SDC 19

September 23-26, 2019
Santa Clara, CA

Volatile Use of Persistent Memory

Usha Upadhyayula
Intel Corporation

Agenda

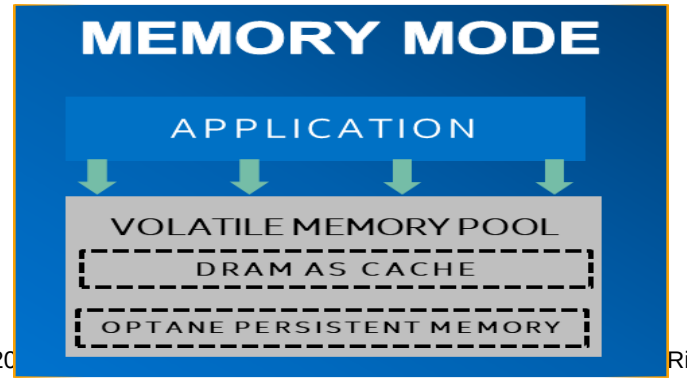
- Motivation
  - Why use persistent memory as volatile memory
- Exposing persistent memory as volatile memory
  - File-Backed memory
  - As a NUMA node
- Call to action

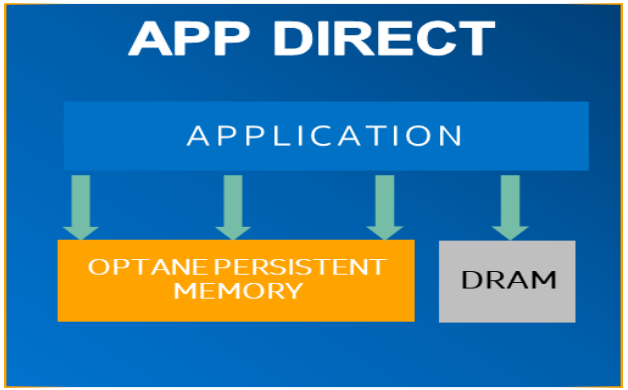
MOTIVATION

Intel® Optane™ DC Persistent Memory Operating Modes

- **Memory Mode (volatile)**
  - Data Placement
    - Application does not have control over data placement
      - DRAM and Intel® Optane™ DC Persistent Memory
  - Ease of adoption: No Code Changes
  - Performance slower than DRAM

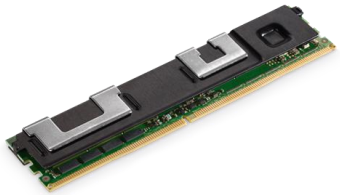
- **App Direct Mode (persistent)**
  - Data Placement
    - Control over data placement
  - Ease of Adoption:
    - Need Code Changes
  - Native hardware latencies





Motivation

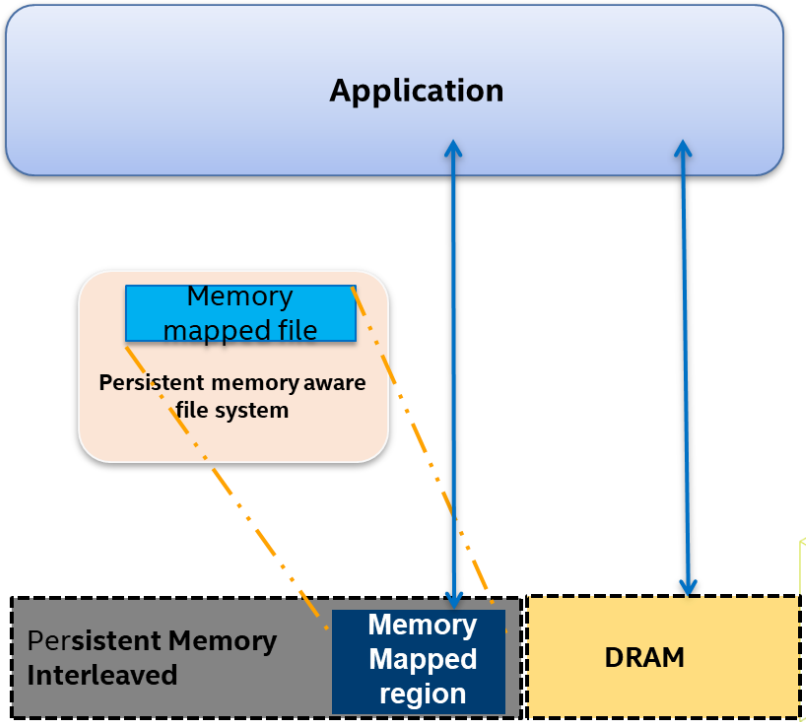
- Applications need large memory capacity
- Don’t need persistence
- Need control over data placement
  - DRAM and other storage tiers
- Native latencies of persistent memory



File-Backed Memory: Using Memory Mapped Files

File-Backed Memory

- Memory mapped files
  - Requires DAX aware file system
    - XFS, EXT4, NTFS
  - Bypasses file system page cache
- Fastest IO path possible
  - No Kernel code or Interrupts
- Code changes required for Load/Store Access



Memory Allocation Challenges

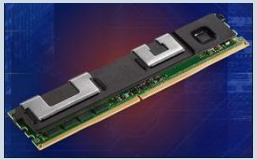
- malloc/free don’t work on memory mapped files
- Improve ease of use
  - stdlib like API to allocate memory

Solution: libmemkind

Unified Memory Management

DRAM

High BW Memory



Intel® Optane™ DC Persistent Memory

Allocator

Jemalloc 5.0

Familiar API

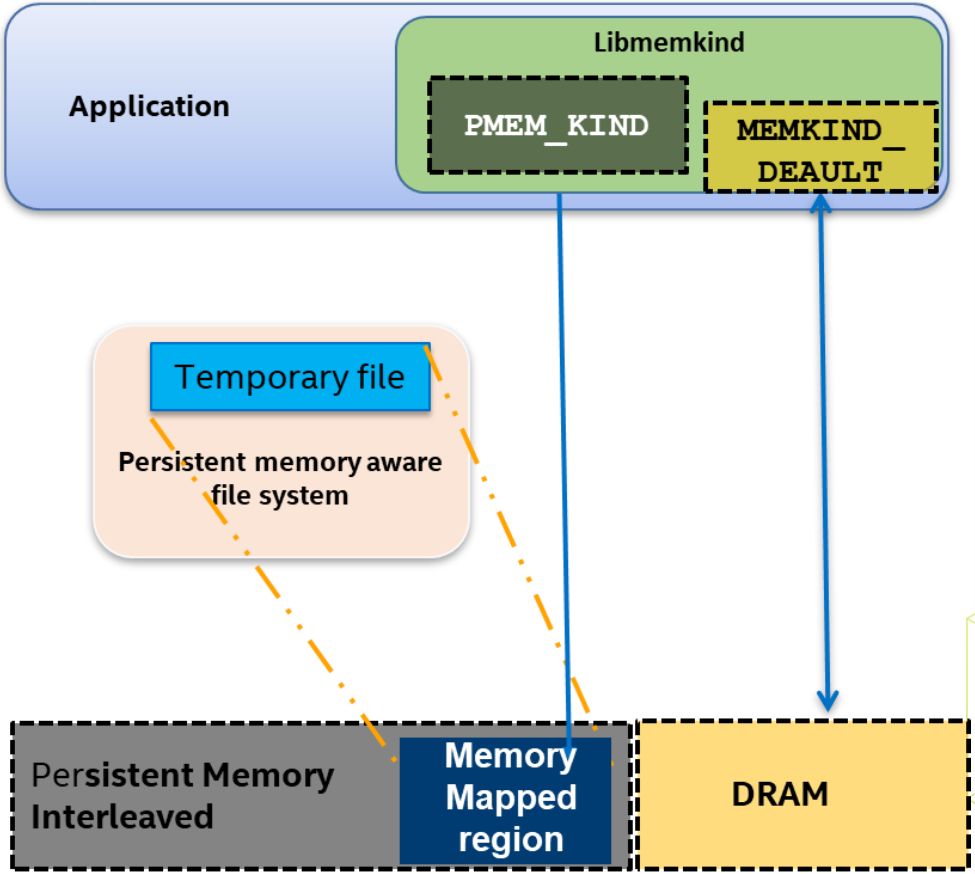
stdlib like API

Availability

https://github.com/memkind

libmemkind- How it Works

- PMEM_KIND
  - Supports malloc/free interface
- File-backed
  - Temporary file created & memory mapped on a persistent memory-aware file system
- Allocations not persistent
  - Temporary file deleted when the application exits
- Need simple modifications to the applications



Memkind API

- KIND Creation
  - Fixed & variable size heap
  - Automatic KIND detection
    - Static and dynamic KINDs
  - Destroy KIND
- KIND HEAP Management
  - Allocate
  - Free
  - Usable size
  - Detect KIND
- KIND Configuration Management
  - Usage Policy, Set Path, Set Size

Code Walkthrough

Create & Allocate from Different KINDs

```
int main(int argc, char *argv[])
{
        struct memkind *pmem_kind = NULL;
        memkind_create_pmem("/mnt/pmem", PMEM_MAX_SIZE, &pmem_kind);
        //allocate in DRAM
        char * ptr_default = (char *)memkind_malloc(MEMKIND_DEFAULT, size);
        //allocate in file backed "Kind" of memory
        char * ptr_pmem = (char *)memkind_malloc(pmem_kind, size);
        //Free allocated memory
        memkind_free(MEMKIND_DEFAULT, ptr_default);
        memkind_free(pmem_kind, ptr_pmem);
        memkind_destroy_kind(pmem_kind);
        return 0;
}
```

https://github.com/memkind/memkind/tree/master/examples/pmem_and_default_kind.c

Create with Defined and Unlimited Size

```
// Create first PMEM partition with specific size
struct memkind *pmem_kind = NULL
err = memkind_create_pmem(/mnt/pmem, PMEM_MAX_SIZE,
&pmem_kind);
// Create second PMEM partition with unlimited size
err = memkind_create_pmem(/mnt/pmem, 0, &pmem_kind_unlimited);
// Destroy both PMEM partitions
err = memkind_destroy_kind(pmem_kind);
err = memkind_destroy_kind(pmem_kind_unlimited);
```

https://github.com/memkind/memkind/tree/master/examples/pmem_kinds.c

Using KIND Configuration API

```
// Create new configuration
struct memkind_config *pmem_cfg = memkind_config_new();
memkind_config_set_path(pmem_cfg, "/mnt/pmem/");
memkind_config_set_size(pmem_cfg, 1024 * 1024 * 64);
memkind_config_set_memory_usage_policy(pmem_cfg,
    MEMKIND_MEM_USAGE_POLICY_CONSERVATIVE);
// Create pmem_kind with configuration set
memkind_create_pmem_with_config(pmem_cfg, &pmem_kind);
memkind_config_delete(pmem_cfg);
memkind_destroy_kind(pmem_kind);
```

https://github.com/memkind/memkind/tree/master/examples/pmem_config.c

Code Samples

File Name	Description
pmem_kinds.c	Create and destroy PMEM kind with defined or unlimited size.
pmem_malloc.c	Allocate memory and the possibility to exceed PMEM kind size.
pmem_malloc_unlimited.c	Allocate memory with unlimited kind size.
pmem_usable_size.c	View the difference between the expected and the actual allocation size.
pmem_alignment.c	Use memkind alignment and how it affects allocations.
pmem_multithreads.c	Use multithreading with independent PMEM kinds.
pmem_multithreads_onekind.c	Use multithreading with one main PMEM kind.
pmem_and_default_kind.c	Allocate in standard memory and file-backed memory (PMEM kind).
pmem_detect_kind.c:	Distinguish allocation from different kinds using the detect kind function.
pmem_config.c	Use custom configuration to create PMEM kind.
pmem_free_with_unknown_kind.c	Allocate in standard memory & file-backed memory (PMEM kind), and free memory without needing to remember which kind it belongs to.

https://github.com/memkind/memkind/tree/master/examples.

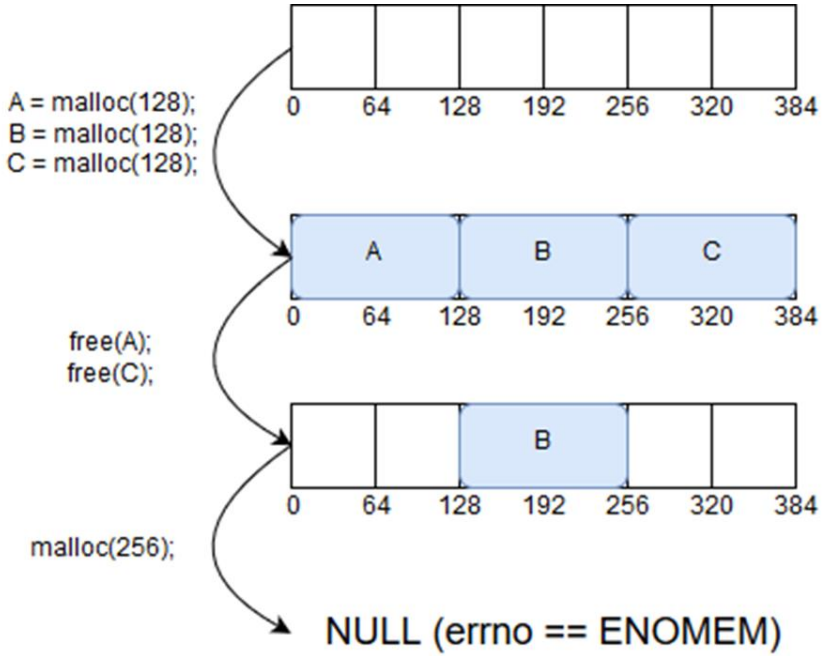
Libmemkind - Recap

- General purpose library
- Uses jemalloc for memory management
- Provides stdlib like interface
- Uses file-backed memory
  - Memory map temp files

An Efficient & Scalable Cache for Volatile use of Persistent Memory

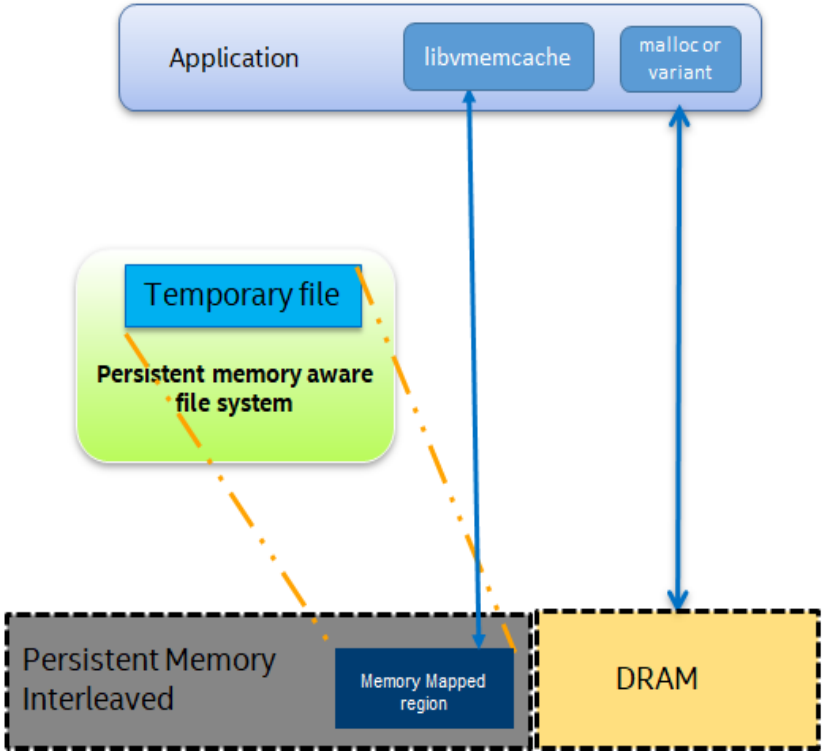
Cache Design Challenges: Fragmentation & Scalability

- Challenges
  - Random memory allocation sizes
  - Long runtime durations
  - Large memory capacities
- Problem
  - Failure to allocate a contiguous chunk of memory although the requested chunk is availble
- Existing solutions
  - Compacting GC (Java, .NET)
  - Defragmentation (Redis, Apache Ignite)
  - Slab allocation (memcached)



libvmemcache – Caching Solution for Large Memory Capacity

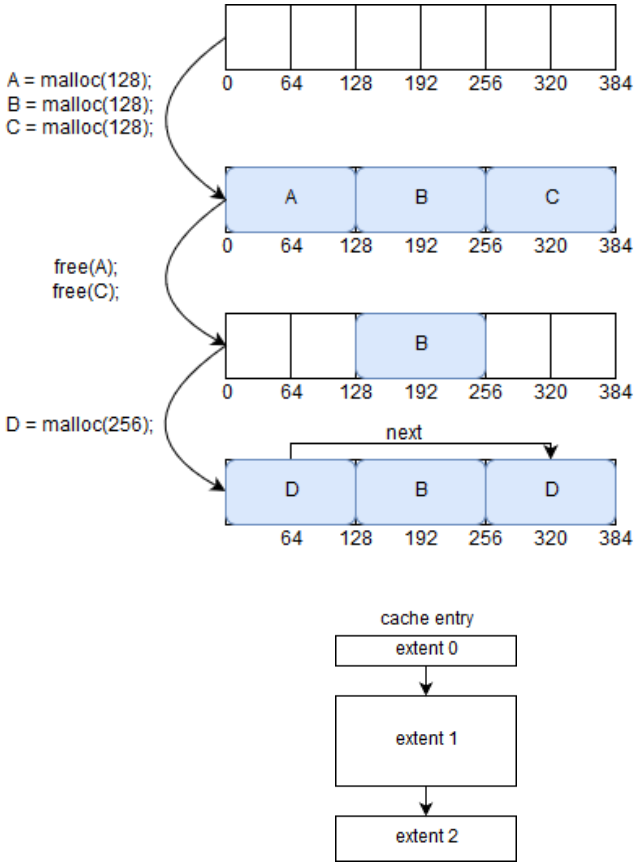
- Main Features
  - Custom memory allocator
    - Extent-based
      - Reduces fragmentation
    - Control over allocations
    - Improves space efficiency
  - Buffered LRU
    - Delivers scalability
      - Works with Large Memory Capacities
  - Uses memory mapped files



https://github.com/pmem/vmemcache

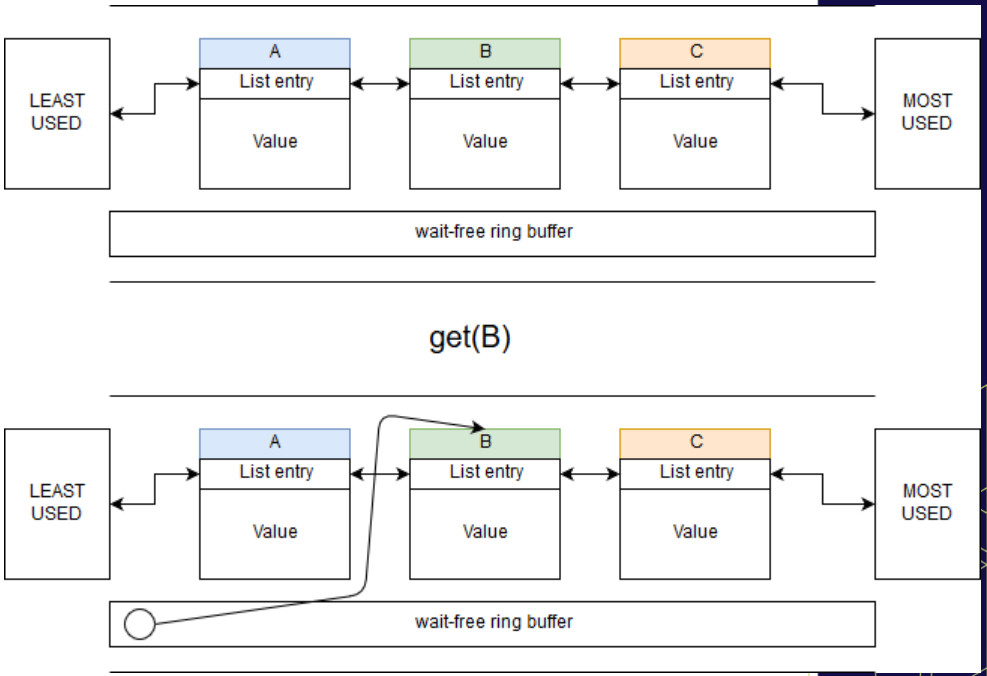
Extent Based Allocation

- Similar to file system extents
  - Extent = contiguous set of blocks
- Multiple non-contiguous blocks allocated
- Non-contiguous allocations appear as a single allocation



Cache Design - Buffered LRU

- Added a wait-free ring-buffer
  - buffers the list-move operations
- List only needs to get locked only during eviction or when the ring-buffer is full.



Design Aspects: libmemkind vs libvmemcache

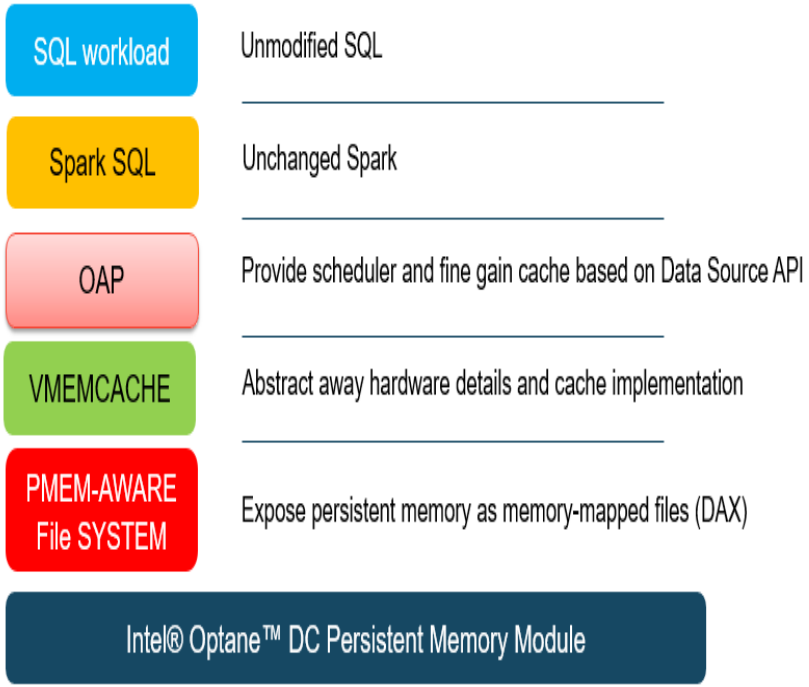
	libmemkind (PMEM)	libvmemcache
Allocation Scheme	Dynamic allocator	Extent based
Purpose	General purpose	Key-value store optimized for large memory capacities
Fragmentation	Apps with random size allocations/deallocations that run for a longer period	Minimized

Cache Design - Lightweight, embeddable, in-memory caching

```
VMEMcache *cache = vmemcache_new("/mnt/pmem", VMEMCACHE_MIN_POOL,
        VMEMCACHE_MIN_EXTENT, VMEMCACHE_REPLACEMENT_LRU);

const char *key = "foo";
vmemcache_put(cache, key, strlen(key), "bar", sizeof("bar"));

char buf[128];
ssize_t len = vmemcache_get(cache, key, strlen(key),
                buf, sizeof(buf), 0, NULL);

vmemcache_delete(cache);
```

Libvmemcache - Recap

- libvmemcache
  - get/put APIs
  - optional replacement policy
  - configurable extent size
- Works with terabyte-sized in-memory workloads
- High space utilization

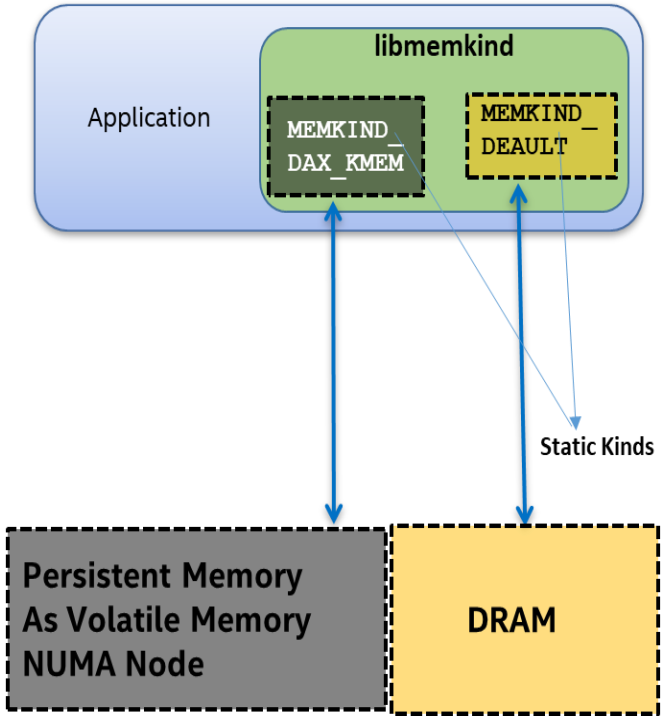


https://github.com/pmem/vmemcache

Persistent Memory as an Extension of DRAM

Persistent Memory as DRAM extension

- New feature in Linux* kernel 5.1
  - Dev_DAX_KMEM Config Option
  - Binds persistent memory device to kernel
  - Appears as a separate NUMA node
- libmemkind support
  - memkind_malloc with new static KIND
    - MEMKIND_DAX_KMEM



Call To Action

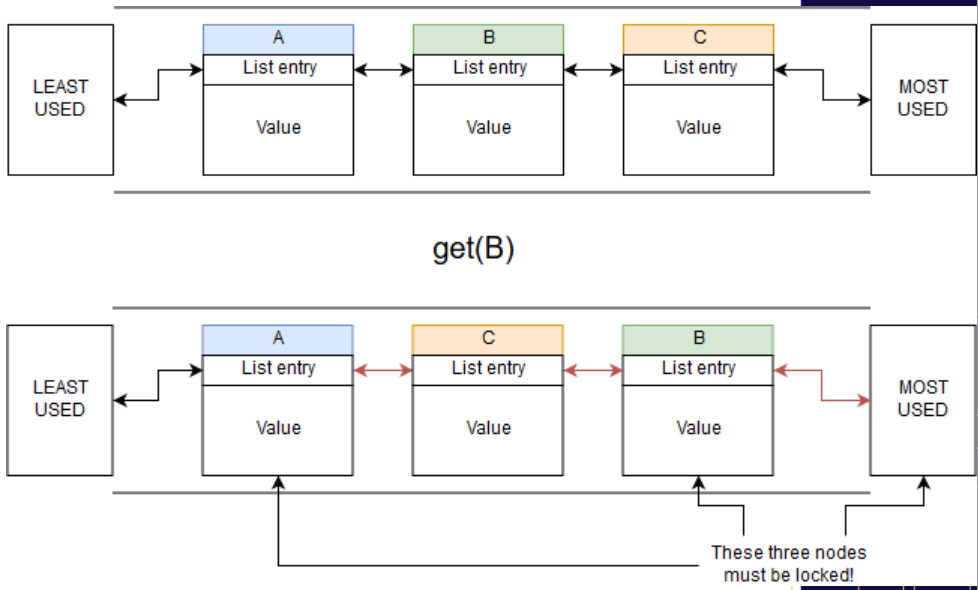
- Get started with persistent memory programming
  - https://github.com/pmem/pmdk
  - Libvmemcache
  - https://github.com/memkind/memkind
- Learn more about volatile usages
  - Accelerate your Apache Spark with Intel® Optane™ DC Persistent Memory
  - Pmem-redis
- Join the development efforts
  - pmem.io - http://pmem.io/
  - Intel Persistent Memory Developer Zone
- Send us your feedback

Let's Innovate As a Community

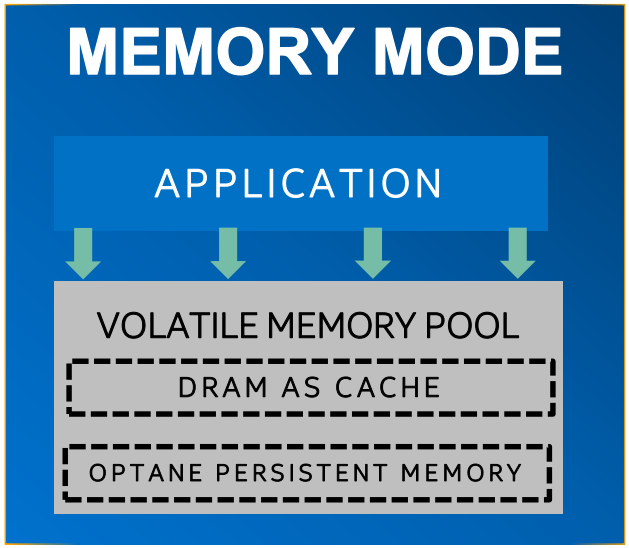
Backup

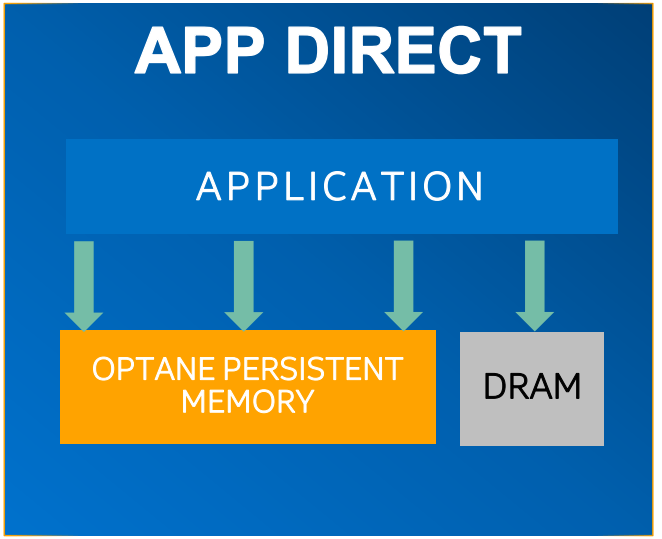
Cache Design - Scalable replacement policy

- Performance of libvmemcache was bottlenecked by naïve implementation of LRU based on a doubly-linked list.
- With 100st of threads, most of the time of any request was spent
  waiting on a list lock…
- Locking per-node doesn't solve the problem…



Intel® Optane™ DC Persistent Memory Operating Modes





AFFORDABLE & LARGE VOLATILE MEMORY CAPACITY

LARGE CAPACITY, PERSISTENT MEMORY