PMDK - STATE OF THE PROJECT

Andy Rudoff, Piotr Balcer

Intel® Corporation

Storage Performance Development Kit (SPDK)
Persistent Memory Development Kit (PMDK)
Intel® VTune™ Profiler

Virtual Forum

AGENDA

01 **Brief historical overview**
How we came to be

02 **Direction and goals of PMDK**
What are we doing and why

03 **Current state of the project**
What have we done so far

04 **A look into the future**
What is next in our journey

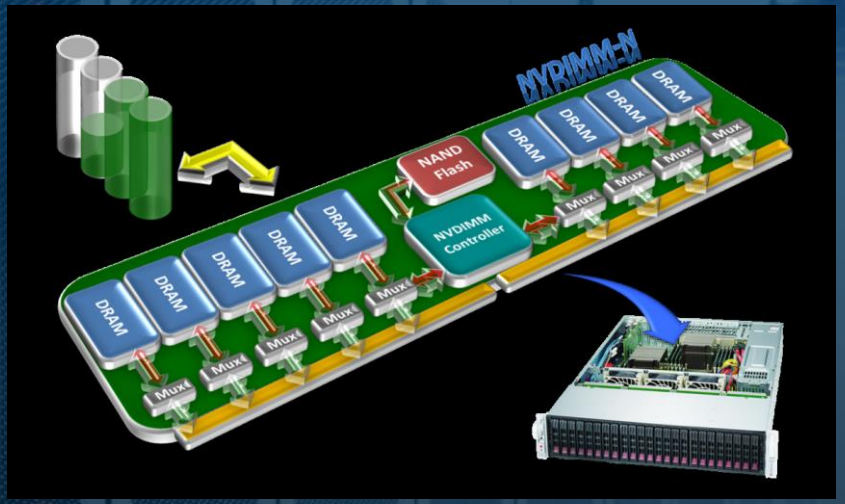
Storage Performance Development Kit (SPDK)
Persistent Memory Development Kit (PMDK)
Intel® VTune™ Profiler

Virtual Forum

01

Brief historical overview

BACK WHEN WE HEARD: "PERSISTENT MEMORY IS COMING..."



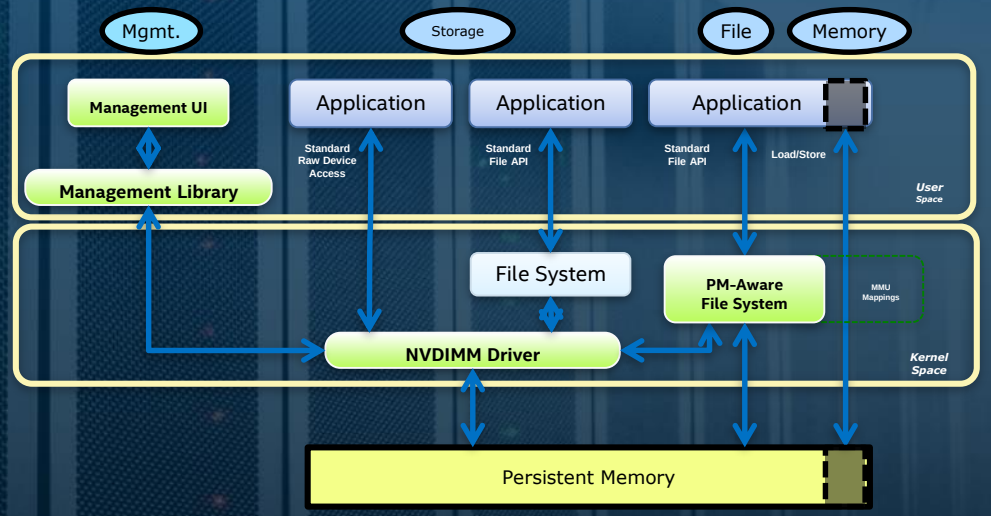
Byte-addressable, use it like memory

- 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_)
```

Storage Performance Development Kit (SPDK)
Persistent Memory Development Kit (PMDK)
Intel® VTune™ Profiler

Virtual Forum

02

Direction and goals of PMDK

SOLVING REAL PROBLEMS USING PERSISTENT MEMORY

PMEM is multidimensional. It's both memory and storage.

- As memory, it's more affordable and bigger than DRAM.
  - Enabling previously impossible (or impossibly expensive) use-cases on multi-terabyte heterogenous memory systems.
- As storage, it's an order of magnitude faster compared to other solutions.
  - Enabling ultra-low latency retrievals and transactions, potentially also reducing overall memory cost by bypassing the cache.
- As both, it's unique.
  - Enabling new designs that require new unique solutions.

PERSISTENT MEMORY AS MEMORY

- Persistent Memory is bigger, but slower than DRAM.
- PMEM is one kind of memory that can be present in a heterogeneous memory system.
  - Applications typically assume that all memory is the same.
    - The OS kernel can be made to emulate this status quo (Memory Tiering).
  - … but, even today, that's simply not the case.
    - NUMA, High-Bandwidth Memory, PMEM and more.
- **PMDK aids applications is intelligent and scalable memory placement.**

PERSISTENT MEMORY AS STORAGE

- Persistent Memory is smaller, but faster than traditional storage.
  - This is not unprecedented. SSDs were a similar disruption.
  - Techniques developed then, make sense now.
    - Storage caching & tiering, separating data from write-ahead logs, …
- Thanks to DAX, Persistent Memory can also reduce the reliance on page cache in applications that use memory-mapped I/O.
  - This reduces cost and guarantees stable latency unhindered by page faults.

- **PMDK aids in modifications of existing storage solutions.**

PERSISTENT MEMORY AS BOTH STORAGE AND MEMORY

- Database storage engine design is essentially a study on how to mask the large difference between storage and memory.
  - We don't have to do that any more… sort of :)
- Persistent Memory is a new tier that bridges the gap between Memory and Storage.
  - Enables new techniques that reduce access latency and write amplification.
    - Fault tolerant algorithms still need to log data, but can now do so using a single load/store instructions at cacheline granularity.
- **PMDK aids in using novel techniques that merge memory and storage.**

GENERAL DIRECTIONS AND GOALS

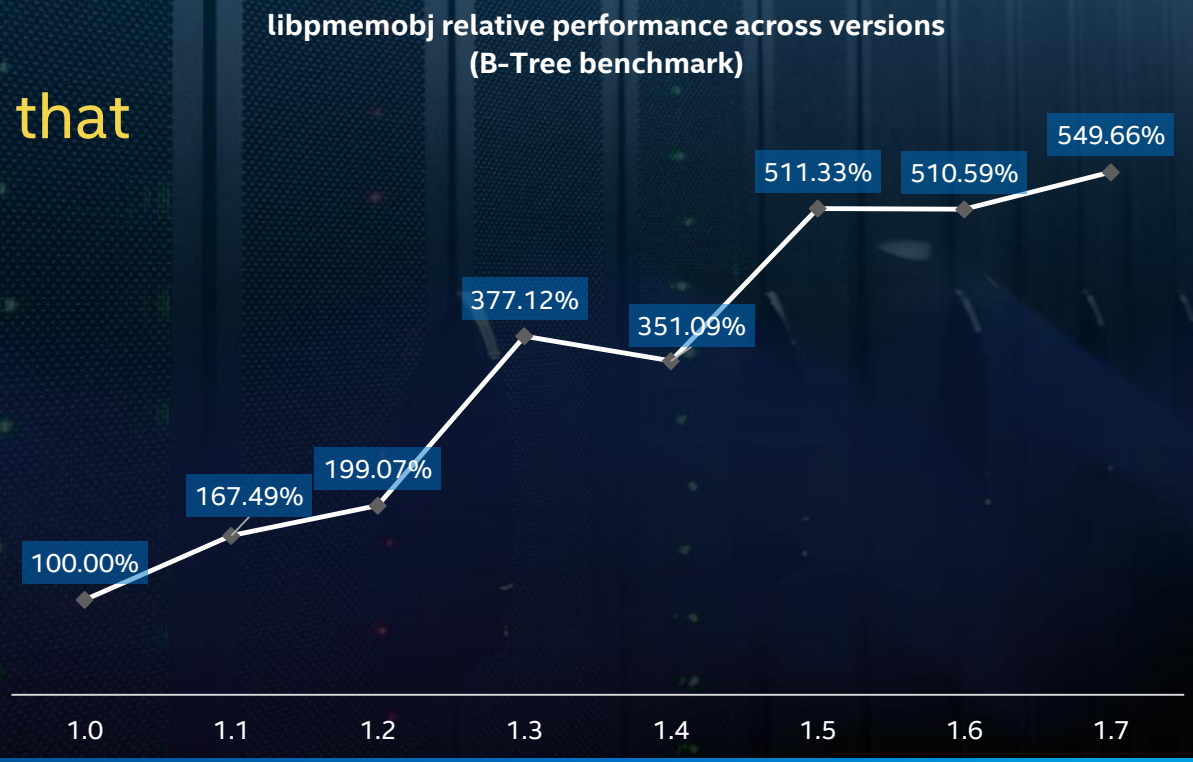
“Make easy things easy and hard things possible”

- Larry Wall, about Perl programming language.

- PMDKs goal was, is, and always will be making Persistent Memory programming easy.
- But also enable solving complex and possibly challenging problems commonly encountered by users.
  - This is done through a multi-layered stack of solutions, with each building block adding new functionality on top of the previous one.
  - Applications can choose their desired level of abstraction.

DURABILITY, CONSISTENCY, RELIABILITY, PERFORMANCE

- Performance isn't everything...
  - Things that are fast, and superficially appear to work, are not only not useful, but actively harmful.
- PMDK's **primary** focus is on making sure that the functionality it provides is reliable.
  - We run thousands of tests, some with novel techniques, like byte-level crash consistency checking.
- But at the same time, we don't neglect perform.



Storage Performance Development Kit (SPDK)
Persistent Memory Development Kit (PMDK)
Intel® VTune™ Profiler

Virtual Forum

03

Current state of the project

PMDK LIBRARIES

http://pmem.io
https://github.com/pmem/pmdk

High Level Interfaces

C++ Persistent Containers

LLPL (Java) Low-Level Persistent Library

pmemkv

Language support

C++ | C | Java | Python

Transaction Support

Interface to create a persistent memory resident log file

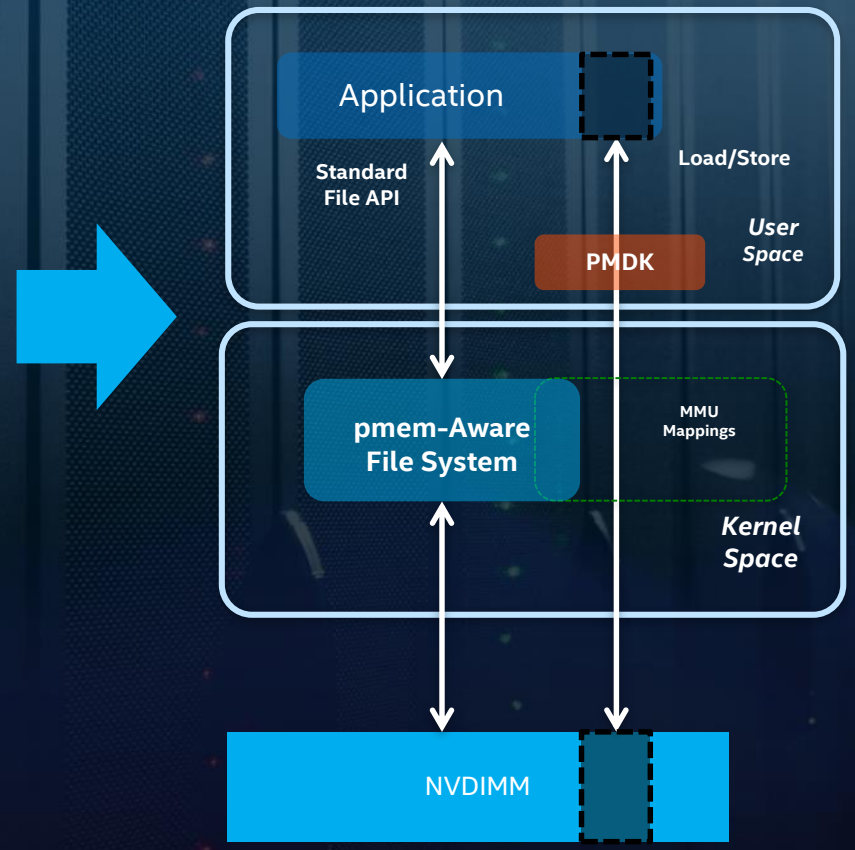
libpmemlog

Interface for persistent memory allocation, transactions and general facilities

libpmemobj

Interface to create arrays of pmem-resident blocks, of same size, atomically updated

libpmemblk



Support for **volatile** memory usage

memkind

vmemcache

Low level support for local persistent memory

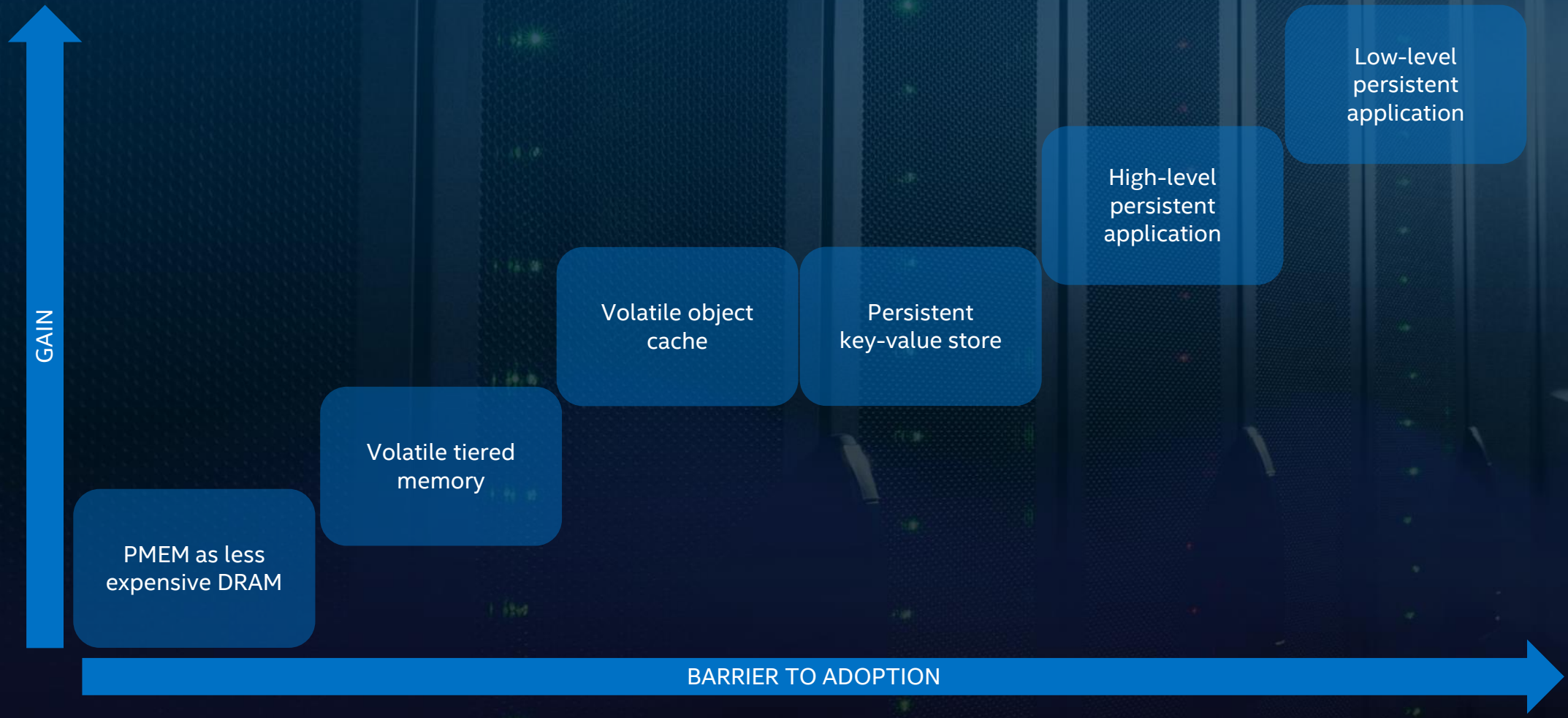
libpmem

Low level support for remote access to persistent memory

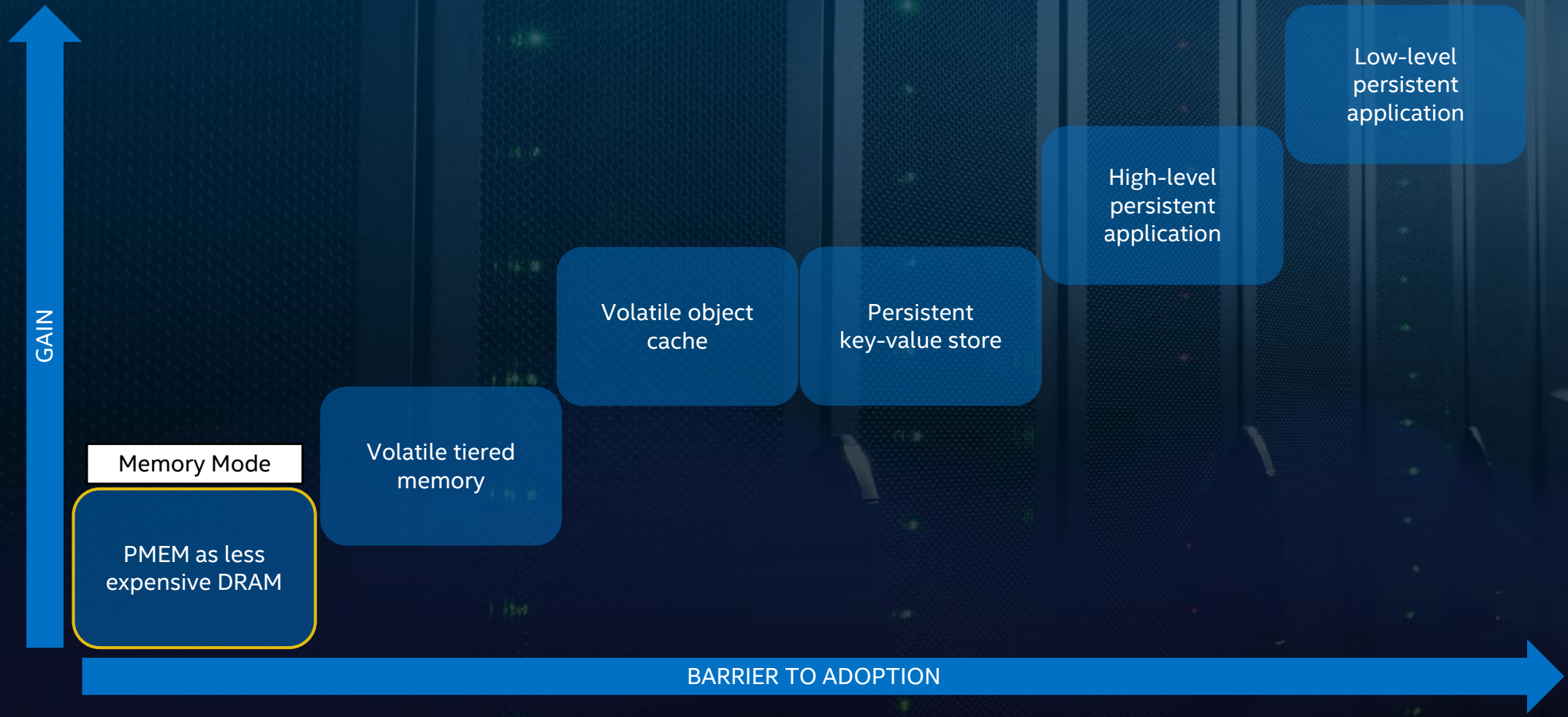
librpmem

Low-level support

DIFFERENT WAYS TO USE PERSISTENT MEMORY



DIFFERENT WAYS TO USE PERSISTENT MEMORY



MEMORY MODE

- Not really a part of PMDK…
- … but it's the easiest way to take advantage of Persistent Memory

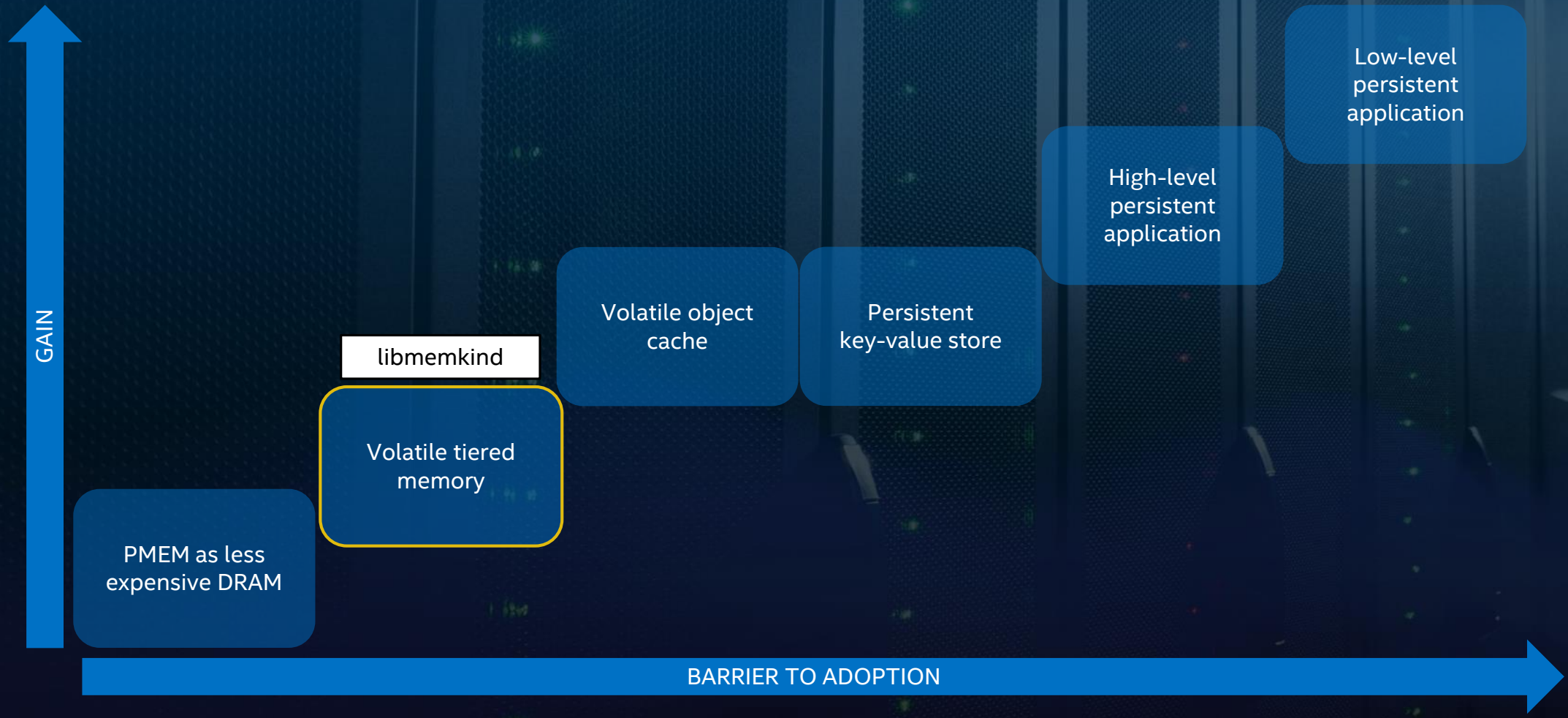
When To Use

- modifying applications is not feasible
- massive amounts of memory is required (more TB)
- CPU utilization is low in shared environment (more VMs)

```
char *memory = malloc(sizeof(struct my_object));
strcpy(memory, "Hello World");
```

- Memory is automatically placed in PMEM, with caching in DRAM

DIFFERENT WAYS TO USE PERSISTENT MEMORY



LIBMEMKIND

When To Use

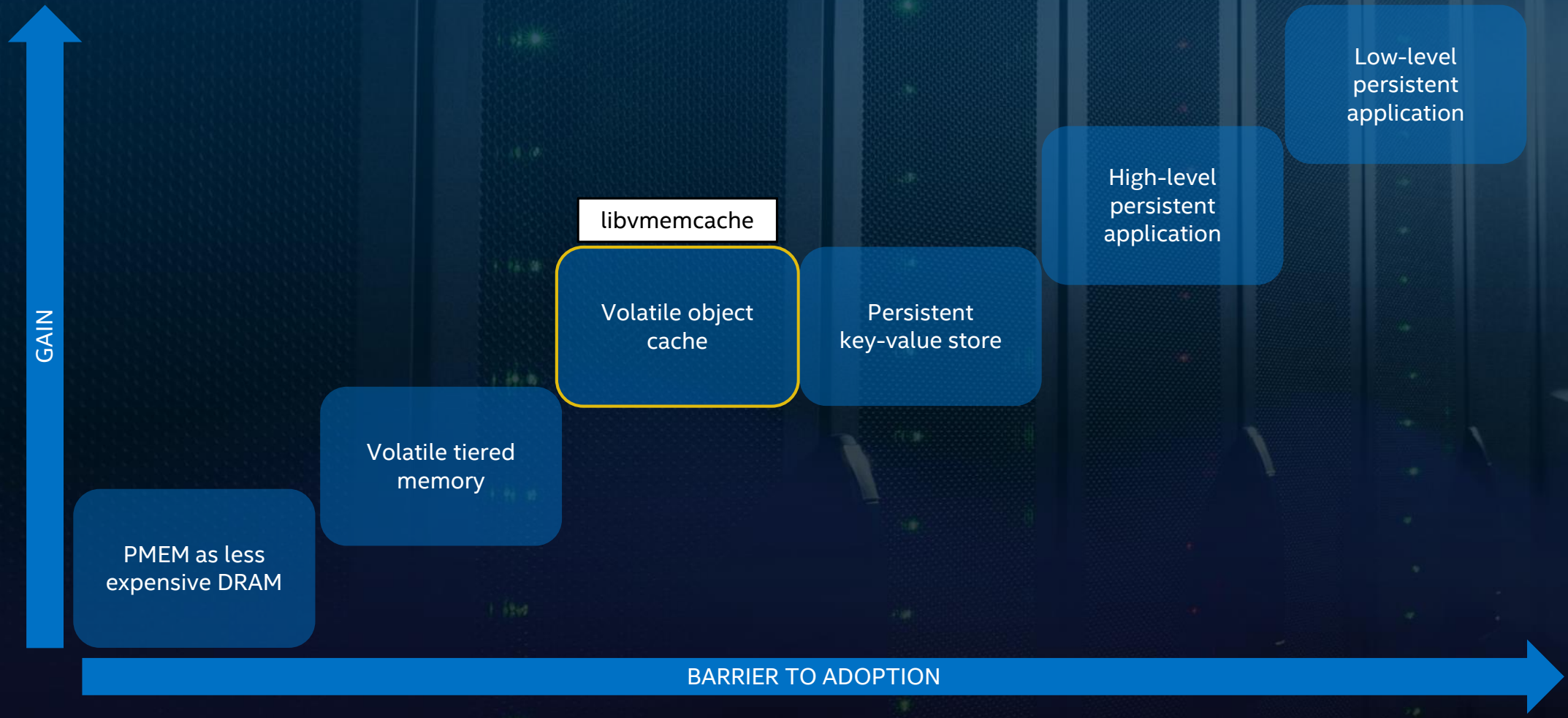
- application can be modified
- different tiers of objects (hot, warm) can be identified
- persistence is not required

- Explicitly manage allocations from PMEM, allowing for fine-grained control of memory placement

```
struct memkind *pmem_kind = NULL;
size_t max_size = 1 << 30; /* gigabyte */

/* Create PMEM partition with specific size */
memkind_create_pmem(PMEM_DIR, max_size, &pmem_kind);

/* allocate 512 bytes from 1 GB available */
char *pmem_string = (char *)memkind_malloc(pmem_kind, 512);

/* deallocate the pmem object */
memkind_free(pmem_kind, pmem_string);
```

- Application can decide what type of memory to use for objects.

DIFFERENT WAYS TO USE PERSISTENT MEMORY



LIBVMEMCACHE

When To Use

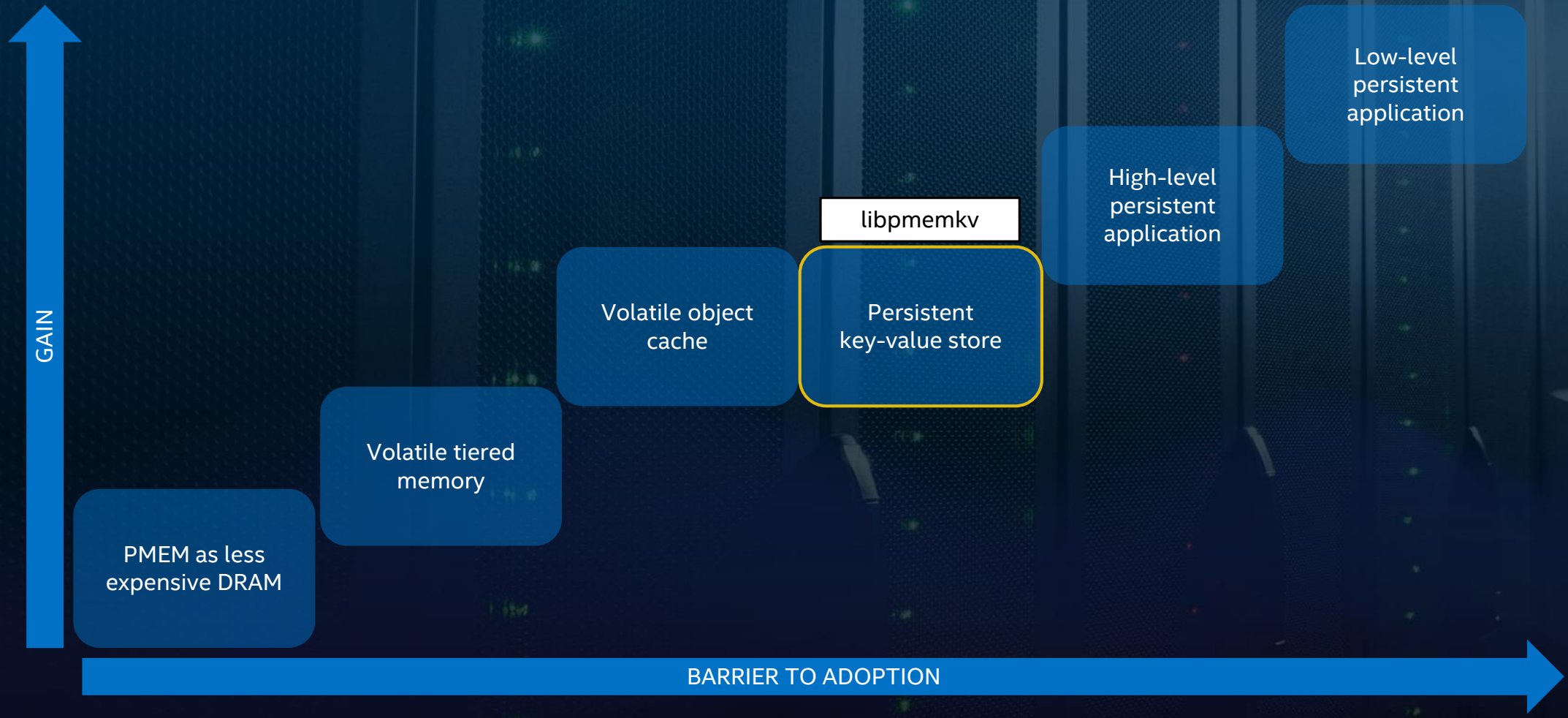
- caching large quantities of data
- low latency of operations is needed
- persistence is not required

- Seamless and easy-to-use LRU caching solution for persistent memory
  Keys reside in DRAM, values reside in PMEM

```
VMEMcache *cache = vmemcache_new();
vmemcache_add(cache, "/tmp");

const char *key = "foo";
vmemcache_put(cache, key, strlen(key), "bar", sizeof("bar"));

char buf[128];
ssize_t len = vmemcache_get(cache, key, strlen(key),
    buf, sizeof(buf), 0, NULL);

vmemcache_delete(cache);
```

- Designed for easy integration with existing systems

DIFFERENT WAYS TO USE PERSISTENT MEMORY



LIBPMEMKV

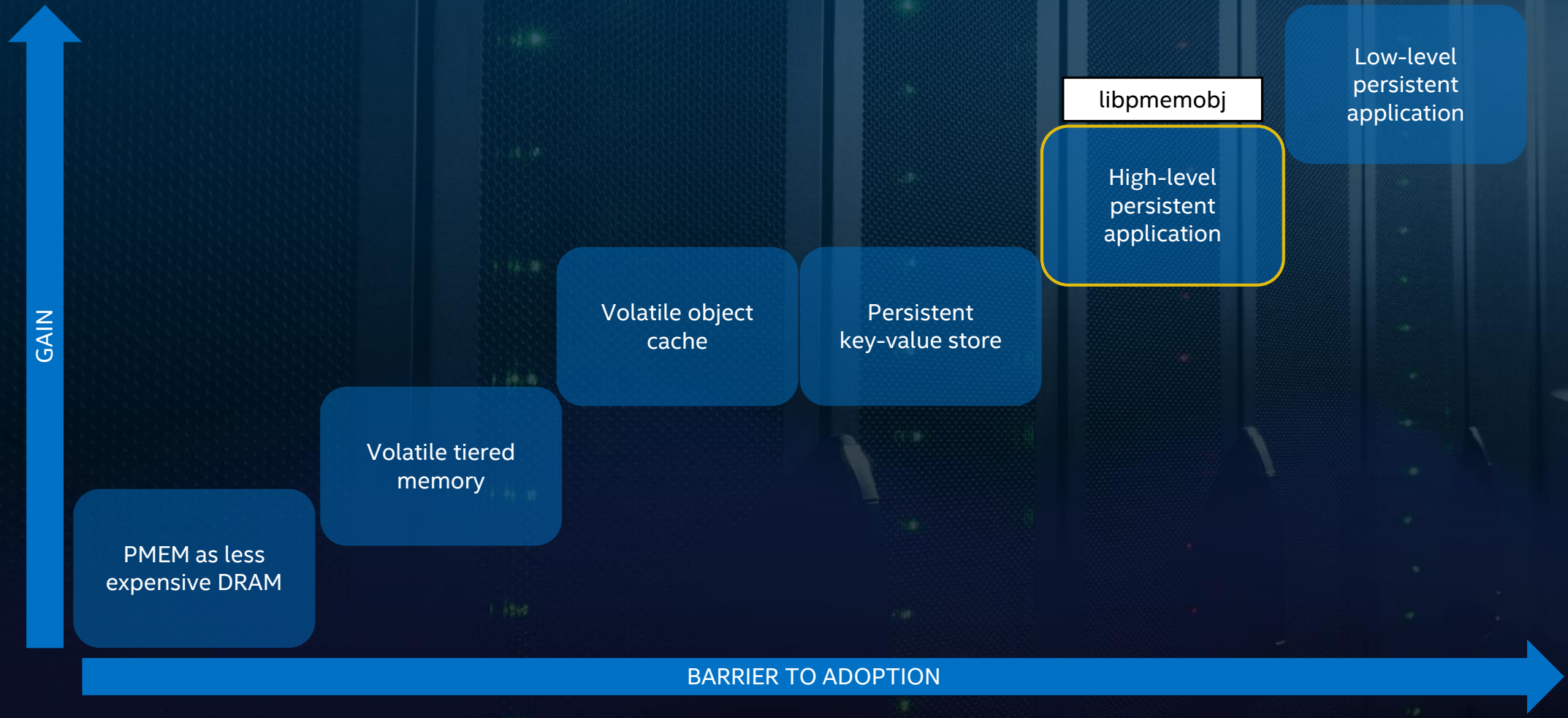
When To Use

- storing large quantities of data
- low latency of operations is needed
- persistence is required

- Local/embedded key-value datastore optimized for persistent memory. Provides different language bindings and storage engines.

```
// add the given key-value pair
if (kv->put(argv[2], argv[3]) != status::OK) {
    cerr << db::errormsg() << endl;
    exit(1);
}
// lookup the given key and print the value
auto ret = kv->get(argv[2], [&](string_view value) {
    cout << argv[2] << "=\"" << value.data() << "\"" << endl;
});
if (ret != status::OK) {
    cerr << db::errormsg() << endl;
    exit(1);
}
```

DIFFERENT WAYS TO USE PERSISTENT MEMORY



LIBPMEMOBJ

When To Use

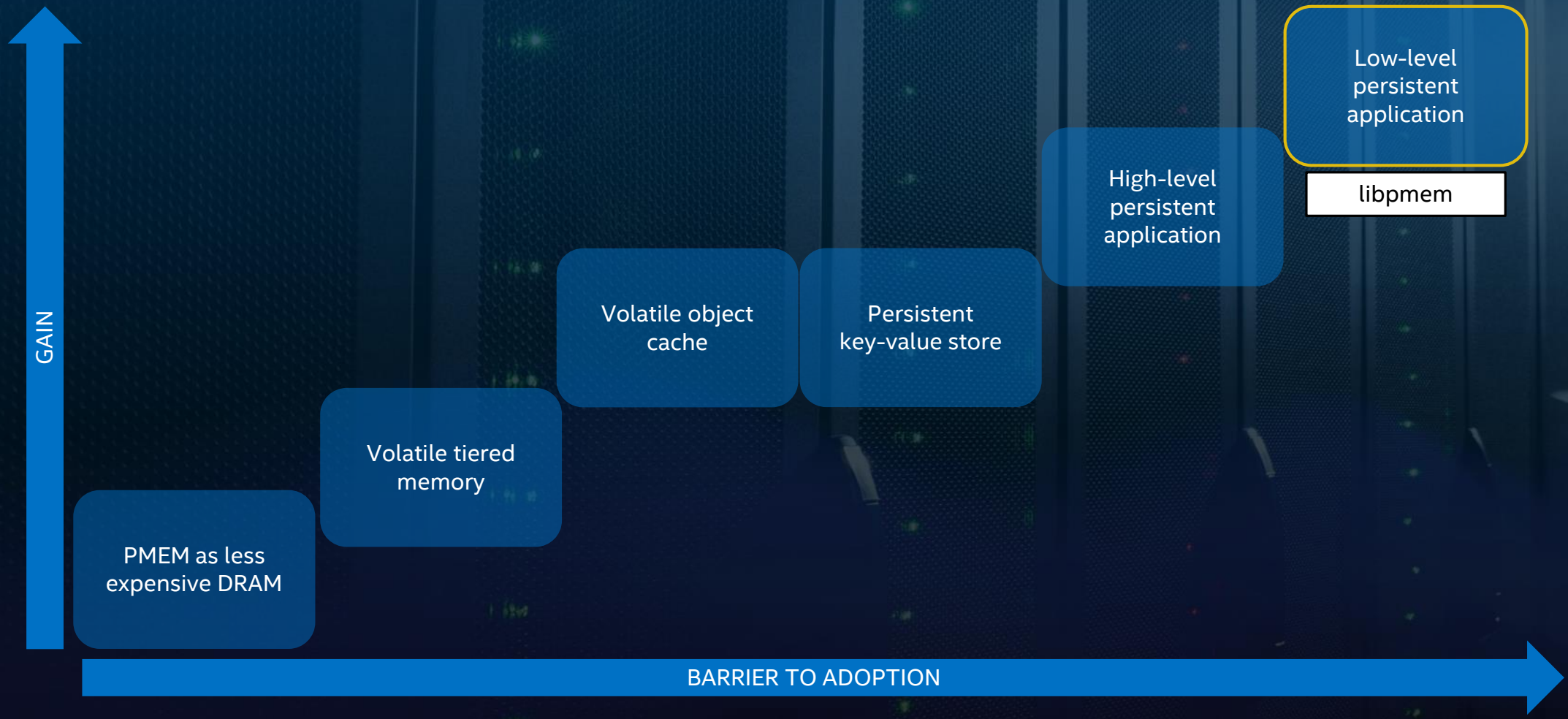
- direct byte-level access to objects is needed
- using custom storage-layer algorithms
- persistence is required

- Transactional object store, providing memory allocation, transactions, and general facilities for persistent memory programming.

```
static void
doubly_linked_list_insert(pool_base &pop, persistent_ptr<doubly_linked_list_node> prev,
   uint64_t data) {
      transaction::run(pop, [&] {
               auto node = make_persistent<doubly_linked_list_node>();
               auto next = prev->next;
               node->prev = prev; node->next = next; node->data = data;

               prev->next = node;
               next->prev = node;
      });
}
```

- Flexible and relatively easy way to leverage PMEM

DIFFERENT WAYS TO USE PERSISTENT MEMORY



LIBPMEM

When To Use

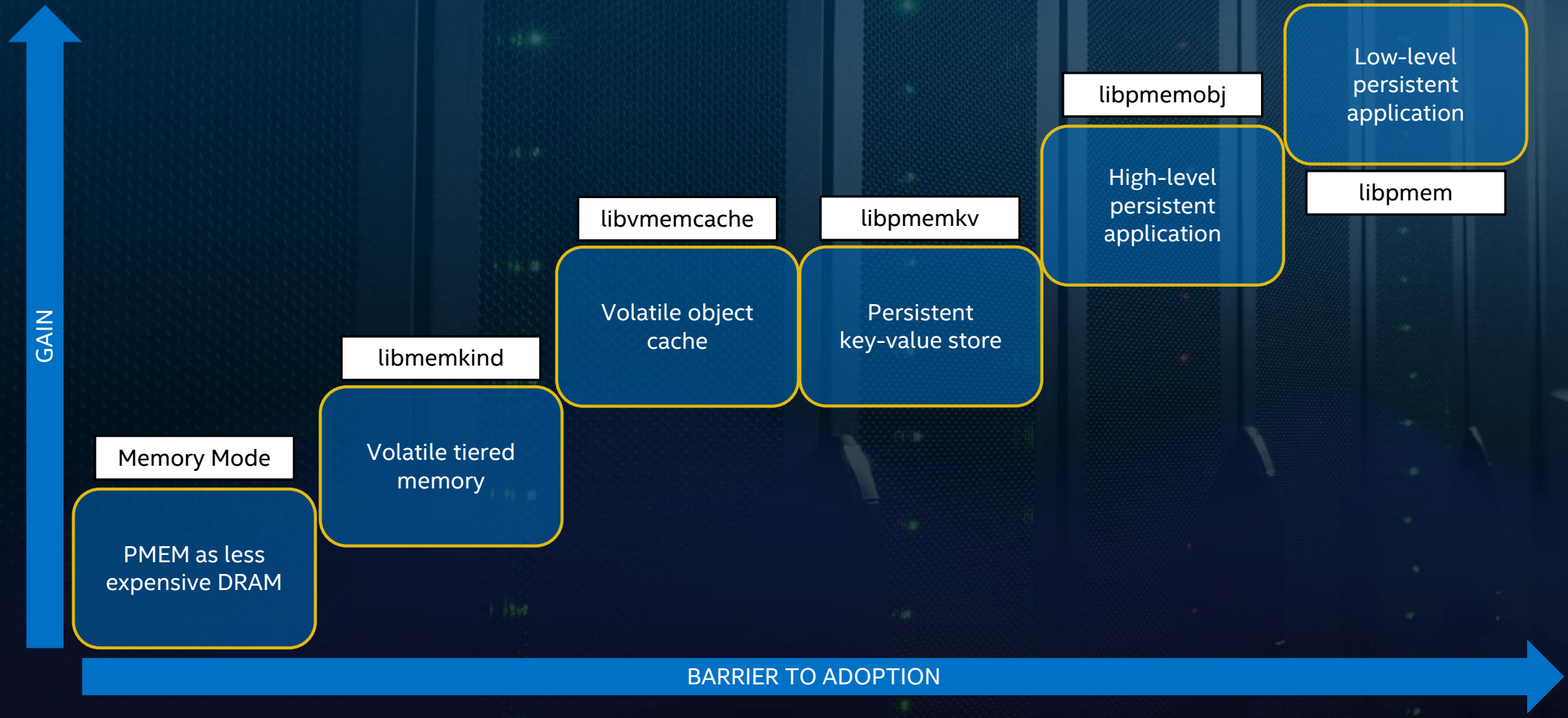
- modifying application that already uses memory mapped I/O
- other libraries are too high-level
- only need low-level PMEM-optimized primitives (memcpy etc)

- Low-level library that provides basic primitives needed for persistent memory programming and optimized memcpy/memmove/memset

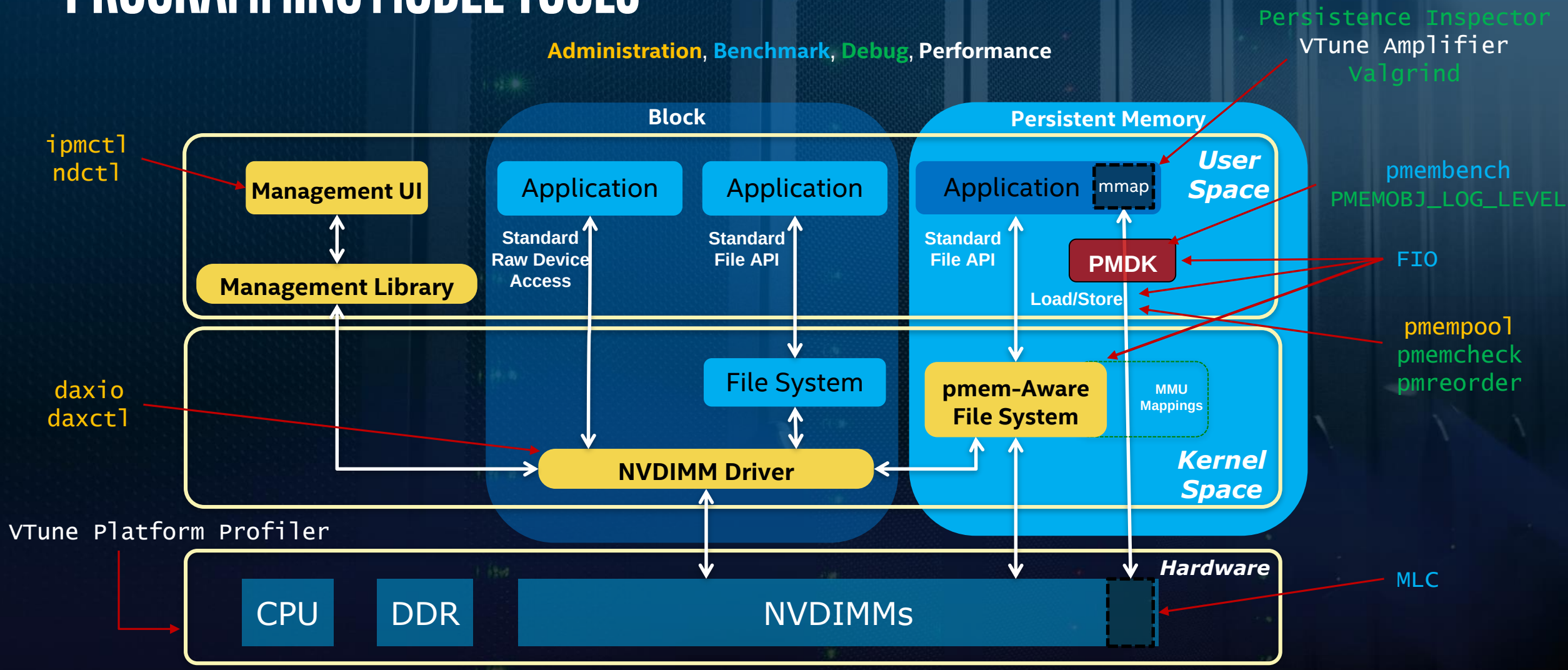
```
void *pmemaddr = pmem_map_file("/mnt/pmem/data", BUF_LEN,
            PMEM_FILE_CREATE|PMEM_FILE_EXCL,
            0666, &mapped_len, &is_pmem));
const char *data = "foo";
if (is_pmem) {
    pmem_memcpy_persist(pmemaddr, data, strlen(data));
} else {
    memcpy(pmemaddr, data, strlen(data));
    pmem_msync(pmemaddr, strlen(data));
}
close(srcfd);
pmem_unmap(pmemaddr, mapped_len);
```

- The very basics needed for PMEM programming

DIFFERENT WAYS TO USE PERSISTENT MEMORY



PROGRAMMING MODEL TOOLS



Storage Performance Development Kit (SPDK)
Persistent Memory Development Kit (PMDK)
Intel® VTune™ Profiler

Virtual Forum

04

A look into the future

EASY TO USE AND POWERFUL LOW-LEVEL PERSISTENCE PRIMITIVES

- We are introducing a new, improved, library for low-level programming.
  libpmem2
  - First-class OS abstraction | RAS APIs | Flexible mapping API

```
pmem2_config_new(&cfg);
pmem2_source_from_fd(&src, fd);
pmem2_config_set_required_store_granularity(cfg,
        PMEM2_GRANULARITY_PAGE);

pmem2_map(cfg, src, &map));

char *addr = pmem2_map_get_address(map);
pmem2_get_memcpy(map)(addr, "hello, persistent memory", ...);
```

EASY TO USE SCALABLE SOLUTIONS

- Concurrent programming is *hard*, and Persistent Memory only makes it harder.
  - But we are observing significant interest in this area.
  - Two solutions:
    improved atomic operations | built-in scalable data structures
- libpmemkv – PMDK's Key-Value store, already supports scalable operations out of the box.
- libpmemobj++ - new STL-like ordered and unordered map for PMEM.
- libpmemobj – new atomic operations for easier lock-free programming.

BETTER SUPPORT FOR HETEROGENEOUS MEMORY SYSTEMS

- Remember memkind example?
  - It explicitly allocates memory **from PMEM**.
  - … but does that matter?
- We expect that future hardware platforms will have a wide range of different memory tiers available.
  - Ideally, applications would be modified *once* and scale from homogenous single-node systems to multi-node heterogeneous ones.

```
char *fast_string = (char *)memkind_malloc(KIND_FASTMEM, 512);
char *capc_string = (char *)memkind_malloc(KIND_CAPACITY, 512);
```

CALL TO ACTION

- **“Solving real problems using persistent memory”**
  - Do you have a real problem that Persistent Memory can help solve?
    - Great! Get involved and tell us about it.
- Do you think this is an interesting research opportunity?
  - So do we! Get involved and share your ideas with the community.
- Want to just play around with examples?
  - You can get started right now. No need for real hardware.

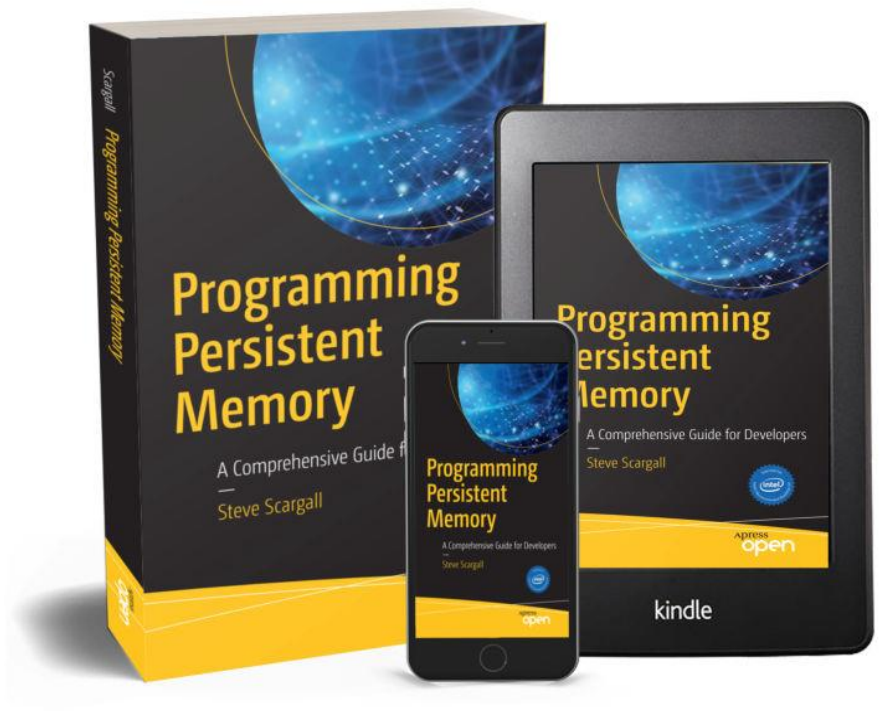
https://pmem.io/

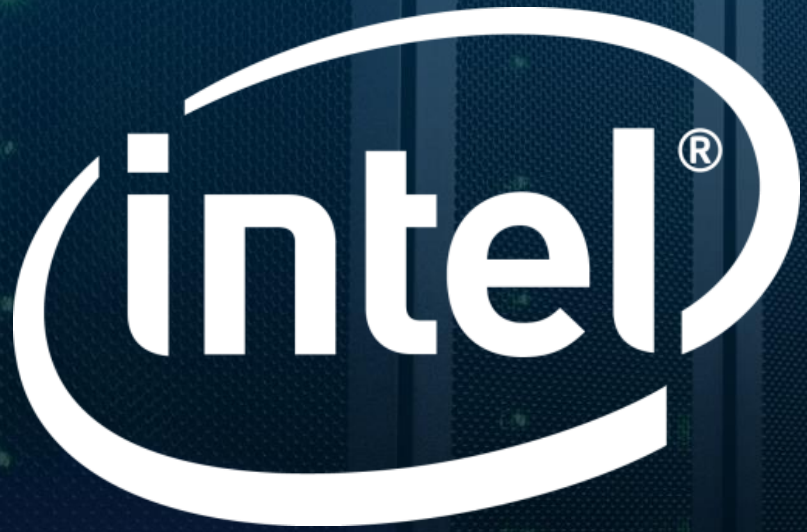
PROGRAMMING PERSISTENT MEMORY -- A COMPREHENSIVE GUIDE FOR DEVELOPERS

https://pmem.io/book/

This is the first book to fully explain the revolutionary persistent memory technology and how developers can fully utilize it.

eBook is freely available online.





Storage Performance Development Kit (SPDK)
Persistent Memory Development Kit (PMDK)
Intel® VTune™ Profiler

Virtual Forum