CSE 543: Computer Security
Module: Hardware Security

Prof. Syed Rafiul Hussain
Department of Computer Science and Engineering
Pennsylvania State University

What is Trust?

- dictionary.com
  - Firm reliance on the integrity, ability, or character of a person or thing.
- What do you trust?
  - Trust Exercise
- Do we trust our computers?

Trust

- “a system that you are forced to trust because you have no choice” -- US DoD

- “A ‘trusted’ computer does not mean a computer is trustworthy” -- B. Schneier

Trusted Computing Base

- Trusted Computing Base (TCB)
  - Hardware, Firmware, Operating System, etc
- There is always a level at which we must rely on trust

Trusted Computing Base

PennState

- Helps us enforce security
  - E.g., reference monitor in OS for access control
- Historically, security features have been added to OSes or into programs directly
  - But, may be slow and/or complex enforce security
- How about adding security features into the hardware?
  - May still need support from the OS/compilers
  - But maybe we don’t have to trust them…

Buffer Overflows

- Can hardware help prevent buffer overflows from being exploited?
  - How could it help?

Buffer Overflows - MPX

- Can hardware help prevent buffer overflows from being exploited?
  - How could it help?
- One Approach: Intel MPX
  - Instruction set architecture (ISA) extension
  - Set bounds registers - update these from a bounds table
  - Check bounds - check bounds for a pointer
  - Set status - store error code to enable error handling
- Approach
  - Store upper and lower bound addresses in bounds register
  - Use selected bounds register with a pointer use
  - Pointer must be within bounds

Buffer Overflows - MPX

- Of course, somebody needs to setup the bounds information and decide when to check the pointers
  - And deal with violations when they occur
- Operating systems
  - Provides support for memory management for bounds table and exception handling on violation
- Compilers
  - Instruments the original program to track and check bounds
- Runtime libraries
  - Initialize MPX and check bounds before library calls
- Ecosystem for Intel MPX is now available although researchers are just starting to evaluate

Another Use for MPX

- Paper "*LMP: Light-Weighted Memory Protection with Hardware Assistance*" in ACSAC 2016 used MPX for implementing a shadow stack
- A shadow stack compares return values on stack with expected return values
  - LMP implements such checks by
    - On Call: Copy expected return address to shadow stack
    - On Return: Load expected return address into bounds register and compare to actual return address
  - To protect the shadow stacks, all stores except those in instrumentation are prohibited from accessing shadow stack memory by bounds checks

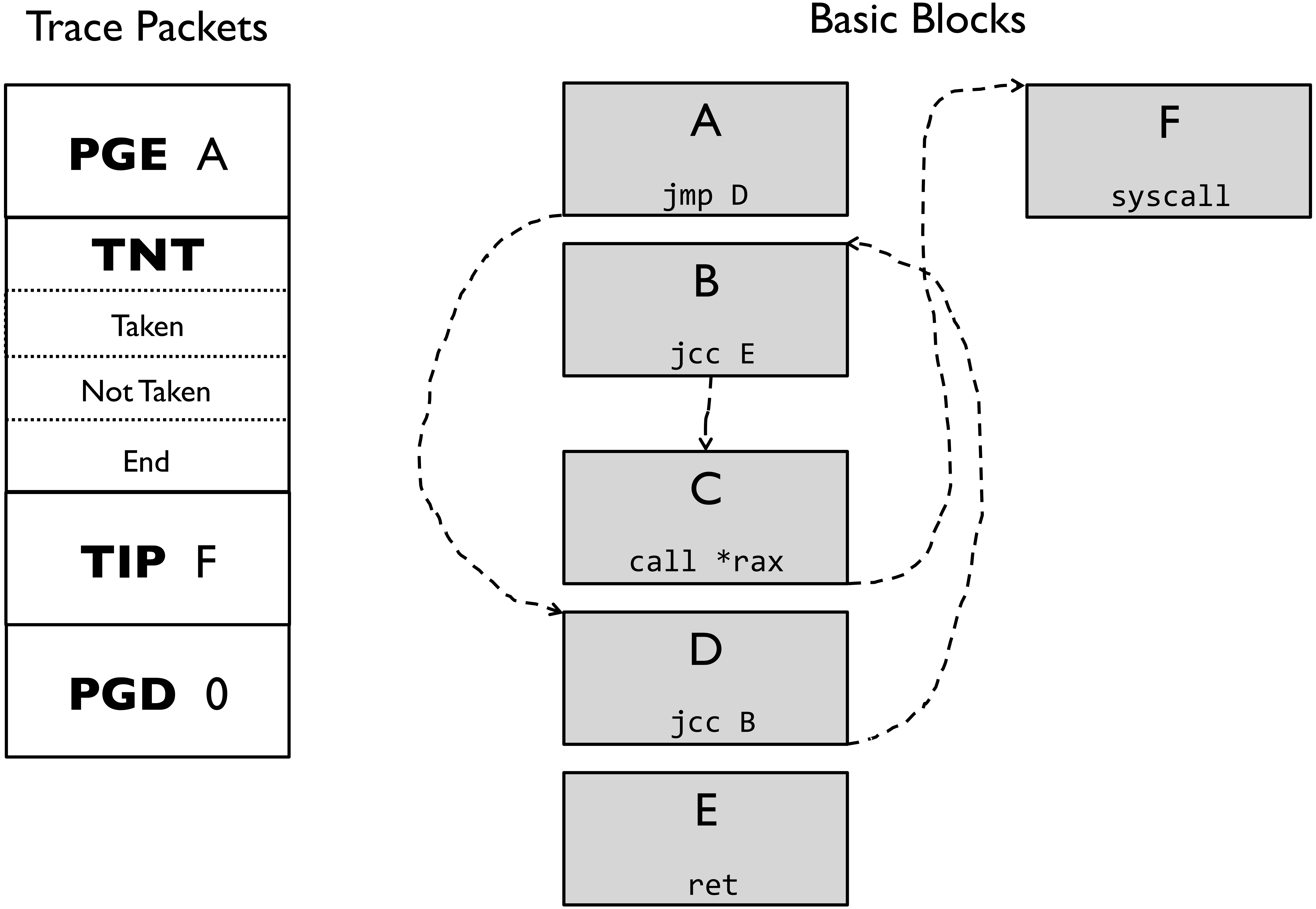
Control Flow Hijacking

- Can hardware help prevent control flow hijacking using function pointers (call/jmp) and returns?
    - How could it help?

Control Flow Hijacking - PT

- Can hardware help prevent buffer overflows from being exploited?
  - How could it help?
- One Approach: Intel PT
  - Record the control flow decisions made by a program at runtime in a trace buffer
  - Use the trace buffer to evaluate the program control flow to detect errors
- Use for control-flow integrity enforcement
  - Record trace buffers from execution
  - Compare indirect call/jmp targets to expected targets
  - Collect call sites and match returns to expected returns

An Example


Trace Packets
PGE A
TNT
Taken
Not Taken
End
TIP F
PGD 0
Basic Blocks
A
jmp D
B
jcc E
C
call *rax
D
jcc B
E
ret
F
syscall

Control Flow Hijacking - PT

- Coarse-grained Policy (any legal target for source)
  - Check if the targets of indirect control transfers are valid
  - Requires decoding the trace packets
- Fine-grained Policy (specific targets for source)
  - Check if the source and destination are a legitimate pair
  - Requires control-flow recovery
- Shadow Stack
  - Check if an indirect control transfer is legitimate based on the reconstructed call stack for entire run
  - Requires sequential processing

Untrusted OS?

- Can hardware help protect your programs from compromised operating systems?
  - Do you really need to trust the OS?

Untrusted OS?

- Can hardware help protect your programs from compromised operating systems?
  - Do you really need to trust the OS?

- What do you need to do to protect your process from the OS?

Untrusted OS?

- Can hardware help protect your programs from compromised operating systems?
  - Do you really need to trust the OS?

- What do you need to do to protect your process from the OS? Use OS services safely
  - Memory management
  - Device access
  - Scheduling (availability)
- Ideally, protect secrecy and integrity of application data when using memory and device resources

Intel SGX

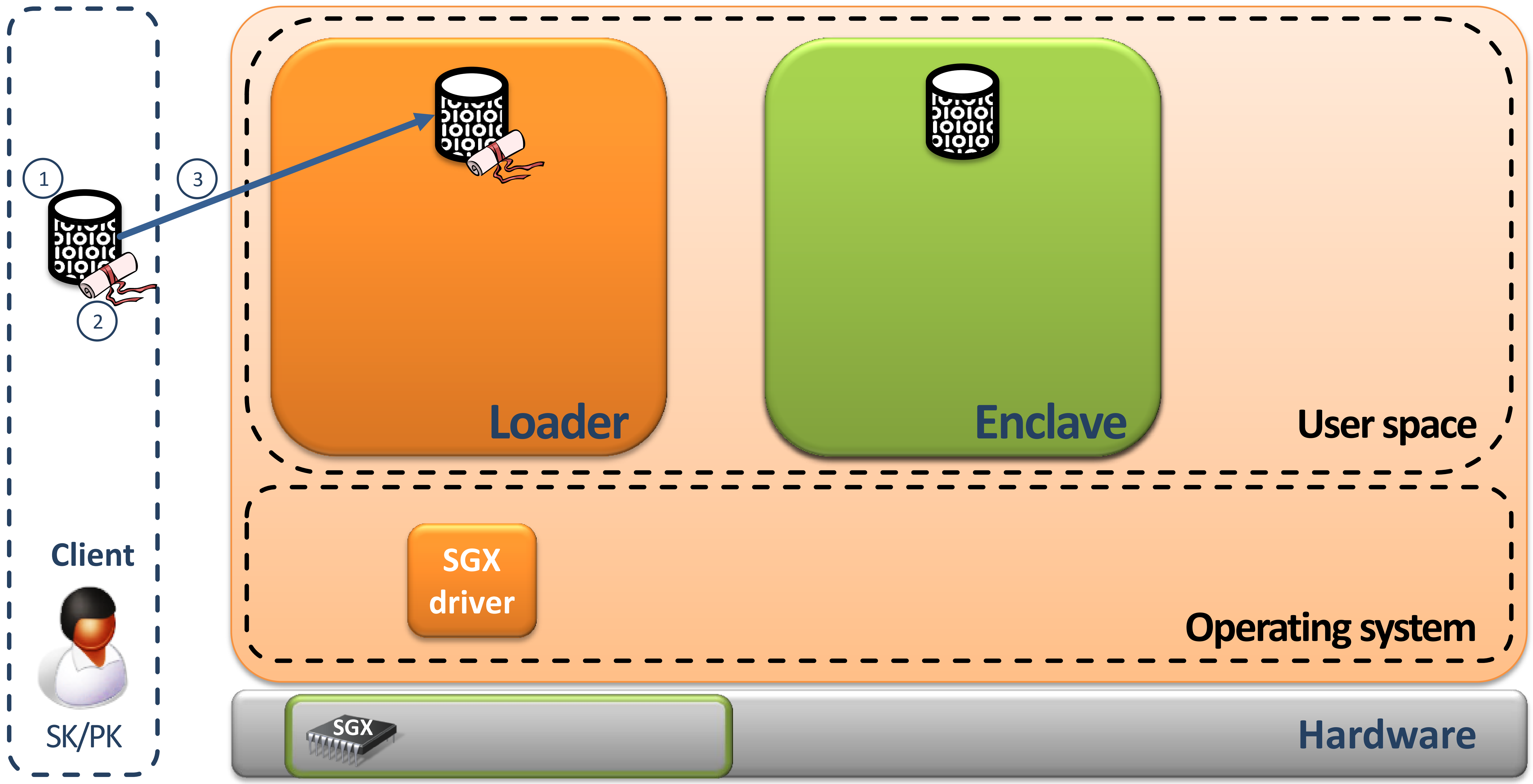
- Can hardware help protect your programs from compromised operating systems?
  - Do you really need to trust the OS?
- One Approach: Intel SGX
  - Define a protected memory "enclave" to run programs
  - Load and run your programs in that enclave
  - Use OS as a untrusted server of resources (encrypted memory and system resources)
- For a program that processes secret data
  - Load program and keys into enclave
  - Read encrypted data from system
  - Decrypt and process that data

SGX Enclaves

- **Enclaves are isolated memory regions of code and data**
- **One part of physical memory (RAM) is reserved for enclaves**
  - It is called **Enclave Page Cache (EPC)**
  - EPC memory is encrypted in the main memory (RAM)
  - Trusted hardware consists of the CPU-Die only
  - EPC is managed by OS/VMM

RAM: Random Access Memory
OS: Operating System
VMM: Virtual Machine Monitor (also known as Hypervisor)

SGX – Create Enclave

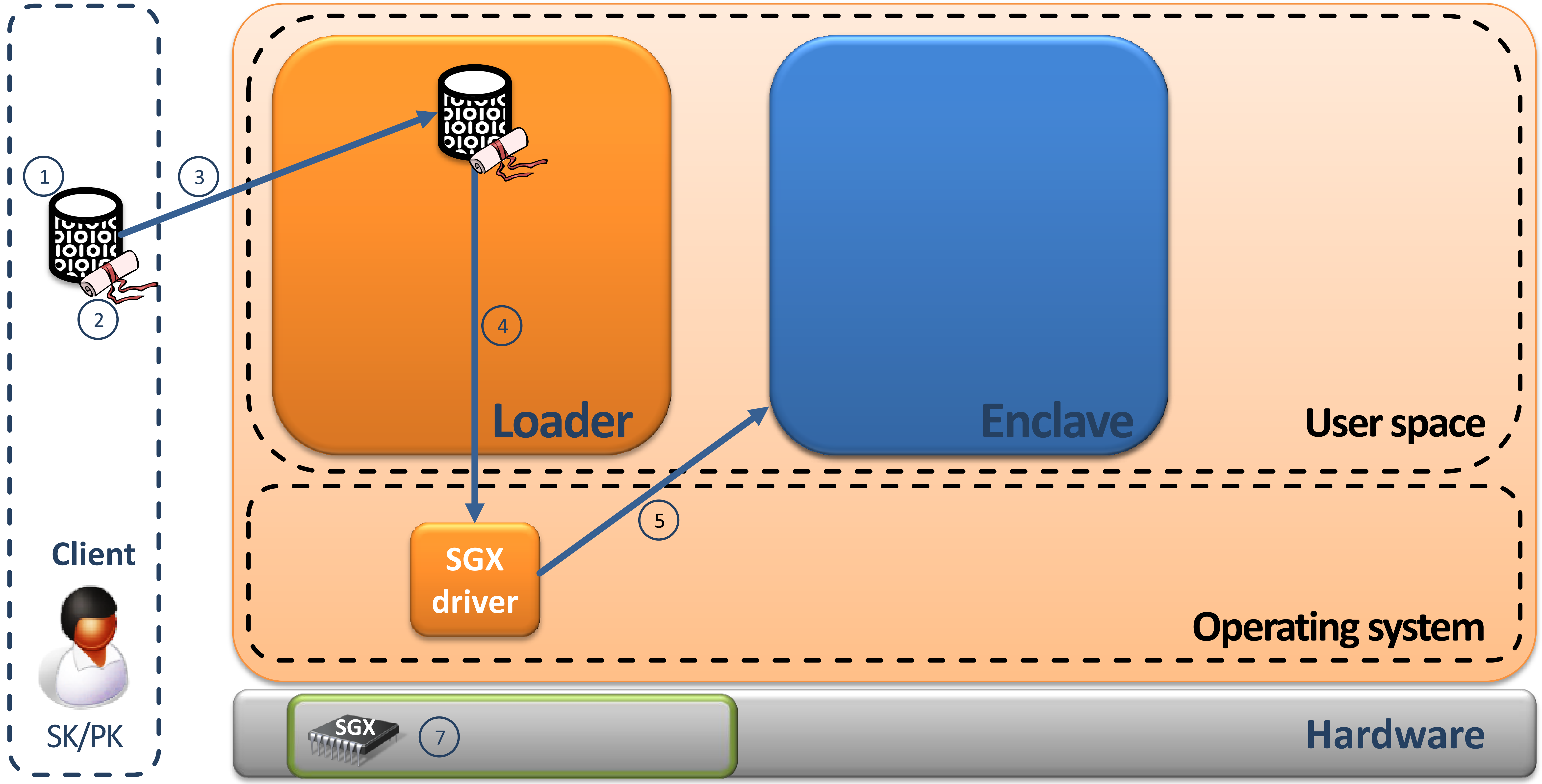


1. Create App
2. Create app certificate (includes HASH(App) and Client PK)
3. Upload App to Loader

Trusted Untrusted

A.-R. Sadeghi ©TU Darmstadt, 2007-2014 Slide Nr. 19, Lecture Embedded System Security, SS 2014 Trusted Execution Environments / Intel SGX

SGX – Create Enclave



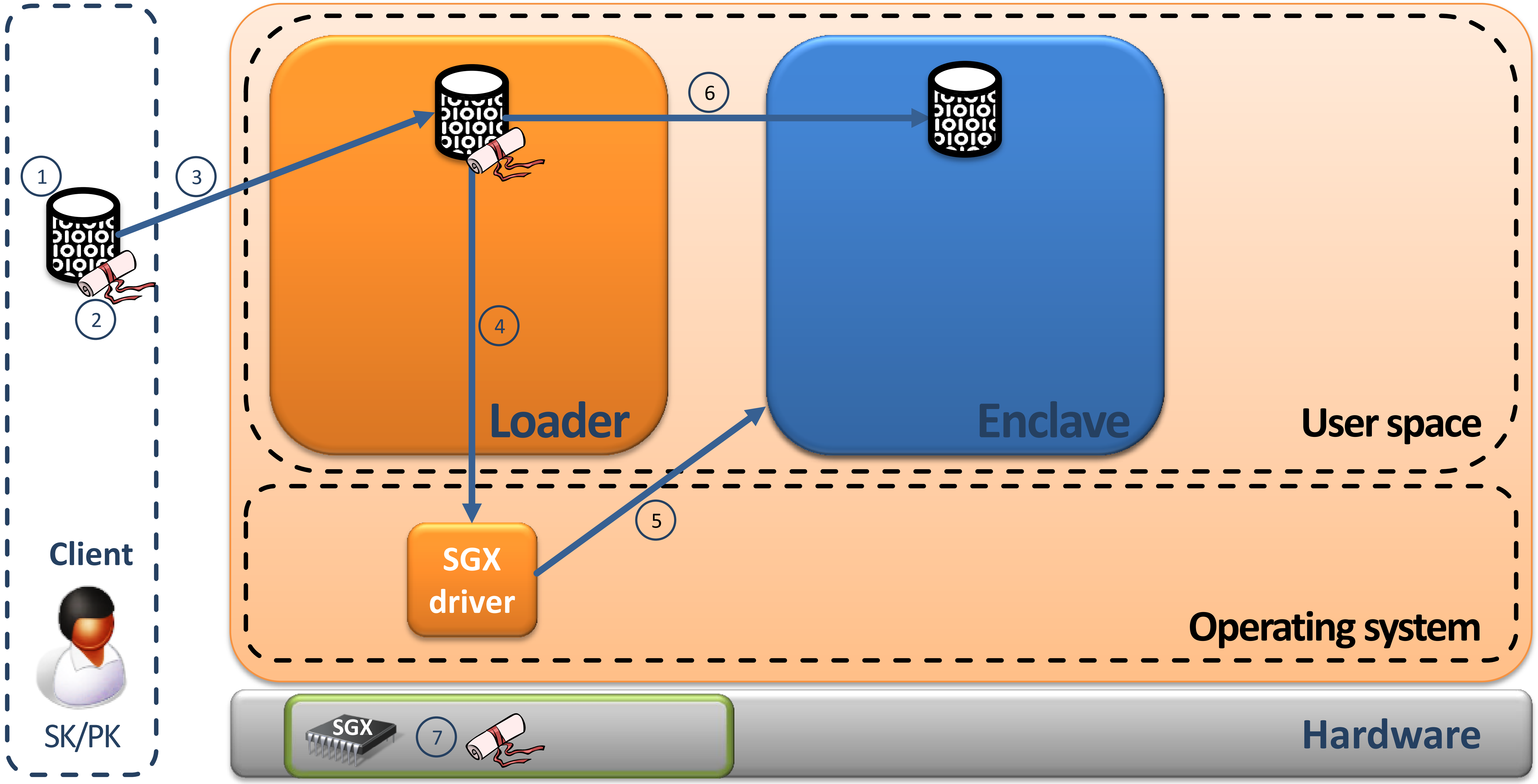
1. Create App
2. Create app certificate (includes HASH(App) and Client PK)
3. Upload App to Loader
4. Create enclave
5. Allocate enclave pages

Trusted | Untrusted

A.-R. Sadeghi ©TU Darmstadt, 2007-2014 Slide Nr. 20, Lecture Embedded System Security, SS 2014 Trusted Execution Environments / Intel SGX

Intel Software Guard Extension

SGX – Create Enclave



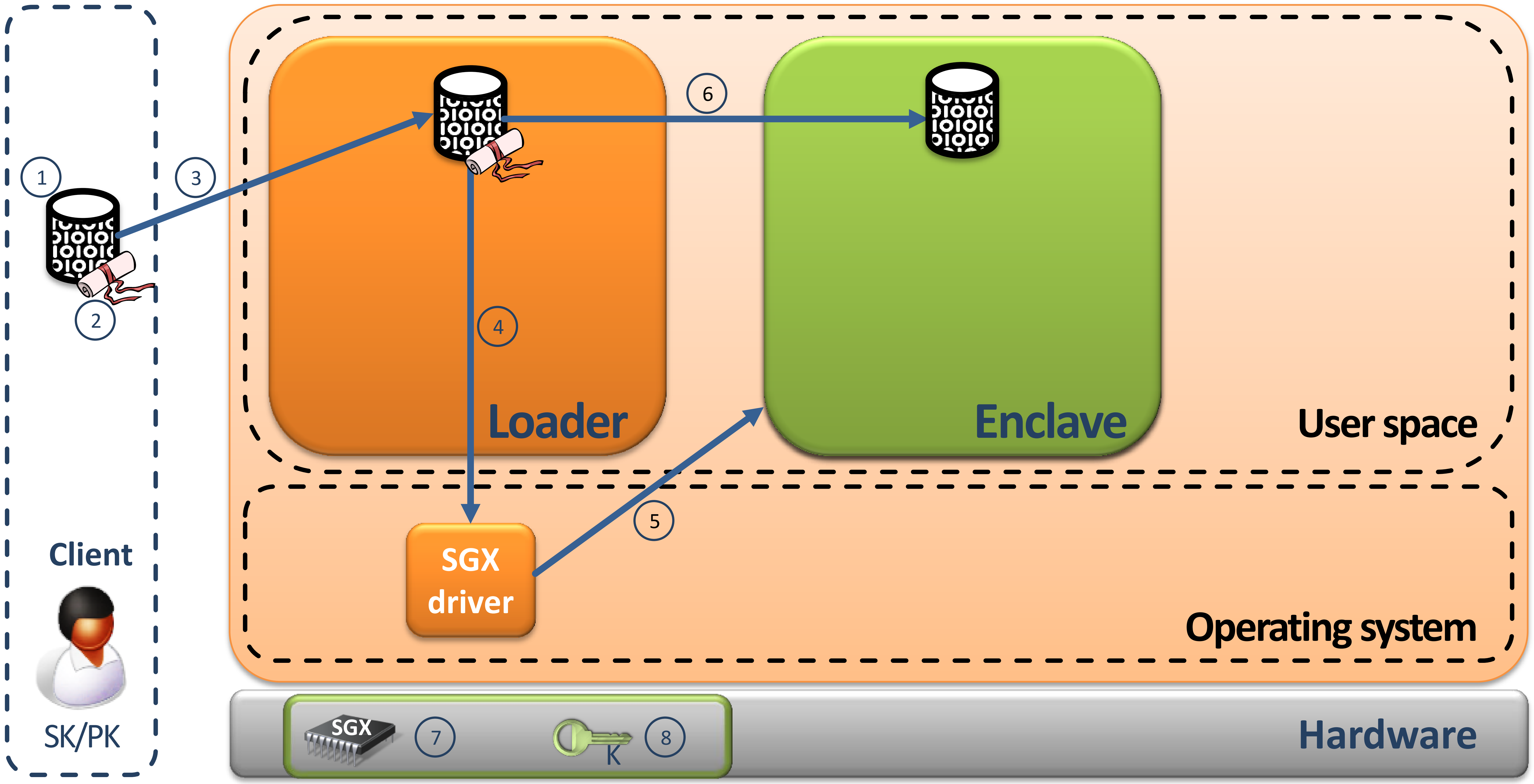
1. Create App 2. Create app certificate (includes HASH(App) and Client PK) 3. Upload App to Loader
4. Create enclave 5. Allocate enclave pages 6. Load & Measure App 7. Validate certificate and enclave integrity

Trusted Untrusted

A.-R. Sadeghi ©TU Darmstadt, 2007-2014 Slide Nr. 21, Lecture Embedded System Security, SS 2014 Trusted Execution Environments / Intel SGX

Intel Software Guard Extension

SGX – Create Enclave



1. Create App
2. Create app certificate (includes HASH(App) and Client PK)
3. Upload App to Loader
4. Create enclave
5. Allocate enclave pages
6. Load & Measure App
7. Validate certificate and enclave integrity
8. Generate enclave **K** key
9. Protect enclave

Trusted
Untrusted

A.-R. Sadeghi ©TU Darmstadt, 2007-2014 Slide Nr. 22, Lecture Embedded System Security, SS 2014 Trusted Execution Environments / Intel SGX

Untrusted OS vs SGX

- Challenges in running an environment that
  - (1) Does not trust the OS
  - (2) Yet uses the OS services
    - Memory management (e.g., page fault handling)
    - System calls
- What could go wrong?

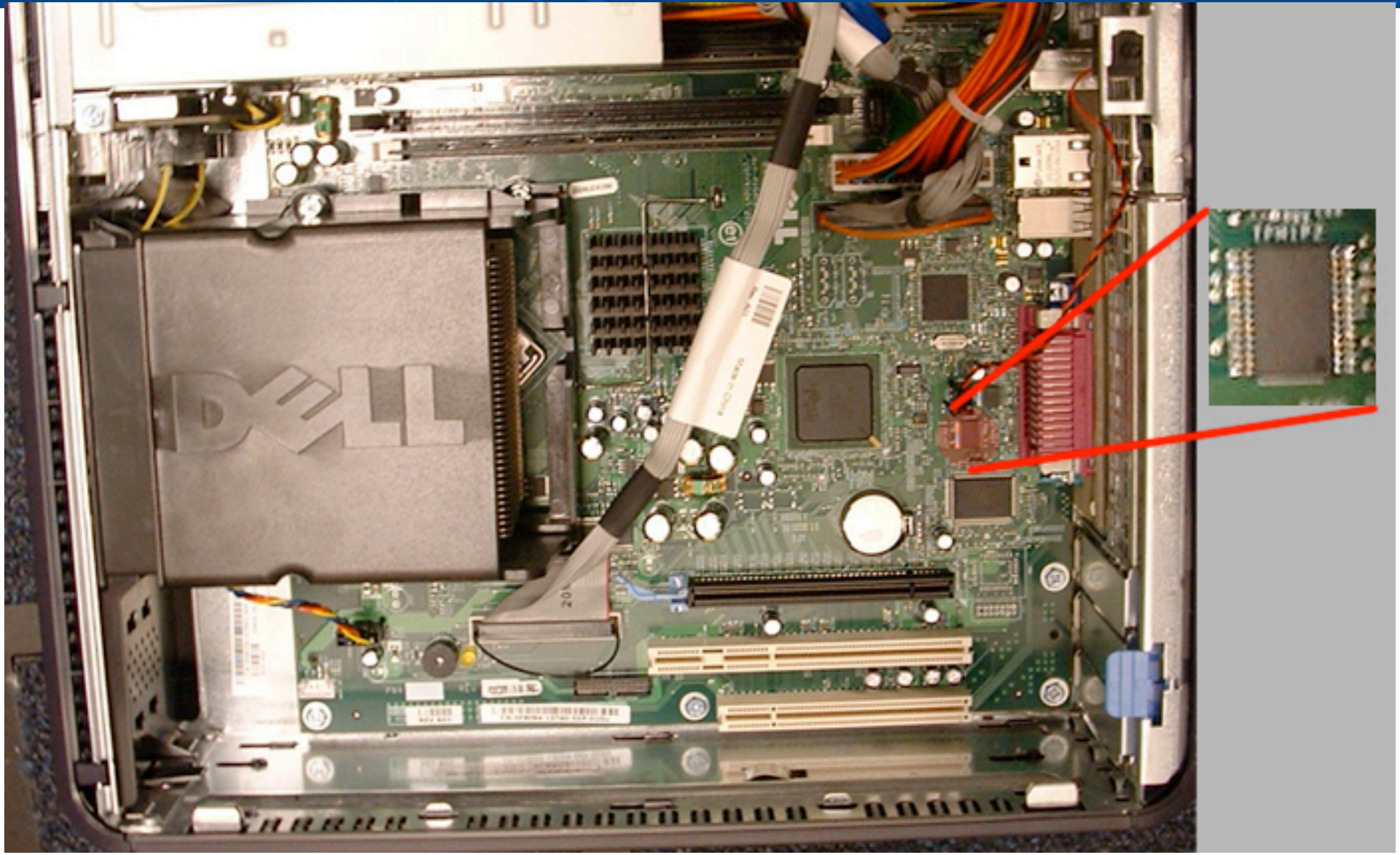
Side Channels

- Challenge - Side Channels
- Untrusted operating system can see all the page faults from each enclave
- Untrusted operating system can cause page faults to occur by unmapping pages
- Researchers have found that such malice can be done on a fine granularity to enable single-stepping of enclaves
- Provides untrusted operating system with a powerful method for detecting the operation of enclaves and possibly leaking data based on their operation

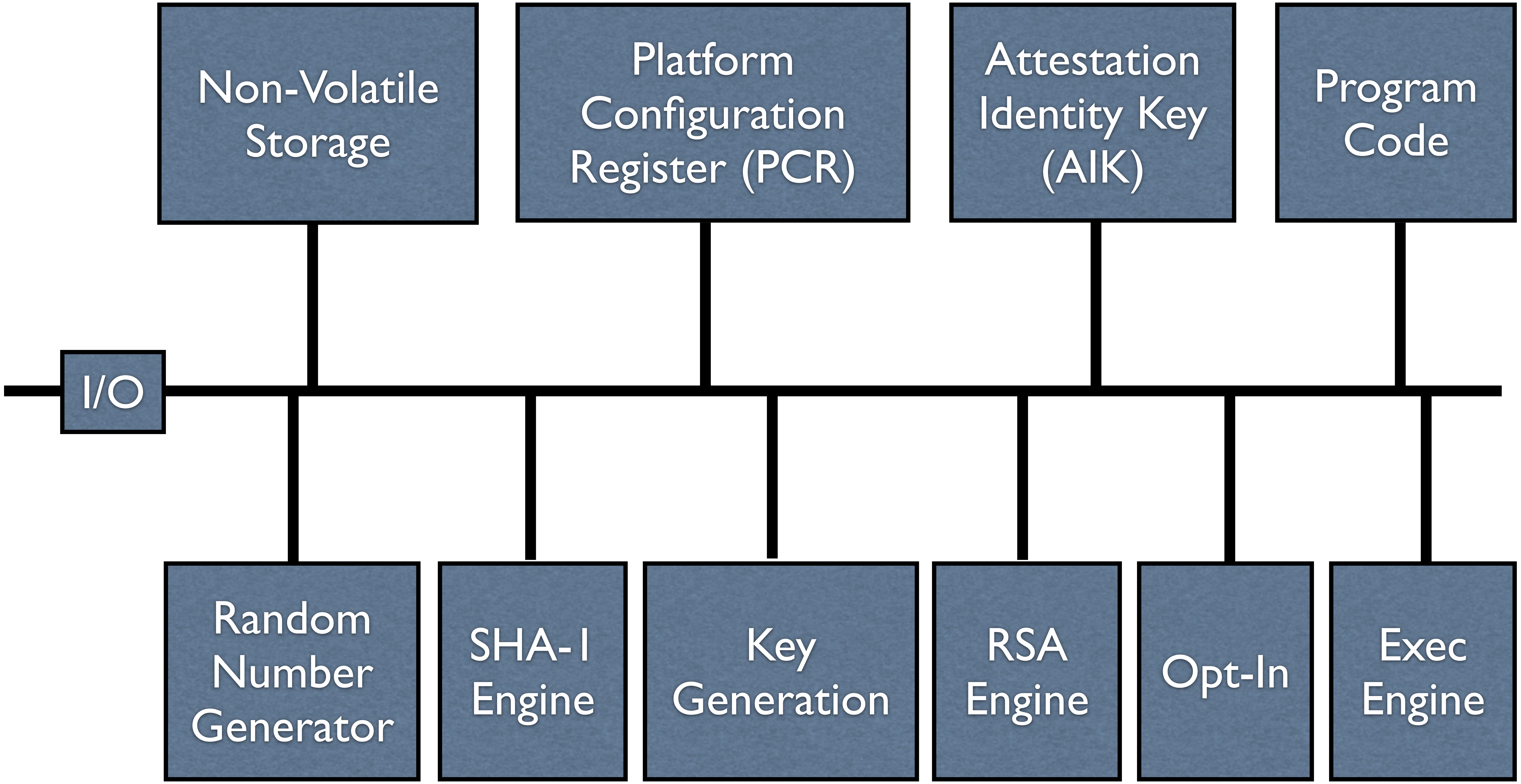
Trusted Platform Module

- The Trusted Platform Module (TPM) provides hardware support for *sealed storage* and *remote attestation*
- What else can it do?
  - www.trustedcomputinggroup.org

Where are the TPMs?



TPM Components Architecture

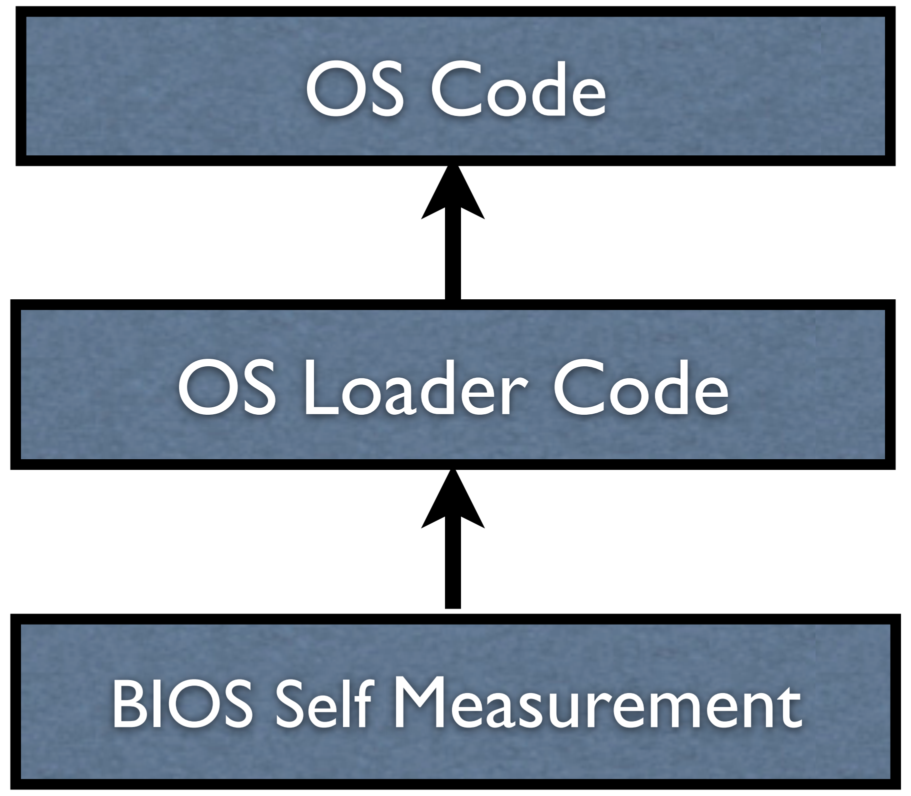


Tracking State

- Platform Configuration Registers (PCRs) maintain state values.
- A PCR can only be modified through the Extend operation
  - Extend(PCR[i], value) :
    - $PCR[i] = SHA1(PCR[i] \cdot value)$
- The only way to place a PCR into a state is to extend it a certain number of times with specific values

Measurement Flow

(Transitive Trust)



Secure vs. Authenticated Boot

- Secure boot *stops execution* if measurements are not correct
- Authenticated boot measures each boot state and lets *remote systems determine if it is correct*
- The Trusted Computing Group architecture uses *authenticated boot*

Integrity Measurement Problem

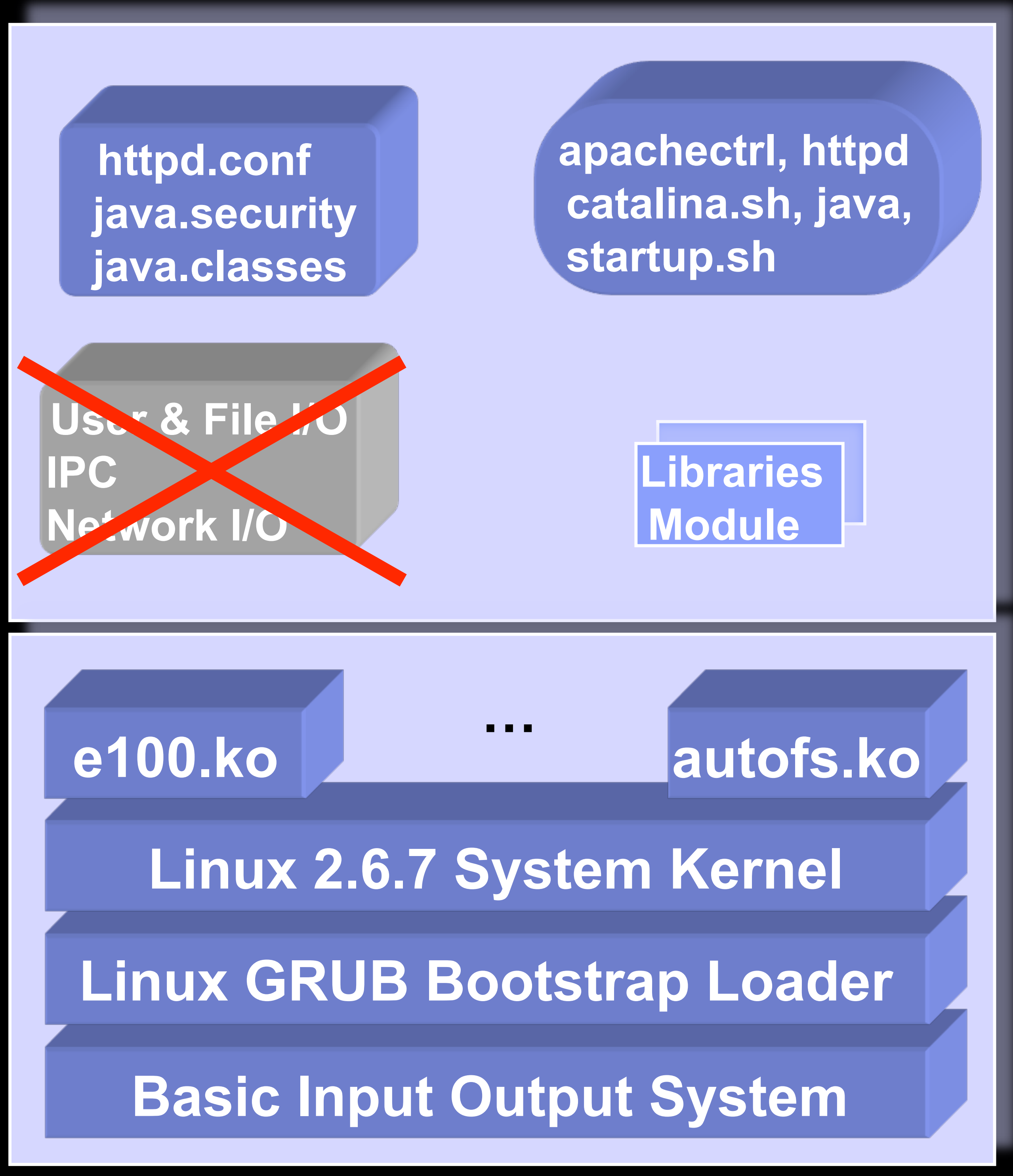
- IPsec and SSL provide secure communication
  - But with whom am I talking?

Integrity Measurement Problem

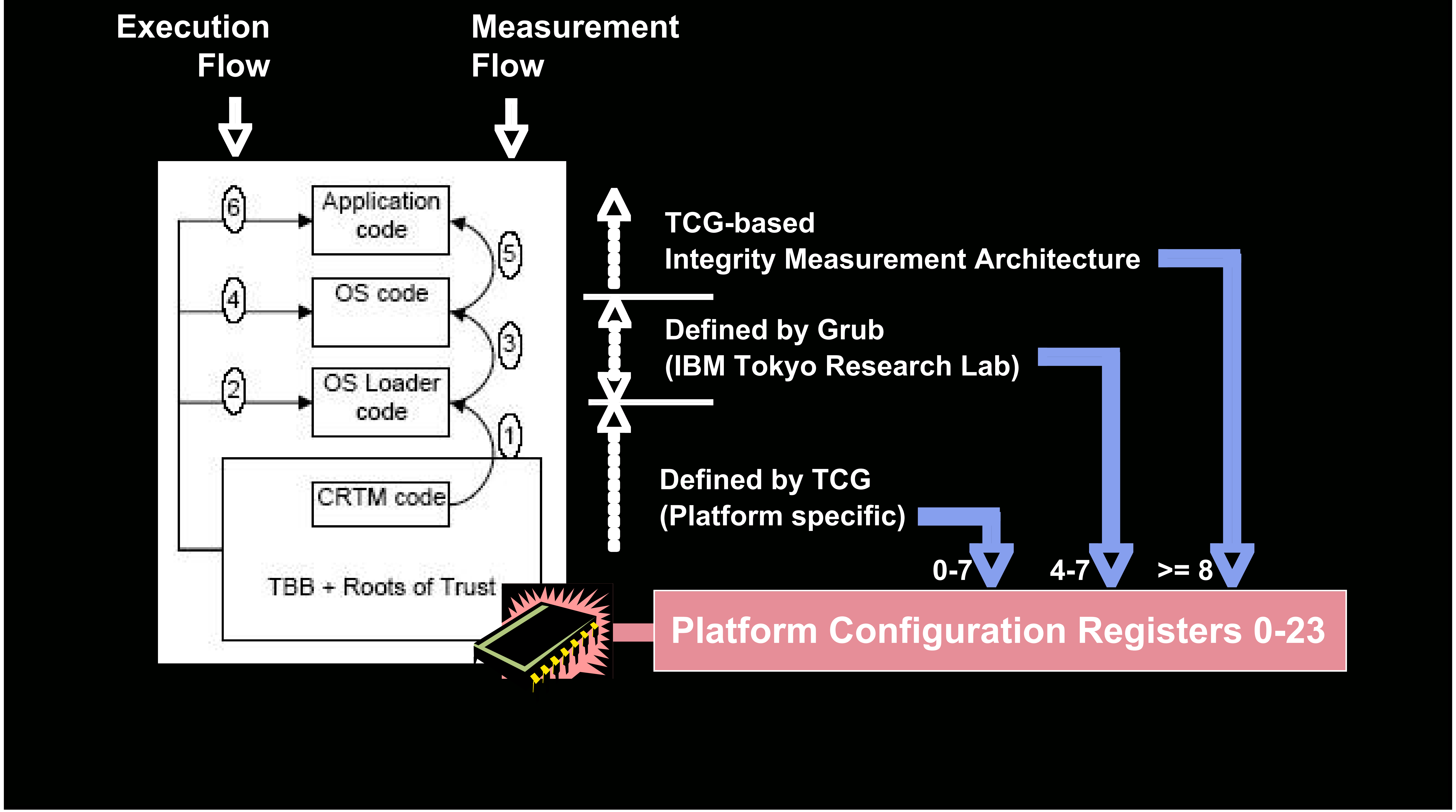
- Measure a web server application is loaded correctly
  - I.e., without malware
  - What should you measure?

Example: Web Server

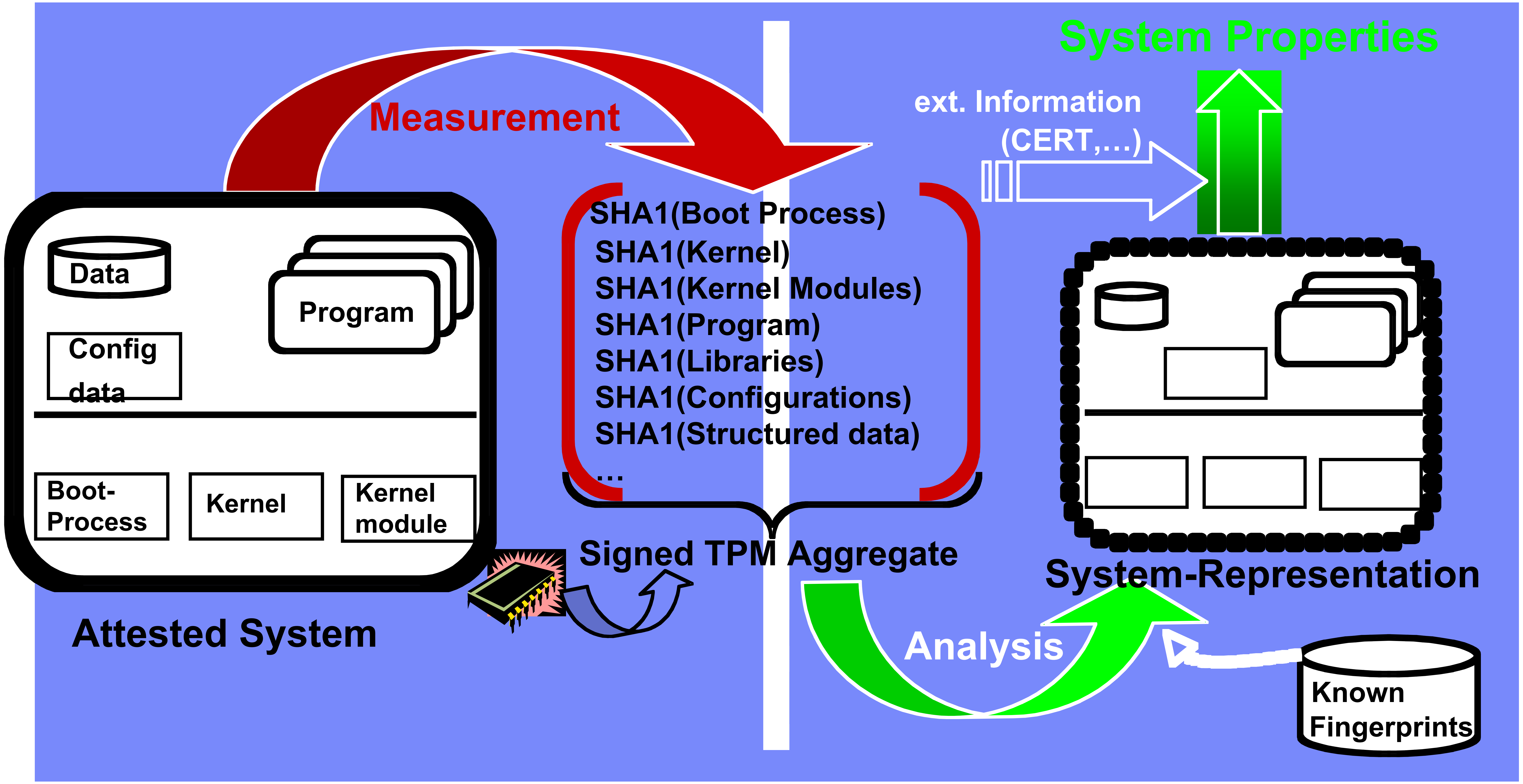
- **Executables** (Program & Libraries)
  - apachectrl, httpd, java, ..
  - mod_ssl.so, mod_auth.so, mod_cgi.so,..
  - libc-2.3.2.so libjvm.so, libjava.so, …
- **Configuration Files**
  - httpd.conf, html-pages,
  - httpd-startup, catalina.sh, servlet.jar
- **Unstructured Input**
  - HTTP-Requests
  - Management Data



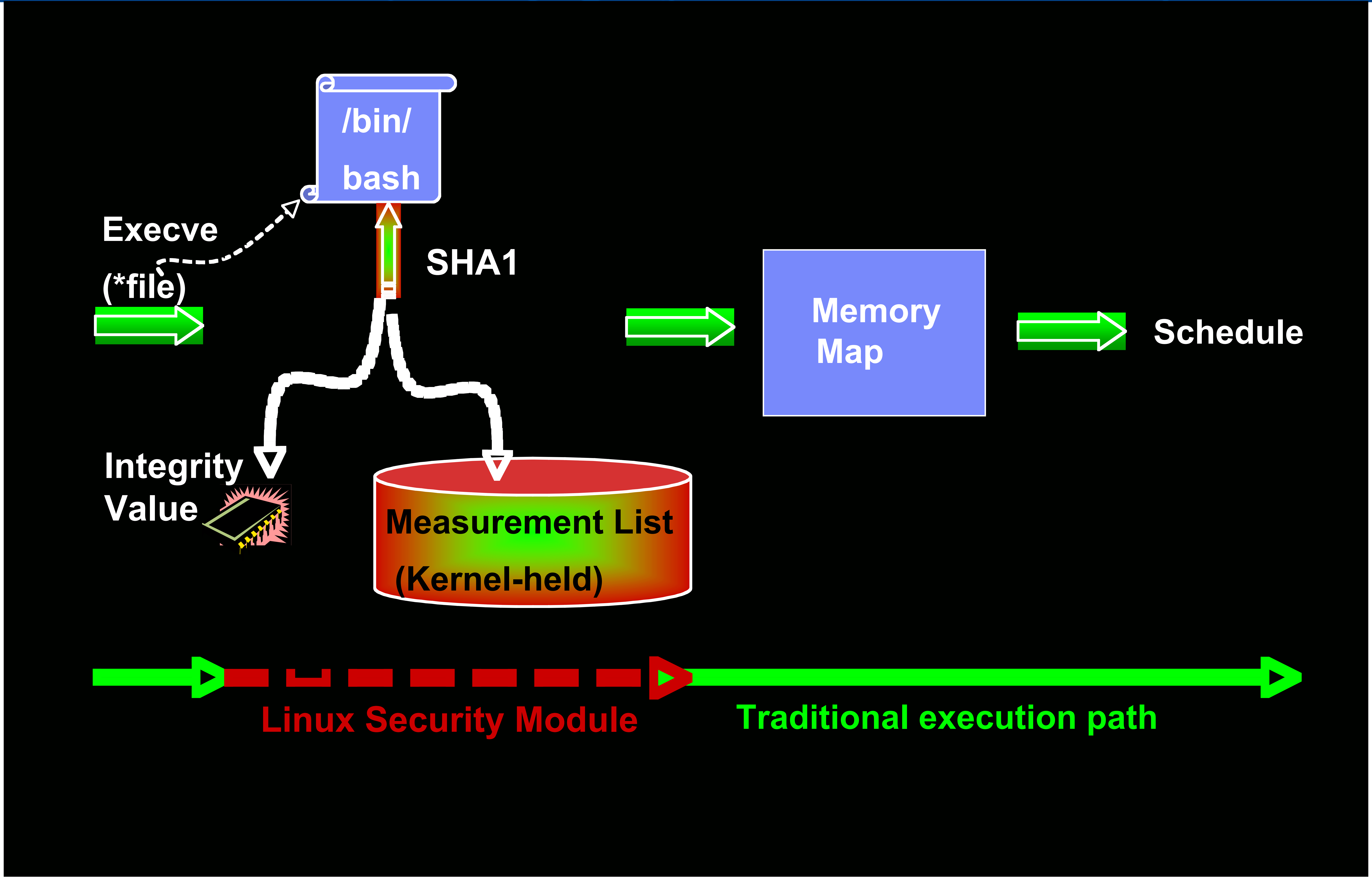
Integrity Measurement Architecture



Collect Hashes



Measurement List



Hardware Security Issues

- Meltdown and Spectre attacks
  - Both based on branch prediction and speculative execution
    - A branch prediction causes a speculative execution to occur that is only committed when the prediction is correct
  - But the speculative execution causes measurable side effects
    - That can enable an adversary to read arbitrary memory from a victim process
- Sound solutions require fixes to processors and updates to ISAs – ad hoc solutions used for now

Spectre Attack

- Attacker locates a sequence of instructions within a victim program that would act as a covert channel
  - From knowledge of victim binary
- Attacker tricks the CPU to execute these instructions speculatively and erroneously
  - Leak victim's info to measurable channel
    - Cache contents can survive nominal state reversion
- To make real, use a cache-based side channel, such as Flush+Reload

Spectre Attack

- Exploiting Conditional Branches

```
if (x < array1_size)
    y = array2[array1[x] * 256];
```

- Suppose an adversary controls the value of ‘x’
- Adversary performs the following sequence
  - First, invoke the program with legal inputs to train the branch predictor to speculatively execute the branch to compute ‘y’
  - Next, invoke the program with an ‘x’ outside bounds of `array1` and where `array1_size` is uncached
  - The operation will read a value from outside the array, and update the cache at a memory location based on the value at `array1[x]`
    - Can learn the value at `array1[x]` from location of cache update

Meltdown Attack

- Meltdown has some similarities

```
raise_exception();
// the line below is never reached
access(probe_array[data * 4096]);
```

- Uses the speculative execution of the above code with an illegal address in ‘data’ to read arbitrary kernel memory
- Adversary performs the following sequence
  - Set data to a kernel memory address
  - The cache entry corresponding to probe_array(data*4096) will be updated based on the value at ‘data’
    - Flush+Reload to detect
- Can leak entire kernel memory

Spectre vs. Meltdown

- Which is worse?
- Meltdown exploits a privilege escalation vulnerability in Intel processors that bypasses kernel memory protections
  - That is a big channel, but only applies to Intel processors
  - Also, the KAISER patch has already been proposed to address the vulnerability being exploited
  - Can be fixed
- Spectre applies to AMD, ARM, and Intel
  - And there is no patch
  - And there are variants that can be exploited – e.g., via JavaScript
  - Do need to find some appropriate victim code tho