SNIA PERSISTENT MEMORY PM SUMMIT

JANUARY 24, 2019 | SANTA CLARA, CA

Impact on Application Development: SNIA NVM Programming Model in the Real World

Andy Rudoff
pmem SW Architect, Intel

Agenda

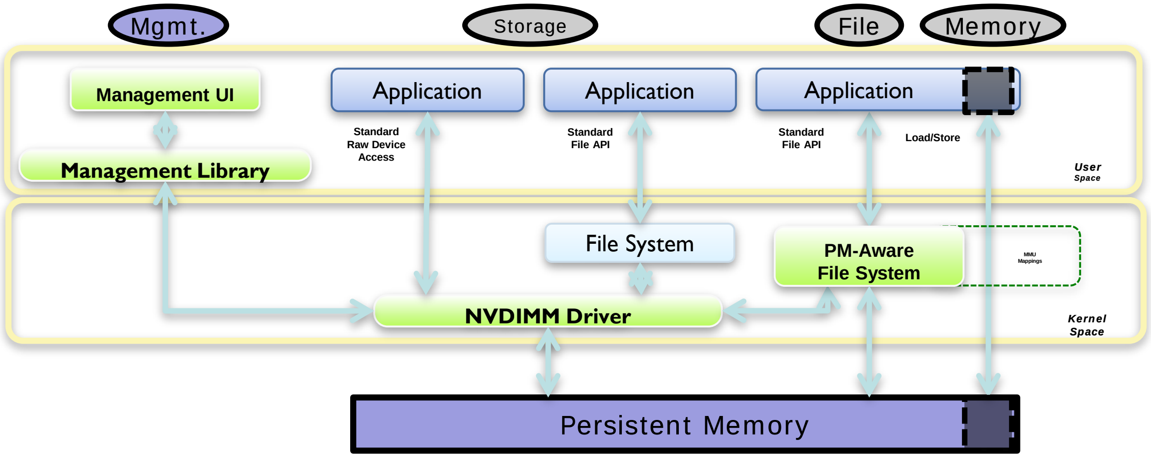
- What everyone already knows about pmem…
- What everyone forgets…
- Ways to use pmem with no app modifications
- Ways to use pmem with app modifications
- Learnings so far
- Where we're heading

Everyone knows…

- Persistent memory…
  - Allows load/store access like memory
  - Is persistent like storage
  - Exposed to applications using SNIA NVM TWG model
- What isn't persistent memory:
  - Something that can only speak blocks (like a disk/SSD)
  - Something that is too slow for load/store access
    - TWG's language:
    - Would reasonably stall the CPU waiting for a load to complete

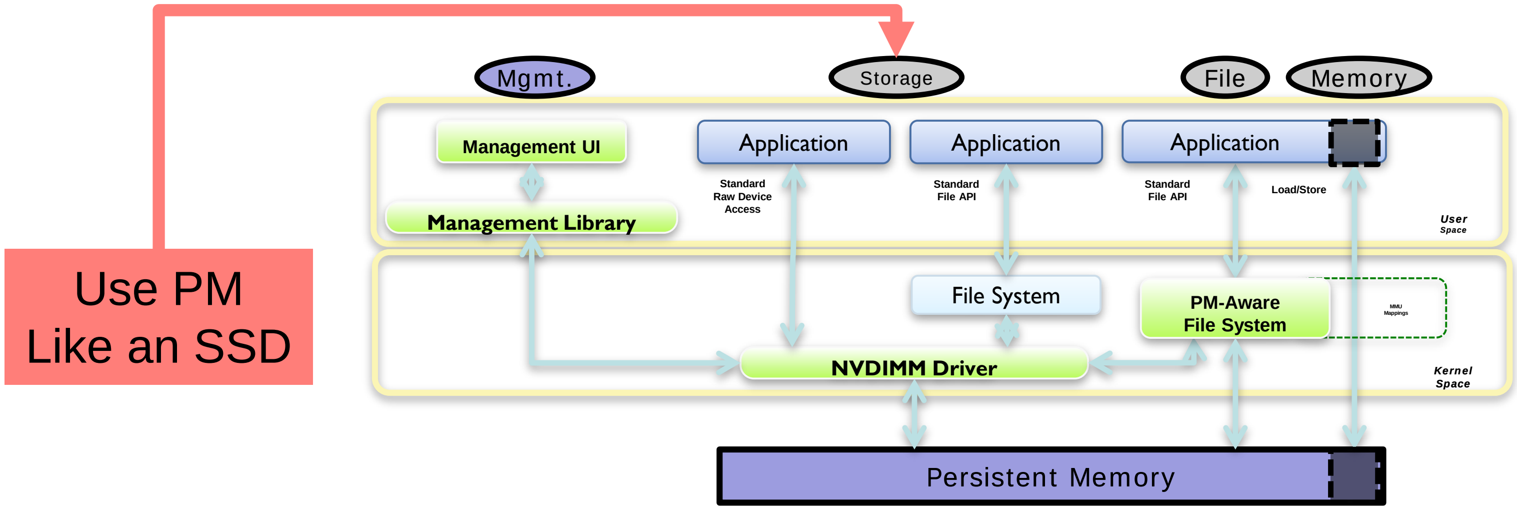
Often forgotten

- The programming model includes the storage APIs!



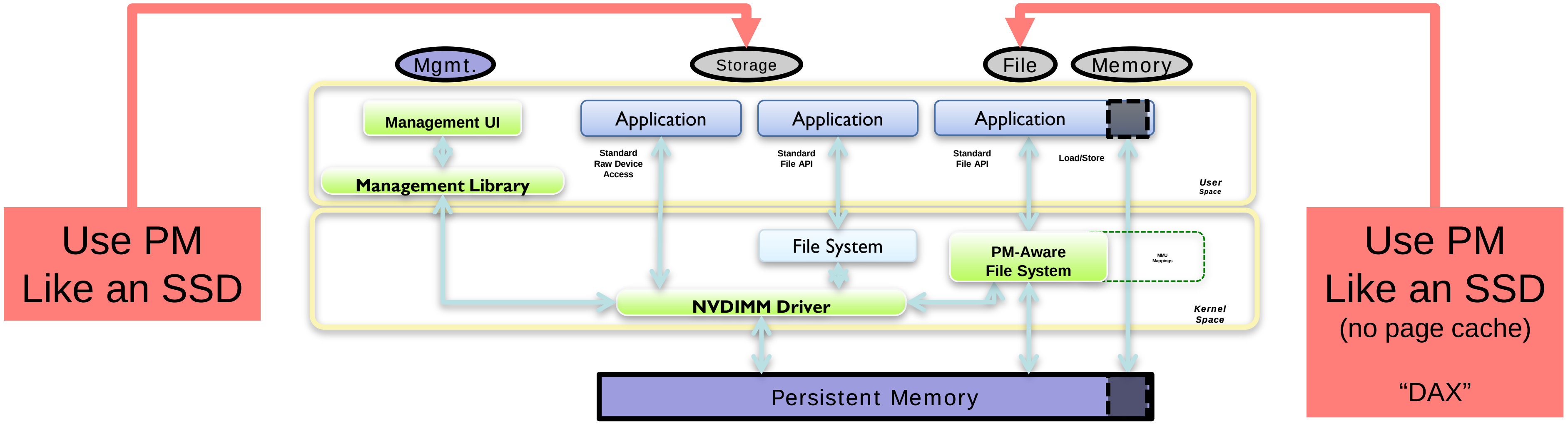
Often forgotten: Storage Access

- The programming model includes the storage APIs!

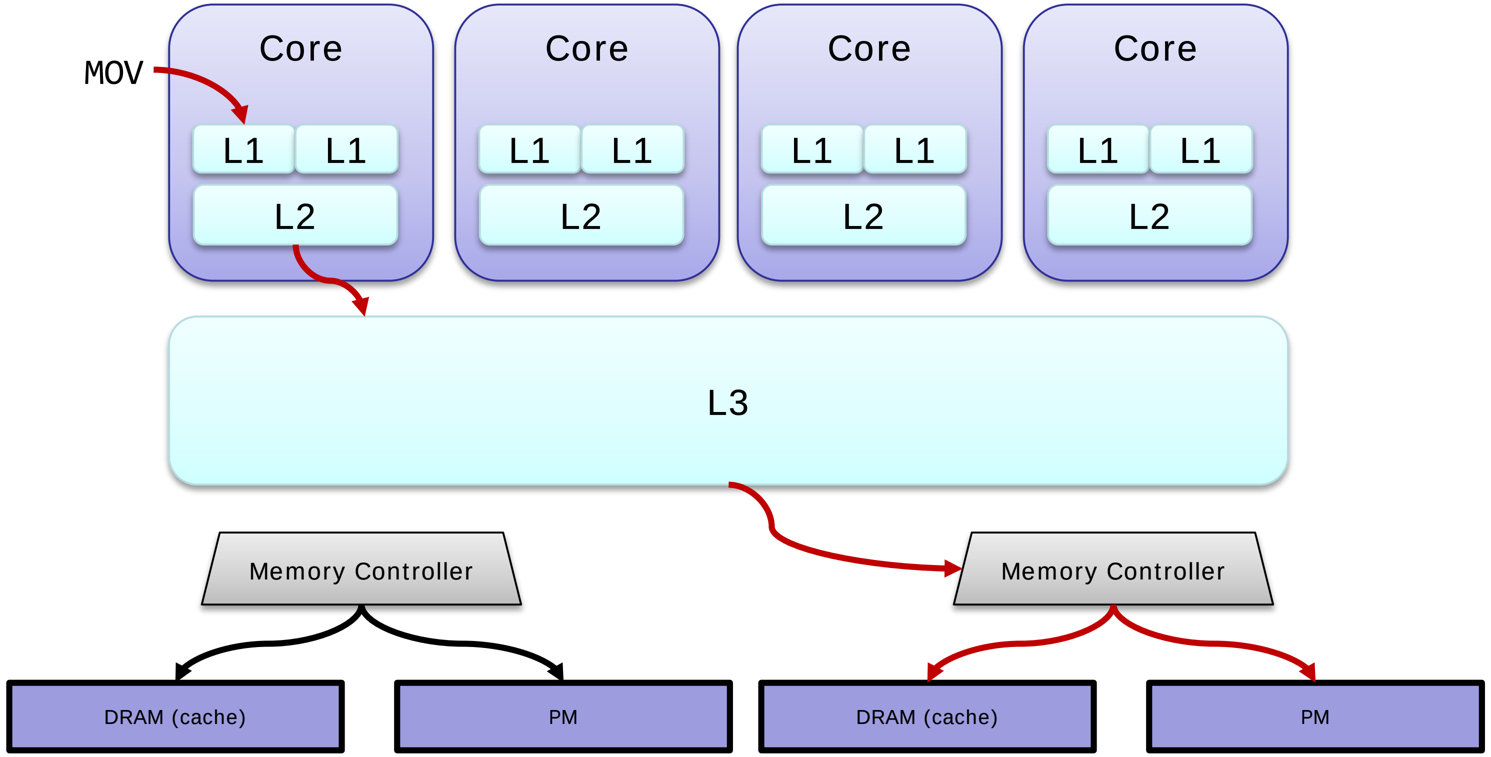


Often forgotten: DAX Access

- The programming model includes the storage APIs!



Memory Mode: Volatile Capacity

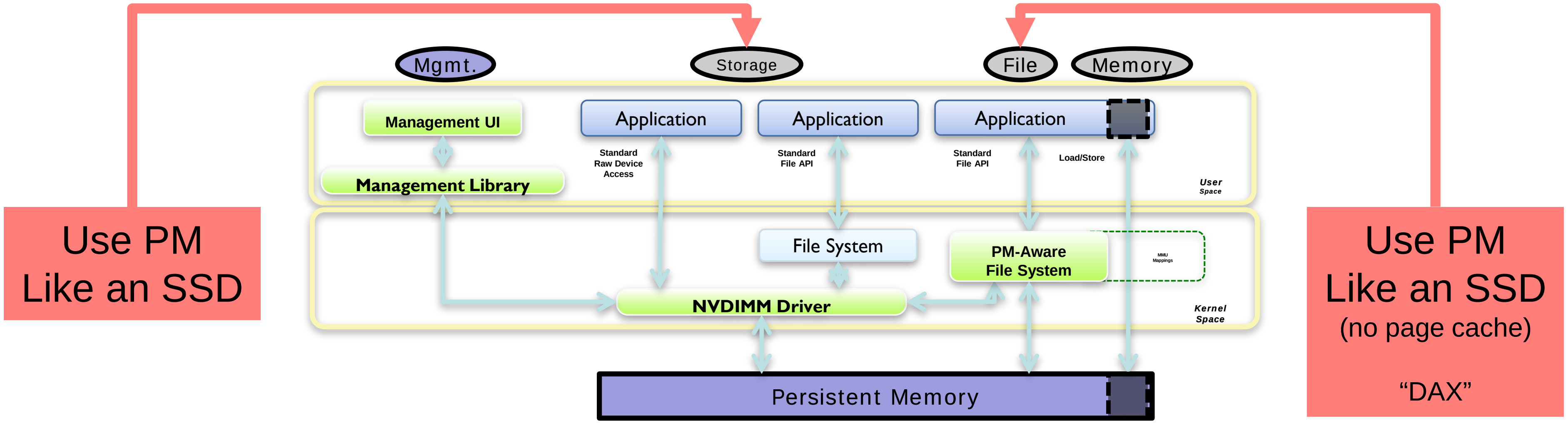


No Application Modification

- Using PM as a fast SSD
  - Storage APIs work as expected
  - Memory-mapping files will page them into DRAM
- Using PM as DAX
  - Storage APIs work as expected
  - No paging (DAX stands for “Direct Access”)
- Using PM as volatile capacity
  - Just big main memory
  - Vendor-specific feature

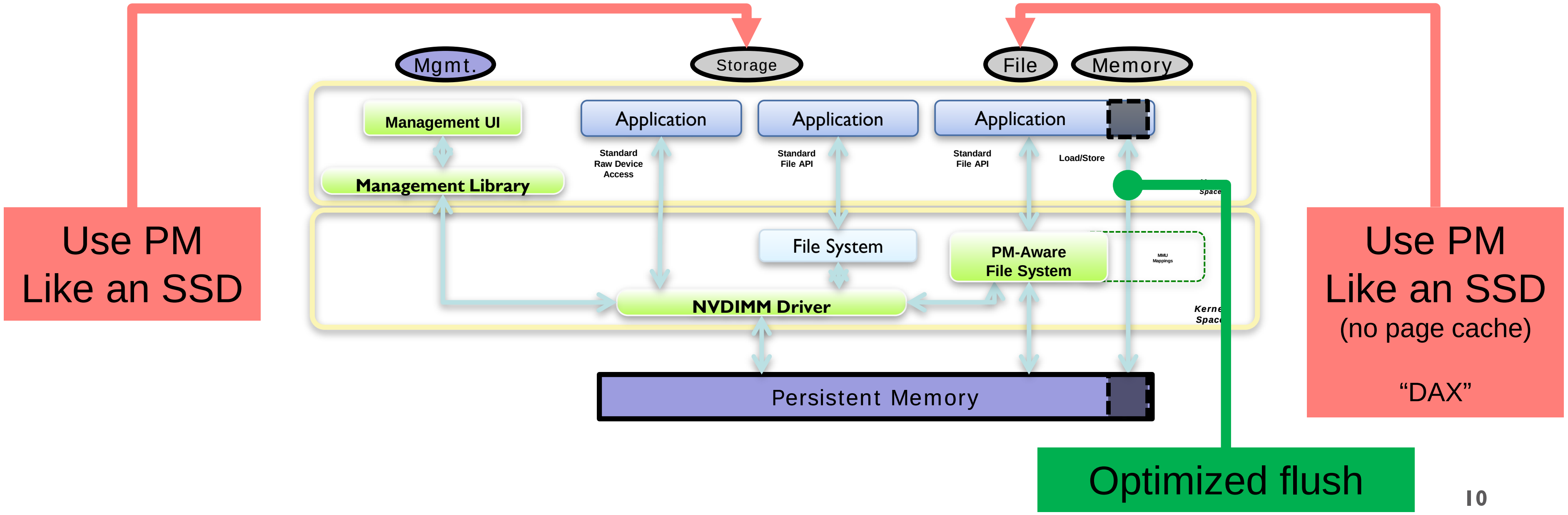
Often forgotten: DAX Access

- The programming model includes the storage APIs!



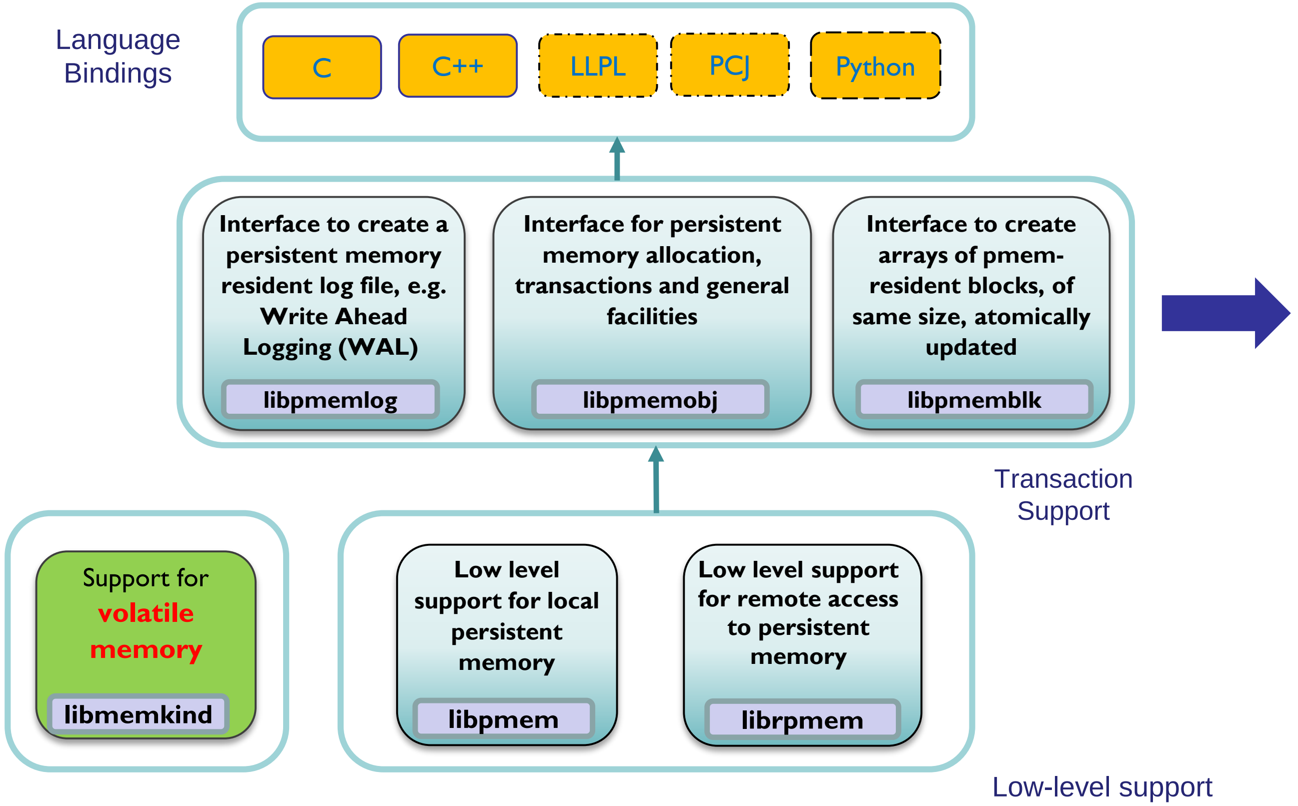
Optimized Flush: Flushing from Userspace

- The programming model includes the storage APIs!

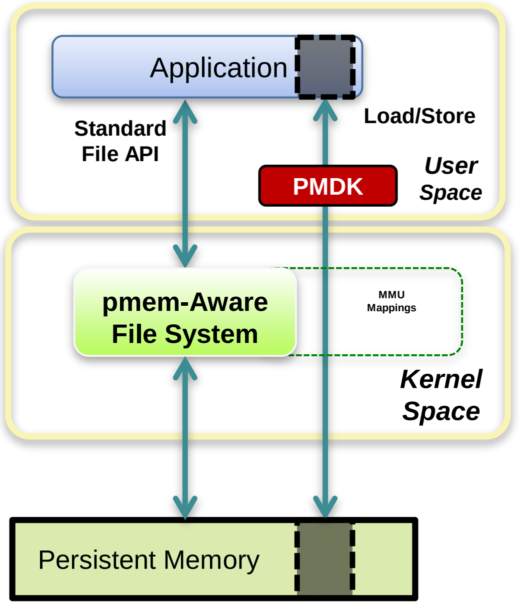


PERSISTENT MEMORY PM SUMMIT
JANUARY 24, 2019 | SANTA CLARA, CA

Application Modification

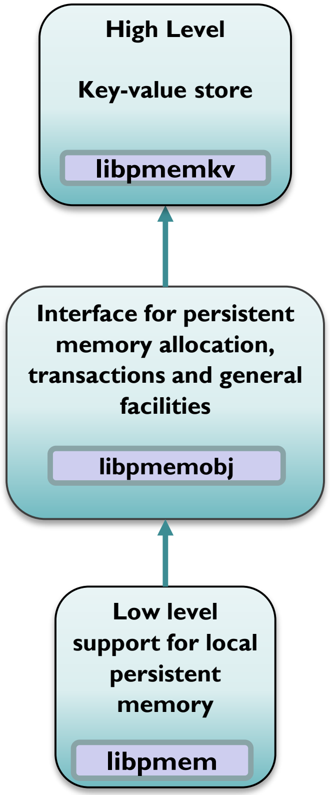


Language Bindings
C
C++
LLPL
PCJ
Python
Interface to create a persistent memory resident log file, e.g. Write Ahead Logging (WAL)
libpmemlog
Interface for persistent memory allocation, transactions and general facilities
libpmemobj
Interface to create arrays of pmem-resident blocks, of same size, atomically updated
libpmemblk
Transaction Support
Support for volatile memory
libmemkind
Low level support for local persistent memory
libpmem
Low level support for remote access to persistent memory
librpmem
Low-level support



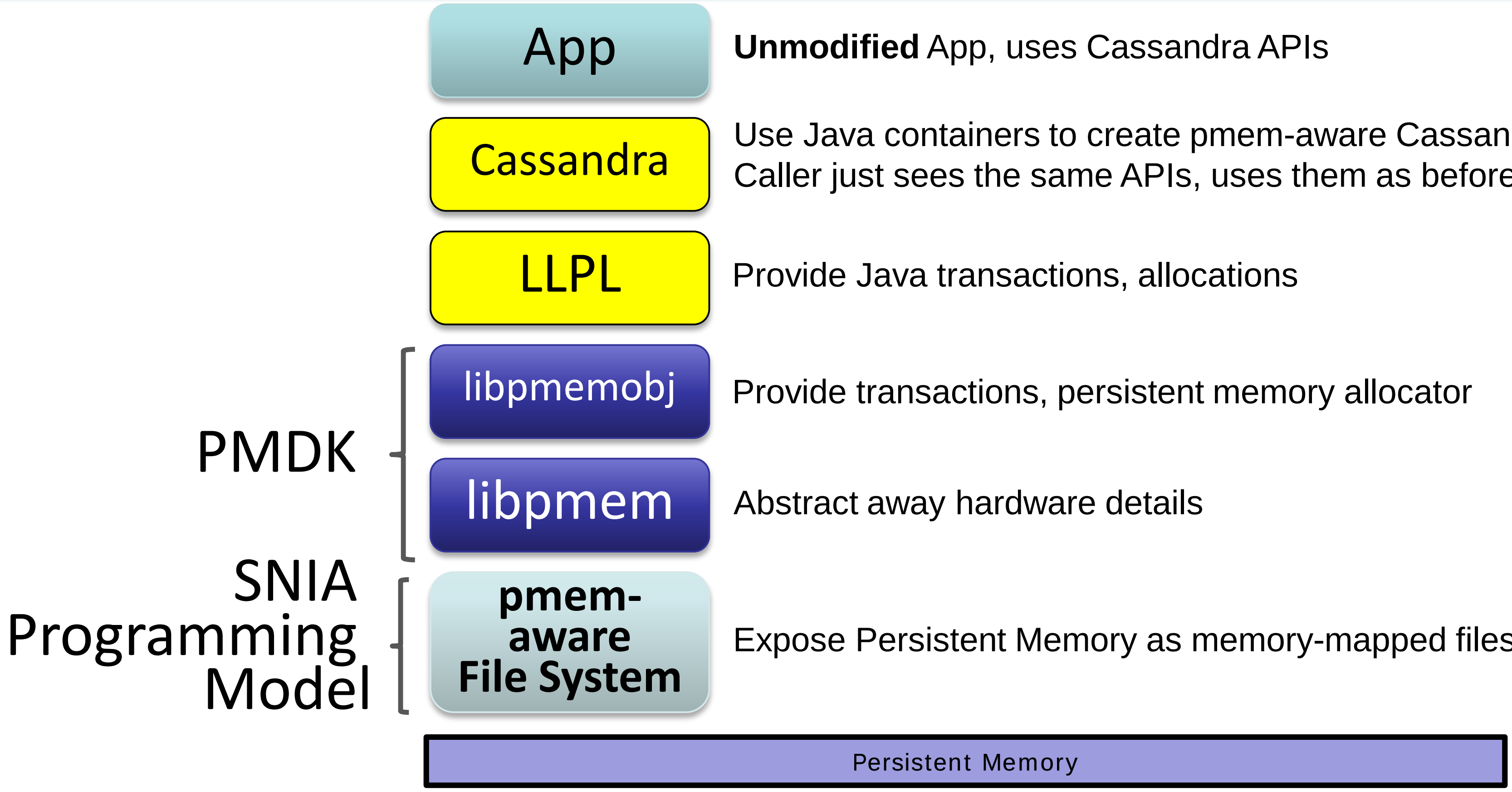
In Development:
PCJ – Persistent Collection for Java
LLPL – Low-Level Persistence Java Library

Application Modification: pmemkv


High Level
Key-value store
libpmemkv
Interface for persistent memory allocation, transactions and general facilities
libpmemobj
Low level support for local persistent memory
libpmem

- libpmemkv
  - Experimental
  - General-purpose key-value store
  - Multiple pluggable engines
  - Multiple language bindings
  - Productization underway
- Caller uses simple API
  - But gets benefits of persistent memory

Full Stack Example



Unmodified App, uses Cassandra APIs

Use Java containers to create pmem-aware Cassandra
Caller just sees the same APIs, uses them as before

Provide Java transactions, allocations

Provide transactions, persistent memory allocator

Abstract away hardware details

Expose Persistent Memory as memory-mapped files (DAX)

Learnings so far…

- Lots of ways to use PM without app modifications
- Try first to use existing APIs
  - Example: app that can be configured for SSD tier
- Try next to use highest abstraction possible
  - Key-value store, simple block or log interfaces
- Try next to use a transaction library
  - libpmemobj
- Finally, if you must program to raw mapped access

Where we're heading

- More transparent use cases
  - Either kernel or library features, transparent to app
- More high-level abstractions
  - Easier to program, less error prone
- More support for experts as well
  - More features in transaction libraries
  - More language integration
  - Faster remote access