Asterinas

A safe and efficient Rust-based OS kernel for TEE and beyond

Hongliang Tian
Ant Group

Edmund Song
Intel Corporation

OC3 | March 13, 2024

- **Part 1: How Iago attacks threaten Linux's memory safety**

- **Part 2: Why Asterinas is memory safe despite of Iago attacks**

- **Part 3: How Asterinas is ported to Intel TDX**

- Part 1: How Iago attacks threaten Linux’s memory safety
- Part 2: Why Asterinas is memory safe despite of Iago attacks
- Part 3: How Asterinas is ported to Intel TDX

Game: can you spot the memory safety bug (1)

- The following code snippet* from Linux kernel suffers a memory safety issue caused by Iago attacks

```
// file: linux/drivers/virtio/virtio_ring.c

static inline int virtqueue_add_split(struct virtqueue *_vq, /* more args */) {
    // ...

    for (n = 0; n < out_sgs; n++) {
        for (sg = sgs[n]; sg; sg = sg_next(sg)) {
            dma_addr_t addr = vring_map_one_sg(vq, sg, DMA_TO_DEVICE);

            desc[i].flags = cpu_to_virtio16(_vq->vdev, VRING_DESC_F_NEXT);
            desc[i].addr = cpu_to_virtio64(_vq->vdev, addr);
            desc[i].len = cpu_to_virtio32(_vq->vdev, sg->length);
            prev = i;
            i = virtio16_to_cpu(_vq->vdev, desc[i].next);
        }
    }

    // ...
}
```

Untrusted input from device

* Hetzelt, Felicitas, et al. "Via: Analyzing device interfaces of protected virtual machines." *Annual Computer Security Applications Conference*. 2021.

Game: can you spot the memory safety bug (1)

- The following code snippet* from Linux kernel suffers a memory safety issue caused by Iago attacks

```
// file: linux/drivers/virtio/virtio_ring.c

static inline int virtqueue_add_split(struct virtqueue *_vq, /* more args */) {
    // ...

    for (n = 0; n < out_sgs; n++) {
        for (sg = sgs[n]; sg; sg = sg_next(sg)) {
            dma_addr_t addr = vring_map_one_sg(vq, sg, DMA_TO_DEVICE);

            desc[i].flags = cpu_to_virtio16(_vq->vdev, VRING_DESC_F_NEXT);
            desc[i].addr = cpu_to_virtio64(_vq->vdev, addr);
            desc[i].len = cpu_to_virtio32(_vq->vdev, sg->length);
            prev = i;
            i = virtio16_to_cpu(_vq->vdev, desc[i].next);
        }
    }

    // ...
}
```

Untrusted input from device

Out-of-bound indexing

* Hetzelt, Felicitas, et al. "Via: Analyzing device interfaces of protected virtual machines." *Annual Computer Security Applications Conference*. 2021.

Game: can you spot the memory safety bug (2)

- The following code snippet* from Linux kernel suffers a memory safety issue caused by Iago attacks

```
// file: drivers/char/virtio_console.c

static int init_vqs(struct ports_device *portdev) {
    // ...

    nr_ports = portdev->max_nr_ports;
    nr_queues = use_multiport(portdev) ? (nr_ports + 1) * 2 : 2;
    vqs = kmalloc_array(nr_queues, sizeof(struct virtqueue *), GFP_KERNEL);
    if (!vqs) {
        err = -ENOMEM;
        goto free;
    }

    // ...
}
```

Untrusted input from device

* Hetzelt, Felicitas, et al. "Via: Analyzing device interfaces of protected virtual machines." *Annual Computer Security Applications Conference*. 2021.

Game: can you spot the memory safety bug (2)

- The following code snippet* from Linux kernel suffers a memory safety issue caused by Iago attacks

```c
// file: drivers/char/virtio_console.c

static int init_vqs(struct ports_device *portdev) {
    // ...

    nr_ports = portdev->max_nr_ports;
    nr_queues = use_multiport(portdev) ? (nr_ports + 1) * 2 : 2;
    vqs = kmalloc_array(nr_queues, sizeof(struct virtqueue *), GFP_KERNEL);
    if (!vqs) {
        err = -ENOMEM;
        goto free;
    }

    // ...
}
```

Untrusted input from device

Integer overflow

Allocation of zero-sized objects

* Hetzelt, Felicitas, et al. "Via: Analyzing device interfaces of protected virtual machines." *Annual Computer Security Applications Conference*. 2021.

Game: can you spot the memory safety bug (3)

- The following code snippet* from Linux kernel suffers a memory safety issue caused by Iago attacks

```
// file: linux/drivers/net/virtio_net.c

static int virtnet_probe(struct virtio_device *vdev) {
    // ...

    if (mtu < dev->min_mtu) {
        /* Should never trigger: MTU was previously validated
         * in virtnet_validate.
         */
        goto free;
    }

    // ...

    return 0;

    // ...
free:
    free_netdev(dev);
    return err;
}
```

Untrusted input from device

* Hetzelt, Felicitas, et al. "Via: Analyzing device interfaces of protected virtual machines." *Annual Computer Security Applications Conference*. 2021.

Game: can you spot the memory safety bug (3)

- The following code snippet* from Linux kernel suffers a memory safety issue caused by Iago attacks

```
// file: linux/drivers/net/virtio_net.c

static int virtnet_probe(struct virtio_device *vdev) {
    // ...

    if (mtu < dev->min_mtu) {
        /* Should never trigger: MTU was previously validated
         * in virtnet_validate.
         */
        goto free;
    }

    // ...

    return 0;

    // ...
free:
    free_netdev(dev);
    return err;
}
```

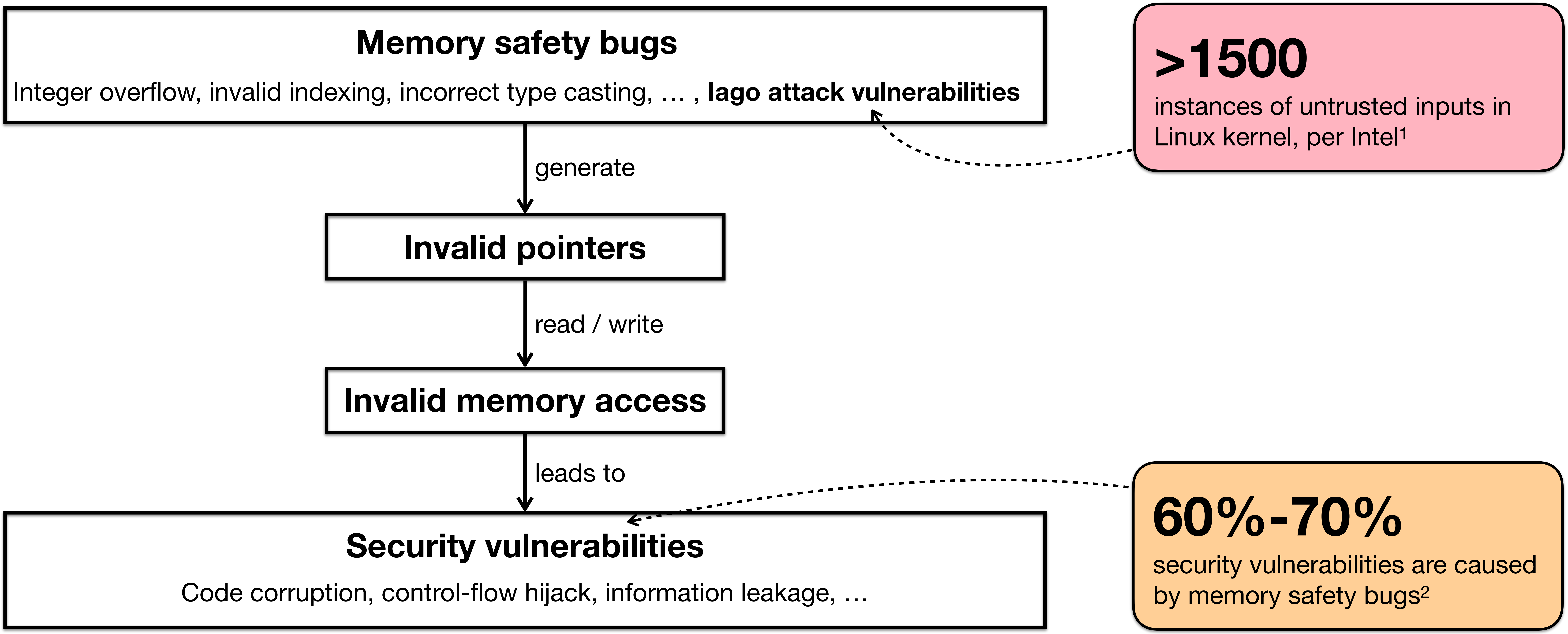
Untrusted input from device

Unset error number

Use-after-free

* Hetzelt, Felicitas, et al. "Via: Analyzing device interfaces of protected virtual machines." *Annual Computer Security Applications Conference*. 2021.

Iago attacks make Linux even more unsafe…



1. Intel® Trust Domain Extension Guest Linux Kernel Hardening Strategy: https://intel.github.io/ccc-linux-guest-hardening-docs/tdx-guest-hardening.html
2. What science can tell us about C and C++'s security: https://alexgaynor.net/2020/may/27/science-on-memory-unsafety-and-security/

- Part 1: How Iago attacks threaten Linux's memory safety
- Part 2: Why Asterinas is memory safe despite of Iago attacks
- Part 3: How Asterinas is ported to Intel TDX

Asterinas

A secure, fast, and general-purpose OS kernel written in Rust and compatible with Linux

http://github.com/asterinas/asterinas

Why Rust kernel != safe kernel

The `unsafe` keyword in Rust has superpowers

- Examples of the superpowers:
  - Dereferencing a raw pointer
  - Inserting assembly code
  - Calling unsafe functions
  - Implementing unsafe traits

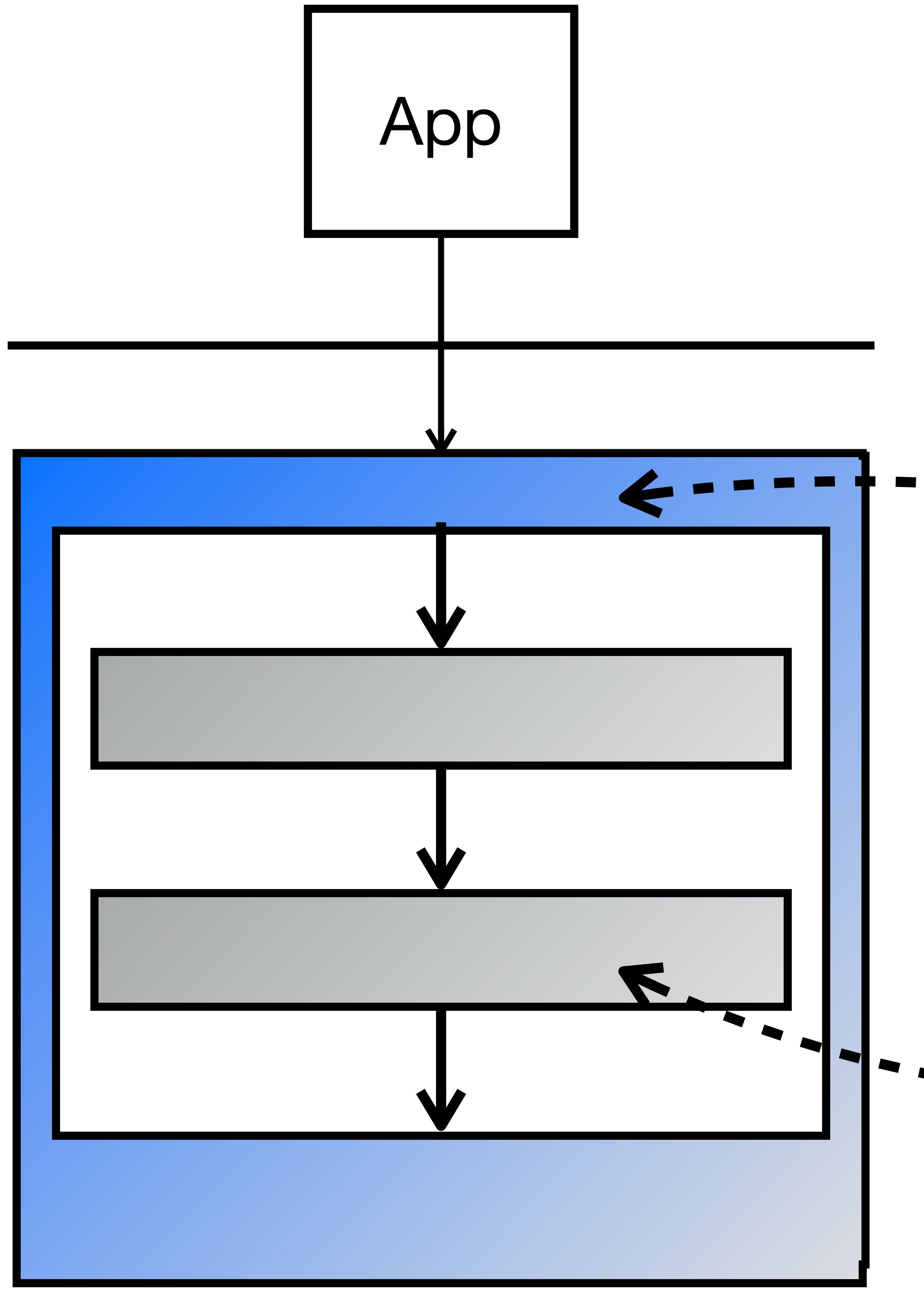
Rust kernels must use the `unsafe` superpowers

- Low-level operations require `unsafe`
  - Manipulating CPU registers
  - Accessing physical memory
  - Doing user-kernel switches
  - Handling interrupts

With great power,
comes with
great responsibility

Introducing the framekernel OS architecture

Framekernel = single address space + safe language + safe/unsafe partition

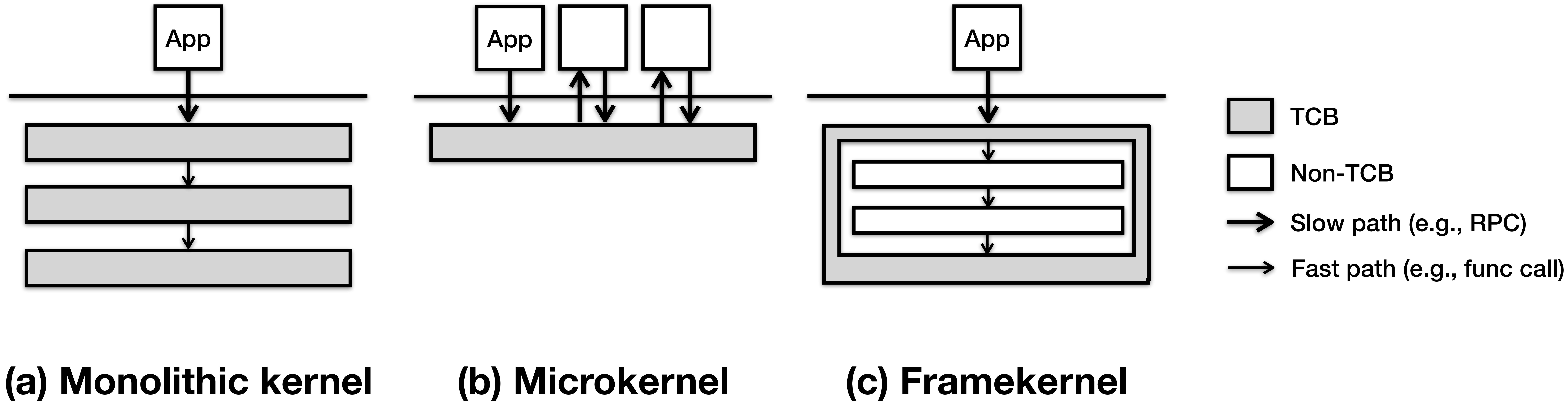


Framekernel

	Allow unsafe?	Responsibilities	Code Sizes	Memory Safety
OS Framework	Yes	Encapsulate low-level unsafe code into safe abstractions	Small	Examined by programmers
OS Services	No	Implement OS functionalities, including device drivers	Large	Guaranteed by Rust compiler

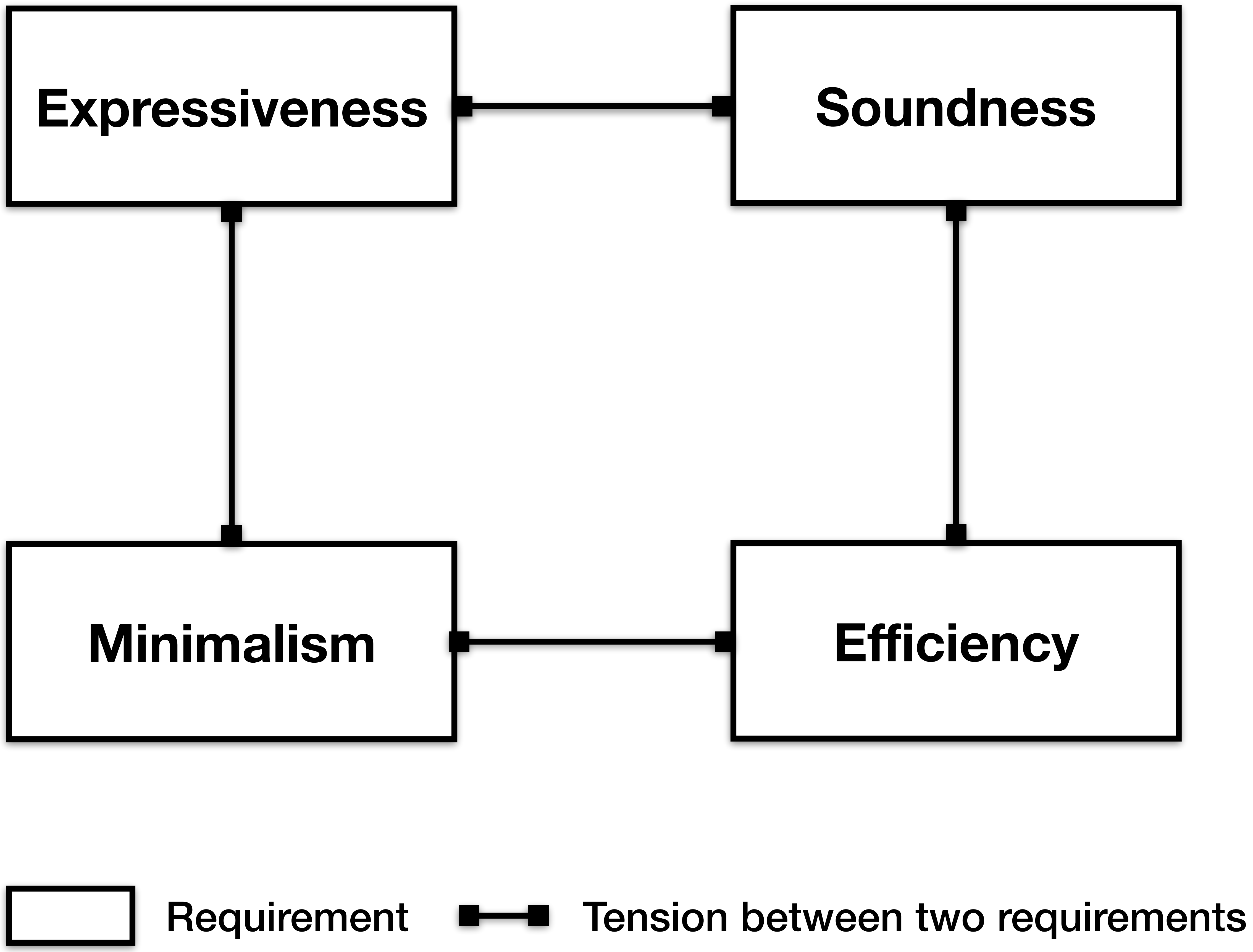
Framekernel promises both security & performance

Figure. A comparison between different OS architectures

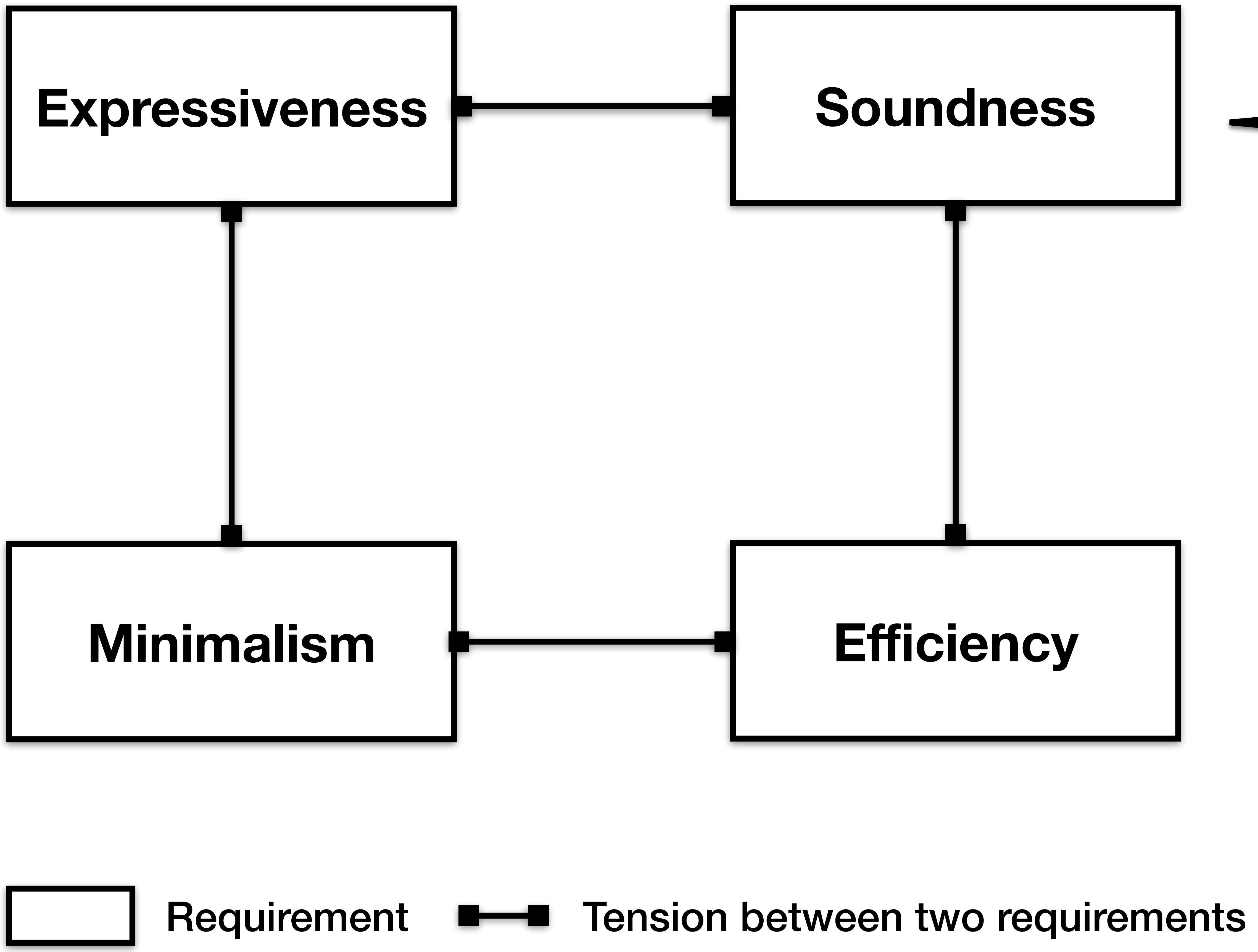


👉 **The speed of a monolithic kernel, the security of a microkernel**

The four requirements for the OS framework



The four requirements for the OS framework



A Rust crate is sound
if *any* safe Rust system based upon it
does not exhibit undefined behaviors.

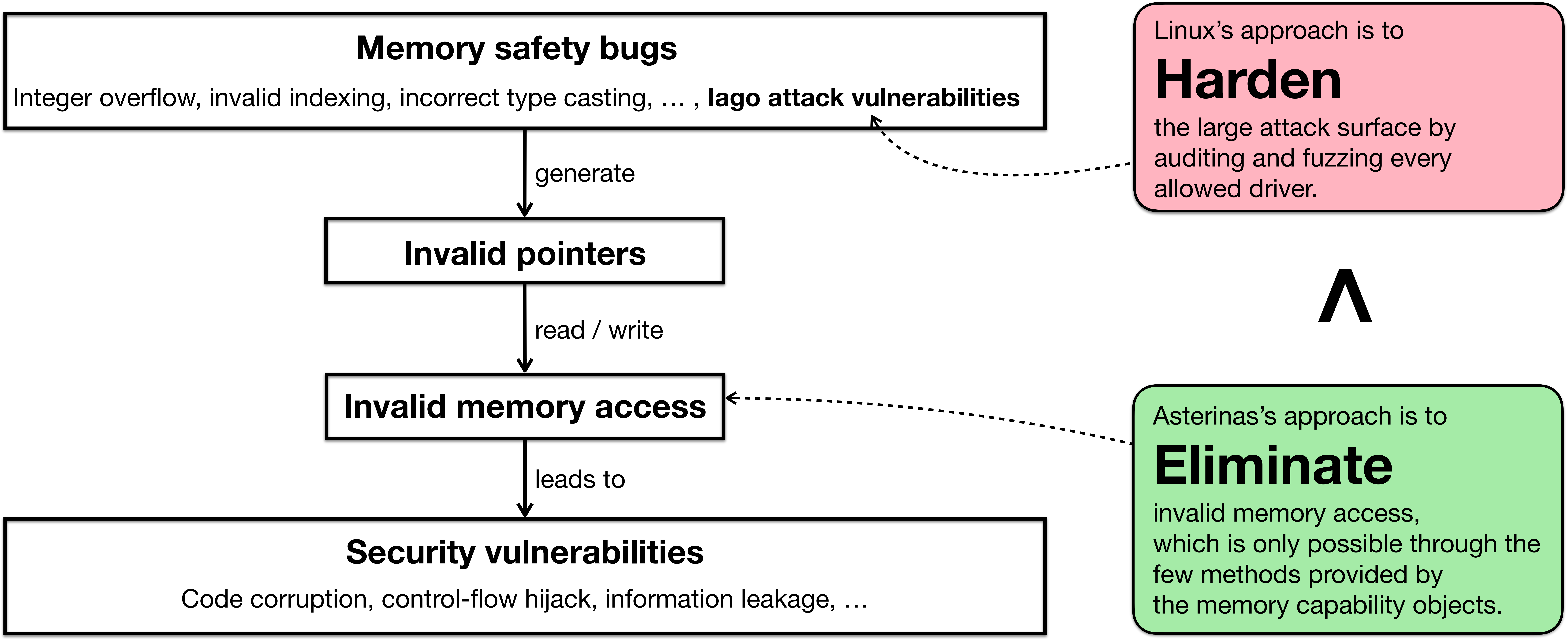
A safe Rust system may
contain arbitrary safe Rust code,
may be executed in arbitrary timings,
and may take arbitrary inputs.

This implies the resistance against
malicious inputs from **Iago attacks**

Asterinas Framework: Typed vs untyped memory

- Physical memory pages are classified into two categories.
  - Typed memory are the one that may affect Rust's type safety, e.g., the code, stack, heap, page tables of the kernel and BIOS.
  - Untyped memory are the one that does not affect Rust's type safety, including any usable physical pages that are not marked as typed yet.
- The Framework API only allows access to the untyped memory and it must be done through carefully-designed Rust capability objects:
  - `VmFrame`: a physical memory page
  - `VmSpace`: a user memory space
  - `DmaCoherent`: a coherent DMA mapping
  - `DmaStream`: a streaming DMA mapping
- Use the safe methods provided by these memory capability objects, instead of dereferencing raw pointers!

Defense against Iago attacks: Linux vs Asterinas



👉 **Asterinas is more memory safe than Linux, or any other Rust kernels**

Project status and plan

Asterinas has been made open source: https://github.com/asterinas/asterinas

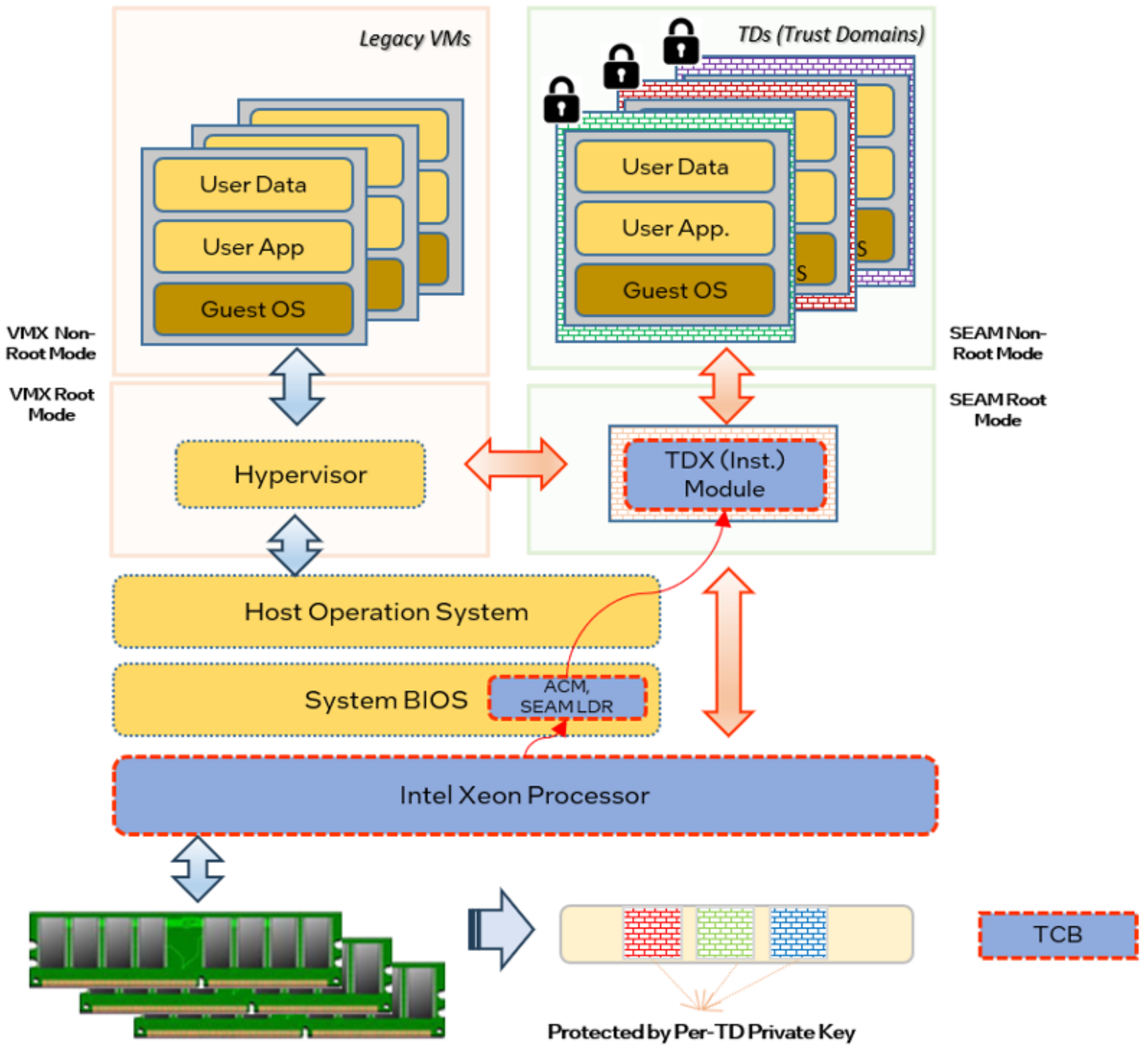
- Current status

50K	120	80%	4
Lines of Rust	Linux syscalls	Safe Rust	Sponsors

- Goal for 2024
  - Get the project ready for production deployment in x86-64 VMs
  - Find early adopters in TEE usage

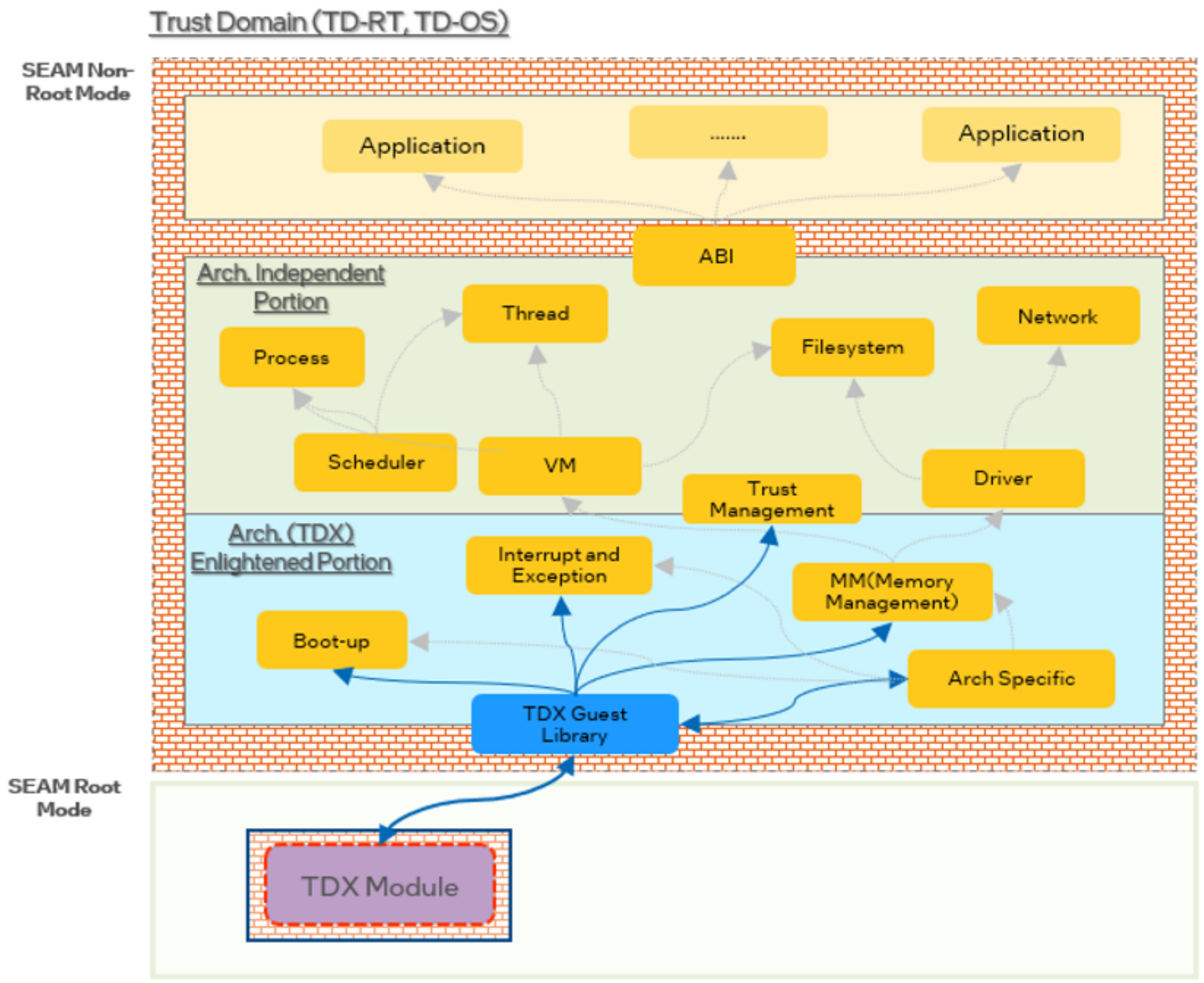
- Part 1: How Iago attacks threaten Linux’s memory safety
- Part 2: Why Asterinas is memory safe despite of Iago attacks
- Part 3: How Asterinas is ported to Intel TDX

Intel Trust Domain Extensions (TDX)



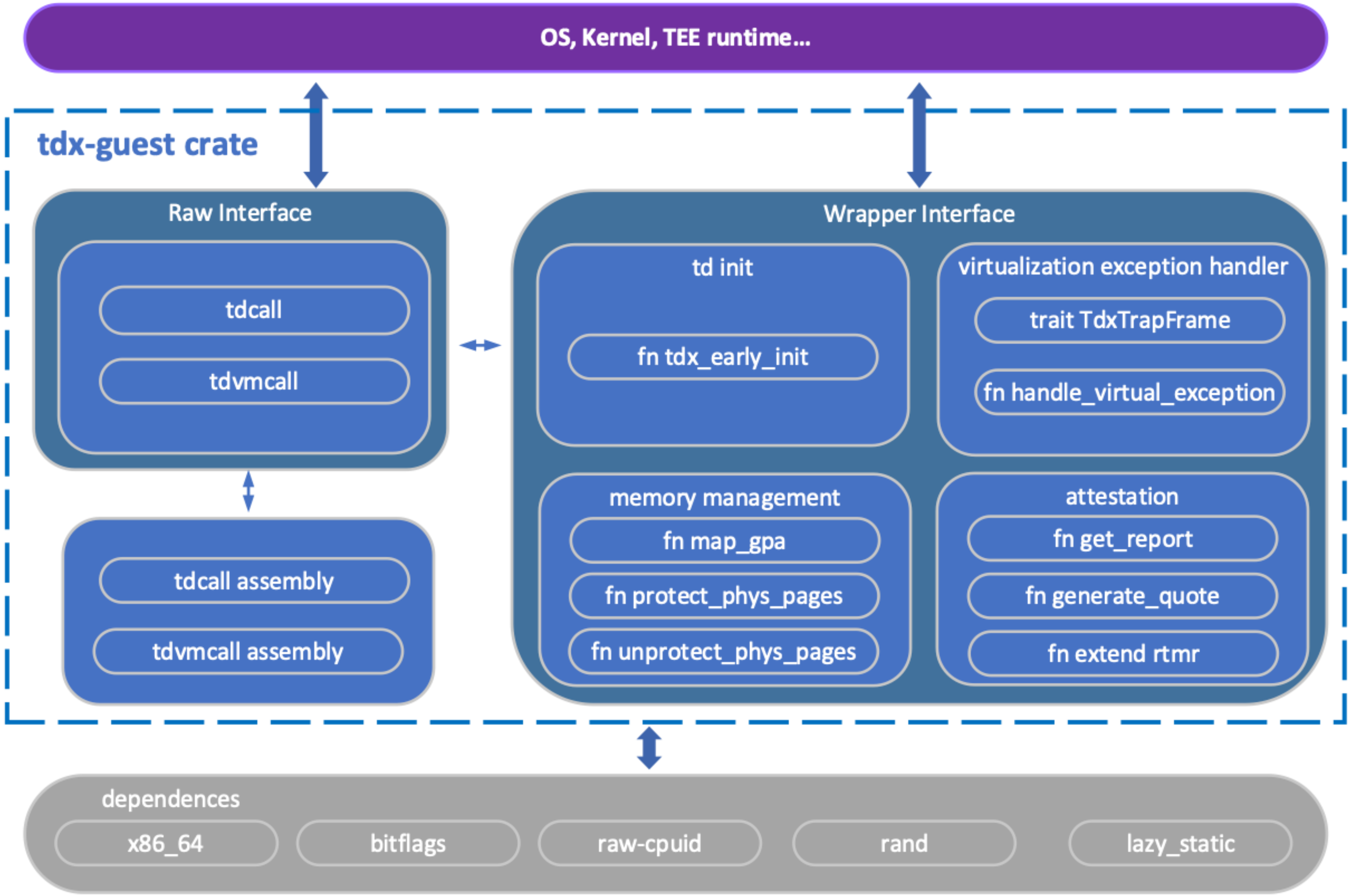
- uArch extensions for confidential computing based on Intel virtualization (VMX)
- "Lift-and-shift" model to migrate application from legacy to confidential computing
- Multi-key memory encryption engine to encrypt user data in-flight, and TDX instruction module to isolate hypervisor from trust boundary
- TCB (Trust Computing Base) limited to silicon level, minimize the cost of trust chai

TDX enablement in the guest environment



- TDX introduces u-Arch enforcement to harden data protection for virtualization instance
- TDX agnostic portion (Arch. Independent portion) vs. TDX enlightened portion .
- Most of TDX modifications fall in boot-up, trap, memory management, and device MMIO etc.

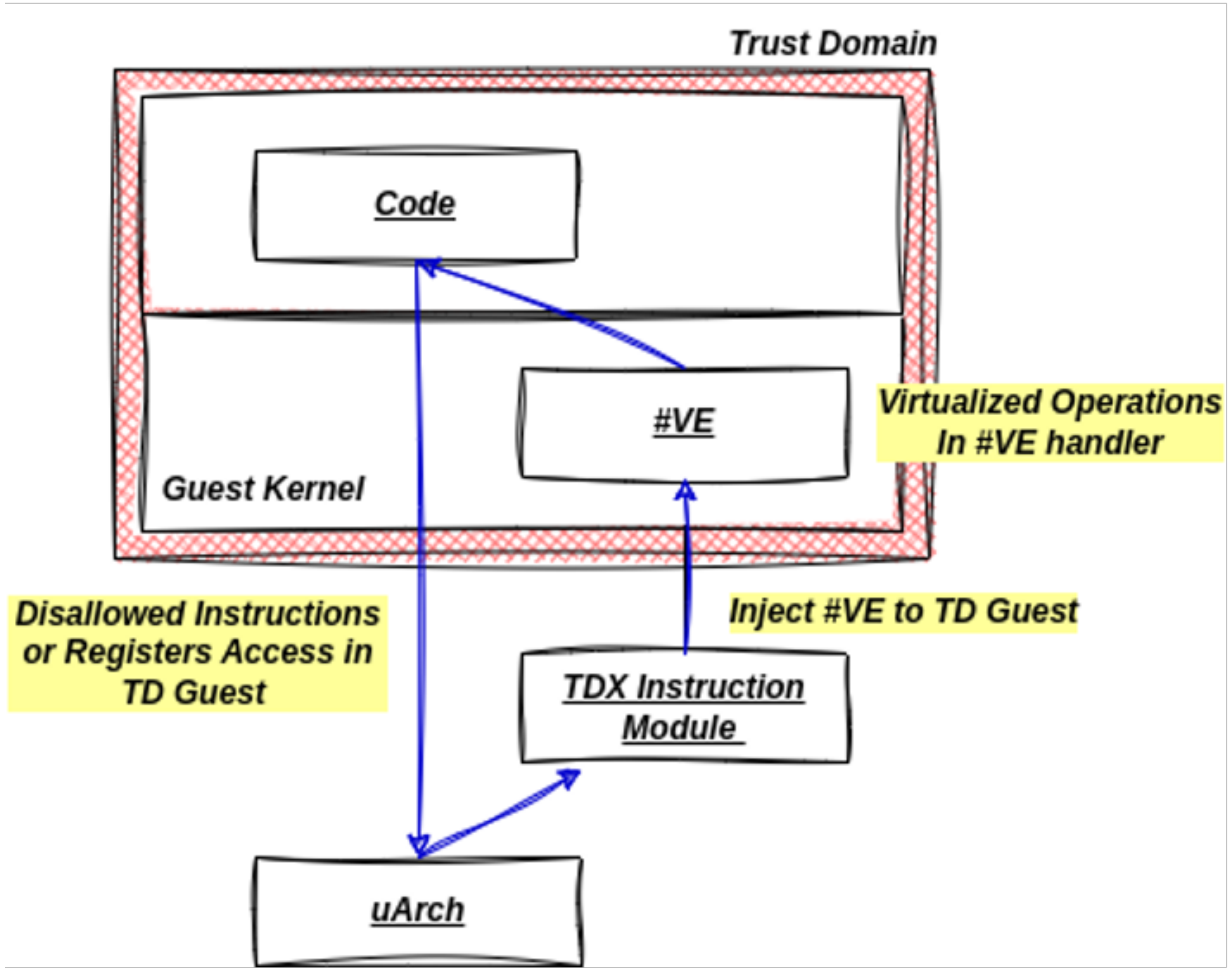
The tdx-guest crate



- An open source project to encapsulate TDX instruction interface for guest environment
- TDX Guest ABI support
  - TDCALL
  - TDVMCALL
- Wrapping interface for TDX guest flow
  - TD Initialization
  - Virtualization Exception (#VE)
  - Memory mapping
  - Measurement and Attestation

#VE: TD-specific virtualization exception

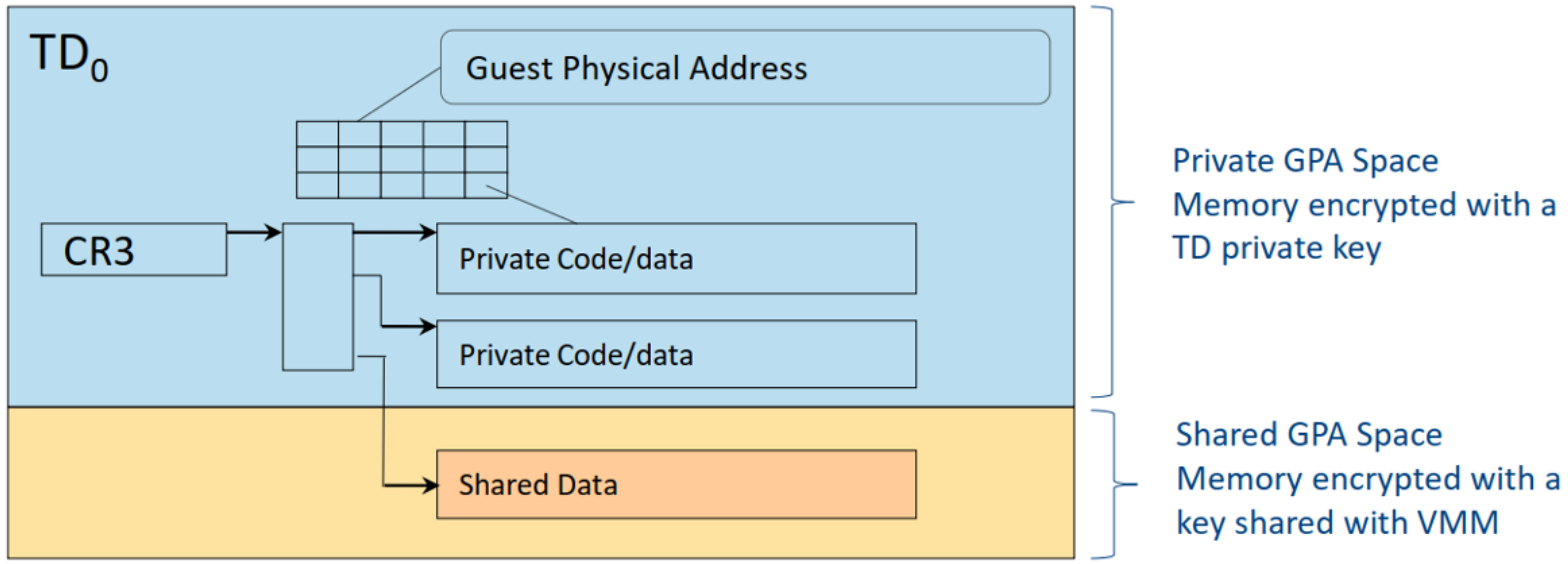
- Why need #VE?
  - Confidential computing enforcement to uArch for security
  - Some cases valid in legacy instance for direct access, but trigger uArch behavior for injecting exception into TD Guest
    - Some instructions access
    - Some registers access, MMIO access



- How to implement?
  - TDX Enlightened Guest setup #VE handler
  - #VE handler analyze exception context and virtualize requested operations for non-Enlightened portions

Memory management

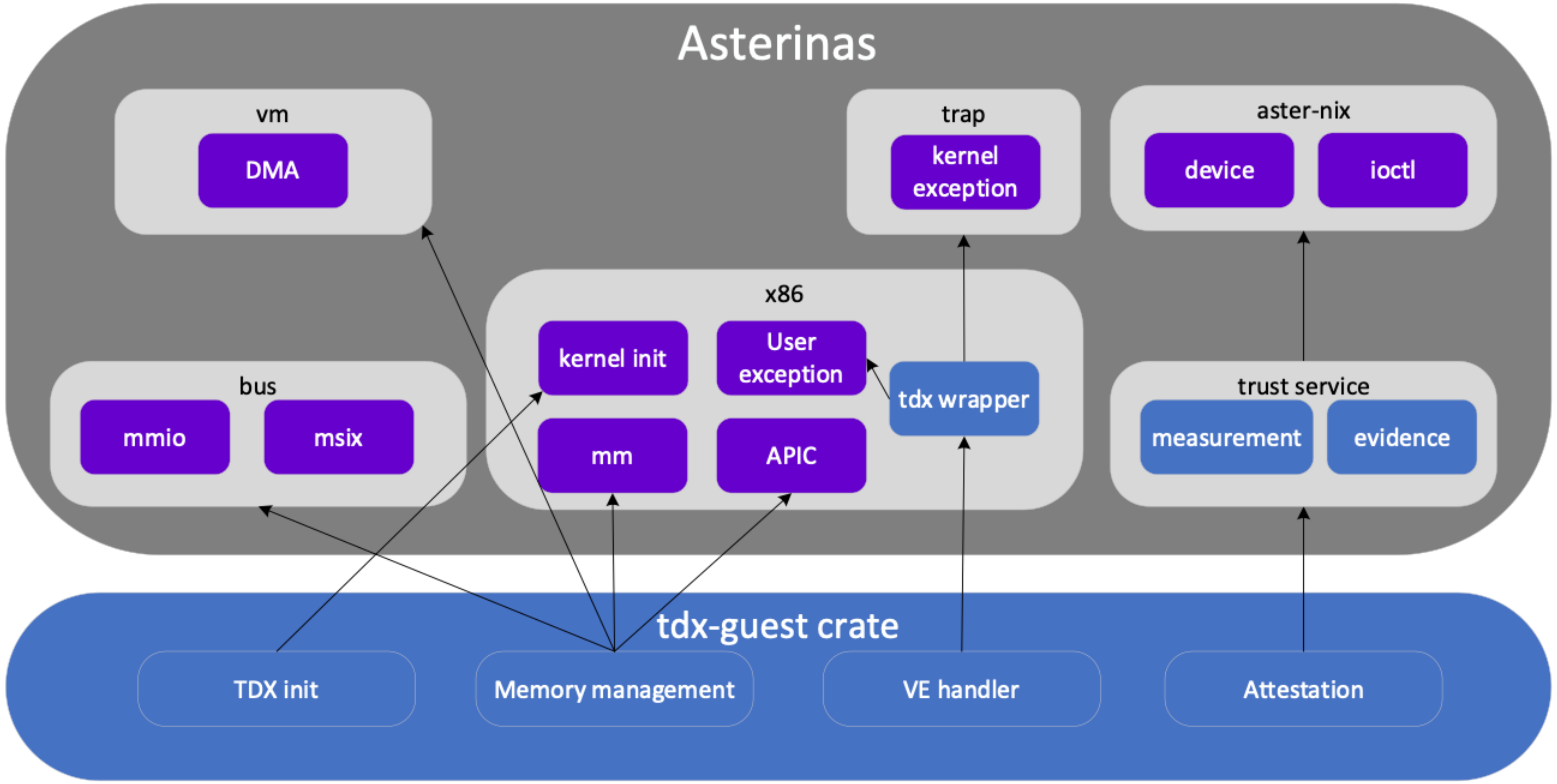
- Private Memory vs. Shared Memory
  - Private: Secure EPT via TDX instruction module
  - Shared: Shared EPT owned by VMM



- Private Memory Allocation
  - Guest pages allocated by VMM in PENDING state
  - TD Guest need to accept private page explicitly for using as private memory
- Private and Shared Conversion
  - TD Guest notify VMM for page remapping.
  - VMM call TDX instruction module remap page between shared EPT and secure EPT
  - TD Guest need additional page acceptance flow for shared page to private page

Asterinas and TDX integration update

- Asterinas successfully support Intel TDX hardware environment



- Validated Asterinas & TDX features
  - TD Guest: Boot-up, Virtualization Exception, Memory and MMIO
  - Driver: virtio, console, storage, network, attestation
- Future Plan
  - Features: TDX 1.5 & 2.0, Debug, Trust Service
  - Test with more workloads and devices
  - Performance Benchmark

Thank You

Asterinas

http://github.com/asterinas/asterinas