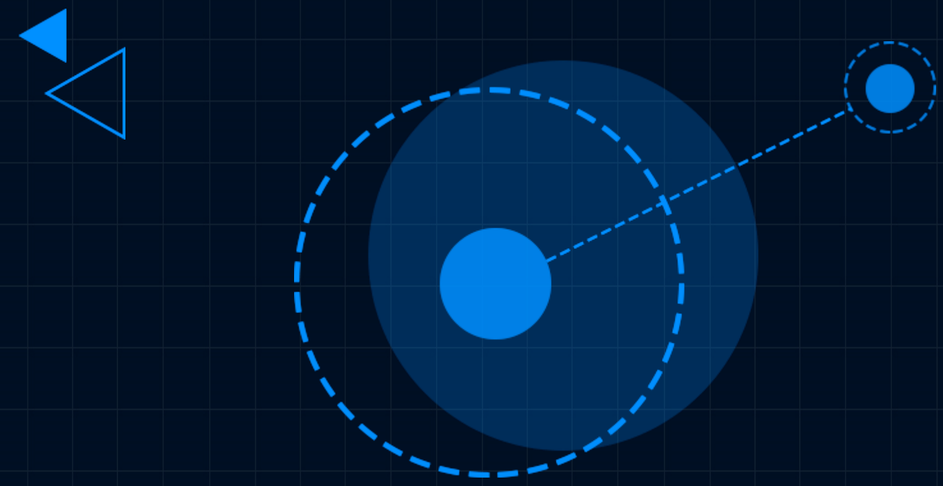
利用SPDK加速框架来加速（持久）内存之间的数据操作

杨子夜
资深云软件工程师，英特尔亚太研发有限公司

Agenda

- **Background & Problem**
- **SPDK acceleration framework introduction**
- **New PMEM bdev (pmem_accel) development status**
- **Conclusion**

Background & Problem



Background & Problem

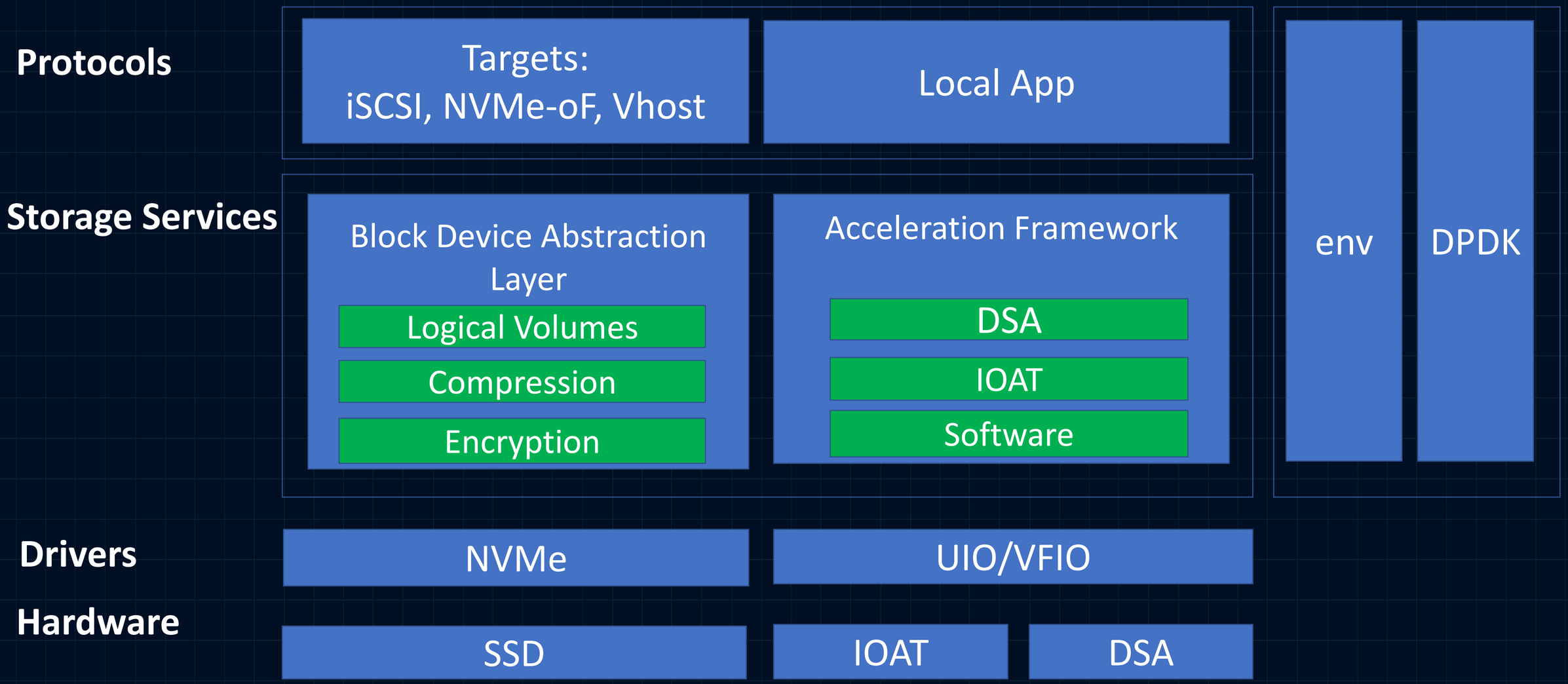
- With more and more deployment of fast NICs and storage devices in data center, there is more and more pressures on CPU, e.g., many users/customers find that CPU becomes a bottleneck to conduct operations between **DRAM** and **PMEM** (persistent memory, e.g., Intel® Optane™ Persistent Memory).
  - Data operations “between (persistent) Memory” is different from “**data operations between Memory and device**”. For the latter case, there is (R)DMA capability on device, and CPU will not be very busy.
- How to offload the work from CPU with some possible memory offloading engines (e.g., IOAT, DSA) is a real and valid requirement.
  - For example, cloud providers would like to utilize the important CPU resources in customer visible workloads (e.g., those host CPUs should be virtualized to launch more customers’ Virtual Machines)

SPDK acceleration framework introduction

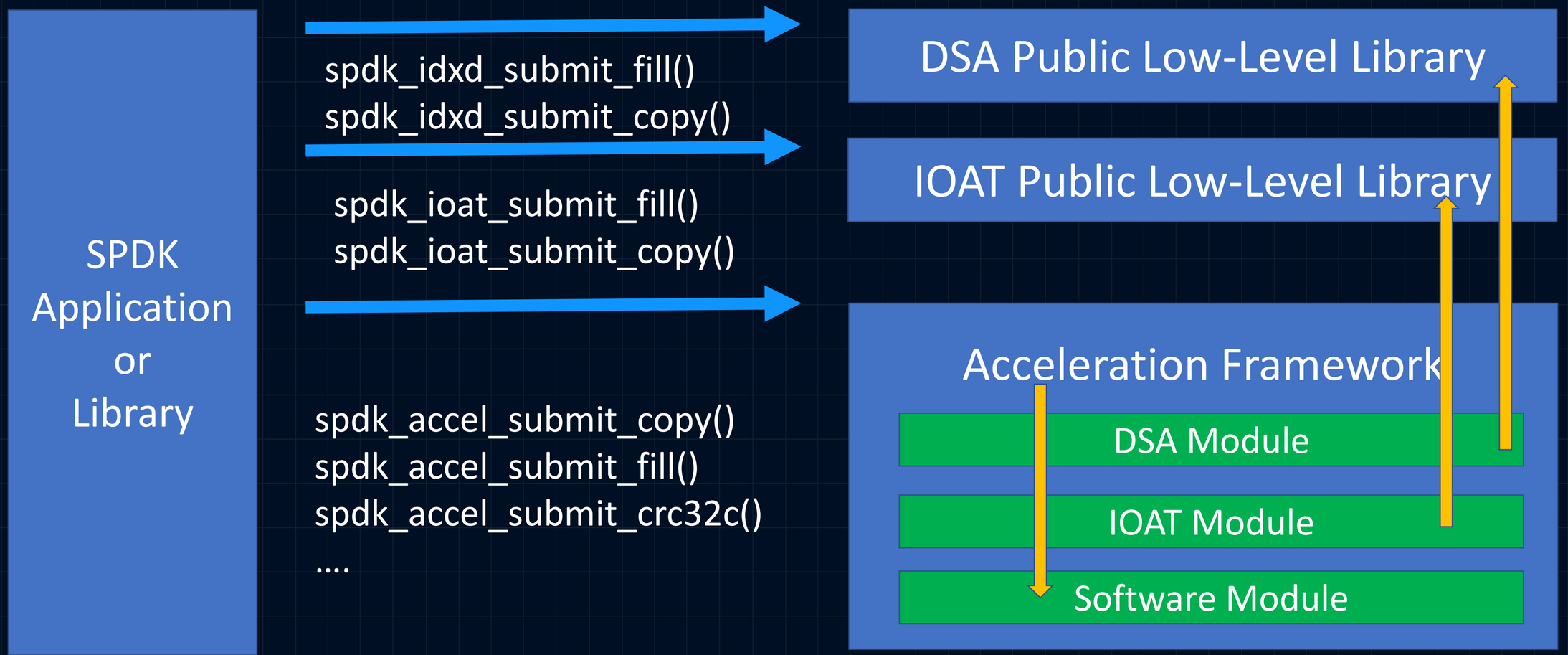
Some memory offloading devices

- Functionality: Support operations between (persistent) memory
- There are some memory offloading engines provided by Intel:
  - Intel IOAT: I/O Acceleration Technology.
  - Intel DSA: Intel® Data Streaming Accelerator (Intel® DSA).
    - Intel DSA is a high-performance data copy and transformation accelerator that will be integrated in future Intel® processors, targeted for optimizing **streaming data movement** and transformation operations common with applications for high-performance storage, networking, persistent memory, and various data processing applications.
    - Supported operations: Mov (e.g., memory move, crc generation…), Fill, Compare, Flush
    - Available on Intel's Sapphire Rapids Platform.
- In SPDK: How to leverage those memory offloading devices?

SPDK high level architecture



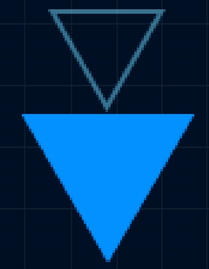
SPDK acceleration framework (spdk accel_fw)

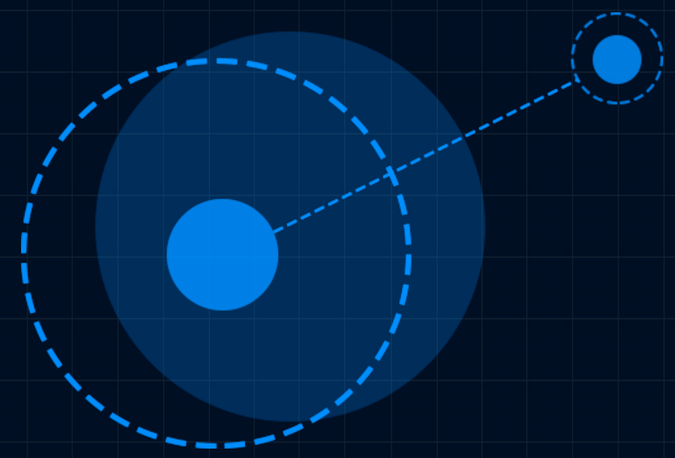


SPDK DSA Feature Support
(Publicly available from 20.10)

- *Copy, Fill, CRC32C, Dual-cast, Compare*
- Optional batching support, application configurable batch sizes
  - (May be deprecated in the future to make batch supporting inside the library)
- Dedicated work queues
- 2 pre-defined group/engine configurations selectable via API, easily changed in code as new/different configs are identified
- All DSA units are ganged together and collectively used, load balanced amongst SPDK threads
- Dynamic load balancing and flow control as threads come and go
- No interrupts or locking
- No dependency on other DSA software

New PMEM bdev (pmem_accel) development status





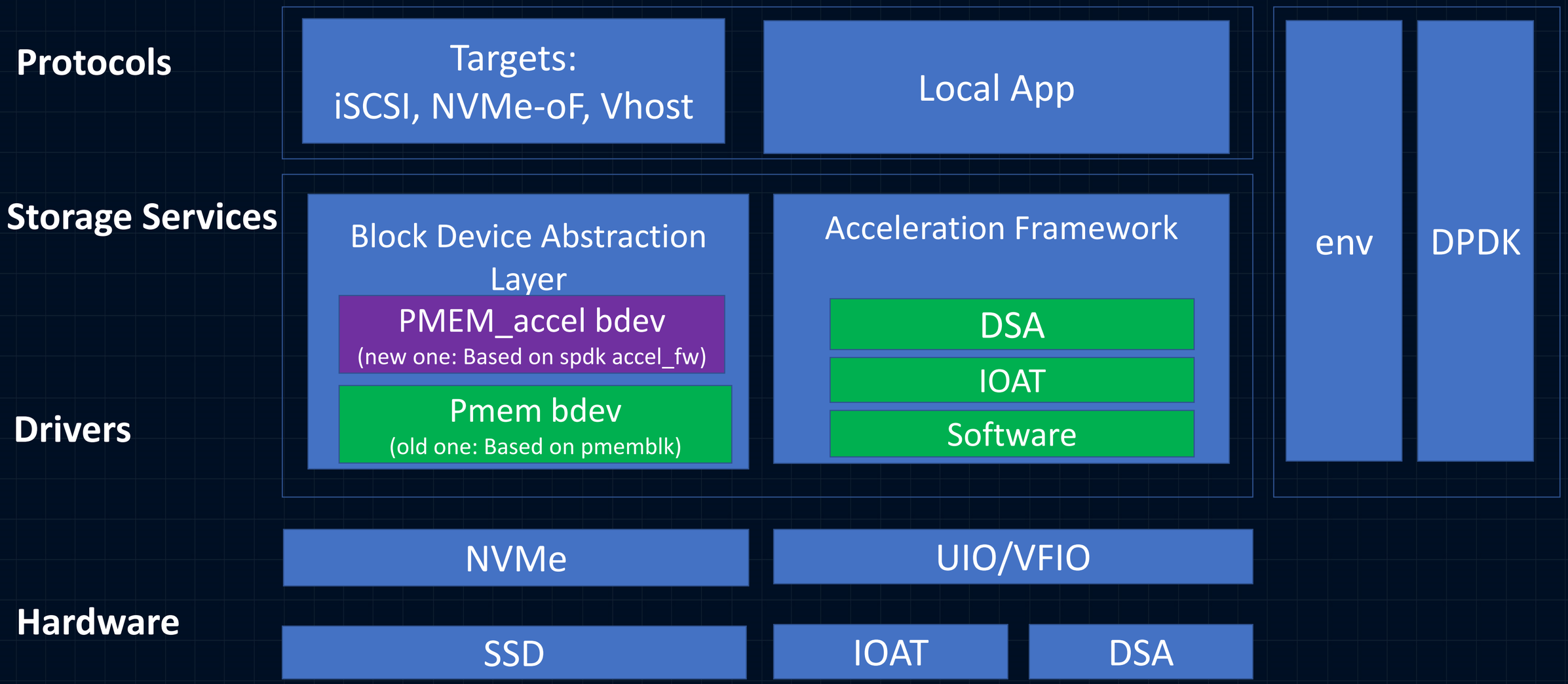
Purpose

- Design and implement a new bdev (named as pmem_accel) based on persistent memory and spdk accel_fw to demonstrate that hardware based memory offloading engine (e.g., IOAT/DSA) can help improve the performance.
- For this new bdev, we will not consider some complicated I/O usage scenarios (e.g., I/O atomicity for large size, sequence order, which should be addressed and guaranteed by upper layer logic).
- Our purpose is to leverage IOAT/DSA in spdk accel_fw to reduce the CPU participation.
  - Ideally, such offloading should be provided by PMDK, but currently PMDK library has some limitations, i.e.,

    1) It only provides the synchronized API interface.

    2) It does not have the ability to plugin memory offloading engines inside PMDK.

PMEM_accel bdev based on PMEM device and spdk accel_fw



Two possible PMEM integration in SPDK to implement pmem_accel bdev

We use PMEM in App Direct (AD) mode for persistency purpose. And the PMEM usage in AD mode can be divided into

- Fsdax mode
  - Format the pmem device with “fsdax” choice. Users get block devices, e.g., /dev/pmem0, and users can create file system (especially kernel supported file system) upon that.
- Dax Mode
  - Format the pmem device with “dax” choice . Then we get char devices, e.g., /dev/dax0.0, then you directly operate on such device.

	FSDAX (mount choice: -o DAX)	**FSDAX**	**DAX**
CPU	Y	Y	Y
IOAT	N	Y	Y
DSA	N	Y	Y

PMEM_accel bdev summary

- **Current decision** (According to analysis in previous page):
  - We choose to use DAX mode directly. But we also want to discuss with customers whether DAX mode can really satisfy customers' requirements. And We provide the ongoing patch.
    - In this situation, file system can not be viewed directly on the char device. The pmem devices can be partitioned into different character devices. A SPDK process will directly use one or several /dev/dax* devices directly.
- Ideally, we think that "*PMEM + FSDAX + -O DAX*" should be the perfect solution.
  - But it requires the kernel support especially in file system in Linux kernel , and it may not be available for a long time.
  - We will not use PMEM + FSDAX, because it can not provide the ***failover*** while using offloading devices.
    - Data path: Cache + DRAM -> BLOCK device -> PMEM

Conclusion

Conclusion

- We realize the overhead/bottleneck of CPU to conduct data operations between (persistent) memory while performing storage workloads.
  - Data operations “between (persistent) Memory” is different from “**data operations between Memory and device**”. For the latter case, there is (R)DMA capability on device, and CPU is not very busy.
  - So, **additional memory operation offloading engines** (e.g., IOAT/DSA) are needed to use in order to save CPU resources.
- We introduce SPDK acceleration framework (*spdk accel_fw*) to leverage different memory offloading engines to address such problems and give the usage on PMEM device.
- Next step: We will continue improving *spdk accel_fw* and leverage it to solve different problems in storage aspect.
- For performance report will be shown in spdk website in the future.

Legal Information

Thanks!

Q&A