



Lecture 19:

Trusted Execution Environments

CPSC 4440/5440, 2026 Spring

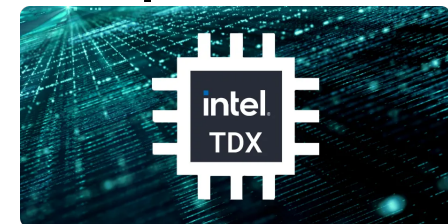
Instructor: Fan Zhang

Yale



Hardware-based Security

- Today: What, why, and key concepts
- Case study of **Intel SGX/TDX**
 - Available in all current Intel server-grade CPUs
- **Security risks** of TEEs
 - Side channels attacks
- **Applications** of TEEs to everything we have learnt so far.



Phases in deploying crypto

1. **Cryptography design**: design secure and efficient protocols
 - e.g., yields RFCs like the one for TLS
2. **Implementation**: translate crypto protocols to efficient and secure code
 - e.g., yields hardened library such as OpenSSL
3. **Deploy & Execution**: users & services obtain code & run in some execution environment.

Is execution
environment secure?

Security concerns if running code in untrusted env

- ***Isolation*** from other software. E.g.,
 - We run many applications on our laptop/phones. We may worry that **malicious software** can access sensitive data (memory, disk, network).

Security concerns if running code in untrusted env

- ***Isolation*** from ***privileged*** software (i.e., root privileges).
 - Service may need to secure *secrets* stored on client devices
 - e.g., Cellular networks, device identity, DRM systems
 - Client may gain root privileges, or malicious code on client device may gain root privileges via vulnerabilities.
 - so software can't be trusted.

Security concerns if running code in untrusted env

- ***Modifications*** of code. E.g.,
 - eBook platforms install custom eBook reader to enforce contract: prevent copying/printing, limiting borrow time, etc.
 - What if malicious user **modifies** the code to remove protection?
 - (Same for movie streaming)

Security concerns if running code in untrusted env

- ***Modifications*** of code by **service providers**. E.g.,
 - User pays for \$100/month to access smartest LLM models.
 - A malicious service providers may use cheap model.
 - e.g., they can save significantly by aggressive quantization, at the cost of accuracy.

The ideal: trusted execution environment

Integrity



Other processes
and even OS
cannot tamper with
control flow of X

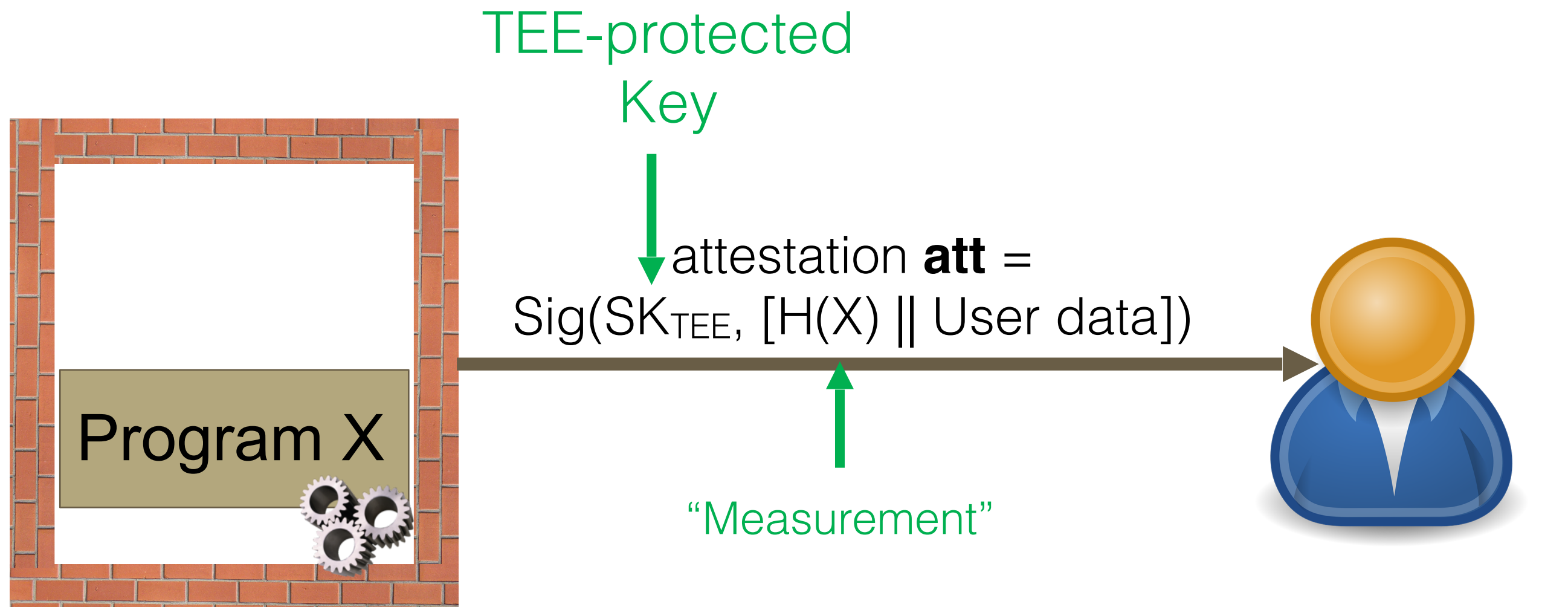


Confidentiality



Other processes
and even OS can
learn nothing about
the state of X

The ideal: trusted execution environment

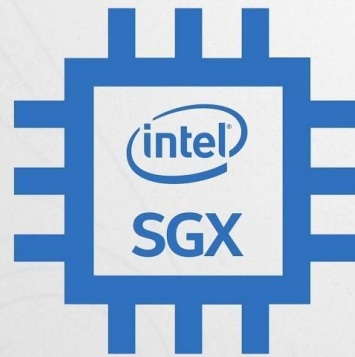
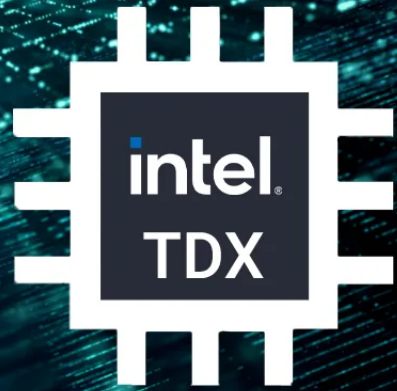


TEE running on a remote system

User is ensured that he is interacting with code X running on a “genuine” TEE.

Summary of ideal TEEs:
**integrity, confidentiality, and
verifiability (remote attestation)**

TEEs of varying capacities



AWS Nitro System

A combination of dedicated hardware and lightweight hypervisor enabling faster innovation and enhanced security

Get started with a Nitro-based Instance today



Apple has something similar

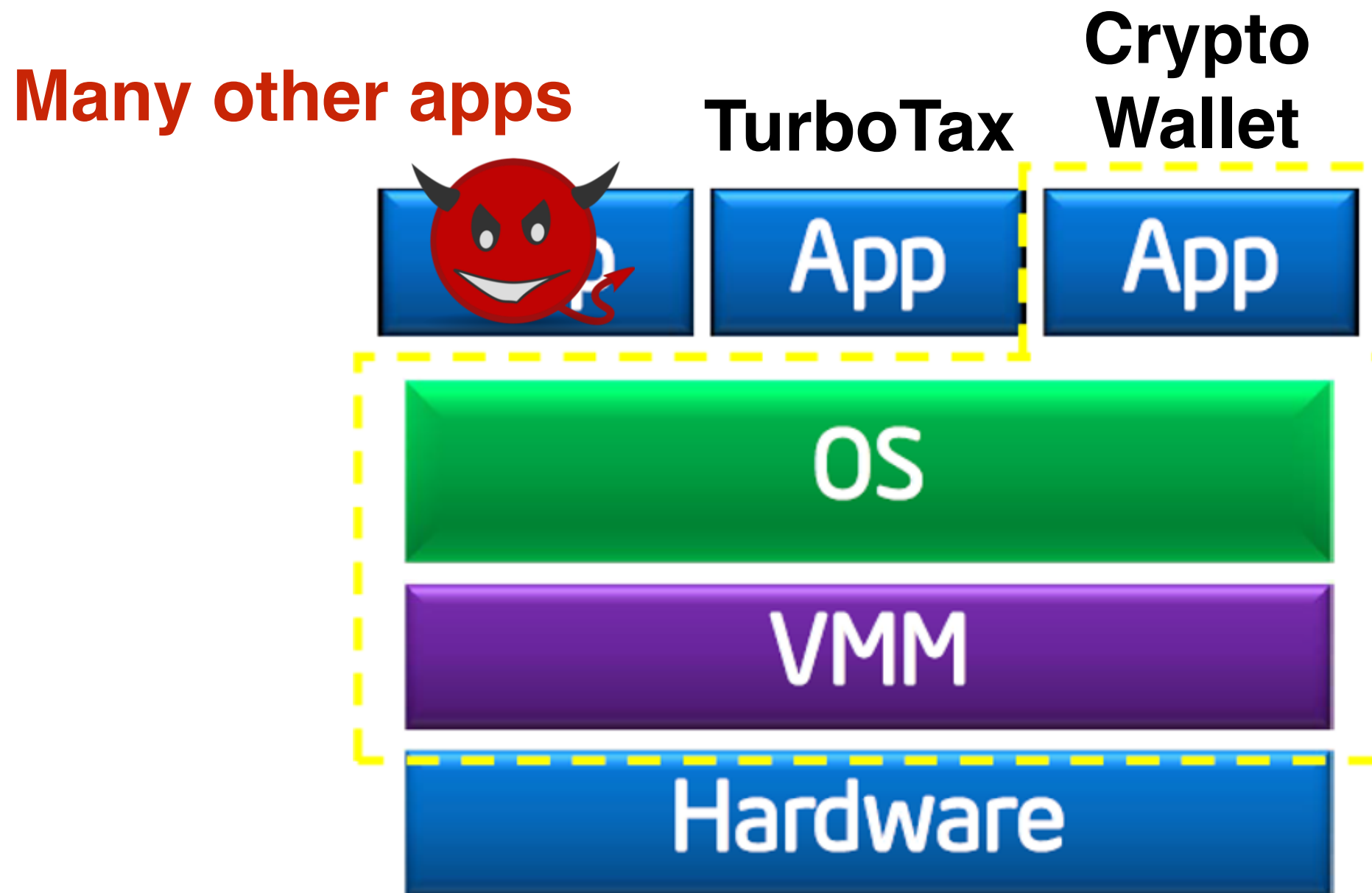
Also available in Google Cloud, Azure, etc.



- Also known as **Confidential Computing** in the industry

Why hardware?

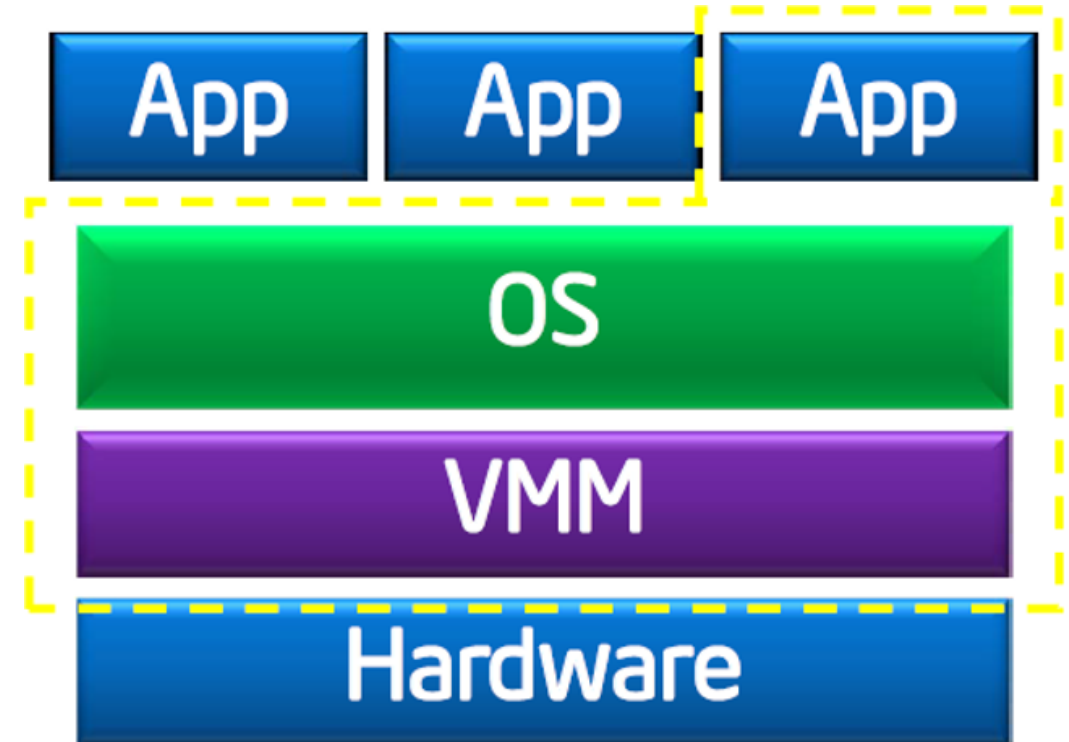
OS-based isolation



Good news: modern Operating Systems (OS) and CPUs provide **strong isolation** between apps (processes).

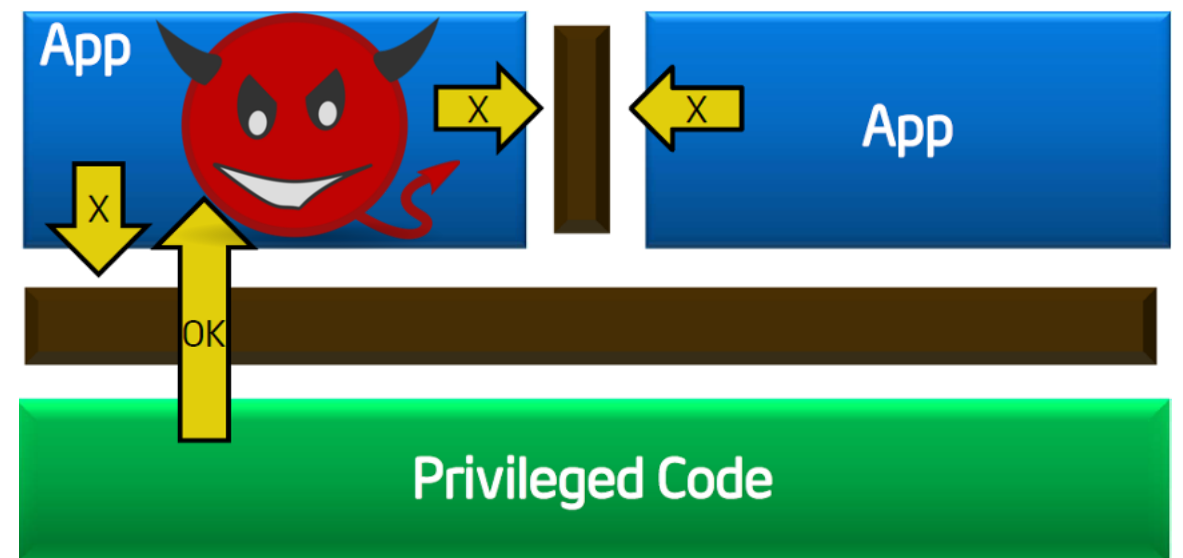
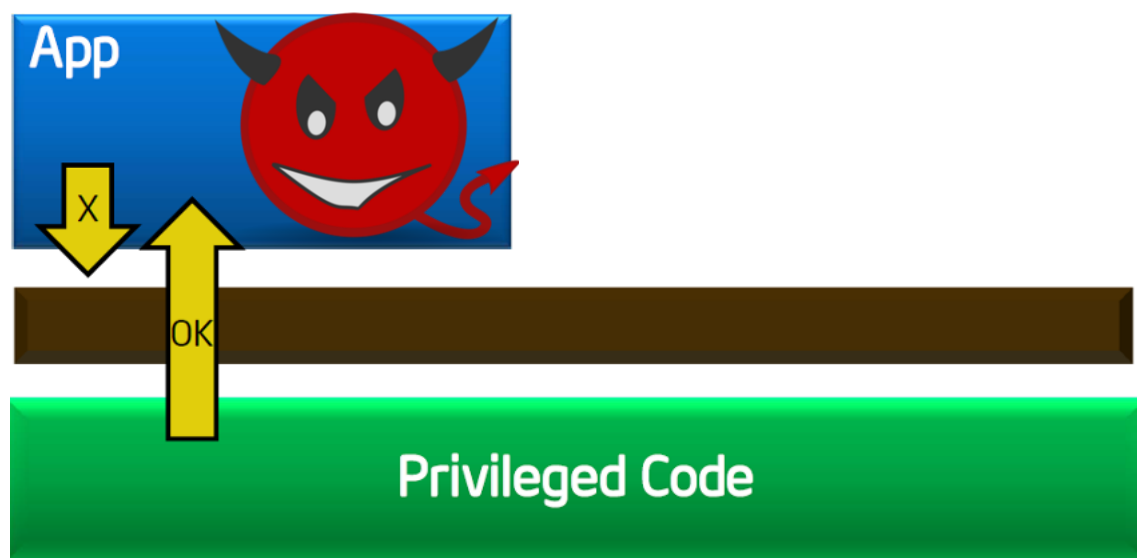
OS-based isolation

- **Virtual memory**
- Note: memory isolation is not a software-only mechanism
 - Can you name a key security policy enforced by hardware (e.g. CPU)?



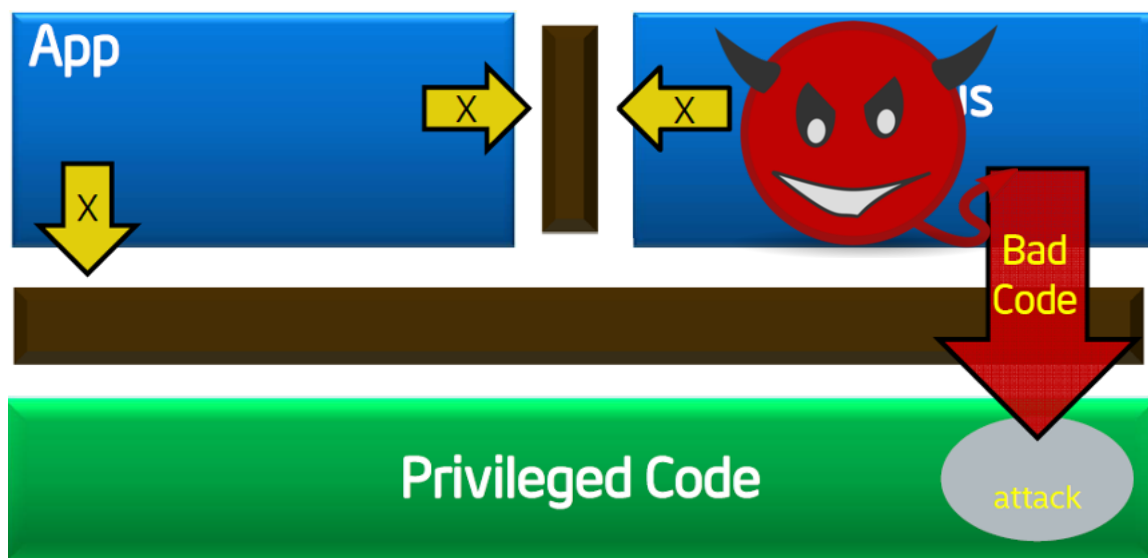
Limitations of OS-based isolation

- OSs can **protect itself from apps** and **protect apps from each other**.



Limitations of OS-based isolation

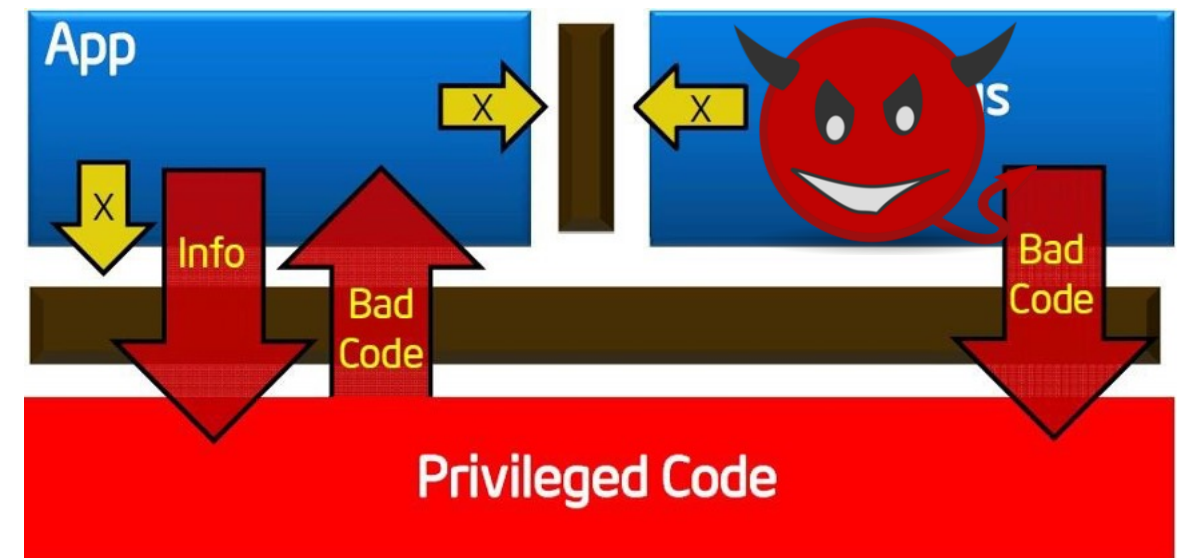
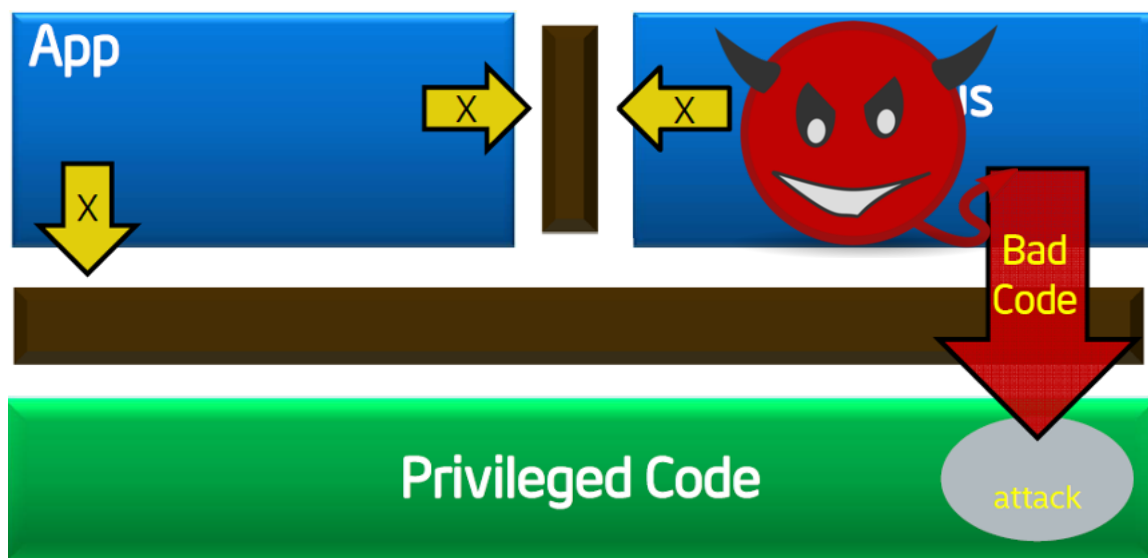
- ...*until* a malicious app exploits a flaw to gain full privileges



- OS code bases are so huge that they ***almost certainly will contain bugs***
 - Linux kernel: 27.8 million lines
 - Microsoft Windows: estimated 45–50 million lines
 - ~25k–30k vulnerabilities per year (according CVE)

Limitations of OS-based isolation

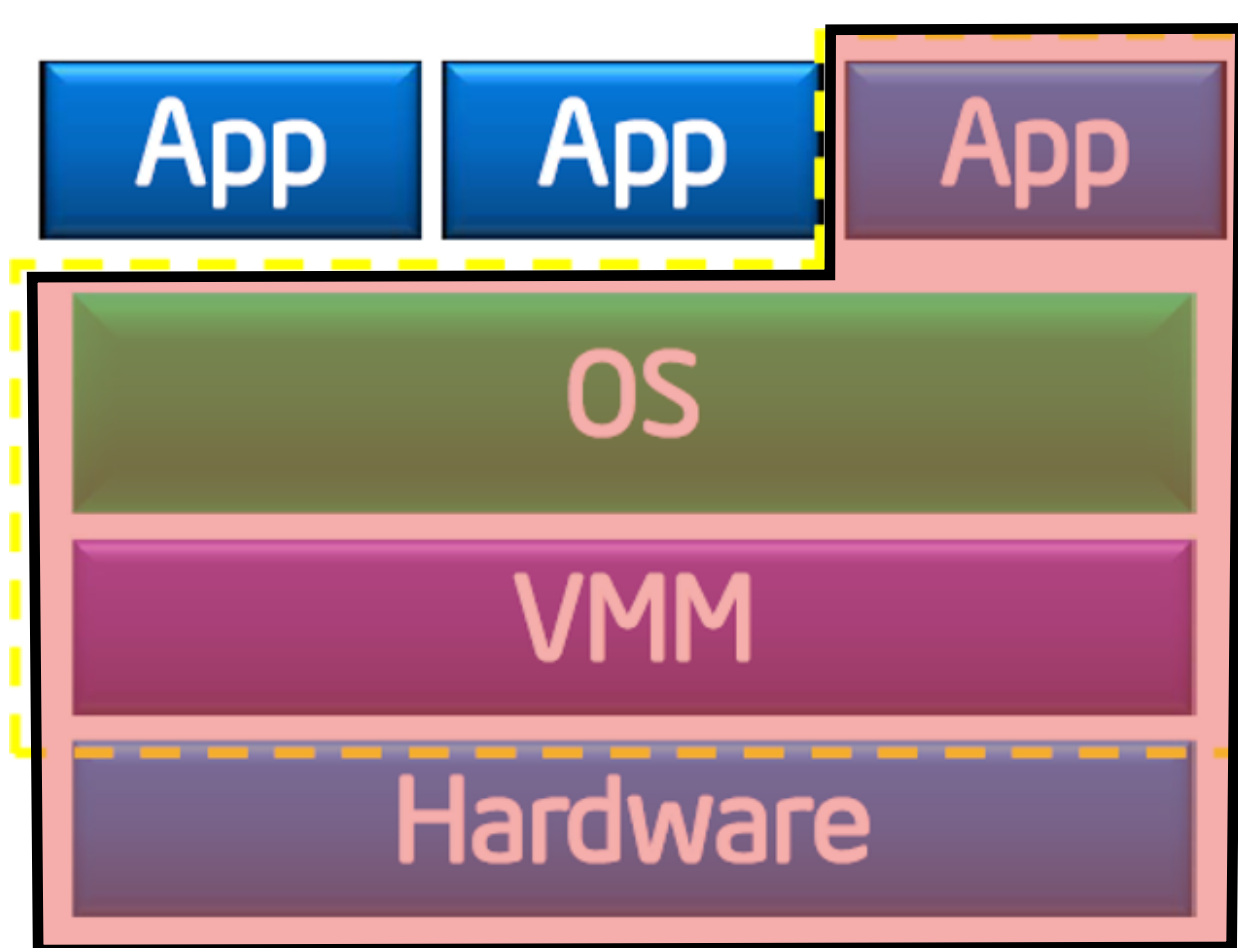
- Apps are typically not protected from privileged code attacks.
- Therefore: **security of app relies on security of OS.**
 - undesirable: $\text{LoC}(\text{app}) \ll \text{LoC}(\text{OS})$



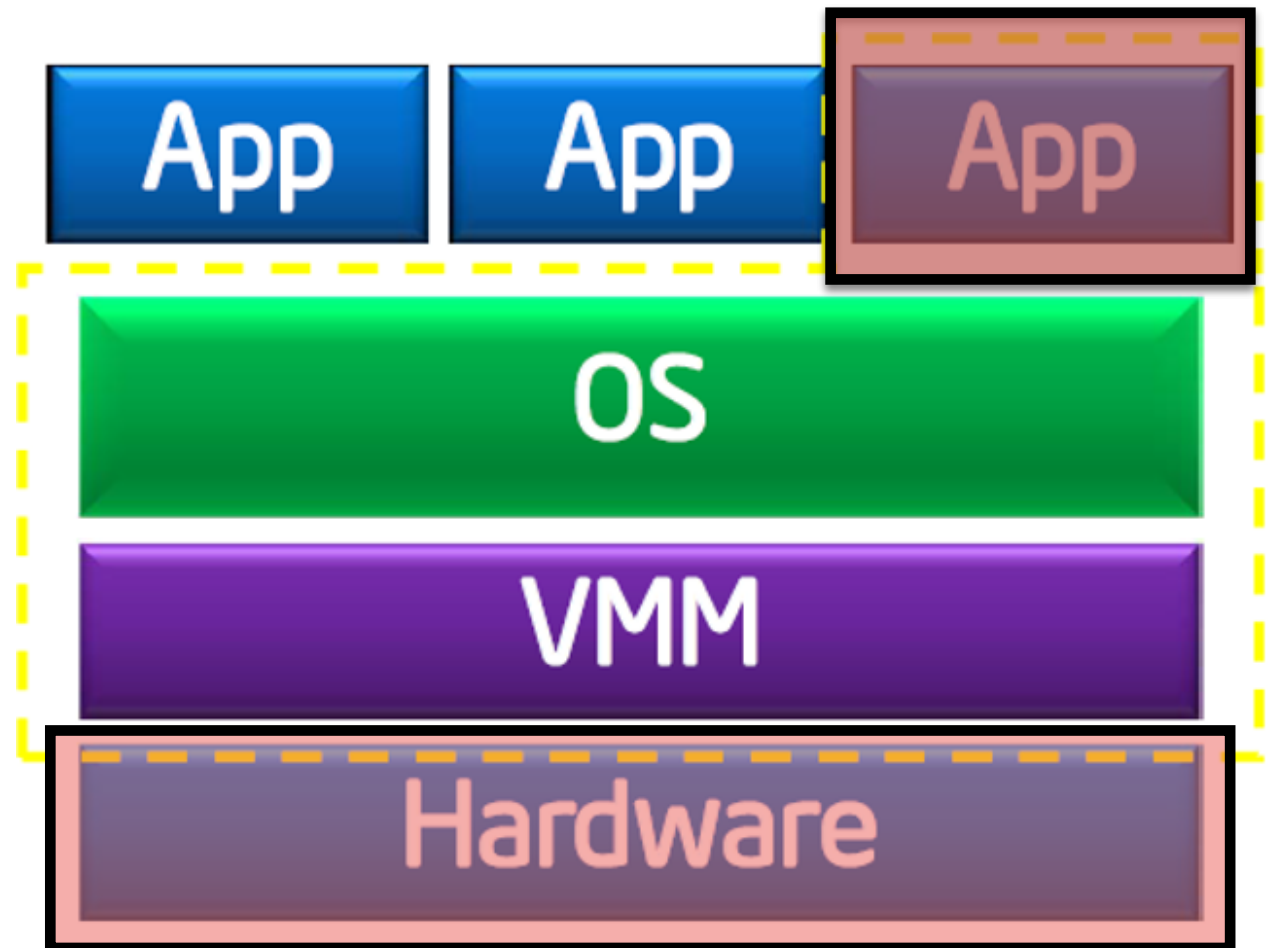
New term: TCB

- **Trusted computing base (TCB):**
all the hardware, firmware, and software components within a computer system that are essential for security.
- Compromise of any component in TCB breaches security

Reasons to use hardware: reducing software TCB



TCB in a typical system



Ideal TCB: app itself, plus hardware

Reason #2: Performance

- A limited version of TEE can be hypothetically built via FHE + ZKP
 - User encrypts input $c = \text{FHE} . \text{Enc}(k, d)$
 - Server performs $c' = \text{FHE} . \text{Eval}(c, \text{Circuit})$
 - User decrypts $d' = \text{FHE} . \text{Dec}(k, c')$
 - FHE ensures: $d' = \text{Circuit}(d)$
- **Slow:** $\sim 10^3 - 10^6 \times$ slower [though improving]
- **No integrity:** need, e.g., ZKP, and additional overhead
- Support **multiple users** require multi-key FHE (even less practical)

Comparison of primitives

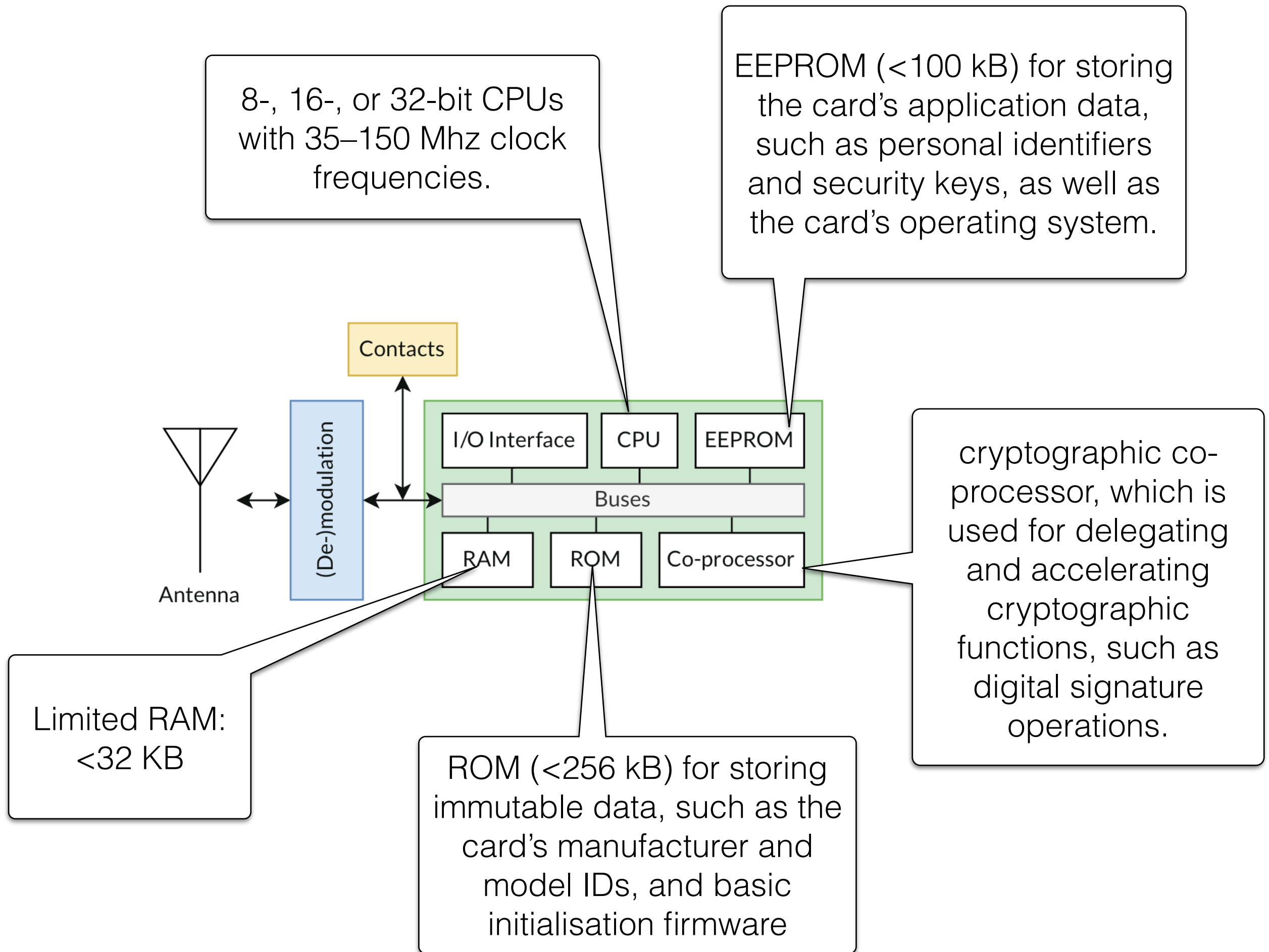
- **Zero-knowledge proofs:** no privacy from the prover
- **Multi-party computation:** relies on threshold assumption, subject to collusion.
- **FHE:** no integrity by default. Don't support multi user inputs by default.
- **TEEs:** a pragmatic alternative, though introducing hardware assumptions and risks.
 - We will study these risks & how to mitigate some of them

History of TEEs

Smart Card

- Smart card is a type of plastic card that has an **embedded microprocessor** and **associated memory**.
 - E.g, credit cards, ID cards
 - First developed in 1980s
- Isolated execution
- Secure storage





Security goal:

Isolation of secrets

- Malicious users can't exfiltrate security assets (e.g., keys, sensitive data)
- Hardware & software protection
 - tamper-resistant hardware
 - shielding layers and randomized layouts
 - Immutable code or immutable
- **Verifiability** is out of scope b/c of immutable code.

Example: payments

Card is *personalized* before issued to customer: Certificate chain C, key pair (SK, PK), unique per-card key MK.



C, (SK, PK), MK



Nonce n

C, Sig(SK, n)

Transaction T

$M = \text{Mac}(K, T)$

Derive session key
K from MK

T, M



This is extremely simplified (full standard called EMV).

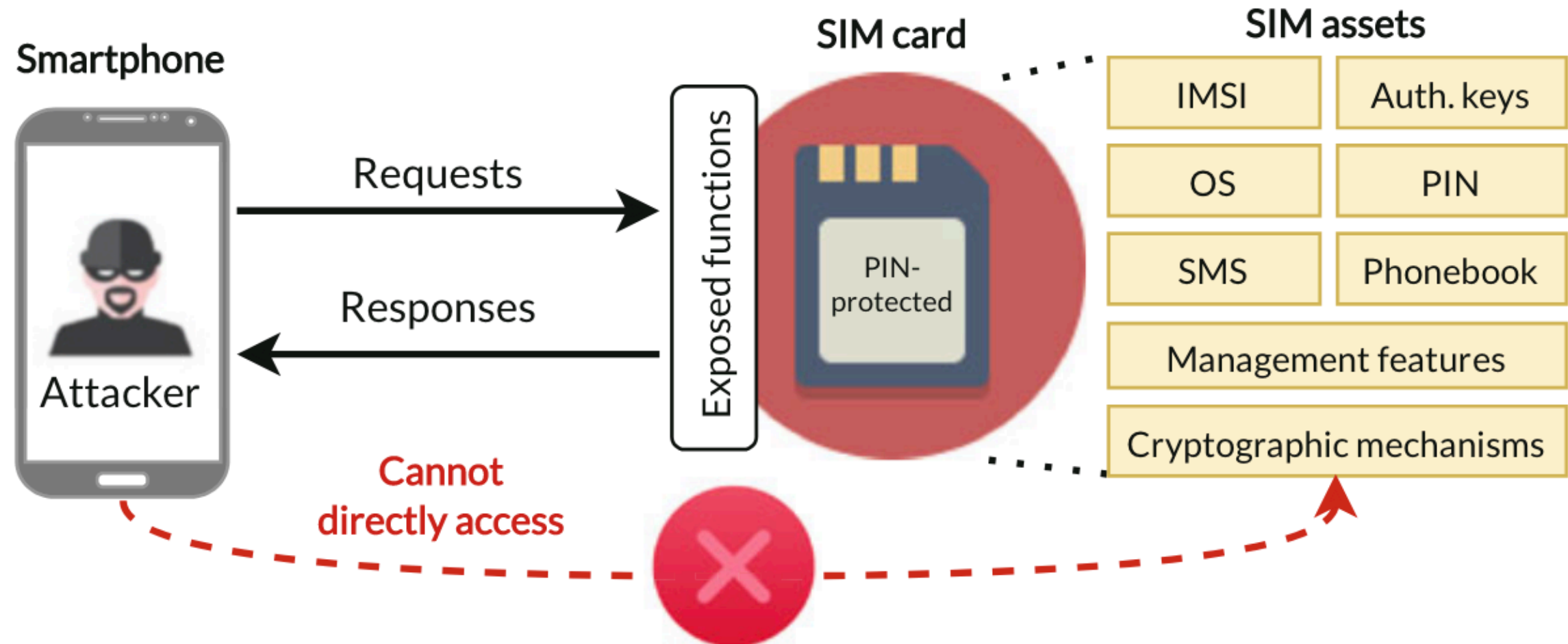
Using hardware to
protect secrets from
privileged software

Secure Elements (SEs)

- Early examples include SIM card
- Modern examples include:
 - Apple's Secure Enclave
 - Google's Titan M
 - TPM

Example of SE: SIM card

- Phone invokes **well-defined APIs**, such as “send SMS”
- Security goals: an attacker cannot access the secrets on a SIM card by gaining **root access** on the phone



Trusted Platform Modules (TPM)

- Early 2000s, the idea of “trusted computing” emerged: shifts the focus from isolated secure storage to **bootstrapping trust**



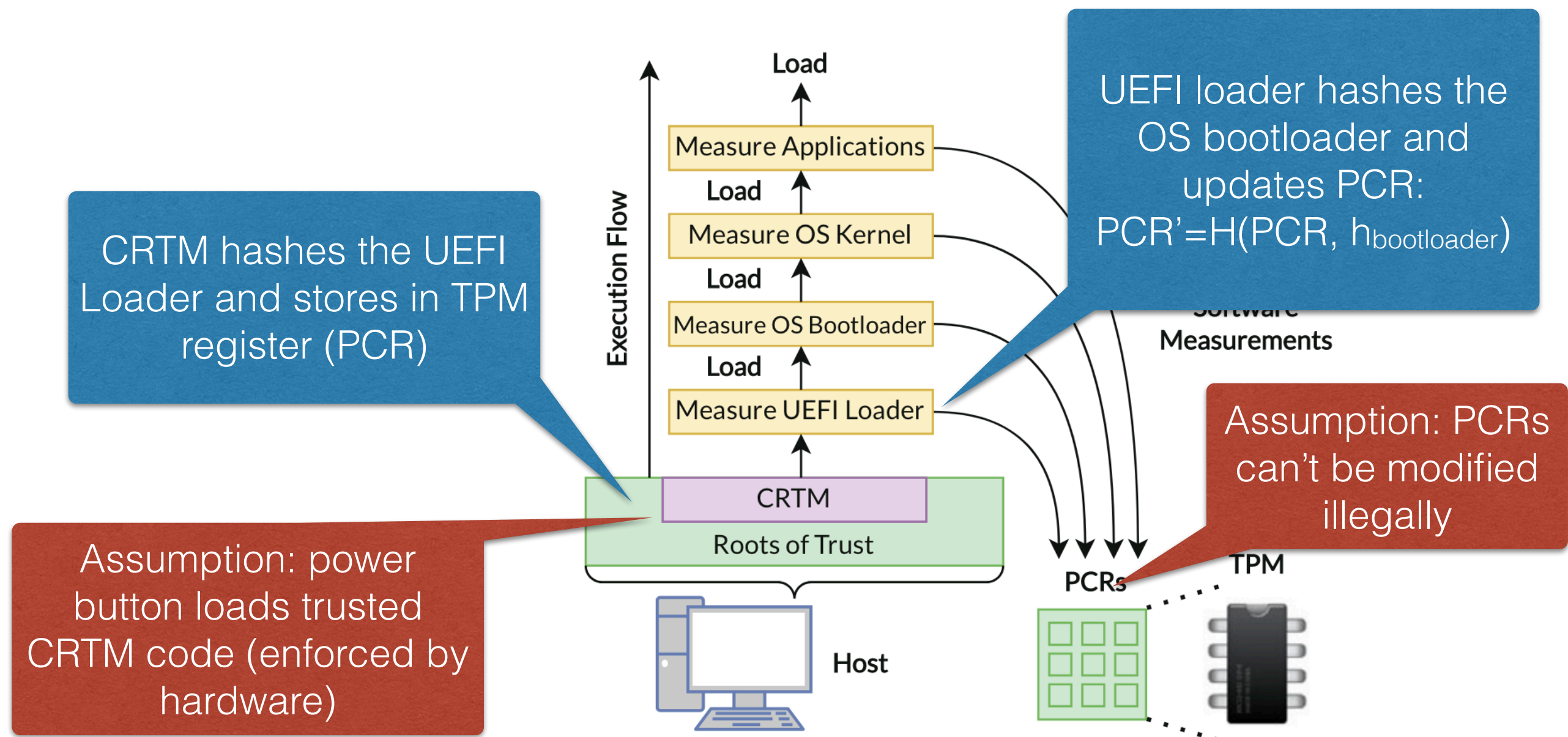
Windows 11 officially requires TPM 2.0



Apple has something similar

How to? Measured boot

- Goal: record what's booted in Platform Configuration Registers (PCRs).



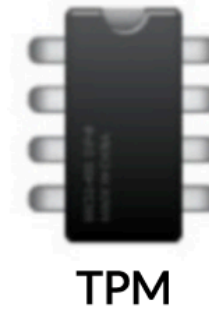
Measured boot (cont.)

- Goal: record what's booted in Platform Configuration Registers (PCRs).
- Each boot component generates a hash (**measurement**) of the next component.
 - stored in PCRs
- PCR values be used for:
 - **Remote attestation** (proving system state to a remote verifier)
 - **Sealing secrets** (only accessible if system matches a known-good state)

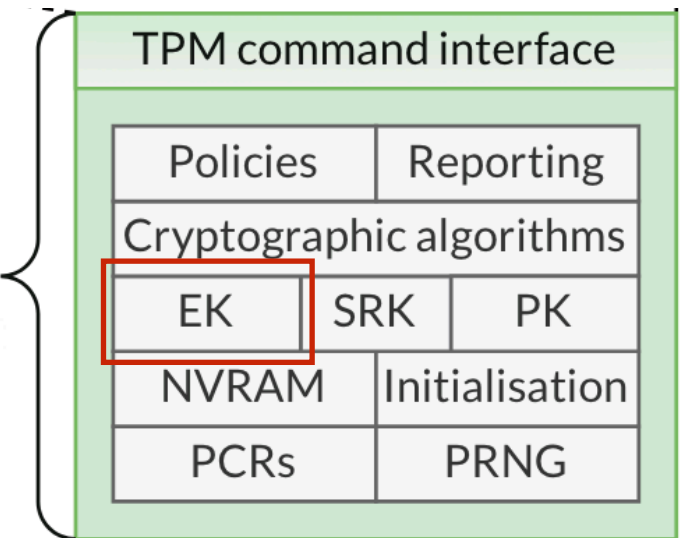
Remote Attestation

- Verifier has expected PCR values.
- TPM signs on the *actual* PCR values.
- Verifier compares the two.
- A wrinkle to the problem: what public key should the Verifier use to verify signatures?
- Typical PKI solution: involves **TPM manufacturer**
 - Manufacturer embeds unique **root keys** to each TPM
 - can't be changed or migrated.
 - Manufacturer publishes all PKs that belong to genuine TPMs

Root keys in TPM



TPM



- Endorsement Key (EK)
 - used for attestation
 - a RSA or ECC key pair
 - a unique key used to *identify* a TPM
 - only usable through well defined APIs and never exported!

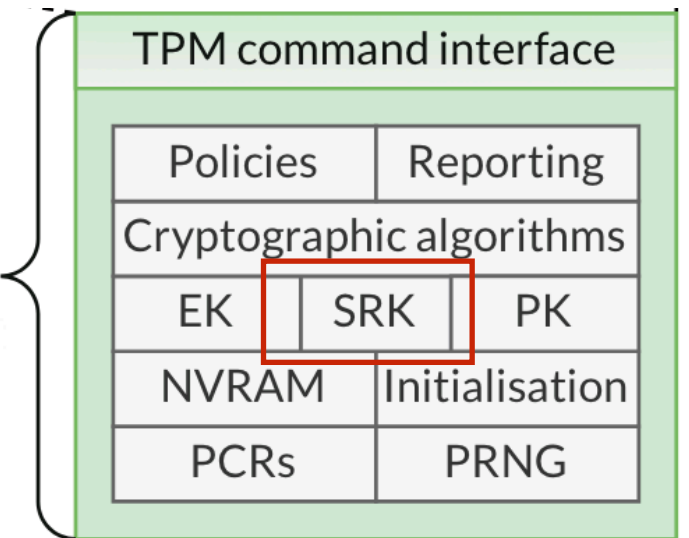
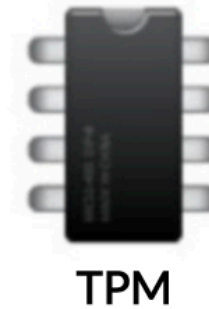
Remote Attestation (cont.)

- Privacy concern:
 - Typically, verifiers are companies (e.g., movie steaming platforms, e-book rental, etc.) and TPM run on user's computer.
 - Leaking TPM's unique PK enables tracking
 - We typically don't like to be tracked cross services.
 - Cryptography comes to rescue: group signatures
- Security concern: game over if Endorsement Key is leaked...

Sealing Secrets

- TPM can encrypt data that can only decrypt by **the exact same code**.
 - “binds” data to a specific system state
- Implementation in TPM: derive internal encryption keys with PCR value as inputs.
- A root secret key is needed since PCR values are non-secret

Root keys in TPM

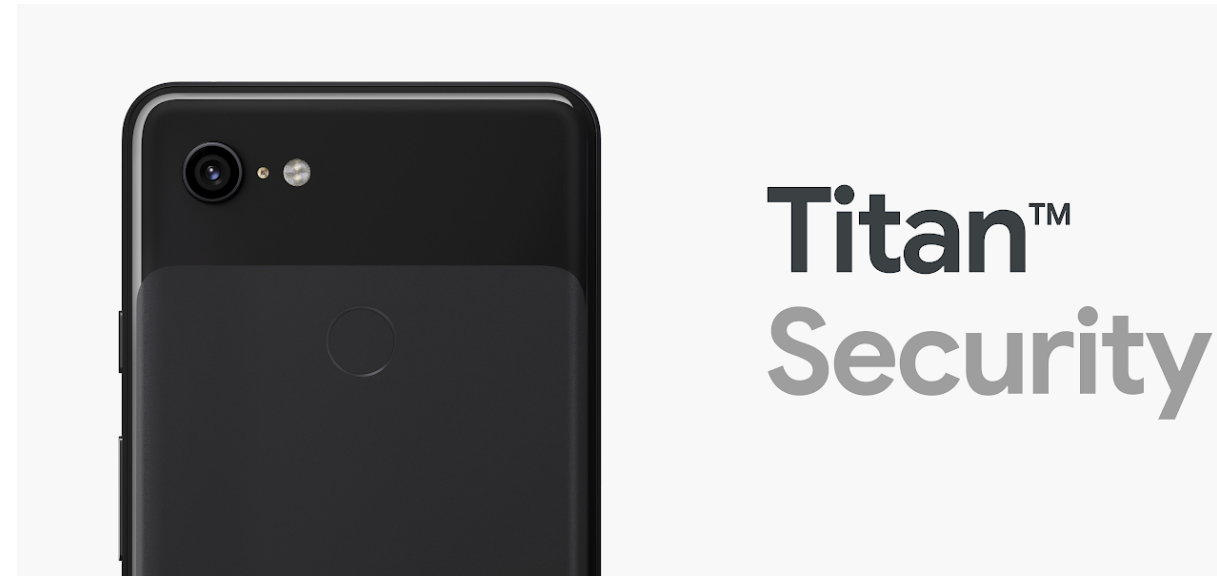


- Storage Root Key (SRK)
 - used for encrypting/decrypting data and generate other keys
 - generated locally upon user initialization
 - *Not* known to the manufacturer
 - only usable through well defined APIs and never exported!
- Implies some challenges: leakage, migration.

Key concepts introduced in TPM

- **Measurement:** digest of the system state
- **Remote attestation:** proving the system state to a remote verifier
- **Sealing:** encrypt data to a specific measurement

TPM-like SE



- Google Titan M
- an embedded SE used by flagship Google phones since Pixel 2 in 2017
- Proprietary; limited public info on its design.
- Used for
 - Android Verified Boot (secure boot)
 - Boot process verifies the signature of bootloader
 - Android Strongbox Keymaster (generate and manage cryptographic keys)

Apple Secure Enclave

- An isolated processor with its own cache and runs a mini OS “SEP OS”
- Hardware-based key manager
 - e.g., for file based encryption
- Biometric is protected by SEP
 - Leakage is fatal, as it’s hard to get new fingers/faces
- Secure boot
- Equipped with various protection against *physical attacks* (e.g., undisclosed countermeasures against simple and differential power analysis attacks)
- Few details about its design is known to public



Hardware standards are messy

- While most cryptography primitives are standardized (and no-one uses non-standard crypto), hardware is messy
- The definitions of smart cards, SE, TPM, TEEs all all kind of blurry
 - E.g., some classify sim card as SE
- But: certain terms are consistent: attestation, sealing.

Limitations of TPM

- Recall our goal: a “execution environment” isolated from the rest of the system, supported by a minimal TCB
- TPM can measure TCB, but doesn't reduce it

Intel Software Guard Extensions (SGX)

- Security goals: “secure enclave” for application code
 - i.e., TCB = app + SGX
- Non-security goals: speed, compatibility with x86
- Implementation: 18 instructions added to x86
 - Since Intel Skylake architecture (2016)



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

`sysenter/sysexit` Instructions

Background: Security Mechanisms in ***standard*** CPUs

- CPU and memory

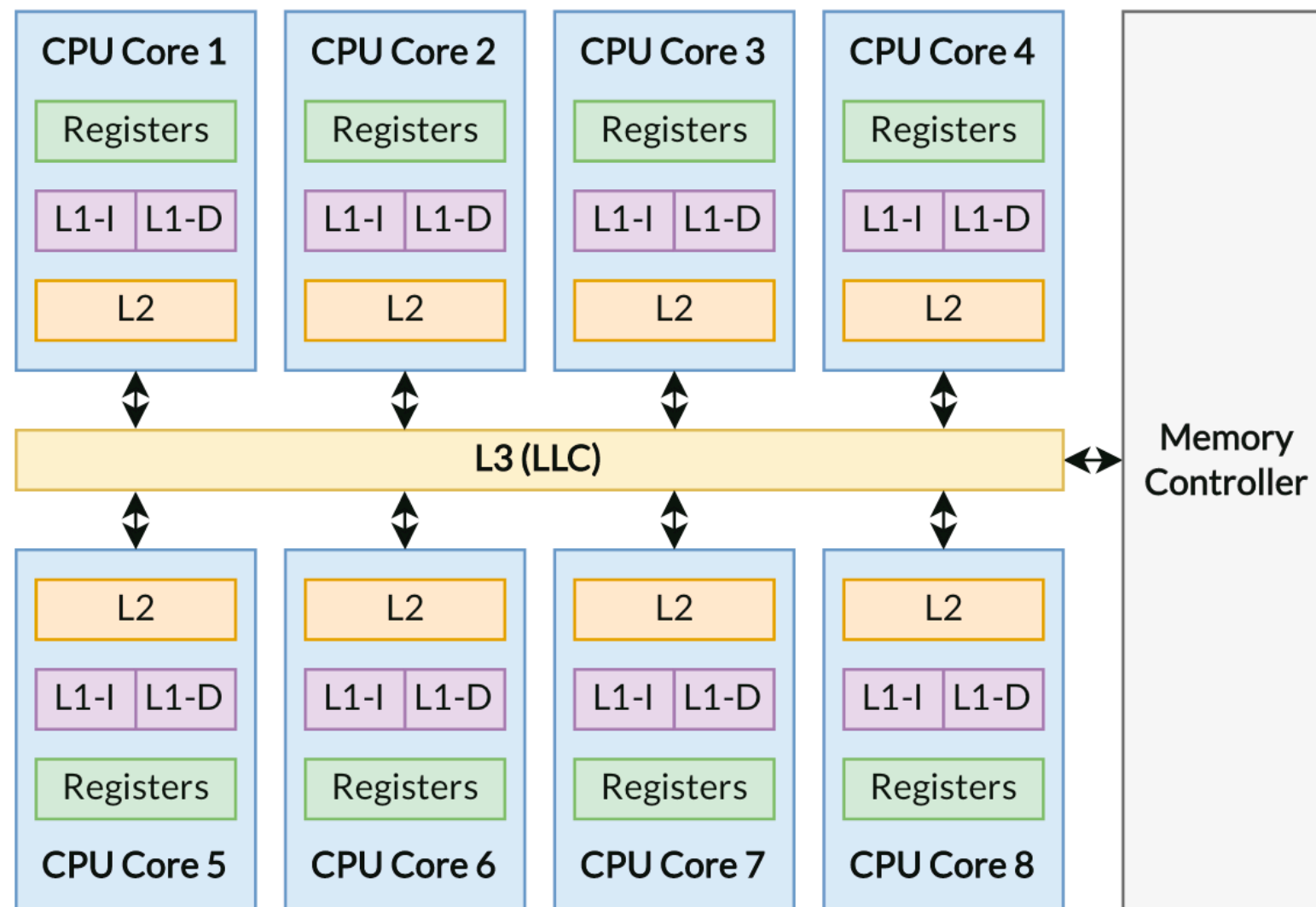


Fig. 2.1 High-level system diagram of a modern eight-core X86-64 CPU

- Memory hierarchy

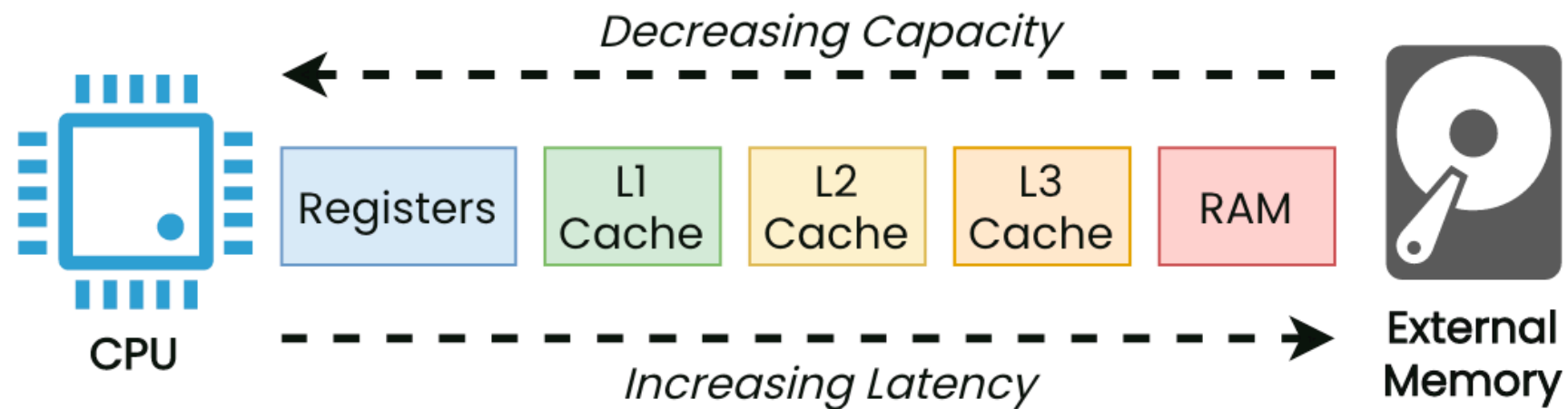
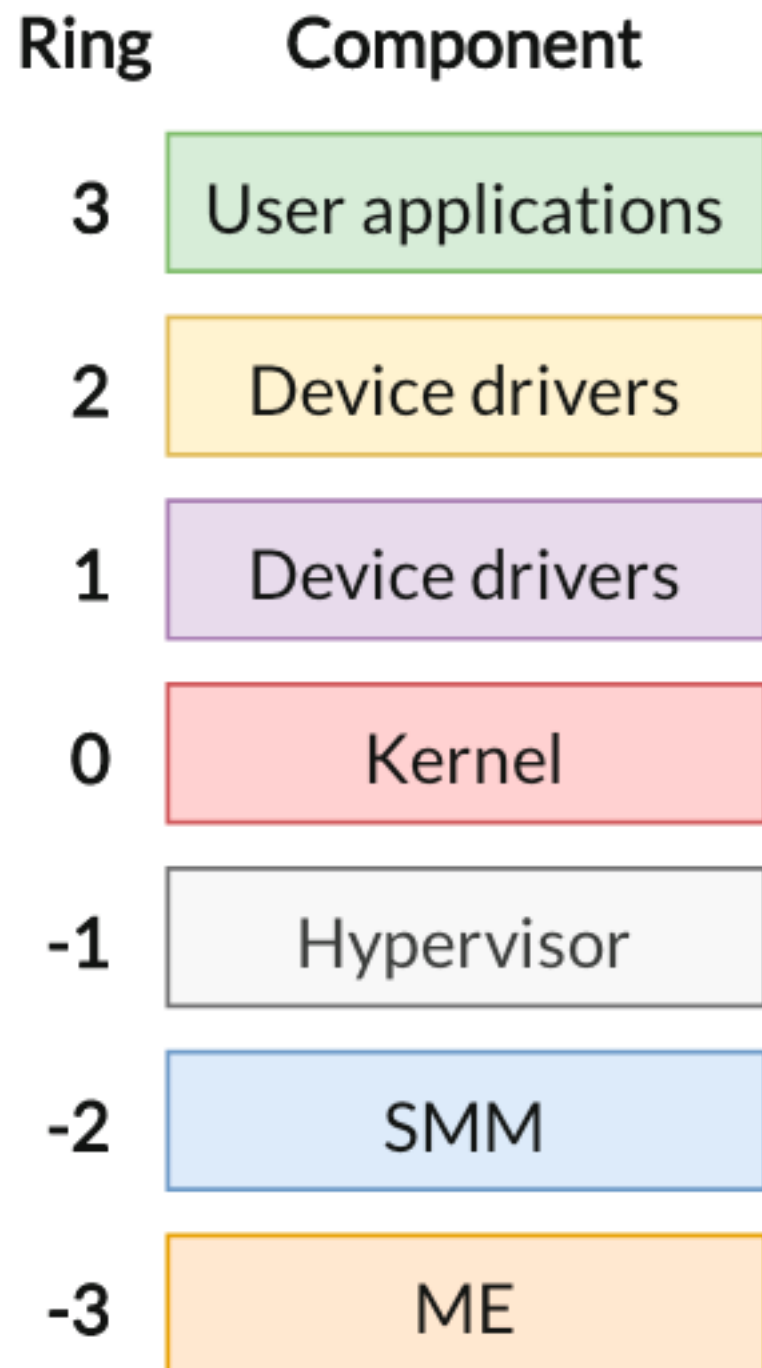


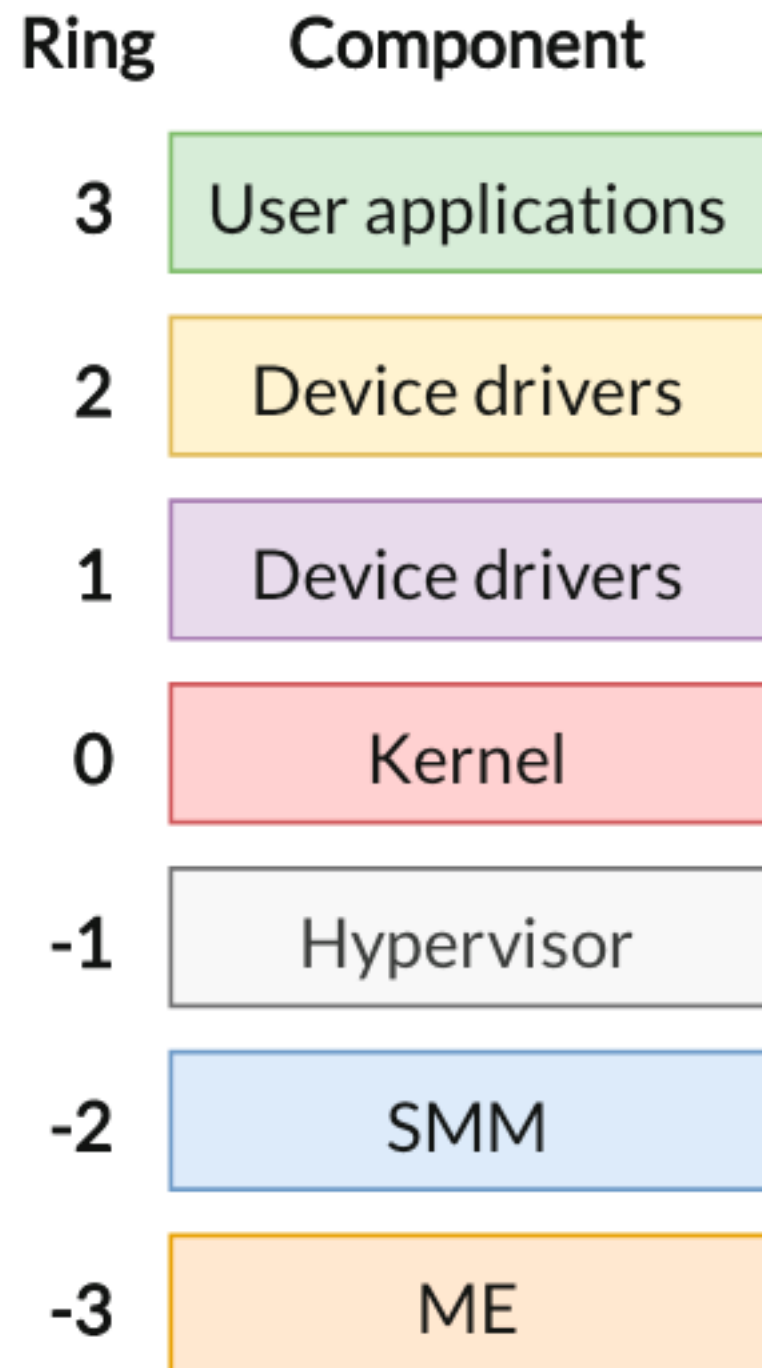
Fig. 2.2 A conventional memory hierarchy

Security mechanisms



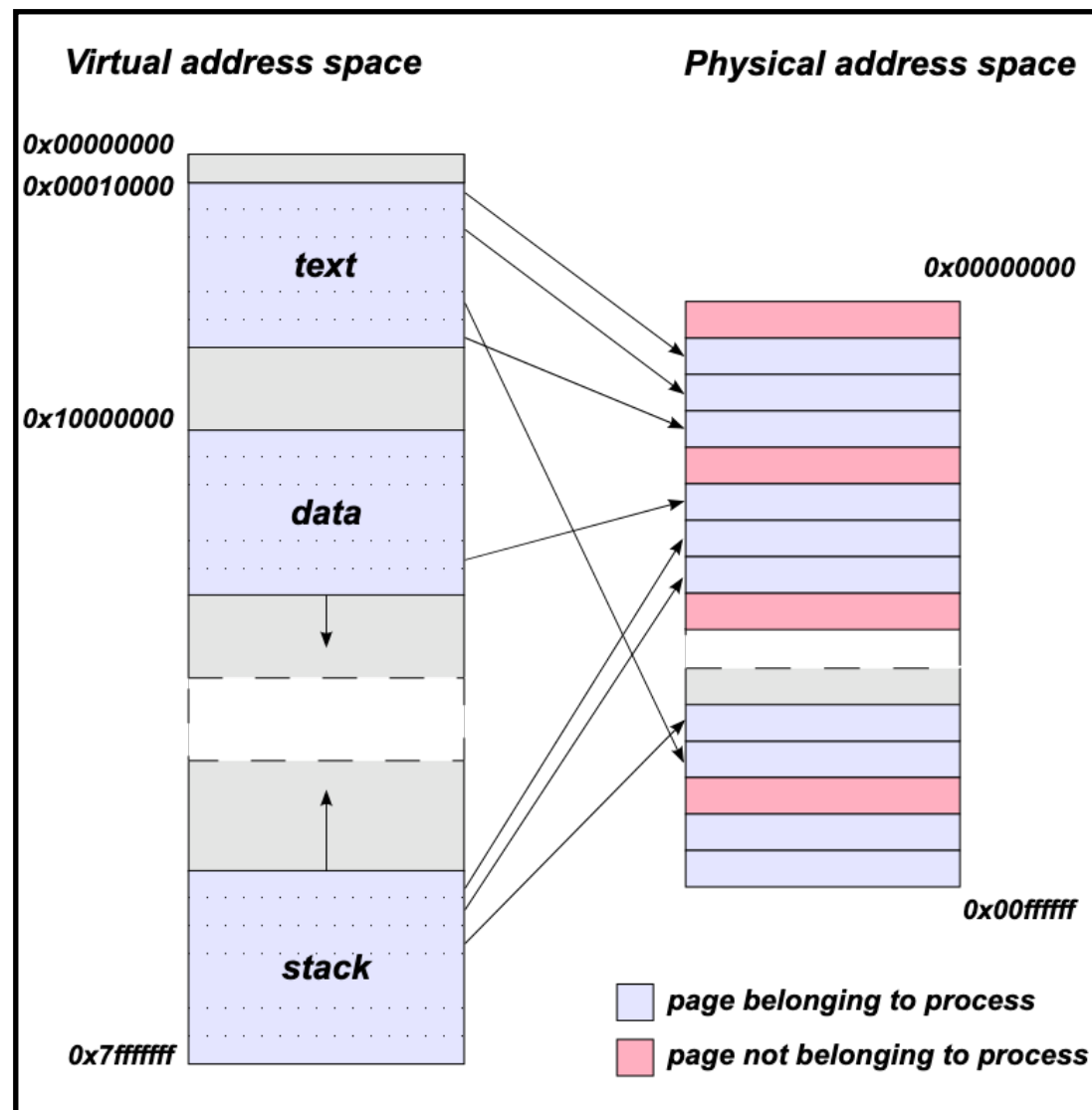
- CPU-enforced restrictions
 - User code can't run privileged instructions
- Memory isolation
 - Between processes & between user and kernel memory
- Controlled kernel access

Security mechanisms: OS self-protection



- CPU tracks Current Privilege Level (CPL), aka, the ring
 - CPU errors if a **privileged instruction** is invoked in ring 3
- OS puts user code in ring 3
- Carefully designed so that user code can't escalate to ring 0 unless through well defined interfaces
- Mechanism: storing “code privilege” and privileged code in ring-0 memory

Security mechanisms: memory isolation



- Each “app” (process) owns certain pages in the memory
 - OS tracks page ownership and permissions in a page table in ring-0 memory
- CPU errors when
 - Ownership error (pages being accessed do not belong to the current process)
 - Permission error (e.g., write to a read-only page)
- User app can't modify page table b/c previous slides.

Controlled kernel access

- Ring 3 code can temporarily escalate to ring 0 through *system calls*
 - Linux: 300-400
 - Win32: 100+
 - macOS: ~300+
- Assumption: syscalls are correctly designed such that users can't take over the OS.
 - Some OSes are formally verified (e.g., seL4)

End of Background: Security
Mechanisms in ***standard***
CPUs

Next: Changes by Intel SGX

A challenging adversary!

- Malicious OS is a **very powerful** adversary!
- Enclaves are created by untrusted code
 - it may modify code, or bypass SGX (emulate a fake enclave)
- All requests go through untrusted code
 - it may pass requests to wrong enclaves, or fake enclaves.
- OS can read any part of the memory
- Page tables are managed by untrusted OS
 - OS can manipulate the mapping from virtual to physical address
- Exceptions and interrupts will stop trusted code and start handlers (by untrusted OS)
 - Untrusted OS can inject interrupts and exceptions at will

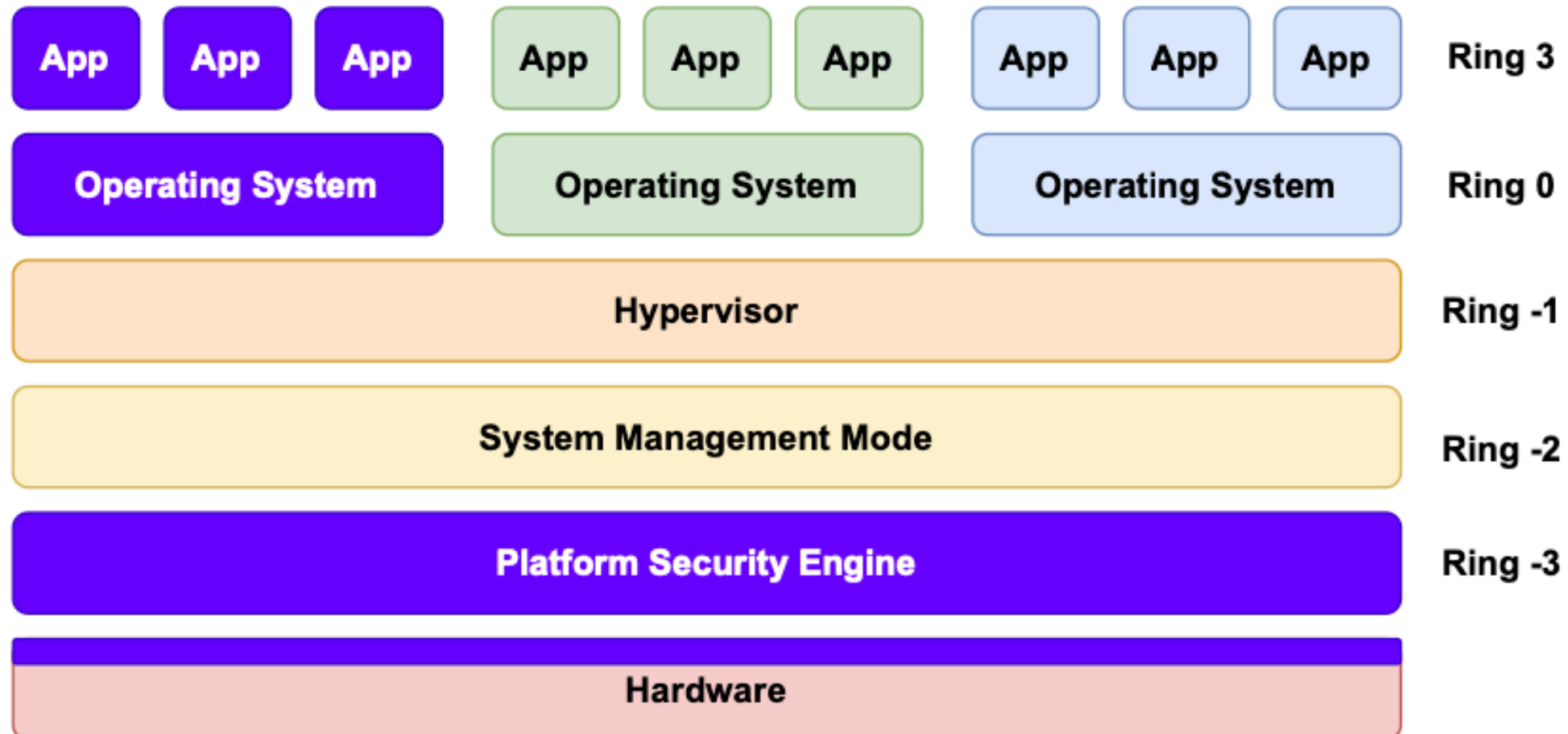
Quiz

- Availability is straight up impossible. Why?

Alternative “TEE” designs

Secure VM

- Larger TCB in general (guest OS and apps)
- Easier development (not need to modify apps)
- Main examples: Intel TDX and AMD SEV (more on this later)



GPU

- Nvidia H100 GPU: can run AI in TEEs

 **NVIDIA DEVELOPER** [Home](#) [Blog](#) [Forums](#) [Docs](#) [Downloads](#) [Training](#)

Technical Blog [Filter](#)

Data Center / Cloud

Confidential Computing on NVIDIA H100 GPUs for Secure and Trustworthy AI

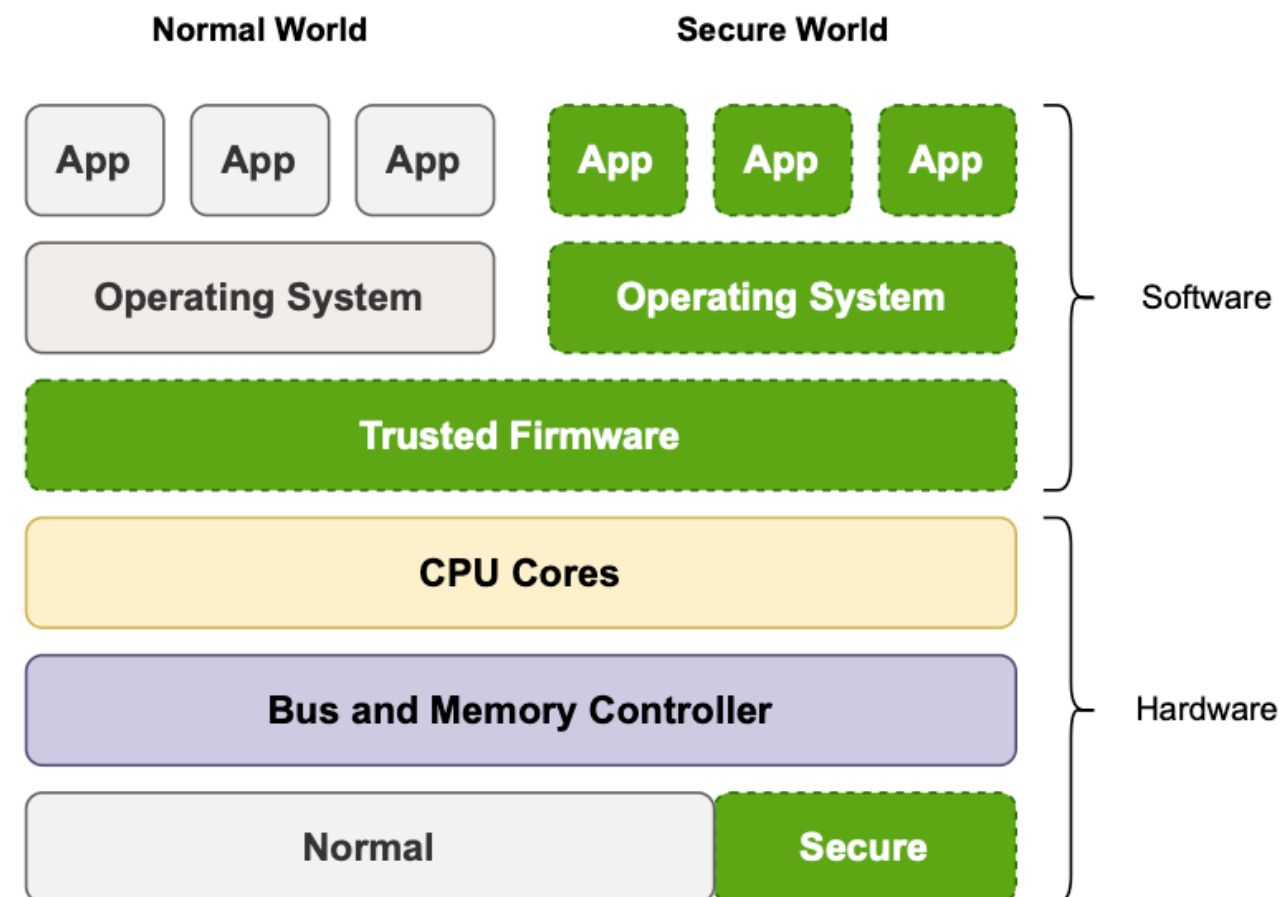
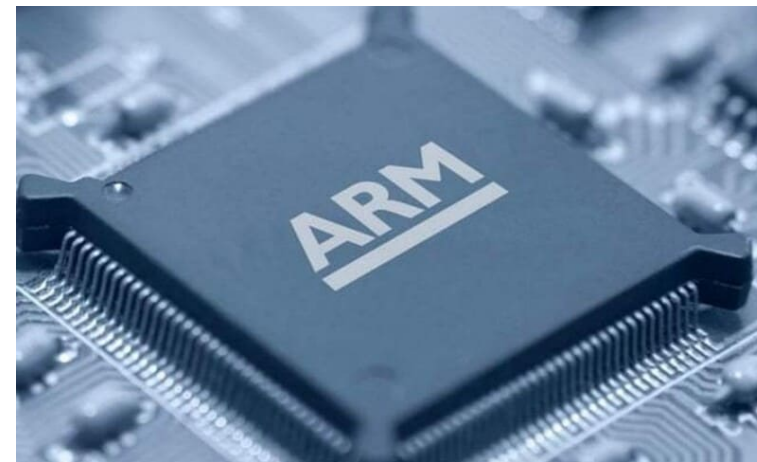
Aug 03, 2023

 +12 Like  Discuss (1)

By [Emily Apsey](#), [Phil Rogers](#), [Michael O'Connor](#) and [Rob Nertney](#)

ARM TrustZone

- Very different from SGX
- Mainly about isolation (no remote attestation)
- One TEE per system
 - Runs security critical applications and a small trusted OS
- Normal world can call secure world through well-defined interface



Summary

- **Trusted execution environments (TEEs):** confidentiality, integrity, remote attestation
- **Why hardware?**
 - Reducing software TCB (e.g., OS out of TCB)
 - Performance advantage (many orders of magnitude than ZKPs - verifiability, FHEs - for confidentiality)
- **Key difficulty in designing TEEs:** strong adversary model: resource manager (OS, hypervisor, etc) is no longer trusted.