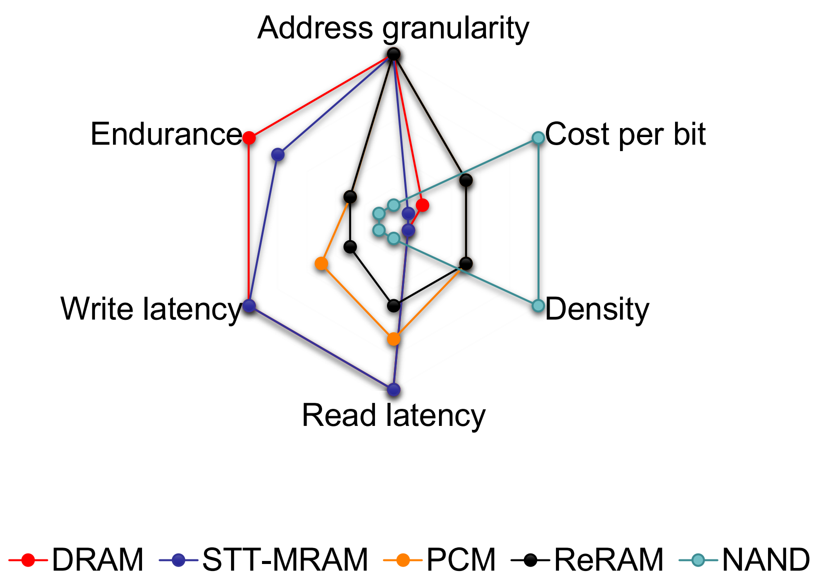
Accessing NVM Locally and over RDMA Challenges and Opportunities

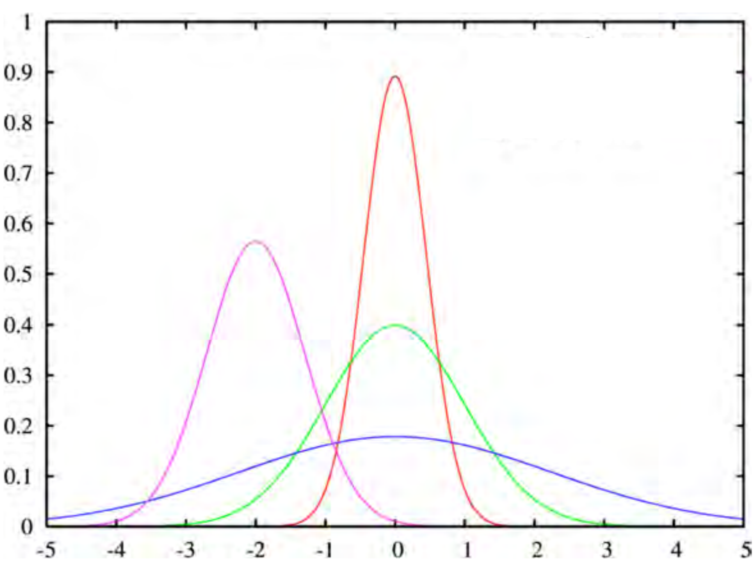
Wendy Elsasser
Megan Grodowitz
William Wang

MSST - May 2018

Emerging NVM

A wide variety of technologies with varied characteristics



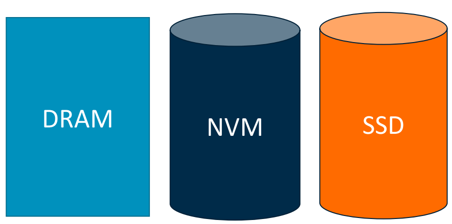


Variable latency and tail distributions

Multiple system use-cases with unique challenges

Faster Storage

1000x faster than NAND



Storage
- Filesystem bottlenecks

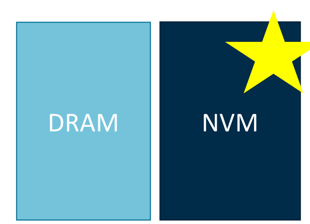
Denser Mem

10x denser than DRAM

Transformative Capacity/TCO-advantage
- Endurance
- Bandwidth
- Caching

Persistent Mem

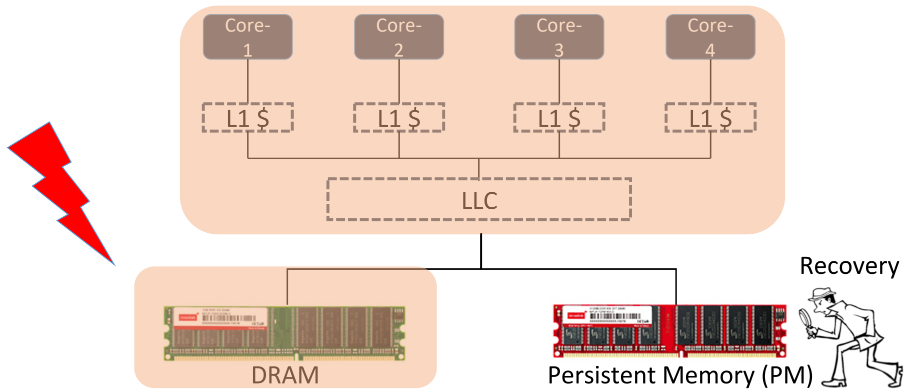
Non-Volatile



Persistency
- Ordering
- Point of Persistence

What about persistence?

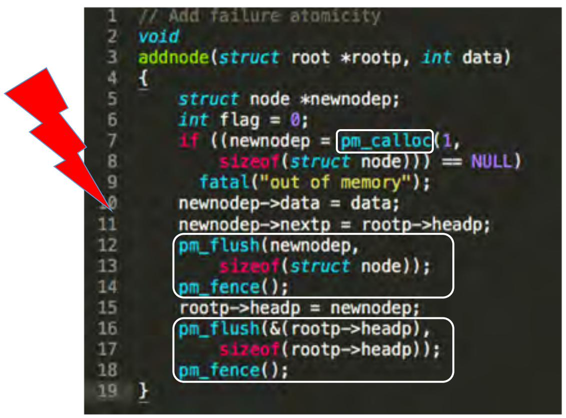
Managing ordering requirements



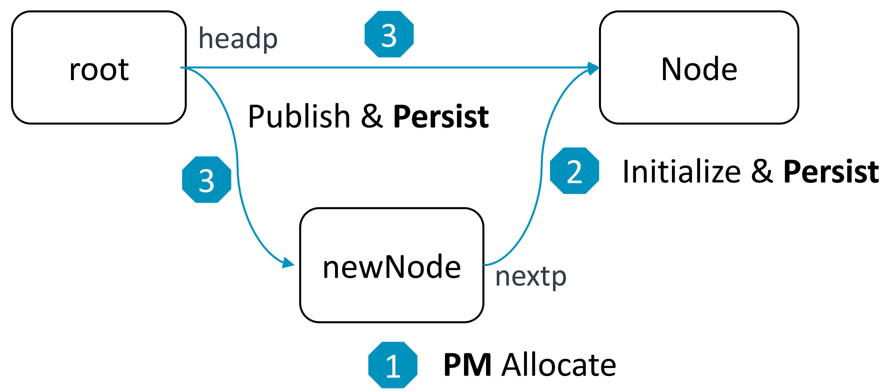
Recovery can inspect the data-structures in PM to restore system to a consistent state

- Crash consistency (failure atomicity) is needed to ensure recovery can restore system to a consistent state
  - Data move through volatile memories before they get written to PM
  - Using CPU cache flushes and fence instructions
- Direct connect PMEM protocols (NVDIMM) include explicit FLUSH semantics

Example: Add a node to a linked list with PMEM



```
// Add failure atomicity
void
addnode(struct root *rootp, int data)
{
    struct node *newnodep;
    int flag = 0;
    if ((newnodep = pm_calloc(1,
        sizeof(struct node))) == NULL)
      fatal("out of memory");
    newnodep->data = data;
    newnodep->nextp = rootp->headp;
    pm_flush(newnodep,
        sizeof(struct node));
    pm_fence();
    rootp->headp = newnodep;
    pm_flush(&(rootp->headp),
        sizeof(rootp->headp));
    pm_fence();
}
```



Persistent Memory Programming Models

Native Persistence

```
pt->x = 1;
pt->y = 1;
dccvap(&pt->x)
dccvap(&pt->y)
dsb

flag=1;
dccvap(&flag)
dsb
```

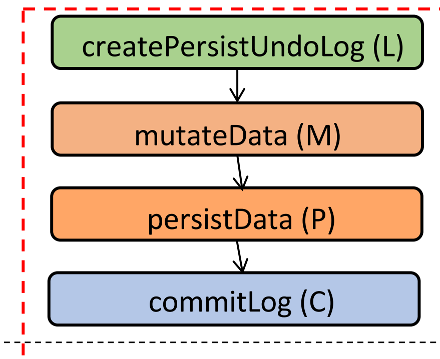
Library Persistence – Atomic

```
pt->x = 1;
pt->y = 1;
pmem_persist(&pt,
sizeof(pt))

flag = 1;
pmem_persist(&flag,
sizeof(flag))
```

Library Persistence – Durable TXs

```
TX_BEGIN{
pt->x = 1;
pt->y = 1;
} TX_END
```

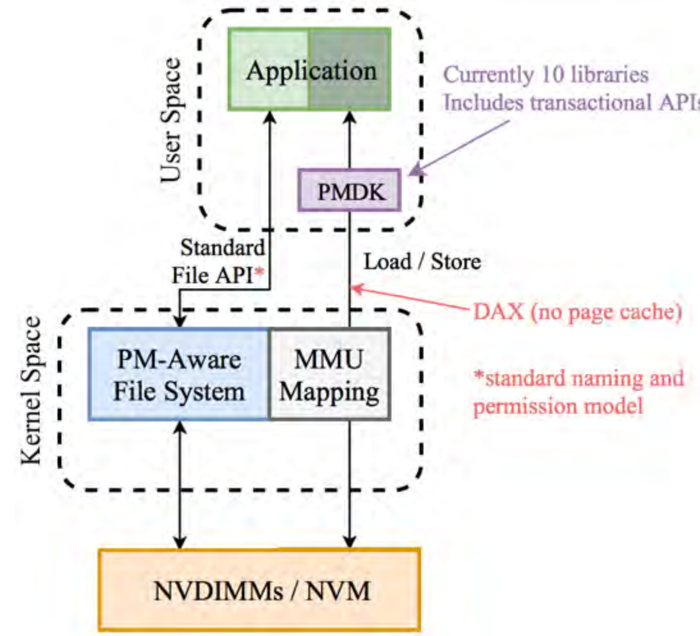


Programming simpler, overhead higher

PMDK (Persistent Memory Development Kit)

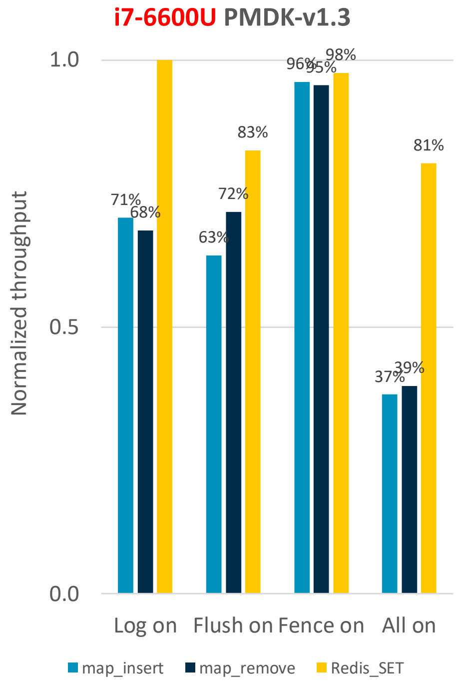
Formally NVML, ‘pmem libraries’

- PMDK provides transactional APIs for persistent memory programming
  - libpmemobj transactional APIs
  - Use fine-grained logging and cache flushes
- Works on 64-bit Linux, Windows and 64-bit FreeBSD



Ref: pmem.io

Flushing, logging and fencing overheads



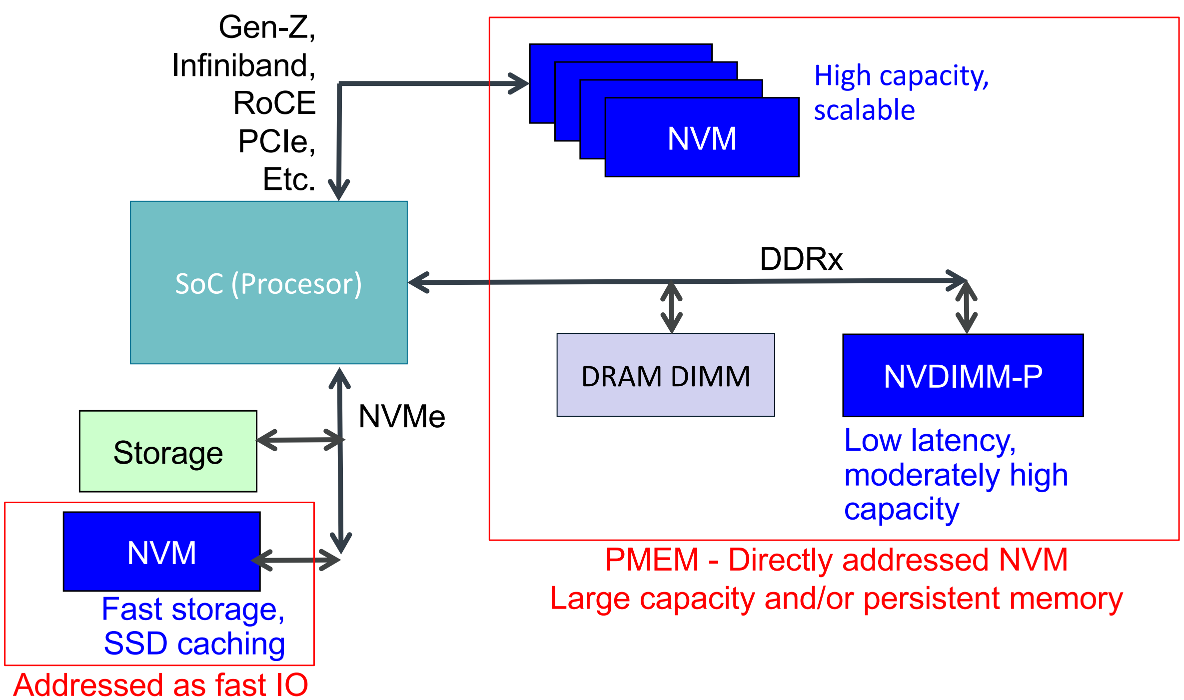
- Moving NVM from storage to local, byte addressable memory greatly improves performance
- But... overheads still exist to maintain a point of persistency. Can be minimized with:
  - Architectural optimizations
  - Software optimizations
  - Hardware acceleration

Baseline: PMDK without flushing/fencing and logging on

- Workloads: Map insert/remove, Redis Set. Implemented with NVML v1.3 libpmemobj transactions
- Platform: Intel i7-6600U with CLFLUSHOPT, single node with local DRAM

Fully incorporating NVM into your system

Numerous attachment points for the varied use cases



- Local and remote / distributed NVM both of interest
- New interfaces take advantage of byte addressable NVM
- How can we leverage RDMA for PMEM?

arm

Remote Direct Memory Access

What? Direct access to memory on a remote system without OS involvement

How? Zero-copy networking; read/write from main memory with network adaptor

Why? Lower latency, higher bandwidth communication between distributed processes

When? Late 90's: "Virtual Interface Architecture" tried to standardize zero-copy networking
Mid-late 00's: First Infiniband implementations stable and mature.
Today (2018): Still be described as a "new technology"

Who / Where? Well, supercomputers, but also…

- Chelsio Terminator 5 & 6 iWARP adapters
- GlusterFS internetwork filesystem
- Intel Xeon Scalable processors and Platform Controller Hub
- Mellanox ConnectX family of network adapters and InfiniBand switches
- Microsoft Windows Server (2012 and higher) via SMB Direct supports RDMA-capable network adapters, Hyper-V virtual switch and the Cognitive Toolkit.
- Apache Hadoop and Apache Spark big data analysis
- Baidu Paddle (PArallel Distributed Deep LEarning) platform
- Broadcom and Emulex adapters
- Caffe deep learning framework
- Cavium FastLinQ 45000/41000 Series Ethernet NICs
- Ceph object storage platform
- ChainerMN Python-based deep learning open source framework
- Nutanix's upcoming NX-9030 NVM Express flash appliance is said to support RDMA.
- Nvidia DGX-1 deep learning appliance
- Oracle Solaris 11 and higher for NFS over RDMA
- TensorFlow open source software library for machine intelligence
- Torch scientific computing framework
- VMware ESXi

RDMA programming

Often abstracted underneath some other library layer

- MPI and other HPC communication libraries
- Lustre, NFS_RDMA and other I/O libraries
- SDP, rsockets, or other socket type interface

Explicit programming of RDMA uses Verbs

- Verbs is not actually an API, but is instead a functional description of RDMA
- libibverbs is the standard Linux verbs implementation API
- APIs for verbs register *byte array contiguous memory* regions to make them available for remote access

Same API for all RDMA enabled networks

- Infiniband
- RDMA Over Converged Ethernet (RoCE)
- Internet Wide Area RDMA Protocol (iWARP)

NVM API's could leverage old ideas

- E.g. Memory mapped files
- Add a couple of more things like
  - Allocation, Flush
- Great for adaption but must also ensure functionality and performance with new features and limitations

RDMA, PMEM, and filesystems – current state

Block device APIs already support concepts like flushing and persistence

- E.g. fflush() an IO stream means the data will "be there" after power outage ← Fundamental NVM value
  - Data persistence

Linux PMEM drivers are available for NVDIMM (byte addressable) support

- Byte level access with DAX to bypass the page cache and get memory like speeds
- Three device modes for NVDIMM namespaces include:
  - Memory mode: DAX byte level access + DMA support ← Fundamental PMEM value
    - Byte Addressable NVM

But there is a small problem

- With direct PMEM access, pinned RDMA pages may be corrupted when the file is truncated
- Patch is available (*https://patchwork.kernel.org/patch/10028887/)

Where can we go from here?

Emerging NVM is creating opportunities to redefine the memory sub-system

Will still have slow, cheap storage, but will have fast, distributed PMEM in front of it

FLUSH capability required for persistency across power-fail events

- Linux PMEM drivers currently available and NVDIMM-P natively supports FLUSH capabilities

Optimizations possible to reduce overheads for persistency

Must also ensure persistent capabilities work with RDMA

- Let’s start with a bottom-up approach, leveraging existing technologies and developing new APIs

Incorporate into distributed applications (work-flow model) to gain performance benefits

- Data sharing and synchronization in PMEM

Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!

감사합니다

धन्यवाद