Better Foundations for Secure Software using Trusted Hardware & Verification

Shweta Shinde
ETH Zürich

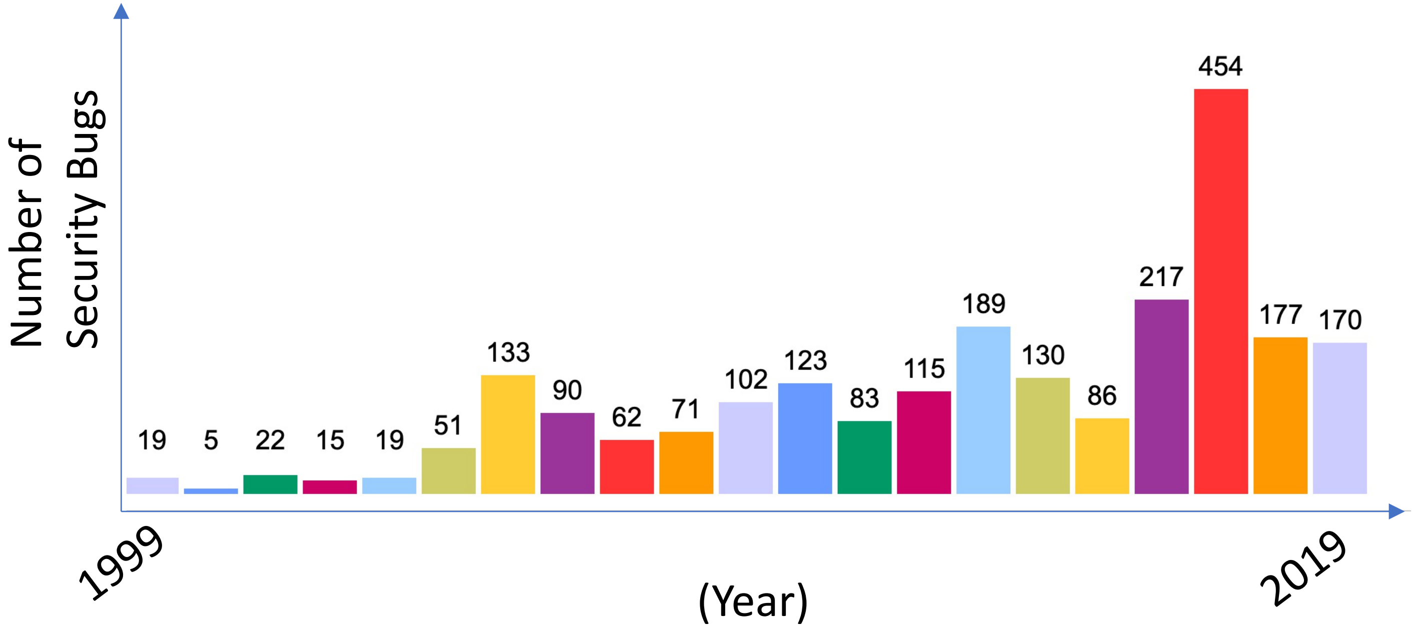
Security breaches are on the rise

37 billion records exposed through data breaches in 2020

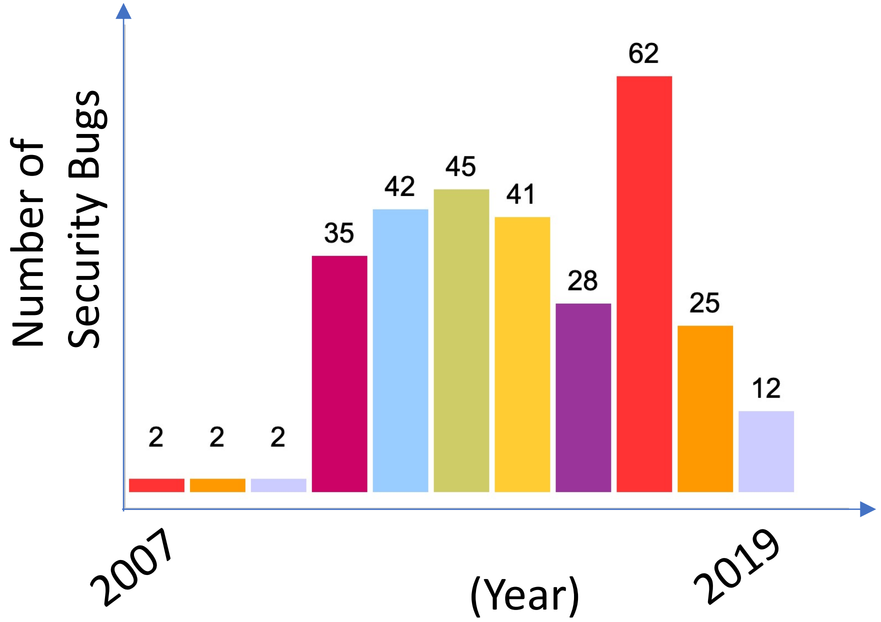
Average of 1-25 Bugs per 1000 lines of code

The new normal: Hundreds of bugs a year in Linux

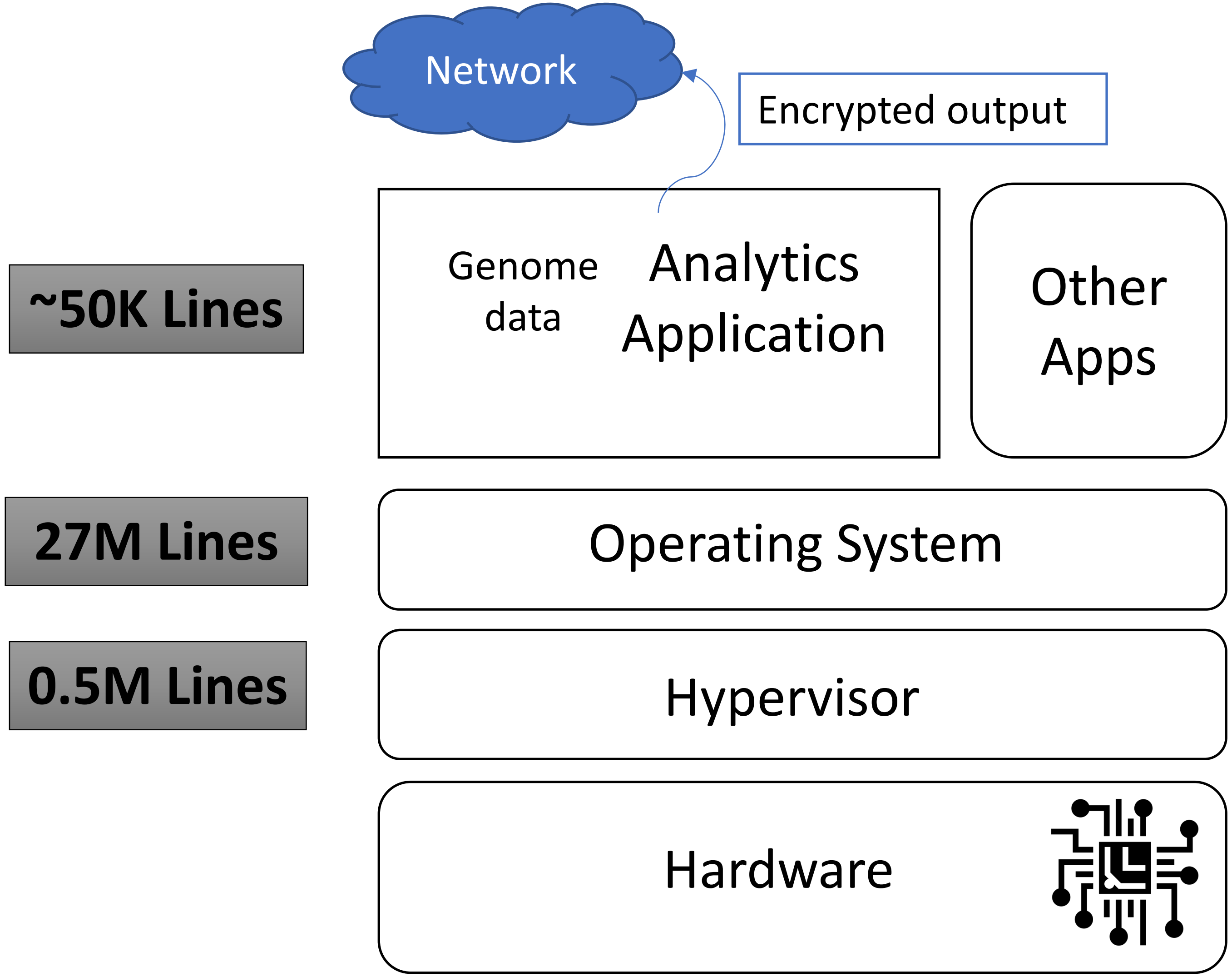
Operating System (Linux Kernel)
27 Million Lines



Hypervisor (XEN)
0.5 Million Lines

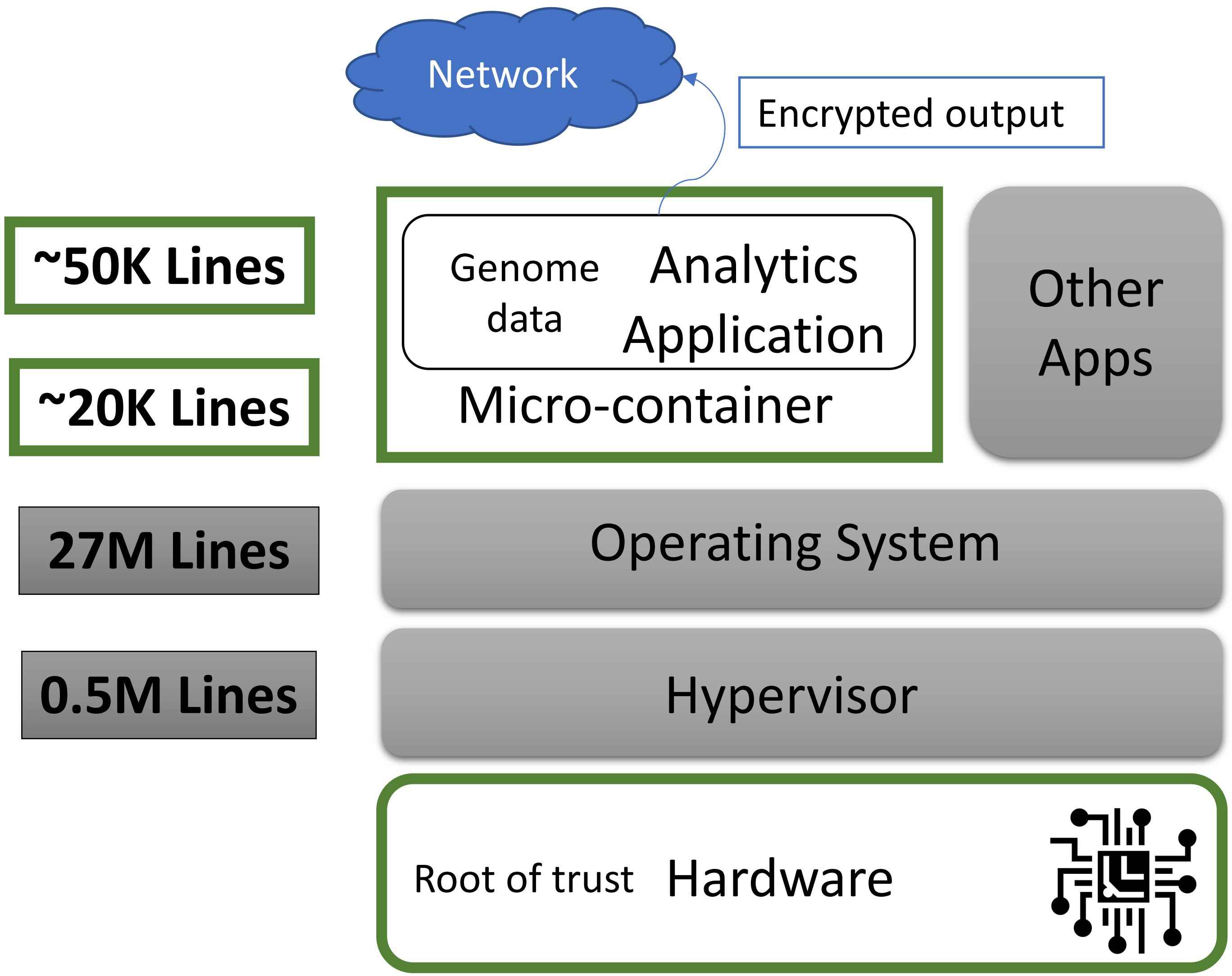


Current computing stack is prone to attacks



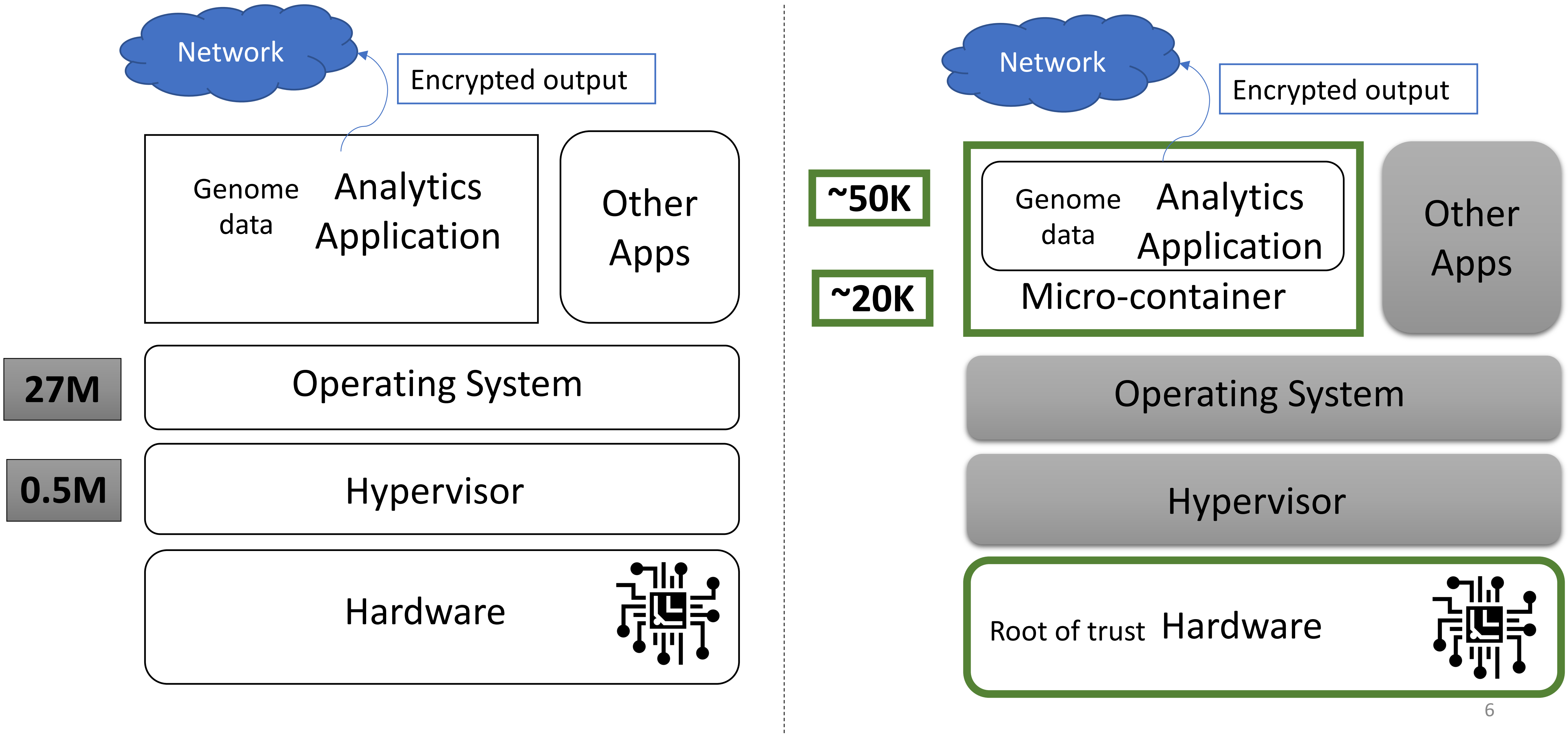
- Encryption or other sophisticated techniques at the application layer
- Bug in lower layers → Compromise the security of the app
- Large size → High probability

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



Building the components of this stack

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

Secure Computation [CCS'13]

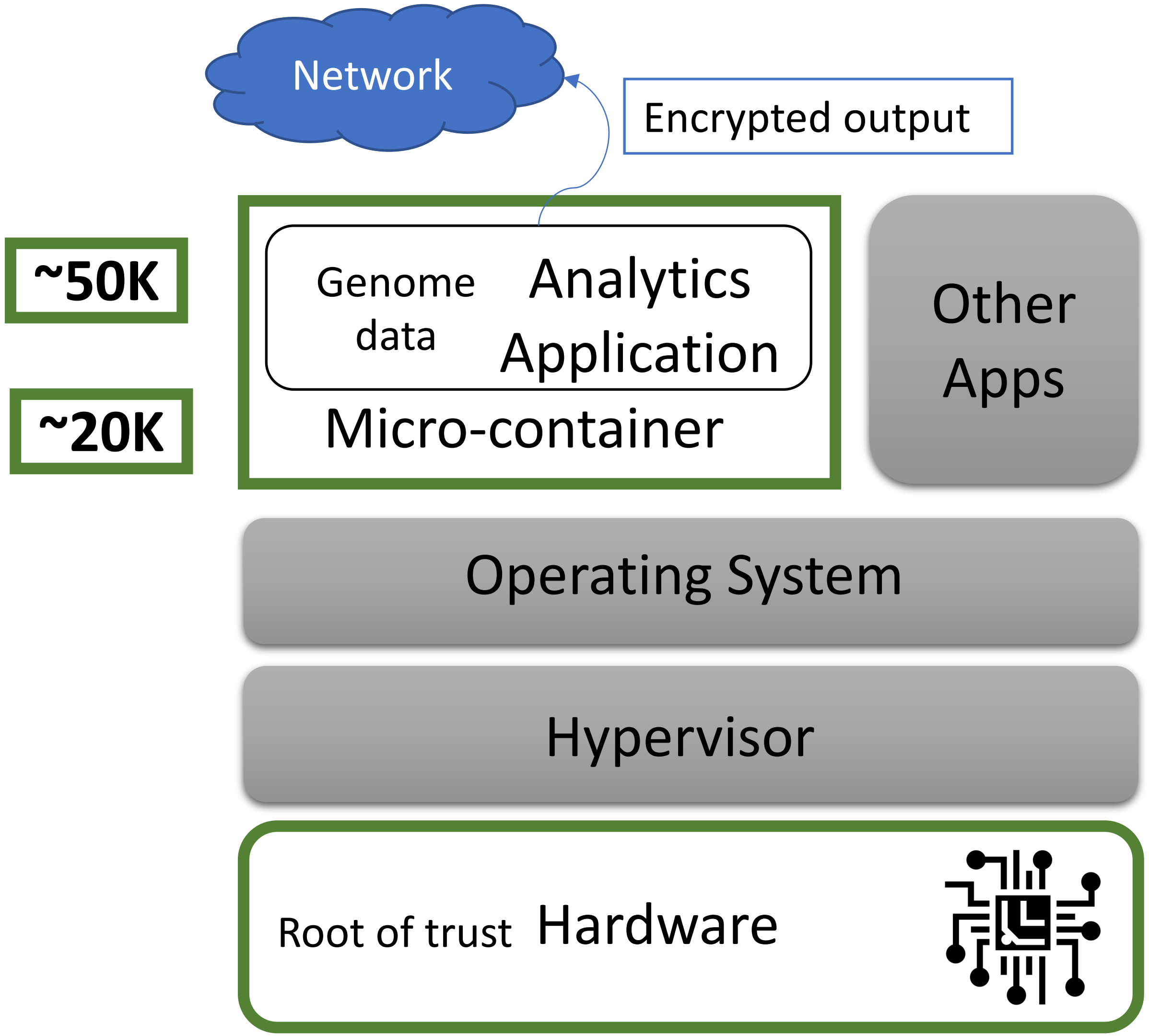
Analysis & hardening
[PLDI'14], [FSE'15], [NDSS'19], [CCS'20]

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

Formal verification [Usenix Security'20]

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

Trusted Computing Primitives
[TR'15], [Eurosys'20]



Practical Relevance: Initial Adoption

New Applications
Microsoft, Largest Asia-Pacific ISP

Secure Computation
SAP Labs

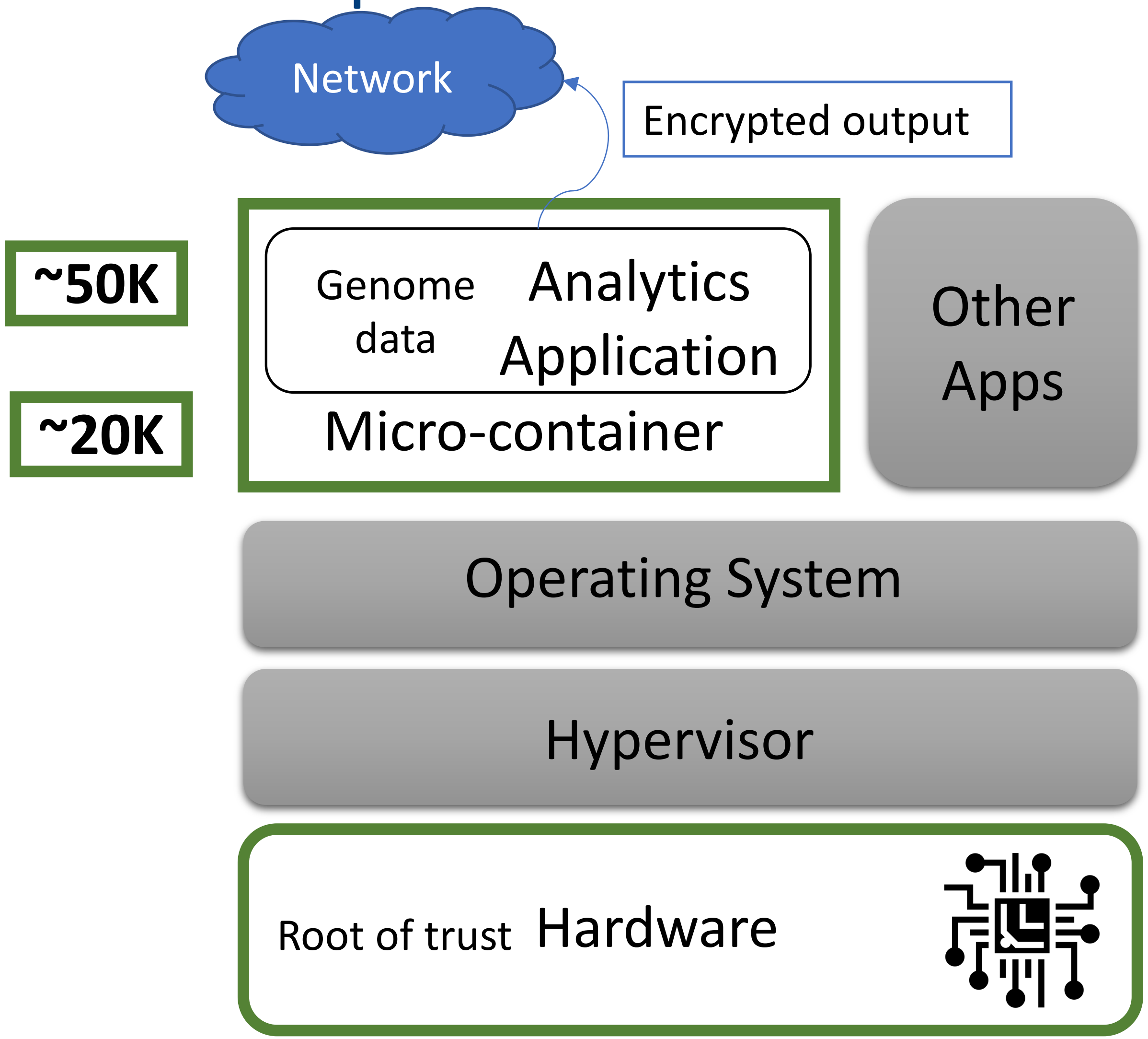
Analysis & hardening
Dexecure

Rich functionality
Anqlave, Anquan, Community

Formal verification
Intel, Google, Microsoft, Anqlave

Attacks & Defenses
Intel, Community

Trusted Computing Primitives
Qualcomm, Seagate, Baidu, Community



1st Component of this stack

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

Secure Computation [CCS'13]

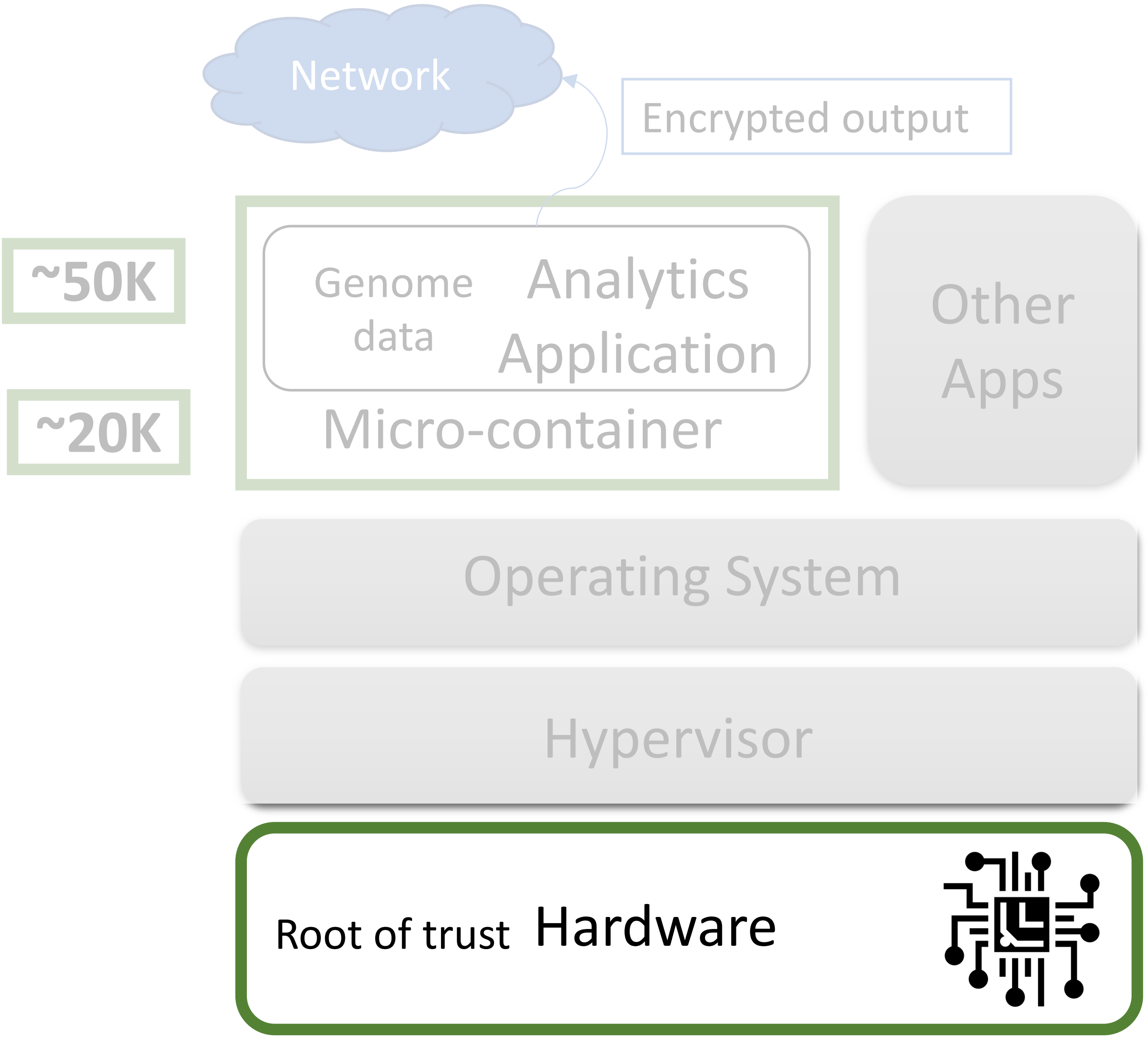
Analysis & hardening
[PLDI'14], [FSE'15], [NDSS'19], [CCS'20]

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

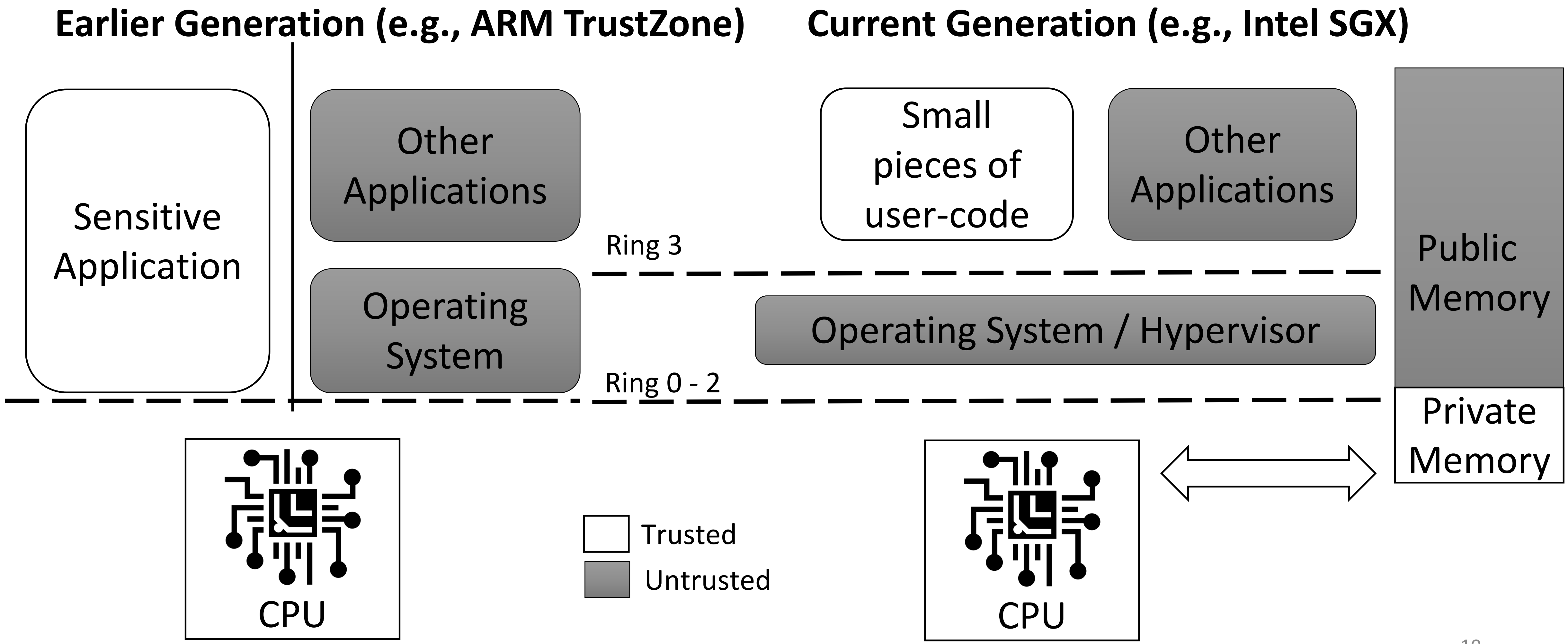
Formal verification [Usenix Security'20]

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

Trusted Computing Primitives
[TR'15], [Eurosys'20]



Trusted Execution Environments (TEEs)



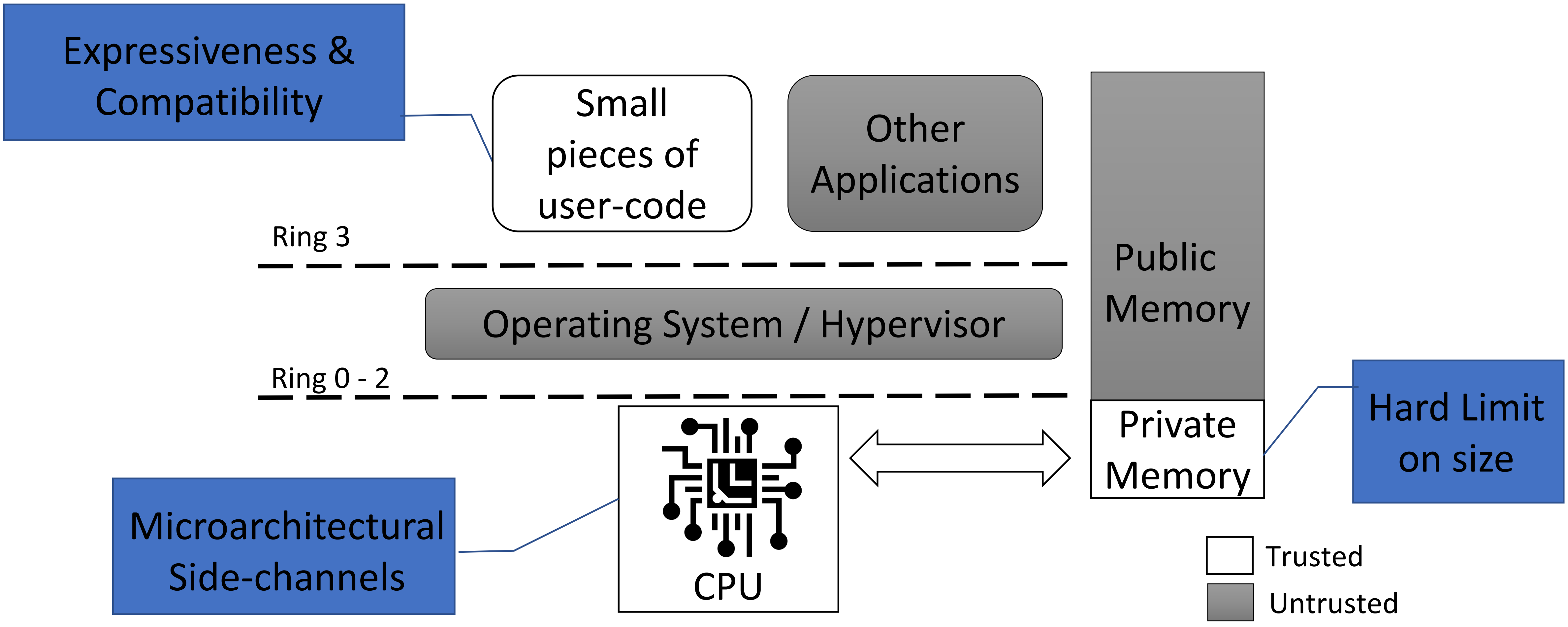
Inflexible Design & Closed Implementation

- TEEs in commercial hardware: Intel SGX, ARM TrustZone, AMD SEV

- One particular design point in the space
  - Intel SGX – small server/desktop apps (e.g., DRM, cryptography)
  - ARM TZ – vendor-provisioned mobile apps (e.g., fingerprint, ledger)
  - AMD SEV – full VM isolation only (e.g., cloud computing)

- Implemented on closed-source hardware
  - Slow iteration dictated by a company
  - Adding new features/defenses is cumbersome

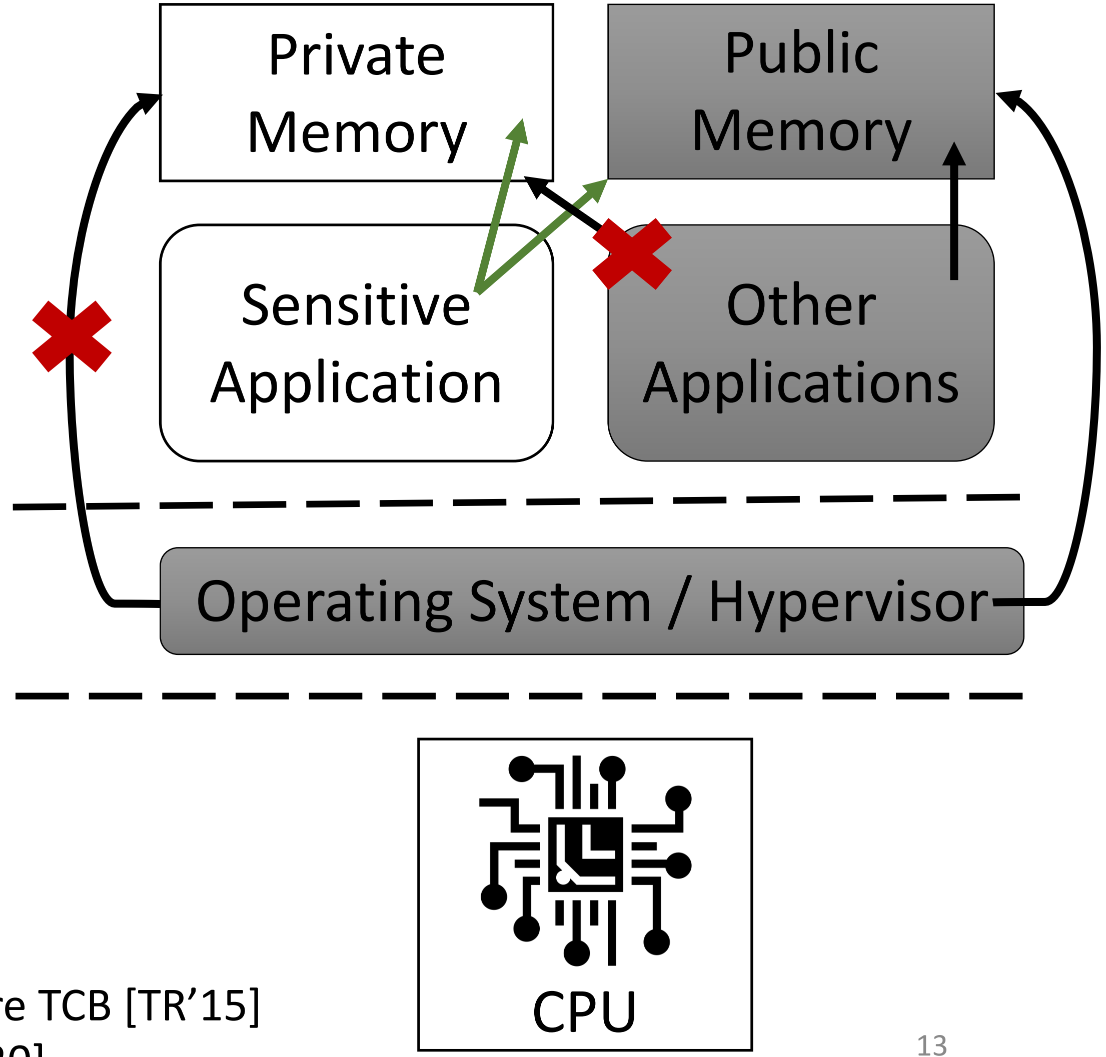
Limitations of Commercial TEEs



Binary Compatibility For SGX Enclaves [arXiv'20]

Better TEEs

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



PodArch: Protecting Legacy Applications with a Purely Hardware TCB [TR'15]
Keystone: An Open Framework for Architecting TEEs [EuroSys'20]

Focus on commercial TEEs (e.g., Intel SGX), since they are widely available

2nd component of this stack

New Applications [Arxiv'18], [ICDCS'19]
Secure Computation [CCS'13]

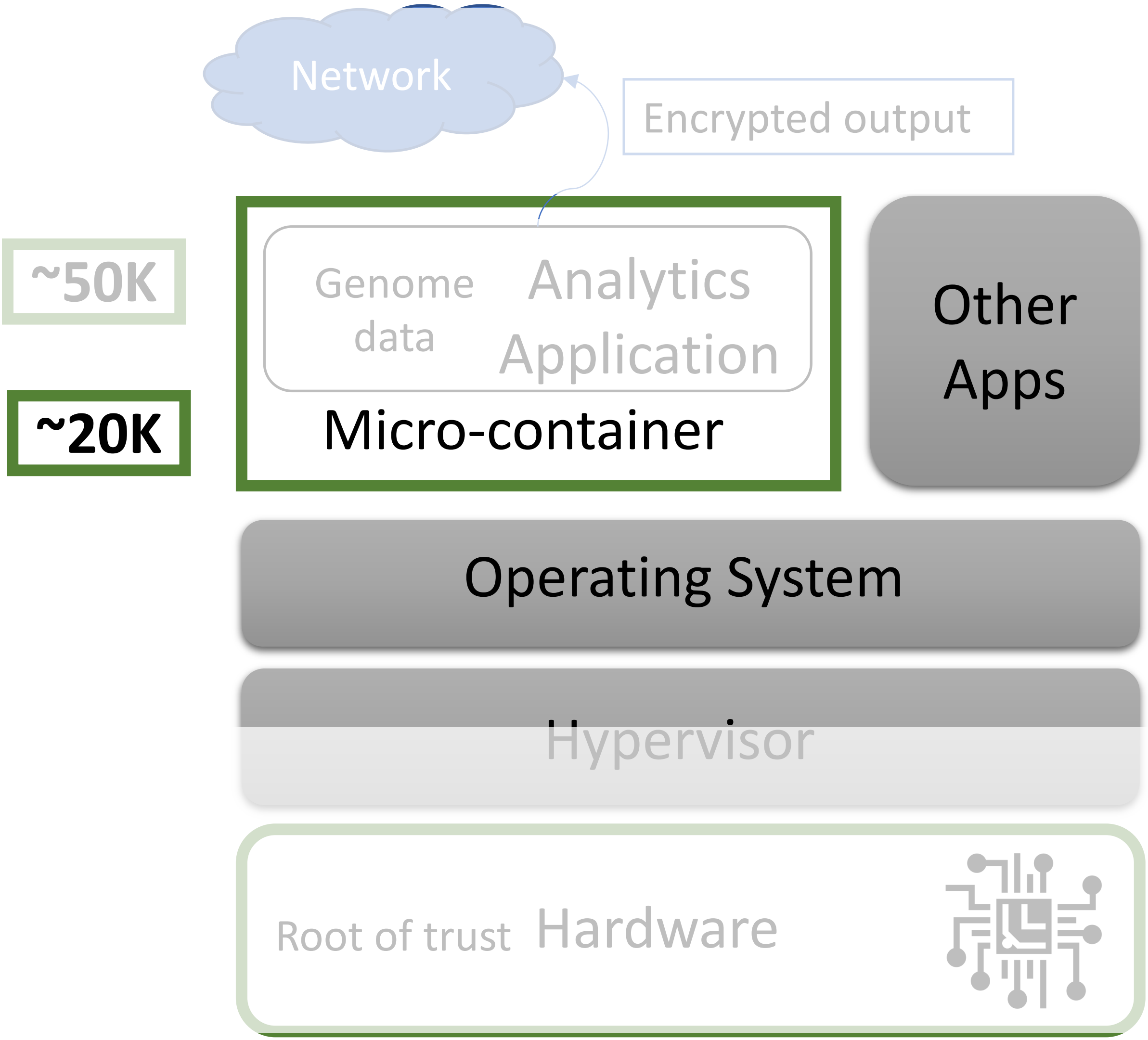
Analysis & hardening
[PLDI'14], [FSE'15], [NDSS'19], [CCS'20]

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

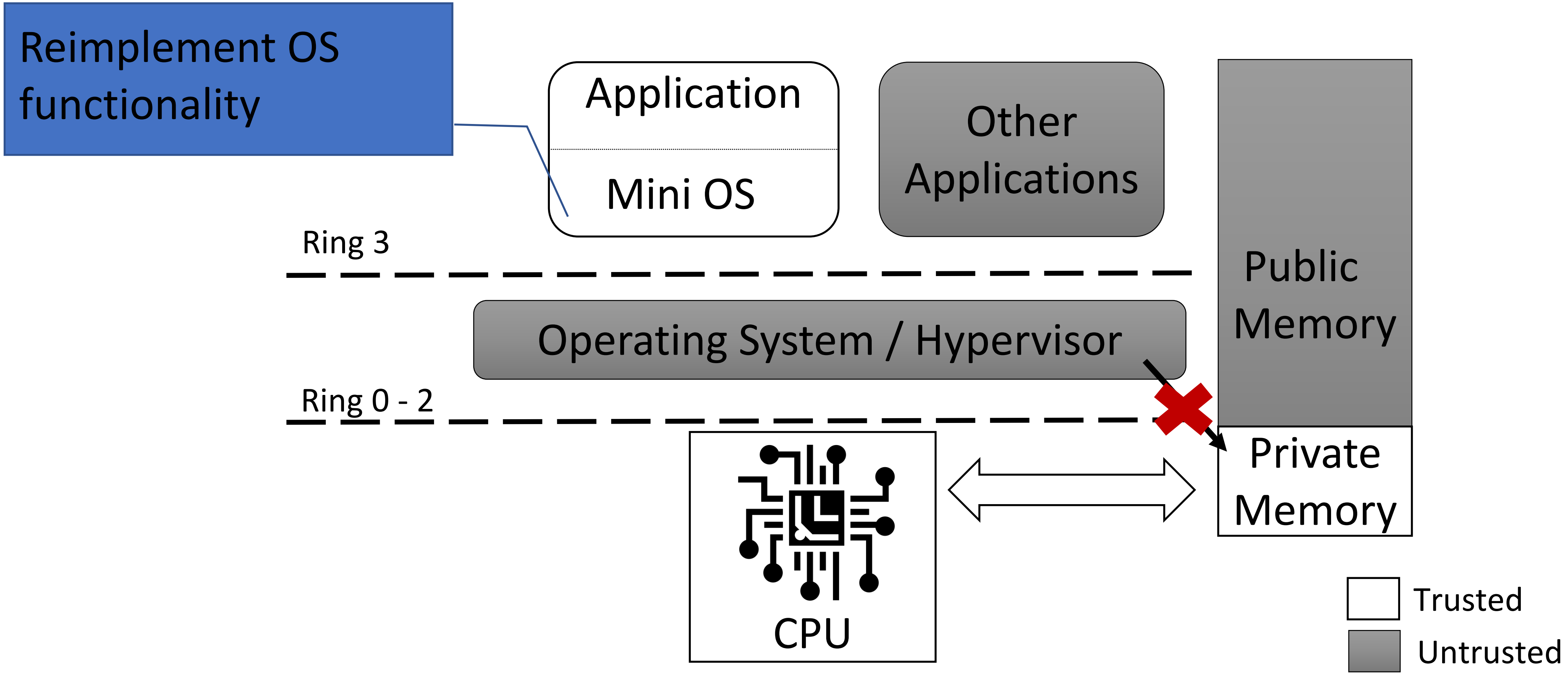
Formal verification [Usenix'20]

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

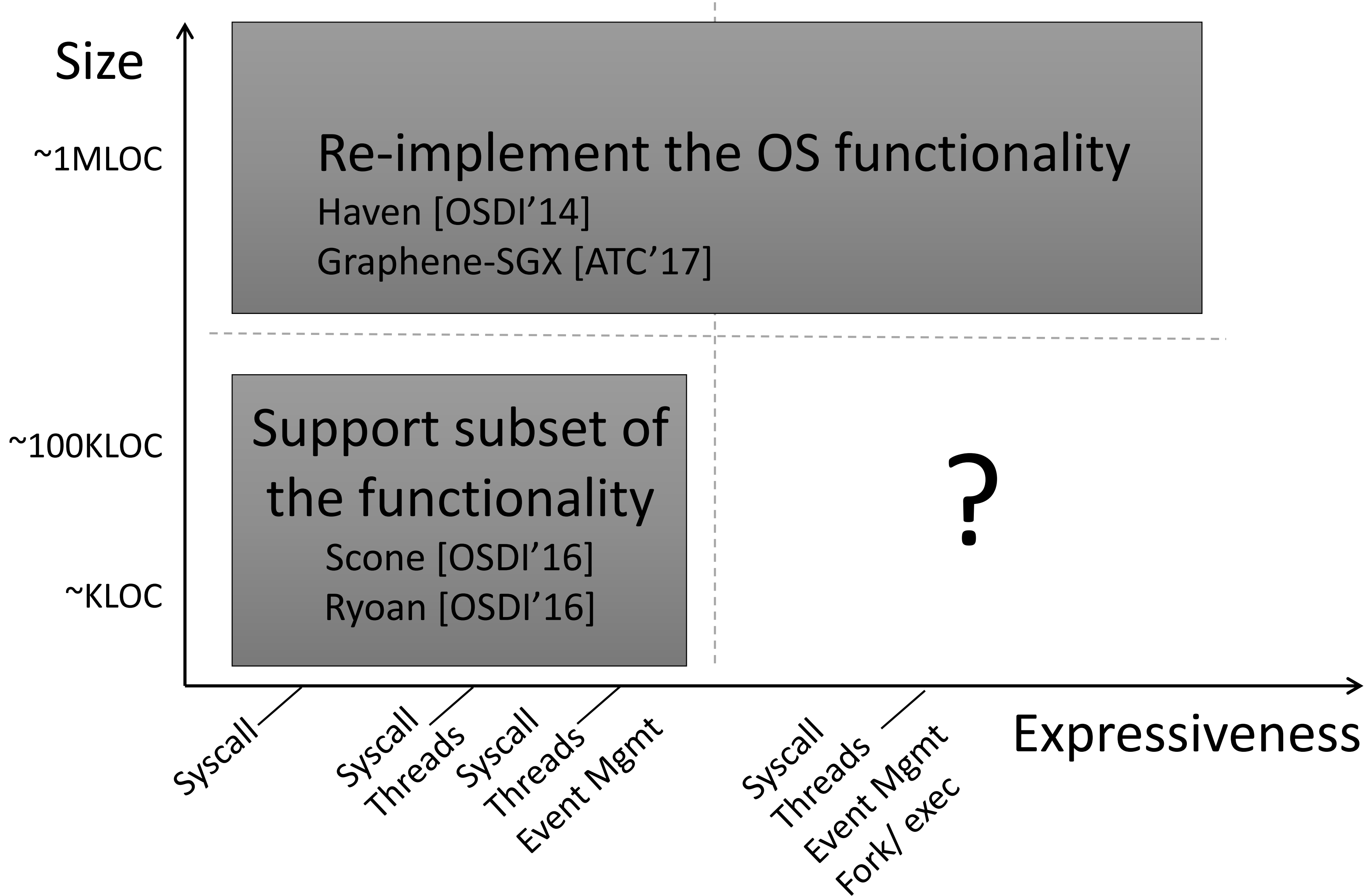
Trusted Computing Primitives
[TR'15], [Eurosys'20]



Adding Expressiveness to Commercial TEEs

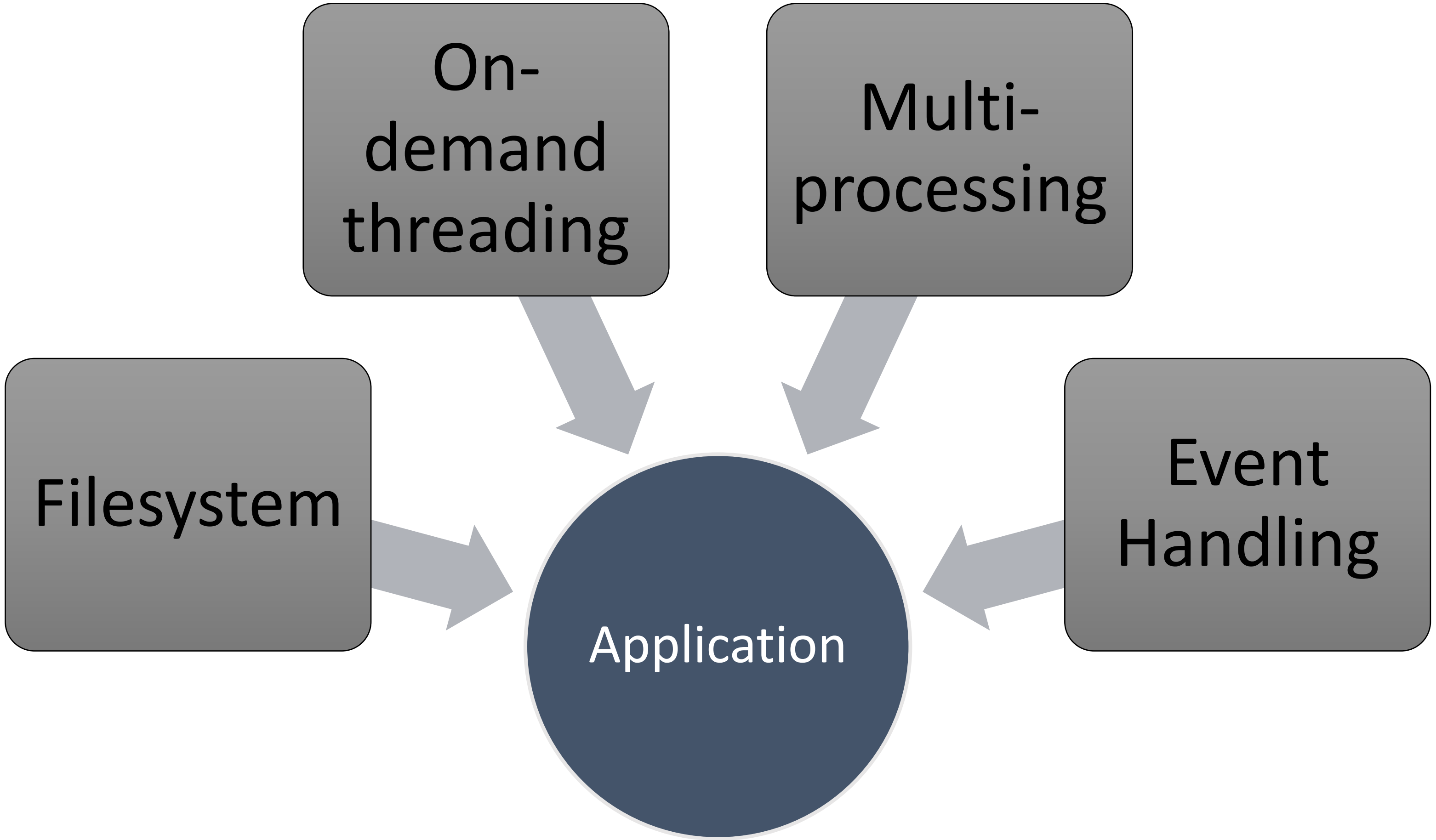


Code Size & Expressiveness Trade-off

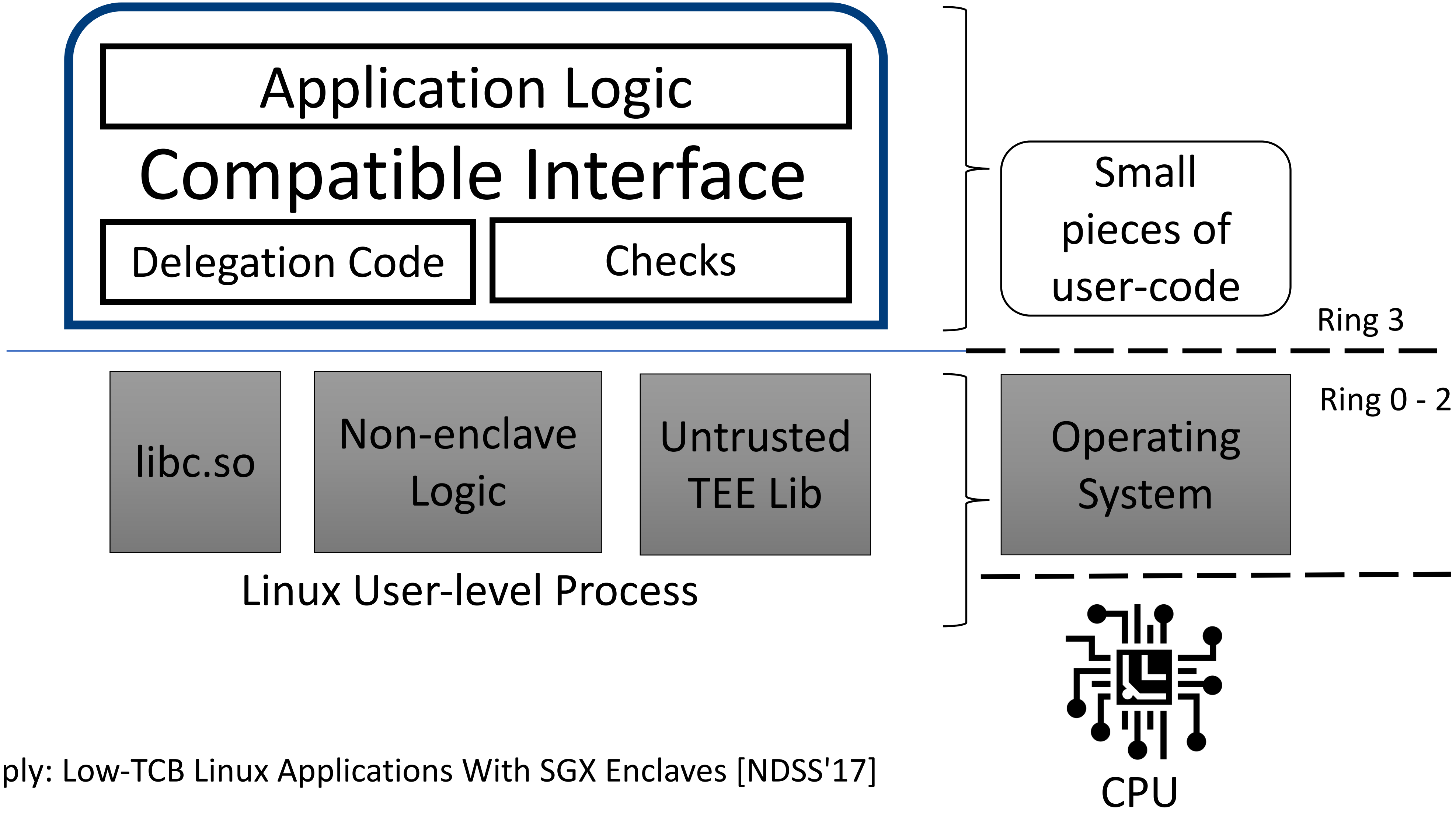


Challenge I: Expressiveness

Delegate rather than emulate



Building micro-container abstractions for TEEs

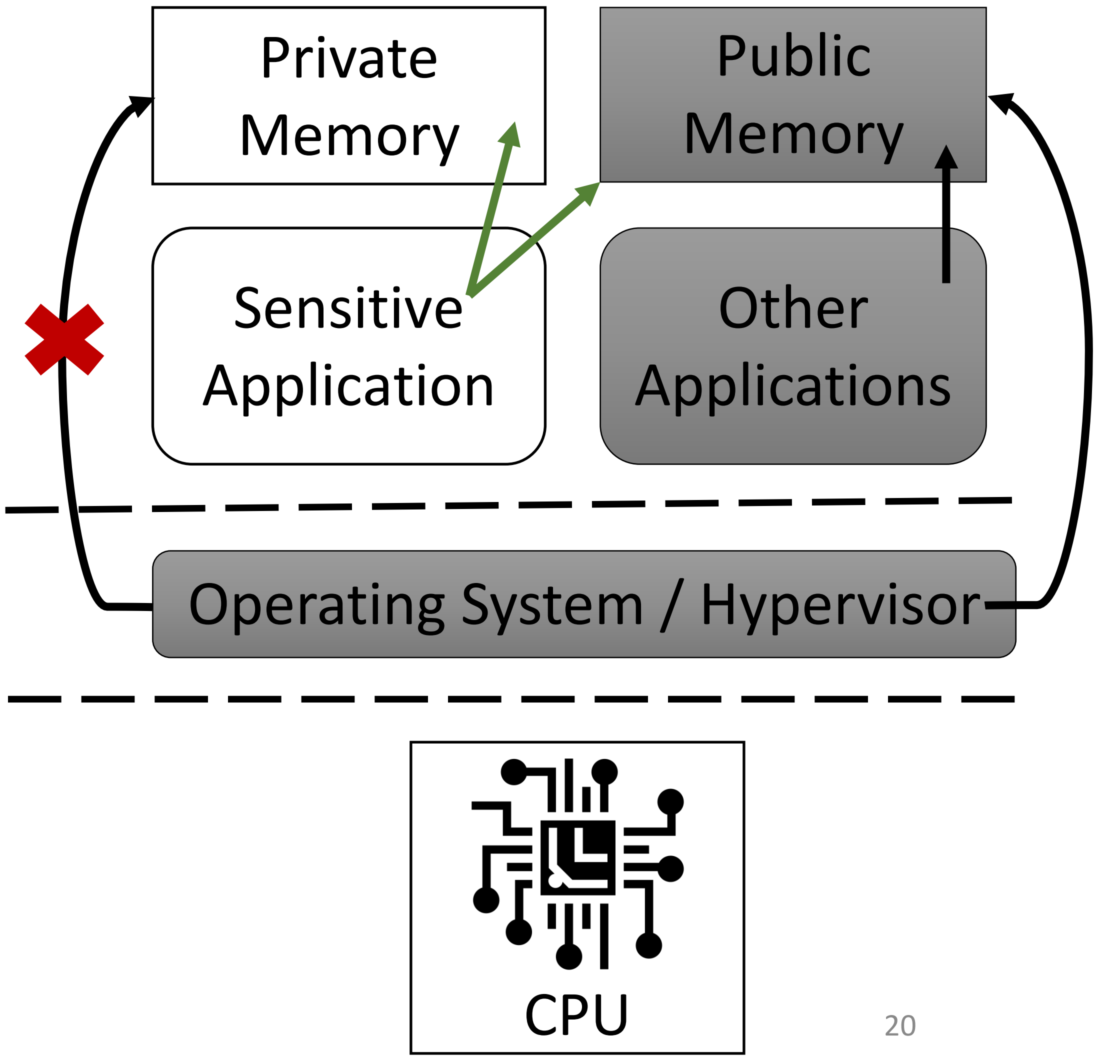


Panoply: Low-TCB Linux Applications With SGX Enclaves [NDSS'17]

Challenge II: Delegation with isolation

- Two memory model:
  - private and public memory

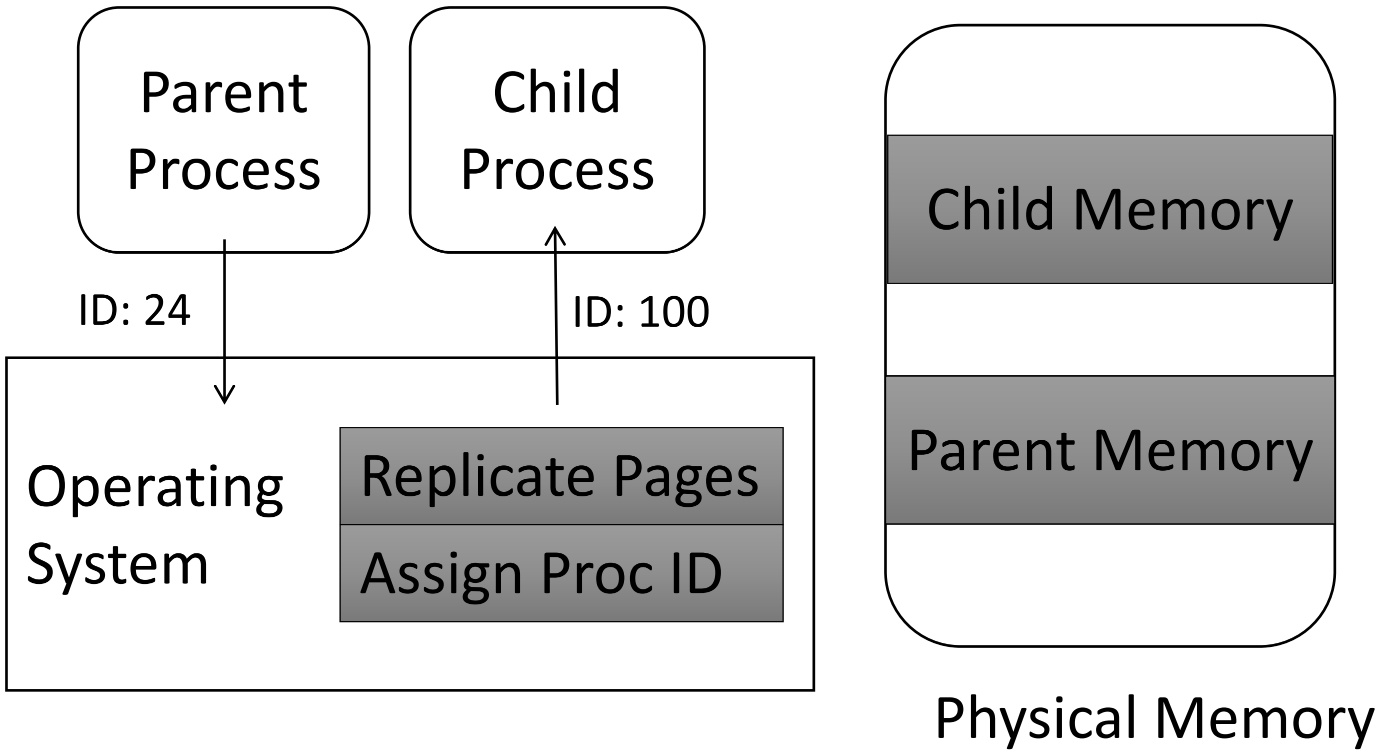
- Process abstraction breaks
  - locks are in public memory
  - shared memory for processes
  - passing data to system calls



Expressiveness Example: Fork

- Fork Semantics:
  - Assigns new process id
  - Makes a memory replica

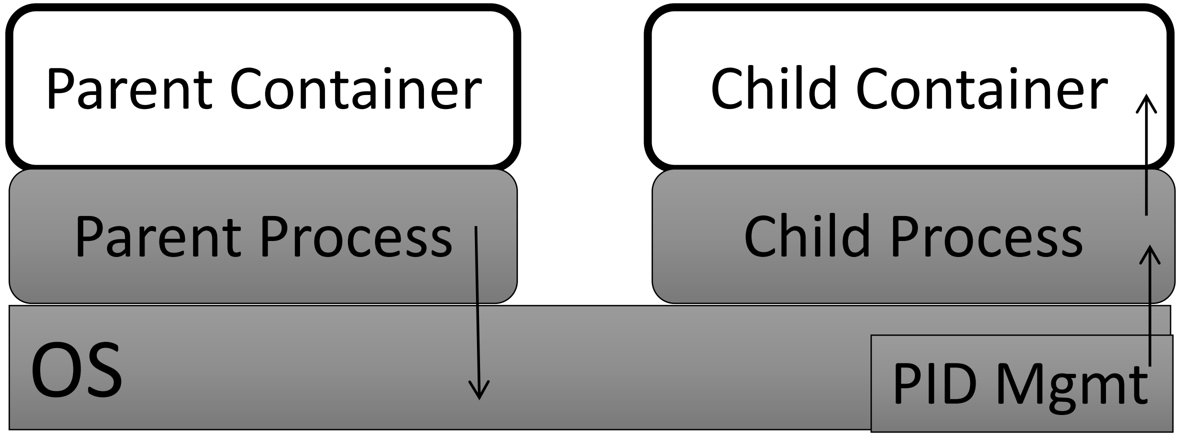
- How to maintain fork semantics if the OS cannot access private memory?



Expressiveness Example: Delegating Fork

- Creating child process and child micro-container

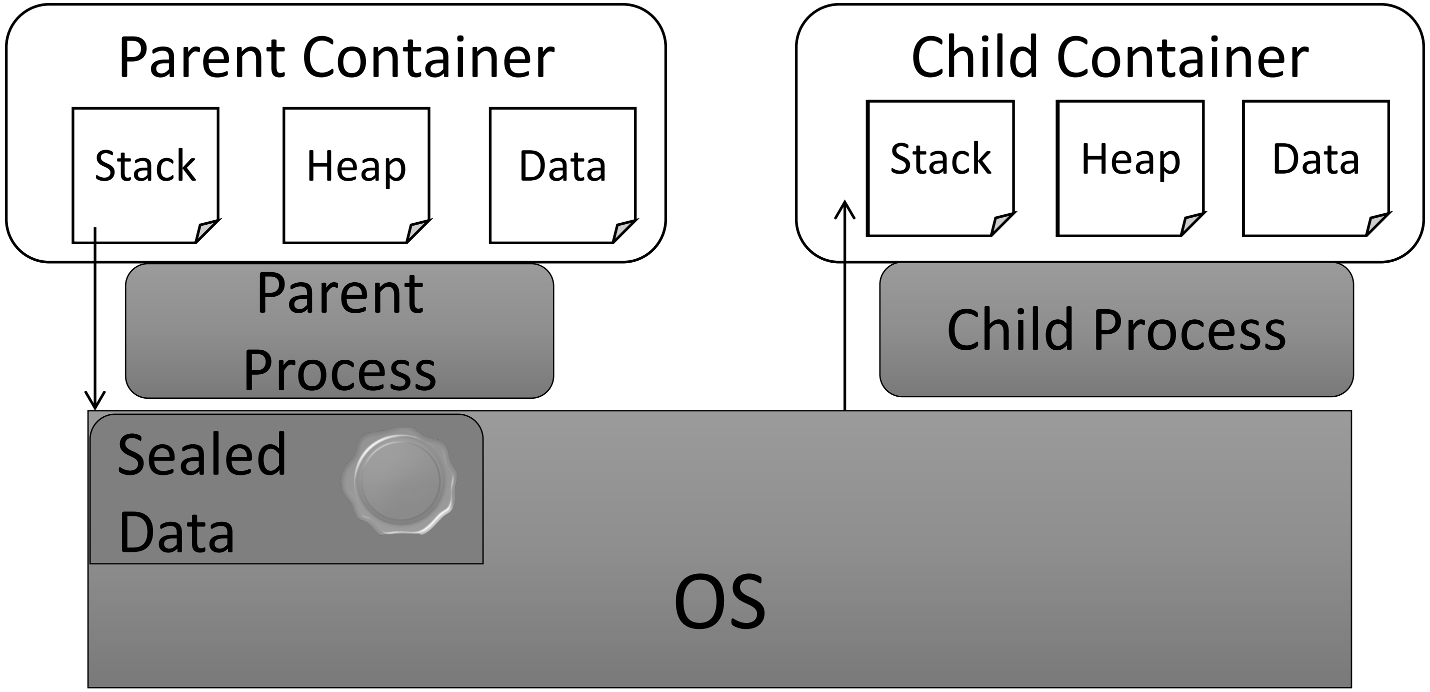
Micro-container Fork



- 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

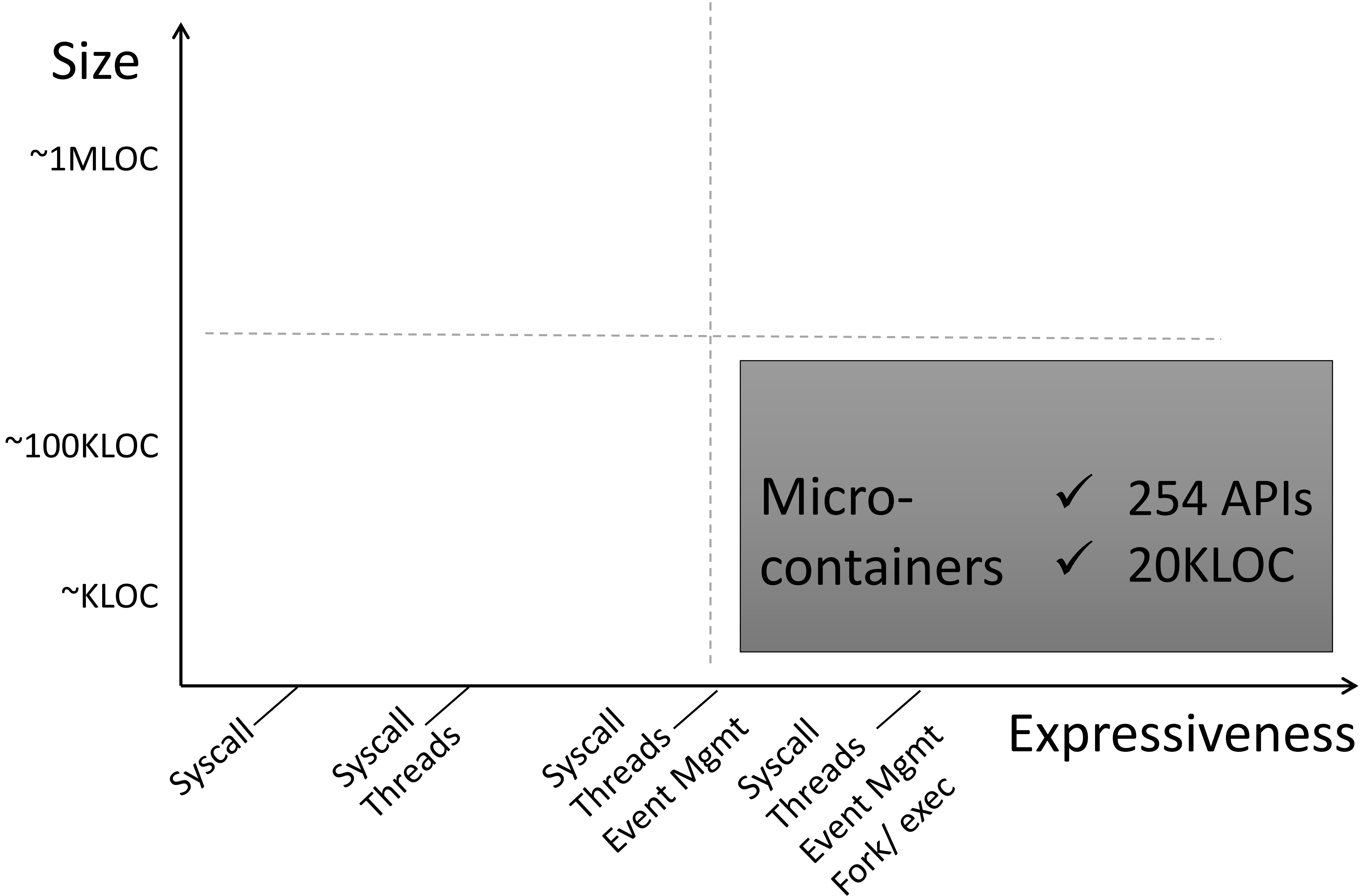
WEBSERVERS

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

My Work

Intel
SGX Protected
File System

Microsoft
OpenEnclave

Baidu
Rust SGX SDK

Google
Asylo

Early Dec 2016 — **Late Dec 2016** — **August 2017** — **Jan 2018** — **May 2018**

3rd component of this stack

New Applications [Arxiv'18], [ICDCS'19]
Secure Computation [CCS'13]

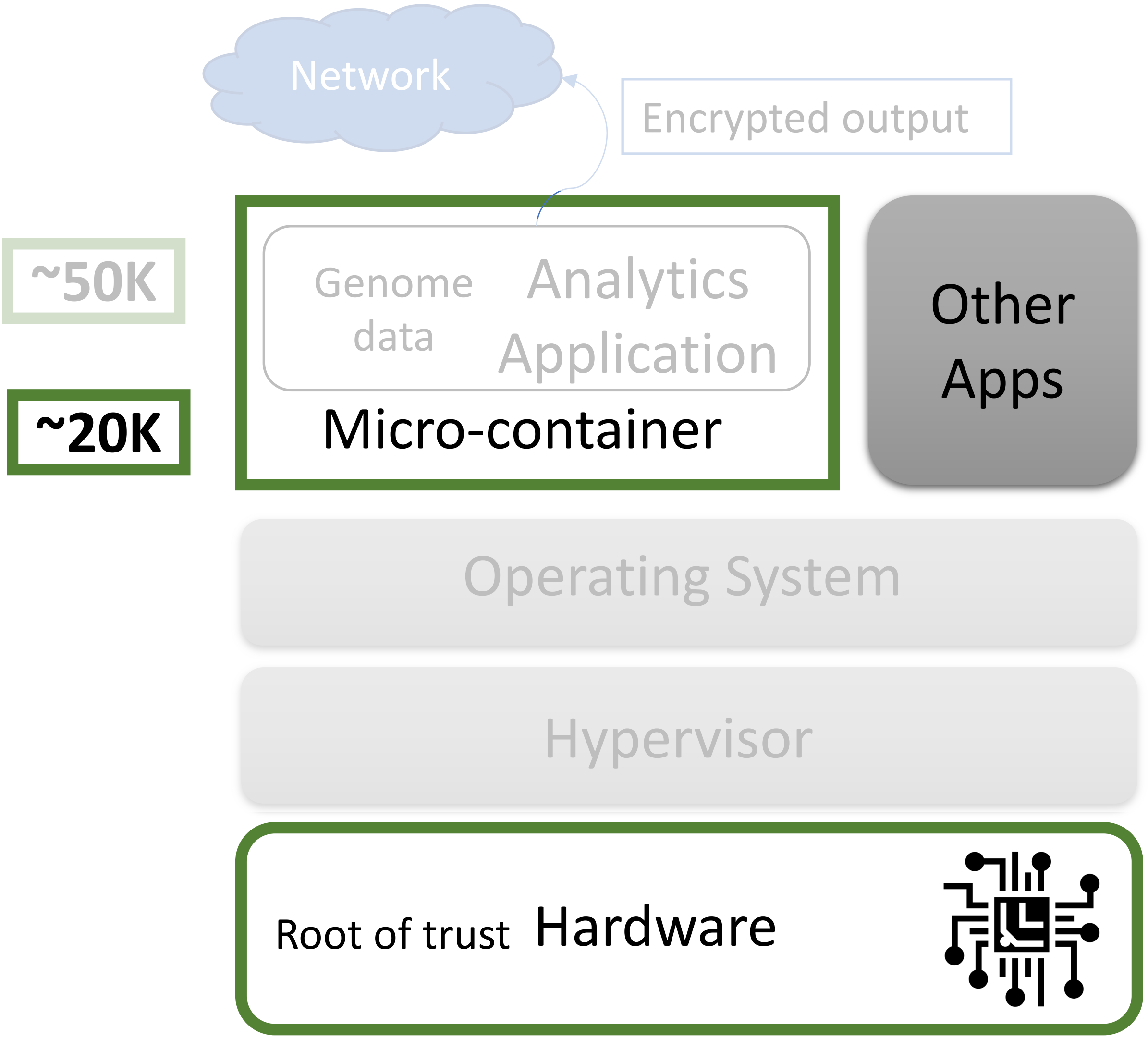
Analysis & hardening
[PLDI'14], [FSE'15], [NDSS'19], [CCS'20]

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

Formal verification [Usenix '20]

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

Trusted Computing Primitives
[TR'15], [Eurosys'20]



Example: What is the damage via OS interface?

```
FILE* fd = fopen(fname, mode);
 if (fd == NULL) {
    errnum = errno;
    if (errnum == EINVAL)
       fd = fopen (fname, “a”);
    if (errnum == ENOENT)
       if (fname == NULL)
       fname = “vote.log”;
       fd = create_log(fname);
    if (errnum == EINTR)
       fd = fopen(fname, mode);
}
 if (fd)
    cnt = fwrite(buf, 1, len, fd);
return cnt;
```

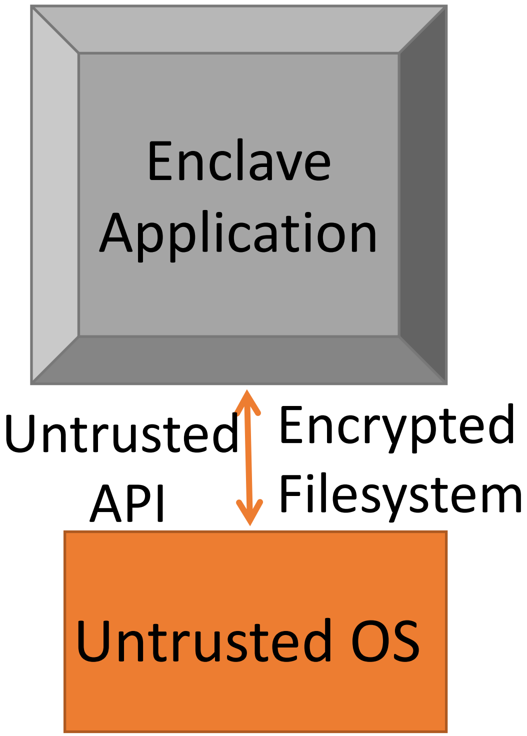
Open the vote file

Failed to open the vote file

Create new file → overwrite previous vote

Register the vote

Tamper the vote sequence



BesFS: A POSIX Filesystem for Enclaves with a Mechanized Safety Proof [Usenix Security'20]

Attacks are possible in delegation frameworks

fopen: Google Asylo

```
int enc_untrusted_open(const char *path_name, int flags) {
  uint32_t mode = 0;
  int result;
  sgx_status_t status = ocall_enc_untrusted_open(&result,
        path_name, flags, mode);
  if (status != SGX_SUCCESS) {
    errno = EINTR;
    return -1;
  }
  return result;
}
```

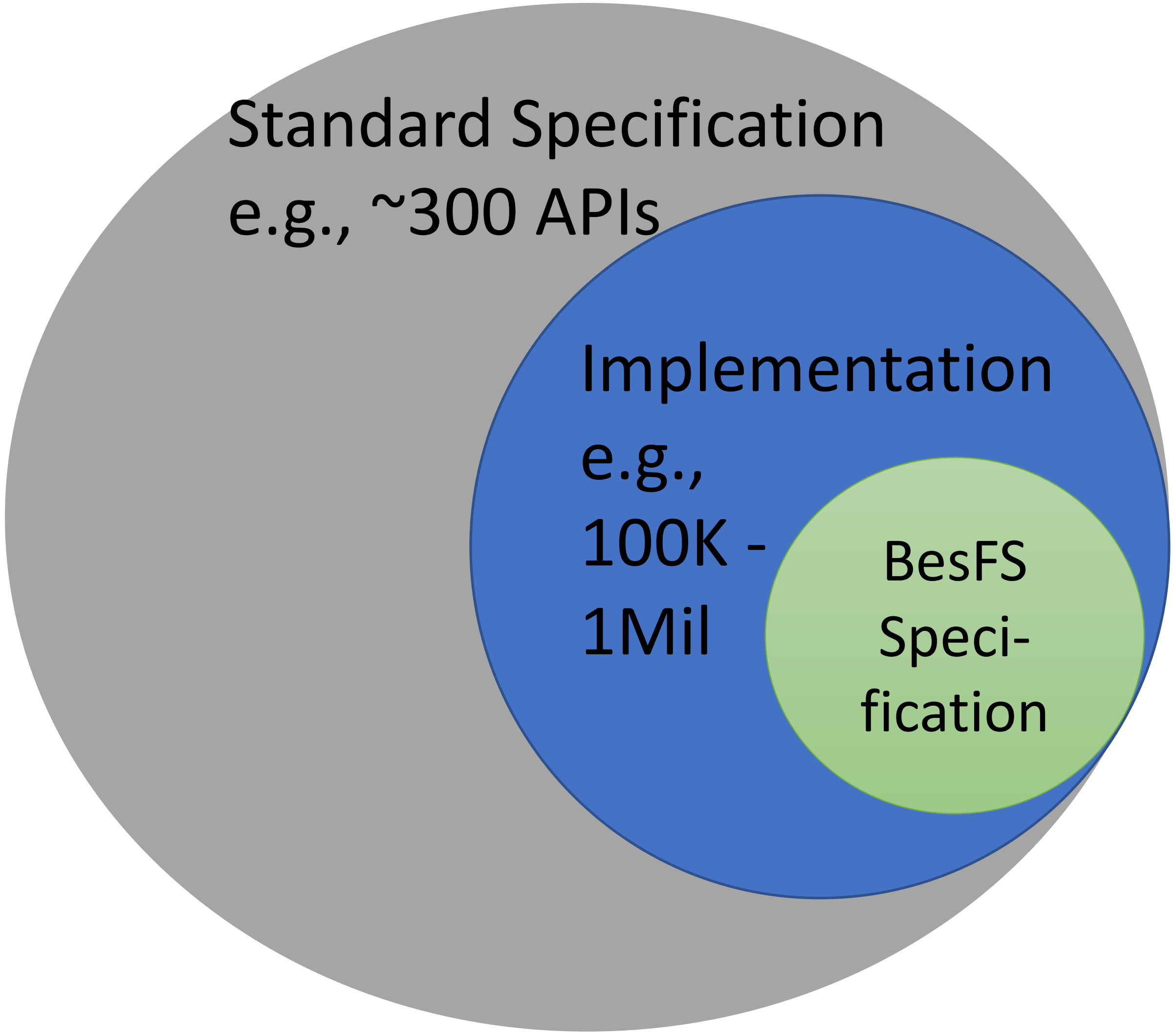
fopen: Graphene-SGX

```
static int sgx_ocall_open(void * pms) {
    ms_ocall_open_t * ms = (ms_ocall_open_t *) pms;
    int ret;
    ODEBUG(OCALL_OPEN, ms);
    ret = INLINE_SYSCALL(open, ...);
    return IS_ERR(ret)?unix_to_pal_error(ERRNO(ret)):ret;
}
```

fopen: Intel SDK

```
static SGX_FILE* sgx_fopen_internal
(const char* filename, const char* mode) {
  protected_fs_file* file = NULL;
  if (filename == NULL || mode == NULL) {
    errno = EINVAL;
    return NULL;
  }
    ...
}
```

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)

Towards Next Generation Computation Stack

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

Secure Computation [CCS'13]

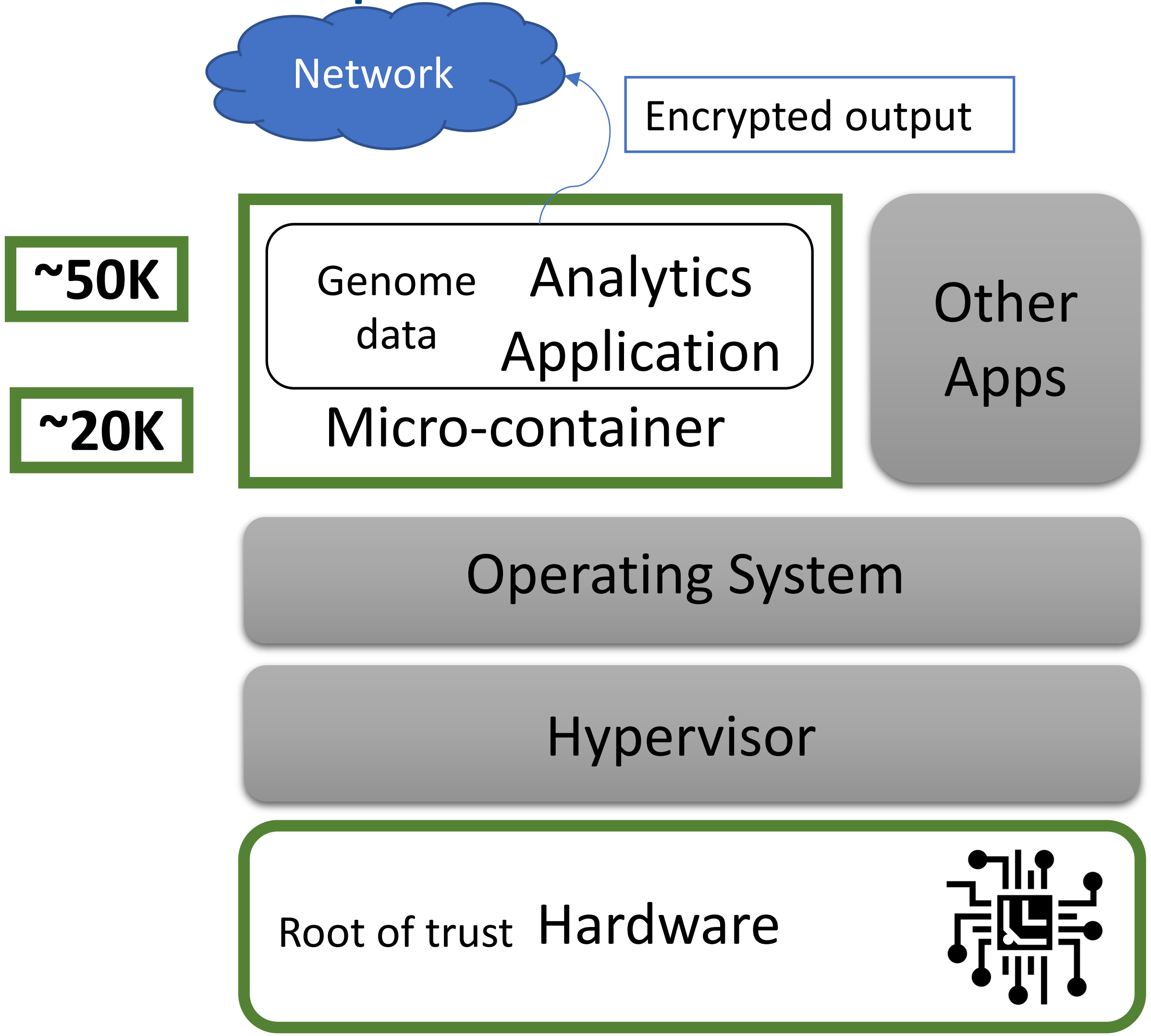
Analysis & hardening
[PLDI'14], [FSE'15], [NDSS'19], [CCS'20]

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

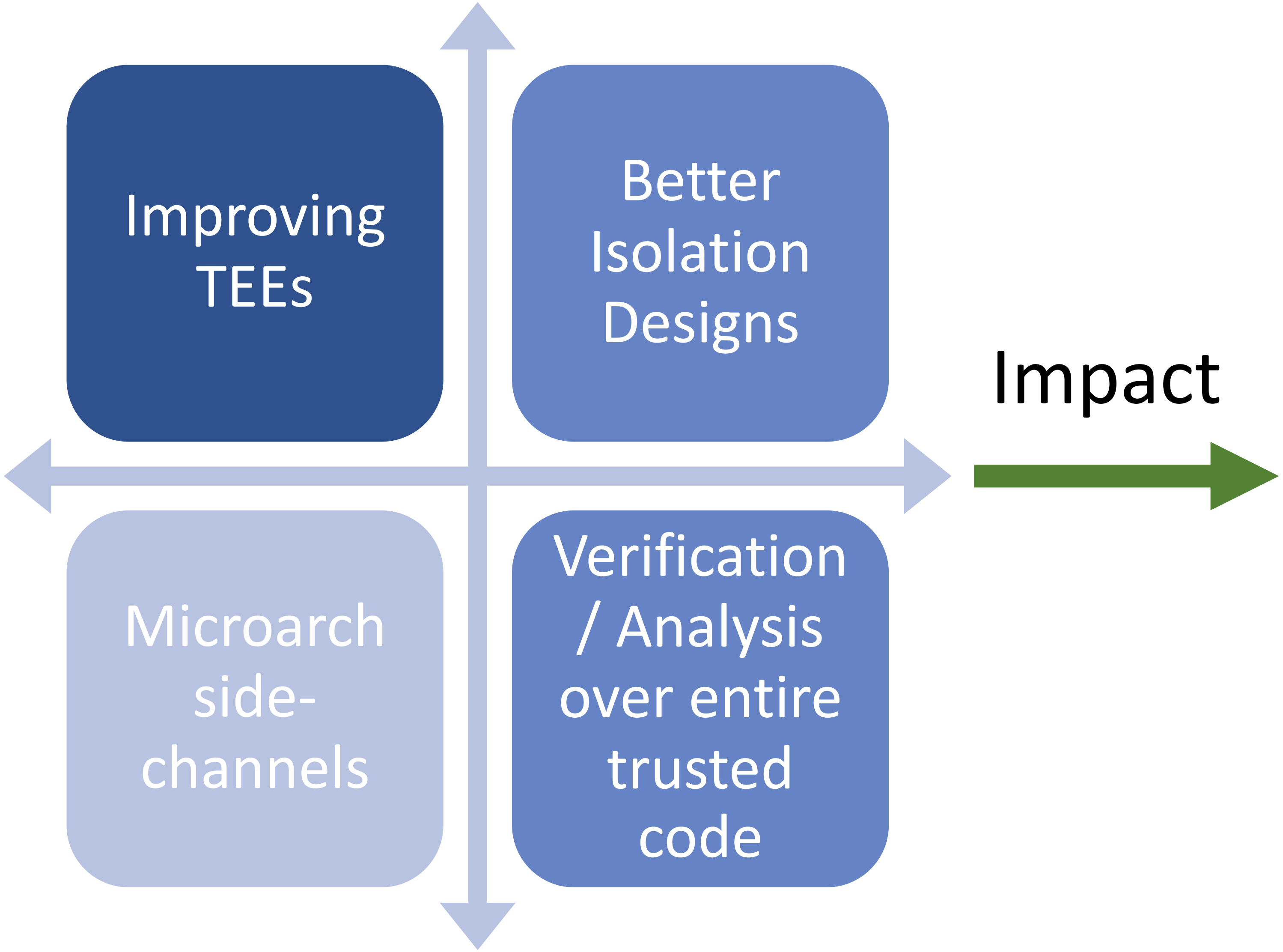
Formal verification [Usenix'20]

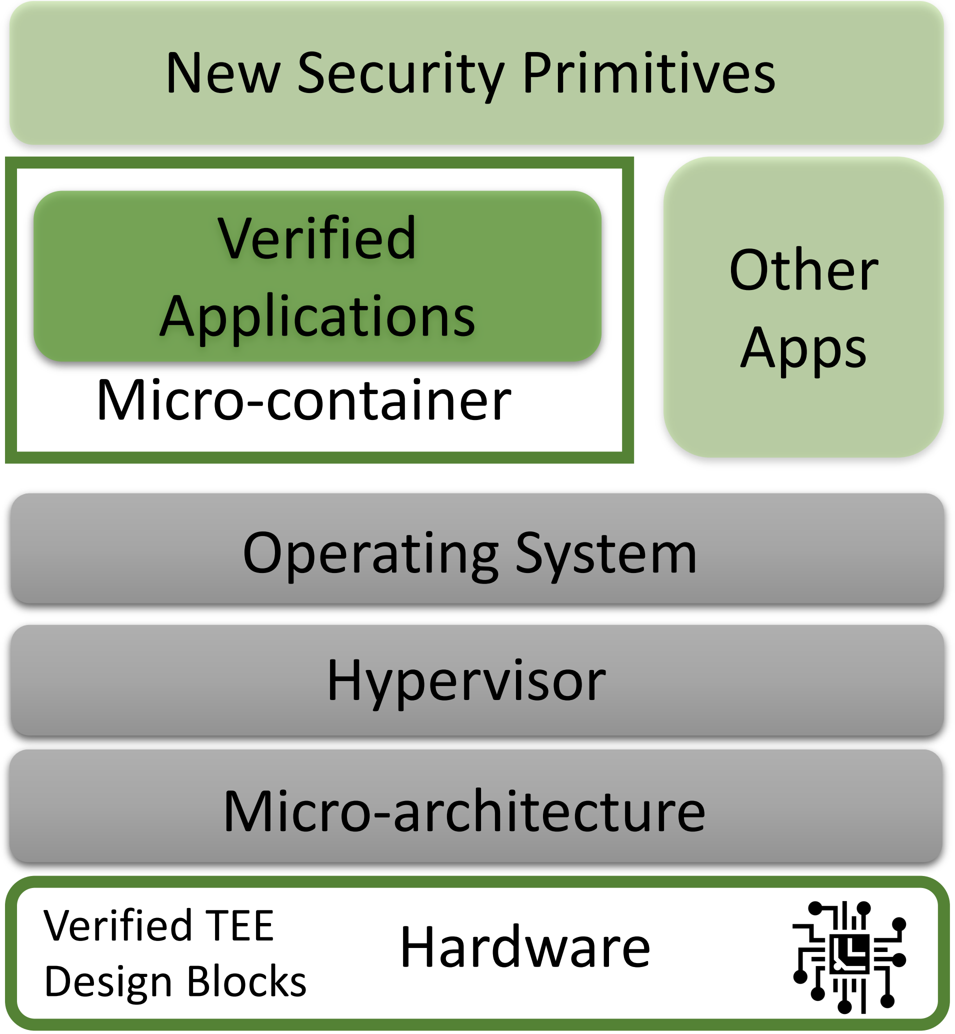
Attacks & Defenses [AsiaCCS'16, CCS'21]

Trusted Computing Primitives
[TR'15], [Eurosys'20]



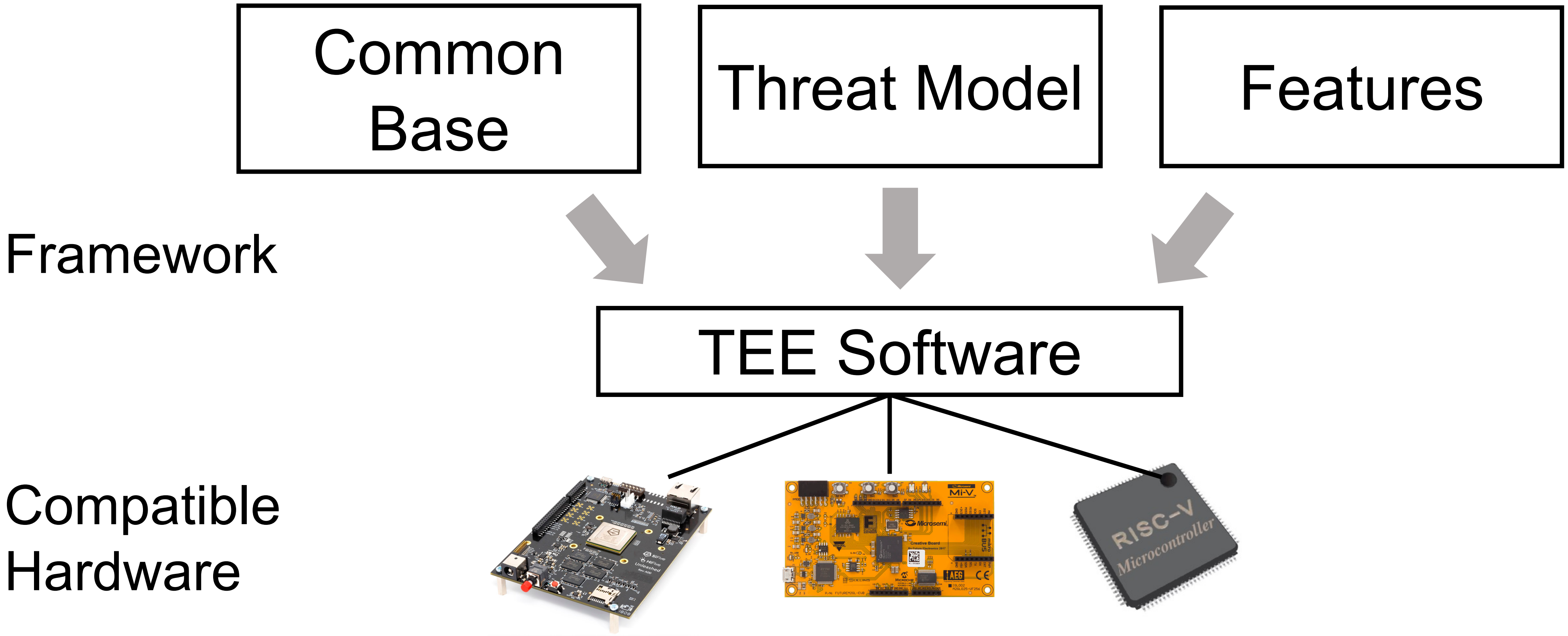
Research Directions





Customizable TEEs

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



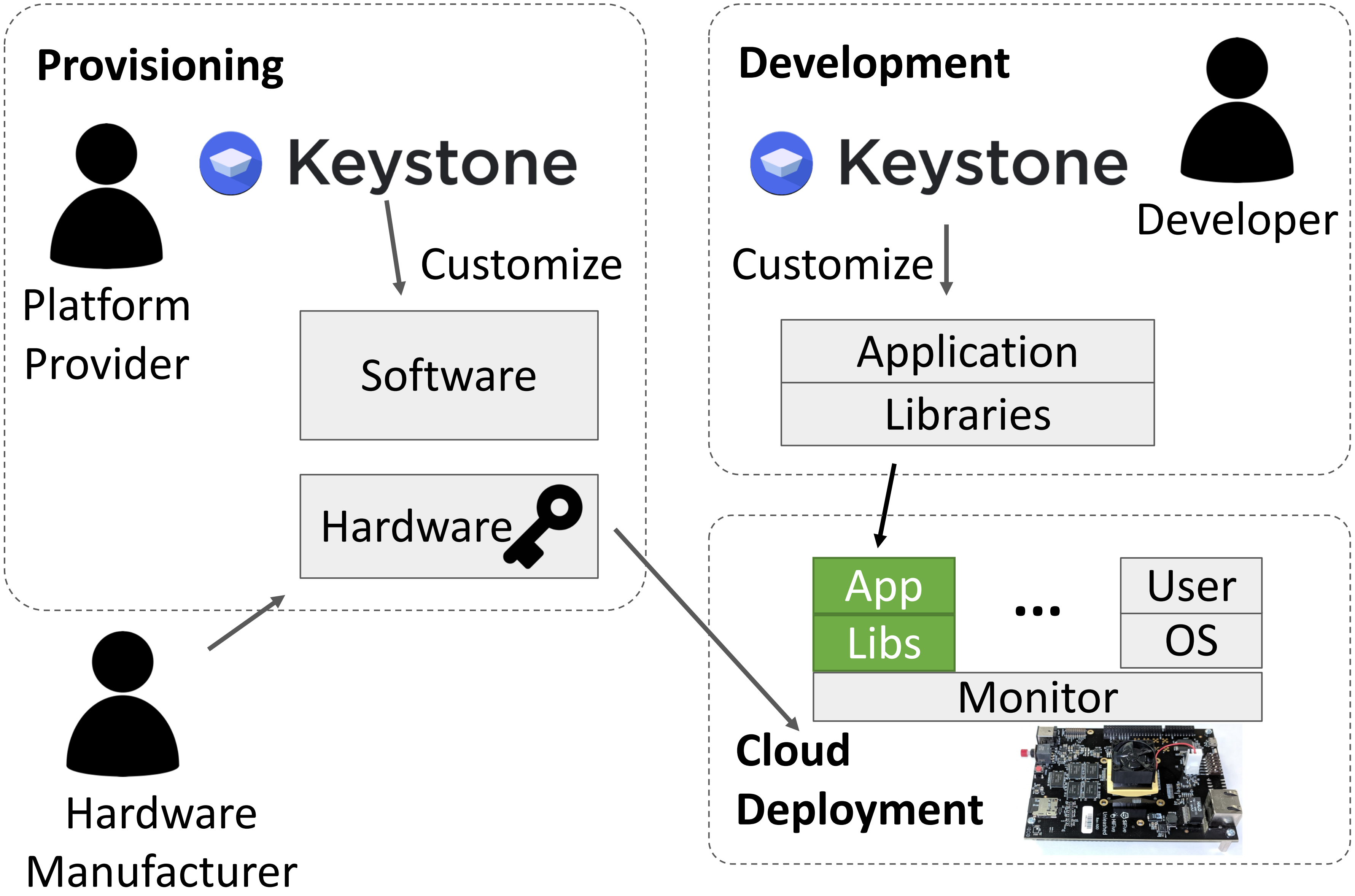
Keystone

A software framework for TEEs on RISC-V

No micro-architectural changes

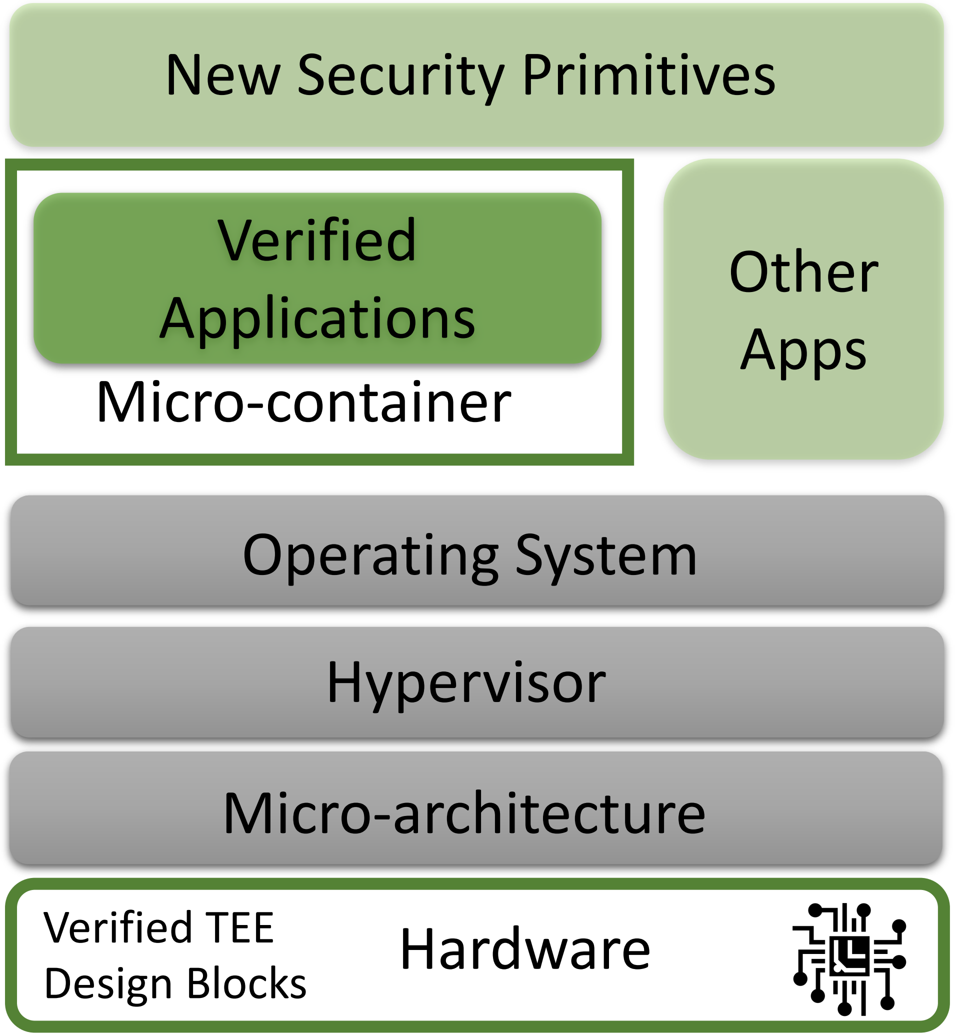
Minimal added hardware

Keystone Workflow for Customizable TEEs



Research Goals for Future TEE Platforms

- Modular TCB, easy to reduce and verify
- Binary compatibility with legacy applications
- Enable support for various backend hardware platforms
- Evolve to better hardware designs for TEE independently of the software



ETH zürich

SECTRS

Secure & Trustworthy Systems Group

Shweta Shinde
Assistant Professor
shweta.shivajishinde@inf.ethz.ch

ETH Zurich
Department of Computer Science

@shw3ta_shinde

https://shwetashinde.org
https://www.sectrs.ethz.ch