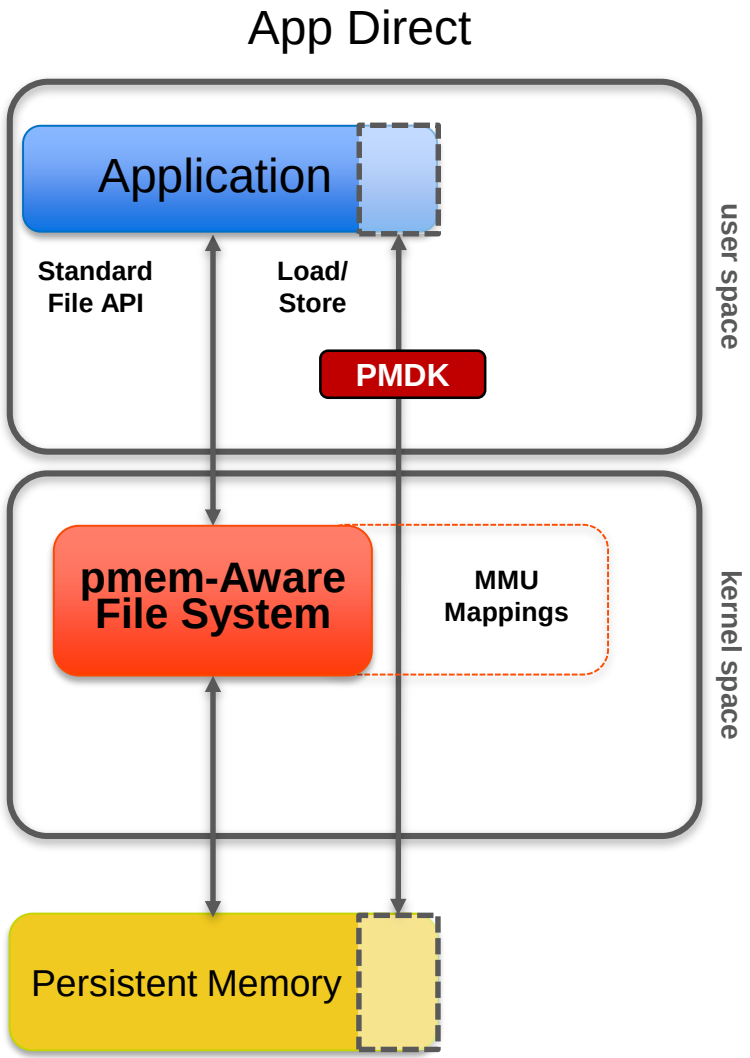
Persistent Memory Development Kit overview

PMDK overview

- http://pmem.io/
- open-source https://github.com/pmem
- vendor-agnostic
- user-space
- production quality, fully documented
- performance optimized and tuned



PMDK overview

High-level support

- Volatile key-value store optimized for PMEM — **libvmemchache**
- Local/embedded key-value store optimized for PMEM — **pmemkv**

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

Language bindings

C | C++ | LLPL | PCJ | Python

Transaction support

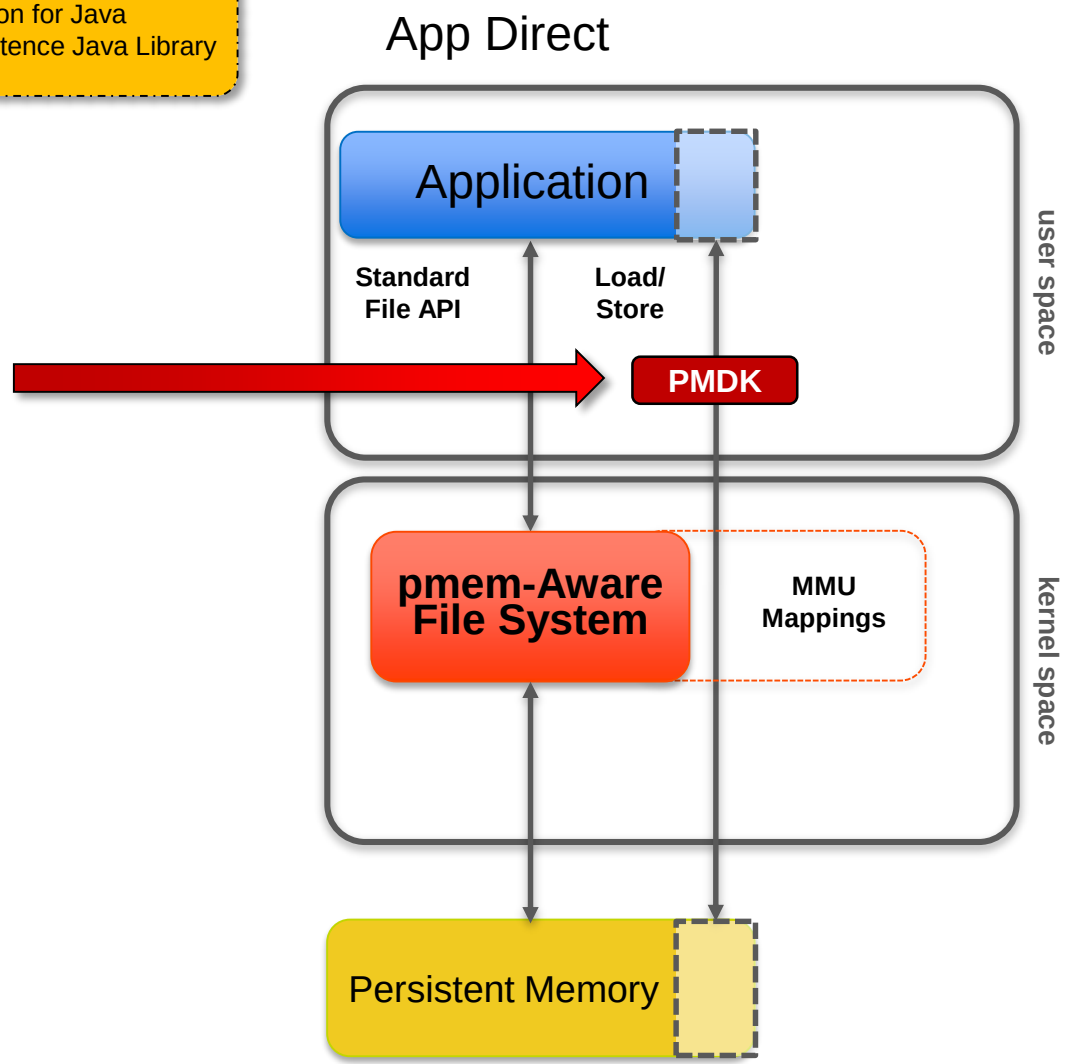
- Interface to create a persistent memory resident log file, e.g. Write Ahead Logging (WAL) — **libpmemlog**
- Interface for persistent memory allocation, transactions and general facilities — **libpmemobj**
- Interface to create arrays of PMEM-resident blocks, of same size, atomically updated — **libpmemblk**

Volatile memory support

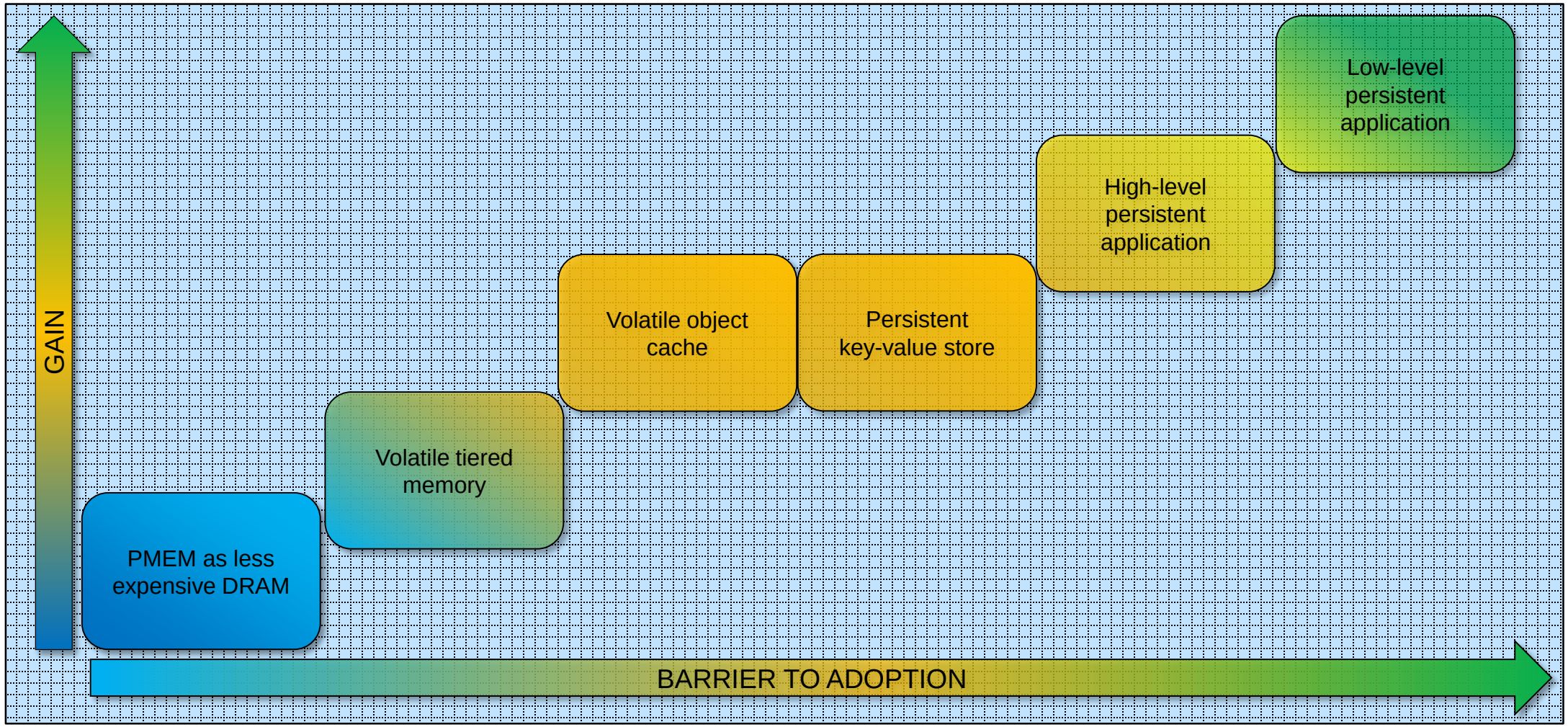
- Support for allocations from DRAM and PMEM — **libmemkind**

Low-level support

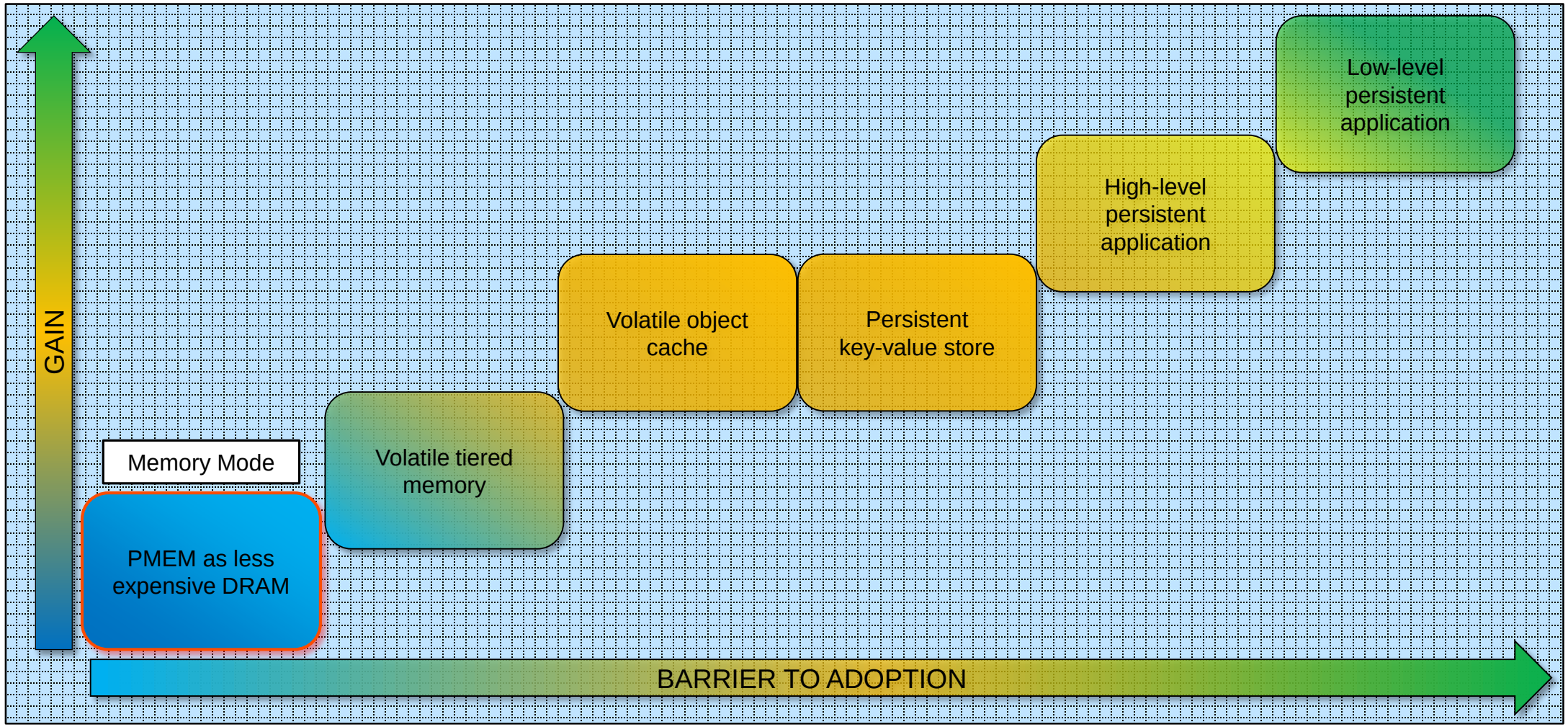
- Low level support for local persistent memory — **libpmem**
- Low level support for remote access to persistent memory — **librpmem**



Different ways to use persistent memory



Different ways to use persistent memory



Memory Mode

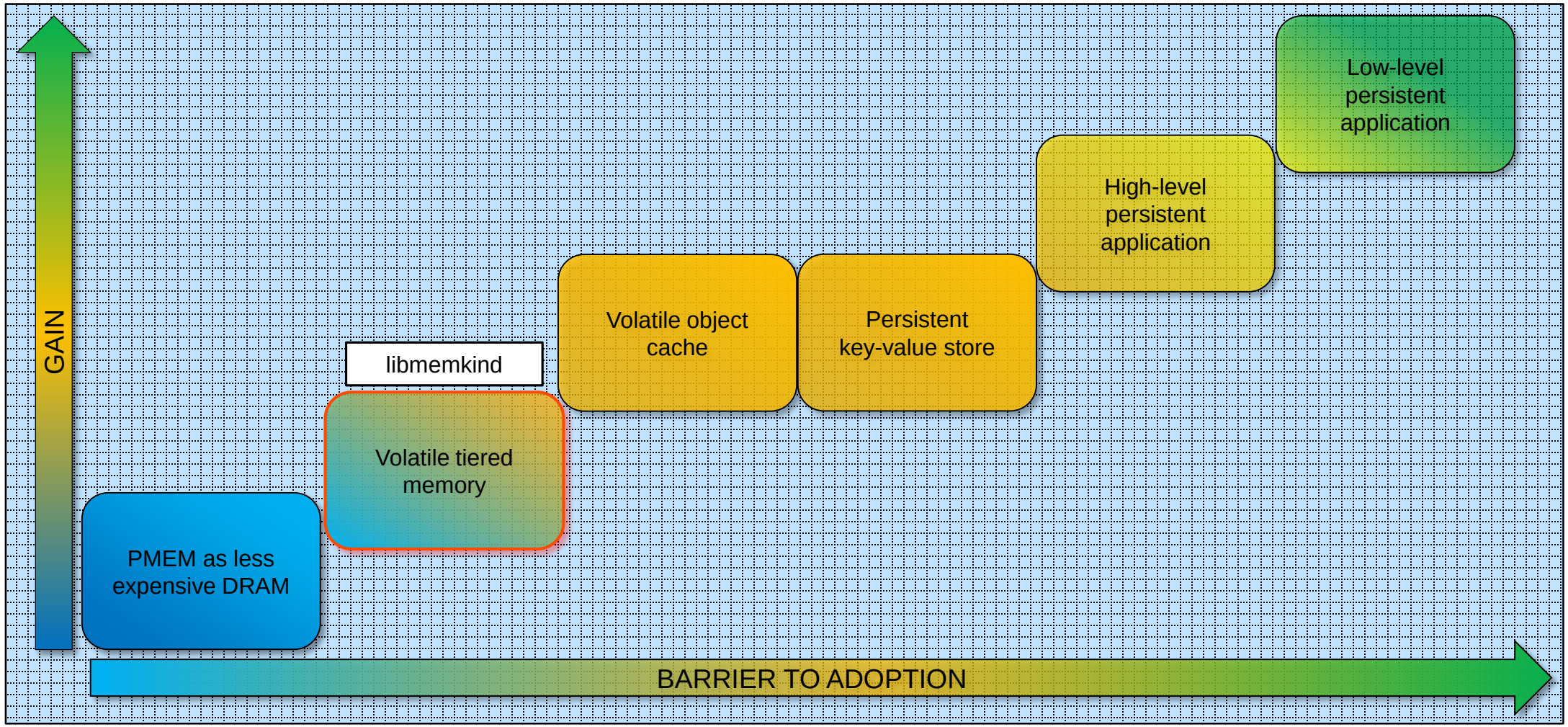
Use when:
- modifying applications is not feasible
- massive amounts of memory is required (more TB)
- CPU utilization is low in shared environment (more VMs)

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

```
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

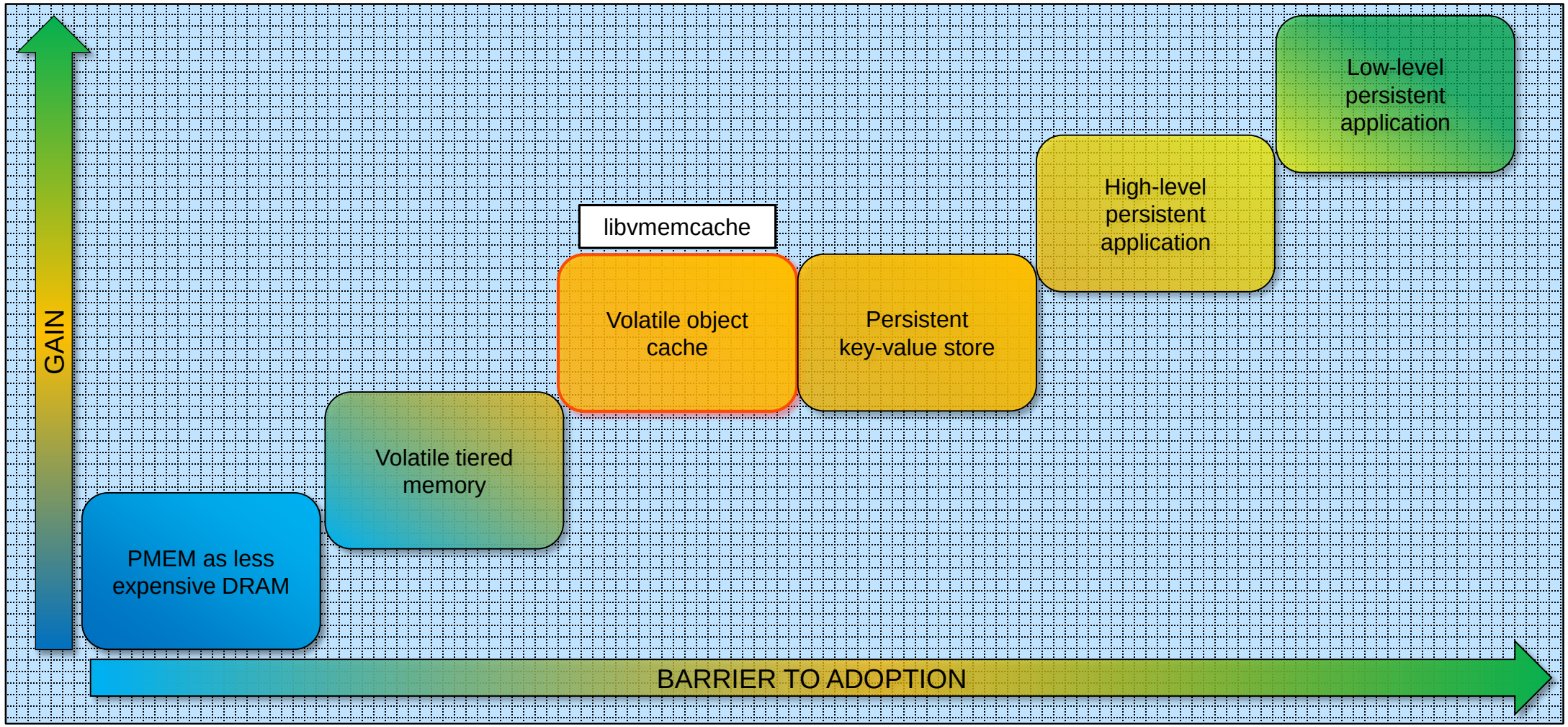
Use when:
- application can be modified
- different tiers of objects (hot, warm) can be identified
- persistence is not required

- Explicitly manage allocations from App Direct, allowing for fine-grained control of DRAM/PMEM

```
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);
```

- The application can decide what type of memory to use for objects

Different ways to use persistent memory



libvmemcache

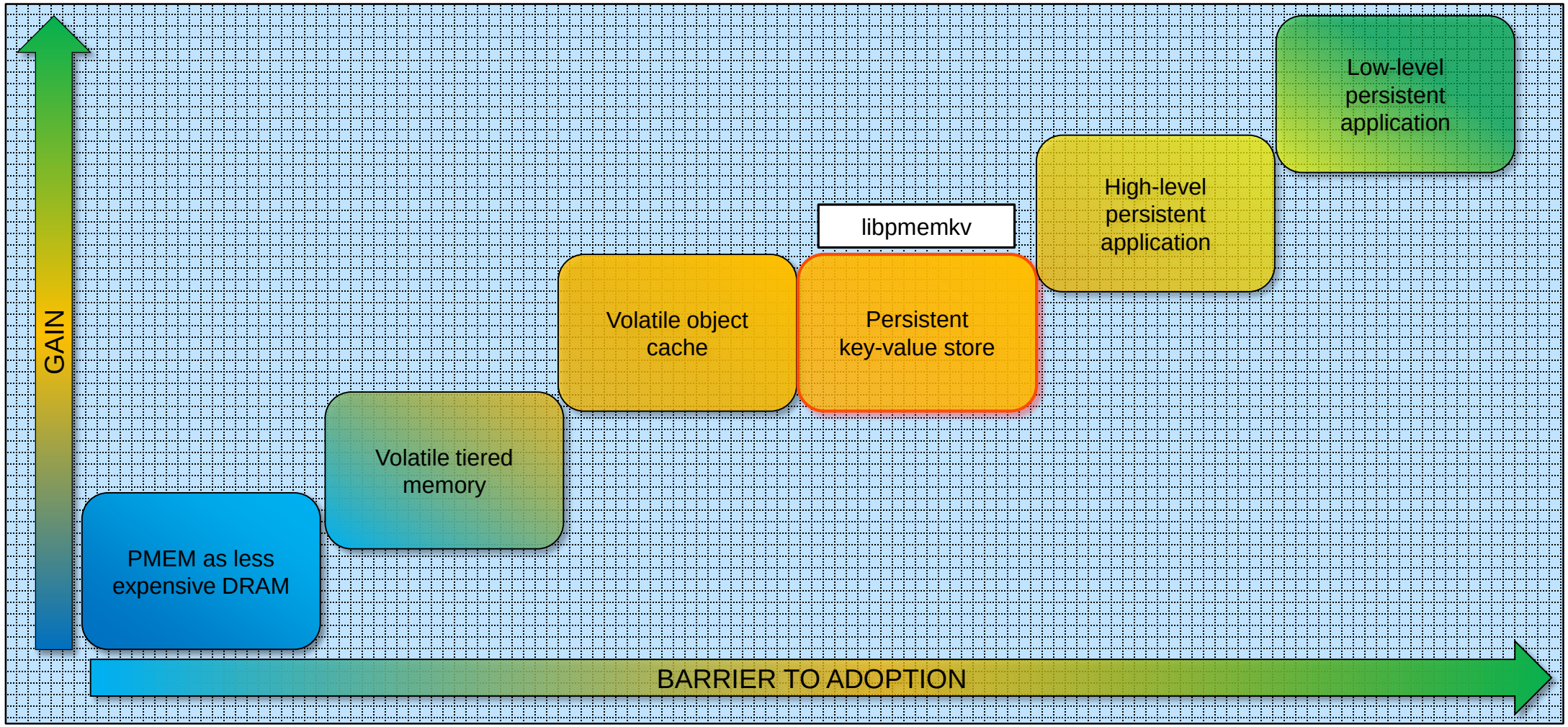
Use when:
- caching large quantities of data
- low latency of operations is needed
- persistence is not required

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

```
VMEMcache *cache = vmemcache_new("/tmp", VMEMCACHE_MIN_POOL,
VMEMCACHE_MIN_EXTENT, VMEMCACHE_REPLACEMENT_LRU);

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

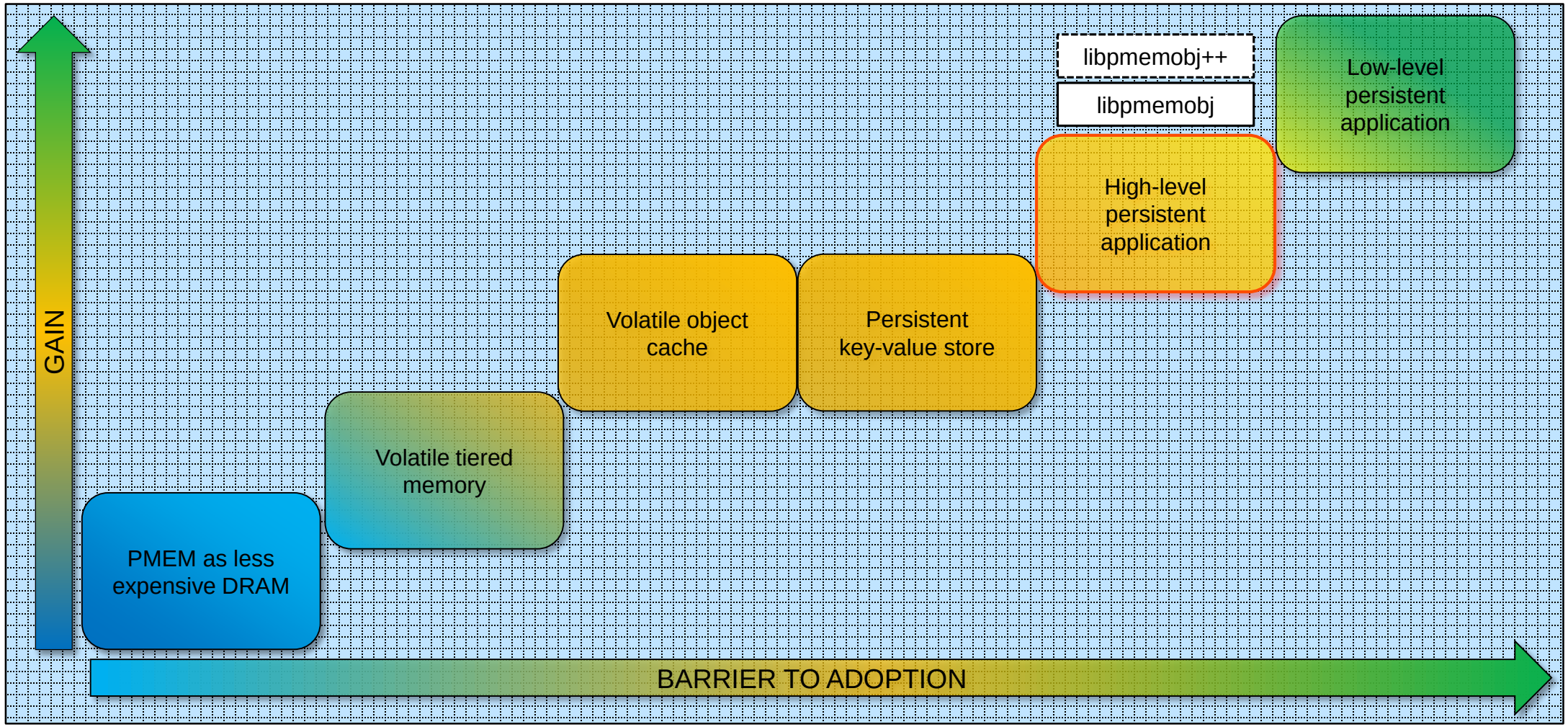
Use when:
- storing large quantities of data
- low latency of operations is needed
- persistence is required

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

```
const pmemkv = require('pmemkv');

const kv = new KVEngine('vsmap', '{"path":"/dev/shm/"}');

kv.put('key1', 'value1');
assert(kv.count === 1);
assert(kv.get('key1') === 'value1');

kv.all((k) => console.log(`  visited: ${k}`));

kv.remove('key1');
kv.stop();
```

- High-level storage layer optimized for PMEM

Different ways to use persistent memory



libpmemobj

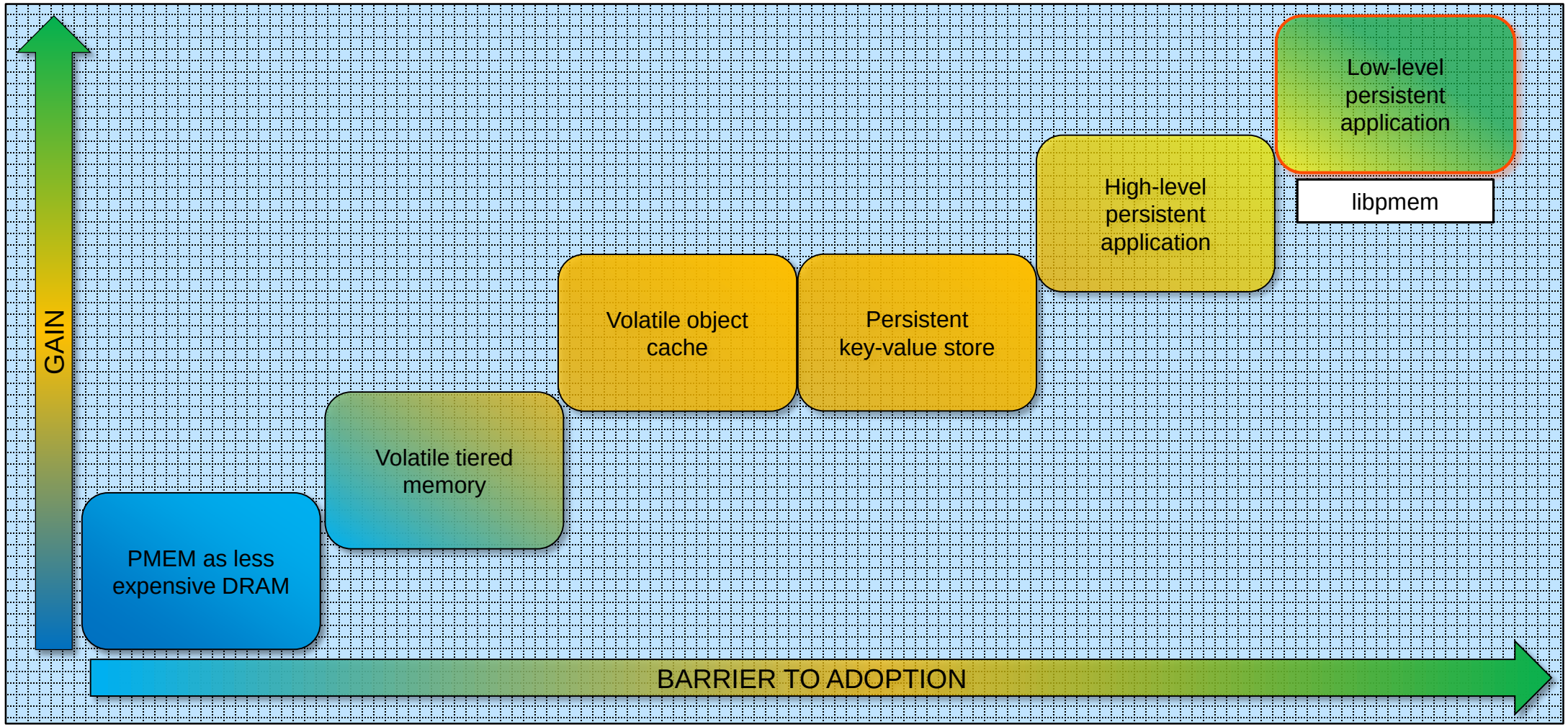
Use when:
- direct byte-level access to objects is needed
- using custom storage-layer algorithms
- persistence is required

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

```
typedef struct foo {
    PMEMoid bar; // persistent pointer
    int value;
} foo;

int main() {
    PMEMobjpool *pop = pmemobj_open (...);
    TX_BEGIN(pop) {
        TOID(foo) root = POBJ_ROOT(foo);
        D_RW(root)->value = 5;
    } TX_END;
}
```

- Flexible and relatively easy way to leverage PMEM

Different ways to use persistent memory



libpmem

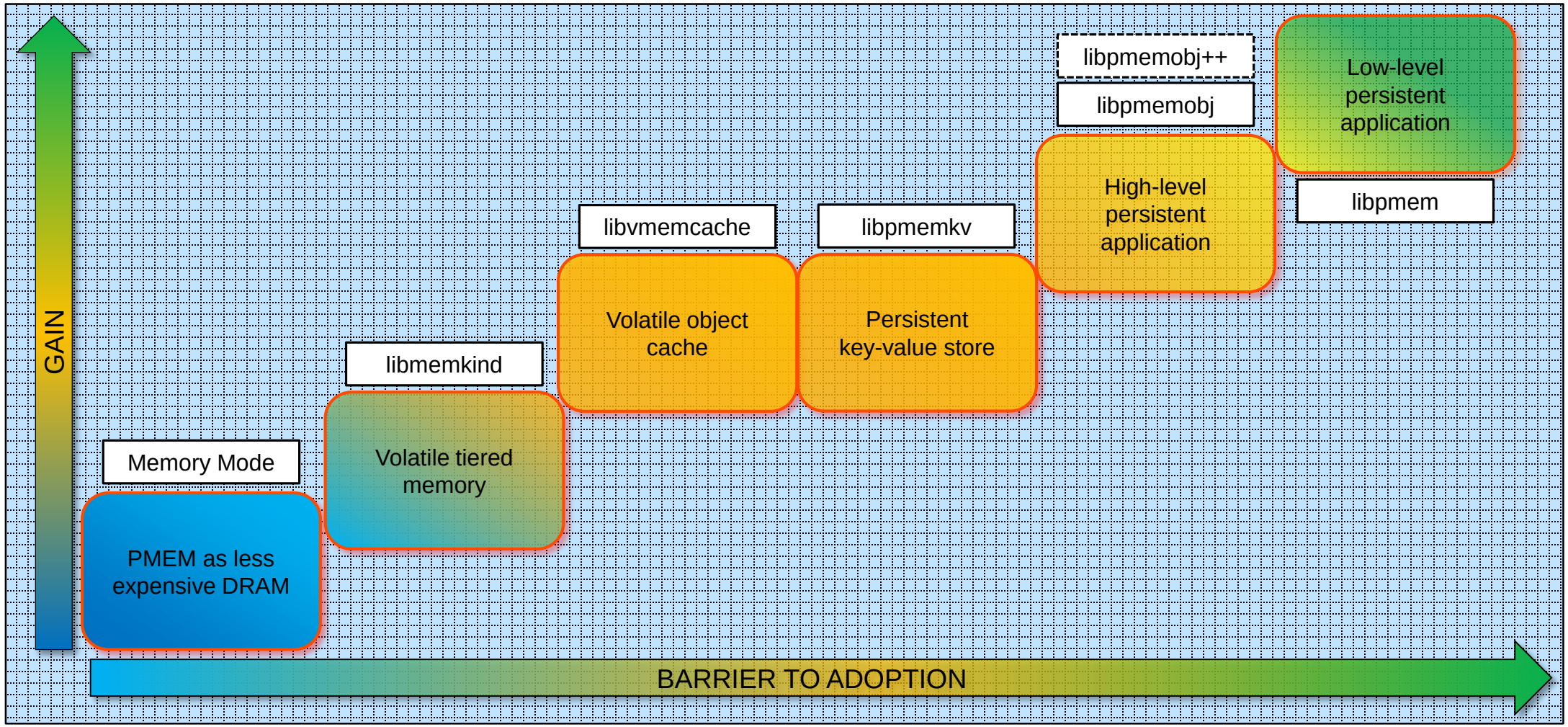
Use when:
- modifying application that already uses memory mapped I/O
- other libraries are too high-level
- only need low-level PMEM-optimized primitives (memcpy etc)

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

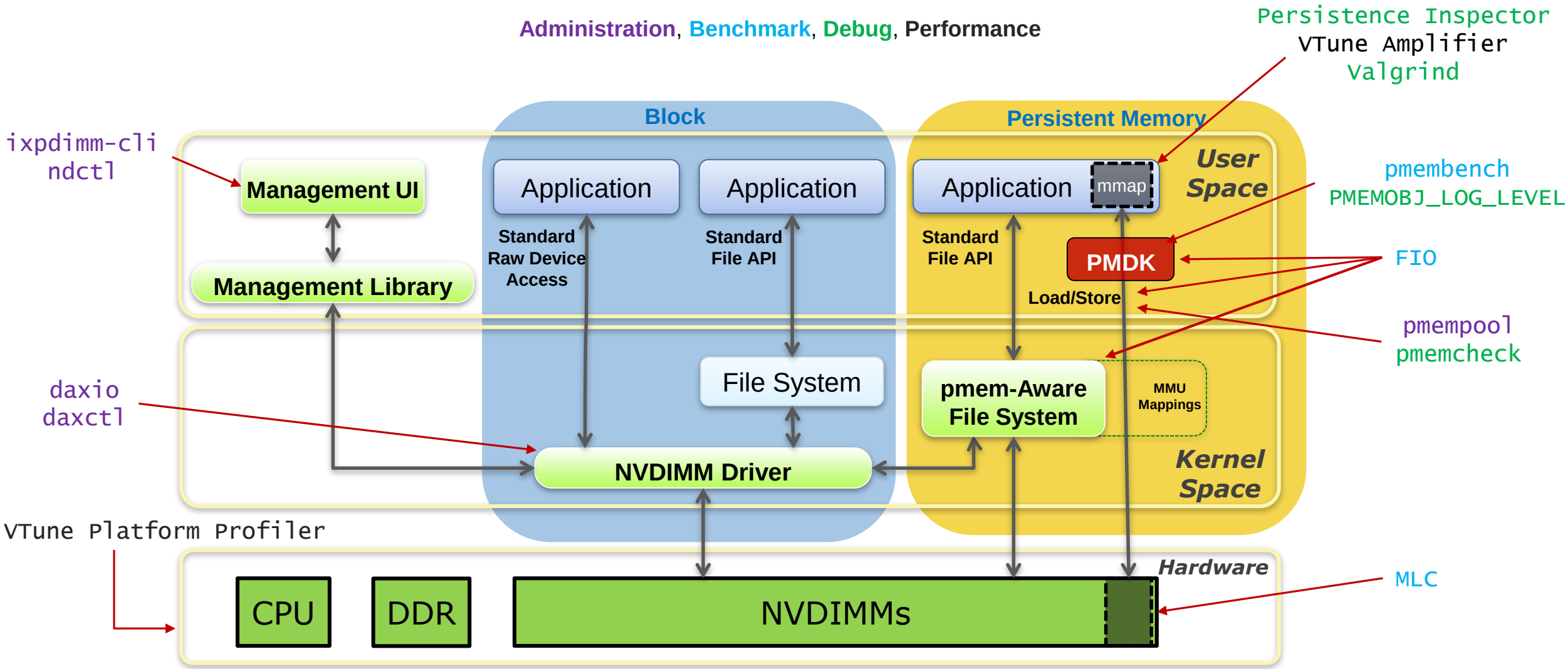
```
void *pmemaddr = pmem_map_file("/mnt/pmem/data", BUF_LEN,
            PMEM_FILE_CREATE|PMEM_FILE_EXCL,
            0666, &mapped_len, &is_pmem));
const char *data = "foo";
if (is_pmem) {
    pmem_memcpy_persist(pmemaddr, data, strlen(data));
} else {
    memcpy(pmemaddr, data, strlen(data));
   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



Summary

PMDK is a comprehensive collection of solutions

- Developers pull only what they need
  - Low level programming support
  - Transaction APIs
- Fully validated
- Performance tuned.

Open Source & Product neutral

More developer resources

Find the PMDK (Persistent Memory Development Kit) at http://pmem.io/pmdk/

Getting Started

- Intel IDZ persistent memory- https://software.intel.com/en-us/persistent-memory
- Entry into overall architecture - http://pmem.io/2014/08/27/crawl-walk-run.html
- Emulate persistent memory - http://pmem.io/2016/02/22/pm-emulation.html

Linux Resources

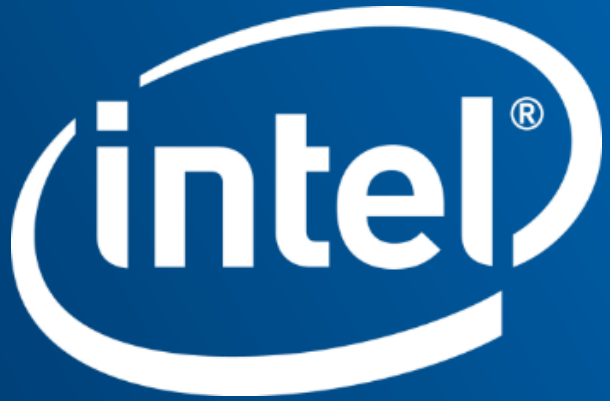
- Linux Community Pmem Wiki - https://nvdimm.wiki.kernel.org/
- Pmem enabling in SUSE Linux Enterprise 12 SP2 - https://www.suse.com/communities/blog/nvdimm-enabling-suse-linux-enterprise-12-service-pack-2/

Windows Resources

- Using Byte-Addressable Storage in Windows Server 2016 -https://channel9.msdn.com/Events/Build/2016/P470
- Accelerating SQL Server 2016 using Pmem - https://channel9.msdn.com/Shows/Data-Exposed/SQL-Server-2016-and-Windows-Server-2016-SCM--FAST

Other Resources

- SNIA Persistent Memory Summit 2018 - https://www.snia.org/pm-summit
- Intel manageability tools for Pmem - https://01.org/ixpdimm-sw/


intel