SNIA PERSISTENT MEMORY PM SUMMIT

JANUARY 24, 2019 | SANTA CLARA, CA

# Hackathon part 2: Programming

https://github.com/pmemhackathon/2019-01-23

Andy Rudoff
pmem SW Architect, Intel

# Essential Background

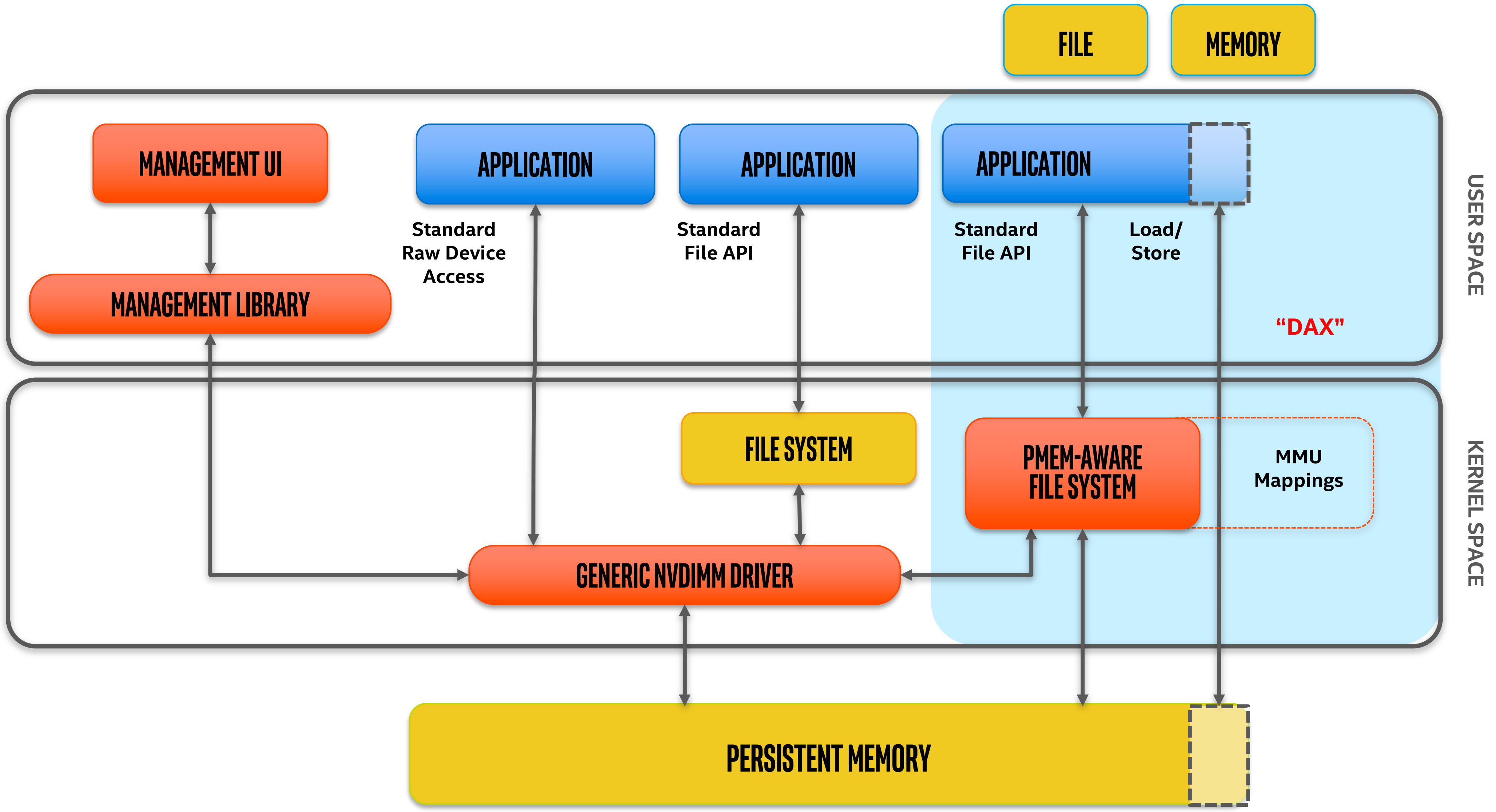
## Lots of ways to use pmem with existing programs

- 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



# Contents 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”

## INSTALLING PACKAGES AND REPOS

## Hackathon repo

https://github.com/pmemhackathon/2019-01-23

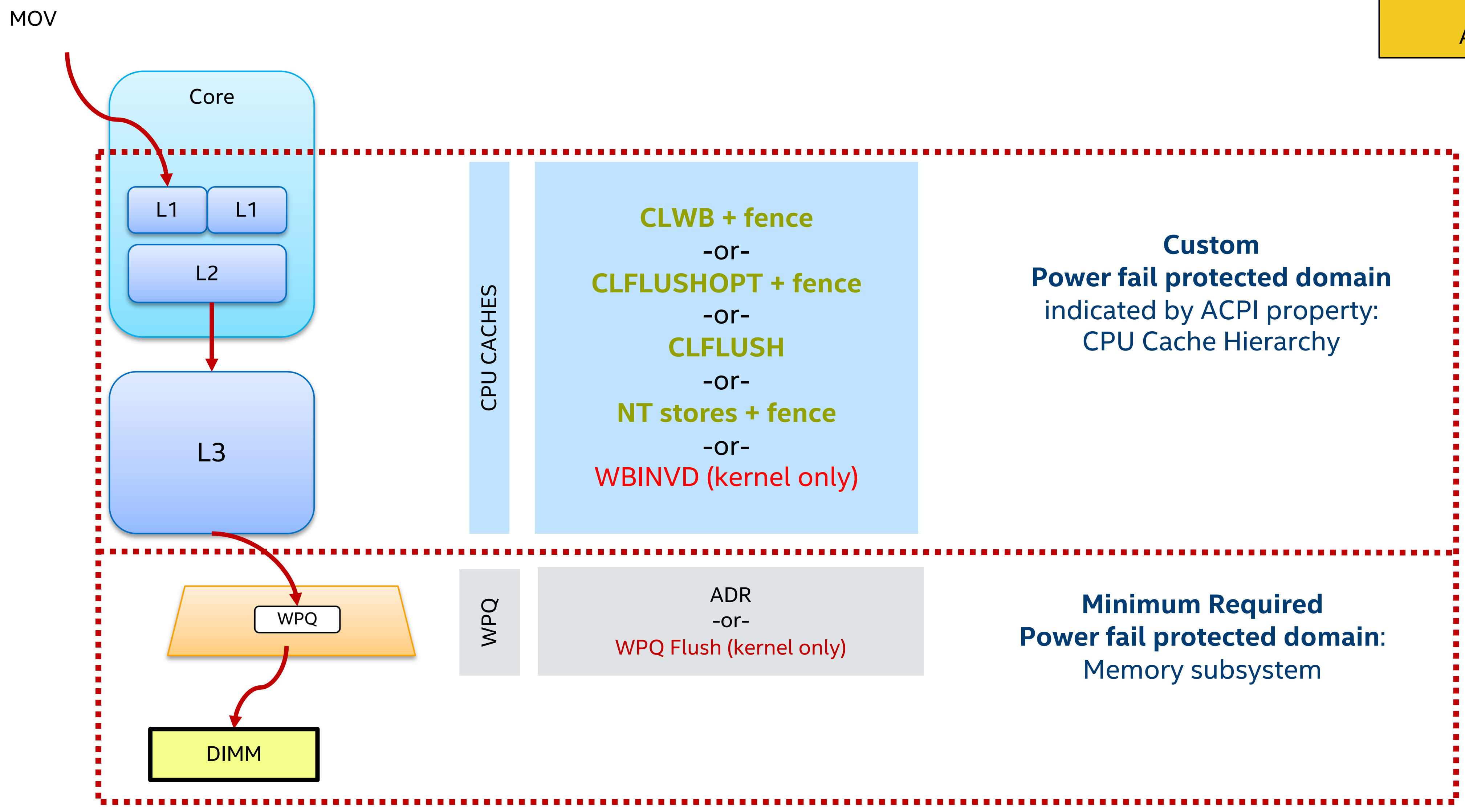
README.txt

- We will walk through the examples using the README.txt
- Switch back to slides now and then for additional details
- Bringing up the README.txt in a window will help avoid typos
  - Cut and paste commands instead of reading and typing them

Repo contains these slides as well (slides.pdf) for future reference.

# How the Hardware Works

Not shown:
MCA
ADR Failure Detection



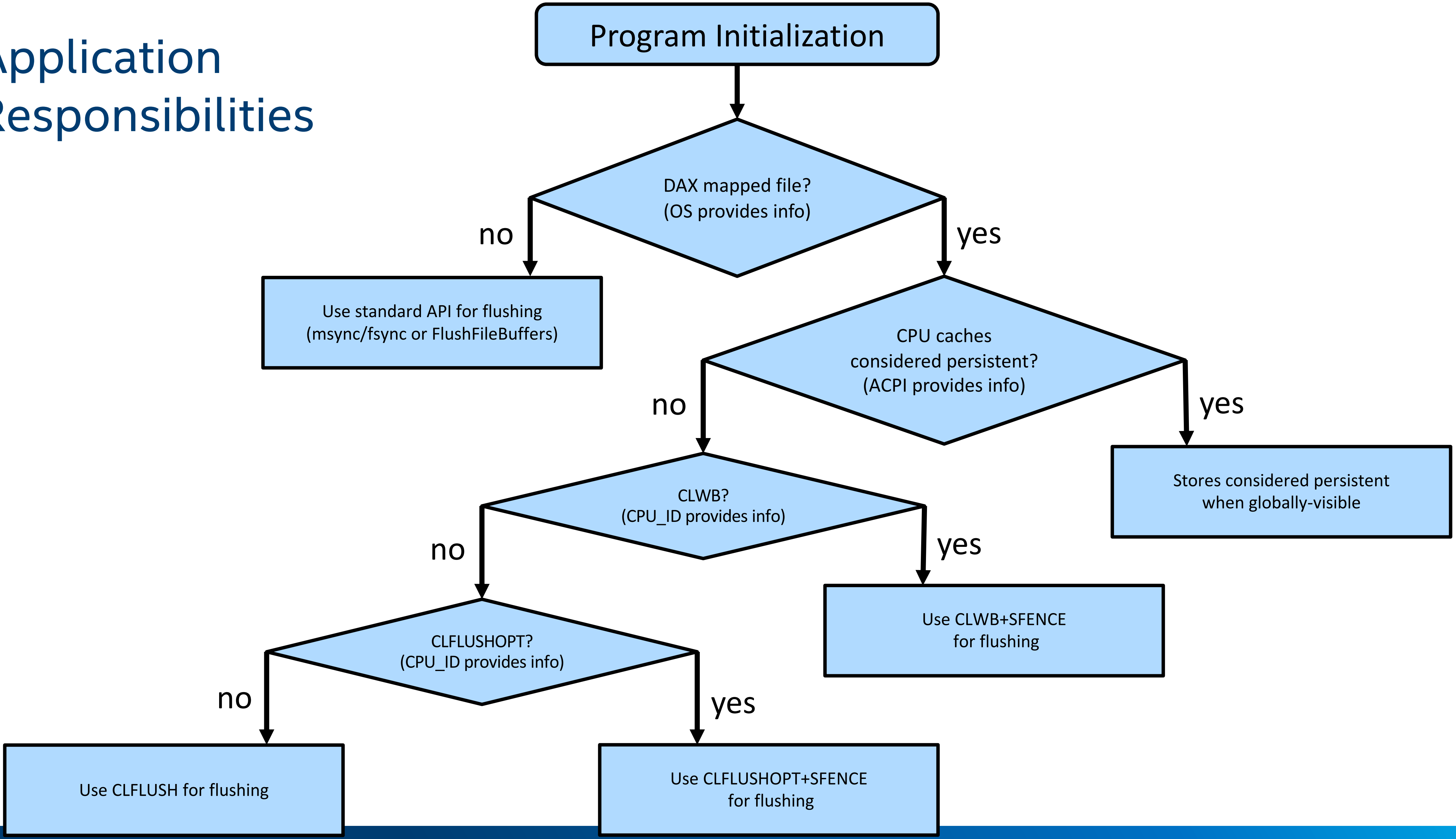
**CLWB + fence**
-or-
**CLFLUSHOPT + fence**
-or-
**CLFLUSH**
-or-
**NT stores + fence**
-or-
WBINVD (kernel only)

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

ADR
-or-
WPQ Flush (kernel only)

**Minimum Required**
**Power fail protected domain**:
Memory subsystem

## Application Responsibilities


Program Initialization
DAX mapped file? (OS provides info)
no
yes
Use standard API for flushing (msync/fsync or FlushFileBuffers)
CPU caches considered persistent? (ACPI provides info)
no
yes
Stores considered persistent when globally-visible
CLWB? (CPU_ID provides info)
no
yes
Use CLWB+SFENCE for flushing
CLFLUSHOPT? (CPU_ID provides info)
no
yes
Use CLFLUSH for flushing
Use CLFLUSHOPT+SFENCE for flushing

## PMDK

# 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 – Persistant Collection for Java

C++

C

PCJ/ LLPL

Python

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**

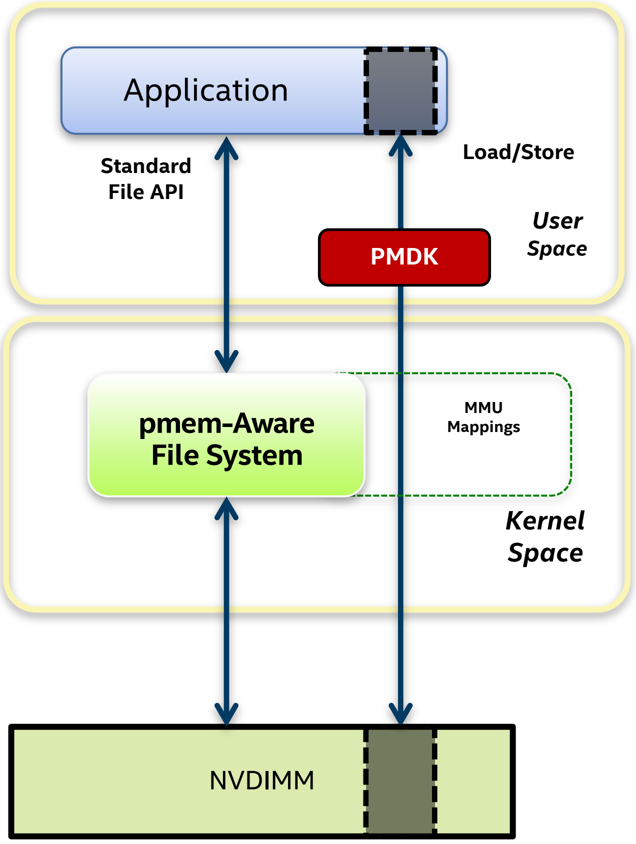
Low level support for local persistent memory

**libpmem**

Low level support for remote access to persistent memory

**librpmem**

Low-level support



In Development

## ESSENTIAL LIBPMEM KNOWLEDGE

## 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);
}
```

## ESSENTIAL LIBPMEMBLK KNOWLEDGE

# libpmemblk examples

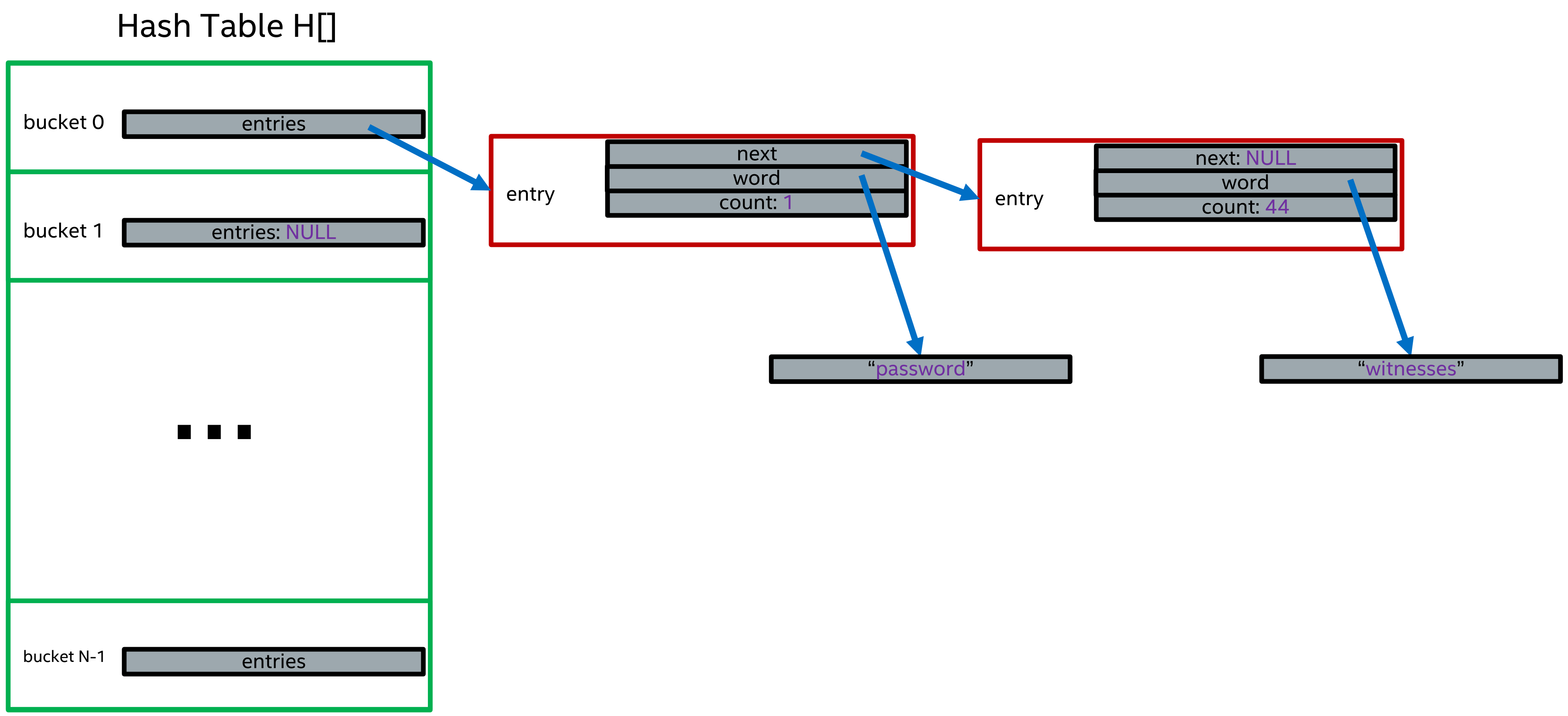
Source: [https://github.com/pmem/pmdk/blob/master/src/examples/libpmemblk](https://github.com/pmem/pmdk/blob/master/src/examples/libpmemblk)

```
/*
 * manpage.c – simple example for libpmemblk manpage
 */
```

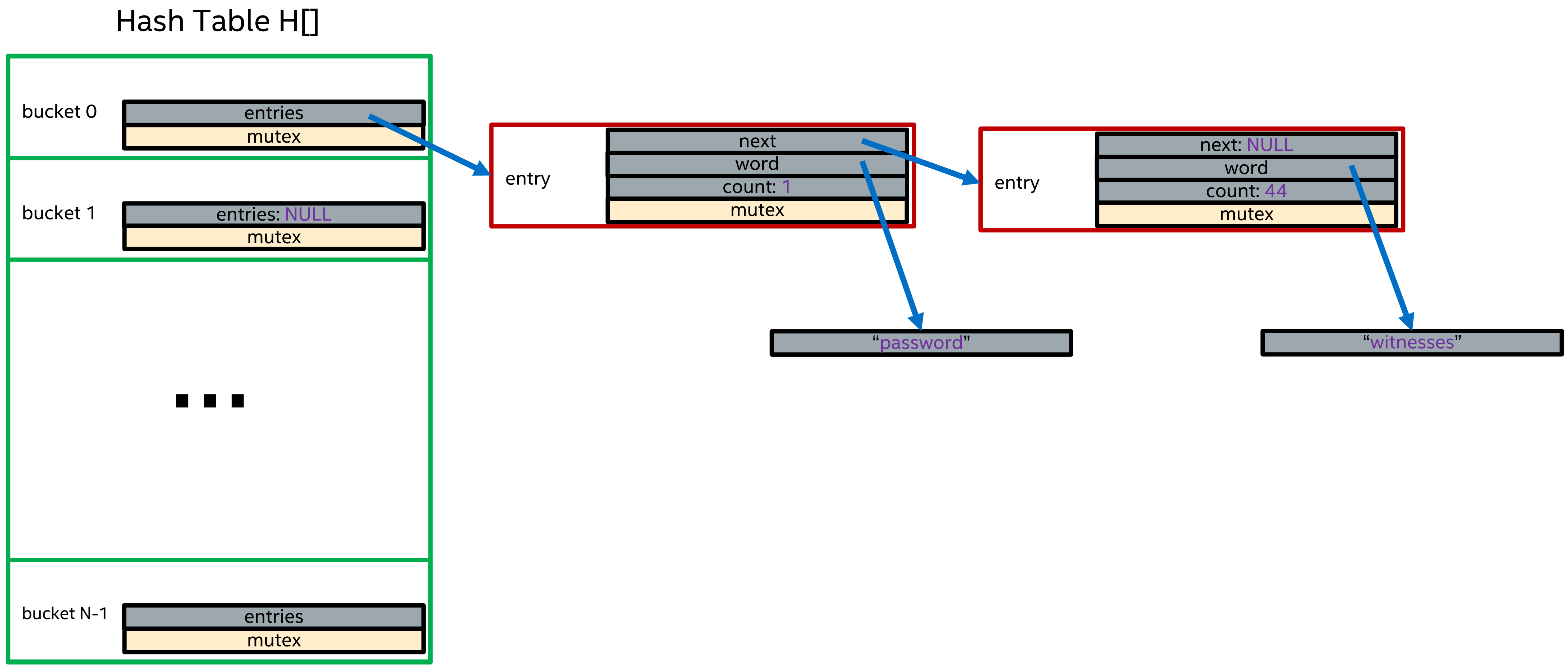
```
 /* store a block at index 5 */
strcpy(buf, "hello, world");
if (pmemblk_write(pbp, buf, 5) < 0) {
        perror("pmemblk_write");
        exit(1);
}

/* 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 EXAMPLE

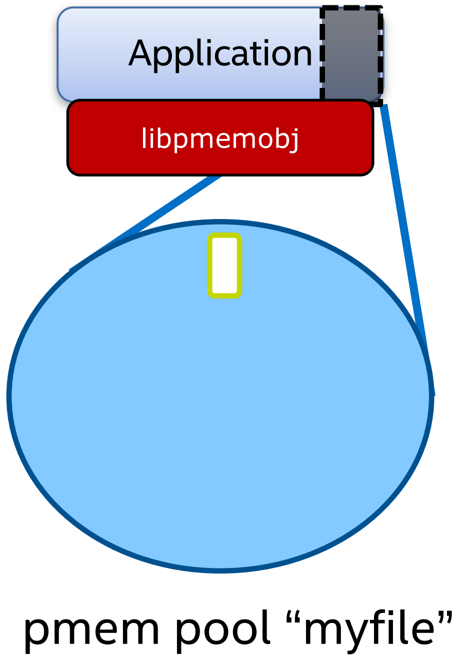
# Simple C program to build example on (nothing related to pmem yet)



# Adding multi-threading support (nothing related to pmem yet)



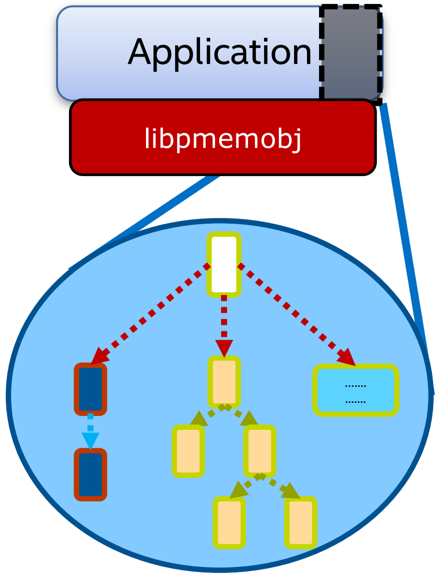
# The *Root Object*



root object:

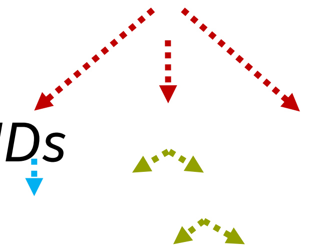
- `assume it is always there`
- `created first time accessed`
- `initially zeroed`

# Using the Root Object

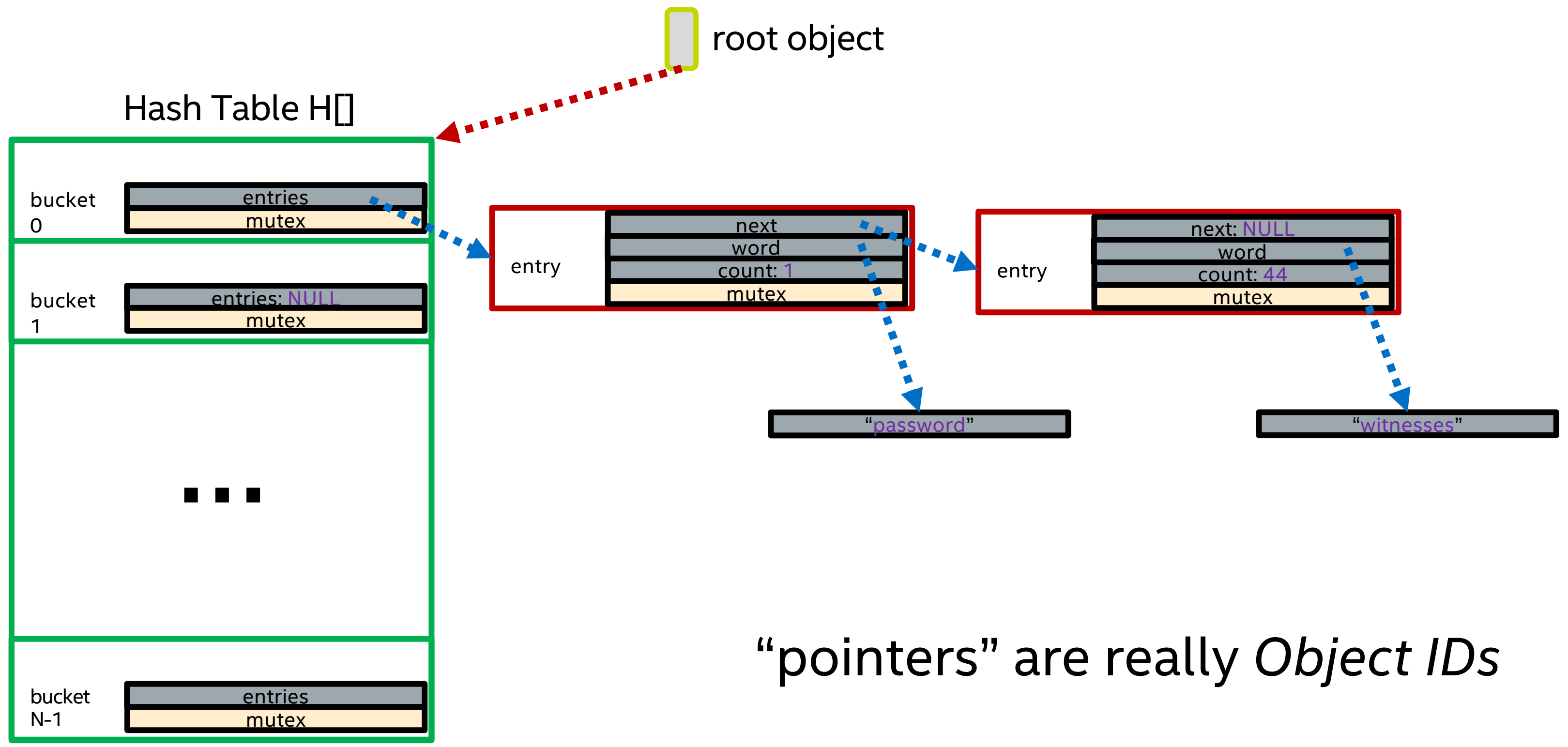


Link pmem data structures in pool off the root object to find them on each program run

“pointers” are really *Object IDs*



# Moving data 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

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

# 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)

## PCJ EXAMPLE

# Persistent Containers for Java

## Library of persistent classes

- object state stored on a persistent heap
- stored in object layout form, no serialization or deserialization
- instances behave like regular Java objects, just longer-lived
- reachability-based lifetime
- easy-to-understand data consistency model

## API for defining persistent classes

- expressiveness similar to that of regular classes
- no change to developer toolchain

## Separate library for low-level access to pmem

- byte-addressable persistent memory regions
- developer can roll their own abstractions

https://github.com/pmem/pcj

# Persistent Classes

**Primitive arrays (e.g. PersistentByteArray, mutable and immutable)**
**PersistentArray<E extends AnyPersistent> (mutable and immutable)**
**PersistentTuple<T1 extends AnyPersistent, ...> (mutable and immutable)**
**PersistentArrayList<E extends AnyPersistent>**
**PersistentHashMap<K extends AnyPersistent, V extends AnyPersistent>**
**PersistentLinkedList<E extends AnyPersistent>**
**PersistentLinkedQueue<E extends AnyPersistent>**
**PersistentSkipListMap<K extends AnyPersistent, V extends AnyPersistent>**
**PersistentFPTreeMap<K extends AnyPersistent, V extends AnyPersistent>**
**PersistentSIHashMap<K extends AnyPersistent, V extends AnyPersistent>**
**ObjectDirectory – indefinitely reachable root map of <String, T extends AnyPersistent>**
**---------------------------------------------------------------------------------------------------------------**
**Primitive types (as field and array element values, no separate class)**
**Boxed primitives (e.g. PersistentLong)**
**PersistentString**
**PersistentByteBuffer**
**PersistentUUID**
**PersistentAtomicReference<T extends AnyPersistent>**

# WordFrequency.java

```
public class WordFrequency {
    private static Map<String, Integer> counts = new TreeMap<>();

    public static void main(String[] args) {
        if (args.length == 0) System.out.println("usage: WordFrequency <list of files to process>");
        for (int i = 0; i < args.length; i++) {
            try {
                Scanner scanner = new Scanner(new File(args[i]));
                while (scanner.hasNext()) {
                    String word = scanner.next();
                    counts.merge(word, 1, Integer::sum);
                }
            }
            catch (FileNotFoundException fnf) {throw new RuntimeException(fnf.getCause());}
        }


        // print counts
        for (Map.Entry<String, Integer> e : counts.entrySet()) {
            System.out.format("%d %s\n", e.getValue().intValue(), e.getKey());
        }
    }
}
```

# ParallelWordFrequency.java

```
public class ParallelWordFrequency {
    private static Map<String, Integer> counts = new ConcurrentSkipListMap<>();

    public static void main(String[] args) throws InterruptedException {
        if (args.length == 0) System.out.println("usage: ParallelWordFrequency <list of files to process>");
        Thread[] ts = new Thread[args.length];
        for (int i = 0; i < args.length; i++) {
            int ii = i;
            ts[ii] = new Thread(() -> {
                try {
                    Scanner scanner = new Scanner(new File(args[ii]));
                    while (scanner.hasNext()) {
                        String word = scanner.next();
                        counts.merge(word, 1, Integer::sum);
                    }
                }
                catch (FileNotFoundException fnf) {throw new RuntimeException(fnf.getCause());}
            });
        }
        for (Thread t : ts) t.start();
        for (Thread t : ts) t.join();
…
```

# PersistentParallelWordFrequency.java

```
package examples.wordfrequency;


import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import lib.util.persistent.PersistentHashMap;
import lib.util.persistent.PersistentSkipListMap;
import lib.util.persistent.PersistentString;
import lib.util.persistent.PersistentInteger;
import lib.util.persistent.ObjectDirectory;
import java.util.Map;
import java.util.function.BiFunction;
```

## PersistentParallelWordFrequency.java

```
public class PersistentParallelWordFrequency {
        private static Map<PersistentString, PersistentInteger> counts = getOrInitializeCounts();


        @SuppressWarnings("unchecked")
        private static Map<PersistentString, PersistentInteger> getOrInitializeCounts() {
                String DATA_KEY = "WordFrequencyData";
                PersistentSkipListMap<PersistentString, PersistentInteger> map =
                                ObjectDirectory.get(DATA_KEY, PersistentSkipListMap.class);
                if (map == null) ObjectDirectory.put(DATA_KEY, map = new PersistentSkipListMap<>());
                return map;
        }
```

# PersistentParallelWordFrequency.java

```
public static void main(String[] args) throws InterruptedException {
        if (args.length == 0) System.out.println("usage: PersistentParallelWordFrequency <list of files to process>"
        final PersistentInteger ONE = new PersistentInteger(1);
        Thread[] ts = new Thread[args.length];
        for (int i = 0; i < args.length; i++) {
            int ii = i;
            ts[ii] = new Thread(() -> {
                try {
                    Scanner scanner = new Scanner(new File(args[ii]));
                    while (scanner.hasNext()) {
                        PersistentString word = new PersistentString(scanner.next());
                        counts.merge(word, ONE, PersistentParallelWordFrequency::sum);
                    }
                }
                catch (FileNotFoundException fnf) {throw new RuntimeException(fnf.getCause());}
            });
        }
        for (Thread t : ts) t.start();
        for (Thread t : ts) t.join();
```

# PersistentParallelWordFrequency.java

```
// print current counts
            for (Map.Entry<PersistentString, PersistentInteger> e : counts.entrySet()) {
                  System.out.format("%d %s\n", e.getValue().intValue(), e.getKey());
            }
      }


   public static PersistentInteger sum(PersistentInteger x, PersistentInteger y) {
      return new PersistentInteger(x.intValue() + y.intValue());
   }
}
```

## LINKS TO MORE INFORMATION

# Using Persistent Memory 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

# Q&A