PMDK: THE STATE OF THE PROJECT

Andy Rudoff (Intel Data Center Group)

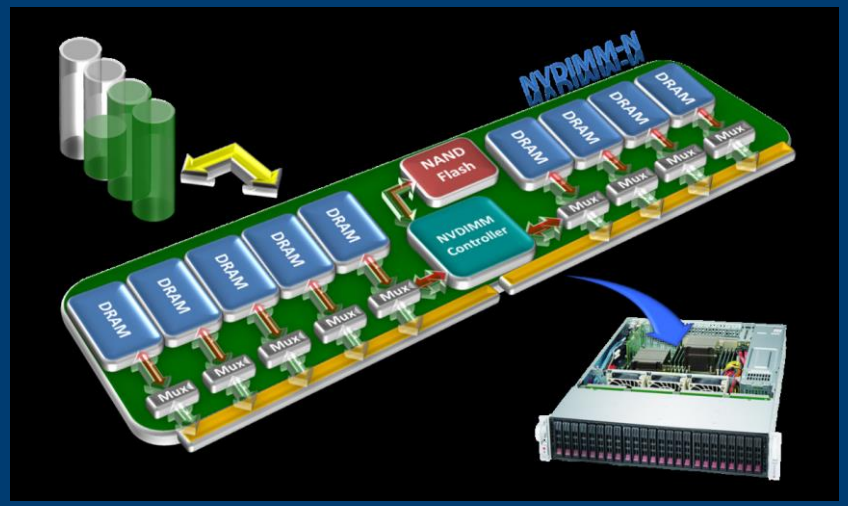
September 5th, 2019

AGENDA

- Goals
- History
- Current State of PMDK
- Future

GOALS OF PMDK

BACK WHEN WE HEARD: “PERSISTENT MEMORY IS COMING…”

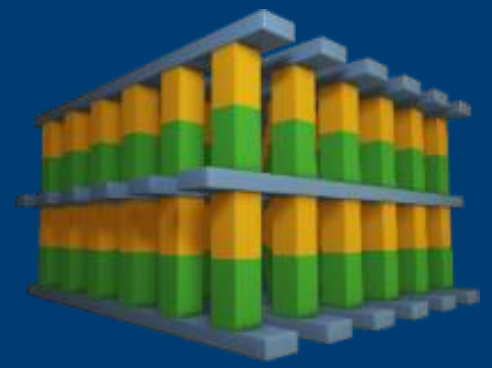


Byte-addressable, use it like memory

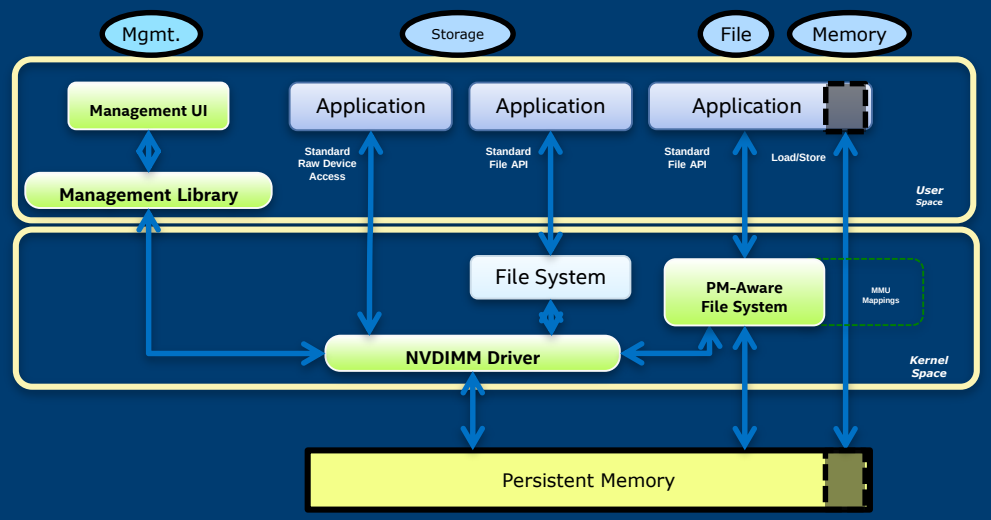
- But it is persistent

Actually had been shipping from some vendors

- Later named NVDIMM-N
- Small capacity 16-32 GB
- All access was through a driver interface when I first started looking at them



PERSISTENT MEMORY FIRST STEPS…



Step 1: how should it be exposed to applications

- How to name it, re-attach to it
- How to enforce permissions
- How to back it up, manage it
- And some less technical goals, but just as important
  - Represent the interests of the ISVs
  - Avoid vendor lock-in to a product-specific API
  - As an Intel employee, acknowledge that Intel-specific doesn't work here

Headed to SNIA…

ANCIENT HISTORY

June 2012

- Formed the NVM Programming TWG
- Immediate participation from key OSVs, ISVs, IHVs

January 2013

- Held the first PM Summit (actually called “NVM Summit”)

July 2013

- Created first GitHub thought experiments (“linux-examples”)

January 2014

- TWG published rev 1.0 of the NVM Programming Model

SNIA MODEL SUCCESS… AND THEN WHAT?!

Open a pmem file on a pmem-aware file system

Map it into your address space

Okay, you've got a pointer to 3TB of memory, have fun!

- The model is necessary, but not sufficient for an easy to program resource

Gathering requirements yielded fairly obvious top priorities:

- Need a way to track pmem allocations (like malloc/free, but pmem-aware)
- Need a way to make transactional updates
- Need a library of pmem-aware containers: lists, queues, etc.
- Need to make pmem programming not so error-prone

THE FIRST FEW TRIES

```
// volatile
char *ptr = malloc(size);

// persistent
char *ptr = pm_malloc(size);

// crash before using ptr => pmem leak!
```

```
NAME
        libpmemalloc -- Persistent Memory malloc-like library

SYNOPSIS
        #include <pmemalloc.h>
        cc ... -lpmemalloc

        void *pmemalloc_init(const char *path, size_t size);
        void *pmemalloc_static_area(void *pmp);
        void *pmemalloc_reserve(void *pmp, size_t size);
        void pmemalloc_persist(void *pmp, void **parentp_,
                               void *ptr_);
        void pmemalloc_onactive(void *pmp, void *ptr_,
                               void **parentp_, void *nptr_);
        void pmemalloc_onfree(void *pmp, void *ptr_,
                               void **parentp_, void *nptr_);
        void pmemalloc_activate(void *pmp, void *ptr_);
        void pmemalloc_free(void *pmp, void *ptr_);
        void pmemalloc_check(const char *path);

        PMEM(pmp, ptr_)
```

GOALS

Make persistent programming easier

- Especially allocation, transactions, atomic operations

Validate thoroughly to save developers implementation time

Performance tune it, improving over time

Later we realized we needed additional goals…

- Help simplify RAS (bad block tracking, recovery)
- Create new libraries for new use cases as they come up
- Track new hardware features (example: MOVDIR64B)

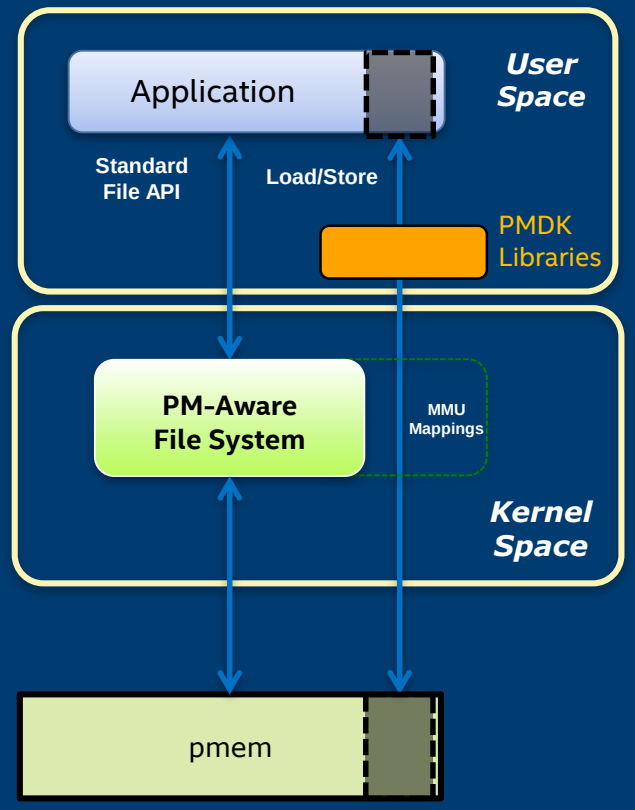
THE RESULT… PMDK

PMDK Provides a Menu of Libraries

- Developers pull in just what they need
  - Transaction APIs
  - Persistent memory allocators
- Instead of re-inventing the wheel
  - PMDK libraries are fully validated
  - PMDK libraries are performance tuned

PMDK Provides Tools for Developers

PMDK is Open Source and Product-Neutral



CURRENT STATE OF PMDK

Core libraries, roughly ten of them, in PMDK repo on GitHub

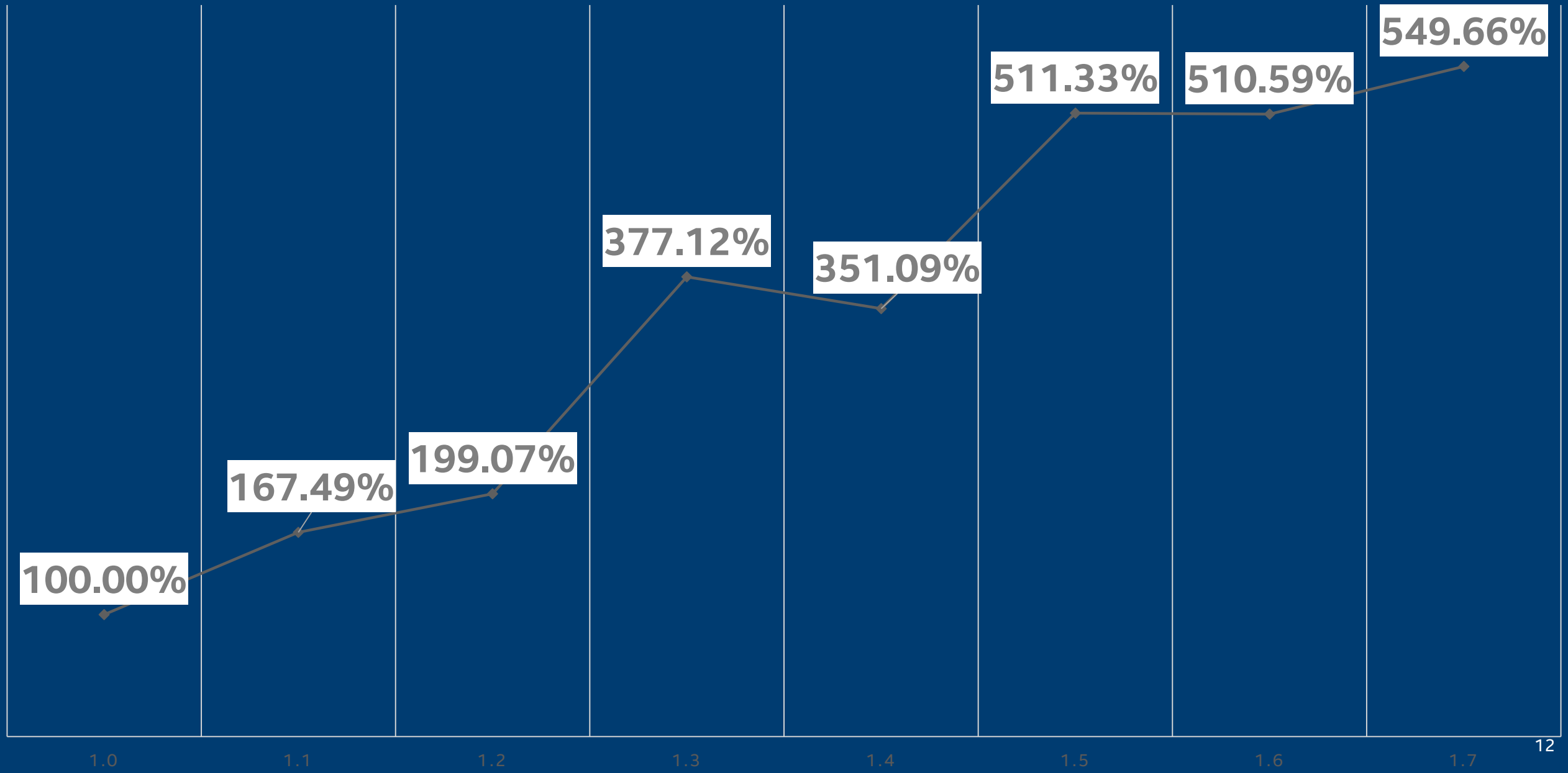
- Over 8000 commits over a period of about five years
- Dozens of users that we know about
  - Some open source
  - Some closed source
  - Some code stealers (which we encourage)
- Most intense activity has been on libpmemobj, the most flexible library

Team took over maintenance of libmemkind

- For volatile use cases

Lots of interesting additions since the initial set of libraries…

LIBPMEMOBJ RELATIVE PERFORMANCE ACROSS VERSIONS (B-TREE BENCHMARK)



PMDK EVOLUTION

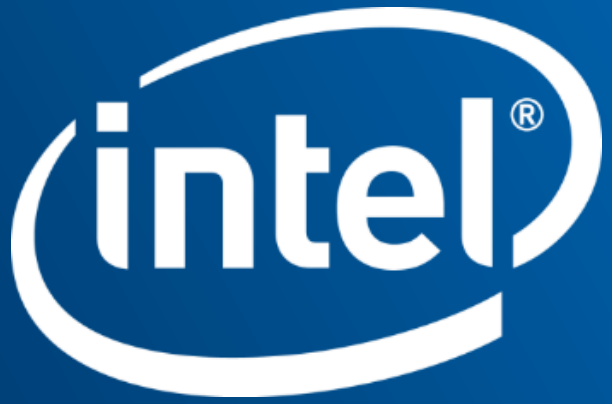
New libraries based on use cases and customer feedback (see talks on many of these!)

- Java support
- C++ support
  - Some of the most interesting & challenging work in this space
  - Lots more in this summit about C++
- Libpmemkv
- libvmemcache
- Tools support (VTune, pmemcheck, pmreorder, etc.)

PMDK FUTURE

We're not done!

- As more use cases emerge, decide if current libraries cover them
  - Invent new libraries when it makes sense
- Get community more engaged
  - So far, only a few pull requests from outside PMDK team
  - Use SPDK as a model!
- Continue to tune, enhance, refine what we have
  - Example: more C++ containers, better C++ performance
  - Example: support more languages


intel®