SNIA CMSI | COMPUTE, MEMORY, AND STORAGE

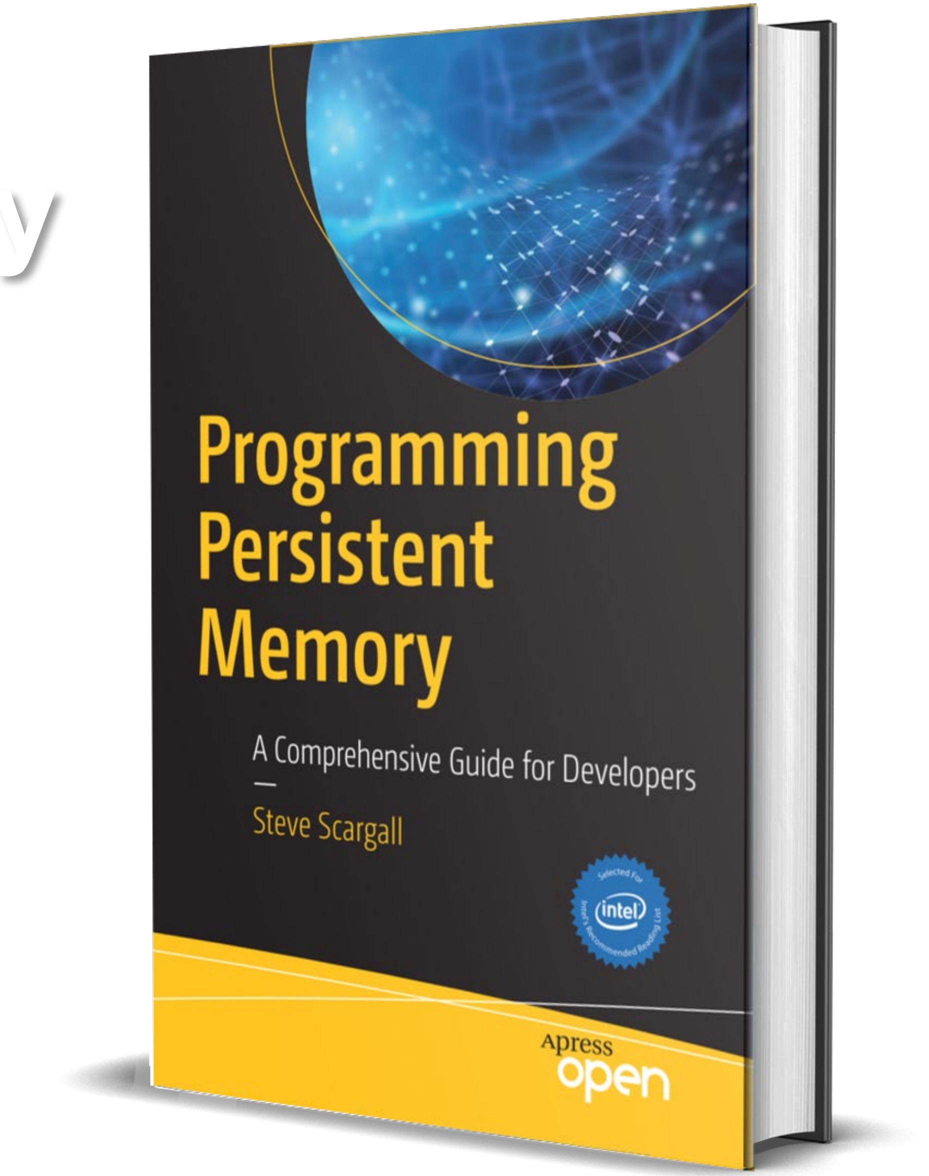
Persistent Memory and CXL.mem Programming Workshop

https://github.com/pmemhackathon/hackathon

Presented by Igor Chorazewicz

Master Persistent Memory Programming

Start Reading

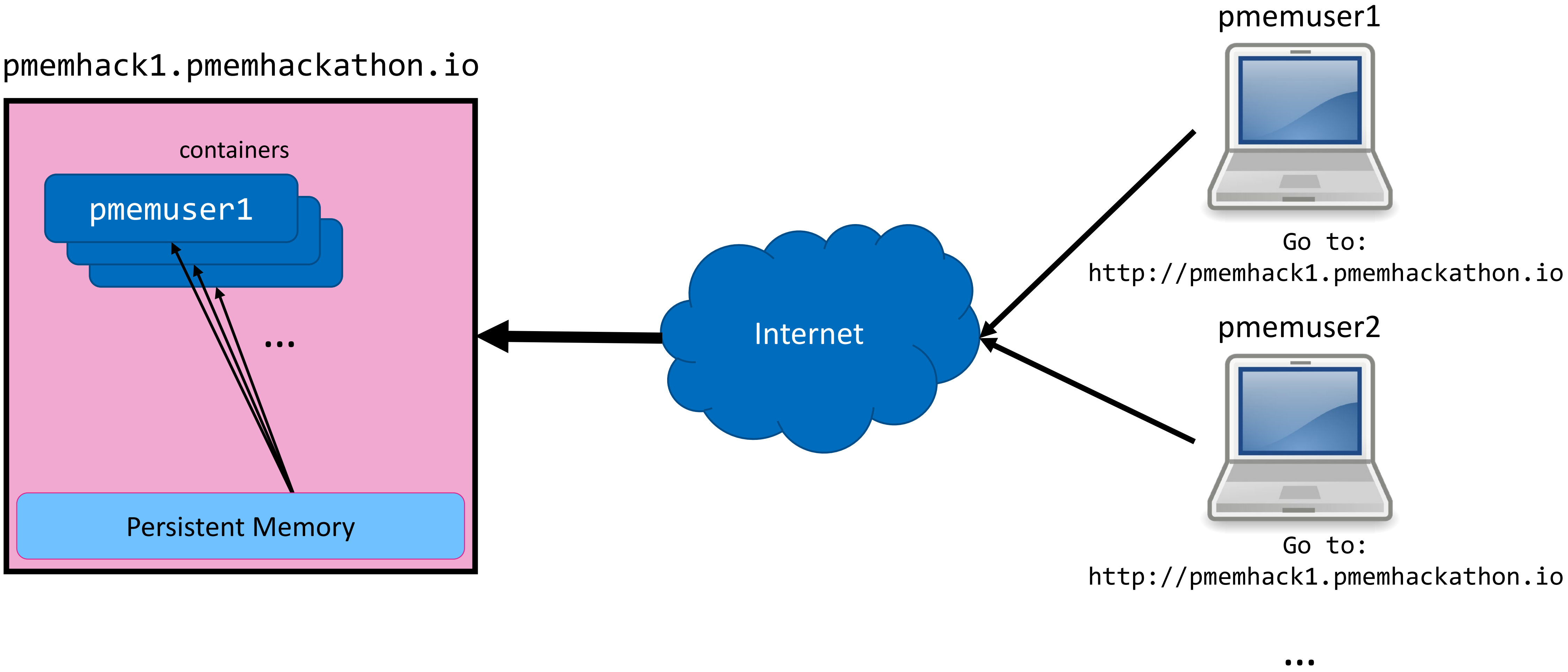

Programming Persistent Memory
A Comprehensive Guide for Developers
Steve Scargall
Selected For intel
Apress open

https://www.apress.com/us/book/9781484249314

Agenda

- Essential Background Slides, covering:
  - Logistics: how you access persistent memory from your laptop
  - The minimum you need to know about persistent memory
  - Persistent memory to CXL.mem transition
- Goal is to get you hands-on with pmem programming quickly and show how Pmem-optimized application can run on CXL.
  - All slides and examples are in the repo
  - Lots more detail in additional slide decks in the repo

Logistics: The *webhackathon* Tool


pmemhack1.pmemhackathon.io
containers
pmemuser1
...
Persistent Memory
Internet
pmemuser1
Go to:
http://pmemhack1.pmemhackathon.io
pmemuser2
Go to:
http://pmemhack1.pmemhackathon.io
...

Webhackathon Basics

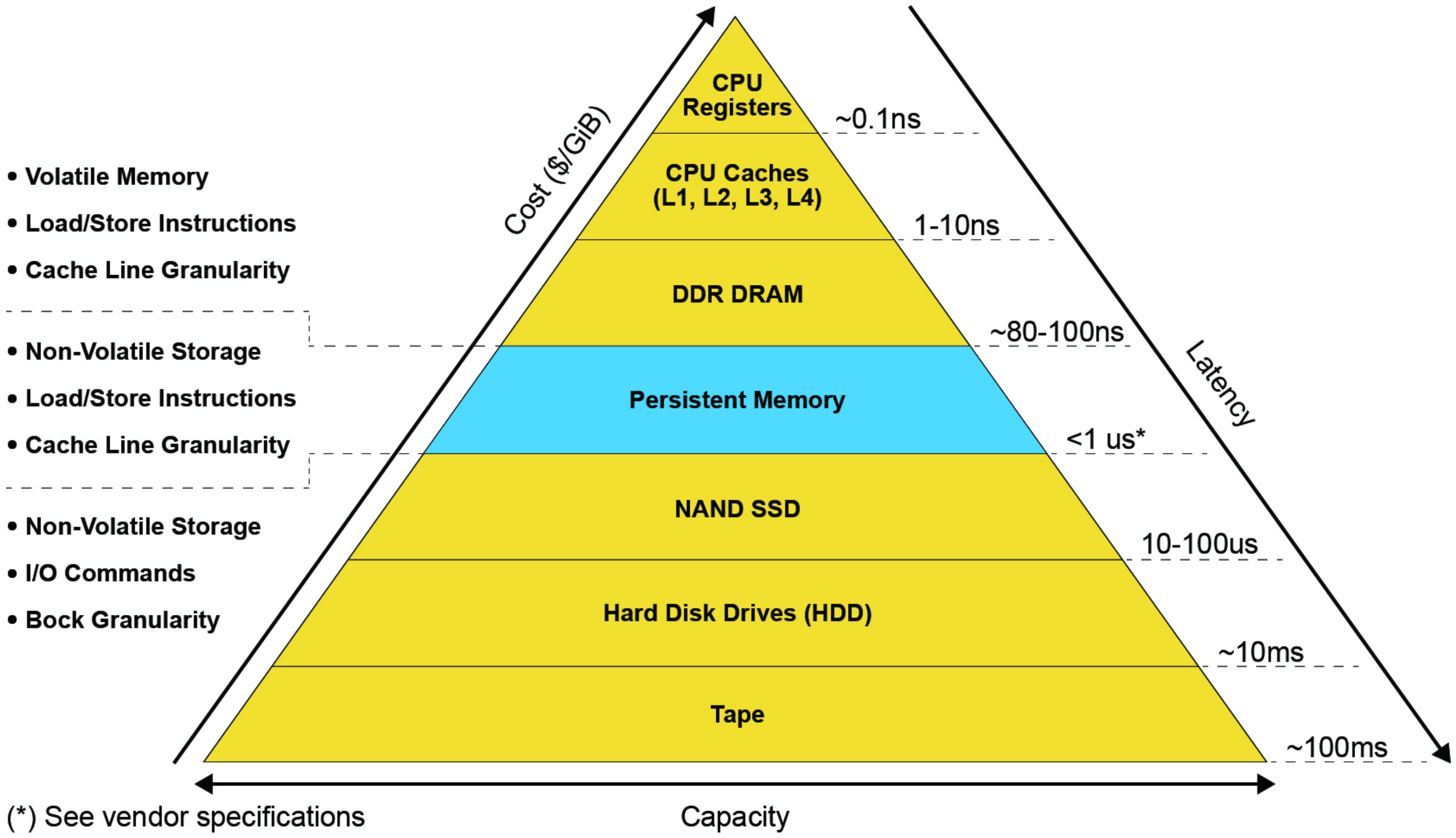
- List of examples presented on main page
  - First three recommended to provide essential background
    - We will walk through some of these together
  - Pick examples that are interesting to you (task, language, etc)
  - Use them as a starting point for your own code
- Menu provides:
  - Access to 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

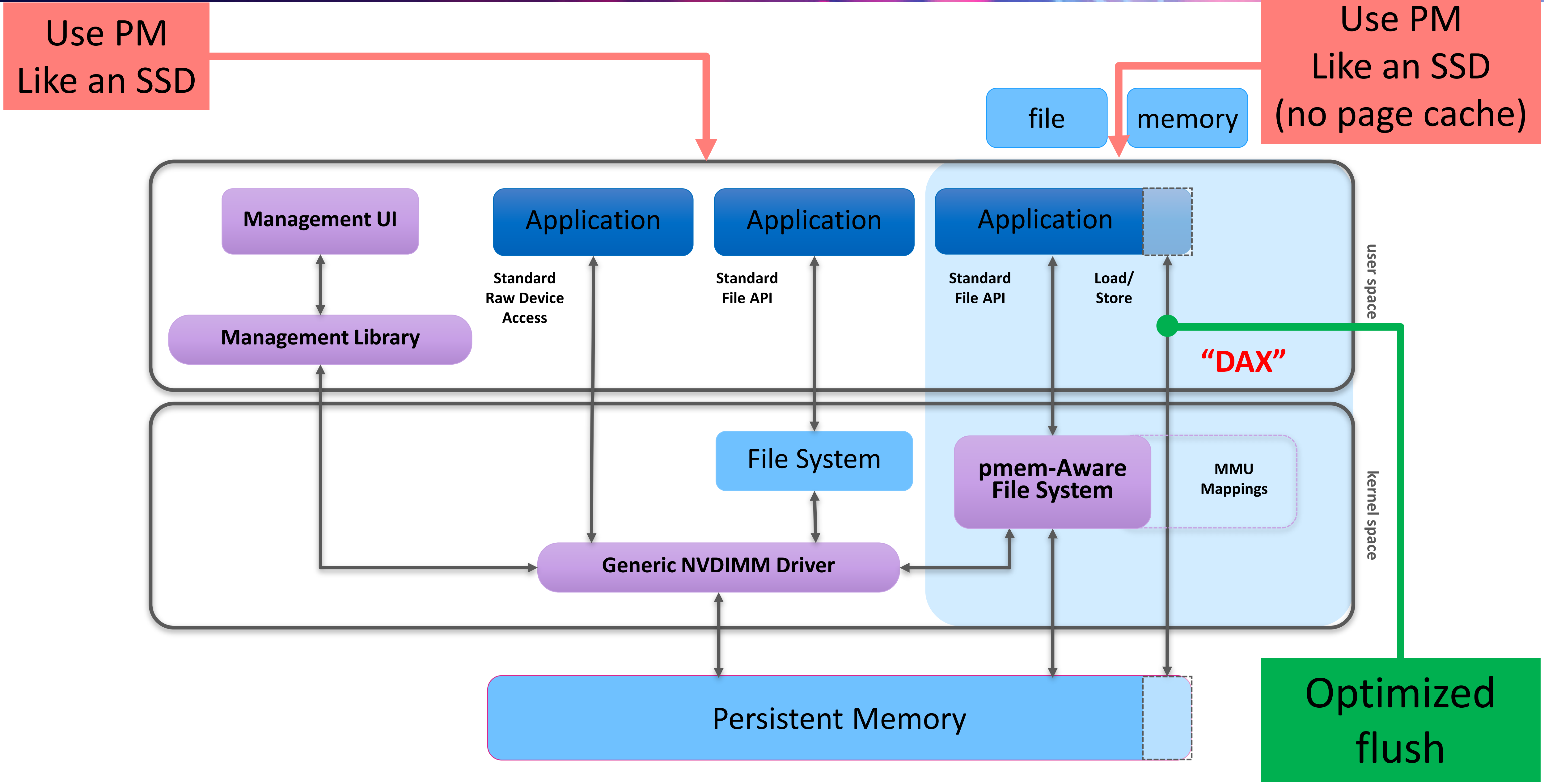
- Lots of ways to use pmem with existing programs
  - Storage APIs
  - Libraries or kernels using pmem transparently
  - Memory Mode
- This workshop doesn't cover the above (too easy!)
  - We assume you want direct access to pmem
  - We show code, but also concepts
  - There are lots of paths you can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

Persistent memory

Performance considerations



(*) See vendor specifications



Use PM Like an SSD

Use PM Like an SSD (no page cache)

file

memory

Management UI

Management Library

Application

Standard Raw Device Access

Application

Standard File API

Application

Standard File API

Load/ Store

"DAX"

user space

File System

pmem-Aware File System

MMU Mappings

Generic NVDIMM Driver

kernel space

Persistent Memory

Optimized flush

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

LLPL

Python

pmemkv

C | C++ | Java | JS | Ruby

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

vmemcache

Low level support for local persistent memory

libpmem

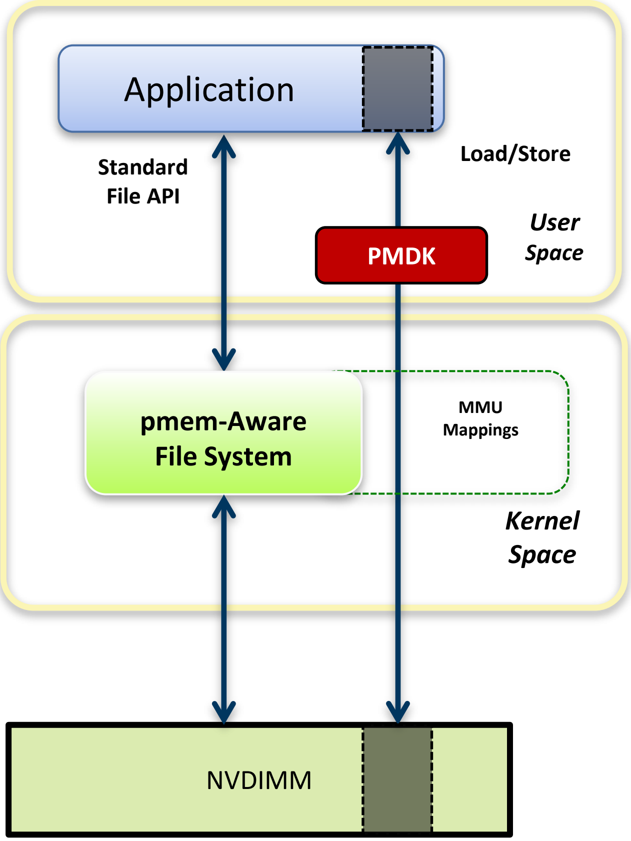
libpmem2

Low level support for remote persistent memory

libpmem

librpma

Low-level support

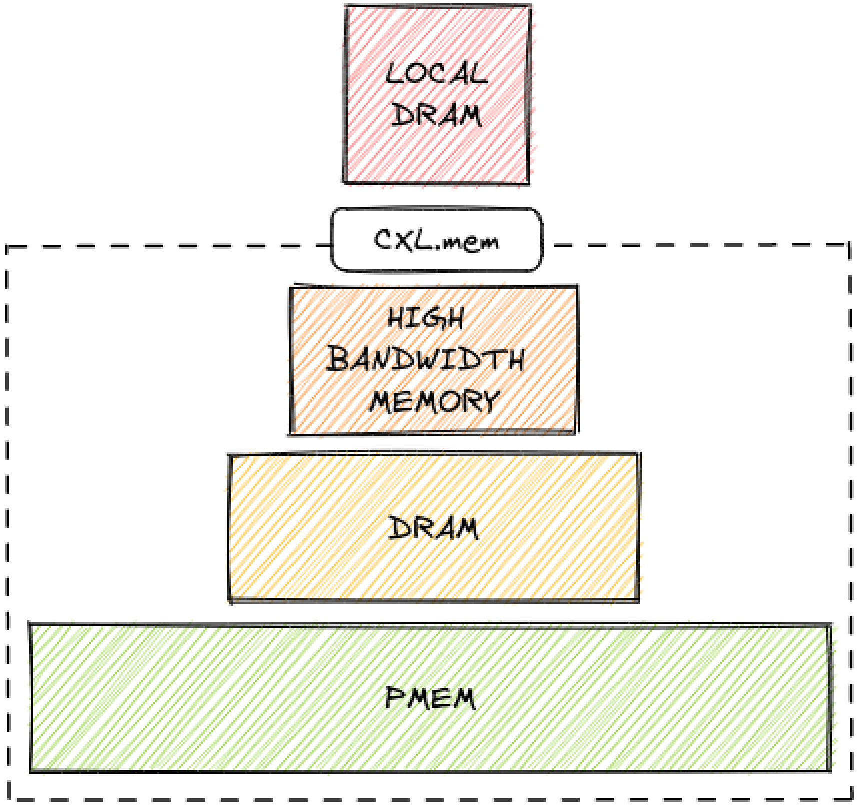


CXL.mem

CXL

- Interconnect standard built on top of PCIe
- Facilitates cache-coherent memory access between CPUs and supporting PCIe-attached devices (pure memory devices but also accelerators) – CXL.cache and CXL.mem
- Supports memory pooling and sharing
- Memory connected through CXL can be exposed similarly as Pmem

Heterogenous memory hierarchy

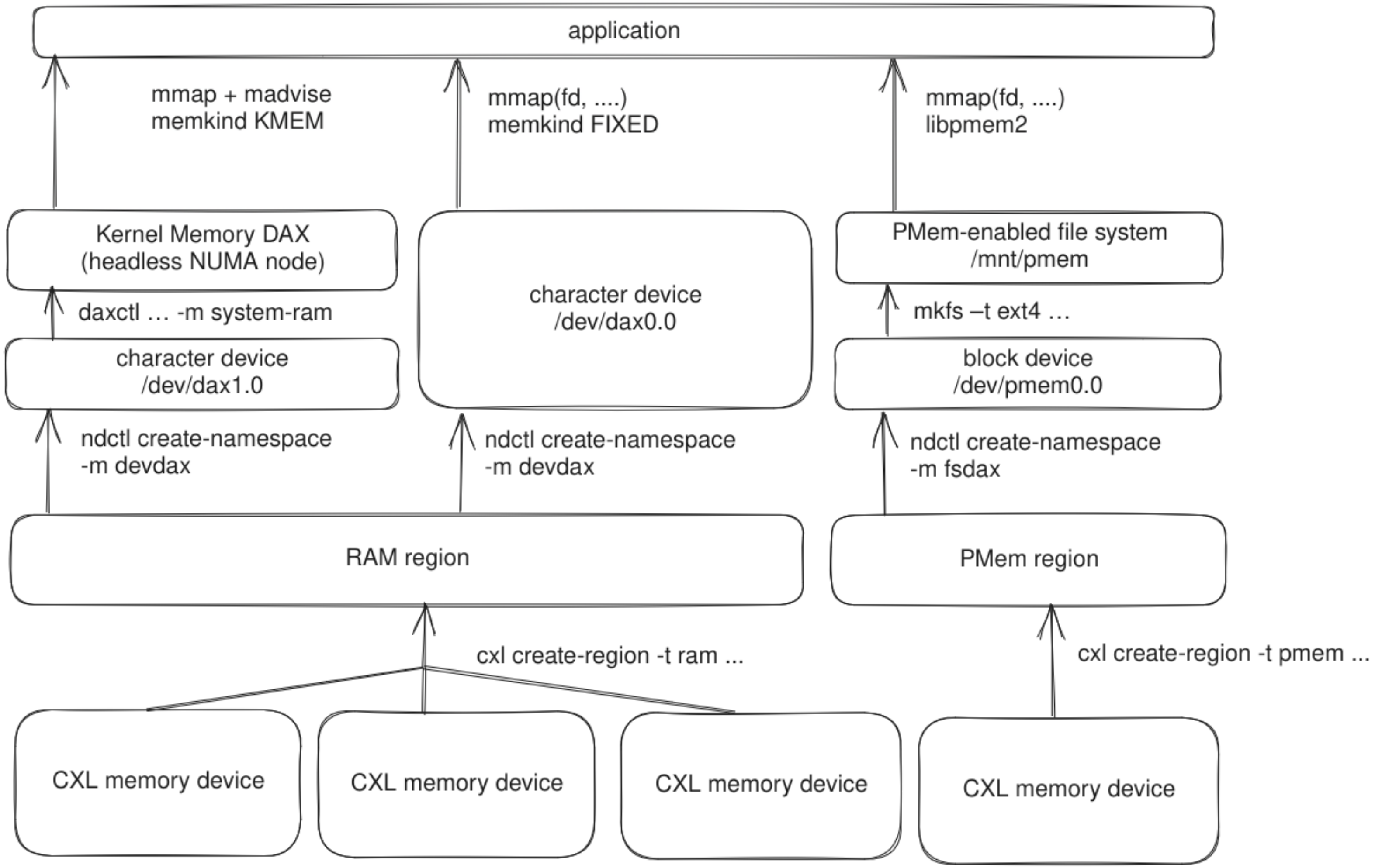


LOWEST LATENCY
HIGHEST COST

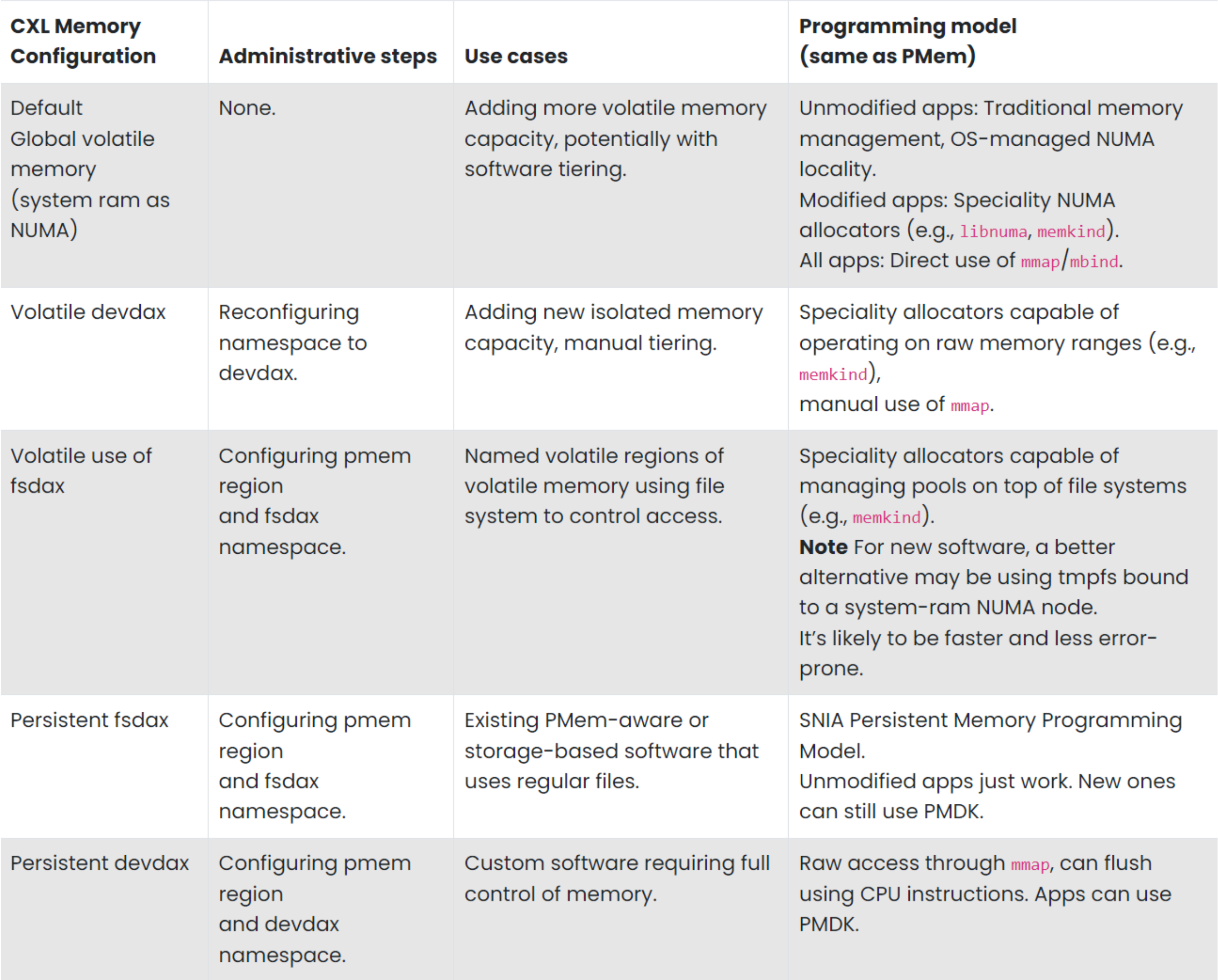
HIGHEST LATENCY
LOWEST COST

System topology with CXL

- CXL Type 3 (memory) devices can provide both volatile or persistent capacity
- Transitioning from PMem to CXL is straightforward for most use-cases

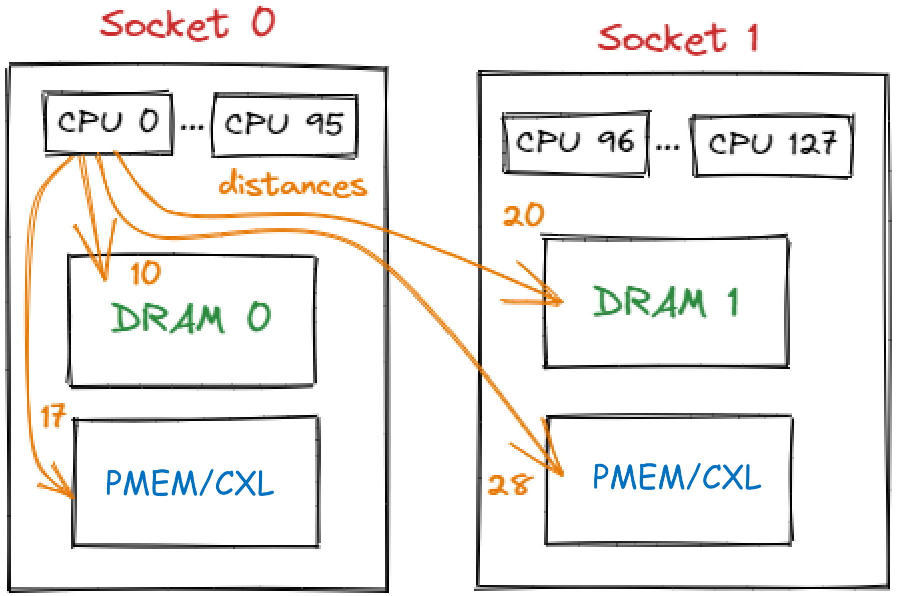


CXL Software ecosystem



CXL Memory Configuration	Administrative steps	Use cases	Programming model (same as PMem)
Default Global volatile memory (system ram as NUMA)	None.	Adding more volatile memory capacity, potentially with software tiering.	Unmodified apps: Traditional memory management, OS-managed NUMA locality. Modified apps: Speciality NUMA allocators (e.g., `libnuma`, `memkind`). All apps: Direct use of `mmap`/`mbind`.
Volatile devdax	Reconfiguring namespace to devdax.	Adding new isolated memory capacity, manual tiering.	Speciality allocators capable of operating on raw memory ranges (e.g., `memkind`), manual use of `mmap`.
Volatile use of fsdax	Configuring pmem region and fsdax namespace.	Named volatile regions of volatile memory using file system to control access.	Speciality allocators capable of managing pools on top of file systems (e.g., `memkind`). **Note** For new software, a better alternative may be using tmpfs bound to a system-ram NUMA node. It's likely to be faster and less error-prone.
Persistent fsdax	Configuring pmem region and fsdax namespace.	Existing PMem-aware or storage-based software that uses regular files.	SNIA Persistent Memory Programming Model. Unmodified apps just work. New ones can still use PMDK.
Persistent devdax	Configuring pmem region and devdax namespace.	Custom software requiring full control of memory.	Raw access through `mmap`, can flush using CPU instructions. Apps can use PMDK.

NUMA nodes


Socket 0
Socket 1
CPU 0
...
CPU 95
CPU 96
...
CPU 127
distances
10
DRAM 0
17
PMEM/CXL
20
DRAM 1
28
PMEM/CXL

Login to server…

```
http://pmemhack1.pmemhackathon.io
```

Click Request Access to get a login

```
Workshop ID:cmsiintel
```

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

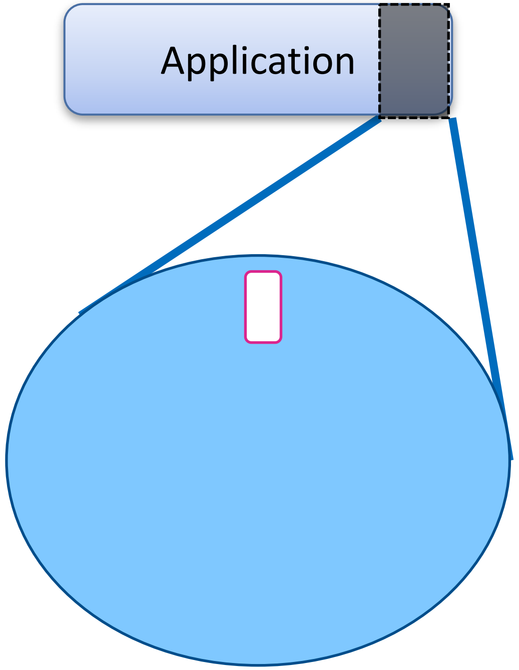
- Find the PMDK (Persistent Memory Development Kit) at http://pmem.io/pmdk/
- Getting Started
  - Intel IDZ persistent memory- https://software.intel.com/en-us/persistent-memory
  - Entry into overall architecture - http://pmem.io/2014/08/27/crawl-walk-run.html
  - Emulate persistent memory - http://pmem.io/2016/02/22/pm-emulation.html
- 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

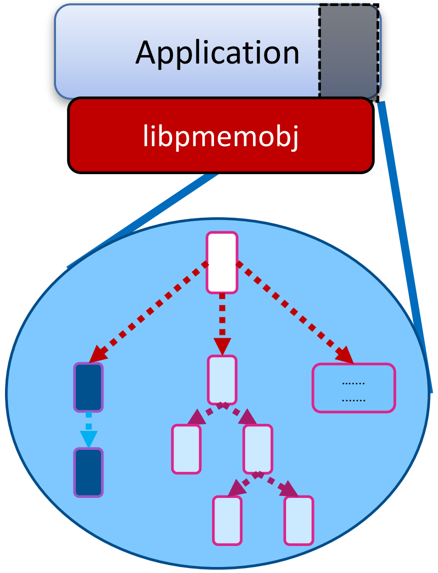


pmem pool “myfile”

root object:

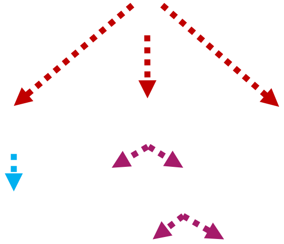
- assume it is always there
- created first time accessed
- initially zeroed

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*

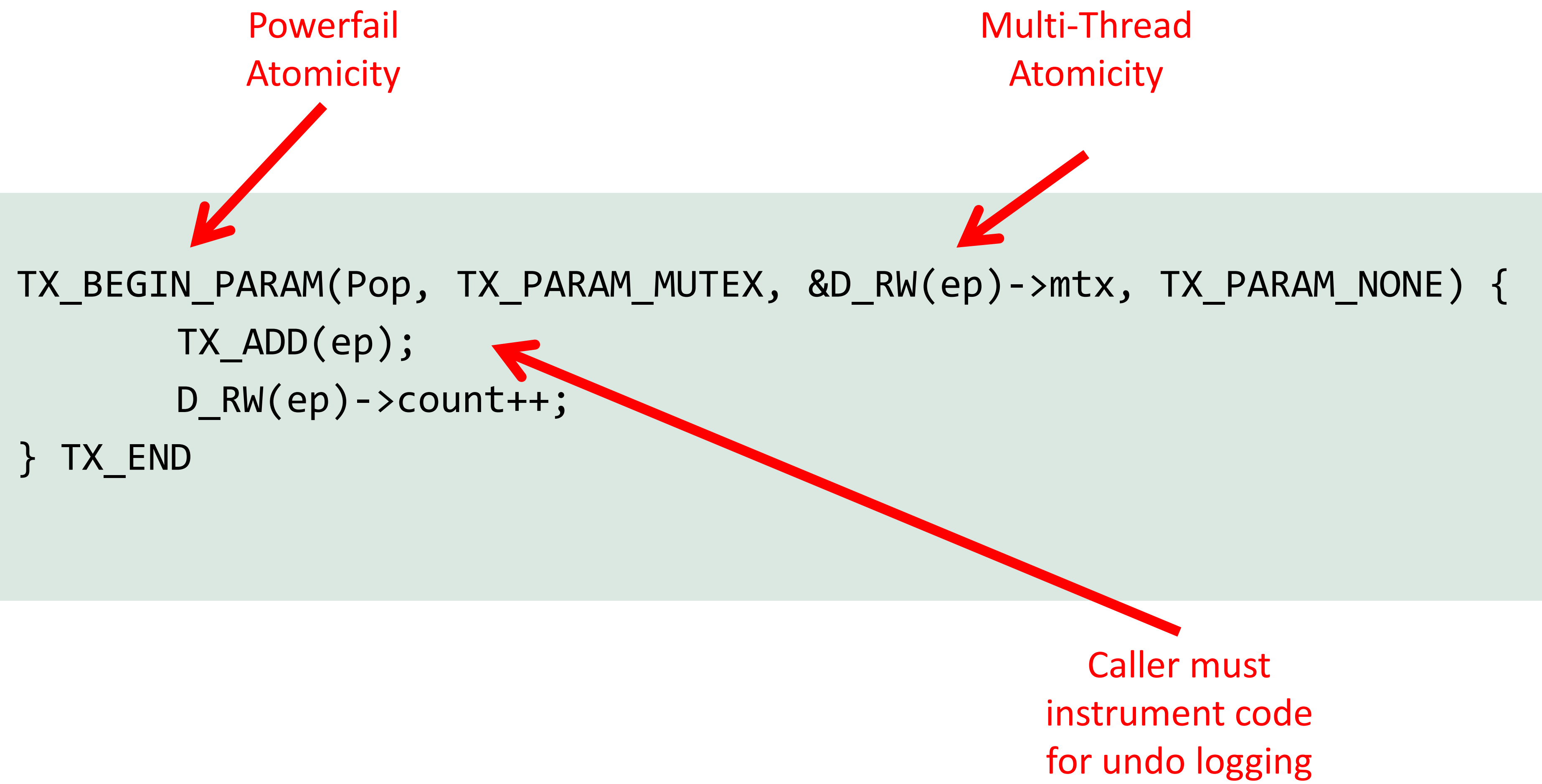


C Programming with libpmemobj

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



Powerfail Atomicity

Multi-Thread Atomicity

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

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

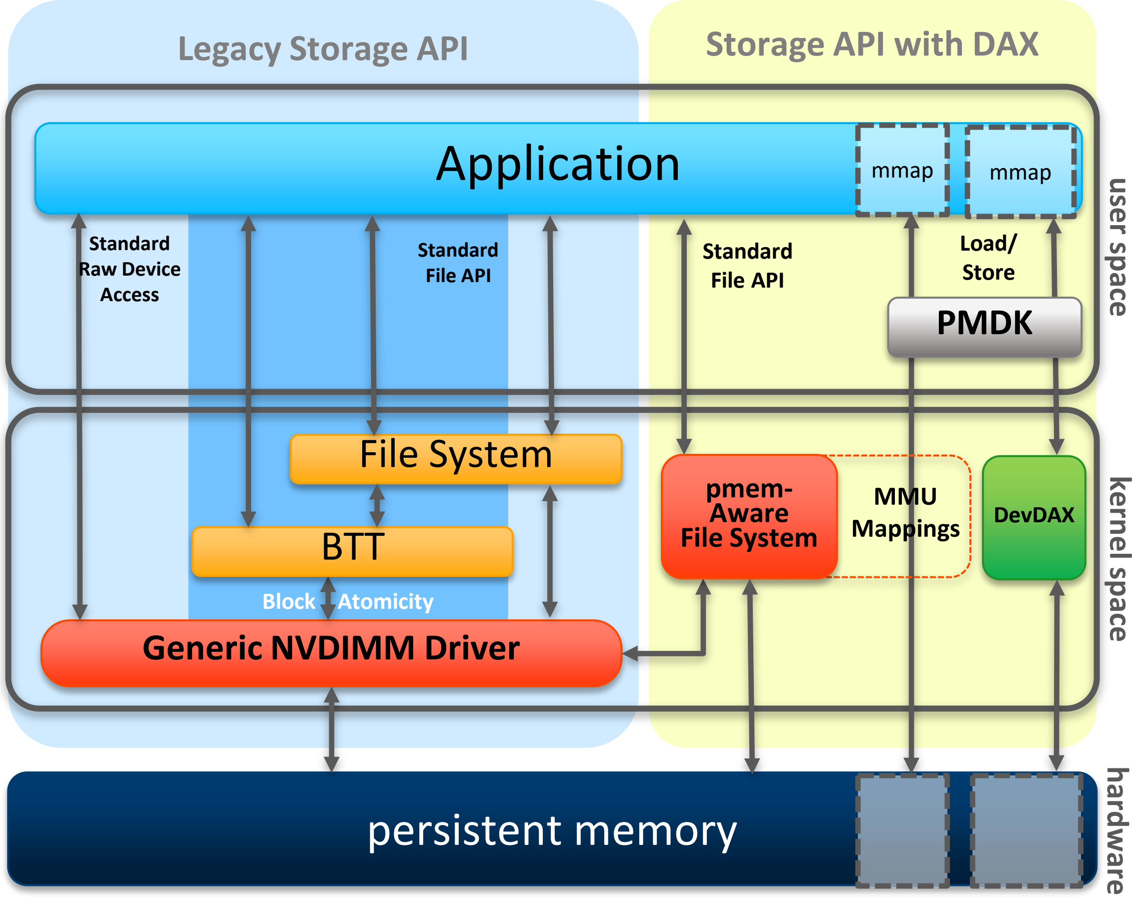
- **New features to support Intel® Optane™ DC persistent memory**
  - Intel® VTune™ Amplifier – Performance Analysis
  - Intel® Inspector – Persistence Inspector finds missing cache flushes & more
  - Free downloads available

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
- Steve Dohrmann
- 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…