TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Confidential Computing

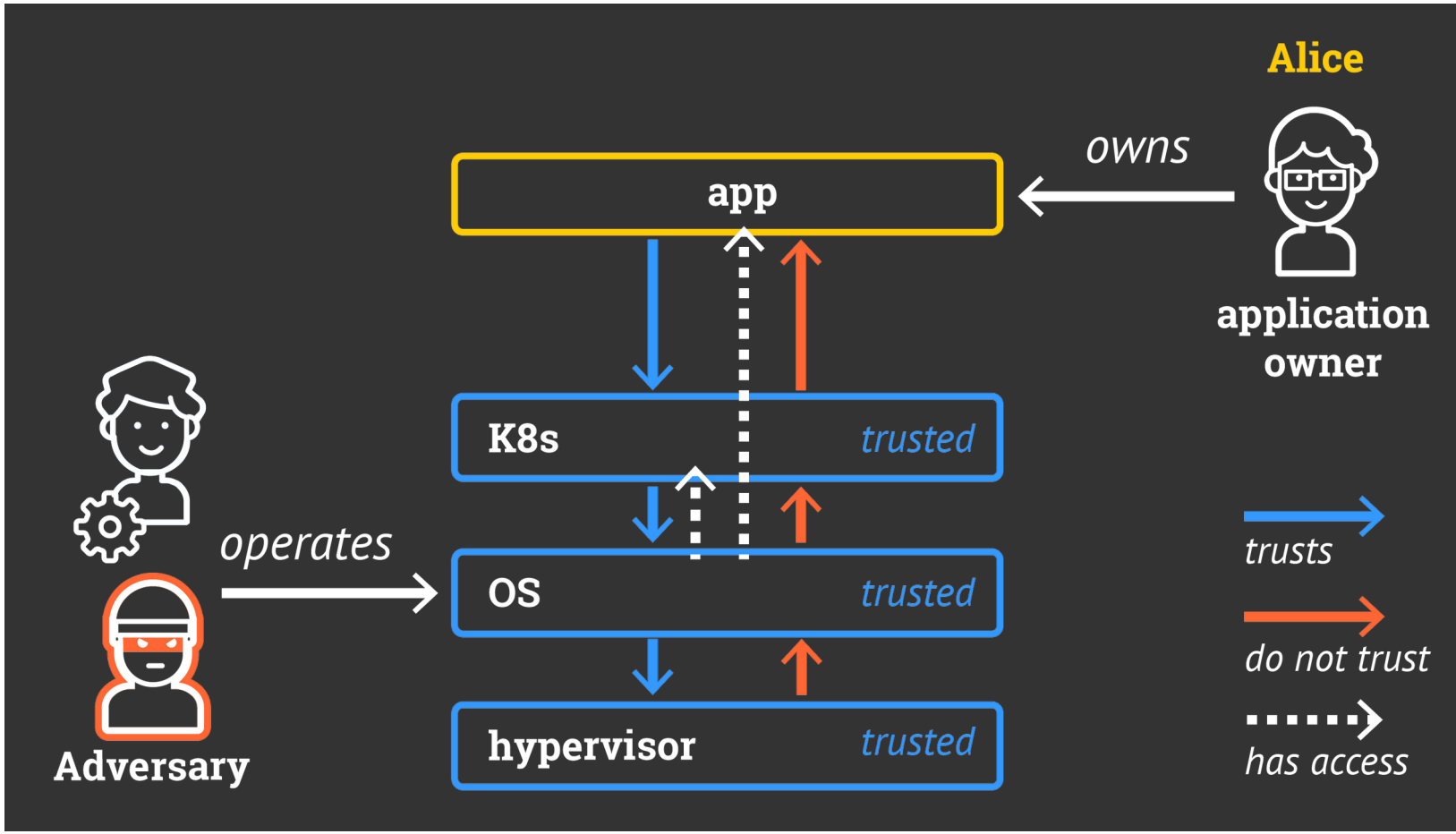
Exercise 1: Introduction to Confidential Computing (CC)

Ardhi Putra Pratama Hartono

Chair of System Engineering

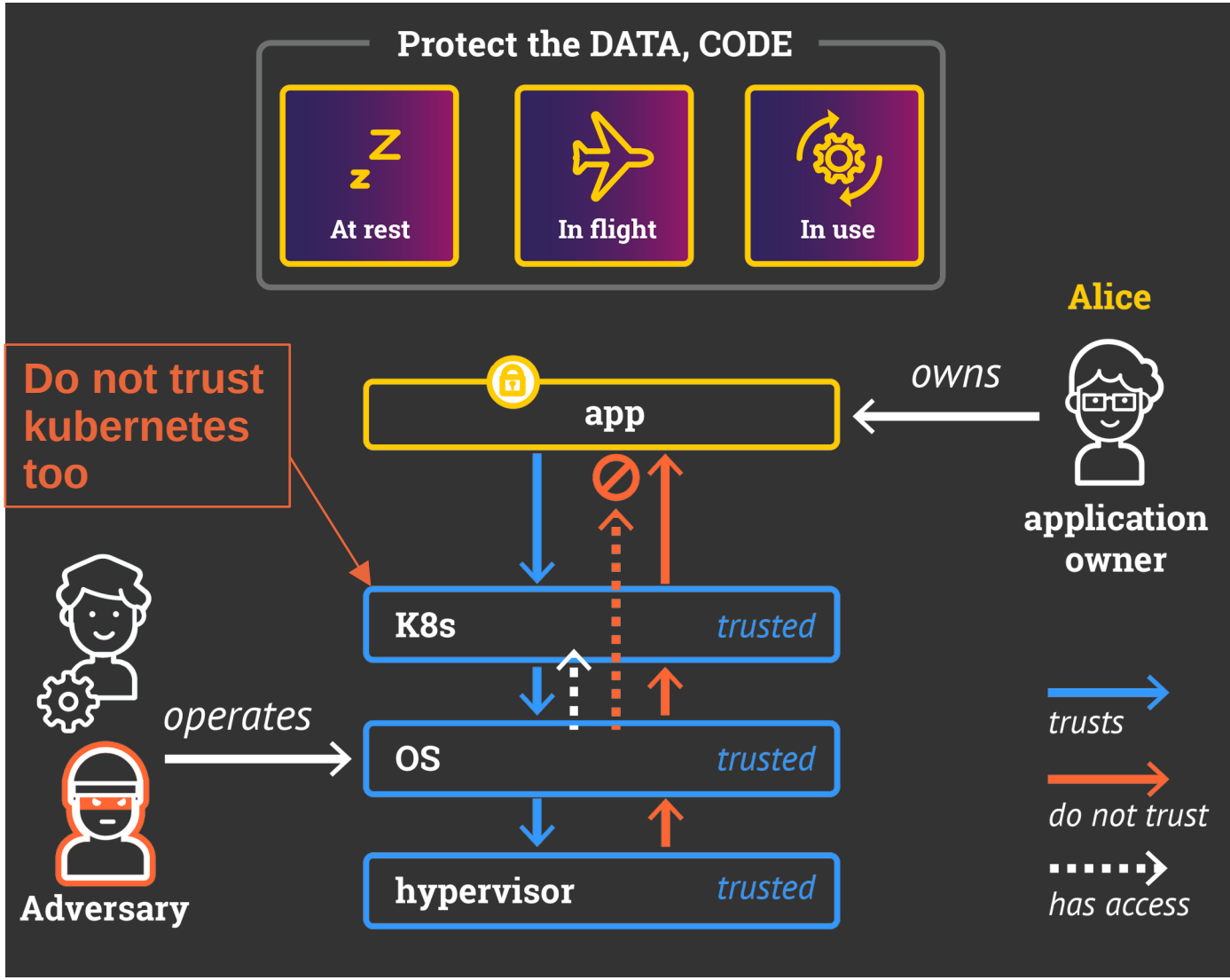
Why Confidential Computing matters

- **Systems security permits**
  - administrators can access layers above
- **Problem:**
  - we need to limit access of admins to maintain the integrity of the different layers
- **Approach:**
  - we put app and some layers in separate TEEs (Trusted Execution Environments)



Why Confidential Computing matters – cont'd

- **Confidential Computing**
  - confidential service executes in an encrypted memory region (aka enclave, TEE)
  - only the application code can access the data, code, and secrets of the application (in plain text)
  - attestation: application owner/clients establish trust with the application
- **Cloud deployment**
  - Most of the time, you need to **trust** the provider
  - Often, on-premise deployment is not a solution



What is typically addressed?

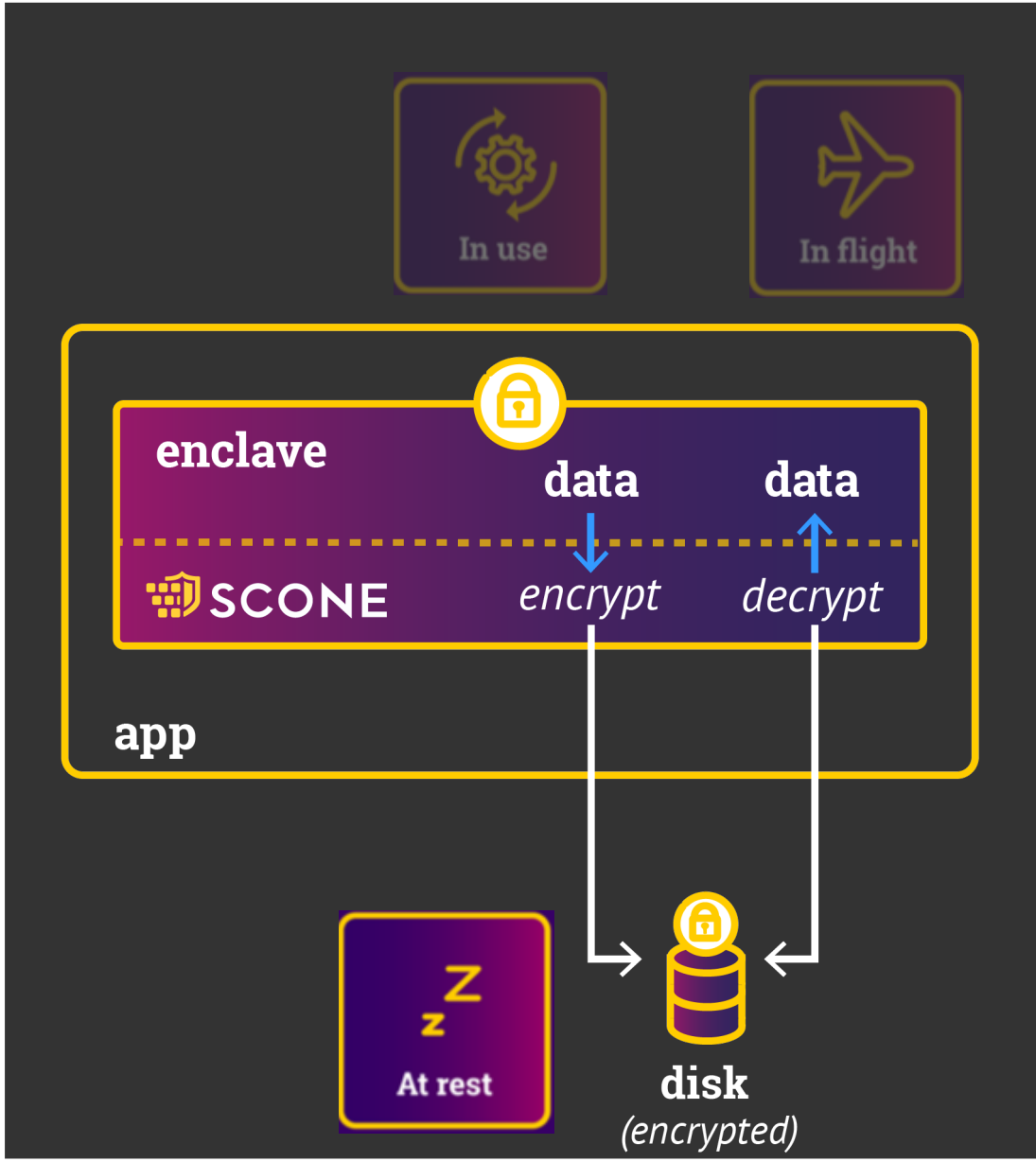
- **Some attacks :**
  - Software attacks (attacks on OS, software bugs, privilege excalation, etc.)
  - Protocol attacks (in attestation, on data transport, ensuring data integrity when leaving the application)
  - Basic physical attacks (cold DRAM extraction, pluggable device, voltage manipulation, etc.)
- **Powerful adversary :**
  - Could control the entire stack (OS, network, kernel, reorder network messages, etc)
  - Have physical access to the hardware
  - Malicious software deployment and supply chain attacks
- But you still need to **trust the hardware (+its manufacturer)**
  - Along with the process of building the TCB (Including attestation, etc.)

Confidential Computing in a nutshell

- **Confidential computing is** Hardware-enabled features that isolate and process encrypted data in memory so that the data is at less risk of exposure and compromise from concurrent workloads or the underlying system and platform.[NIST]
- Rely on **hardware** root-of-trust, which include memory isolation and encryption.
- Provide end-user verifiable “report“ (remote attestation), trusted launch , and secure key management
- More popular term : **Trusted execution environment (TEE)**
  - A Trusted Execution Environment (**TEE**) is an isolated environment which aims to protect executions within against high privileged adversaries.
    - Software TEEs solely rely on software mechanisms for protection, while
    - Hardware TEEs use additional hardware mechanisms to protect the confidentiality and integrity of code and data within the environment.

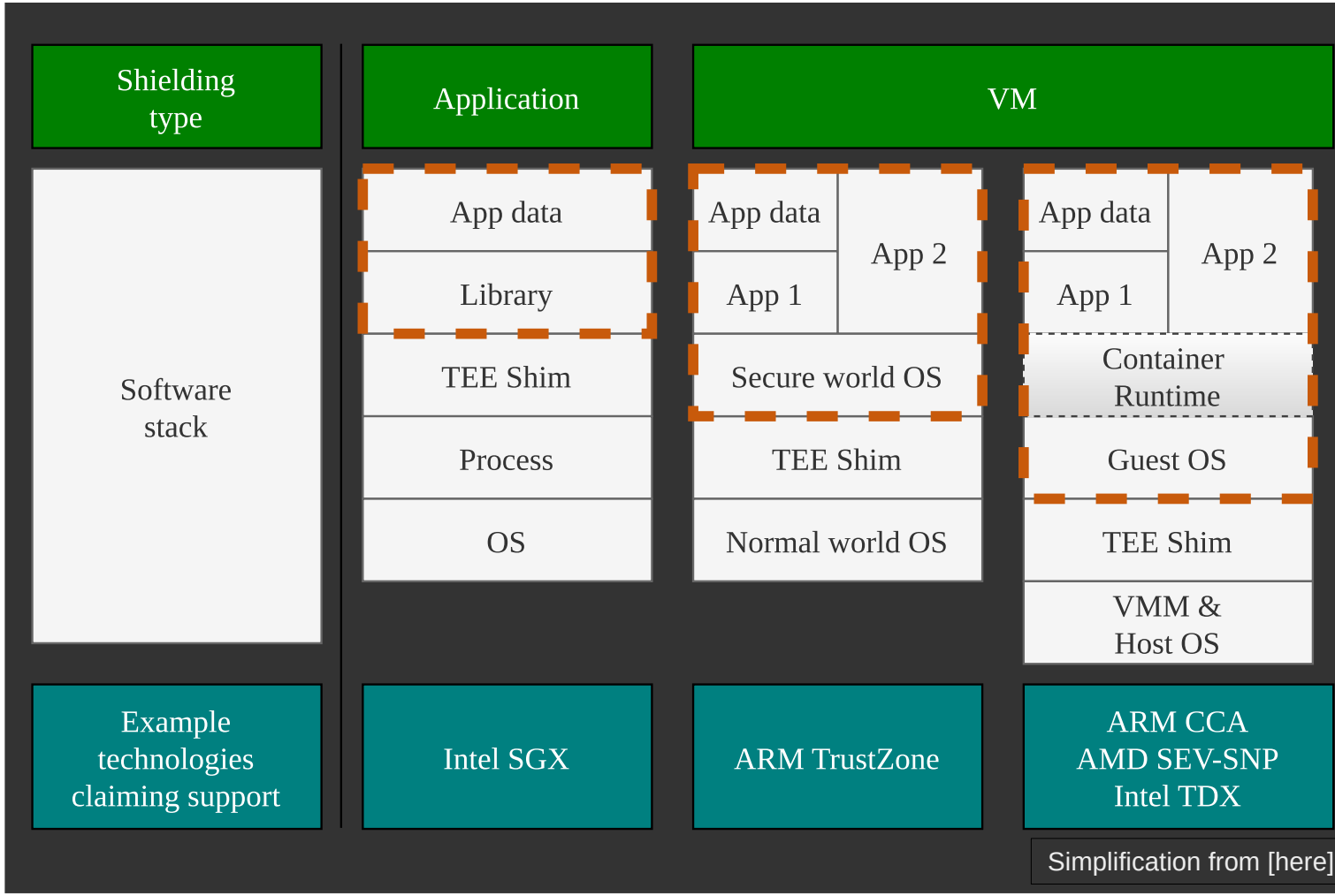
Confidential Computing in a nutshell - cont'd

- Some cases : exchange data with multi-party securely (and whom you can trust)
- Mostly focused on **data in use,** while :
  - Data in transit : TLS, etc.
  - Data at rest : your typical encryption algorithm
- Protecting the **Data** and **Code** :
  - **Data** Protection: TEEs encrypt data and ensure its confidentiality, protecting it from unauthorized access
  - **Code** Integrity: Unauthorized entities cannot add, remove, or alter code executing in the TEE



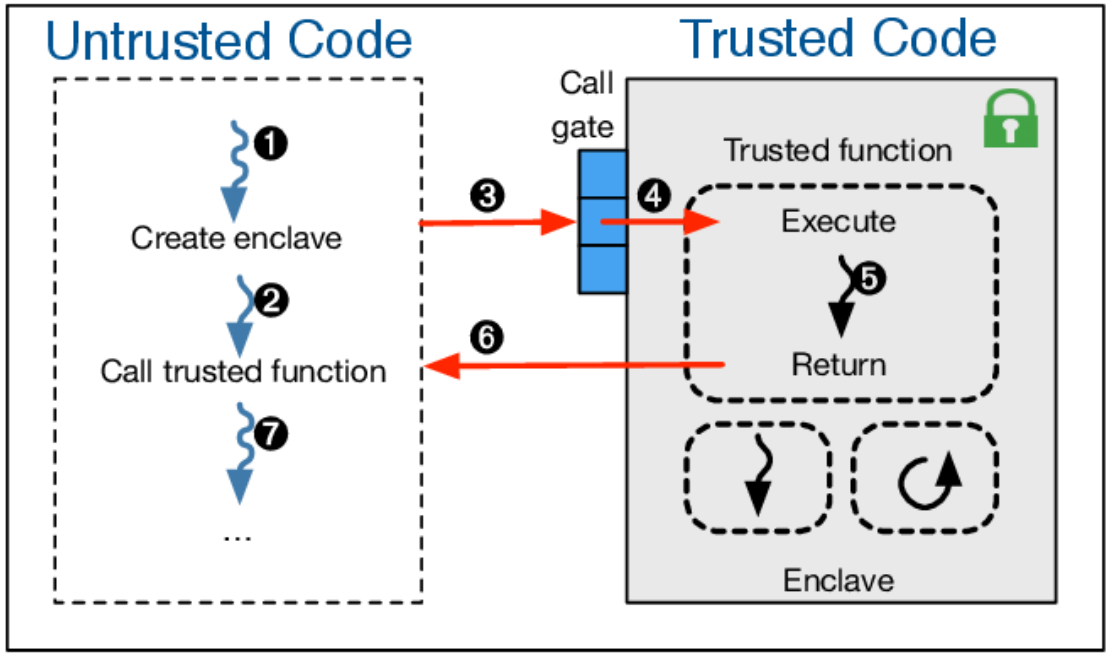
Confidential Computing implementations

- Different approach for different requirements
  - Process → Intel SGX
  - Virtual Machine → SEV, TDX, TrustZone/CCA
- Some frameworks may also help with the implementation :
  - SCONE : lib-c
  - Ego : somewhere in-between, perhaps closer to lib-c (?)
  - Gramine : libOS
  - Occlum: libOS

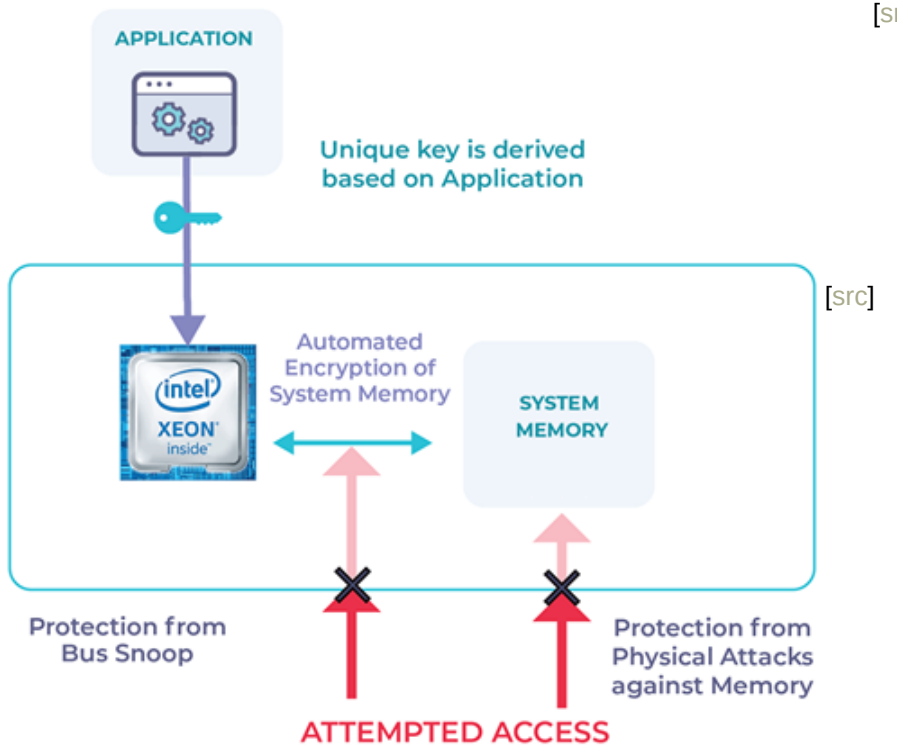


Intel SGX

- **Enclave** is a secure area in memory, protected by CPU special instructions
  - Compomised OS, hypervisor, other application cannot access this
- **Attestation** : prove to local/remote system what code is running in an enclave
- Minimum TCB : only processor is trusted
- Have trusted and untrusted part, trusted part can ony accessible within the enclave

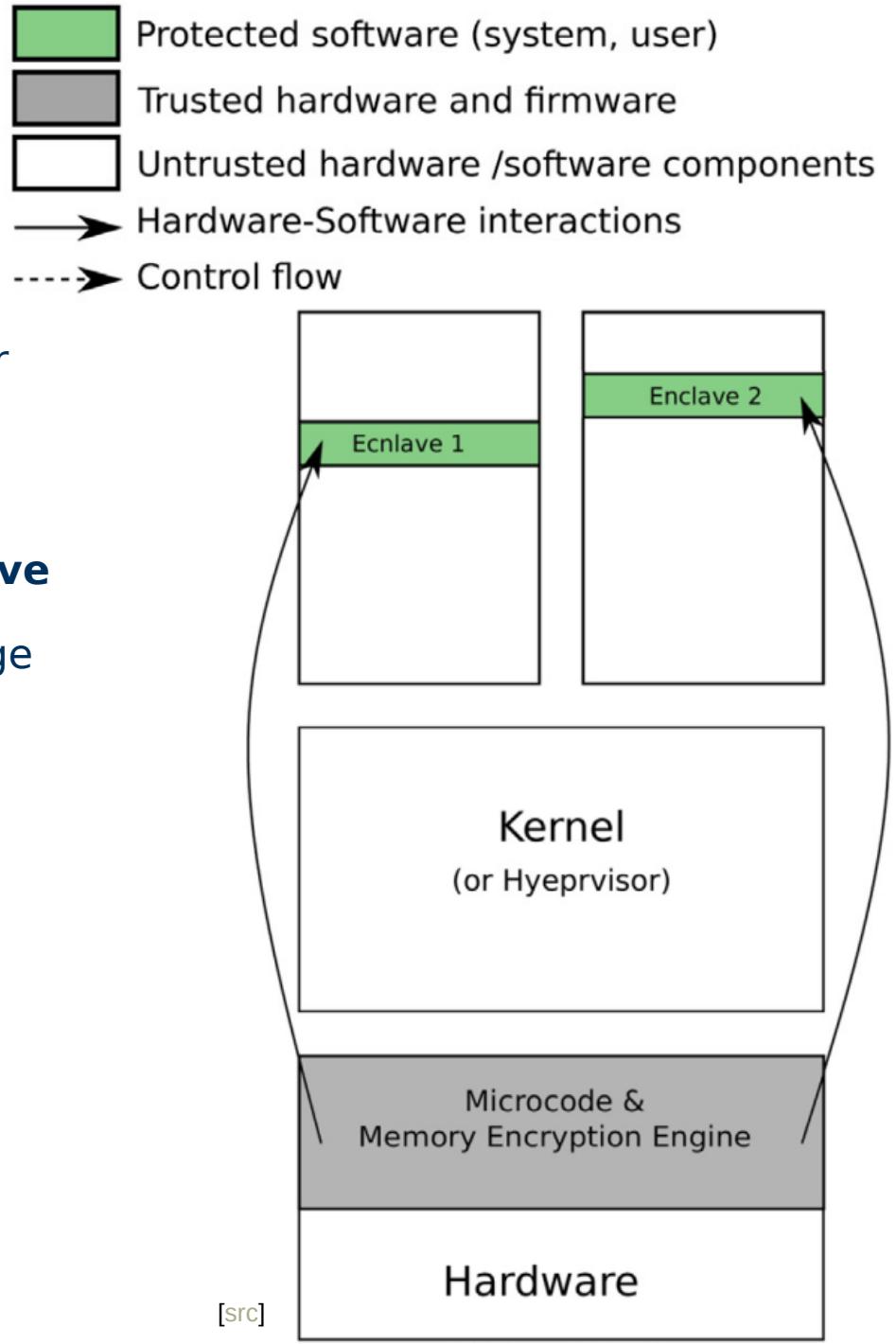


[src]



[src]

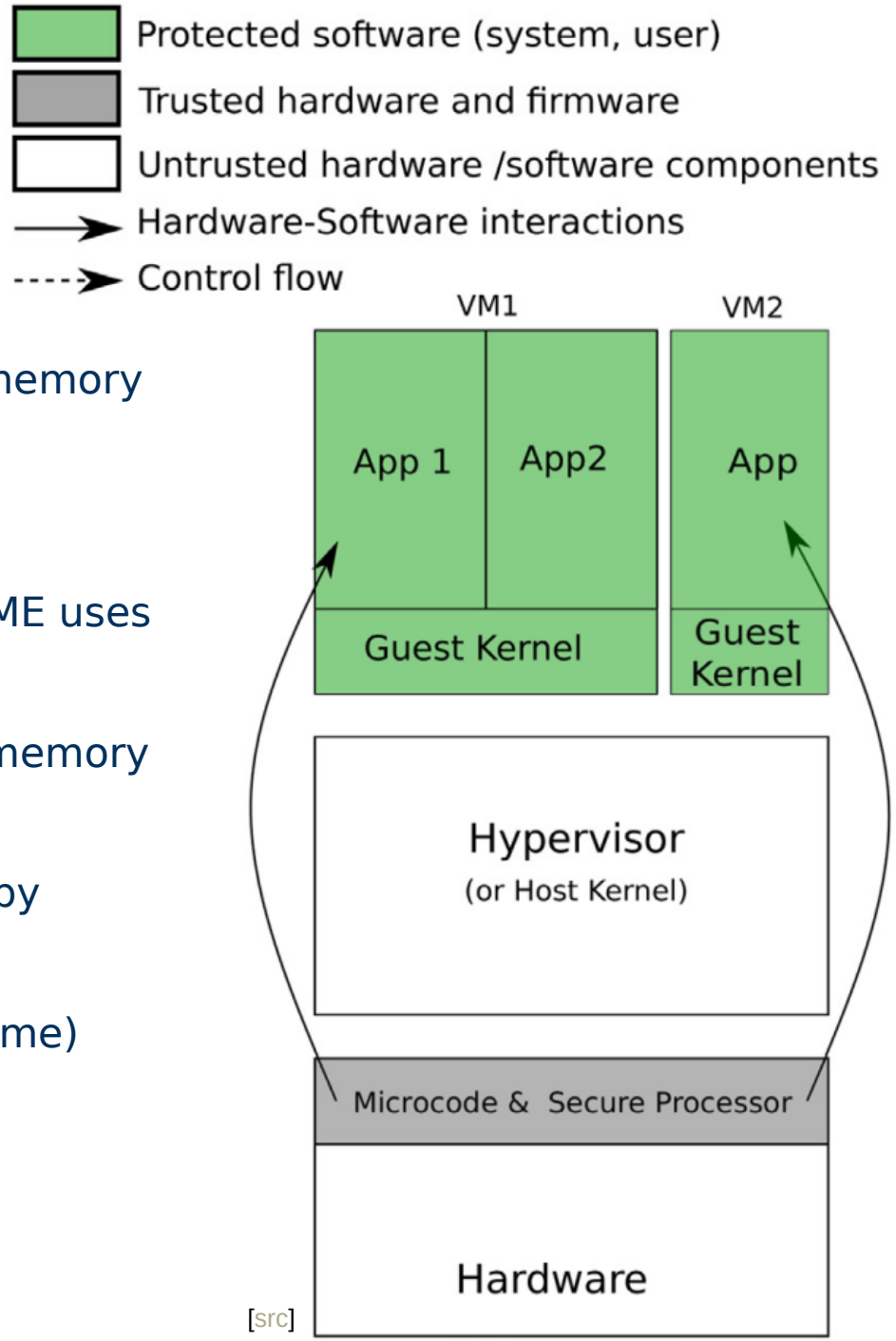
Intel SGX - cont'd



- Shipped within every Intel Skylake CPU of the 6th generation or newer
  - But nowadays only on server's CPU (Xeon)
- Allows code parts to be executed in hardware separated **enclave**
- On Intel Xeon it limits to 512MB-512GB RAM simultaneous usage of SGX enclaves – old machine has ~ 128MB (Skylake)
  - It's called EPC : **Enclave Page Cache**
- Possible to **attest** that code is running inside an SGX enclave
- TLS session can be terminated directly in enclave
- Enclave guarantee's code integrity despite malicious OS
  - It resides in the isolated memory region

[src]

AMD SEV

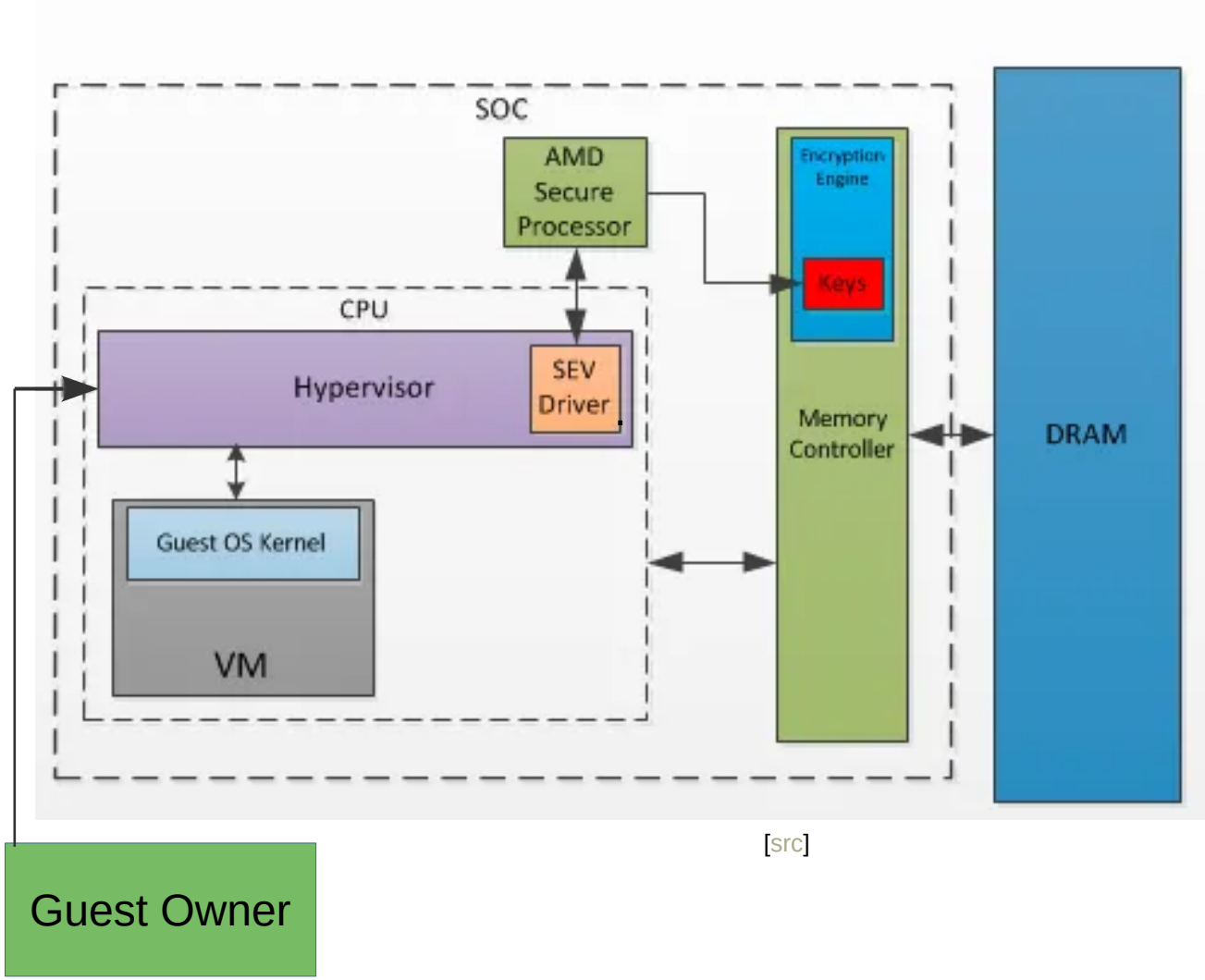


- **AMD Secure Memory Encryption (SME)** allows to encrypt memory content before writing in to RAM
  - Prevents an attacker from physical RAM reading attacks
- **AMD Secure Encrypted Virtualization (SEV)** is based on SME uses a different key for each virtual machine.
  - This prevents a malicious hypervisor from reading a VM's memory content.
  - Moving ciphertext between memory location is prevented by encryption of physical memory address
- However, the encryption slots is limited : 16 (Naples) – 255 (Rome)
- One can attest the Confidential VM

[src]

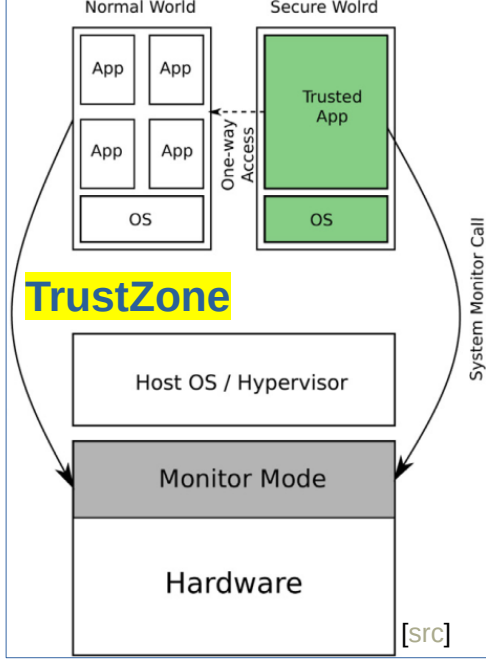
AMD SEV - cont'd

- It's a VM, therefore, changes to code is unnecessary
- Hypervisor only manage (launching, running, snapshotting, etc.), but doesn't know the key
- The generic process is the following :
  - Owner ask hypervisor to deploy encrypted VM image
  - Hypervisor tell SP to encrypt memory and launch VM
  - MC measure VM, produce hash, confirming with owner
  - Owner send key to decrypt data in VM



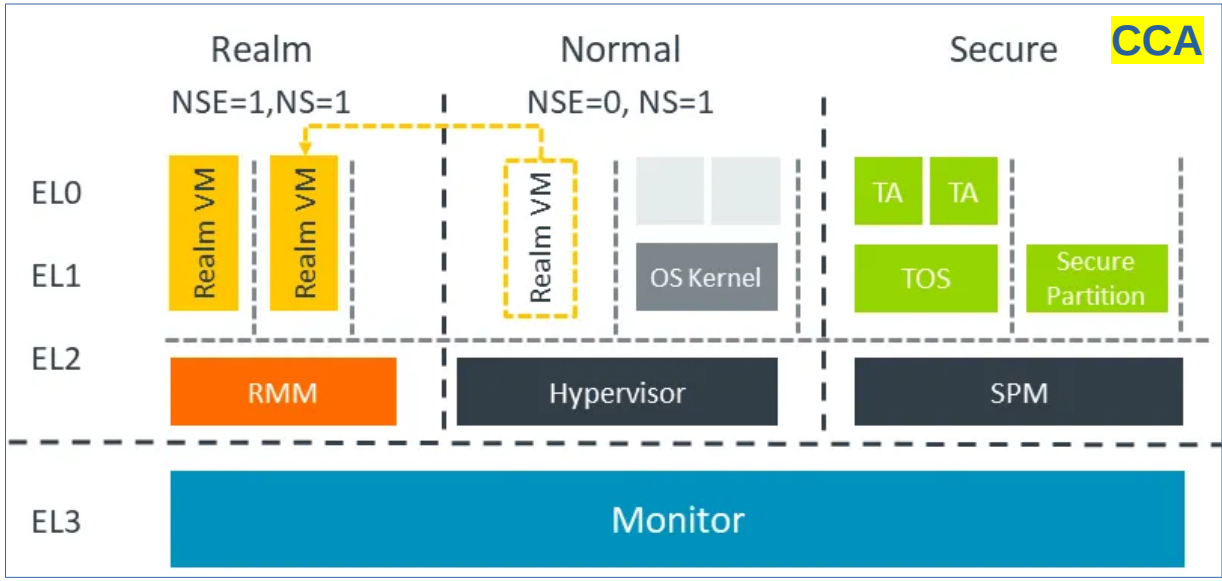
[src]

ARM CCA



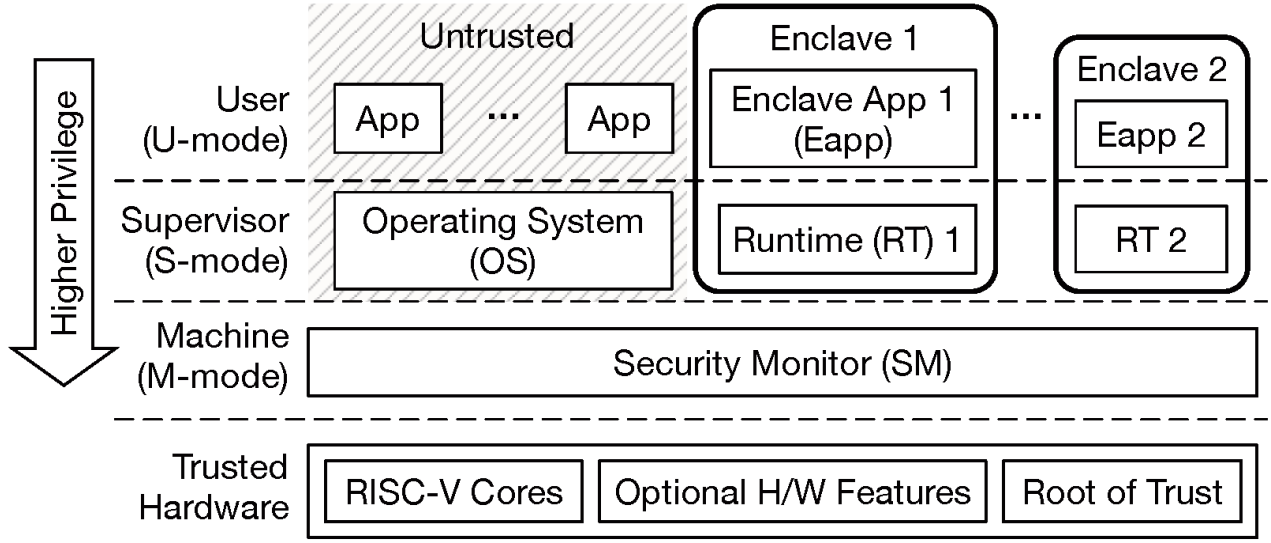
- Very similar to AMD SEV (and Intel TDX)
- Introduce “Realm state” → processor state, and „Root“ (to switch worlds)
- Inspired from **TrustZone**, old CC solution (backward-compatible)
  - In TrustZone, ARM introduce normal and secure/trusted world
  - More dynamic (Realm creation, memory allocation), add attestation, not trust OS anymore

- Some terms :
  - EL => exception levels, privilege level
  - **RMM**: Realm management monitor
  - „Monitor“ control MMU for memory isolation; the Root world.
- A VM in Realm world is controlled from a Normal world, but its content is isolated



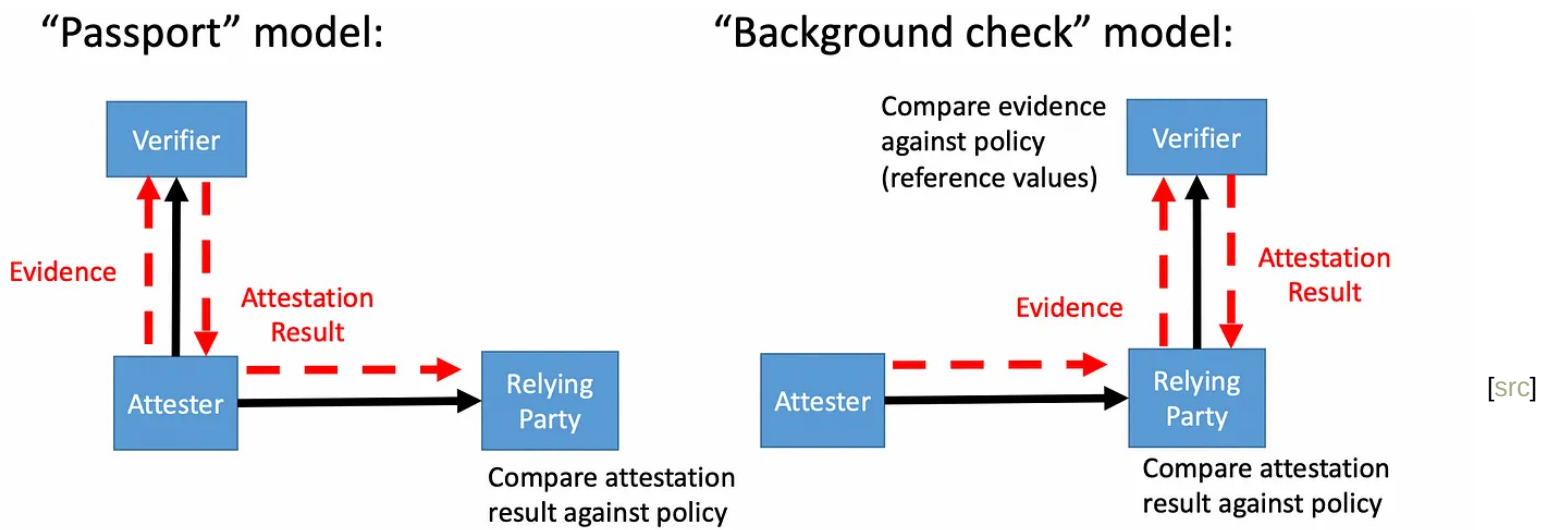
RISC-V: Keystone

- Open source project to build **customizable TEE** based on RISC-V
  - Provide building blocks for custom TEE
  - Include only required functionality → reduced TCB
- Important components :
  - **SM**: ensuring security guarantee → enforce memory isolation
  - **RT**: like a enclave's kernel. Isolated from OS. Implements enclave functionality
  - **Root of Trust**: SM and security primitives in the hardware



Attestation

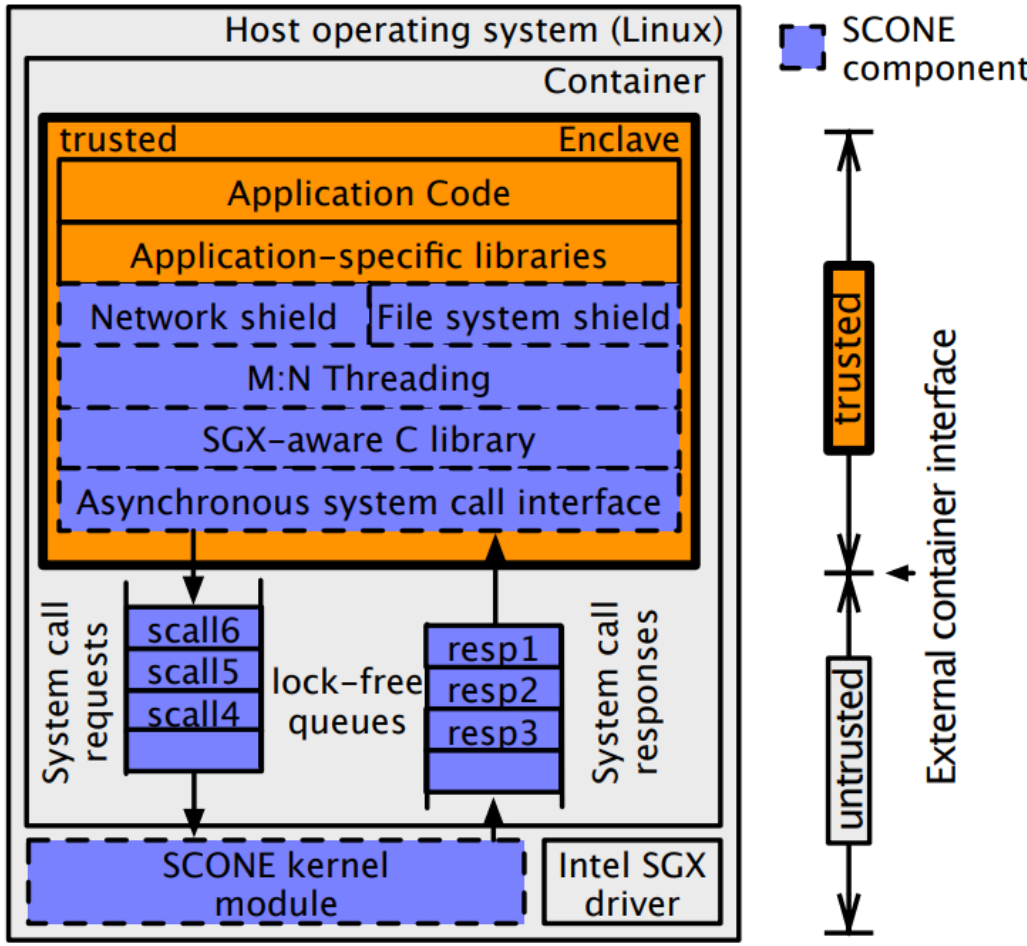
- **Attestation** is a process that verifies the trustworthiness and authenticity of a TEE or confidential computing environment. Typically initiated by the TEE when it loads.
  - It provides evidence that a system is genuine and operating in a secure state before sensitive data or code is entrusted to it.
  - Imagine like a passport/background check model
- Attestation could be done remotely, called **remote attestation.** After everything has been attested, then the application can be launched in a trusted manner.



[src]

SCONE: Secure Linux Containers with Intel SGX

- SCONE include a **shielding layer** as an interface to application code to be able to run inside an SGX enclave effortlessly
  - Use musl-based libc inside the enclave
  - Able to run unmodified application on Intel SGX (Remember the trusted-untrusted part)
- System calls will still be executed outside of the enclave
  - Protect against system call attack to some extent
- Offers additional protection mechanism : network, file system



TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Hands-on / Demo

Confidential Computing in action using SCONE

Protecting secrets in memory: Native vs SCONE

- Consider a C program :
  - It receives an input from user interactively
  - Then **stores a „secret“** in variable „input“
- This program will be compiled with and without Intel SGX capabilities, i.e., using an enclave
- Another terminal will dump the memory for this program (when it runs)

```
.....
.....
int main() {
    printf("Enter a secret: ");

    // store input to buffer
    char* input = get_input();

    // hash the secret and print only that
    uint32_t thehash = simple_hash(input);

    printf("Hash of your input: %u",
            thehash);
    printf("\n");

    // pause the program, dump memory in
    // another terminal
    getchar();

    free(input);
    return 0;
}
```

Protecting secrets in memory: dumping memory

- Different script will analyze the stack to extract the “secret“
- Basically inspecting the ***/proc*** filesystem to see the memory footprint

- We will see that in Native execution, a secret could be extracted.
- While in SCONE-SGX (Production mode), this is prevented

Source: https://sconedocs.github.io/memory_dump/

```
import sys,re

pid = sys.argv[1]
print("PID = %s \n" % pid)
maps_file = open("/proc/%s/maps" % pid, 'r')
mem_file= open("/proc/%s/mem" % pid, 'rb')

for line in maps_file.readlines():
    m = re.match(r'([0-9A-Fa-f]+)-([0-9A-Fa-f]+)
([-r][-w])', line)
    if m.group(3) == "rw" or m.group(3) == "r-" :
        try:
            start = int(m.group(1), 16)
            if start > 0xFFFFFFFFFFF:
                continue
            print("\nOK : \n" + line+"\n")
            end = int(m.group(2), 16)
            mem_file.seek(start)
            chunk = mem_file.read(end - start)
            print(chunk)
            sys.stdout.flush()
        except Exception as e:
            print(str(e))
    else:
        print("\nPASS : \n" + line+"\n")

print("END")
```

Protecting secrets in memory: Native

```
ardhi@icelake2:~/cc-exercise$ ps x | grep app.native.exe
 545534 pts/6    S+     0:00 ./app.native.exe
 545823 pts/5    S+     0:00 grep --color=auto app.native.exe
ardhi@icelake2:~/cc-exercise$ sudo python dumpmem.py 545534 >& native.log
ardhi@icelake2:~/cc-exercise$ grep -o FUNFOO native.log
FUNFOO
FUNFOO
ardhi@icelake2:~/cc-exercise$
```

```
ardhi@icelake2:~/cc-exercise$ ./app.native.exe
Enter a secret: FUNFOO
Hash of your input: 2084933351
```

Protecting secrets in memory: with SGX (Production mode)

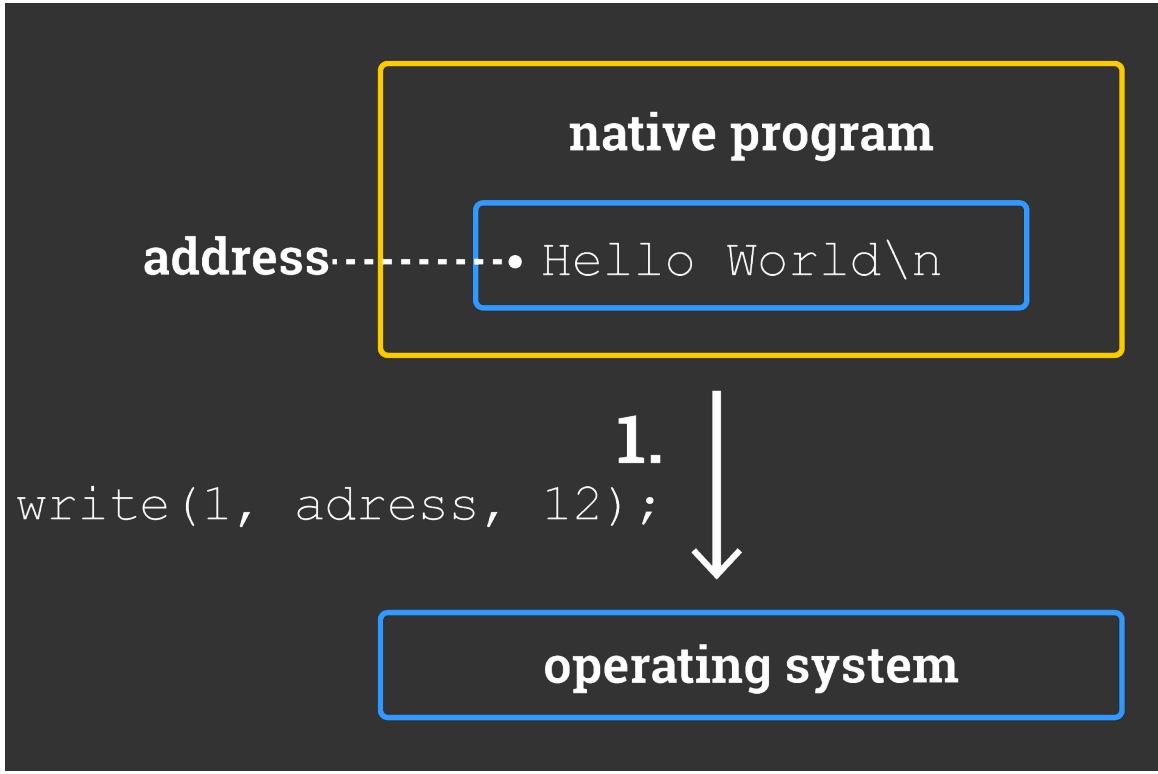
```
ardhi@icelake2:~/cc-exercise$ ps x | grep app.scone-prod.exe
 547142 pts/6    Sl+    0:00 ./app.scone-prod.exe
 547241 pts/5    S+     0:00 grep --color=auto app.scone-prod.exe
ardhi@icelake2:~/cc-exercise$ sudo python dumpmem.py 547142 >& scone-prod.log
ardhi@icelake2:~/cc-exercise$ grep -o FUNFOO scone-prod.log
ardhi@icelake2:~/cc-exercise$
```

```
ardhi@icelake2:~/cc-exercise$ SCONE_PRODUCTION=1 SCONE_KEY=~/scone-ref/compiler/musl-cross-make/tests/dummy_key.pem ~/scone-ref/built/bin/scone-gcc app.c -o app.scone-prod.exe
/usr/local/bin/ld: /home/ardhi/scone-ref/built/cross-compiler/x86_64-linux-musl/bin/../lib/misc/libsgx.a(std-11c02606063fb1b5.std.b98470a71b0fbcc3-cgu.0.rcgu.o): in function `std::env::home_dir':
(.text._ZN3std3env8home_dir17h94aee902005cb344E+0x102): warning: Using 'getpwuid_r' in statically linked applications requires at runtime the shared libraries from the glibc version used for linking
/usr/local/bin/ld: /home/ardhi/scone-ref/built/cross-compiler/x86_64-linux-musl/bin/../lib/misc/libsgx.a(std-11c02606063fb1b5.std.b98470a71b0fbcc3-cgu.0.rcgu.o): in function `<std::sys_common::net::LookupHost as core::convert::TryFrom<(&str,u16)>>::try_from::{{closure}}':
(.text._ZN104_$LT$std..sys_common..net..LookupHost$u20$as$u20$core..convert..TryFrom$LT$$LP$$RF$str$C$u16$RP$$GT$$GT$8try_from28_$u7b$$u7b$closure$u7d$$u7d$17h555ee5a61f007770E+0x5a): warning: Using 'getaddrinfo' in statically linked applications requires at runtime the shared libraries from the glibc version used for linking
ardhi@icelake2:~/cc-exercise$ SCONE_MODE=hw ./app.scone-prod.exe
[SCONE|WARN] tools/starter/main.c:553:enclave_create(): Enclave is signed to run in production mode. Environment variables affecting enclave measurement will be ignored
[SCONE|WARN] src/enclave/dispatch.c:168:print_runtime_info(): Application runs in SGX production mode, but the SCONE runtime is compiled in debug mode. Recompile the runtime in production mode or obtain a production version from Scontain
FUNFOO
Enter a secret: Hash of your input: 2084933351
```

Inspecting syscalls

- System call can be traced by using **`strace`** tool
- Argument passing via addresses, the actual stuff will be done by the operating system
- On the application, *syscall* write:
  - passes address of string, and size of string
- While in the operating system:
  - copies string to local buffer, and
  - prints it to „*stdout*“

In **Trusted Execution**, *`strace`* output will not show anything obvious!



Inspecting syscalls: Native vs SGX

```
execve("./app.native.exe", ["./app.native.exe"], 0x7fff2c1c1bc0 /* 117 vars */) = 0
brk(NULL)                               = 0x55e33706d000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffebe3e3c60) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=77051, ...}) = 0
mmap(NULL, 77051, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fcbf1a49000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\300A\2\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\304"..., 68, 880) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2029592, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcbf1a47000
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\304"..., 68, 880) = 68
mmap(NULL, 2037344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fcbf1855000
mmap(0x7fcbf1877000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7fcbf1
877000
mmap(0x7fcbf19ef000, 319488, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) = 0x7fcbf19ef000
mmap(0x7fcbf1a3d000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7fcbf1
a3d000
mmap(0x7fcbf1a43000, 13920, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fcbf1a43000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fcbf1a48540) = 0
mprotect(0x7fcbf1a3d000, 16384, PROT_READ) = 0
mprotect(0x55e335864000, 4096, PROT_READ) = 0
mprotect(0x7fcbf1a89000, 4096, PROT_READ) = 0
munmap(0x7fcbf1a49000, 77051)           = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x6), ...}) = 0
brk(NULL)                               = 0x55e33706d000
brk(0x55e33708e000)                     = 0x55e33708e000
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x6), ...}) = 0
write(1, "Enter a secret: ", 16)        = 16
read(0, "FUNFOO\n", 1024)               = 7
write(1, "Hash of your input: 2084933351\n", 31) = 31
read(0, 0x55e33706d6d0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGINT {si_signo=SIGINT, si_code=SI_KERNEL} ---
+++ killed by SIGINT +++
~
~
~
```

```
mprotect(0x7f0e26d88000, 8388608, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
clone(child_stack=0x7f0e27586c78, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[550411], tls=0x7f0e275
87700, child_tidptr=0x7f0e275879d0) = 550411
clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000}, NULL) = 0
clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000}, NULL) = 0
dup(2)                                  = 3
fcntl(2, F_GETFD)                       = 0
rt_sigprocmask(SIG_SETMASK, ~[RTMIN RT_1], NULL, 8) = 0
brk(0x52b2000)                          = 0x52b2000
openat(AT_FDCWD, "/etc/sgx-musl.conf", O_RDONLY) = 4
fstat(4, {st_mode=S_IFREG|0644, st_size=22, ...}) = 0
read(4, "Q 1\ne 81 0 0\ns 82 0 0\n", 4096) = 22
sched_get_priority_min(SCHED_OTHER)     = 0
sched_get_priority_max(SCHED_OTHER)     = 0
getpid()                                = 550408
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f0e26066000
mprotect(0x7f0e26067000, 8388608, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
clone(child_stack=0x7f0e26865c78, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[550412], tls=0x7f0e268
66700, child_tidptr=0x7f0e268669d0) = 550412
sched_setscheduler(550412, SCHED_OTHER, [0]) = 0
sched_setaffinity(550412, 128, [82])    = 0
read(4, "", 4096)                       = 0
close(4)                                = 0
openat(AT_FDCWD, "/etc/sgx-musl.conf", O_RDONLY) = 4
fstat(4, {st_mode=S_IFREG|0644, st_size=22, ...}) = 0
read(4, "Q 1\ne 81 0 0\ns 82 0 0\n", 4096) = 22
sched_get_priority_min(SCHED_OTHER)     = 0
sched_get_priority_max(SCHED_OTHER)     = 0
getpid()                                = 550408
mmap(NULL, 4198400, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f0e26986000
mprotect(0x7f0e26987000, 4194304, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
clone(child_stack=0x7f0e26d85c78, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[550413], tls=0x7f0e26d
86700, child_tidptr=0x7f0e26d869d0) = 550413
sched_setscheduler(550413, SCHED_OTHER, [0]) = 0
futex(0x7f0e26d86d18, FUTEX_WAKE_PRIVATE, 1) = 1
sched_setaffinity(550413, 128, [81])    = 0
read(4, "", 4096)                       = 0
close(4)                                = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x29f6c50, FUTEX_WAIT, 0, NULL)   = ?
+++ exited with 0 +++
```

Take away

- If you **don't trust** everyone: Confidential computing matters
  - It offers limited parties you could/might trust
- With a TEE, you need to trust **the manufacturer**, but then that's it.
  - You might trust the framework, via audit or source inspection
- TEE comes with various **implementation**
  - The most popular one still by Intel and AMD
  - They share the similarities : isolation, encryption, attestation
- From our **hands-on**: you can inspect the memory to get secrets.
  - TEE (e.g., Intel SGX), however, prevents that.