Persistent Memory Support in Red Hat Enterprise Linux

Jeff Moyer, Red Hat, Inc.
Andy Rudoff, Intel

June 30, 2016

Agenda

- A Bit of Background Information
- Software Architecture
- pmem Configuration and Management
- pmem Advantages/Challenges
- pmem Examples

Background

Persistent Memory

- Order-of-magnitude DRAM Performance
- Byte-addressable
- Persistent
- DMA Target
- High capacity

- Use Cases:
  - Rapid start-up (data set already in memory)
  - Random, odd-shaped accesses (avoid transferring blocks)
  - Fast write-cache

Flavors of NVDIMMs

- NVDIMM-N
  - Energy-backed DRAM
  - Flash used for persistence
    (not exposed to OS)
  - Performance on par with DRAM
  - Small Capacity
  - Expensive

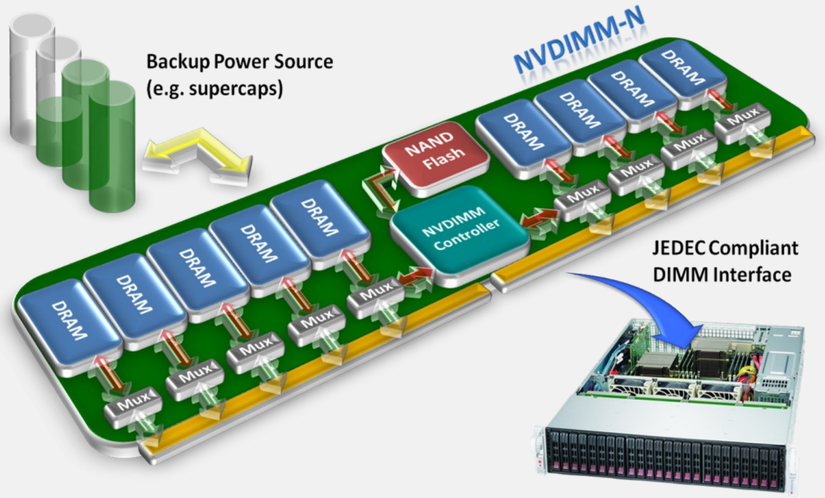


Image Source: SNIA_NVDIMM

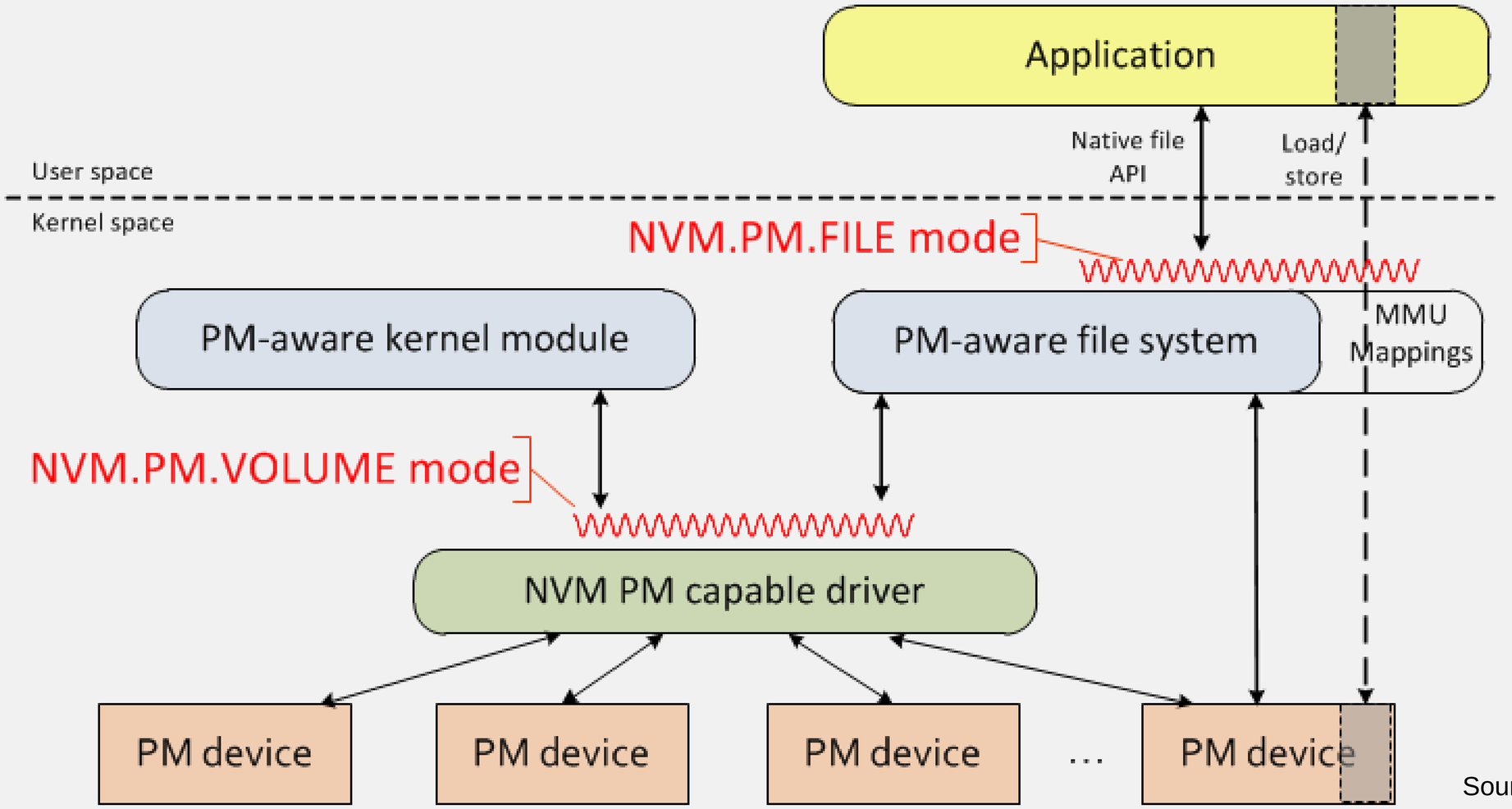
- NVDIMM-P
  - Same order of magnitude performance as DRAM (read: may be slightly slower)
  - Much larger capacity
  - Cheaper (?)

Software Architecture

NVM Programming Model

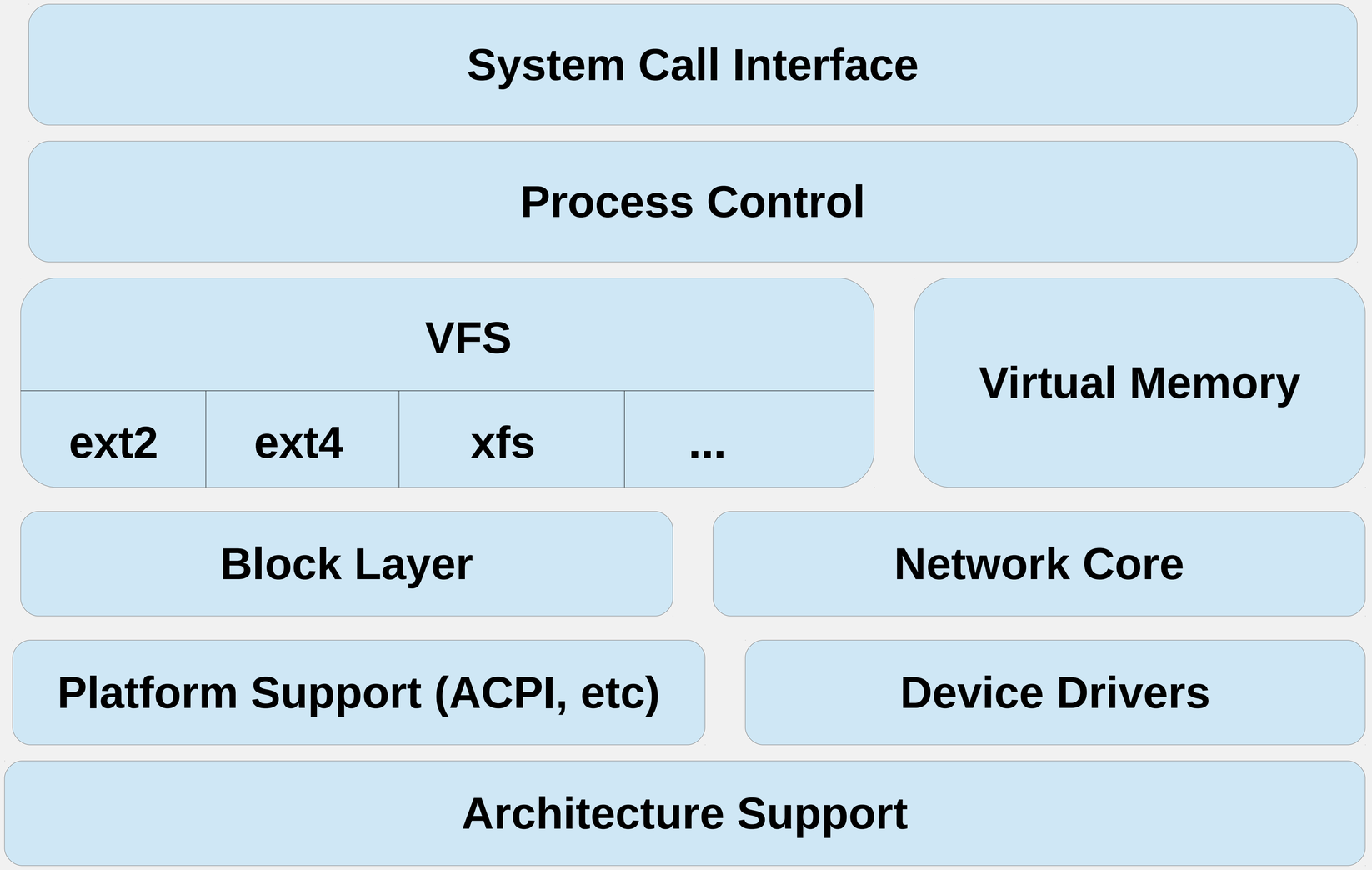
http://snia.org/sites/default/files/NVMProgrammingModel_v1.pdf

NVM.PM Modes

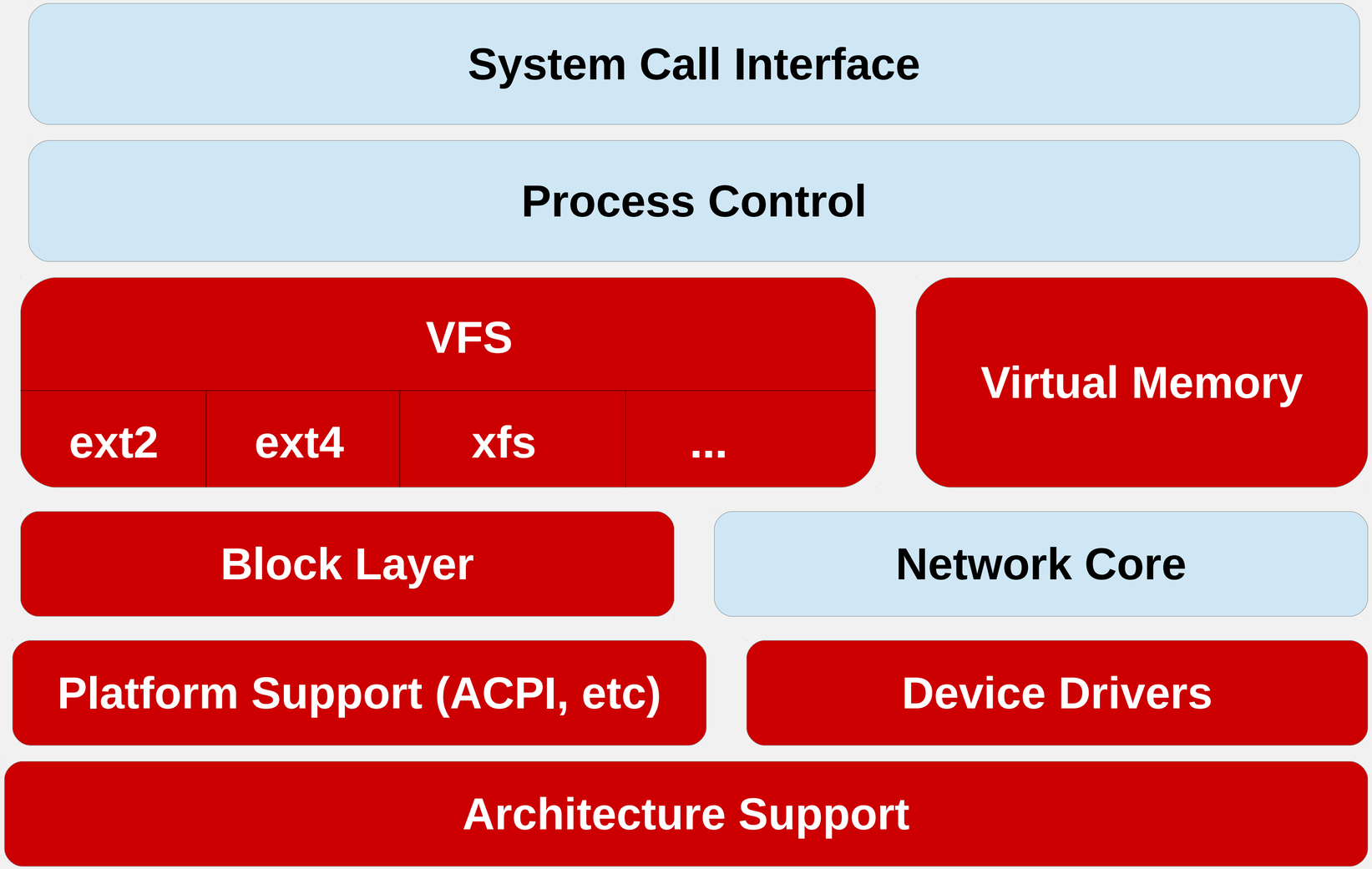


Source: ProgModel

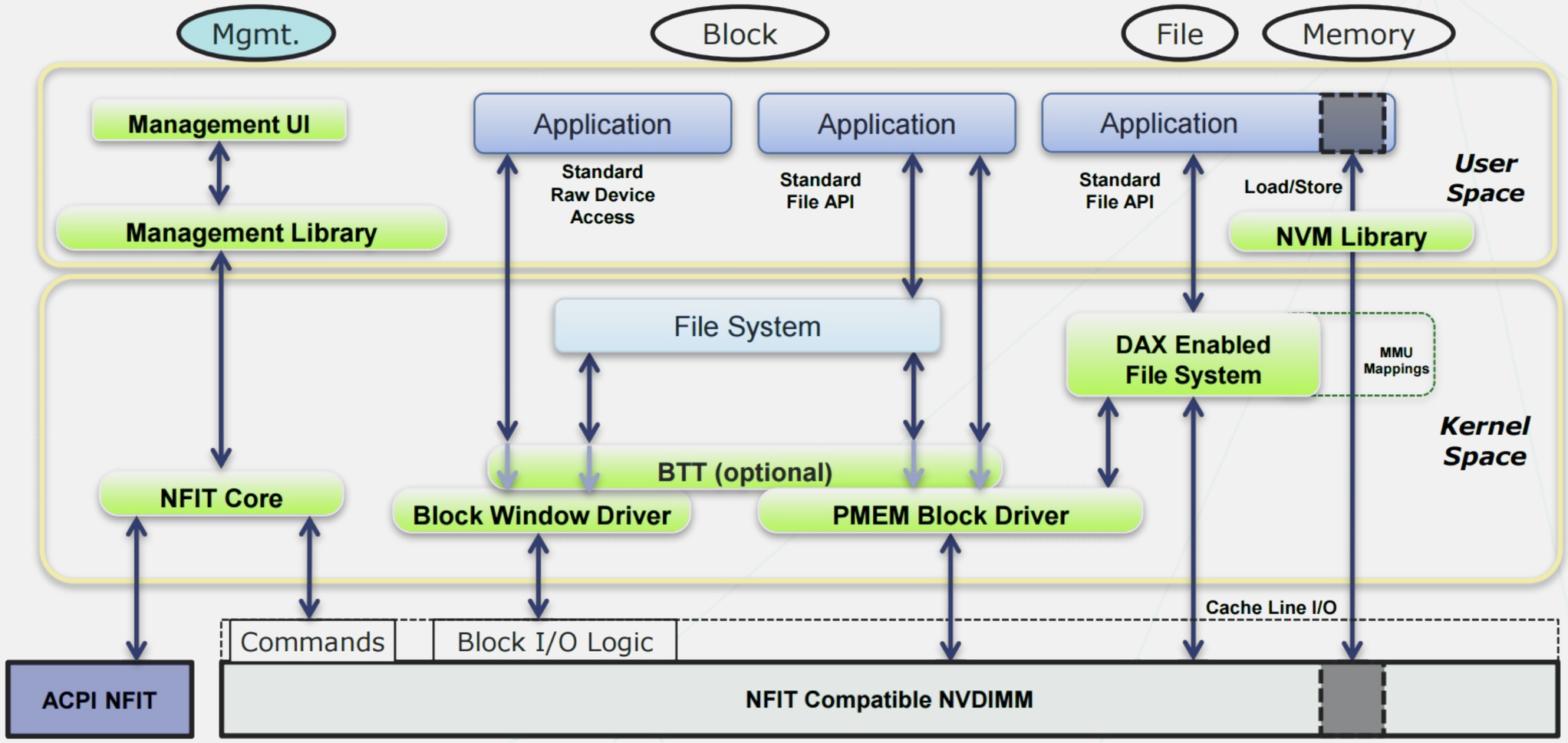
Major Kernel Subsystems



Modified Kernel Subsystems



Software Architecture



Source: Namespace

pmem Configuration and Management

PMEM Namespace Configurations

RAW	SECTOR	MEMORY

- Default, but don't use it!

PMEM Namespace Configurations

RAW	SECTOR	MEMORY
• Default, but don't use it!	• Atomic Sector Updates (provided by the btt) • Configurable Sector Size (includes DIF/DIX) • Applicable to both PMEM and BLK namespaces	

PMEM Namespace Configurations

RAW

- Default, but don't use it!

SECTOR

- Atomic Sector Updates
  (provided by the btt)
- Configurable Sector Size
  (includes DIF/DIX)
- Applicable to both PMEM and BLK namespaces

MEMORY

- DAX Support
- Applies only to PMEM namespaces
- Requires space for kernel data structures

“Memory” Namespaces

- Need to reserve space for kernel page structures
- 2 options:
  1) Eat up DRAM
  2) Lose storage space

64 bytes per 4K page = **16GB/TB**

32GB DIMM = **512 MB**

Configuring DAX

```
# ndctl list
[
  {
    "dev":"namespace0.0",
    "mode":"raw",
    "size":17179869184,
    "blockdev":"pmem0"
  }
]
# fdisk -l /dev/pmem0

Disk /dev/pmem0: 17.2 GB, 17179869184 bytes, 33554432 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
```

Configuring DAX
using DRAM to host struct pages

```
# ndctl create-namespace -f -e namespace0.0 --mode=memory --map=mem
{
  "dev":"namespace0.0",
  "mode":"memory",
  "size":17177772032,
  "uuid":"3c88e67f-8b25-4661-adf9-f0ed390cbd6a",
  "blockdev":"pmem0"
}

# fdisk -l /dev/pmem0

Disk /dev/pmem0: 17.2 GB, 17177772032 bytes, 33550336 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
```

Configuring DAX
using DRAM to host struct pages

```
# ndctl create-namespace -f -e namespace0.0 --mode=memory --map=mem
{
  "dev":"namespace0.0",
  "mode":"memory",
  "size":17177772032,
  "uuid":"3c88e67f-8b25-4661-adf9-f0ed390cbd6a",
  "blockdev":"pmem0"
}

# fdisk -l /dev/pmem0

Disk /dev/pmem0: 17.2 GB, 17177772032 bytes, 33550336 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
```

2MB Shy of 16GB

Configuring DAX
using the NVDIMM to host struct pages

```
# ndctl create-namespace -f -e namespace0.0 --mode=memory --map=dev
{
  "dev":"namespace0.0",
  "mode":"memory",
  "size":16909336576,
  "uuid":"b5c852b2-75c2-4e8b-94b2-06694d6ff243",
  "blockdev":"pmem0"
}

# fdisk -l /dev/pmem0

Disk /dev/pmem0: 17.2 GB, 17177772032 bytes, 33550336 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
```

Configuring a BTT Namespace

```
# ndctl list
[
  {
    "dev":"namespace0.0",
    "mode":"raw",
    "size":17179869184,
    "blockdev":"pmem0"
  }
]
```

Configuring a BTT Namespace

```
# ndctl create-namespace -f -e namespace0.0 -m sector
{
  "dev":"namespace0.0",
  "mode":"sector",
  "uuid":"9e24b27a-bb46-44ad-b7fb-81ebfee0a3d6",
  "sector_size":4096,
  "blockdev":"pmem0s"
}

# fdisk -l /dev/pmem0s

Disk /dev/pmem0s: 17.2 GB, 17162027008 bytes, 4189948 sectors
Units = sectors of 1 * 4096 = 4096 bytes
Sector size (logical/physical): 4096 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
```

File System Setup for DAX

```
# mkfs -t xfs -d su=1g,sw=1 /dev/pmem0
# mount -t xfs -o dax /dev/pmem0 /mnt/dax
```

```
# mkfs -t ext4 /dev/pmem0
# mount -t ext4 -o dax /dev/pmem0 /mnt/dax
```

NOTE: Inconsistent Behavior:

- Ext4 fails if DAX unavailable
- Xfs logs a message

pmem Advantages/Challenges

pmem Challenges

- Non-transparent usage means application changes
  - App must decide what data lives in each tier
  - Any app change is impactful
- Do volatile memory algorithms “just work”?
  - Sure, for volatile use cases
  - Algorithms for persistence are different

- Primary challenge: decide where to spend effort

pmem Examples

Programming Model Summary

- pmem exposed as memory-mapped files
  - Always safe to use standard API: `msync()`
- Only when Linux says it is safe:
  - *Optimized flush* from user space
    - `CLFLUSH` or `CLFLUSHOPT`+fence or `CLWB`+fence or NT store+fence
  - libpmem's `pmem_is_pmem()` function tells you if it is safe
- Only when Linux says platform supports it (future use):
  - CPU caches are part of persistence domain
  - libpmem's `pmem_persist()` will handle this
- Standard API may flush to smaller failure domain than optimized flush

POSIX Load/Store Persistence

```
open(…);
pmem = mmap(…);

strcpy(pmem, "hello");

msync(pmem, 6, MS_SYNC);
```

pmem Programming Model Load/Store Persistence

```
open(…);
pmem = mmap(…);

assert(pmem_is_pmem(pmem, len));

strcpy(pmem, "hello");

pmem_persist(pmem, 6);
```

Storing More Than 8 Aligned Bytes

```
open(…);
pmem = mmap(…);

assert(pmem_is_pmem(pmem, len));

strcpy(pmem, "hello there");

pmem_persist(pmem, 12);
```

crash

```
“\0\0\0\0\0\0\0\0\0\0\0\0”

“hello the\0\0\0\0”

“\0\0\0\0\0\0\0\0ere\0”

“hello there\0”
```

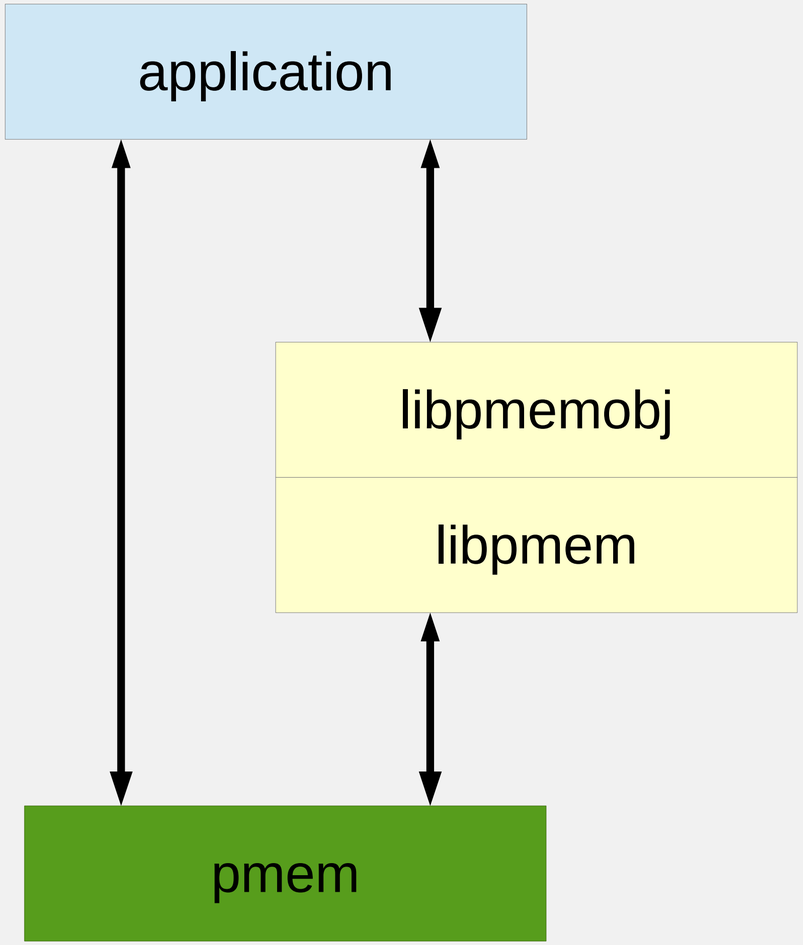
Visibility versus Powerfail Atomicity

Feature	Atomicity
Atomic Store	8 byte powerfail atomicity Much larger visibility atomicity
TSX	Programmer must comprehend that `XABORT`, cache flush can abort
`LOCK CMPXCHG`	*non-blocking* algorithms depend on CAS, but CAS doesn't include flush to persistence

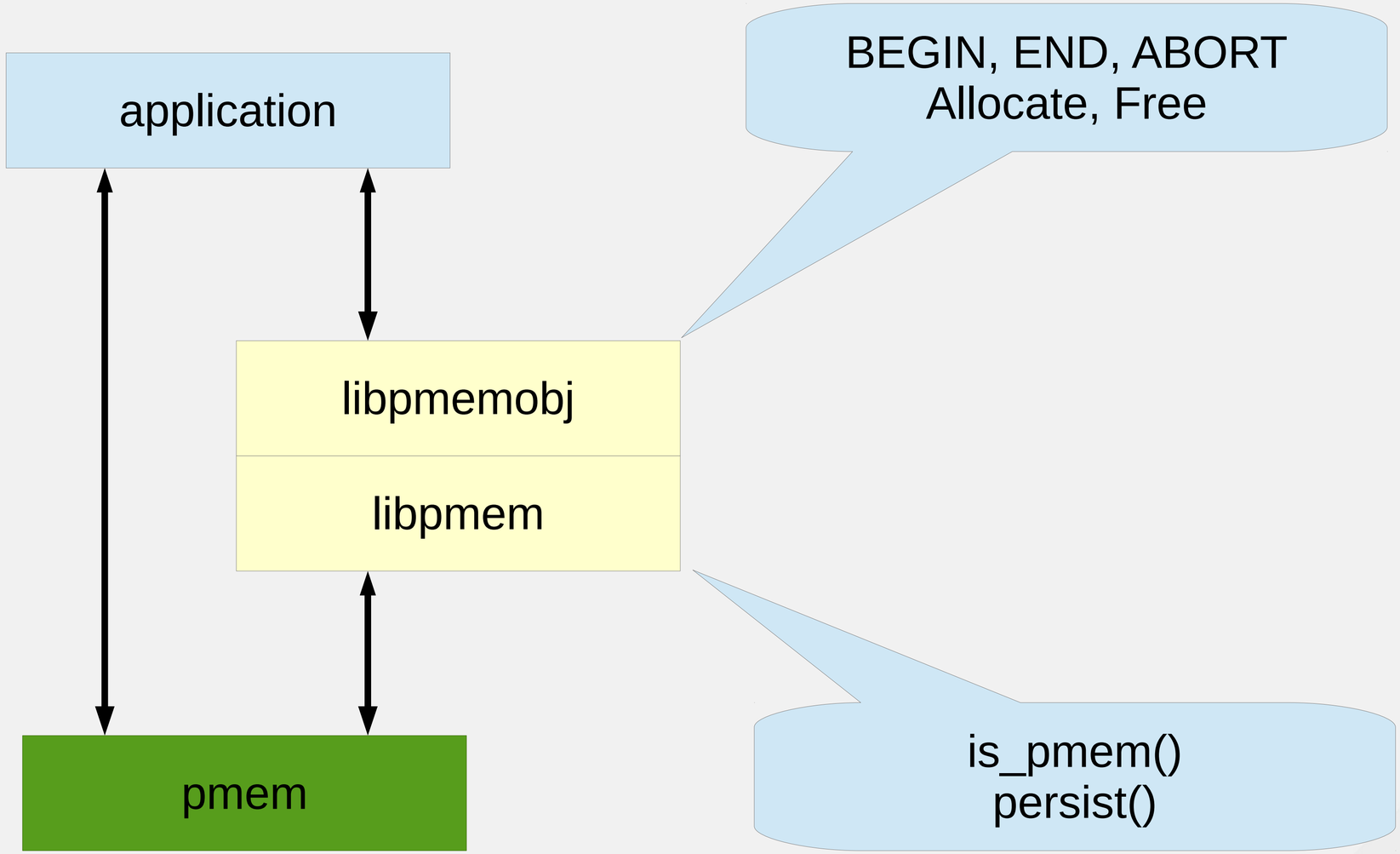
NVM Libraries

- Transactions
  - Hardest part to get right, still non-trivial to use in library
- Persistent Memory Allocation
  - Always-consistent heap (no persistent memory leaks)
- Common Set of Atomic Operations
  - Lists, Allocation onto/off of lists
- Replication
  - Local active/passive now
  - Remote active/passive next
  - More flexible later
- More transparent usages supported over time

Transactional Object Store



Transactional Object Store



Simple pmemobj Transaction

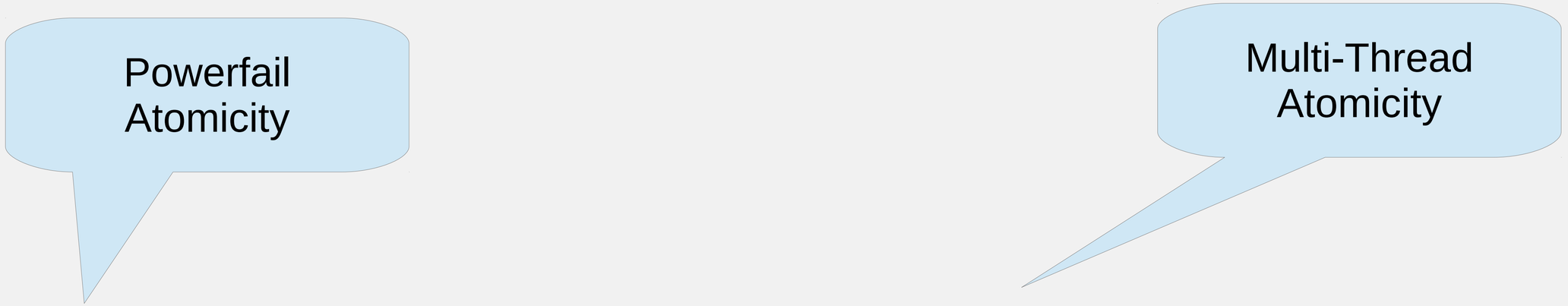
```
struct myobj {
  PMEMmutex mylock;
  char greeting[GREETINGLEN];
};
```

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

  TX_STRCPY(op->greeting, “hello there”);

} TX_END
```

Two Types of Atomicity

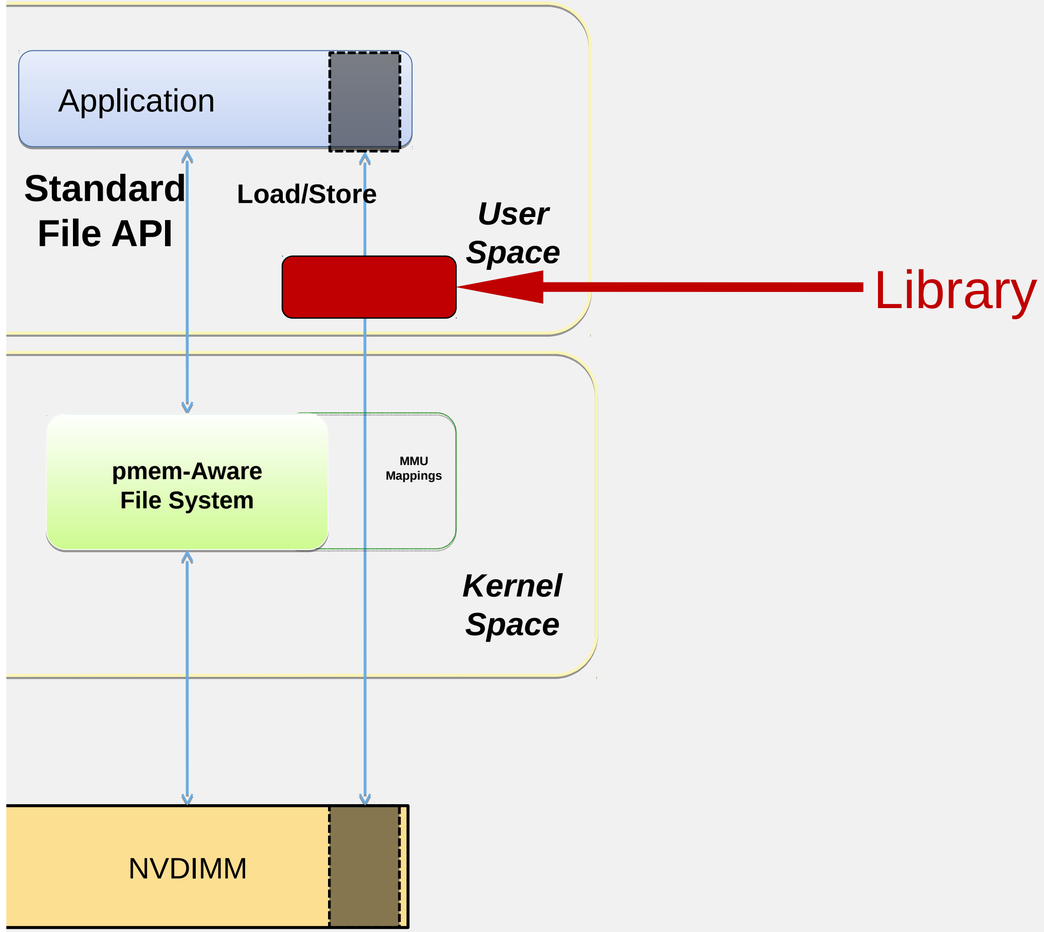



```
TX_BEGIN_LOCK(pop, TX_LOCK_MUTEX, &op->mylock) {

  TX_STRCPY(op->greeting, “hello there”);

} TX_END
```

NVM Library: pmem.io



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

Transactional

Summary

- Persistent Memory products available today
  - Capacities about to explode
- Linux is prepared
  - pmem driver stack, DAX, ext4, xfs, etc.
- RHEL is prepared
  - ndctl & other tools, validation
- Potential value of pmem programming is quite large
  - Applications re-organize data into memory, storage, and pmem
- Numerous challenges
  - NVM Libraries provide some solutions that applications can leverage

References

- ProgModel - http://www.snia.org/tech_activities/standards/curr_standards/npm
- Namespace - http://pmem.io/documents/NVDIMM_Namespace_Spec.pdf
- SNIA_NVDIMM - http://www.snia.org/forums/sssi/NVDIMM
- Williams_Vault – http://events.linuxfoundation.org/sites/events/files/slides/Managing%20Persistent%20Memory_0.pdf
- WIKI – https://nvdimm.wiki.kernel.org/