CMPSC 447
Hardware Security

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University

Security Problems

- We have discussed lots of security problems
  - Attacks on memory errors
  - Return-oriented attacks
  - Compromised software
- Is there any way new hardware features could prevent some attack vectors?

Areas

- Control-Flow Integrity
  - Can be enforced in software, but is a not as efficient as needed to be applied broadly
    - Instrumentation is a bit complex
- Operating Systems Integrity
  - What to do about the possibility that operating systems may be compromised?
  - Can we prevent code injection and reuse?
  - Do we really need to trust operating systems?
- Hardware features have been made available that start to answer these kinds of questions

Control-Flow Integrity

PENNSTATE 1855

- What do you need to do to enforce control flow integrity?

Control-Flow Integrity

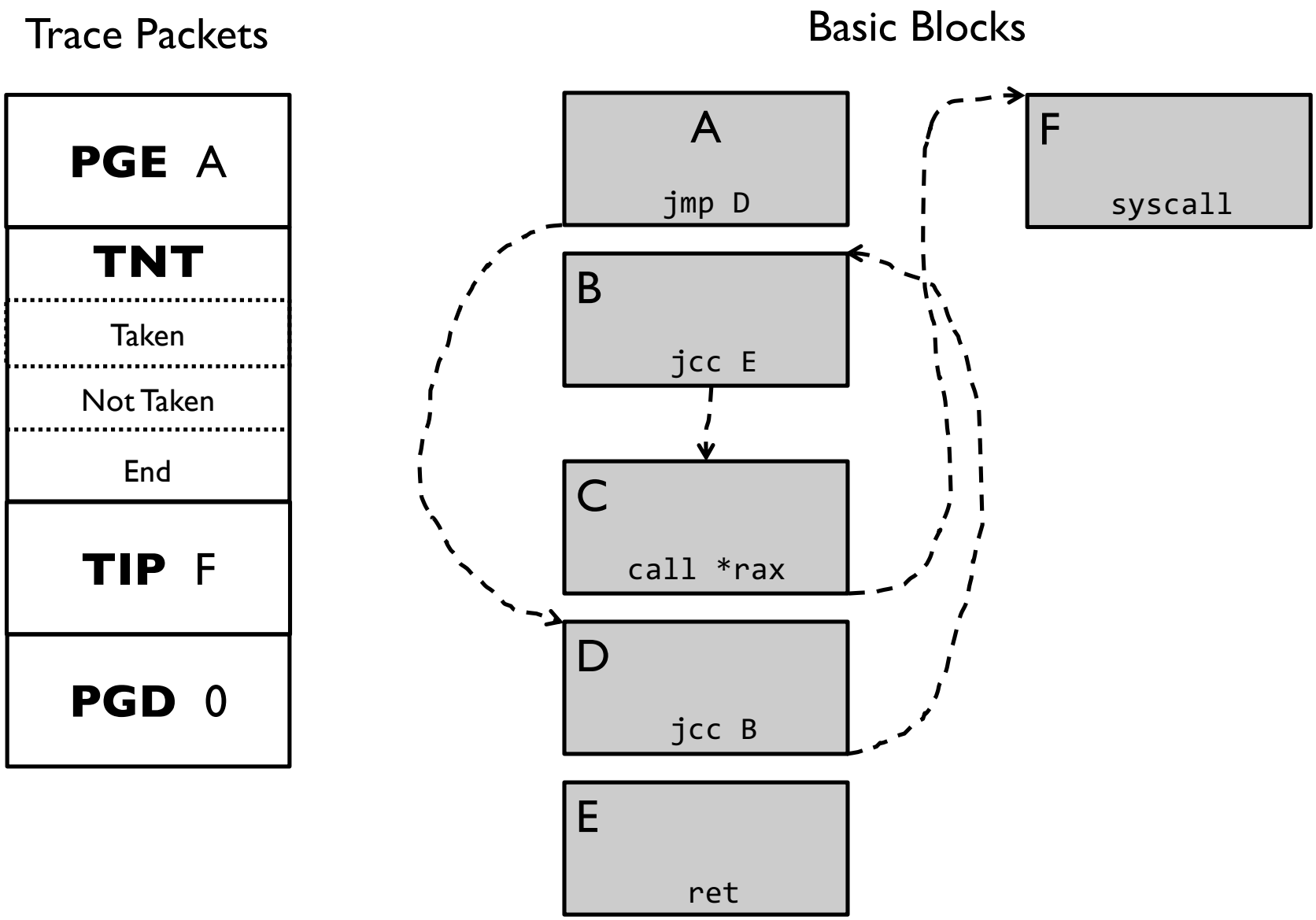
PENNSTATE 1855

- What do you need to do to enforce control flow integrity?
- Forward edges (indirect calls and jumps)
  - For each indirect control transfer (source), ensure that the chosen target complies with the program' CFG for that source (Fine-grained CFI)
- Backward edges (returns)
  - For each return, ensure that the target is associated with the originating call site (Shadow Stack)
    - May be exceptions, but handle exceptionally

Intel Processor Trace

- A new hardware feature that enables efficient recording of control-flow and timing information about software execution (3-5% overhead)
  - Initially available on the Broadwell processor
  - Fully implemented on the Skylake processor
- At each control choice, record a packet in memory
  - Conditional branches
  - Indirect call
  - Returns
- Enough to reconstruct the actual control flow

Intel PT Example



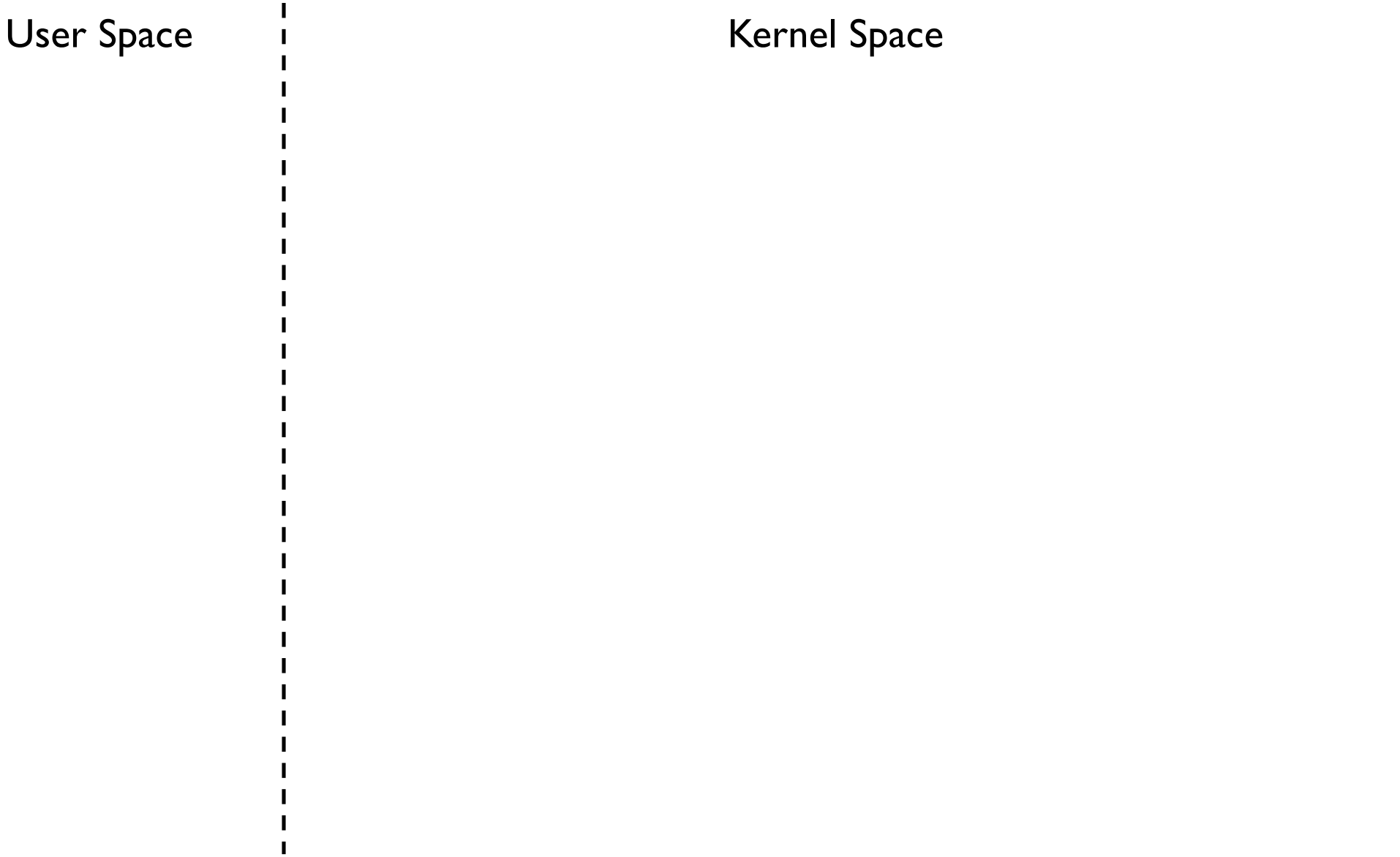
When to Check?

- Since we are using Intel PT to log the program's execution, we are naturally running behind
  - Is this sufficient to enforce CFI?
  - A forward or backward edge may already have been exploited

When to Check?

- Since we are using Intel PT to log the program's execution, we are naturally running behind
  - Is this sufficient to enforce CFI?
  - A forward or backward edge may already have been exploited
- While an exploit may be underway, the exploit cannot really have an impact until a system call occurs
  - Modify unauthorized data persistently (except for memory-mapped files)
  - Leak sensitive data to others

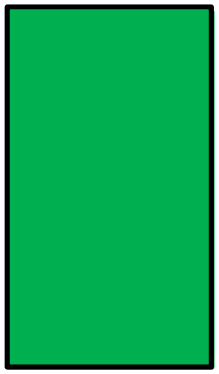
System Overview



System Overview

User Space

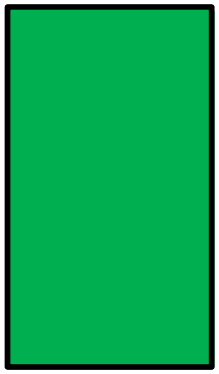
Kernel Space



System Overview

User Space

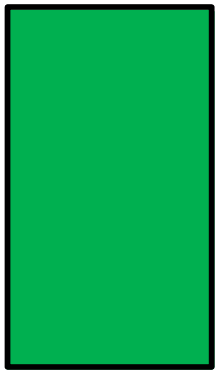
Kernel Space

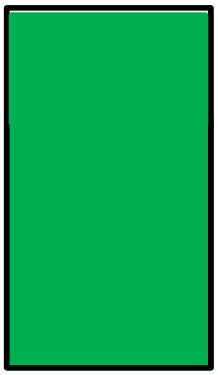


System Overview

User Space

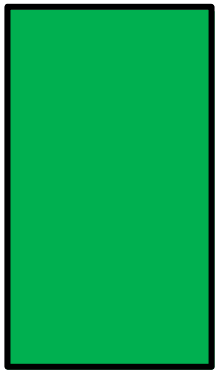
Kernel Space



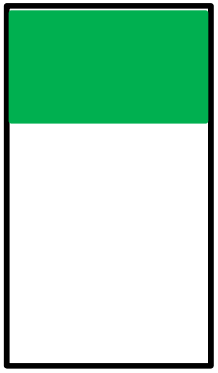


System Overview

User Space

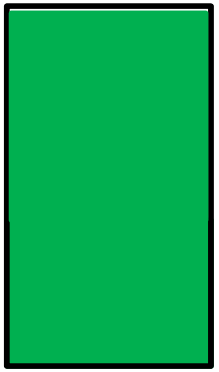


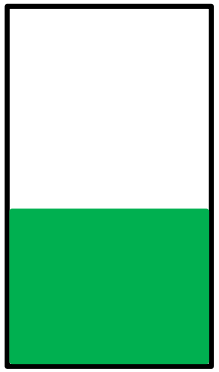
Kernel Space



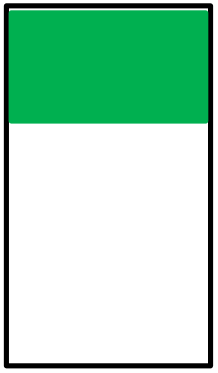
System Overview

User Space





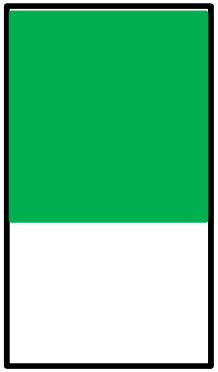
Kernel Space



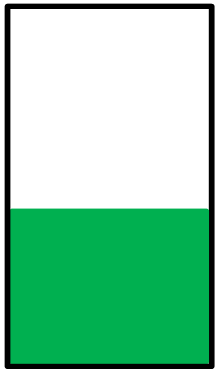
System Overview

User Space

Kernel Space



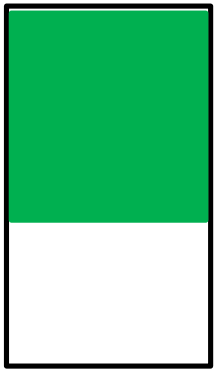
SYSCALL

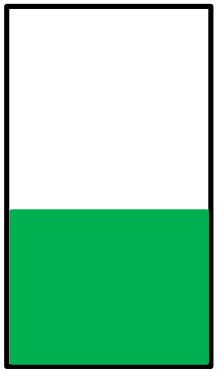


System Overview

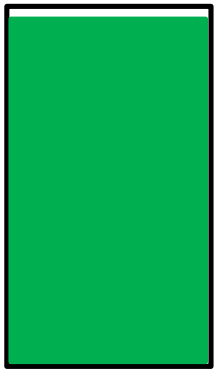
User Space

Kernel Space





SYSCALL



What To Do?

Depends on the enforced policy

CFI Policies

- Coarse-grained Policy
  - Check if the targets of indirect control transfers are valid
  - Requires decoding the trace packets to find each target
- Fine-grained Policy
  - Check if the source and destination are a legitimate pair
  - Requires control-flow recovery to identify source
- Stateful Policy
  - Check if an indirect control transfer is legitimate based on the program state (e.g., shadow stack)
  - Requires sequential processing if state spans trace buffers

Using Intel PT for CFI

PENNSTATE 1855

- What do you need to do to leverage an Intel PT trace to enforce fine-grained CFI?

Using Intel PT for CFI

- What do you need to do to leverage an Intel PT trace to enforce fine-grained CFI?
  - Need to collect the source and target of each indirect call

Using Intel PT for CFI

- What do you need to do to leverage an Intel PT trace to enforce fine-grained CFI?
  - Need to collect the source and target of each indirect call
- How do you find these from Intel PT trace?
  - Target is recorded in a packet
  - But how do we find the source?

Using Intel PT for CFI

- What do you need to do to leverage an Intel PT trace to enforce fine-grained CFI?
  - Need to collect the source and target of each indirect call
- How do you find these from Intel PT trace?
  - Target is recorded in a packet
  - But how do we find the source?
- Reconstructing the control flow from the trace identifies the sources
  - Then can perform authorization

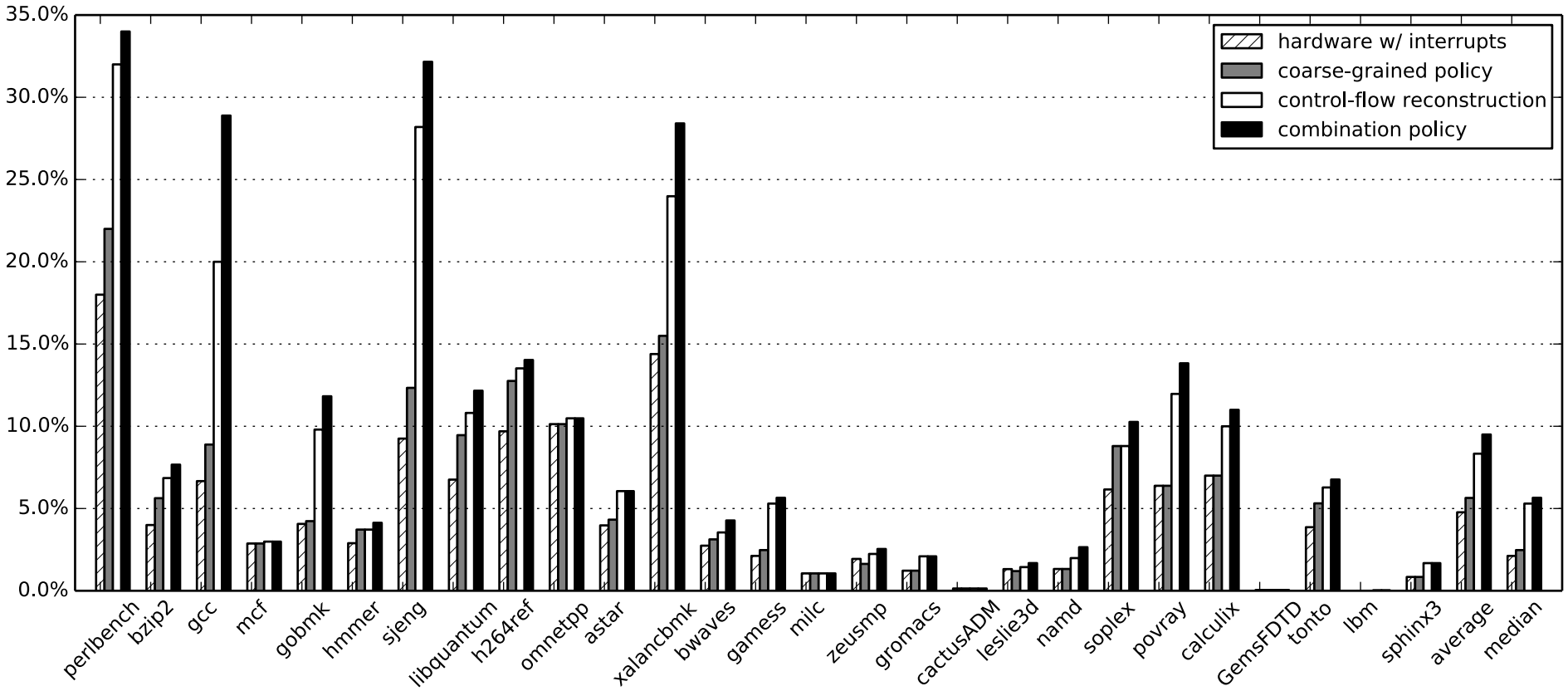
Fine-Grained CFI

PENNSTATE 1855

- Recover the control flow from the trace buffer and the program binaries to identify sources
  - Disassemble the binary online in basic blocks
  - Traverse basic blocks using the trace buffer to find sources of indirect control transfers
- Authorize each indirect control transfer target against that program's fine-grained policy for source
  - For each indirect control transfer found in the trace ensure that the destination is in the legal target set of the corresponding source

Evaluation

- SPEC CPU2006
  - Average: 9.5%, Median: 5.6% for complete enforcement
  - Shadow stack (backward) and fine-grained CFI (forward)



CFI-Focused Logging

- Could you further optimize the hardware logging for CFI enforcement?
  - Can we eliminate need for control-flow recovery to enforce fine-grained CFI policies?

CFI-Focused Logging

- Could you further optimize the hardware logging for CFI enforcement?
  - Can we eliminate need for control-flow recovery to enforce fine-grained CFI policies?
- Intel PT could record the source in a packet as well as the target packet
  - And ignore recording other information not necessary for fine-grained CFI – taken/not-taken
    - This combination of changes reduces overhead for checking by over 90% on average
    - But, not clear what impact on hardware overhead

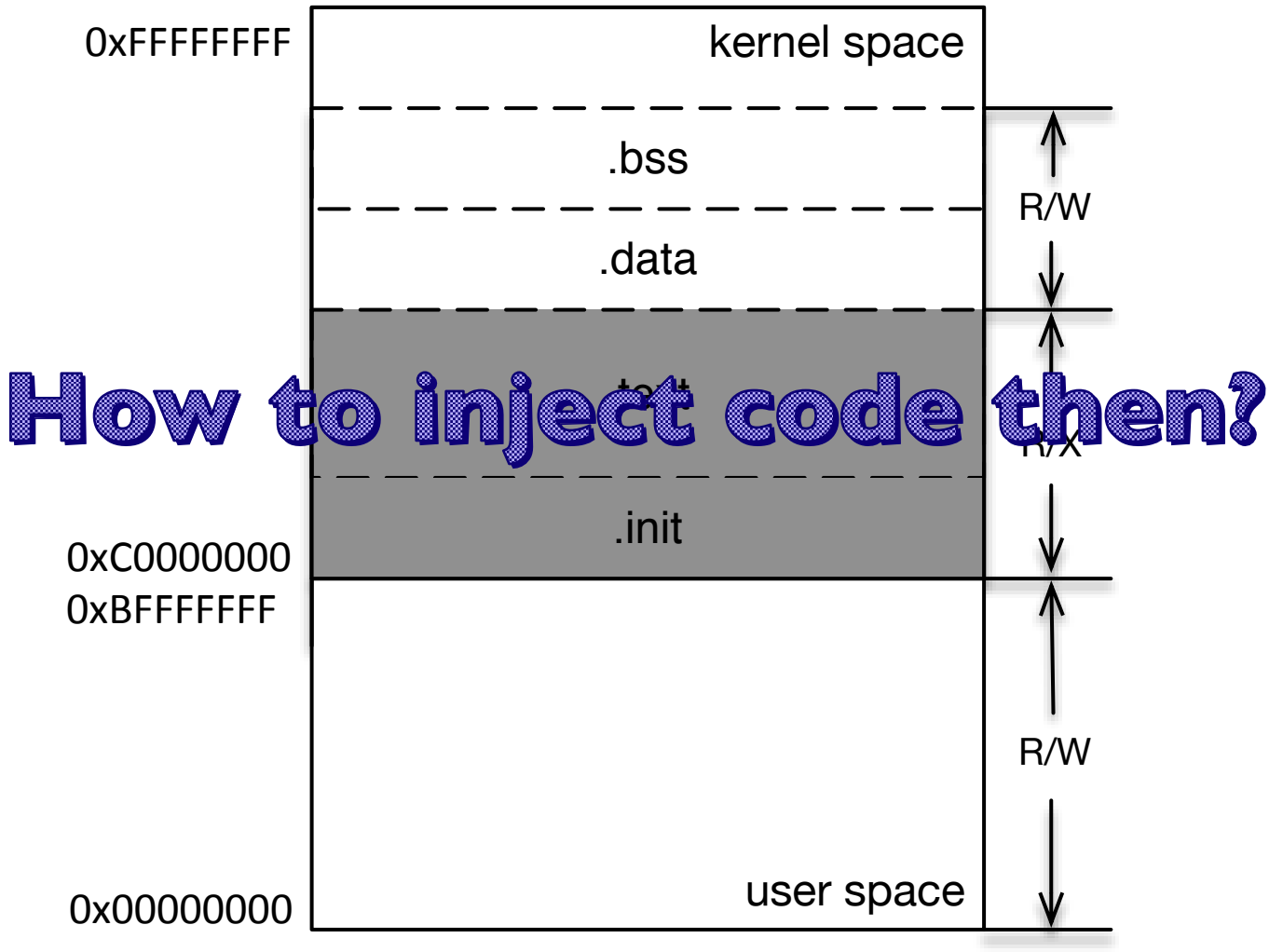
Intel CET

- Intel Control-Flow Enforcement Technology (CET) aims to enforce shadow stack defenses in hardware
  - Announced in June 2017
  - Now available in Intel's 11th generation CPU
- Shadow Stack on backward edge
  - Exception on failure – for handler to deal with
- Indirect Branch Tracking on forward edge
  - Restrict indirect calls/jumps to valid targets
  - Weak – Single class of valid targets for all calls (coarse)

Preventig Code Injection

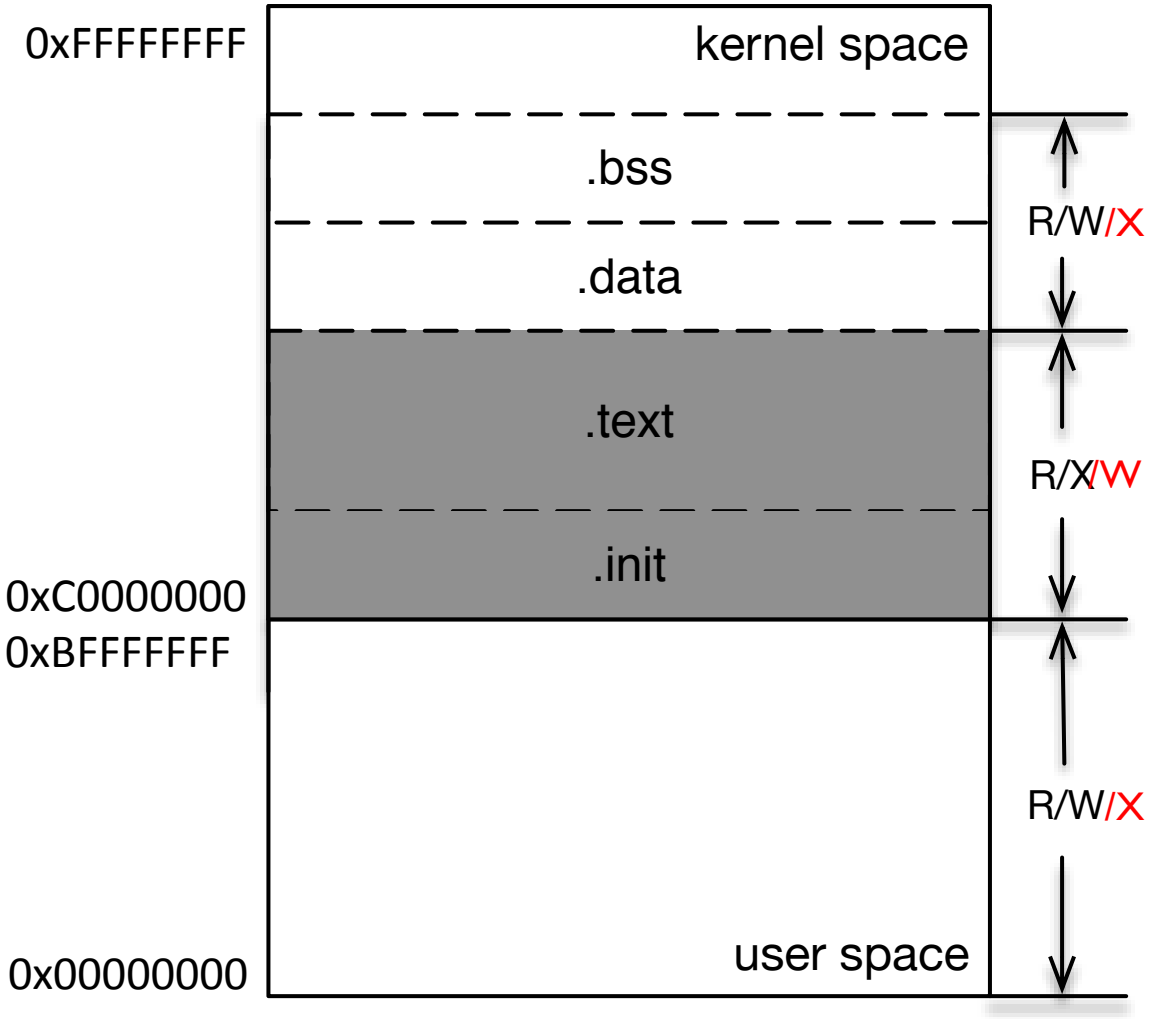
- Preventing code injection is a key defense
  - We prevent code injection in user space using W xor X
  - Which is implemented by the kernel
- What if the kernel itself is compromised? Or hijacked program tries to disable protection?
  - Turn off W xor X
  - So, code injection itself is trivial
- Can we prevent kernel code injection – even when the kernel is compromised?

Lifetime Kernel Code Integrity



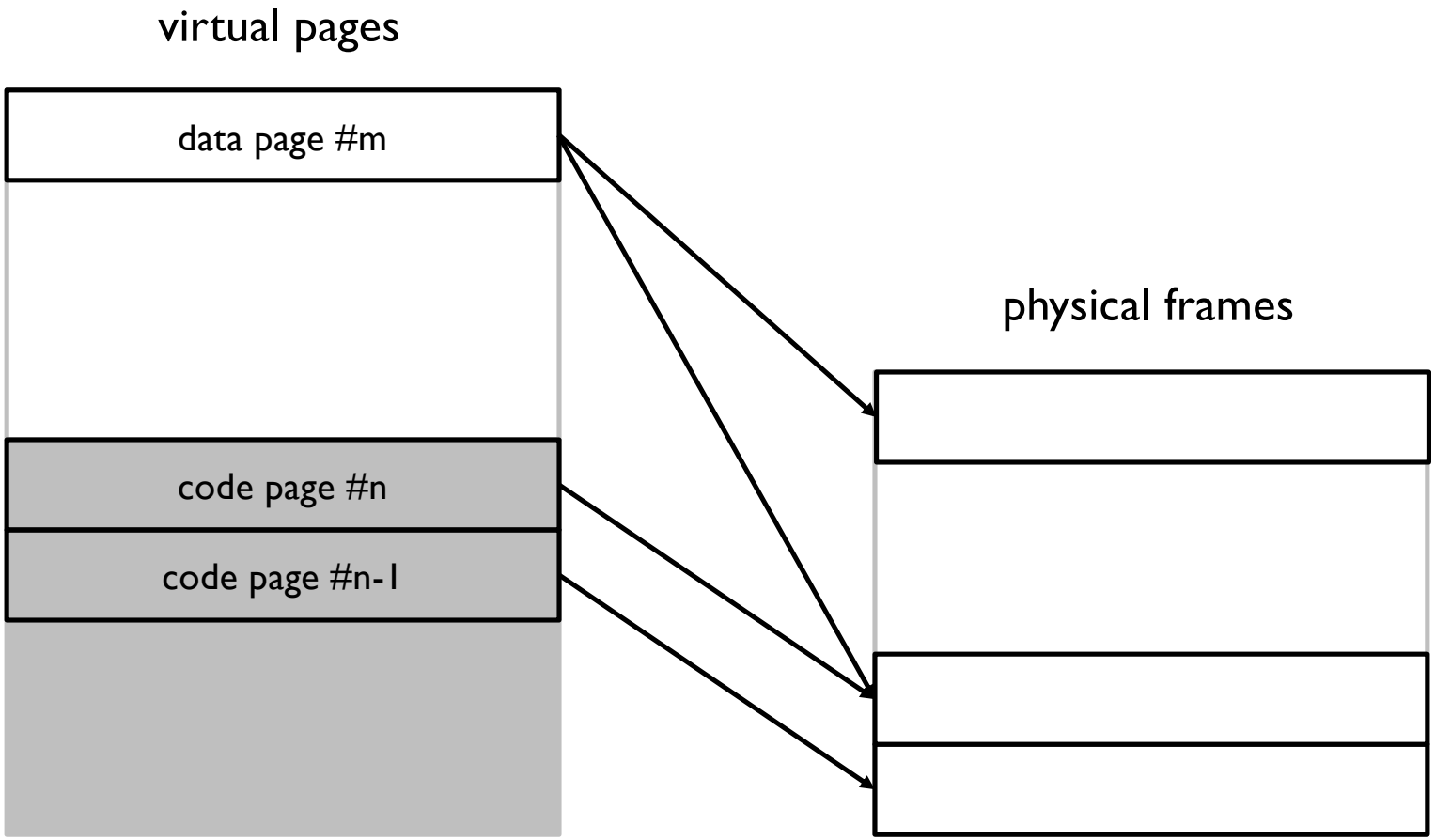
Attack on Permissions

- Tamper with permissions



Attack on Mappings

- Tamper with mappings



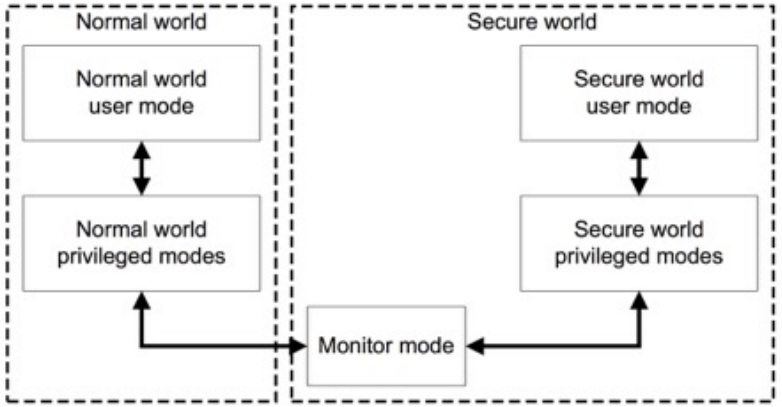
Goal

PENNSTATE 1855

Prevent both types of attacks and limit the adversary to approved kernel code on the TrustZone architecture

Background: TrustZone

- Resources are partitioned into two distinct worlds
  - Physical memory, interrupts, peripherals, etc.
- Each world has its autonomy over its own resources
- Secure world can access normal world resources, but not vice versa
- Run in time-sliced fashion



SPROBE Placement

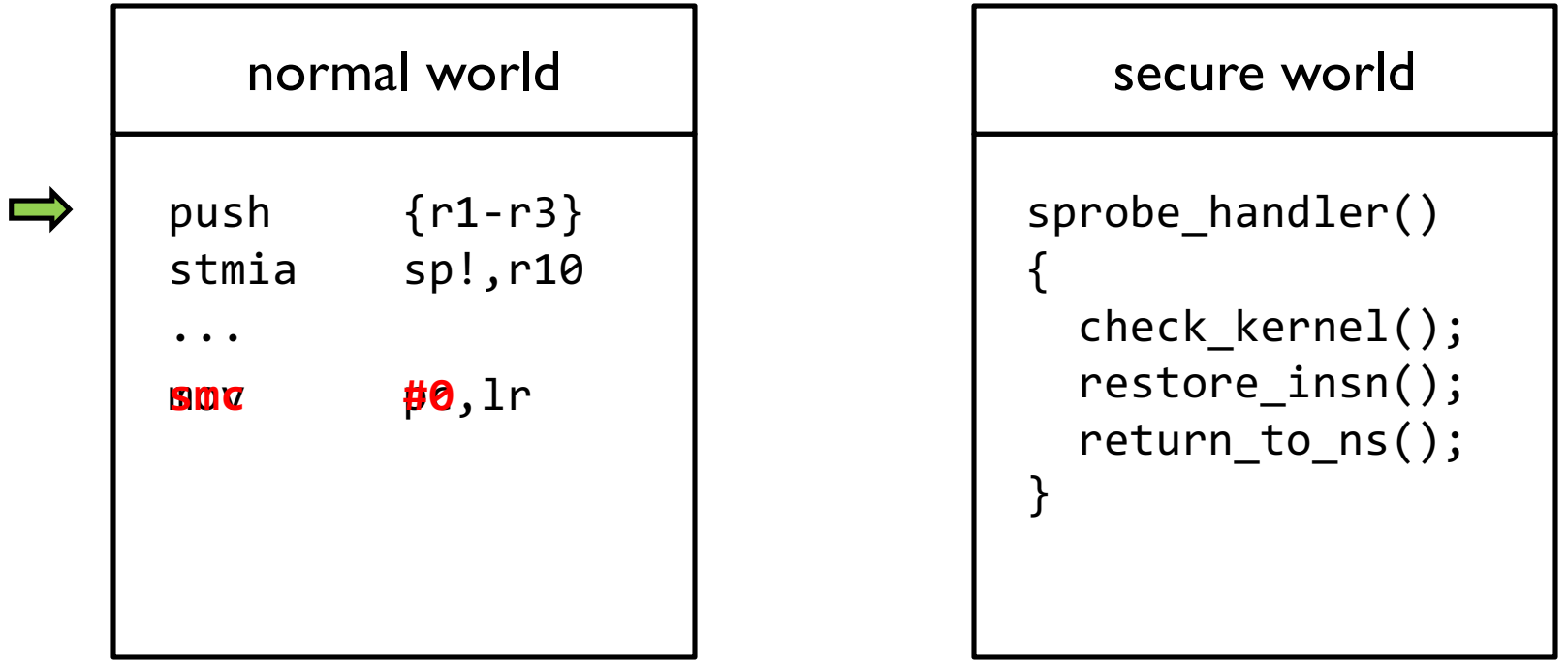
- Recall the specific attacks
  - Change to a different set of page tables that are under attacker's control
    - instrument all instructions that can be potentially used to switch the page table root
  - Modify page table entries in place
    - write-protect the whole page tables and instrument the first instruction in page fault handler

SPROBES Invariants

- **S1**: Execution of user space code from the kernel must never be allowed.
- **S2**: W⊕X protection employed by the operating system must always be enabled.
- **S3**: MMU must be kept enabled to ensure all existing memory protections function properly.
- **S4**: The page table base address must always correspond to a legitimate page table.
- **S5**: Any modification to the page table entry must not make a kernel code page writable or make a kernel data page executable.

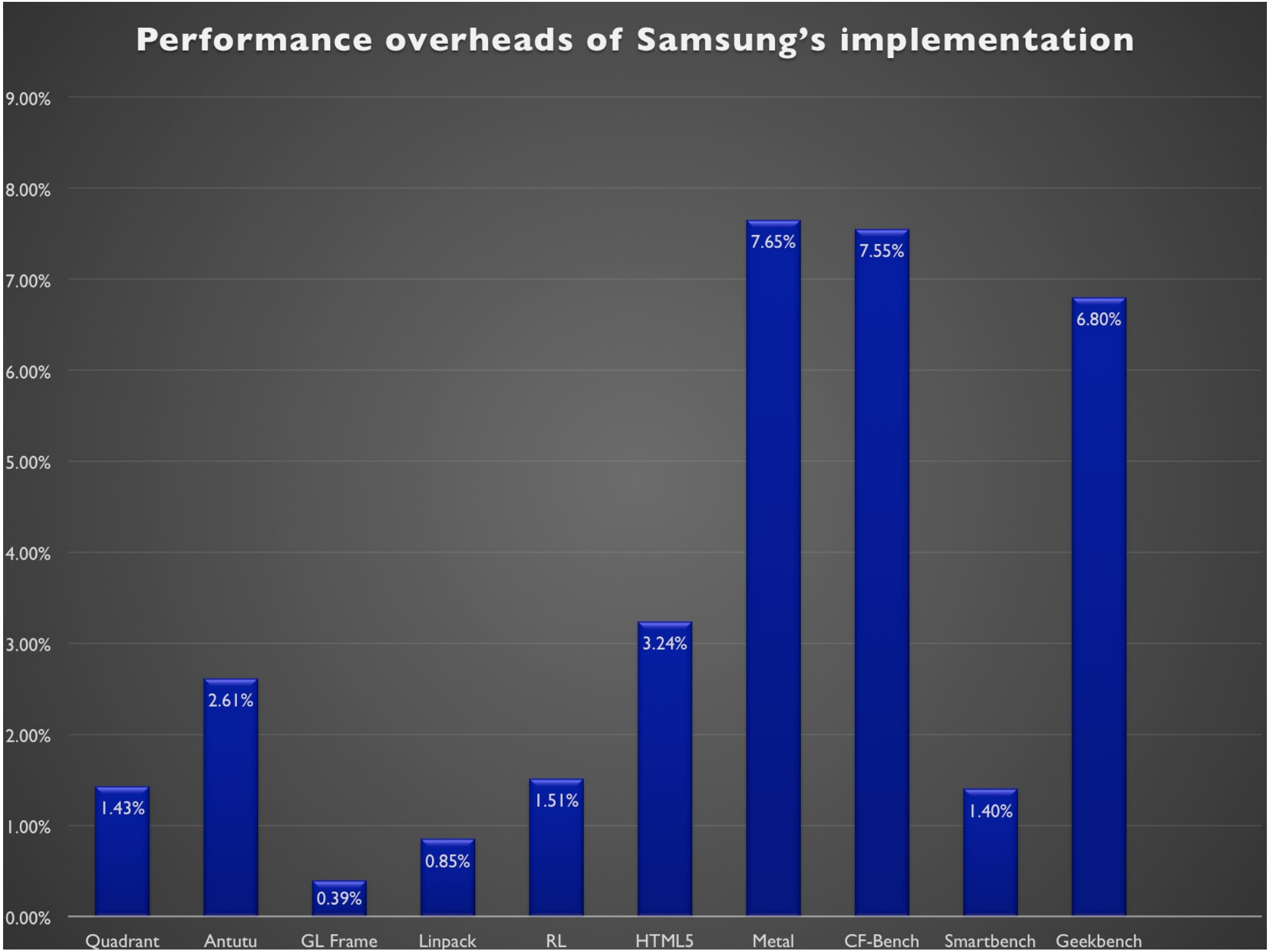
SPROBE Mechanism

- We need an instrumentation mechanism that enables the secure world to be notified upon events of its choice in the normal world



A Little Bit More…

- Samsung has implemented the same idea and deployed this technique on millions of devices


Performance overheads of Samsung's implementation
9.00%
8.00%
7.00%
6.00%
5.00%
4.00%
3.00%
2.00%
1.00%
0.00%
1.43%
2.61%
0.39%
0.85%
1.51%
3.24%
7.65%
7.55%
1.40%
6.80%
Quadrant
Antutu
GL Frame
Linpack
RL
HTML5
Metal
CF-Bench
Smartbench
Geekbench

Eliminate Trust in OS

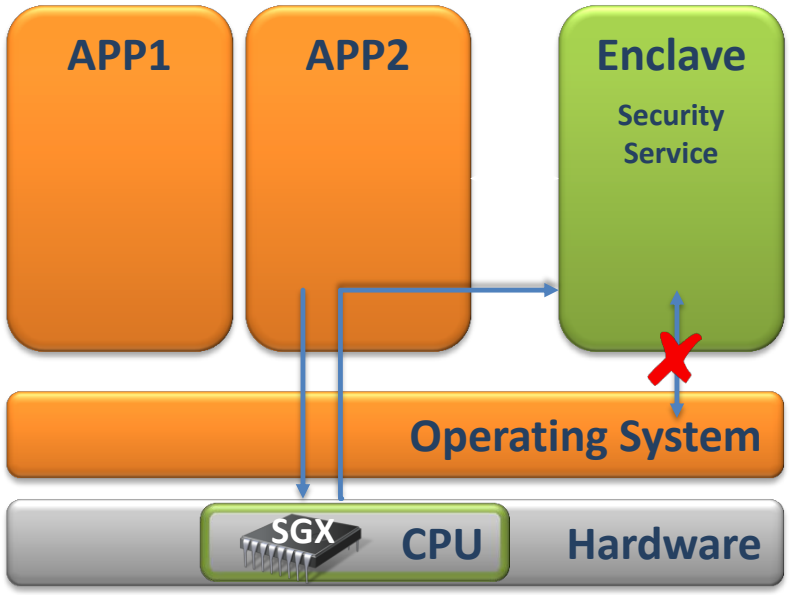
- The OS may not be secure itself
  - Millions of lines of code
  - Complex and evolving codebase, including device drivers
- What if you want to eliminate trust in the OS altogether?

PENNSTATE 1855

Intel® Software Guard Extensions (SGX)

[McKeen et al, Hoekstra et al., Anati et al., HASP'13]

- **Security critical code isolated in enclave**
- **Only CPU is trusted**
  - Transparent memory encryption
  - 18 new instructions
- **Enclaves cannot harm the system**
  - Only unprivileged code (CPU ring3)
  - Memory protection
- **Designed for Multi-Core systems**
  - Multi-threaded execution of enclaves
  - Parallel execution of enclaves and untrusted code
  - Enclaves are interruptible
- **Programming Reference available**



SYSTEM SECURITY LAB

PENNSTATE 1855

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)

PENNSTATE 1855

SGX Memory Access Control

- **Access control in two direction**
  - From enclaves to “outside“
    - Isolating malicious enclaves
    - Enclaves needs some means to communicate with the outside world, e.g., their “host applications”
  - From “outside“ to enclaves
    - Enclave memory must be protected from
      - Applications
      - Privileged software (OS/VMM)
      - Other enclaves

OS: Operating System
VMM: Virtual Machine Monitor (also known as Hypervisor)

SYSTEM SECURITY LAB

PENNSTATE 1855

SGX MAC "outside" to enclaves

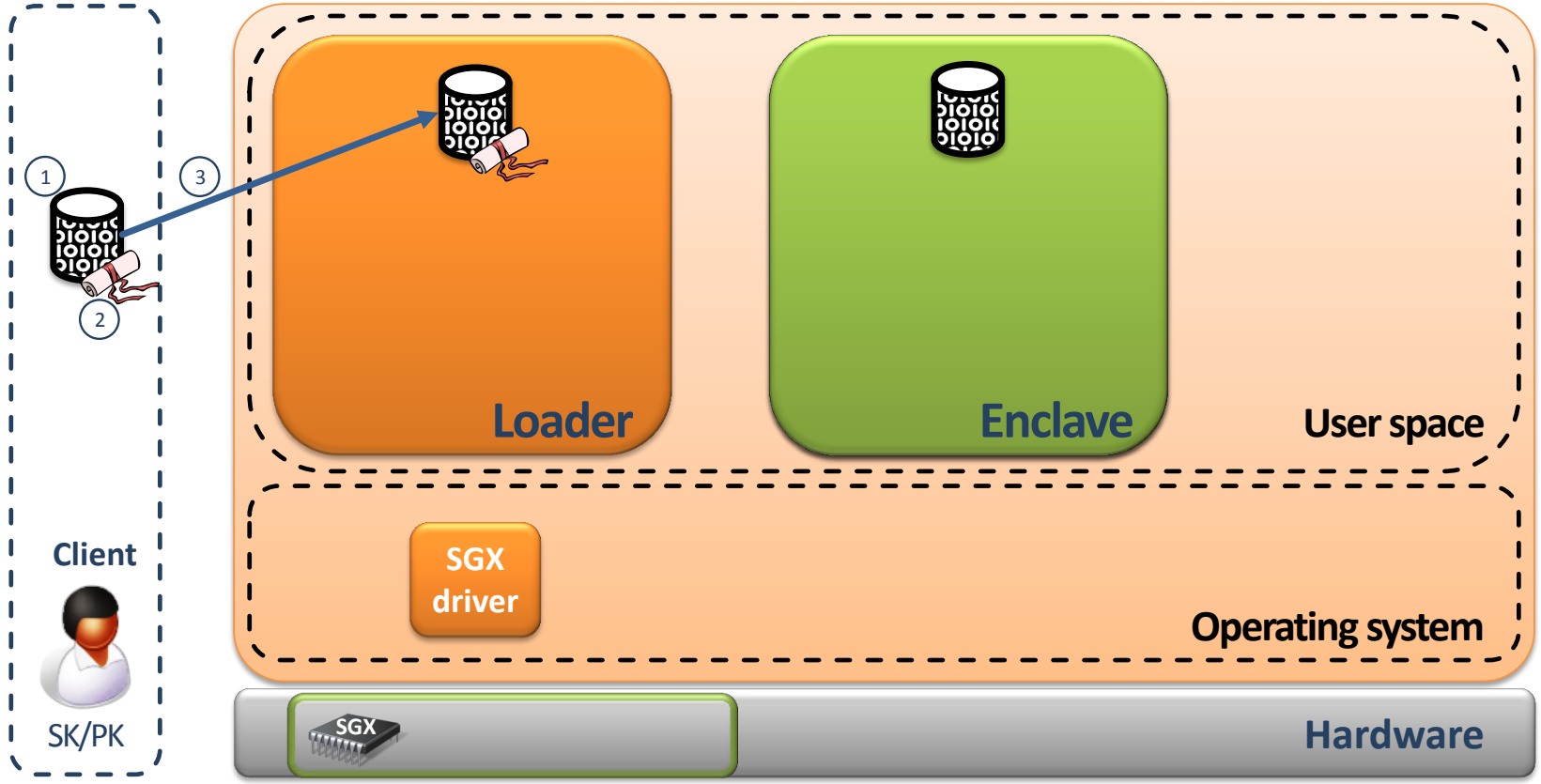
- **From "outside" to enclaves**
  - Non-enclave accesses to EPC memory results in abort page semantics
  - Direct jumps from outside to any linear address that maps to an enclave do not enable enclave mode and result in a about page semantics and undefined behavior
  - Hardware detects and prevents enclave accesses using logical-to-linear address translations which are different than the original direct EA used to allocate the page. Detection of modified translation results in #GP(0)

MAC: Memory Access Control
EA: Enclave Access
#GP(0): General Protection Fault

SYSTEM SECURITY LAB

PENNSTATE 1855

SGX – Create Enclave

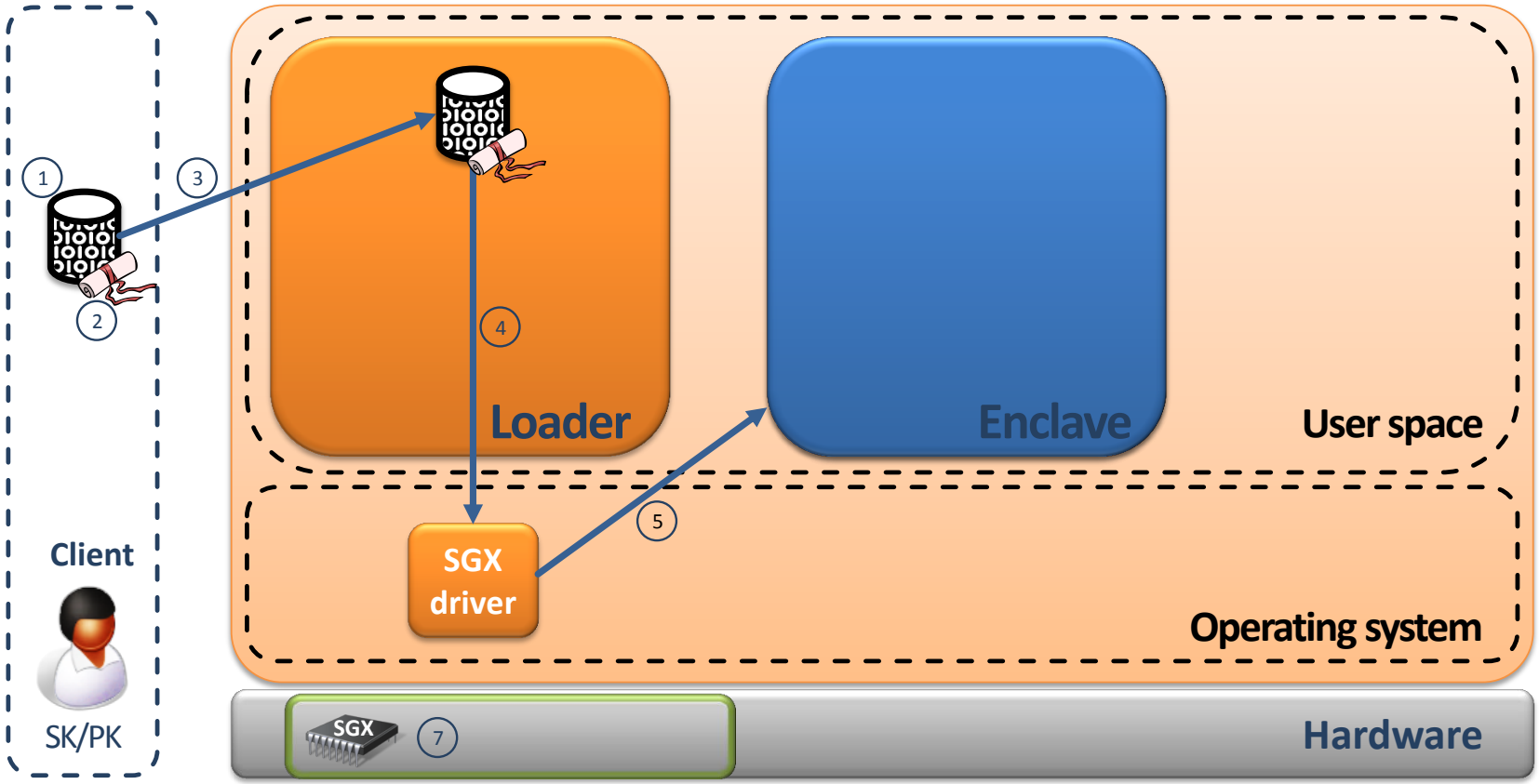


1. Create App
2. Create app certificate (includes HASH(App) and Client PK)
3. Upload App to Loader

Trusted Untrusted

PENNSTATE 1855

SGX – Create Enclave

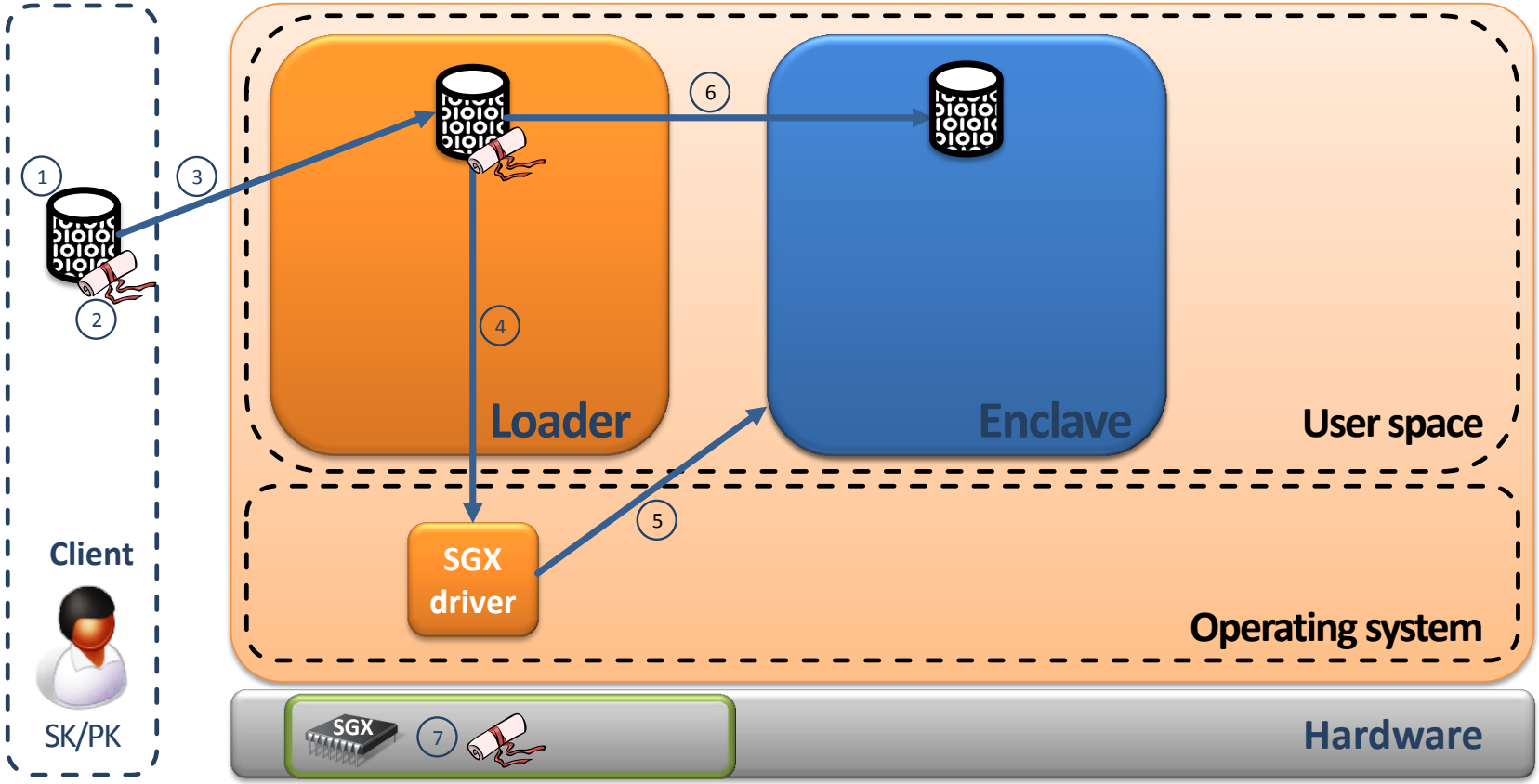


1. Create App
2. Create app certificate (includes HASH(App) and Client PK)
3. Upload App to Loader
4. Create enclave
5. Allocate enclave pages

Trusted Untrusted

Intel Software Guard Ext

SGX – Create Enclave

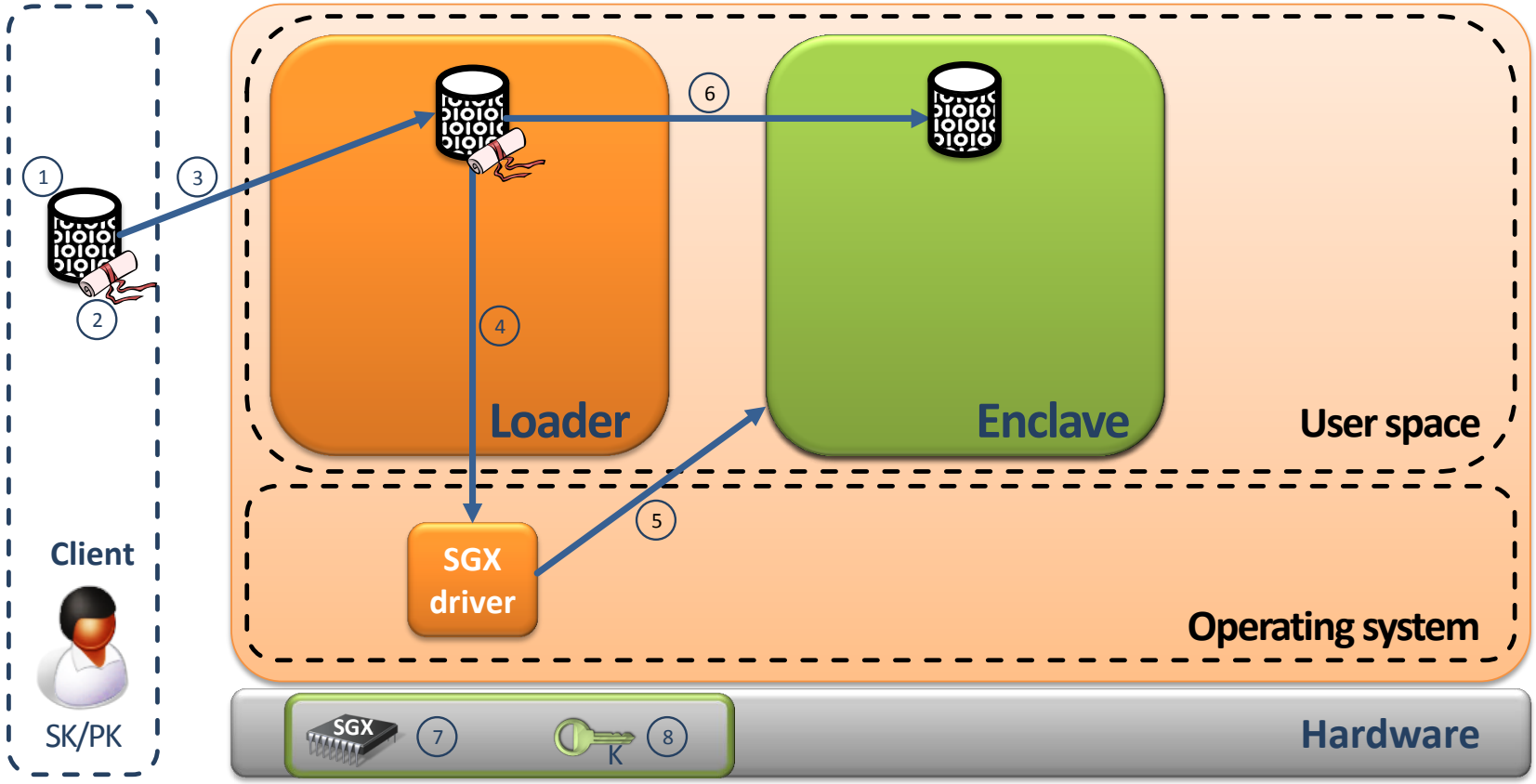


1. Create App
2. Create app certificate (includes HASH(App) and Client PK)
3. Upload App to Loader
4. Create enclave
5. Allocate enclave pages
6. Load & Measure App
7. Validate certificate and enclave integrity

Trusted | Untrusted

Intel Software Guard Ext

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

SGX Security Issues

- Lots of ways to leak information about a program running in an enclave if the adversary controls the operating system
  - Operating system can see…
  - Page faults
  - Cache effects
  - Branch prediction
  - Speculative execution
- As a result, the broad use of SGX has been limited

Take Away

- Lots of efforts in exploring hardware features to improve security
  - CFI enforcement via Intel PT
    - Hardware may need to be optimized further
  - Isolate code from untrusted kernel – SGX and TZ
- However, there are also security issues with such hardware mechanisms
  - Side Channels