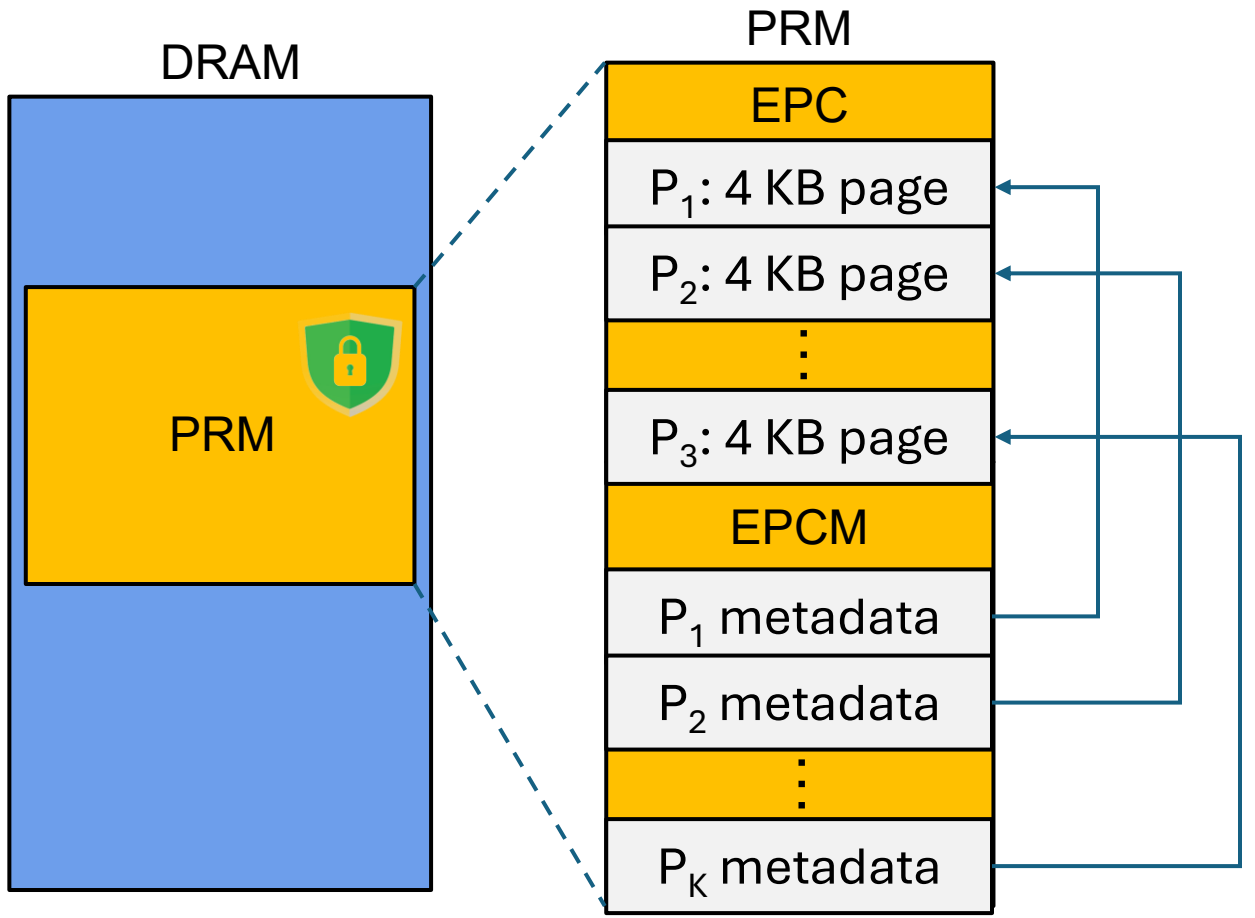
- 128 MB (SGX V1.0)
- Protected from accesses by CPU
- Divided into two parts

Enclave page cache (EPC):

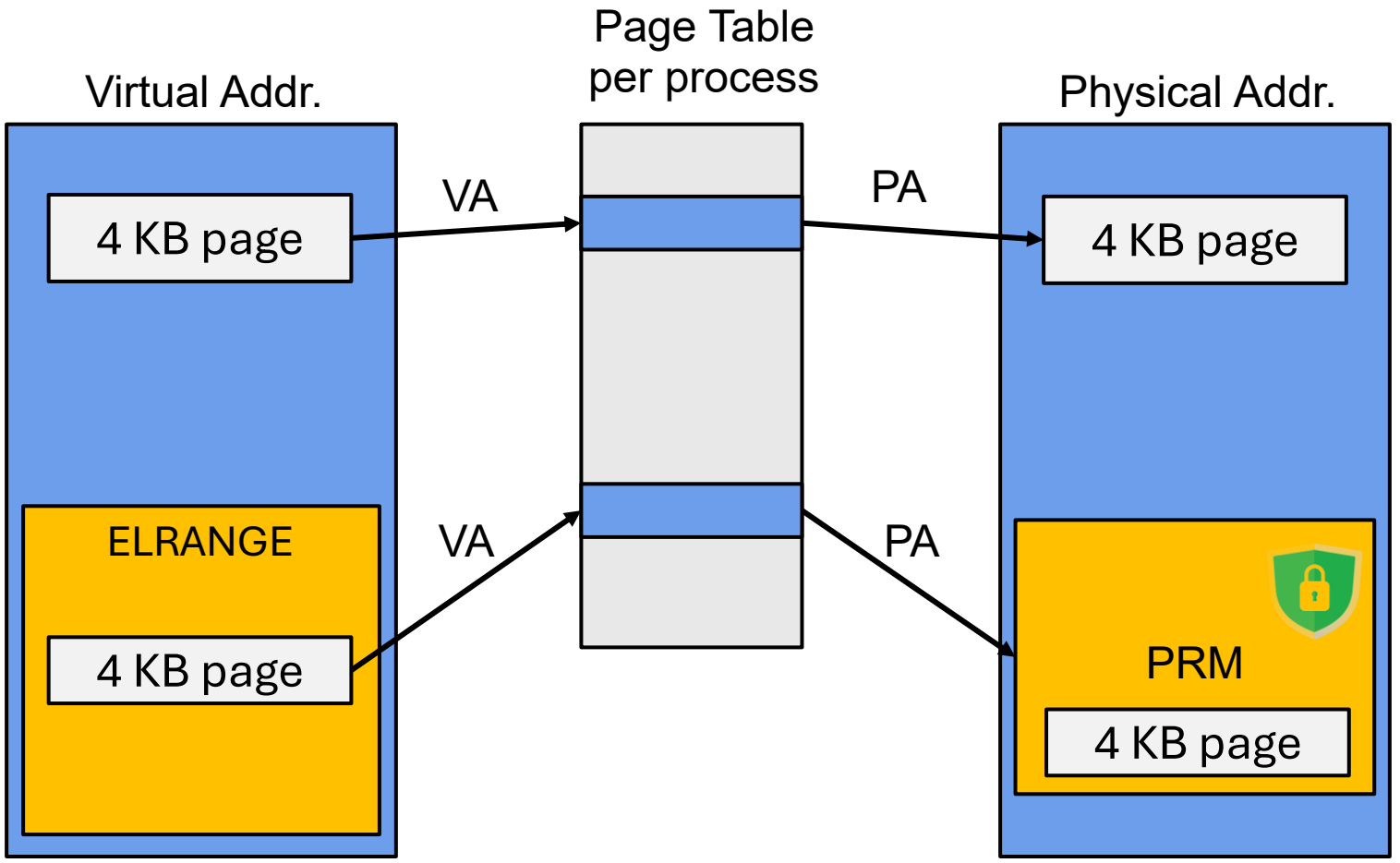
- Stores enclave pages (code & data)

EPC Map (EPCM):

- Used to store enclave metadata
- Used for security checks



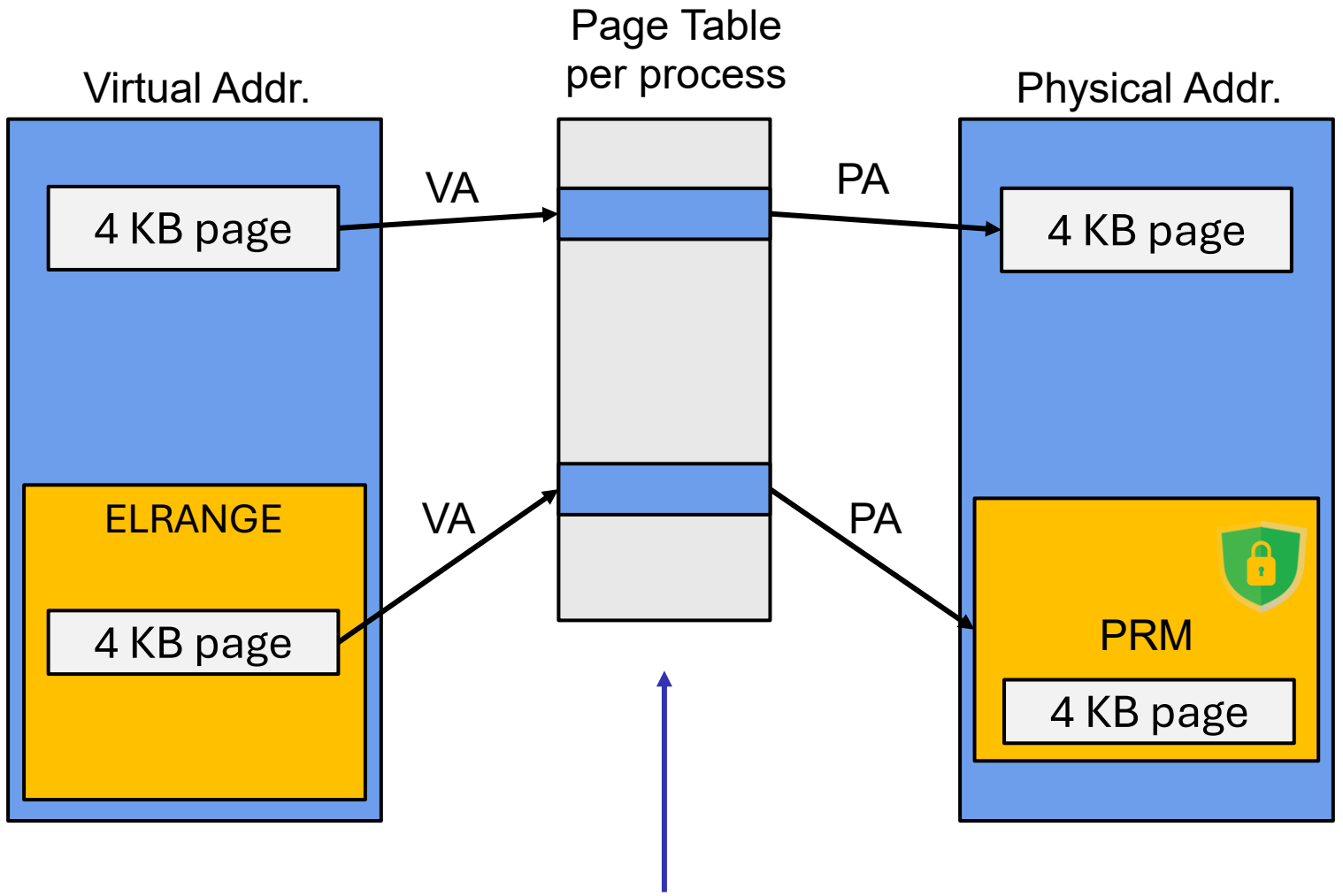
Intel SGX Architecture – Isolation



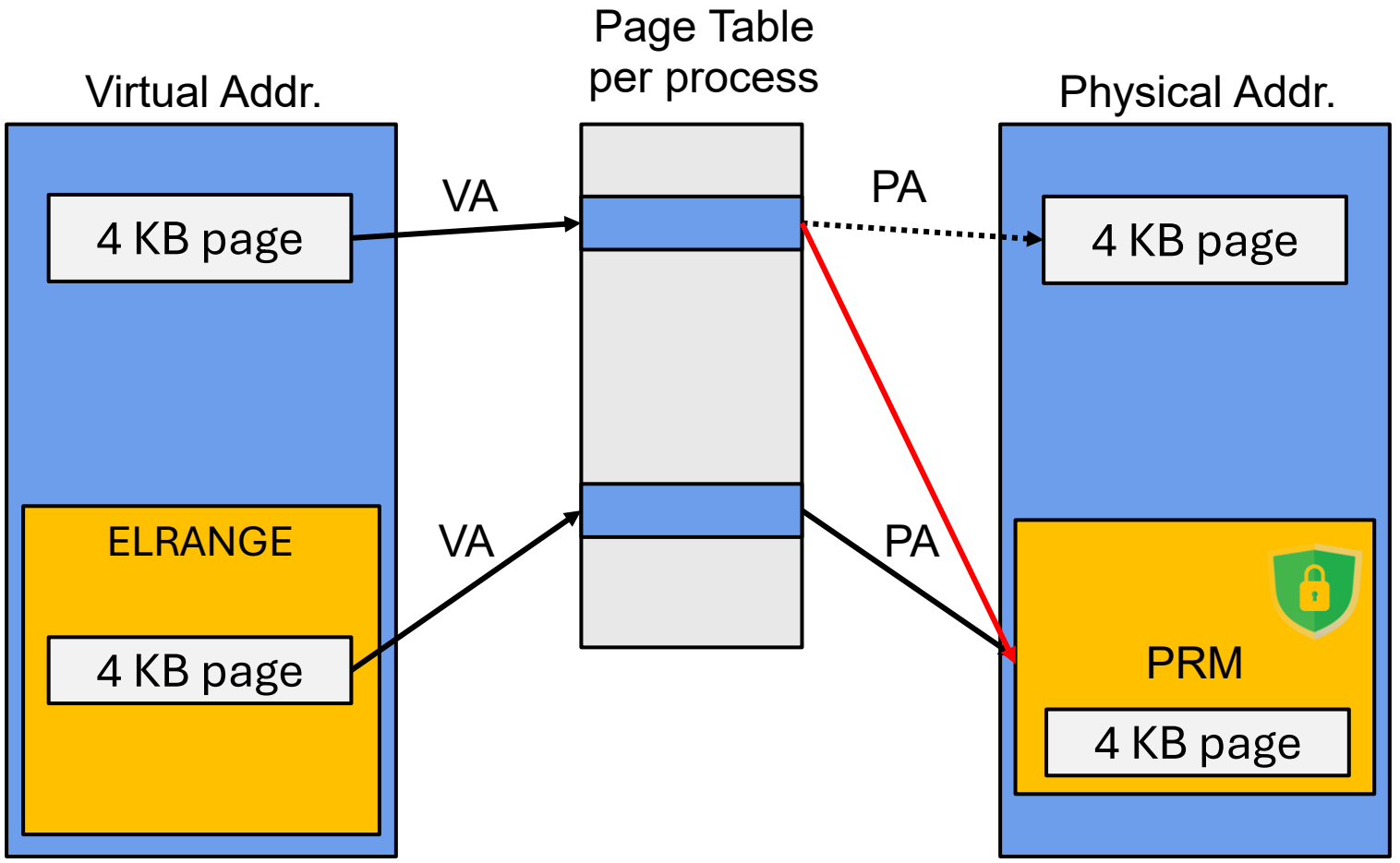
Enclave Linear Range (ELRANGE)

Virtual address range for an enclave

Intel SGX Architecture – Isolation



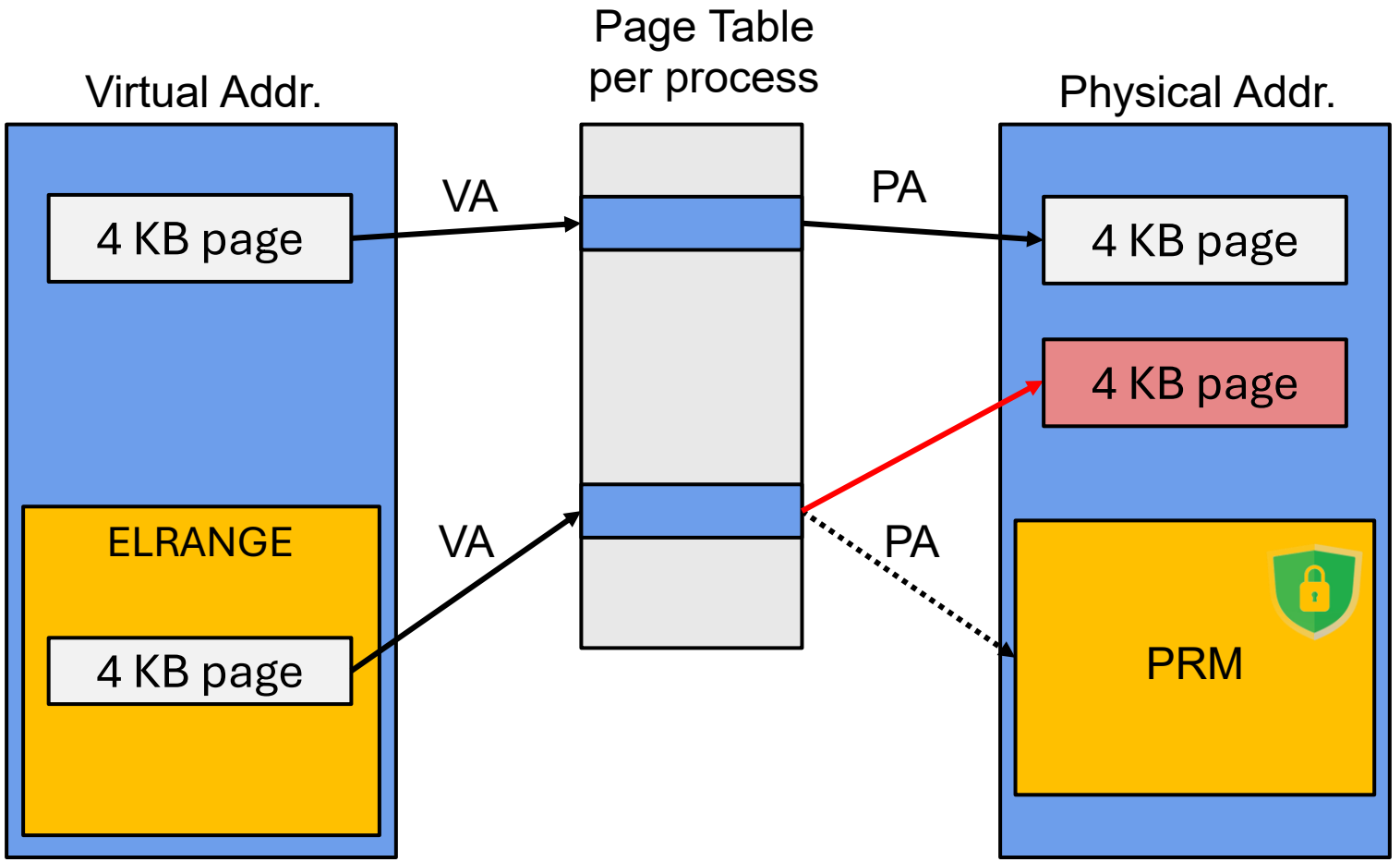
Intel SGX Architecture – Isolation



Malicious Address Translation (v1)

Illegal access via outside world

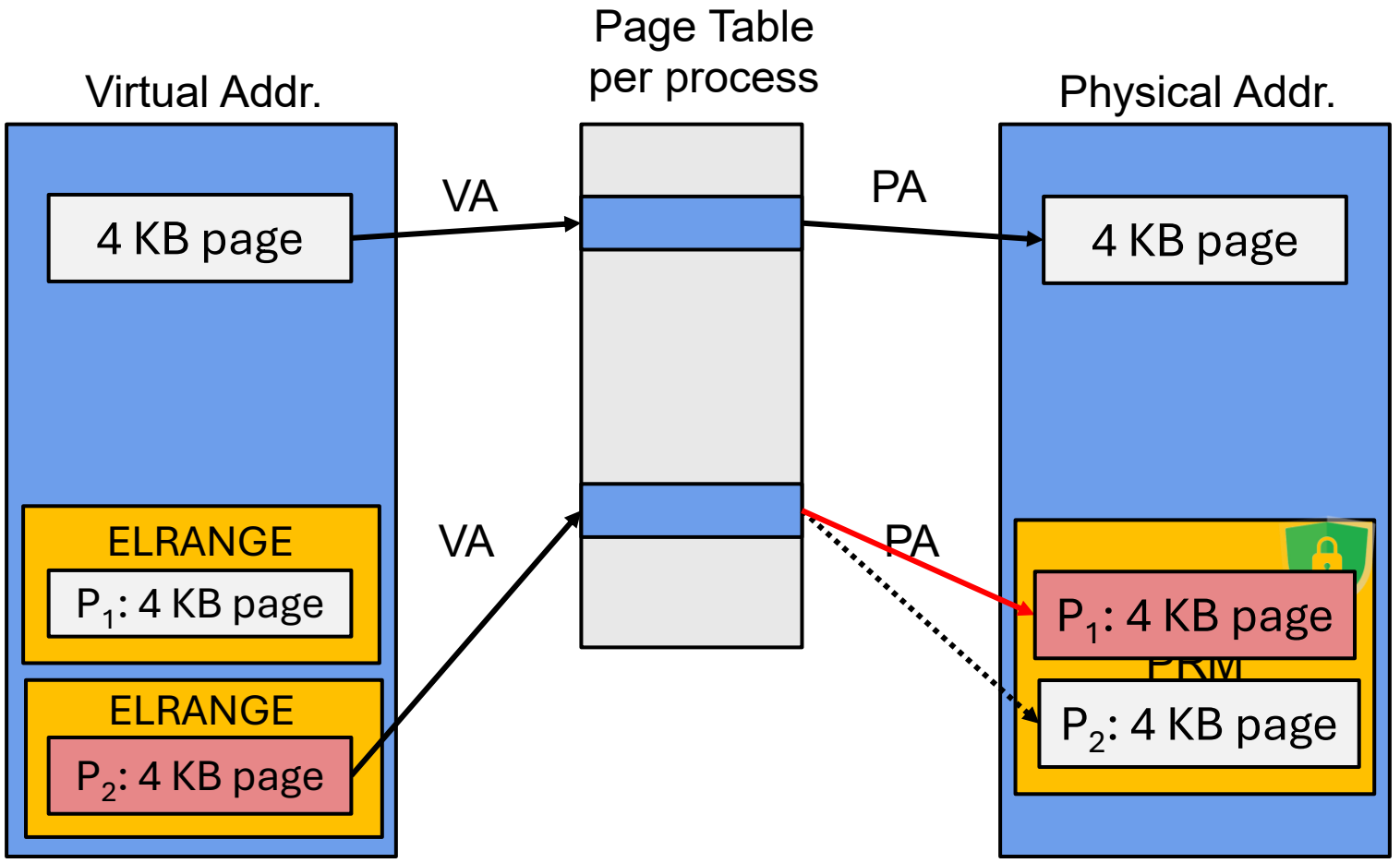
Intel SGX Architecture – Isolation



Malicious Address Translation (v2)

Information leak

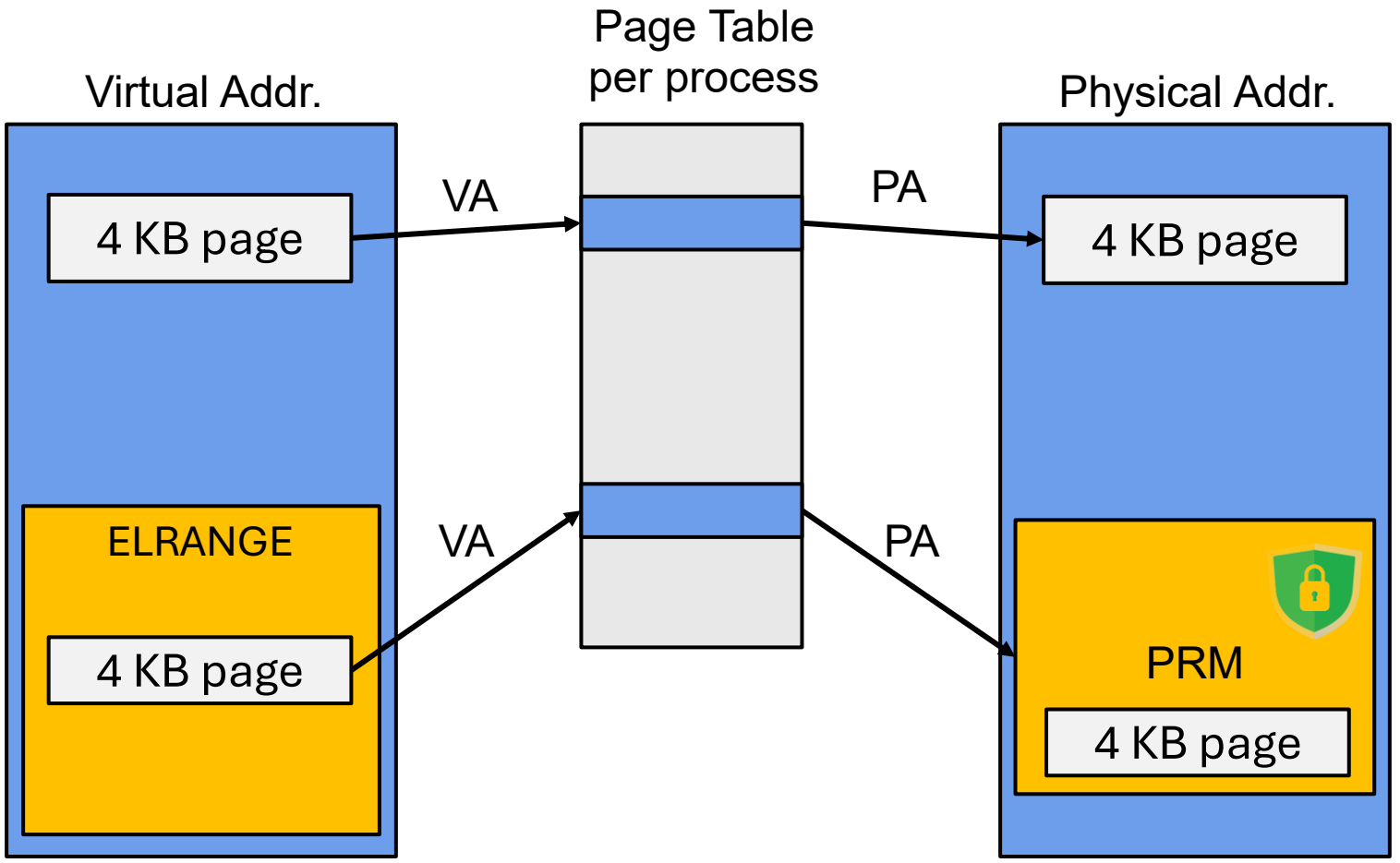
Intel SGX Architecture – Isolation



Malicious Address Translation (v3)

Illegal access via malicious enclave

Intel SGX Architecture – Isolation

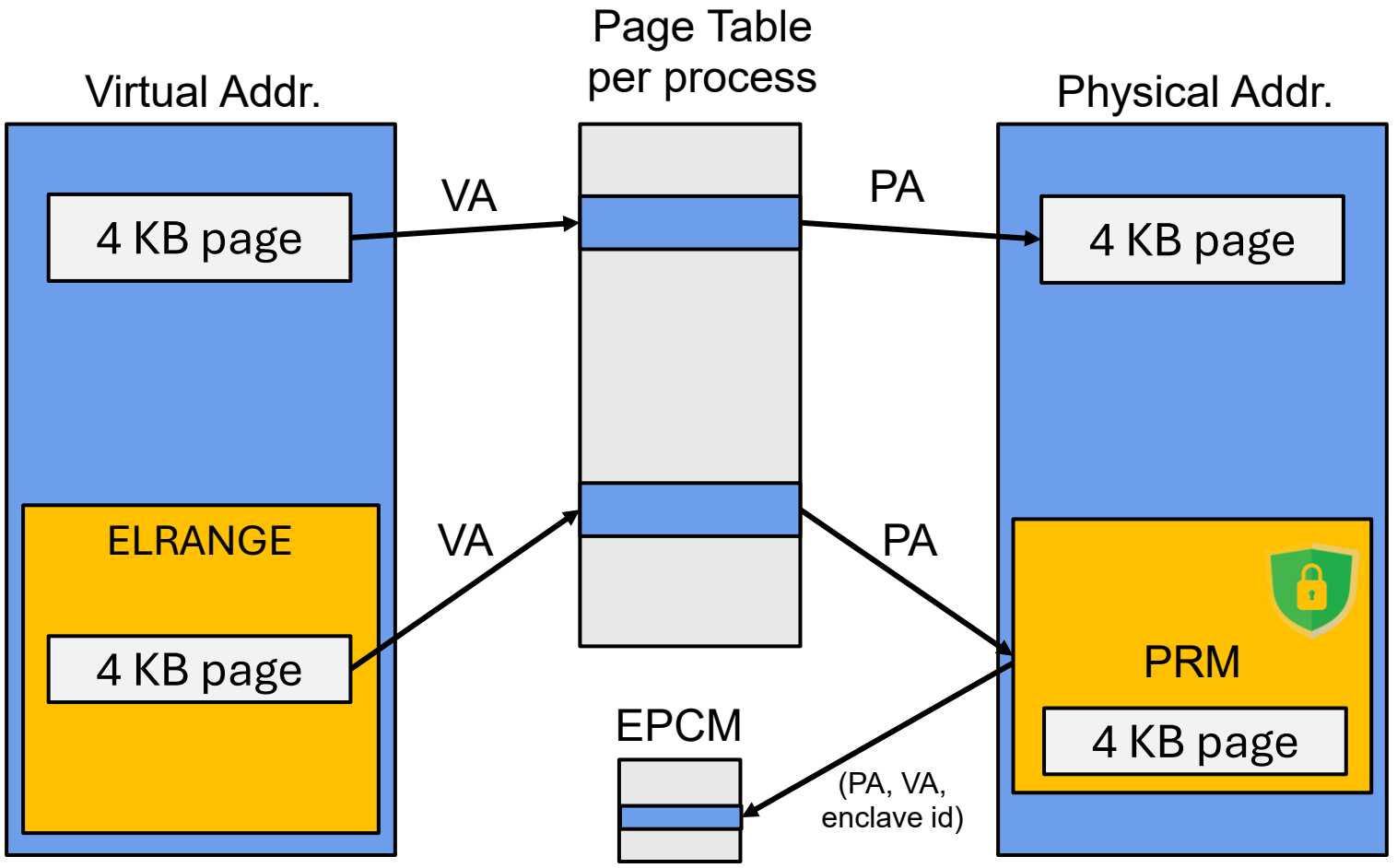


SGX Solution:

Keep page mapping metadata inside PRM (EPCM)

MMU performs additional check for specific enclave

Intel SGX Architecture – Isolation

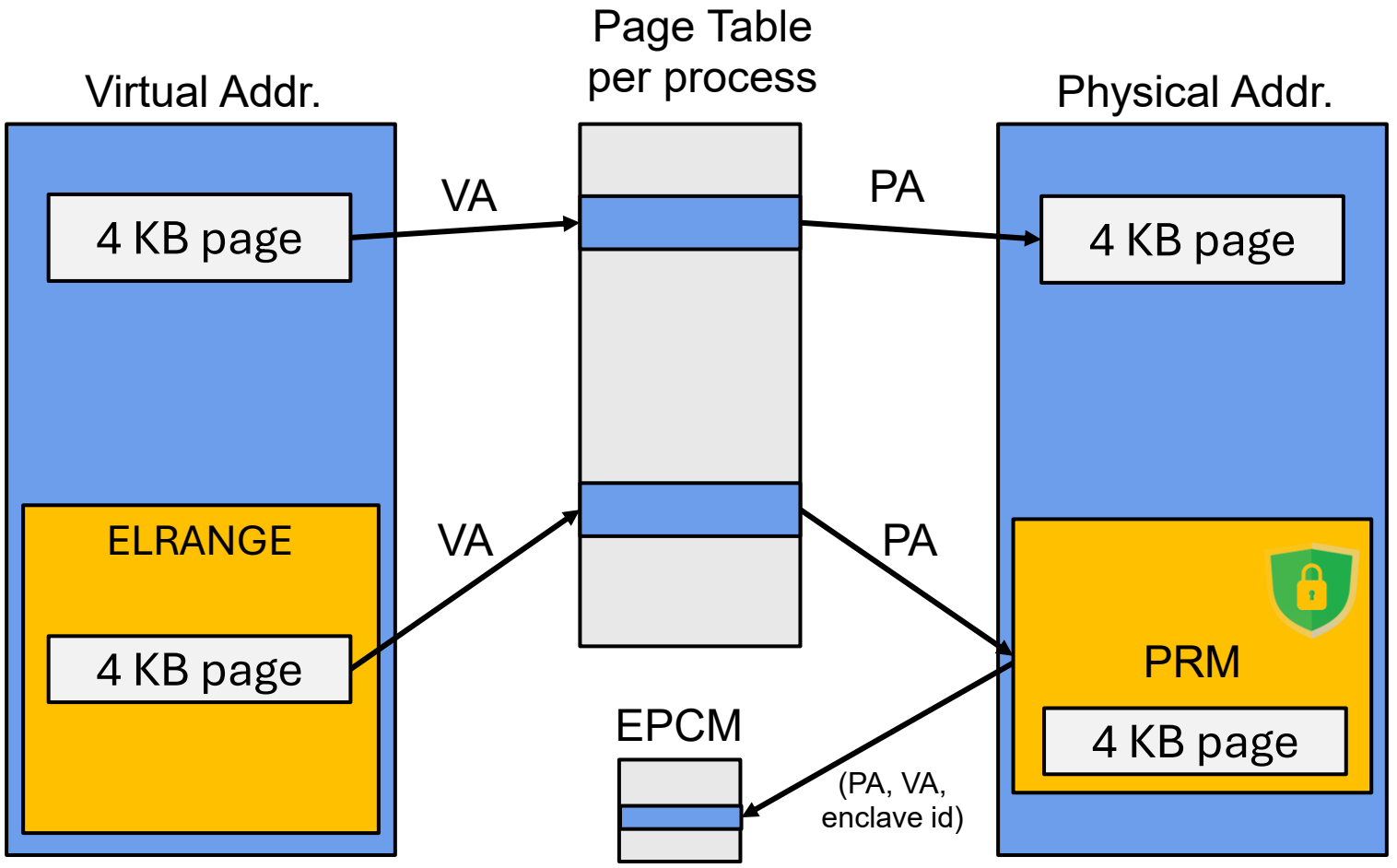


SGX Solution:

Keep page mapping metadata inside PRM (EPCM)

MMU performs additional check for specific enclave

Intel SGX Architecture – Isolation

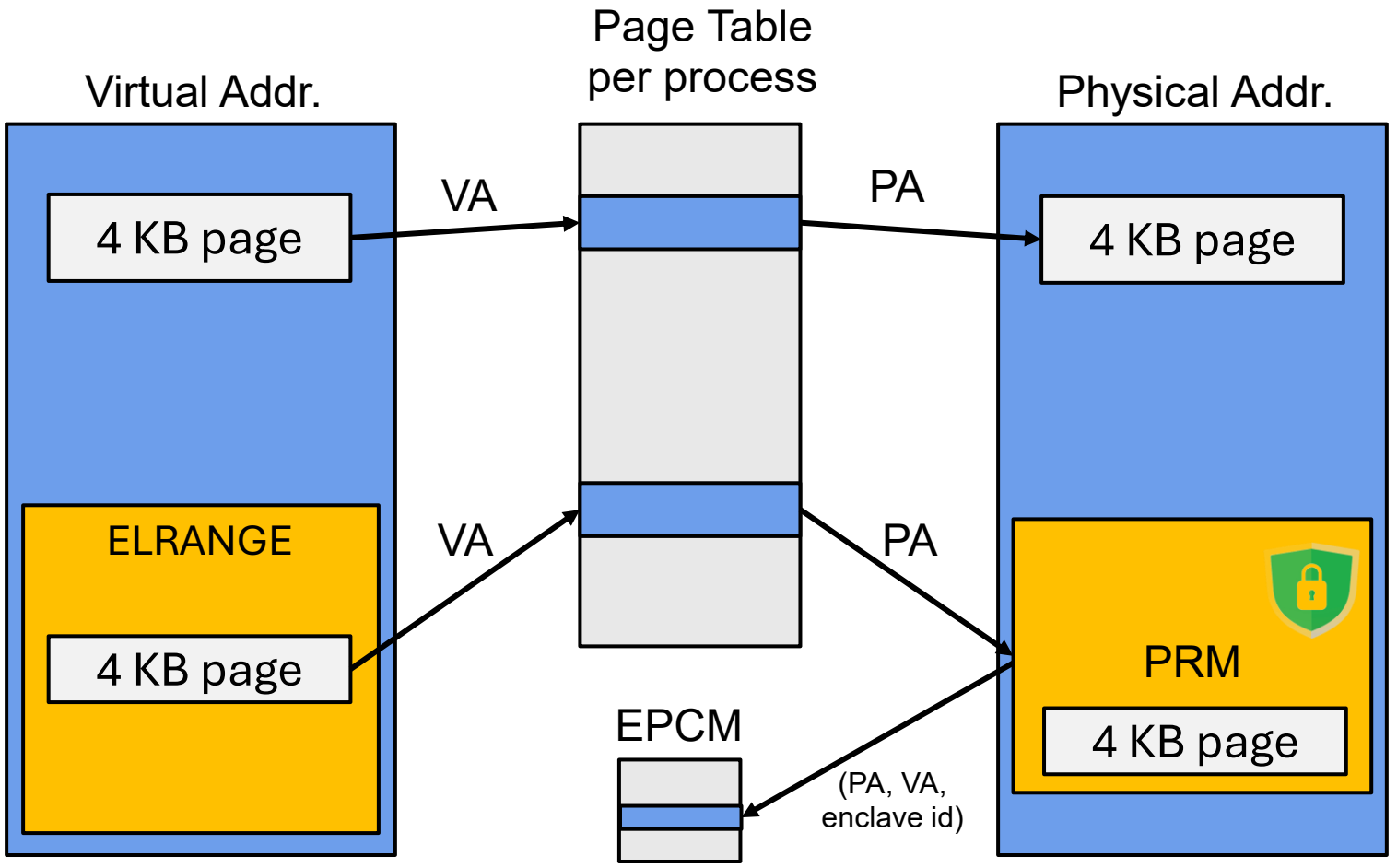


Important:

EPCM metadata is set by CPU hardware when enclave is initialized

It is not changed while enclave runs (only after it dies)

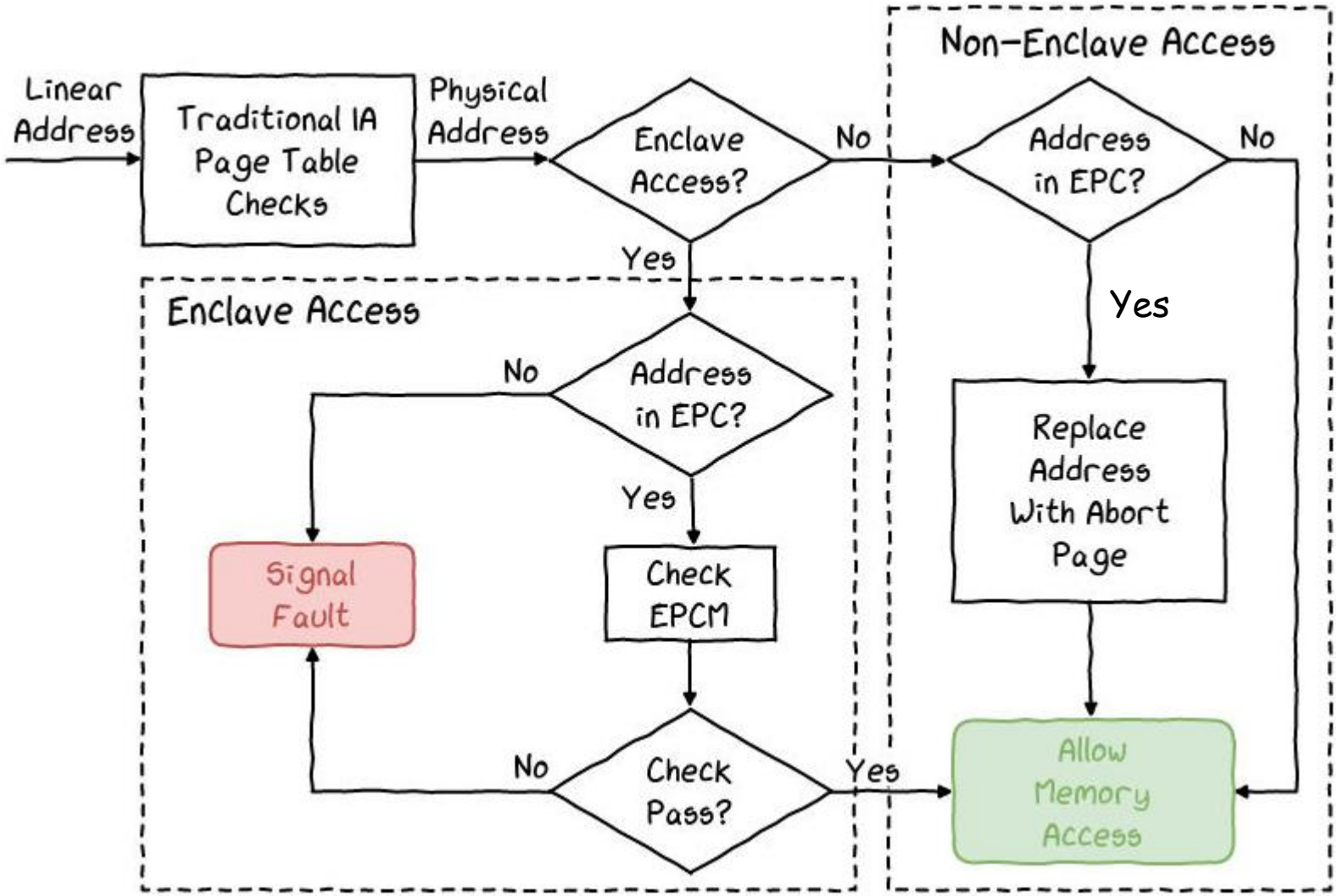
Intel SGX Architecture – Isolation



EPCM checked by each MMU translation

Intel SGX Architecture – Isolation

EPCM-MMU Checks as a whole:



Intel SGX Architecture

Isolation in Intel SGX

Enclave life cycle

Memory Translation in SGX

Remote Attestation

Intel SGX Architecture – Life Cycle

Enclave Initialization

SGX-specific instructions on Intel CPUs are used to support **creation** of an enclave within a process:

- **ECREATE** – establish memory address for an enclave
- **EADD** – copies memory pages into an enclave
- **EEXTEND** – computes a hash of the enclave contents
- **EINIT** – initializes the enclave

Intel SGX Architecture – Life Cycle

Post-creation control flow:

Dedicated functions to enter and exit from the enclave:

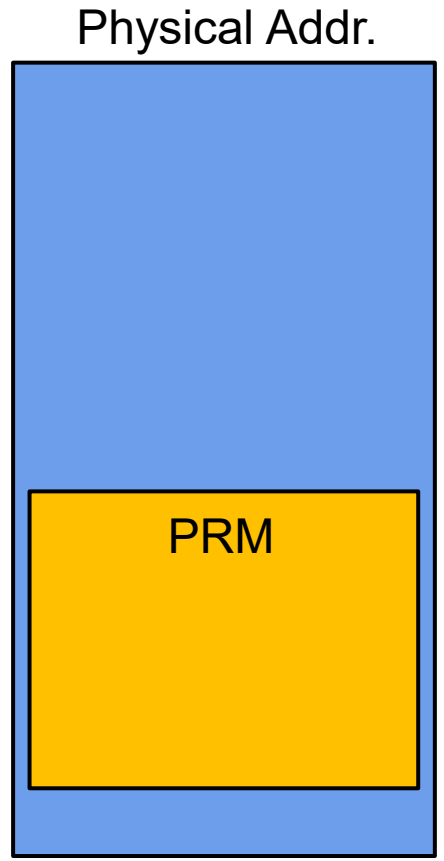
- **EENTER** – call a function inside the enclave
- **EEXIT** – return from enclave into untrusted region

Teardown

- **EREMOVE** – remove the enclave

Intel SGX Architecture – Life Cycle

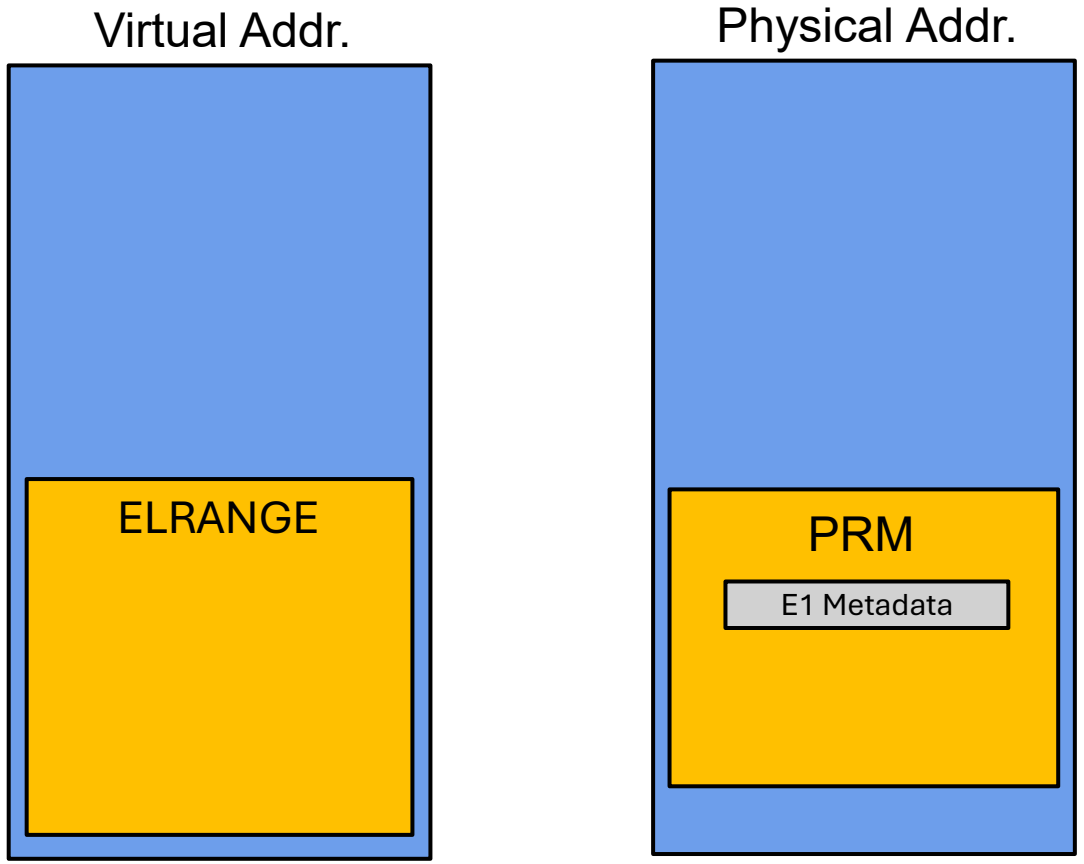
- BIOS sets up the PRM region
  - At boot time
  - Application starts creating an enclave



Intel SGX Architecture – Life Cycle

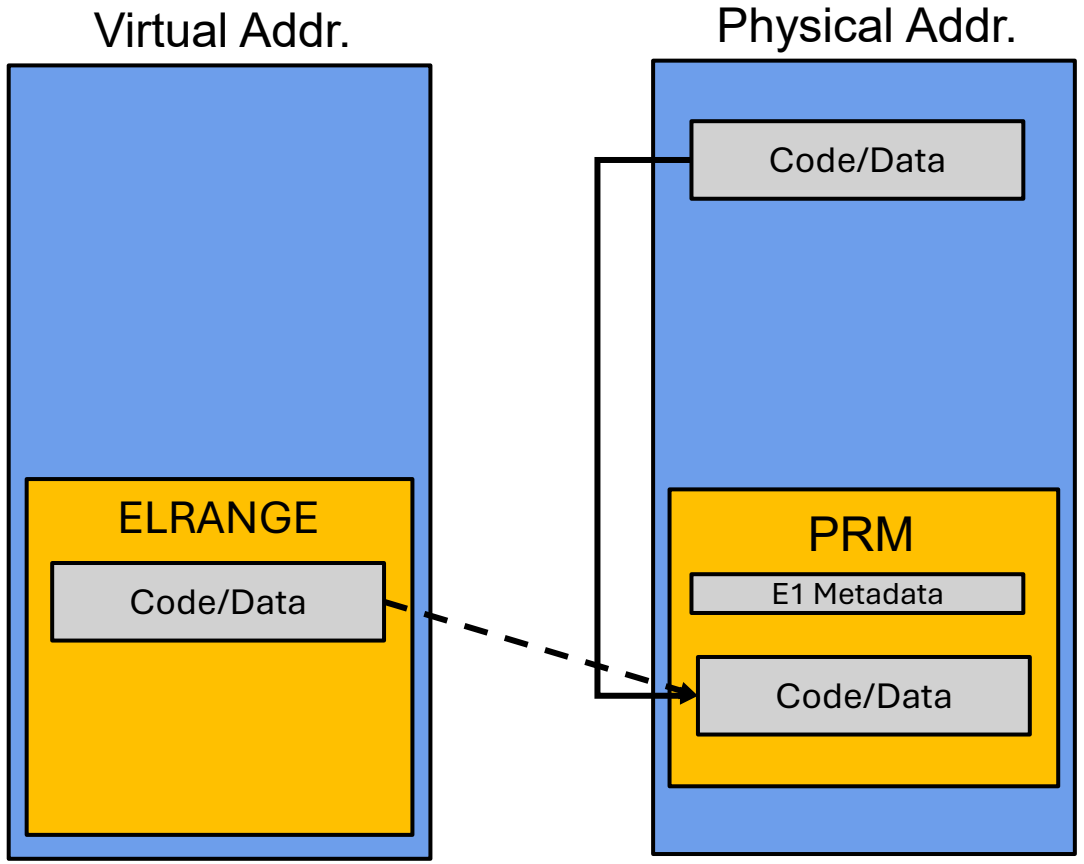
- BIOS sets up the PRM region
  - At boot time
  - Application starts creating an enclave

- **ECREATE**
  - Creates a memory address in PRM and assigns ELRANGE in virtual address space.



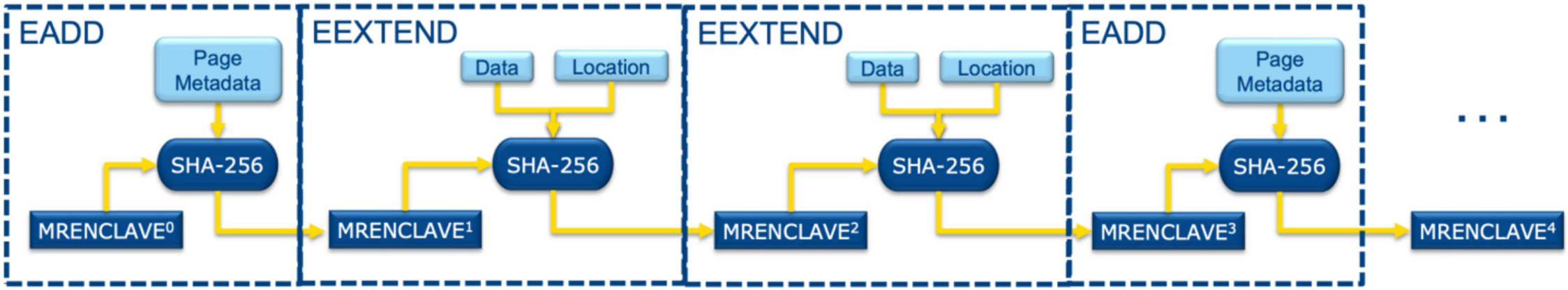
Intel SGX Architecture – Life Cycle

- **EADD**
  - Copy existing code/data into the PRM
  - Update mapping information
  - Repeat to add as many pages as needed into the EPC region
  - Each added page will update mapping info in EPCM entry accordingly
  - One EPCM entry per page
  - **All through hardware**



Intel SGX Architecture – Life Cycle

- **EEXTEND**
  - Optional step
  - Extends a hash-chain with added pages
  - Recorded into a dedicated internal register → MRENCLAVE
  - Sound familiar?

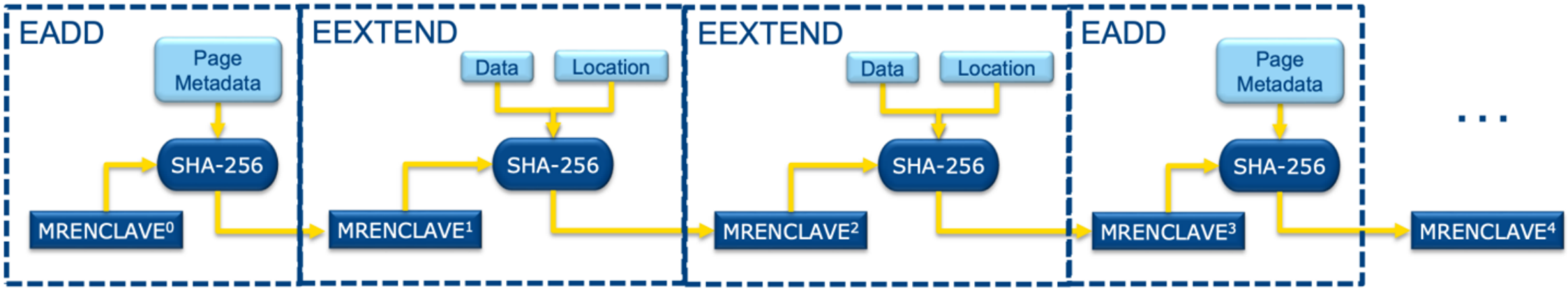


Intel SGX Architecture – Life Cycle

- **EEXTEND**
  - Optional step
  - Extends a hash-chain with added pages
  - Recorded into a dedicated internal register → MRENCLAVE
  - Sound familiar? Works just like *extend* in TPMs
  - Also includes location of extended data

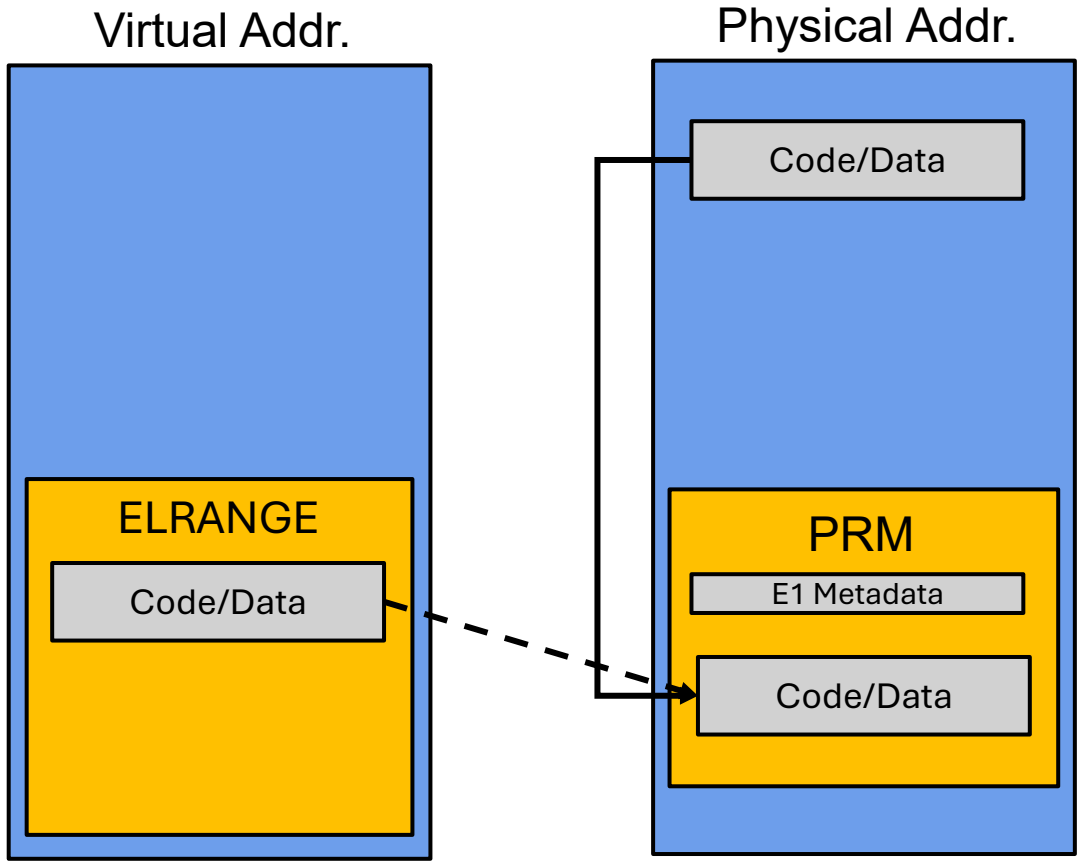
Works just like a PCR inside the TPM

One MRENCLAVE per enclave is in the PRM



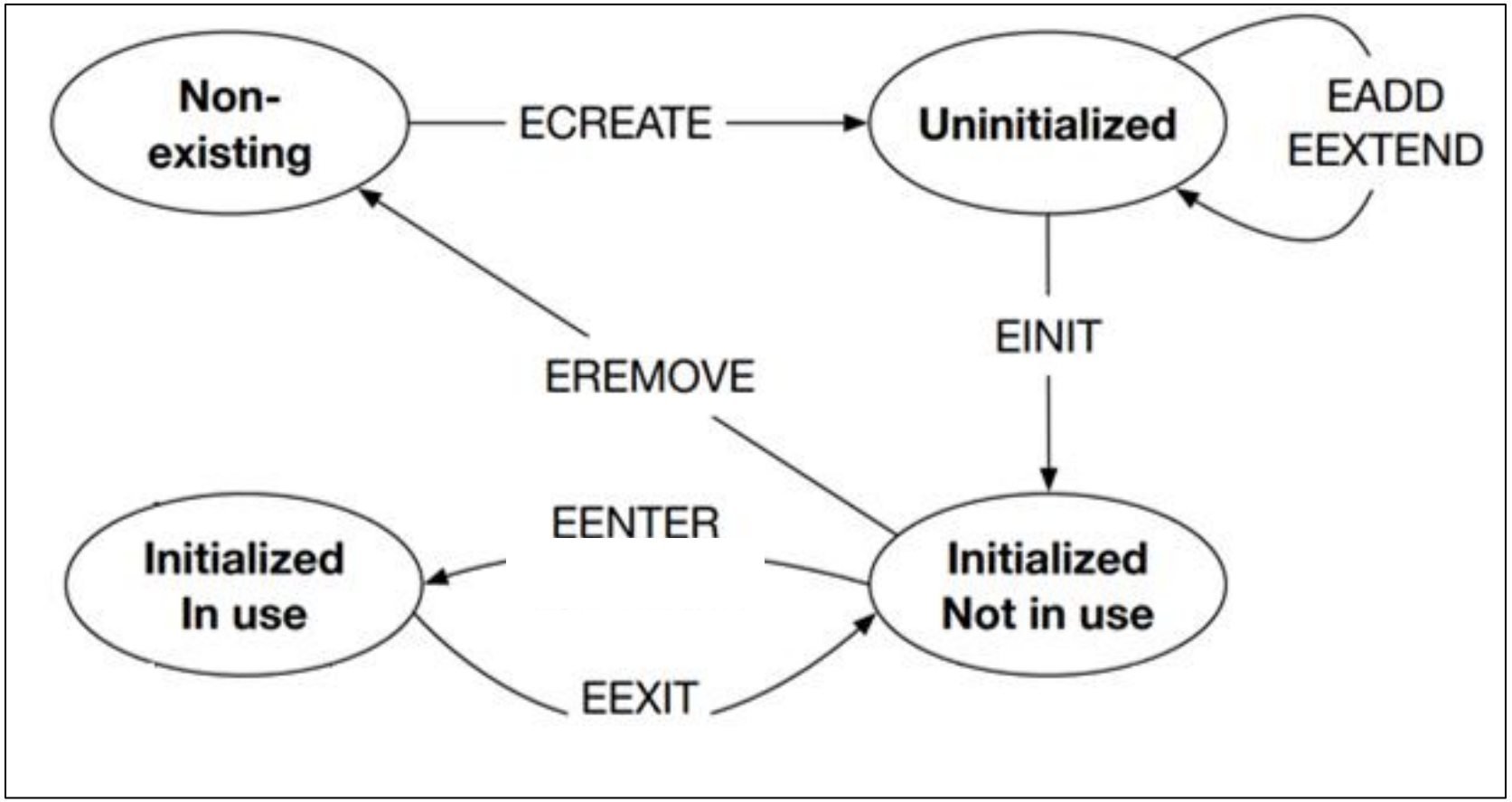
Intel SGX Architecture – Life Cycle

- **EINIT**
  - Initialize the enclave
  - Finalize the measurements made by EEXTEND
  - **Cannot call EADD, EEXTEND after EINIT**
- **EENTER**
  - Activate and enter the enclave
  - Switches to “enclave mode”
  - **Cannot call ENTER before calling EINIT**
- **EEXIT**
  - Exits enclave, switches to “normal mode”



Intel SGX Architecture – Isolation

All together:



Intel SGX Architecture

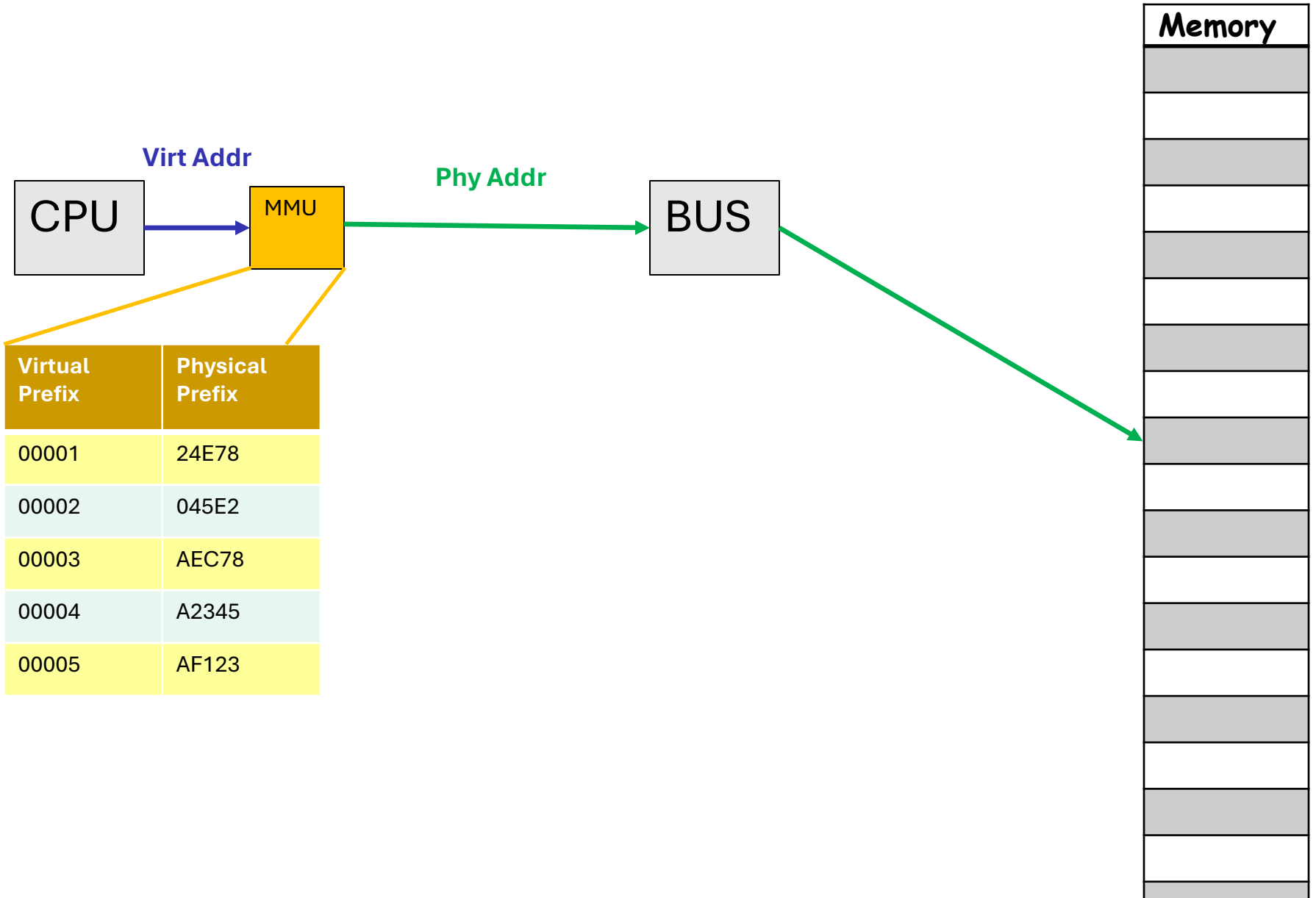
Isolation in Intel SGX

Enclave life cycle

Memory Translation in SGX

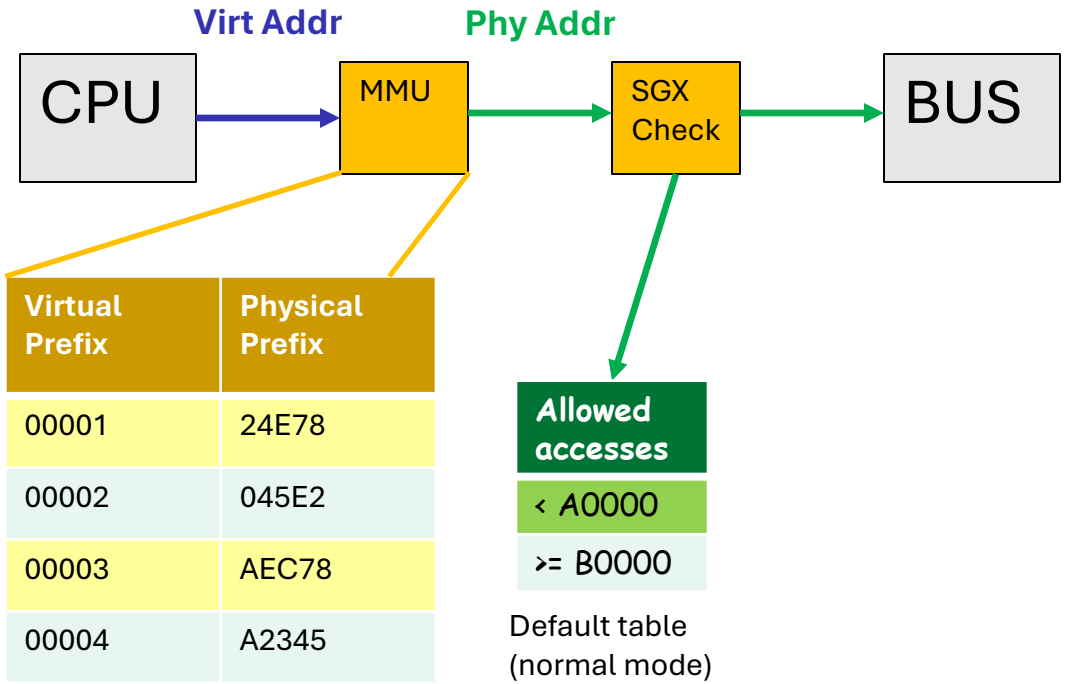
Remote Attestation

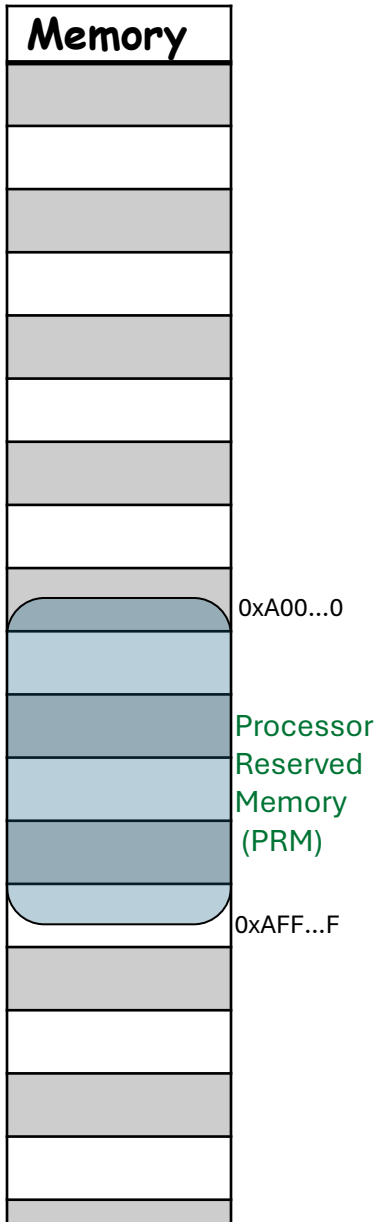
Intel SGX Architecture – Memory Translation



Virtual Prefix	Physical Prefix
00001	24E78
00002	045E2
00003	AEC78
00004	A2345
00005	AF123

Intel SGX Architecture – Memory Translation

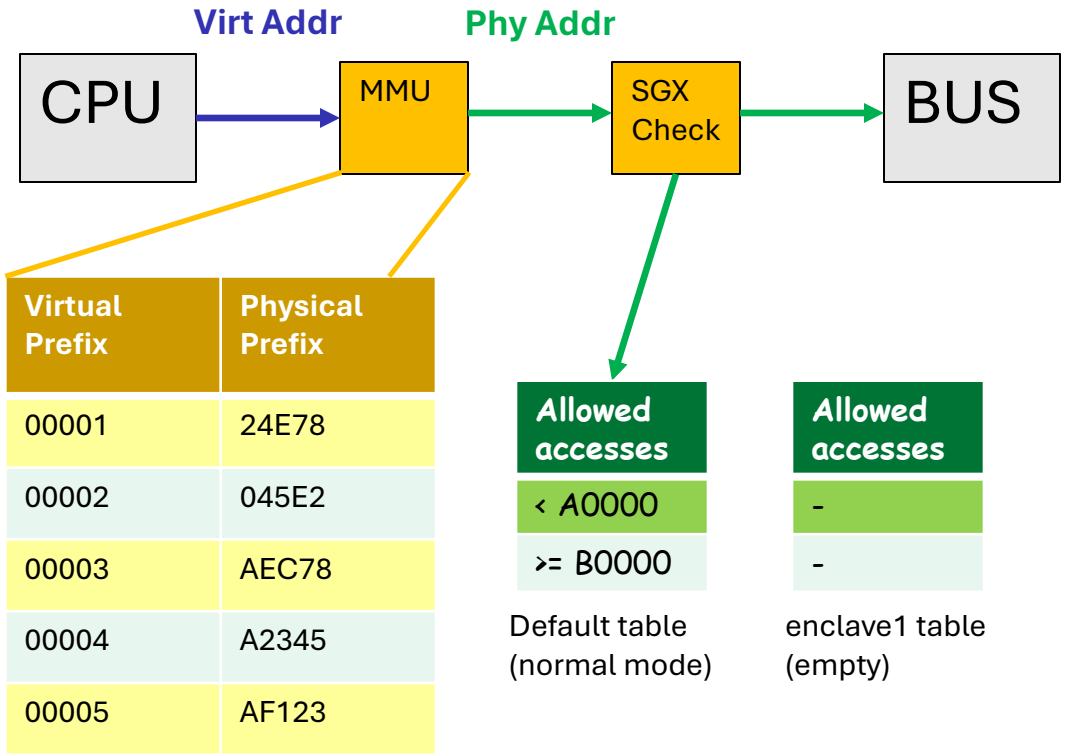




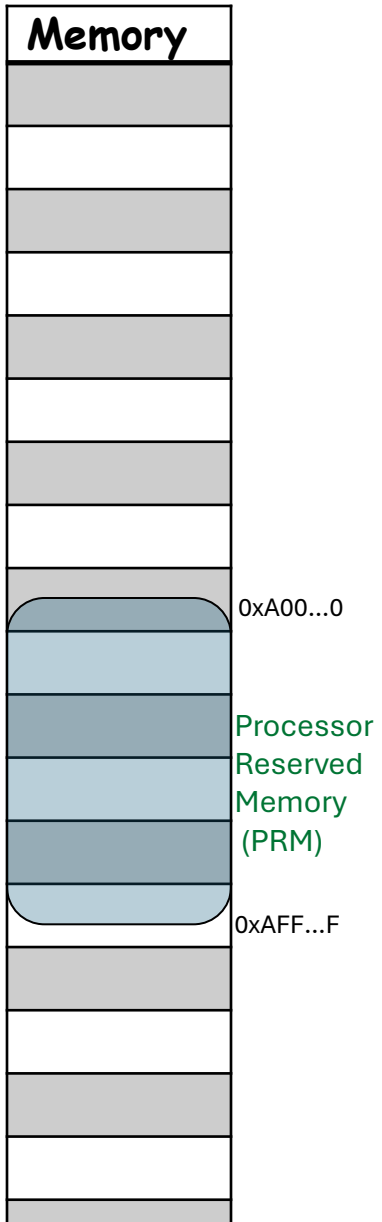
When running in normal mode (non-enclave)
=> prevent all software accesses to PRM.

Intel SGX Architecture – Memory Translation

ECREATE(enclave1);

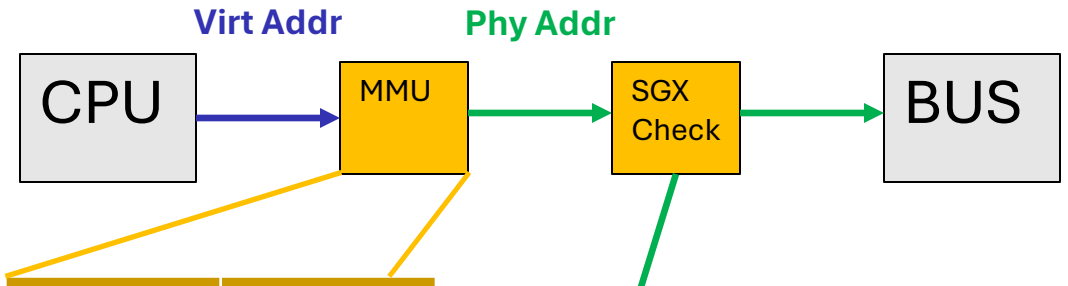


Create → create metadata entry for enclave



Intel SGX Architecture – Memory Translation

ECREATE(enclave1); ECREATE(enclave2);



Virtual Prefix	Physical Prefix
00001	24E78
00002	045E2
00003	AEC78
00004	A2345
00005	AF123

Allowed accesses
< A0000
>= B0000

Default table (normal mode)

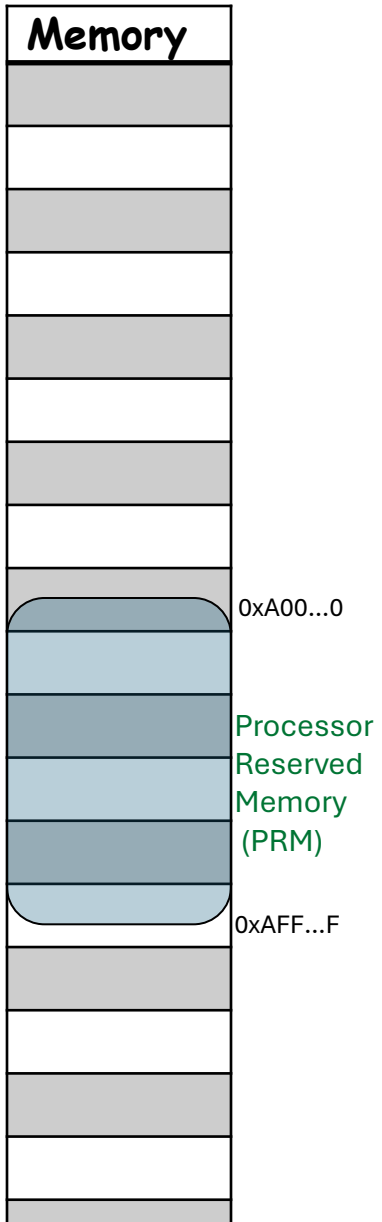
Allowed accesses
-
-

enclave1 table (empty)

Allowed accesses
-
-

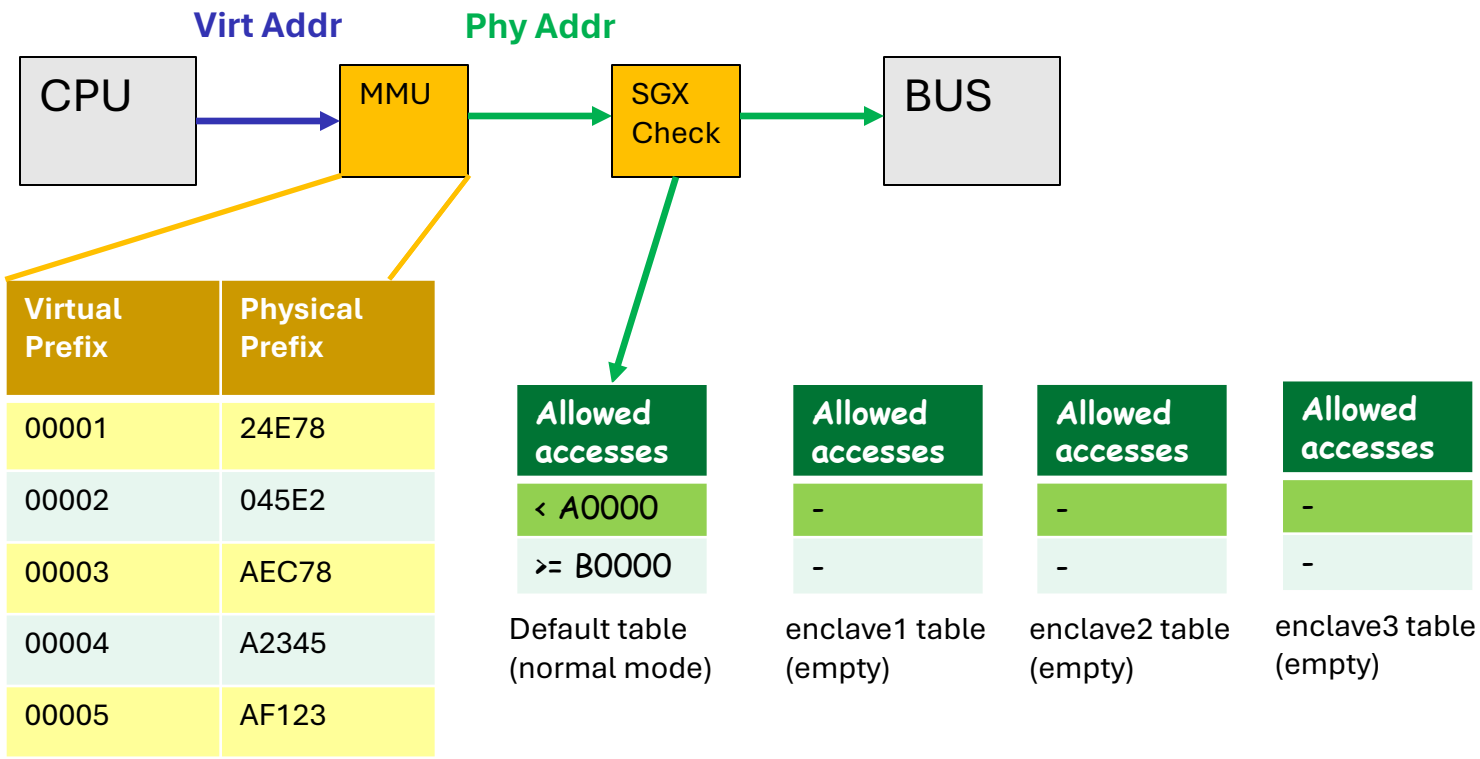
enclave2 table (empty)

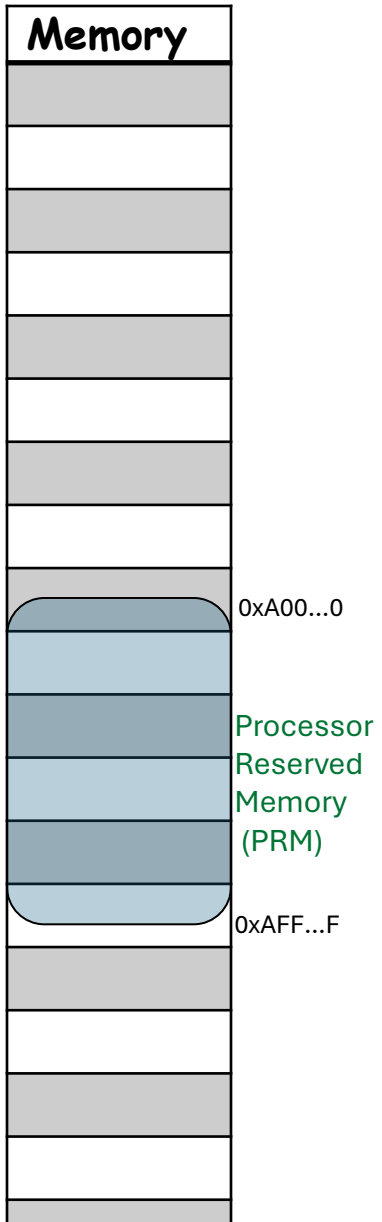
Create → create metadata entry for enclave



Intel SGX Architecture – Memory Translation

ECREATE(enclave1); ECREATE(enclave2); ECREATE(enclave3);





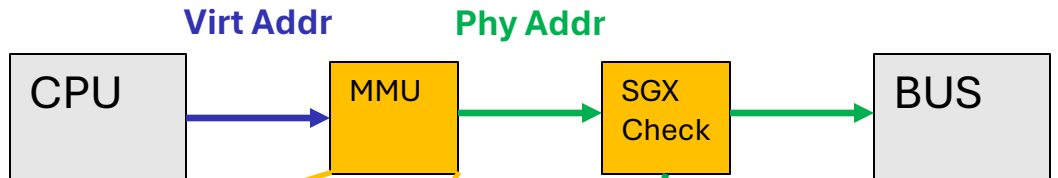
Create → create metadata entry for enclave

Stored within the PRM

"Allowed access table" → SGX Enclave Control Structure

Intel SGX Architecture – Memory Translation

EADD(enclave2, <pageA>)



Virtual Prefix	Physical Prefix
00001	24E78
00002	045E2
00003	AEC78
00004	A2345
00005	AF123

Allowed accesses
< A0000
>= B0000

Default table (normal mode)

Allowed accesses
-
-

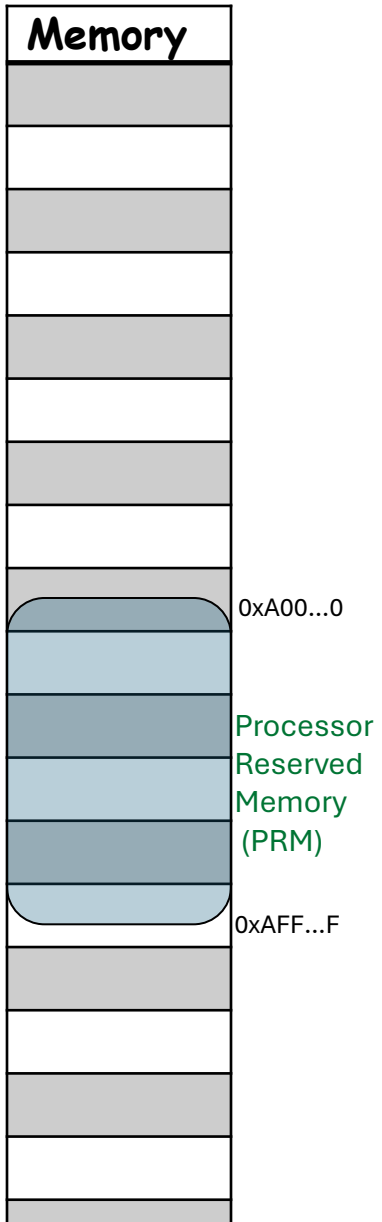
enclave1 table (empty)

Allowed accesses
-
-

enclave2 table (empty)

Allowed accesses
-
-

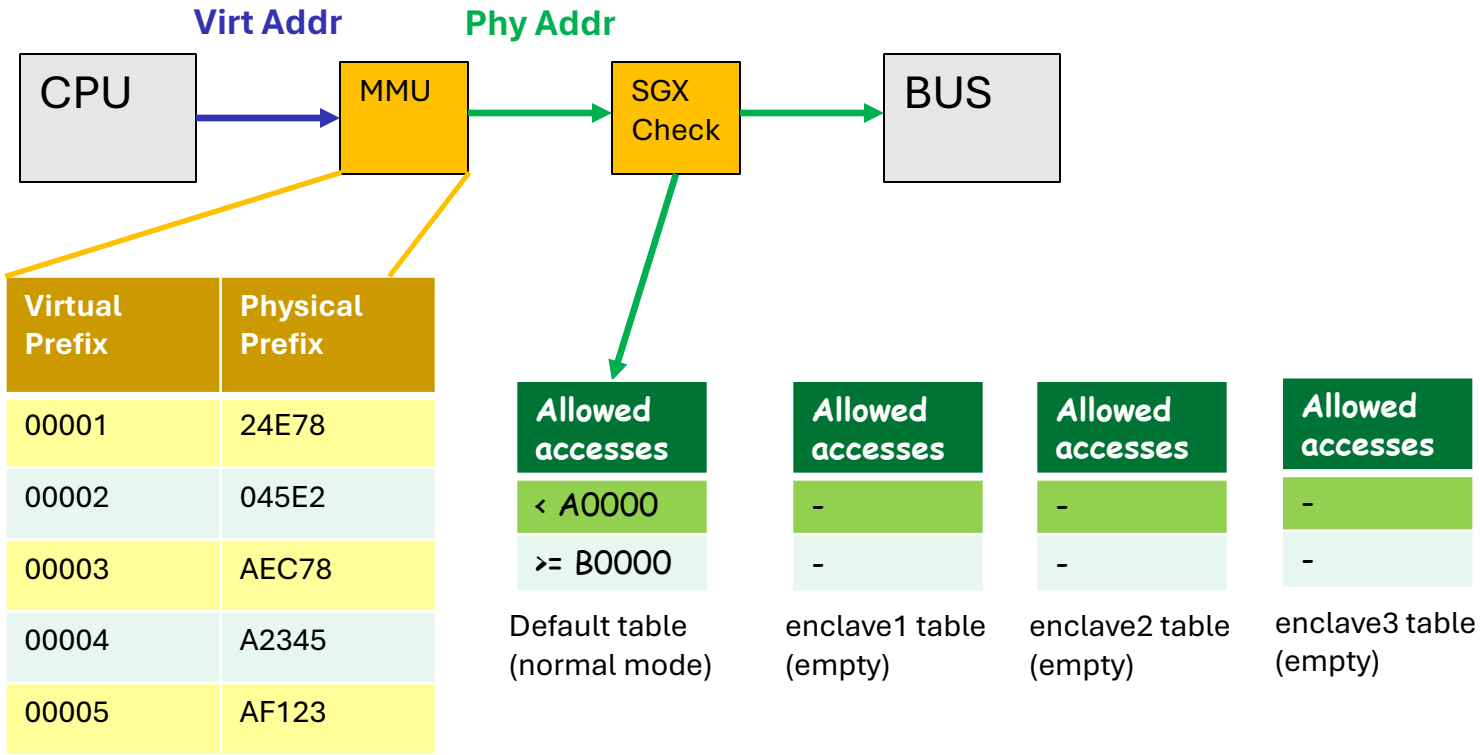
enclave3 table (empty)



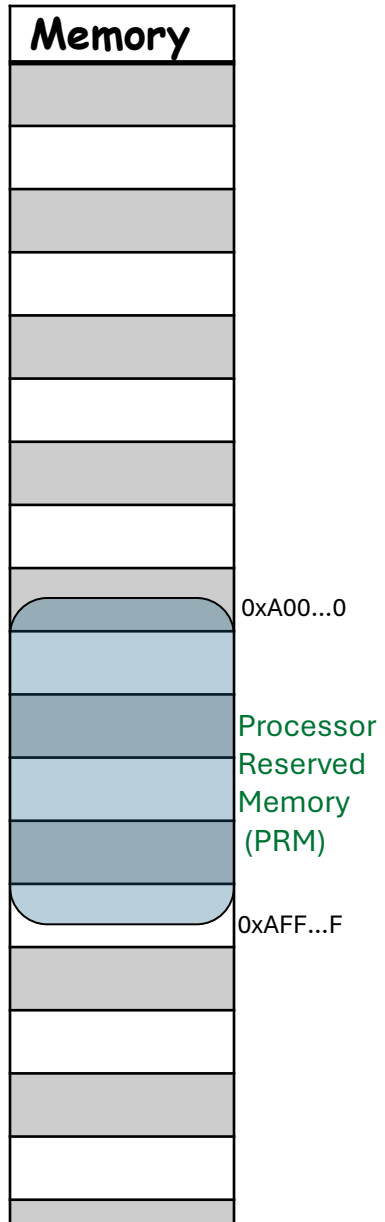
What happens now?

Intel SGX Architecture – Memory Translation

EADD(enclave2, <pageA>)

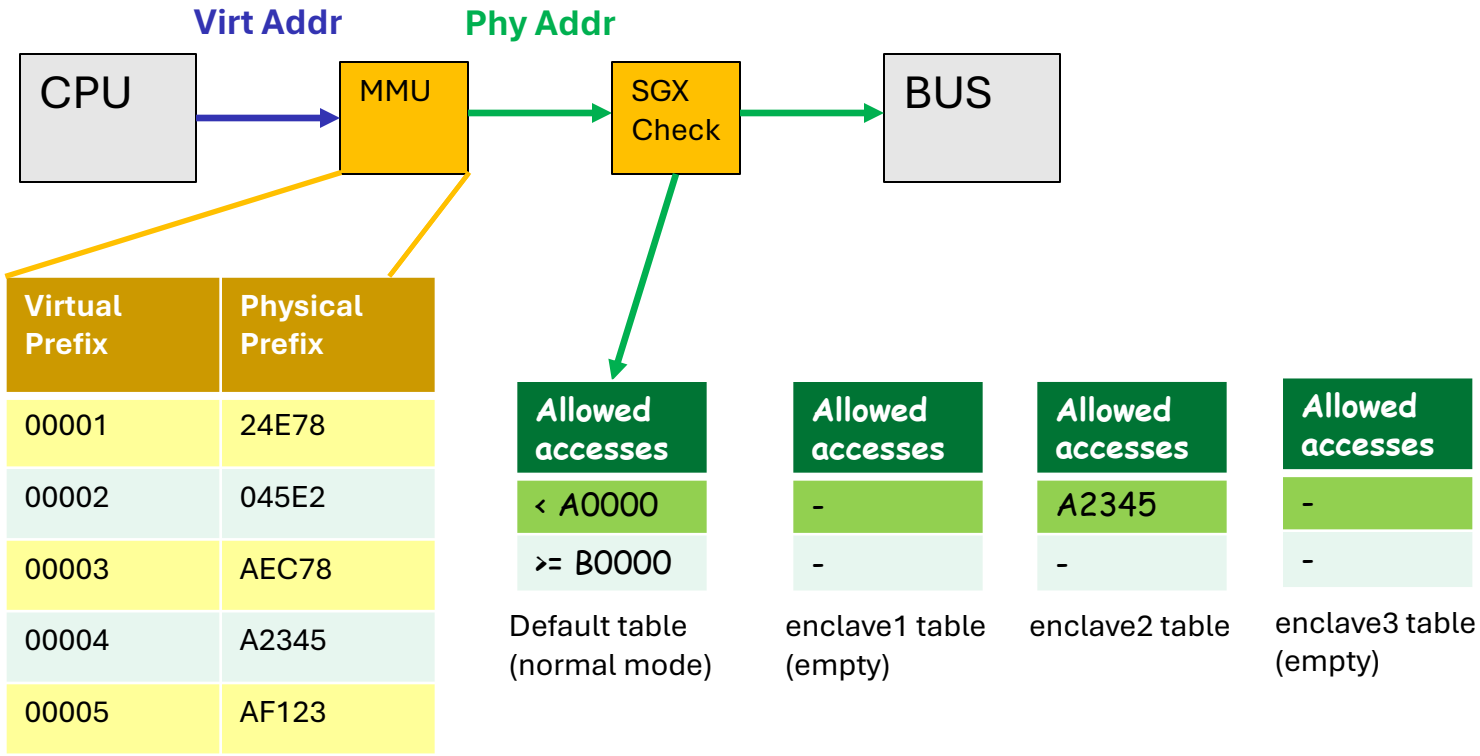


1. CPU finds a free space in PRM for <pageA>
2. CPU allocates <pageA>
3. CPU updates "allowed access" table

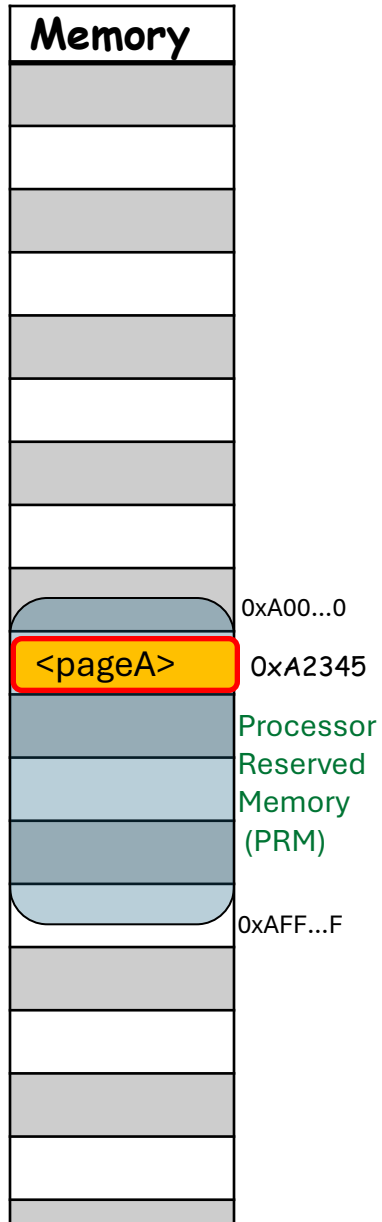


Intel SGX Architecture – Memory Translation

EADD(enclave2, <pageA>)

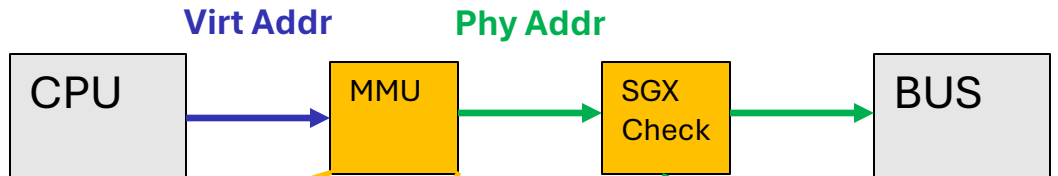


1. CPU finds a free space in PRM for <pageA>
2. CPU allocates <pageA>
3. CPU updates "allowed access" table



Intel SGX Architecture – Memory Translation

EADD(enclave2, <pageA>); EADD(enclave2, <pageB>)



Virtual Prefix	Physical Prefix
00001	24E78
00002	045E2
00003	AEC78
00004	A2345
00005	AF123

Allowed accesses
< A0000
>= B0000

Default table (normal mode)

Allowed accesses
AEC78
-

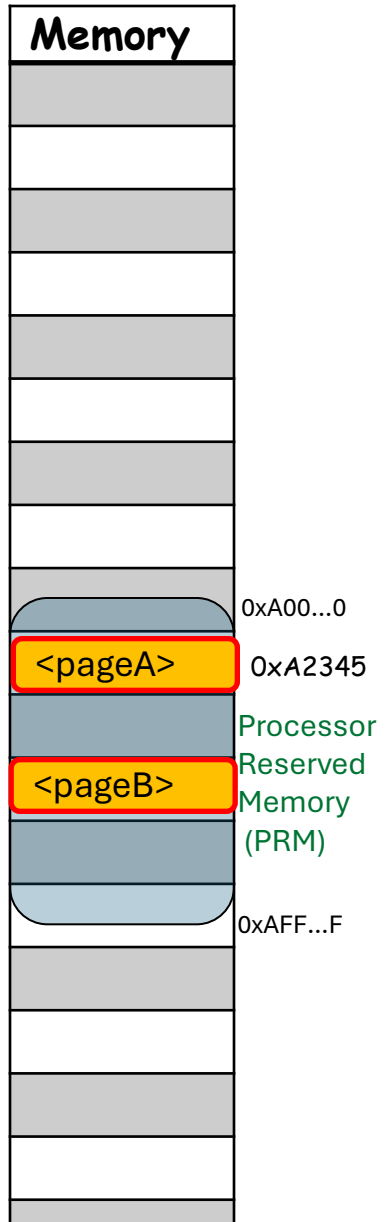
enclave1 table

Allowed accesses
A2345
-

enclave2 table

Allowed accesses
-
-

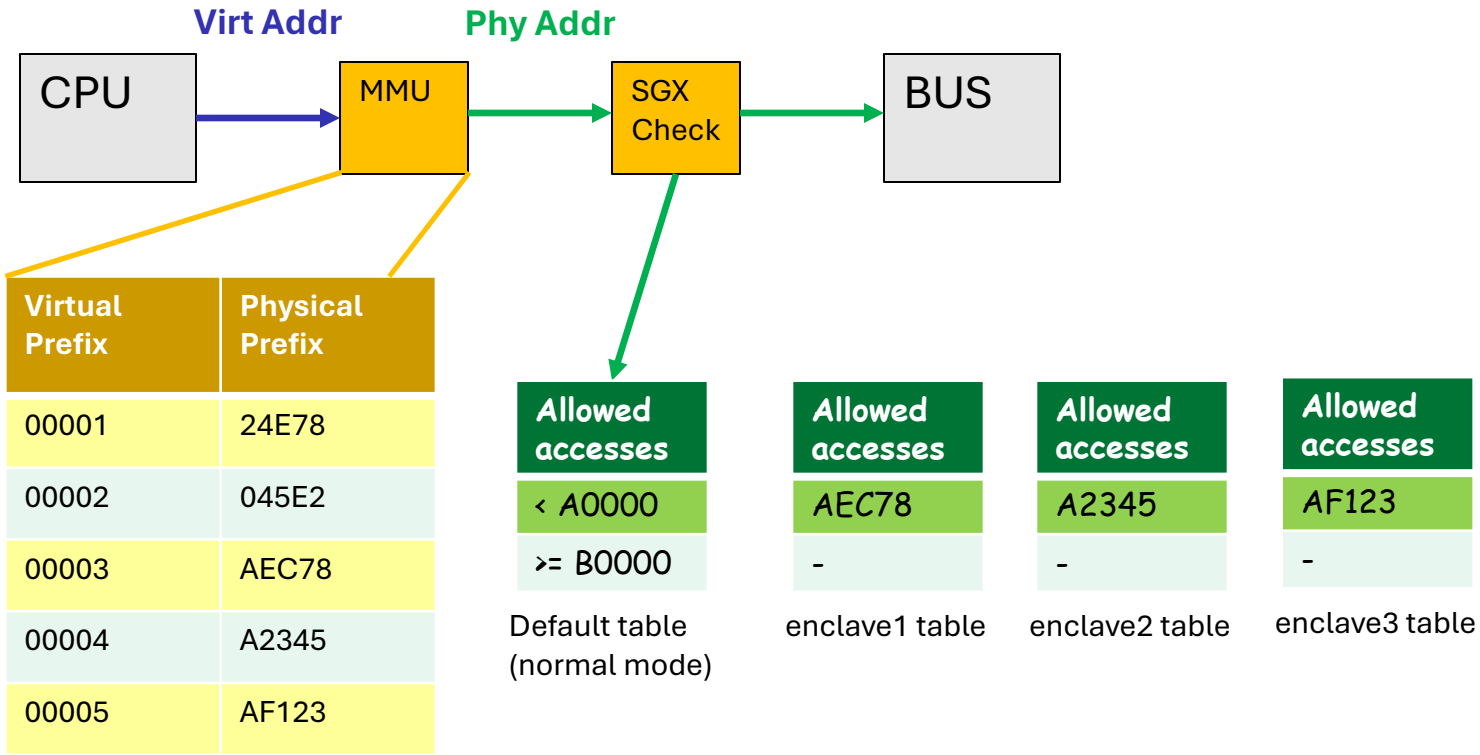
enclave3 table (empty)

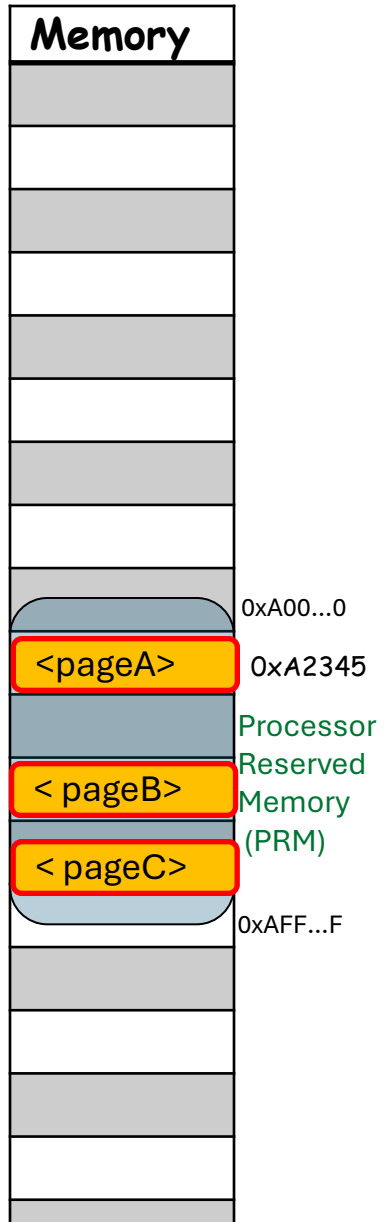


1. CPU finds a free space in PRM for <pageB>
2. CPU allocates <pageB>
3. CPU updates "allowed access" table

Intel SGX Architecture – Memory Translation

EADD(enclave2, <pageA>); EADD(enclave2, <pageB>); EADD(enclave2, <pageC>)

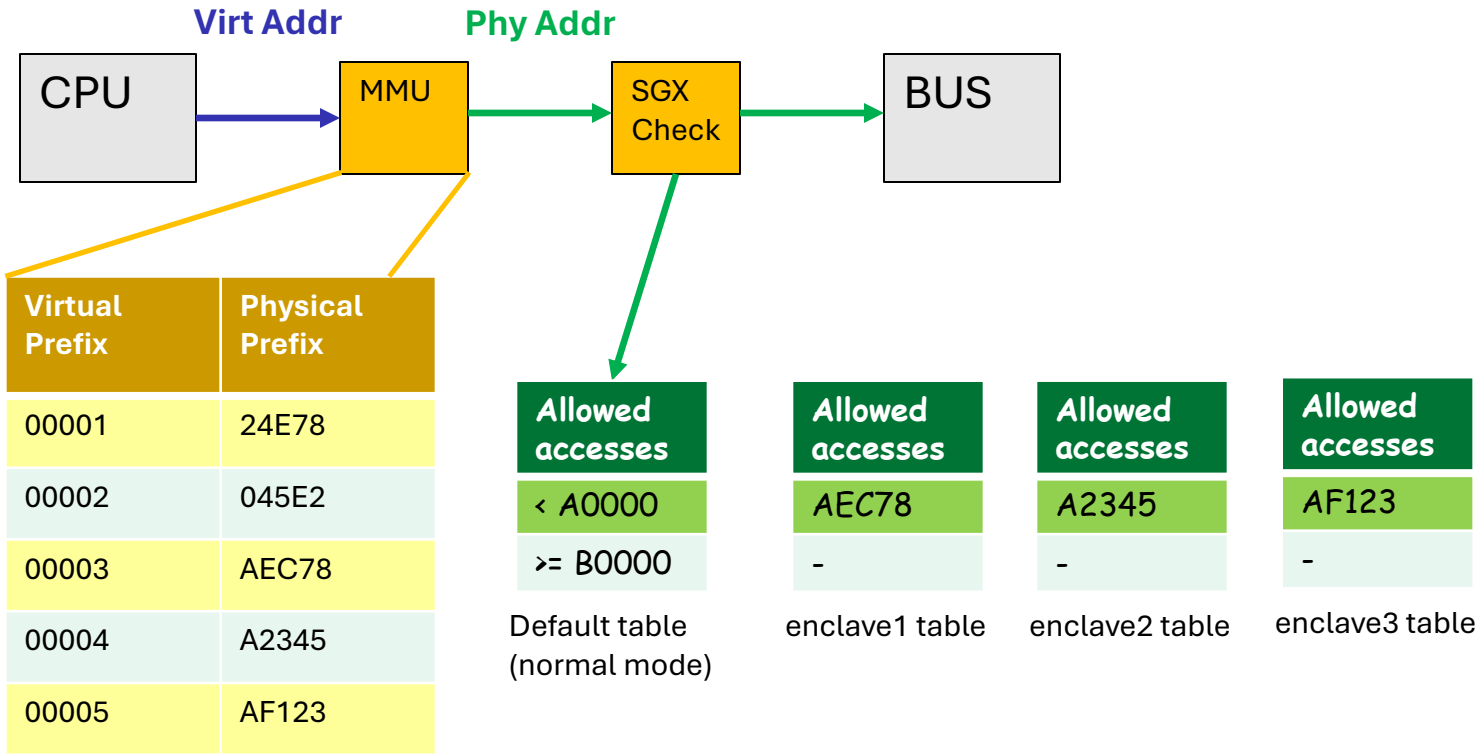




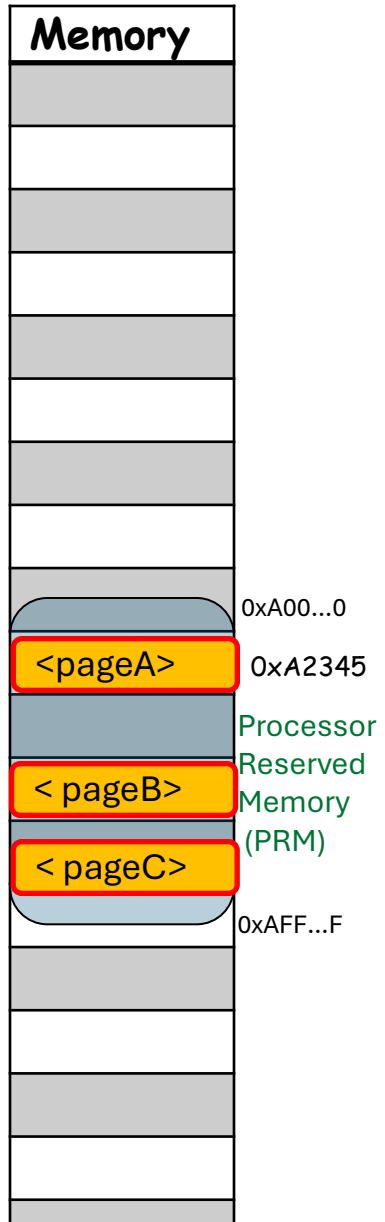
1. CPU finds a free space in PRM for <pageC>
2. CPU allocates <pageC>
3. CPU updates "allowed access" table

Intel SGX Architecture – Memory Translation

EINIT(enclave2)

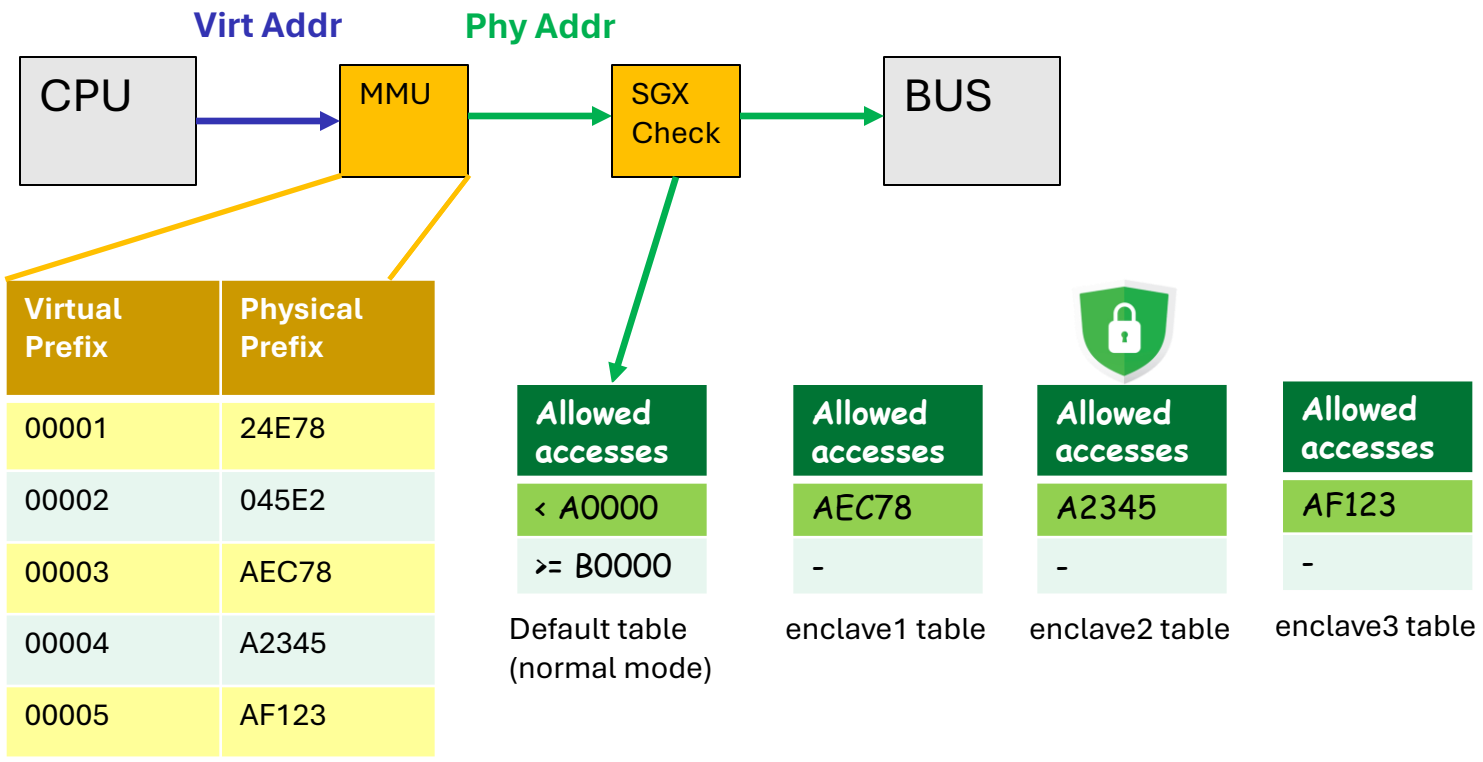


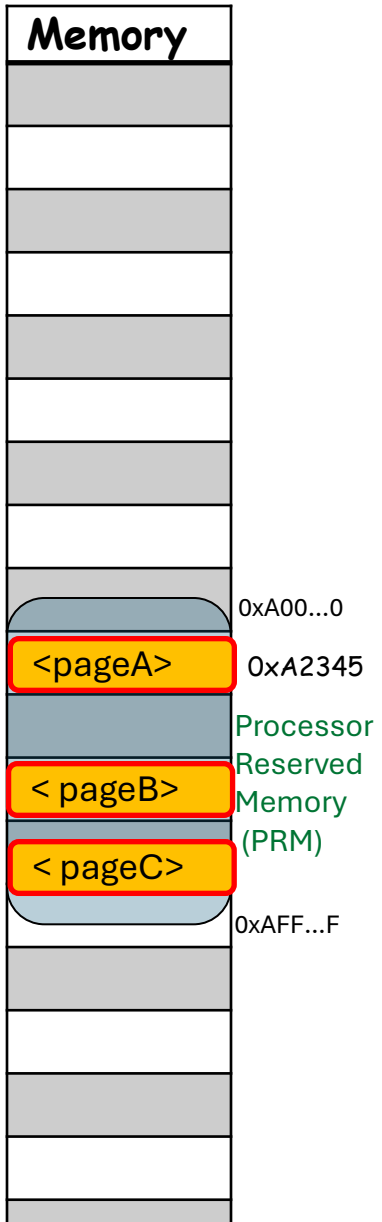
Initialized enclave2… what does it mean?



Intel SGX Architecture – Memory Translation

EINIT(enclave2);



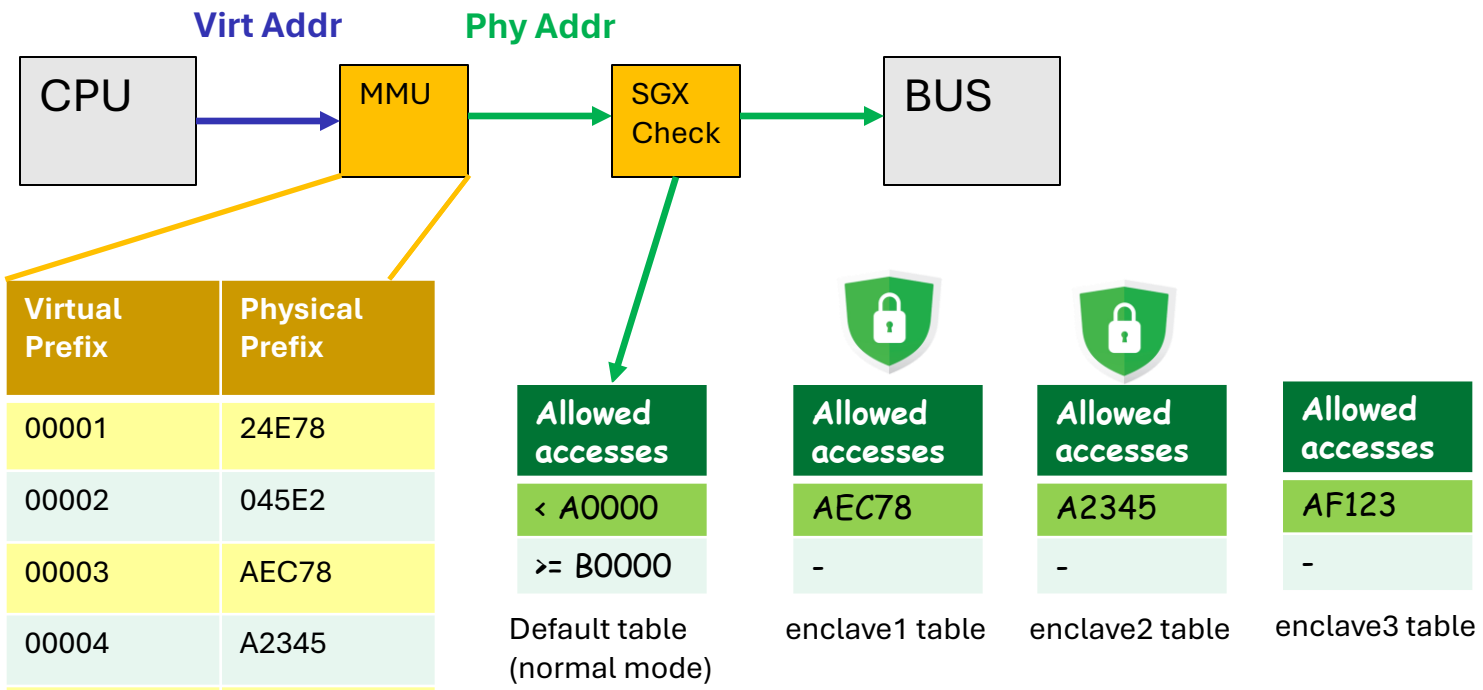


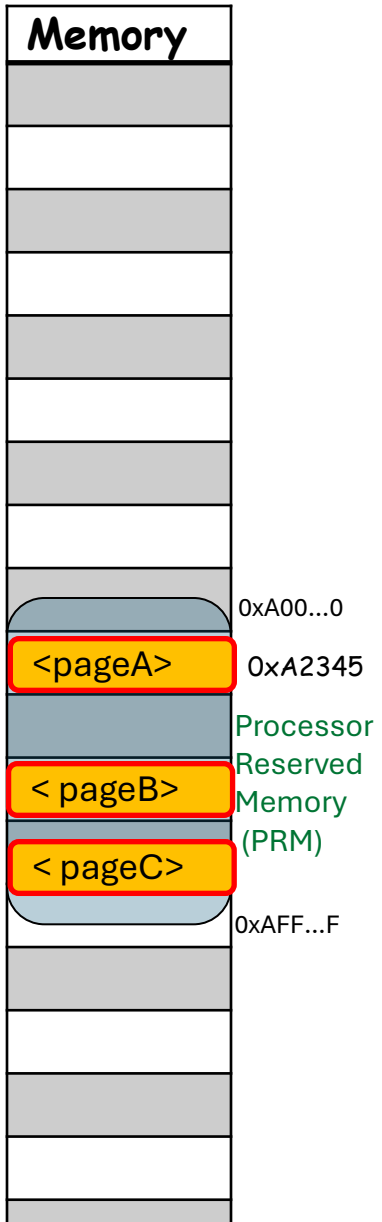
Initialized enclave2… what does it mean?

No modifications to enclave 2 past this point

Intel SGX Architecture – Memory Translation

EINIT(enclave2); EINIT(enclave1)





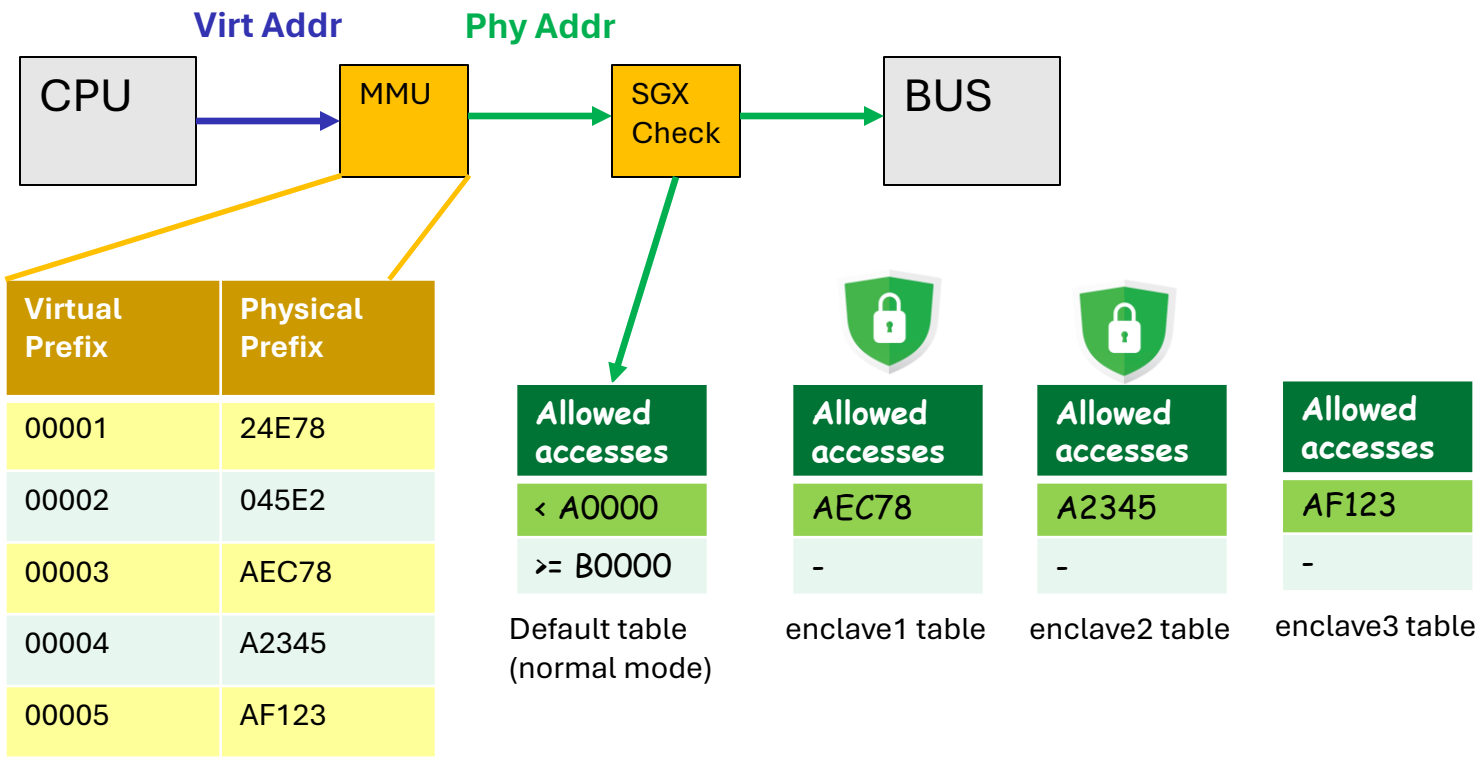
Initialized enclave2… what does it mean?

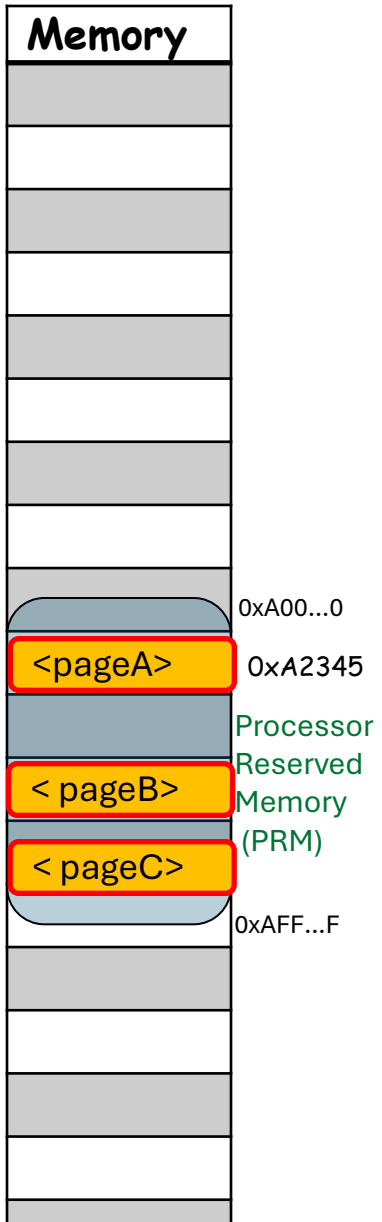
No modifications to enclave 2 past this point

No modifications to enclave 1 past this point

Intel SGX Architecture – Memory Translation

EENTER

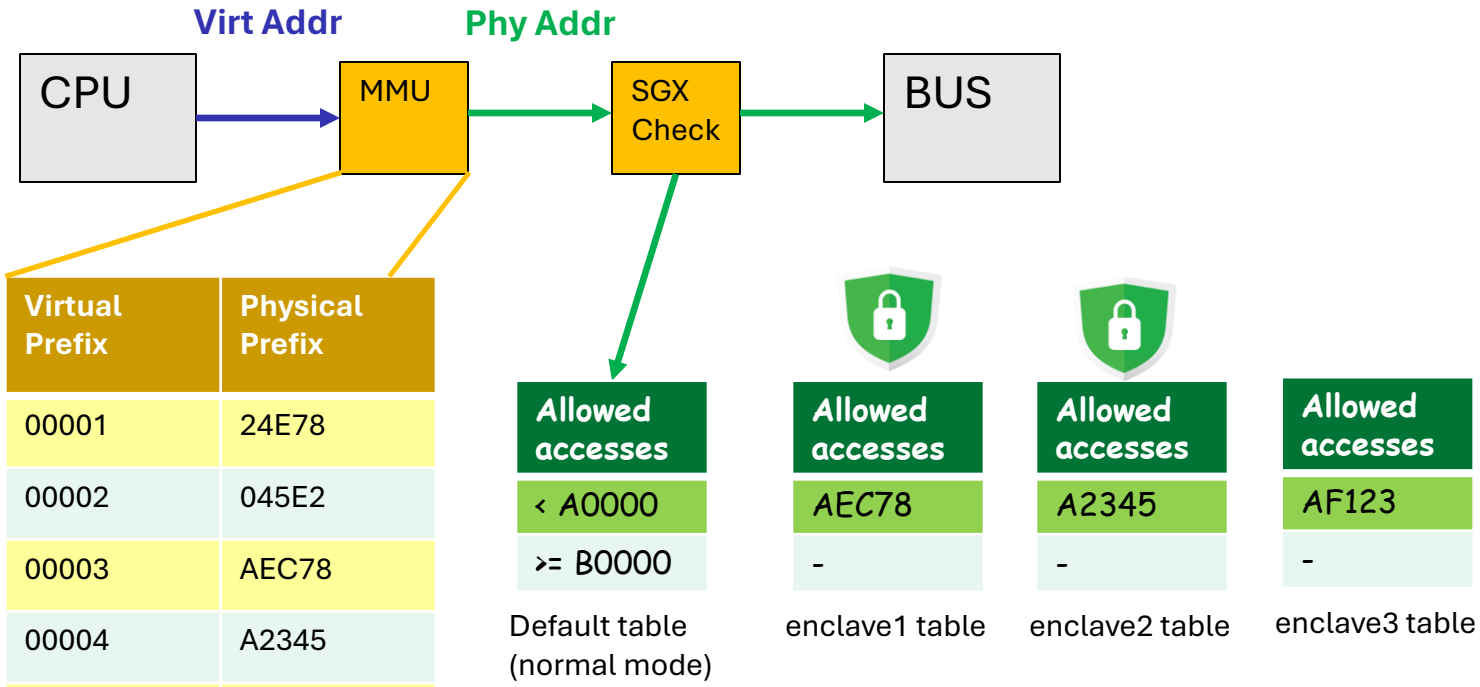




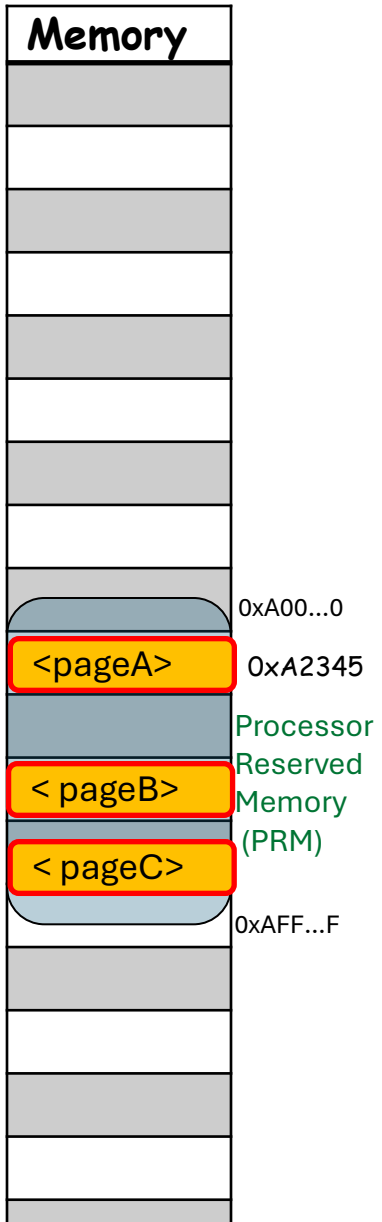
Next step: execute the enclave with EENTER

Intel SGX Architecture – Memory Translation

Enclave Execution Example 1:
EENTER(enclave3); Virt Access: 0x00005123

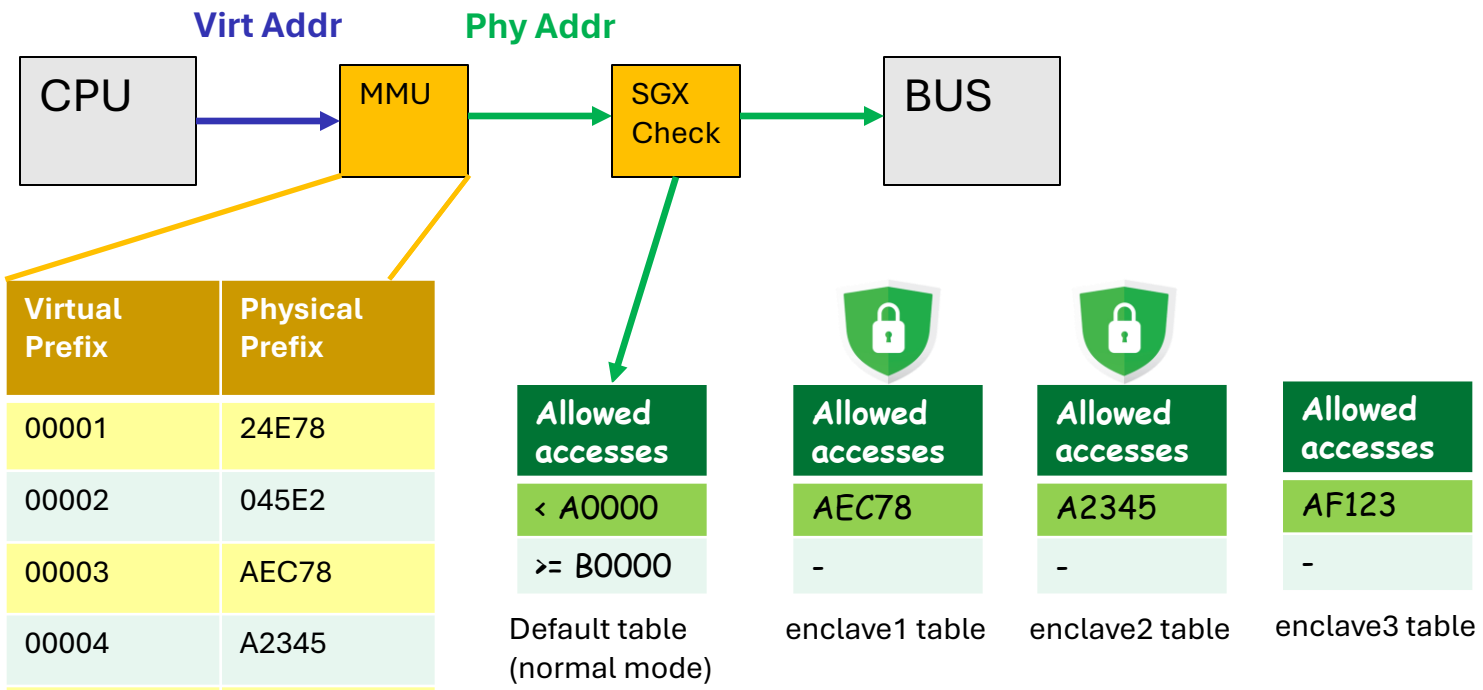


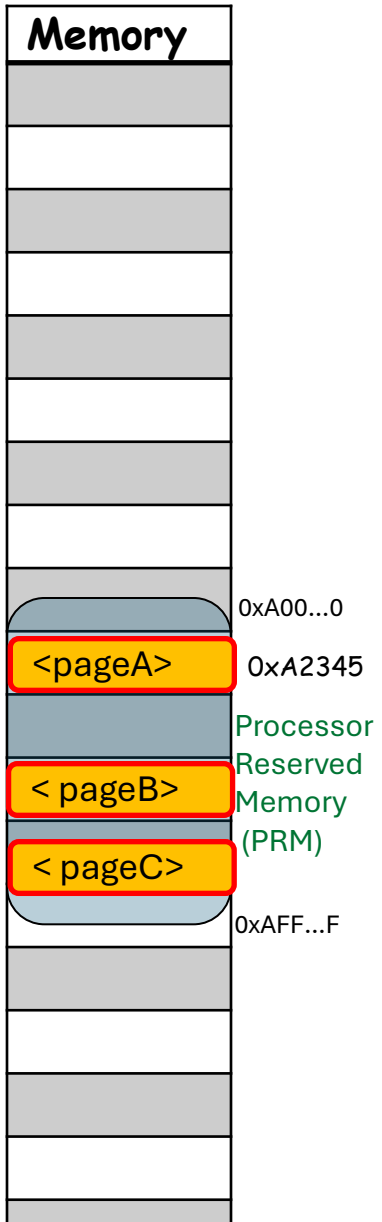
What happens here?



Intel SGX Architecture – Memory Translation

Enclave Execution Example 1:
EENTER(enclave3); Virt Access: 0x00005123



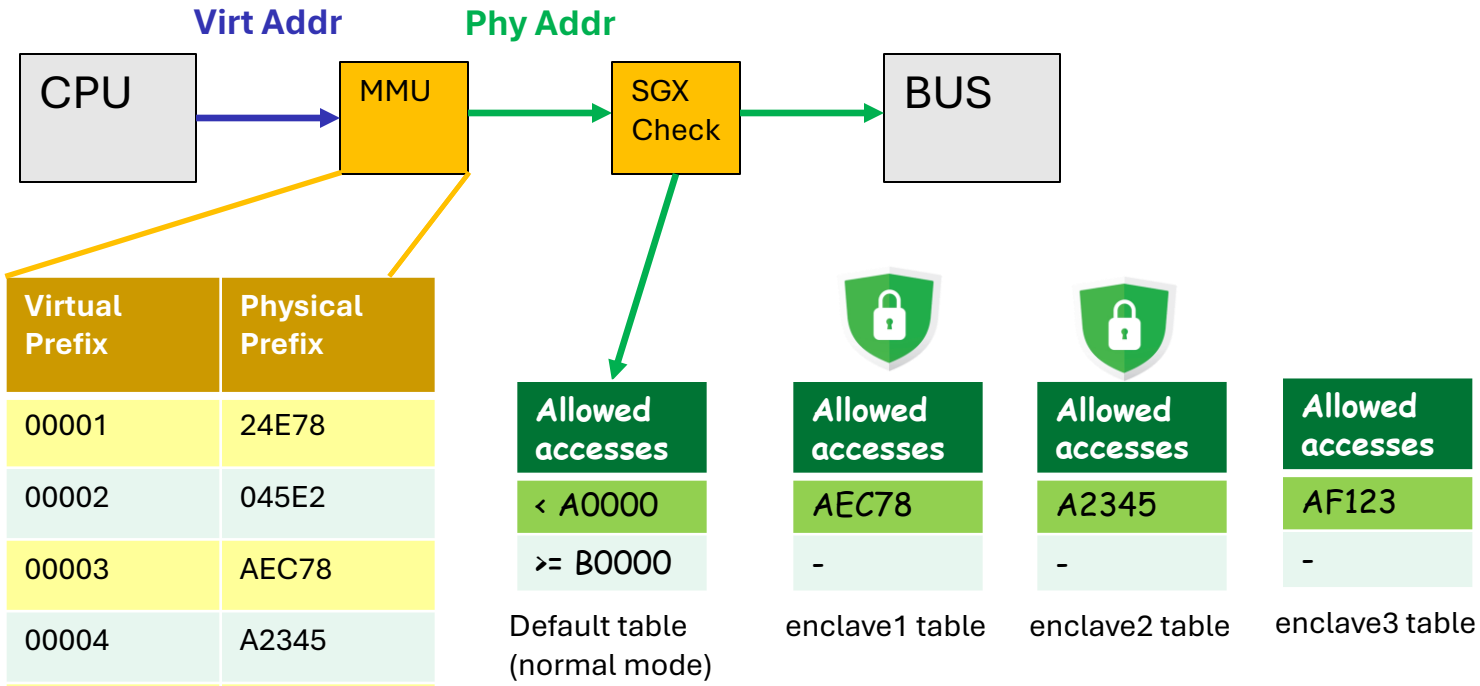


What happens here?

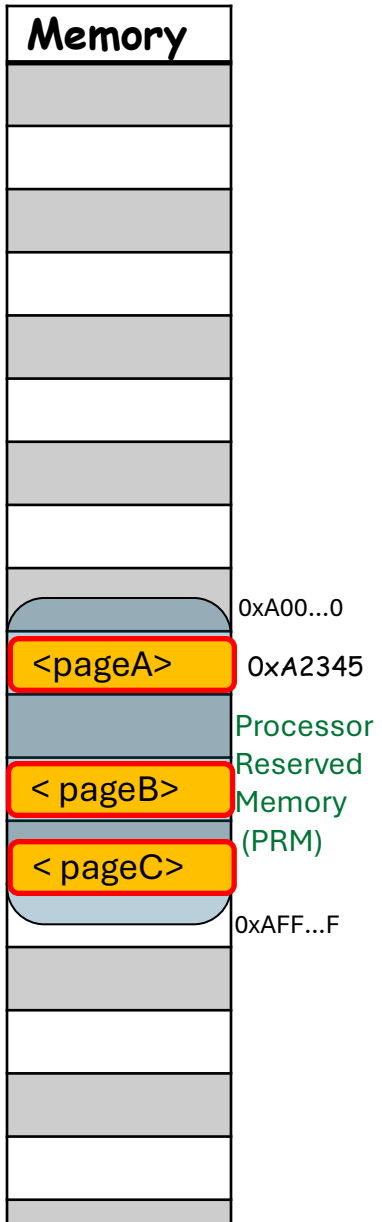
Blocked: enclave3 never initialized

Intel SGX Architecture – Memory Translation

Enclave Execution Example 2:
EENTER(enclave2); Virt Access: 0x00004123

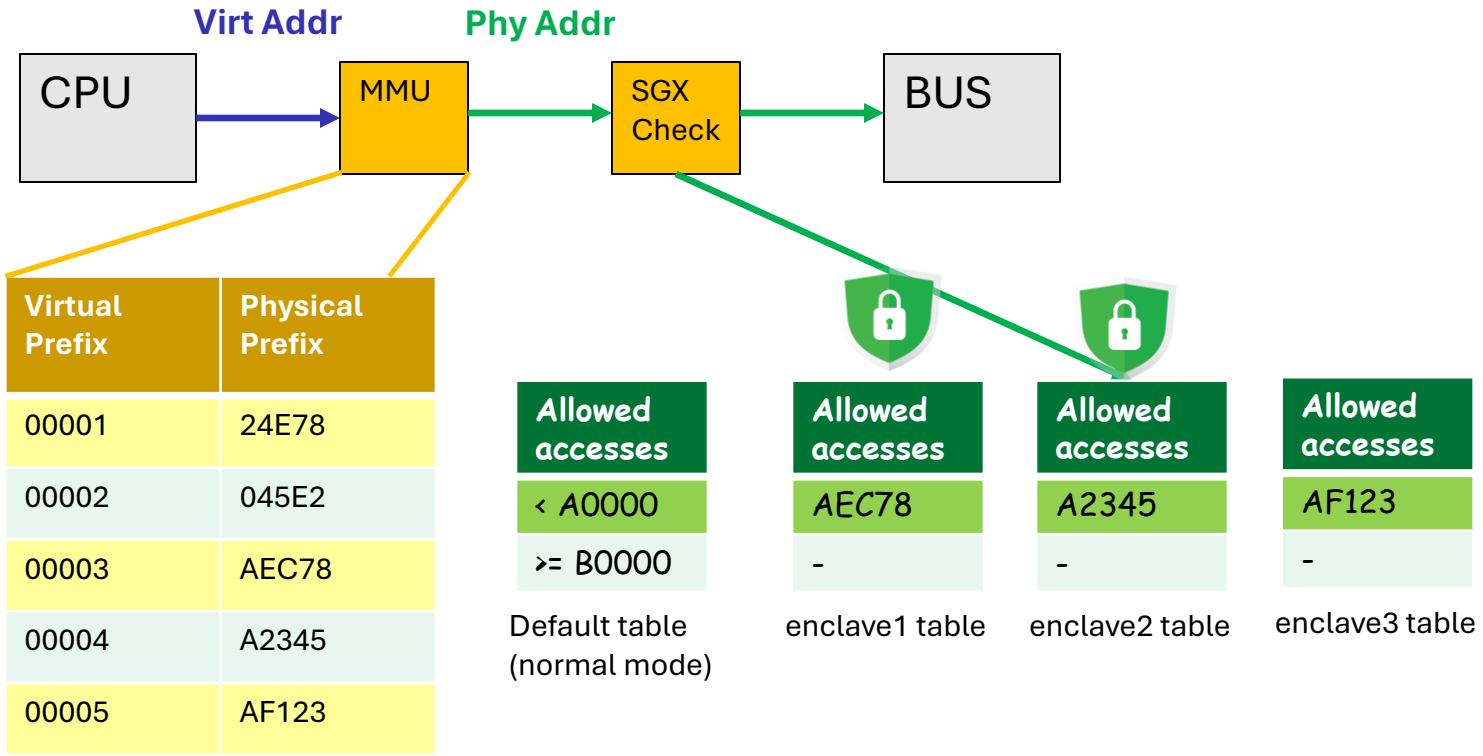


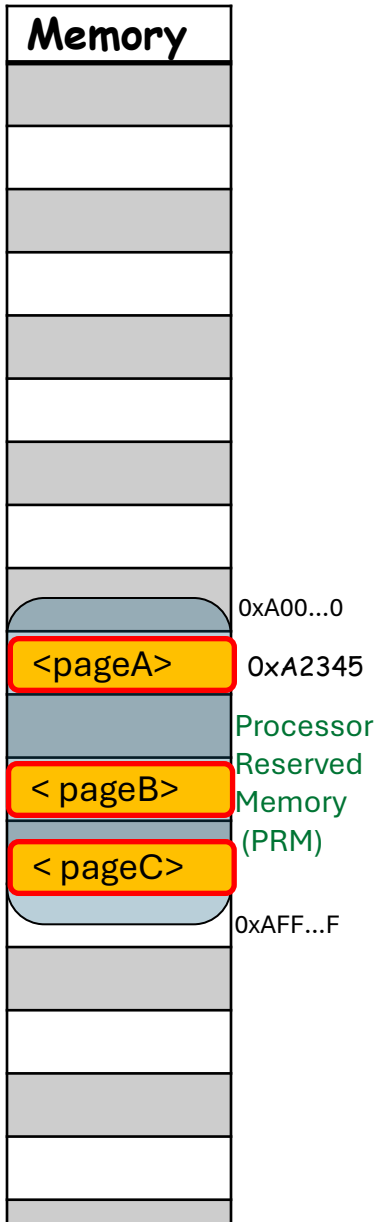
What happens here?



Intel SGX Architecture – Memory Translation

Enclave Execution Example 2:
EENTER(enclave2); Virt Access: 0x00004123



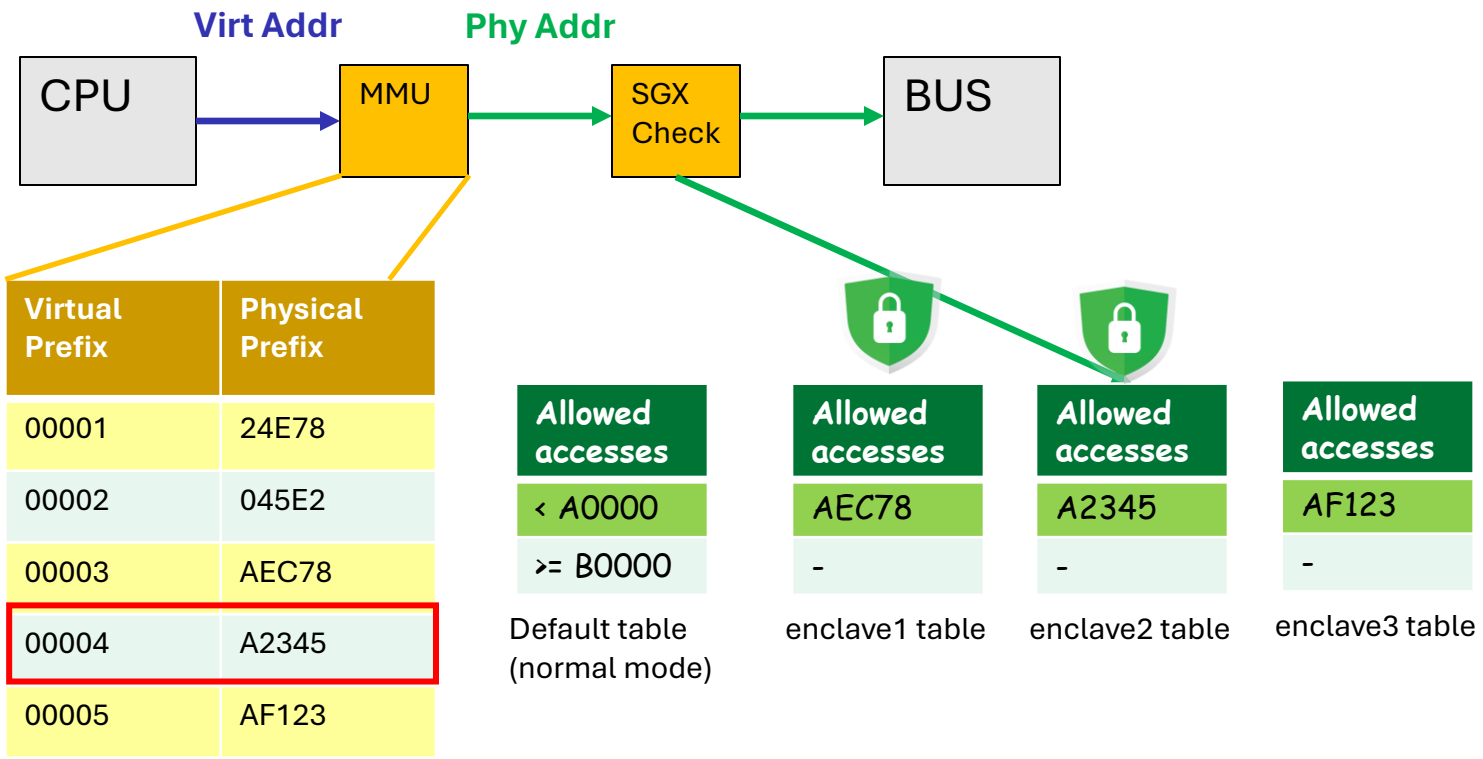


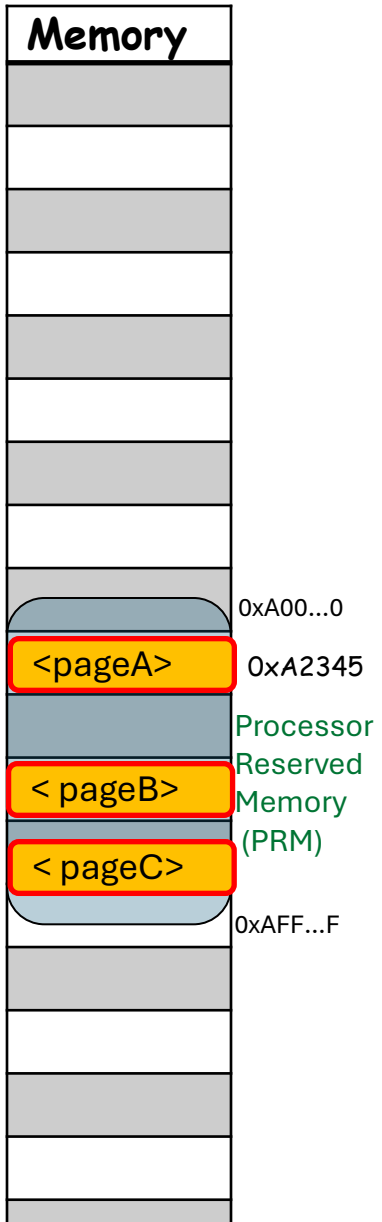
What happens here?

1. HW switches SGX check to enclave 2 table

Intel SGX Architecture – Memory Translation

Enclave Execution Example 2:
EENTER(enclave2); Virt Access: 0x00004123



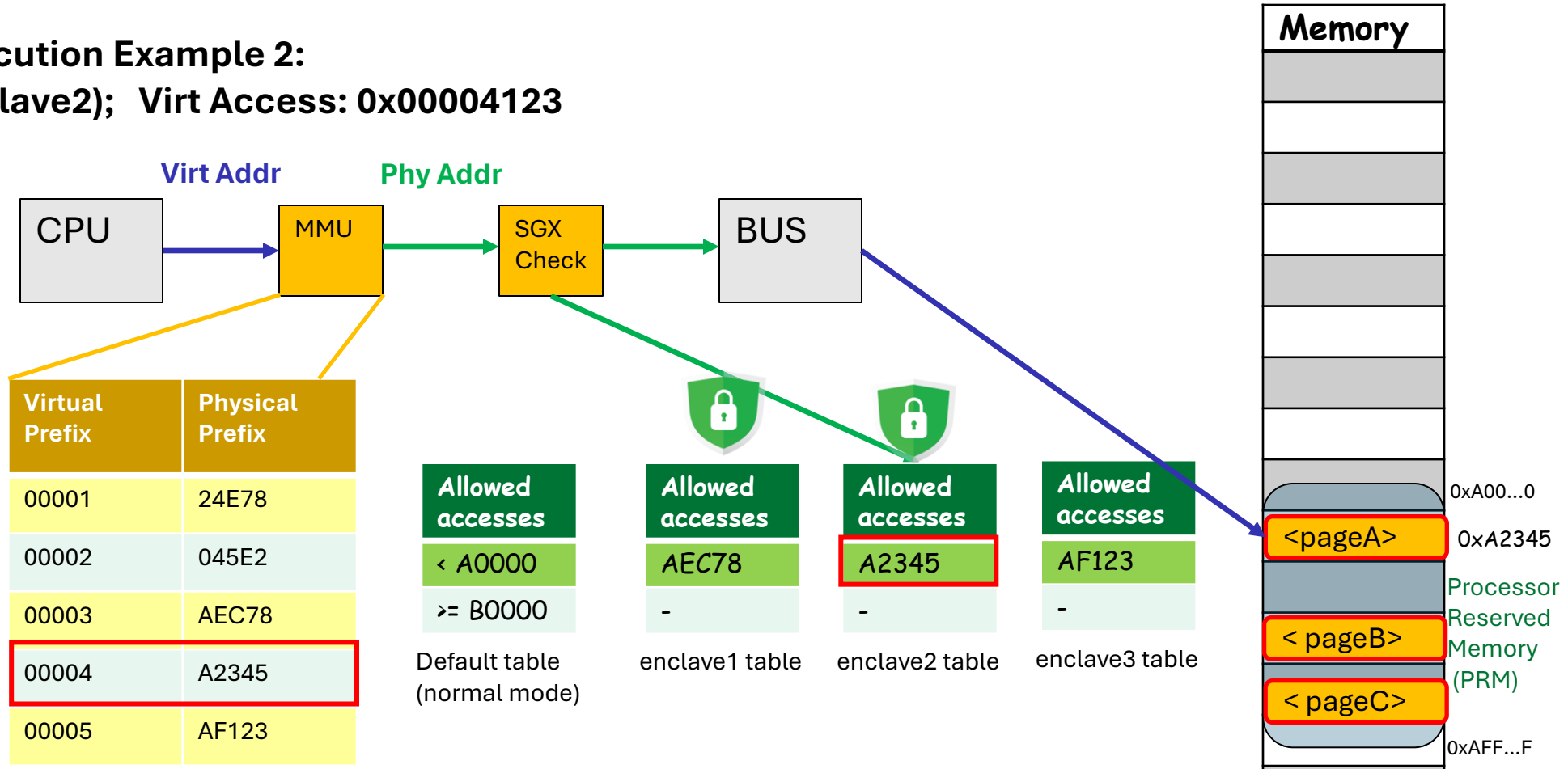


What happens here?

1. HW switches SGX check to enclave 2 table
2. MMU lookup → 0x0004 : A2345

Intel SGX Architecture – Memory Translation

Enclave Execution Example 2:
EENTER(enclave2); Virt Access: 0x00004123

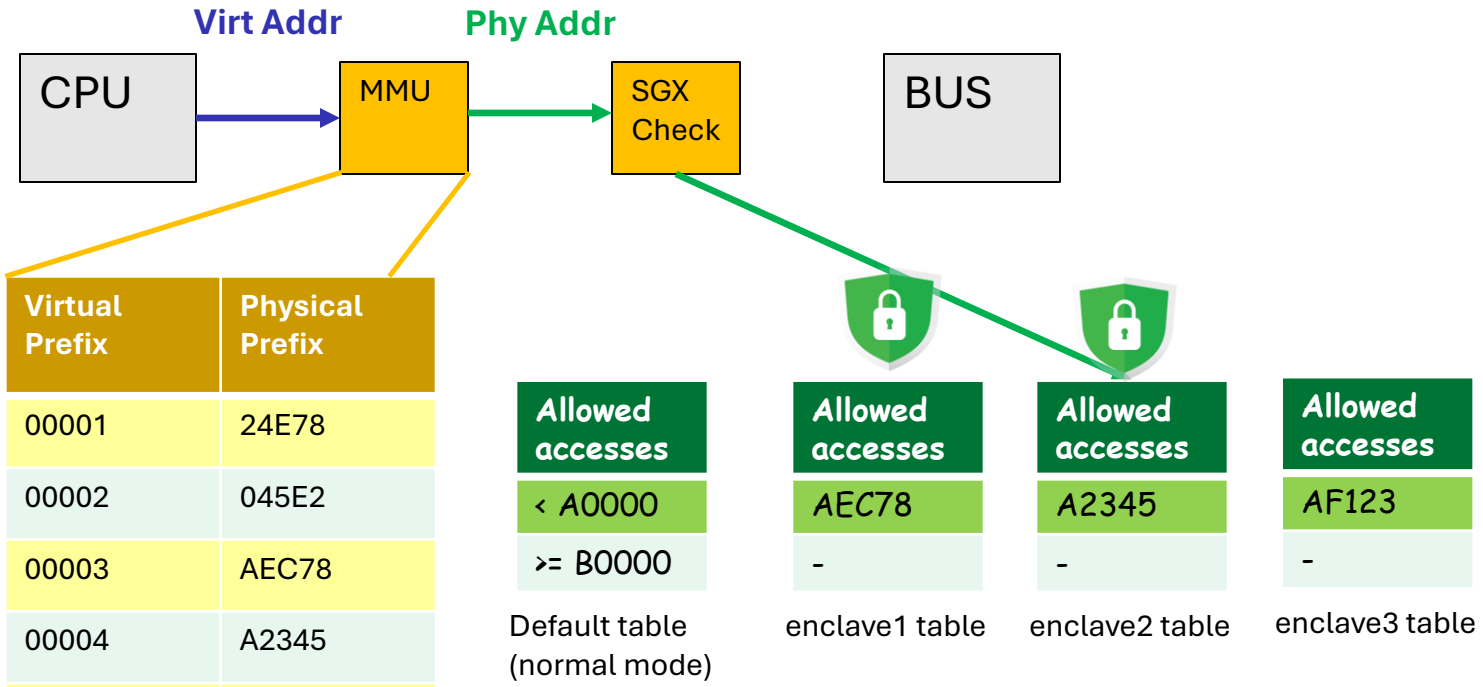


What happens here?

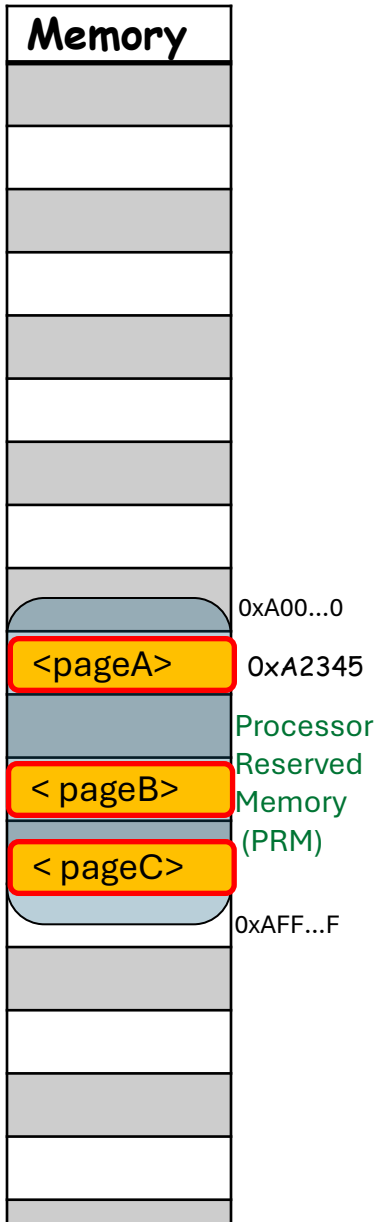
1. HW switches SGX check to enclave2 table
2. MMU lookup → 0x0004 : A2345
3. Check if access is allowed for enclave2 → pass!

Intel SGX Architecture – Memory Translation

Enclave Execution Example 3:
EENTER(enclave1); Virt Access: 0x00004123

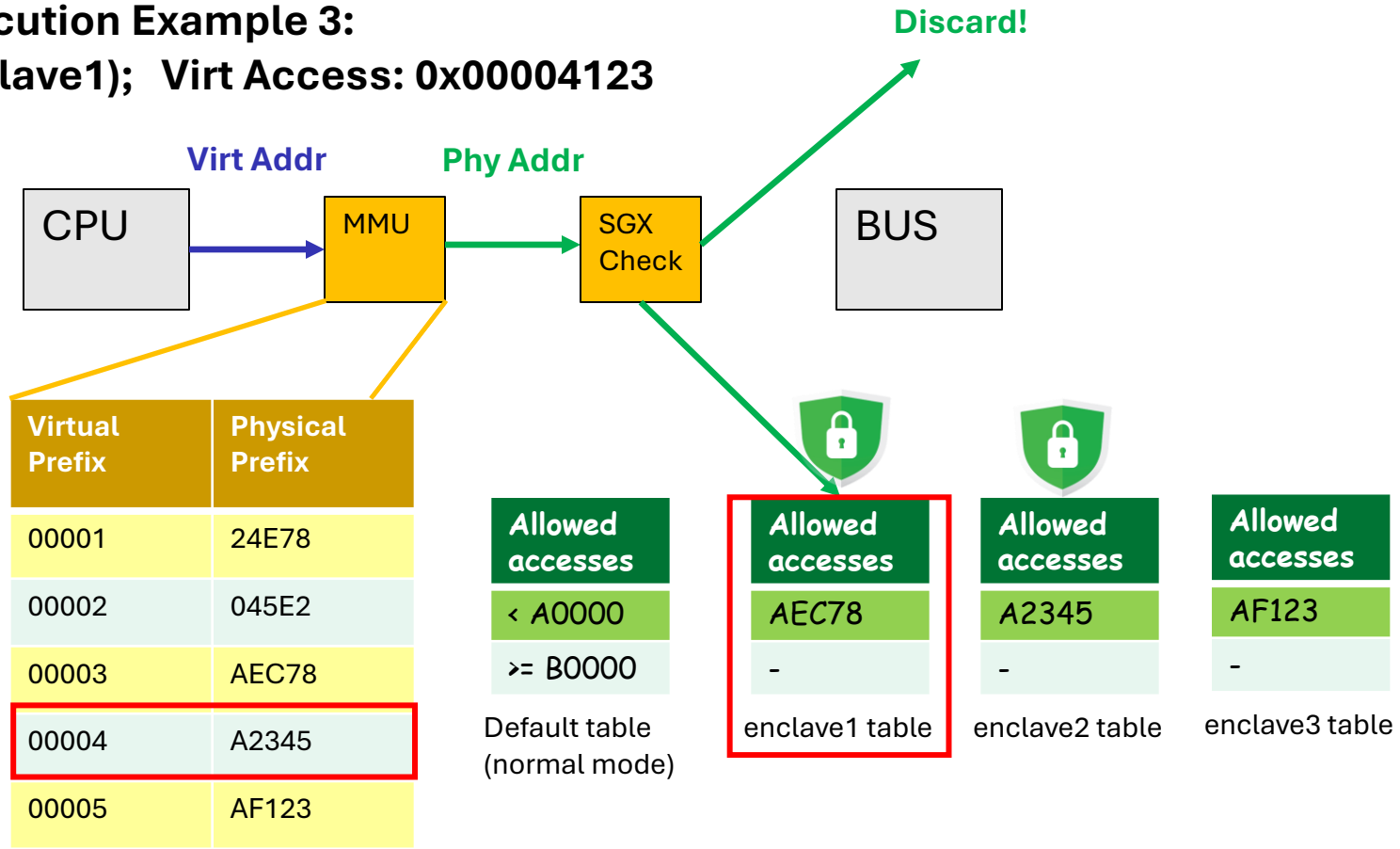


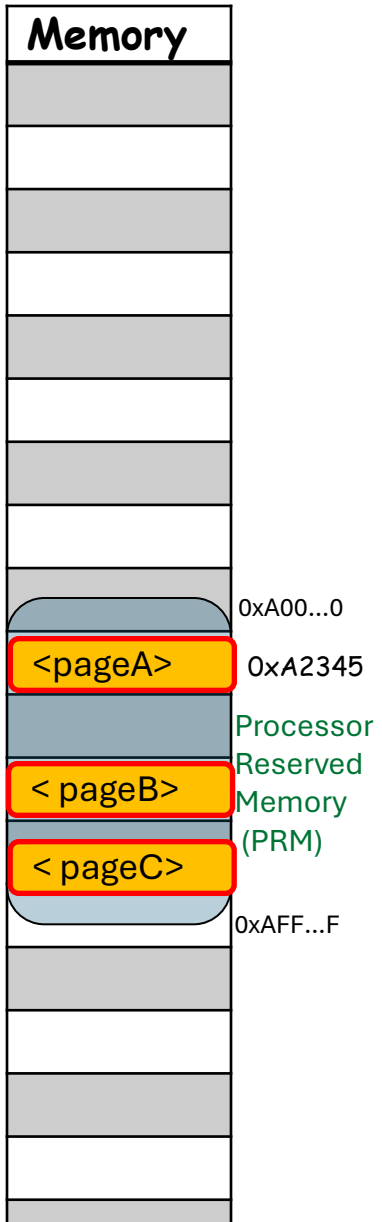
What happens here?



Intel SGX Architecture – Memory Translation

Enclave Execution Example 3:
EENTER(enclave1); Virt Access: 0x00004123



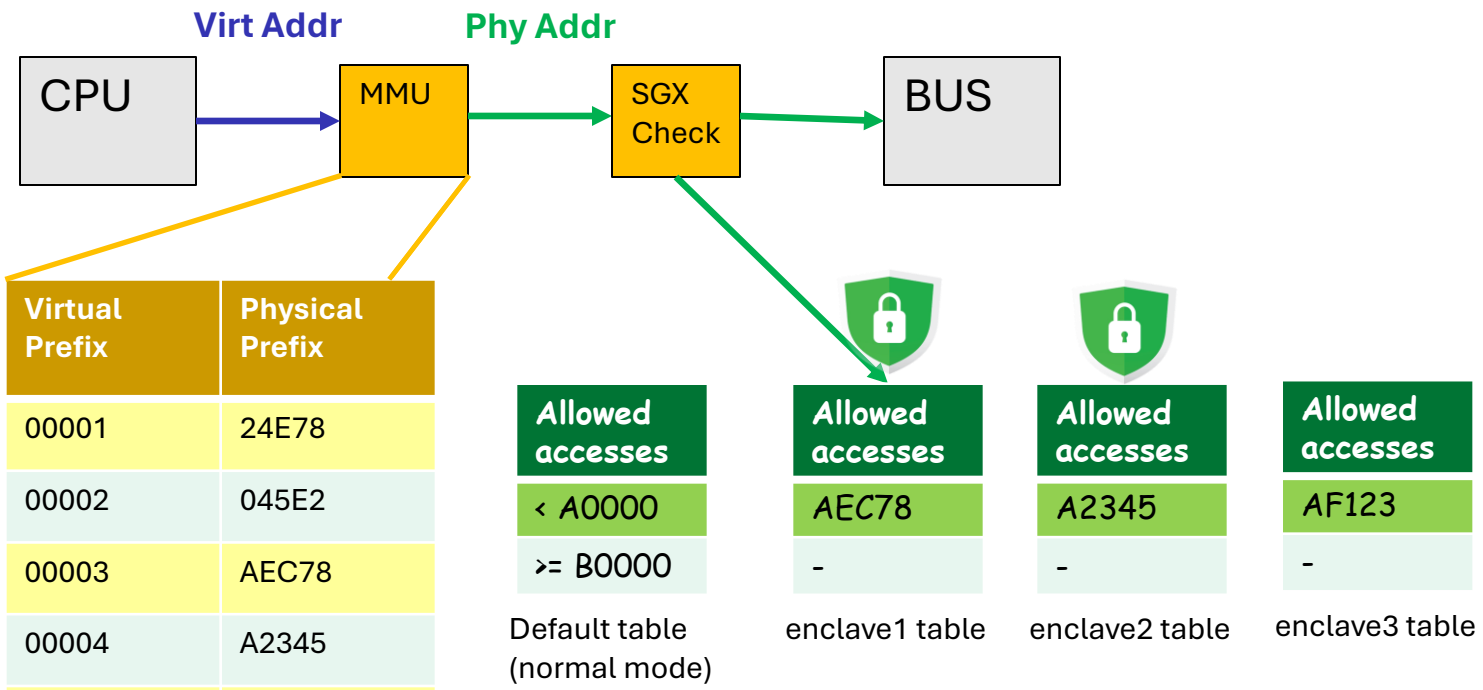


What happens here?

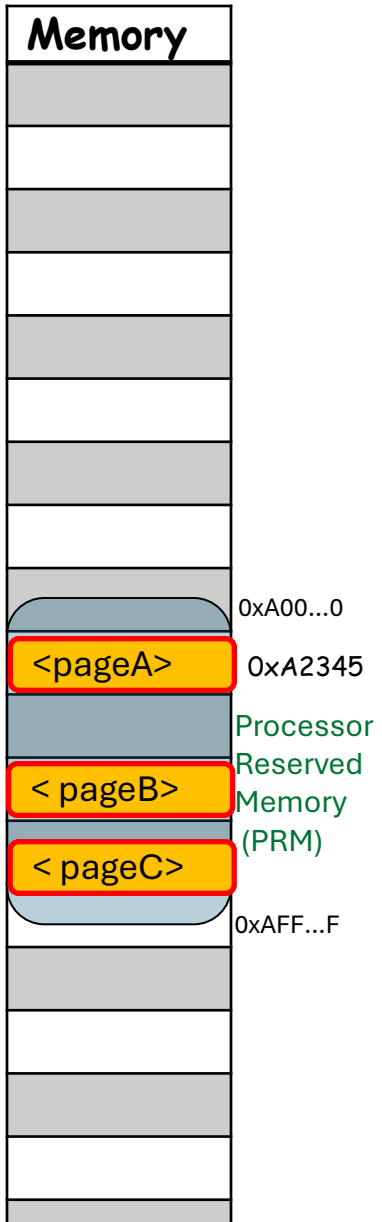
1. HW switches SGX check to enclave1 table
2. MMU lookup → 0x0004 : A2345
3. Check if access is allowed for enclave1 → discard!

Intel SGX Architecture – Memory Translation

EEXIT

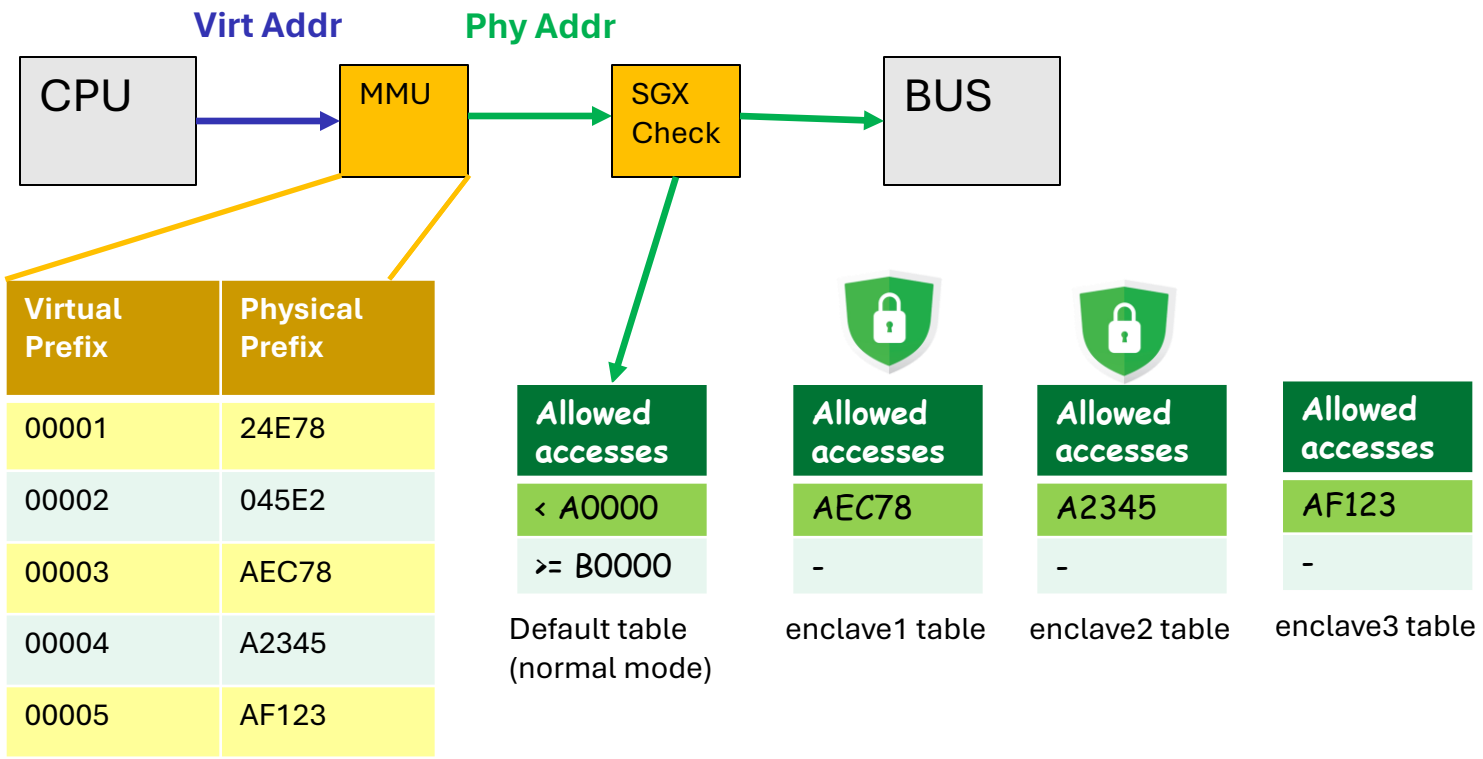


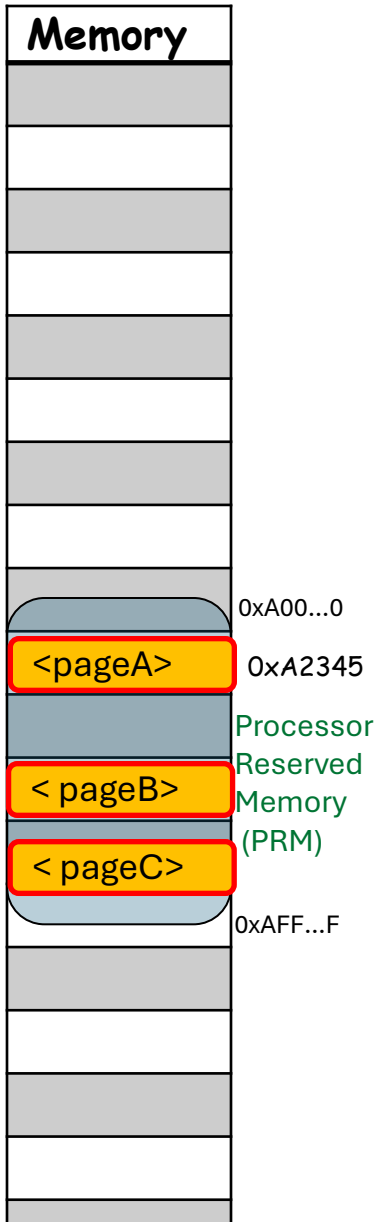
What happens here?



Intel SGX Architecture – Memory Translation

EEXIT





EEXIT switches back to the default table

Further Access not possible until EENTER again

Intel SGX Architecture

Isolation in Intel SGX

Enclave life cycle

Memory Translation in SGX

Remote Attestation

Intel SGX Architecture – Remote Attestation

SGX Remote Attestation:

- Use EEXTEND to measure data/code pages into MRENCLAVE
- Use EQUOTE to get a quote of the MRENCLAVE contents
- **Problem:** How to get signing keys to the Enclave?
- **Intel's Approach:** Provision a ***Quoting Enclave*** with **Provisioning key**
  - Provisioning key burned into the device
  - Quoting enclave attests itself with provisioning key
  - If passes, Intel passes an Attestation Key to Quoting Enclave
  - Quoting then enclave uses Attestation Key for Remote Attestation of App Enclaves

Intel SGX Architecture – Remote Attestation

(1) An off platform challenger requests that its enclave produce an attestation

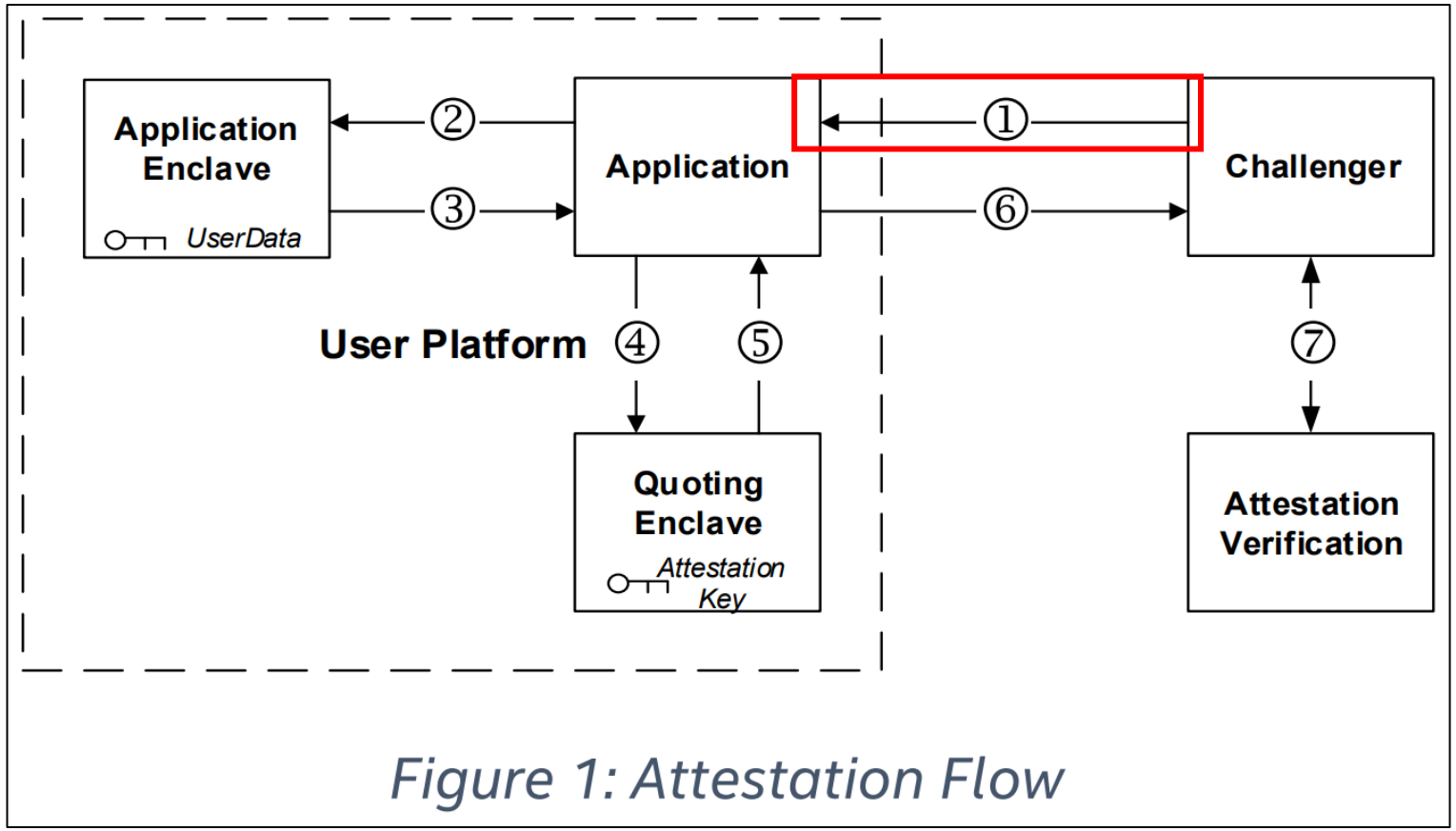


Figure 1: Attestation Flow

Intel SGX Architecture – Remote Attestation

(2) Application requests attestation from the enclave

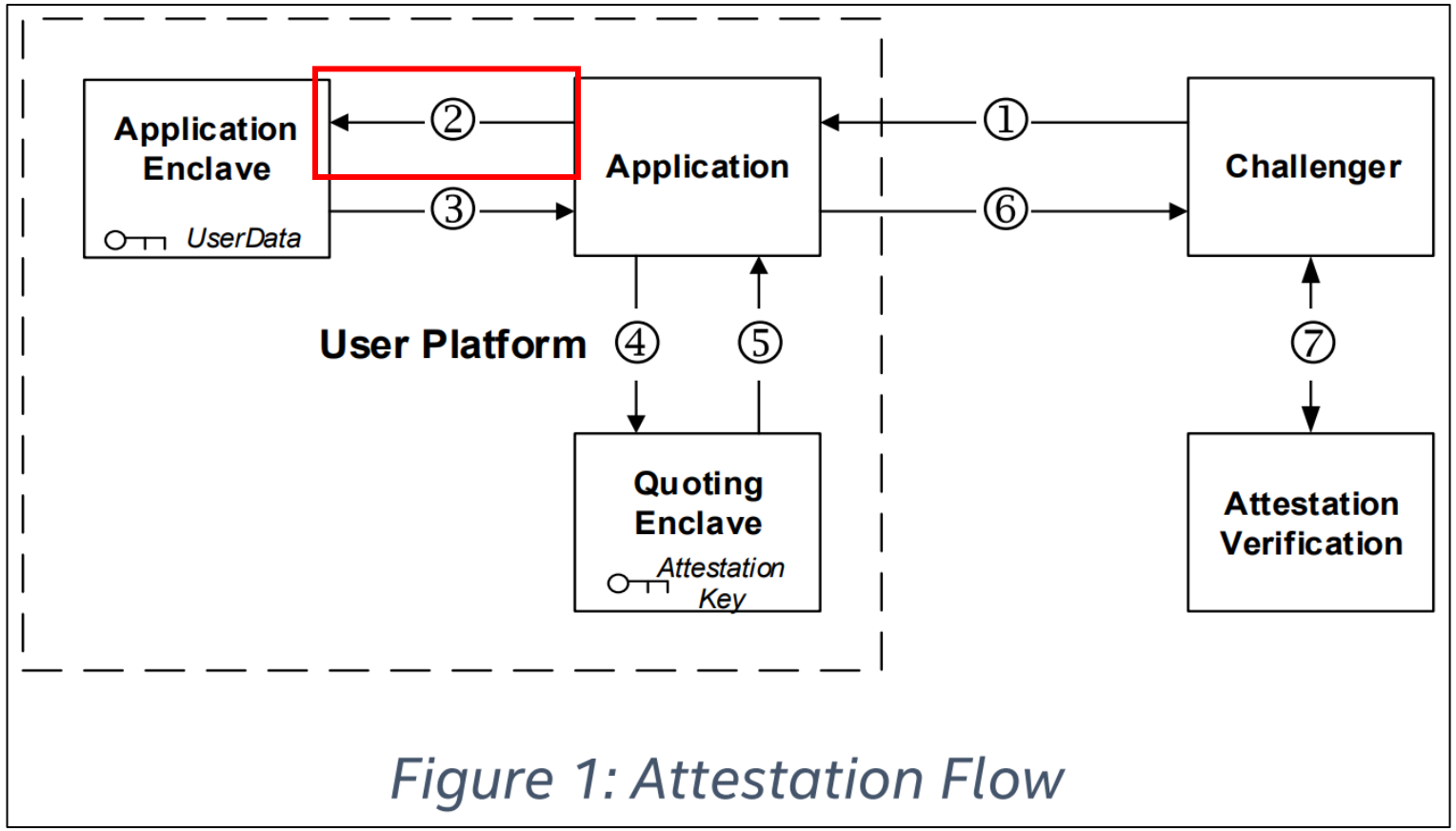


Figure 1: Attestation Flow

Intel SGX Architecture – Remote Attestation

A local attestation occurs between the (3) Application Enclave and (4) **Quoting Enclave**

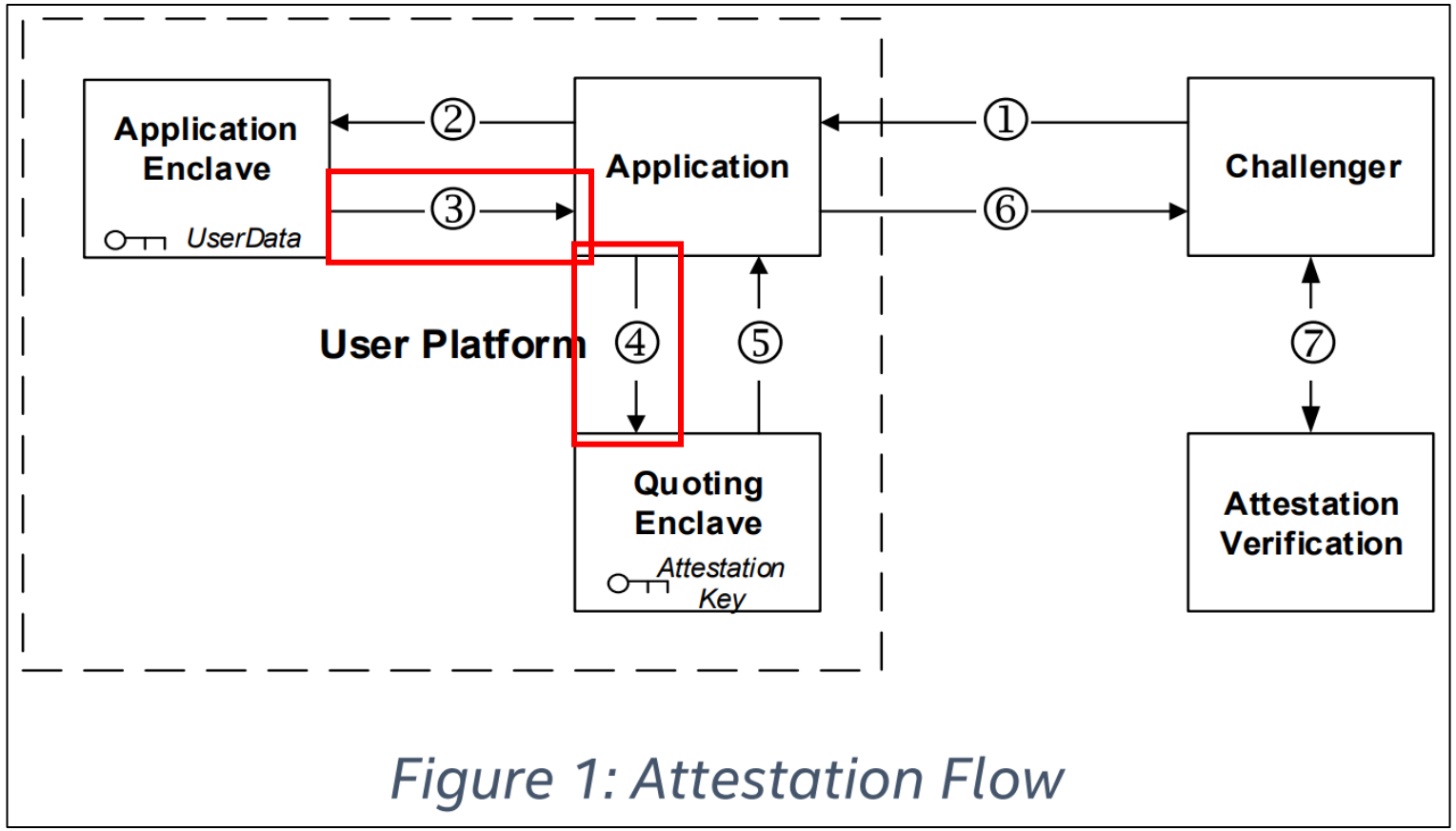


Figure 1: Attestation Flow

Intel SGX Architecture – Remote Attestation

(5) Quote is returned to the application and (6) sent back to the challenger

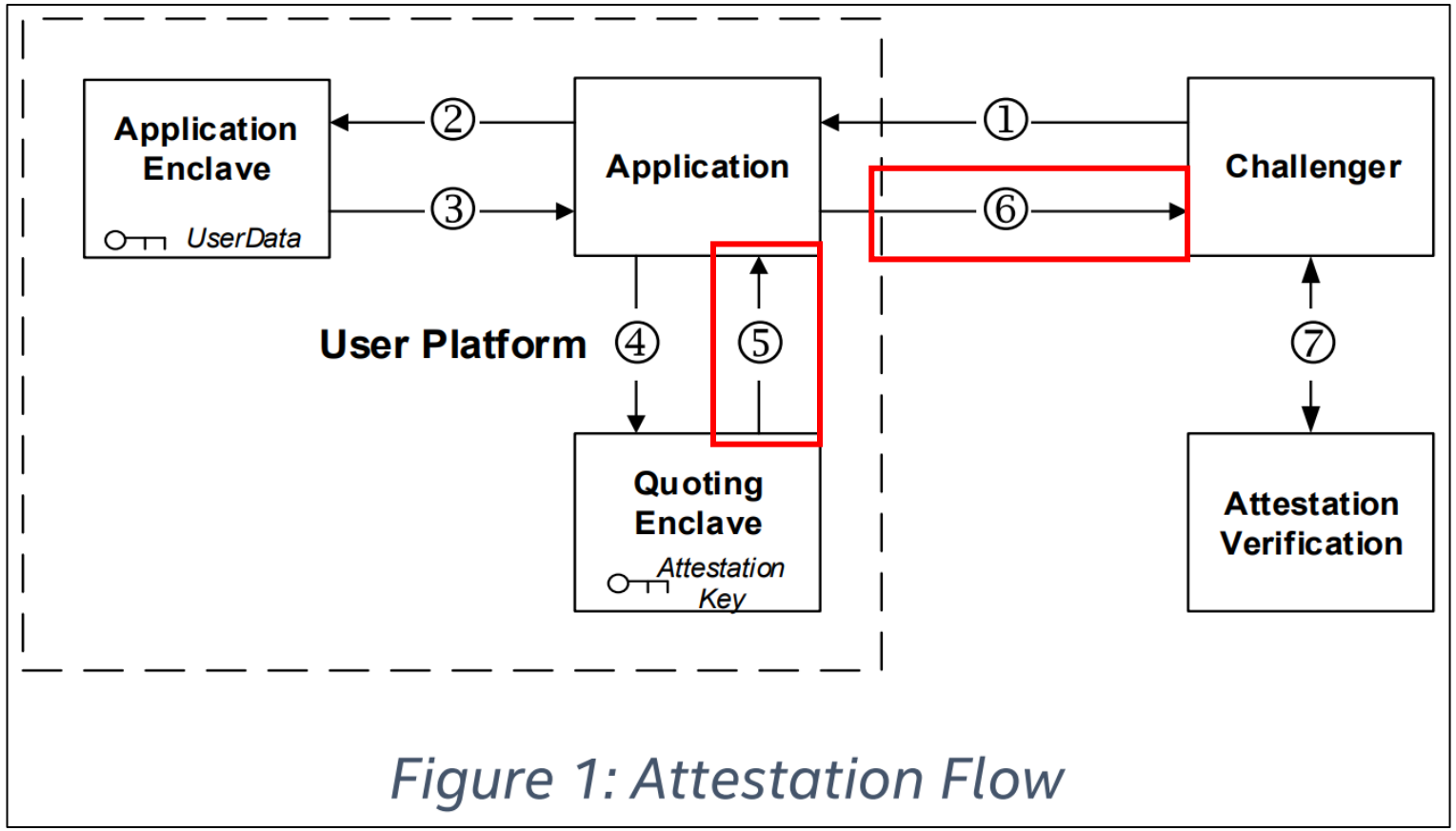


Figure 1: Attestation Flow

Intel SGX Architecture – Remote Attestation

(7) The challenger uses a verification service to verify the Quote

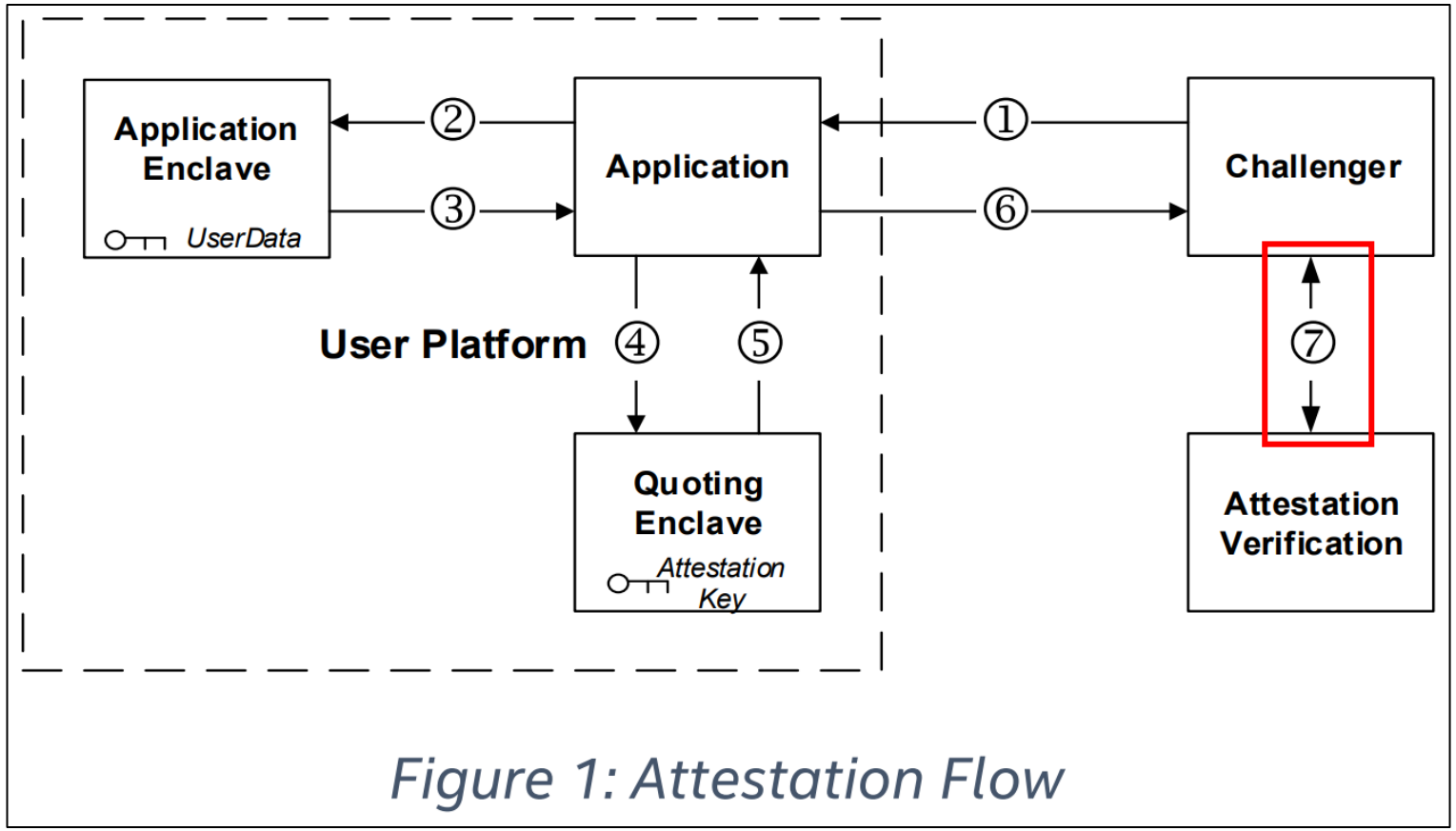


Figure 1: Attestation Flow

Intel SGX Architecture – Remote Attestation

Much more complicated process than TPM-based

Check out these detailed slides for more

Demo and examples of ECDSA Attestation with Intel SGX

Intel SGX Architecture – Other features

Writing data to external storage

- Risky, also challenging without overheads
- Large data requires moving pages in and out often
- Requires encryption + integrity protection
- Enabled through **Memory Encryption Engine –** read more
  - General use enables physical protection
    - All memory writes out of the CPU are encrypted
    - All memory reads into the CPU are decrypyted

Intel SGX Architecture – Other features

Sealing:

- Bind measurement of the current enclave in MRENCLAVE to a key (EGETKEY)
- Bind identity of enclave author to a key
- Similar idea to what is provided by *Wrap Keys* in TPM
- **Read more**

That's all for today!

Coming up....

- System-wide TEE in Android → ARM TrustZone

Reminders:

- A4 is due on July 25
- Research project proposals

Resources:

- "Intel's SGX In-depth Architecture"-- Great Intel SGX slides by Syed Kamran
- "Quote Generation, Verification, and Attestation with Intel SGX DCAP"
- "Confidential Computing 101 – Intel SGX Technology"
- "Intel SGX Explained"
- "Life Cycle of an SGX Enclave"
- SGX 101