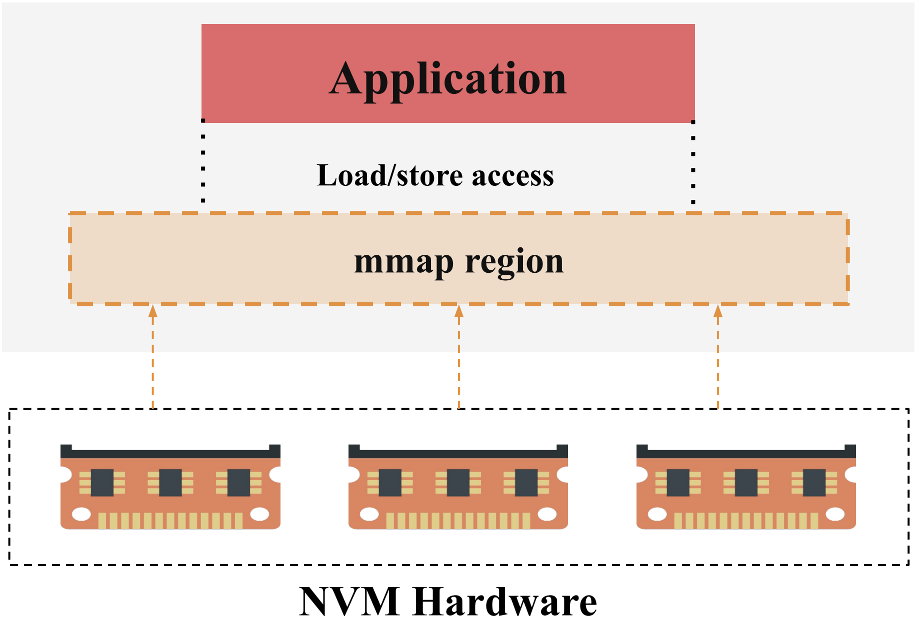
TENET: Memory Safe and Fault Tolerant Persistent Transactional Memory

R. Madhava Krishnan, Diyu Zhou, Wook-Hee Kim, Sudarsun Kannan, Sanidhya Kashyap, Changwoo Min

Boon and bane of Non-volatile Memory (NVM)

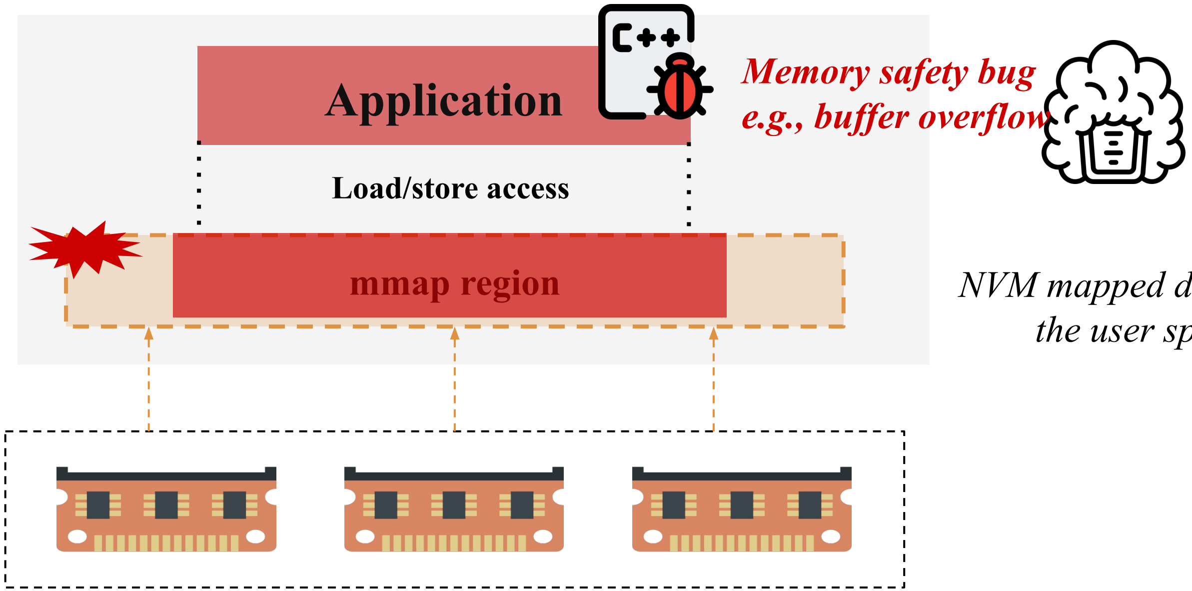
- **Byte-addressability enables application to directly access NVM using load/store instructions**
  - NVM is directly mapped to the application's address space



NVM mapped directly to the user space

Boon and **bane** of Non-volatile Memory (NVM)

Byte-addressability makes NVM data vulnerable to memory safety bugs in the application



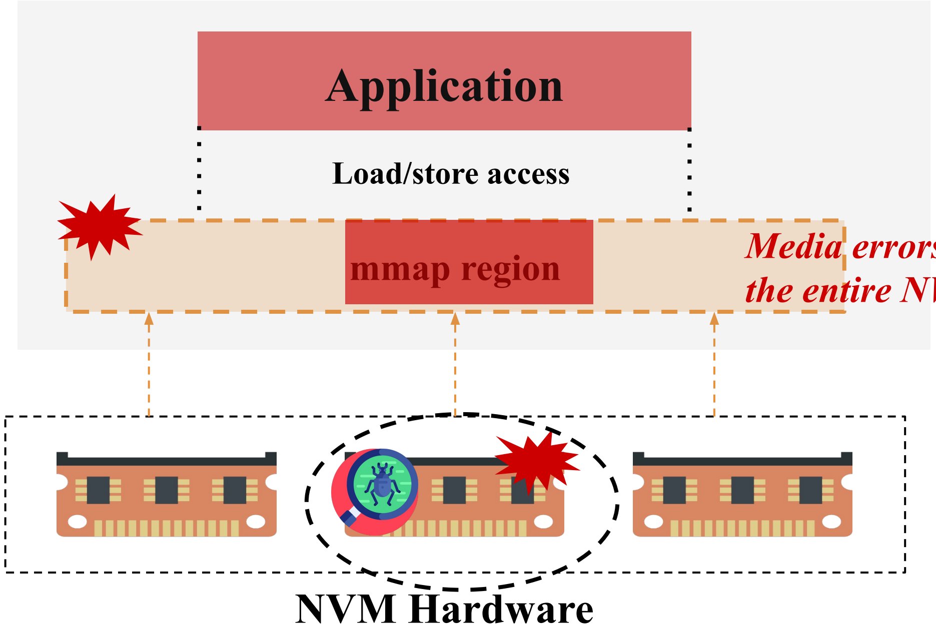
NVM mapped directly to the user space

NVM Hardware

Hardware (Media) errors are a threat too!

- **NVM data is vulnerable to Media Errors**
  - Device wear-out, power spikes, soft media faults etc

NVM mapped directly to the user space



Media errors corrupts the NVM data and the entire NVM page (data) is lost

Research problem that we tackle..

How to detect memory safety bugs in the application and prevent it from corrupting the NVM data?

How to prevent data loss due to the NVM media errors?

TENET

Talk Outline

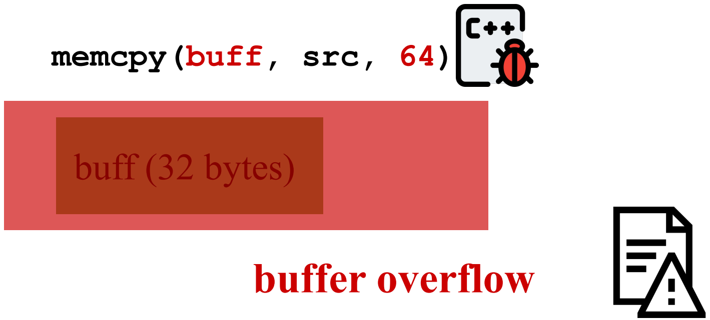
- **Background NVM memory safety errors**
- TENET Overview
- TENET Design
- Evaluation
- Conclusion

Background on types of memory safety violations

01 **Memory Safety Violations**

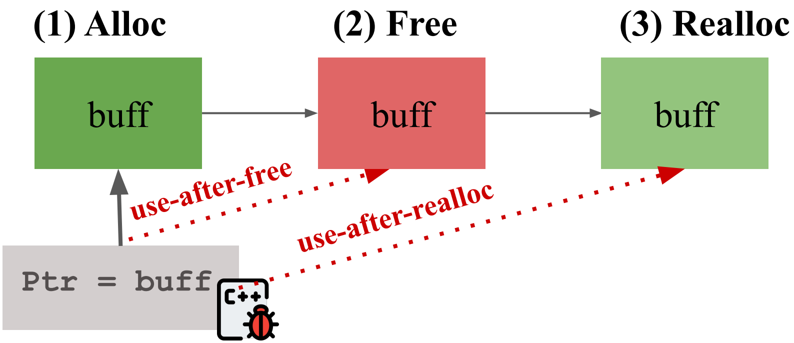
- Spatial Safety Violations
- Temporal Safety Violations

Spatial Safety Violations



Spatial safety violations happens when applications access the memory ***beyond the allocated range***

Temporal Safety Violations



Temporal safety violations happens when applications access the memory ***using dangling pointers***

VIRGINIA TECH.

Background on types of **Media Errors**

02 **NVM Media Errors**

- Correctable Media Errors
- **Uncorrectable Media Errors**

- NVM has high Random Bit Error Rate (RBER) ~= NAND flash
- **Uncorrectable media errors (UME) are detected by the hardware ECC but can not be corrected**
    - UME can happen at random offset and the OS kernel offlines the corrupted NVM page
    - **Application is responsible for fixing the corrupted NVM page**

Applications are required to maintain a backup of NVM data to rollback the affected NVM page to prevent data loss

Summary of prior Persistent Transactional Memory (PTM) works

PTM	Baseline PTM*	Spatial Safety	Temporal Safety	Fault Tolerance	Performance Overhead	NVM Cost Overhead
Libpmemobj-**R**	libpmemobj	✗	✗	✓	100%	High
SafePM [Eurosys-22]	libpmemobj	✓	✓	✗	55%	
Pangolin [ATC-19]	libpmemobj	✓	✗	✓	67%	Moderate

Guaranteeing memory safety and fault tolerance at a lower performance overhead and cost is a very challenging problem

*Performance overhead reported for hash table
*Pangolin's overhead is directly referenced from the paper

Talk Outline

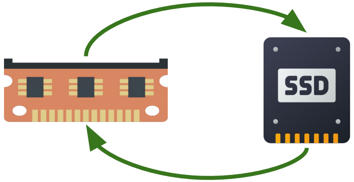
- Background NVM memory safety errors
- **TENET Overview**
- TENET Design
- Evaluation
- Conclusion

TENET overview: **Goals and Assumptions**

TENET is a NVM programming framework to develop memory safe and fault tolerant NVM data structures and applications

- Protect NVM data from a buggy application code
  - **Guarantee spatial safety and temporal safety**

- Protect NVM data against Uncorrectable Media Errors (UME)
  - **Guarantee a performance and cost efficient fault tolerance**



- Adversarial attacks are out-of-scope
- TENET library code and OS kernel are trusted (TCB)

TENET overview: **Programming Model**

TENET provides persistent transaction programming model

- TENET uses TimeStone persistent transactional memory (PTM)
- TimeStone is the state-of-the-art high-performing, highly scalable PTM
- **TimeStone does not provide memory safety or fault tolerance**

Session 4B: Speculation and consistency — Brain teasers. ASPLOS'20, March 16–20, 2020, Lausanne, Switzerland

Durable Transactional Memory Can Scale with TimeStone

Talk Outline

- Background NVM memory safety errors
- TENET Overview
- **TENET Design**
- Evaluation
- Conclusion

Spatial safety design in TENET

*Application code or any code outside the TENET library is **not allowed** to perform direct NVM writes*

*Only the TENET library code **is allowed** to perform writes to the NVM data*

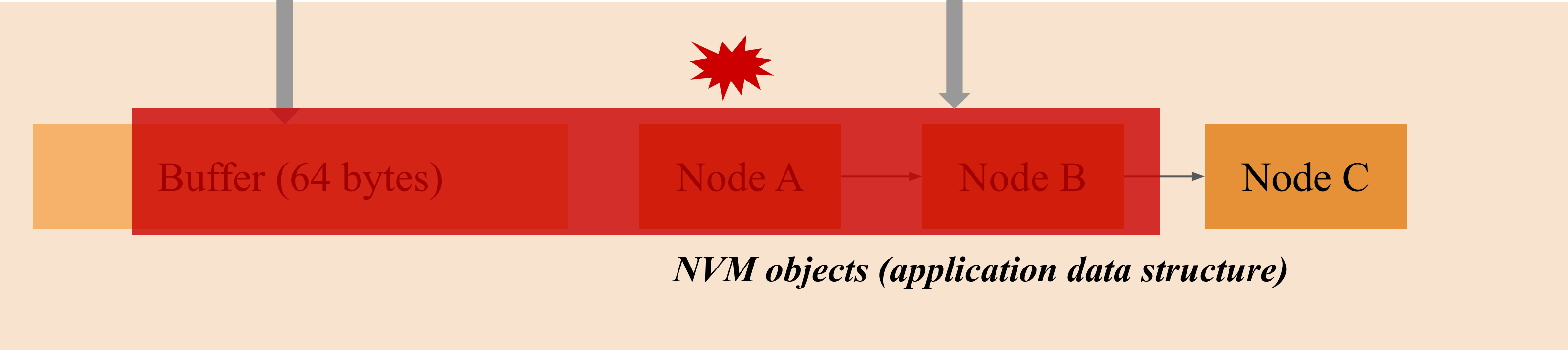
Direct NVM writes in the application code is dangerous

A buggy application write on the NVM can cause spatial safety violation

```
Application code:
memcpy(buffer, src, 128)
```

Readers access the corrupted data

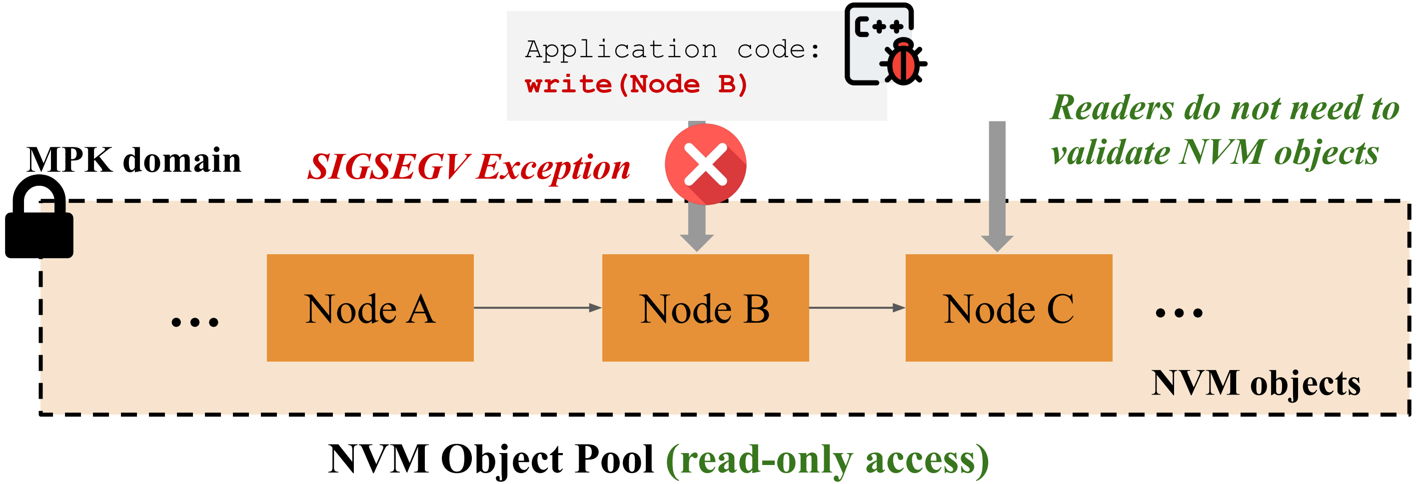
NVM Pool



*NVM is **read-only** for the application code to prevent buggy writes from corrupting the NVM data*

Prevent direct NVM writes using Memory Protection Keys (MPK)

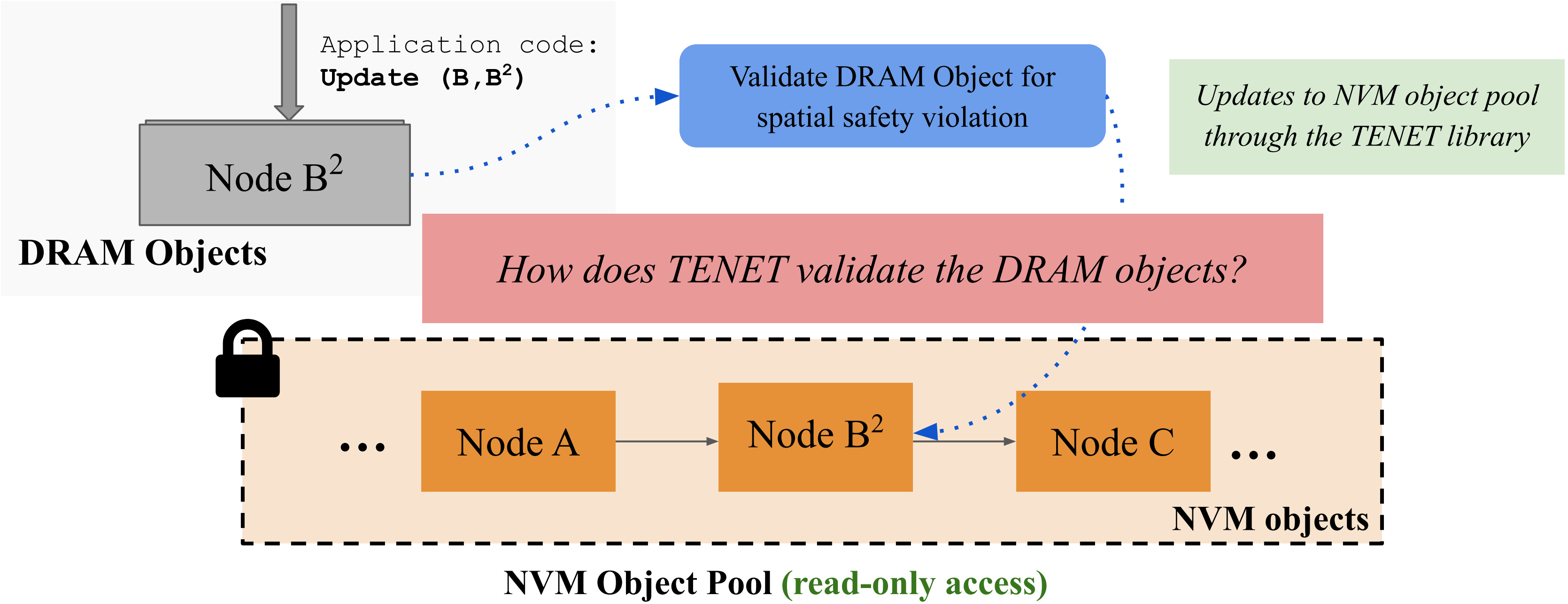
TENET uses MPK to enforce read-only access to the NVM object pool for all the code outside of the TENET library



How does application writes to the NVM objects?

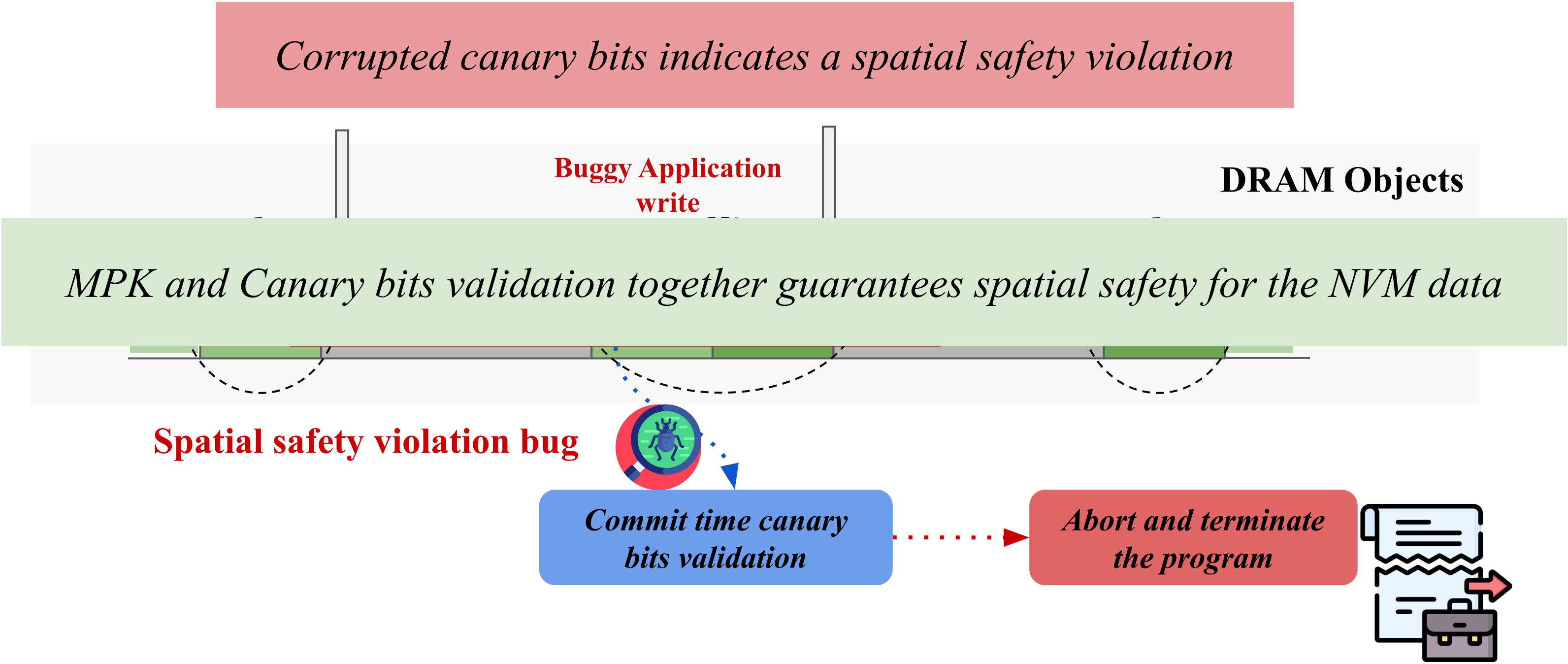
Prevent direct NVM writes using Memory Protection Keys (MPK)

Application writes only on the DRAM region and TENET writes back the DRAM object to the NVM after validating it for spatial safety



Protecting DRAM objects using canary bits

- TENET assigns **8 byte canaries** at the boundary of a DRAM object at the time of its creation
- Canary bits are inspected when the application **commits its transaction**

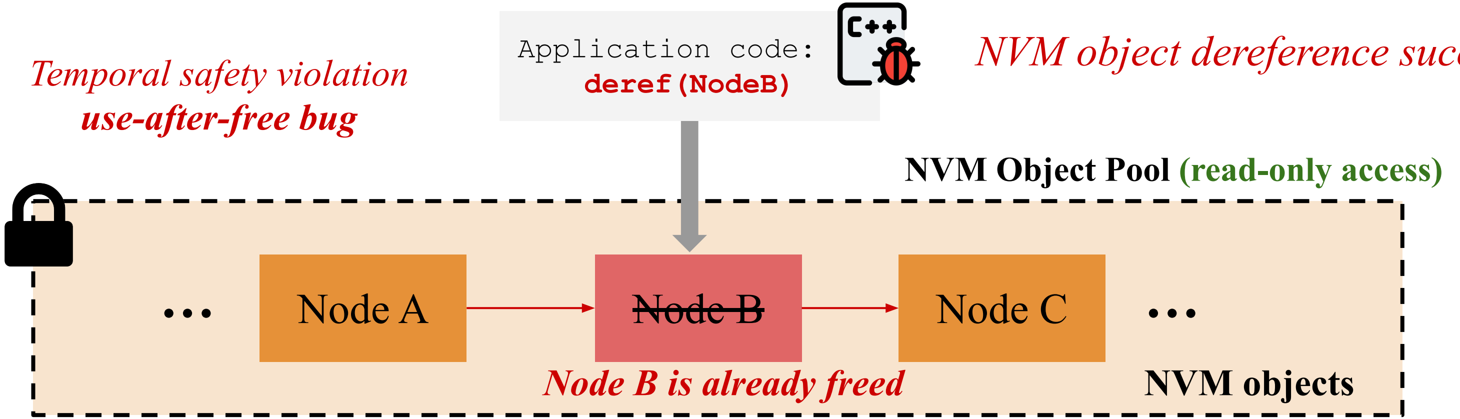


Read-only NVM access can cause temporal safety violations

Does making NVM read-only solve all the problems and prevent NVM data corruption?

Temporal safety violation
use-after-free bug

NVM object dereference succeeds



How does TENET enforce temporal memory safety for the NVM objects?

Enforcing temporal safety for NVM objects using pointer tags

*NVM address is **tagged** at the time of creation; the tag is stored in the allocated NVM object and a copy of the tag is encoded in the upper 16 bits of the NVM pointer*

The encoded pointer to Node B is stored in Node A

Node A
tag = 0xFACE

Encoded pointer →

Node B
tag = 0xCAFE

tag is stored in the NVM object at the time of creation

Upper 16 bits are unused

- **Node B's address** → 0x00001265FFCAB734; **Tag** → 0xCAFE
- **Encoded pointer** → Node B || Tag << 48 → **0xCAFE1265FFCAB734**

tag bits | **NVM address**

A copy of the tag is encoded to the upper 16 bits of Node B's address

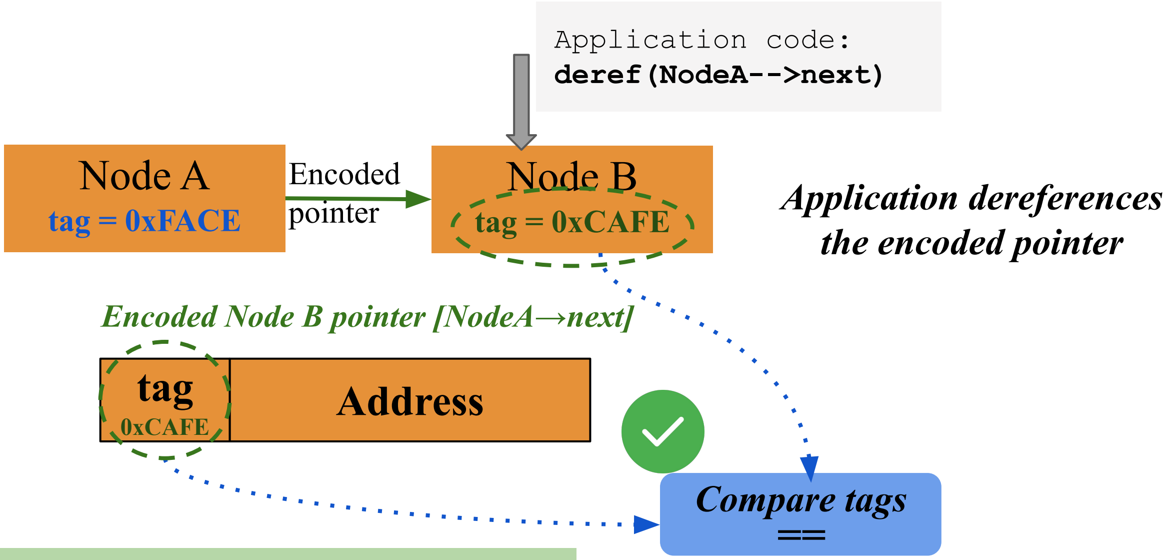
Encoded pointer layout

63 48 0

tag 0xCAFE	**Address**

Enforcing temporal safety for NVM objects using pointer tags

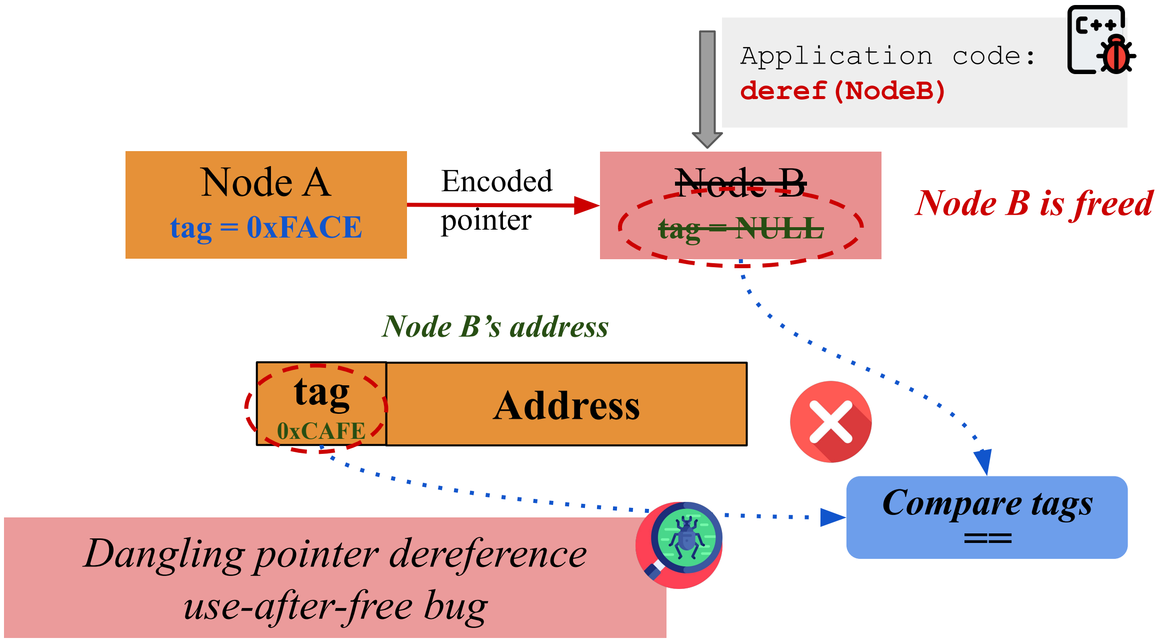
Application accesses the NVM objects using the encoded pointer -- the encoded tag in the pointer is compared with the tag stored in the corresponding NVM object



Valid NVM pointer dereference

Enforcing temporal safety for NVM objects using pointer tags

Dangling pointer is detected by comparing the tag stored in the NVM object with the tag encoded in the pointer to the NVM object



Replicating NVM data for fault tolerance against UME

- **NVM data corruption due to software errors**
  - Spatial memory safety → MPK + canary bits validation ✓
  - Temporal memory safety → Pointer tags validation ✓

How does TENET make the NVM data fault tolerance against the UME?

- TENET replicates the NVM data to the local SSD to maintain backup copy
- **Restore the corrupted NVM page from the SSD replica**
- TENET's replication provides many desirable properties
  - **Cost efficiency** → replicating to the local SSD
  - **Performance efficiency** → replicating the data out-of-the critical path
  - **Consistent loss-less recovery**

Refer to the paper for more details

Talk Outline

- Background NVM memory safety errors
- TENET and TimeStone Overview
- TENET Design
- **Evaluation**
- Conclusion

Evaluation of TENET

Evaluation Questions

- How does TENET compare against the prior PTM works in terms of features and performance overhead?
- How much overhead does TENET incurs over its baseline PTM system TimeStone?

Evaluation Settings

- We use a 2 socket server with 64 core Intel Xeon Gold CPU
  - 64GB DRAM, 512GB NVM, 1TB SSD
- We evaluate two different versions of TENET
  - **TENET-MS → supports only memory safety**
  - **TENET → supports memory safety and fault tolerance**
- We evaluate TENET with different data structures for different read/write ratios
  - YCSB workloads and microbenchmarks

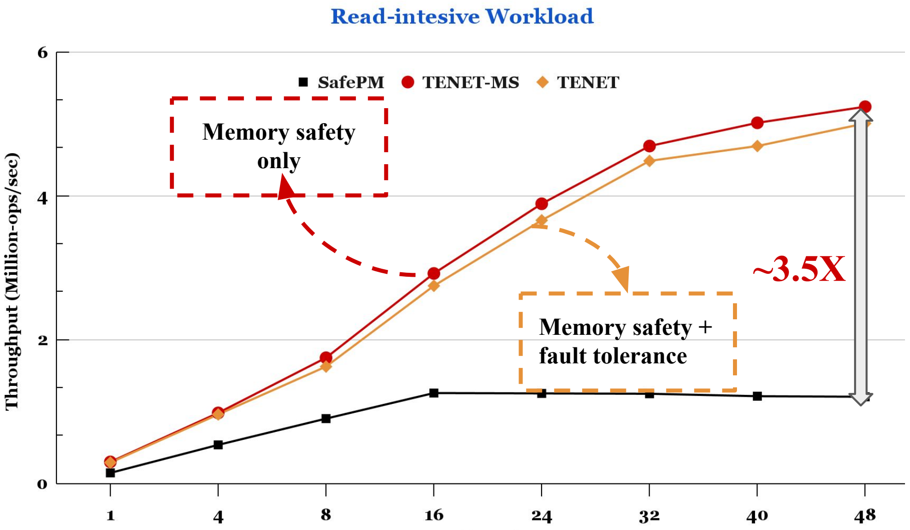
Comparison of TENET with the other PTMs

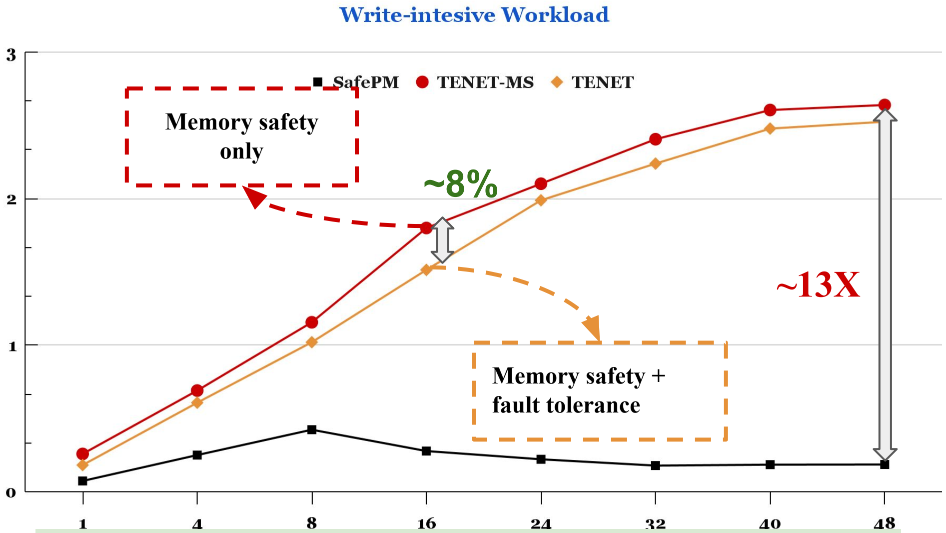
PTM	Baseline PTM*	Spatial safety	Temporal Safety	Fault tolerance	
Libpmemobj-**R**	libpmemobj	✗	✗	✓	Replicates NVM data to a local NVM pool
SafePM [Eurosys-22]	libpmemobj	✓	✓	✗	Adds address sanitizer (Asan) to the libpmemobj
Pangolin [ATC-19]	libpmemobj	✓	✗	✓	Supports parity based replication and object checksums
TENET	TimeStone	✓	✓	✓	

TENET is the only PTM to provide spatial memory safety, temporal memory safety, and fault tolerance for the NVM data

*PTM - persistent transactional memory
*Libpmemobj is a transactional library in the PMDK

Performance of TENET and SafePM for a concurrent hash table



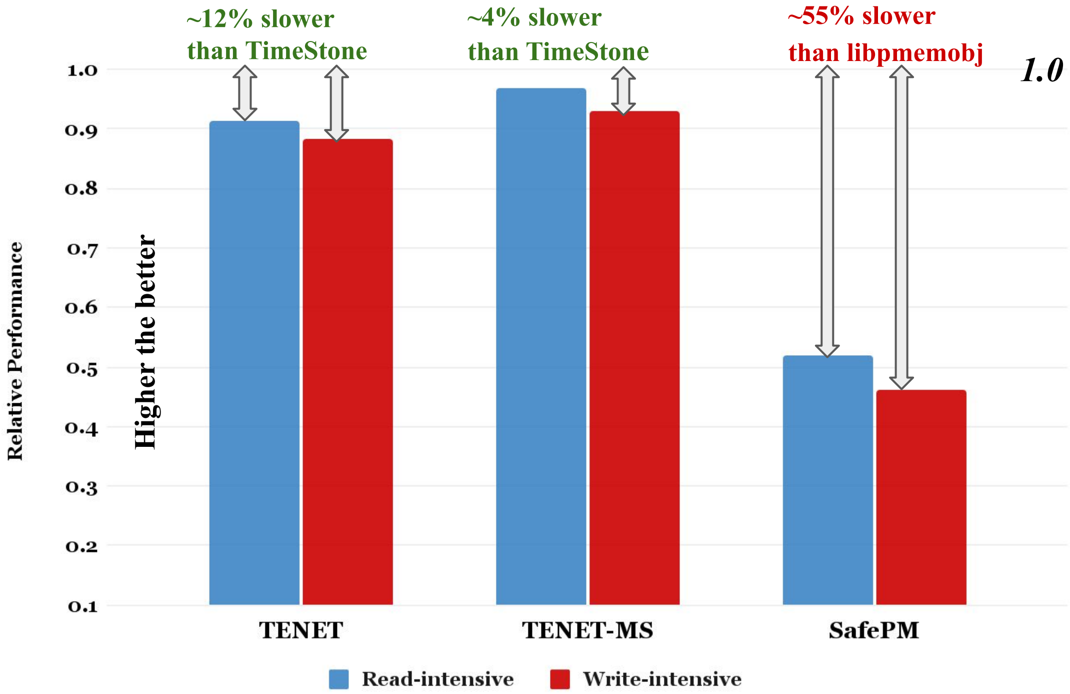


TENET is up to 13x faster than SafePM

Adding replication (TENET) incurs only a 8% overhead

For a fair comparison lets compare the relative performance slowdown against their respective baseline PTM

Performance of TENET and SafePM for a concurrent hash table



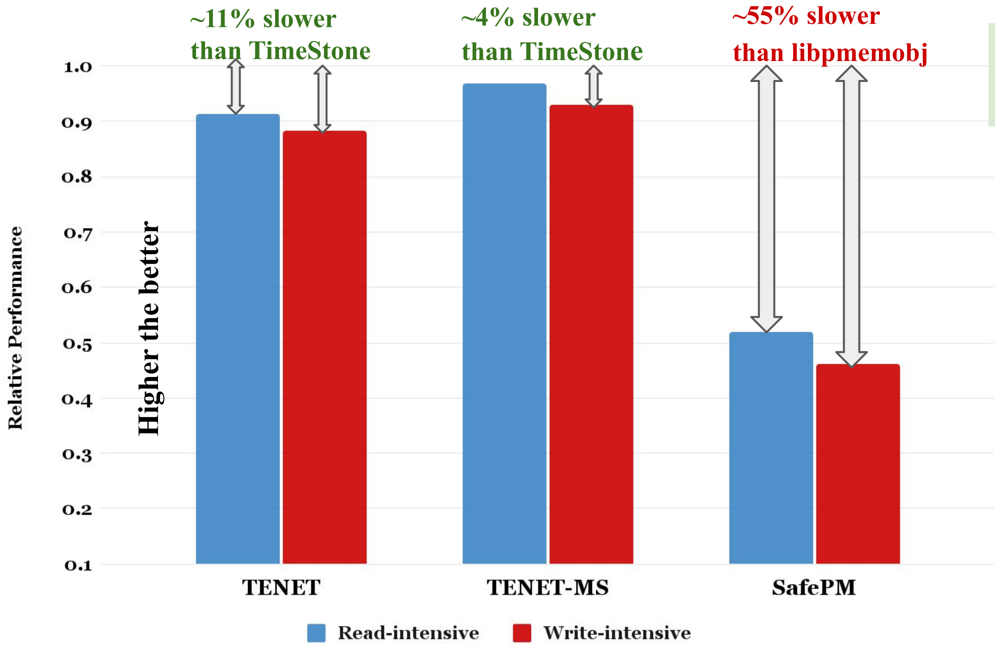
1.0 → is the maximum possible performance

SafePM incurs ~55% performance overhead for only memory safety guarantee

TENET adds ~12% performance overhead for memory safety and fault tolerance

- Performance is normalized to their respective baseline PTMs
  - **SafePM normalized to the libpmemobj → throughput (safePM)/throughput (libpmemobj)**
  - **TENET normalized to the TimeStone → throughput (TENET)/throughput (TimeStone)**

Performance of TENET and SafePM for a concurrent hash table



- Performance is normalized to their respective baseline PTMs
  - **SafePM normalized to the libpmemobj**
  - **TENET normalized to the TimeStone**

TENET does not require additional crash consistency operations for its memory safety metadata

- MPK → hardware primitive
- Pointer tags → embedded directly into the object

TENET does not perform memory safety validation for every NVM access

- Spatial safety checks performed only at the commit time
- Temporal safety checks performed only at the first-dereference of an NVM object

Refer to the paper for more details on these optimizations

Other interesting insights in the paper

- How the spatial, temporal safety, and fault tolerance techniques works in tandem?
- How TENET leverages concurrency properties of PTM (ACID properties) for performance efficiency?
- How TENET leverages RCU style grace period to guarantee a consistent recovery?
- Array interface design for guaranteeing spatial and temporal safety
- How can the TENET's techniques be applied to the other PTM systems?
- More evaluations and in-depth analysis on TENET's design

Conclusion

NVM is vulnerable to data corruption due to software bugs and media errors

- NVM is exposed to the user space thus it is vulnerable to spatial and temporal memory safety violations

TENET a NVM programming framework to design memory safe and fault tolerant NVM data structures and applications

- **Spatial memory safety** → Memory protection keys (MPK) + Canary bits validation
- **Temporal memory safety** → Encoded pointer tag validation during dereference
- TENET guarantees **fault tolerance** for NVM data against uncorrectable media errors (UME)
  - Replicates the NVM objects to the local SSD
- **TENET guarantees a robust memory protection and fault tolerance at a modest performance overhead**

Thank You!