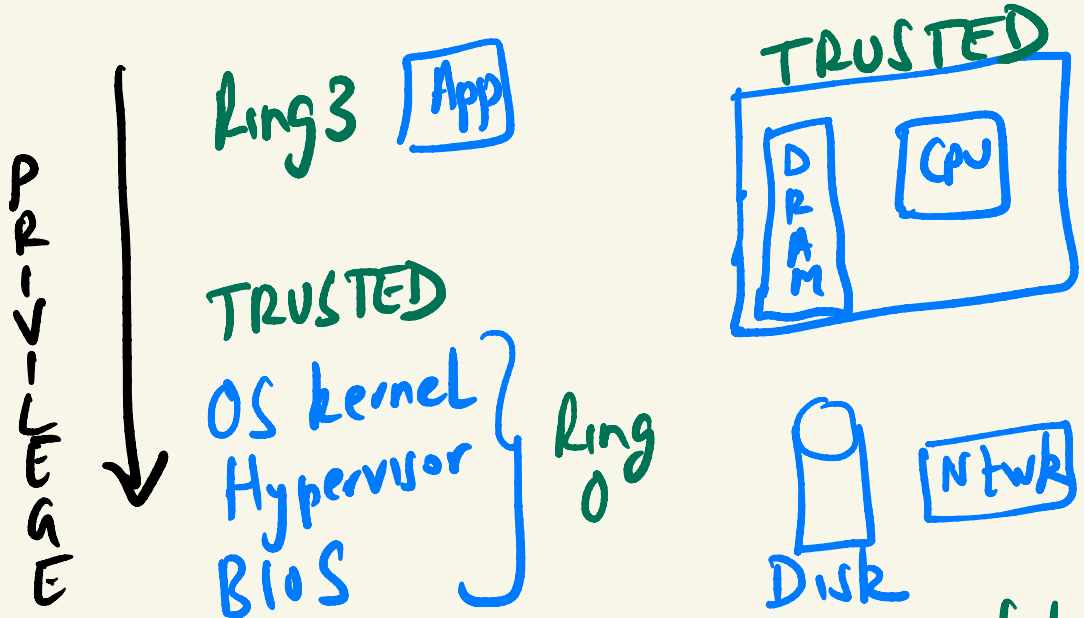
Secure Processors

Typical Computer Trusted Computing Base (TCB)

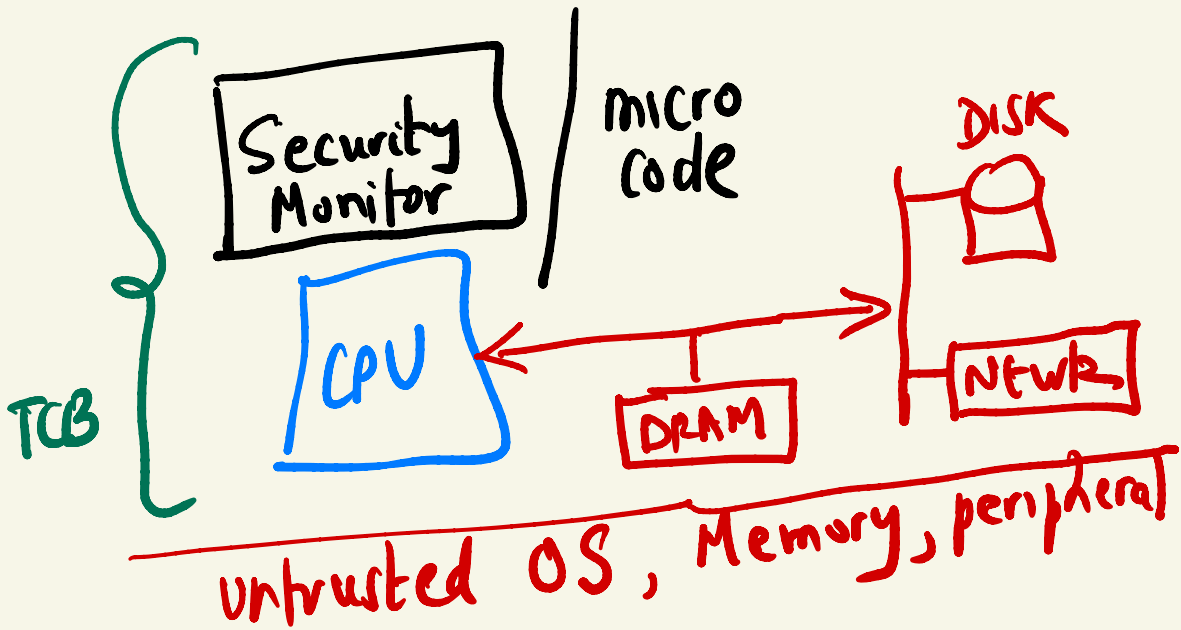
Software

Hardware



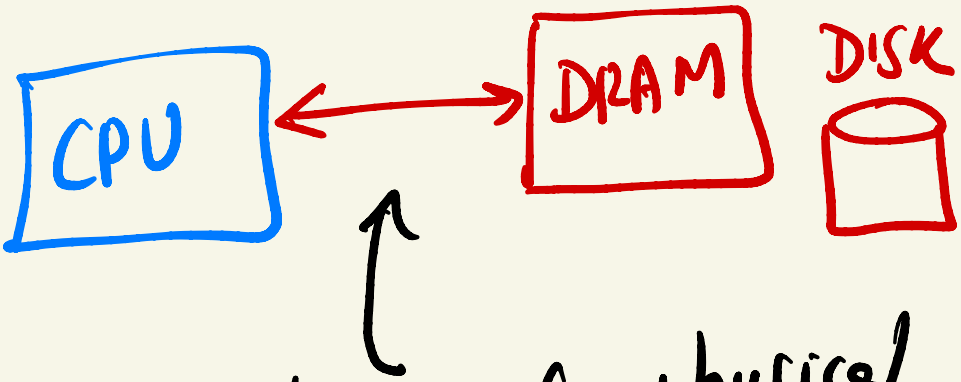
100's of millions of LOC, billions of logic gates!

Single-Chip Secure Processor



- Shrink software TCB to thousands of LOC.
- Still trust the hardware processor.

Untrusted Memory



attacker: OS, physical access

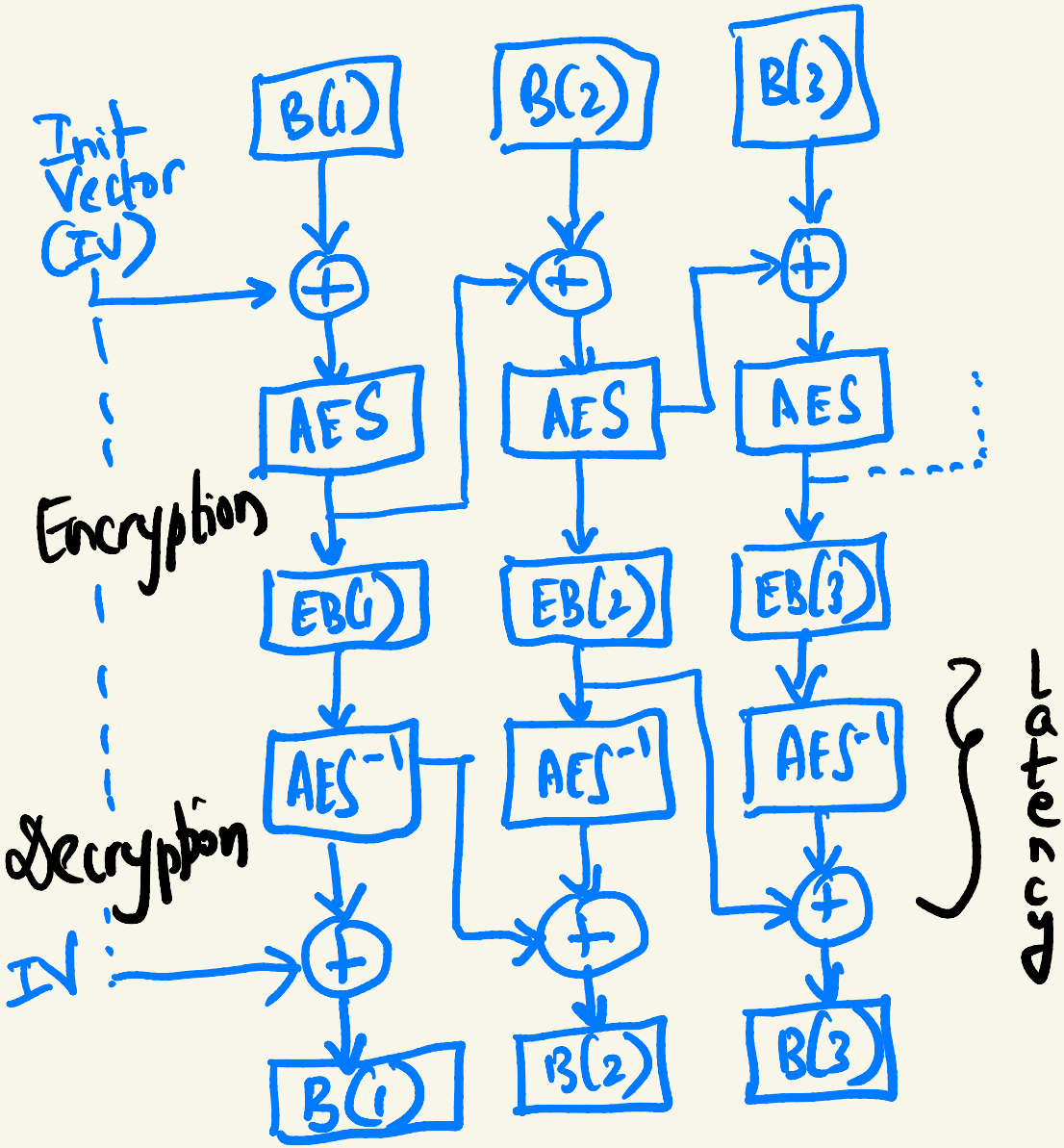
Confidentiality: Encryption

Memory latency increase hurts performance (most programs are memory bottlenecked!)

Direct - Block Encryption

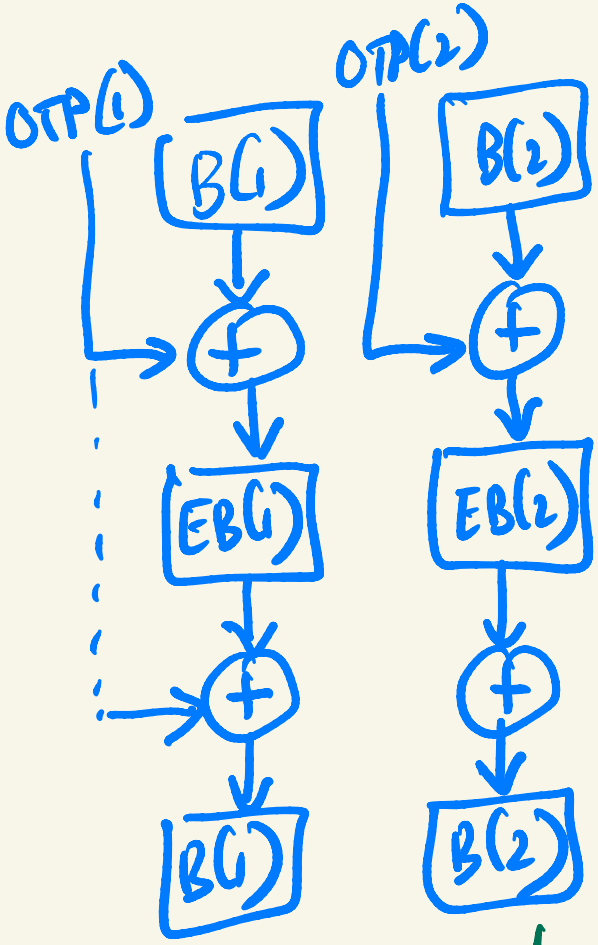
Cipher Block Chaining (CBC) mode

Cache Block



One-Time-Pad Encryption

One-time-pad (OTP)



encryption

decryption

XOR'ing is fast!

How to store OTP[i]'s? — can't store in untrusted DRAM!

Counter-Mode Encryption

Generate OTPs from timestamps with AES! Store timestamps with encrypted blocks in memory.

L2-Cache-Writeback:

1. TS = TS + 1
2. a. OTP = AES_K(Addr, TS)
   b. EB = B ⊕ OTP
3. Write TS, EB to memory

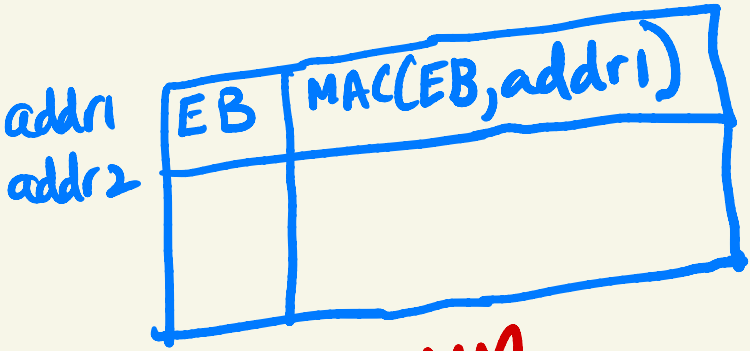
L2-Cache-Miss:

1. Read TS from memory.
2. in parallel { OTP = AES_K(Addr, TS); Read EB from Addr in Memory }
3. B = EB ⊕ OTP

Cache TS's on-chip, Speculate

Active Attacker / Integrity

Just use a MAC:



Untrusted DRAM

Problem: Replay attacks

Suppose program writes:

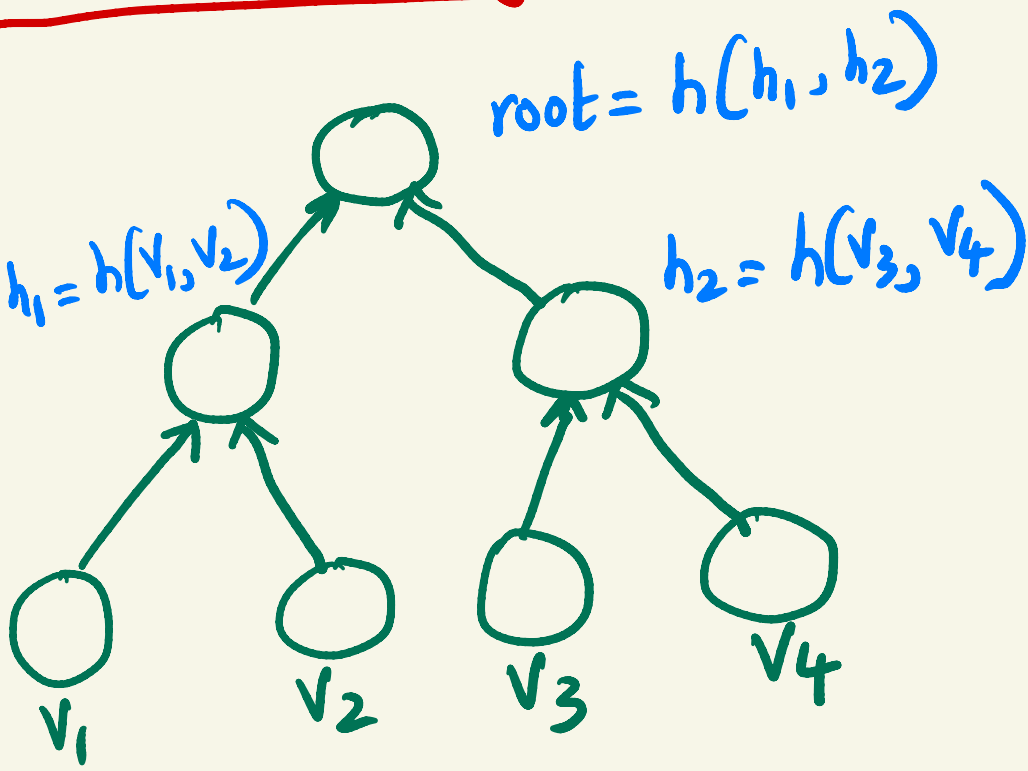
EB_1, $MAC(EB_1, addr_1)$

EB_2, $MAC(EB_2, addr_1)$

adversary ignores this write.

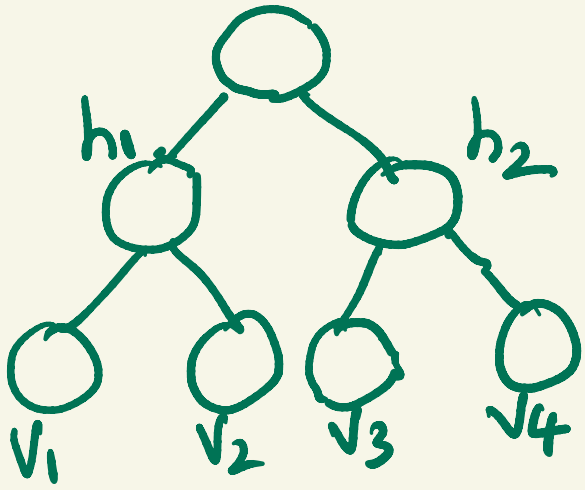
return this value on next read ⇒ no failure/detection.

Merkle/Integrity Tree



Store root in processor secure memory ⇒ cannot be tampered with. root updated checked on reads, on writes.

Tree Operations: Read



Read V_3

Read V_4

Read h_2

Compute $h_2' = h(V_3, V_4)$

Check $h_2' = h_2$

Read h_1

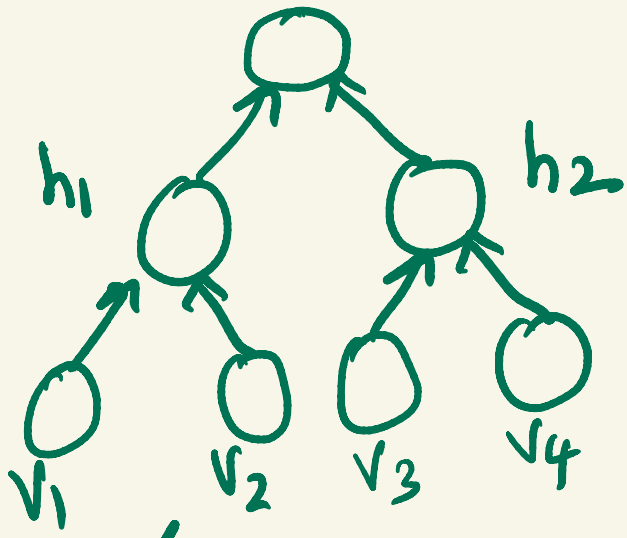
Compute $root' = h(h_1, h_2')$

Check $root' =$ processor root

Collision-resistant $h()$ provides security

Adversary can't modify V_3, V_4, h_2, h_1 to match root.

Tree Operations : Write



Write v_2'

read v_1

Compute $h(v_1, v_2') = h_1'$

write h_1'

read h_2

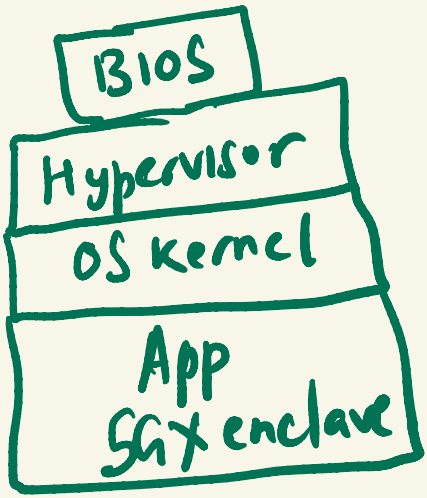
Compute $h(h_1', h_2) = root'$

Update processor root with $root'$

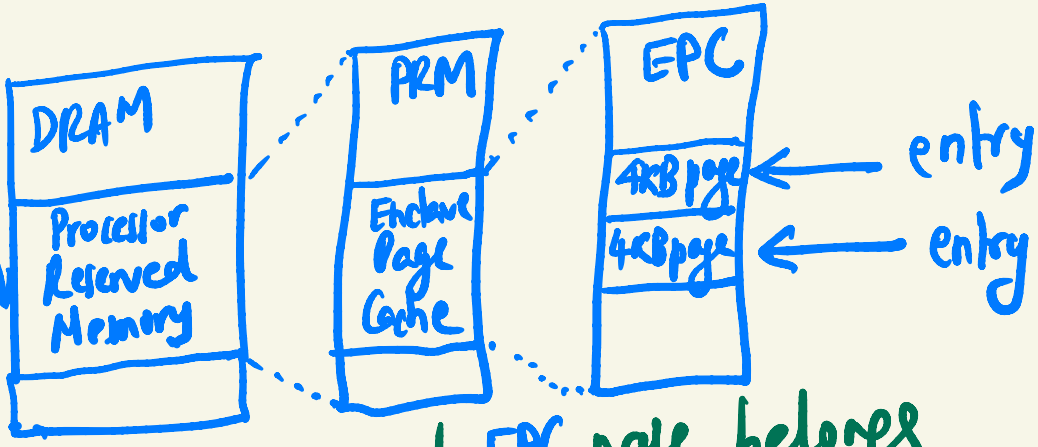
Caching intermediate hashes in trusted processor cache improves performance significantly.

Intel SGX

- Memory encryption & integrity verif.



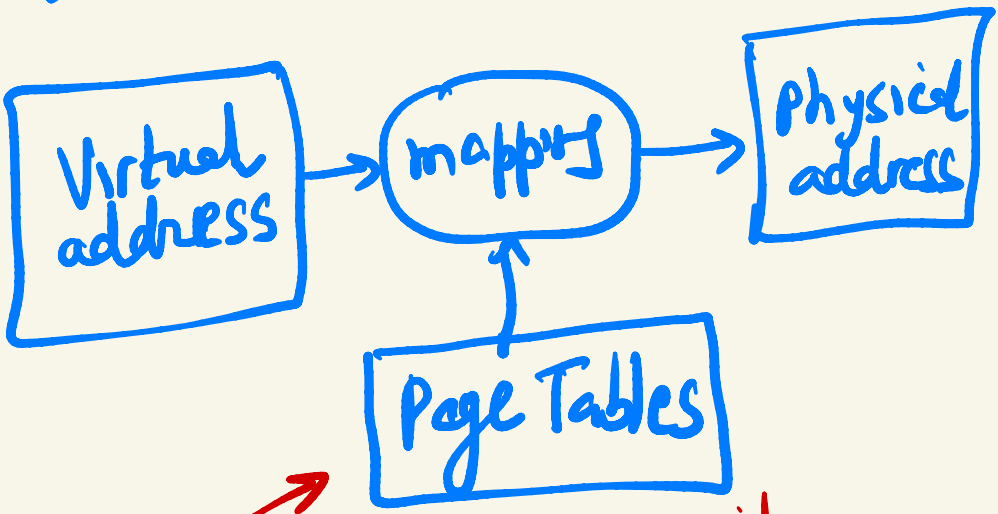
← TRUSTED
+ MICROCODE ROM



CPU ensures each EPC page belongs to exactly one enclave.

SGX Leaks

- Untrusted OS/app can attack enclave via (shared) cache timing attacks.
- Address Translation Leak



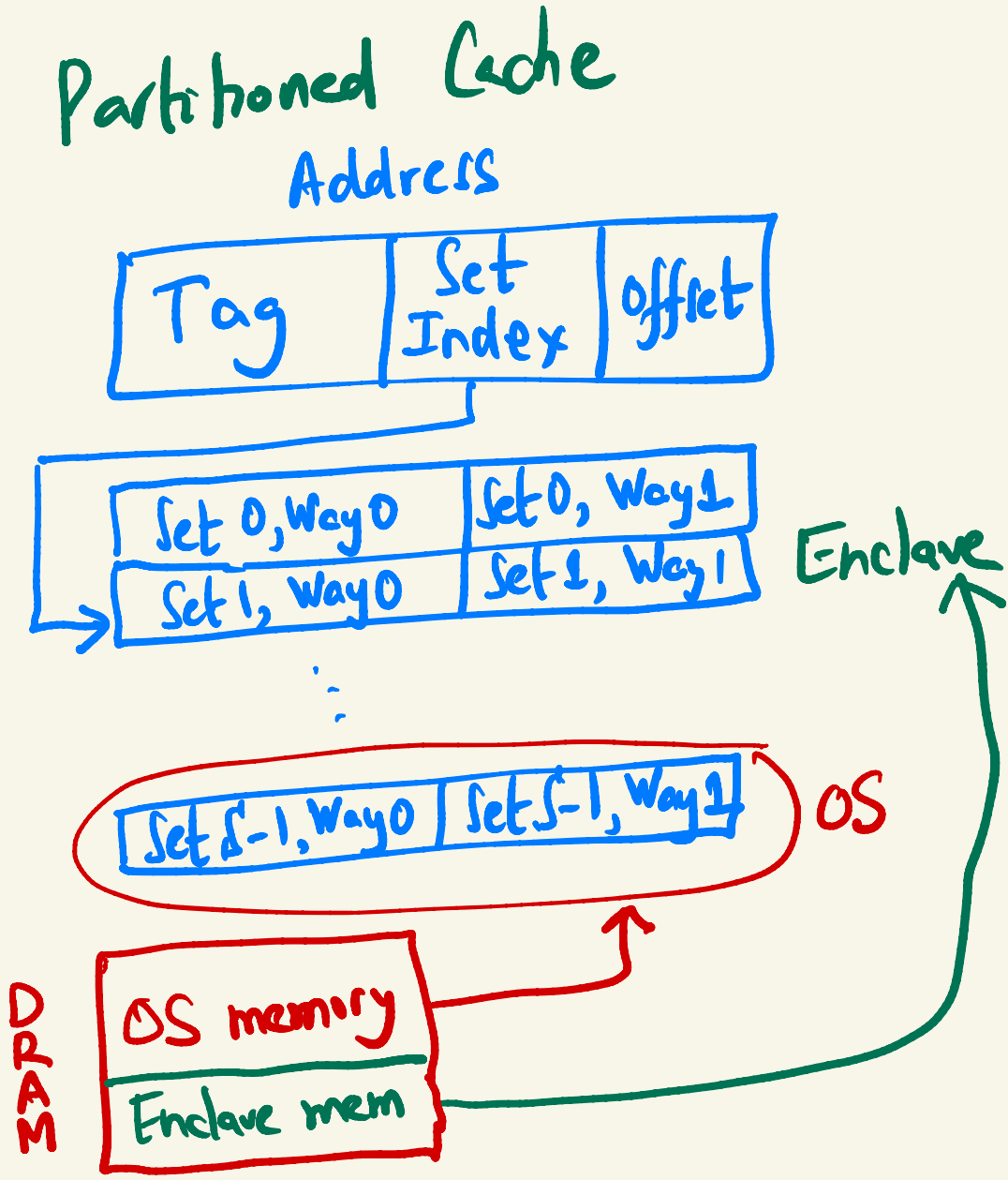
Untrusted OS manages its and enclaves' page tables!

Controlled-Channel Attack Paper

1. Malicious Hypervisor sets P (present) flag to 0 on all page table entries, lets enclave execute.
2. First memory access causes a page fault. Hypervisor maps faulting page & resumes enclave execution. ← sees page address
3. Next page fault, hypervisor maps in new page, and unmaps previous page, so it can see enclave's memory access pattern at page granularity (minus offset)
4. Instruction execution ↔ pages mapping
   ↳ using offline analysis.
   ↳ gives control/secret data.

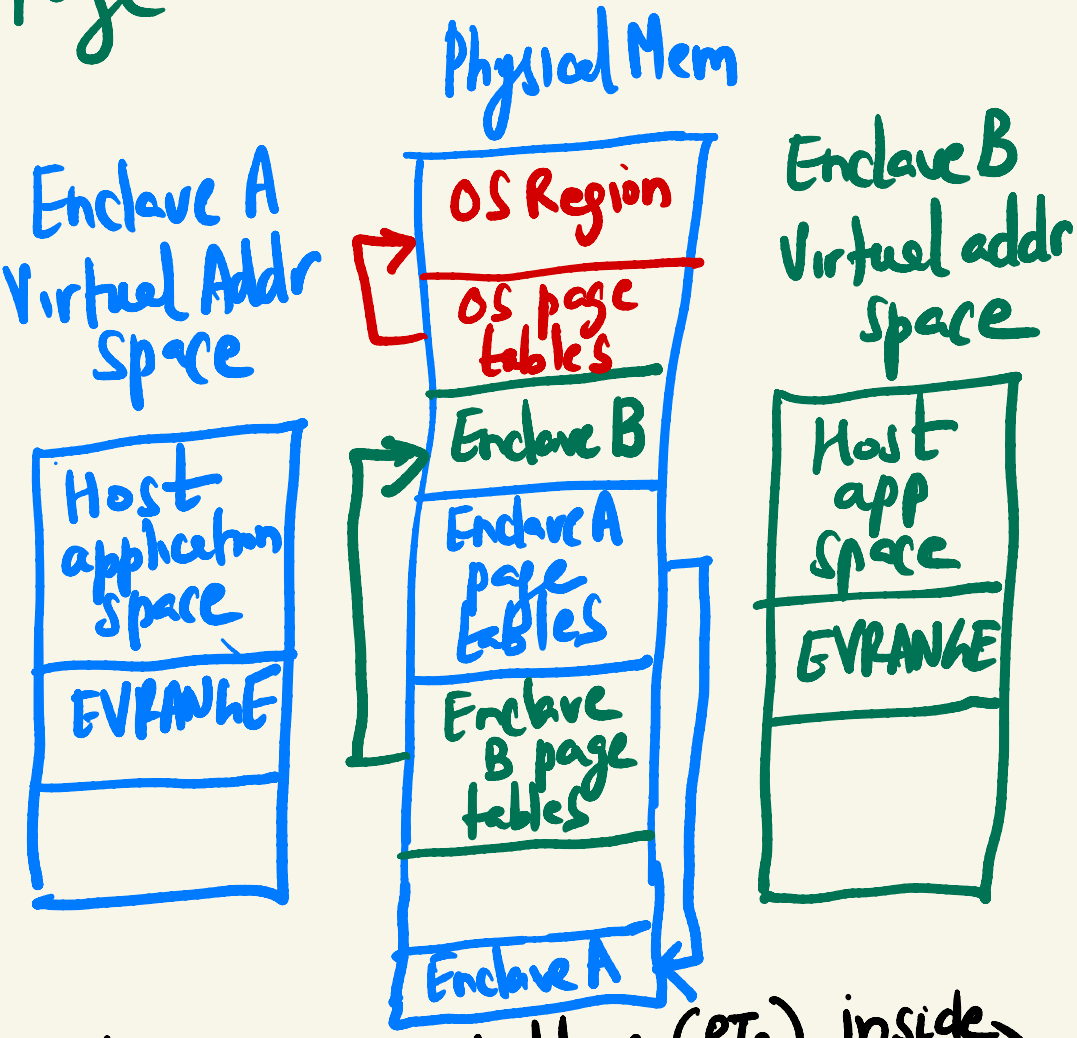
Sanctum Design

Partitioned Cache



Sanctum Design

Page table isolation



Enclave page tables (PTs) inside enclave memory (isolated from OS)
- need some hardware to multiplex PTs

Enclave Life cycle (simplified)