Tuning and Programming Data-Intensive Systems with **OX** and **Open-Channel SSDs**

Ivan Luiz Picoli[1,2], Philippe Bonnet[1]

ivpi@itu.dk; phbo@itu.dk

[1]IT University of Copenhagen

[2]Samsung Semi-conductor Division Research (SSDR) Lab

Agenda

- Open-Channel SSDs Primer
- OX Primer
- Hands-on Labs

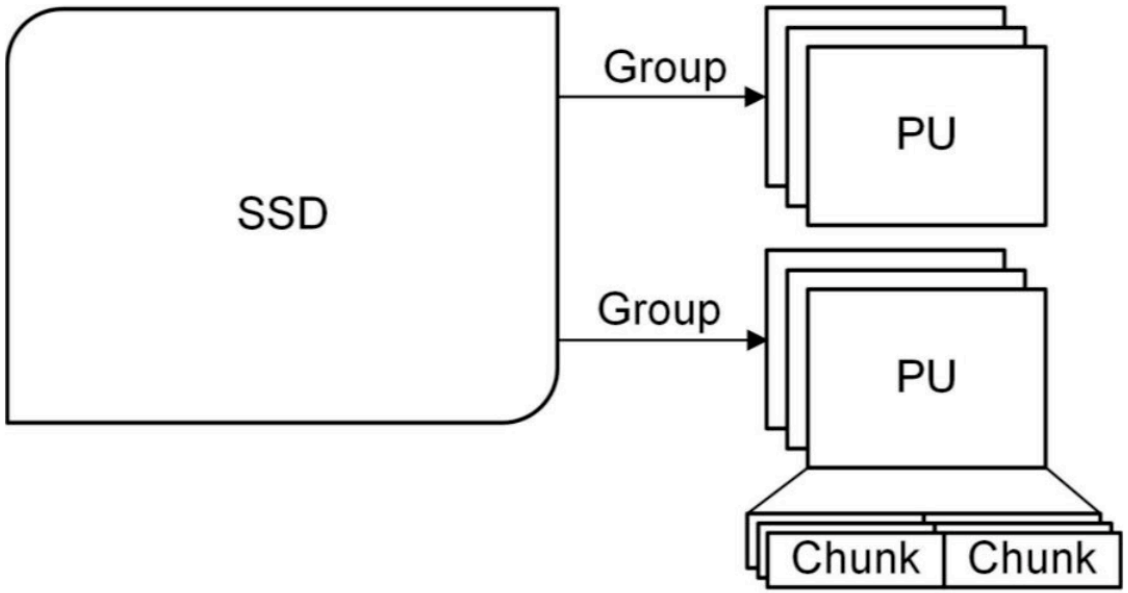
Open-Channel SSD: The Idea

- Physical address space exposed to the host
  - Read, write, erase
  - SSD Parallelism

- Hosts can make decisions about data placement and I/O scheduling
  - SSD management split between
    - Back-end (embedded on SSD): Block meta-data and wear levelling (for warrantee)
    - Front-end (host-based): Flash Translation Layer
      - Mapping of logical to physical address spaces
      - Manage overprovisioning and garbage collection

Open-Channel SSD: The Interface

Chunk Model



Vector Chunk Write/Read:

- Vectored IO
- writes logical blocks sequentially within chunks
- Limited Retry option

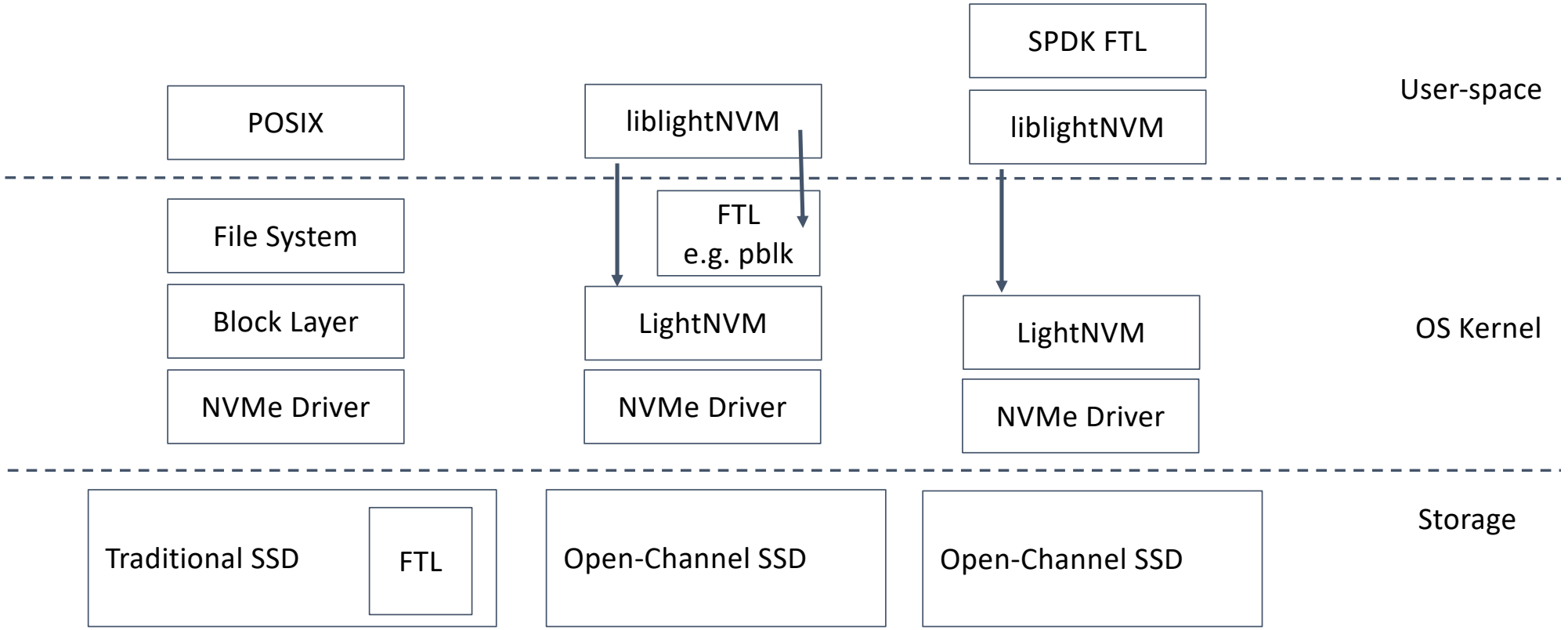
Vector Chunk Erase

Vector Chunk Copy:

- Copies from one set of LBA to another within the device.

Asynchronous Event Info

Open-Channel SSD: The Architecture



Open-Channel SSD: The Potential Impact

WHAT?

1. I/O Isolation
   - Enable host management of device internal resources for contention avoidance
   - Control latency predictability -- beyond NVMe IOD (IO Determinism)

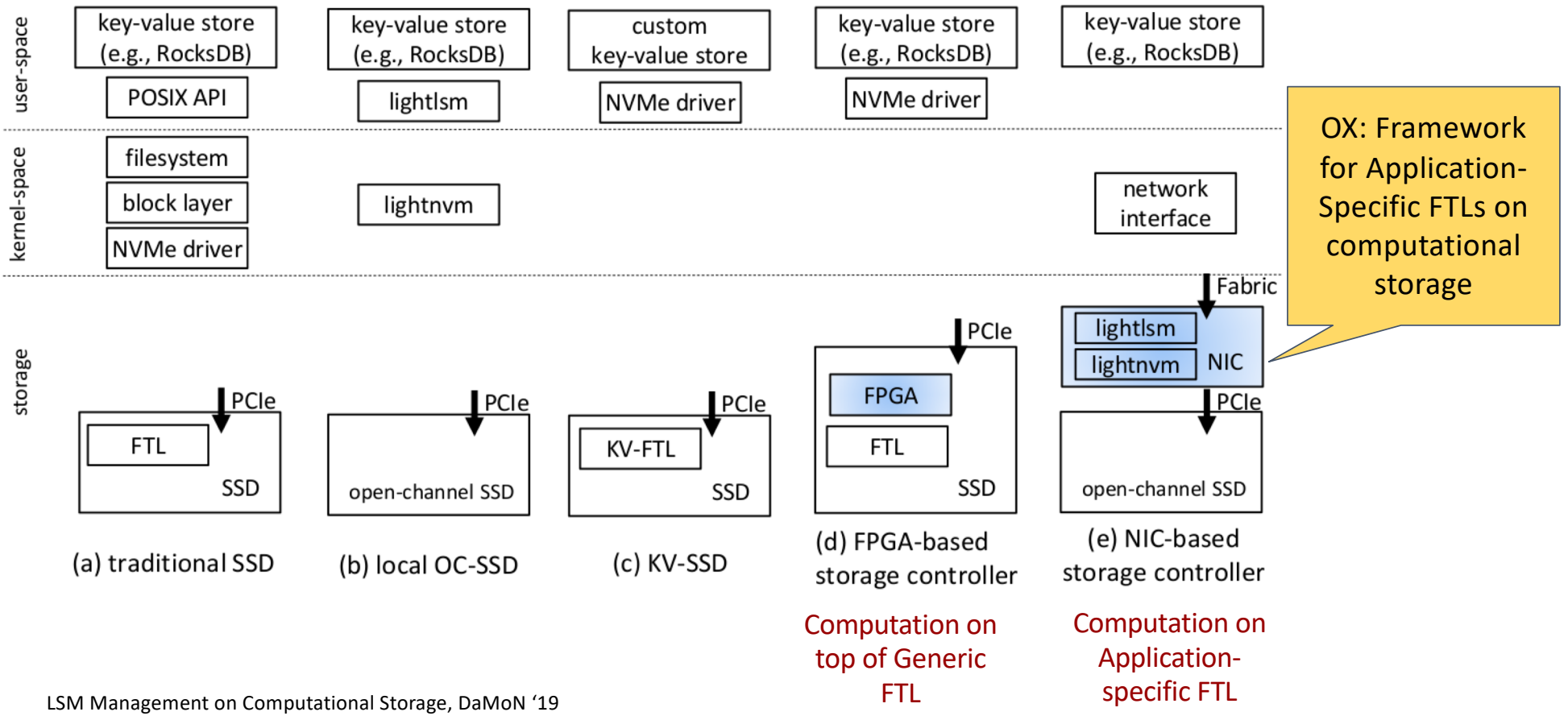
2. Resource Utilization
   - Controlled data placement to reduce Write Amplification (WA) – beyond NVMe streams

3. Streamline data path
   - **Application-Specific FTL**

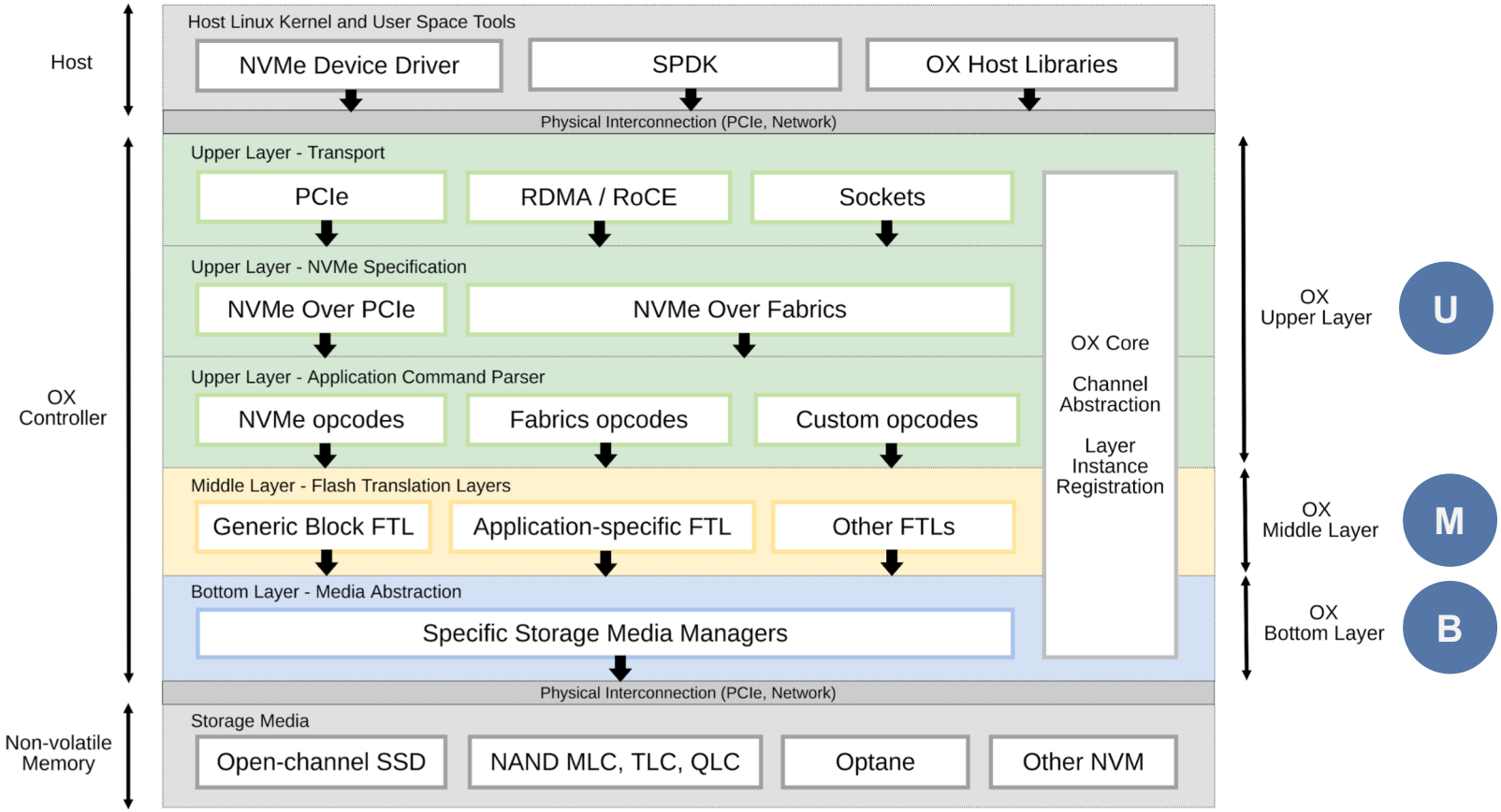
HOW?

Computational storage
   - Offload CPU
   - Shield host application from complexity of managing the physical space (e.g., flash characteristics)
   - Co-design of Application-specific FTL and Open-Channel SSD

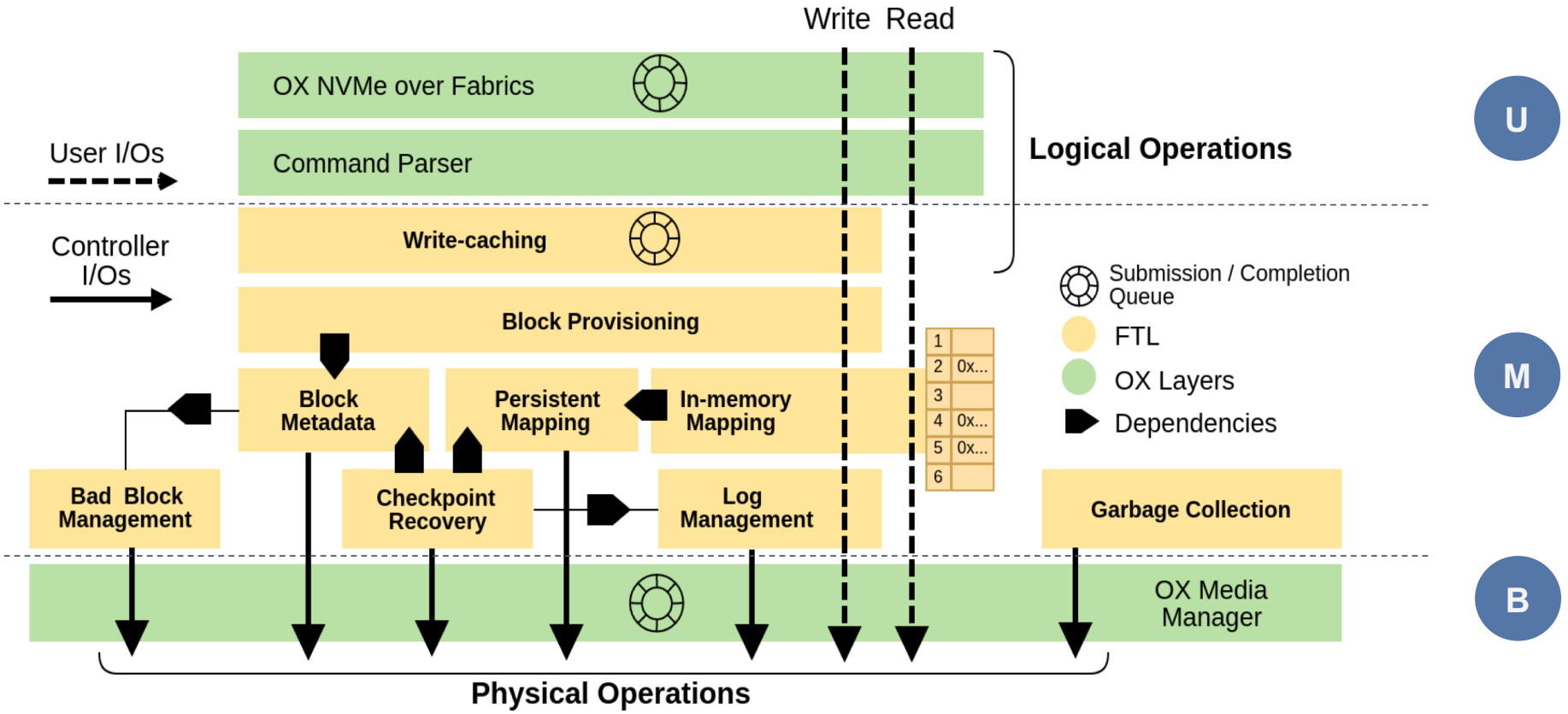
OX: The idea


user-space
kernel-space
storage
key-value store (e.g., RocksDB)
POSIX API
filesystem
block layer
NVMe driver
key-value store (e.g., RocksDB)
lightlsm
lightnvm
custom key-value store
NVMe driver
key-value store (e.g., RocksDB)
NVMe driver
key-value store (e.g., RocksDB)
network interface
Fabric
lightlsm
lightnvm
NIC
PCIe
FTL
SSD
PCIe
open-channel SSD
PCIe
KV-FTL
SSD
PCIe
FPGA
FTL
SSD
PCIe
open-channel SSD
(a) traditional SSD
(b) local OC-SSD
(c) KV-SSD
(d) FPGA-based storage controller
(e) NIC-based storage controller
OX: Framework for Application-Specific FTLs on computational storage
Computation on top of Generic FTL
Computation on Application-specific FTL

OX: The Architecture



OX: The Architecture



OX: Status

- OX v2.6 just released
  - https://github.com/DFC-OpenSource/ox-ctrl/releases/tag/v2.6
- OX developed and tested
  - on DFC (PCIex8 or 40GE, NXP LS2088 Soc (ARMv8) + DDR3 DRAM + M.2 connectors)
    - being ported on Broadcom stingray (100GE with hw ROCE, ARMv8, DDR4 DRAM + PCIex8)
  - With OCSSD Spec v1
- OX equipped with
  - OX-Block: generic FTL in user-space
  - OX-ELEOS: log-structured storage for LLAMA (not public)
  - LightLSM: computational storage for RocksDB (not public)

LSM Management on Computational Storage, DaMoN '19
Improving CPU I/O Performance via SSD Controller FTL Support for Batched Writes, DaMoN'19

Hands-On Lab

- Ubuntu
  - Local install
  - Contact us if you need access to a remote machine

- Dependencies

```
$ sudo apt-get install build-essential
```

  - Ubuntu 16

```
$ sudo apt-get install cmake libreadline6 libreadline6-dev
```

  - Ubuntu 18

```
$ sudo apt-get install cmake libreadline-dev
```

Hands-on Lab #0

Installation:

Possible Ubuntu packages:

```
$ sudo apt-get install cmake libreadline6 libreadline6-dev
```

Install OX:

```
$ git clone https://github.com/ivpi/ox-ctrl.git
$ cd ox-ctrl
$ git checkout ox-public
$ mkdir build
$ cd build
$ cmake -DVOLT_GB=4 .. (up to 32)
$ sudo make install
```

Hands-on Lab #0

Terminal 0:

```
$ cd <ox-ctrl>/build
$ ./ox-ctrl-nvme-volt start

Wait until you see OX startup
Try some commands:
> help
> show memory
> show io
> show gc
> show cp
> debug on
> debug off
> show mq status
> exit
```

Terminal 1:

```
$ cd <ox-ctrl>/build
Write 20000 4K-blocks starting from block 1
$ ./ox-test-nvme-thput-w 1 20000
Read what you wrote:
$ ./ox-test-nvme-thput-r 1 20000
Run again, with 'show io' activated on Terminal 0
```

Hands-on Lab#0

Terminal 0:

```
phbo@ubuntu18 ~/Systems/ox-ctrl/build (ox-public*) $ ./ox-test-nvme-thput-w 1 20000
 100 % - Time 2746.30 ms, written: 78.12 MB, thput: 28.45 MB/s, IOPS: 910.3

 Time elapsed  : 2746.31 ms
 Written LBAs  : 20000 (1 to 20000)
 Written data  : 78.13 MB
 Throughput    : 28.45 MB/s
 IOPS          : 910.3
 Block size    : 4 KB
 I/O size      : 32 KB
 Issued I/Os   : 2500
 Failed I/Os   : 0
```

Terminal 1:

```
OX Controller started succesfully. Log: /var/log/syslog
ox> show io

 Physical I/O count: 7846
   write : 7812    -> user: 5000    meta+gc: 2812
   read  : 34      -> user: 0       meta+gc: 34
   erase : 402


 Data transferred (to/from NVM): 122.59 MB (128548864 bytes)
   data written     :     122.06 MB (127991808 bytes)
      namespace+pad :      78.12 MB (81920000 bytes)
      meta+gc       :      43.94 MB (46071808 bytes)
   data read        :       0.53 MB (557056 bytes)
      namespace+pad :       0.00 MB (0 bytes)
      meta+gc       :       0.53 MB (557056 bytes)


 Namespace blocks (4096 bytes each):
   write          : 20000   ->      78.12 MB (81920000 bytes)
   read           : 0       ->       0.00 MB (0 bytes)

 Metadata blocks (4096 bytes each):
   write          : 11248   ->      43.94 MB (46071808 bytes)
      map (BIG)   : 80      ->       0.31 MB (327680 bytes)
      map (SMALL) : 128     ->       0.50 MB (524288 bytes)
      blk (SMALL) : 9016    ->      35.22 MB (36929536 bytes)
      log (WAL)   : 506     ->       1.98 MB (2072576 bytes)
      padding     : 1294    ->       5.05 MB (5300224 bytes)
      checkpoint  : 160     ->       0.62 MB (655360 bytes)
      reserved    : 64      ->       0.25 MB (262144 bytes)
      other       : 0       ->       0.00 MB (0 bytes)
   read           : 136     ->       0.53 MB (557056 bytes)
      map (BIG)   : 0       ->       0.00 MB (0 bytes)
      map (SMALL) : 0       ->       0.00 MB (0 bytes)
      blk (SMALL) : 0       ->       0.00 MB (0 bytes)
      log (WAL)   : 0       ->       0.00 MB (0 bytes)
      padding     : 0       ->       0.00 MB (0 bytes)
      checkpoint  : 8       ->       0.03 MB (32768 bytes)
      reserved    : 128     ->       0.50 MB (524288 bytes)
      namespace * : 0       ->       0.00 MB (0 bytes)
      other     * : 0       ->       0.00 MB (0 bytes)

 Garbage Collection blocks (4096 bytes each):
   write          : 0       ->       0.00 MB (0 bytes)
      namespace   : 0       ->       0.00 MB (0 bytes)
      map (BIG)   : 0       ->       0.00 MB (0 bytes)
      padding     : 0       ->       0.00 MB (0 bytes)
   read           : 0       ->       0.00 MB (0 bytes)
      namespace   : 0       ->       0.00 MB (0 bytes)
      map (BIG)   : 0       ->       0.00 MB (0 bytes)
      map (SMALL) : 0       ->       0.00 MB (0 bytes)
      blk (SMALL) : 0       ->       0.00 MB (0 bytes)
      log (WAL)   : 0       ->       0.00 MB (0 bytes)
      padding     : 0       ->       0.00 MB (0 bytes)
      unknown     : 0       ->       0.00 MB (0 bytes)
```

Hands-on Lab #1

Experiment with different types of Open-Channel SSDs:

- Same workload
- Different Volt setups
  - Different SSD topologies (nb channels, nb LUN/channel)
  - Different latency characteristics
  - Different storage chip characteristics (nb planes)

Hands-on Lab #2

Collection of test programs as entry points to OX framework

- test-connect.c
- test-nvme-rw.c
- test-nvme-thput-r.c
- test-nvme-thput-w.c
- test-ox-mq.c
- test-queue.c

Experiment with NVMe submission/completion:

- Modify test-nvme-rw.c to submit 10 random writes
- Hint: check how test-nvme-thput-w.c handles sequential writes

Experiment with OX API:

- Add NVMe command (e.g., swap-lbas)
  - Extend parser (nvme_parser) to create the command that accesses and modifies the mapping table (ftl) and returns completion
- Write test program for the new command