Lecture 20:
Intel SGX Demo & Deep Dive

CPSC 4440/5440, 2025 Spring

Instructor: Fan Zhang

Yale

Computer Science

Logistics

- HW4 graded — grab a copy of solutions if you want them
- Please choose your talk!

Trusted Execution Environments (TEES)

- Idea: carve out an “execution environment” isolated from the rest of the system
- **minimal TCB**
  - (OS or hypervisor out of TCB)
- **verifiability**

Intel Software Guard Extensions (SGX)

- The **first** TEE that leaves OS/hypervisor out of TCB

SGX Version	User Space `enclu`	Kernel Space `encls`	Total
SGX-v1	5	13	18
SGX-v2	8	16	24

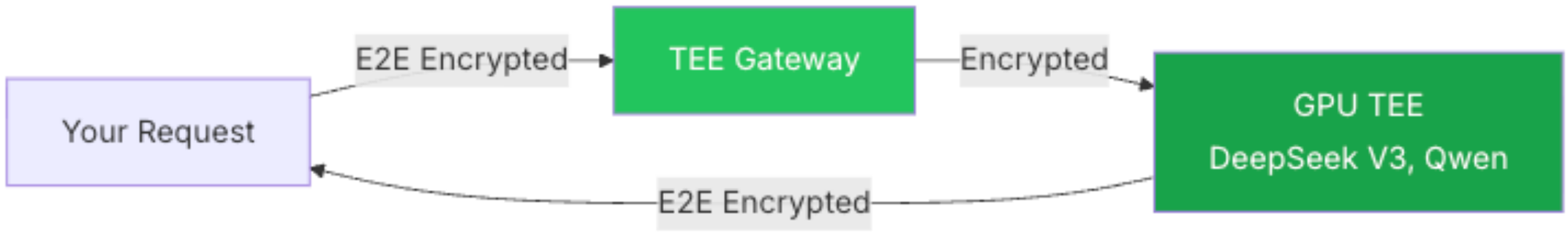
`sysenter/sysexit` Instructions

Our roadmap

1. **Live demo**
2. Security mechanism in SGX
3. Summary & pitfalls

RedPill

- End-to-end TEE confidential AI models using Intel TDX & Nvidia GPU TEE
- Hides input/output; proves inference integrity w.r.t. a model hash.



https://docs.redpill.ai/privacy/confidential-ai/overview

RedPill

Trust Model

You must trust:

- CPU vendor (Intel) - TEE hardware correctness
- Cryptographic algorithms - AES, RSA, ECDSA
- Open source code - Auditable on GitHub

You do NOT need to trust:

- RedPill operators
- Cloud infrastructure provider
- Operating system
- Other applications on the server

RedPill

GPU TEE Providers

RedPill offers 17 confidential AI models across 4 GPU TEE providers:

Phala Network (8 models)

Model	Parameters	Context	Use Case
`phala/glm-5`	Large	202K	Systems engineering
`phala/gpt-oss-120b`	117B (MoE)	131K	OpenAI-compatible
`phala/qwen3-vl-30b-a3b-instruct`	30B (MoE)	128K	Vision + language
`phala/gemma-3-27b-it`	27B	53K	Multilingual
`phala/uncensored-24b`	24B	32K	Unrestricted
`phala/gpt-oss-20b`	21B (MoE)	131K	Efficient inference
`phala/glm-4.7-flash`	~30B	202K	Agentic coding
`phala/qwen-2.5-7b-instruct`	7B	32K	Budget-friendly

- Supports 17 models, and 4 GPU TEE providers

RedPill

Performance

Near-Native Speed

GPU TEE adds minimal overhead:

Metric	Native	TEE Mode	Overhead
Throughput	100 tok/s	99 tok/s	~1%
Latency	50ms	51ms	~2%
TFLOPS	1979	1959	~1%

RedPill

- Try it live https://chat.redpill.ai/

Verification Flow

- How do we know any of the promises is true?
- Answer: verify attestations

Verification Flow

Like a certificate

TDX

pk_{model}

Docker Stack for LLM X

$\text{Sig}(SK_{\text{Intel}}, [H(\text{stack}) \| pk_{\text{model}})$

Inp

Outp, σ

$\sigma = \text{Sig}(sk_{\text{model}}, H(Inp, Outp))$

e.g., X = phala/qwen3-vl-30b-a3b-instruct

Verification Flow

- We built a verification demo: https://github.com/hackingdecentralized/redpill-ai-verification-demo

Case study: security mechanisms in SGX

While we focus on SGX….

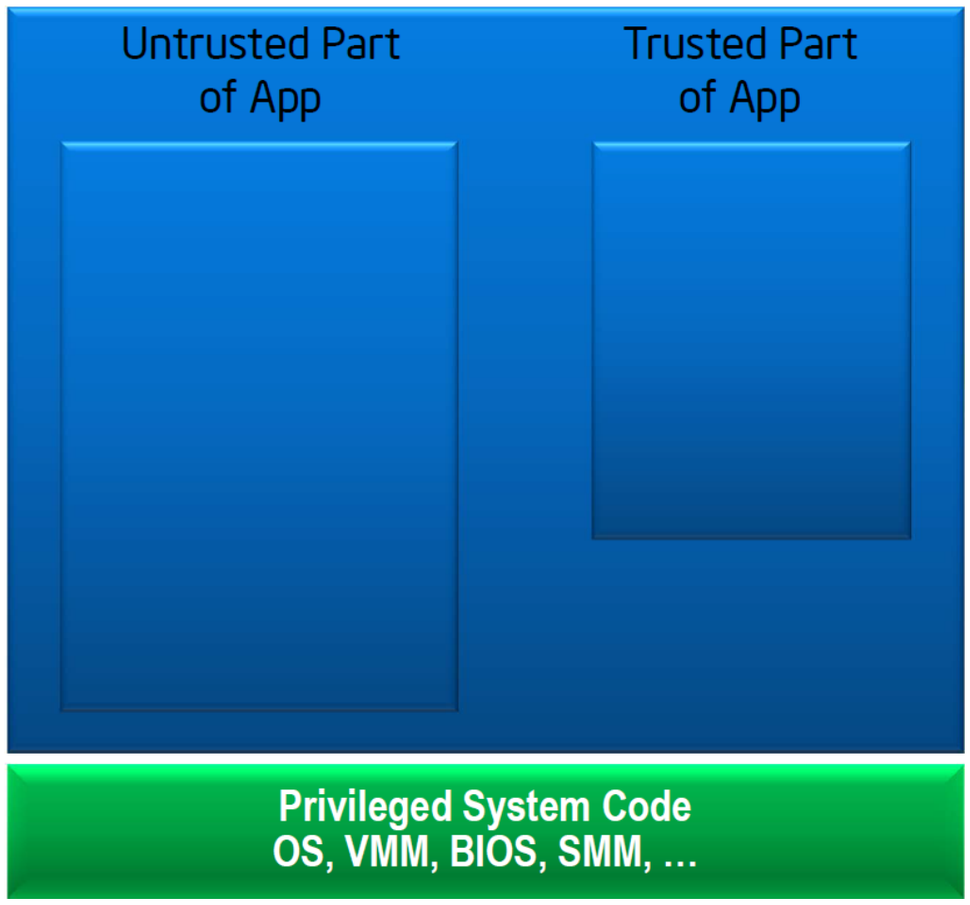
- There are other implementations of TEEs
  - Intel TDX, Intel SGX, Nvidia H100 GPU, ARM TrustZone, AWS Nitro, AMD SEV … (not exhaustive) and research prototypes…
- We study SGX because there is plenty learning materials
- Our goal is to **learn enough about the principles to use TEE in a productive way**
  - What we learn should be fundamental & generalizable.

Things to know

1. Memory isolation **<- let's look at this!**
2. Remote attestation <- saw it in demo
3. Sealing <- not so different from TPM
4. Life cycle of an enclave <- a bit tedious

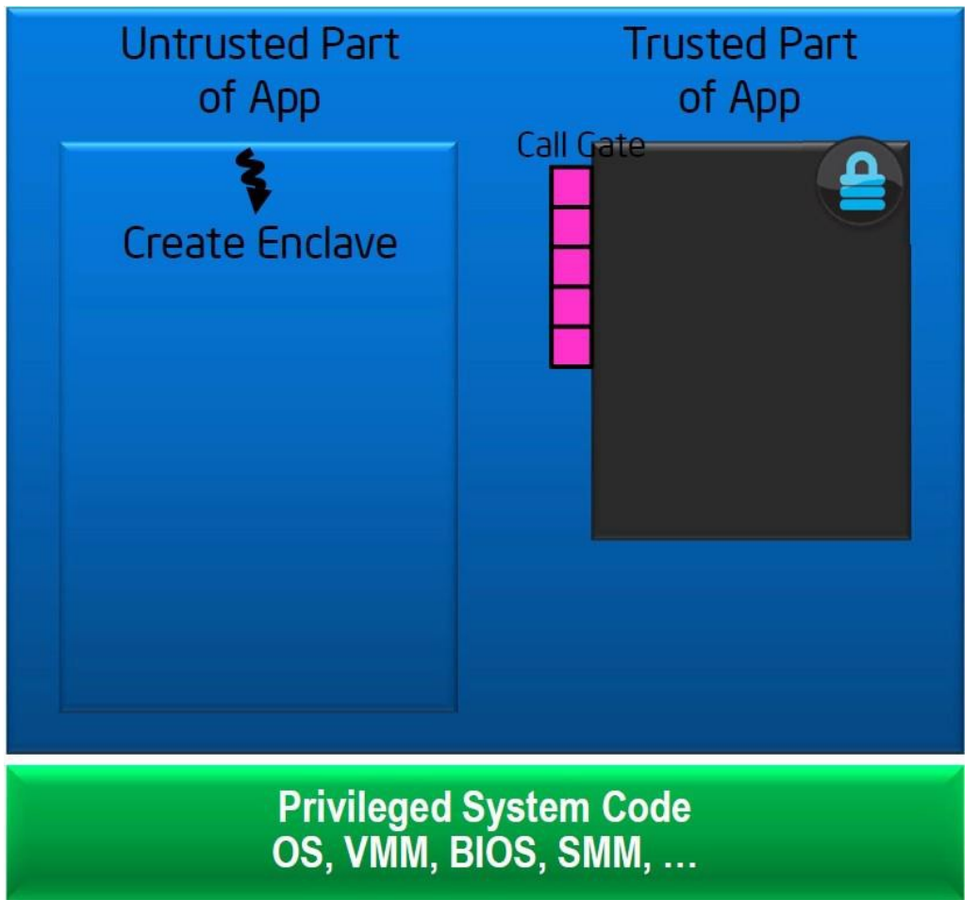
SGX Programming Model

- **Development:** App is built with **trusted** and **untrusted** parts
- Trusted part: security sensitive code to run in TEEs
- Untrusted part: creating & destroying trusted part, passes messages to/from it
- Ideally, trusted part should be as small as possible.
  - e.g., ideally just the LLM itself; in practice, also includes the entire docker stack (for convenience)



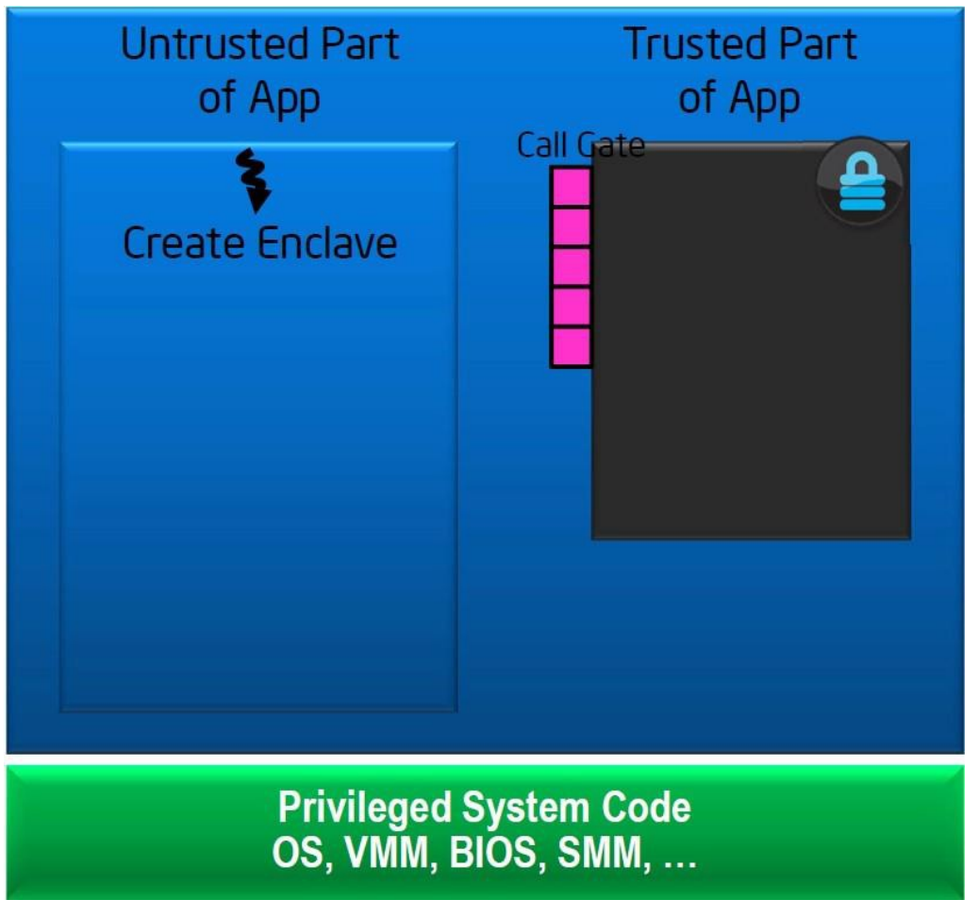
SGX Programming Model

- **Development:** App is built with trusted and untrusted parts
- **Initialization:** Untrusted part loads trusted part in protected memory
  - E.g., untrusted part = the code that loads LLMs from disk
  - Trusted part = LLM itself
- Trusted part exposes APIs for untrusted part to call
  - e.g., LLM inferences



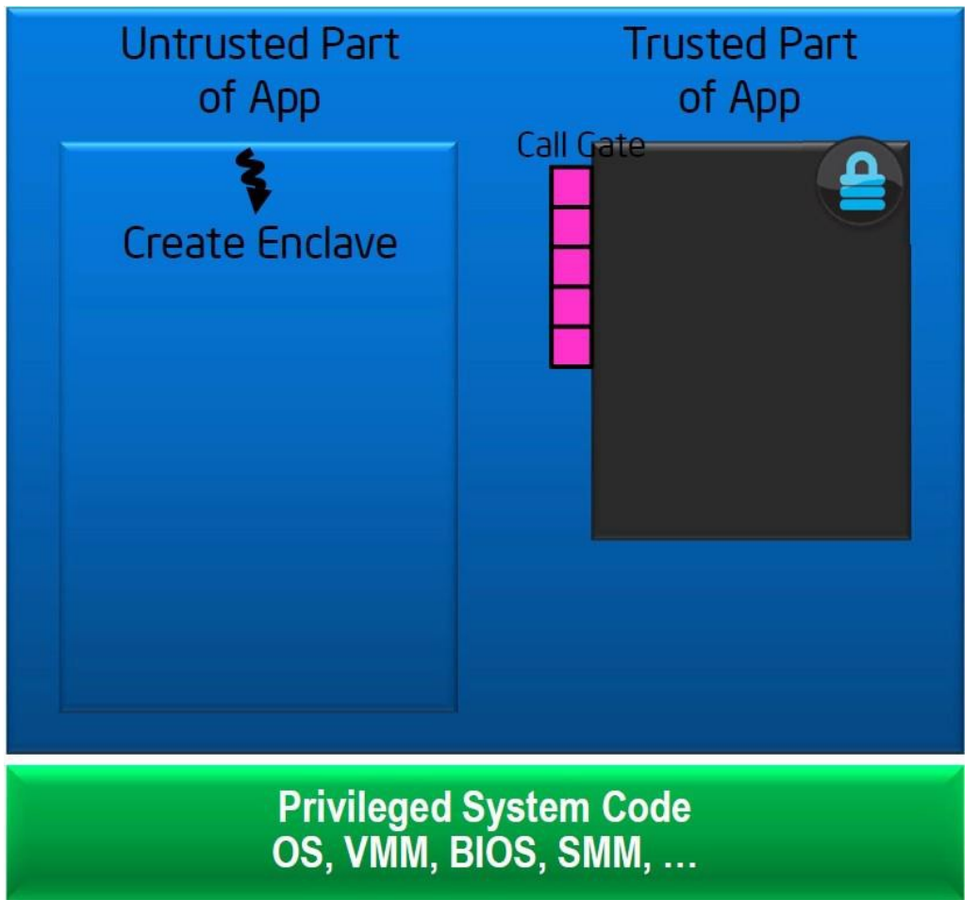
SGX Programming Model

- **Development:** App is built with trusted and untrusted parts
- **Initialization:** Untrusted part loads trusted part in protected memory
  - E.g., untrusted part = the code that loads LLMs from disk
  - Trusted part = LLM itself
- Trusted part exposes APIs for untrusted part to call
  - e.g., LLM inferences



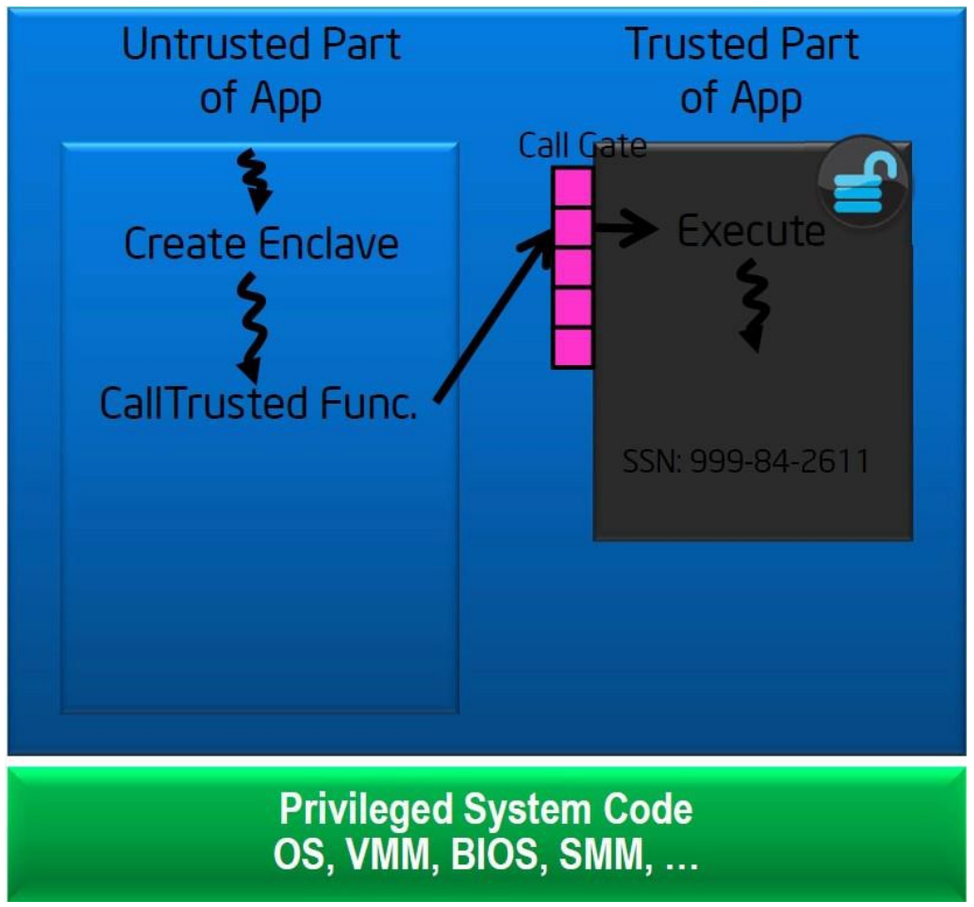
SGX Programming Model

- **Attestation:**
  - At any time after initialization, user can request for an attestation
  - Served by an API exposed by trusted part



SGX Programming Model

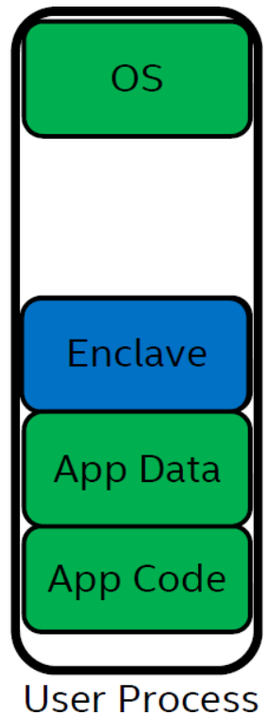
- **Call into enclave:** Untrusted app invokes exposed APIs
  - E.g., to serve LLM request upon user request
- Now CPU runs trusted code in "**enclave mode**"; allow code to access protected memory
- **Exit** enclave model when API call finishes



1) Memory isolation in SGX

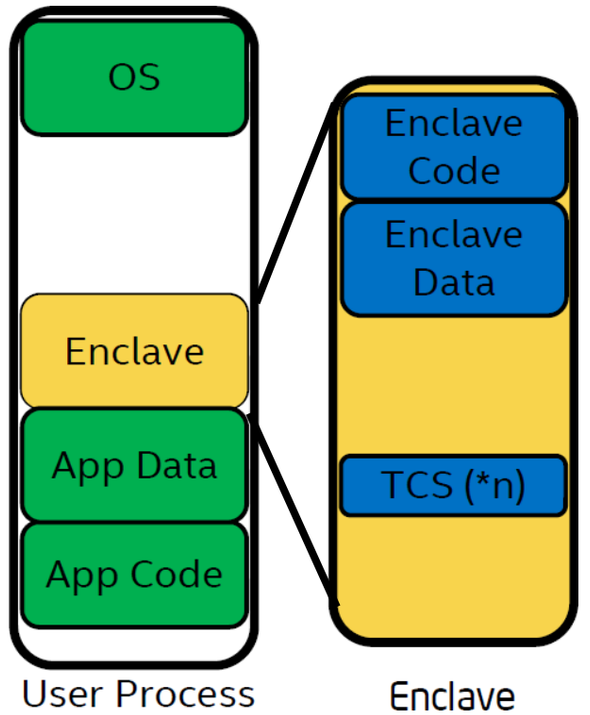
Virtual memory layout

- A secure Enclave is essentially a protected memory area

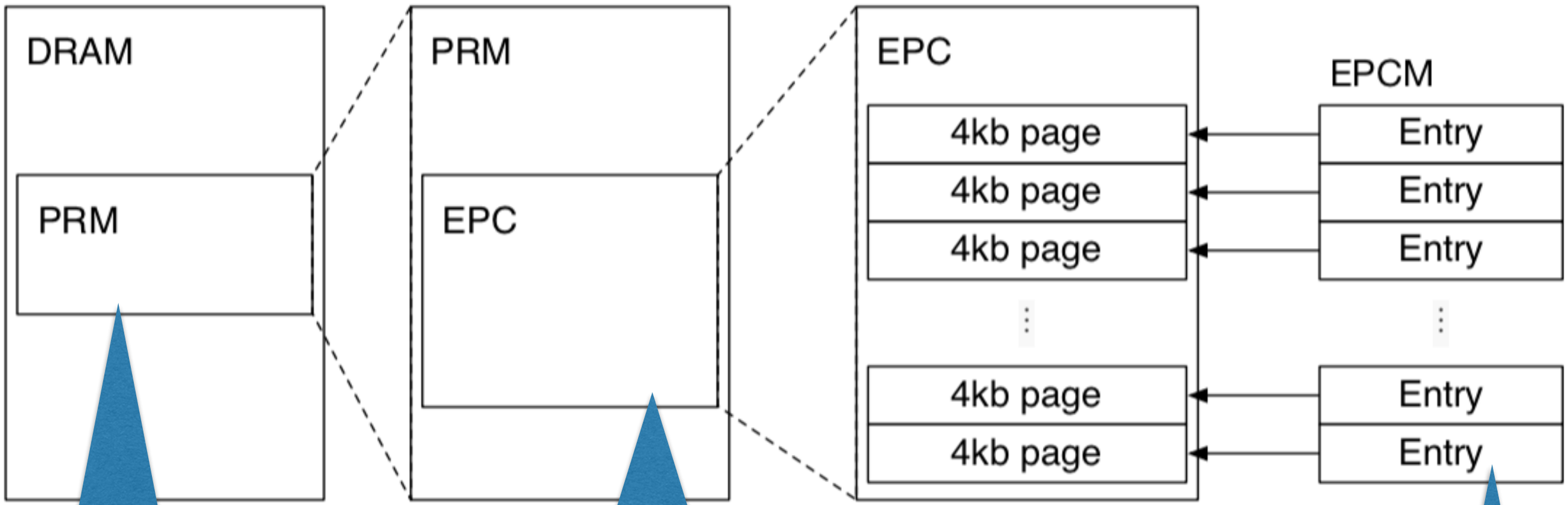


Virtual memory layout

- A secure **enclave** is essentially a protected memory area
  - Code
  - Data
  - Metadata (many types)



Physical memory layouts



Processor Reserved Memory (PRM): a subset of DRAM that **cannot** be directly accessed by other software (e.g., OS).

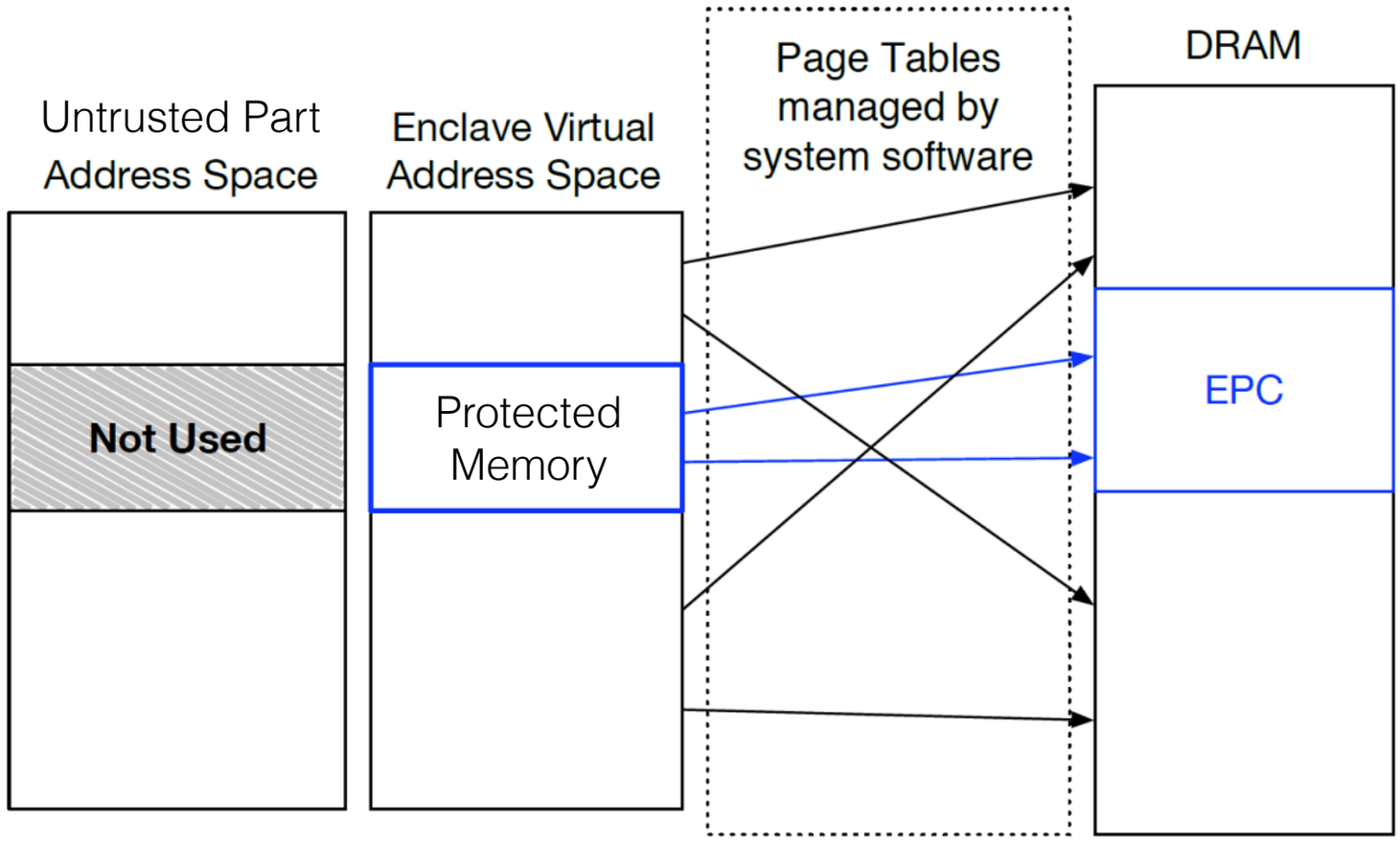
- Enclave Page Cache (EPC): where enclave pages resides
- EPC is split into 4 KB pages

- Security info about each EPC page
- One EPCM entry per EPC page

Virtual to physical translation

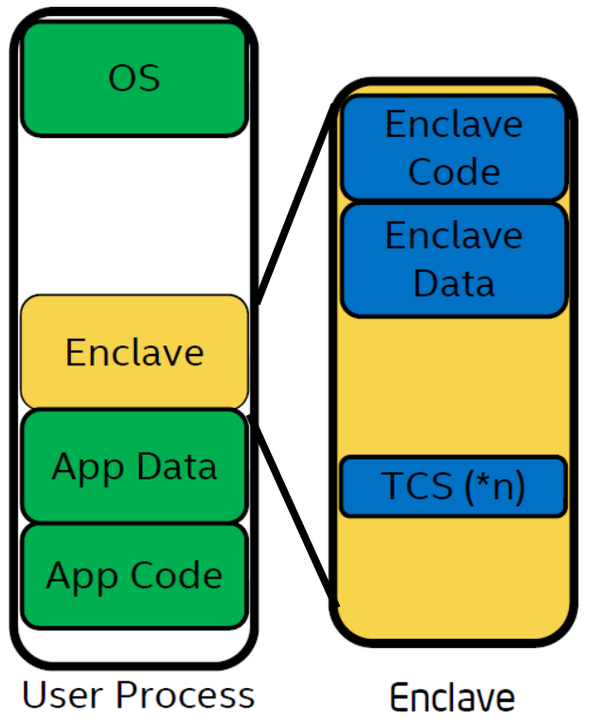
(§5.2.1)

- virtual address space of untrusted part & parted part



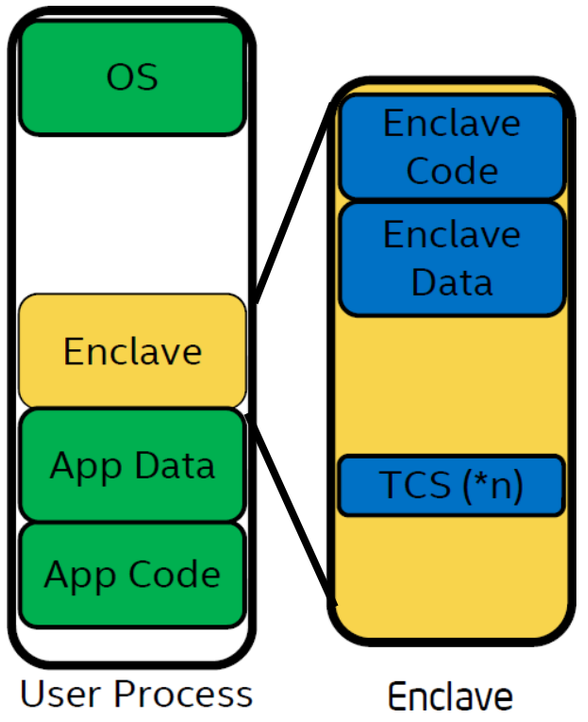
Memory Isolation

- Security goals: enclave memory pages can only be accessed by its **owner enclave**
  - Enclave A can't access enclave B's memory
  - OS can't access *any* enclave's memory



Attacks by OS

- Mapping from virtual to physical memory is managed by OS
  - Rationale: minimizes changes required to system software
- Leads to **passive** and **active attacks**
- A good amount of the complexity in SGX's design can be attributed to the need to prevent these attacks.



Passive Attacks

- **Access pattern leakage**: OS observes an enclave's page-level access pattern (which pages are accessed)
  - Page = 4KB memory regions
  - Note: OS does not see the exact address (SGX zeroes out the offset bits)

Passive Attacks (Example)

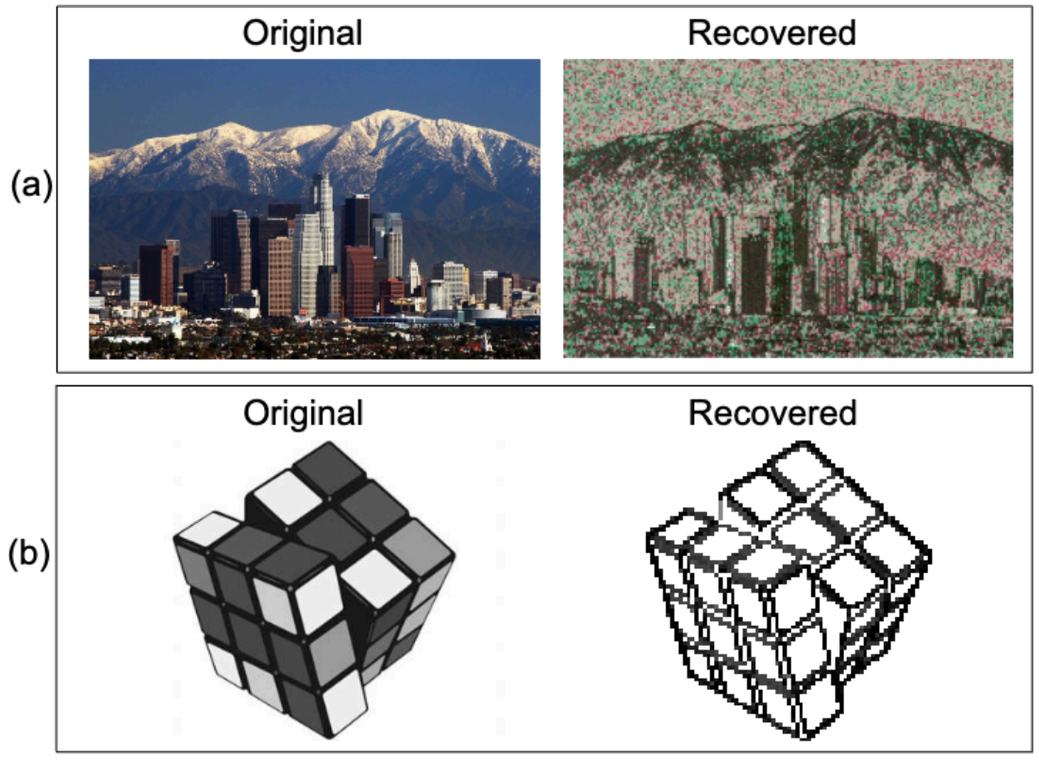
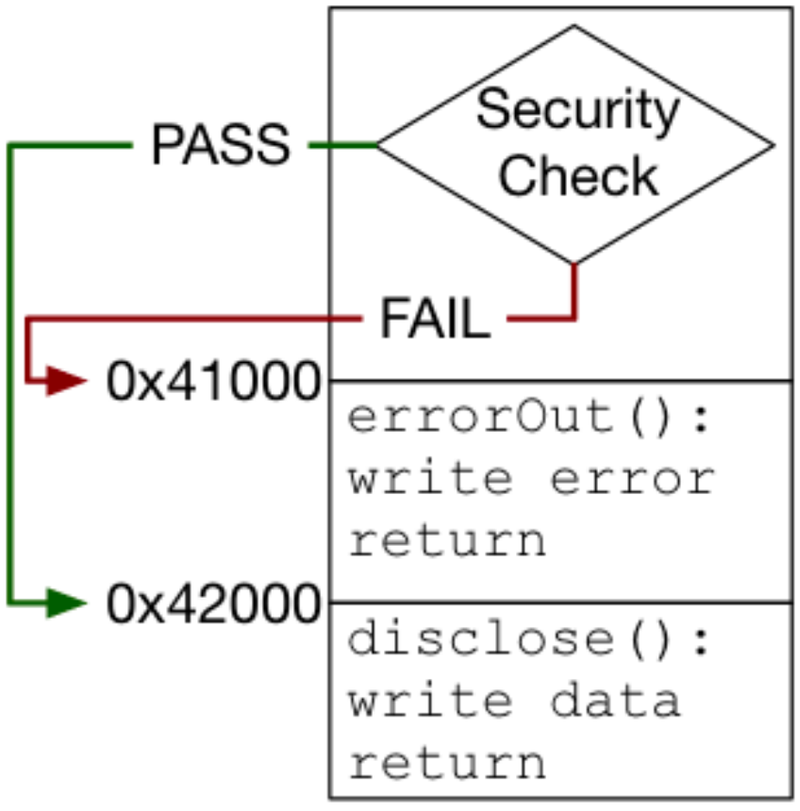


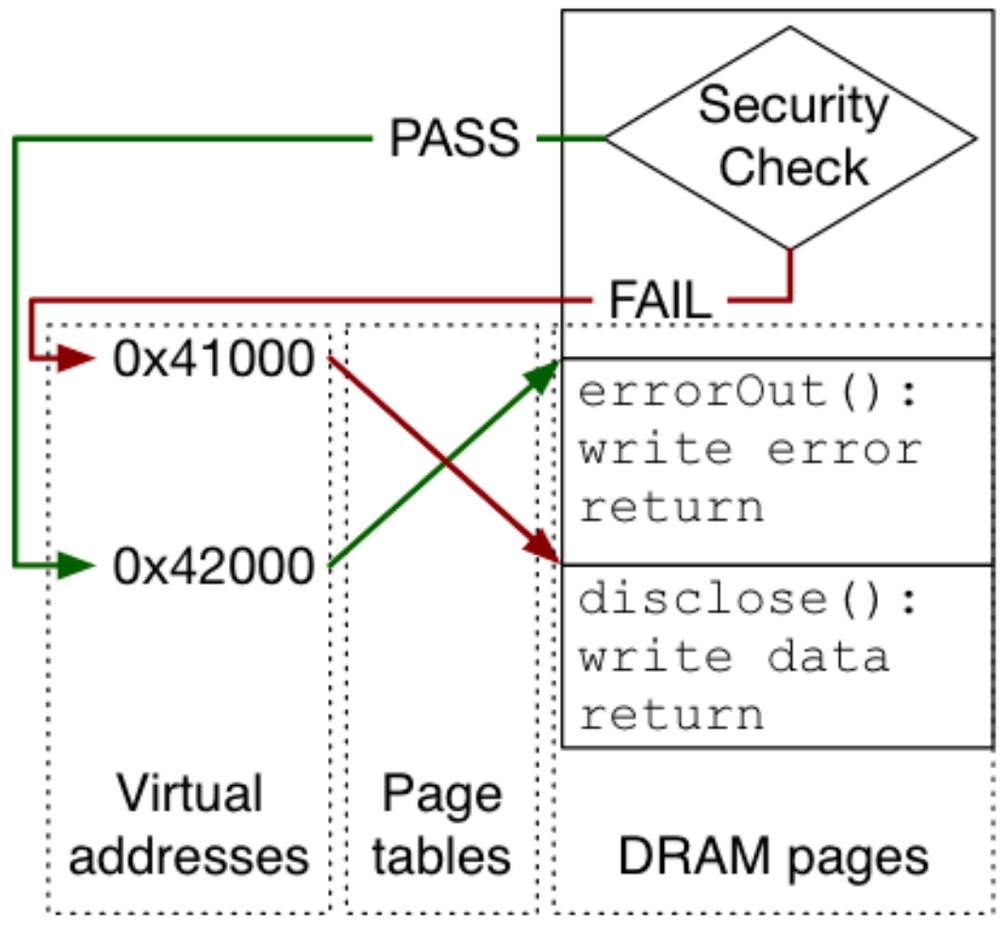
Fig. 11: A small sample of the images we used to test the libjpeg attack.

- XCP'15: the outline of a image can be recovered from how compression algorithm accesses memory.

"Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems", IEEE S&P 2015, by Y. Xu, W. Cui, and M. Peinado

Active Attacks I





Application: disclose only if security check passes.

Attack by malicious OS: map to wrong physical memory addresses!

Active Attacks I

- **How to prevent?**
- SGX tracks virtual address of each page in hardware
- When a EPC page is allocated, it's *intended virtual address* is record in EPCM entry

Field	Bits	Description
ADDRESS	48	the virtual address used to access this page
R	1	allow reads by enclave code
W	1	allow writes by enclave code
X	1	allow execution of code inside the page, inside enclave

Used for address translation

- **What if OS uses wrong *intended* virtual addresses?**
- Attestation will fail.

Other stuff being tracked

- One EPCM per EPC page
- To check that the (untrusted) OS is behaving as expected.

Field	Bits	Description
ADDRESS	48	the virtual address used to access this page
R	1	allow reads by enclave code
W	1	allow writes by enclave code
X	1	allow execution of code inside the page, inside enclave

Used for address translation

Permission of this page

Field	Bits	Description
VALID	1	0 for un-allocated EPC pages
PT	8	page type
ENCLAVESECS		identifies the enclave owning the page

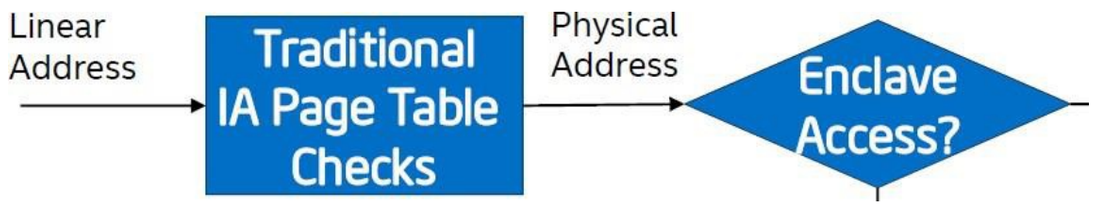
If this bit is 1, allocating it again causes an error.

The owner enclave of this page

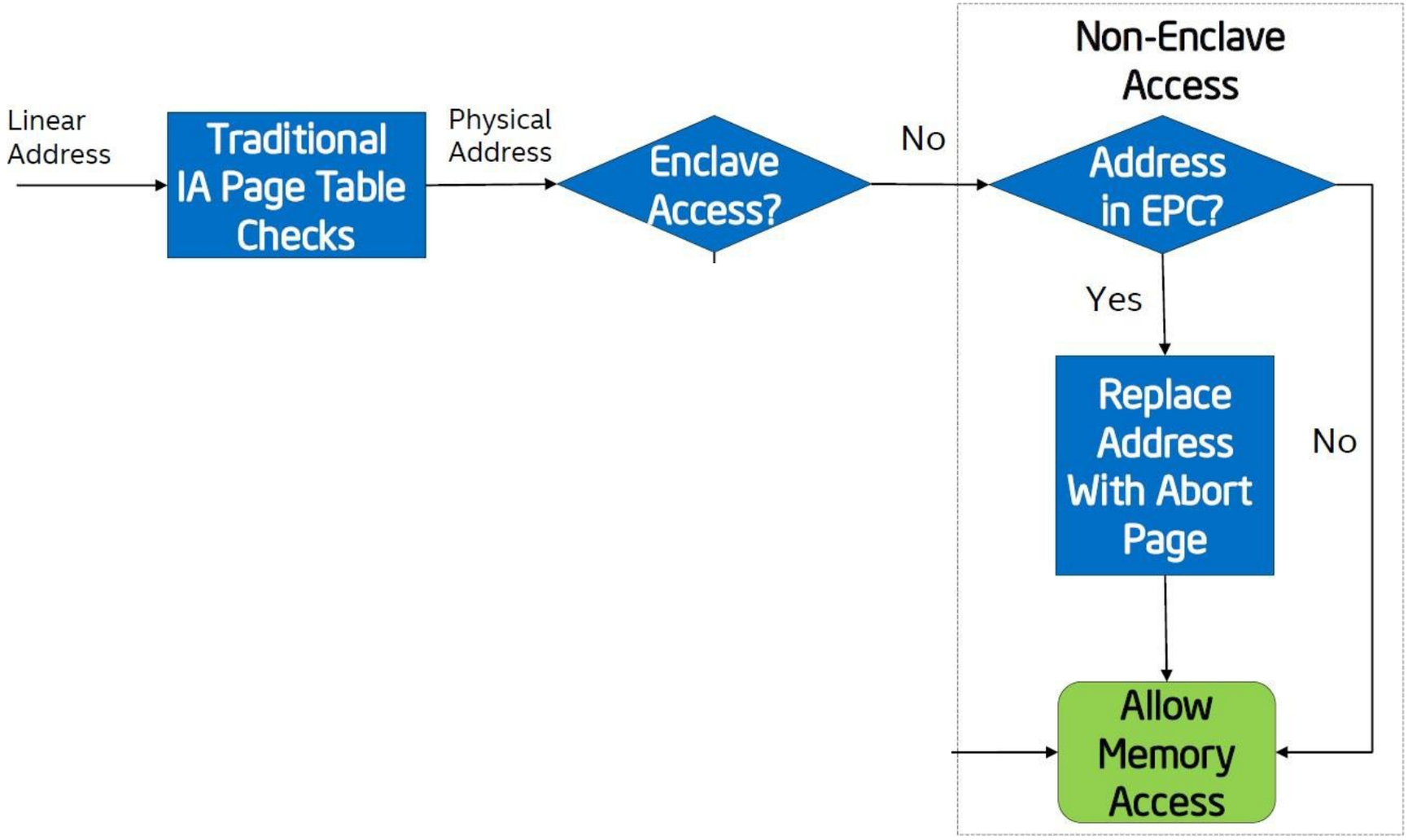
Address translation in SGX

Linear = virtual

Address translation in SGX

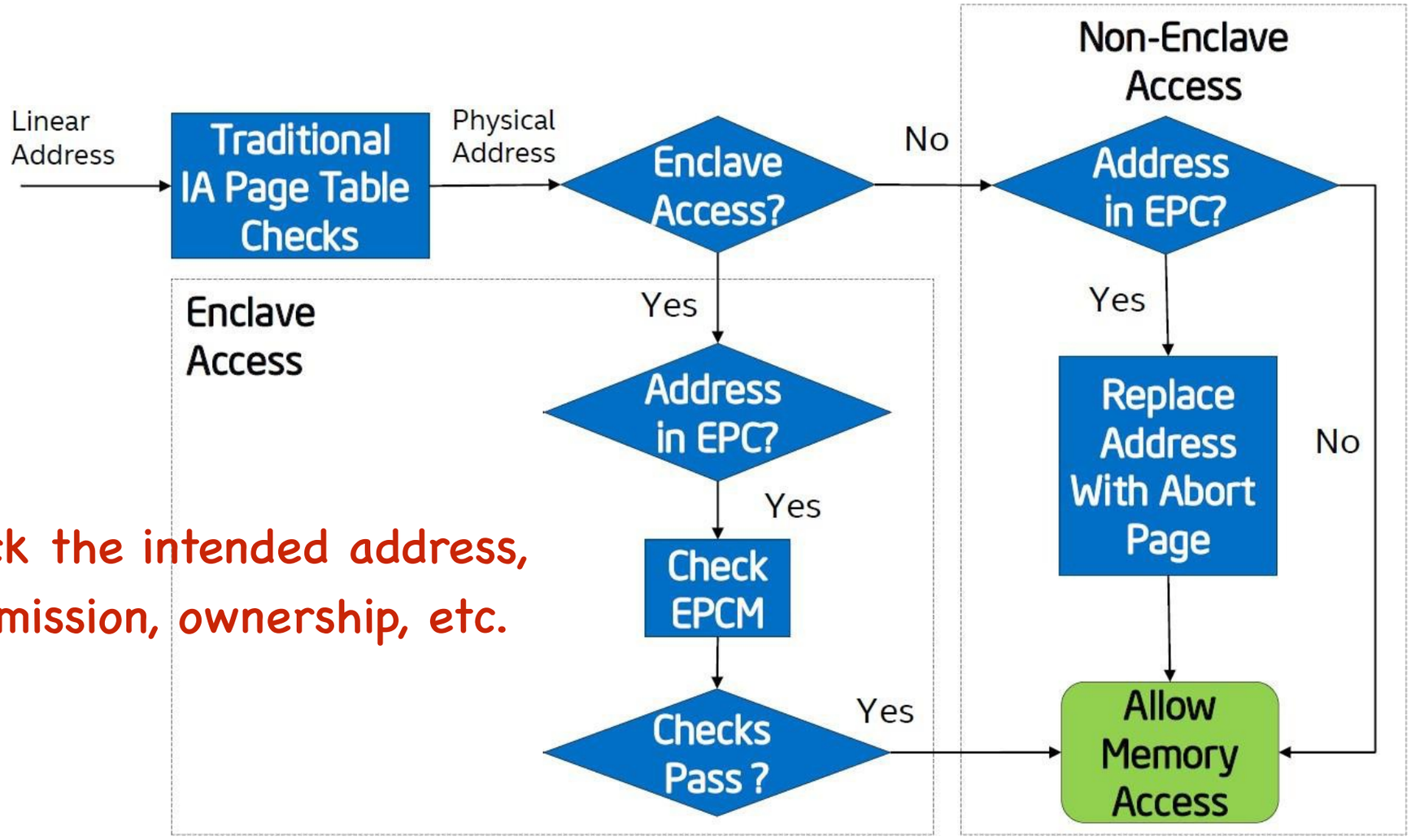


Address translation in SGX



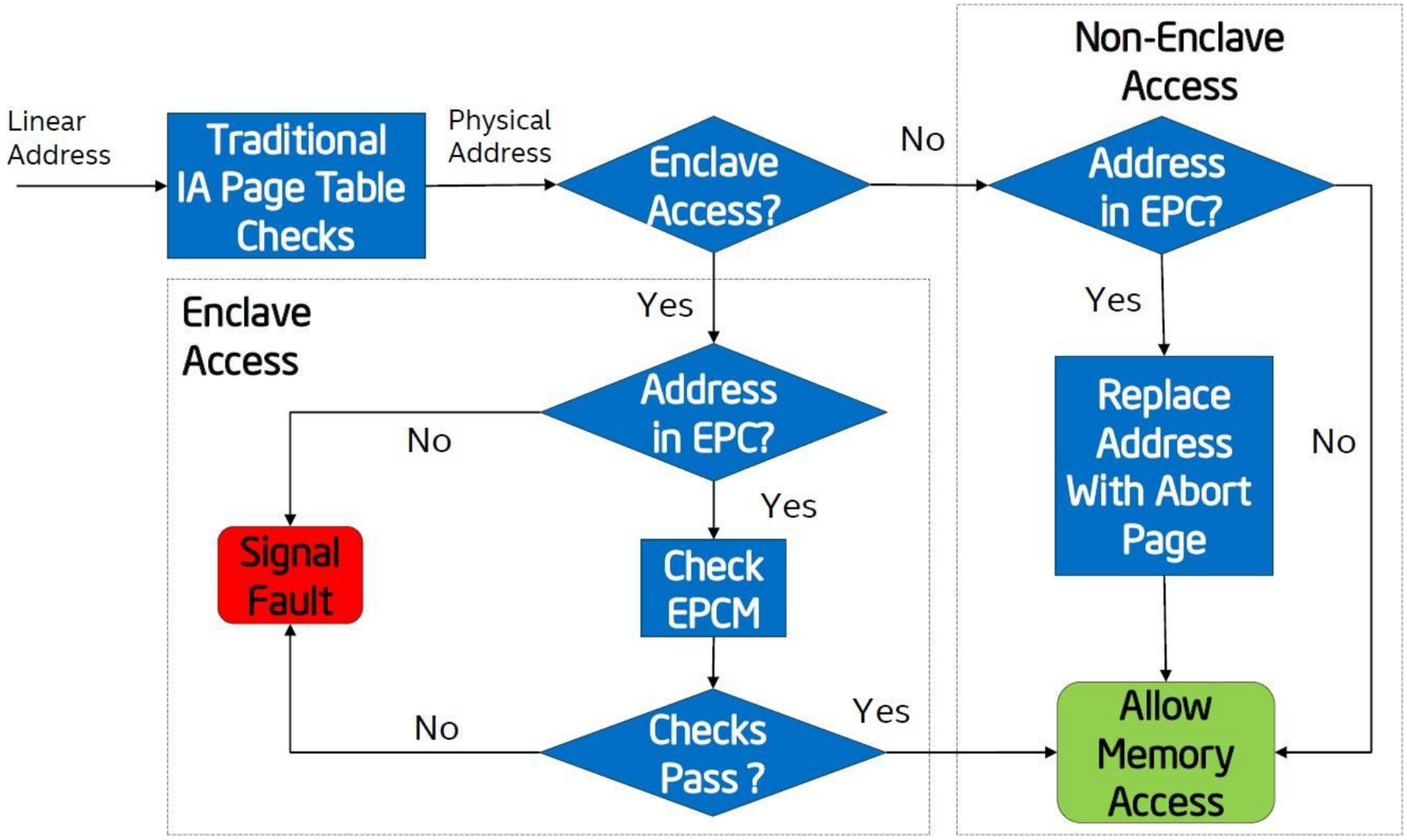
Non-enclave code can't access EPC

Address translation in SGX



Check the intended address, permission, ownership, etc.

Address translation in SGX



Attacks during **Page Swapping**

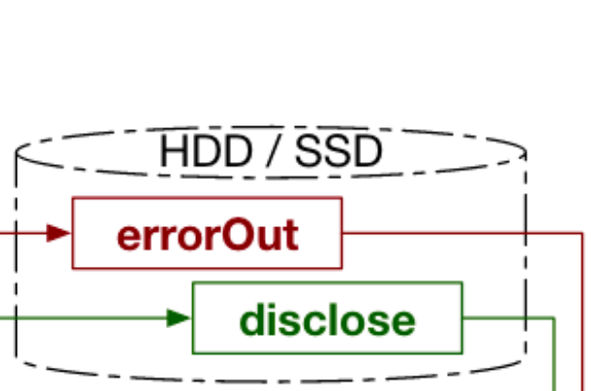
- Swap: remove EPC pages, place into unprotected storage, and restore it later.
  - Aka *eviction*
- Why? To support applications that uses more memory than EPC
- Gives OS an vector to attack…

Active Attacks II

- The malicious OS evicts the two pages to disk.
- OS swaps the two pages' contents
- OS brings them back into DRAM.

Page tables and DRAM before swapping

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



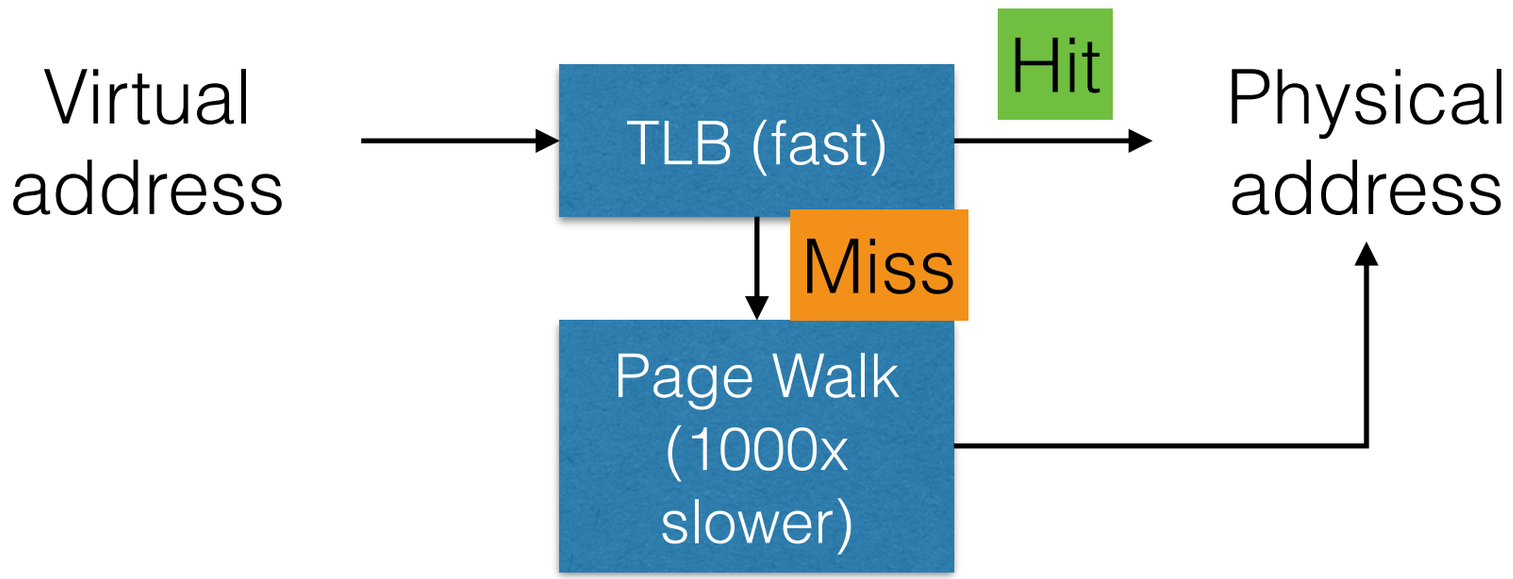
Page tables and DRAM after swapping

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

Solution: cryptographically binding each evicted page to the virtual address (using AEAD cipher).

Active Attacks III

- Page walk is expensive (1000+ cycles) so the mapping is cached in TLB (a cache on CPU).



- Intel CPUs rely on **OS** to inform cores to update TLB upon page table changes
- We can’t count on the OS for doing that anymore!

Active Attacks III

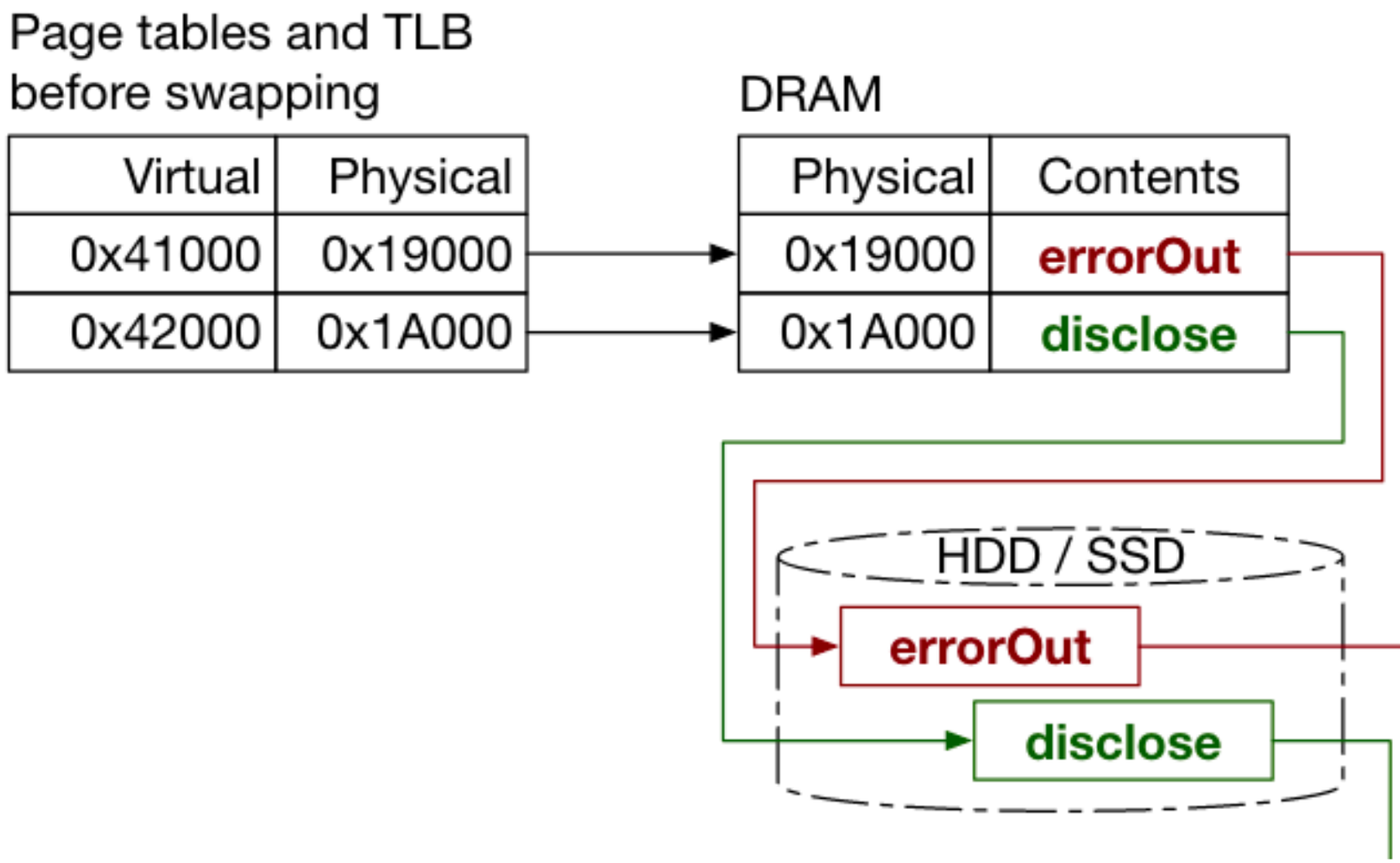
Page tables and TLB
before swapping

Virtual	Physical
0x41000	0x19000
0x42000	0x1A000

DRAM

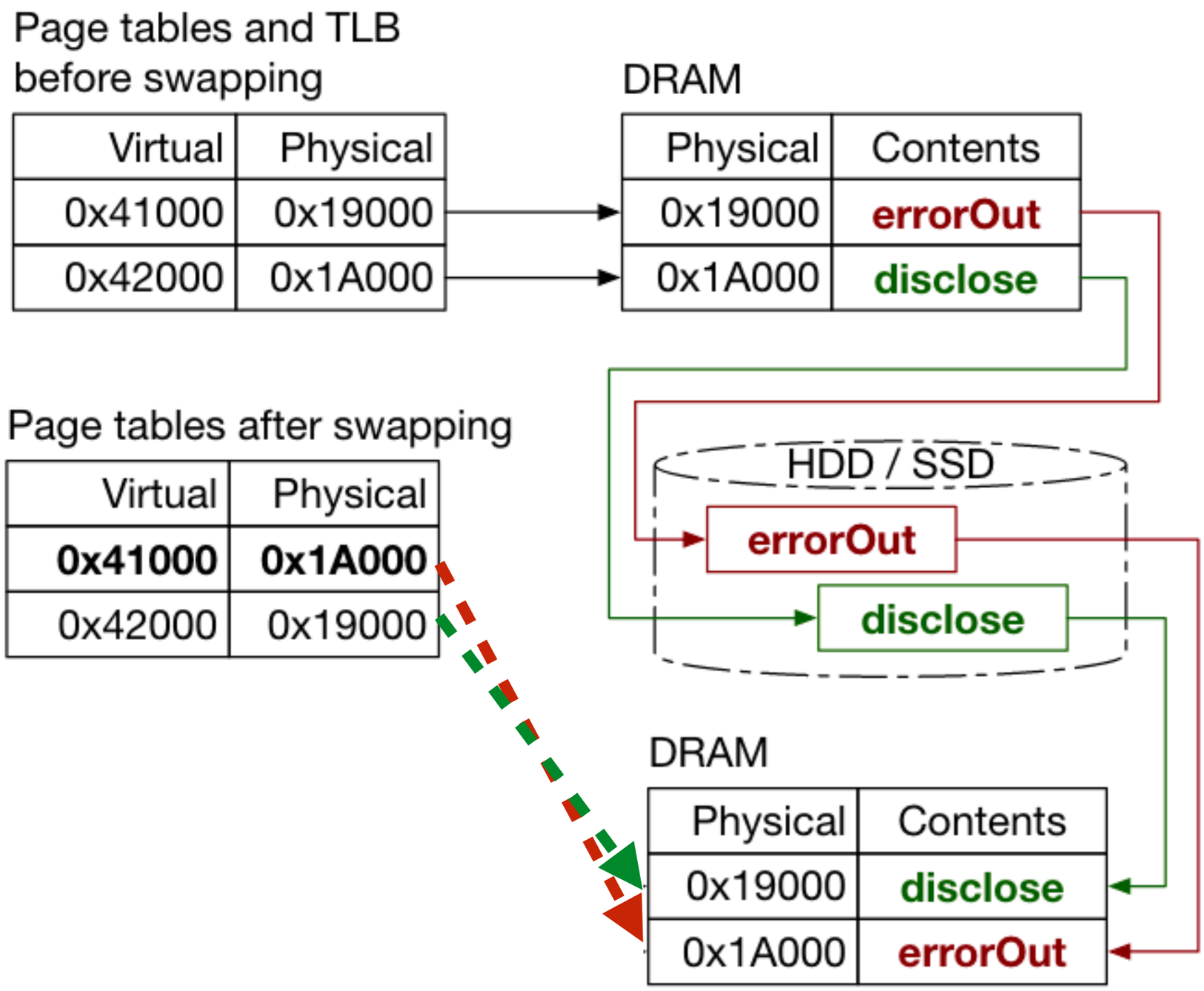
Physical	Contents
0x19000	**errorOut**
0x1A000	**disclose**

Active Attacks III



Active Attacks III

Up-to-date mapping



Solution: SGX flushes TLB when enclave page is brought back…

Active Attacks III

Page tables and TLB
before swapping

Virtual	Physical
0x41000	0x19000
0x42000	0x1A000

DRAM

Physical	Contents
0x19000	**errorOut**
0x1A000	**disclose**

HDD / SSD

errorOut

disclose

Up-to-date mapping

Page tables after swapping

Virtual	Physical
0x41000	**0x1A000**
0x42000	0x19000

Wrong mapping!

Stale TLB after swapping

Virtual	Physical
0x41000	**0x19000**
0x42000	0x1A000

DRAM

Physical	Contents
0x19000	**disclose**
0x1A000	**errorOut**

More attacks

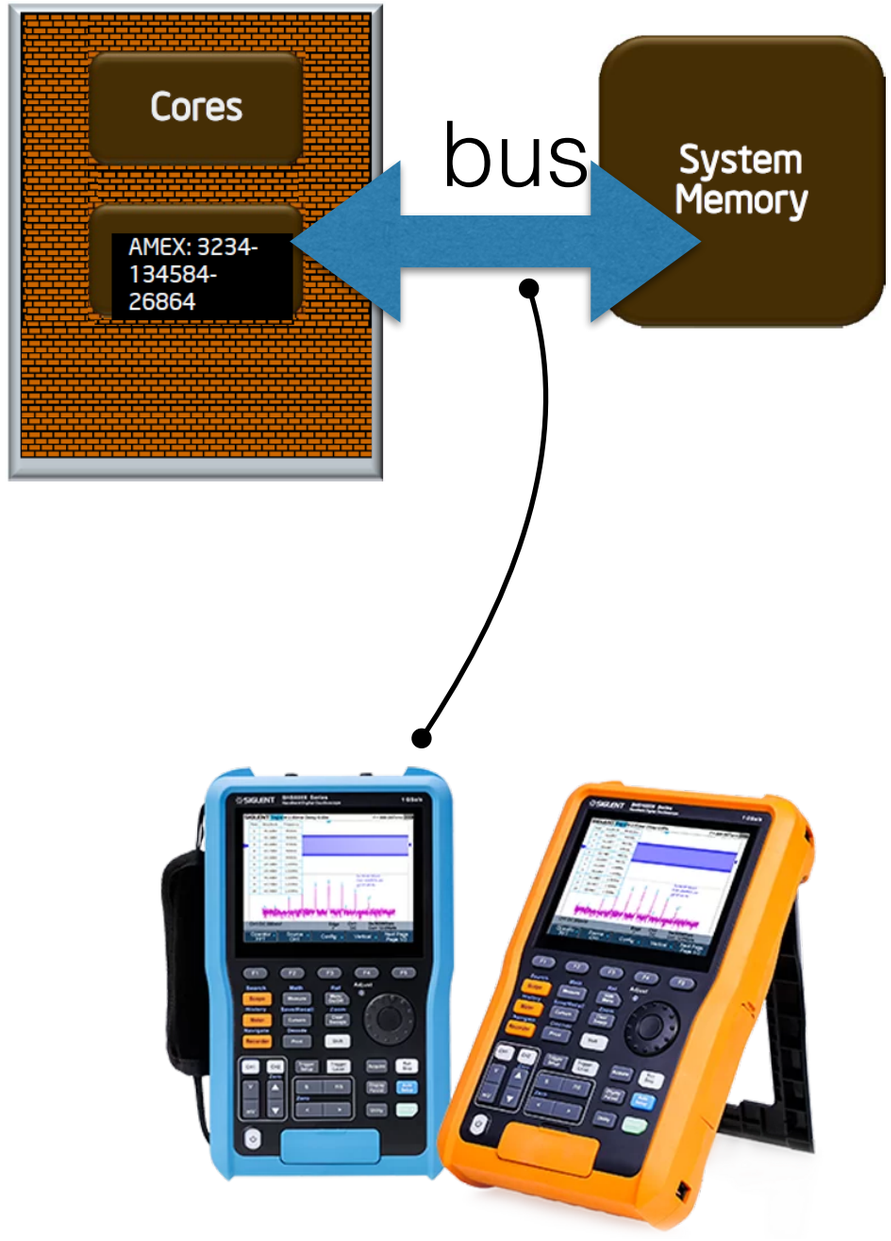
- Say password is used to authenticate access to an SGX enclave. Enclave locks an account after 10 failed authentication attempts.
- Here is a **replay attack**
  - OS locates the page storing the counter.
  - When counter is 0, evict the page to disk
  - Try 9 passwords
  - Evict counter again, and replace it with C
  - Load page back, now counter is 0.

SGX had to deal with all of these concerns

- Introduced instructions to ensure TLB shutdown (EBLOCK, ETRACK)
- Introduced special EPC page to store version number for every evicted pages
  - anti-replay
- Encrypts the evicted page using AES-GCM with its EPCM fields as metadata
  - Including its virtual address (c.f. active attack II)

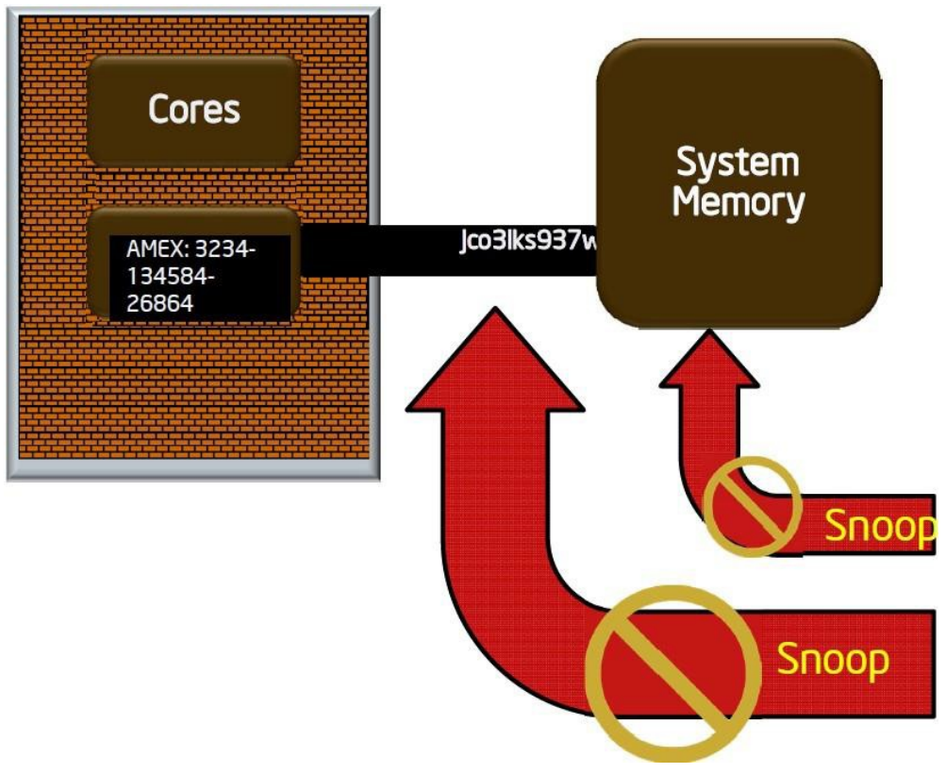
Plus, there is memory encryption

- Assumption: data inside CPU package is not directly visible to attackers
  - Registers, caches, etc.
- A *physical* attacker can snoop on memory bus



Memory Encryption

- On top of everything we have seen, SGX pages are encrypted while in DRAM as well
  - done by Intel MEE (Memory Encryption Engine)
  - Uses a Merkle Tree for integrity of memory blocks.



Summary of SGX's security guarantees

Threat model

- Malicious system software (OS, hypervisor, firmware, etc)
- Apt for cloud computing settings
- Assuming no physical attacks against CPU chips
  - E.g., e-fuses may be readable by a high-resolution microscope.
- Also excludes **side-channel attacks** (next lecture) and leaves them for developers to take care of.

Privileged Software Attacks

- SGX prevents malicious software from directly reading or from modifying the EPC pages, e.g., via
  - Malicious page mapping
  - Swapping content on disk
  - Stale TLB
- But system software can still attack side channel attacks (**next lecture**)

Acknowledgments

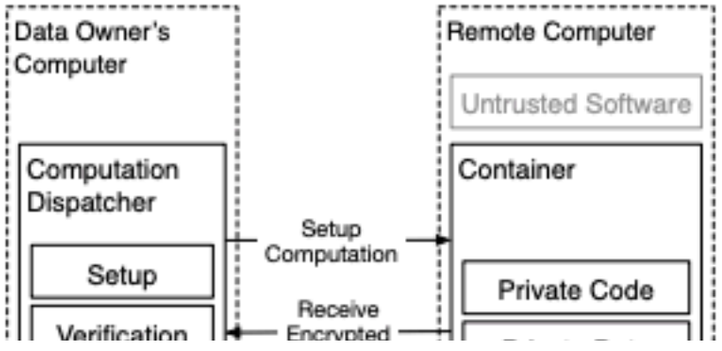
- Intel SGX Tutorial (Reference Number: 332680-002) presented at ISCA 2015
- Slides from ECE 4451 by Marten van Dijk
- “Intel SGX Explained”, Victor Costan and Srinivas Devadas, CSAIL MIT

Intel SGX Explained

Victor Costan and Srinivas Devadas
victor@costan.us, devadas@mit.edu
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

ABSTRACT

Intel’s Software Guard Extensions (SGX) is a set of extensions to the Intel architecture that aims to provide integrity and confidentiality guarantees to security-sensitive computation performed on a computer where all the privileged software (kernel, hypervisor, etc) is potentially malicious.



A 118-page gold mine for SGX study

https://eprint.iacr.org/2016/086.pdf