More Isolation and Hardware Security

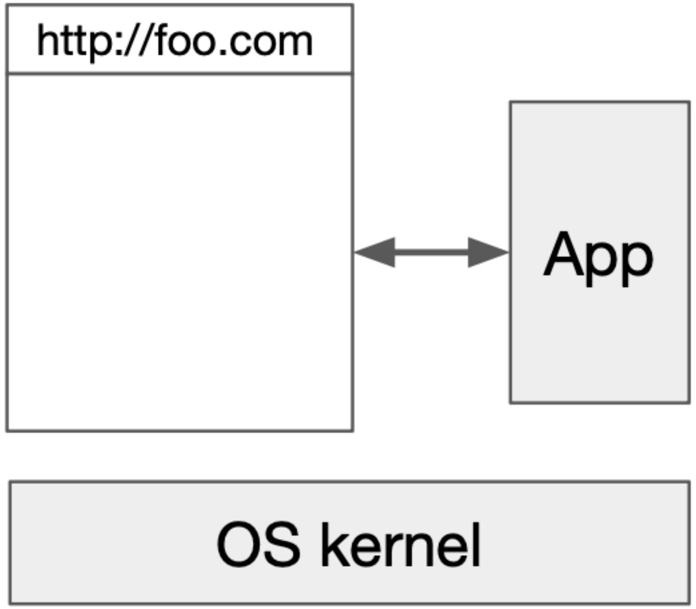
Dileepa Fernando

Overview

- Software Fault Isolation
  - Ensuring the apps are trusted
- Trusted Computing
  - Ensuring the trust in platform (OS, HW)

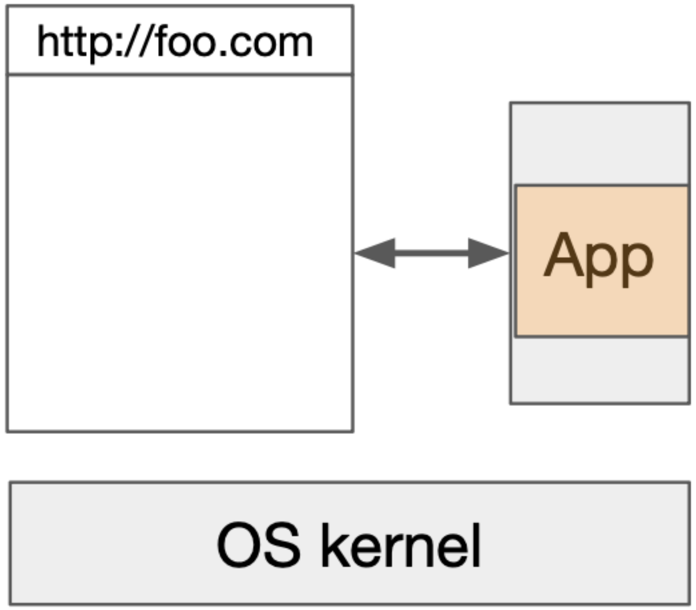
More Isolation

- Isolation So Far
  - Process
  - VM
  - Container
- Isolation enforced by manager
- Problem 1: Isolation good enough?
  - Example:
    - Native Code running on browser
    - Running as different process
    - Communicate with browser process
    - Data and Code are isolated



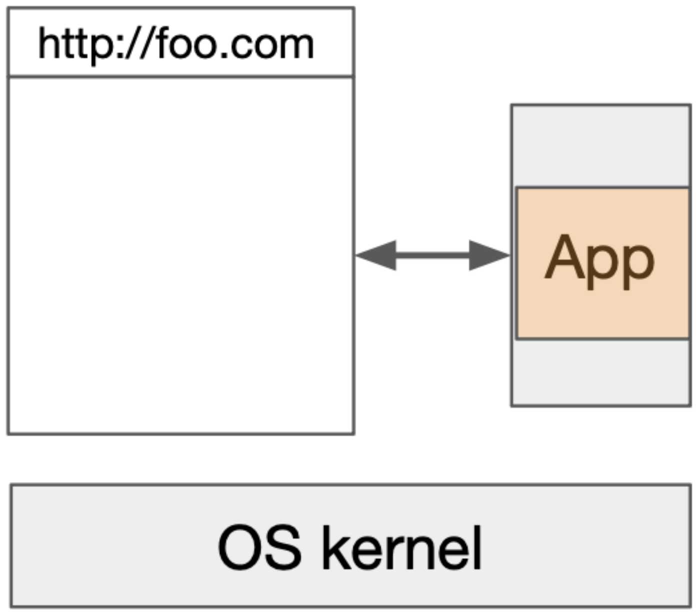
More Isolation

- Problem 1: Isolation good enough?
  - Example:
    - Native Code running on browser
    - Running as different process
    - Communicate with browser process
    - **Browser is Paranoid**
      - Does not Trust app
      - Does not Trust OS
      - Implement own isolation
    - Idea: Software Fault Isolation



More Isolation

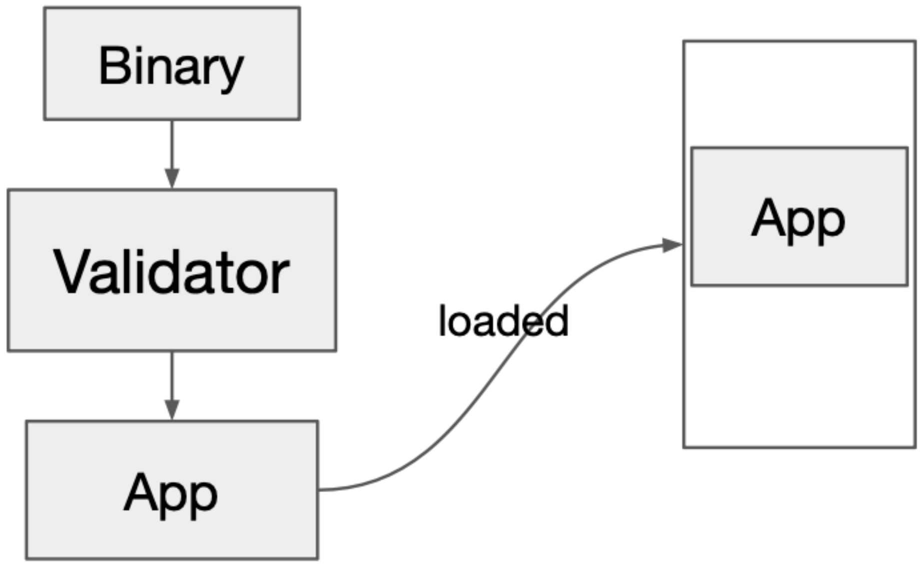
- Problem 2: TCB size
  - Idea: Secure Hardware
  - Trusted Execution Environment (TEE)



Software Fault Isolation

- Confining apps inside sandbox https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/34913.pdf
- Security Goal: App only access its own memory
  - Can communicate with other processes
  - Idea
    - Static Validation of App binary
    - Run time check of App binary

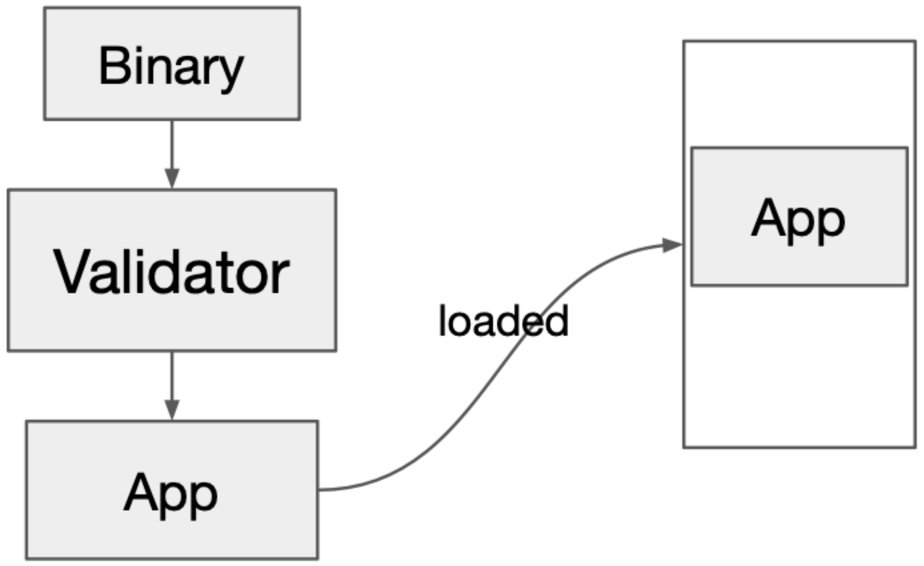
  - How sand box differs from container?
    - Security goal?
    - Threat Model?



Software Fault Isolation

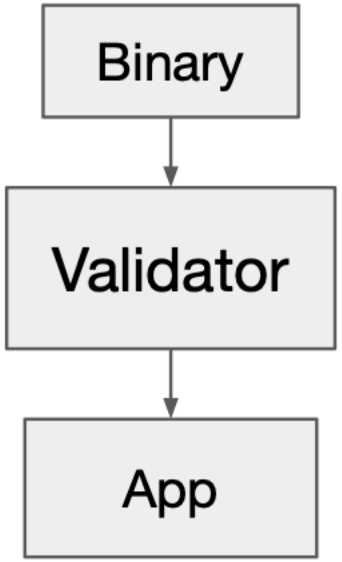
- Idea 1: Restricted Instructions
- Idea 2: Controlled Interaction

- Recall the goal
  - No memory access outside the app



Software Fault Isolation

- Idea 1: Restricted Instructions
  - Some instructions are safe
    - ADD, XOR
    - Allowed
    - Checked later in run time
  - Some instructions are dangerous
    - JMP
    - Insert Check
  - Some are hard to make safe
    - INT, SYSCALL
    - Disallowed



Software Fault Isolation

- Idea 1: Restricted Instructions (Validation)
  - Safe → Do nothing
  - Dangerous → Rewrite
  - Unsafe → Abort

  How to rewrite?
  Challenges:
  - JMP addr (wrong addr)
    - JMP 0x0, JMP 0xA ok
  - JMP *EAX (Not known)

  - Any remedy?
    - Alignment (32bit)
    - No instruction consume more than 32 bit

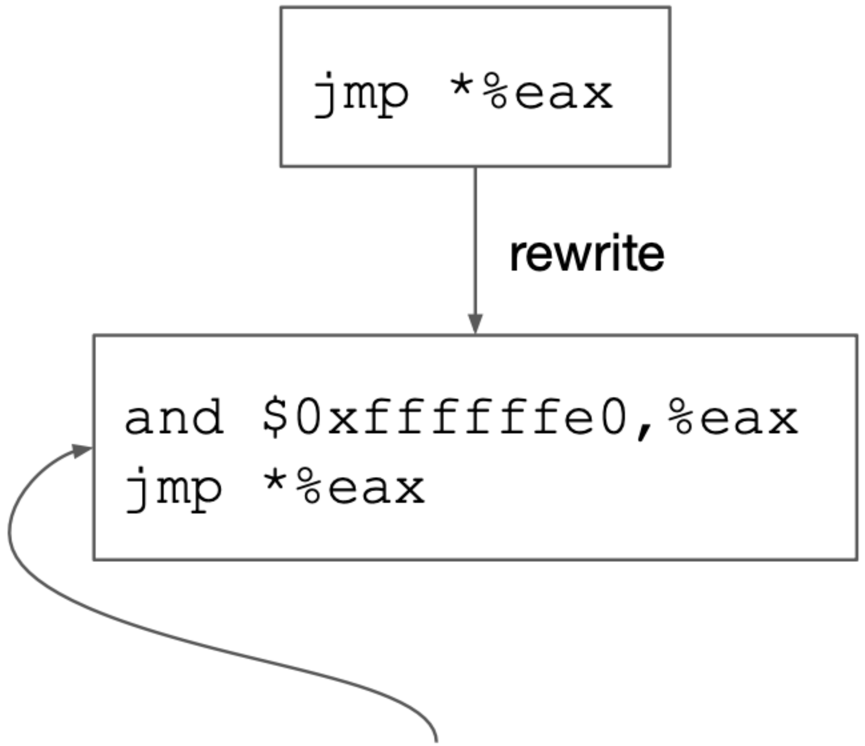
```
f7 c7 07 00 00 00      test $0x00000007, %edi
0f 95 45 c3            setnzb -61(%ebp)

c7 07 00 00 00 0f      movl $0x0f000000, (%edi)
95                     xchg %ebp, %eax
45                     inc %ebp
c3                     ret
```

Software Fault Isolation

- Idea 1: Restricted Instructions (Validation)
  - Safe → Do nothing
  - Dangerous → Rewrite
  - Unsafe → Abort

  Challenges:
  - JMP addr
    - Simple check
  - JMP *EAX

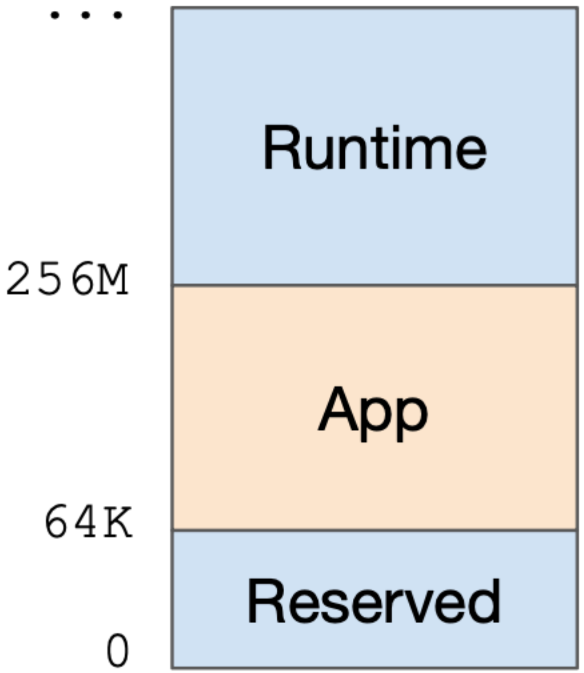
  - Can you jump to the middle
    - Of rewritten instruction?
    - NO
    - How RET is handlef?



Software Fault Isolation

- Idea 1: Restricted Instructions (Validation)
  - Safe → Do nothing
  - Dangerous → Rewrite
  - Unsafe → Abort

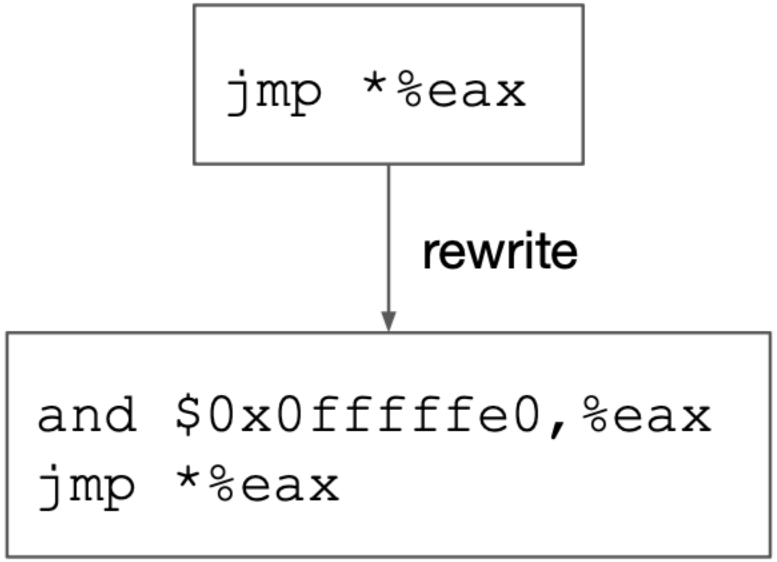
  More Challenges:
  - Out of range access
    - JMP addr
    - JMP *EAX
  - Predefine range (256MB)
  - Start from 0
  - Similar check before jump
    - How to?



Software Fault Isolation

- Idea 1: Restricted Instructions (Validation)
  - Safe → Do nothing
  - Dangerous → Rewrite
  - Unsafe → Abort

  More Challenges:
  - Out of range access
    - JMP addr
    - JMP *EAX
  - Predefine range (256MB)
  - Start from 0
  - Similar check before jump
    - How to?



Software Fault Isolation

- Idea 1: Restricted Instructions (Validation)
  - Safe → Do nothing
  - Dangerous → Rewrite
  - Unsafe → Abort

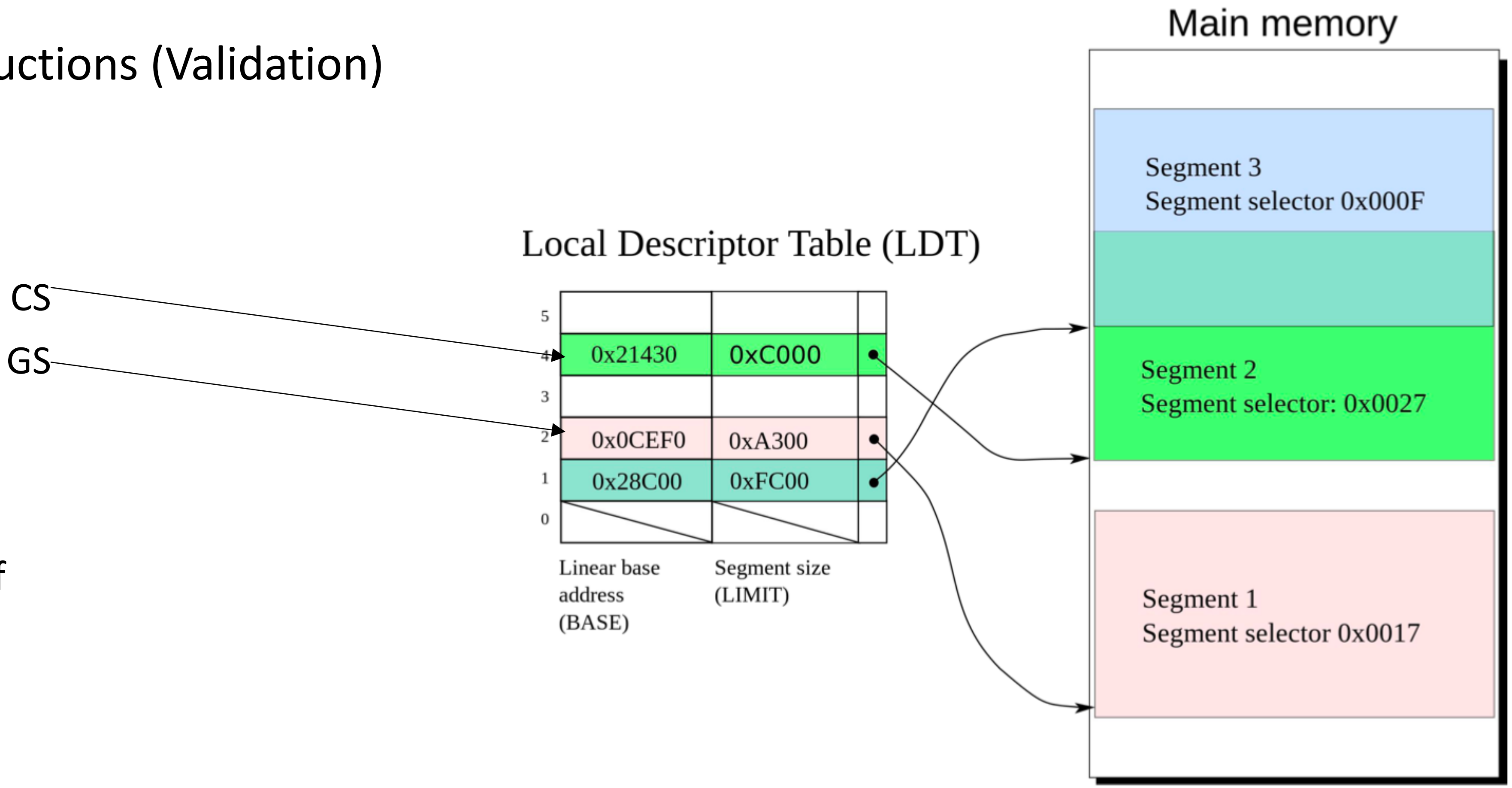
  Any issue with the rewriting?
  - Jump to forced rewrite locations
  - May not be intended by a legitimate developer
  - Better to just detect wrong jumps
  - Use **segmentation**

Software Fault Isolation

- Idea 1: Restricted Instructions (Validation)
  - Safe → Do nothing
  - Dangerous → Rewrite
  - Unsafe → Abort

  - Segment
    - Base
    - Length

  - Ex: Base=0xa0000000
    Length=0x1000
  Addr=0xff → 0xa00000ff

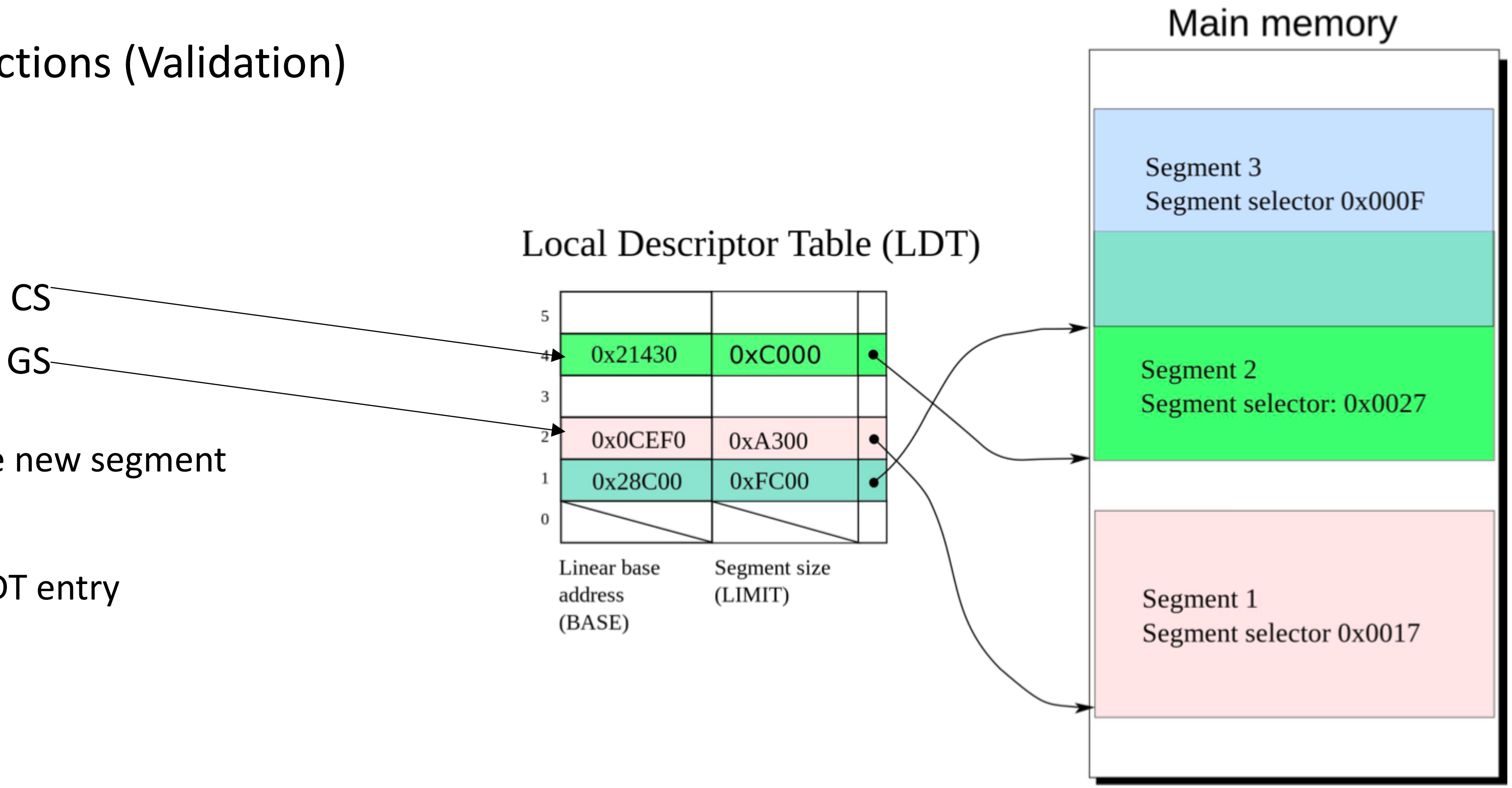


Software Fault Isolation

- Idea 1: Restricted Instructions (Validation)
    - Safe → Do nothing
    - Dangerous → Rewrite
    - Unsafe → Abort

    - Segment
        - Base
        - Length

    - New app loads → Create new segment
        - Length = 256MB
        - Jump out
    - Recall Segment part in IDT entry



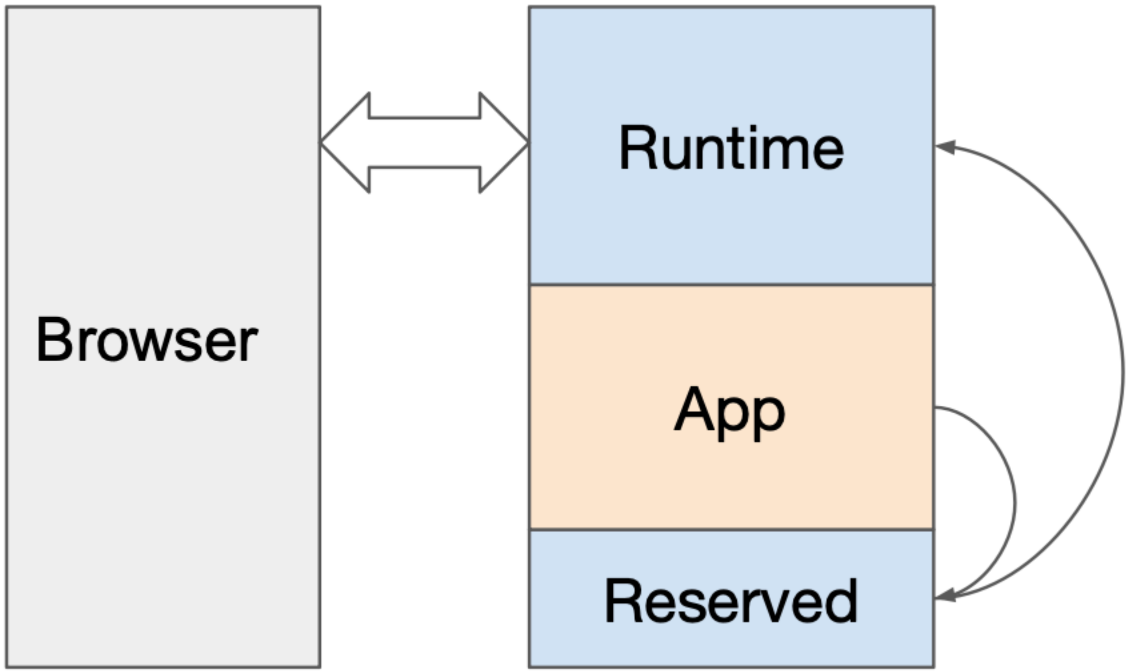
Software Fault Isolation

- Idea 2: Controlled Interaction

  - Send things to outside
  - Get things from outside

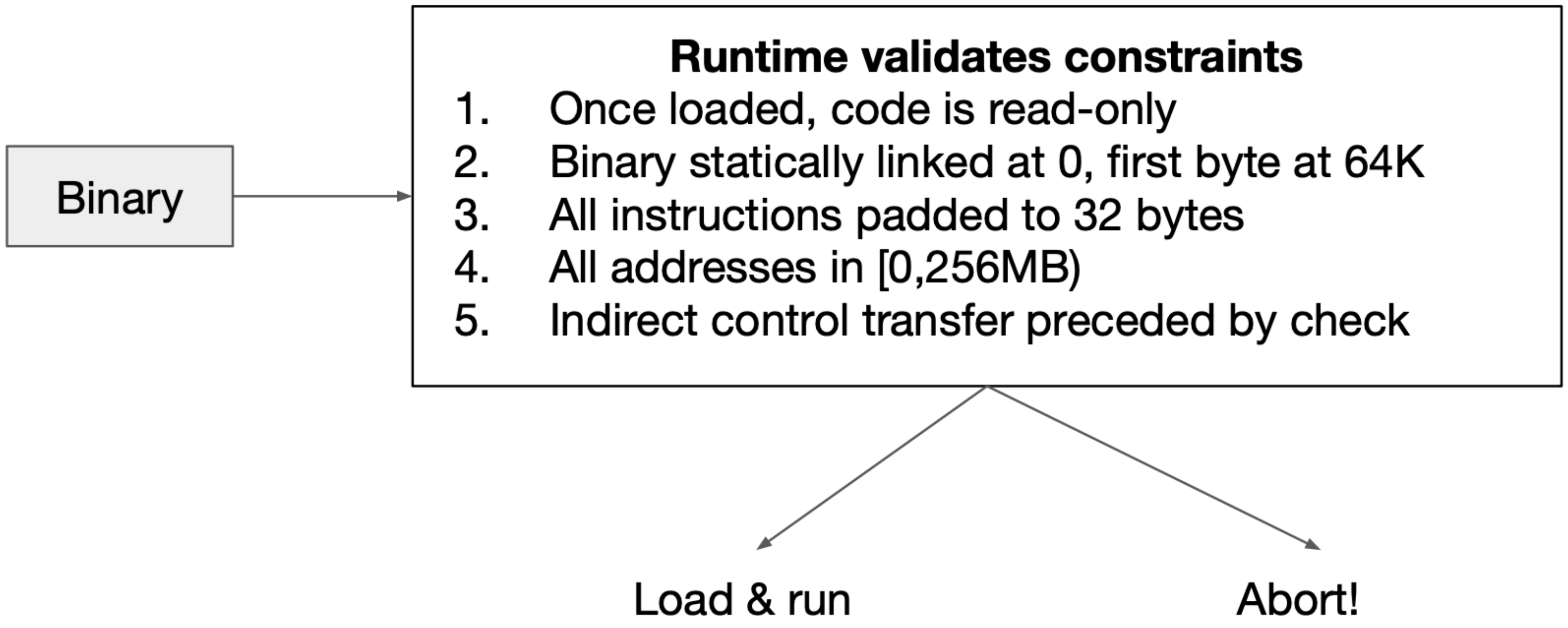
  - How?
    - Apps jump to reserved code
    - Reserved code jump to runtime (safe)
    - Runtime manages IPC with browser

  - Can the app directly jump to runtime?
  - Analogy: Process and kernel



Software Fault Isolation

- Put together



Software Fault Isolation

- Good performance
- Good Isolation

Run #	Native Client	Linux Executable
1	143.2	142.9
2	143.6	143.4
3	144.2	143.5
Average	143.7	143.3

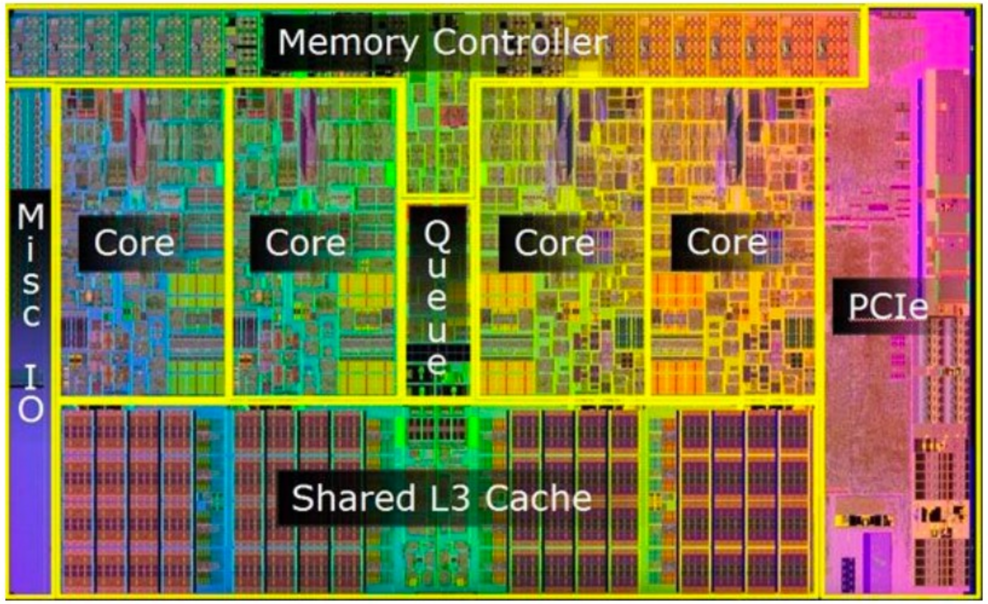
Table 8: Quake performance comparison. Numbers are in frames per second.

- But
  - Static Code
  - Less efficient for system call heavy apps (Why?)

Story So Far

- Secure the **Applications**
- Isolation to the depth (OS, VMM, Container etc.)
- TCB = OS + Hardware (CPU rings, Interrupts, MMU)
  - But how to ensure the **trust** of OS and hardware

- What is **trust**?
  - Secure?(CIA)
  - Verified?
  - Manageable?



Hardware Security

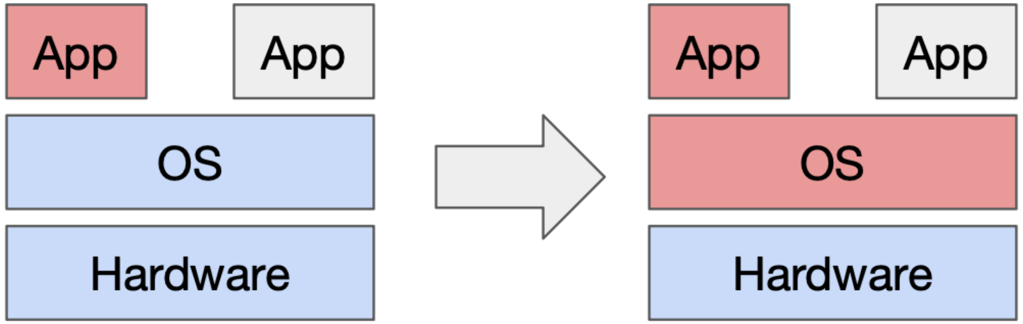
- Security Goals: Application security

- Can we trust any OS? Can Trust a **baseline** OS
- Can we trust any Hardware? Can trust a **baseline** hardware

- Secure and Verifiable?
  - Verify the current OS/Hw with the baseline
  - But cannot always use a trusted OS baseline

Hardware Security

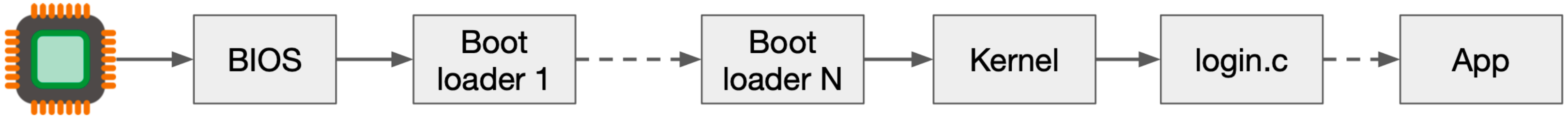
- Revised Security Goal: Application security Despite Malicious OS



- Detect If Cannot Prevent
  - Detect Malicious OS (TPM)
- Minimize TCB and Isolation of Secure apps
  - Trusted OS + Trusted Hw (ARM TrustZone)
  - Trusted Hw Only (Intel SGX)

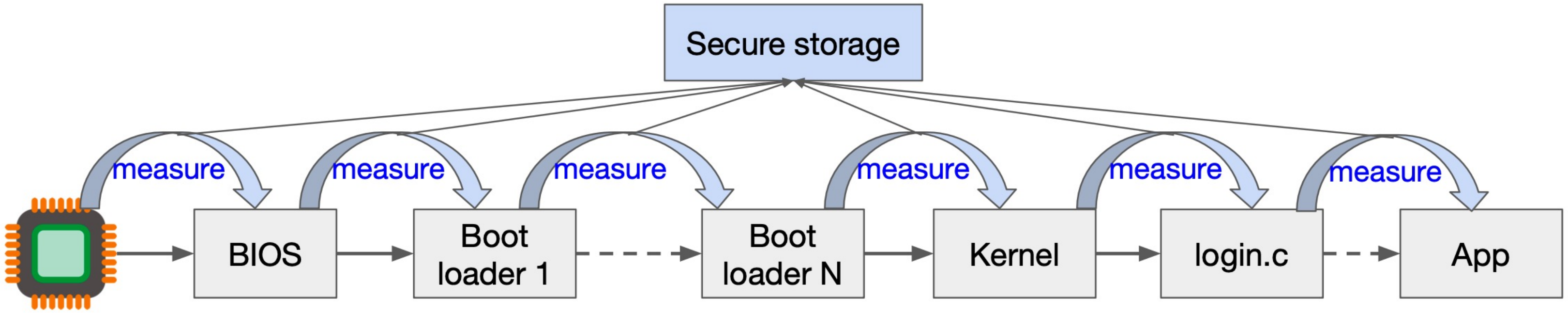
Hardware Security

- **Detect malicious OS**
  - Boot Loader N Verifies Kernel
  - What if Boot Loader N is malicious?
  - Verify recursively until hardware left to trust
  - Remember digital certificates?



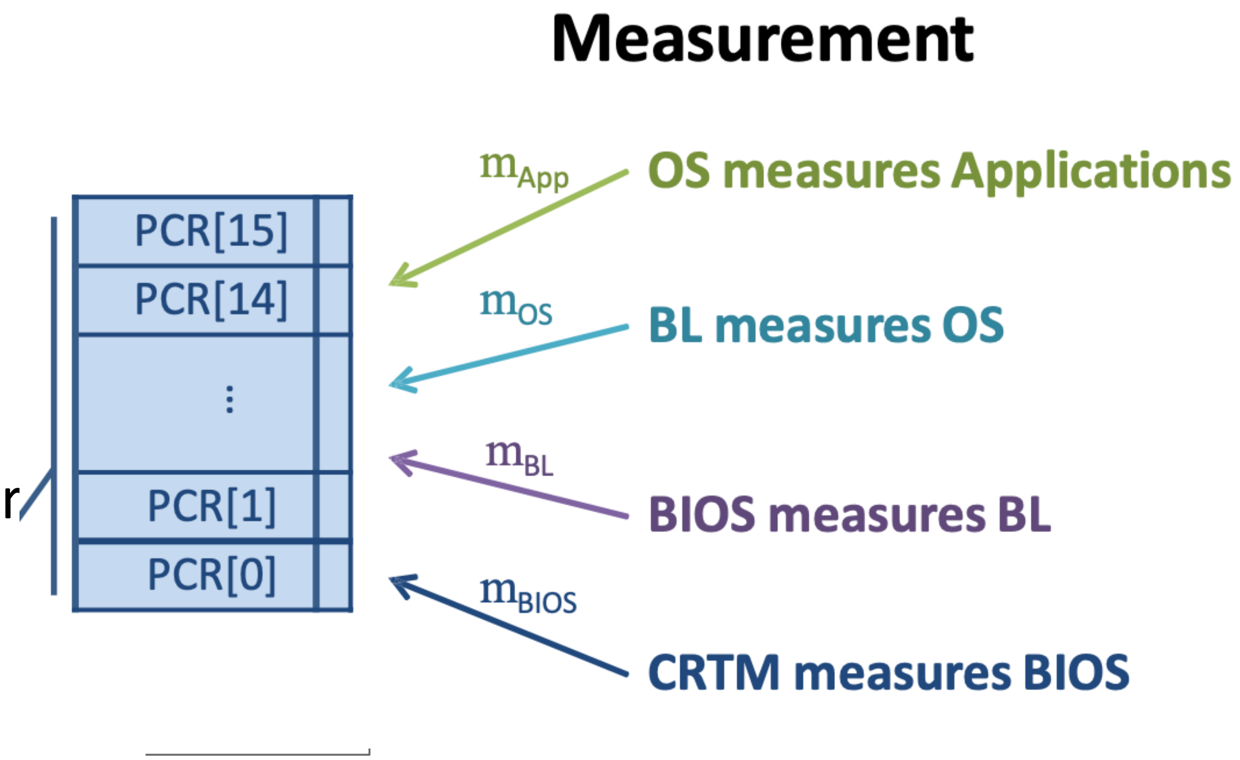
Hardware Security

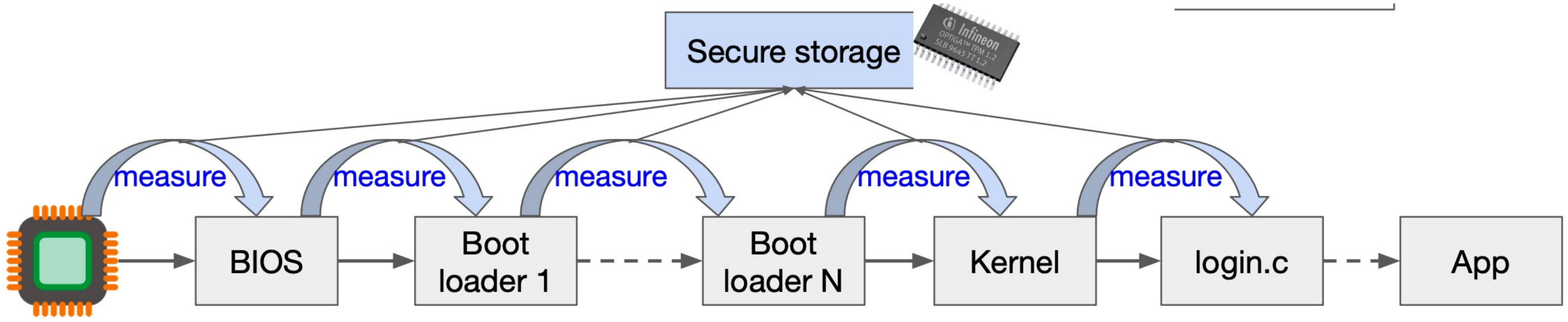
- **Detect malicious OS**
  - You need to trust something
  - Hardware is **root of trust**
    - Hardware measures BIOS (Core Root of Trust Management - CRTM)
    - Verified BIOS is trusted →
    - Boot Loader 1, verification is trusted →
    - ….Kernel,..,App verification are trusted (**chain of trust**)



Hardware Security

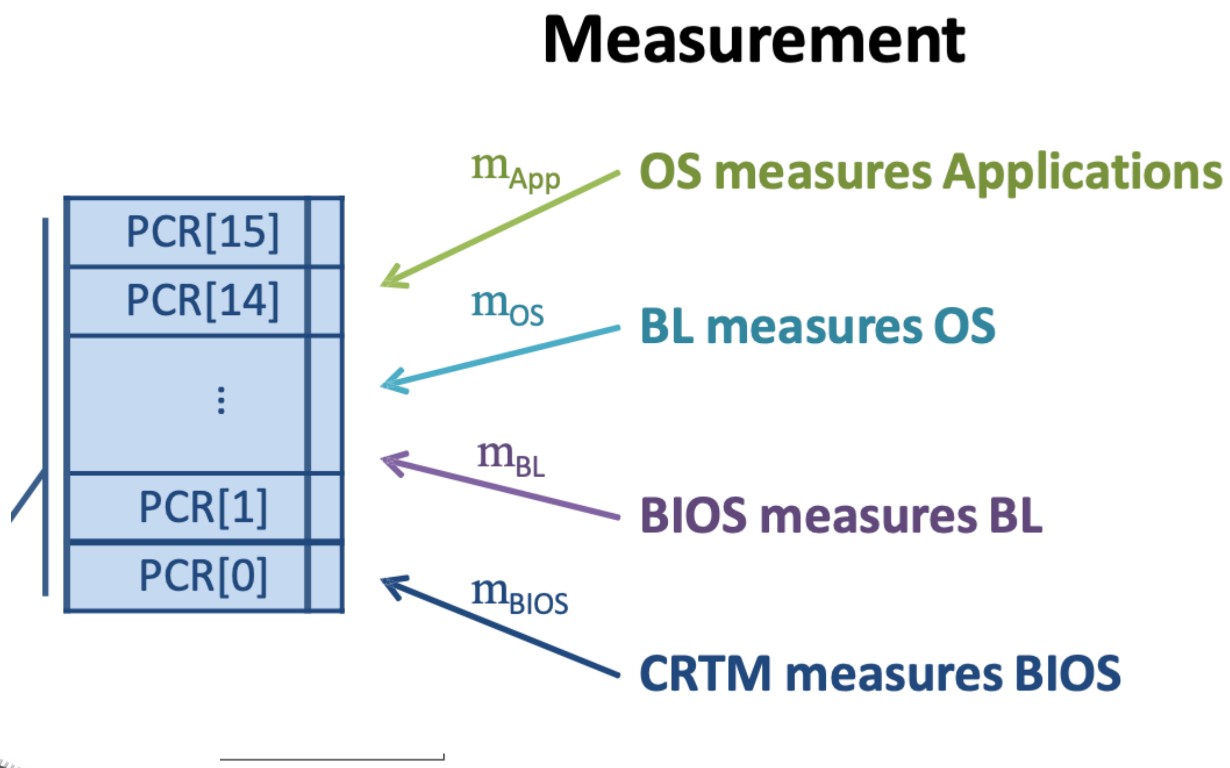
- **Detect malicious OS**
- Trusted Platform Module (TPM)
  - Next to the CPU packaging
  - Tamper Resistant Micro Chip
  - PCR – Platform Configuration Register
  - Vendors implement compatible
    - HW, SW

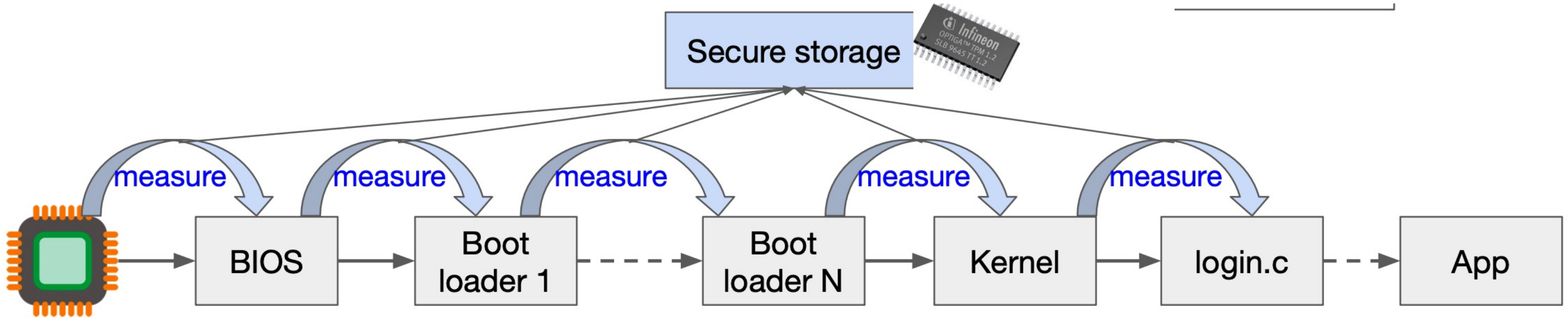




Hardware Security

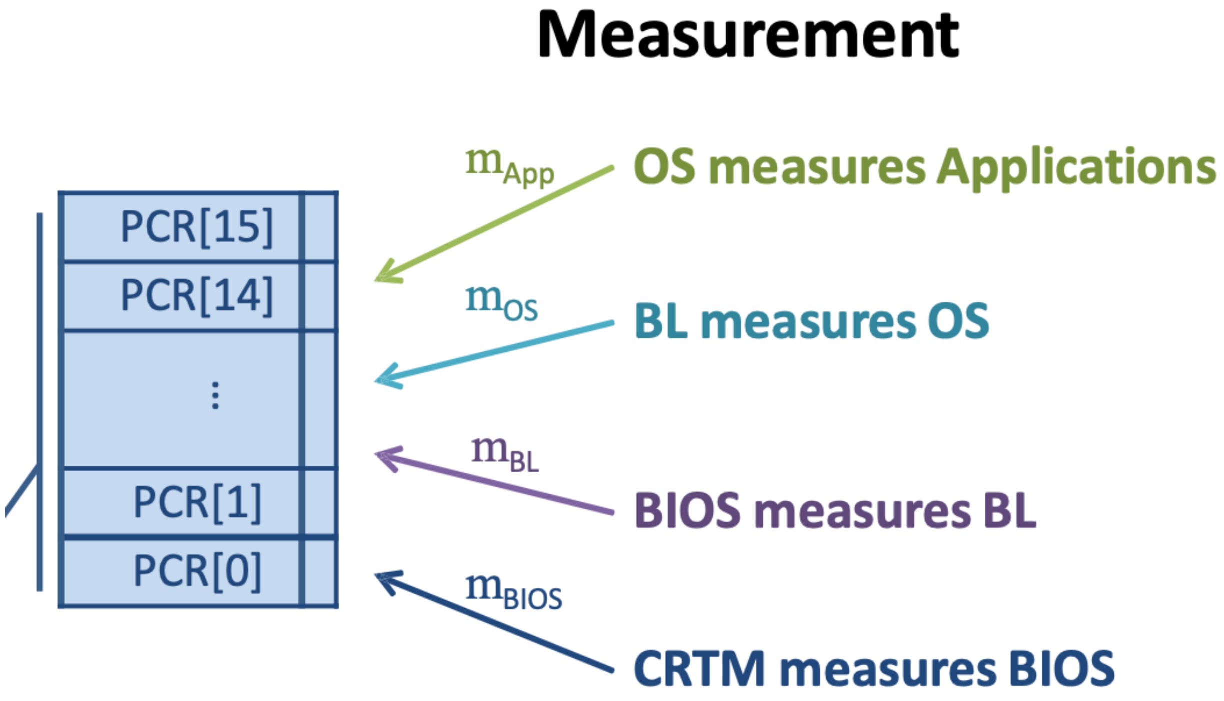
- **Detect malicious OS**
- Trusted Platform Module (TPM)
  - Initializes at OS installation
    - Saving Measurements in TPM
  - Verifies with every reboot
    - With TPM saved values

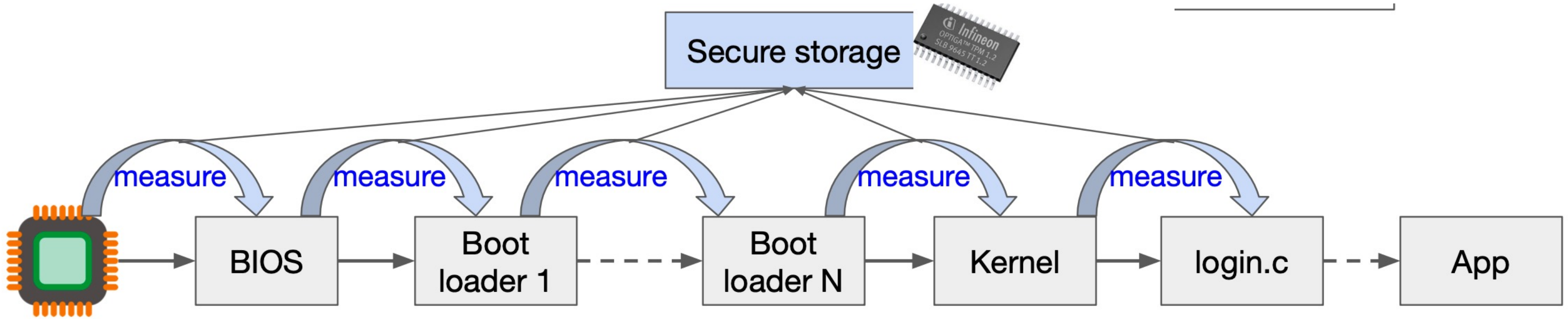




Hardware Security (Initialization)

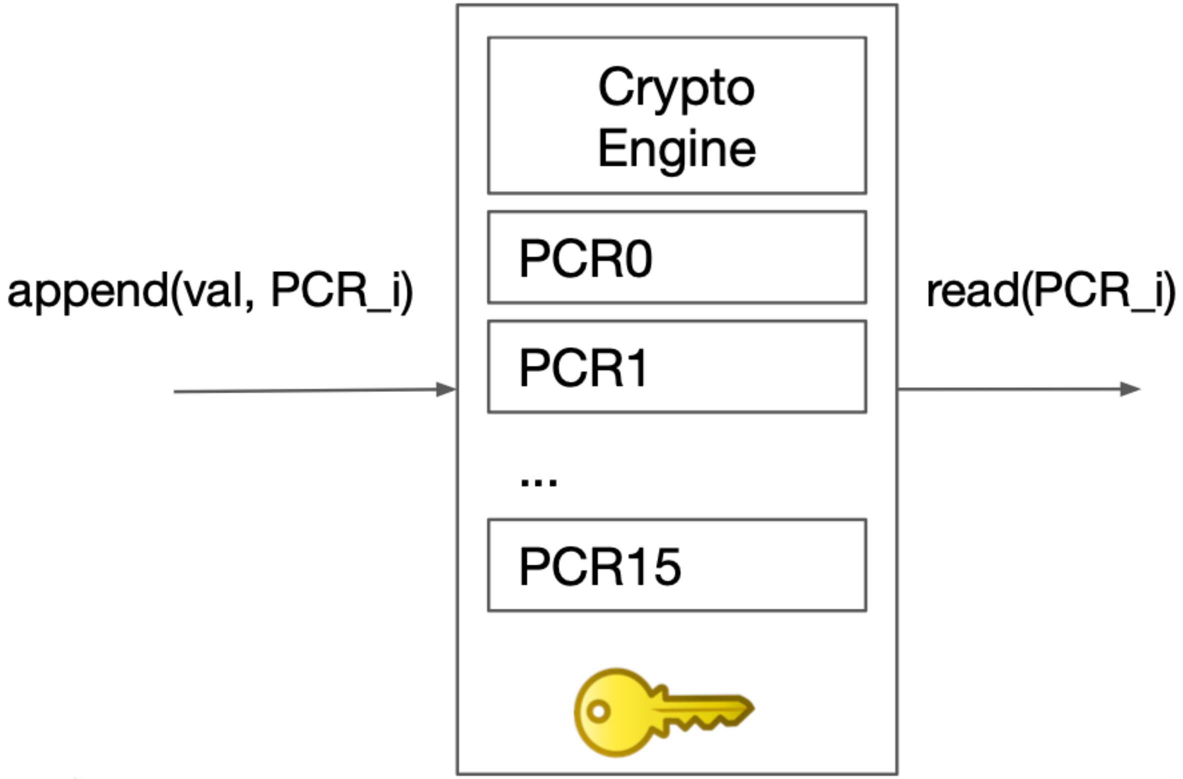
- **Detect malicious OS**
- Trusted Platform Module (TPM)
  - CRTM computes m_{BIOS}= H(BIOS)
  - CPU call TPM.append(m_{BIOS},PCR_0)
    - SET PCR_0 = H(m_{BIOS}||CRTM)

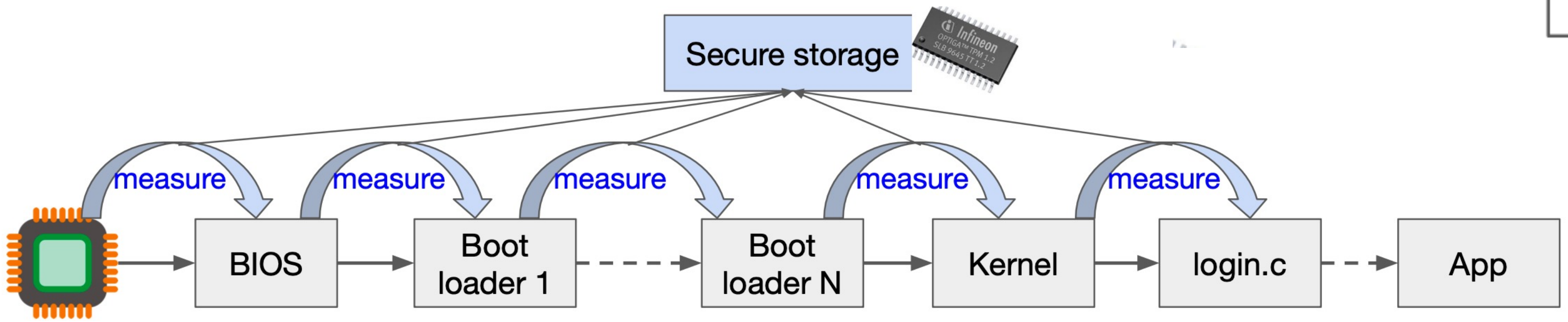




Hardware Security (Initialization)

- **Detect malicious OS**
- Trusted Platform Module (TPM)
  - m_{BL1}= H(BL_1), m_{BL2}= H(BL_2),…
  - TPM.append(m_i,PCR_{i-1}) (In order)
    - SET PCR_1 = H(m_{BL1}||PCR_0)
    - Follow the pattern
  - Why other PCR registers?

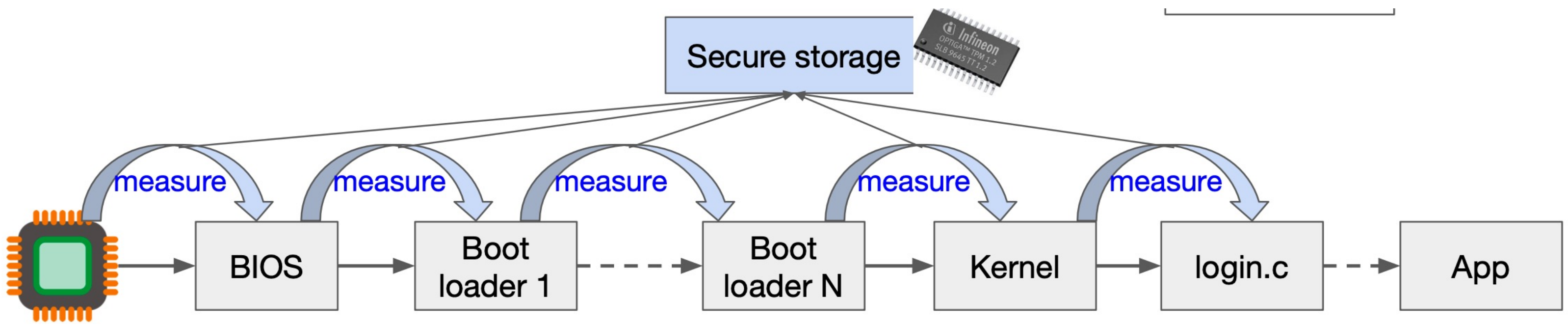




Hardware Security (Verification)

- **Detect malicious OS**
- Trusted Platform Module (TPM)
  - Repeated (read, append, hash) for each boot stage
  - Compare to PCR register values
    - https://community.infineon.com/t5/Blogs/Storing-and-reporting-system-measurements-with-TPM/ba-p/443590#:~:text=PCRs%20are%20registers%20in%20TPM,storing%2020%20bytes%20of%20data.

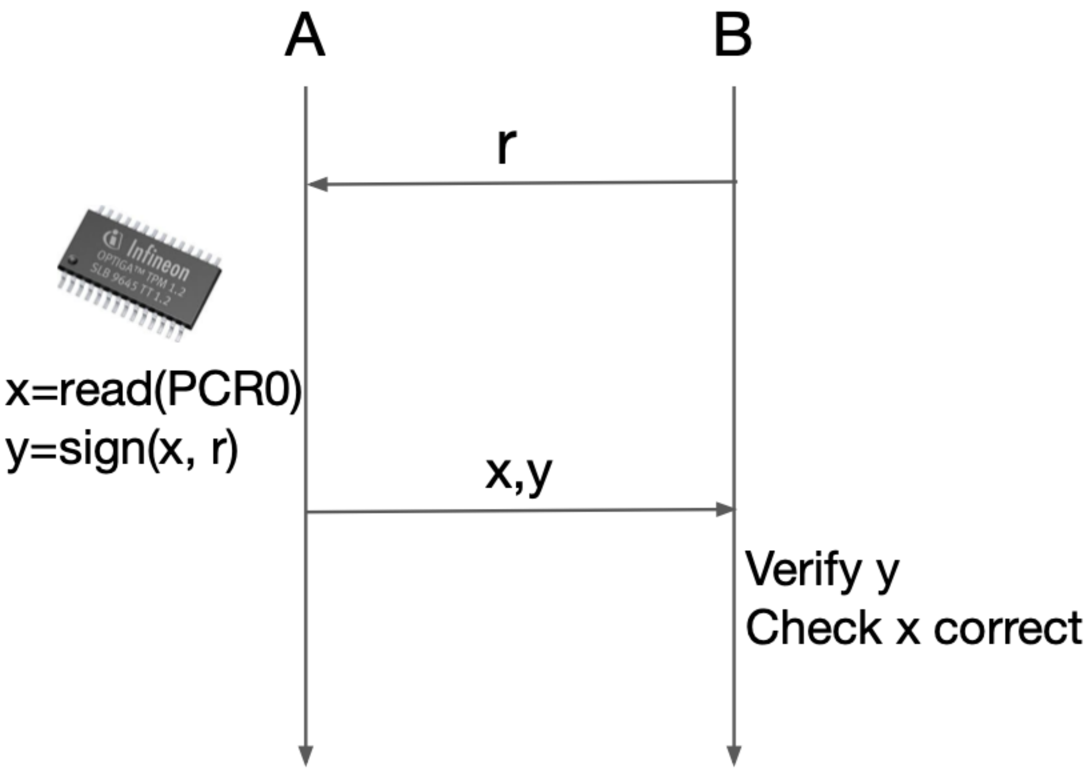
  Why does UEFI prevents dual boot?



Hardware Security

- **Detect malicious OS**
- Trusted Platform Module (TPM)
    - Remote Attestation
    - A wants to convince B
        - That it runs correct OS

    - Assumptions
        - B knows correct hashes
        - B knows TPM’s public keys



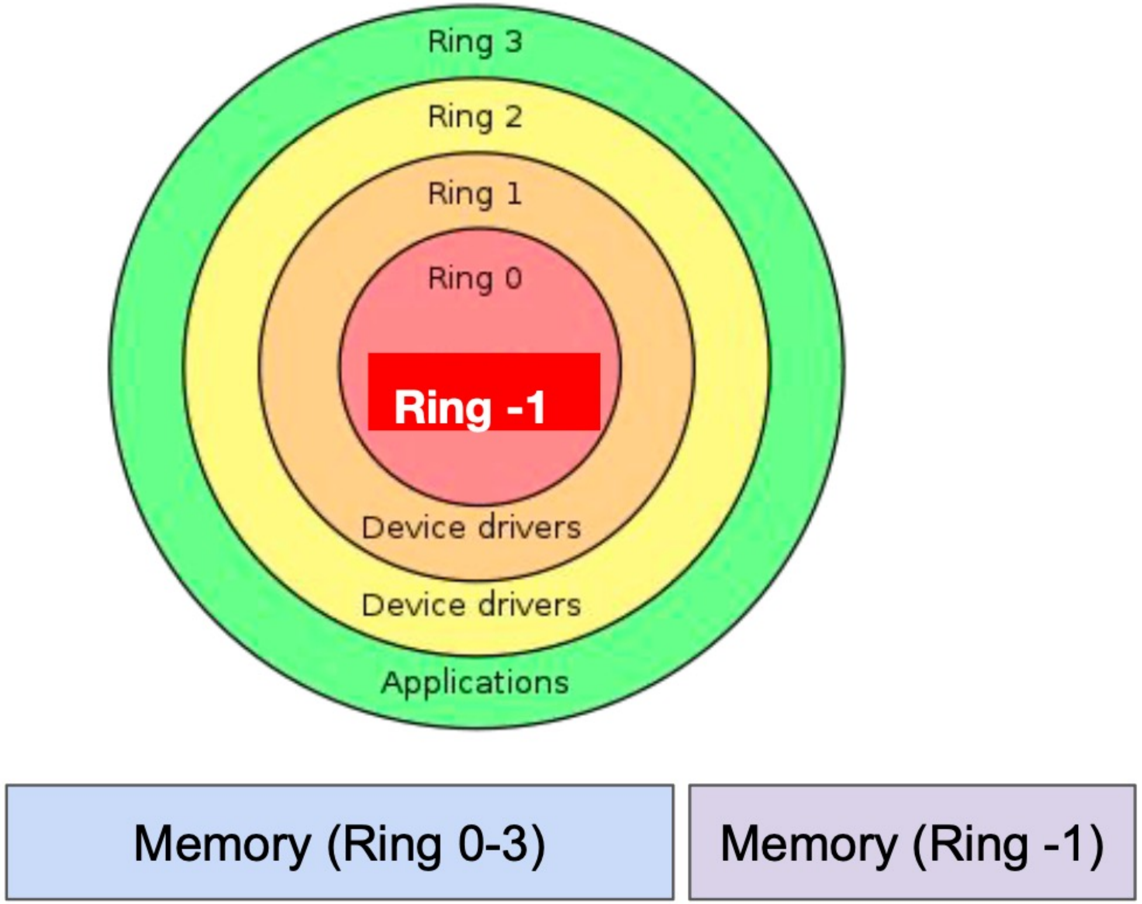
Hardware Security

- **Detect malicious OS**
- Trusted Platform Module (TPM)
  - Advantages
    - Very Cheap
    - Good enough for many embedded systems
    - Introduced hardware security to the main stream
  - Disadvantages
    - Slow
    - Does not guarantee OS is trusted
      - Only that it is not modified
      - Not flexible for OS updates

TRUSTED® COMPUTING GROUP

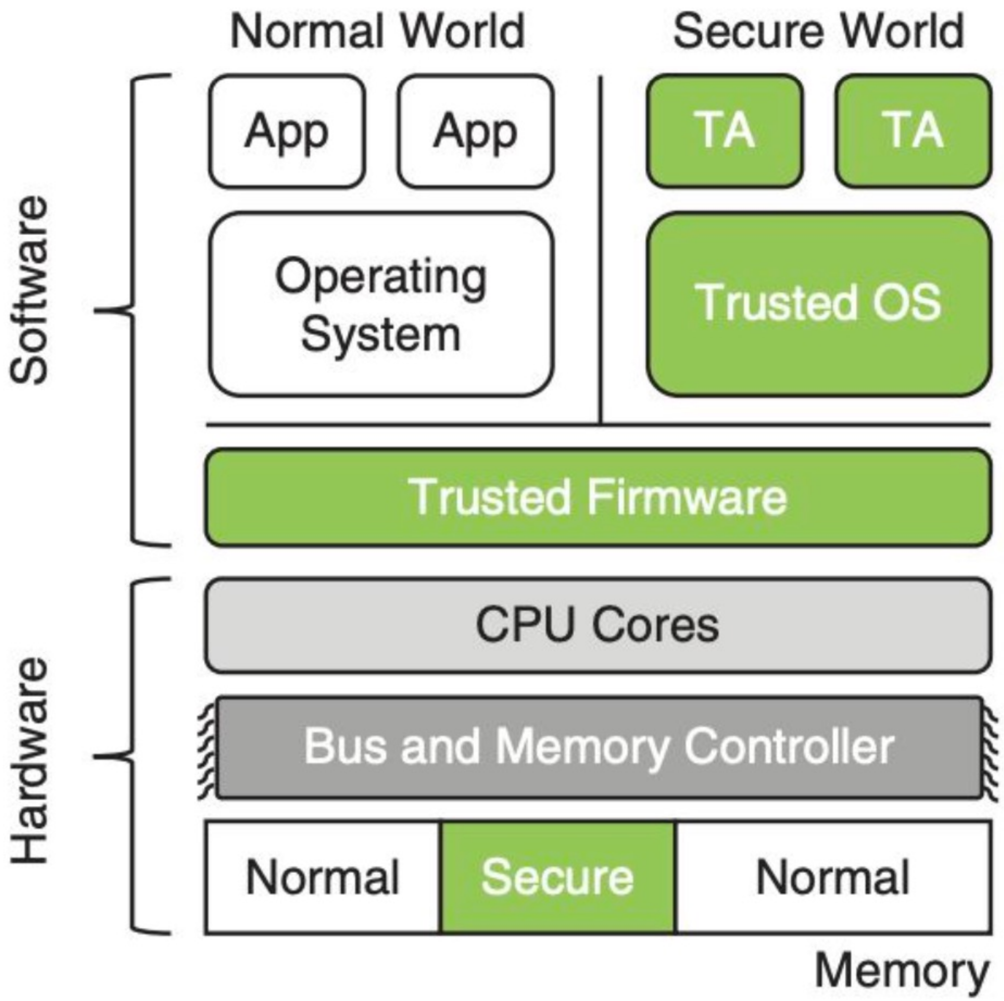
Hardware Security

- **Detect malicious OS**
- Trusted Platform Module (TPM)
  - Run off-chip
  - Limited memory/functions
- We want
  - More powerful/Flexible
  - Extending exiting chips
  - Memory even OS cannot access
    - More privileged than OS
    - Isolated



Hardware Security

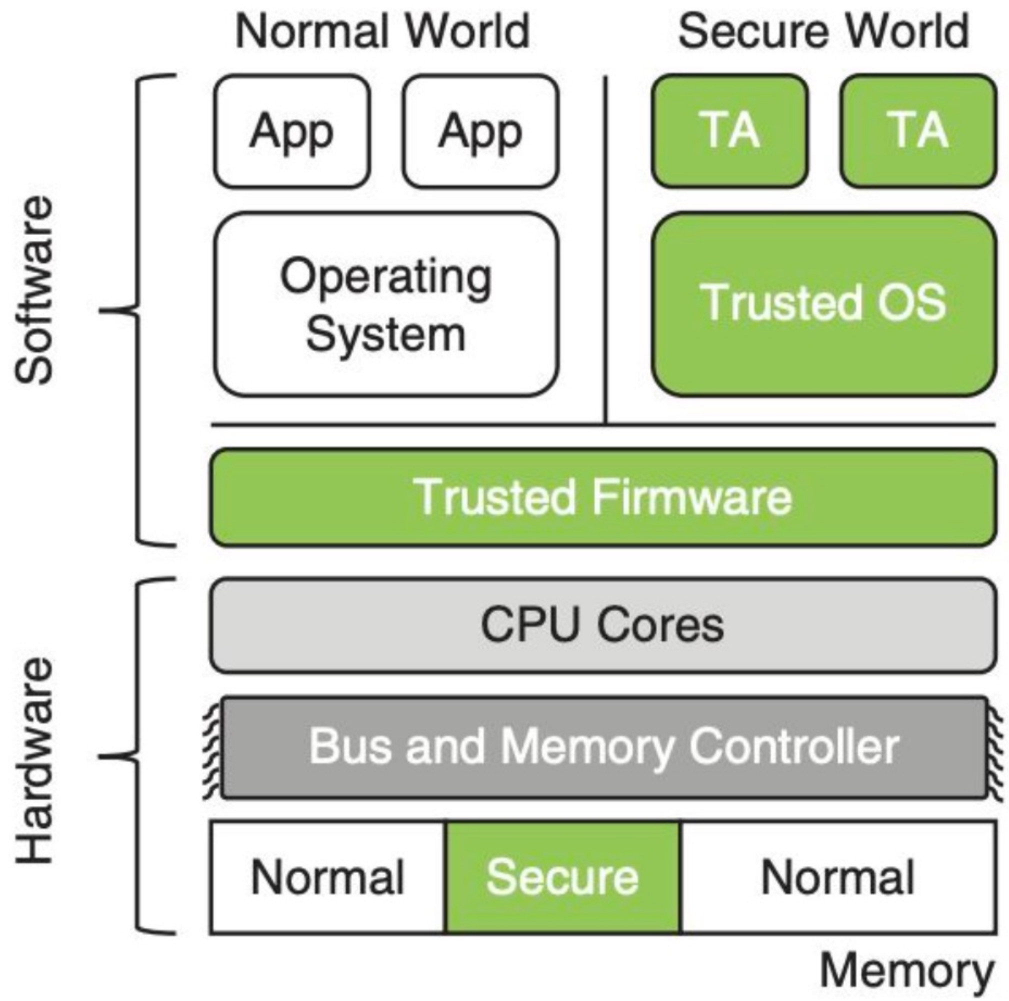
- **Minimize TCB and Isolate secure apps**
- ARM’s TrustZone
  - Application **Isolated** with new privilege mode
    - Non Secure Bit→ 0:Secure, 1:Non-secure
    - Use Secure Memory (Physically Isolated)
    - Enforced by trusted firmware
  - **Minimal TCB**
    - Trusted OS
    - Only security essential features



Hardware Security

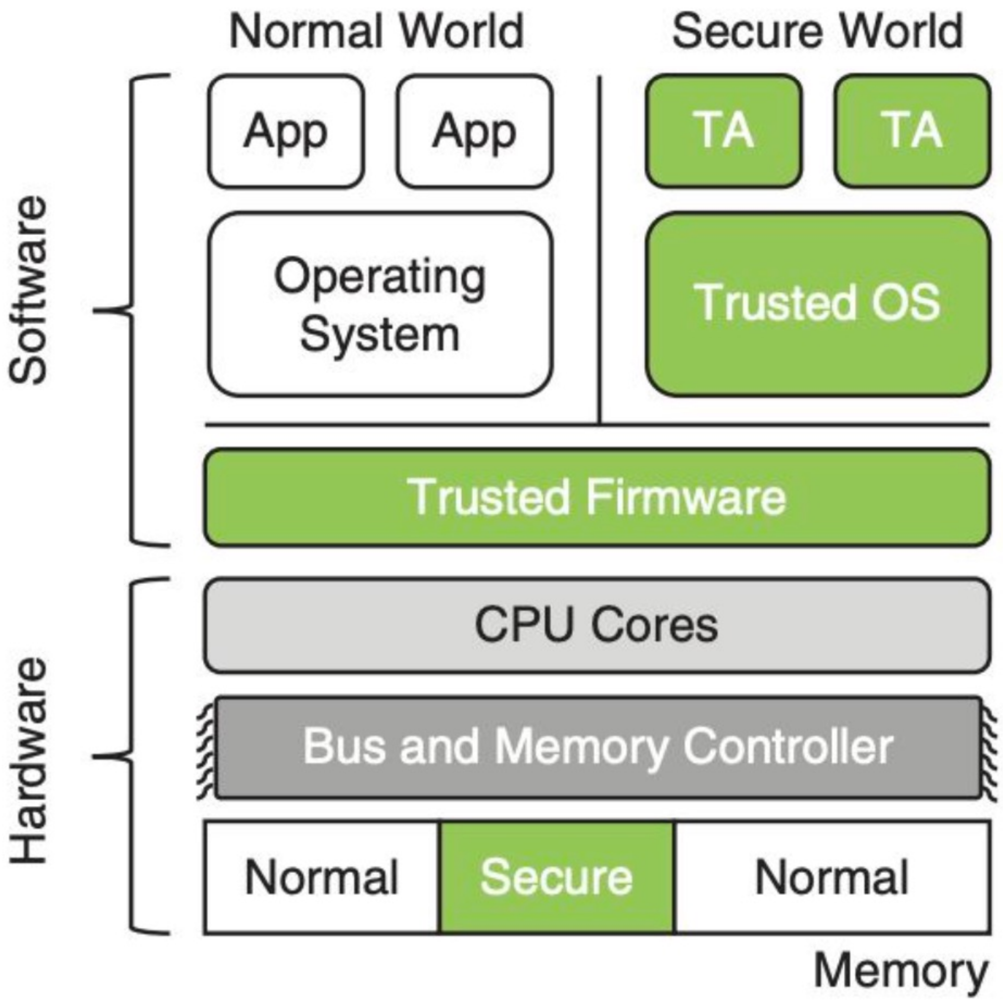
- **Minimize TCB and Isolate secure apps**
- ARM's TrustZone
  - Normal world – Non sensitive app
  - Secure world – Bank app

  - Trusted Firmware
    - Implements isolation (Like Kernel)
    - Set up during boot
    - Run in NS=0 mode (Monitoring Mode)
    - Secure Transition (SMC)



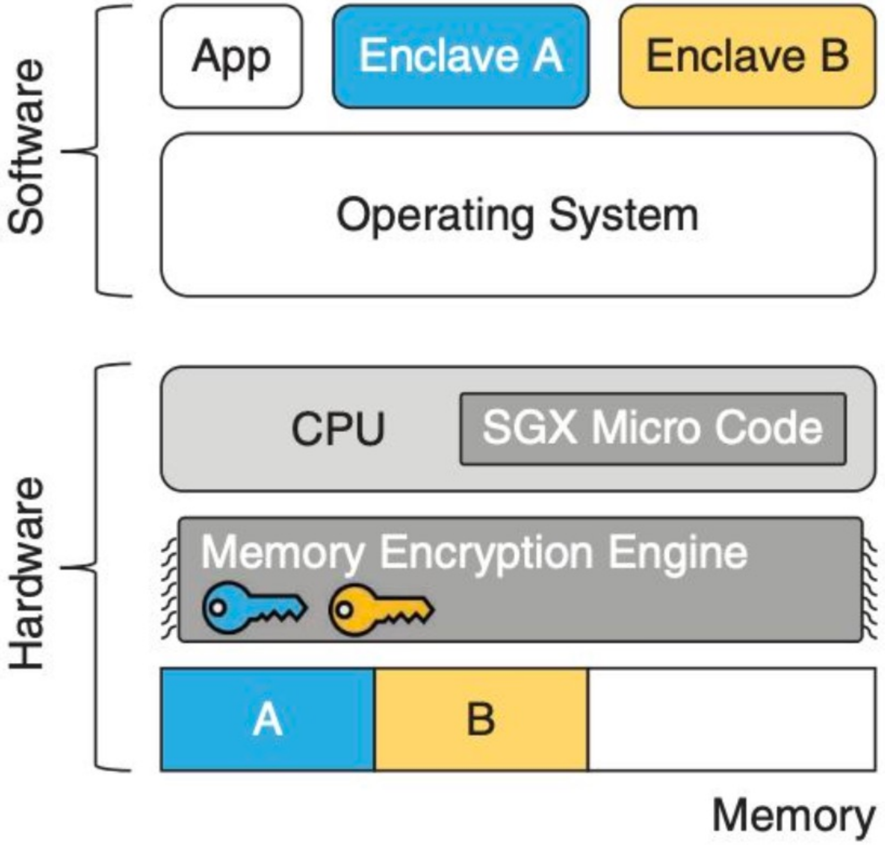
Hardware Security

- **Minimize TCB and Isolate secure apps**
- Application Security
  - Trust verification
    - Authenticated Boot
    - Remote Attestation
  - TCB reduced
    - Trusted OS/Firmware
    - Still big
  - Physical memory attacks?
    - Secure memory is not encrypted



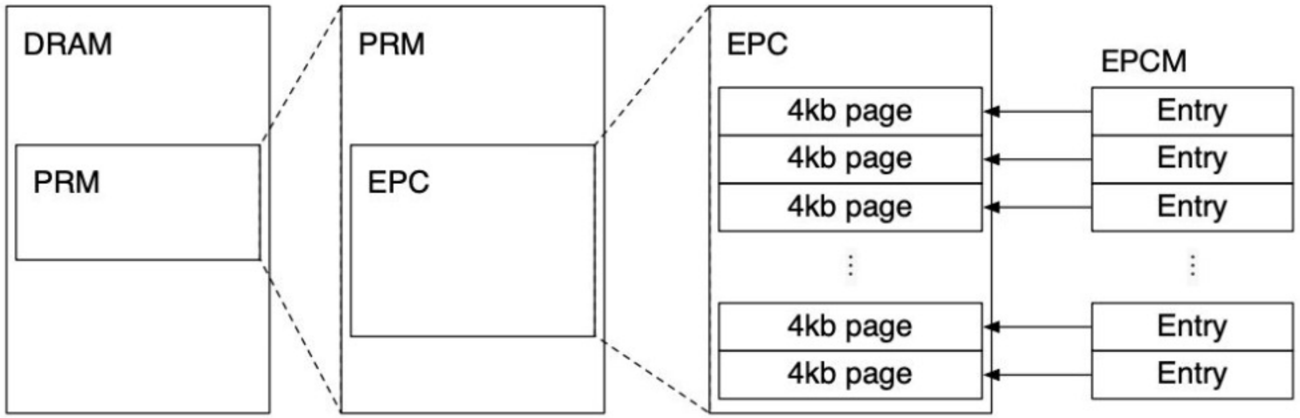
Hardware Security

- **Minimize TCB and Isolate secure apps**
- Intel's Secure Guard Extension (SGX)
    - Application **Isolated**
        - Enclave mode vs Non-enclave mode
        - Enclave mode is highest privilege
        - Processor Reserved Memory (PRM)
            - Encrypted
        - Enforced by SGX Micro Code
    - **Minimal TCB**
        - SGX Micro Code
        - Memory Encryption Engine
        - CPU does secure memory management



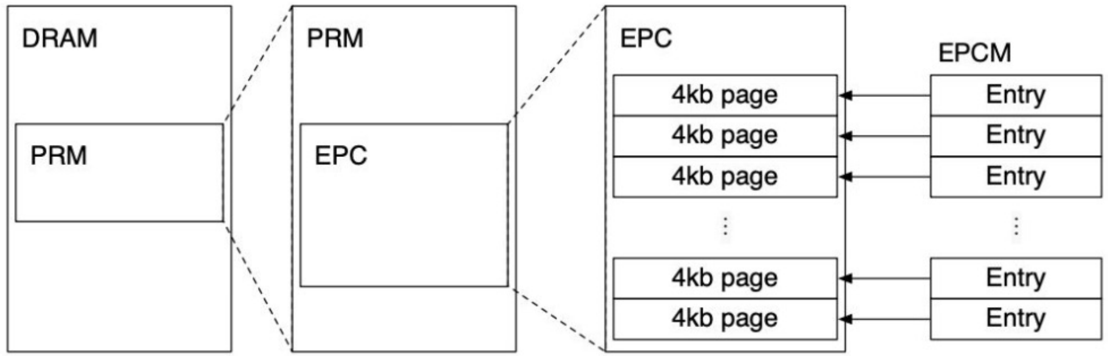
Hardware Security

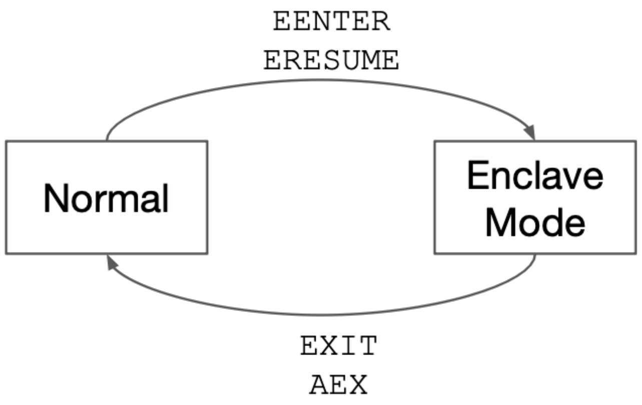
- **Minimize TCB and Isolate secure apps**
- Intel's Secure Guard Extension (SGX)
    - Memory Isolation
        - PRM dedicated for enclaves
        - Enclave Page Cache (EPC)
            - Allocatable pages
            - Allocated pages **encrypted**
        - Enclave Page Cache Map (Page Table)
            - One Entry for each page
            - **CPU manages** (not OS)
                - Only CPU can see the mem. layout
                - Only CPU can edit page table



Hardware Security

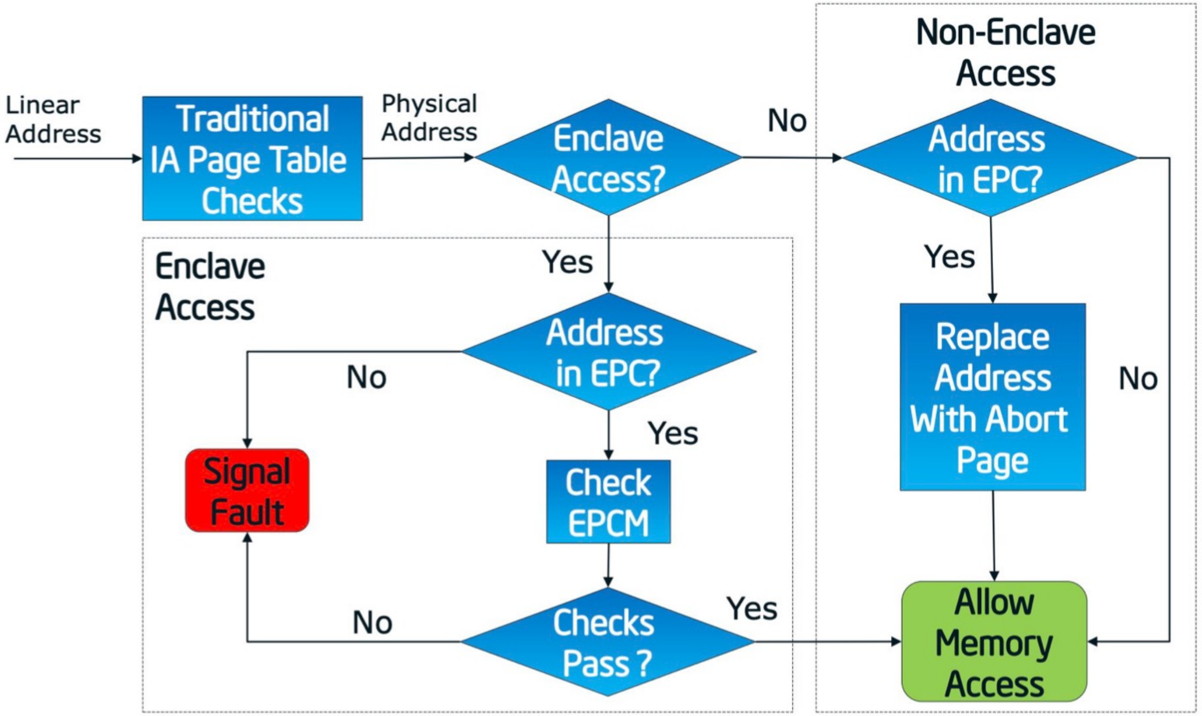
- **Minimize TCB and Isolate secure apps**
- Intel's Secure Guard Extension (SGX)
  - Memory Isolation
    - Role of OS
      - Keep track of EPC/normal spaces
      - Normal process → Normal page
      - Enclave process → EPC
        - EADD instruction
        - CPU update the EPCM
    - Transition between modes
      - EENTER, ERESUME
      - EXIT, AEX





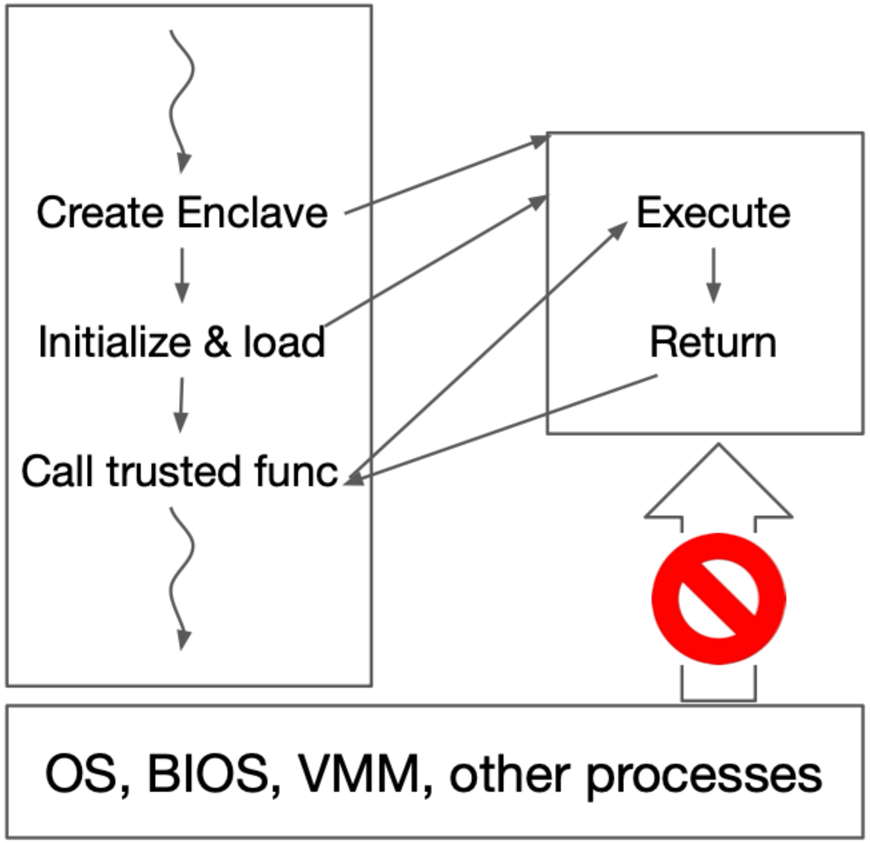
Hardware Security

- **Minimize TCB and Isolate secure apps**
- Intel's Secure Guard Extension (SGX)
  - Memory Translation
    - MMU translation
    - Based on page tables
      - Non enclave page table – OS managed
      - Enclave page table – CPU managed
    - Faults
      - Signal fault – CPU error
      - Abort page – OS error
        - OS never sees EPC layout



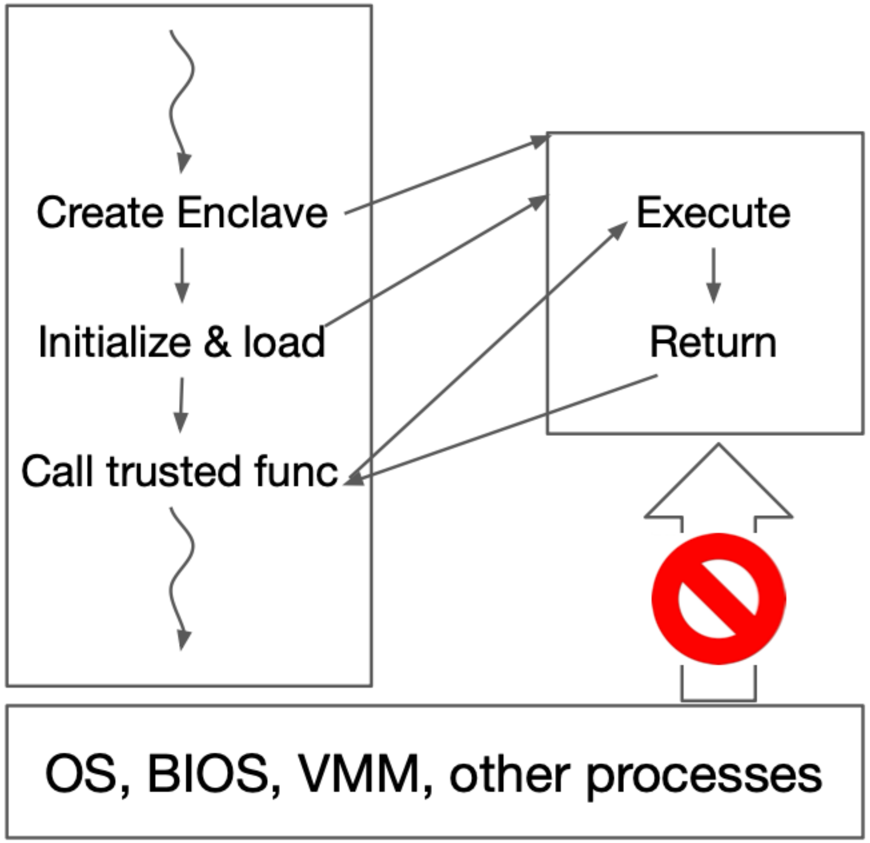
Hardware Security

- **Minimize TCB and Isolate secure apps**
- Intel's Secure Guard Extension (SGX)
  - Programming Model
    - Application has two parts
      - Trusted (Enclave)
      - Non-trusted
    - Create, Init, Run, Exit
    - Instructions
      - Can OS read from enclave?
        - MOV <addr>, EAX
        - No.
      - Can Enclave access untrusted memory?
        - Not directly



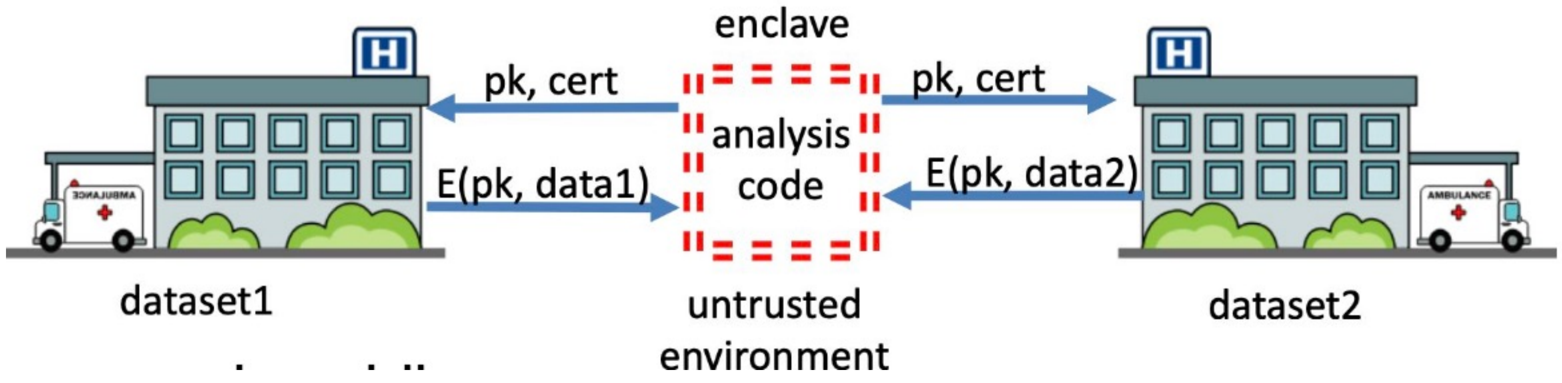
Hardware Security

- **Minimize TCB and Isolate secure apps**
- Intel's Secure Guard Extension (SGX)
  - Programming Model
    - Multi Threading
      - Context Switch from Enclave to OS
      - CPU cleans up
  - Attestation
    - CPU measures Enclave (Code and data)
    - Signs
    - Remote party verifies



Hardware Security

- **Minimize TCB and Isolate secure apps**
- Intel's Secure Guard Extension (SGX)
  - Example
    - Hospital encrypts a dataset (Source Code Public)
    - Runs analysis on a server (SGX supported)
    - Communicate using public key
      - Private key stored in Memory encryption engine



Hardware Security

- **Minimize TCB and Isolate secure apps**
- Intel’s Secure Guard Extension (SGX)
  - Advantages
    - Small TCB
    - Commercially Available
  - Disadvantages
    - Side channel attacks
    - Physical attacks?
      - Encrypted
      - Can someone physically access memory encryption engine?

Hardware Security

Summary:

- OS and HW, TCB
- How to Trust OS and HW

- Verify Known Firmware and OS (TPM by Trusted Computing Group)
- Extended with hardware enforced isolation (ARM Trustzone)
  - Trusted OS/Firmware
- Extended with memory encryption (Intel SGX)
  - Trusted CPU/Firmware