Anchor

A Library for Building Secure Persistent Memory Systems

Dimitrios Stavrakakis, Dimitra Giantsidi, Maurice Bailleu,
Philip Sändig, Shady Issa, Pramod Bhatotia

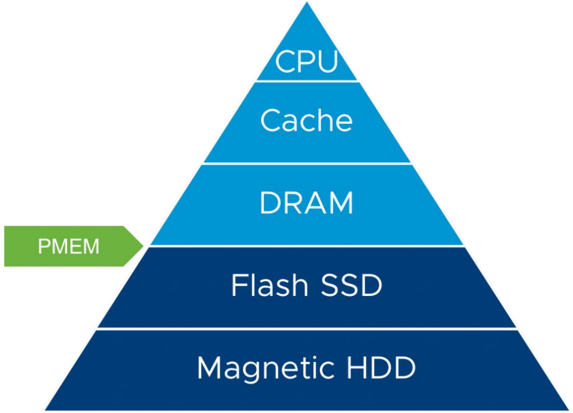
Technische Universität München TUM

THE UNIVERSITY of EDINBURGH

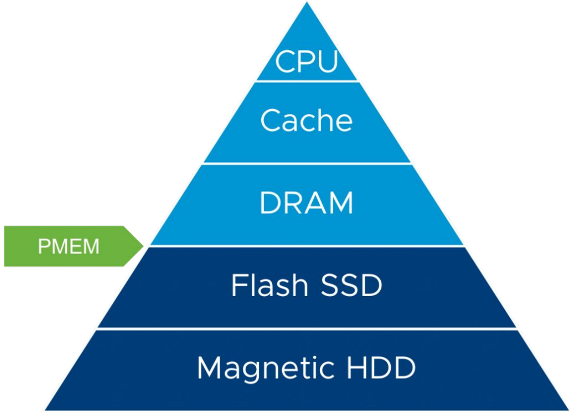
ACM SIGMOD 2024

Persistent memory in the cloud

Persistent memory in the cloud



Persistent memory in the cloud

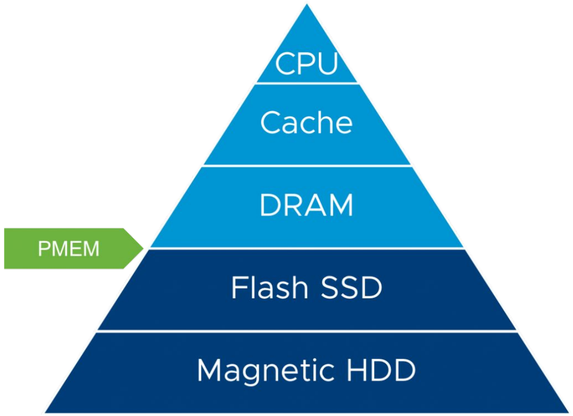


RocksDB

redis

pmemkv

Persistent memory in the cloud



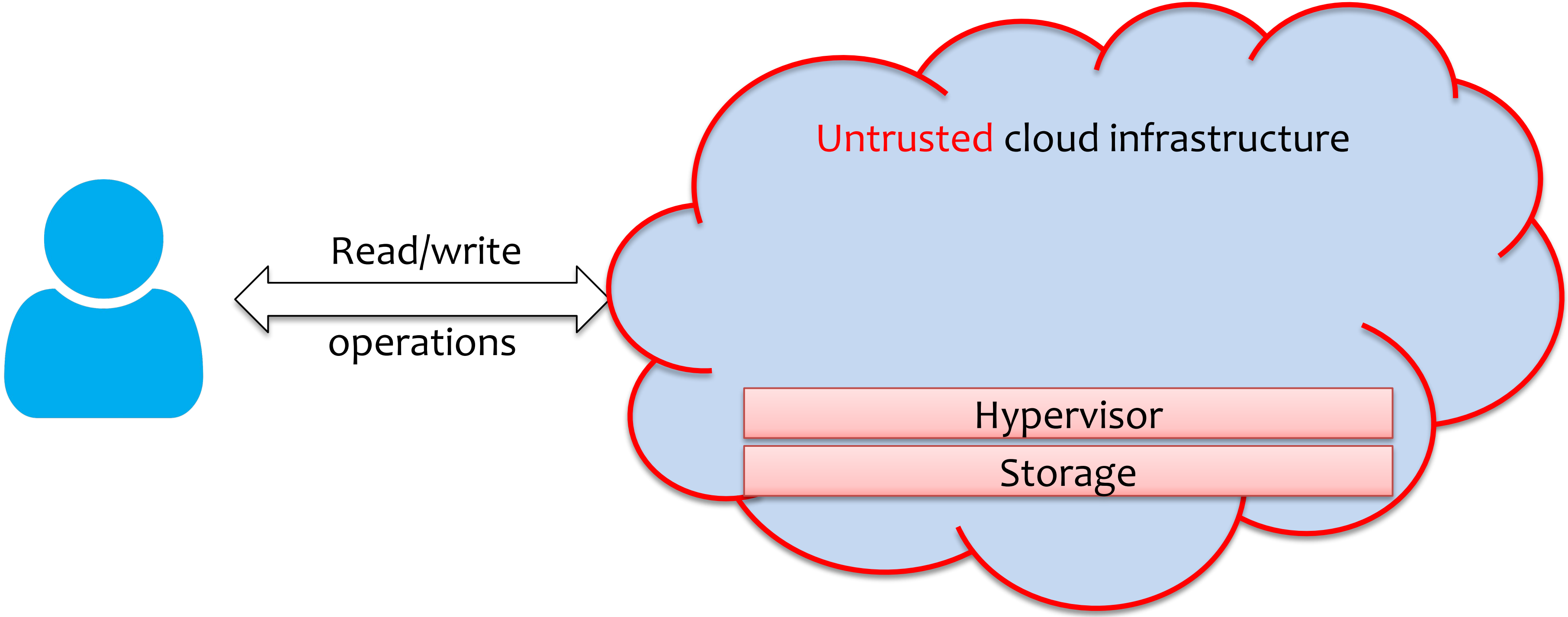
RocksDB

redis

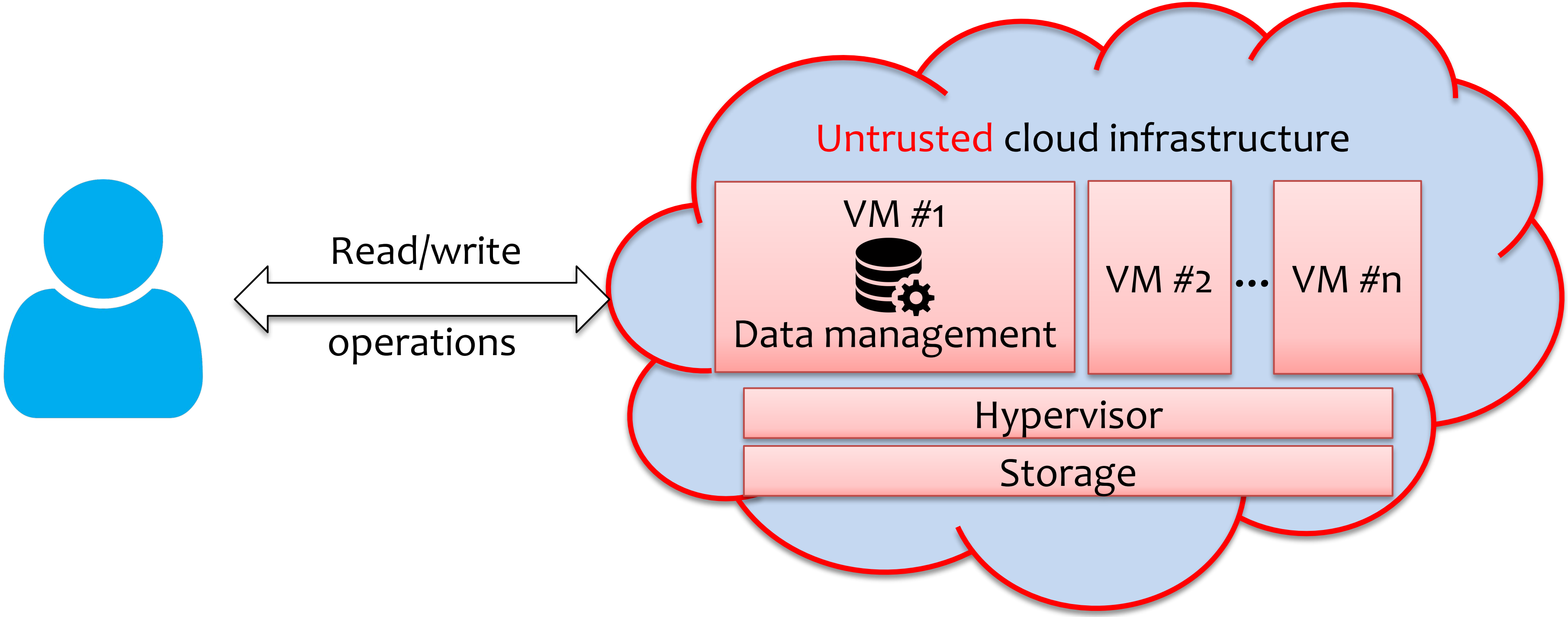
pmemkv

Persistent memory can benefit the offered cloud providers’ services

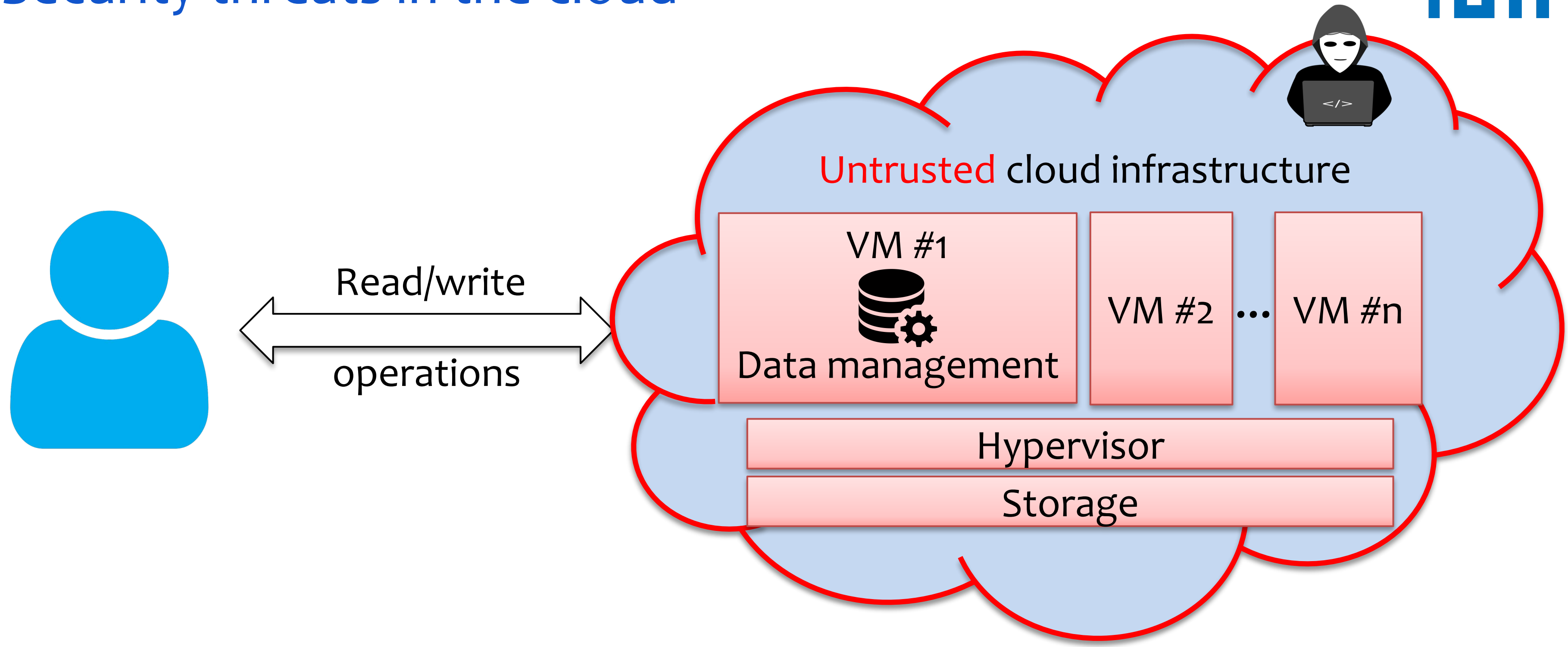
Security threats in the cloud



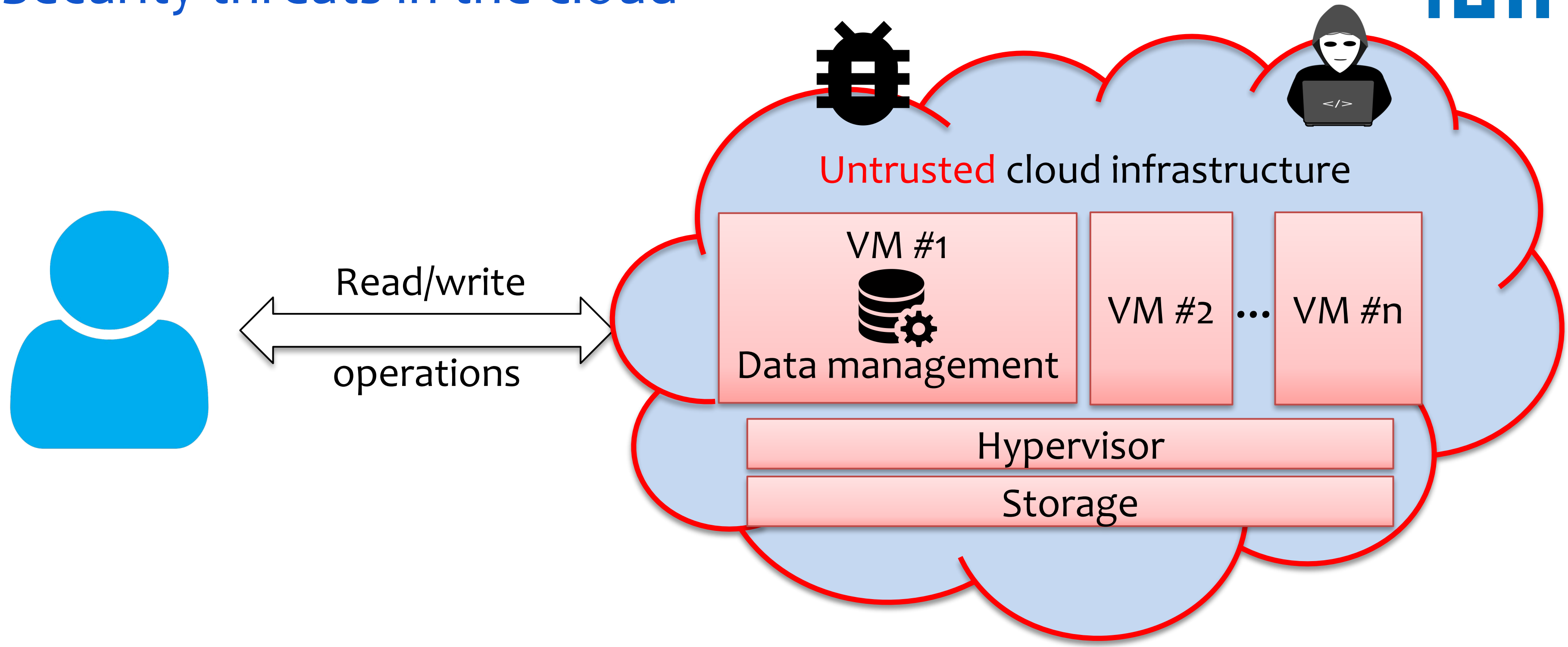
Security threats in the cloud



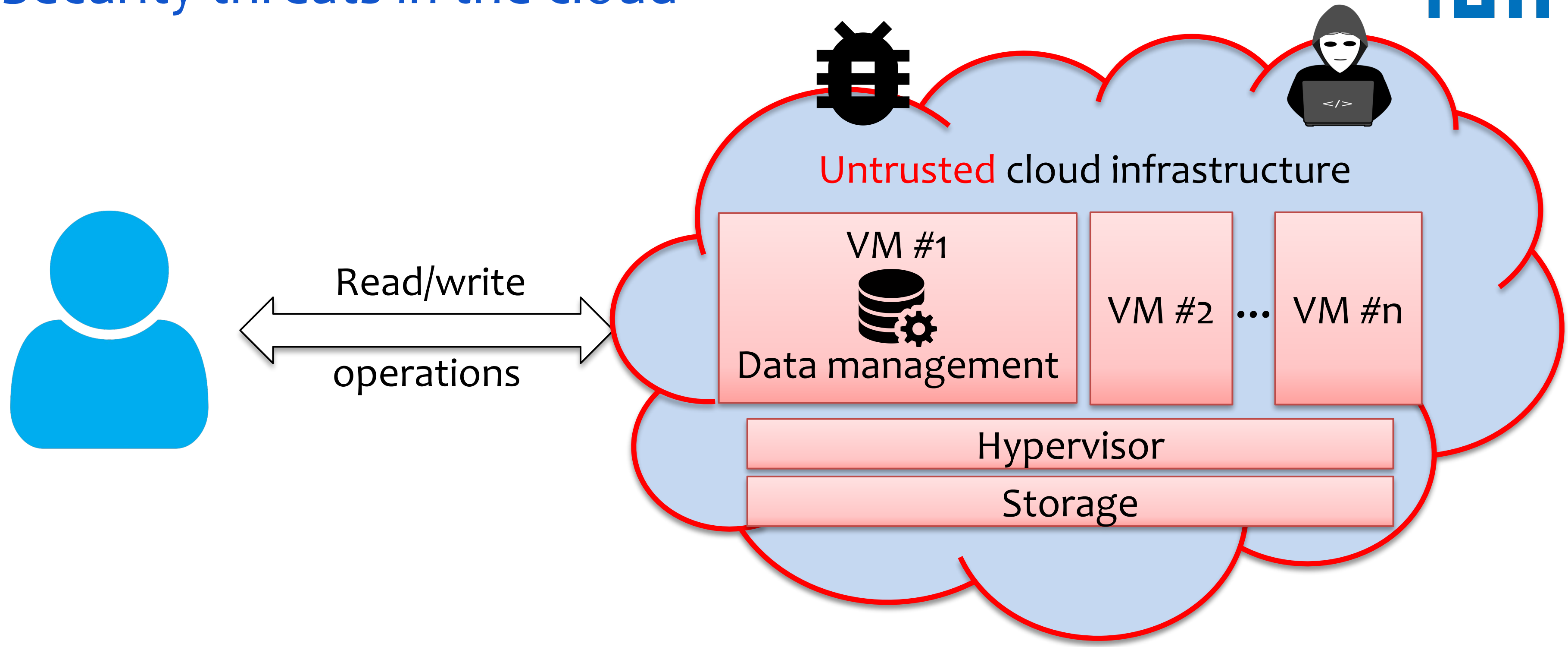
Security threats in the cloud



Security threats in the cloud

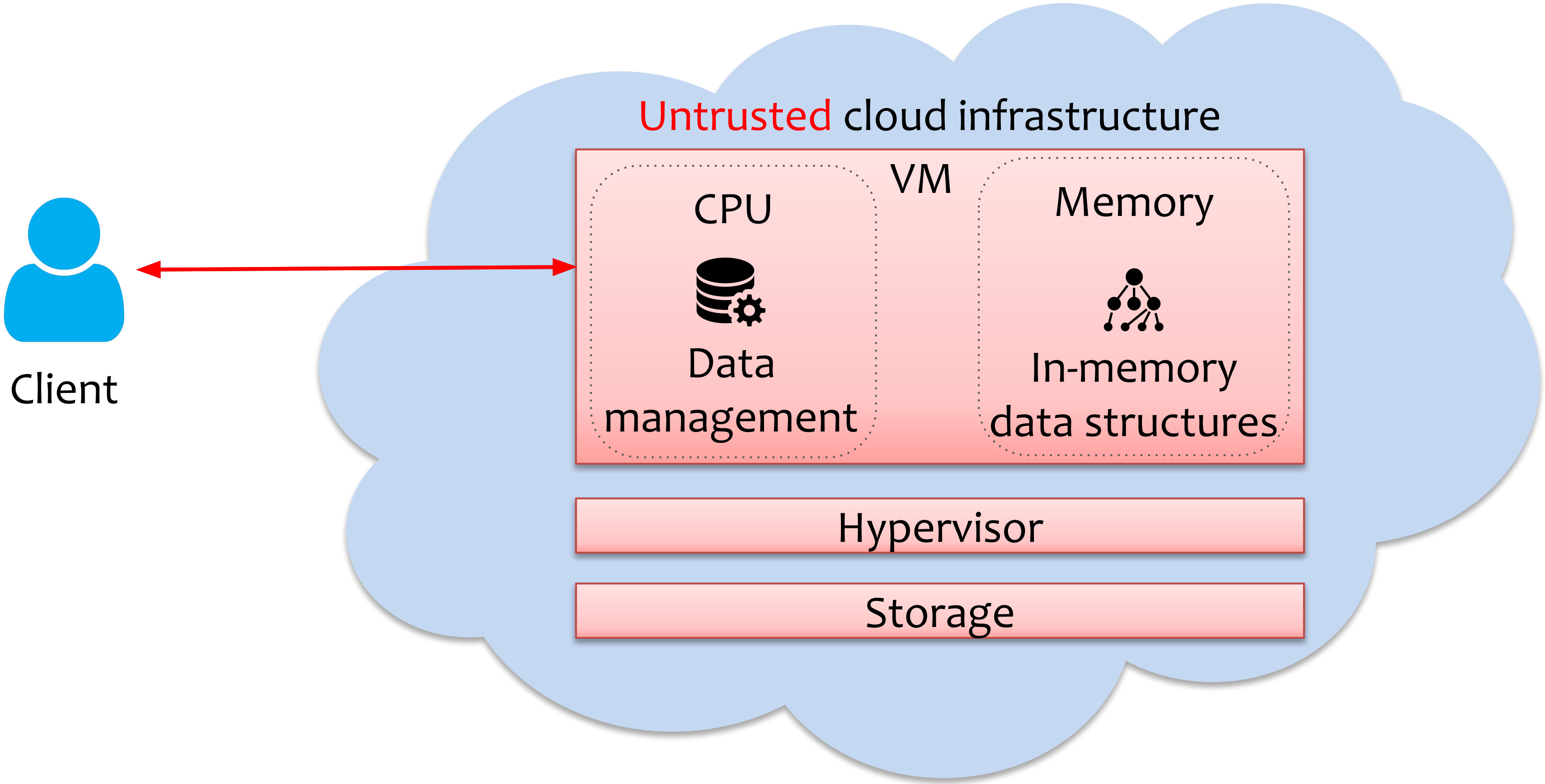


Security threats in the cloud

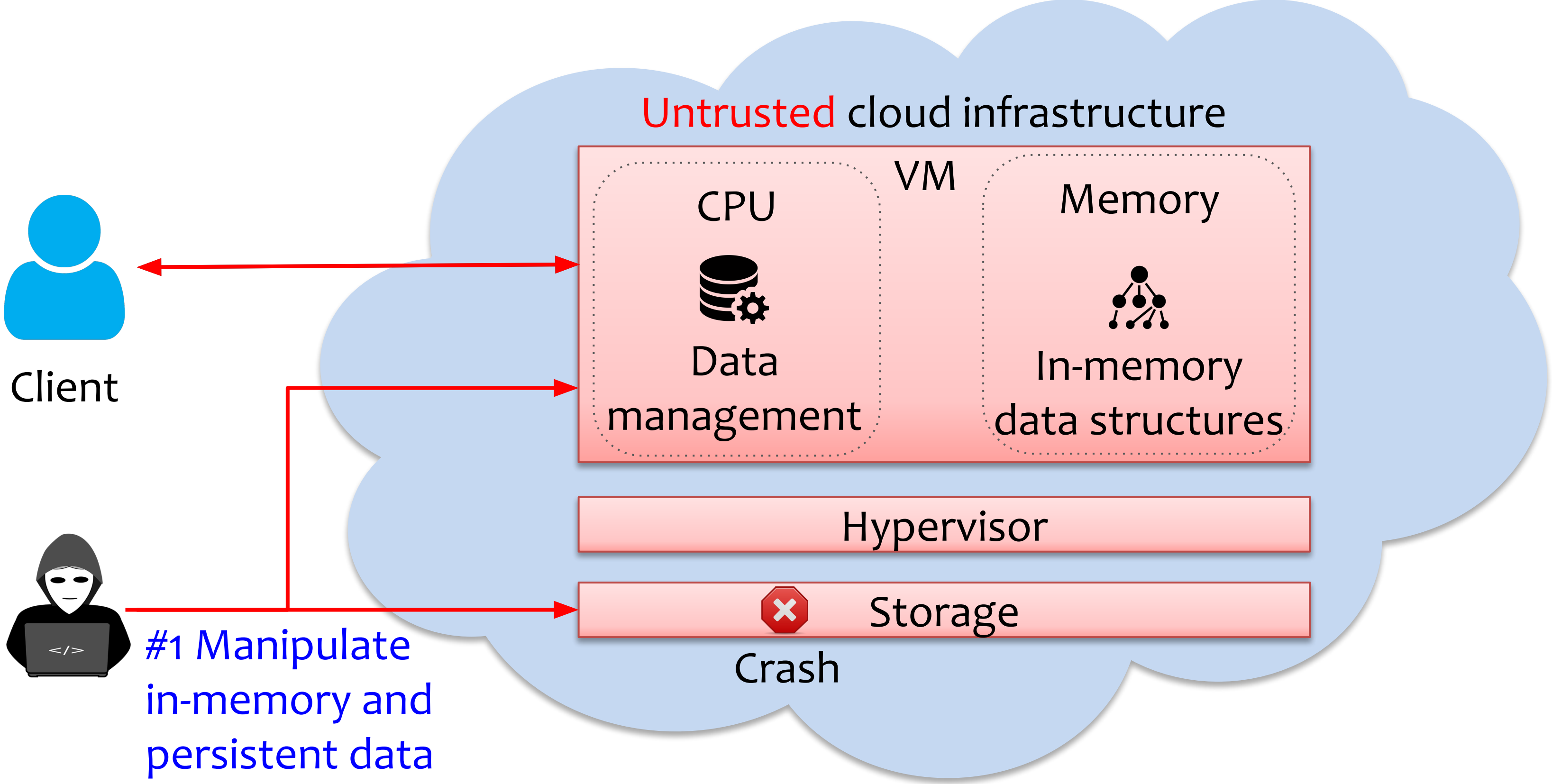


How can we protect client's data in untrusted cloud infrastructures?

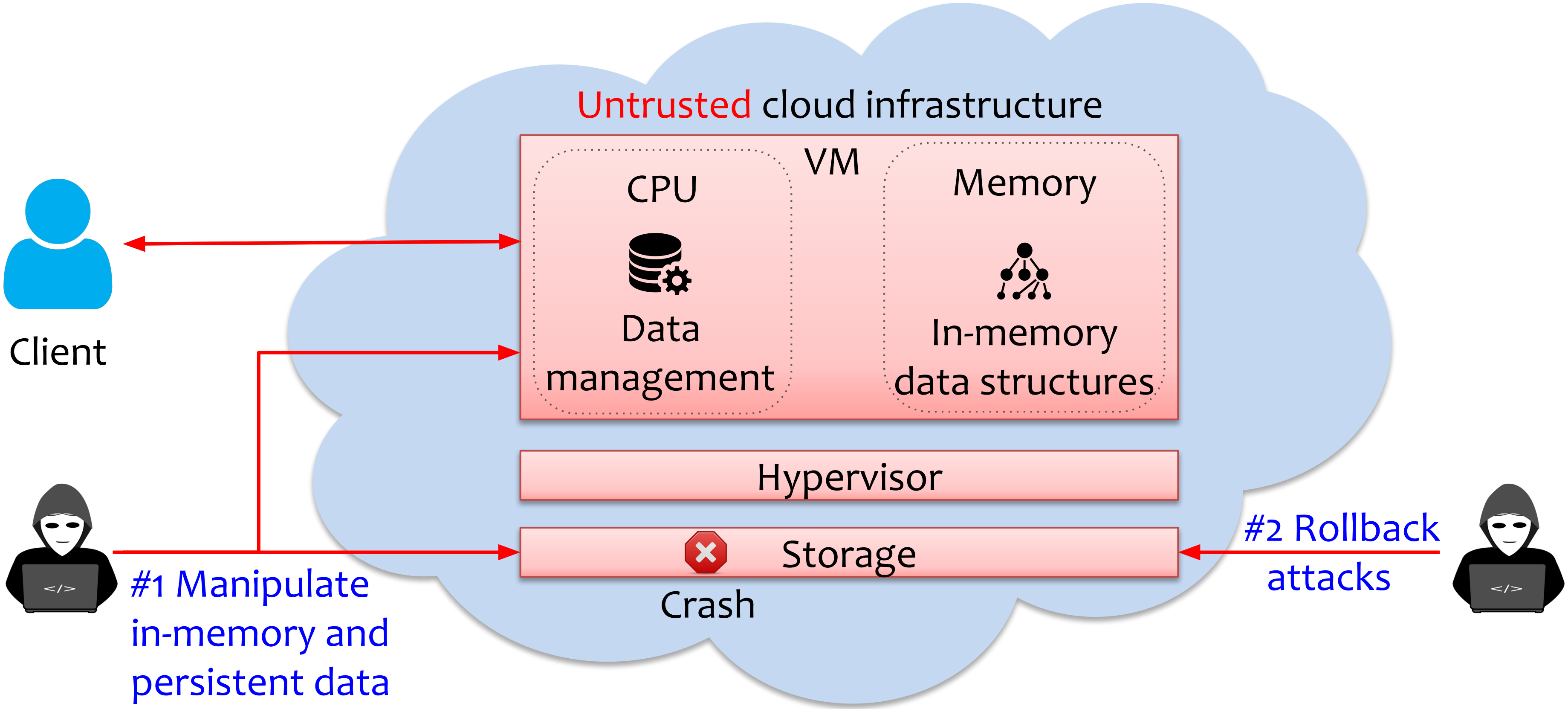
Security threats in the cloud



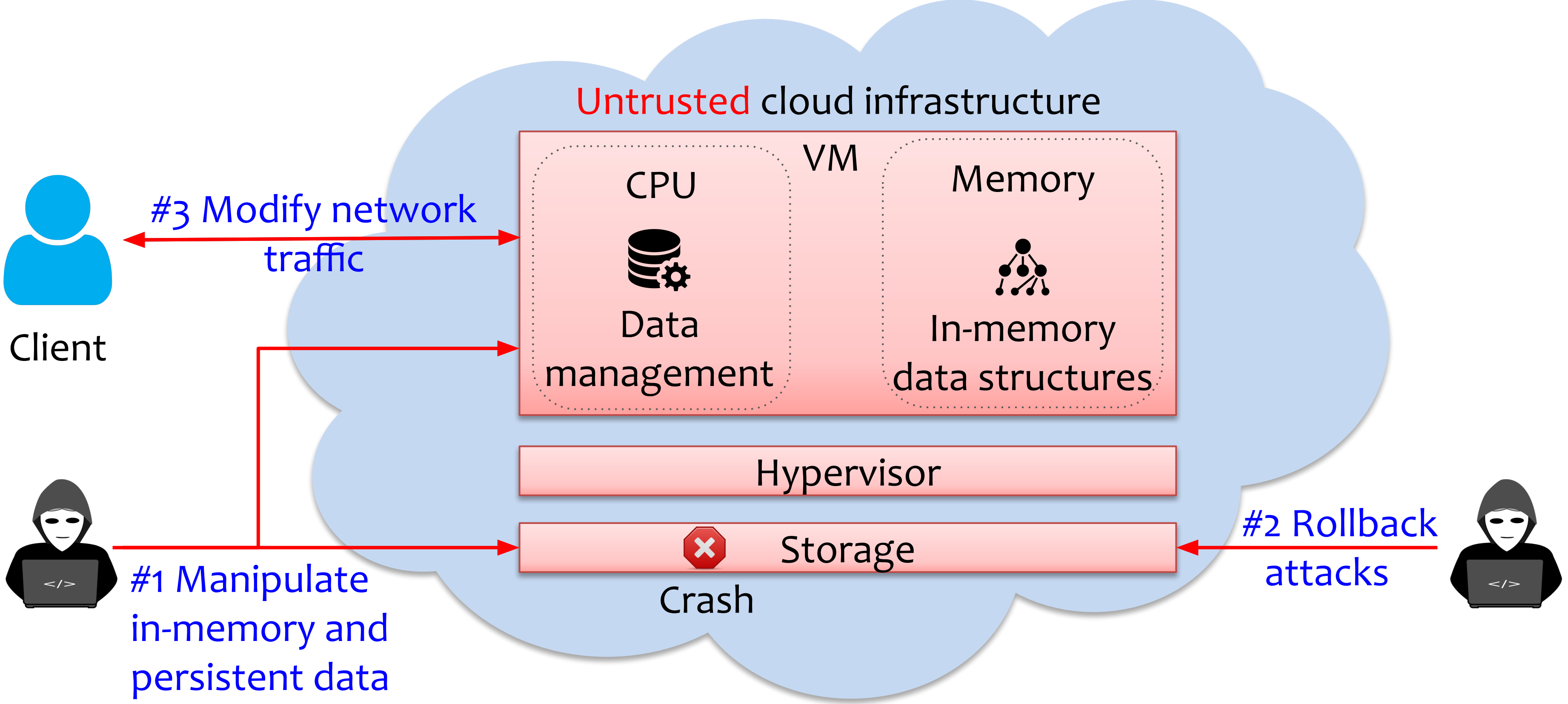
Security threats in the cloud


Untrusted cloud infrastructure
VM
CPU
Data management
Memory
In-memory data structures
Hypervisor
Storage
Crash
Client
#1 Manipulate in-memory and persistent data

Security threats in the cloud


Untrusted cloud infrastructure
VM
CPU
Data management
Memory
In-memory data structures
Hypervisor
Storage
Crash
Client
#1 Manipulate in-memory and persistent data
#2 Rollback attacks
</>

Security threats in the cloud


Untrusted cloud infrastructure
VM
CPU
Data management
Memory
In-memory data structures
Hypervisor
Storage
Crash
Client
#3 Modify network traffic
#1 Manipulate in-memory and persistent data
#2 Rollback attacks
</>
</>

Problem statement

How to design a **secure PM management system** for untrusted cloud environments?

Our proposal

Anchor: A Library for Building Secure Persistent Memory Systems

System properties:

- **End-to-end security:** Confidentiality, integrity & freshness
- **Fault tolerance:** Secure crash consistency
- **Programmability:** PMDK programming model
- **Verifiability:** Formal proofs of security protocols

Our proposal

Anchor: A Library for Building Secure Persistent Memory Systems

System properties:

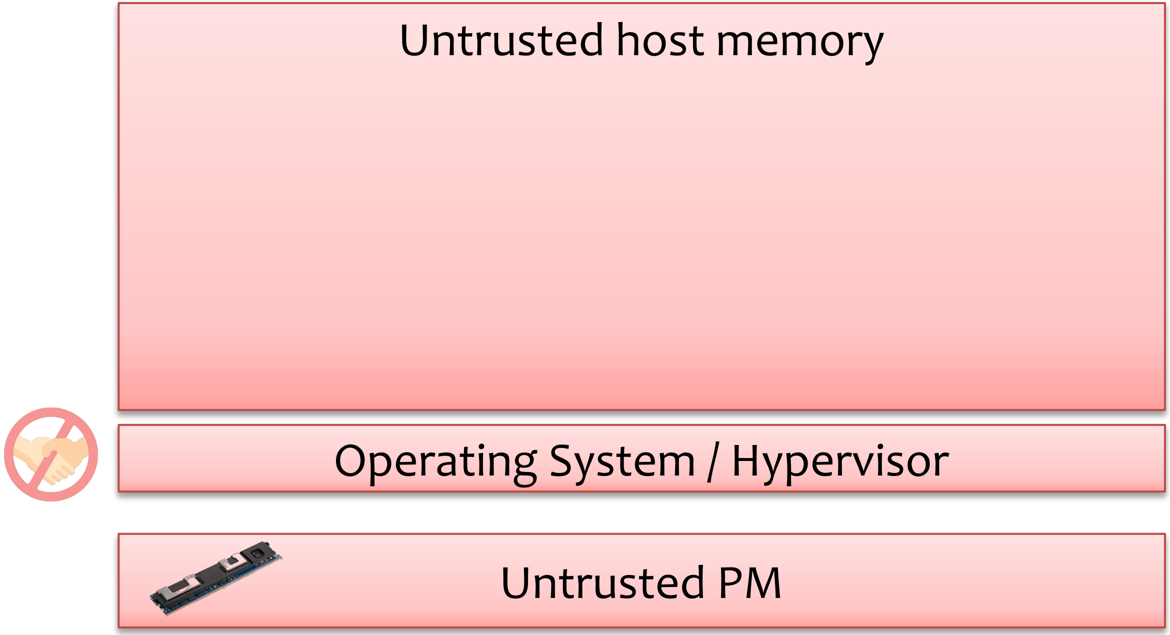
- **End-to-end security:** Confidentiality, integrity & freshness
- **Fault tolerance:** Secure crash consistency
- **Programmability:** PMDK programming model
- **Verifiability:** Formal proofs of security protocols

Performance

Outline

- ~~Introduction & Motivation~~
- System design
  - Design challenges
  - System overview
  - System operations
- Evaluation

Anchor basic design



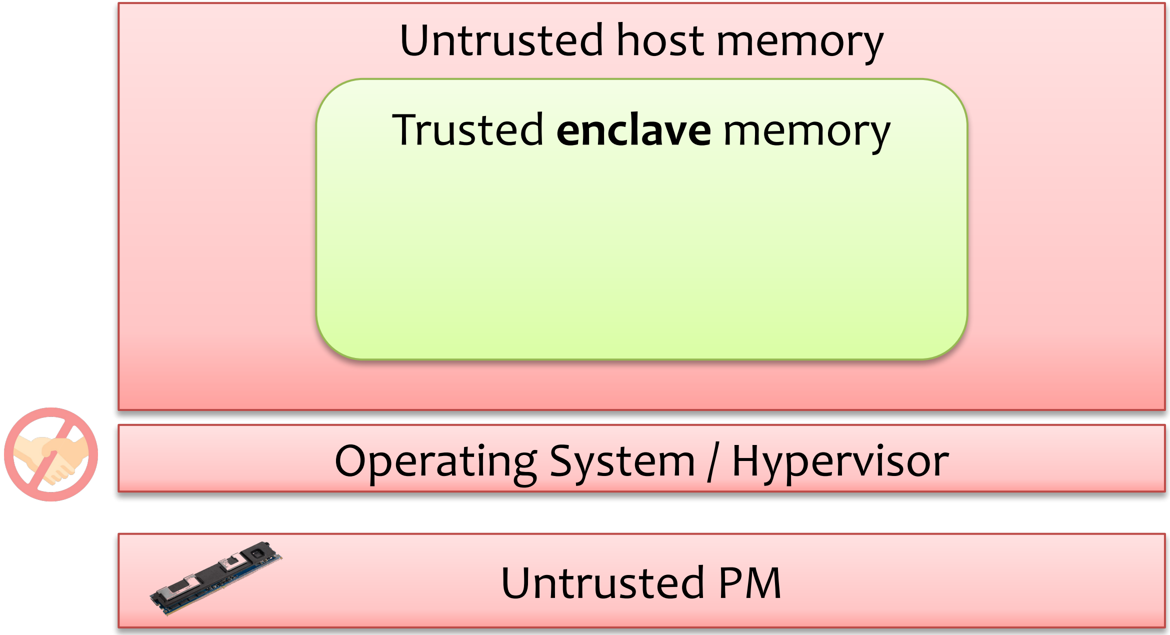
Untrusted host memory

Operating System / Hypervisor

Untrusted PM

Anchor basic design

Common insight: Why not just use modern hardware extensions that provide **TEEs**?



Untrusted host memory

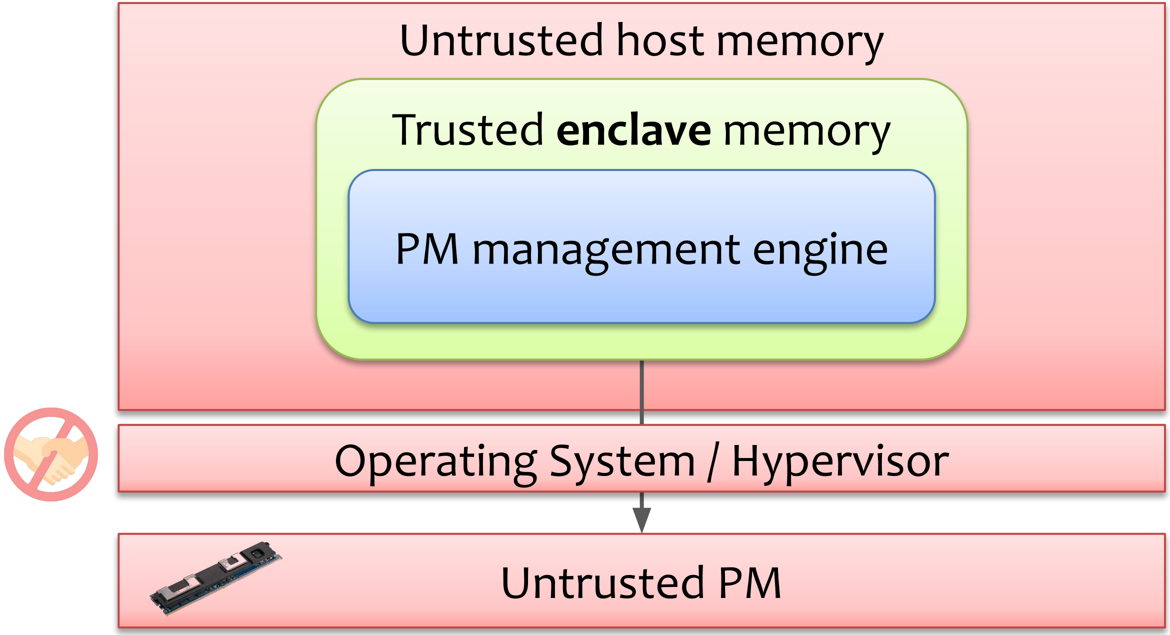
Trusted **enclave** memory

Operating System / Hypervisor

Untrusted PM

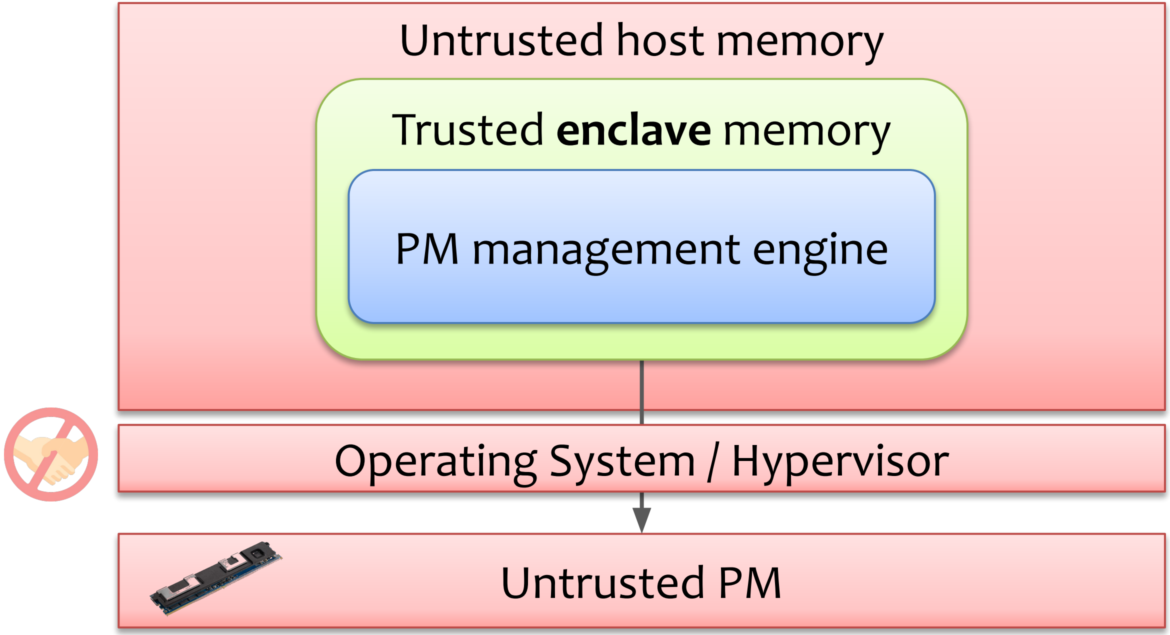
Anchor basic design

Common insight: Why not just use modern hardware extensions that provide **TEEs**?



Anchor basic design

Common insight: Why not just use modern hardware extensions that provide **TEEs**?



Unfortunately, it is not enough out-of-the-box!

Design challenges

#1

Untrusted PM &
architectural limitations of SGX

Design challenges

#1

Untrusted PM &
architectural limitations of SGX

#2

Secure crash consistency
for data & metadata

Design challenges

#1

Untrusted PM &

architectural limitations of SGX

#2

Secure crash consistency

for data & metadata

#3

Secure network communication &

attestation

Design challenges

#1

Untrusted PM &
architectural limitations of SGX

#2

Secure crash consistency
for data & metadata

#3

Secure network communication &
attestation

#4

Formal verification &
security analysis

Design challenges

#1

Untrusted PM &
architectural limitations of SGX

#2

Secure crash consistency
for data & metadata

#3

Secure network communication &
attestation

#4

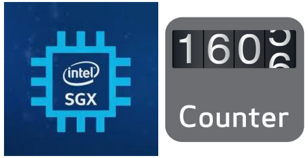
Formal verification &
security analysis

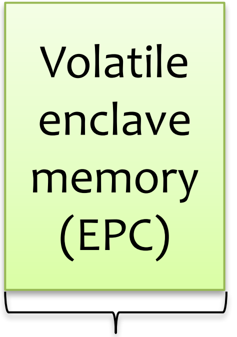
Challenge #1: Untrusted PM & architectural limitations of SGX

- TEEs protect only the volatile enclave memory

- Limited EPC size & expensive EPC paging

- Slow SGX trusted counters





Volatile enclave memory (EPC)

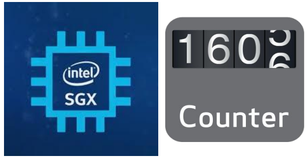
e.g., SGX v.1
~128 MiB

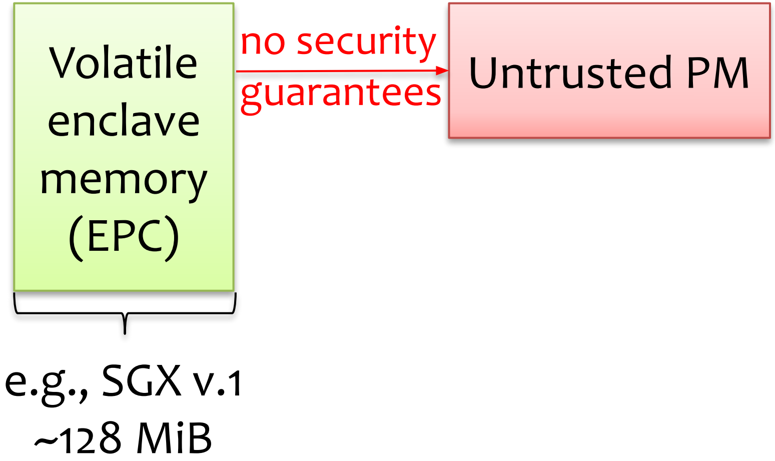
Challenge #1: Untrusted PM & architectural limitations of SGX

- TEEs protect only the volatile enclave memory

- Limited EPC size & expensive EPC paging

- Slow SGX trusted counters





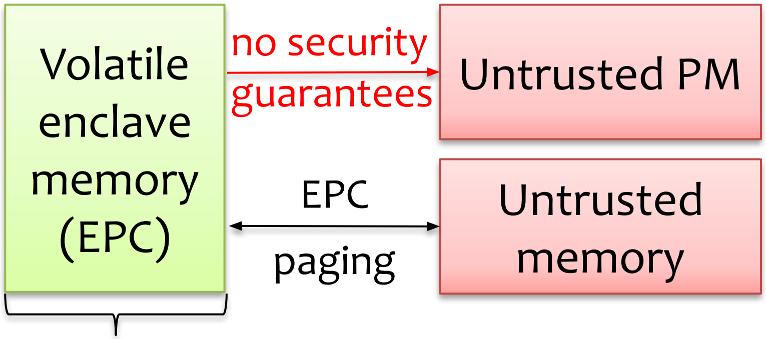
e.g., SGX v.1
~128 MiB

Challenge #1: Untrusted PM & architectural limitations of SGX

- TEEs protect only the volatile enclave memory

- Limited EPC size & expensive EPC paging

- Slow SGX trusted counters



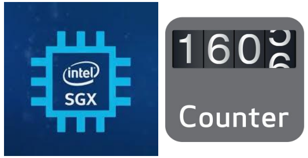
e.g., SGX v.1
~128 MiB

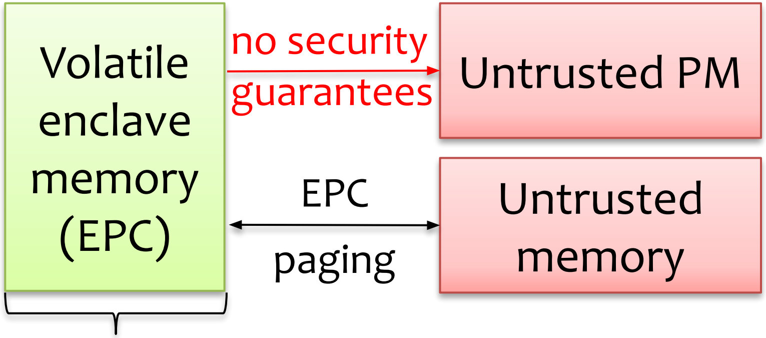
Challenge #1: Untrusted PM & architectural limitations of SGX

- TEEs protect only the volatile enclave memory

- Limited EPC size & expensive EPC paging

- Slow SGX trusted counters





e.g., SGX v.1
~128 MiB

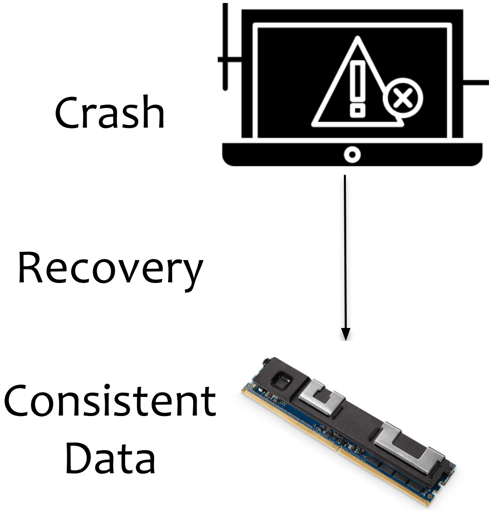
Add a PM metadata log to secure the untrusted PM, minimize EPC utilization and introduce an asynchronous trusted counter interface

Challenge #2: Secure crash consistency for data & metadata

- PM guarantees atomicity only for aligned 8-byte stores

- Transactions with insecure redo/undo logs

- Security guarantees should be valid for the logs



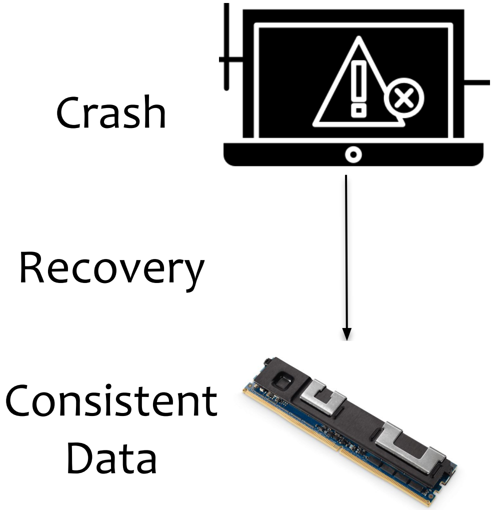
Crash

Recovery

Consistent Data

Challenge #2: Secure crash consistency for data & metadata

- PM guarantees atomicity only for aligned 8-byte stores
- Transactions with insecure redo/undo logs
- Security guarantees should be valid for the logs



Crash

Recovery

Consistent Data

Enhance the log structure with security metadata to ensure secure logging and introduce a secure recovery protocol

Challenge #3: Secure network communication & attestation

- Network buffers cannot be placed inside the enclave memory

- Ensure the security properties & crash consistency for remote operations

- The clients must be able to verify the authenticity of the running instance

Challenge #3: Secure network communication & attestation

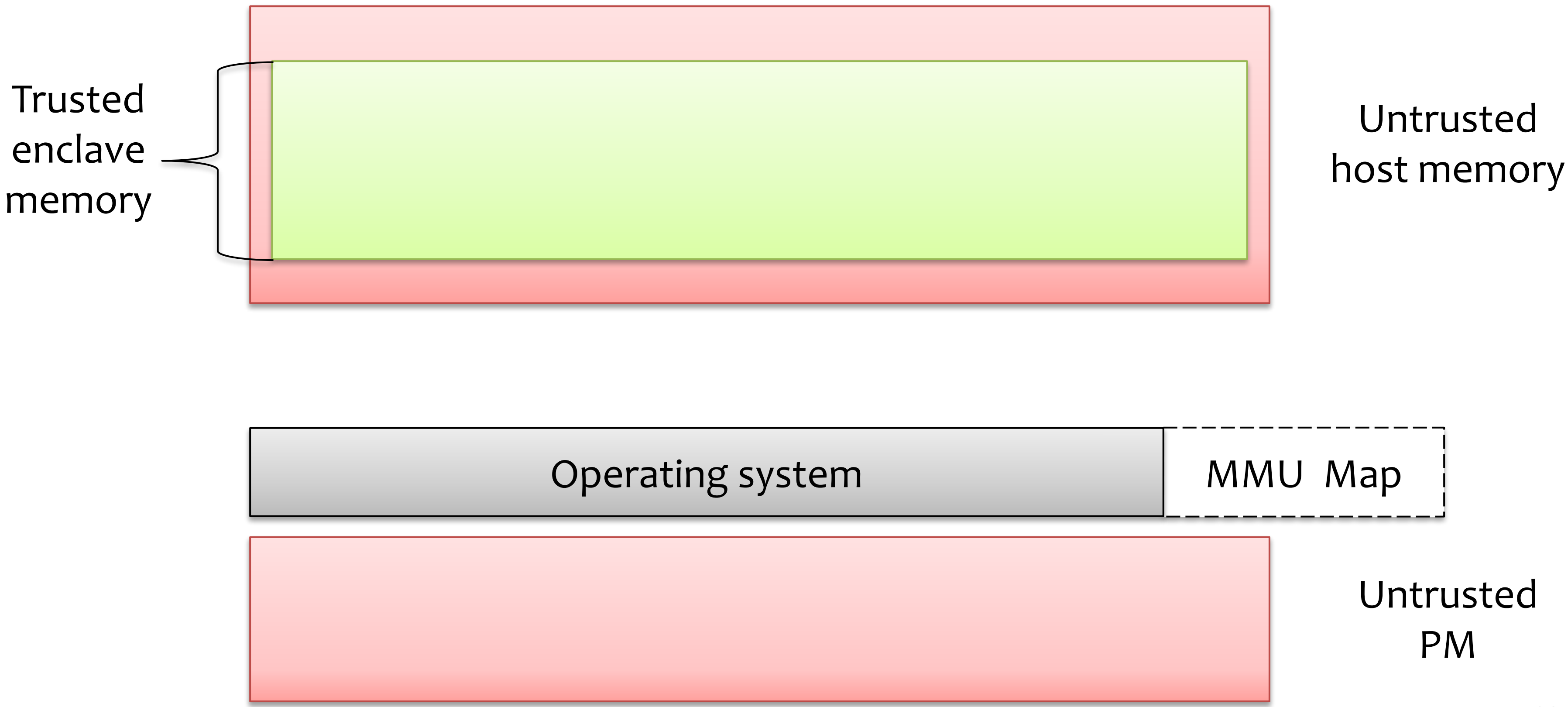
- Network buffers cannot be placed inside the enclave memory

- Ensure the security properties & crash consistency for remote operations

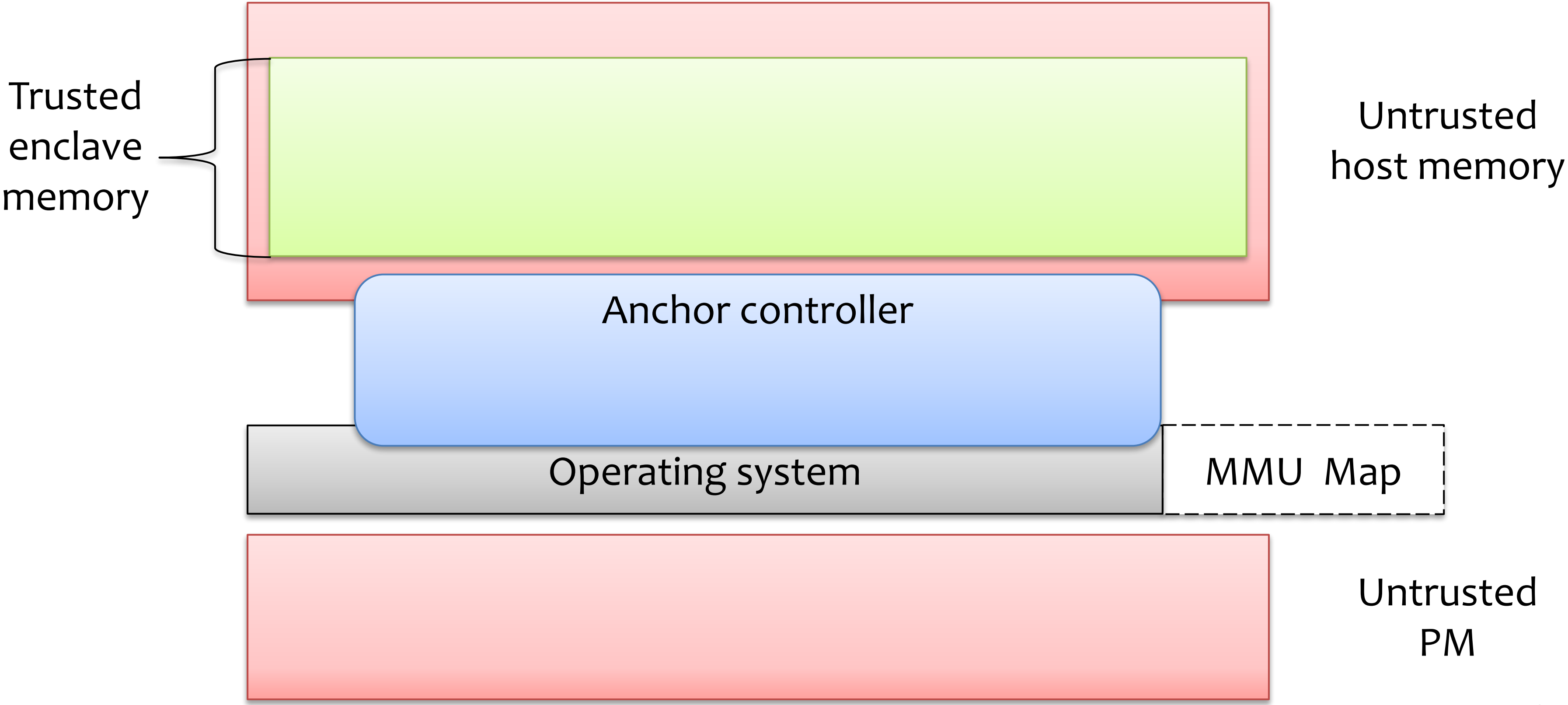
- The clients must be able to verify the authenticity of the running instance

Design a secure network stack and introduce a secure remote attestation protocol

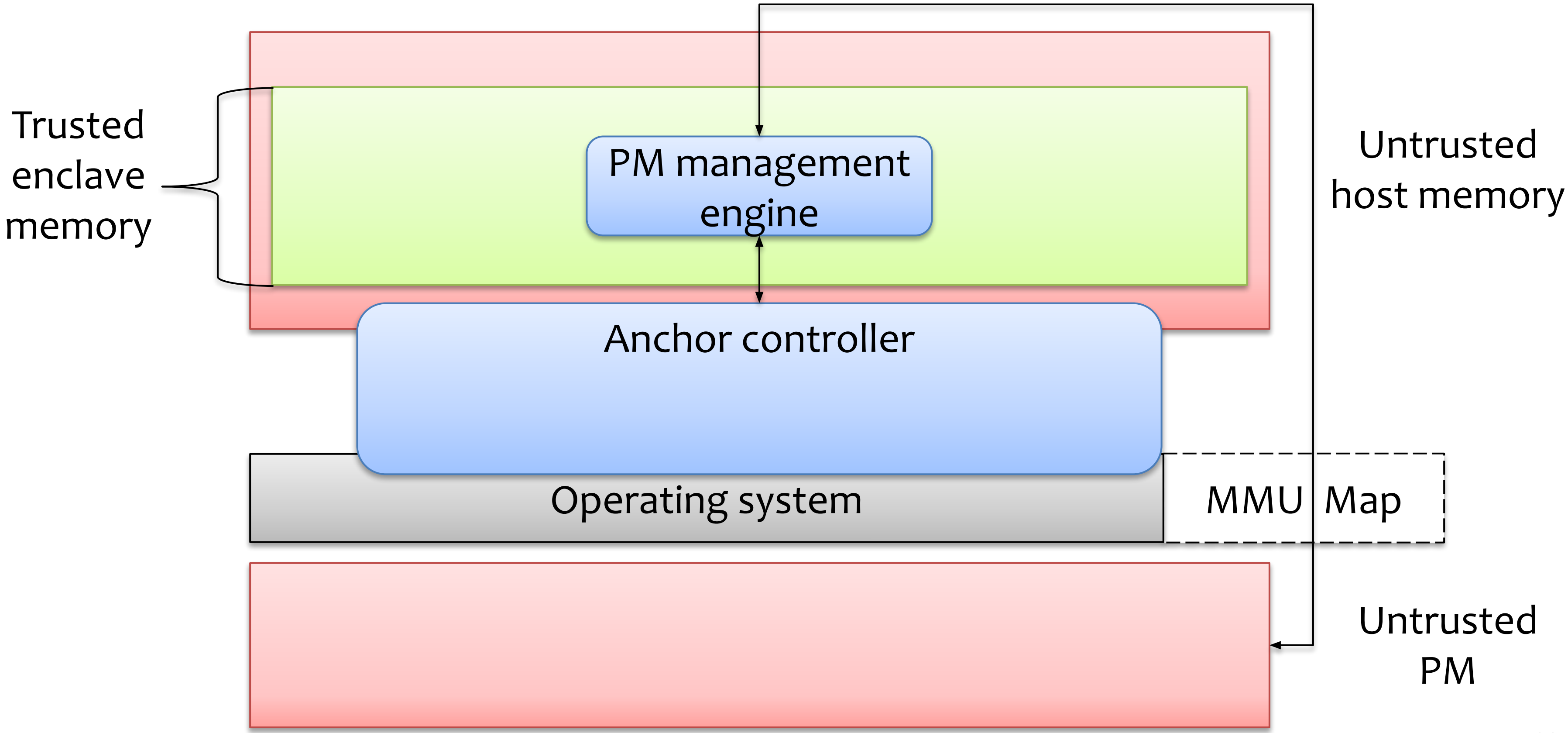
System overview



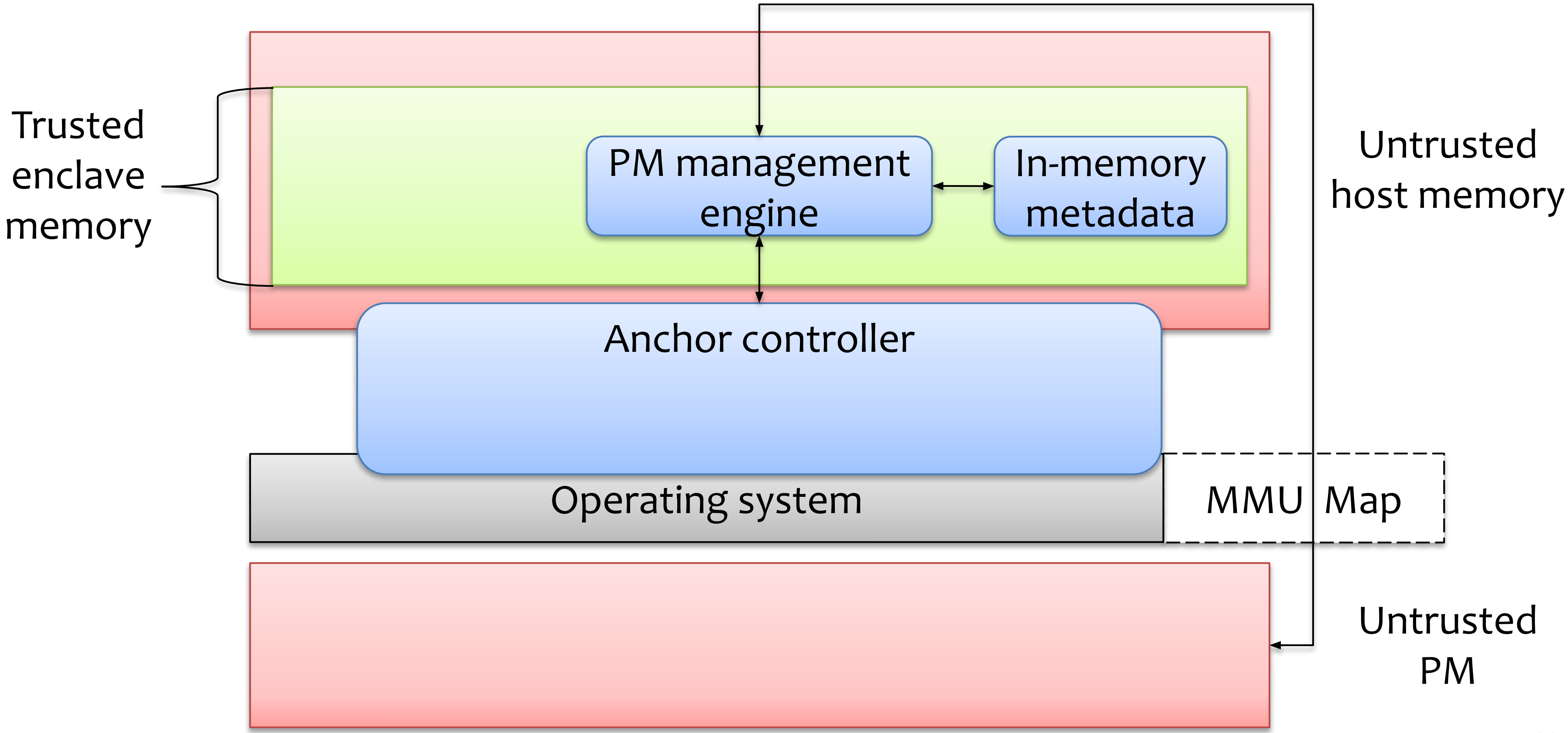
System overview



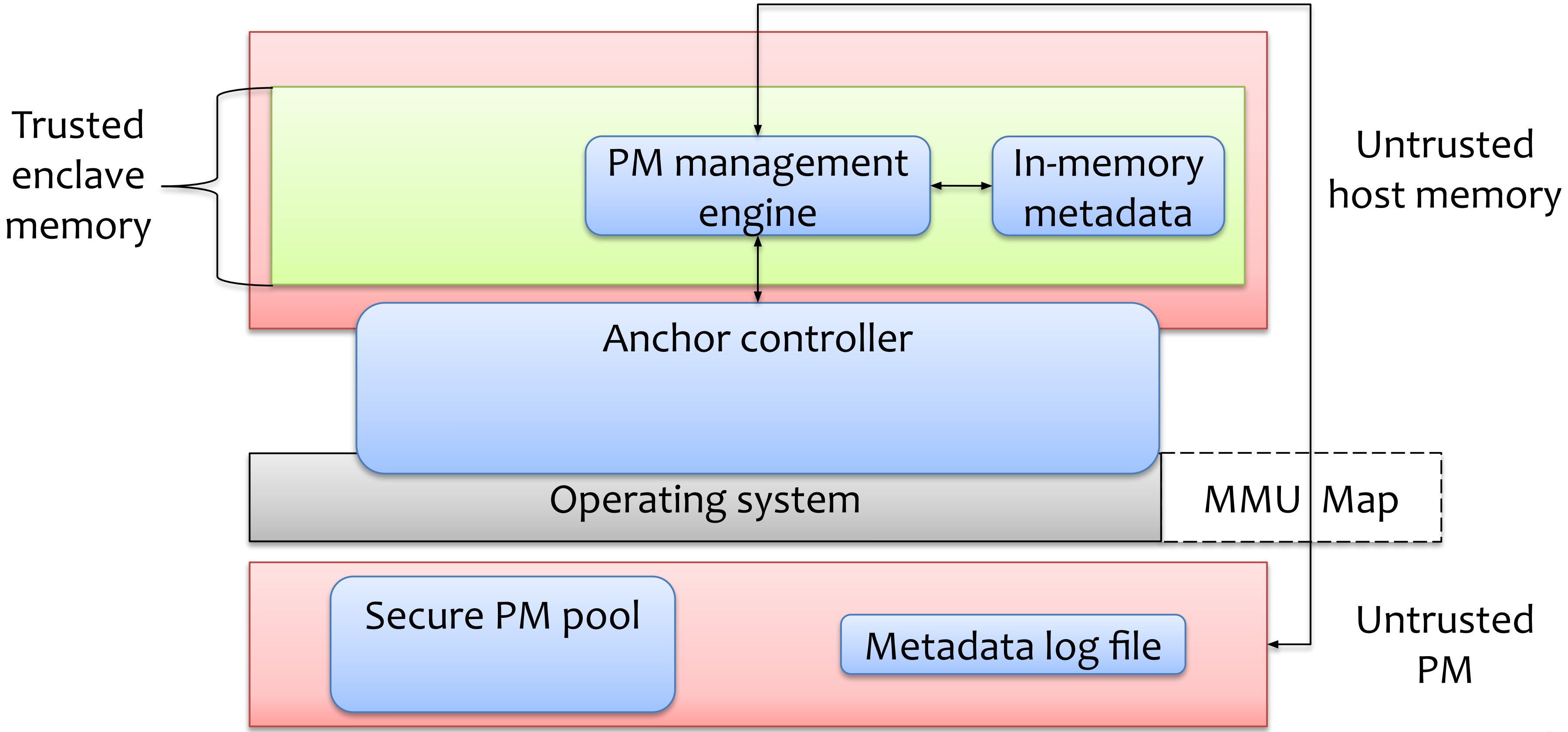
System overview



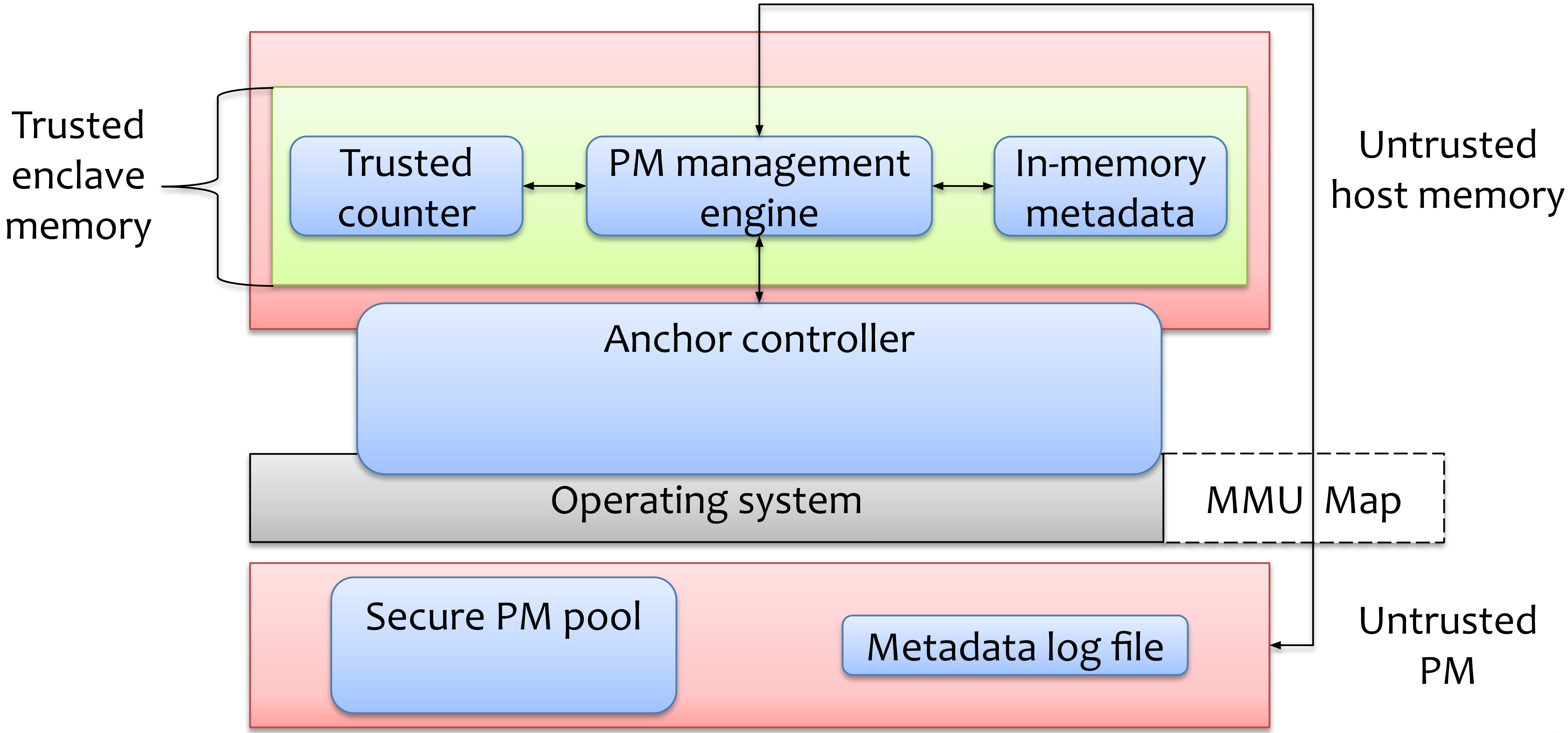
System overview



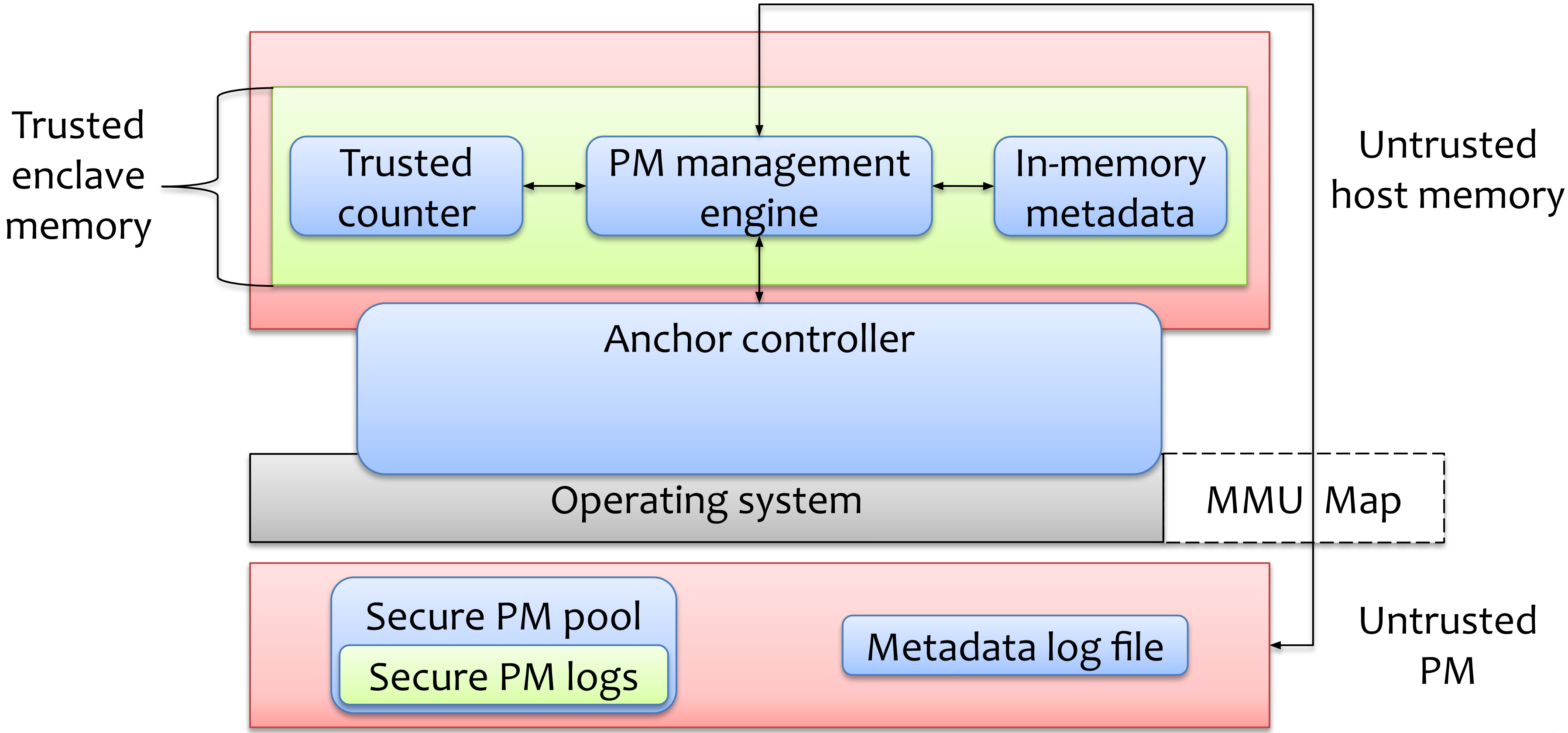
System overview



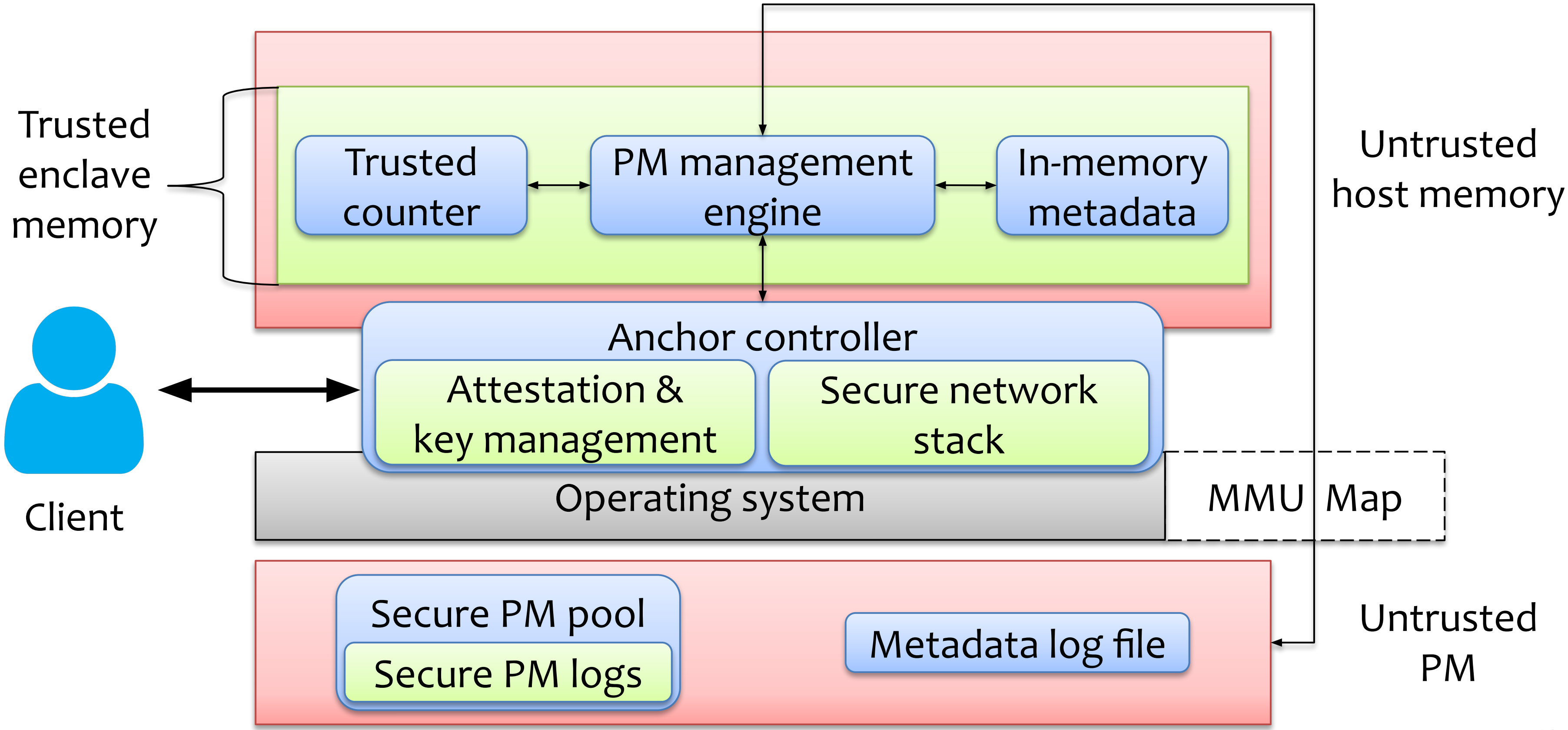
System overview



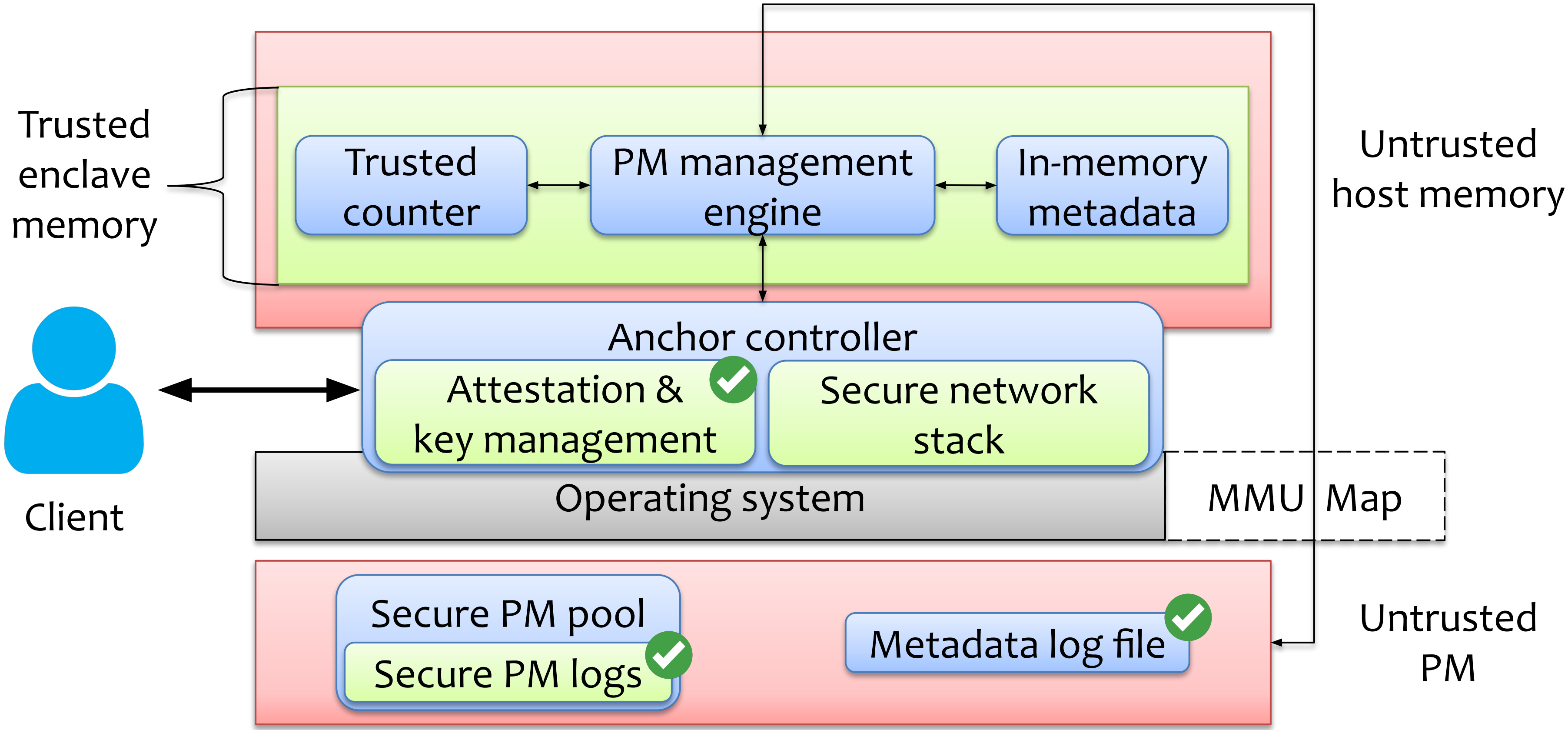
System overview



System overview



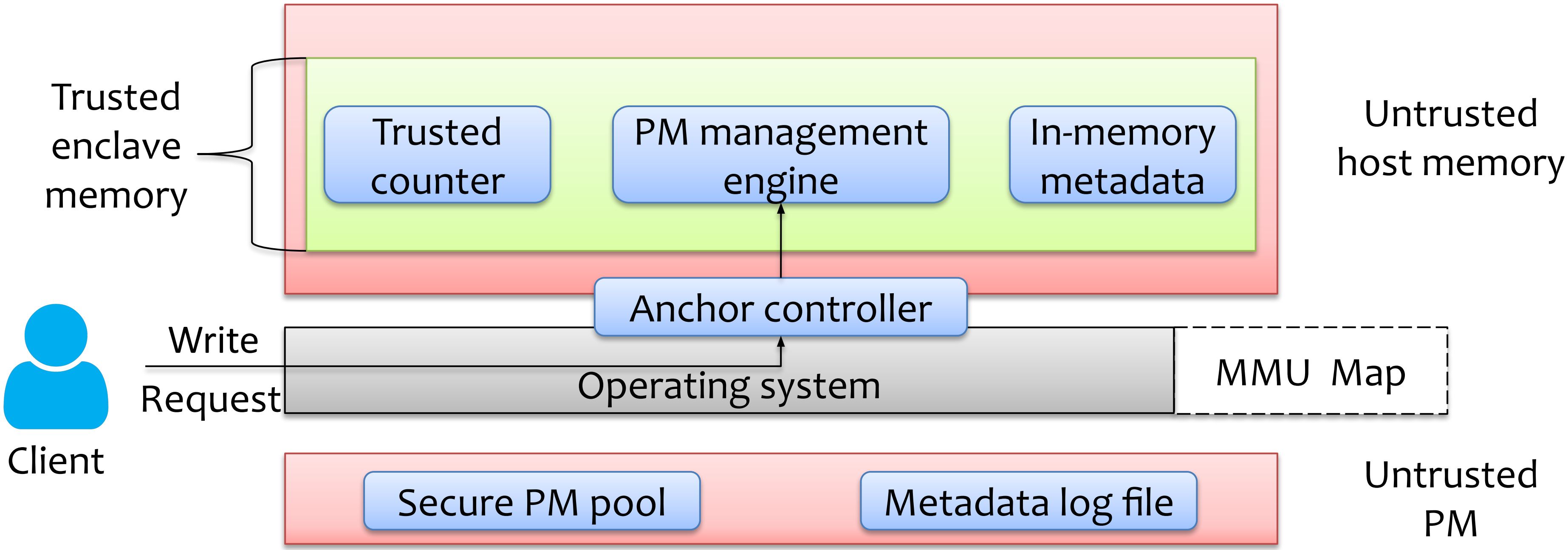
System overview



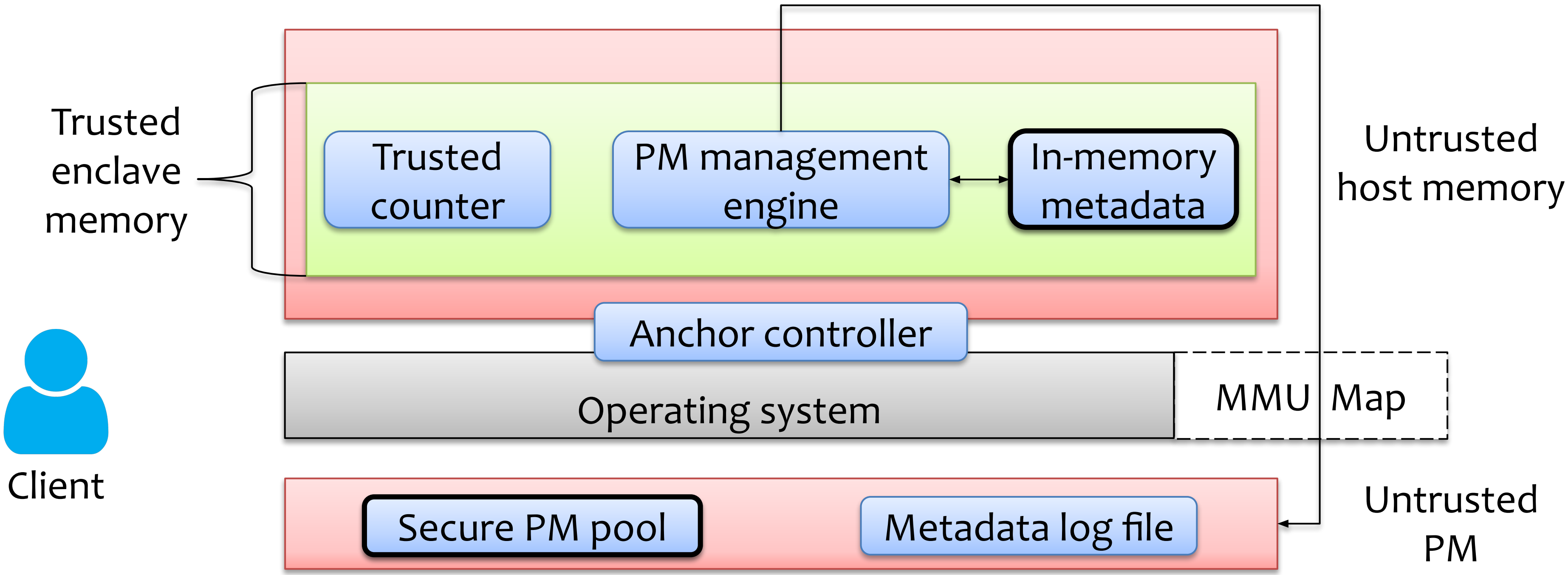
System operations - Write

System operations - Write

1. Write request

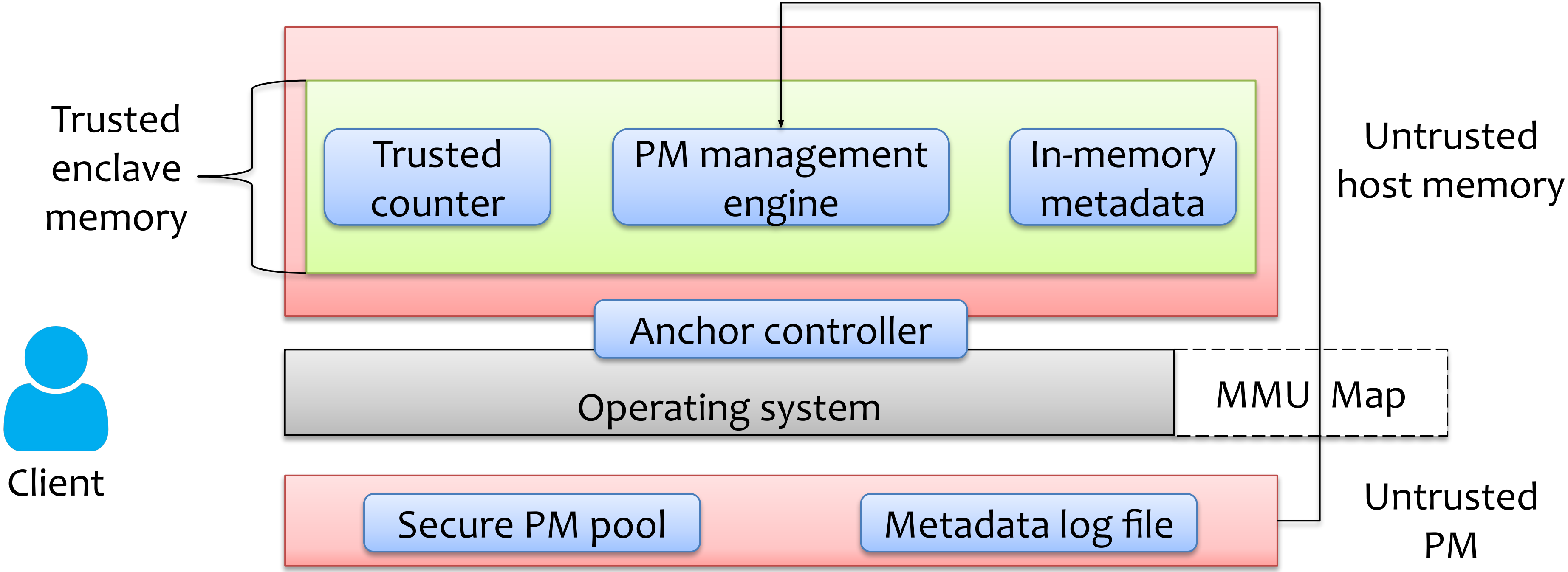


System operations - Write



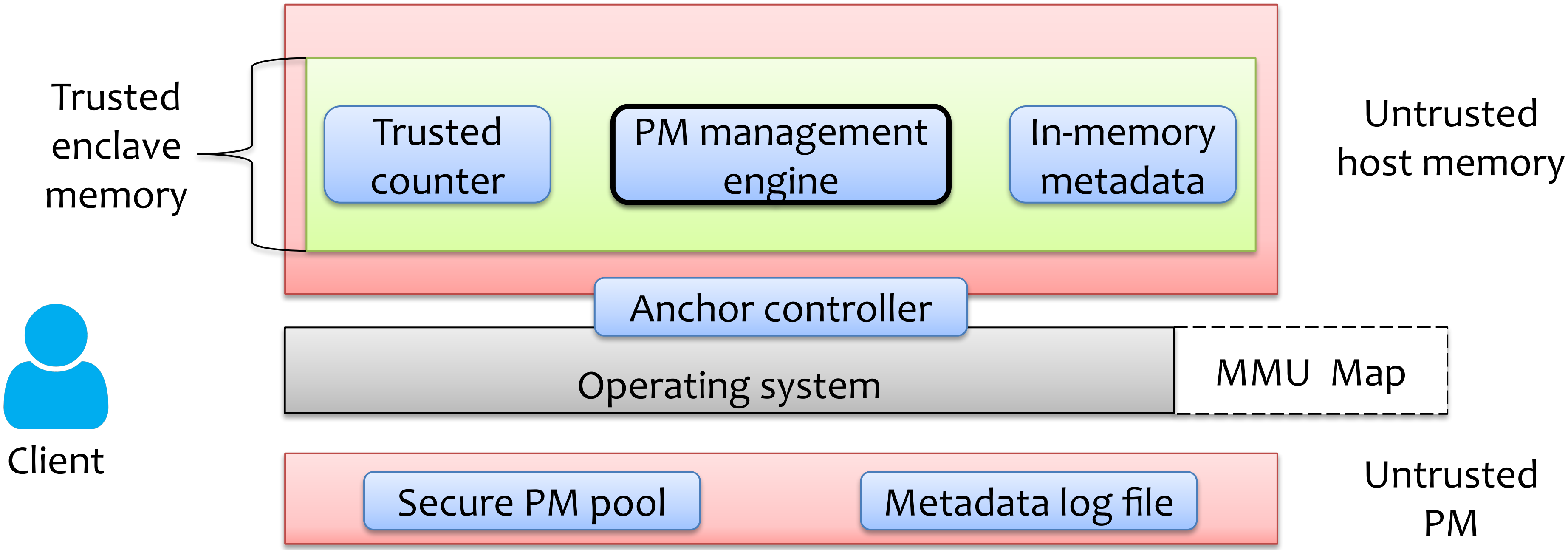
System operations - Write

3. If the object exists, fetch the data

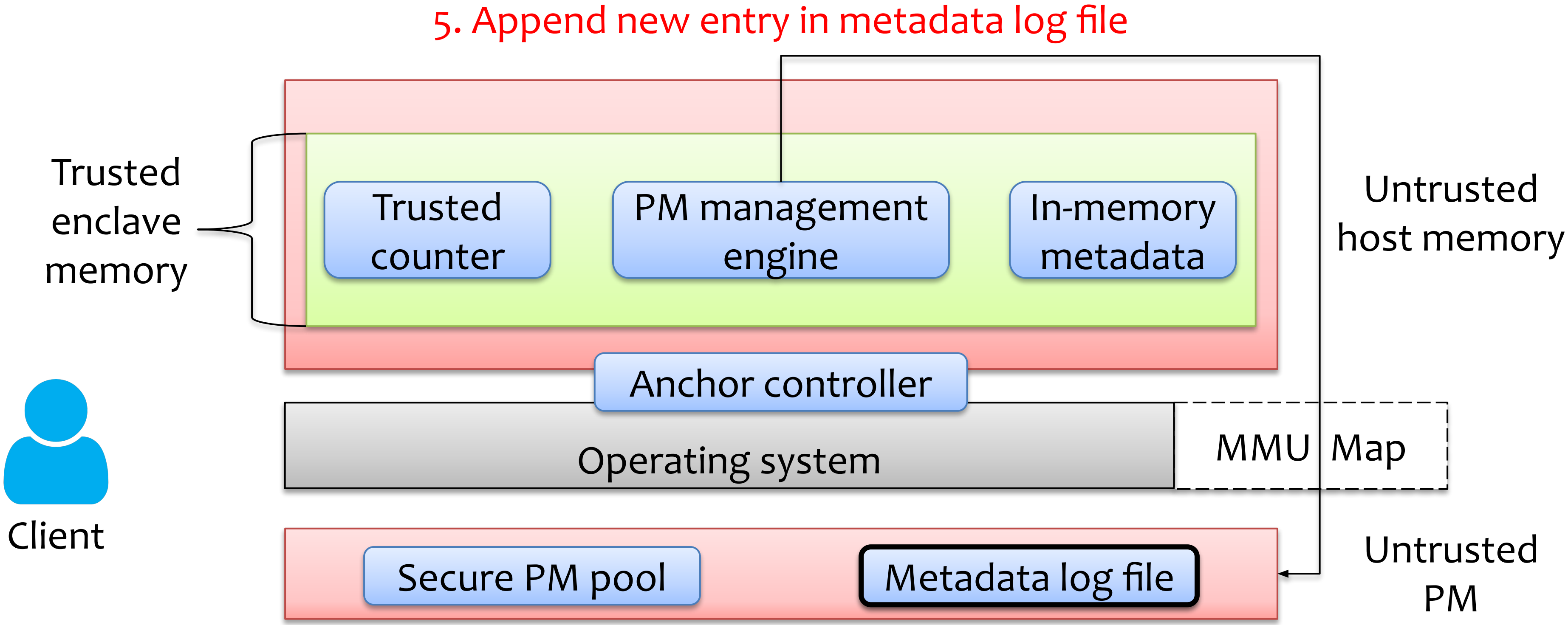


System operations - Write

4. Integrity signature verification & decryption

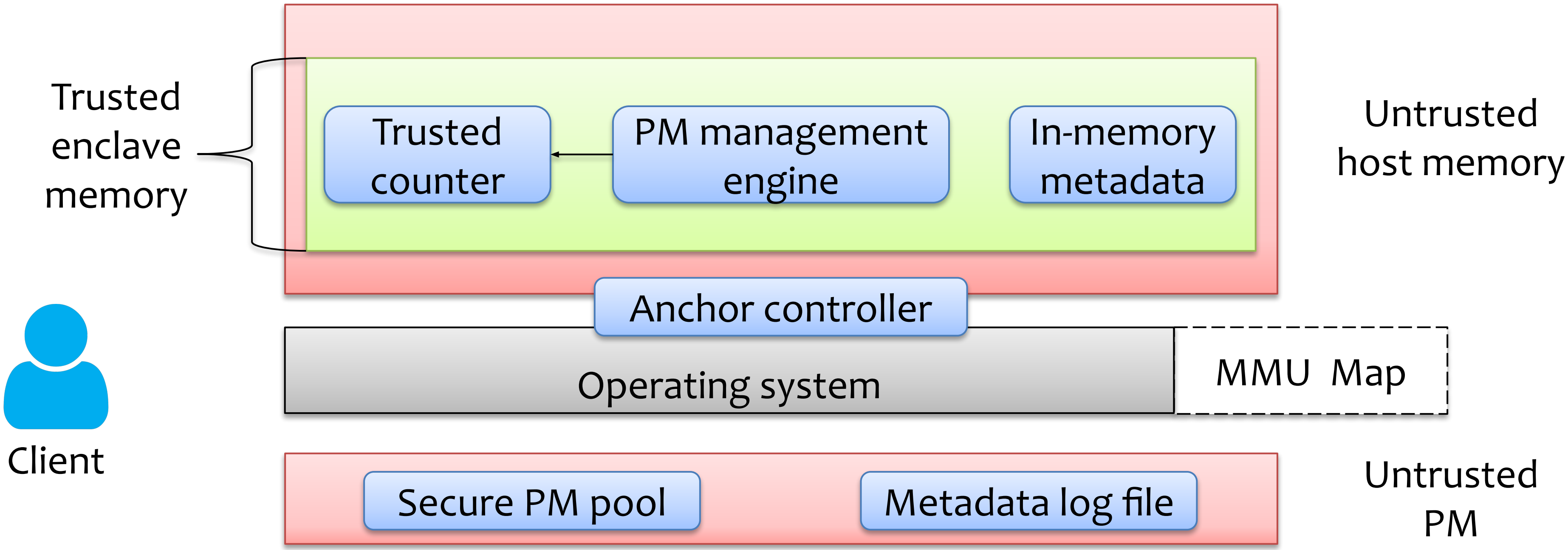


System operations - Write



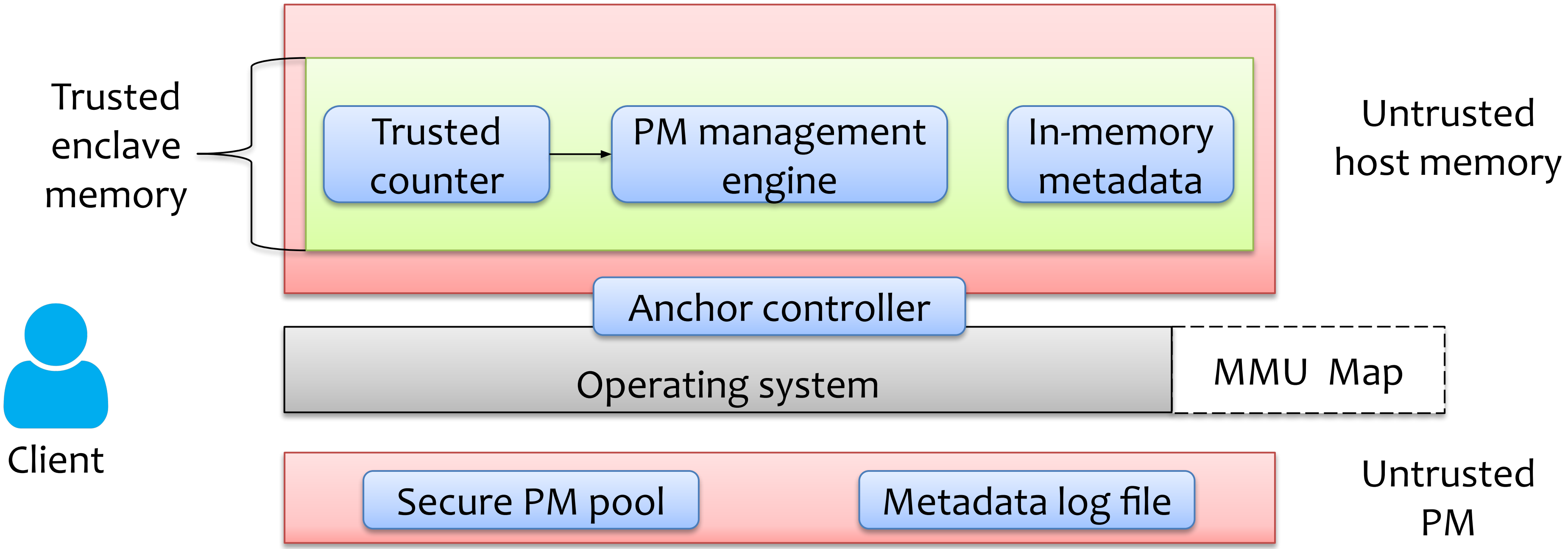
System operations - Write

6. Trusted counter increment



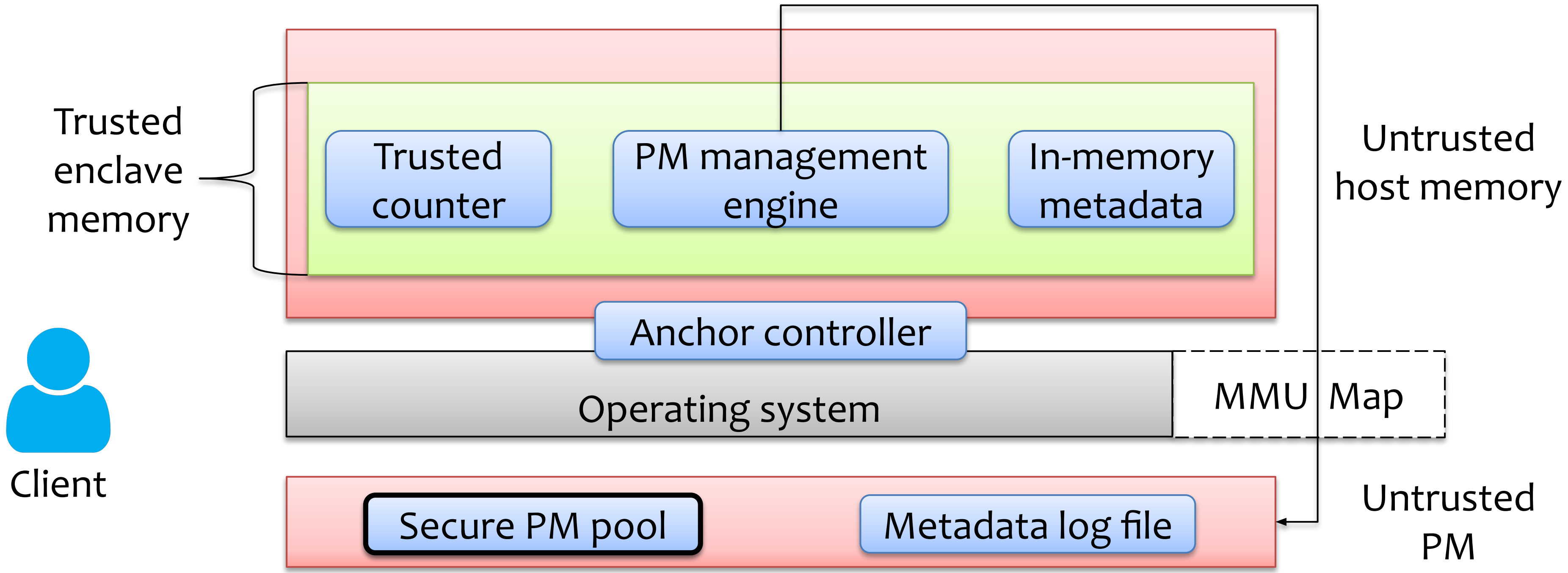
System operations - Write

7. Get next counter and expected time



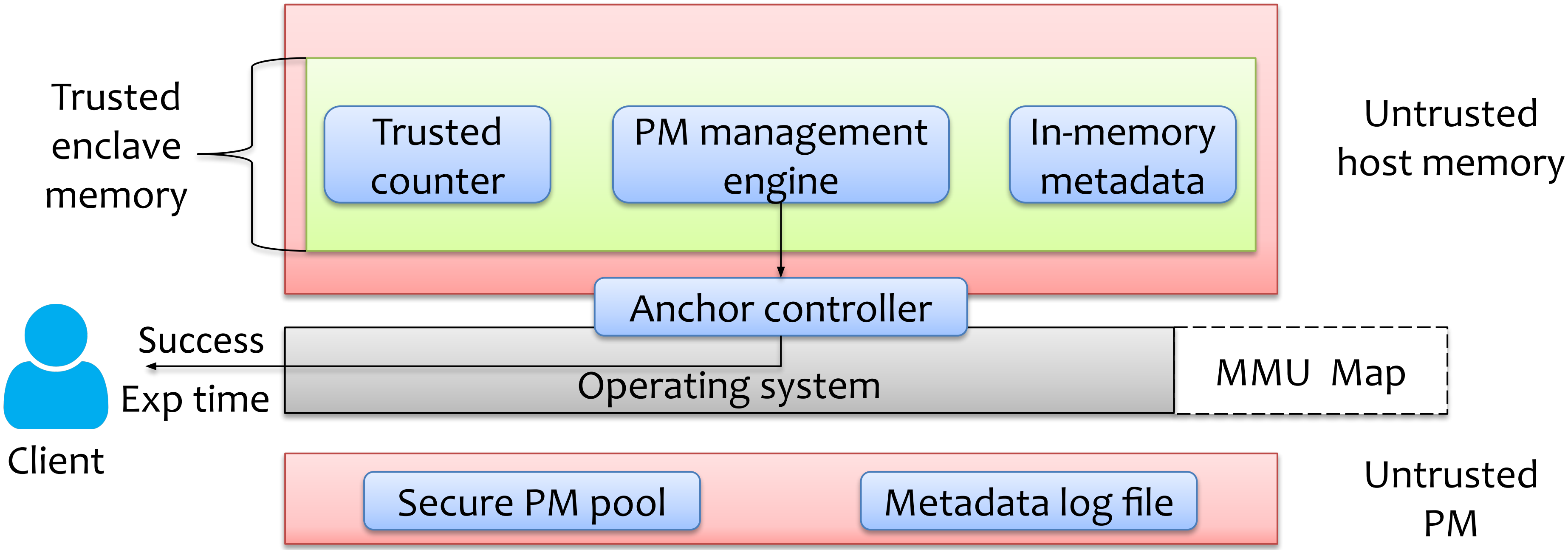
System operations - Write

8. Store updated data in PM pool



System operations - Write

9. Return success & expected time



Outline

- ~~Introduction & Motivation~~
- ~~System design~~
- Evaluation

Evaluation

- What is the performance overhead of Anchor?
  - Persistent indices (ctree, btree, rtree, rbtree, hashmap)

- How does Anchor affect basic PM management operations?
  - PM operations (alloc, update, free)

- What is the recovery and boot-up time of a PM pool with Anchor?
  - Variable metadata log & log sizes

- How do we ensure the security properties of Anchor?
  - Dynamic security analysis & formal verification of security protocols

Evaluation

- **What is the performance overhead of Anchor?**
  - Persistent indices (ctree, btree, rtree, rbtree, hashmap)

- **How does Anchor affect basic PM management operations?**
  - PM operations (alloc, update, free)

- What is the recovery and boot-up time of a PM pool with Anchor?
  - Variable metadata log & log sizes

- How do we ensure the security properties of Anchor?
  - Dynamic security analysis & formal verification of security protocols

Evaluation

- Experimental setup:
  - Intel(R) Core(TM) i9-9900K CPU (3.60GHz, 8 cores) with SGX v.1
  - 64 GB DRAM
  - PM emulation and DAX file system backed by DRAM

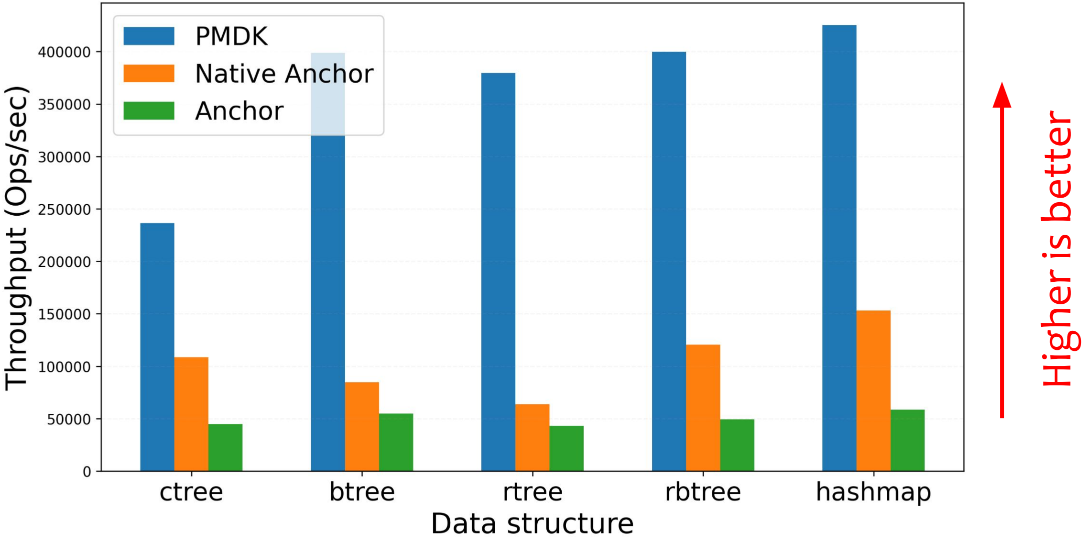
Evaluation

- Experimental setup:
    - Intel(R) Core(TM) i9-9900K CPU (3.60GHz, 8 cores) with SGX v.1
    - 64 GB DRAM
    - PM emulation and DAX file system backed by DRAM

- Variants:
    - *PMDK* → Plain PMDK running in the native environment
    - *Native Anchor* → Anchor running outside the TEE (native environment)
    - *Anchor* → Anchor running inside the TEE

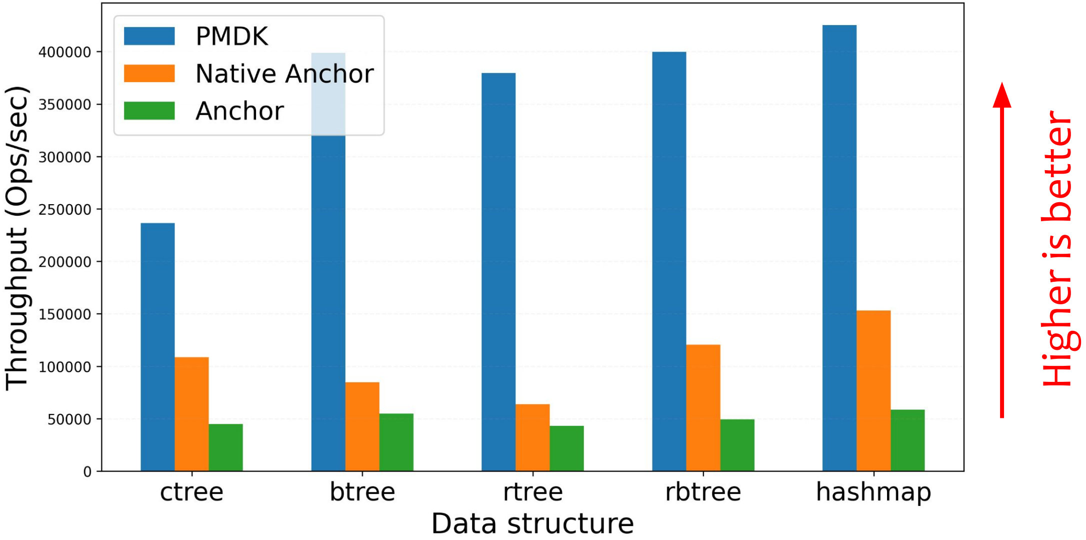
Performance overheads

- PM data structures: *ctree, btree, rtree, rbtree, hashmap*
- *YCSB* workload **10M** ops, **50%** reads / **50%** writes



Performance overheads

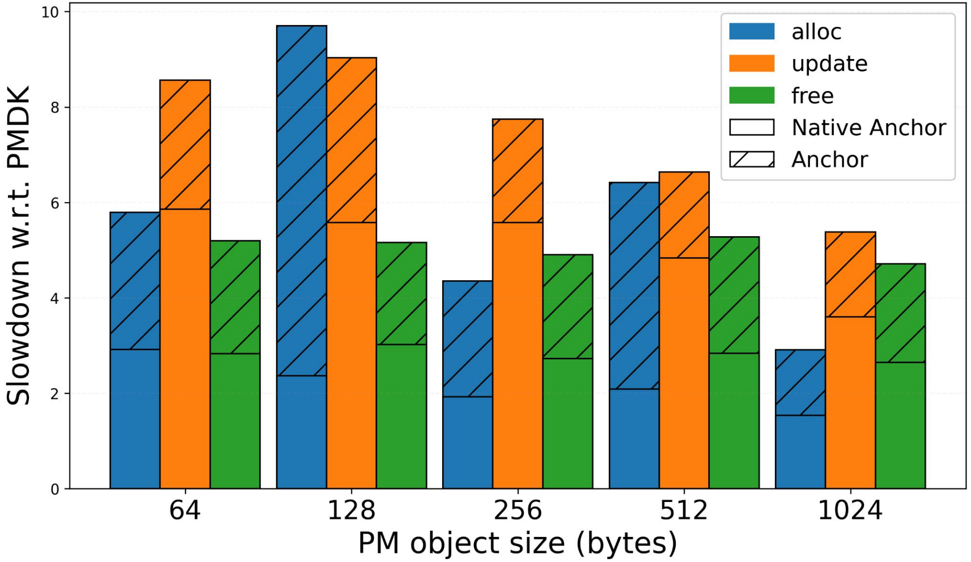
- PM data structures: *ctree, btree, rtree, rbtree, hashmap*
- *YCSB* workload **10M** ops, **50%** reads / **50%** writes



Anchor's slowdown is reasonable considering its strong security properties

PM management operations

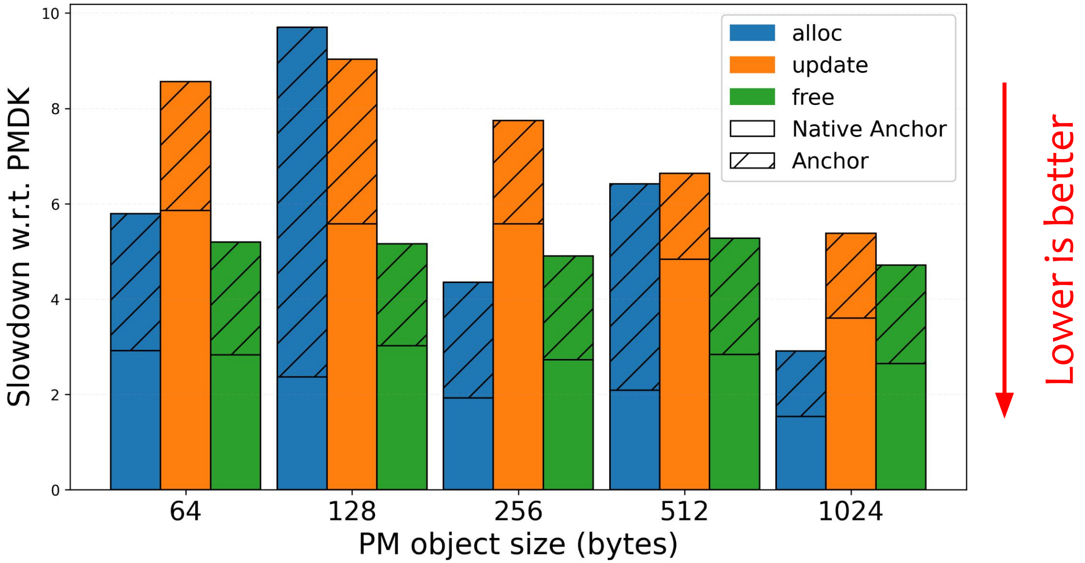
- PM management operations: *alloc, update, free*
- PM object size: *64, 128, 256, 512, 1024 bytes*



Lower is better

PM management operations

- PM management operations: *alloc, update, free*
- PM object size: *64, 128, 256, 512, 1024 bytes*



Anchor incurs lower overheads in PM operations as the PM object size increases

Summary

How to leverage TEEs to design a *secure, performant* PM system that preserves *crash consistency* while following the PM *programming model*?

Anchor: A Library for Building Secure Persistent Memory Systems

- Security properties: *confidentiality, integrity* & *freshness*
- PMDK-like programming model
- Secure crash consistency via a formally verified secure logging protocol
- Secure network stack and formally verified remote attestation protocol

Backup!

Recovery and boot-up time

- Metadata log size: *138, 224 MiB*
- Log size: *0, ~1, ~5 MiB*

Metadata log size (MiB)	138			224		
Log size (MiB)	0	0.98	4.88	0	0.98	4.88
Recovery/boot time (s)	3.02	3.02	3.09	4.17	4.11	4.12

Recovery and boot-up time

- Metadata log size: *138, 224 MiB*
- Log size: *0, ~1, ~5 MiB*

Metadata log size (MiB)	138			224		
Log size (MiB)	0	0.98	4.88	0	0.98	4.88
Recovery/boot time (s)	3.02	3.02	3.09	4.17	4.11	4.12

Anchor has low boot-up times – mostly determined by the metadata log size

Challenge #4: Formal verification & security analysis

- The secure logging protocol must preserve the required security properties

- The attestation protocol must be correct and adhere to the security principles

- The data management operations do not introduce additional attack vectors

Challenge #4: Formal verification & security analysis

- The secure logging protocol must preserve the required security properties
- The attestation protocol must be correct and adhere to the security principles
- The data management operations do not introduce additional attack vectors

Formally verify the secure logging and the remote attestation protocols & leverage dynamic analysis tools for security analysis

Security analysis

- Dynamic security analysis
    - Memory safety guarantees using Address Sanitizer
    - Crash consistency using Valgrind’s memcheck

- Formal verification of Security Protocols using Tamarin
    - Remote attestation protocol
    - Secure logging protocol

Security analysis

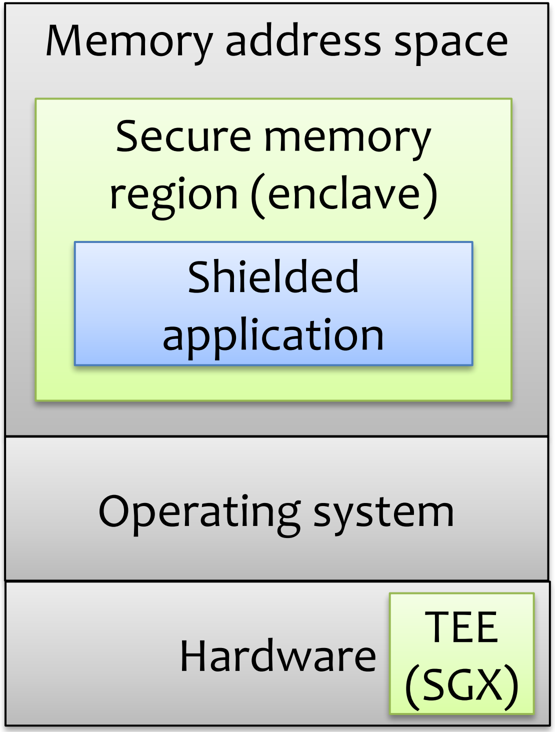
- Dynamic security analysis
  - Memory safety guarantees using Address Sanitizer
  - Crash consistency using Valgrind’s memcheck

- Formal verification of Security Protocols using Tamarin
  - Remote attestation protocol
  - Secure logging protocol

Anchor does not introduce memory safety bugs, preserves the crash consistency property and uses formally verified security protocols

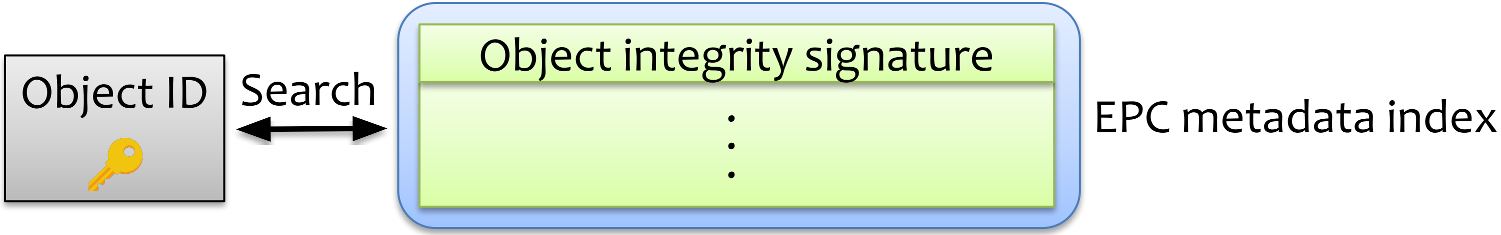
Trusted execution environments

- **TEE:** Hardware extensions (ISAs) for trusted computing (e.g. Intel SGX, ARM TrustZone)
- **Abstraction:** Secure memory region where application code and data are secured
- **Shielded execution:** Runtime framework for running unmodified applications inside a TEE



Component #1: In-memory metadata

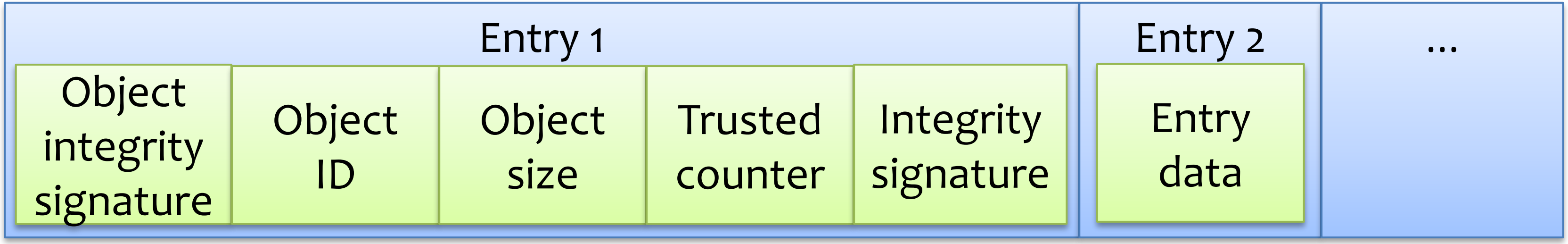
In-memory structures maintain object metadata



EPC index for secure metadata store and data caching for performance

Component #2: Metadata log file (manifest)

Manifest file maintains pool object metadata



Entry 1

Object integrity signature

Object ID

Object size

Trusted counter

Integrity signature

Entry 2

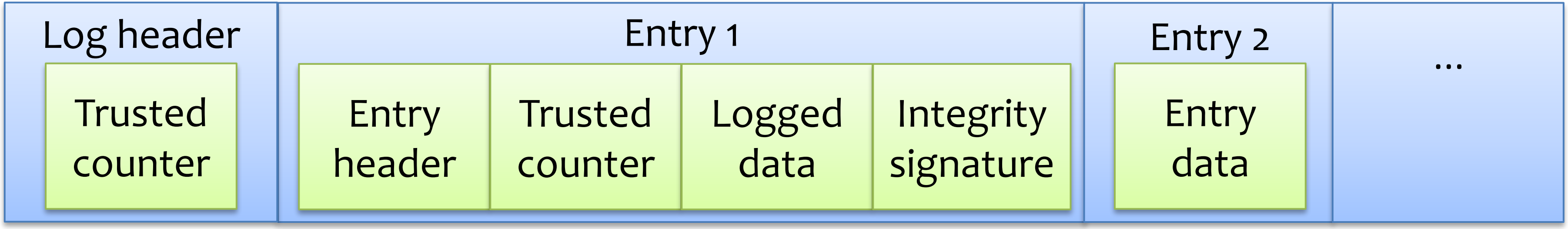
Entry data

...

Loaded manifest data is the base for integrity and freshness checks

Component #3: Secure undo/redo log

Log mechanism to preserve crash consistency and security principles



Log header

Trusted counter

Entry 1

Entry header

Trusted counter

Logged data

Integrity signature

Entry 2

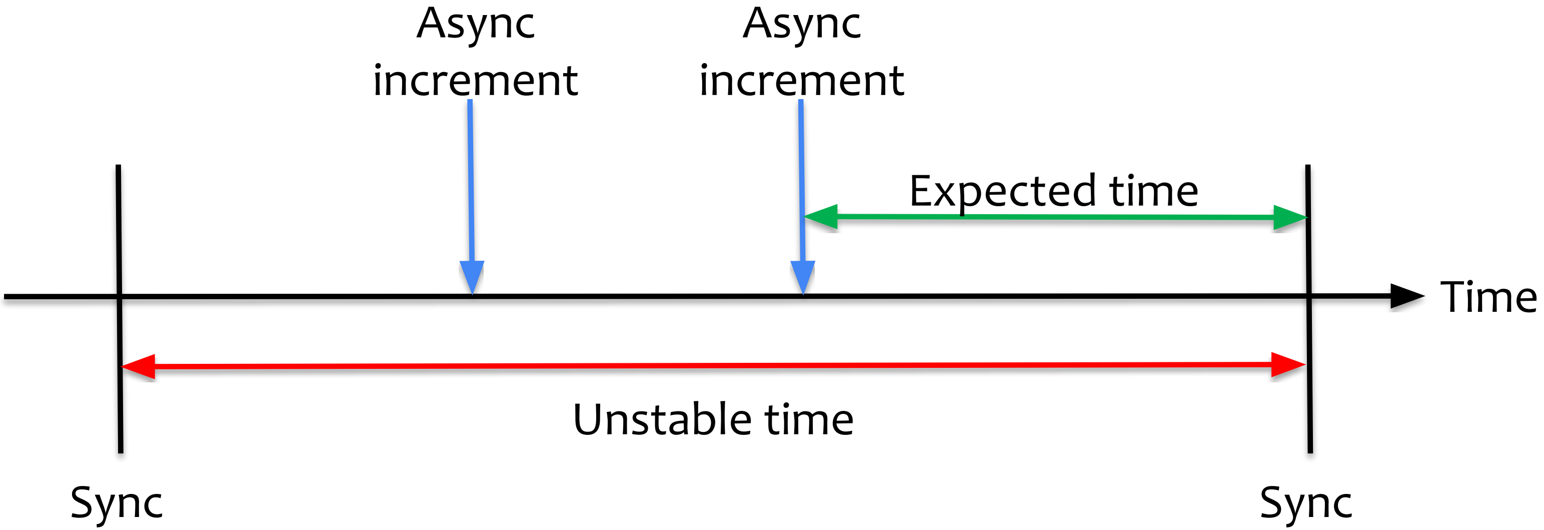
Entry data

...

Achieve secure logging leveraging integrity signatures and trusted counters

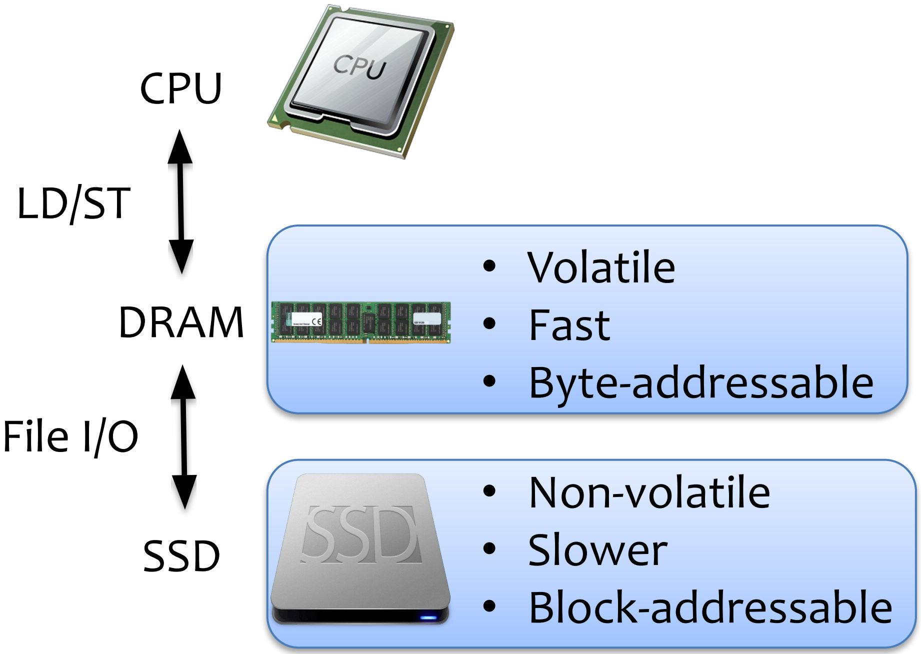
Component #4: Trusted counter

Trusted counter helps us argue about the freshness property



Trusted counter checks performed for freshness verification

Persistent memory



CPU

LD/ST

DRAM

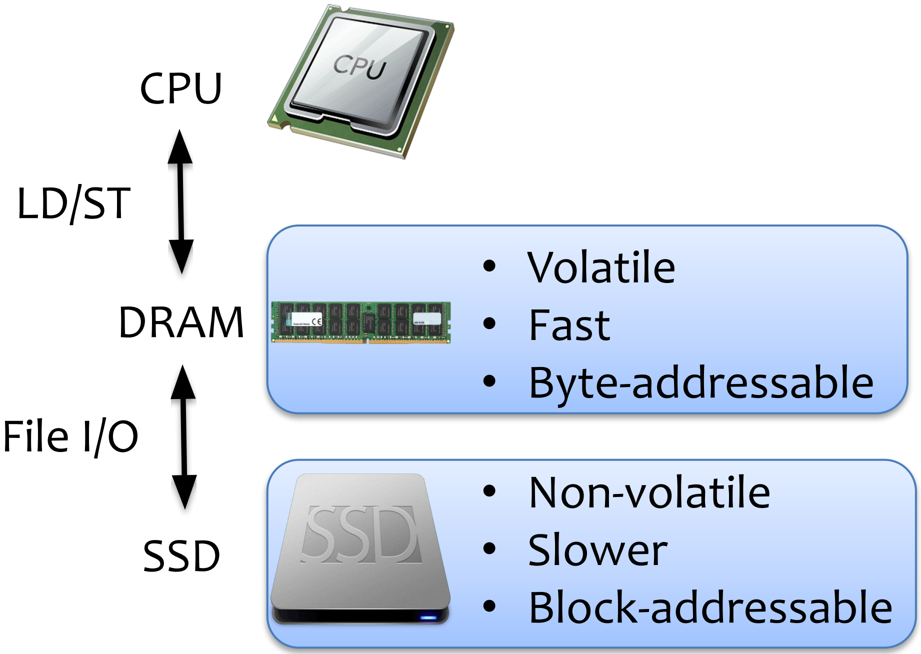
- Volatile
- Fast
- Byte-addressable

File I/O

SSD

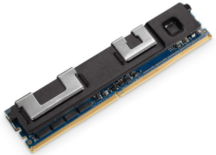
- Non-volatile
- Slower
- Block-addressable

Persistent memory



- Volatile
- Fast
- Byte-addressable

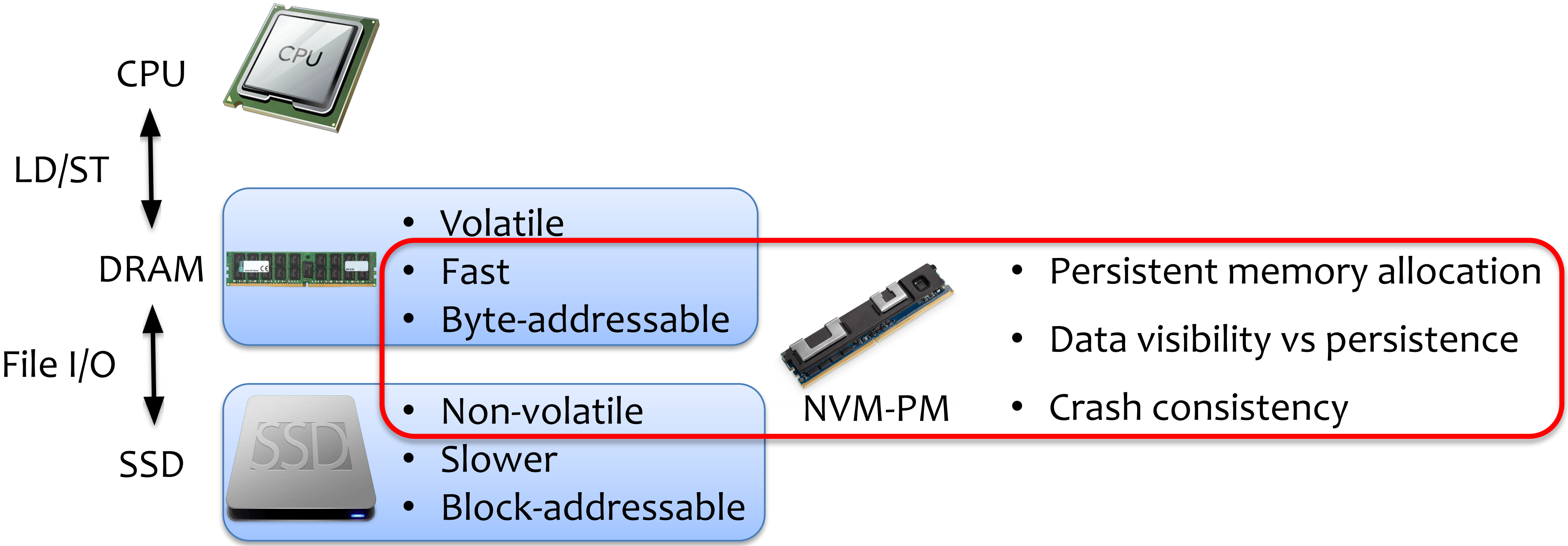
- Non-volatile
- Slower
- Block-addressable



NVM-PM

- Persistent memory allocation
- Data visibility vs persistence
- Crash consistency

Persistent memory



- Volatile
- Fast
- Byte-addressable

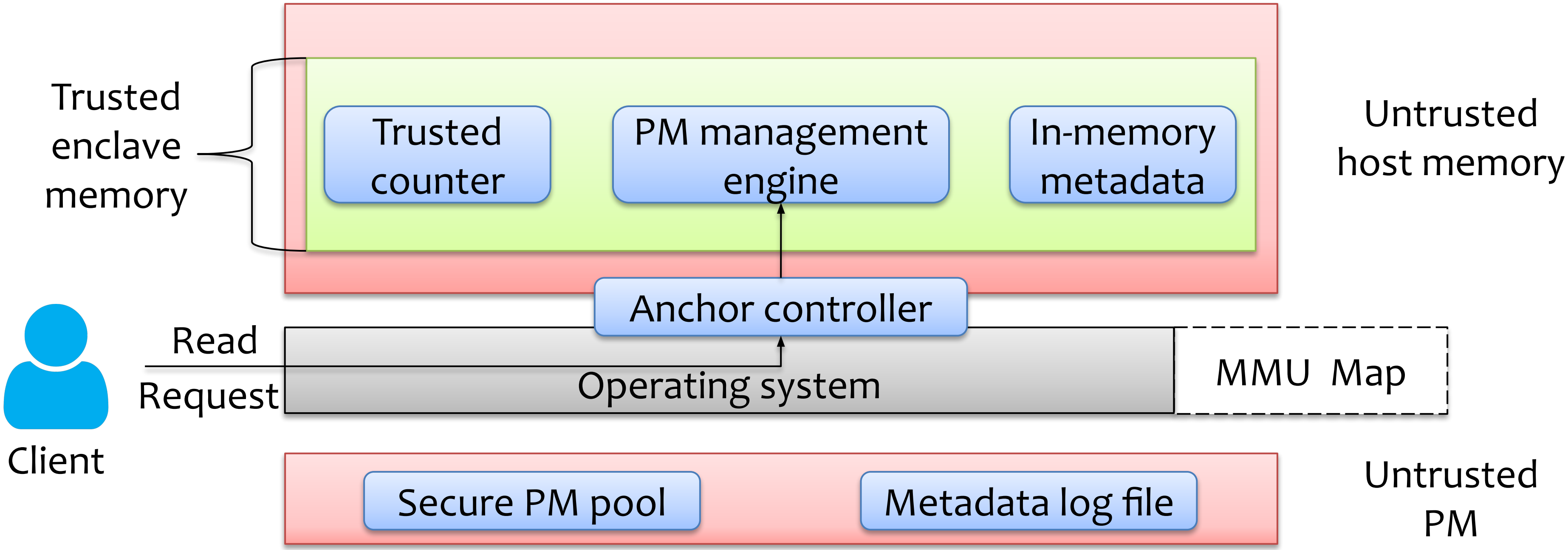
- Non-volatile
- Slower
- Block-addressable

- Persistent memory allocation
- Data visibility vs persistence
- Crash consistency

System operations - Read

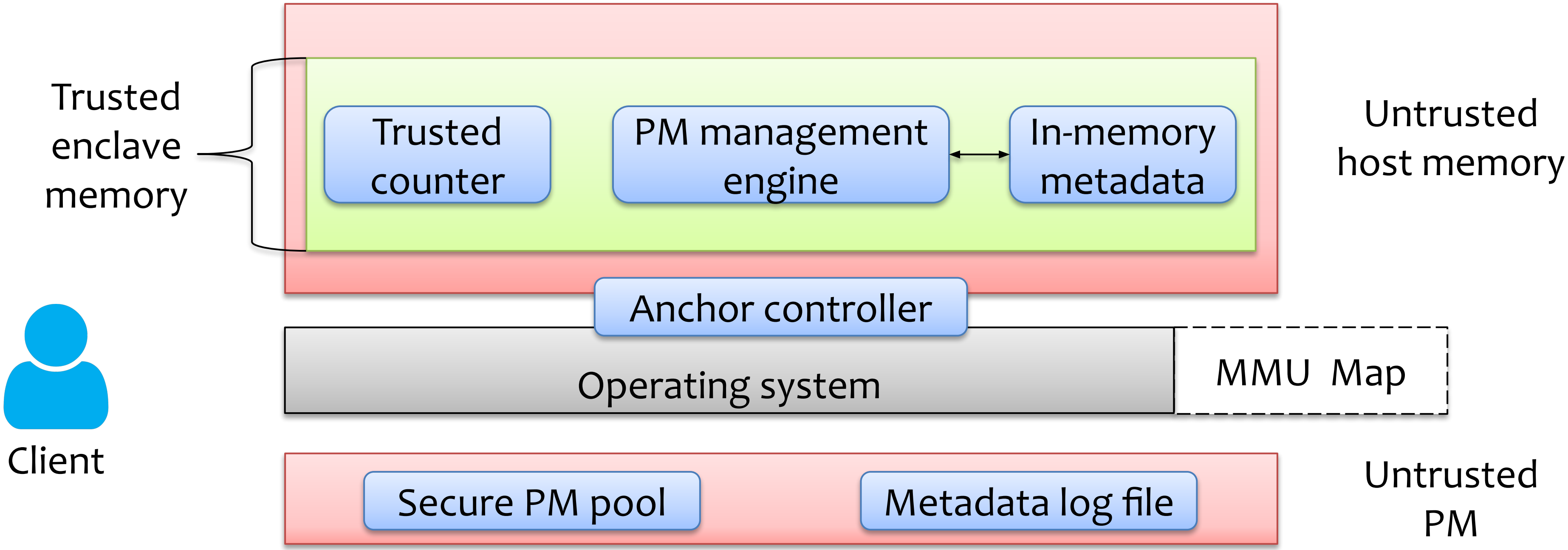
System operations - Read

1. Read request

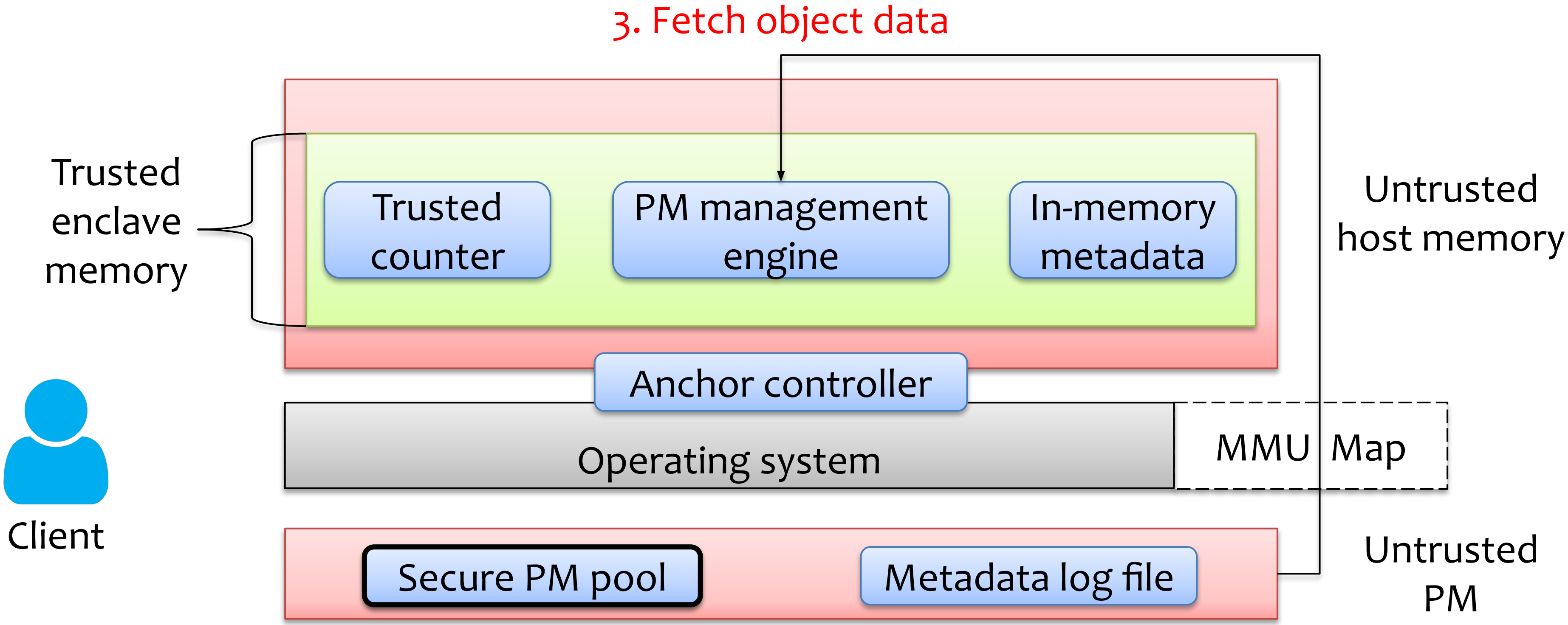


System operations - Read

2. Integrity signature lookup

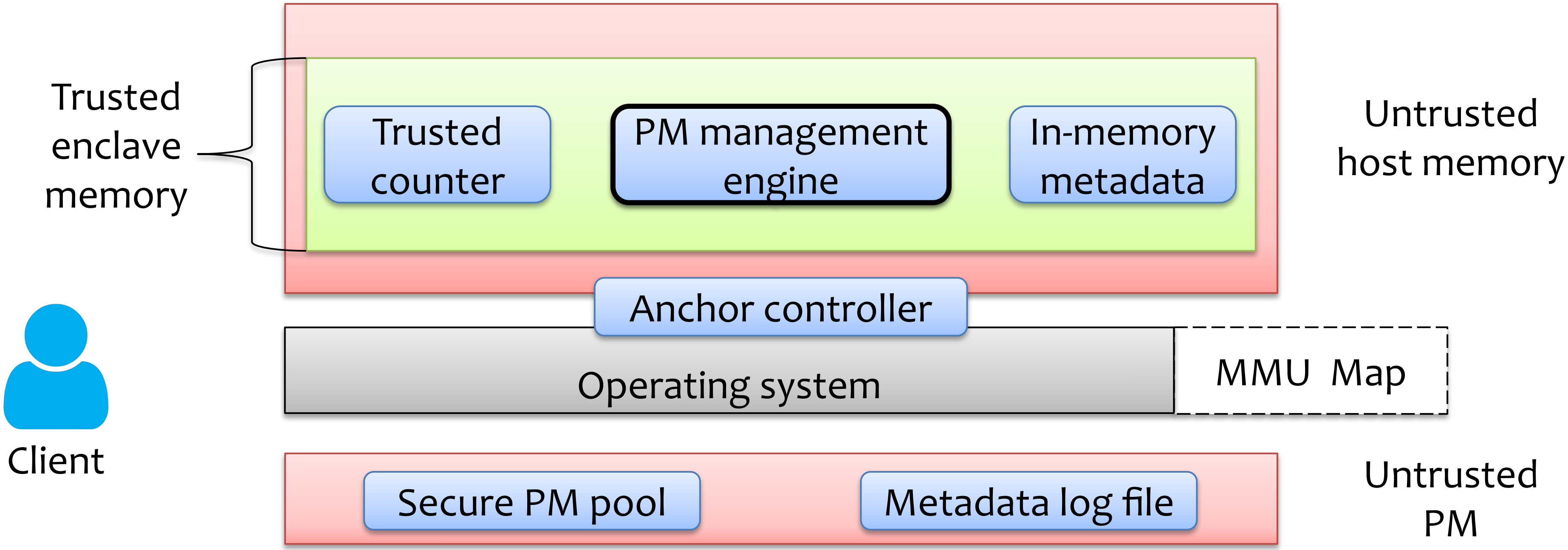


System operations - Read



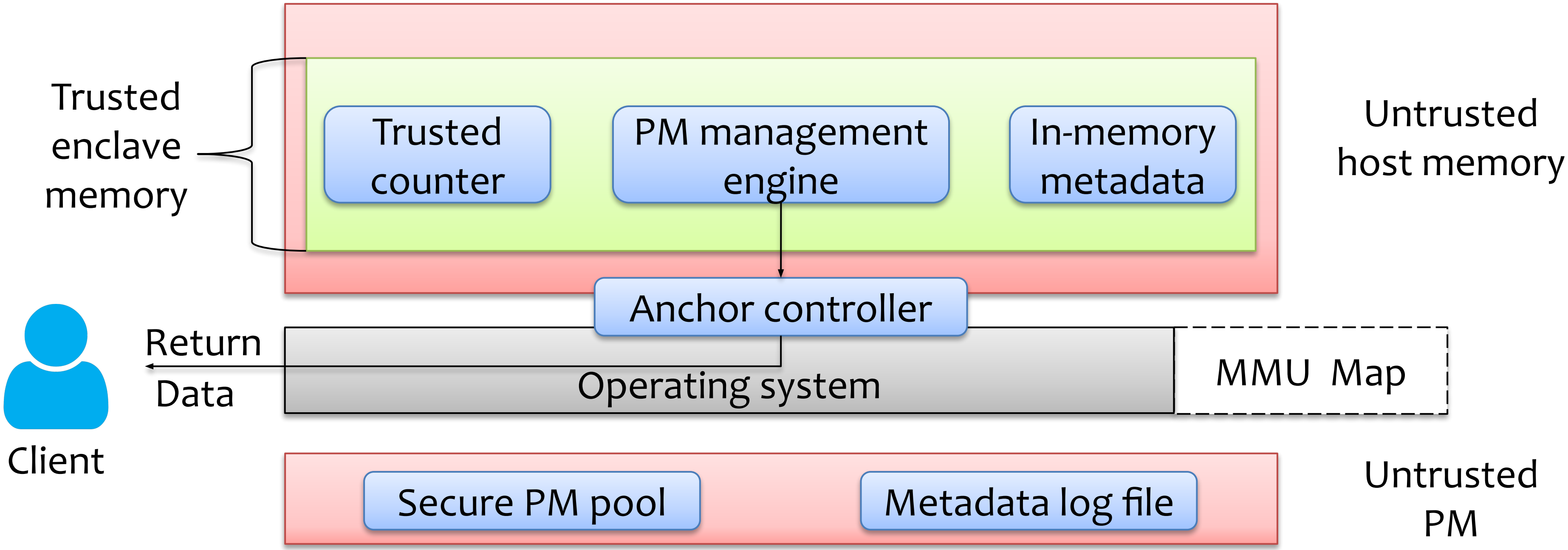
System operations - Read

4. Integrity signature verification & decryption



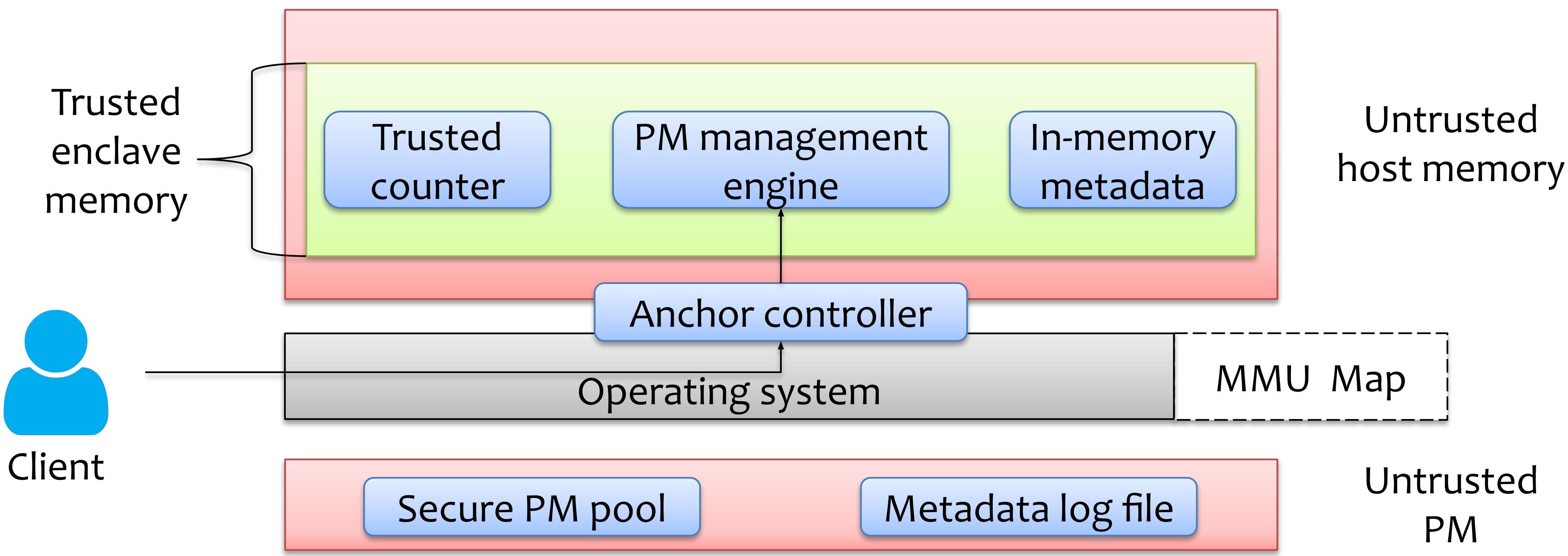
System operations - Read

5. Return object data to the client

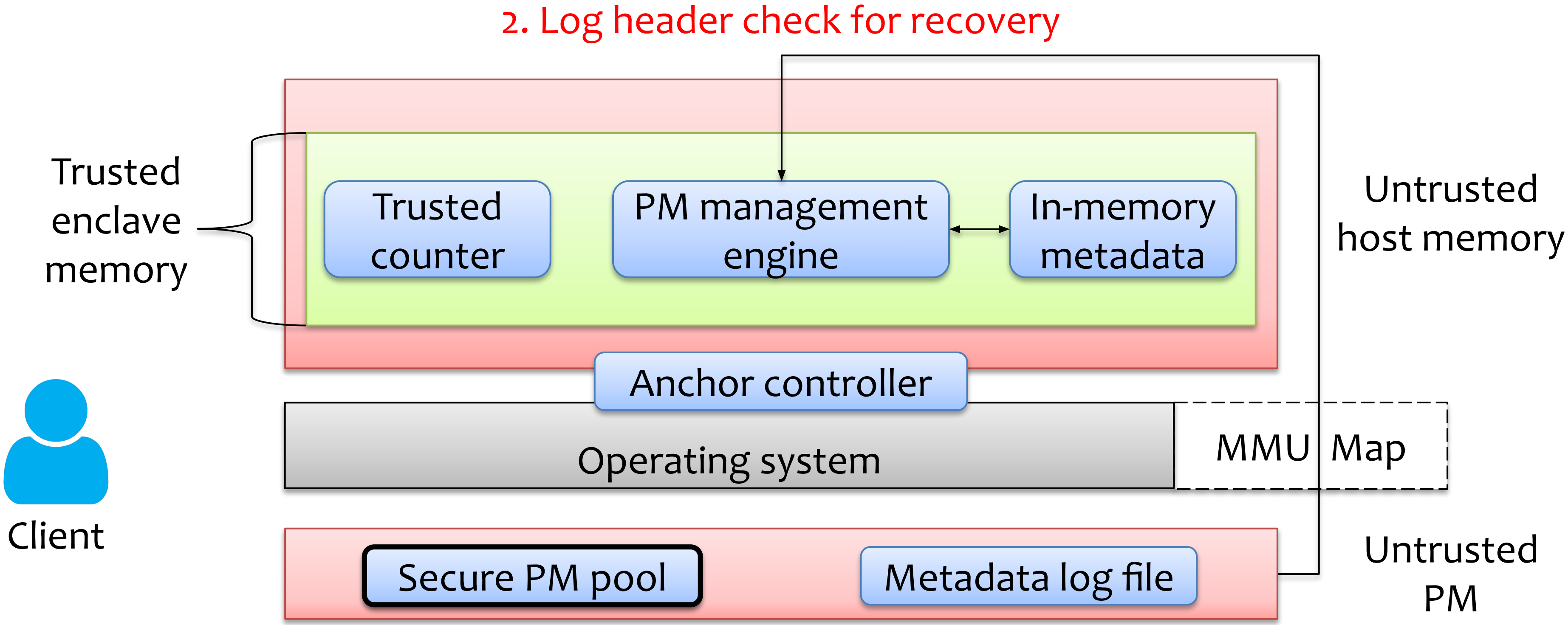


System operations - Recovery

System operations - Recovery

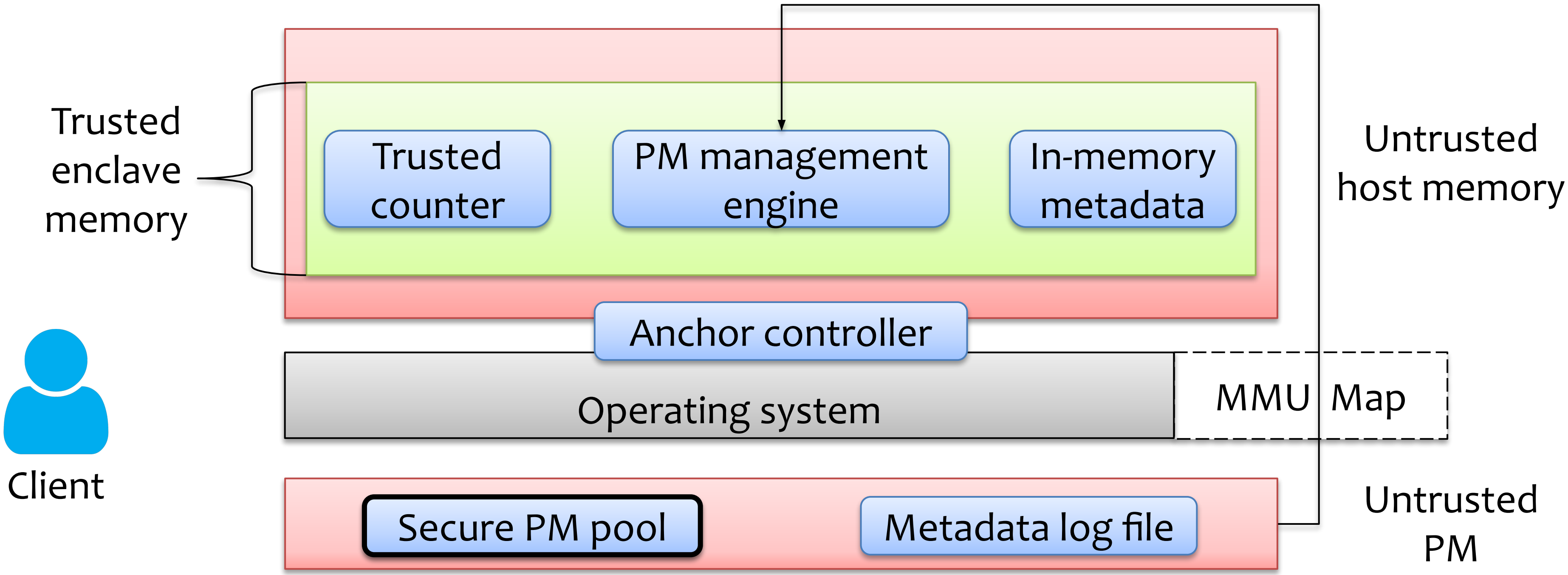


System operations - Recovery



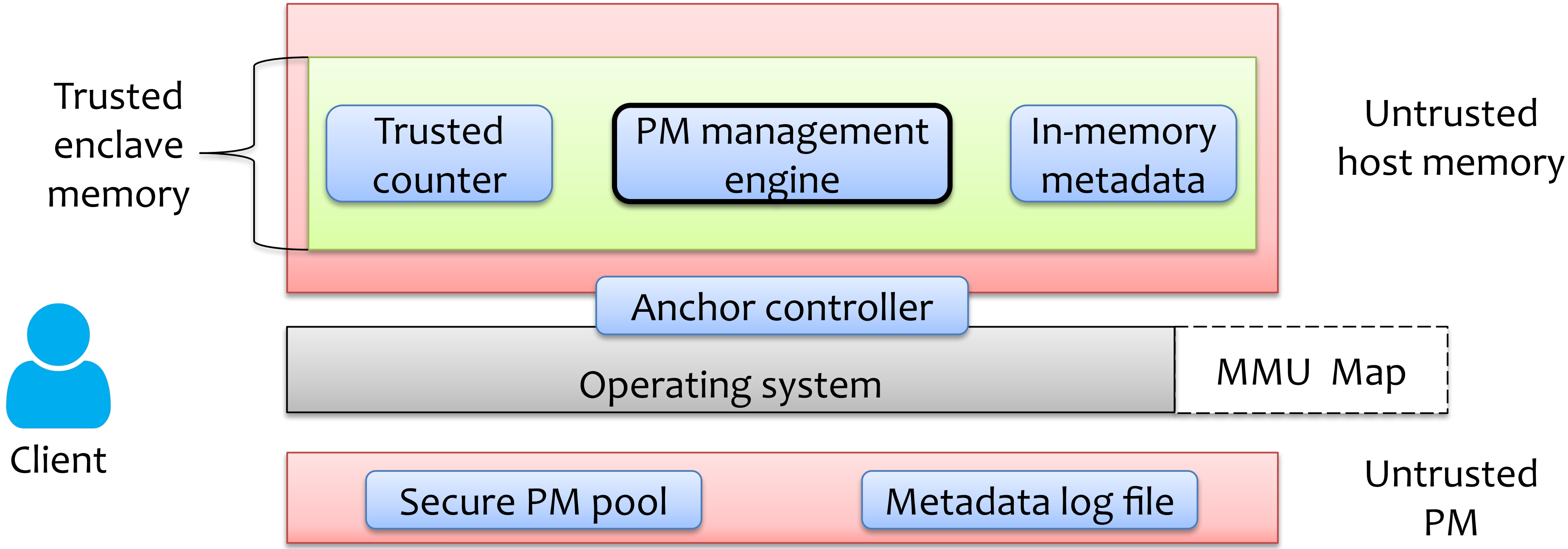
System operations - Recovery

3. Fetch log entries in secure memory



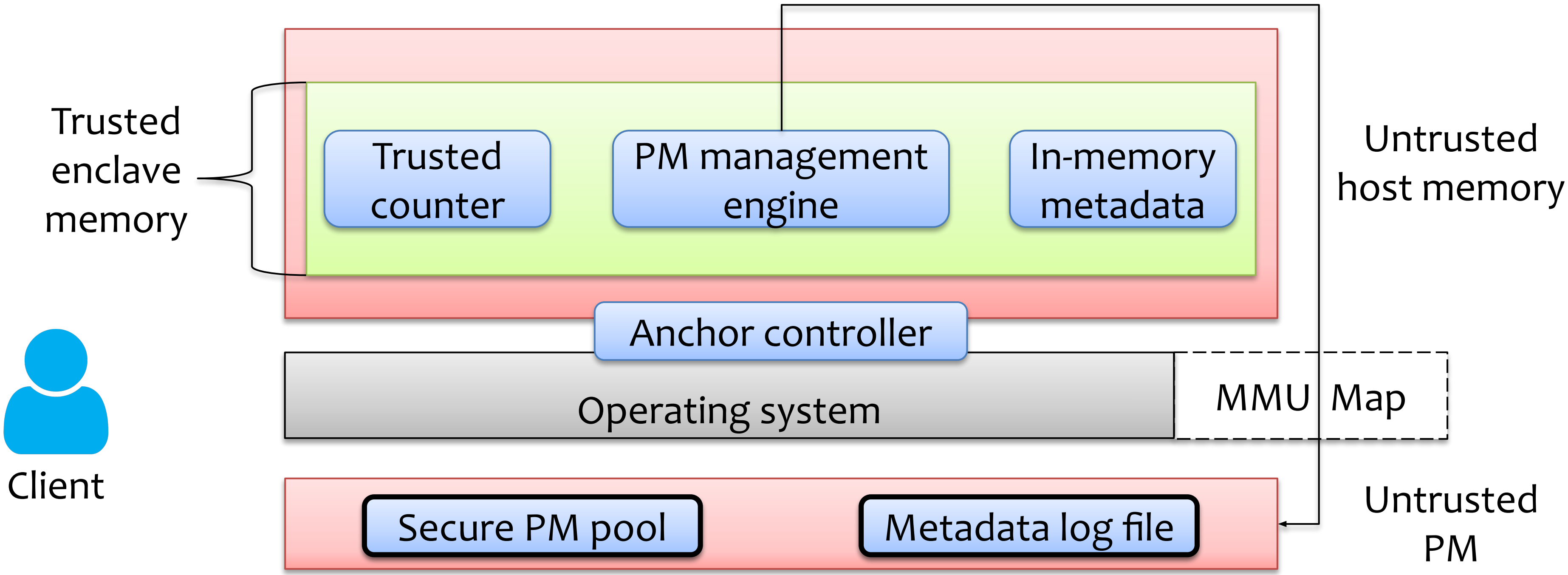
System operations - Recovery

4. Perform integrity & freshness check

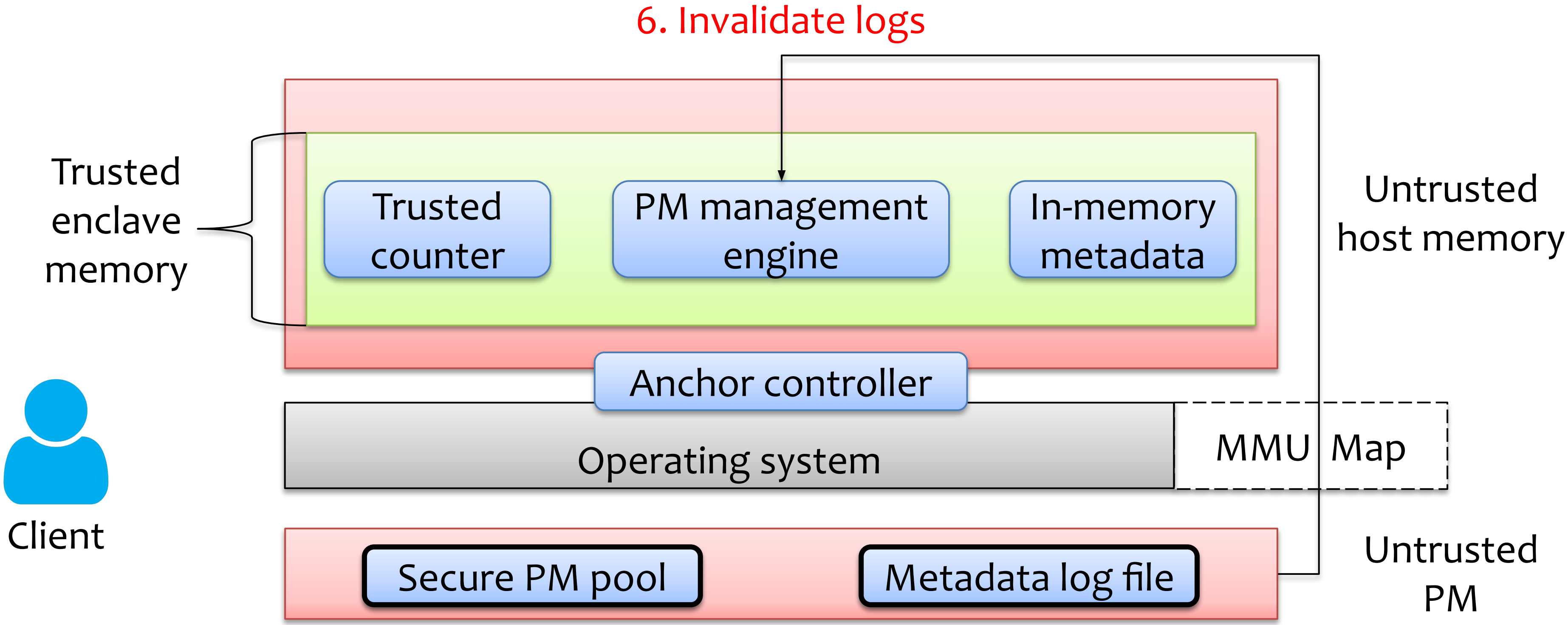


System operations - Recovery

5. Apply (undo/redo) logged operations to PM

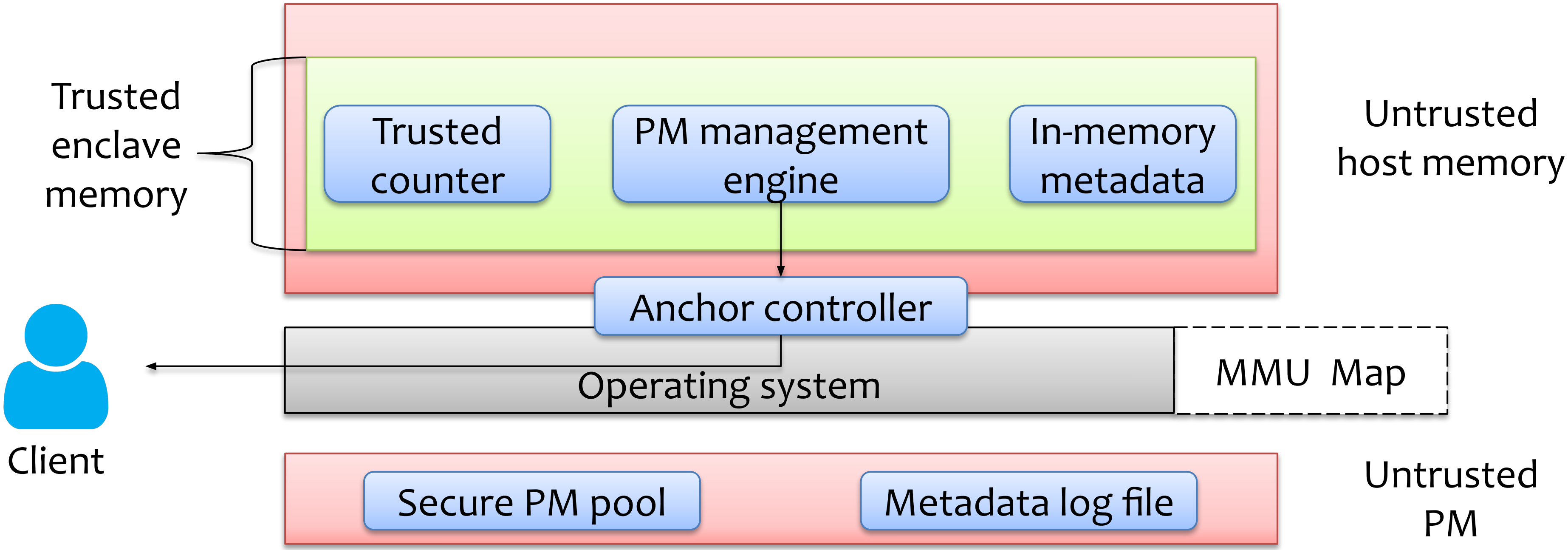


System operations - Recovery



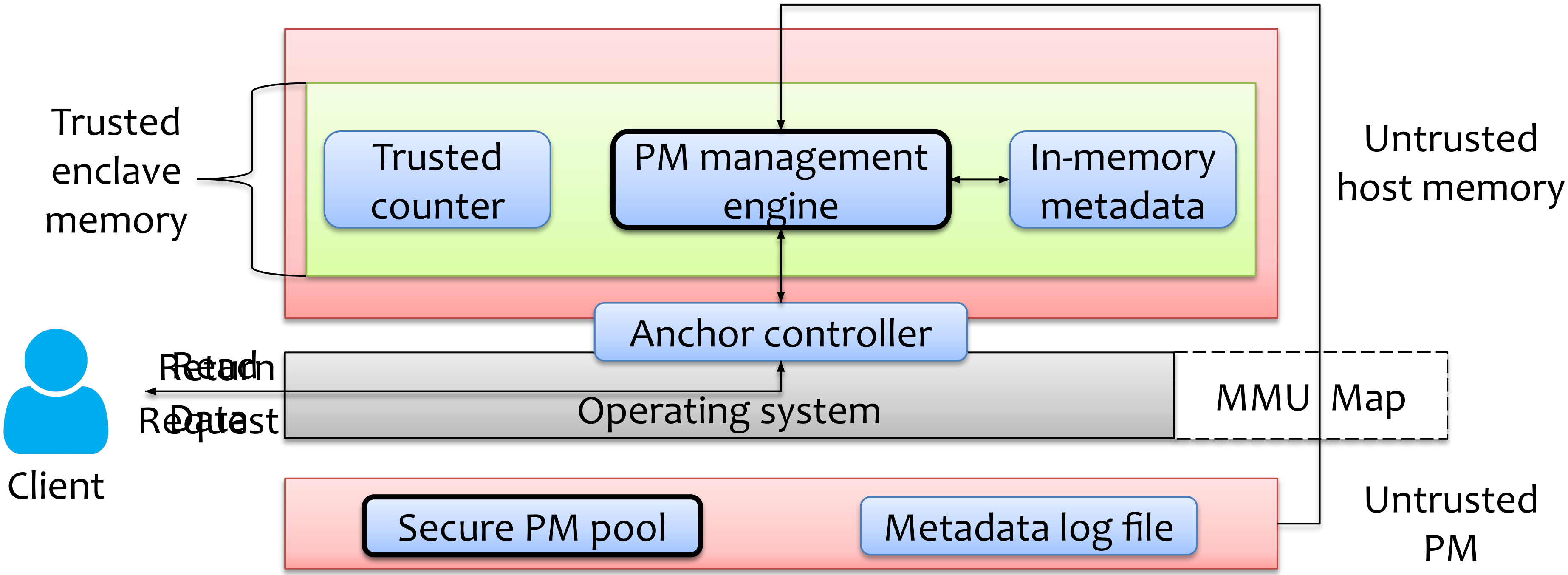
System operations - Recovery

7. Return successful recovery message



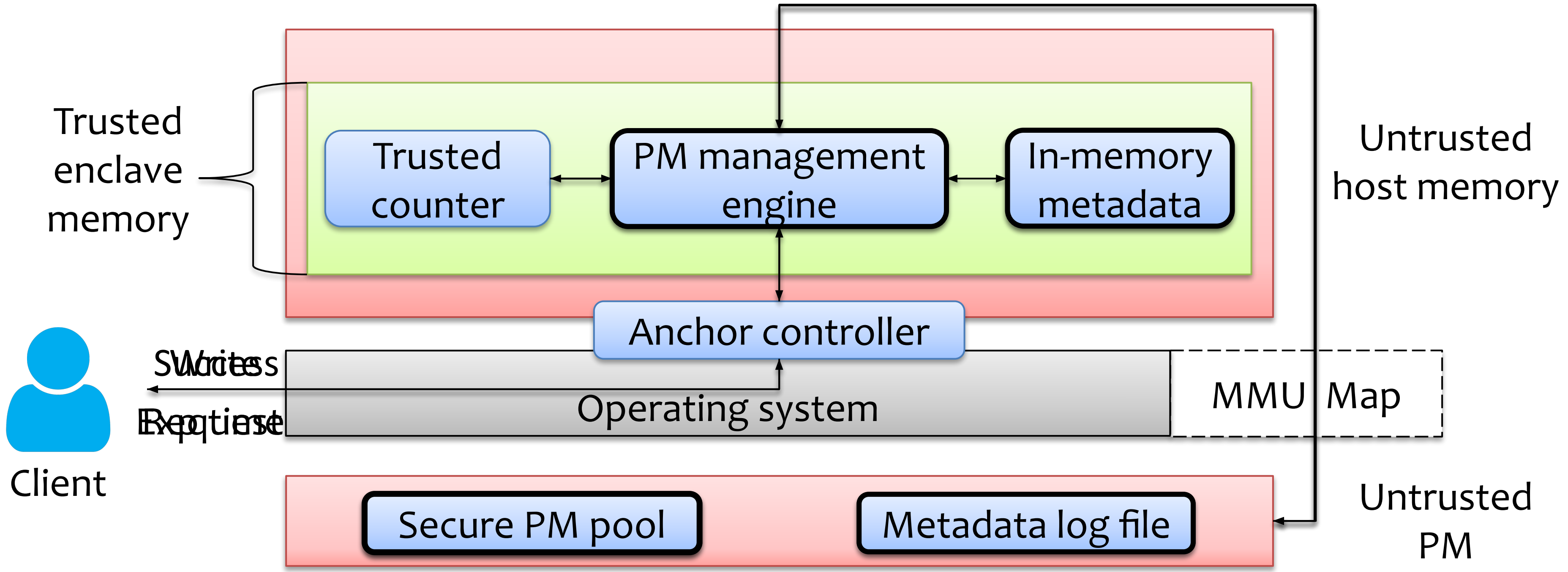
System operations - Read (embedded animations)

4. Integrity [illegible] & decryption



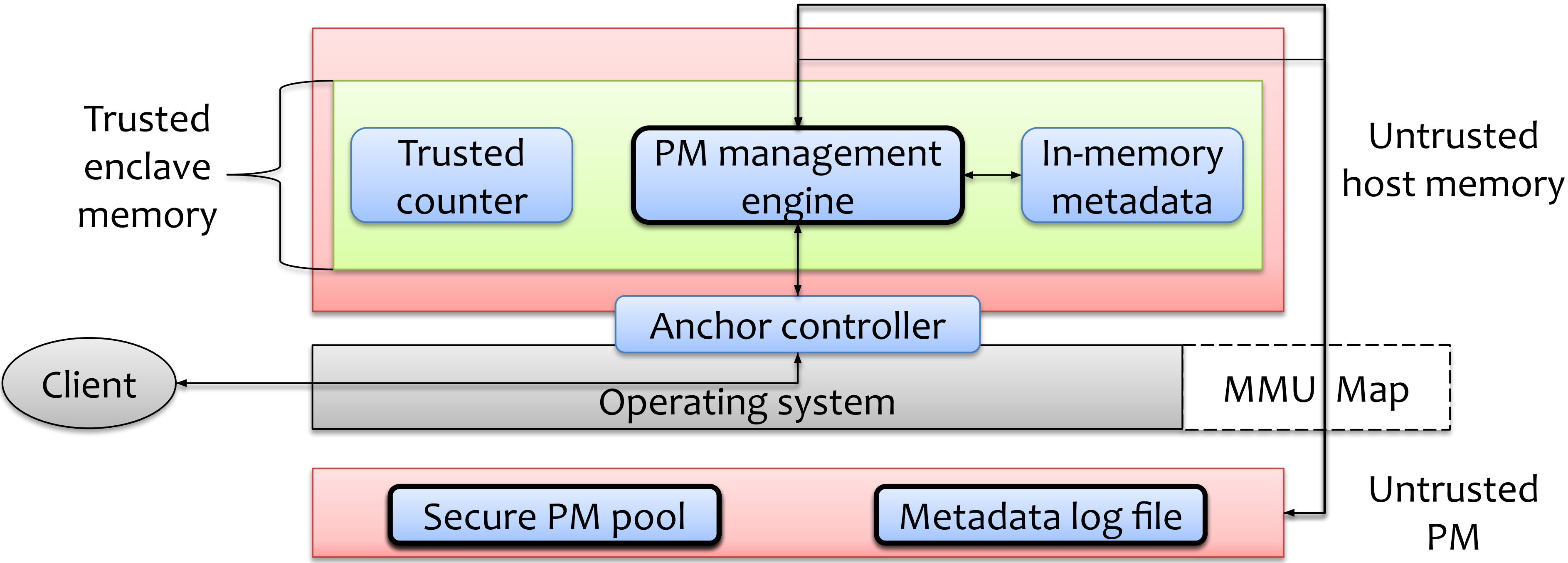
System operations - Write (embedded animations)

4. I[illegible]ion



System operations - Recovery (embedded animations)

5. Ap [illegible] PM


Trusted enclave memory
Trusted counter
PM management engine
In-memory metadata
Untrusted host memory
Anchor controller
Client
Operating system
MMU
Map
Secure PM pool
Metadata log file
Untrusted PM