# ICT Risk Assessment

Fabrizio Baiardi
f.baiardi@unipi.it

## Syllabus

- Security
  - New Threat Model
  - New Attacks
  - Countermeasures

Cloud provider

Enclaves encryption + execution

## Outline

1. Motivation: Trustworthy data processing in untrusted clouds

2. Overview of Intel SGX

3. Description of SGX-LKL Design

4. Description of preliminary SGX-Spark Design

5. Source code release of Java support on GitHub

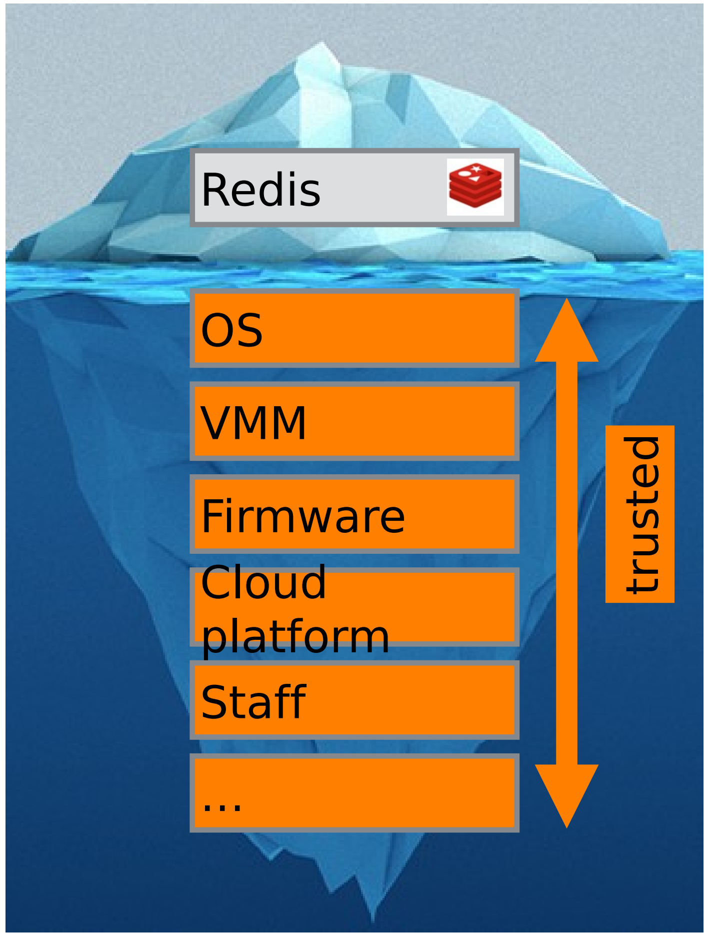
## 1. Motivation: Trustworthy Data Processing

# Trust Issues: Provider Perspective

Cloud provider does not trust users

Use virtual machines to isolate users from each other and the host

VMs only provide one way protection

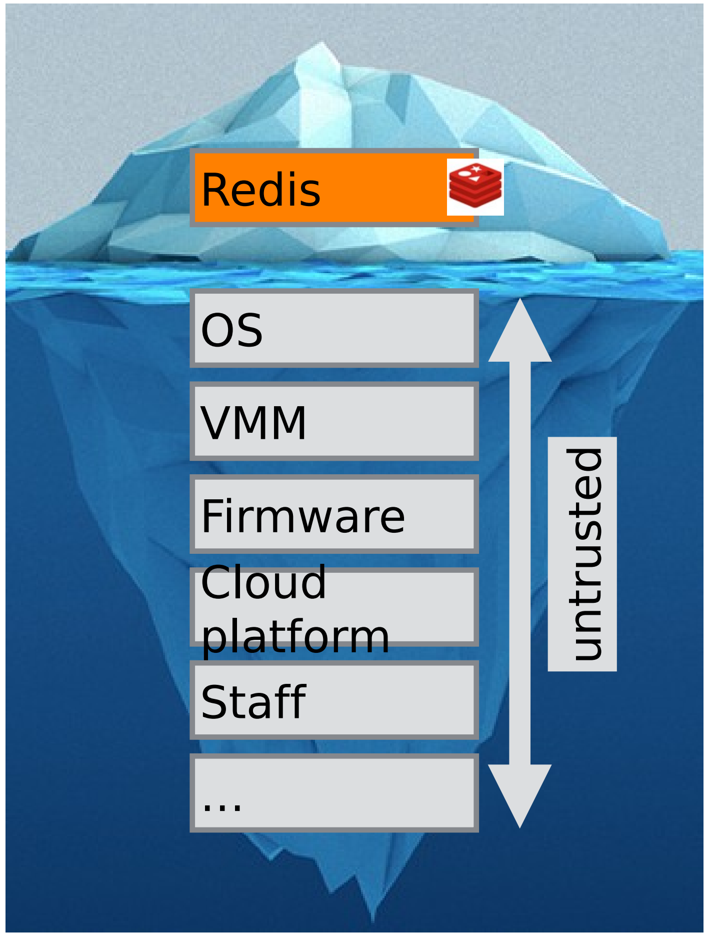


# Trust Issues: User Perspective

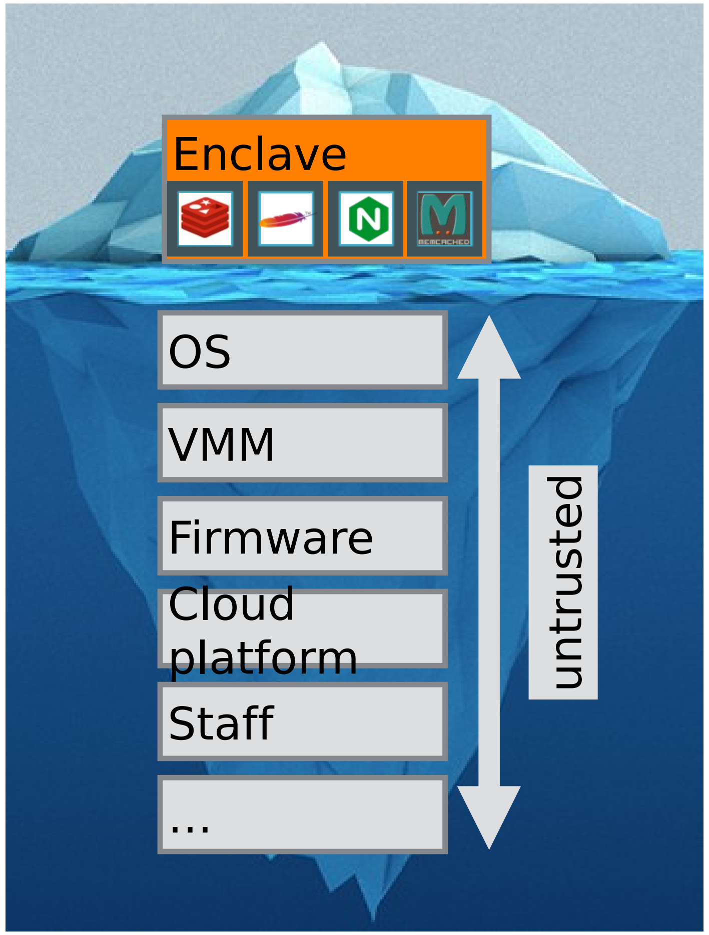
Users trust their applications

Users must implicitly trust cloud provider

Existing applications implicitly assume trusted operating system



# Trusted Execution Support with Intel SGX



Users create HW-enforced trusted environment (enclave)

Supports unprivileged user code

Protects against strong attacker model

Remote attestation

Available on commodity CPUs

# Trusted Execution Support with Intel SGX

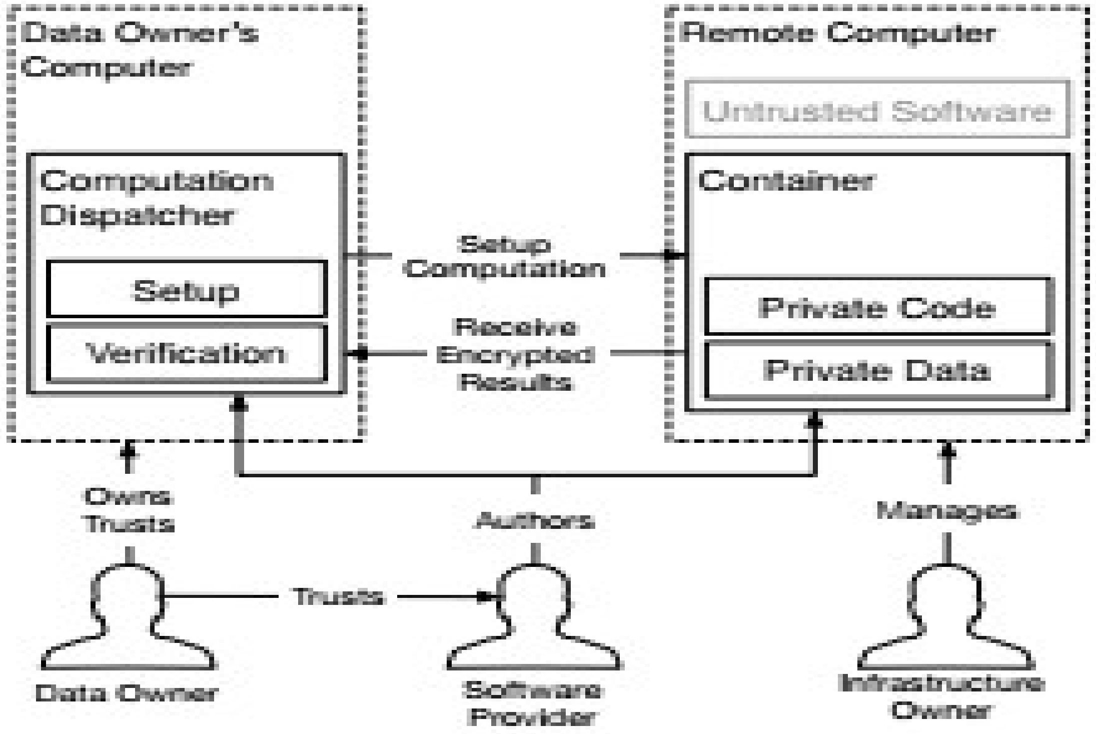
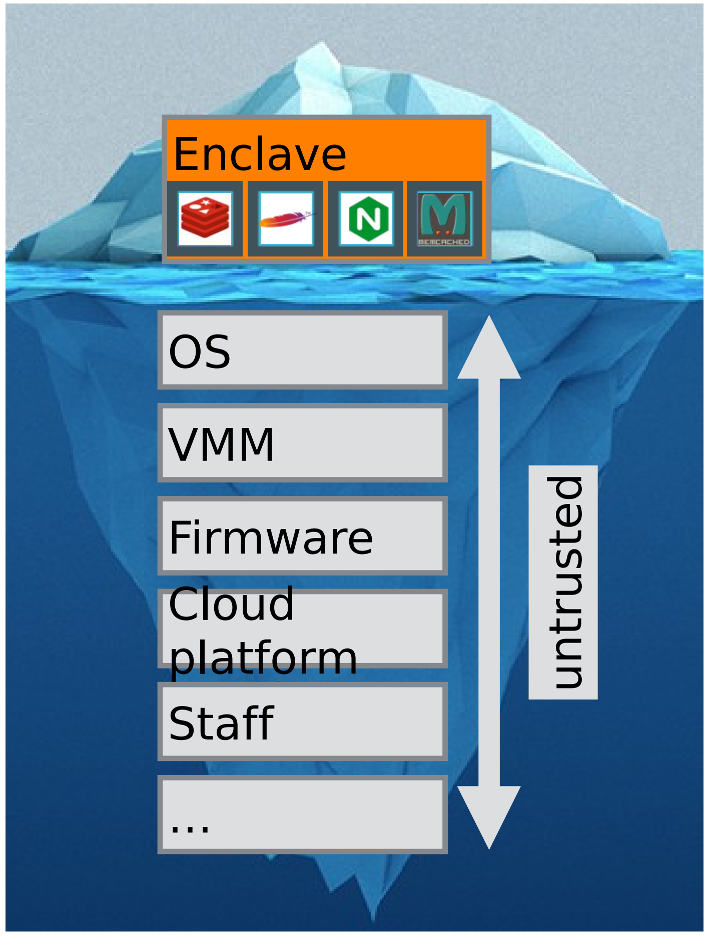


**Figure 1:** Secure remote computation. A user relies on a remote computer, owned by an untrusted party, to perform some computation on her data. The user has some assurance of the computation's integrity and confidentiality.

# Trusted Execution Support with Intel SGX



Users create HW-enforced trusted environment (enclave)

Supports unprivileged user code

Protects against strong attacker model

Remote attestation

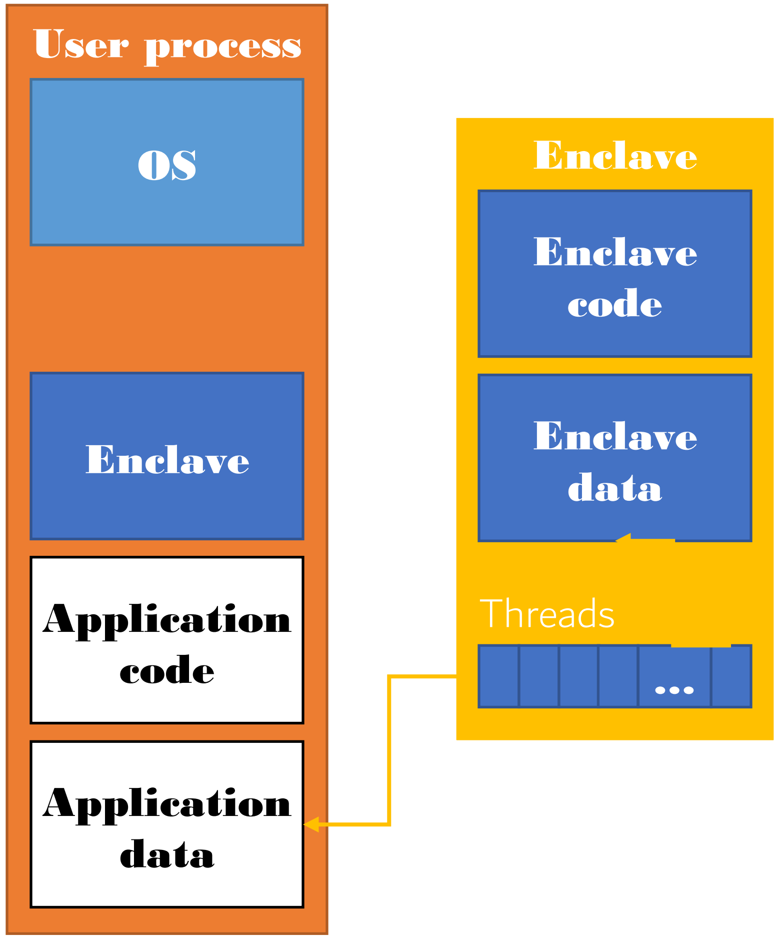
Available on commodity CPUs

## 2. Overview of Intel SGX

## Trusted Execution Environments

Trusted execution environment (TEE) in process

- Own code & data
- Controlled entry points
- Provides confidentiality & integrity
- Supports multiple threads
- Full access to application memory



## Intel Software Guard Extensions (SGX)

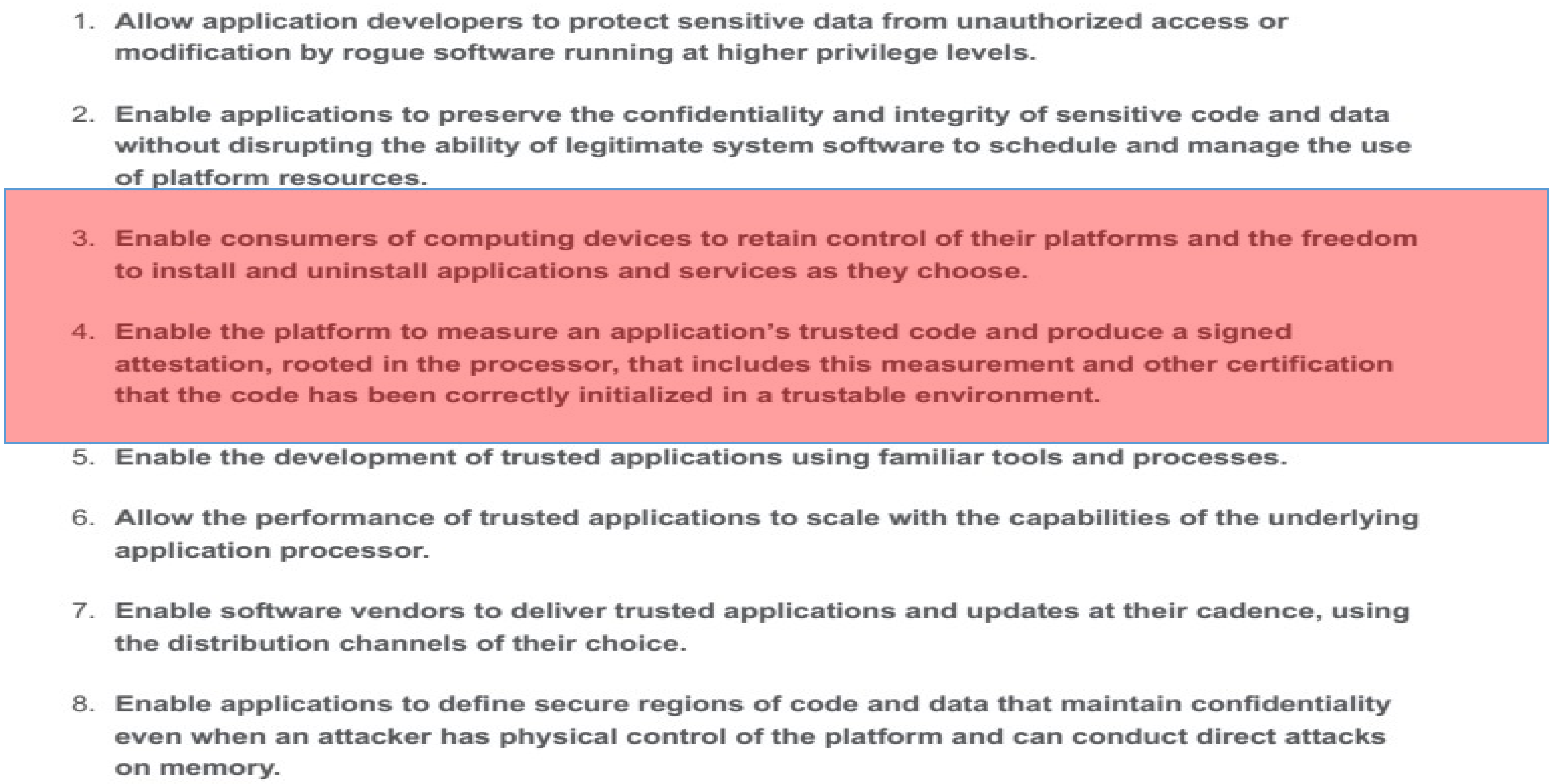
Extension of Instruction Set Architecture (ISA) in recent Intel CPUs

- Skylake (2015), Kaby lake (2016)

Protects confidentiality and integrity of code & data in untrusted environments

- Platform owner considered malicious
- Only CPU chip and isolated region trusted

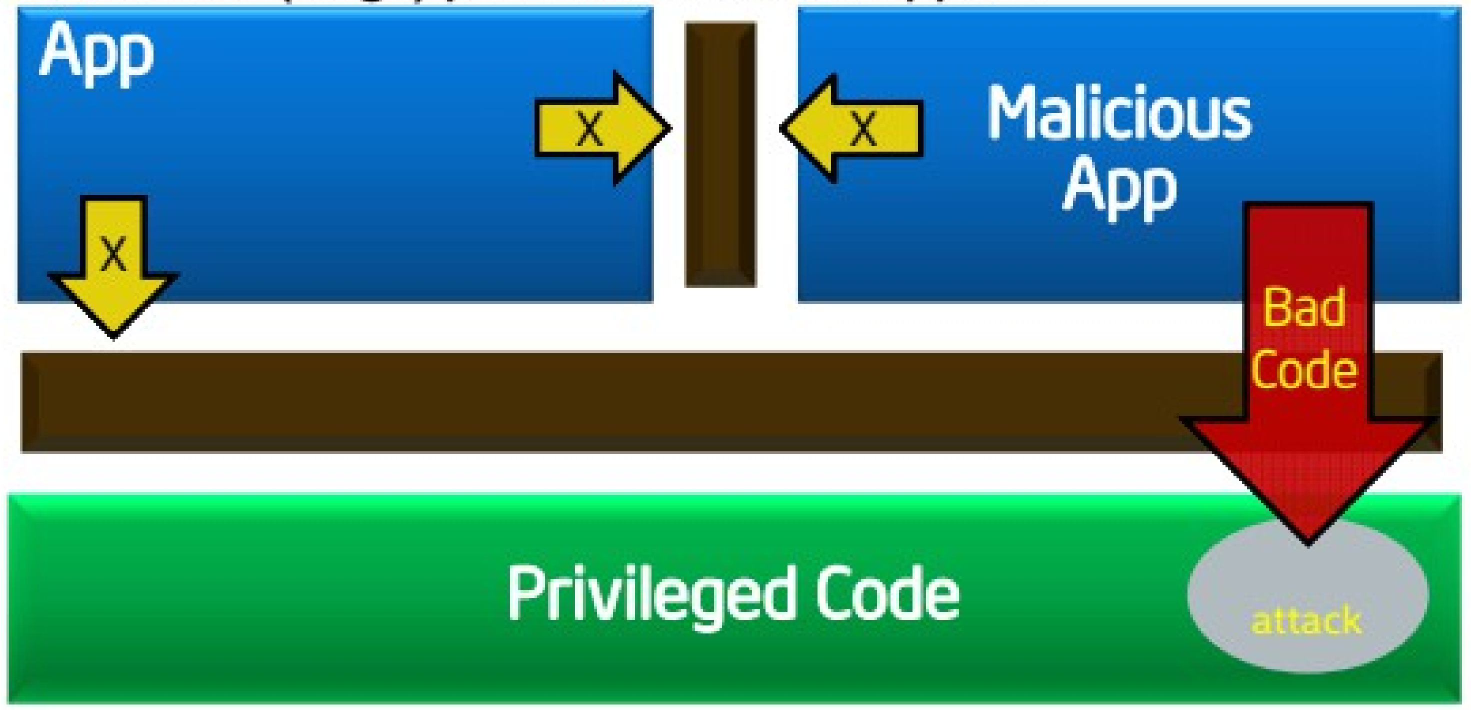
## In a few words



1. Allow application developers to protect sensitive data from unauthorized access or modification by rogue software running at higher privilege levels.
2. Enable applications to preserve the confidentiality and integrity of sensitive code and data without disrupting the ability of legitimate system software to schedule and manage the use of platform resources.
3. Enable consumers of computing devices to retain control of their platforms and the freedom to install and uninstall applications and services as they choose.
4. Enable the platform to measure an application's trusted code and produce a signed attestation, rooted in the processor, that includes this measurement and other certification that the code has been correctly initialized in a trustable environment.
5. Enable the development of trusted applications using familiar tools and processes.
6. Allow the performance of trusted applications to scale with the capabilities of the underlying application processor.
7. Enable software vendors to deliver trusted applications and updates at their cadence, using the distribution channels of their choice.
8. Enable applications to define secure regions of code and data that maintain confidentiality even when an attacker has physical control of the platform and can conduct direct attacks on memory.

## The Basic Issue: Why Aren't Compute Devices Trustworthy?

Protected Mode (rings) protects OS from apps ...



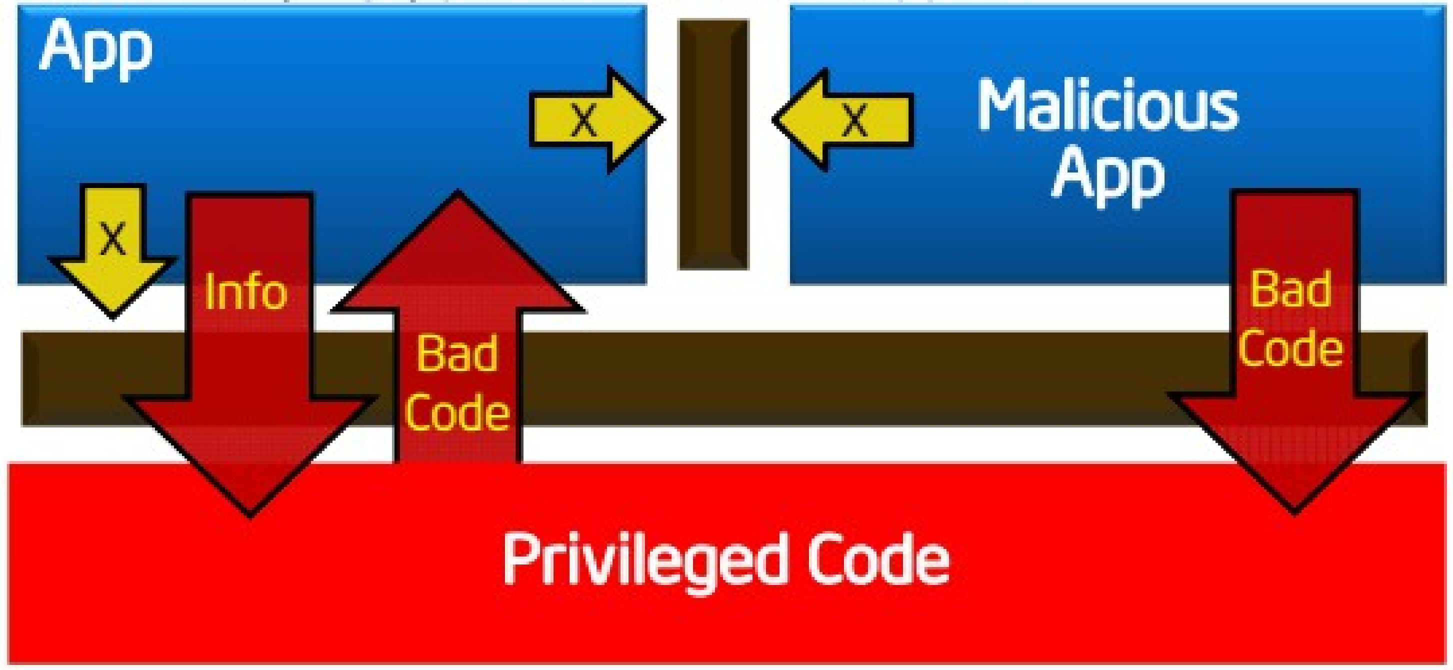
... and apps from each other ...

... UNTIL a malicious app exploits a flaw to gain full privileges and then tampers with the OS or other apps

**Apps not protected from privileged code attacks**

## The Basic Issue: Why Aren't Compute Devices Trustworthy?

Protected Mode (rings) protects OS from apps ...



... and apps from each other ...

... UNTIL a malicious app exploits a flaw to gain full privileges and then tampers with the OS or other apps

**Apps not protected from privileged code attacks**

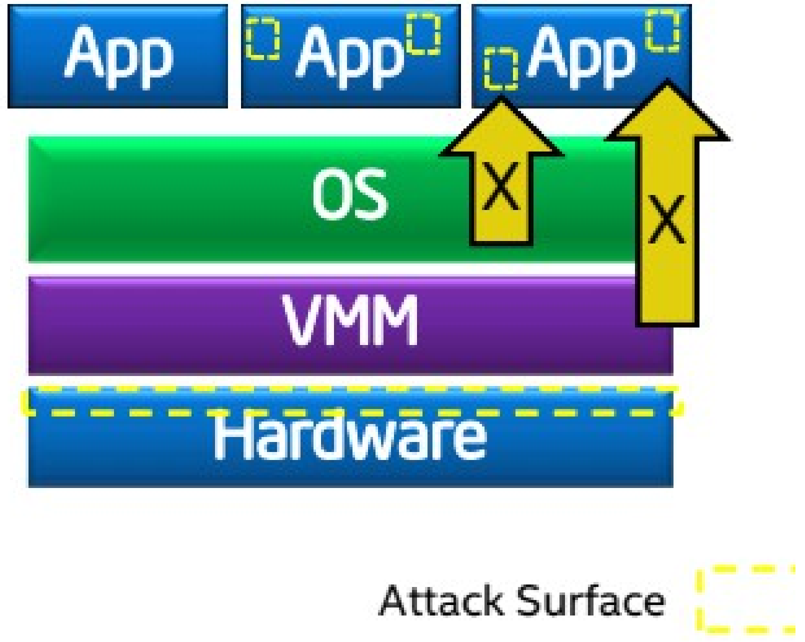
## Application gains ability to defend its own secrets

- Smallest attack surface (App + processor)
- Malware that subverts OS/VMM, BIOS, Drivers etc. cannot steal app secrets

## Familiar development/debug

- Single application environment
- Build on existing ecosystem expertise

### Attack surface with Enclaves



## 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

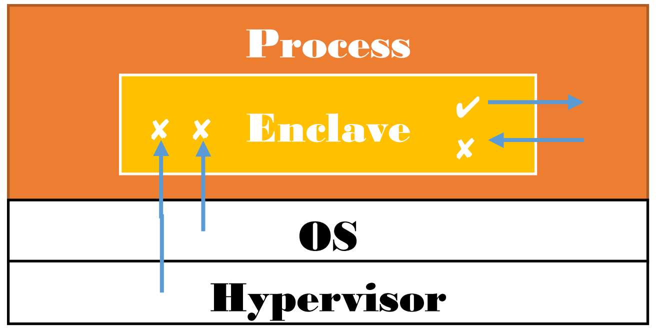
- Memory encryption engine (MEE)
- No plaintext secrets in main memory

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



## Reduced attack surface with SGX

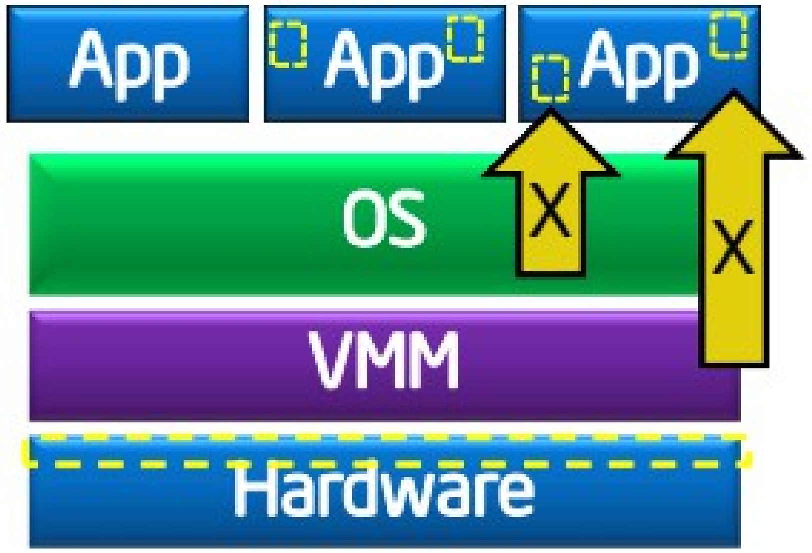
### Application gains ability to defend its own secrets

- Smallest attack surface (App + processor)
- Malware that subverts OS/VMM, BIOS, Drivers etc. cannot steal app secrets

### Familiar development/debug

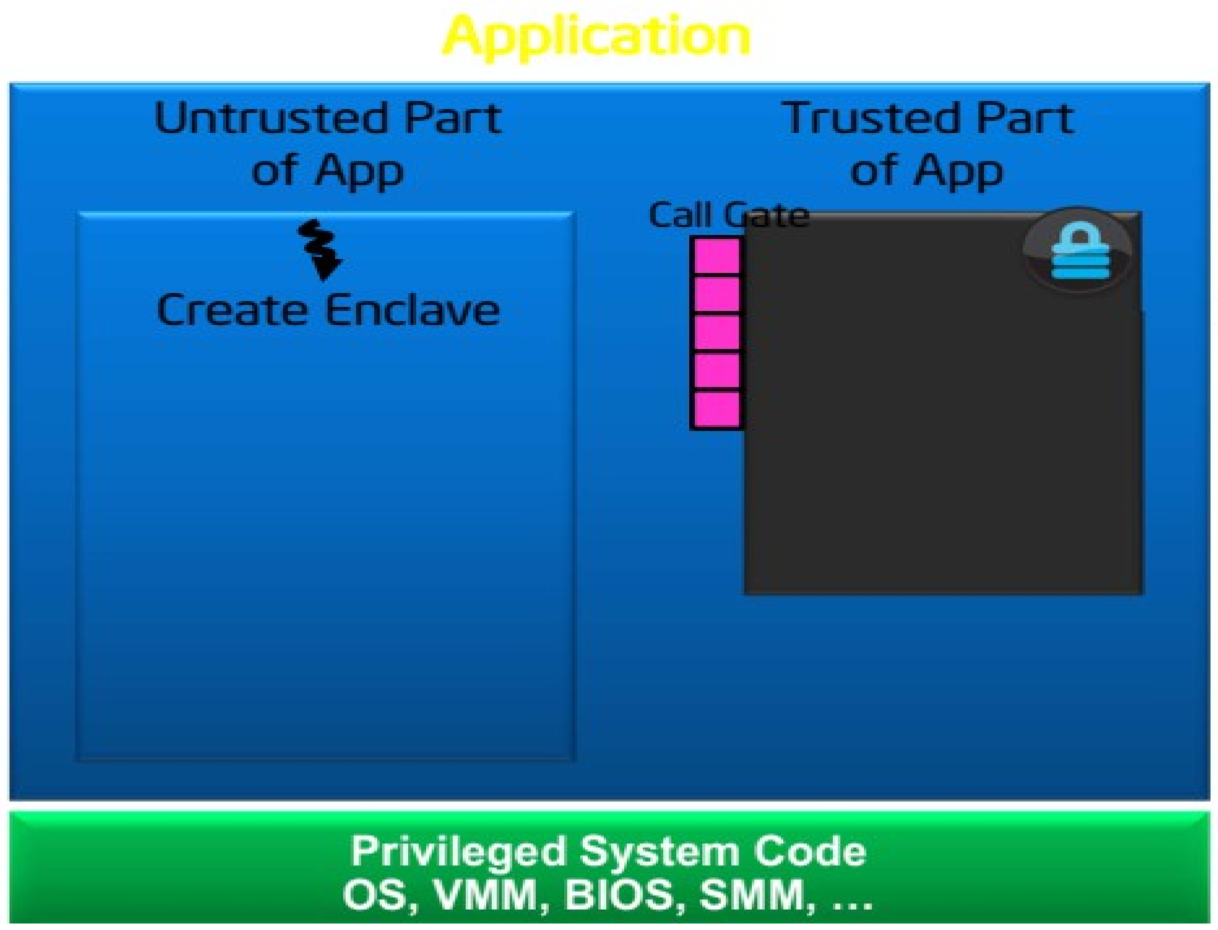
- Single application environment
- Build on existing ecosystem expertise

Attack surface with Enclaves



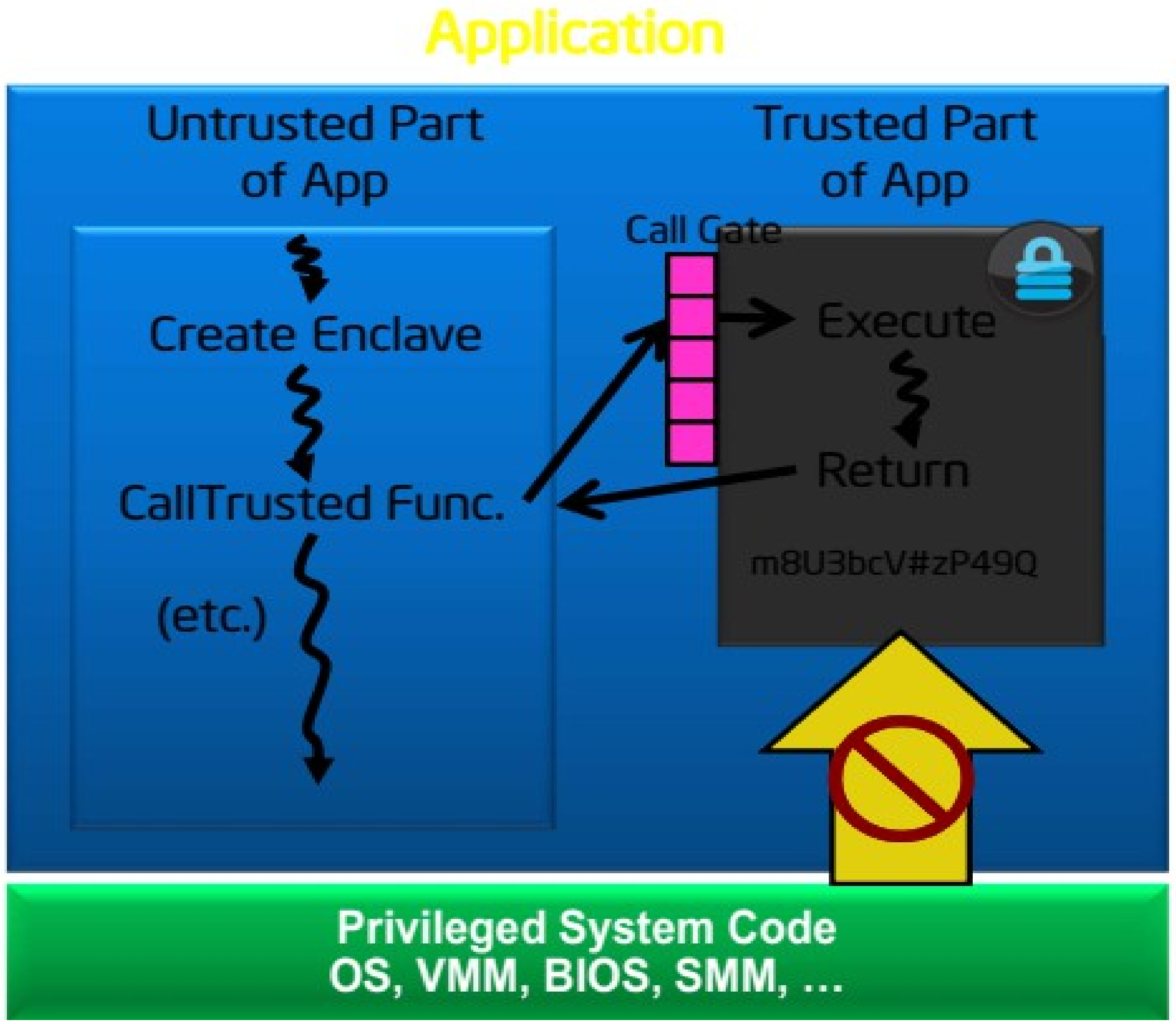
Attack Surface ⬚

## How SE Works: Protection vs. Software Attack



1. App is built with trusted and untrusted parts
2. App runs & creates enclave which is placed in trusted memory

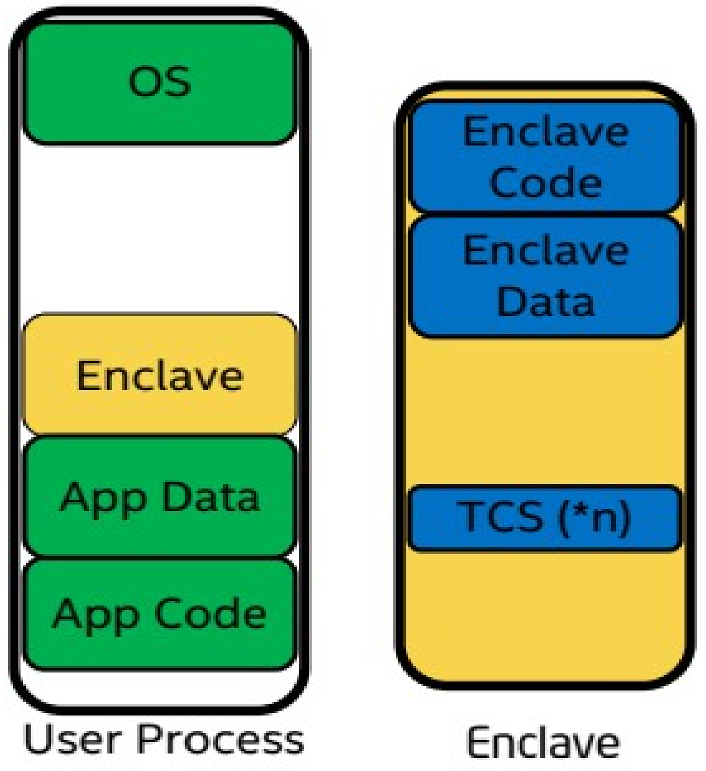
## How SE Works: Protection vs. Software Attack



1. App is built with trusted and untrusted parts
2. App runs & creates enclave which is placed in trusted memory
3. Trusted function is called; code running inside enclave sees data in clear; external access to data is denied
4. Function returns; enclave data remains in trusted memory

## SGX Programming Environment

**Trusted execution environment embedded in a process**



With its own code and data

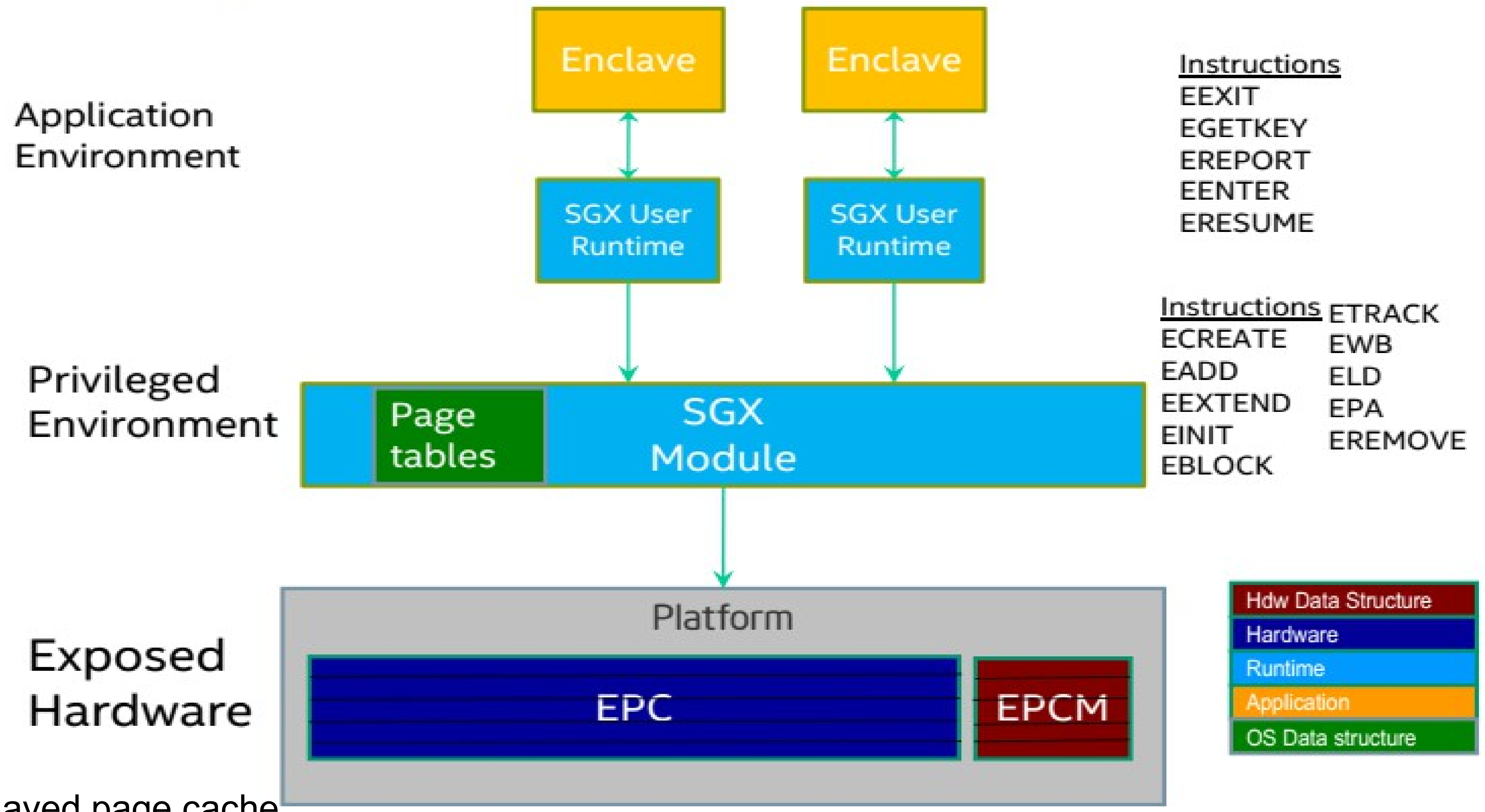
Provide Confidentiality

Provide integrity

With controlled entry points

Supporting multiple threads

## SGX High-level HW/SW Picture



Instructions
EEXIT
EGETKEY
EREPORT
EENTER
ERESUME

Instructions
ECREATE
EADD
EEXTEND
EINIT
EBLOCK
ETRACK
EWB
ELD
EPA
EREMOVE

Enclaved page cache

## Memory Access Control

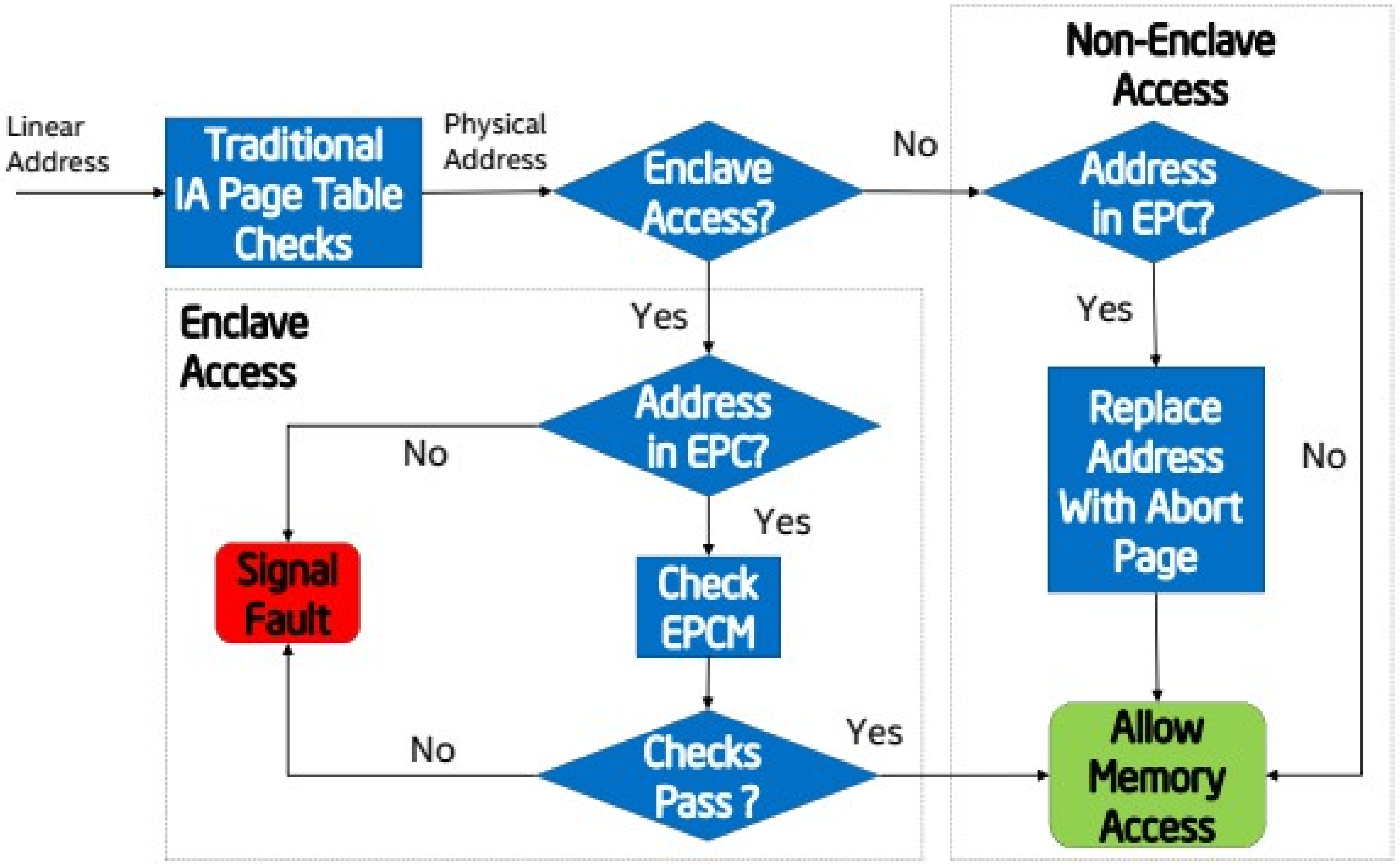
MAC from enclaves to "outside":

- All memory access has to conform to segmentation and paging policies by the OS/VMM.
- Enclaves cannot manipulate those policies, only unprivileged instructions inside an enclave.
- Code fetches from inside an enclave to a linear address outside that enclave will results in a General Protection Fault (0)exception.
-

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 General Protection Fault (0).

## SGX Access Control



## SGX Instructions and Data Structures:

### 18 Instruction

- 13 Supervisor Instructions.
- 5 User Instructions.

### 13 Data Structures

- 8 data structures associated to a certain enclave.
- 3 data structures associated to certain memory page(s).
- 2 data structures associated to overall resource management.

## SGX Supervisor Instructions:

| | |
|---|---|
| ENCLS[EADD] | Add a page |
| ENCLS[EBLOCK] | Block an EPC page |
| ENCLS[ECREATE] | Create an enclave |
| ENCLS[EDBGRD] ENCLS[EDBGWR] | Read/Write data by debugger |
| ENCLS[EEXTEND] | Extend EPC page measurement |
| ENCLS[EINIT] | Initialize an enclave |
| ENCLS[ELDB] | Load an EPC page as blocked |
| ENCLS[ELDU] | Load an EPC page as unblocked |
| ENCLS[EPA] | Add version array |
| ENCLS[EREMOVE] | Remove a page from EPC |
| ENCLS[ETRACK] | Activate EBLOCK checks |
| ENCLS[EWB] | Write back/invalidate an EPC page |

## SGX User Instructions:

| **User Instruction** | **Description** |
|---|---|
| ENCLU[EENTER] | Enter an Enclave |
| ENCLU[EEXIT] | Exit an Enclave |
| ENCLU[EGETKEY] | Create a cryptographic key |
| ENCLU[EREPORT] | Create a cryptographic report |
| ENCLU[ERESUME] | Re-enter an Enclave |

# SGX Data Structures in Details:

| | |
|---|---|
| **SGX Enclave Control Structure (SECS)** | Represents one enclave and it store Hash, ID, size etc. |
| **Thread Control Structure (TCS)** | one for each thread in the enclave. It stores Entry point, pointer to SSA. |
| **State Save Area (SSA)** | It save the state of the running threat when an AEX occurs |
| **Page Information (PAGEINFO)** | data structure used as a parameter to the EPC-management instruction |
| | Linear Address, Effective address of the page (aka virtual address) |
| | SECINFO + SECS |
| **Security Information (SECINFO)** | Meta-data about an enclave page |
| | R/W/X,Page type (SECS, TCS, normal page or VA) |
| **Paging Crypto MetaData (PCMD):** | Crypto meta-data of a paged-out page. With PAGEINFO it used to verify, decrypt, and reload a paged-out pag |
| | EWB writes out (the reserved field and) MAC values. |

## SGX Data Structures in Details:

### Version Array (VA)

Each VA page is an EPC page to securely store the versions of evicted EPC pages with  512 slots, each with an 8-byte version number for a page evicted from the EPC.

- When an EPC page is evicted, an empty slot in a VA page receives the unique version number of the  evicted page
- When the EPC page is reloaded, a VA slot must hold the page version when the VA slot is cleared.
- When evicting a VA page, a version slot in some other VA page must be used to receive the version for the VA being evicted.

# SGX Data Structures in Details:

**Enclave Page Cache Map (EPCM):** used by the processor to track the contents of the EPC.

- EPCM is a secure structure the processor uses to track the contents of the EPC. It holds exactly one entry for each page currently loaded into the EPC. EPCM is not accessible by software, and the field layout is implementation specific. Contains, for instance, RWX, page type, linear address, state etc.one entry for each page currently loaded into the EPC.

**Enclave Signature Structure (SIGSTRUCT):** information about the enclave from the enclave signer.

- ENCLAVEHASH as SHA256 and four 3072-bit integers (MODULUS, SIGNATURE, Q1, Q2).

**EINT Token Structure (EINITTOKEN):**

- used by EINIT to verify that the enclave is permitted to launch.
- Contains, attributes, hash and signer of the enclave.
- Authenticated with a cryptographic MAC on EINITTOKEN using Launch key.

**Report (REPORT):** the output of the EREPORT instruction

  - Attributes of the enclave
  - Hash and signer of the enclave + data for communication between the enclave and the target enclave
  - A CMAC on the report using report key

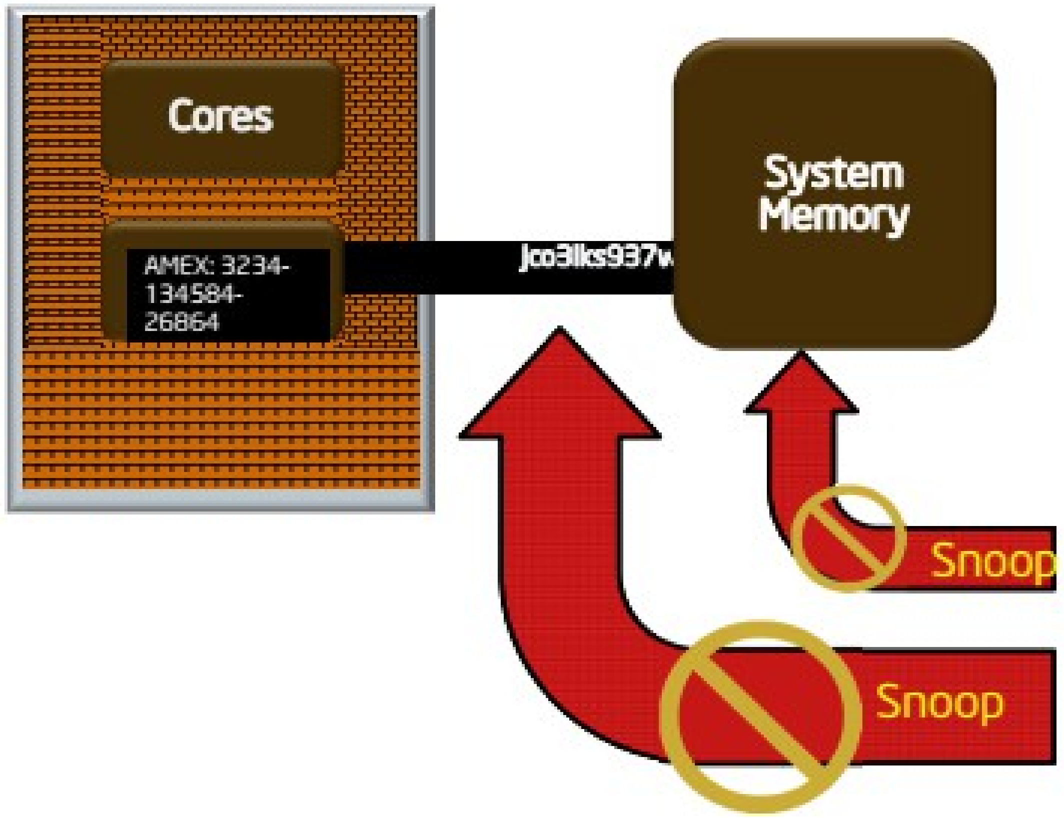
**Report Target Info (TARGETINFO):**

- An input parameter to EREPORT to identify the enclave able to cryptographically verify the REPORT structure returned by EREPORT.
- Contains attributes and hash of target enclave.

**Key Request (KEYREQUEST):**

- An input parameter to the EGETKEY instruction to select the key and any additional parameters to derive that key.

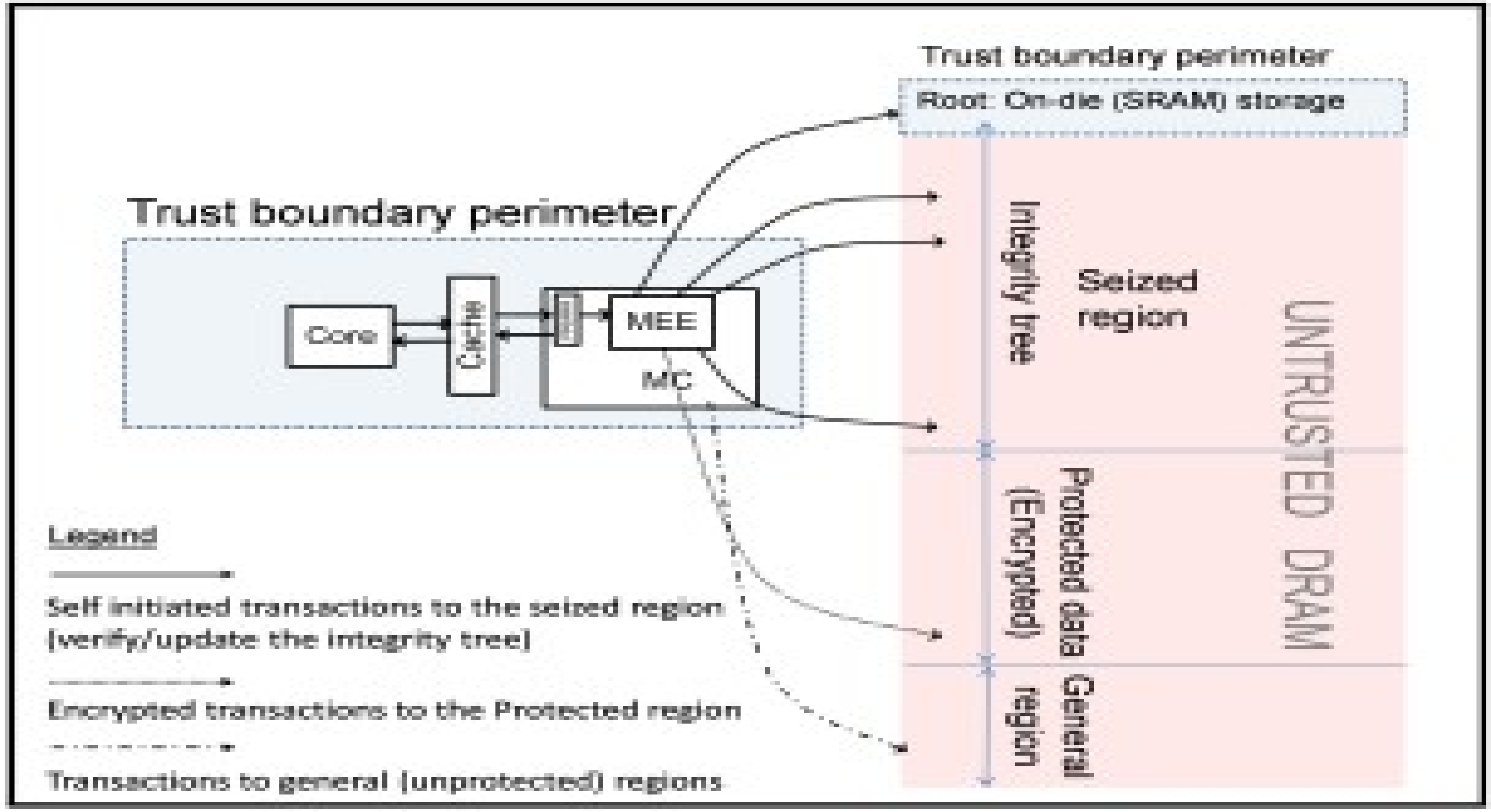
## Protection vs. Memory Snooping Attacks



**Non-Enclave Access**

- Security perimeter is the CPU package boundary
- Data and code unencrypted inside CPU package
- Data and code outside CPU package is encrypted and/or integrity checked
- External memory reads and bus snoops see only encrypted data

# Memory Encryption Engine



Trust boundary perimeter
Root: On-die (SRAM) storage
Trust boundary perimeter
Core
Cache
MEE
MC
Integrity tree
Seized region
UNTRUSTED DRAM
Protected data (Encrypted)
General region
Legend
Self initiated transactions to the seized region (verify/update the integrity tree)
Encrypted transactions to the Protected region
Transactions to general (unprotected) regions

## Memory Encryption Engine

**Objective 1.** *Providing confidentiality for the data that is written to the Protected region (on the DRAM).*

**Objective 2.** *Data integrity with replay prevention, assuring that data which is read back from the DRAM's Protected region to the CPU, is the same data that was most recently written from the CPU to the DRAM.*

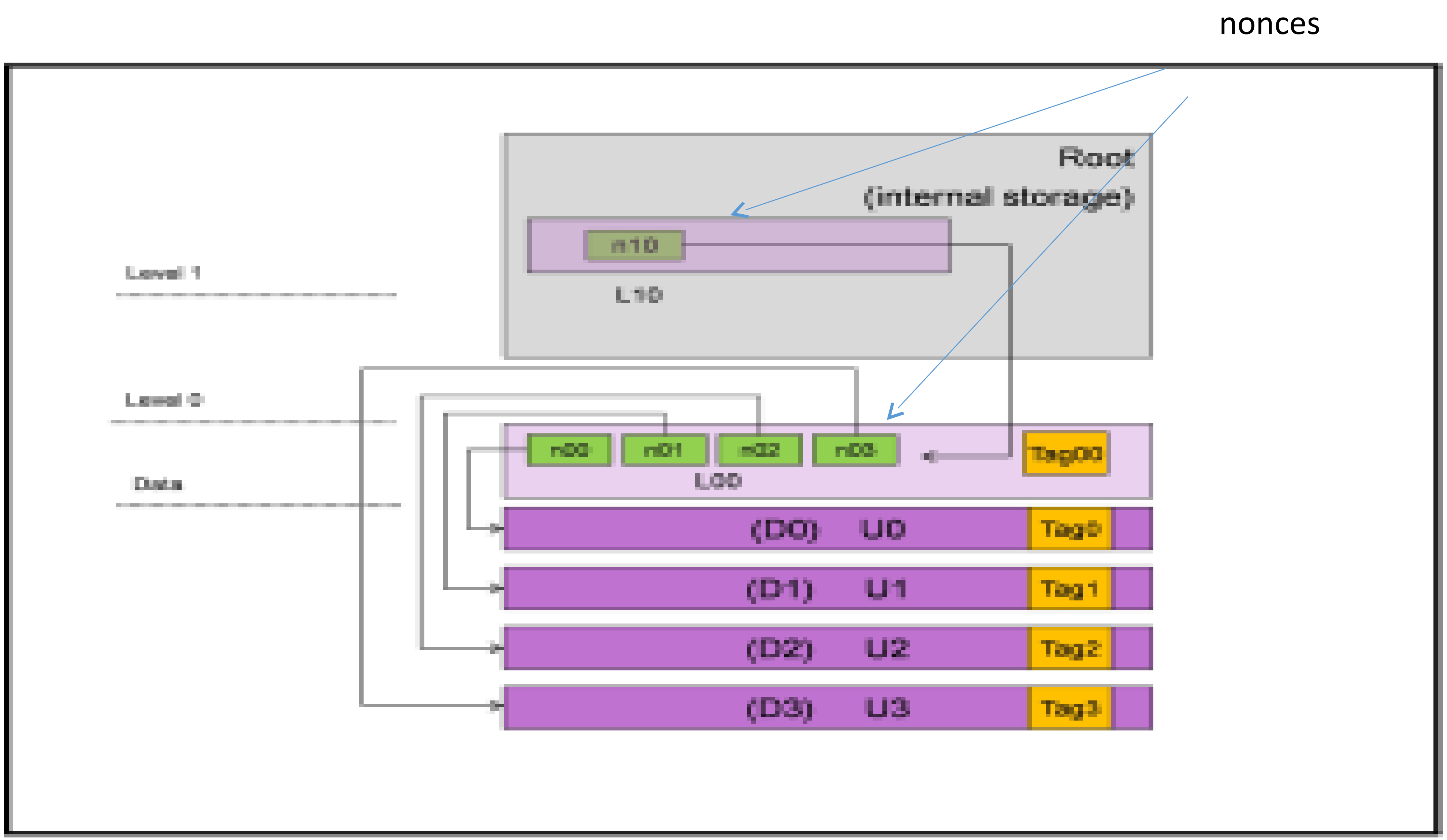
**Remark 1.** *The MEE is not designed to be an Oblivious RAM. An adversary with the assumed ability to track DRAM changes over time, can, by definition, carry out traffic analysis. He can learn when CL's are written, and to which CL addresses (though the contents of this traffic remains confidential). Preventing such analysis is not an objective of the MEE.*

**Property 1.** *The MEE keys are generated uniformly at random at boot time, and never leave the die.*

**Property 2.** *The encryption keys and the authentication keys are separate.*

**Property 3** (Drop-and-lock policy). *Tree verifications (and updates) enforce the following "drop-and-lock" policy. The MEE computes the MAC tags of data that it reads, and compares them to expected values, fetched from the integrity tree on the DRAM. If all comparisons match, the operation continues. However, as soon as any mismatch is detected, the MEE emits a failure signal, drops the transaction (i.e., no unverified data ever reaches the cache) and immediately locks the MC (i.e., no further transactions are serviced). This causes the system to hang, and it needs to be re-booted. After re-boot, the MEE starts over with newly generated keys.*

# Memory Encryption Engine: Integrity Tree



# Memory Encryption Engine: Performance

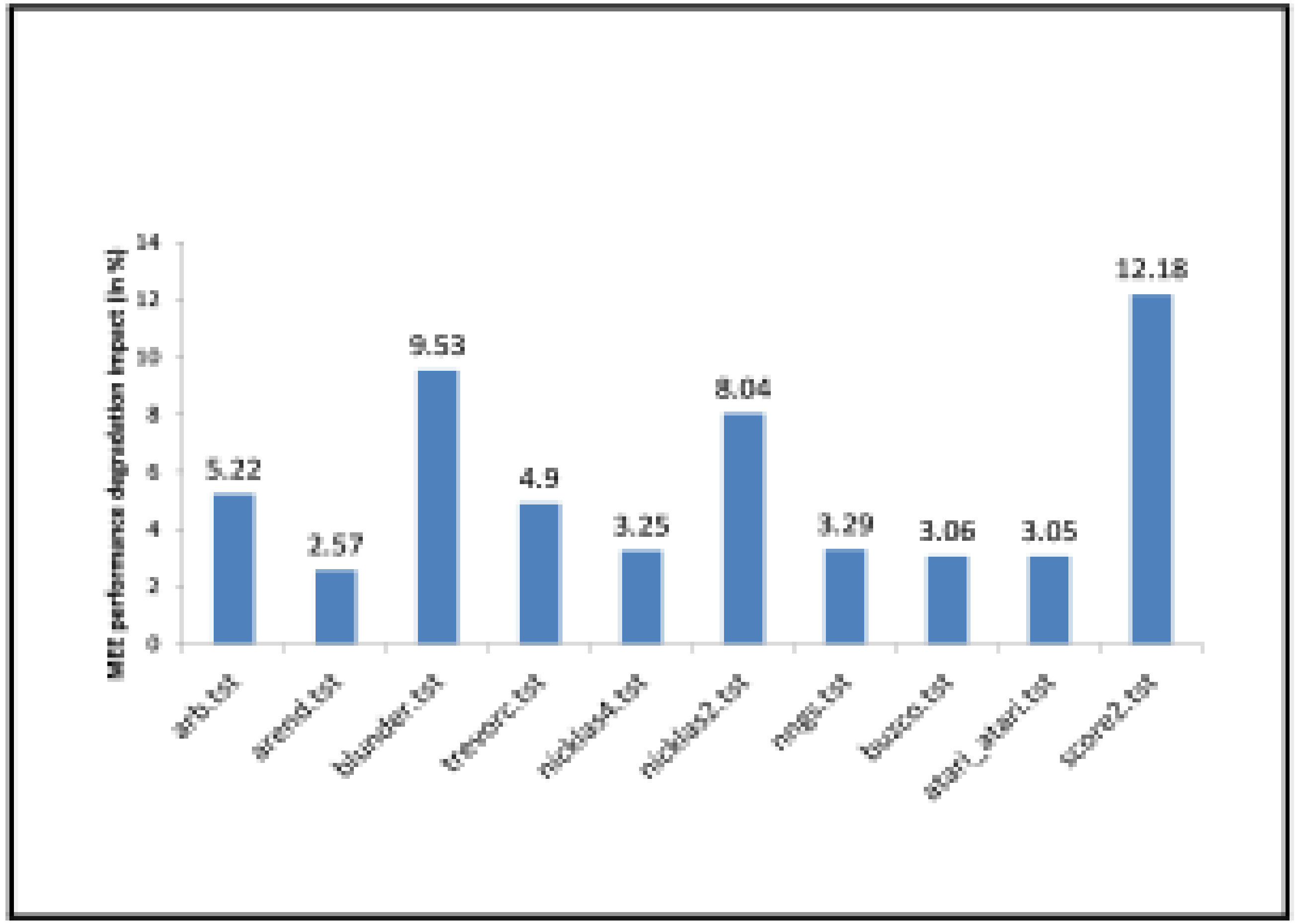


Figure 5: Performance comparison of the 445.gobmk component of SPECINT 2006, with 10 input files (see explanations in the text). The bars show that the performance degradation (in %) incurred by enabling the MEE, varies from 2.2% to 14%, with an average of 5.5%.

## SGX SDK Code Sample

**SGX application: untrusted code**

```
char request_buf[BUFFER_SIZE];
char response_buf[BUFFER_SIZE];

int main()
{
  ...
  while(1)
  {
    receive(request_buf);
    ret = EENTER(request_buf, response_buf);
    if (ret < 0)
      fprintf(stderr, "Corrupted message\n");
    else
      send(response_buf);
  }
  ...
}
```

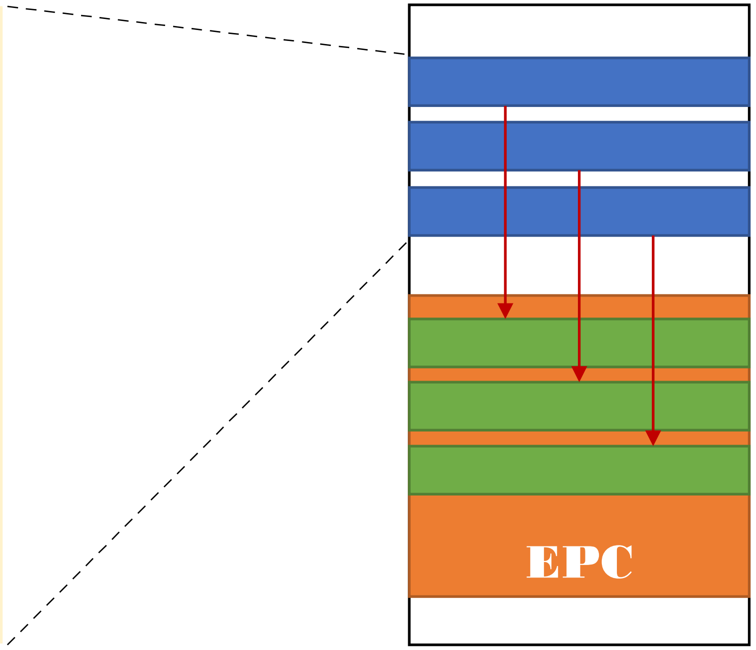
**Enclave: trusted code**

```
char input_buf[BUFFER_SIZE];
char output_buf[BUFFER_SIZE];

int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

- Receives encrypted requests
- Processes them in enclave
- Sends encrypted responses

# SGX Enclave Construction

```
char input_buf[BUFFER_SIZE];
char output_buf[BUFFER_SIZE];

int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```



Enclave populated using special instruction (EADD)
- Contents initially in untrusted memory
- Copied into EPC in 4KB pages

Both data & code copied before execution commences in enclave

# SGX Enclave Construction

Enclave contents distributed in plaintext

- Must not contain any (plaintext) confidential data

Secrets provisioned after enclave constructed and integrity verified

Problem: what if someone tampers with enclave?

- Contents initially in untrusted memory

```
int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

```
int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    copy_msg(output_buf, external_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

Write unencrypted response t

## 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

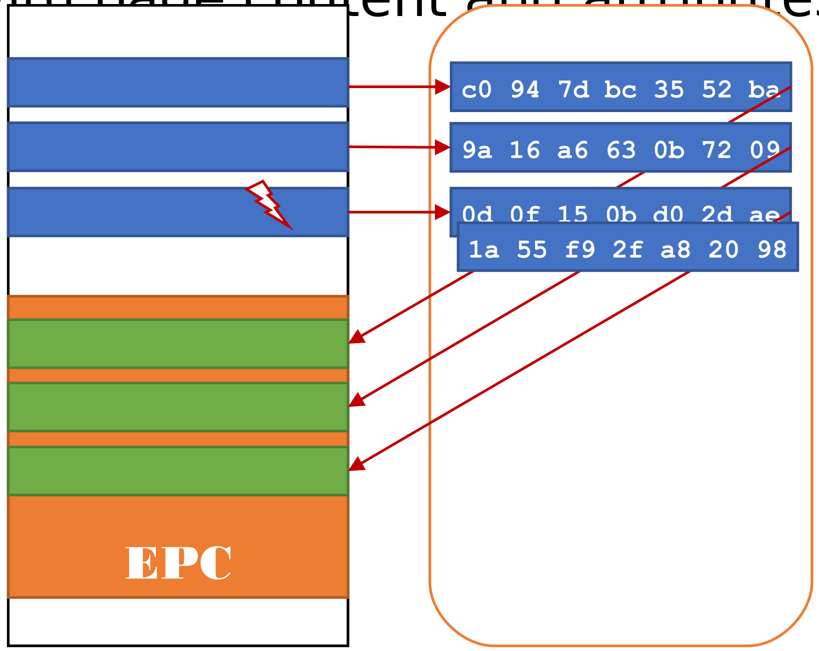
Once attested, enclave can be trusted with secrets

# 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

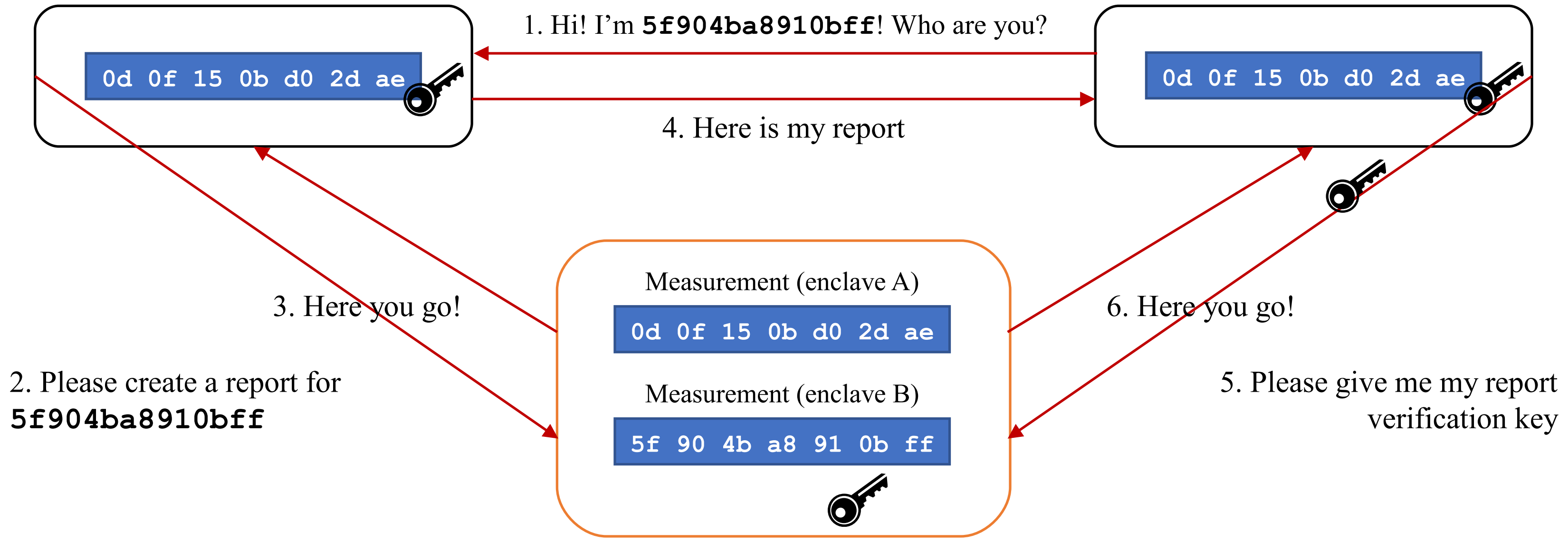
Measurement can be used to attest enclave to local or remote entity



CPU calculates enclave measurement hash during enclave construction
Different measurement if enclave modified

# 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

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

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

# SGX Limitations & Research Challenges

Amount of memory enclave can use needs to be known in advance

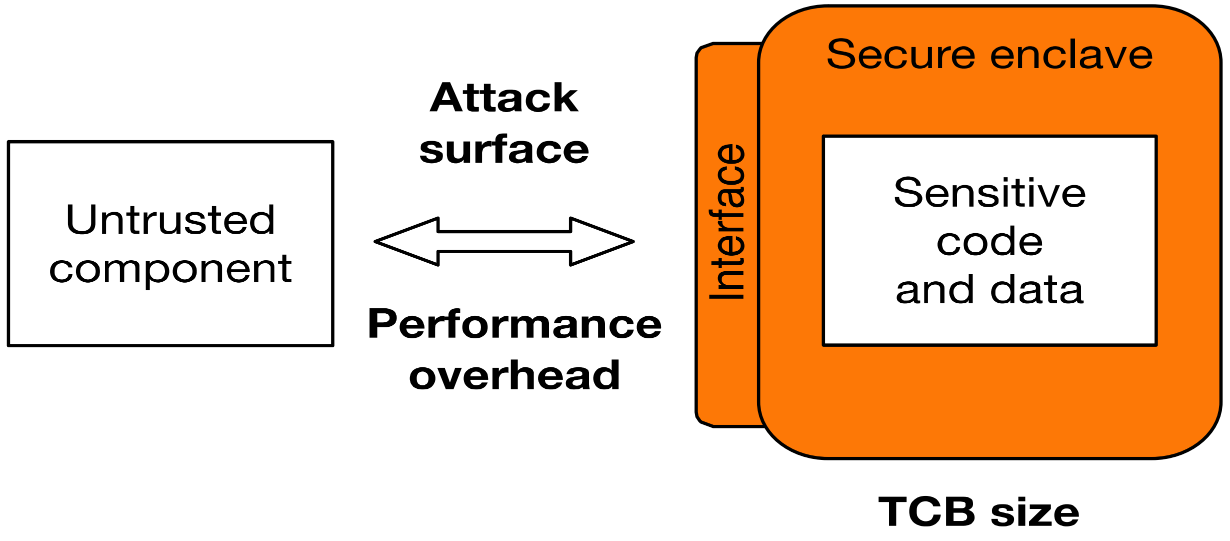
- Dynamic memory support in SGX v2

Security guarantees not perfect

- Vulnerabilities within enclave can still be exploited
- Side-channel attacks possible

Performance overhead

- Enclave entry/exit costly
- Paging very expensive



Application partitioning? Legacy code?

## 3. Description of SGX-LKL

## SGX-LKL: Supporting Managed Runtimes in SGX

Many applications need runtime support

- JVM
- .NET
- JavaScript/V8/Node.js

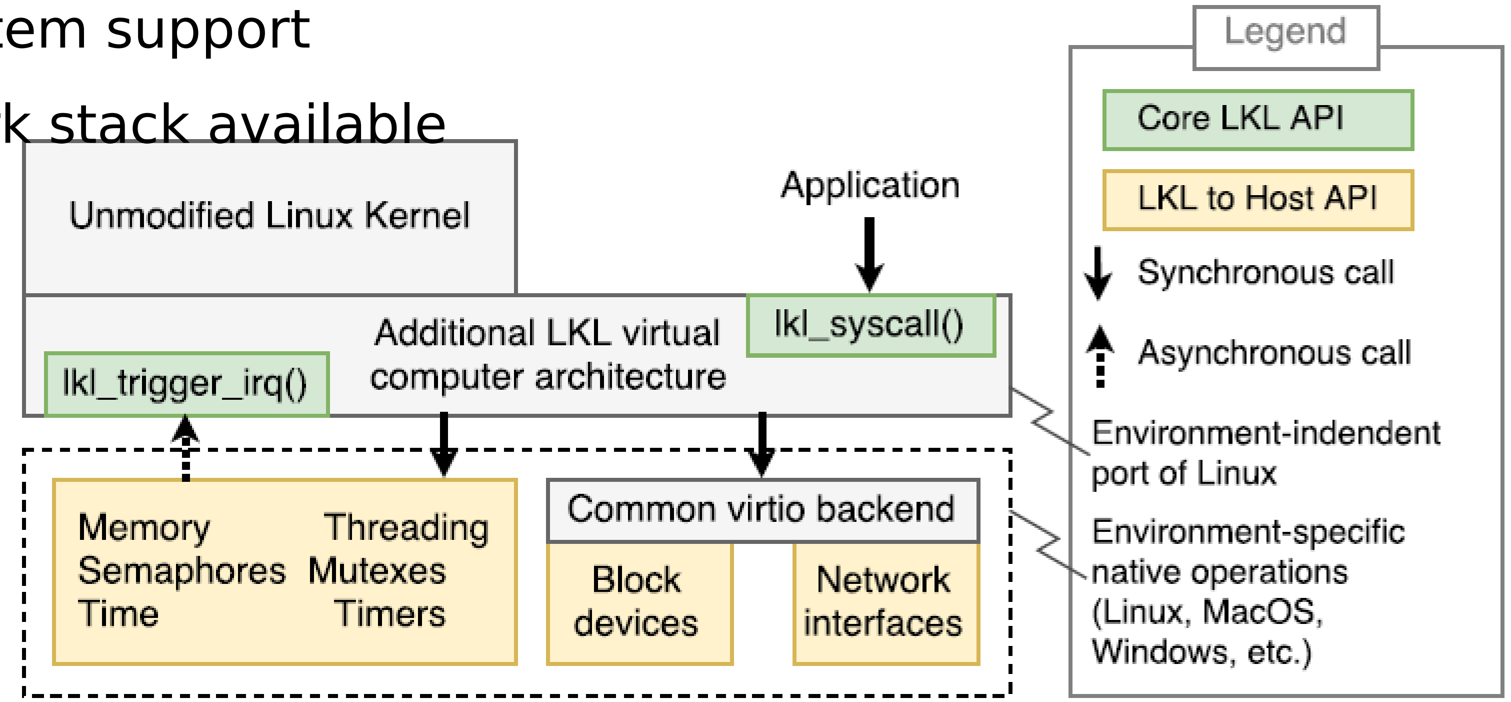
Requires complex system support

- Dynamic library loading
- Filesystem support
- Signal handling
- Complete networking stack

# SGX-LKL: Linux Kernel Library in SGX Enclaves

Based on **Linux Kernel Library (LKL)**

- Implemented as architecture-specific port of mainline Linux (`github.com/lkl`)
- Follows Linux no MMU architecture
- Full filesystem support
- Full network stack available



## SGX-LKL Architecture

Runs **unmodified Linux applications** in SGX enclaves

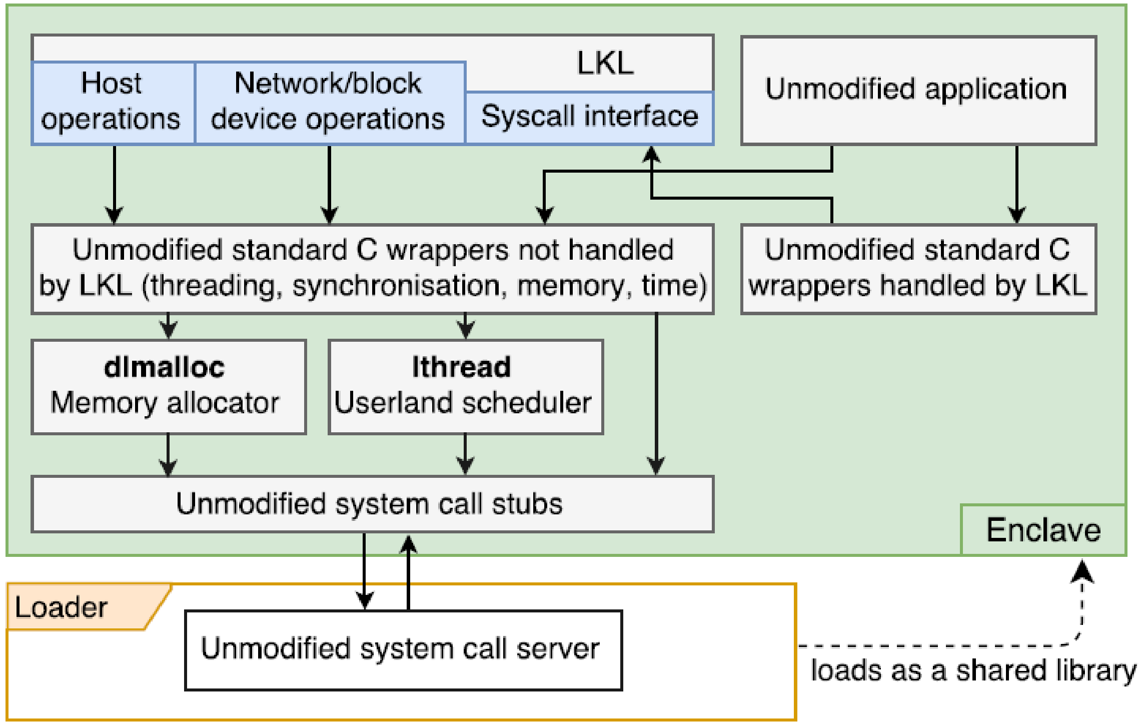
Applications and dependencies provided via **disk image**

**Full Linux kernel functionality** available

**Custom memory allocator**

**User-level threading**

- In-enclave synchronisation primitives


LKL
Host operations
Network/block device operations
Syscall interface
Unmodified application
Unmodified standard C wrappers not handled by LKL (threading, synchronisation, memory, time)
Unmodified standard C wrappers handled by LKL
dlmalloc Memory allocator
lthread Userland scheduler
Unmodified system call stubs
Enclave
Loader
Unmodified system call server
loads as a shared library

## SGX-LKL Architecture: How many instructions in enclaves

| 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%)** |

## 4. Description of SGX-Spark

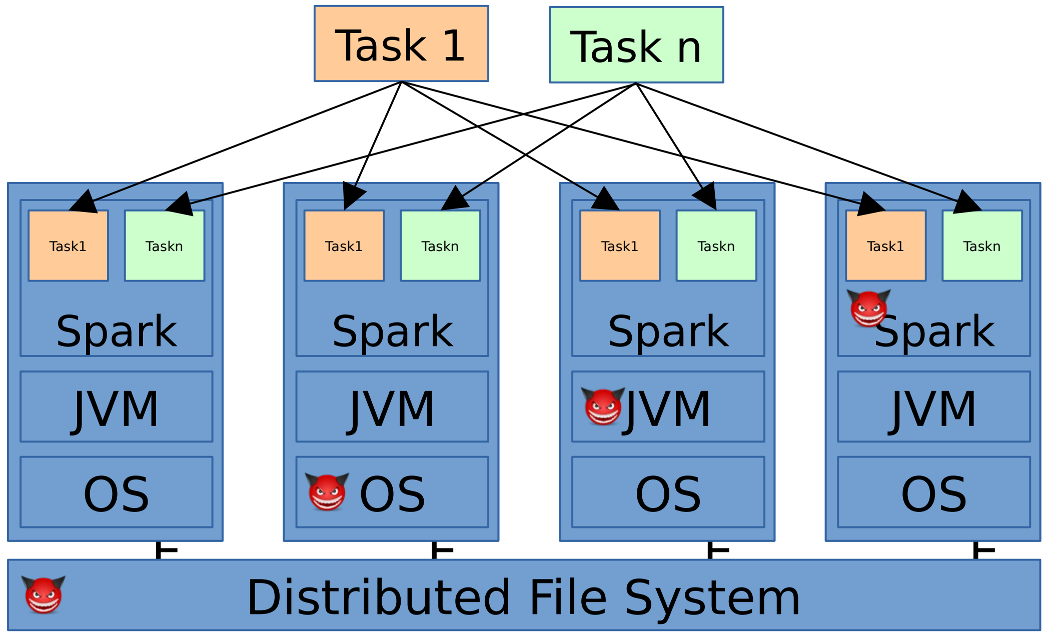
# Secure Big Data Processing

Processing of large amounts of sensitive information

Outsourcing of data storage and processing

Cloud provider can access processed data

- Not acceptable for number of industries



```
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))
}
```

## Apache Spark

Spark is built on the concept of *distributed datasets*, which contain arbitrary Java or Python objects.

You create a dataset from external data, then apply parallel operations to it. The building block of the Spark API is its RDD API. In the RDD API, there are two types of operations: *transformations*, which define a new dataset based on previous ones, and *actions*, which kick off a job to execute on a cluster.

On top of Spark’s RDD API, high level APIs are provided, e.g. DataFrame APIand Machine Learning API. These high level APIs provide a concise way to conduct certain data operations. In this page, we will show examples using RDD API as well as examples using high level APIs.

## Secure Machine Learning

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

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

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

**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

**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

# SGX Support for Spark

## SGX-Spark

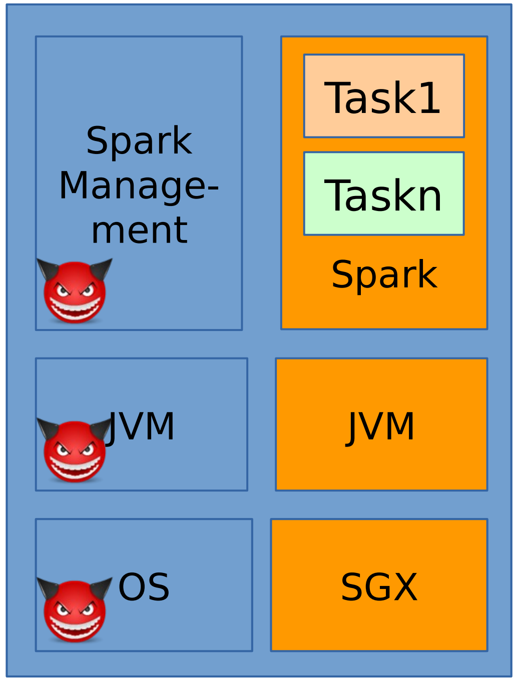
- Protect data processing from infrastructure provider
- Protect confidentiality & integrity of existing jobs
- No modifications for end users
- Acceptable performance overhead

Idea:
Execute only sensitive parts of Spark inside enclave

- Code that accesses/processes sensitive data

Code outside of enclave only accesses encrypted data

- Partition Spark
- Run two collaborating JVMs, inside enclave and outside of enclave

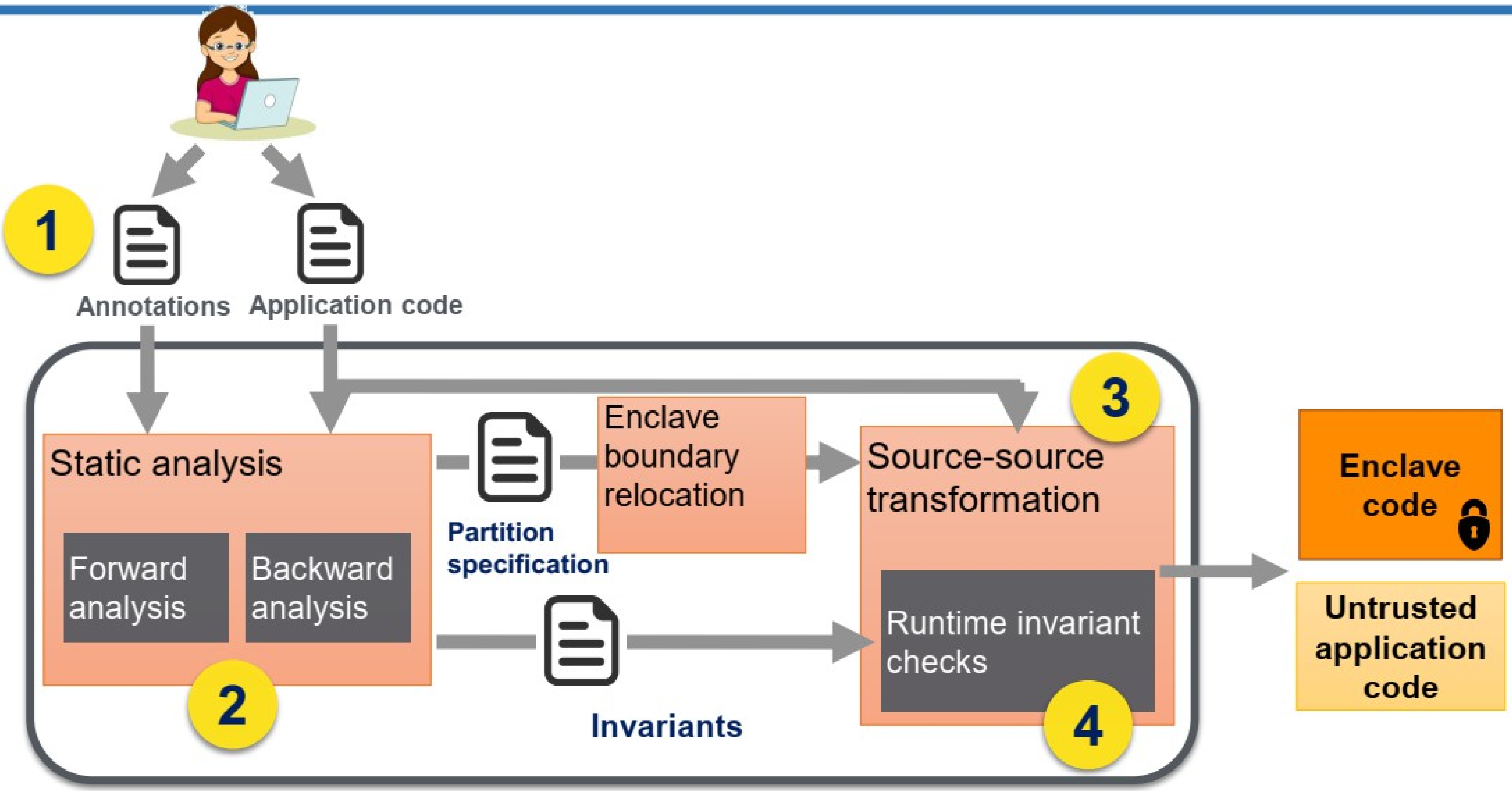


# 1. Partitioning Spark

Goal: Move minimal amount of Spark code to enclave

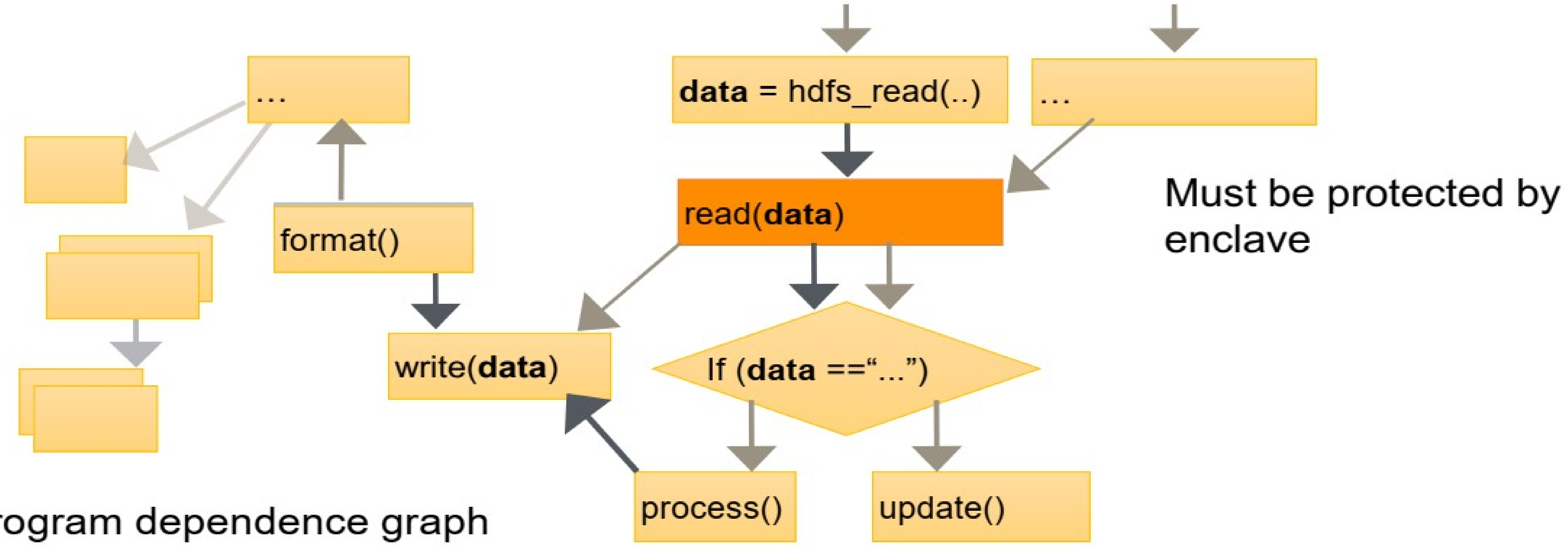
| **Outside** | **Enclave** |
|---|---|
| **HadoopRDD**<br>Provide iterator over input data partition (encrypted) | |
| `it`<br>**MapPartitionsRDD**<br>Execute user-provided function (f)<br>(eg `flatMap(line => {line.split(" ")})`<br>(i) Serialise user-provided function `f`<br>(ii) Send `f` and `it` to enclave JVM<br>(iv) Receive result iterator `it_result`<br>`f,it` →<br>`it2 = it_result` | (iii) Decrypt input data<br>(iv) Compute `f(it) = it_result`<br>(v) Encrypt result |
| **ExternalSorter**<br>Execute user-provided reduce function g<br>(eg `reduceByKey{case (x, y) => x + y}`)<br>`g,it2` →<br>`it2_result` | (iii) Decrypt input data<br>(iv) Compute `g(it2) = it2_result`<br>(v) Encrypt result |
| **ResultTask**<br>Output results | |

# A partitioning framework



Compiler-based framework for partitioning C applications

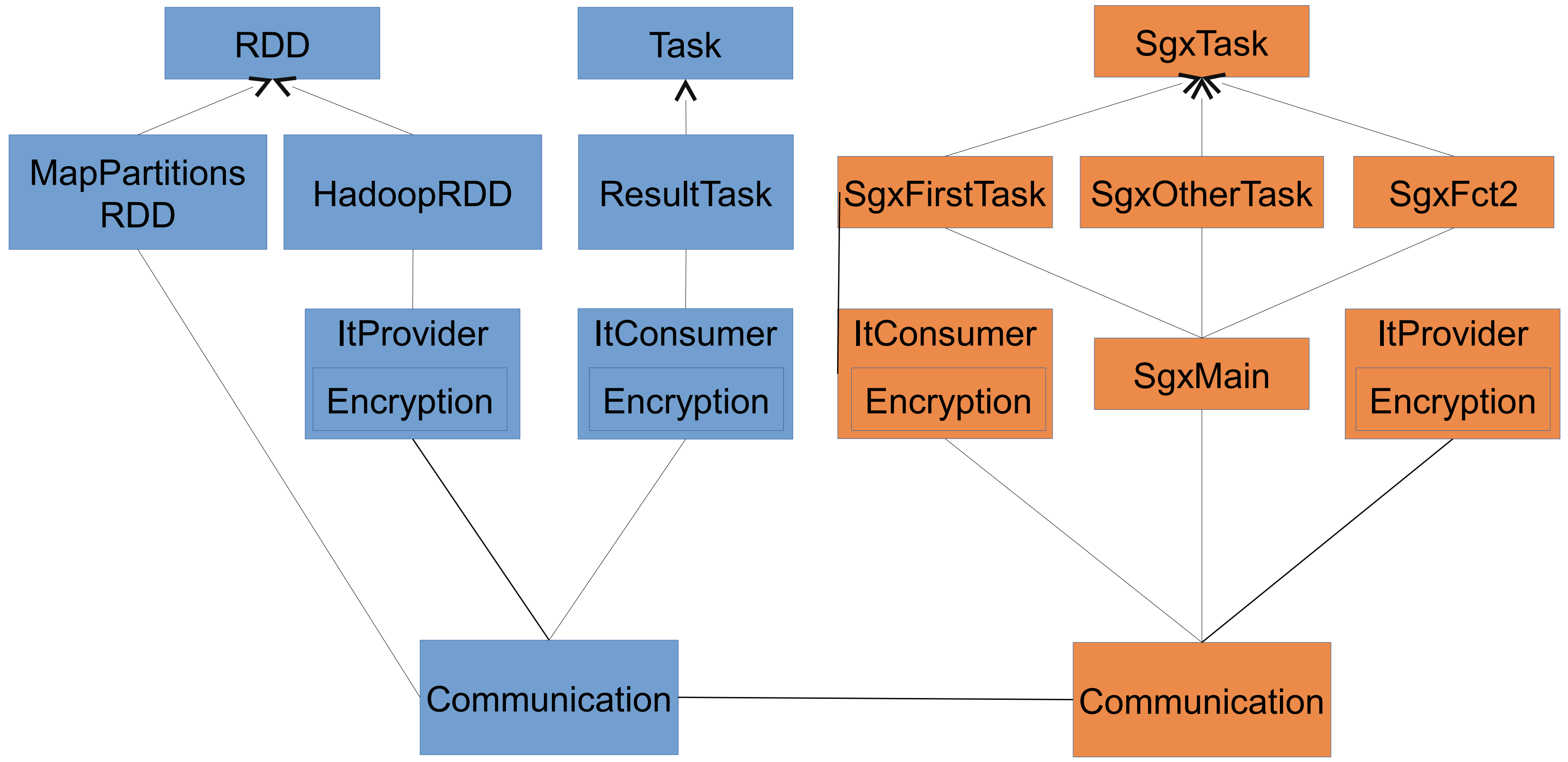
# Data flow analysis for partitioning



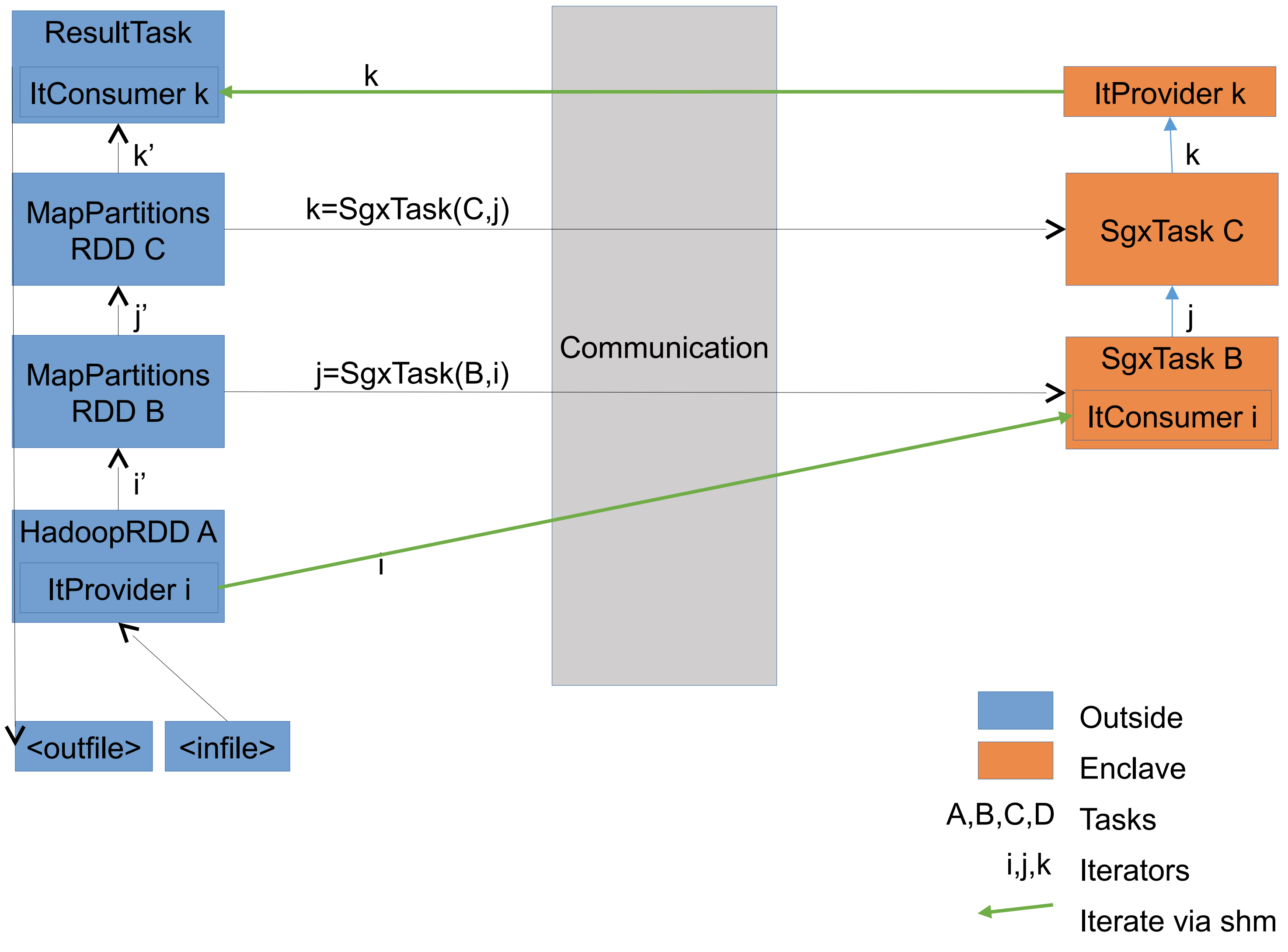
To ensure **data confidentiality:** **forward dataflow analysis**

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

## Partitioning Spark



# Partitioning Spark


ResultTask
ItConsumer k
k
ItProvider k
k'
k
MapPartitions RDD C
k=SgxTask(C,j)
SgxTask C
j'
j
Communication
MapPartitions RDD B
j=SgxTask(B,i)
SgxTask B
ItConsumer i
i'
HadoopRDD A
ItProvider i
i
<outfile>
<infile>
🟦 Outside
🟧 Enclave
A,B,C,D Tasks
i,j,k Iterators
Iterate via shm