CSE 5095 & ECE 4451 & ECE 5451 – Spring 2017

Lecture 7a

# Sanctum: Minimal HW Extensions for Strong SW Isolation

**Marten van Dijk**

**Syed Kamran Haider, Chenglu Jin, Phuong Ha Nguyen**

Department of Electrical & Computer Engineering
University of Connecticut

UCONN

Based on the corresponding paper authored by V. Coston, I. Lebedev, and S. Devadas published at Usenix Security 2016 as well as V. Costan's PhD thesis

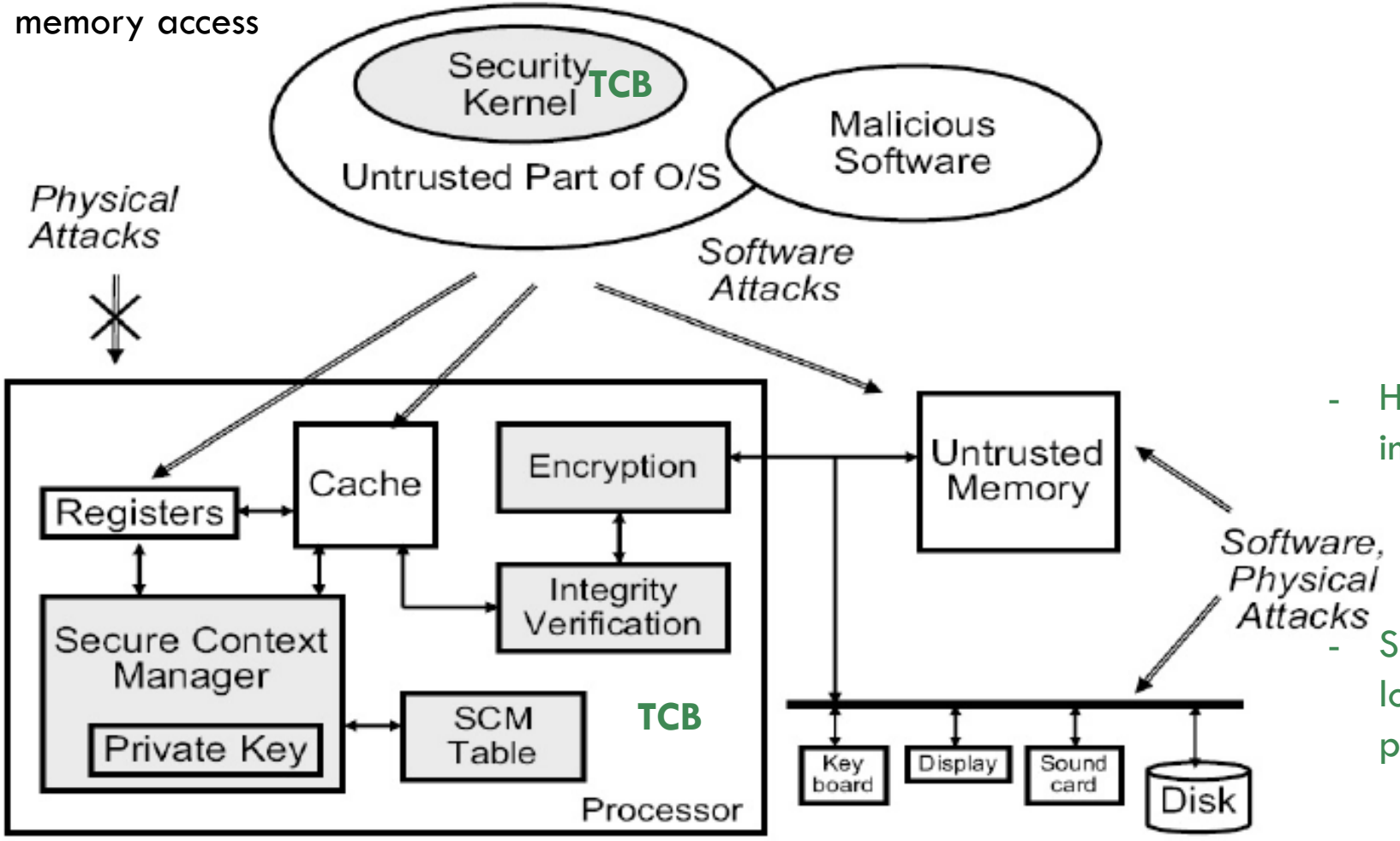
# Outline

- Adversarial Models AEGIS, Intel SGX, and Sanctum revisited
- Sanctum: HW Isolation – For remote attacks, no need for encryption and memory integrity checking!
- Sanctum: Replaceable security software (security monitor) – Gives control!
- Sanctum: Protection against cache side channel attacks – and still practical!

# AEGIS

SCM: Specialized HW (see MEE in the MC in SGX) that ensures protection of each process:

- Computes hash of program and data
- Assigns a Secure Process ID for on-chip memory access
- Performs memory integrity checking for off-chip memory access




Adversarial Model of AEGIS:

- Adversary can launch remote SW attacks and physical tampering of main memory
  - No attention is given to how observation of access patterns can leak sensitive information
  - No protection against the cache covert channel or other leakage from not properly flushing buffers
  - No protection against access to DRAM by peripherals
- HW TCB = CPU chip (with caches, mem. interface), Package
  - AEGIS forbids access by untrusted OS to reserved DRAM for Secure Containers (think Enclaves)
- SW TCB = App. Mod. (in Secure Enclave @ lowest priv. level), Sec. Kernel (@ highest priv. level in SMM) has no vulnerabilities
  - Allows multiple modules
  - Allows untrusted OS

Suh et al., "AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing", ICS, 2003

# Adversarial Models SGX and Sanctum

Adversarial Model SGX:

- Adversary can launch remote SW attacks and may actively alter/observe DRAM content (for the latter we need the MEE responsible for integrity checking and encryption):
  - The adversary is not attempting to derive information from access patterns to DRAM – either from actively observing the bus or leakage from page misses or from not properly flushing the branch history buffer (which can be easily fixed) or from the cache covert channel
- HW TCB = CPU chip (with caches, mem. interface), Package
  - Access to DRAM by peripherals is controlled by a trusted MC (as in Intel TXT) in the MEE on chip
  - SGX forbids access by untrusted OS to reserved DRAM for Secure Enclaves
- SW TCB = App. Mod. (in Secure Enclave @ lowest priv. level), SGX micro code (@ highest priv. level, higher than SMM's level which includes BIOS)
  - Allows multiple modules
  - Allows untrusted OS

Adversarial Model of Sanctum:

- Adversary can only launch remote SW attacks
  - Because of HW isolation no encryption is necessary
- HW TCB = CPU chip (with caches, mem. interface), Package
  - Access to DRAM by peripherals is controlled by a trusted MCU (as in Intel SGX)
  - Sanctum forbids access by untrusted OS to reserved DRAM for Secure Enclaves
- SW TCB = App. Mod. (in Secure Enclave @ lowest priv. level), Sec. Monitor (@ highest priv. level)
  - Allows multiple modules (Sanctum prevents cache timing channel attacks using locality preserving cache-coloring)
  - Allows untrusted OS

**Intel SGX follows AEGIS' blueprint !**

# Outline

- Adversarial Models AEGIS, Intel SGX, and Sanctum revisited
- Sanctum: HW Isolation – For remote attacks, no need for encryption and memory integrity checking!
- Sanctum: Replaceable security software (security monitor) – Gives control!
- Sanctum: Protection against cache side channel attacks – and still practical!

# HW Isolation

- HW isolation of secure enclaves protects against a remote adversary
  - MEE/MC protects against an untrusted OS accessing DRAM through a peripheral
  - Untrusted OS cannot access reserved enclave DRAM
  - Untrusted OS cannot observe access pattern die to page misses …
    - There is no automatic paging.
    - If an enclave collaborates with an OS to implement paging, the OS sees anything the enclave explicitly reveals. Enclaves cannot perform I/O directly.
    - Enclaves do not have to store anything in untrusted DRAM to work, and are able to protect their page-level access pattern if they absolutely must page. We will discuss Oblivious RAM (ORAM) next lecture – this technique can also be used for page level access by an enclave at the trusted enclave DRAM memory to untrusted DRAM boundary
      - A nice implementation exercise for the Sanctum framework
      - Notice this can also be used in Intel SGX (albeit Intel SGX may swap out an enclave's page on a page miss by another enclave because there is automatic paging)
  - Sanctum can easily implement flushing of the branch history table at an EEXIT and AEX
  - Prevent cache side channel attacks by cache coloring (explained later)
  - Prevent timing side channel leakage (in particular, by the security monitor)

# Paging

- Writing enclaved applications is somewhat distinct from other software: an enclave is not automatically managed by an OS
- Enclaves' isolation from supervisor software means that enclaves do not offer transparent handling of page faults: an OS would need to be involved in any interaction with the disk, and interactions with the OS may leak private information.
- Instead, the security monitor routes all faults to a handler in the enclave itself!
  - The enclave is then able to make a decision as to its handling of the fault.
  - Suicide is a reasonable option for many enclaves (the app under-budgeted its available memory, and does not wish to notify the OS of this fact, so it terminates).
  - If the enclaved program does wish to ask the OS to page to/from disk, it may choose to protect its privacy, for example by implementing a *page-level ORAM scheme* (and pay the associated performance overhead) including randomized *encryption* and *memory integrity checking*, hiding its page-level access pattern from the OS. This can become arbitrarily complex, including *periodic accesses* to protect the timing of page faults, which too may leak private information.
- The enclave can of course choose to leak all this information and collaborate with the OS to implement paging: the OS would manage some pages in its own memory, and the enclave would copy these to/from its private pages.
- A reasonable tactic is to avoid implementing any page table management in the enclave, and to use static enclave page tables prepared while the enclave is created.
  - In order to see enclaves as trusted code, a small TCB is desirable (decrypt, process, encrypt).
  - It makes a lot of sense to avoid implementing a large system within an enclave. Anything that requires paging sounds like a large system.

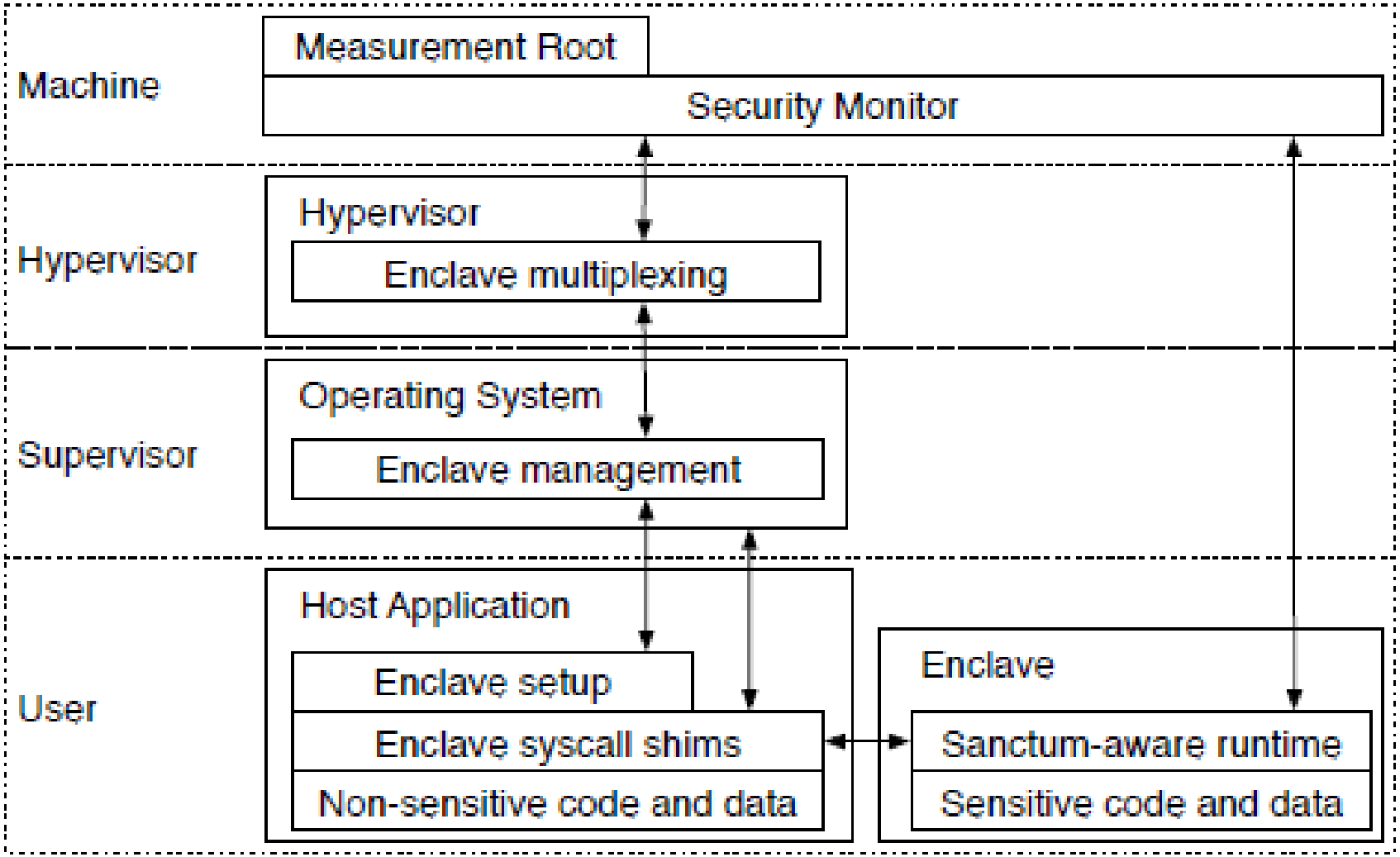
# Remote Adversary

- HW isolation implies that there is no need for
  - Encryption
  - Memory integrity checking
- Why?
  - The remote adversary (untrusted OS) cannot access and tamper with the reserved enclave memory!!
  - And cannot observe its access patterns (when access happens and to which address in memory)

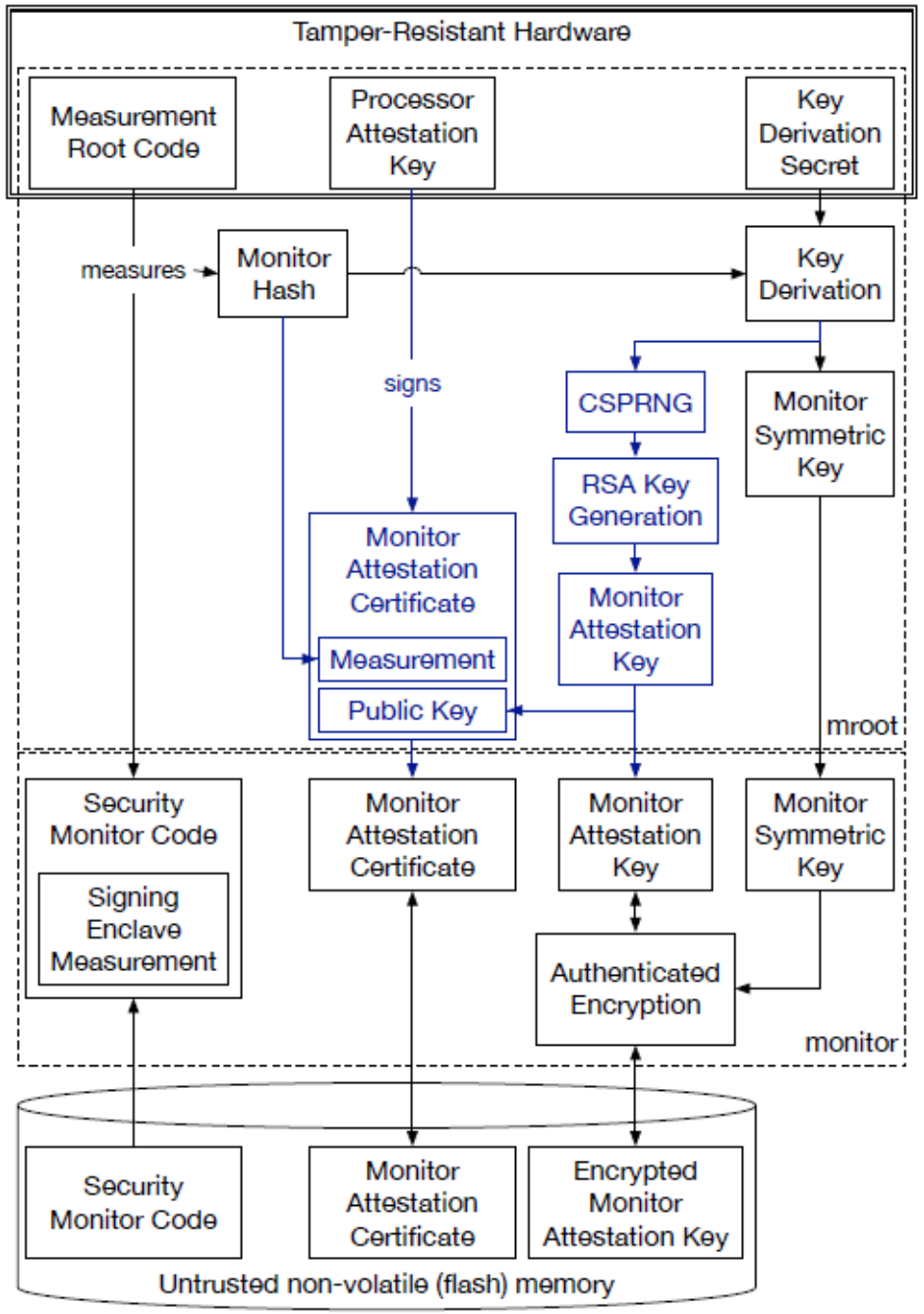
# Outline

- Adversarial Models AEGIS, Intel SGX, and Sanctum revisited
- Sanctum: HW Isolation – For remote attacks, no need for encryption and memory integrity checking!
- Sanctum: Replaceable security software (security monitor) – Gives control!
- Sanctum: Protection against cache side channel attacks – and still practical!

# Security Monitor

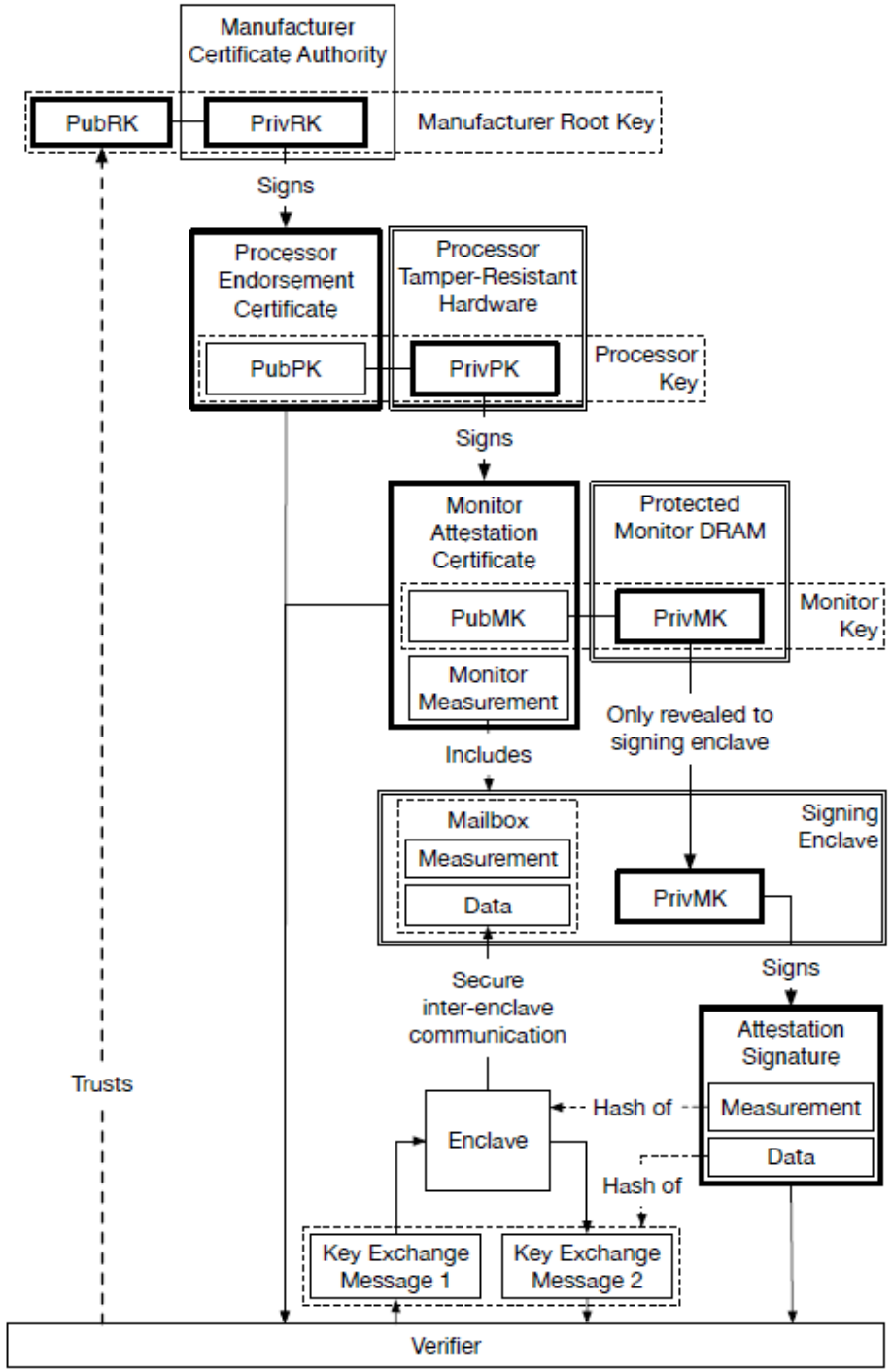



- SGX's microcode is replaced by a trusted software component, the security monitor, which runs at the highest privilege level an therefore is immune to compromised system software
- Management of computation resources is relegated to untrusted system software (as in SGX)




# The Measurement Root

Sanctum's root of trust is a measurement root routine burned into the CPU's ROM. This code reads the security monitor from flash memory and generates an attestation key and certificate based on the monitor's hash. Asymmetric key operations, colored in blue, are only performed the first time a monitor is used on a computer.



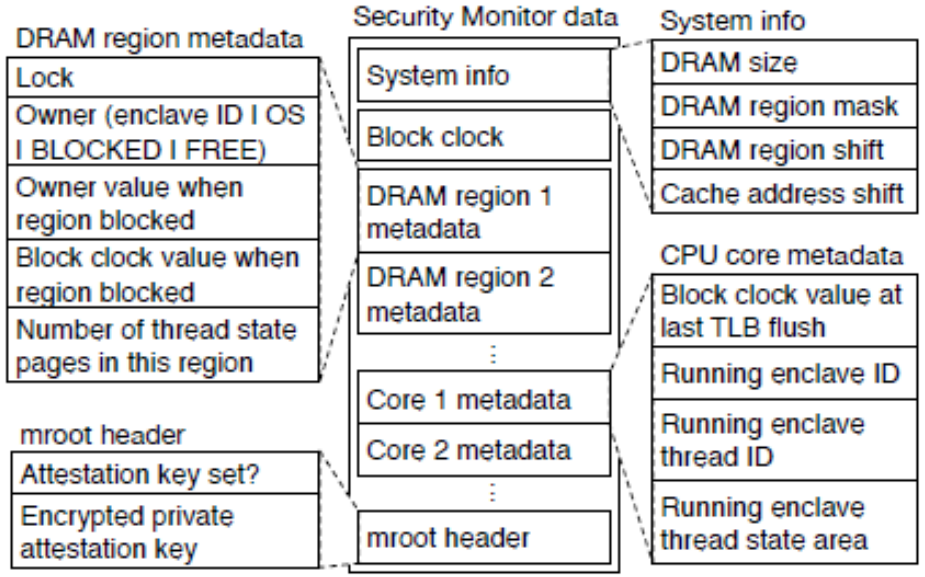

# The Signing Enclave

Certificate chain behind Sanctum's SW attestation signatures

- Trusted manufacturer signs the PubPK of the Processor Key
  - This endorses the processor with this key
  - Can be verified using the signature/certificate with (trusted) PubRK of the Manufacturer Root Key
- Processor Key (previous slide) is used to sign a certificate which binds the PubMK of the Monitor Attestation Key and the Monitor Meaurement which includes measuring the Signing Enclave
- The security monitor does not compute attestation signatures directly as it is not executed in an enclave, hence, not isolated (and vulnerable to leakage over the timing side channel)
- The signing enclave receives the monitor's private attestation key via an API call: When receiving the call, it compares the calling enclave's measurement with a hard-coded value and if matching, copies the attestation key into the enclave's memory.
- The signing enclave uses a mailbox to receive a report (using simplified SGX's local attestation), which includes
  - The application enclave's measurement
  - Data to be reported/communicated

# Security Monitor

- The monitor receives control after mroot finishes setting up the attestation mechanism.
- The monitor provides API calls to the OS and enclaves for DRAM region allocation and enclave management.
- After the system boots up, all DRAM regions are allocated to the OS, which can free up DRAM regions so it can re-assign them to enclaves or to itself.
- A DRAM region can only become free after it is blocked by its owner. While blocked, any address translation mapping to it causes page faults, so no new TLB entries will be created for that region.
- The monitor ensures that the OS performs TLB flushes before the OS frees blocked regions (in order to remove stale entries for that region).
- This only requires TLB flushes on the cores (LPs) running that enclave's threads.



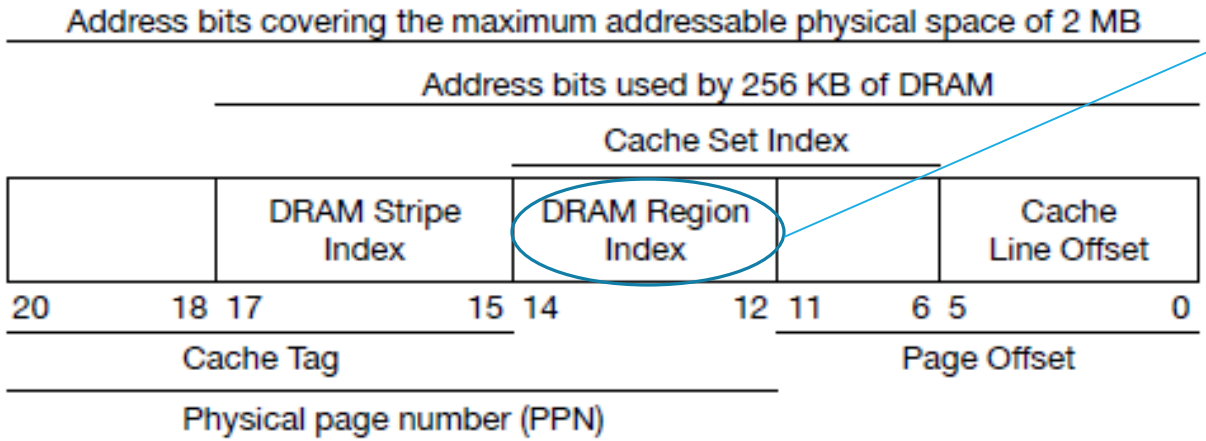
# Local Attestation

- Cannot follow SGX's approach because it relies on key derivation and MAC algorithms, and Sanctum promises to avoid timing side channel leakage implying that the security monitor is not allowed to perform cryptographic operations that use private keys.
- Each enclave has an array of mailboxes specified at its creation time
- An enclave that wishes to receive a message in a mailbox (e.g. signing enclave) declares its intent by performing an *accept message* monitor call. The API call is used to specify the mailbox that will receive the message and the identity of the enclave that is expected to send the message.
- The sending enclave (e.g. the one wishing to be authenticated) performs a *send message* call that specifies the identity of the receiving enclave and a mailbox within that enclave. The monitor delivers messages to mailboxes that expect them.
- Hen the receiving enclave is notified via an out-of-band mechanism that it has received a message, it issues a *read message* call to the monitor, which moves the message from the mailbox into enclave's memory.

# Outline

- Adversarial Models AEGIS, Intel SGX, and Sanctum revisited
- Sanctum: HW Isolation – For remote attacks, no need for encryption and memory integrity checking!
- Sanctum: Replaceable security software (security monitor) – Gives control!
- Sanctum: Protection against cache side channel attacks – and still practical!

# Cache Coloring




Defined as the intersection between cache index and PPN

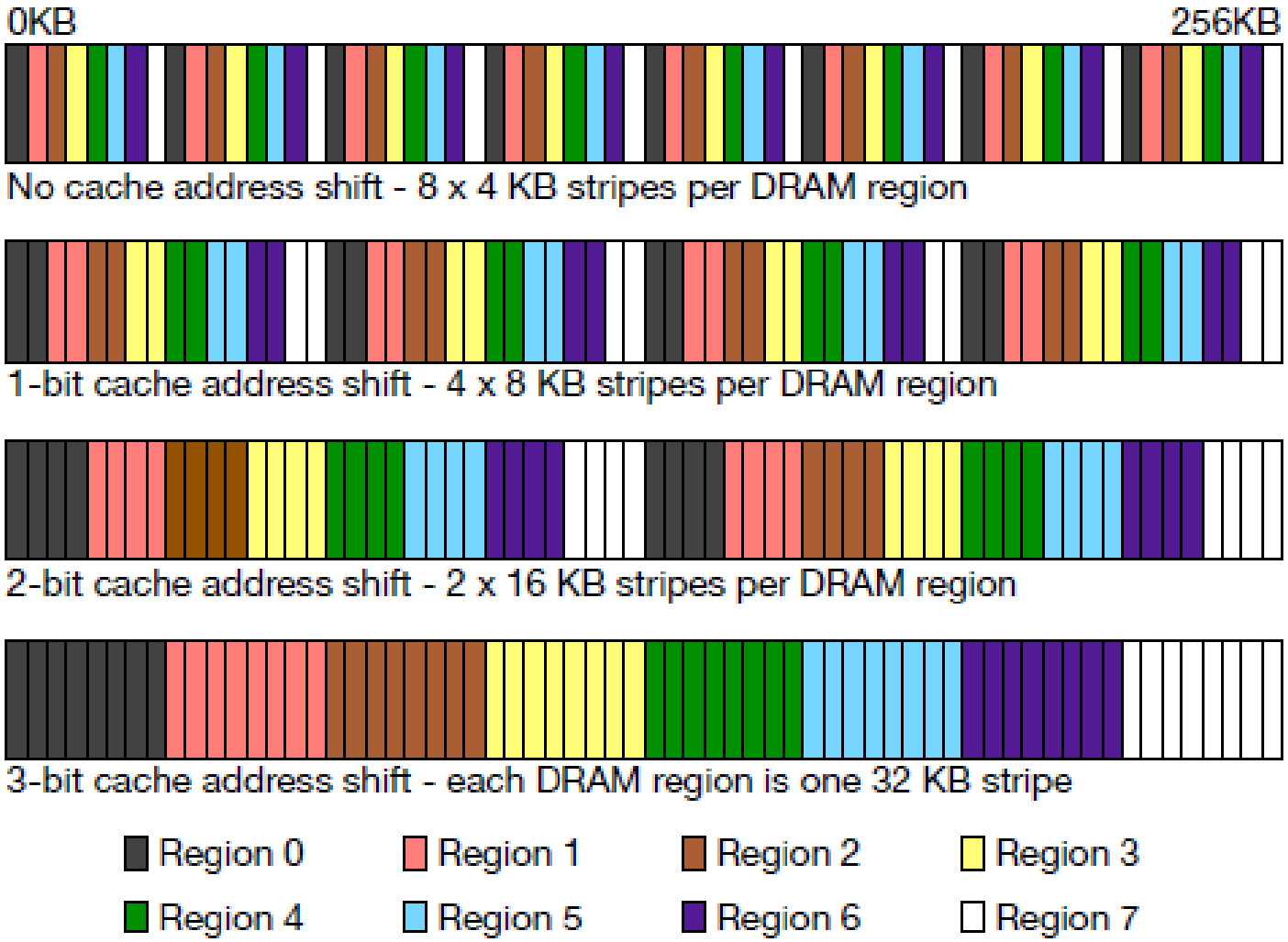
- The maximal set of bits that impact cache placement *and* are determined by privileged SW via page tables
- Bits [14..12] define the subset of DRAM with addresses having the same DRAM region index

- Toy example: 32-bit virtual addresses; 21-bit physical addresses; 4KB pages, a set-associative LLC with 512 sets and 64B cache lines, and 256KB of DRAM
- Location where a byte of data is cached in LLC depends on the low-order bits in the byte's *physical address*:
  - Set index determines which of the LLC lines can cache the line containing the byte
  - The line offset locates the byte in the cache line
- Virtual address's low order bit make up
  - The page offset
  - Virtual Page Number (VPN)
- Address translation leaves the page offset unchanged, and translates the VPN into a Physical Page Number (PPN)

# Cache Coloring

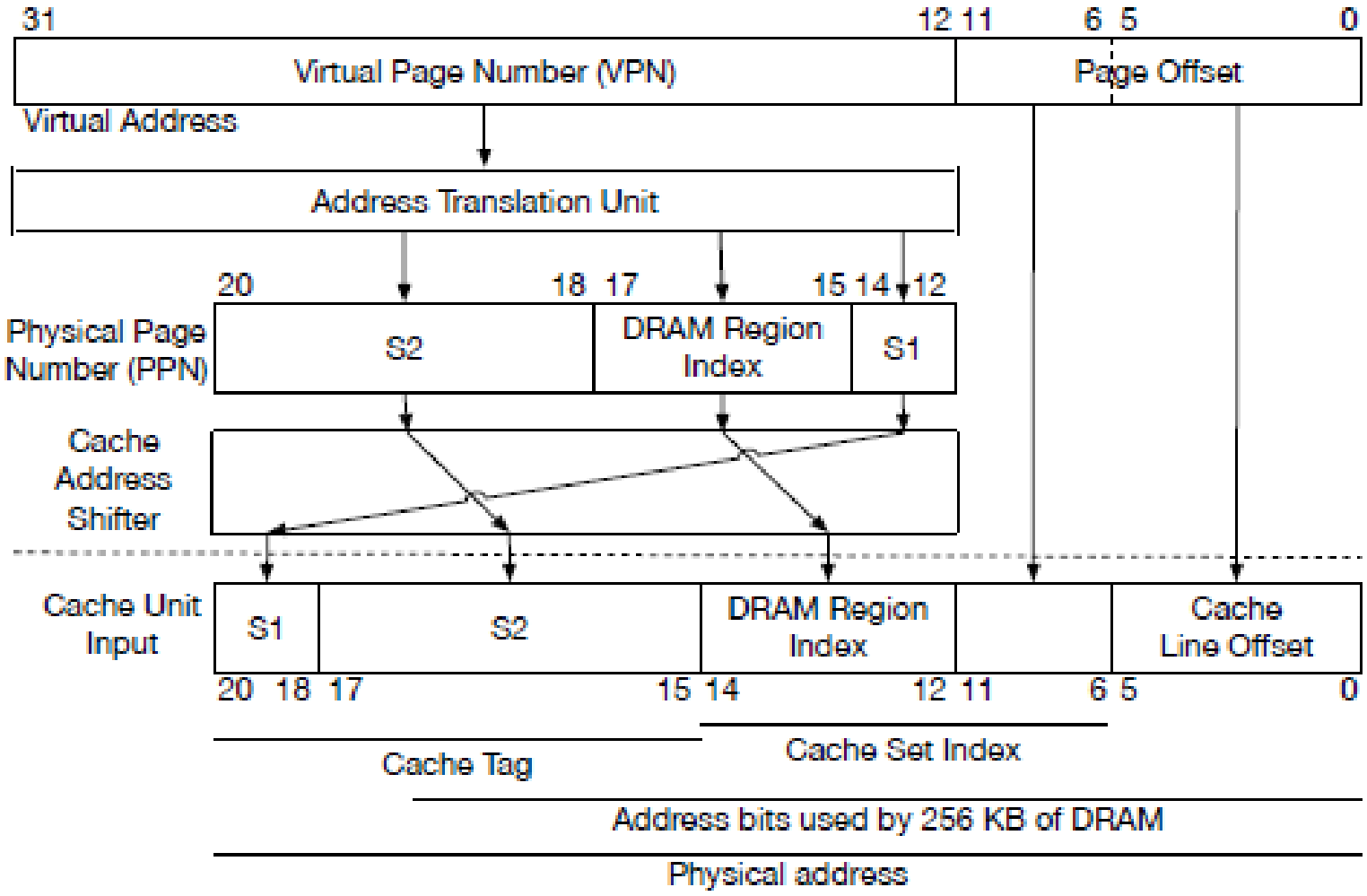
- Addresses in a DRAM region do not collide in the LLC with addresses from any other DRAM region!!
- If Alice and Eve use disjoint DRAM regions, then they do not interfere in the LLC and a cache covert channel attack is impossible.
- Without Sanctum's HW extension: DRAM regions are made up of multiple continuous DRAM stripes where each stripe is exactly 1 page long.

# Cache Coloring

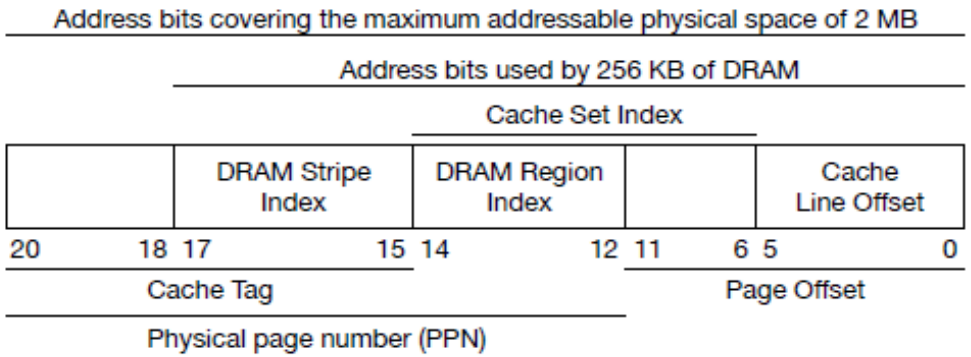



- Fragmentation of DRAM regions makes it difficult for the OS to allocate contiguous DRAM buffers (which are essential to the efficient DMA transfers used by high performance devices)
- If the OS only owns 4 DRAM regions, the largest contiguous DRAM buffer it can allocate is 16KB
- Circularly shifting the PPN to the right by 1 bit, before it enters the LLC, doubles the size of each DRAM stripe and halves the number of stripes in a DRAM region

# Cache Coloring

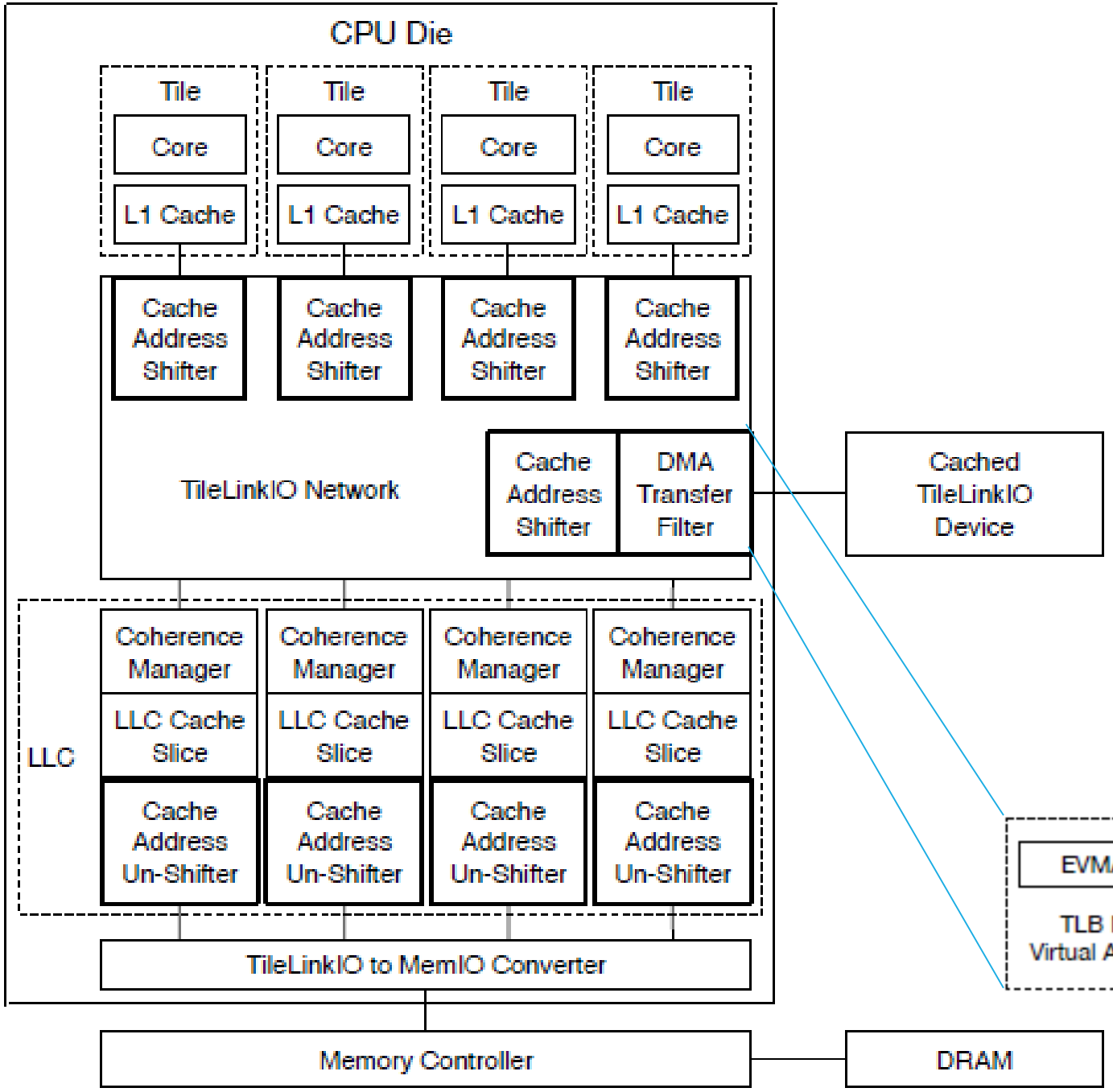


# Cache Address Shifter

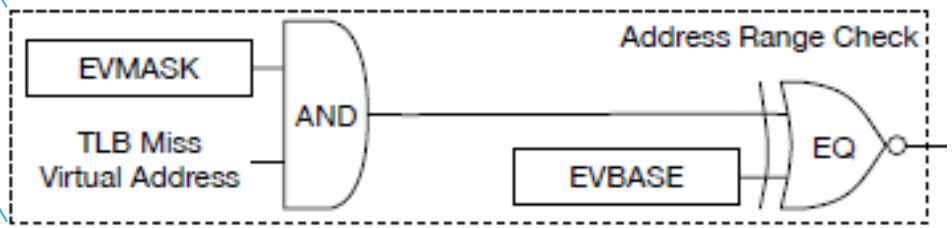



- Example: Which PPN addresses are in DRAM region with index 011?
  - These are addresses ******011**********, i.e., address bits with positions 17, 16, 15 can be arbitrarily chosen and the page offset can be arbitrarily chosen
  - This means that the DRAM region has all the pages which start with a PPN of the form ******011000000000000
  - Since we only have freedom in the higher order address bits, these pages are separated in DRAM (since 011 is three bits, every 2^3=8$^{th}$ page is in the DRAM region)
- After cache address shifting over 3 positions the intersection of the Cache Set Index and the **shifted PPN** are bits 17, 16, 15
  - the shifted PPN has the PPN bit positions in the following order: 14, 13, 12, 20, 19, 18, 17, 16, 15 and the lower 3 of these are in the cache set index
  - Now the pages in DRAM region 011 start with a PPN of the form ***011***00000000000
  - This means that they contain 8=2^3 consecutive pages! (Because there are 3 degrees of freedom in the lower order PPN bits right after the page offset.)
  - It does not help to shift more: The 3 highest order PPN bits are not used since the DRAM is only 256B (i.e., only PPN bits 12 to 17 are used)

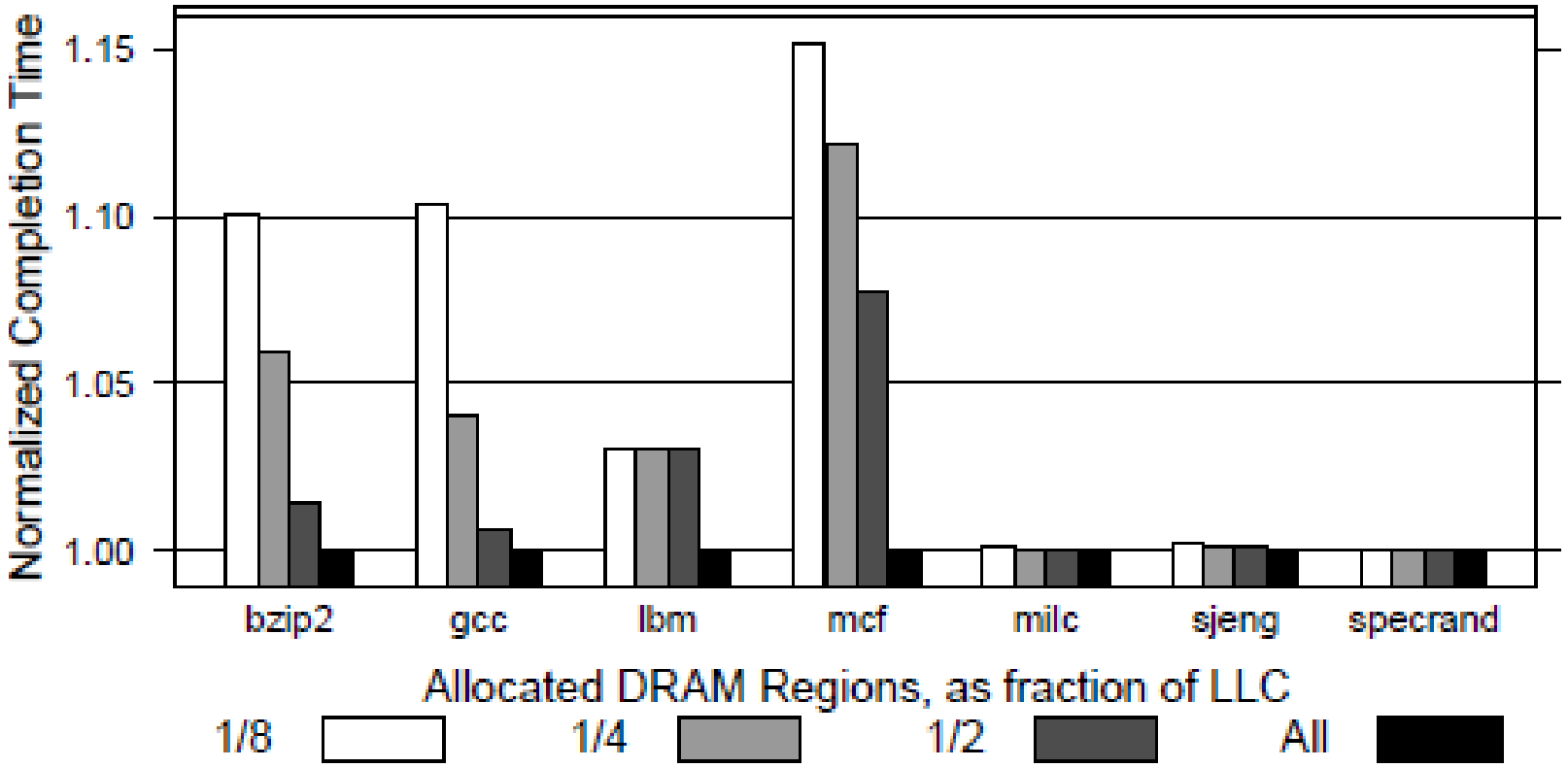
# Transformation Logic




- DMA bus master rejects DMA transfers pointing into DRAM regions allocated to enclaves (similar to modifications done by SGX and TXT revisions to the integrated Memory Controller).
- Uses a whitelist approach instead of following SGX's blacklist approach

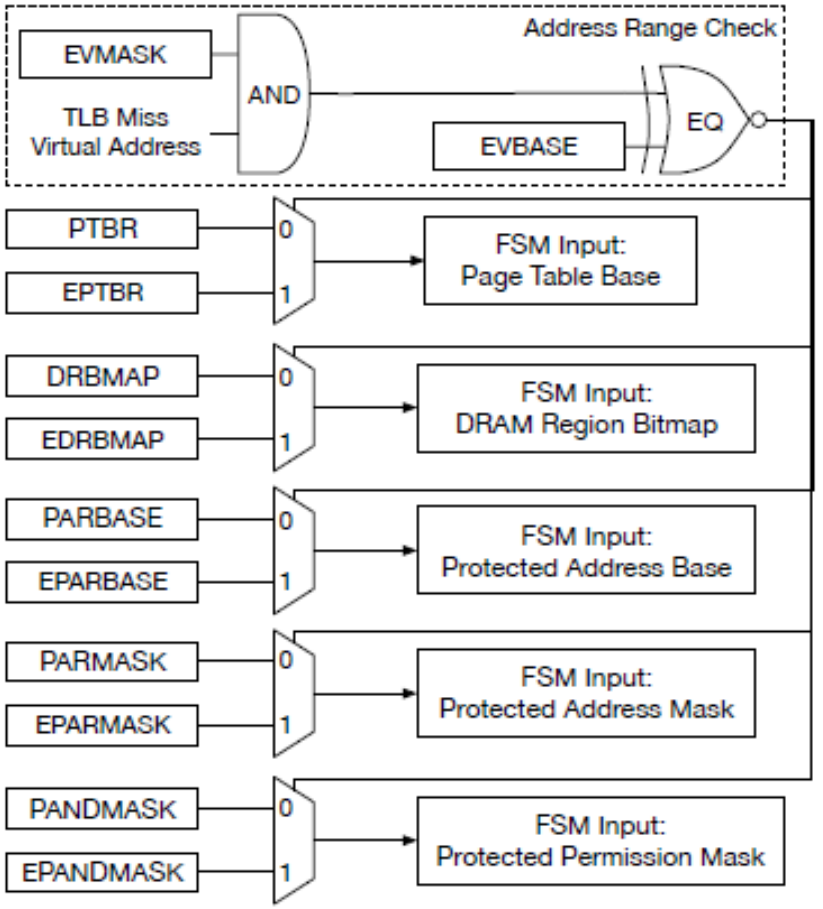


# Performance



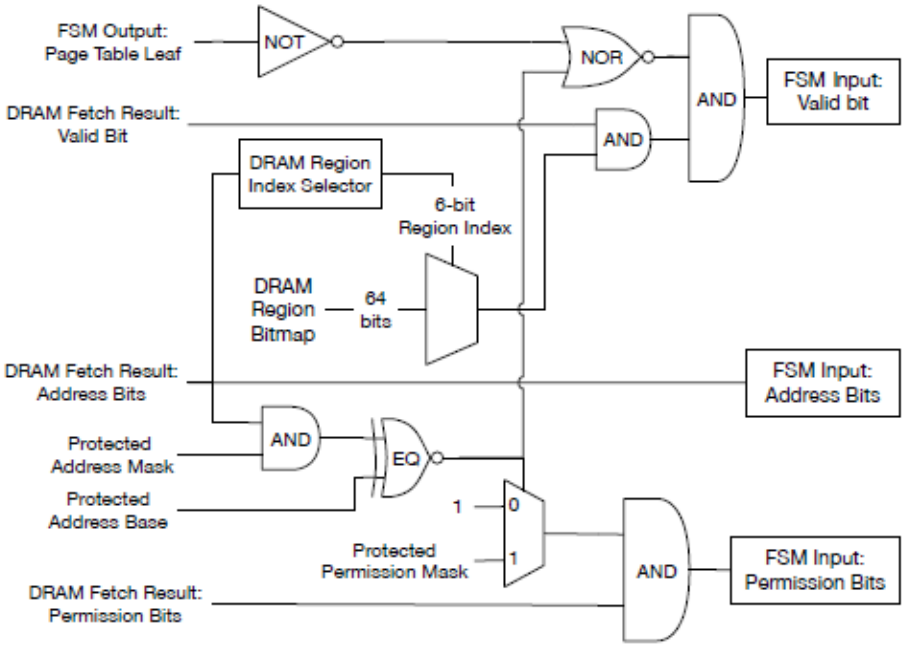
# Page Walker Input

- Sanctum's enclave page tables require an enclave page table register eptbr storing the physical address of the currently running enclave's page tables (each enclave stores its own page tables)
  - ptbr points to OS managed page tables
  - The per-enclave eptbr can only be accessed by the security monitor
- TLB misses:
  - Switches between ptbr and eptbr based on two registers that indicate the current enclave's EVRANGE: evbase (enclave virtual address space base) and evmask (enclave virtual address space mask)
  - Select appropriate page table base by ANDing the faulting virtual address with the mask register and comparing output against the base register
  - Depending on result either eptbr or ptbr is forwarded to the page walker as the page table base address



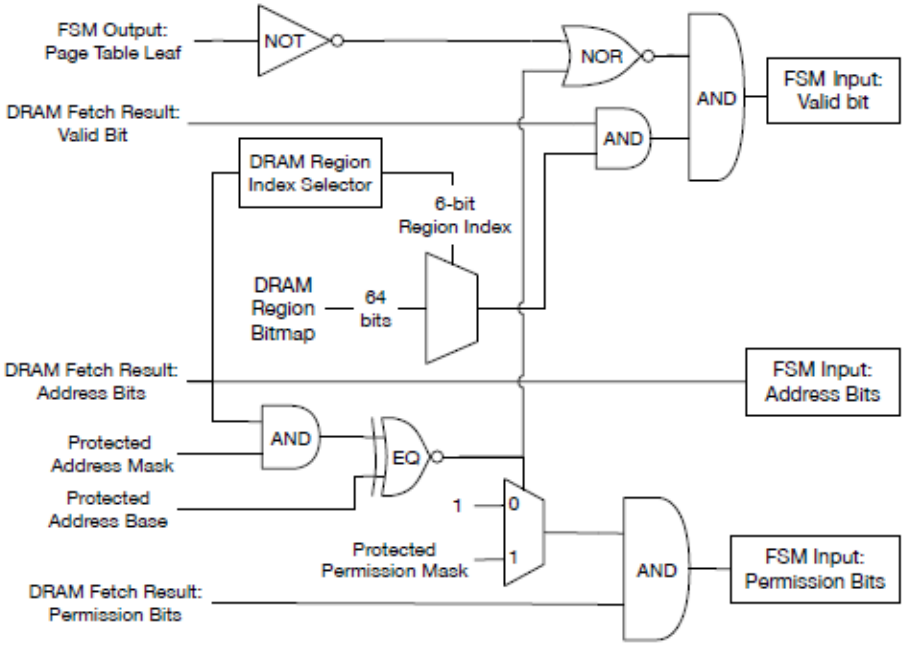
# Page Walker Memory Access

- Address translation is performed by a HW page walker that traverses page tables when a TLB miss occurs.
  - Page walker's latency greatly impacts the CPU's performance
  - It is implemented as a FSM that reads page table entries by issuing DRAM read requests using physical addresses over a dedicated bus to L1 cache
  - Modifications require a lot of engineering effort; yet, Sanctum demands that the page walker only references enclave memory when traversing the enclave page tables and only references OS memory when translating the OS page tables
  - Solution: No FSM modification; Security monitor works in concert with the circuit on the right
  - The circuit receives each page table entry fetched by the FSM, and sanitizes it before it reaches the page walker FSM
  - Security monitor configures the set of DRAM regions that page tables may reference by writing to a DRAM region bitmap (drbmap) register



# Page Walker Memory Access

- Circuit extracts the DRAM region index from the address in the page table entry and looks it up in the DRAM region bitmap
- If it does not belong to allowable DRAM region, then the page table entry's valid bit is forced to 0 (causing the page walker FSM to abort the address translation and signal a page fault)
- Security monitor maintains metadata about each enclave in the enclave's DRAM regions. Metadata must not be writable by the enclave.
- Sanctum extends the page table entry transformation to implement per-enclave read-only areas. The resulting protected physical address range is indicated by parbase and parmask registers.
- Circuit checks if each page table entry points into the protected range.
- If a leaf page table entry is seen with a protected address, its permission bits are masked with a protected permissions mask parpmask register. If a protected address is discovered in an intermediate page table entry, its valid bit is cleared to 0 – prevents the page walker FSM from modifying the protected region by setting accessed and dirty bits.



# Transformation Logic