TENCENT SECURITY CONFERENCE 2018
2018腾讯安全国际技术峰会

Enabling Confidential Computing in Cloud with Intel SGX and Library OSes

Hongliang Tian

Research Scientist at Intel Labs China

- *System researcher*
- Expert on trusted hardware (e.g, *Intel SGX*)
- Email: hongliang.tian@intel.com

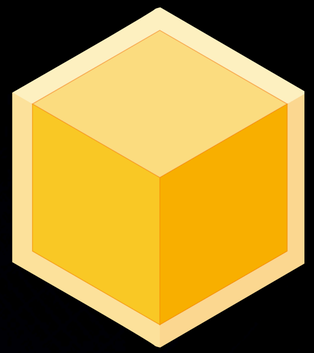
A New Trend of Cloud Security is *Confidential Computing*

Sep. 2017

Azure Confidential Computing

Started accepting preview requests

May 2018



Google Asylo

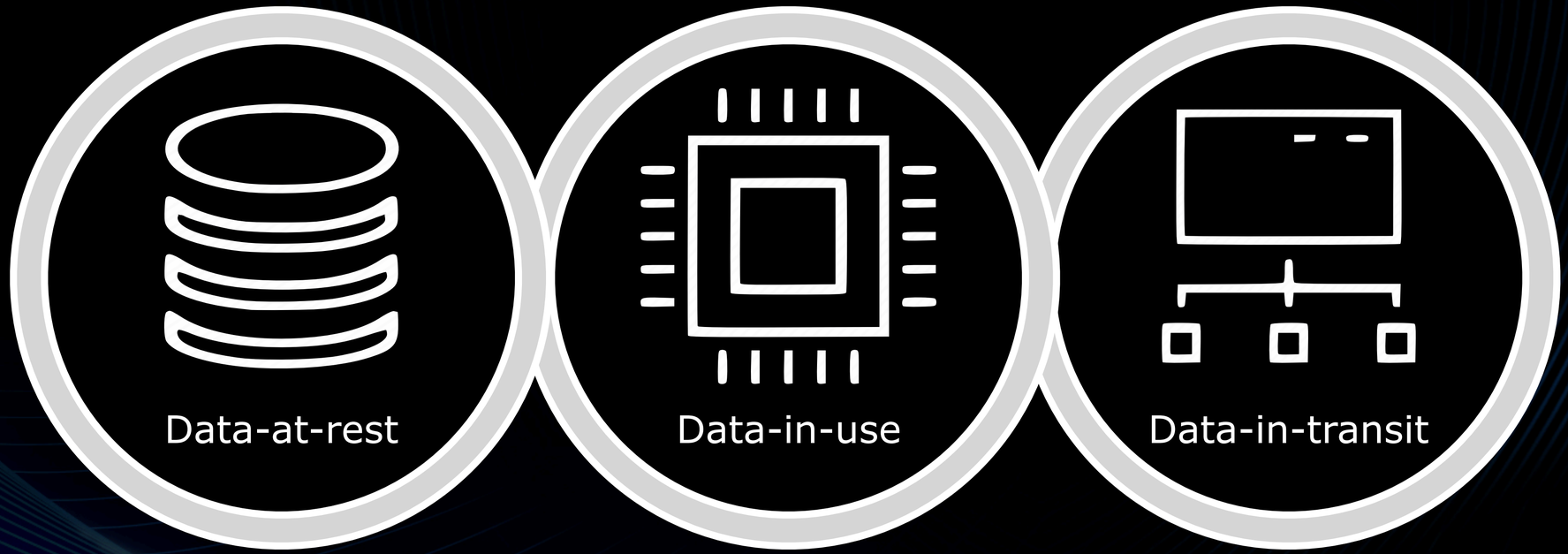
The framework announced

Sep. 2018

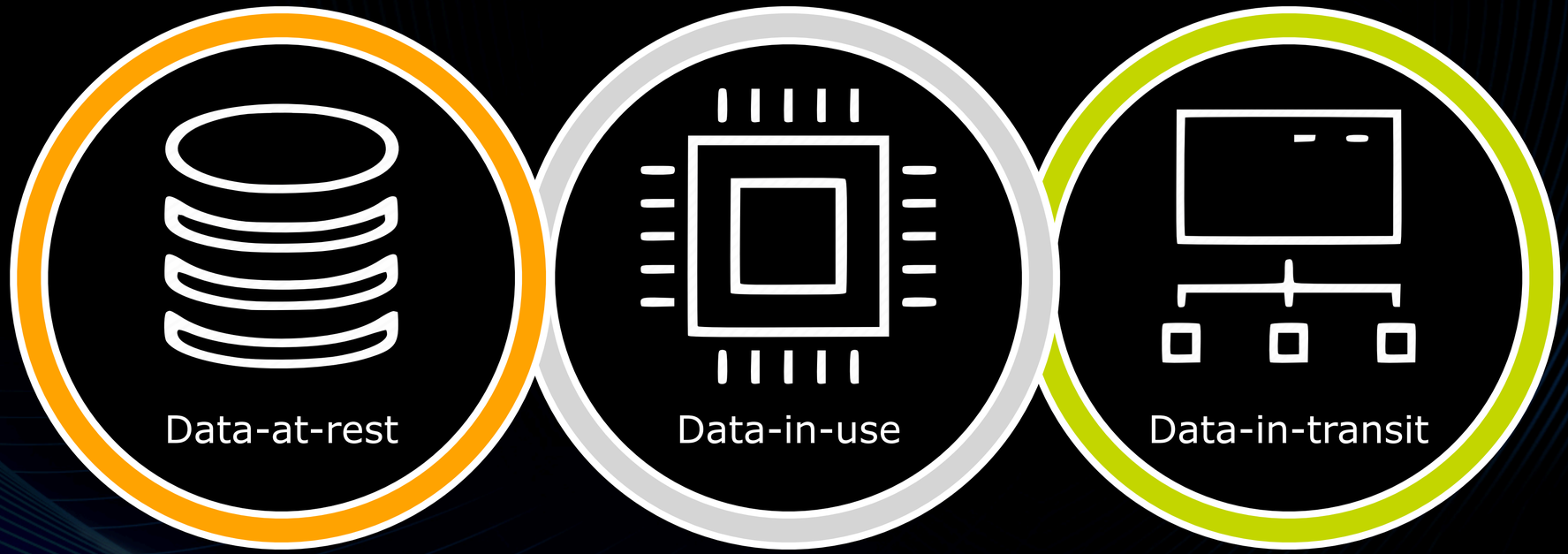
Baidu MesaTEE

The framework announced

What is Confidential Computing?



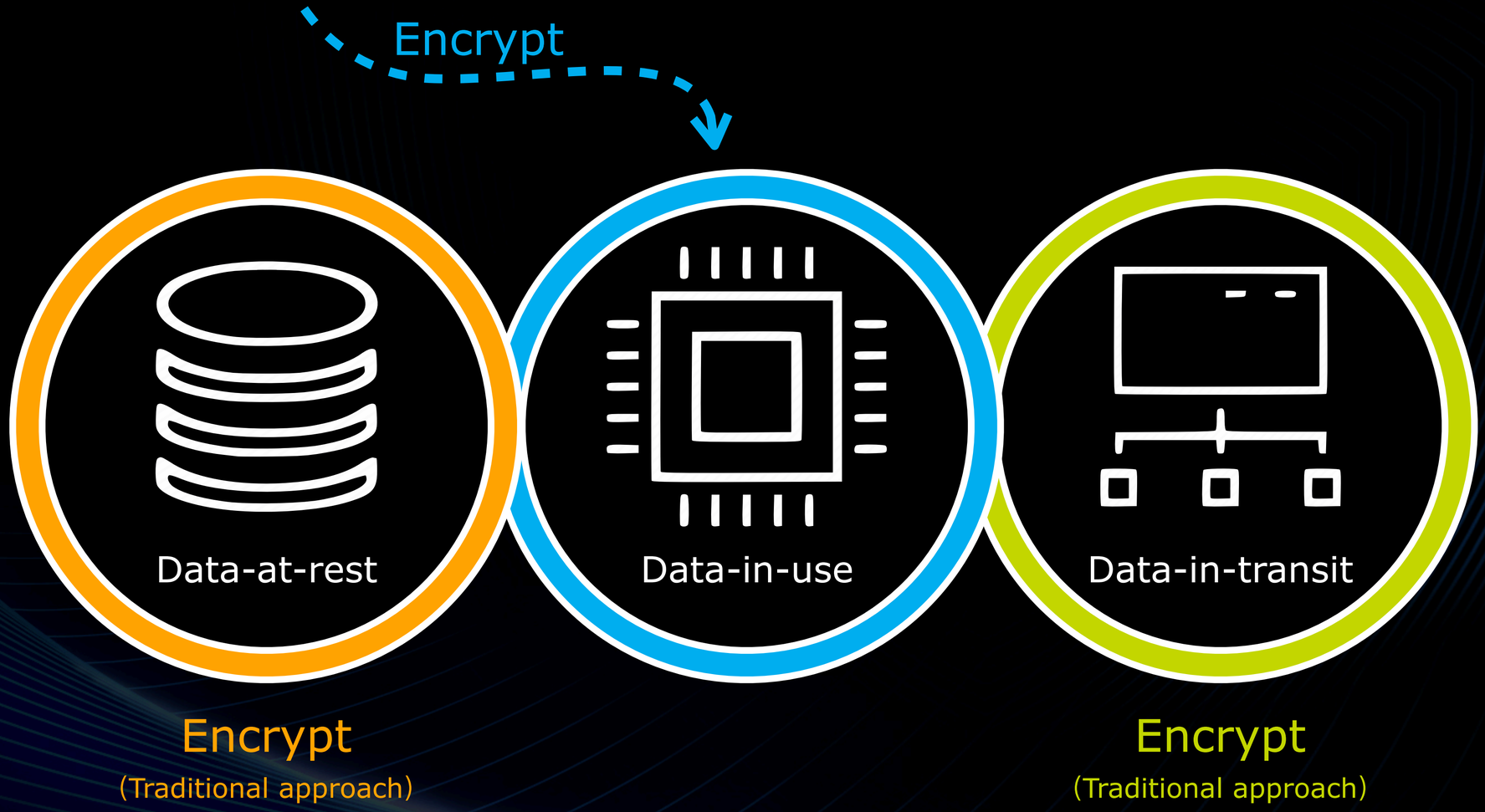
What is Confidential Computing?



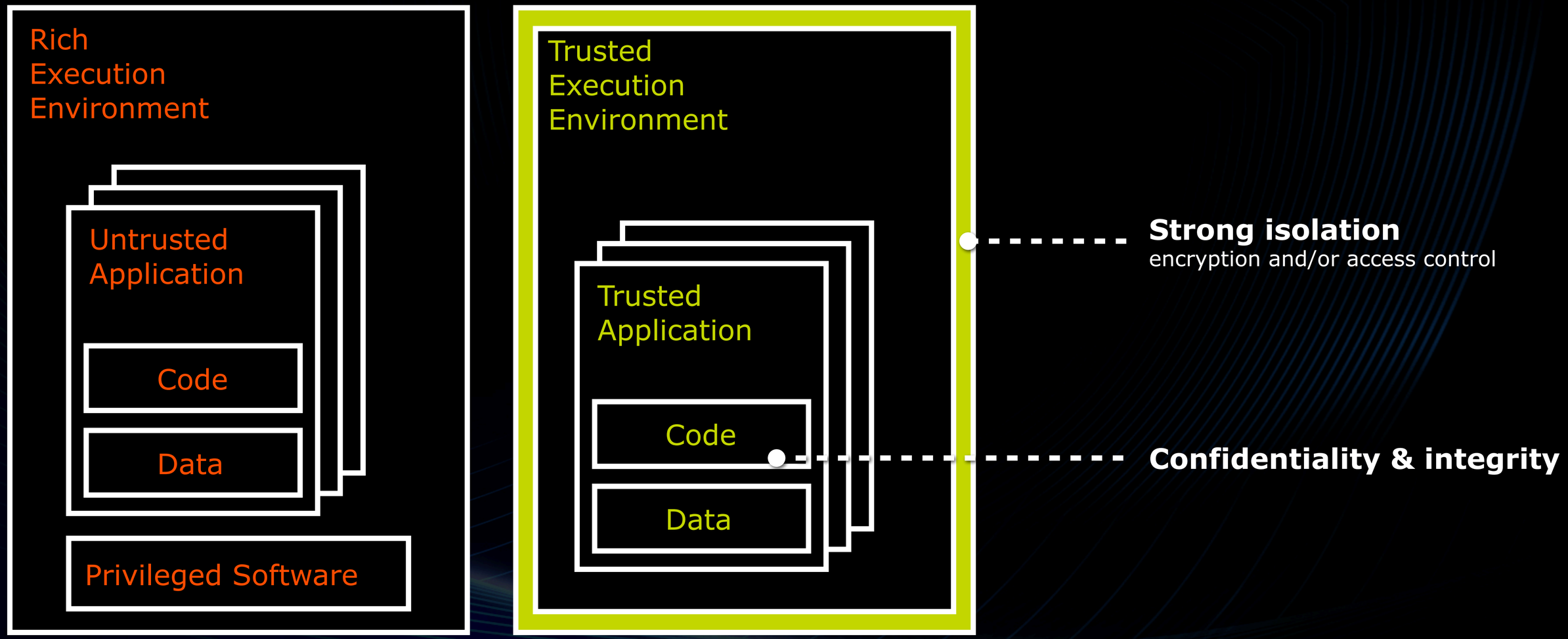
Encrypt
(Traditional approach)

Encrypt
(Traditional approach)

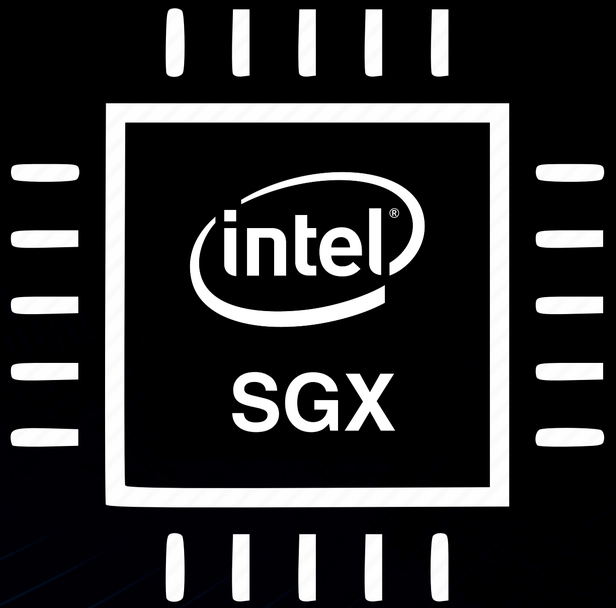
What is Confidential Computing?



Confidential Computing via *Trusted Execution Environments (TEEs)*



The *most advanced* TEE suitable for cloud is ...



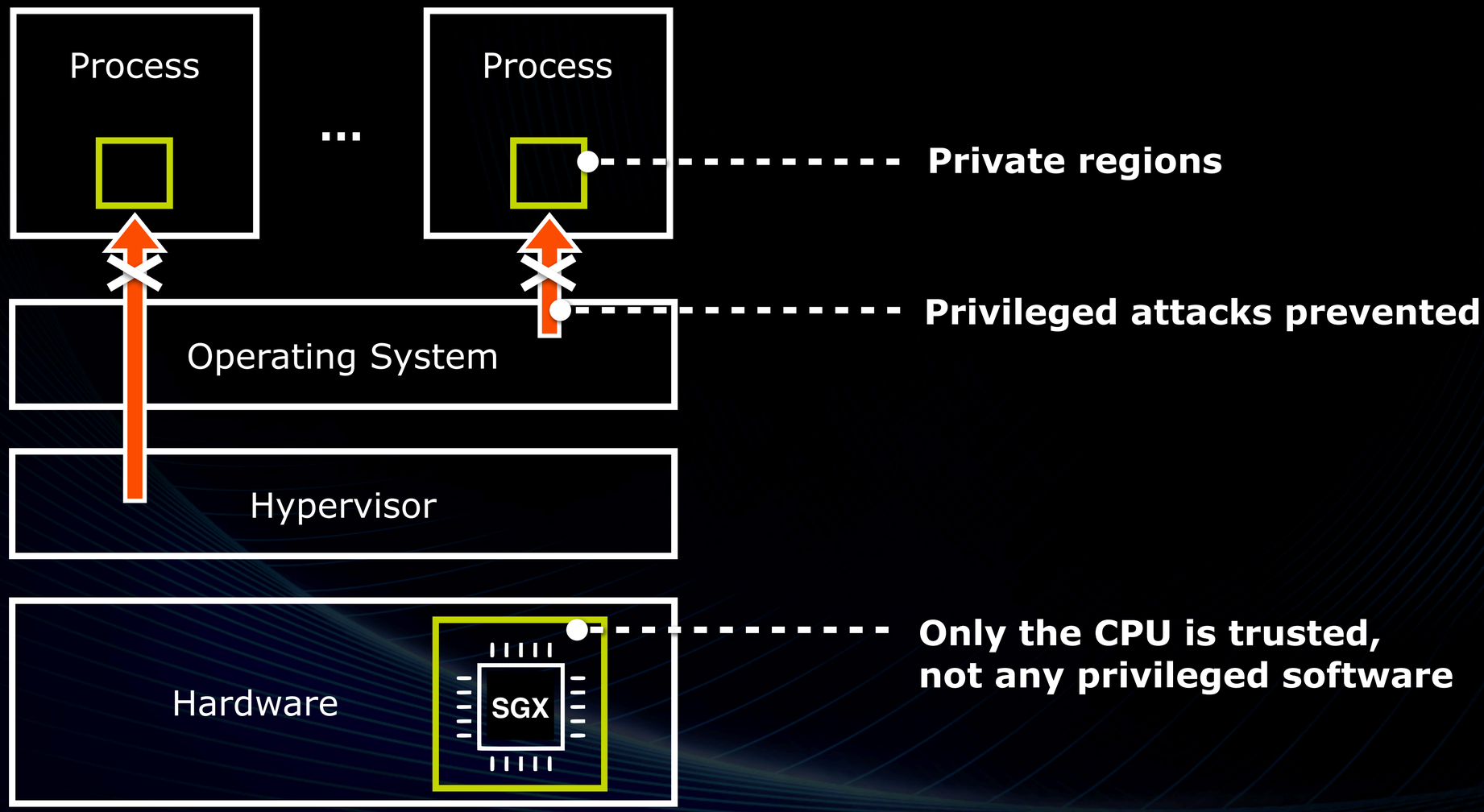
Agenda

- Intel Software Guard Extensions (SGX)
- An Overview of Project Occlumency
- Improving Security with Software Isolated Processes
- Improving Performance with Switchless Calls

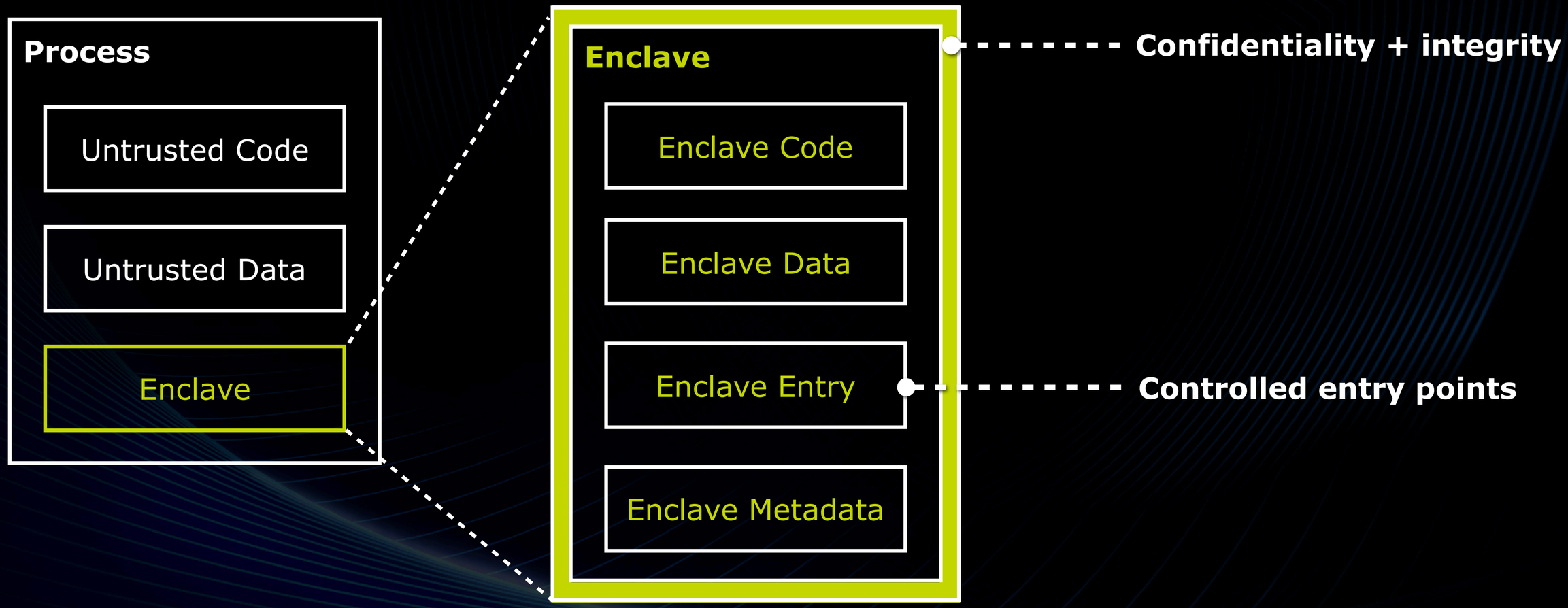
Agenda

- Intel Software Guard Extensions (SGX)
- An Overview of Project Occlumency
- Improving Security with Software Isolated Processes
- Improving Performance with Switchless Calls

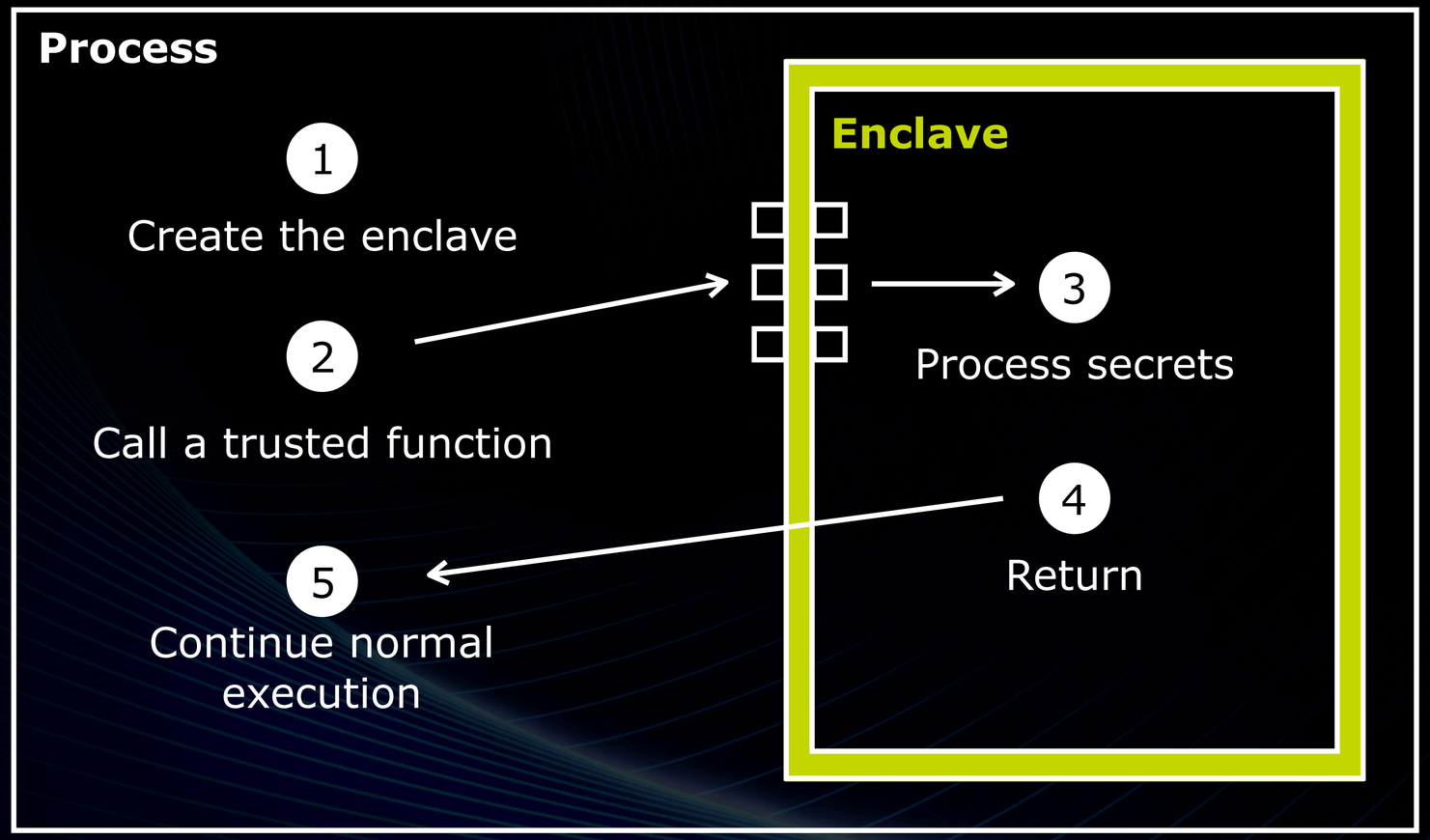
Intel Software Guard Extensions (SGX)



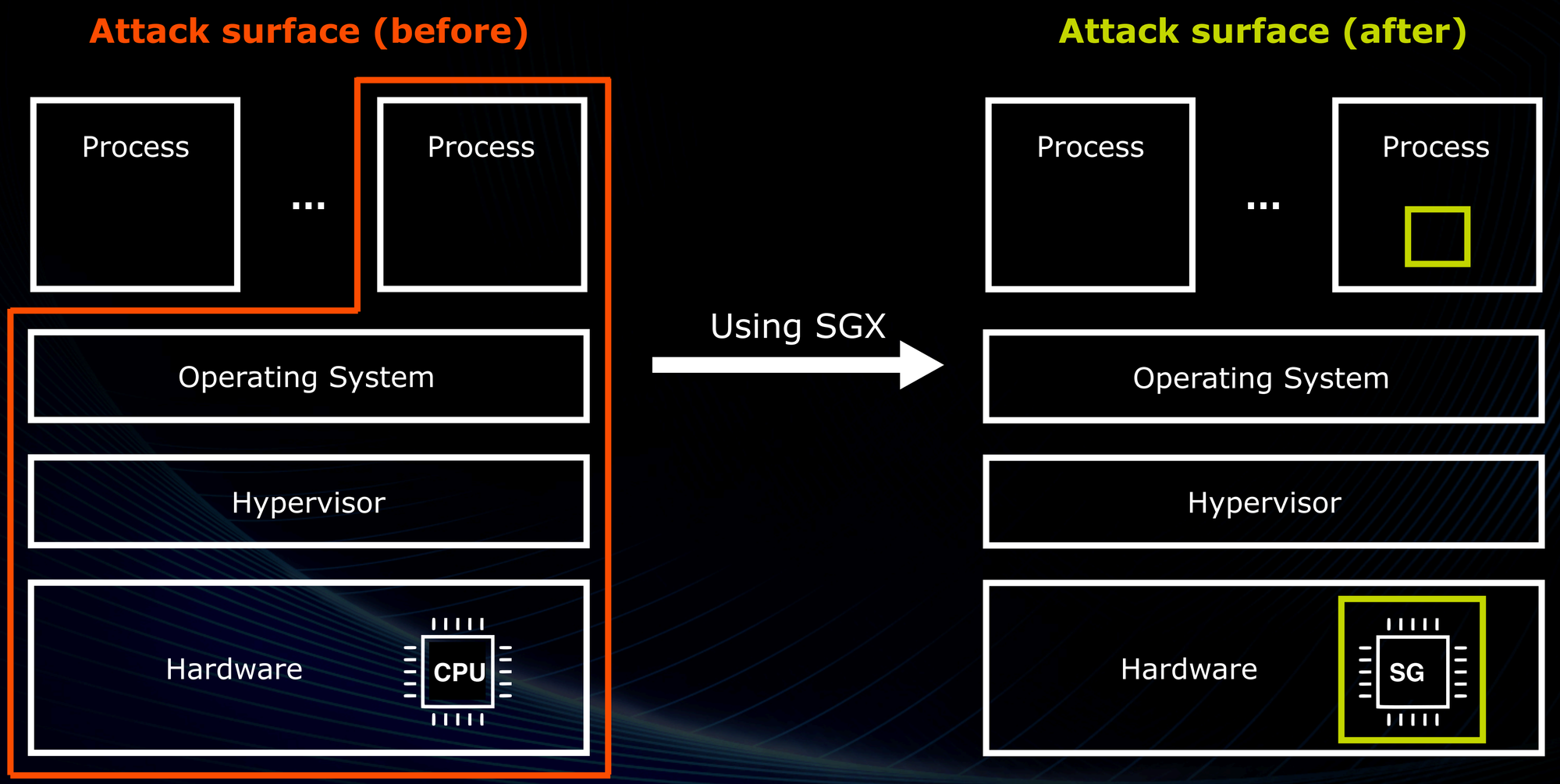
How SGX Apps are Programmed: The Memory Layout



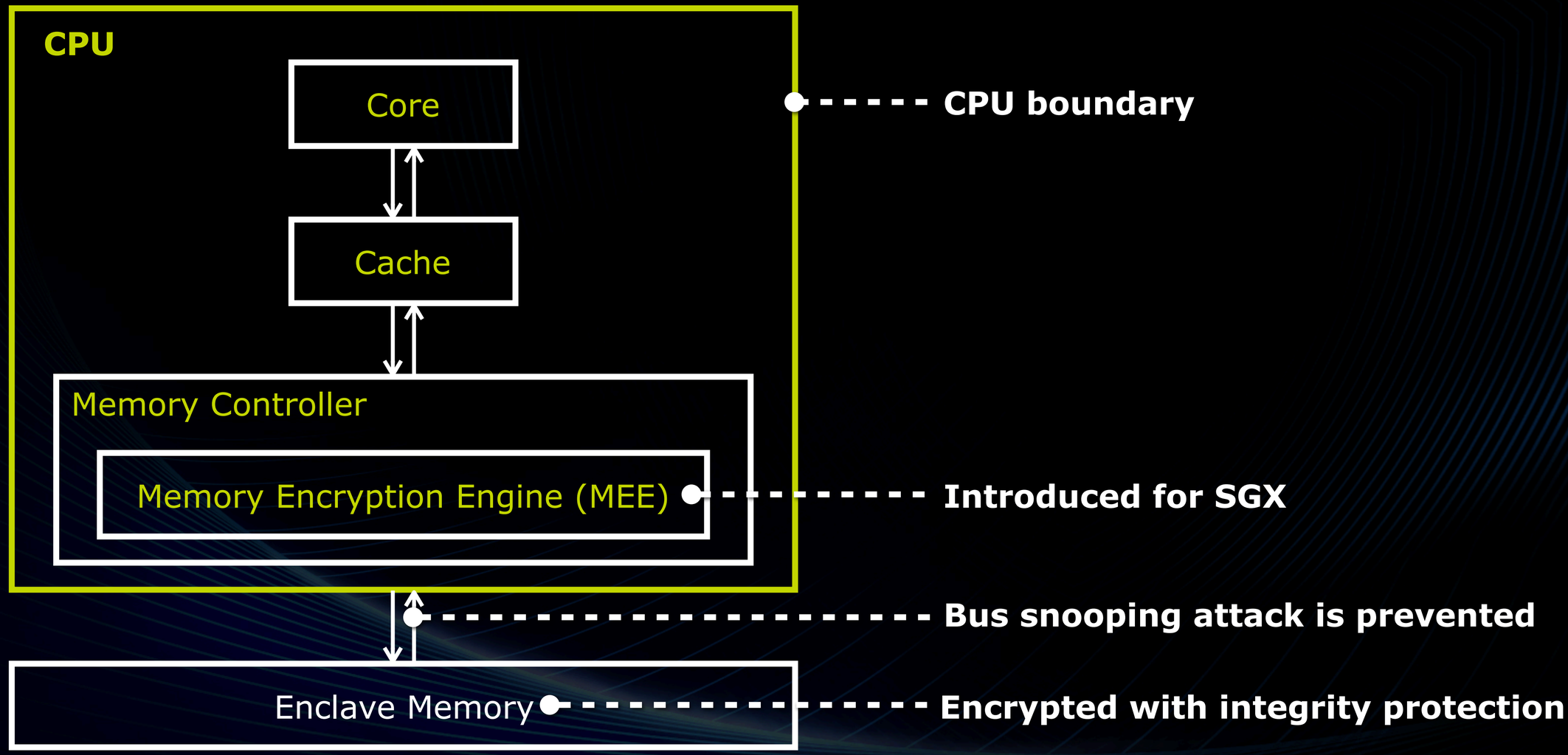
How SGX Apps are Programmed: The Workflow



How SGX Protects against Software Attacks



How SGX Protects against Hardware Attacks

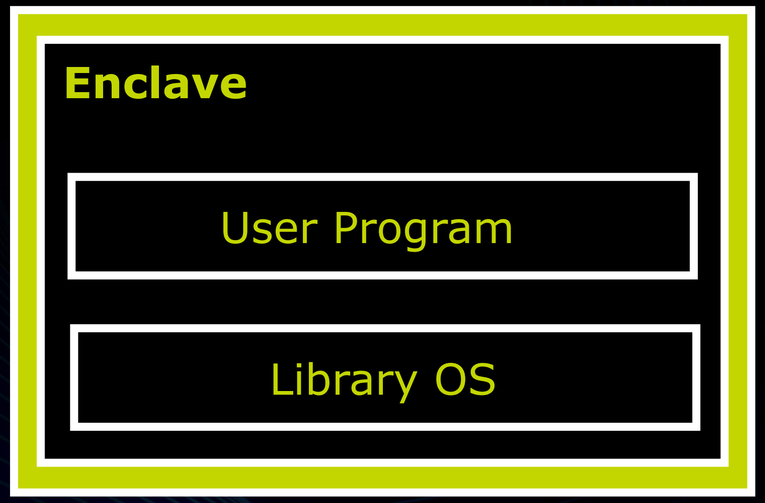


SGX Limitation: No Syscall inside Enclaves

- SGX forbids certain CPU instructions inside enclaves for security reasons, including
  - `IN, SYSENTER, SYSCALL, etc.`
- Due to no syscalls, the default libc shipped with Intel SGX SDK is incomplete
- Hard to develop new applications
- Hard to protect existing applications with SGX

So… Library OSes are Introduced into SGX

- A library OS is linked as a library with the target program, running in the same address space



- Library OSes enable running *unmodified* programs inside enclaves

Agenda

- Intel Software Guard Extensions (SGX)
- An Overview of Project Occlumency
- Improving Security with Software Isolated Processes
- Improving Performance with Switchless Calls

“The magical defence of the mind against external penetration.

An obscure branch of magic, but a highly useful one.”

— The definition of **Occlumency**
from *Harry Porter and the Order of the Phoenix*

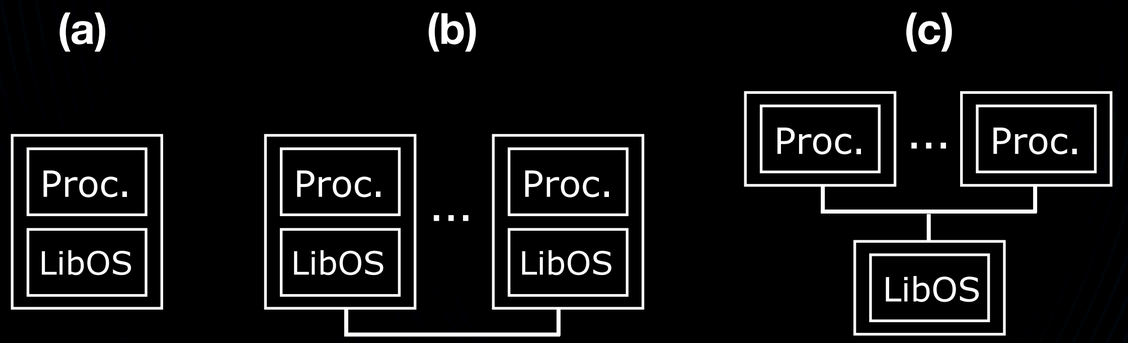
Project Occlumency

A research project that aims at building

a *memory-safe*, *multi-process* library OS for Intel SGX

The State-of-the-Art Library OSes: The *Multi-Process Support*

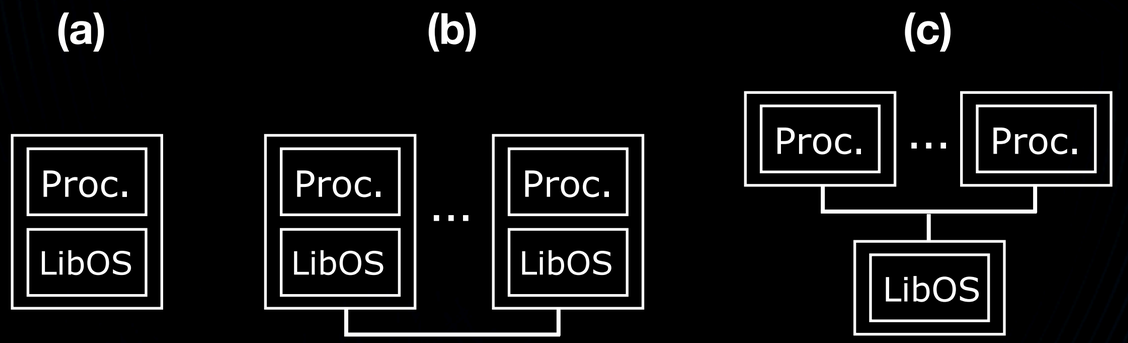
Architectures of Library OSes



Examples	Scone	Graphene-SGX	OBLIVIATE
# of processes : # of enclaves	1 : 1	N : N	N : N + 1
Process creation	No	Slow	Slow
Inter-process communication	No	Slow	Slow
Shared file systems (e.g., read)	No	No	Slow
***Shared* libraries**	No	No	No

The State-of-the-Art Library OSes: The *Memory Safety*

Architectures of Library OSes



Examples	Scone	Graphene-SGX	OBLIVIATE
# of processes : # of enclaves	1 : 1	N : N	N : N + 1
Inter-process isolation	No	Yes	Yes
User-kernel isolation	No	No	Yes
Out-of-enclave bug mitigation	No	No	No
Kernel memory safety	No	No	No

Out-of-Enclave Bugs are *Unintended* Memory Access to Outside the Enclave

```c
// file: enclave_with_out_of_enclave_bugs.c
#include <string.h>
#include <sgx_trts.h>  // Import APIs from Intel SGX SDK
#include "enclave_t.h" // Import ECall/OCall declaration

// A struct for storing and processing secrets
struct secret_struct
{
    void (*func)(const char* str);
    char data[64];
};

// This pointer should have been initialized somewhere but is not
struct secret_struct* secret; // = NULL, as it is not initialized
// Note: most commonly, address 0 is not inside an enclave

// A buggy ECall that consists of two out-of-enclave bugs
void ecall_process_secret_sloppily(void) {
    // Bug 1: unintented, out-of-enclave memory writes
    char secret_bytes[64];
    sgx_read_rand((unsigned char*)secret_bytes, 64);
    memcpy(&secret->data[0], secret_bytes, 64); // => data leakage!!!

    // Bug 2: unintented, out-of-enclave memory reads
    const char* secret_str = "Top Secret!!!\n";
    secret->func(secret_str); // => control hijhack!!!
}
```

Project Occlumency

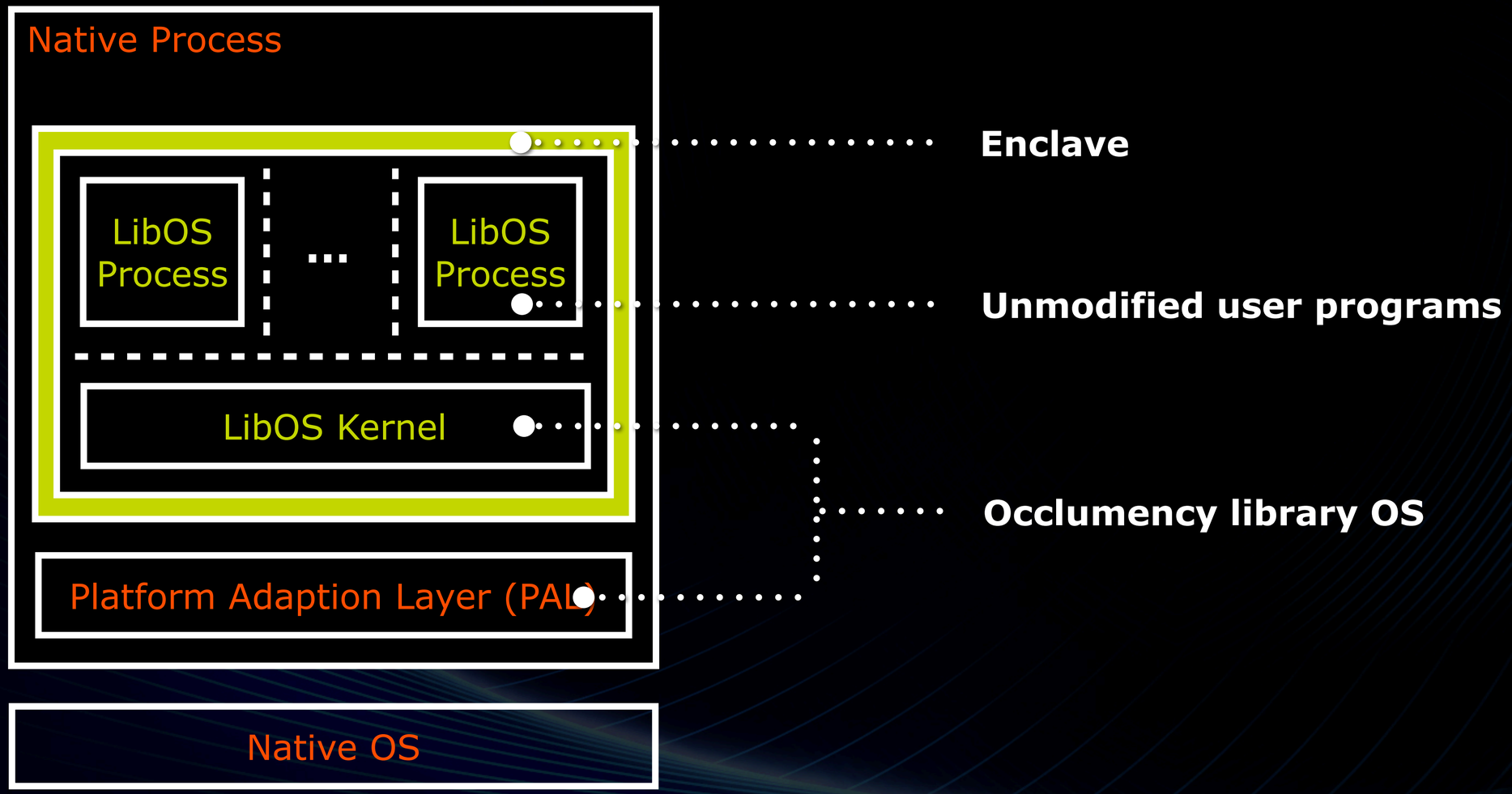
Occlumency is a *memory-safe*, *multi-process* library OS for Intel SGX

The Single-Address-Space Approach

Occlumency is a *single-address-space* library OS;

it runs all user-level processes and the library OS inside a single enclave.

Architecture Overview



Salient Features: Efficient Multi-Process Support

- Why multi-process support is important?
  - Almost all non-trivial service consists of more than one process
  - “Rule of Composition: Design programs to be connected with other programs.”
    — The Art of Unix Programming

- Low-cost process creation
- Efficient Inter-Process Communication (IPC)
- Shared (encrypted) file systems
- Shared libraries

Salient Features: Memory-Safety by Design

- Why memory-safety is important?
  - The memory-safety of C/C++ is largely resolved
  - Google syzkaller project found 600+ memory bugs in Linux kernel

- Inter-process isolation
- User-kernel isolation
- Out-of-enclave bug mitigation
- LibOS memory safety

Salient Features: Memory-Safety by Design

- Why memory-safety is important?
  - The memory-safety of C/C++ is largely resolved
  - Google syzkaller project found 600+ memory bugs in Linux kernel

- Inter-process isolation
- User-kernel isolation
- Out-of-enclave bug mitigation
- LibOS memory safety ·····> **Rust programming language**

Salient Features: Memory-Safety by Design

- Why memory-safety is important?
  - The memory-safety of C/C++ is largely resolved
  - Google syzkaller project found 600+ memory bugs in Linux kernel

- Inter-process isolation
- User-kernel isolation → **?**
- Out-of-enclave bug mitigation
- LibOS memory safety → **Rust programming language**

Agenda

- Intel Software Guard Extensions (SGX)
- An Overview of Project Occlumency
- Improving Security with Software Isolated Processes
- Improving Performance with Switchless Calls

Software Isolated Processes (SIPs)

- Software-isolated processes (SIPs) are isolated by software means; this is in contrast with ordinary processes, which are isolated by hardware means.
- The concept is first pioneered in Singularity OS (by Microsoft, 2003-2010), and revived here in Occlumency LibOS
- Occlumency's SIP are different from Singularity's SIP:

	Singularity's SIP	**Occlumency's SIP**
Motivation	Achieving a low-cost isolation	Enabling isolation inside an enclave
Means	A safe programming language	**Software Fault Isolation (SFI)**
Memory Safety	Fine-grained	Corse-grained
Programmers	Must write in Sing#	**Write in any language, including C/C+**

Our MPX-Based, Multi-Domain SFI (MMD-SFI)

- Software Fault Isolation (SFI)
  - A software-instrumentation technique that sandboxes untrusted modules (called **domains**) from the rest of the system
  - E.g., NativeClient, which sandboxes native plugins in Chrome browser
- We propose a ***MPX-based, Multi-Domain* SFI** that enables SIPs
  - **Data domains** and **code domains**

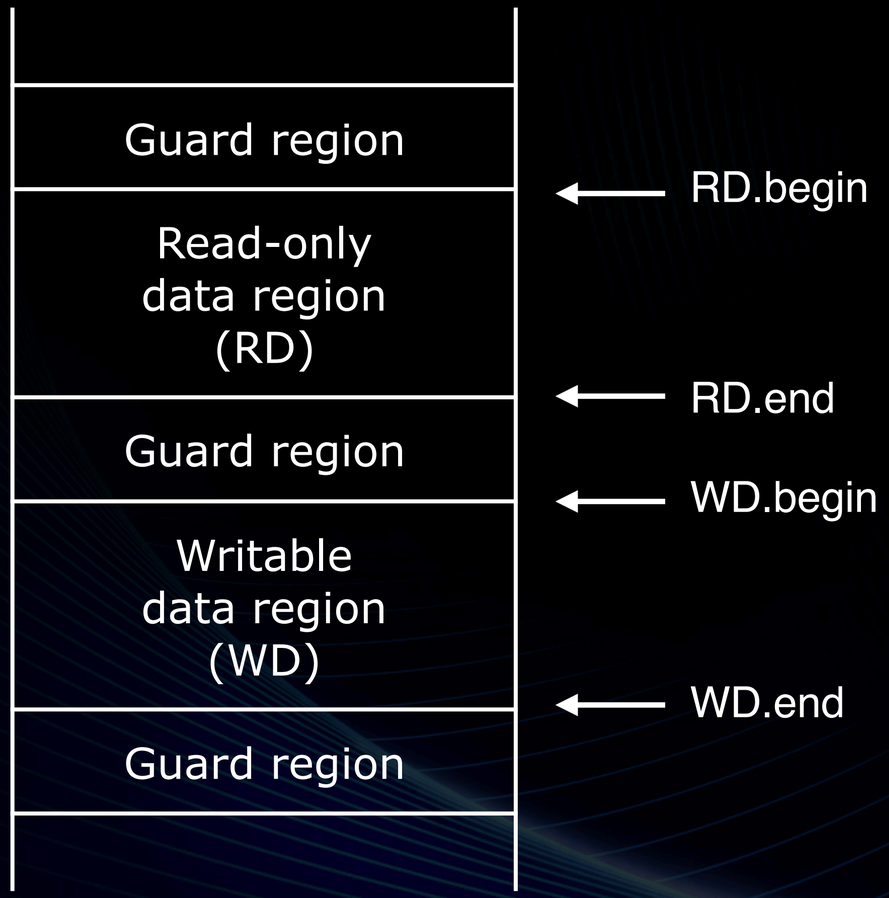
Repurposing Intel MPX for Software Fault Isolation

- Intel Memory Protection Extensions (MPX)
  - Four bound registers: `bnd0 – bnd3`
  - Bound init instructions: `bndmk`
  - Bound checking instructions: `bndcl, bndcu`
  - Bound loading instruction: `bndldx`
- Originally intended for fine-grained memory safety

```
1  obj* a[10]
2  total = 0
3  a_b = bndmk a, a+79              ;; Make bounds [a, a+79]
4  for (i=0; i<M; i++):
5     ai = a + i
6     bndcl a_b, ai                 ;; Lower-bound check of a[i]
7     bndcu a_b, ai+7               ;; Upper-bound check of a[i]
8     objptr = load ai
9     objptr_b = bndldx ai          ;; Bounds for pointer at a[i]
10    lenptr = objptr + 100
11    bndcl objptr_b, lenptr        ;; Checks of obj.len
12    bndcu objptr_b, lenptr+3
13    len = load lenptr
14    total += len
```

- This code snippet is from *Intel MPX Explained*

Data Domains: Enforcing Data Access Constraints



A Data Domain

1. All addresses of memory *reads* must be within range

   [RD.begin - GSize, WD.end + GSize)

2. All addresses of memory *writes* must be within range

   [WD.begin - GSize, WD.end + GSize)

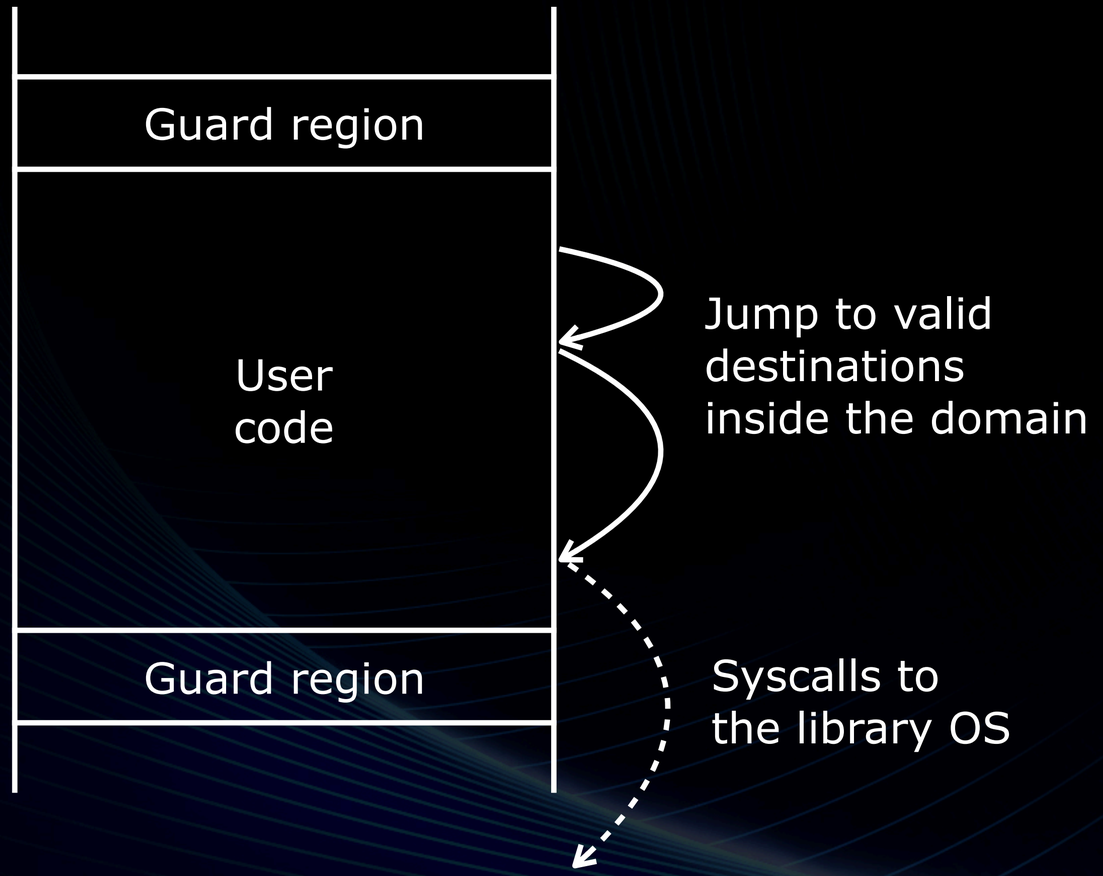
Data Domains: An Instrumentation Example

```
/* A memory write */

mov %rax, (%rcx)
```

Instrumented

```
/* A safe memory write */

bndcl %rcx, %bnd0

bndcu %rax, %bnd0

mov %rax, (%rcx)
```

Code Domains: Enforcing Control Flow Constraints



A Code Domain

1. The destination of any *direct* control-flow transfer is within the code domain or the syscall entry;
2. The destination of any *indirect* control-flow transfer is within the code domain;
3. Any destination of control-flow transfer is a valid instruction;
4. Any destination of control-flow transfer does not bypass instrumentation code.

Code Domains: Specifying Valid Destinations with `cfi_label`

- We introduce a pseudo-instruction `cfi_label`
- `cfi_label` has the following special properties

  1. Appears and only appears at valid destinations;

  2. No visible, architectural impact to CPU;

  3. Unique to a code domain.

- We choose a special 8-byte `nop` x86 instruction as `cfi_label`

Code Domains: An Instrumentation Example

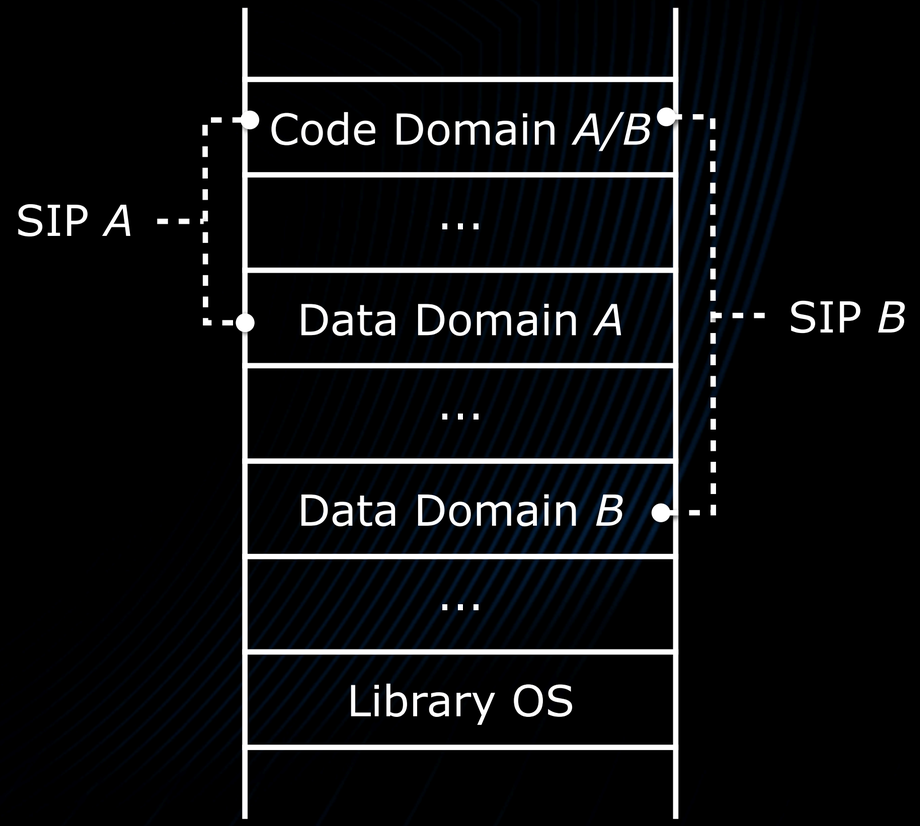
```
/* A function call */

call *%r15
```

Instrumented

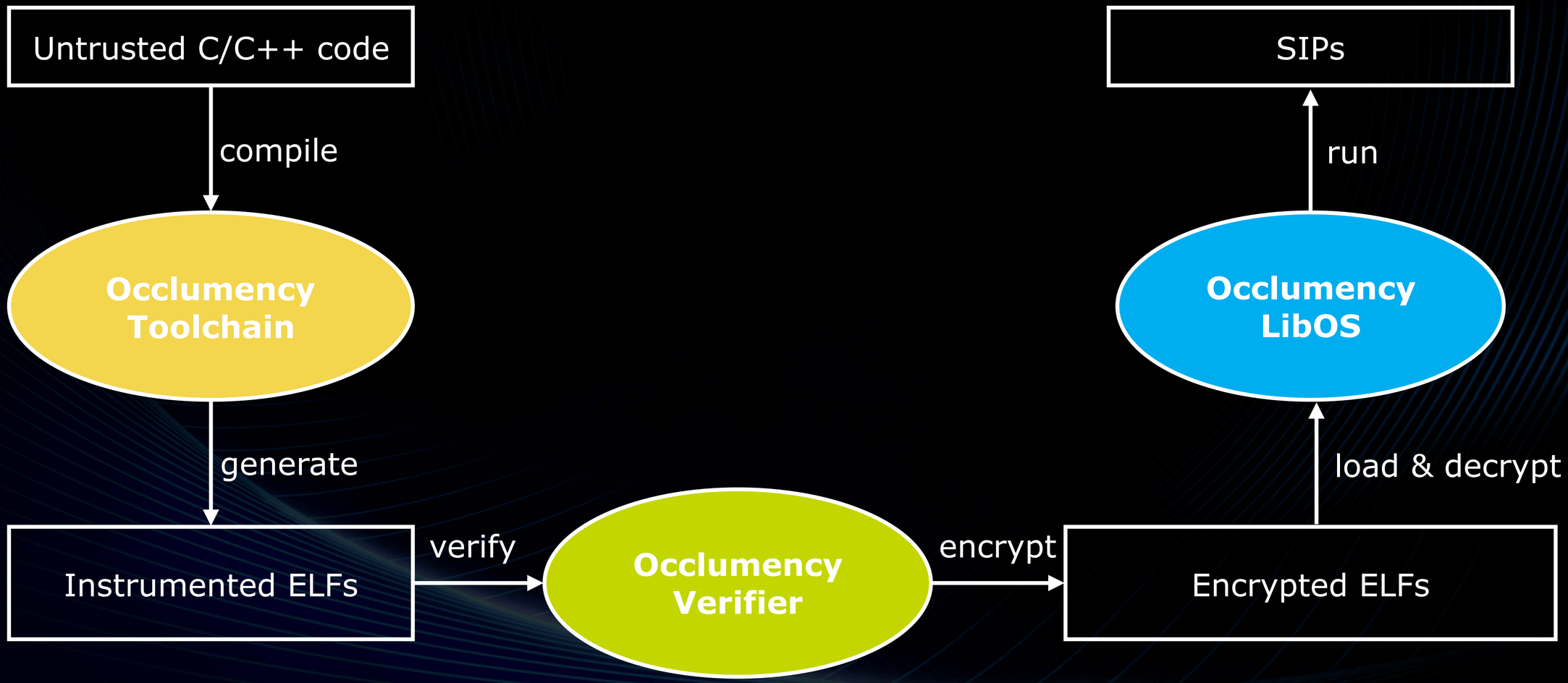
```
/* A safe function call */

mov (%r15), %rax

bndcl %rax, %bnd2

bndcu %rax, %bnd2

call *%r15

cfi_label
```

Code Domain + Data Domain = SIP

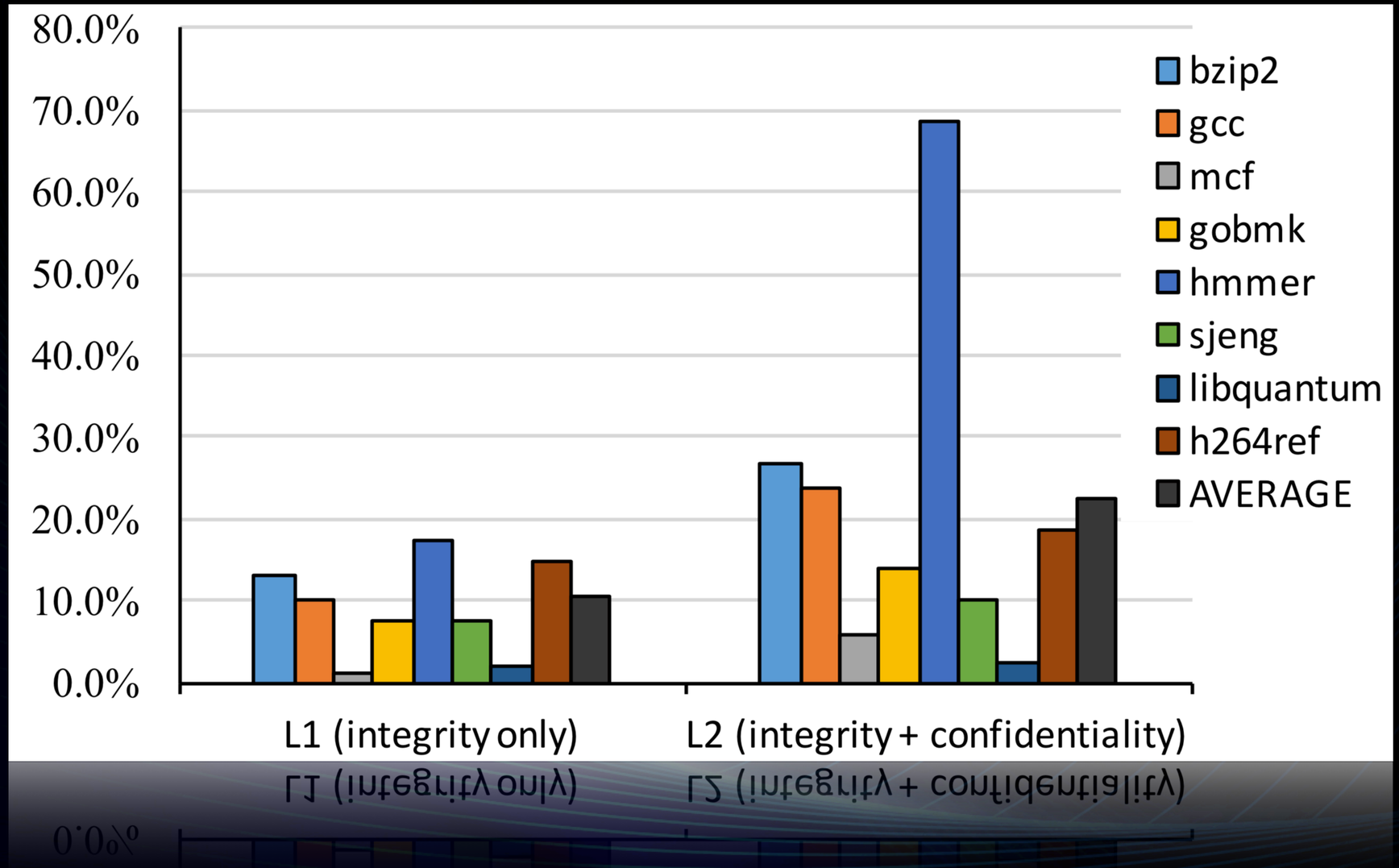
- A SIP has a code domain and a data domain for its code and data, respectively
- Sandboxed via code and data domains, SIPs automatically acquires the following security properties
  - Inter-process isolation
  - User-privileged isolation
  - Out-of-enclave-bug mitigation



From User Code to SIPs



Performance Overhead of MMD-SFI (SPECint2006)



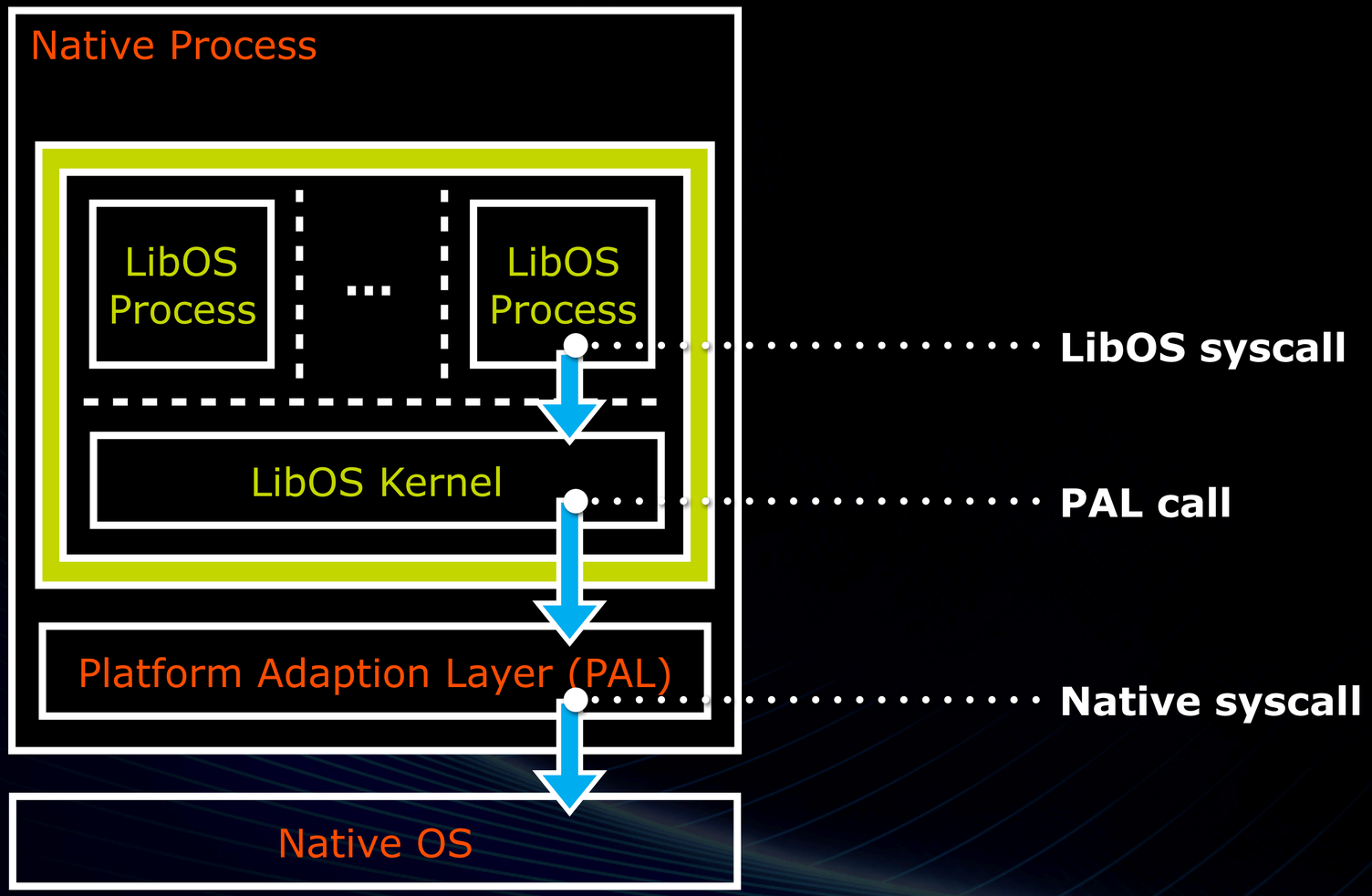
Section Review

- The single-address-space approach is based on SIPs
- Multiple SIPs are run inside a single enclave
- SIPs are sandboxed via a novel SFI scheme
- The runtime overhead of SFI is ~10% (integrity-only)

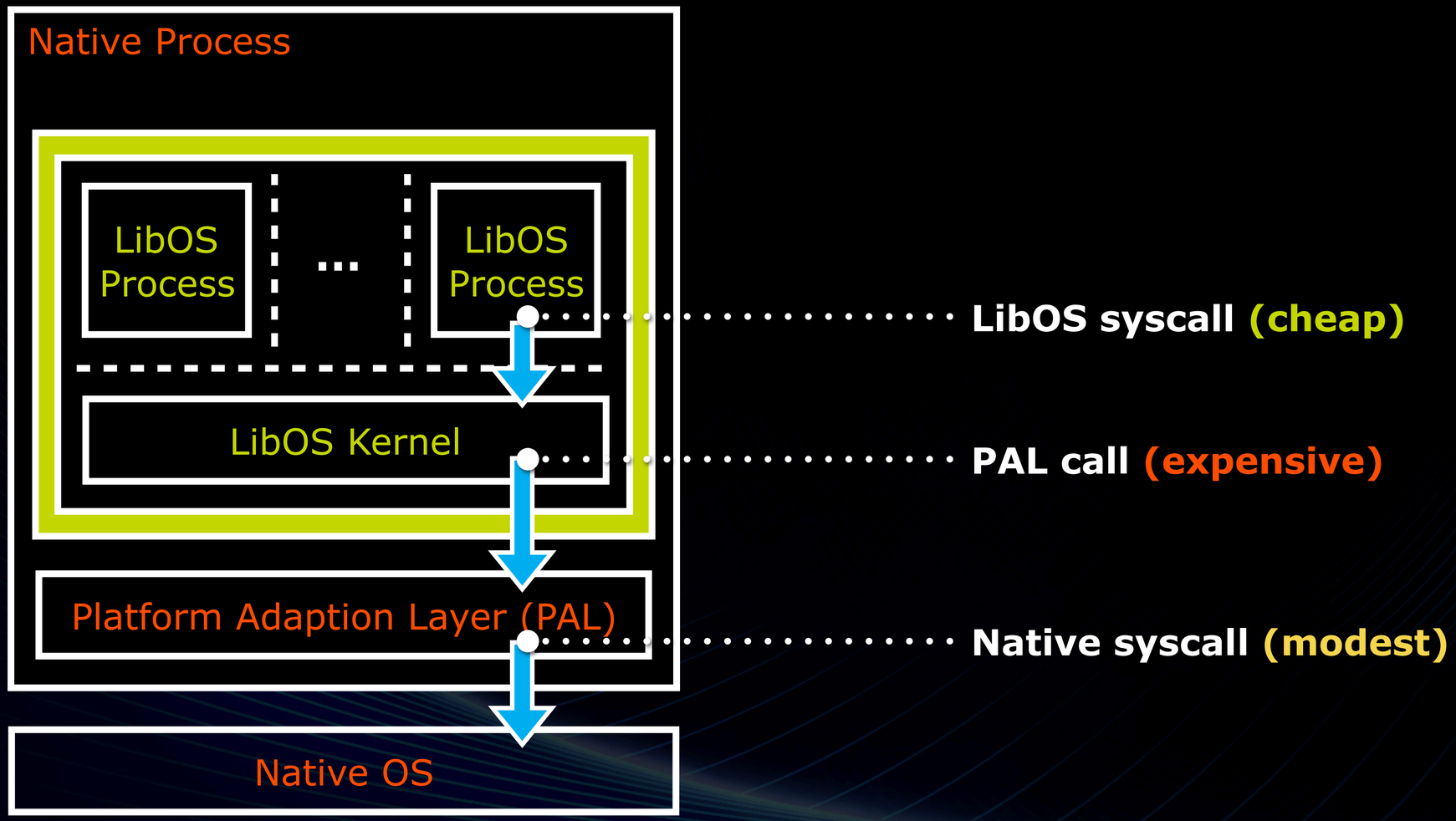
Agenda

- Intel Software Guard Extensions (SGX)
- An Overview of Project Occlumency
- Improving Security with Software Isolated Processes
- Improving Performance with Switchless Calls

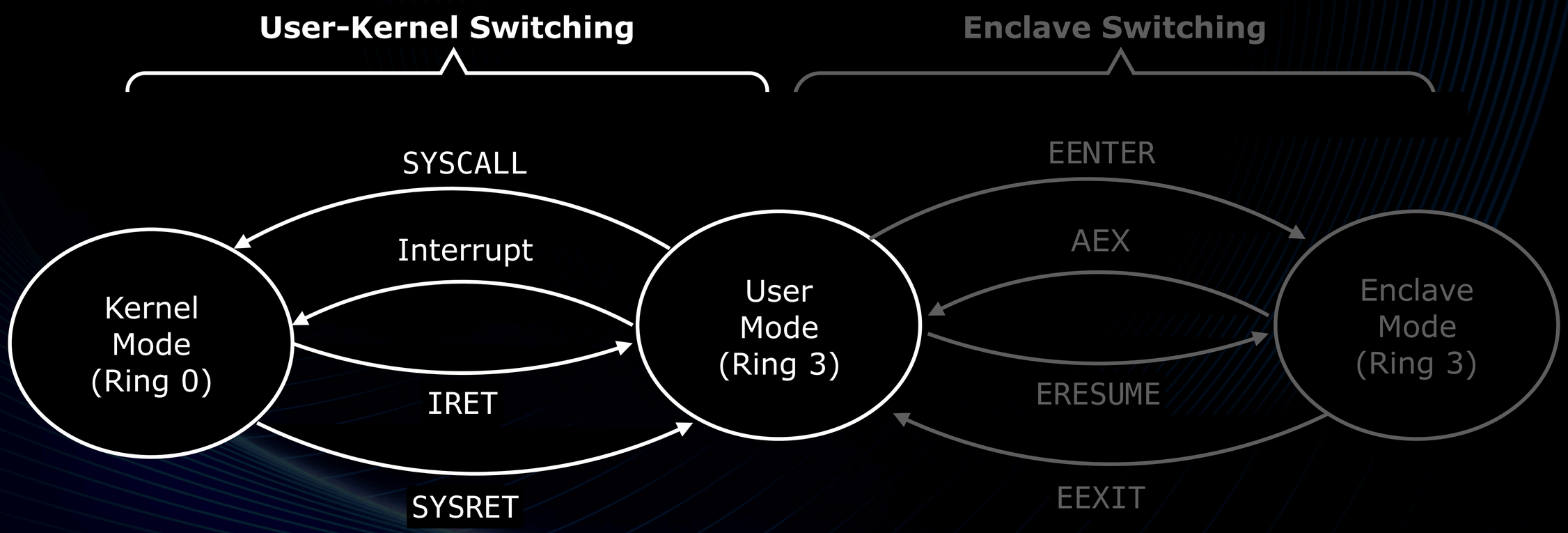
LibOS Syscall Has a Performance Bottleneck



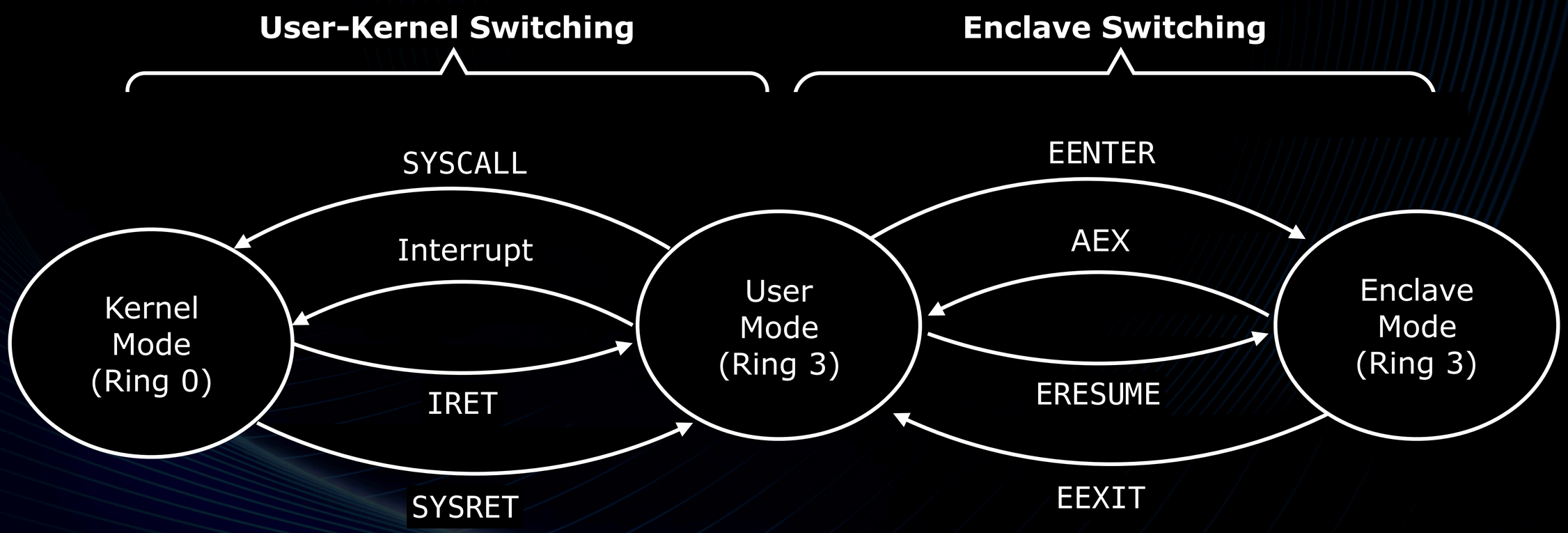
LibOS Syscall Has a Performance Bottleneck



What is *Enclave Switching*?

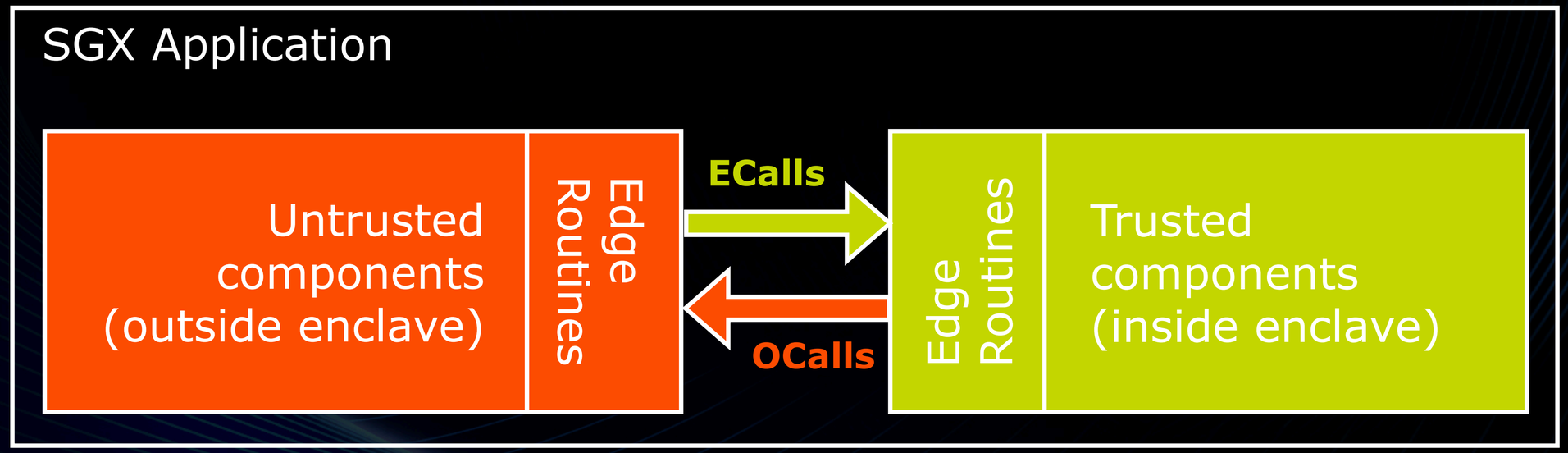


What is *Enclave Switching*?

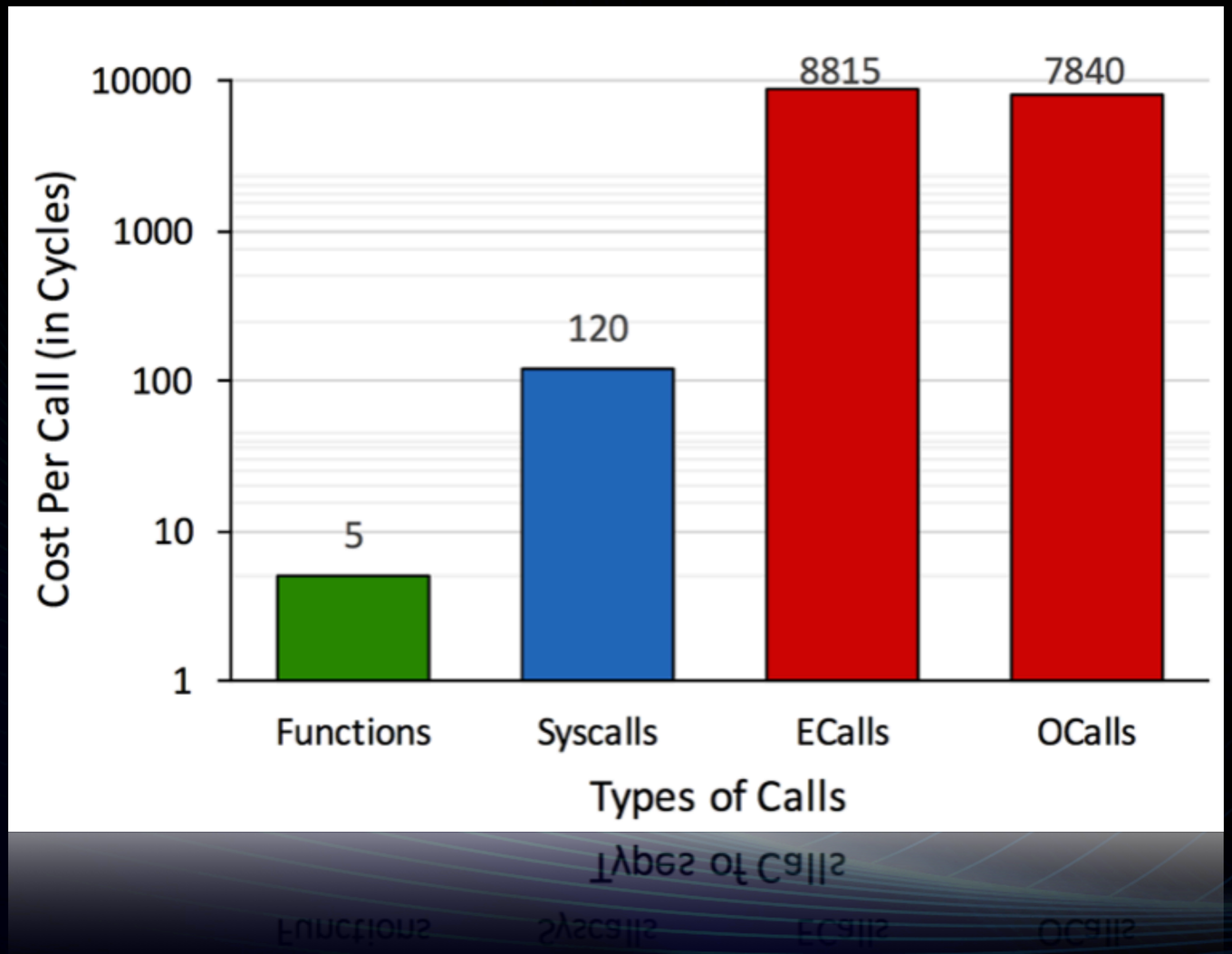


Enclave Switches are Triggered by ECalls/OCalls

- Intel SGX SDK provides a standard mechanism to do cross-enclave function calls: **ECalls** and **OCalls**

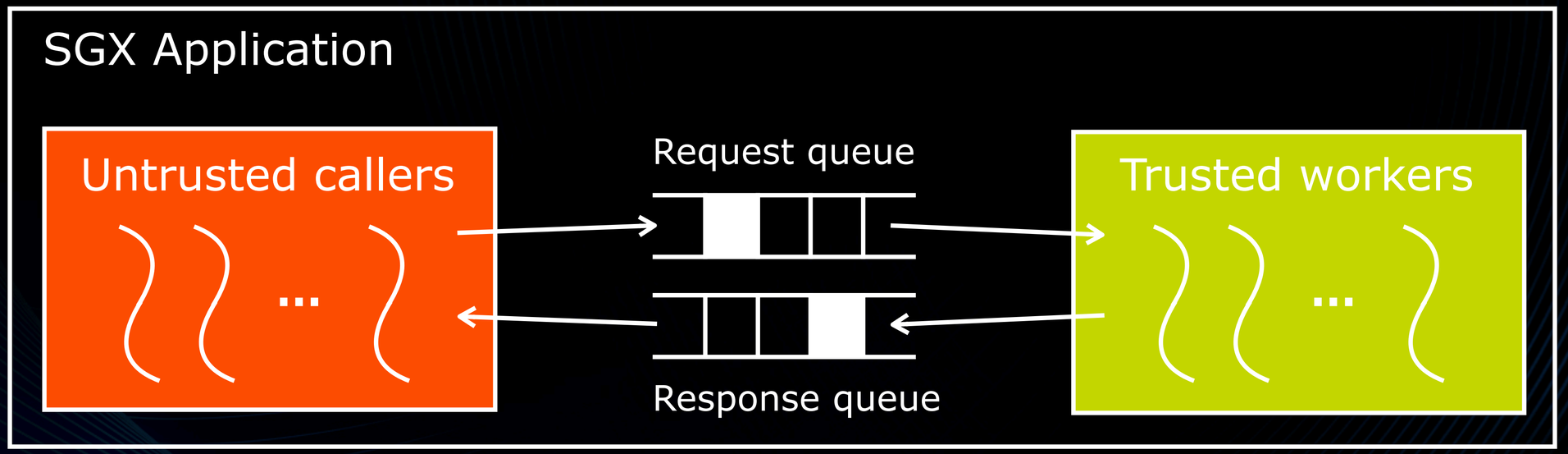


Enclave Switches are Expensive



The Solution: Switchless Calls

- The core idea is simple: **shared queue + worker threads + async execution**



How Switchless ECalls work

- Independently proposed in several works, e.g., SCONE, HotCall, SGXKernel

Many Practical Concerns that are Not Addressed Yet...

- What is the overall performance speedup?
- What about the extra CPU core used?
- How to determine the number of workers?
- How to handle real-world, dynamic workloads?
- When to wake up or sleep workers?

Our Contributions

- We establish a comprehensive performance model
- We propose a self-adaptive worker scheduling algorithm
- We integrate Switchless Calls seamlessly into Intel SGX SDK (from v2.2)

- **Any library OSes built upon Intel SGX SDK can benefit from our Switchless Call implementation**

Worker Efficiency

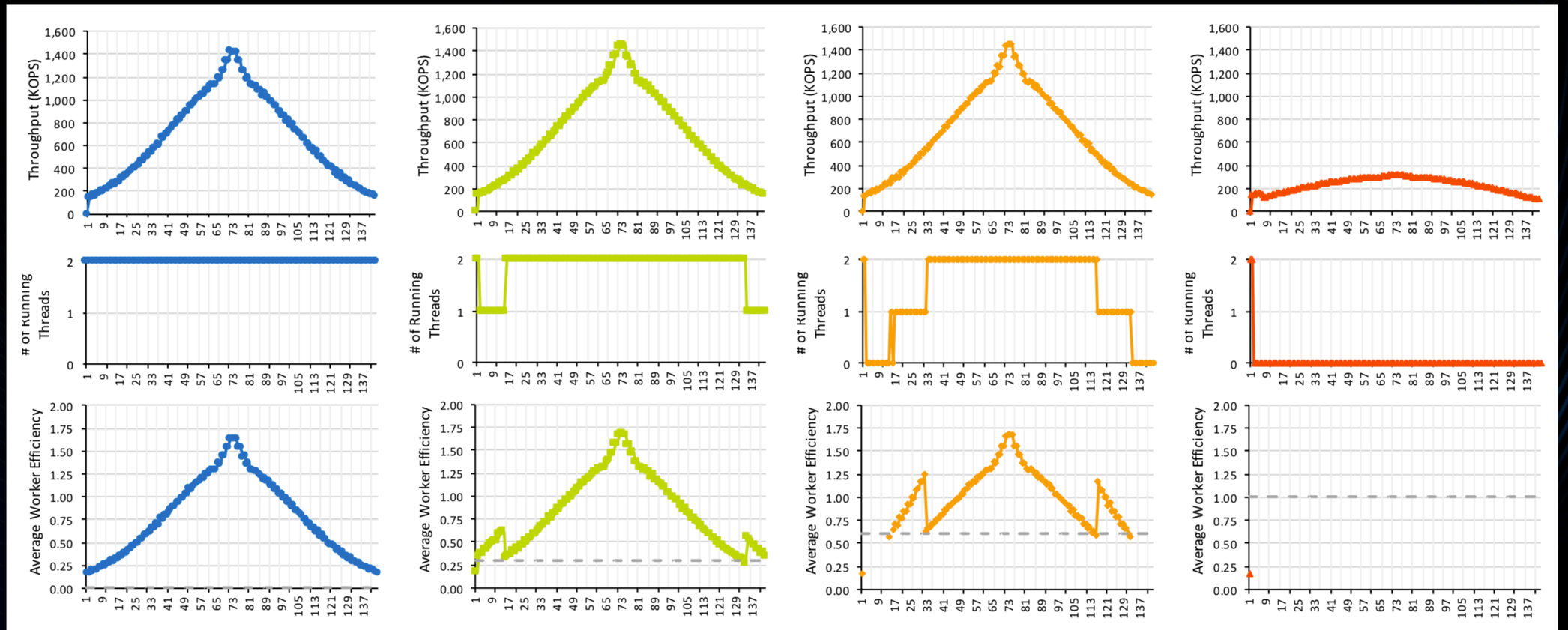
- During a given period of time, we define the efficiency of a worker

$$\epsilon = \frac{\text{The CPU time saved by the worker}}{\text{The CPU time consumed by the worker}} = \frac{X \cdot T_{es}}{T}$$

  where *X* is the number of Switchless Calls processed by the worker and *T* is the cost of an enclave switch.

- We prove that the performance speedup $p = 1 + \frac{N}{M}\epsilon$, where *N* is the number of worker threads and M is the number of caller threads

Performance Evaluation on Dynamic Workloads



(a) $e_{min} = 0.001$ (b) $e_{min} = 0.3$ (c) $e_{min} = 0.6$ (d) $e_{min} = 1.0$

Section Review

- Enclave switches is a major source of SGX performance overhead
- Switchless Calls is an effective technique to avoid triggering enclave switches
- Switchless Calls is now available in Intel SGX SDK from v2.2
- Switchless Calls can strike a good balance between performance and energy consumption

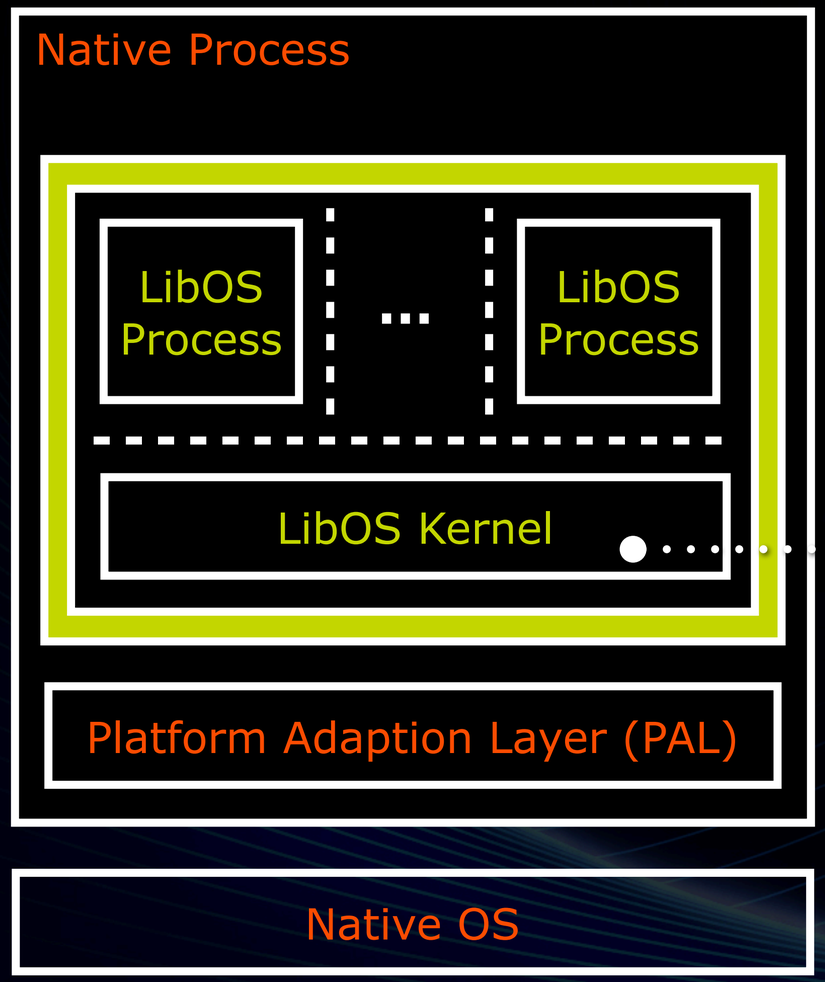
Agenda

- Intel Software Guard Extensions (SGX)
- An Overview of Project Occlumency
- Improving Security with Software Isolated Processes
- Improving Performance with Switchless Calls

Take Home Messages

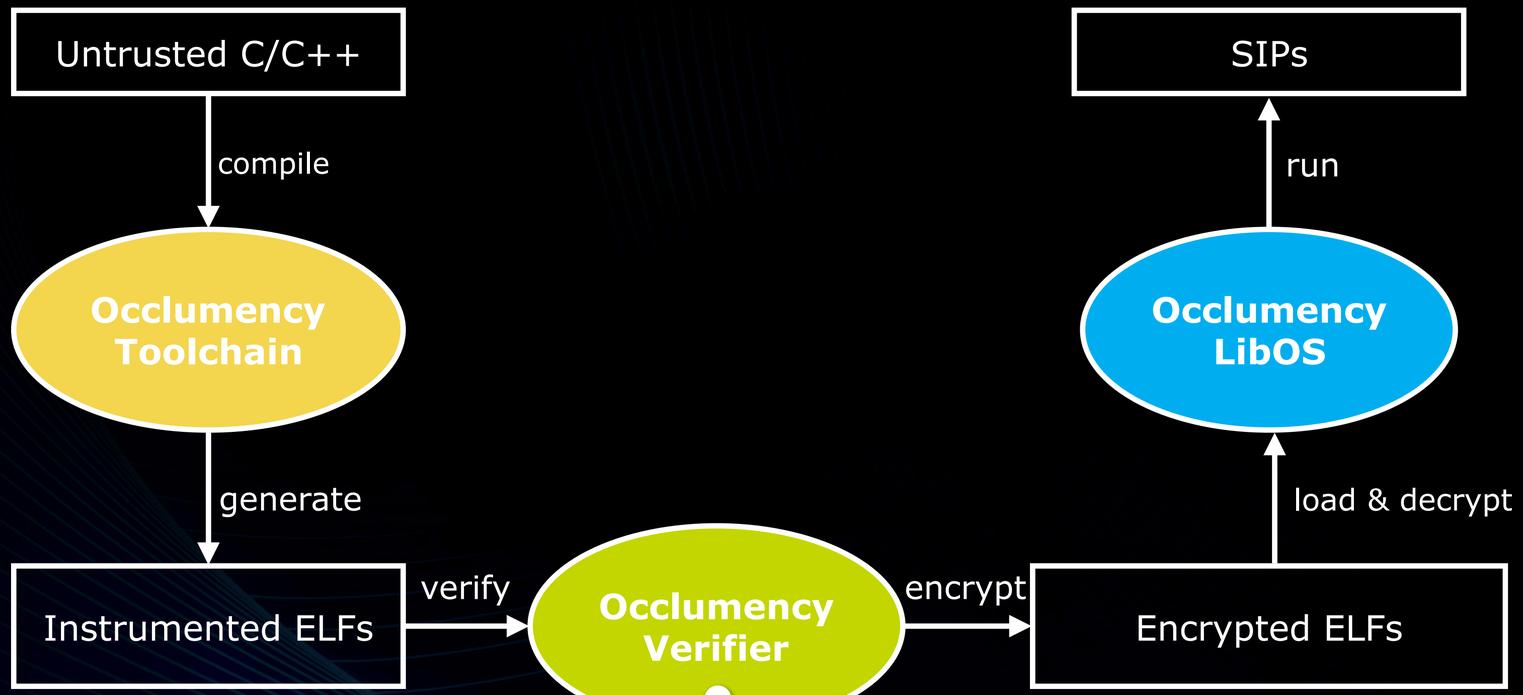
- Intel SGX enables confidential computing in cloud
- Occlumency project is a memory-safe, multi-process library OS for Intel SGX
- Software-isolated processes (SIPs) can be implemented with Software Fault Isolation (SFI)
- Switchless Calls is an effective technique to improve SGX performance

Future Work



We are implementing the library OS in *Rust* programming language

Future Work



We are designing and implementing the verifier

Let’s work together toward a *confidential cloud*