Persistent Memory Hackathon and Workshop

Flash Memory World 2019

August 23, 2019

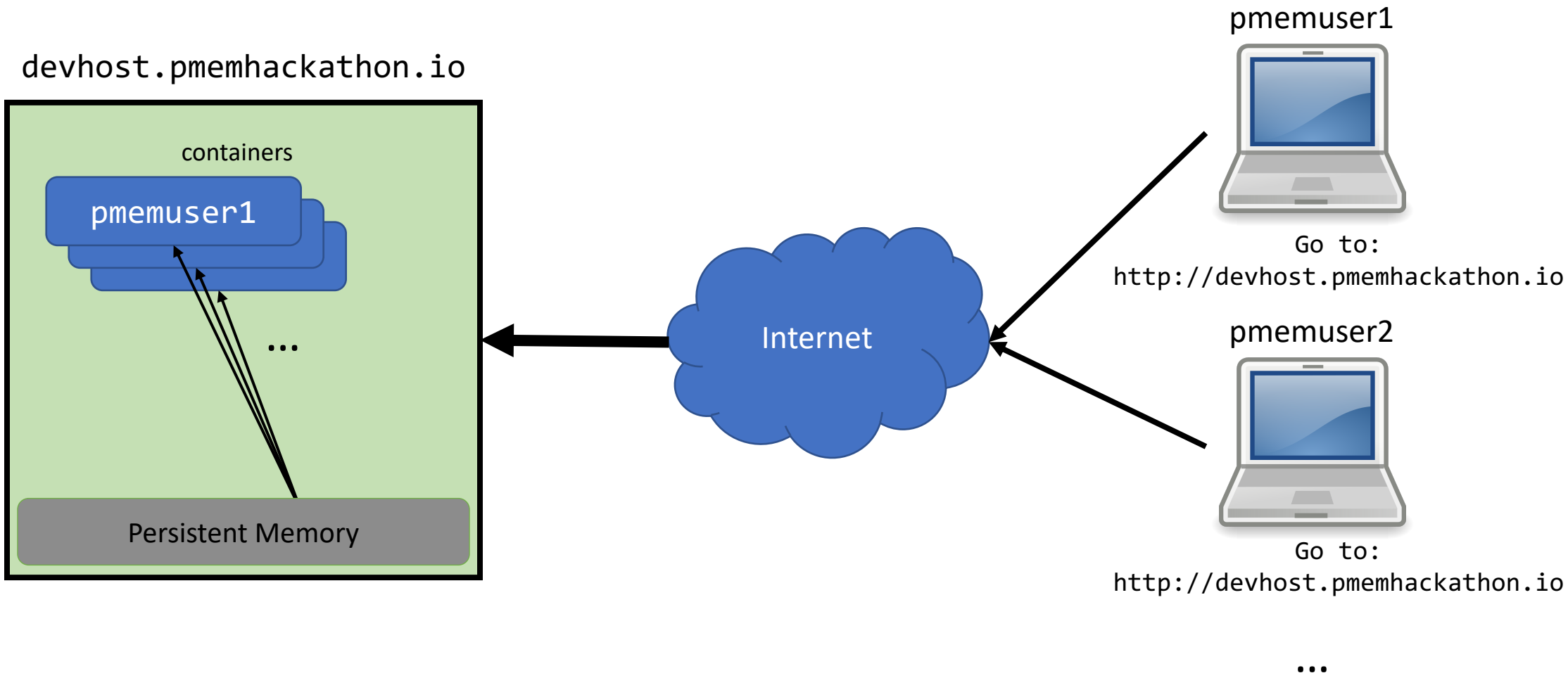
https://github.com/pmemhackathon/2019-08-23

SNIA

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
- Less talk, more hack…
  - Work your way through the examples, in any order after the first three
  - Helpers will be available in the room
  - If FAQs come up, we'll present answers to the entire room

Logistics: The *webhackathon* Tool



Username and Password handed out to each attendee

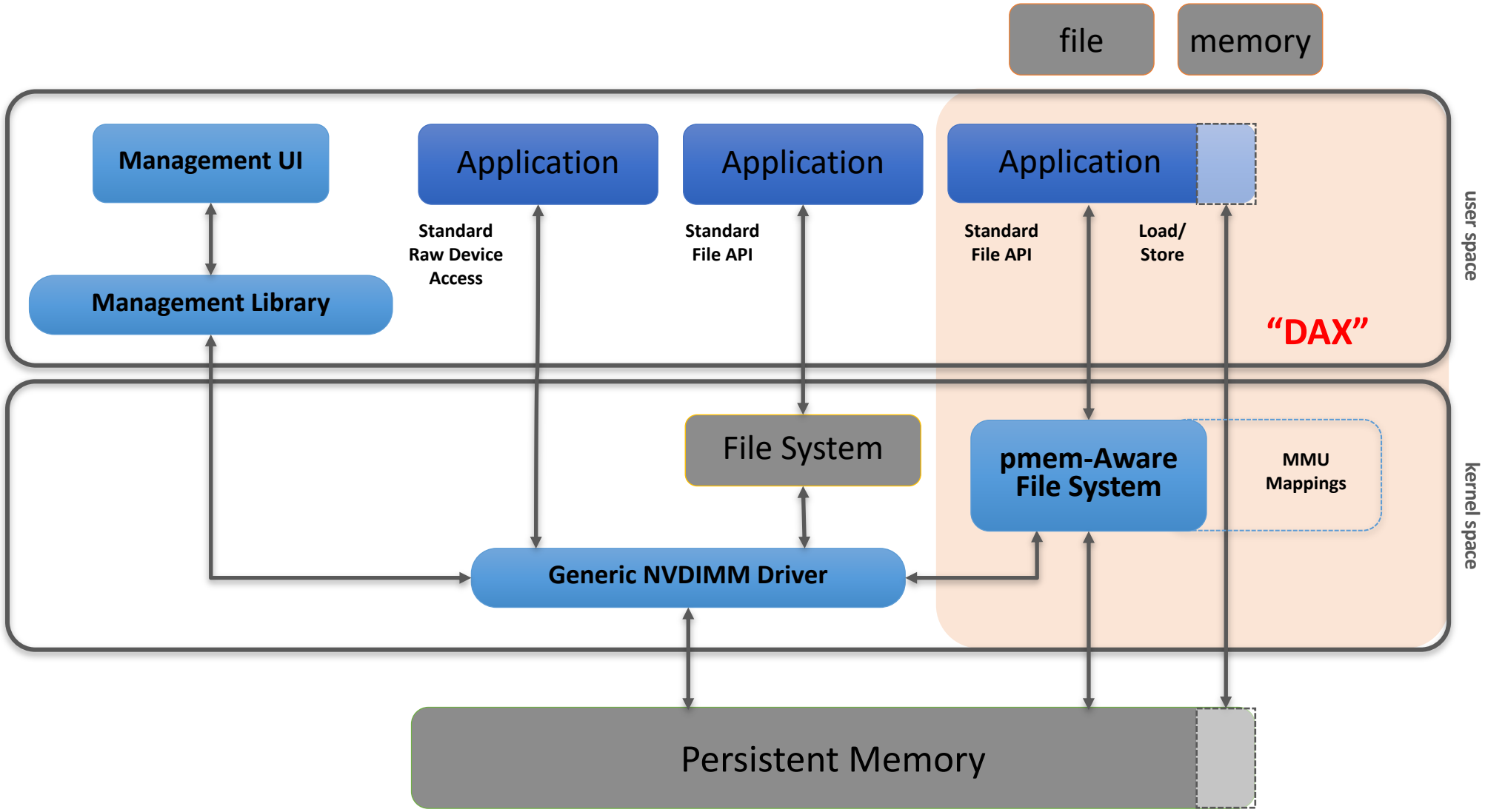
Webhackathon Basics

- 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!

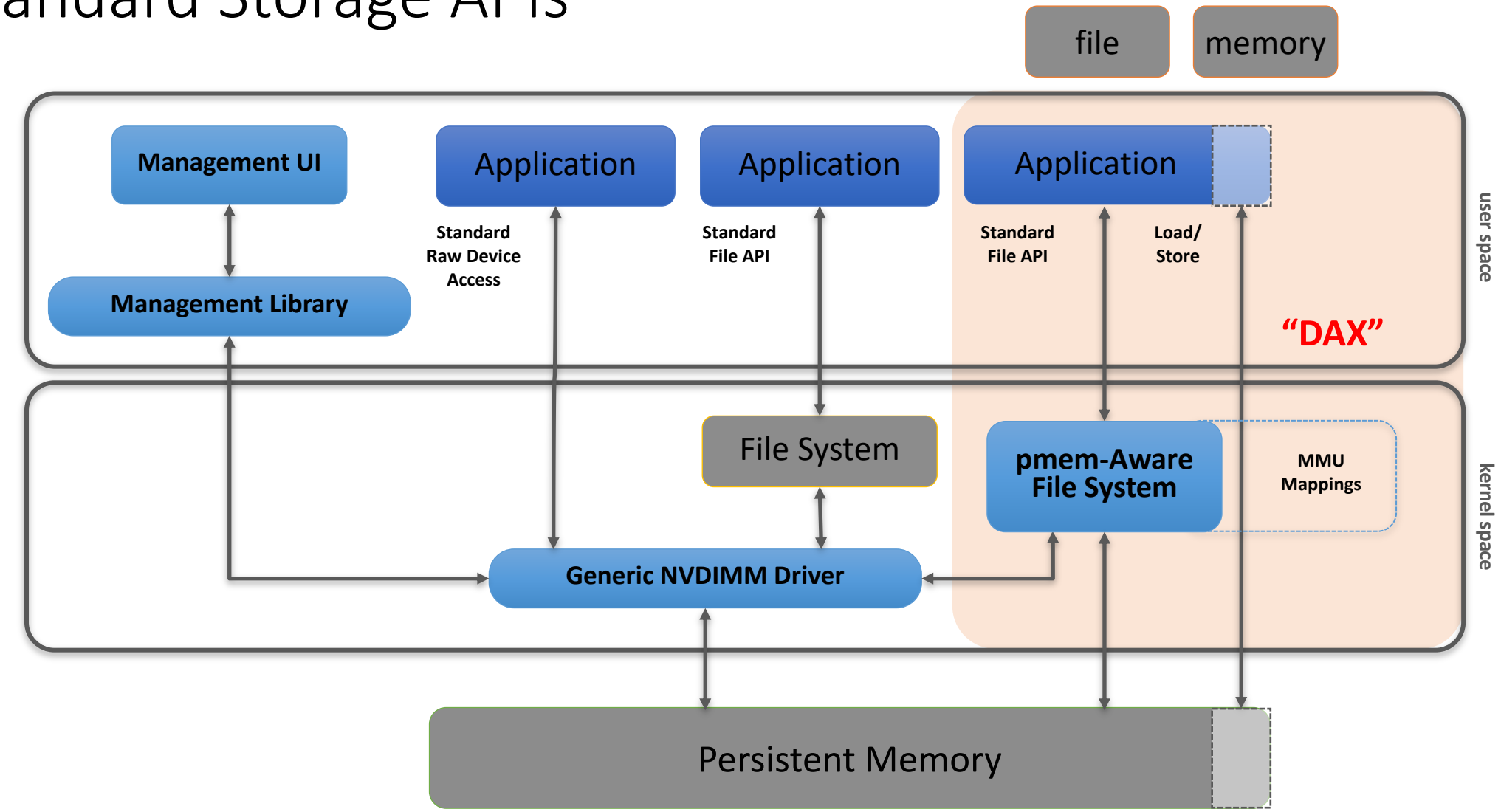
Essential pmem Programming Background

- 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

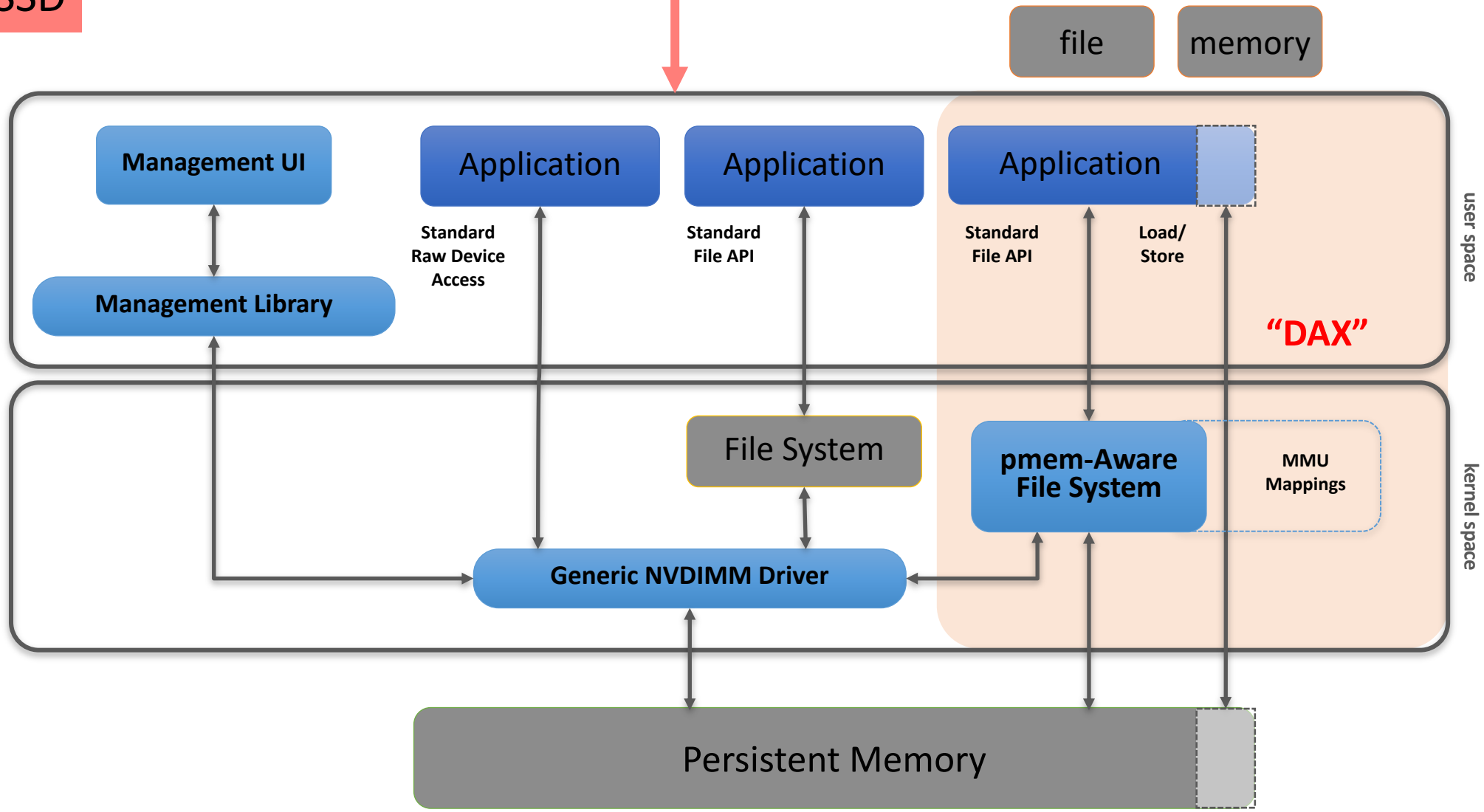
The SNIA NVM Programming Model

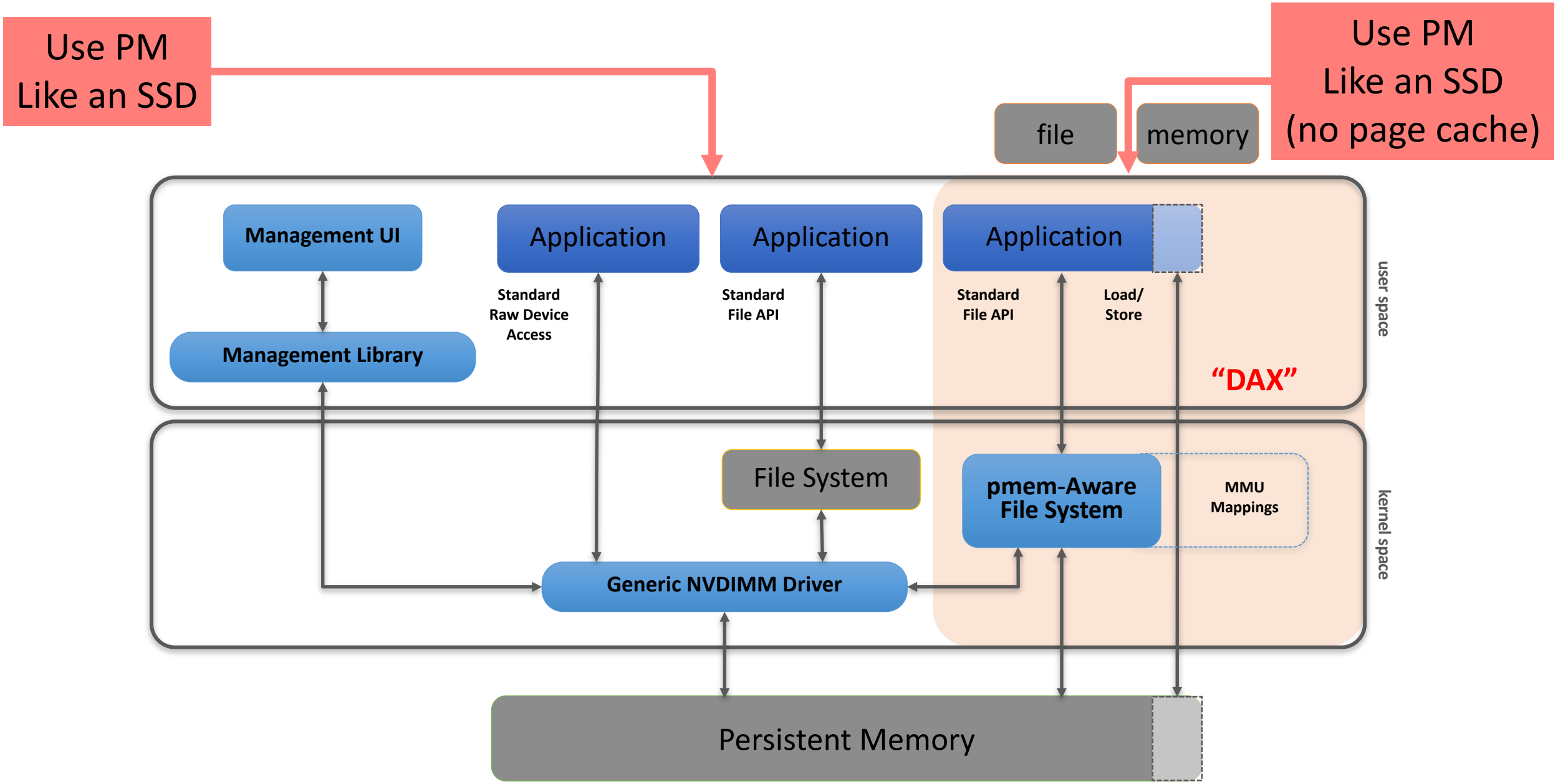


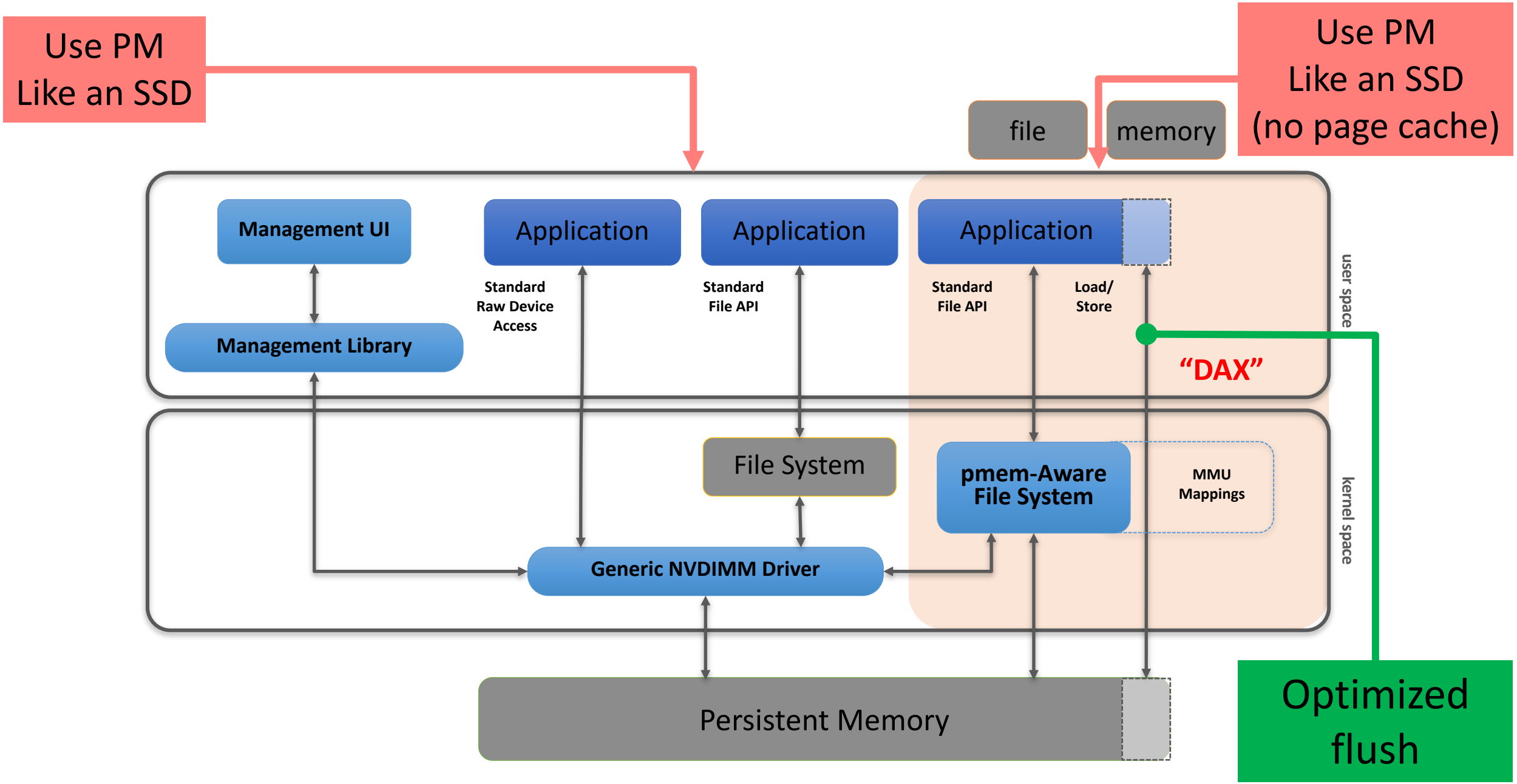
Don’t Forget: The NVM Programming Model Starts With Standard Storage APIs



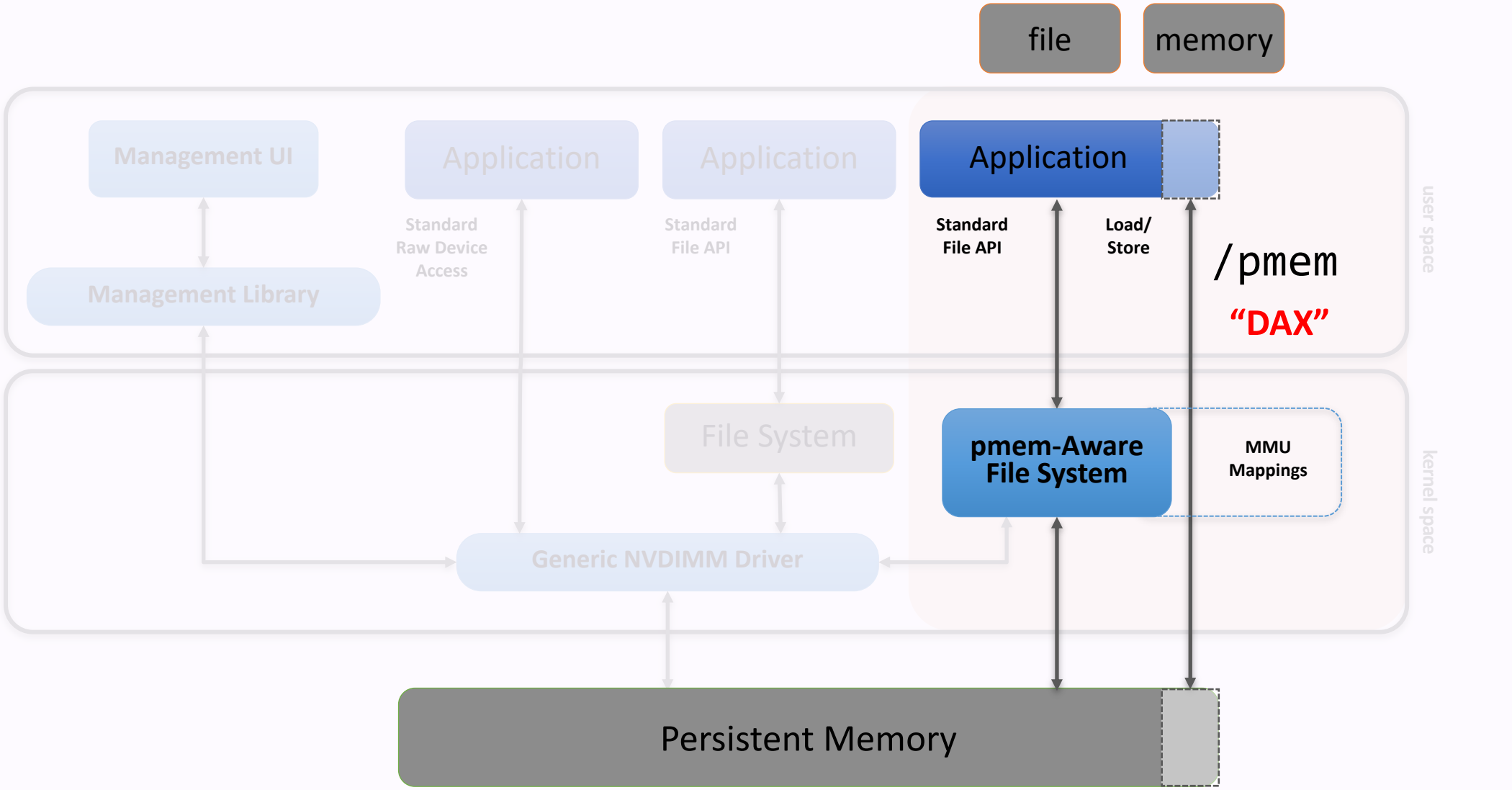
Use PM
Like an SSD





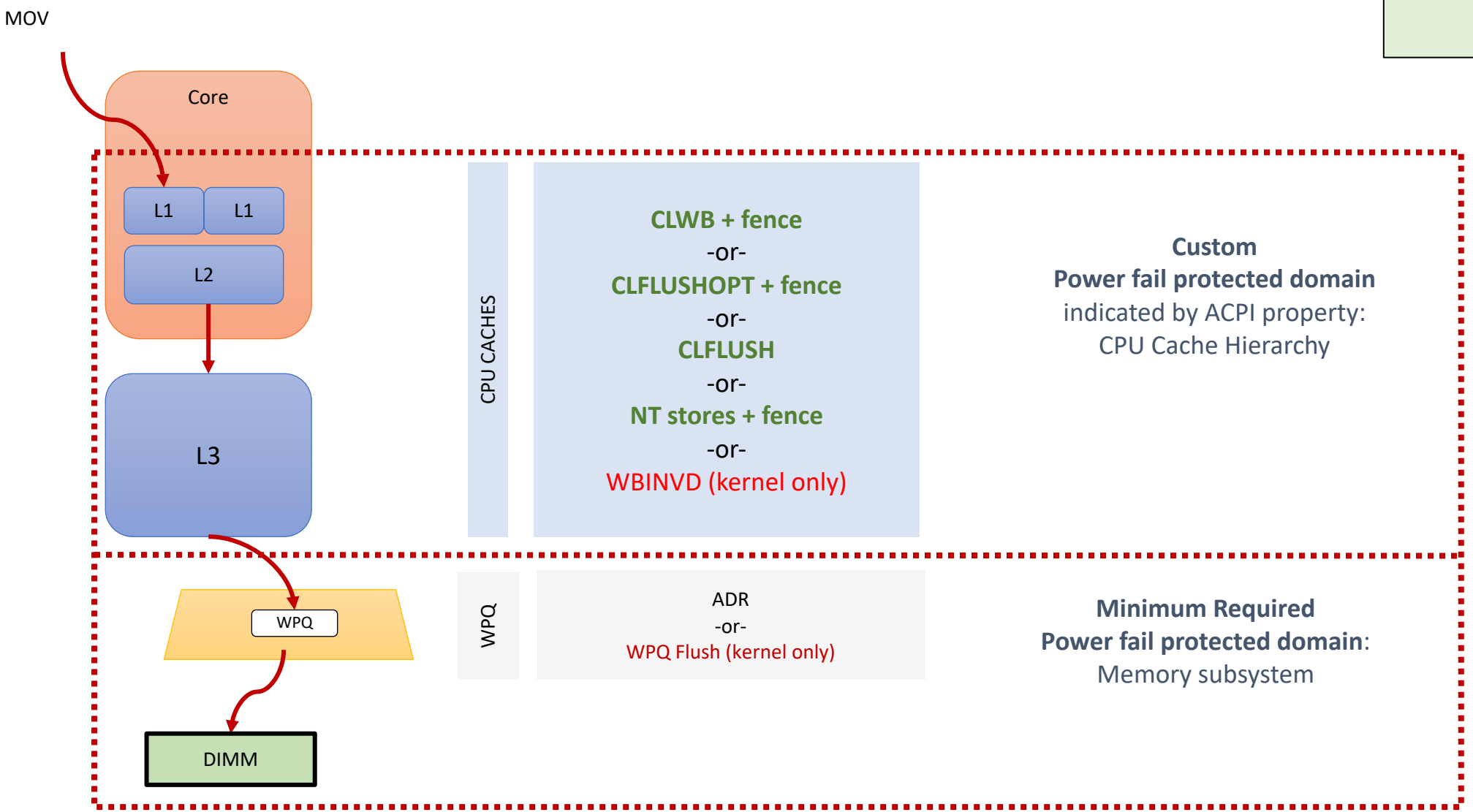


Today's focus

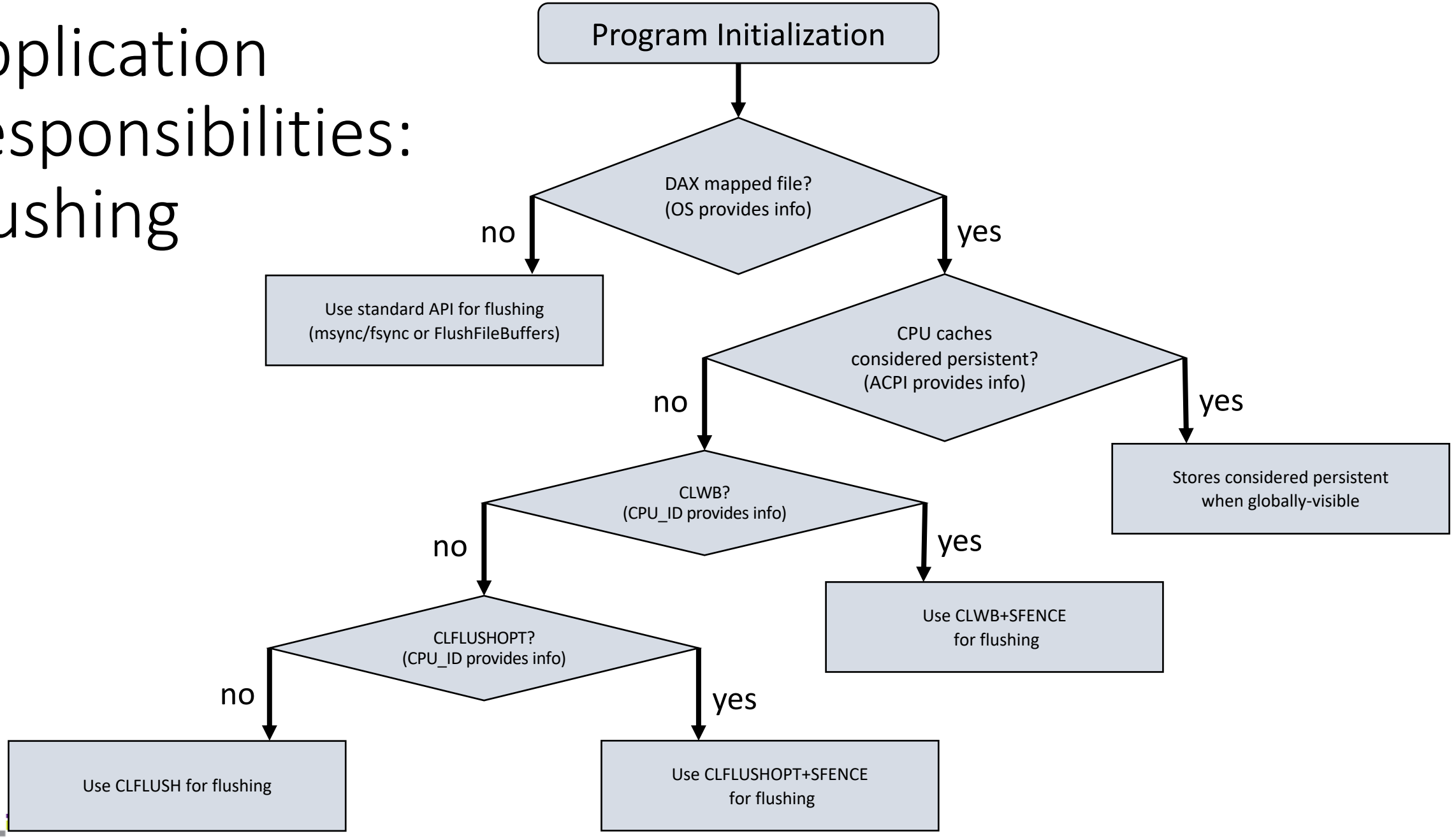


How the Hardware Works

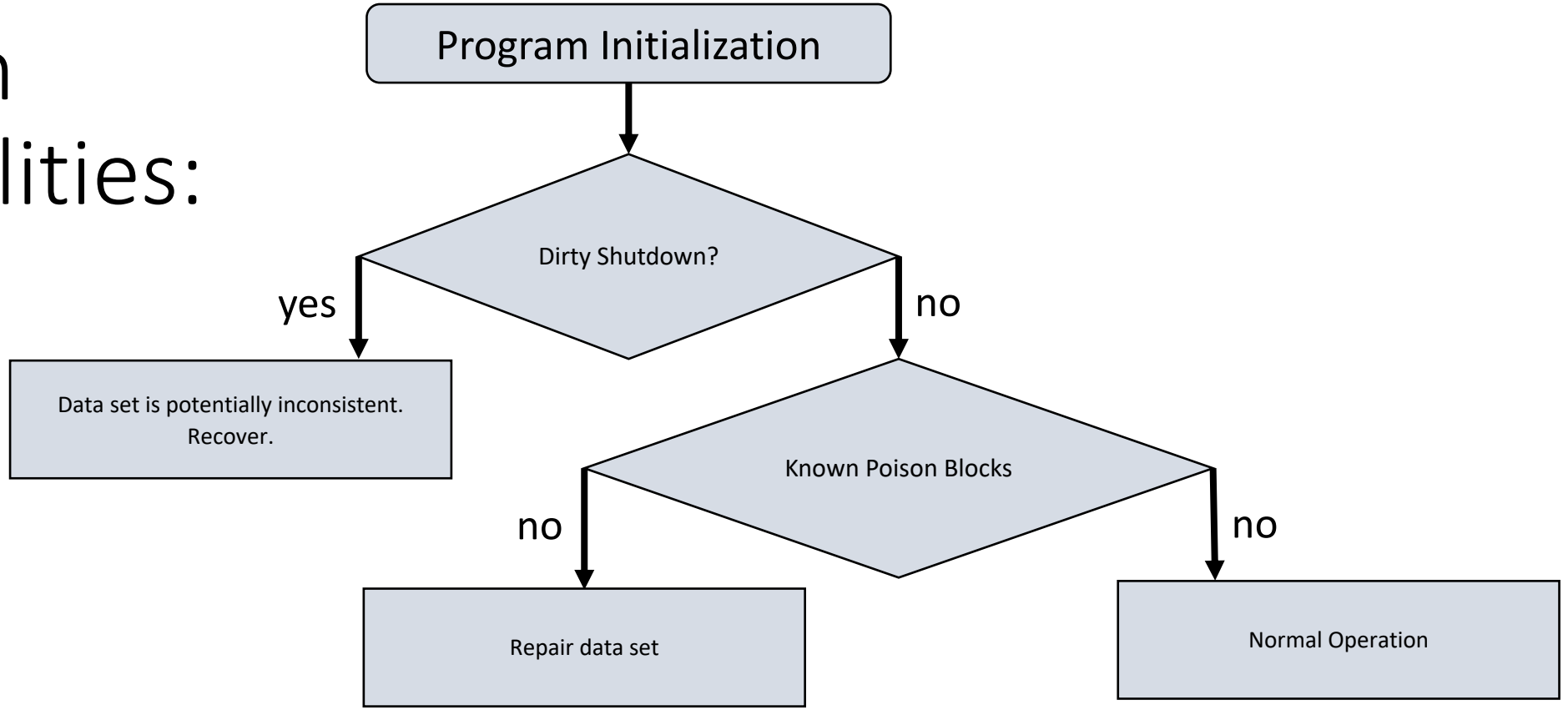
Not shown:
MCA
ADR Failure Detection



Application Responsibilities: Flushing



Application Responsibilities: Recovery



Application Responsibilities: Consistency

```
open(…);

mmap(…);

strcpy(pmem, "Hello, World!");

msync(…);
```

Crash

Result

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

Application Responsibilities: Consistency

```
open(…);

mmap(…);

strcpy(pmem, "Hello, World!");

pmem_persist(pmem, 14);
```

Crash

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

Result

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

PCJ – Persistent Collection for Java

C++ | C | PCJ/ LLPL | Python

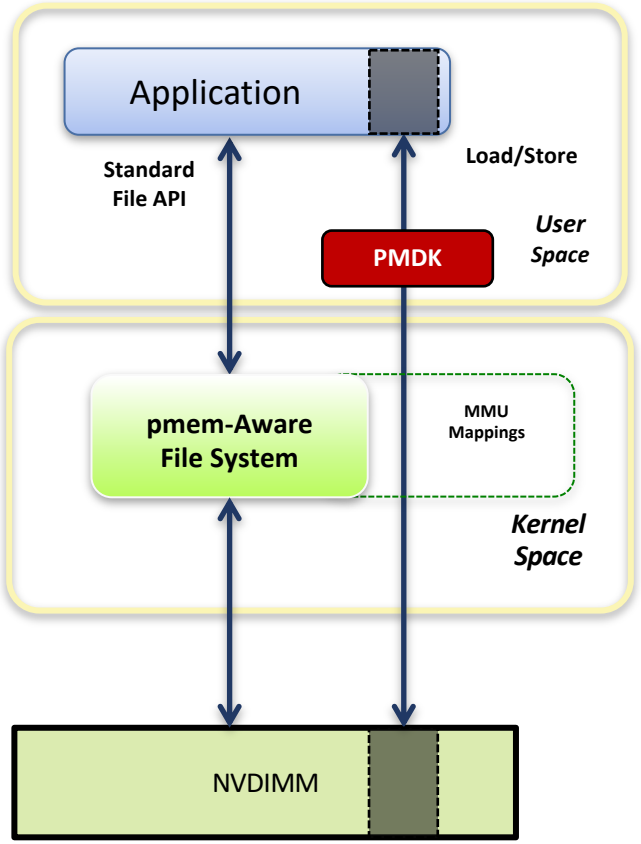
pmemkv

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

Hack, hack, hack…

`http://devhost.pmemhackathon.io`

Username: `pmemuserX` (handed out)

Password: (handed out)

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

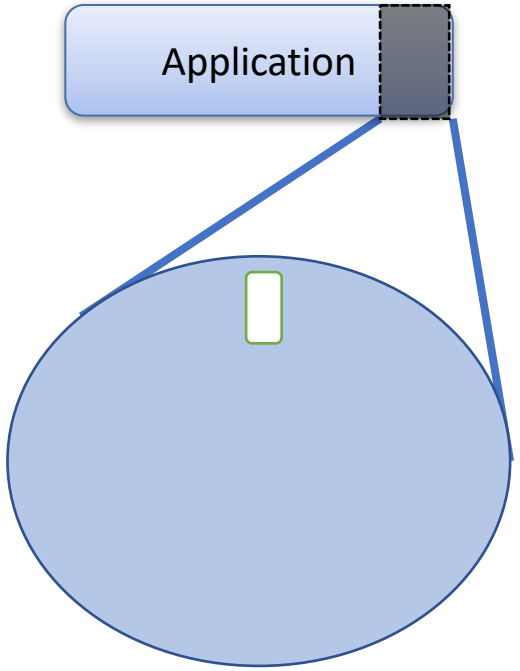
More Developer Resources

- 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

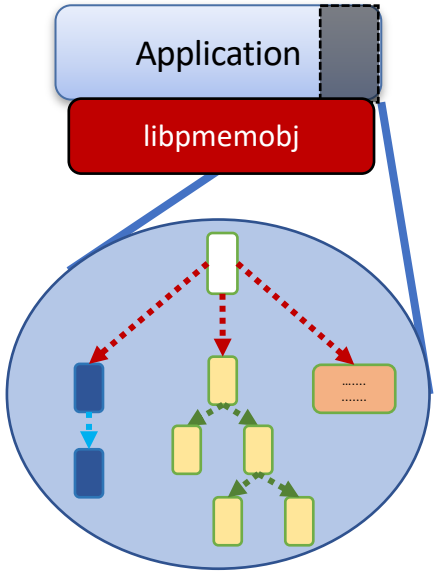


pmem pool “myfile”

root object:

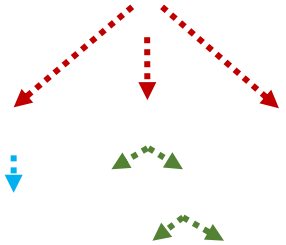
- assume it is always there
- created first time accessed
- initially zeroed

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*

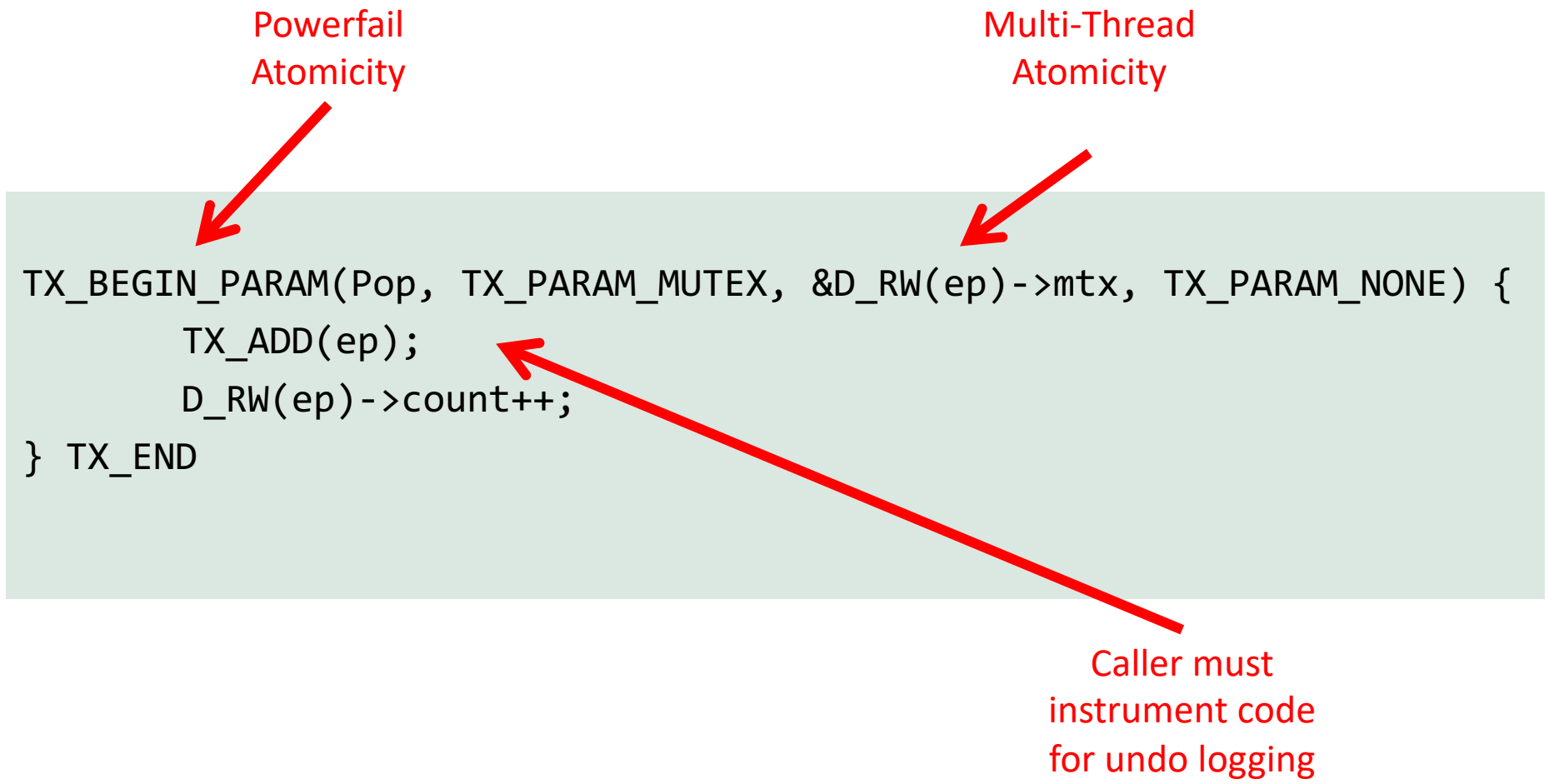


C Programming with libpmemobj

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



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

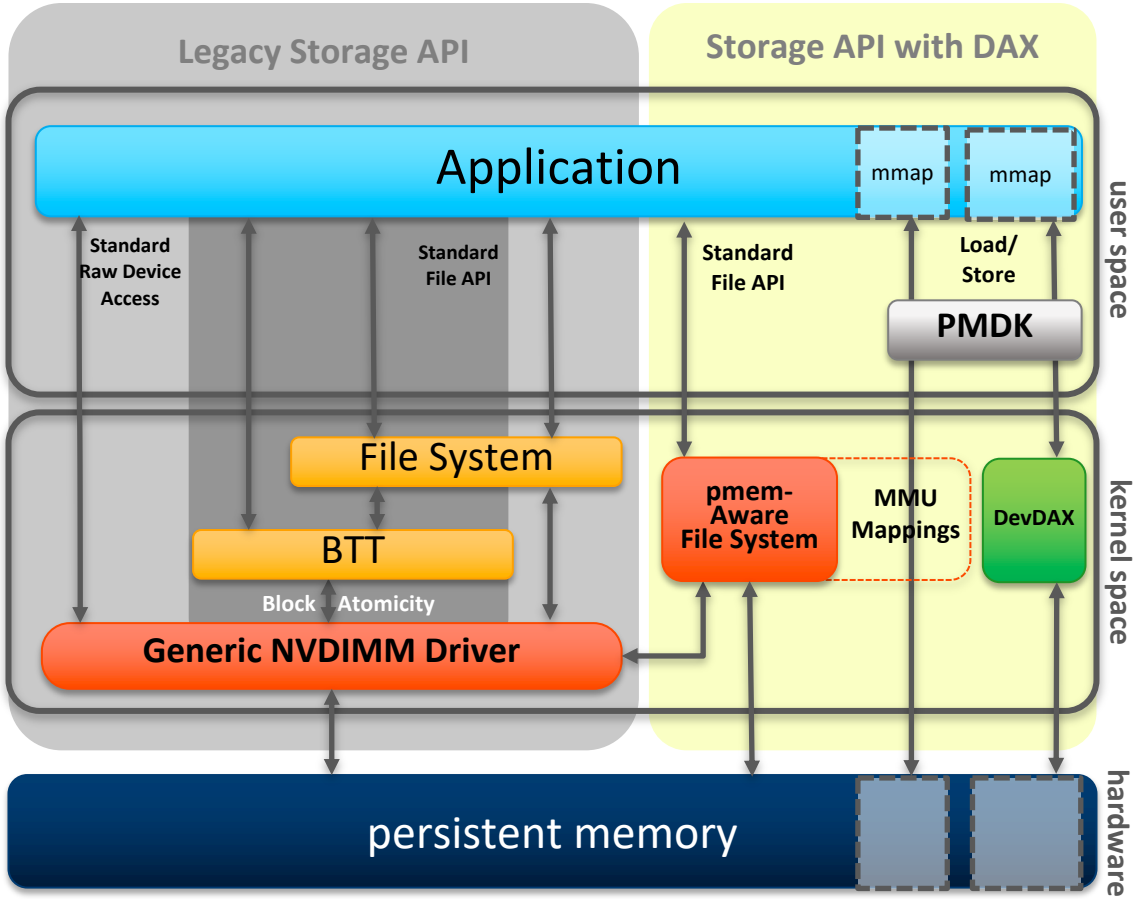
- **New features to support Intel® Optane™ DC persistent memory**
  - Intel® VTune™ Amplifier – Performance Analysis
  - Intel® Inspector – Persistence Inspector finds missing cache flushes & more
  - Free downloads available

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…