

Better Foundations for Secure Software using Trusted Hardware and Verification

Shweta Shinde

ETH Zürich

Shweta Shinde
Assistant Professor

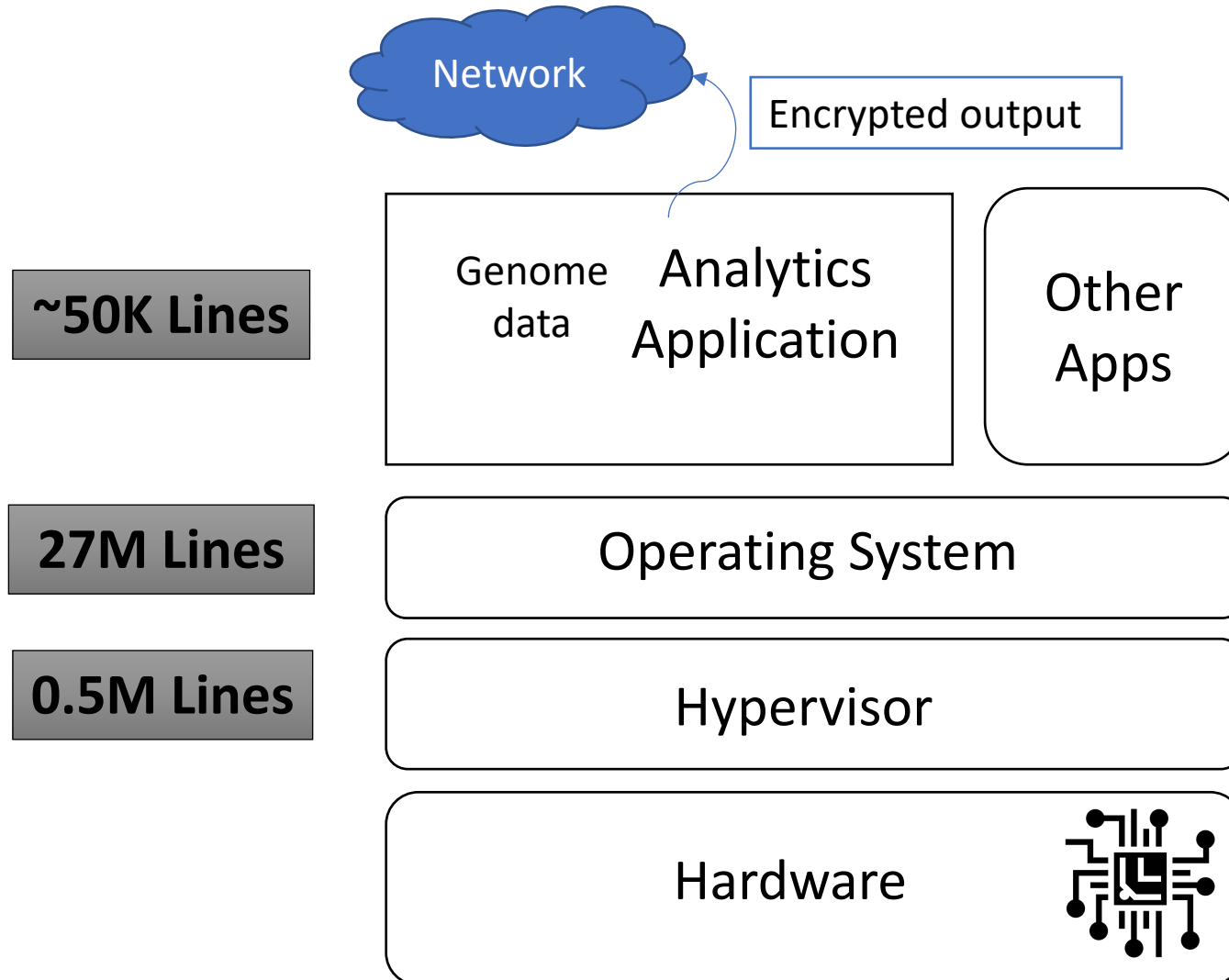
Institute of Information Security
Dept. of Computer Science
ETH Zürich



SECTRS

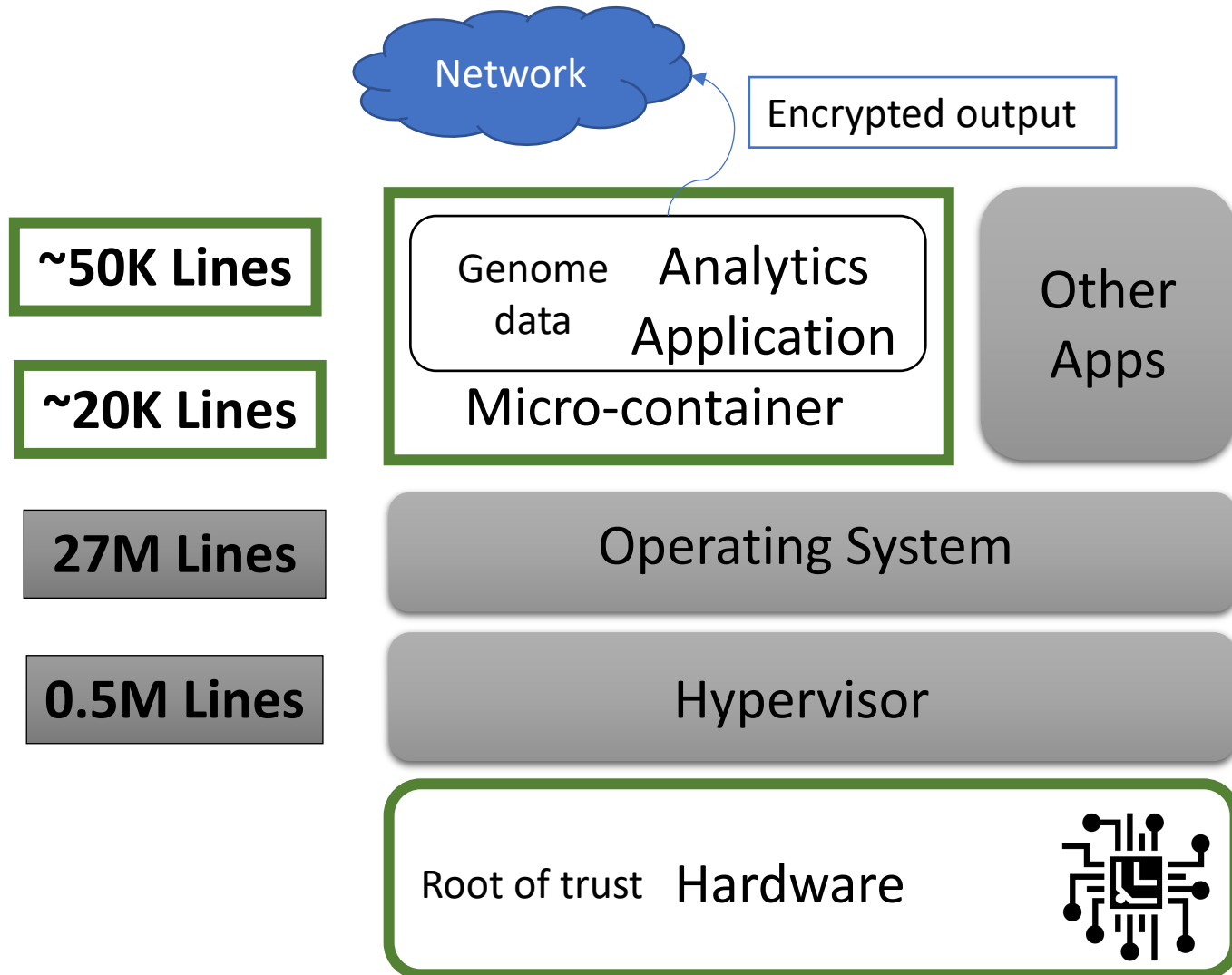
**Secure & Trustworthy
Systems Group**

Current computing stack is prone to attacks



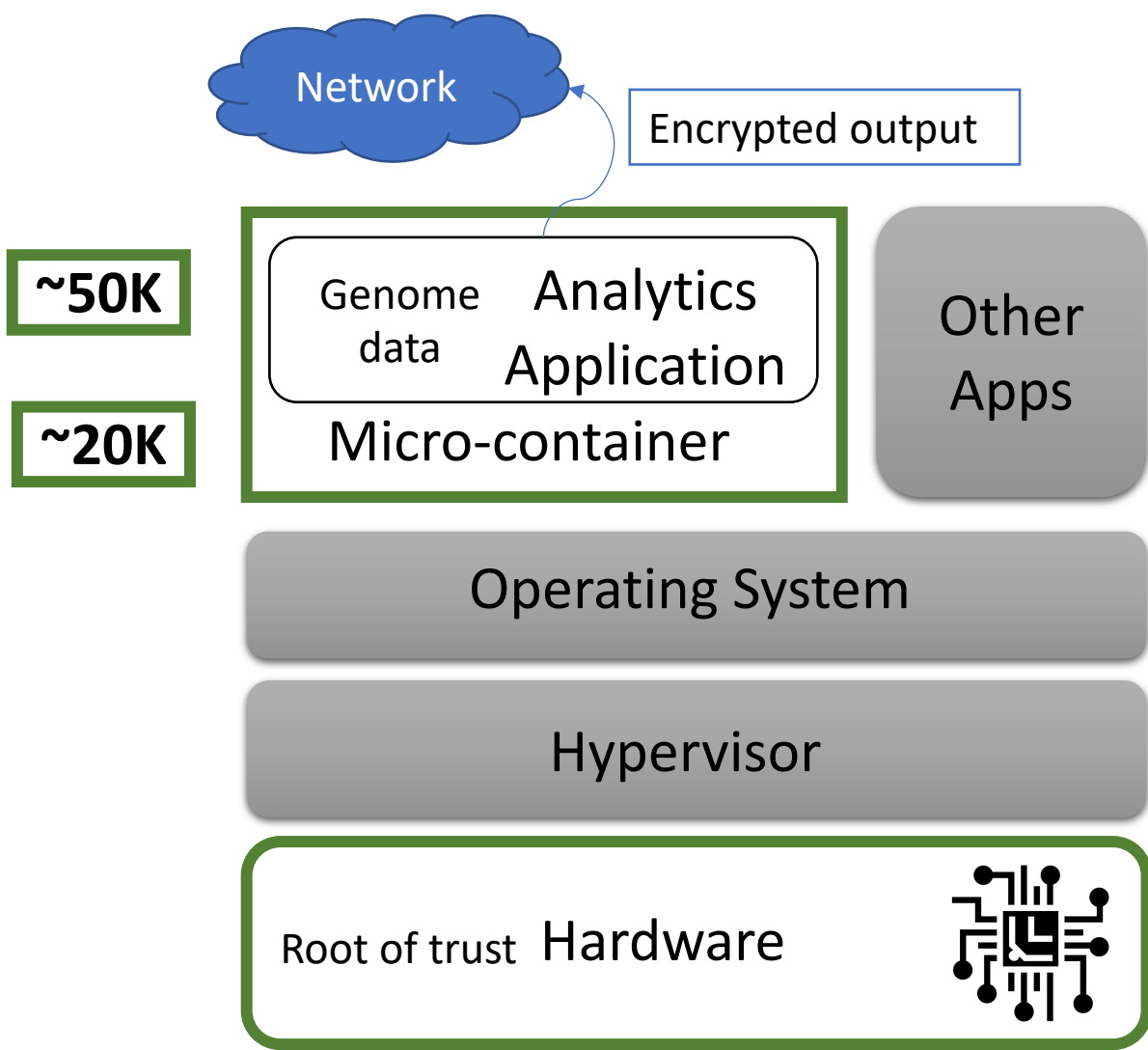
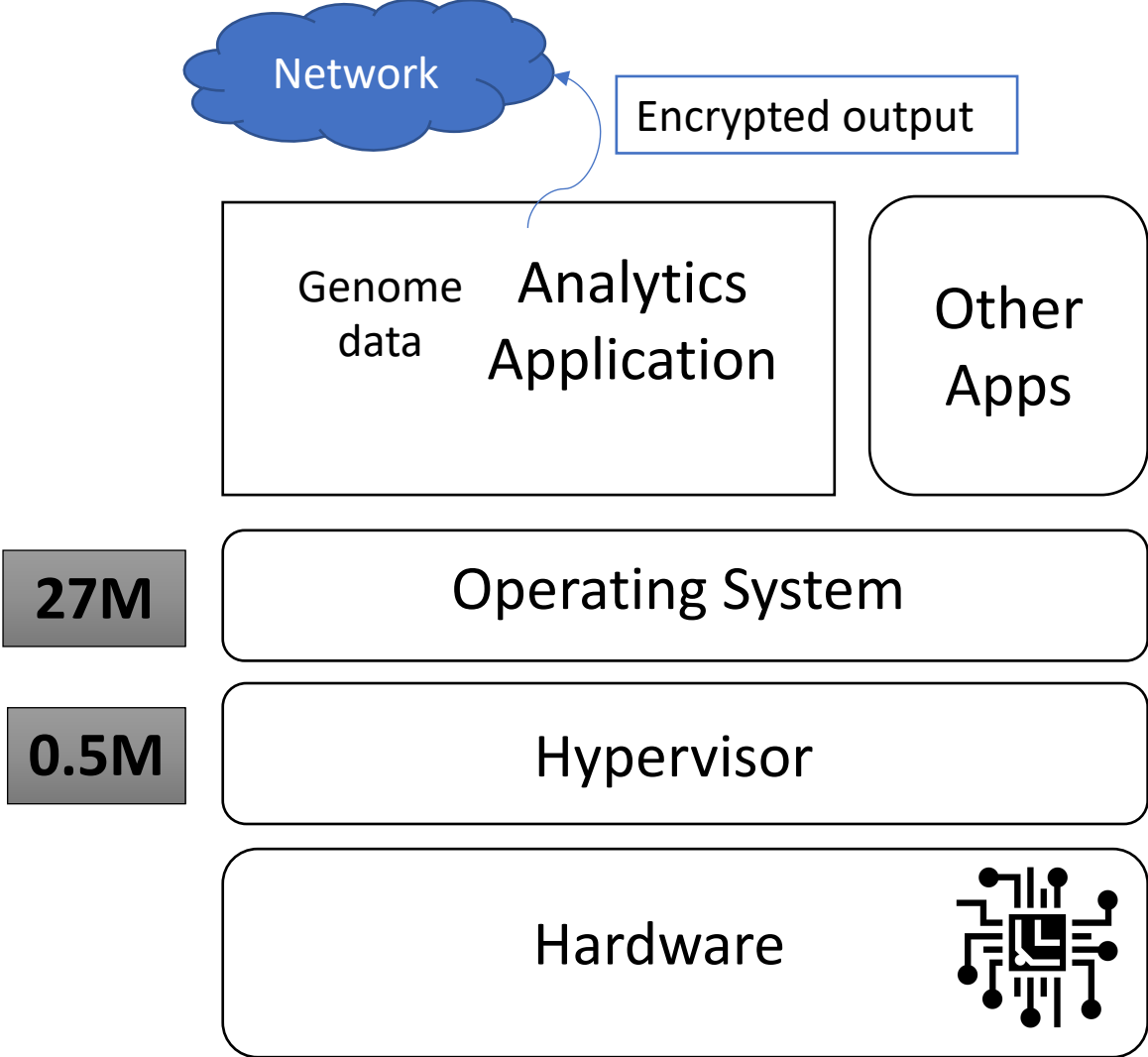
- Encryption or other sophisticated techniques at the application layer
- Bug in lower layers → Compromise the security of the app
- Large size → High probability

Computation stack for the decades to come



- Thin layer for running applications
- Trusted hardware
- Formal guarantees for defense against
 - Third party attacks
 - Internal bugs in the app

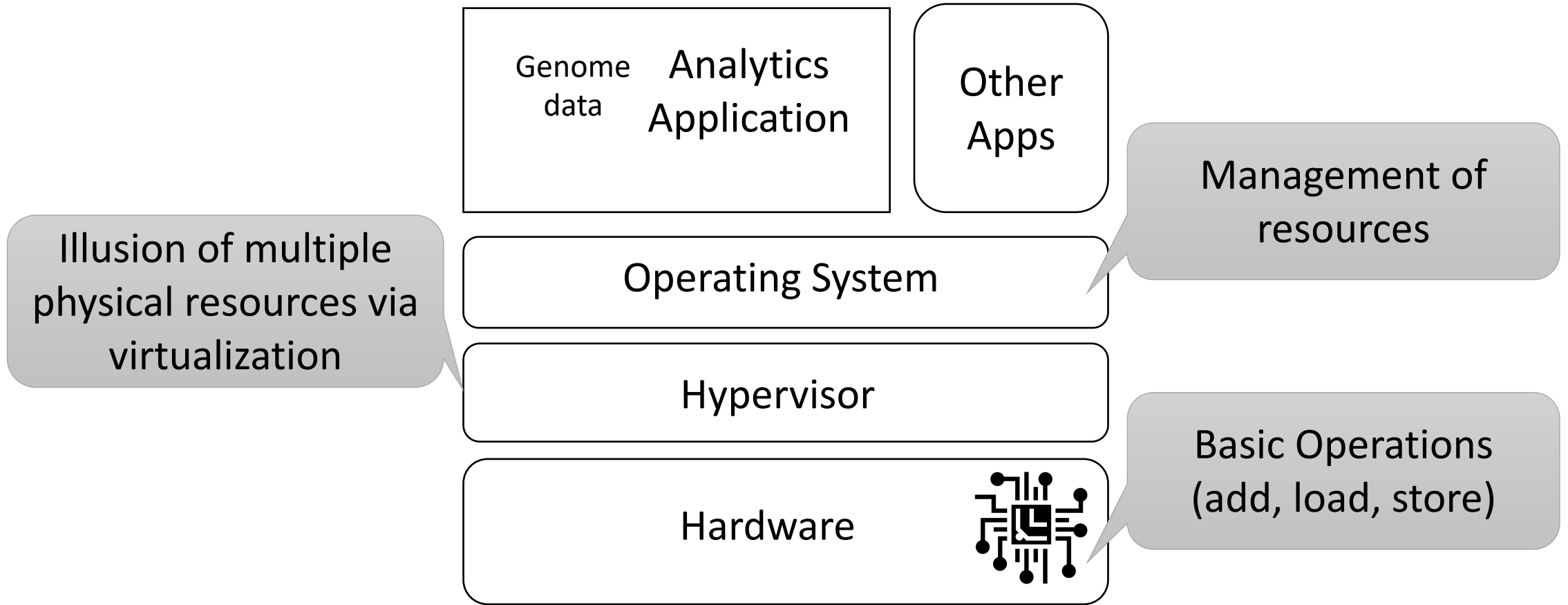
Design Contrast



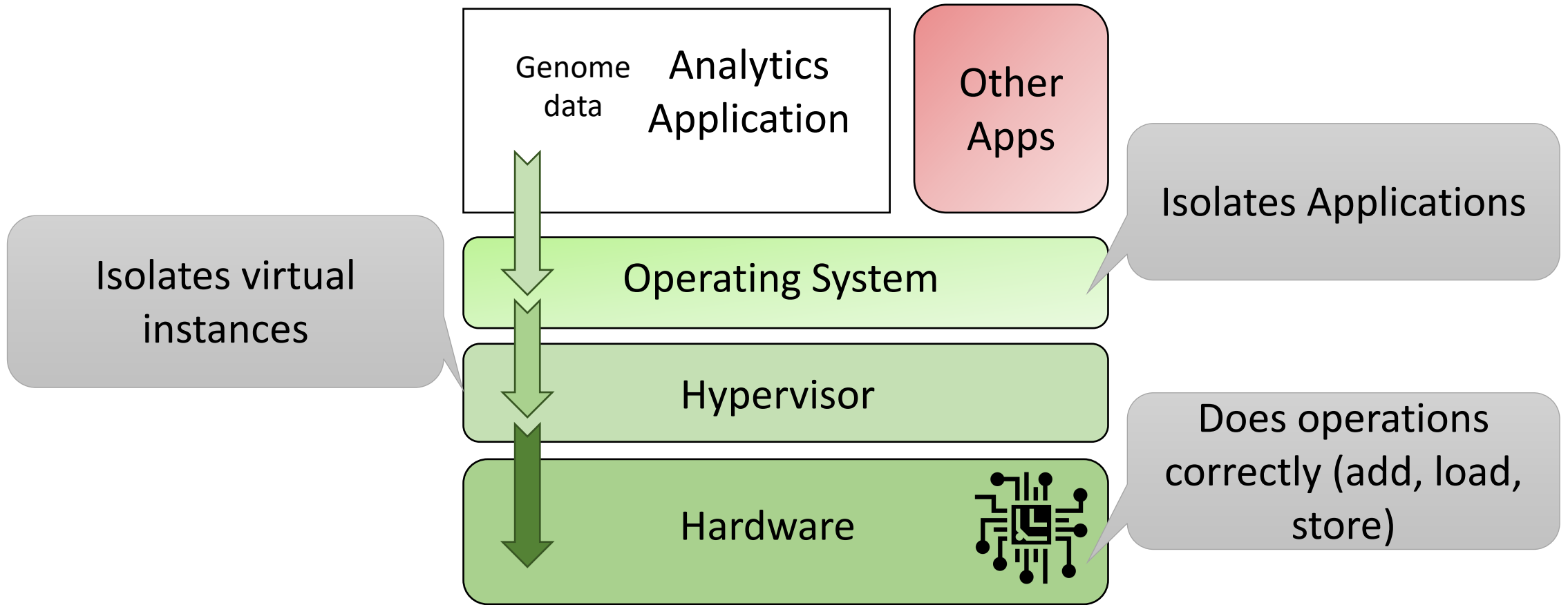
This Talk

- Trusted Execution 101
- Executing legacy applications on existing hardware
 - Delegation
 - Verification
- Building better support for trusted execution

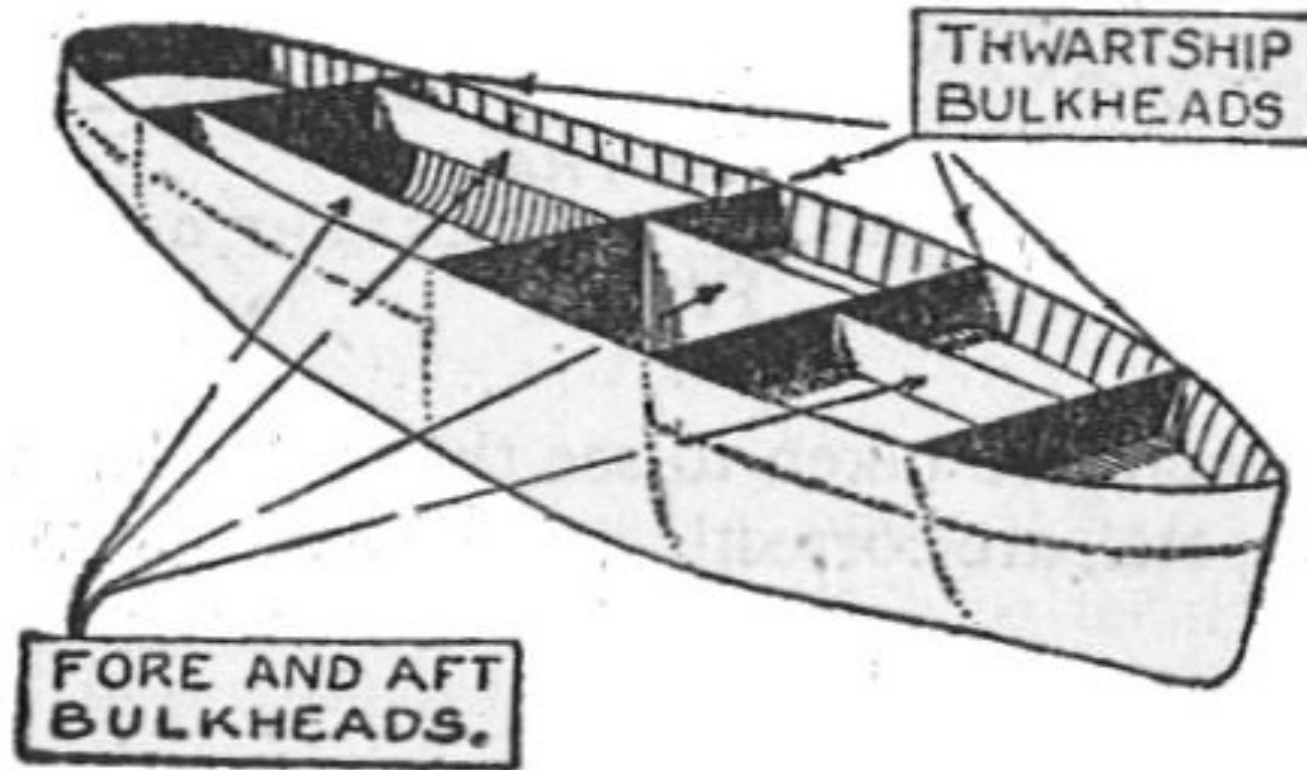
A Brief History of Computer Software Systems



A Brief History of Computer Software Security



Principle of Privilege Separation

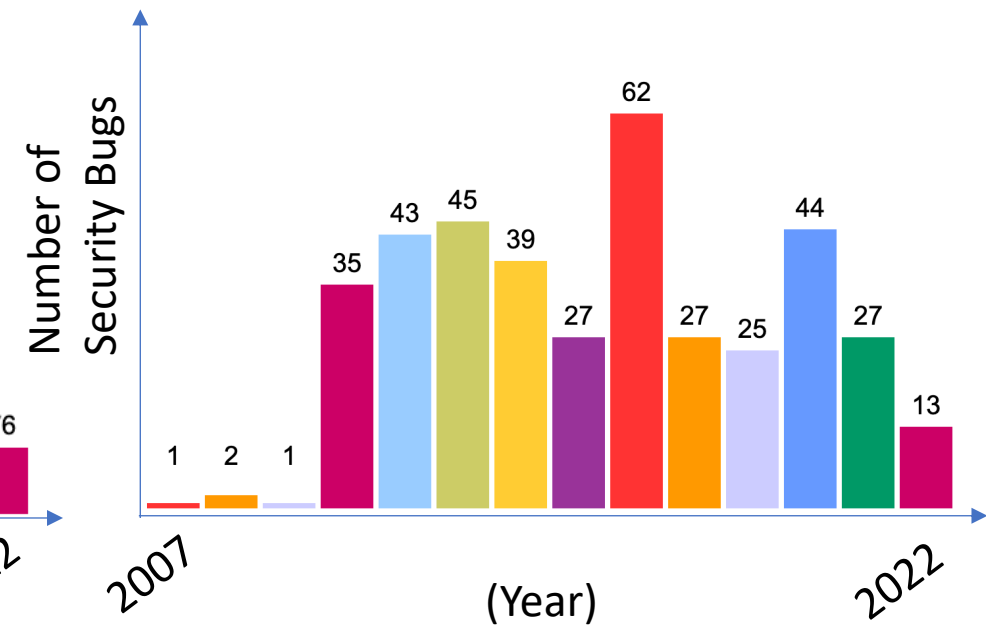
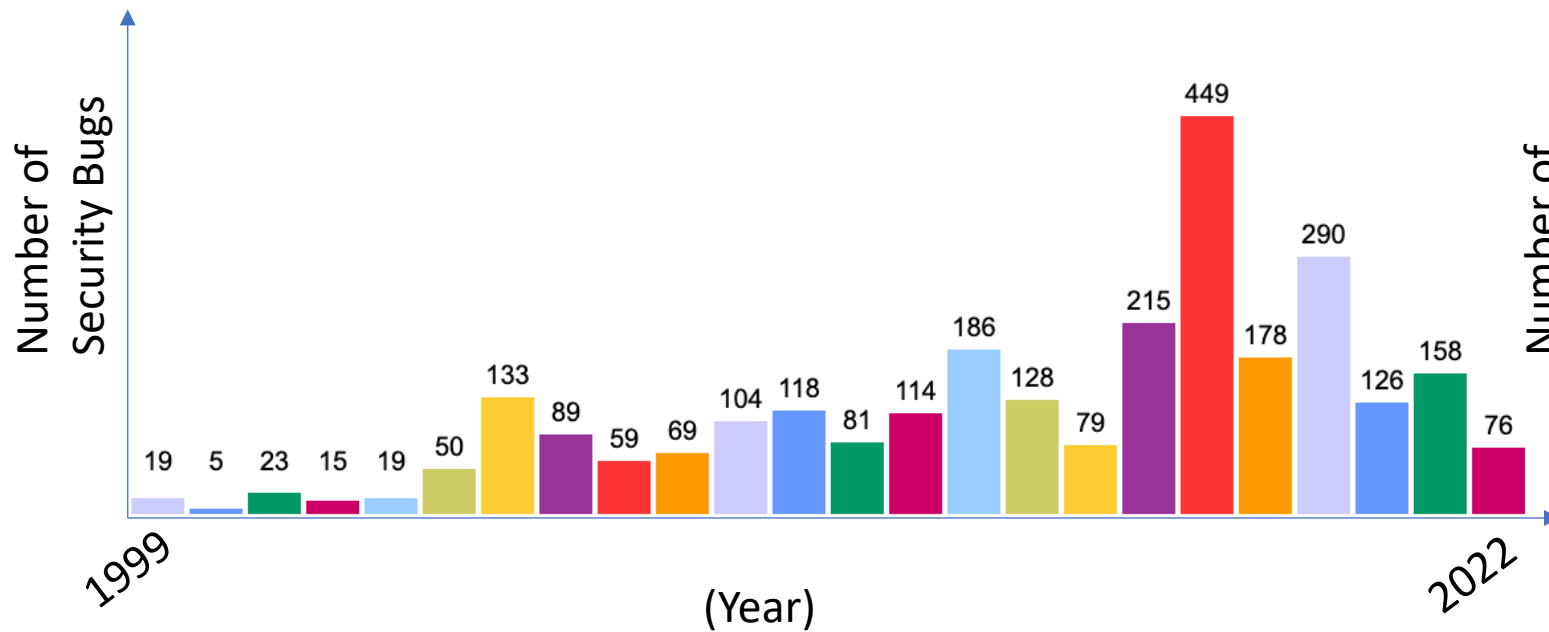


Compartmentalisation of a ship, to reduce floodability

Software-based Separation is Bug-prone

Operating System (Linux Kernel)
27 Million Lines

Hypervisor (XEN)
0.5 Million Lines



Part 1: Trusted Execution Environments 101

Threat Model

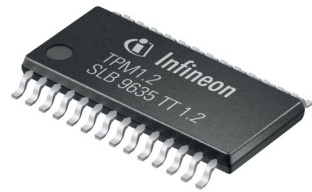
Assumptions

- Trusted Hardware
 - Manufacturer
 - Design & Implementation
- No side-channels (hw & sw)
- No bugs in TEE-bound code
- Secure crypto primitives

Attacker Capabilities

- Cloud Service Provider
 - Admin access
 - Physical access
- Untrusted privileged software
 - OS, Hypervisor
- Peripherals

Evolution of Trusted Computing



ARM TRUSTZONE



2004

2005

2006

2014

2016

2022

2023

TPM 1.2

ARM TrustZone

Intel TXT

TPM 2.0

Intel SGX

NVIDIA CC

Intel TDX
ARM CCA

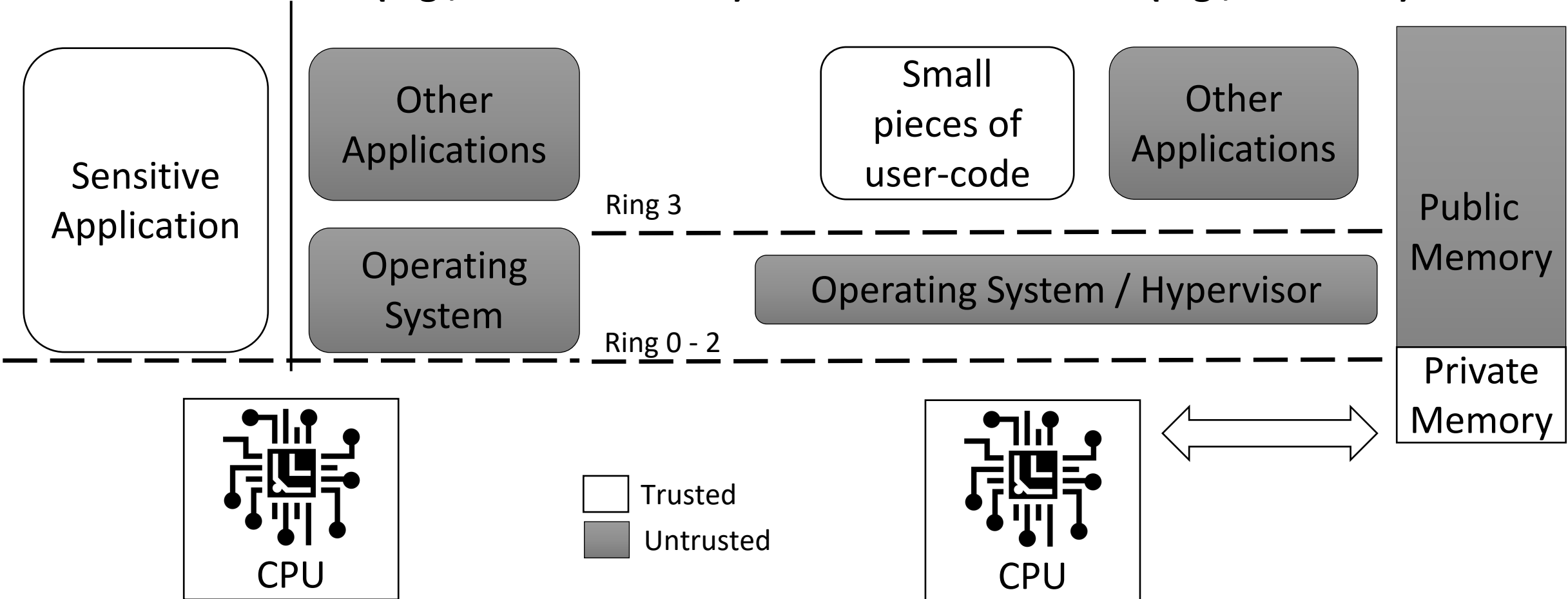
Trusted Computing Primitives

- Remote Attestation
- Trusted Boot
- Sealed Storage
- Isolated Execution

Modern Trusted Execution Environments

Earlier Generation (e.g., ARM TrustZone)

Current Generation (e.g., Intel SGX)



Check our recent SoK for more

SoK: Hardware-supported Trusted Execution Environments

Moritz Schneider
ETH Zurich

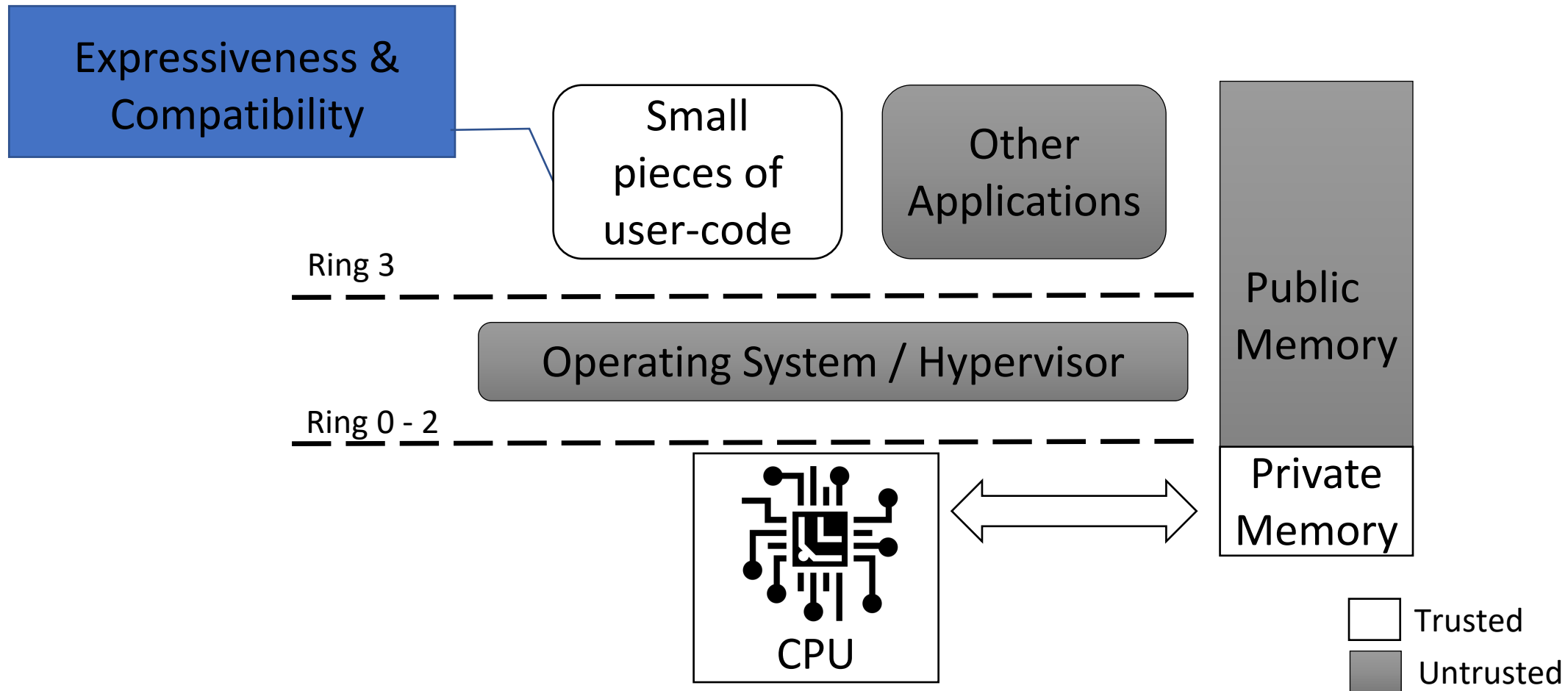
Ramya Jayaram Masti
Intel Cooperation

Shweta Shinde
ETH Zurich

Srdjan Capkun
ETH Zurich

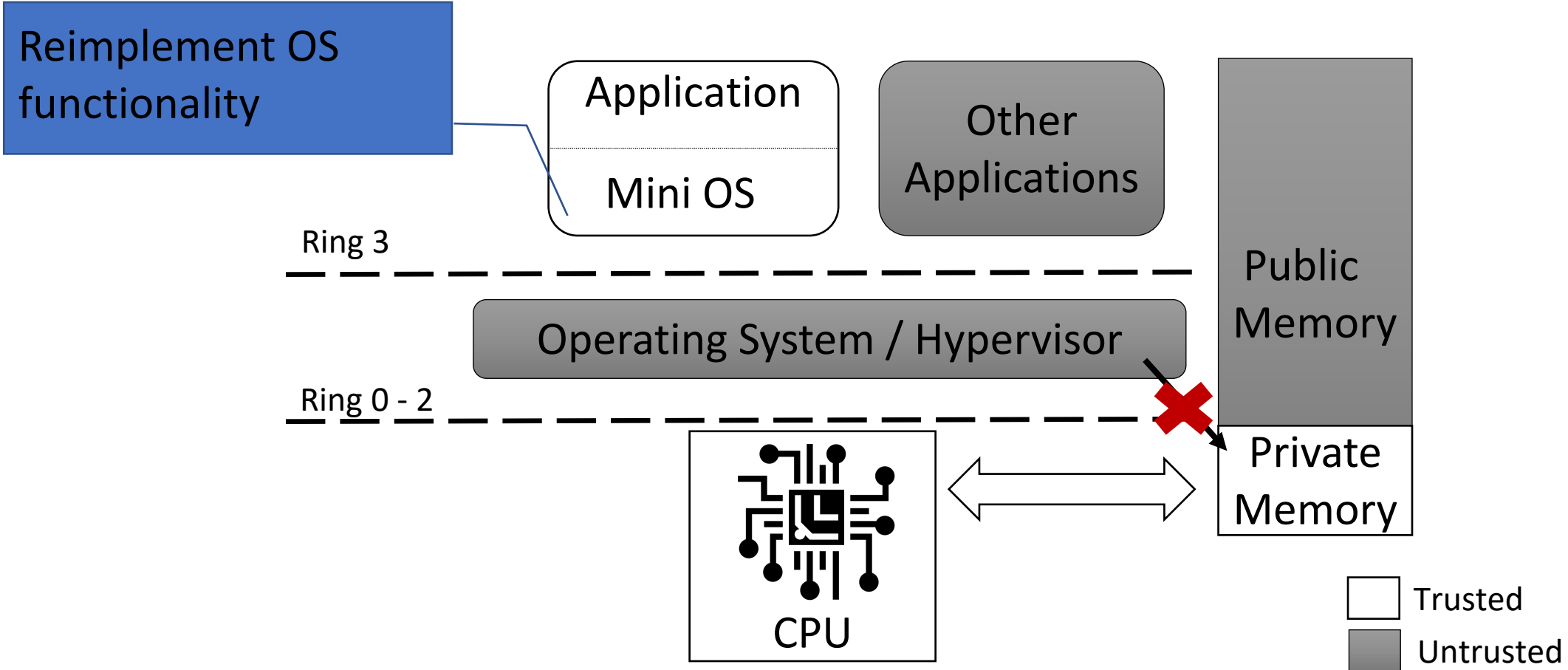
Ronald Perez
Intel Cooperation

Limitations of Intel SGX

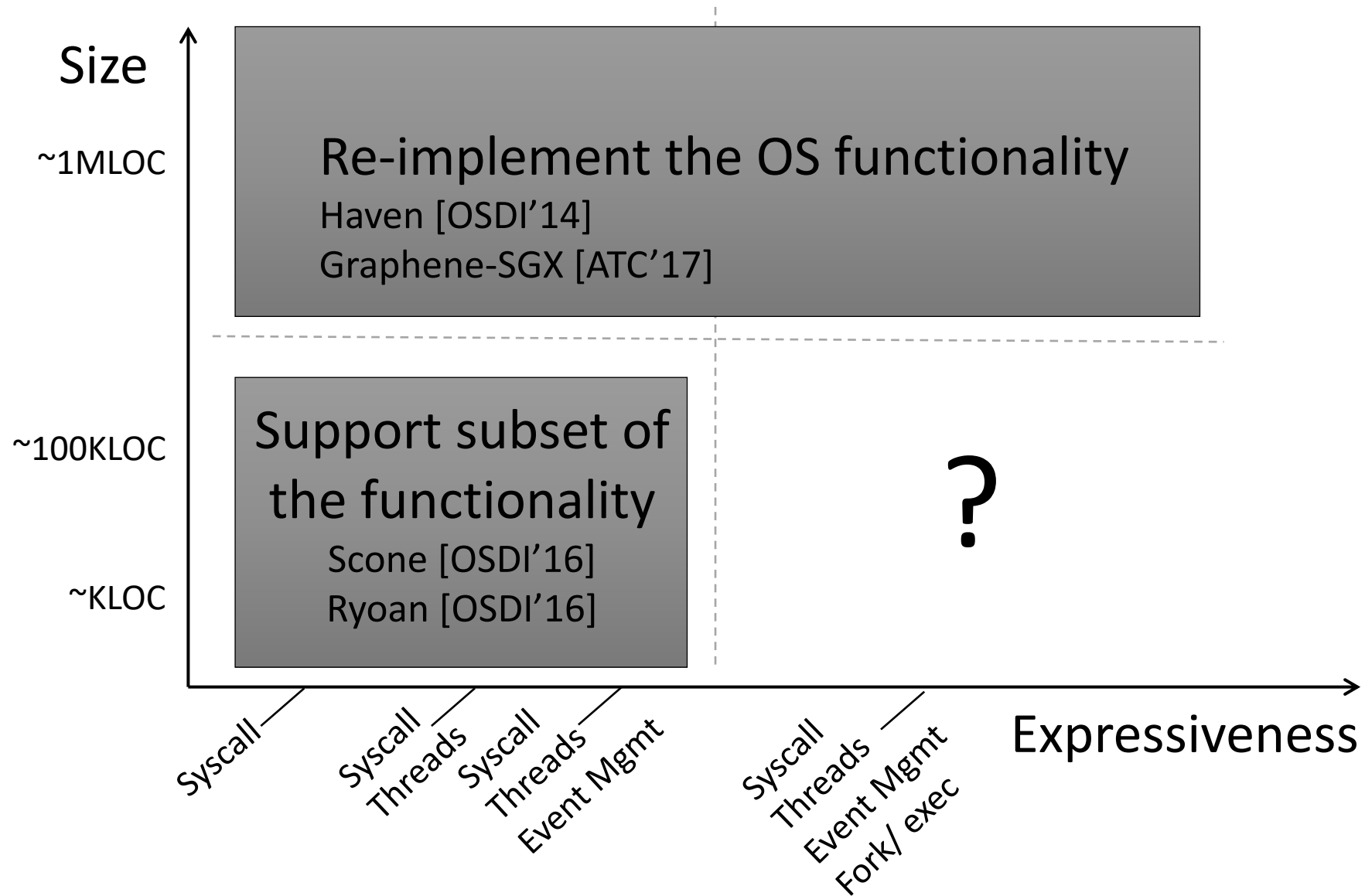


Part 2: Executing Legacy Applications on Intel SGX

Adding Expressiveness to Commercial TEEs

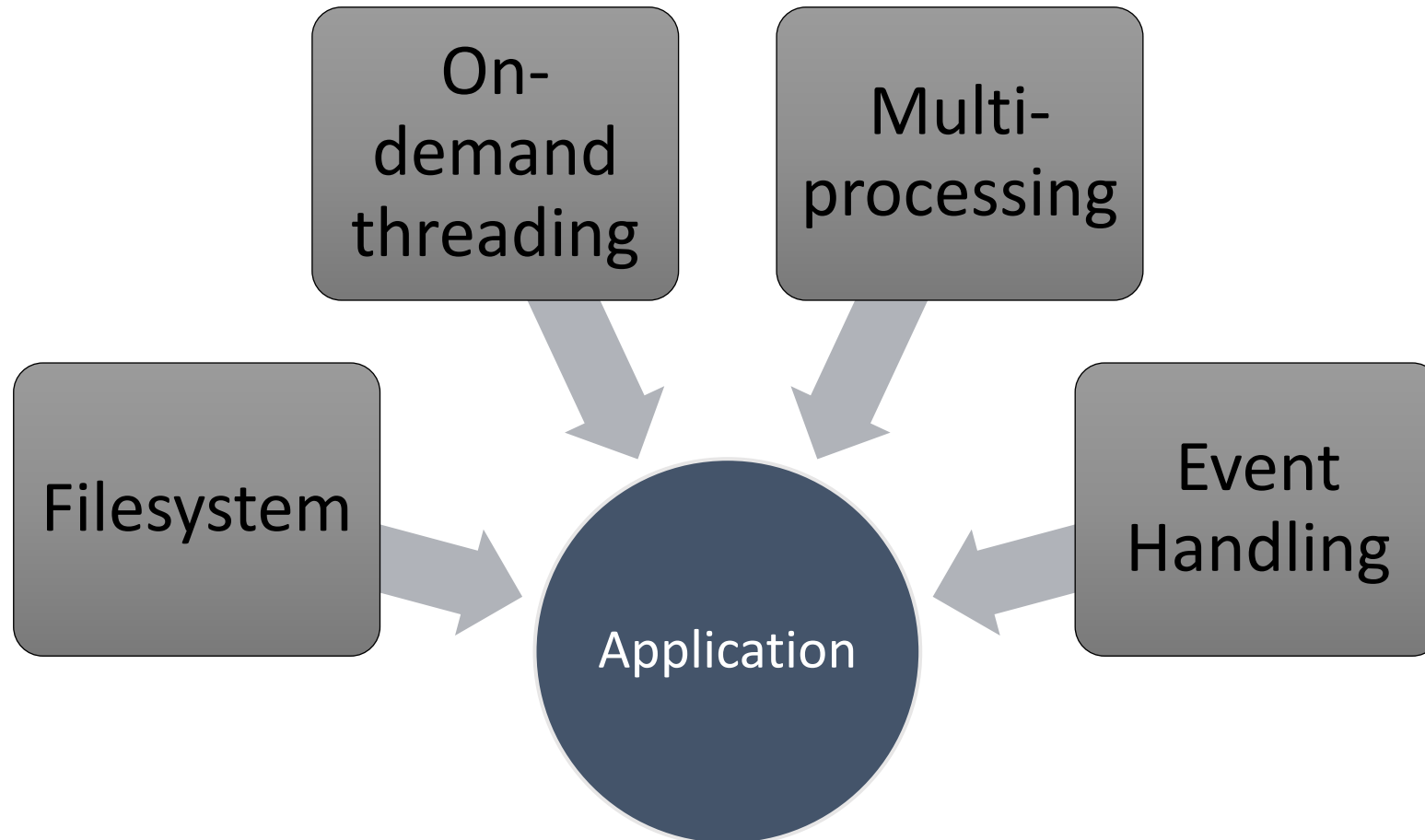


Code Size & Expressiveness Trade-off

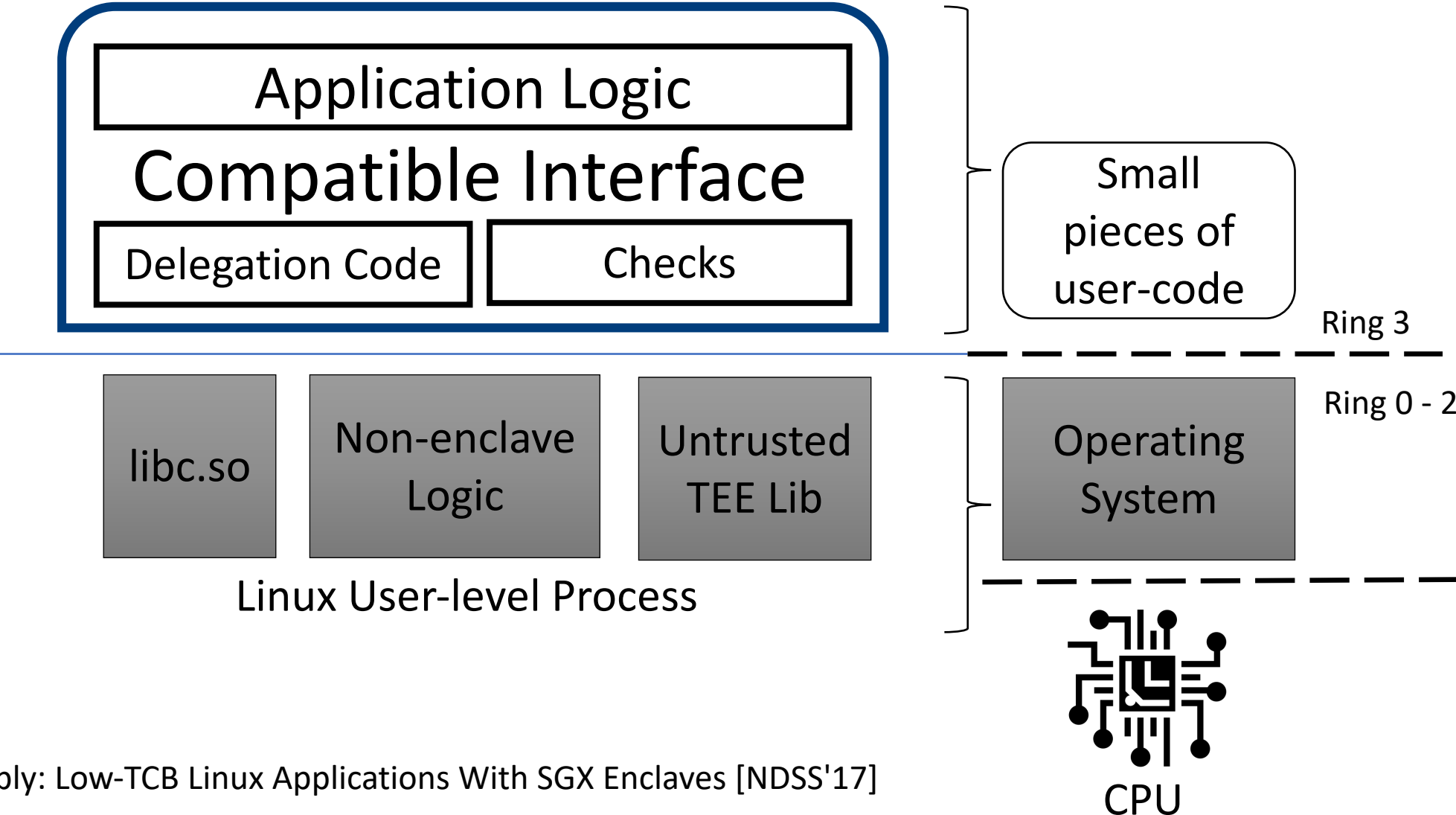


Challenge I: Expressiveness

Delegate rather than emulate

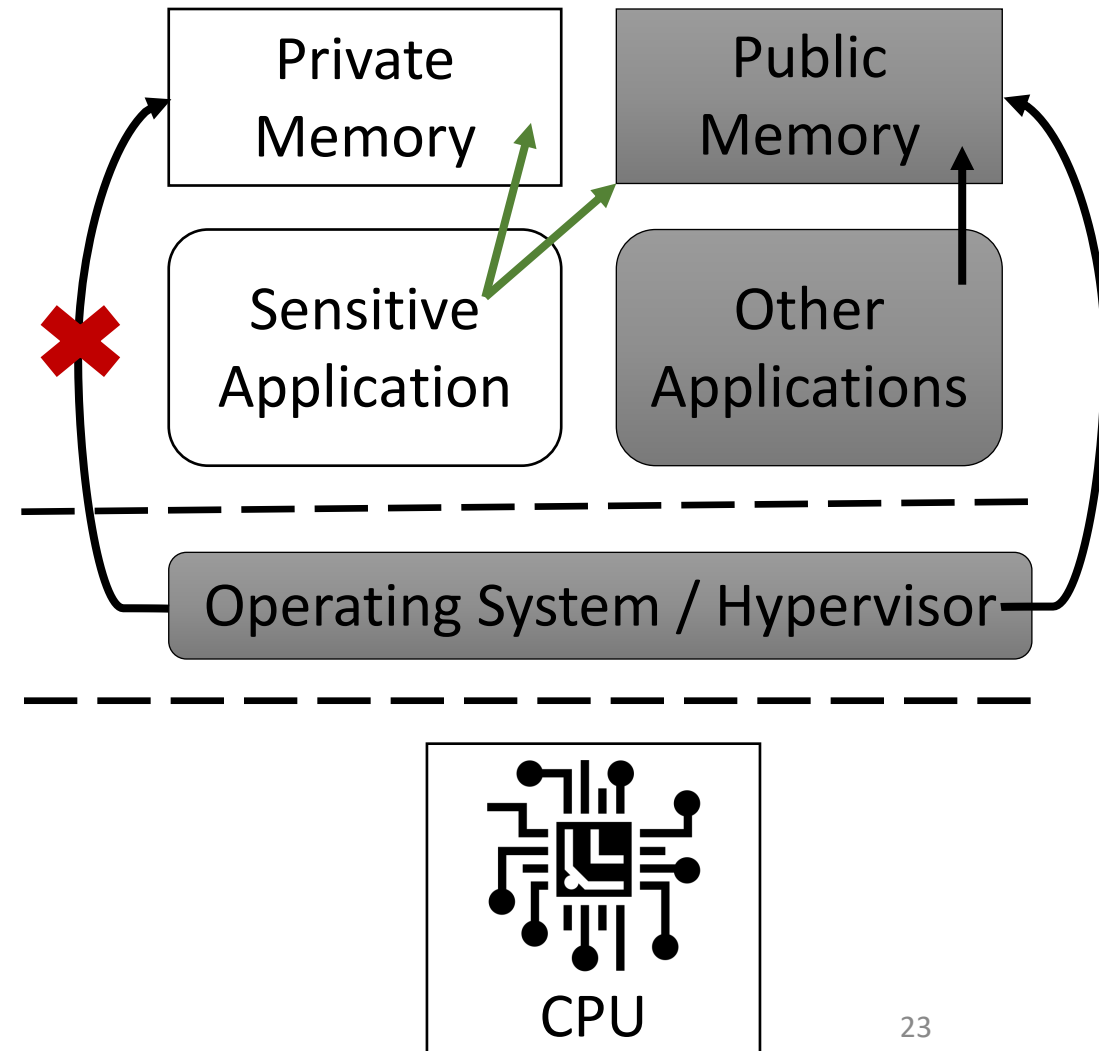


Building micro-container abstractions for TEEs



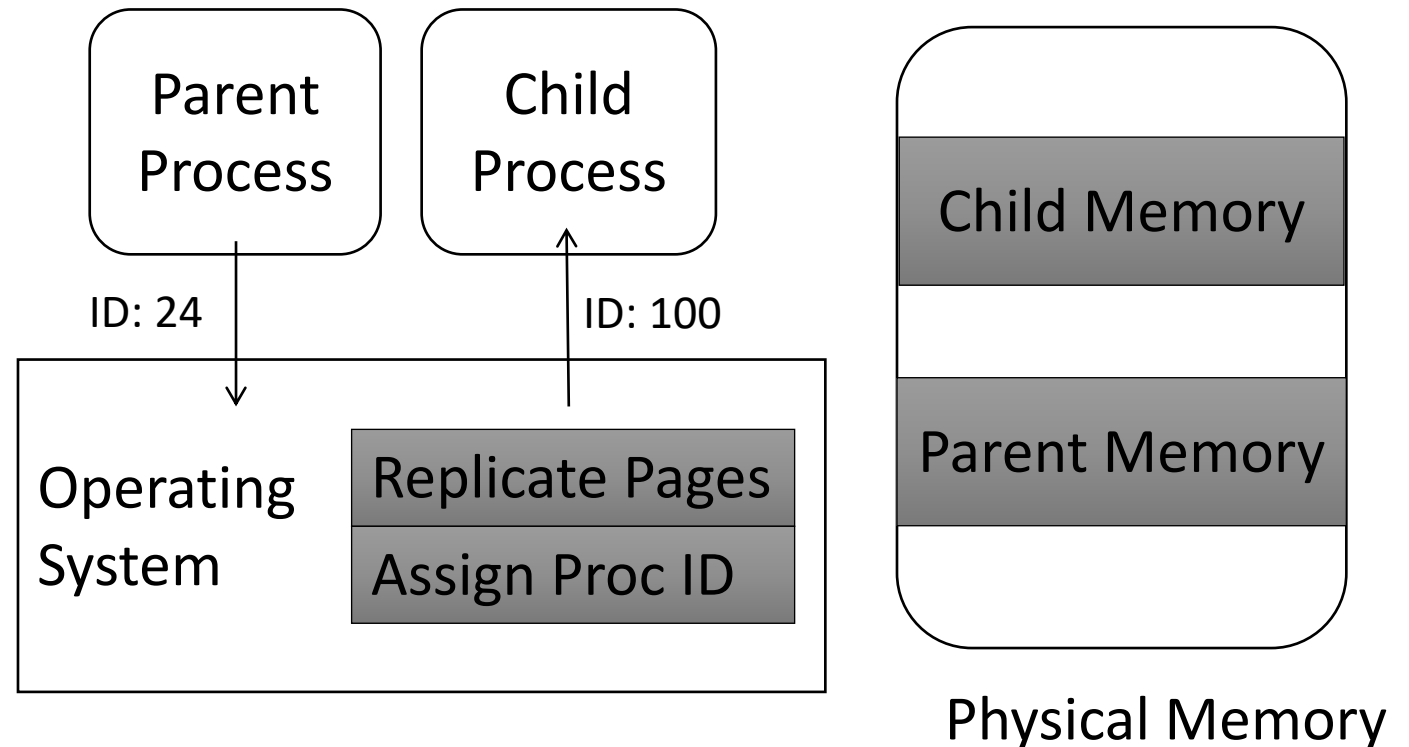
Challenge II: Delegation with isolation

- Two memory model:
 - private and public memory
- Process abstraction breaks
 - locks are in public memory
 - shared memory for processes
 - passing data to system calls



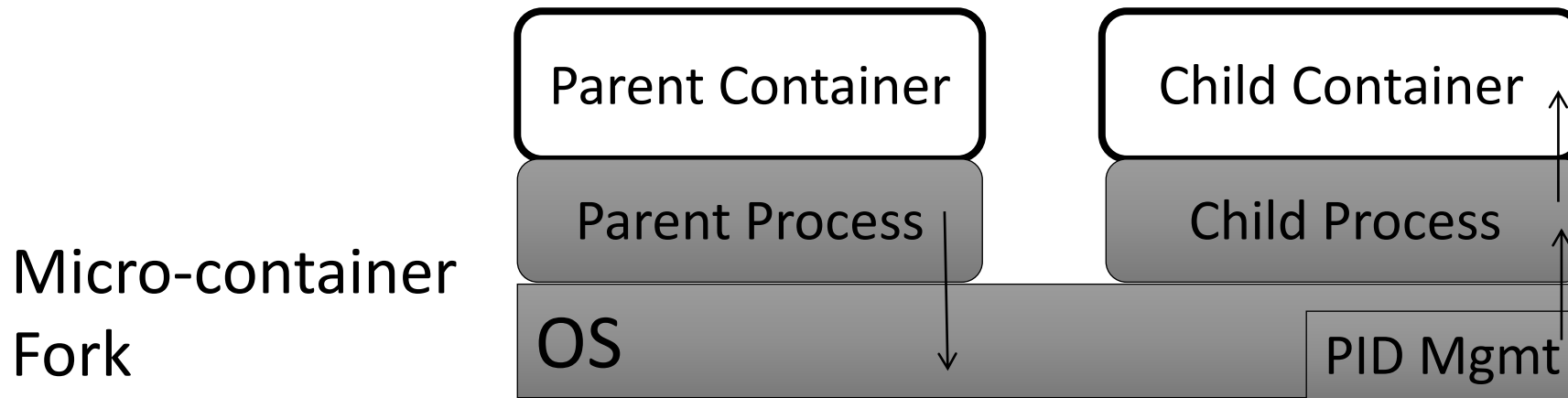
Expressiveness Example: Fork

- Fork Semantics:
 - Assigns new process id
 - Makes a memory replica
- How to maintain fork semantics if the OS cannot access private memory?



Expressiveness Example: Delegating Fork

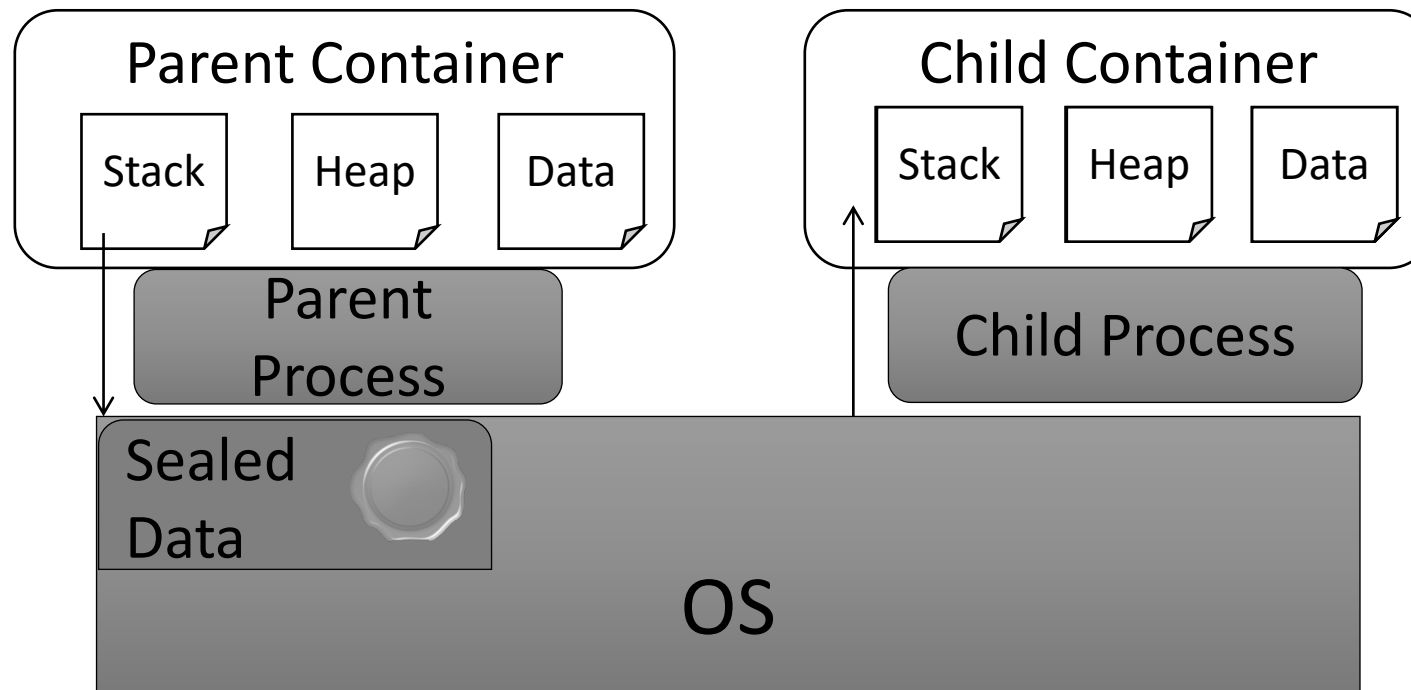
- Creating child process and child micro-container



- Child enclave has a clean memory state

Expressiveness Example: Achieving Fork Semantics

- Mirroring parent's memory in child micro-container
 - After the fork call, before resuming execution



Expressiveness: Supporting POSIX APIs

Core Services

Process Creation and Control	5
Signals	6
Timers	5
File and Directory Operations	37
Pipes	4
C Library (Standard C)	66
I/O Port Interface and Control	40

Thread Extensions

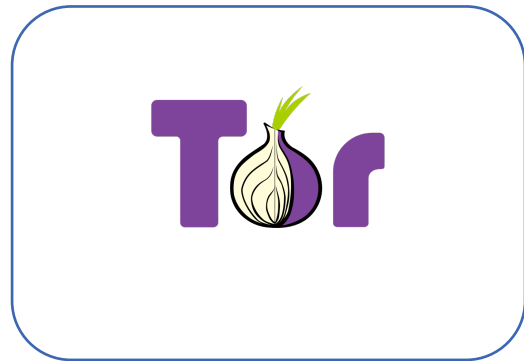
Thread Creation, Control, and Cleanup	17
Thread Scheduling	4
Thread Synchronization	10
Signal Delivery	2
Signal Handling	3

Real-time Extensions

Real-Time Signals	4
Clocks and Timers	1
Semaphores	2
Message Passing	7
Shared Memory	6
Asynchronous and Synchronous I/O	29
Memory Locking Interface	6

**POSIX APIs
Supported for
Commodity Linux Apps**

Micro-containers execute TEE use-cases



ANONYMITY
PROTOCOLS



WEBSERVERS



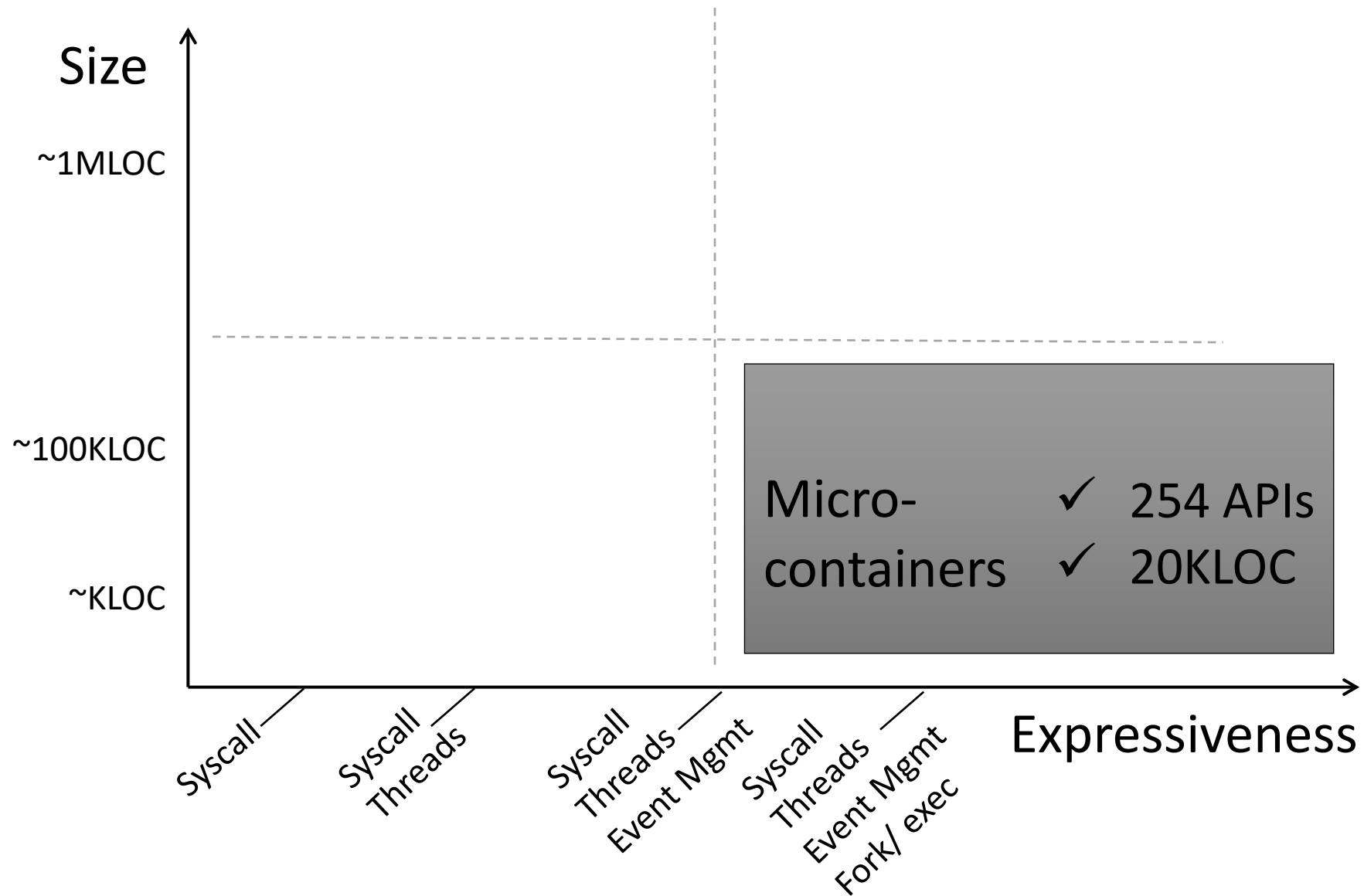
DATABASE
CLIENTS



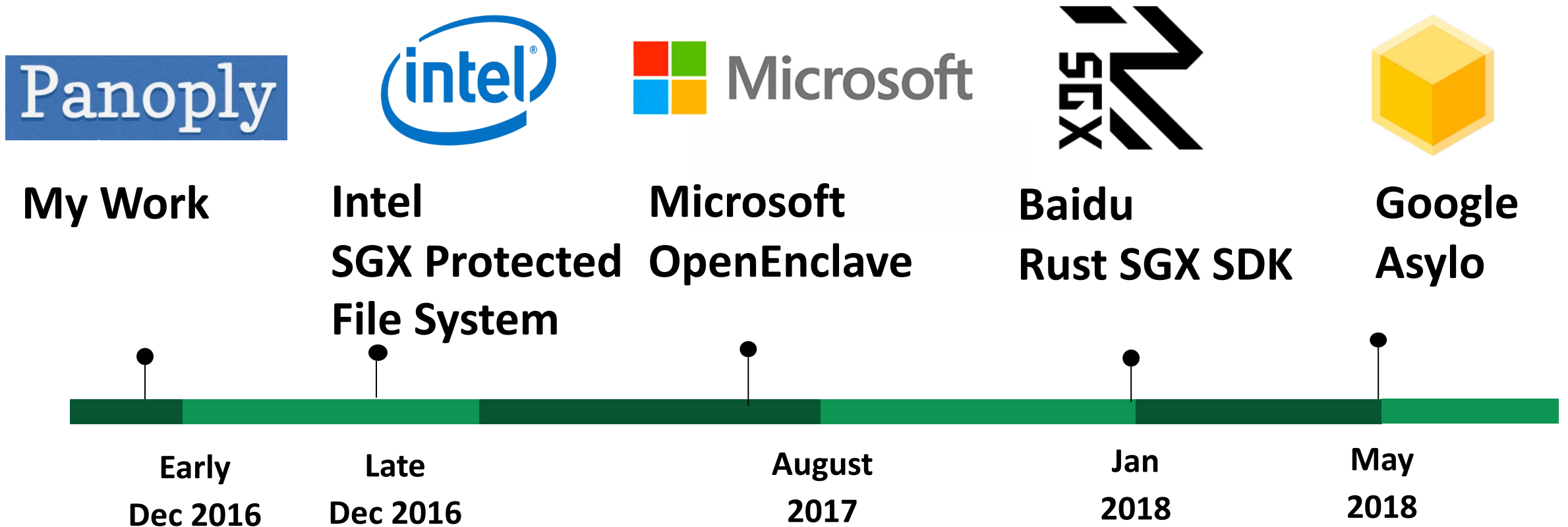
CRYPTOGRAPHIC
LIBRARIES

Performance is comparable to
importing a mini-OS

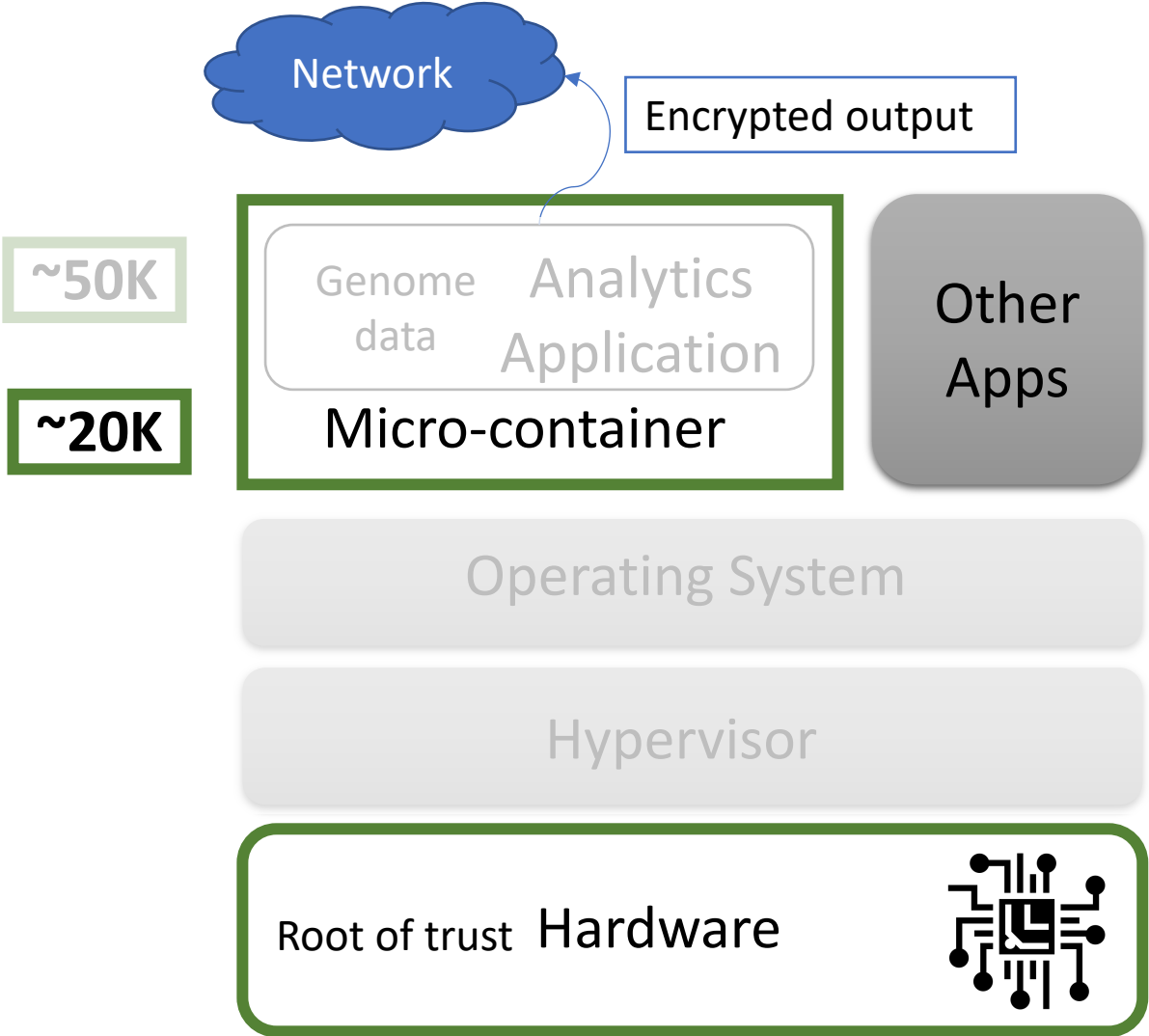
Minimize Trust to 20,000 lines of code



Adoption of the Delegation Approach



Trust, but Verify



Example: What is the damage via OS interface?

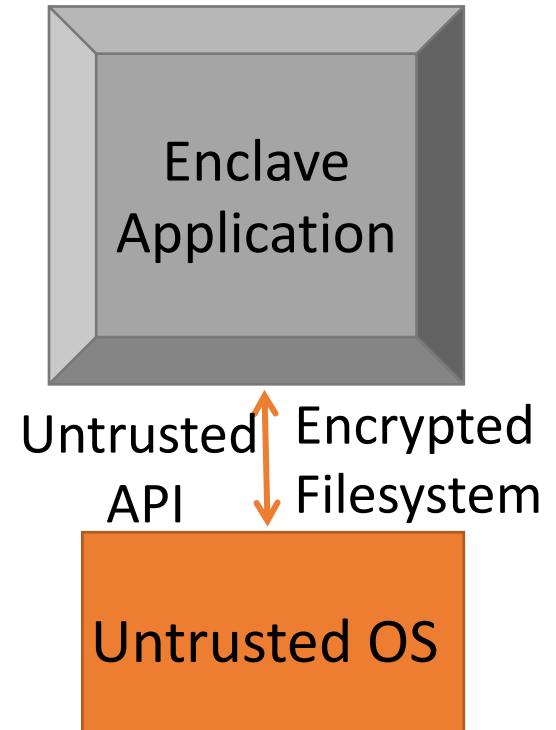
```
1 FILE* fd = fopen(fname, mode);
2 if (fd == NULL) {
3     errnum = errno;
4     if (errnum == EINVAL)
5         fd = fopen (fname, "a");
6     if (errnum == ENOENT)
7         if (fname == NULL)
8             fname = "vote.log";
9             fd = create_log(fname);
10    if (errnum == EINTR)
11        fd = fopen(fname, mode);
12 }
13 if (fd)
14     cnt = fwrite(buf, 1, len, fd);
15 return cnt;
```

Open the vote file

Failed to open the
vote file

Create new file →
overwrite previous vote

Register the vote sequence



Attacks are possible in delegation frameworks

```
9 int enc_untrusted_open(const char *path_name, int flags) {
10  uint32_t mode = 0;
11  int result;
12  sgx_status_t status = ocall_enc_untrusted_open(&result,
13  path_name, flags, mode);
14  if (status != SGX_SUCCESS) {
15    errno = EINTR;
16    return -1;
17  }
18  return result;
19 }
```

fopen: Google Asylo

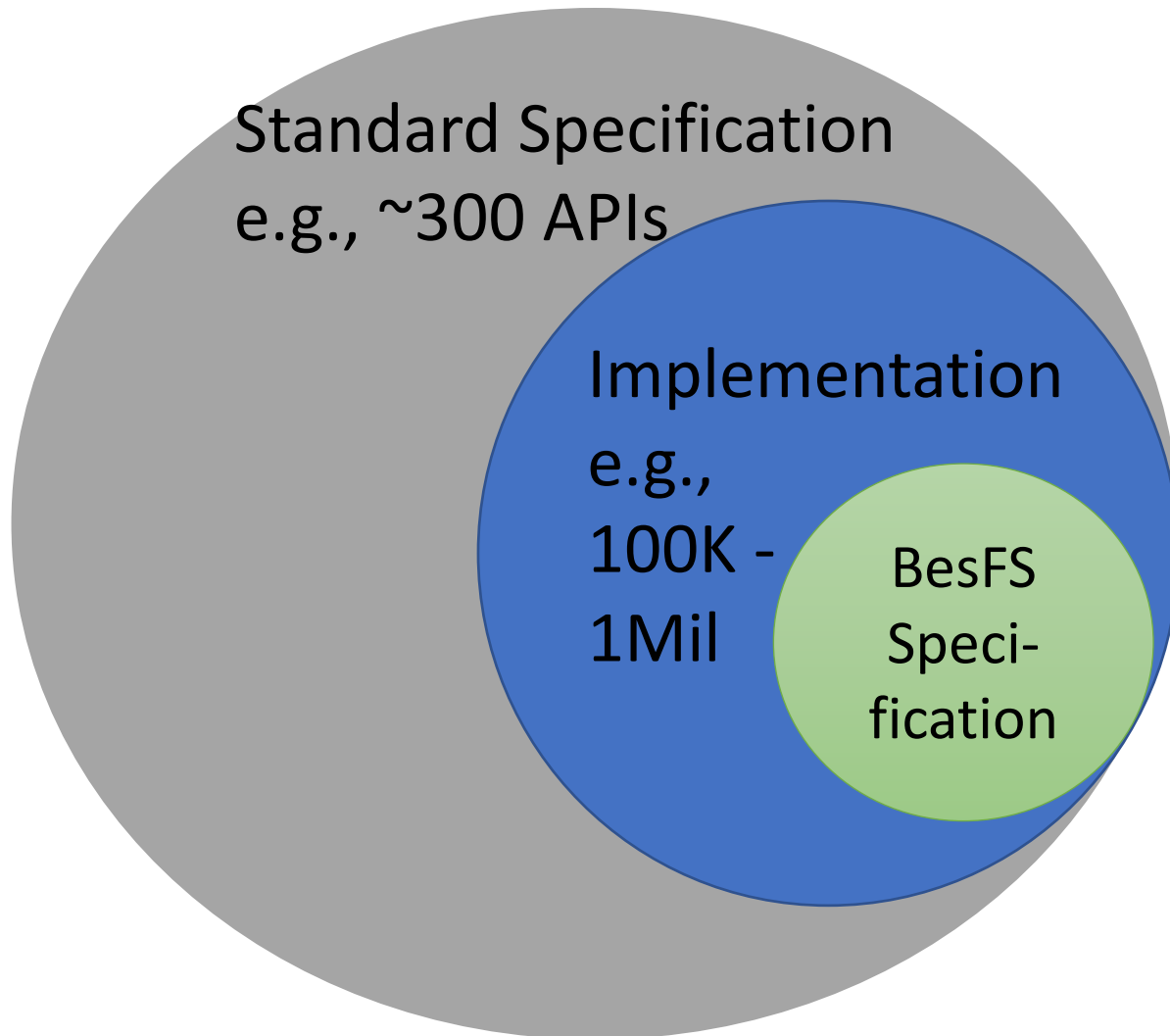
```
7 static int sgx_ocall_open(void * pms) {
8  ms_ocall_open_t * ms = (ms_ocall_open_t *) pms;
9  int ret;
10  ODEBUB(OCALL_OPEN, ms);
11  ret = INLINE_SYSCALL(open, ...);
12  return IS_ERR(ret)?unix_to_pal_error(ERRNO(ret)):ret;
13 }
```

fopen: Graphene-SGX

```
4 }
5 static SGX_FILE* sgx_fopen_internal
6 (const char* filename, const char* mode) {
7  protected_fs_file* file = NULL;
8  if (filename == NULL || mode == NULL) {
9    errno = EINVAL;
10   return NULL;
11  }
12  ...
13 }
```

fopen: Intel SDK

A Formal Verification Approach: How to scale to POSIX?



The scalability challenge:

- Specification for safe behavior for the entire POSIX API
- Proving safe implementation
 - entire libc (glibc, musl)
 - filesystem (ext4)

Designing Scalable Specification: BesFS Interface

- Our Approach
 - 15 core APIs: e.g., `open`, `close`, `read`, `write`
 - Allow to execute any sequence of these while maintaining safety property
- Can be composed to express higher-level interfaces
 - e.g., `fwrite` can be composed with `write` and `fstat`
 - Created 22 auxiliary APIs witnessed in applications

BesFS Highlights



4625 lines in Coq
167 lemmas
($< 1.5K$ in C code)



Not over restrictive
Supports all applications
from Panoply (& more)

Total 31 tested

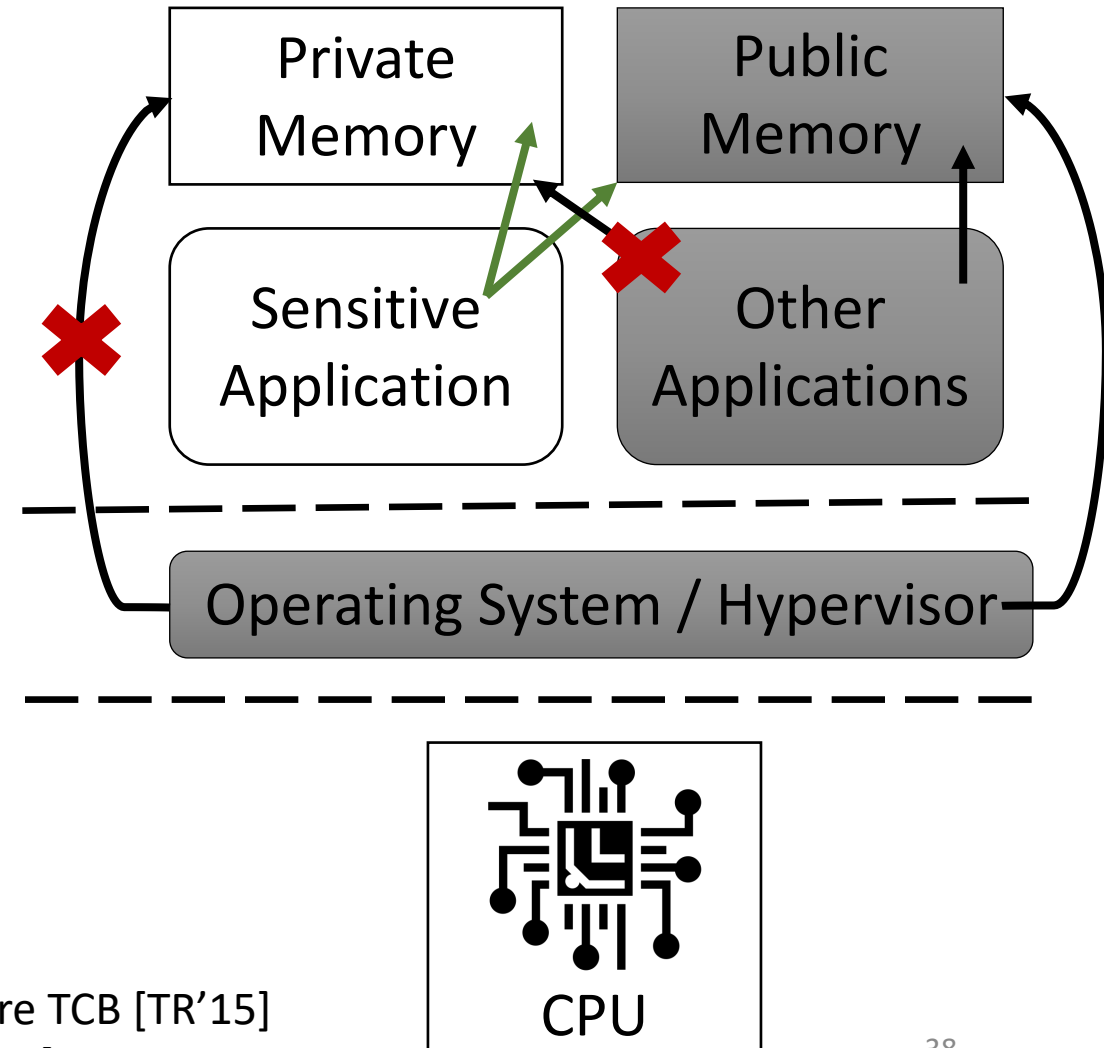


Helped in eliminating
bugs (from Panoply, Intel
SDK, Google SDK)

Part 3: Building Better Trusted Execution Environments

Better TEEs

- Main Observation:
 - Physical memory isolation
 - Simpler ways to achieve
- Similar abstraction to Intel SGX
- Novelty: Designed to maintain
 - Compatibility
 - Performance



Building the next-generation TEEs



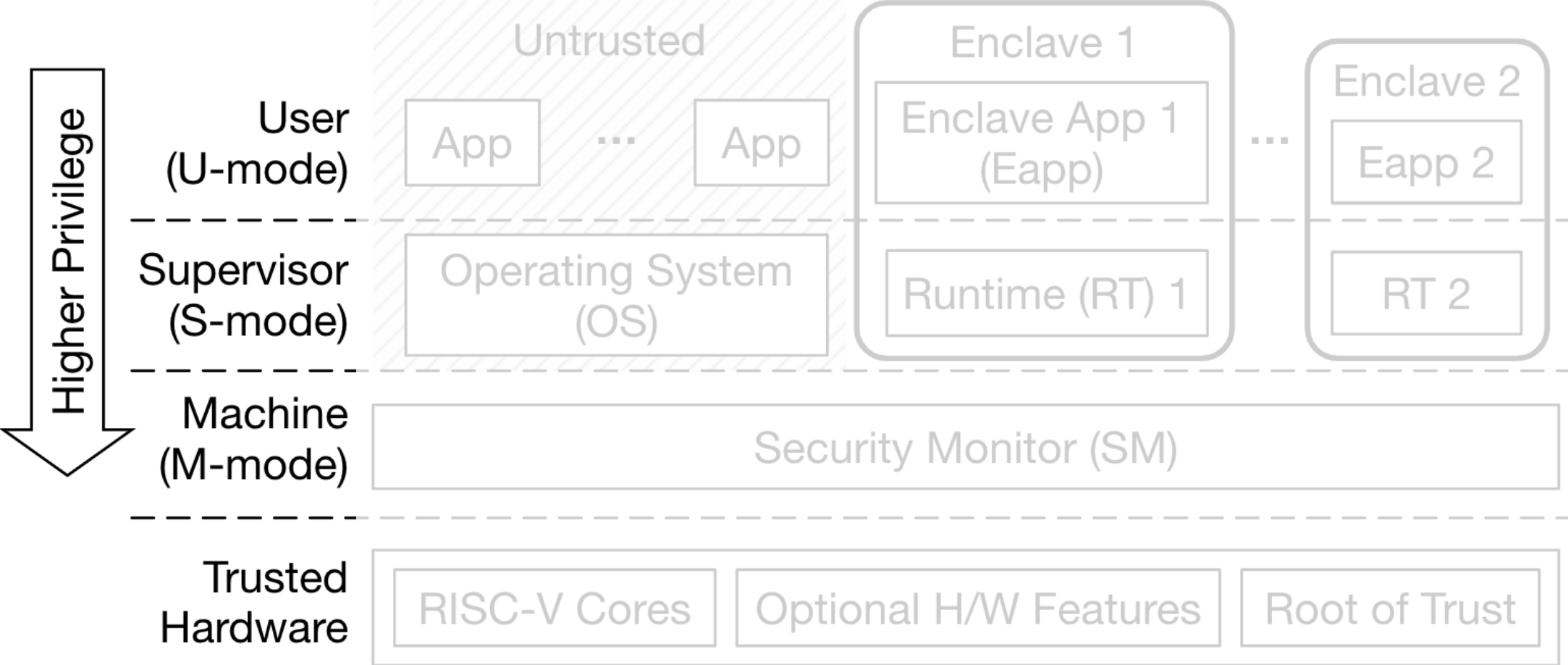
Keystone

A software framework for TEEs on RISC-V

No micro-architectural changes

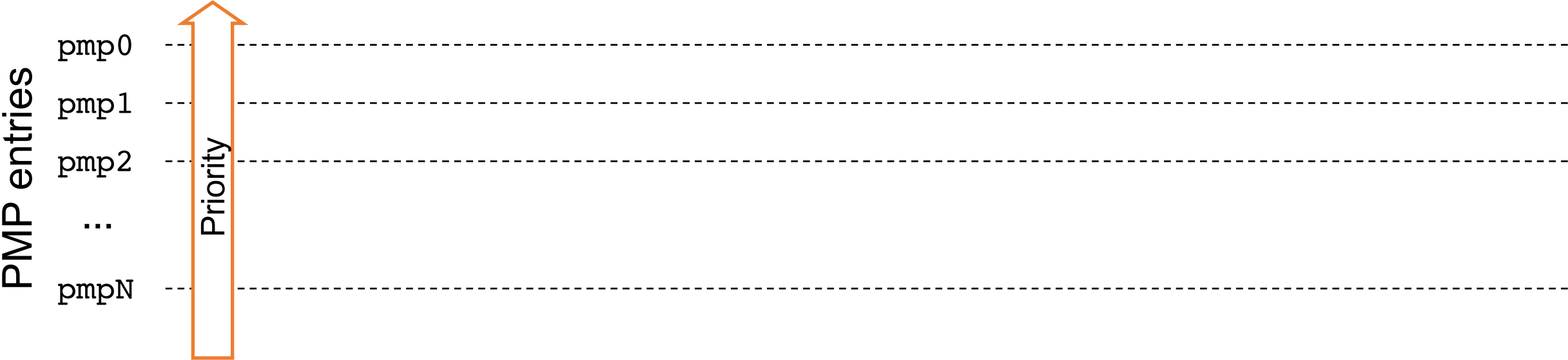
Minimal added hardware

Keystone: Architecture and Trust Model

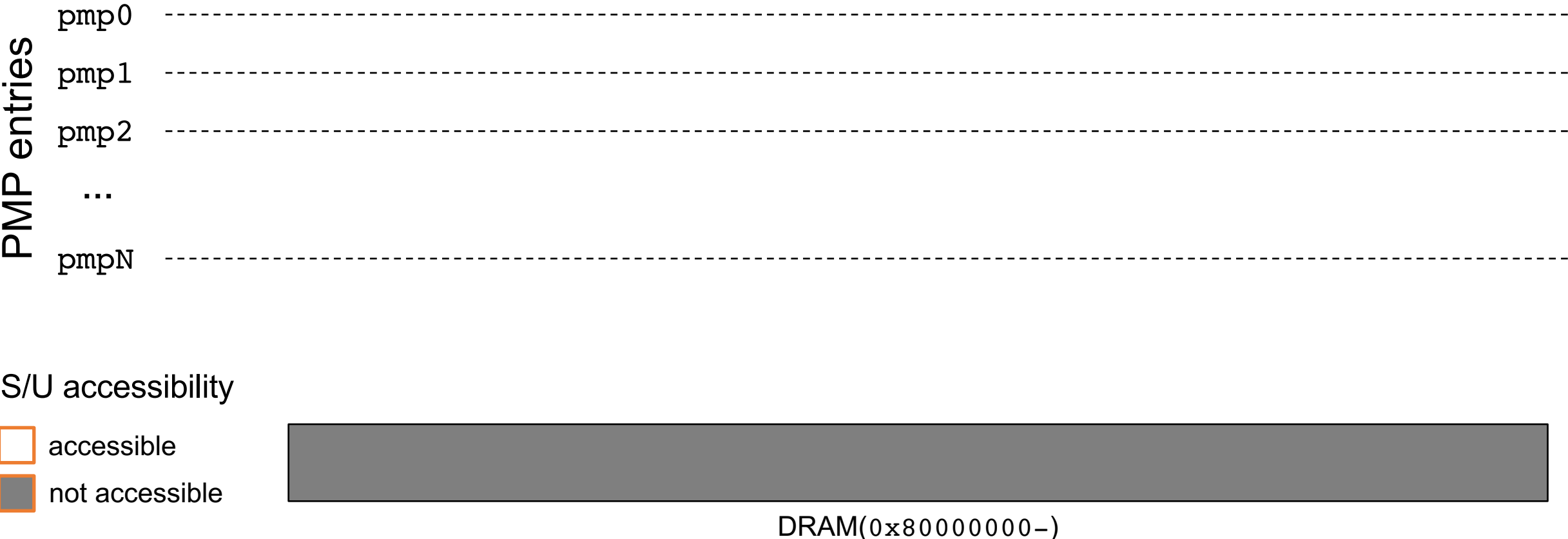


Physical Memory Protection (PMP)

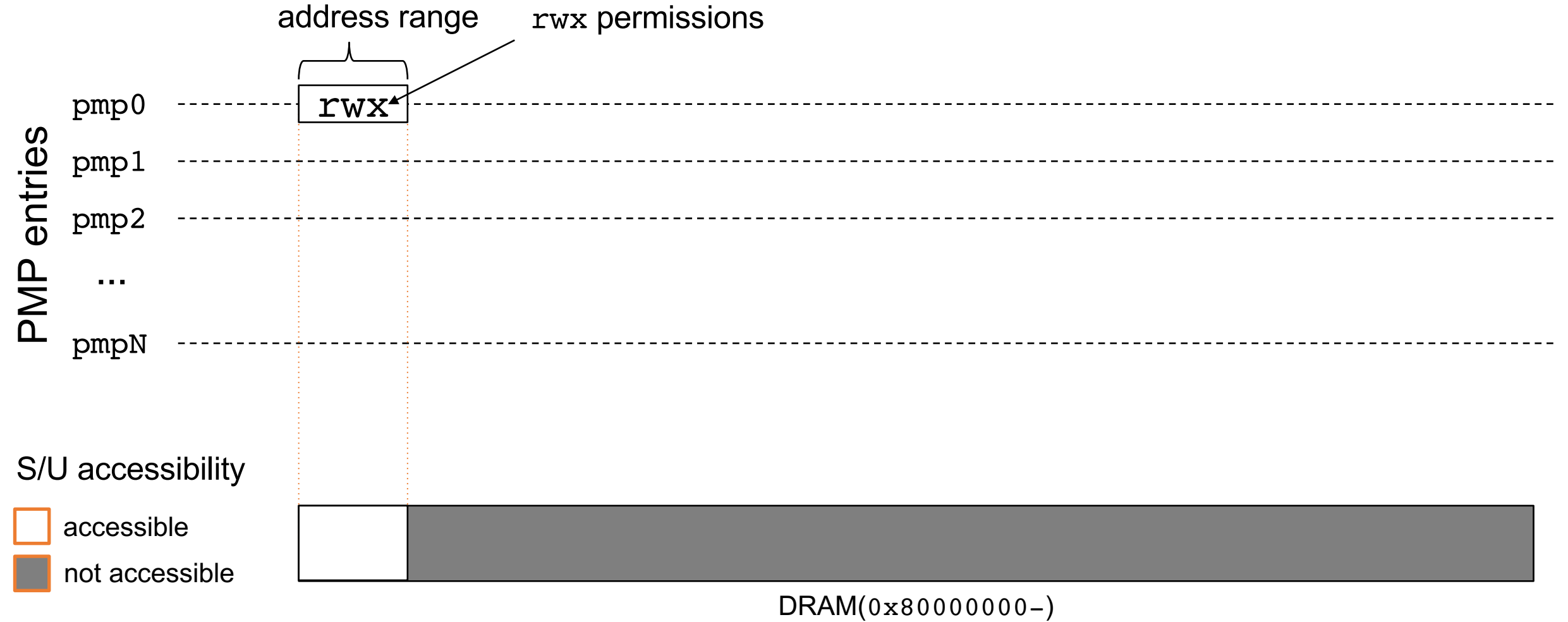
Isolation with RISC-V PMP



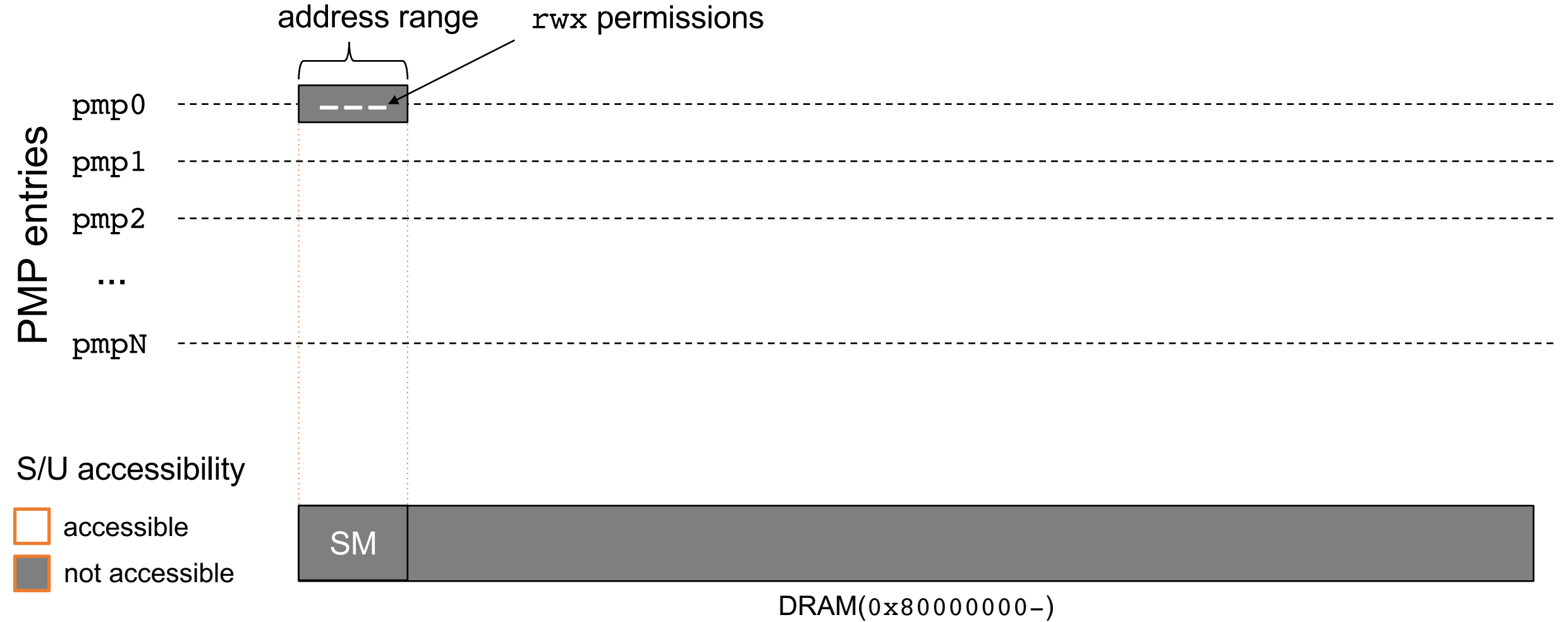
Isolation with RISC-V PMP



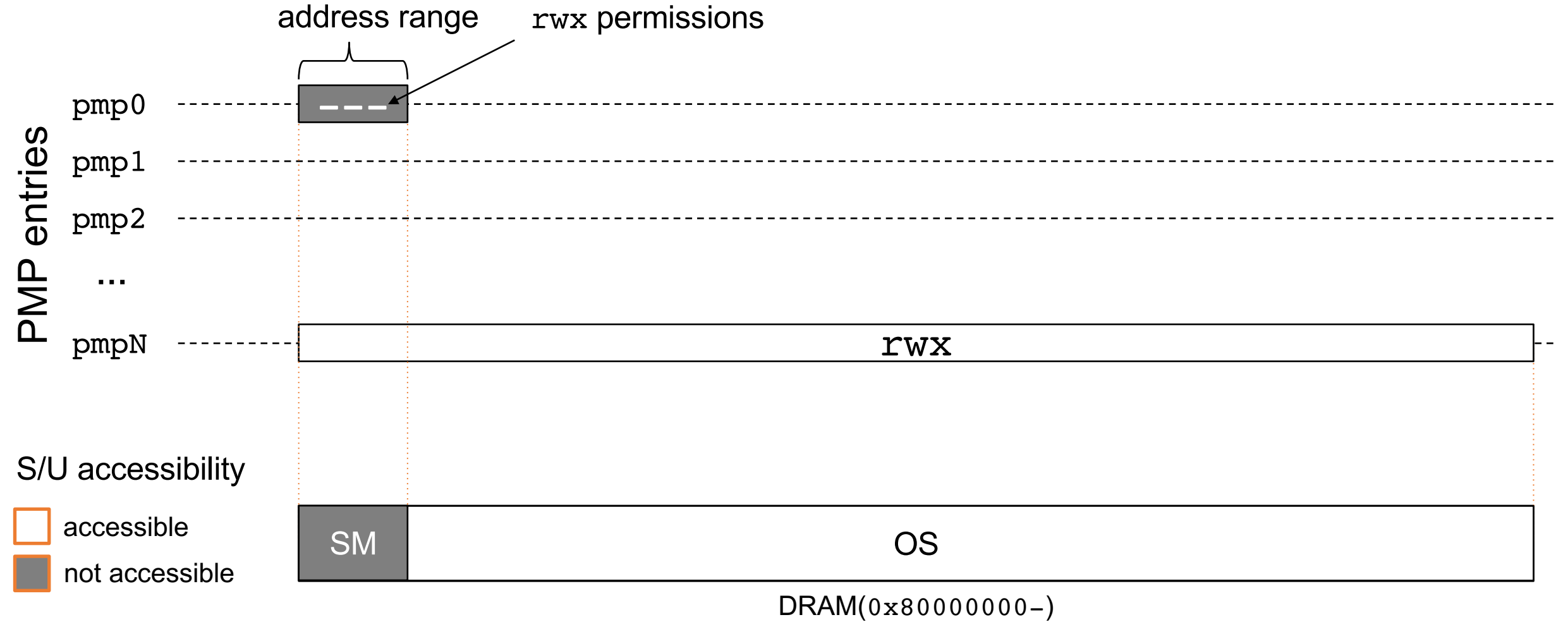
Isolation with RISC-V PMP



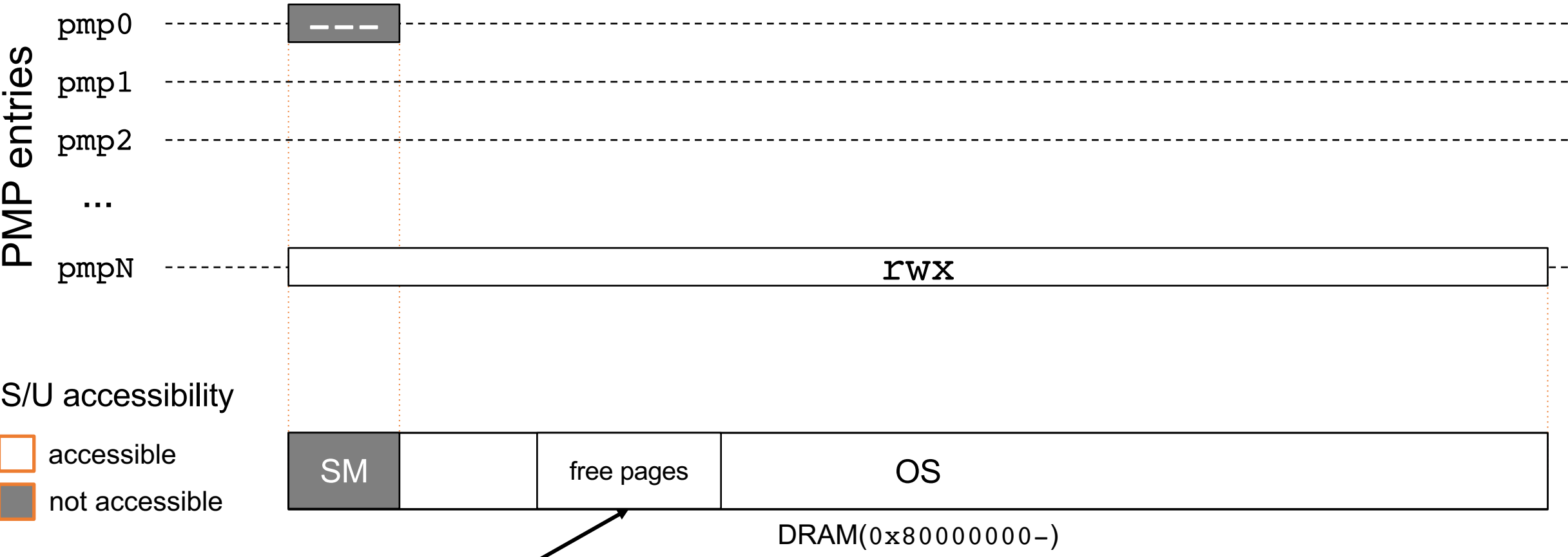
Isolation with RISC-V PMP



Isolation with RISC-V PMP

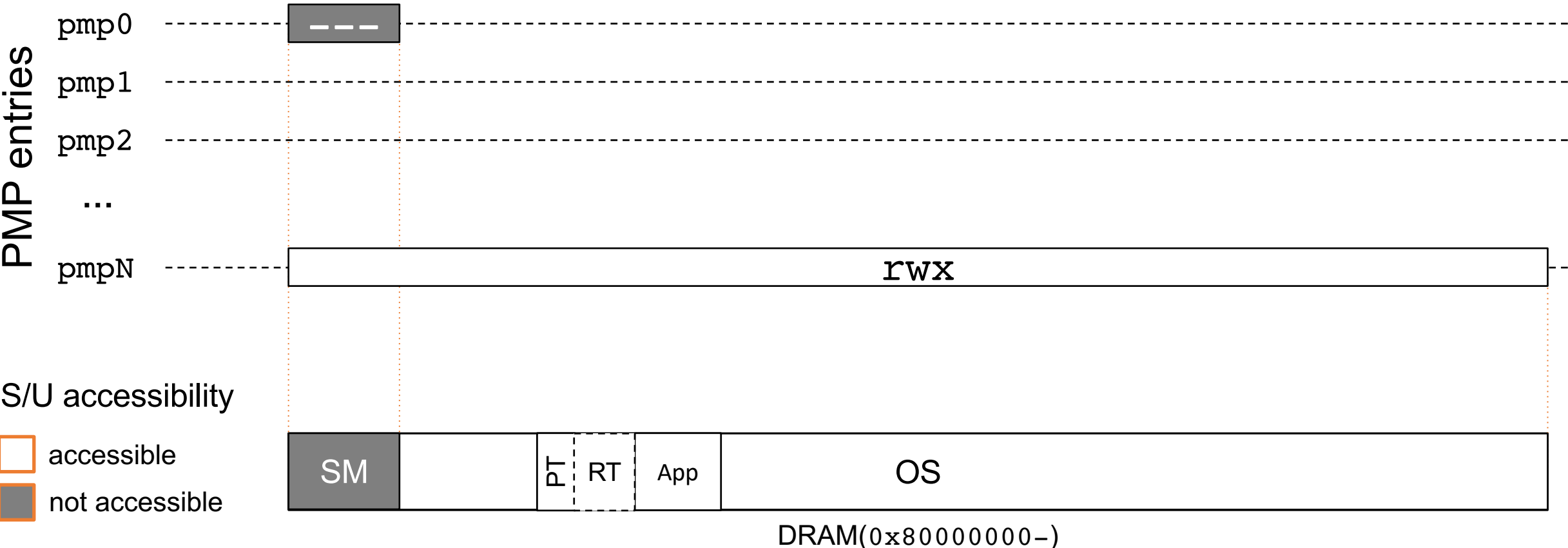


Creating an Isolated Enclave

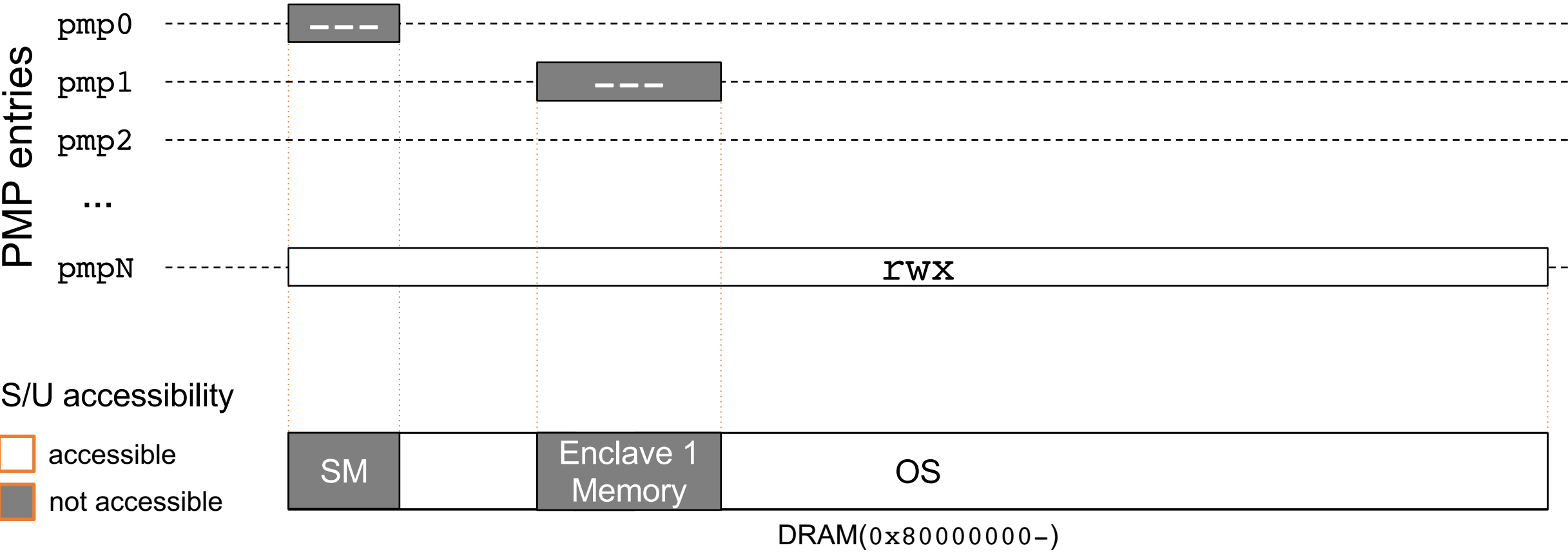


OS allocates a contiguous chunk

Creating an Isolated Enclave

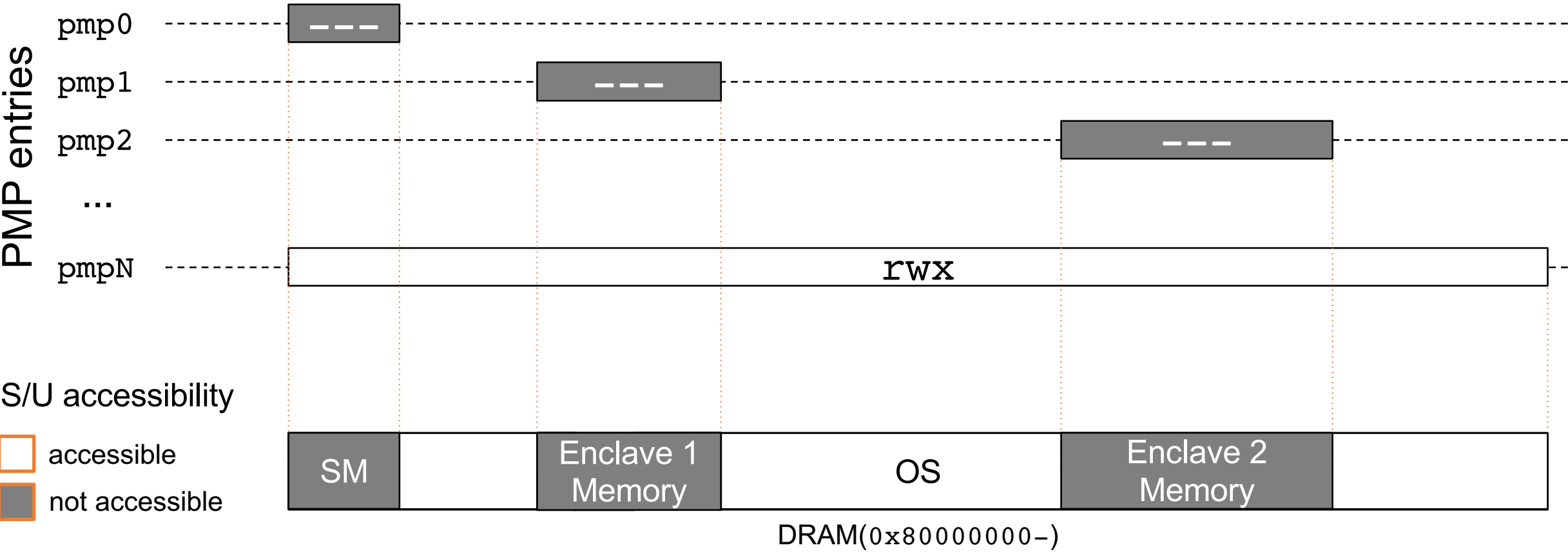


Creating an Isolated Enclave

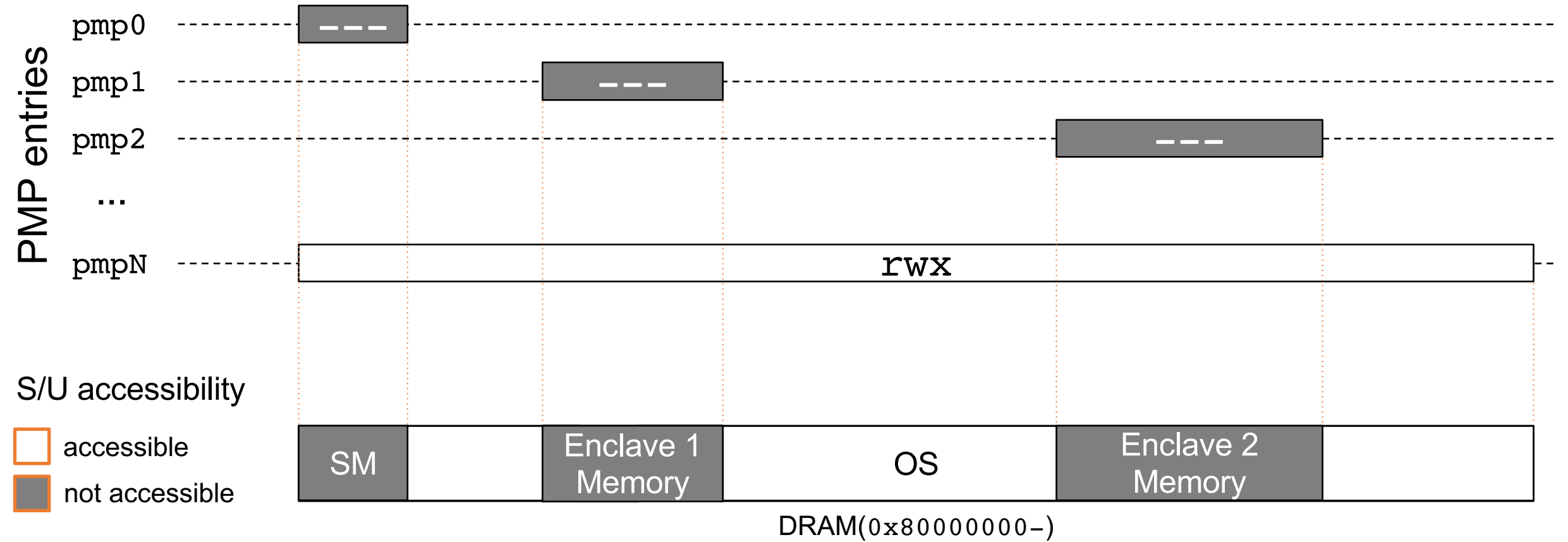


Creating an Isolated Enclave

OS can ask the SM to create multiple enclaves

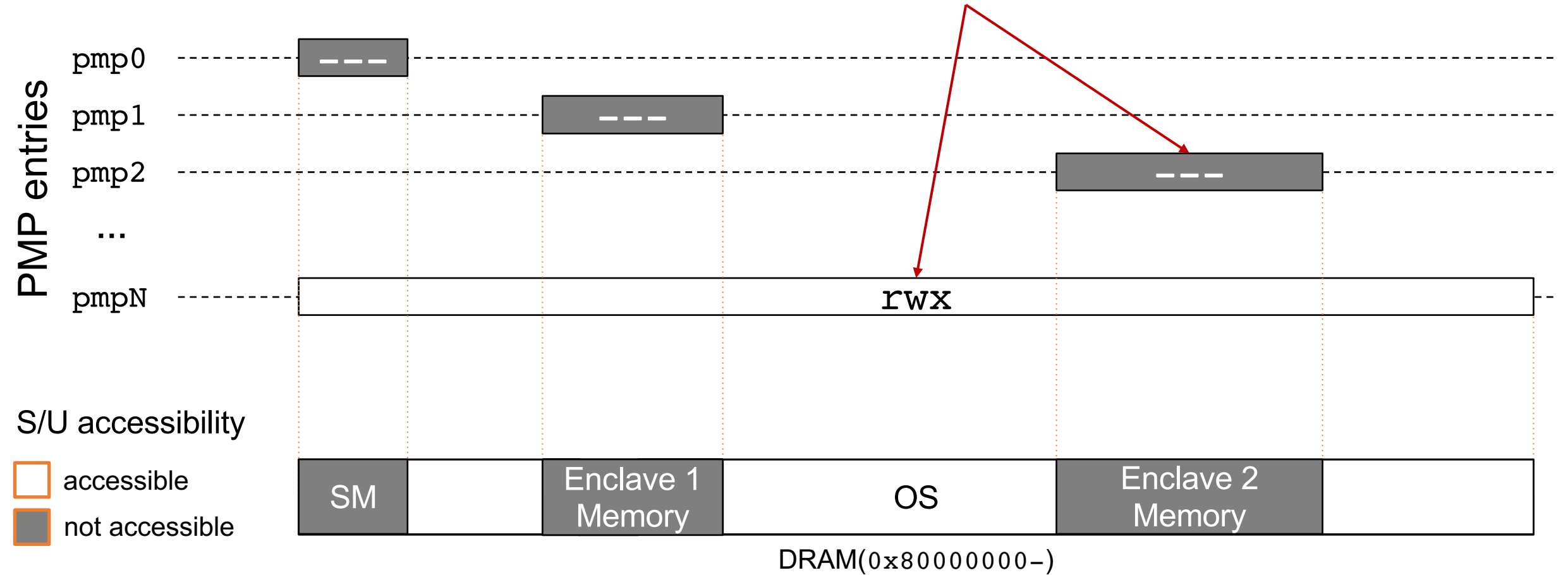


Executing an Enclave

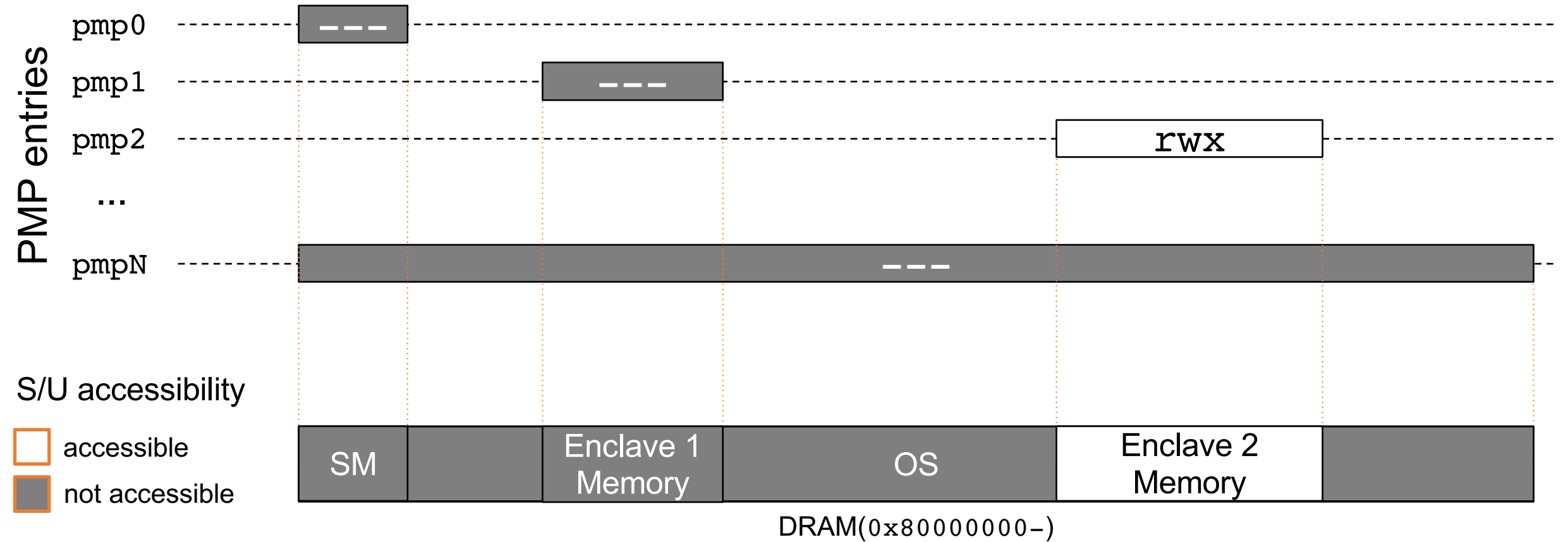


Executing an Enclave

For Enclave 2 SM sets `rwX` for `pmp2` and sets `---` for `pmpN`

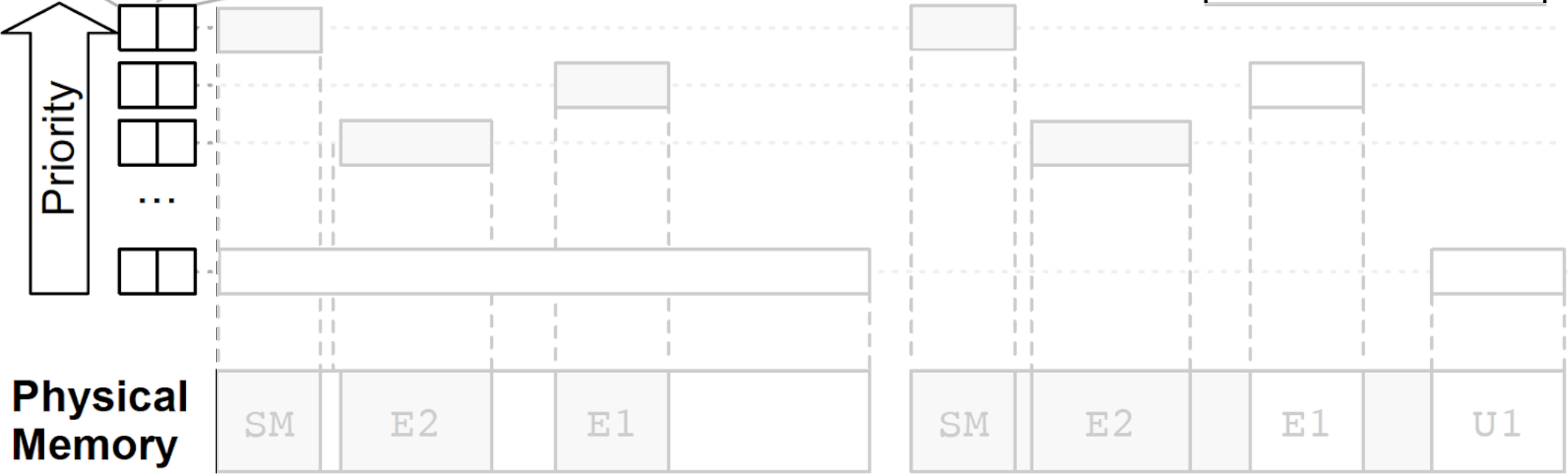
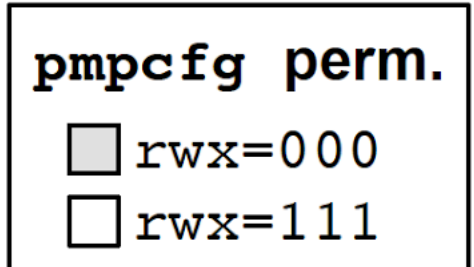
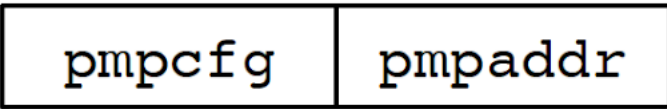


Executing an Enclave



Keystone: Memory Isolation via RISC-V PMP

PMP registers

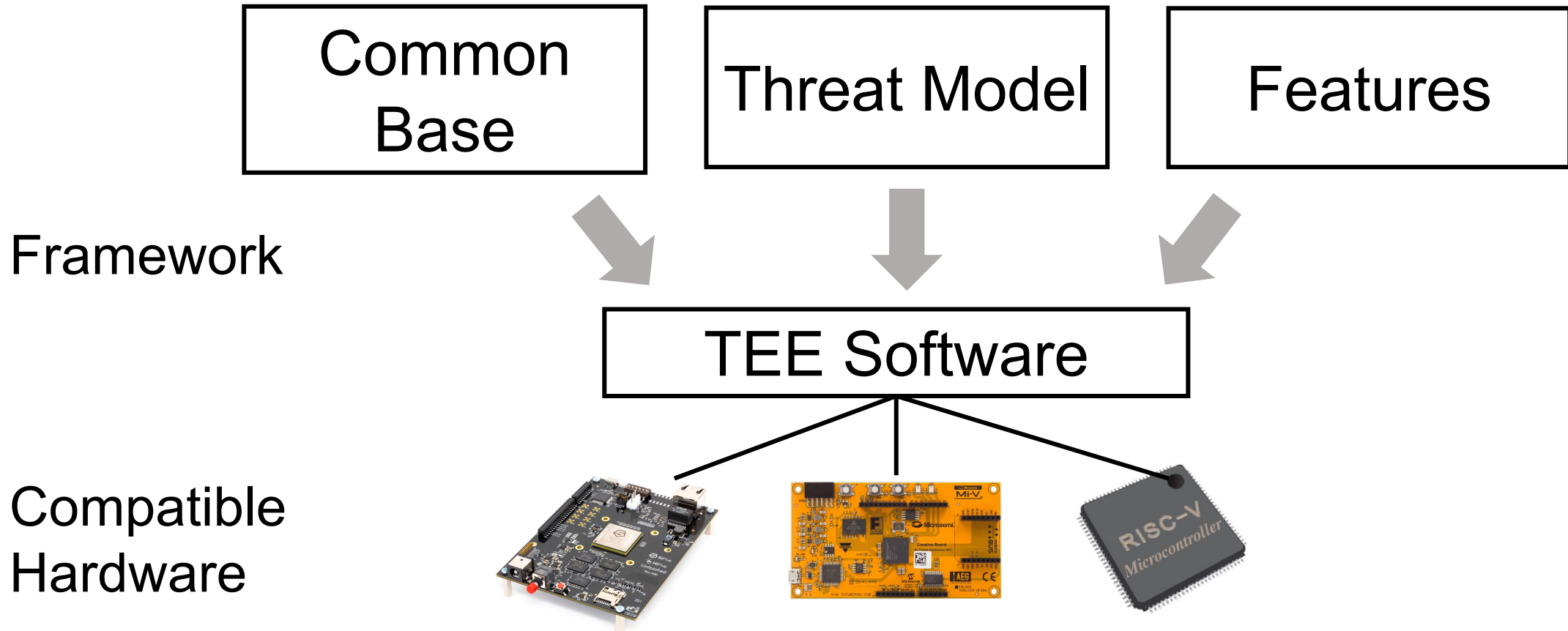


Untrusted Context

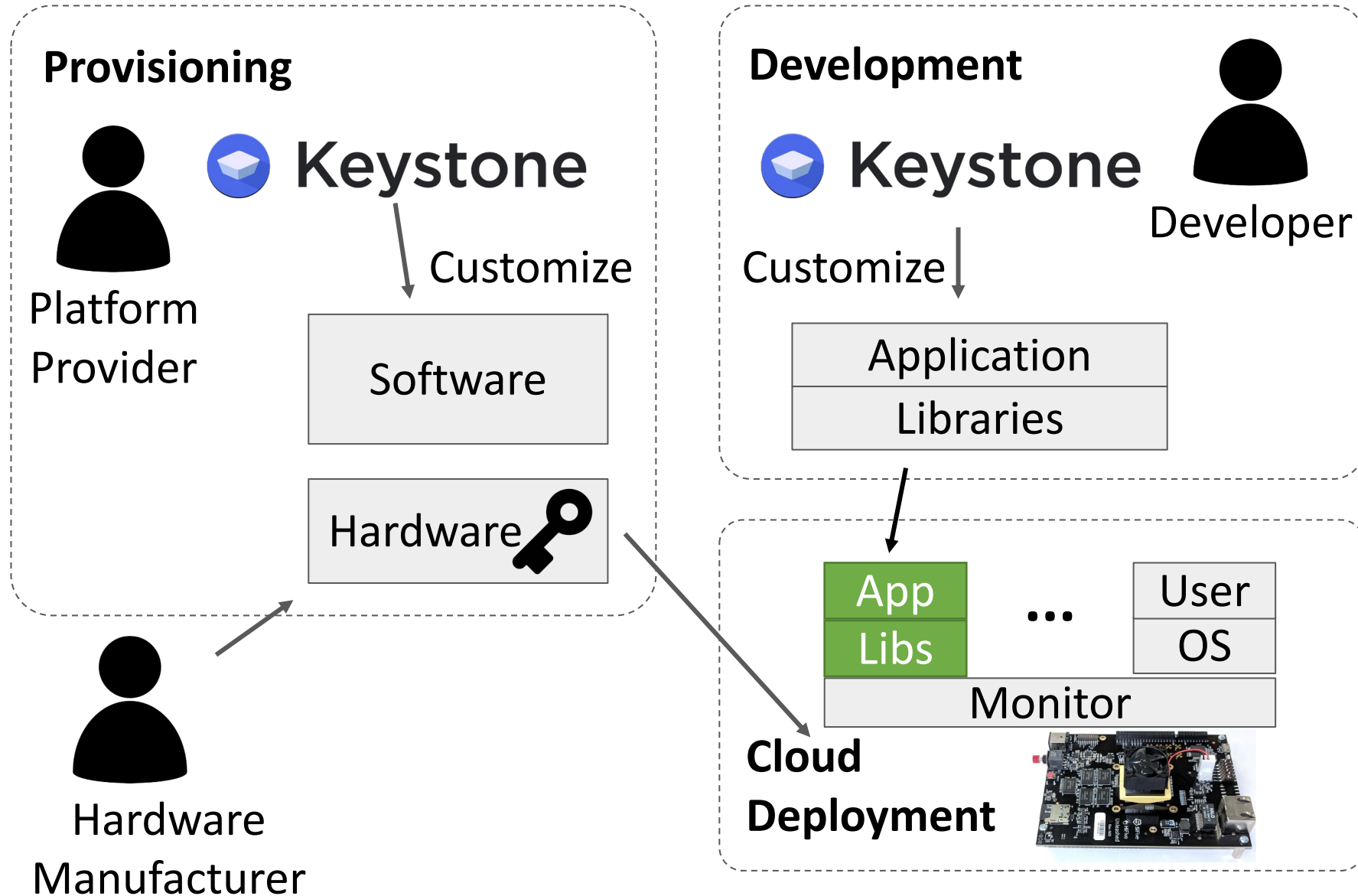
Enclave (E1) Context

Customizable TEEs

- A framework that provides building blocks of TEEs
- The platform provider and the enclave developer “customizes” the TEE



Keystone Workflow for Customizable TEEs

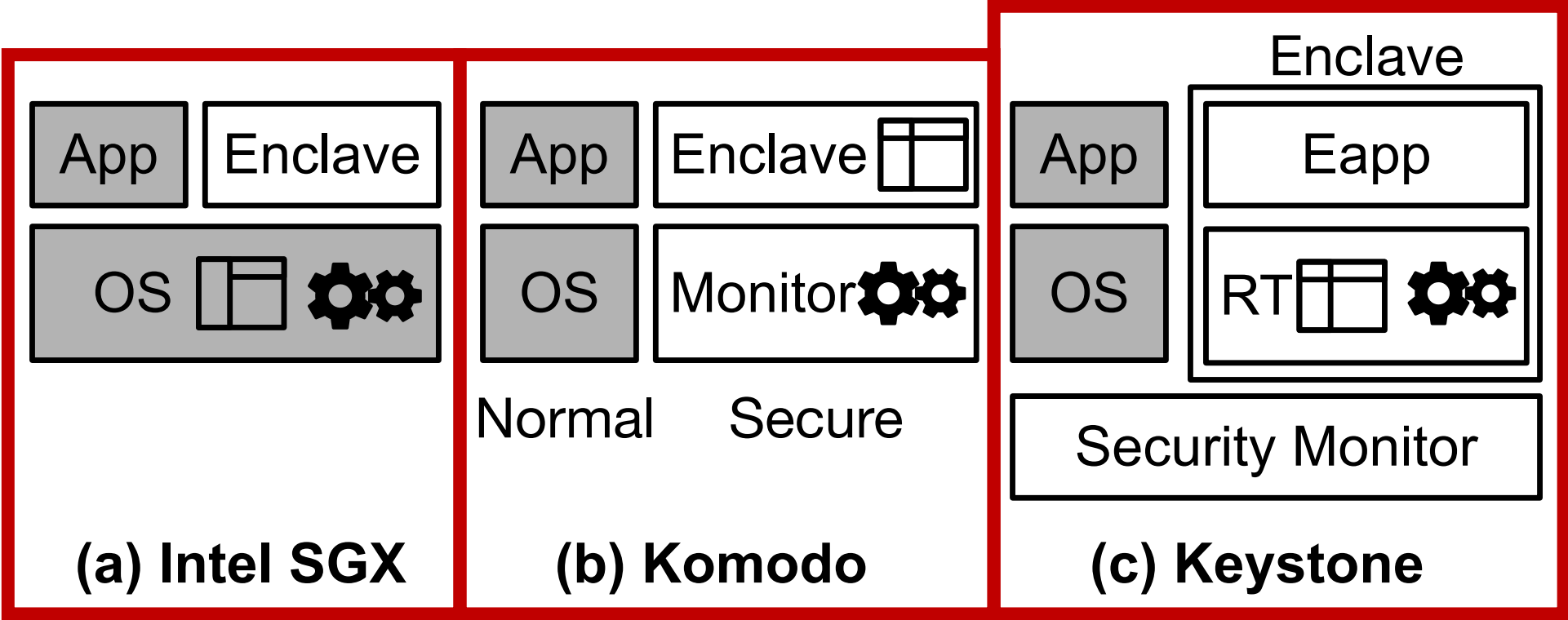


What does the Keystone Runtime Do?

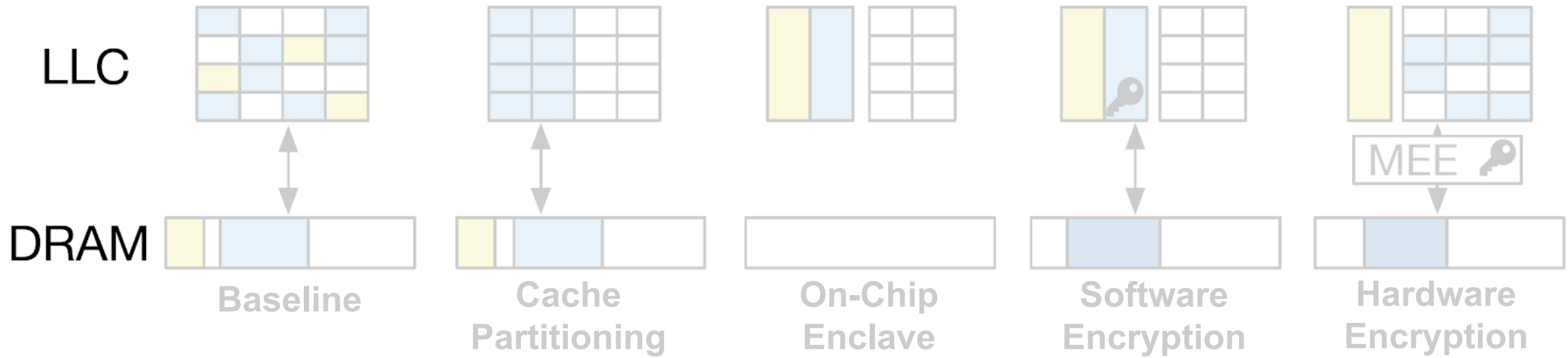
- A kernel-privileged trusted component for each enclave
 - Separation of security & functionality
- Flexible layer of abstraction
 - Minimal interface & functionality for small TCB (<4,000 LoC)
 - Fully featured, formally verified kernel (i.e., seL4)
- Fine-grained customizability for enclaves
 - Memory management: free memory, self-paging, memory encryption
 - Functionality: libraries (e.g., libc, musl-libc) and system calls

Memory Management in Keystone

- (1) Does the host share virtual addresses with the enclave?
- (2) Who owns the memory management unit (MMU)?



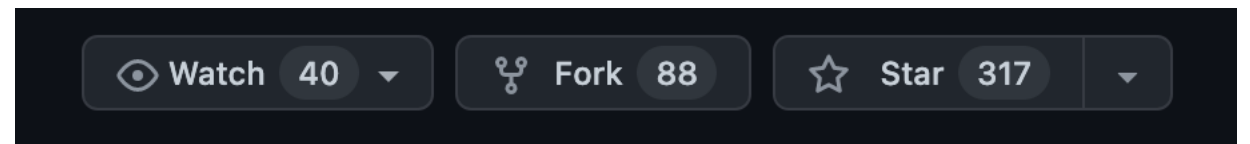
Various Memory Protection Mechanisms



Adversary	Baseline	Cache Partitioning	On-Chip Enclave	Software Encryption	Hardware Encryption
SW + CC	●	●	●	●	●
Cache SC		●	●	●	●
HW			●	●	●

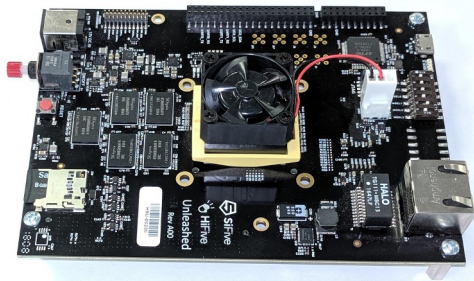
Keystone: an Open Framework for TEEs

- Provide a **common base** for diverse TEEs
 - Security monitor: programmable trusted layer below kernel/hypervisor
 - Hardware-enforced memory and context isolation
 - Hardware root of trust and attestation
- A software framework for **customizable TEE**
 - Separation of security and functionality (e.g., resource mgmt.)
 - Fine-grained configuration of modular extensions
- An open-source, full-stack implementation for further research

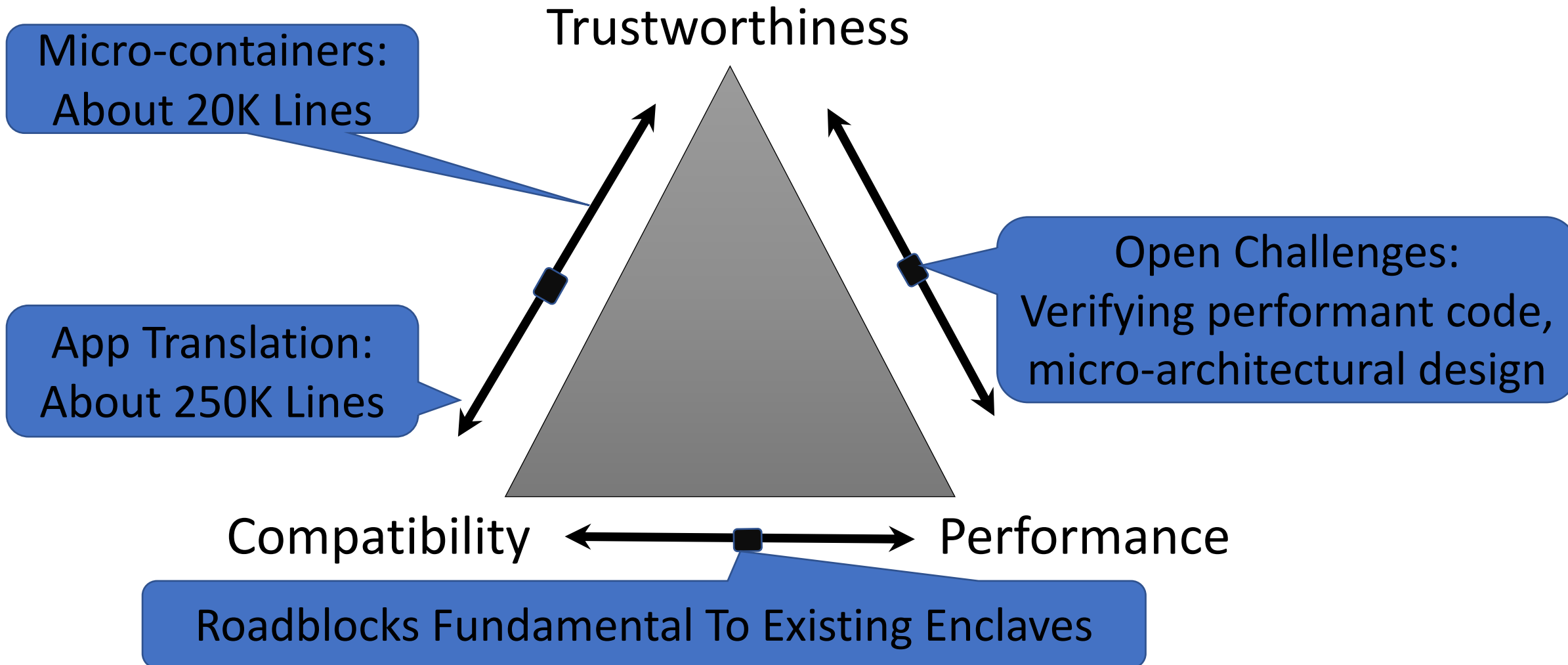


Getting Started with Keystone

- Setup and Tutorials: <https://keystone-enclave.org/getting-started>
- Tested & Available on RISC-V boards, FPGAs, FireSim, and QEMU

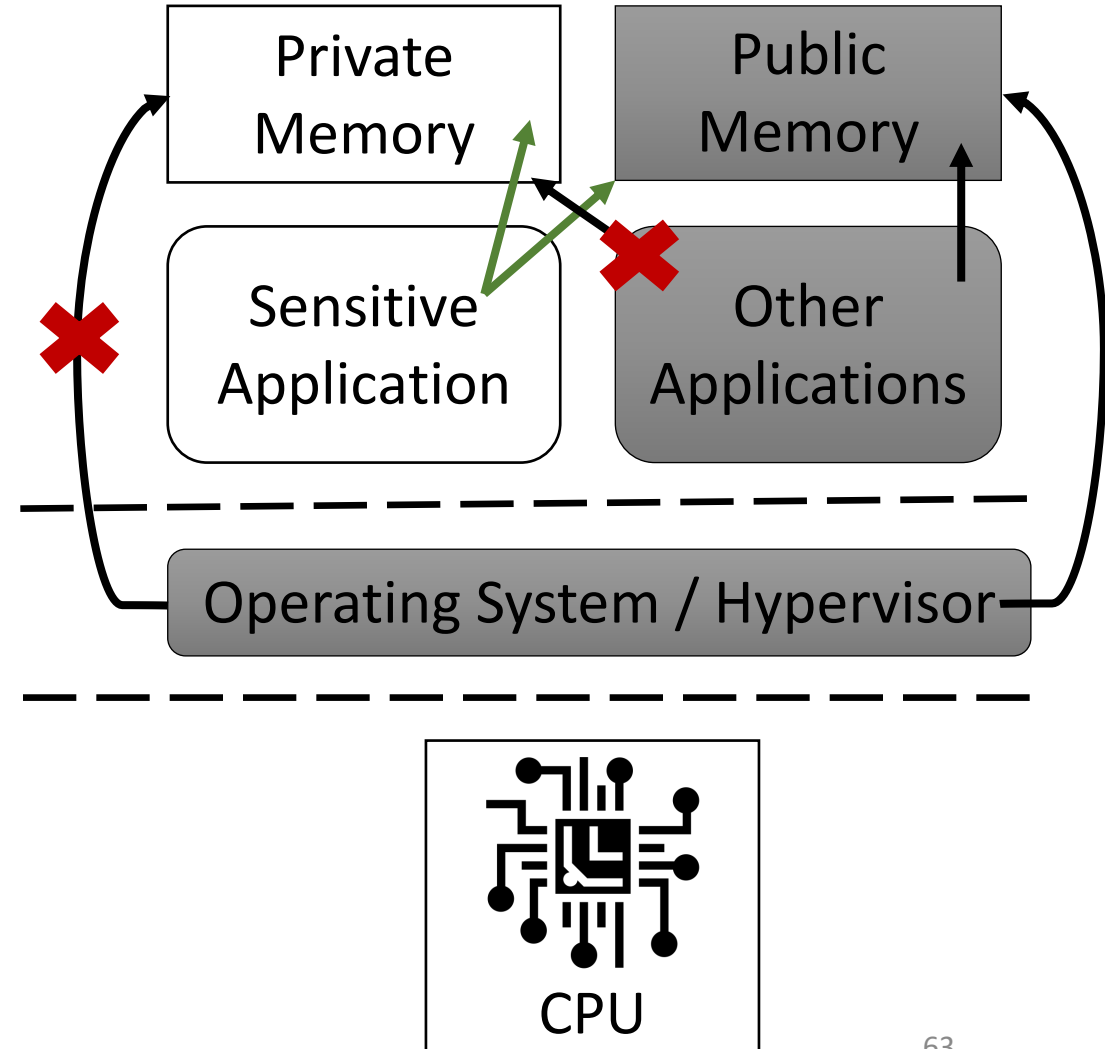


Compatibility, Performance, Trustworthiness

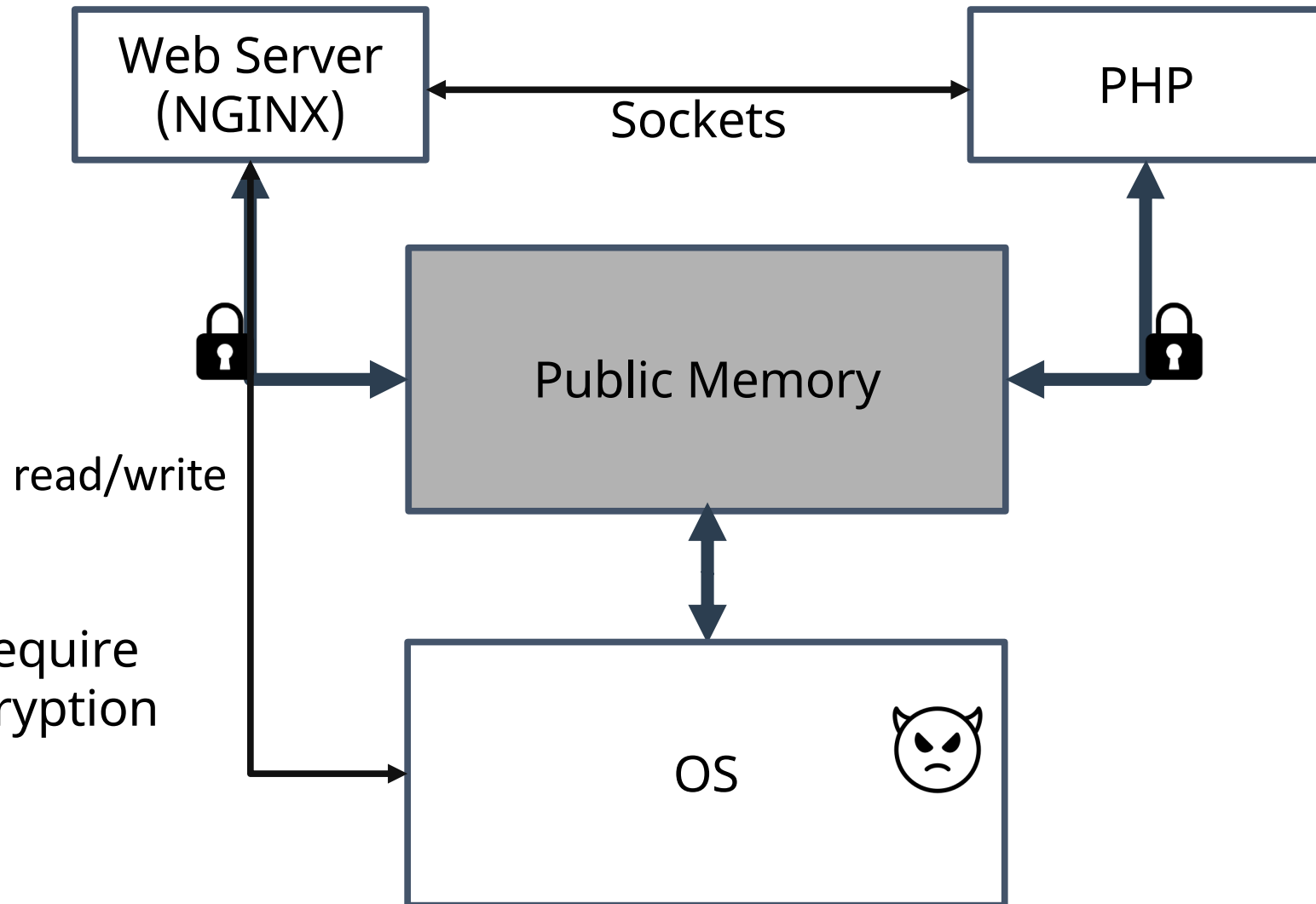


TEE Memory Model: Spatial Isolation

- Strong enclave boundaries
 - Size and Permissions
 - Static in SGX1, dynamic in SGX2
- Exclusive memory access
 - No sharing
 - In all TEEs

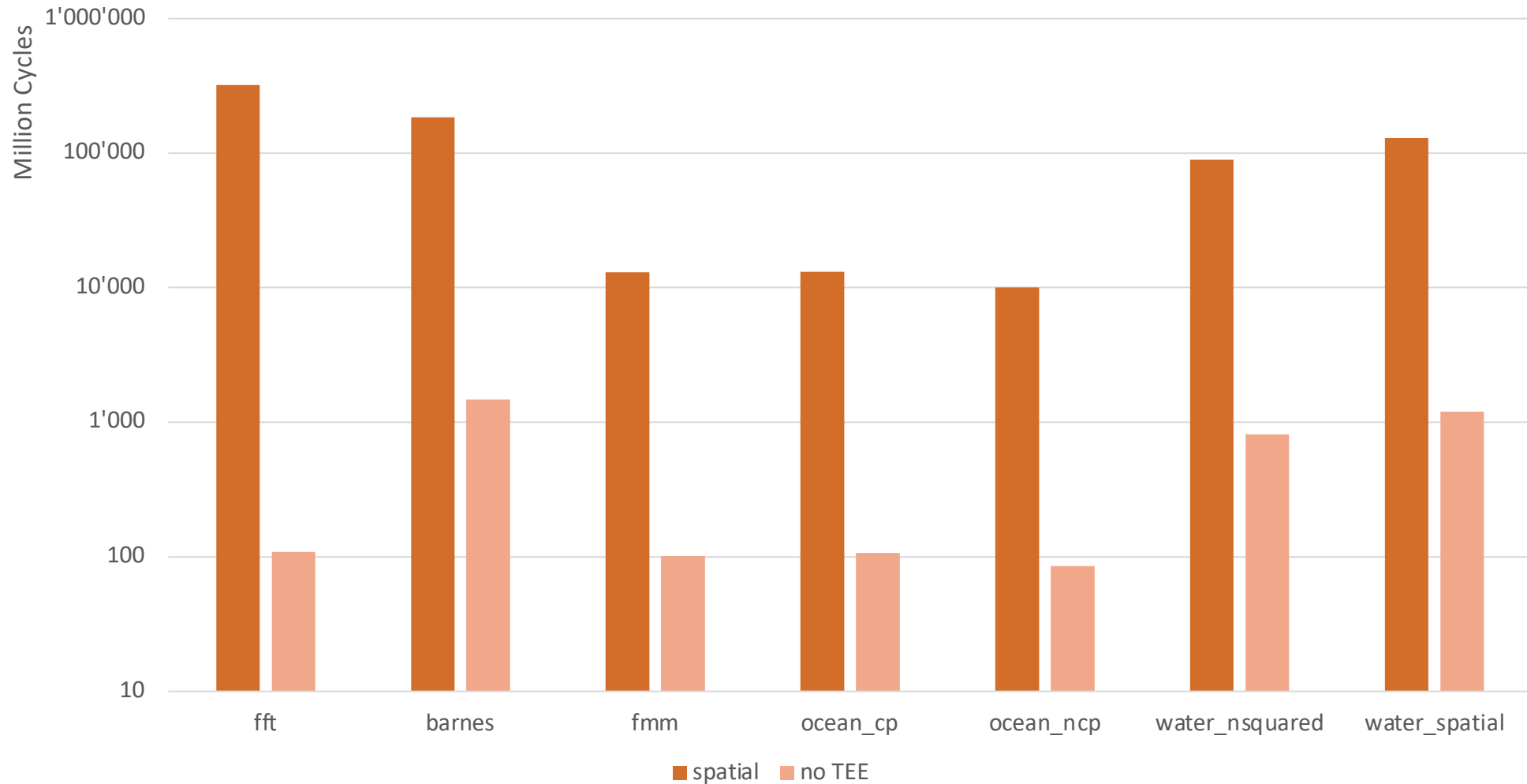


Performance Impact of Spatial Isolation

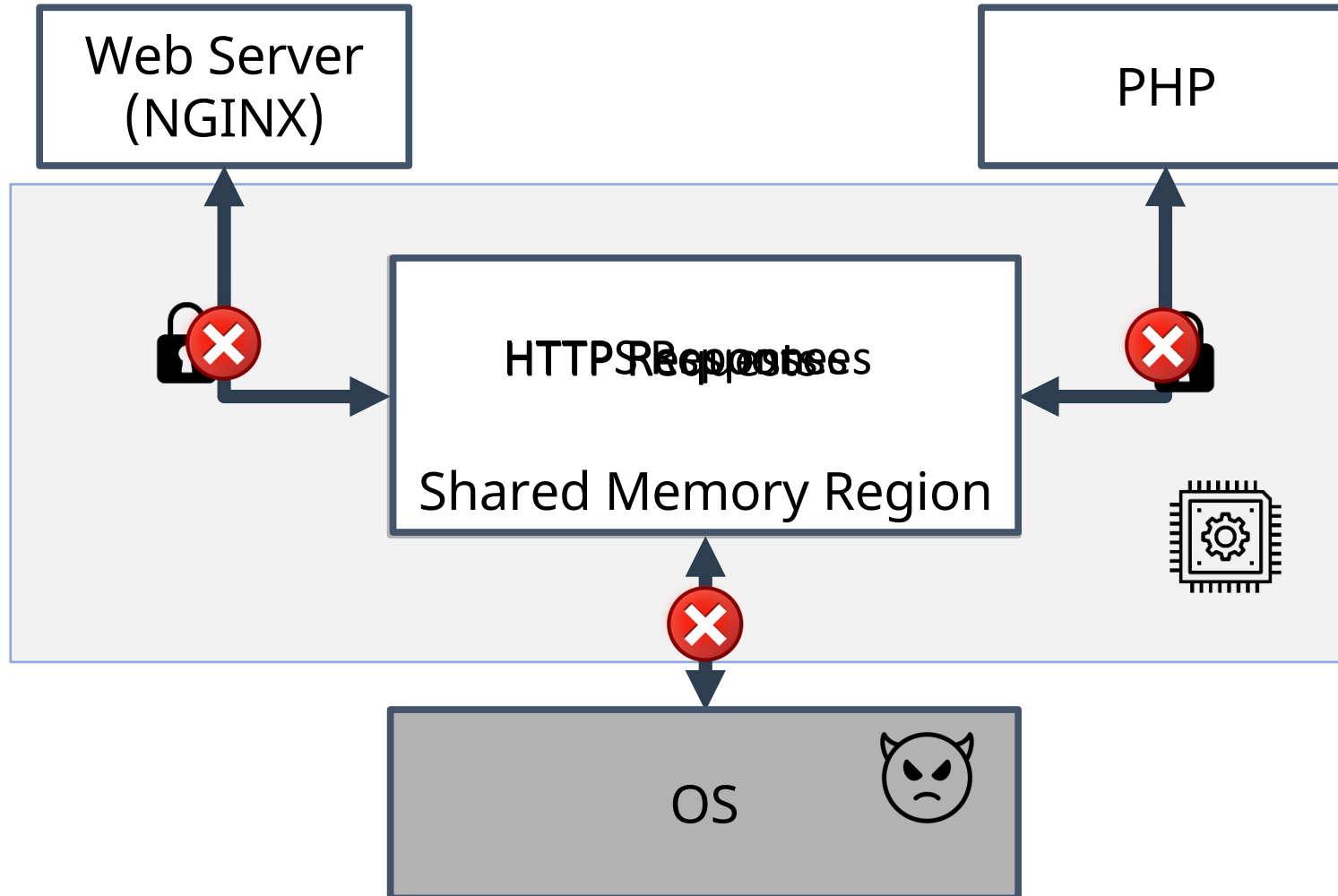


- All data transfers require
- Encryption + decryption
 - 2 extra copies

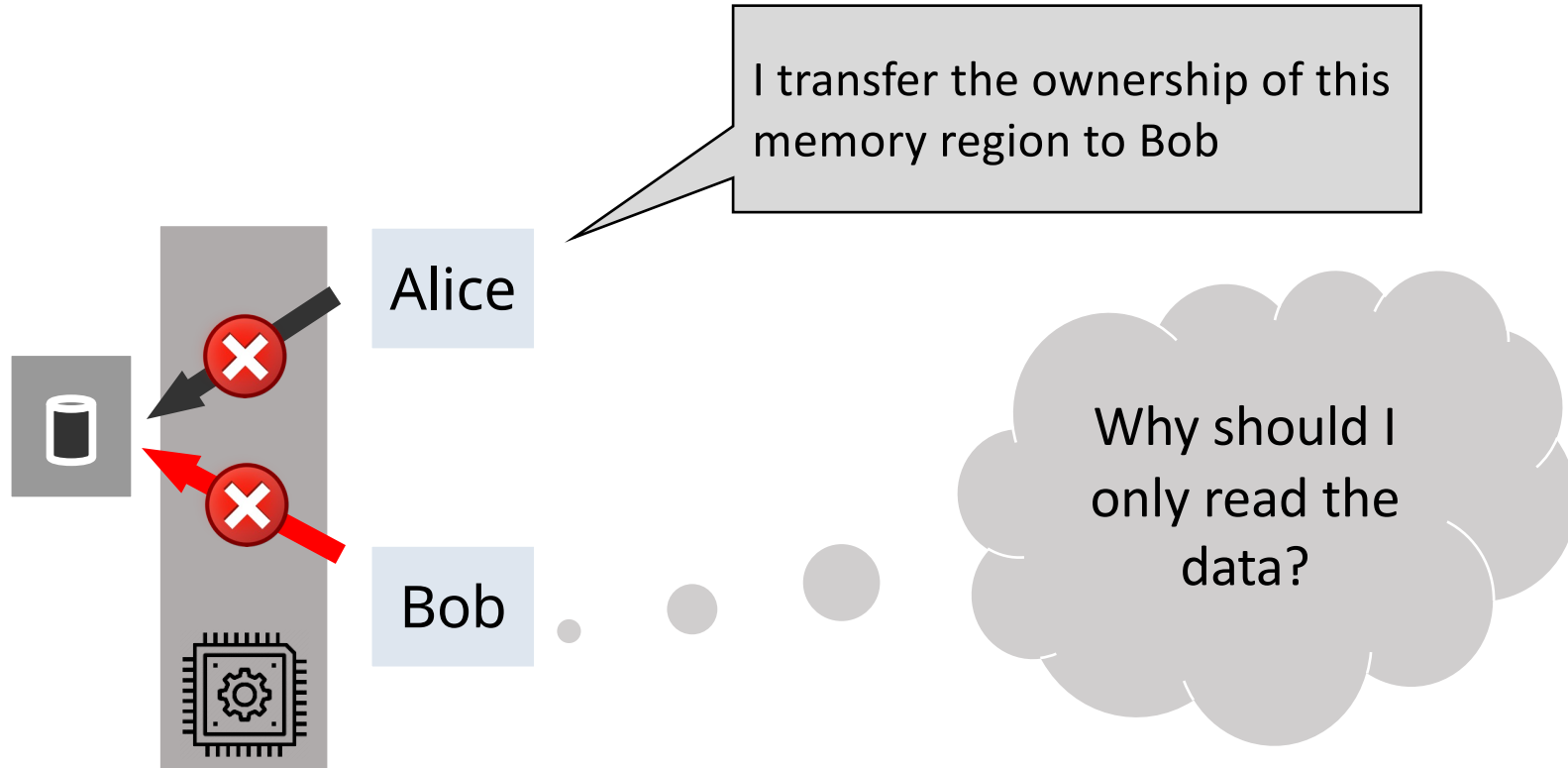
Overheads in orders of magnitude



Alternative: Dynamic Memory Isolation Model

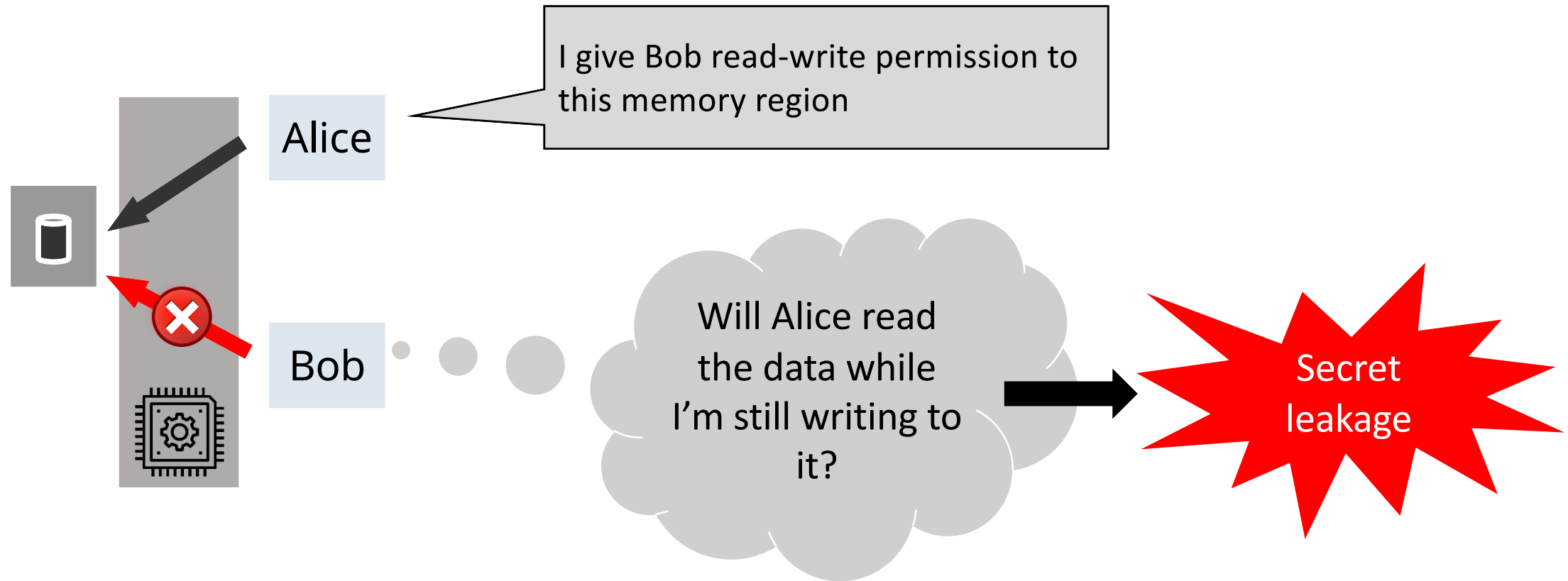


Naïve Design I: Ownership Delegation



Alice has no control over what Bob does

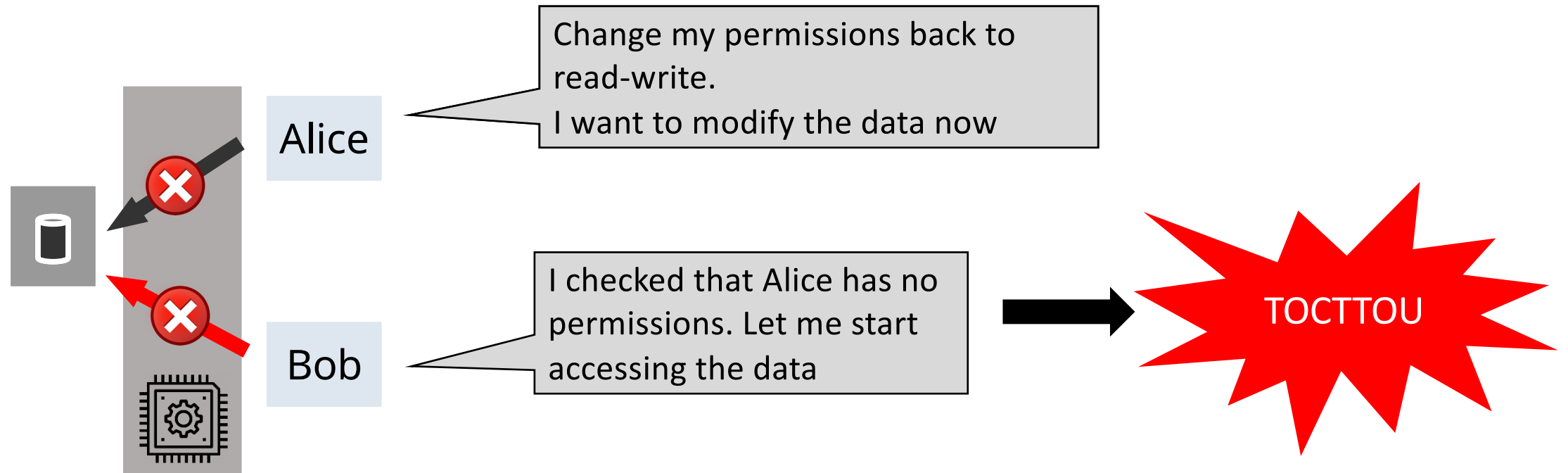
Naïve Design II: Permission Delegation



Bob has no guarantee about what Alice can do

Naïve Design III

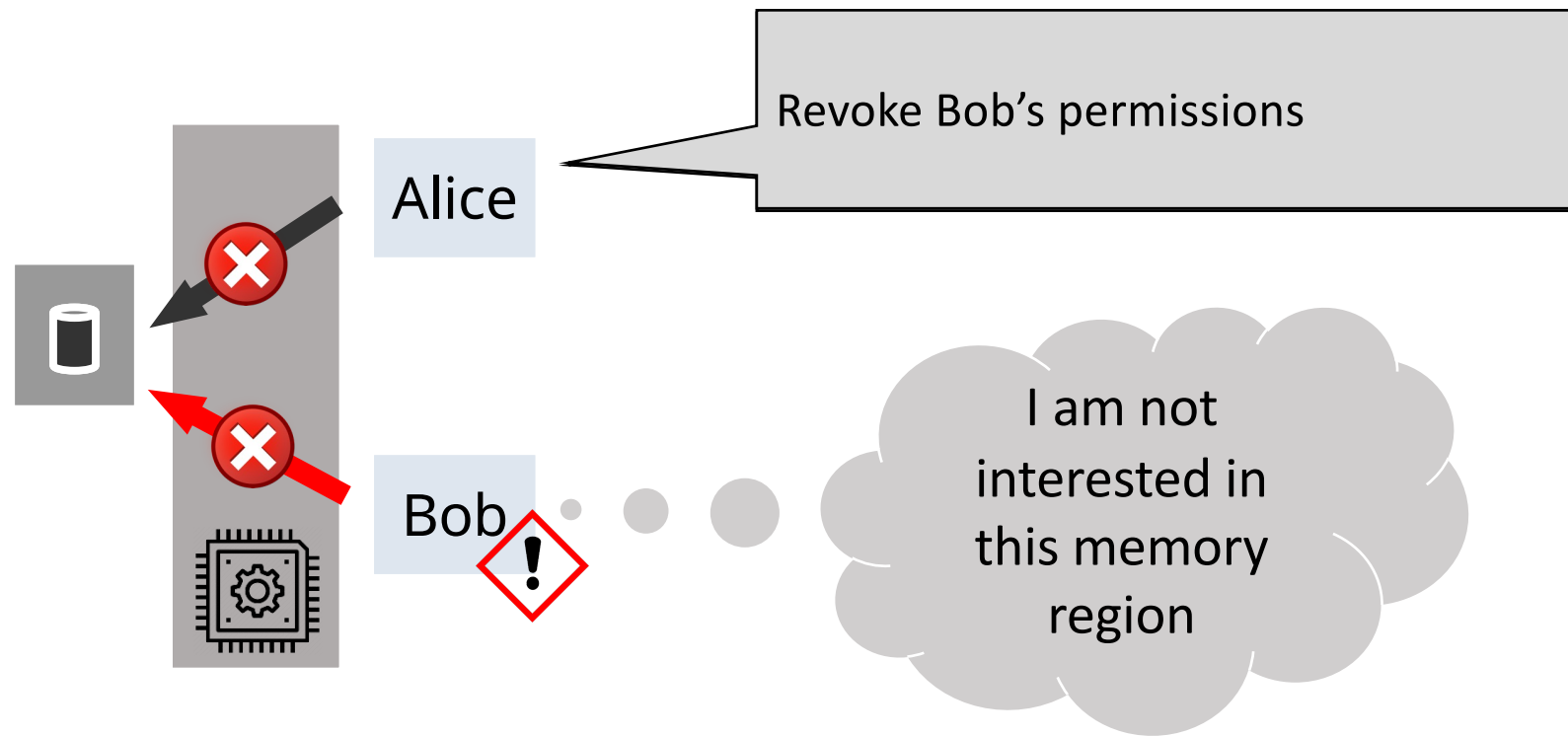
Permission Delegation with “Get Permissions”



Bob (still) has no guarantees about what Alice can do

Naïve Design IV

Permission Delegation with Notifications



Bob has no way to express what he wants

Dynamicity vs. Simplicity

Dynamic permissions:

- Flexibility and high performance
- Complicated security analysis

Static permissions:

- Simple for security analysis
- Lack of flexibility and performance

Can we have both
high performance and simple security decisions?

Getting the Best of Both Worlds

	Alice	Bob	Carol
A	----	r--	----
B	r---	rw--	rwX-

∩

	Alice	Bob	Carol
A	rwX	rw-	r---
B	rw--	rw--	rwX

Dynamic View

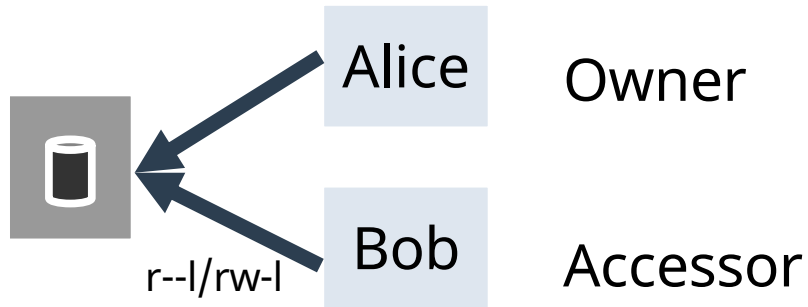
- Effective permissions
- Adjusted by accessors
- Localised

Lock bit: exclusive access guarantee

Static View

- Maximum permissions
- Set by owners
- Localised

Elasticlave Memory Model



Permissions: **rwxl/rwxl**

- **Effective permissions:** dynamically set by accessor
- **Maximum permissions:** set by owner (rwxl for owner)

Enforced Constraints

- Effective bounded by maximum
 - r---/rw-- allowed
 - r--l/rw-- not allowed
- Memory access bounded by rwx in effective
 - r---/rw-- read-only
 - rw--/rwx- read-write
- Lock bit (l) guarantees exclusive accesses

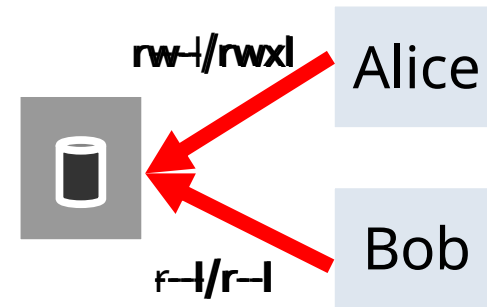
Elasticlave Interface

Owner: share (encl, region, max_perm)

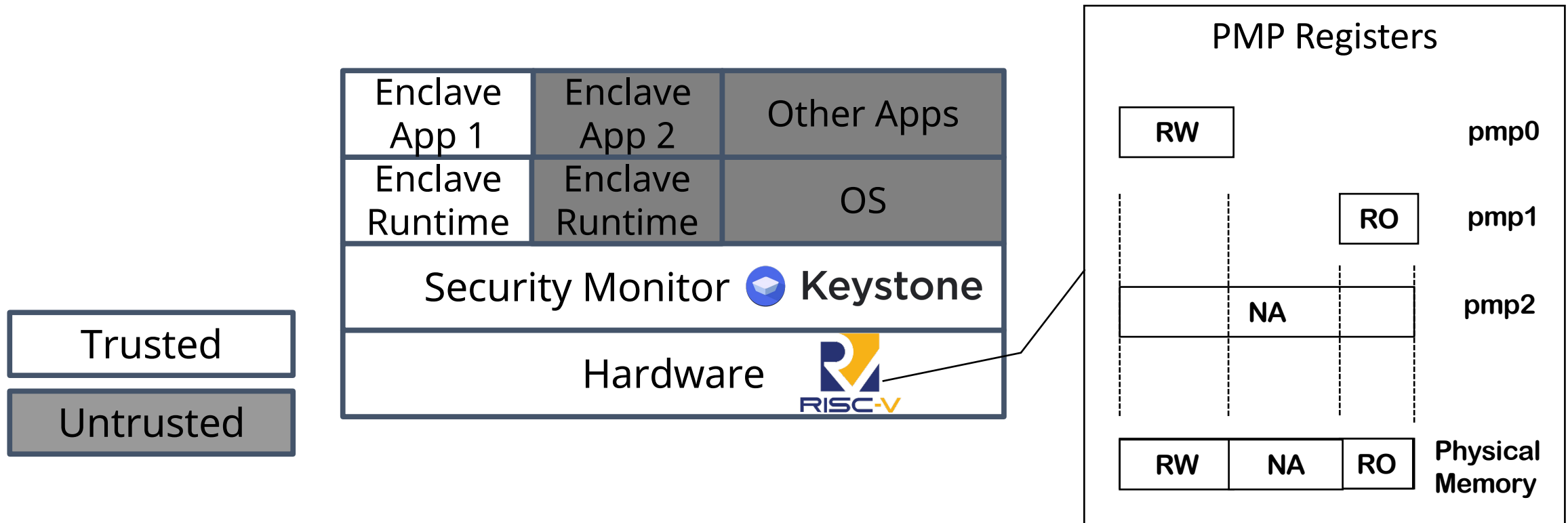
Accessor: change (region, effective_perm)

Lock holder: transfer (encl, region)

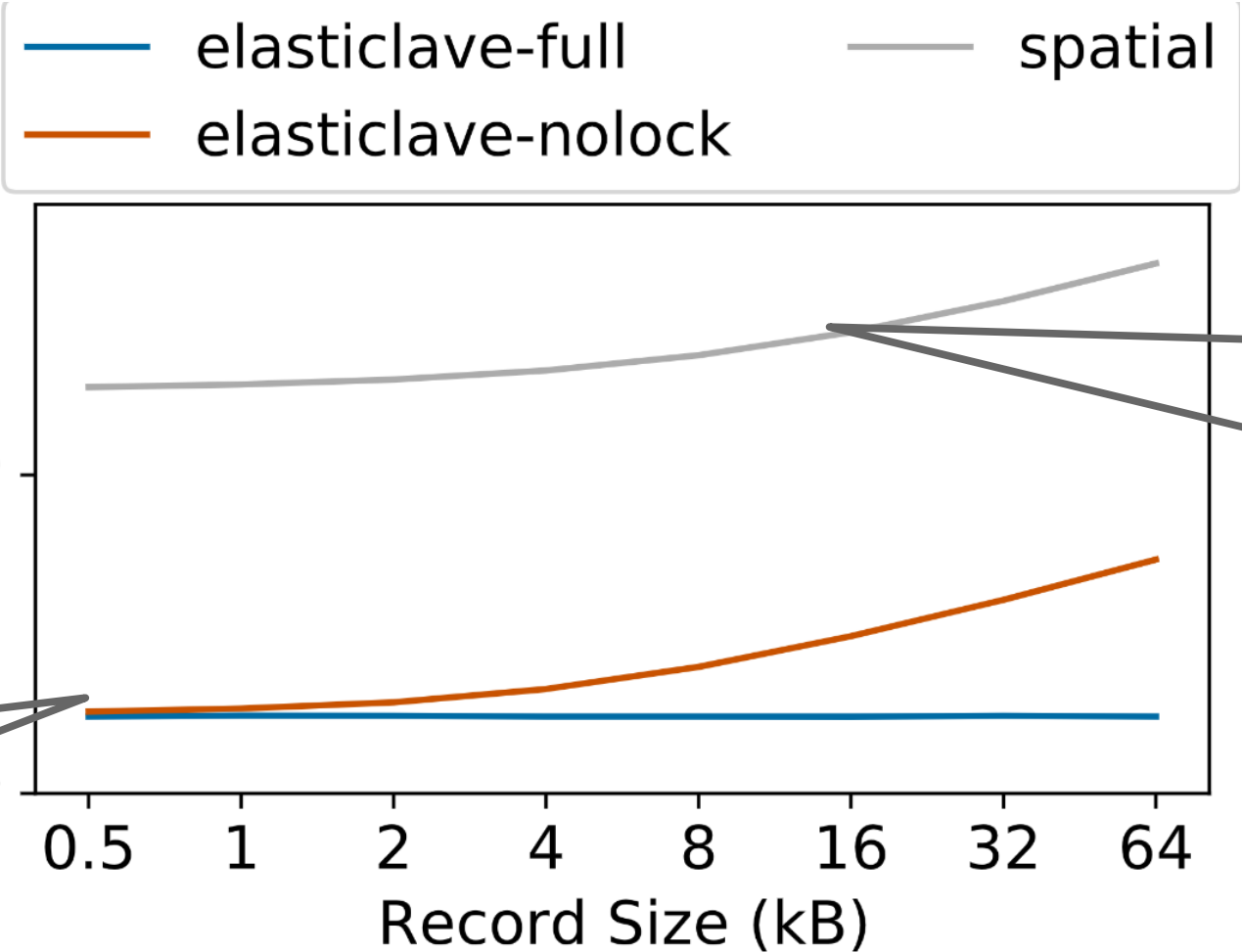
Owner: destroy (region)



Elasticlave Support in Keystone



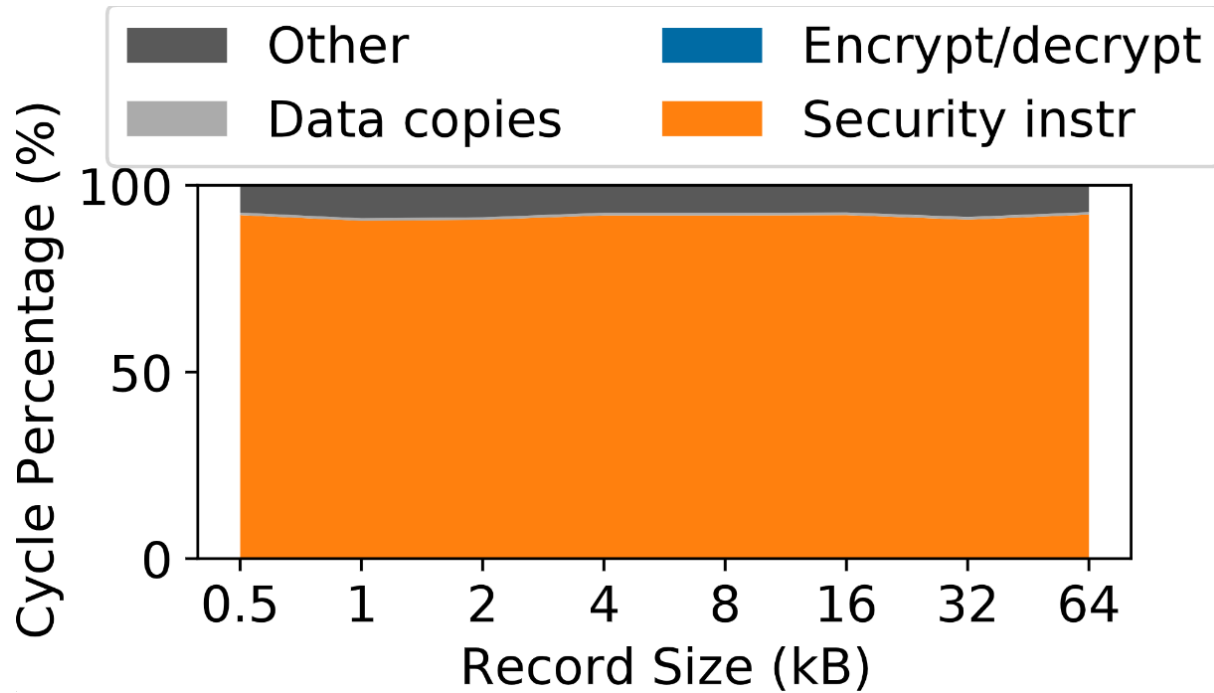
Elasticlave Performance



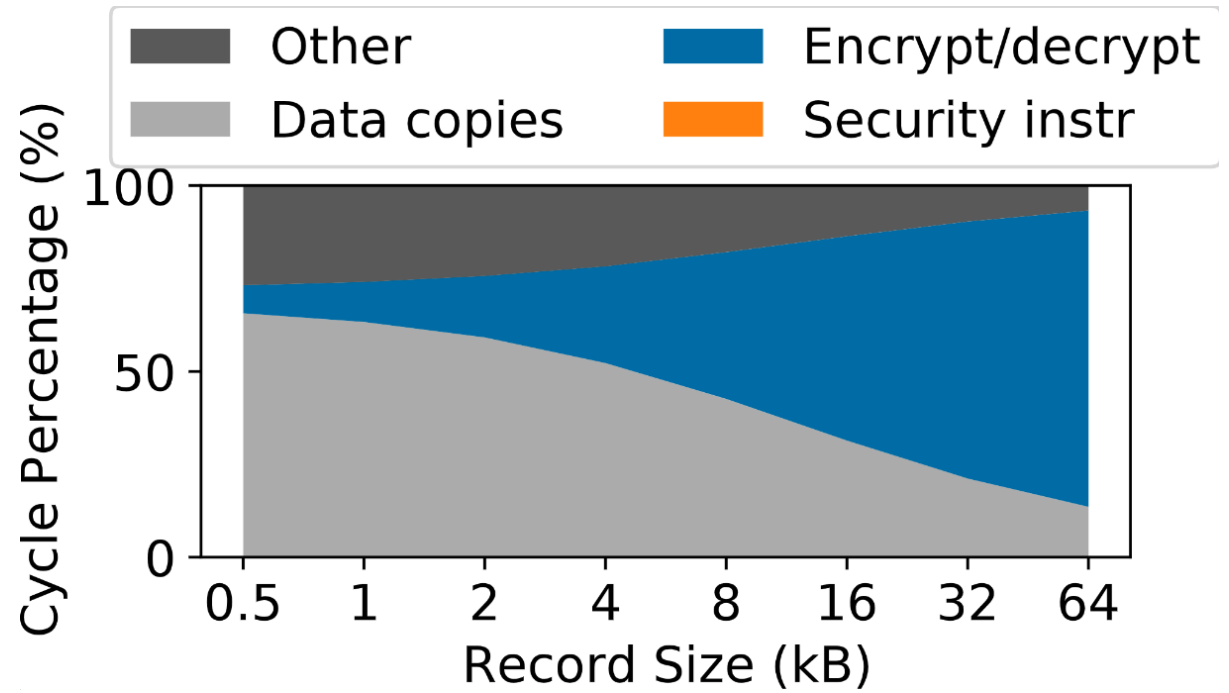
Overhead of Elasticlave is constant

Spatial isolation > 100x slower. Overhead increases with record size

Overhead Breakdown



Elasticlave



Spatial Isolation

Our Work: Building components of this stack

New Applications [Arxiv'18], [ICDCS'19]

Secure Computation [CCS'13]

Analysis & hardening

[PLDI'14], [FSE'15], [NDSS'19], [CCS'20]

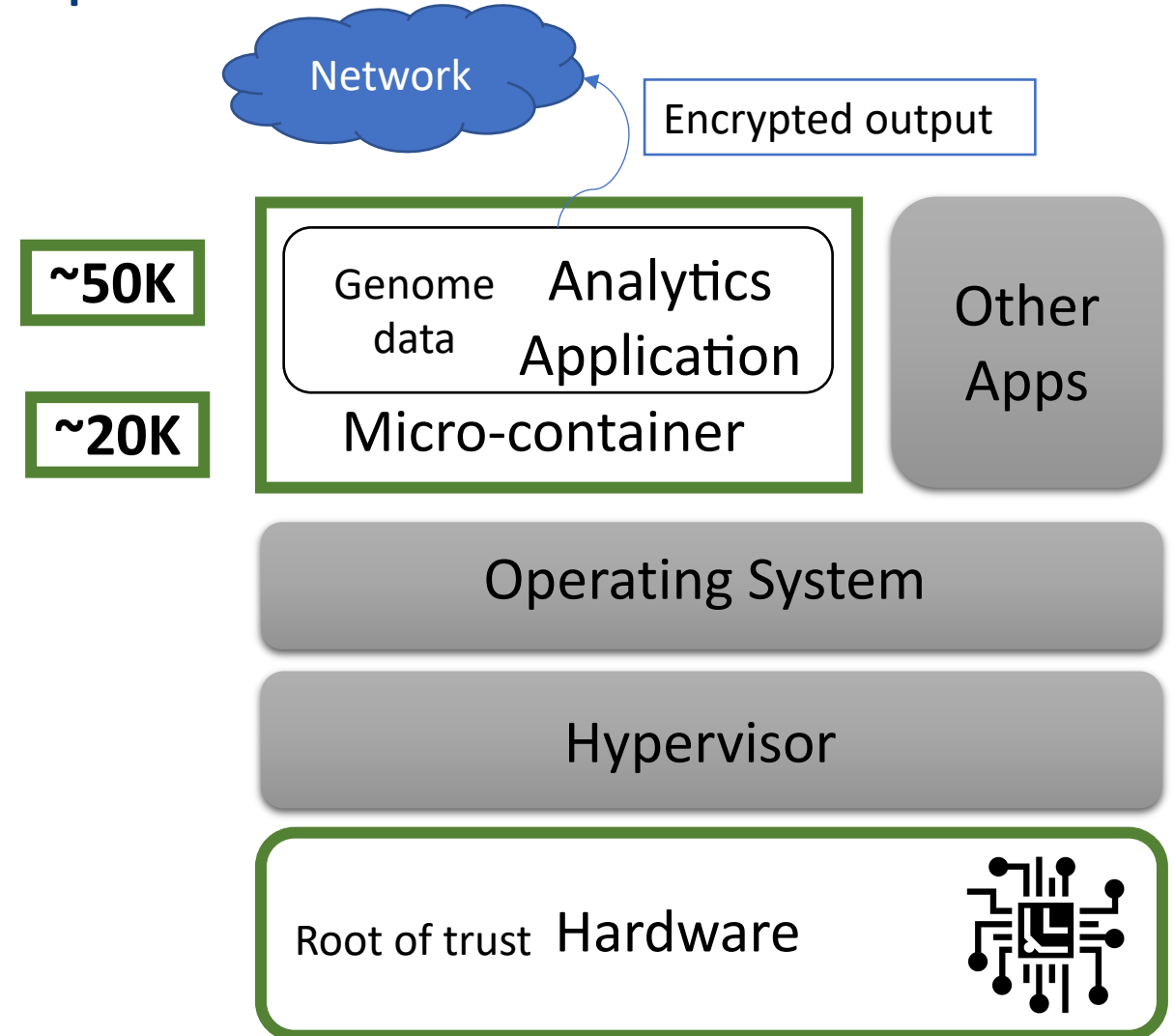
Rich functionality [NDSS'17], [TOPS'22]

Formal verification [Usenix Security'20]

Attacks & Defenses [AsiaCCS'16], [CCS'21]

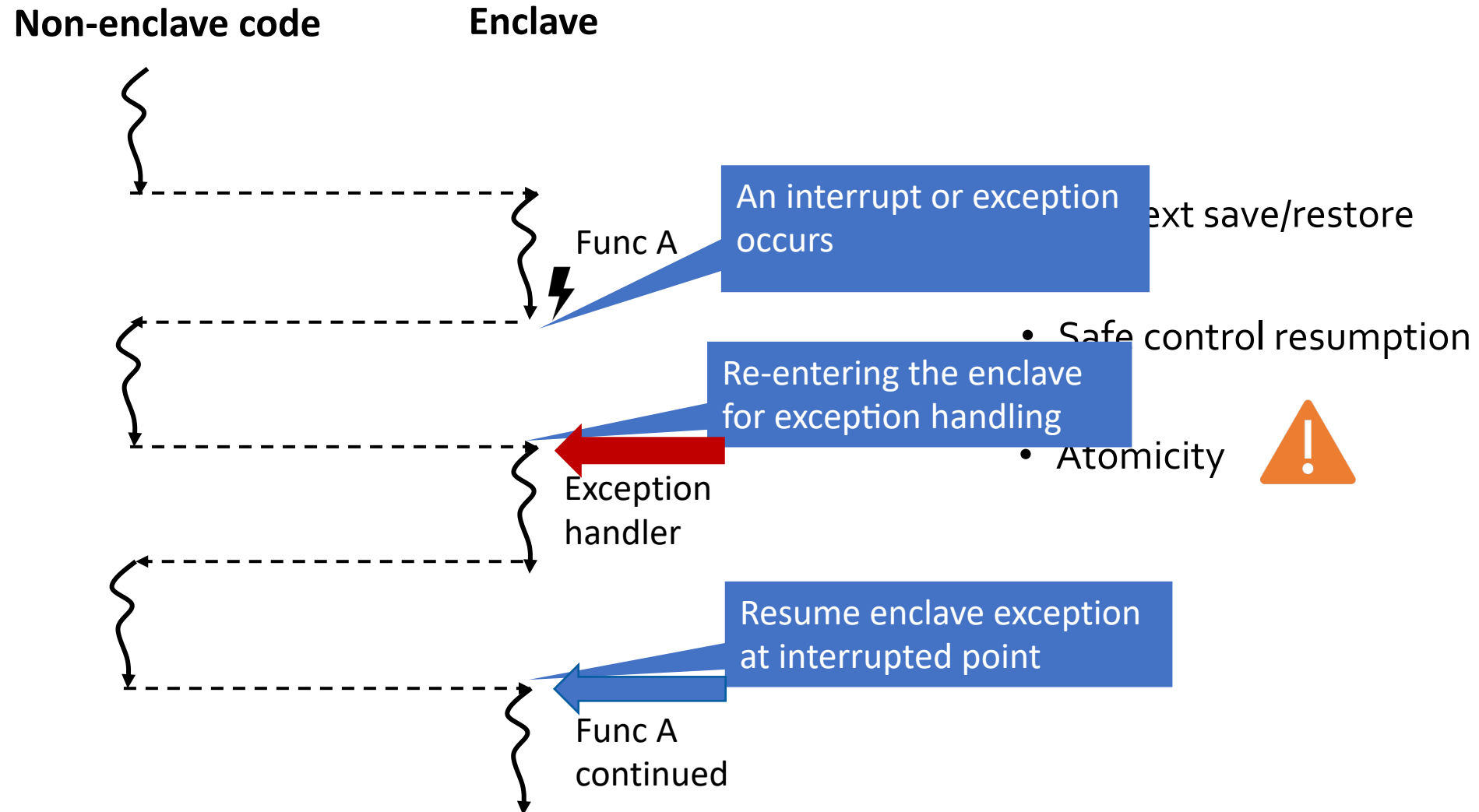
Trusted Computing Primitives

[TR'15], [Eurosys'20], [Usenix Security'22]



Smashing SGX Enclaves Using Exceptions

Intel SGX Secure Context Switching



Lack of atomicity

Asynchronous Exception Handling

Func A

```
try {  
    index = 1024 / count;  
} catch {  
    ...  
}  
...  
data = list.data;  
list = list.next;  
process(data);
```

Exception handler

```
if(divide_by_zero) {  
    error = DIVIDE_BY_ZERO;  
    IP = CATCH_BLOCK;  
}  
if(data_ready) {  
    data = read();  
    new_head.data = data;  
    new_head.next = list;  
    list = new_head;  
}
```

Lack of Atomicity Leads to Problems

Func A

```
try {  
    index = 1024 / count;  
} catch {  
    ...  
}  
...  
data = list.data;  
list = list.next;  
process(data);
```

But... new data is removed!

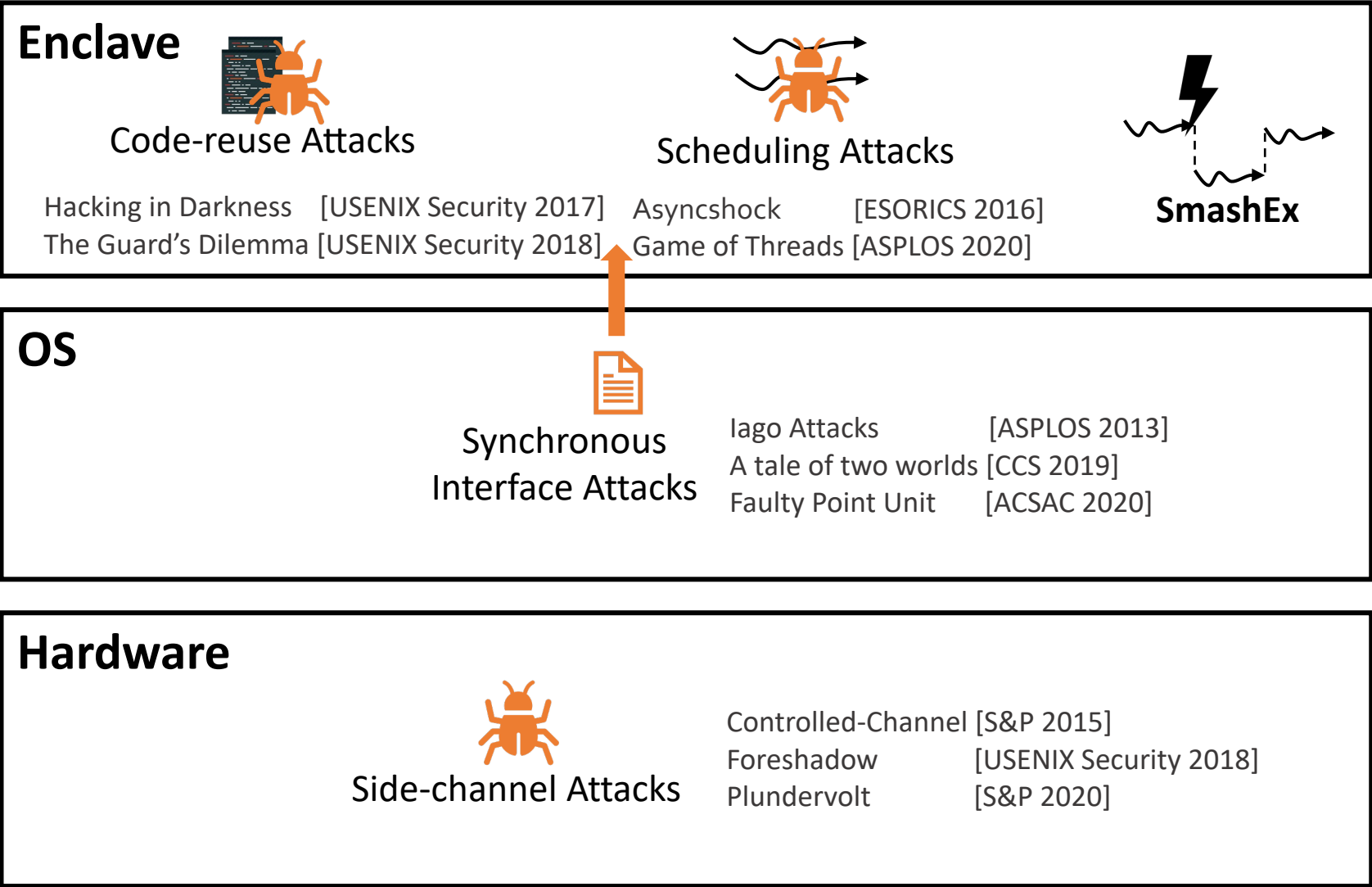
Exception handler

```
if(divide_by_zero) {  
    error = DIVIDE_BY_ZERO;  
    IP = CATCH_BLOCK;  
}  
if(data_ready) {  
    data = read();  
    new_head.data = data;  
    new_head.next = list;  
    list = new_head;  
}
```

New data added to list

Head of list has changed

Difference to Previous Work



Benign SGX Exception Handling

Control flow

Func A

```
try {  
    index = 1024 / count;  
} catch {  
    ...  
}  
...  
data = list.data;  
list = list.next;  
process(data);
```

Exception handler

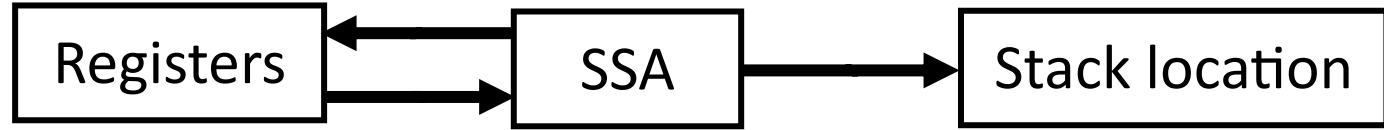
```
sp = ssa.rsp;  
info = (info*) sp;  
info.r8 = ssa.r8;  
...  
if (divide_by_zero) {  
    error = DIVIDE_BY_ZERO;  
    IP = CATCH_BLOCK;  
}
```

Set up stack for exception handling

Reuse stack location stored in SSA (RSP before exception)

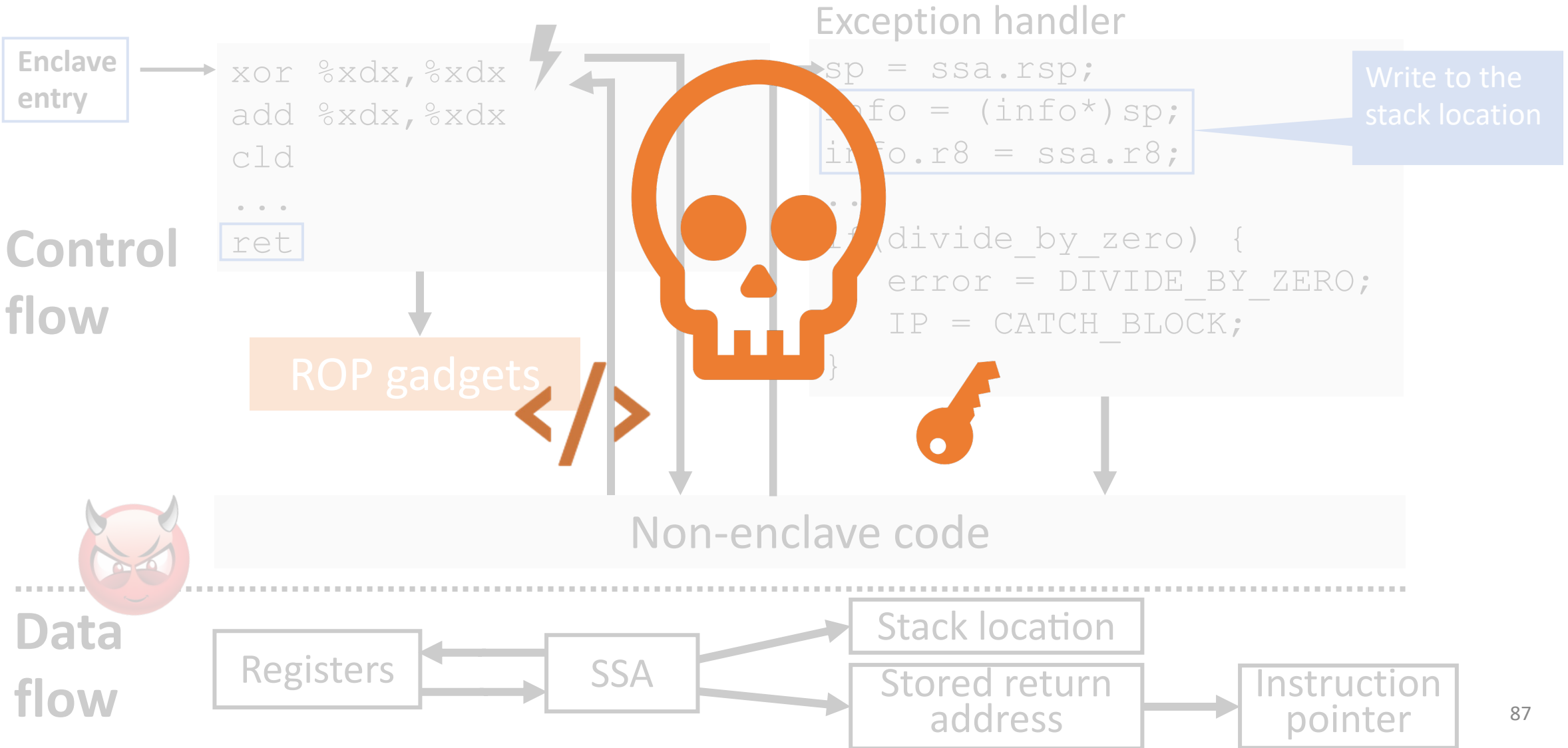
Non-enclave code

Data flow



Interesting code locations
for an attack?

Exploiting SGX Exception Handling



Inducing Exceptions at Attacker-Desired Locations



- Prepare payload
- Set page non-executable (mprotect)
- Enter enclave
- Restore page permissions (mprotect)
- Re-enter enclave



Enclave entry

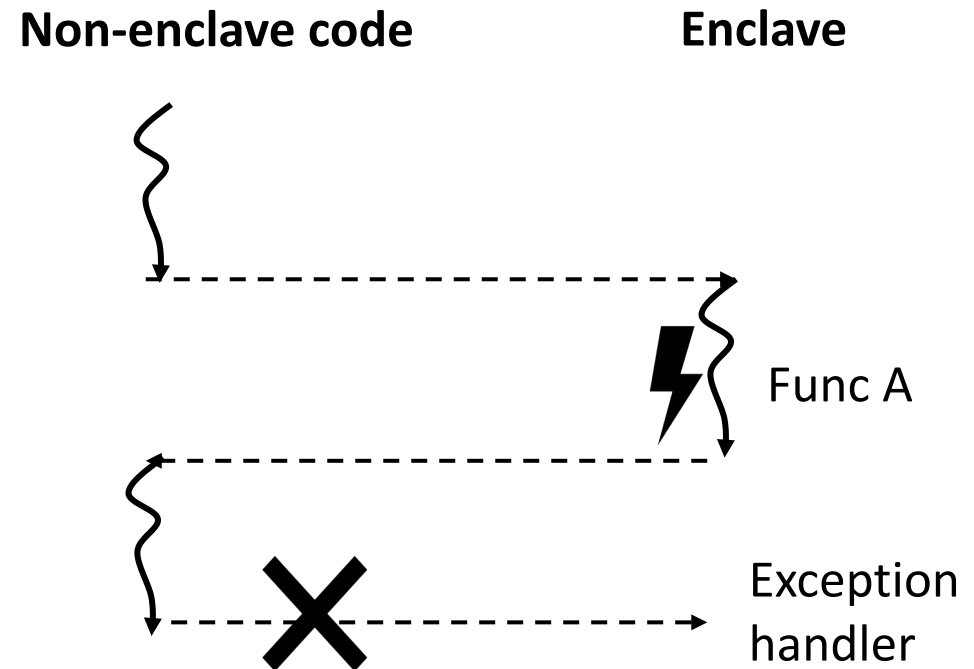
```
xor %edx, %edx  
add %edx, %edx  
cld  
...
```

Missing Re-entry Prevention in Critical Sections

Traditional programming models



Next best option on SGX

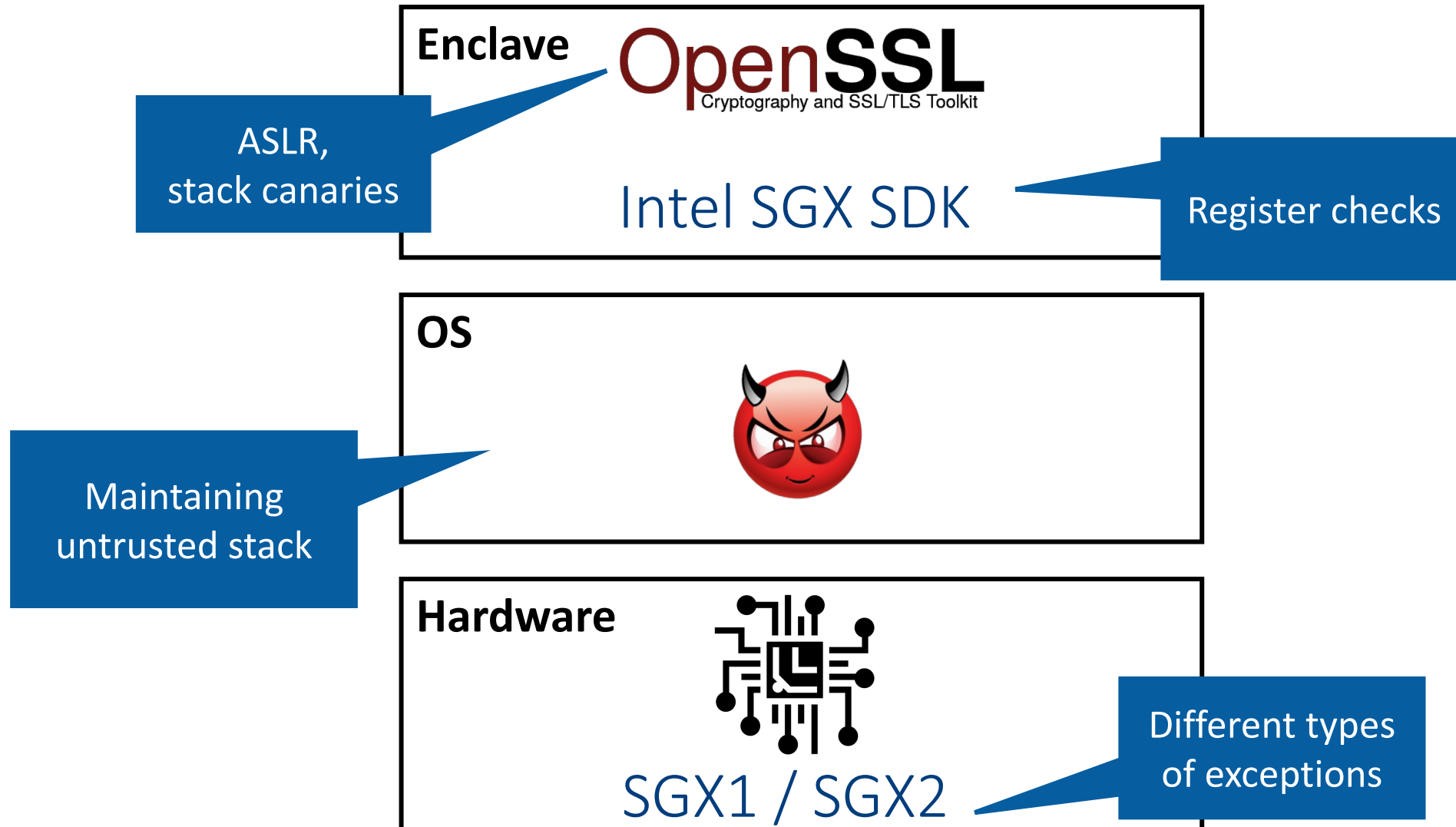


38.8.4 Number of State Save Area Frames (NSSA)

NSSA specifies the number of SSA frames available for this TCS. There must be at least one available SSA frame when EENTER-ing the enclave or the EENTER will fail.

— Intel® 64 and IA-32 Architectures Software Developer's Manual

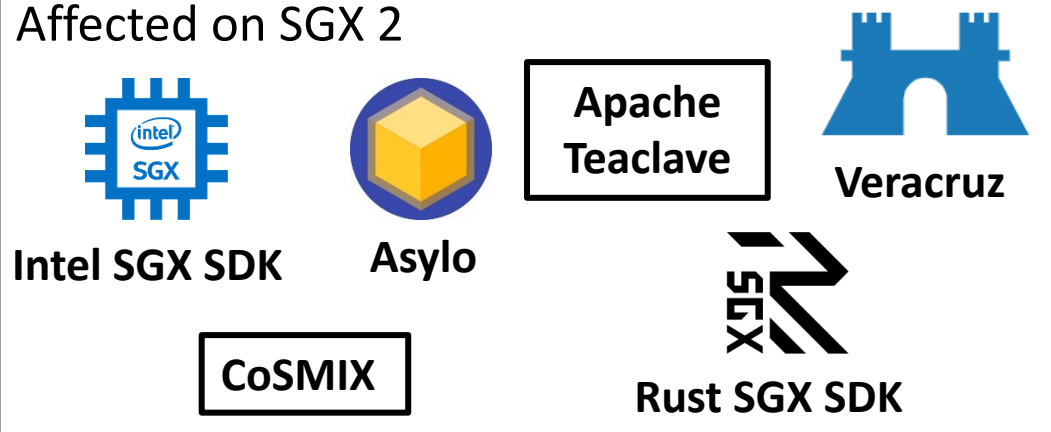
End-to-end attacks: Devil is in the details



Which Runtimes are Vulnerable?

- Examined 14 runtimes on SGX1 and SGX
- Created PoCs for 9

Affected on SGX 2



Intel SGX SDK Asylo Apache Teaclave Veracruz

CoSMIX Rust SGX SDK

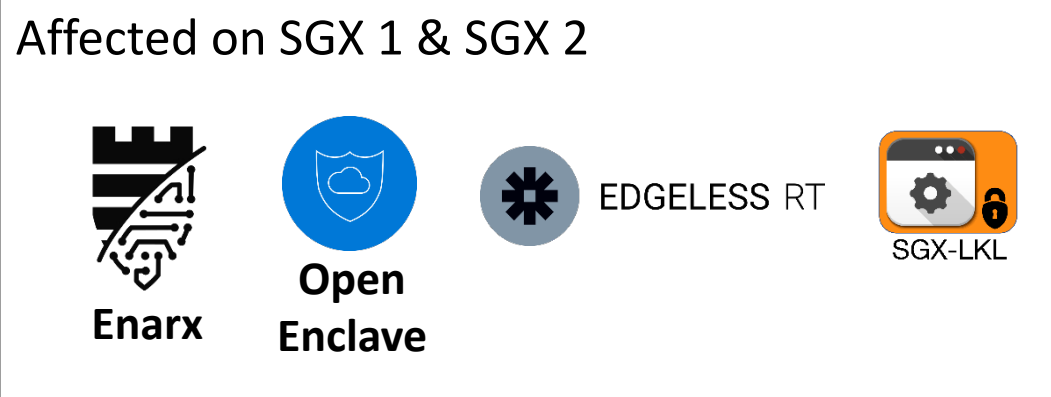
Unaffected



GRAMINE INCLAVARE CONTAINERS

Fortanix® EDP Ratel

Affected on SGX 1 & SGX 2



Enarx Open Enclave EDGELESS RT SGX-LKL

Responsible Disclosure

May 2021

Notified Intel and Microsoft.

October 2021

Intel assigned CVE and released advisory.

SmashEx released publicly.

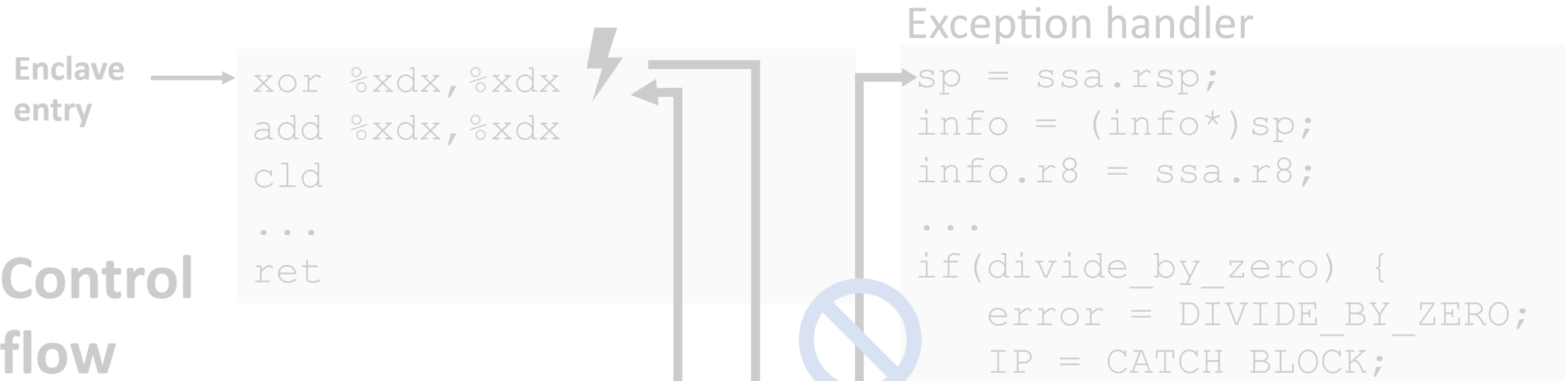
July 2021

Intel and Microsoft released patches.

Microsoft assigned CVE and released advisory.

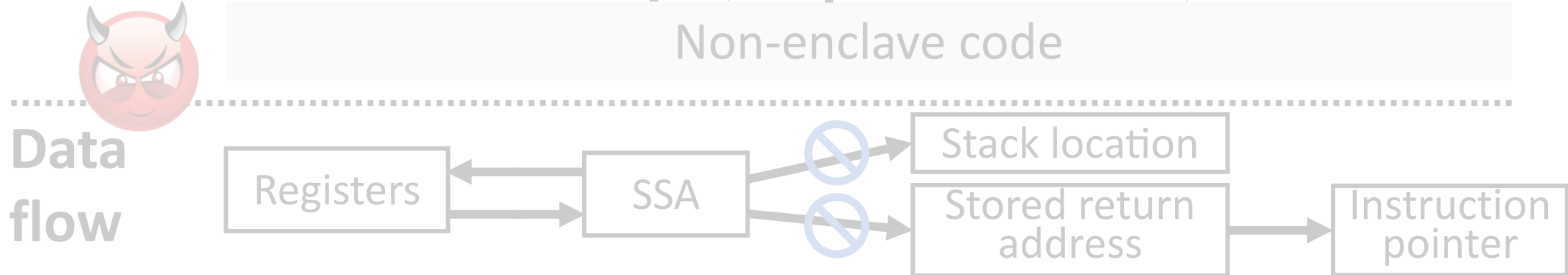
How should we defend against
SmashEx?

Disabling Exception Handling



Limits enclave functionality

Non-enclave code



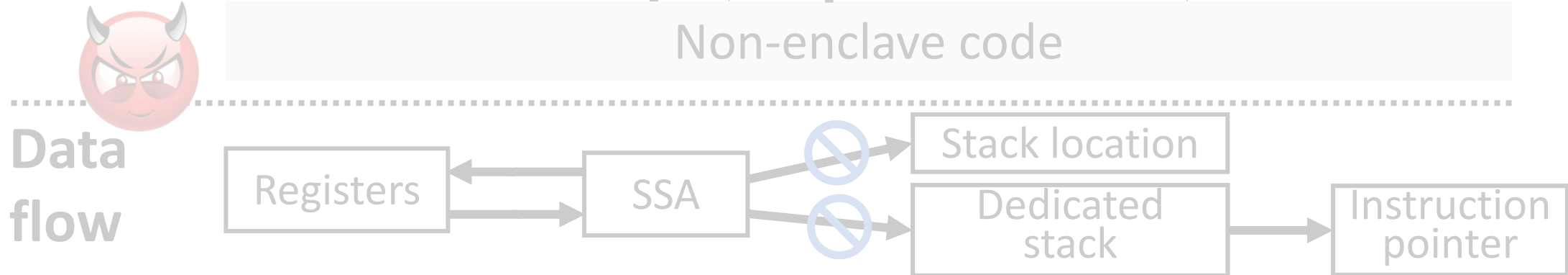
Dedicated Exception Handling Stack

Ratel

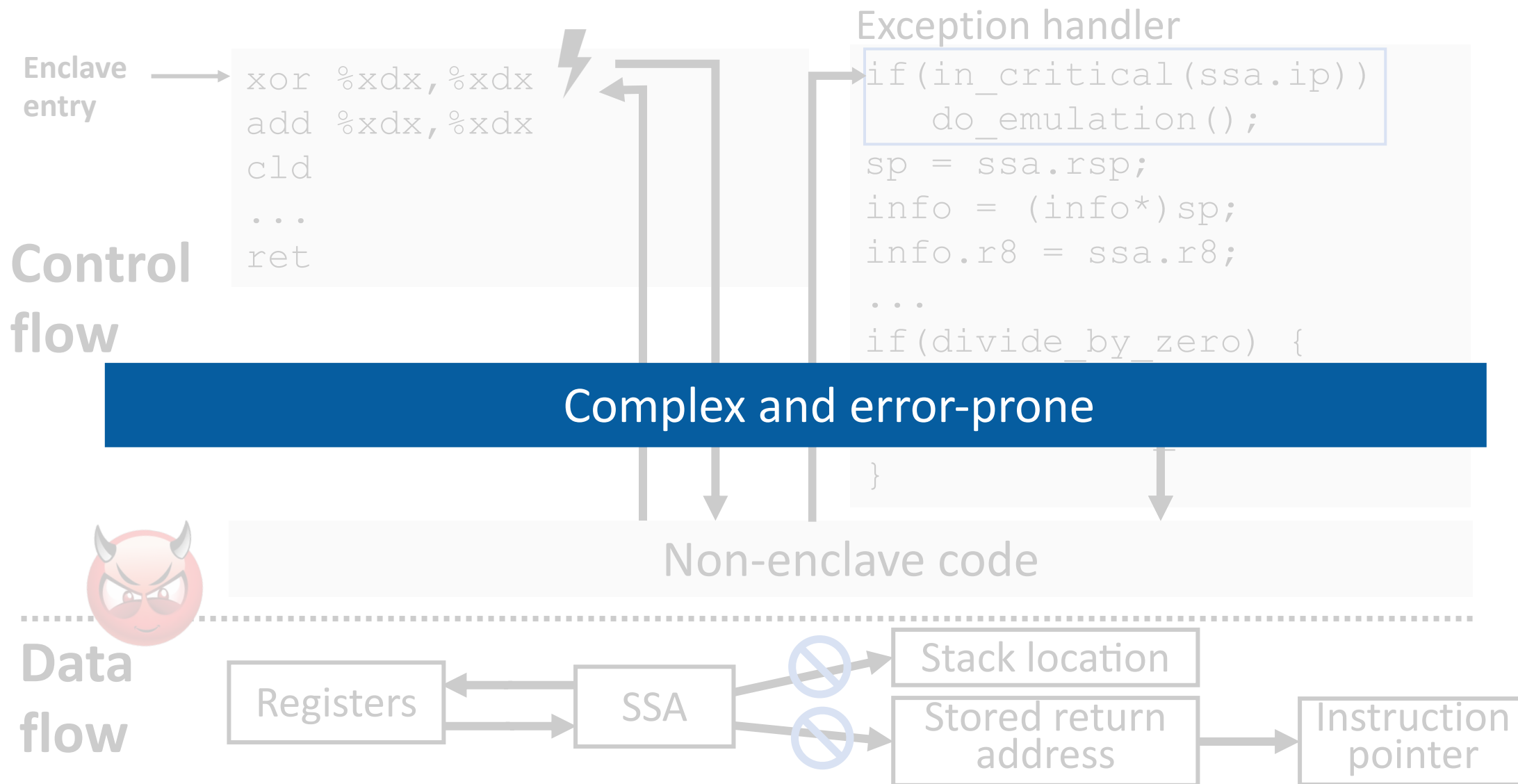


Limits enclave scalability and compatibility

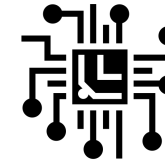
Non-enclave code



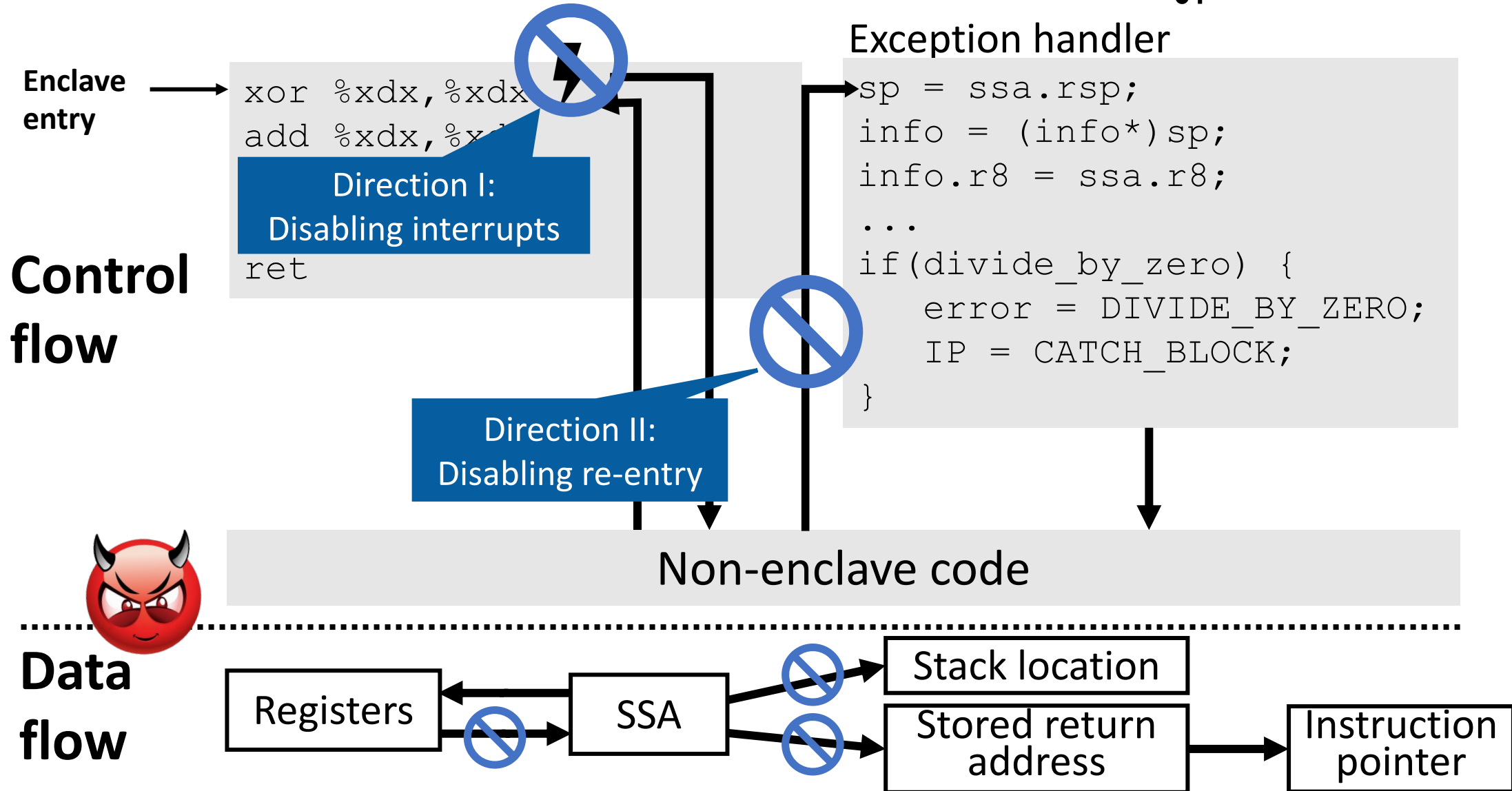
Re-entrant Runtime Design



Hardware Atomicity Support



interrupt_flag = 1



SmashEx Take-aways



- New attack that exploits lack of atomicity on SGX
- Breaks both confidentiality and integrity
 - Does not assume bugs in the enclave code
- Affects 10/14 popular enclave runtimes
- Call for
 - Careful runtime design and implementation
 - Atomicity support in future SGX designs

Putting it all together

New Applications [Arxiv'18], [ICDCS'19]

Secure Computation [CCS'13]

Analysis & hardening

[PLDI'14], [FSE'15], [NDSS'19], [CCS'20]

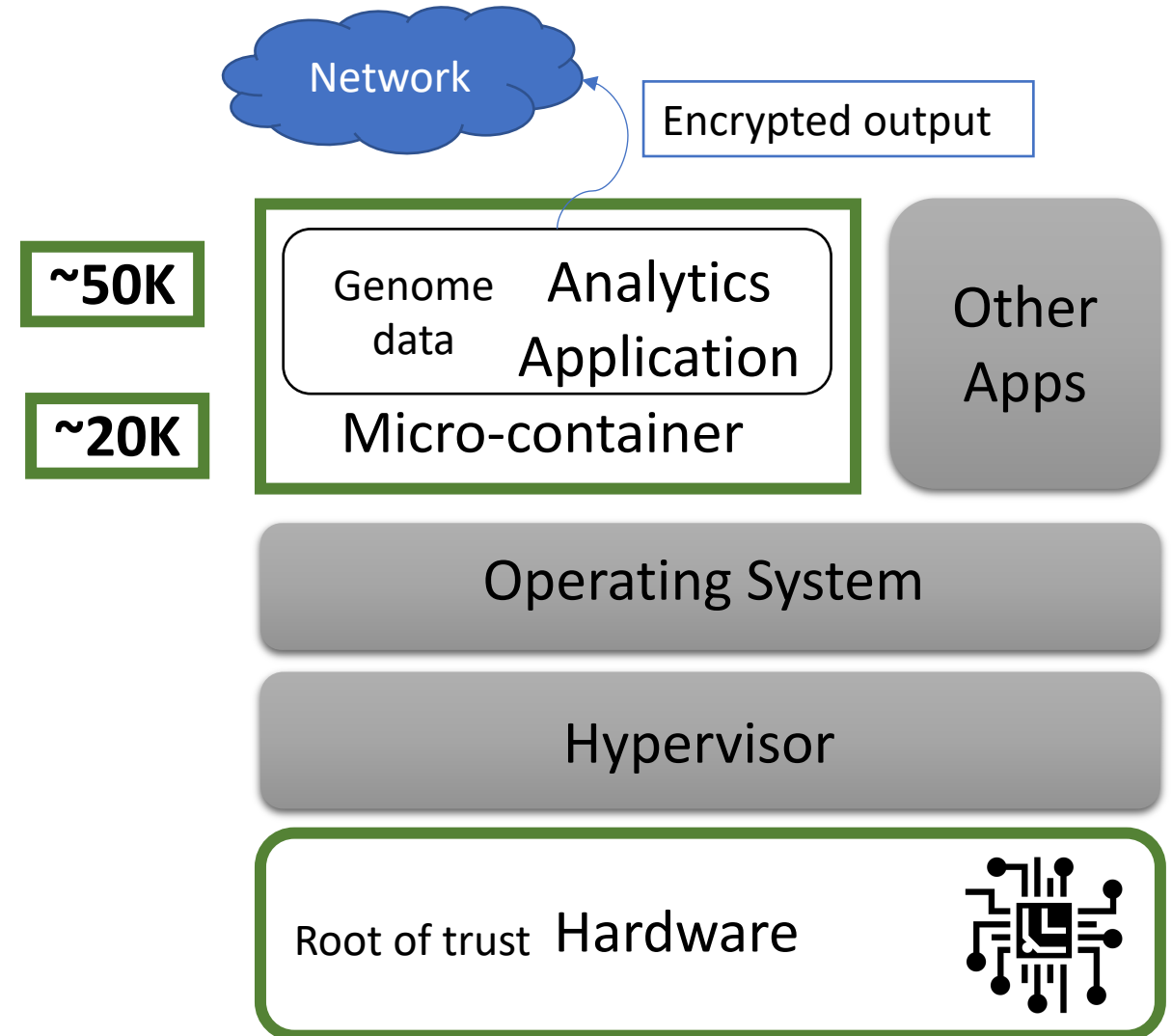
Rich functionality [NDSS'17], [TOPS'22]

Formal verification [Usenix Security'20]

Attacks & Defenses [AsiaCCS'16], [CCS'21]

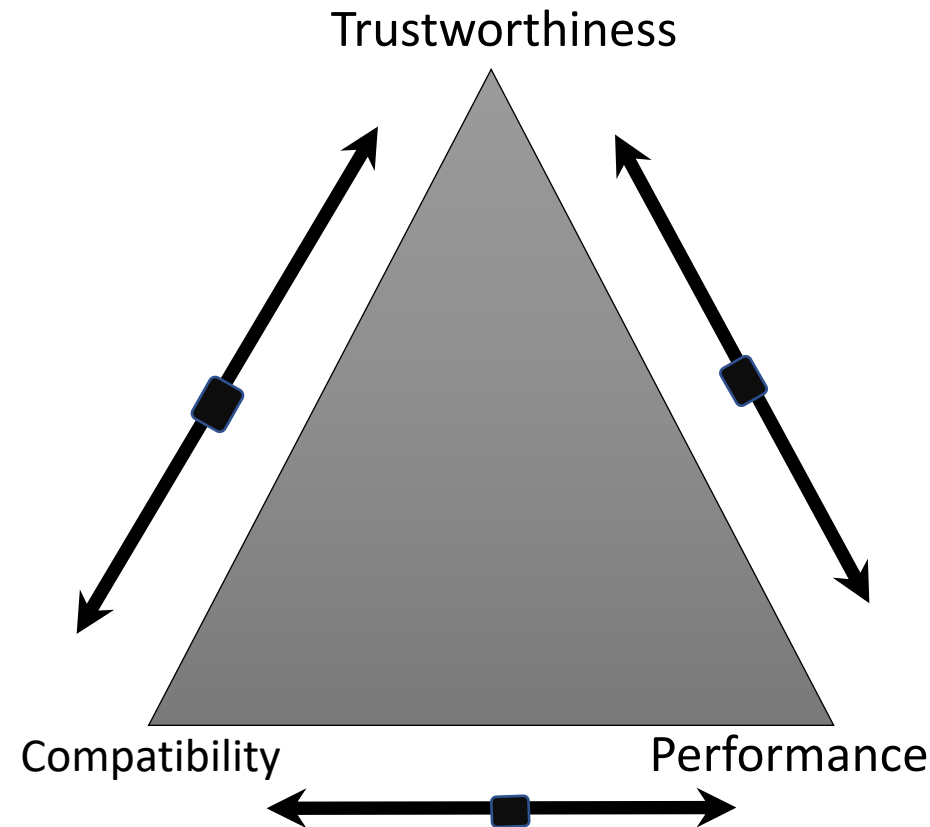
Trusted Computing Primitives

[TR'15], [Eurosys'20], [Usenix Security'22]



Summary

- Next-generation computing stack using TEEs
 - **1-3 orders of magnitude** reduction of trusted code
- Big questions ahead:
 - Resolving the trade-offs
 - Scaling up to cloud model & tackling the last mile
- Existing TEEs have fundamental limitations
- So, build new TEE designs!



Shweta Shinde

<https://shwetashinde.org>

References

- **Panoply: Low-TCB Linux Applications With SGX Enclaves**
Shweta Shinde, Dat Le Tien, Shruti Tople, Prateek Saxena, Network and Distributed System Security Symposium (**NDSS 2017**)
- **BesFS: A POSIX Filesystem for Enclaves with a Mechanized Safety Proof**
Shweta Shinde, Shengyi Wang, Pinghai Yuan, Aquinas Hobor, Abhik Roychoudhury, Prateek Saxena, USENIX Security Symposium (**USENIX Security 2020**)
- **Keystone: An Open Framework for Architecting Trusted Execution Environments**
Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, Dawn Song, European Systems Conference (**EuroSys 2020**)
- **Elasticlave: An Efficient Memory Model for Enclaves**
Jason Zhijingcheng Yu, Shweta Shinde, Trevor E. Carlson, Prateek Saxena, USENIX Security Symposium (**USENIX Security 2022**)
- **SmashEx: Smashing SGX Enclaves Using Exceptions**
Jinhua Cui, Jason Zhijingcheng Yu, Shweta Shinde, Prateek Saxena, Zhiping Cai, ACM Conference on Computer and Communications Security (**CCS 2021**)