Computer Architecture Lab.

KAIST

ShieldStore: Shielded In-memory Key-value Storage with SGX

Taehoon Kim, Joongun Park, Jaewook Woo,
Seungheun Jeon, and Jaehyuk Huh

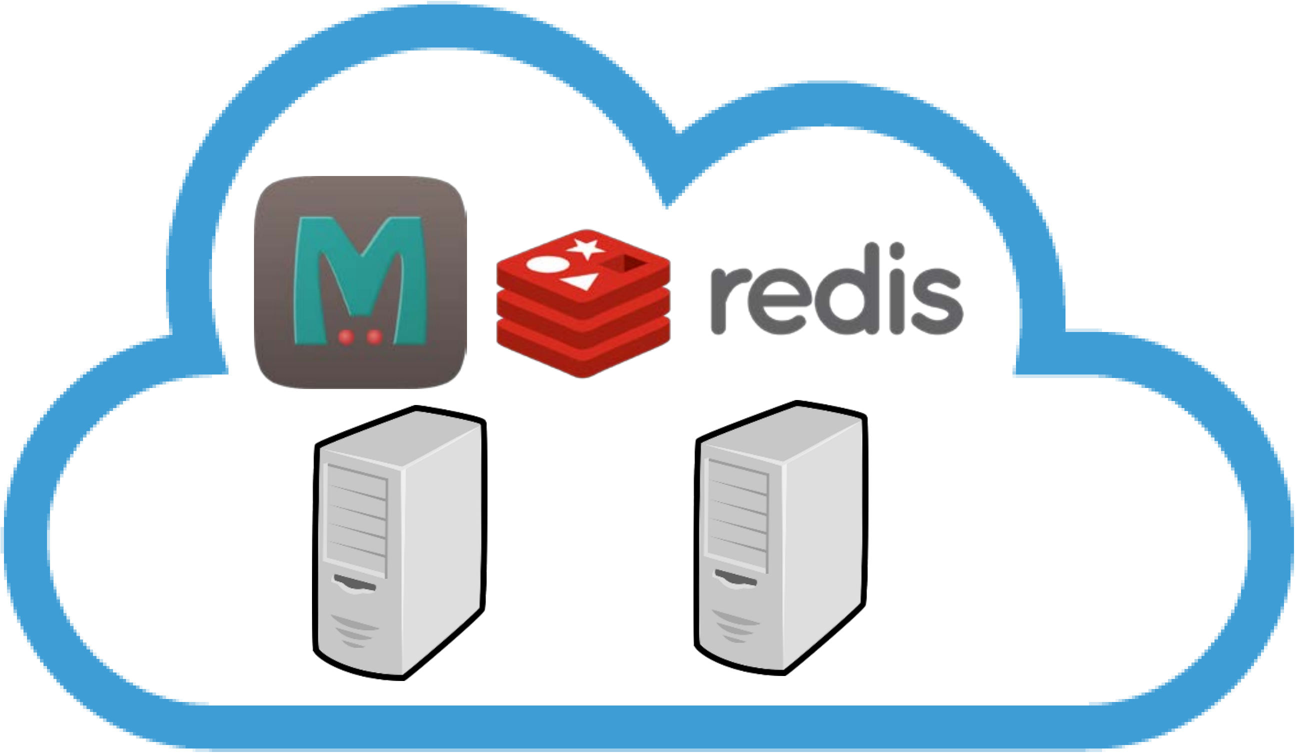
Dresden, Germany
March 25-28, 2019

<EURO/SYS'19>

KAIST School of Computing

Trusted Key-value Stores

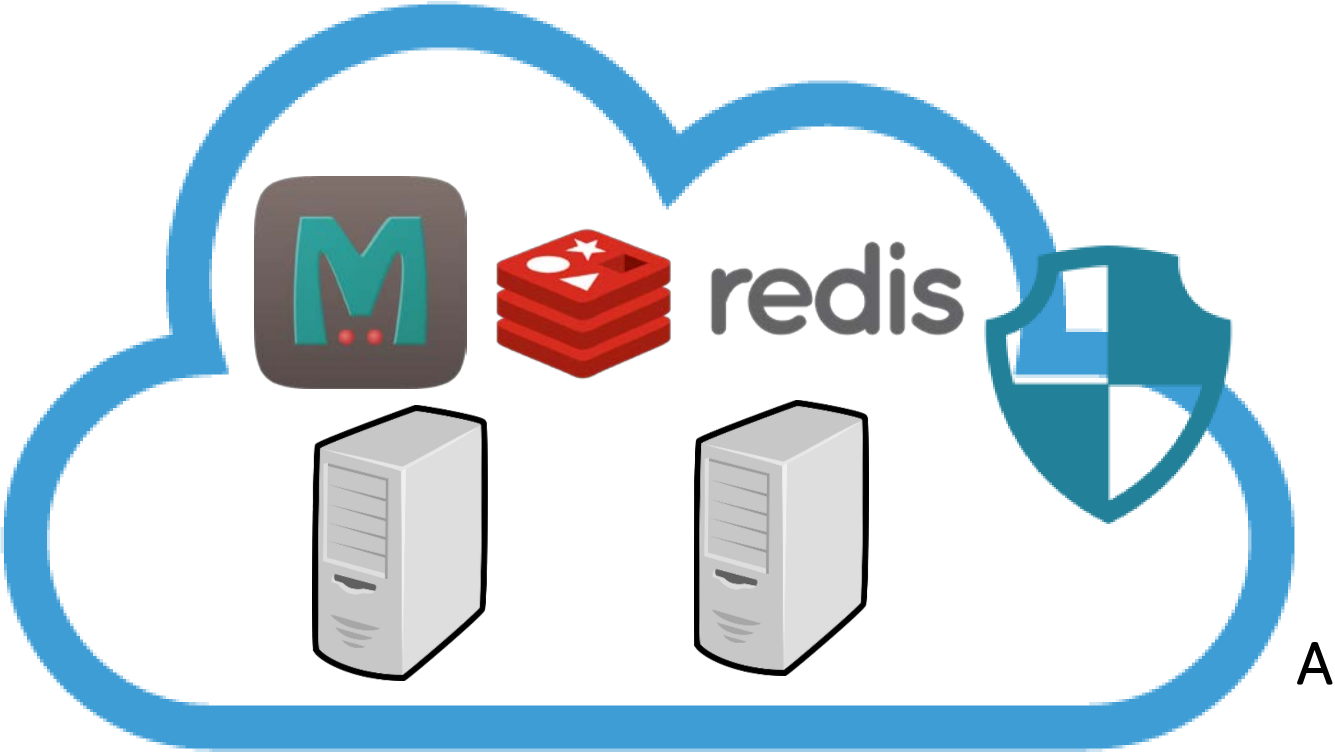
- User data is exposed to malicious attackers in clouds
- Hardware-based security supports
  - Provide *trusted execution environment* for remote server



Trusted Key-value Stores

- User data is exposed to malicious attackers in clouds

- Hardware-based security supports
  - Provide *trusted execution environment* for remote server

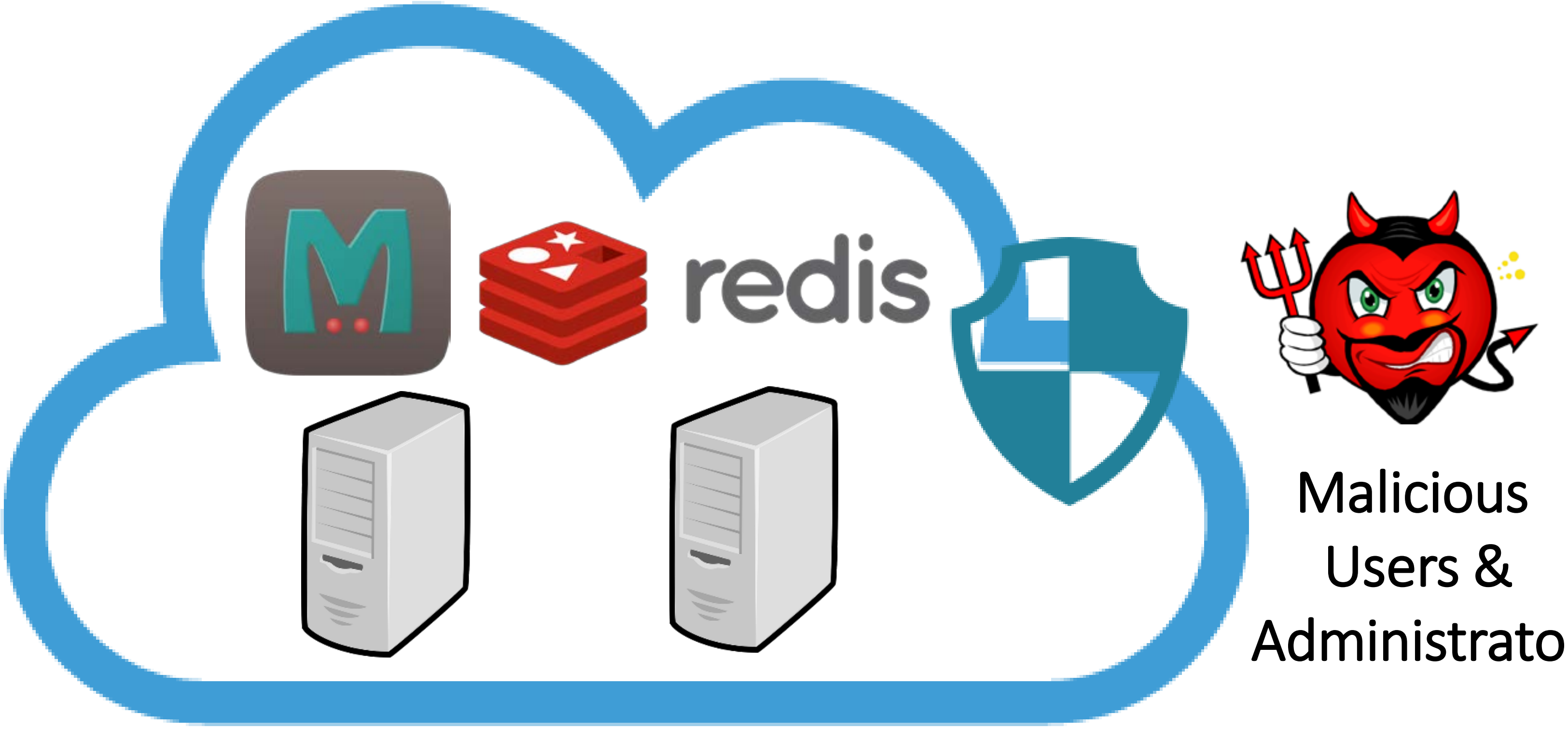


Malicious Users & Administrator

Trusted Key-value Stores

- User data is exposed to malicious attackers in clouds

- Hardware-based security supports
  - Provide *trusted execution environment* for remote server

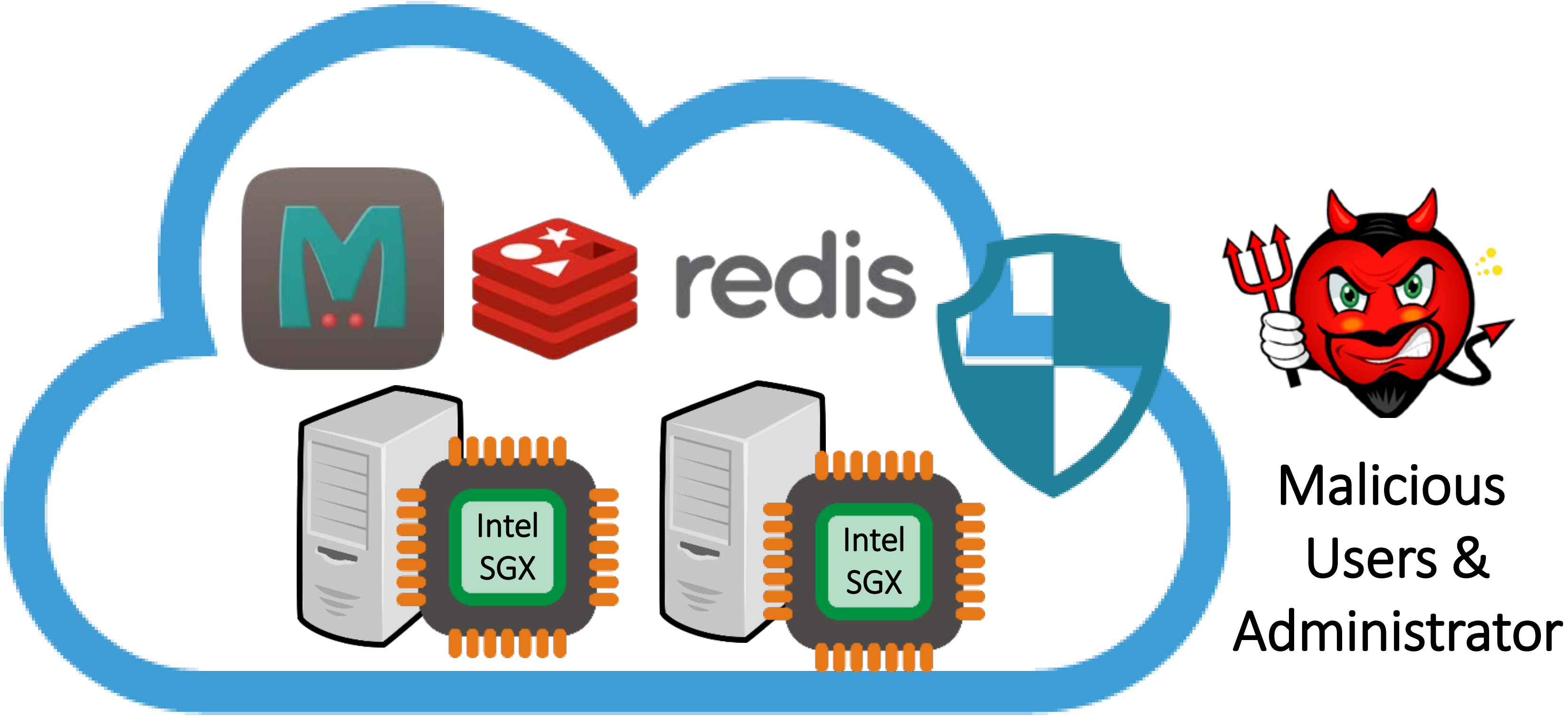


Malicious Users & Administrator

Trusted Key-value Stores

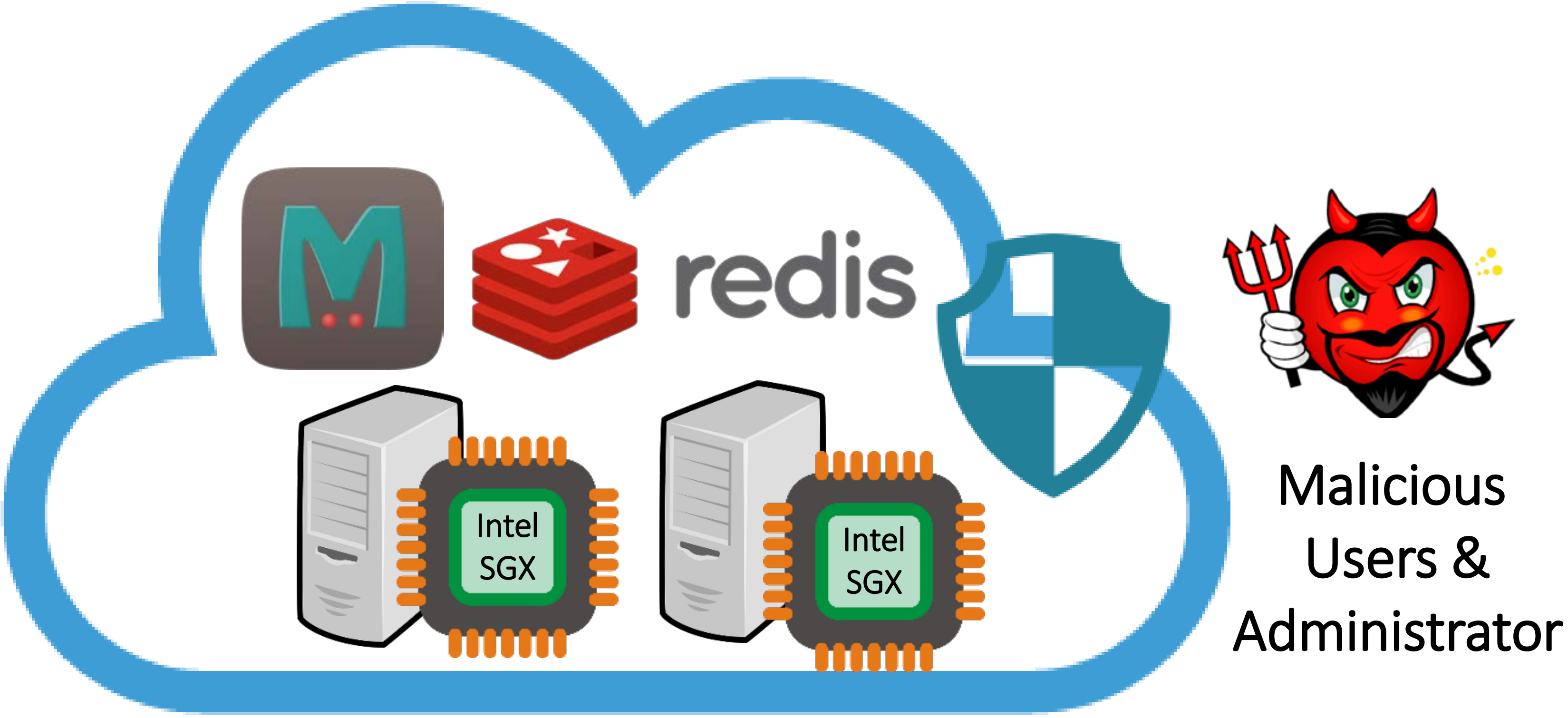
- User data is exposed to malicious attackers in clouds

- Hardware-based security supports
  - Provide *trusted execution environment* for remote server



Trusted Key-value Stores

- User data is exposed to malicious attackers in clouds
- Hardware-based security supports
  - Provide *trusted execution environment* for remote server

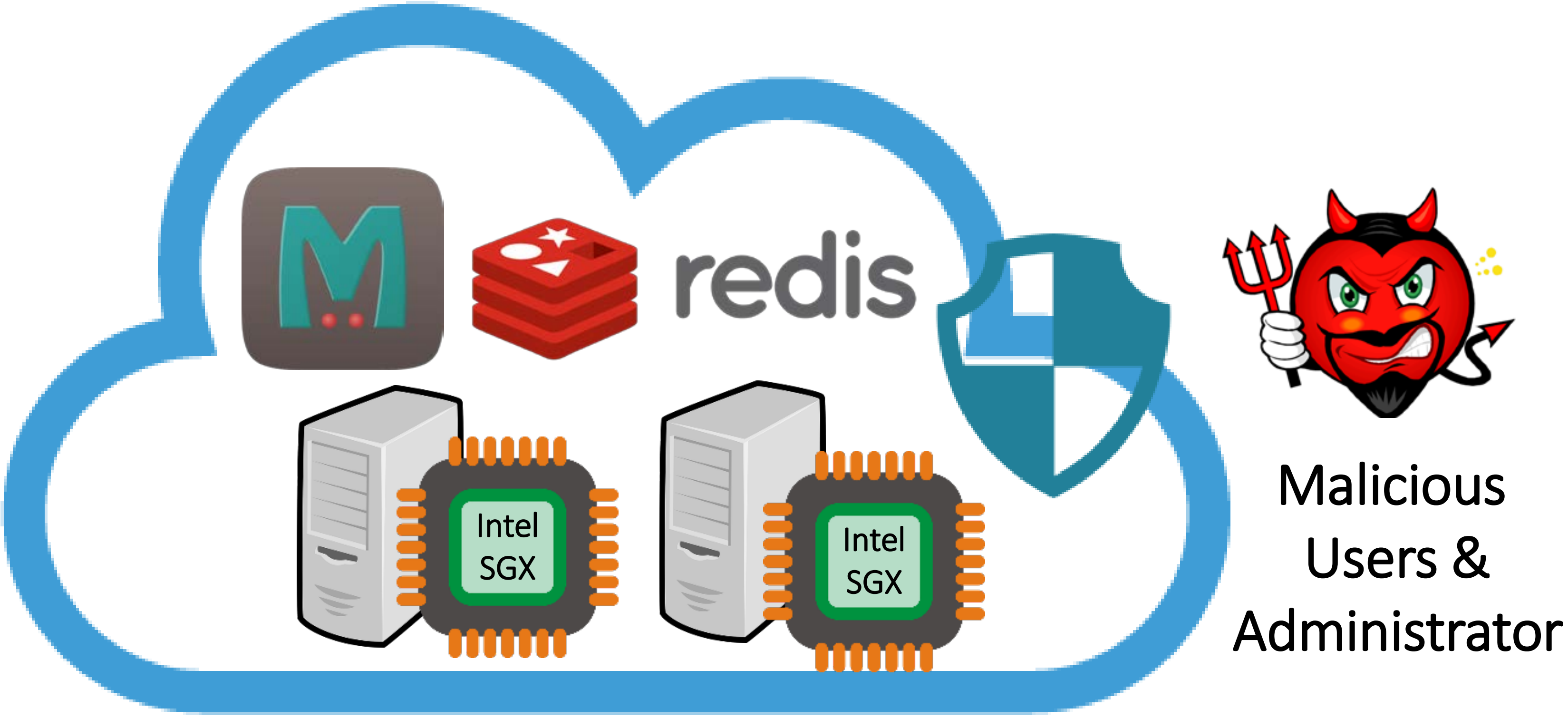


Malicious Users & Administrator

Trusted Key-value Stores

- User data is exposed to malicious attackers in clouds

- Hardware-based security supports
  – Provide *trusted execution environment* for remote server



Trusted Key-value Stores

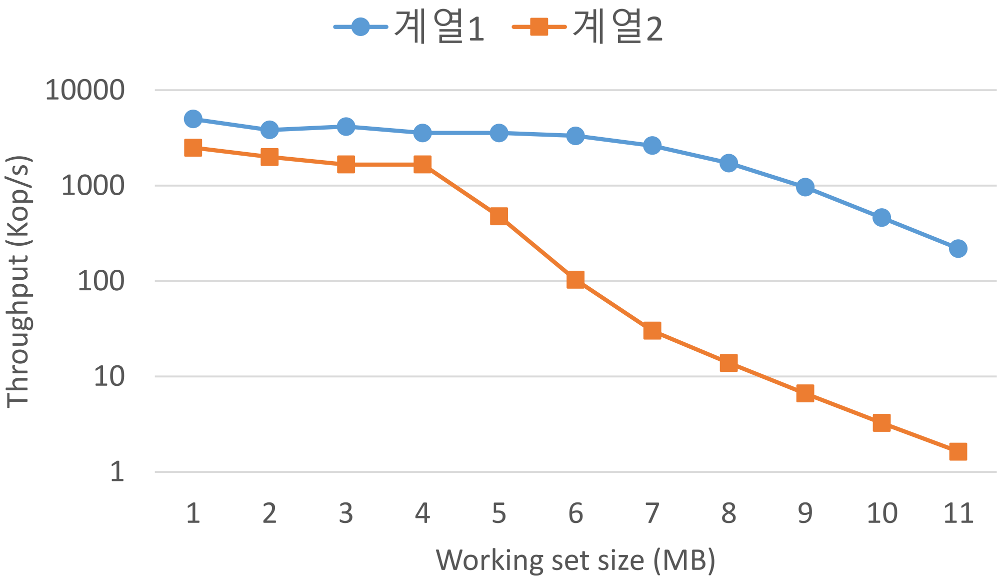
- User data is exposed to malicious attackers in clouds

- Hardware-based security supports
  - Provide *trusted execution environment* for remote server

Need trusted key-value stores

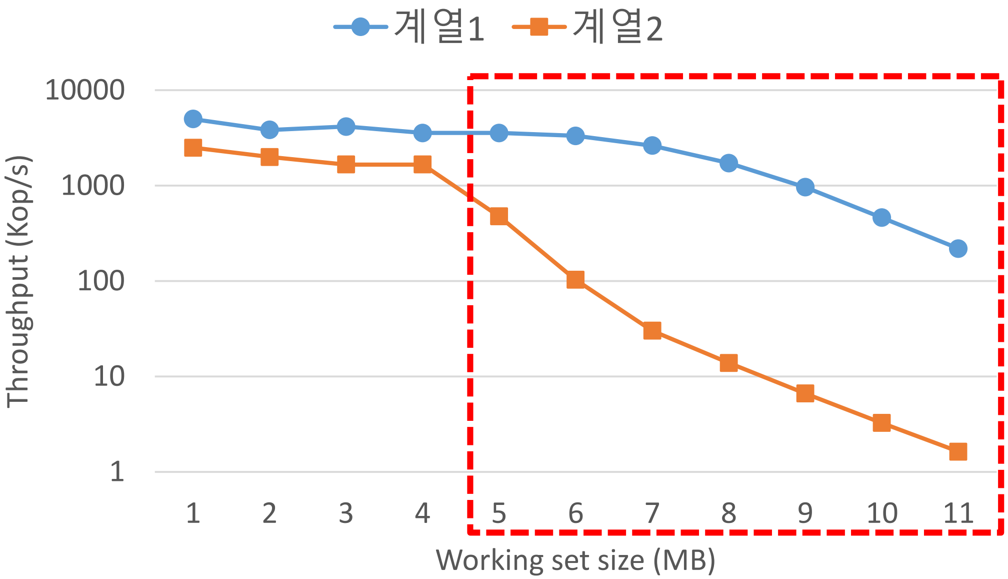
Malicious Users & Administrator

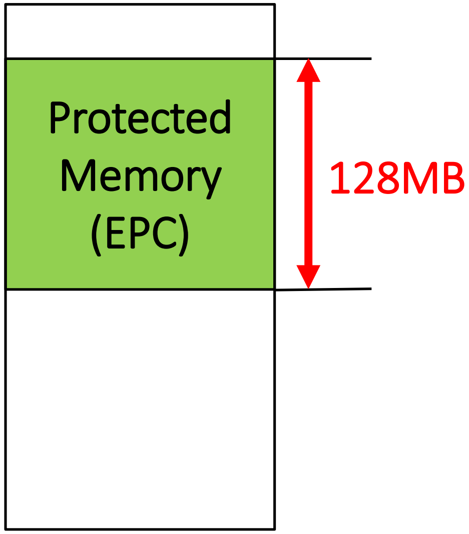
Trusted Key-value Stores with SGX



Trusted Key-value Stores with SGX

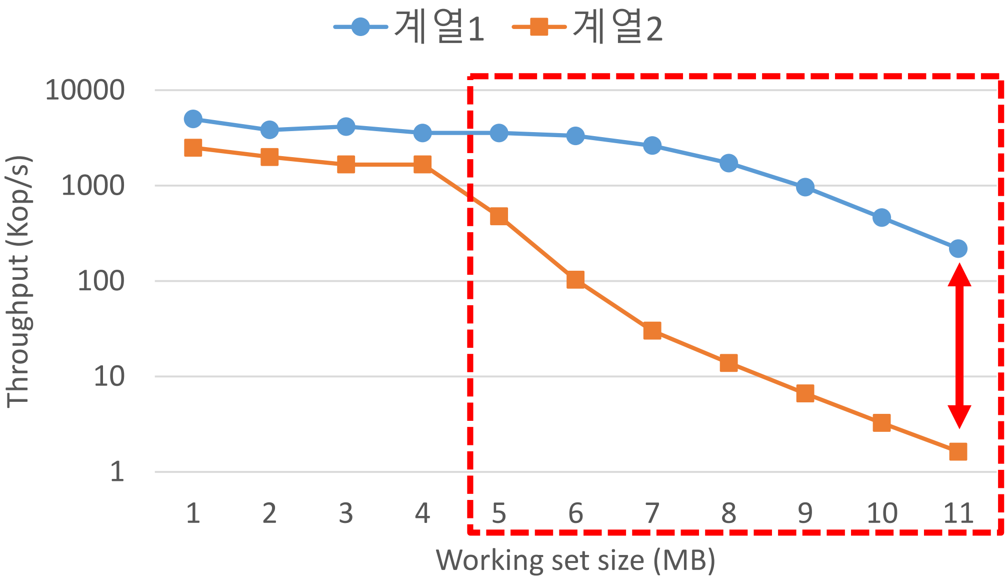
- Protected memory is limited to 128MB
  - Application can use about 92MB

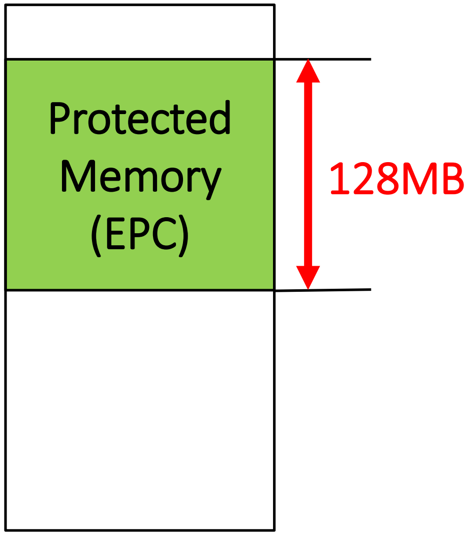




Trusted Key-value Stores with SGX

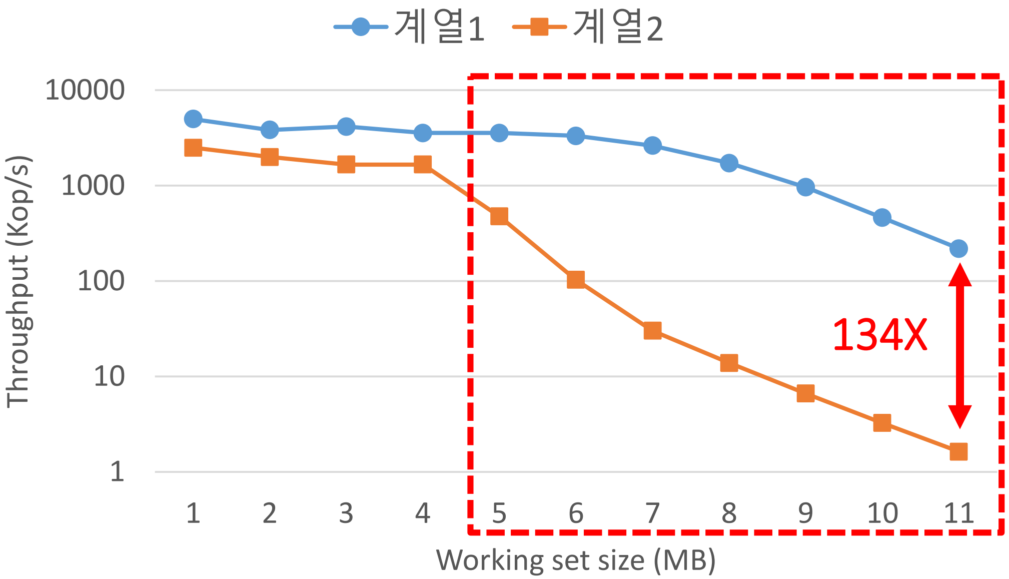
- Protected memory is limited to 128MB
  - Application can use about 92MB

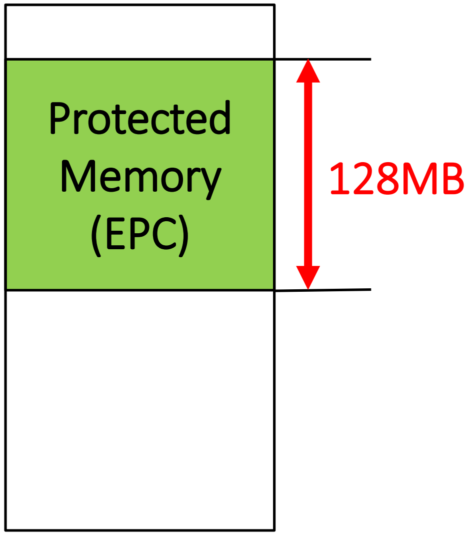




Trusted Key-value Stores with SGX

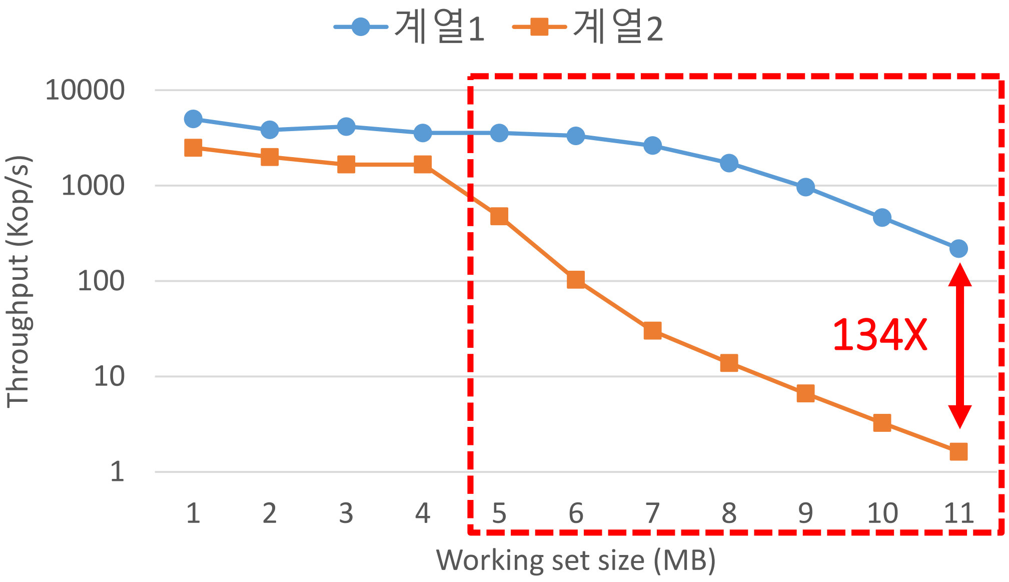
- Protected memory is limited to 128MB
  - – Application can use about 92MB

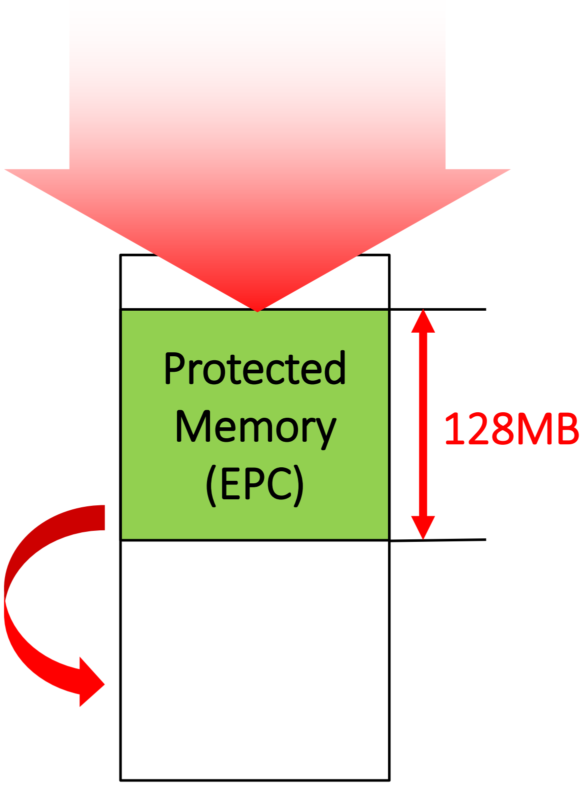




Trusted Key-value Stores with SGX

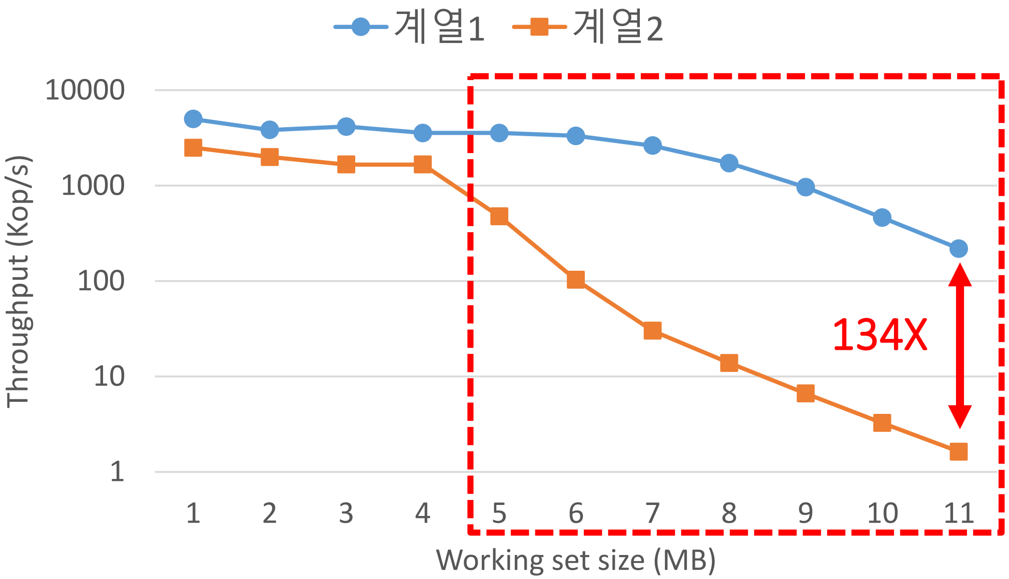
- Protected memory is limited to 128MB
  – Application can use about 92MB

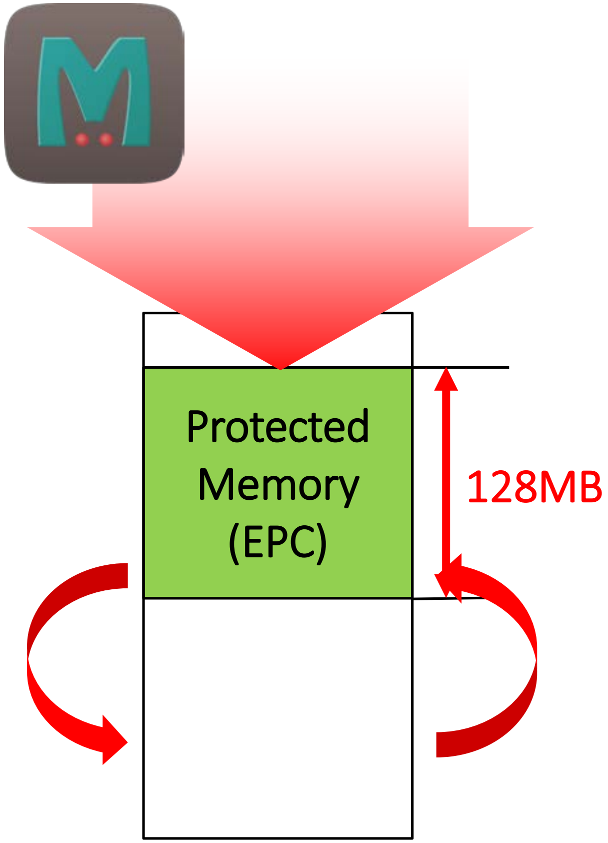




Trusted Key-value Stores with SGX

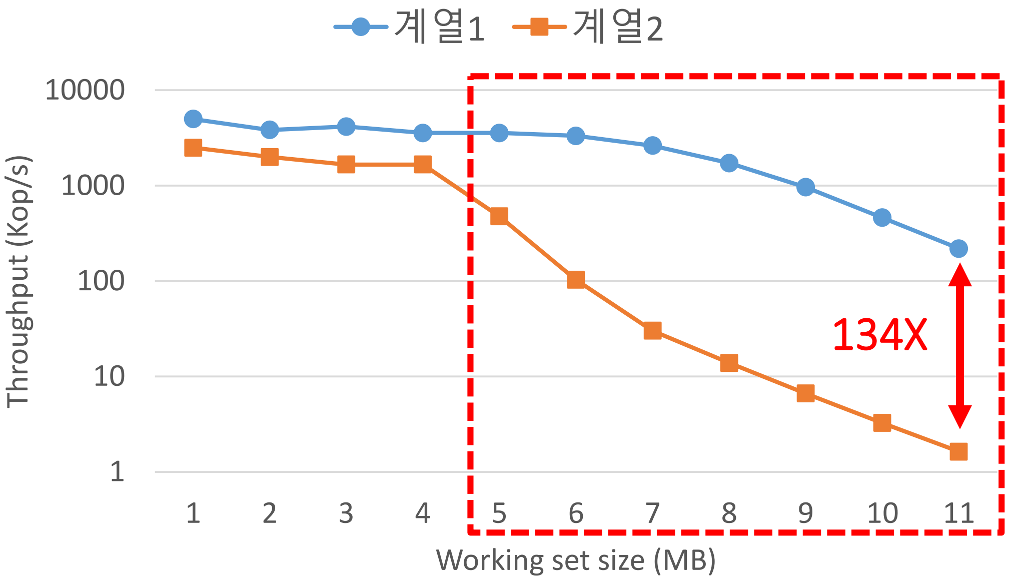
- Protected memory is limited to 128MB
  - Application can use about 92MB

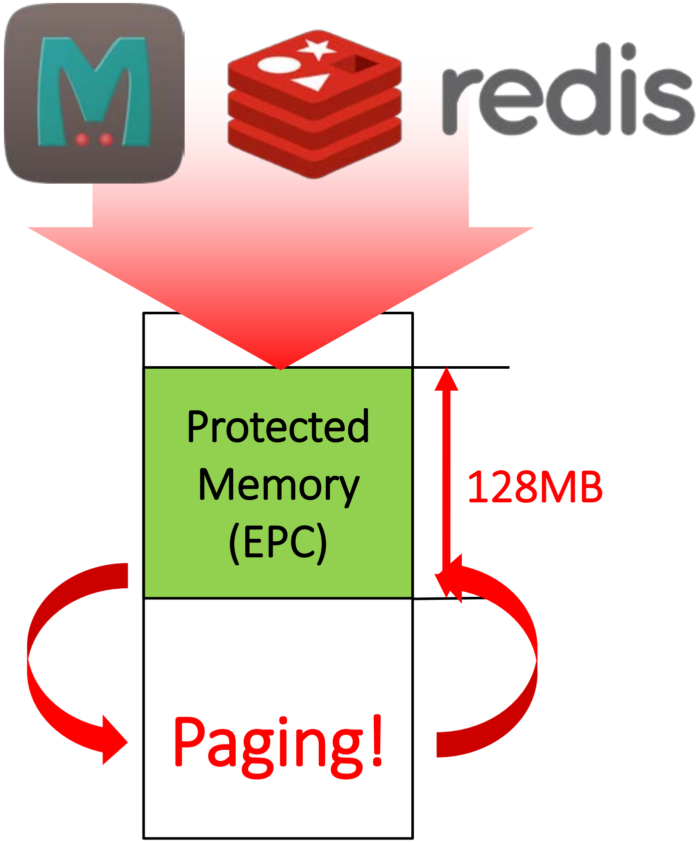




Trusted Key-value Stores with SGX

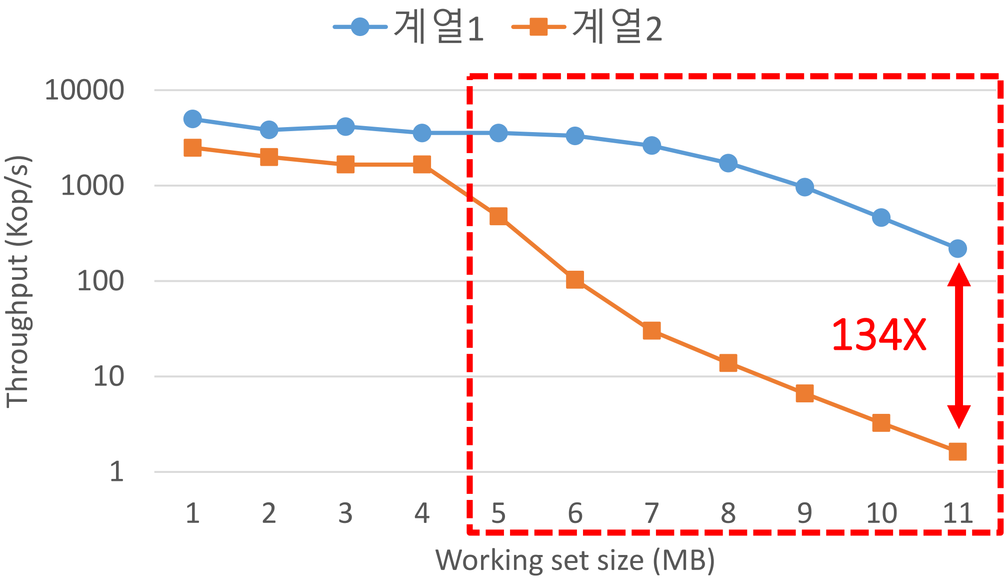
- Protected memory is limited to 128MB
  - Application can use about 92MB

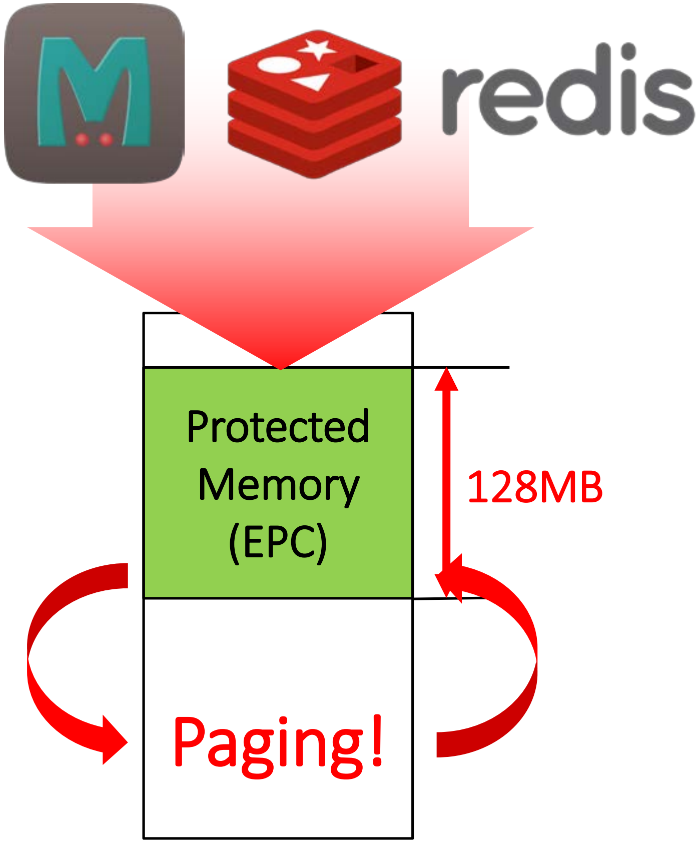




Trusted Key-value Stores with SGX

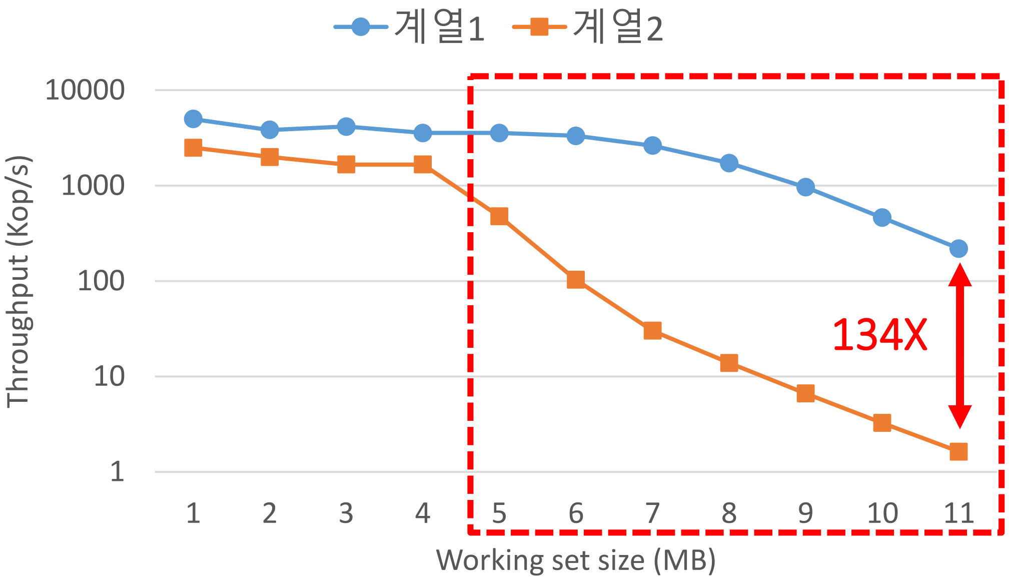
- Protected memory is limited to 128MB
  - Application can use about 92MB

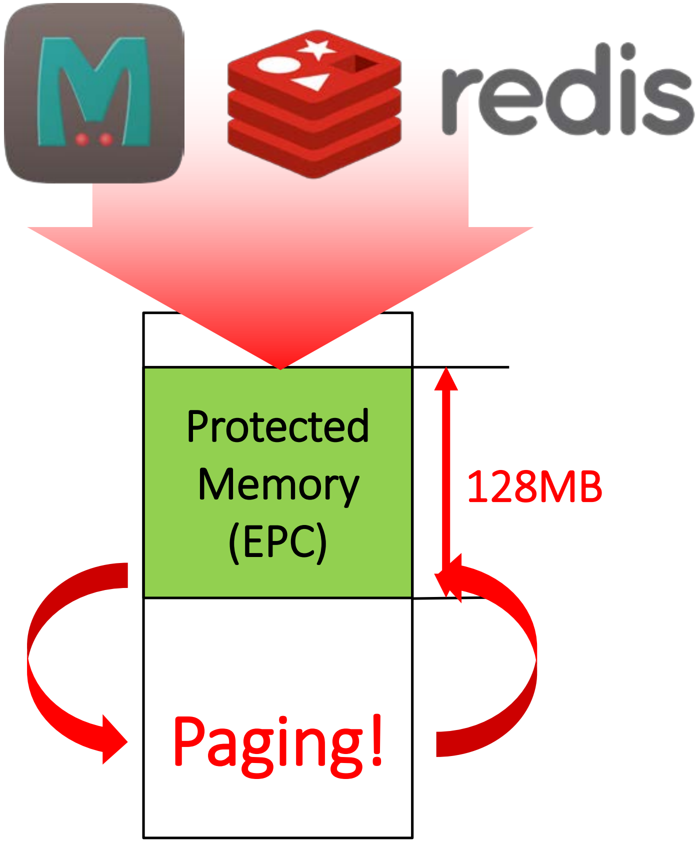




Trusted Key-value Stores with SGX

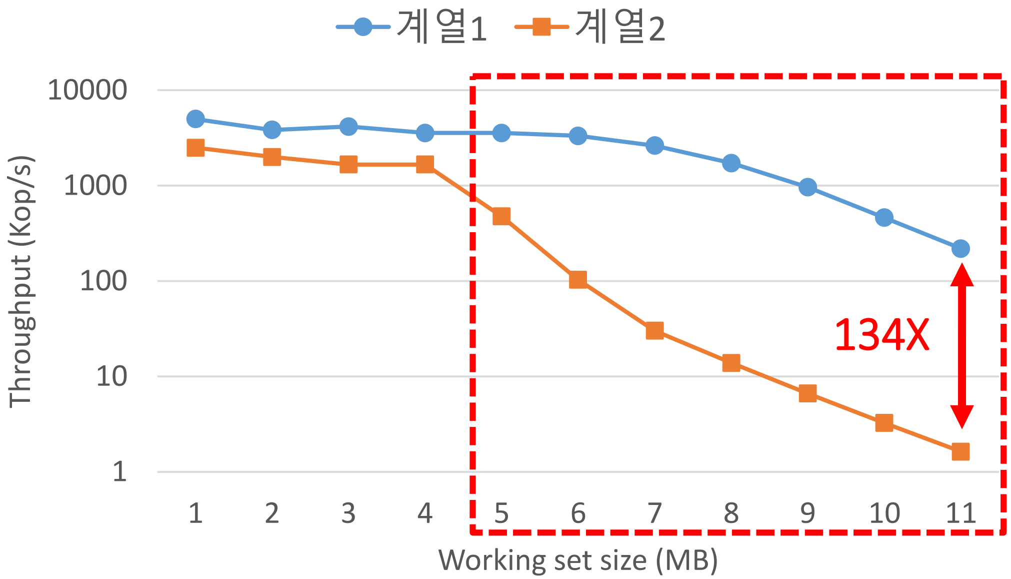
- Protected memory is limited to 128MB
  - – Application can use about 92MB

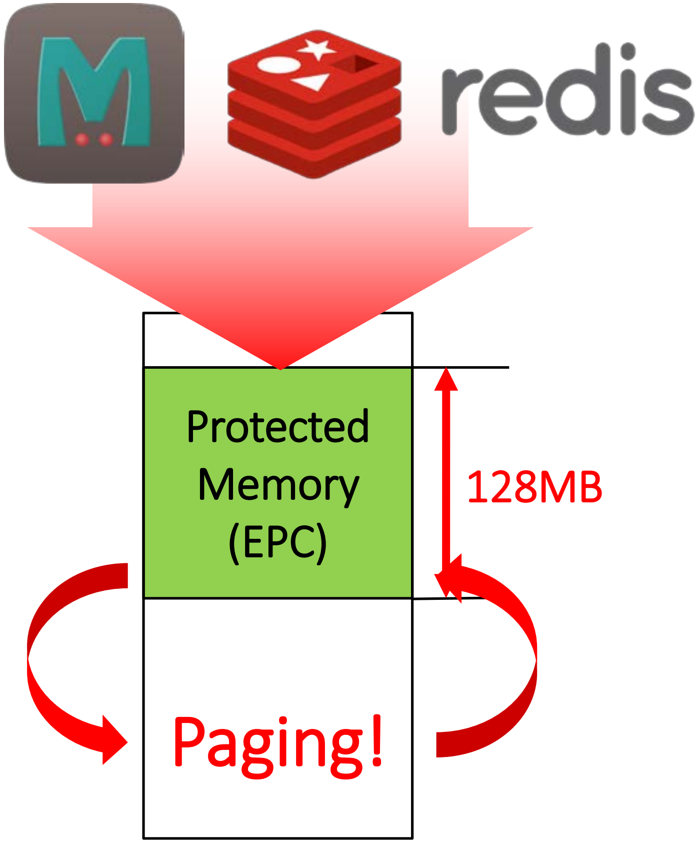




Trusted Key-value Stores with SGX

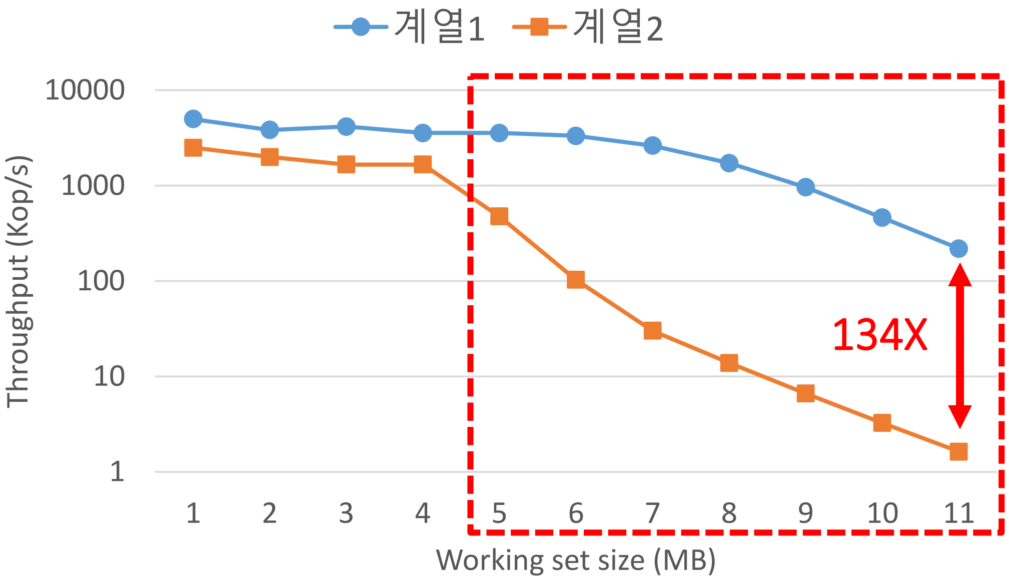
- Protected memory is limited to 128MB
  - – Application can use about 92MB

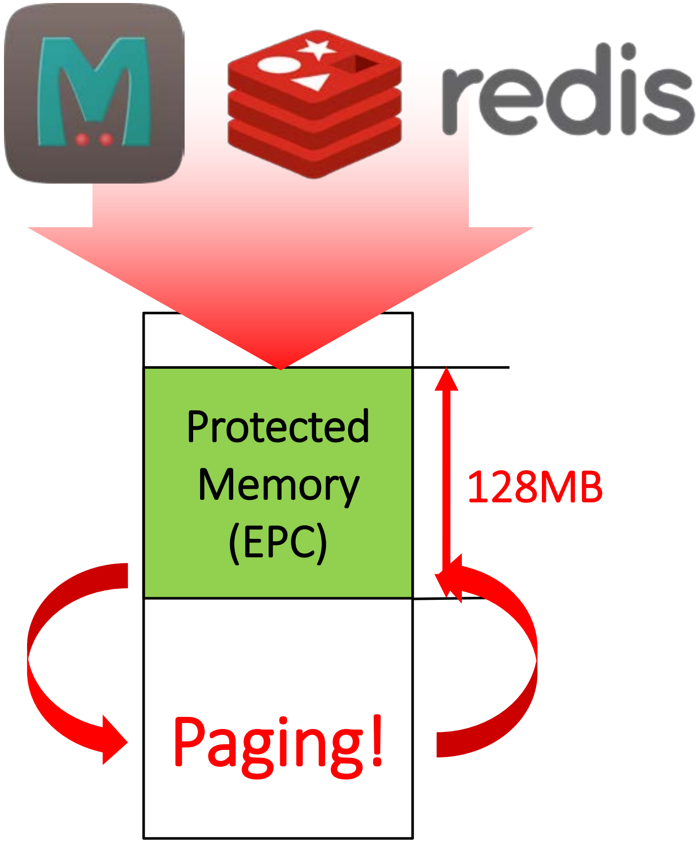




Trusted Key-value Stores with SGX

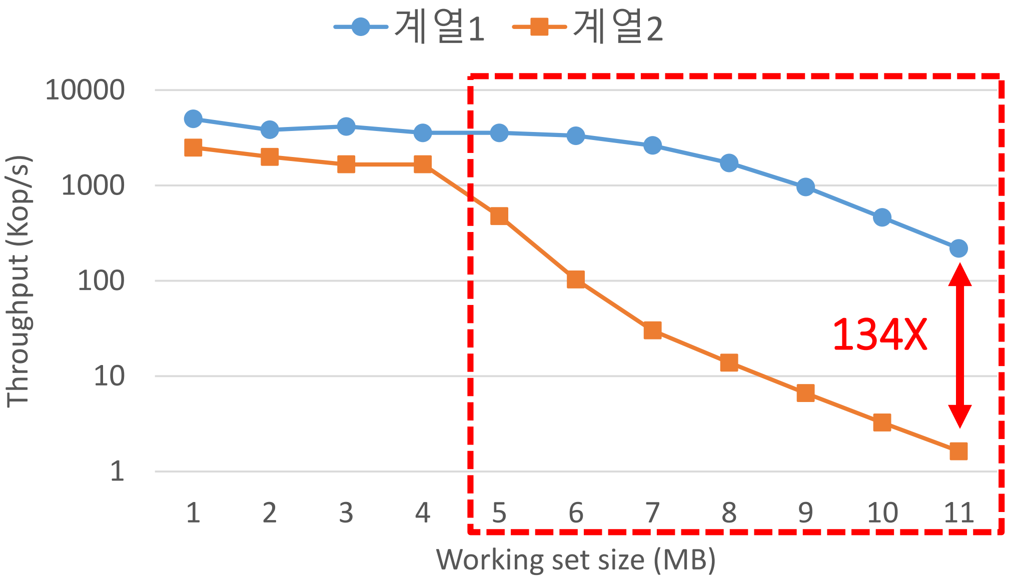
- Protected memory is limited to 128MB
  - Application can use about 92MB

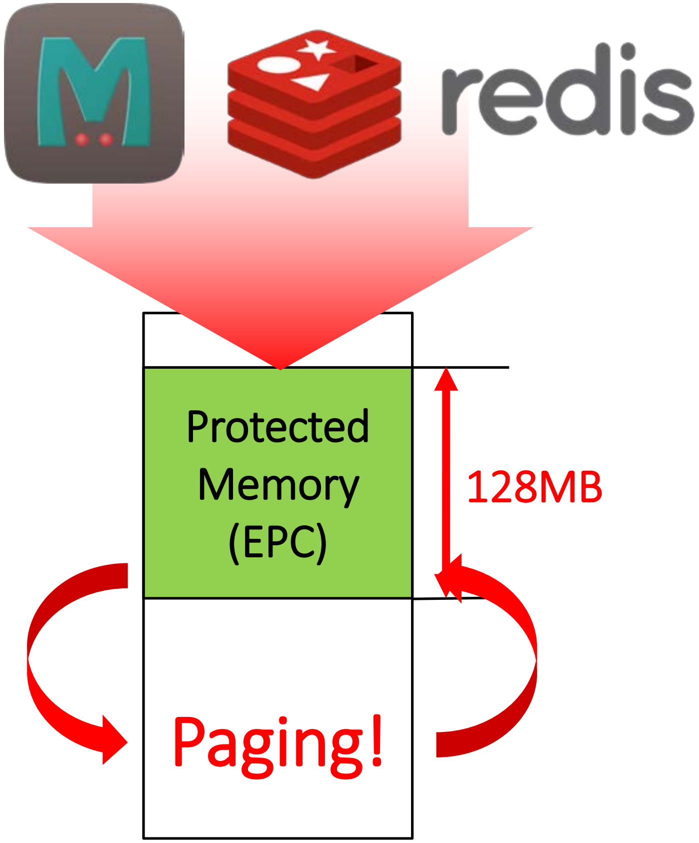




Trusted Key-value Stores with SGX

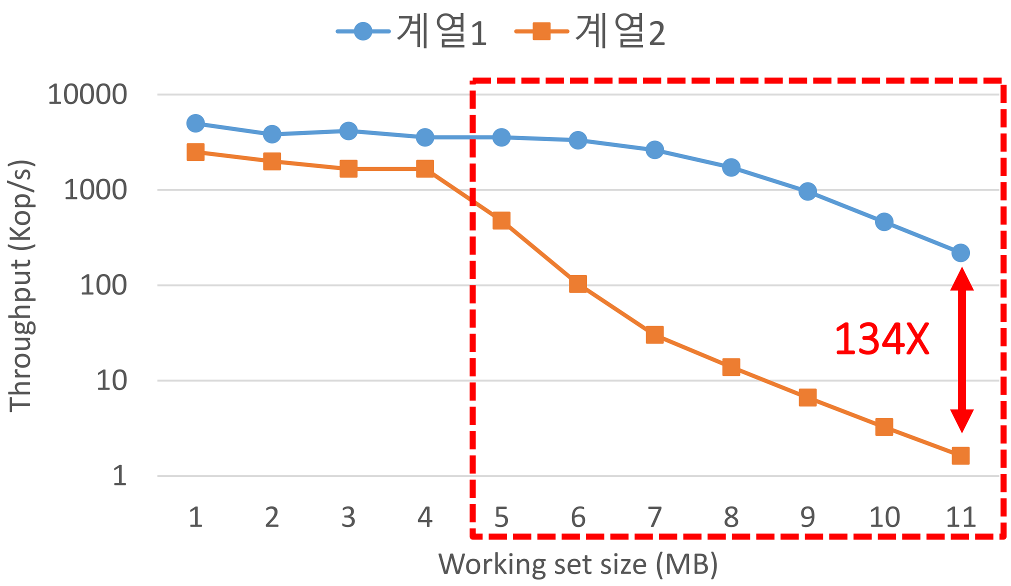
- Protected memory is limited to 128MB
  - Application can use about 92MB



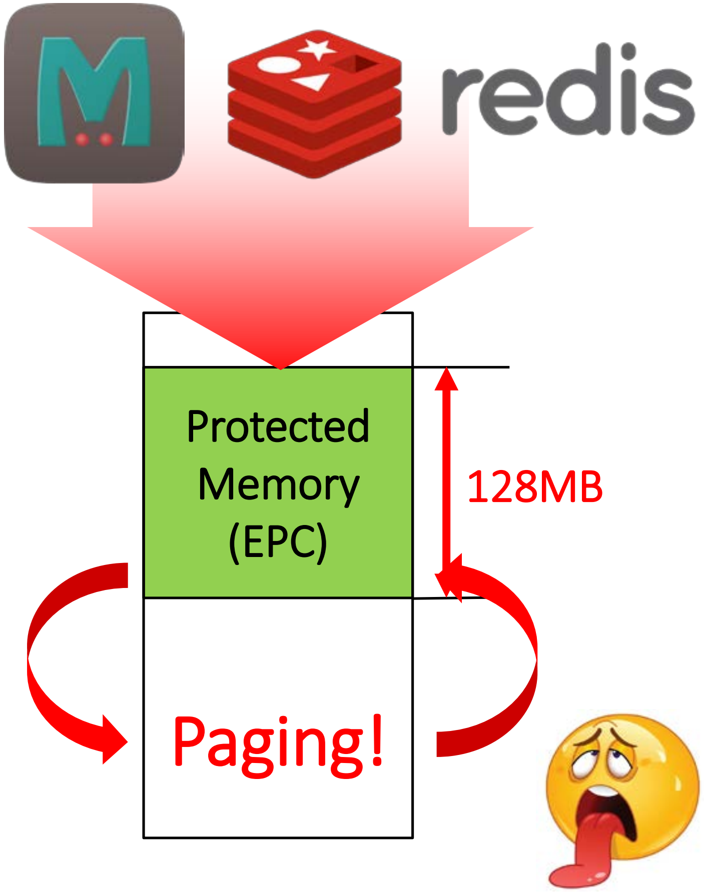


Trusted Key-value Stores with SGX

- Protected memory is limited to 128MB
  - Application can use about 92MB



134X



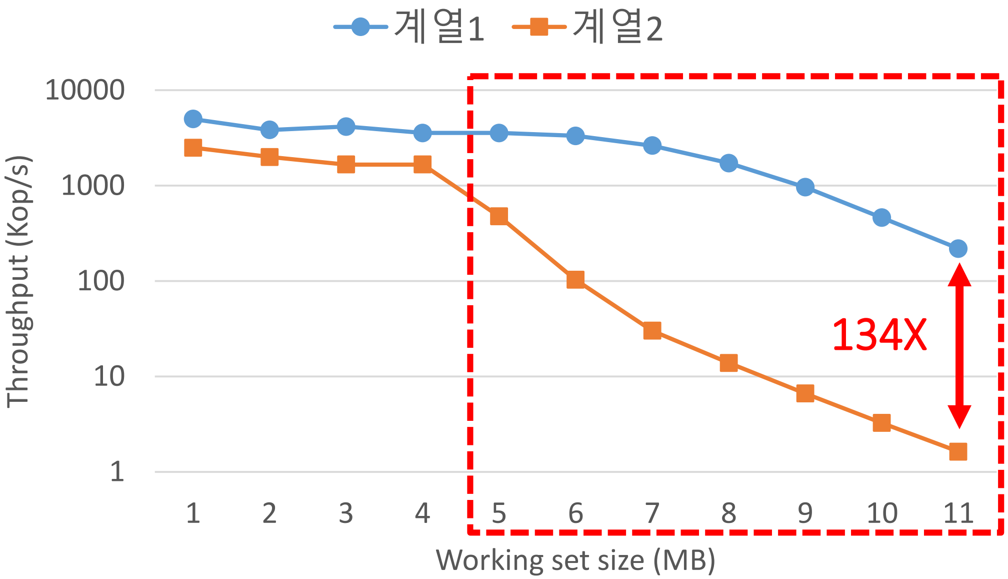
Protected Memory (EPC)

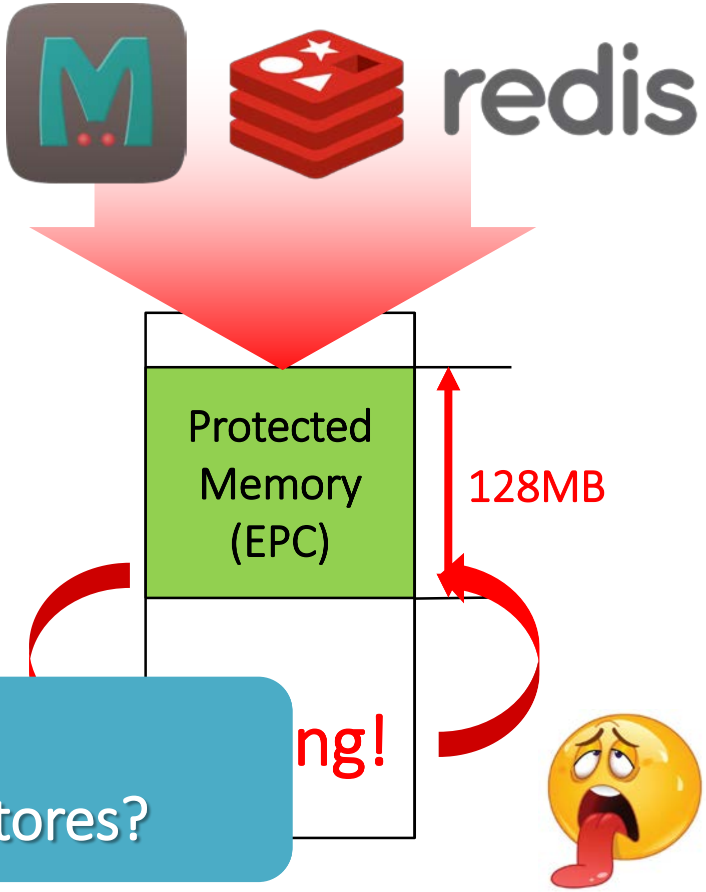
128MB

Paging!

Trusted Key-value Stores with SGX

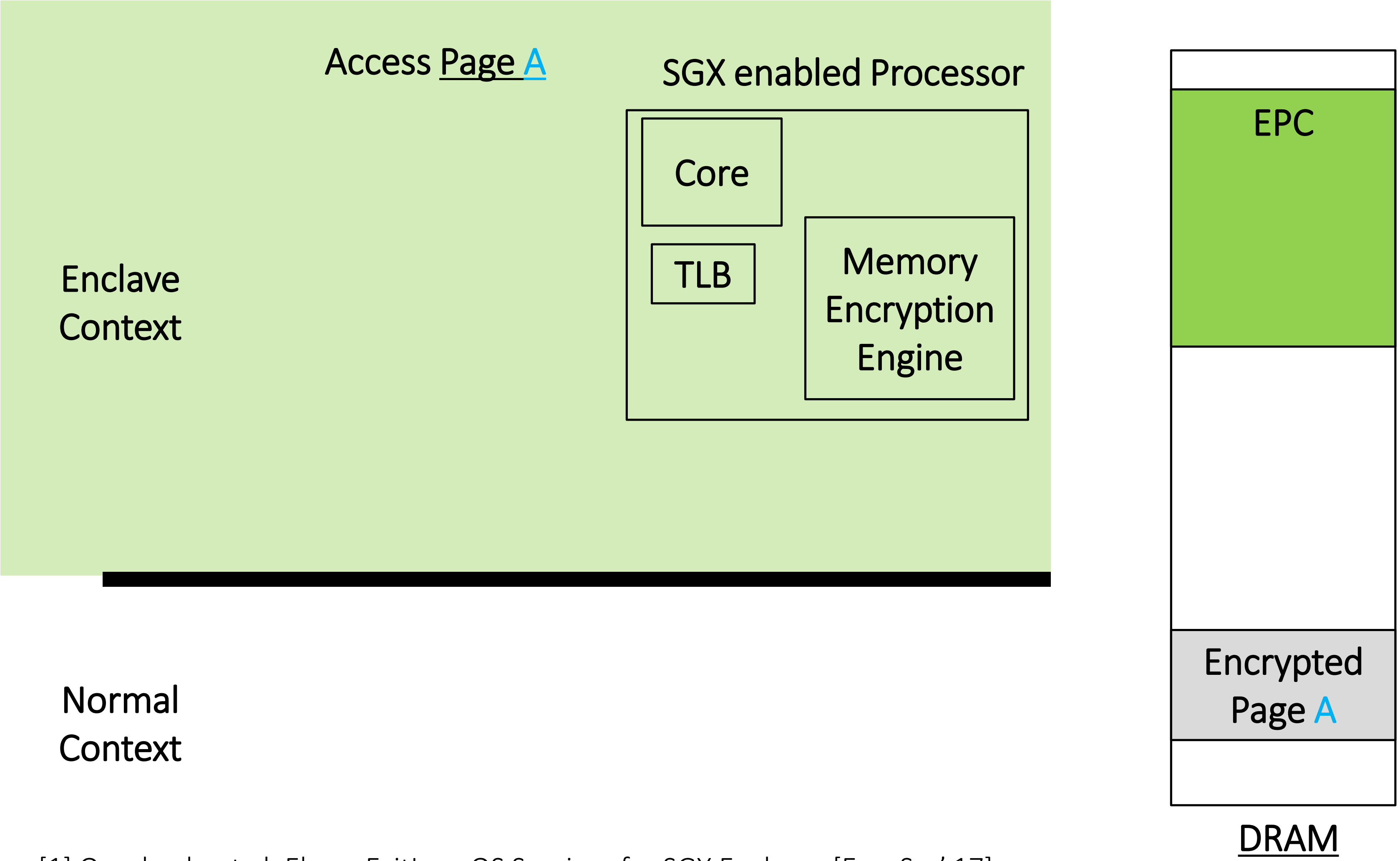
- Protected memory is limited to 128MB
  - Application can use about 92MB





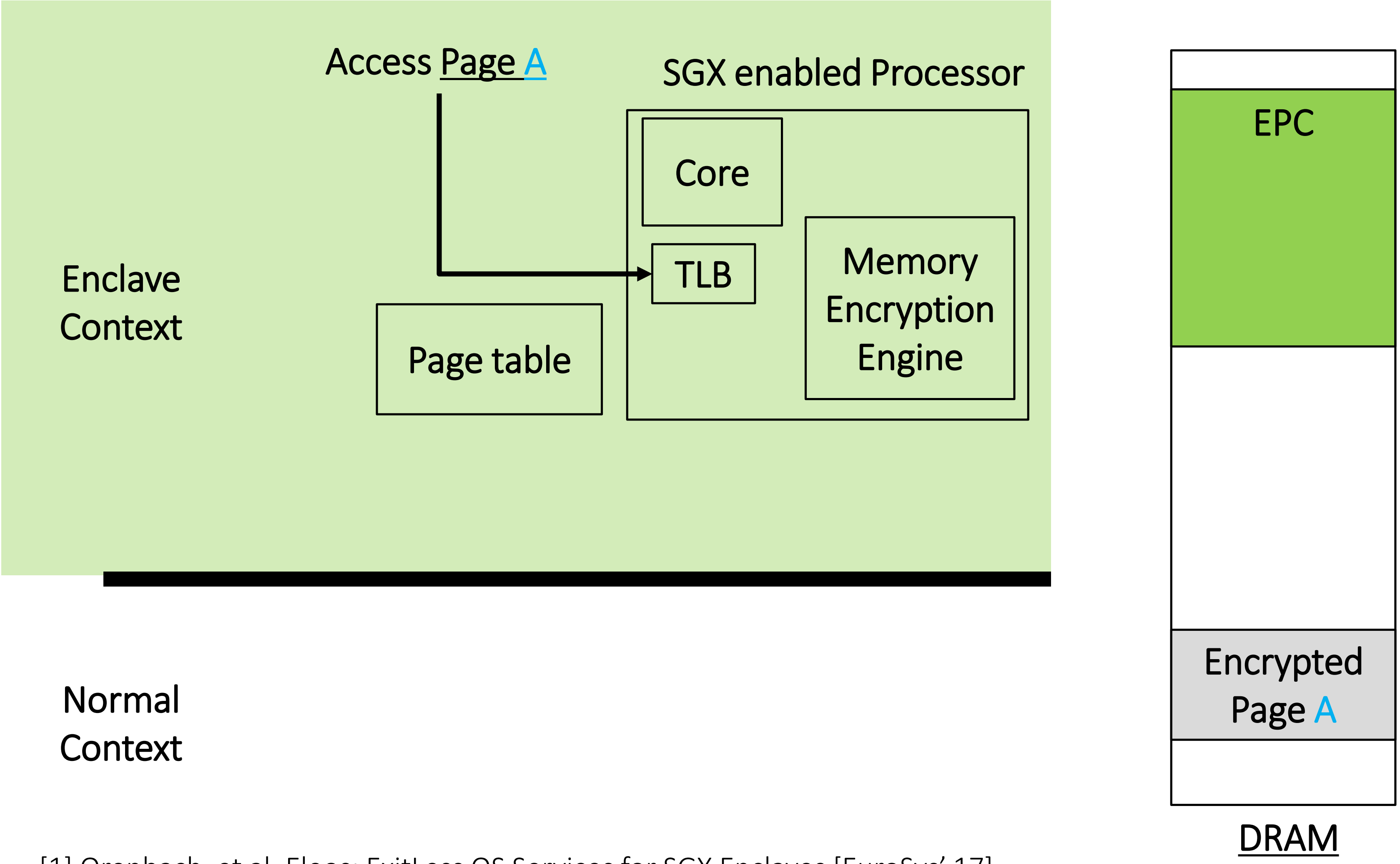
How can we have
efficient trusted key value stores?

Paging Mechanism of SGX



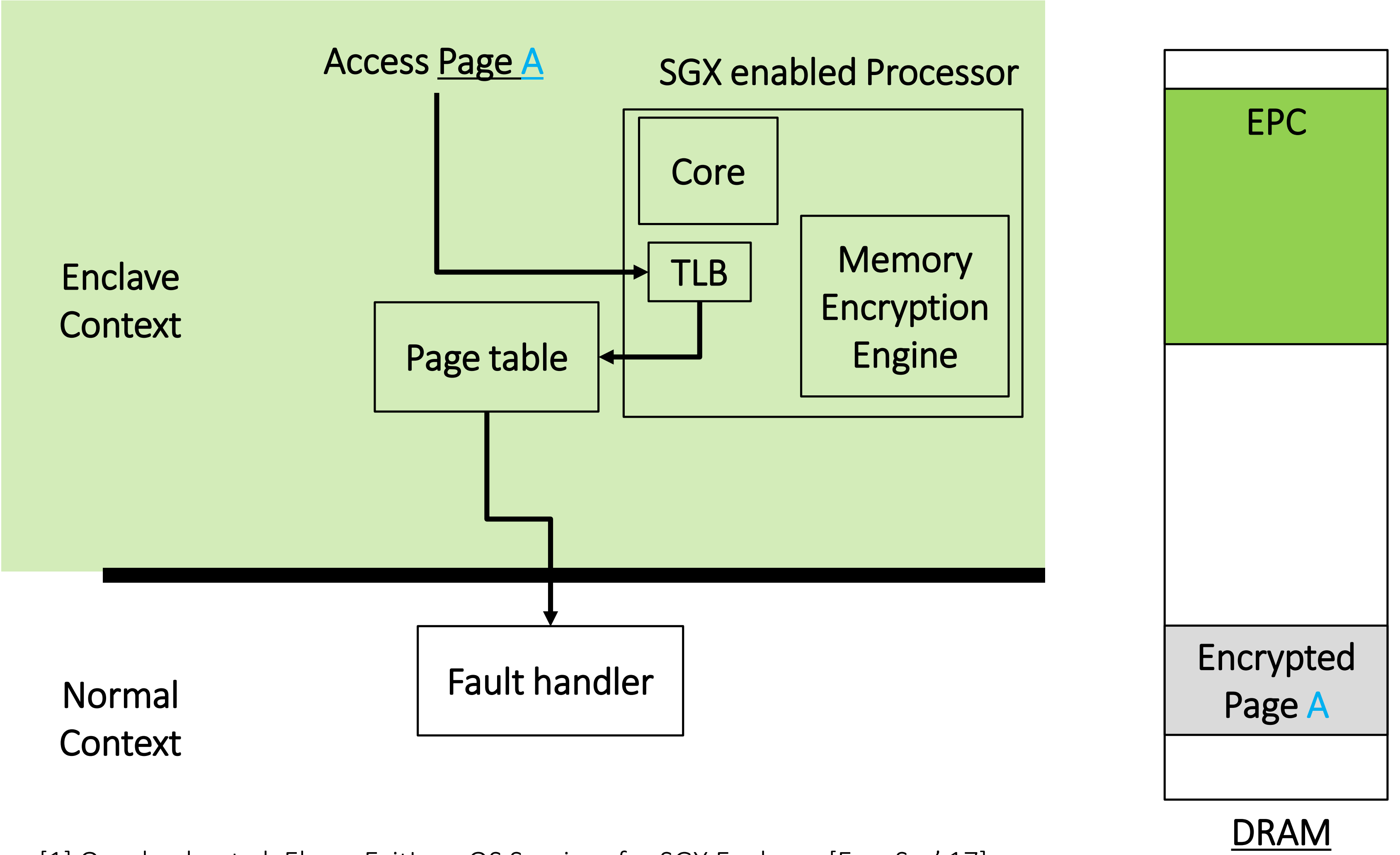
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



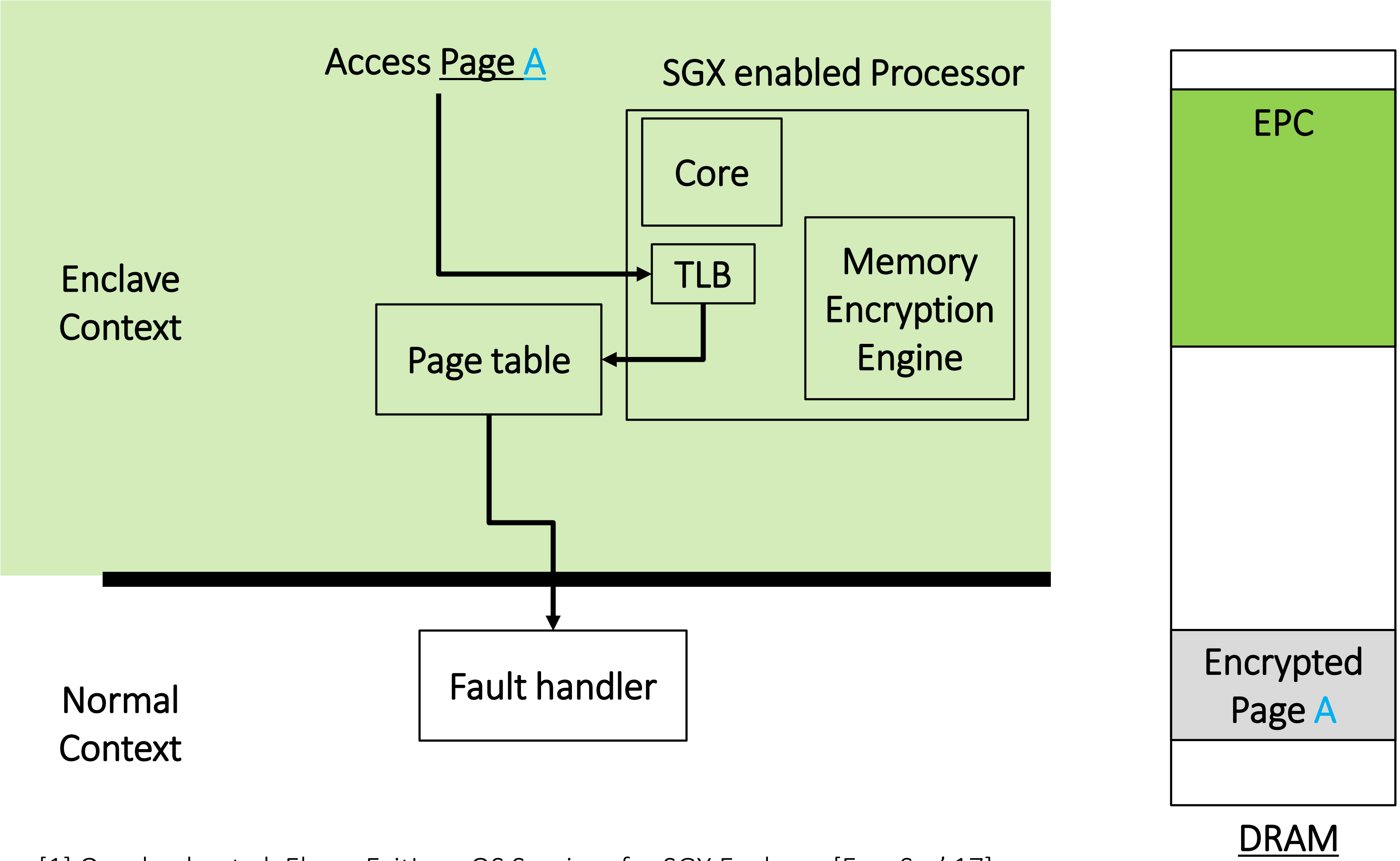
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



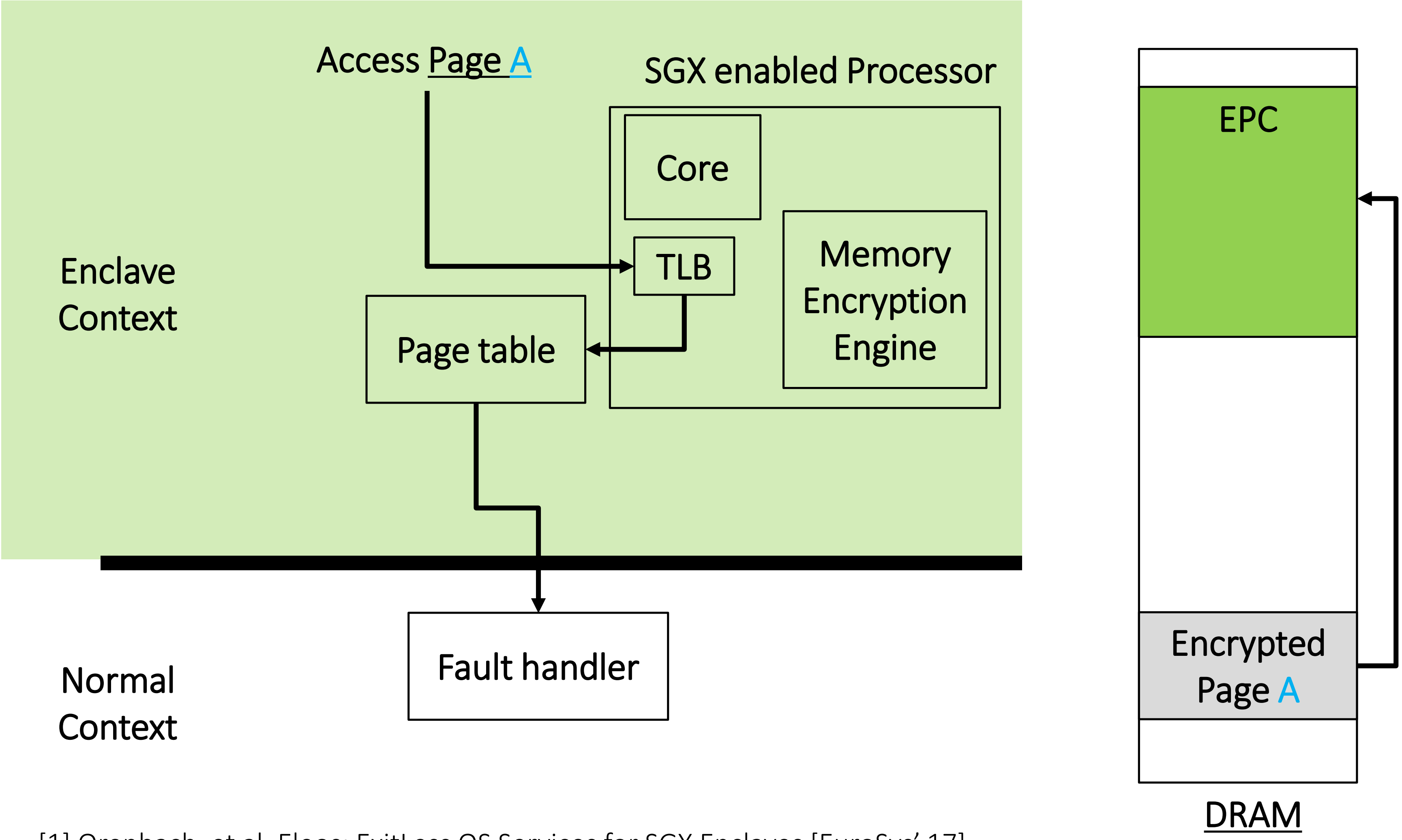
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



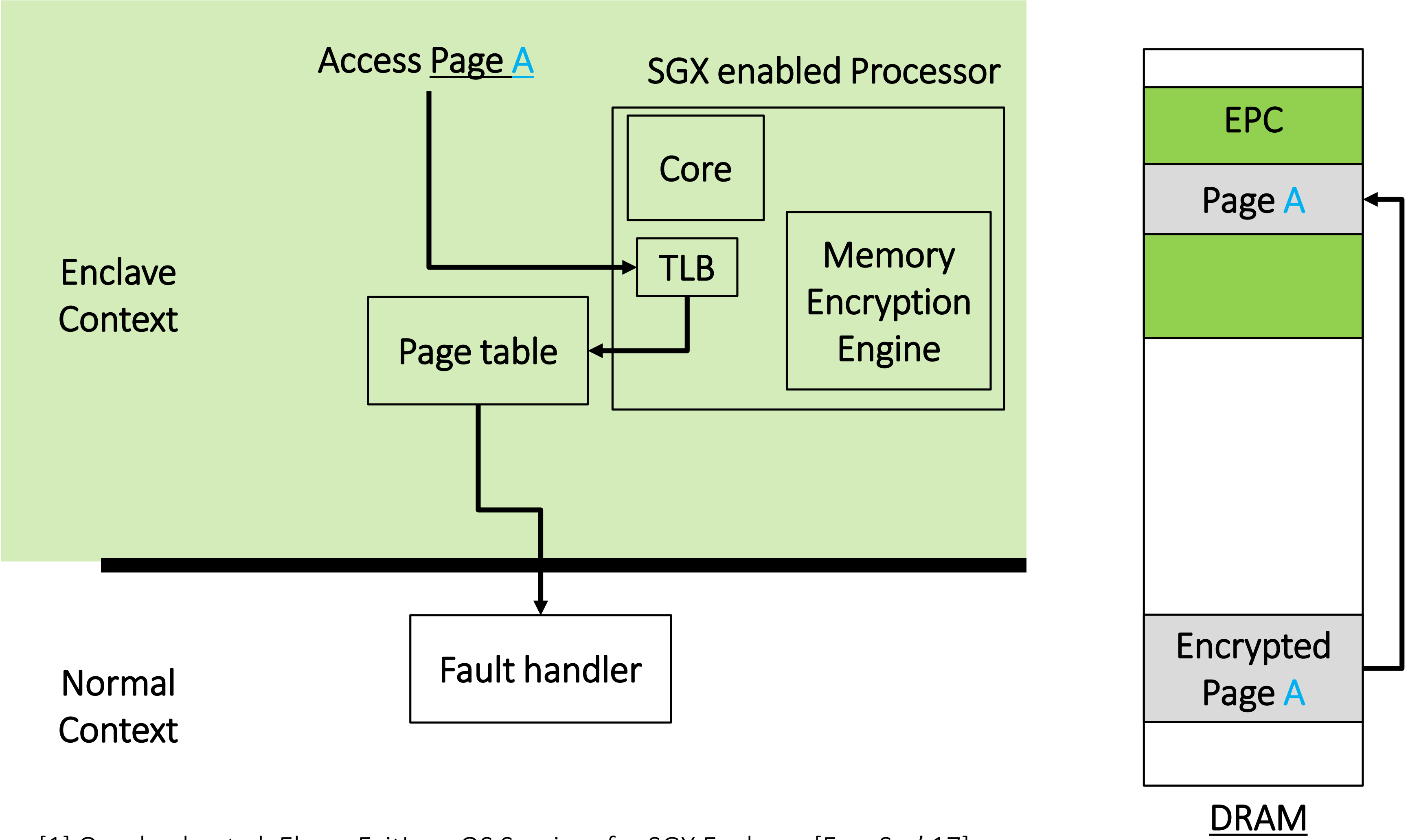
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



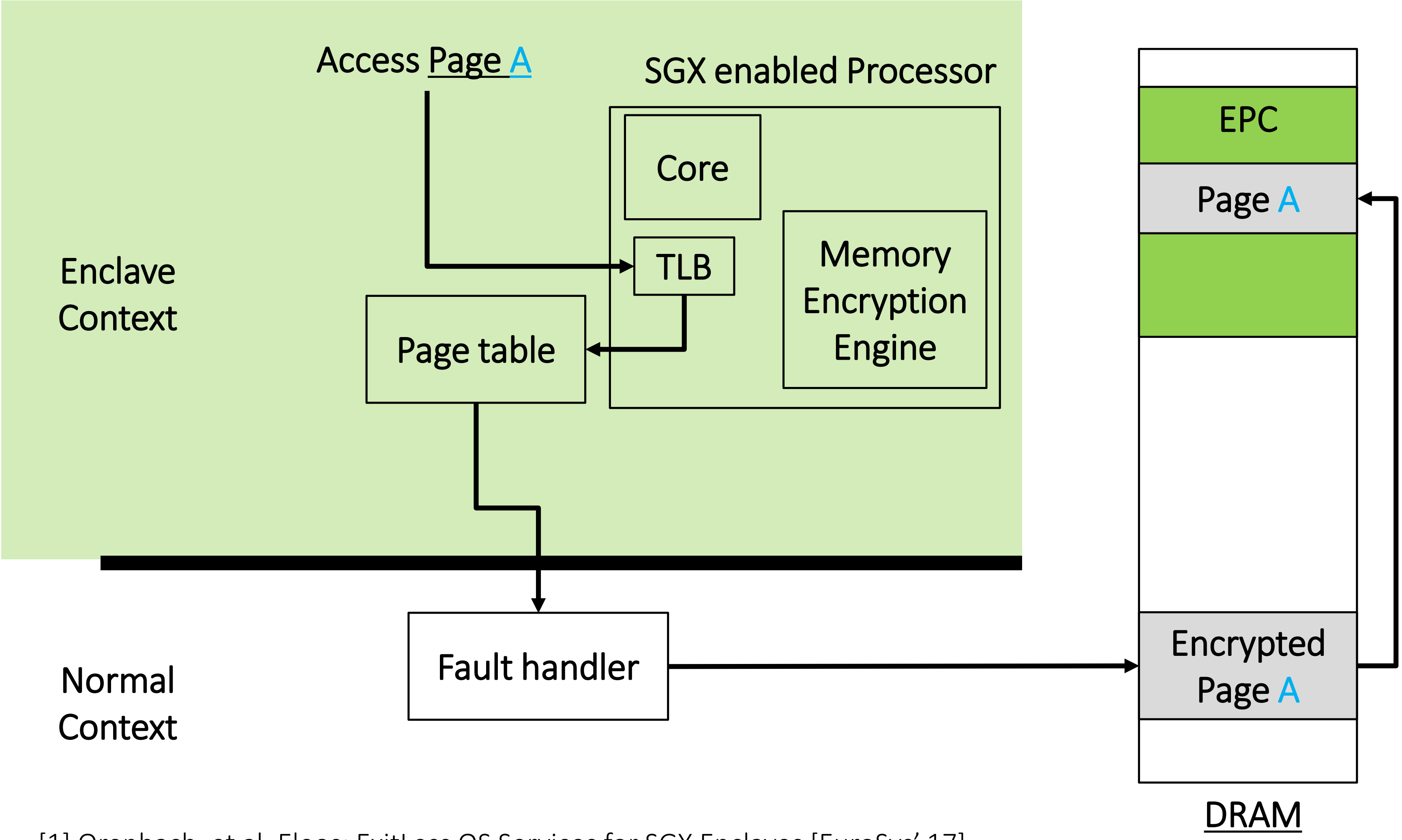
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



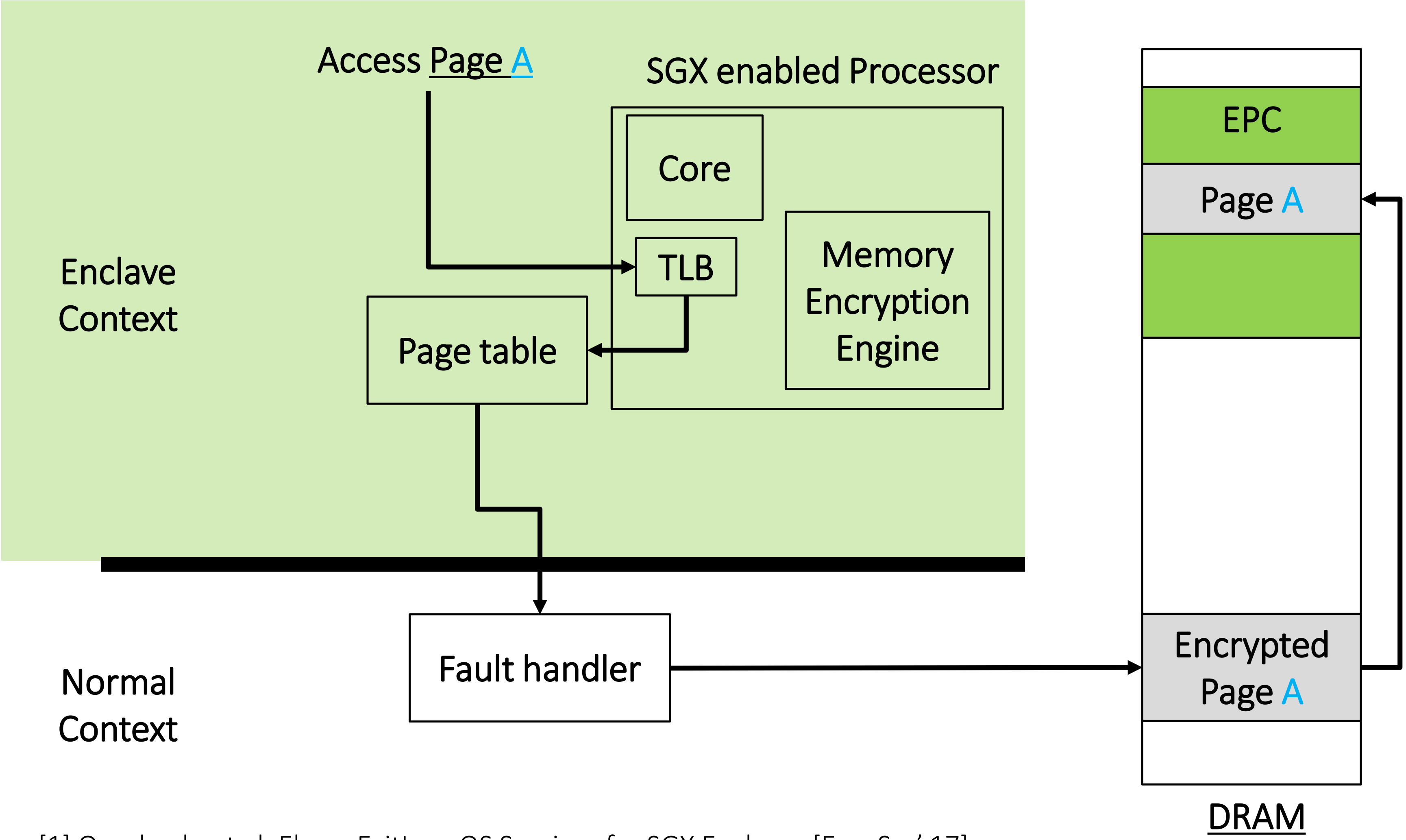
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



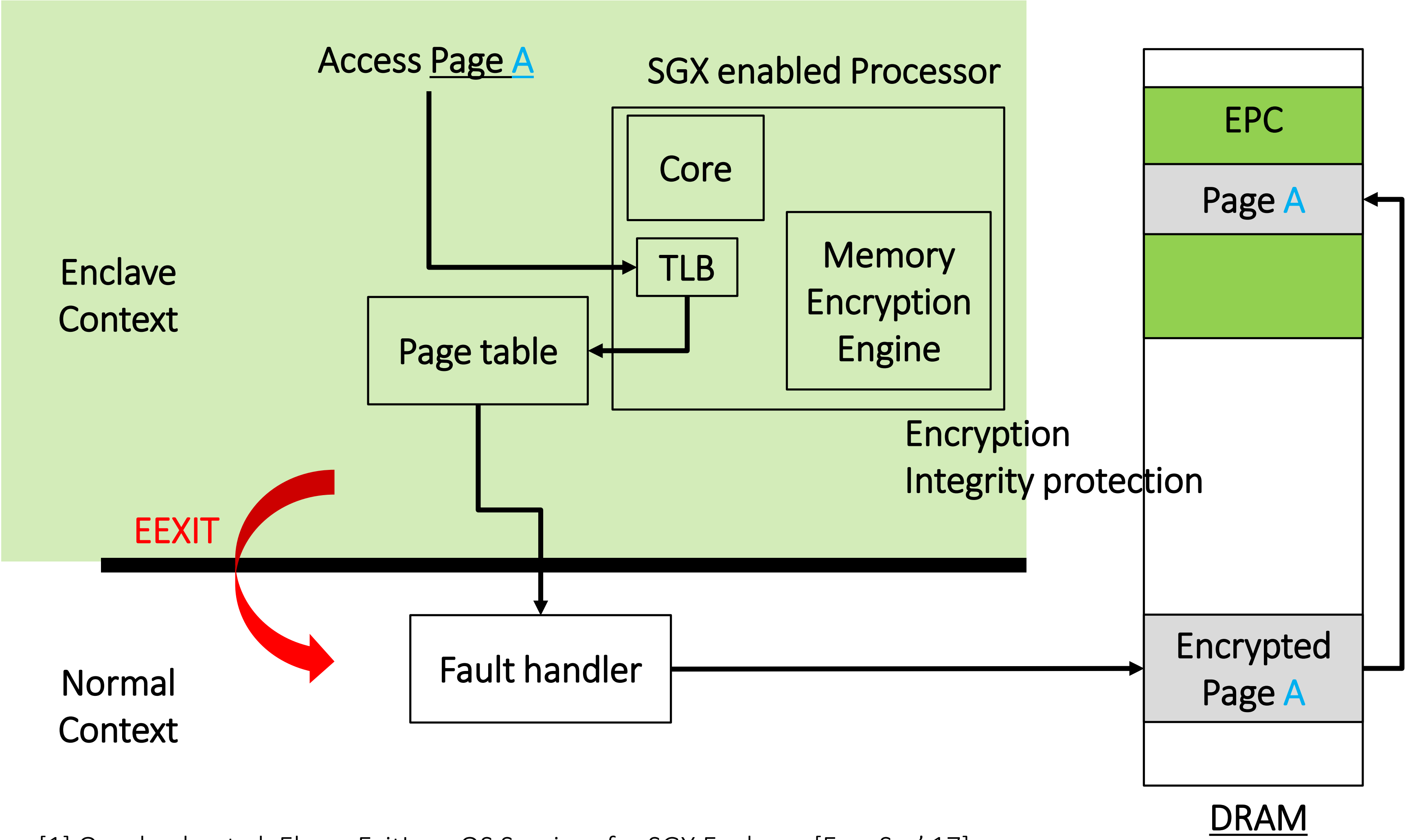
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



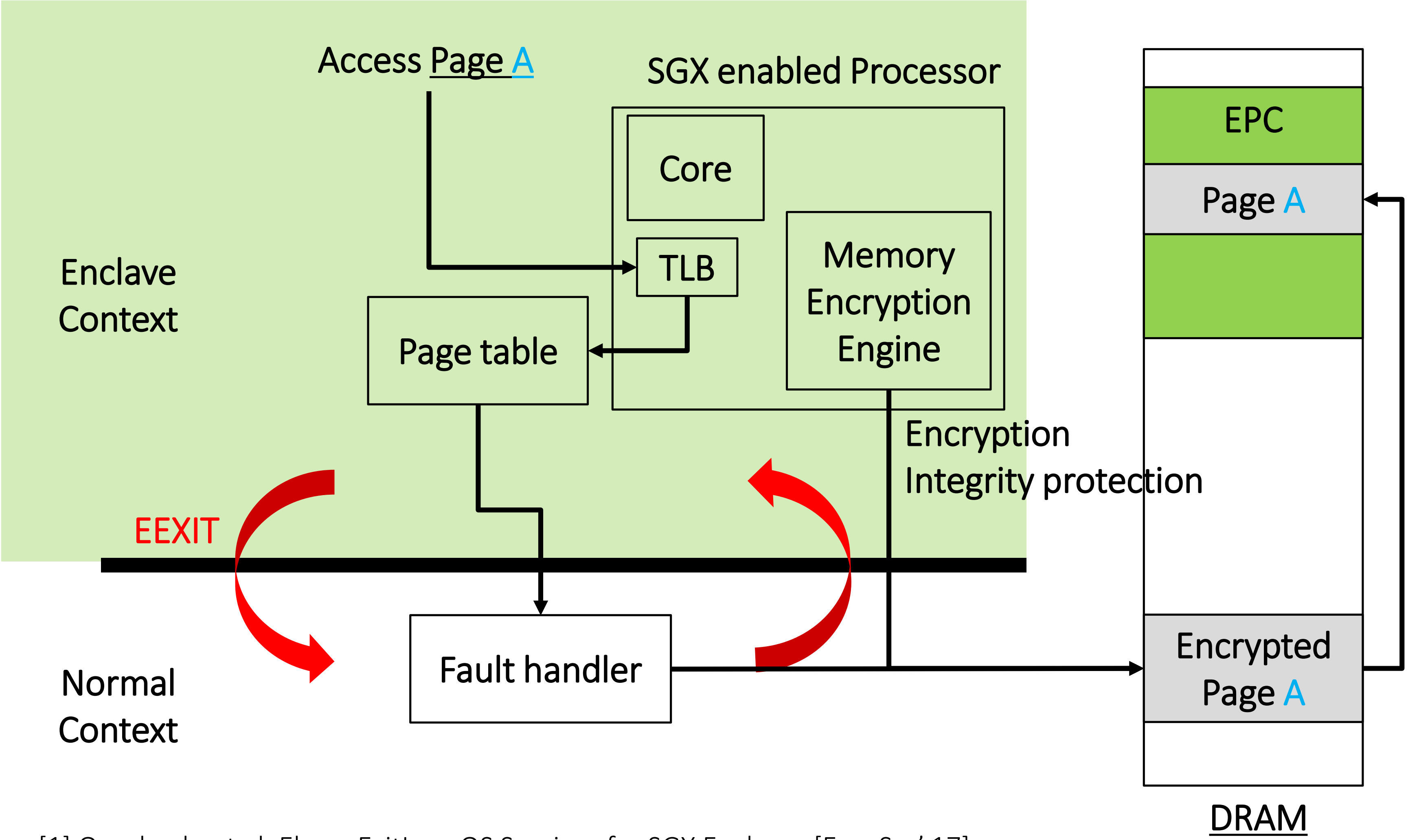
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



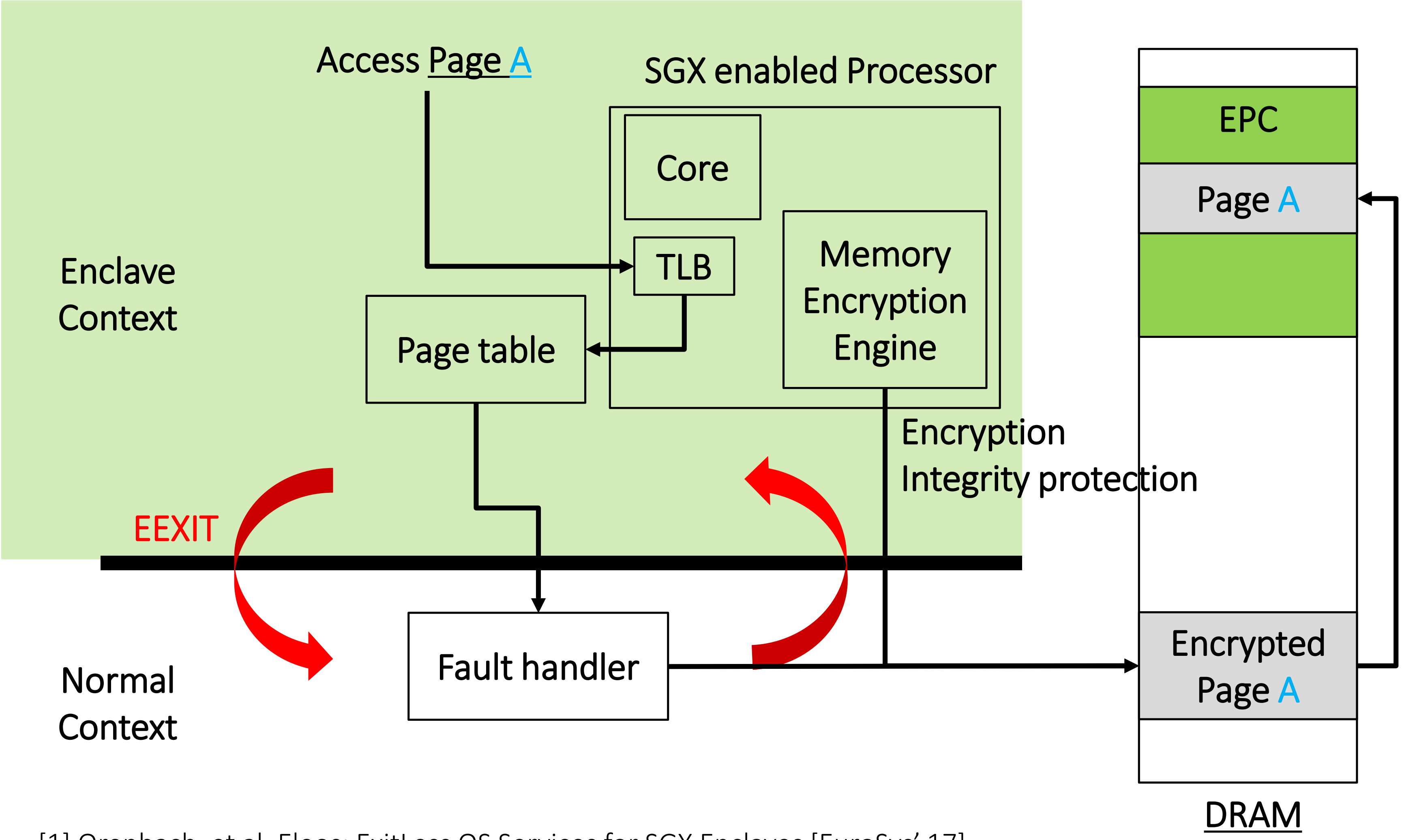
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



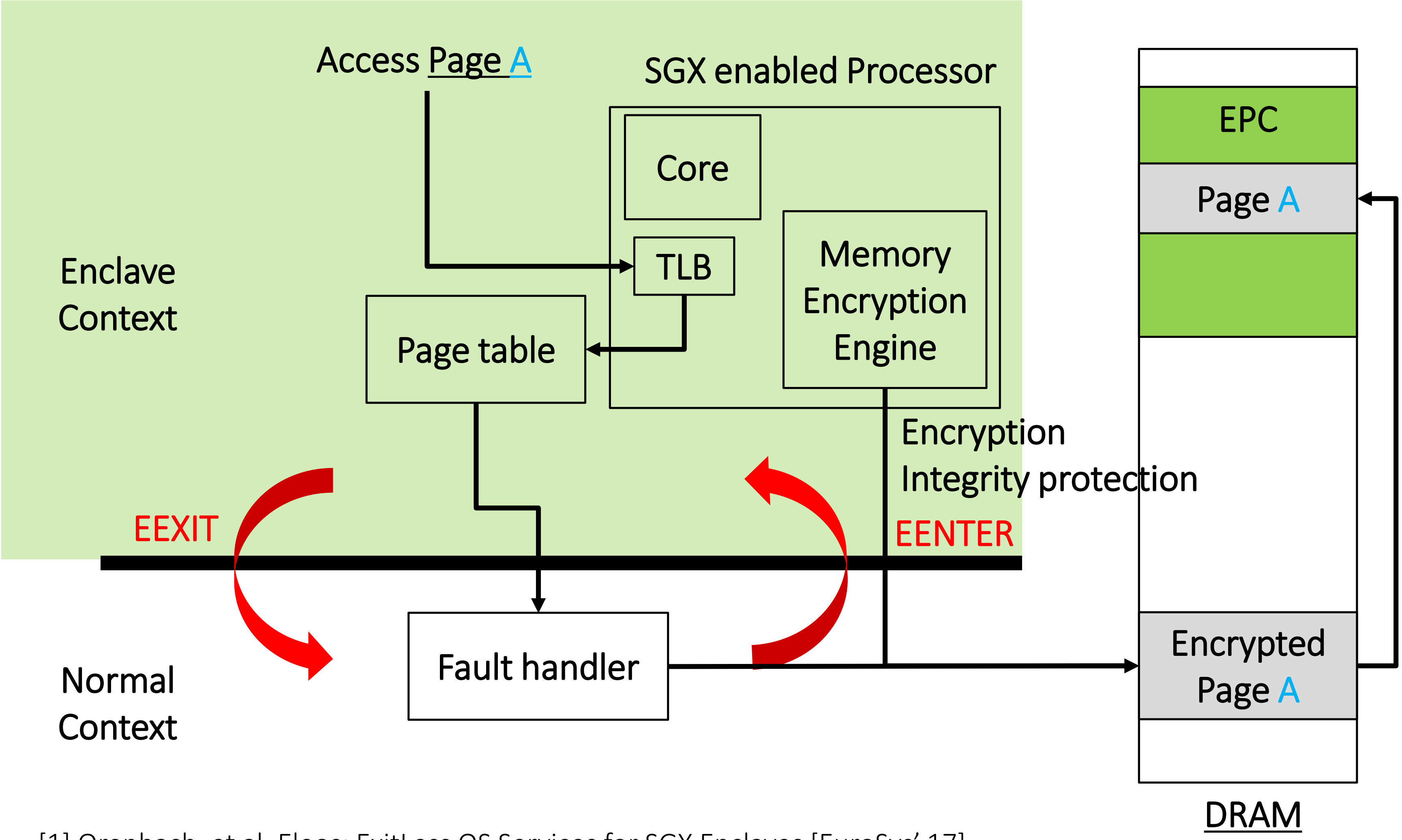
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



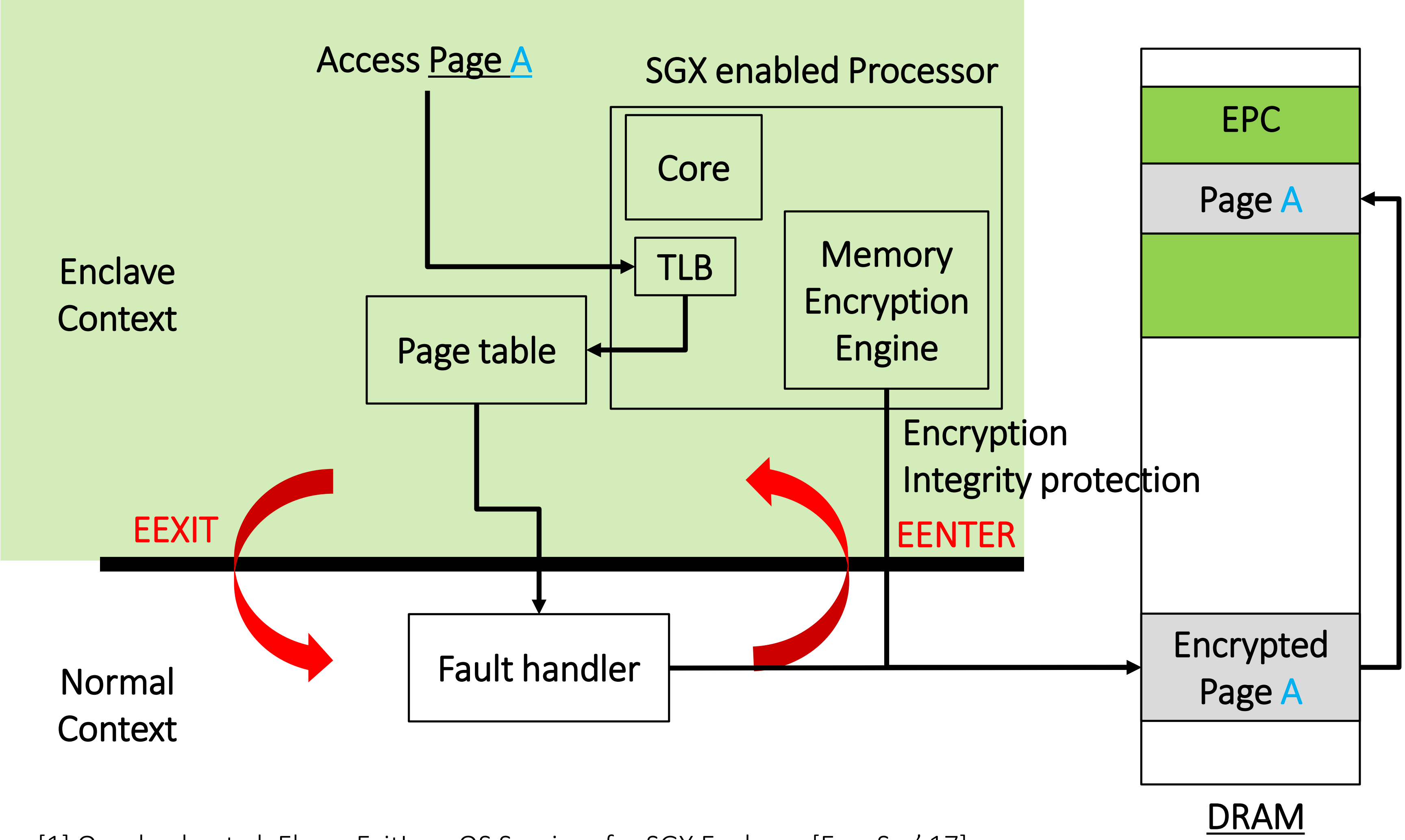
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



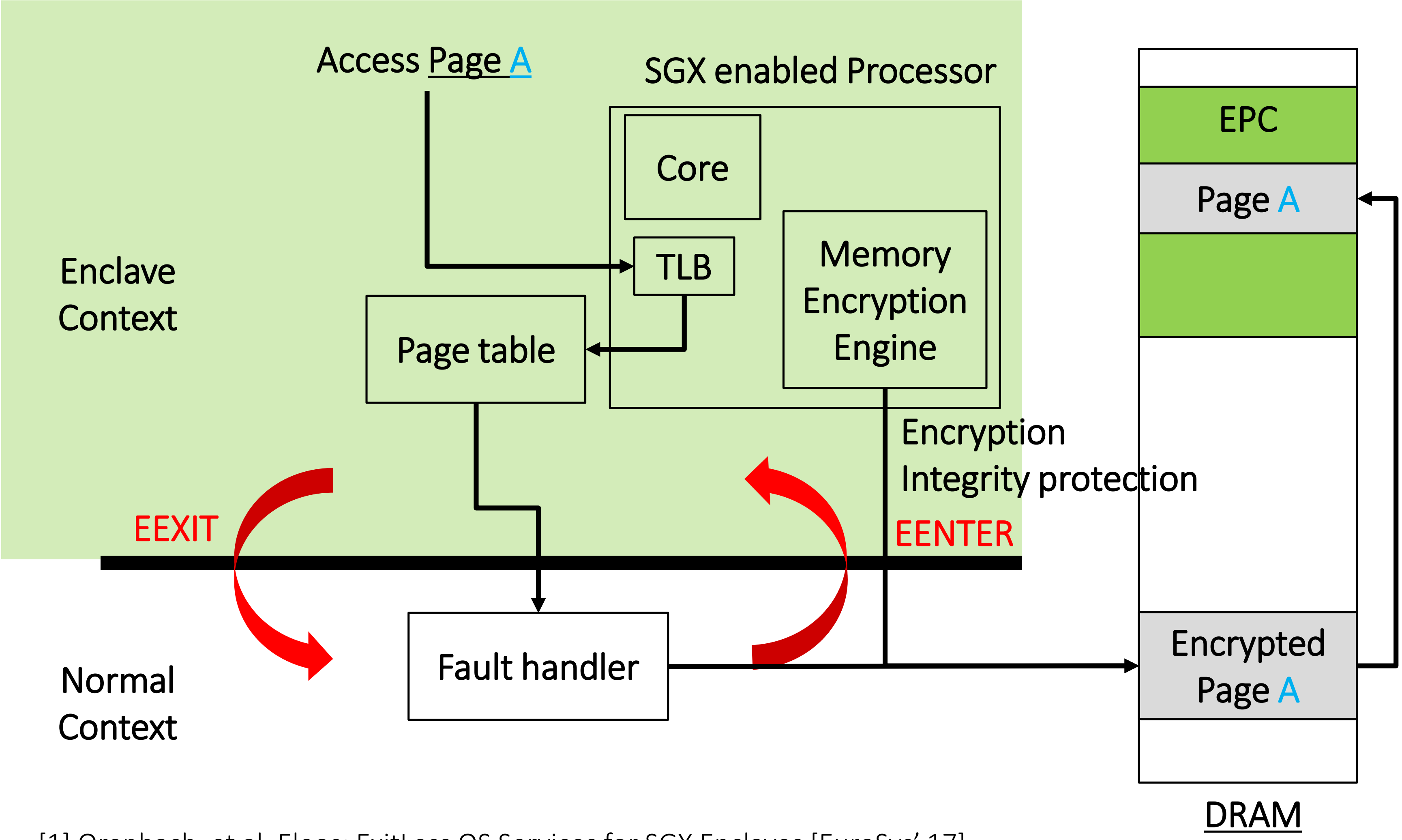
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



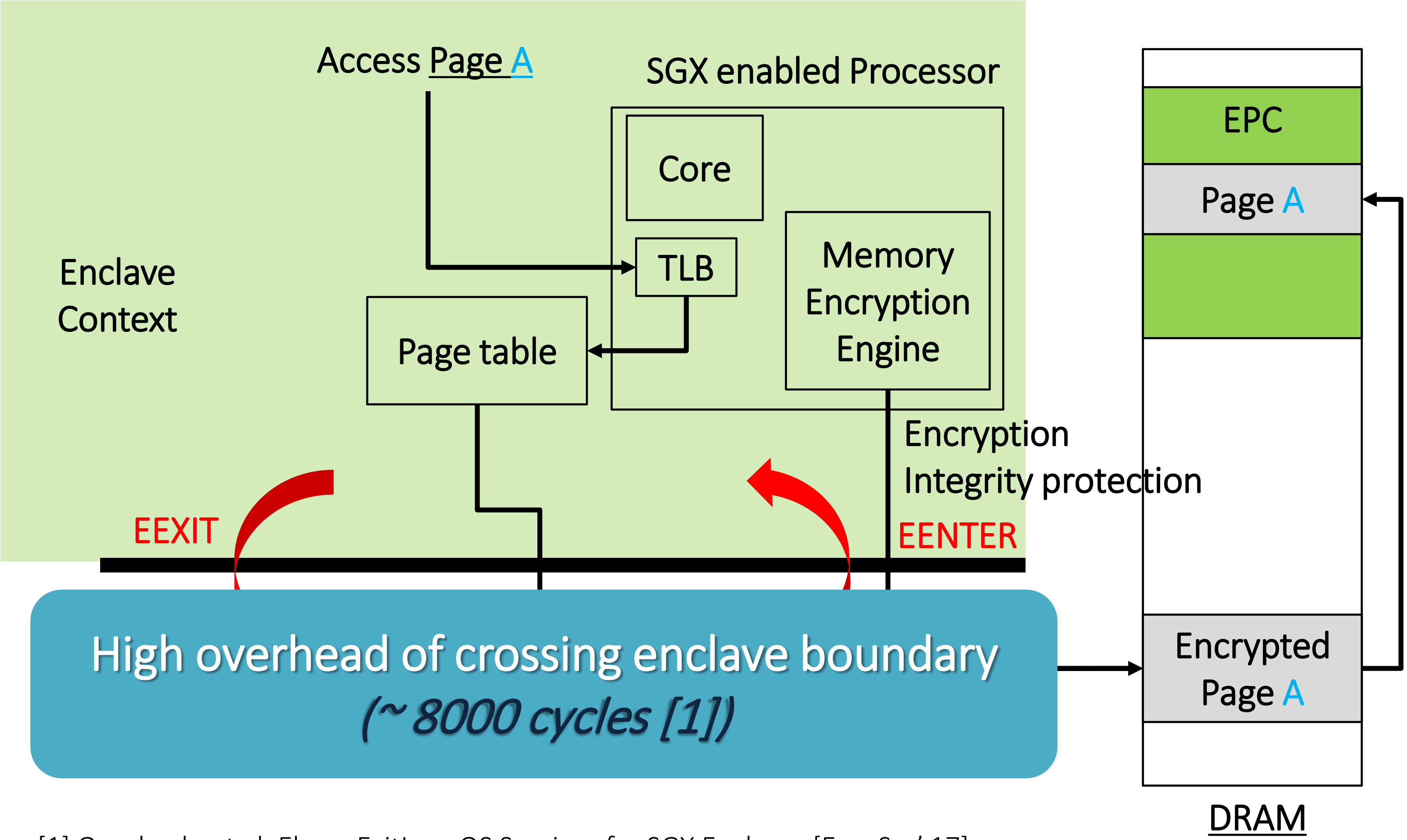
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX

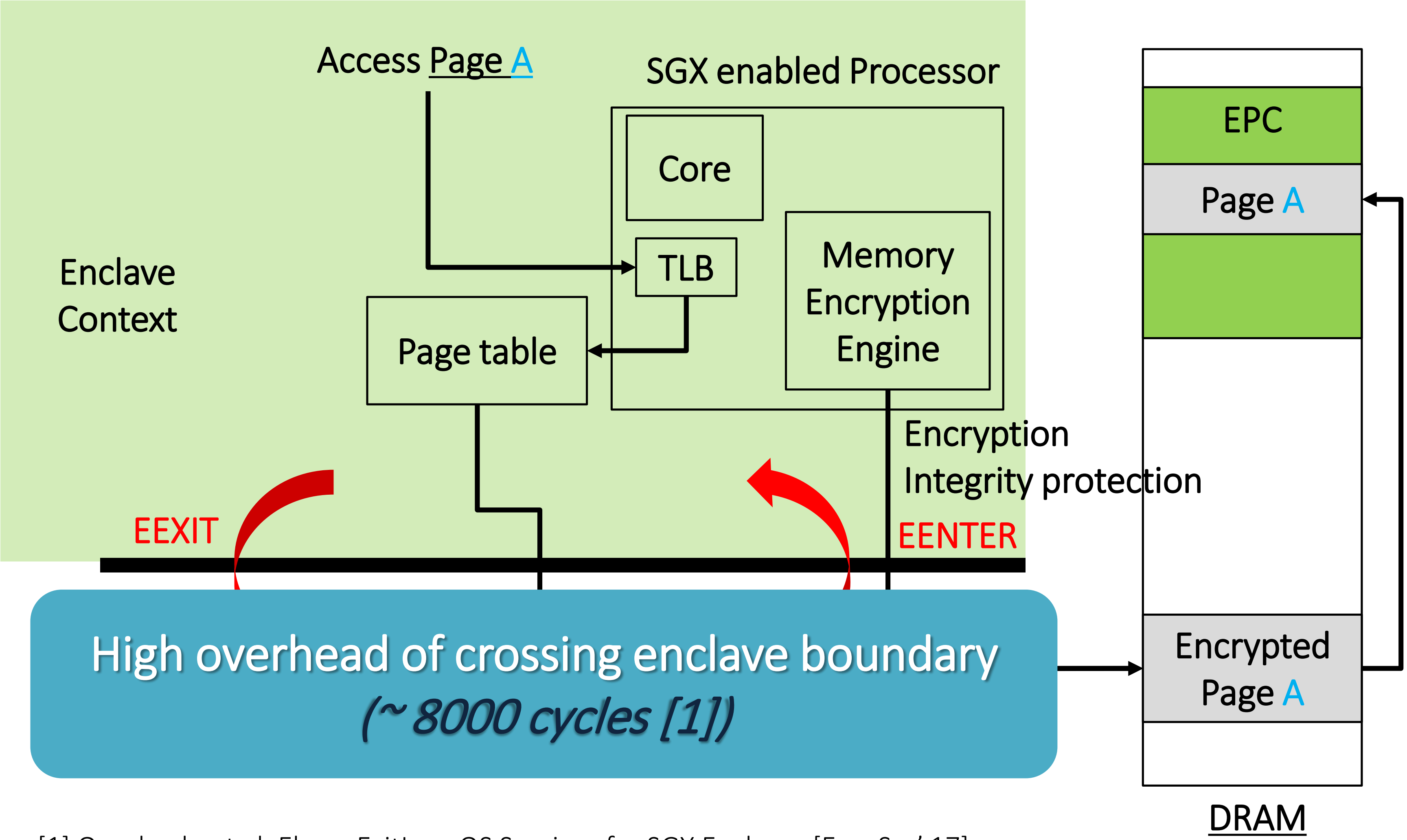


High overhead of crossing enclave boundary

(~ 8000 cycles [1])

[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

Paging Mechanism of SGX



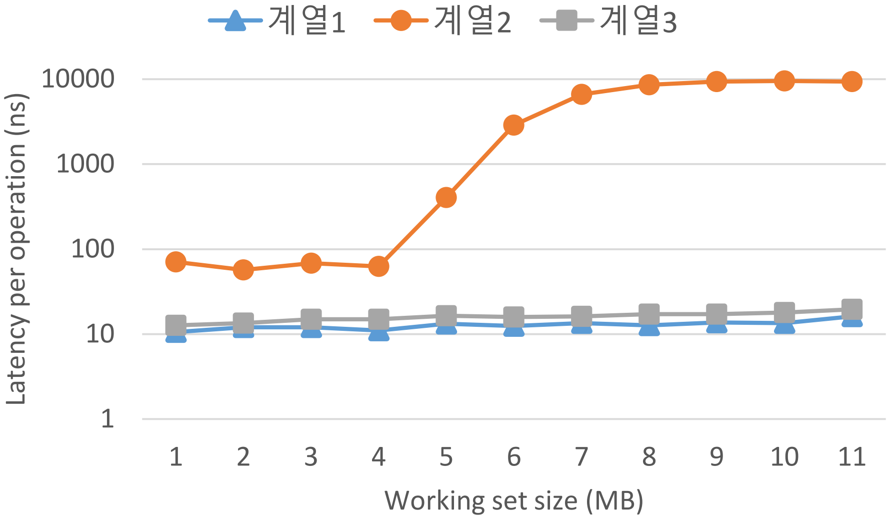
High overhead of crossing enclave boundary

(~8000 cycles [1])

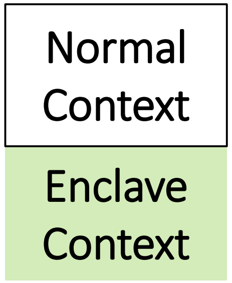
[1] Orenbach, et al. Eleos: ExitLess OS Services for SGX Enclaves [EuroSys' 17]

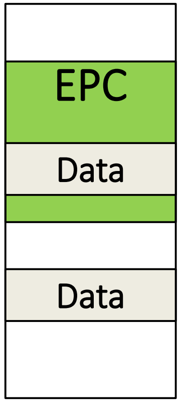
Observations

- Accessing untrusted memory incurs low overhead



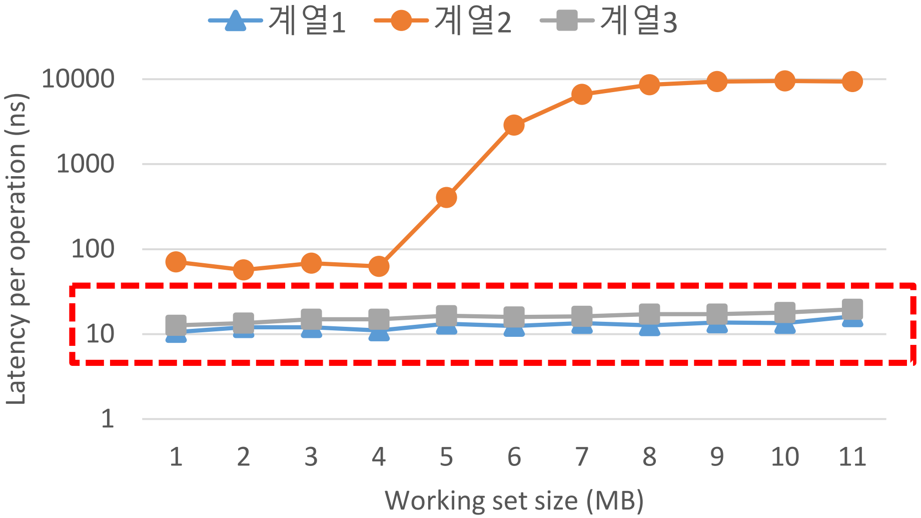
NoSGX
SGX_Enclave
SGX_Unprotected

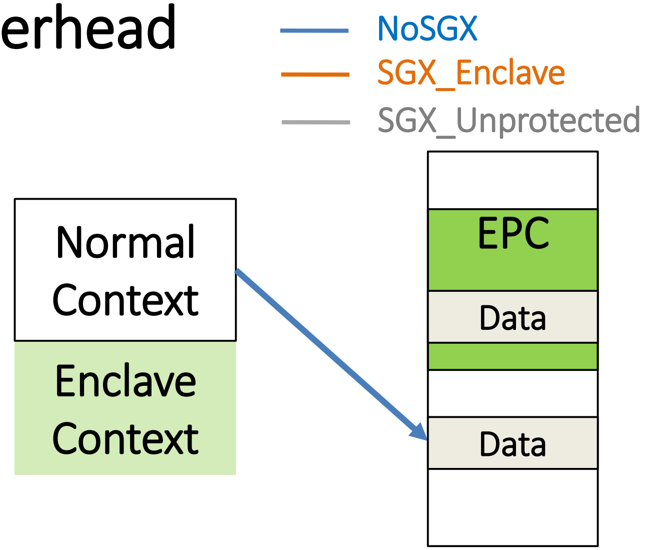


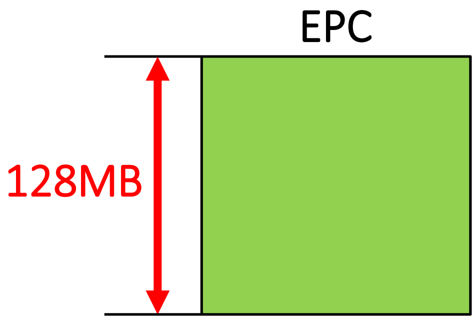


Observations

- Accessing untrusted memory incurs low overhead

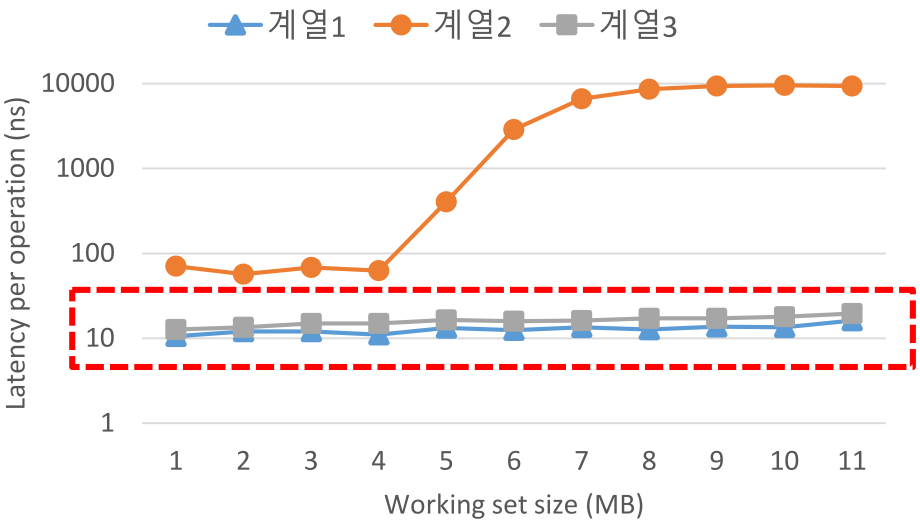


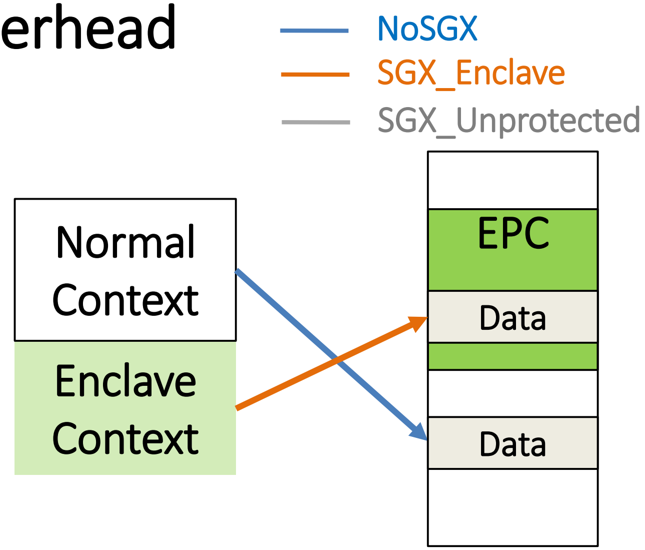


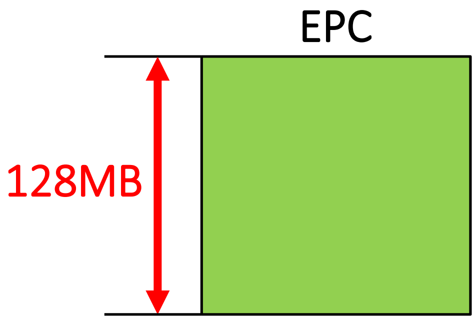


Observations

- Accessing untrusted memory incurs low overhead

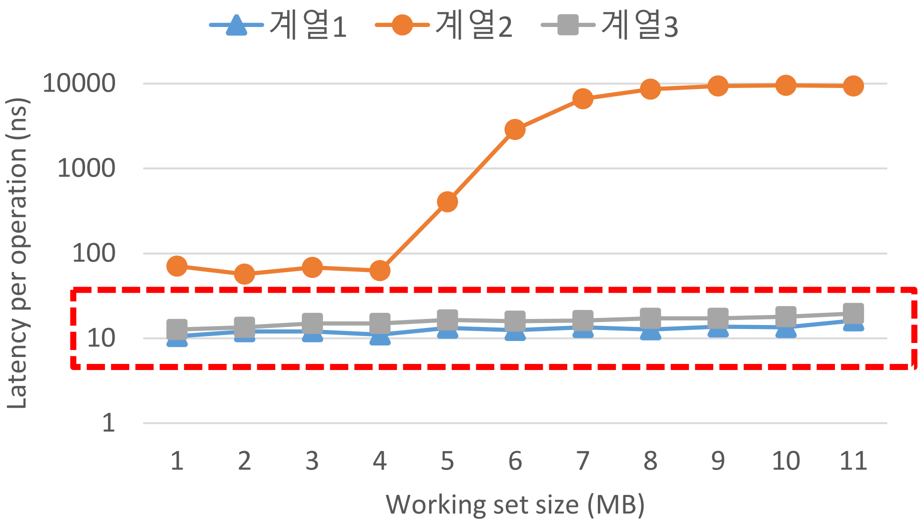


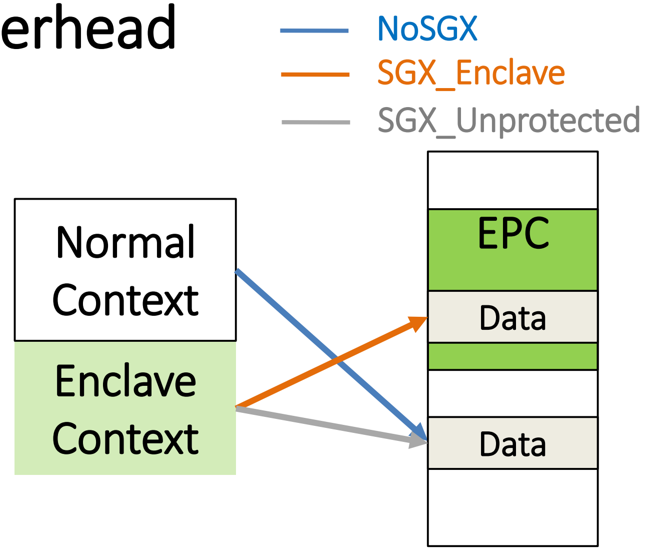


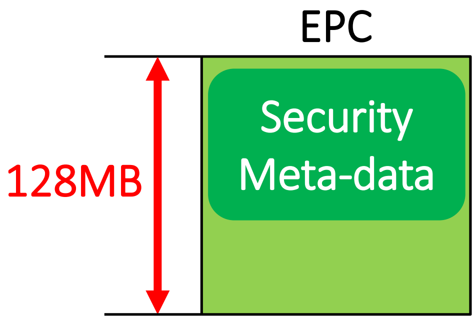


Observations

- Accessing untrusted memory incurs low overhead

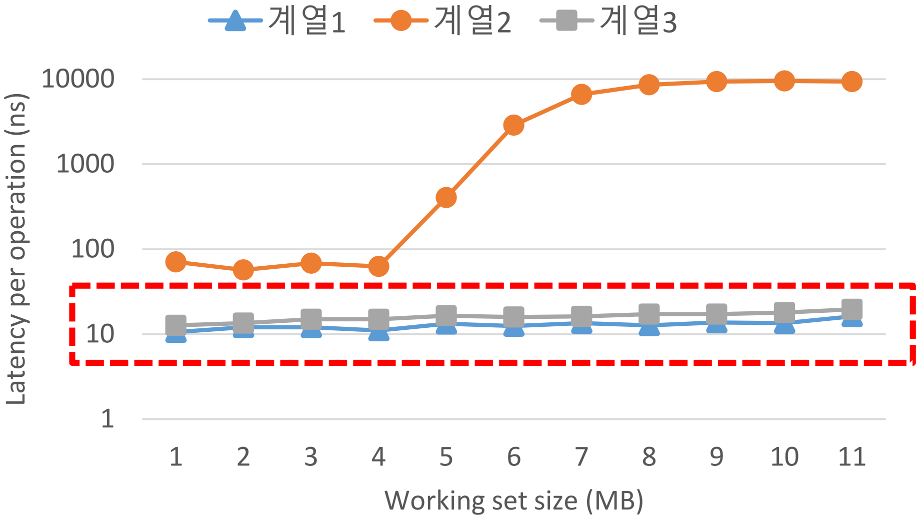




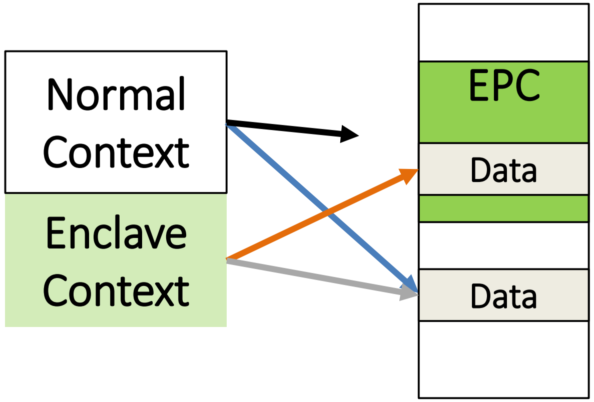


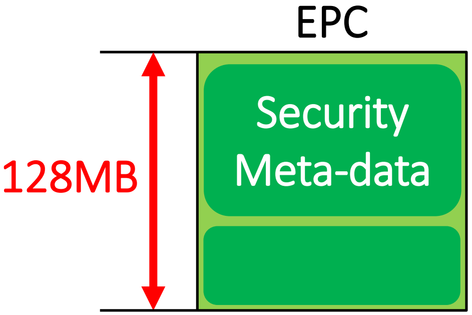
Observations

- Accessing untrusted memory incurs low overhead



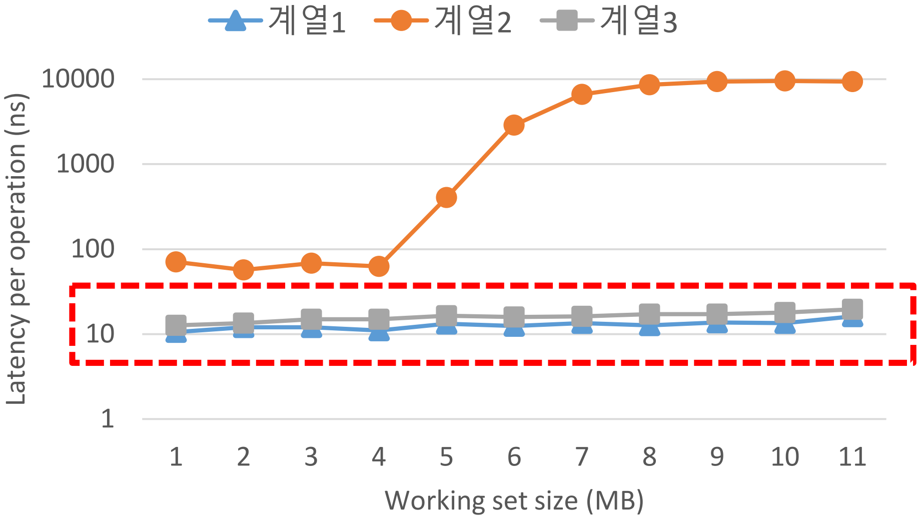
NoSGX
SGX_Enclave
SGX_Unprotected

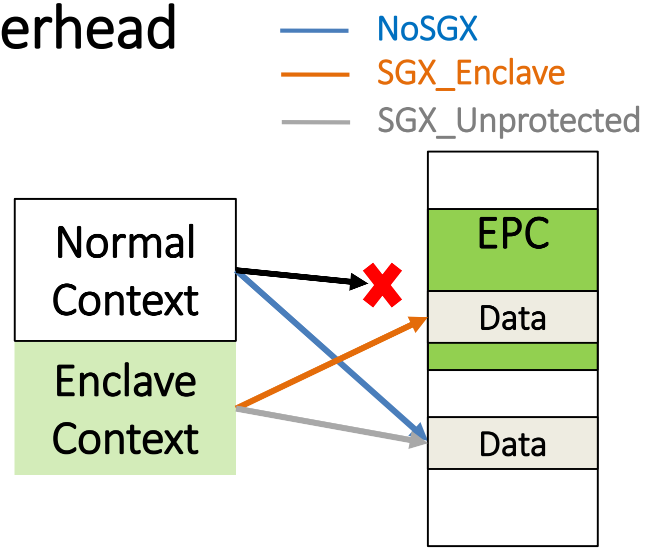


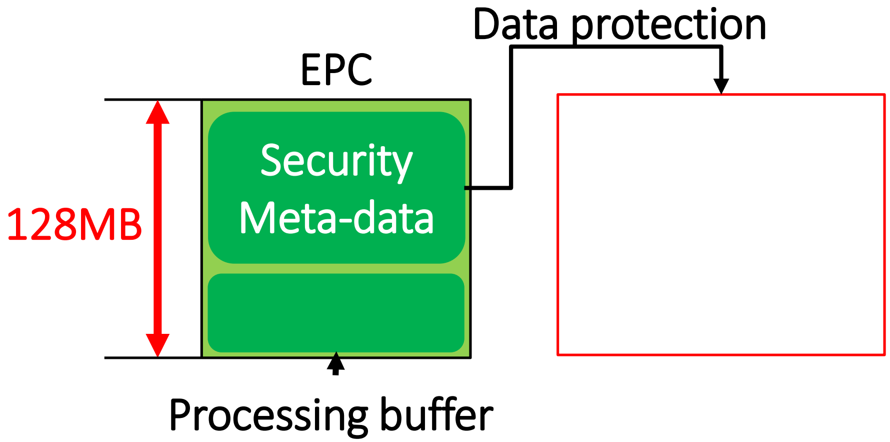


Observations

- Accessing untrusted memory incurs low overhead

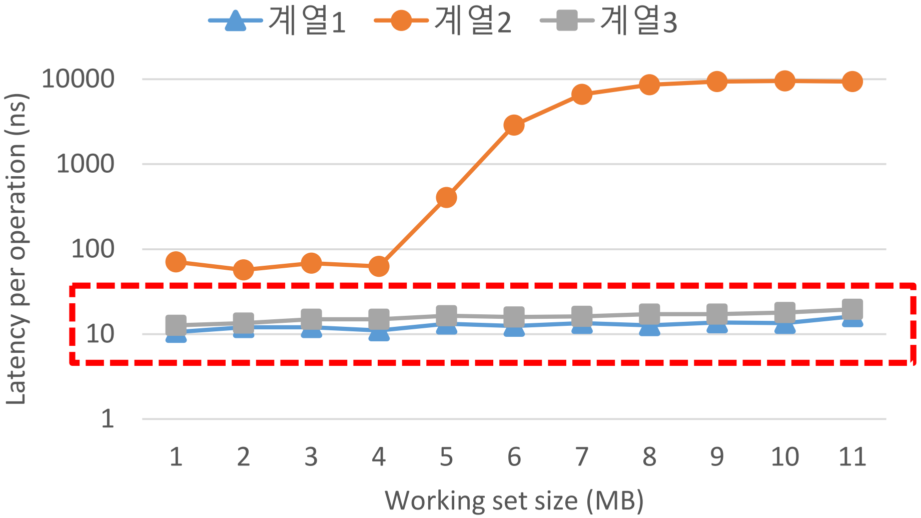


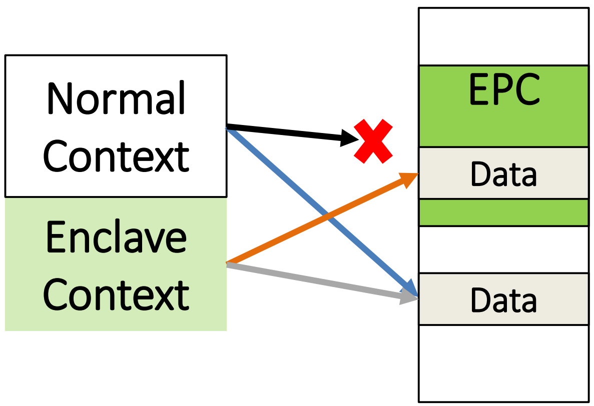


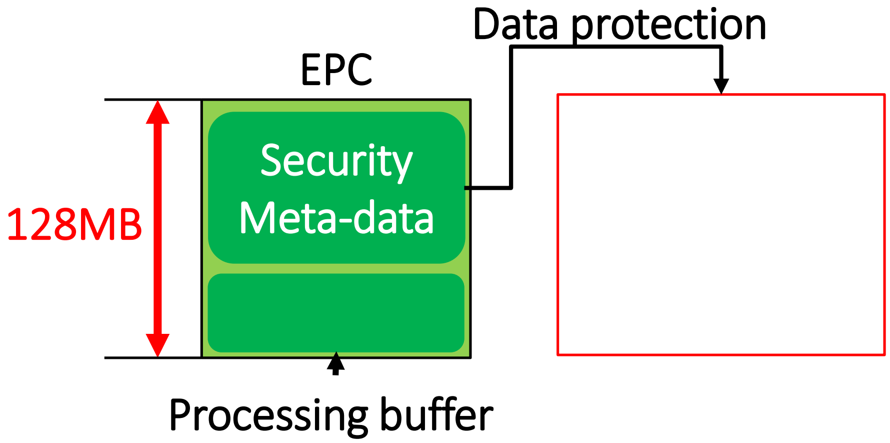


Observations

- Accessing untrusted memory incurs low overhead

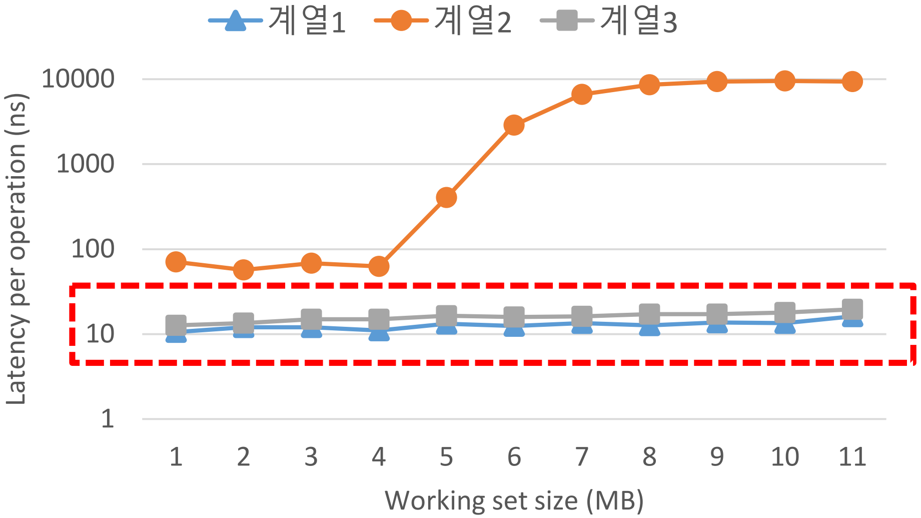


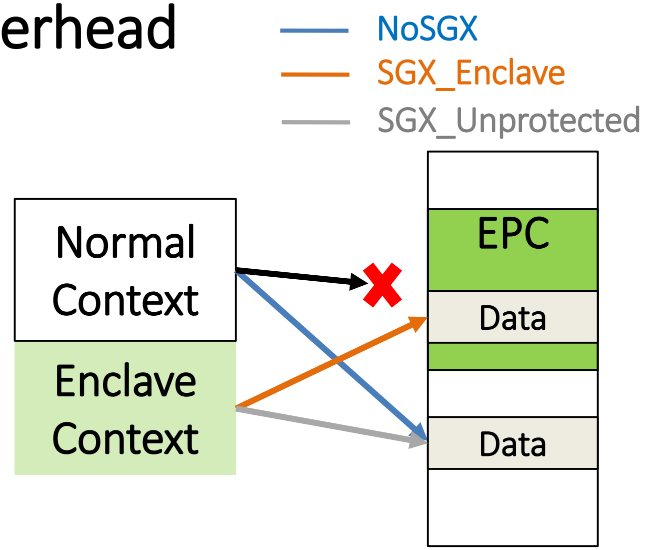


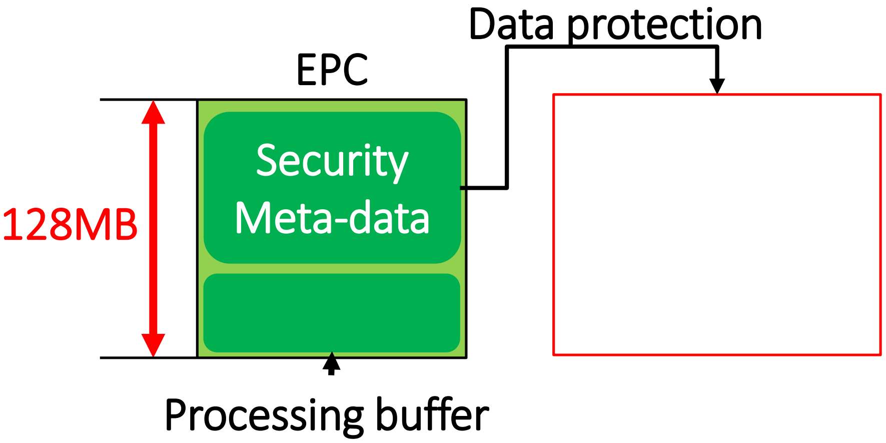


Observations

- Accessing untrusted memory incurs low overhead

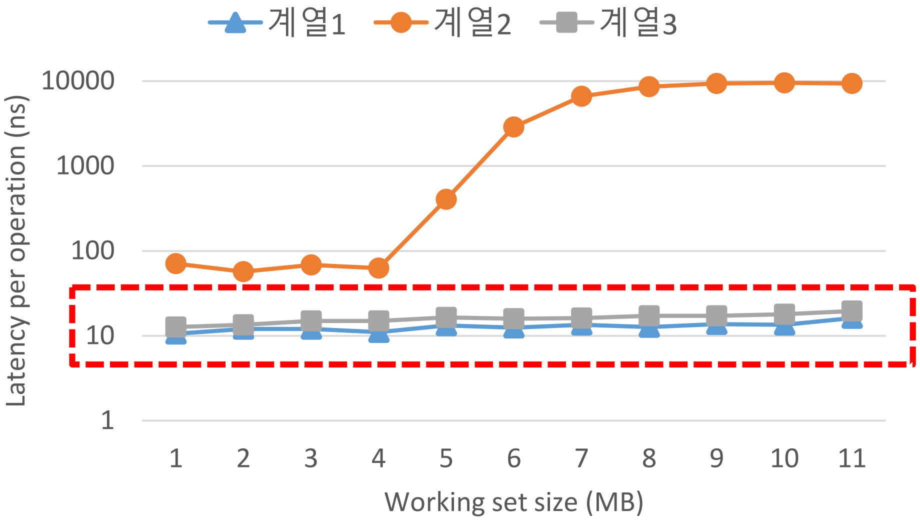


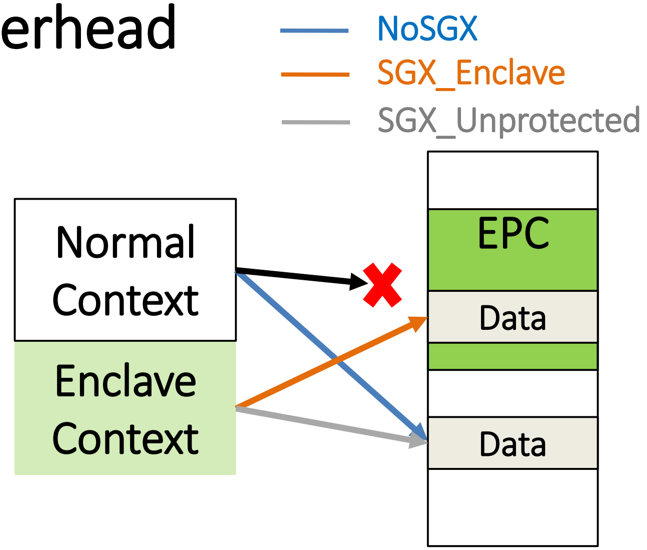


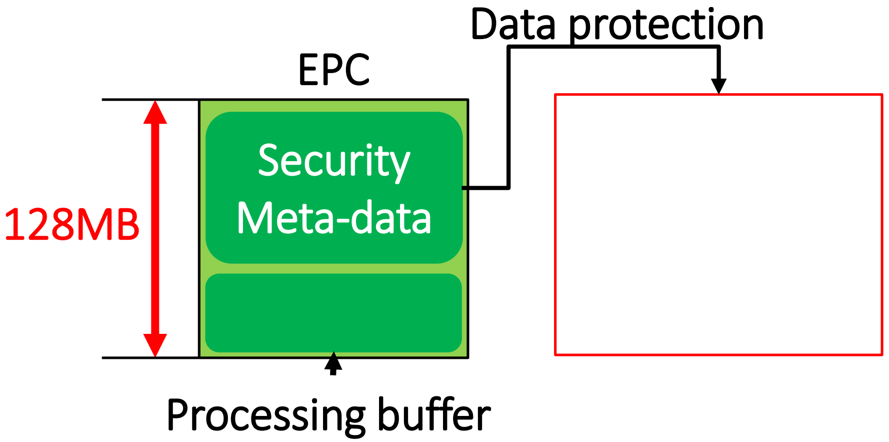


Observations

- Accessing untrusted memory incurs low overhead

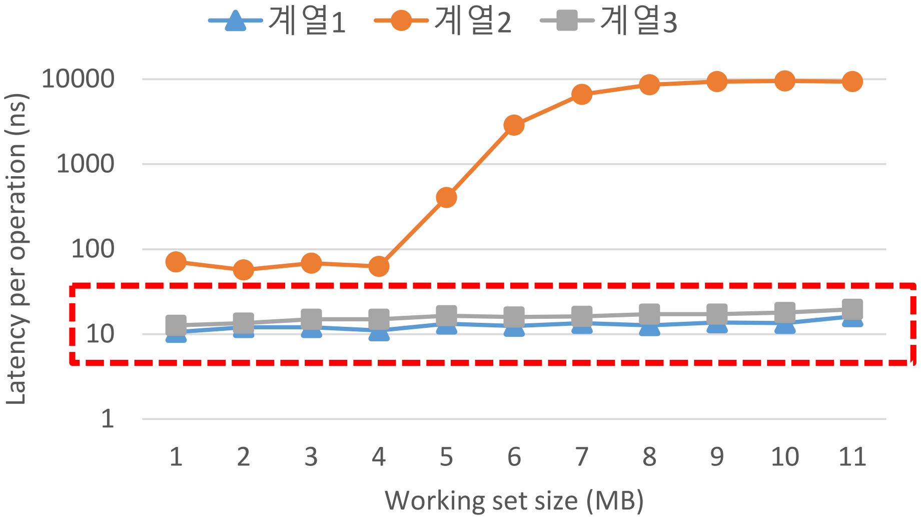


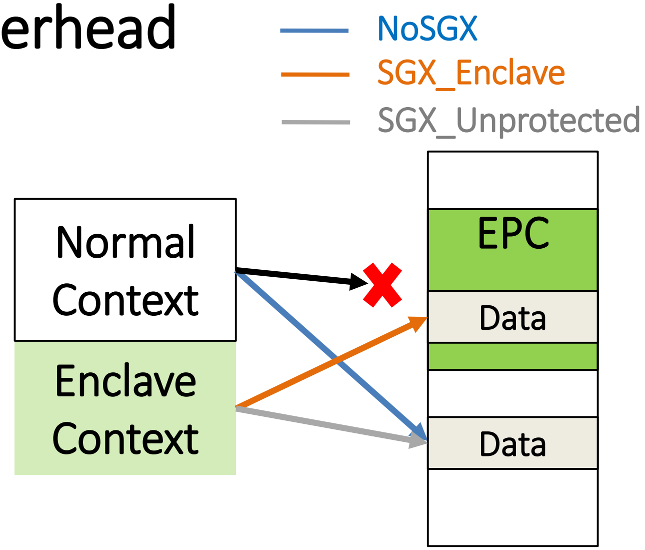


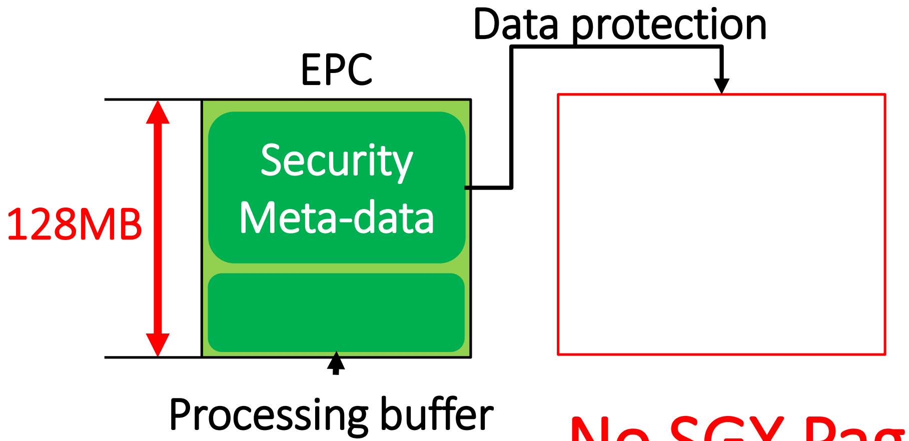


Observations

- Accessing untrusted memory incurs low overhead



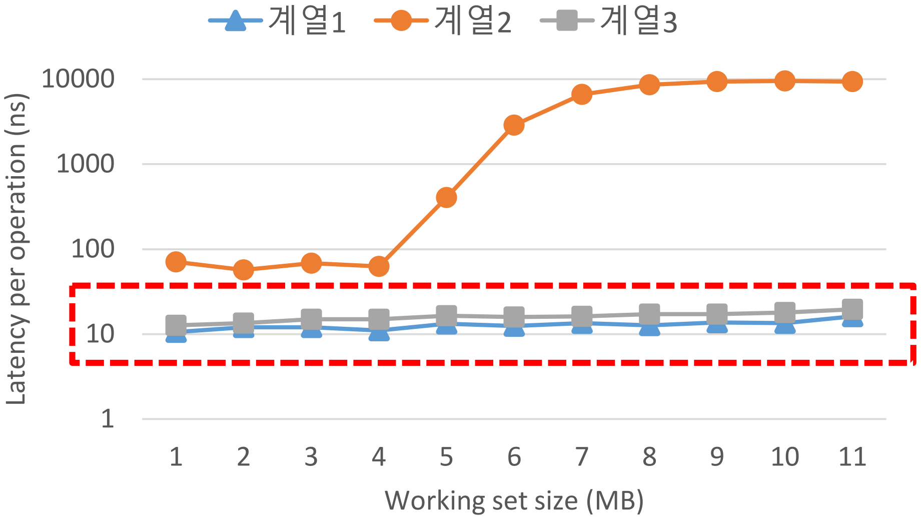


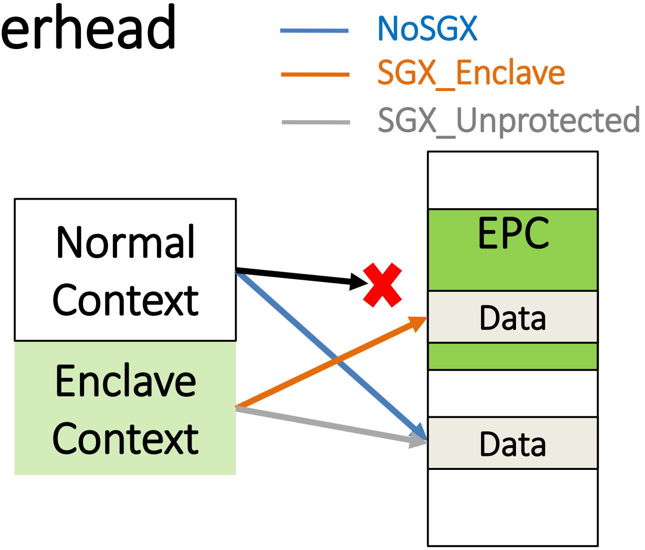


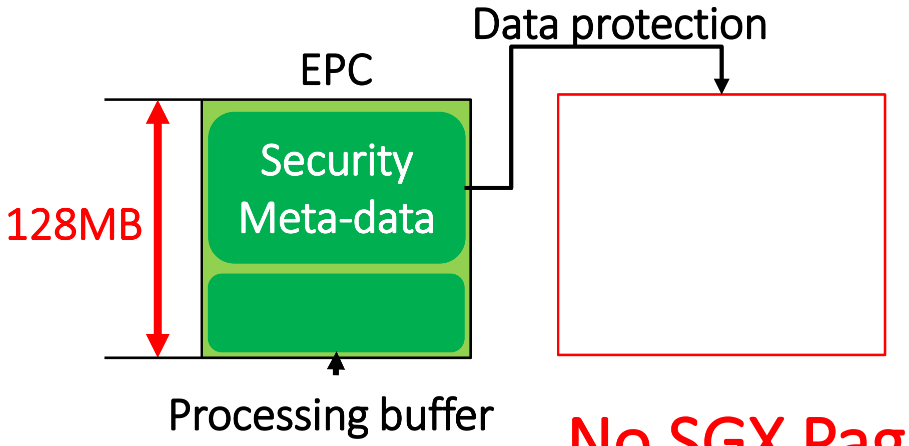
No SGX Paging!

Observations

- Accessing untrusted memory incurs low overhead



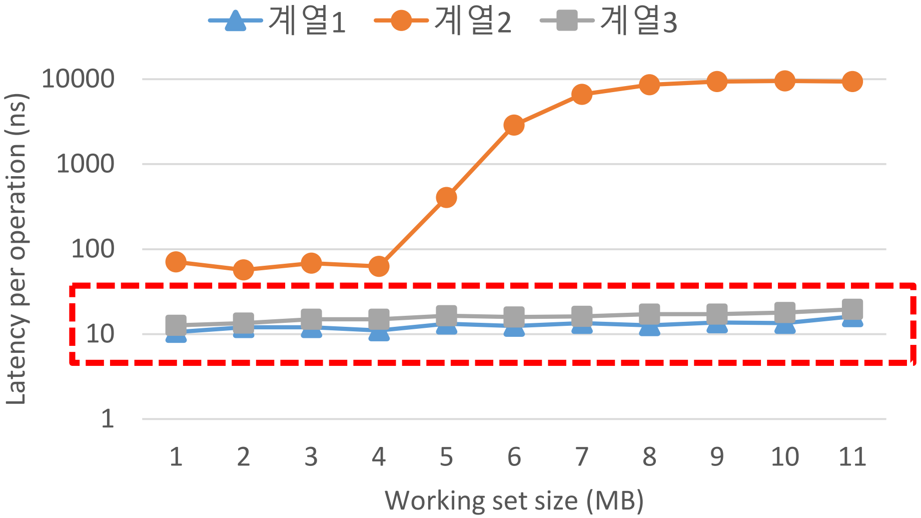


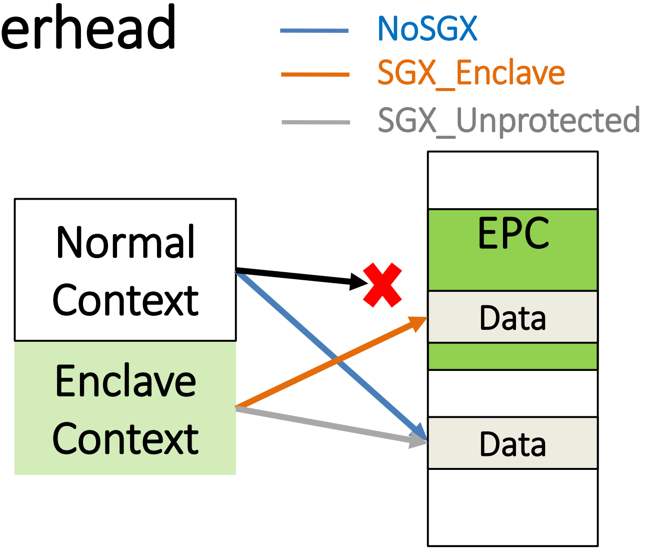


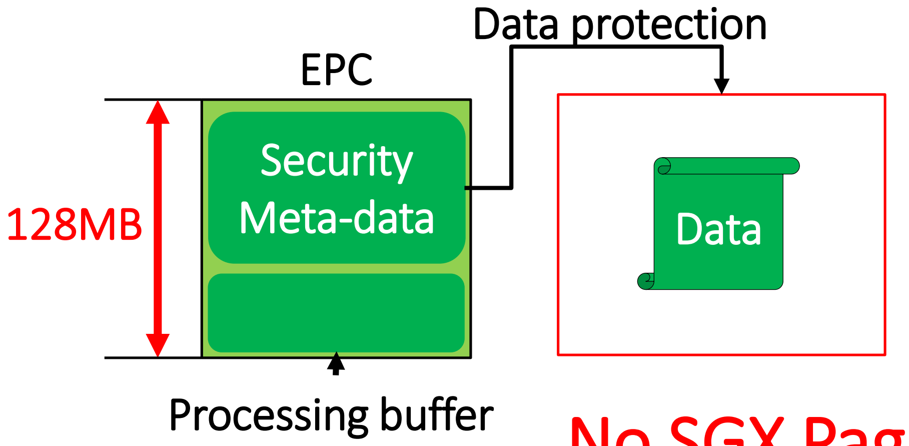
No SGX Paging!

Observations

- Accessing untrusted memory incurs low overhead



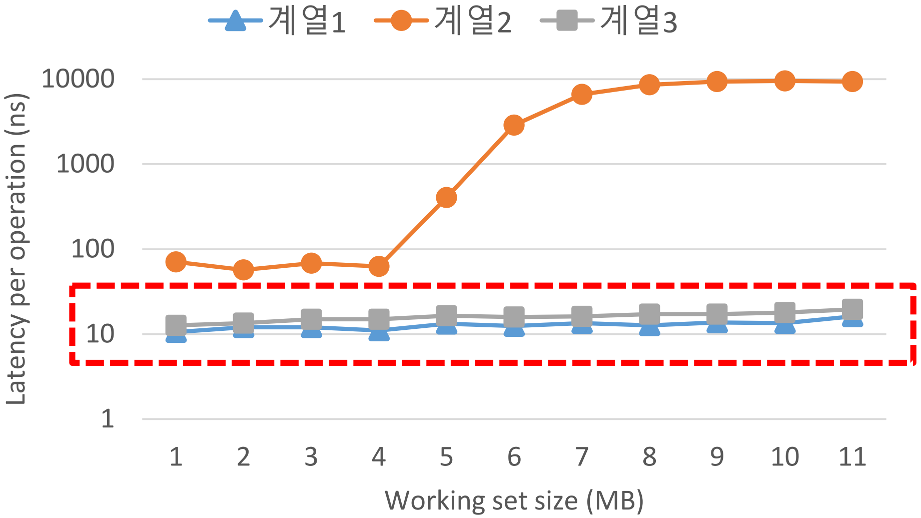


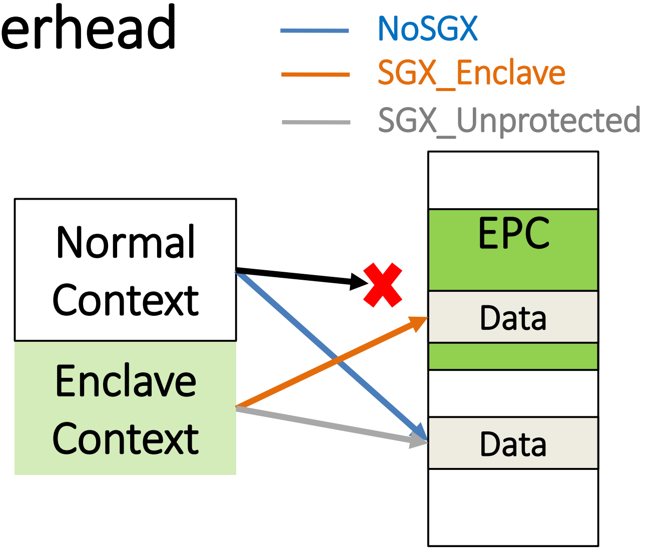


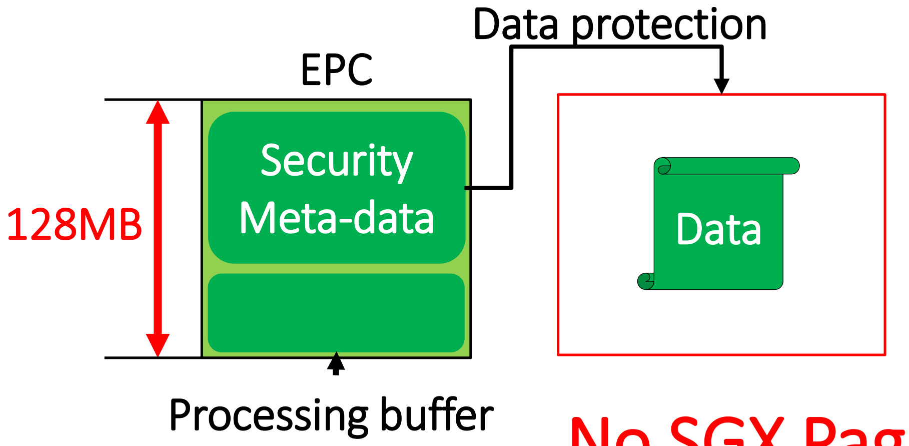
No SGX Paging!

Observations

- Accessing untrusted memory incurs low overhead



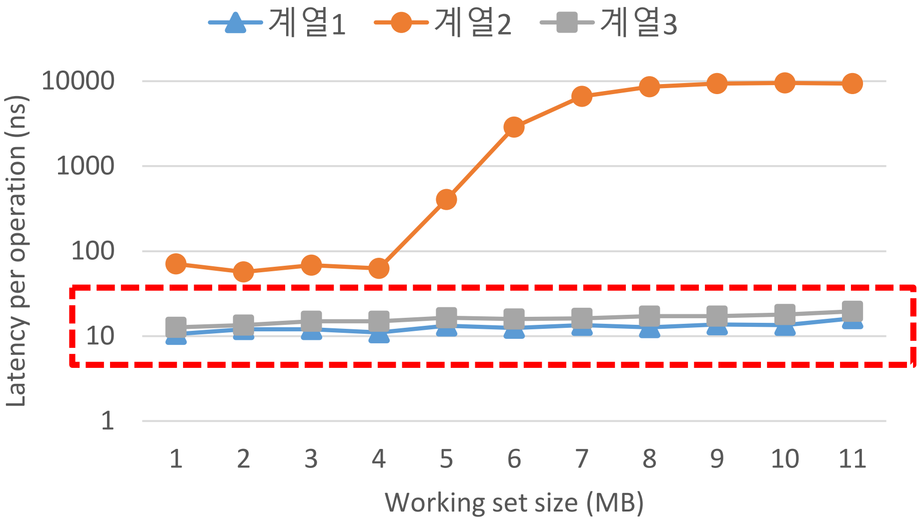


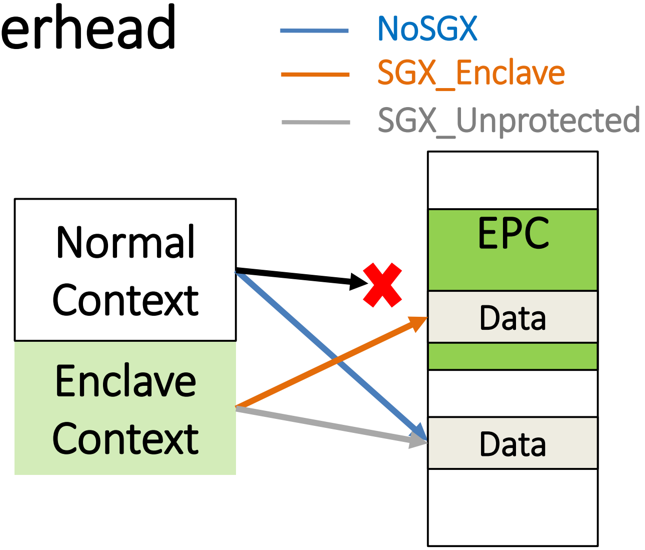


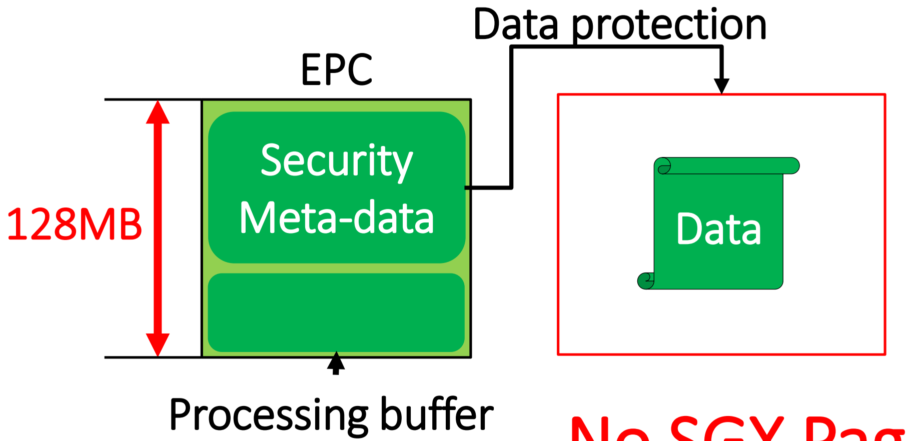
No SGX Paging!

Observations

- Accessing untrusted memory incurs low overhead



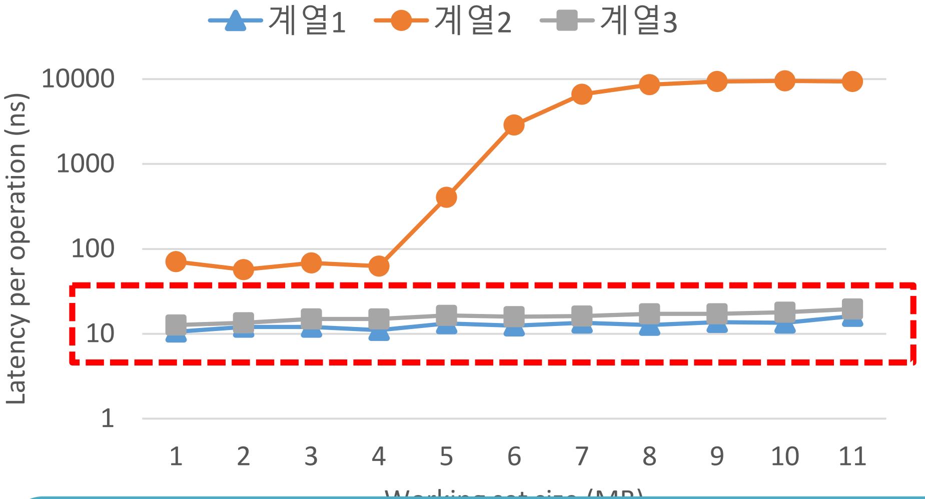


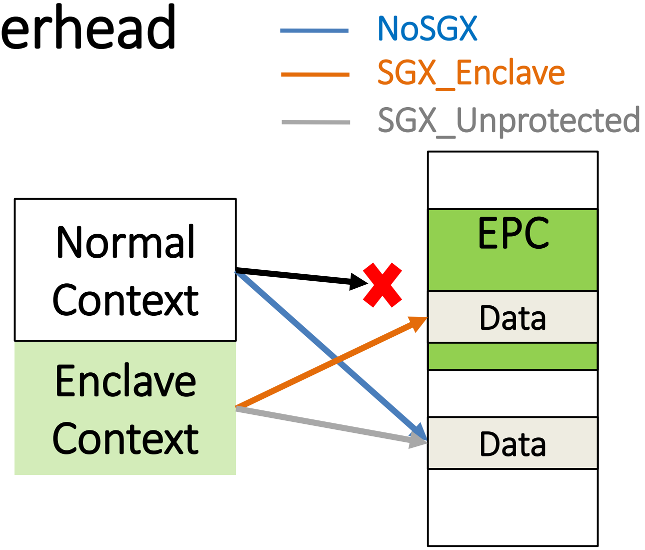


No SGX Paging!

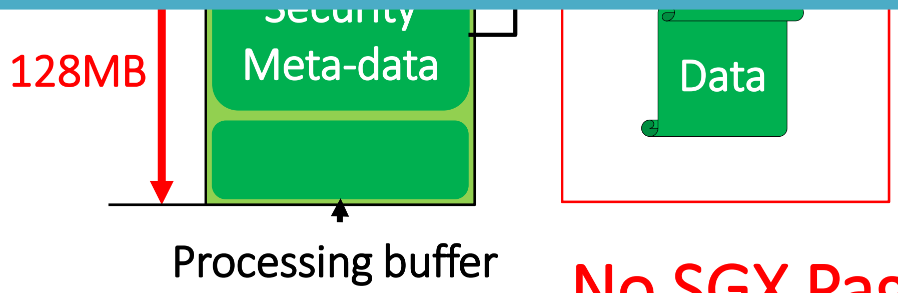
Observations

- Accessing untrusted memory incurs low overhead





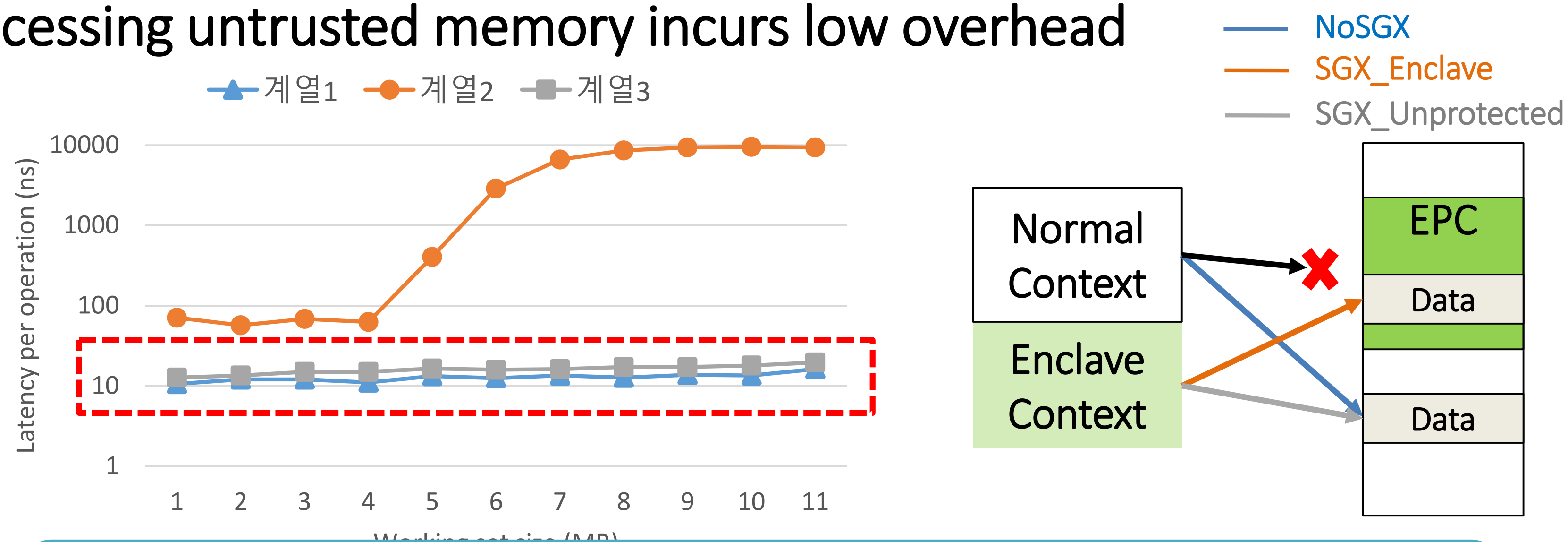
- *Reduce sgx-paging!*
  - Use protected memory as a secure processing buffer



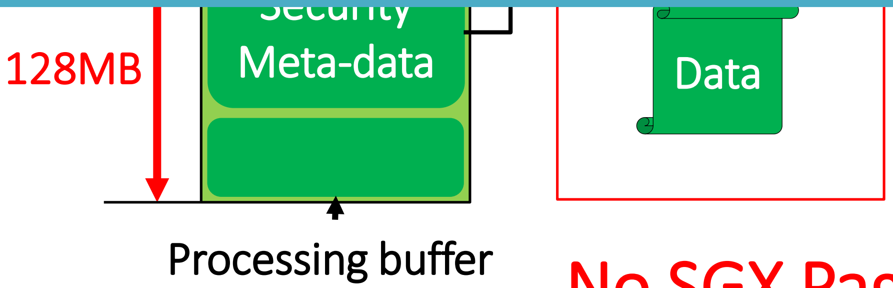
No SGX Paging!

Observations

- Accessing untrusted memory incurs low overhead



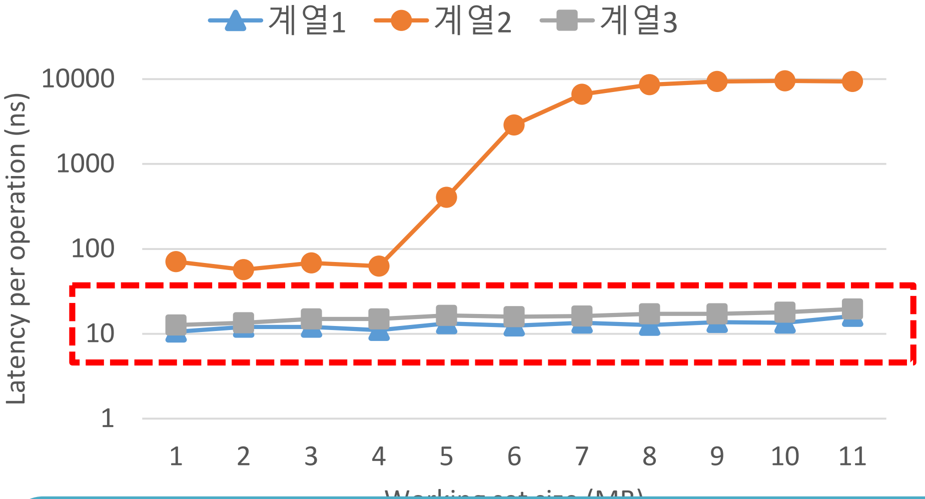
- *Reduce sgx-paging!*
  - Use protected memory as a secure processing buffer

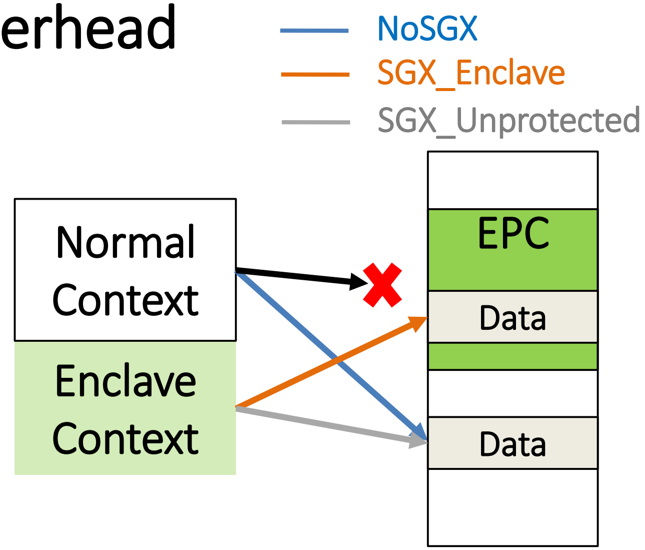


No SGX Paging!

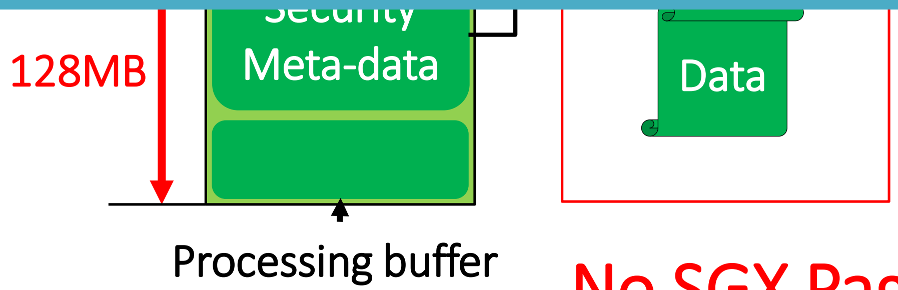
Observations

- Accessing untrusted memory incurs low overhead



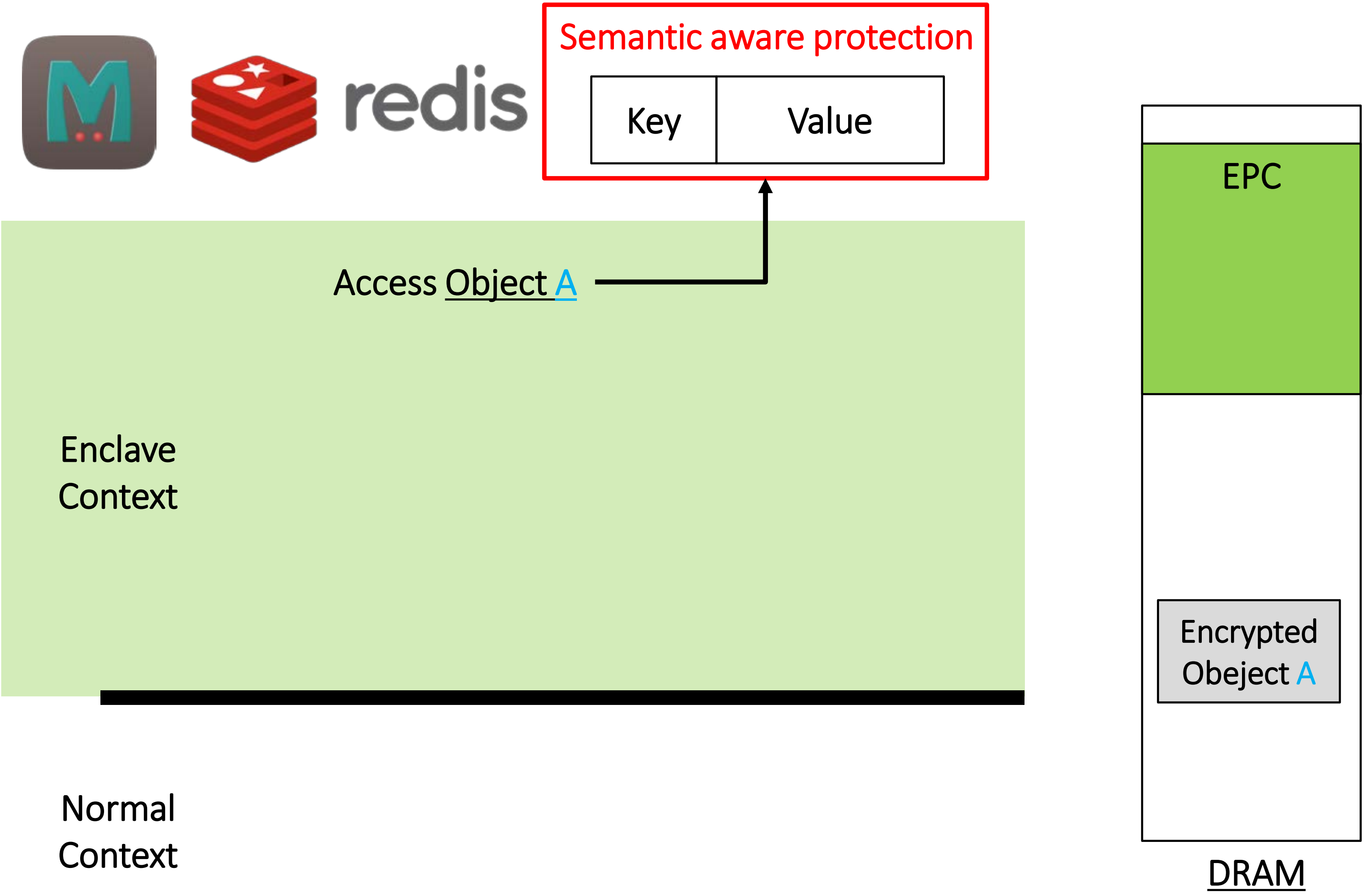


- *Reduce sgx-paging!*
  - Use protected memory as a secure processing buffer



No SGX Paging!

Proposed Design: Semantic Aware Protection



redis

Semantic aware protection

Key

Value

Access Object A

Enclave Context

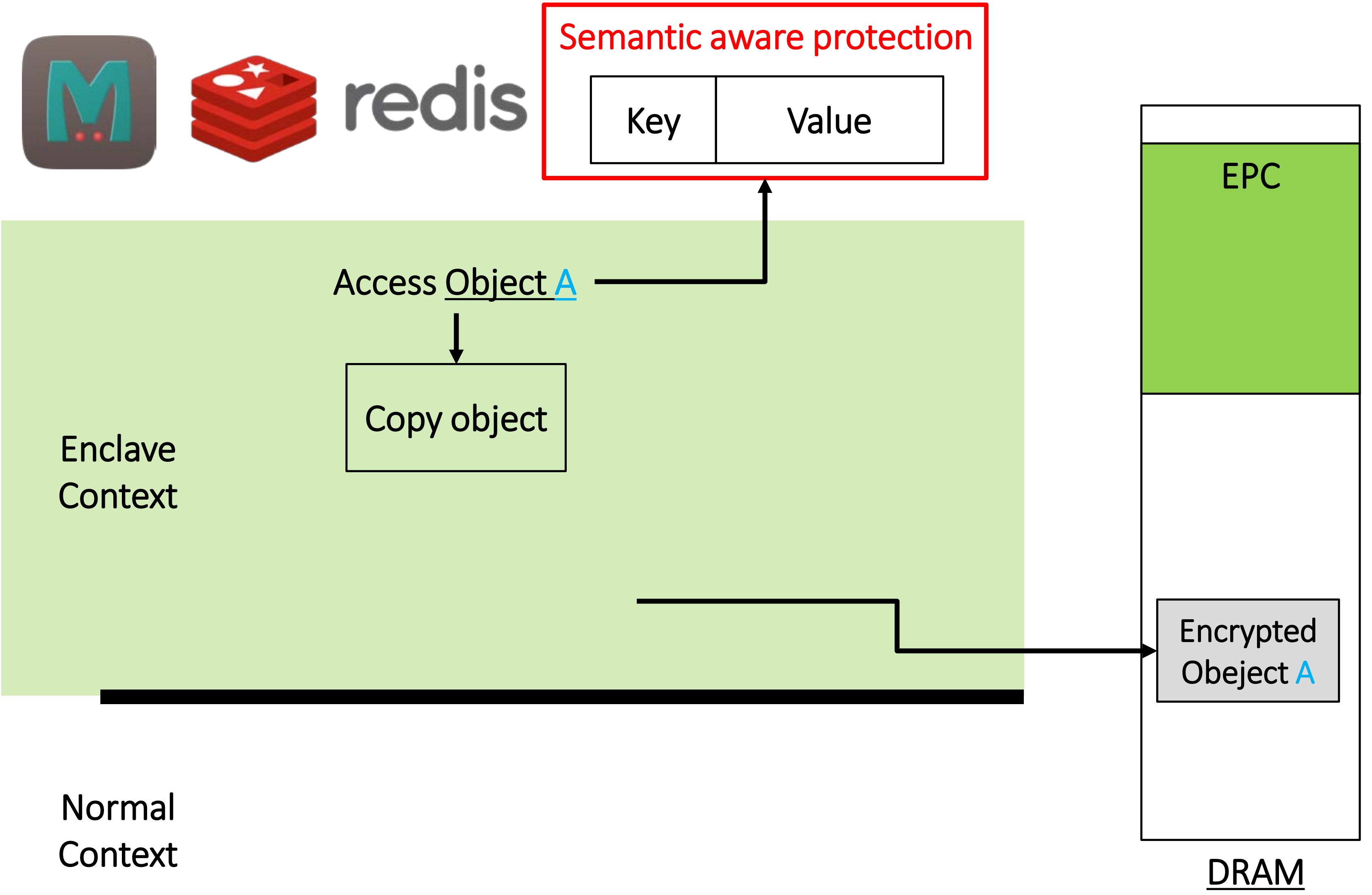
Normal Context

EPC

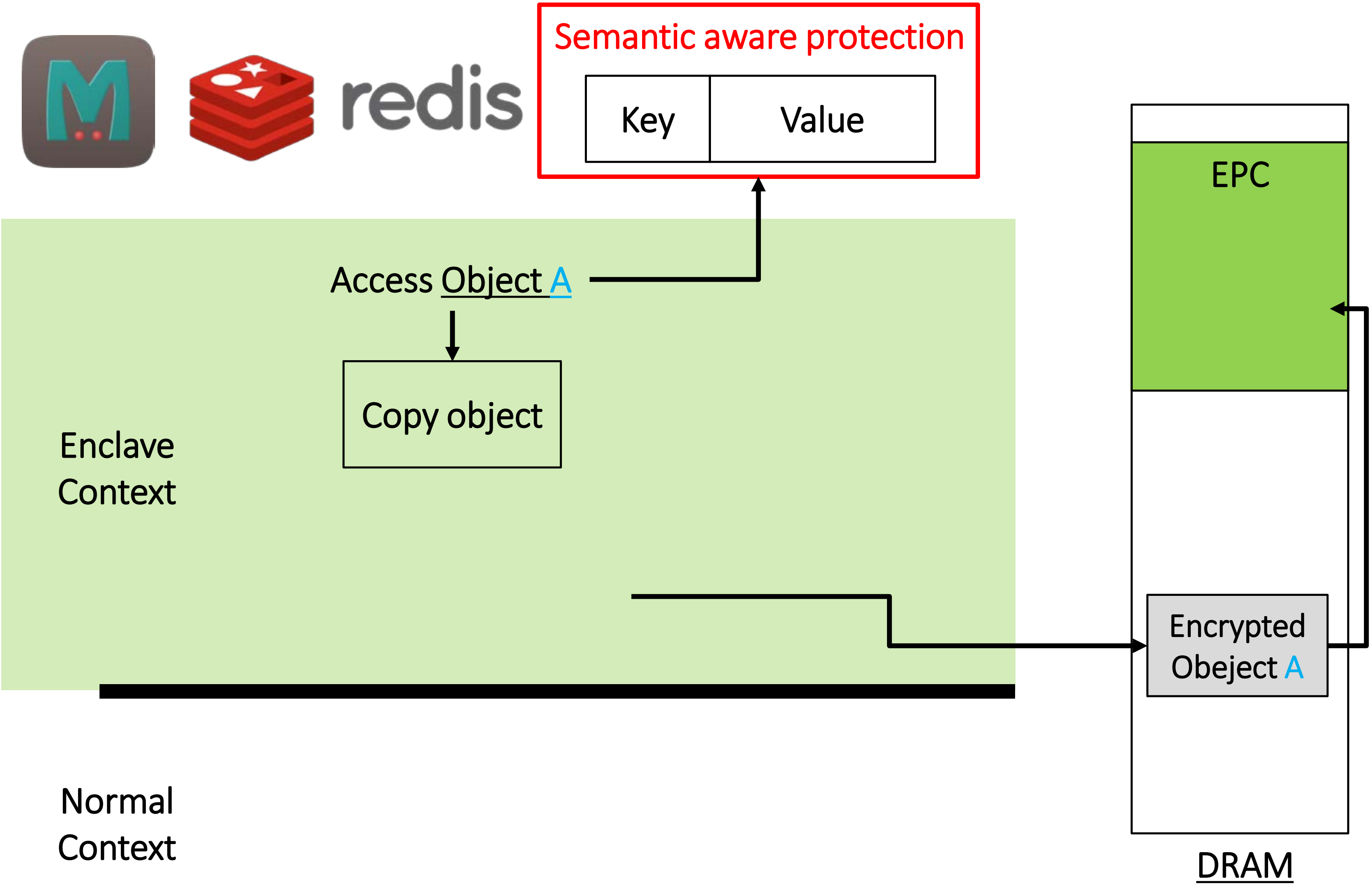
Encrypted Obeject A

DRAM

Proposed Design: Semantic Aware Protection



Proposed Design: Semantic Aware Protection



Semantic aware protection

Key

Value

Access Object A

Copy object

Enclave Context

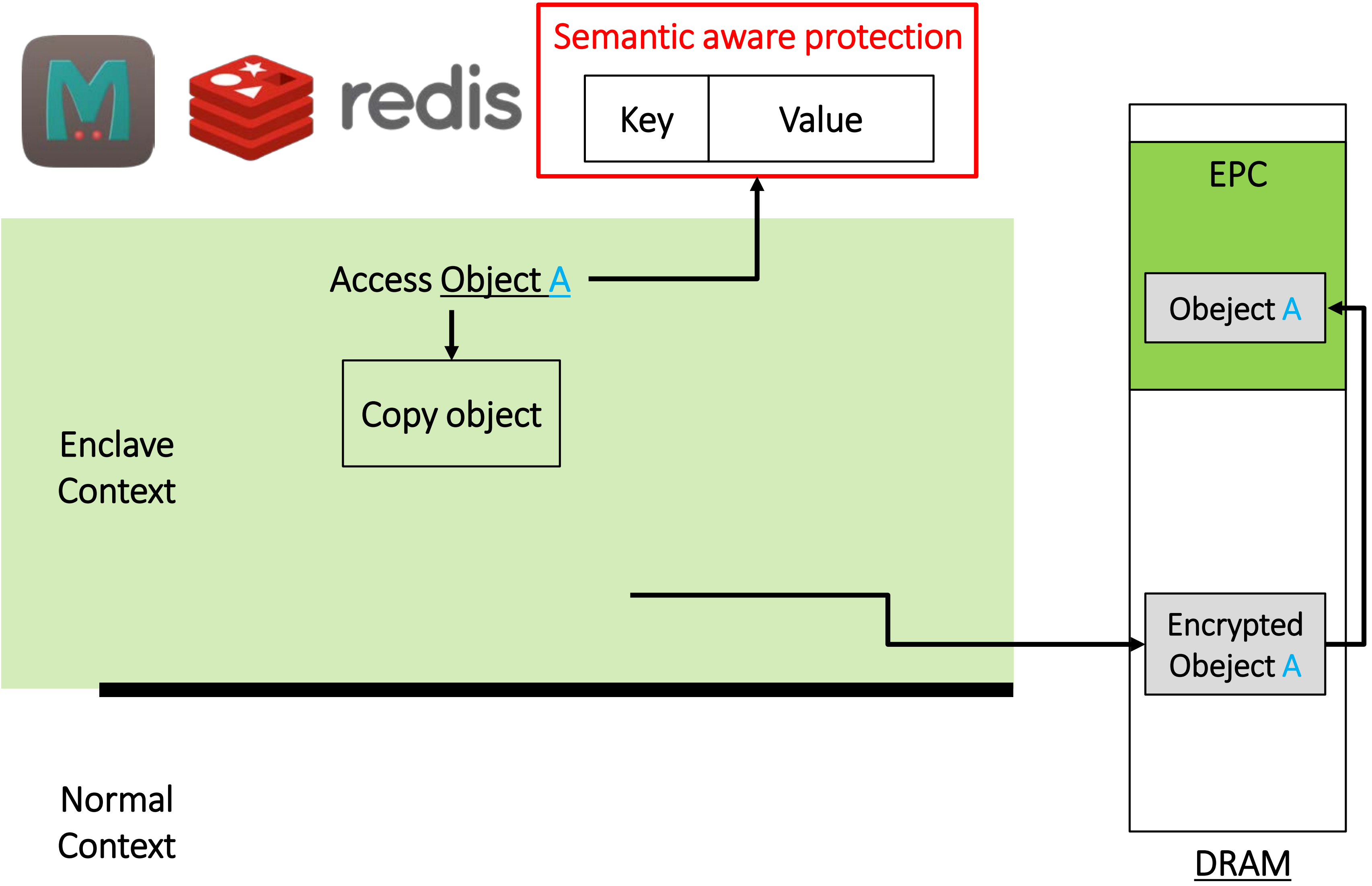
Normal Context

EPC

Encrypted Obeject A

DRAM

Proposed Design: Semantic Aware Protection



Semantic aware protection

Key	Value

Access Object A

Copy object

Enclave Context

Normal Context

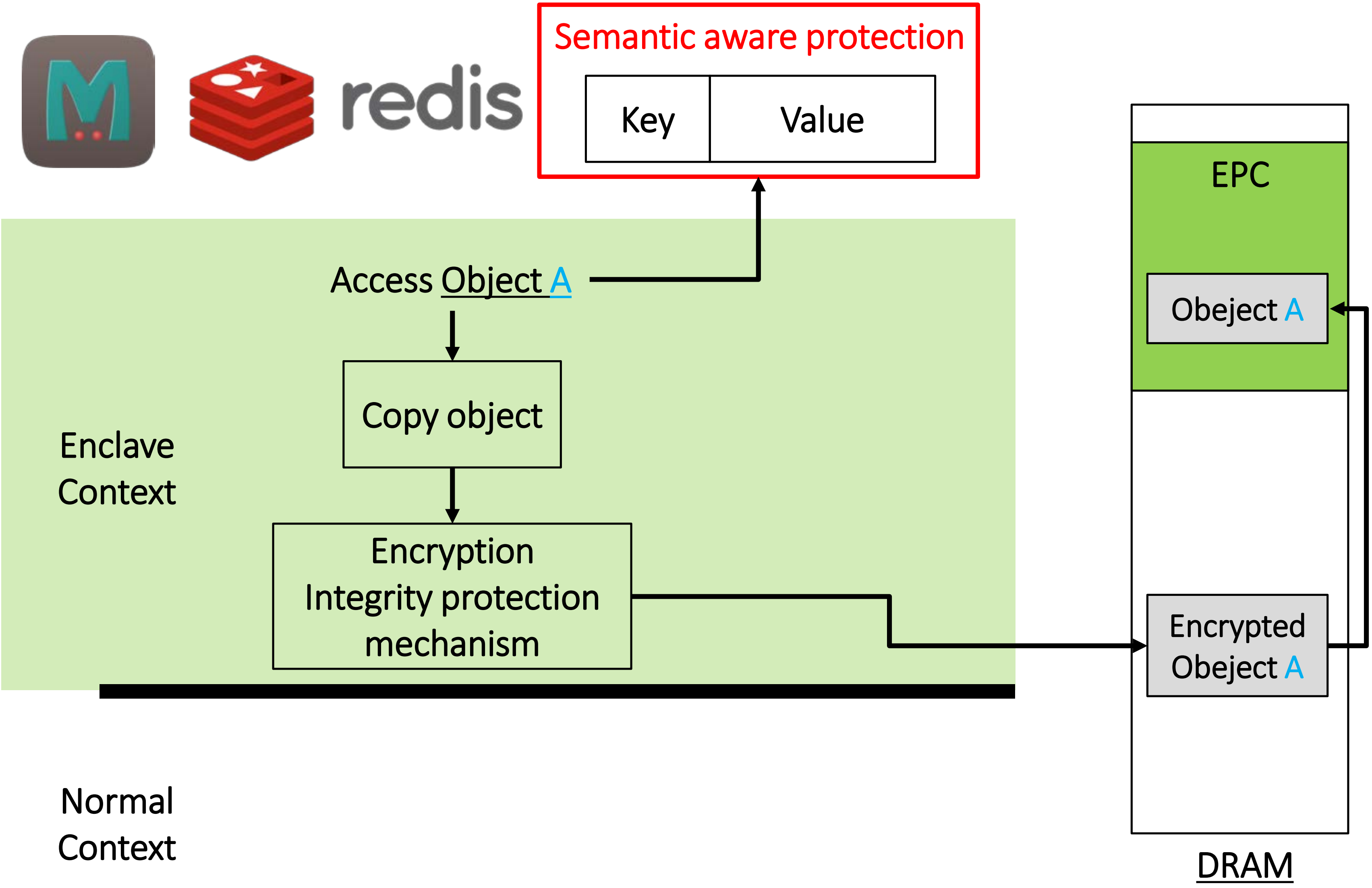
EPC

Obeject A

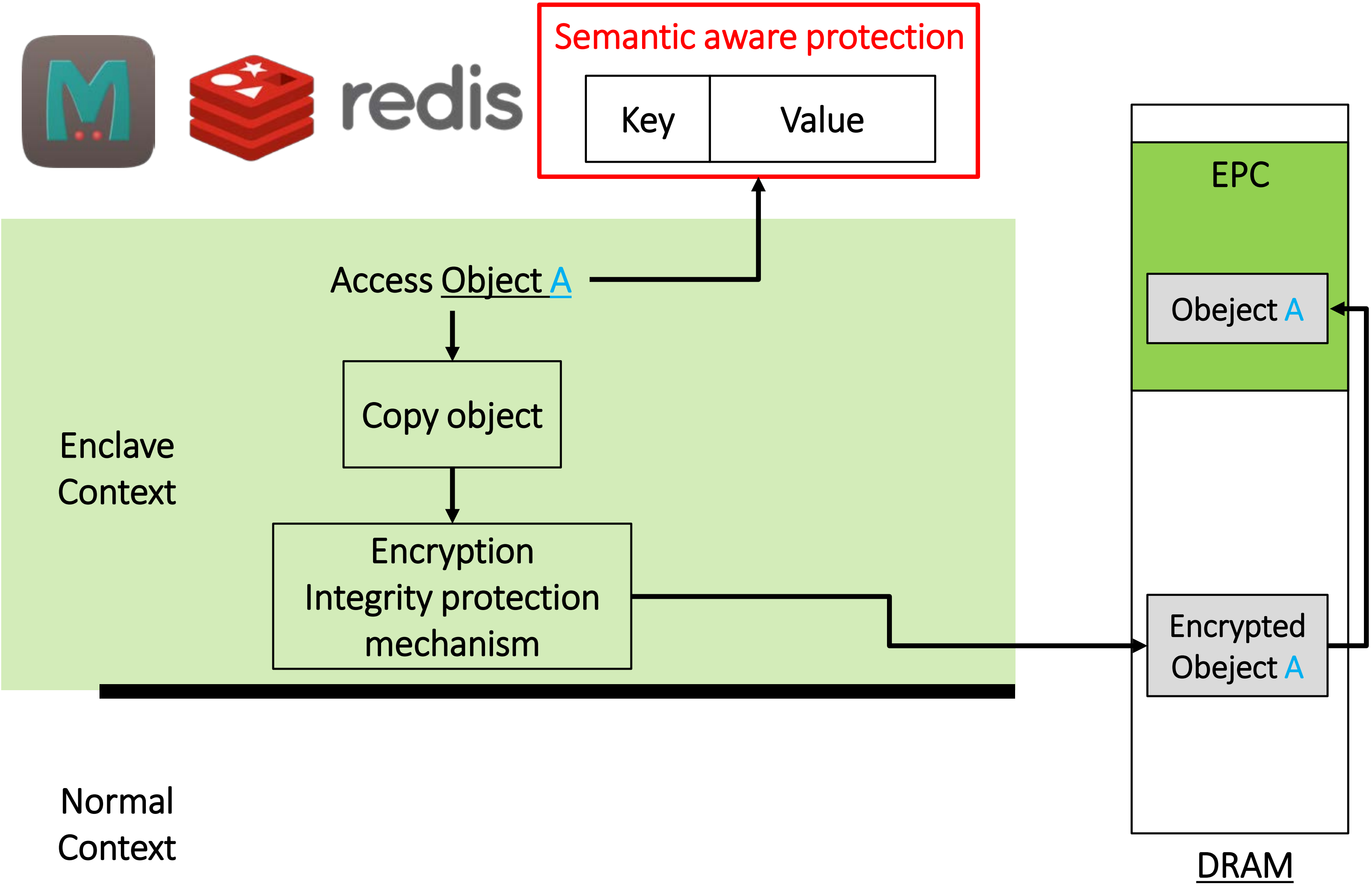
Encrypted Obeject A

DRAM

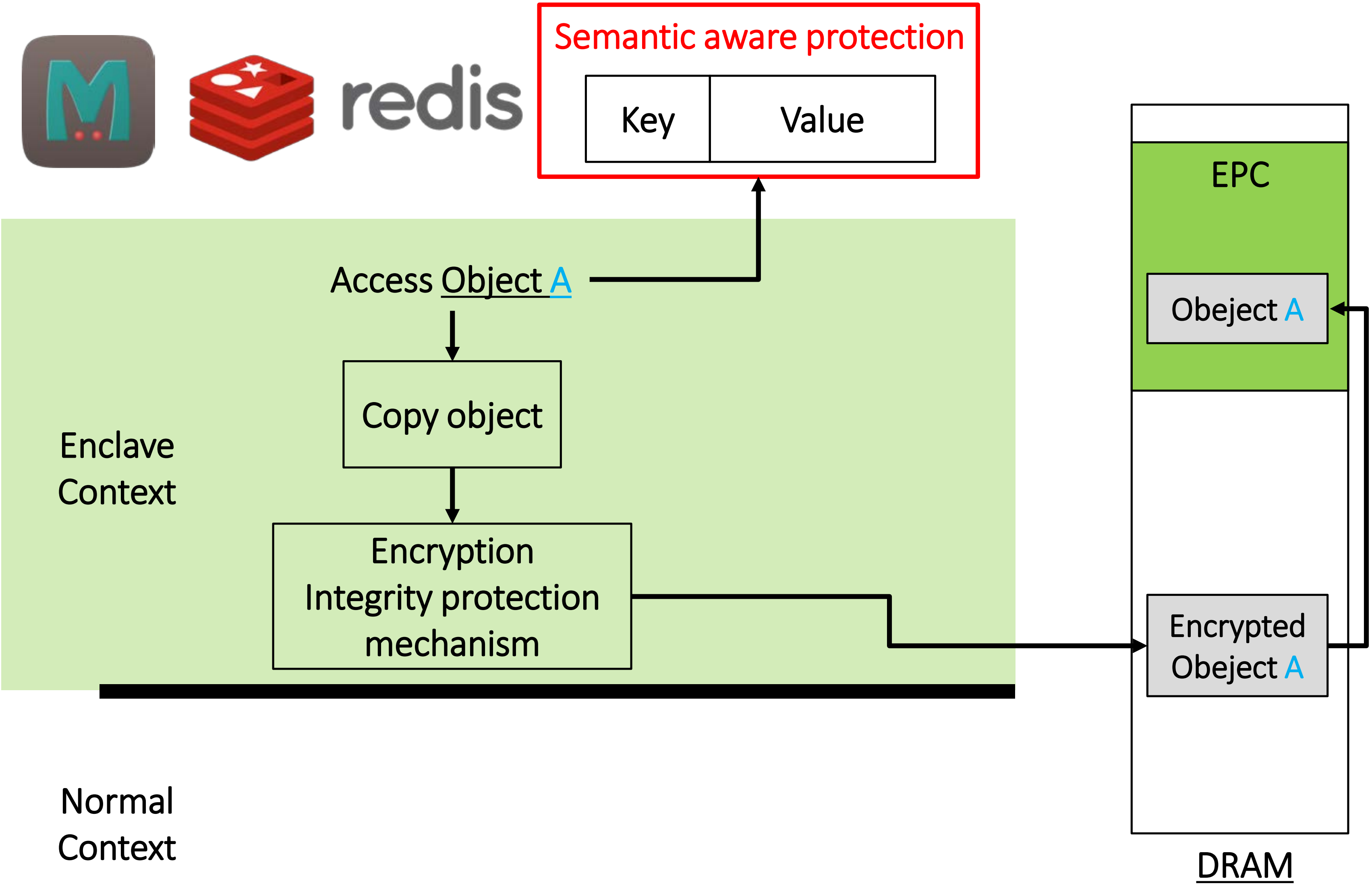
Proposed Design: Semantic Aware Protection



Proposed Design: Semantic Aware Protection



Proposed Design: Semantic Aware Protection



Semantic aware protection

Key	Value

Access Object A

Copy object

Encryption
Integrity protection
mechanism

Enclave
Context

Normal
Context

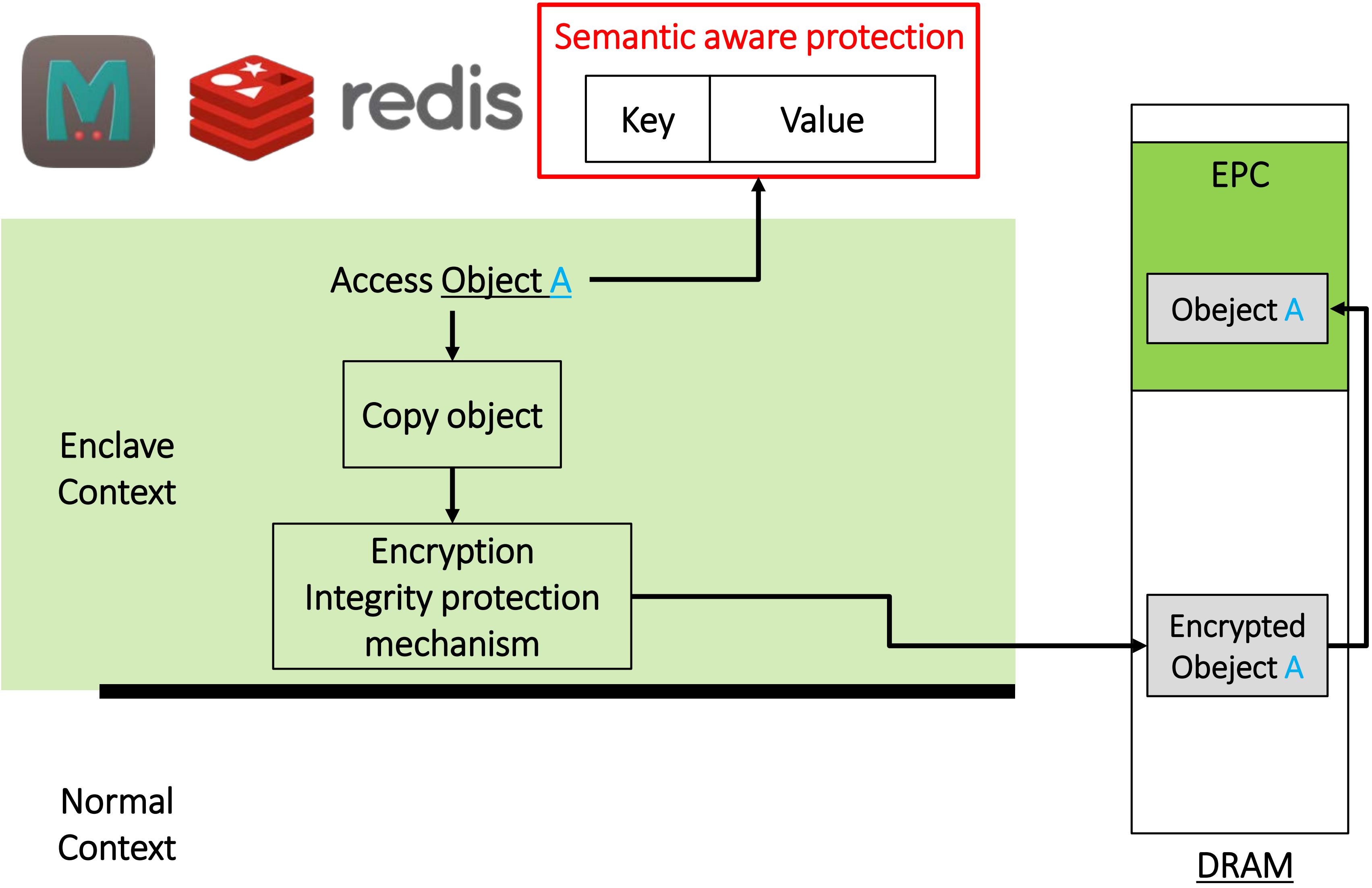
EPC

Obeject A

Encrypted
Obeject A

DRAM

Proposed Design: Semantic Aware Protection



Semantic aware protection

Key | Value

Access Object A

Copy object

Enclave Context

Encryption
Integrity protection
mechanism

Normal Context

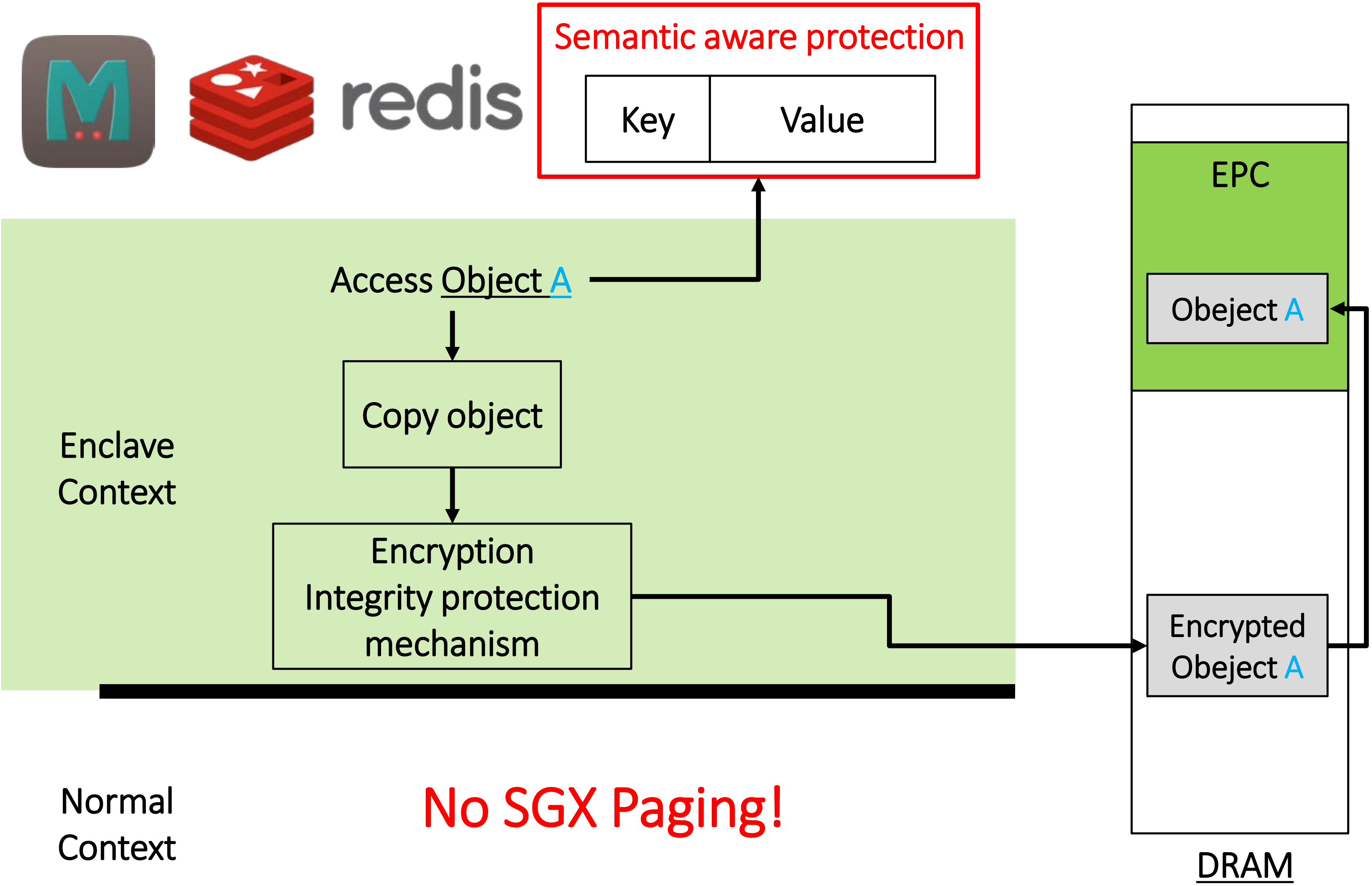
EPC

Obeject A

Encrypted
Obeject A

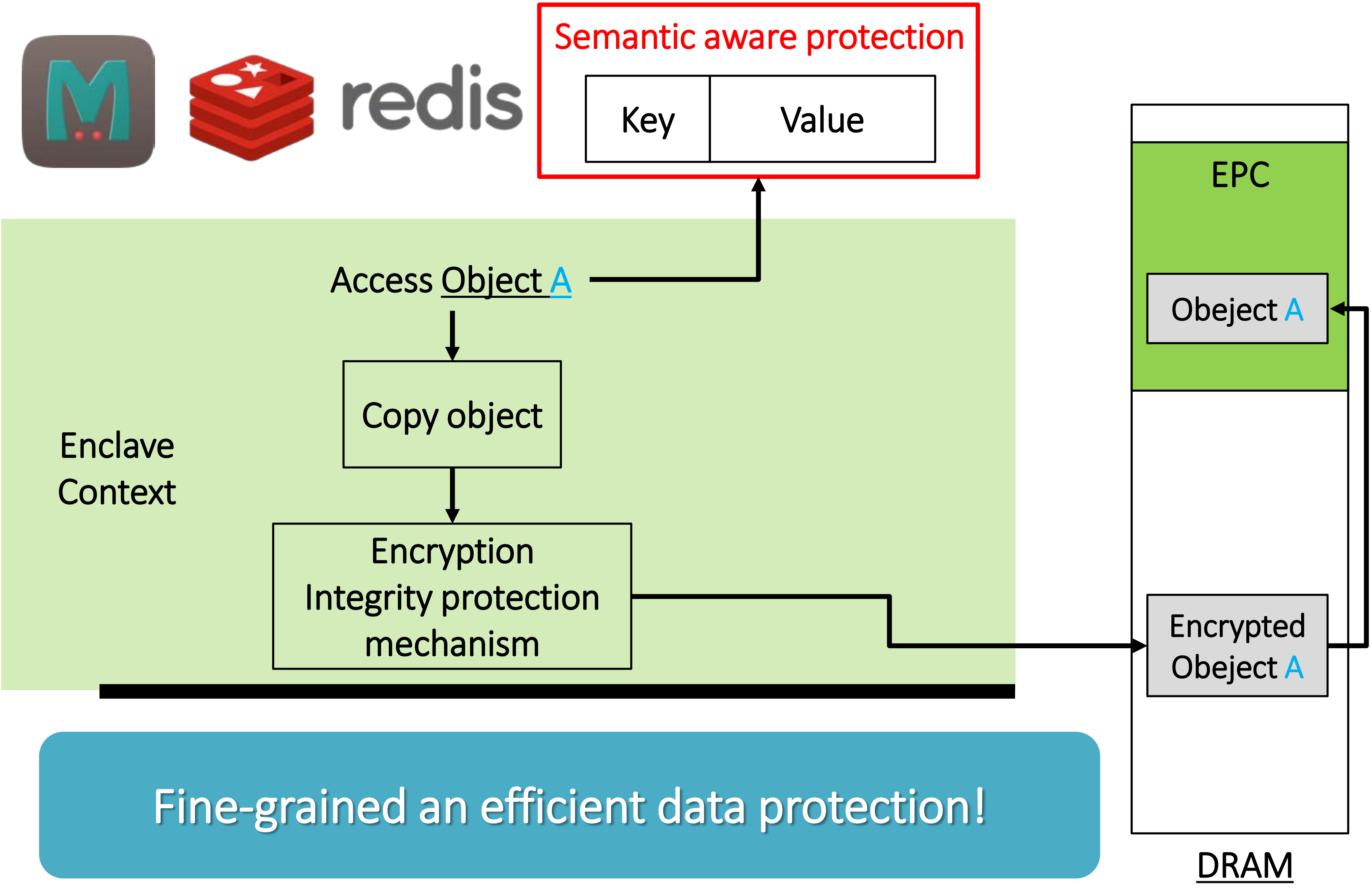
DRAM

Proposed Design: Semantic Aware Protection



No SGX Paging!

Proposed Design: Semantic Aware Protection



Fine-grained an efficient data protection!

Threat Model

- ShieldStore protects *confidentiality* and *integrity* of key/values

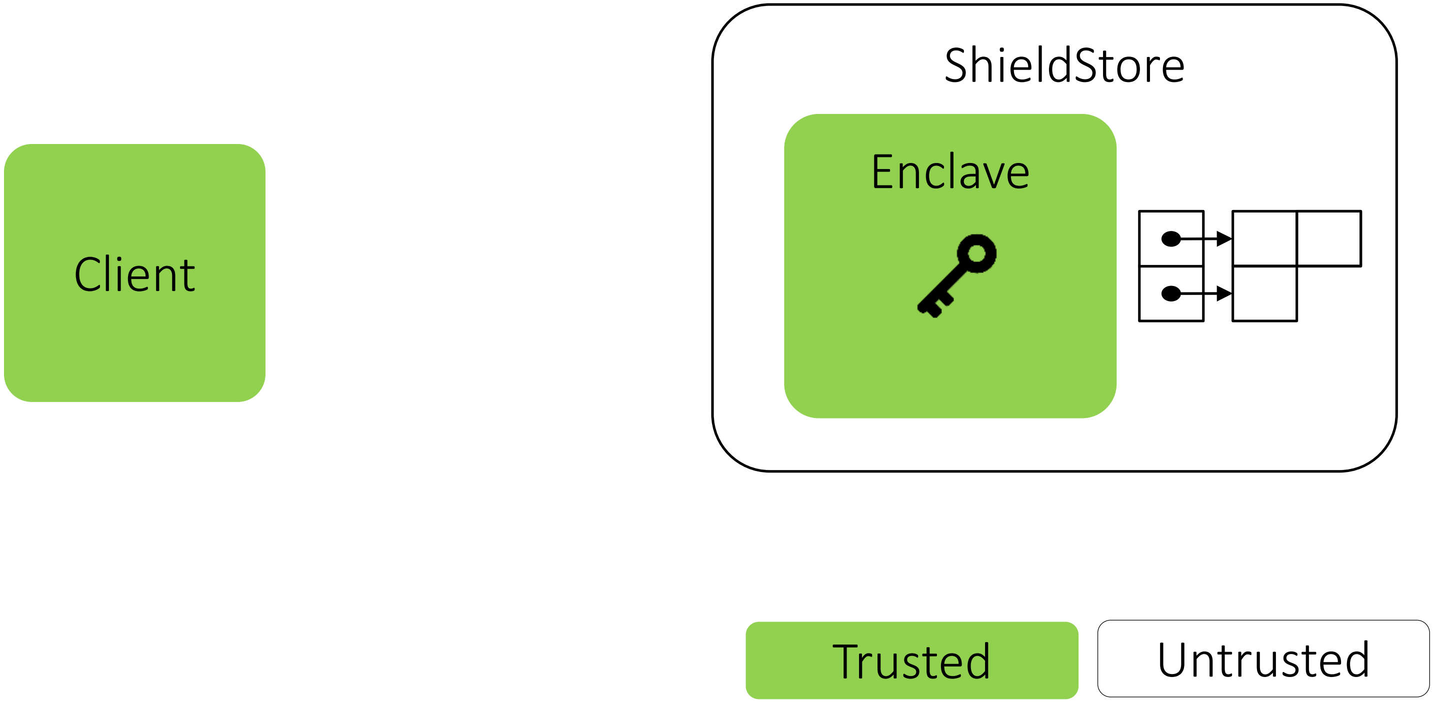
- Trusted Computing Base (TCB) of ShieldStore
  - SGX enabled Processor chip
  - Code & data in *enclave*

- Out of scope
  - Side channel attacks (ex. Foreshadow, controlled channel attacks)
  - Availability attacks

Overall Design of ShieldStore

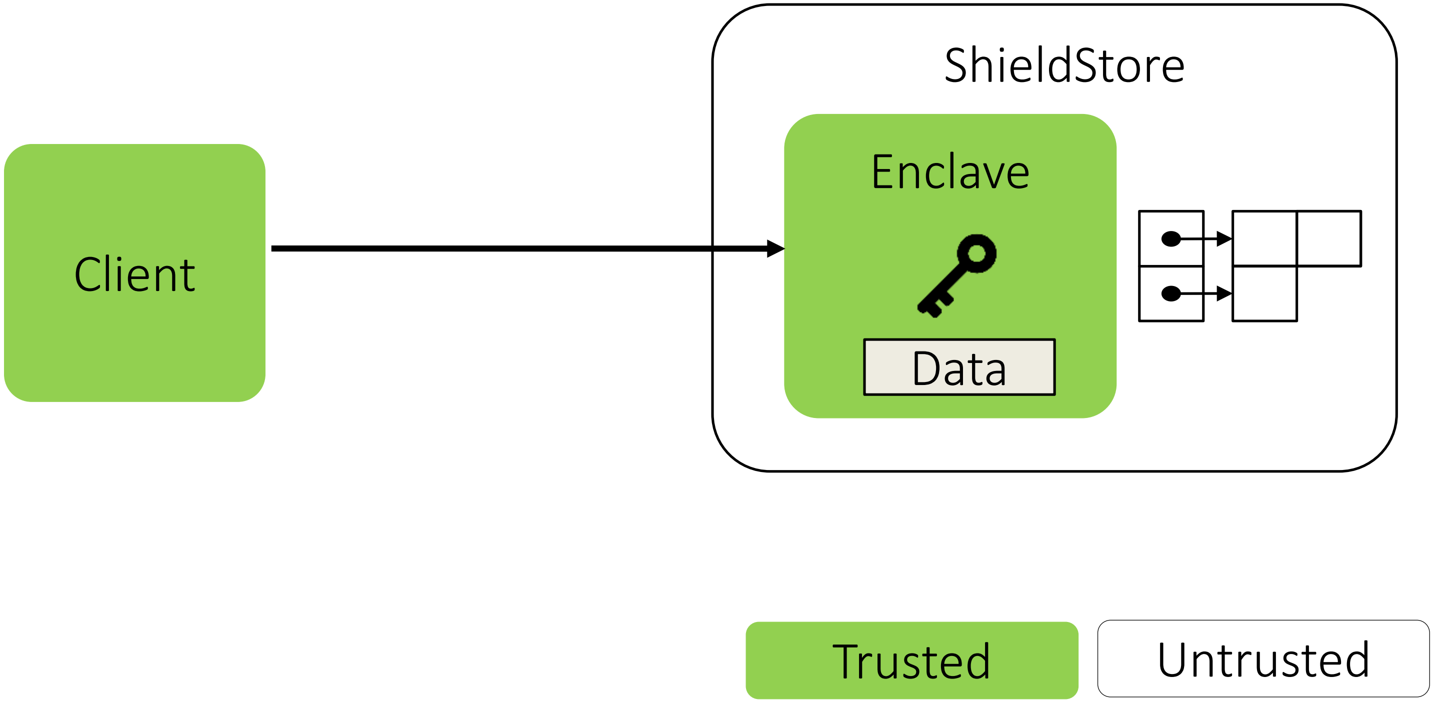
- Maintain small secure meta-data in trusted memory region

- Store main data structure on untrusted memory region
  - With encrypted and integrity-protected key-value entries



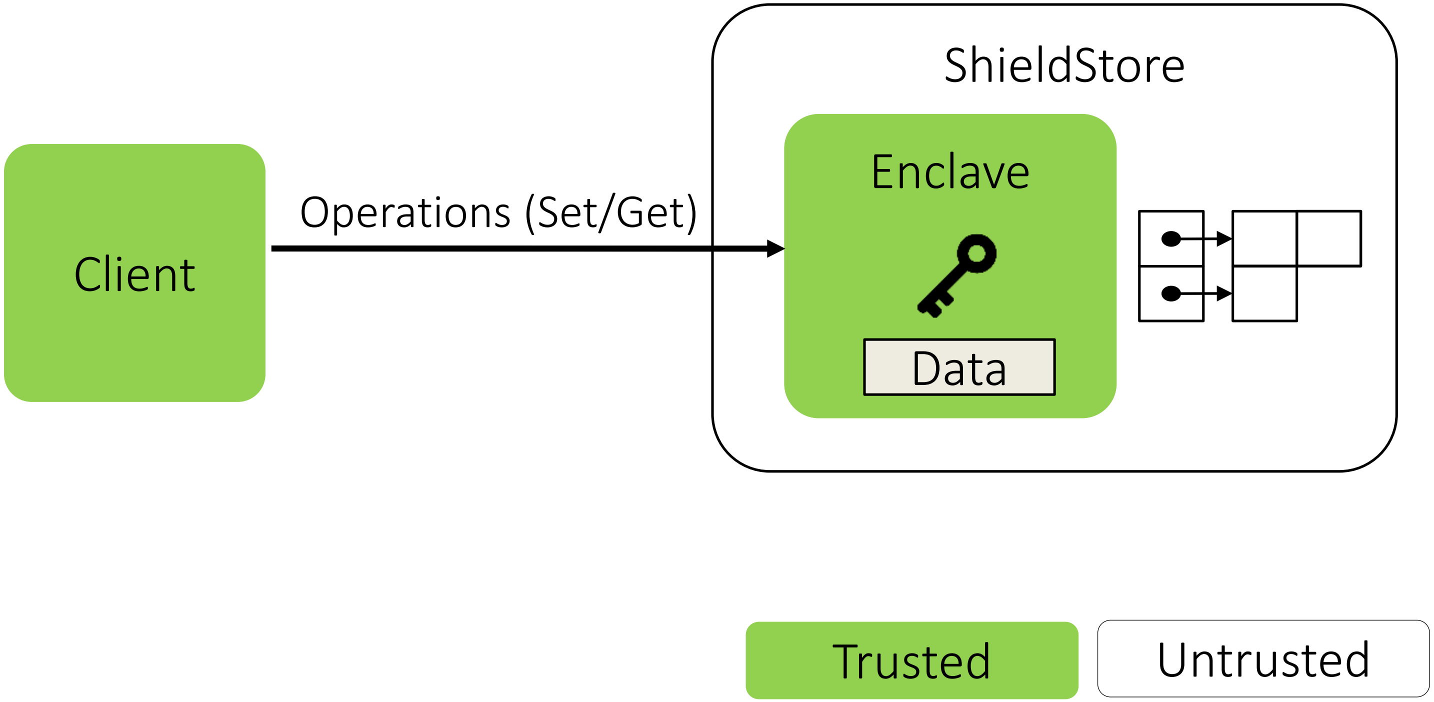
Overall Design of ShieldStore

- Maintain small secure meta-data in trusted memory region
- Store main data structure on untrusted memory region
  - – With encrypted and integrity-protected key-value entries



Overall Design of ShieldStore

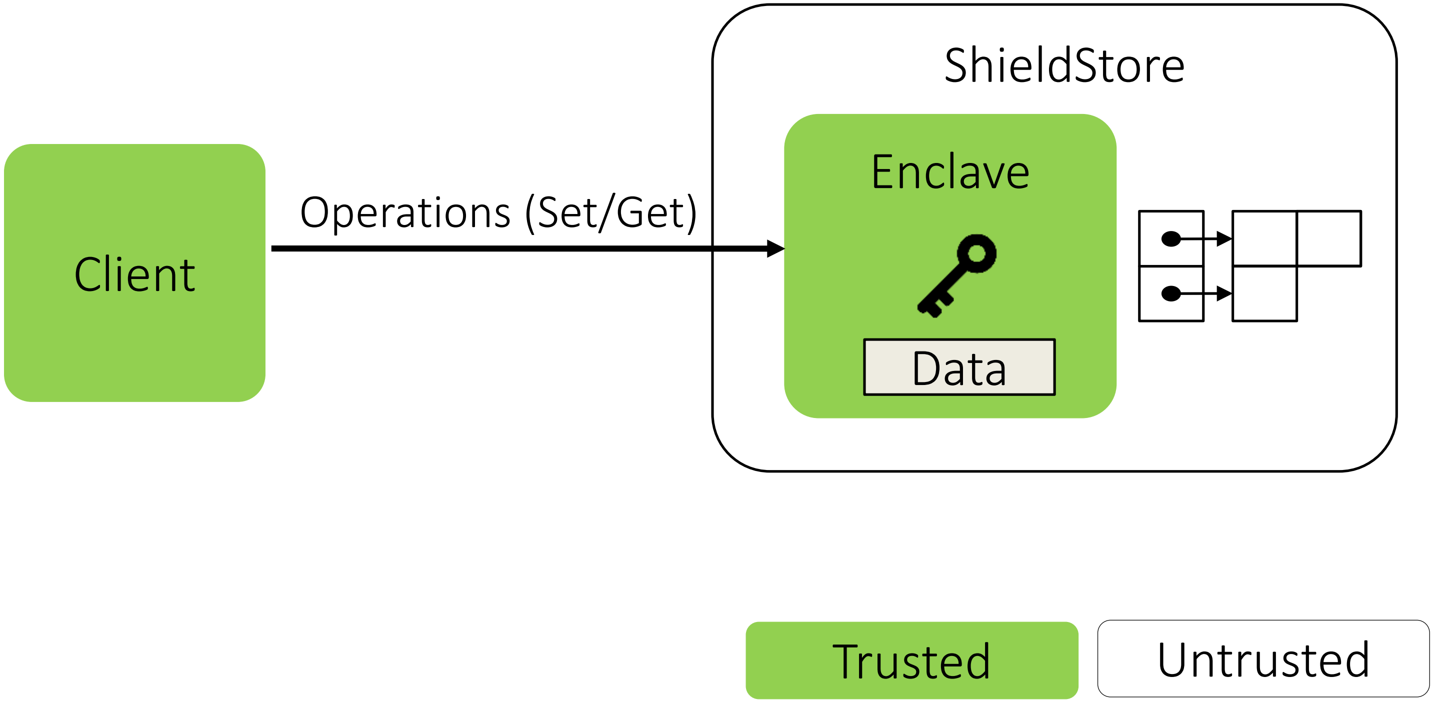
- Maintain small secure meta-data in trusted memory region
- Store main data structure on untrusted memory region
  – With encrypted and integrity-protected key-value entries



Overall Design of ShieldStore

- Maintain small secure meta-data in trusted memory region

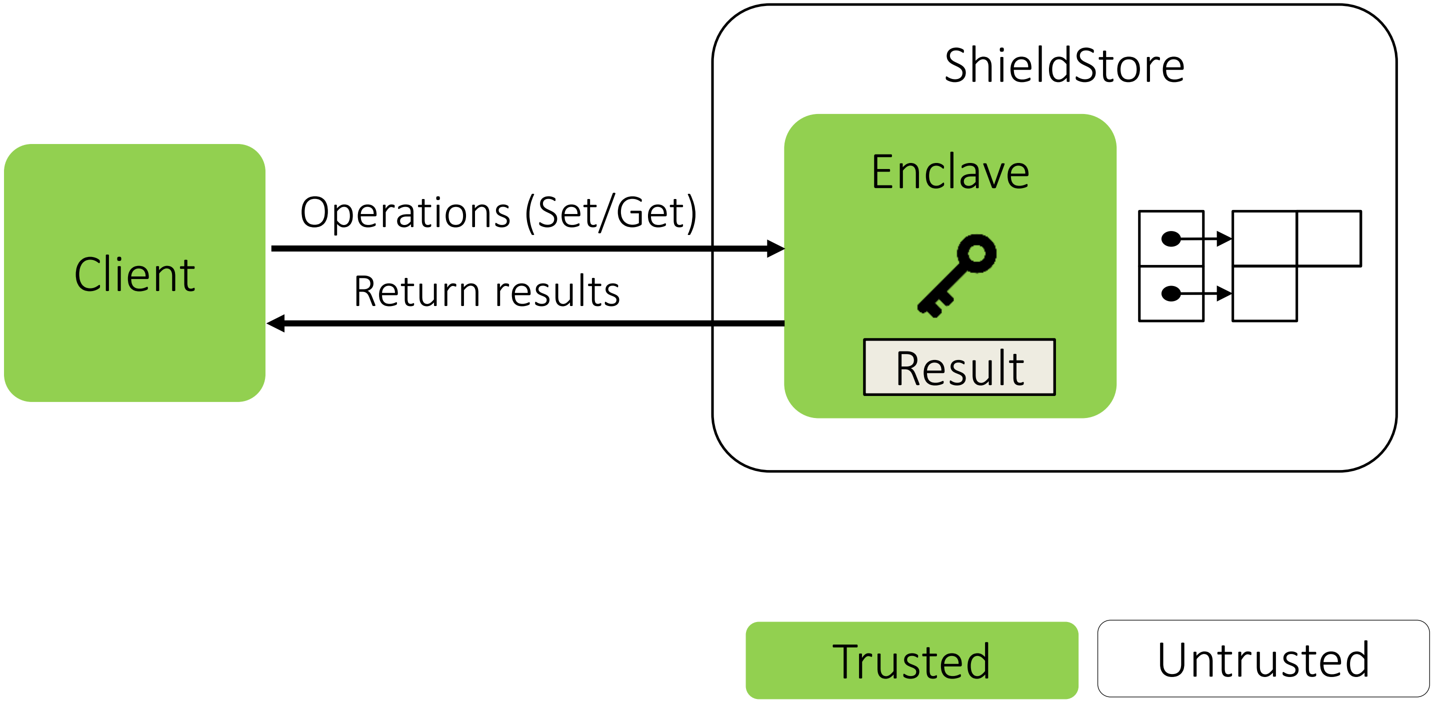
- Store main data structure on untrusted memory region
  – With encrypted and integrity-protected key-value entries



Overall Design of ShieldStore

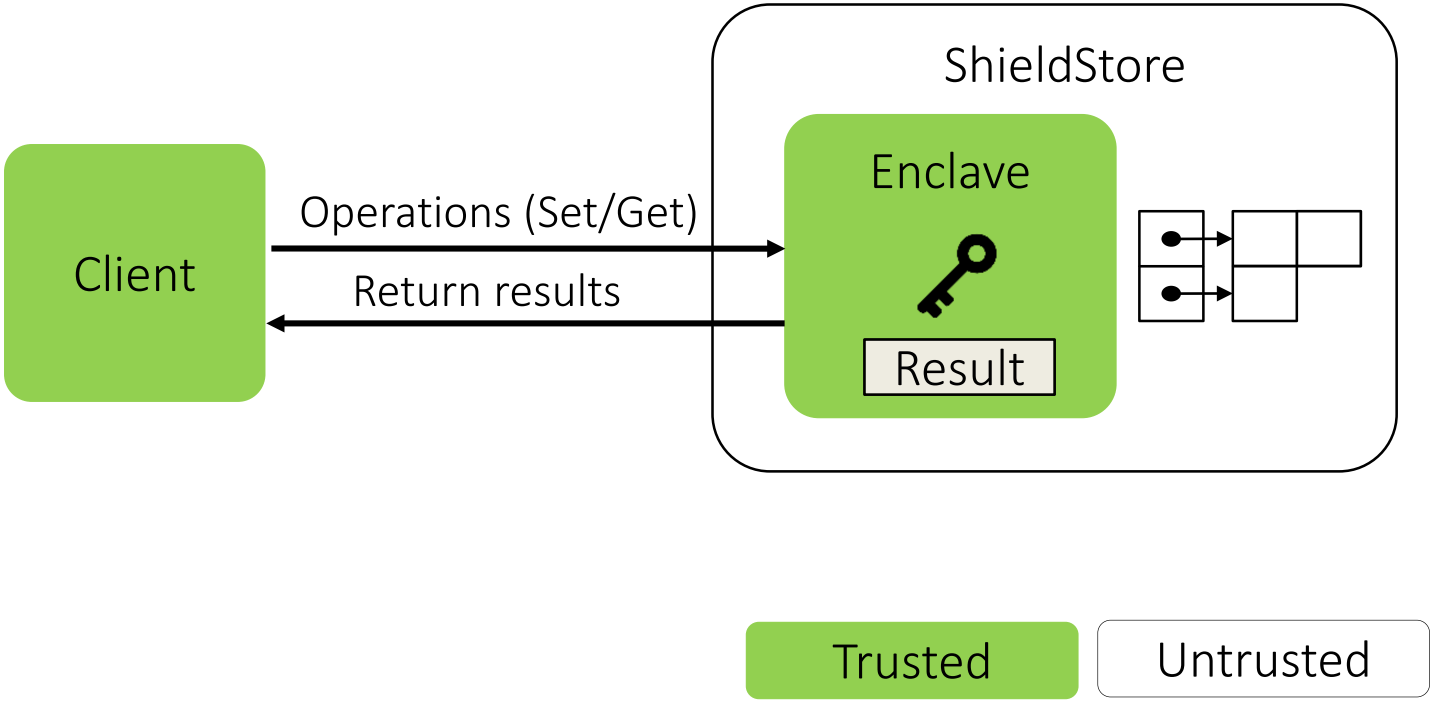
- Maintain small secure meta-data in trusted memory region

- Store main data structure on untrusted memory region
  – With encrypted and integrity-protected key-value entries



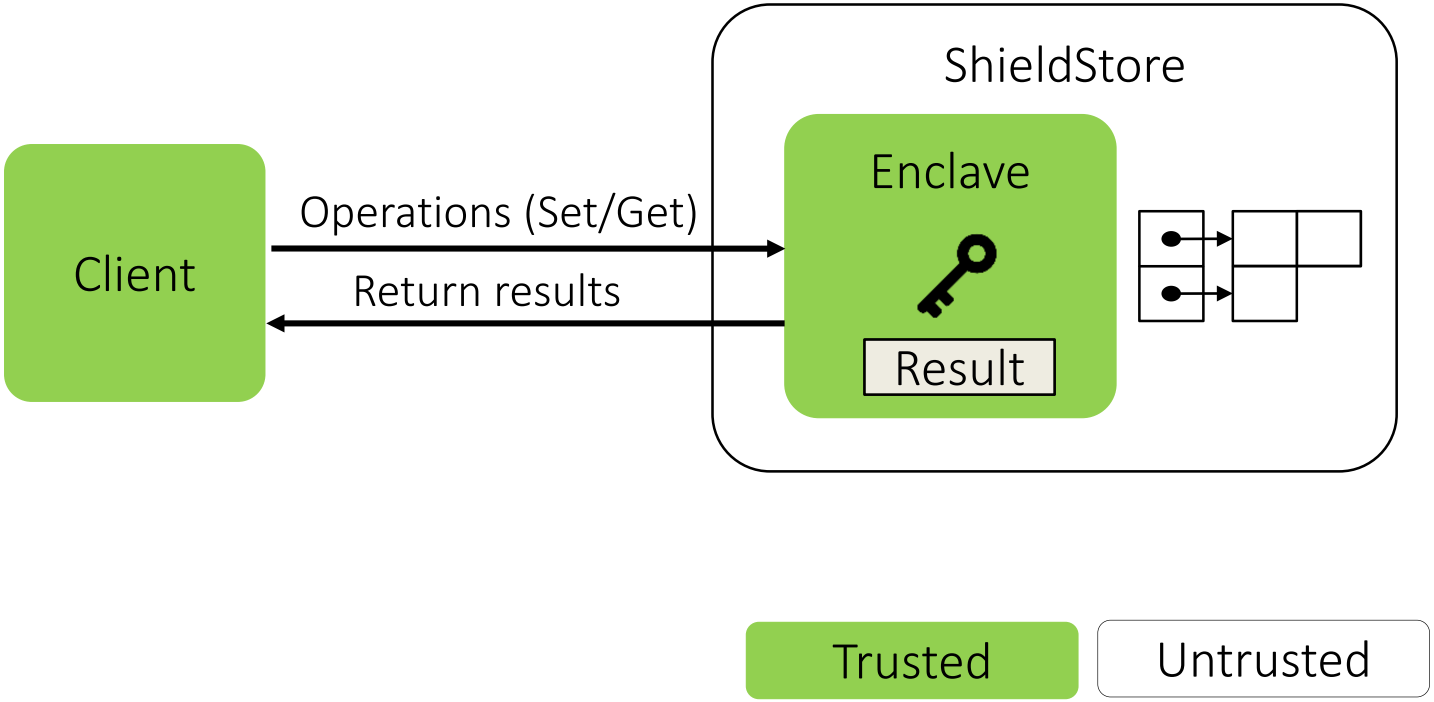
Overall Design of ShieldStore

- Maintain small secure meta-data in trusted memory region
- Store main data structure on untrusted memory region
  - – With encrypted and integrity-protected key-value entries



Overall Design of ShieldStore

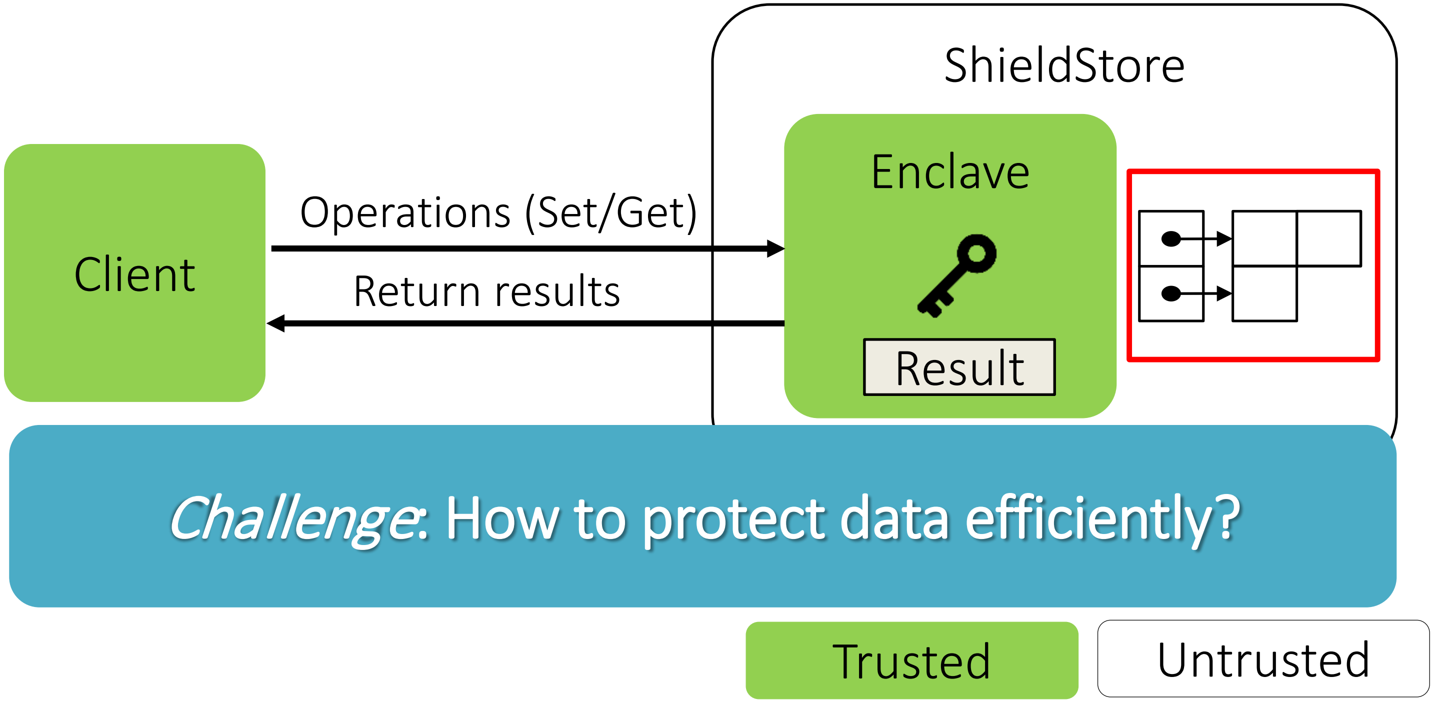
- Maintain small secure meta-data in trusted memory region
- Store main data structure on untrusted memory region
  – With encrypted and integrity-protected key-value entries



Overall Design of ShieldStore

- Maintain small secure meta-data in trusted memory region

- Store main data structure on untrusted memory region
  - With encrypted and integrity-protected key-value entries

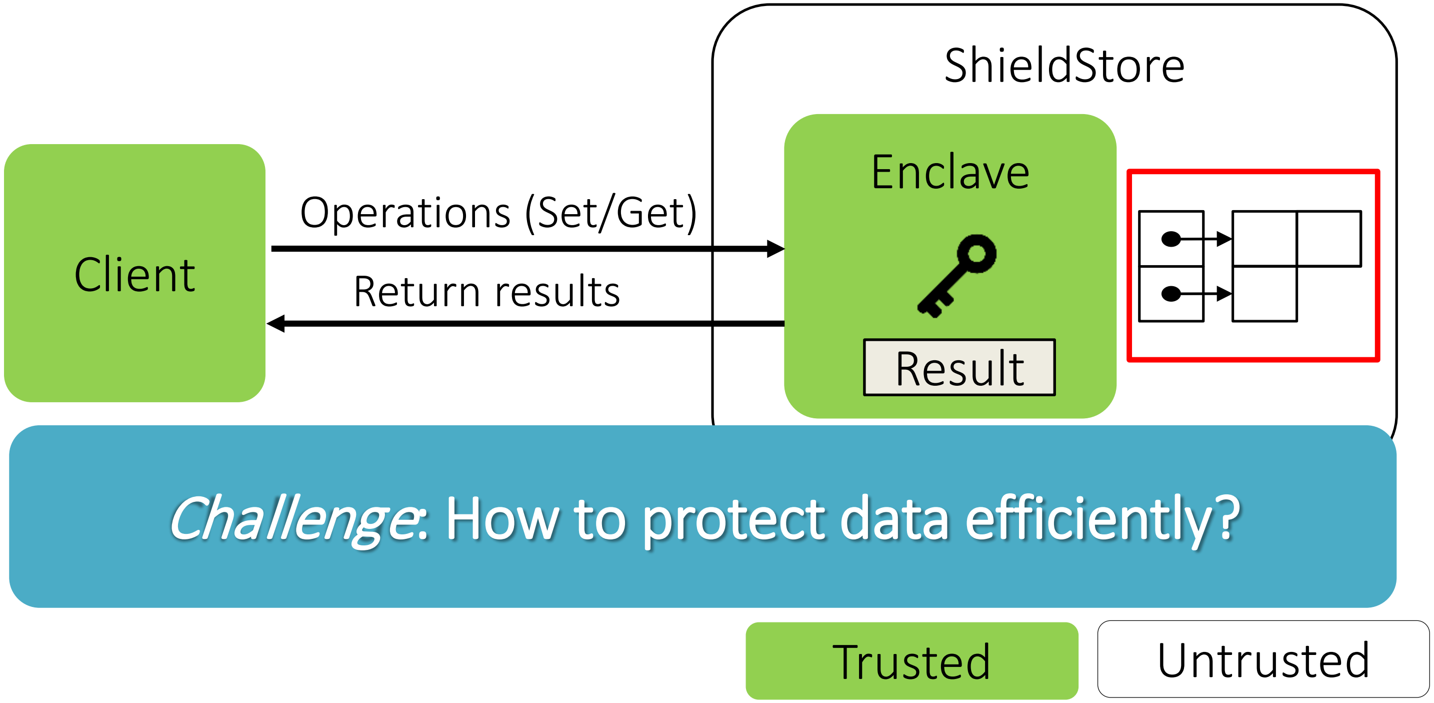


Challenge: How to protect data efficiently?

Overall Design of ShieldStore

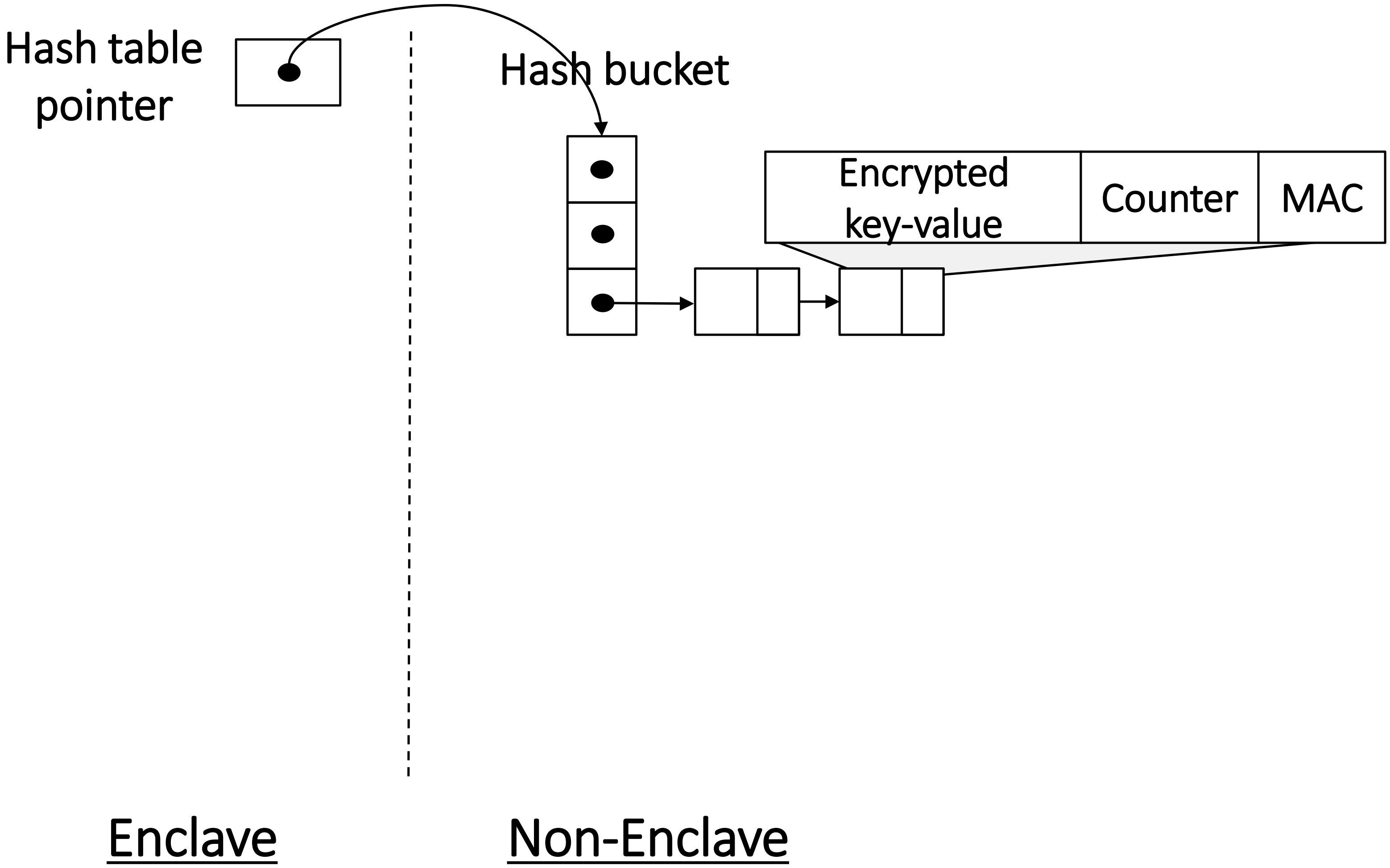
- Maintain small secure meta-data in trusted memory region

- Store main data structure on untrusted memory region
  – With encrypted and integrity-protected key-value entries

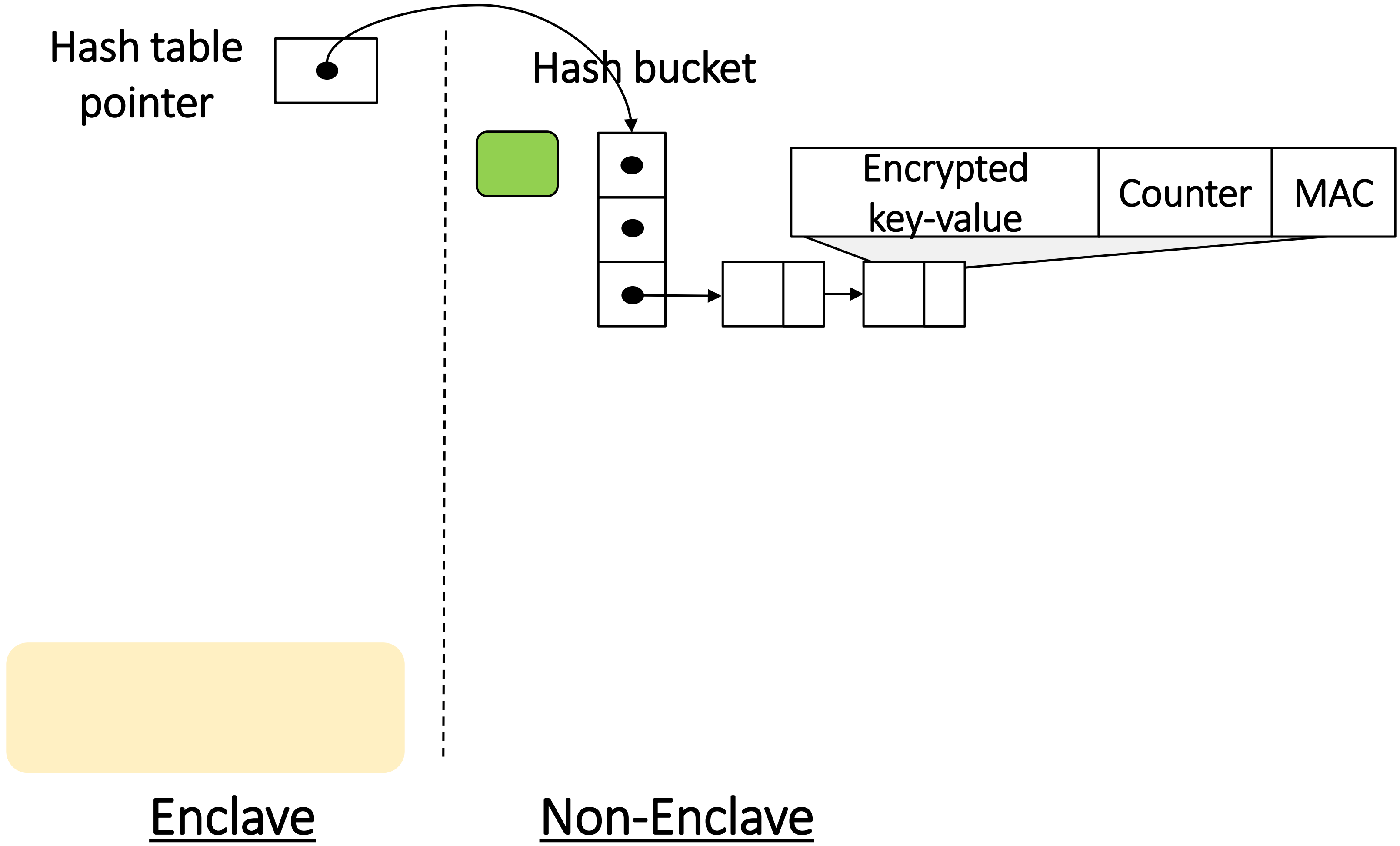


Challenge: How to protect data efficiently?

How To Protect Data?



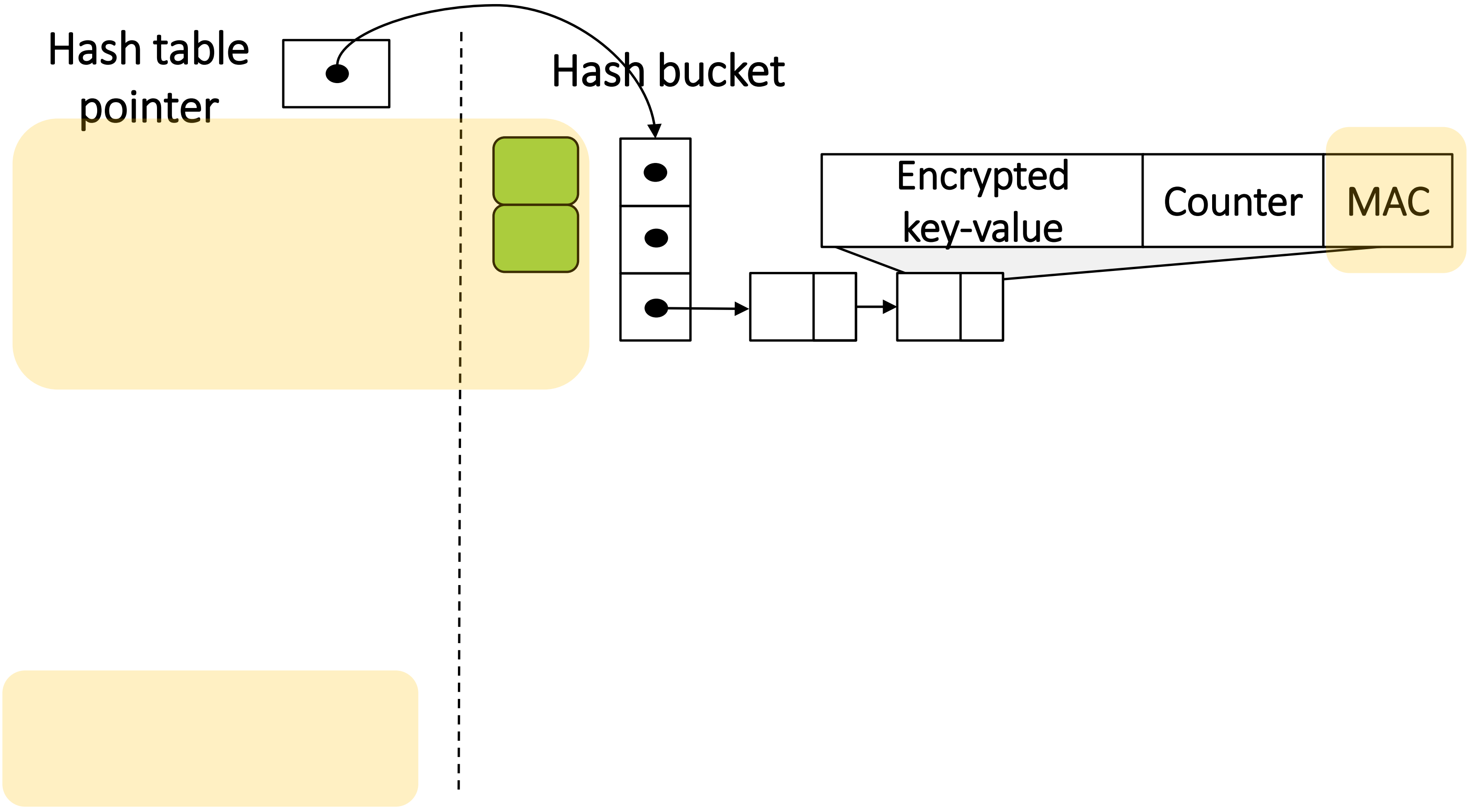
How To Protect Data?



Enclave

Non-Enclave

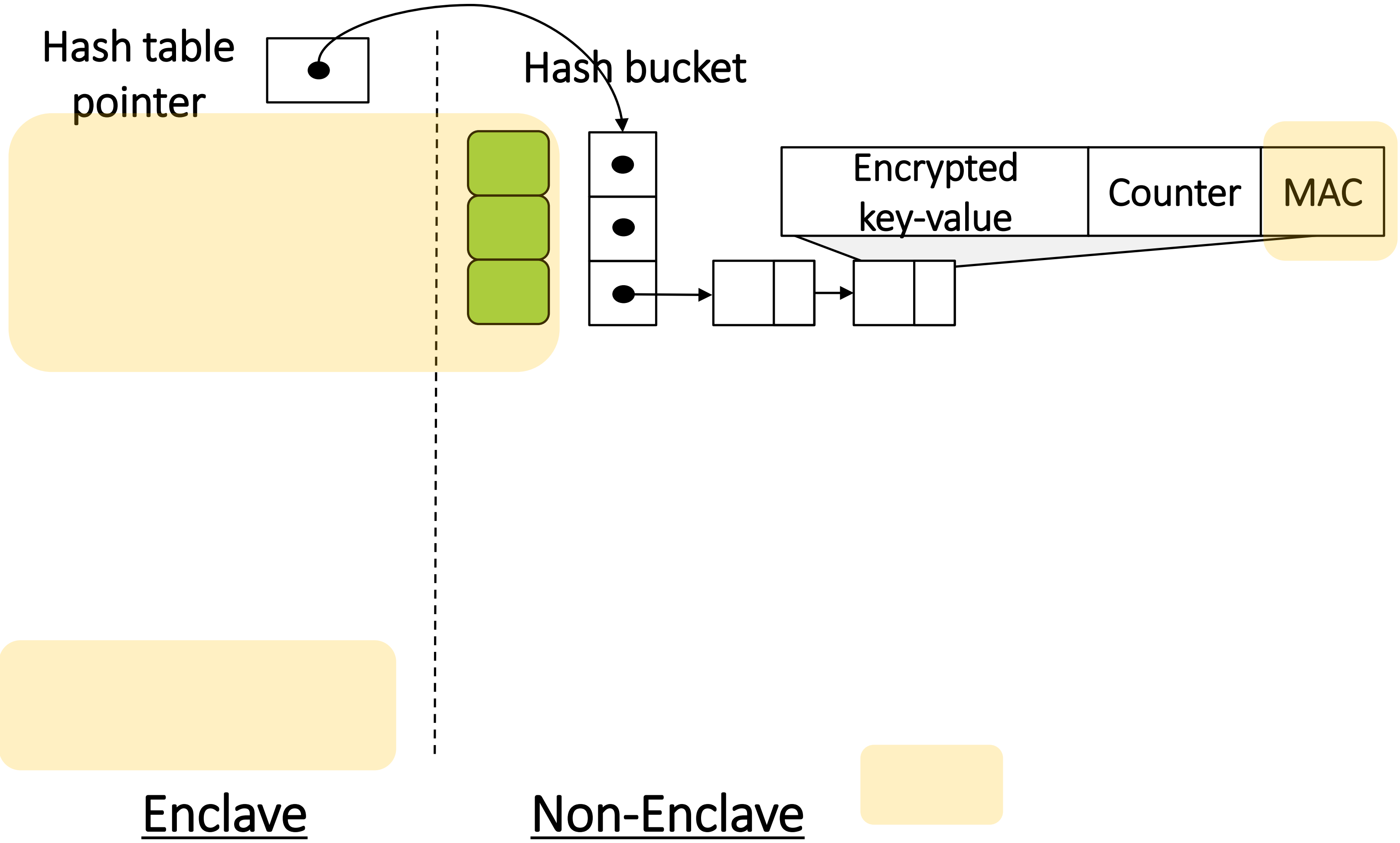
How To Protect Data?



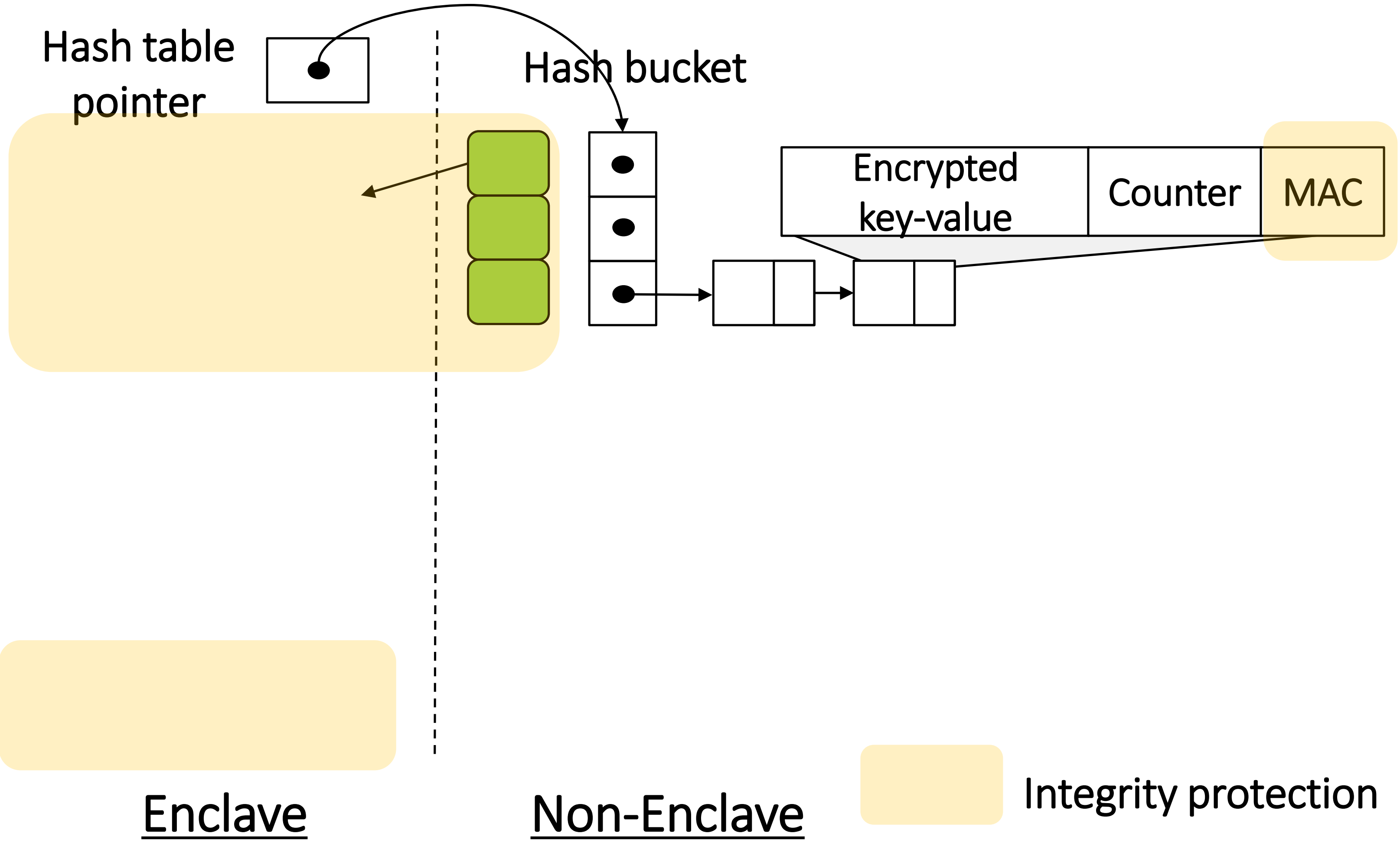
Enclave

Non-Enclave

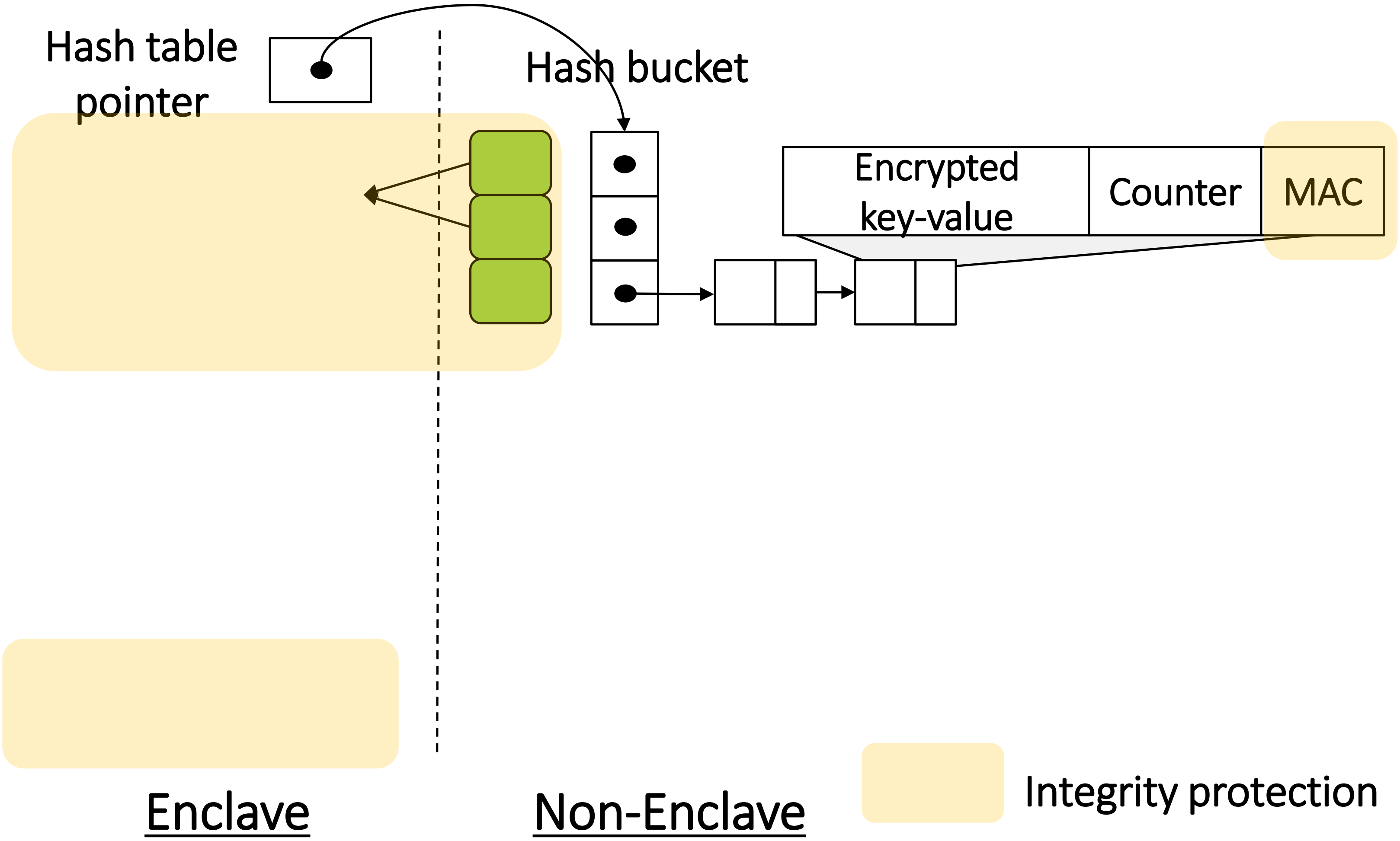
How To Protect Data?



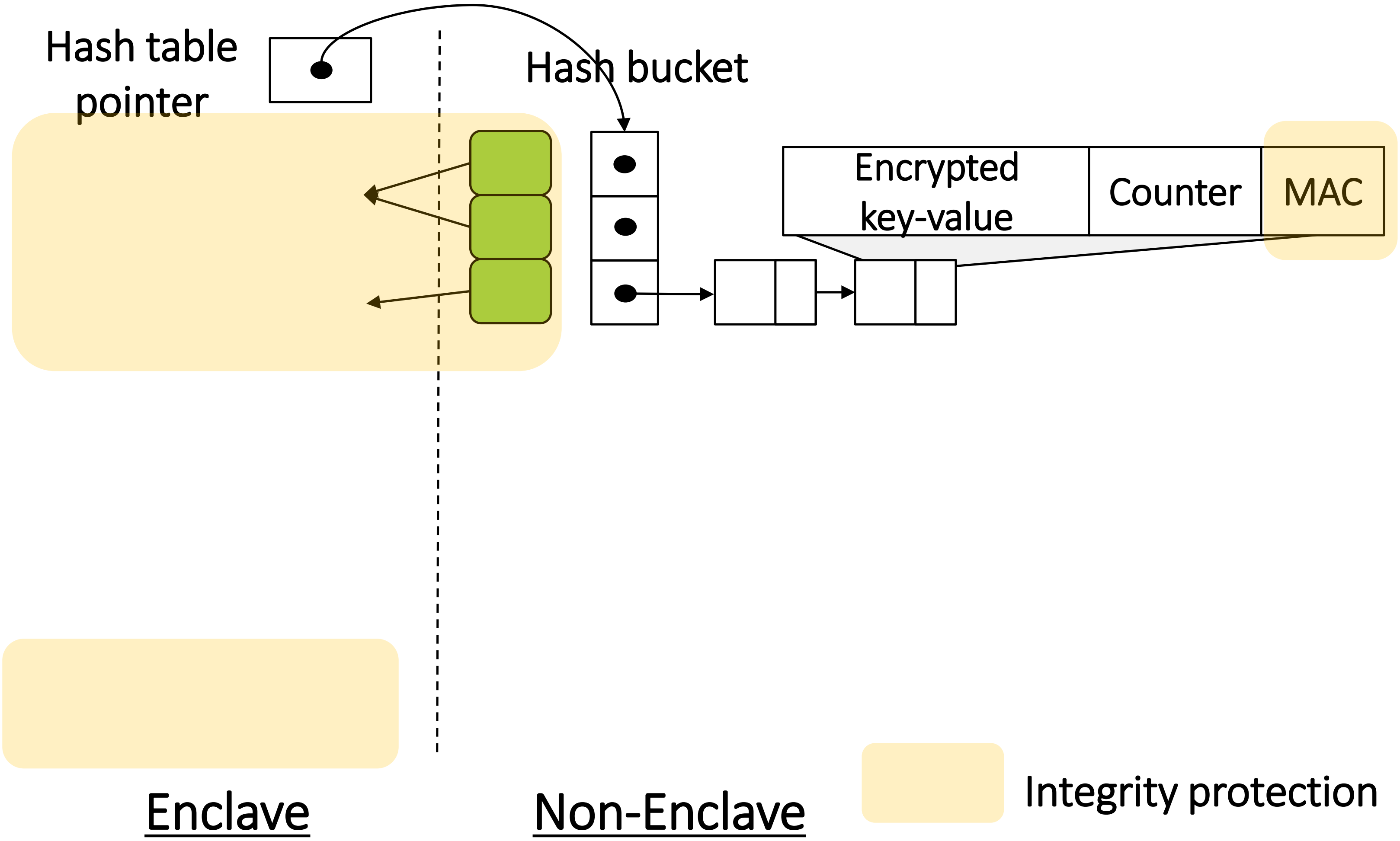
How To Protect Data?



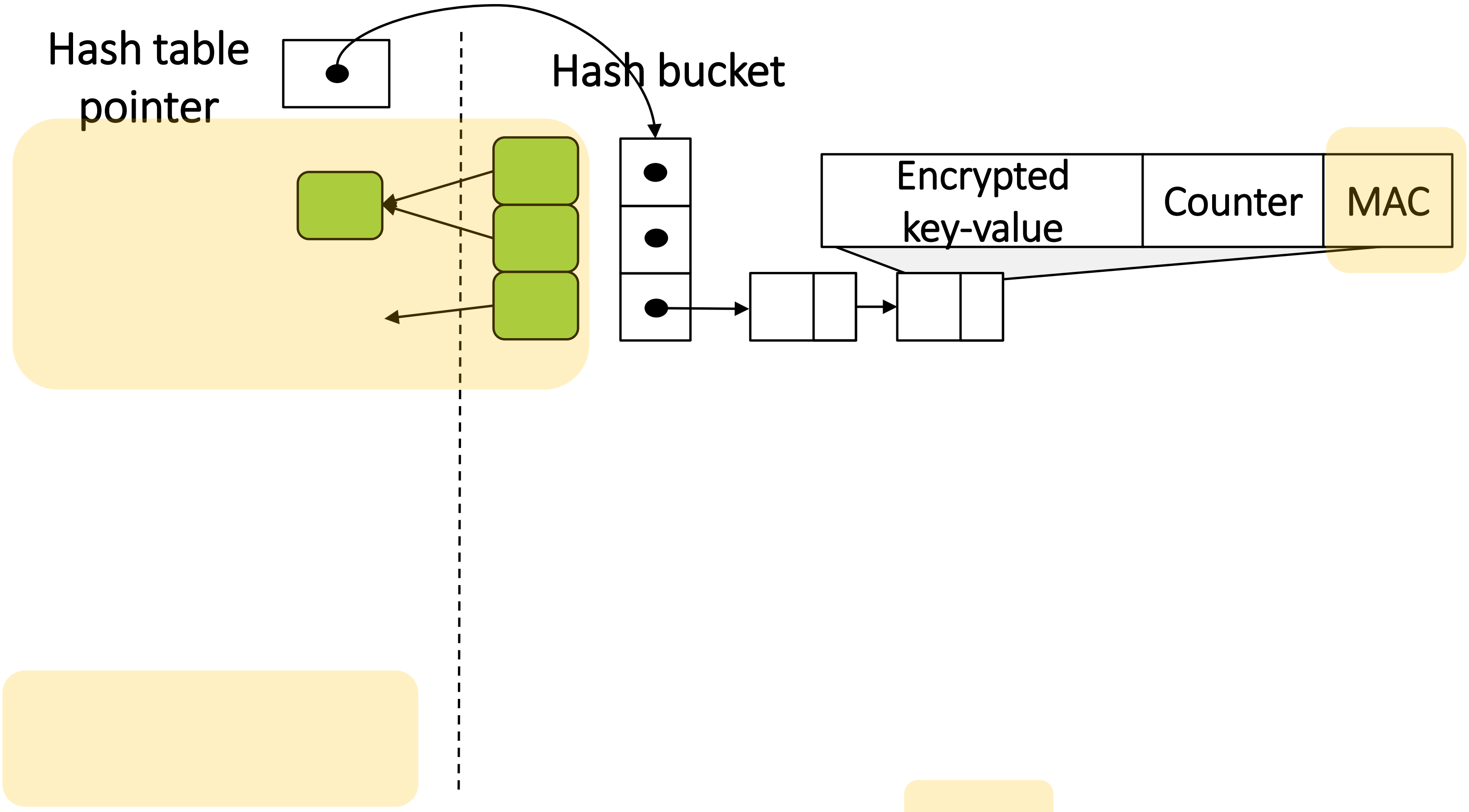
How To Protect Data?



How To Protect Data?



How To Protect Data?



Hash table pointer

Hash bucket

Encrypted key-value

Counter

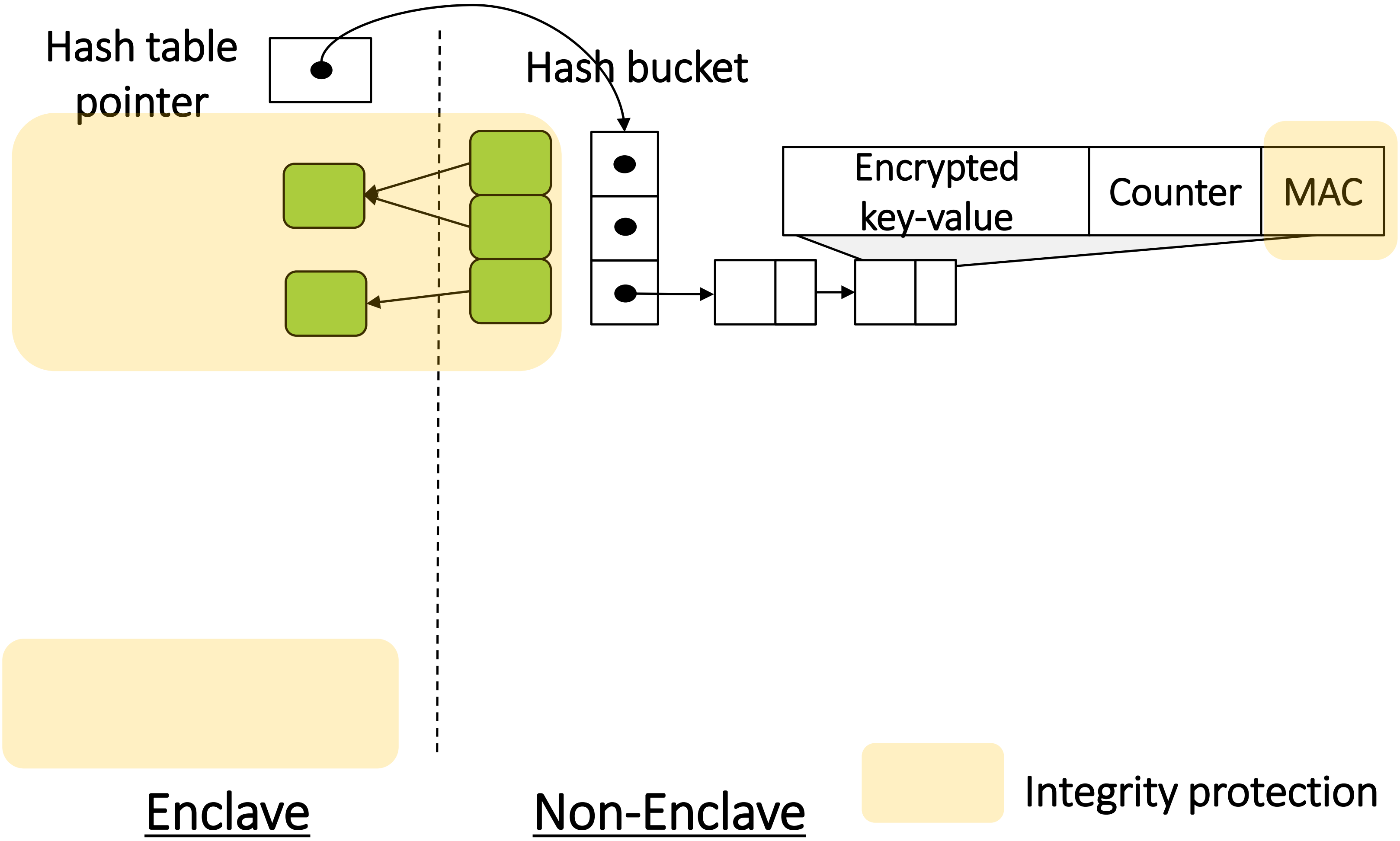
MAC

<u>Enclave</u>

<u>Non-Enclave</u>

🟨 Integrity protection

How To Protect Data?



Hash table pointer

Hash bucket

Encrypted key-value

Counter

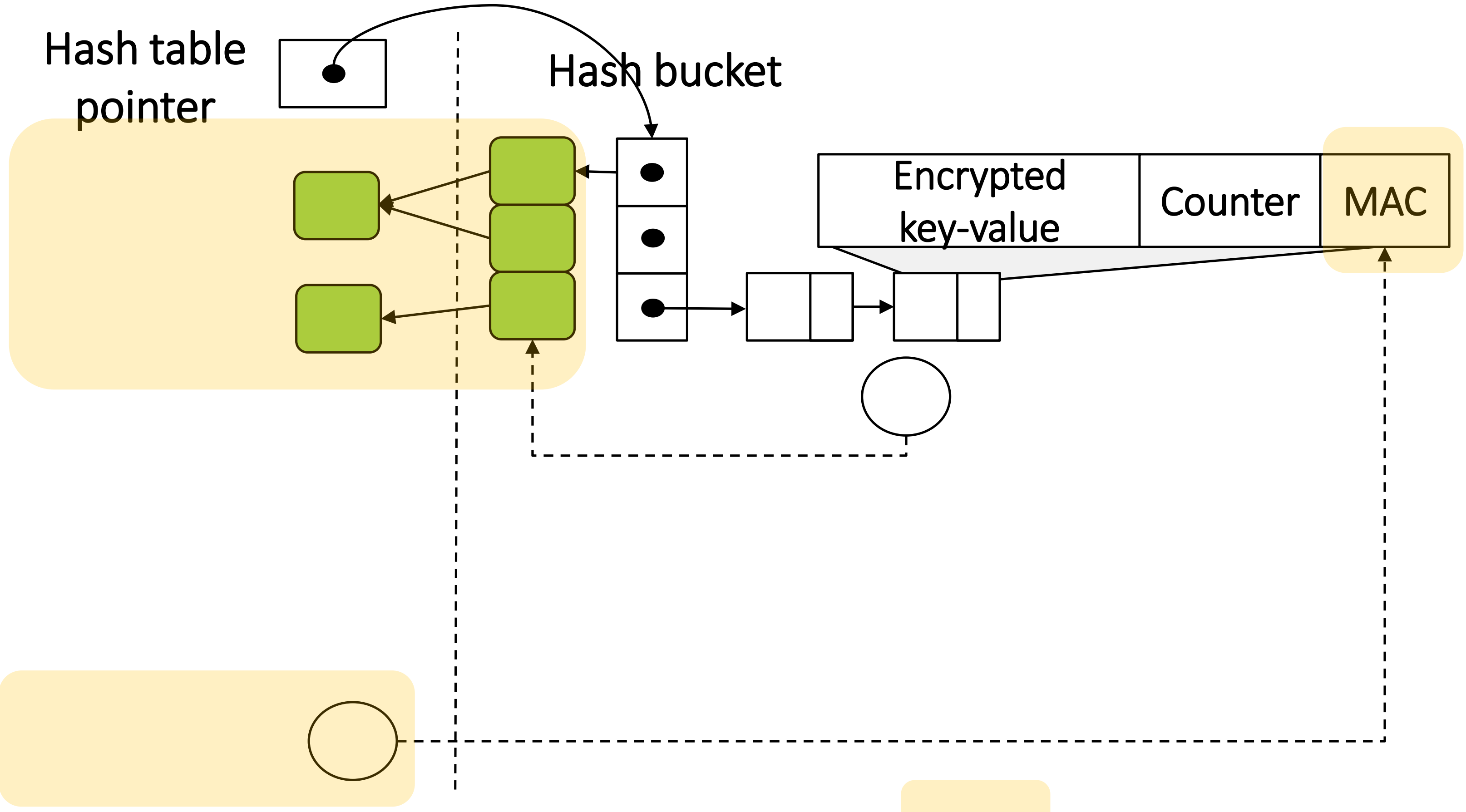
MAC

Enclave

Non-Enclave

Integrity protection

How To Protect Data?

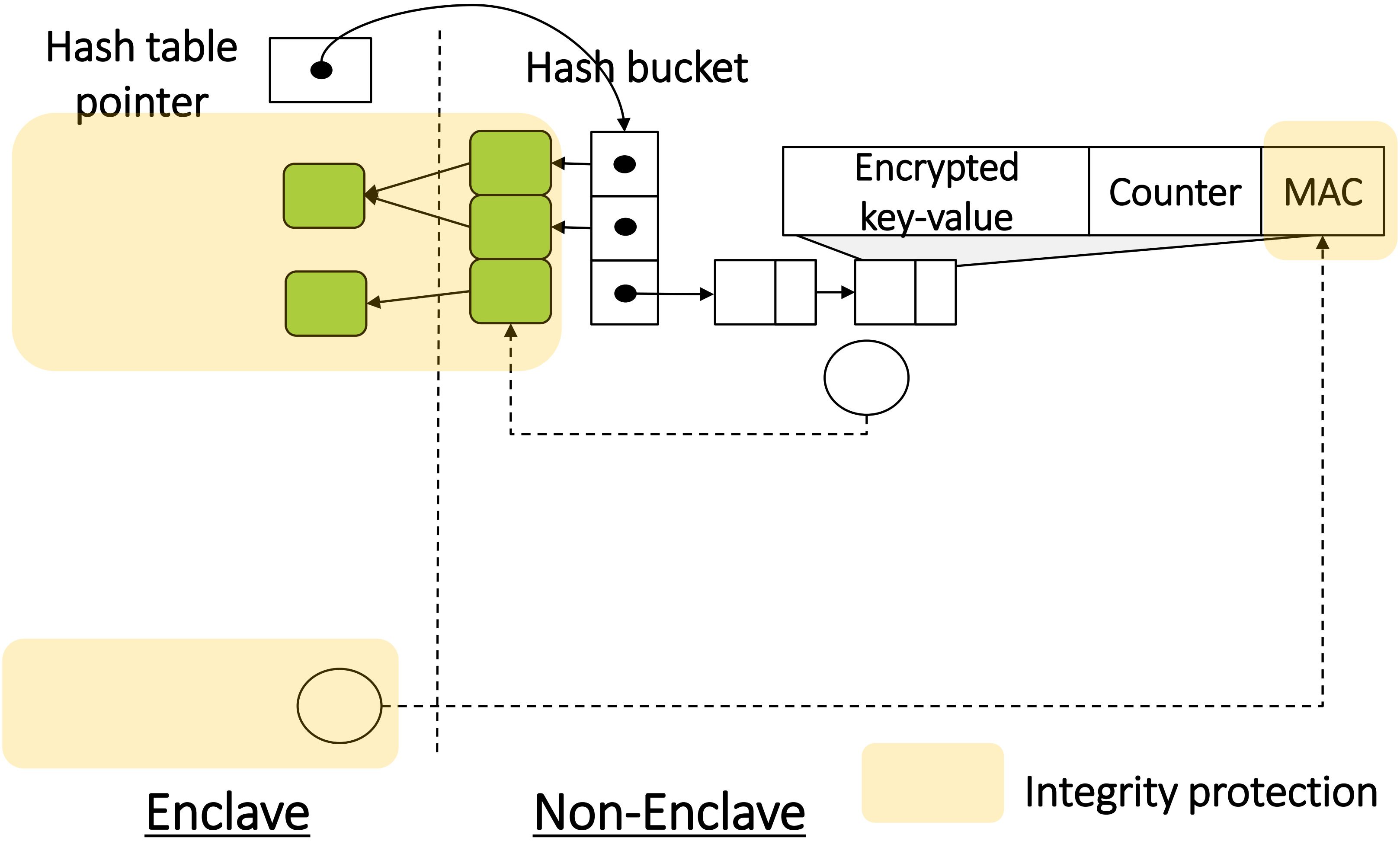


Enclave

Non-Enclave

🟨 Integrity protection

How To Protect Data?



Hash table pointer

Hash bucket

Encrypted key-value

Counter

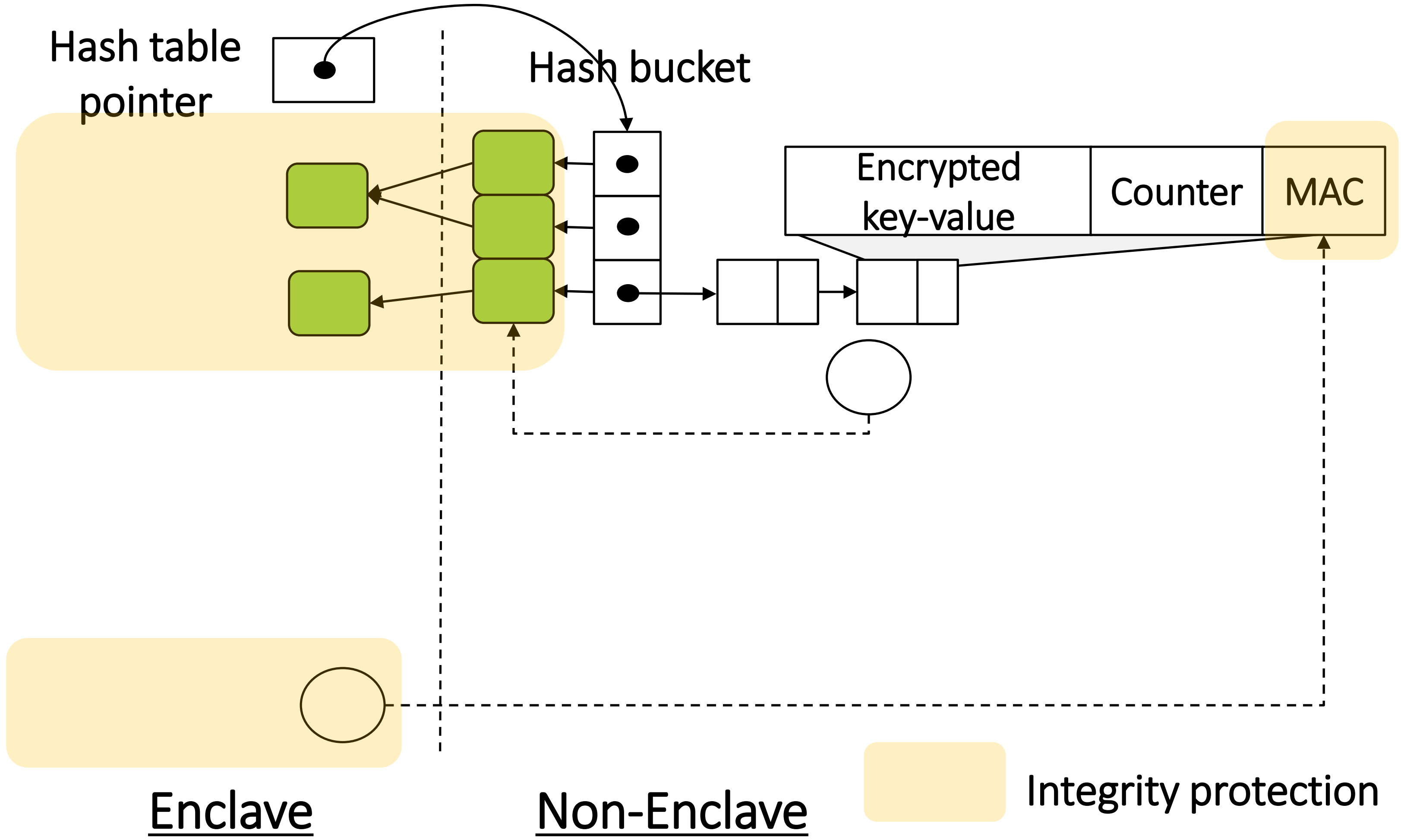
MAC

Enclave

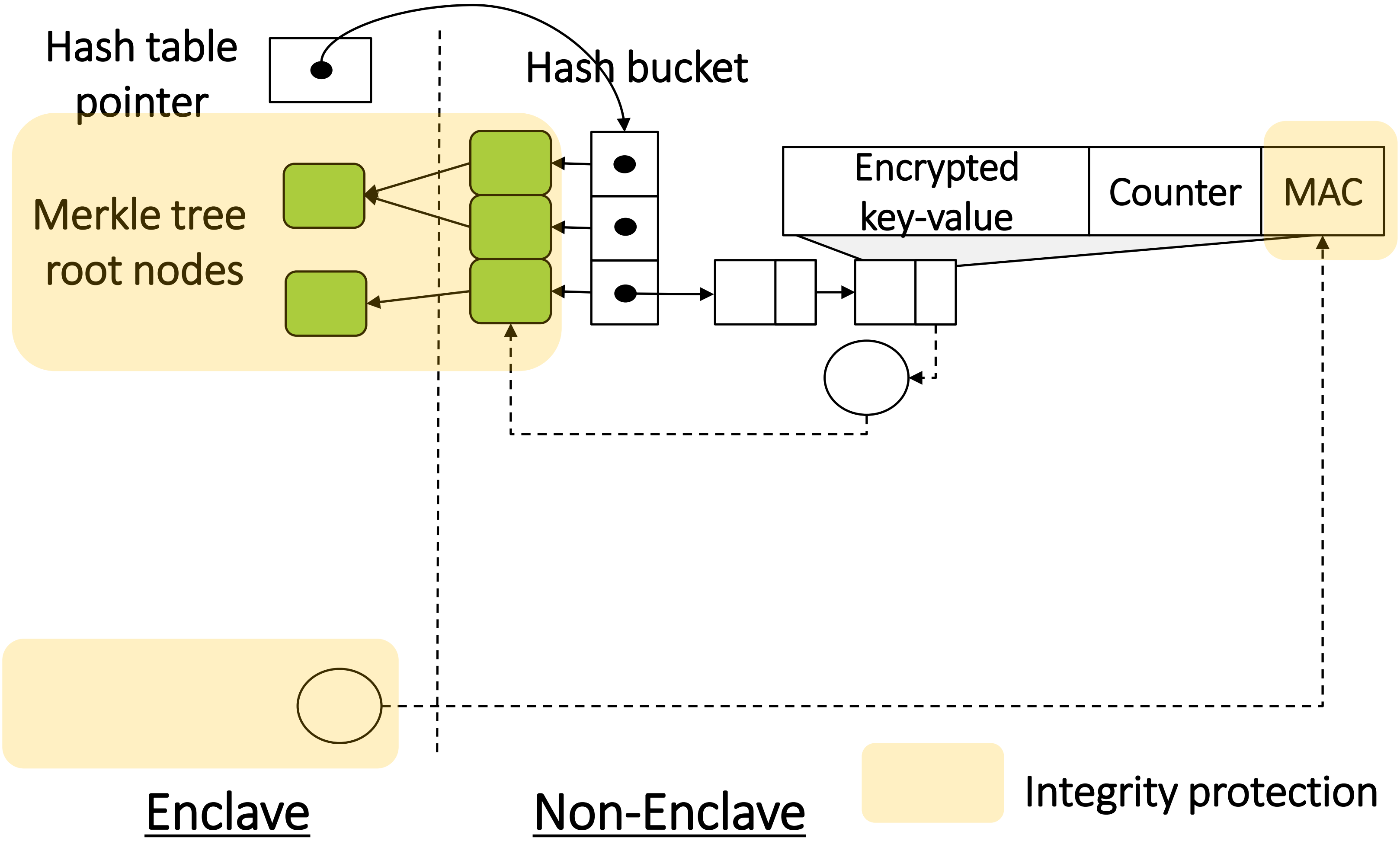
Non-Enclave

🟨 Integrity protection

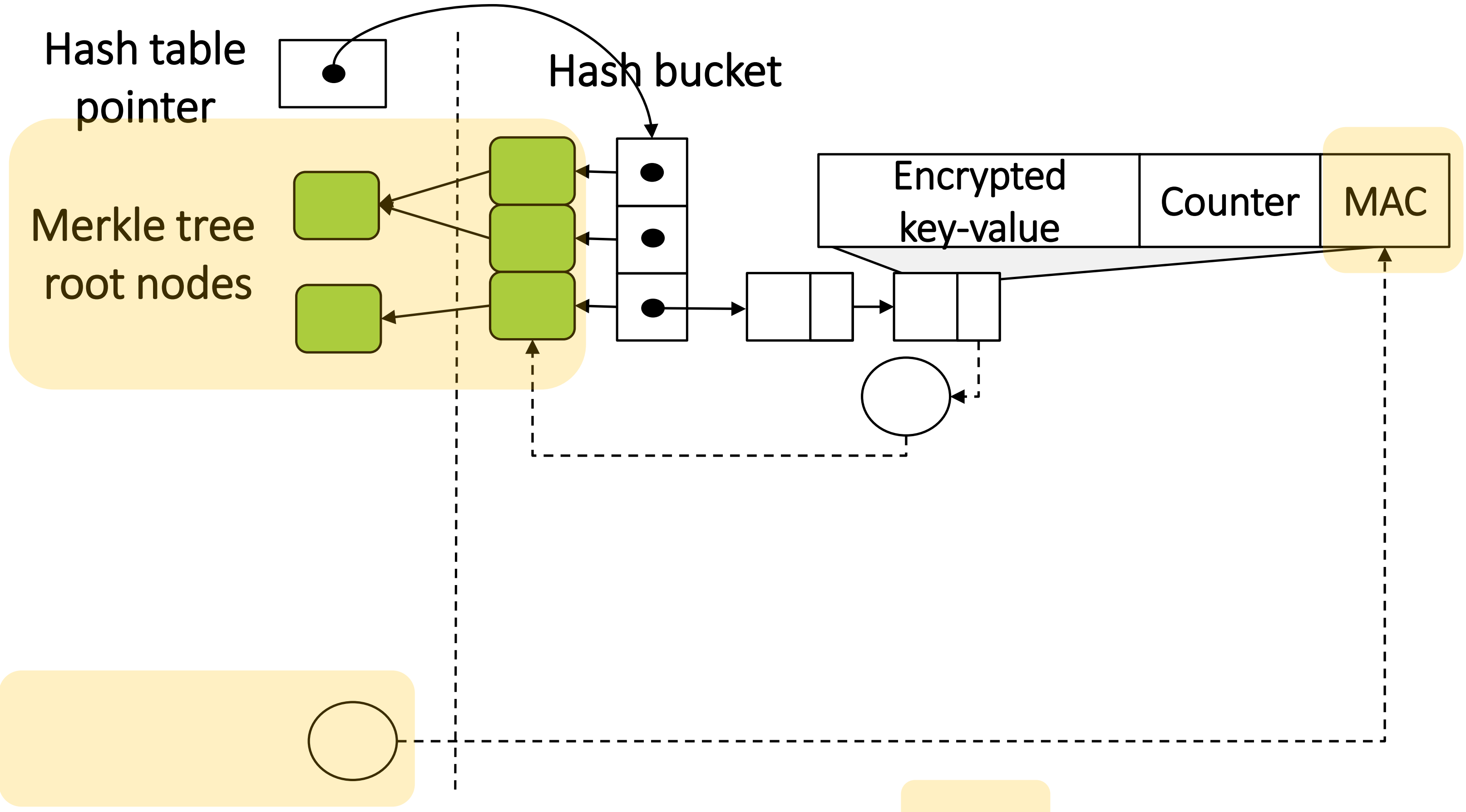
How To Protect Data?



How To Protect Data?



How To Protect Data?

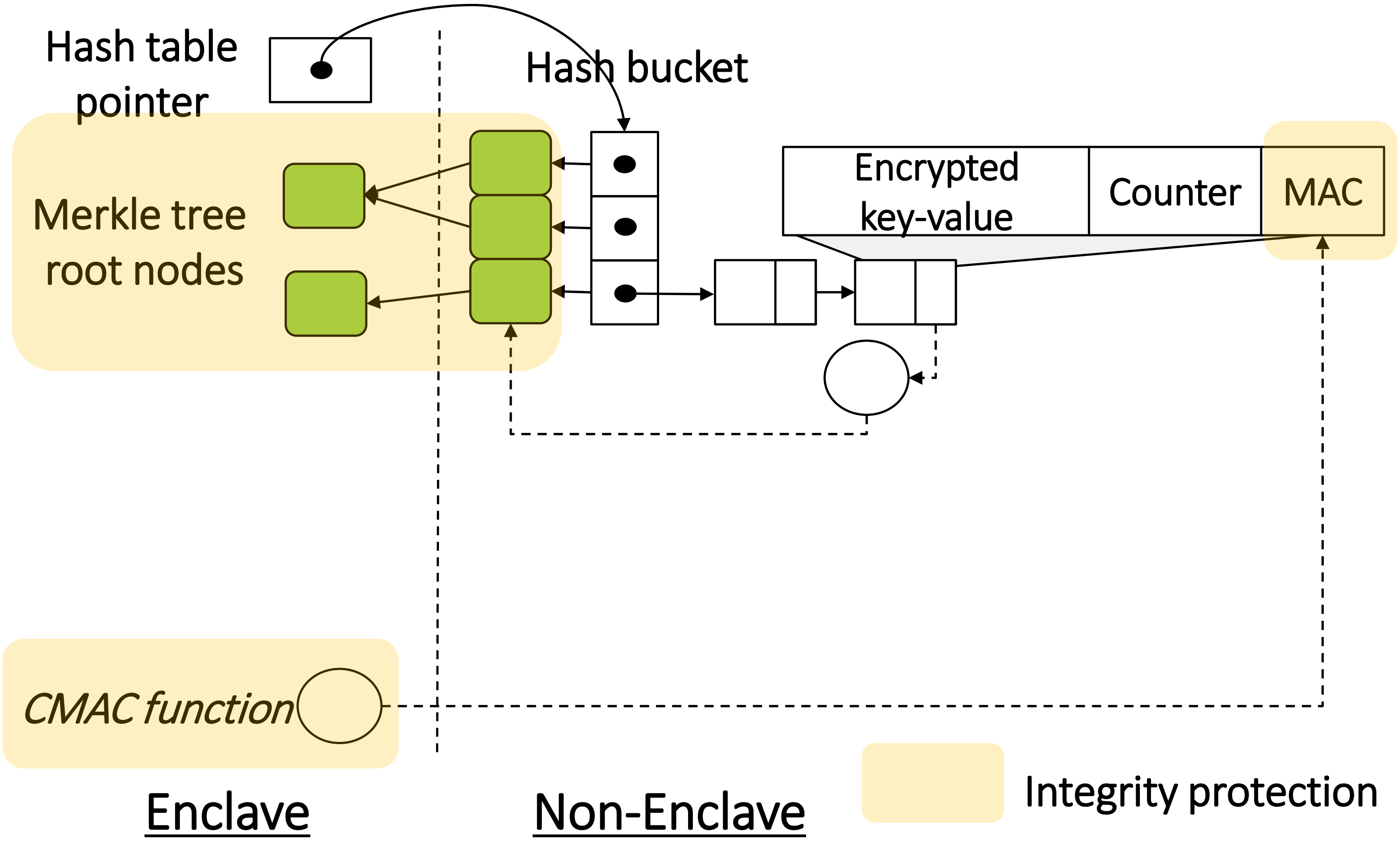


Enclave

Non-Enclave

🟨 Integrity protection

How To Protect Data?



Hash table pointer

Hash bucket

Merkle tree root nodes

Encrypted key-value

Counter

MAC

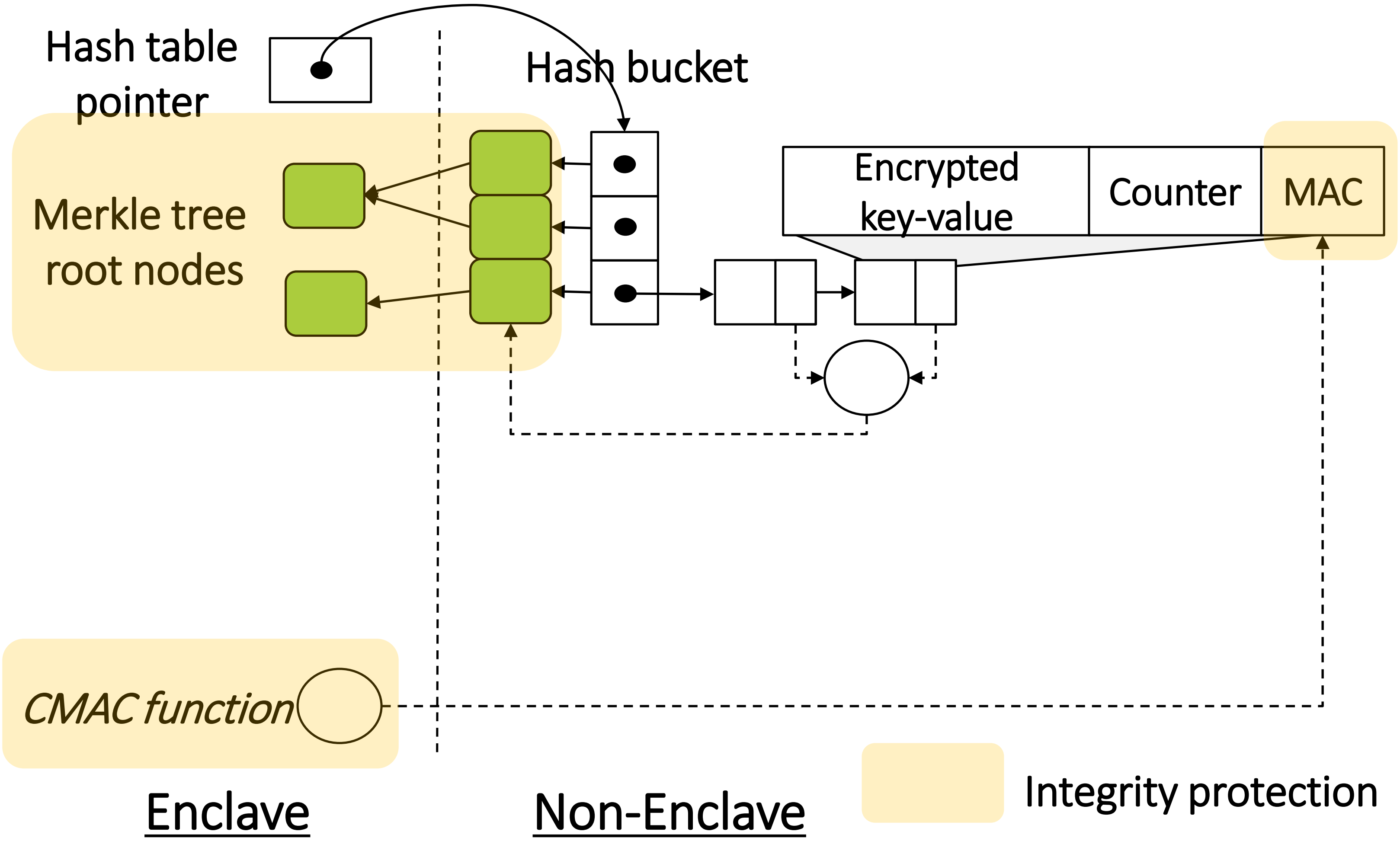
CMAC function

Enclave

Non-Enclave

🟨 Integrity protection

How To Protect Data?



Hash table pointer

Hash bucket

Merkle tree root nodes

Encrypted key-value

Counter

MAC

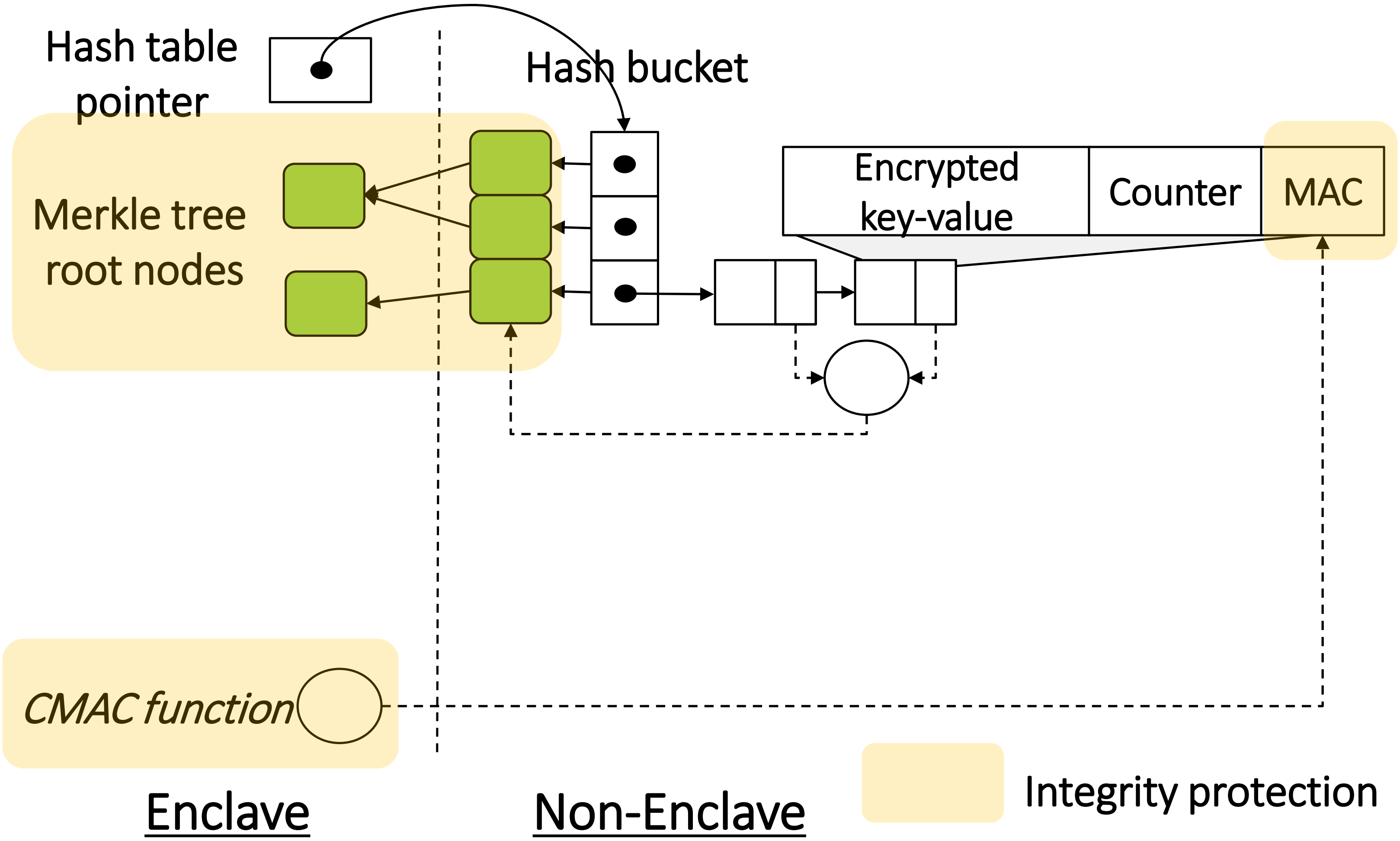
CMAC function

Enclave

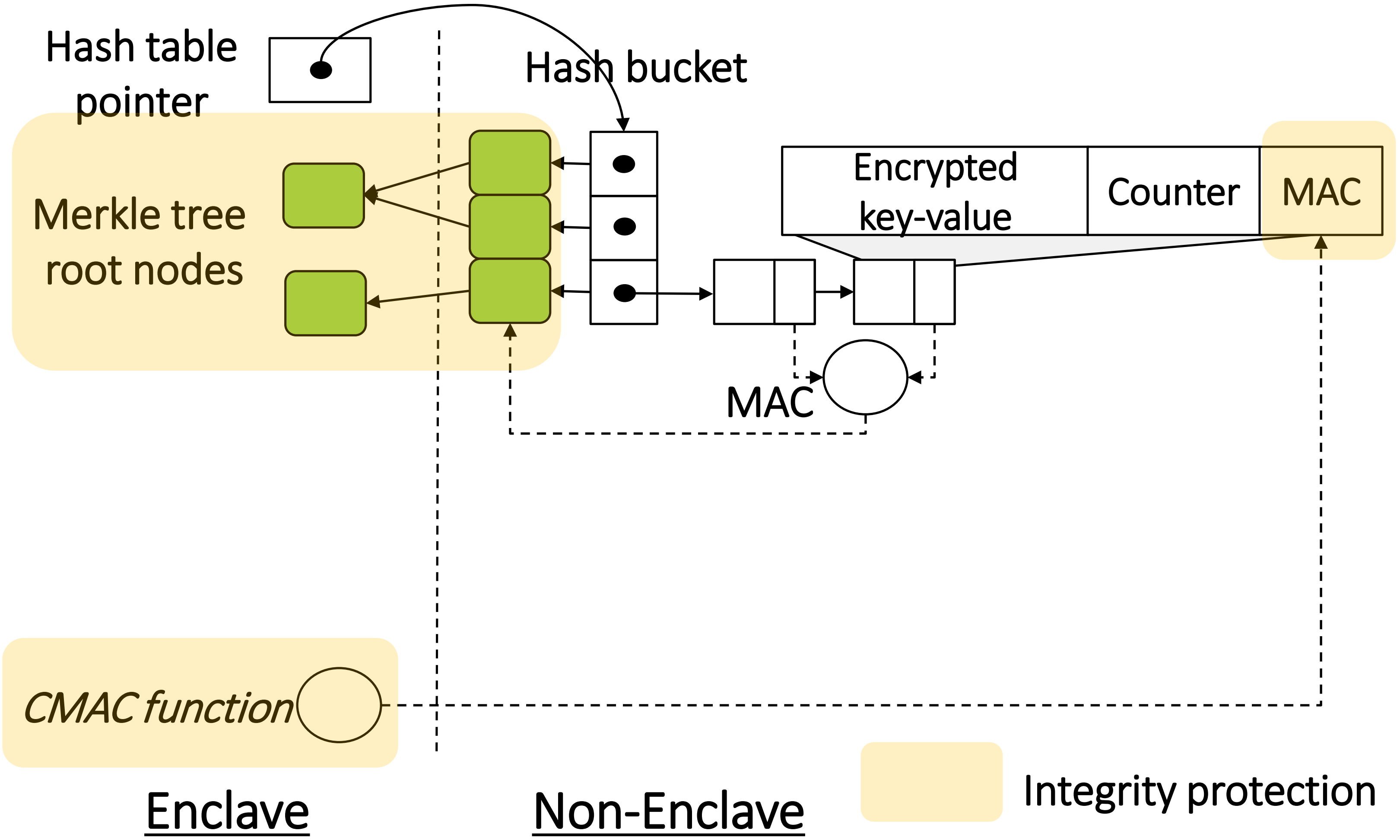
Non-Enclave

Integrity protection

How To Protect Data?



How To Protect Data?



Hash table pointer

Hash bucket

Merkle tree root nodes

Encrypted key-value

Counter

MAC

MAC

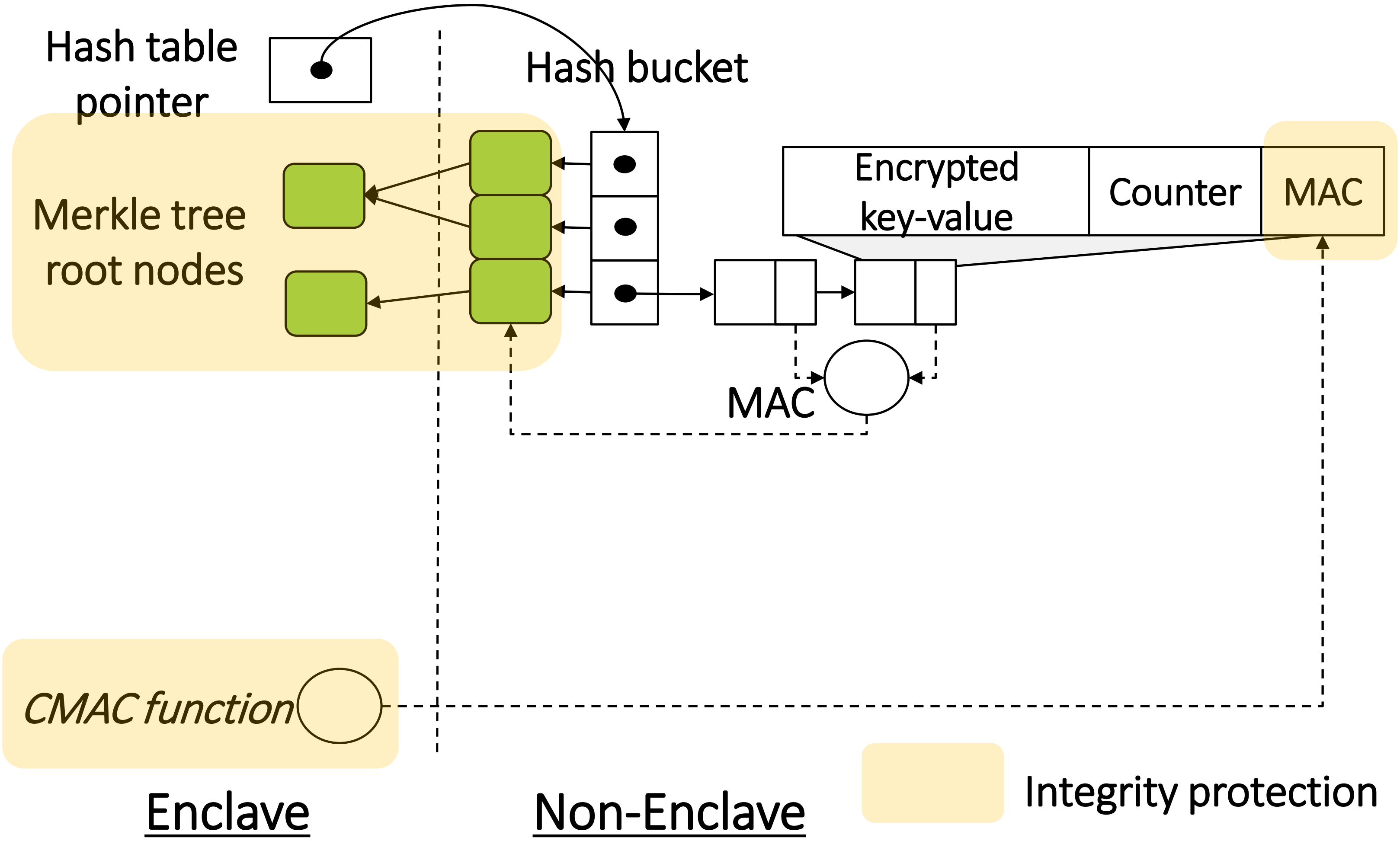
CMAC function

Enclave

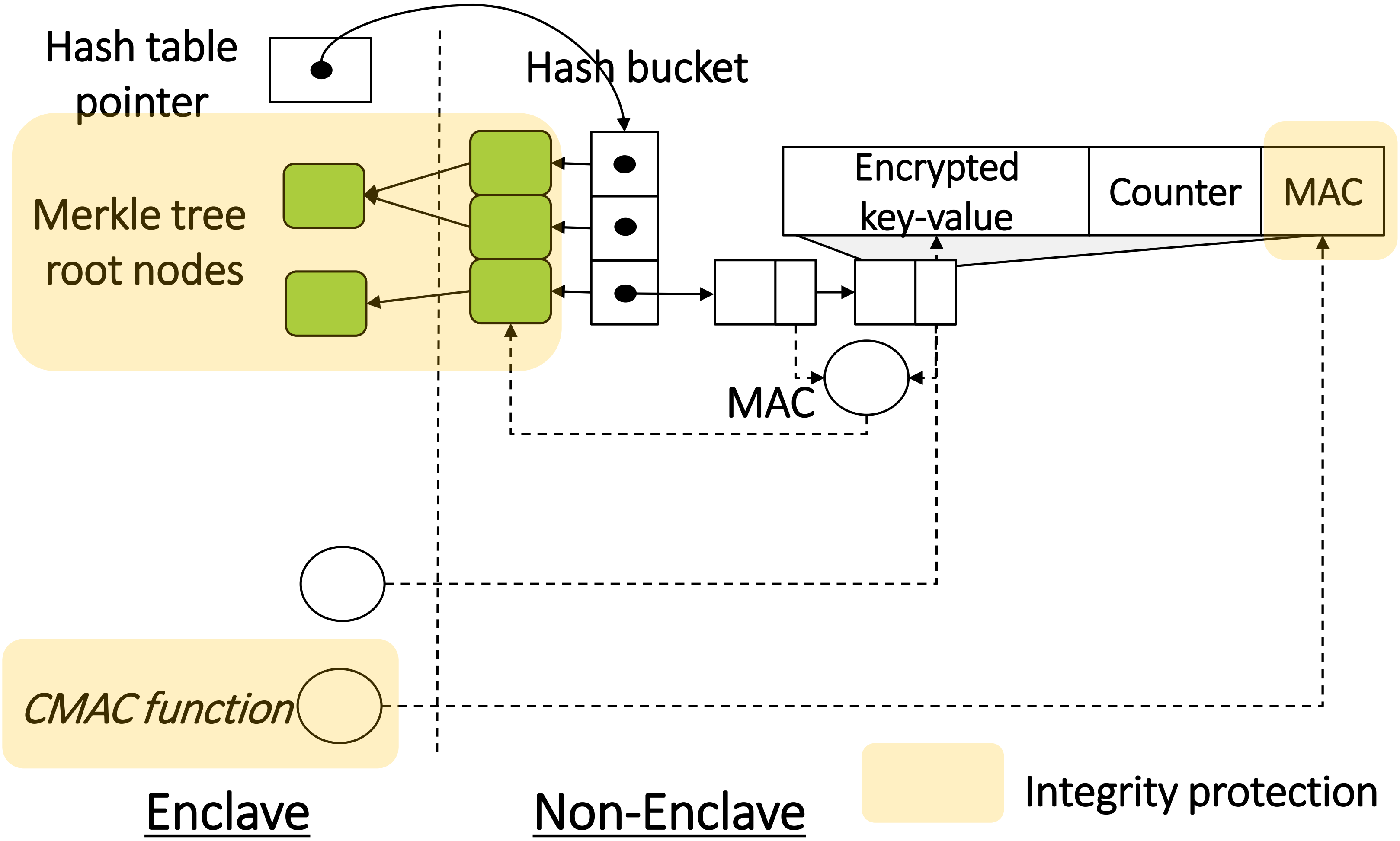
Non-Enclave

🟨 Integrity protection

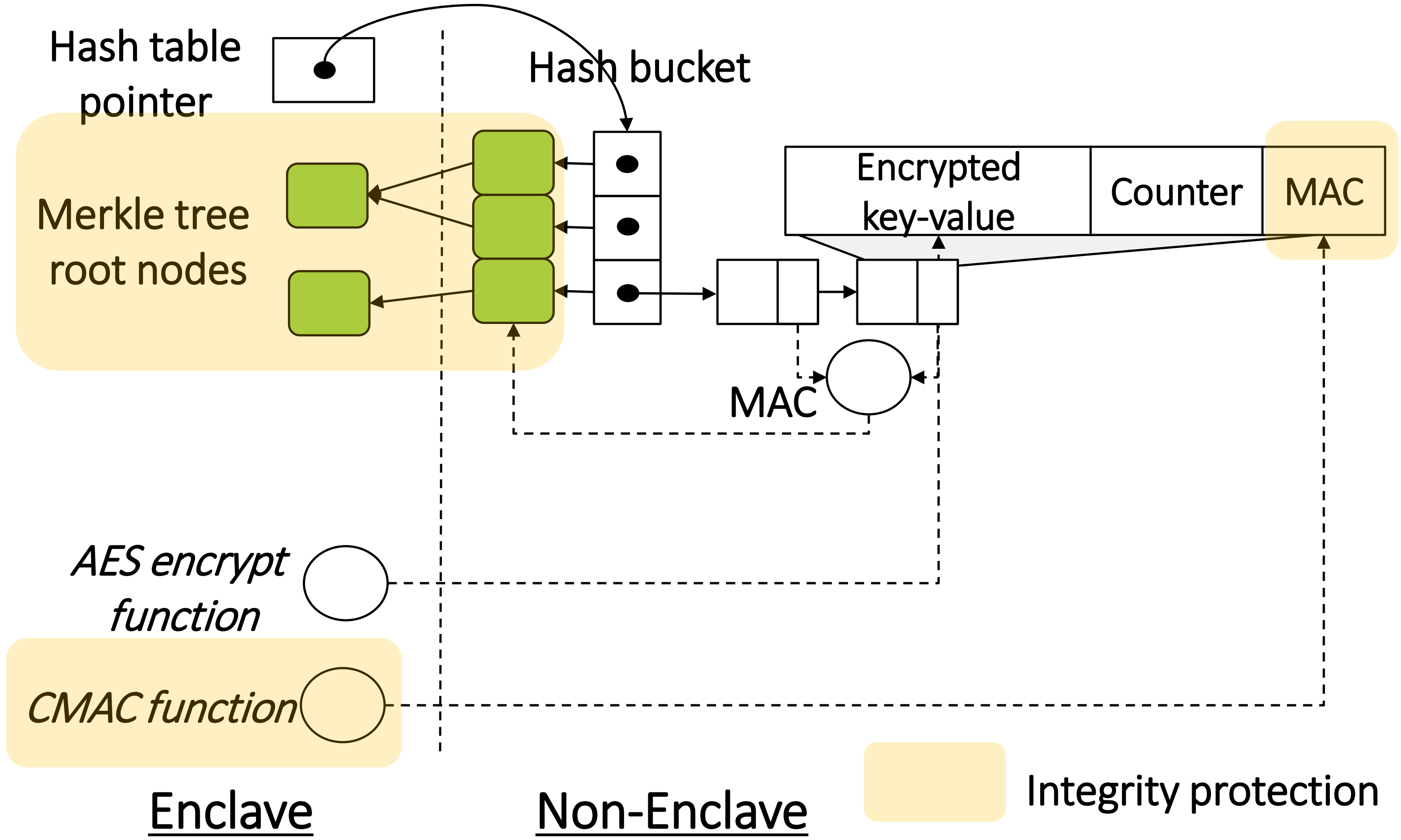
How To Protect Data?



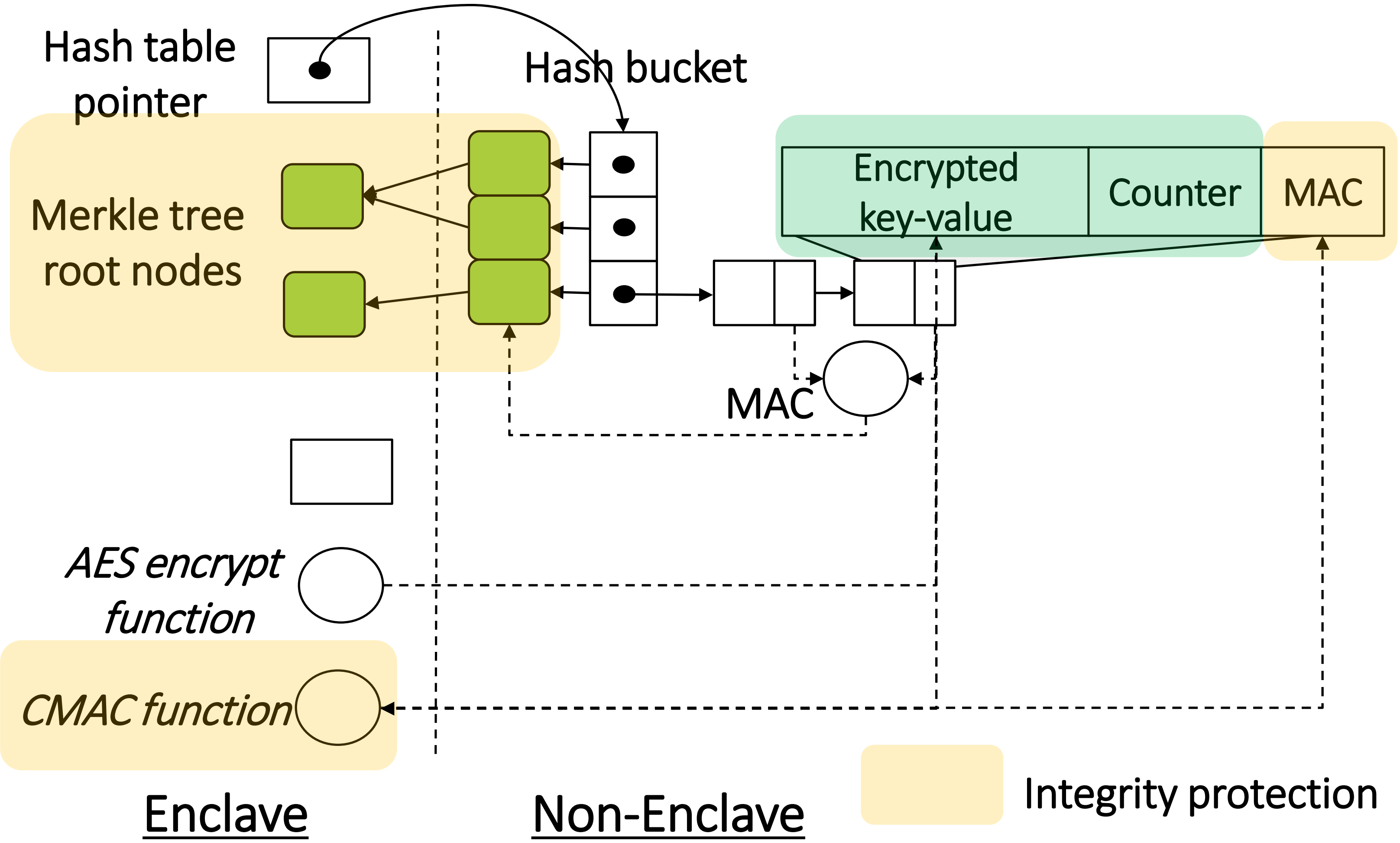
How To Protect Data?



How To Protect Data?



How To Protect Data?



Hash table pointer

Hash bucket

Merkle tree root nodes

Encrypted key-value

Counter

MAC

MAC

AES encrypt function

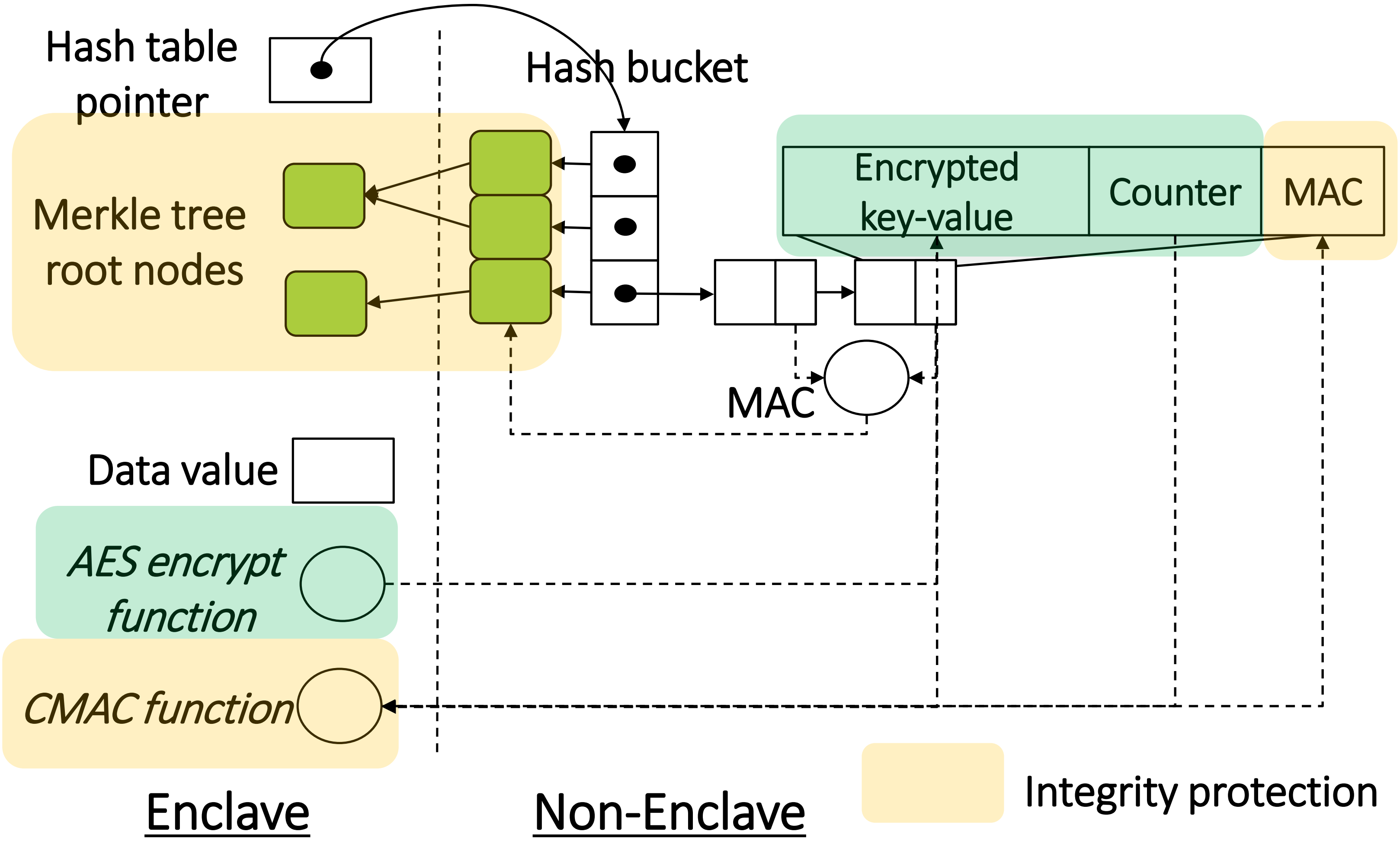
CMAC function

Enclave

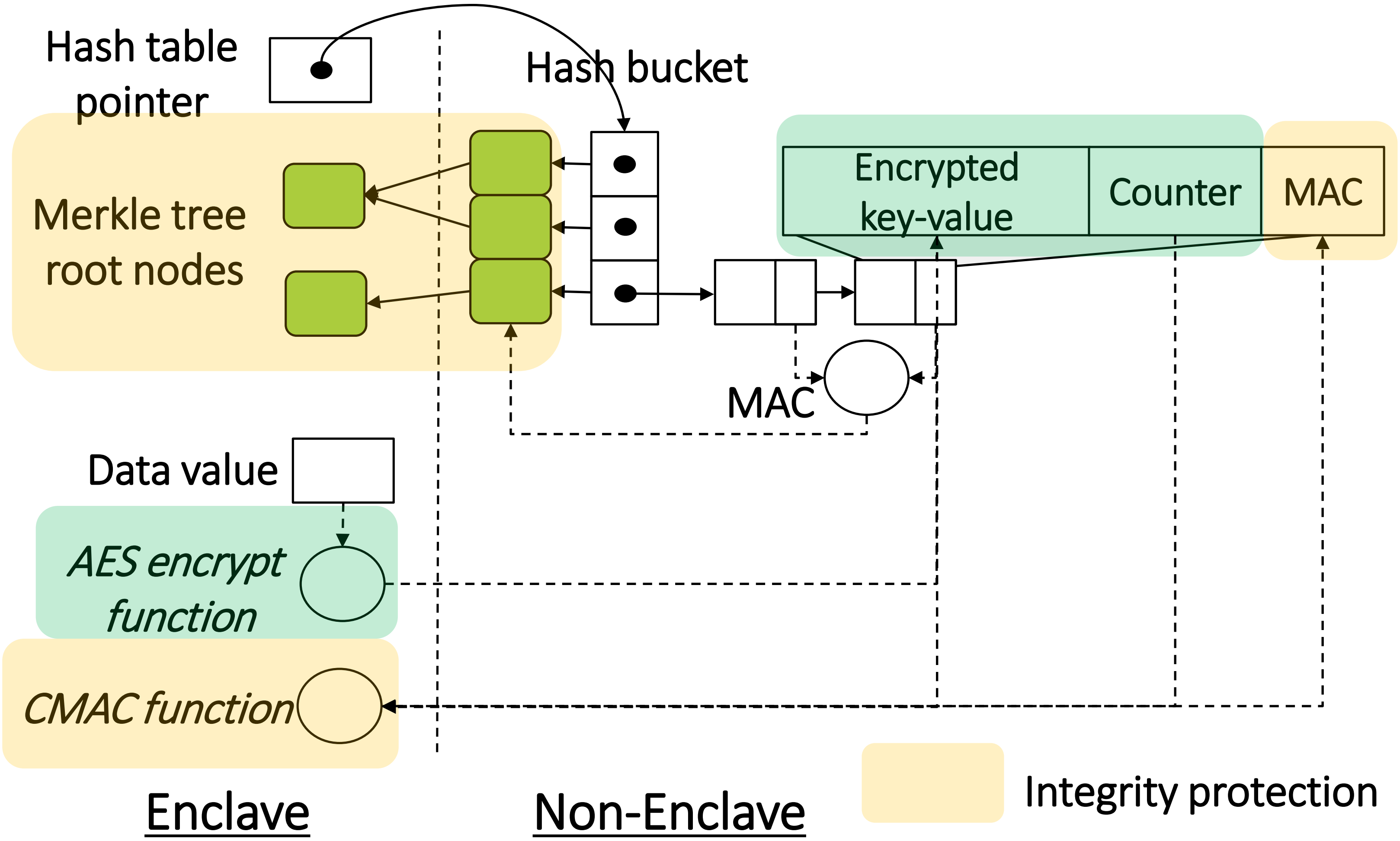
Non-Enclave

🟨 Integrity protection

How To Protect Data?

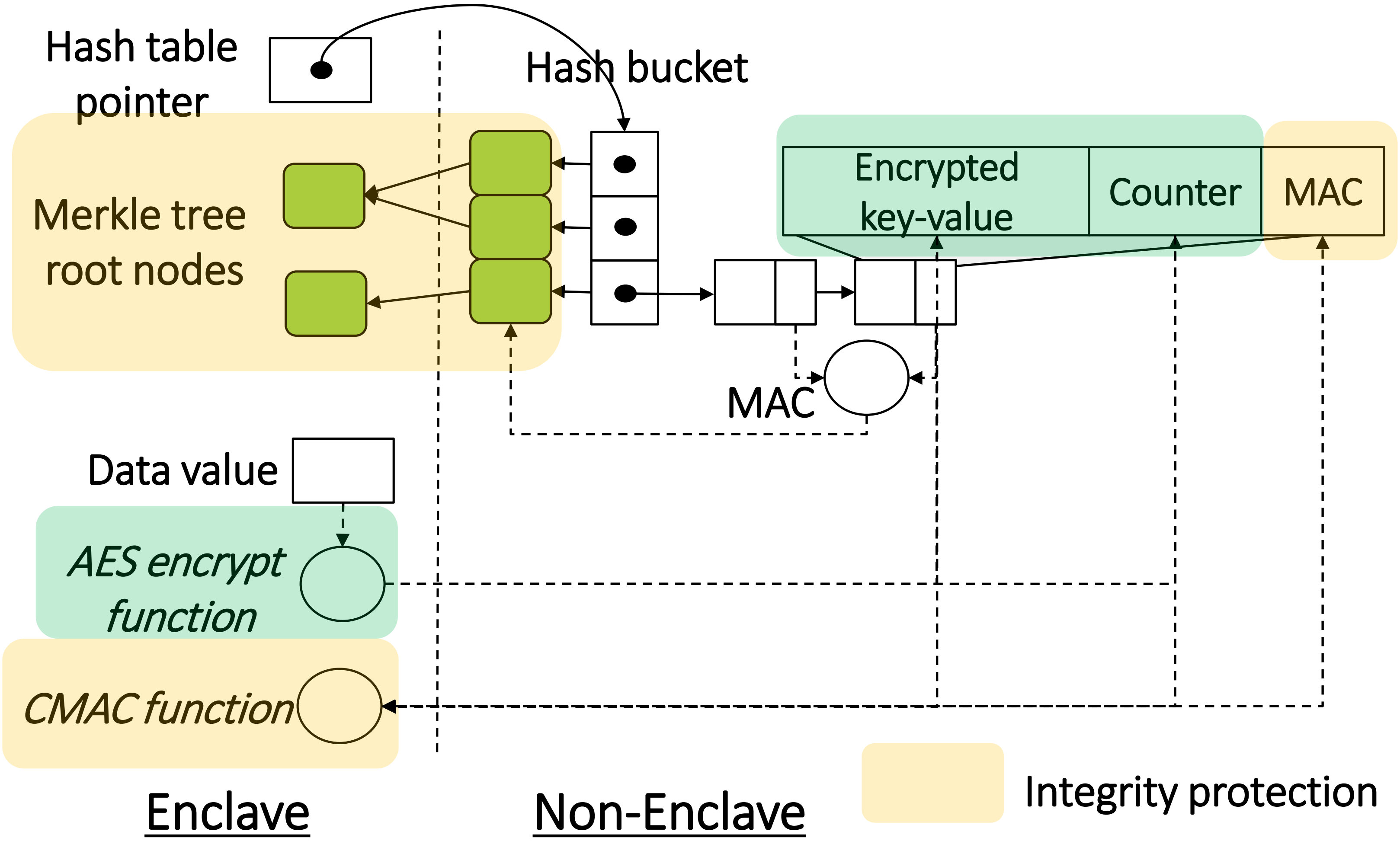


How To Protect Data?



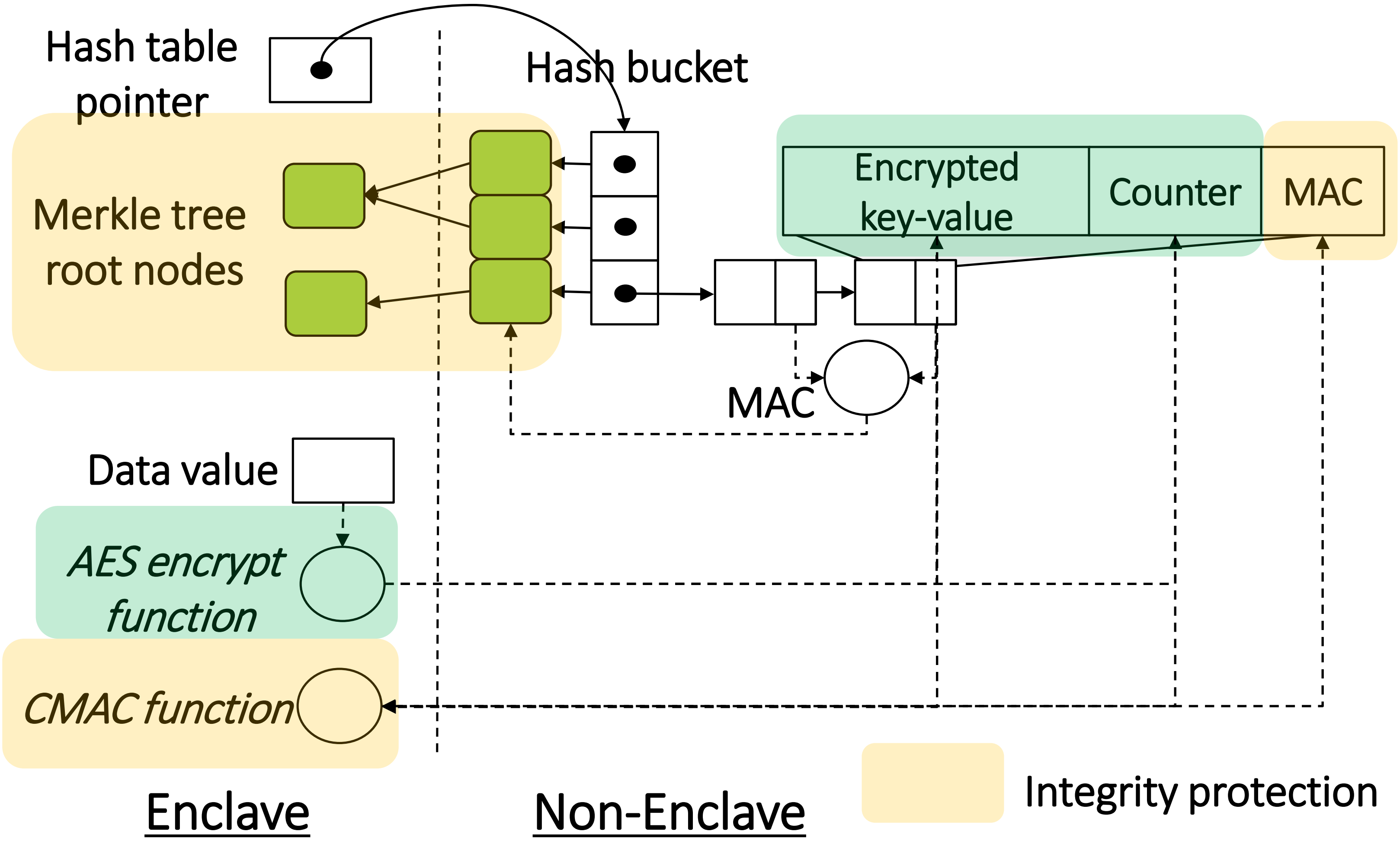
🟨 Integrity protection

How To Protect Data?

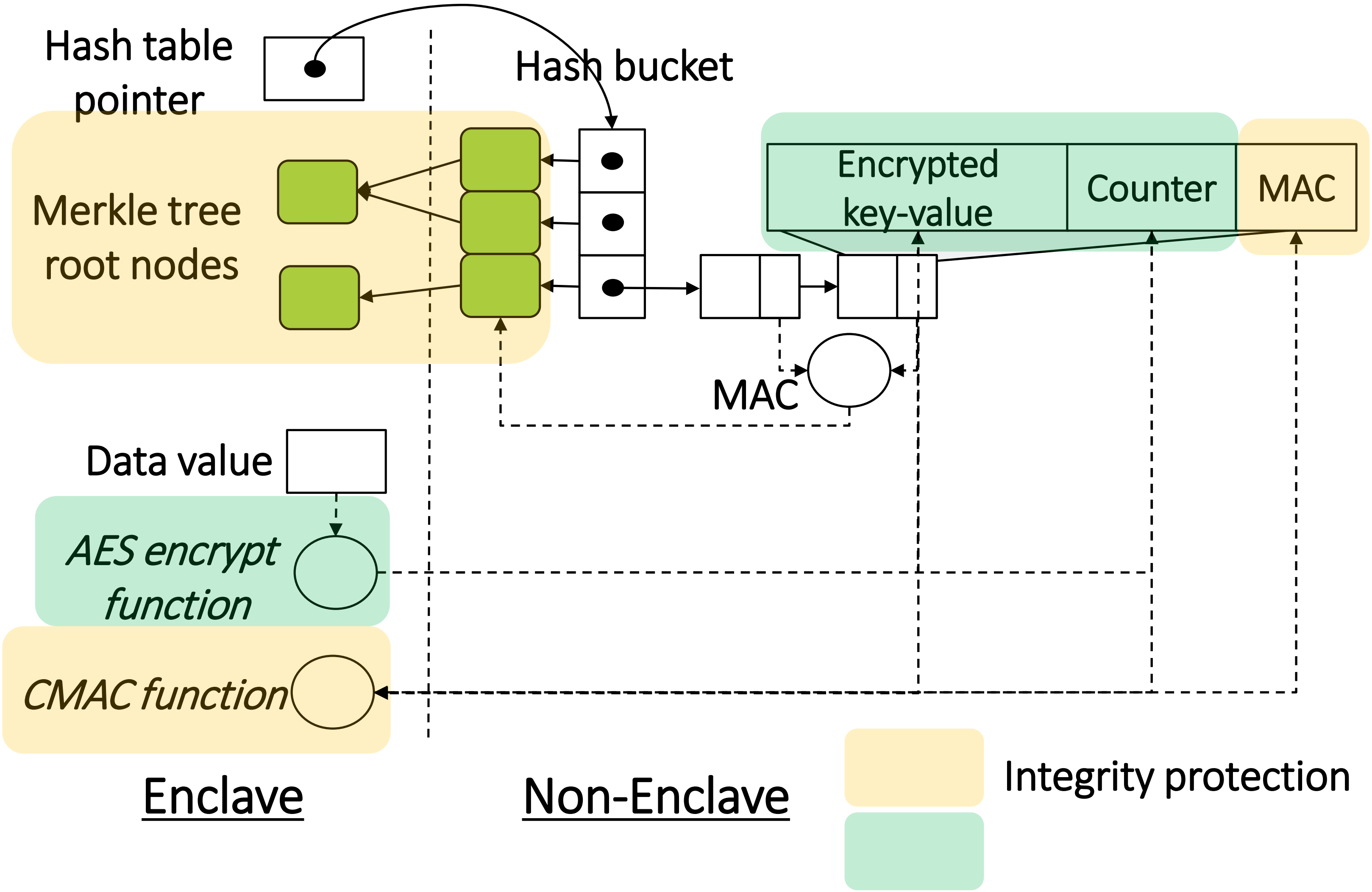


🟨 Integrity protection

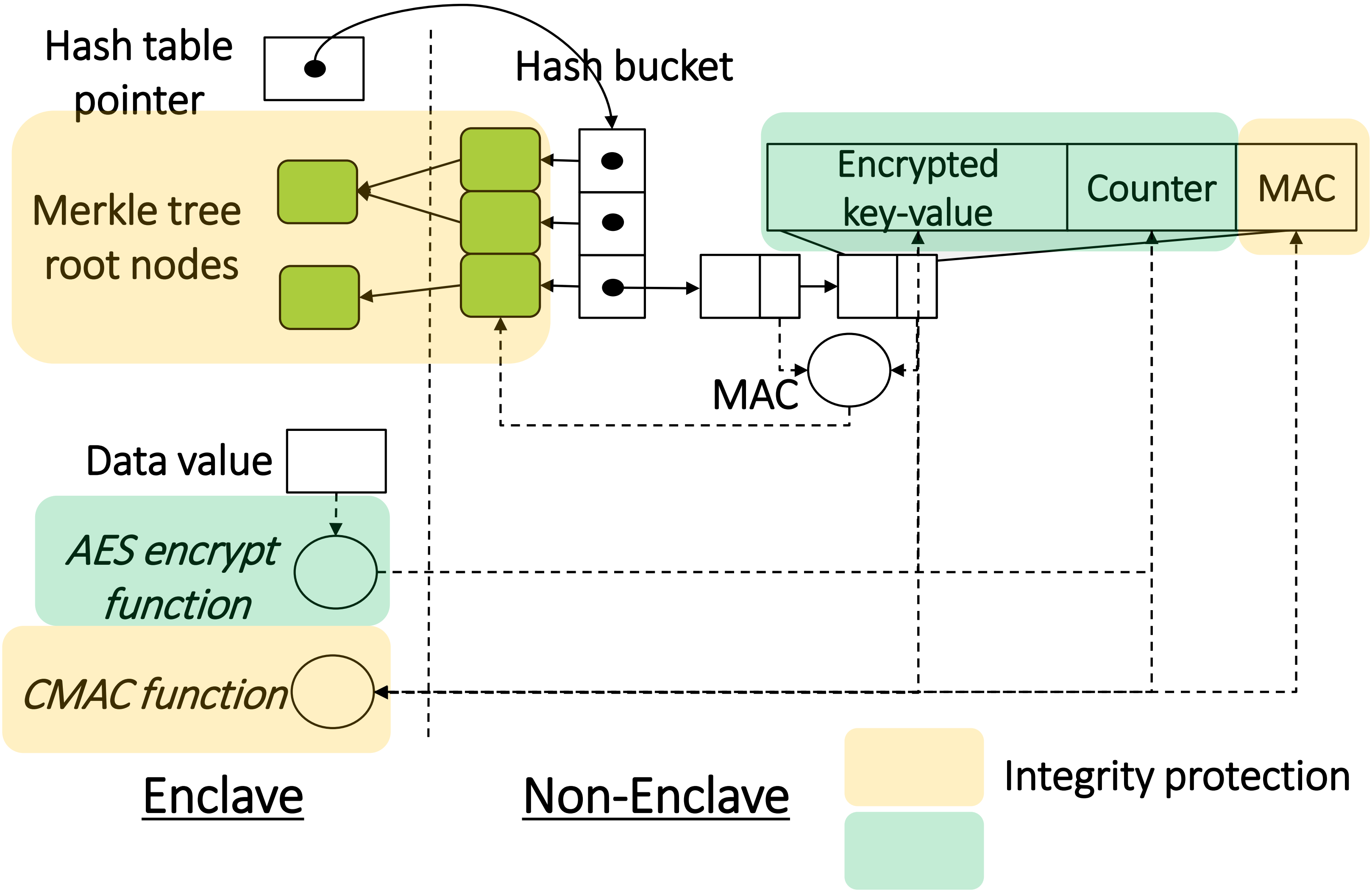
How To Protect Data?



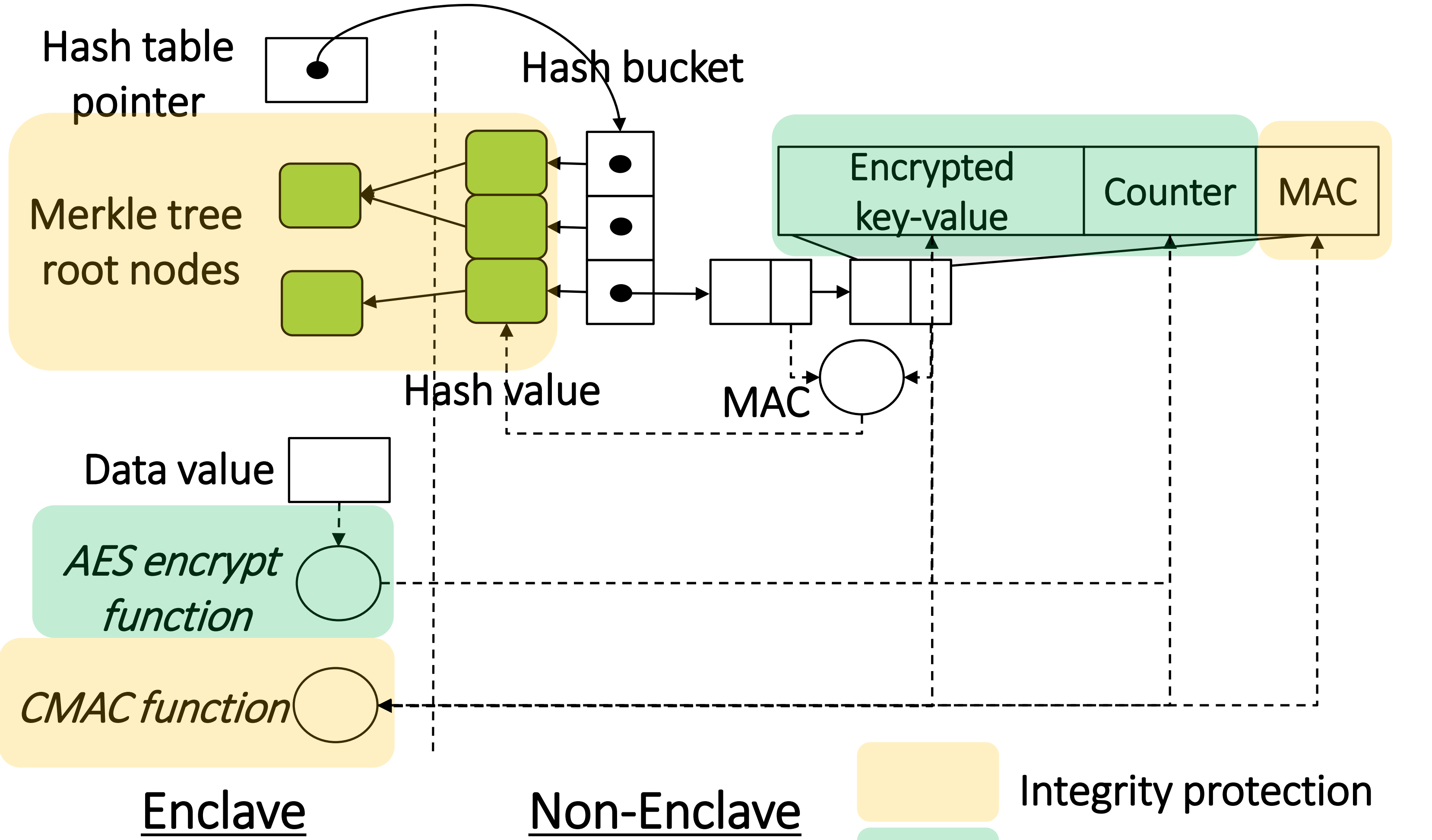
How To Protect Data?

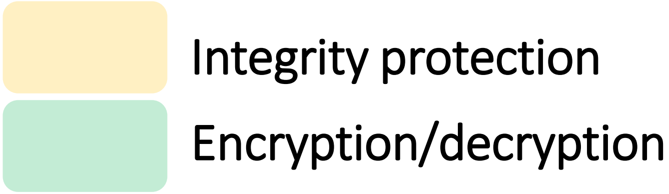


How To Protect Data?

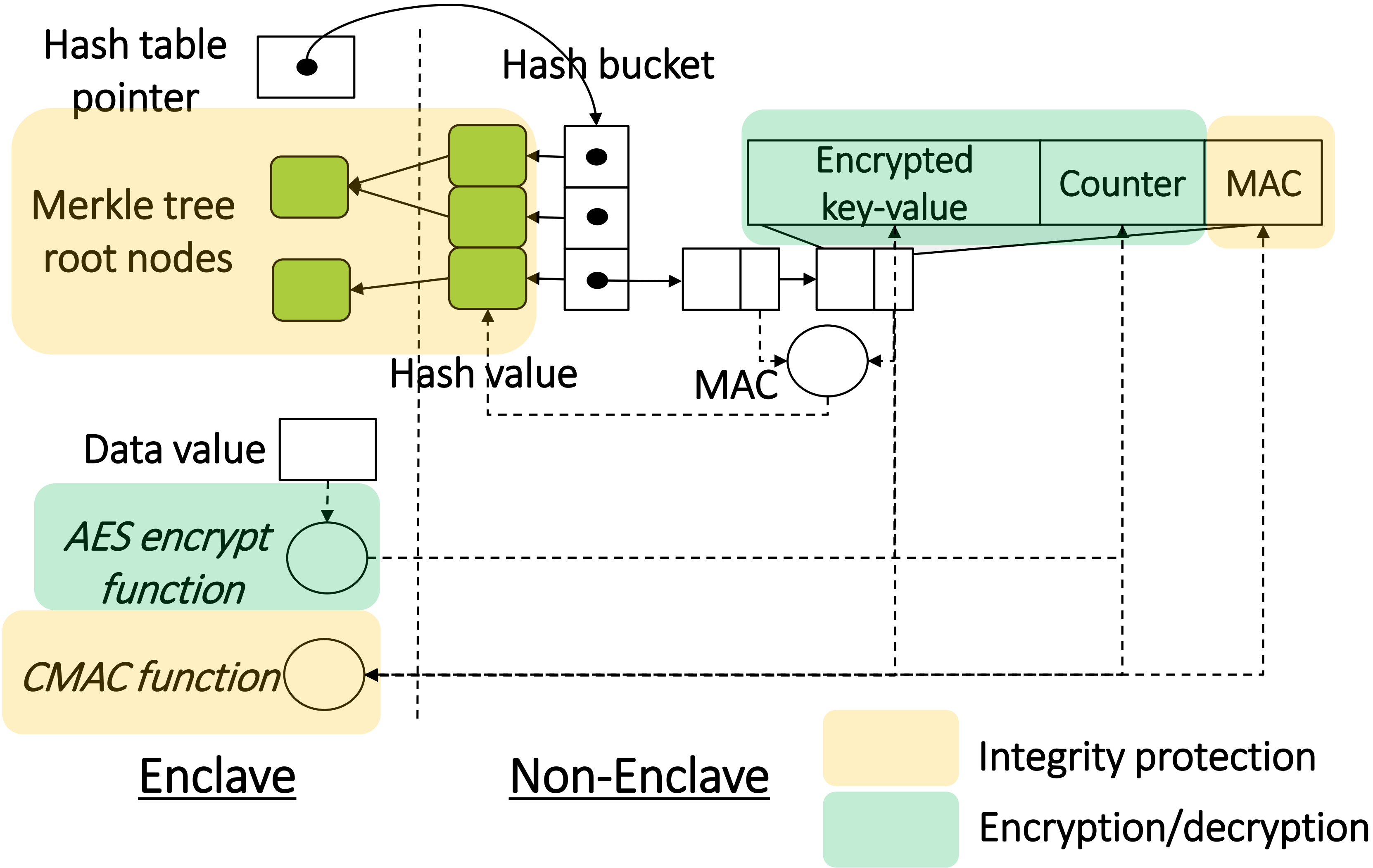


How To Protect Data?

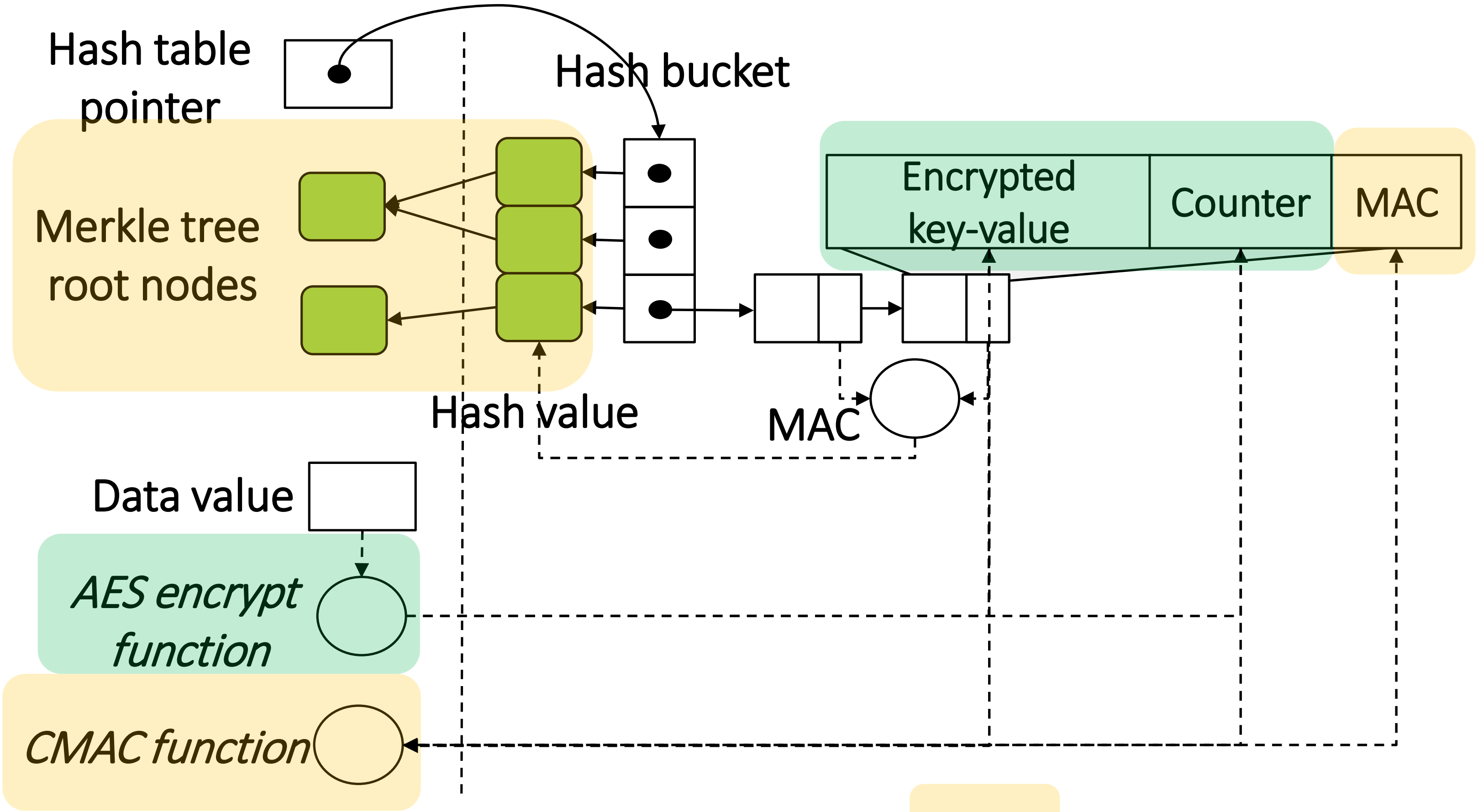




How To Protect Data?



How To Protect Data?



Hash table pointer

Hash bucket

Merkle tree root nodes

Encrypted key-value

Counter

MAC

Hash value

MAC

Data value

AES encrypt function

CMAC function

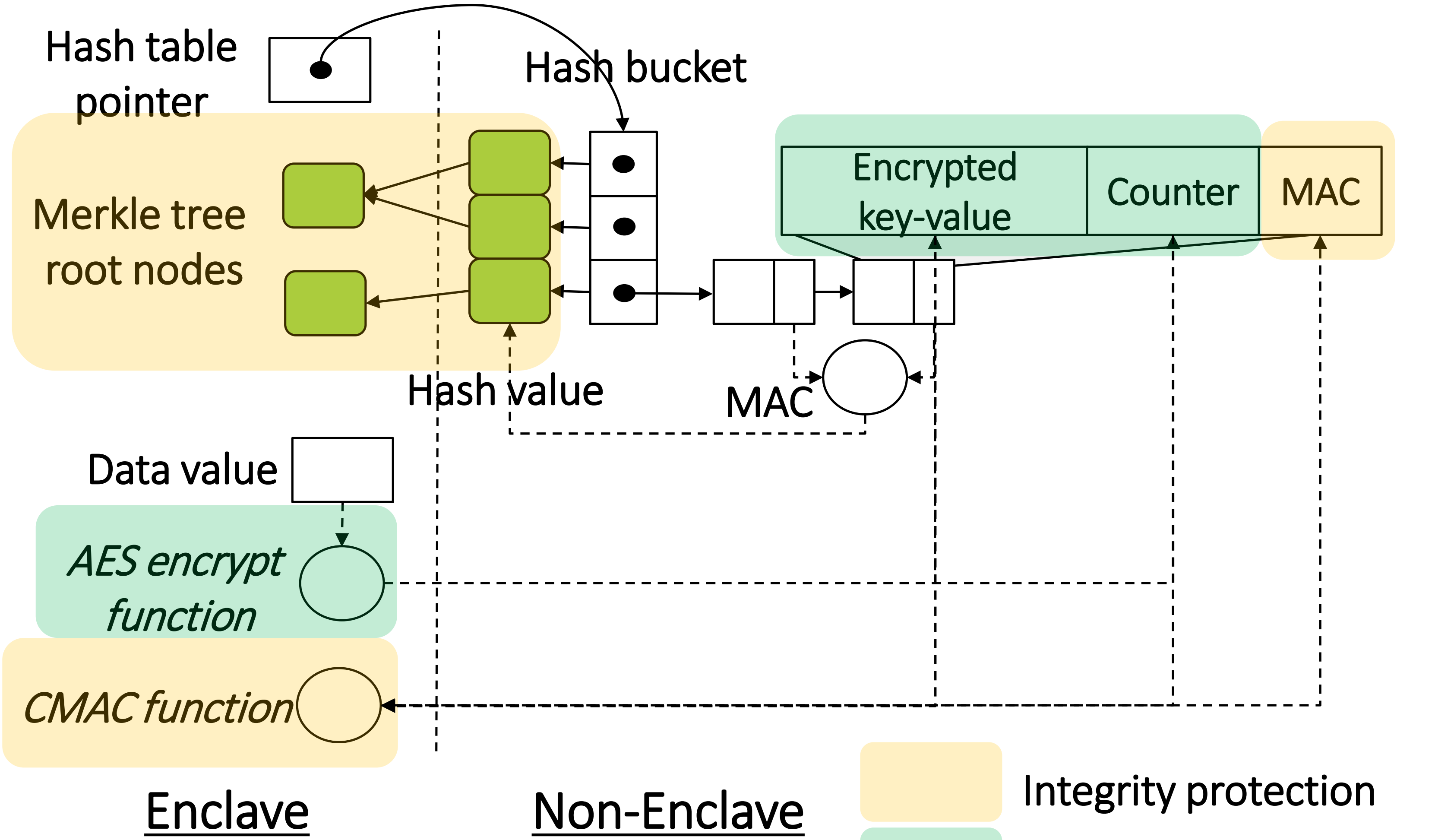
Enclave

Non-Enclave

🟨 Integrity protection

🟩 Encryption/decryption

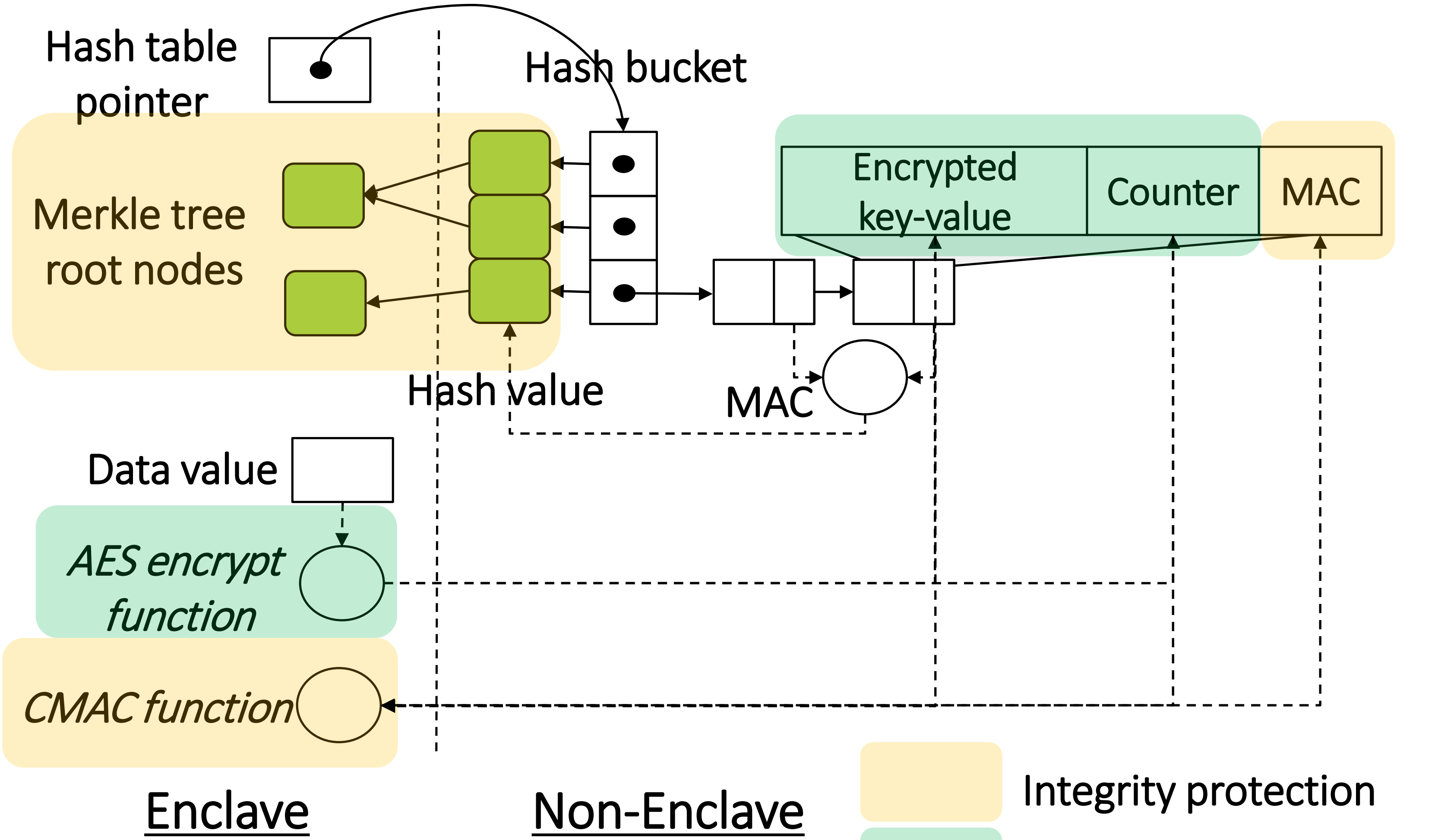
How To Protect Data?



🟨 Integrity protection

🟩 Encryption/decryption

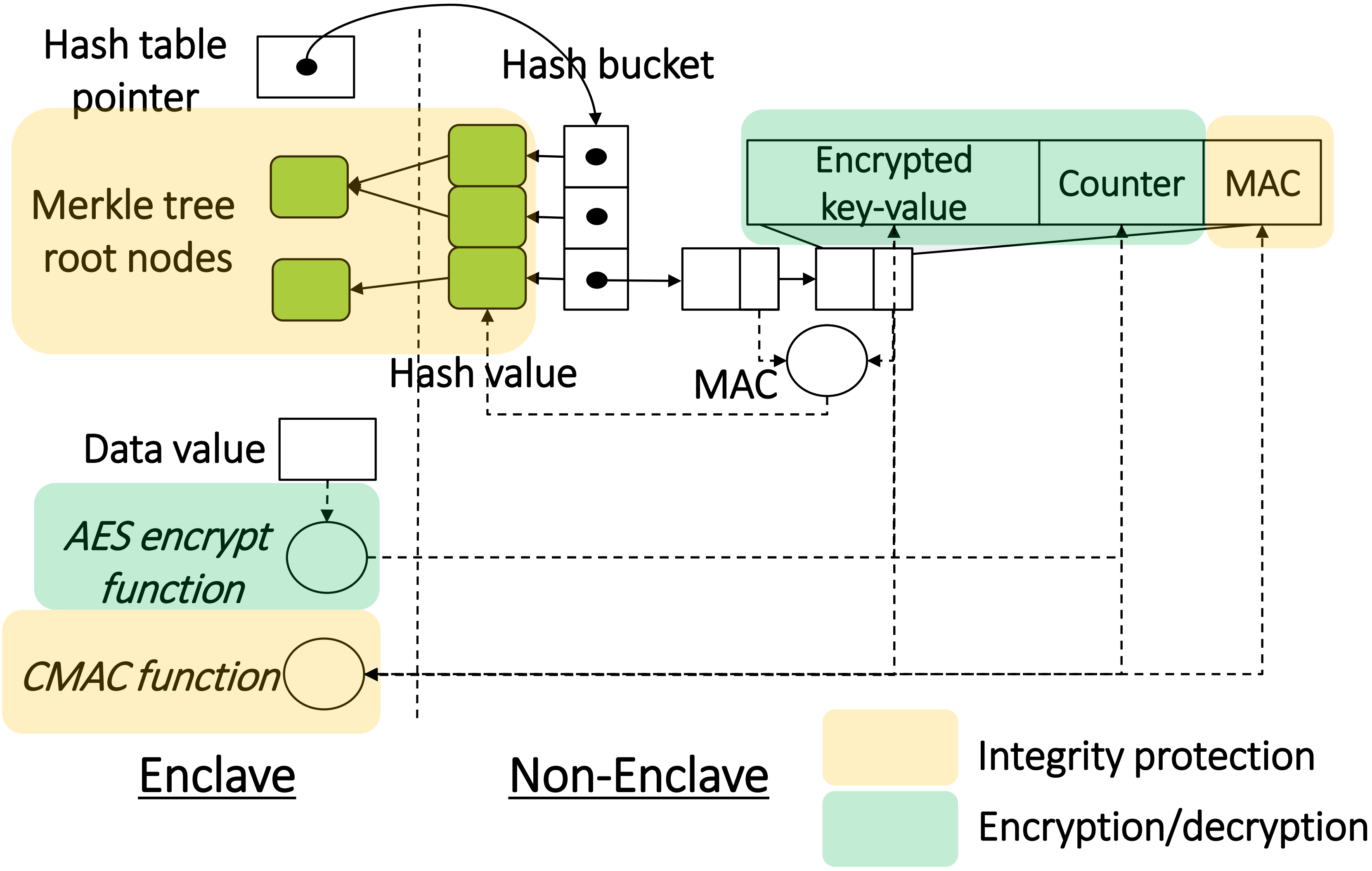
How To Protect Data?



🟨 Integrity protection

🟩 Encryption/decryption

How To Protect Data?



Hash table pointer

Hash bucket

Merkle tree root nodes

Encrypted key-value

Counter

MAC

Hash value

MAC

Data value

AES encrypt function

CMAC function

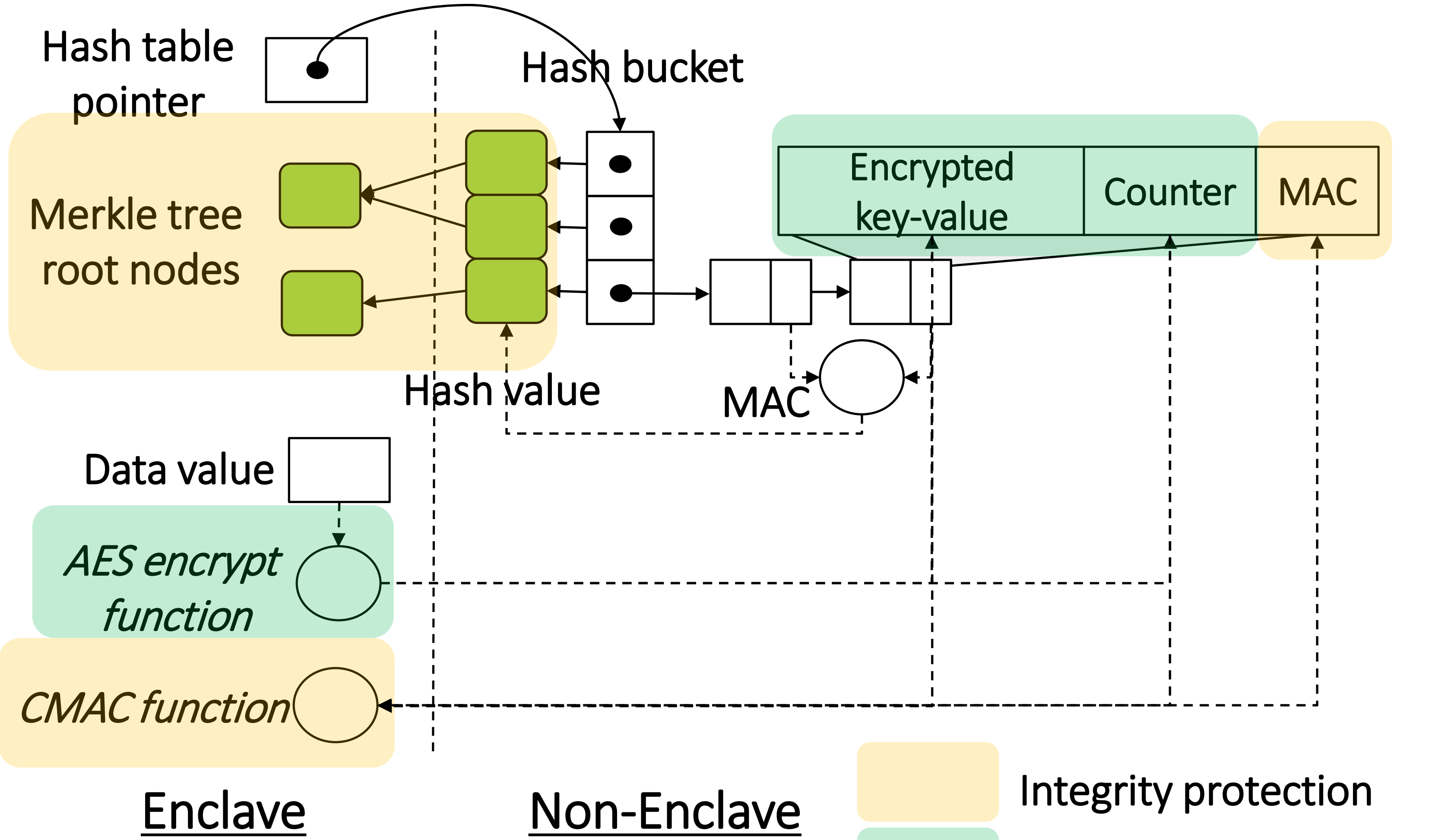
Enclave

Non-Enclave

Integrity protection

Encryption/decryption

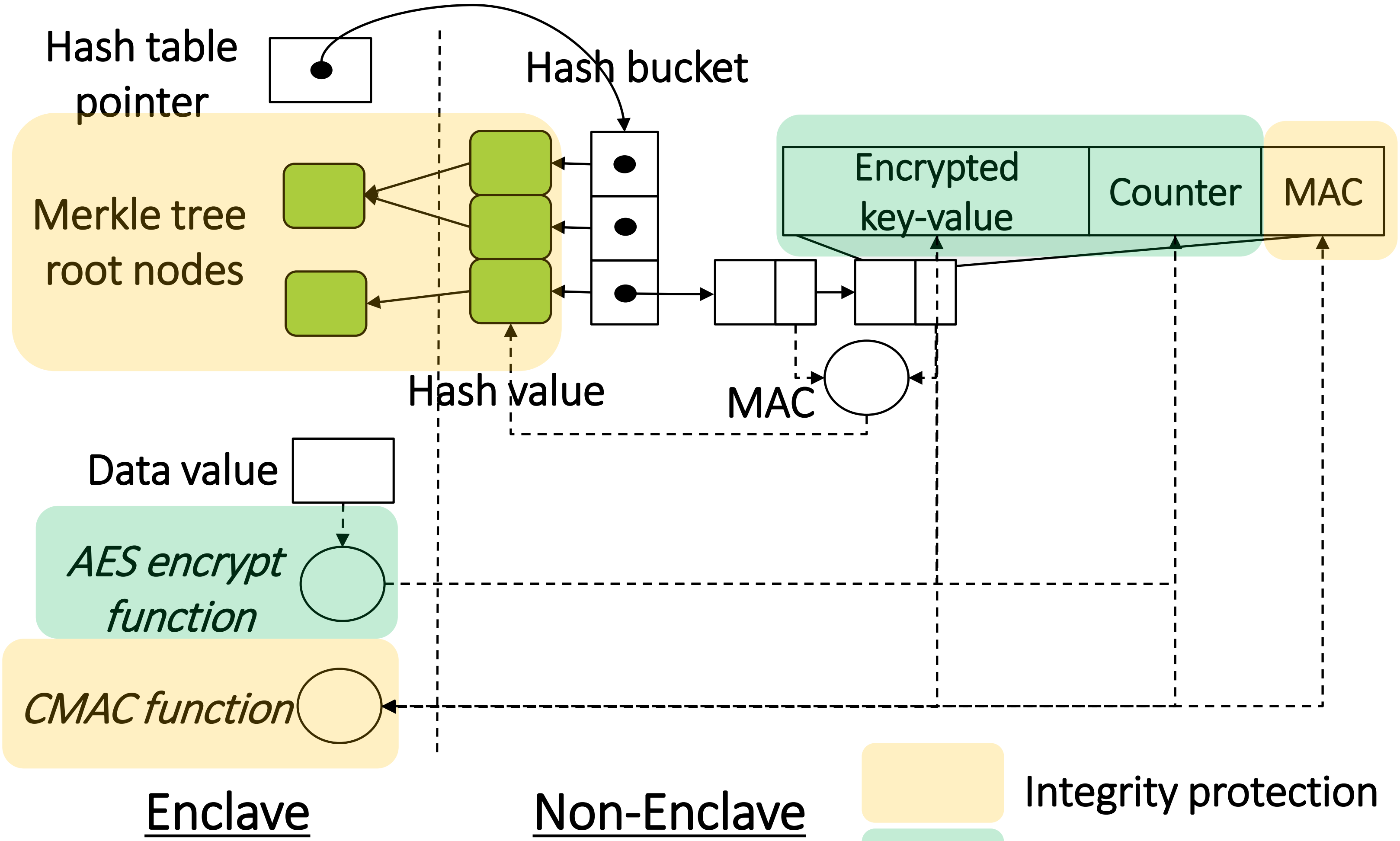
How To Protect Data?



🟨 Integrity protection

🟩 Encryption/decryption

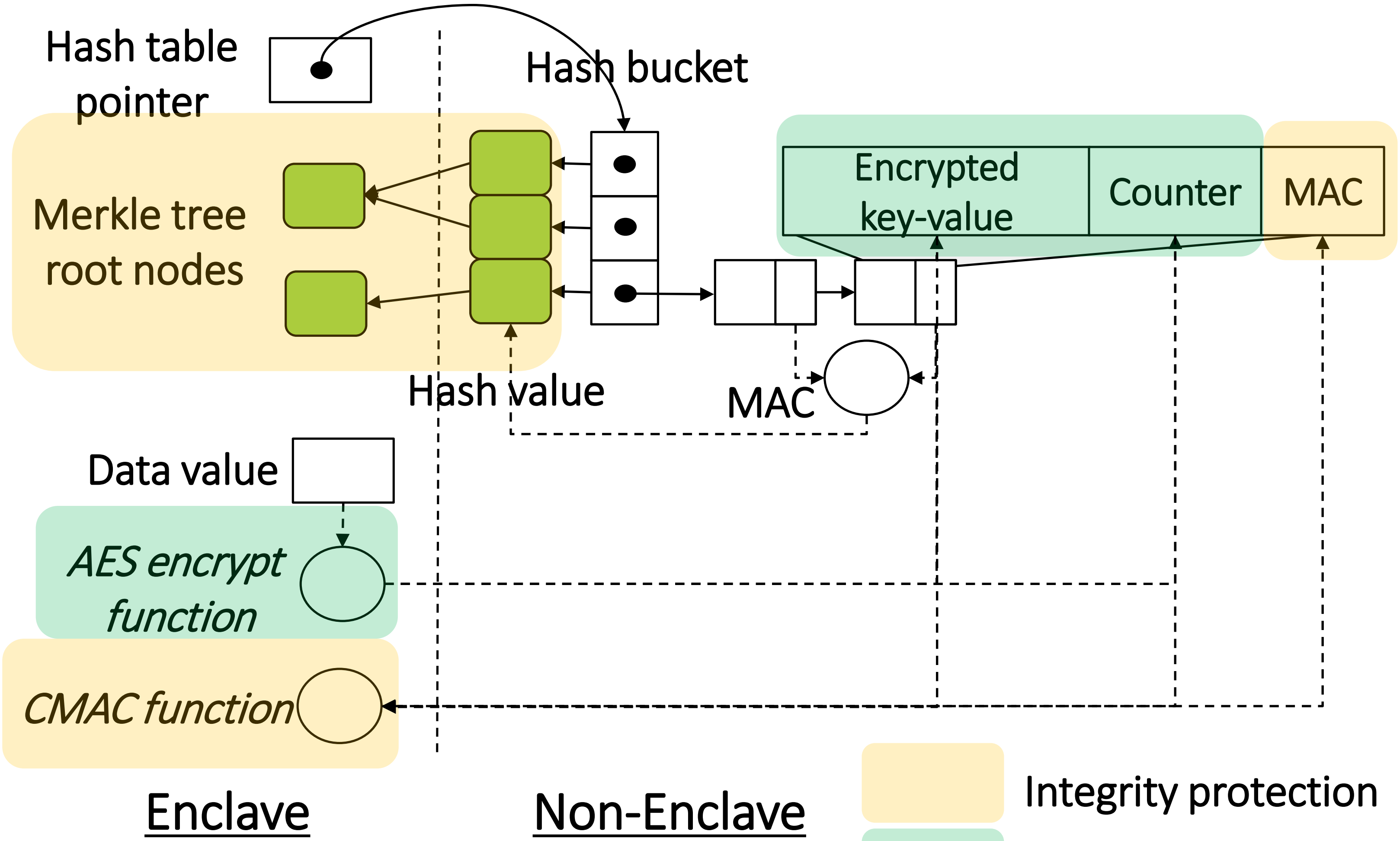
How To Protect Data?



🟨 Integrity protection

🟩 Encryption/decryption

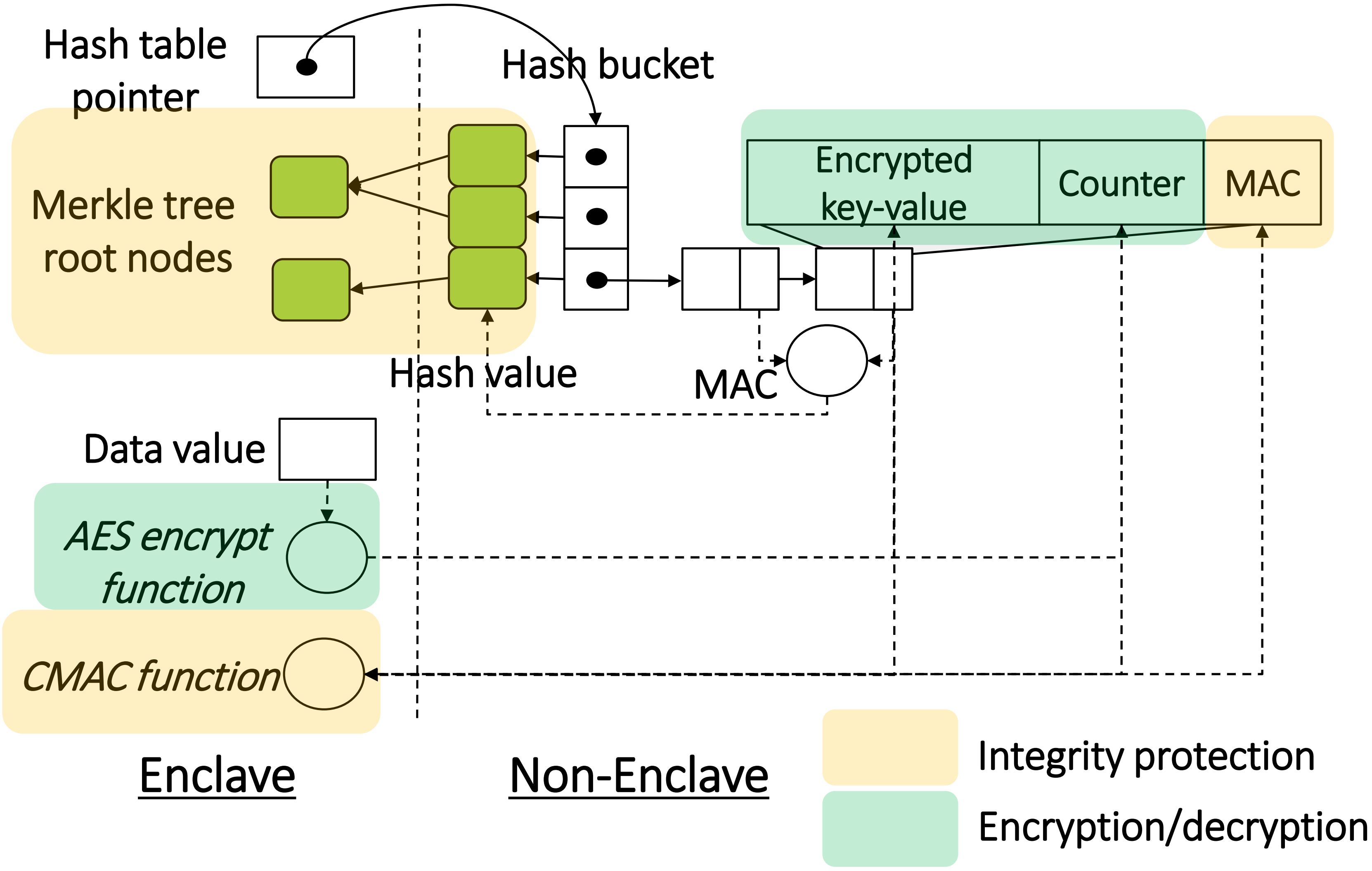
How To Protect Data?



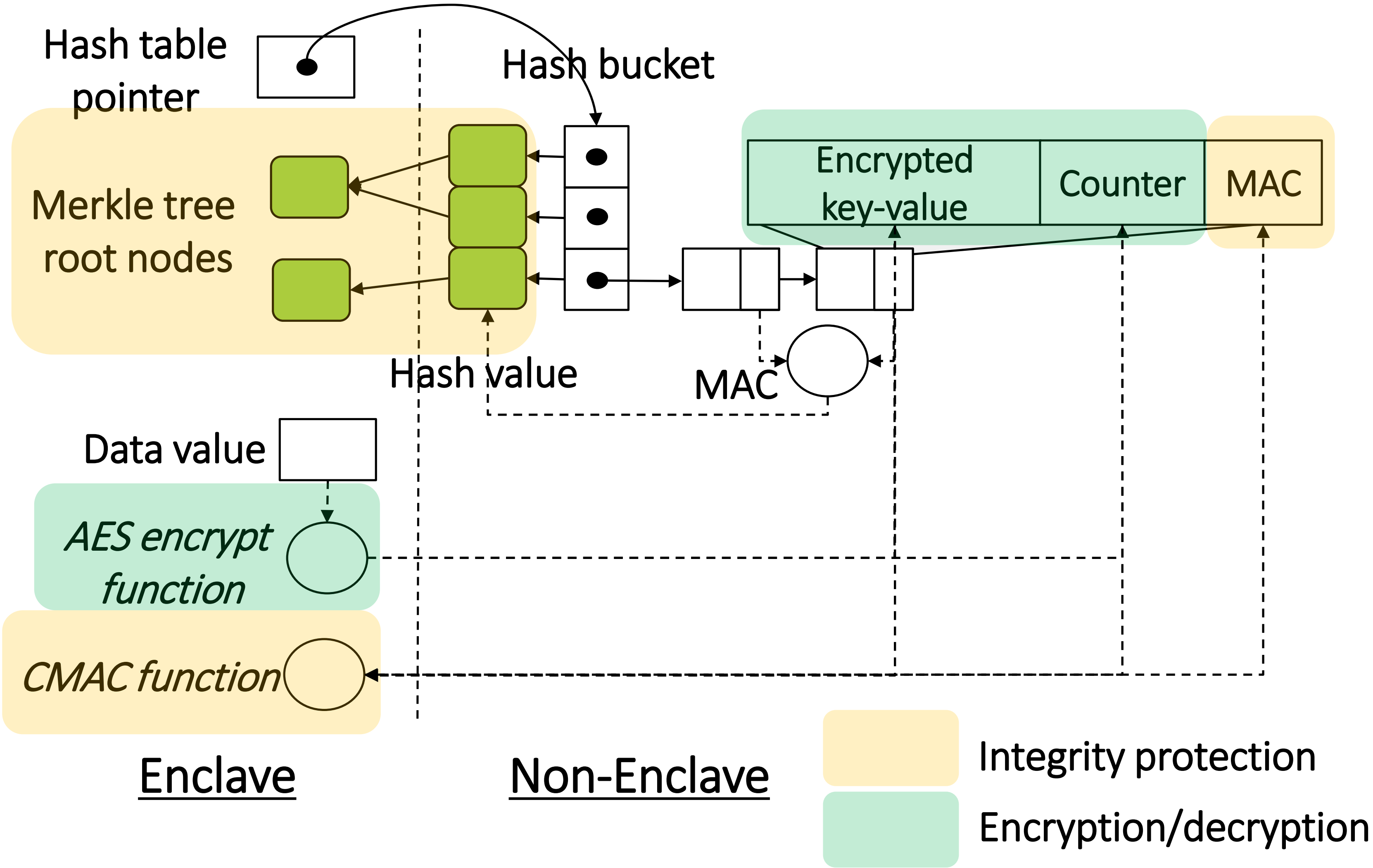
🟨 Integrity protection

🟩 Encryption/decryption

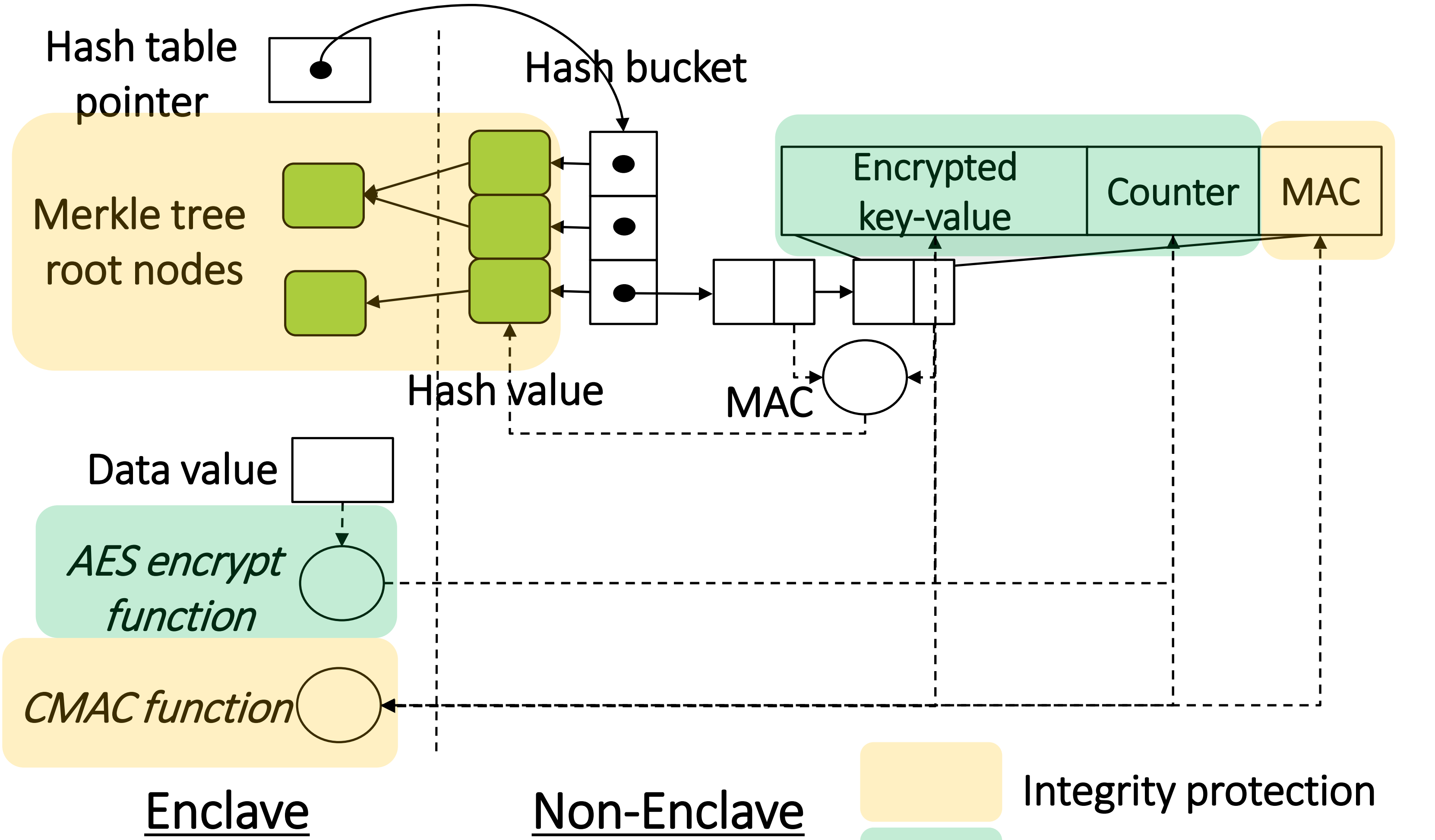
How To Protect Data?



How To Protect Data?



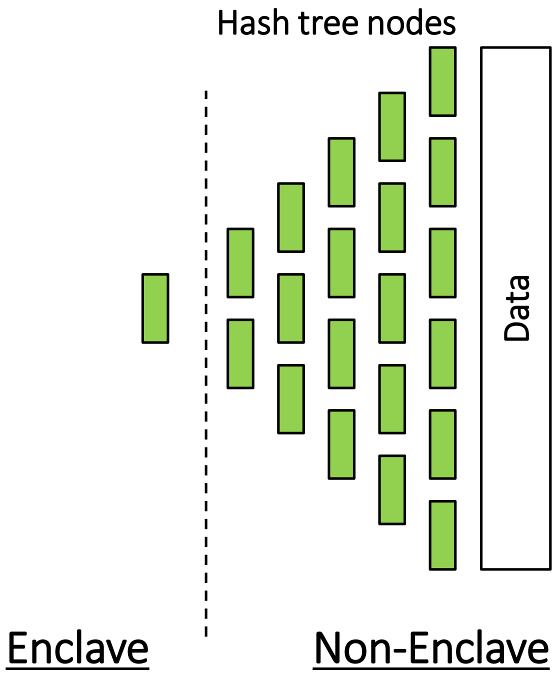
How To Protect Data?



Integrity Protection

- ShieldStore employs *Merkle Tree* mechanism
  - – Exploits the hash-based index structure to verify integrity efficiently

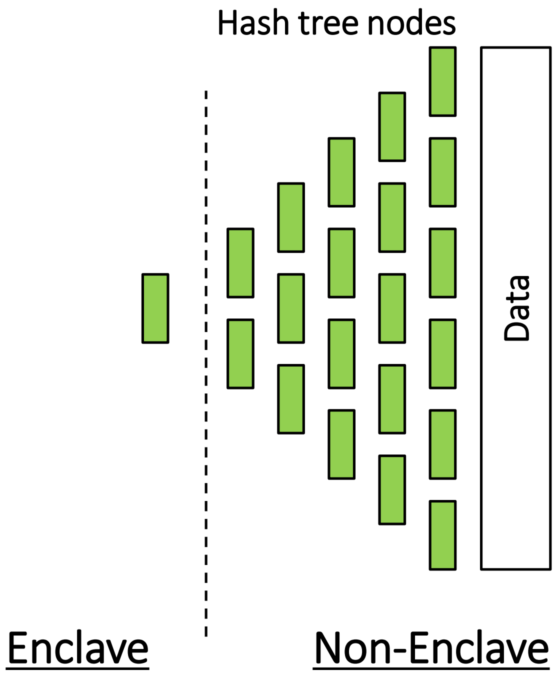
Traditional Merkle Tree



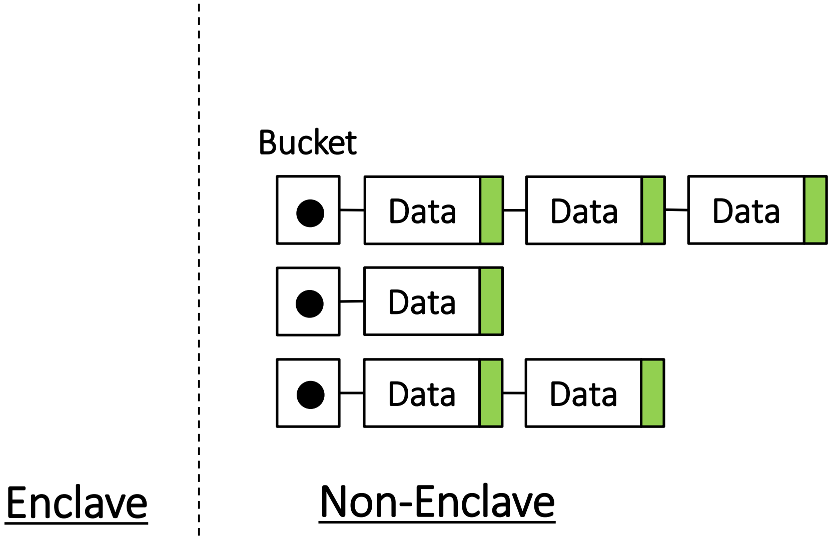
Integrity Protection

- ShieldStore employs *Merkle Tree* mechanism
  - Exploits the hash-based index structure to verify integrity efficiently

Traditional Merkle Tree



Data structure aware Merkle Tree



+ Reduce the depth of tree

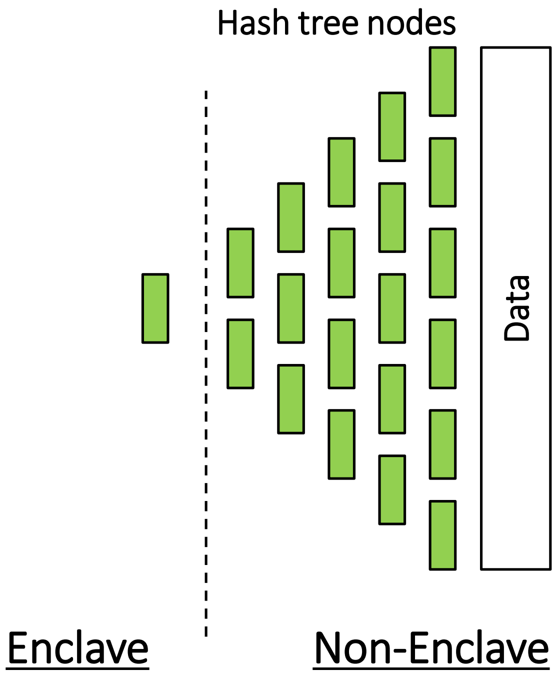
+ Keep subtree root node on *enclave*

- Traverse all the MAC entries

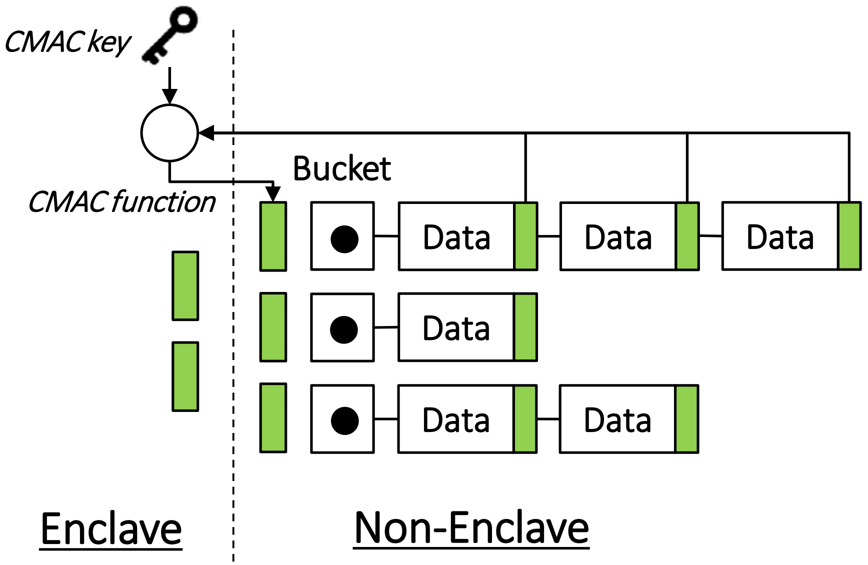
Integrity Protection

- ShieldStore employs *Merkle Tree* mechanism
  - Exploits the hash-based index structure to verify integrity efficiently

Traditional Merkle Tree



Data structure aware Merkle Tree



+ Reduce the depth of tree

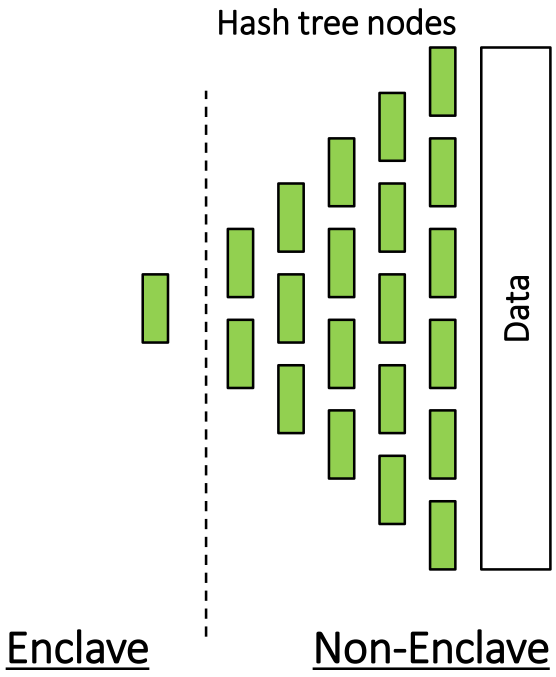
+ Keep subtree root node on *enclave*

- Traverse all the MAC entries

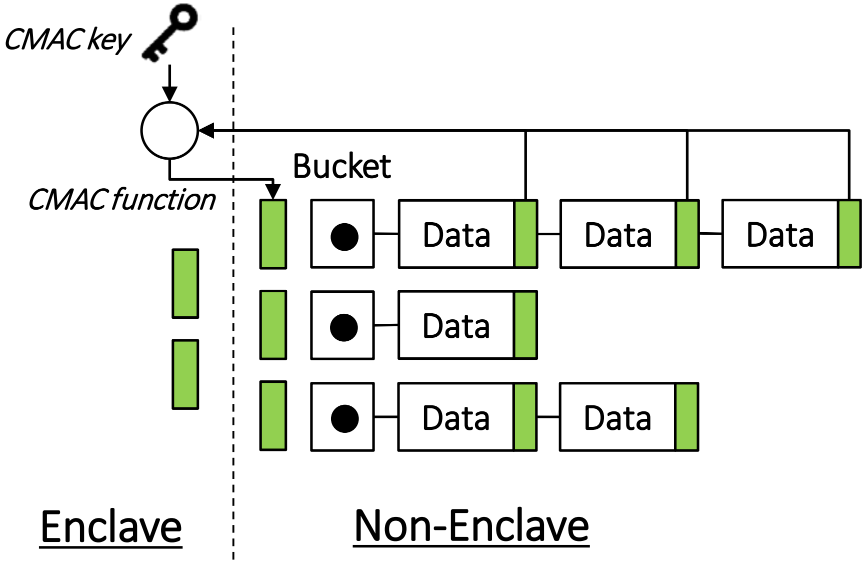
Integrity Protection

- ShieldStore employs *Merkle Tree* mechanism
  - Exploits the hash-based index structure to verify integrity efficiently

Traditional Merkle Tree



Data structure aware Merkle Tree

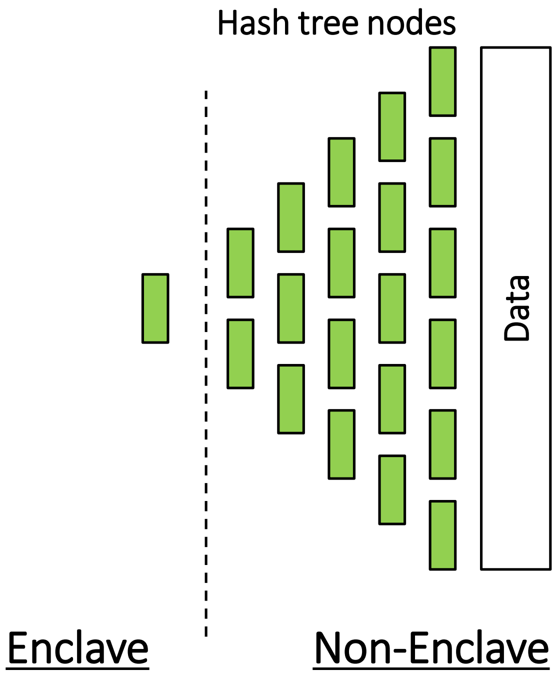


+ Reduce the depth of tree
+ Keep subtree root node on *enclave*
- Traverse all the MAC entries

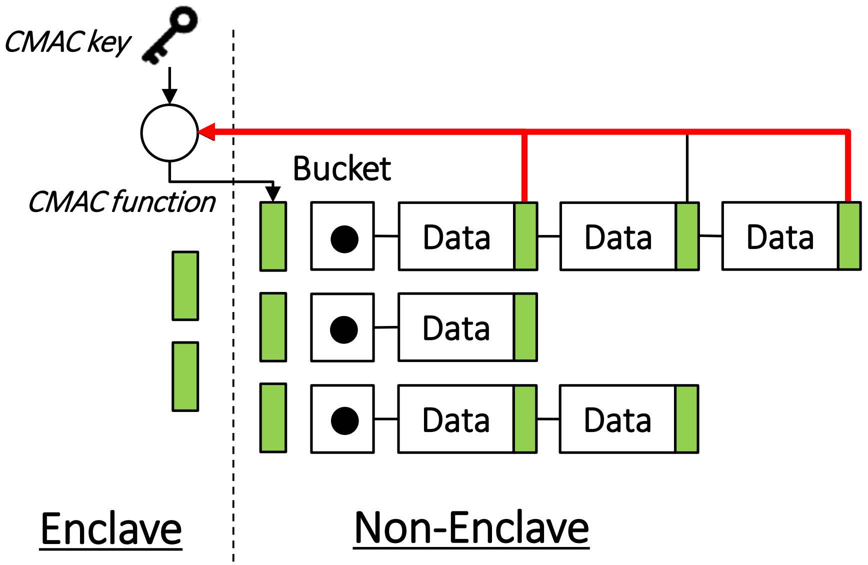
Integrity Protection

- ShieldStore employs *Merkle Tree* mechanism
  - Exploits the hash-based index structure to verify integrity efficiently

Traditional Merkle Tree



Data structure aware Merkle Tree



+ Reduce the depth of tree

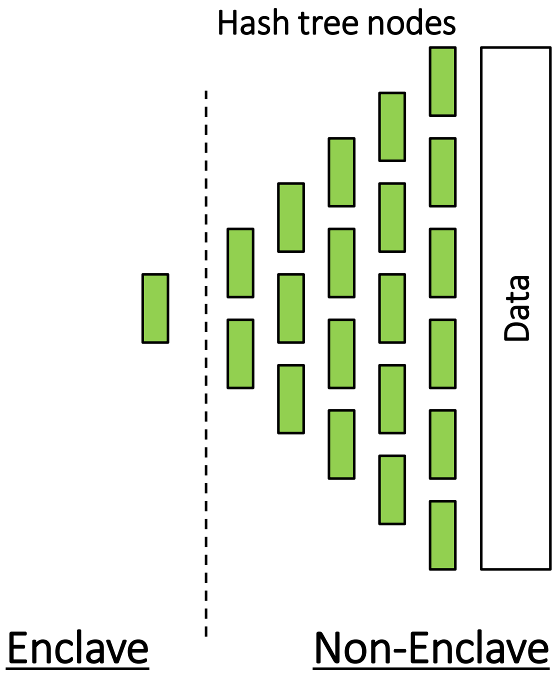
+ Keep subtree root node on *enclave*

- Traverse all the MAC entries

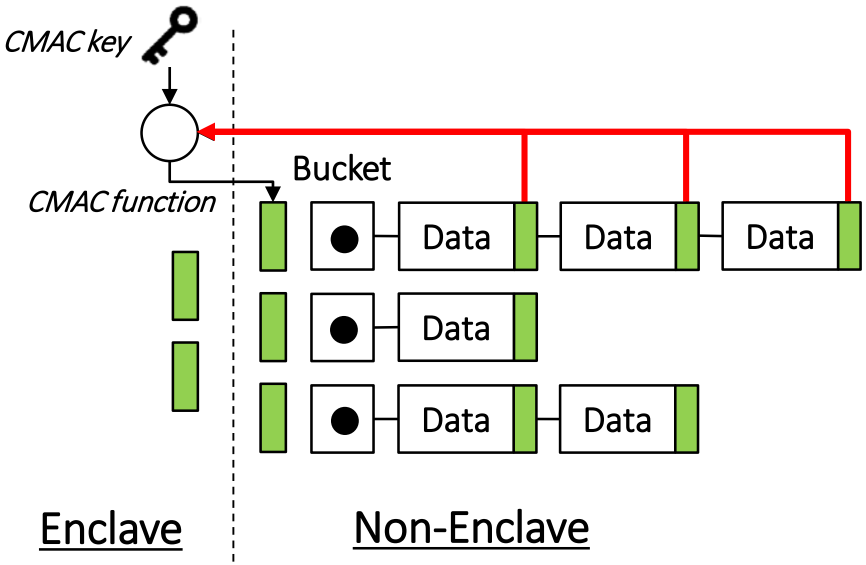
Integrity Protection

- ShieldStore employs *Merkle Tree* mechanism
  - Exploits the hash-based index structure to verify integrity efficiently

Traditional Merkle Tree



Data structure aware Merkle Tree



+ Reduce the depth of tree

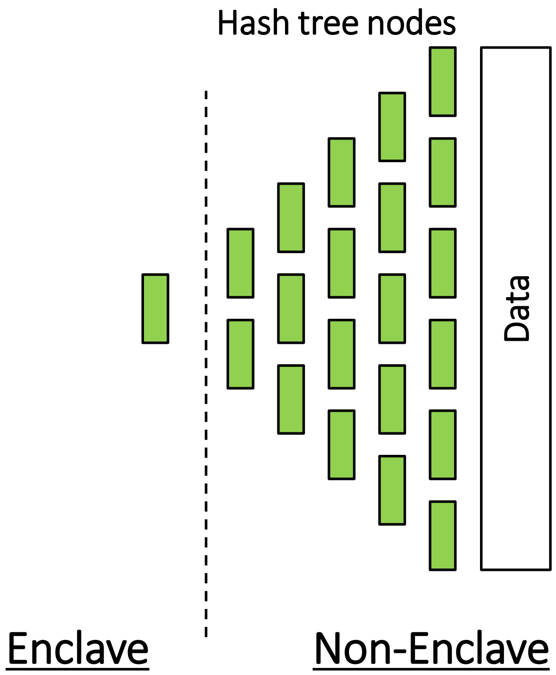
+ Keep subtree root node on *enclave*

- Traverse all the MAC entries

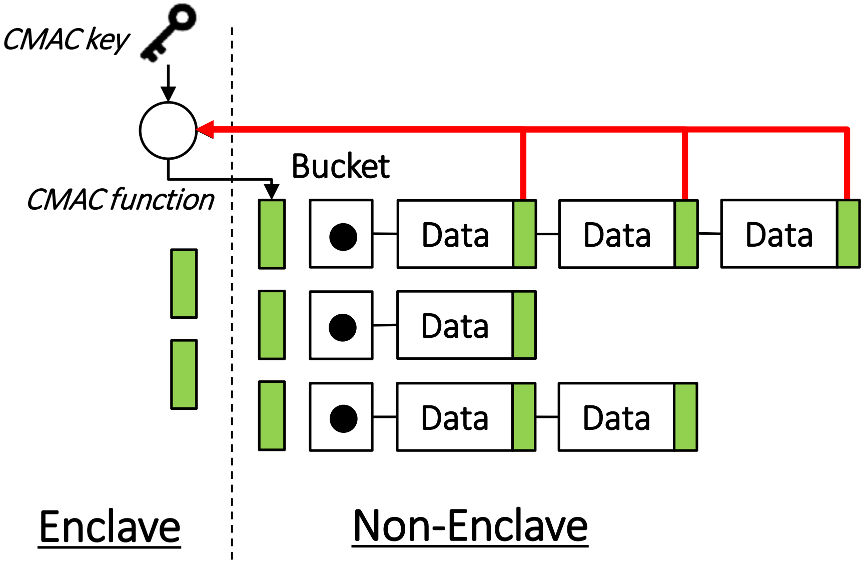
Integrity Protection

- ShieldStore employs *Merkle Tree* mechanism
  - Exploits the hash-based index structure to verify integrity efficiently

Traditional Merkle Tree



Data structure aware Merkle Tree

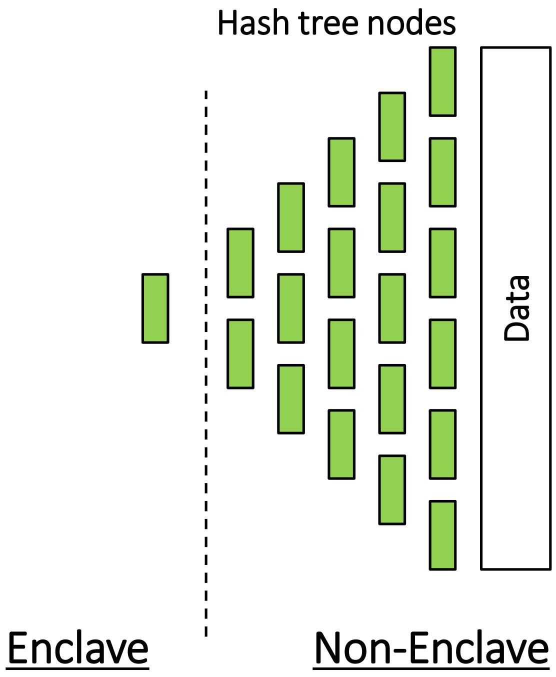


+ Reduce the depth of tree
+ Keep subtree root node on *enclave*
- Traverse all the MAC entries

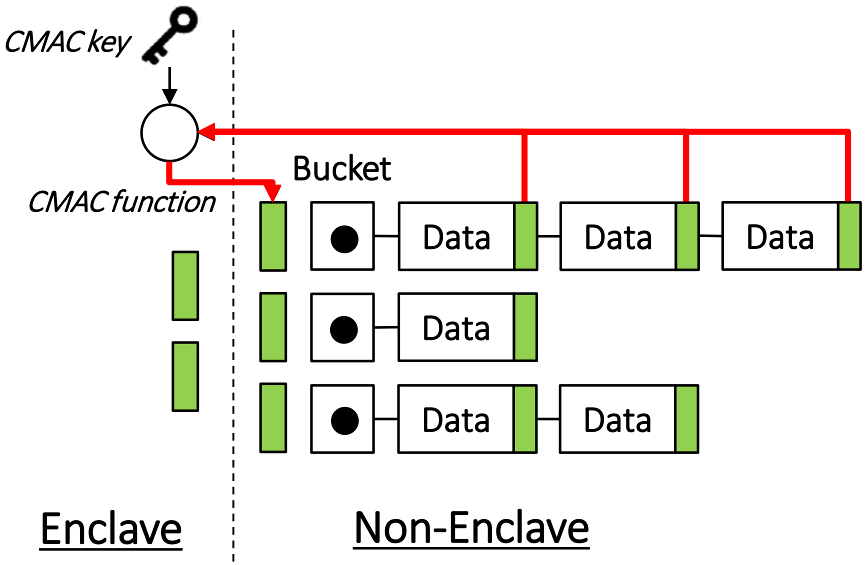
Integrity Protection

- ShieldStore employs *Merkle Tree* mechanism
  - Exploits the hash-based index structure to verify integrity efficiently

Traditional Merkle Tree



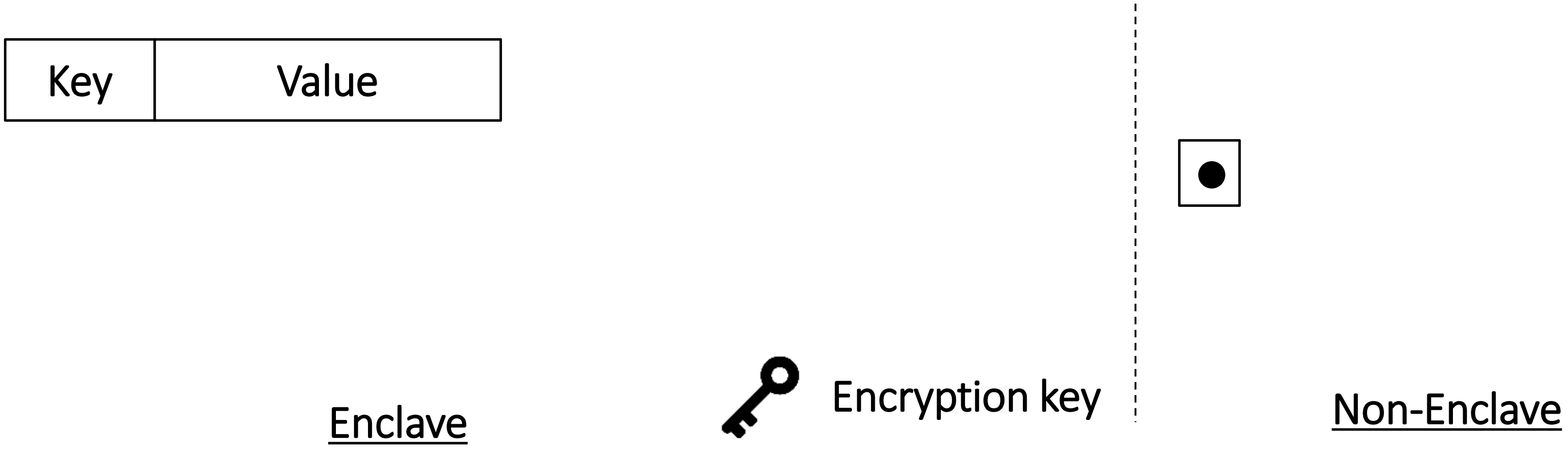
Data structure aware Merkle Tree



+ Reduce the depth of tree
+ Keep subtree root node on *enclave*
- Traverse all the MAC entries

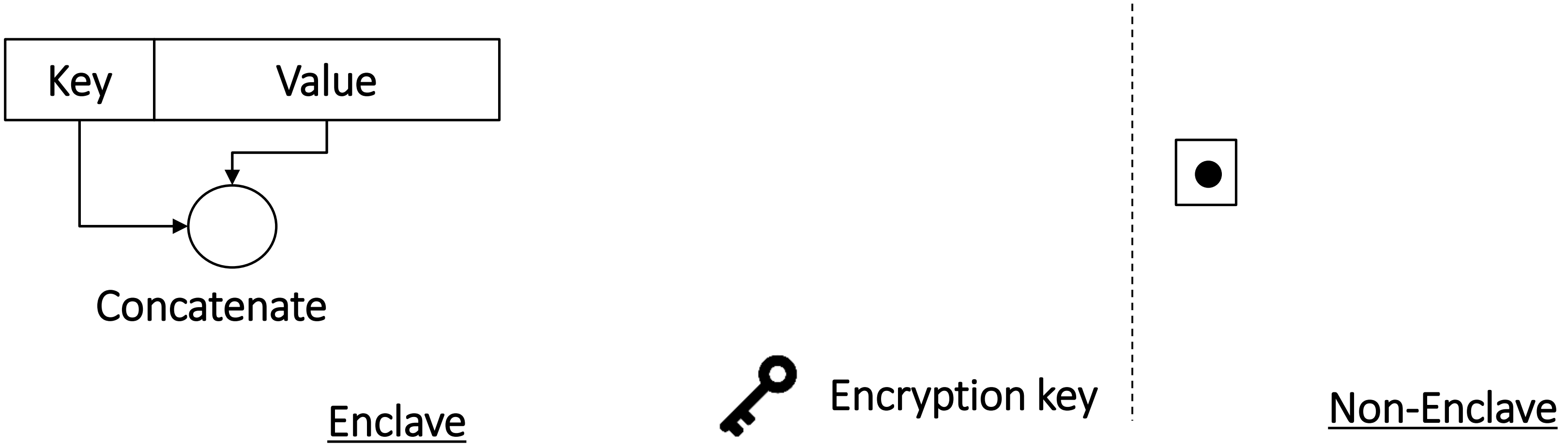
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information



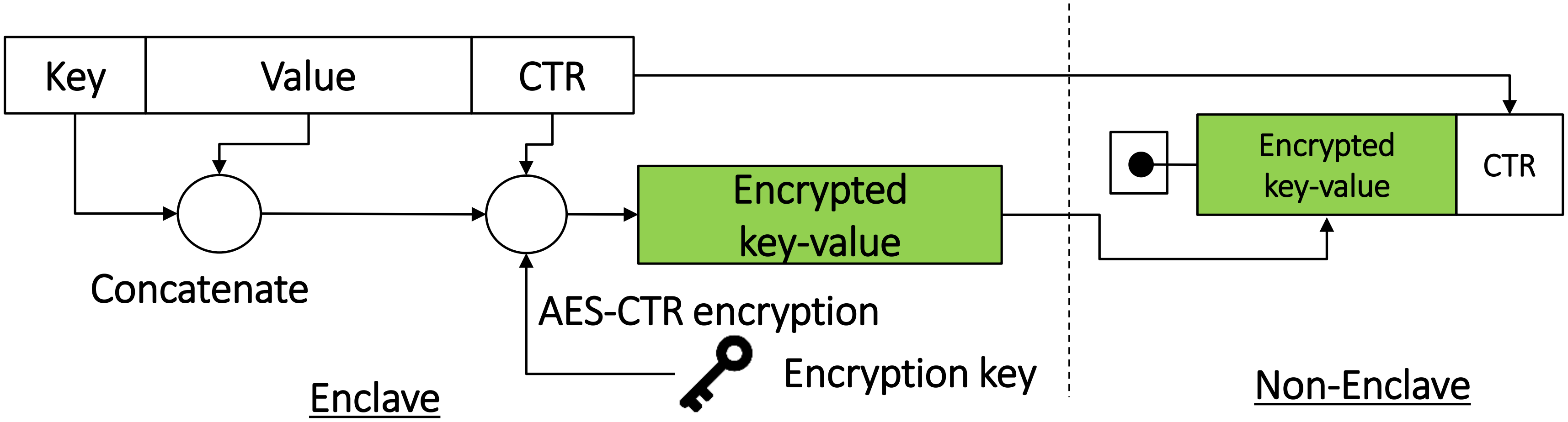
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information



Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

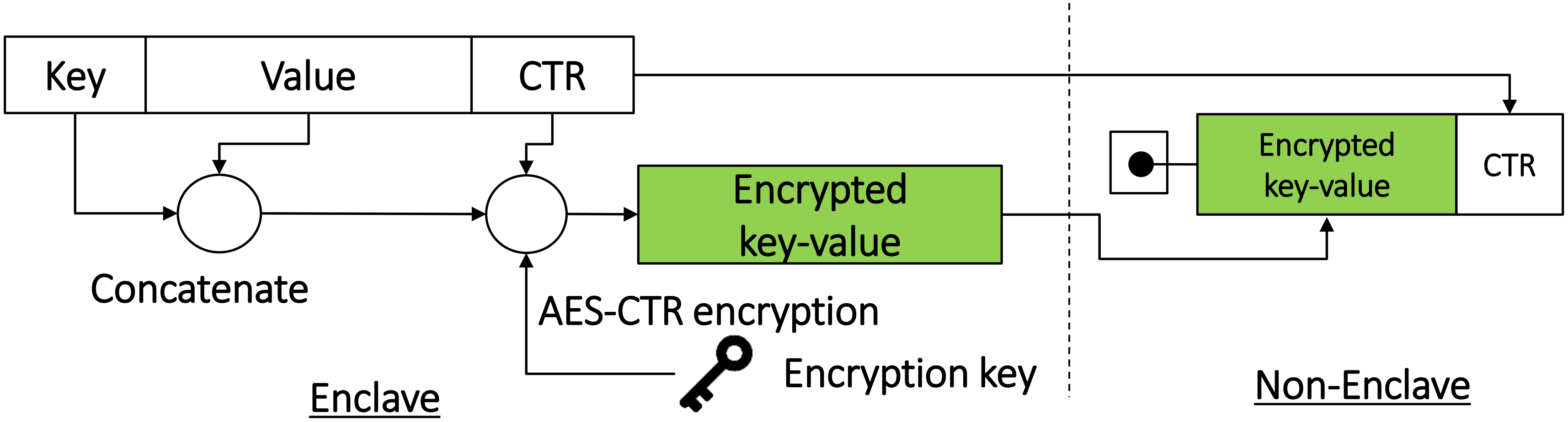


Encryption

- ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket

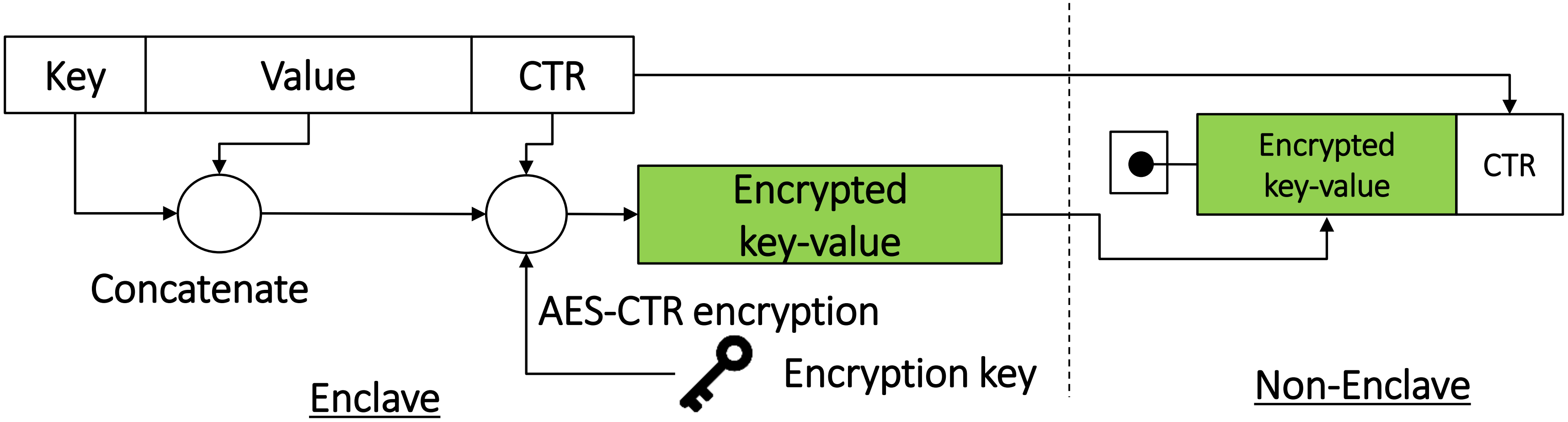


Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket

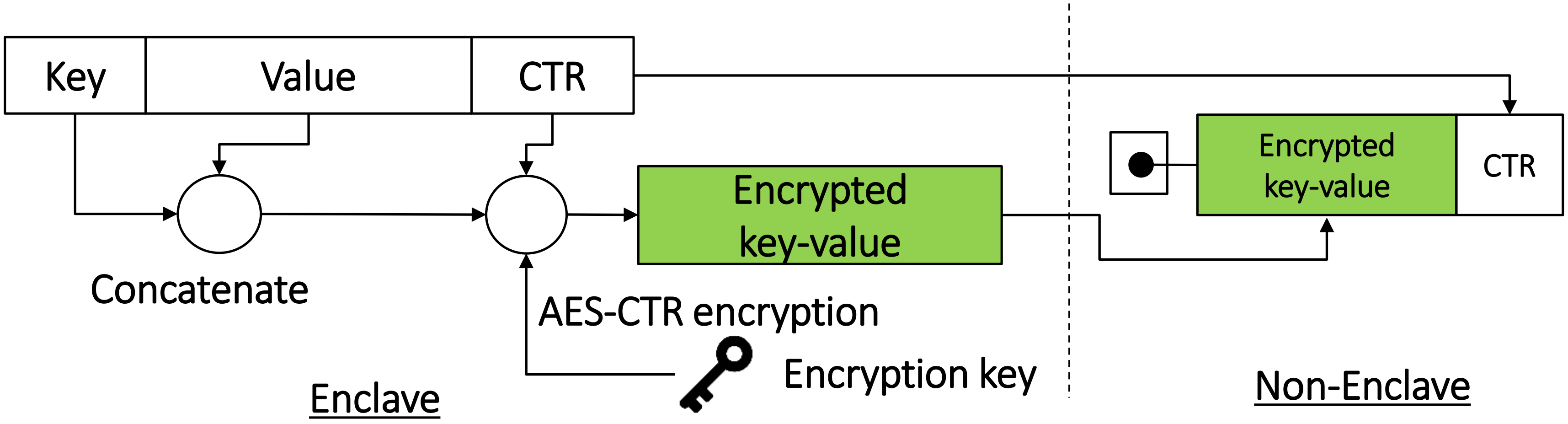


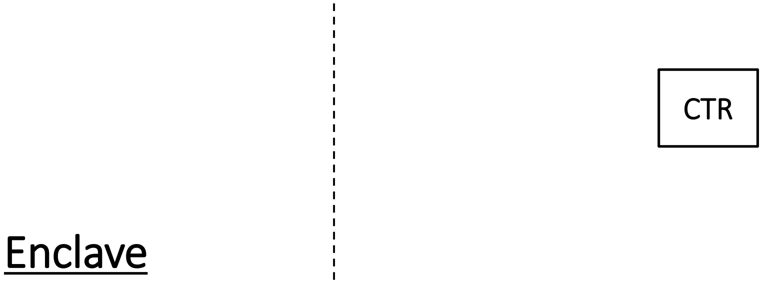
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



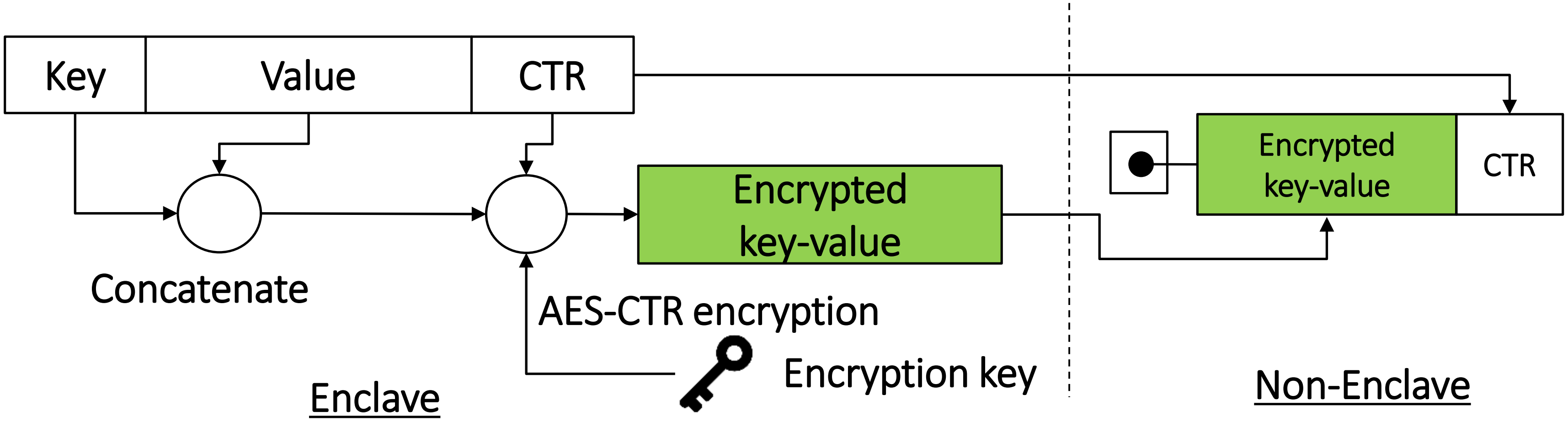


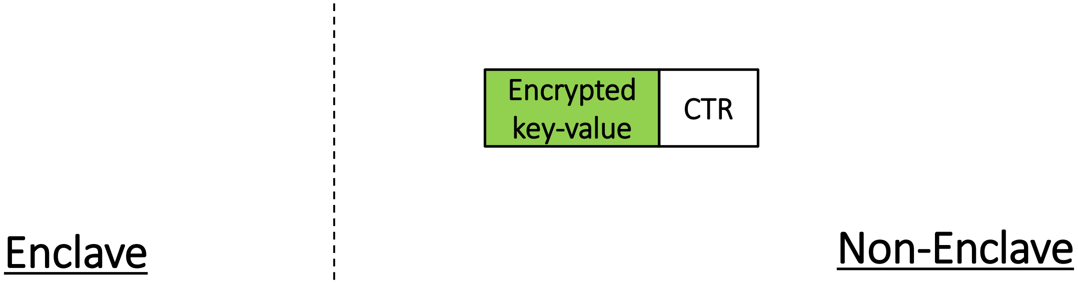
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



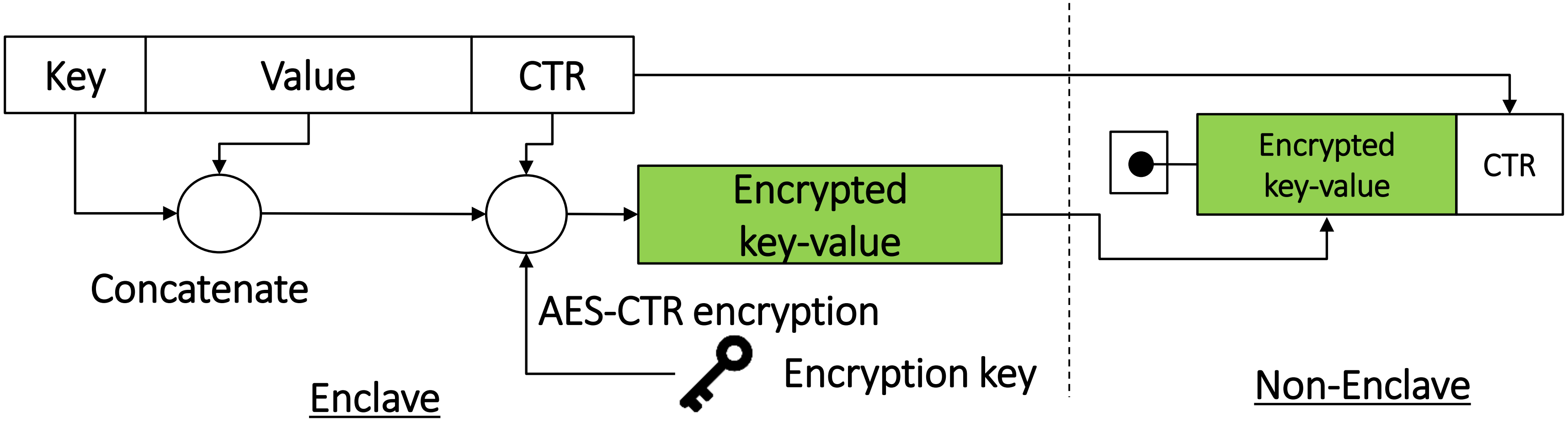


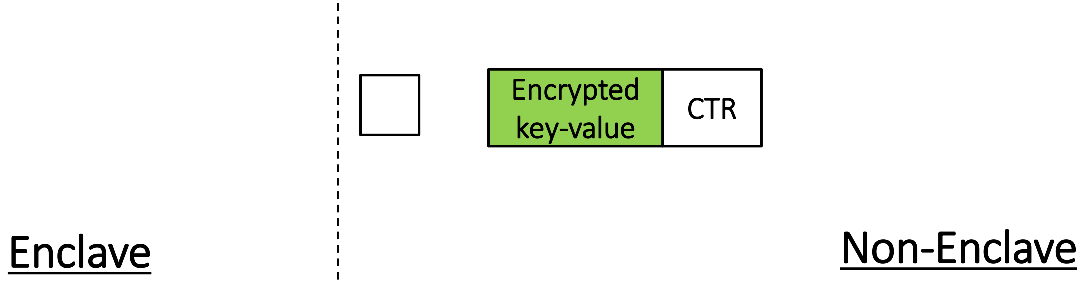
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



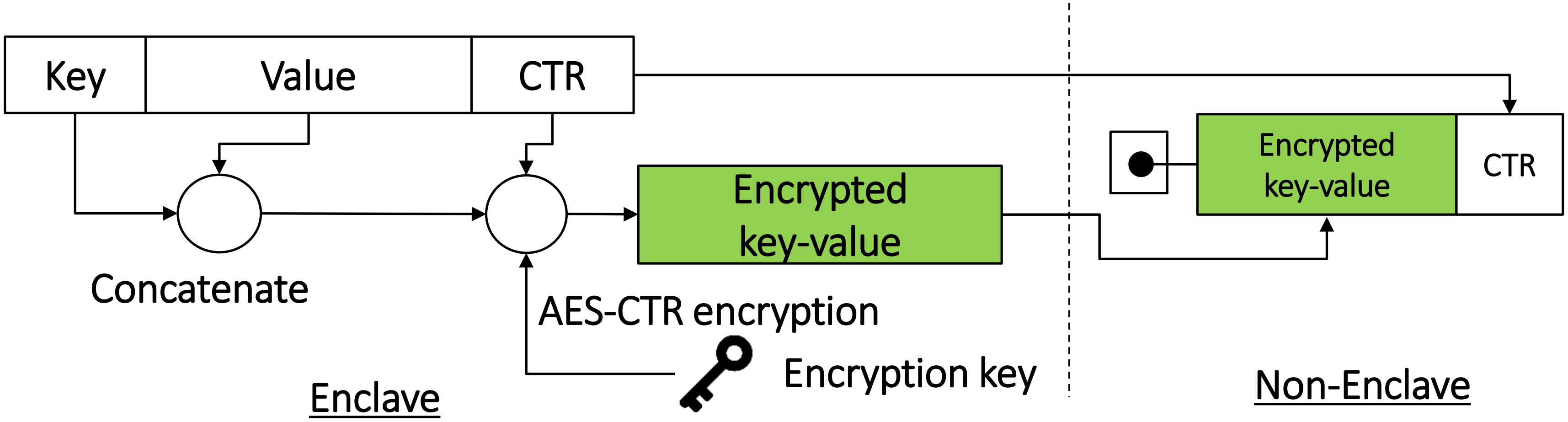


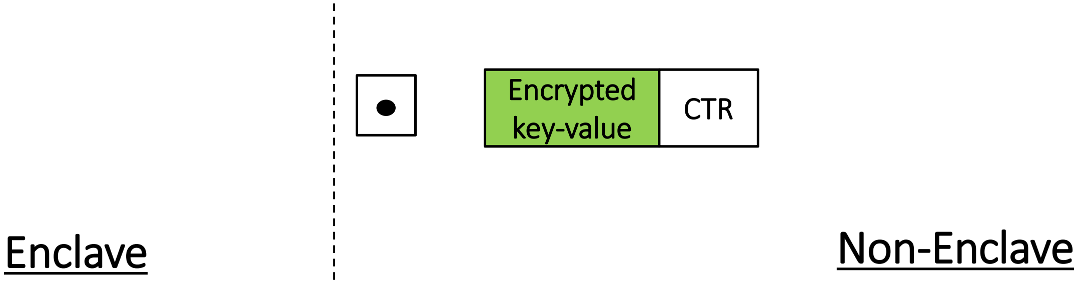
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



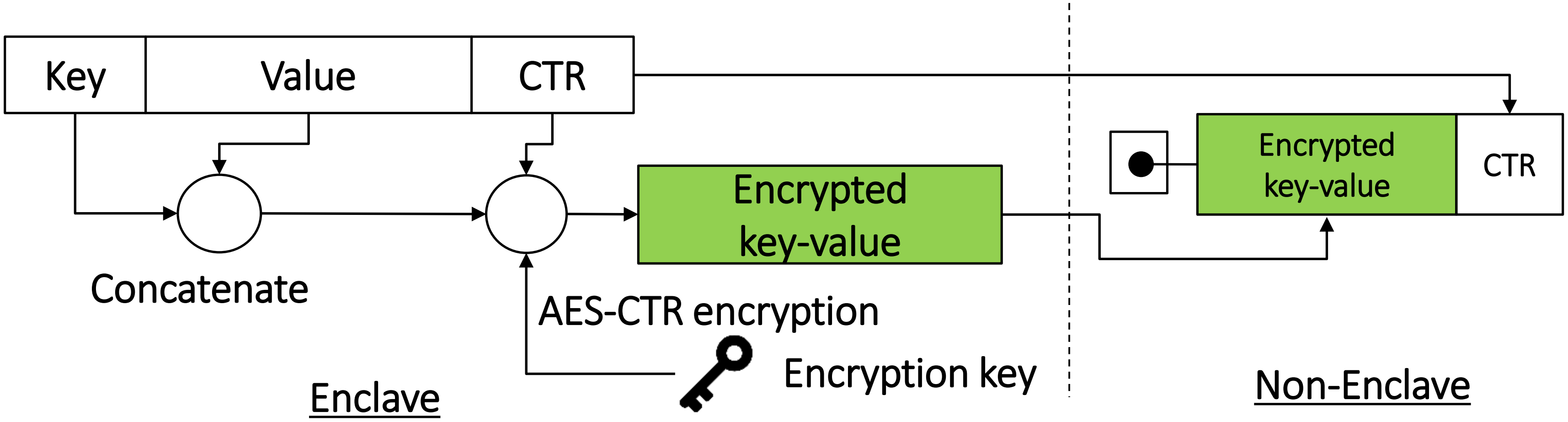


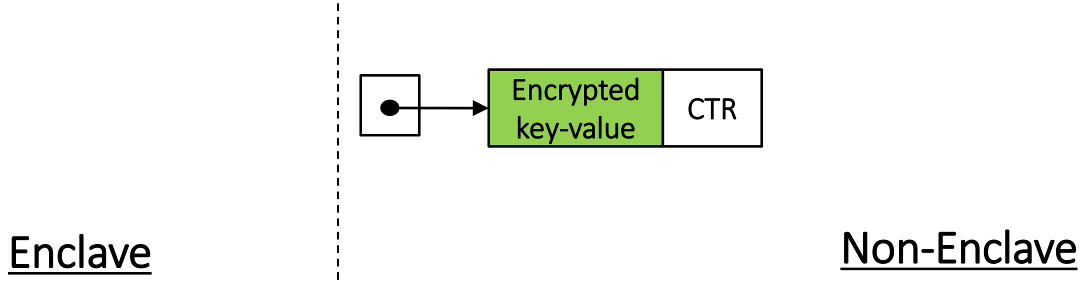
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



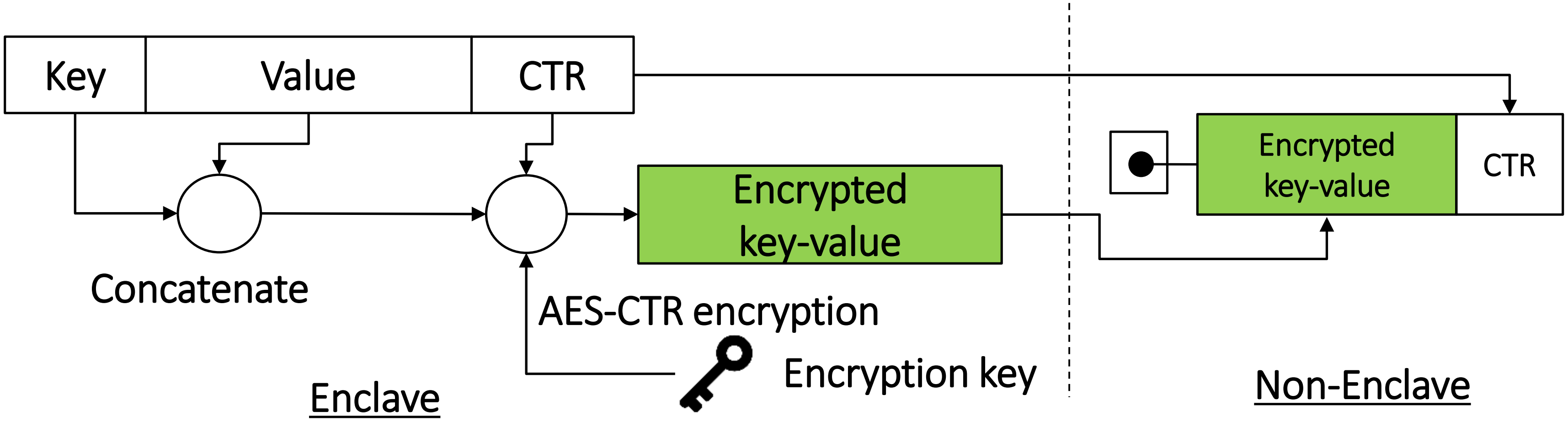


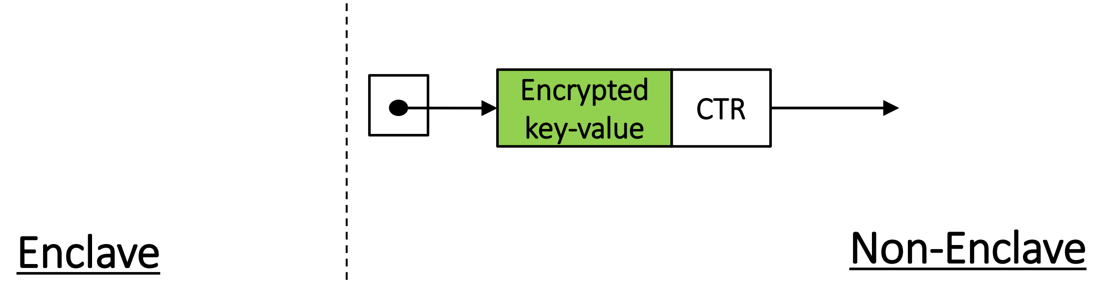
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



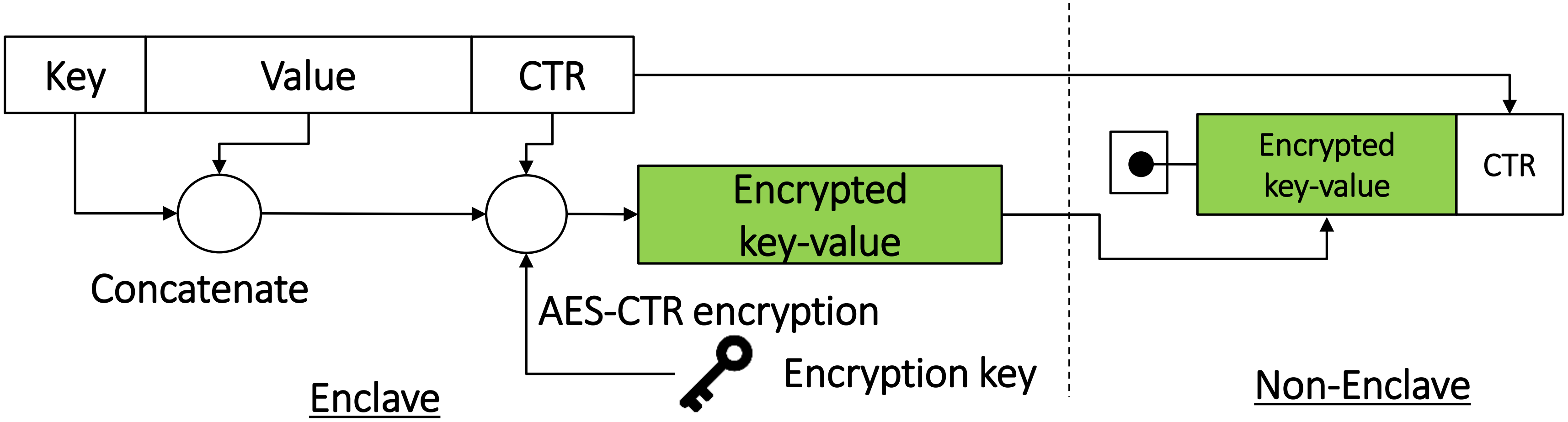


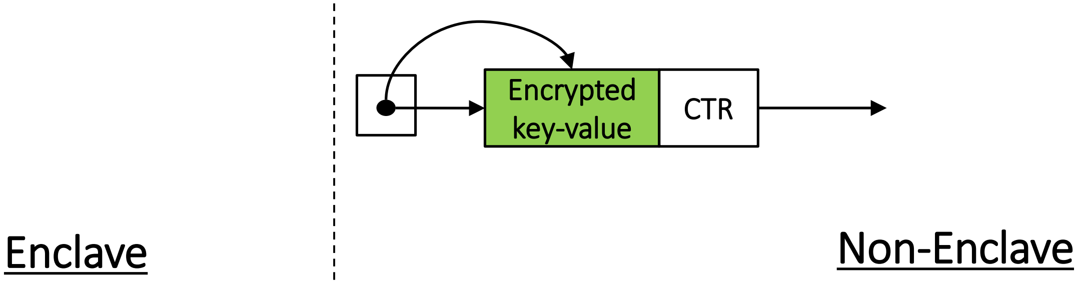
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



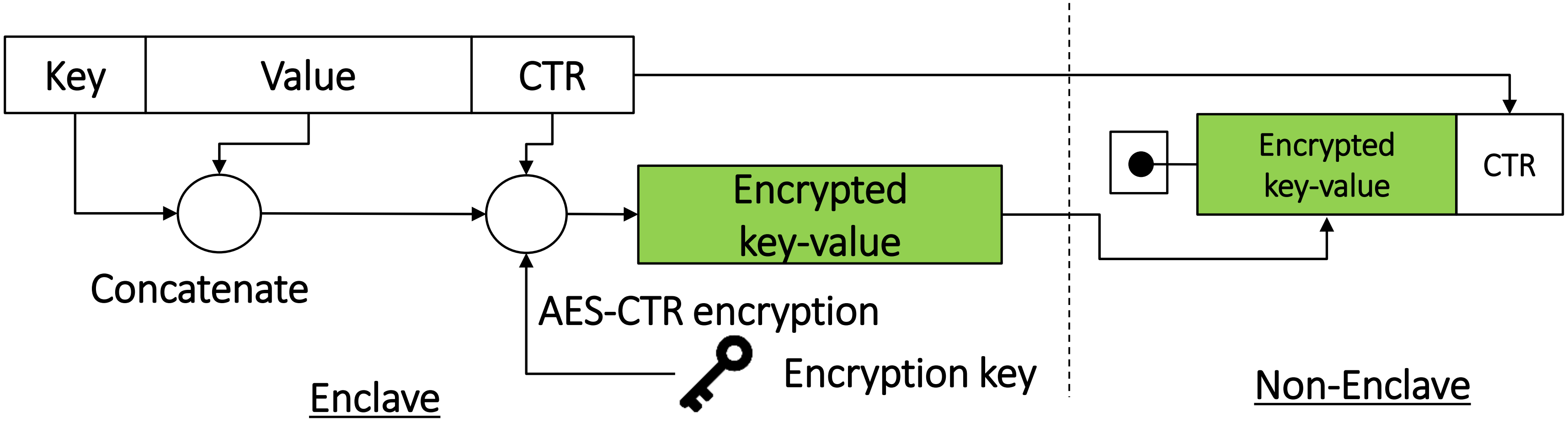


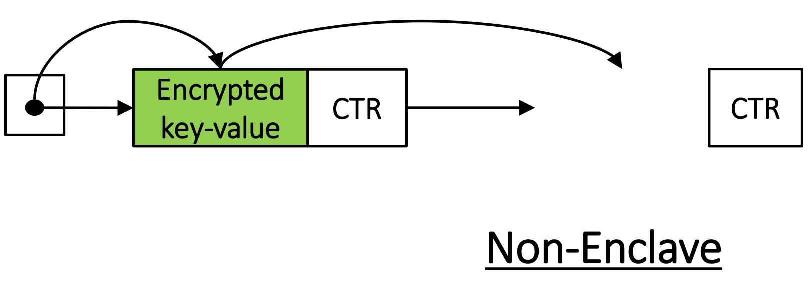
Encryption

- ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



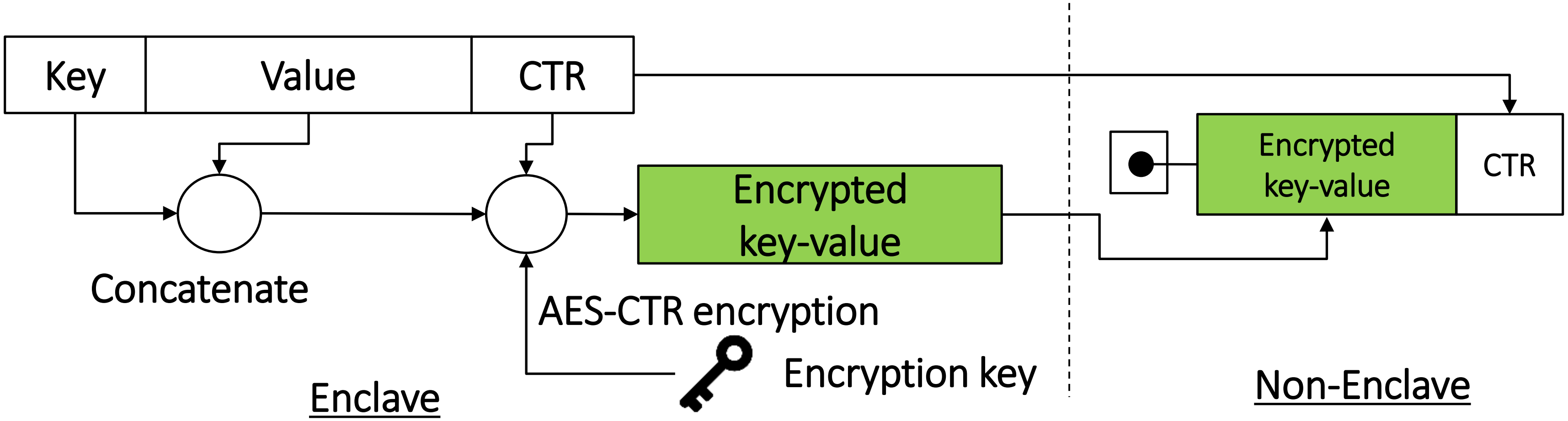


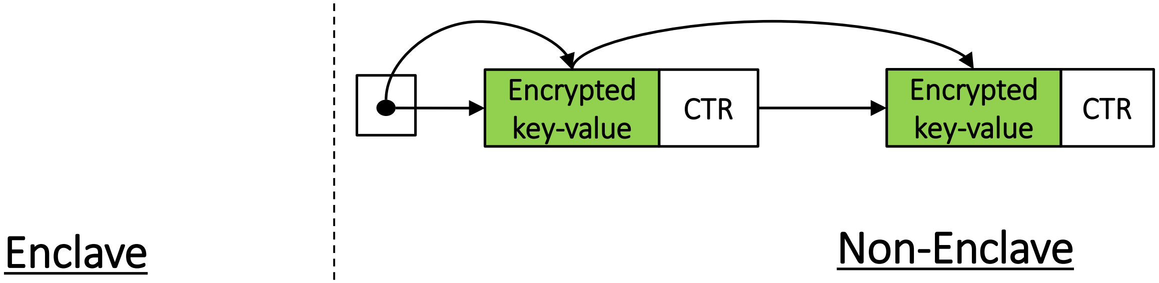
Encryption

- ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



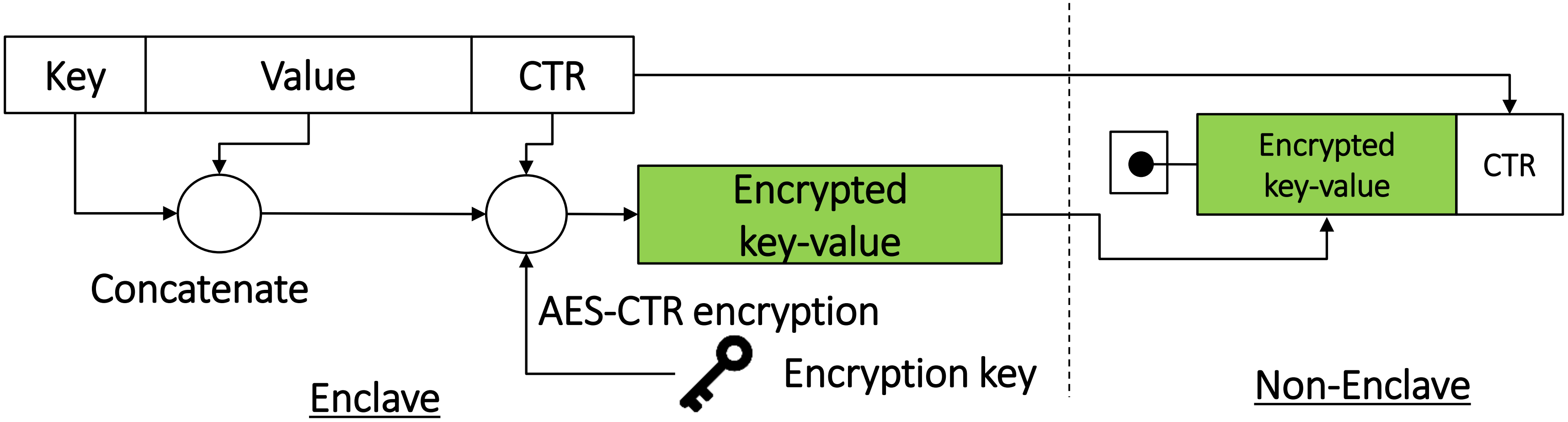


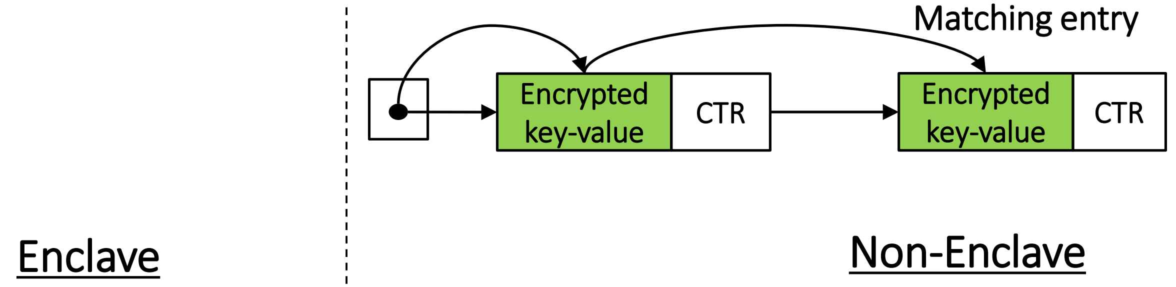
Encryption

- ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



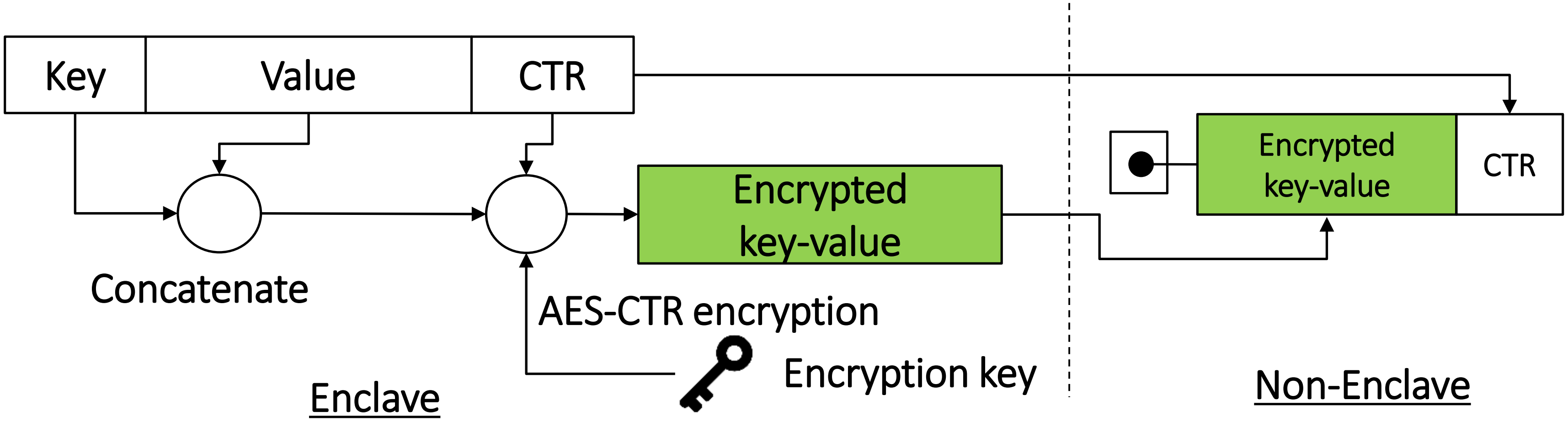


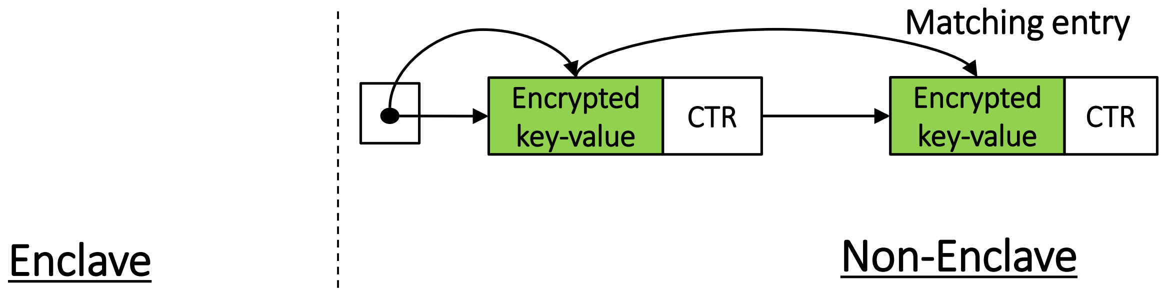
Encryption

- ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



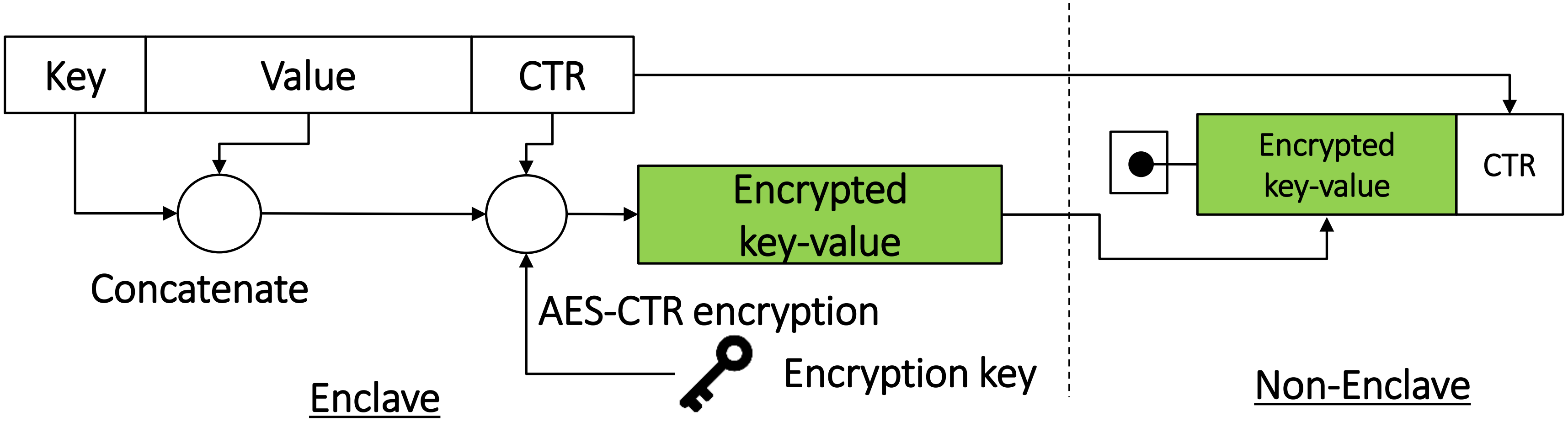


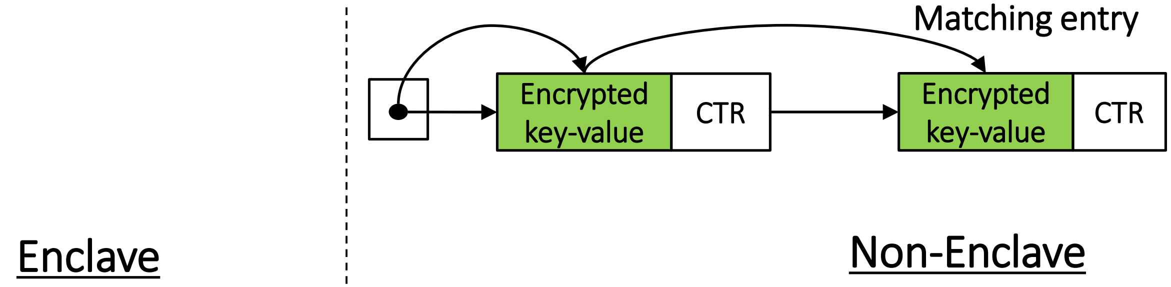
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



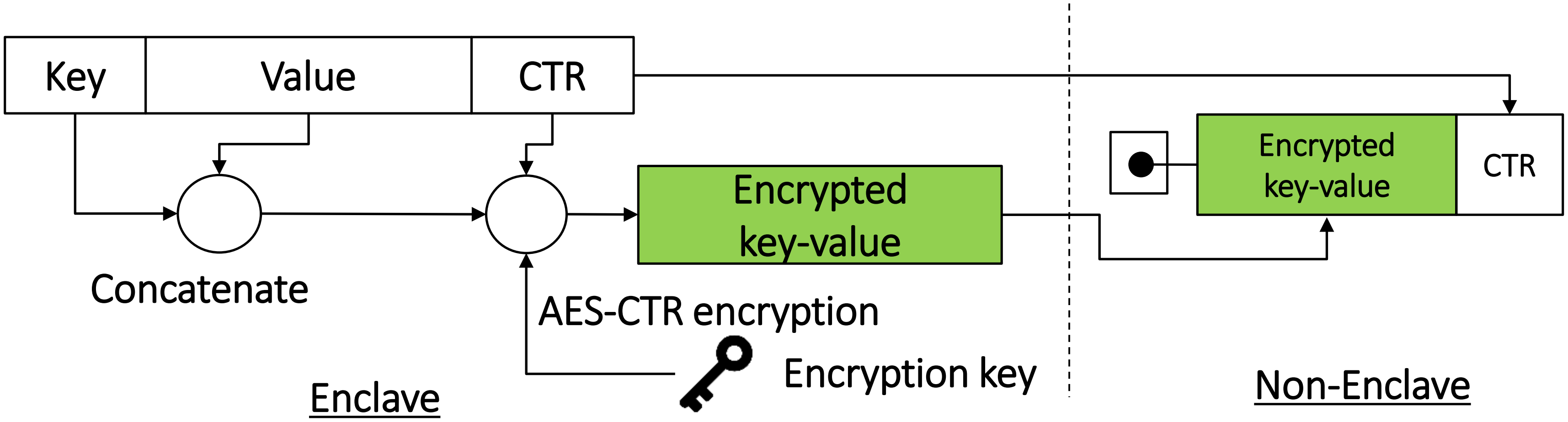


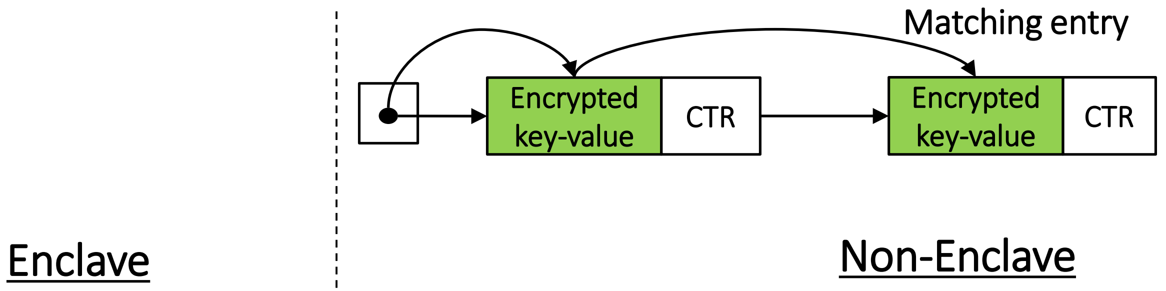
Encryption

- ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



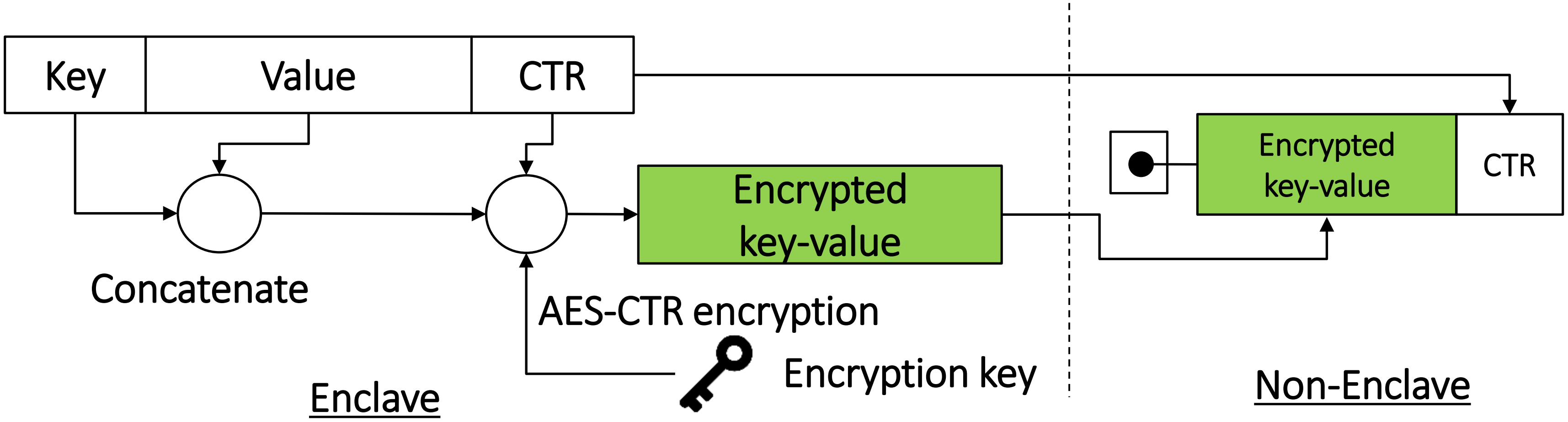


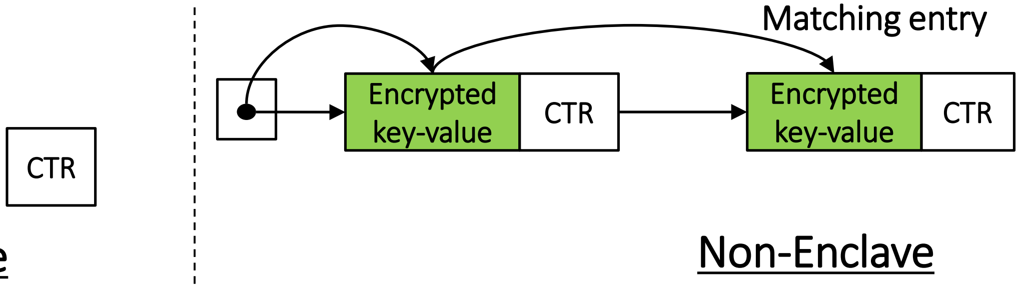
Encryption

- ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



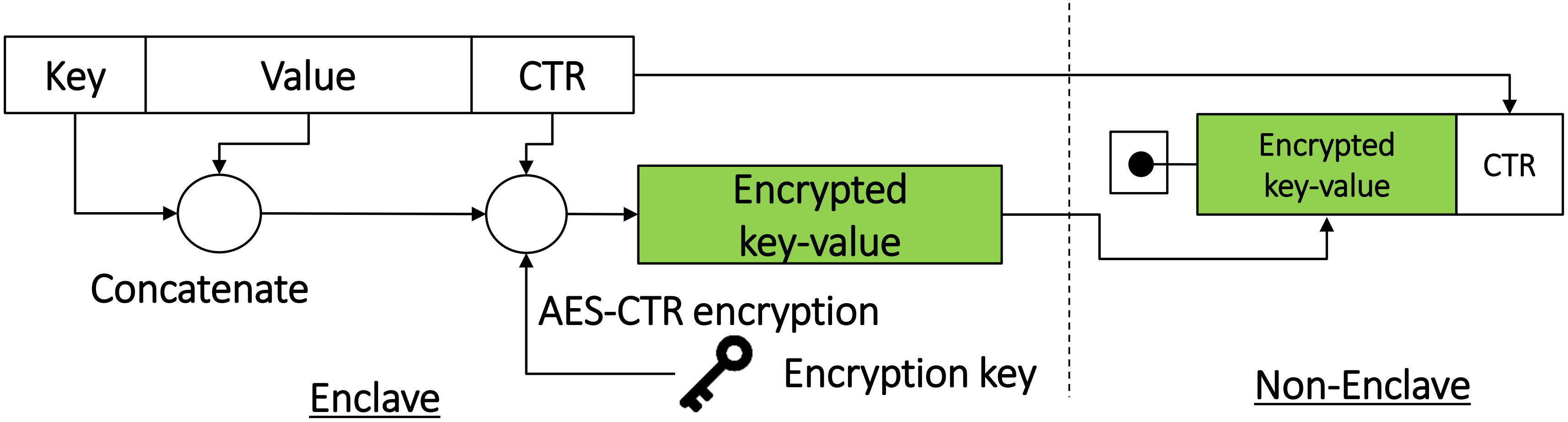


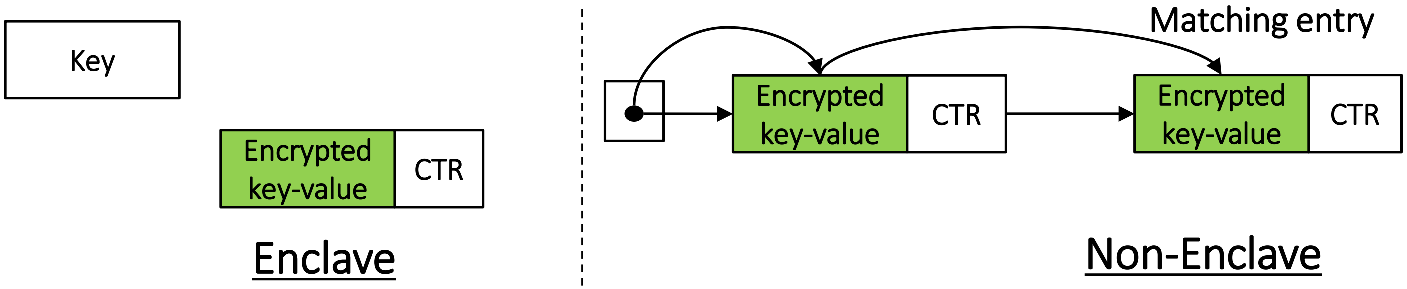
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



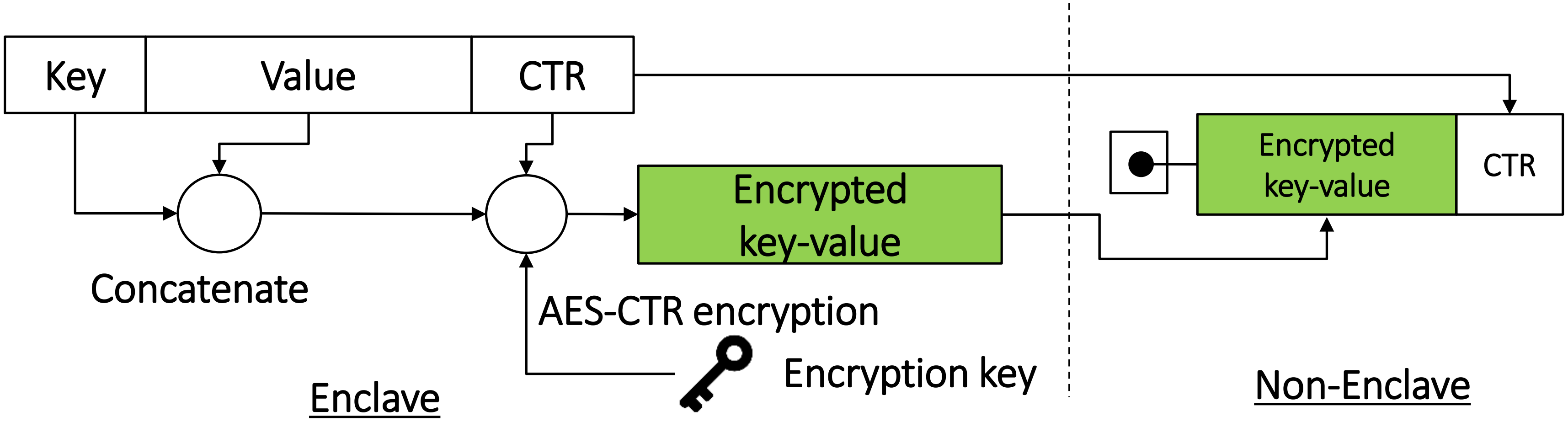


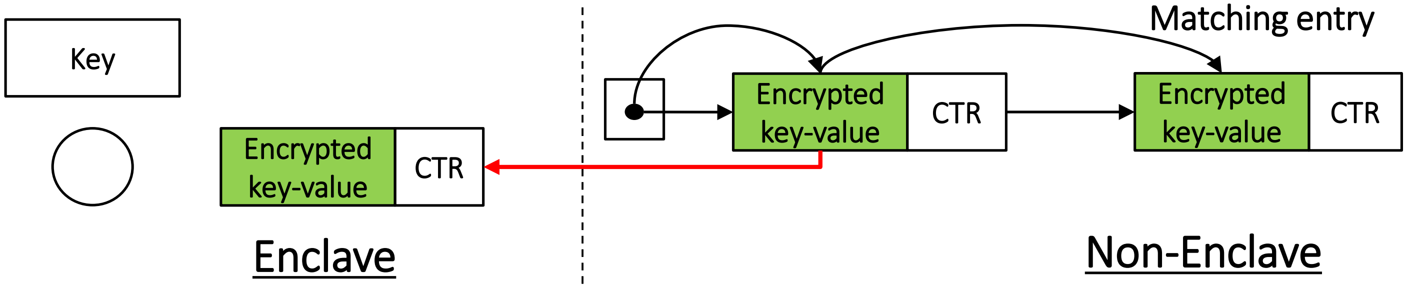
Encryption

- ShieldStore encrypts both key and value of the entry
  - – Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



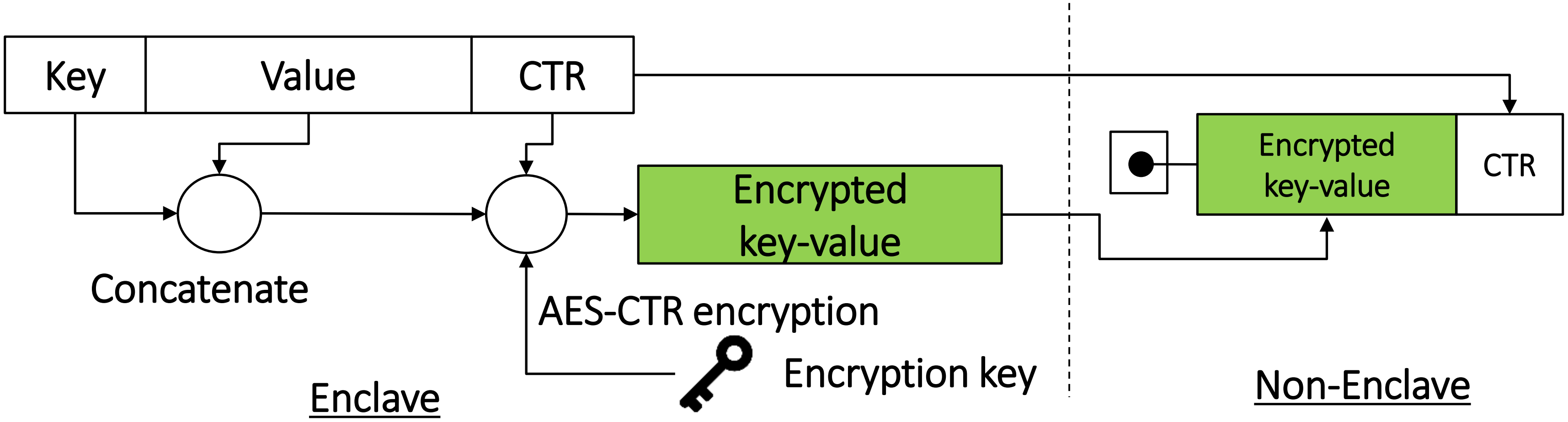


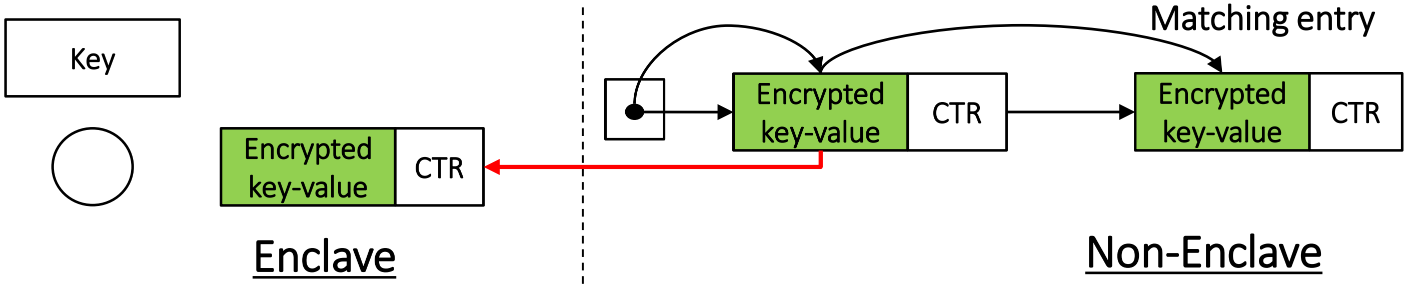
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



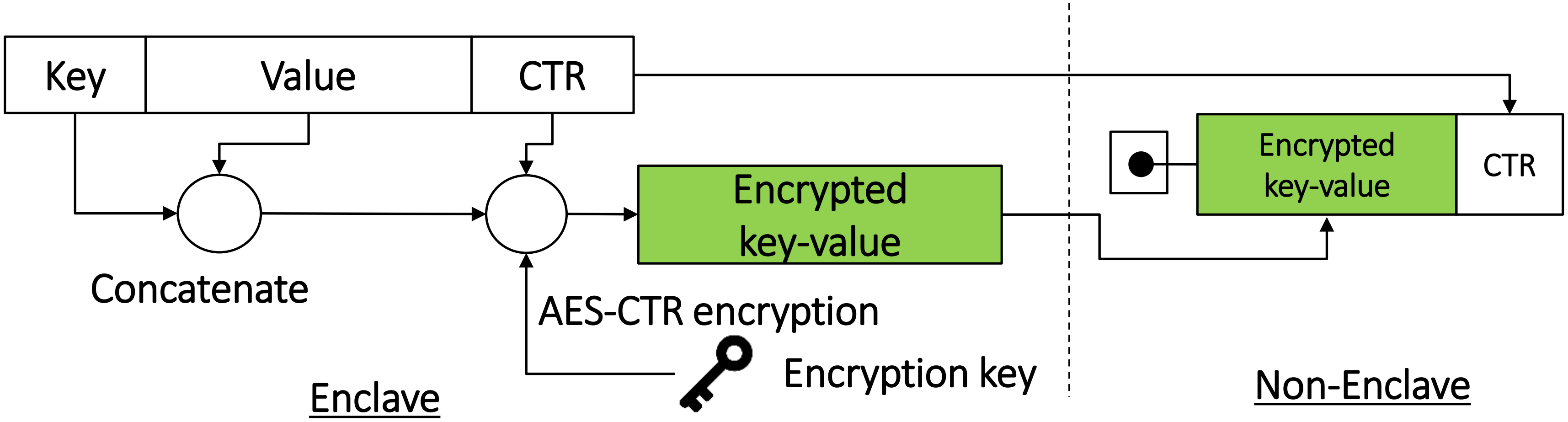


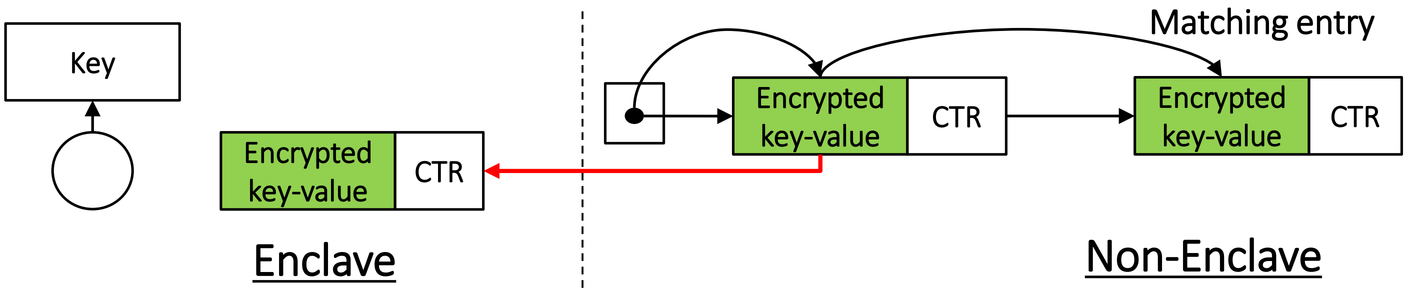
Encryption

- ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



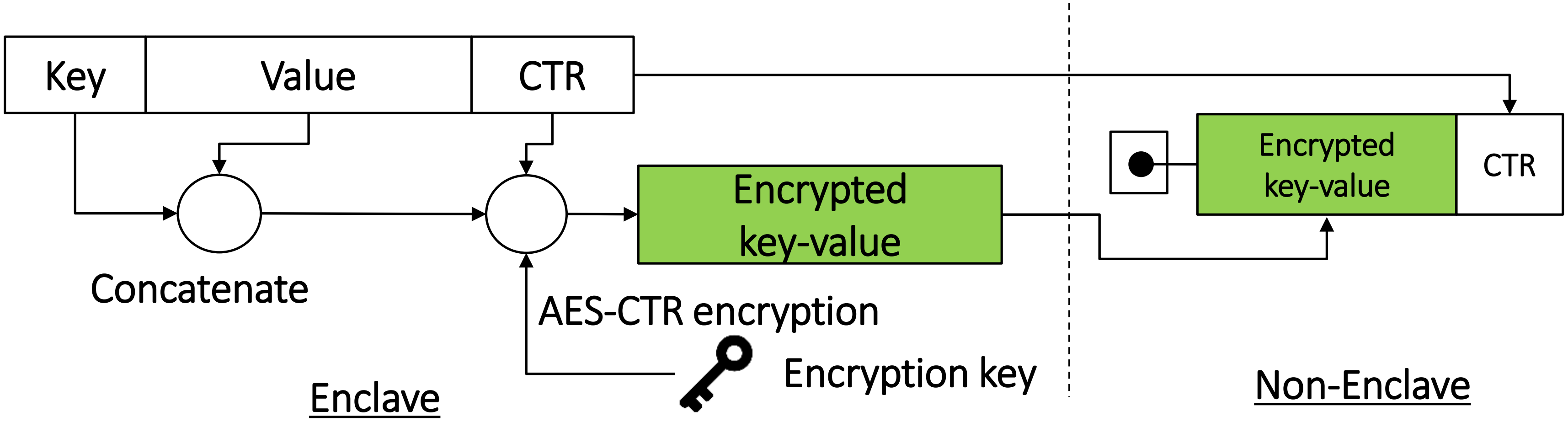


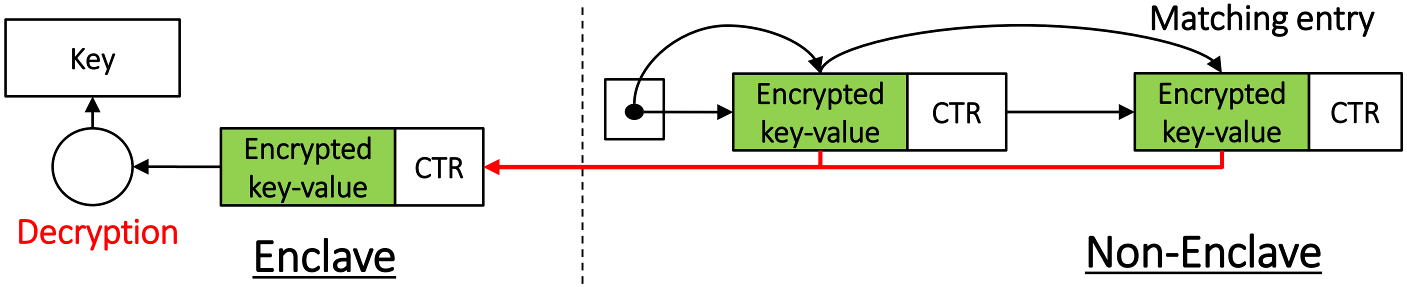
Encryption

- ShieldStore encrypts both key and value of the entry
  - – Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



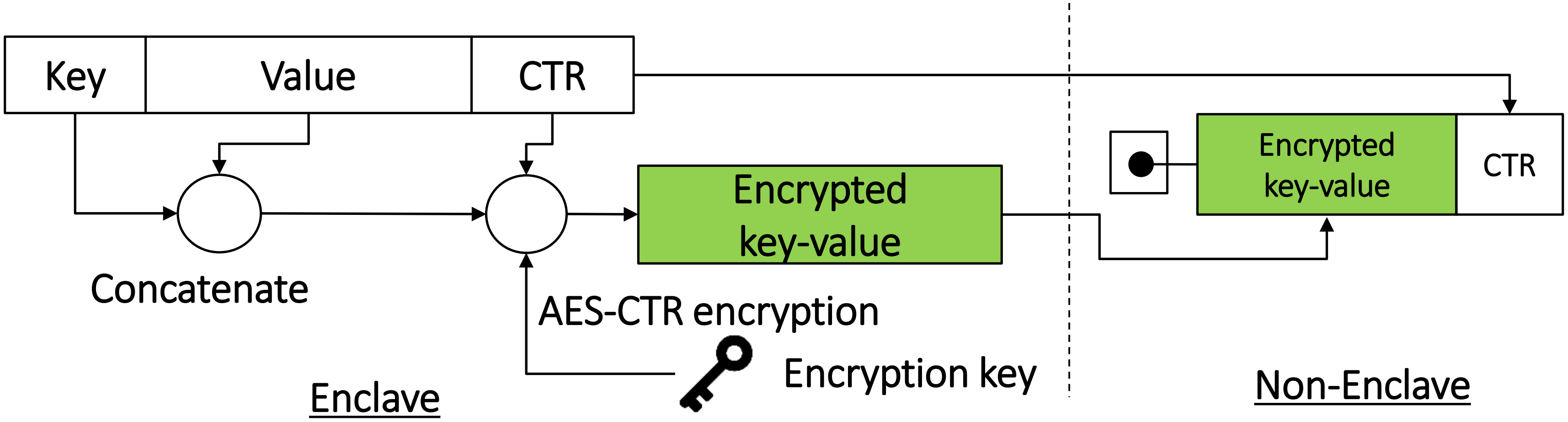


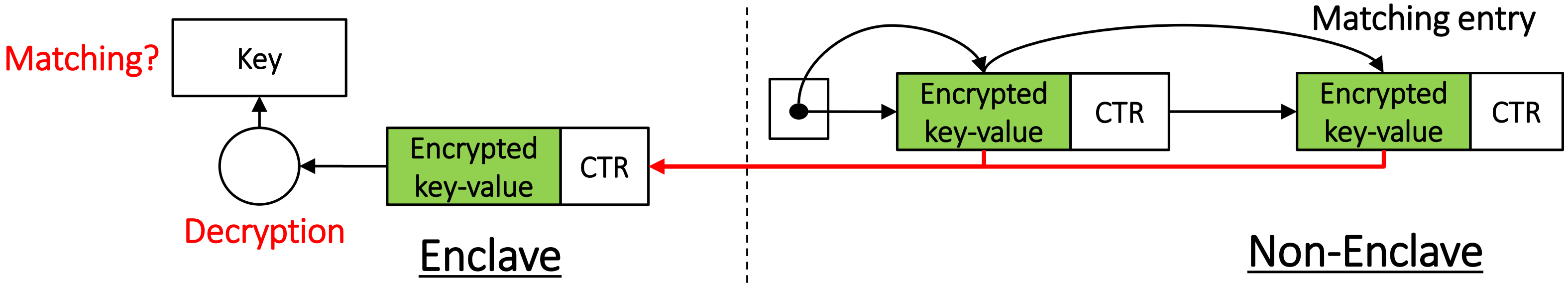
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



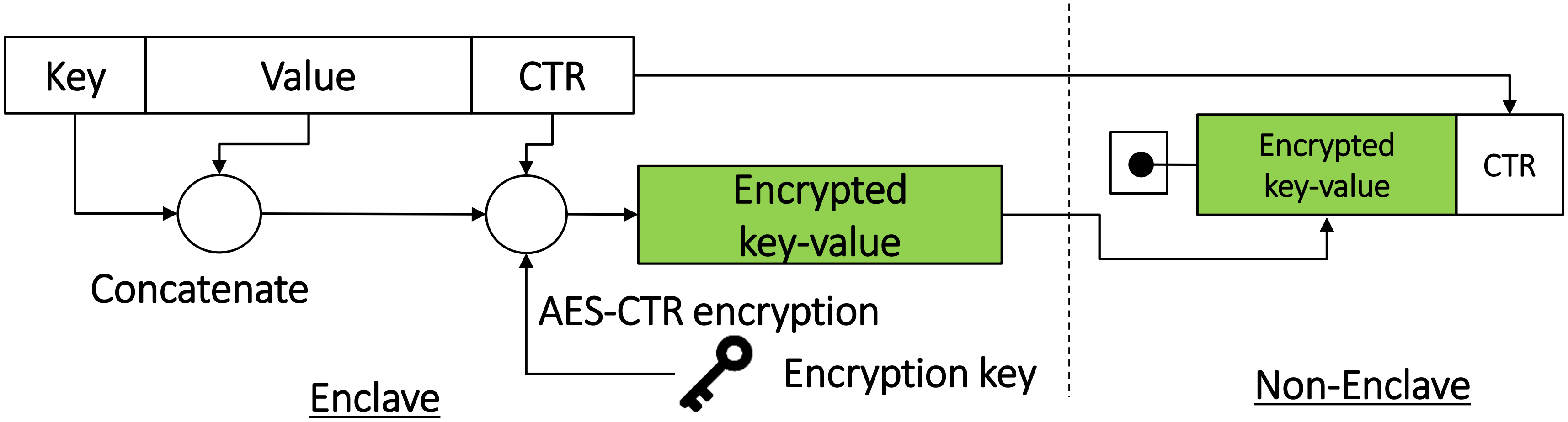


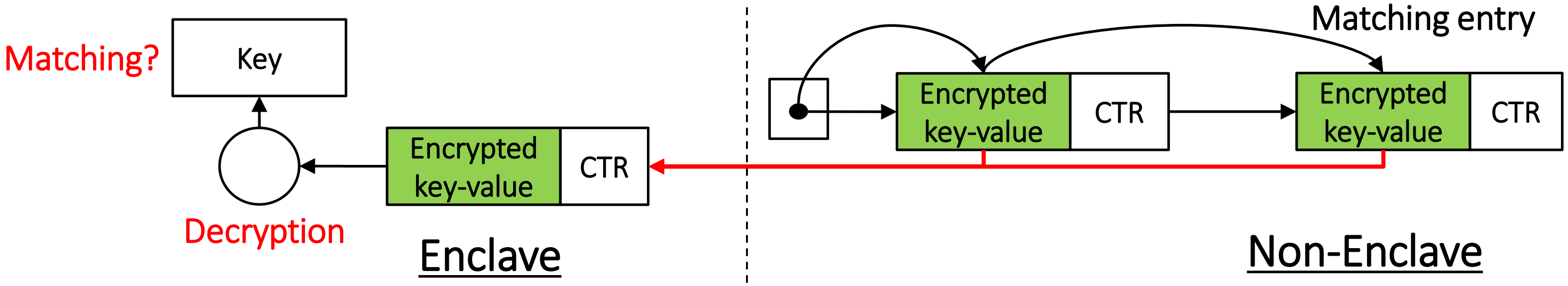
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



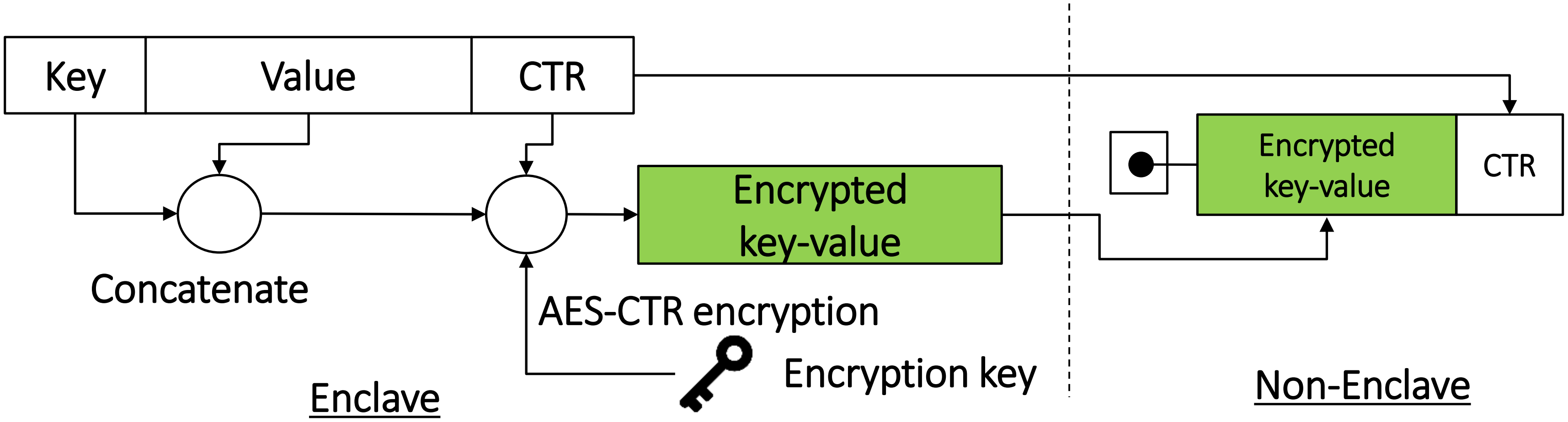


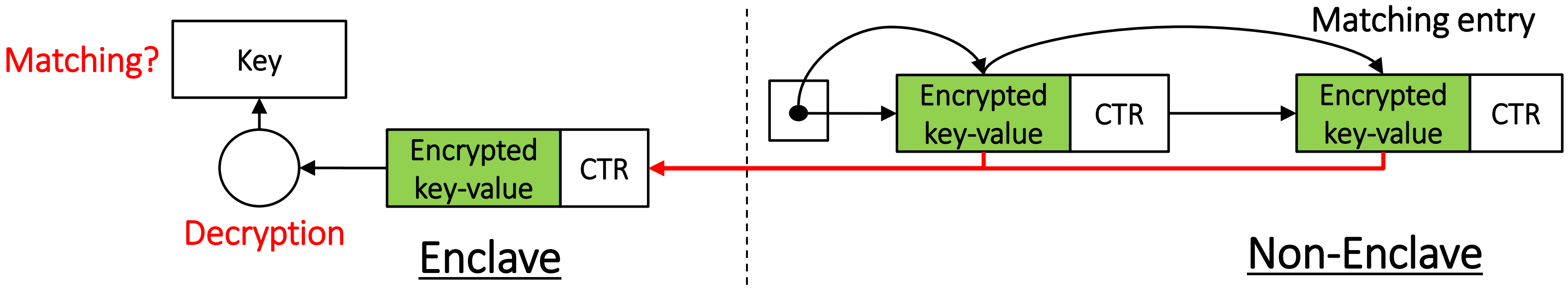
Encryption

- ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



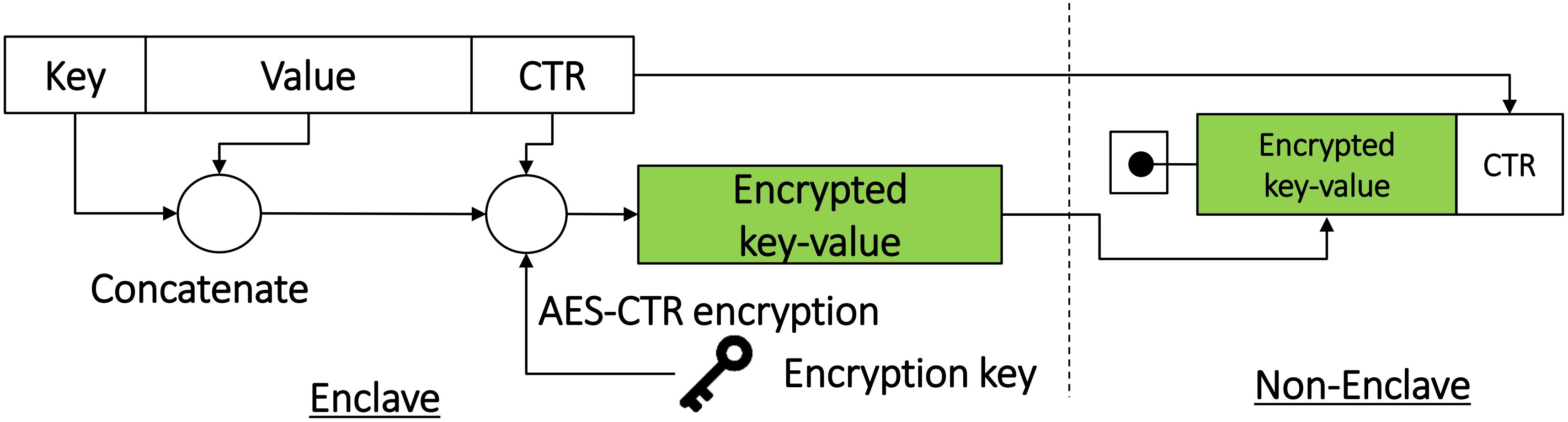


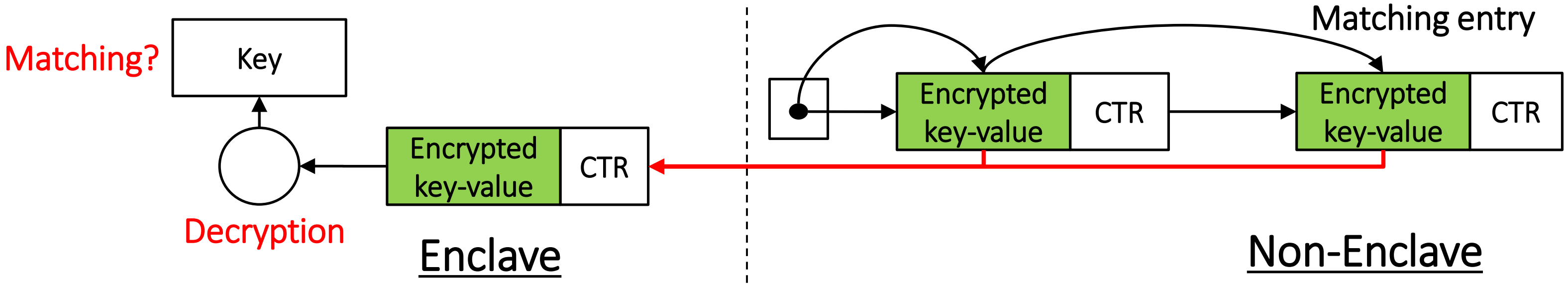
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



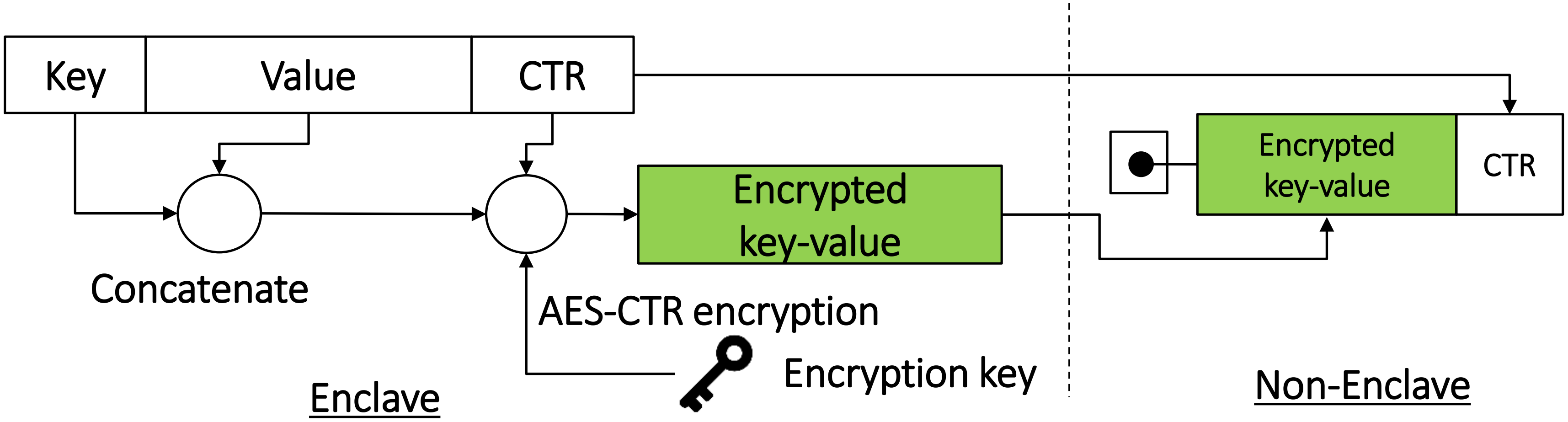


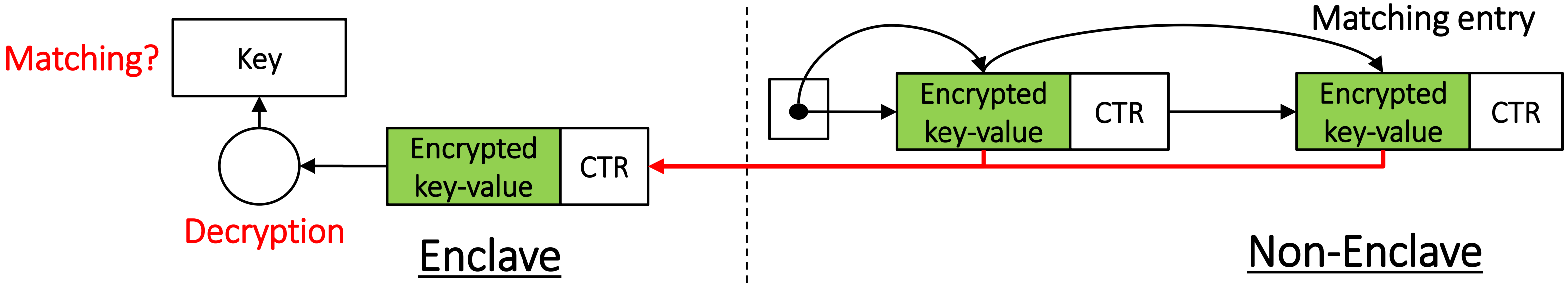
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket



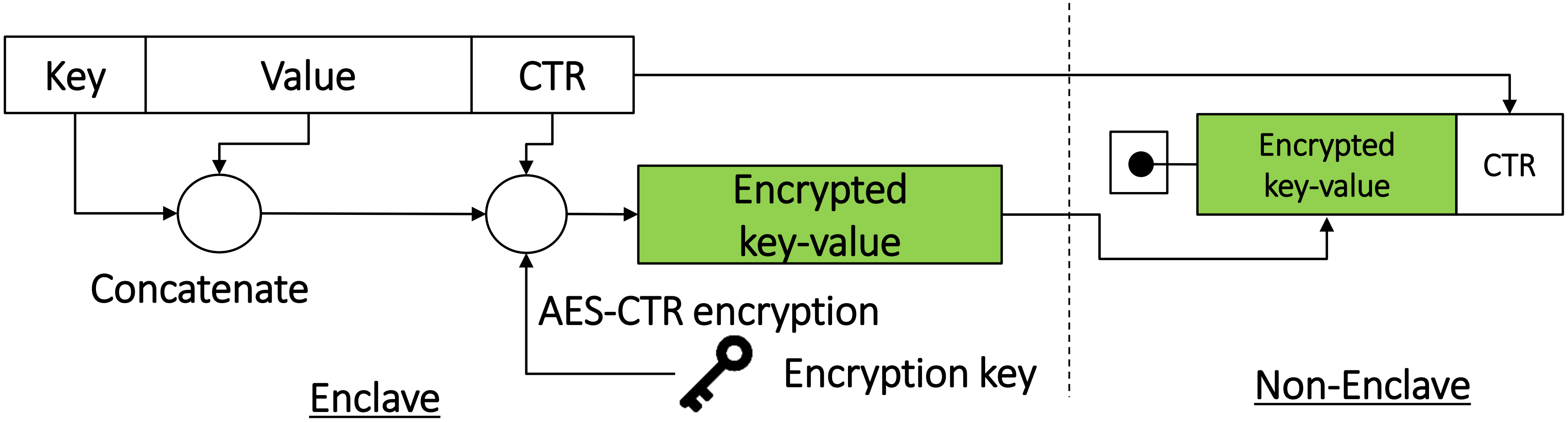


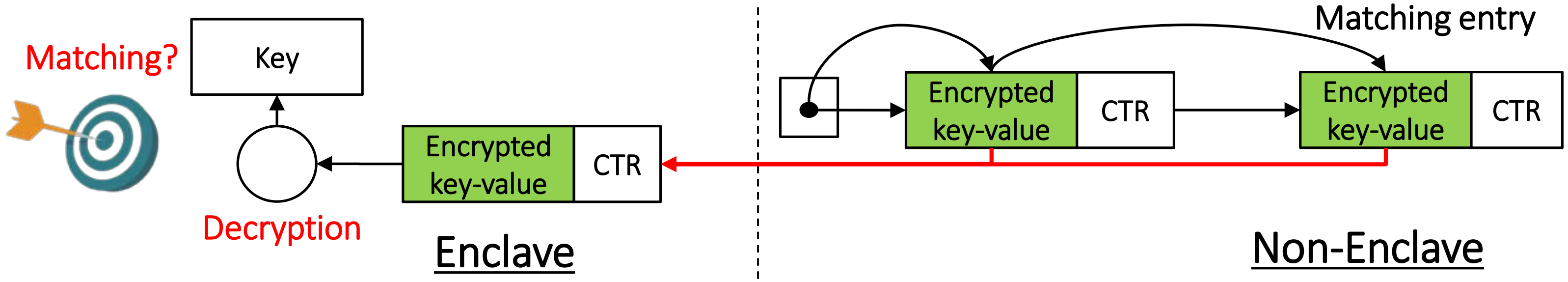
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket





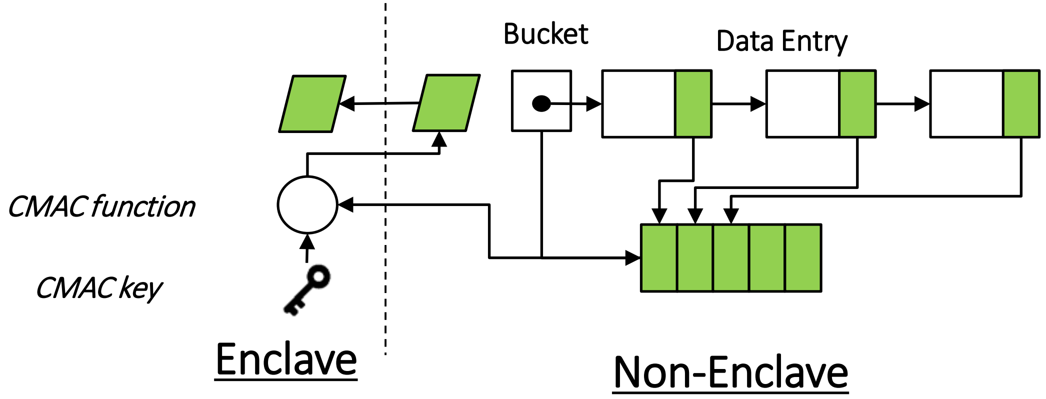
Optimizations

- MAC bucketing
  - Maintain the MAC buffer per a hash bucket

- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading

Optimizations

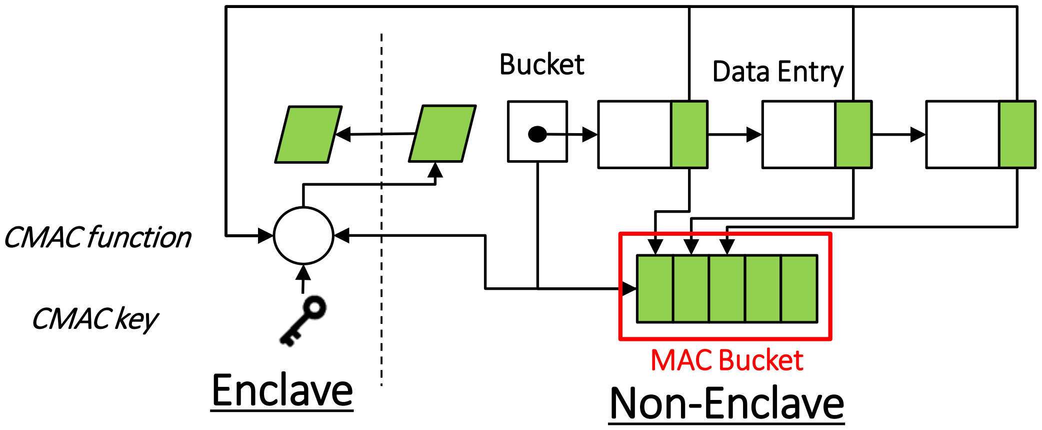
- MAC bucketing
  - Maintain the MAC buffer per a hash bucket



- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading

Optimizations

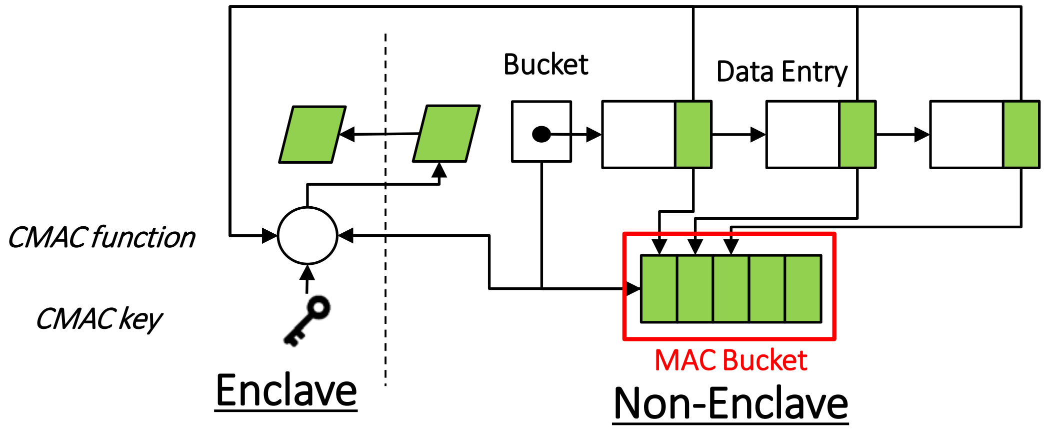
- MAC bucketing
  - Maintain the MAC buffer per a hash bucket



- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading

Optimizations

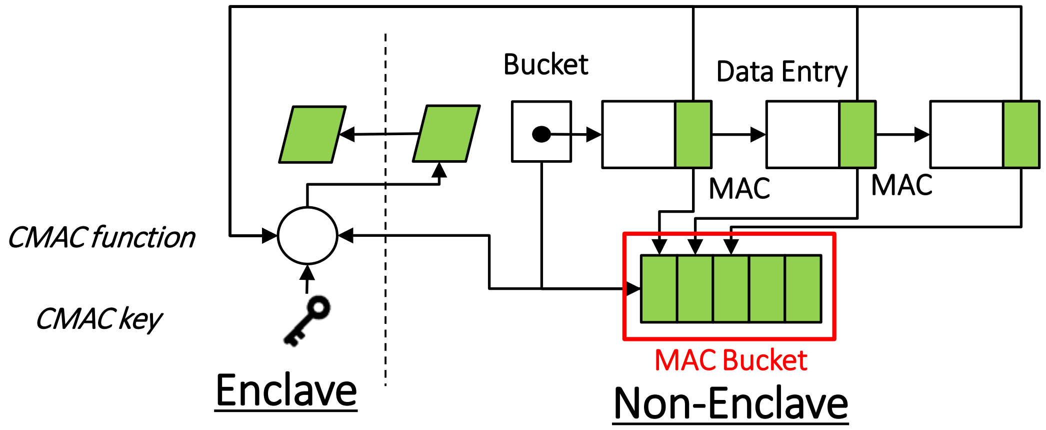
- MAC bucketing
  - Maintain the MAC buffer per a hash bucket



- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading

Optimizations

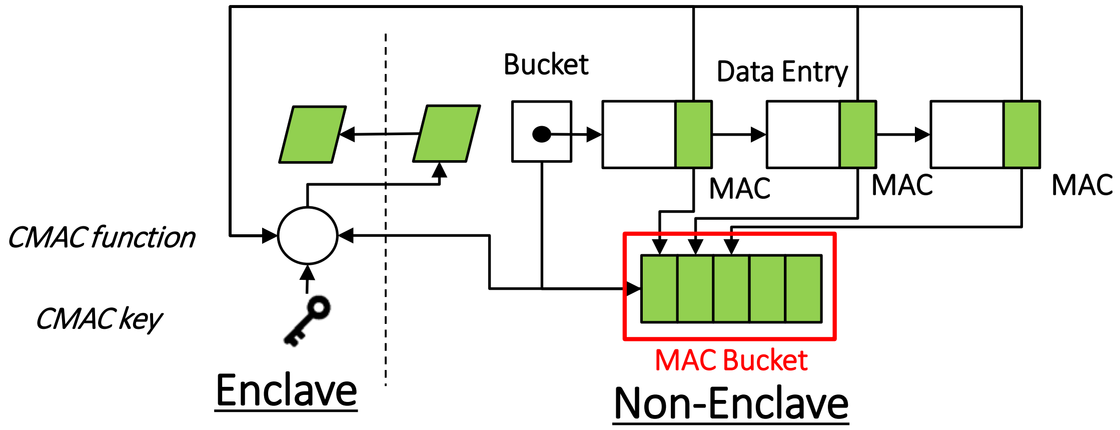
- MAC bucketing
  - Maintain the MAC buffer per a hash bucket



- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading

Optimizations

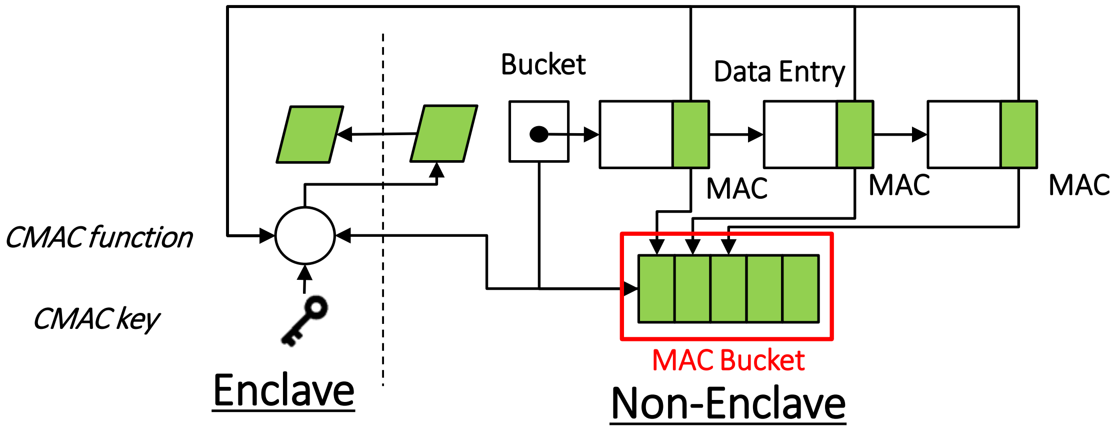
- MAC bucketing
  - Maintain the MAC buffer per a hash bucket



- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading

Optimizations

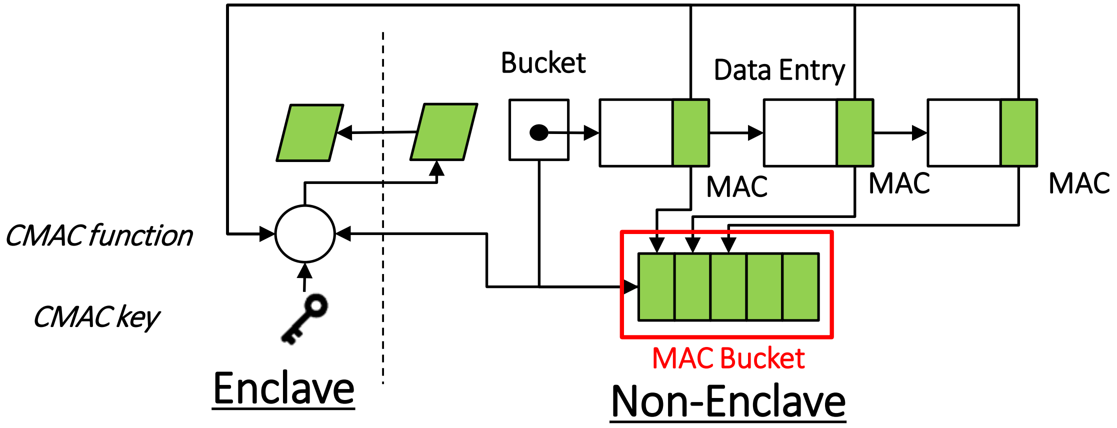
- MAC bucketing
  - Maintain the MAC buffer per a hash bucket



- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading

Optimizations

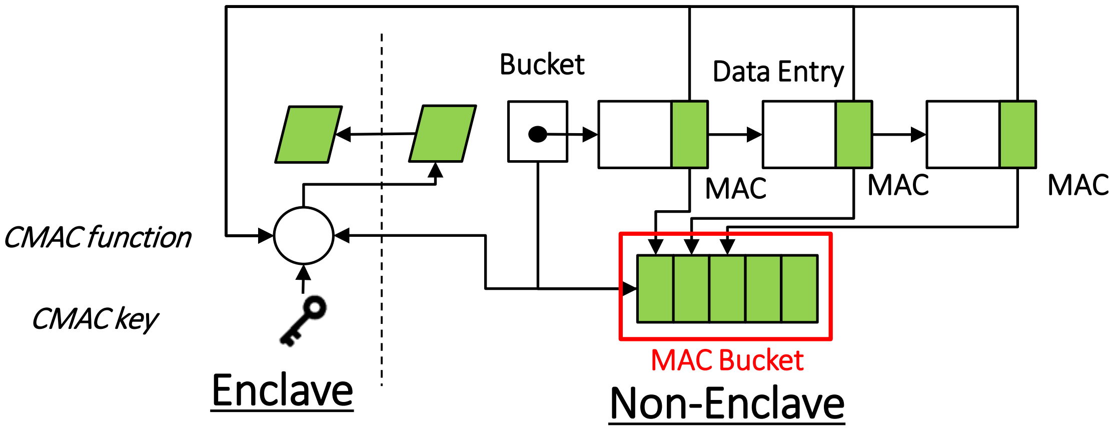
- MAC bucketing
  - Maintain the MAC buffer per a hash bucket



- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading

Optimizations

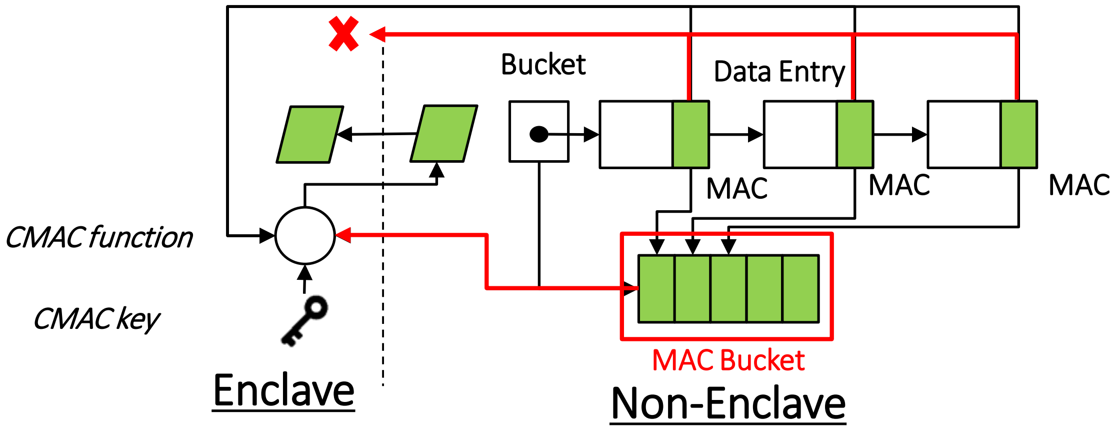
- MAC bucketing
  - Maintain the MAC buffer per a hash bucket



- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading

Optimizations

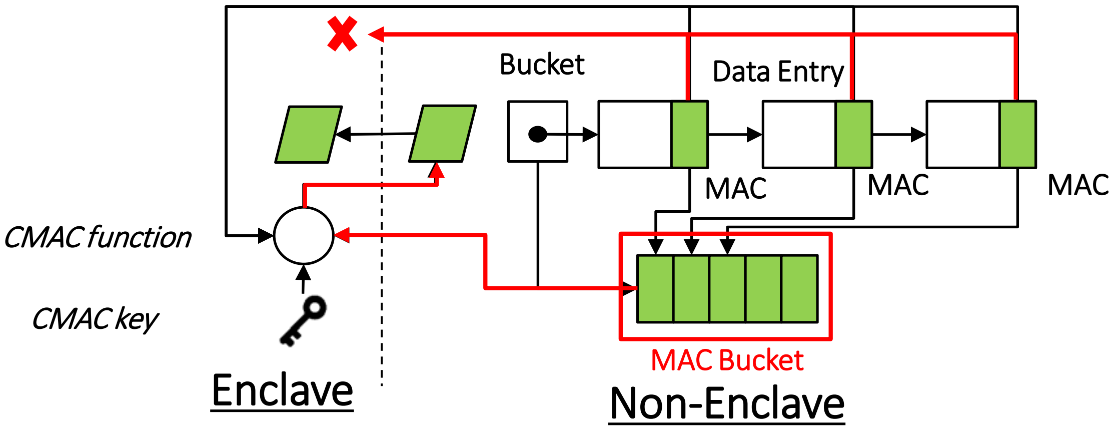
- MAC bucketing
  - Maintain the MAC buffer per a hash bucket



- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading

Optimizations

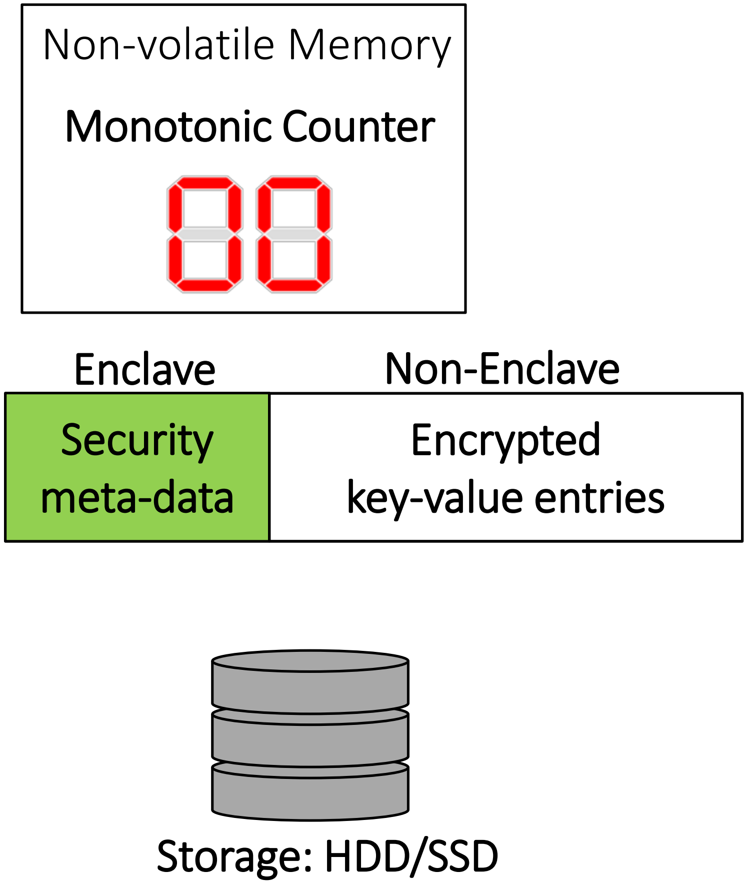
- MAC bucketing
  - Maintain the MAC buffer per a hash bucket



- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading

Persistent Support

- Intel SGX supports *sealing mechanism*
  - Using *monotonic counter* stored in non-volatile memory
  - Protect data from rollback attacks



Non-volatile Memory

Monotonic Counter

Enclave

Non-Enclave

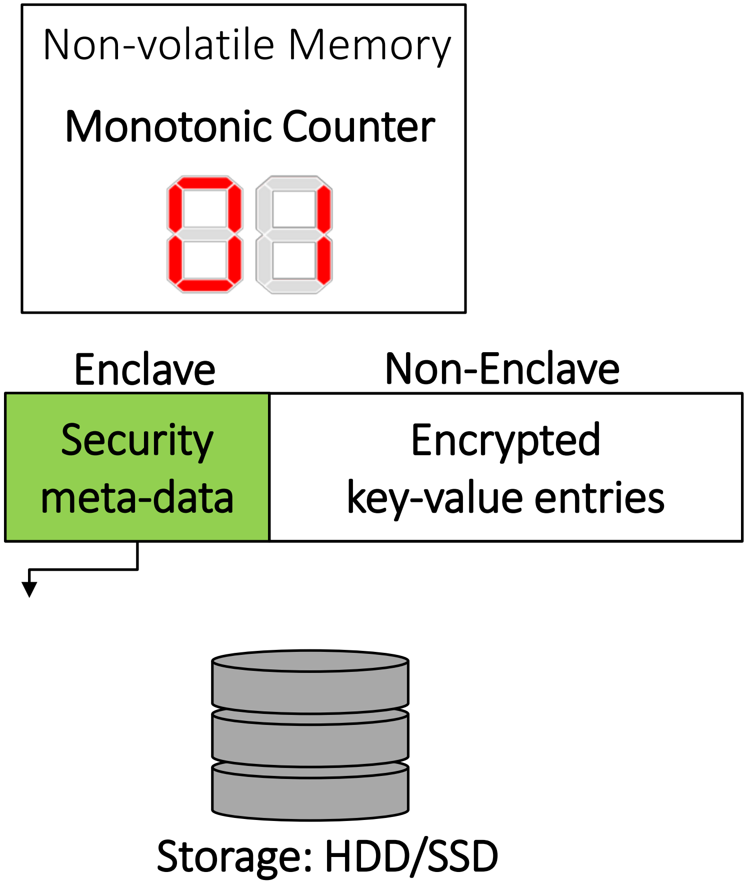
Security meta-data

Encrypted key-value entries

Storage: HDD/SSD

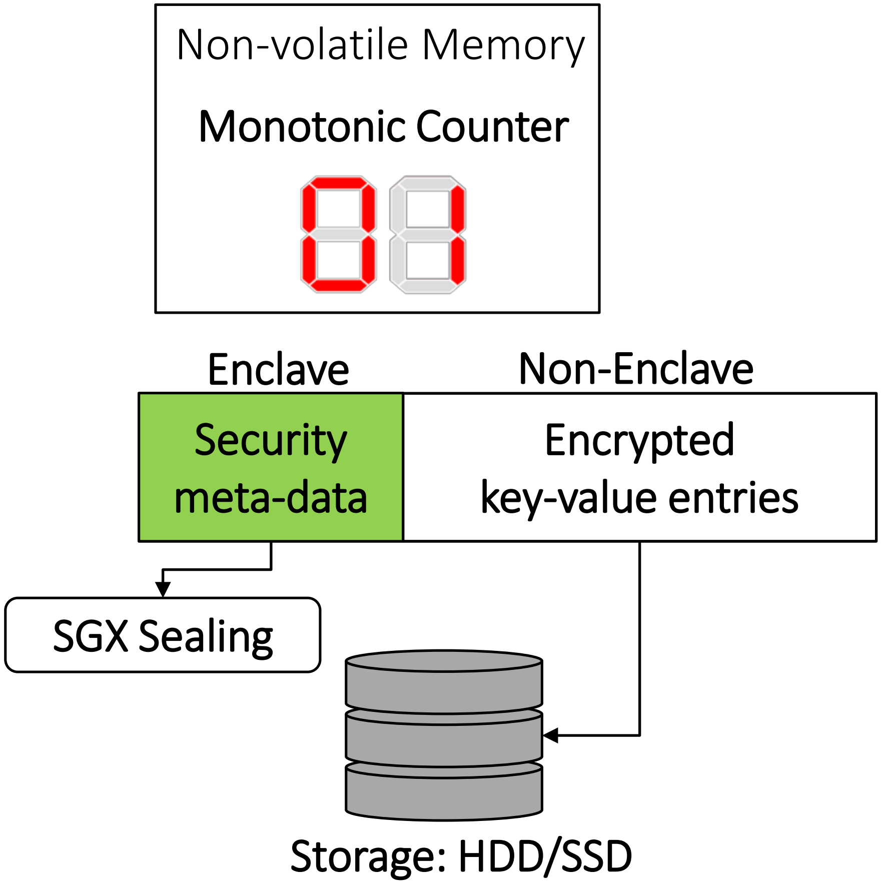
Persistent Support

- Intel SGX supports *sealing mechanism*
  - Using *monotonic counter* stored in non-volatile memory
  - Protect data from rollback attacks



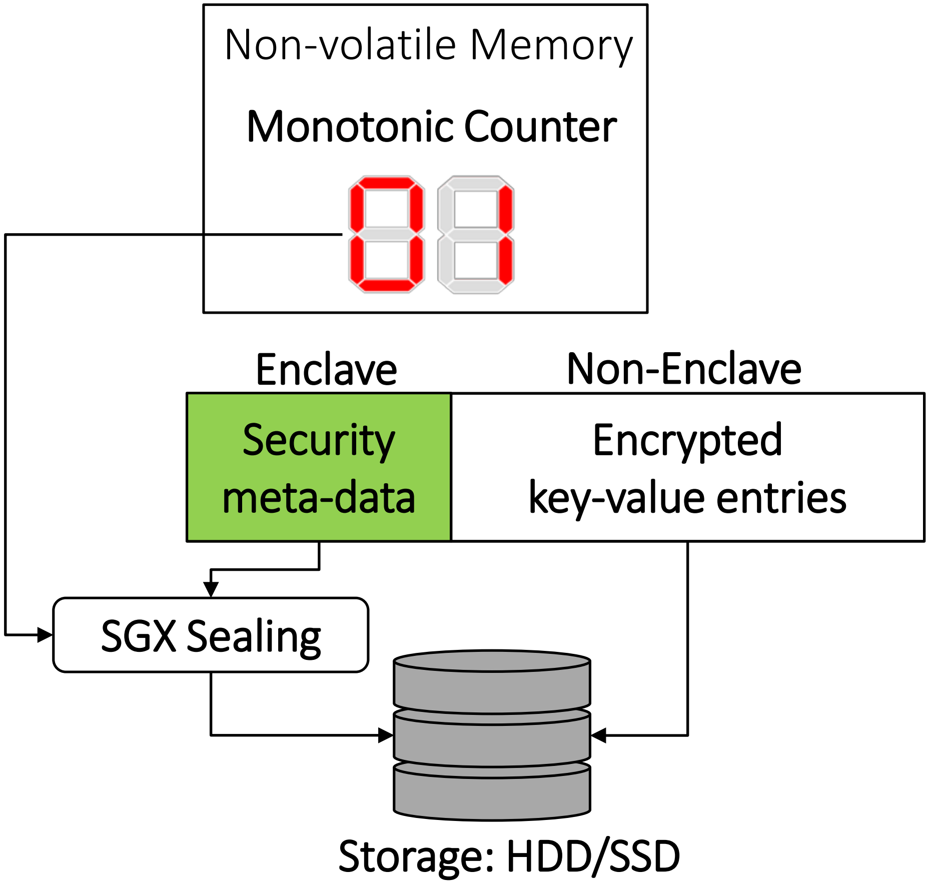
Persistent Support

- Intel SGX supports *sealing mechanism*
  - Using *monotonic counter* stored in non-volatile memory
  - Protect data from rollback attacks



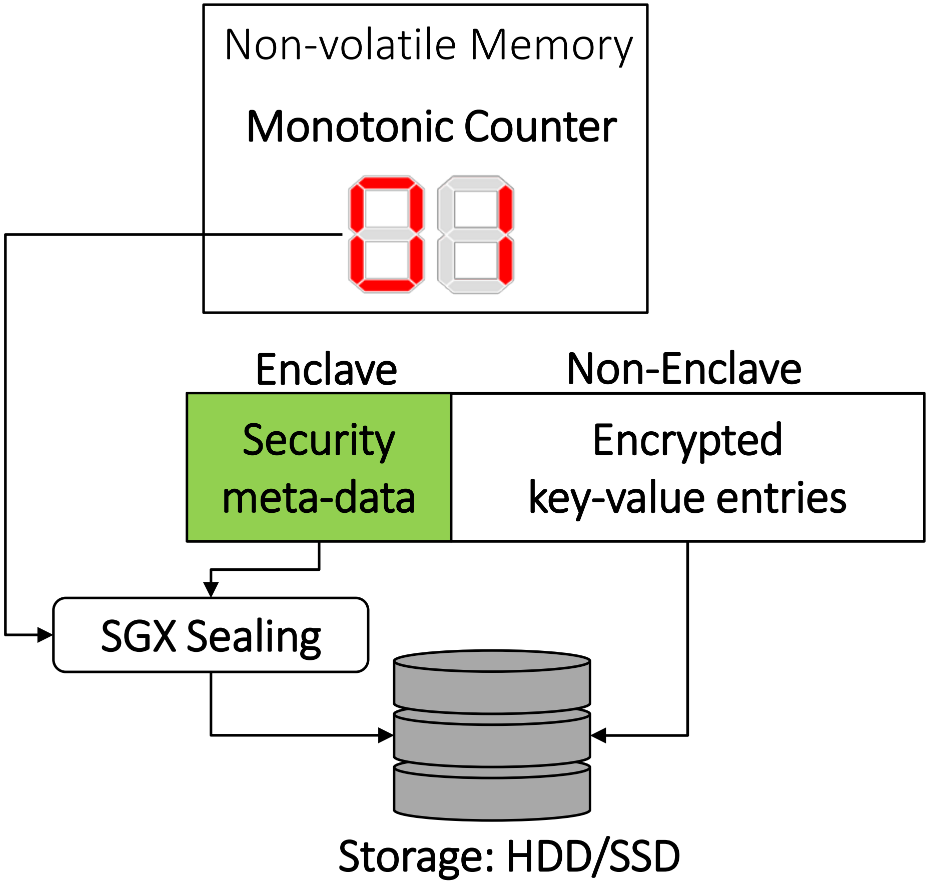
Persistent Support

- Intel SGX supports *sealing mechanism*
  - Using *monotonic counter* stored in non-volatile memory
  - Protect data from rollback attacks



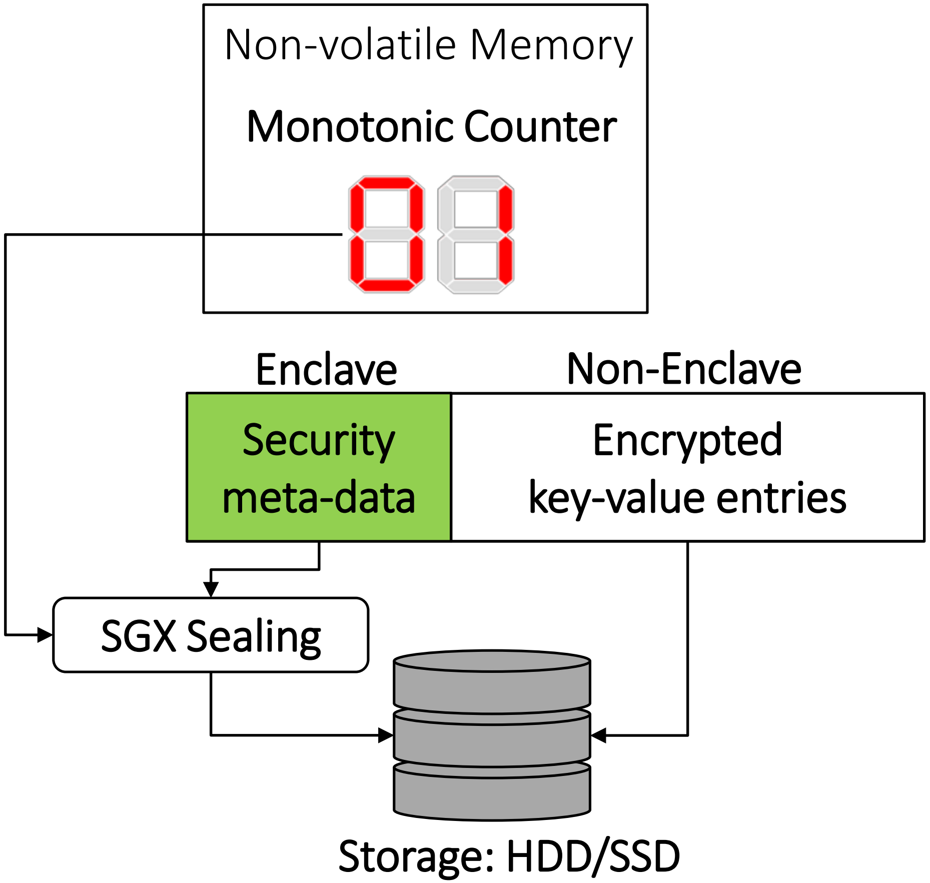
Persistent Support

- Intel SGX supports *sealing mechanism*
  - Using *monotonic counter* stored in non-volatile memory
  - Protect data from rollback attacks



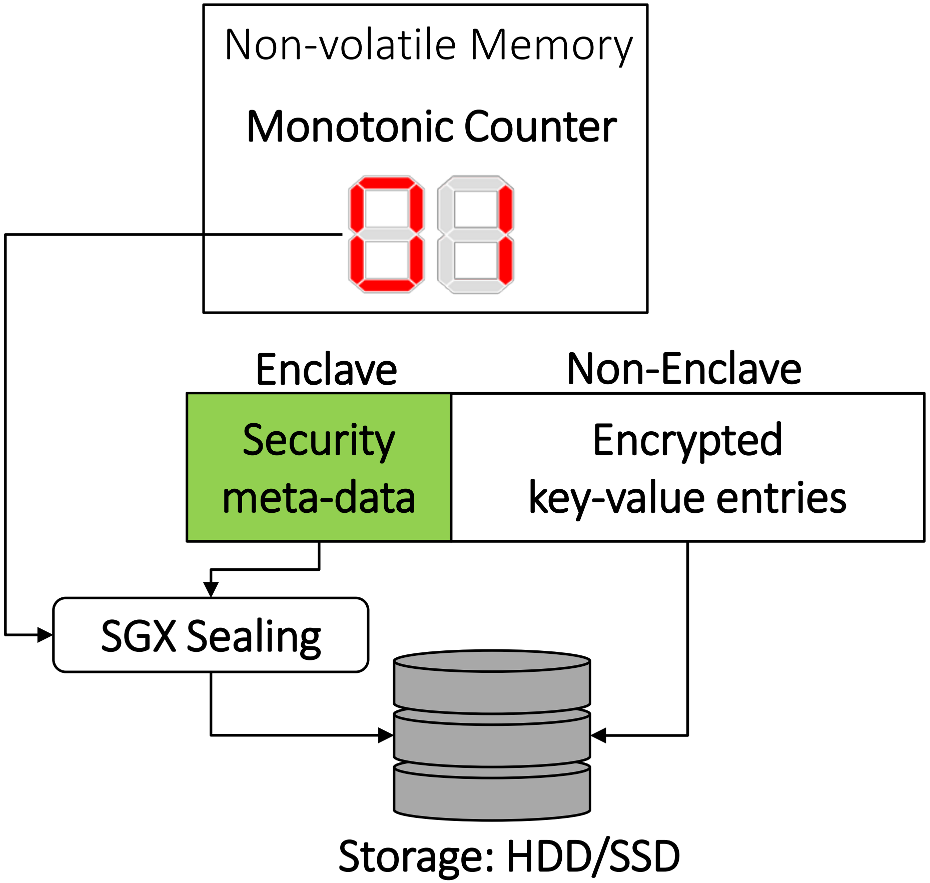
Persistent Support

- Intel SGX supports *sealing mechanism*
  - Using *monotonic counter* stored in non-volatile memory
  - Protect data from rollback attacks



Persistent Support

- Intel SGX supports *sealing mechanism*
  - Using *monotonic counter* stored in non-volatile memory
  - Protect data from rollback attacks



Experimental Setup

- Evaluation
  - Standalone: Focus on data store aspect without network
  - Network: Socket interface with a 10Gb NIC and 256 concurrent clients

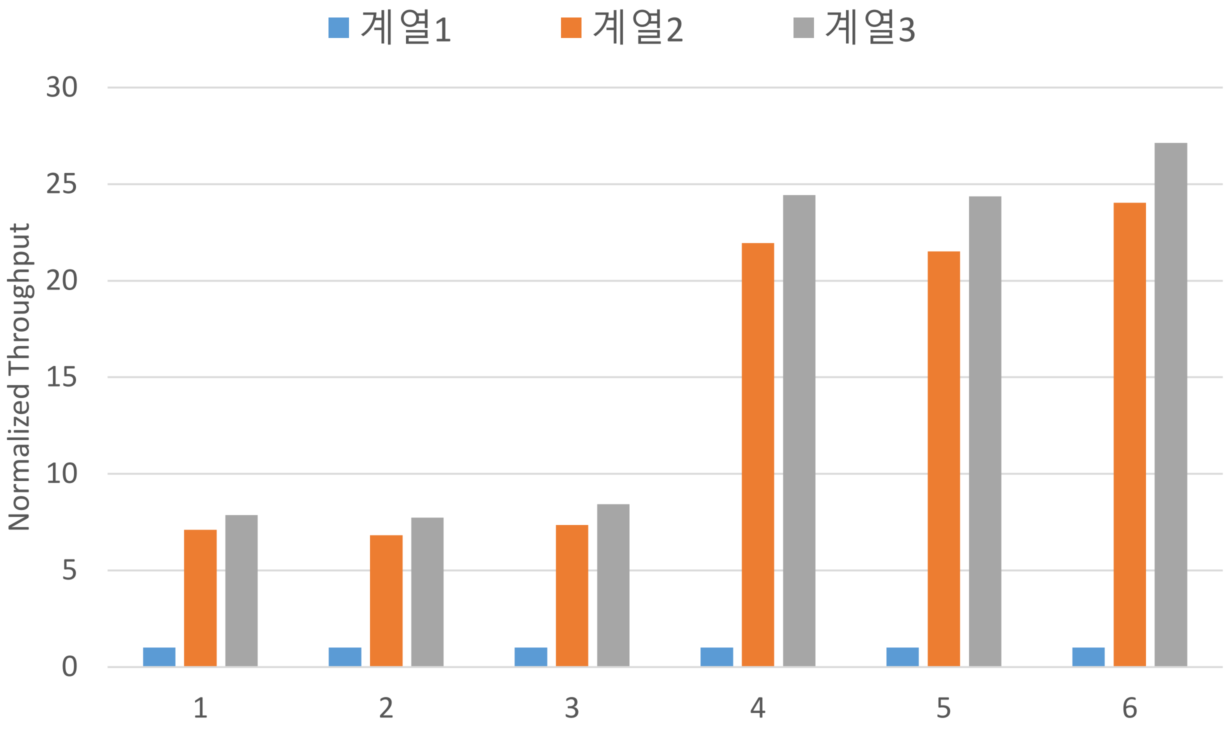
- Metrics
  - *Secure Memcached*: memcached with grapheneSGX [2]
  - *ShieldBase*: ShieldStore without optimizations
  - *ShieldOPT*: ShieldStore with optimizations

Data Set	Key Size(B)	Value Size(B)	Working set(MB)
Small	16	16	305
Medium	16	128	1,373
Large	16	512	5,035

[2] Tsai, et al. Grahpene-SGX: A Practical Library OS for Unmodified Applications on SGX [USENIX ATC' 17]

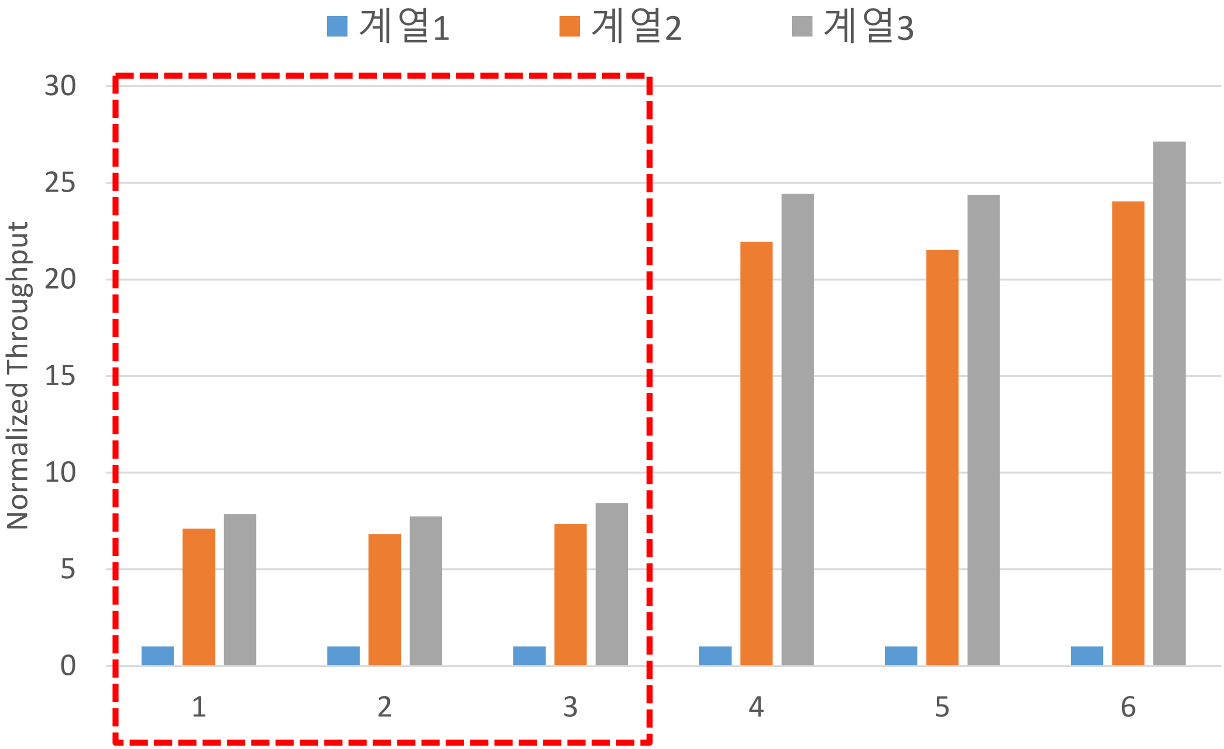
Standalone Evaluation

- ShieldStore performs
  - – 7 – 8 times better than *Secure Memcached* on 1 thread
  - – 24 – 27 times better than *Secure Memcached* on 4 threads



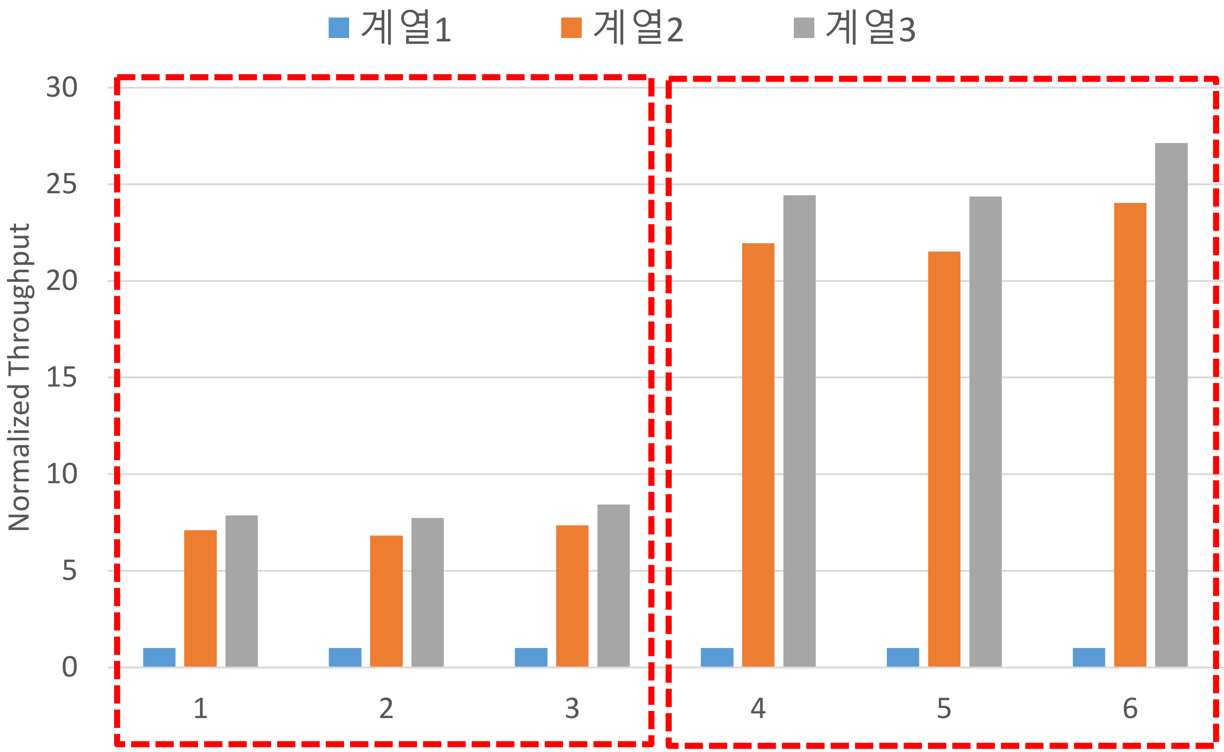
Standalone Evaluation

- ShieldStore performs
  - 7 – 8 times better than *Secure Memcached* on 1 thread
  - 24 – 27 times better than *Secure Memcached* on 4 threads



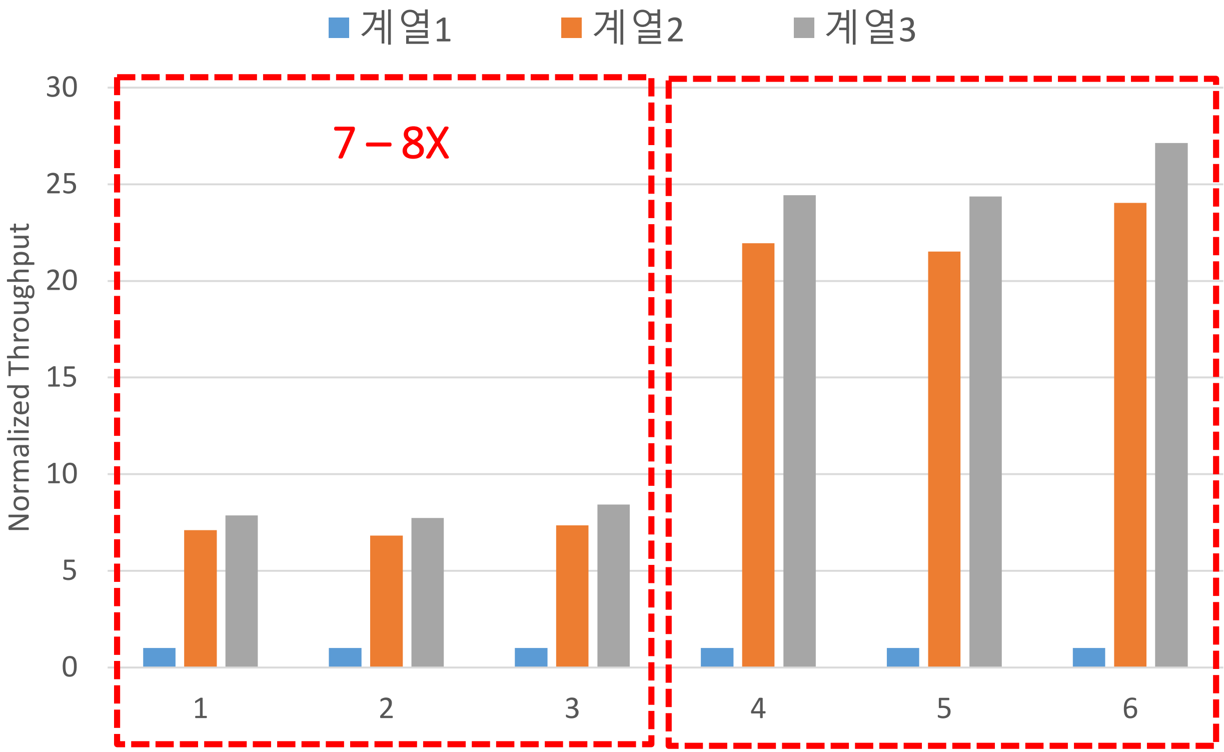
Standalone Evaluation

- ShieldStore performs
  - 7 – 8 times better than *Secure Memcached* on 1 thread
  - 24 – 27 times better than *Secure Memcached* on 4 threads



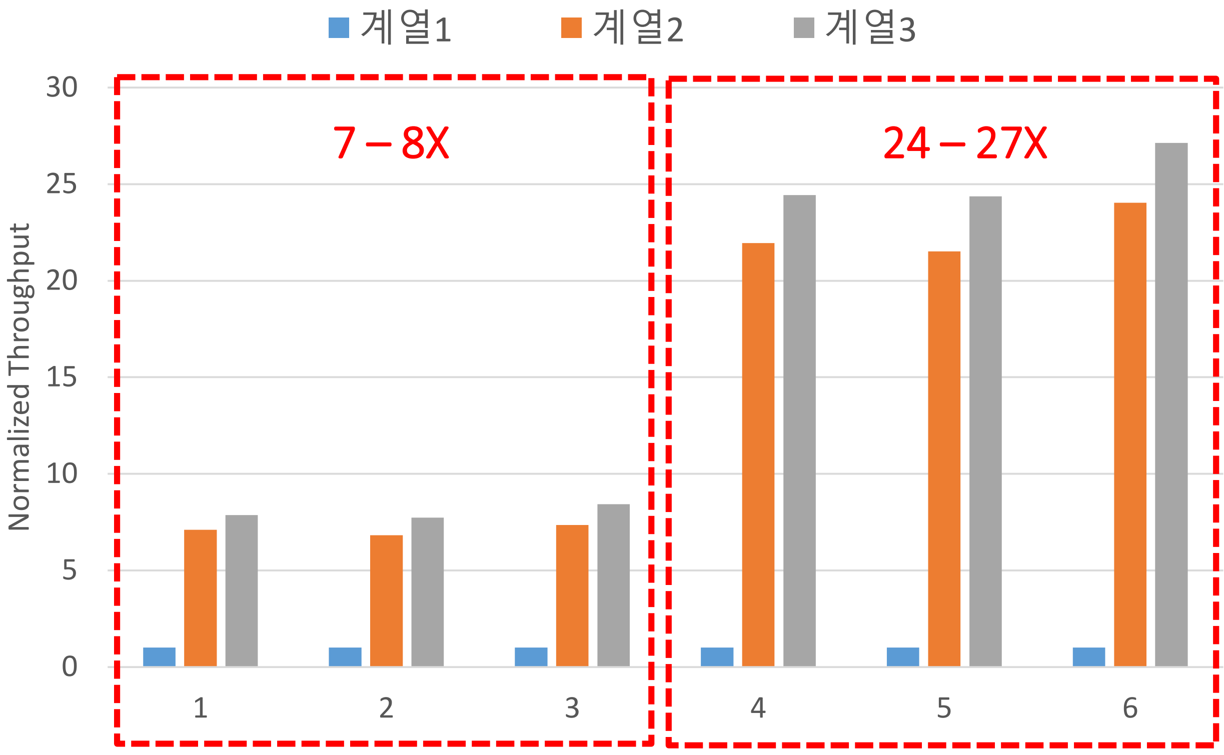
Standalone Evaluation

- ShieldStore performs
  - – 7 – 8 times better than *Secure Memcached* on 1 thread
  - – 24 – 27 times better than *Secure Memcached* on 4 threads



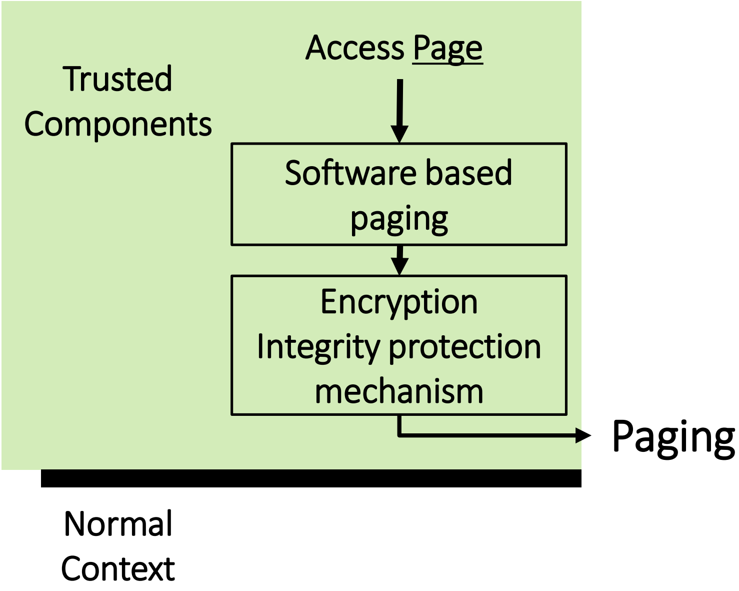
Standalone Evaluation

- ShieldStore performs
  - 7 – 8 times better than *Secure Memcached* on 1 thread
  - 24 – 27 times better than *Secure Memcached* on 4 threads



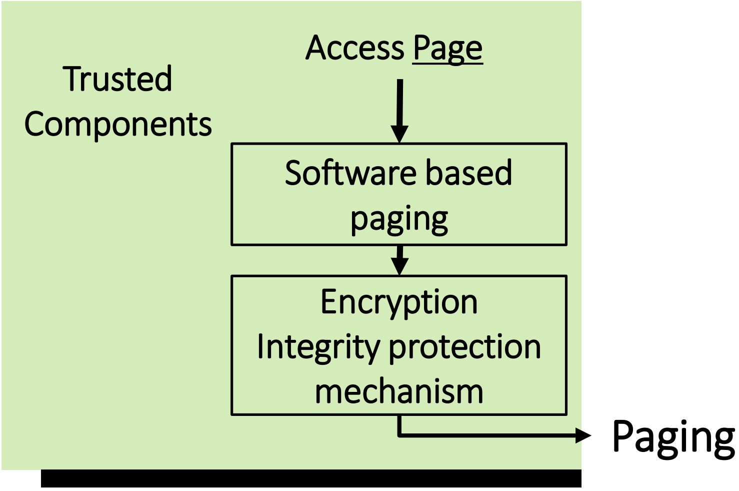
Comparison to Key-value Store on Eleos [1/2]

- Eleos provides coarse-grained user space memory paging
  - Eleos provides 1KB/4KB page-grained protection
  - ShieldStore provides fine-grained data protection



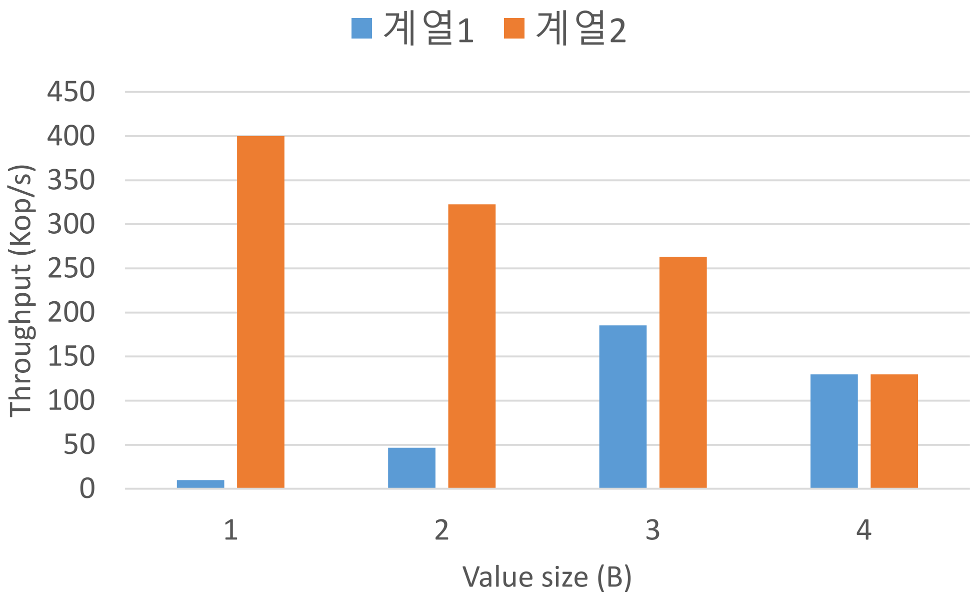
Comparison to Key-value Store on Eleos [1/2]

- Eleos provides coarse-grained user space memory paging
  - Eleos provides 1KB/4KB page-grained protection
  - ShieldStore provides fine-grained data protection



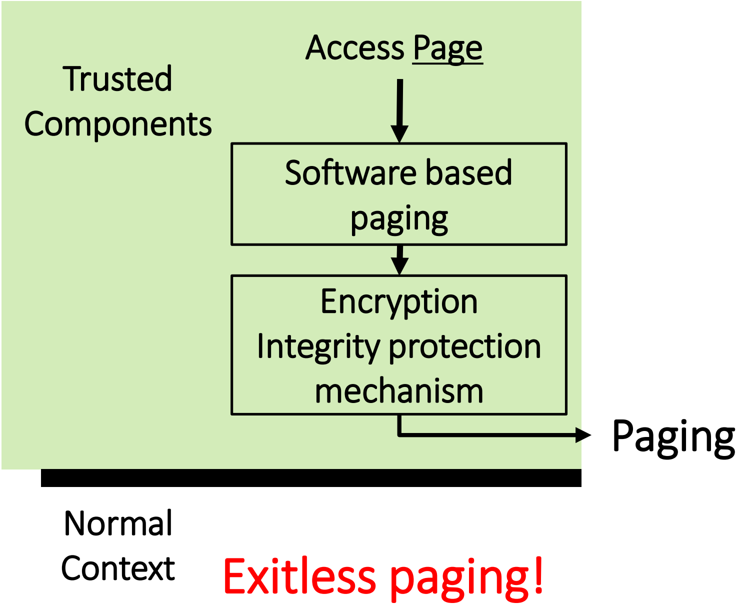
Normal Context

Exitless paging!



Comparison to Key-value Store on Eleos [1/2]

- Eleos provides coarse-grained user space memory paging
  - Eleos provides 1KB/4KB page-grained protection
  - ShieldStore provides fine-grained data protection


Trusted Components

Access Page

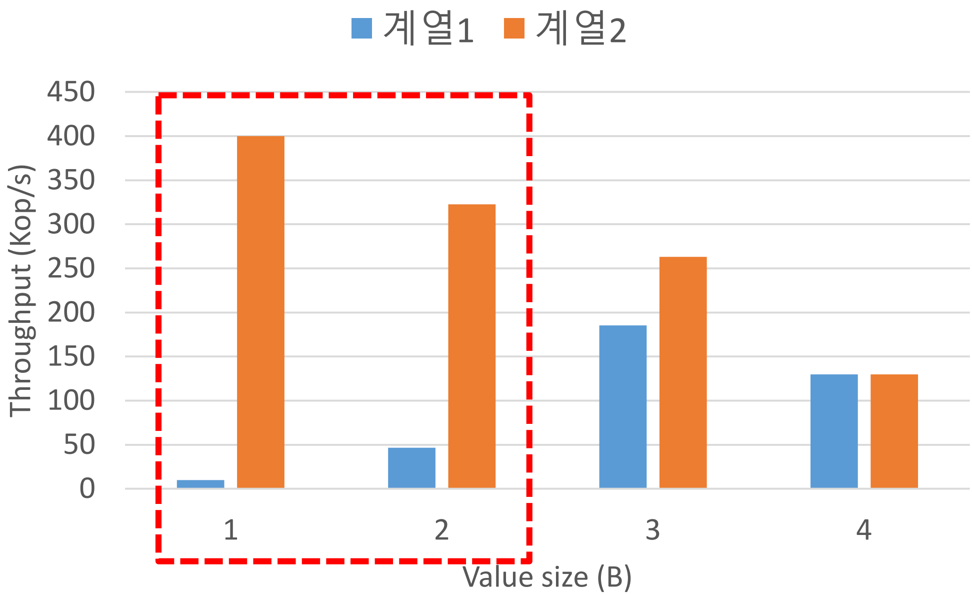
Software based paging

Encryption Integrity protection mechanism

Paging

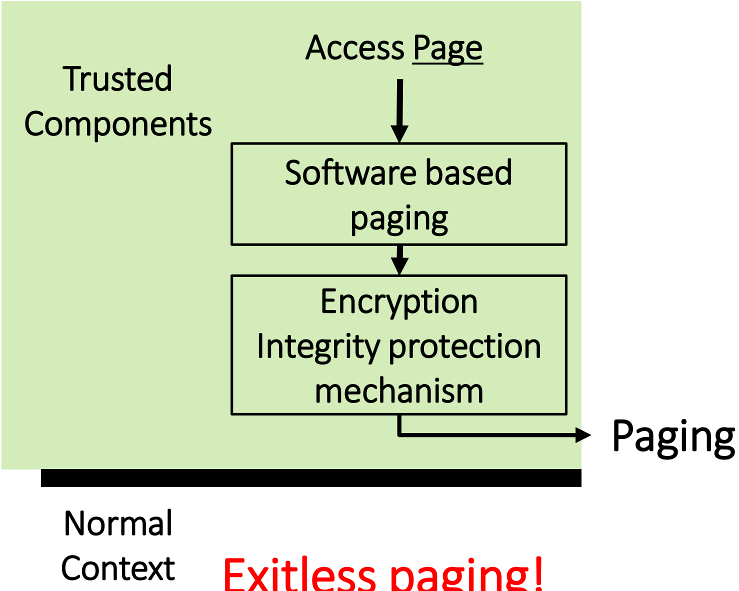
Normal Context

Exitless paging!

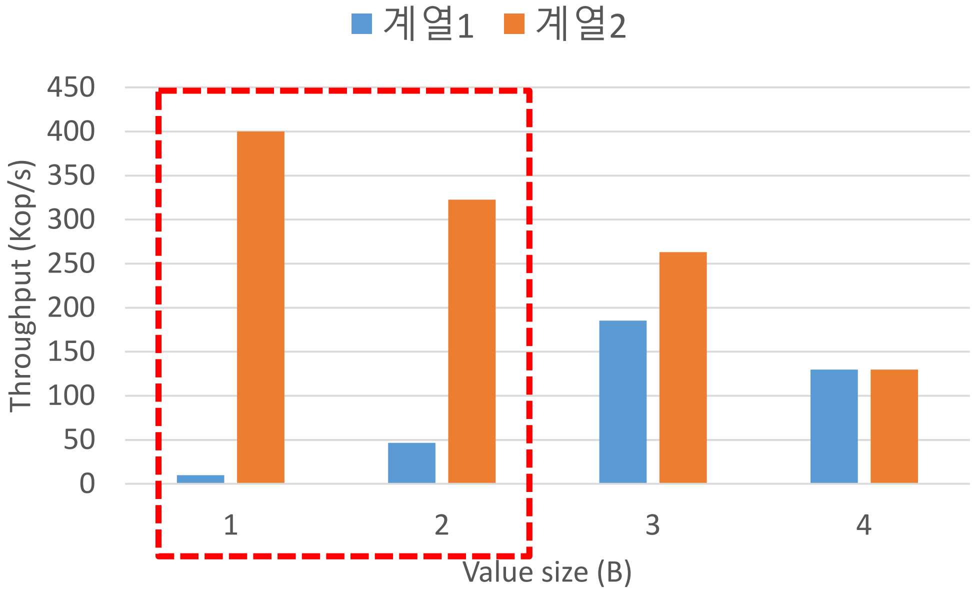


Comparison to Key-value Store on Eleos [1/2]

- Eleos provides coarse-grained user space memory paging
  - Eleos provides 1KB/4KB page-grained protection
  - ShieldStore provides fine-grained data protection

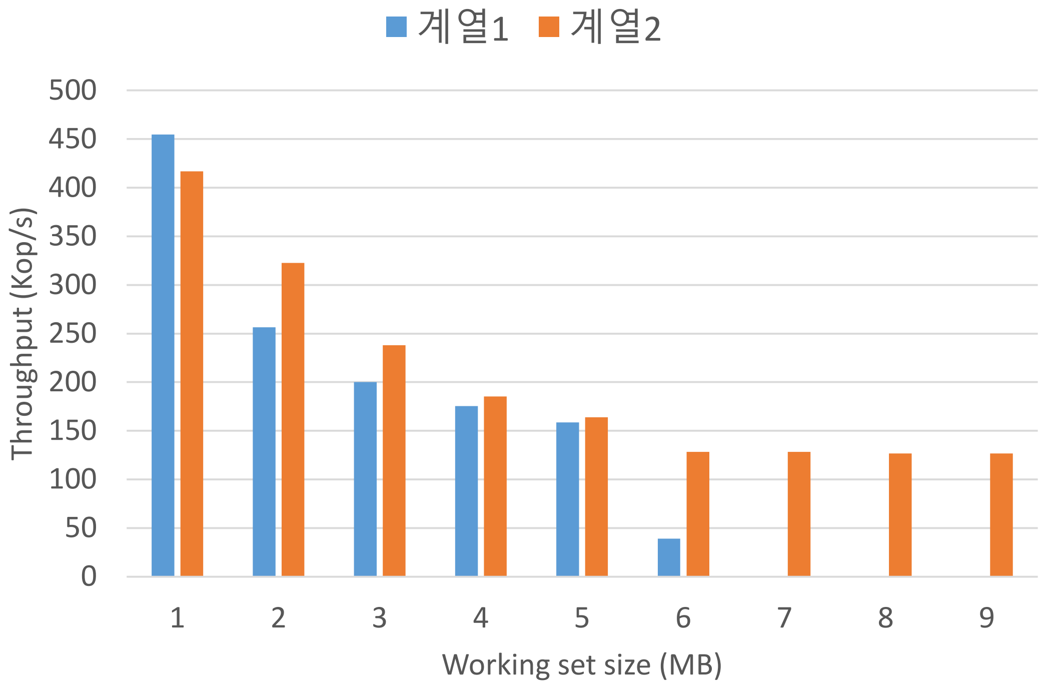


Exitless paging!



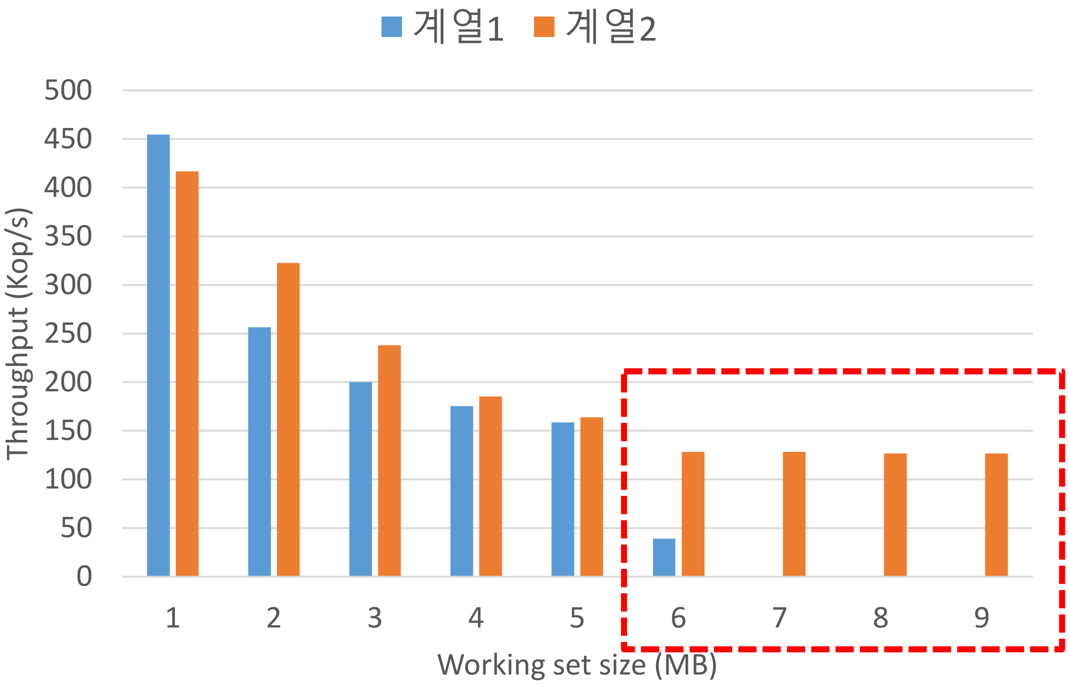
Comparison to Key-value Store on Eleos [2/2]

- ShieldStore performs better than Eleos even with 4KB value
  - Efficient data protection improves the performance of ShieldStore



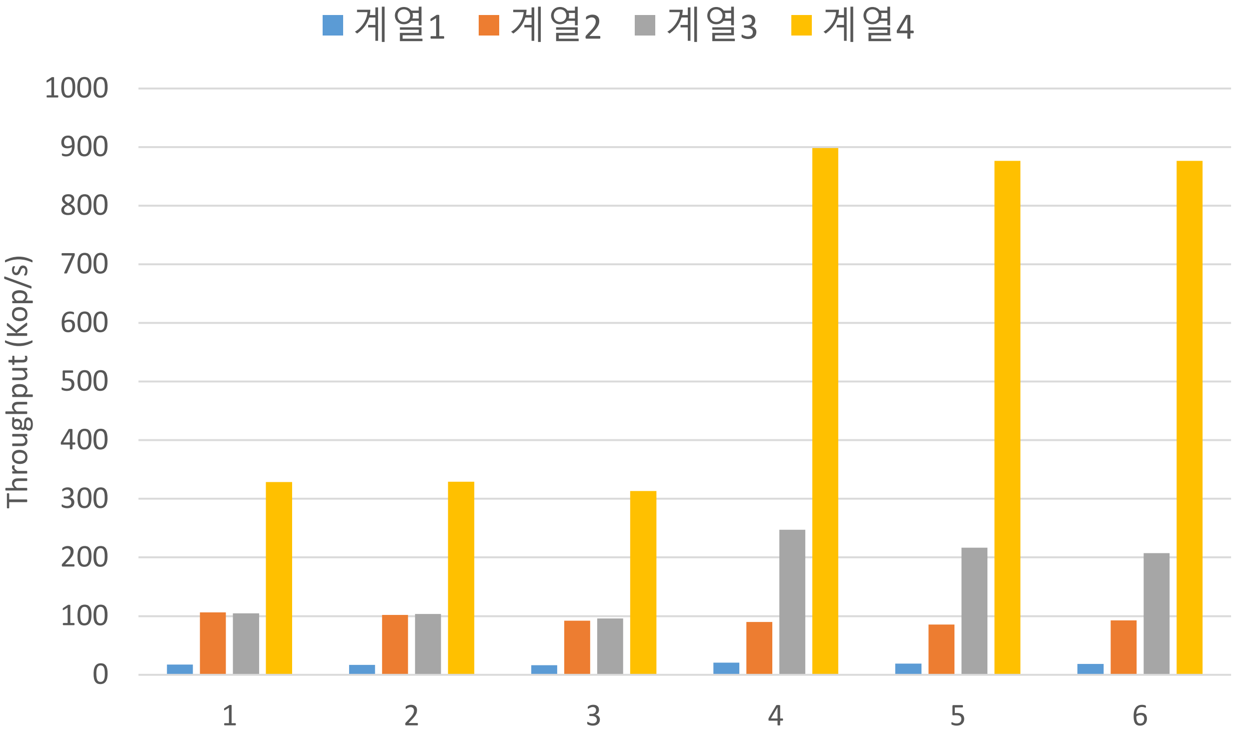
Comparison to Key-value Store on Eleos [2/2]

- ShieldStore performs better than Eleos even with 4KB value
  - – Efficient data protection improves the performance of ShieldStore



Network Evaluation

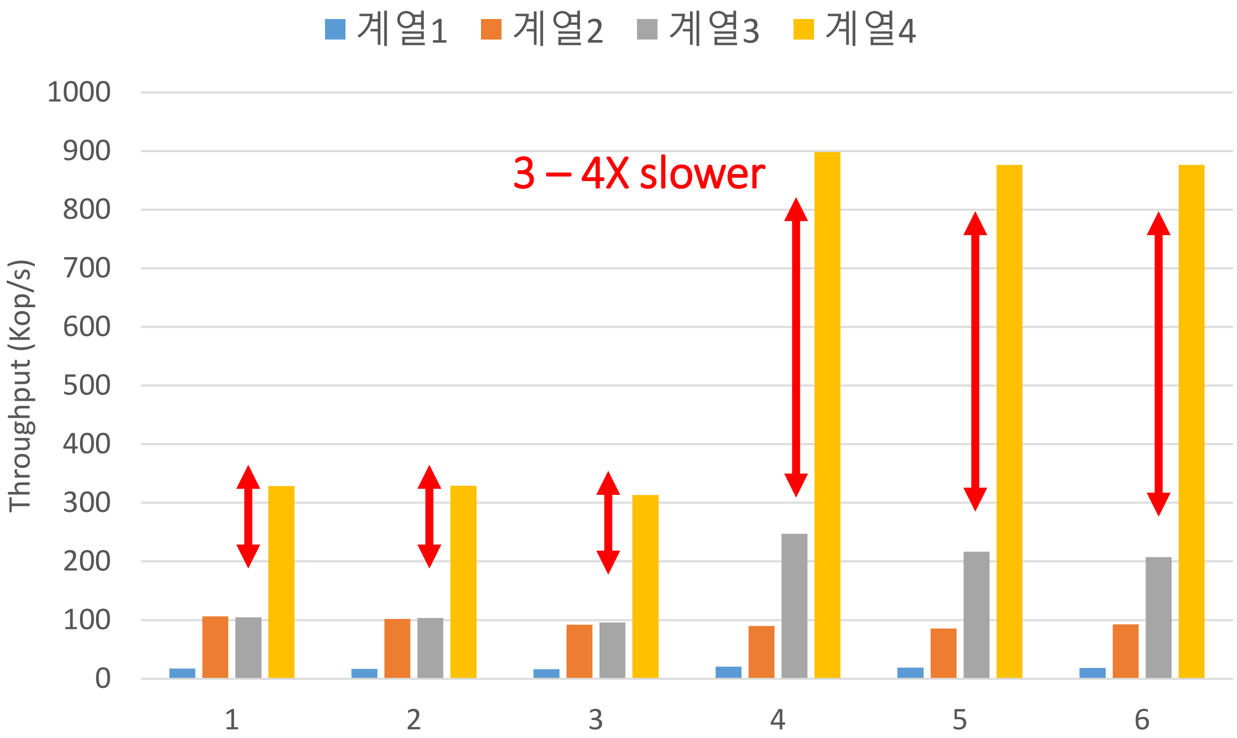
- ShieldStore with HotCalls [3] performs
  - 6 – 11 times better than *Secure Memcached* on 1 thread and 4 threads
  - 3 – 4 times slower than *Insecure Memcached* on 1 thread and 4 threads



[3] Weisse, et al. Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves [ISCA' 17]

Network Evaluation

- ShieldStore with HotCalls [3] performs
  - 6 – 11 times better than *Secure Memcached* on 1 thread and 4 threads
  - 3 – 4 times slower than *Insecure Memcached* on 1 thread and 4 threads



[3] Weisse, et al. Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves [ISCA' 17]

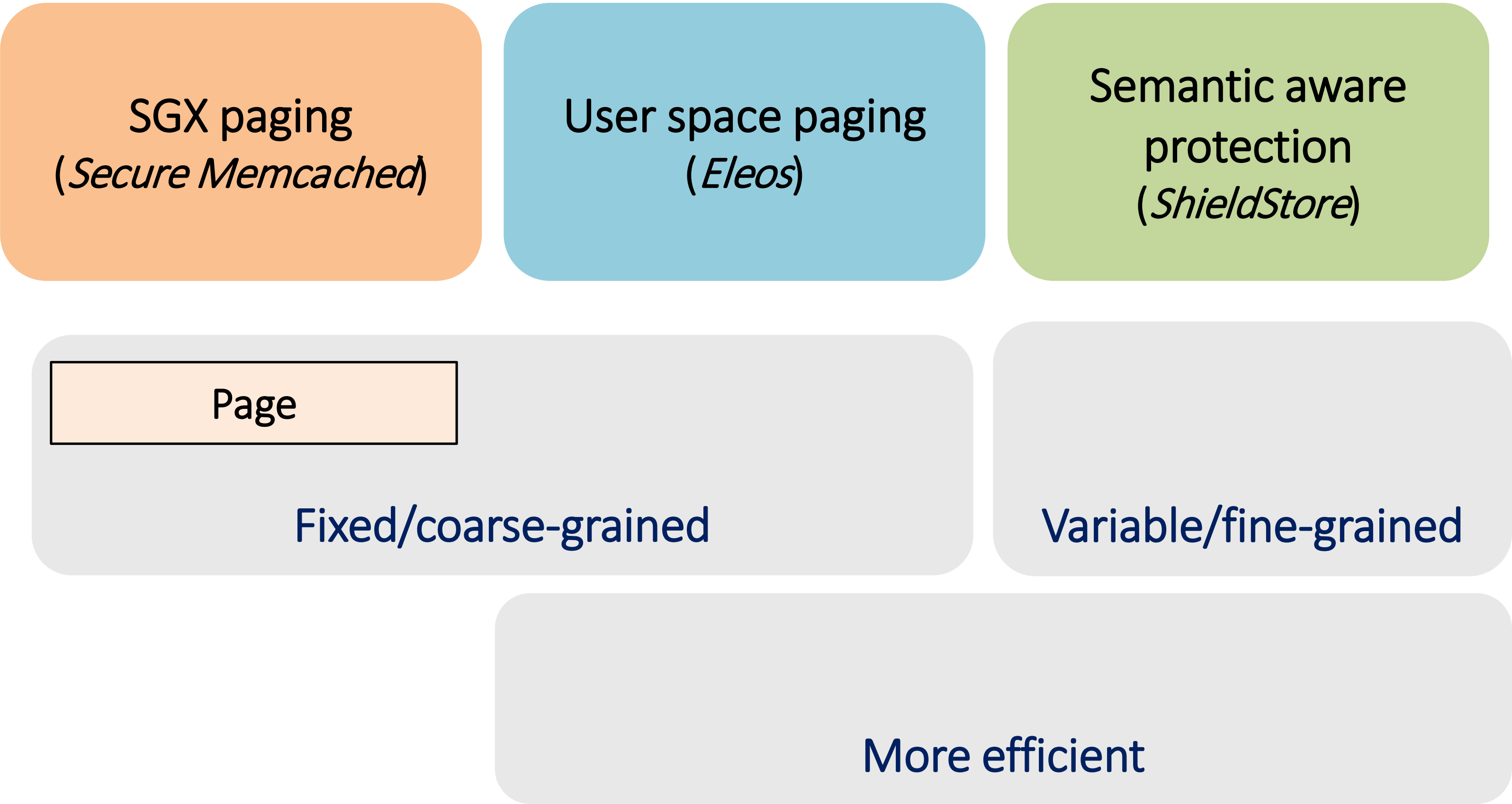
Summary of Paging Principles

SGX paging
(*Secure Memcached*)

User space paging
(*Eleos*)

Semantic aware protection
(*ShieldStore*)

Summary of Paging Principles



SGX paging (*Secure Memcached*)

User space paging (*Eleos*)

Semantic aware protection (*ShieldStore*)

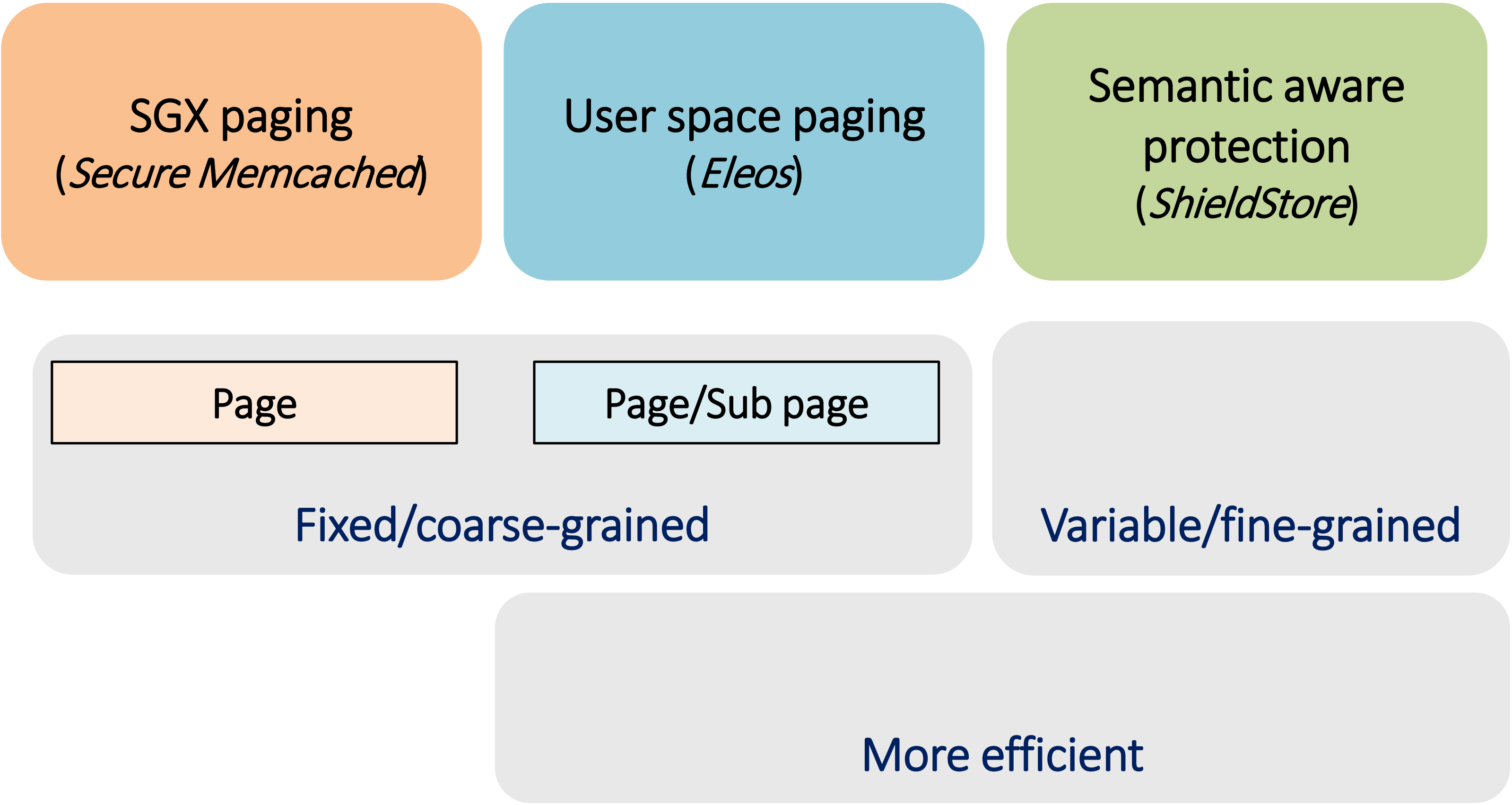
Page

Fixed/coarse-grained

Variable/fine-grained

More efficient

Summary of Paging Principles



SGX paging (*Secure Memcached*)

User space paging (*Eleos*)

Semantic aware protection (*ShieldStore*)

Page

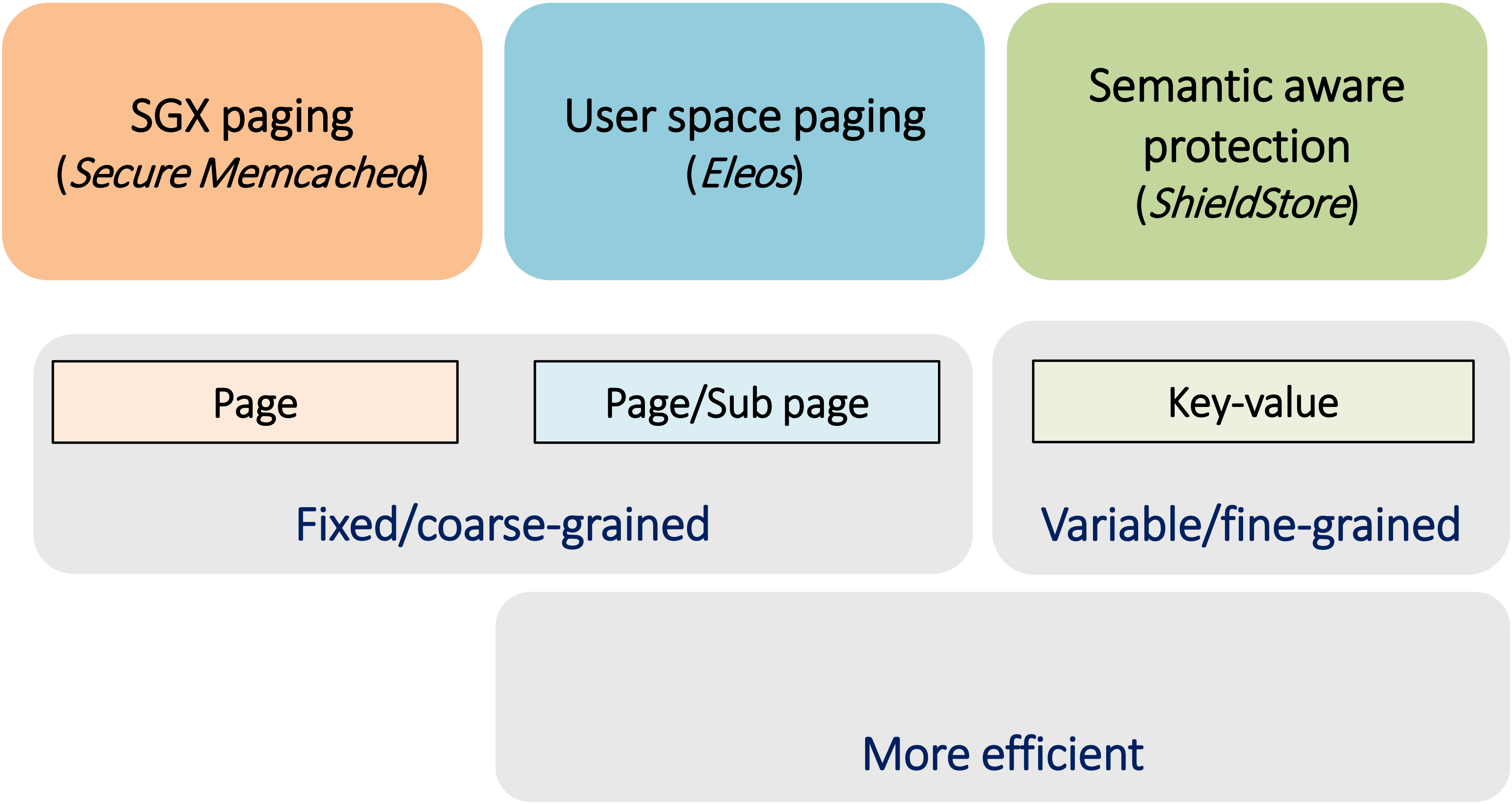
Page/Sub page

Fixed/coarse-grained

Variable/fine-grained

More efficient

Summary of Paging Principles



SGX paging
(*Secure Memcached*)

User space paging
(*Eleos*)

Semantic aware protection
(*ShieldStore*)

Page

Page/Sub page

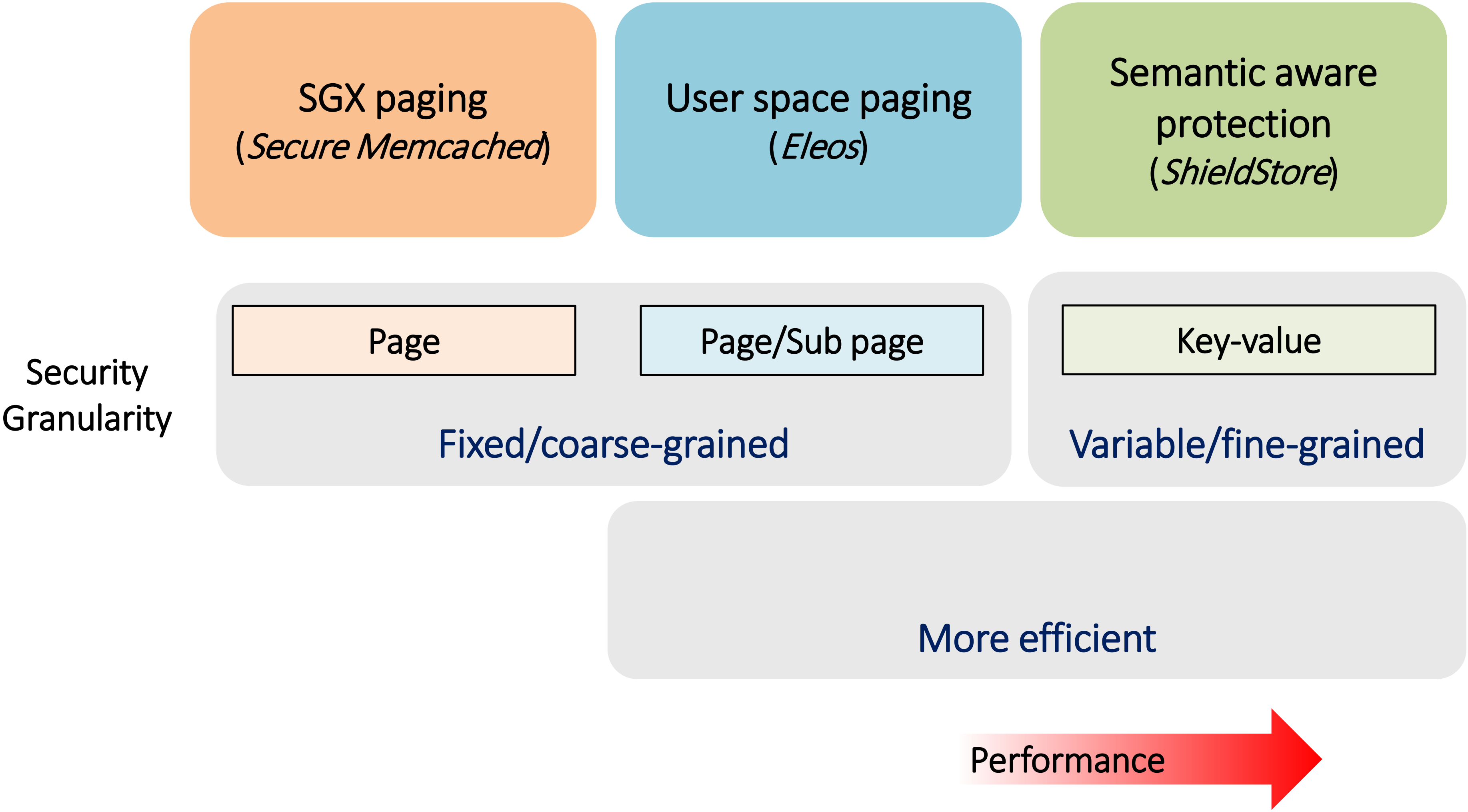
Fixed/coarse-grained

Key-value

Variable/fine-grained

More efficient

Summary of Paging Principles



SGX paging (*Secure Memcached*)

User space paging (*Eleos*)

Semantic aware protection (*ShieldStore*)

Security Granularity

Page

Page/Sub page

Fixed/coarse-grained

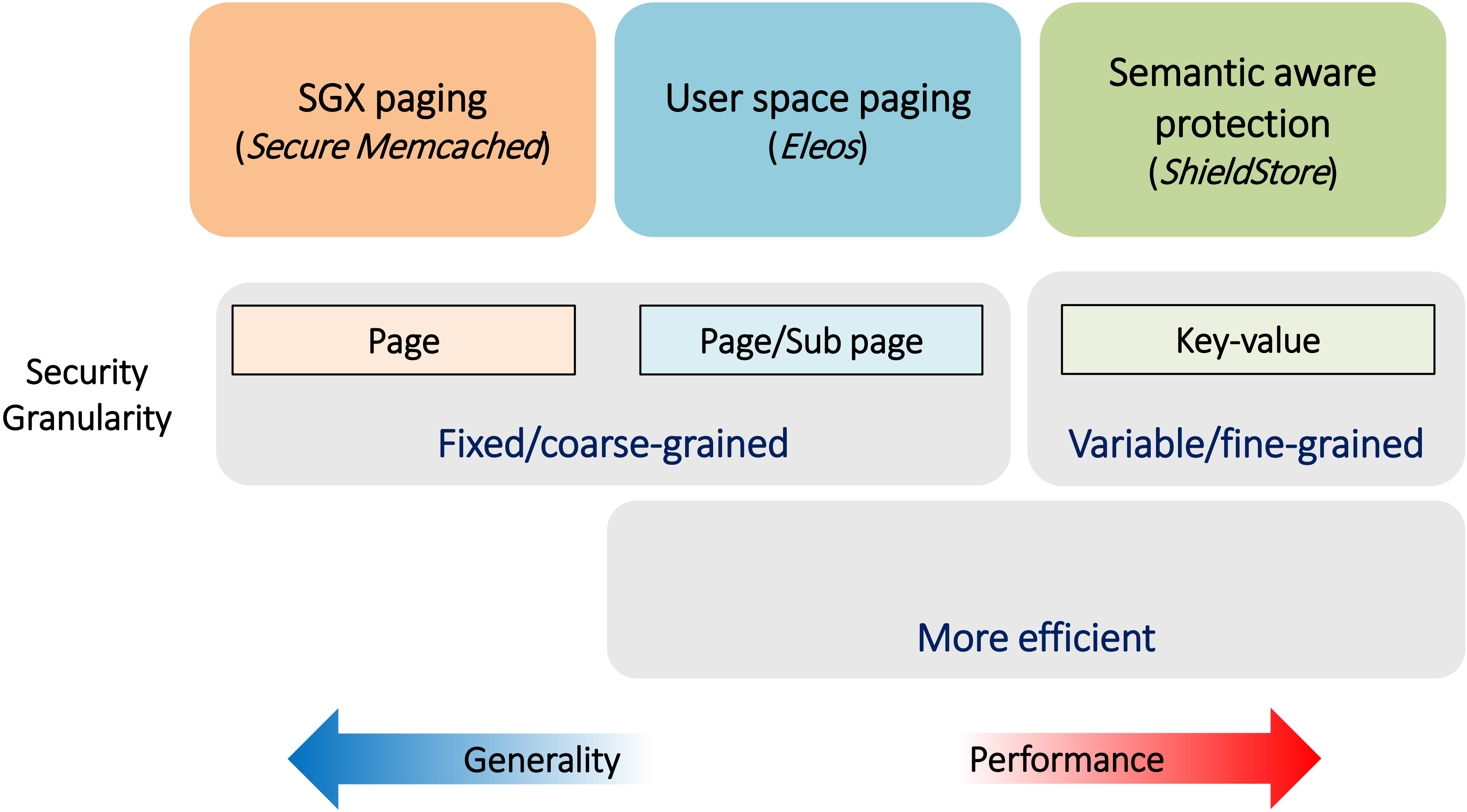
Key-value

Variable/fine-grained

More efficient

Performance

Summary of Paging Principles



SGX paging (*Secure Memcached*)

User space paging (*Eleos*)

Semantic aware protection (*ShieldStore*)

Security Granularity

Page

Page/Sub page

Fixed/coarse-grained

Key-value

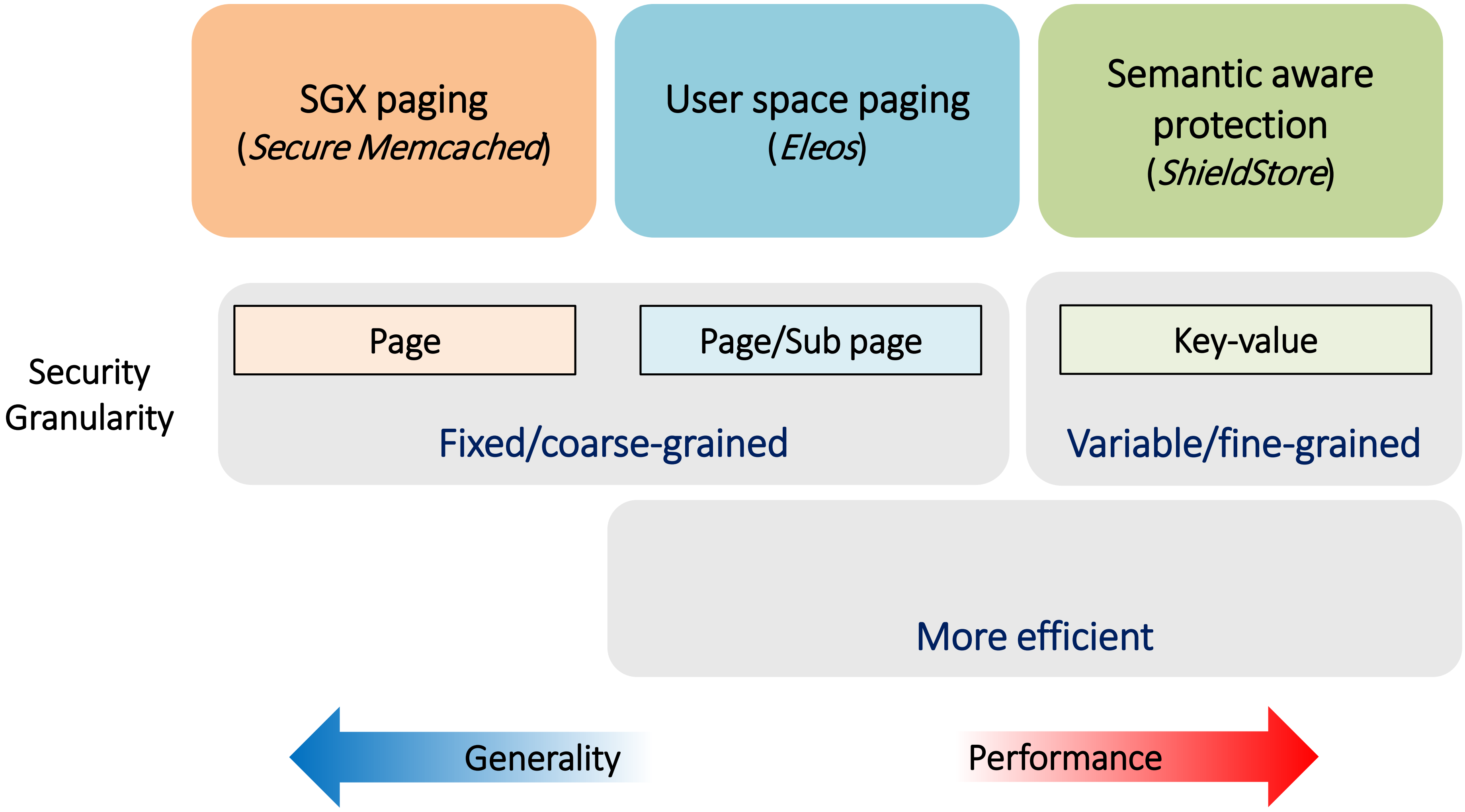
Variable/fine-grained

More efficient

Generality

Performance

Summary of Paging Principles



SGX paging (*Secure Memcached*)

User space paging (*Eleos*)

Semantic aware protection (*ShieldStore*)

Security Granularity

Page

Page/Sub page

Fixed/coarse-grained

Key-value

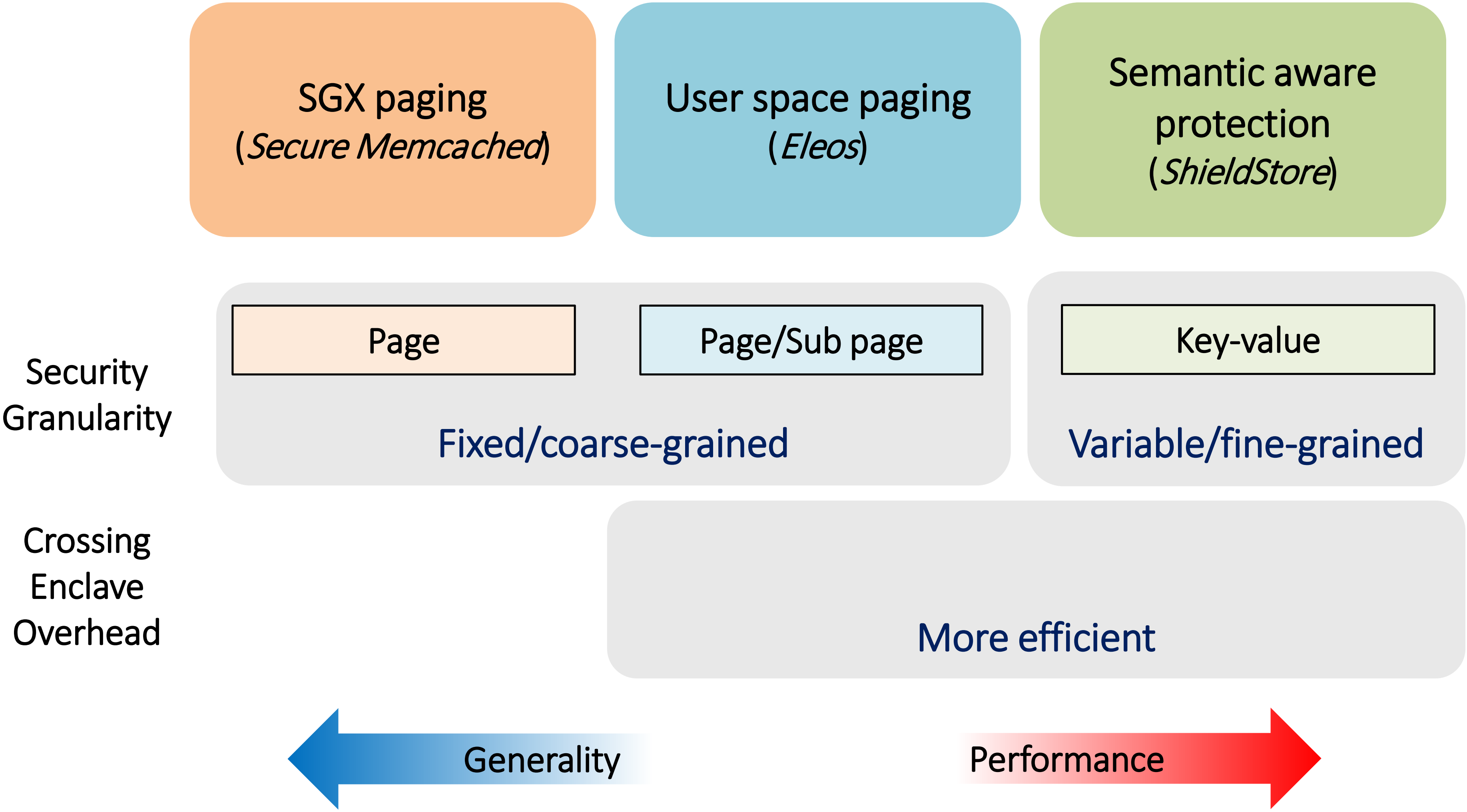
Variable/fine-grained

More efficient

Generality

Performance

Summary of Paging Principles



SGX paging (*Secure Memcached*)

User space paging (*Eleos*)

Semantic aware protection (*ShieldStore*)

Security Granularity

Page

Page/Sub page

Key-value

Fixed/coarse-grained

Variable/fine-grained

Crossing Enclave Overhead

More efficient

Generality

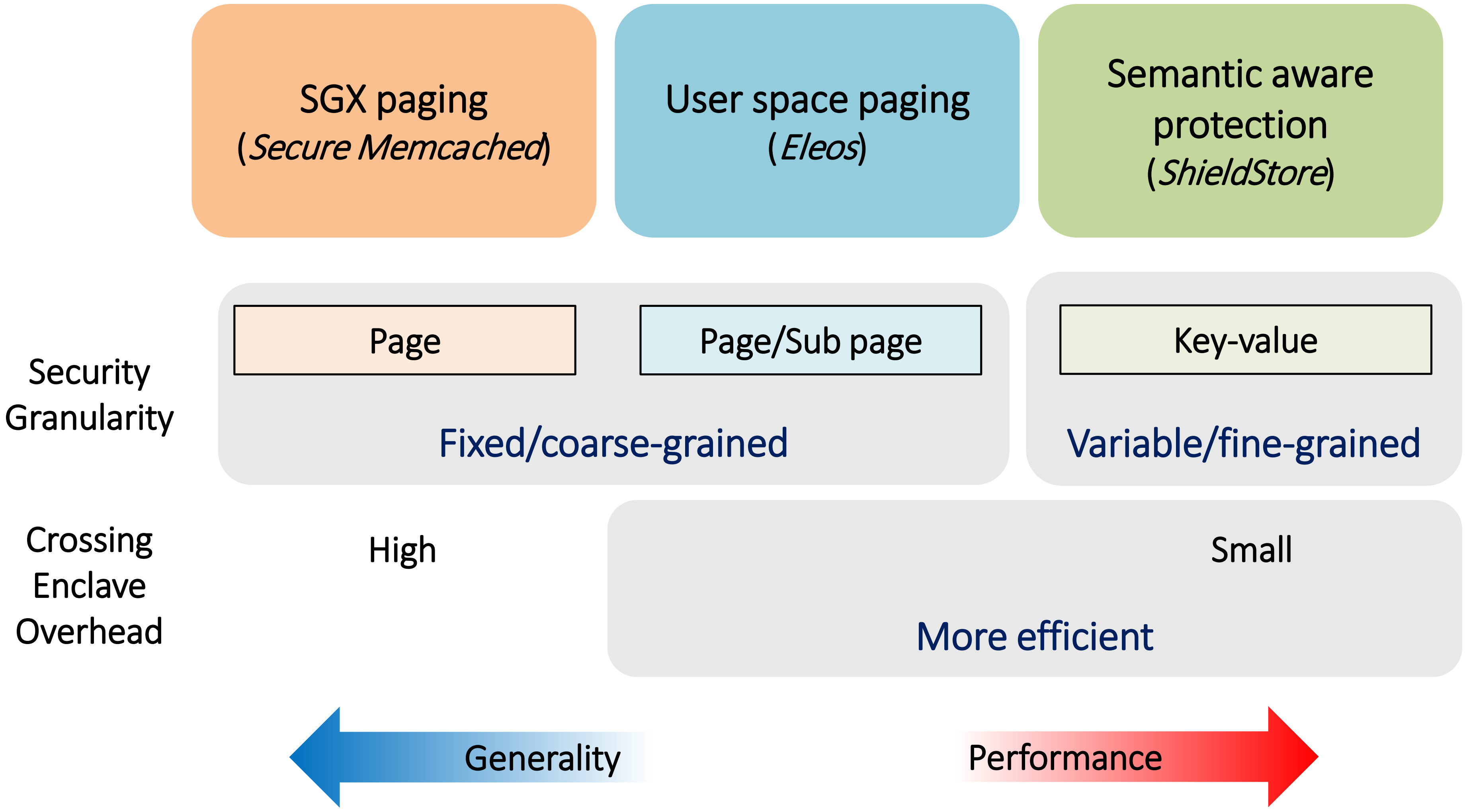
Performance

Summary of Paging Principles

	SGX paging (*Secure Memcached*)	User space paging (*Eleos*)	Semantic aware protection (*ShieldStore*)
Security Granularity	Page	Page/Sub page	Key-value
	Fixed/coarse-grained		Variable/fine-grained
Crossing Enclave Overhead	High	More efficient	

Generality
Performance

Summary of Paging Principles



SGX paging (*Secure Memcached*)

User space paging (*Eleos*)

Semantic aware protection (*ShieldStore*)

Security Granularity

Page

Page/Sub page

Fixed/coarse-grained

Key-value

Variable/fine-grained

Crossing Enclave Overhead

High

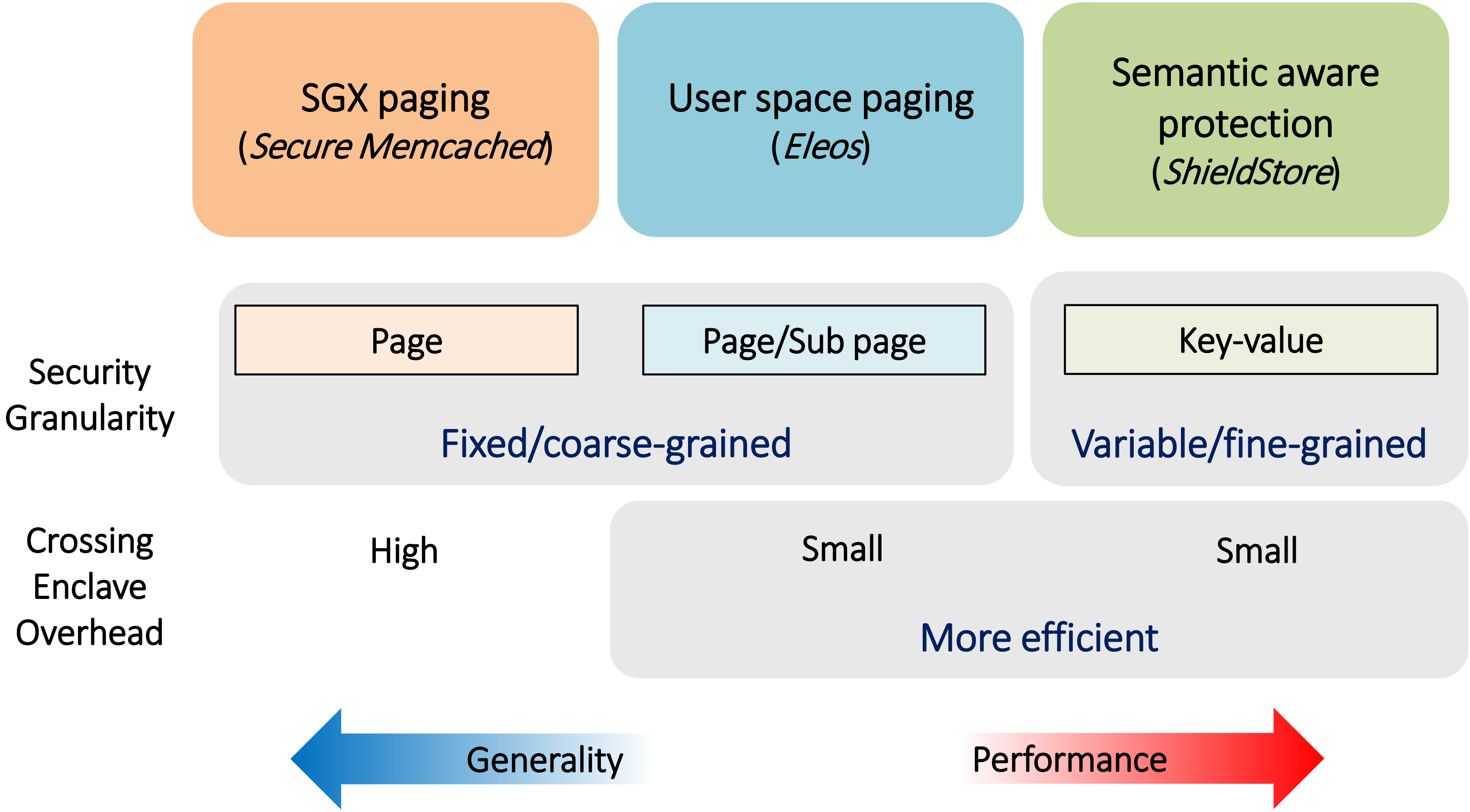
Small

More efficient

Generality

Performance

Summary of Paging Principles



	SGX paging (*Secure Memcached*)	User space paging (*Eleos*)	Semantic aware protection (*ShieldStore*)
Security Granularity	Page Fixed/coarse-grained	Page/Sub page	Key-value Variable/fine-grained
Crossing Enclave Overhead	High	Small More efficient	Small

Summary of Paging Principles

	SGX paging (*Secure Memcached*)	User space paging (*Eleos*)	Semantic aware protection (*ShieldStore*)
Security Granularity	Page Fixed/coarse-grained	Page/Sub page	Key-value Variable/fine-grained
Crossing Enclave Overhead	High	Small More efficient	Small

Generality
Performance

Summary of Paging Principles

	SGX paging (*Secure Memcached*)	User space paging (*Eleos*)	Semantic aware protection (*ShieldStore*)
Security Granularity	Page	Page/Sub page	Key-value
	Fixed/coarse-grained		Variable/fine-grained
Crossing Enclave Overhead	High	Small	Small
		More efficient	

Generality

Performance

Summary of Paging Principles

	SGX paging (*Secure Memcached*)	User space paging (*Eleos*)	Semantic aware protection (*ShieldStore*)
Security Granularity	Page	Page/Sub page	Key-value
	Fixed/coarse-grained		Variable/fine-grained
Crossing Enclave Overhead	High	Small	Small
		More efficient	

← Generality

Performance →

Summary of Paging Principles

	SGX paging (*Secure Memcached*)	User space paging (*Eleos*)	Semantic aware protection (*ShieldStore*)
Security Granularity	Page	Page/Sub page	Key-value
	Fixed/coarse-grained		Variable/fine-grained
Crossing Enclave Overhead	High	Small	Small
		More efficient	

ShieldStore code will be available at https://github.com/cocoppang/ShieldStore

Computer Architecture Lab.

ShieldStore: Shielded In-memory Key-value Storage with SGX

Taehoon Kim, Joongun Park, Jaewook Woo,
Seungheun Jeon, and Jaehyuk Huh

Dresden, Germany
March 25-27, 2019