Rust SGX SDK: Towards Memory Safety in Intel SGX

Yu Ding, Ran Duan , Long Li , Yueqiang Cheng , Lenx Wei

Baidu X-Lab

CONTENTS

PART 1

Why SGX ?

Why SGX

War in memory

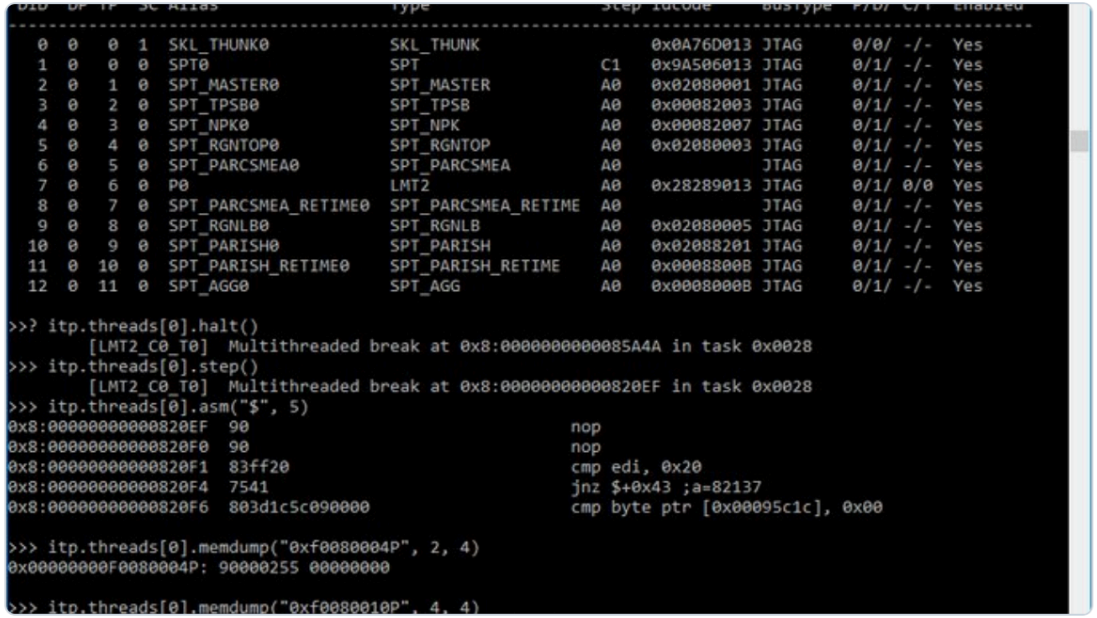
- Ring 3 vs Ring 0
- Ring 0 vs Hypervisor (Ring -1)
- Hypervisor vs SMM (Ring -2)
- SMM vs AMT/ME (Ring -3)

Why SGX

War in memory

Maxim Goryachy
@h0t_max

Game over! We (I and @_markel___) have obtained fully functional JTAG for Intel CSME via USB DCI. #intelme #jtag #inteldci



```
 DID  BP TP  SC Alias                Type                 Step IdCode     BusType  P/D/ C/T  Enabled
-------------------------------------------------------------------------------------------------
  0   0   0   1  SKL_THUNK0           SKL_THUNK                0x0A76D013 JTAG     0/0/ -/-  Yes
  1   0   0   0  SPT0                 SPT                   C1 0x9A506013 JTAG     0/1/ -/-  Yes
  2   0   1   0  SPT_MASTER0          SPT_MASTER            A0 0x02080001 JTAG     0/1/ -/-  Yes
  3   0   2   0  SPT_TPSB0            SPT_TPSB              A0 0x00082003 JTAG     0/1/ -/-  Yes
  4   0   3   0  SPT_NPK0             SPT_NPK               A0 0x00082007 JTAG     0/1/ -/-  Yes
  5   0   4   0  SPT_RGNTOP0          SPT_RGNTOP            A0 0x02080003 JTAG     0/1/ -/-  Yes
  6   0   5   0  SPT_PARCSMEA0        SPT_PARCSMEA          A0            JTAG     0/1/ -/-  Yes
  7   0   6   0  P0                   LMT2                  A0 0x28289013 JTAG     0/1/ 0/0  Yes
  8   0   7   0  SPT_PARCSMEA_RETIME0 SPT_PARCSMEA_RETIME   A0            JTAG     0/1/ -/-  Yes
  9   0   8   0  SPT_RGNLB0           SPT_RGNLB             A0 0x02080005 JTAG     0/1/ -/-  Yes
 10   0   9   0  SPT_PARISH0          SPT_PARISH            A0 0x02088201 JTAG     0/1/ -/-  Yes
 11   0  10   0  SPT_PARISH_RETIME0   SPT_PARISH_RETIME     A0 0x0008800B JTAG     0/1/ -/-  Yes
 12   0  11   0  SPT_AGG0             SPT_AGG               A0 0x0008000B JTAG     0/1/ -/-  Yes

>>? itp.threads[0].halt()
       [LMT2_C0_T0]  Multithreaded break at 0x8:000000000085A4A in task 0x0028
>>> itp.threads[0].step()
       [LMT2_C0_T0]  Multithreaded break at 0x8:00000000000820EF in task 0x0028
>>> itp.threads[0].asm("$", 5)
0x8:00000000000820EF  90                              nop
0x8:00000000000820F0  90                              nop
0x8:00000000000820F1  83ff20                          cmp edi, 0x20
0x8:00000000000820F4  7541                            jnz $+0x43 ;a=82137
0x8:00000000000820F6  803d1c5c090000                  cmp byte ptr [0x00095c1c], 0x00

>>> itp.threads[0].memdump("0xf0080004P", 2, 4)
0x00000000F0080004P: 90000255 00000000

>>> itp.threads[0].memdump("0xf0080010P", 4, 4)
```

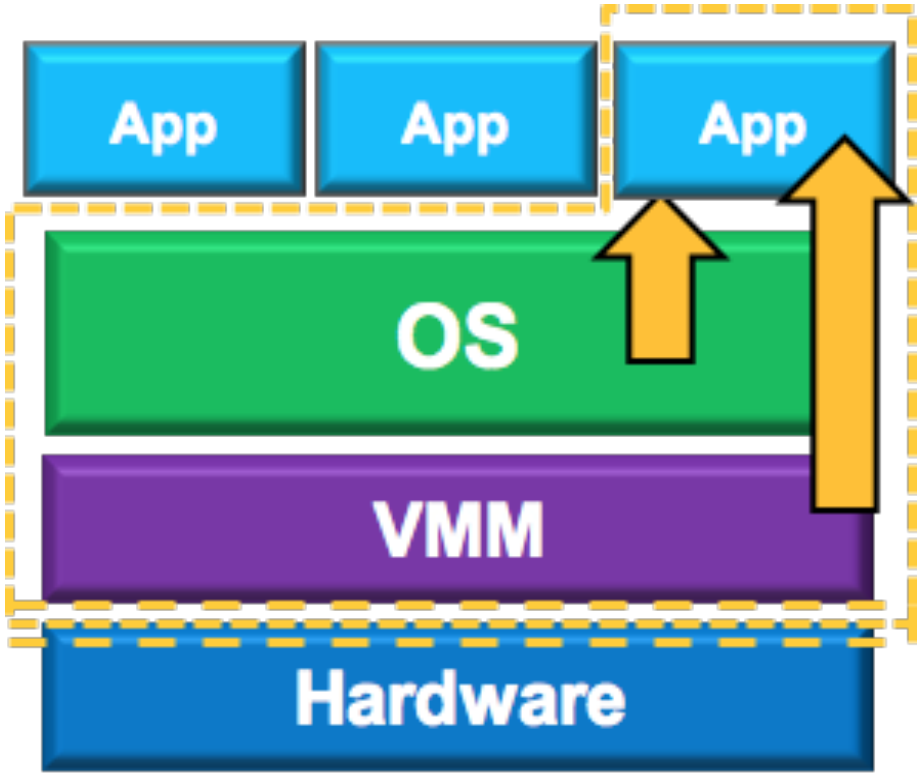
上午6:33 · 2017年11月8日

Why SGX

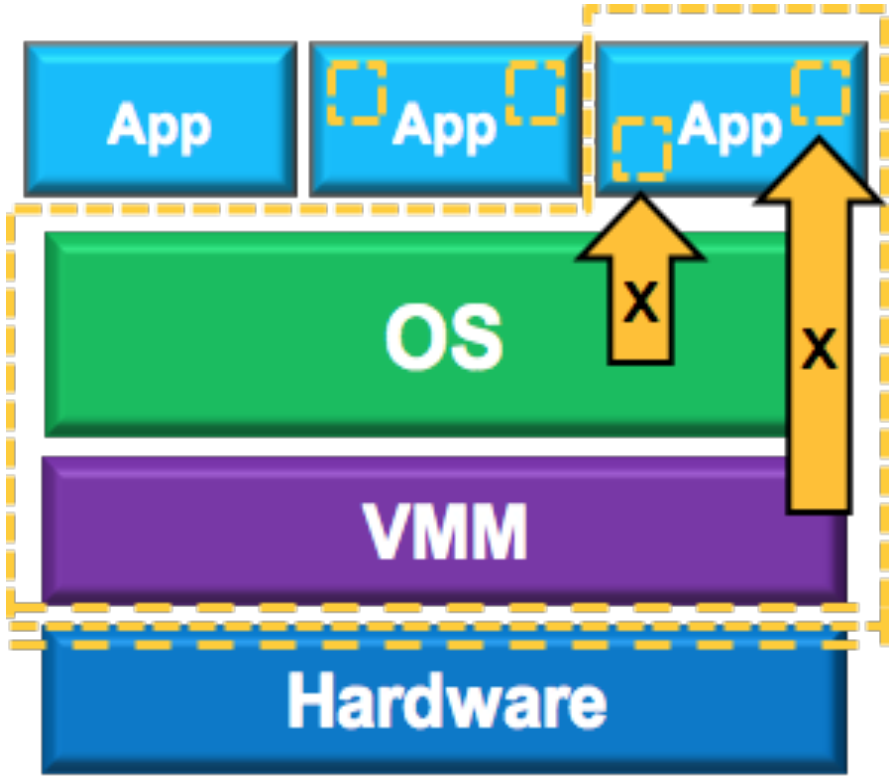
Hardware based trusted execution environment

- Intel System Management Mode
- Intel Management Engine
- Trusted Platform Module (TPM)
- AMD Platform Security Processor
- DRTM (Dynamic Root of Trust for Measurement)
- ARM Trustzone
- Intel Trusted Execution Technology
- **Intel SGX**

Why SGX：Memory Encryption Engine



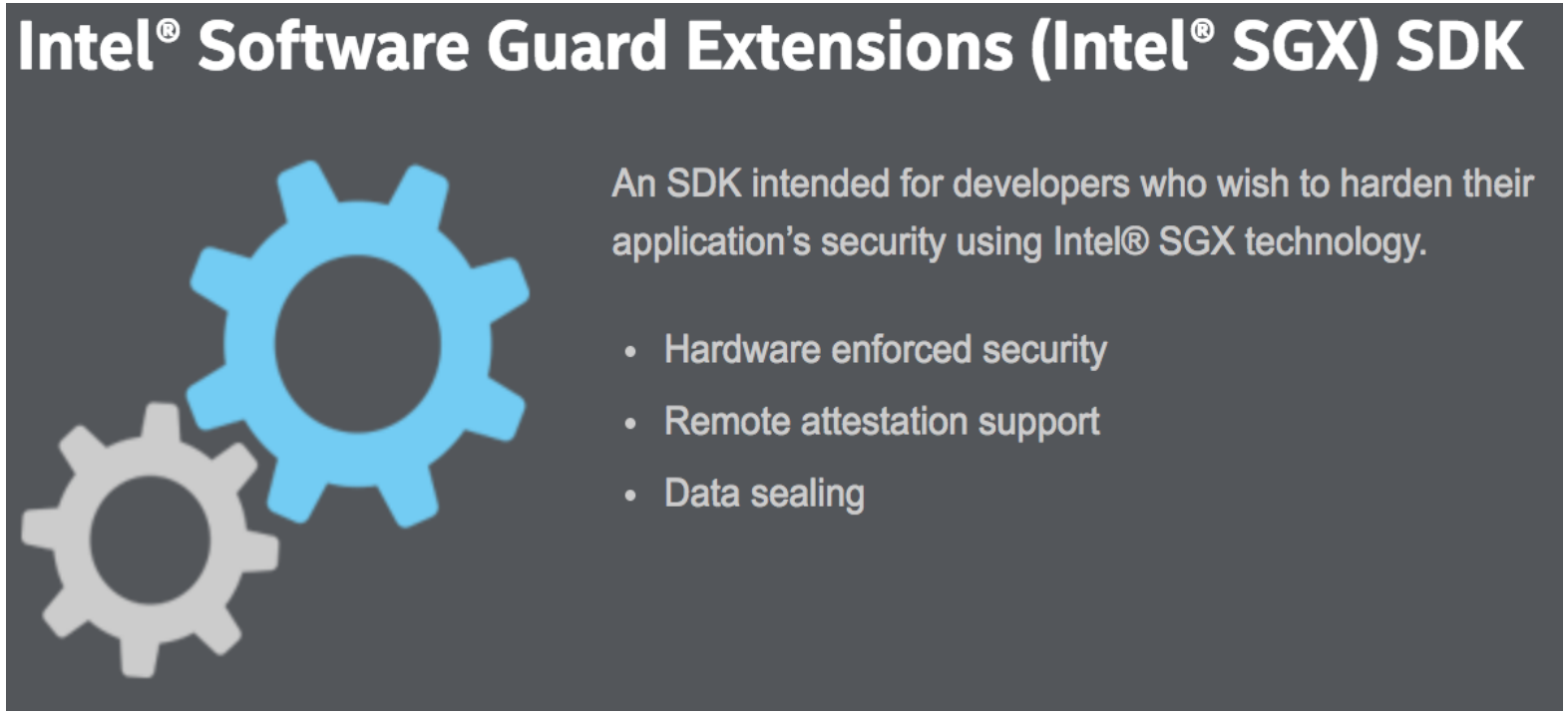
Without SGX



SGX Enforced

Figures are from Intel ISCA'15 SGX Turtorial

Why SGX：Root of Trust



- **Hardware Enforced Security: MEE**
- **Remote Attestation Support: Build trust with Intel**
- **Data Sealing: Transfer/store data**

PART 2

Why Rust ?

Why Rust：Rust Programming Language

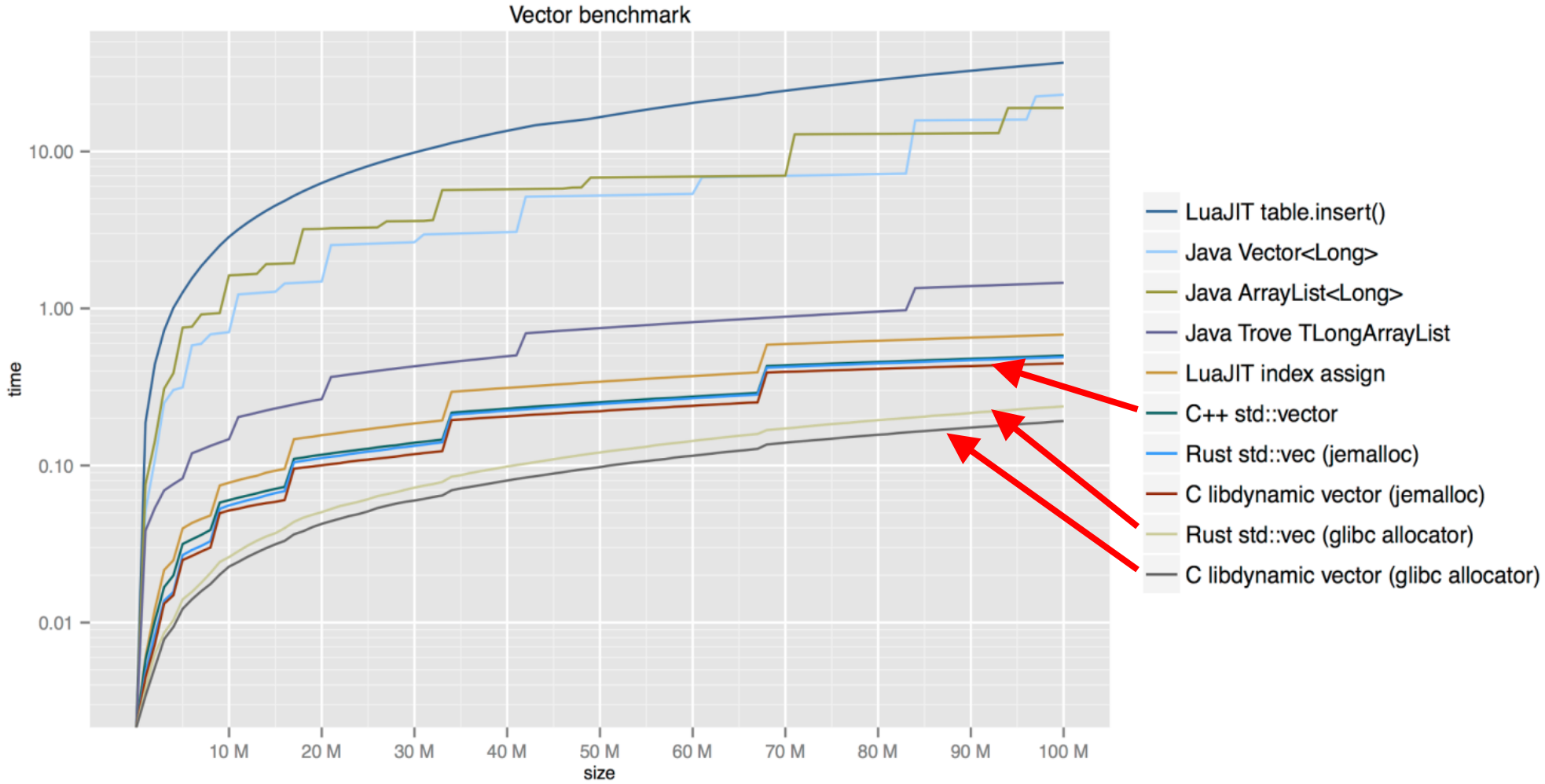
Endorsed by Mozilla, competing with Go and Swift

- Guarantees memory safety
- No data racing
- Blazingly fast

Masterpieces in Rust

- Redox: A Rust Operating System https://www.redox-os.org
- The Servo Browser Engine https://servo.org

Why Rust：Excellent Performance



Why Rust : Strong Checkers

- **Borrow、Ownership、Lifetime**

```
fn main() {
    let a = String::from("book"); // a owns "book"
    let b = a;                    // transfer ownership
    println!("a = {}", a);        // Error! a is not owner
}
```

  - **“One writer, or multiple reader” guaranteed by Rust**
  - **Keep each variable's ownership、lifetime in mind**

    —Fight against borrow checker

PART 3

Rust SGX SDK

SGX Needs Memory Safety Guarantees

Intel SGX is designed to protect secret data

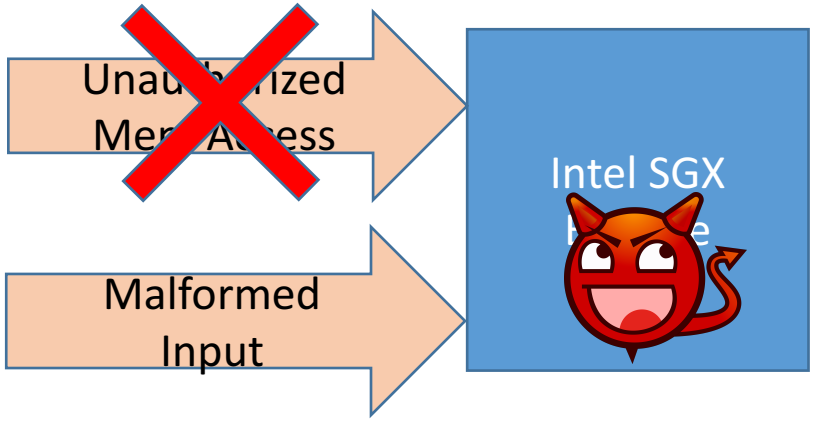
- Private keys
- User privacy (health data/personal data etc.)
- Raw Blu-ray video stream
- DRM enforcement

But, only **C/C++** SDK is available.

Should be very very very careful when writing SGX enclaves in **C/C++**

- Buffer overflow … Yes!
- Return-oriented-programming … Yes!
- Use-after-free … Yes!
- Data racing … Yes!

Memory bugs are exploitable!



SGX Needs Memory Safety Guarantees

Code in Trusted Execution Engine may be **vulnerable**

- Memory corruption vulnerability is exploitable
- Code needs to be audited

To better protect secrets in SGX, we need **memory safety**

- Provide best security guarantees
- Provide latest SGX APIs by Intel

Our Solution : Intel SGX + Rust Programming Language

- Use Intel SGX for data protection
- Develop Intel SGX enclaves in Rust
- Develop Intel SGX untrusted components in Rust *
- More details in https://github.com/baidu/rust-sgx-sdk

Hybrid Memory-Safe Architecture: Rules-of-thumb

Goals

- Memory safety guarantees
- Good functionality

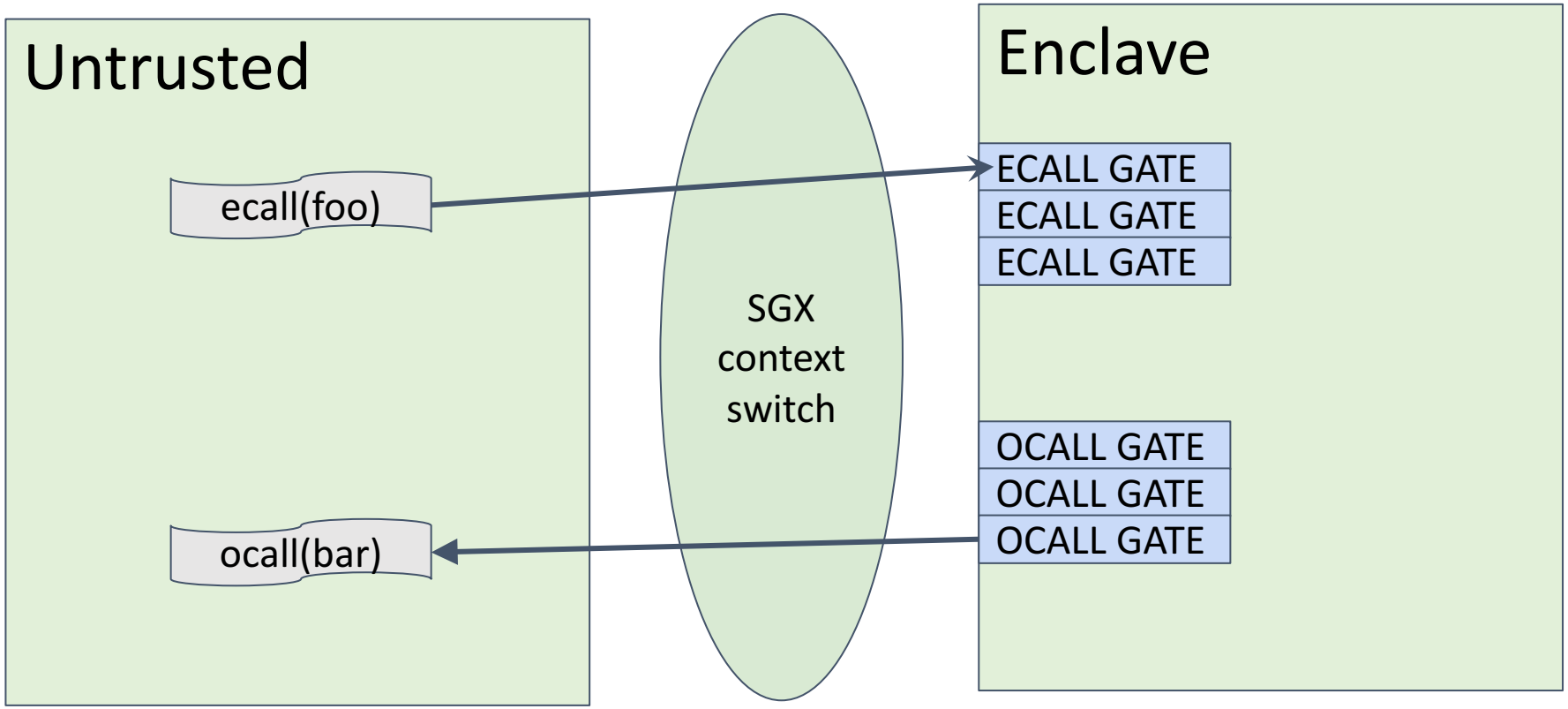
Challenges

- Intel SGX library is written in C/C++

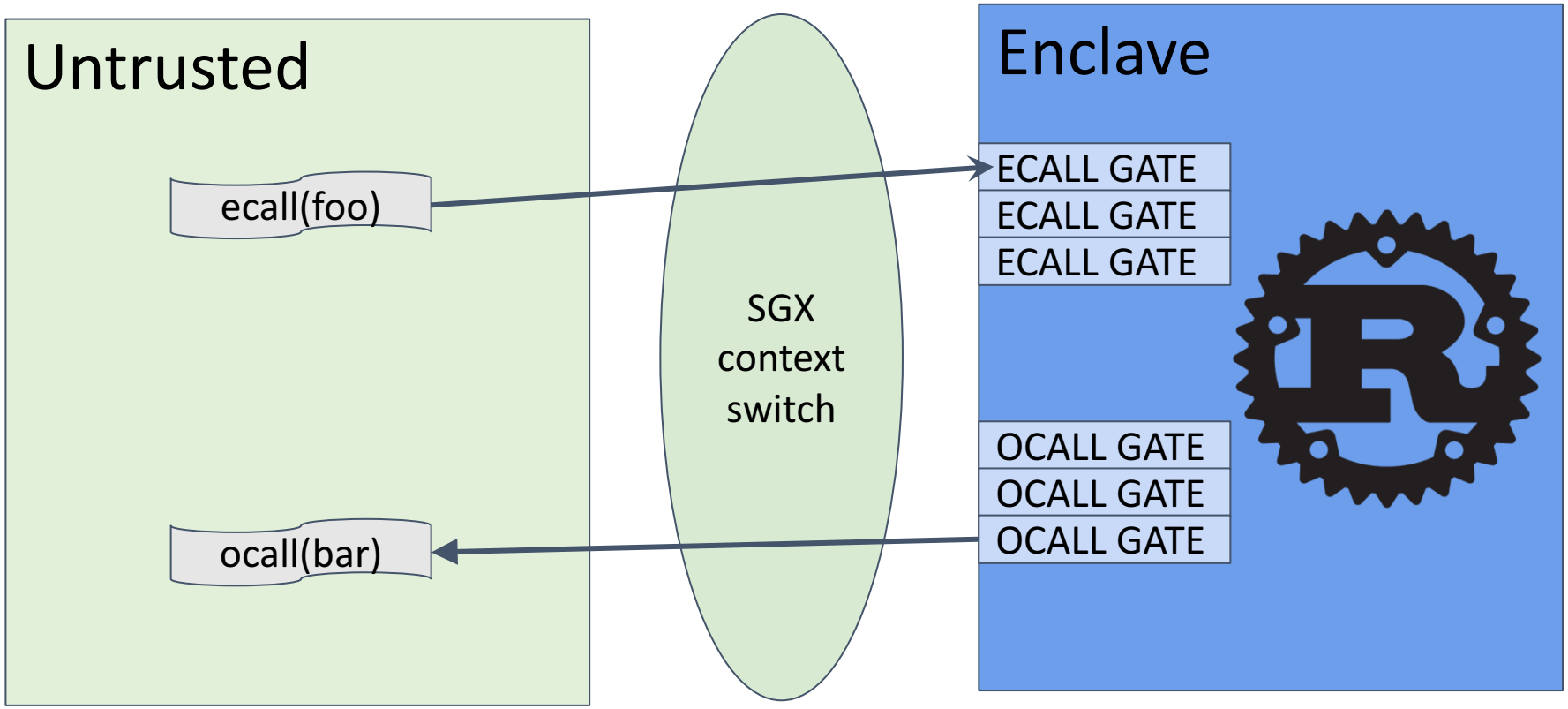
Memory safety rule-of-thumb for **hybrid memory-safe architecture** designing

1. Unsafe components should be appropriately isolated and modularized, and the size should be small (or minimized).
2. Unsafe components should not weaken the safe, especially, public APIs and data structures.
3. Unsafe components should be clearly identified and easily upgraded.

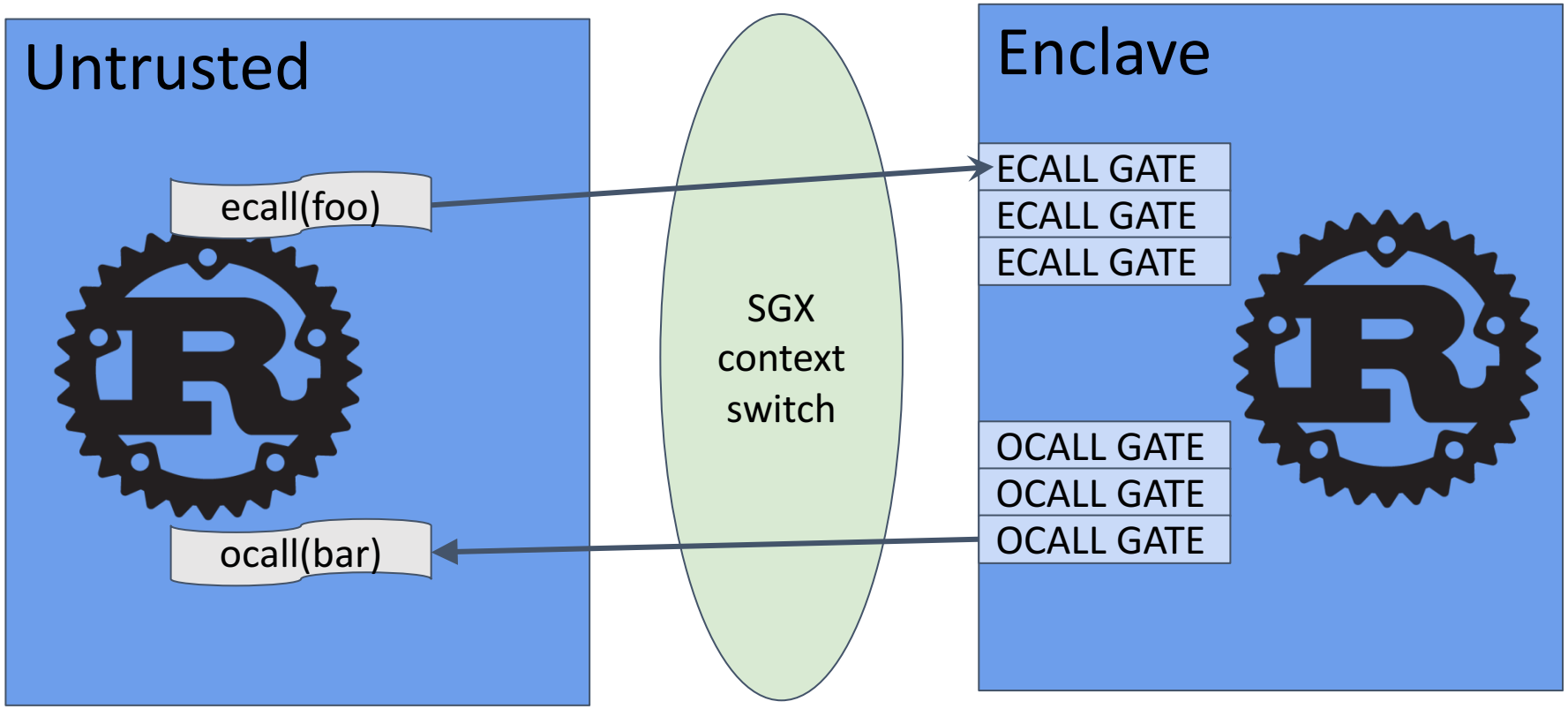
Overview without Rust SGX SDK



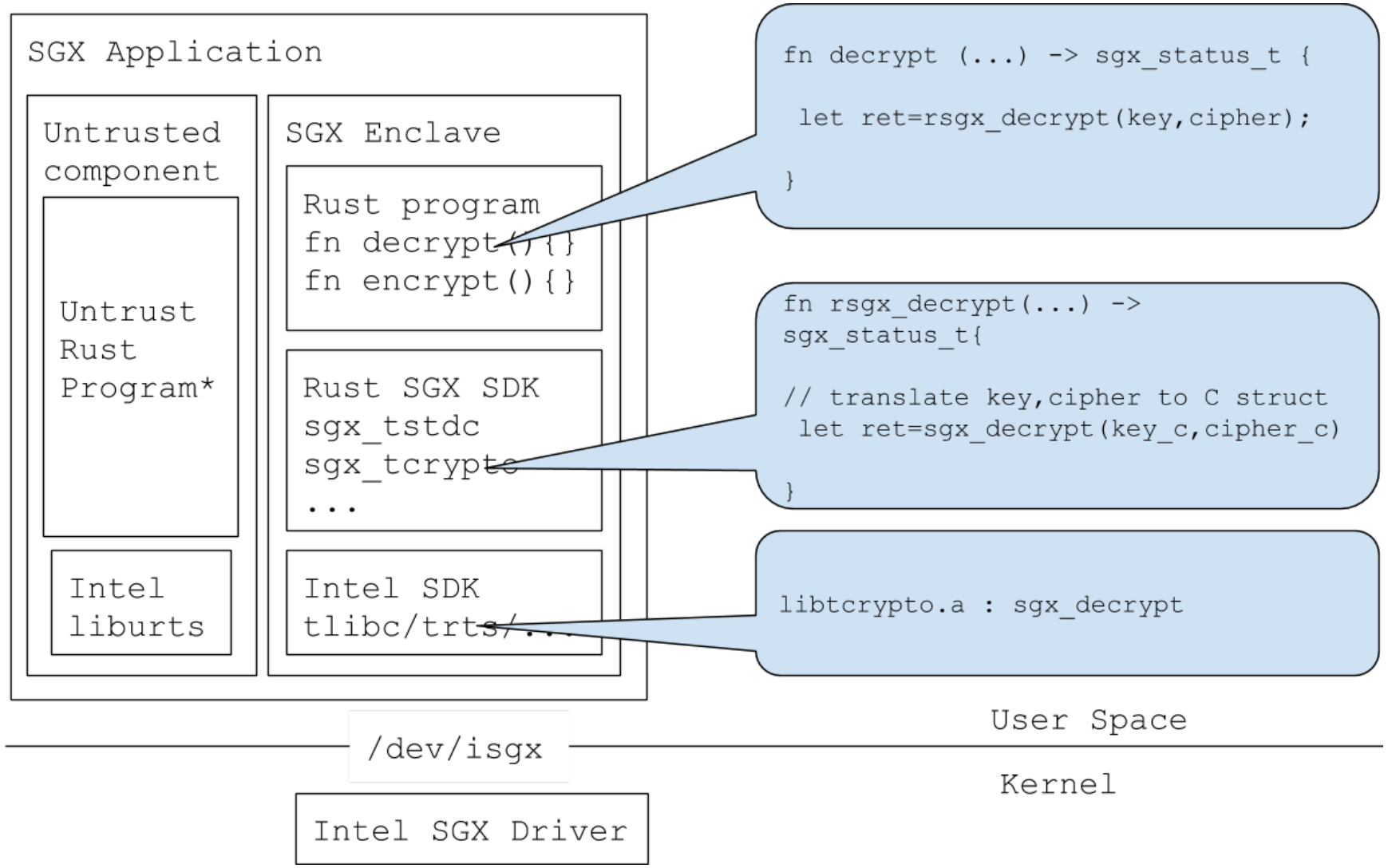
Rust SGX SDK : v0.1.0, v0.2.0



Rust SGX SDK : v0.9.0



Rust SGX SDK：An Overview



Rust SGX SDK : Hello the world

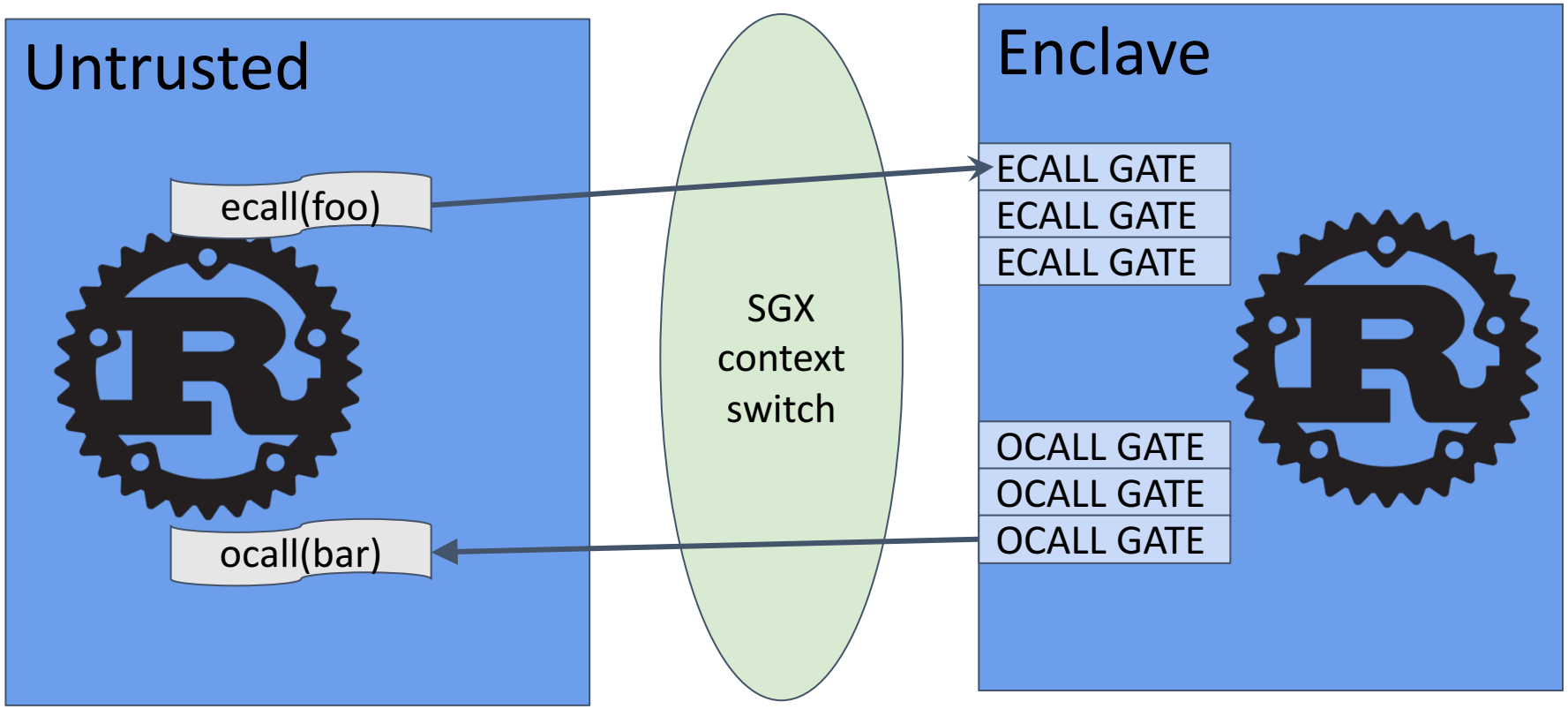
- **Untrusted part**

```
let mut retval = sgx_status_t::SGX_SUCCESS;
let result = unsafe {
    say_something(enclave.geteid(),
                  &mut retval);
};
```

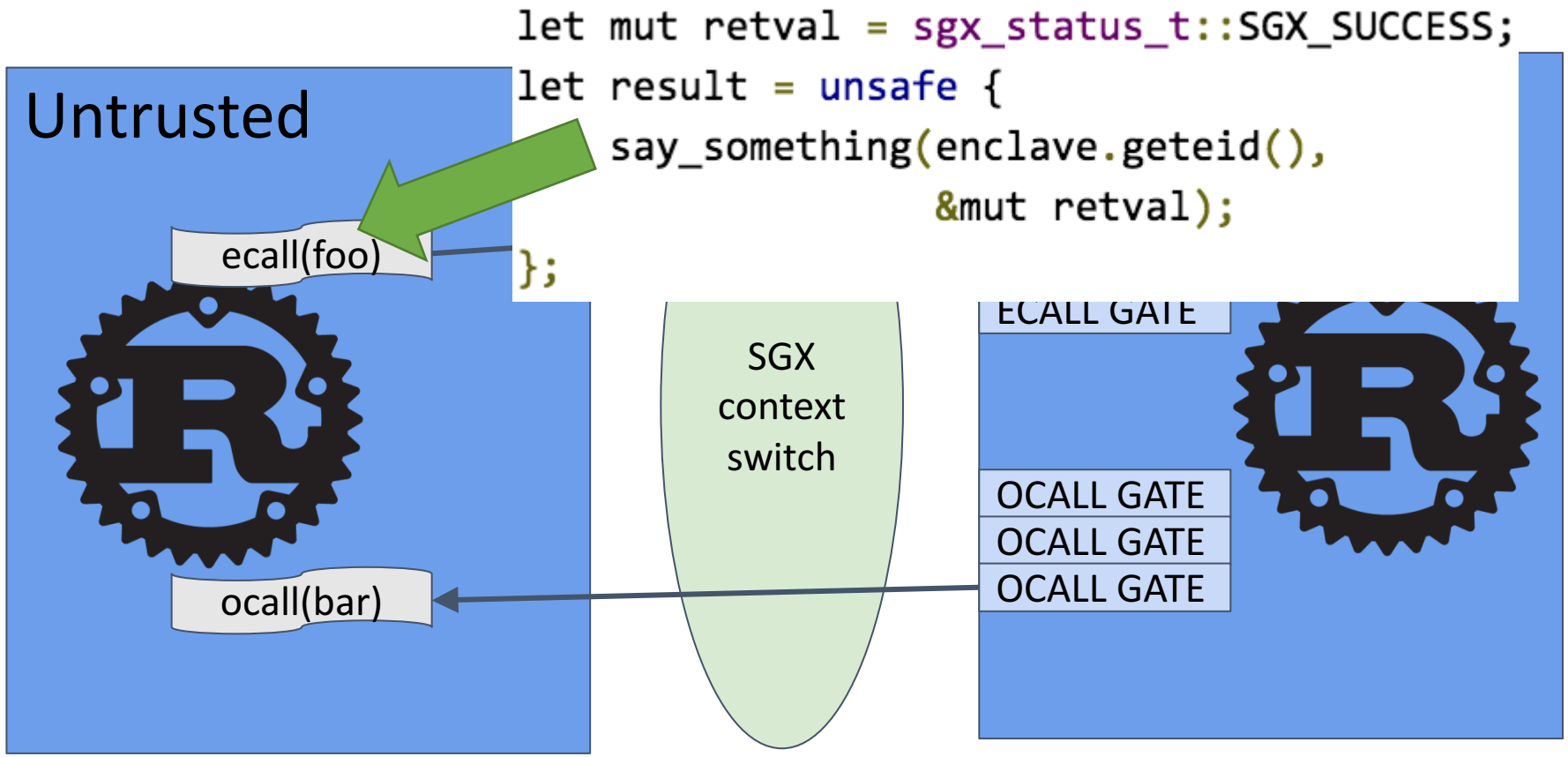
- **Enclave**

```
#[no_mangle]
pub extern "C"
fn say_something () -> sgx_status_t {
        println!("Hello from the SGX enclave!");
        sgx_status_t::SGX_SUCCESS
}
```

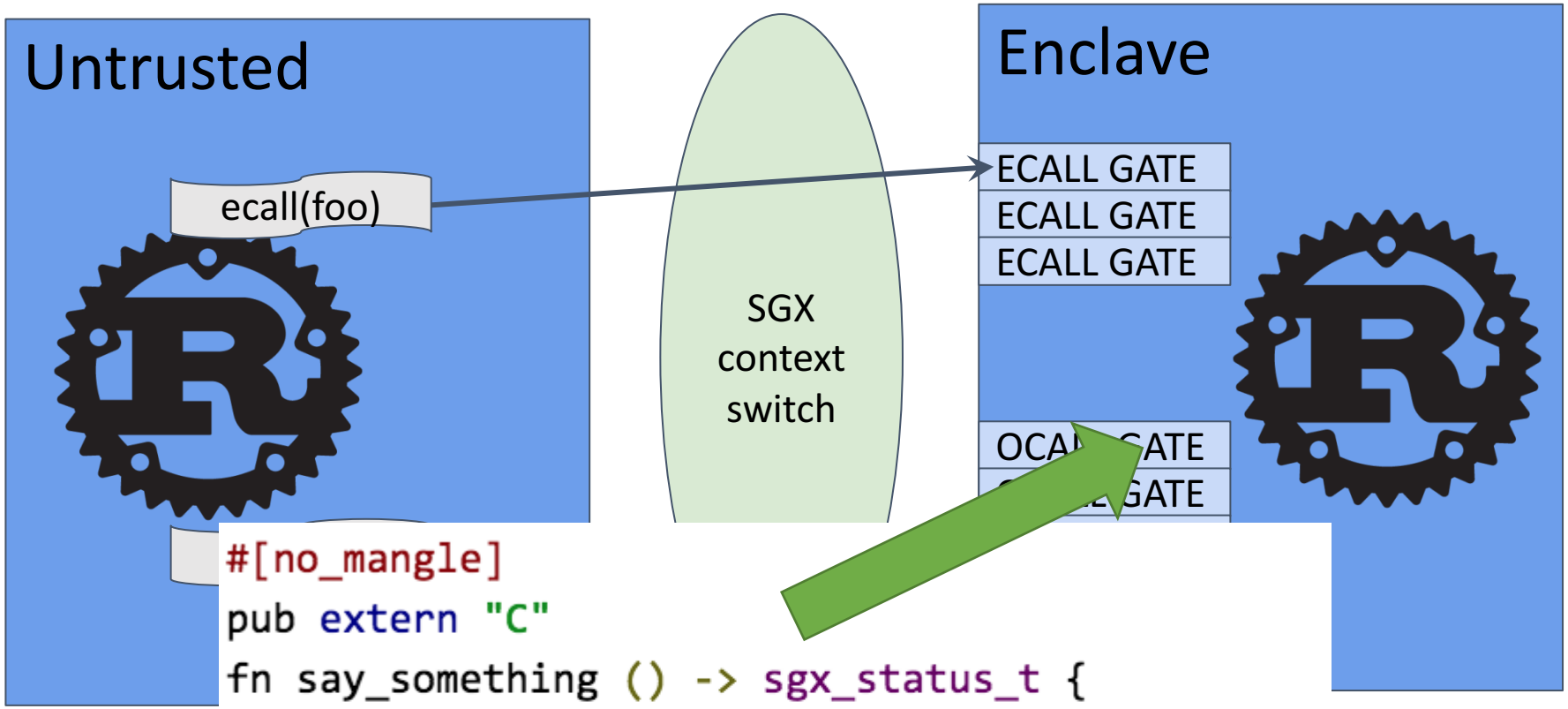
Rust SGX SDK : v0.9.0



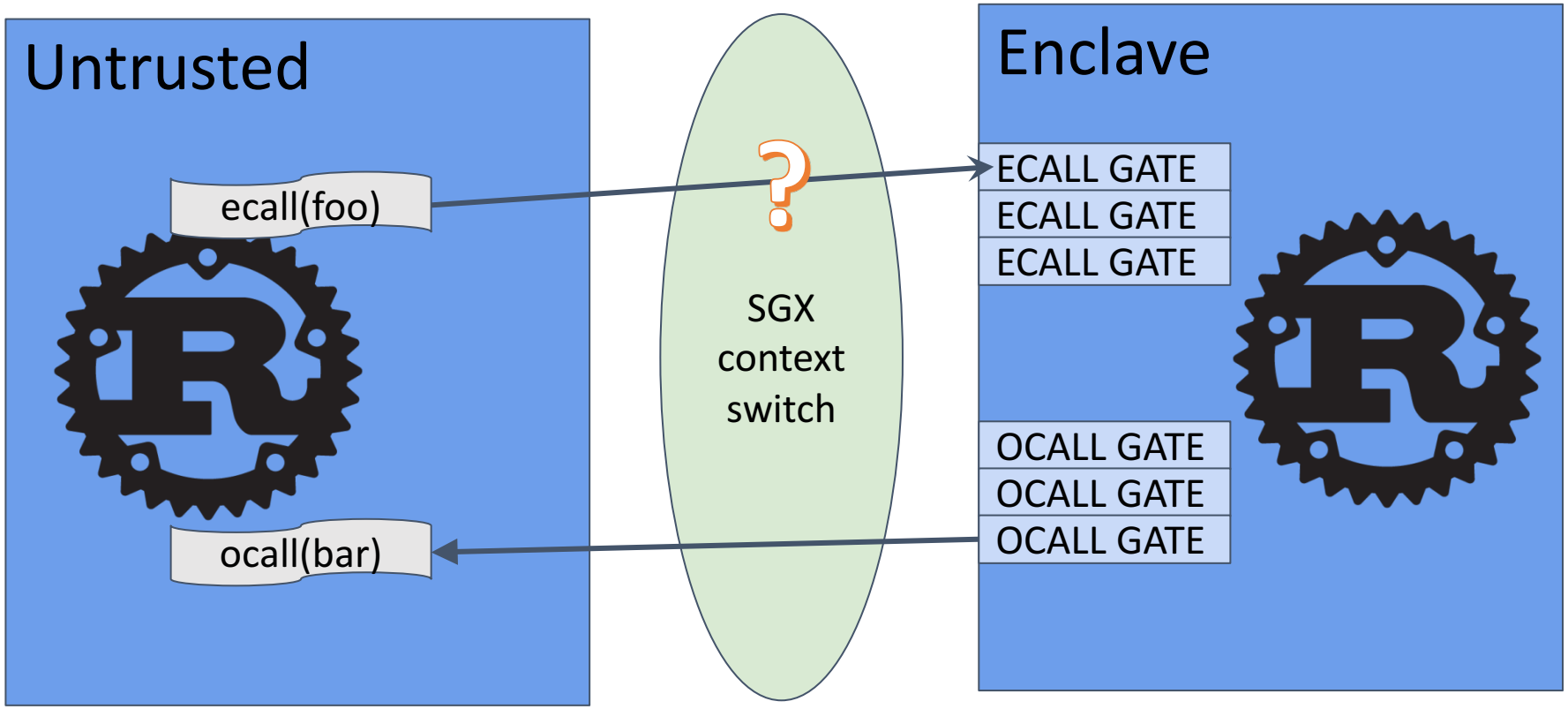
Rust SGX SDK : v0.9.0



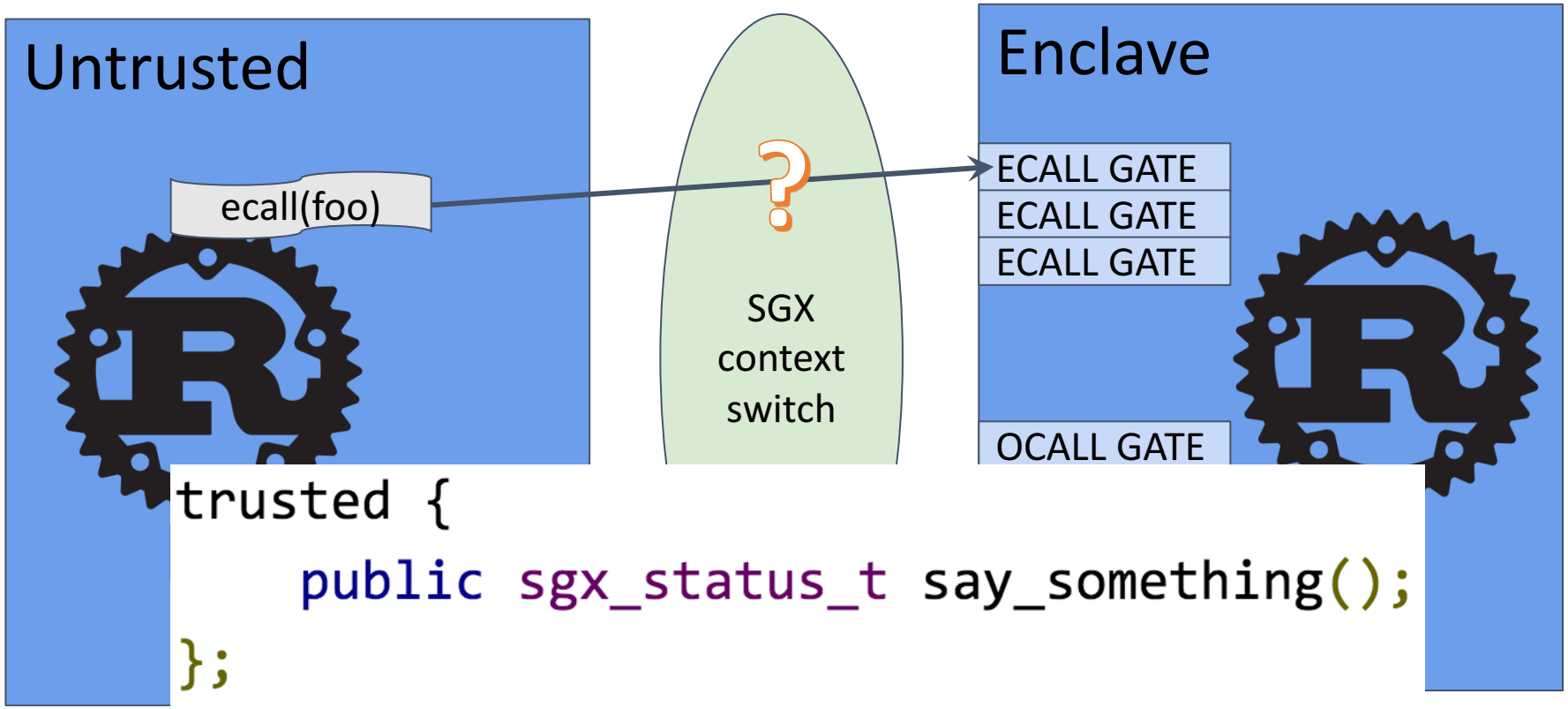
Rust SGX SDK : v0.9.0



Rust SGX SDK : v0.9.0



Rust SGX SDK : v0.9.0



EDL File

Rust SGX SDK：Partition

Question: Which part of a program should be inside SGX enclave?

- **Decryption/Encryption using private key**
- **Seal data/Unseal data**
- **Analysis on secret data**
- **…**

However, most SGX developers are not SGX experts, not experienced in partition an SGX app.

Good and NG Examples
node-secureworker, wolfSSL SGX Samples

Node-secureworker [GOOD]

- In-enclave DukTape Javascript engine
- Remote Attestation on bootstrap
- Seal all outputs

WolfSSL SGX Sample [NG]

- In-encla
- Pass in-

Tamper the ctx pointer may:
1) misguide app
2) cause DOS

```
WOLFSSL* enc_wolfSSL_new([user_ch   k]
                         WOLFSSL_CTX* ctx);
```

Rust SGX SDK : Partition by SDK

Our Goals

- Partition basic libraries correctly
- Provide an easy-to-use interface
- Let developers feel easy in programming Intel SGX enclaves

Rust SGX SDK : Short summary

1. The Memory safety is necessary to Intel SGX enclaves.

2. Rust SGX SDK is valuable and promising
   - Allows to programming Intel SGX Enclaves in Rust.
   - Intends to build up a hybrid memory-safe architecture with Rust and Intel SGX libraries.
   - Provides a series of crates (libraries), such as Rust-style std, alloc etc, and Intel-SGX-style crypto, seal, protected_fs etc.
   - Partitions the basic libraries correctly.

Challenges

What we do?

Intel SGX : Limitations

Dynamic loading? NO!	Static linking!
System call? NO!	We need partition!
Threading model? Different!	Redefine thread/sync!
Exception/Signal? New!	Reimplement exception/signal!

CPUID instruction? NO in SGXv1

RDTSC instruction? NO in SGXv1

Rust SGX SDK : Dependency

Rust binaries depends on libc by default (linux-x86_64, dynamic loading)

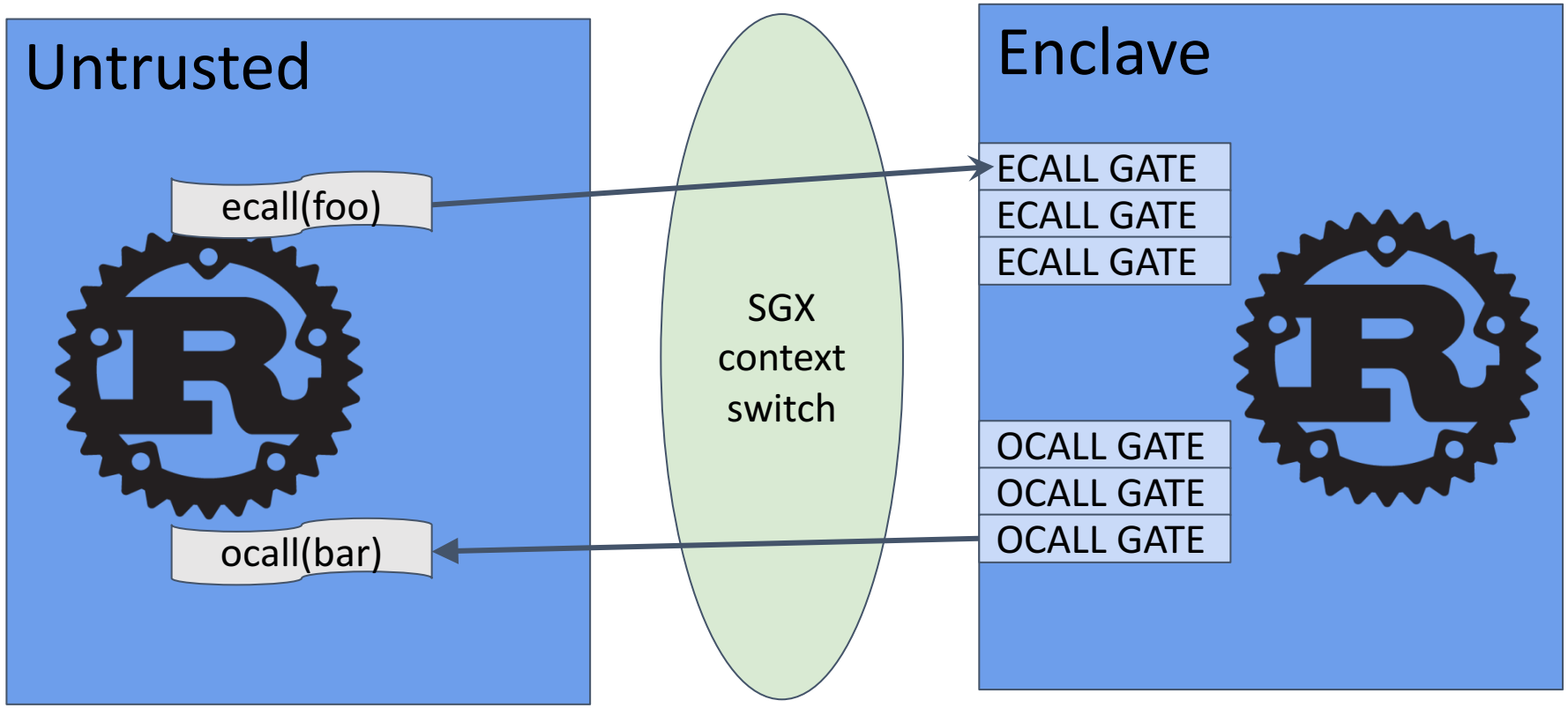
```
Dynamic section at offset 0x61af8 contains 29 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libdl.so.2]
 0x0000000000000001 (NEEDED)             Shared library: [librt.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libpthread.so.0]
 0x0000000000000001 (NEEDED)             Shared library: [libgcc_s.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x0000000000000001 (NEEDED)             Shared library: [ld-linux-x86-64.so.2]
```

Intel provides static trusted libc (tlibc.a) for Intel SGX enclave

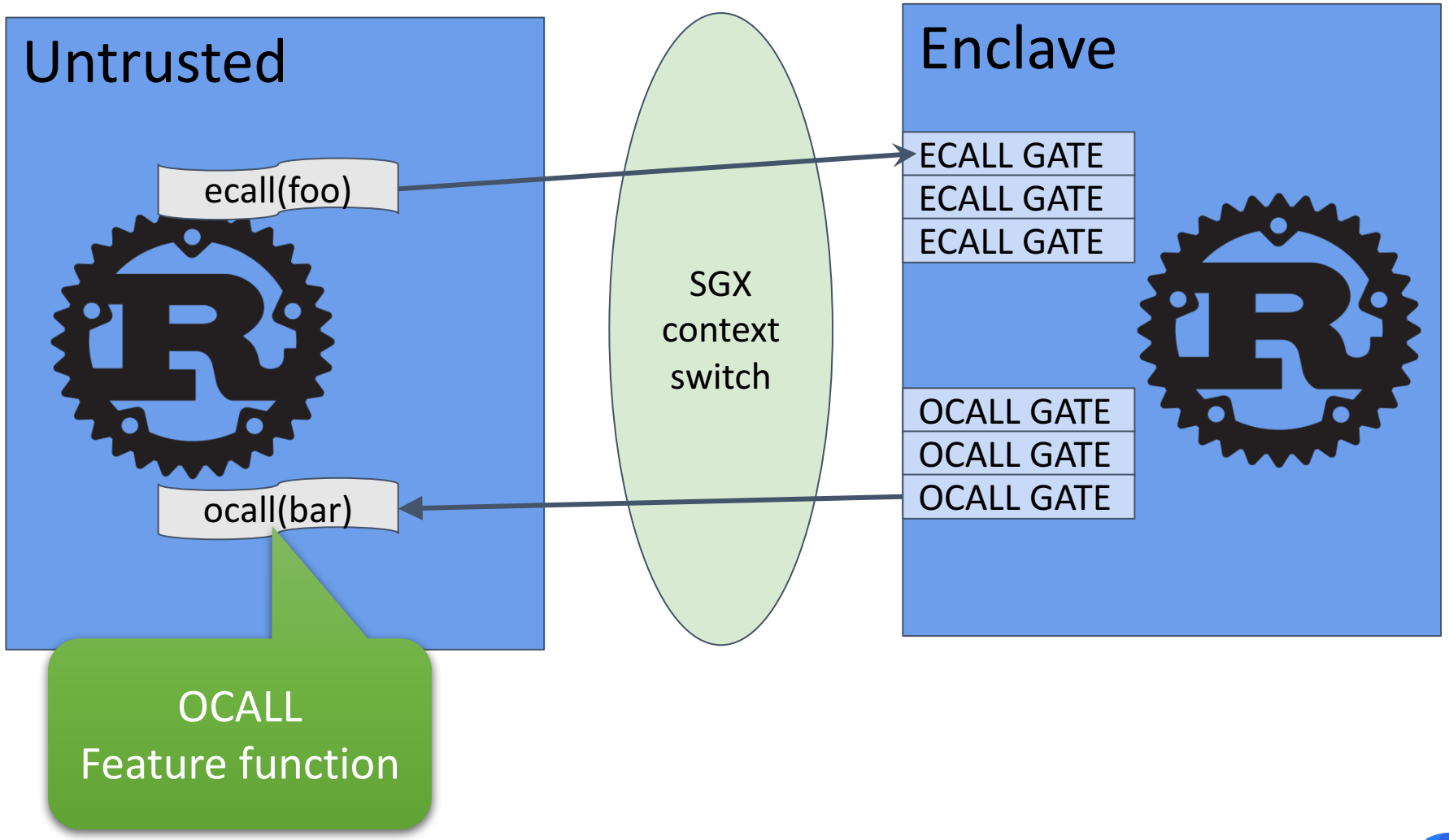
- **SGX features are provided in other static libraries**

Rust SGX SDK statically link to Intel SGX libraries

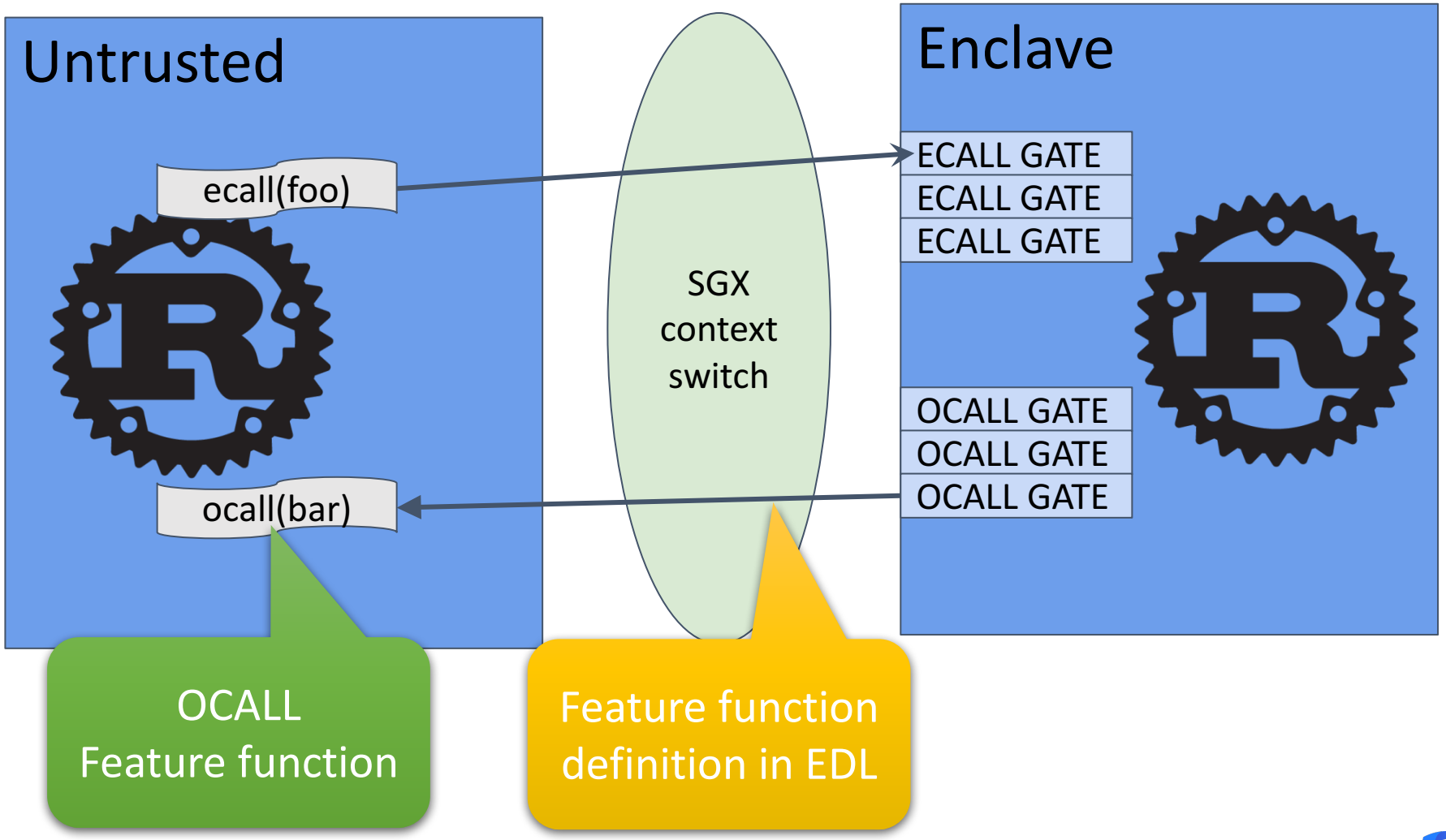
Rust SGX SDK : Partition and Interacting with OS



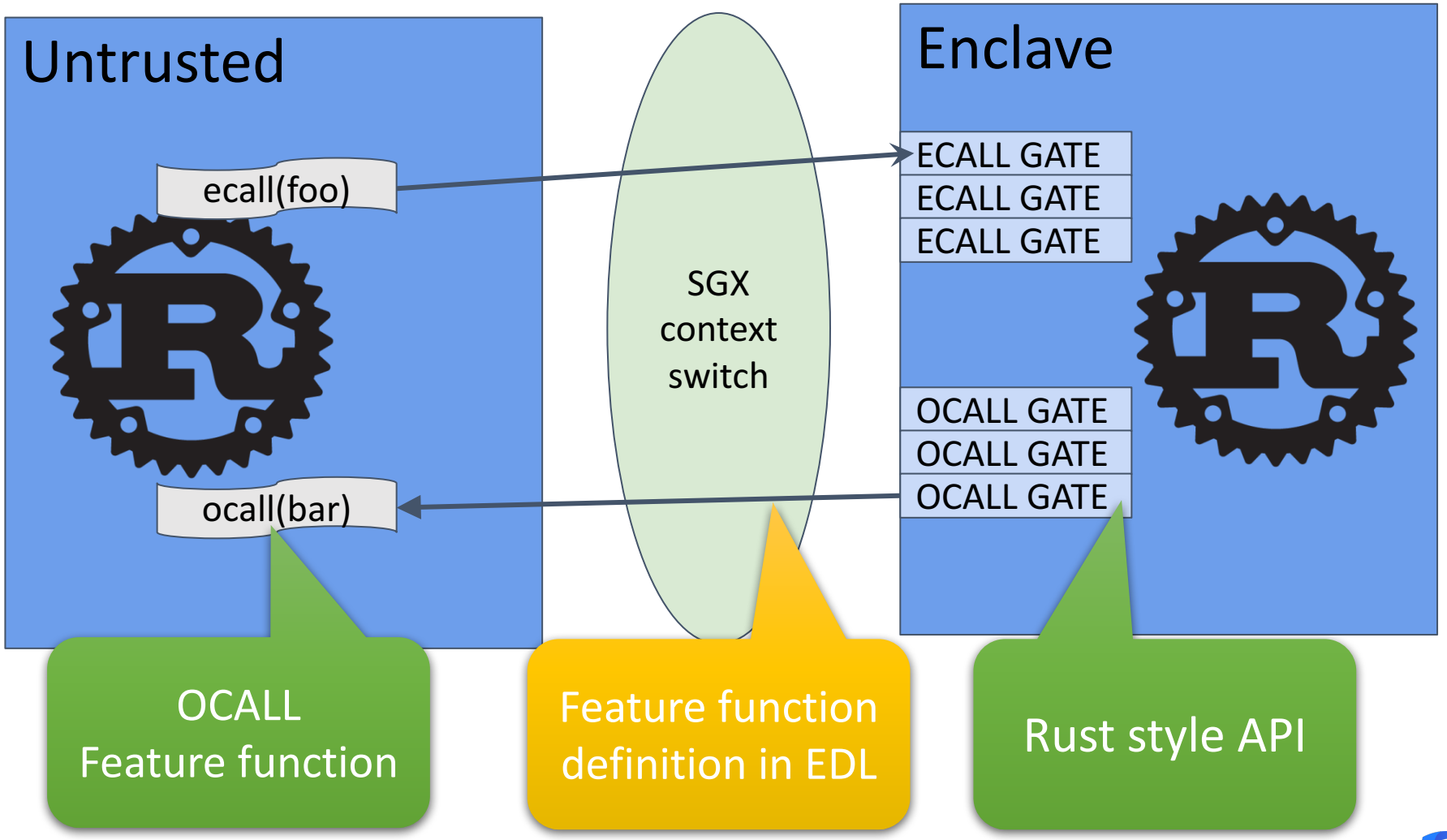
Rust SGX SDK : Partition and Interacting with OS



Rust SGX SDK : Partition and Interacting with OS



Rust SGX SDK : Partition and Interaction with OS



Rust SGX SDK : Partition and Interaction with OS

In enclave source

- `println!("Hello QConf!");`

In sgx_tstd, macro are expanded and invoke io API:

- `println! => print! => sgx_tstd::io::_print()`

sgx_tstd::io maintains a global Stdout object and makes it a LineWriter

- `fn stdout_init() -> Arc<SgxReentrantMutex<RefCell<LineWriter<Maybe<StdoutRaw>>>>>`

`StdoutRaw` is a wrapper structure of `sgx_tstd::sys::Stdout`

```
impl Stdout {
    pub fn write(&self, data: &[u8]) -> io::Result<usize> {
        …
        u_stdout_ocall(&mut result as * mut isize as * mut usize,
        data.as_ptr() as * const c_void,
        cmp::min(data.len(), max_len()))};
```

Rust SGX SDK : Partition and Interaction with OS

In enclave source

- `println!("Hello QConf!");`

In sgx_tstd, macro are expanded and invoke io API:

- `println! => print! => sgx_tstd::io::_print()`

sgx_tstd::io maintains a global Stdout object and makes it a LineWriter

- `fn stdout_init() ->`
  `Arc<SgxReentrantMutex<RefCell<LineWriter<Maybe<StdoutRaw>>>>>`

`StdoutRaw` is a wrapper structure of `sgx_tstd::sys::Stdout`

```
impl Stdout {
    pub fn write(&self, data: &[u8]) -> io::Result<usize> {
        …
        u_stdout_ocall(&mut result as * mut isize as * mut usize,
        data.as_ptr() as * const c_void,
        cmp::min(data.len(), max_len()))};
```

Defined in EDL file

Rust SGX SDK : Partition and Interaction with OS

Feature function definition in EDL file

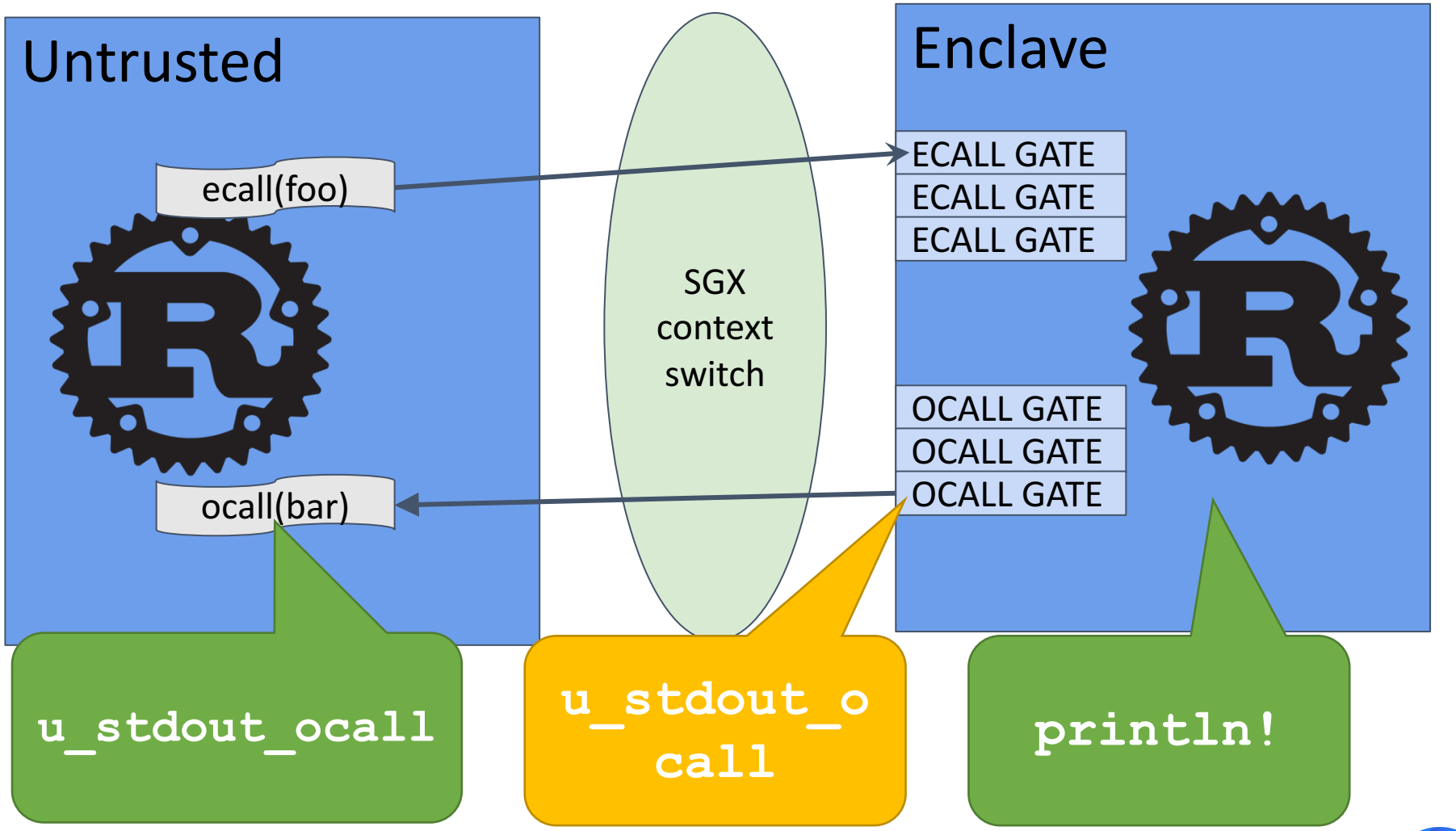
stdio.edl

```
enclave {
    untrusted {
        size_t u_stdin_ocall([out, size=nbytes] void *buf, size_t nbytes);
        size_t u_stdout_ocall([in, size=nbytes] const void *buf, size_t nbytes);
        size_t u_stderr_ocall([in, size=nbytes] const void *buf, size_t nbytes);
    };
};
```

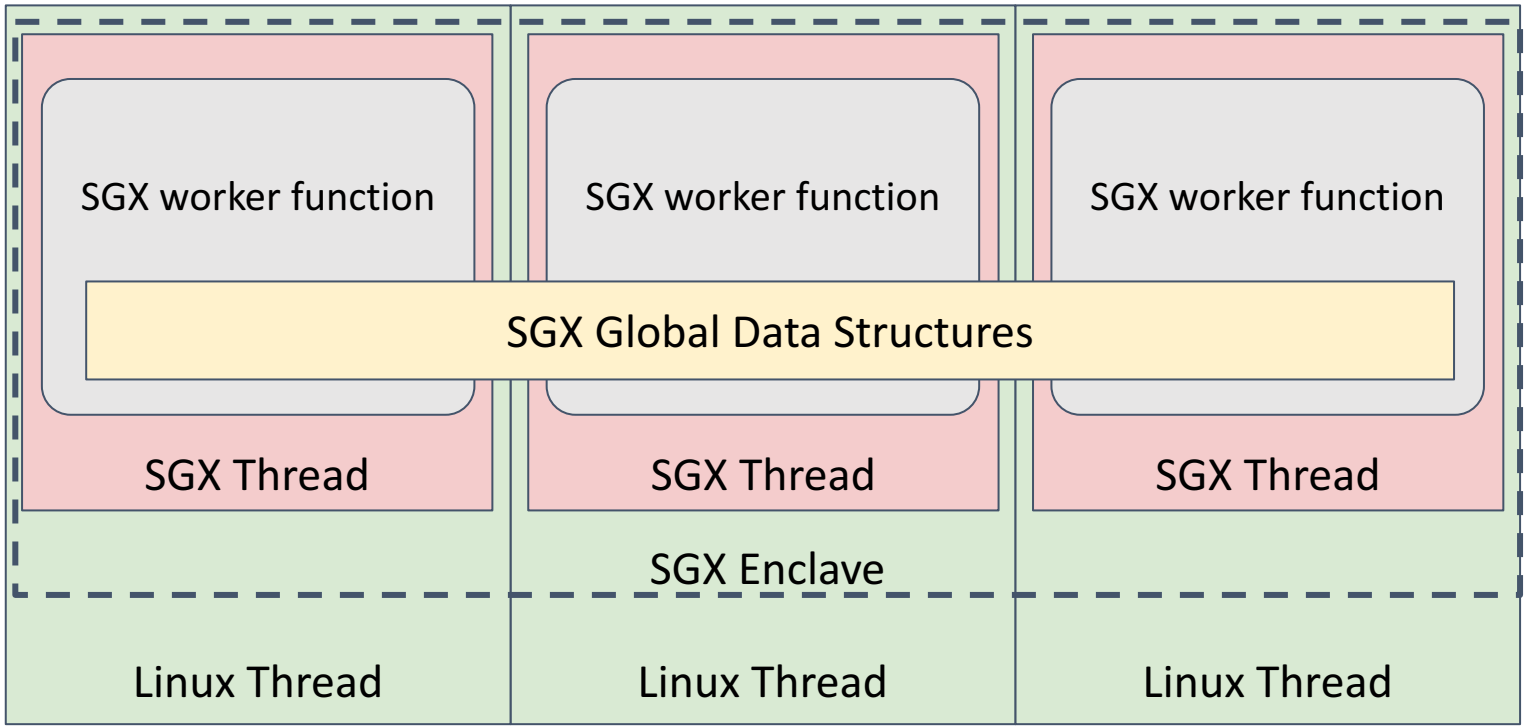
Untrusted Run-time library sgx_urts implements the feature functions

```
#[no_mangle]
pub extern "C"
fn u_stdout_ocall(buf: * const libc::c_void, nbytes: libc::size_t) -> libc::size_t
{
    unsafe { libc::write(libc::STDOUT_FILENO, buf, nbytes) as libc::size_t}
}
```

Rust SGX SDK : Partition and Interaction with OS



Rust SGX SDK : Threading by Sample



"Re-entry" using TCS pool

TCSPolicy

- **BOUND vs UNBOUND**

TCSNUM

- **Max SGX TCS number**

Rust SGX SDK：Major Differences

Rust	Intel SGX in C	Rust SGX
Mutex Mutex::new(0);	sgx_thread_mutex_t struct { sgx_thread_mutex_t mutex; uint32_t n; };	SgxMutex SgxMutex::new(0);
Thread Posix Thread	"Re-entry" Bound: stick to pthread Unbound: random pick	"Re-entry" Bound: stick to pthread Unbound: random pick
Thread-Local Storage ThreadLocal::new(); ctor/dtor supported	get_thread_data() BOUND: no ctor/dtor UNBOUND: no ctor/dtor	thread_local! BOUND: **ctor/dtor** UNBOUND: no ctor/dtor

Rust SGX SDK：Exceptions and Signals

Exception Handling

- Implement `panic-unwind` mechanism
  - Unwind safely in Rust style
- Implement stack backtrace mechanism
  - (optional) Dump call stack on panicking

Signals

- Intel SGX: AEX mechanism, exception handler registration
- Rust SGX SDK
  - Re-export handler_register and handler_unregister function
  - Provide handlers to some sigs
    - CPUID/RDTSC etc

How to use?

It's easy!

Rust SGX SDK : Features

std => sgx_tstd

- Most of std's features are supported.
- Partially support of std::fs, std::os, std::path, std::sync, std::thread
- No support of std::env, std::net, std::process, std::time

Intel SGX related libraries

- sgx_tcrypto, sgx_tdh, sgx_tkey_exchange, sgx_tprotected_fs, sgx_trts, sgx_tse, sgx_tseal, sgx_tservice

Rust style libraries

- sgx_alloc, sgx_rand, sgx_serialize, sgx_tunittest, sgx_types

Supportive libraries in untrusted world

- sgx_ubacktrace, sgx_urts, sgx_ustdio

Rust SGX SDK : Porting Rust Crates to Intel SGX

Replace dependency of Rust's std to sgx_tstd

1. Add dependency in Cargo.toml

```
sgx_tstd = { path = "path/to/sgx_tstd" }
```

2. Change to a no_std environment in lib.rs

```
#[no_std]
```

3. Include sgx_tstd in namespace of std

```
extern crate sgx_tstd as std;
```

4. Fix all incompatible usage

```
Mutex => SgxMutex
```

5. Use **`sgx_tstd`** as usual

```
use std::vec::Vec;
```

Rust SGX SDK：An Easy-to-use SDK

- **Shipped with a docker image**

  —docker pull baiduxlab/sgx-rust

- **Complete Rust-style documents**

  — https://dingelish.github.io/

- **Rich code samples**

  —hello-rust, file, backtrace, hugemem(31.75GB), local attestation, remote attestation, data sealing, serialization, threading, unit testing, 3rd party code samples

- **Support latest Intel SGX SDK (v1.9)**
- **Support latest Rust nightly build**
- **A better choice than sgx-utils (libenclave)**

Rust SGX SDK : Now and Future

- **Recommended by Intel, adopted by chain.com**

French Alternative Energies and Atomic Energy Commission (CEA) wins iDash'17 competition using Rust SGX SDK

Rust SGX SDK : Now and Future

- **Getting Hot!**

- **Future**

  - New target : x86_64-unknown-linux_sgx
  - Support rust style #[test]
  - std::net
  - std::time
  - Porting Rust's ring to SGX
  - Porting Rust's rustls to SGX

THANKS

Baidu X-Lab