

# KV Userspace Driver

(OpenMPDK-uNVMe)

Kyungsan Kim / SW Dev Team / Memory Division, Samsung Electronics

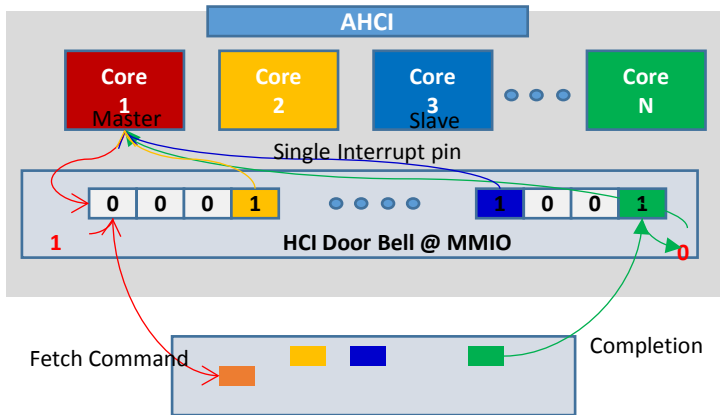
# KeyValue NVMe? Userspace Driver?

# NVMe – High performance storage bus interface

- Scalable storage bus interface over PCI-e controller, emerged in the era of multi core and higher performance SSD

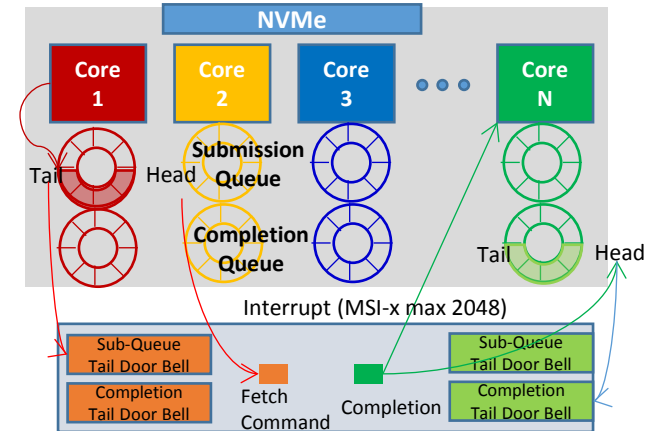
- SATA(AHCI): Master/Slave architecture**

- IO Submission : performance limitation when slave core issue commands
- IO Completion : performance limitation due to Broadcasting interrupt from master to slave core



- NVMe: Parallelism**

- Per CPU Queuing : minimum lock contention / maximum cache hit ratio
- Paired Queuing : IO Submission and Completion Q is separated
- Interrupt balancing with MSI-X : maximum 2048 interrupts can be balanced on all cores



	UFS (AHCI)	SATA (AHCI)	NVMe
Queue : CPU mapping	1:N	1:N	1:1 (per cpu)
Maximum Queue Count	1	1	64K
Maximum Queue Depth	32	32	Max 64K (host/ssd negotiation)
Maximum Interrupt handler	1	1	2048(MSI-X)

# Kernel and Userspace Driver

- POSIX OS consists of kernel sub components such as VM, scheduler, network, file system, and block layer
- Kernel device driver for NVMe storage located beneath block layer

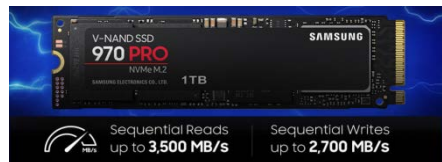
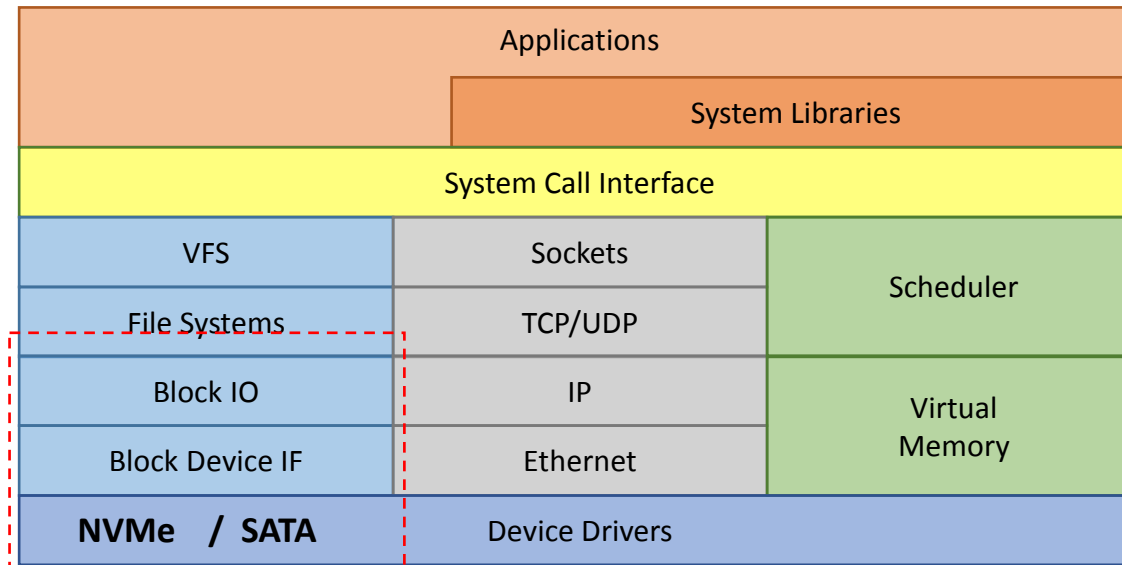


Linux



various distros...

redhat, ubuntu, debian, centos, fedora, suse, and so on

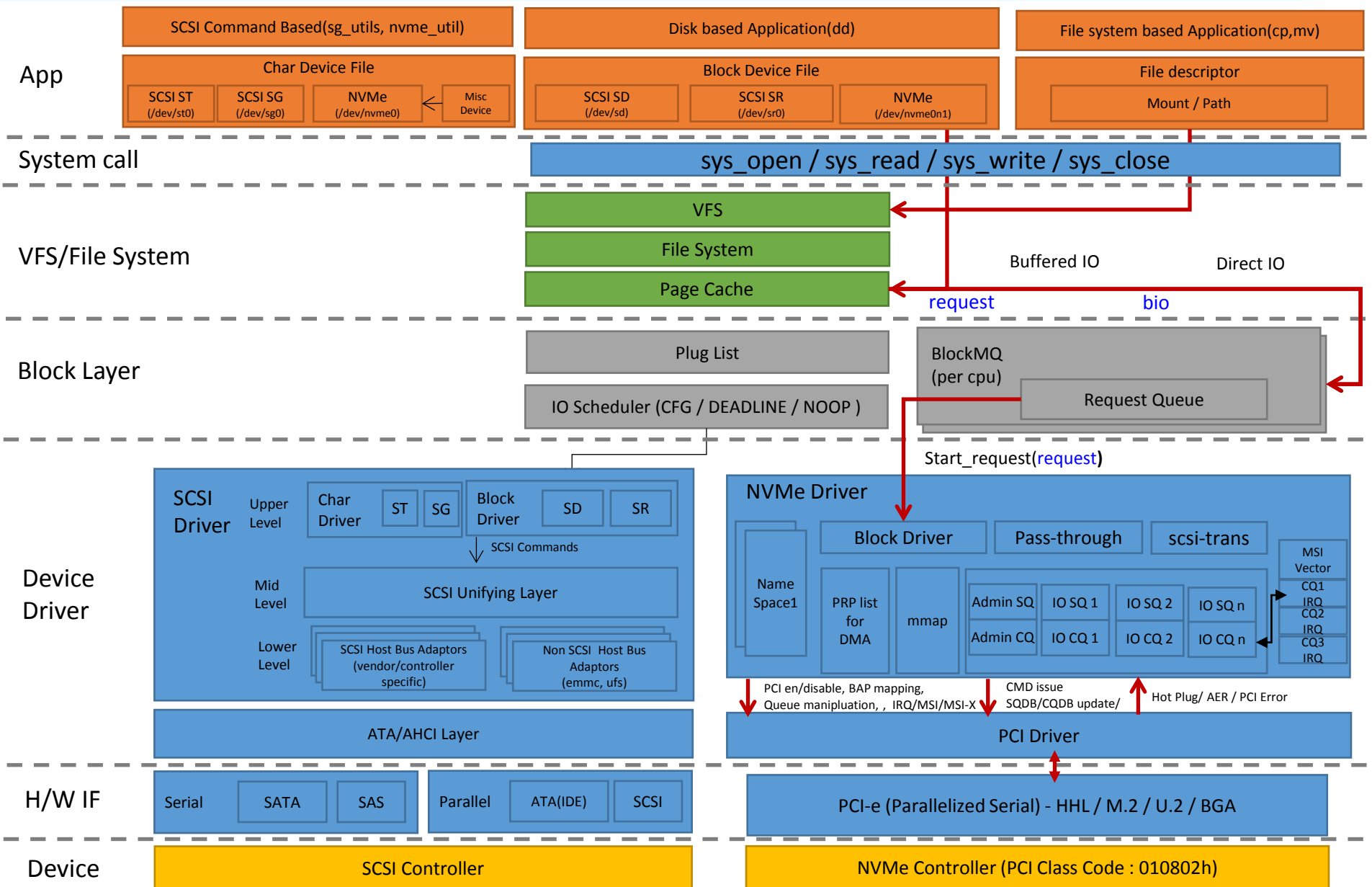


NVMe SSD



SATA SSD

# Kernel Driver - NVMe IO Path

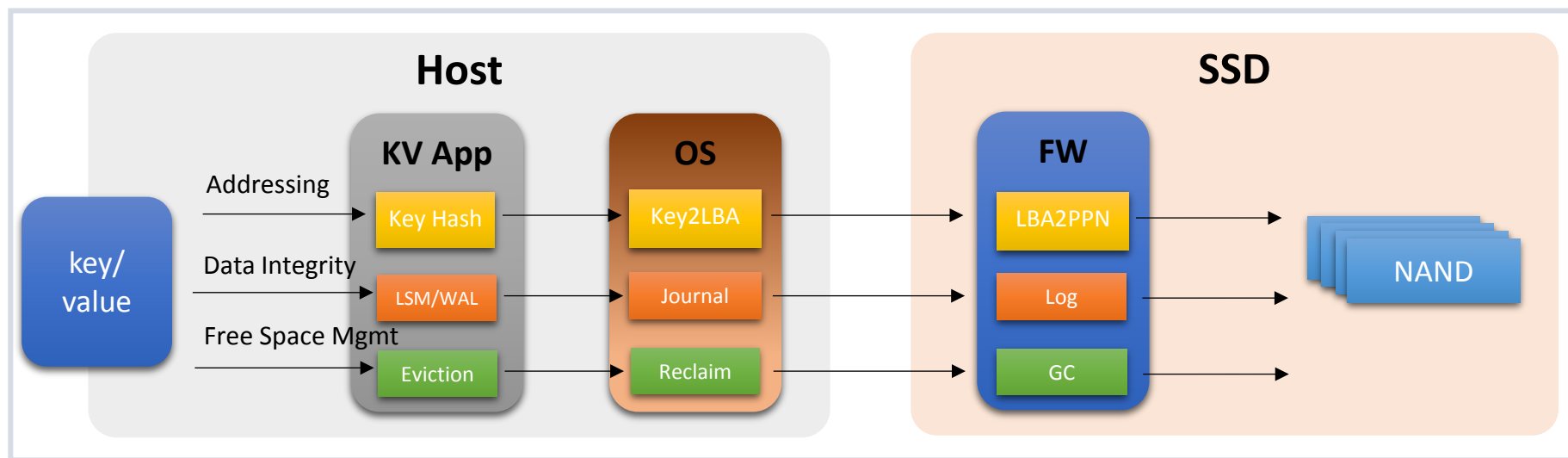


# KV User Driver Background - KV SSD Premise

■ KV SSD benefits are originated from eliminating functional **redundancy** ranging from host application to SSD internal

- **Addressing , Data Integrity, and Free Space Management**

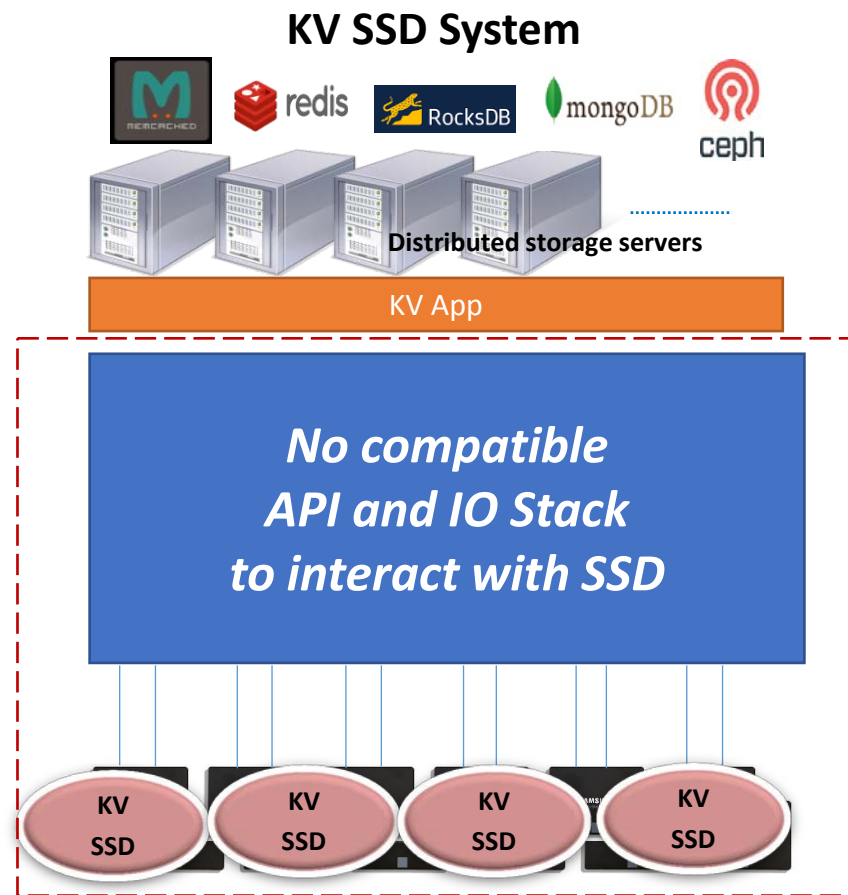
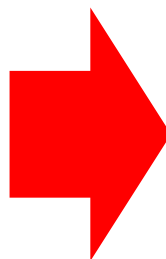
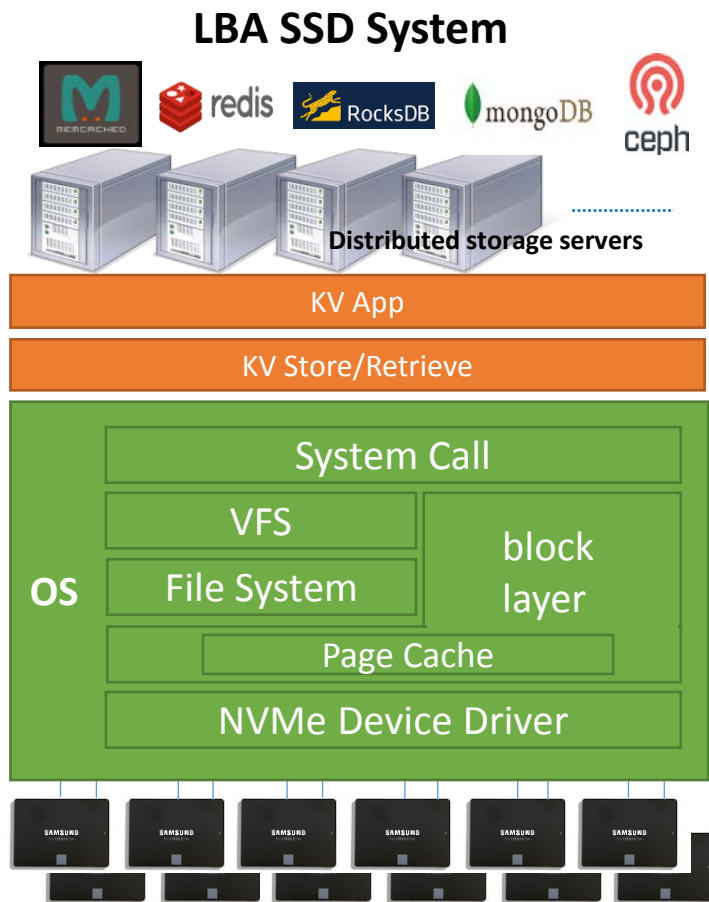
■ Especially, data integrity redundancy causes increased WAF/RAF, which makes SSD longevity worse and app IO latency longer, consuming more host cpu/memory resources



■ KV SSD introduces new NVMe commands such as iterate and interfacing protocol to deal with dynamic size key/value pair via host driver

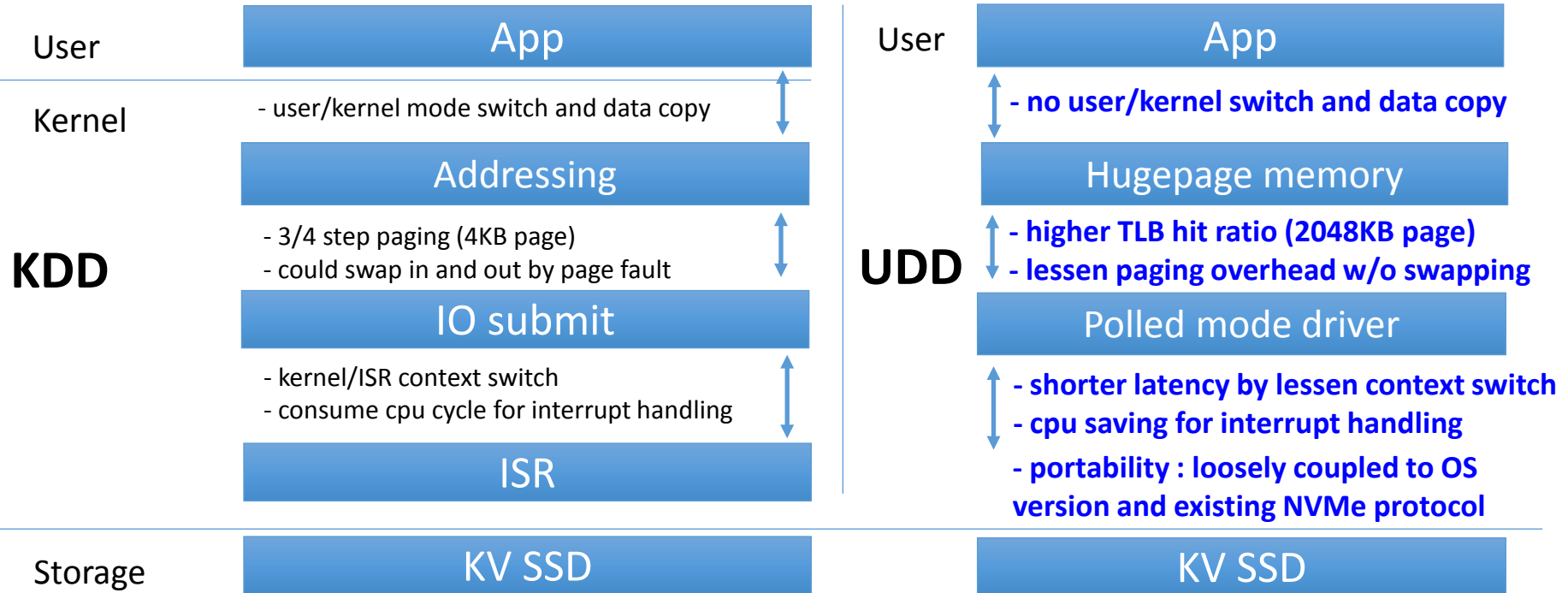
# KV User Driver Background - KV SSD System

■ Accordingly, KV SSD system is designed to bypass typical operating system storage stack as many as possible. Also, the modified storage protocol requires changes in device driver



# KV Userspace Driver: OpenMPDK-uNVM

- **Why Userspace driver?** UDD can provide an optimized IO stack with smaller OS overhead and dependency

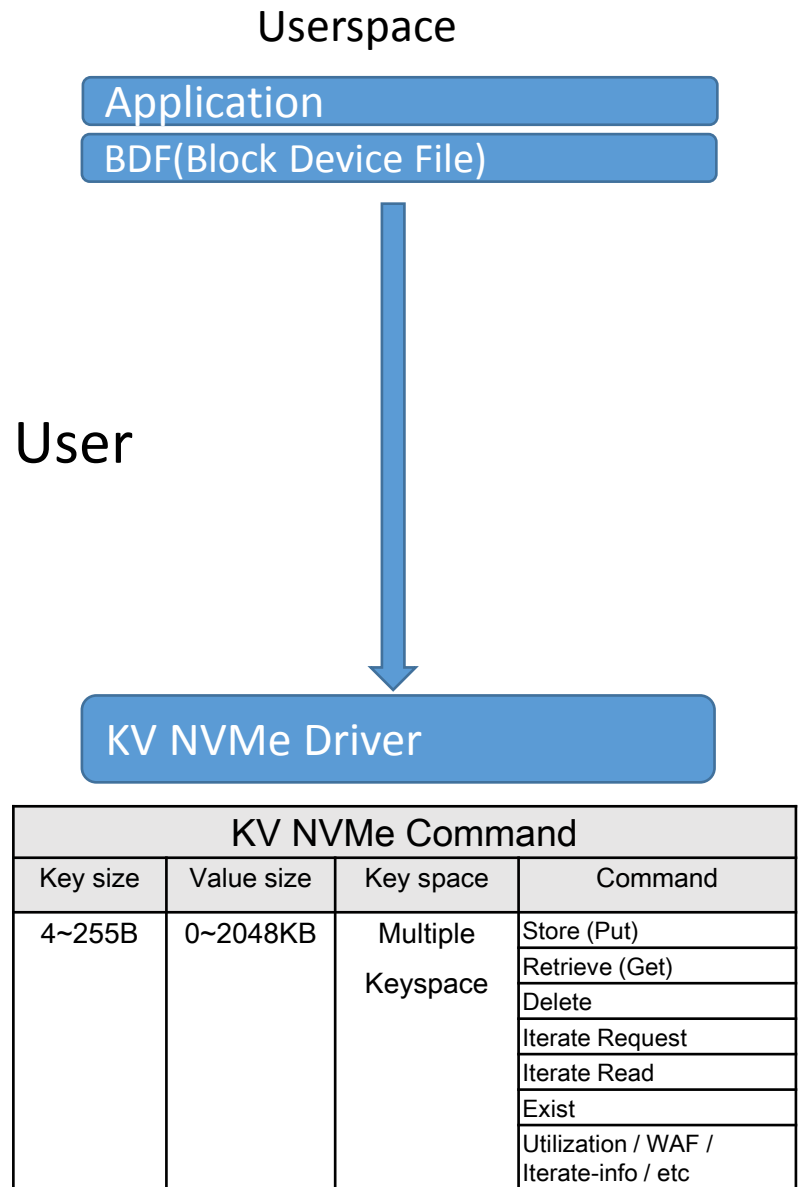
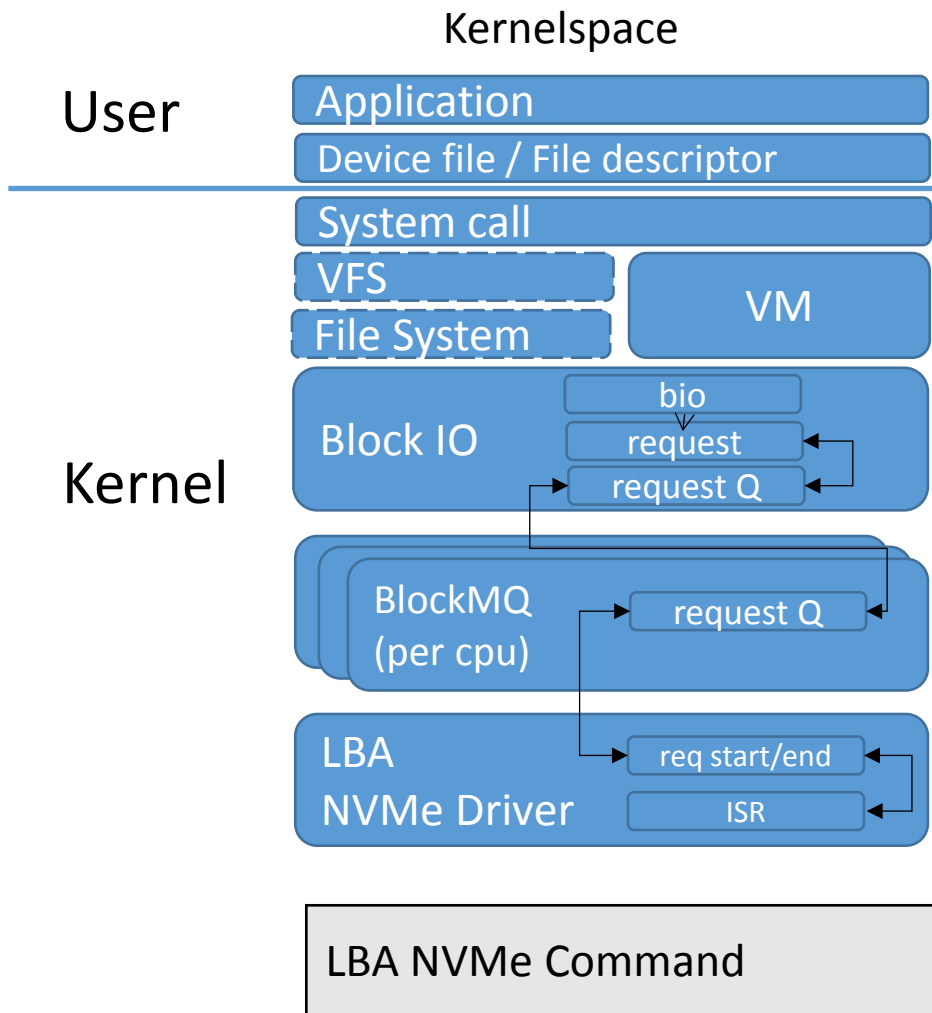


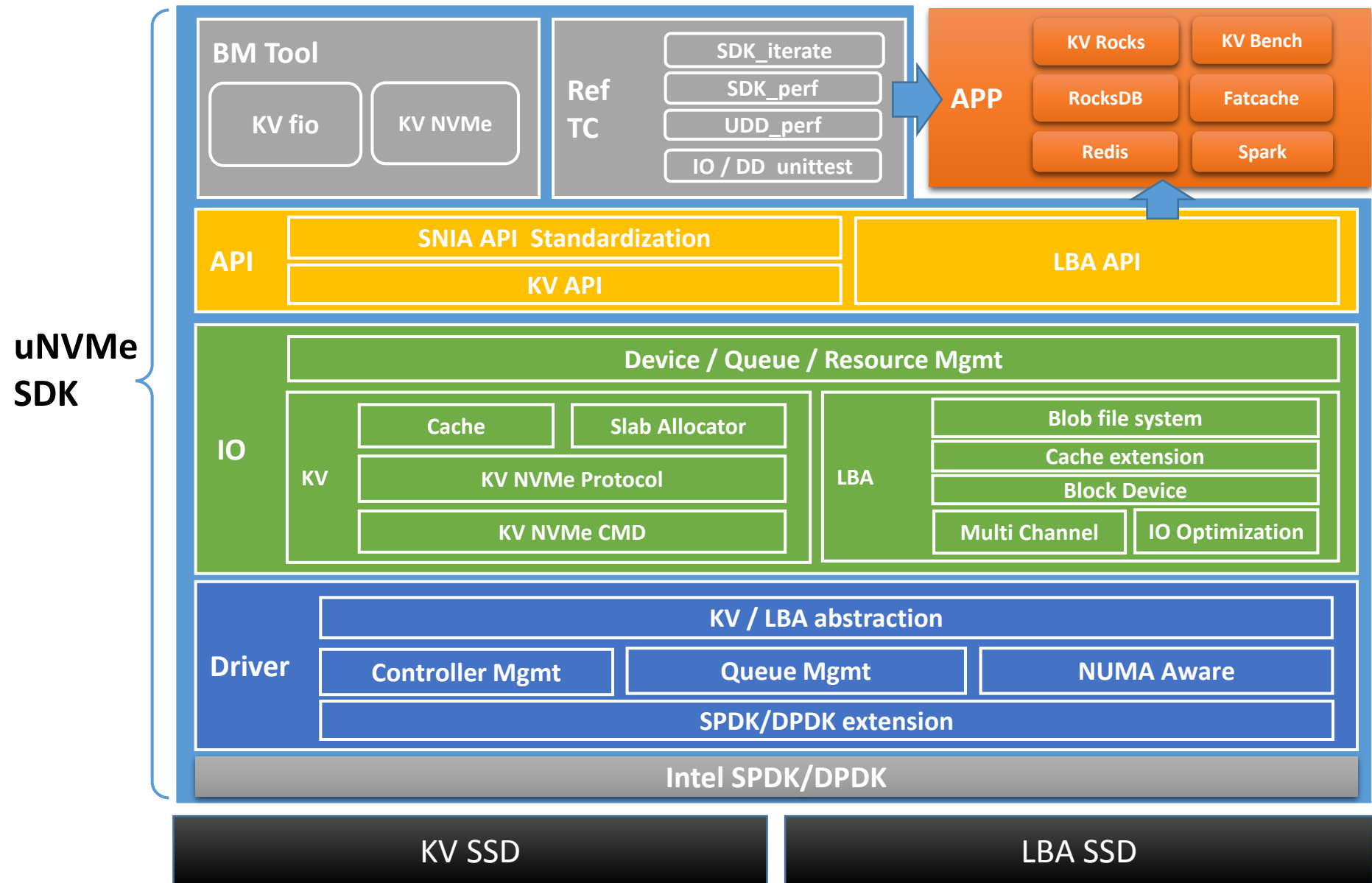
➔ For the reasons, uNVM has been implemented as an extension of *Intel SPDK*. uNVM further provides means to optimize host CPU/DRAM usage and cope with diverse IO characteristics

- Apparently, KDD is worthwhile in that it covers various IO patterns out of real-world applications over commercial systems and a fundamental approach to widespread KV SSD echo systems. Thus, Samsung provides both KDD and UDD



# Kernel / Userspace IO Flow





uNVMe SDK

# Programming Syntax Change(1)

## IO on LBA SSD

Application

```
fd = open("path,,)
```

KV to Buffer

```
write/read(fd, buffer)
```

Buffer to KV

```
close(fd)
```

## IO on KV SSD

Application

```
write/read(KV)
```

# Programming Syntax Change(2)

- uNVM primary API and data structures below
- Please be aware that SNIA API standardized is also provided

## /\*\* SDK init / finalize \*\*/

```
int kv_sdk_init(int init_from, void *option);
int kv_sdk_load_option(kv_sdk *sdk_opt, char* log_path);
int kv_sdk_finalize(void);
void kv_sdk_info(void);
```

## /\*\* KV SSD IO \*\*/

```
int kv_store(uint64_t handle, kv_pair *kv);
int kv_store_async(uint64_t handle, kv_pair *kv);
int kv_retrieve(uint64_t handle, kv_pair *kv);
int kv_retrieve_async(uint64_t handle, kv_pair *kv);
int kv_delete(uint64_t handle, kv_pair *kv);
int kv_delete_async(uint64_t handle, kv_pair *kv);
uint32_t kv_iterate_open(uint64_t handle, const uint8_t keyspace_id, const
uint32_t bitmask, const uint32_t prefix, const uint8_t iterate_type);
int kv_iterate_close(uint64_t handle, const uint8_t iterator);
int kv_iterate_read(uint64_t handle, kv_iterate* it);
int kv_iterate_read_async(uint64_t handle, kv_iterate* it);
int kv_iterate_info(uint64_t handle, kv_iterate_handle_info* info, int nr_handle);
int kv_exist(uint64_t handle, kv_pair* kv);
int kv_exist_async(uint64_t handle, kv_pair* kv);
int kv_format_device(uint64_t handle, int erase_user_data);
```

## /\*\* Utilities \*\*/

```
uint64_t kv_get_total_size(uint64_t handle);
uint64_t kv_get_used_size(uint64_t handle);
uint64_t kv_get_waf(uint64_t handle);
uint32_t kv_get_sector_size(uint64_t handle);
int kv_get_log_page(uint64_t handle, uint8_t log_id, void* buffer, uint32_t
buffer_size);
```

```
/** * @brief A pair of structures of key, value, and kv_param. */
```

```
typedef struct {
    uint8_t keyspace_id;
    kv_key key;
    kv_value value;
    kv_param param;
} kv_pair;
```

```
typedef struct {
    void *key;          /**< a pointer to a key */
    uint16_t length;   /**< key length in byte unit */
} kv_key;
```

```
/** * @brief A value consists of a buffer pointer and its length */
```

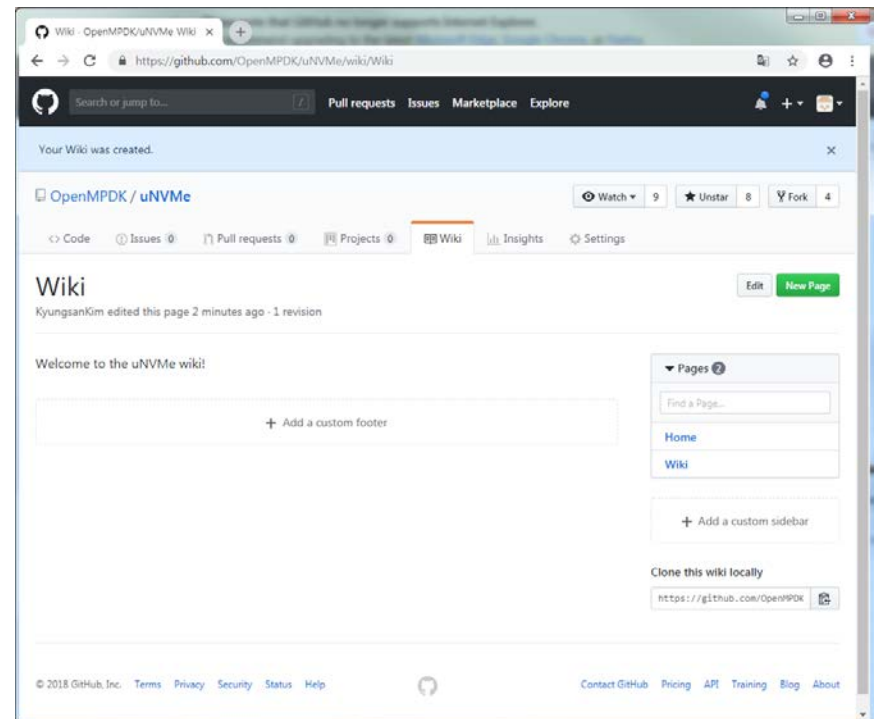
```
typedef struct {
    void *value;       /**< buffer address for value */
    uint32_t length;  /**< value buffer size in byte unit for input and the returned value
length for output*/
    uint32_t actual_value_size;  /* full value size that is stored in disk (only applied on KV
SSD, not on LBA SSD) */
    uint32_t offset;   /**< offset for value */
} kv_value;
```

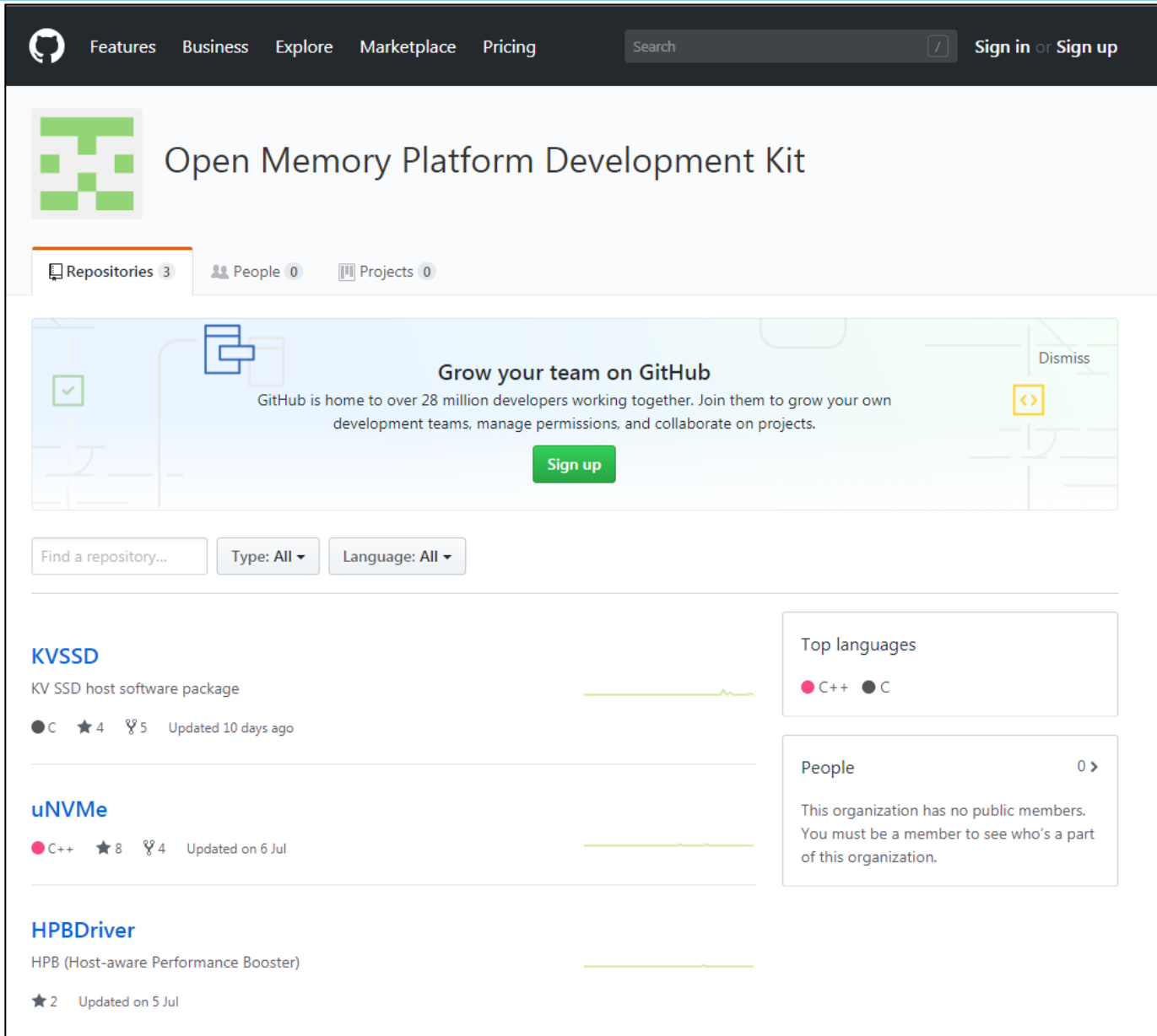
```
/** * @brief A structure which contains I/O option, and callback function(for async I/O) */
```

```
typedef struct {
    void (*async_cb)();  /**< async notification callback (valid only for async I/O) */
    void* private_data;  /**< private data address used in callback (valid only for async
I/O) */
    union {
        int store_option;
        int retrieve_option;
        int delete_option;
        int iterate_request_option;
        int iterate_read_option;
        int exist_option;
    }io_option;          /**< options for operations */
} kv_param;
```

## ■ Guide Document

- uNVMe/doc/uNVMe2.0\_SDK\_Programmung\_Guide\_v1.1.pdf
- uNVMe/doc/uNVMe2.0\_SDK\_Evaluation\_Guide\_v1.1.pdf





The screenshot shows the GitHub repository page for the Open Memory Platform Development Kit (uNVMe). The page features a navigation bar with links for Features, Business, Explore, Marketplace, and Pricing, along with a search bar and a 'Sign in or Sign up' button. The repository name 'Open Memory Platform Development Kit' is prominently displayed, accompanied by a repository icon. Below the name, there are statistics for Repositories (3), People (0), and Projects (0). A large banner encourages users to 'Grow your team on GitHub' with a 'Sign up' button. The main content area includes a search bar for repositories, filters for Type and Language, and a list of repositories. The first repository listed is 'KVSSD', a C-based package for KV SSD host software, with 4 stars and 5 forks. The second is 'uNVMe', a C++ package, with 8 stars and 4 forks. The third is 'HPBDriver', a C-based package for HPB (Host-aware Performance Booster), with 2 stars. On the right side, there are sections for 'Top languages' (C++ and C) and 'People' (0 members).

Features Business Explore Marketplace Pricing Search 7 Sign in or Sign up

## Open Memory Platform Development Kit

Repositories 3 People 0 Projects 0

**Grow your team on GitHub**  
GitHub is home to over 28 million developers working together. Join them to grow your own development teams, manage permissions, and collaborate on projects.  
[Sign up](#)

Dismiss

Find a repository... Type: All Language: All

**KVSSD**  
KV SSD host software package  
● C ★ 4 🍴 5 Updated 10 days ago

**uNVMe**  
● C++ ★ 8 🍴 4 Updated on 6 Jul

**HPBDriver**  
HPB (Host-aware Performance Booster)  
★ 2 Updated on 5 Jul

Top languages  
● C++ ● C

People 0 >  
This organization has no public members. You must be a member to see who's a part of this organization.

# THE NEXT CREATION STARTS HERE

Placing **memory** at the forefront of future innovation and creative IT life

