Persistent Memory Hackathon and Workshop
NVMW '19

March 10, 2019

Contributors:
Jim Fister (SNIA)
Stephen Bates (Eideticom)
Andy Rudoff (Intel)

https://github.com/pmemhackathon/2019-03-10

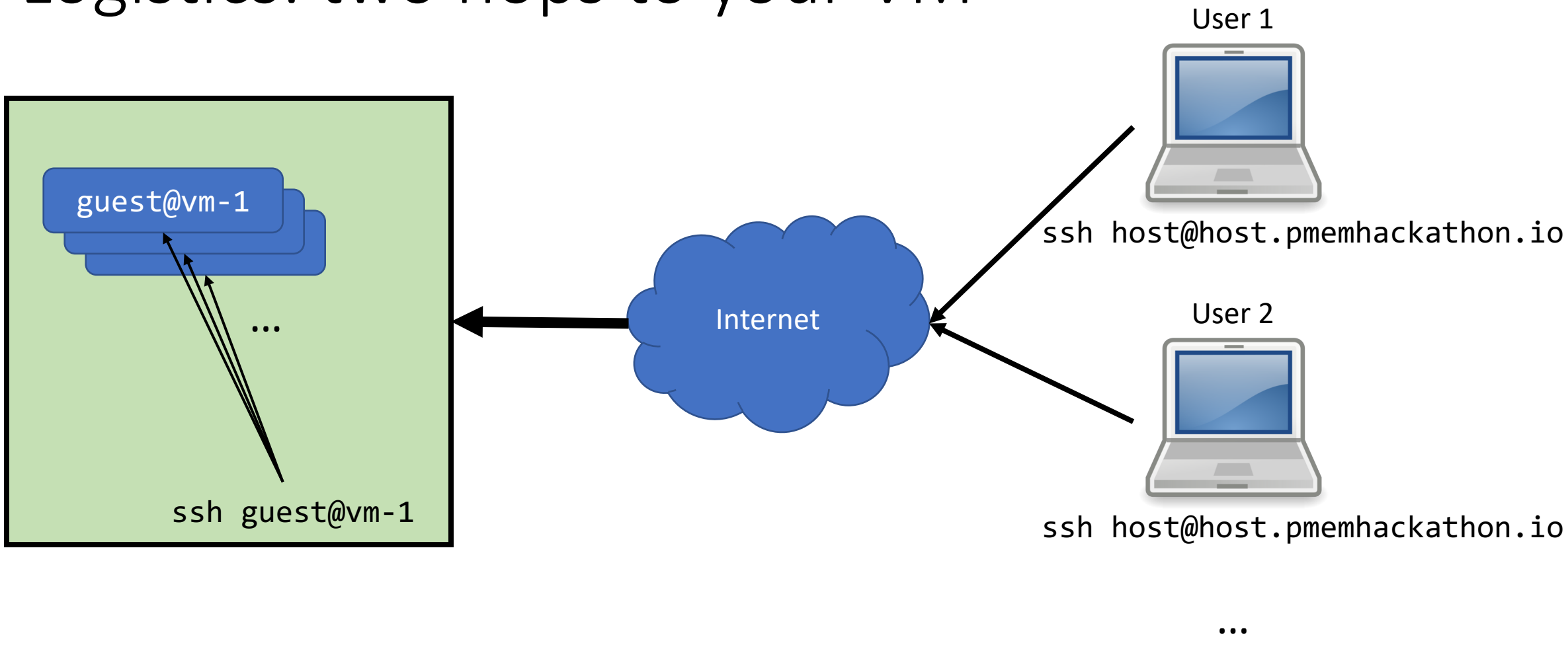
Agenda

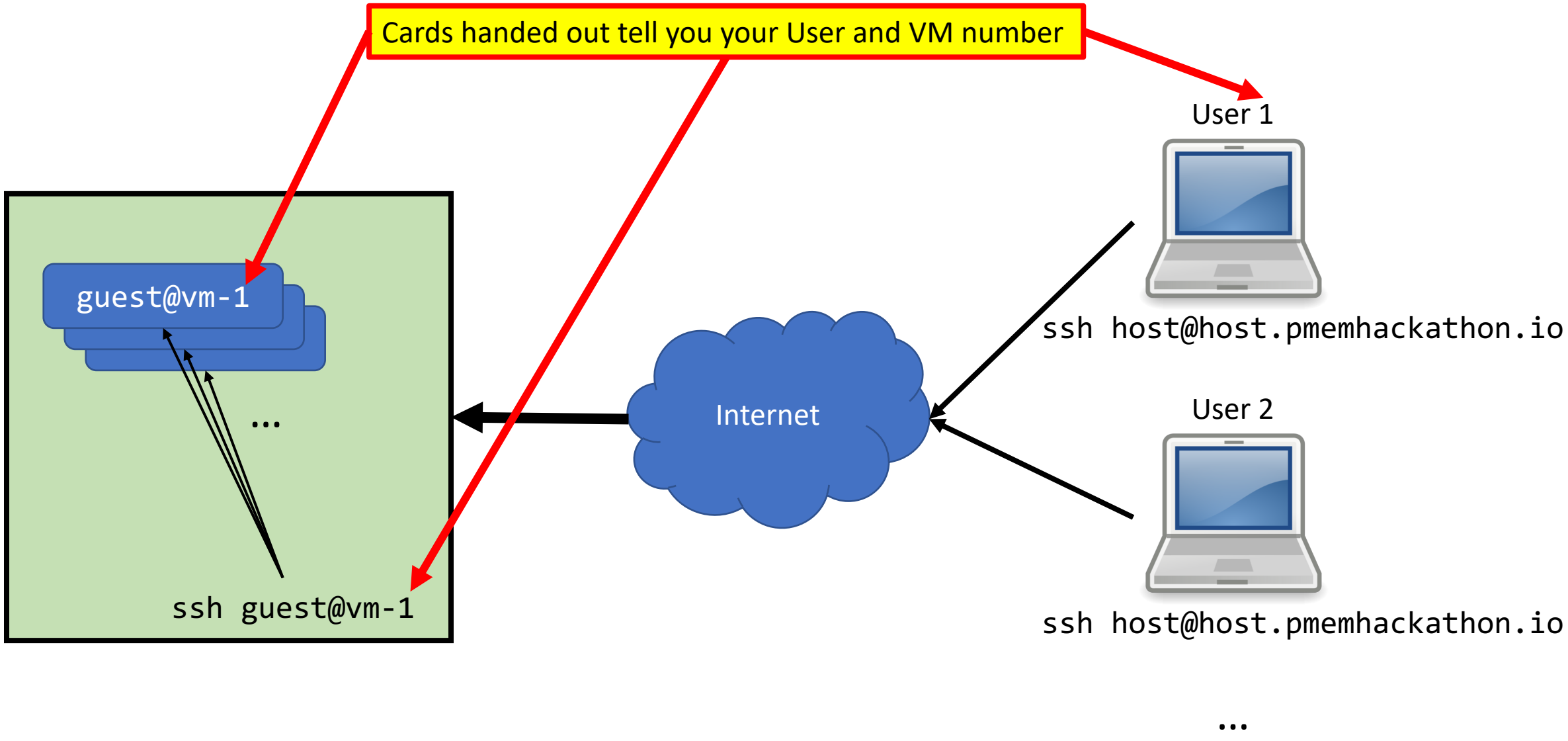
- Logistics
  - How to login to your VM & get it ready
- Persistent Memory Platform Support
  - Platform level support
  - Checking out your kernel
  - Finding and configuring your pmem
- Persistent Memory Programming
  - The basics: storage APIs and memory-mapped files
  - An application's responsibilities when using pmem
  - Installing libraries to help
    - High-level, easy-to-use APIs
    - Lower-level, more flexible (more error-prone) APIs

What Does “Hackathon” Mean To Us?

- Main goal is to show you how to find, configure, and program pmem
  - All slides are in the GitHub repo
  - All shell commands we type are in the GitHub repo
    - You probably don’t need to write them down
    - You probably don’t even need to type many of them, just cut & paste into the shell
  - Go to https://github.com/pmemhackathon/2019-03-10 to see today’s repo
    - But in a minute, we’ll demonstrate cloning the repo to your VM
- Mostly we will show you how to install stuff and get you going
  - After installing samples, try them out, or write your own
  - We’ll walk through some for everyone, then will walk around & help you

Logistics: two hops to your VM





VM Basics

- Set your hostname
  - `sudo sh –c 'echo vm-1 >/etc/hostname'`
- Make sure your SW is up to date
  - `sudo apt update && sudo apt upgrade`
  - `sudo reboot`

- NOTE: Today's Hackathon is using **emulated** pmem!
  - (but some demos will be on the real stuff)

Make a local clone of the hackathon repo

```
$ cd
$ git clone https://github.com/pmemhackathon/2019-03-10
Cloning into '2019-03-10'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 25 (delta 5), reused 20 (delta 4), pack-reused 0
Unpacking objects: 100% (25/25), done.
$ cd 2019-03-10
$ more README.txt
```

Most of the shell commands we type during demos are in this README.txt

Does your System Support Persistent Memory?

- Does my platform support persistent memory?
  - Your vendor determines this. Buy a system meant for it.
    - Don’t just buy an NVDIMM and plug it into a random system – you need platform support (like BIOS, ADR, power supply). You want validated configurations.

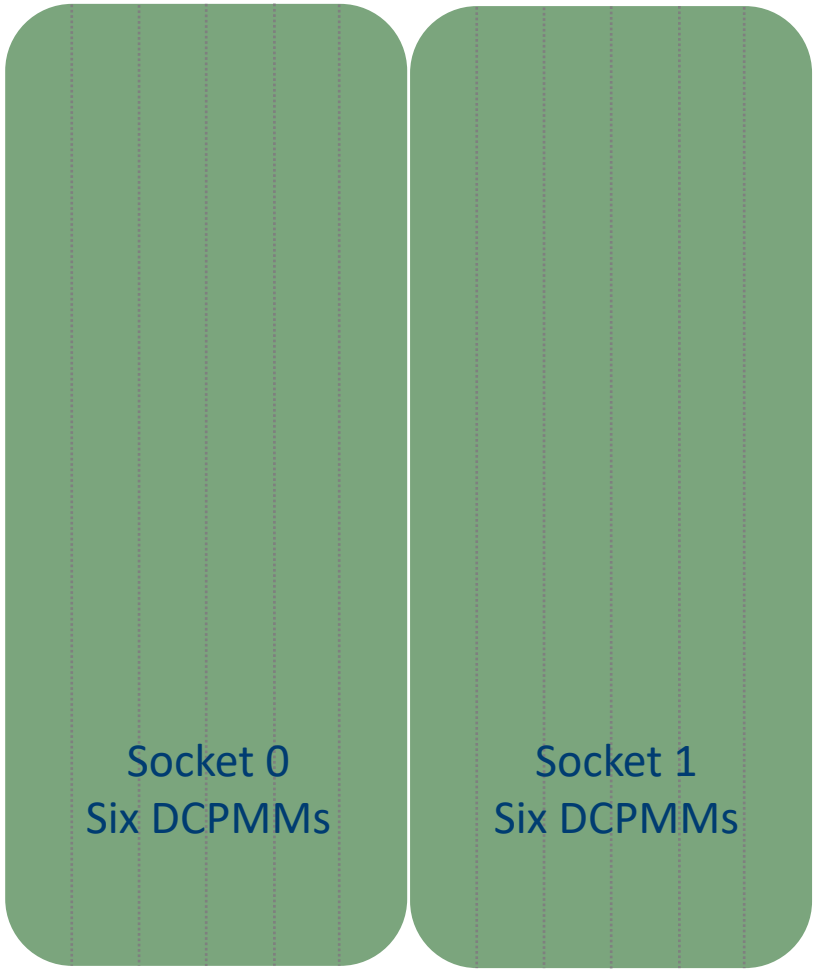
```
ndctl list –BN     # check the "provider" field for ACPI.NFIT
```

- Does my OS support persistent memory?
  - Major OS vendors (and Linux distros) will tell you which version supports it
  - Linux kernel support is enabled in the config file used to build the kernel

```
uname -r         # see kernel currently running
grep -i pmem /boot/config-`uname -r`
grep -i nvdimm /boot/config-`uname -r`
```

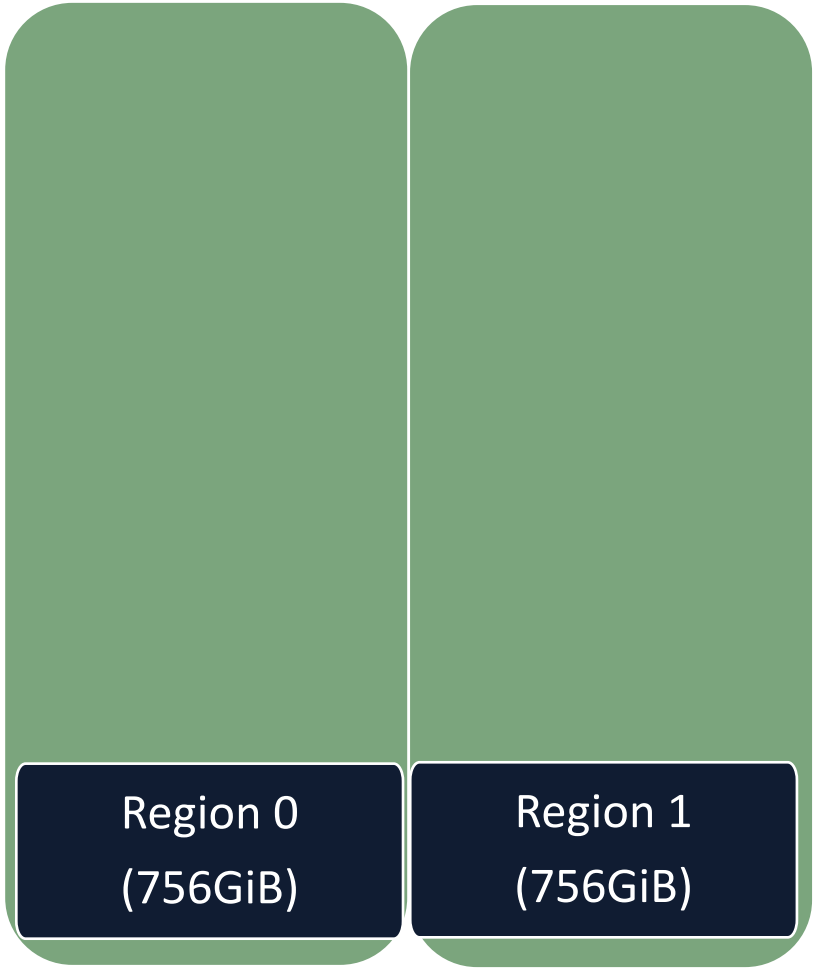
Example: Provisioning Intel® Optane DC Persistent Memory Modules

Hardware



Persistent Memory Modules
(Interleave sets)

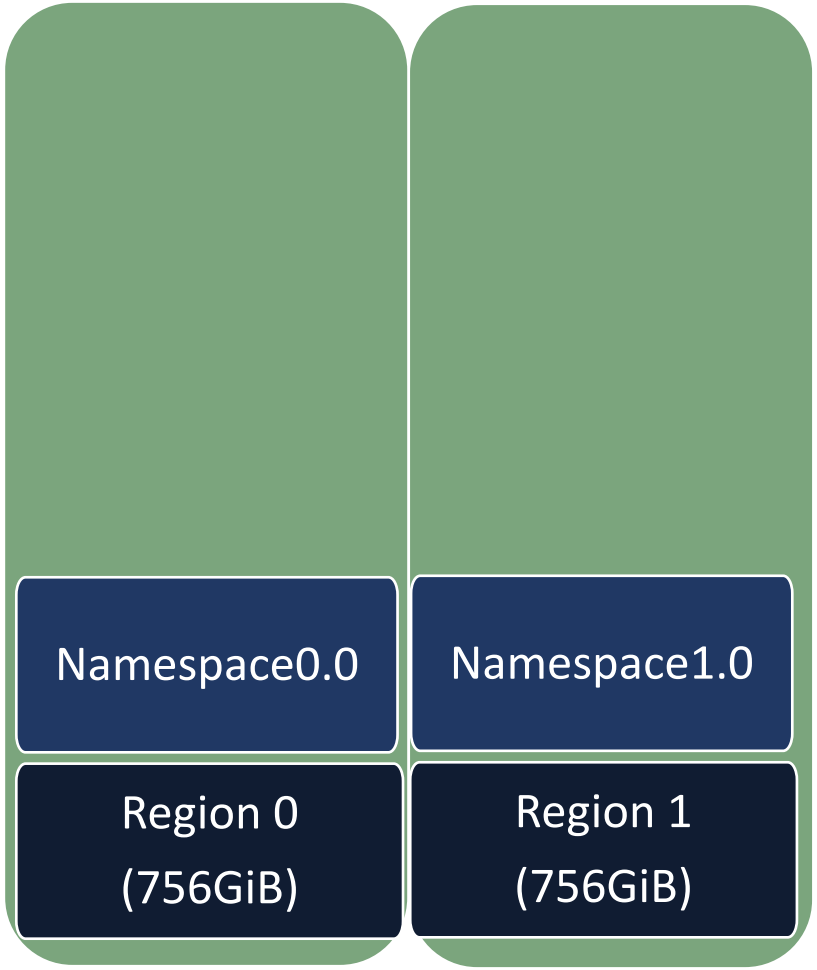
Hardware



Persistent Memory Modules
(Interleave sets)

```
# ipmctl create -goal PersistentMemoryType=AppDirect
```

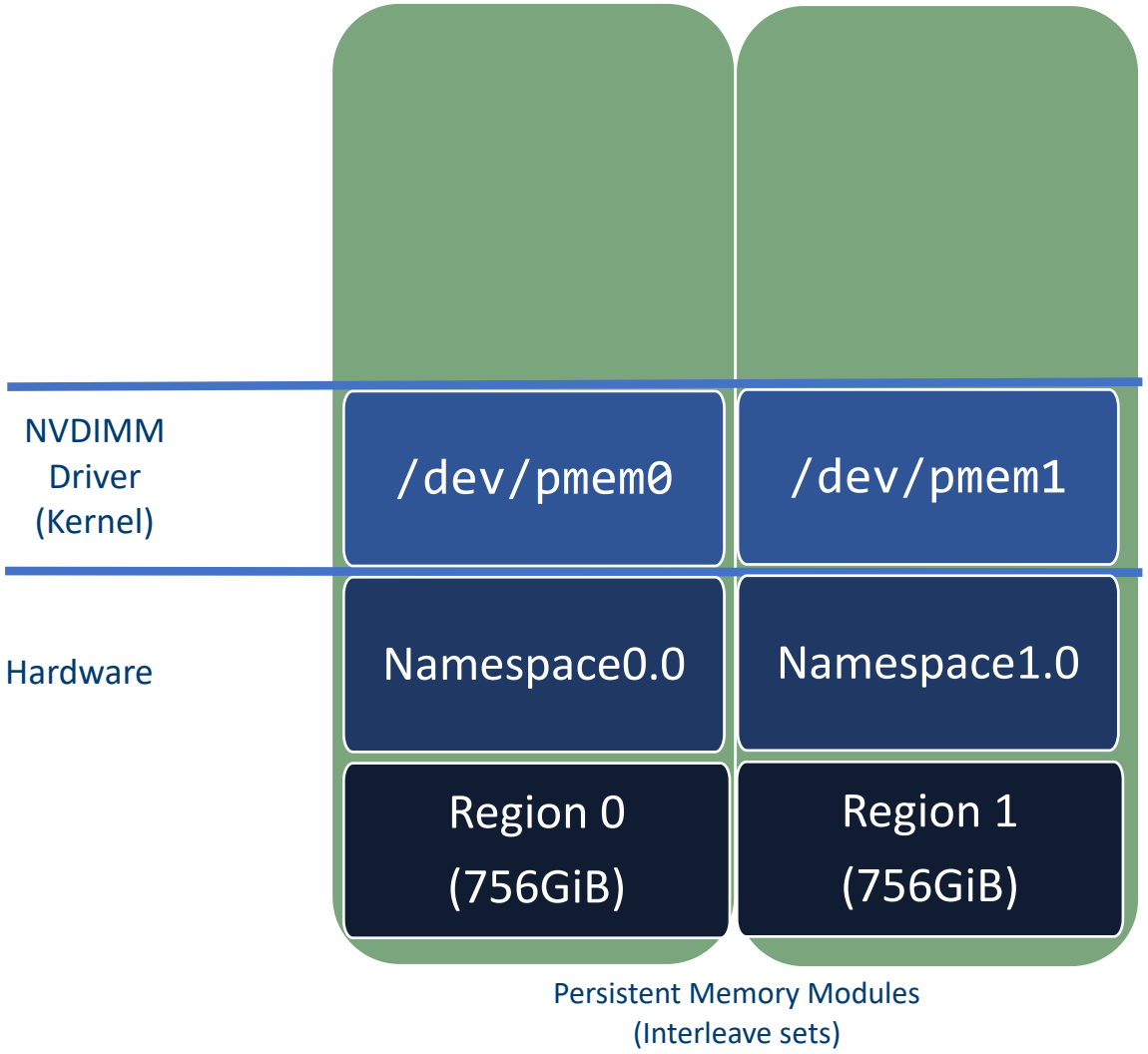
Hardware



Persistent Memory Modules
(Interleave sets)

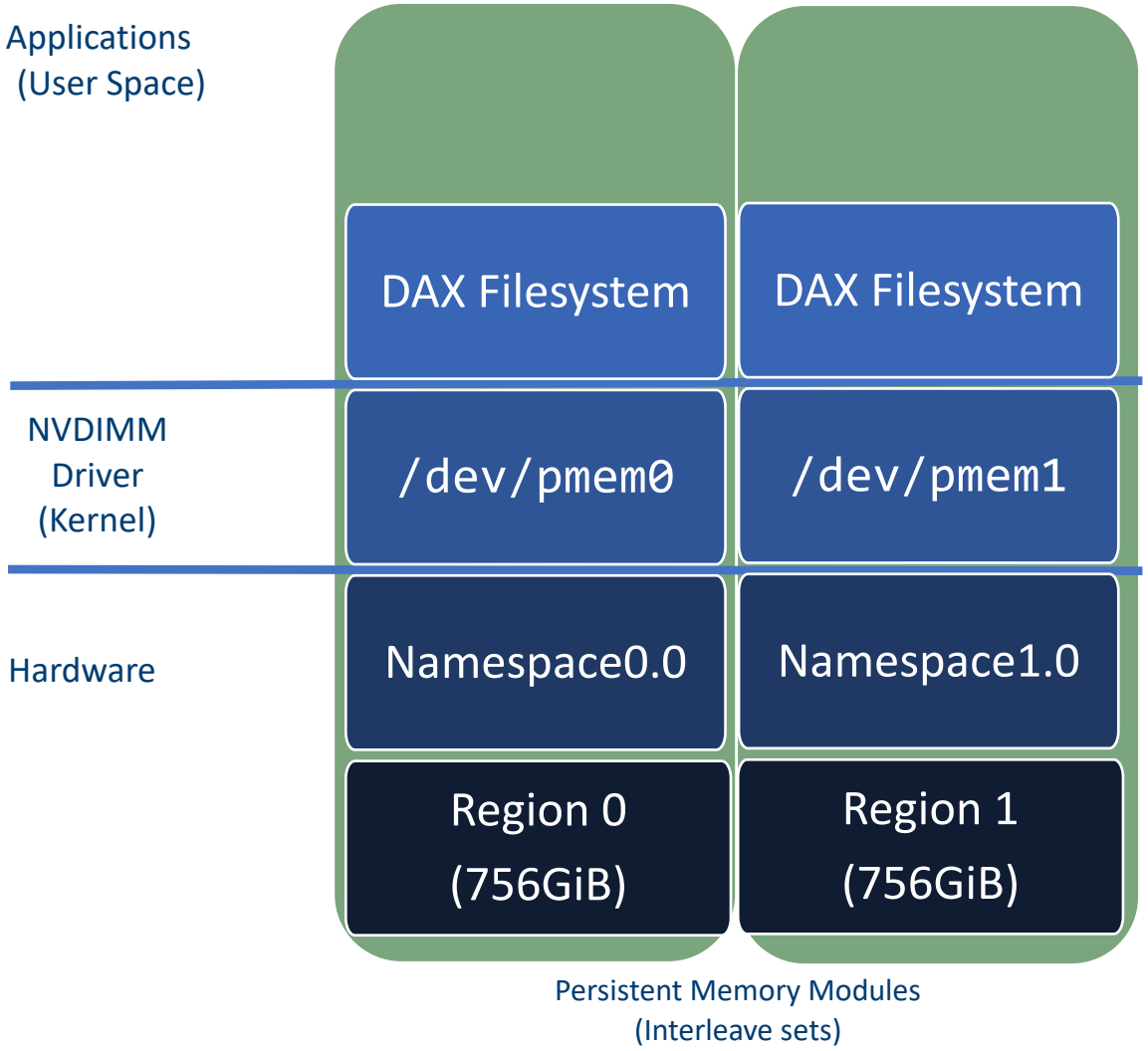
```
# ndctl create-namespace
# ndctl create-namespace
```

```
# ipmctl create -goal PersistentMemoryType=AppDirect
```




```
# ndctl create-namespace
# ndctl create-namespace
```

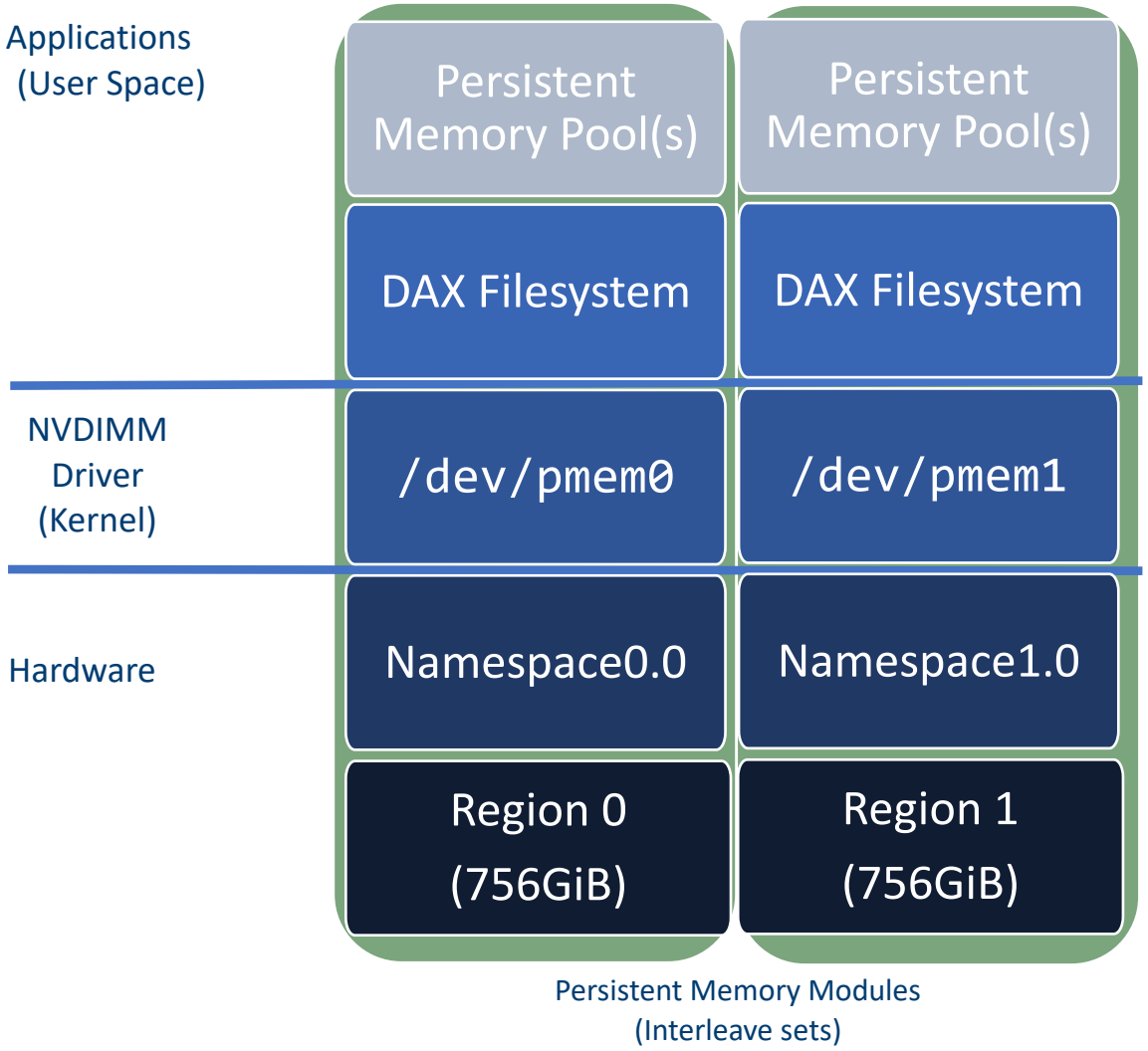
```
# ipmctl create -goal PersistentMemoryType=AppDirect
```




```
# mkfs.ext4 /dev/pmem0
# mkfs.ext4 /dev/pmem1
# mount -o dax /dev/pmem0 /mnt/pmem-fsdax0
# mount -o dax /dev/pmem1 /mnt/pmem-fsdax1
```

```
# ndctl create-namespace
# ndctl create-namespace
```

```
# ipmctl create -goal PersistentMemoryType=AppDirect
```

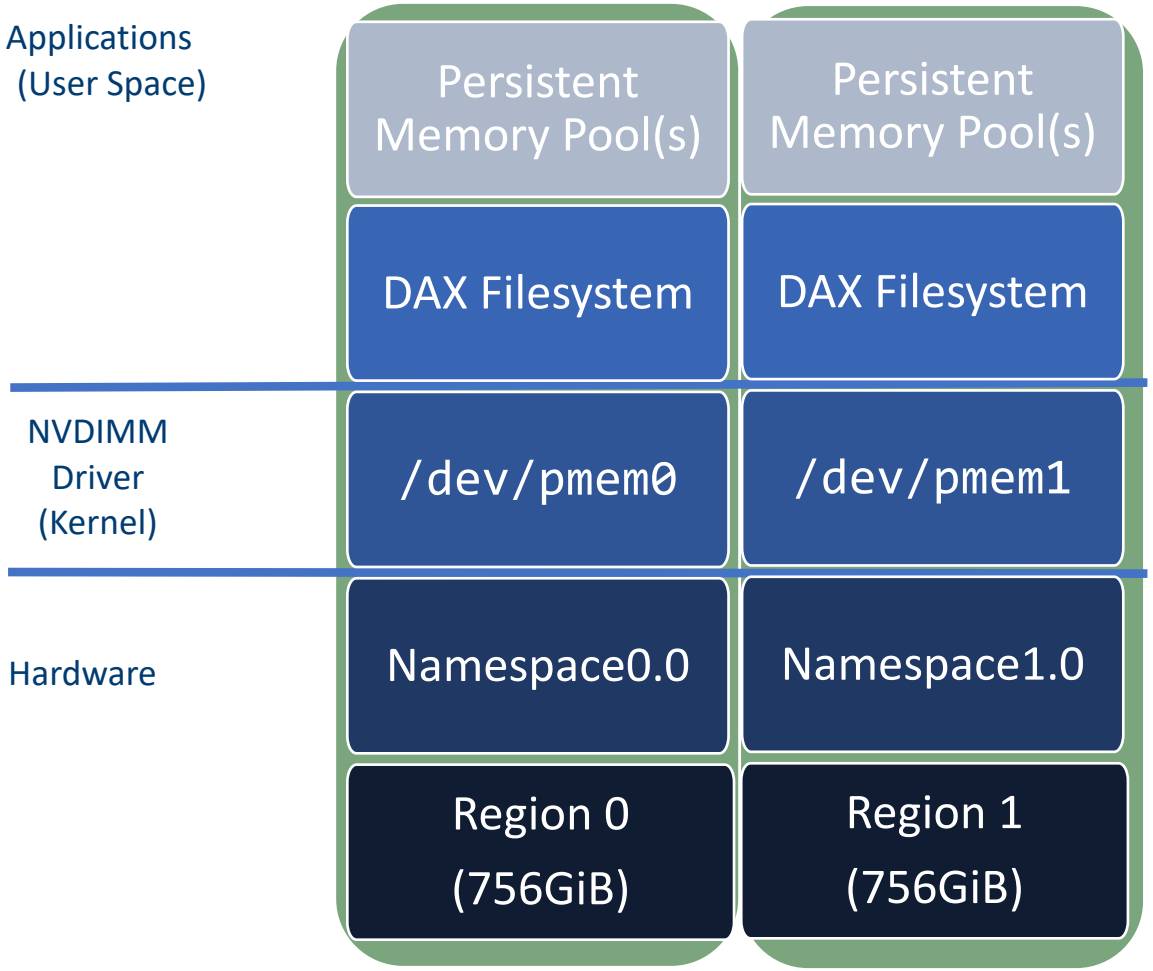



```
$ pmempool create /mnt/pmem-fsdax0/pool0
$ pmempool create /mnt/pmem-fsdax1/pool1
```

```
# mkfs.ext4 /dev/pmem0
# mkfs.ext4 /dev/pmem1
# mount –o dax /dev/pmem0 /mnt/pmem-fsdax0
# mount –o dax /dev/pmem1 /mnt/pmem-fsdax1
```

```
# ndctl create-namespace
# ndctl create-namespace
```

```
# ipmctl create -goal PersistentMemoryType=AppDirect
```




```
$ pmempool create /mnt/pmem-fsdax0/pool0
$ pmempool create /mnt/pmem-fsdax1/pool1


# mkfs.ext4 /dev/pmem0
# mkfs.ext4 /dev/pmem1
# mount -o dax /dev/pmem0 /mnt/pmem-fsdax0
# mount -o dax /dev/pmem1 /mnt/pmem-fsdax1




# ndctl create-namespace
# ndctl create-namespace
```

Vendor Neutral

```
# ipmctl create -goal PersistentMemoryType=AppDirect
```

Vendor Specific

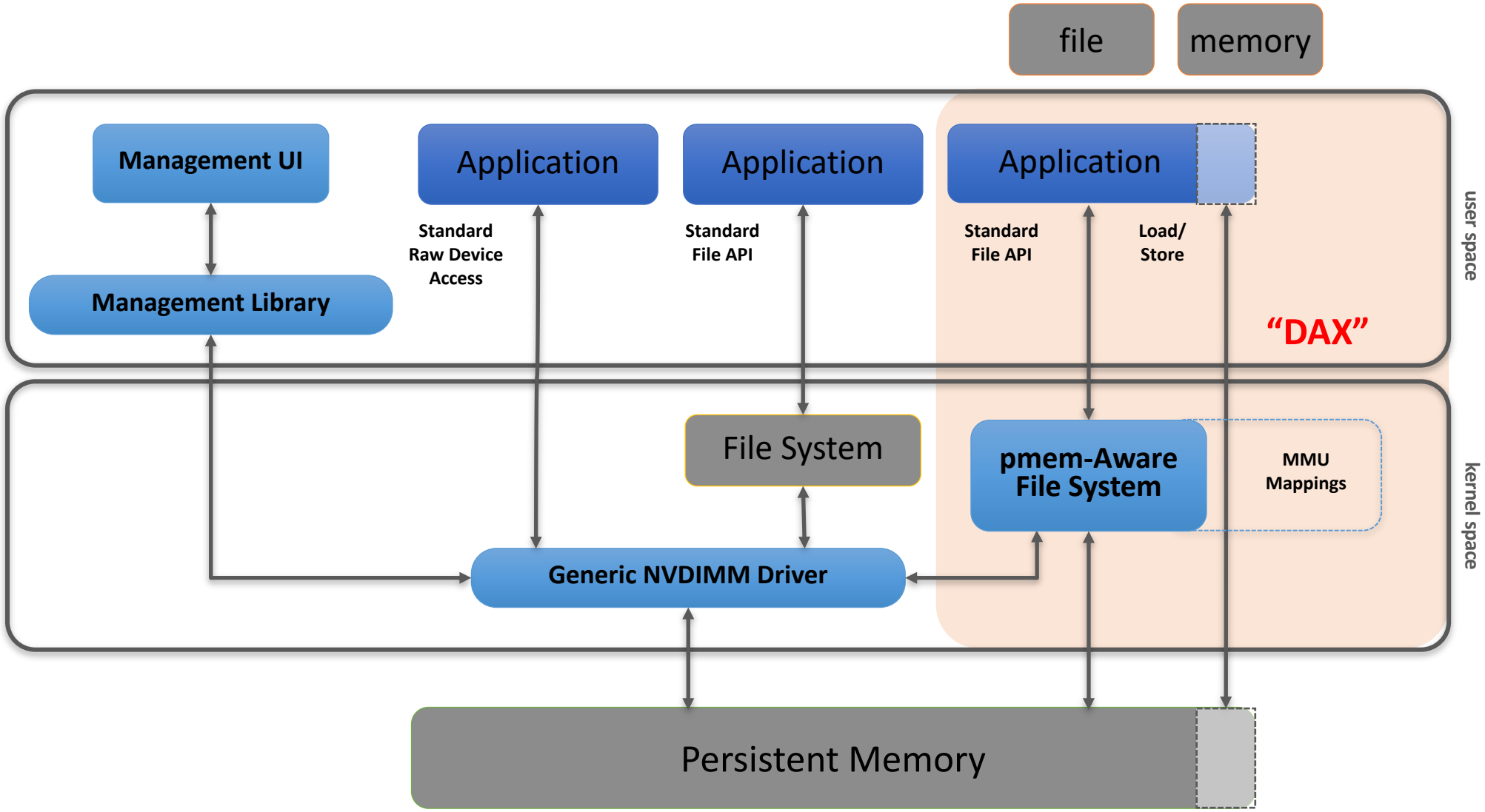
In your VM…

```
$ sudo ndctl list -u
$ sudo ndctl create-namespace –f –e namespace0.0 --mode fsdax

$ ls –l /dev/pmem*

$ sudo mkfs.ext4 /dev/pmem0

$ sudo mkdir /mnt/pmem-fsdax
$ sudo mount -o dax /dev/pmem0 /mnt/pmem-fsdax

$ sudo chmod 777 /mnt/pmem-fsdax    # open up perms for this hackathon

$ df –h
... other file-related stuff works as expected…
```

Essential Programming Background

- Lots of ways to use pmem with existing programs
  - Storage APIs
  - Libraries or kernels using pmem transparently
  - Memory Mode
- This hackathon doesn't cover the above (too easy!)
  - We assume you want direct access to pmem
  - We show code, but also concepts
  - There are lots of paths you can take, these are just examples

The SNIA NVM Programming Model



Programming Examples For This Hackathon

- RAW Access to pmem
  - mmap() -- you get a pointer to pmem, the rest is up to you
  - Only 8-byte stores are powerfail atomic
- libpmem
  - One step above RAW access, still only 8-byte stores are powerfail atomic
  - mmap(), memcpy() helper functions, optimized flush functions
- libpmemblk
  - Very simple transactional library, read/write fixed sized block only
- libpmemobj
  - **General-purpose allocations, transactions, atomics (series of examples)**
- Pointers to related info:
  - libmemkind, libpmemkv

Resources

- PMDK Resources:
  - Home: https://pmem.io
  - PMDK: https://pmem.io/pmdk
  - PMDK Source Code : https://github.com/pmem/PMDK
  - Google Group: https://groups.google.com/forum/#!forum/pmem
  - Intel Developer Zone: https://software.intel.com/persistent-memory
  - Memkind: https://github.com/memkind/memkind (see memkind_pmem(3))
  - libpmemkv: https://github.com/pmem/pmemkv
- NDCTL: https://pmem.io/ndctl
- SNIA NVM Programming Model: https://www.snia.org/tech_activities/standards/curr_standards/npm
- Getting Started Guides: https://docs.pmem.io

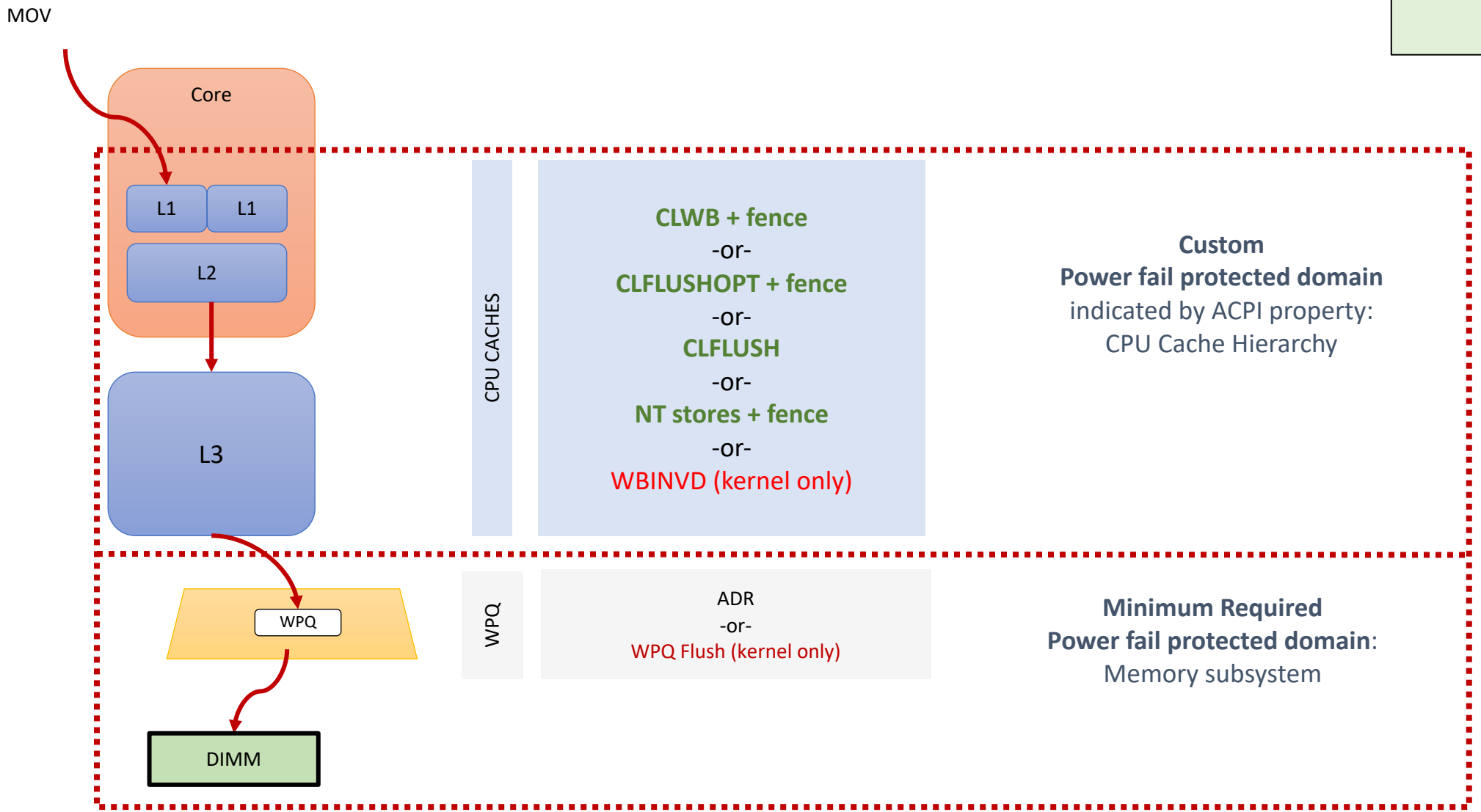
A Programmer's View (mapped files)

```
fd = open("/my/file", O_RDWR);
…
base = mmap(NULL, filesize,
                  PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
close(fd);
…
base[100] = 'X';
strcpy(base, "hello there");
*structp = *base_structp;
…
```

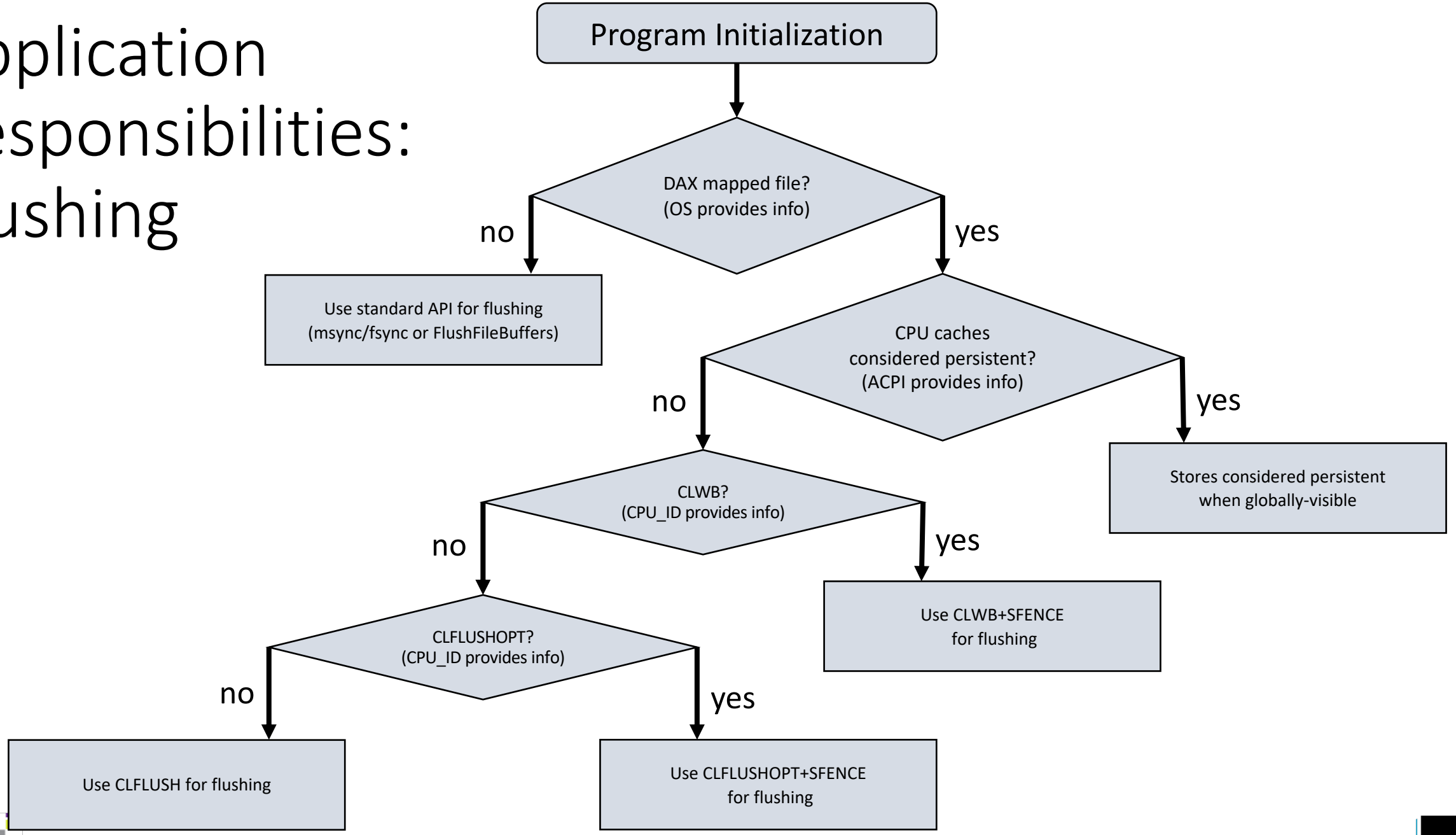
"Load/Store"

How the Hardware Works

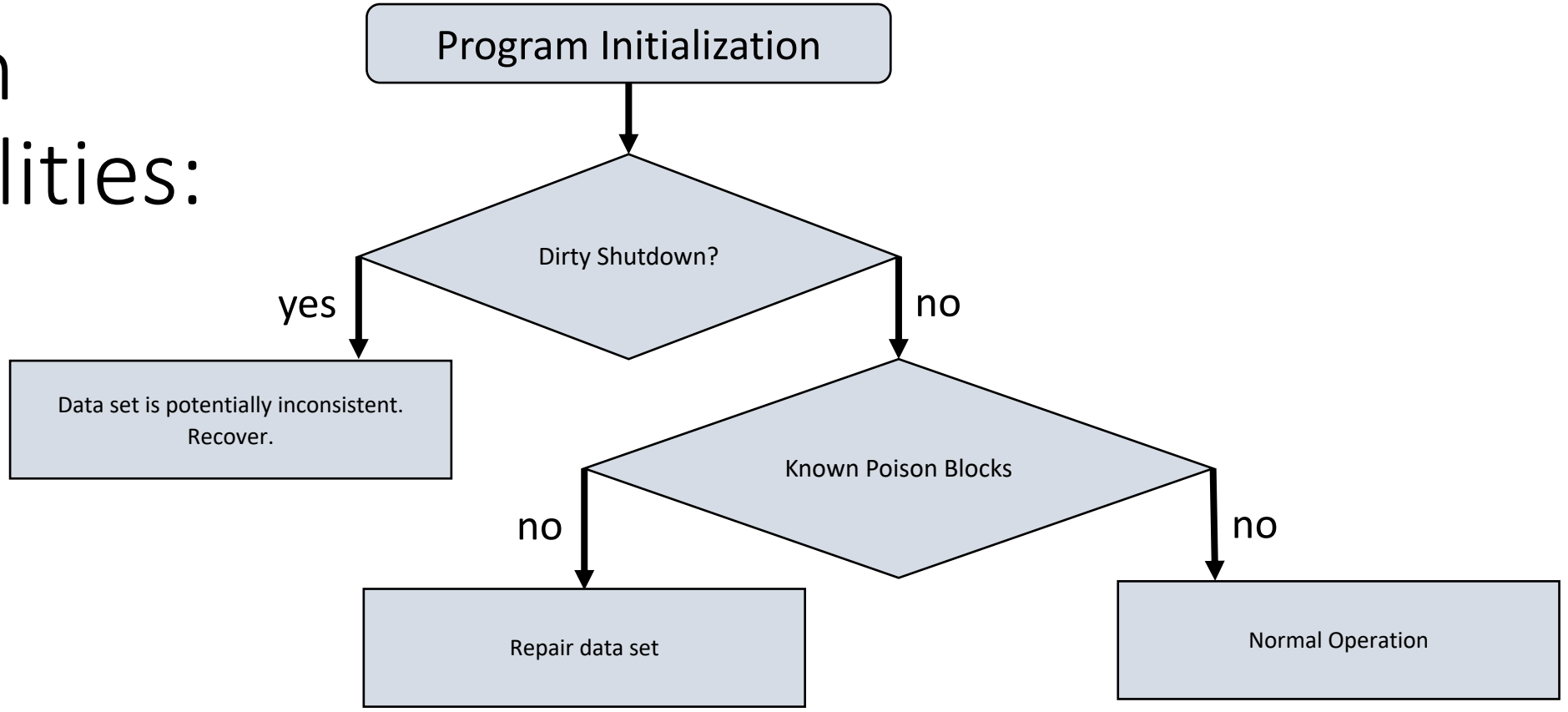
Not shown:
MCA
ADR Failure Detection



Application Responsibilities: Flushing



Application Responsibilities: Recovery



The Persistent Memory Development Kit PMDK http://pmem.io

- PMDK is a collection of libraries
  - Developers pull only what they need
    - Low level programming support
    - Transaction APIs
  - Fully validated
  - Performance tuned.
- Open Source & Product neutral

PMDK Libraries

PCJ – Persistent Collection for Java

C++ | C | PCJ/ LLPL | Python

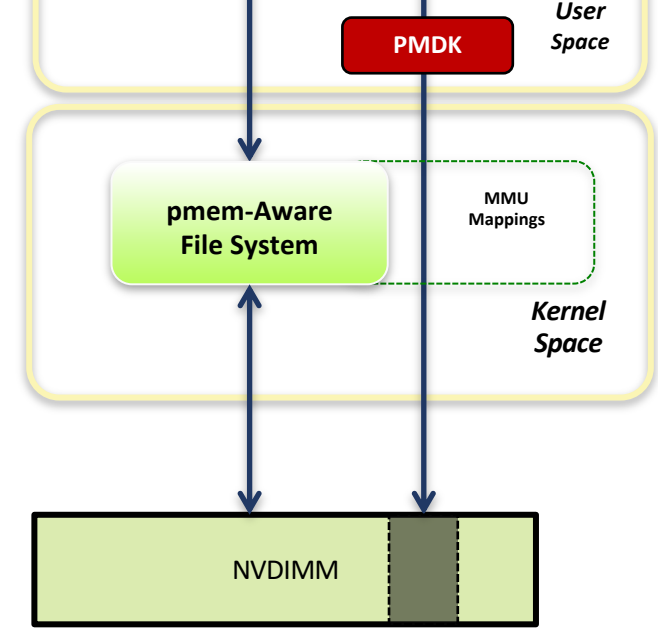
pmemkv

Transaction Support

Interface to create a persistent memory resident log file — libpmemlog

Interface for persistent memory allocation, transactions and general facilities — libpmemobj

Interface to create arrays of pmem-resident blocks, of same size, atomically updated — libpmemblk



Support for **volatile** memory usage — memkind

Low level support for local persistent memory — libpmem

Low level support for remote access to persistent memory — librpmem

Low-level support

In Development

libpmem examples

Source: https://github.com/pmem/pmdk/tree/master/src/examples/libpmem

```
/*
 * simple_copy.c -- show how to use pmem_memcpy_persist()
 *
 * usage: simple_copy src-file dst-file
 *
 * Reads 4k from src-file and writes it to dst-file.
 */
```

```
/* create a pmem file and memory map it */
if ((pmemaddr = pmem_map_file(argv[2], BUF_LEN,
                        PMEM_FILE_CREATE|PMEM_FILE_EXCL,
                        0666, &mapped_len, &is_pmem)) == NULL) {
        perror("pmem_map_file");
        exit(1);
}
```

Using `is_pmem`

```
if (is_pmem) {
        pmem_memcpy_persist(pmemaddr, buf, cc);
} else {
        memcpy(pmemaddr, buf, cc);
        pmem_msync(pmemaddr, cc);
}
```

libpmemblk Examples

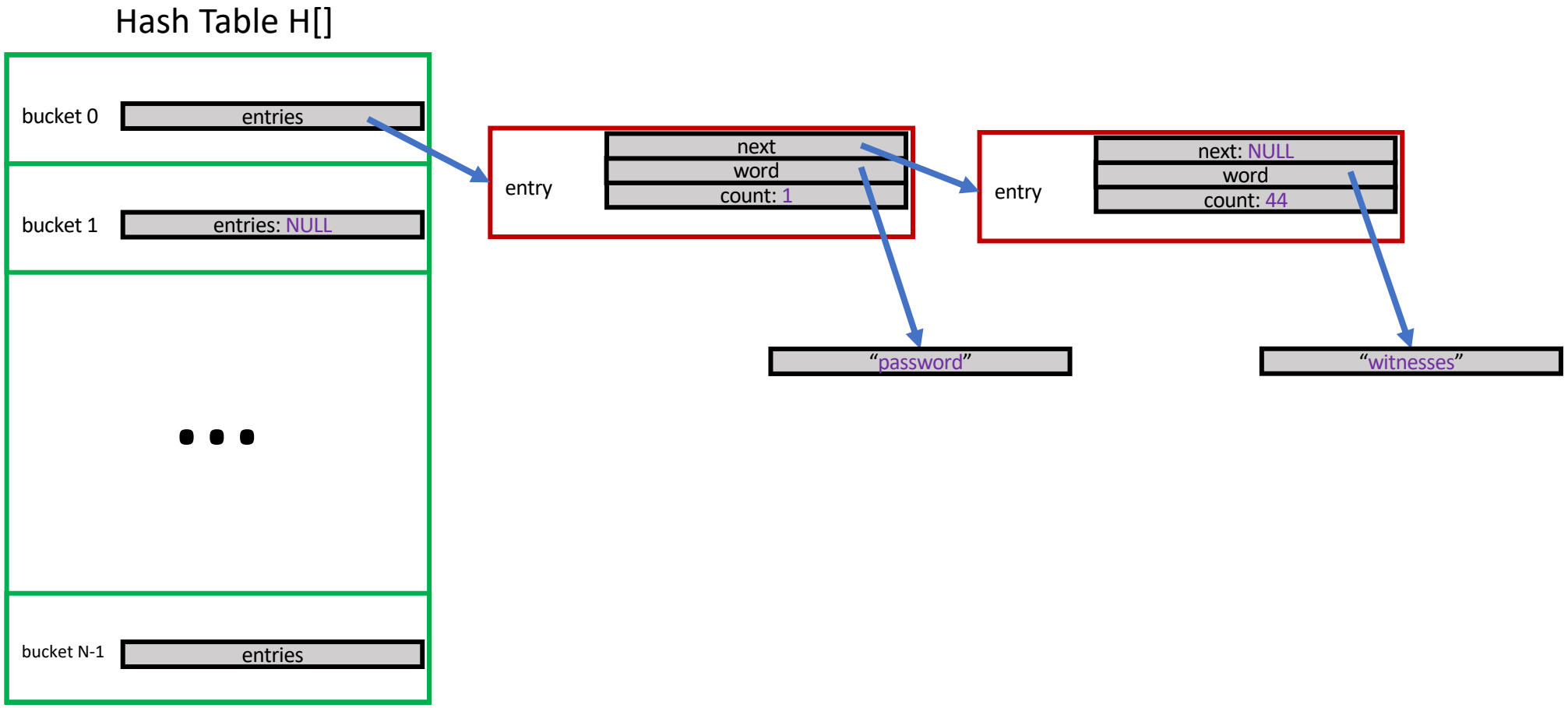
Source: https://github.com/pmem/pmdk/blob/master/src/examples/libpmemblk

```
/* store a block at index 5 */
strcpy(buf, "hello, world");
if (pmemblk_write(pbp, buf, 5) < 0) {
        perror("pmemblk_write");
        exit(1);
}

/* read the block at index 10 (reads as zeros initially) */
if (pmemblk_read(pbp, buf, 10) < 0) {
        perror("pmemblk_read");
        exit(1);
}

/* zero out the block at index 5 */
if (pmemblk_set_zero(pbp, 5) < 0) {
        perror("pmemblk_set_zero");
        exit(1);
}
```

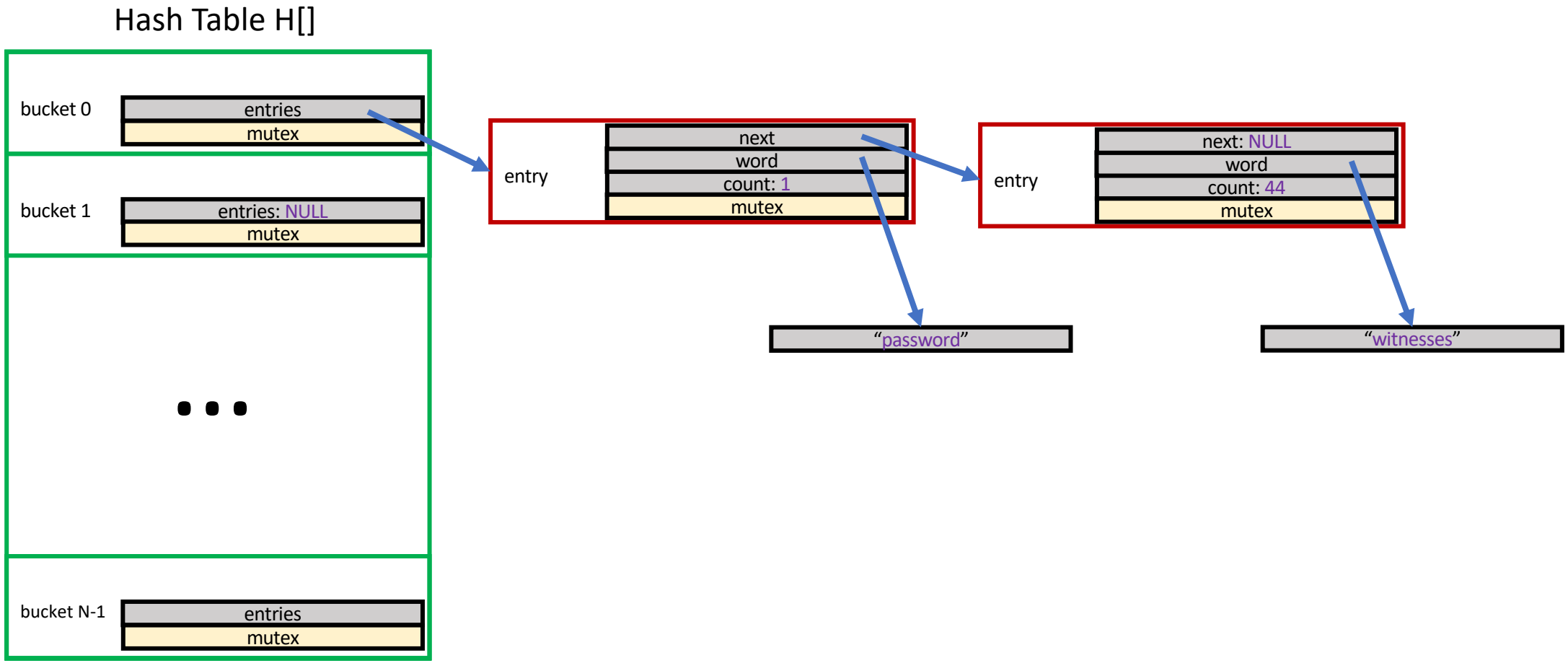
libpmemobj Examples

This is the most flexible of the PMDK libraries, supporting general-purpose allocation & transactions

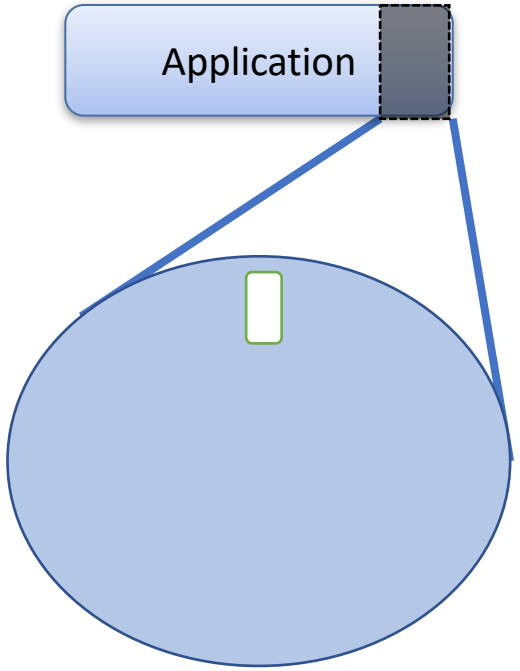
Simple C Example to Build On (no pmem yet)



Adding Multi-thread Support (no pmem yet)



The Root Object

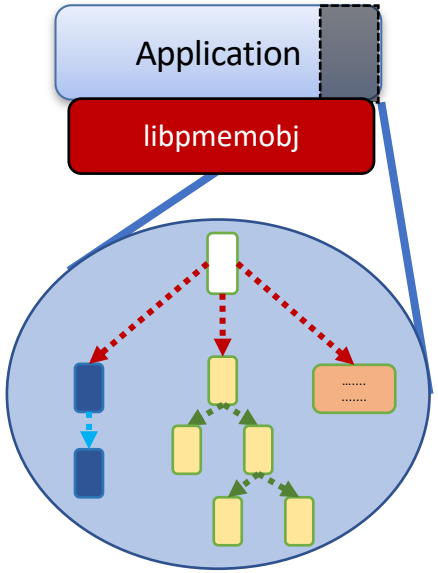


pmem pool “myfile”

root object:

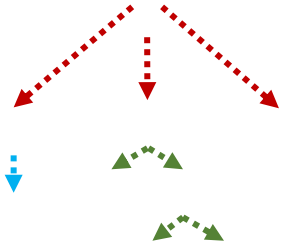
- assume it is always there
- created first time accessed
- initially zeroed

Using the Root Object

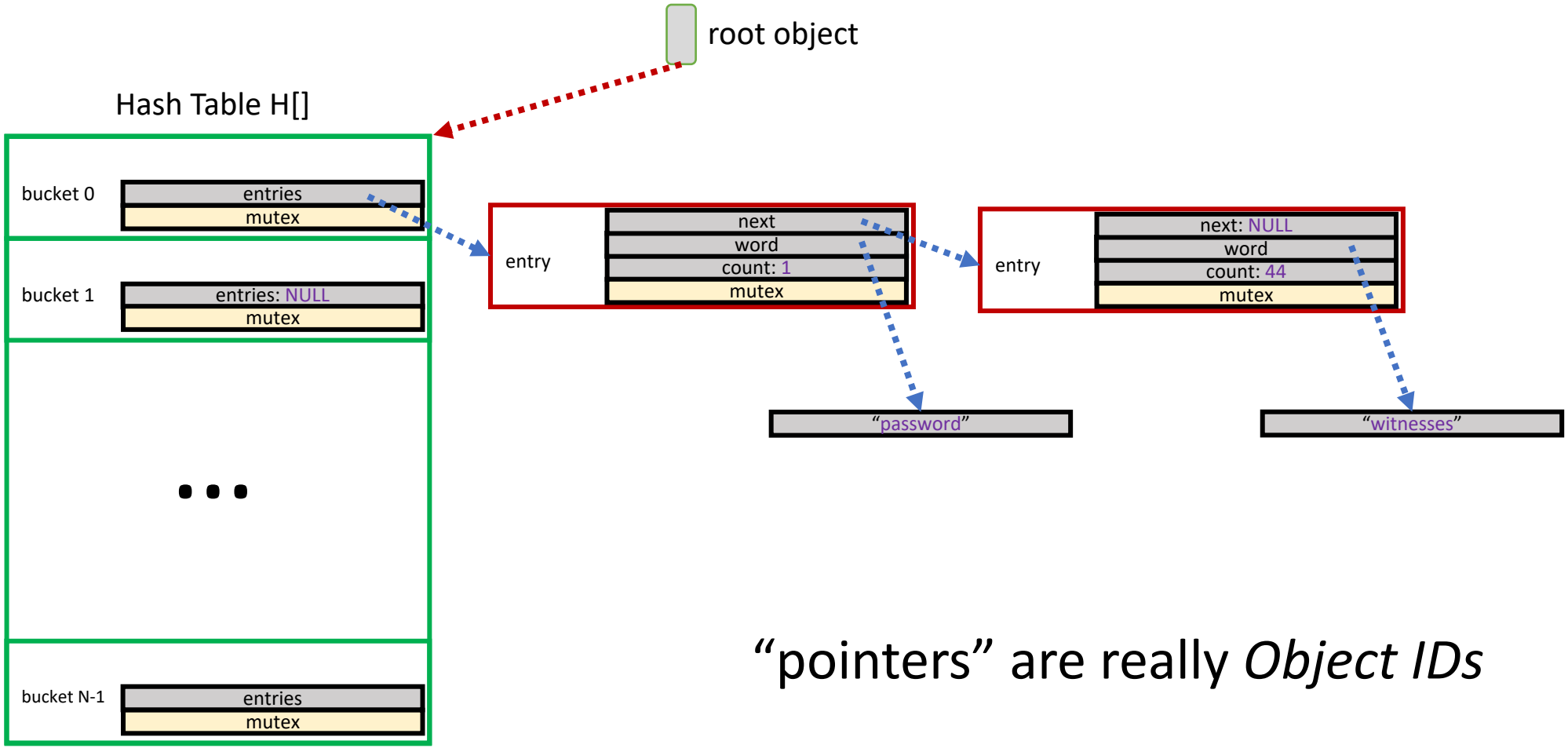


Link pmem data structures in pool off the root object to find them on each program run

"pointers" are really *Object IDs*



Moving the Example to pmem



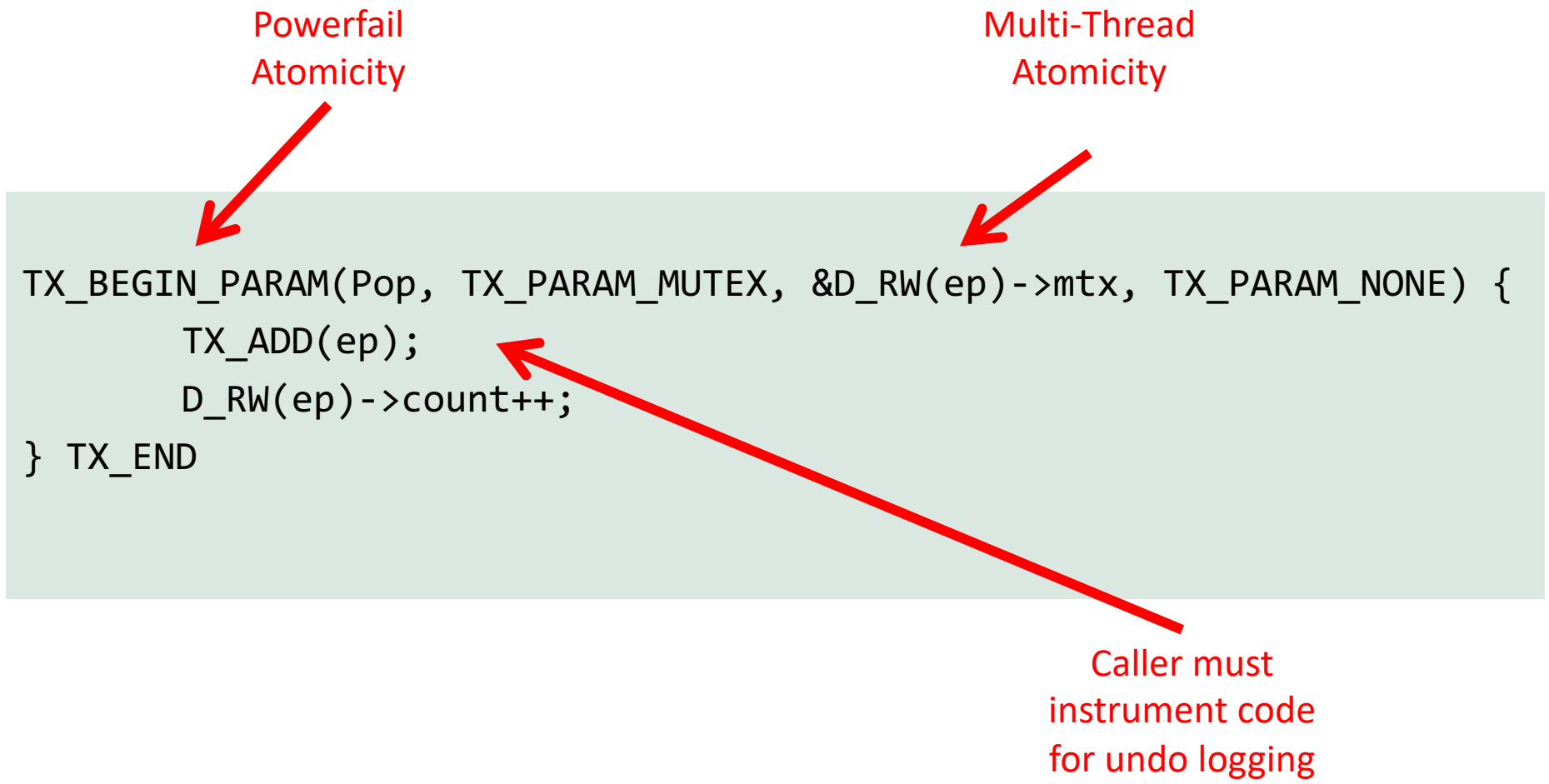
"pointers" are really *Object IDs*

C Programming with libpmemobj

Transaction Syntax

```
TX_BEGIN(Pop) {
                    /* the actual transaction code goes here... */
} TX_ONCOMMIT {
                    /*
                     * optional – executed only if the above block
                     * successfully completes
                     */
} TX_ONABORT {
                    /*
                     * optional – executed if starting the transaction fails
                     * or if transaction is aborted by an error or a call to
                     * pmemobj_tx_abort()
                     */
} TX_FINALLY {
                    /*
                     * optional – if exists, it is executed after
                     * TX_ONCOMMIT or TX_ONABORT block
                     */
} TX_END /* mandatory */
```

Properties of Transactions



Persistent Memory Locks

- Want locks to live near the data they protect (i.e. inside structs)
- Does the state of locks get stored persistently?
  - Would have to flush to persistence when used
  - Would have to recover locked locks on start-up
    - Might be a different program accessing the file
  - Would run at pmem speeds

- PMEMmutex
  - Runs at DRAM speeds
  - Automatically initialized on pool open

C++ Programming with libpmemobj

C++ Queue Example: Declarations

```
/* entry in the queue */
struct pmem_entry {
    persistent_ptr<pmem_entry> next;
    p<uint64_t> value;
};
```

`persistent_ptr<T>`	Pointer is really a position-independent Object ID in pmem. Gets rid of need to use C macros like D_RW()
`p<T>`	Field is pmem-resident and needs to be maintained persistently. Gets rid of need to use C macros like TX_ADD()

C++ Queue Example: Transaction

```
void push(pool_base &pop, uint64_t value) {
    transaction::run(pop, [&] {
        auto n = make_persistent<pmem_entry>();

        n->value = value;
        n->next = nullptr;
        if (head == nullptr) {
            head = tail = n;
        } else {
            tail->next = n;
            tail = n;
        }
    });
}
```

Transactional
(including allocations & frees)

Links to More Information

Using pmem As Volatile Memory

- Bigger/cheaper than DRAM
- Application decides what lives in DRAM, what lives in persistent memory
  - Unlike Memory Mode, where HW decides
- Similar to NUMA programming
  - app allocates different “kinds” of memory
- memkind library: http://memkind.github.io/memkind/
  - Familiar malloc/free style programming with multiple pools
    - NUMA nodes, HBM, etc.
  - Can construct pools with persistent memory

libpmemkv

- https://github.com/pmem/pmemkv
- General-purpose key-value store
  - Simple API, handles pmem transactions, etc so caller doesn't need to
  - Multiple storage engines, tuned for pmem
  - Multiple language bindings: C, C++, Java, Ruby, JavaScript

- Still "experimental" – in the process of validating to product quality

More Developer Resources

- Find the PMDK (Persistent Memory Development Kit) at http://pmem.io/pmdk/
- Getting Started
  - Intel IDZ persistent memory- https://software.intel.com/en-us/persistent-memory
  - Entry into overall architecture - http://pmem.io/2014/08/27/crawl-walk-run.html
  - Emulate persistent memory - http://pmem.io/2016/02/22/pm-emulation.html
- Linux Resources
  - Linux Community Pmem Wiki - https://nvdimm.wiki.kernel.org/
  - Pmem enabling in SUSE Linux Enterprise 12 SP2 - https://www.suse.com/communities/blog/nvdimm-enabling-suse-linux-enterprise-12-service-pack-2/
- Windows Resources
  - Using Byte-Addressable Storage in Windows Server 2016 -https://channel9.msdn.com/Events/Build/2016/P470
  - Accelerating SQL Server 2016 using Pmem - https://channel9.msdn.com/Shows/Data-Exposed/SQL-Server-2016-and-Windows-Server-2016-SCM--FAST
- Other Resources
  - SNIA Persistent Memory Summit 2018 - https://www.snia.org/pm-summit
  - Intel manageability tools for Pmem - https://01.org/ixpdimm-sw/

Intel Developer Support & Tools

- **PMDK Tools**
  - Valgrind plugin: pmemcheck
  - Debug mode, tracing, pmembench, pmreorder

pmem.io

- **New features to support Intel® Optane™ DC persistent memory**
  - Intel® VTune™ Amplifier – Performance Analysis
  - Intel® Inspector – Persistence Inspector finds missing cache flushes & more
  - Free downloads available

software.intel.com/pmem

Hack, hack, hack…