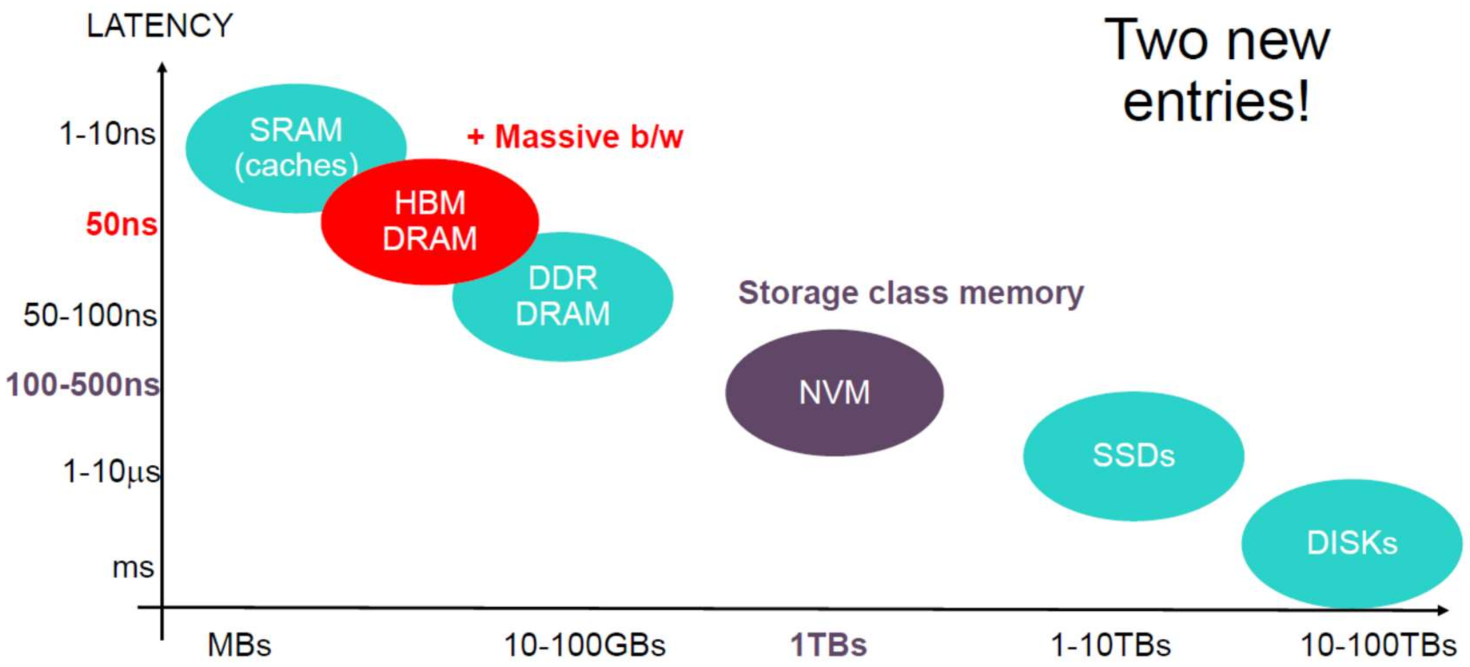
SafeNVM: A Non-volatile Memory Store with Thread-Level Page Protection

Pradeep Kumar, H Howie Huang

The George Washington University

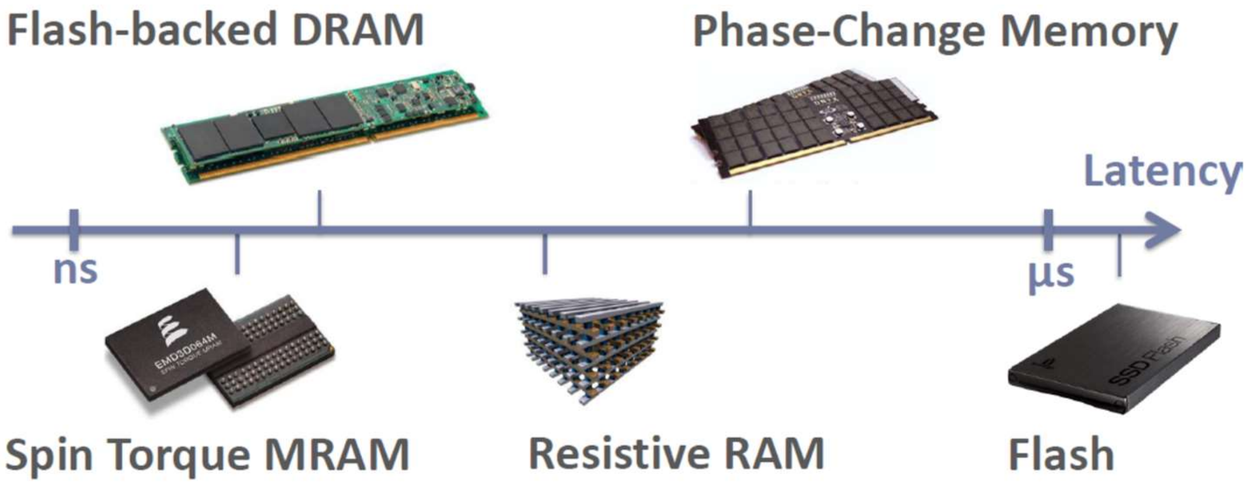
New Members in the Persistent Media



- ➢ NVMs well suited for big data
- ➢ Can ingest high volume of data at very high velocity
- ➢ Others (HPC burst buffer, POSIX file system) likely to benefit

* Keeton, K. “Memory Driven Computing”. FAST’17 Keynote.

Non Volatile Memory (NVM)

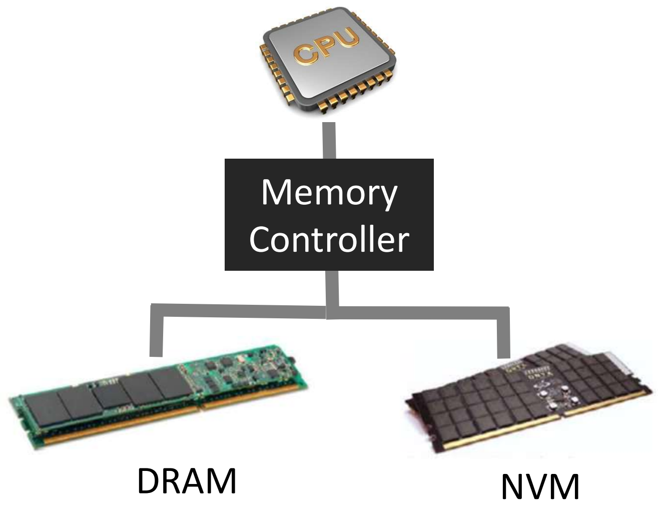


- Persistent
- Byte addressable
- Comparable to DRAM latency
- Denser than DRAM

- A load-store device like DRAM
- Envisioned to be used as storage media
- Lower energy requirement than DRAM
- Standardizing initiatives: GEN Z

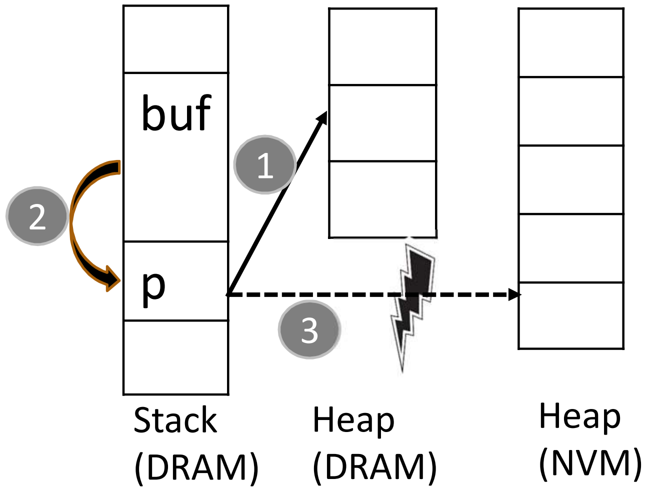
*Haris Volos, et al. "Aerie: Flexible File-System Interfaces to Storage-Class Memory," *Proc. EuroSys 2014*

Persistent Data at Risk



- ➢ Same address space
- ➢ Memory corruption are common
- ➢ Persistent data in NVM at risk

```
  int foo(char** argv) {
      char buf[8]; //Buffer
1     char *p = malloc(sizeof(int));
2     strcpy(buf,argv[1]);
3     *p = magic_num;
      return 0;
  }
```



Related Work

Proposal Name	Description	Issues
Linux mprotect	NVM pages change from read-only to read-write	High overhead due to TLB-Shootdown
PMFS[Eurosys'14]	NVM pages change from read-only to read-write momentarily using CR0.WP	1.Interrupt and context switching are disabled 2.Kernel-mapped only
PMBD[MSST'14]	NVM pages mapped privately during each read-write	1.Interrupt and context switching are disabled 2.Kernel-mapped only 3.Write-window for many threads
Mnemosyne[ASPLOS'11]	User space data store	Data safety is not covered
NV-Heaps[ASPLOS'11]	User space data store	Only a subset are coverted
Write Integrity Testing [IEEE S&P'08]	Allowing pointer modification to points-to-set	1. Memory/CPU overhead 2. No Safety against escaped dangling pointer
🟩 SafeNVM (Proposed)	A Thread momentarily gets write-permission to needed NVM pages	None

Outline

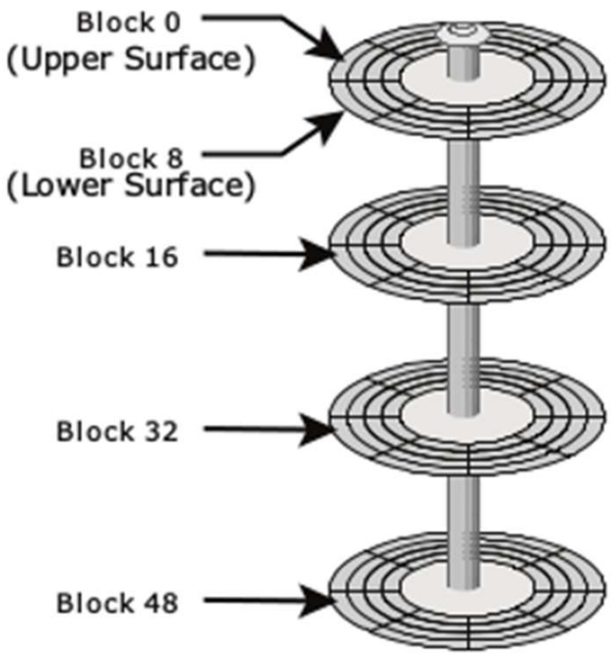
01 – Overview and Problem Statement ✓

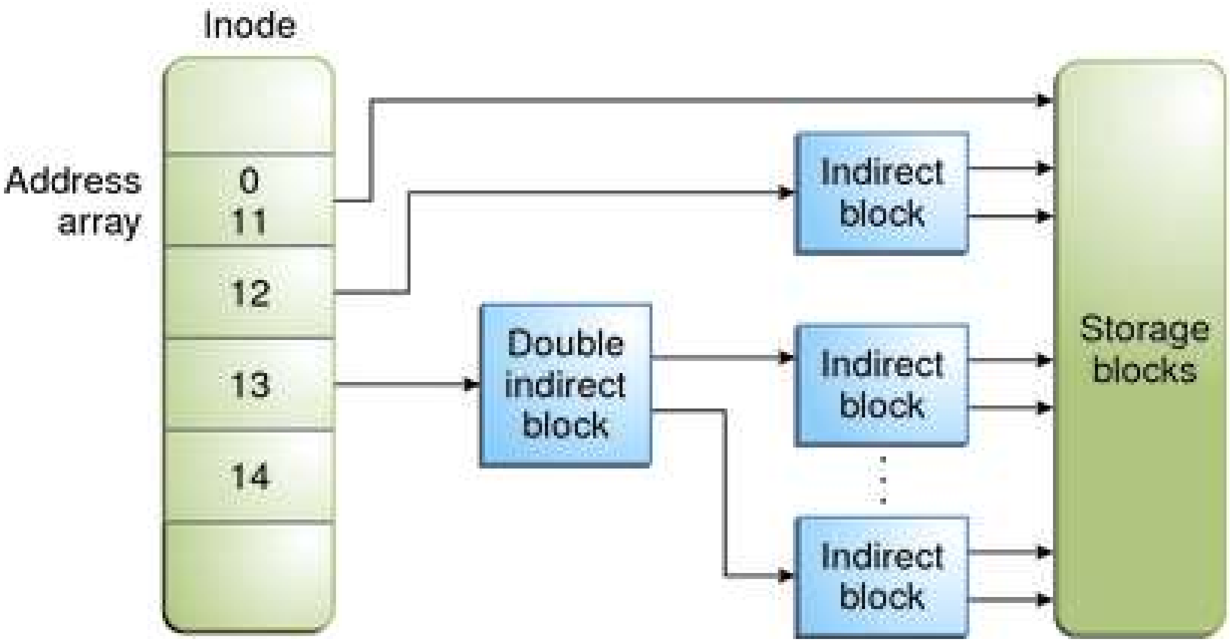
02 – Background

03 – SafeNVM Architecture

04 – Evaluation

Disk Based Systems and Data Safety





Block Interfaces Vs load-store interface

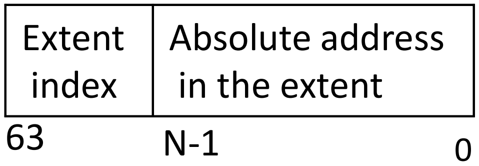
- Logical block address (LBA) for block devices
- Virtual address space for memory devices

File system inode as bounds checker

- File offset to LBA conversion = bounds checking

Persistent Pointers and Deswizzling

- Virtual pointers are tied to application's address space
- Sharing or loading at new address is tough
- A mapping is required to use persistent pointers
- Swizzling
  - Virtual address to Persistent pointer
- Deswizzling
  - Persistent pointer to Virtual address
- Deswizzling implies a bound checker



Persistent Pointer Layout

Outline

01 – Overview and Problem Statement ✓

02 – Background ✓

03 – SafeNVM Architecture

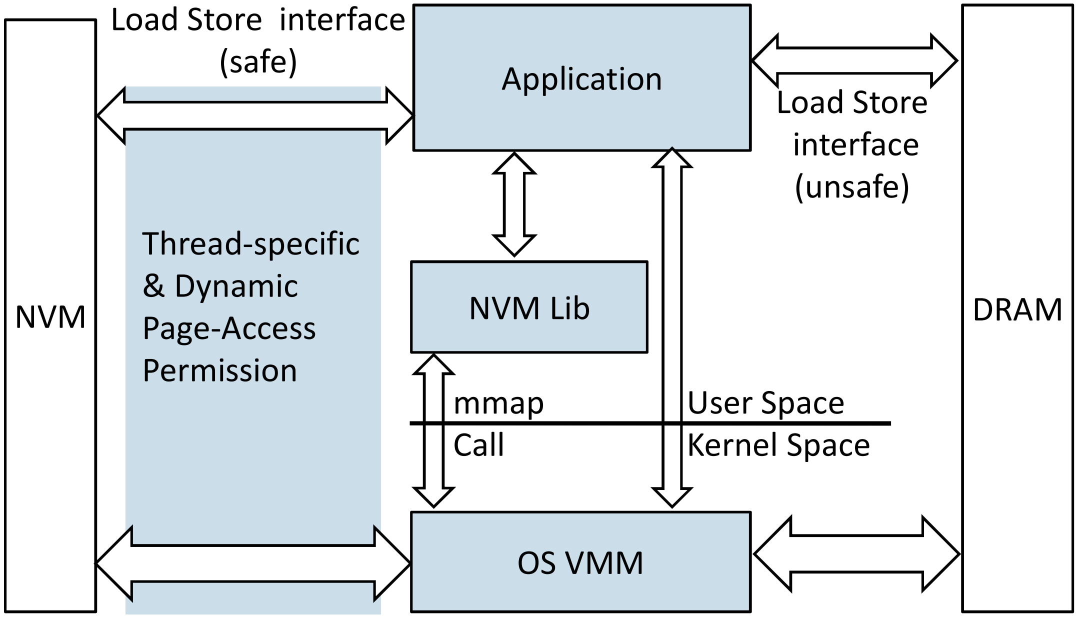
04 – Evaluation

SafeNVM Architecture

- Data Reliability Model

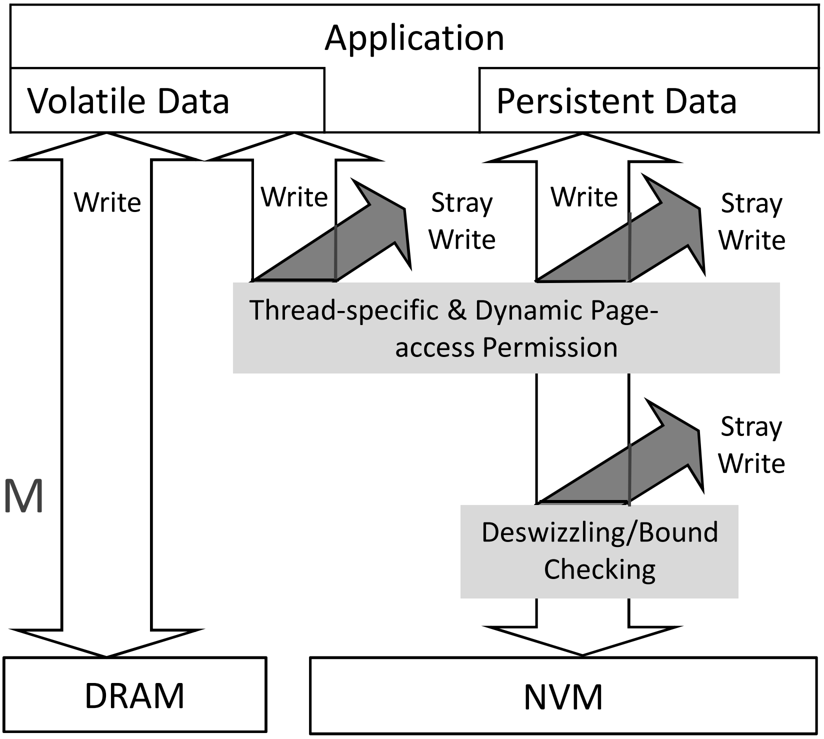
- Thread Level Page Protection

- Application Specific Object Store Design


Load Store interface (safe)
Application
Load Store interface (unsafe)
NVM
Thread-specific & Dynamic Page-Access Permission
NVM Lib
DRAM
mmap Call
User Space
Kernel Space
OS VMM

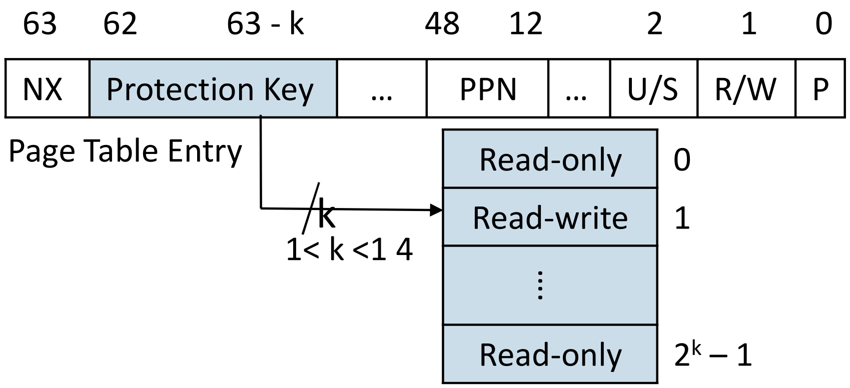
Data Safety Model of SafeNVM

- Equivalent to block devices
- Specialized interfaces
  - Block interface in block devices
  - Special instruction in SafeNVM
- Bound Checking
  - File System inode for block devices
  - Deswizzling of persistent pointers in SafeNVM



Thread Level Page Protection

- New Page-table and Permission-level Buffer
- TLB is changed similarly
- 6 bit Protection key => 64 protection domains
- New Hardware instruction for page access change



63 62 63 - k 48 12 2 1 0

NX | Protection Key | … | PPN | … | U/S | R/W | P

Page Table Entry

k

1< k <1 4

Read-only 0

Read-write 1

⋮

Read-only $2^k - 1$

Permission-Level Buffer(PLB)

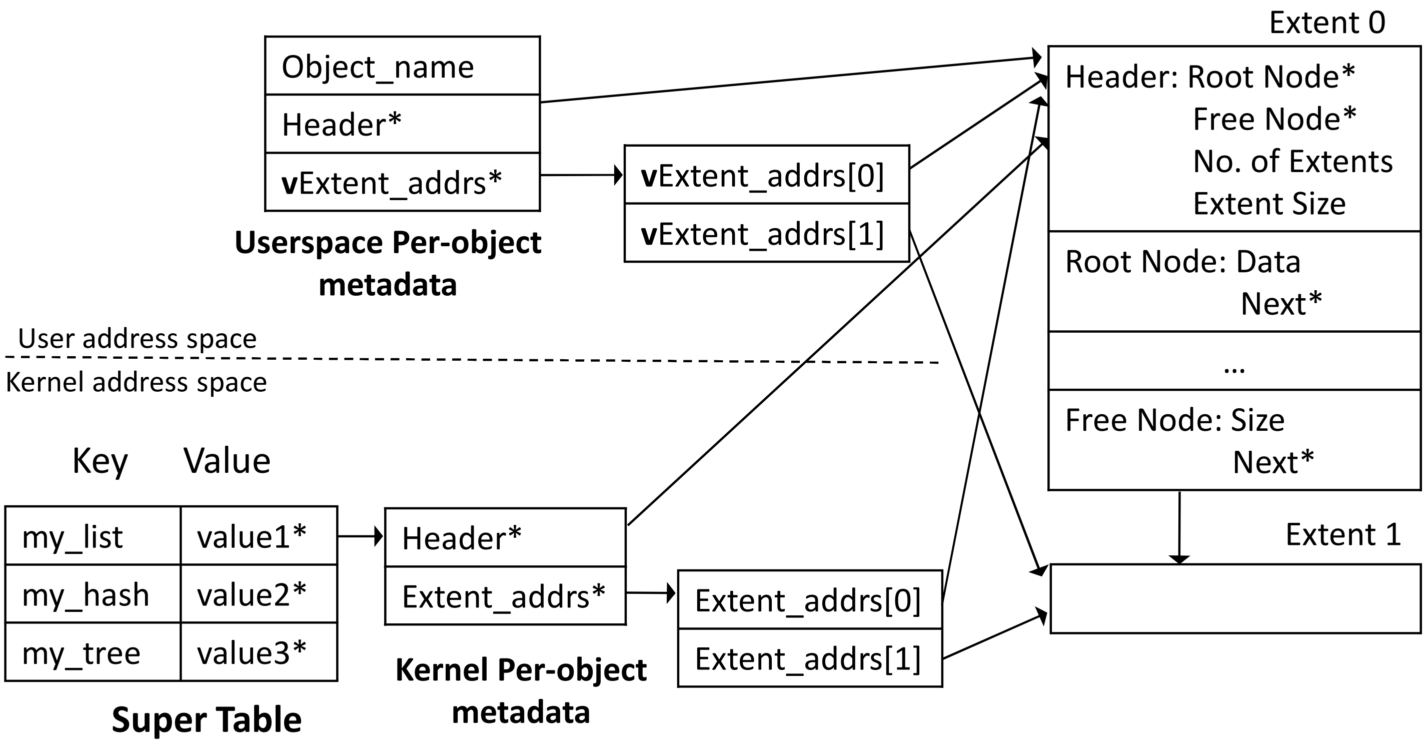
Instruction Name	PL bit	Action	Comment
set_write_access (Protection Key)	0=>1	Executing thread only gets NVM write-access	Permission stays during context switch
clear_write_access (Protection Key)	1=>0	Executing thread releases NVM Write-access	Read-access remains with all the threads

Proposed X86_64 Instruction

PL	R/W	Access Rights
R	R	read-only
W	R	Read-write

Page-access right calculation

Application Specific Object Store



Key	Value
my_list	value1*
my_hash	value2*
my_tree	value3*

All pointers inside extent are persistent pointers

APIs of SafeNVM

- Provides memory management
- Easy to change an application to use NVM.

Library API	
status	***create_object*** (incore_pobj*, objname, flag)
status	***delete_object*** (incore_pobj*, objname)
status	***load_object*** (incore_pobj*, objname, flag)
void*	***decode_ptr*** (incore_pobj*, splptr t)
splptr t	***encode_ptr*** (incore_pobj*, void*, extent index)
splptr t	***alloc*** (incore_obj*, size, void**)
void	***free*** (incore_pobj*, splptr t)
System Call	
status	***sys_create_object*** (incore pobj*, objname, flag)
status	***sys_delete_object*** (incore pobj*, objname, flag)
status	***sys_load_object*** (incore pobj*, objname, flag)
status	***sys_alloc_extent*** (incore pobj*, objname)
status	***sys_free_extent*** (incore pobj*, objname)

Outline

01 – Overview and Problem Statement ✓

02 – Background ✓

03 – SafeNVM Architecture ✓

04 – Evaluation

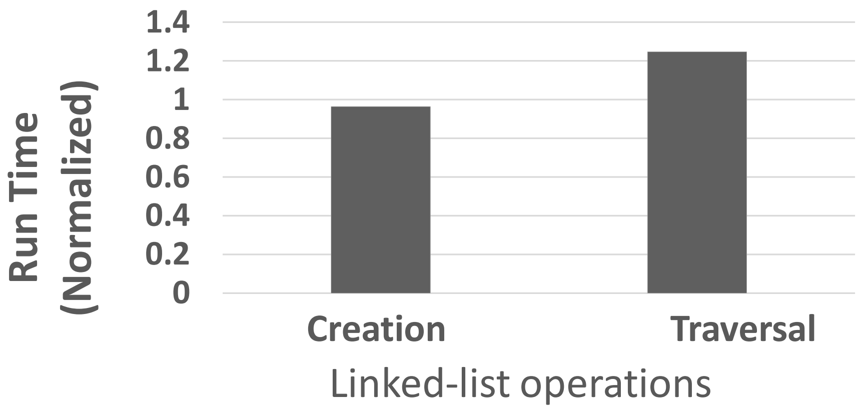
Evaluations

- Hardware changes in QEMU
  - 1 bit protection key in Page Table, TLB
  - 1 unused bit of EFLAGS as protection level buffer, part of context switch
  - 2 new hardware instruction

- Linux Kernel changes
  - mmap system call to pass protection key
  - Page table changes

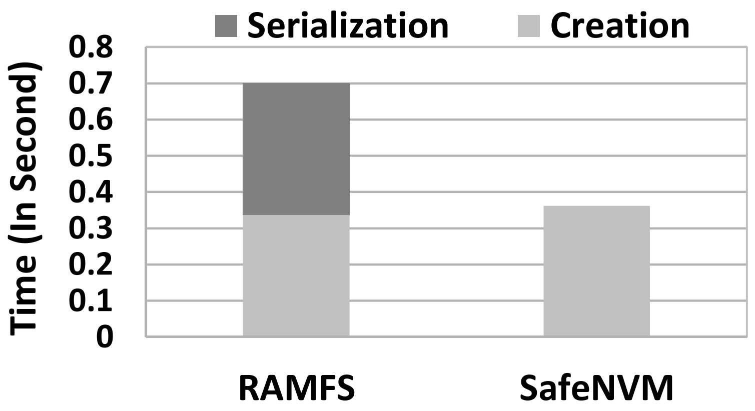
Evaluations

Case Number	Issues	Effect on NVM
CVE-2010-2160	Buffer Overflow	Data Corruption
CVE-2007-1211	Dangling Pointer	Data Corruption
CVE-2007-4000	Uninitialized Pointer	Data Corruption
CVE-2008-5187	Pointer Arithmetic	Data Corruption

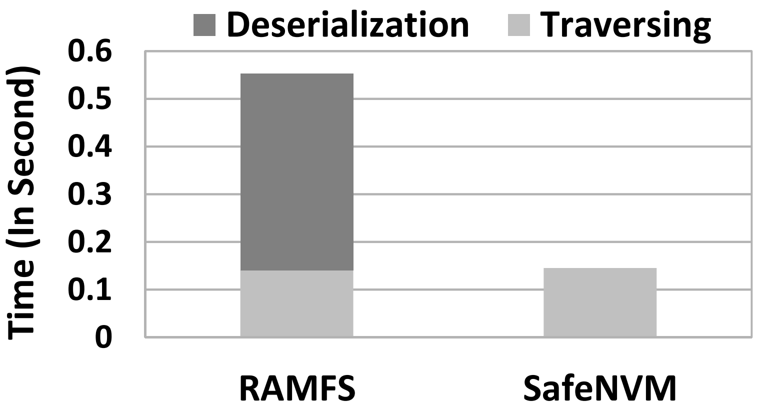


- 131,072 nodes of size 128 bytes
- 3.6% better then RAMFS for creation
- 24.5% performance degradation for traversal discarding deserialization cost

Persistent Pointer Overhead



(a) Linked-list Creation

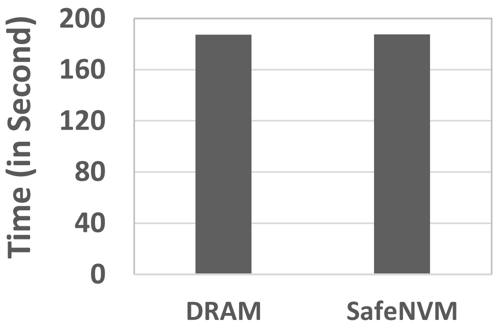


(b) Linked-list Traversal

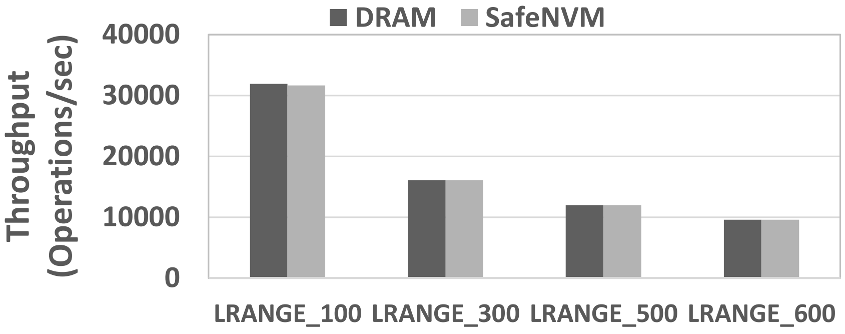
- 48% and 73% better than RAMFS.
- Ignoring serialization/deserialization cost
  - 7.4% and 3.4% worst than DRAM based linked-list.

*No QEMU changes involved. Measuring the overhead on Persistent Pointer.

Persistent Pointer Overhead



(a) Redis LPUSH Operation



(b) Redis LRANGE Operation

- Redis LPUSH operation: Creating the list of 10 million nodes
- Redis LRANGE operation: Traversing and getting specified number of nodes (e.g. 100 in LRANGE 100) from the list.
- Performance difference is less than 1%

Conclusion

- ➢ Data Safety is an important problem for NVM

- ➢ SafeNVM provides required data reliability
  - ▪ equivalent of disk-based system