Trust-V: Toward Secure and Reliable Storage for Trusted Execution Environments*

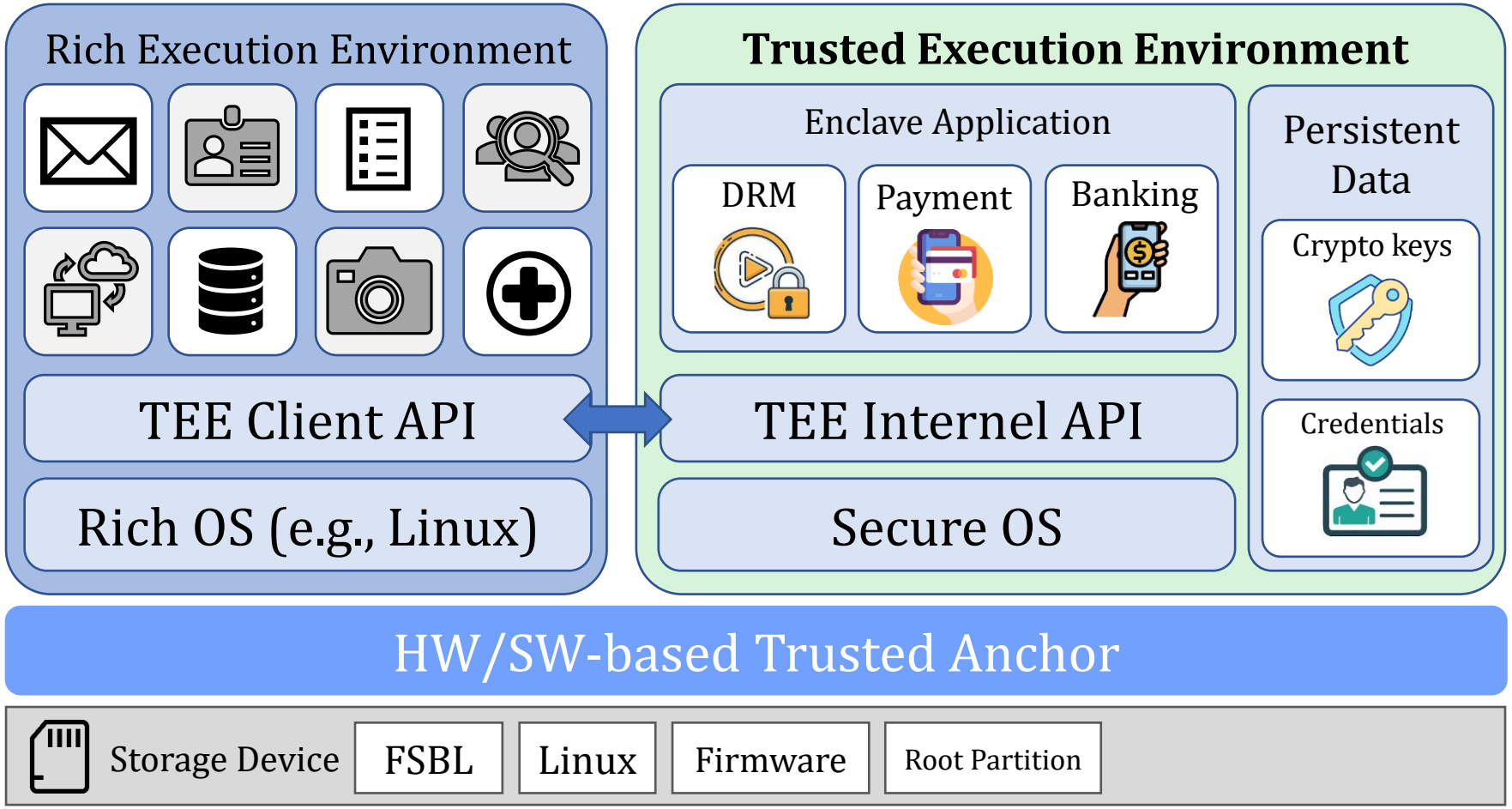
Jinsoo Jang

Chungnam National University

*International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS' 2026)

Trusted Execution Environment (TEE)

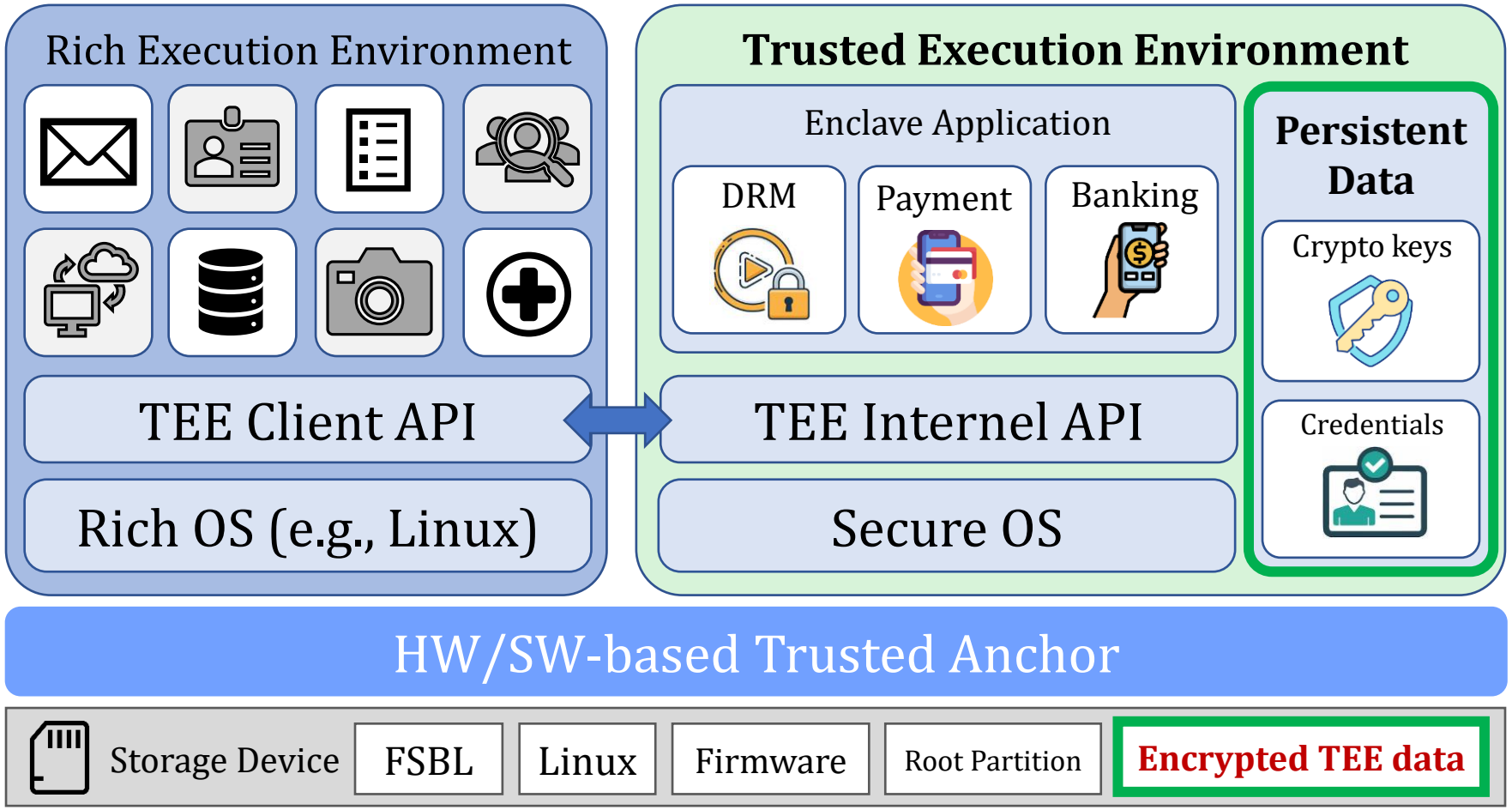
TEE Isolates and protects security-critical services from REE



Secure Storage for TEE

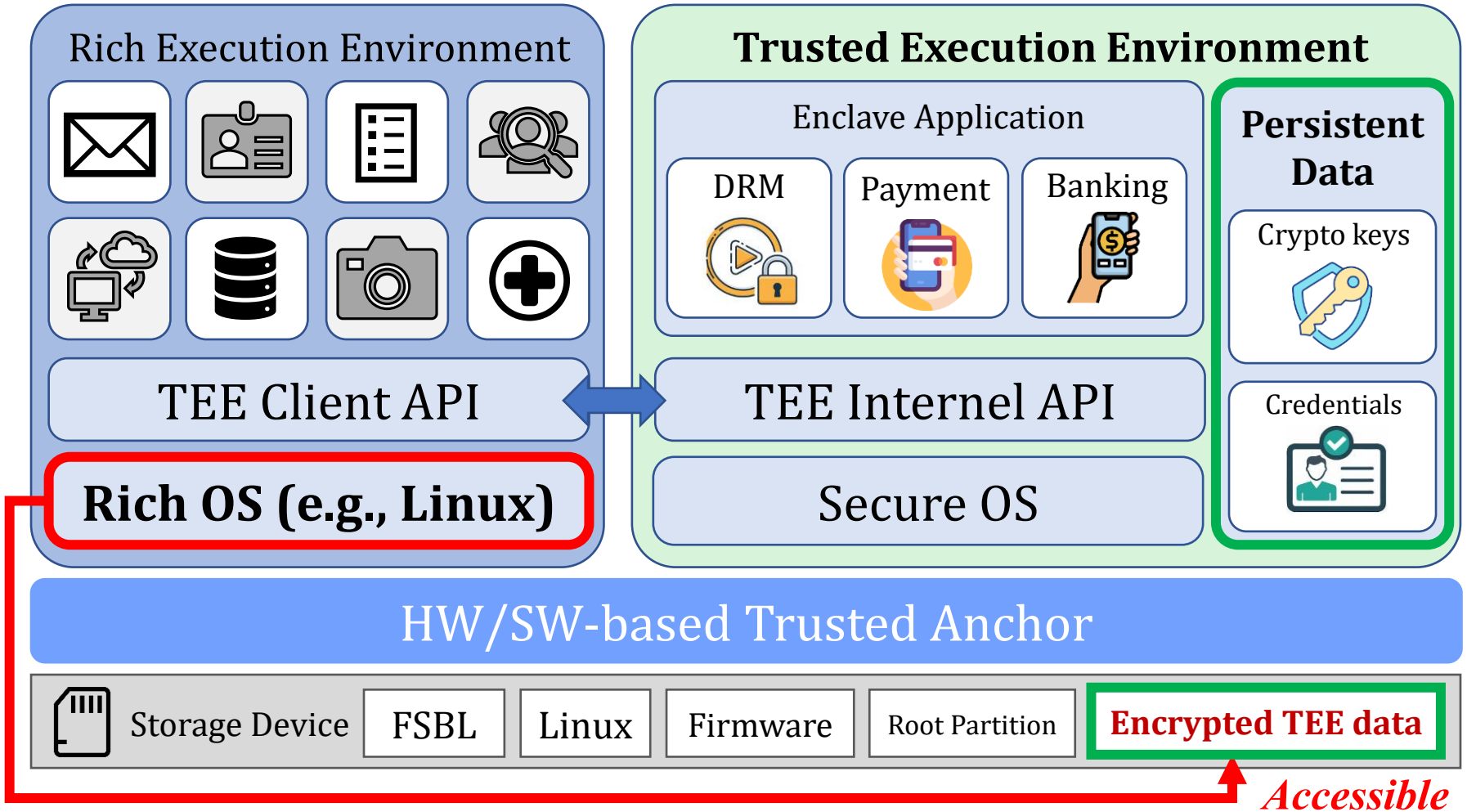
Existing solutions persistently store TEE data by ***encrypting it with a TEE key***

✓ *E.g., Intel SGX data sealing; OP-TEE's Secure Storage*



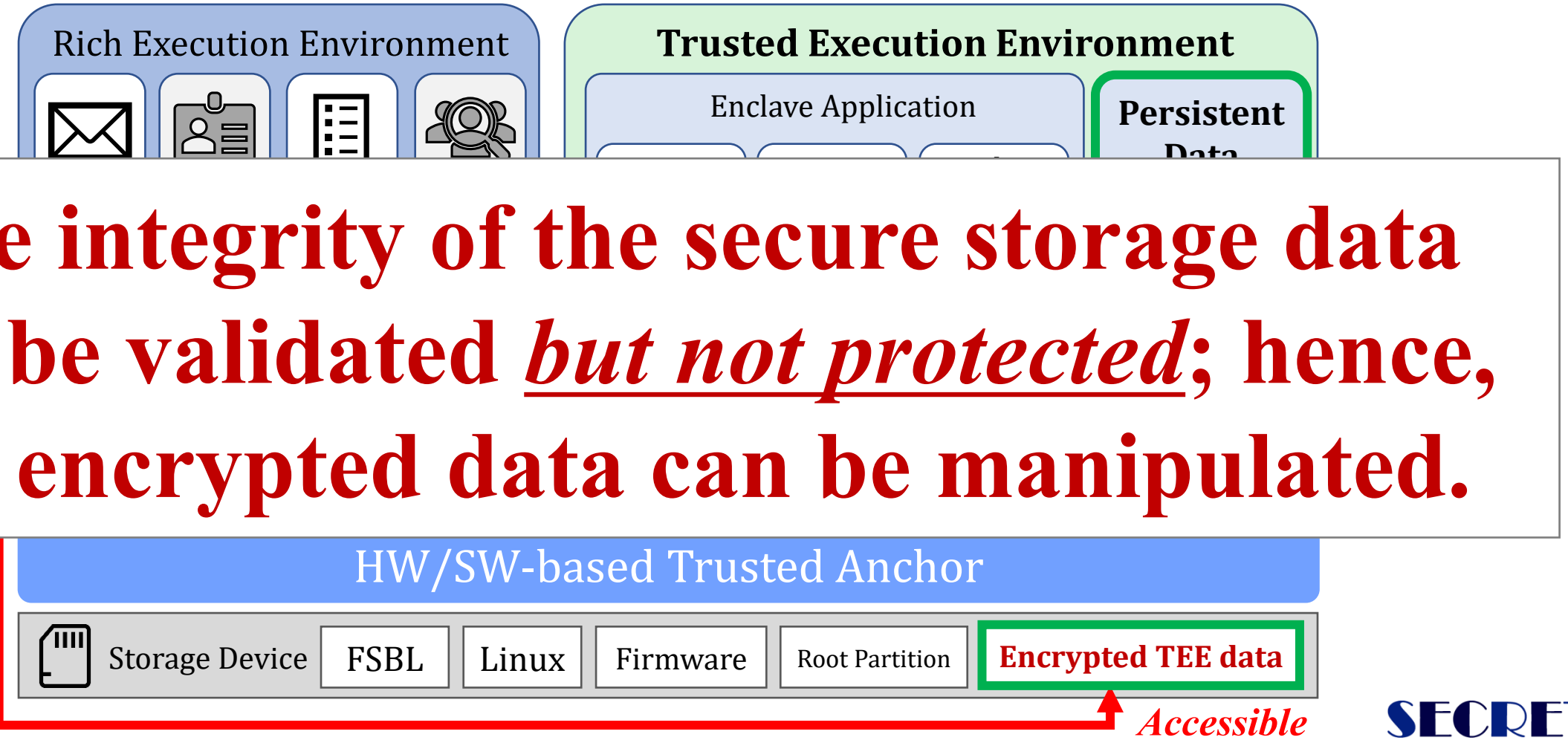
Limitation of Secure Storage

*The encrypted data **can be accessed (manipulated)** by the untrusted OS*



Limitation of Secure Storage

*The encrypted data **can be accessed (manipulated)** by the untrusted OS*



The integrity of the secure storage data can be validated *but not protected*; hence, the encrypted data can be manipulated.

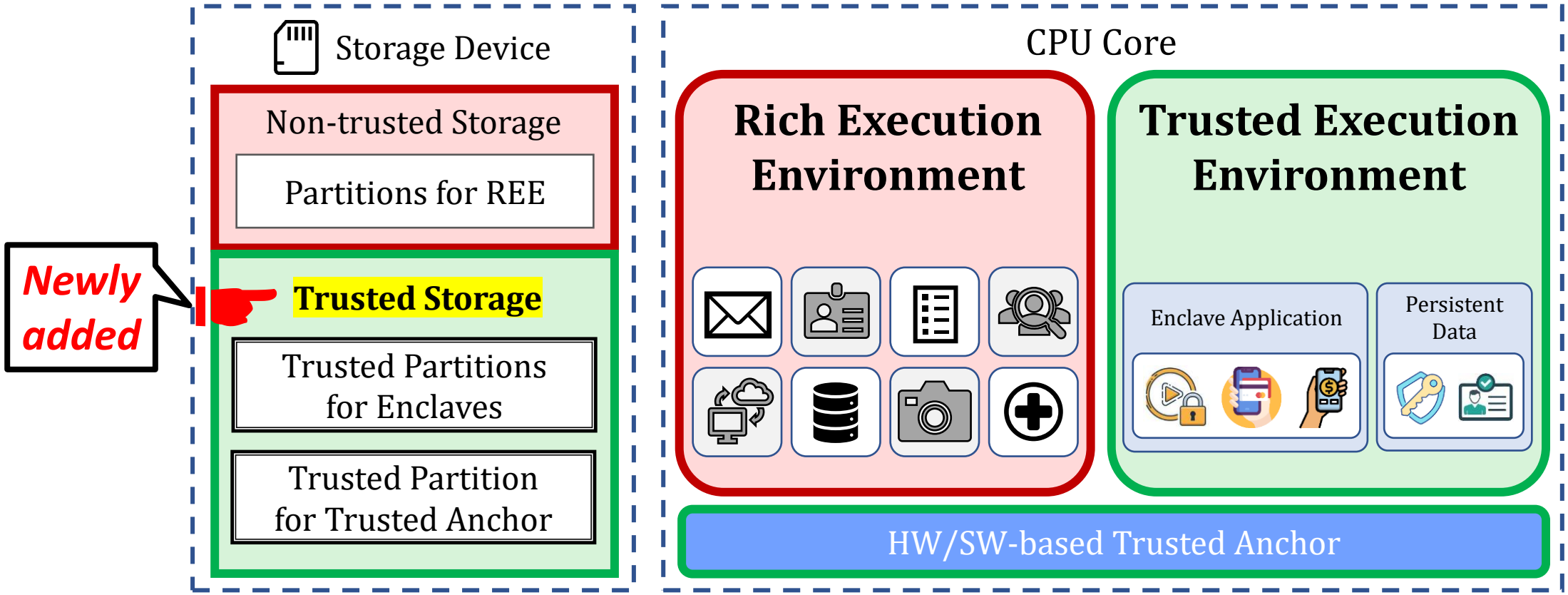
Our Goal & Assumption

- Ensure integrity and confidentiality of TEE persistent data
  - ✓ Provide runtime isolation for TEE data on the storage device

- Assumption
  - ✓ Availability attacks on the TEE are not considered
  - ✓ An attacker can issue malicious block I/O requests to the storage device
  - ✓ Physical attacks (e.g., SD card replacement or removal) are not considered
  - ✓ Side channel attacks are not considered
  - ✓ The monitor firmware (including the security monitor) is trusted
  - ✓ Secure boot

Our Approach: Trusted Storage

Trusted Storage *is a storage partition reserved for the TEE*



Ensure integrity of the TEE data stored in Trusted Storage

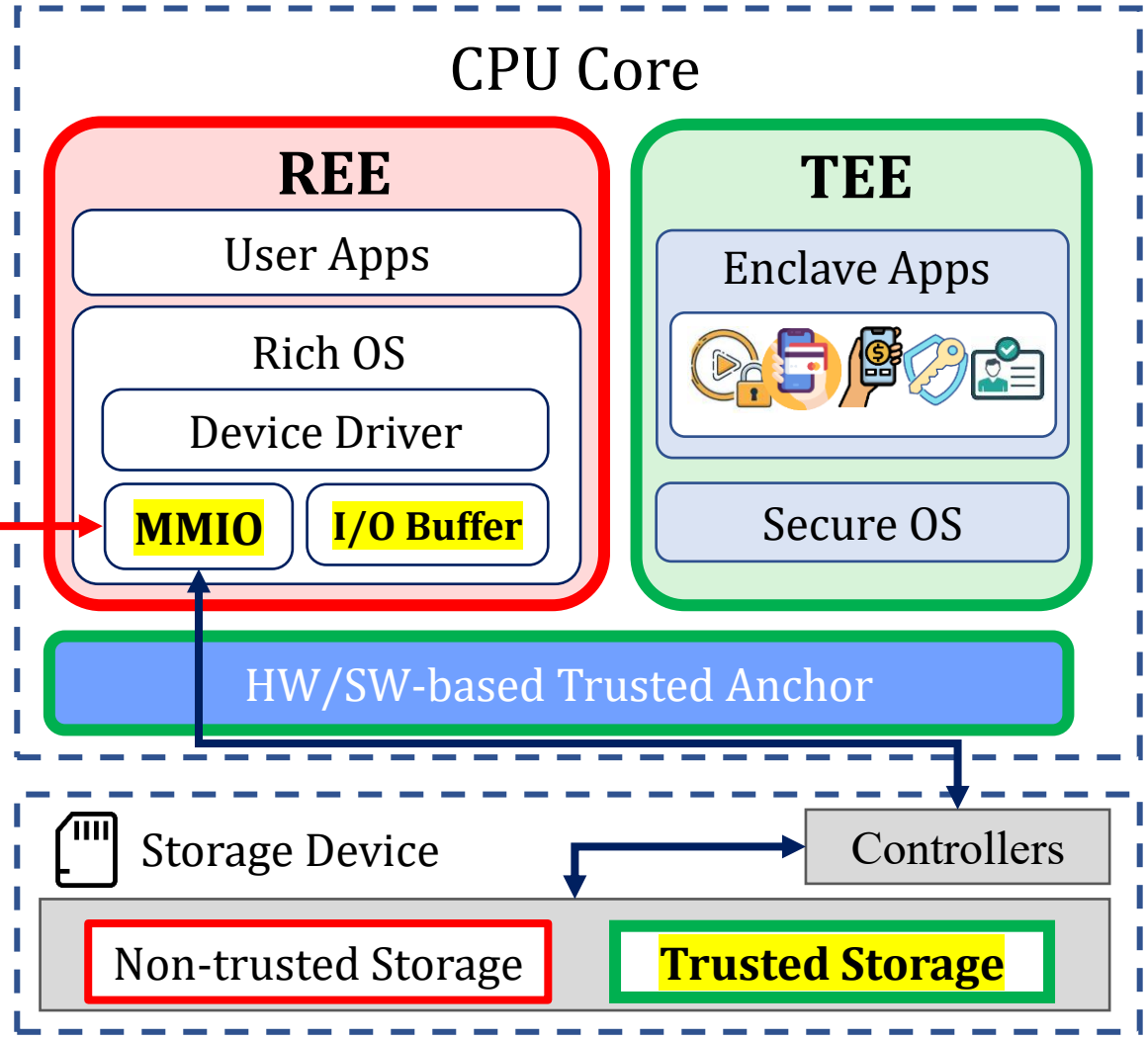
Our Approach: Trusted Storage

Protocol Message (e.g., SPI)

```
struct spi_transfer {
  ...
  const void * tx_buf;
  void * rx_buf;
  ...
};
```

I/O Buffer

Protocol message for communication with peripheral is written to MMIO region



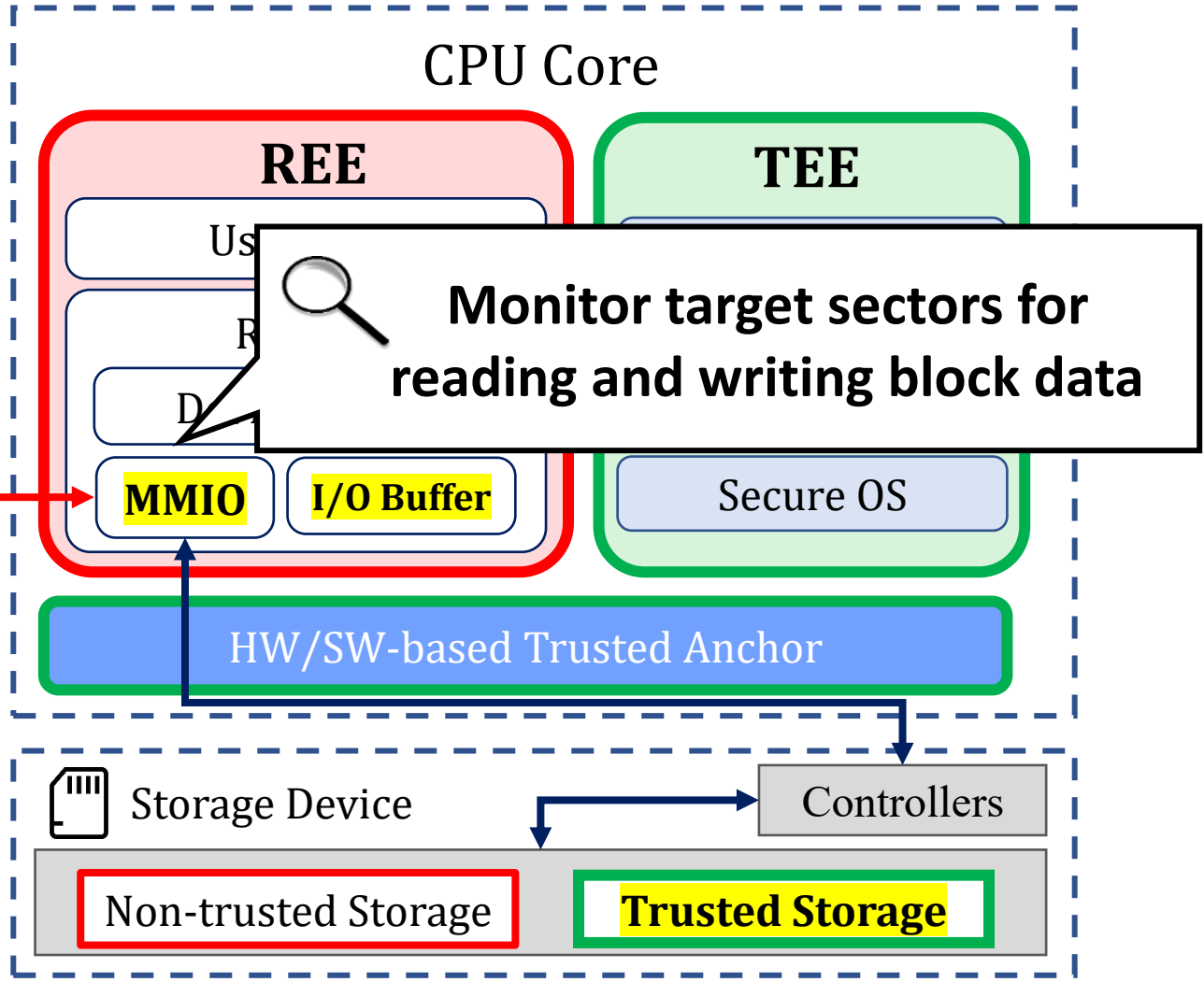
Our Approach: Trusted Storage

Protocol Message (e.g., SPI)

```
struct spi_transfer {
  ...
  const void * tx_buf;
  void * rx_buf;
  ...
};
```

I/O Buffer

Protocol message for communication with peripheral is written to MMIO region

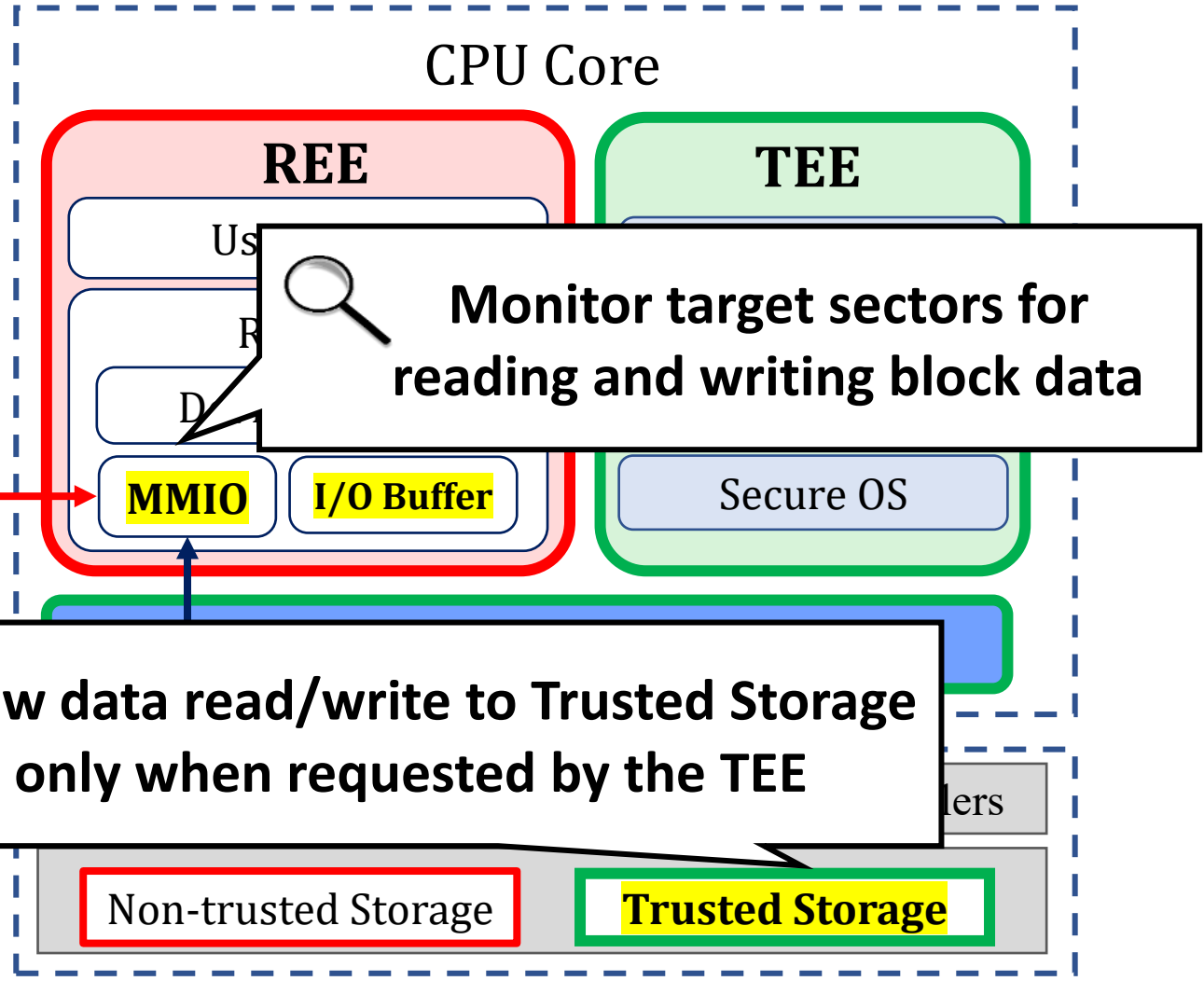


Our Approach: Trusted Storage

Protocol Message (e.g., SPI)

```
struct spi_transfer {
  ...
  const void * tx_buf;
  void * rx_buf;
  ...
};
```

I/O Buffer

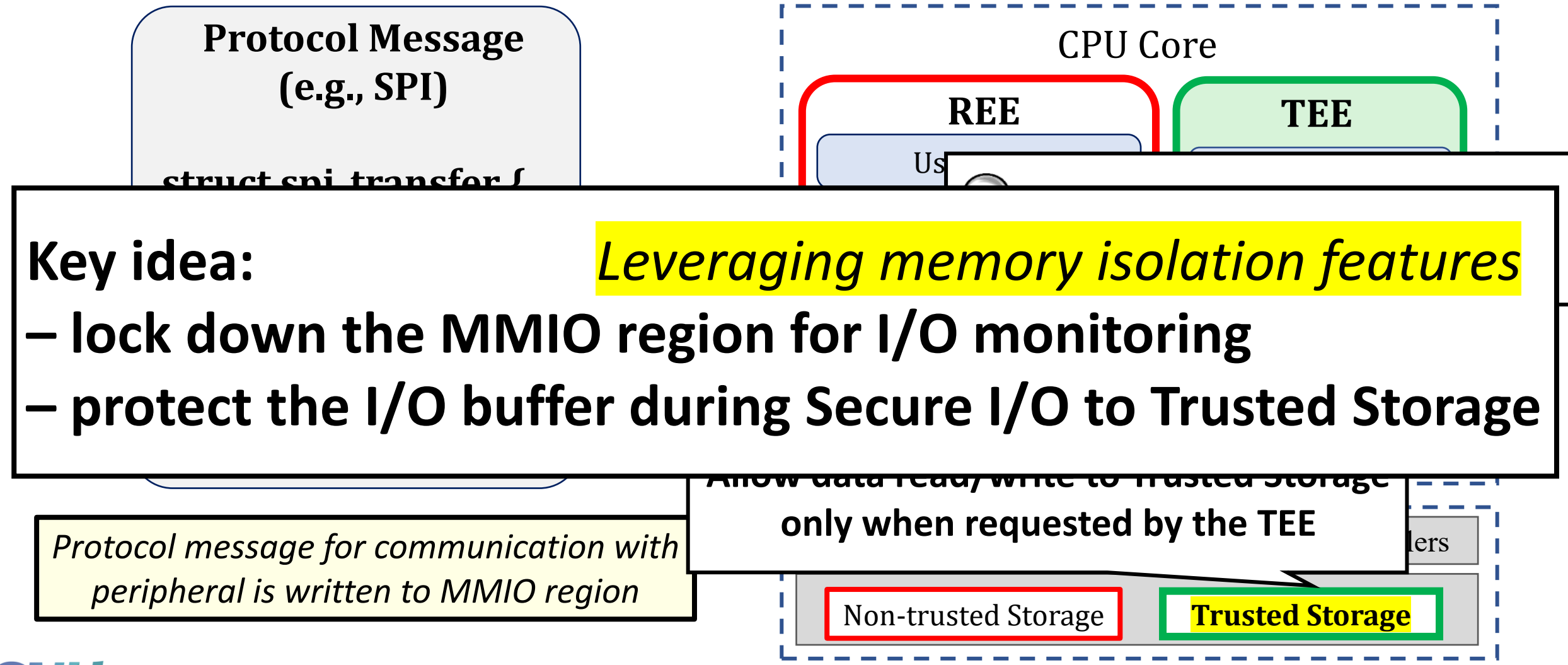


Monitor target sectors for reading and writing block data

Allow data read/write to Trusted Storage only when requested by the TEE

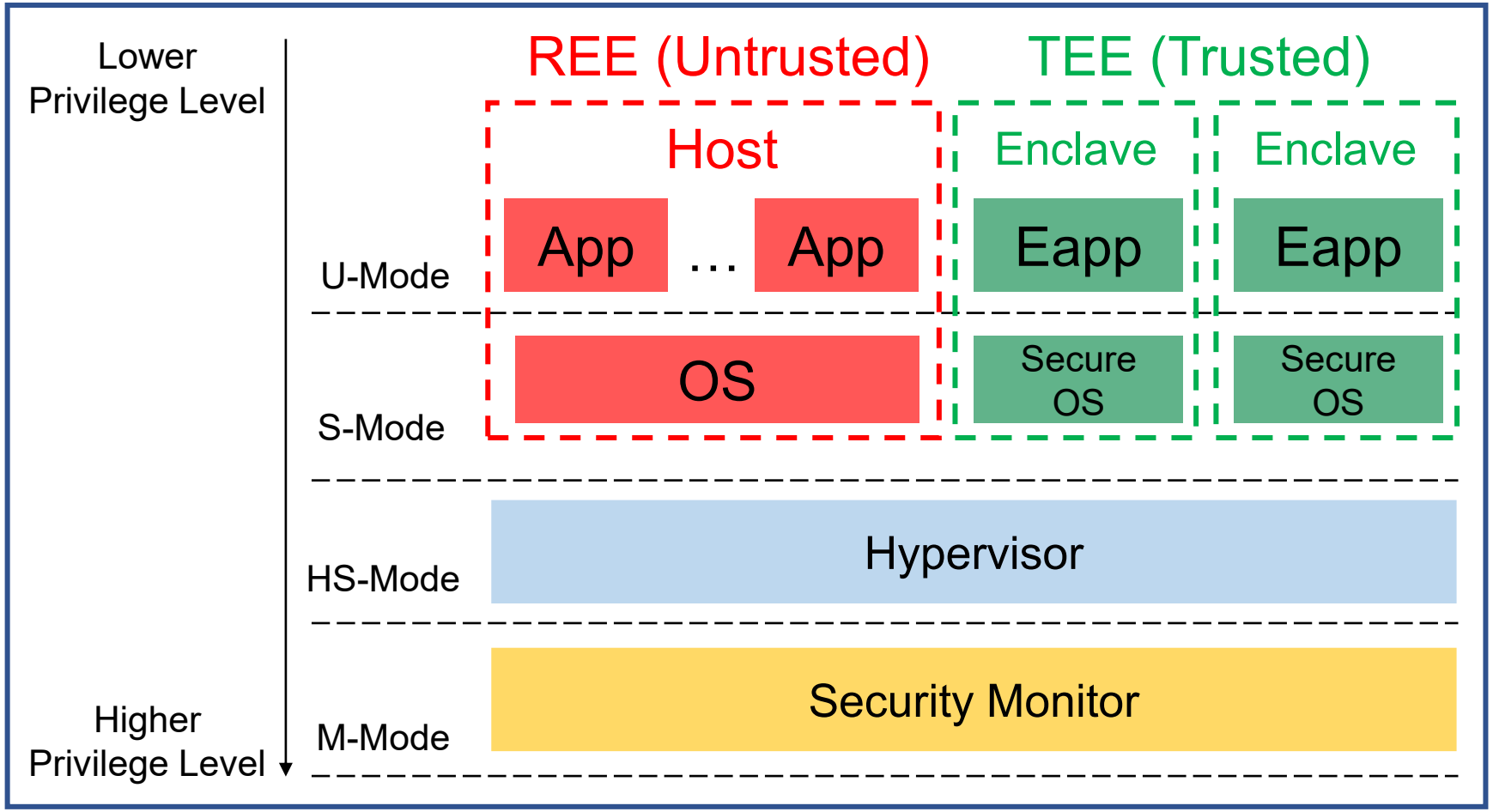
Protocol message for communication with peripheral is written to MMIO region

Our Approach: Trusted Storage



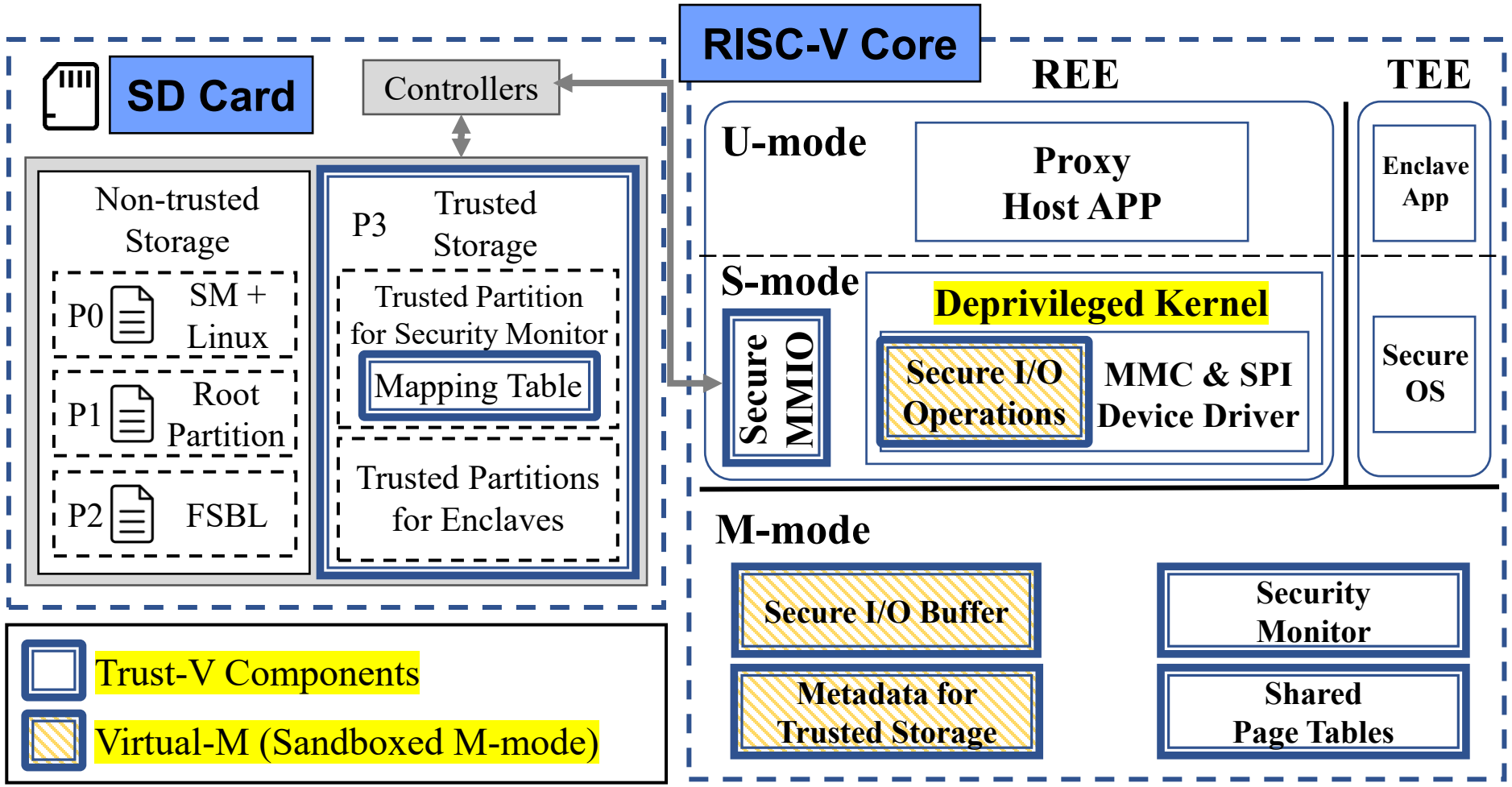
RISC-V Architecture

CPU privilege modes in RISC-V architecture



Trust-V – Overview

Prototype of Trusted Storage in RISC-V architectures



CNU

SECRET

Trust-V – Overview

Prototype of Trusted Storage in RISC-V architectures

Key primitives for secure IO

① Two memory isolation approaches
- Paging-based and SUM-based

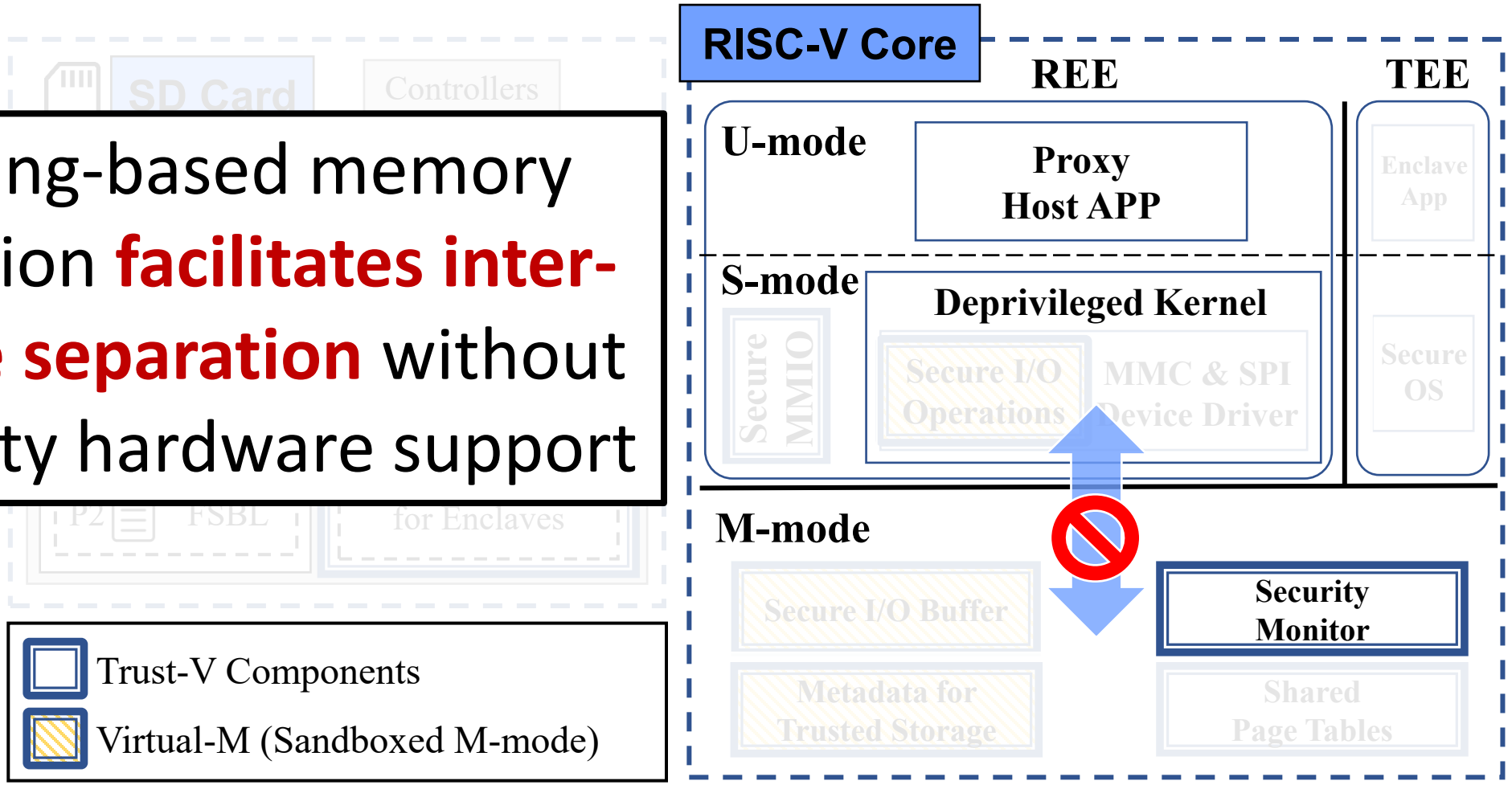
② M-mode sandboxing

③ Trusted partition management

④ I/O context maintenance

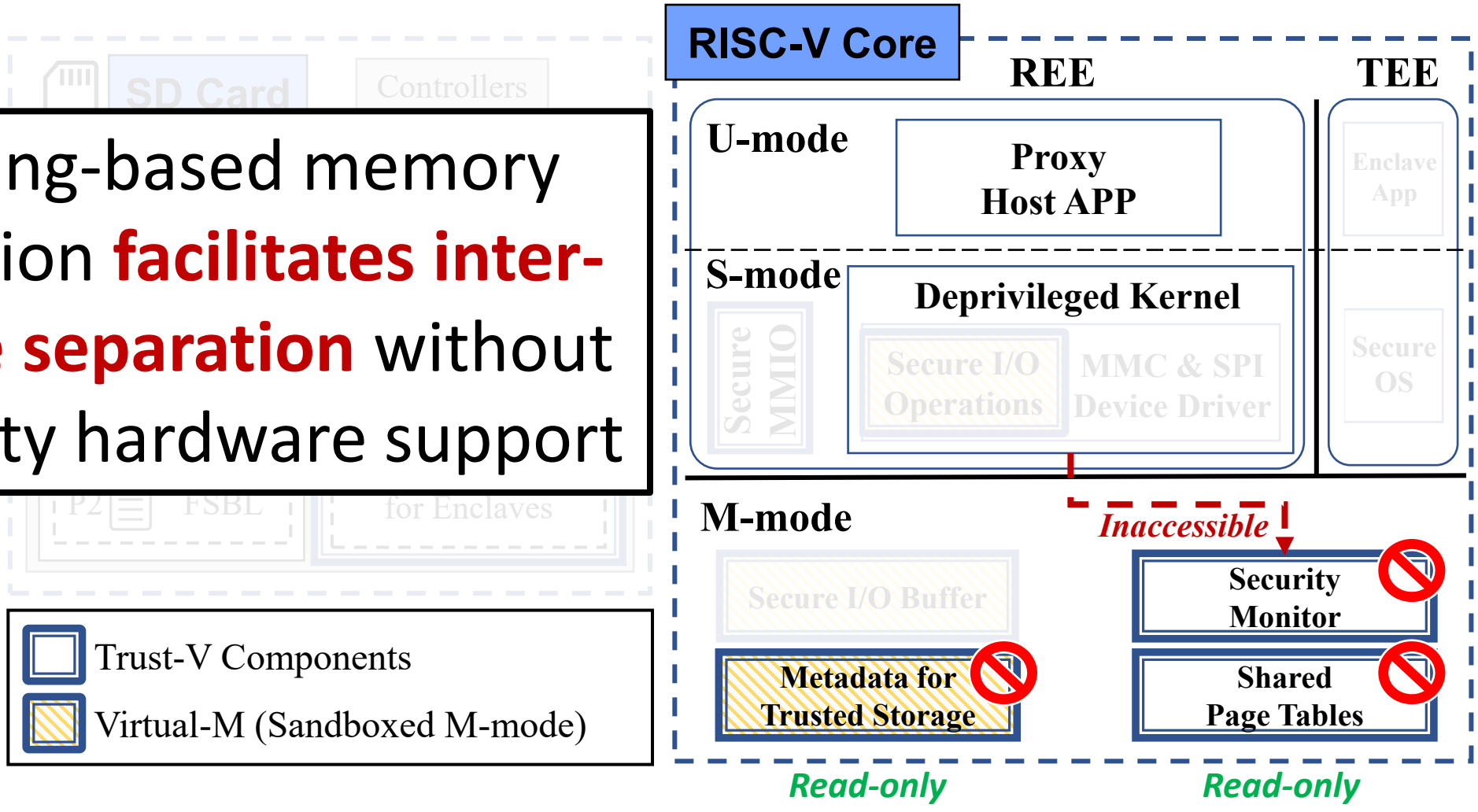
(P1) Memory Isolation : Paging

Paging-based memory isolation **facilitates inter-mode separation** without security hardware support



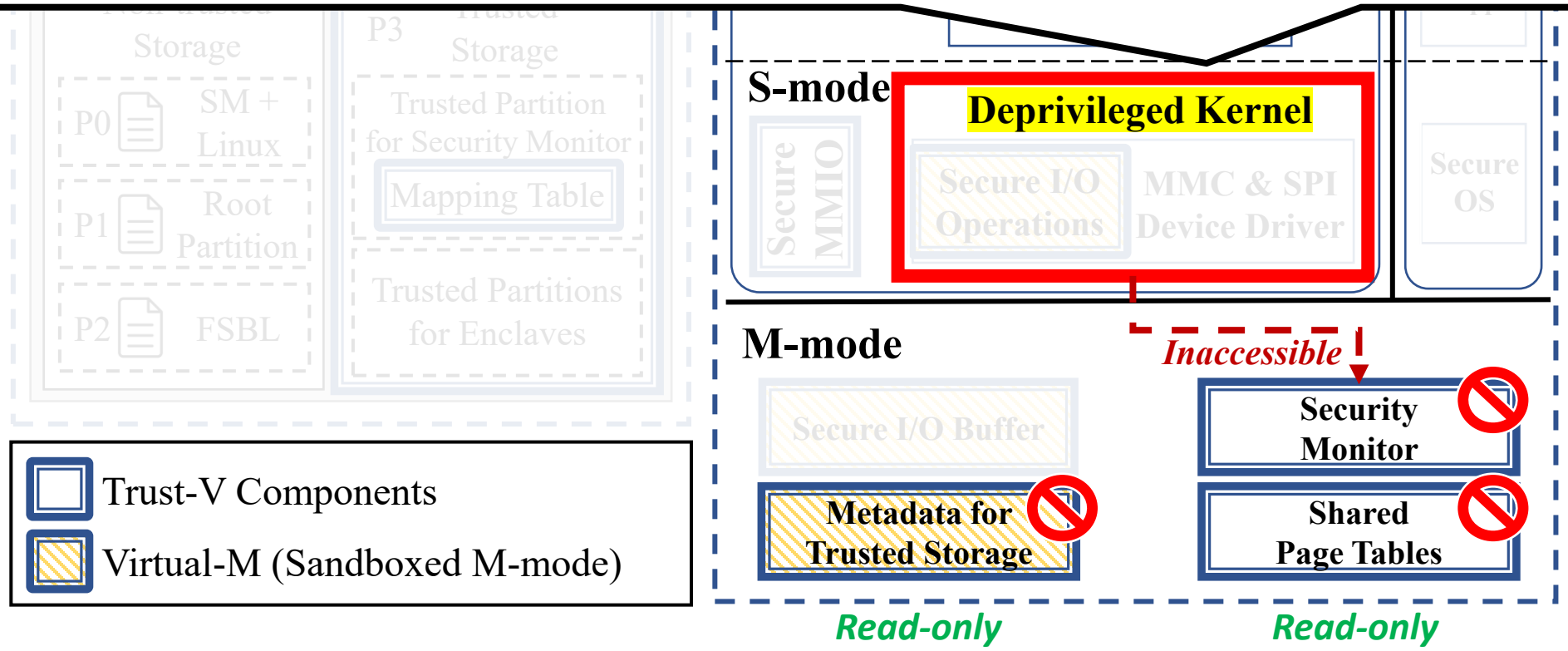
(P1) Memory Isolation : Paging

Paging-based memory isolation **facilitates inter-mode separation** without security hardware support



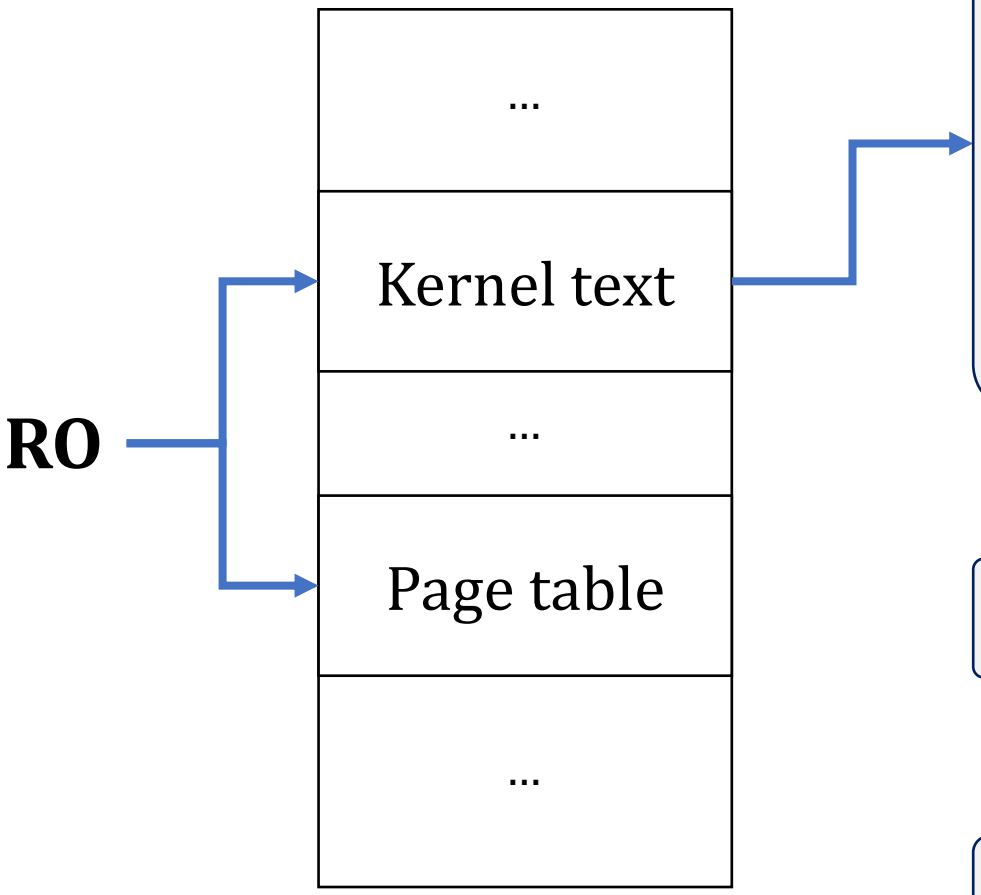
(P1) Memory Isolation : Paging

Trust-V deprivileges the kernel to enforce the paging-based memory isolation



(P1) Memory Isolation : Paging

Deprivileging the kernel



Security critical operations

CSRW SATP
CSRW TVEC
CSRW SSTATUS

Instructions for updating system registers

*ptep = pteval;
*pmdp = pmdval;

Instructions for updating page table entries

Replace them with ECALL

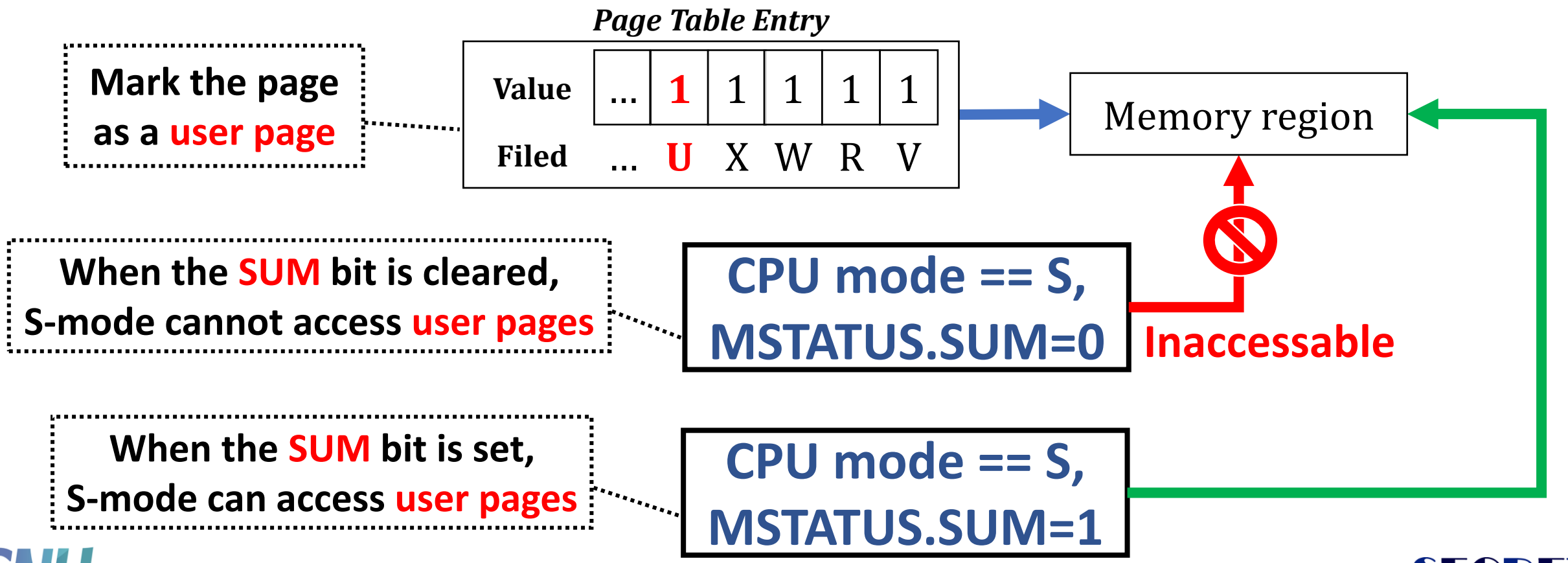
sbi_ecall(TRUSTV_EXT, TRUSTV_EMUL_xx, value);

ECALL instruction trap into M-mode

The security monitor validates and emulates the requested operations

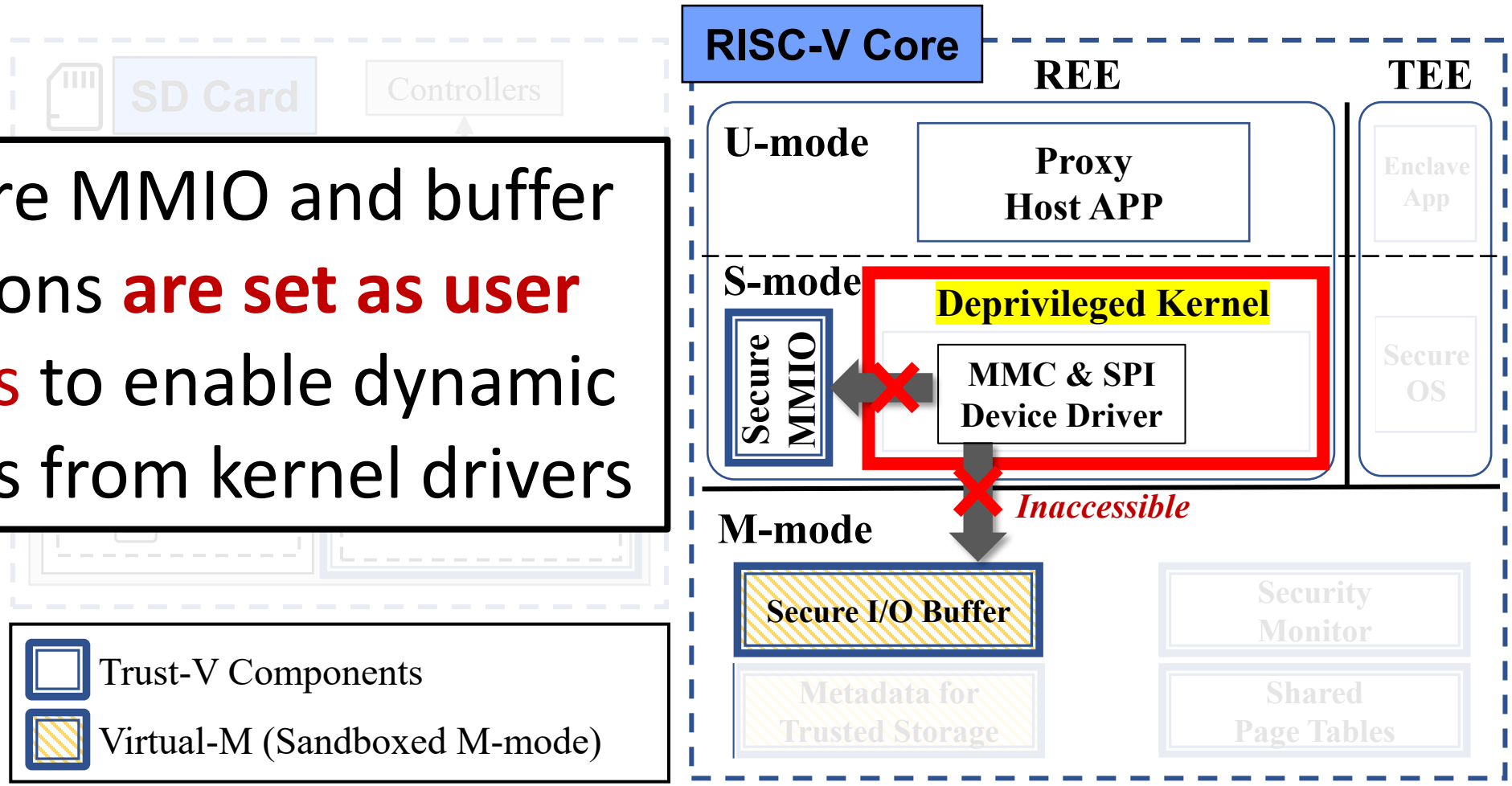
(P1) Memory Isolation : SUM

SUM-based memory isolation facilitates ***on-demand dynamic control of memory access for secure I/O*** *operations*

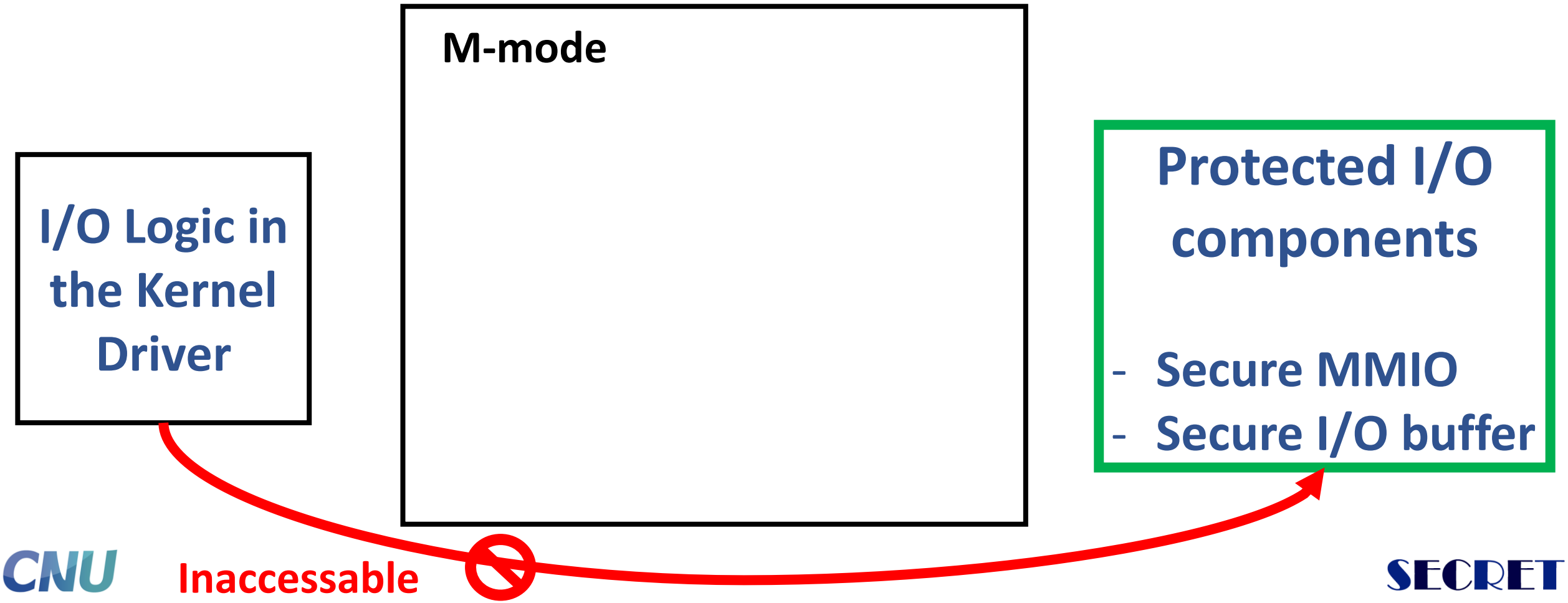


(P1) Memory Isolation : SUM

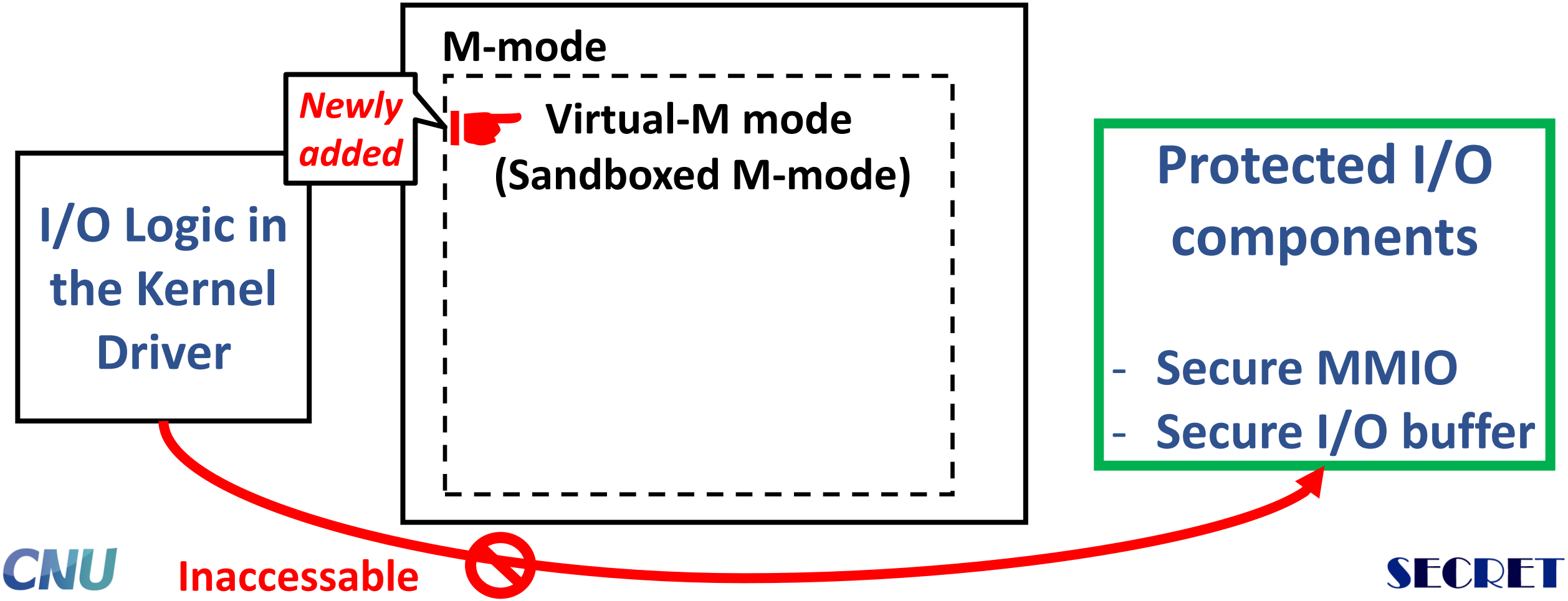
Secure MMIO and buffer regions **are set as user pages** to enable dynamic access from kernel drivers



(P2) M-mode sandboxing



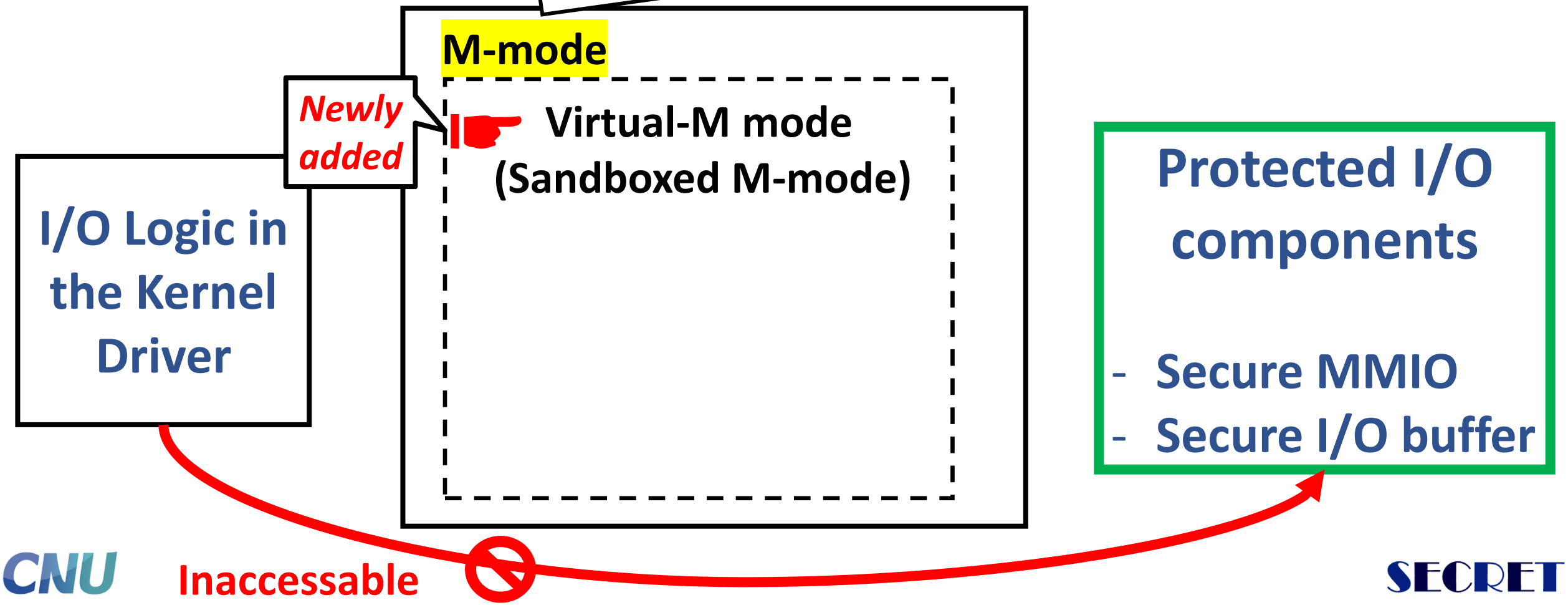
(P2) M-mode sandboxing



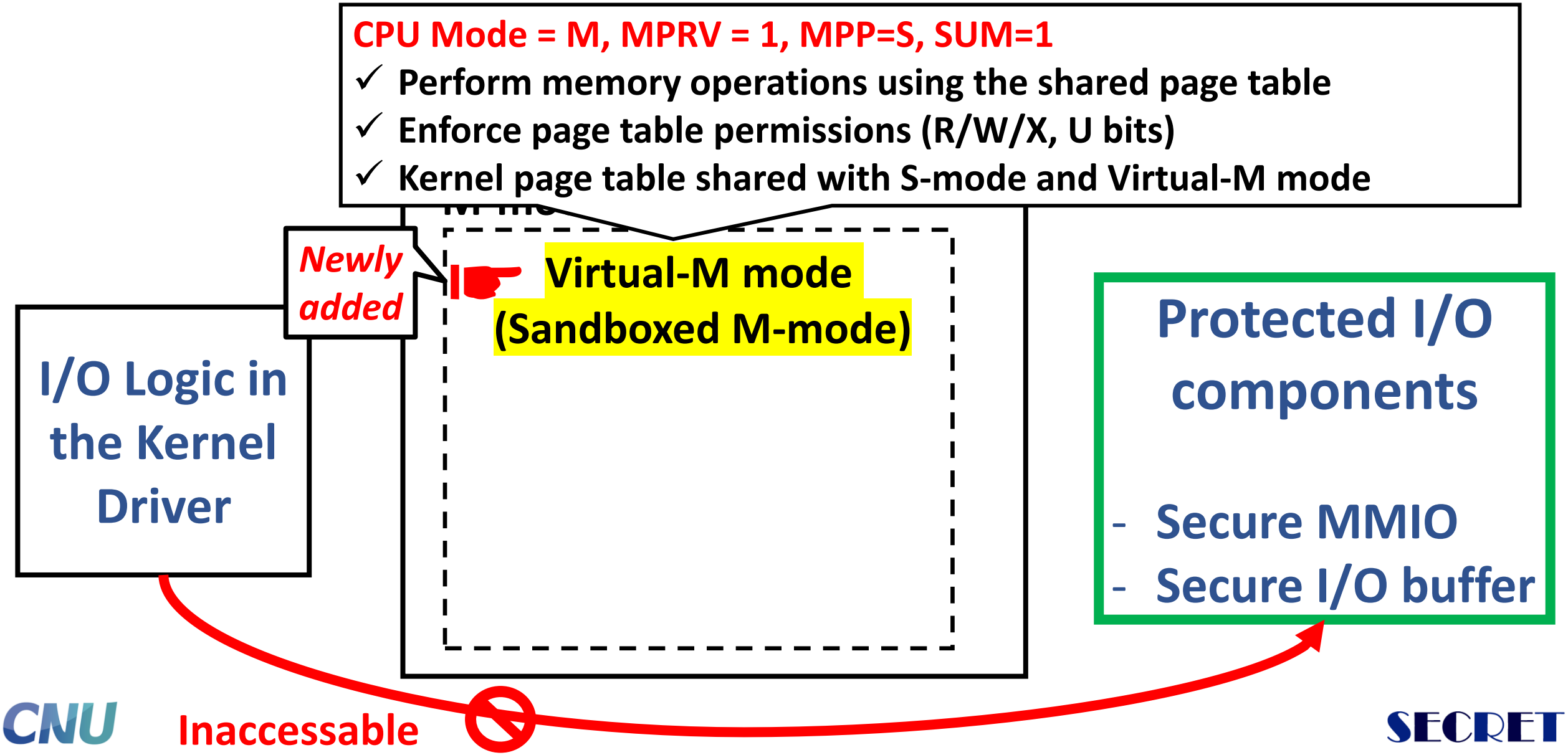
(P2) M-mo

CPU Mode = M, MPRV = 0

- ✓ The native highest-privilege mode in RISC-V
- ✓ Manage system resources (e.g., page tables)
- ✓ Manage components for Secure I/O

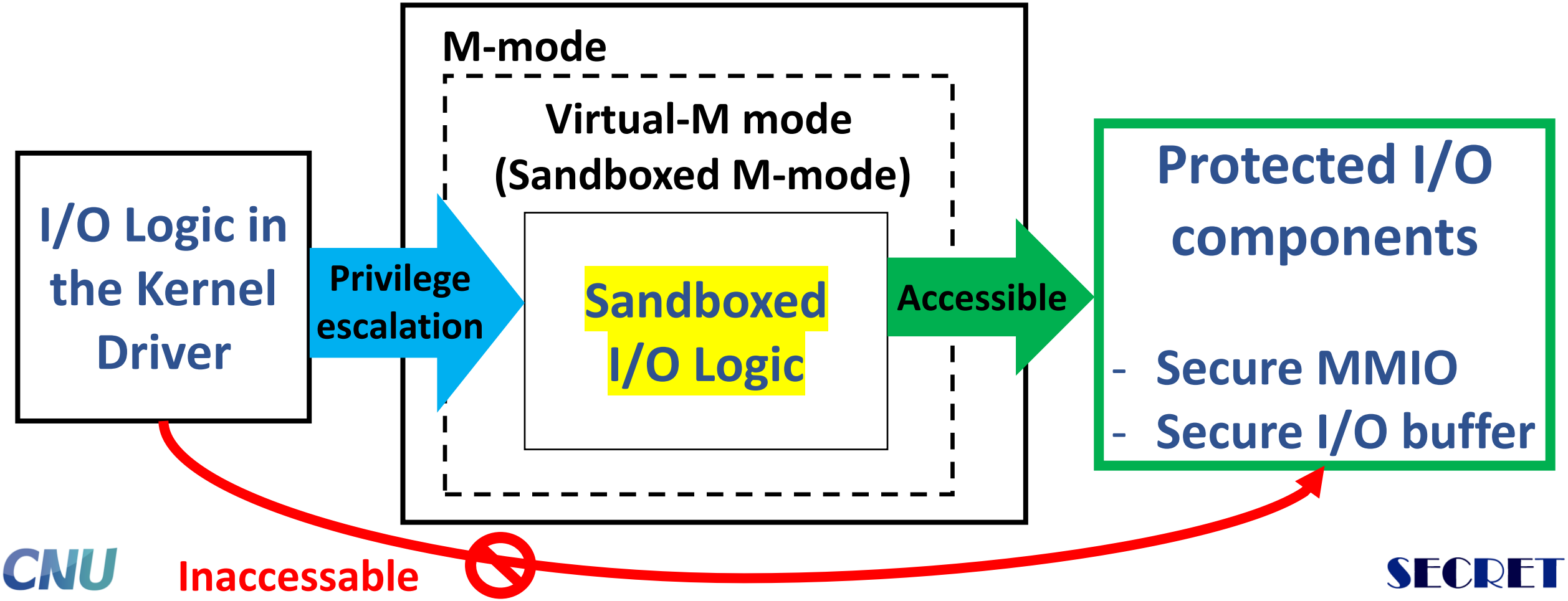


(P2) M-mode sandboxing



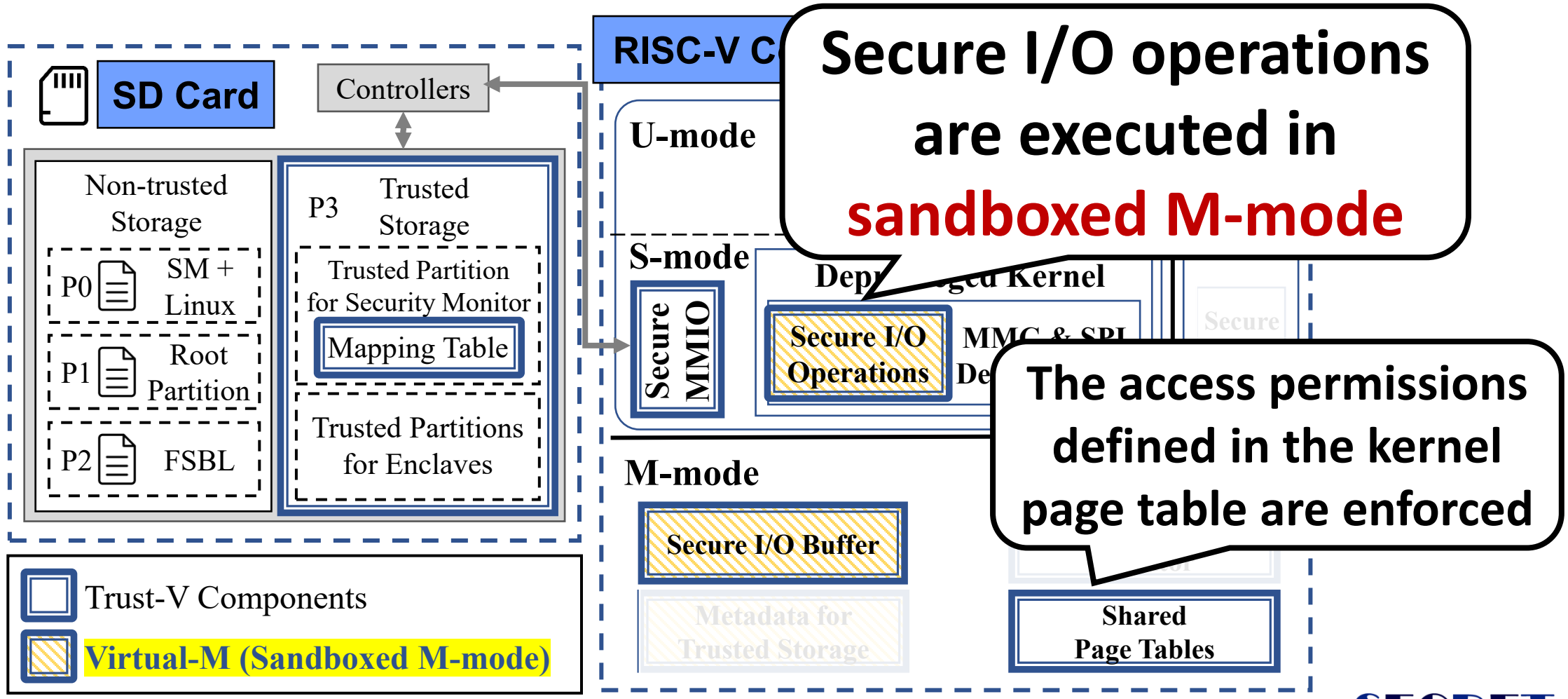
(P2) M-mode sandboxing

The M-mode sandbox enables code for secure I/O in the kernel driver to be executed with escalated privileges securely



(P2) M-mode sandboxing: Security

The M-mode sandbox enables privileged code to be embedded within the kernel

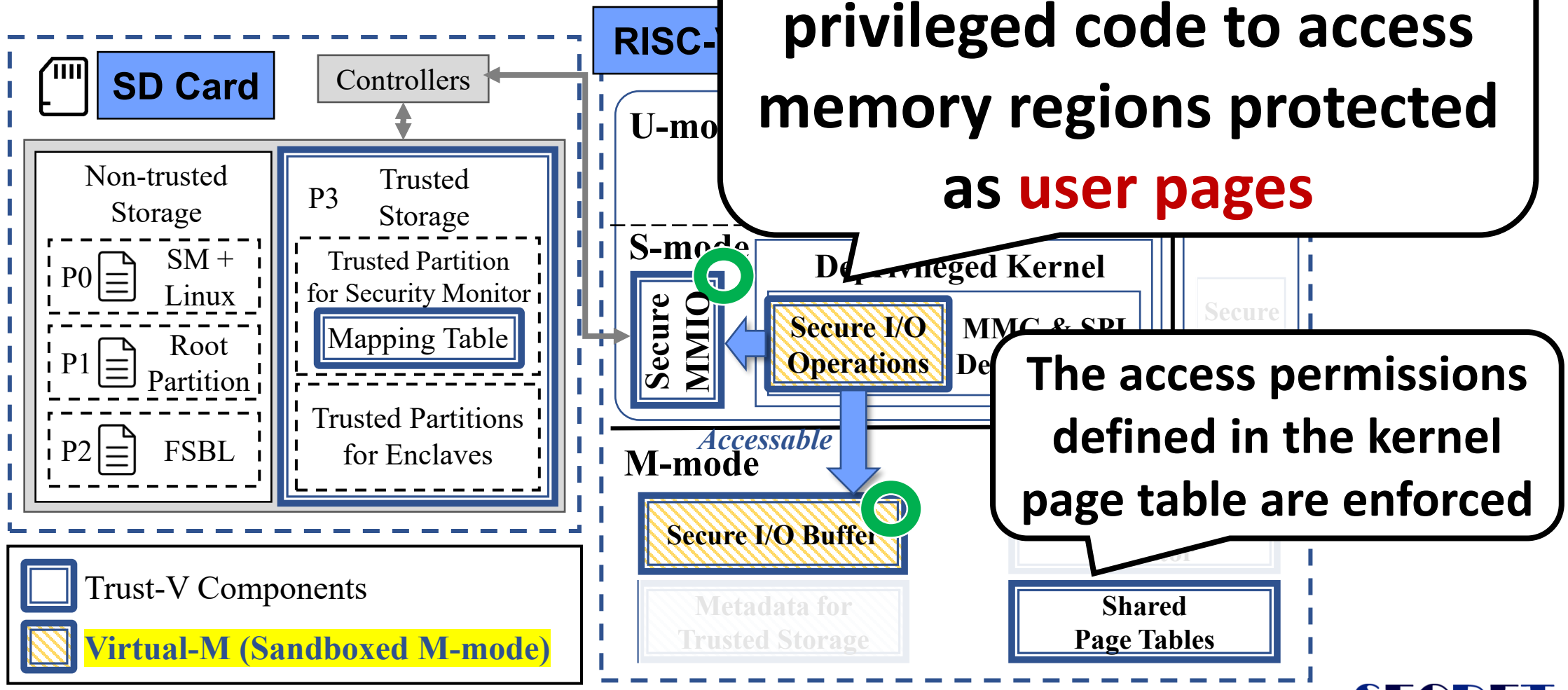


CNU

SECRET

(P2) M-mode sandboxing: Security

The M-mode sandbox enables privileged co



Enabling SUM allows privileged code to access memory regions protected as user pages

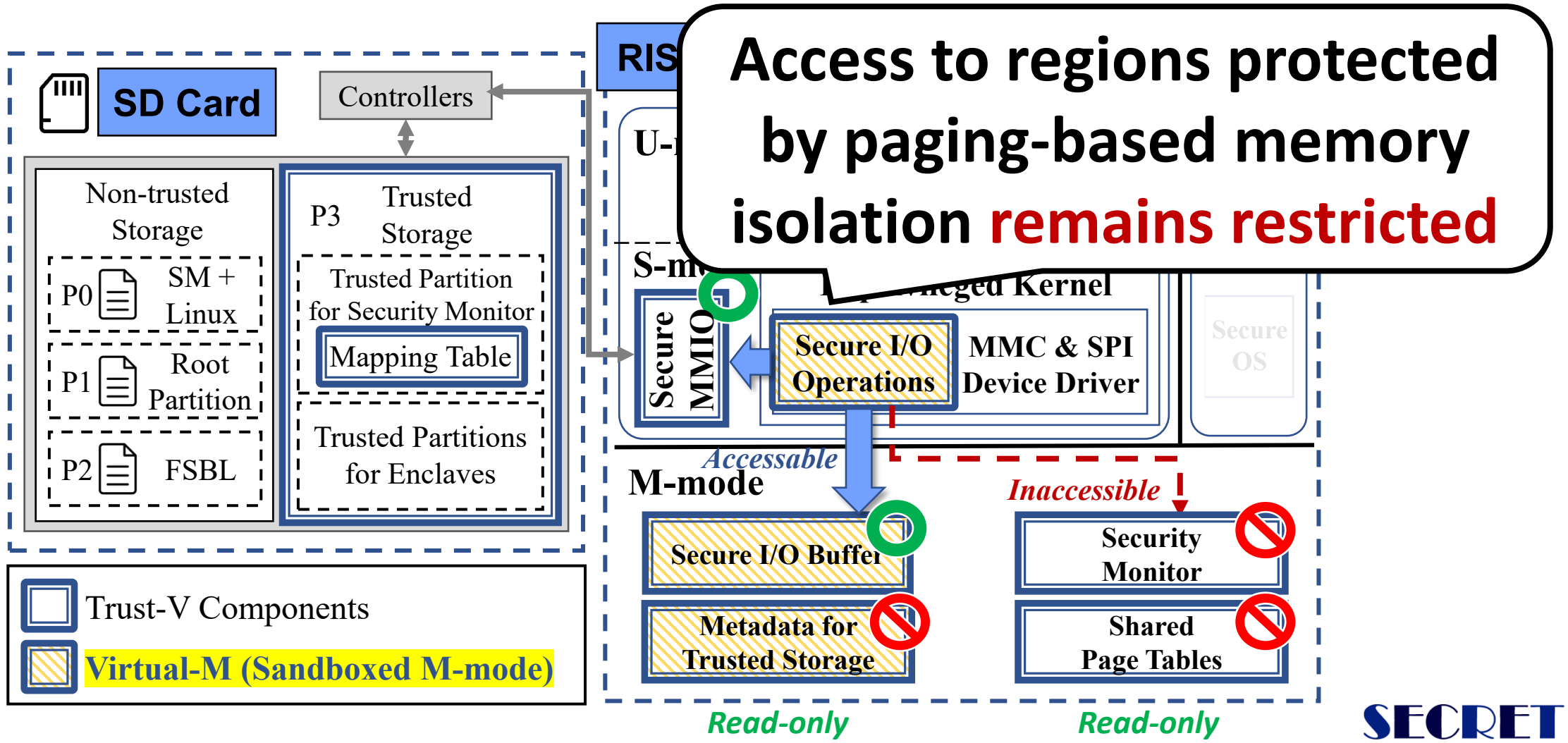
The access permissions defined in the kernel page table are enforced

CNU

SECRET

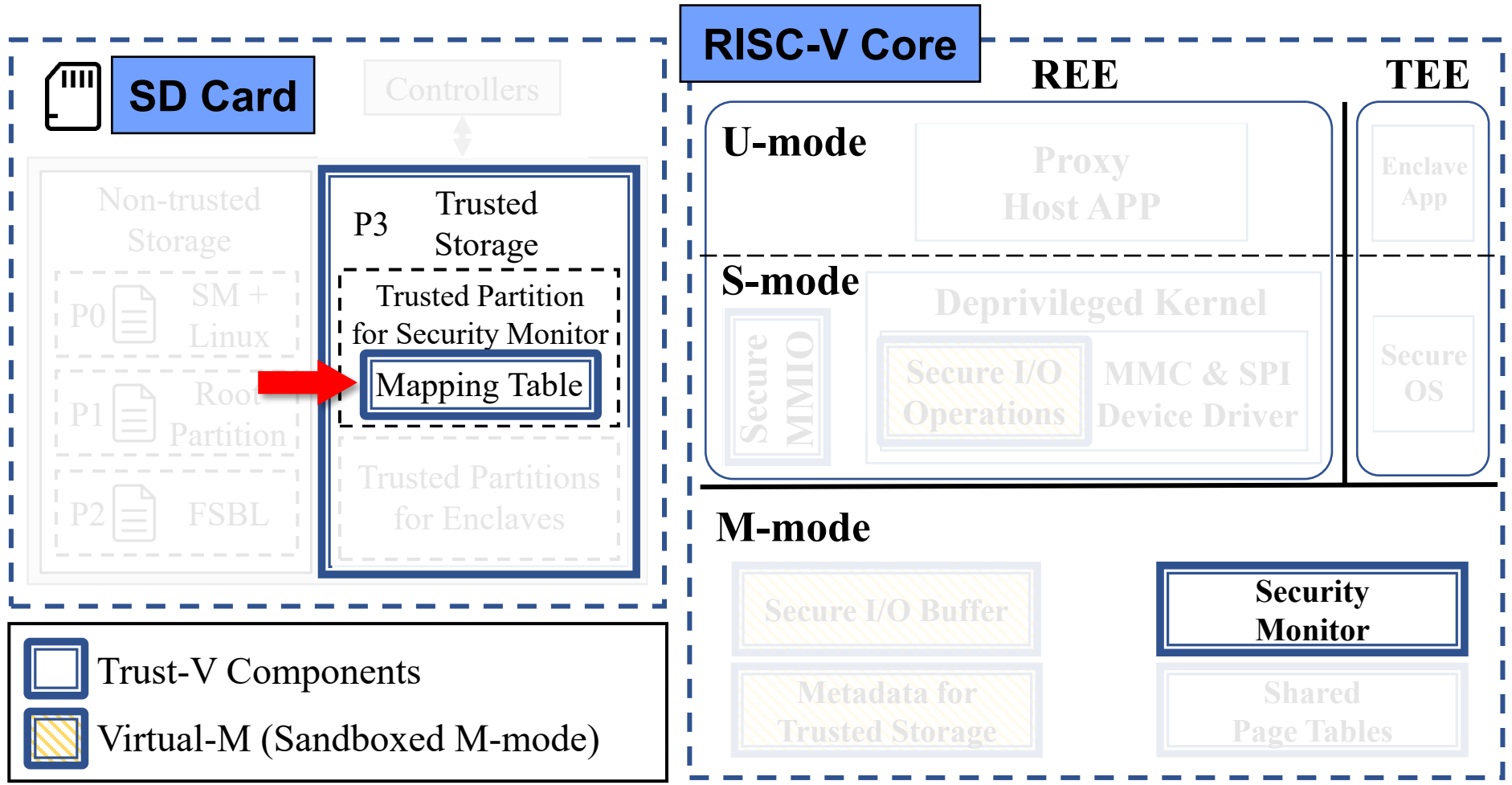
(P2) M-mode sandboxing: Security

The M-mode sandbox enables privileged code to be embedded within the kernel



(P3) Trusted Partition Management

The security monitor maintains the mapping table for managing the Trusted Partition

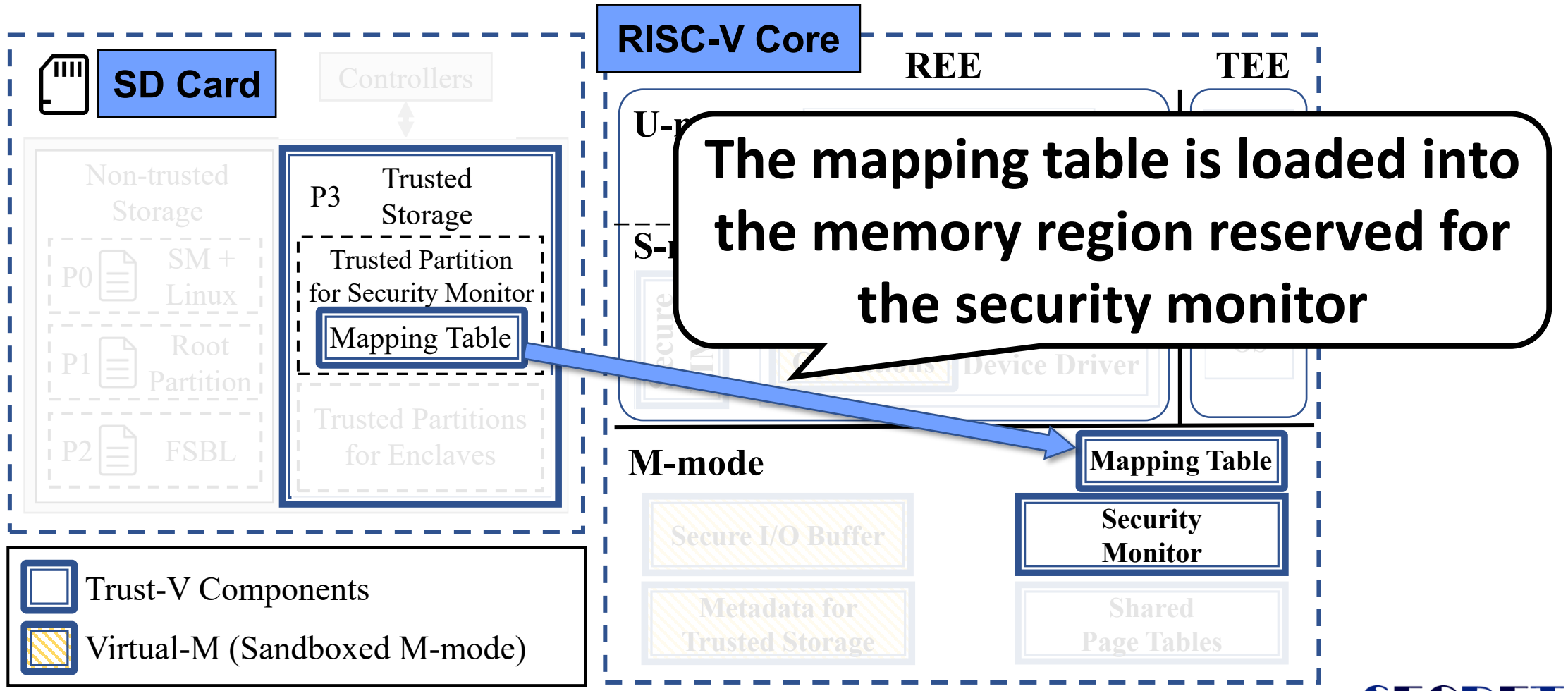


CNU

SECRET

(P3) Trusted Partition Management

The security monitor maintains the mapping table for managing the Trusted Partition



(P3) Trusted Partition Management

The se ***ging the Trusted Partition***

※ Trusted partition[0] used by the security monitor

Trusted partition[1] ➡ Value = Owner enclave hash

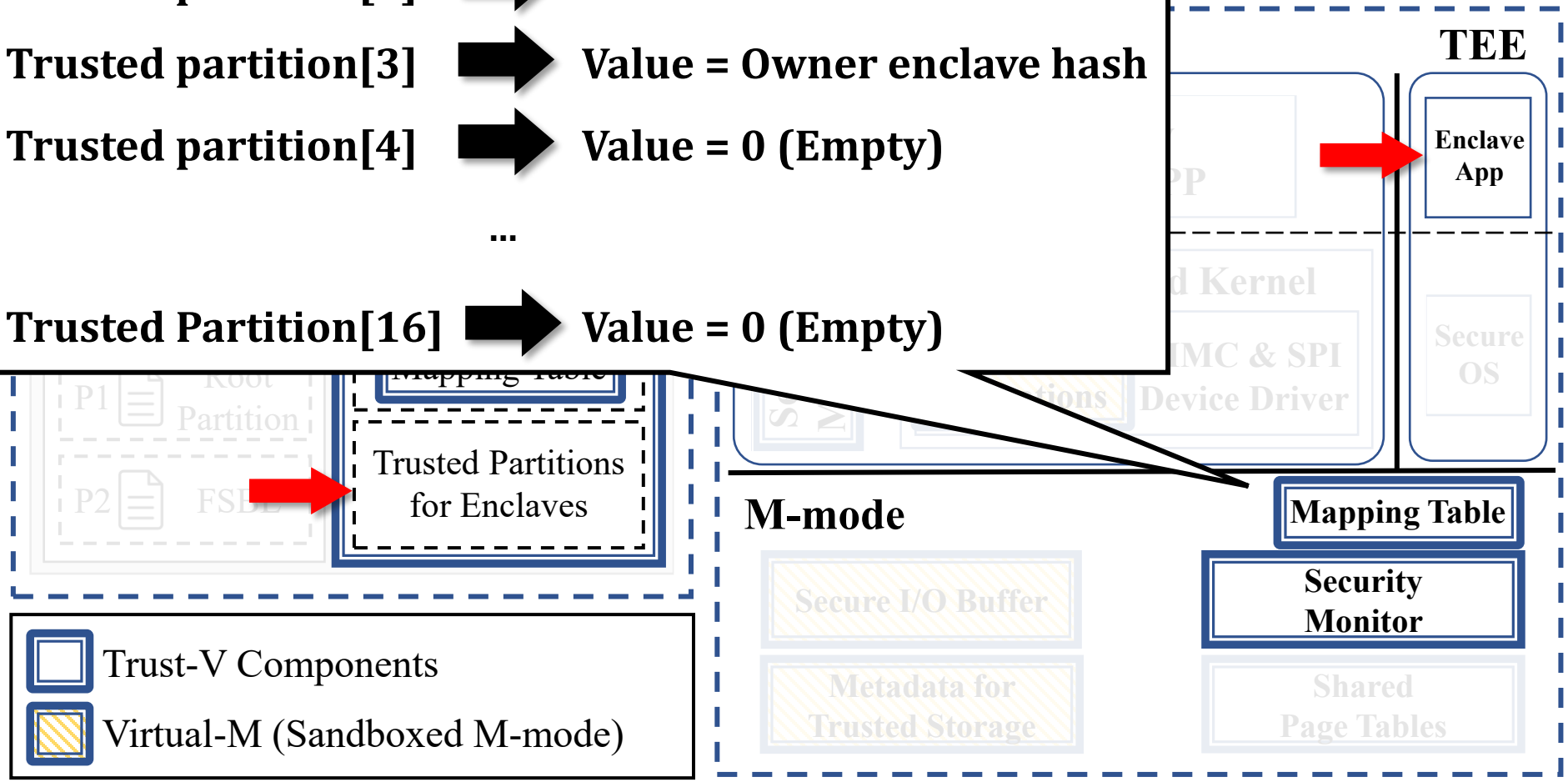
Trusted partition[2] ➡ Value = Owner enclave hash

Trusted partition[3] ➡ Value = Owner enclave hash

Trusted partition[4] ➡ Value = 0 (Empty)

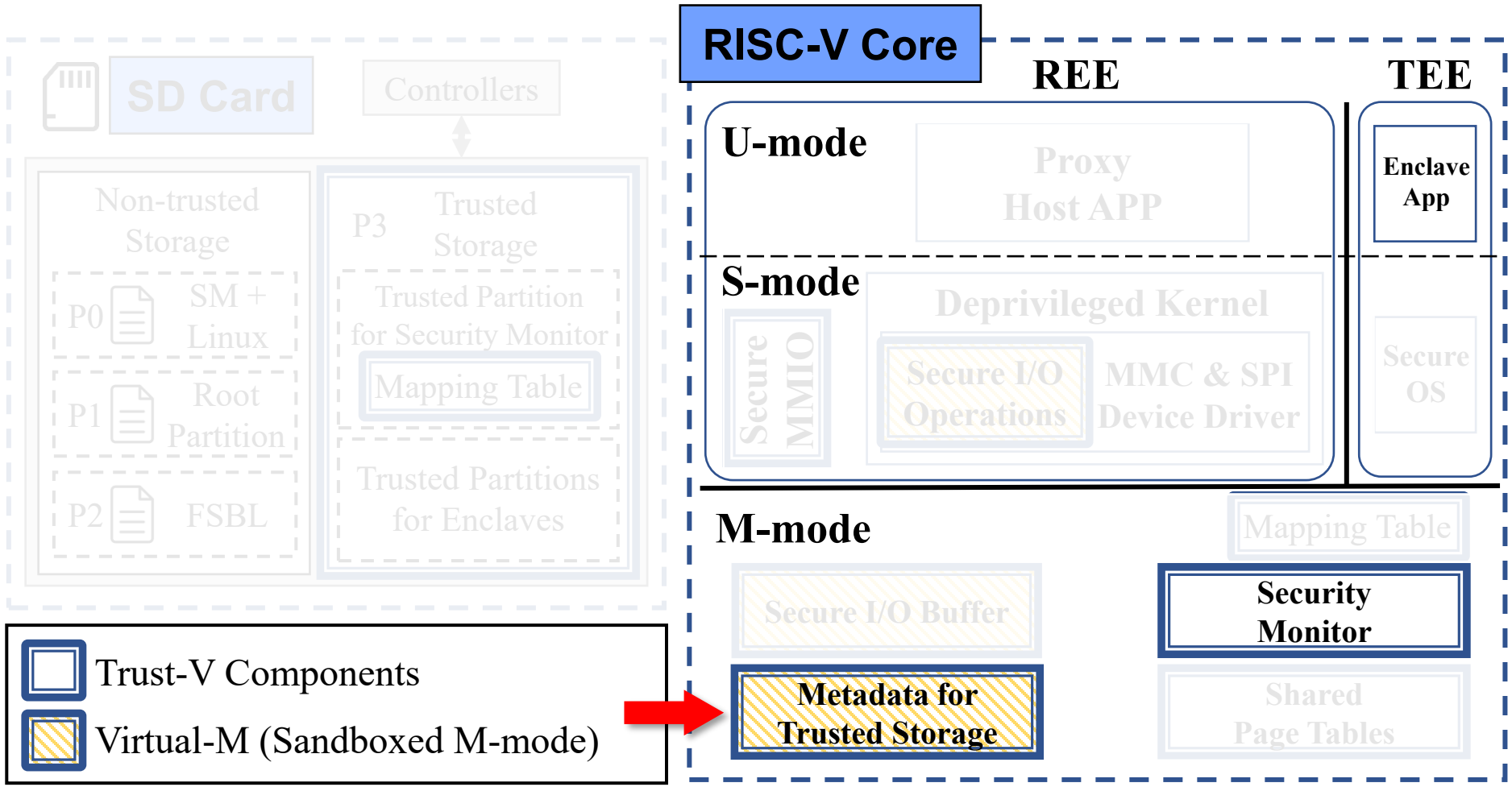
...

Trusted Partition[16] ➡ Value = 0 (Empty)



(P4) I/O Context Maintenance

The security monitor maintains metadata for Secure I/O operations



(P4) I/O Context Maintenance

The ***tions***

Trusted Storage information

→ starting sector

→ size

…

Secure I/O context

→ I/O type (read or write)

→ # of requested blocks

→ trusted partition number owned by the requester

…

The metadata for Trusted Storage is updated during boot



(P4) I/O Context Maintenance

The *tions*

Trusted Storage i

→ starting sector

→ size

…

Secure I/O context

→ I/O type (read or write)

→ # of requested blocks

→ trusted partition number owned by the requester

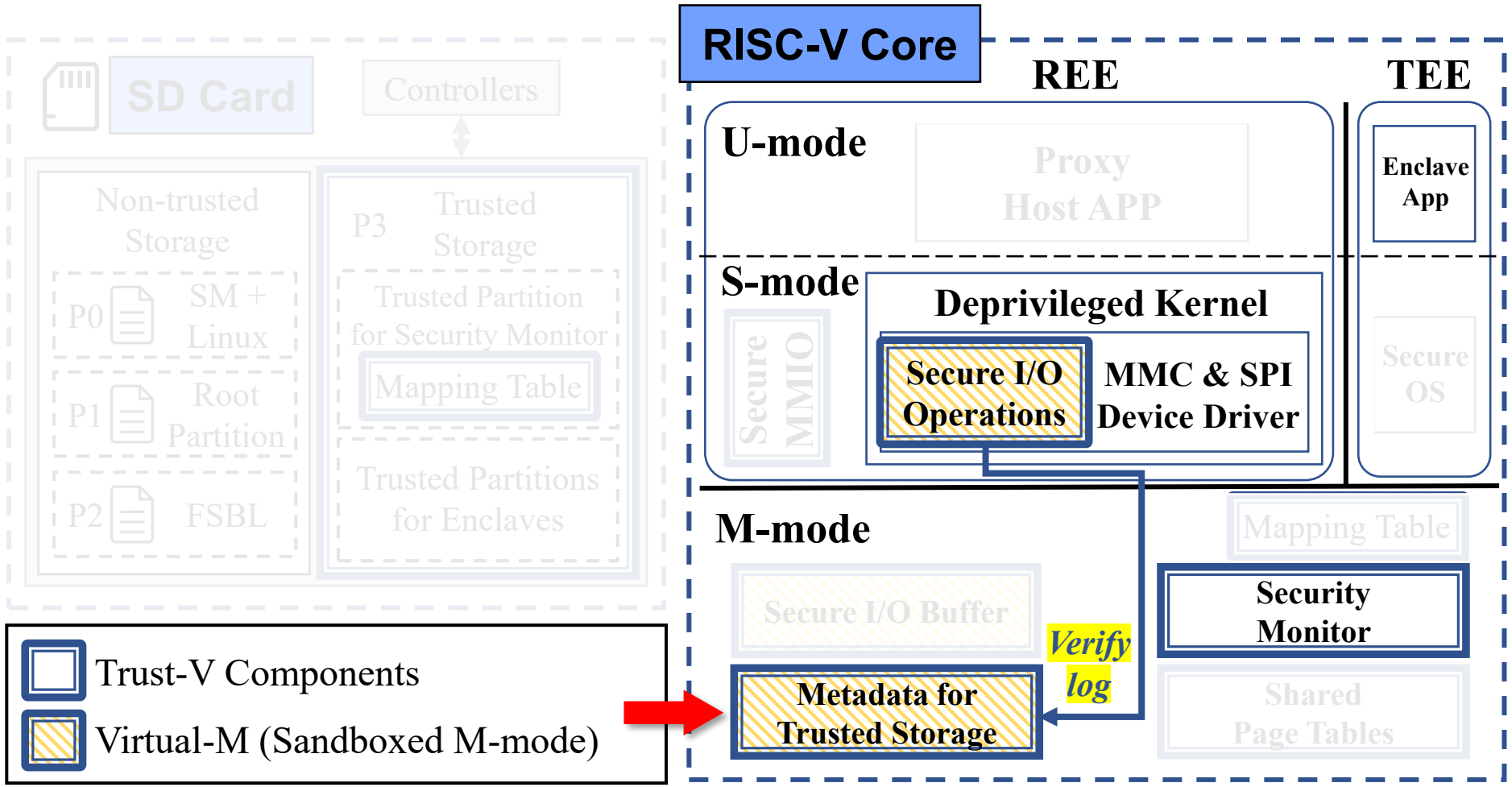
…

The Secure I/O context is updated only after the security monitor verifies the enclave's request

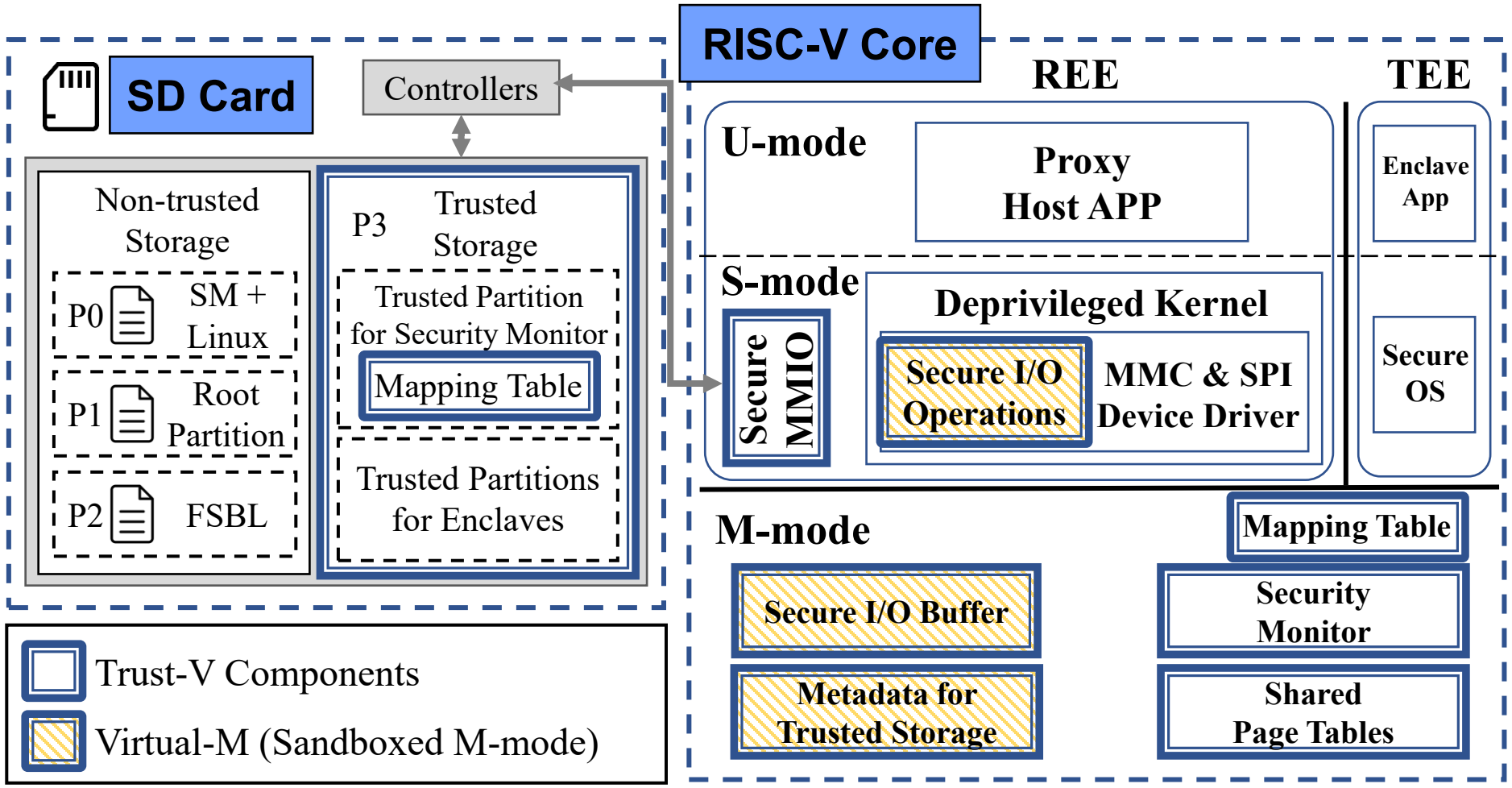


(P4) I/O Context Maintenance

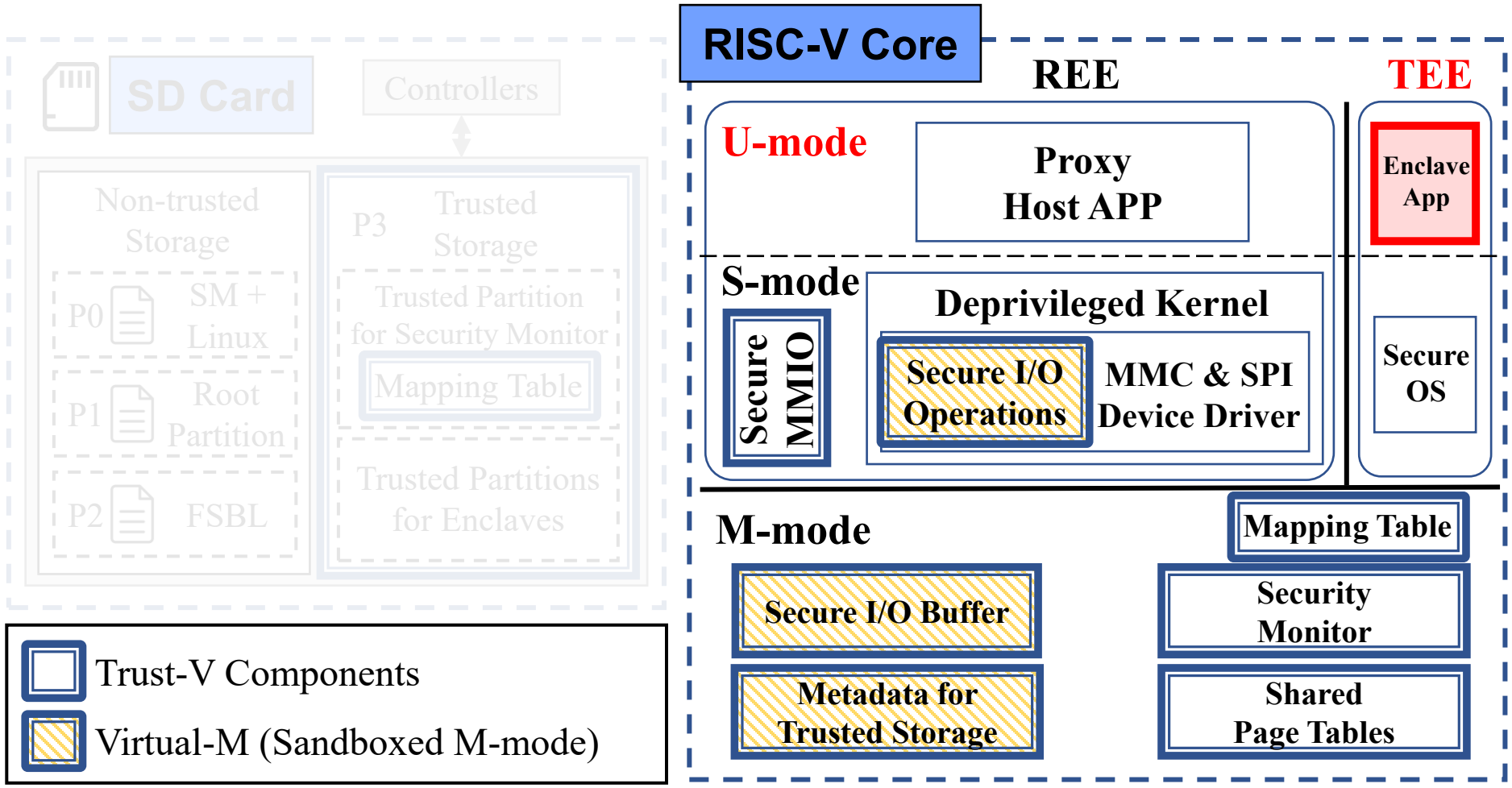
The security monitor maintains metadata for Secure I/O operations



Secure I/O Lifecycle

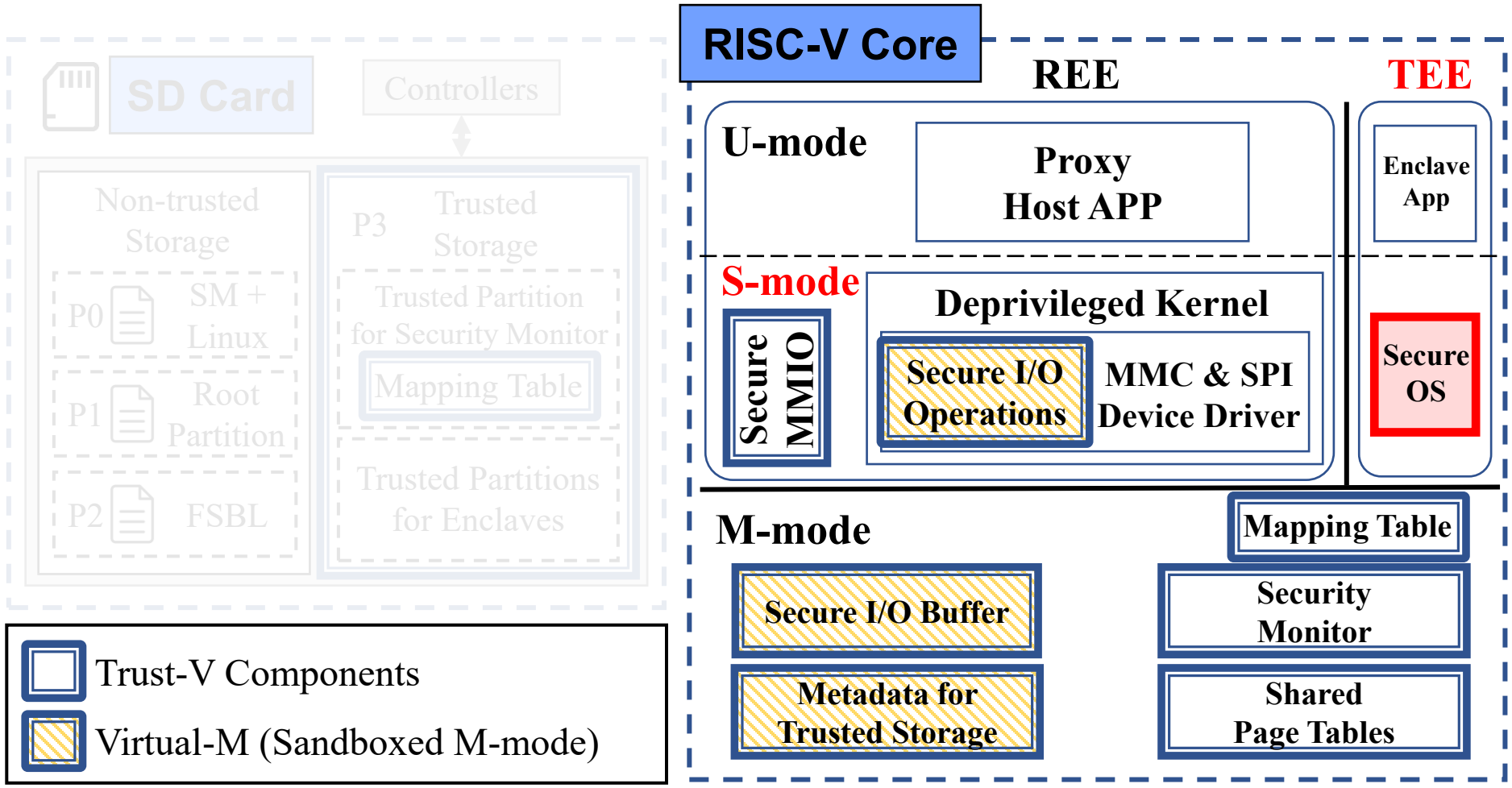


Secure I/O Lifecycle



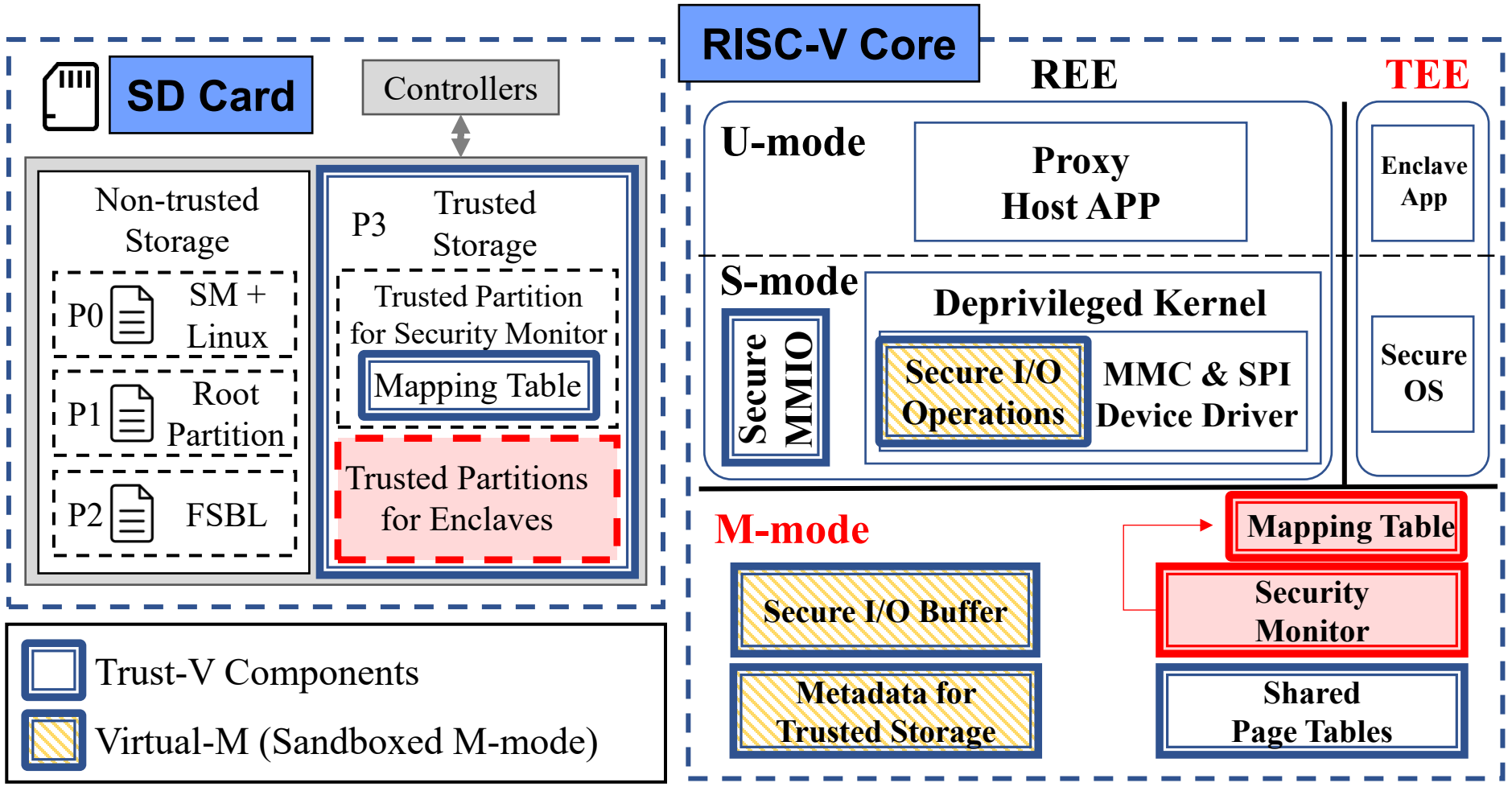
The enclave sends request (block operation type and number of requested blocks) to initiate Secure I/O

Secure I/O Lifecycle



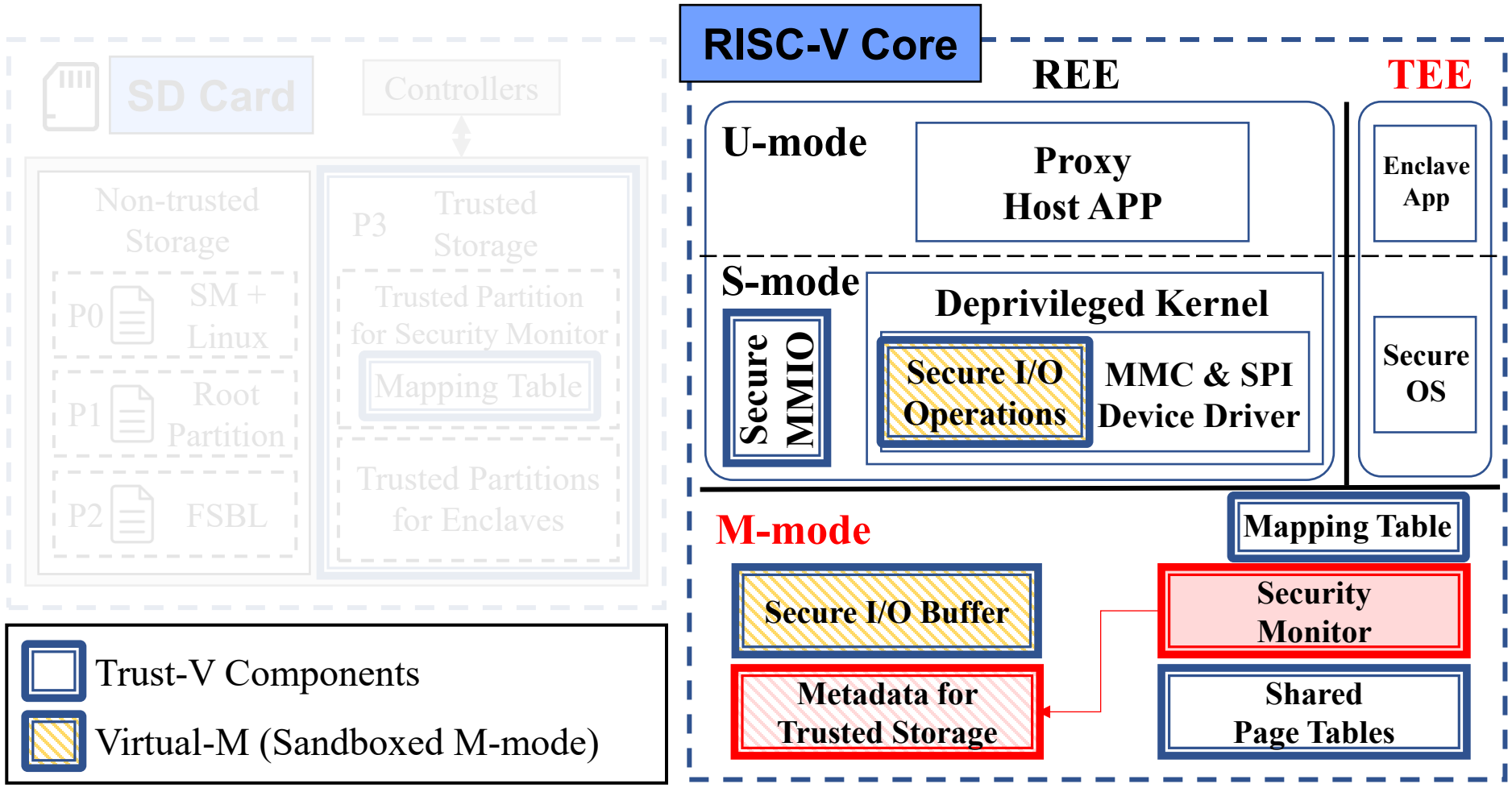
The secure OS forwards the request to the security monitor

Secure I/O Lifecycle



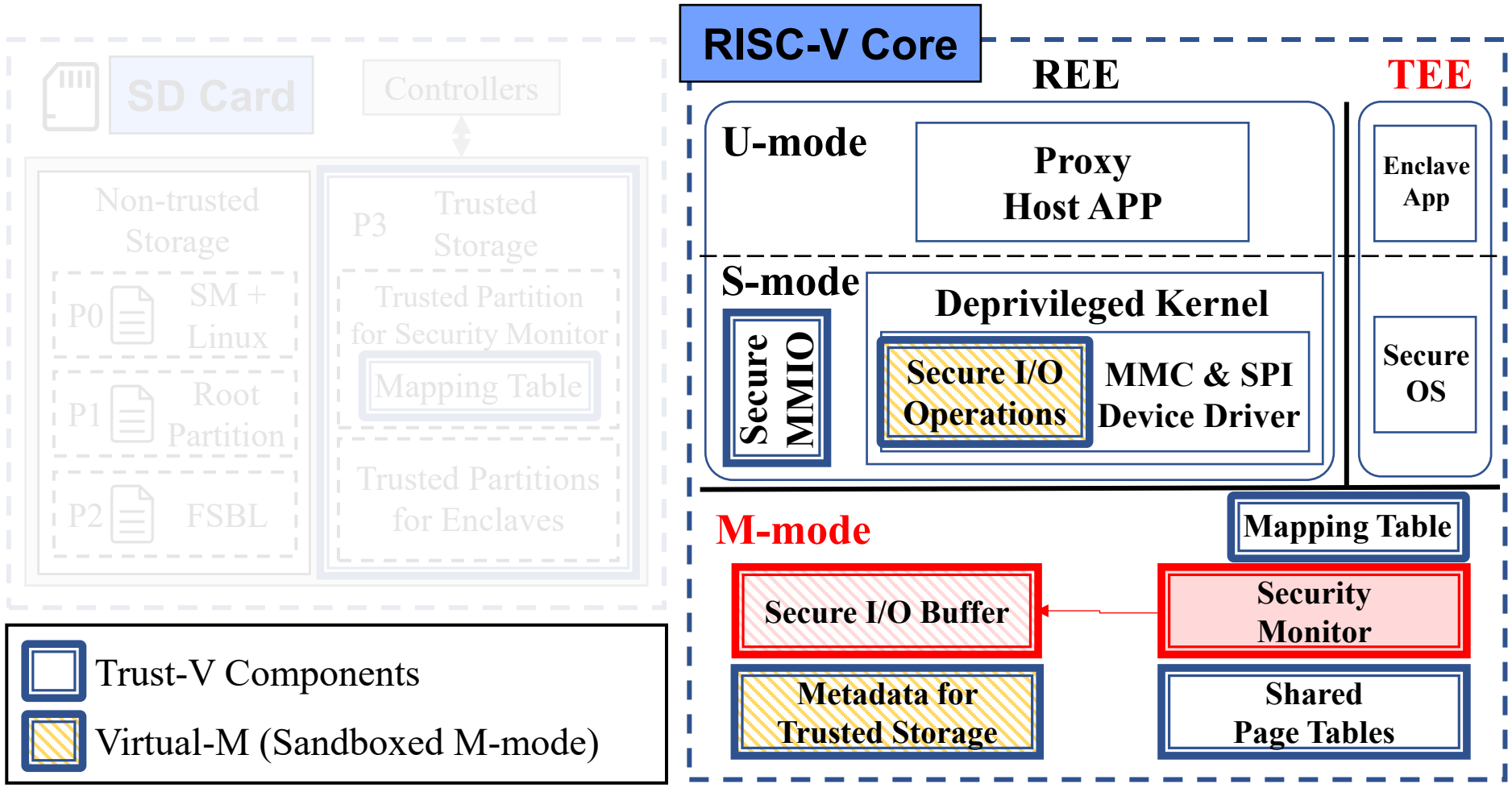
The security monitor consults the mapping table to locate the Trusted Partition owned by the enclave

Secure I/O Lifecycle



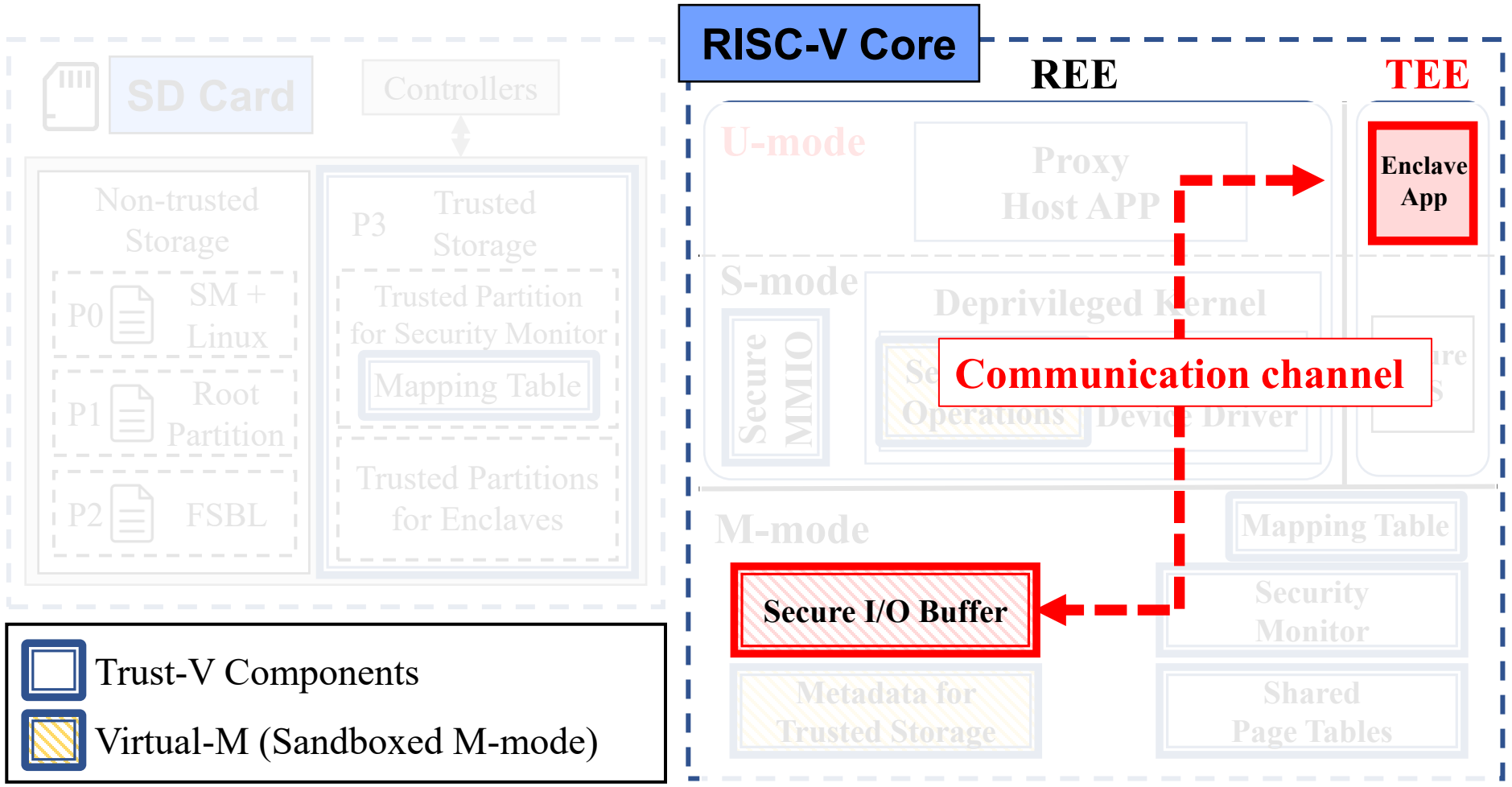
The security monitor stores the Secure I/O context as metadata

Secure I/O Lifecycle



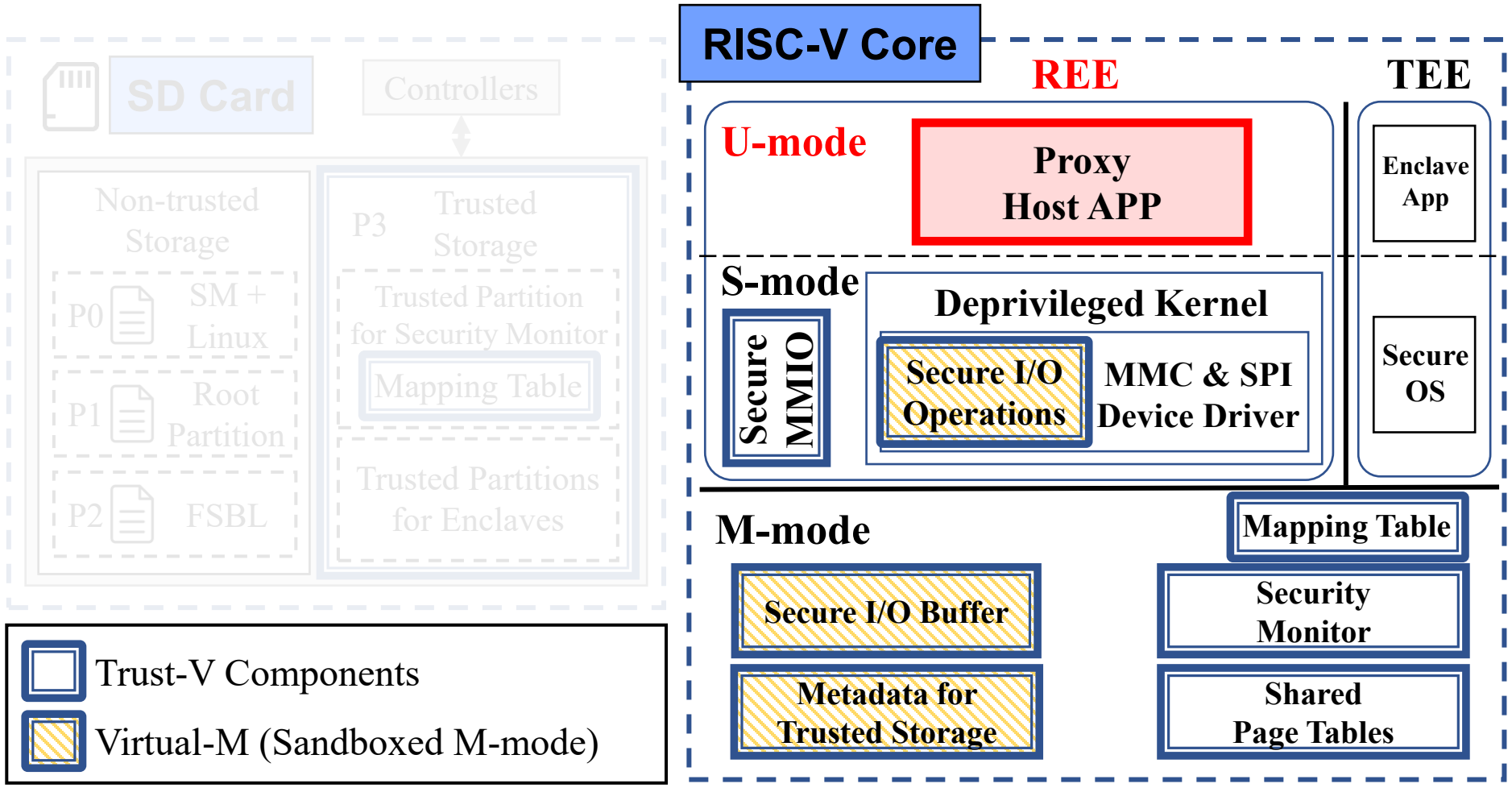
The security monitor maps the Secure I/O buffer into the enclave's address space

Secure I/O Lifecycle



The enclave can read from or write to the I/O buffer

Secure I/O Lifecycle

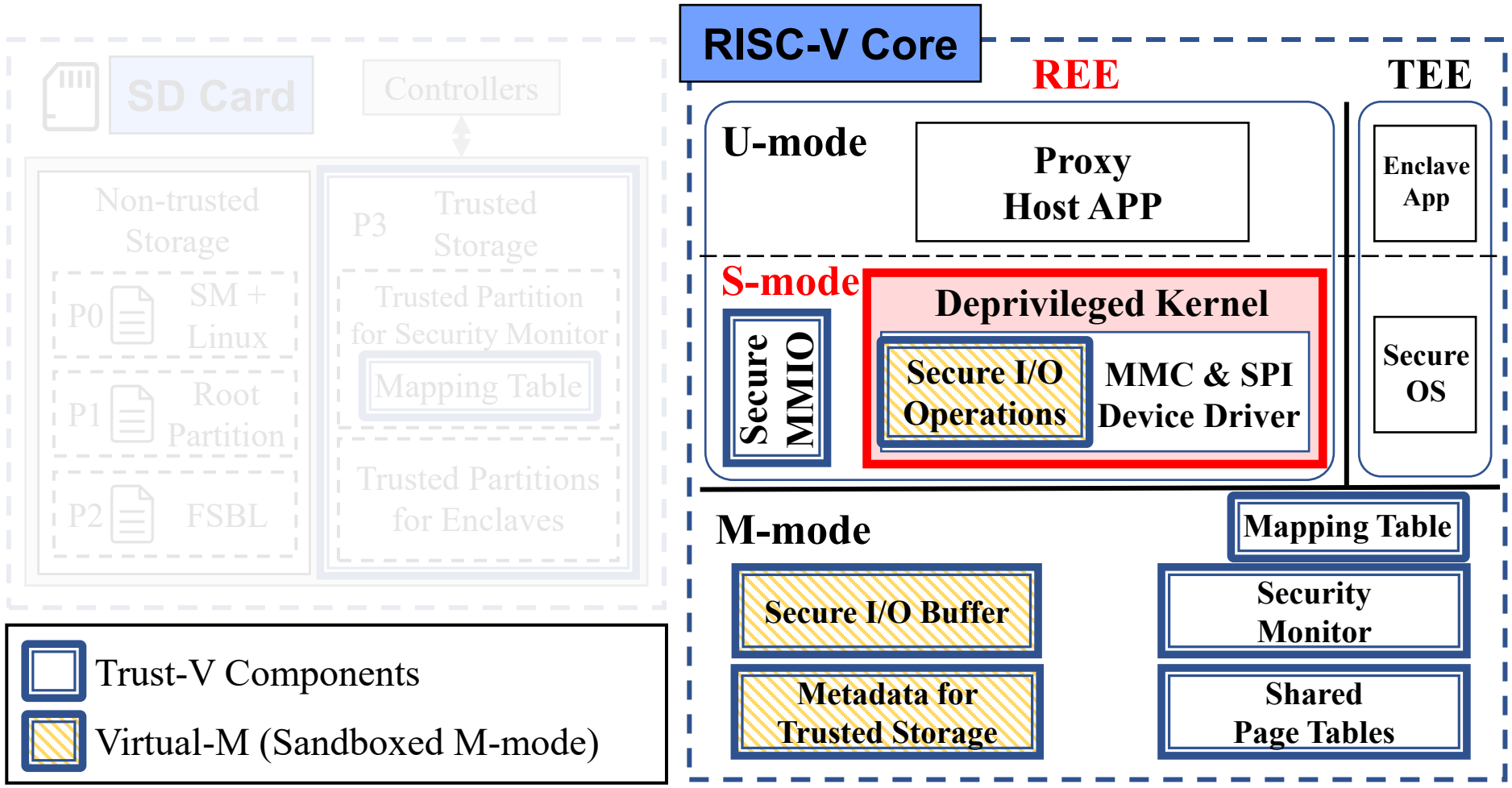


Trust-V reuse the I/O logic of the kernel device driver; hence, the enclave transfers control to the REE to outsource Secure I/O

CNU

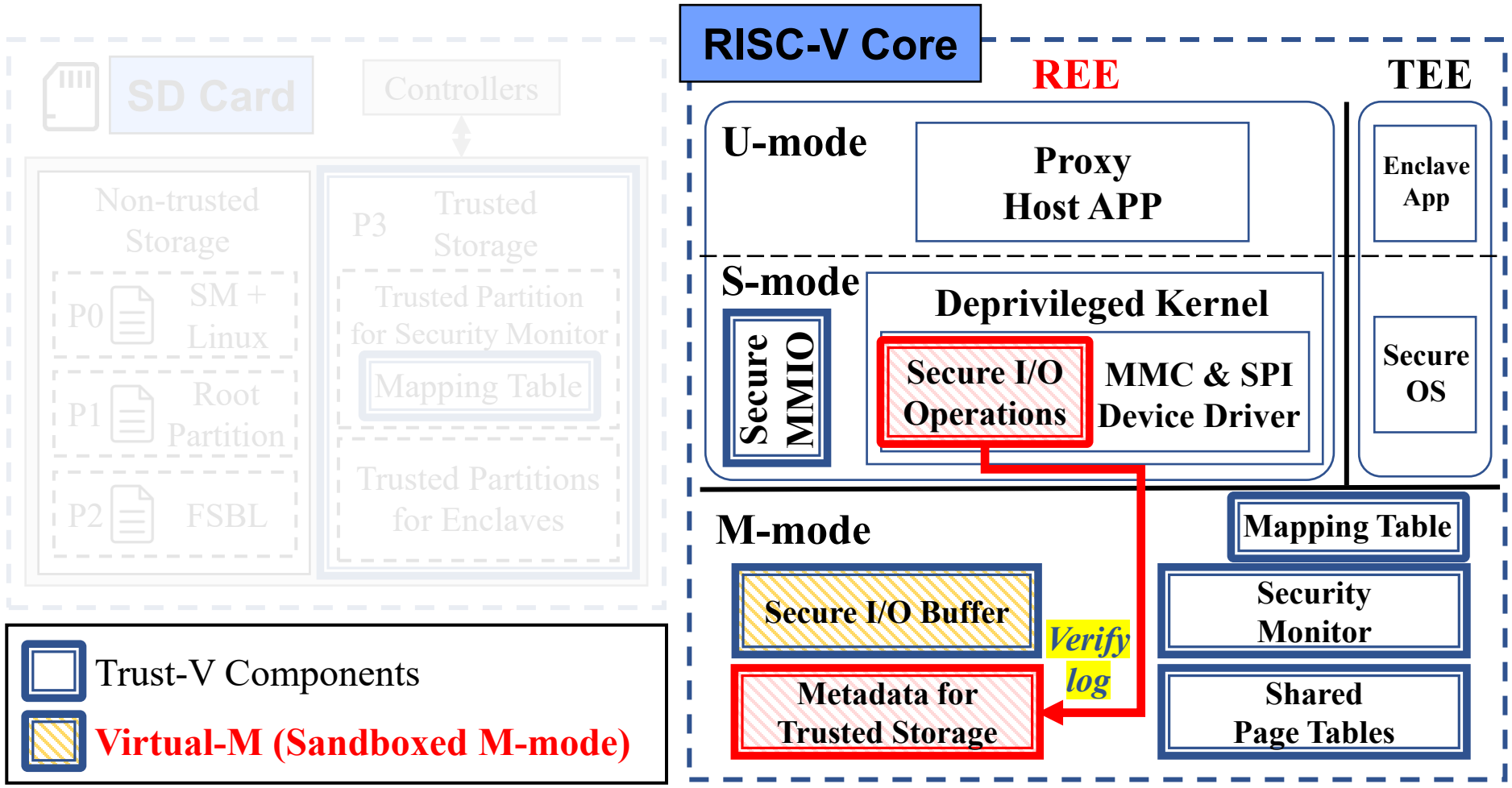
SECRET

Secure I/O Lifecycle



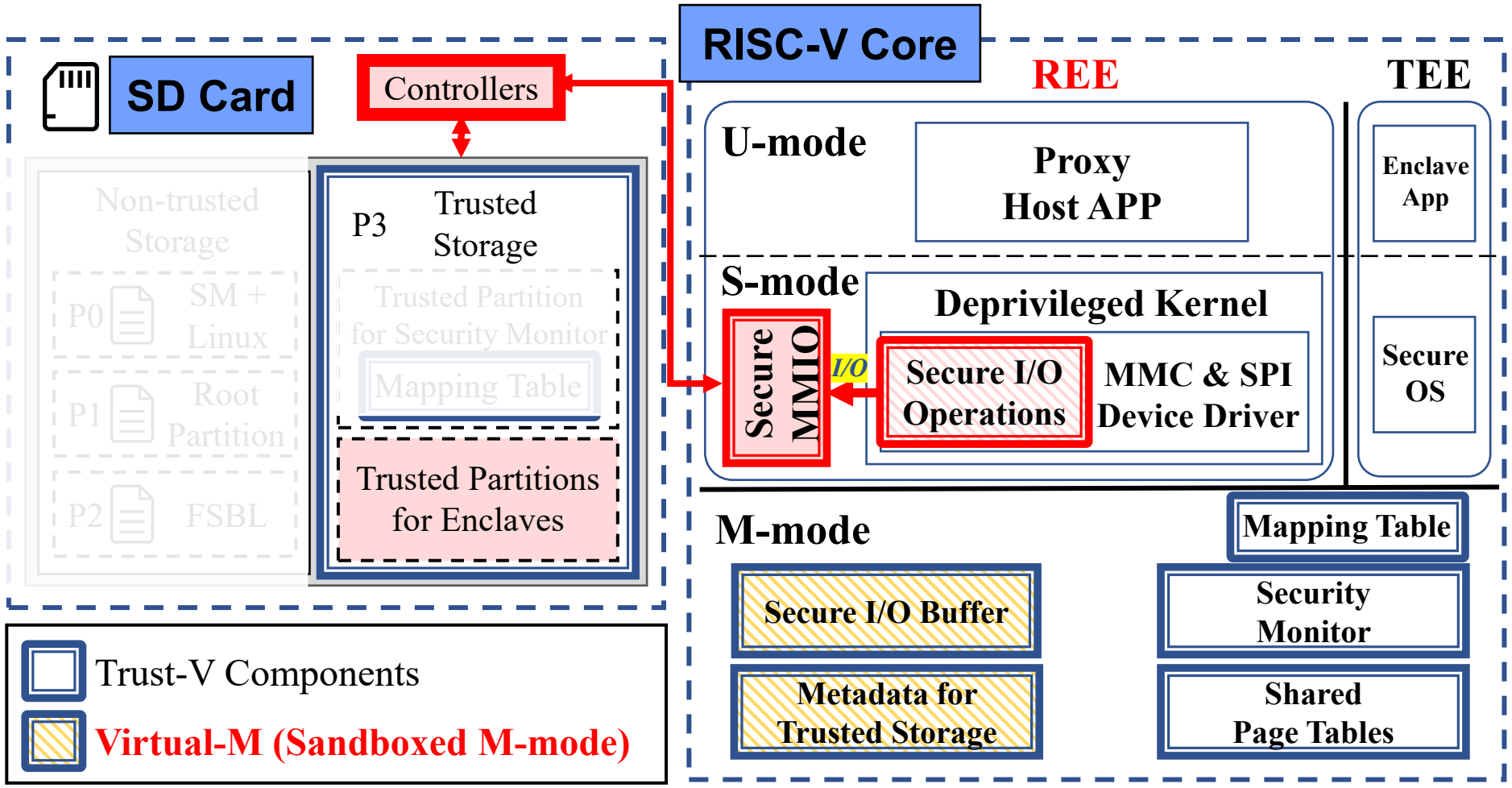
Control flow proceeds to the MMC and SPI device drivers

Secure I/O Lifecycle



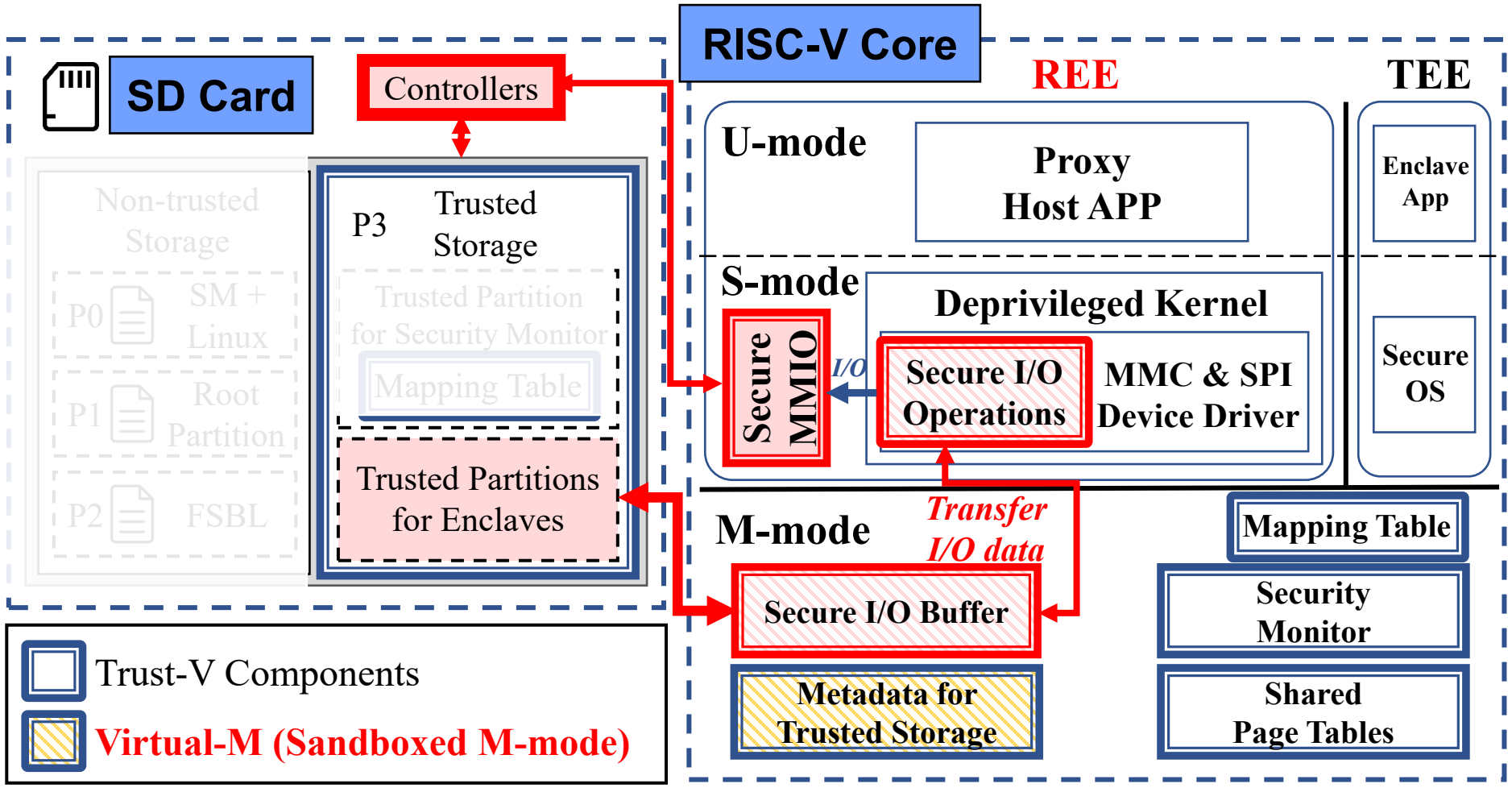
The privileged code verifies the enclave's Secure I/O request using the metadata.

Secure I/O Lifecycle



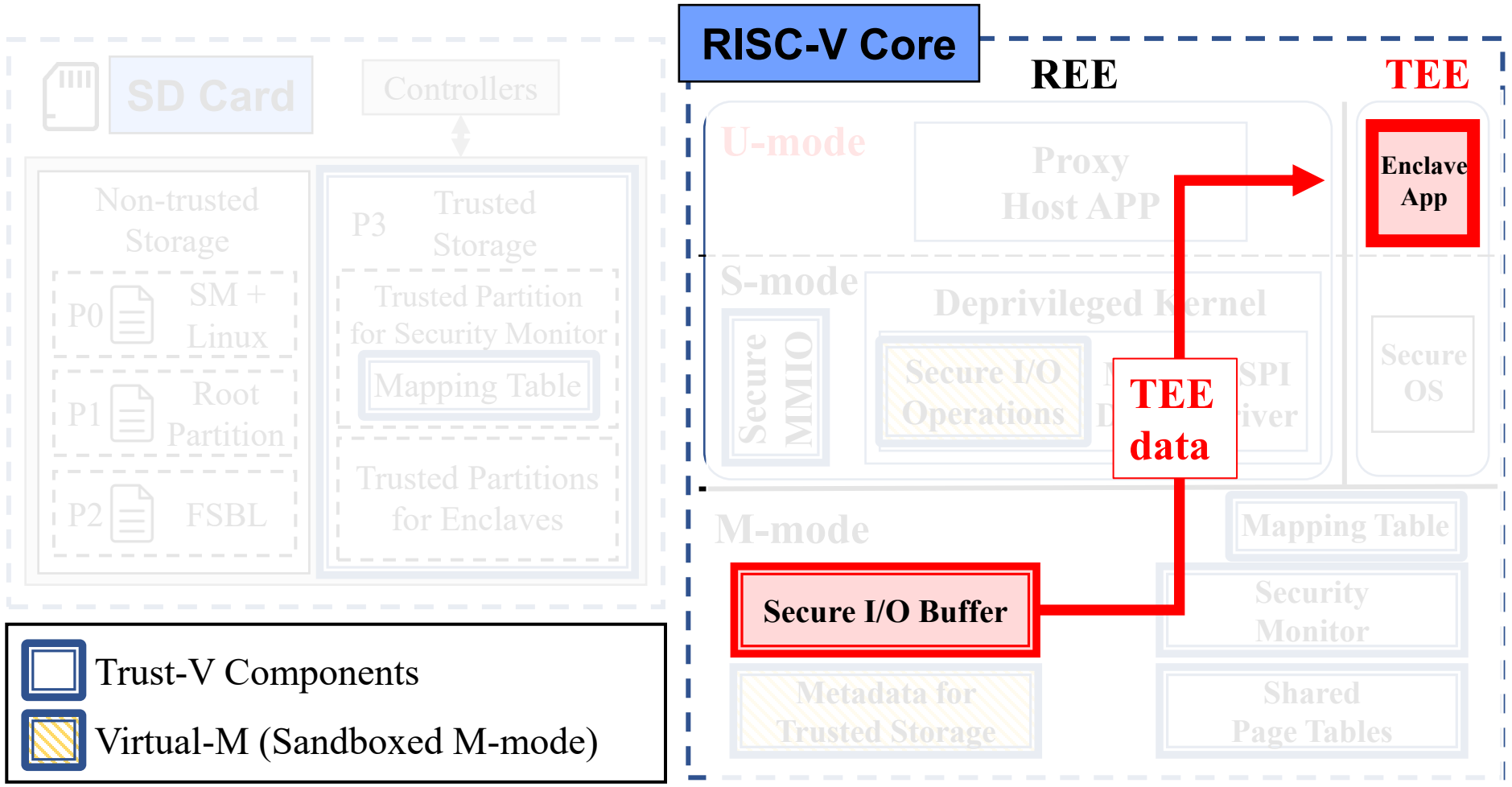
The privileged code sends the protocol message for the requested I/O operation.

Secure I/O Lifecycle



The privileged code transfers I/O data to the secure I/O buffer.

Secure I/O Lifecycle



The enclave consumes the TEE data from the Secure I/O buffer

Development Support: APIs

```
static void sifive_spi_write(struct
sifive_spi *spi, int offset, u32 value)
{
 iowrite32(value, spi->regs + offset);
}
```

Development Support

```
static void sifive_spi_write(struct
sifive_spi *spi, int offset, u32 value)
{
 iowrite32(value, spi->regs + offset);
}
```



Instrument

```
void priv_ops_mmio_write32(u32 * value, virt_addr_t mmio_addr){
   enterVirtualM();
   if(verify_memops(mmio_addr , value))
   asm volatile ("sw %0, 0(%1)“ : : "r"(*value), "r"(mmio_addr));
   exitVirtualM();
}

static void sifive_spi_write (struct sifive_spi * spi, int offset , u32 *
value){
   if ((*METADATA_SHIELD_FLAG == true) && \
   (spi -> regs == * METADATA_LOCKED_MMIO_ADDR)){
      // Privileged I/O operations for locked MMIO
      priv_ops_mmio_write32(value, spi -> regs + offset);
   } else { // Generic I/O operations
      iowrite32 (*value, spi -> regs + offset);
   }
}
```

Development Support

Developer-specified code wrapped by enter/exitVirtualM() executes in VirtualM mode

```
static void sifive_spi_write(struct
sifive_spi *spi, int offset, u32 value)
{
 iowrite32(value, spi->regs + offset);
}
```

Instrument

```
void priv_ops_mmio_write32(u32 * value, virt_addr_t mmio_addr){
   enterVirtualM();
   if(verify_memops(mmio_addr , value))
   asm volatile ("sw %0, 0(%1)“ : : "r"(*value), "r"(mmio_addr));
   exitVirtualM();
}

static void sifive_spi_write (struct sifive_spi * spi, int offset , u32 *
value){
   if ((*METADATA_SHIELD_FLAG == true) && \
   (spi -> regs == * METADATA_LOCKED_MMIO_ADDR)){
      // Privileged I/O operations for locked MMIO
      priv_ops_mmio_write32(value, spi -> regs + offset);
   } else { // Generic I/O operations
      iowrite32 (*value, spi -> regs + offset);
   }
}
```

Security Analysis

- Attack surfaces and defense mechanisms

Attack surface	Defense mechanism
Issue malicious block I/O requests targeting Trusted Storage	Metadata is tamper-proof and is written only by the security monitor upon enclave requests
Race condition against secure I/O	Ensures that only one secure I/O operation is performed at a time
Abusing enterVirtualM() from a malicious loadable kernel module (LKM)	Verify that enterVirtualM() originates from a statically compiled kernel module
If vulnerabilities exist in Virtual-M code, they could be exploited	Minimal offloading to Virtual-M mode simplifies formal verification of privileged code (66 lines)

Security Analysis

- Attack surfaces and defense mechanisms

Attack surface	Defense mechanism
Issue malicious block I/O requests targeting Trusted Storage	Metadata is tamper-proof and is written only by the security monitor upon enclave requests
Race condition against secure I/O	Ensures that only one secure I/O operation is performed at a time
Abusing enterVirtualM() from a malicious loadable kernel module (LKM)	Verify that enterVirtualM() originates from a statically compiled kernel module
If vulnerabilities exist in Virtual-M code, they could be exploited	Minimal offloading to Virtual-M mode simplifies formal verification of privileged code (66 lines)

Security Analysis

- Attack surfaces and defense mechanisms

Attack surface	Defense mechanism
Issue malicious block I/O requests targeting Trusted Storage	Metadata is tamper-proof and is written only by the security monitor upon enclave requests
Race condition against secure I/O	Ensures that only one secure I/O operation is performed at a time
Abusing enterVirtualM() from a malicious loadable kernel module (LKM)	Verify that enterVirtualM() originates from a statically compiled kernel module
If vulnerabilities exist in Virtual-M code, they could be exploited	Minimal offloading to Virtual-M mode simplifies formal verification of privileged code (66 lines)

Security Analysis

- Attack surfaces and defense mechanisms

Attack surface	Defense mechanism
Issue malicious block I/O requests targeting Trusted Storage	Metadata is tamper-proof and is written only by the security monitor upon enclave requests
Race condition against secure I/O	Ensures that only one secure I/O operation is performed at a time
Abusing enterVirtualM() from a malicious loadable kernel module (LKM)	Verify that enterVirtualM() originates from a statically compiled kernel module
If vulnerabilities exist in Virtual-M code, they could be exploited	Minimal offloading to Virtual-M mode simplifies formal verification of privileged code (66 lines)

Security Analysis

- Attack surfaces and defense mechanisms

Attack surface	Defense mechanism
Issue malicious block I/O requests targeting Trusted Storage	Metadata is tamper-proof and is written only by the security monitor upon enclave requests
Race condition against secure I/O	Ensures that only one secure I/O operation is performed at a time
Abusing enterVirtualM() from a malicious loadable kernel module (LKM)	Verify that enterVirtualM() originates from a statically compiled kernel module
If vulnerabilities exist in Virtual-M code, they could be exploited	Minimal offloading to Virtual-M mode simplifies formal verification of privileged code (66 lines)

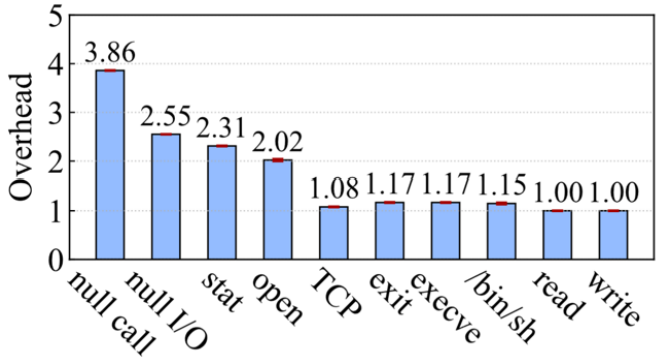
Implementation

SoC	 **HiFive Unleashed (Freedom U540-C000)**
Storage Device	 **Samsung 64 GB Evo Plus (MB-MC64KA)**
REE Components	 **Linux v5.10.186**
TEE Components	 **Keystone v1.0.0**
Enclaves (Secure I/O Apps)	**(1) storing and loading persistent enclave** **(2) a key-value store enclave** **(3) provisioning the device root key (DRK) in the trusted partition for the security monitor**

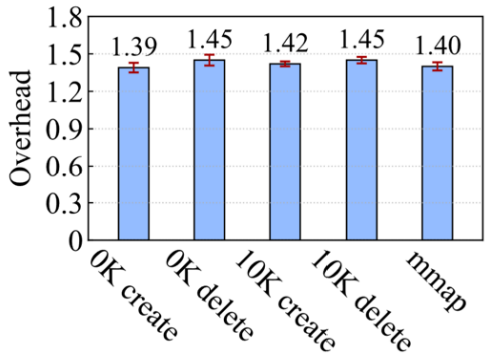
Figures :
(1) https://kr.rs-online.com/web/p/sd-cards/2355212,
(2) https://iconscout.com/free-icon/free-linux-logo-icon_2944967
(3) https://keystone-enclave.org/
(4) https://www.crowdsupply.com/sifive/hifive-unleashed

Performance Evaluation

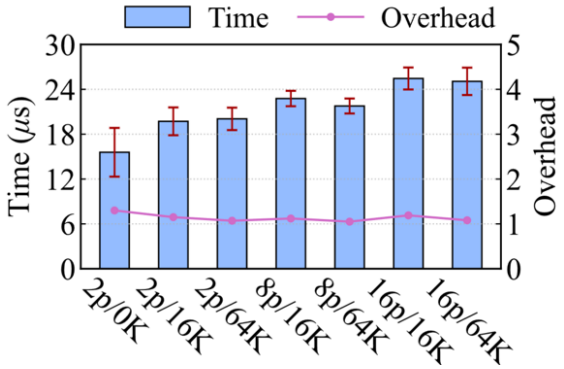
OS bench results in the REE– LMBench and CoreMark-PRO



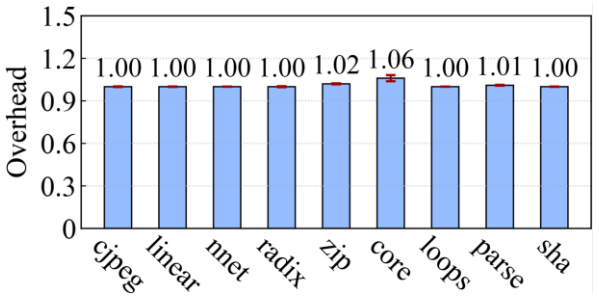
Maximum overhead of 3.86x in LMBench primary operations



39%~45% overhead in file operations



5%~30% overhead in context switching



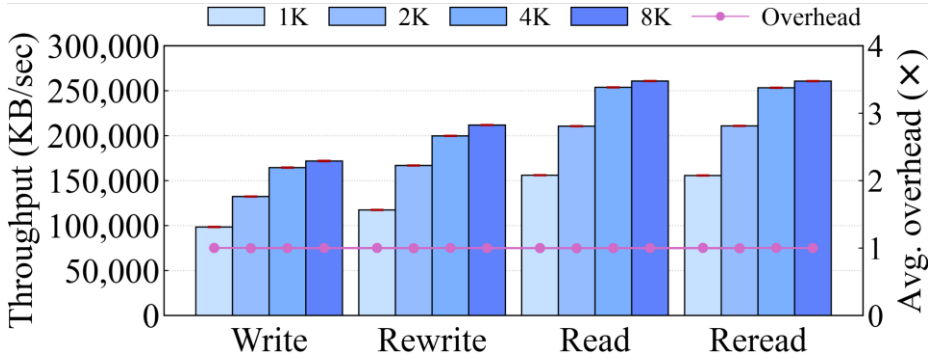
CoarMark-PRO overhead is negligible (1~6%)

CNU

SECRET

Performance Evaluation

I/O bench results in the REE– IOzone and FIO with a 100MB file



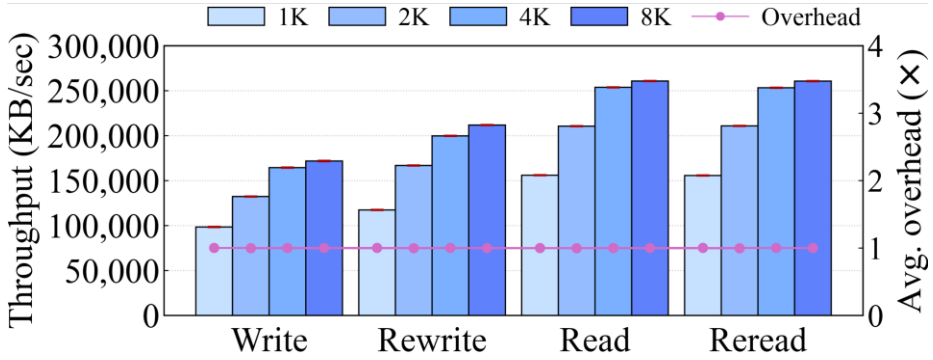
<IOzone throughput with Trust-V (in KB/sec)>

Operation Type	Block IO Size	Baseline	Trust-V (Overhead)
Random Read/Write	**512B**	18.0	16.8 (6.6%)
	1K	32.0	31.5 (1.6%)
	2K	50.0	50.5 (1.2%)
	4K	73.2	73.2 (0.8%)
	8K	91.3	90.6 (0.8%)

<Fio throughput comparison (in MiB/sec)>

Performance Evaluation

I/O bench results in the REE– IOzone and FIO with a 100MB file



<IOzone throughput with Trust-V (in KB/sec)>

Operation Type	Block IO Size	Baseline	Trust-V (Overhead)
Random Read/Write	512B	18.0	16.8 (6.6%)
	1K	32.0	31.5 (1.6%)
	2K	50.0	50.5 (1.2%)
	4K	73.2	73.2 (0.8%)
	8K	91.3	90.6 (0.8%)

<Fio throughput comparison (in MiB/sec)>

Performance evaluation of example enclaves using Trusted Storage

Ops. type	512B	1K	2K	4K
Read	0.14	0.14	0.28	0.55
Write	0.15	0.15	0.29	0.58

<Enclave's block I/O operations (in seconds)>

4 KB I/O to Trusted Storage is 0.3–0.6% faster than non-trusted I/O with data (un)sealing using AES-256 on ext4.

CNU

SECRET

Conclusion

- Define a new problem in TEEs: the availability of general secure storage
- Propose Trusted Storage, which enhances the availability of TEE data on storage
- Implement a PoC on the HiFive Unleashed board (RISC-V) without relying on hardware-specific features (e.g., PMP and virtualization)