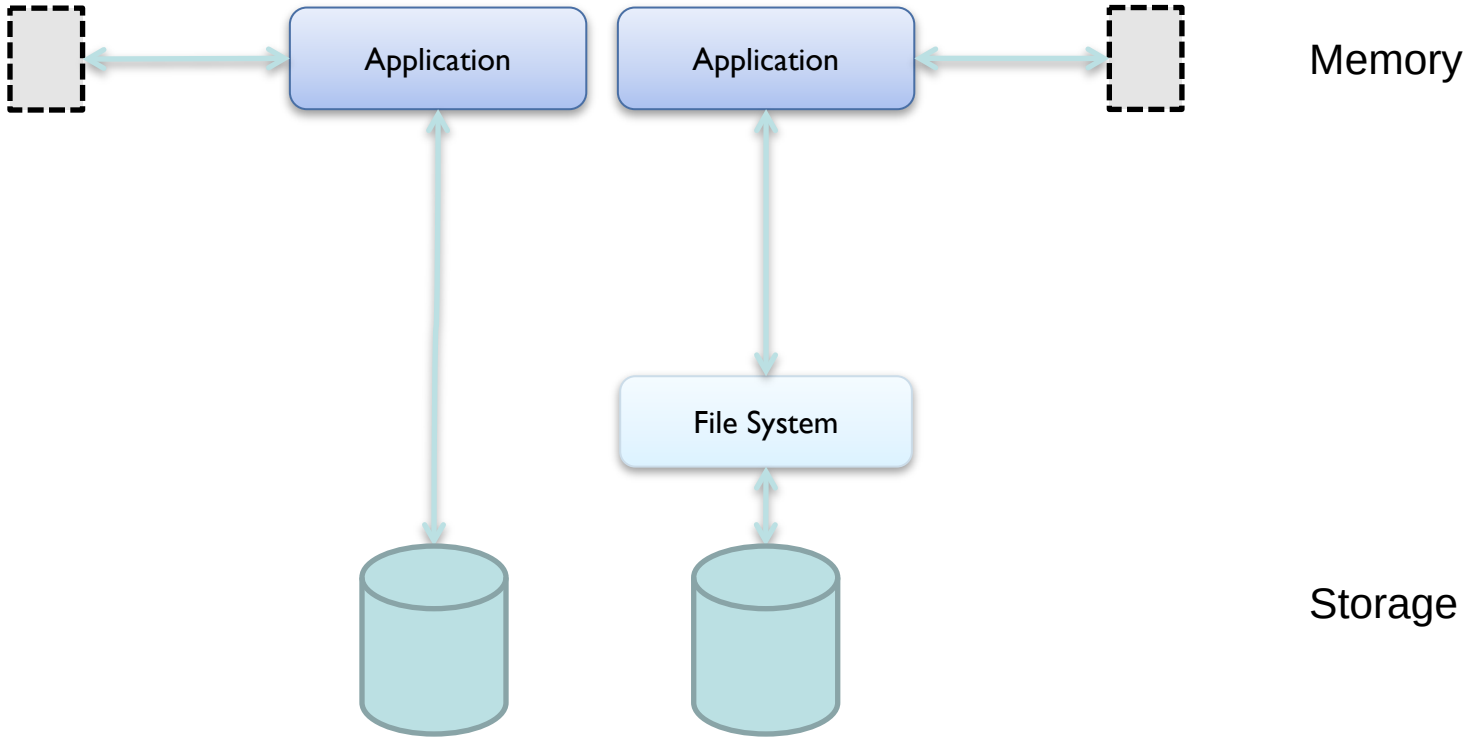
SDC
STORAGE DEVELOPER CONFERENCE
SNIA SANTA CLARA, 2016

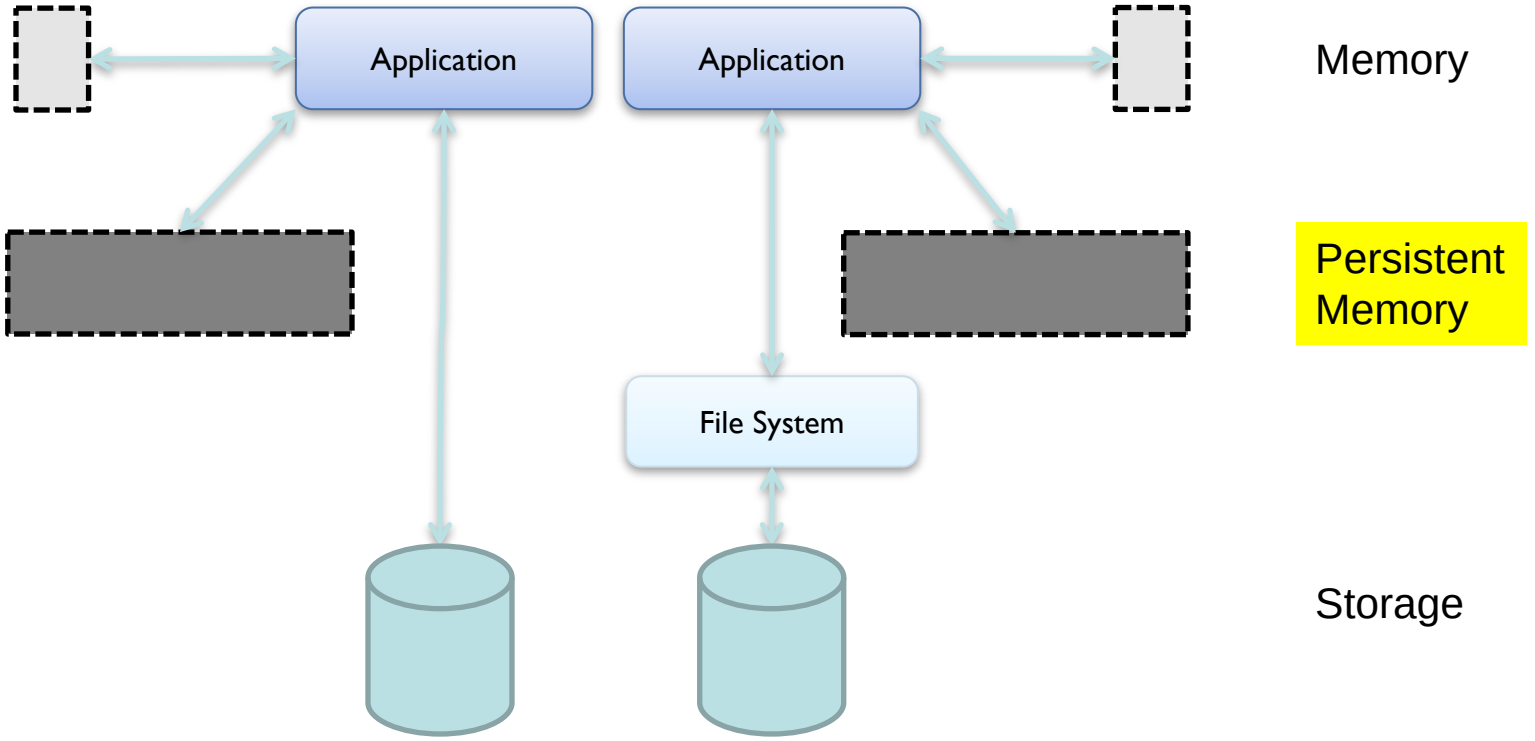
# Persistent Memory Quick Start Programming Tutorial

**Andy Rudoff**
**Intel Corporation**

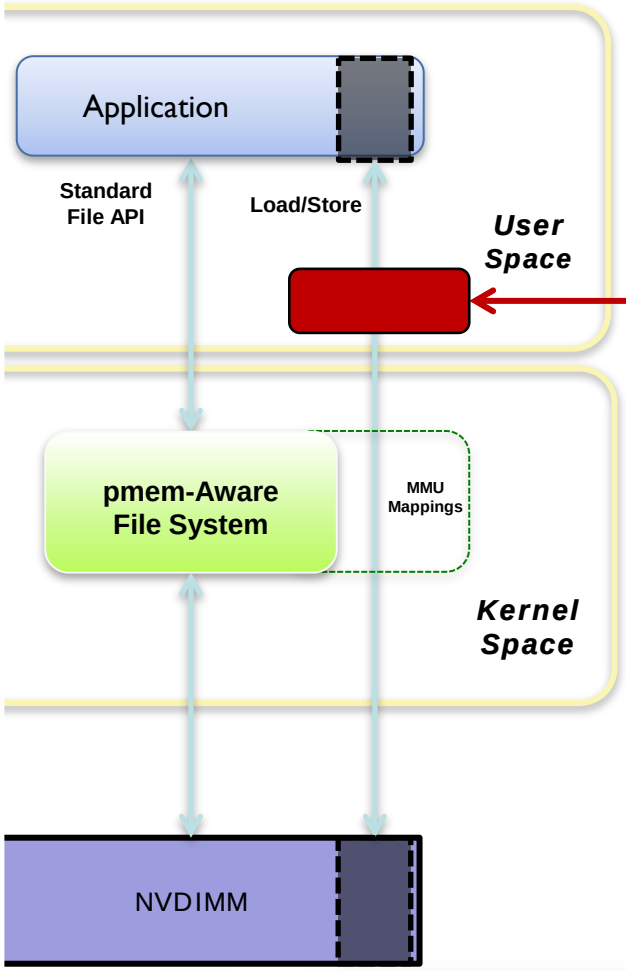
# The Past: Two Primary Tiers for Run-Time Data



# Moving to Three Tiers



# Modifying Applications for pmem…




Library

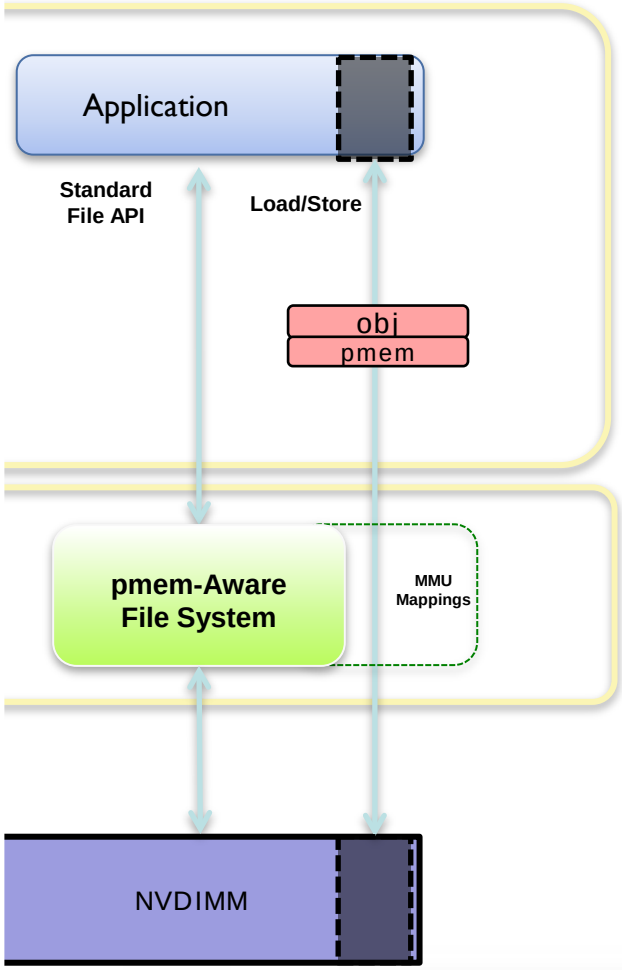
- Open Source
  - http://pmem.io
- libpmem
- libpmemobj
- libpmemblk
- libpmemlog
- libvmem
- libvmmalloc

Transactional

# What is libpmemobj?

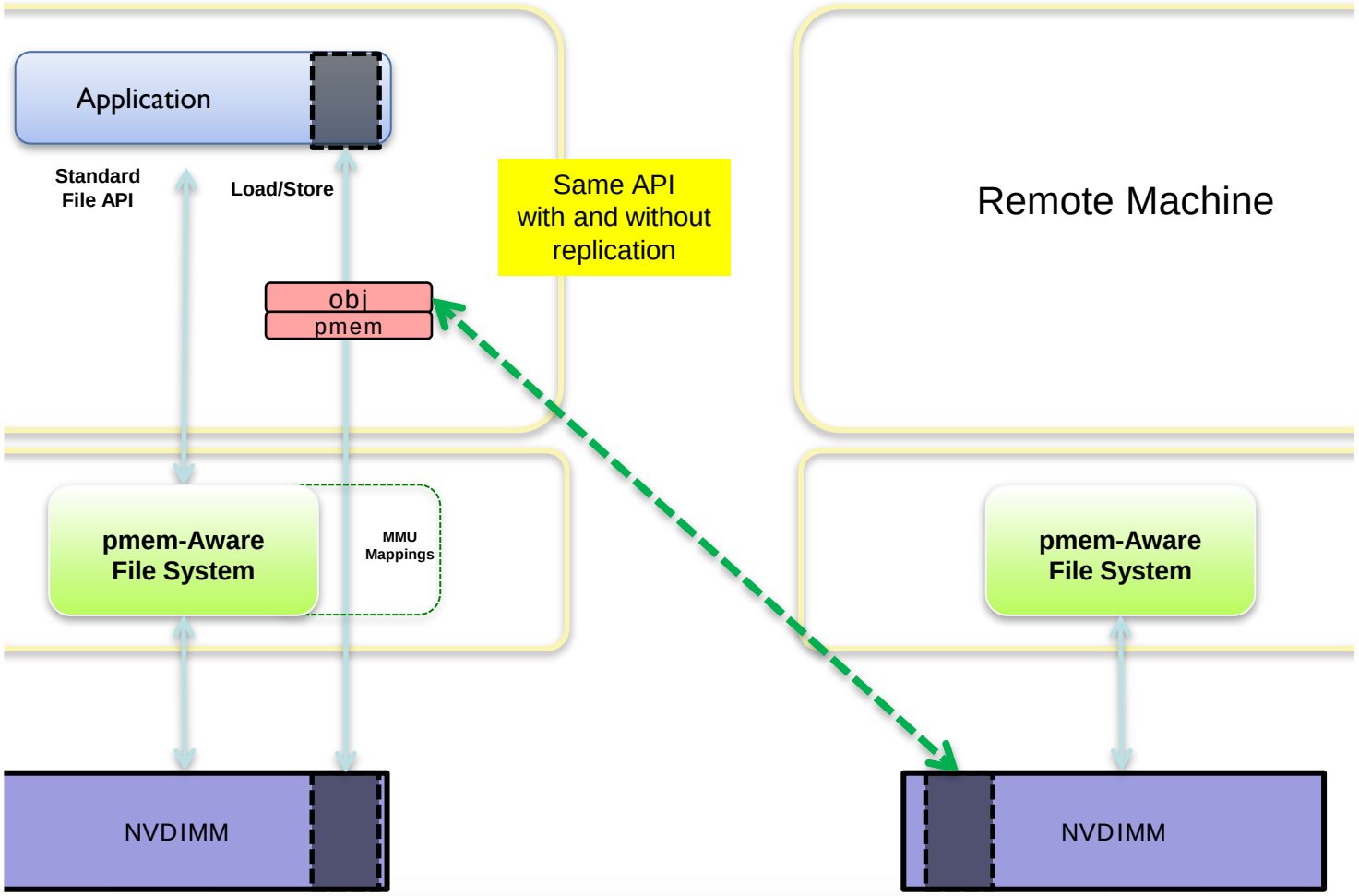
- General purpose pmem transactions
- pmem-aware memory allocator
- Some common operations made atomic
- Application doesn't worry about HW details
  - Library handles CPU cache flushing
  - Library comprehends size of atomic stores
- Lots of examples in the pmem.io source tree
- This is probably the library you want

# libpmemobj Data Path




- Application uses libpmemobj API
- Transactions entirely memory-centric user space code
- Much faster, but…
- App had to change

# Libpmemobj Replication: Application Transparent
## (except for performance overhead)



# What are libpmemlog & libpmemblk?

- Special purpose libraries
  - Append transactionally to a pmem log
  - Write fixed blocks to pmem transactionally
- You could use libpmemobj for this

- These libraries are optimized for specific usages

# What is libpmem?

- Low-level helper functions for basic pmem
  - Used by obj, log, blk libraries
  - Just basic flush, memcpy helpers
  - No transactions
- If you don't want any help from NVML, you probably still use libpmem just to avoid writing your own assembly-language routines
- Most likely, you want libpmemobj

# Volatile Usages of pmem

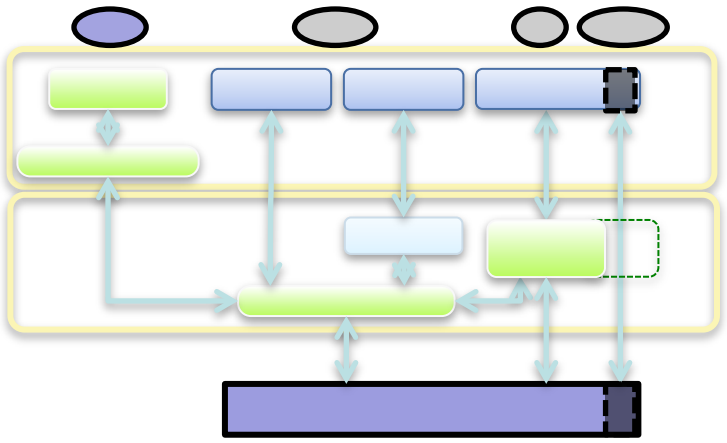
- libvmem: malloc/free interfaces
- libvmmalloc: transparent libvmem
- libmemkind: integrates NUMA, other “kinds”

- Commonly-understood volatile programming

- All volatile focus is on libmemkind at this point

# Where is the SNIA NVM Programming Model in all this?

- The SNIA NVM Programming Model simply says that persistent memory is accessed by mapping files.
  - That support is in the OS
- NVML is a collection of libraries
  - To make programming easier
- NVML may be used to make higher-level languages pmem-aware (examples: Java, Python)



- Using NVML is a convenience, not a requirement

# Emulating Persistent Memory

- The programming model builds on memory-mapped files
  - So development on memory-mapped files work fine
    - NVML will use msync() to flush to persistence
    - Non-optimal performance
  - Use any 64-bit Linux pretty much
- For benchmarking:
  - http://pmem.io/2016/02/22/pm-emulation.html
  - Distros like Fedora 24 are built with DAX/pmem
    - Avoid the kernel build!
    - Can also avoid 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.

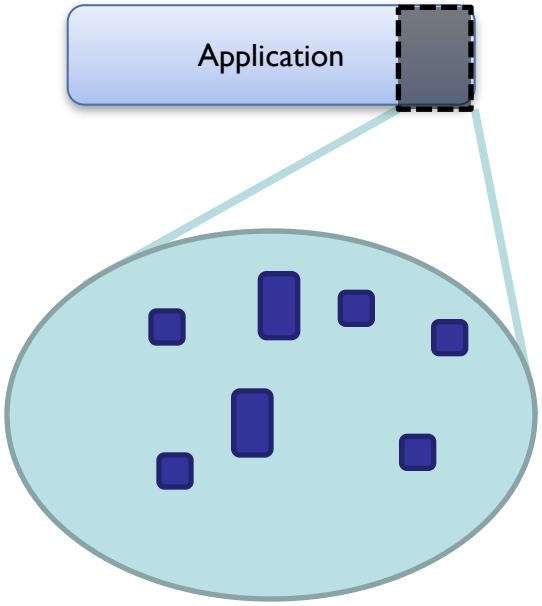
**pmempool-rm(1)**

Removes pool file or all pool files listed in poolset configuration file.

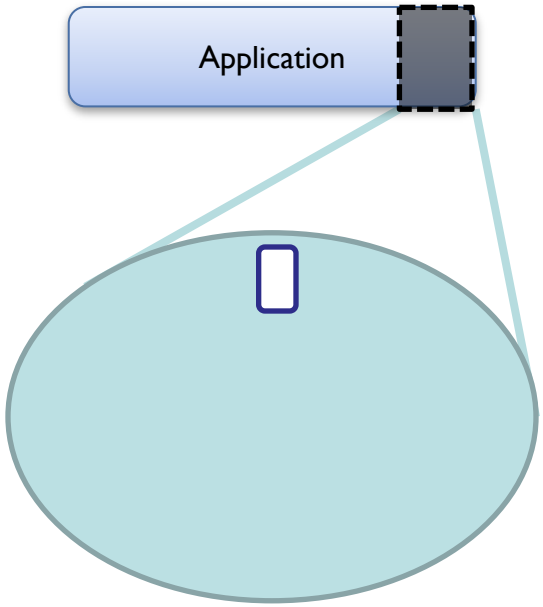
**pmempool-convert(1)**

Updates the pool to the latest available layout version.

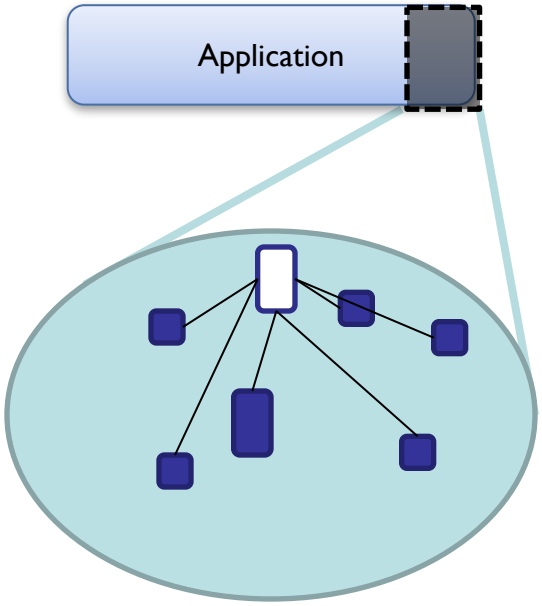
# Three Quick Start Examples




pmem-safe
memory allocation
(no transactions)

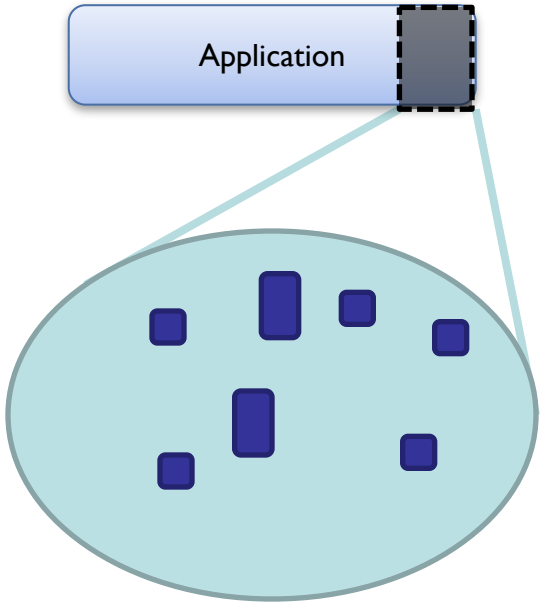



the *root object*




transactional
updates
(hash table)

# pmem-safe Memory Allocation




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

# pmem-safe Memory Allocator
## (snippets we will look at in pminvaders)

```
int pmemobj_alloc(PMEMobjpool *pop, PMEMoid *oidp,
size_t size, uint64_t type_num,
pmemobj_constr constructor, void *arg);
```

```
POBJ_LAYOUT_BEGIN(pminvaders);
POBJ_LAYOUT_ROOT(pminvaders, struct game_state);
POBJ_LAYOUT_TOID(pminvaders, struct player);
POBJ_LAYOUT_TOID(pminvaders, struct alien);
POBJ_LAYOUT_TOID(pminvaders, struct bullet);
POBJ_LAYOUT_END(pminvaders);
```

```
D_RW(plr)->x = dstx;
```

```
POBJ_NEW(pop, NULL, struct alien, create_alien, NULL);
```

```
POBJ_FOREACH(PMEMobjpool *pop, PMEMoid varoid)
```

# Simple (Typeless) Allocation

```
struct stuff {
        int x;
        int y;
        char s[1000];
};

PMEMoid stuffoid;

if (pmemobj_alloc(pop, &stuffoid, sizeof (struct stuff), 5, NULL, NULL) < 0)
        err(1, "pmemobj_alloc on %lu bytes", sizeof(struct stuff));

struct stuff *stuffp = (struct stuff *)pmemobj_direct(stuffoid);

stuffp->x = 100;
stuffp->y = 200;
strcpy(stuffp->s, "stuff!");

pmemobj_persist(pop, stuffp, sizeof (struct stuff));
```

pmem-safe, simple size-based allocation, but typeless!

# libpmemobj Constructor Functions

- Provided to allocation functions
- Function gets a chance to initialize allocation
  - Return 0 from constructor on success
  - Return non-zero to indicate error
    - Cancels the allocation

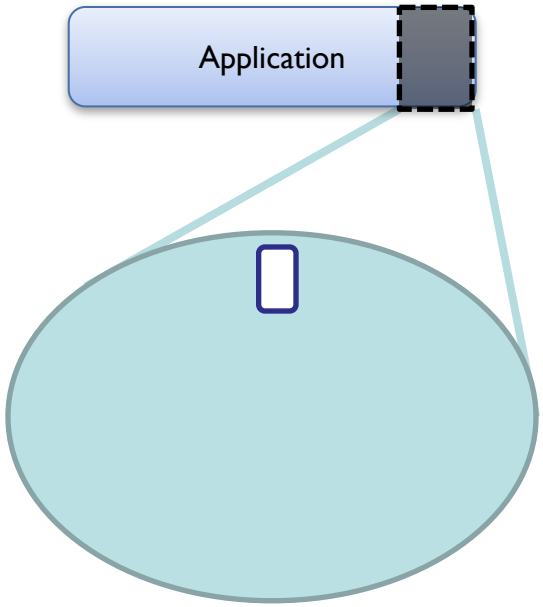
- All the above is power-fail safe!

# Finding your Allocations

- Allocations have a type number
- `POBJ_FOREACH`: iterate over allocations of a specific type
  - This allows a simplistic program to just allocate blobs and use the iterator to find them again
  - Example: Hash table in DRAM with values in pmem

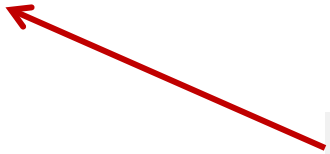
- Most programs don't want to have to iterate to find their pmem-resident data structures, they want to have a well-known *root object* and start from there

# The *root object*

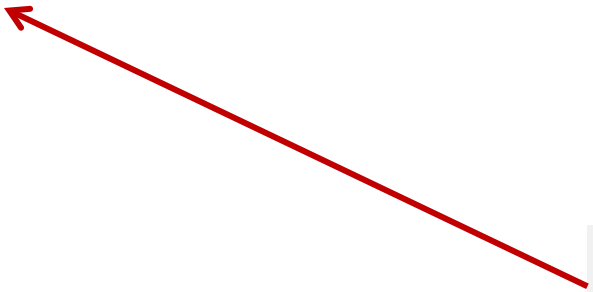


# layout.h

```
#define LAYOUT_NAME "hello example"
#define BUF_LEN 100

struct my_root {
        int len;
        char buf[BUF_LEN];
};
```



Your layout



Your root data structure

# Hello World Example

```
fedora24 $ ls
helloread.c  hellowrite.c  layout.h  Makefile
fedora24 $ make
cc -ggdb -Wall -Werror   -c -o hellowrite.o hellowrite.c
cc -o hellowrite hellowrite.o layout.h -lpmemobj -lpmem -pthread
cc -ggdb -Wall -Werror   -c -o helloread.o helloread.c
cc -o helloread helloread.o layout.h -lpmemobj -lpmem -pthread
```

https://github.com/andyrudoff/sdc16

# hellowrite.c

```
#include <err.h>
#include <libpmemobj.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "layout.h"

#define HELLOSTRING "Hello, pmem!"

int
main(int argc, char *argv[])
{
        if (argc != 2)
                errx(1, "usage: %s file-name", argv[0]);

        PMEMobjpool *pop = pmemobj_create(argv[1], LAYOUT_NAME,
                                PMEMOBJ_MIN_POOL, 0666);

        if (pop == NULL)
                err(1, "pmemobj_create");
```

# hellowrite.c

```
        PMEMoid root = pmemobj_root(pop, sizeof(struct my_root));
        struct my_root *rootp = pmemobj_direct(root);

        rootp->len = strlen(HELLOSTRING);
        pmemobj_persist(pop, &rootp->len, sizeof(rootp->len));

        pmemobj_memcpy_persist(pop, rootp->buf, HELLOSTRING, rootp->len);

        pmemobj_close(pop);

        exit(0);
}
```

```
fedora24 $ hellowrite myfile
```

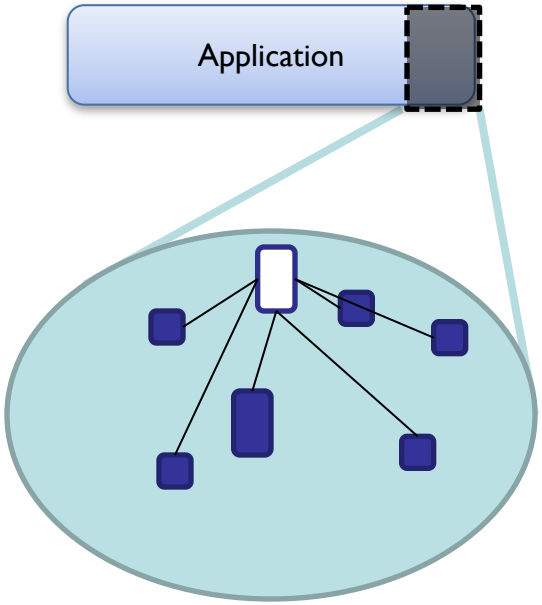
```
fedora24 $ helloread myfile
Hello, pmem!
```

# helloread.c

```
int
main(int argc, char *argv[])
{
        if (argc != 2)
                errx(1, "usage: %s file-name", argv[0]);

        PMEMobjpool *pop = pmemobj_open(argv[1], LAYOUT_NAME);
        if (pop == NULL)
                err(1, "pmemobj_open");

        PMEMoid root = pmemobj_root(pop, sizeof(struct my_root));
        struct my_root *rootp = pmemobj_direct(root);

        if (rootp->len == strlen(rootp->buf))
                printf("%s\n", rootp->buf);

        pmemobj_close(pop);

        exit(0);
}
```

```
fedora24 $ pmempool info myfile
POOL Header:
Signature               : PMEMOBJ
Major                   : 2
Mandatory features      : 0x0
Not mandatory features  : 0x0
Forced RO               : 0x0
Pool set UUID           : 37b2b0ab-3dcb-464e-b6c6-b5dd845502d5
UUID                    : 29ef8c08-20cf-4749-b7ef-462876ee57ee
Previous part UUID      : 29ef8c08-20cf-4749-b7ef-462876ee57ee
Next part UUID          : 29ef8c08-20cf-4749-b7ef-462876ee57ee
Previous replica UUID   : 29ef8c08-20cf-4749-b7ef-462876ee57ee
Next replica UUID       : 29ef8c08-20cf-4749-b7ef-462876ee57ee
Creation Time           : Thu Sep 15 2016 18:28:19
Alignment Descriptor    : 0x000007f737777310 [OK]
Class                   : ELF64
Data                    : 2's complement, little endian
Machine                 : AMD X86-64
Checksum                : 0xb50eedf1 [OK]

PMEM OBJ Header:
Layout                  : hello example
Lanes offset            : 0x2000
Number of lanes         : 1024
Heap offset             : 0x302000
Heap size               : 5234688
Checksum                : 0x51c2179d [OK]
Root offset             : 0x3c2180
```

# Transactional Updates




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

# Transactional Updates
## (snippets we will look at in hashmap)

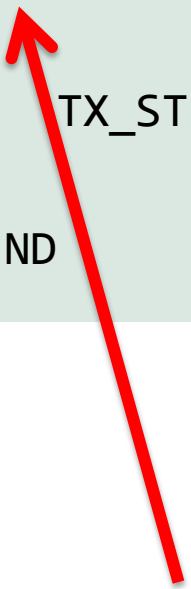
```
struct entry {
        uint64_t key;
        PMEMoid value;

        /* next entry list pointer */
        TOID(struct entry) next;
};

TX_BEGIN(pop) {
        TX_ADD(hashmap);

        D_RW(hashmap)->seed = seed;
          ...
        D_RW(hashmap)->buckets = TX_ZALLOC(struct buckets, sz);
        D_RW(D_RW(hashmap)->buckets)->nbuckets = len;
} TX_ONABORT {
          ...
} TX_END
```

# Multi-Threaded Locking

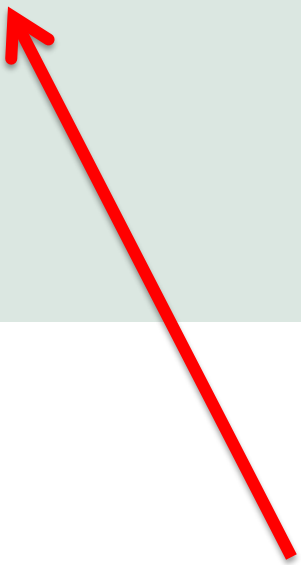
```
TX_BEGIN_LOCK(pop, TX_LOCK_MUTEX, &op->mylock, TX_LOCK_NONE) {

        TX_STRCPY(op->name, newname);

} TX_END
```

# Two Types of Atomicity

```
TX_BEGIN_LOCK(pop, TX_LOCK_MUTEX, &op->mylock, TX_LOCK_NONE) {

        TX_STRCPY(op->name, newname);

} TX_END
```



Powerfail Atomicity



Multi-Thread Atomicity

# Summary

- NVML is for convenience
  - Not required, but time saving
  - Actively validated
- C APIs ready for us, C++ ready for early use
- Lots of examples in NVML source tree
  - And use the man pages & blogs!
- New libraries ongoing
- New language support ongoing
  - Python prototyped, Java underway
- http://pmem.io, Google group: `pmem`