Persistent Memory Programming
A Brief Tutorial

June 23, 2017

Andy Rudoff

Intel Corporation

DCG
Data Center Group

Links Used in This Tutorial

http://pmem.io

- Website for pmem programming, blogs, tutorials, examples

https://github.com/pmem/nvml

- Source for NVM Libraries supporting Windows, Linux in C and C++

http://pmem.io/nvml/manpages/master/libpmemobj.3.html

- libpmemobj man page (for C programming)

http://pmem.io/nvml/cpp_obj/master/cpp_html/index.html

- libpmemobj C++ interface documentation

https://github.com/pmem/nvml/tree/master/src/examples

- NVML examples, all buildable and runnable

https://github.com/andyrudoff/pmemtutorial

- The slides for this tutorial and the code examples (word frequency count)

Links to Additional Information

https://www.usenix.org/system/files/login/articles/login_summer17_07_rudoff.pdf

- An overview of persistent memory programming

http://www.snia.org/PM

SNIA Standards Portfolio

- NVM Programming Model v1.2a – Draft for public review
- NVM Programming Model v1.1- SNIA Technical Position
- NVM Programming Model v1.0 - SNIA Technical Position

SNIA Technical White Papers

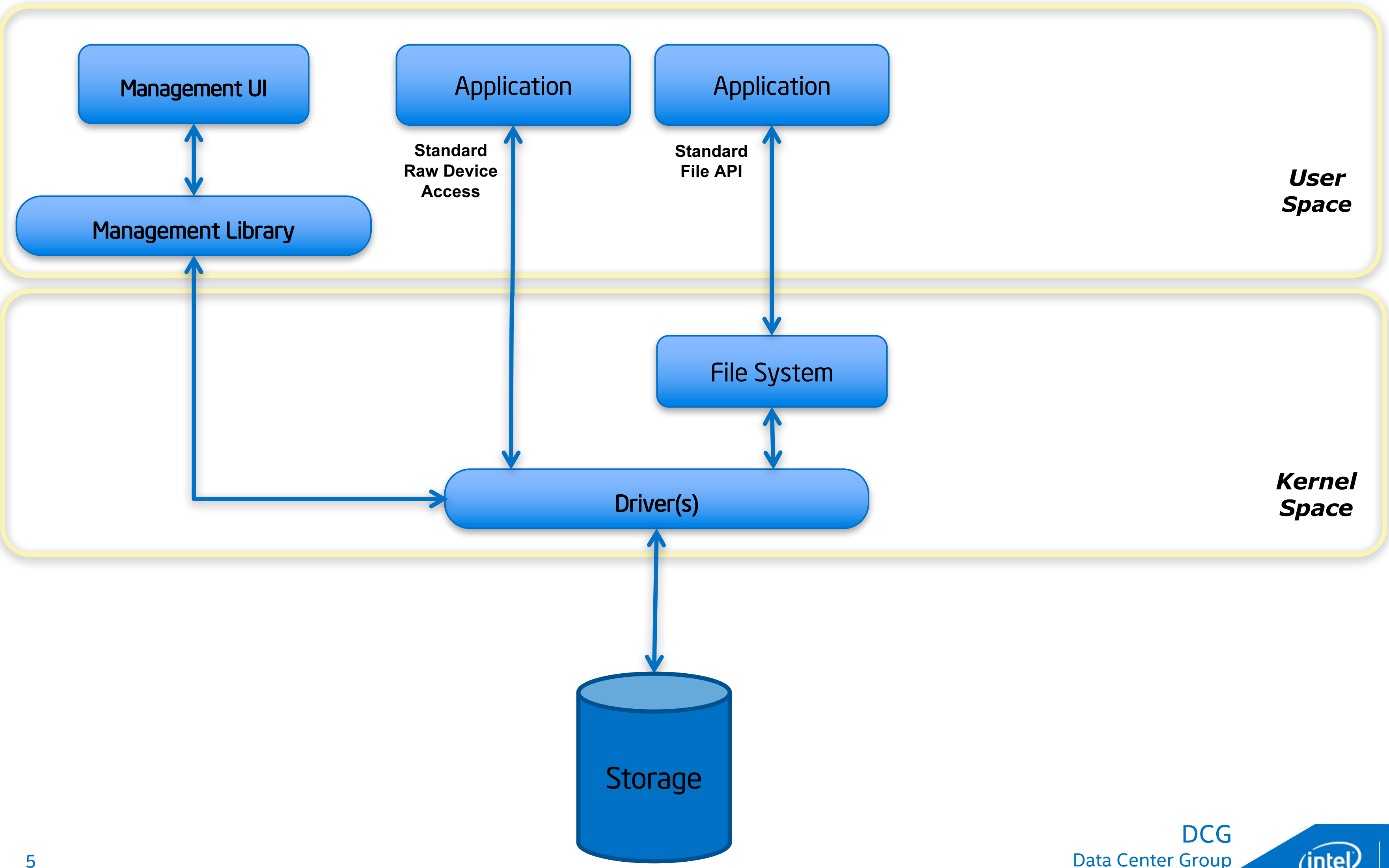
- NVM PM Remote Access for High Availability
- Persistent Memory Atomics and Transactions

SNIA Videos and Presentations

- The SNIA NVM Programming Model – Latest Developments and Challenges
- Persistent Memory Summit 2017

Background (abbreviated)

The Storage Stack (50,000ft view…)



A Programmer’s View

(not just C programmers!)

```
fd = open(“/my/file”, O_RDWR);

…

count = read(fd, buf, bufsize);

…

count = write(fd, buf, bufsize);

…

close(fd);
```

“Buffer-Based”

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”

Memory-Mapped Files

What are memory-mapped files really?

- Direct access to the **page cache**
- Storage only supports block access (paging)

With load/store access, when does I/O happen?

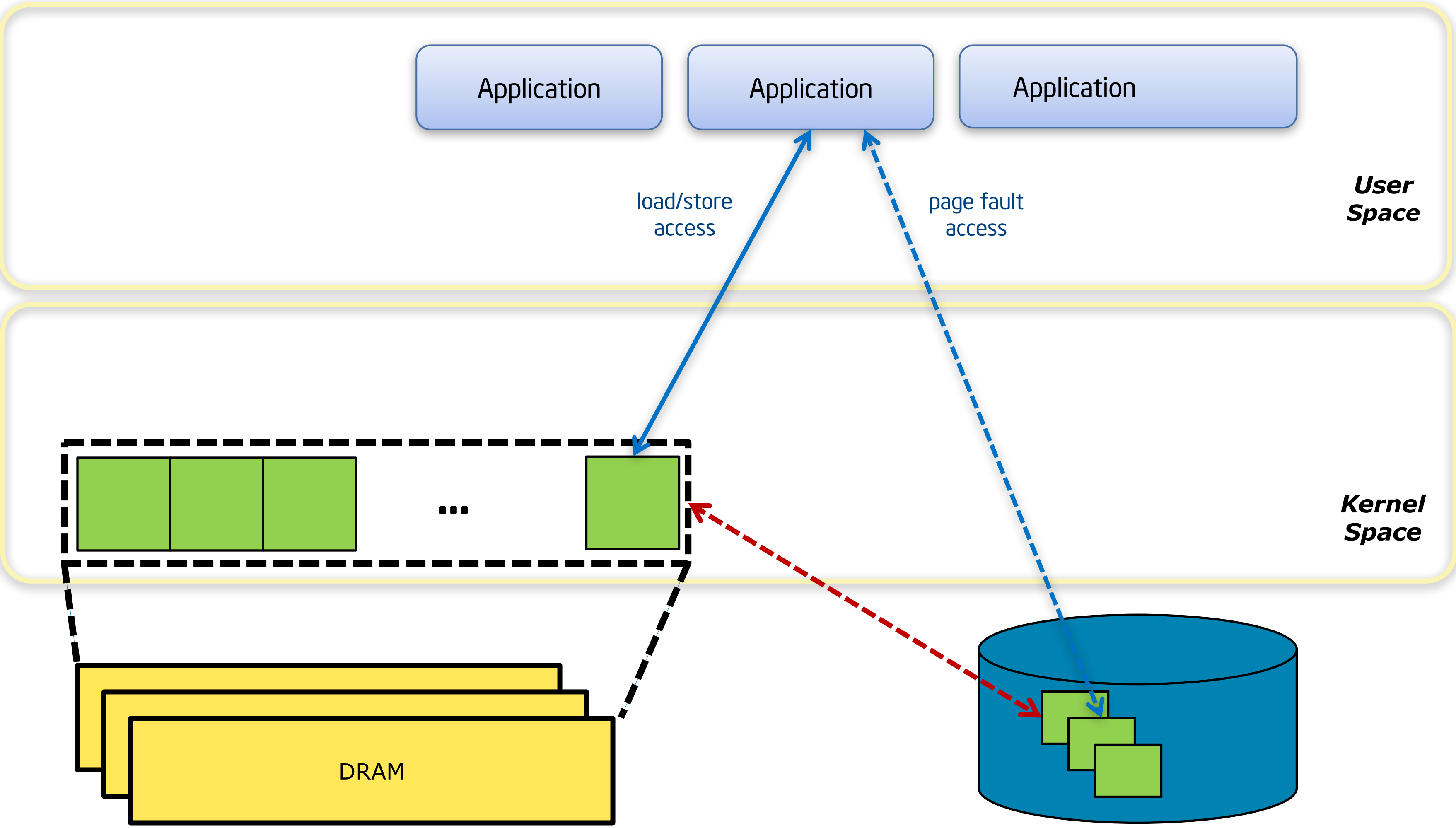
- Read faults/Write faults
- Flush to persistence

Not that commonly used or understood

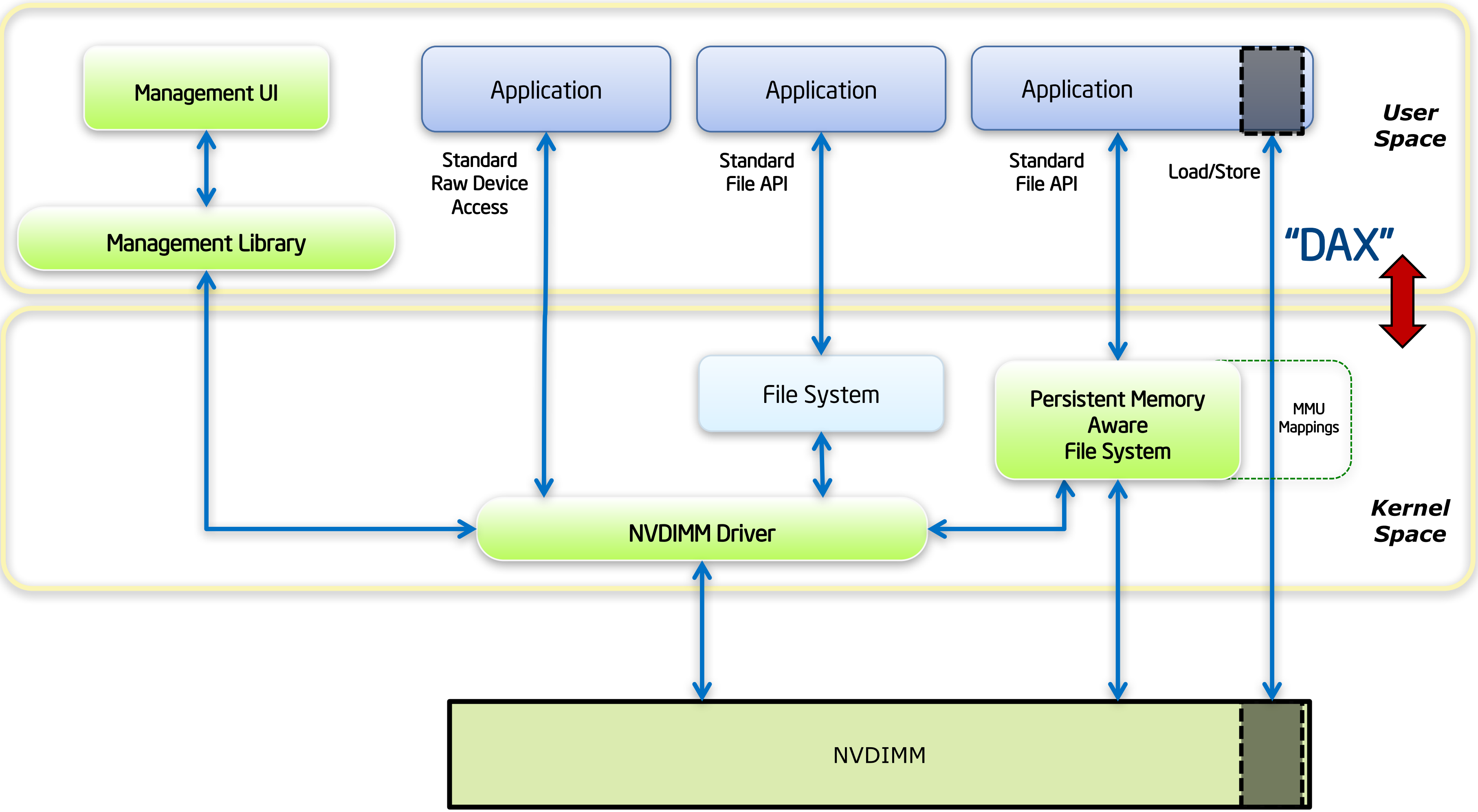
- Quite powerful
- Sometimes used without realizing it

Good reference: http://nommu.org/memory-faq.txt

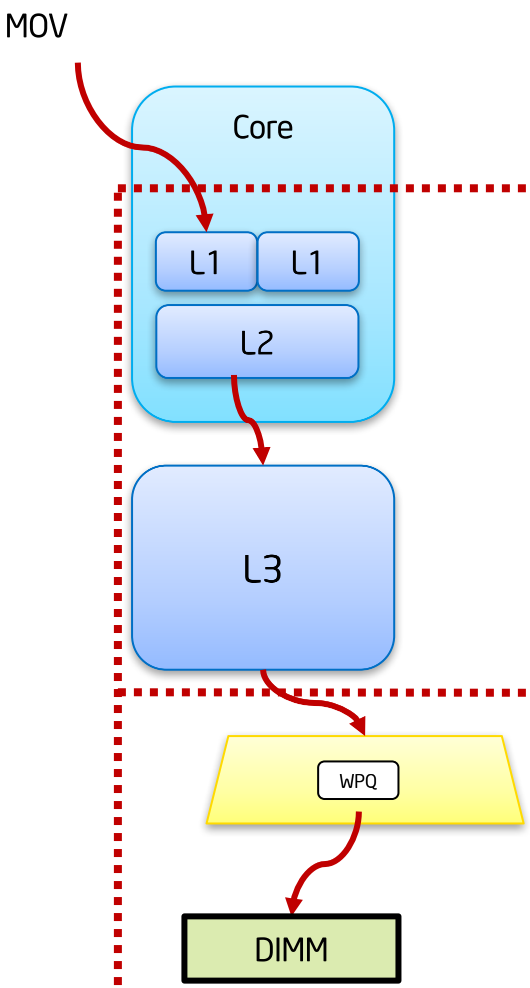
OS Paging


Application
Application
Application
load/store access
page fault access
User Space
...
Kernel Space
DRAM

Exposing pmem to Applications



The Persistent Domain



CPU CACHES

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

WPQ

ADR
-or-
WPQ Flush (kernel only)

Minimum Required
Power fail protected domain:
Memory subsystem

Flushing for Application Programmers

Why is flushing required?

- Memory-mapped files have always worked this way:
  - Stores are not guaranteed persistent until flush API is called
  - Stores are *visible* before they are persistent

Do standard flushing APIs work with pmem?

- Yes, standard APIs work as expected
  - `msync()` on Linux
  - `FlushFileBuffers()` on Windows
  - The kernel will use instructions like CLWB as necessary

Can Applications just flush with CLWB from user space

- Only when supported by the kernel/file system
- Libraries like NVML determine when it is safe

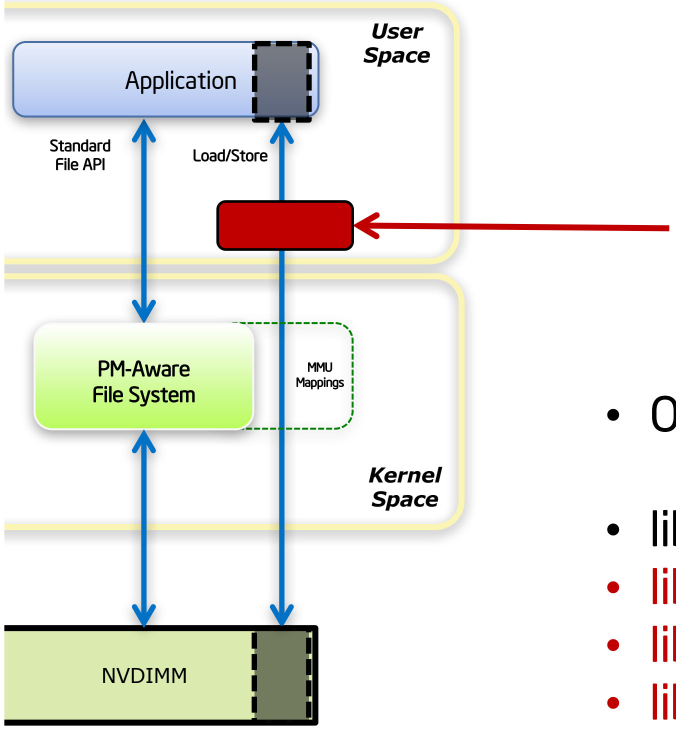
State of Ecosystem Today

OS Detection of NVDIMMs	**ACPI 6.0+**
OS Exposes pmem to apps	***DAX* provides SNIA Programming Model** Fully supported: • Linux (ext4, XFS) • Windows (NTFS)
OS Supports Optimized Flush	Specified, but evolving (ask when safe) • Linux: **unsafe** except Device DAX • (and new file systems like **NOVA**) • Windows: **safe**
Remote Flush	Proposals under discussion (works today with extra round trip)
Deep Flush	Upcoming Specification
Transactions, Allocators	Built on above via libraries and languages: • http://pmem.io **Much more language support to do**
Virtualization	All VMMs planning to support PM in guest (KVM changes upstream, Xen coming, others too...)

NVM Libraries

NVM Libraries: pmem.io

C/C++ on Linux and Windows



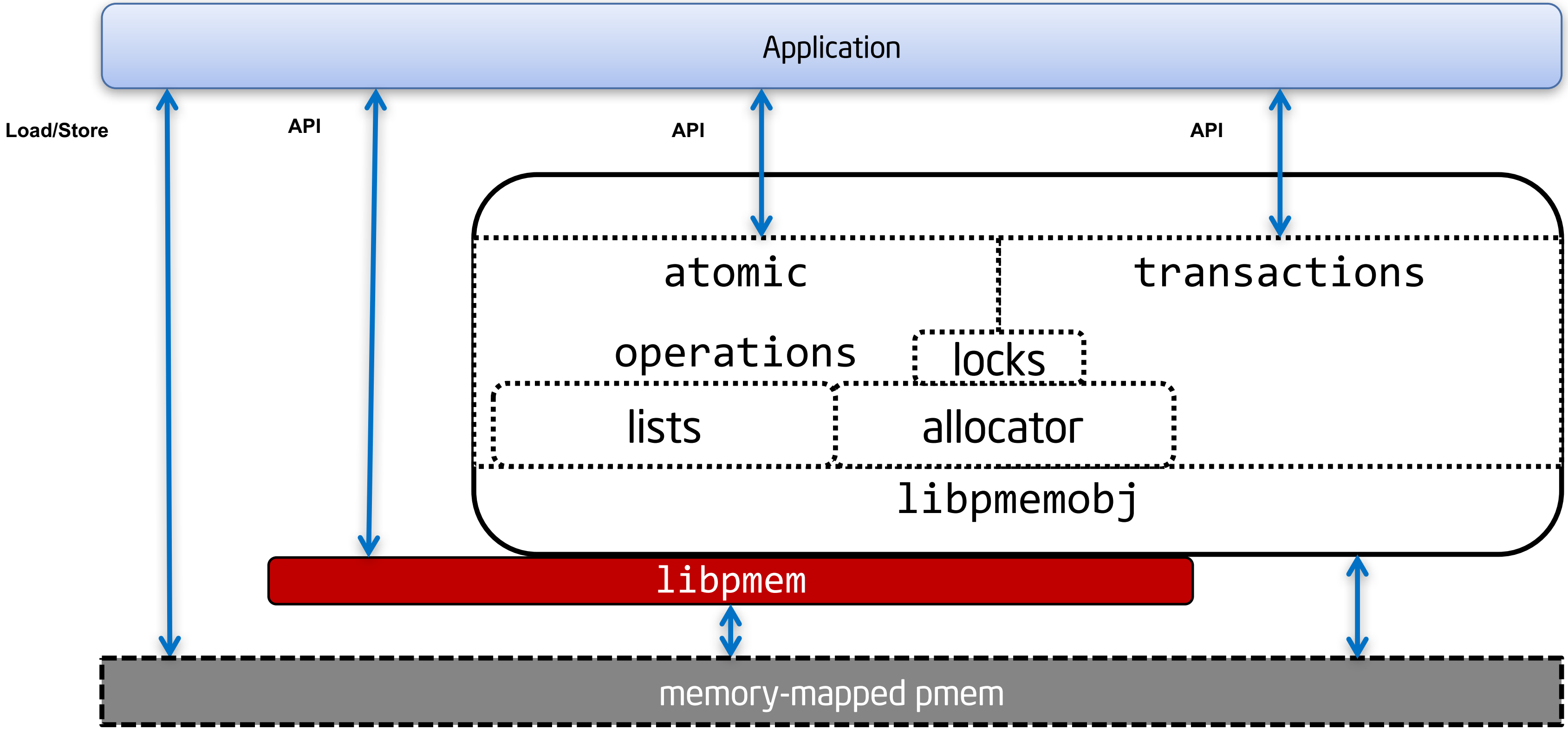
- Open Source
  - http://pmem.io
- libpmem
- libpmemobj
- libpmemblk
- libpmemlog
- libvmem

Transactional

More libraries being added to the suite over time

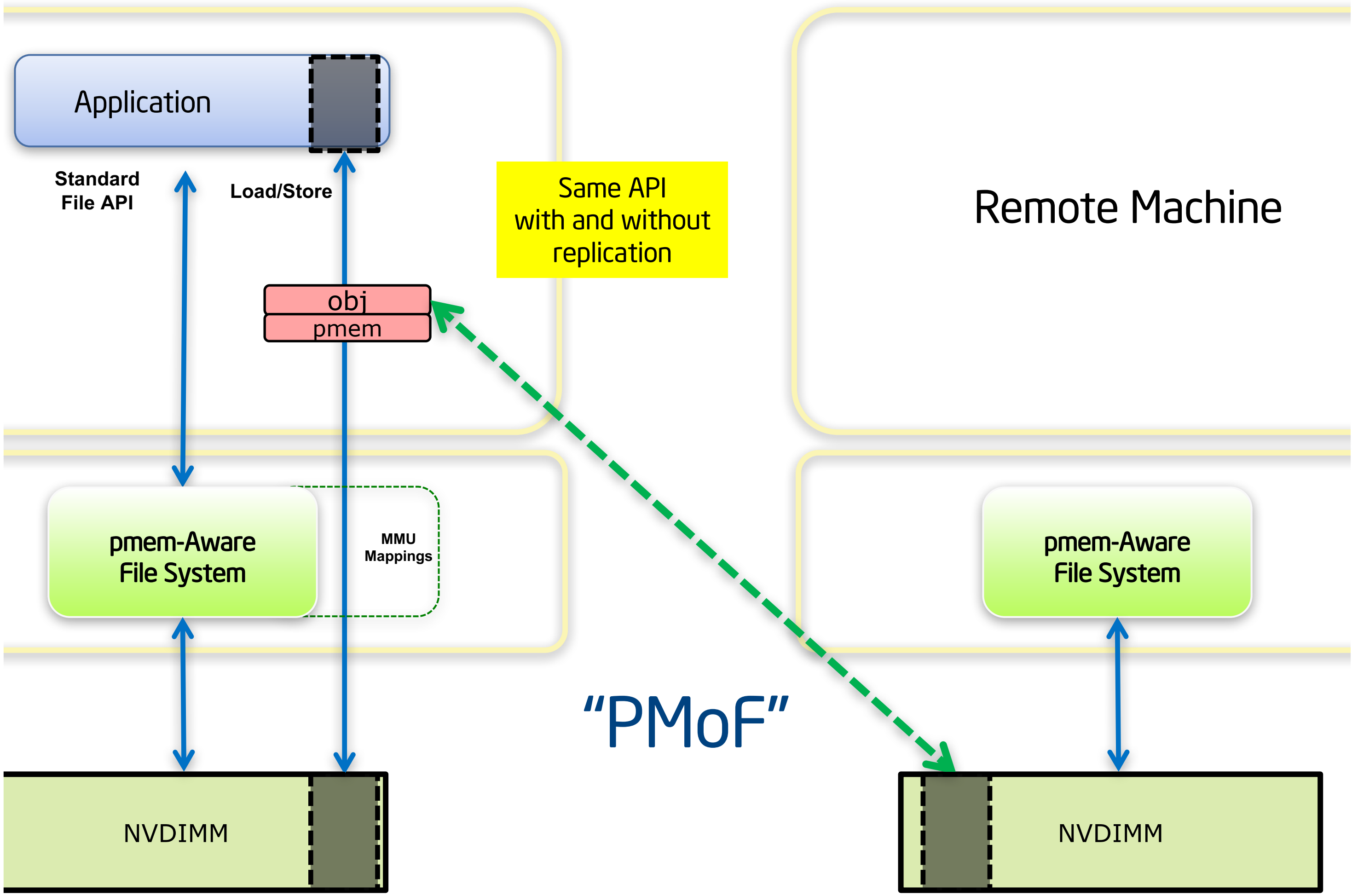
libpmemobj

“transactional object store”



Libpmemobj Replication: Application Transparent

(except for performance overhead)



Using **libpmemobj** Today

From C

- Fully validated, delivered on Linux, early access on Windows
- Can stick to pure API calls, but macros add:
  - Compile-time type safety
  - Transaction syntax, similar to try/catch

From C++

- Fully validated, delivered on Linux, early access on Windows
- Use C++ type system & syntax: much cleaner, less error-prone

From Java

- Persistent Containers for Java (Experimental)

From Python

- PyNVM (Experimental)

Other work

- valgrind (and a similar tool coming from Intel)
- JavaScript (Pre-release)

Emulating Persistent Memory

The programming model builds on memory-mapped files

- So development on memory-mapped files works fine
  - NVML will use msync() to flush to persistence
  - Non-optimal performance
- Use any 64-bit Linux or Windows

For benchmarking:

- http://pmem.io/2016/02/22/pm-emulation.html
- Distros like Fedora 24 are built with DAX/pmem
  - Avoids making you build a kernel
  - Also avoids building NVML

Using NVML on Fedora 24 or later…

```
fedora24 # dnf install libpmemobj-devel
Last metadata expiration check: 0:08:18 ago on Wed Sep 14 14:58:49 2016.
Dependencies resolved.
=====================================================================================
 Package                  Arch           Version               Repository       Size
=====================================================================================
Installing:
 libpmem                  x86_64         1.1-1.fc24            updates          29 k
 libpmem-devel            x86_64         1.1-1.fc24            updates          43 k
 libpmemobj               x86_64         1.1-1.fc24            updates          66 k
 libpmemobj-devel         x86_64         1.1-1.fc24            updates         112 k

Transaction Summary
=====================================================================================
Install  4 Packages

Total download size: 251 k
Installed size: 527 k
Is this ok [y/N]: y
```

```
Downloading Packages:
(1/4): libpmem-devel-1.1-1.fc24.x86_64.rpm            81 kB/s |  43 kB     00:00
(2/4): libpmemobj-devel-1.1-1.fc24.x86_64.rpm        184 kB/s | 112 kB     00:00
(3/4): libpmem-1.1-1.fc24.x86_64.rpm                 209 kB/s |  29 kB     00:00
(4/4): libpmemobj-1.1-1.fc24.x86_64.rpm               98 kB/s |  66 kB     00:00
-------------------------------------------------------------------------------------
Total                                                153 kB/s | 251 kB     00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Installing  : libpmem-1.1-1.fc24.x86_64                                        1/4
  Installing  : libpmem-devel-1.1-1.fc24.x86_64                                  2/4
  Installing  : libpmemobj-1.1-1.fc24.x86_64                                     3/4
  Installing  : libpmemobj-devel-1.1-1.fc24.x86_64                               4/4
  Verifying   : libpmemobj-devel-1.1-1.fc24.x86_64                               1/4
  Verifying   : libpmem-devel-1.1-1.fc24.x86_64                                  2/4
  Verifying   : libpmemobj-1.1-1.fc24.x86_64                                     3/4
  Verifying   : libpmem-1.1-1.fc24.x86_64                                        4/4

Installed:
  libpmem.x86_64 1.1-1.fc24              libpmem-devel.x86_64 1.1-1.fc24
  libpmemobj.x86_64 1.1-1.fc24           libpmemobj-devel.x86_64 1.1-1.fc24

Complete!
```

The `pmempool` command

(`nvml-tools` Package)

pmempool-info(1)
Prints information and statistics in human-readable format about specified pool.

pmempool-check(1)
Checks pool’s consistency and repairs pool if it is not consistent.

pmempool-create(1)
Creates a pool of specified type with additional properties specific for this type of pool.

pmempool-dump(1)
Dumps usable data from pool in hexadecimal or binary format.

pmempool-rm(1)
Removes pool file or all pool files listed in poolset configuration file.

pmempool-convert(1)
Updates the pool to the latest available layout version.

Tour Through http://pmem.io and NVML Source Tree

Essential libpmem Knowledge

libpmem examples

Source: https://github.com/pmem/nvml/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);
}
```

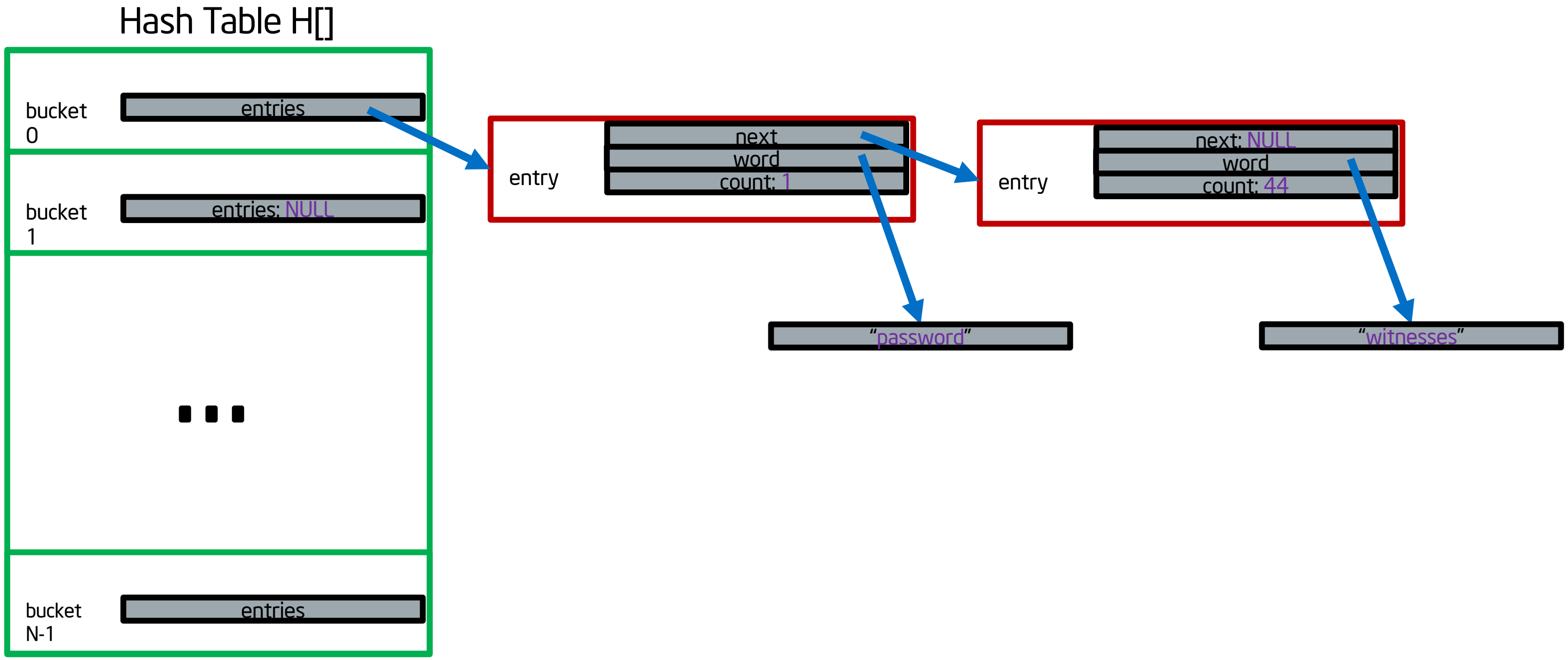
libvmem Example

Volatile use of Persistent Memory

```
if ((vmp = vmem_create("/pmem-fs", VMEM_MIN_POOL)) == NULL) {
    perror("vmem_create");
    exit(1);
}

if ((ptr = vmem_malloc(vmp, 100)) == NULL) {
    perror("vmem_malloc");
    exit(1);
}

strcpy(ptr, "hello, world");

/* give the memory back */
vmem_free(vmp, ptr);

/* ... */

vmem_delete(vmp);
```

libpmemobj Examples

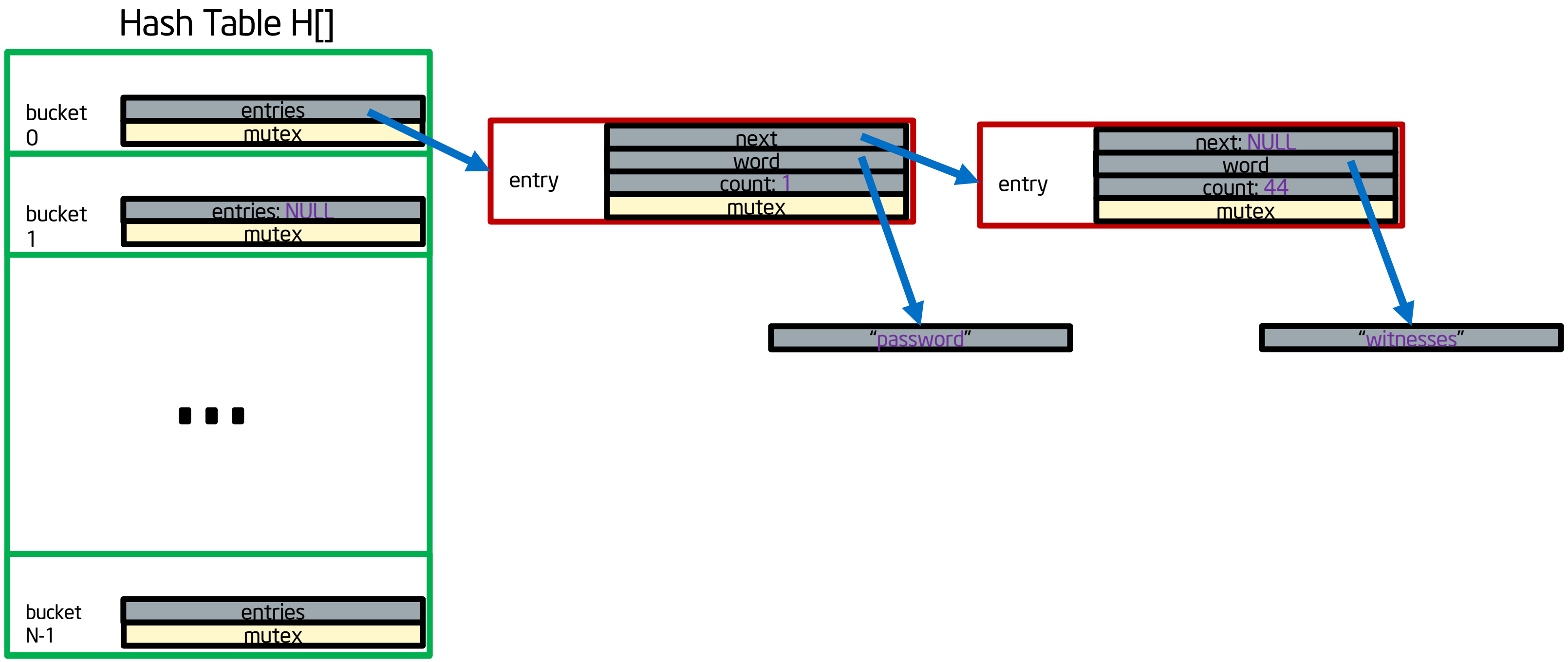
Simple C program to build example on (nothing related to pmem yet)



freq.c

```
$ freq -p words.txt
1 is
1 all
1 for
2 to
1 men
1 good
2 the
1 come
1 their
1 Now
1 time
1 country
1 aid
1 of
```

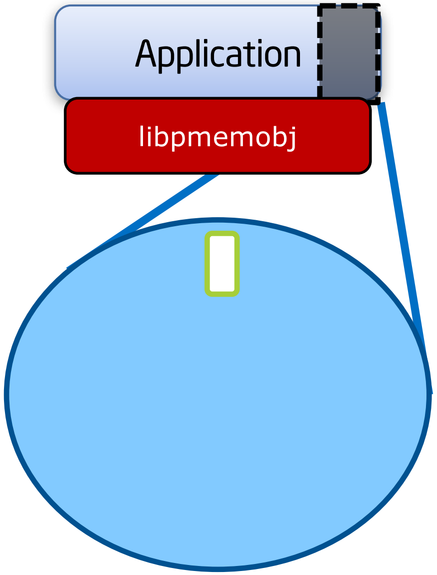
Adding multi-threading support (nothing related to pmem yet)



freq_mt.c

```
$ freq_mt -p words.txt words.txt words.txt
3 is
3 all
3 for
6 to
3 men
3 good
6 the
3 come
3 their
3 Now
3 time
3 country
3 aid
3 of
```

The *Root Object*

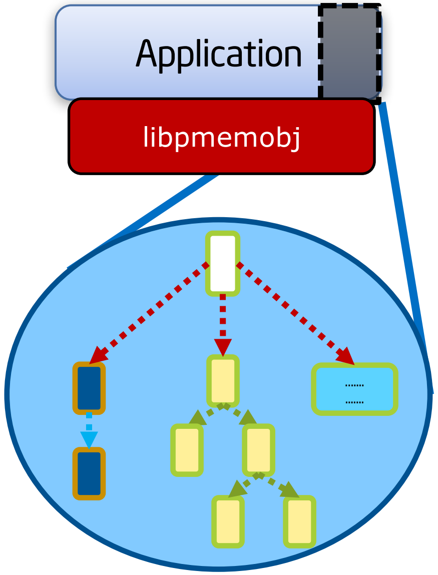


pmem pool “myfile”

root object:

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

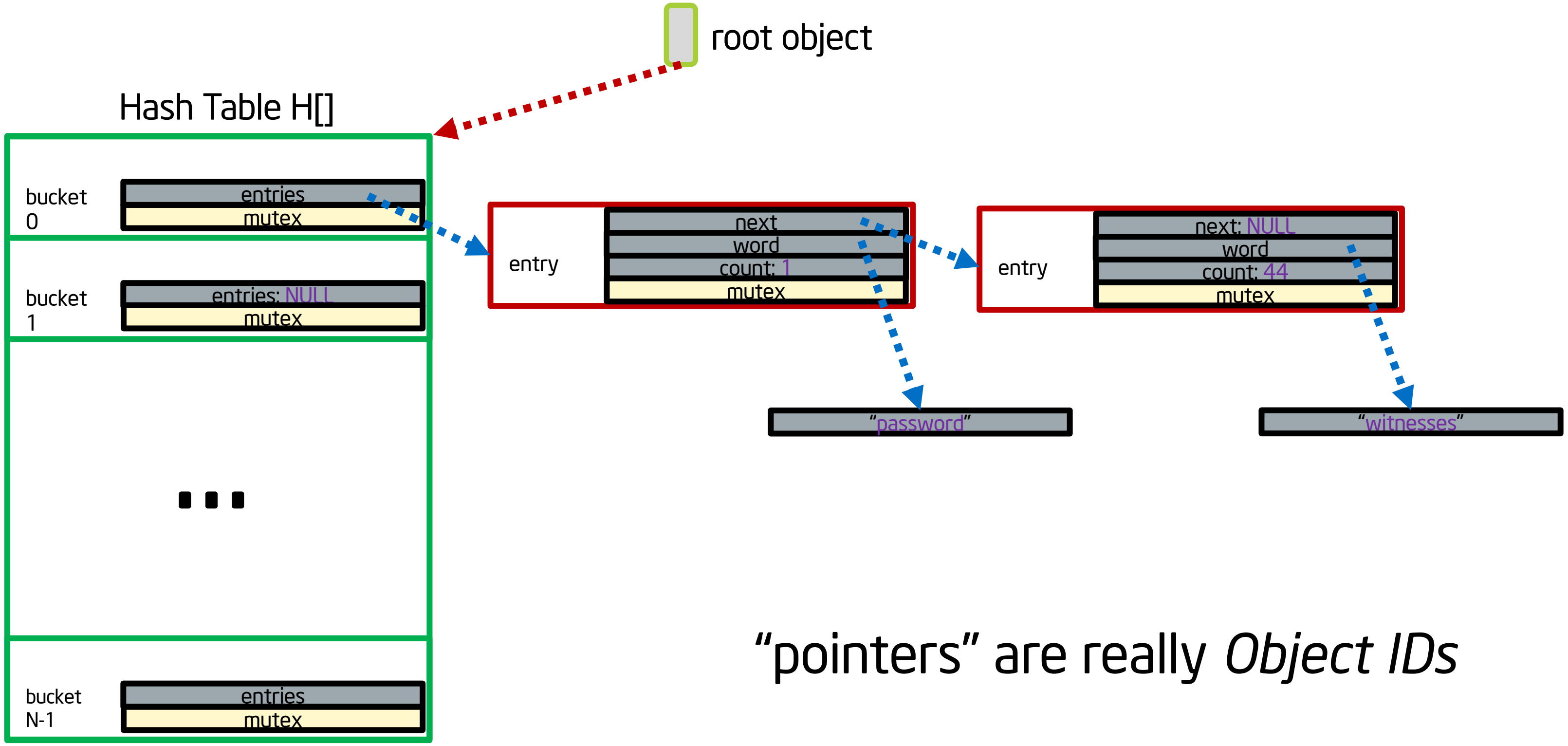
Using the Root Object



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

“pointers” are really *Object IDs*

Moving data the example to pmem



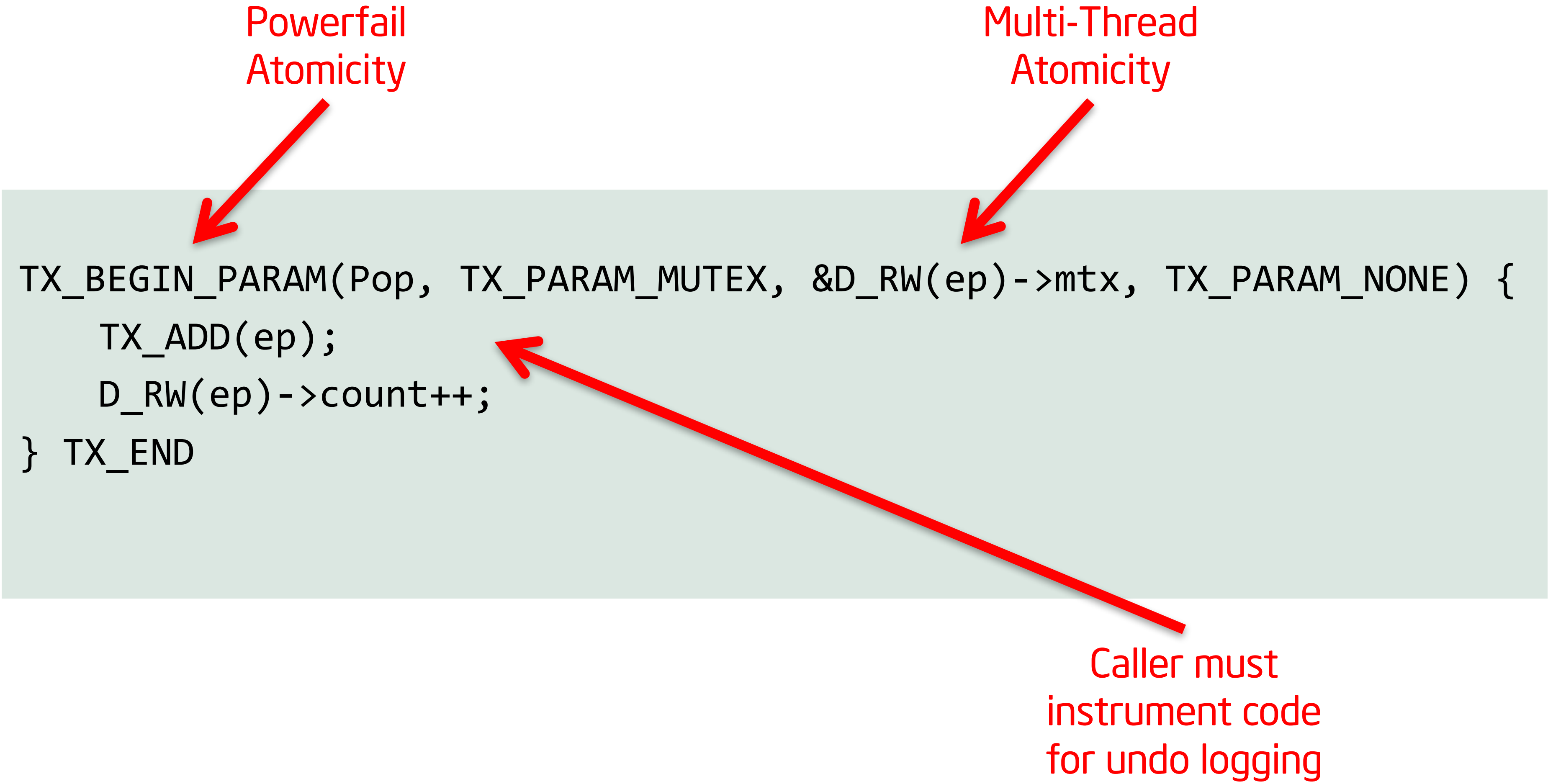
"pointers" are really *Object IDs*

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

freq_pmem.c

```
$ pmempool create obj --layout=freq -s 1G freqcount

$ freq_pmem_print freqcount

$ freq_pmem freqcount words.txt words.txt words.txt

$ freq_pmem_print freqcount
3 is
3 all
3 for
6 to
3 men
3 good
6 the
…
```

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::exec_tx(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)

freq_pmem_cpp.c

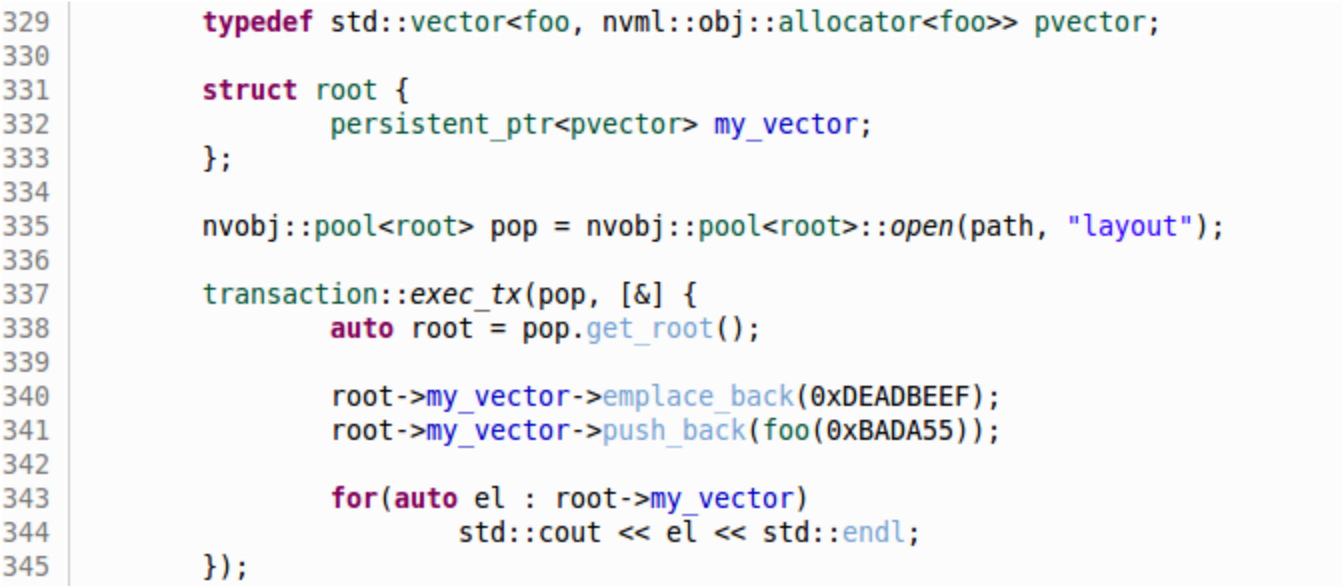
```
$ freq_pmem_cpp freqcount words.txt words.txt words.txt

$ freq_pmem_print freqcount
6 is
6 all
6 for
12 to
6 men
6 good
12 the
6 come
6 their
6 Now
6 time
6 country
6 aid
6 of
```

Future C++ Programming with libpmemobj

Persistent memory containers

A proof of concept under way.

- Targeting libc++ and libstdc++



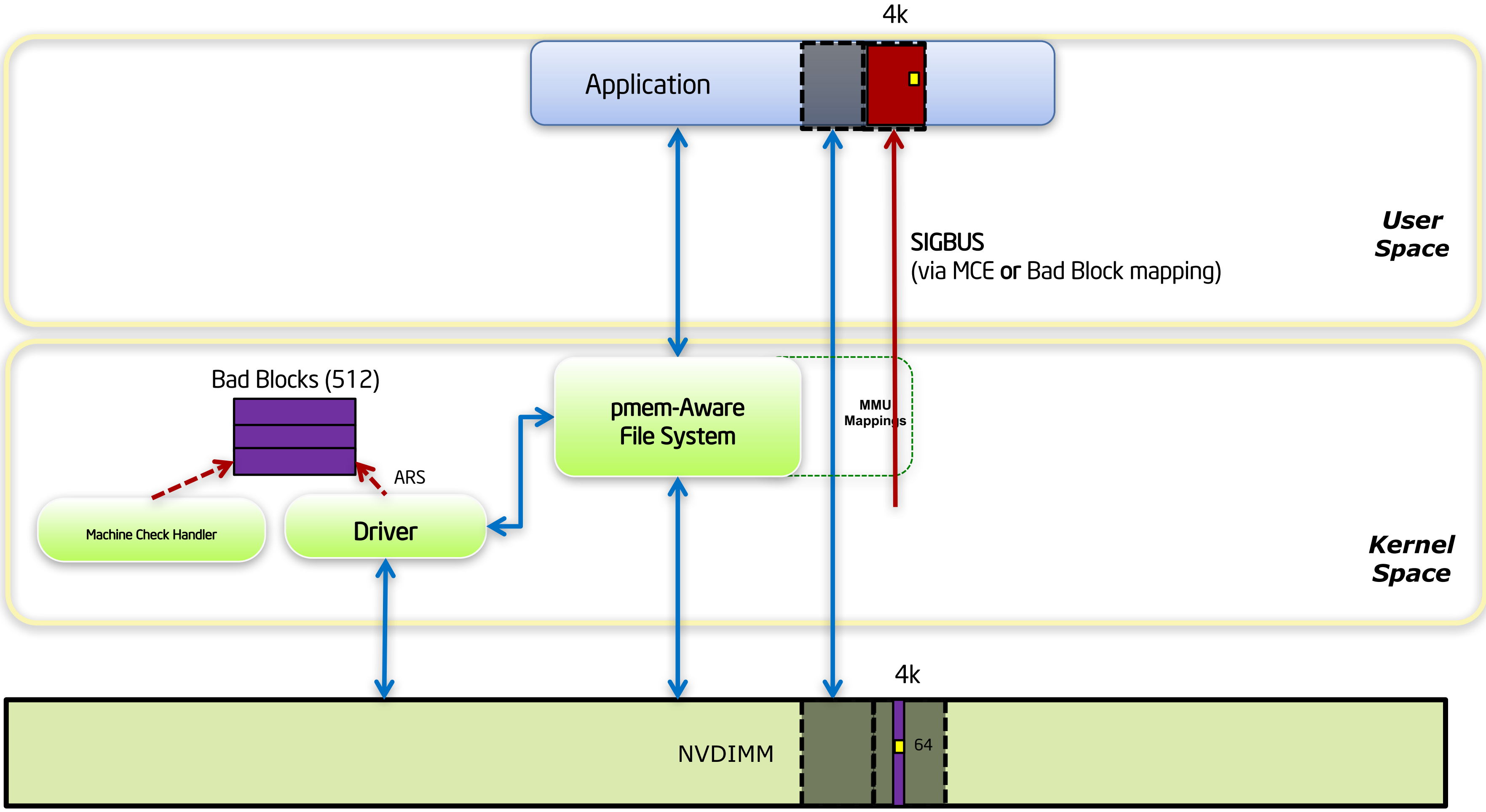
```
        typedef std::vector<foo, nvml::obj::allocator<foo>> pvector;

        struct root {
                persistent_ptr<pvector> my_vector;
        };

        nvobj::pool<root> pop = nvobj::pool<root>::open(path, "layout");

        transaction::exec_tx(pop, [&] {
                auto root = pop.get_root();

                root->my_vector->emplace_back(0xDEADBEEF);
                root->my_vector->push_back(foo(0xBADA55));

                for(auto el : root->my_vector)
                        std::cout << el << std::endl;
        });
```

The Inconvenient Truth

Flush On Fail Fail

- ADR Failure Detection
  - Once detected, what SW knows the recovery action?
    - Usually the App
    - Ordering information lost
  - libpmem additions in progress
- pmem-Based Block Storage Errors
  - Without extra hardware, comes in as Machine Check
  - With hardware, can return to driver model
- Uncorrectables…

Linux Example:
Blast radius 64B → 4k



Summary

Tutorial Summary

Persistent Memory

- Emerging technology, game changing, large capacity
- New programming models allow greater leverage

NVM Libraries

- http://pmem.io
- Convenience, not a requirement
- Transactions, memory allocation, language support
- More coming

We don't know all the answers yet

- The next few years are going to be pretty exciting!

intel®