

Persistent Memory Programming Hackathon

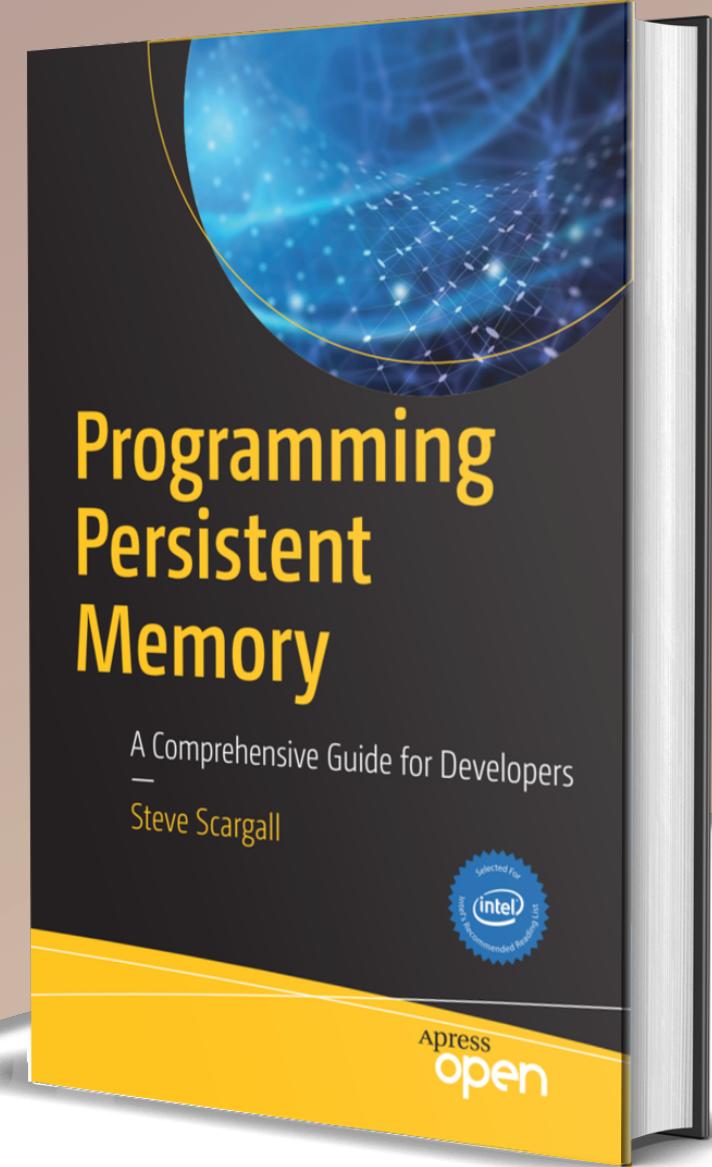
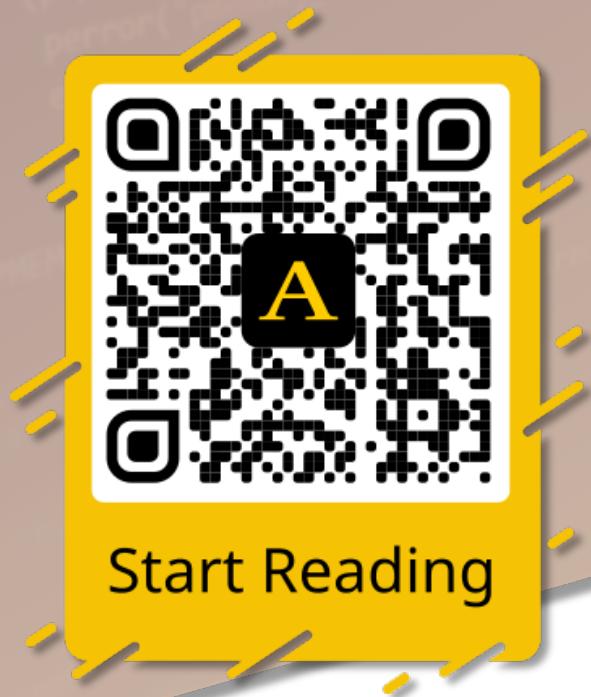
PM Summit 2020

<https://github.com/pmemhackathon/2020-01-22>

```
51 int  
52 main(int argc, char *argv[]){  
53 {  
54     if (argc != 2) {  
55         printf("Usage: %s <file-name>\n", argv[0]);  
56         exit(1);  
57     }  
58     FILE *f = fopen(argv[1], "r");  
59     if (!f) {  
60         perror("Error opening file");  
61         exit(1);  
62     }  
63     /* ... */  
64 }
```

Master Persistent Memory Programming

Are you ready to begin?

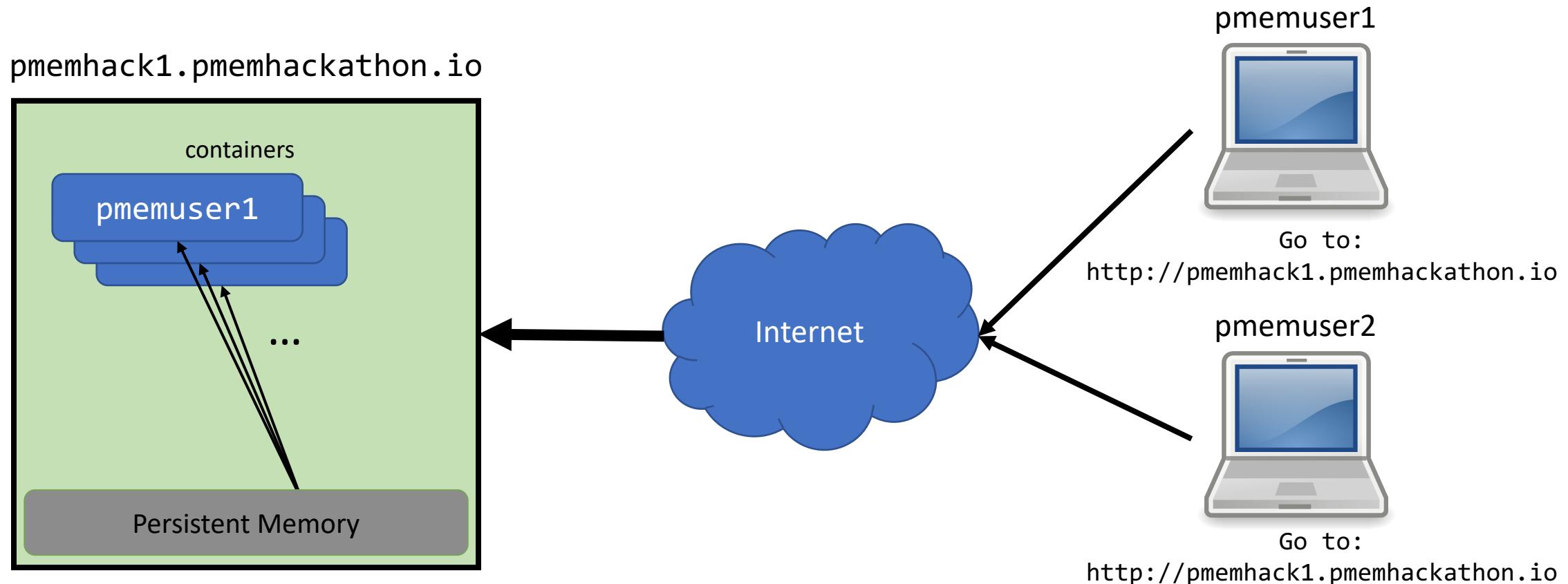


<https://www.apress.com/us/book/9781484249314> ²

Agenda

- Essential Background Slides, covering:
 - Logistics: how you access persistent memory from your laptop
 - The minimum you need to know about persistent memory
 - Walk through the first example or two together
- Goal is to get you hands-on with pmem programming quickly
 - All slides and examples are in the repo
 - Lots more detail in additional slide decks in the repo

Logistics: The *webhackathon* Tool



Username and Password handed out to each attendee

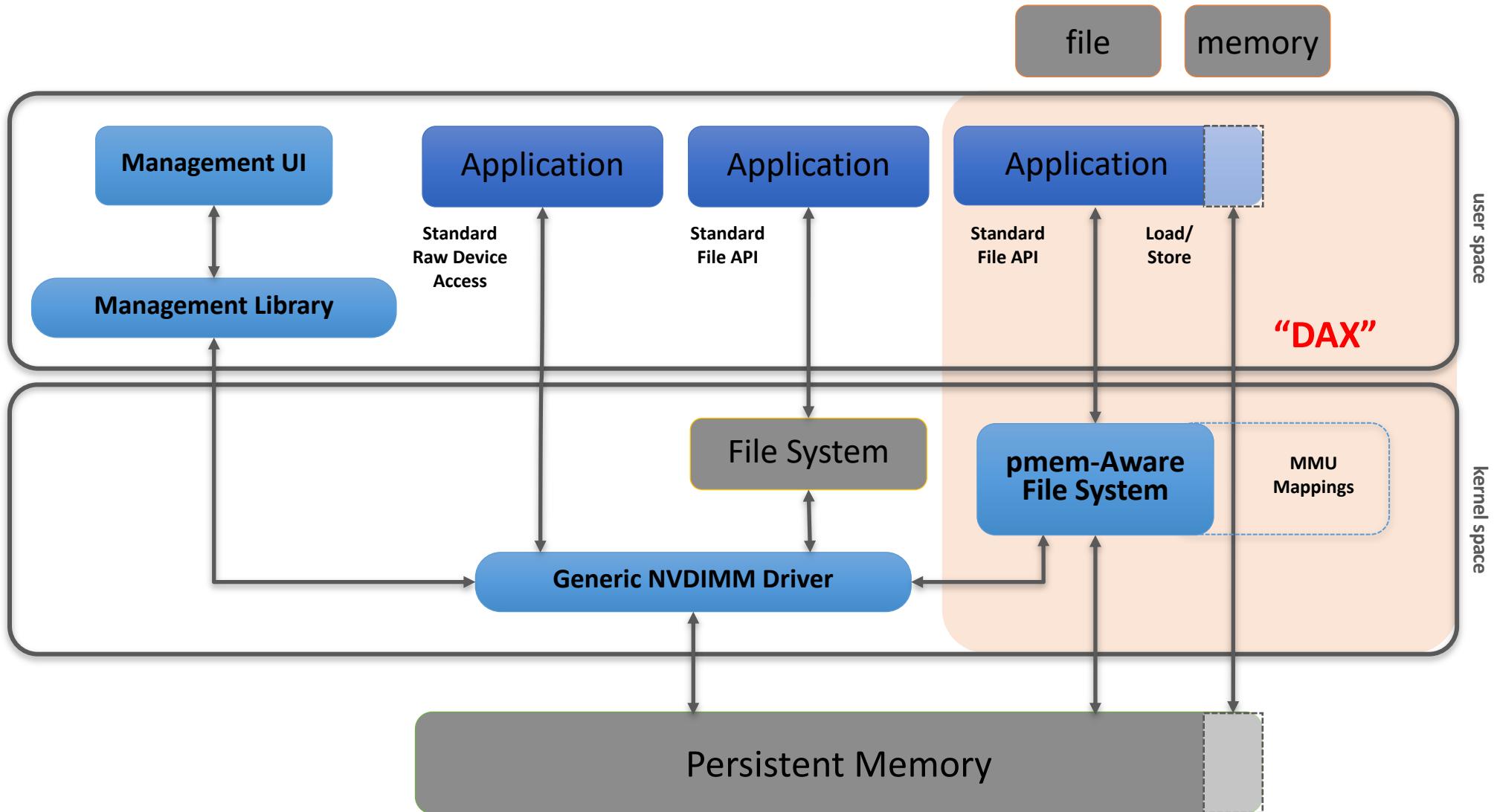
Webhackathon Basics

- List of examples presented on main page
 - First three recommended to provide essential background
 - We will walk through some of these together
 - Pick examples that are interesting to you (task, language, etc)
 - Use them as a starting point for your own code
 - Menu provides:
 - Access to these background slides
 - Browse your copy of the repo (to download something you want to keep)
 - Browser-based shell window for your container (for users who need it)
 - Everything you do runs in your own container on the server
 - With your own copy of the hackathon repo
 - The path to the persistent memory is /pmem
 - We're all friends here: please no denial-of-service attacks on server!

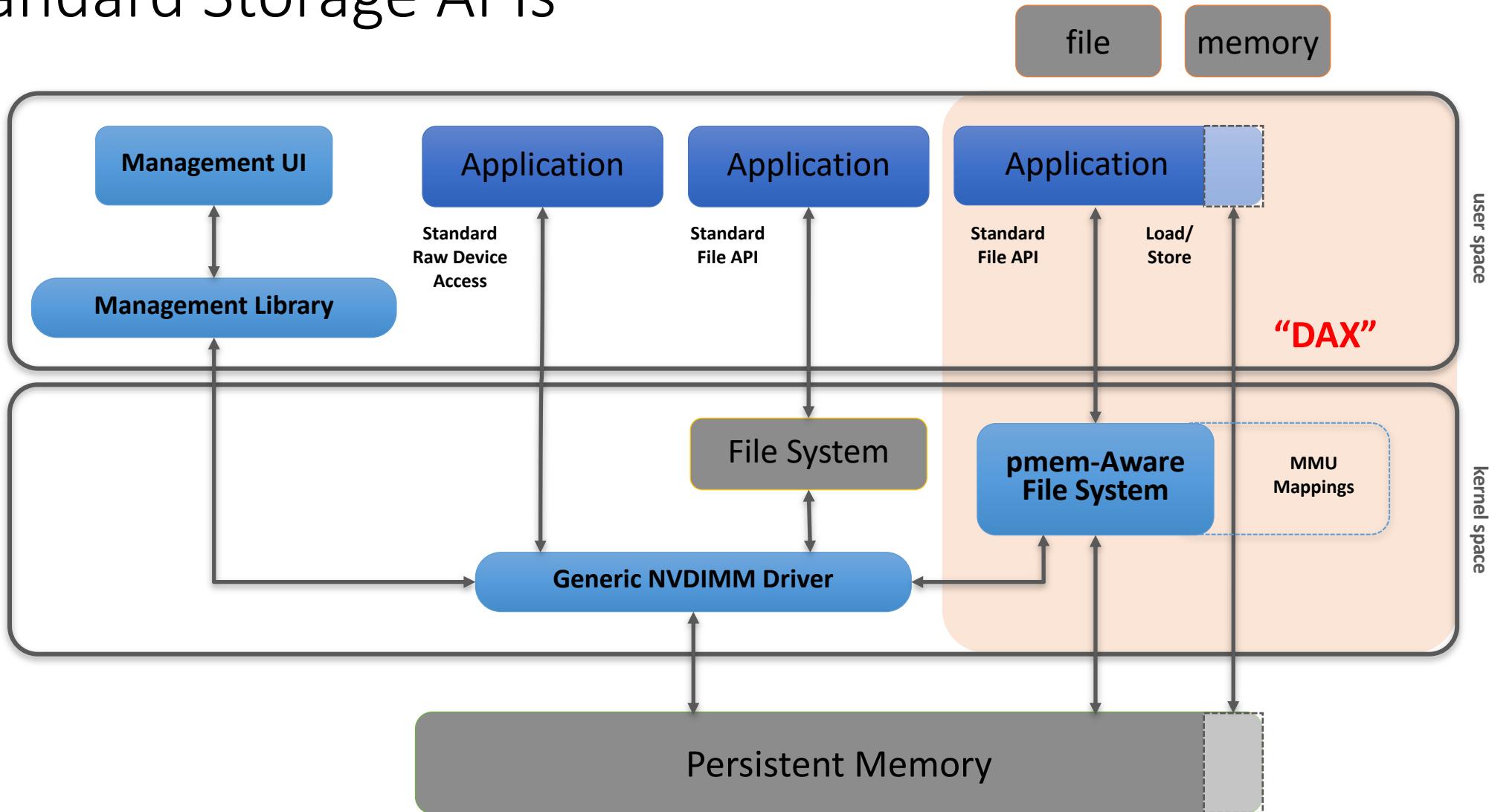
Essential pmem Programming Background

- 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

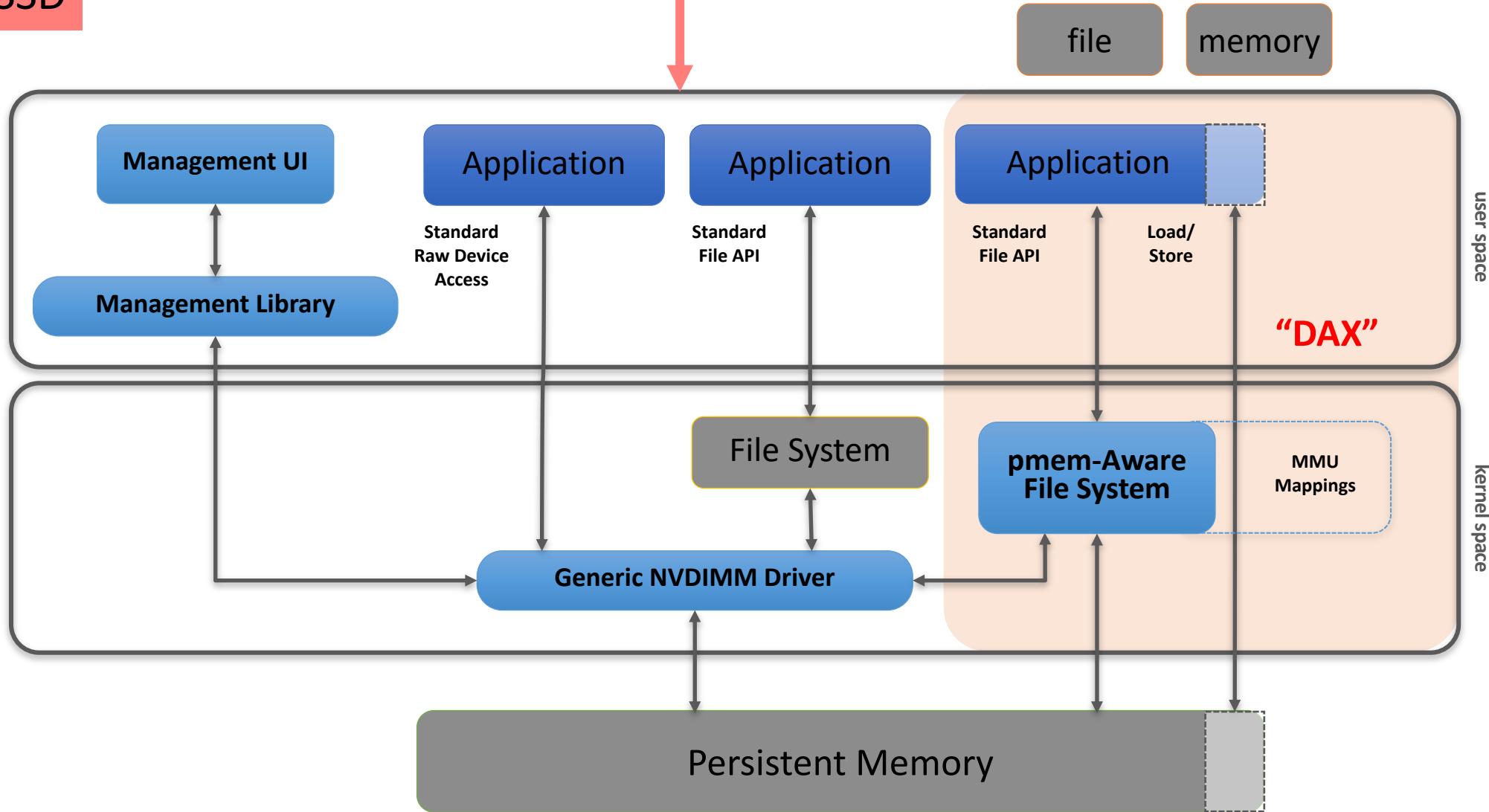
The SNIA NVM Programming Model



Don't Forget: The NVM Programming Model Starts With Standard Storage APIs

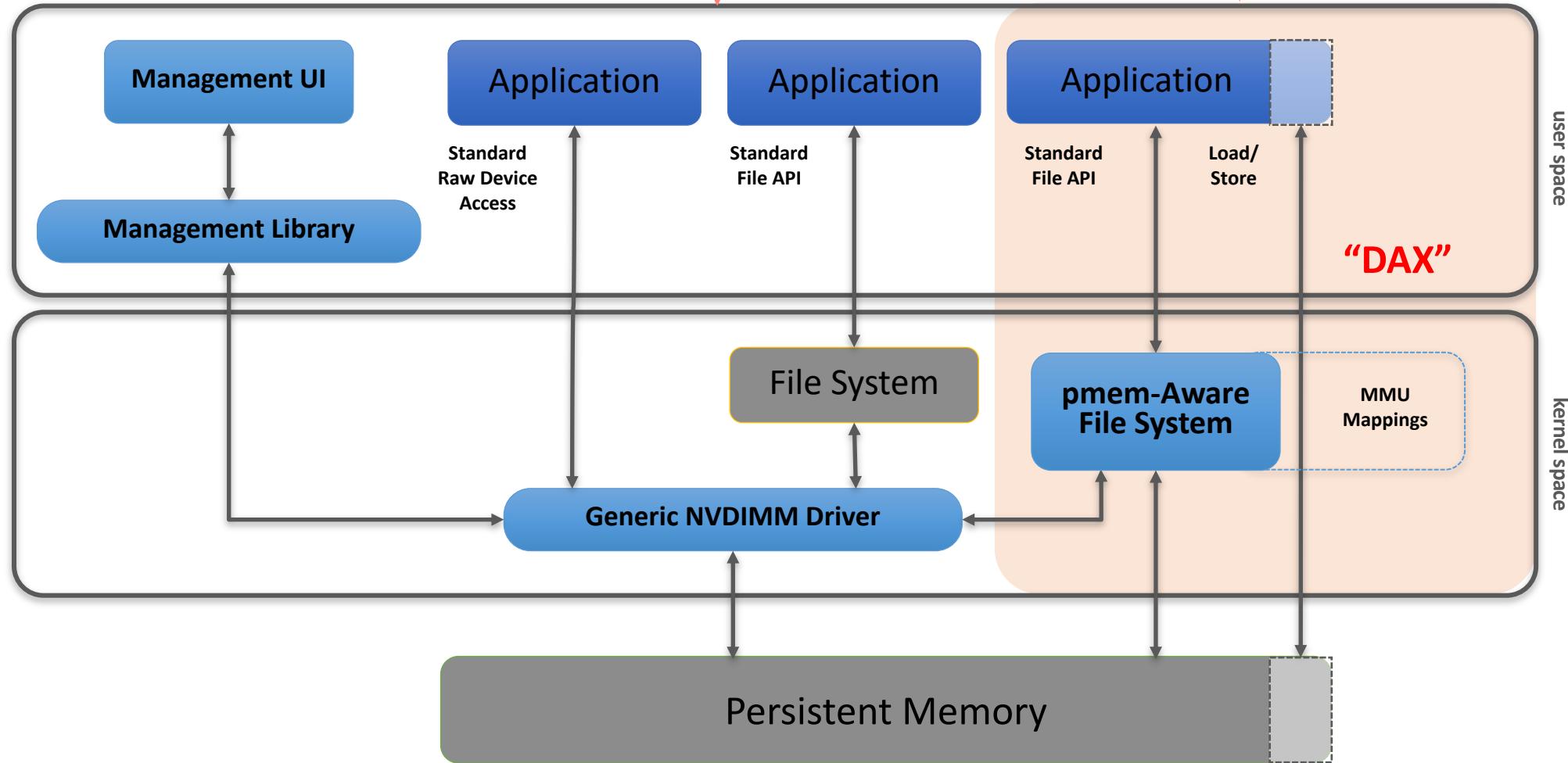


Use PM
Like an SSD



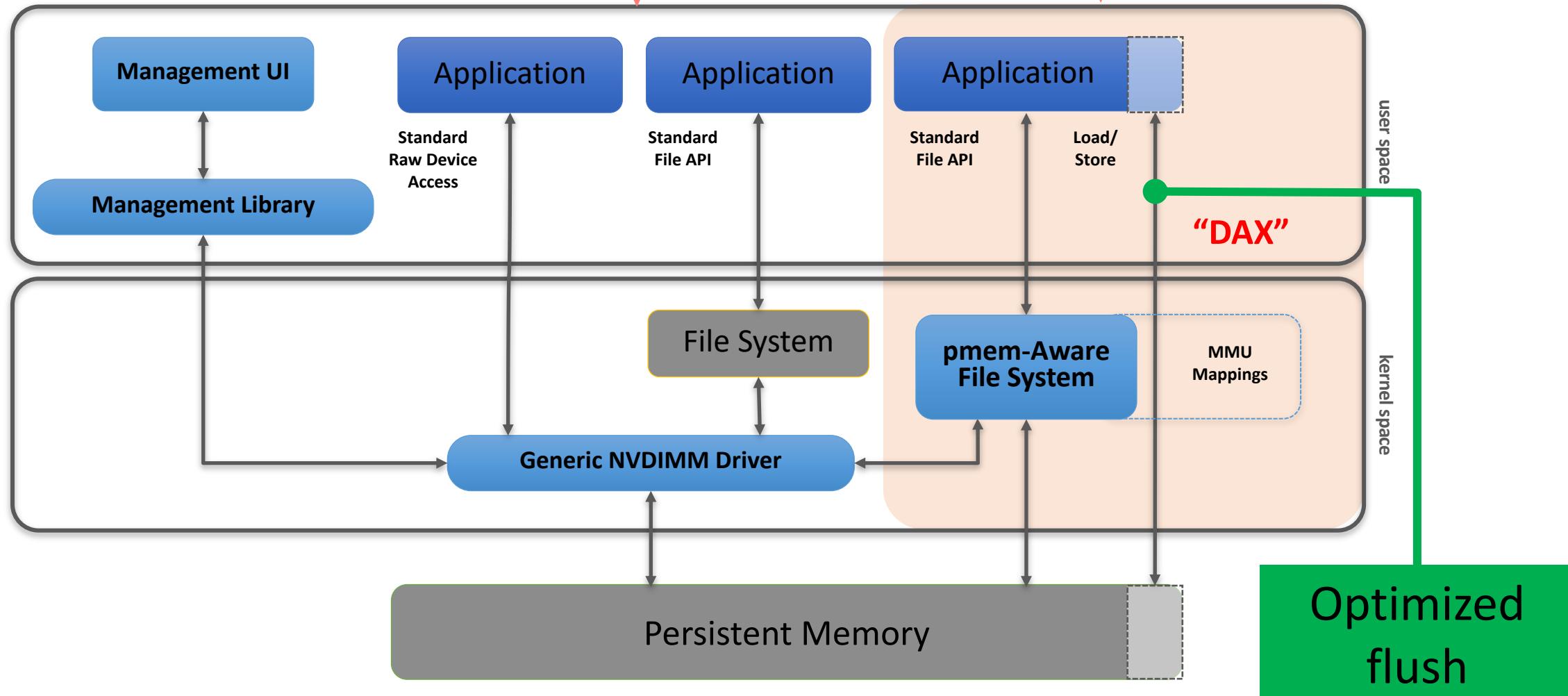
Use PM
Like an SSD

Use PM
Like an SSD
(no page cache)

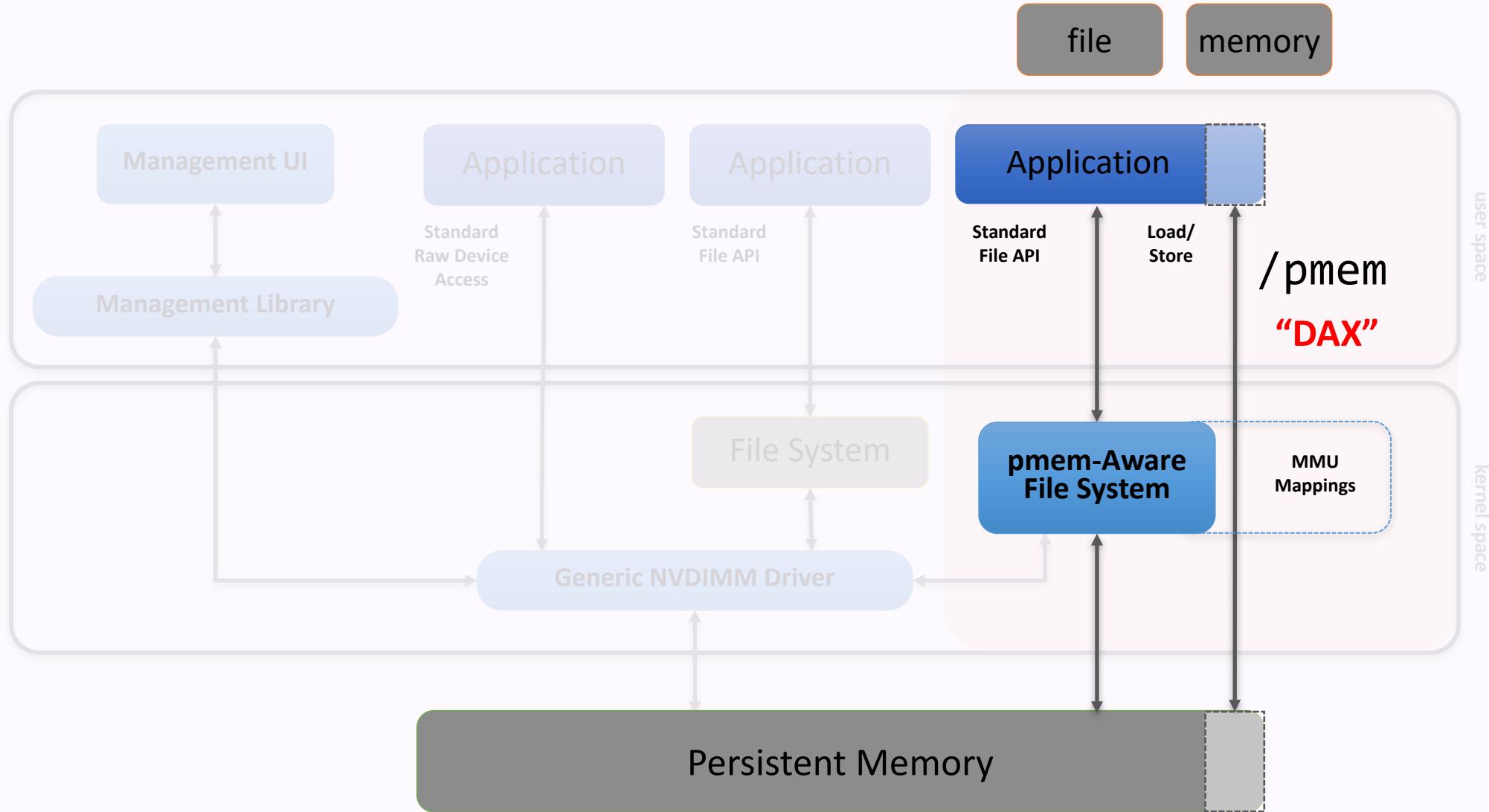


Use PM
Like an SSD

Use PM
Like an SSD
(no page cache)

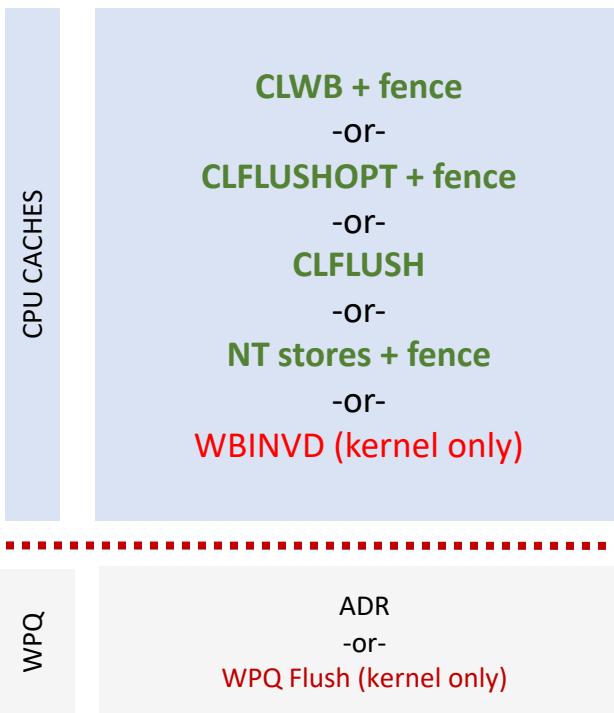
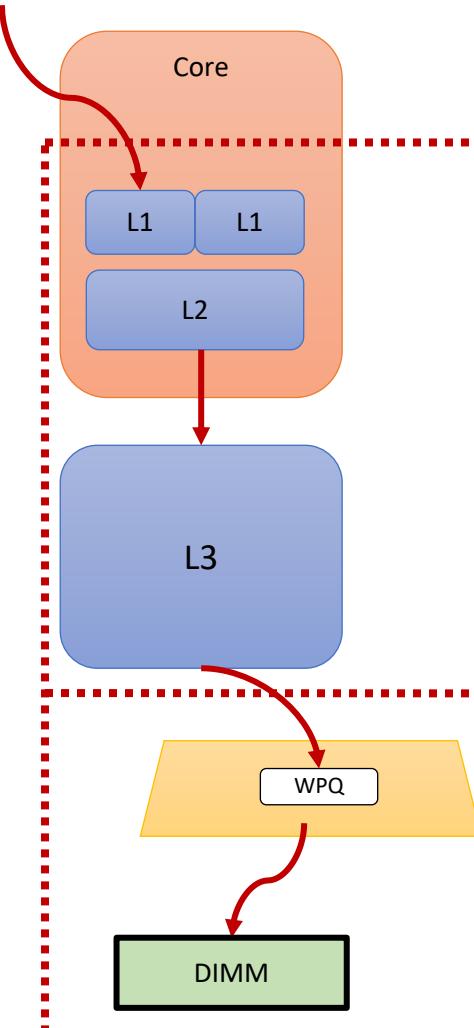


Today's focus



How the Hardware Works

MOV

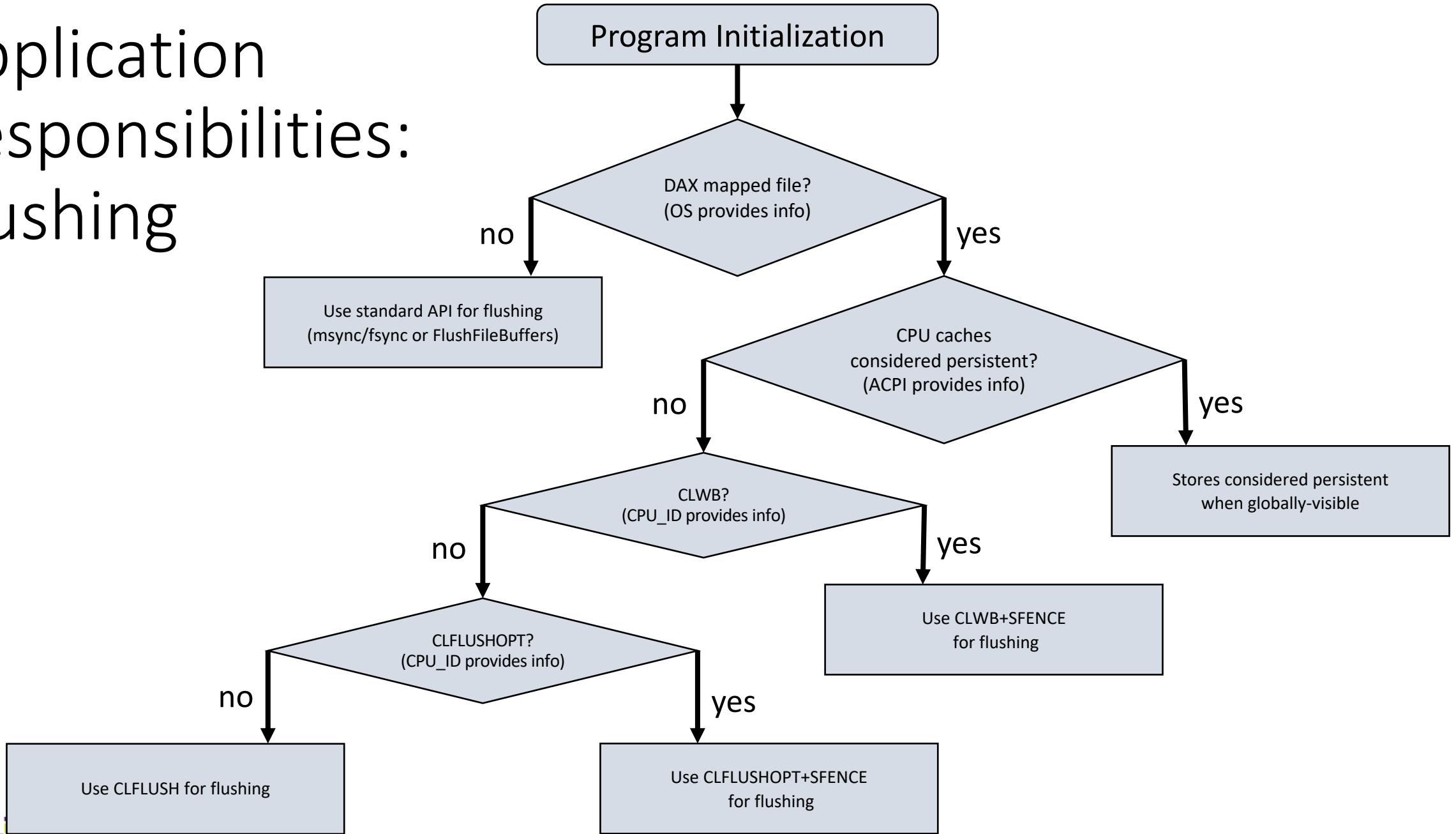


Custom Power fail protected domain
indicated by ACPI property:
CPU Cache Hierarchy

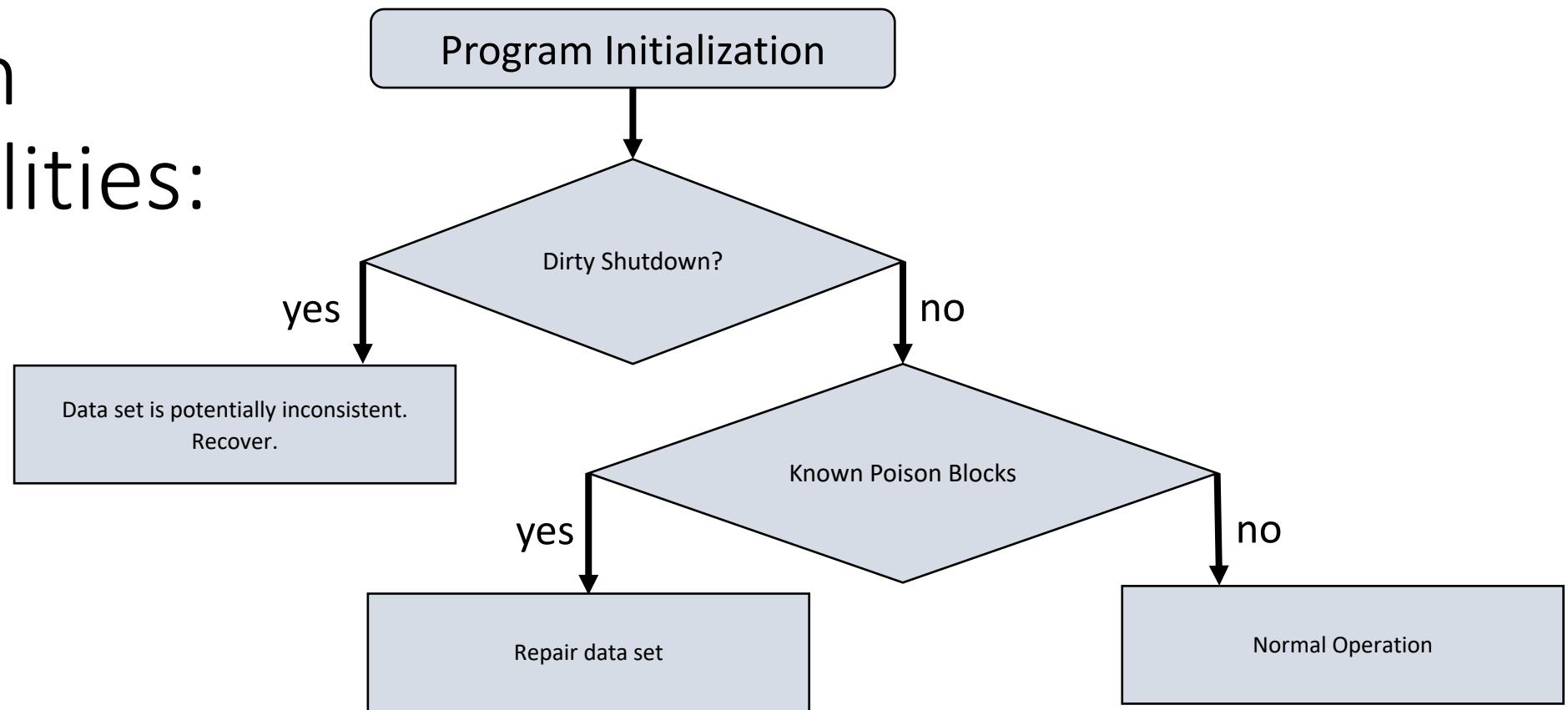
Minimum Required Power fail protected domain:
Memory subsystem

Not shown:
MCA
ADR Failure Detection

Application Responsibilities: Flushing



Application Responsibilities: Recovery



Application Responsibilities: Consistency

```
open(...);  
  
mmap(...);  
  
strcpy(pmemp, "Hello, World!");  
  
msync(...);
```

← Crash

Result

1. "\0\0\0\0\0\0\0\0\0\0..."
2. "Hello, \0\0\0\0\0\0..."
3. "\0\0\0\0\0\0\0\0\0orld!\0"
4. "Hello, \0\0\0\0\0\0\0\0\0"
5. "Hello, World!\0"

Application Responsibilities: Consistency

```
open(...);  
  
mmap(...);  
  
strcpy(pmemp, "Hello, World!");  
  
pmem_persist(pmemp, 14);
```

`pmem_persist()` may be faster,
but is still **not** transactional

← Crash

Result

1. "\0\0\0\0\0\0\0\0\0\0..."
2. "Hello, \0\0\0\0\0\0..."
3. "\0\0\0\0\0\0\0\0\0orld!\0"
4. "Hello, \0\0\0\0\0\0\0\0\0"
5. "Hello, World!\0"

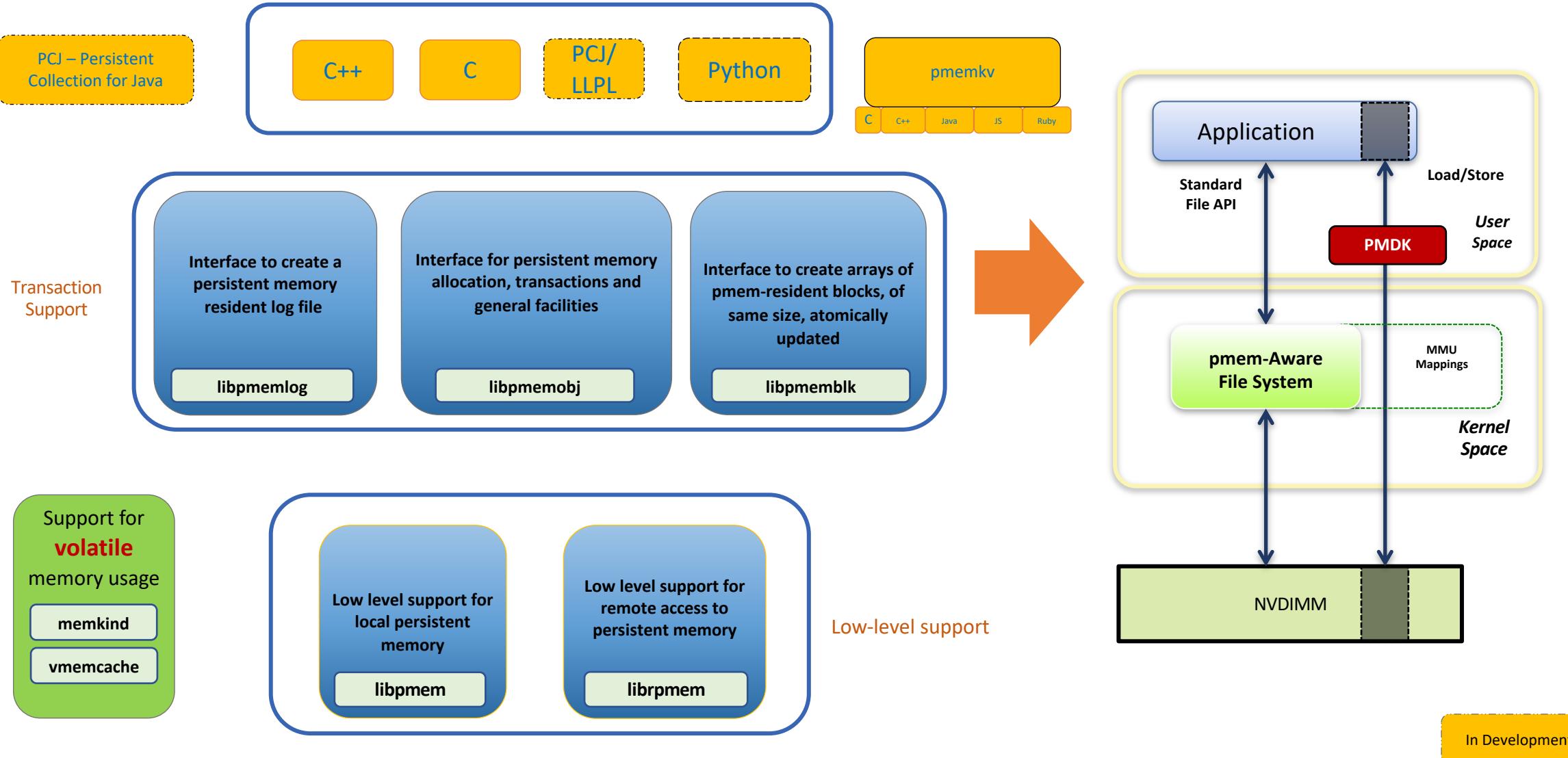
The Persistent Memory Development Kit

PMDK <http://pmem.io>

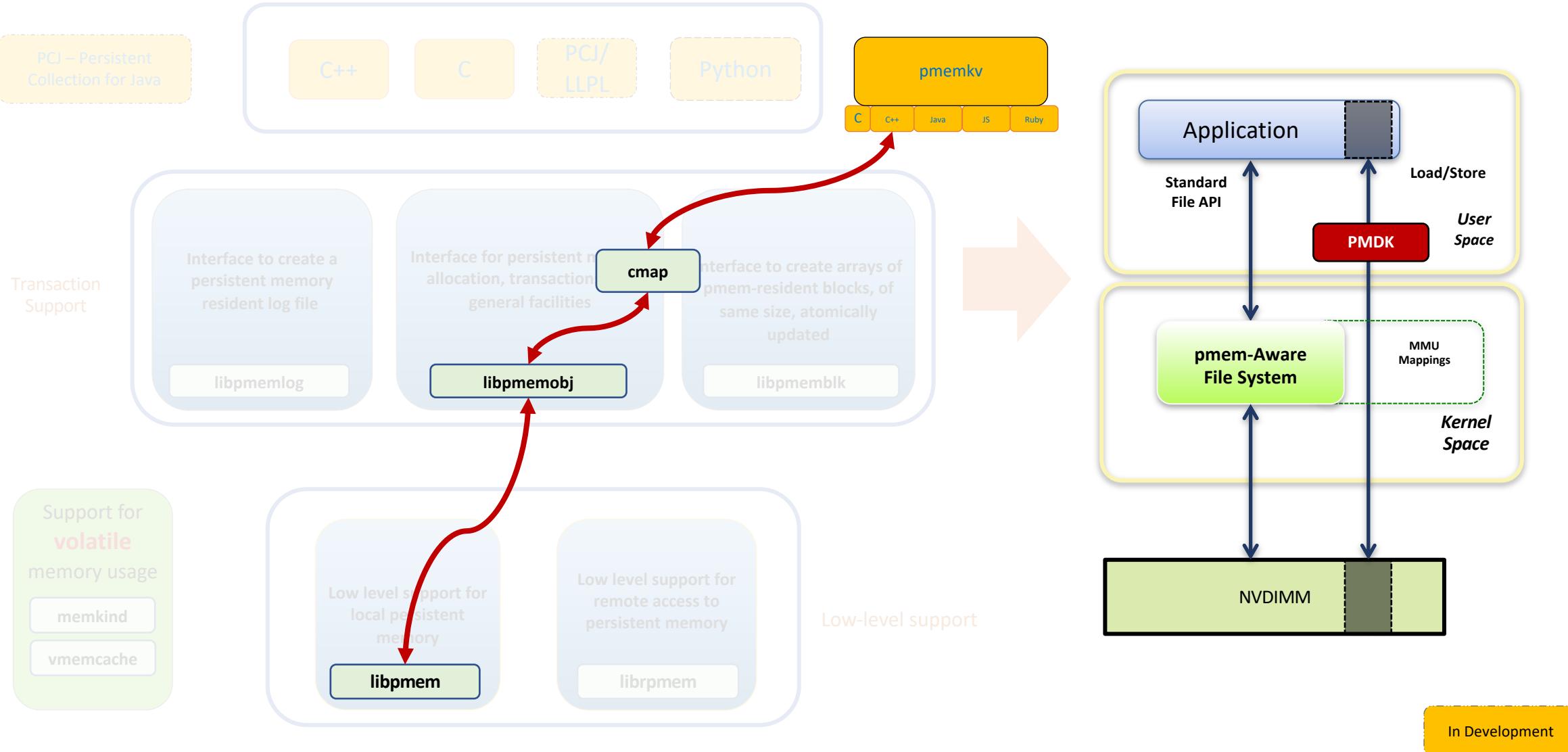
- PMDK is a collection of libraries
 - Developers pull only what they need
 - Low level programming support
 - Transaction APIs
 - Fully validated
 - Performance tuned.
- Open Source & Product neutral



PMDK Libraries

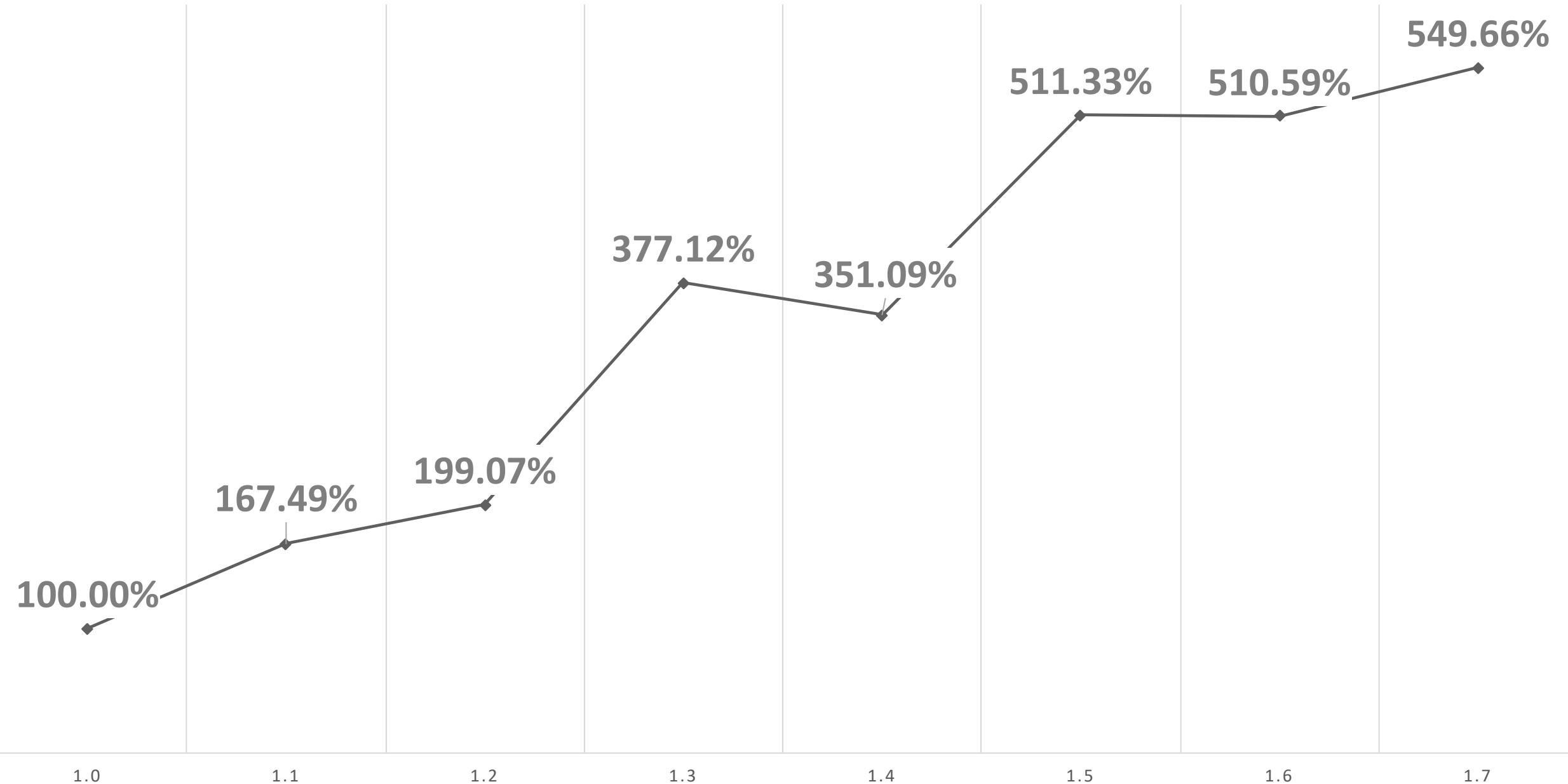


PMDK Libraries



libpmemobj Relative Performance Across Versions

(B-Tree benchmark)



Hack, hack, hack...

<http://pmemhack1.pmemhackathon.io>

Click [Request Access](#) to get a login

More Background Information

Read as necessary, or just keep working through
the examples – whatever works best for you

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>

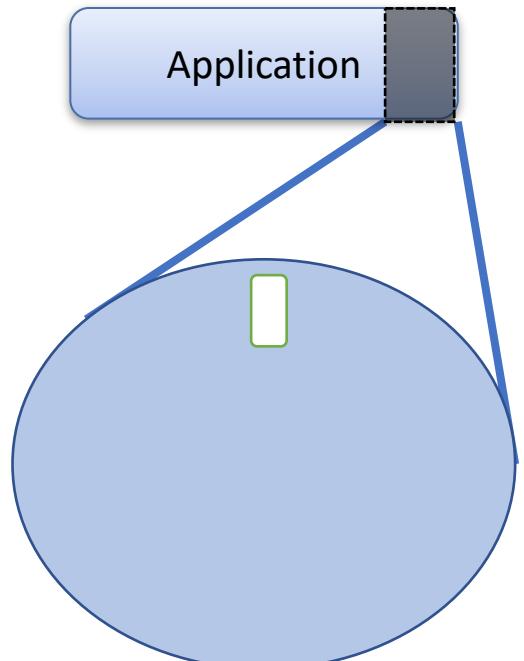
More Developer Resources

- 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/>

Basic libpmemobj Information

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

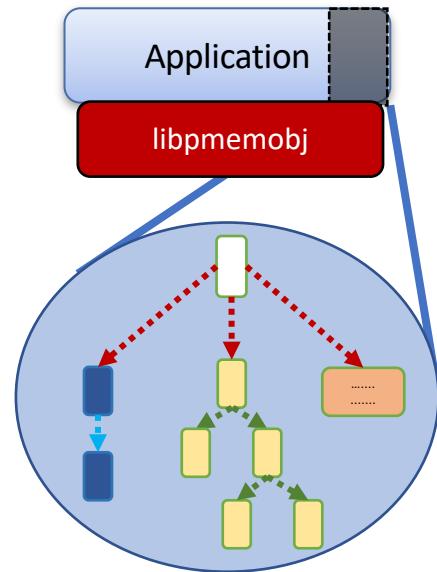
The Root Object



root object:

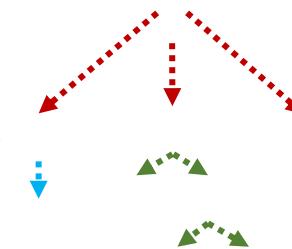
- assume it is always there
- created first time accessed
- initially zeroed

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*



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

```
Powerfail  
Atomicity  
  
TX_BEGIN_PARAM(Pop, TX_PARAM_MUTEX, &D_RW(ep)->mtx, TX_PARAM_NONE) {  
    TX_ADD(ep);  
    D_RW(ep)->count++;  
} TX_END
```

Multi-Thread
Atomicity

Caller must
instrument code
for undo logging

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)

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**

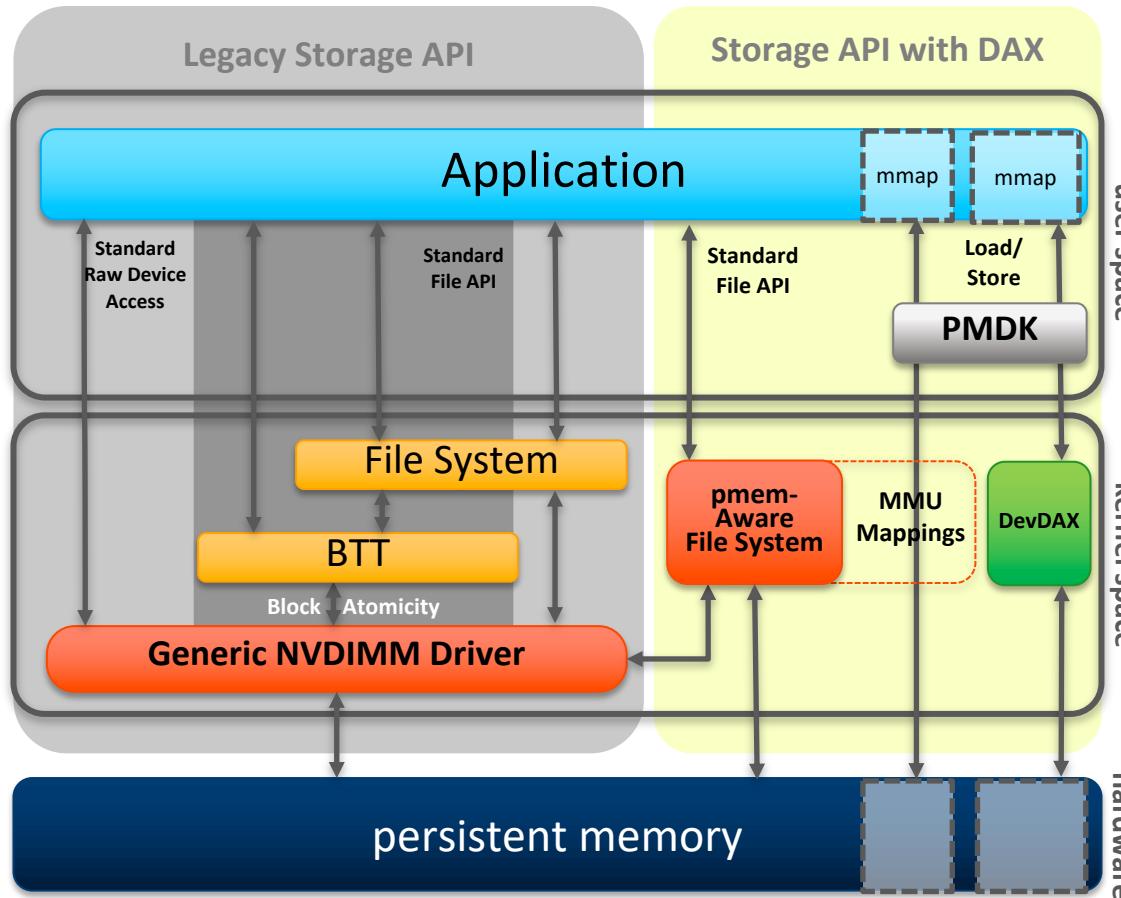
- Intel® VTune™ Amplifier – Performance Analysis
- Intel® Inspector – Persistence Inspector finds missing cache flushes & more
- Free downloads available

software.intel.com/pmem

Possible ways to access persistent memory

- No Code Changes Required
- Operates in Blocks like SSD/HDD
 - Traditional read/write
 - Works with Existing File Systems
 - Atomicity at block level
 - Block size configurable
 - 4K, 512B*
- NVDIMM Driver required
 - Support starting Kernel 4.2
 - Configured as Boot Device
 - Higher Endurance than Enterprise SSDs
 - High Performance Block Storage
 - Low Latency, higher BW, High IOPs

*Requires Linux



- Code changes may be required*
- Bypasses file system page cache
- Requires DAX enabled file system
 - XFS, EXT4, NTFS
- No Kernel Code or interrupts
- No interrupts
- Fastest IO path possible

* Code changes required for load/store direct access if the application does not already support this.

Hackathon Contributors...

- Piotr Balcer
- Eduardo Berrocal
- Jim Fister
- Stephen Bates
- Zhiming Li
- Lukasz Plewa
- Szymon Romik
- Andy Rudoff
- Steve Scargall
- Peifeng Si
- Pawel Skowron
- Usha Upadhyayula

With lots of input & feedback from others along the way...