SafePM

A Sanitizer for Persistent Memory

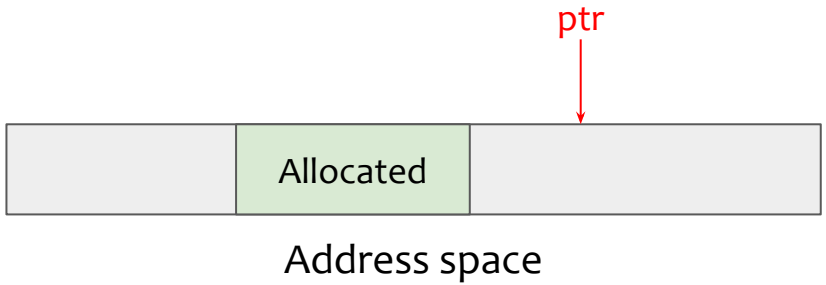
Kartal Kaan Bozdoğan, **Dimitrios Stavrakakis**,
Shady Issa, Pramod Bhatotia

ACM EuroSys 2022

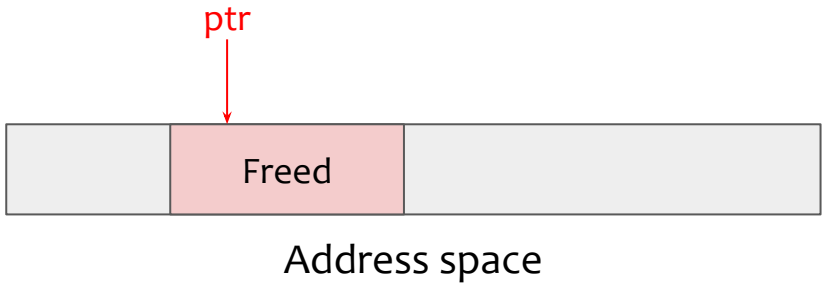
Memory safety

Memory safety violations : Illegal accesses to unintended memory regions

Spatial memory safety
e.g., buffer overflow, stack overflow



Temporal memory safety
e.g., dangling pointer, double free



Memory safety in practice

Prevalent in almost all low-level unsafe C/C++ code

Chromium project [1]

- 70% of vulnerabilities are memory safety problems

Microsoft [2]

- 70% of vulnerabilities fixed in security patches are memory safety violations

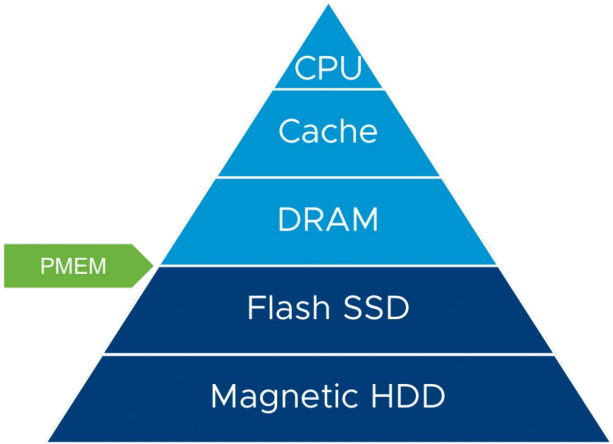
Android [3]

- 75% of vulnerabilities are memory safety issues

[1] Chromium project: https://www.chromium.org/Home/chromium-security/memory-safety
[2] Microsoft: https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code/
[3] Android: https://security.googleblog.com/2019/05/queue-hardening-enhancements.html

Persistent memory (PM)

Persistent memory management is susceptible to memory safety vulnerabilities



- Persistent memory programming model
- Durability & crash consistency
- Recovery code paths

Memory safety approaches for volatile memory are insufficient for PM

SafePM: A sanitizer for persistent memory

Memory safety mechanism for PM-based applications

System properties:

- Spatial & temporal memory safety
- Transparency
- High coverage
- Crash consistency

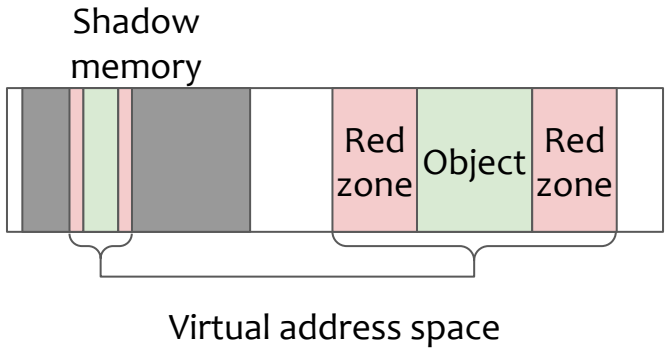
Outline

- ~~Motivation~~
- Design
  - Overview
  - Persistent memory operations
- Implementation
- Evaluation

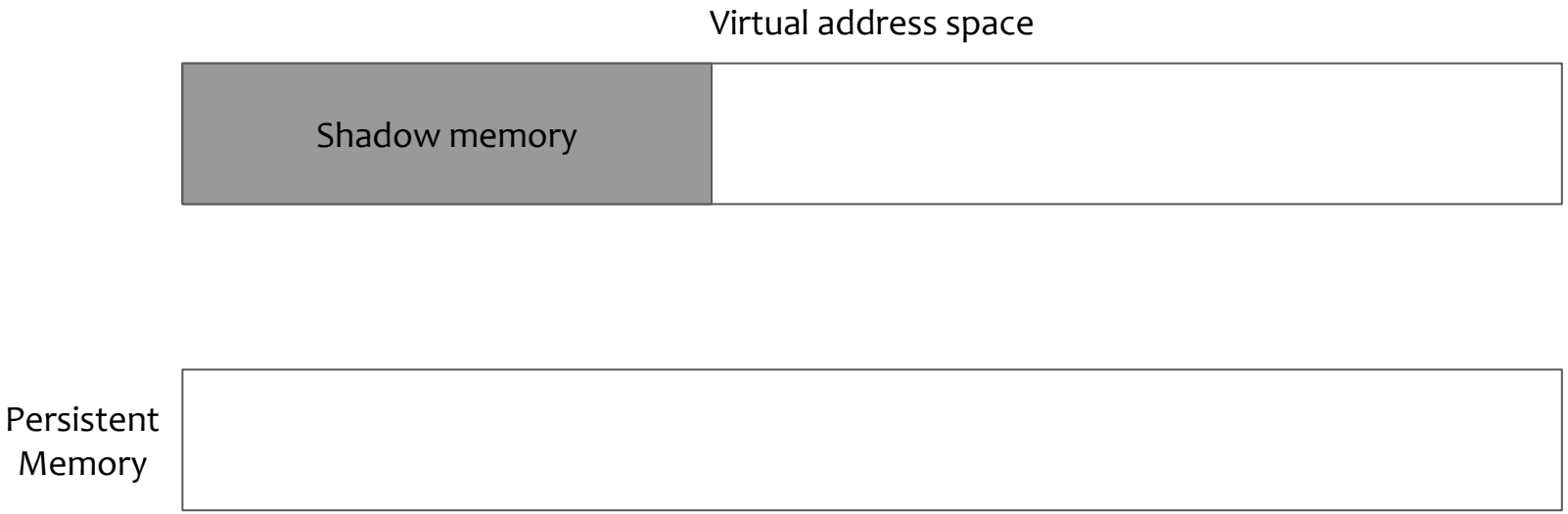
Design overview

SafePM enforces a shadow memory-based approach for memory safety

- Shadow memory
- Red zones
- Runtime checks

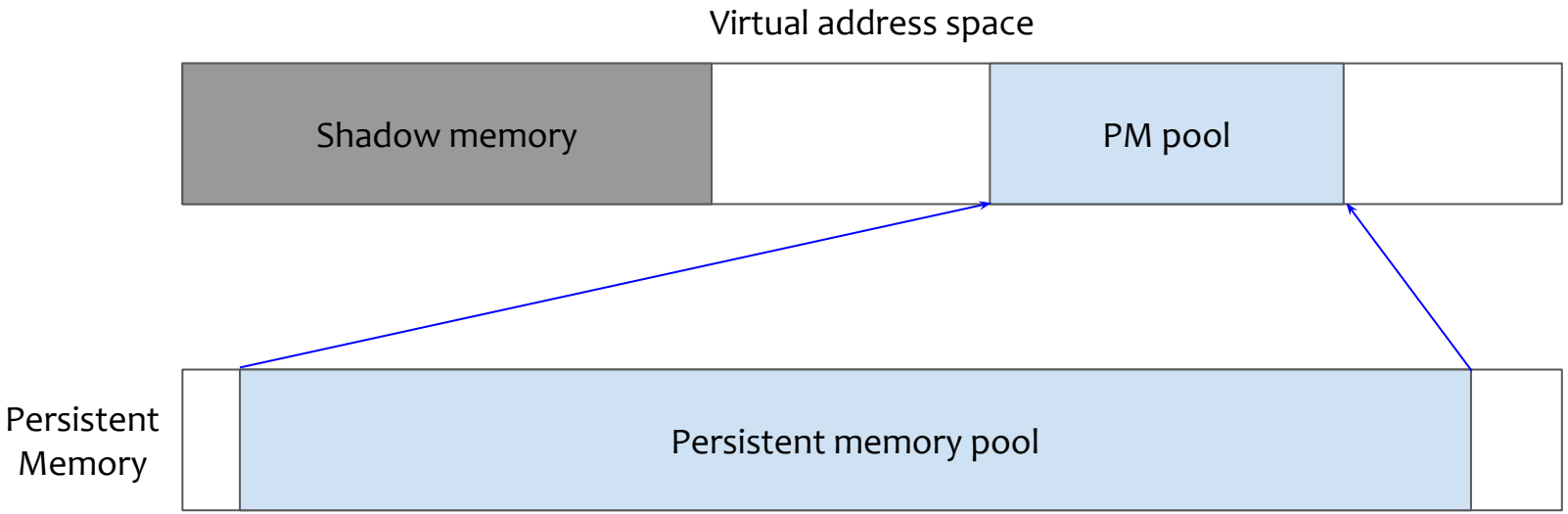


Design overview



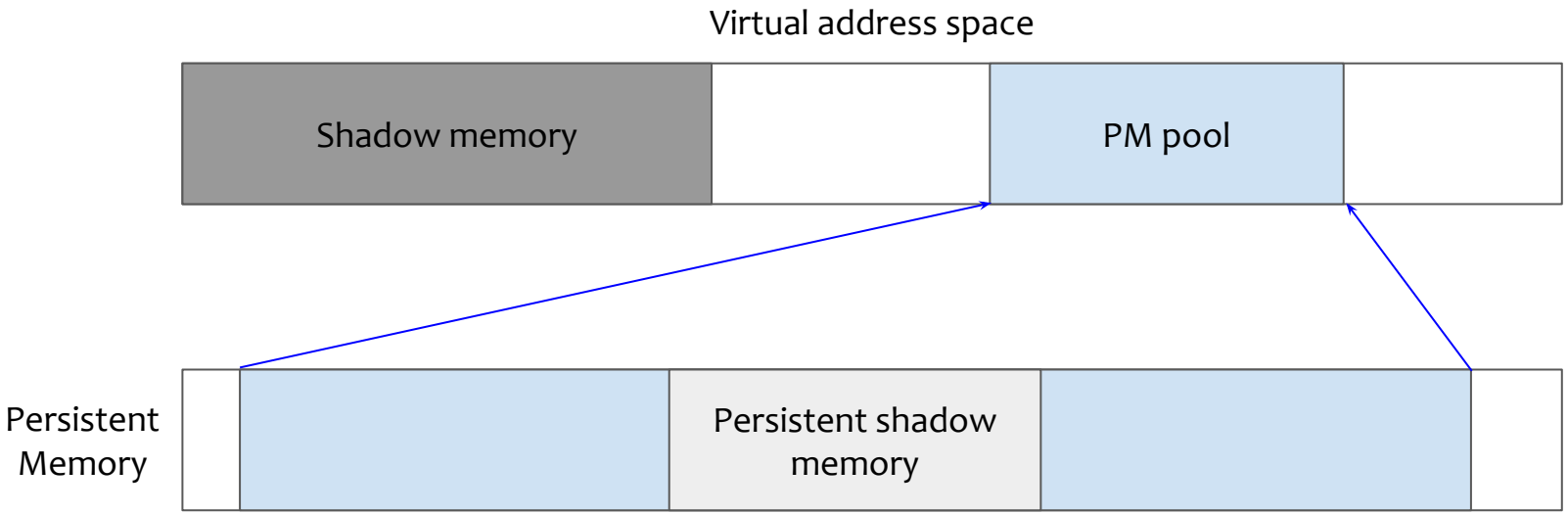
The shadow memory is reserved by SafePM for metadata about each memory region's state

Design overview



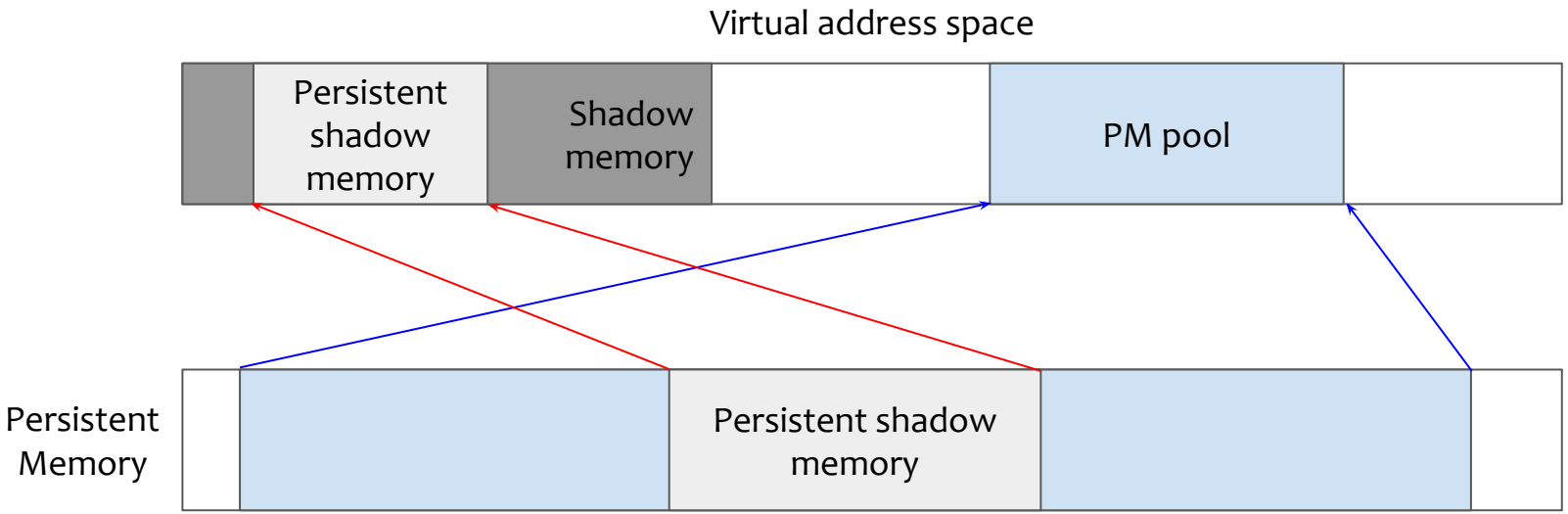
Persistent memory pools are directly mapped to the virtual address space

Design overview



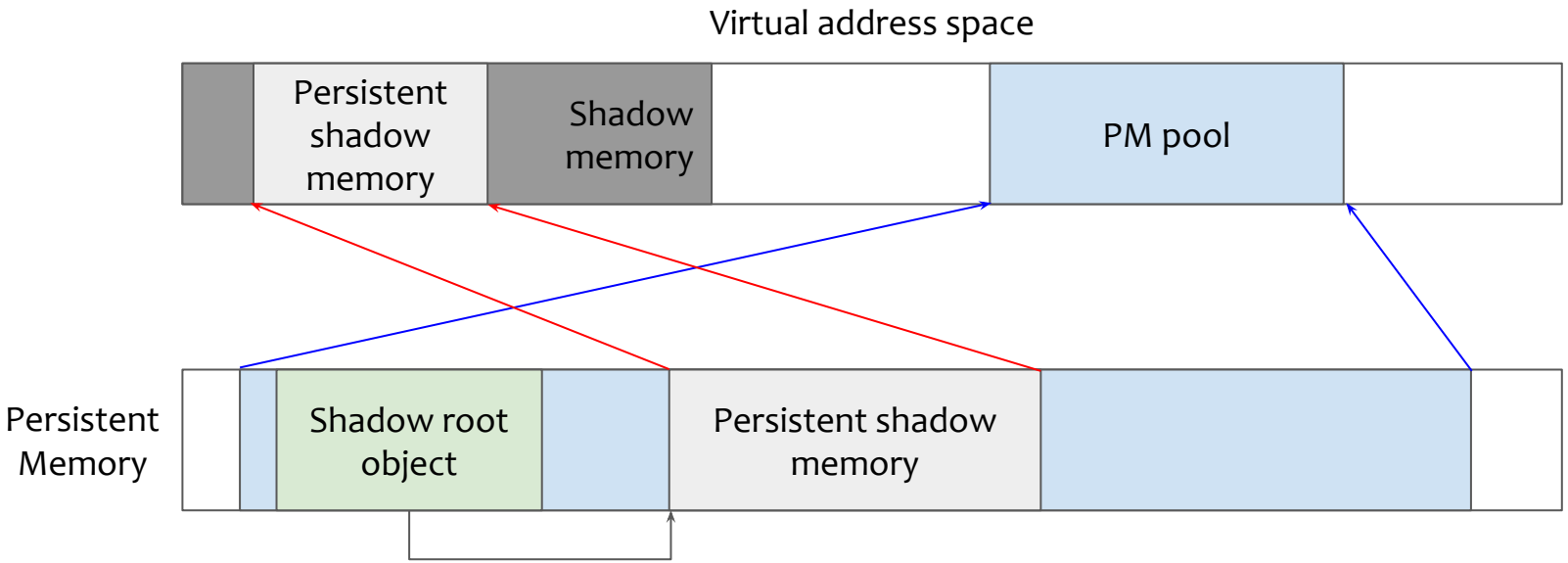
SafePM reserves a part of the PM pool heap for the **Persistent Shadow Memory** (PSM)

Design overview



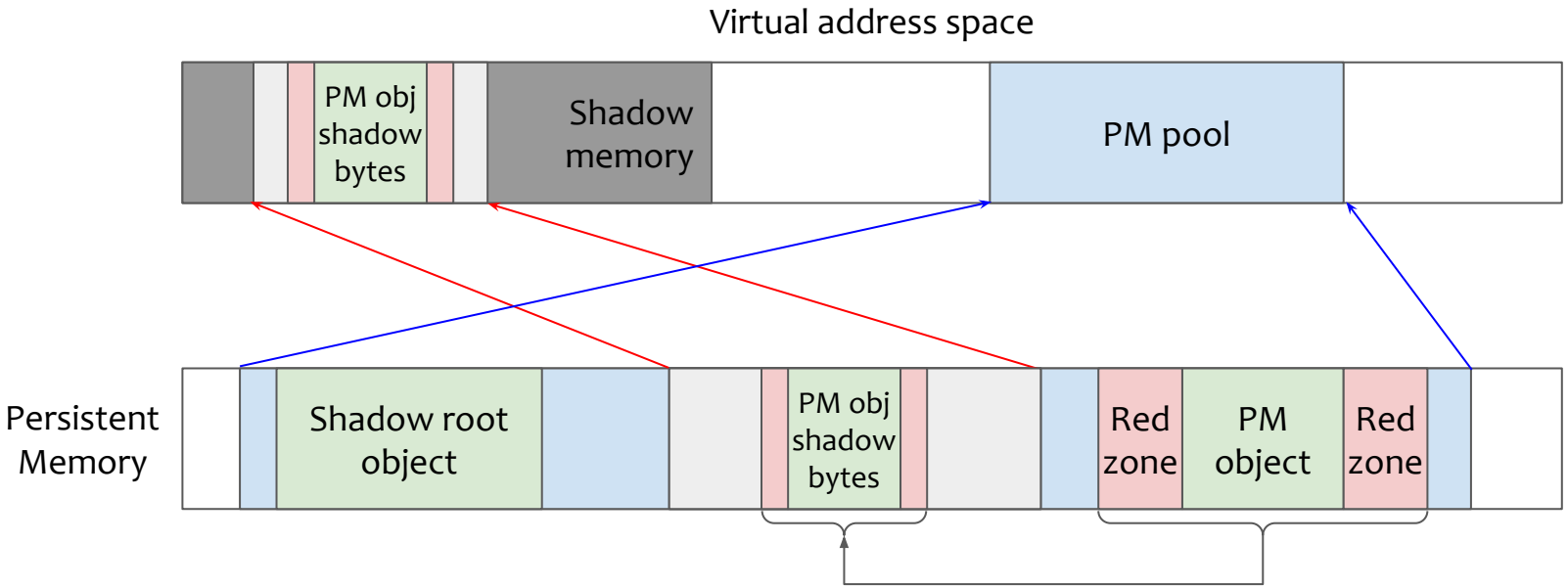
SafePM maps the pool's **PSM** over its corresponding location of the shadow memory space

Design overview



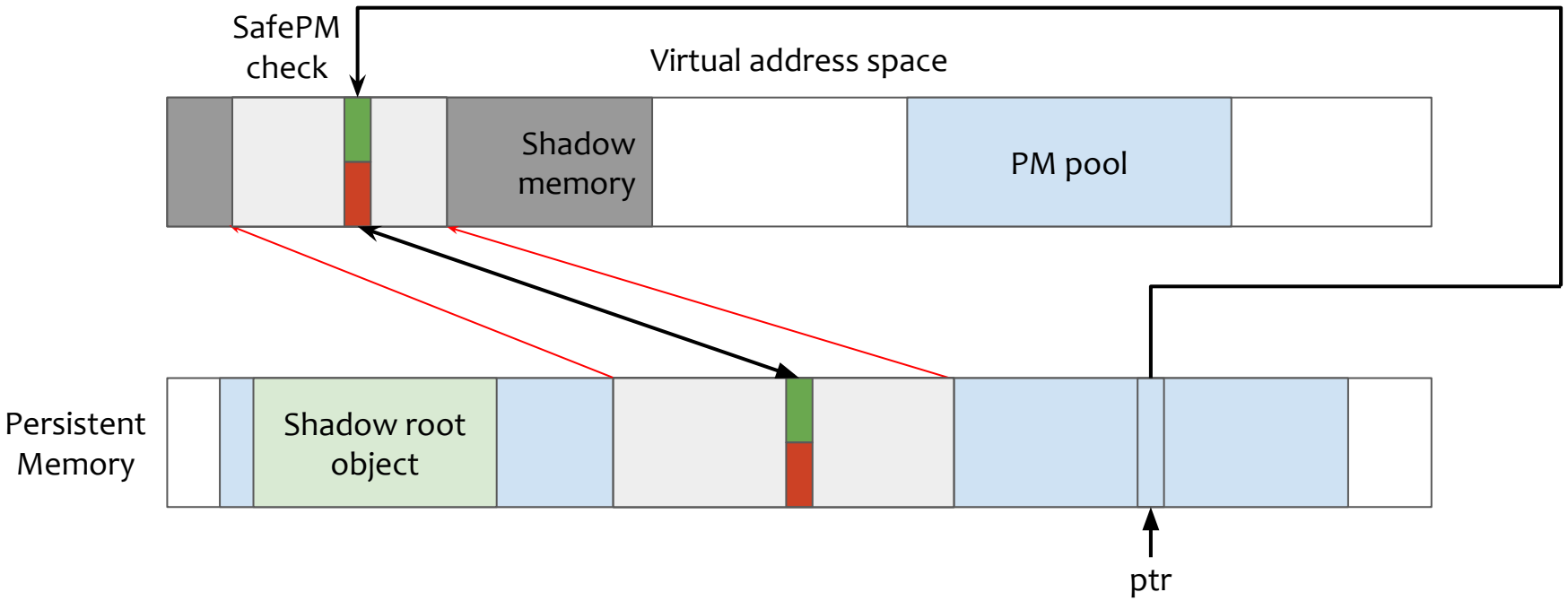
The shadow root object maintains consistent reference to the **PSM** across runs

Persistent memory allocation



SafePM allocates the object along with its red zones and updates the **PSM**

Persistent memory access



On a memory access SafePM checks against the corresponding shadow memory bytes

Outline

- ~~Motivation~~
- ~~Design~~
- Implementation
- Evaluation

Implementation

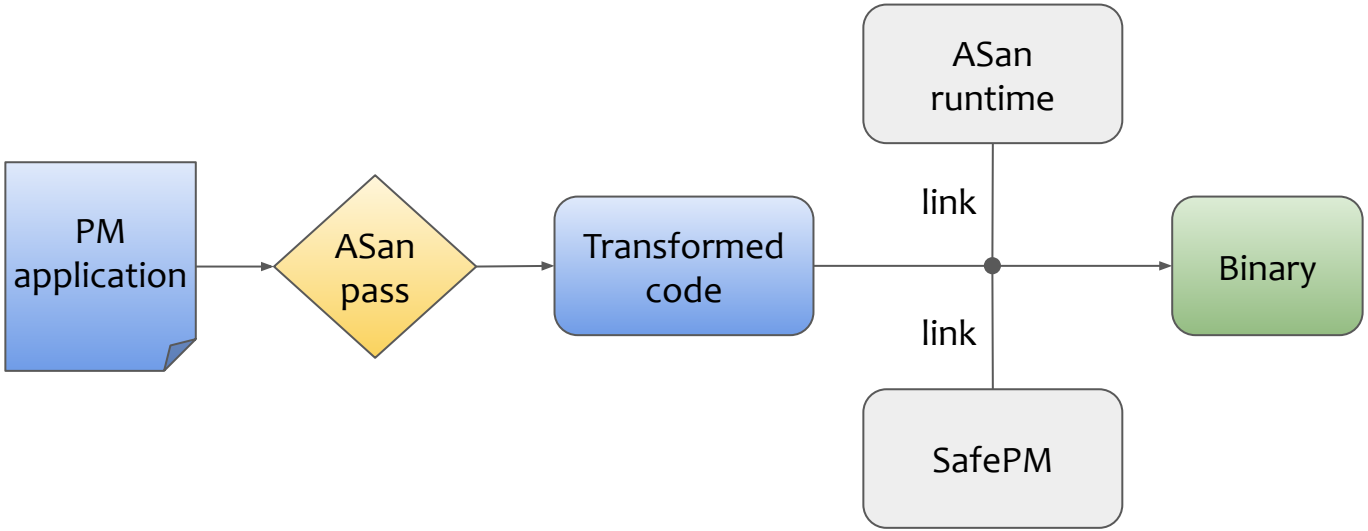
SafePM is built on PMDK[1] and ASan[2]

- Overmap of persistent shadow memory
- Compiler pass of ASan – intact!
- PMDK programming model – transparent support!
- Crash consistency via PMDK transactions

[1]Persistent memory development kit (PMDK): https://github.com/pmem/pmdk

[2]Address Sanitizer (ASan): https://www.usenix.org/system/files/conference/atc12/atc12-final39.pdf

SafePM hardening workflow



The resulting transformed code is linked against SafePM and ASan runtime library

Outline

- ~~Motivation~~
- ~~Design~~
- ~~Implementation~~
- Evaluation

Evaluation

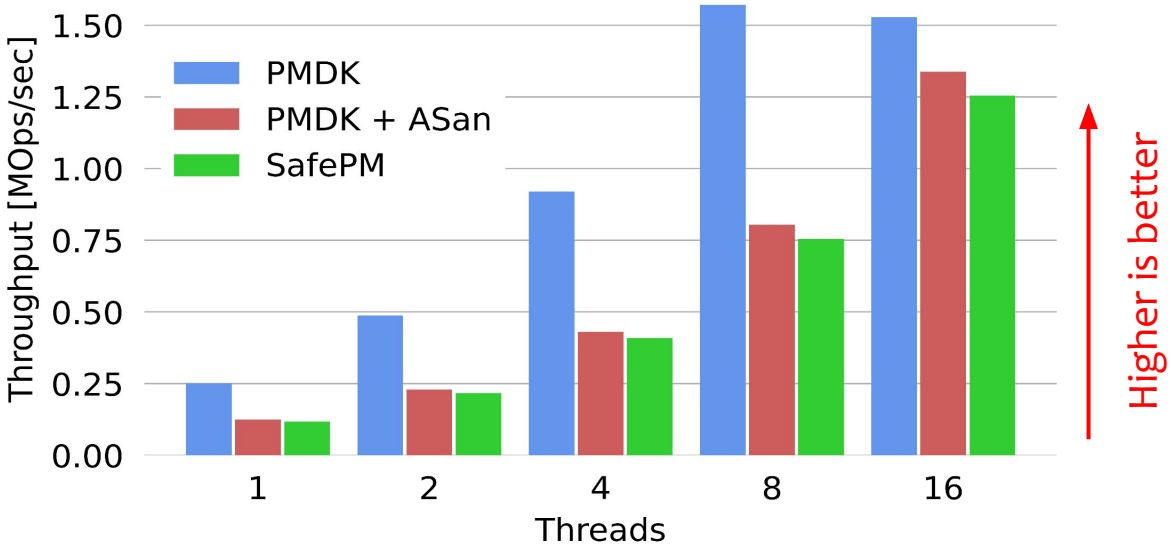
- What is the performance overhead of SafePM?
  - Persistent memory KV store (pmemkv)
- How much space overhead does SafePM introduce?
  - Persistent indices (ctree, rtree, rbtree, hashmap)
- How robust is SafePM in detecting memory safety vulnerabilities?
  - RIPE benchmark framework

Evaluation

- Experimental setup:
  - Intel Xeon Gold 6212U CPU (2.40 GHz, 24 cores)
  - 192 GB DRAM
  - 768 GB Intel Optane DC DIMMs
- Baselines:
  - PMDK / ASan disabled → no memory safety
  - PMDK / ASan enabled → DRAM only safety

Performance overhead

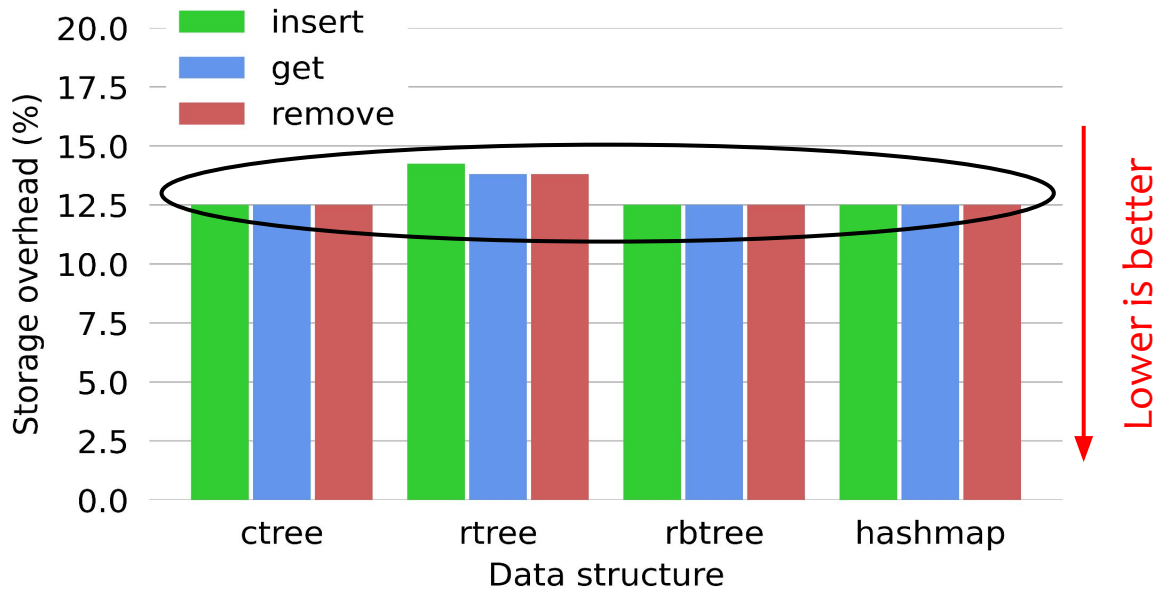
Persistent KV-store benchmark, **10M** ops, **50%** reads / **50%** writes



SafePM incurs similar performance overheads with ASan

Space overhead

Persistent indices, insert/get/remove workloads, relative to PMDK



SafePM increases the required PM space by 12.5% due to the PSM

Efficiency

RIPE benchmark, **1334** memory safety exploits

Variant	Exploitable memory safety bugs
DRAM	**320**
DRAM + ASan	**28**
PM + ASan	**131**
PM + SafePM	**28**

SafePM provides equivalent memory safety effectiveness for PM with ASan

Summary

Current memory safety approaches are not designed for PM applications

- PM programming model
- data/metadata durability & crash consistency
- recovery paths

SafePM:

- comprehensive spatial and temporal memory safety
- no source code modifications
- crash consistency & high coverage

Try it out!

https://github.com/TUM-DSE/safepm

Sources

[1] PM hierarchy image, https://www.starwindsoftware.com/blog/persistent-memory-in-vmware-vsphere-6-7-what-is-it-how-fast-is-it