# SCONE: Secure Linux Containers with Intel SGX

OSDI'2016

Jihoon Jang

Jaeheon Jung

# Contents

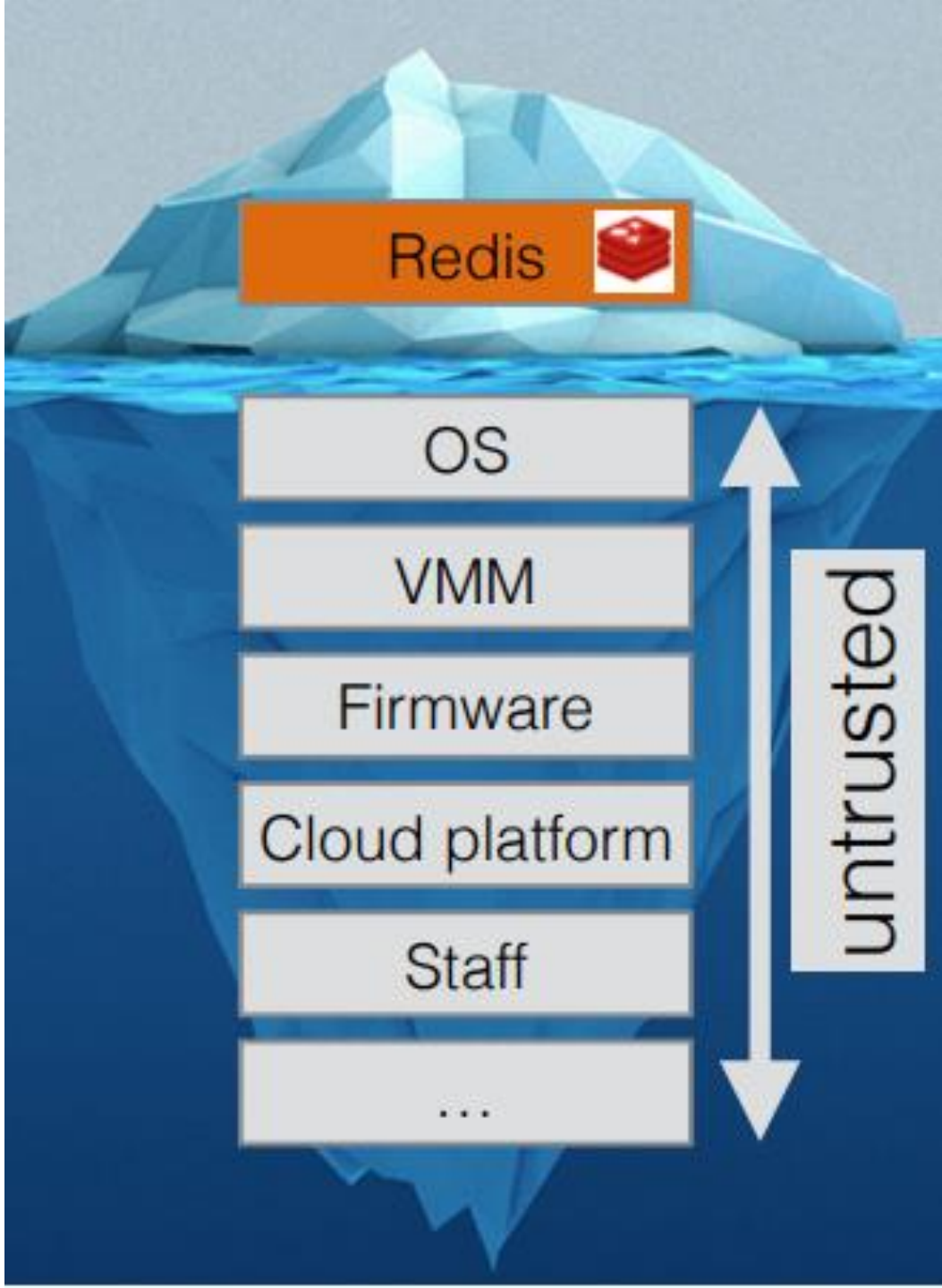
- Introduction
- Secure Containers
- SCONE Design
- Evaluation
- Relative Work
- Conclusion

# Contents

- Introduction
- Secure Containers
- SCONE Design
- Evaluation
- Relative Work
- Conclusion

# Motivation

- Container-based virtualization has become popular recently.
- Users must implicitly trust the cloud provider.
- Users want to protect the **confidentiality** and **integrity** of their application data from unauthorized accesses not only from other containers, but also from higher privileged system software, such as the OS kernel and the hypervisor.



# Goal

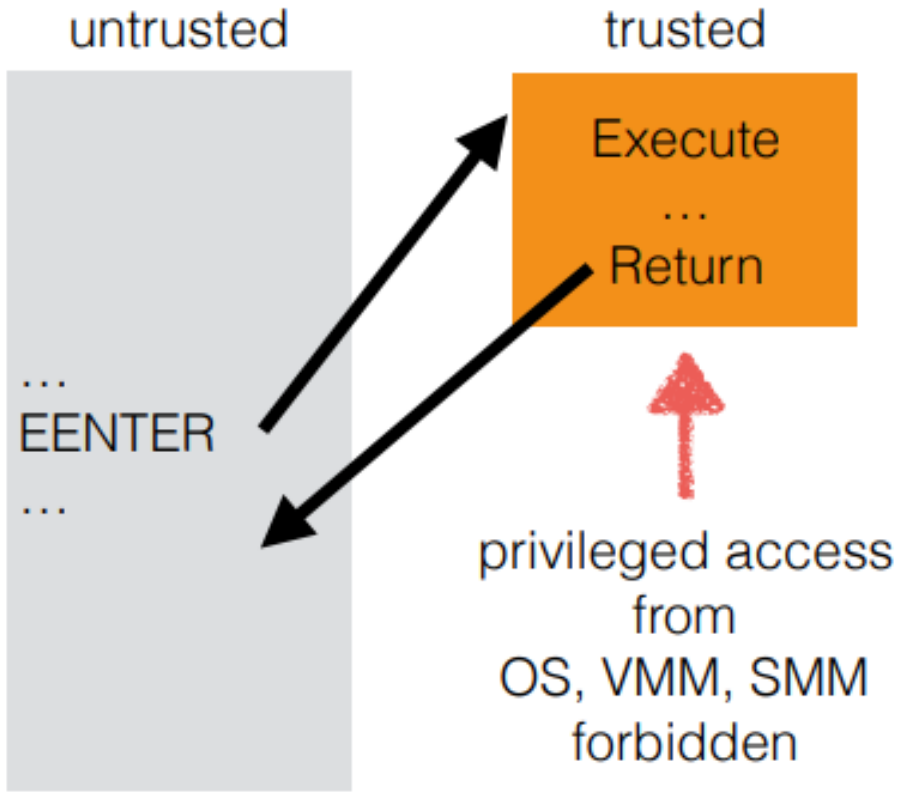
- Run unmodified Linux applications in containers
- in an untrusted clouds securely and,
- with acceptable performance

# Contents

- Introduction
- Secure Containers
- SCONE Design
- Evaluation
- Relative Work
- Conclusion

# Intel Software Guard eXtensions (SGX)

- SGX shields application code and data from access by other software (including higher-privileged software).
- Multiple threads can operate within an enclave, each with its own 4KB thread local storage.
- Enclave memory (Enclave Page Cache) is only accessible from enclave.
- Privileged instructions (e.g., syscall) cannot be executed inside the enclave.

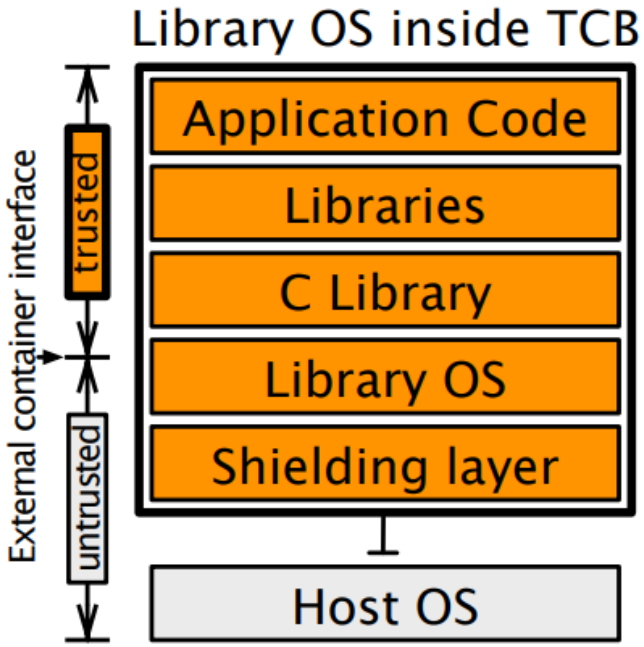


# Performance Overhead of SGX

- Threads must exit the enclave prior to system calls.
  - It causes TLB flush for security reasons, and
  - memory-based enclave arguments must be copied between trusted and untrusted memory.
- Enclave codes pays a penalty for writes to memory and cache misses.
  - Memory Encryption Engine (MEE) must encrypt and decrypt cache lines.
- Applications whose memory requirements exceed the EPC size must swap pages between the EPC and unprotected DRAM.
  - Evicted EPC pages must be encrypted and integrity-protected before being copied to outside DRAM.

# Design trade-offs

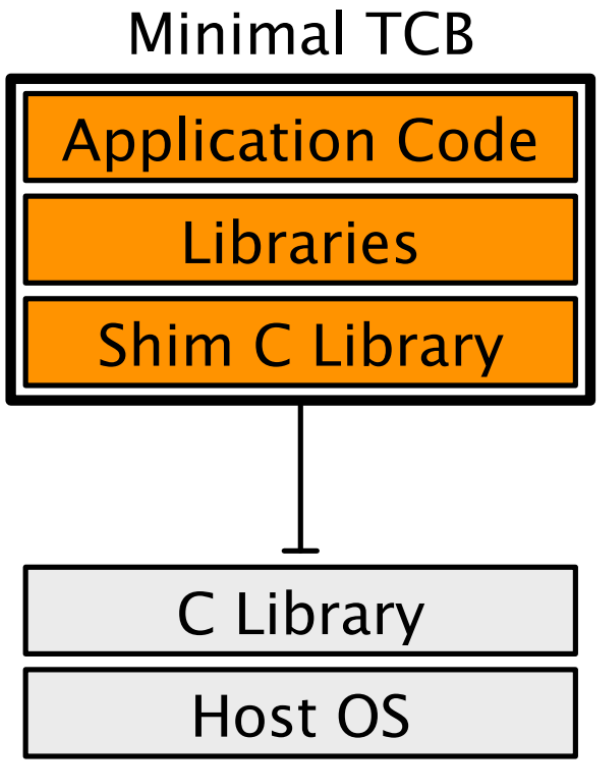
1. External container interface




- Haven (OSDI'14): library operating system in enclave
- It increases the trusted computing base (TCB) size inside of the enclave.
- It may add a performance overhead due to the extra abstraction introduced by the library OS.

# Design trade-offs

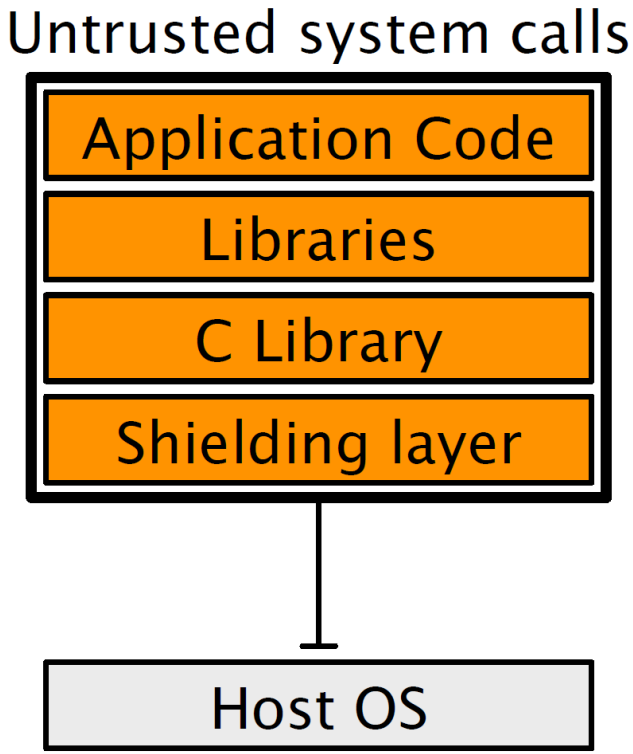
1. External container interface




- Opposite, extreme design point: C library outside enclave
- Hard to protect the confidentiality and integrity of application data whilst exposing a wide interface

# Design trade-offs

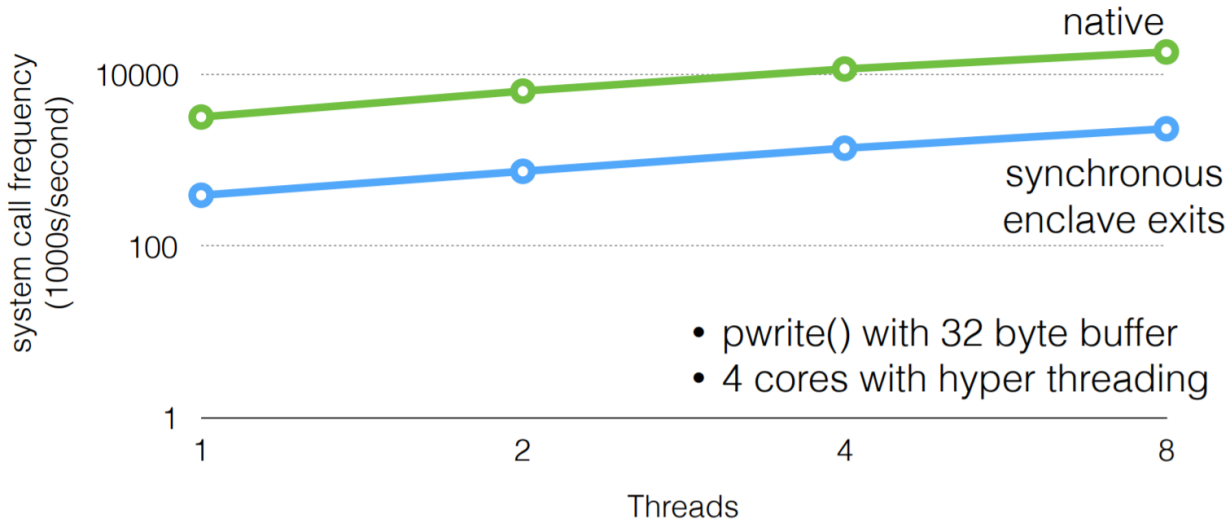
1. External container interface




- Defining the external interface at the level of system calls executed by the libc implementations
- Shield libraries can be used to protect a security-sensitive set of system calls: file descriptor based I/O calls, such as read, write, send, and recv.

# Design trade-offs

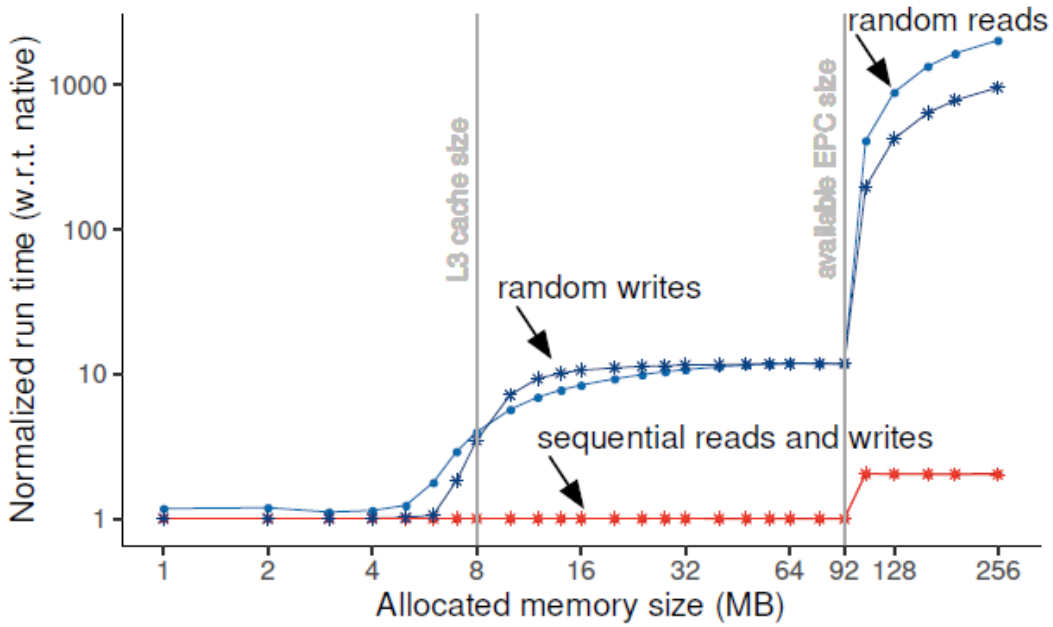
2. System Call Overhead




Secure container design must **go beyond simple synchronous support for system calls** implemented using thread transitions.

# Design trade-offs

3. Memory access overhead

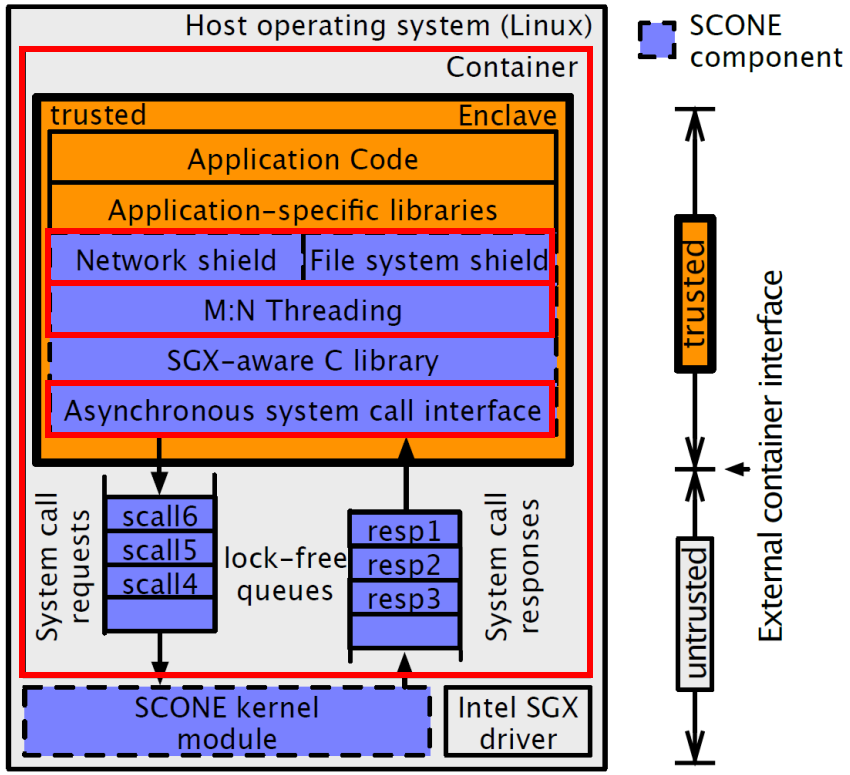



Secure container should use untrusted non-enclave memory **as much as possible**, without compromising the offered security guarantees.

# Contents

- Introduction
- Secure Containers
- SCONE Design
- Evaluation
- Relative Work
- Conclusion

# SCONE Architecture




1) exposes an external interface based on system calls to the host OS, which is shielded from attacks.
2) implements M:N threading to avoid the cost of unnecessary enclave transitions.
3) offers container processes an asynchronous system call interface to the host OS.
4) integrates with existing Docker container environments, and ensures that secure containers are compatible with standard Linux containers.

# External interface shielding

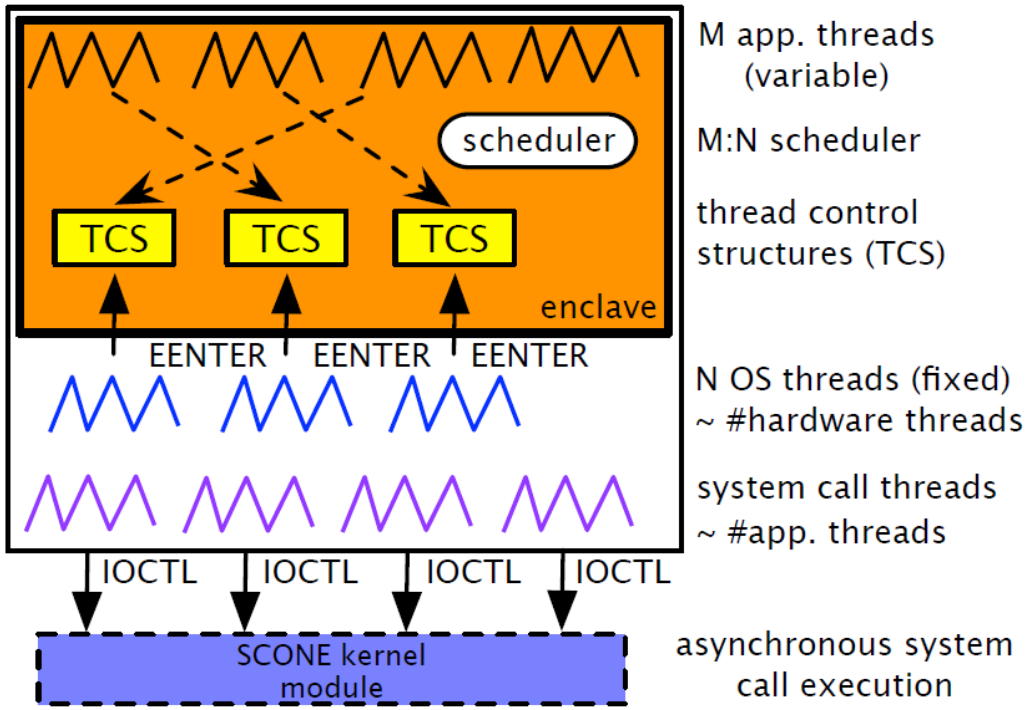
- To protect services created under assumption that the underlying OS is trusted, SCONE supports a set of shield.
- File system shield
  - protects the confidentiality and integrity of files: files are authenticated and encrypted, transparently to the service.
  - splits files into blocks of fixed sizes, keeping an authentication tag and a nonce in a metadata file for each block.
  - supports a dedicated secure ephemeral file system as a lightweight alternative of read-only file system such as Docker `tmpfs`.

# External interface shielding

- Network shield
  - permits clients to establish secure tunnels to container services using TLS.
  - wraps all socket operations and redirects them to a network shield.
  - The private key and certificate are read from the container’s file system.
- Console shield
  - protects the confidentiality of data sent via the stdin, stdout, and stderr streams.
  - The symmetric encryption key is exchanged between the secure container and the SCONE client during the startup procedure.

# M:N Threading model

- SCONE supports an M application threads inside the enclave
- Application threads are mapped to N OS threads, and each thread executes the scheduler.
- SCONE also has multiple system call threads inside the SCONE kernel module to prevent application threads from being blocked by system calls.



# Async System calls

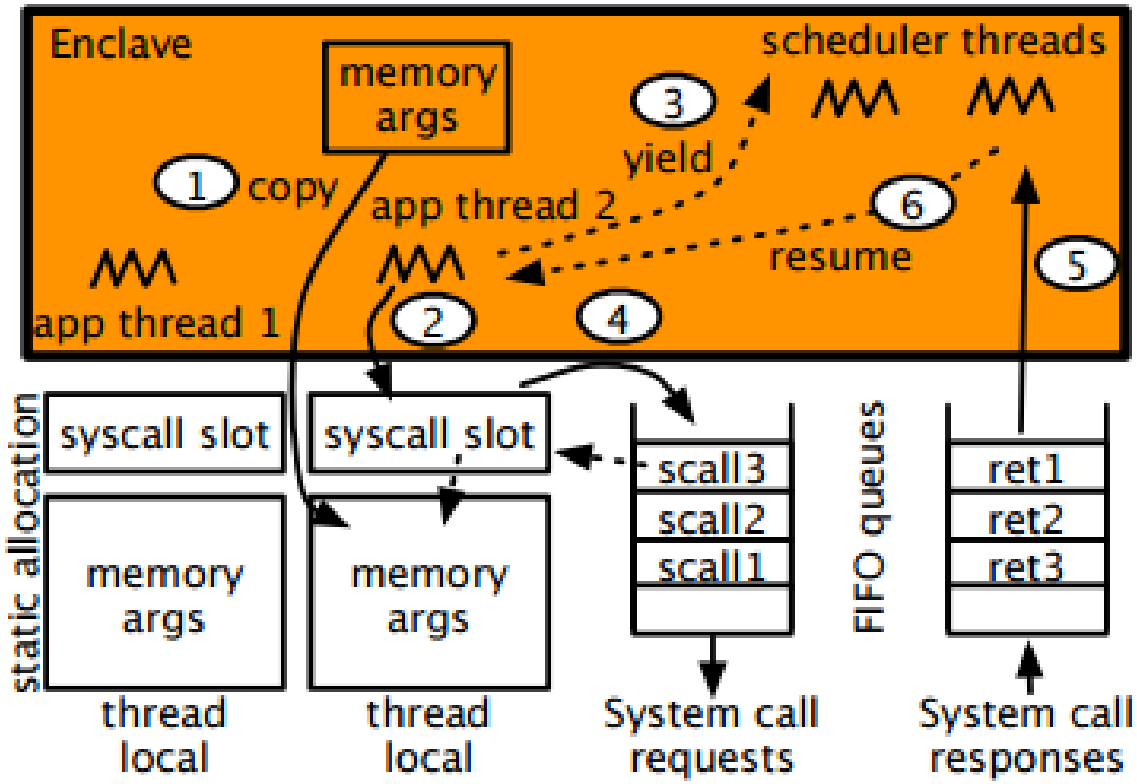



**Figure 6: Asynchronous system calls**

# Integrate with Docker

- chose to integrate SCONE with Docker because it is the most popular and widely used container platform.
- Does not require changes to be made to the Docker Engine or the Docker API
- Uses a wrapper around the Docker client, Scone client.
- A secure SCONE client is used to create configuration files and launch containers in an untrusted environment.
- SCONE supports a typical Docker workflow

# Workflow with Docker

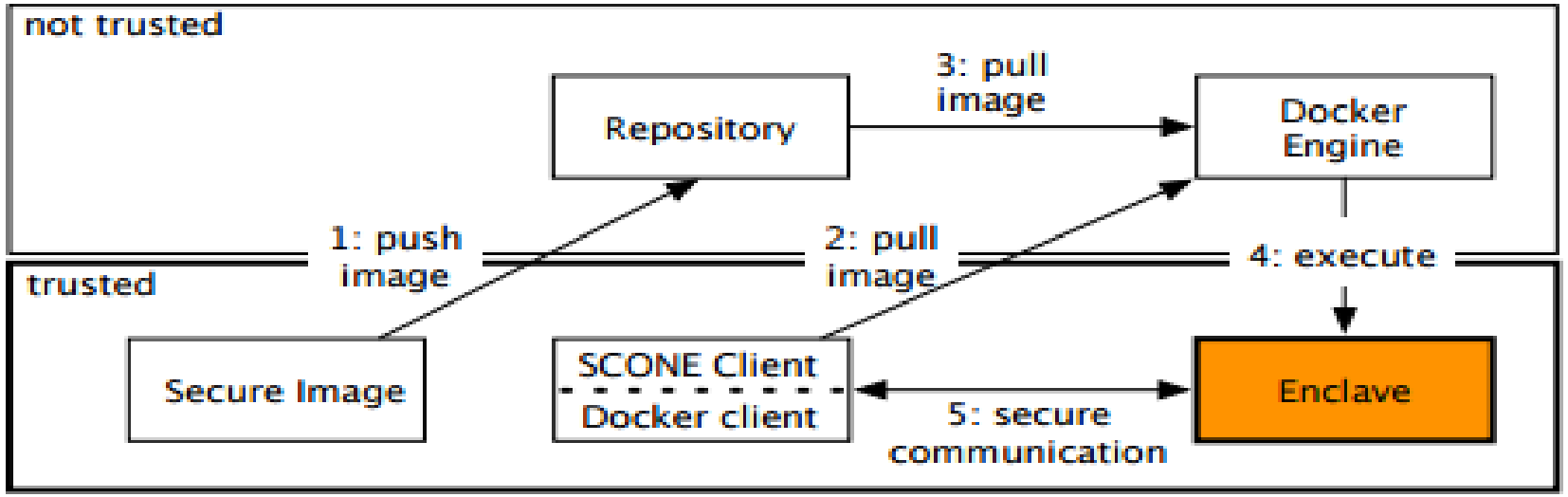



**Figure 7: Using secure containers with Docker**

# Container Startup

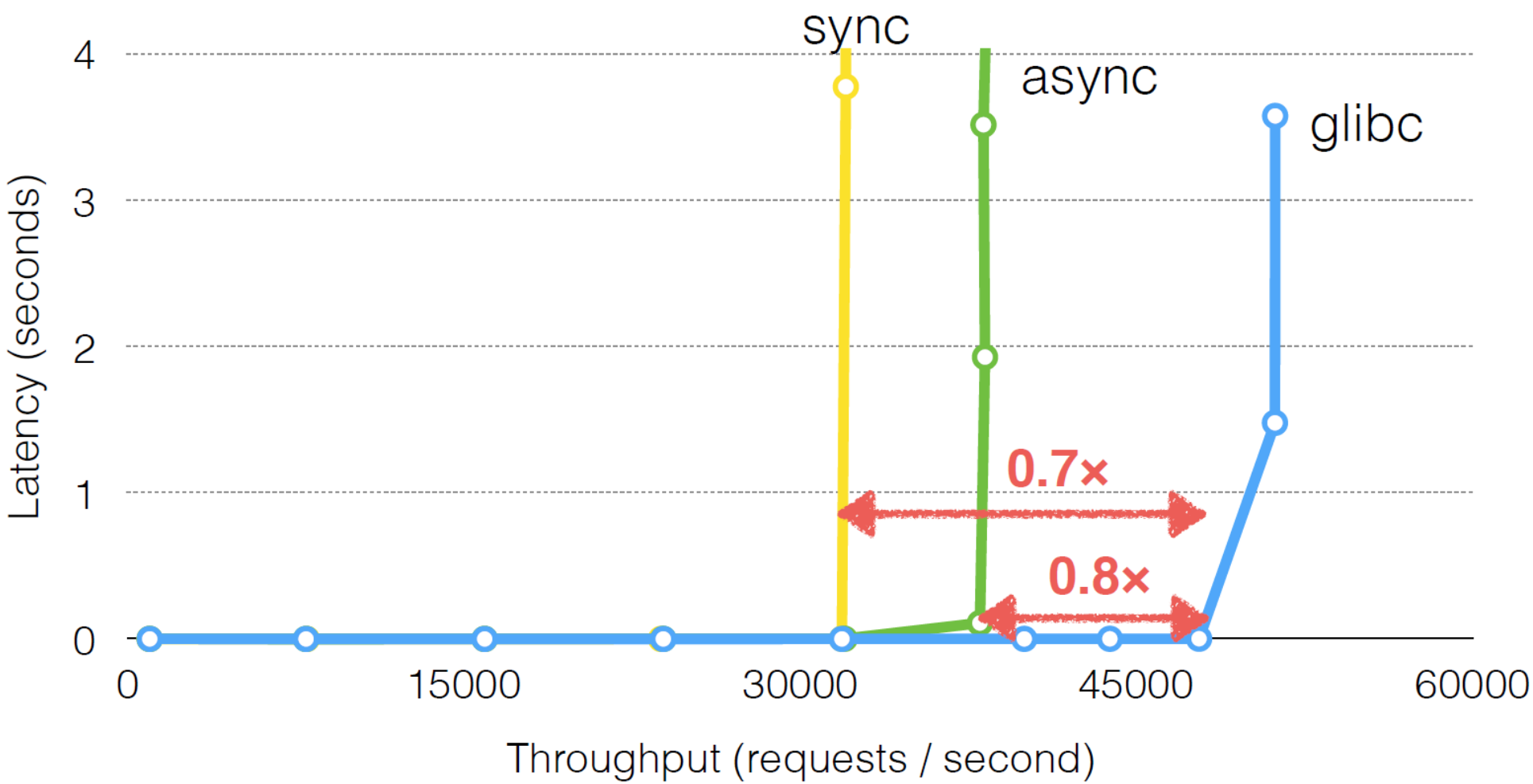
- Each secure container requires a startup configuration file (SCF). The SCF contains keys to encrypt standard I/O streams, a hash of the FS protection file and its encryption key, application arguments and environment variables.

- Only an enclave whose identity has been verified can access the SCF. The SCF is received through a TLS protected network connection, established during enclave startup

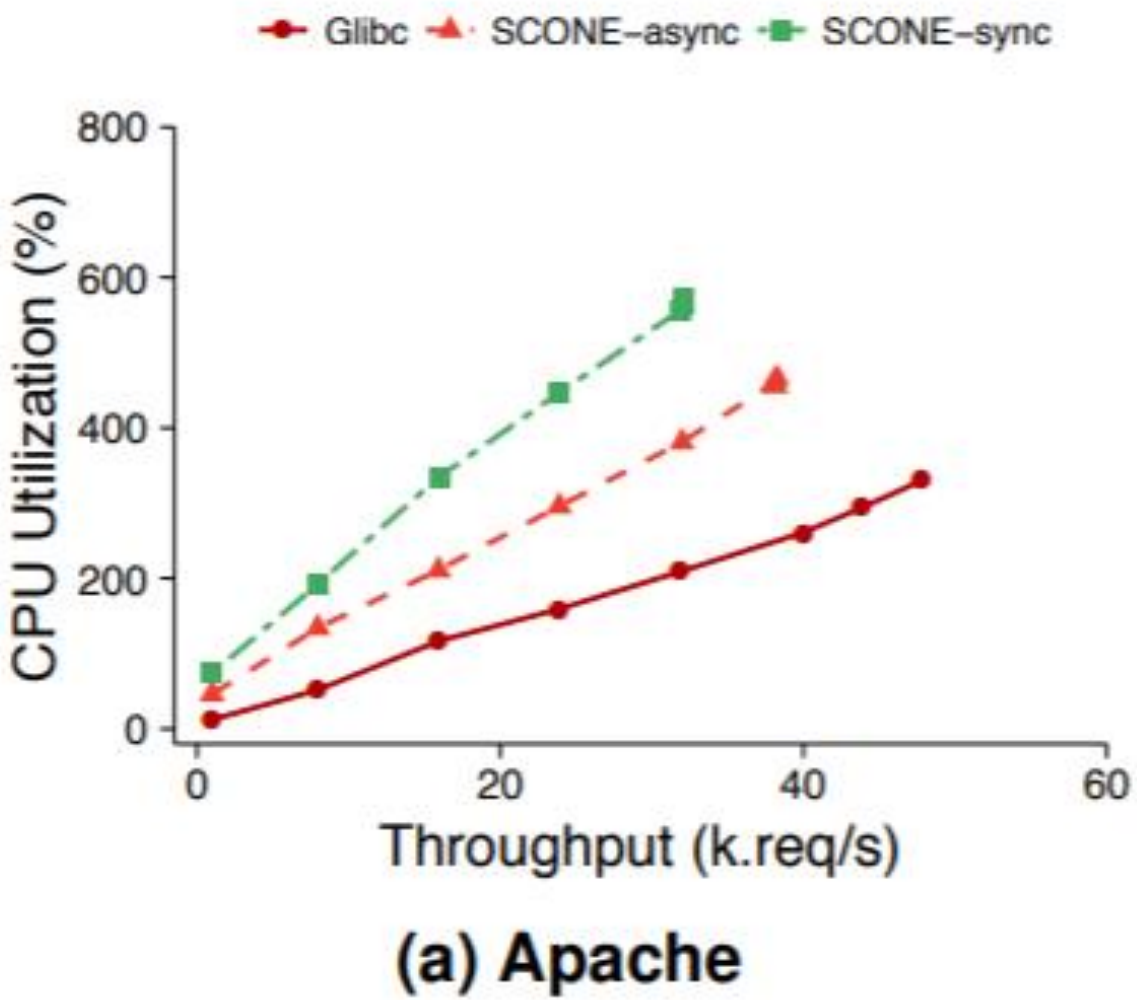
# Contents

- Introduction
- Secure Containers
- SCONE Design
- Evaluation
- Relative Work
- Conclusion

# Apache Throughput



# Application benchmark




**(a) Apache**

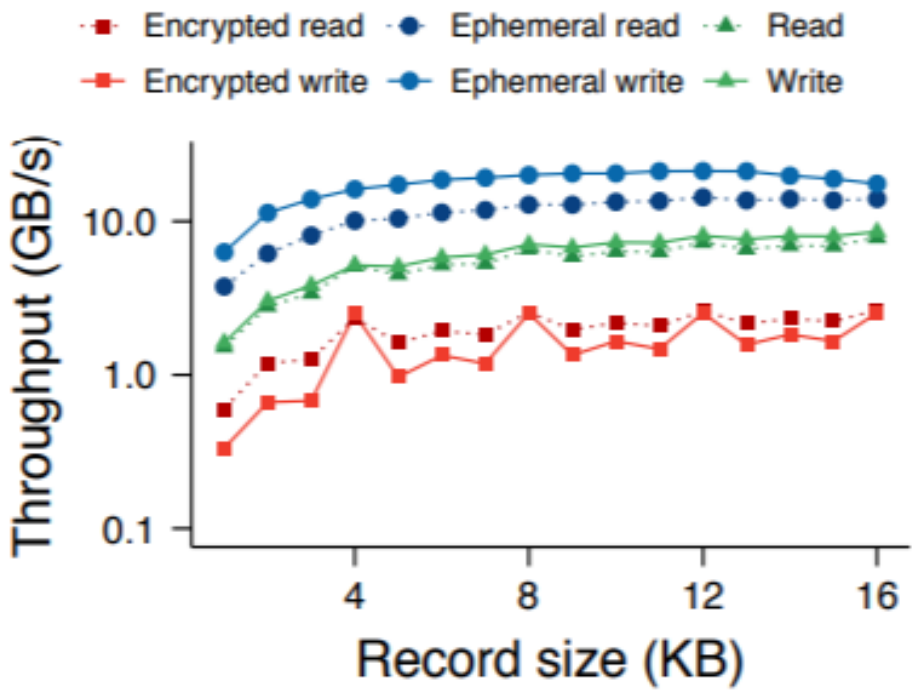



**Figure 12: Throughput of random reads/writes with ephemeral file system versus `tmpfs`**

# Contents

- Introduction
- Secure Containers
- SCONE Design
- Evaluation
- Relative Work
- Conclusion

# Related Work

- Software protection against privileged code
  1. Initial work such as NGSCB and Proxos executes untrusted and trusted OSs side-by side using virtualization, with security-sensitive applications hosted by the trusted OS.
  2. Subsequent work, including Overshadow, SP3, InkTag and Virtual Ghost, has focused on reducing the size of the TCB by directly protecting application memory from unauthorized OS accesses.

- Trusted hardware
    1. Secure co-processors offer tamper-proof physical isolation and can host arbitrary functionality.
    2. Trusted platform modules (TPM) offer tailored services for securing commodity systems.
    3. ARM TrustZone has two system personalities, secure and normal world. This split meets the needs of mobile devices in which a rich OS must be separated from the system software controlling basic operations.
    4. Intel SGX

# Contents

# Conclusion

- SCONE increases the confidentiality and integrity of containerized services using Intel SGX.
- The secure containers of SCONE feature a TCB of 60% to 200% of the application code size and are compatible with Docker.
- Using asynchronous system calls and a kernel module, SGX-imposed enclave transition overheads are reduced effectively

# Thank you

Questions ?