CCS'17 Tutorial:

SGX Shielding Frameworks and Development Tools

Chia-Che Tsai

Stony Brook University / UC Berkeley

Legal Notices & Disclaimer

- This presentation contains the general insights and opinions of Intel Corporation ("Intel"). The information in this presentation is provided for information only and is not to be relied upon for any other purpose than educational. Use at your own risk! Intel makes no representations or warranties regarding the accuracy or completeness of the information in this presentation. Intel accepts no duty to update this presentation based on more current information. Intel is not liable for any damages, direct or indirect, consequential or otherwise, that may arise, directly or indirectly, from the use or misuse of the information in this presentation.
- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.
- No computer system can be absolutely secure.
- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.
- Intel, the Intel Core, and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
- *Other names and brands may be claimed as the property of others.
-

Developing a SGX Application

- SDK model: build your own SGX applications
- Porting an existing application
  - Limitation 1: needs a signed, static image
  - Limitation 2: virtualized ISA (no CPUID/RDTSC)
  - Limitation 3: **no trusted OS services**
- Requires defenses against untrusted OSes

Choose Porting Strategy

- How much OS functionality is needed?
  - Little (e.g., crypto functions) → SDK
  - Medium (e.g., microservices) → Shielding layers
  - Heavy (e.g., language runtimes) → Library OSes
- Always ensure a secure enclave interface
- Performance is a critical factor

Topics

- Porting challenges and OS attack vectors
- Library OS: Graphene-SGX
- System interface shield layers: SCONE, Panoply
- Dynamic page management on SGX2
- Exit-less enclaves with Eleos

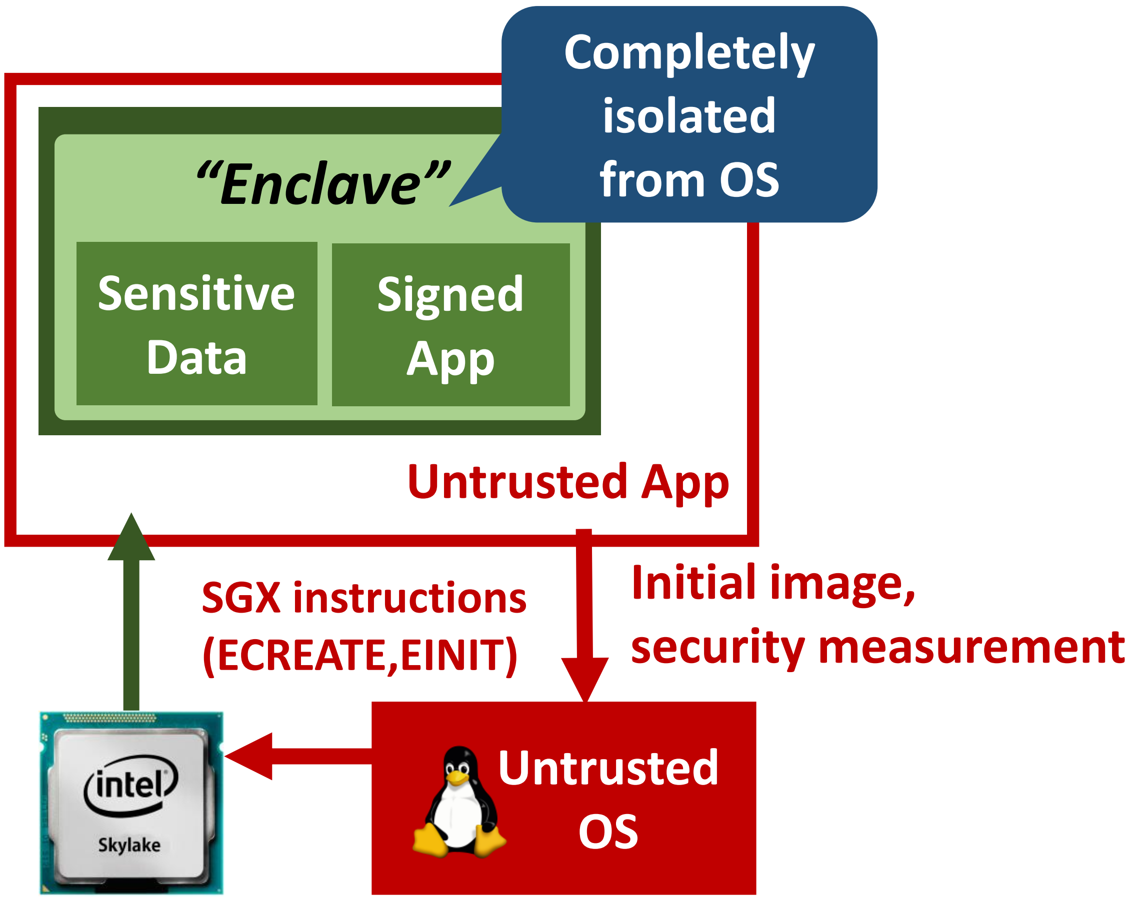
For Each Framework

- What are the target applications?
- What are the key concepts?
- What to expect? How to use?
- Where to obtain the software?

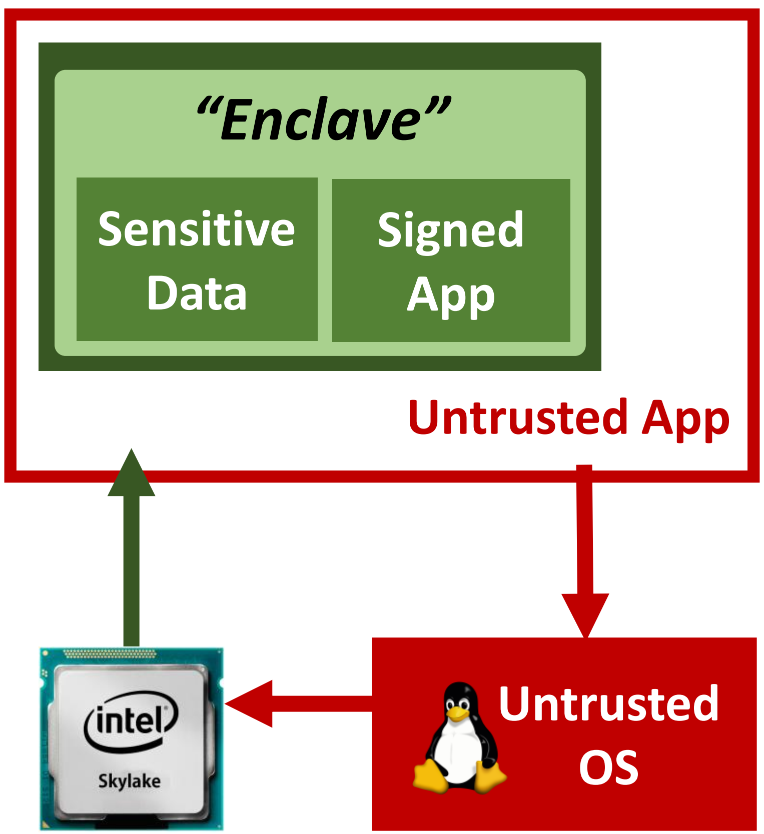
SGX Porting Challenges

- Satisfying enclave requirements
- Defending against untrusted OS services
- Improving performance factors

SGX Application Requirements



SGX Application Requirements



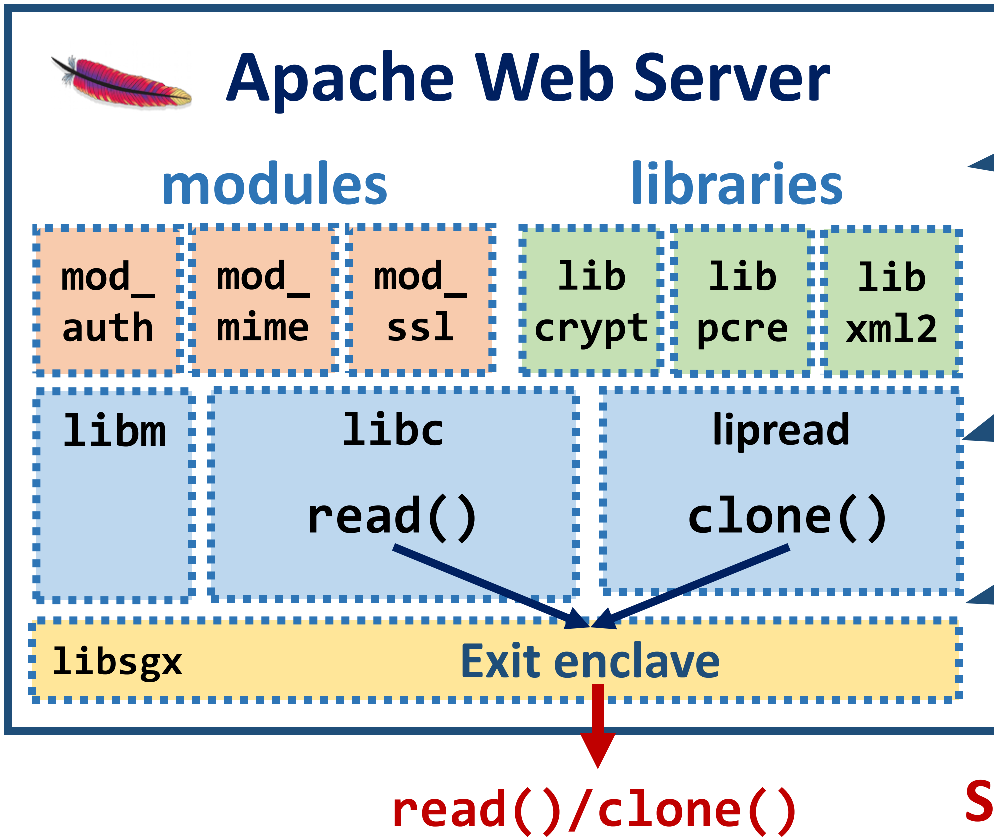
1. **Static initial image**
2. **No system calls**
3. **Check for untrusted inputs**

Most Linux applications:

(1) Dynamic linked

(2) Built-in syscall usage

Porting a Legacy Application



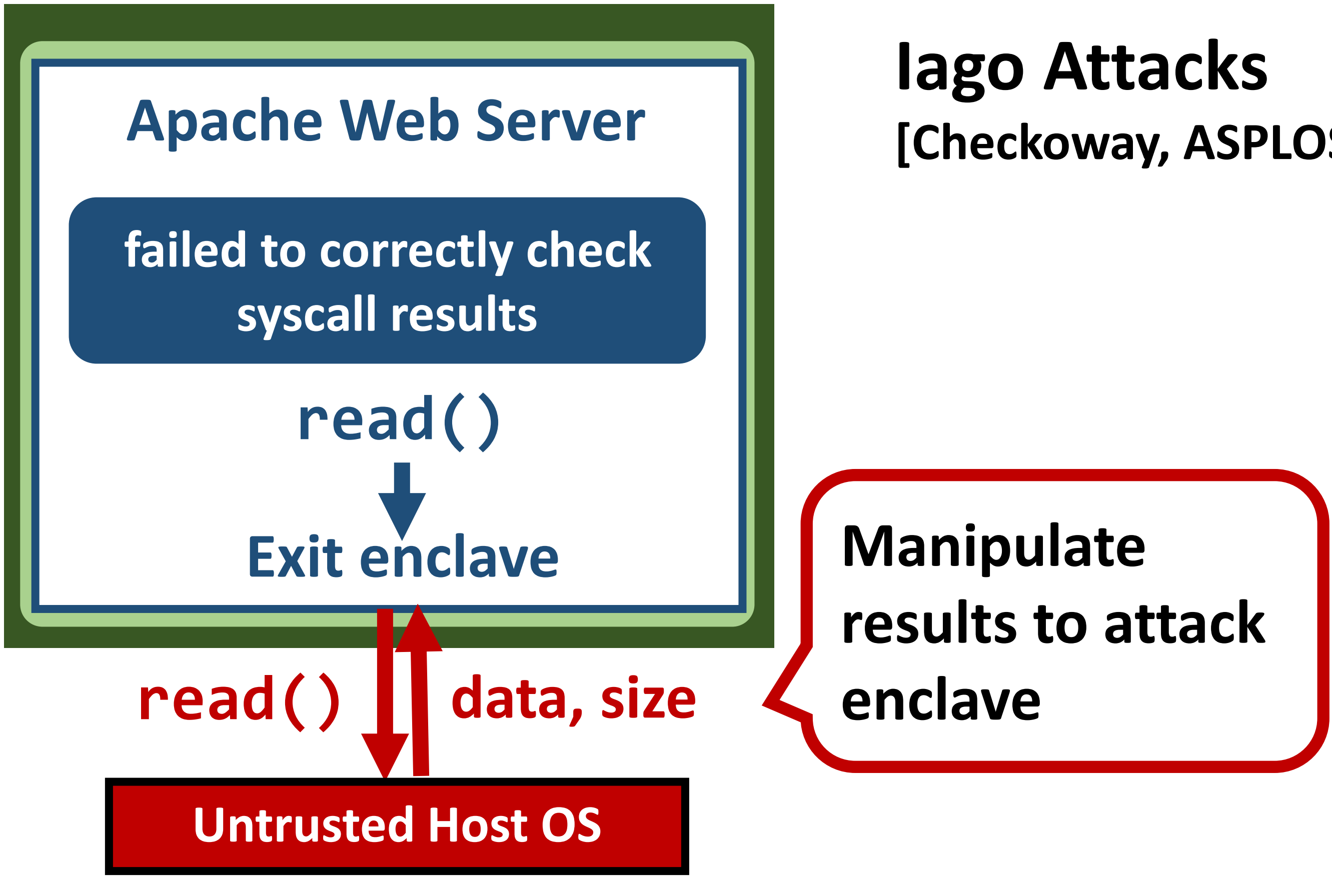
1. Statically linking all binaries
2. Bypassing instructions (CPUID/RDTSC)
3. Exiting enclave for system calls

Security Challenge!

SGX Porting Challenges

- Satisfying enclave requirements
- Defending against untrusted OS services
- Improving performance factors

Attack Vectors from Untrusted OS



Iago Attacks
[Checkoway, ASPLOS 13]

Iago Attacks In A Nutshell

- Semantic attacks by manipulating syscall results
- Application-specific
- Bugs that do not exist on a trusted OS

Iago Attacks: Why the System Call API is a Bad Untrusted RPC Interface

Stephen Checkoway
Johns Hopkins University
s@cs.jhu.edu

Hovav Shacham
UC San Diego
hovav@cs.ucsd.edu

Abstract

In recent years, researchers have proposed systems for running trusted code on an untrusted operating system. Protection mechanisms deployed by such systems keep a malicious kernel from directly manipulating a trusted application's state. Under such systems, the application and kernel are, conceptually, peers, and the system call API defines an RPC interface between them.

We introduce *Iago attacks*, attacks that a malicious kernel can

Listing 1. A Linux program that can be completely compromised by an Iago attack.

```
#include <stdlib.h>
int main() {
    void *p = malloc(100);
}
```

Iago Attack Example:
SSL Random Generator Seed

mod_ssl (Apache)

```
int ssl_rand_seed(…)
{
    …
    if (pRandSeed->nSrc == SSL_RSSRC_BUILTIN) {
        struct {
            time_t t;
            pid_t pid;
        } my_seed;

        my_seed.t = time(NULL);
        my_seed.pid = getpid();

        l = sizeof(my_seed);
        RAND_seed((unsigned char *)&my_seed, l);
    }
```

OS can give the same pid and time

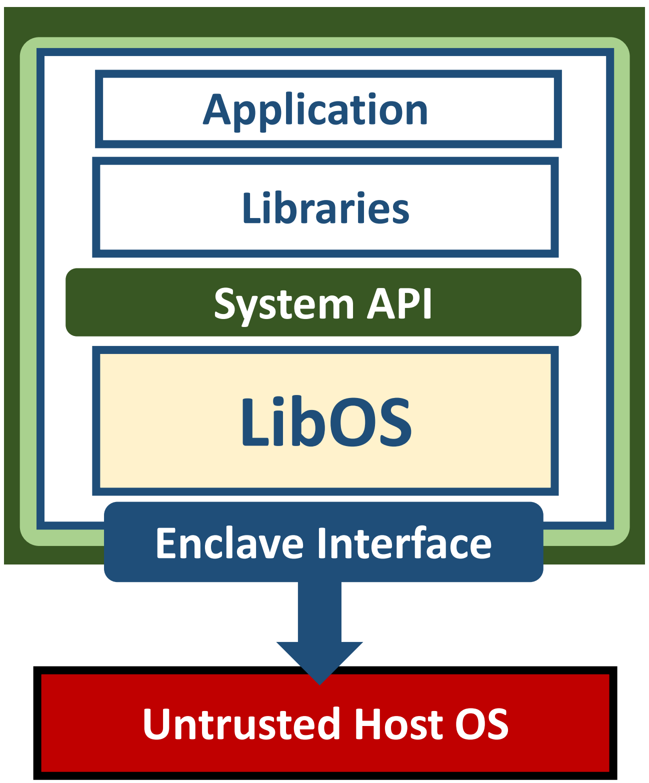
SGX Shielding Frameworks

- Several work address the problem of SGX porting
  - (1) Defenses against Iago attacks
  - (2) Performance optimization
  - (3) Compatibility features (e.g., cross compilers)
- Two approaches:
  - (1) Library OSes
  - (2) Shielding layers

Key Factors

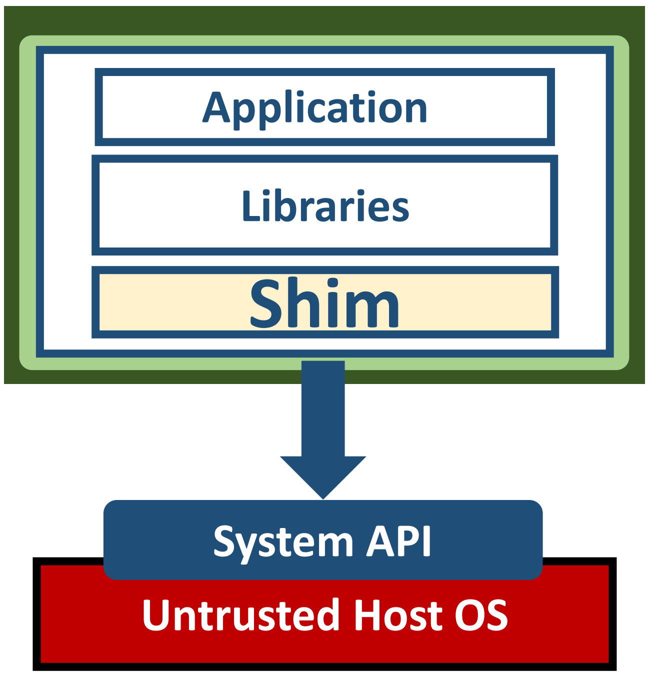
- Shielding mechanisms (especially Iago attacks)
- Attack surface
- Trusted computing base (TCB)
- OS functionality

Library OSes



- OS components in enclave
- Define small enclave interface with security in mind
- Example:
  Haven [OSDI'14]
  Graphene-SGX

Shielding Layers



- Shielding each API
- Avoid library OS overheads
- Small TCB
- Example: SCONE, Panoply

Comparison

	Graphene-SGX	SCONE	Panoply
Approach	**Library OS**	**Shielding Layers**	
Enclave interface	**Fixed interfaces** (regardless of libOS functionality)	**Equals the system API needed by the application**	

Trusted Computing Base

	Graphene-SGX	SCONE	Panoply
LibOS/ Shielding Layer	**53 kLoC**	**97 kLoC**	**10kLoC**
Libc option	GLIBC (1.1 MLoC)	MUSL (88 kLoC)	No Libc in enclave

The choice of Libc is the highest-order bits

SGX Porting Challenges

- Satisfying enclave requirements
- Defending against untrusted OS services
- Improving performance factors

Performance Factors

- Enclave creation time
  - Correlated with enclave memory size (1GB requires ~3s)
- Memory access overheads
  - LLC misses up to 10X
  - EPC paging: 128MB shared among all enclaves
    40,000 cycles for page-out and page-in
- Enclave exits
  - 7,000~8,000 cycles for exit and re-enter

Performance improvement

- Enclave creation time: EDMM on SGX2
  - Dynamically adding pages at run time
- Reduce explicit & implicit exits: Eleos
  - Completely exit-less enclaves
  - Pinning EPC pages with software-based paging

Topics

- Porting challenges and OS attack vectors
- Library OS: Graphene-SGX
- System interface shields: SCONE, Panoply
- EDMM on SGX2
- Exit-less enclaves with Eleos

Graphene-SGX:
A LibOS for Unmodified Applications

- Servers, Command-line, Runtimes:
  Apache, NGINX, GCC, R, Python, OpenJDK, etc
- Multi-process APIs: fork, exec, IPC, etc
- Not perfect, but a quick, practical porting option

Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX

Chia-Che Tsai
Stony Brook University

Donald E. Porter
University of North Carolina at Chapel Hill and Fortanix

Mona Vij
Intel Corporation

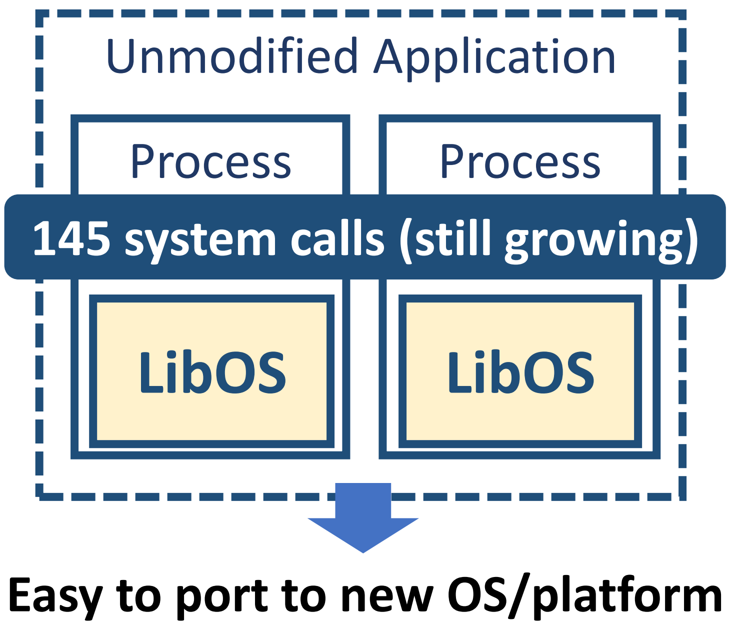
Abstract

Intel SGX hardware enables applications to protect themselves from potentially-malicious OSes or hypervisors. In cloud computing and other systems, many users and applications could benefit from SGX. Unfortunately, current applications will not work out-of-the-box on SGX. Although previous work has shown that a li- of commodity operating systems is not without blemish. Thus, a significant number of users would benefit from running applications on SGX as soon as possible.

Unfortunately, applications do not "just work" on SGX. SGX imposes a number of restrictions on enclave code that require application changes or a layer of indirection. Some of these restrictions are motivated by

The Graphene LibOS Project [Eurosys14]

- Open library OS for reusing Linux applications (**github.com/oscarlab/graphene**)
  - Inspired by Drawbridge [ASPLOS11] and Haven [OSDI14]
  - Under active development



Applications in Graphene-SGX

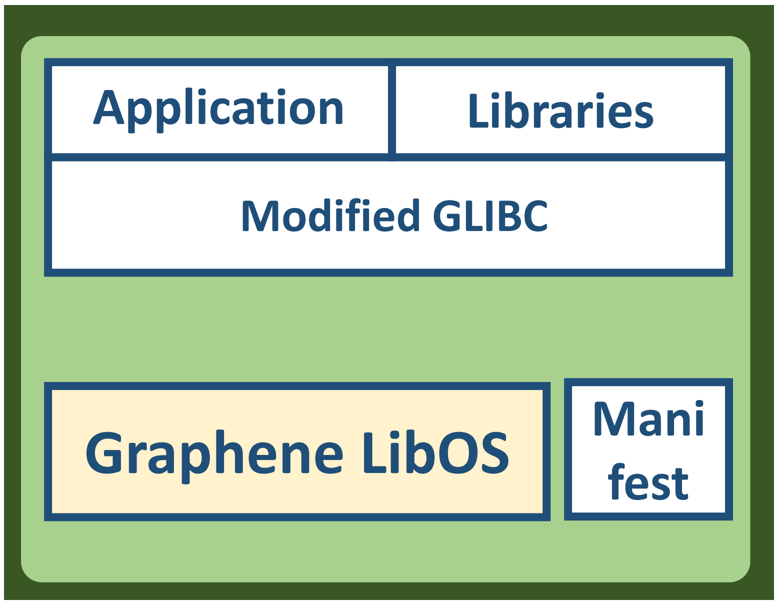
1. **Static initial image**
2. **No system calls**
3. **Check for untrusted inputs**

$ SGX=1 ./pal_loader httpd [args]

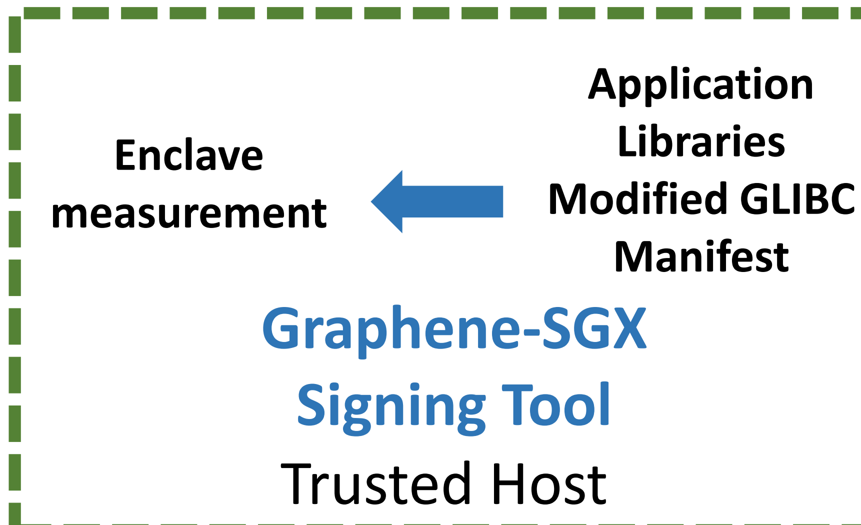
Graphene Loader

Untrusted OS

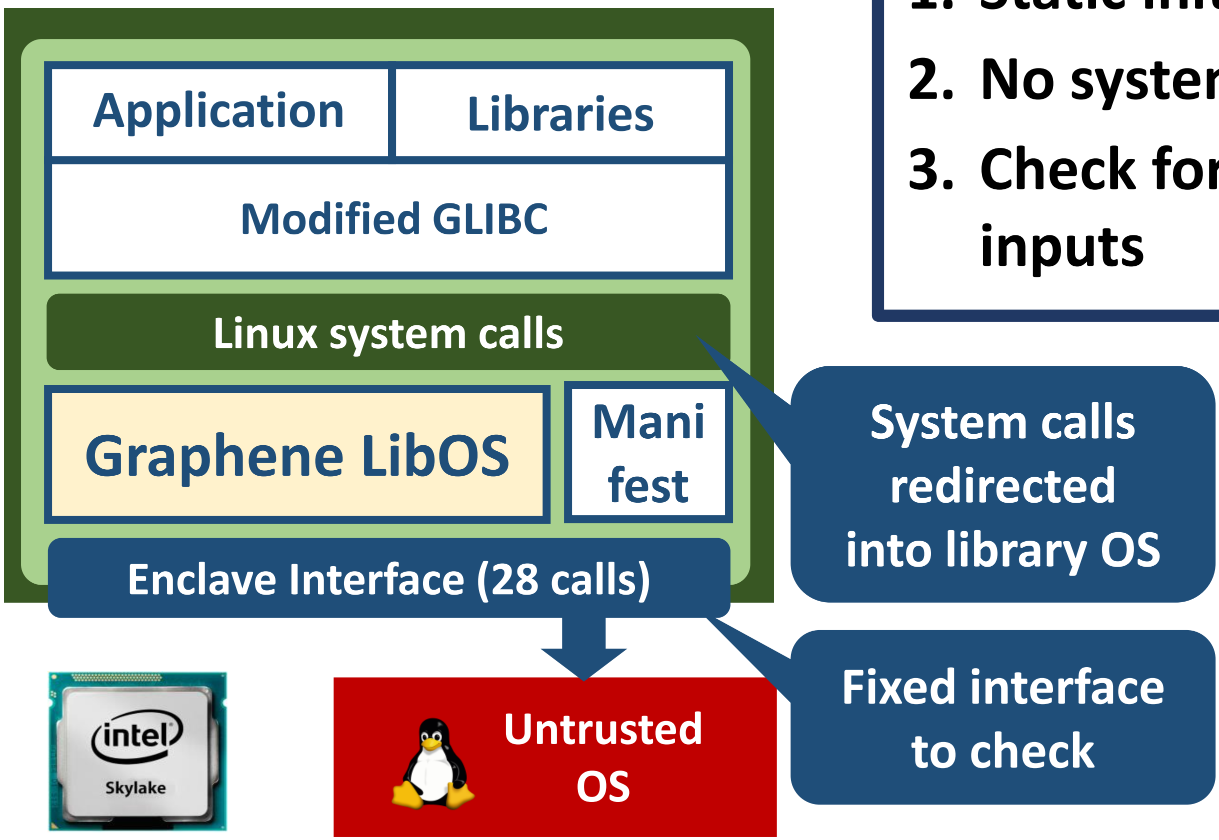
Applications in Graphene-SGX



1. **Static initial image** ✓
2. **No system calls**
3. **Check for untrusted inputs**



Applications in Graphene-SGX

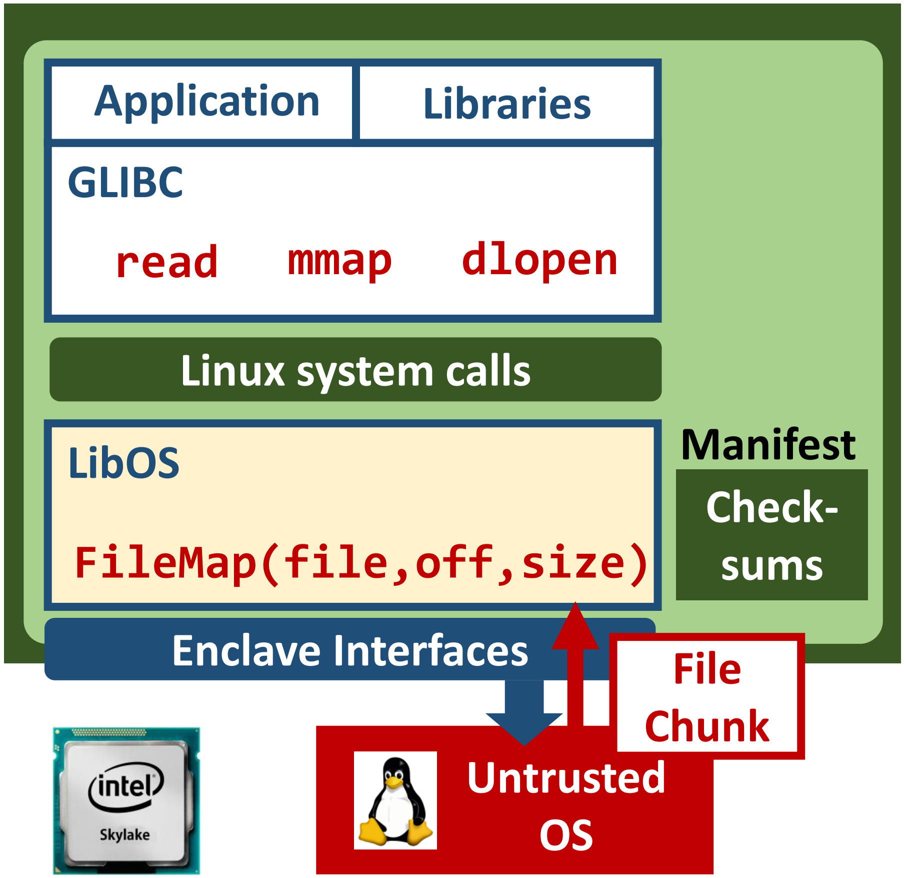


1. **Static initial image** ✓
2. **No system calls** ✓
3. **Check for untrusted inputs**

Checking Enclave Interface

- Reduce enclave interface to 28 calls
- Design defense for each call
  - Define explicit semantics
    ➔ knowing exactly what/how to check
  - Crypto techniques
- Examples:
  - Accessing integrity-sensitive files (binaries / configs)
  - Process creation (see paper)

Ex: File Integrity Check

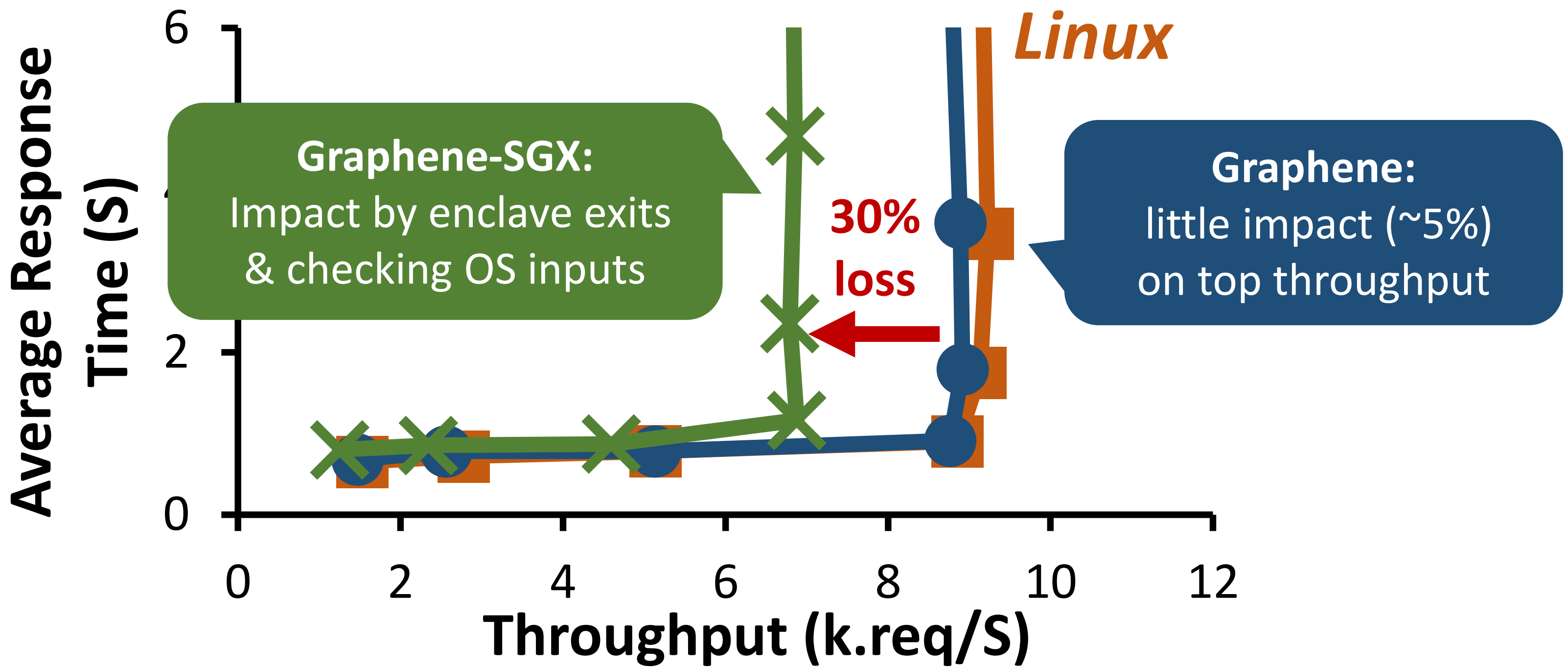


- Ask for exact file content
- Verify by checksums

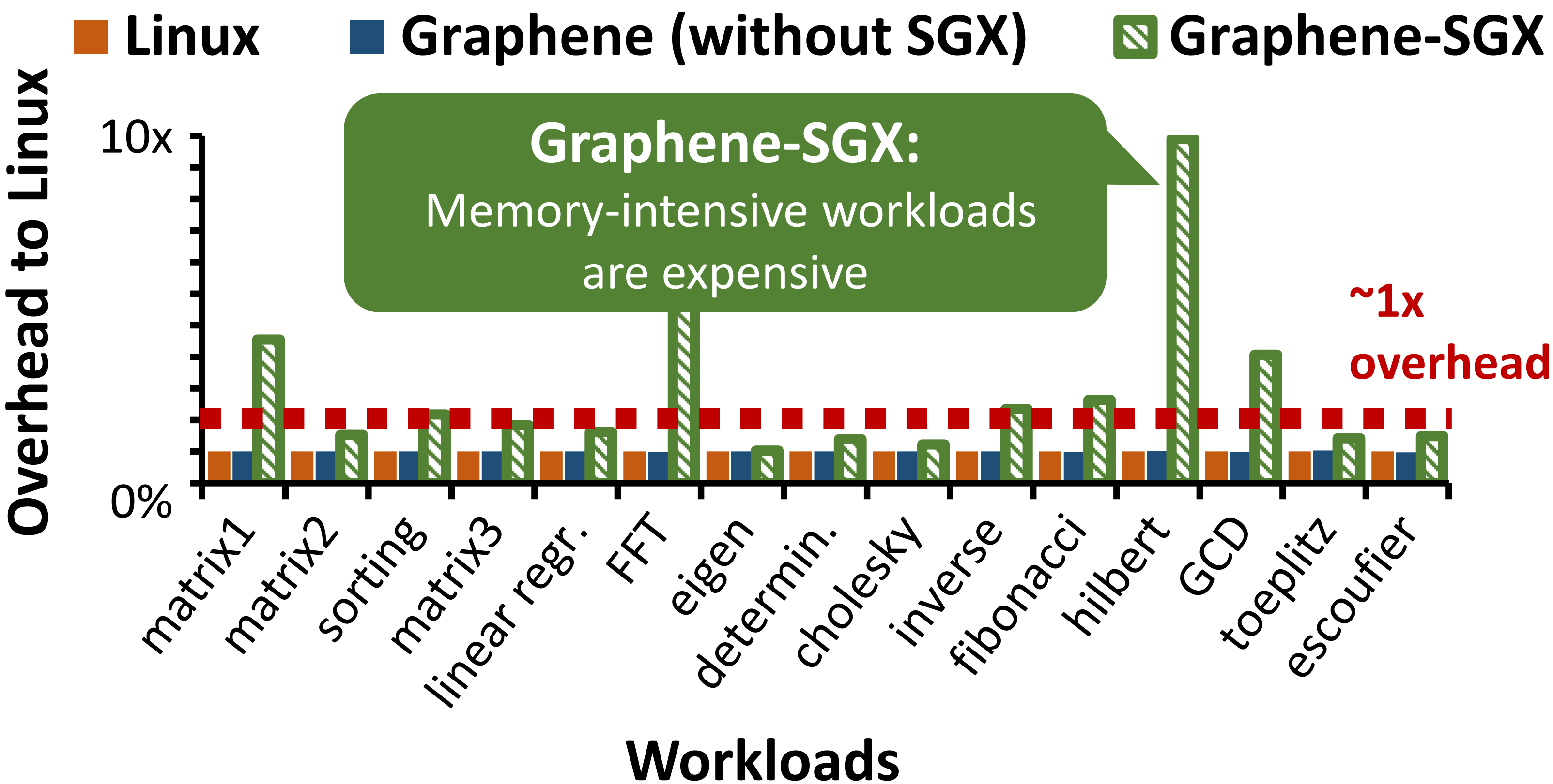
Checking All 28 Enclave Calls

Examples	#	Result	Explanation
(1) Reading a file (2) Inter-proc coordination	**18**	**Fully Checked**	**(1) File checksums (2) CPU attestation + crypto: inter-proc TLS connection**
Yielding a thread	**6**	**Benign**	Do not take any input
(1) Polling (2) File attributes	**4**	**Unchecked**	May cause DoS; Future work

Apache (5 Procs w/ IPC Semaphore)



R Benchmarks



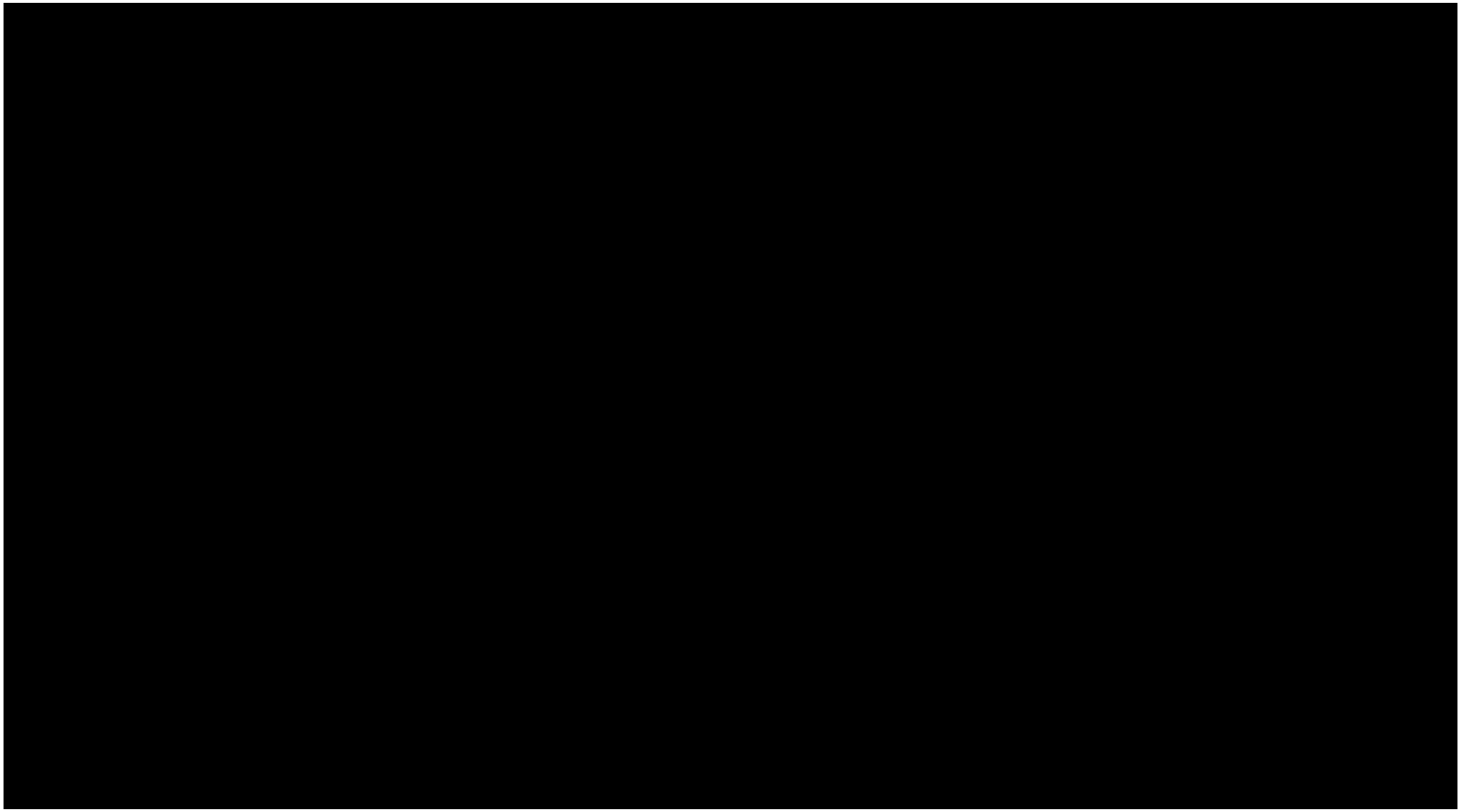
Graphene-SGX Features

- Current features
  - Use GLIBC by default; can use MUSL if acceptable
  - A wide range of servers, command-lines, language runtimes tested
  - Static binary support
  - Limitations: cannot support shared memory

Demo: GCC on Graphene-SGX

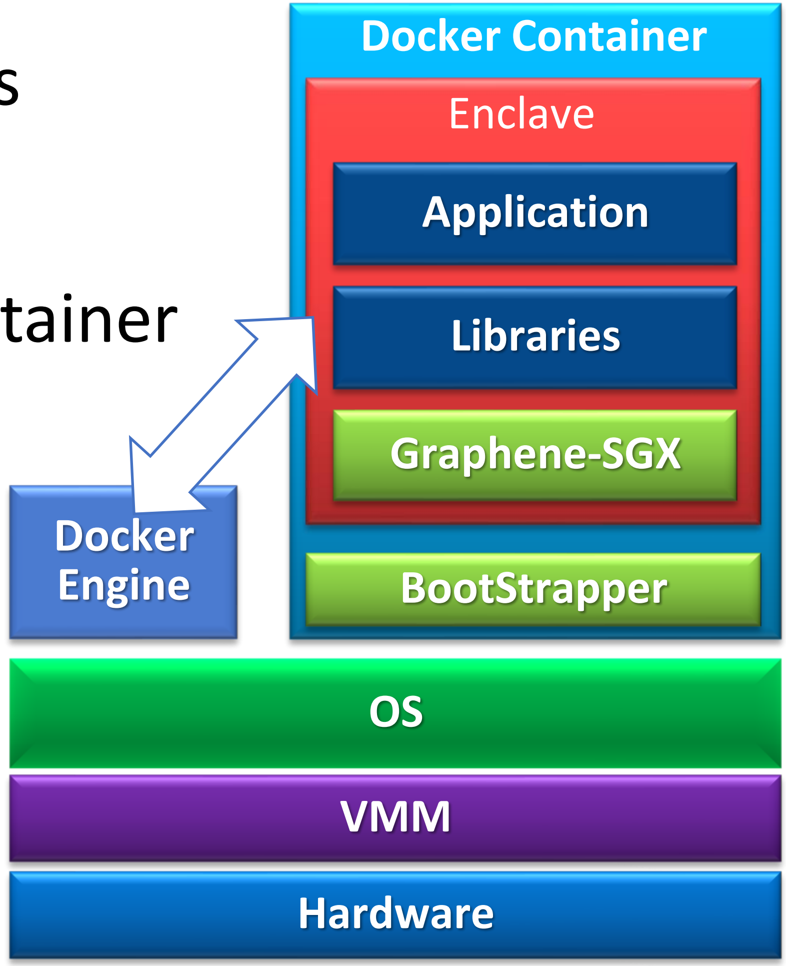
- Multi-process: gcc→cc1→collect2→ld
- Turn on DEBUG=1
- Attack: Try to modify the GCC binary

Demo: GCC on Graphene-SGX

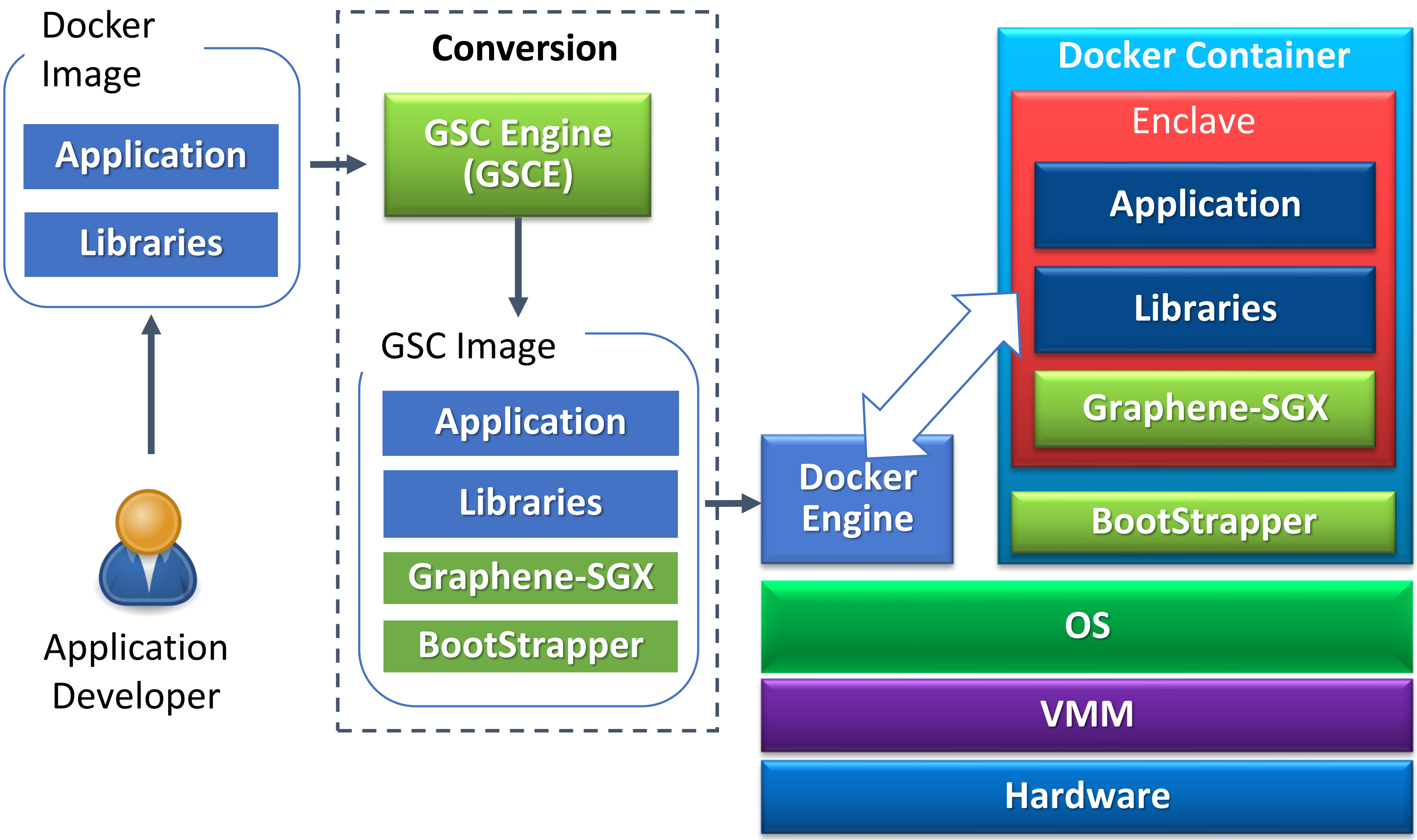


GSC: Graphene Secure Container

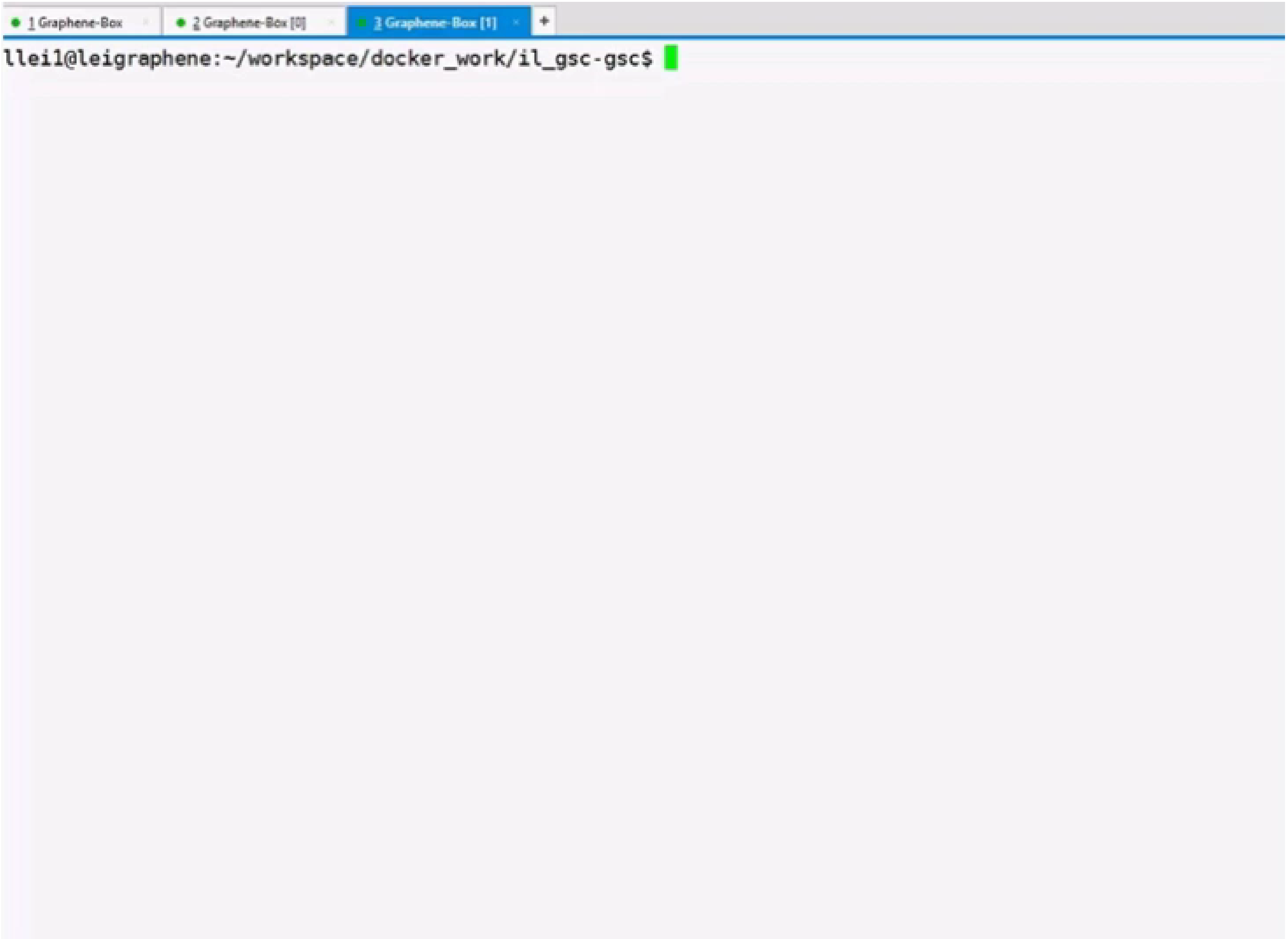
- Docker images ➔ enclaves
  - Dockerfiles ➔ manifests
- Graphene-SGX runs in container
  - Mutual isolation between OS and application



GSC: Graphene Secure Container



Demo: Graphene-SGX Container

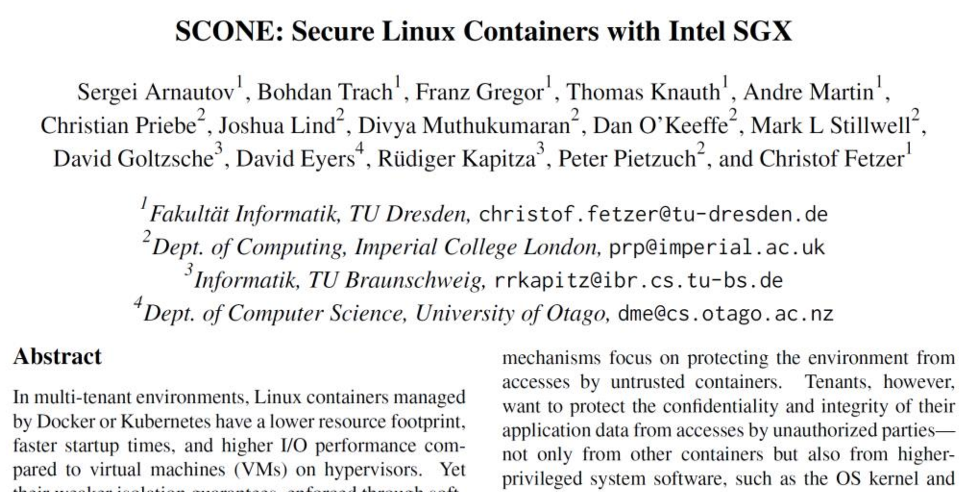

1 Graphene-Box
2 Graphene-Box [0]
3 Graphene-Box [1]
llei1@leigraphene:~/workspace/docker_work/il_gsc-gsc$

Availability

- Open-source at http://github.com/oscarlab/graphene
- Currently under GPLv3, switching to LGPL soon
- Contact:
  - chiache@cs.stonybrook.edu
  - porter@cs.unc.edu
  - https://graphene-libraryos.slack.com (contact me for invitation)

SCONE: A Lightweight Layer for SGX

- An enhanced C library with file and network shields
- Strictly requires no library OS
- Optimized syscall performance for enclaves


SCONE: Secure Linux Containers with Intel SGX

Sergei Arnautov[1], Bohdan Trach[1], Franz Gregor[1], Thomas Knauth[1], Andre Martin[1], Christian Priebe[2], Joshua Lind[2], Divya Muthukumaran[2], Dan O'Keeffe[2], Mark L Stillwell[2], David Goltzsche[3], David Eyers[4], Rüdiger Kapitza[3], Peter Pietzuch[2], and Christof Fetzer[1]

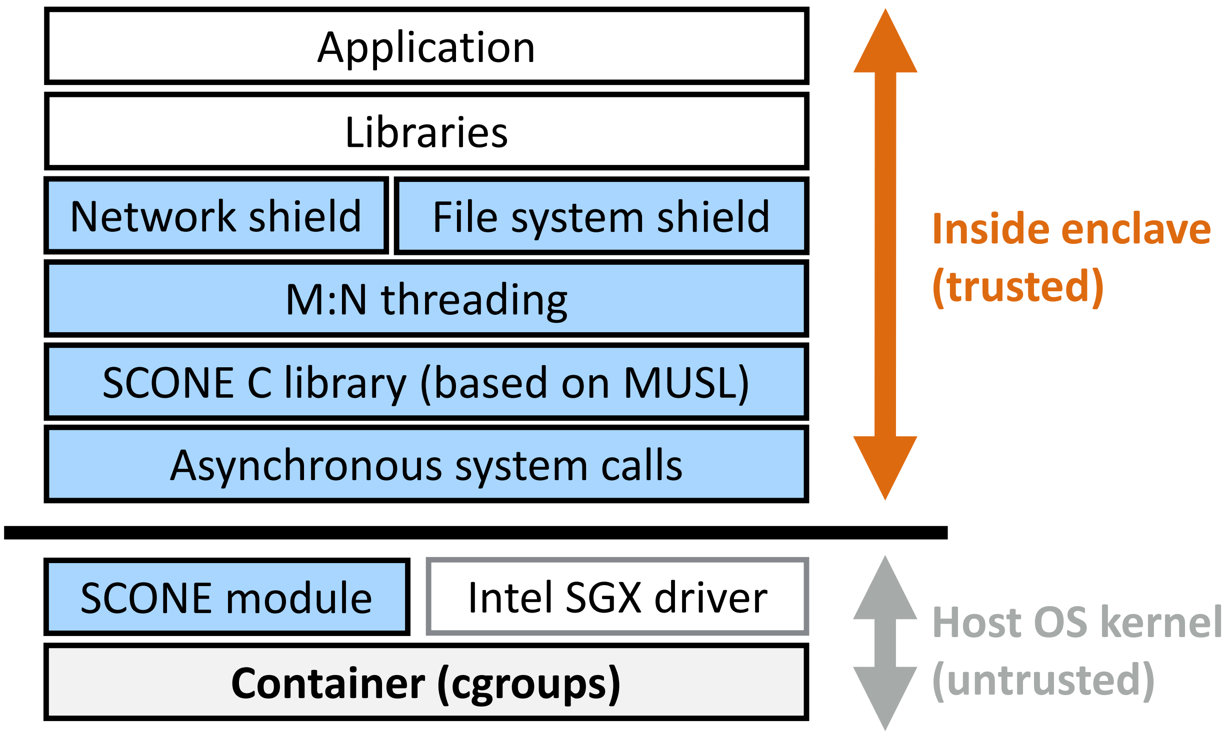
[1] *Fakultät Informatik, TU Dresden,* christof.fetzer@tu-dresden.de
[2] *Dept. of Computing, Imperial College London,* prp@imperial.ac.uk
[3] *Informatik, TU Braunschweig,* rrkapitz@ibr.cs.tu-bs.de
[4] *Dept. of Computer Science, University of Otago,* dme@cs.otago.ac.nz

Abstract

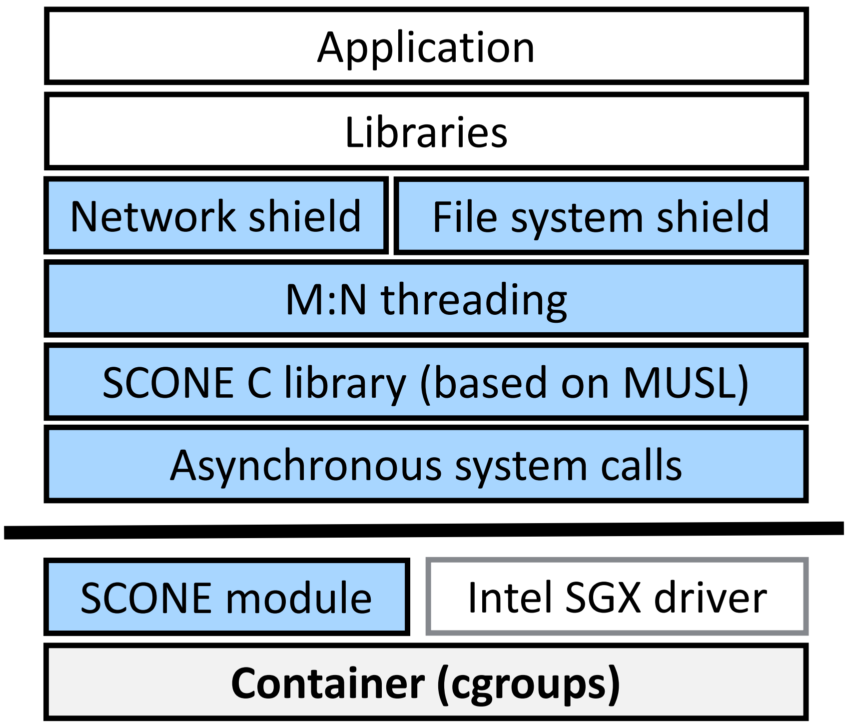
In multi-tenant environments, Linux containers managed by Docker or Kubernetes have a lower resource footprint, faster startup times, and higher I/O performance compared to virtual machines (VMs) on hypervisors. Yet

mechanisms focus on protecting the environment from accesses by untrusted containers. Tenants, however, want to protect the confidentiality and integrity of their application data from accesses by unauthorized parties— not only from other containers but also from higher-privileged system software, such as the OS kernel and

SCONE Architecture

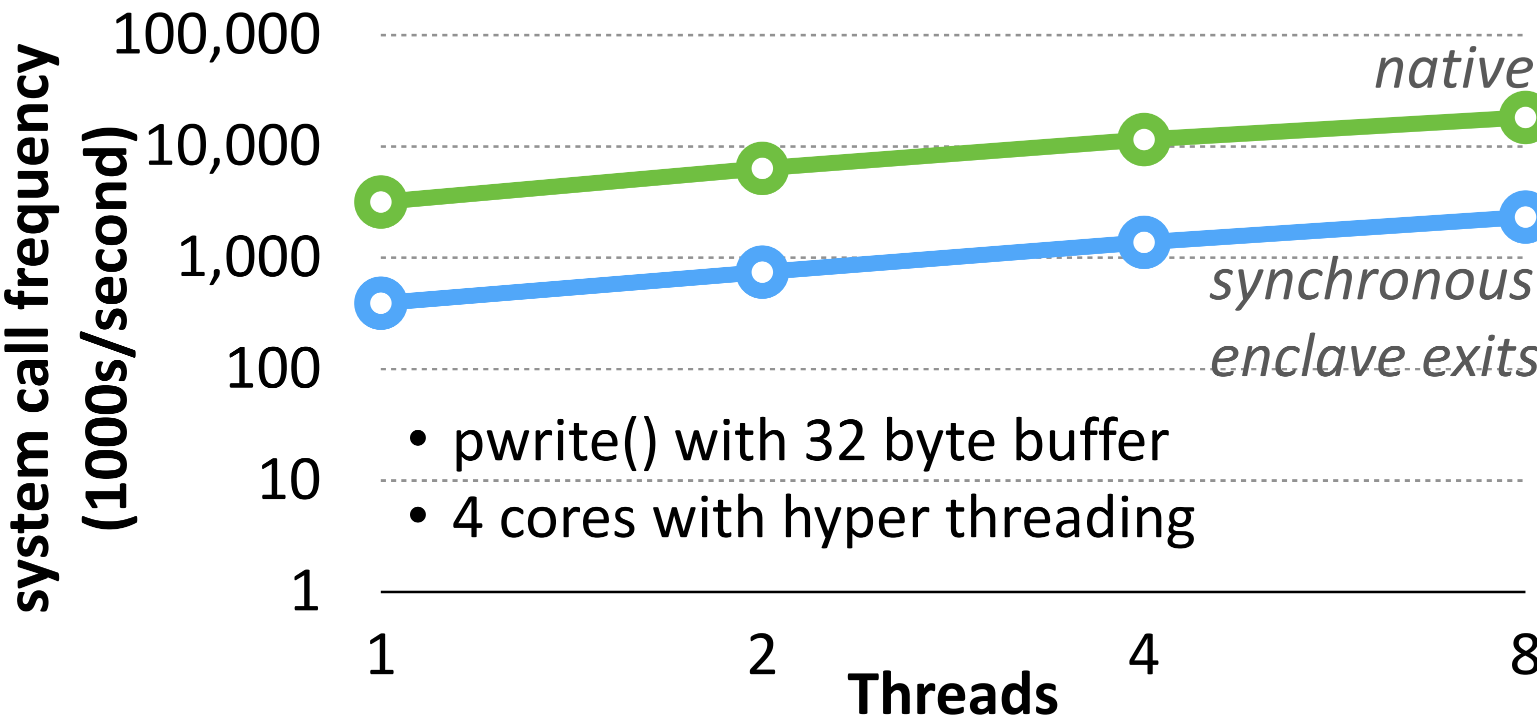


SCONE Architecture



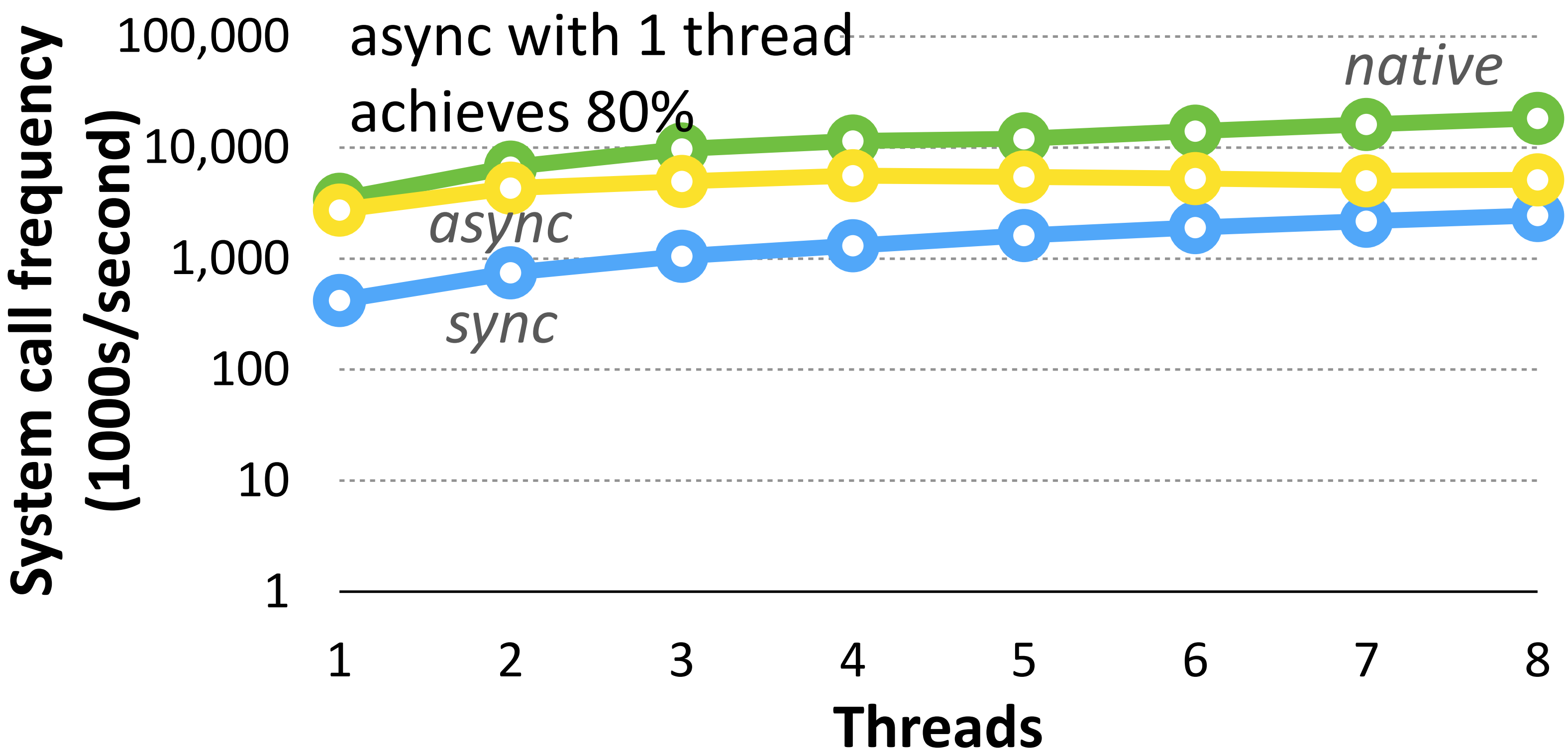
- <u>Network and FS shields:</u> encrypting and authenticating network and file contents
- <u>MUSL:</u> small TCB (88KLoC)
- <u>Asynchronous system calls:</u> avoid enclave exits
- <u>SCONE module (optional):</u> improve performance

System Call Overheads

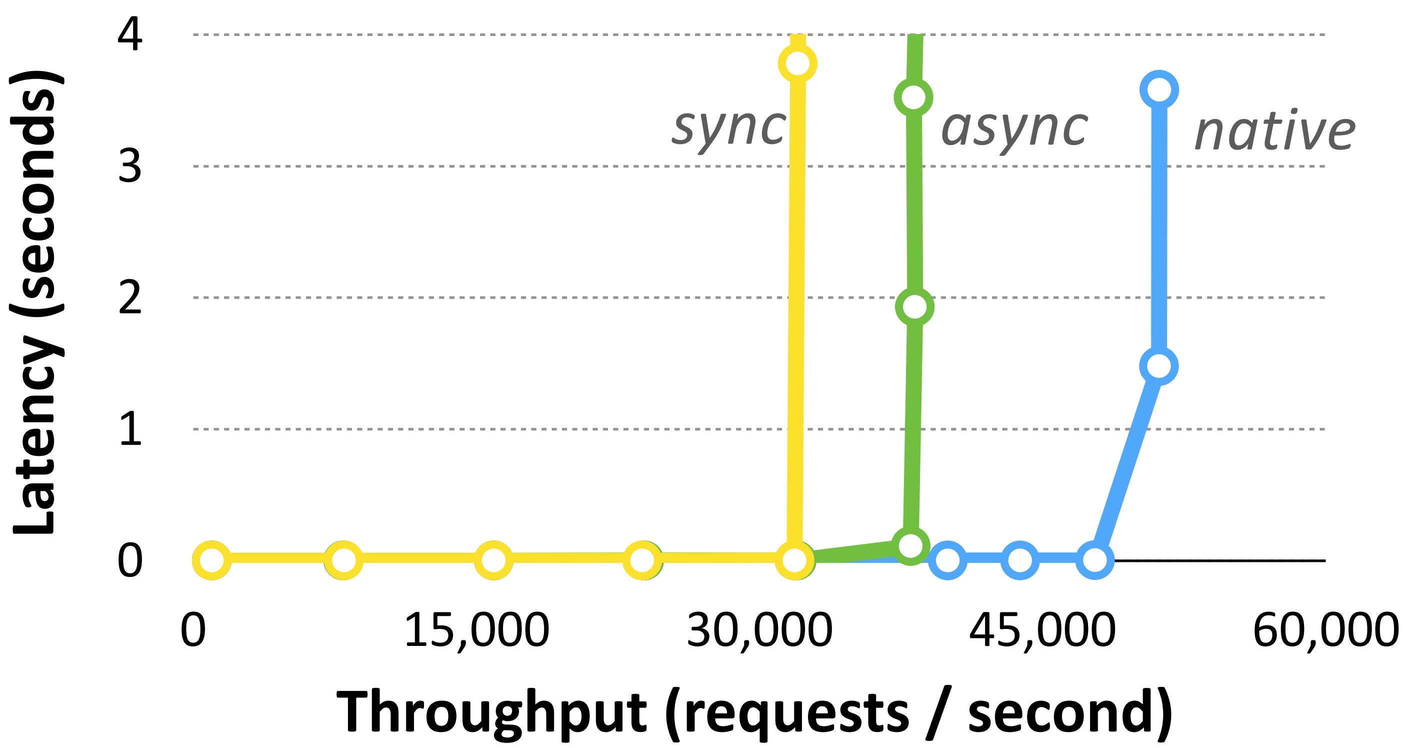


- pwrite() with 32 byte buffer
- 4 cores with hyper threading

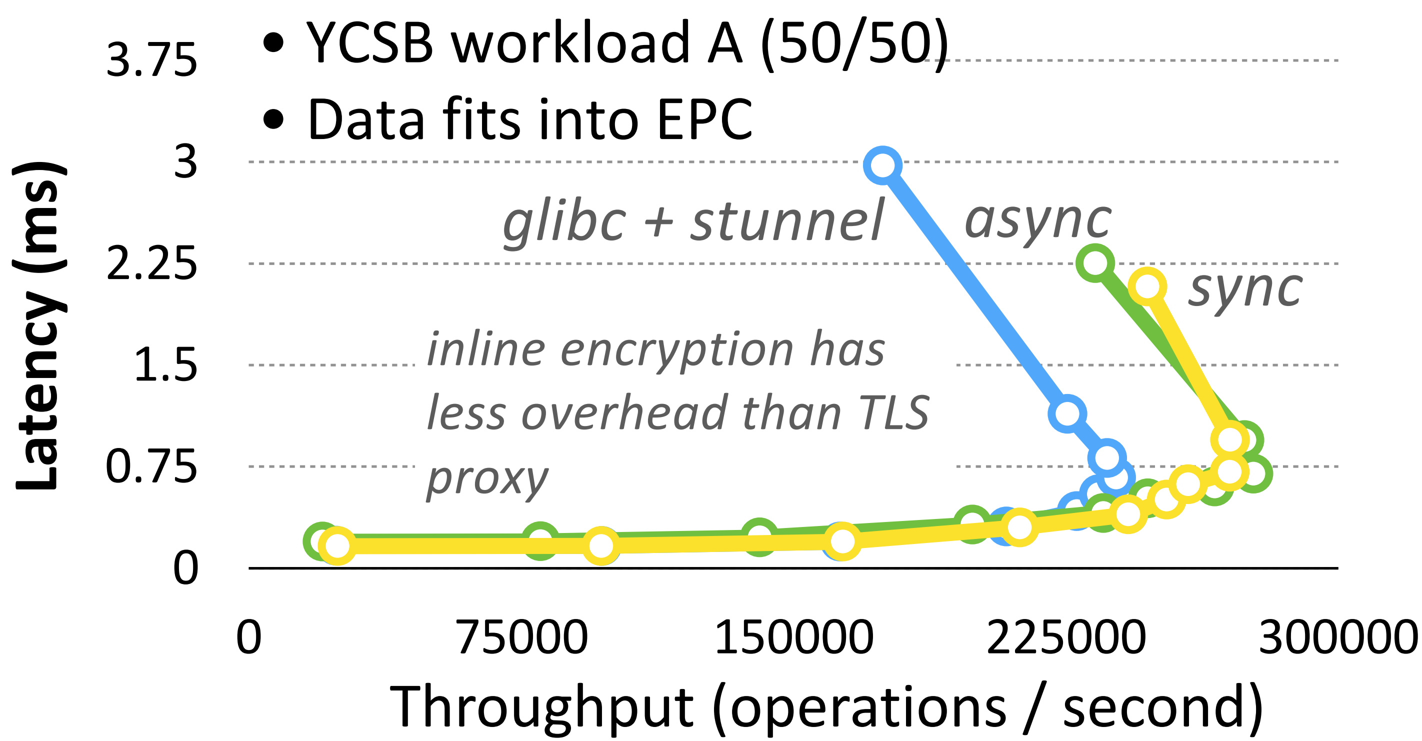
Asynchronous System Calls



Apache Throughput



Memcached Throughput



SCONE Language Support

- Cross compiler for several languages
  - C and C++
  - GO
  - Rust
  - Python
  - PHP
  - Java (partial support, still work in progress)

Demo: SCONE Cross Compiler

SCONE Features

- Current SCONE features
  - Support static and dynamic linking
  - Unmodified binaries must be position independent (built with `-fPIC`)
  - Compatible with MUSL
  - No multi-processing (fork / execve)

SCONE Docker Integration

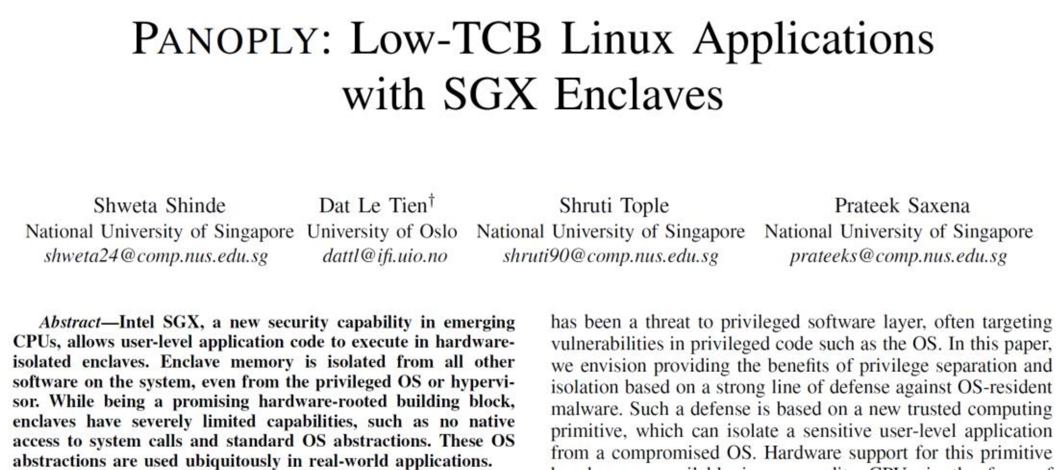
- SCONE supports (extended) Docker compose files
  - Transparent attestation of services
  - Transparent configurations
- Unmodified Docker Engine
  - Docker engine runs outside enclave

Availability

- Commercially available via SCONTAIN
- Acquire the software: www.scontain.com
- Contact: christof.fetzer@gmail.com

Panoply: POSIX API with Small TCB

- A POSIX library without Libc in enclave
- Placing applications and libraries into separate enclaves
- 10kLoC TCB in Panoply shim library



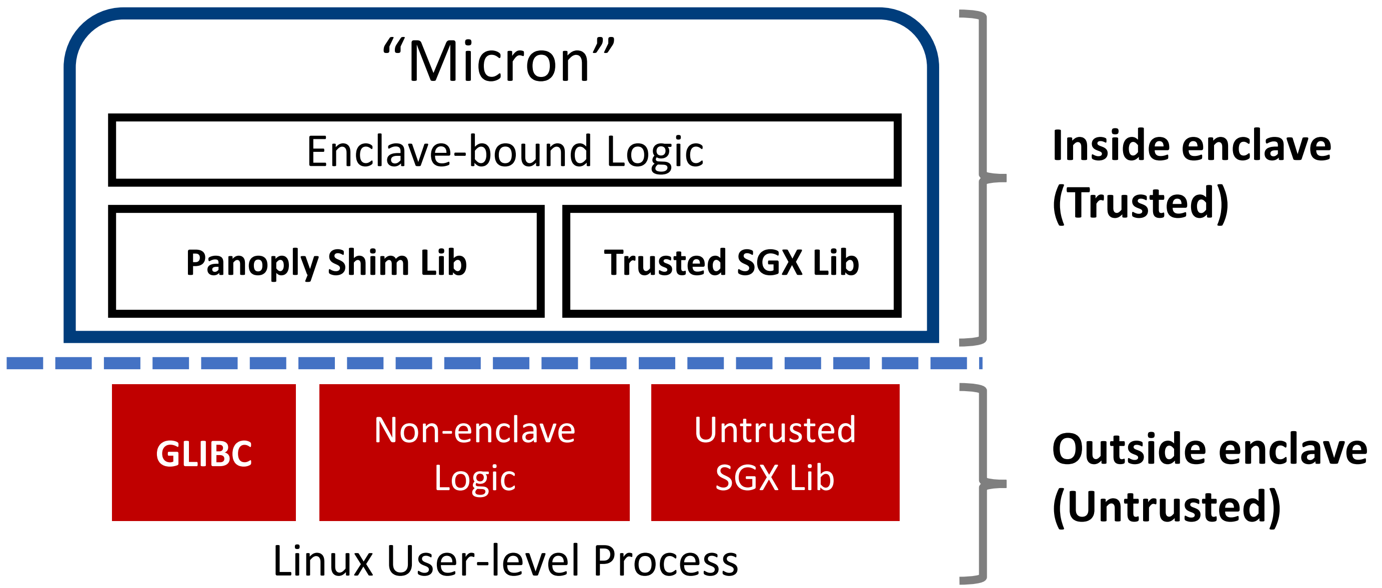
PANOPLY: Low-TCB Linux Applications with SGX Enclaves

Shweta Shinde, National University of Singapore, shweta24@comp.nus.edu.sg
Dat Le Tien[†], University of Oslo, dattl@ifi.uio.no
Shruti Tople, National University of Singapore, shruti90@comp.nus.edu.sg
Prateek Saxena, National University of Singapore, prateeks@comp.nus.edu.sg

Abstract—**Intel SGX, a new security capability in emerging CPUs, allows user-level application code to execute in hardware-isolated enclaves. Enclave memory is isolated from all other software on the system, even from the privileged OS or hypervisor. While being a promising hardware-rooted building block, enclaves have severely limited capabilities, such as no native access to system calls and standard OS abstractions. These OS abstractions are used ubiquitously in real-world applications.**

has been a threat to privileged software layer, often targeting vulnerabilities in privileged code such as the OS. In this paper, we envision providing the benefits of privilege separation and isolation based on a strong line of defense against OS-resident malware. Such a defense is based on a new trusted computing primitive, which can isolate a sensitive user-level application from a compromised OS. Hardware support for this primitive

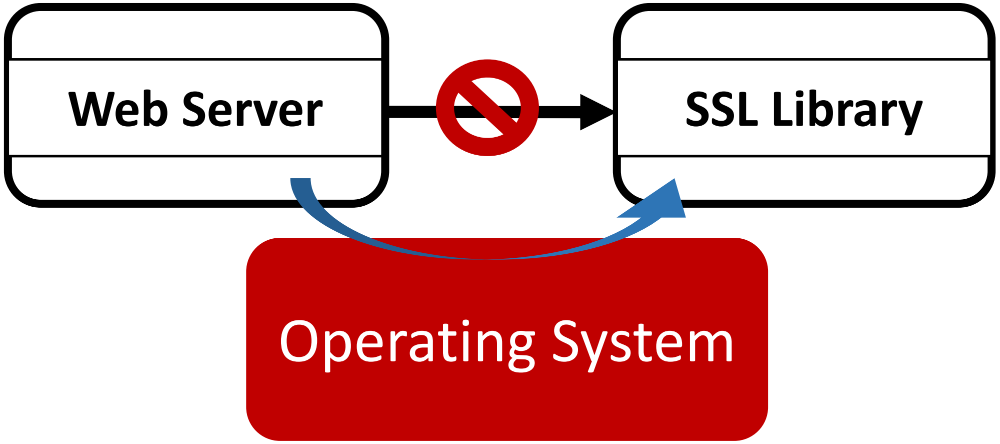
Panoply Architecture



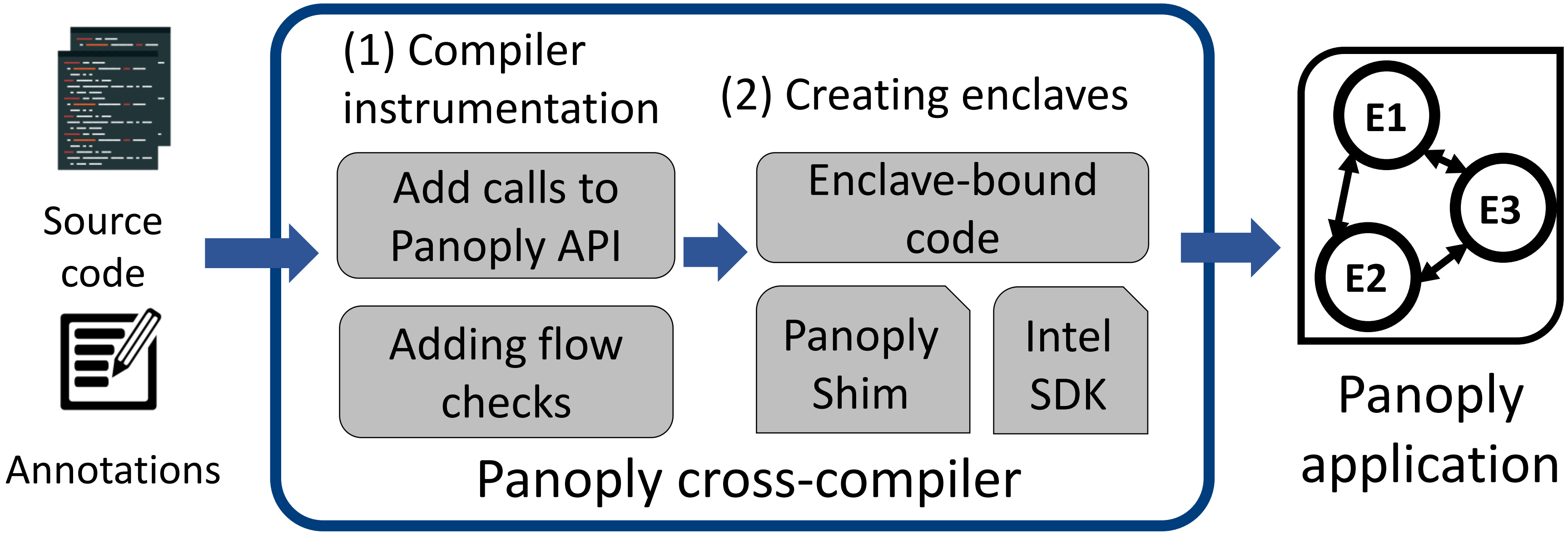
Panoply expels GLIBC outside of the enclave

Panoply Architecture

- Micron can be an application or a library
- Multi-enclave collaboration:



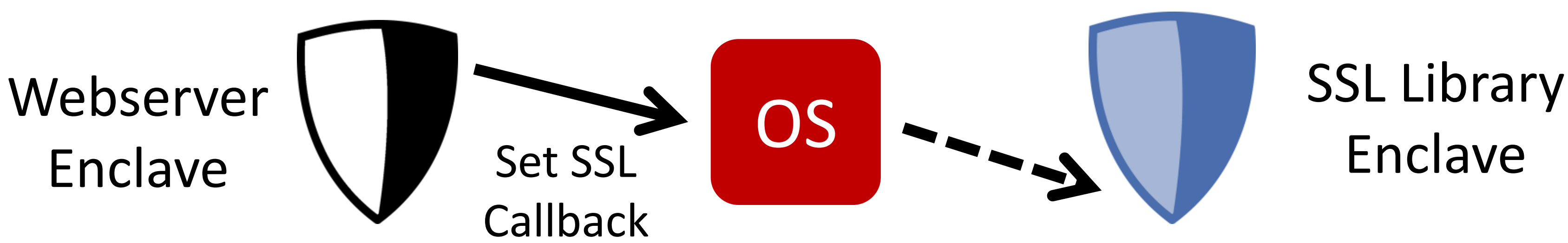
Micron Generation



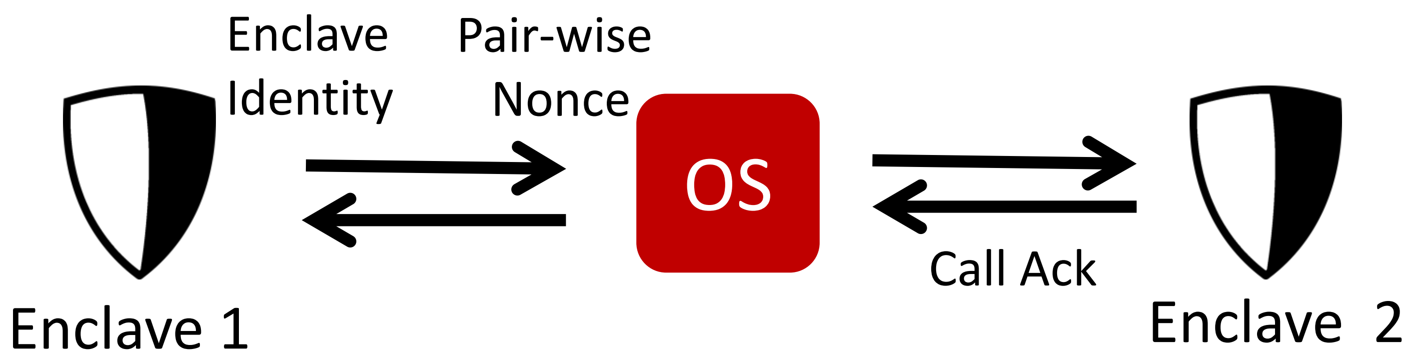
Attacks on Multi-Enclave Applications

```
session_t session;
certificate_credentials_t xcred;

/* Specify callback function*/
certificate_set_verify_function (...);

/* Initialize TLS session */
init (&session, TLS_CLIENT);
```

[SSL Lib]



Securing Multi-Enclave Applications



Attack	Defenses
Spoofing	Sender / Receiver Authentication
Replay	Message Freshness
Silent Drops	Reliable Delivery

Performance Overview

App	Panoply	Empty enclave	Overhead
OpenSSL	3.16	2.79	**13%**
H2O	8.79	6.56	**34%**
FreeTDS	8.74	8.60	**1%**
Tor	6.72	4.54	**48%**

Panoply Features

- Currently support 254 POSIX API
- 91 guarantee to preserve API semantics
- Multi-process: fork and exec

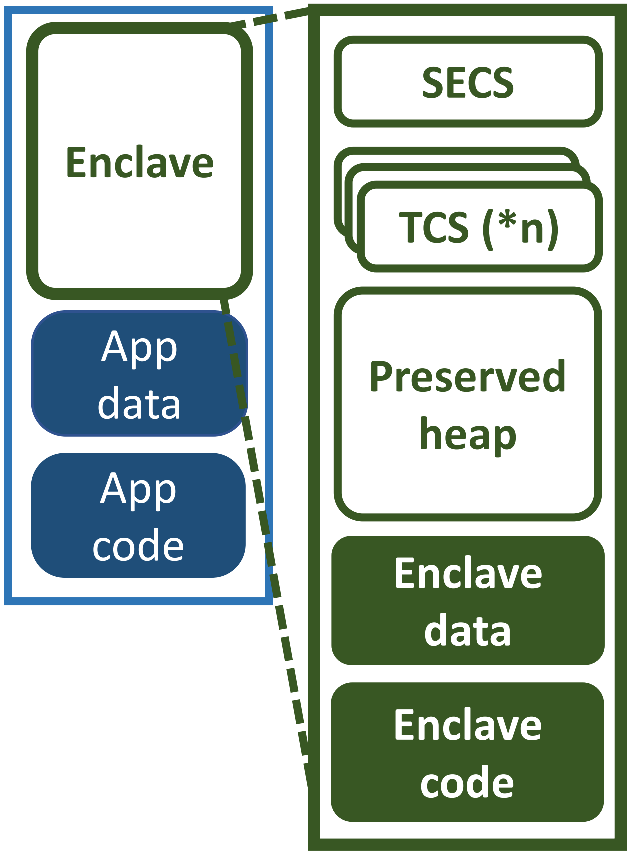
Availability

- Open-source at https://shwetasshinde24.github.io/Panoply/
- Apache 2.0 License
- Contact: shweta24@comp.nus.edu.sg

EDMM: Enclave Dynamic Memory Mgmt

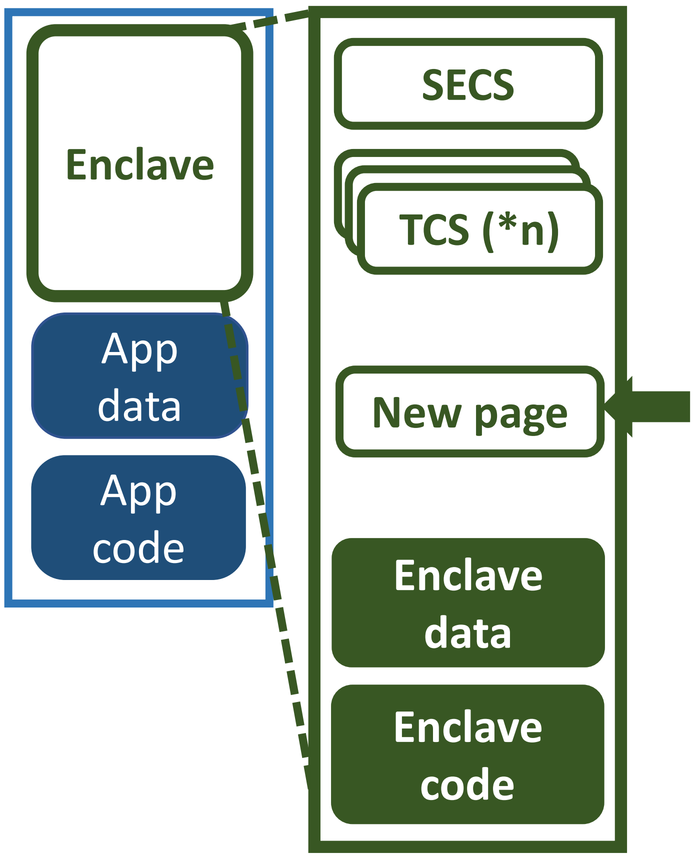
- Current SGX: fixed enclave memory and threads
- SGX2: adding pages at run time
  - Reduce initial enclave memory size
  - Dynamic thread creation
  - Dynamic page protection (for dynamic loading / JIT)
- Supported in future Graphene-SGX

Current SGX Limitations



- For integrity, each enclave has a static memory layout
  - Signed by users
  - Initialized at loading time
- Reserved heap for malloc()
- # TCS = # Threads

EDMM on SGX2



- Adding and protecting enclave pages at run time
- Page adding semantics:
  - Normal or TCS pages
  - Must be zeroed
  - “Approved” by enclave

EDMM Support in Graphene-SGX

- Compatibility and performance features
  - Largely reduce startup time
  - Dynamic thread creation
  - Protect pages after finishing dynamic loading
  - Support mprotect()

Demo: Graphene-SGX with EDMM

```
leifepc@leifepc:~/work/graphene/LibOS/shim/test/apps/python$ ls
benchmarks.tar.gz  gai.conf  hosts  Makefile  Python-2.7.9.tgz  python.manifest.temp
leifepc@leifepc:~/work/graphene/LibOS/shim/test/apps/python$ 
```

Availability

- SGX2 release date expected in 1~2 years
- EDMM support will be open-sourced as part of Graphene
  - http://github.com/oscarlab/graphene

Eleos: Exit-less Enclaves

- Avoids enclave exits and EPC paging
- Combined w/ SDK: Generating RPC-based interface
- Software-based paging: SUVM

Eleos: ExitLess OS Services for SGX Enclaves

Meni Orenbach, Pavel Lifshits, Marina Minkin, Mark Silberstein
Technion - Israel Institute of Technology

Abstract

Intel Software Guard eXtensions (SGX) enable secure and trusted execution of user code in an isolated *enclave* to protect against a powerful adversary. Unfortunately, running I/O-intensive, memory-demanding server applications in enclaves leads to significant performance degradation. Such applications put a substantial load on the in-enclave system call and secure paging mechanisms, which turn out to be the main reason for the application slowdown. In addition to the high direct cost of thousands-of-cycles long SGX management instructions, these mechanisms incur the high indirect

OS and/or a hypervisor, yet the code running in the enclave may access *untrusted* memory of the owner process.

While SGX provides the convenience of a standard `x86` execution environment inside the enclave, there are important differences in the way enclaves manage their private memory and interact with the host OS.

First, because an enclave may only run in user mode, OS services, e.g., system calls, are not directly accessible. Instead, today's SGX runtime forces the enclave to exit, that is, to *securely transition* from trusted to untrusted mode, and to re-enter the enclave after the privileged part of the system

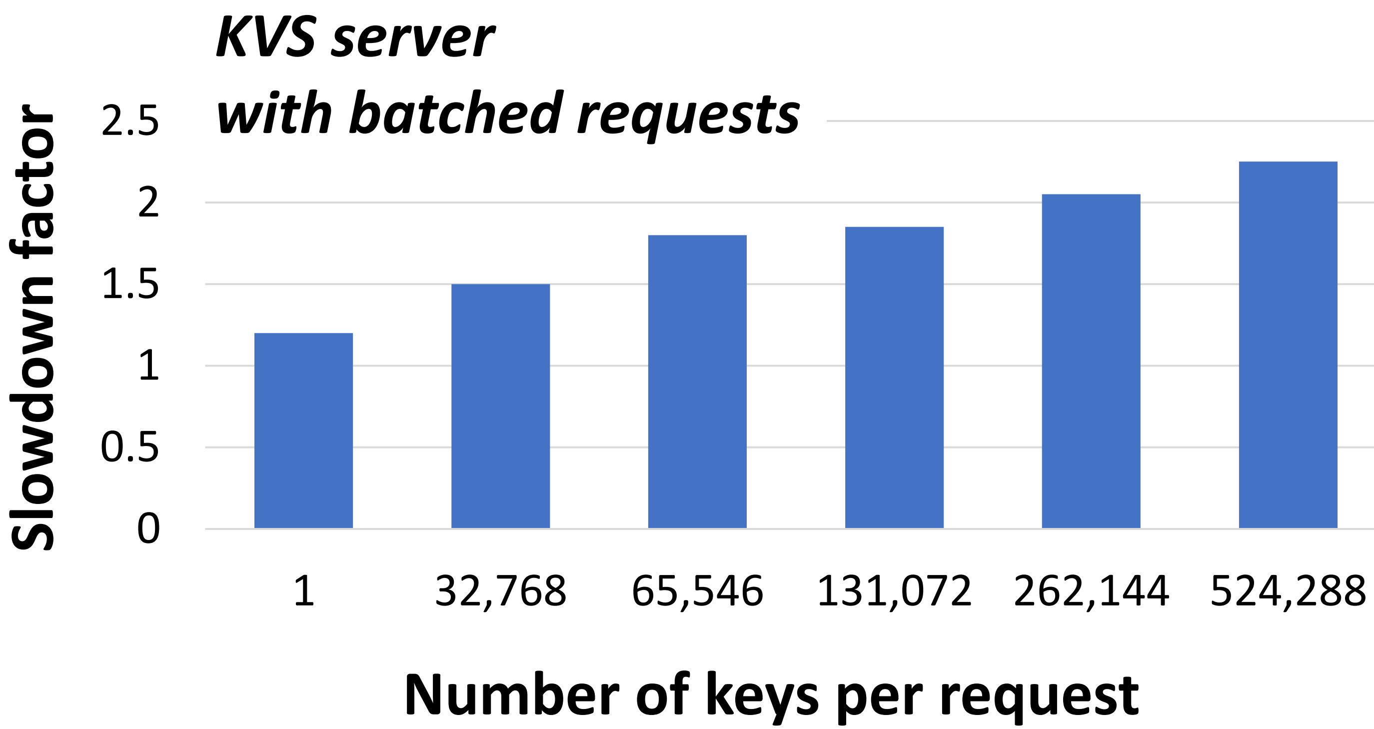
Direct Enclave Costs

- Enclave enter / exit: vs System call:
  3,300 / 3,800 cycles 250 cycles

- LLC misses: 5.6~9.5 X

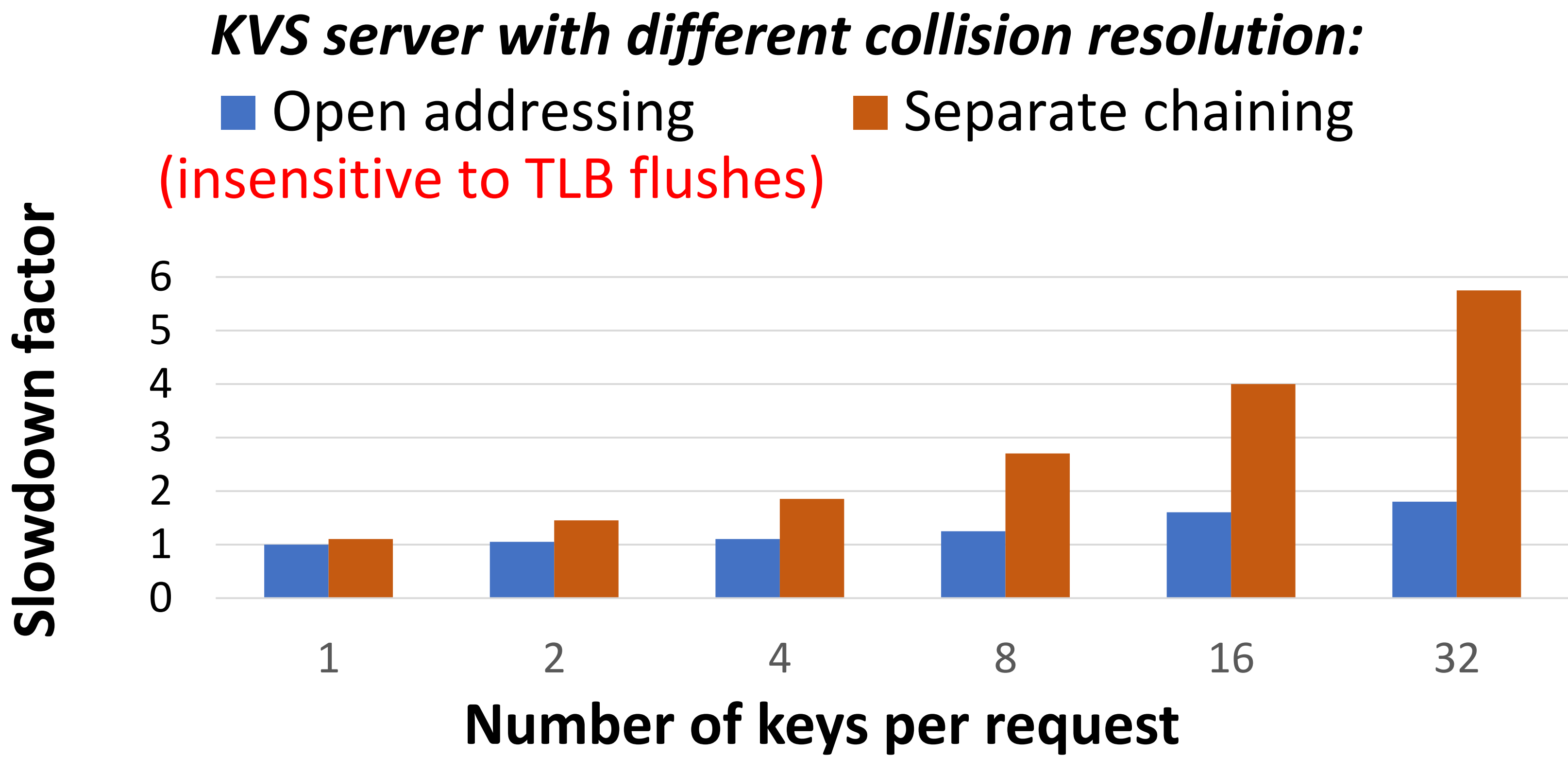
- EPC paging: 40,000 cycles for evict and page-in

Indirect Cost: LLC Pollution



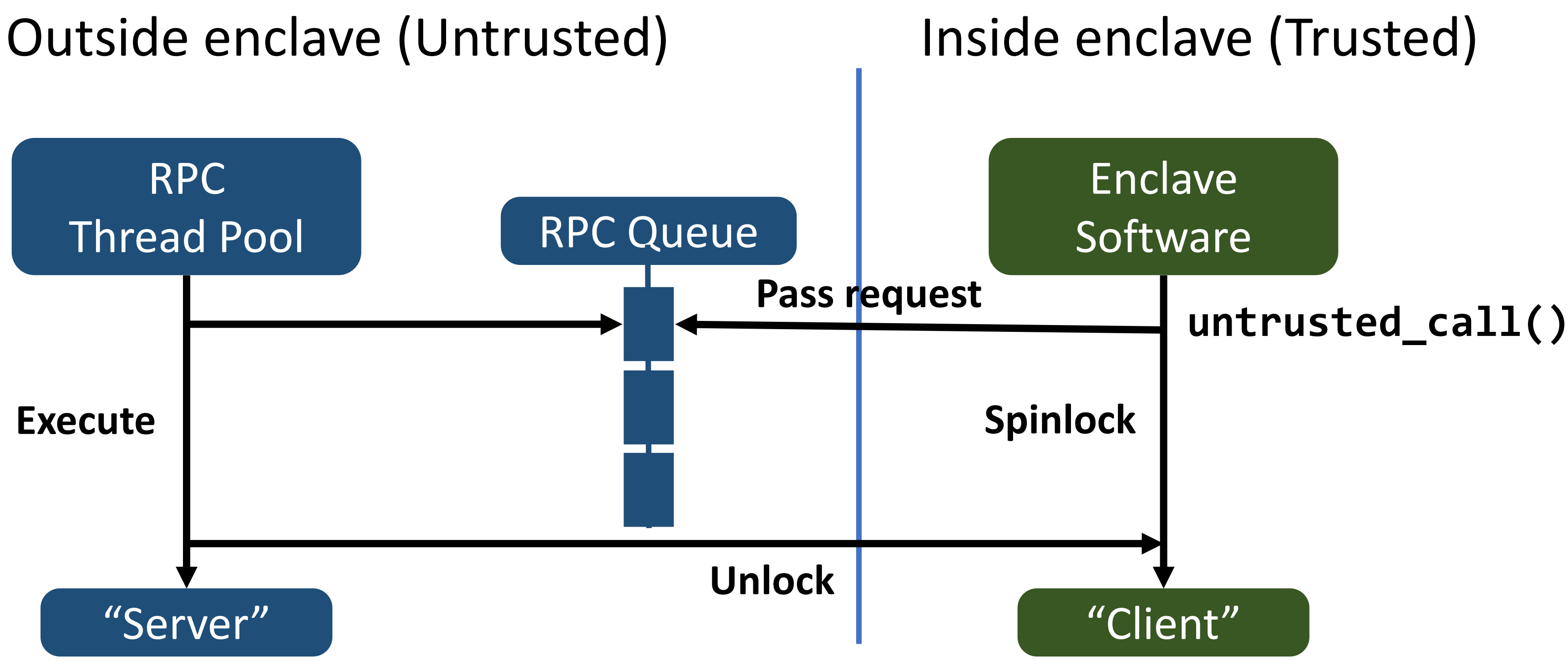
LLC pollution causes up to 2X slowdown

Indirect Cost: TLB Pollution

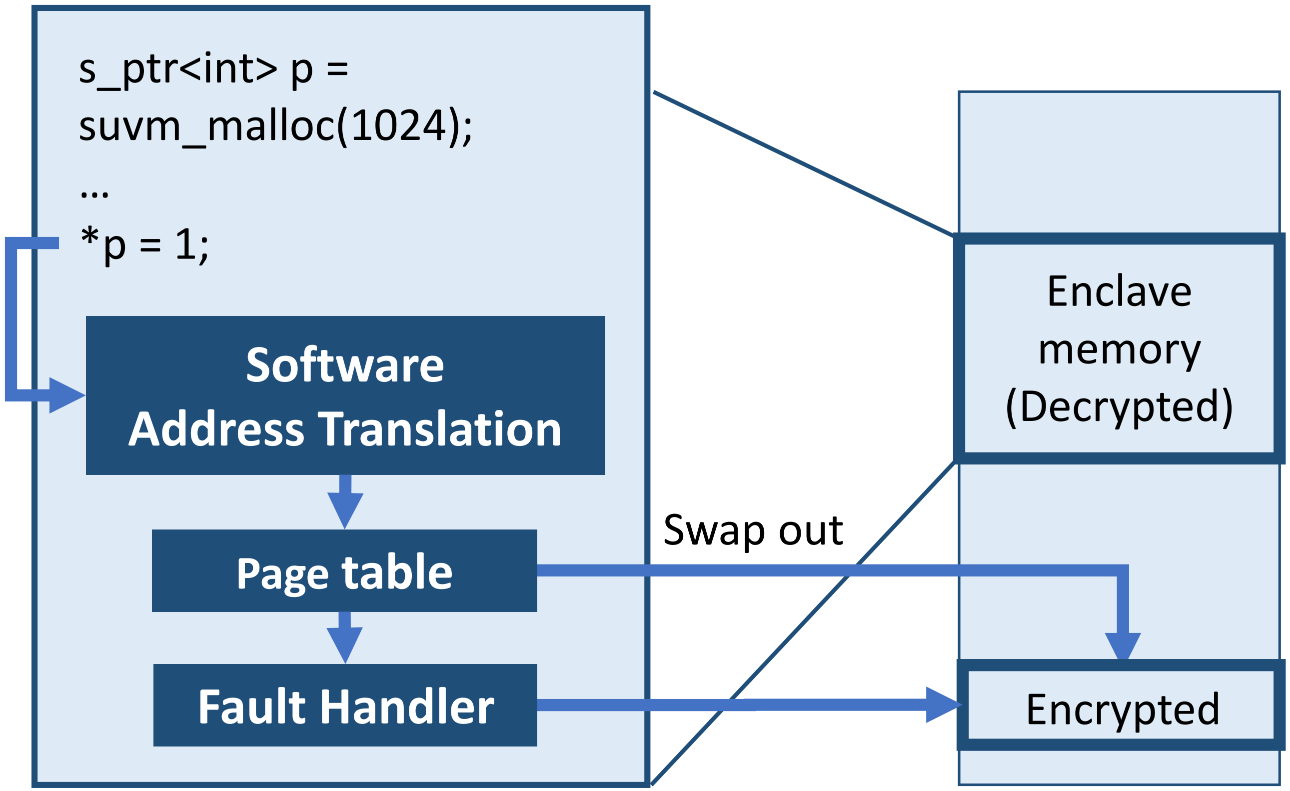


TLB Flushes at every exits cause up to 6X slowdown

RPC-based Enclave Interfaces



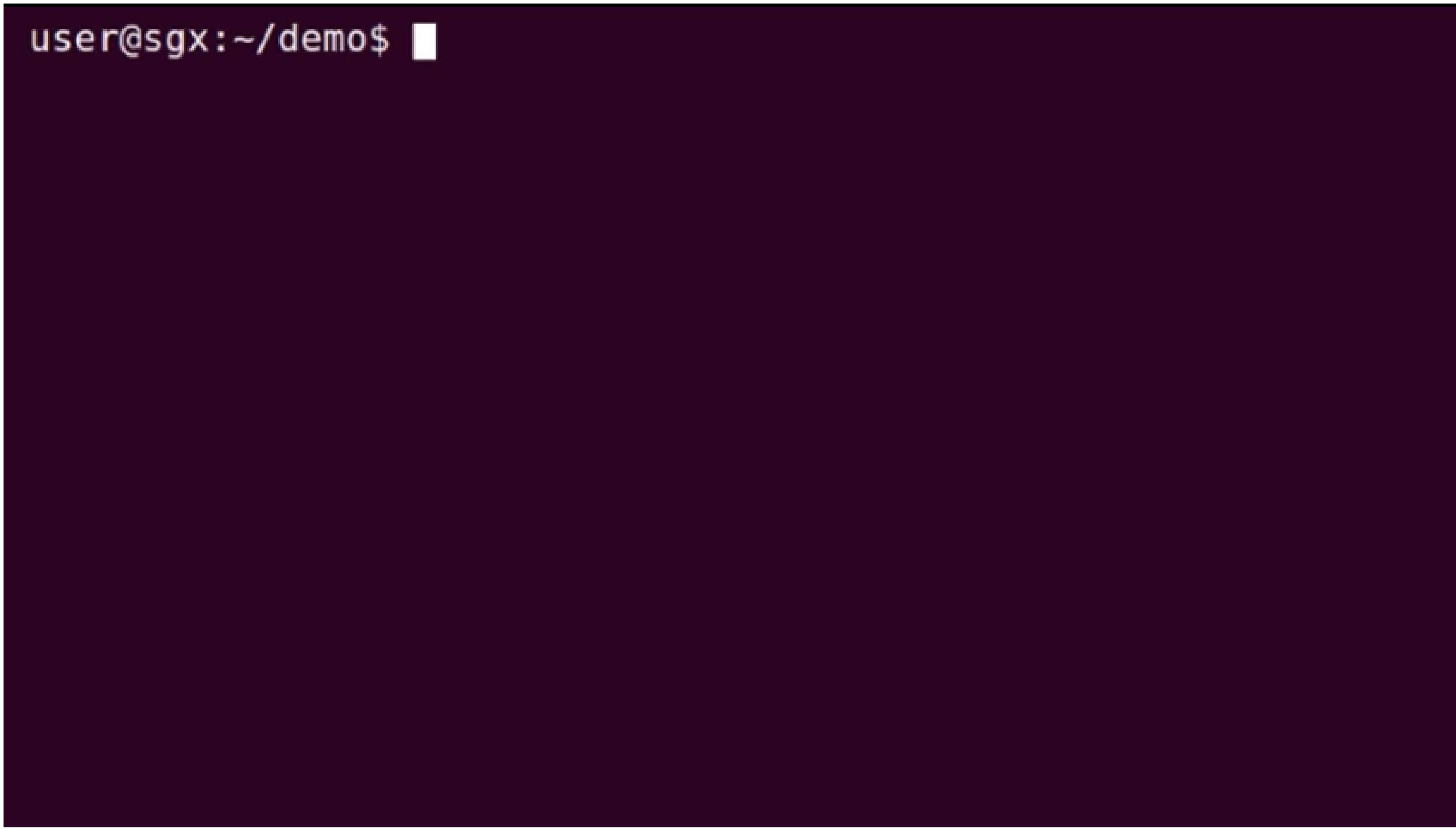
SUVM: Secure User-Space Paging



Eleos keeps EPC footprint static, to avoid fault-based exits

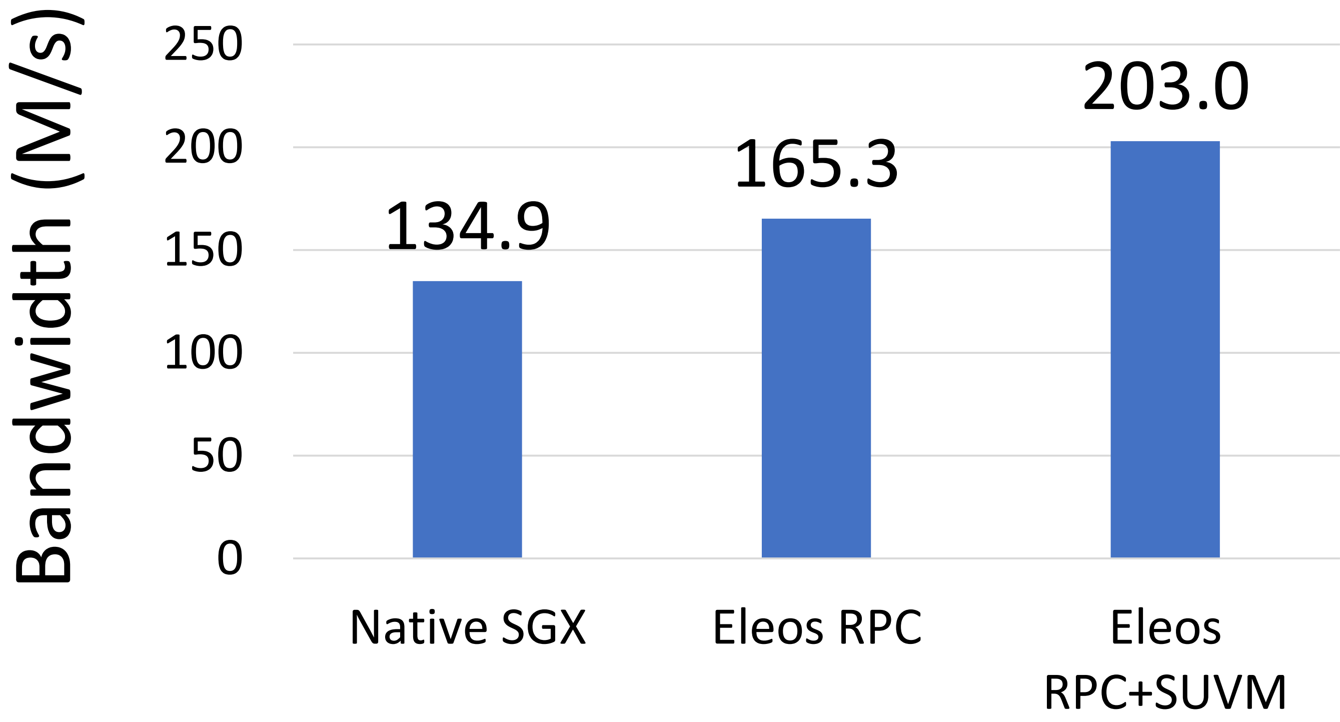
Demo: Memcached on Native SGX

Demo: Memcached with Eleos (RPC)



Demo:
Memcached with Eleos (RPC+SUVM)

Memcached Performance



PRC improves 23%, RPC+SUVM improves 51%

Availability

- Open-source available at: http://github.com/acsl-technion/eleos
- Contact: mark@ee.technion.ac.il

Acknowledgement

Assistance from the following individuals:

• Christof Fetzer	(TU Dresden)
• Li Lei	(Intel Labs)
• Meni Orenbach	(Technion)
• Donald E. Porter	(UNC at Chapel Hill / Fortanix)
• Shweta Shinde	(Natl. Univ. of Singapore)
• Mark Silberstein	(Technion)
• Mona Vij	(Intel Labs)