Microsoft

Research Faculty Summit 2018

Systems | Fueling future disruptions

Microsoft

Confidential Computing

Peter Pietzuch
Massachusetts Institute of Technology

Imperial College London

LSDS
Large-Scale Data & Systems Group

Designing Systems Support for Trusted Execution using Intel SGX

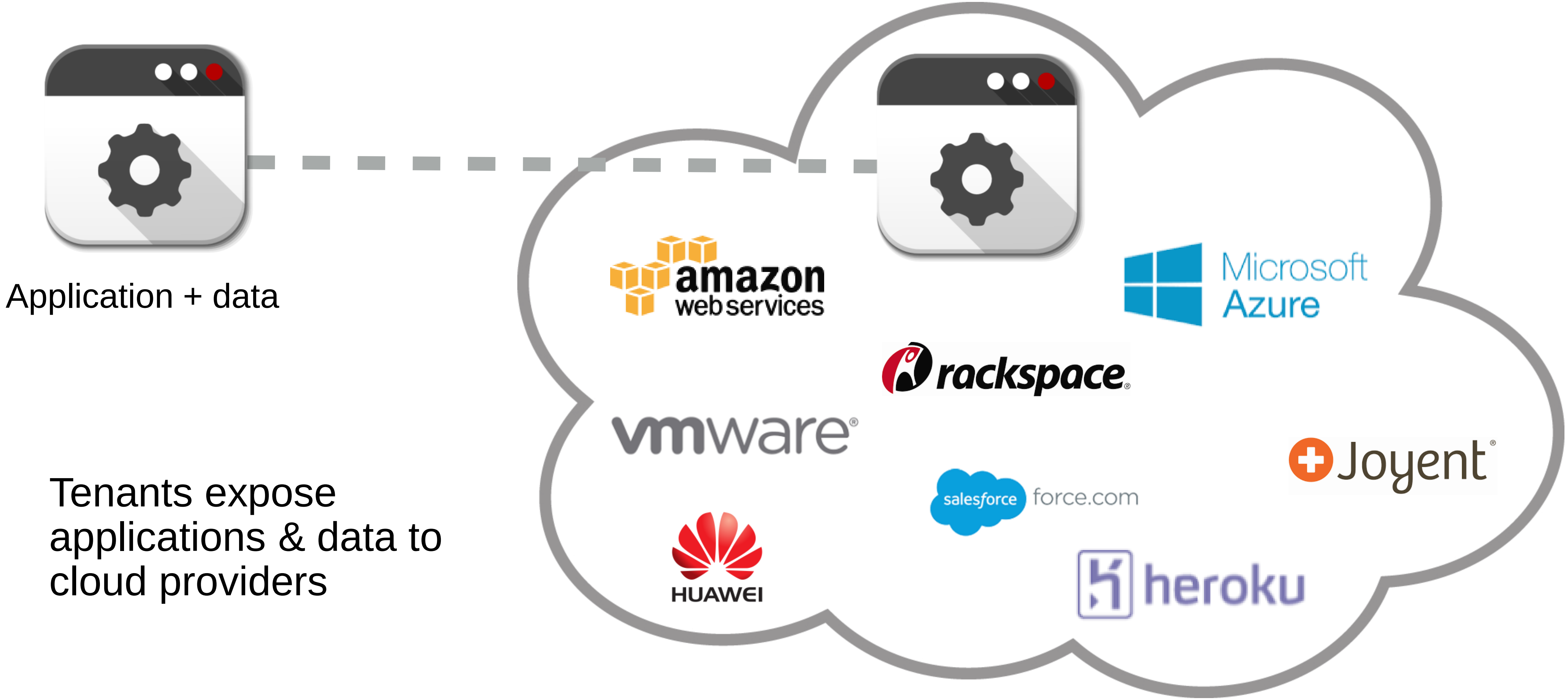
Peter Pietzuch

Imperial College London

http://lsds.doc.ic.ac.uk
<prp@imperial.ac.uk>

Microsoft Research Faculty Summit – Redmond, WA, USA – August 2018

Cloud Tenants Must Trust Cloud Providers



Tenants expose applications & data to cloud providers

Tenants View Clouds as Untrusted Black Boxes

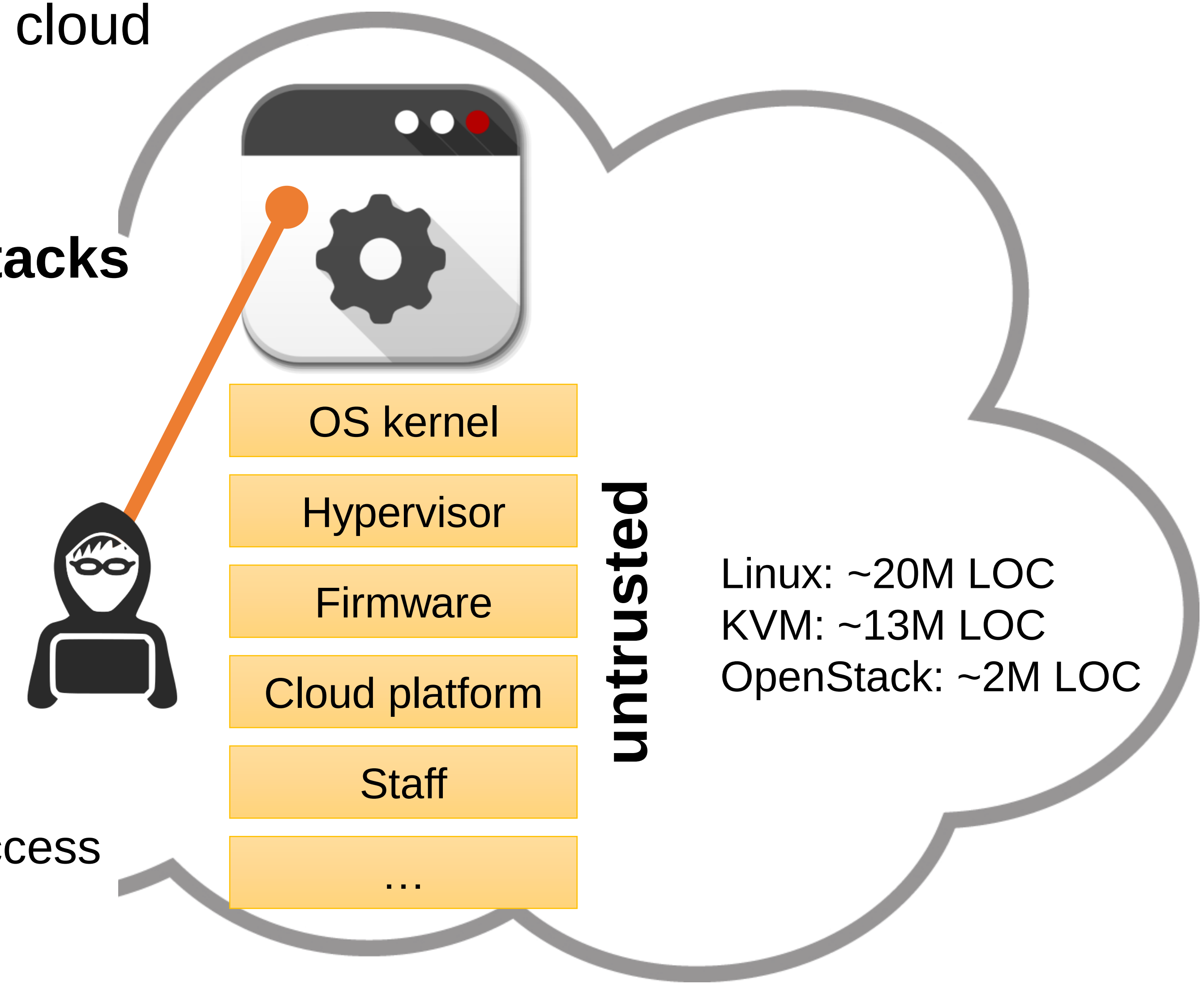
Cloud providers operate complex cloud stacks

Threats from **privileged code attacks**

- Security vulnerabilities exist:
  - Xen hypervisor: 184 (2012-16)
  - Linux kernel: 721 (2012-16)
- Many attacks exploit vulnerabilities
  - Control-flow hijacking, code injection attacks, return-oriented programming

Threats from **insider attacks**

- Administrators, staff with physical access



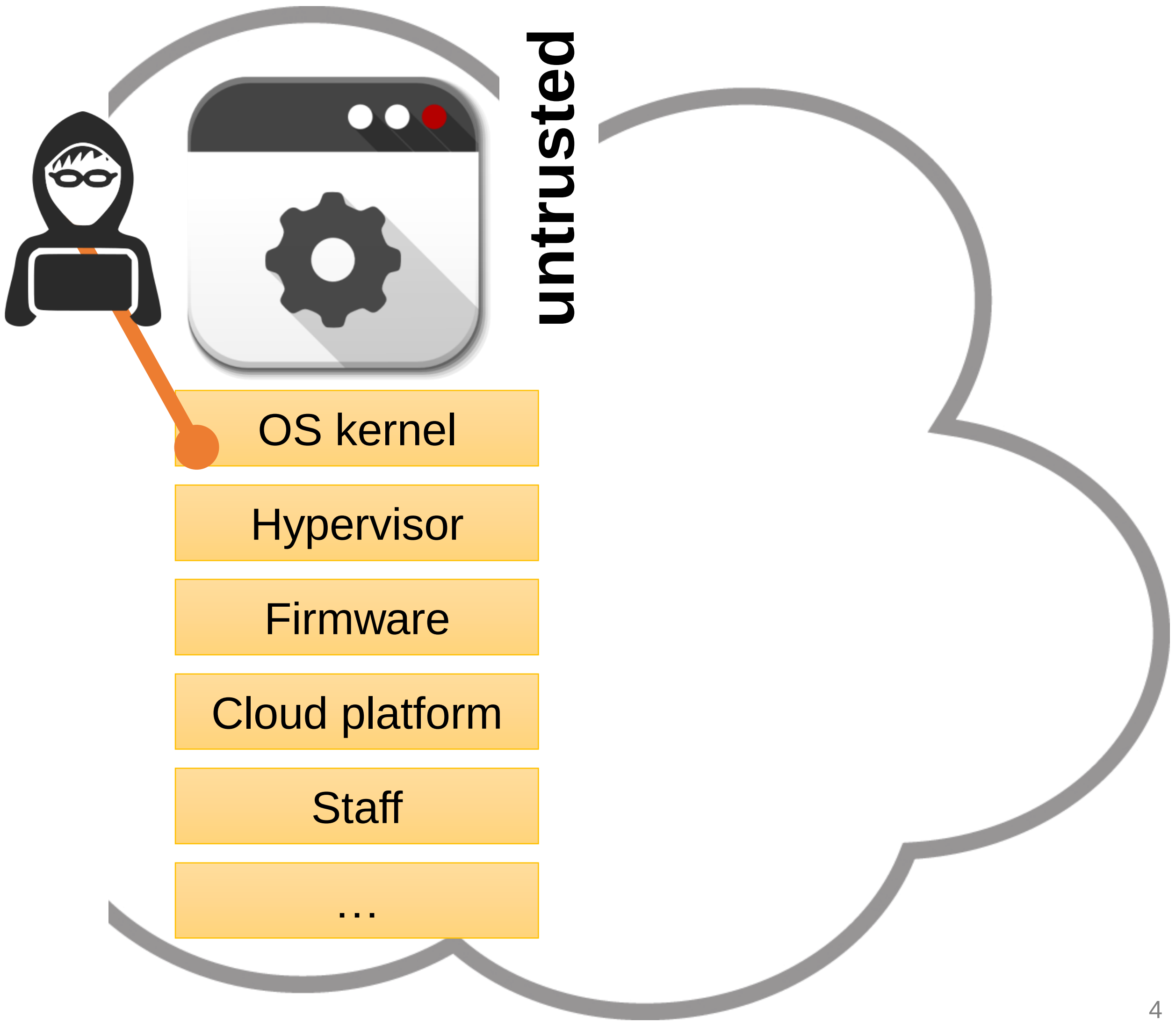
Existing Cloud Security Model Doesn't Help…

Cloud providers do not trust cloud tenants

Cloud security mechanisms focus on protecting privileged system software (OS, hypervisor)

– e.g. tenant isolation using VMs

☛ **Trusted execution gives control over security to cloud tenants**



Trusted Execution with Intel SGX

Introduces concept of **userspace enclaves**

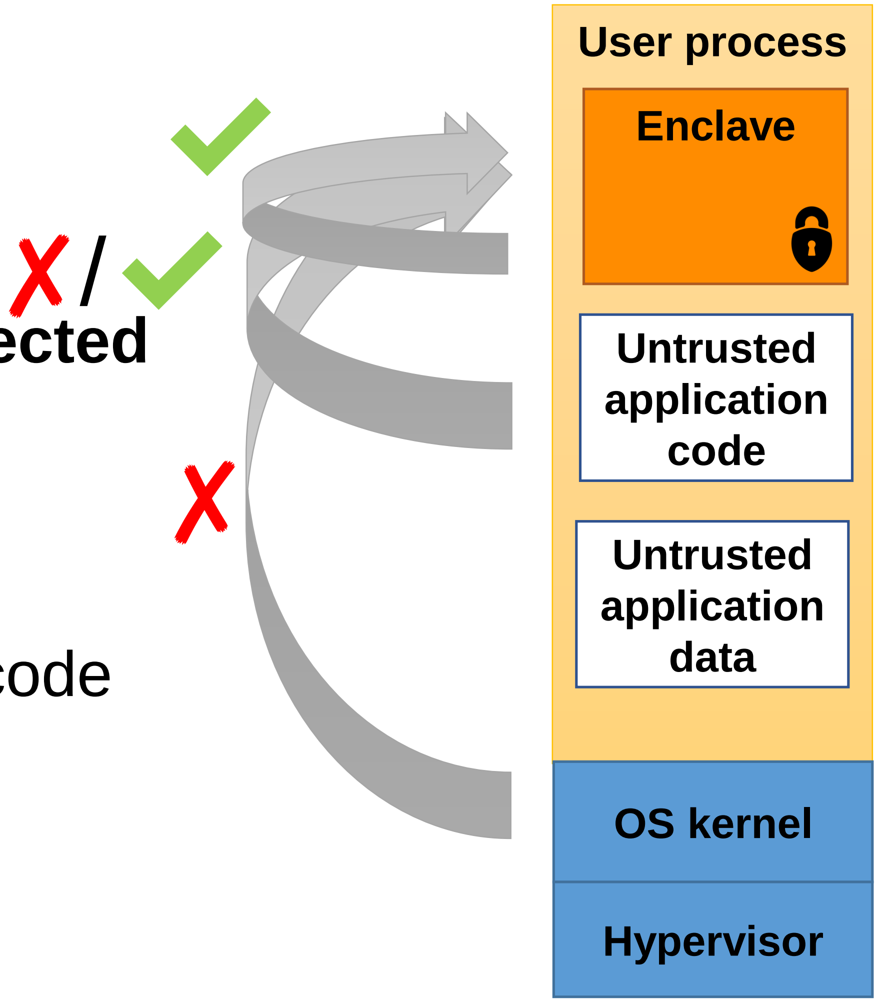
- Isolated memory regions for code and data

Enclave memory **encrypted & integrity-protected**

- Automatically performed by the hardware

Enclave memory only accessible by enclave code

- Protected from privileged code (OS, hypervisor)

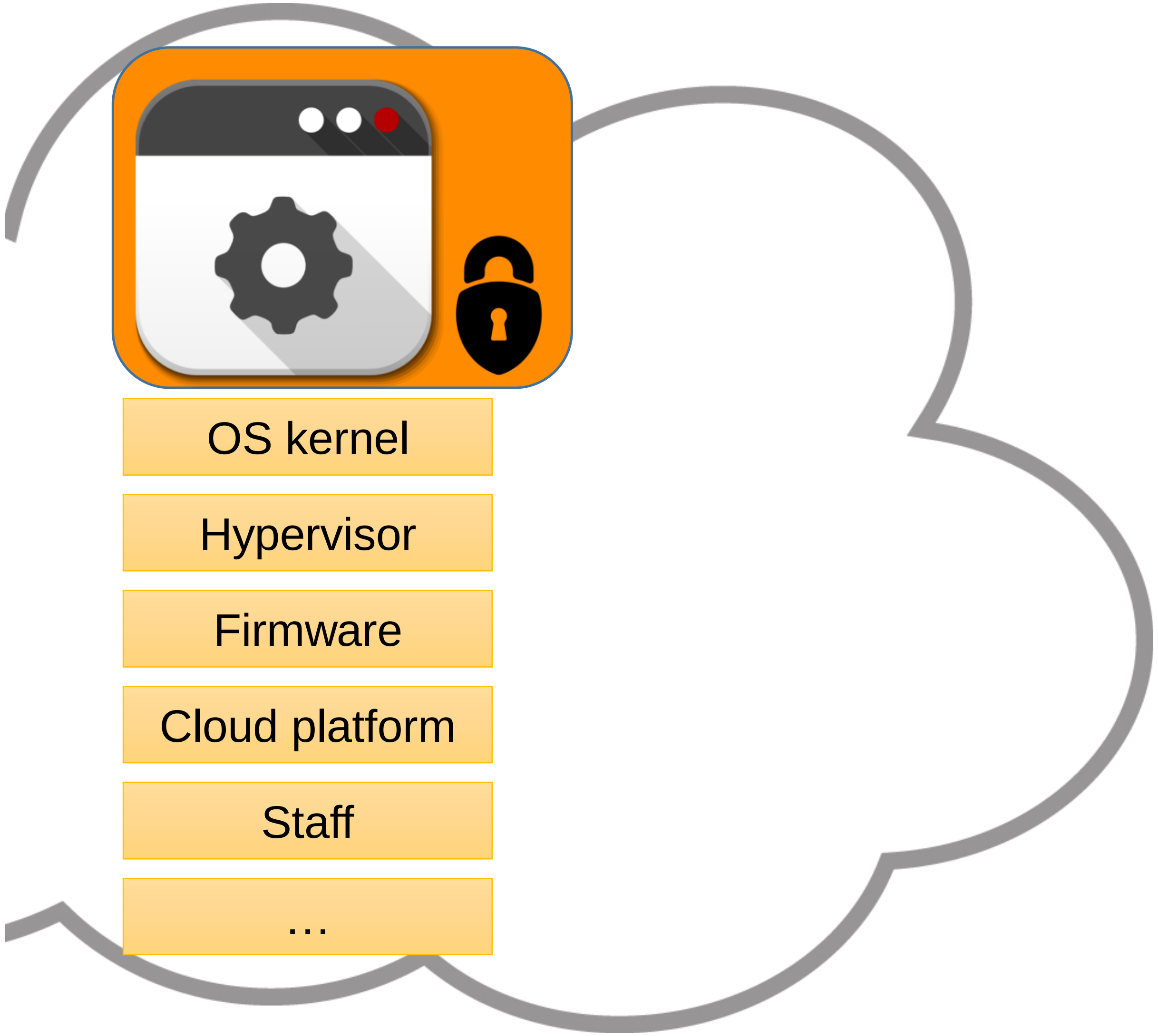


Promise: Protect Cloud Tenants using Enclaves

Enclave retains flexibility to run arbitrary cloud applications

- Unlike approaches based on software encryption, homomorphic encryption, secure multi-party computation, …

☛ **How to support cloud applications inside SGX enclaves?**

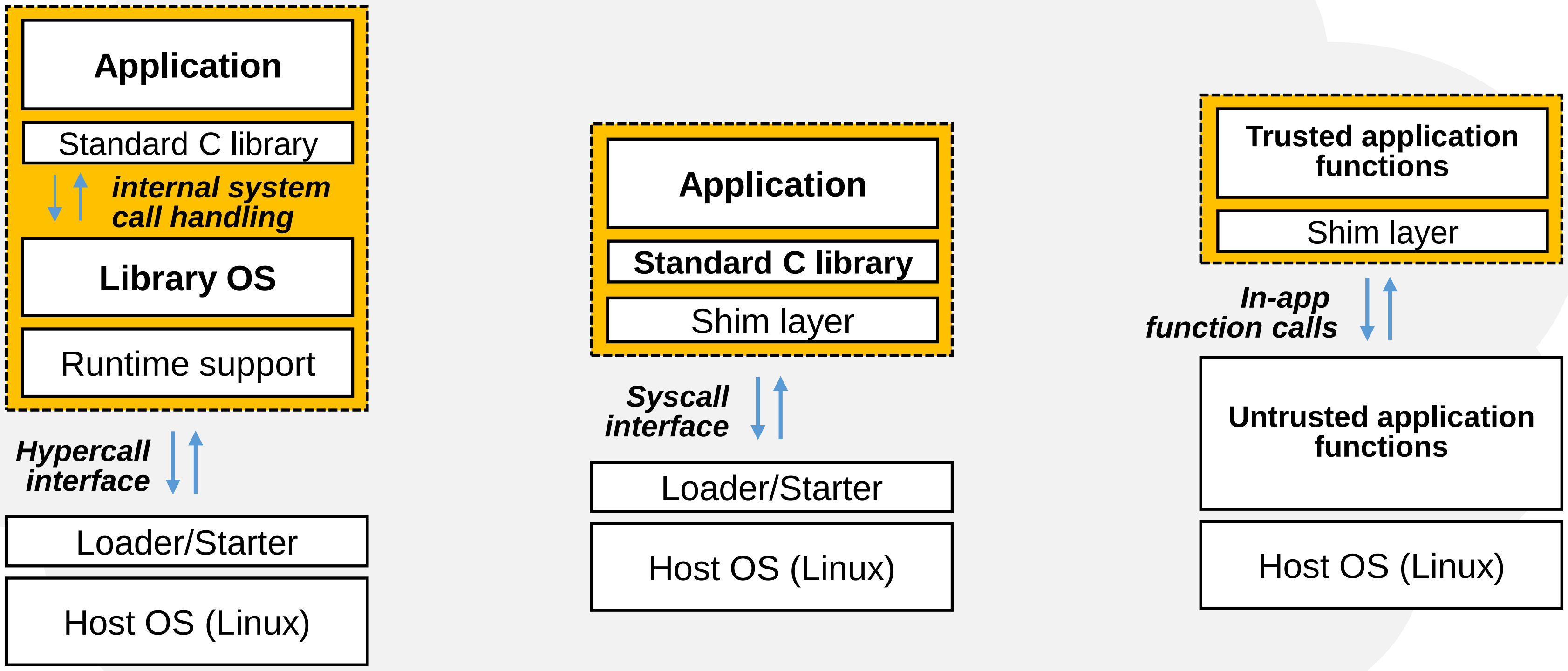


Design Space: Systems Support for SGX Enclaves

(a) **Library OS:**
SGX-LKL, Haven

(b) **System call interface:**
SCONE **[OSDI'16]**

(c) **Partitioned application:**
Glamdring **[USENIX ATC'17]**, Intel SGX SDK



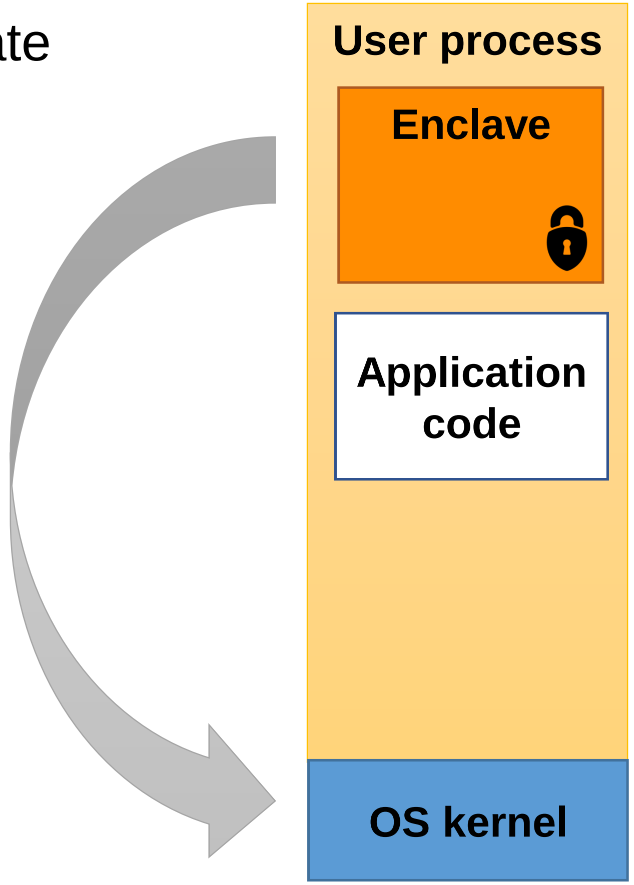
Challenge: Enclave Transitions are Expensive

Entering/exiting an enclave comes with performance cost

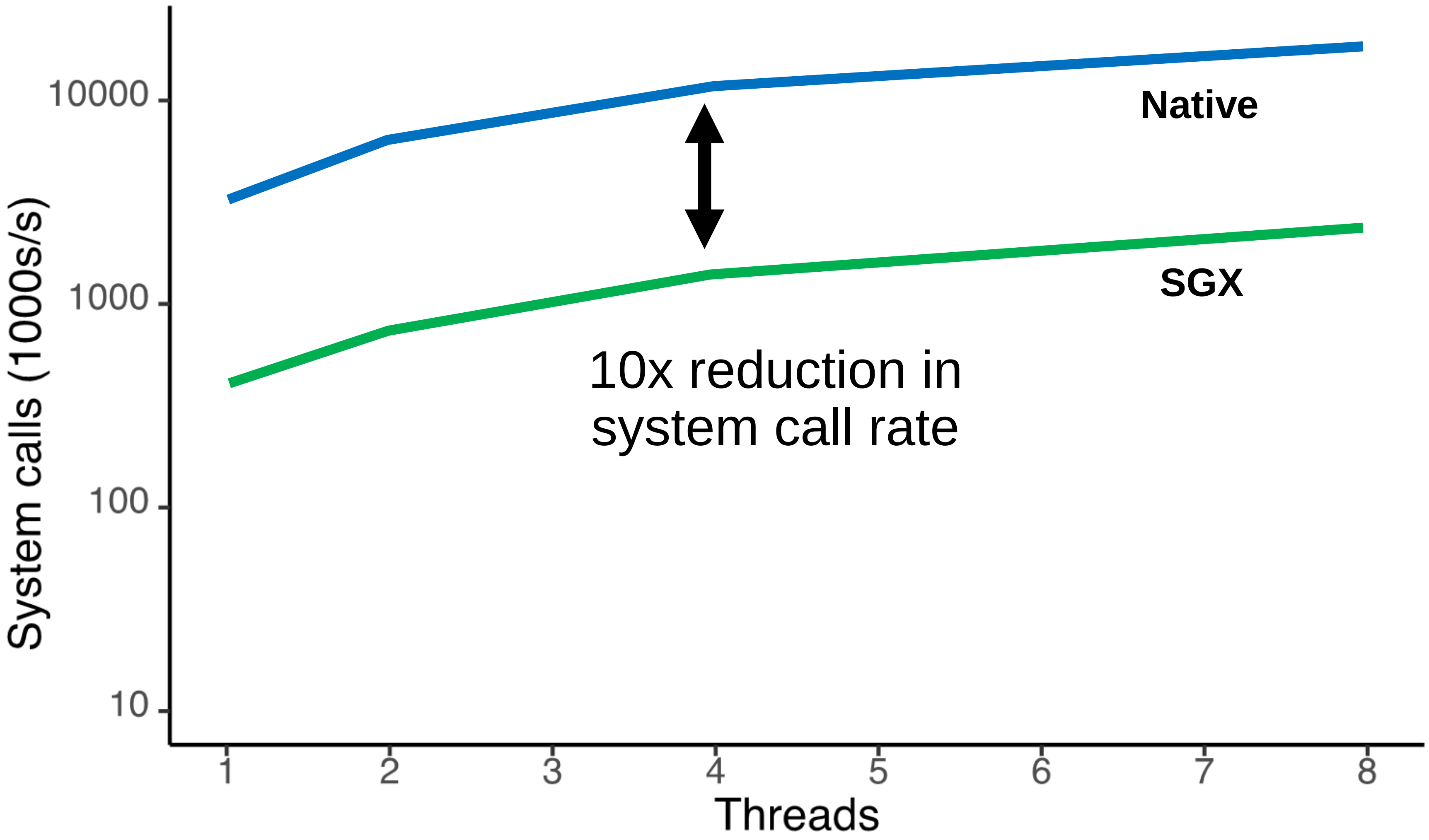
– CPU performs checks and transparently saves/restores state

Must exit enclave to perform system calls

– System calls invoke OS kernel, which is untrusted



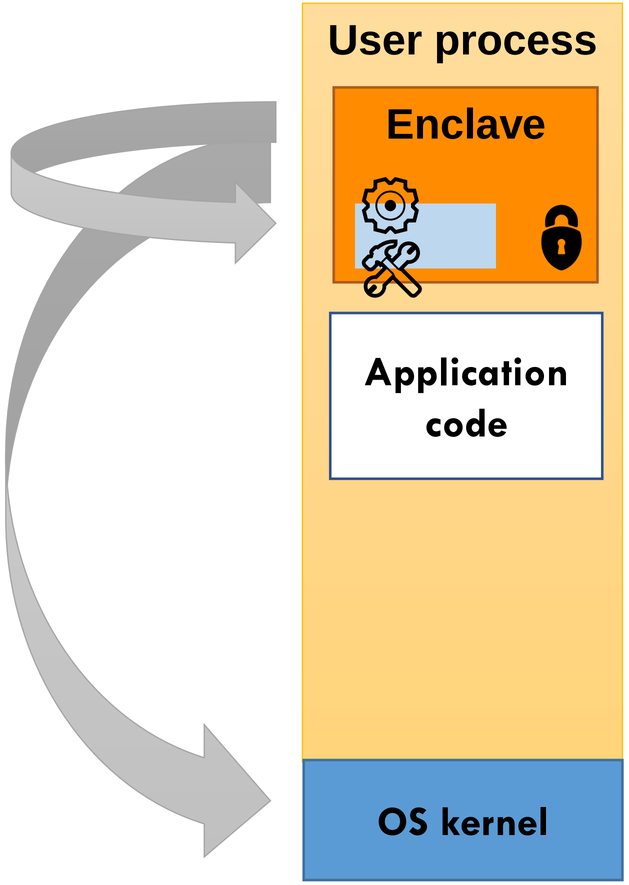
Enclaves: Performance Cost of System Calls



Idea: Reduce Number of Enclave Transitions

1. Provide **user-level threading** inside enclaves
   - Enclave threads can remain inside enclave
   - Thread scheduler switches between user-level threads

2. Provide **OS functionality** inside enclaves to avoid transitions for systems calls
   - Thread synchronisation
   - Memory management
   - File systems
   - Networking
   - Signal handling



SGX-LKL: System Support for Enclaves

SGX-LKL runs **unmodified Linux applications** in SGX enclaves

– Applications and dependencies provided via encrypted disk image

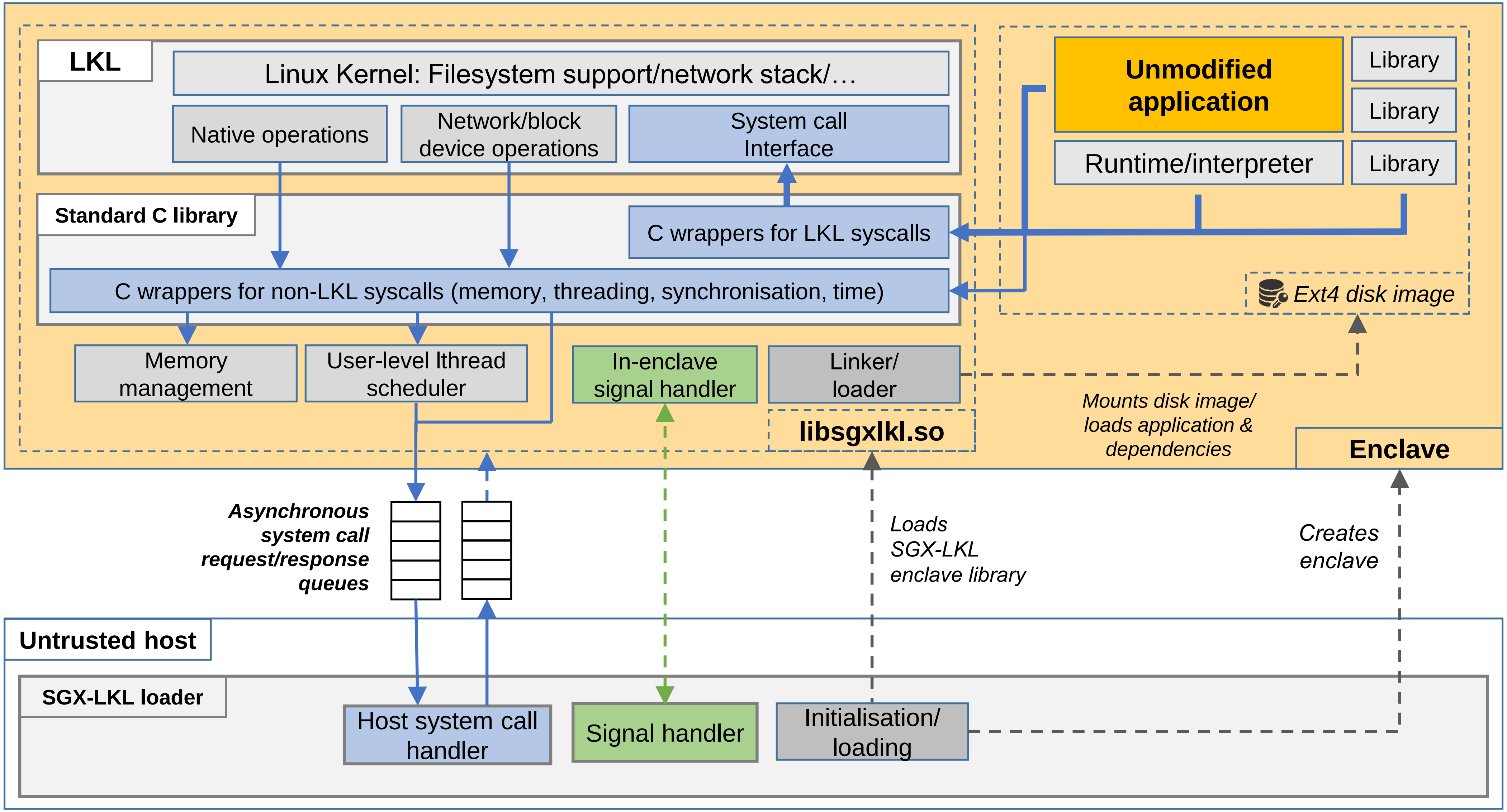
Linux kernel functionality available inside enclaves

– Based on **Linux Kernel Library (LKL):** Architecture-specific port of Linux kernel (**`github.com/lkl`**)

– Trusted file system and network stacks

1. **User-level threading**
   – In-enclave synchronisation primitives (**futex** implementation)
2. **Asynchronous system calls**
   – Similar to SCONE [OSDI'16]
3. **Custom memory allocator**
   – Integration between kernel and enclave memory allocator

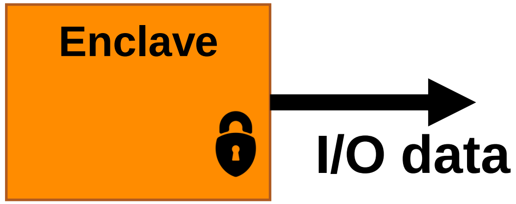
SGX-LKL Architecture



SGX-LKL: File System & Networking Support

SGX-LKL provides **trusted Linux file system** support
- Encryption/integrity protection performed at disk block level

Uses standard Linux device mapper API for disks
- dm-crypt for encryption
- dm-verity for integrity protection



SGX-LKL provides **trusted network stack**
- Enclave applications can use arbitrary network protocols (TCP, UDP, ...) securely

Uses TUN/TAP interface to send/receive packets via host OS kernel
- Performs layer-2/3 encryption inside enclave (e.g. IPSec)

SGX-LKL: Thin Interface to Host OS

Workload-independent host calls

`fcntl`	I/O
`ioctl`	I/O
`lseek`	I/O
`close`	I/O
`mmap`	Memory
`mremap`	Memory
`munmap`	Memory
`exit`	Process
`gettid`	Process
`pipe`	Process

Workload-dependent host calls

`read`	I/O
`readv`	I/O
`pread64`	I/O
`preadv`	I/O
`write`	I/O
`writev`	I/O
`pwrite64`	I/O
`pwritev`	I/O
`fdatasync`	I/O
`mprotect`	Memory
`msync`	Memory
`sigaction`	Signal handling
`sigpending`	Signal handling
`sigprocmask`	Signal handling
`sigsuspend`	Signal handling
`sigtimedwait`	Signal handling
`tkill`	Signal handling
`clock_gettime`	Time
`clock_getres`	Time
`gettid`	Process

Host interface is **side channel**

- Workload-dependent host calls may leak sensitive data

☛ **Ongoing work: Can we make the SGX-LKL host interface oblivious?**

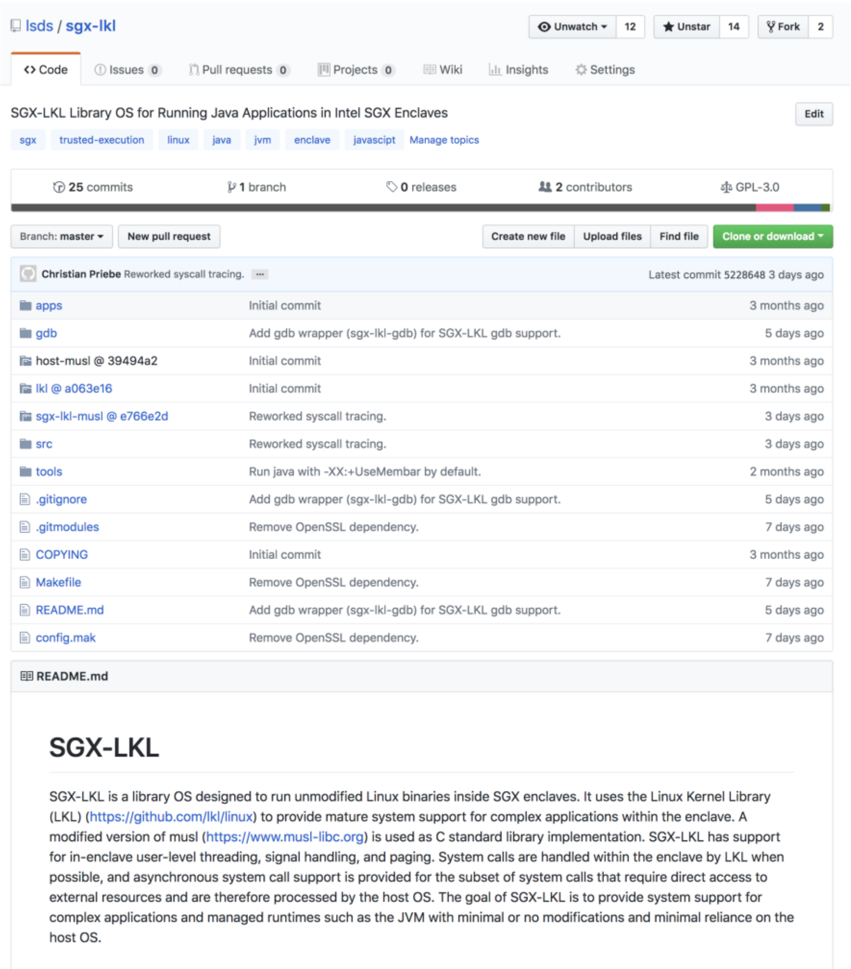
SGX-LKL: Supported Applications

Launches **Linux binaries** from **Alpine Linux** inside enclaves

- Nginx
- Redis
- Memcached
- Python, Perl
- …

Support for **managed language runtimes**

- Oracle Hotspot **JVM** (Java/Scala): OpenJDK
- V8 **JavaScript** Engine



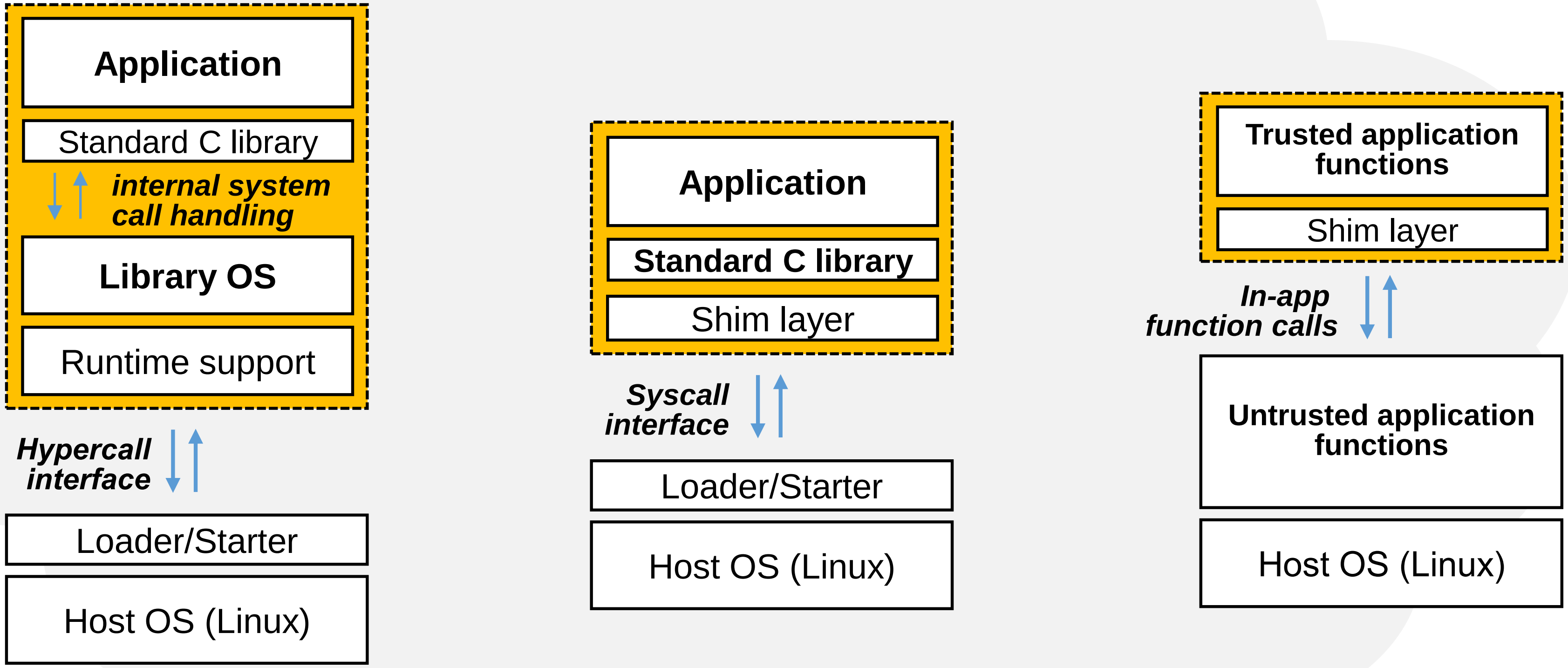
☛ **Try it on GitHub:**
www.github.com/lsds/sgx-lkl

Design Space: Systems Support for SGX Enclaves

(a) **Library OS:**
SGX-LKL, Haven

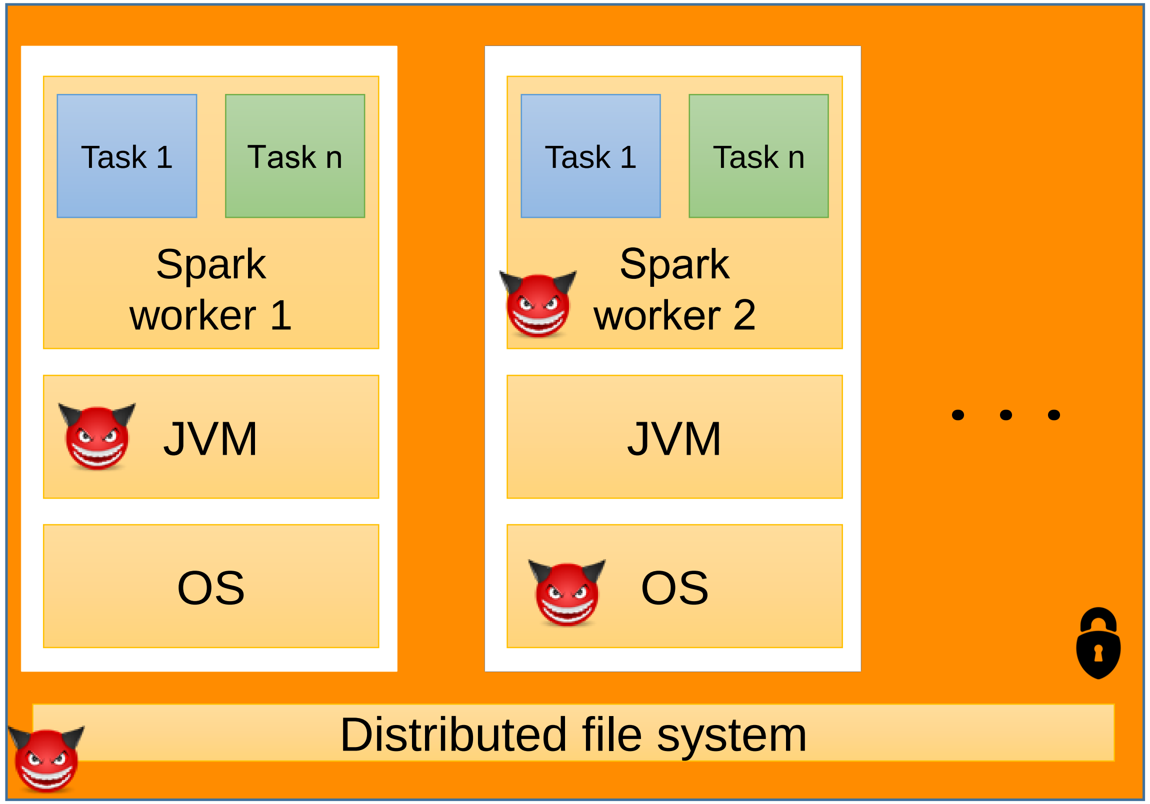
(b) **System call interface:**
SCONE **[OSDI'16]**

(c) **Partitioned application:**
Glamdring **[USENIX ATC'17]**, Intel SGX SDK



How to Protect Large Cloud Applications?

Consider deploying **Apache Spark** inside an SGX enclave



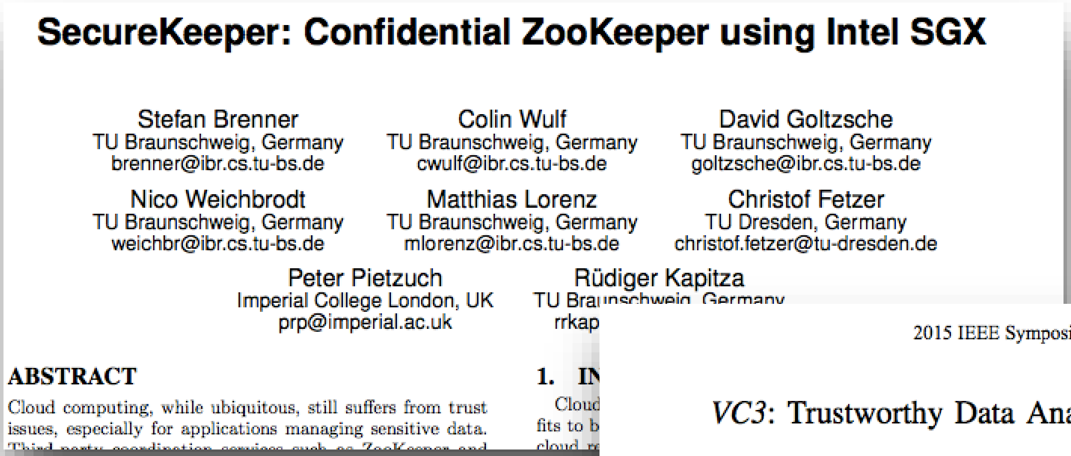

```
def main(args: Array[String]) {
  new SparkContext(new SparkConf())
    .textFile(args(0))
    .flatMap(line => {line.split(" ")})
    .map(word => {(word, 1)})
    .reduceByKey{case (x, y) => x + y}
    .saveAsTextFile(args(1))
}
```

Attackers can exploit vulnerabilities inside enclave code

☛ **Only data and processing code is sensitive**

Partition Cloud Applications to Minimise TCB

Many examples of **manual partitioning** of applications by developers



SecureKeeper: Confidential ZooKeeper using Intel SGX

Stefan Brenner
TU Braunschweig, Germany
brenner@ibr.cs.tu-bs.de

Colin Wulf
TU Braunschweig, Germany
cwulf@ibr.cs.tu-bs.de

David Goltzsche
TU Braunschweig, Germany
goltzsche@ibr.cs.tu-bs.de

Nico Weichbrodt
TU Braunschweig, Germany
weichbr@ibr.cs.tu-bs.de

Matthias Lorenz
TU Braunschweig, Germany
mlorenz@ibr.cs.tu-bs.de

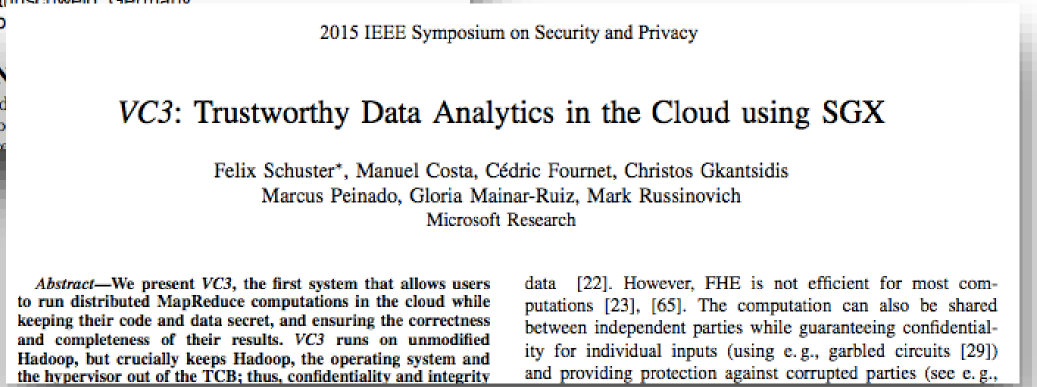
Christof Fetzer
TU Dresden, Germany
christof.fetzer@tu-dresden.de

Peter Pietzuch
Imperial College London, UK
prp@imperial.ac.uk

Rüdiger Kapitza
TU Braunschweig, Germany

ABSTRACT

Cloud computing, while ubiquitous, still suffers from trust issues, especially for applications managing sensitive data.



2015 IEEE Symposium on Security and Privacy

***VC3*: Trustworthy Data Analytics in the Cloud using SGX**

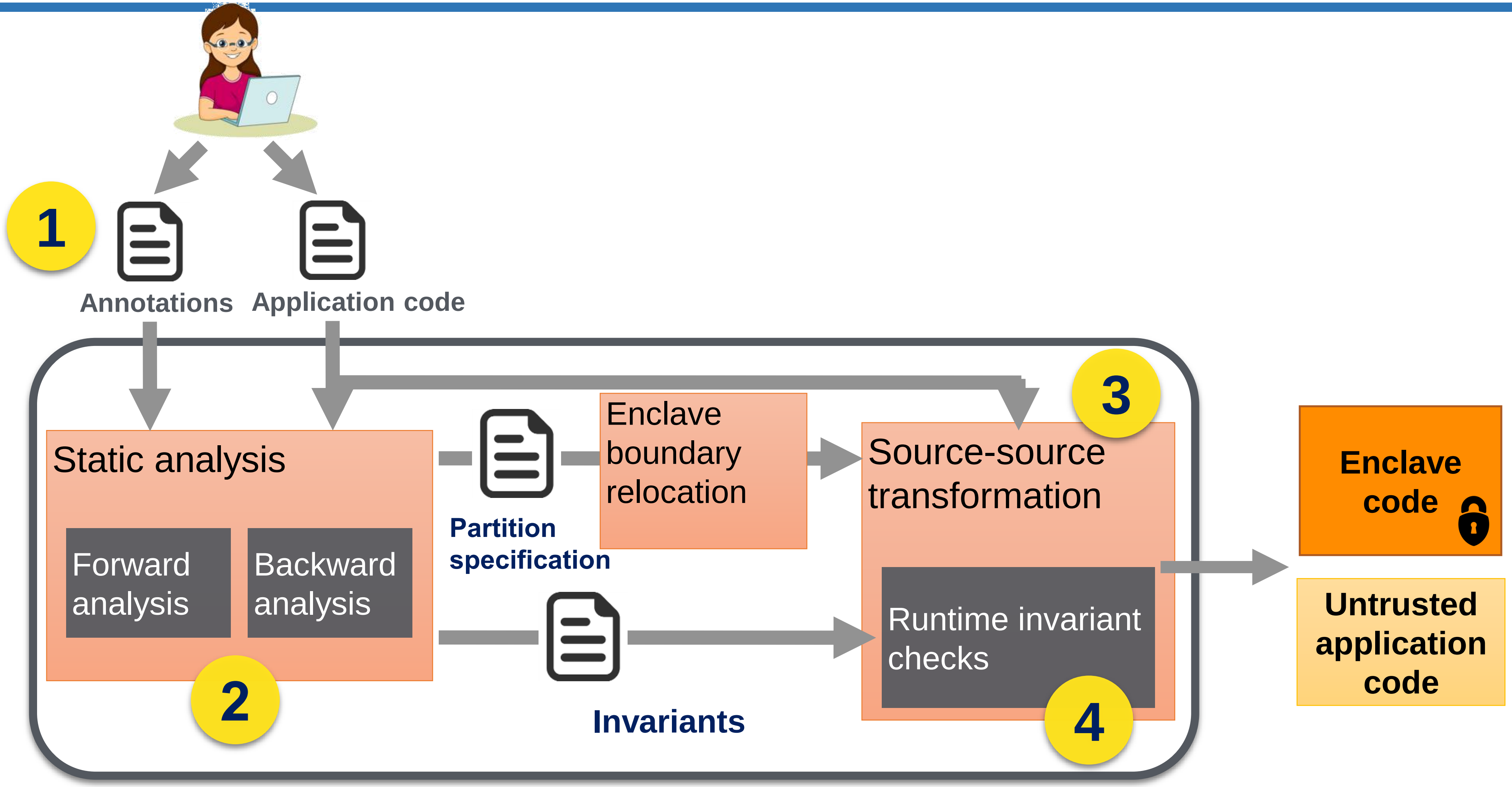
Felix Schuster*, Manuel Costa, Cédric Fournet, Christos Gkantsidis
Marcus Peinado, Gloria Mainar-Ruiz, Mark Russinovich
Microsoft Research

***Abstract*—We present *VC3*, the first system that allows users to run distributed MapReduce computations in the cloud while keeping their code and data secret, and ensuring the correctness and completeness of their results. *VC3* runs on unmodified Hadoop, but crucially keeps Hadoop, the operating system and the hypervisor out of the TCB; thus, confidentiality and integrity**

data [22]. However, FHE is not efficient for most computations [23], [65]. The computation can also be shared between independent parties while guaranteeing confidentiality for individual inputs (using e. g., garbled circuits [29]) and providing protection against corrupted parties (see e. g.,

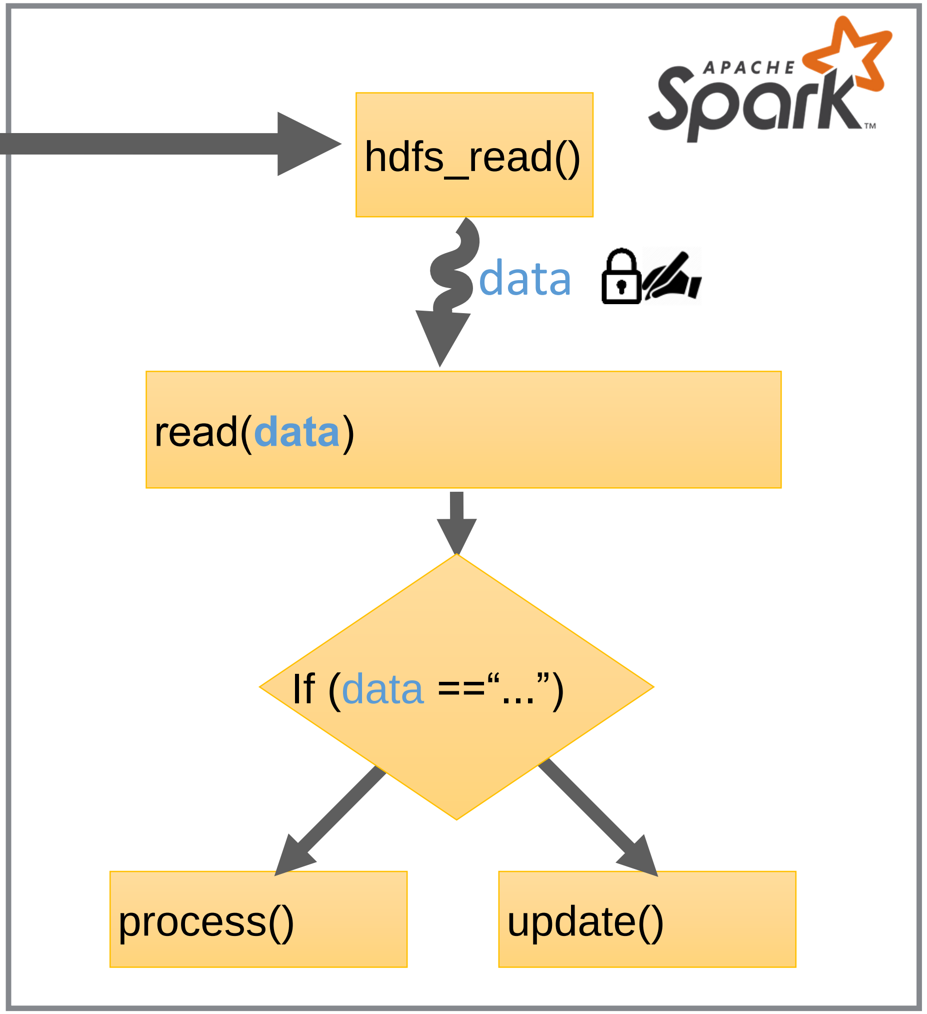
☛ **Can we automatically determine the minimum functionality to run inside an enclave?**

Glamdring: SGX Partitioning Framework [USENIX ATC'17]



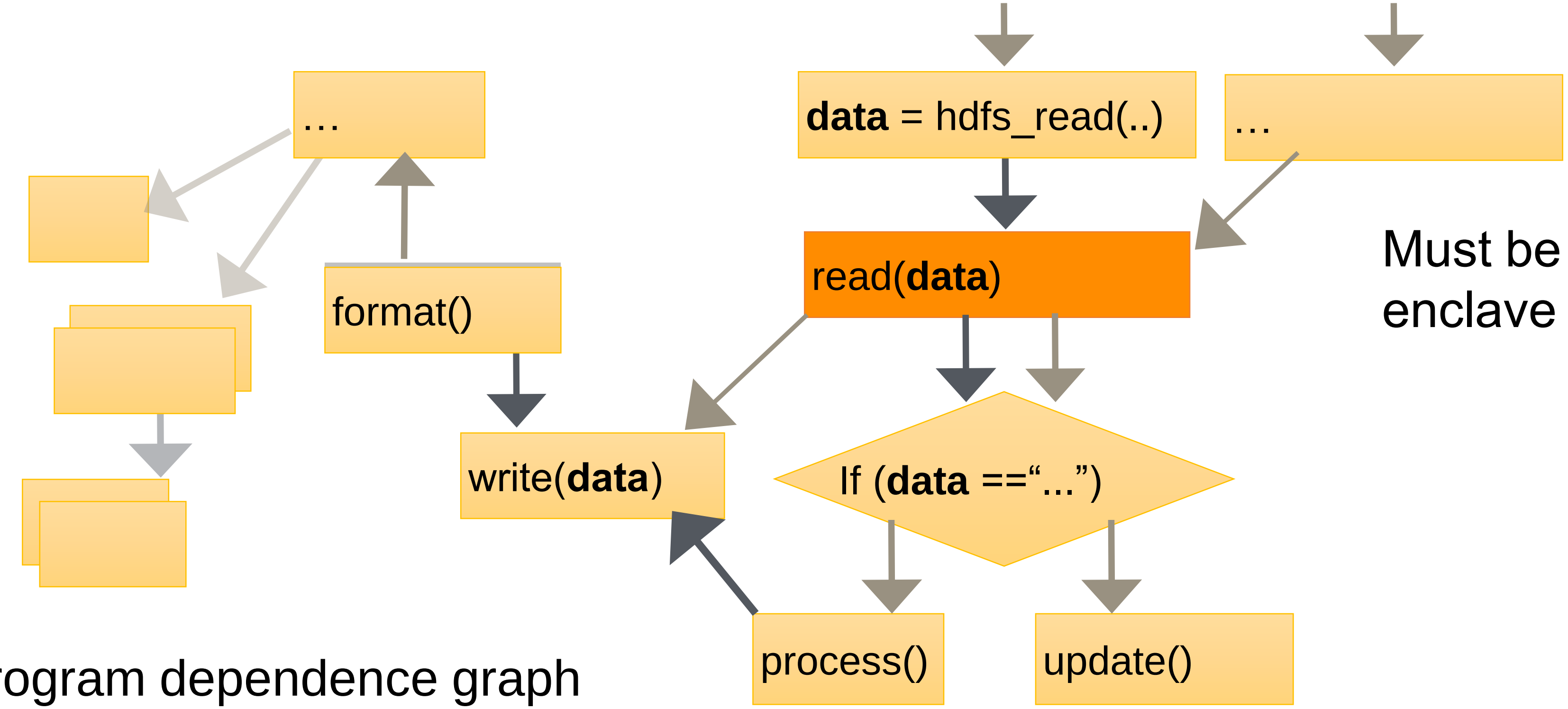
Compiler-based framework for partitioning C applications

1. Developers Annotate Security-Sensitive Data




```
#pragma glamdring sensitive source(data)
void read(char* data) {
 …
 }
```

2. Static Analysis to Identify Sensitive Code



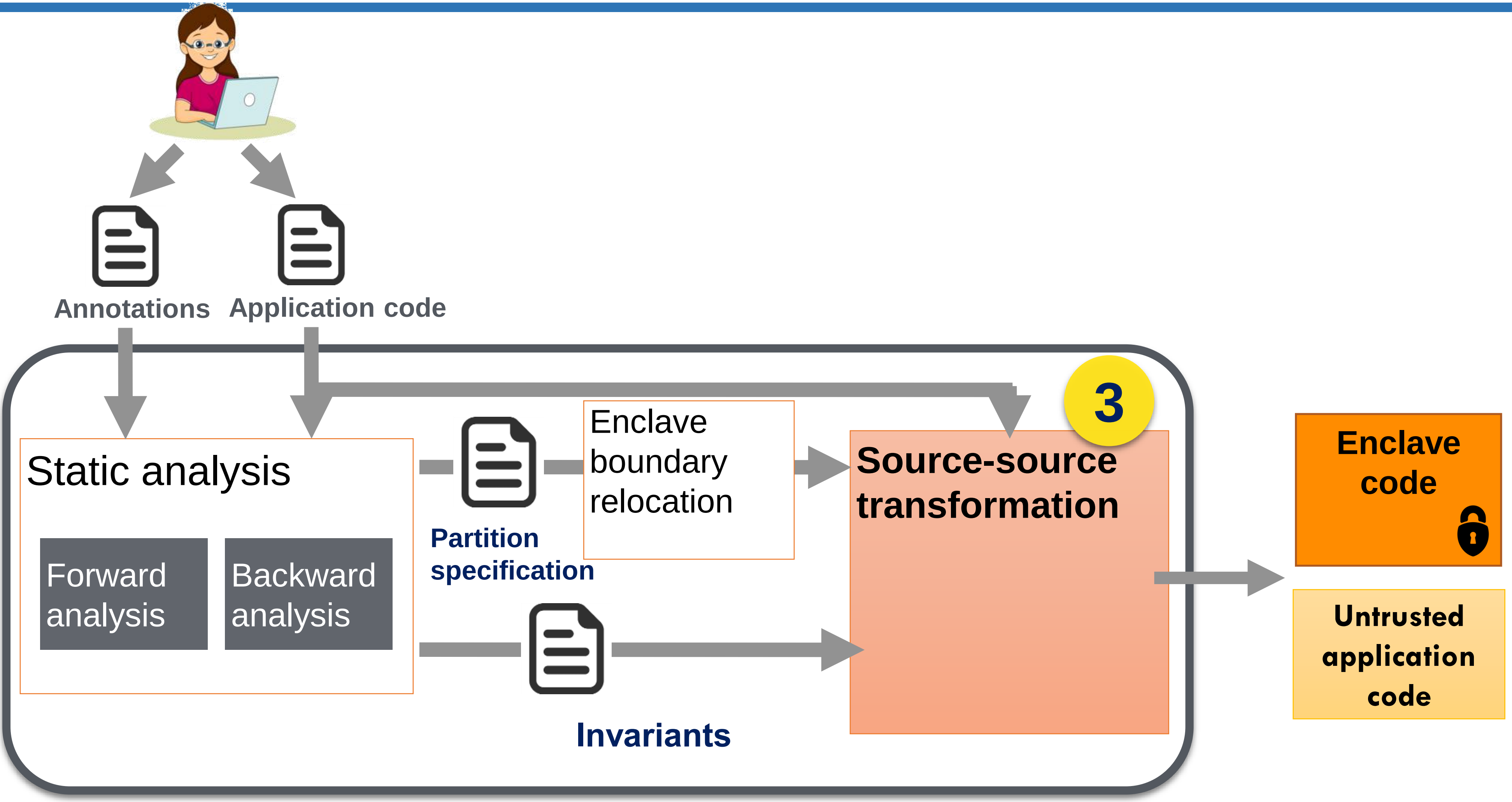
Must be protected by enclave

Program dependence graph

To ensure **data confidentiality:** **forward dataflow analysis**

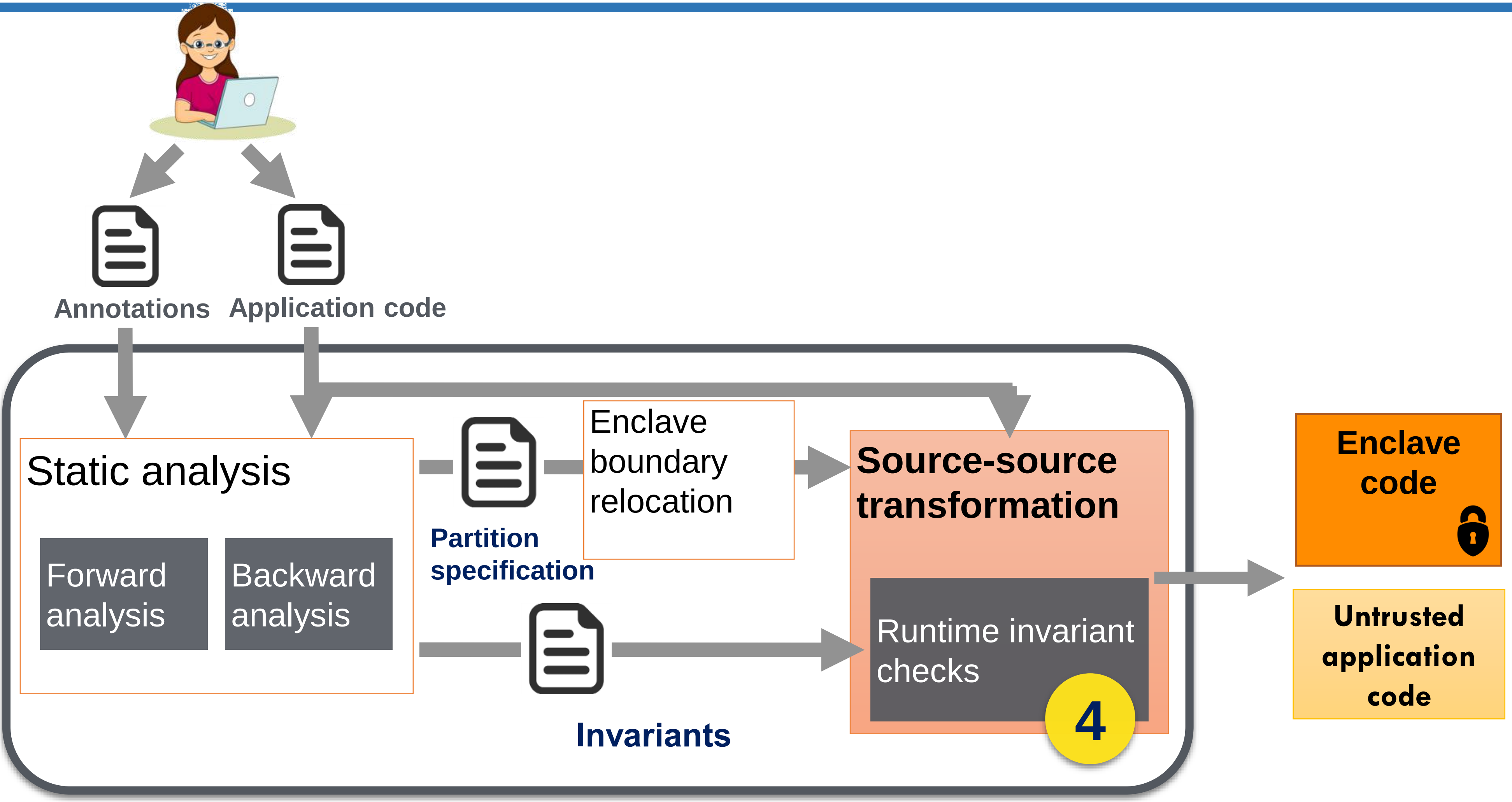
To ensure **data integrity:** **backward dataflow analysis**

3. Producing Partitioned SGX Application



Source-to-source compiler based on LLVM

4. Upholding Static Analysis Invariants



Add runtime checks that enforce invariants on program state used by static analysis

Evaluation: How Much Code Inside Enclave?

Application	Total code size (LOCs)	Enclave size (LOCs)
Memcached	**31,000**	**12,000 (40%)**
DigitalBitbox	**23,000**	**8,000 (38%)**
LibreSSL	**176,000**	**38,000 (22%)**

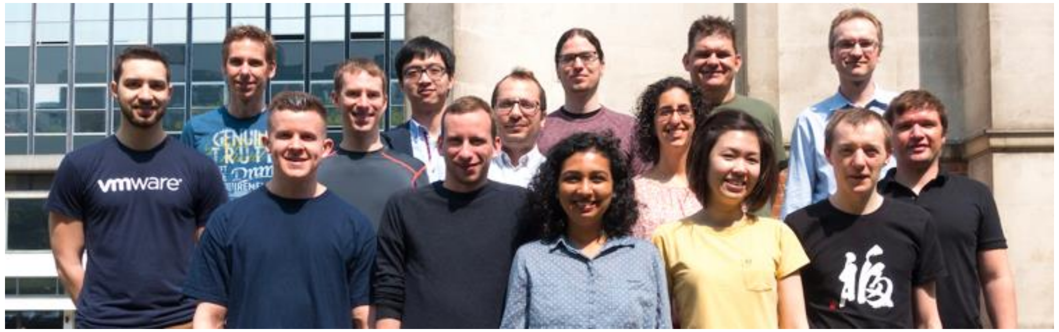
☛ **Enclave contains less than 40% of application code**

Summary: Securing Cloud Apps using Intel SGX

Trusted execution promises to enhance security for cloud tenants
- But requires new systems support and developer tools

Tenants want to run **unmodified existing applications** with SGX
- **SGX-LKL** provides user-level threading, file system and networking support

Developers require **automated tooling** when using enclaves
- **Glamdring** semi-automatically partitions applications for Intel SGX



Thank You — Any Questions?

Peter Pietzuch
https://lsds.doc.ic.ac.uk — prp@imperial.ac.uk

Backup Slides

What Can Cloud Tenants Do Today?

Use **encrypted communication channels** (TLS)?

- Protects data in transit but not once in cloud environment

Encrypt data before sending to cloud environment?

- Only works for some cloud services (e.g. storage)
- Limits functionality of cloud services

Use **homomorphic encryption**?

- Large performance overhead and limited applicability

What about **integrity**?

- Challenging to ensure that computation was executed faithfully

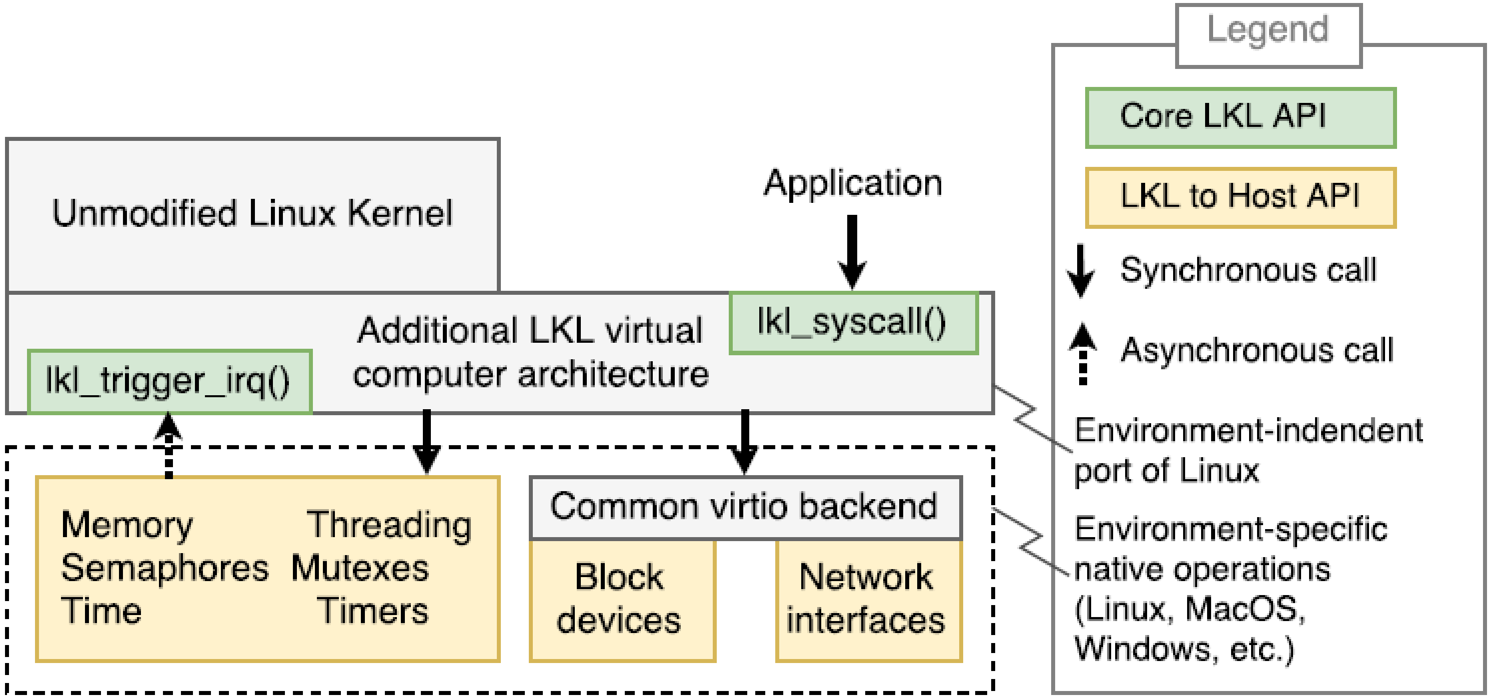
Linux Kernel Library (LKL)

Architecture-specific port of mainline Linux (`github.com/lkl`)

- Good maintainability
- Mature implementation

LKL Architecture

- Follows Linux no MMU architecture
- Full filesystem support
- Full network stack available



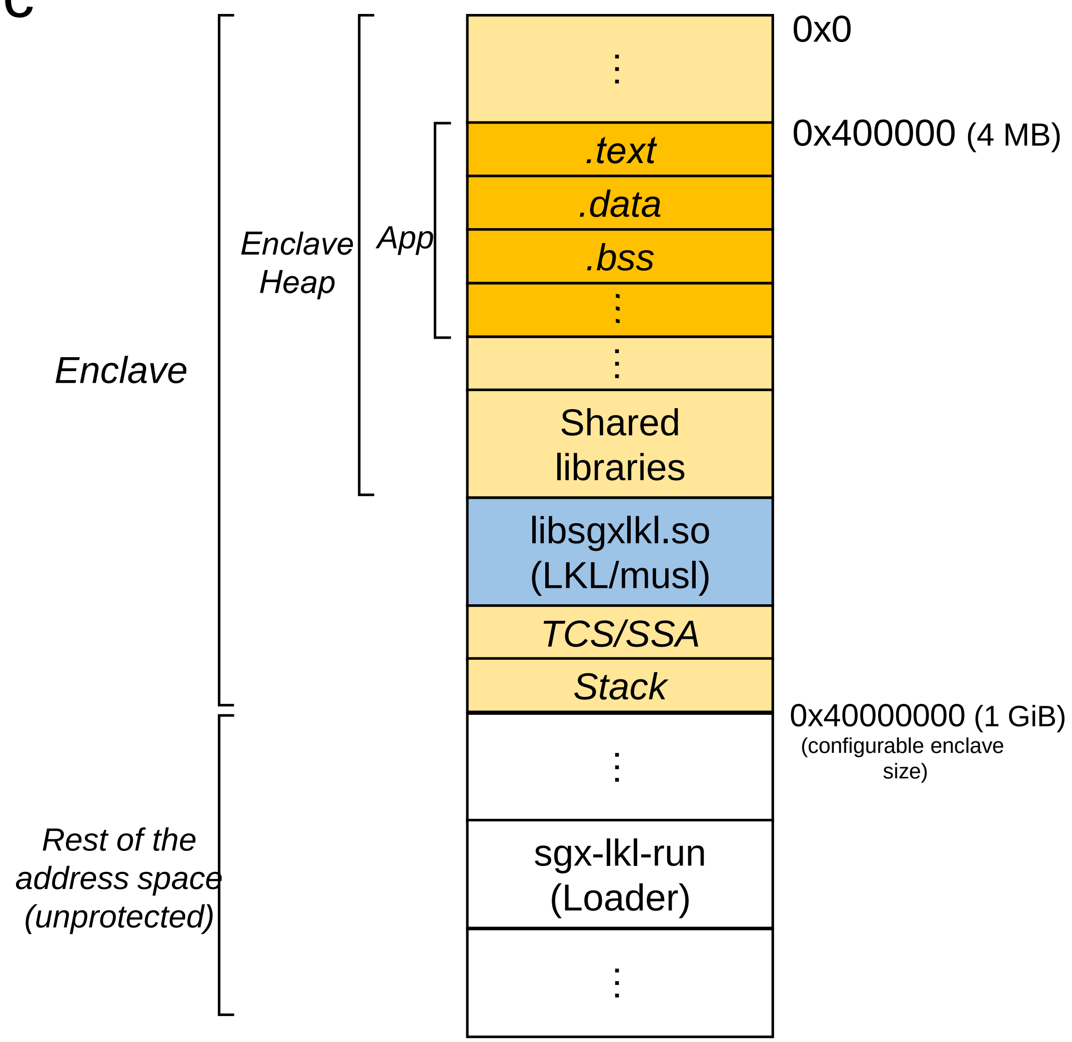
Memory Layout in SGX-LKL

Need correct initialisation of LKL & libc

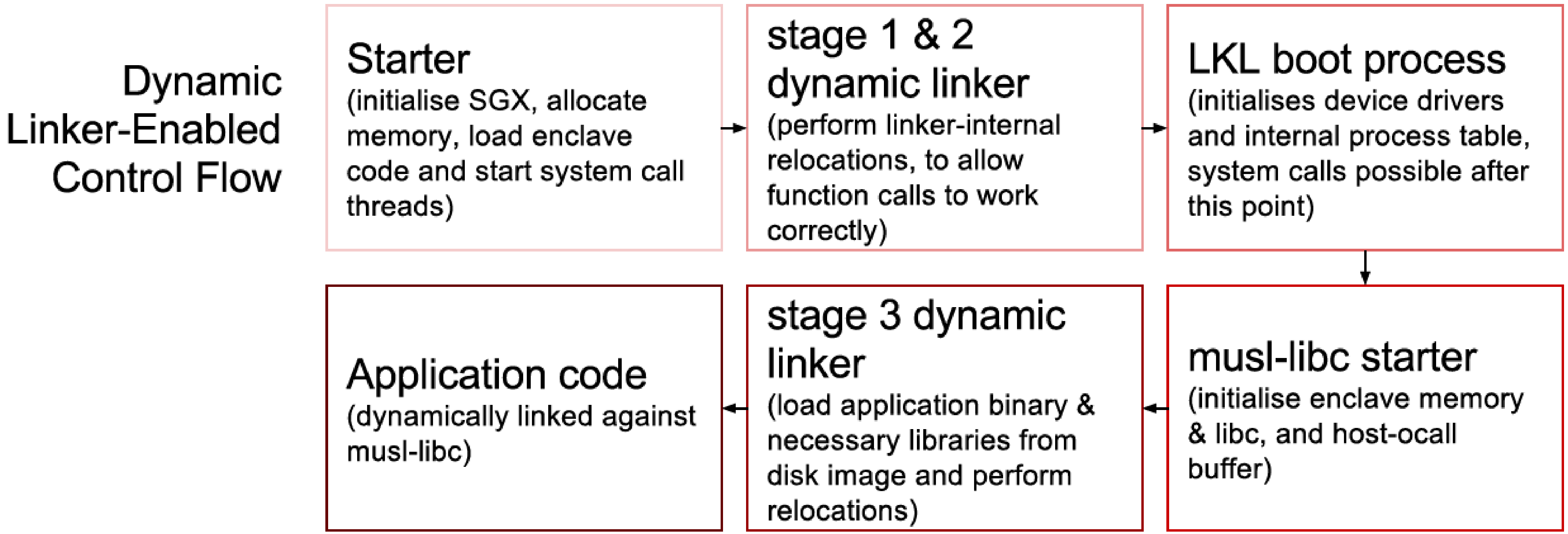
- Relocation/linking/loading

Support for position dependent code

- Leaves gap for application load address (0x400000+)
- Map enclave from 0x0 page due to SGX restrictions



Support for Dynamic Linking



Support for Disk Encryption

Initial enclave code and data measured by CPU

But must ensure confidentiality/integrity of disk image

– Loaded binary and dependent libraries must be trustworthy

Idea: **Support encryption/integrity protection at block level**

Uses standard Linux **device mapper API**

– **dm-crypt** for encryption
– **dm-verity** for integrity protection
    – Merkle tree for disk block verification
    – Leaf nodes contain hashes of disk blocks

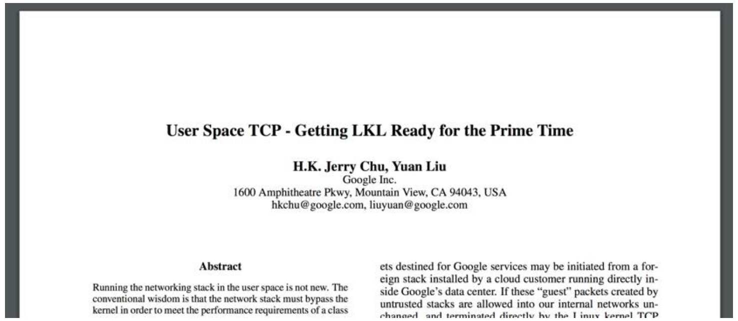
Support for Linux Networking

Use in-enclave **trusted Linux network stack**

TUN/TAP interface to send/receive packets via host kernel

- Layer-2/3 encryption possible in-enclave (e.g. **IPSec, VPNs**)
- Support arbitrary network protocols with encryption

Used by Google in production for app-level proxies:



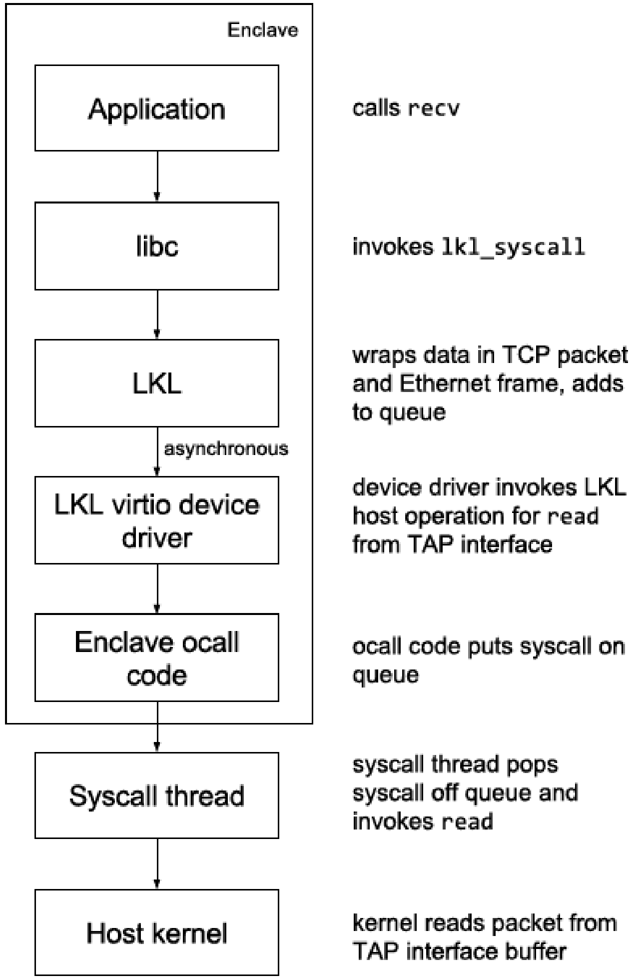
User Space TCP - Getting LKL Ready for the Prime Time

H.K. Jerry Chu, Yuan Liu
Google Inc.
1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA
hkchu@google.com, liuyuan@google.com

Abstract

Running the networking stack in the user space is not new. The conventional wisdom is that the network stack must bypass the kernel in order to meet the performance requirements of a class

ets destined for Google services may be initiated from a foreign stack installed by a cloud customer running directly inside Google's data center. If these "guest" packets created by untrusted stacks are allowed into our internal networks unchanged, and terminated directly by the Linux kernel TCP



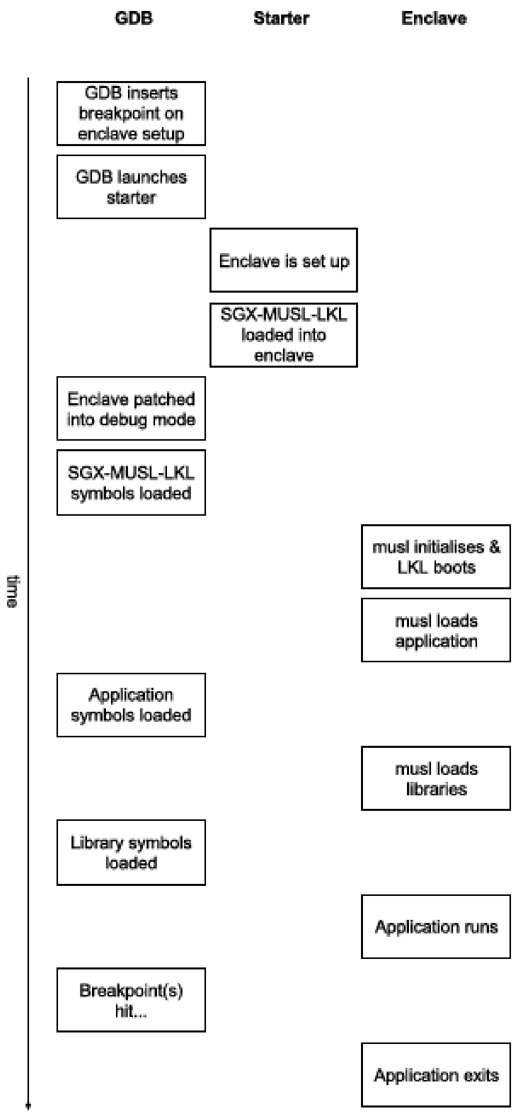
SGX-LKL: Debugging Support

GDB plugin

- Breakpoints, watchpoints, stack traces
- Dynamically loads required symbols
- Supports software simulation and hardware SGX mode

Perf support

- Passes required enclave symbols to perf



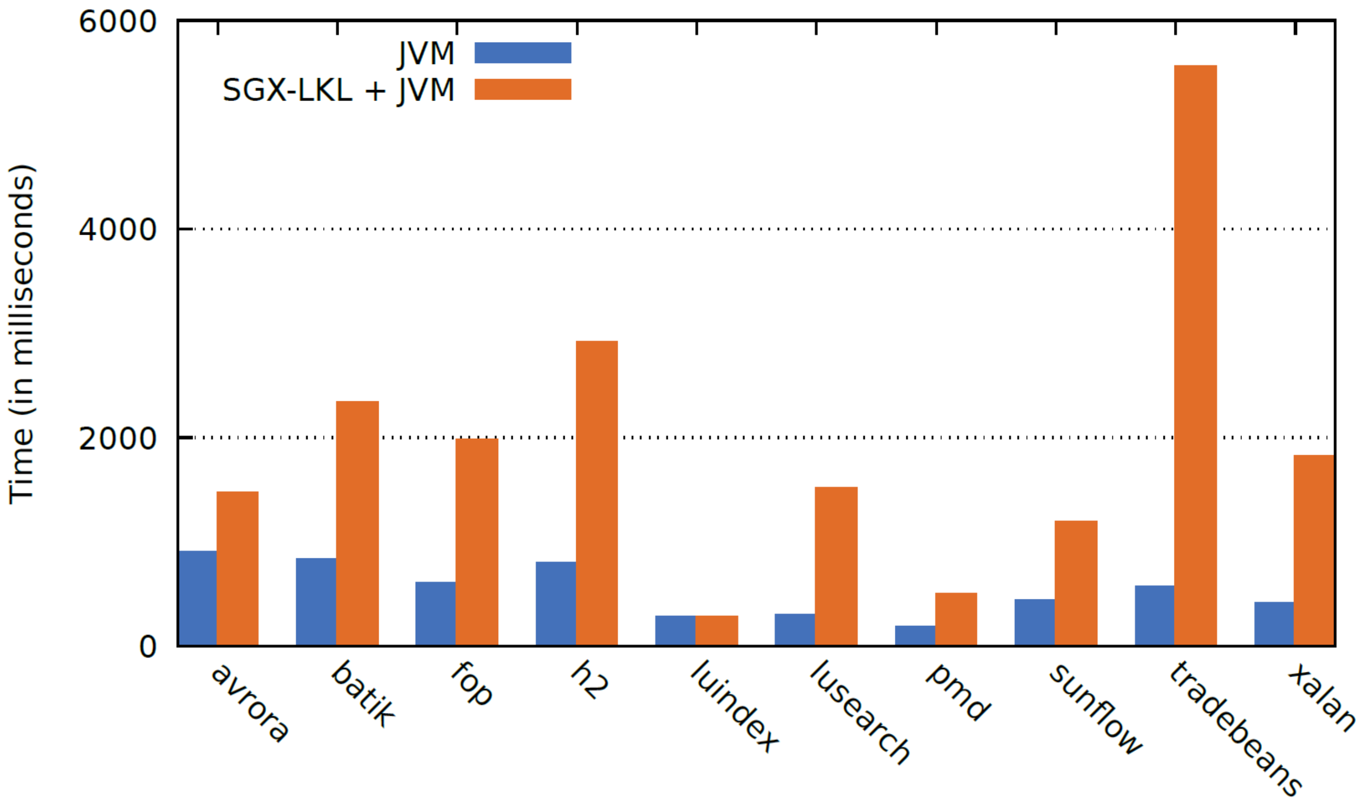
Comparison: SGX-LKL vs Graphene-SGX

	SGX-LKL	Graphene-SGX
Library OS implementation	Linux Kernel Library (LKL) (`github.com/lkl`) Arch-specific fork of Linux kernel	Custom implementation
Process model	Single process	Multi-process (fork(), IPC support)
Standard C library support	musl libc (`www.musl-libc.org`)	glibc
Support for unmodified binaries	✓ (from Alpine Linux)	✓
Application packaging	Encrypted block device image	Encrypted files on host FS + manifest file
File I/O support	Complete Linux VFS impl. in enclave Support for arbitrary Linux FSs (Ext4, btrfs, xfs)	Relies on host OS FS impl. Support for host FS only
Networking I/O support	Complete Linux network stack in enclave Support for arbitrary network protocols Layer 2/3 encryption	Relies on host OS network stack UNIX socket support only Layer 7/4 encryption
Threading model	User-level (N:M) threading	Kernel-level (1:1) threading
Synchronisation support	Enclave futex implementation	Relies on host OS futex impl.
System call support	Asynchronous system call invocations (No enclave transitions)	Synchronous system call invocations (Requires enclave transitions)
Enclave shielding	Relies on Linux kernel impl. for shielding (e.g. block device encryption, IPSec etc)	Custom shield implementation
Support for enclave signal handling	✓ (partial)	✓
Other enclave system support	Anything provided by Linux kernel (!)	✗

SGX-LKL Performance: Java

DaCapo benchmark results for JVM with SGX-LKL vs. non-SGX execution

– Intel Xeon E3-1280 v5 at 3.70 GHz with 64 GB RAM

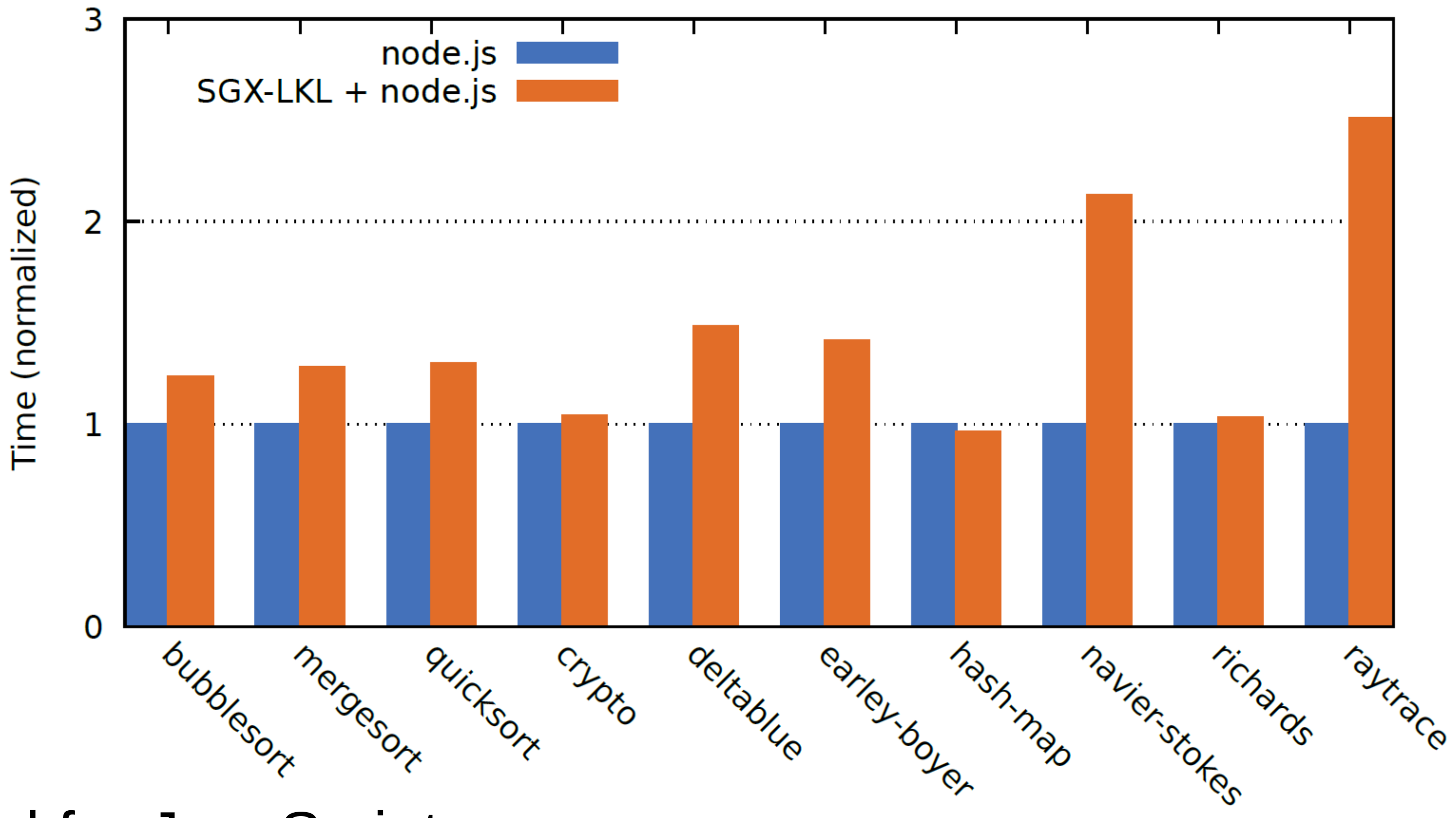


Performance overhead for large enclaves due to SGX memory paging

Few enclave transitions due to asynchronous system call interface

SGX-LKL Performance: JavaScript

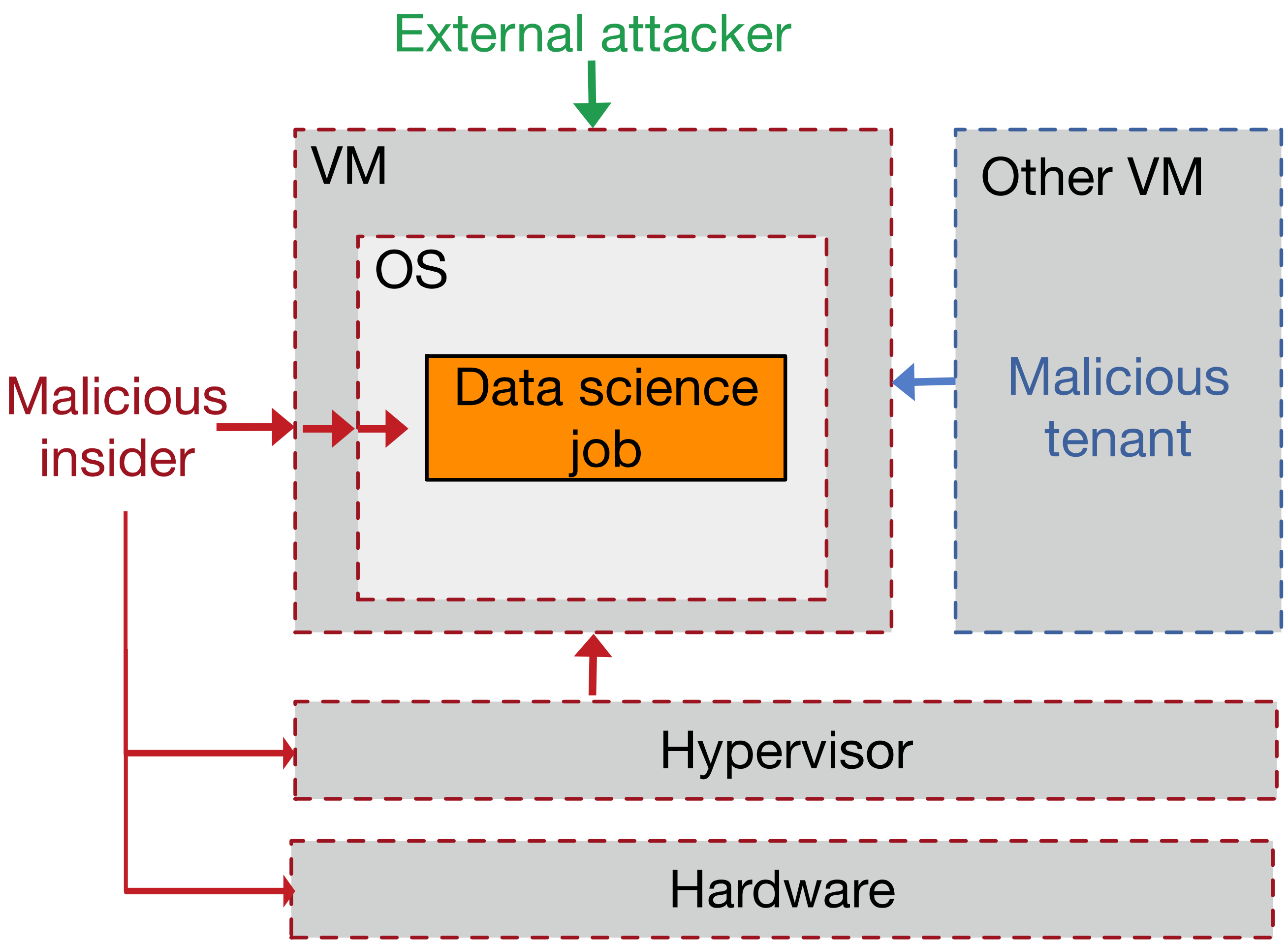
Octane benchmark results for node.js with SGX-LKL vs. non-SGX execution



Competitive overhead for JavaScript

Workloads mostly compute-heavy

Security Threats in Data Science



Secure Machine Learning

Secure **machine learning (ML)** killer application

- Resource-intensive thus good use case for cloud usage
- Raw training data comes with security implications

Complex implementations of ML algorithms cannot be adapted for SGX

- Consider Spark MLlib with 100s of algorithms

Challenges

- Extremely **data-intensive** domain
- Must support **existing frameworks** (Spark, TensorFlow, MXNet, CNTK, …)
- ML requires **accelerators** support (GPUs, TPUs, …)
- Prevention of **side-channel** attacks

State of the Art

Protect confidentiality and integrity of tasks and input/output data

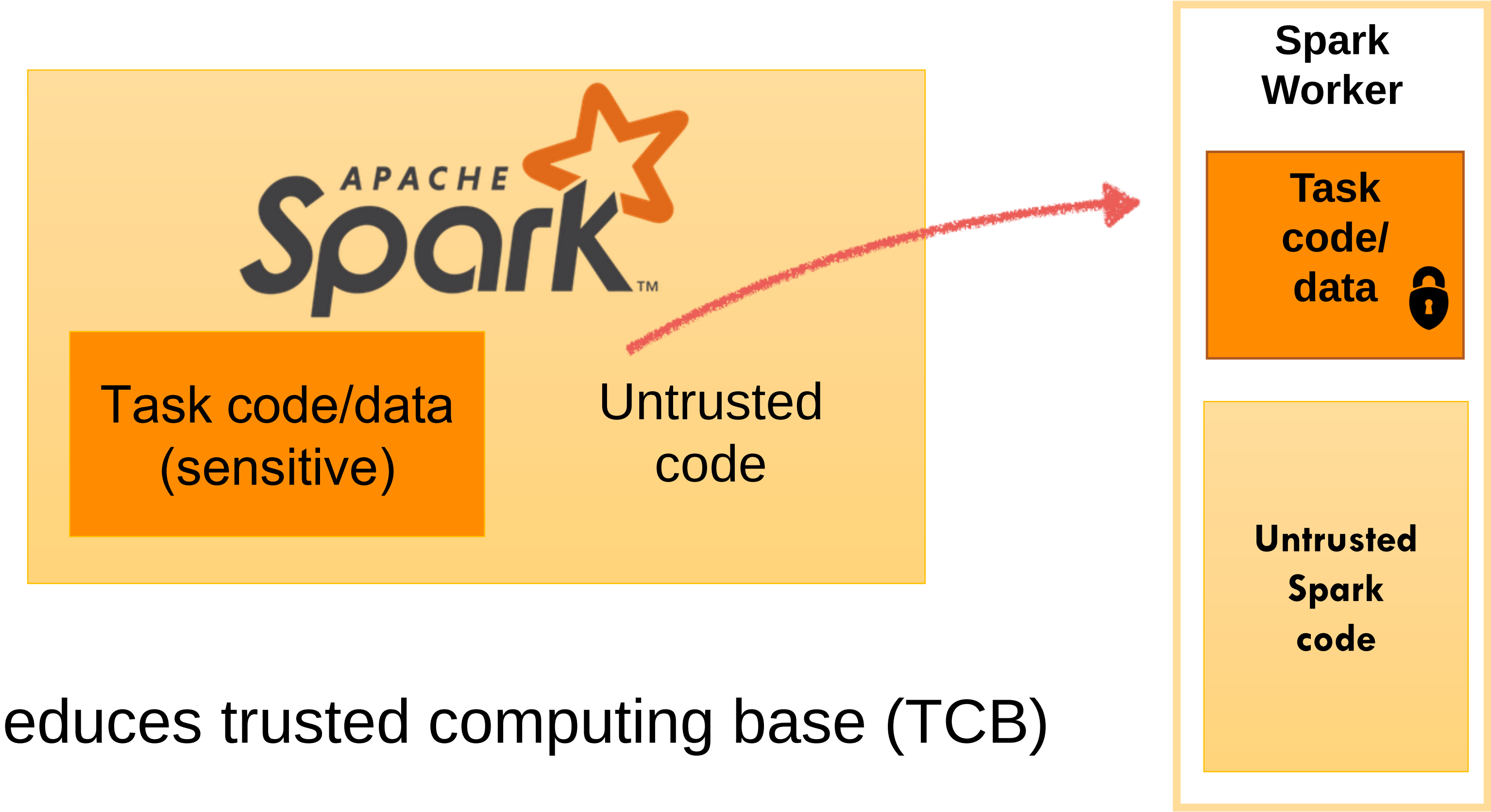
VC3 [Schuster, S&P 2015]

- Protects MapReduce Hadoop jobs
- Confidentiality/integrity of code/data; correctness/completeness of results
- No support for existing jobs → Re-implement for VC3

Opaque [Zheng, NSDI 2017]

- Hide access patterns of distributed data analytics (Spark SQL)
- Introduces new oblivious relational operators
- Does not support arbitrary/existing Scala Spark jobs

Minimising Enclave Code for Spark



Reduces trusted computing base (TCB)

☛ **How should developers identify the sensitive code?**

SGX-Spark Design

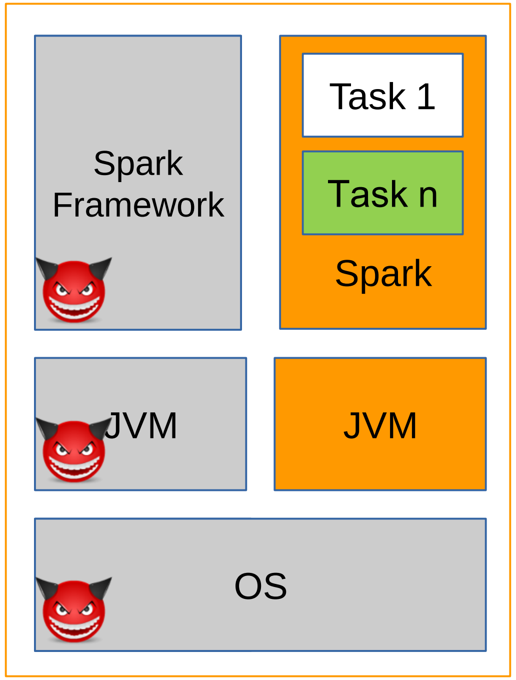
Protects data processing from cloud provider

Ensures confidentiality & integrity of existing big data jobs

No modifications for end users

- Different from Microsoft's VC3

Low performance overhead

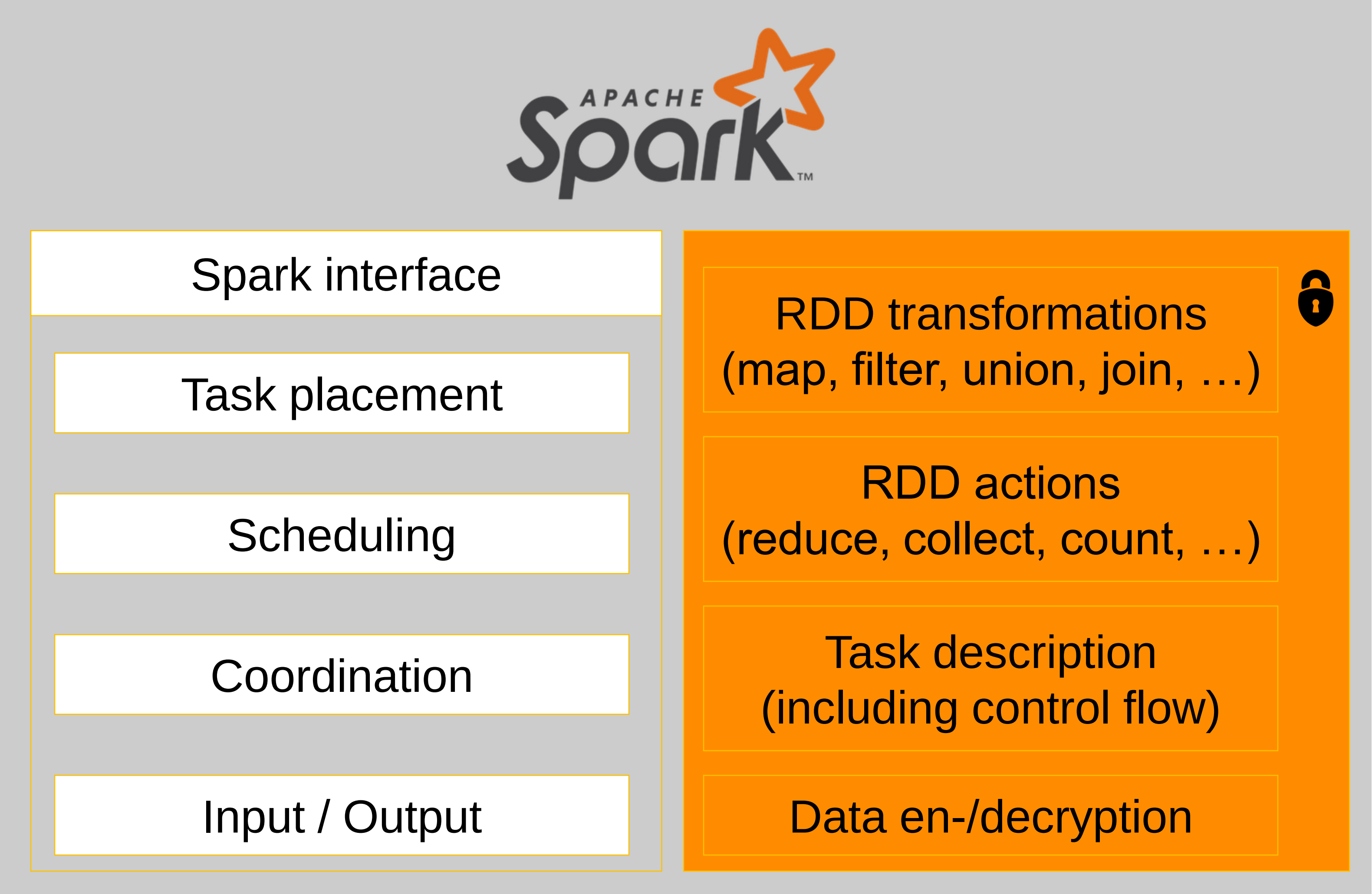


Code outside of enclave only accesses encrypted data

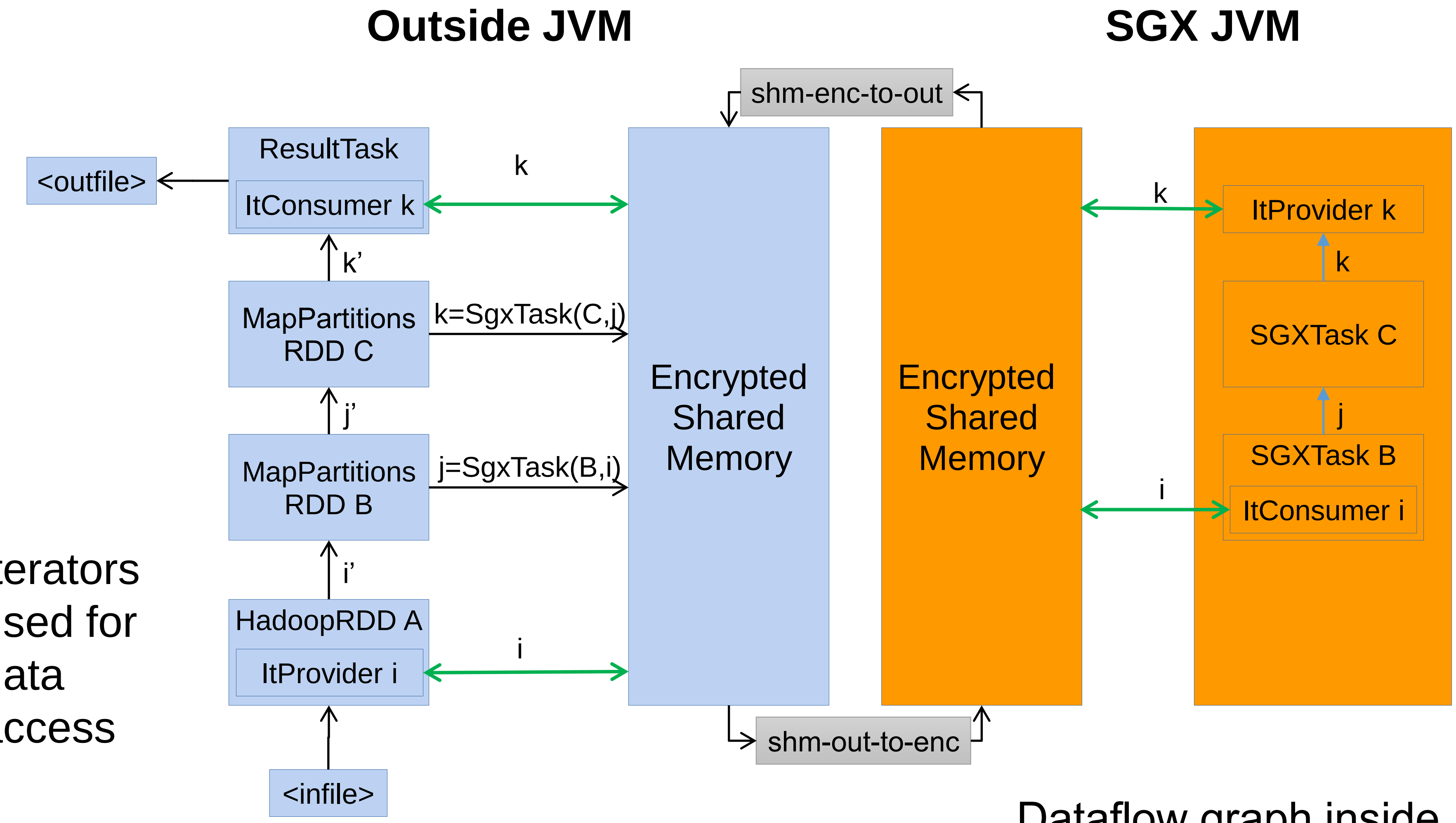
Only **SparkExecutor** inside SGX enclave

Requires two collaborating JVMs

SGX-Spark Architecture



Spark Executor for SGX



Iterators used for data access

Dataflow graph inside SGX enclave

Example: Smart Grid Data Processing

Pre-Processing MySQL

- Measurements every 15 min
- MySQL prefilters invalid data
- Generation of 3 CSV files:
  - Faults
  - Customer
  - DSM (Service and Maintenance Department)

Scala/Spark

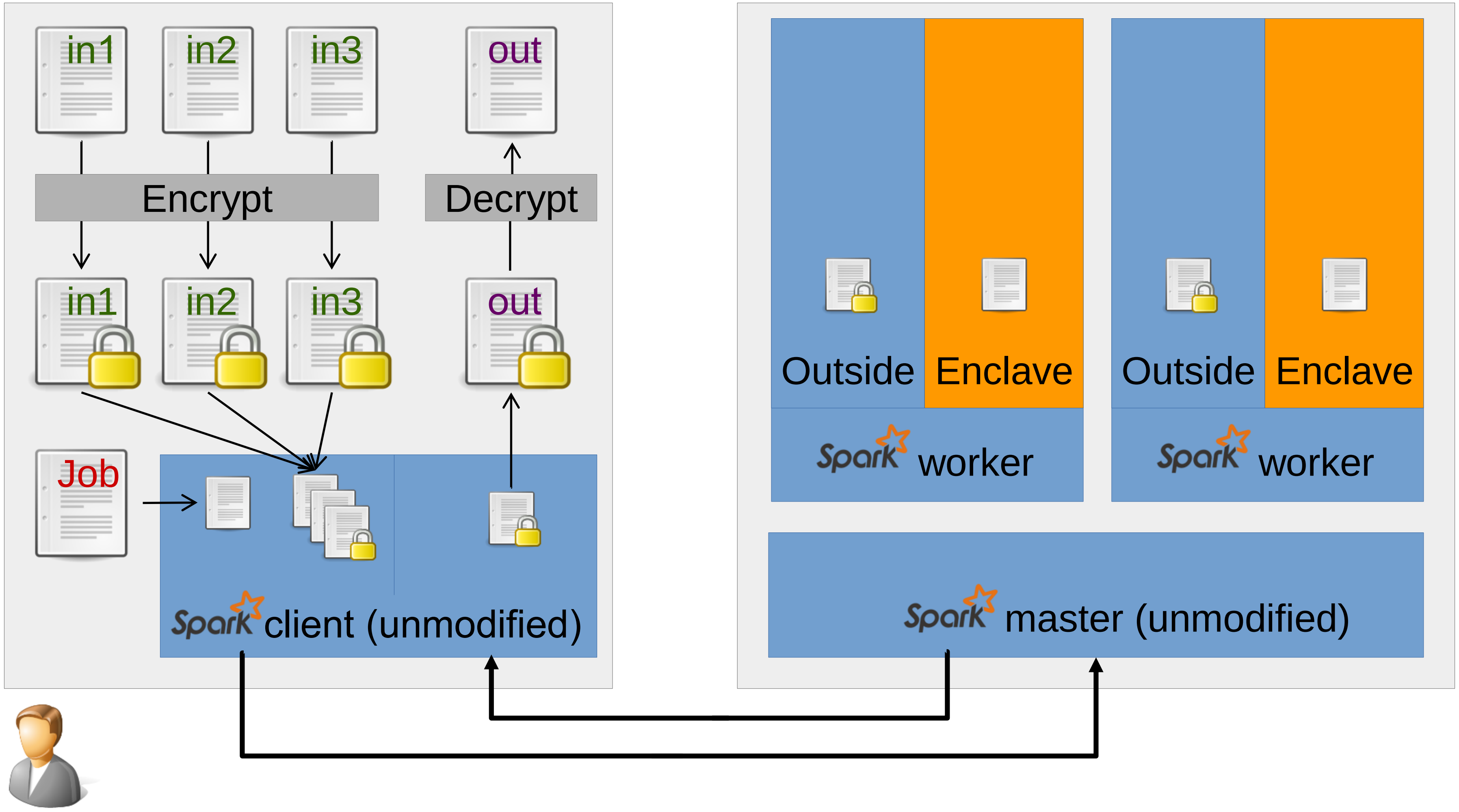
- Reads 3 CSV files
- Filters invalid locations
- Joins Customers with DSMs
- Joins (Customer, DSM) with Faults
- Computes total number of faults and total fault duration for each customer

Output

- Number of faults per consumer with respective service interruption times

Customer _id	Customer Name	Latitude	Longitude	Contract number	Medidor Serial	Service area abrreviation	Service area name	Number of faults	Total fault duration in seconds
5467	USICAP	-23.27197	-51.05277	39633896	31606197	LNA	Londrina	2	1800

Example: Smart Grid Data Processing



Summary: SGX-LKL and SGX-Spark

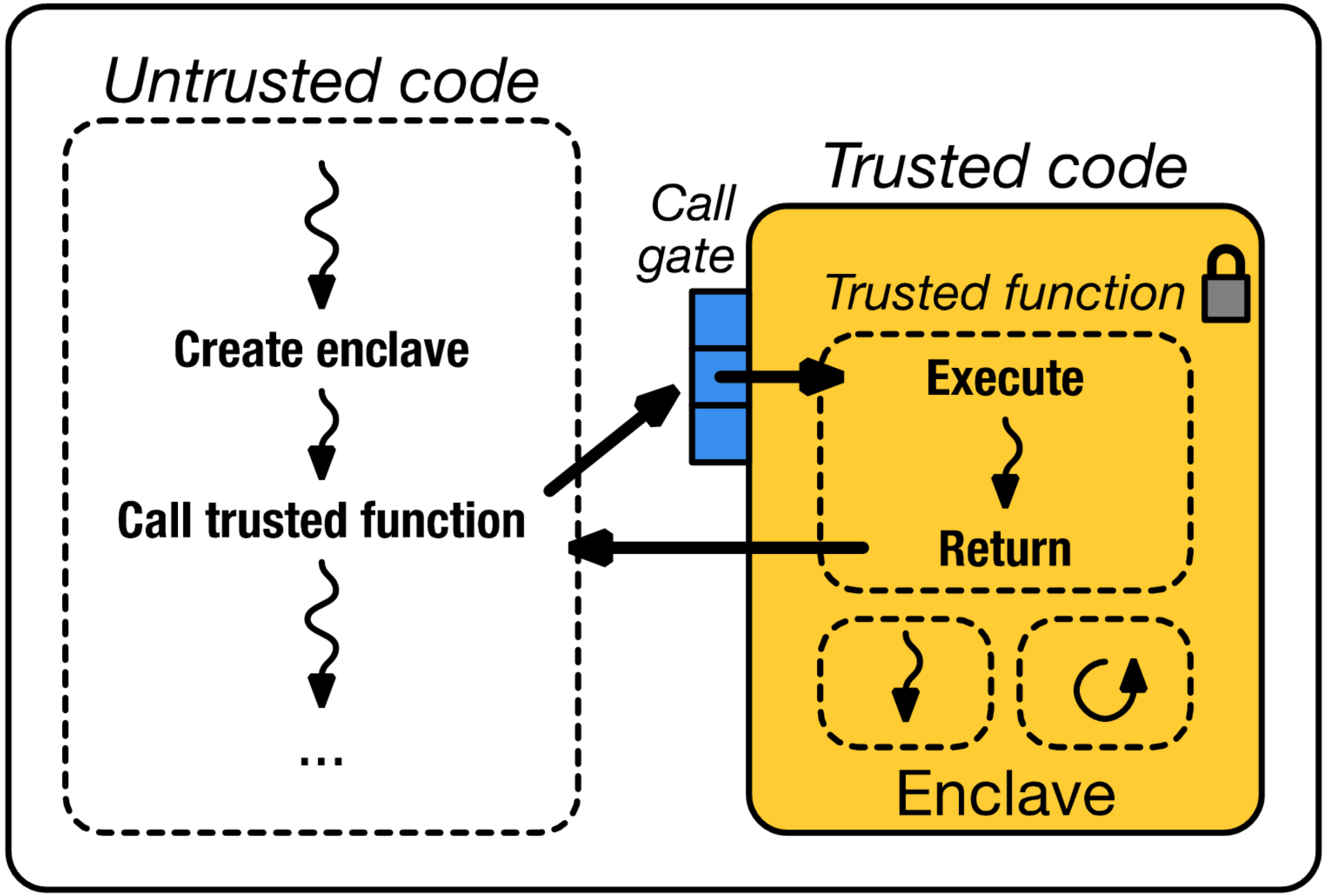
SGX-LKL: Library OS for complex complex Linux applications

- Based on the standard Linux Kernel
- Trusted file system and network stack
- User-level threading and asynchronous system calls
- Lean host OS interface

SGX-Spark: Transparent SGX enclave support for Spark

- Uses SGX-LKL to run Oracle HotSpot JVM
- Designed around SGXSparkExecutor
- Transparent encryption for RDDs

Enclave Transitions



Intel Software Guard Extensions (SGX)

SGX adds **new enclave execution mode**
- **New CPU instructions** to manipulate enclaves

Memory encryption engine (MEE) protects enclave memory
- Current enclave sizes restricted to 128 MB

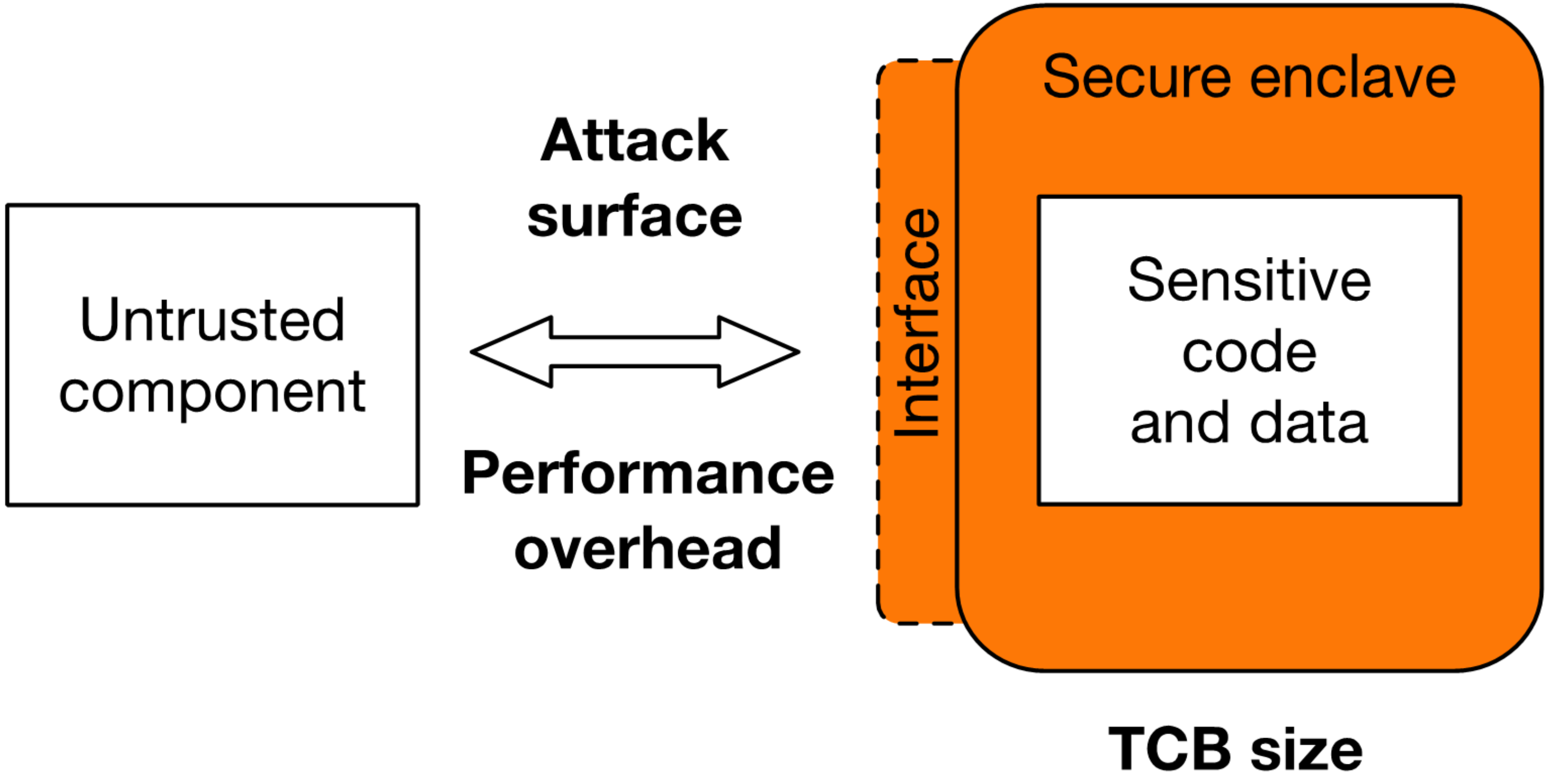
Support for **remote attestation**
- Permits clients to verify that they are interacting with genuine SGX enclave

Intel SGX SDK for Windows & Linux

SGX support available in **recent Intel CPUs**
- Skylake (2015), Kaby lake (2016)

☛ **SGX will become widely available on commodity CPUs**

Trade-Offs When Using Trusted Execution



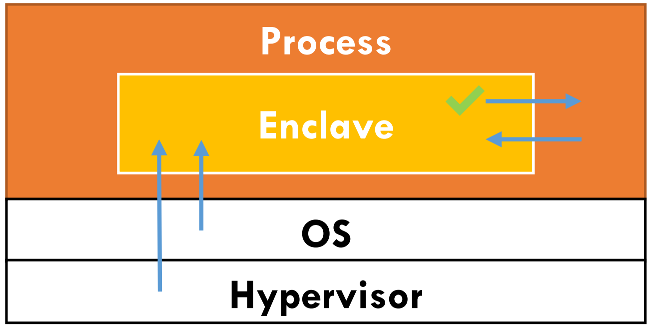
SGX Enclaves

SGX introduces notion of **enclave**

- Isolated memory region for code & data
- New CPU instructions to manipulate enclaves and new enclave execution mode

Enclave memory **encrypted** and **integrity-protected** by hardware

- Memory encryption engine (MEE)
- No plaintext secrets in main memory



Enclave memory can be accessed only by enclave code

- Protection from privileged code (OS, hypervisor)

Application has ability to defend secrets

- Attack surface reduced to just enclaves and CPU
- Compromised software cannot steal application secrets

Enclave Page Cache (EPC)

Physical memory region protected by MEE

- EPC holds enclave contents

Shared resource between all enclaves running on platform

- Currently only 128 MB
- ~96 MB available to user, rest for metadata

Content encrypted while in DRAM, decrypted when brought to CPU

- Plaintext in CPU caches

SGX Multithreading Support

SGX allows multiple threads to enter same enclave simultaneously

- One thread control structure (TCS) per thread
- Part of enclave, reflected in measurement

TCS limits number of enclave threads

- Upon thread entry TCS is blocked and cannot be used by another thread

Each TCS contains address of entry point

- Prevents jumps into random locations inside of enclave

SGX Paging

SGX provides mechanism to evict EPC page to unprotected memory

- EPC limited in size

Paging performed by OS

- Validated by HW to prevent attacks on address translations
- Metadata (MAC, version) kept within EPC

Accessing evicted page results in page fault

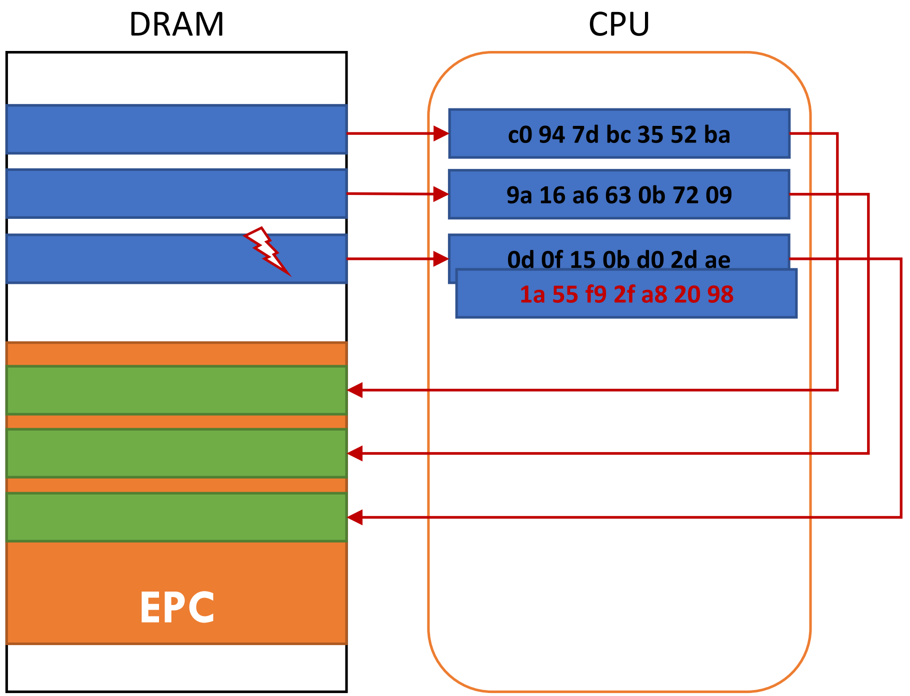
- Page is brought back into EPC by OS
- Hardware verifies integrity of page
- Another page might be evicted if EPC is full

SGX Enclave Measurement

CPU calculates enclave measurement hash during enclave construction

- Each new page extends hash with page content and attributes (read/write/execute)
- Hash computed with SHA-256

Measurement can be used to attest enclave to local or remote entity



CPU calculates enclave measurement hash during enclave construction

Different measurement if enclave modified

SGX Enclave Attestation

Is my code running on remote machine intact?

Is code really running inside an SGX enclave?

Local attestation

– Prove enclave's identity (= measurement) to another enclave on same CPU

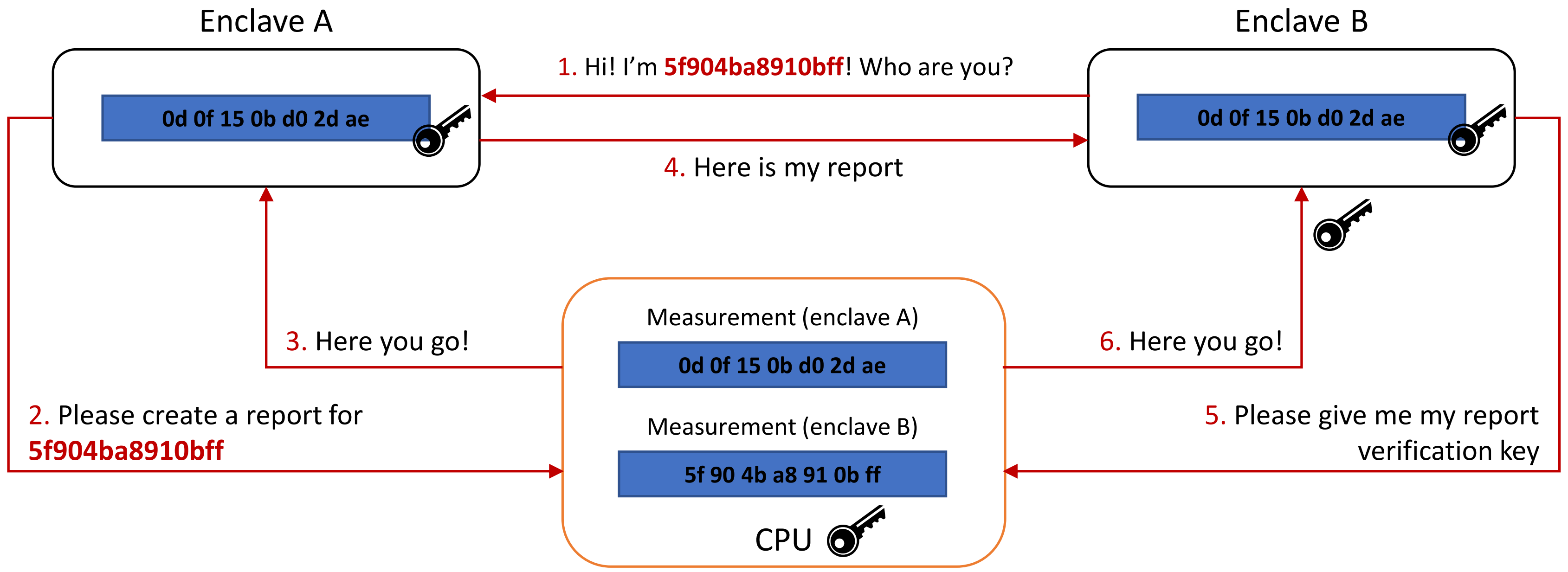
Remote attestation

– Prove enclave's identity to remote party

Once attested, enclave can be trusted with secrets

Local Attestation

Prove identity of A to local enclave B



1. Target enclave B measurement required for key generation
2. Report contains information about target enclave B, including its measurement
3. CPU fills in report and creates MAC using report key, which depends on random CPU fuses and target enclave B measurement
4. Report sent back to target enclave B
5. Verify report by CPU to check that generated on same platform, i.e. MAC created with same report key (available only on same CPU)
6. Check MAC received with report and do not trust A upon mismatch

Remote Attestation I

Transform local report to remotely verifiable “quote”

Based on provisioning enclave (PE) and quoting enclave (QE)
- Architectural enclaves provided by Intel
- Execute locally on user platform

Each SGX-enabled CPU has unique key fused during manufacturing
- Intel maintains database of keys

Remote Attestation II

PE communicates with Intel attestation service

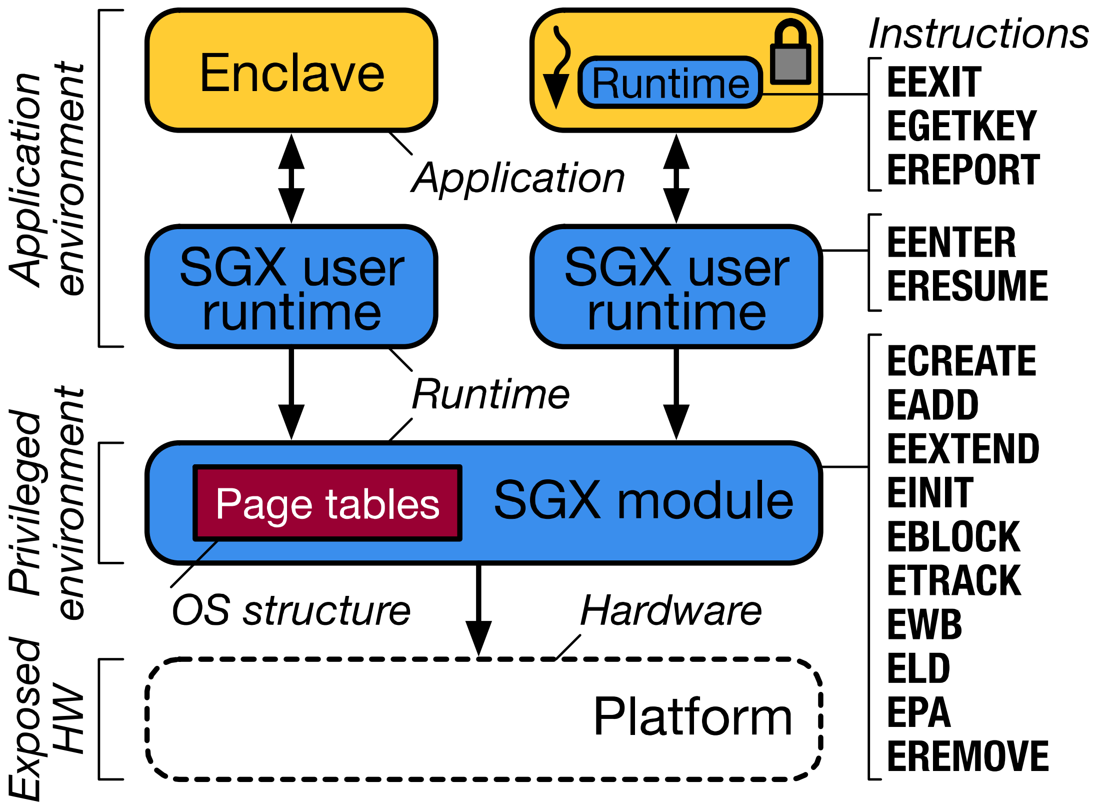
- Proves it has key installed by Intel
- Receives asymmetric attestation key

QE performs local attestation for enclave

- QE verifies report and signs it using attestation key
- Creates quote that can be verified outside platform

Quote and signature sent to remote attester, which communicates with Intel attestation service to verify quote validity

Summary of SGX Architecture



Thank you!

Microsoft