Data Placement at Scale

Landscape, Trade-Offs, and Direction

Javier Gonzalez | LPC'24

Data Placement

- **Data placement is an unsolved problem in storage**
  - Reduce TCO & Improve WAF, Space Amplification, and QoS
  - Goal is to group data
    - Data placement as a logical concept
    - No intention to expose physical layout to host
  - Support increased media density (NAND &* rotational media)
    - Deal with less reliability
    - Provide appealing DWPD
- **Novel Data Placement technologies are hard to adopt**
  - Changes required at the interface / protocol level
    - Changes to *standards* are required
    - Changes to the *open-source* host software are required
    - *Community* Industry leadership to make technology mainstream is required
    - These changes are hard
      - The require a clear use-case accepted by the industry
      - They take time and effort
  - HW/SW Co-Design is difficult to implement
    - Requires tight vendor / customer collaboration

History: Data Placement Technologies

Confidential

- **Data Placement is a prevalent problem across storage consumers & Industries**
  - Impacts: WAF, TCO, predictability (latencies), and overall performance
- **Several approaches in the past few years account for innovation in this area**
  - Well explored design space facilitates a good understanding of the trade-offs

Block Interface (CNS)

Description & Trade-offs

- Traditional block device
- Most innovation in-device
- Mature host software stack
- WAF ~1: No assurances

Status

- Commonplace for storage devices

1970's

Streams / Directives

Description & Trade-offs

- Extension to block device with backwards compatibility
- Use of write hints
- DSM deallocation mechanism
- Minor changes to Host SW
- WAF ~1: Initialize & trust

Status

- Little industry traction

SCSI: 2010's
NVMe: 2017

Flexible Data Plac. (FDP)

Description & Trade-offs

- Extension to block device with backwards compatibility
- Capacity-based placement without seq. write requirement
- Use of write tags
- DSM deallocation mechanism
- Minor changes to Host SW
- WAF ~1: Iterative query/check

Status

- Customer-driver technology
- Blooming adoption & Eco

NVMe: 2023

Zoned Storage

Description & Trade-offs

- Departs from block device
- No backwards compatibility
- LBA-based placement with strict seq. write requirement
- Explicit host deallocation & state machine management
- Major changes to Host SW
- WAF ~1: assured

Status

- Adopted in HDDs (SMR)
- Traction in UFS
- Little traction in NVMe

SCSI: 2018
NVMe: 2021

Open-Channel SSDs

Description & Trade-offs

- Full host-based FTL
- Most innovation in host
- Drastic changes to Host SW
- WAF ~1: Assured

Status

- Not standardized
- Dropped by the industry

NVMe: 2015

WAF: One Proxy-Metric to Rule Them All

Confidential

- **The goal is to compare Data Placement (DP) technologies across these 4 metrics**
  - WAF (+Space Amp), Performance (BW & latency), Device Utilization, and Engineering Effort
  - TCO is a function of *all* these metrics

Technology Viewpoint

- **How is each customer dealing with the trade-off across these metrics?**
- **How much can we push device utilization at acceptable metrics?**
- **Is 80% of the benefit enough? Other %?**

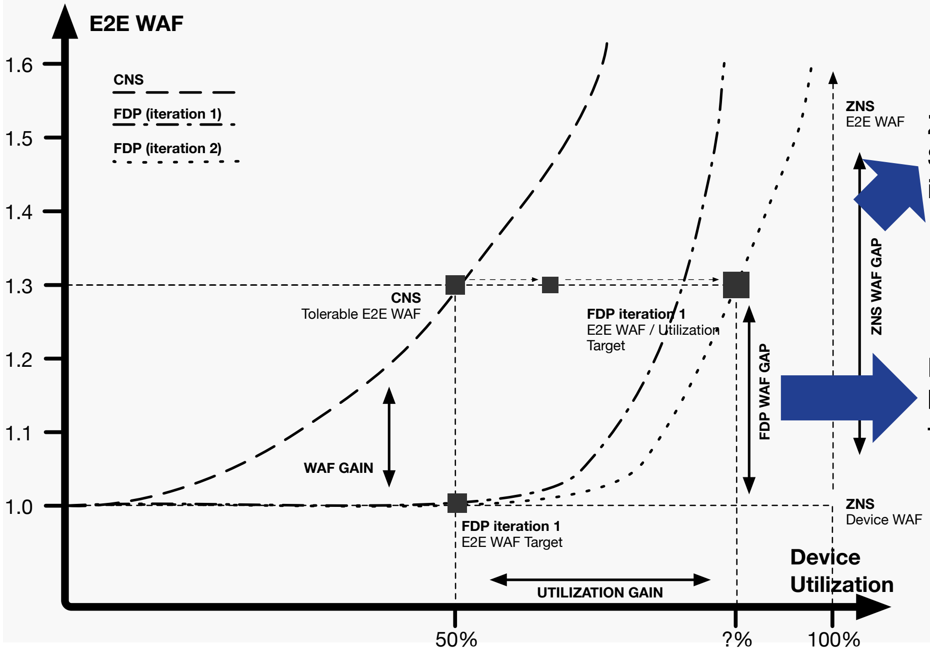
Business Viewpoint

- **Which DP tech. allows customers to meet their requirements at the *lowest* TCO?**
- **What is the trade-off between the initial investment and the expected benefits on a full deployment?**

Experimental Space

FDP & ZNS used as examples

WAF is a critical metric to determine device utilization in real deployments



ZNS WAF can exceed CNS Gap depending on Host implementation

FDP WAF has an upper bound at CNS WAF due to the internal FTL

Data Placement Today: Technology Overview

Confidential

- **Different technologies address the same issue for different use-cases and protocols**
  - Zoned Storage
    - T10/13 stabilized in Zoned Storage (i.e., ZAC/ZBC) for SMR HDDs
    - JEDEC is aligning in Zoned UFS for UFS. Leveraging a lot of the work in ZBC-2
  - Write Hints
    - NVMe is aligning in Flexible Data Placement (FDP) for SSDs

Write Hints

- **Write Model**
  - Backwards compatibility and incremental changes
  - Target generic workloads – with & without SW changes
  - Capacity-based placement through write tags with support for random writes, overwrites, and default
  - Mechanism to deallocate and avoid device-side GC
- **WAF Guarantees**
  - No explicit guarantee by design - WAF improves as a function of the engineering effort in applications
  - Variable through device lifetime as host support is improved
- **Suitability**
  - Applications with in-built data separation (e.g., data / metadata, hot / cold, diff. object sizes, data streams
  - Low engineering effort for first 80% benefit

Zoned Storage

- **Write Model**
  - No backwards combability. Changes are all / nothing
  - LBA-based placement with strict sequential write and no-overwrite constraints. ZAC/ZBC allow RW zones
  - Explicit host deallocation and state machine management
- **WAF Guarantees**
  - Best device WAF: Device WAF ~1 is guaranteed
  - End-to-End WAF varies as a function of the engineering effort
- **Suitability**
  - Fully sequential Apps / FSs with in-built GC and data objects that can be directly mapped zones (e.g., ZonedUFS in F2FS)
  - High engineering effort for 100% of benefit. Especially true for applications with slight unalignment (e.g., metadata overwrites)

Data Placement Today: Industry Alignment

Confidential

Write Hints

↓

NVMe

- **Need to support many different use-cases**
  - Focus on 80%
  - Backwards compatibility
- **Fits block-device Linux model**
  - Tested across different protocols
  - Things to improve (ongoing)

Zoned Storage

↓

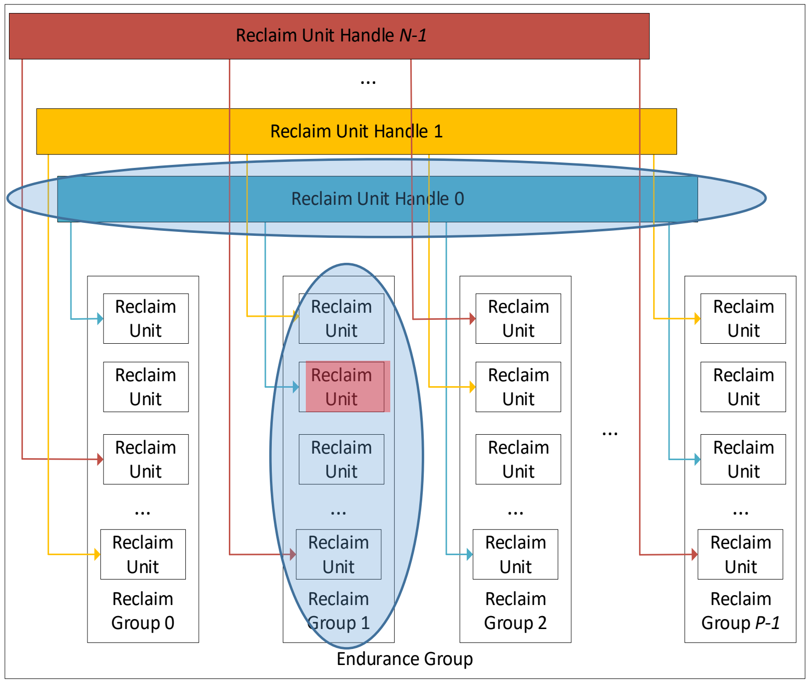
SCSI
UFS

- **SCSI stabilized on ZBC model (SMR HDDs)**
  - Good support in Linux
  - Good decisions for applications (RW zones)
- **UFS targets a controlled environment**
  - Zoned UFS in F2FS for Android Systems
- **Interest in NVMe diminishing due to complexity**

FDP: Overview

Confidential

- **Flexible Data Placement (FDP)**
- **Ratified TP4146 in NVMe**
- **Enables host to provide hint where to place data via virtual handle/pointer**
- **Device changes:**
  - Places data in super block based on hint rather than choosing its own super block
  - Advertises size of super block
- **What functionality does not change**
  - Read
  - Write (Optional write handle added)
  - Deallocate/TRIM
  - Security
- **Backwards compatibility**
  - FDP may be enabled/disabled on standard devices
  - Applications are not required to understand FDP to get benefits
  - Applications which understand FDP have increased benefits



Tech Pit Stop: I/O Passthru (1/3) - Motivation

Confidential

NVMe is no longer tied to the Block Device interface

- Multiple command-sets
  - NVM, Zoned Namespaces (ZNS), Key-Value (KV), Computational Storage, Subsystem local Memory
- Adoption of non-block semantics
- Even block-friendly command-sets define new ways of interacting
  - Zone Append: Nameless write with LBA in completion path
  - FDP Write: Write (LBA, Placement ID)
  - Copy Command: In-device transfer with no payload

Different priorities in NVMe Ecosystem

- Innovation requires fast adoption and prototyping
- Linux kernel values maintainability above cutting edge. Rightfully so!

We need an interface that allows for NVMe Innovation to be deployed in the kernel

- Alternative to SPDK using the in-kernel I/O Path

Tech Pit Stop: I/O Passthru (2/3) - Implementation

Confidential

Char Device

- **Always available. Not dependent on block**

```
>nvme list
Node                  Generic               SN                   Model
--------------------- --------------------- -------------------- ----------------
/dev/nvme0n1          /dev/ng0n1            0123456789ABCDEF0000 SAMSUNG NVMe SSD
/dev/nvme1n1          /dev/ng1n1            PHAL11730018400AGN   INTEL SSDPF21Q40
```

```
static const struct file_operations nvme_ns_chr_fops = {
        .owner          = THIS_MODULE,
        .open           = nvme_ns_chr_open,
        .release        = nvme_ns_chr_release,
        .unlocked_ioctl = nvme_ns_chr_ioctl,
        .compat_ioctl   = compat_ptr_ioctl,
        .uring_cmd      = nvme_ns_chr_uring_cmd,
        .uring_cmd_iopoll = nvme_ns_chr_uring_cmd_iopoll,
};
```

IOCTL

- **Prepare command (80b) and send ioctl**
  - NVME_IOCTL_IO64_CMD
  - NVME_IOCTL_IO64_CMD_VEC
  - NVME_IOCTL_ADMIN64_CMD
- **Submission**

```
if (copy_from_user(&cmd, ucmd, sizeof(cmd)))
        return -EFAULT;
```

- **Completion**

```
if (put_user(cmd.result, &ucmd->result))
        return -EFAULT;
```

io_uring_cmd

- **Prepare command (72b) and send uring-cmd**
  - NVME_URING_CMD_IO
  - NVME_URING_CMD_IO_VEC
  - NVME_URING_CMD_ADMIN
  - NVME_URING_CMD_ADMIN_VEC
- **Submission**
  - Extract cmd from Big SQE
- **Completion**
  - Put result into Big CQE

io_uring capabilities

- **New facility to attach “io_uring capabilities” to any underlying command implemented by the command-provider**
  - Capabilities: Async dispatch, Completion polling, Fixed buffers, Batching
- **Command provider**
  - Can be any kernel component that collaborates with io_uring
  - Example: NVMe driver, Ublk, Sockets
- **User Interface**
  - New opcode: IORING_OP_URING_CMD to go in SQE
  - Command is placed inline in SQE
    - Regular SQE == 16 bytes; Big SQE == 80 bytes
  - SQE->cmd_op contains provider-specific opcode
  - Result arrives in CQE
    - One result in CQE; Additional result in Big CQE

io_uring Big SQE/CQE

- **Double the size of regular**
  - SQE: 128 bytes, CQE: 32 bytes
- **Setup ring with dedicated flags**
  - IORING_SETUP_SQE128 / IORING_SETUP_CQE32
- **Zero-copy for submission / completion command**

Tech Pit Stop: I/O Passthru (3/3) - Status

Confidential

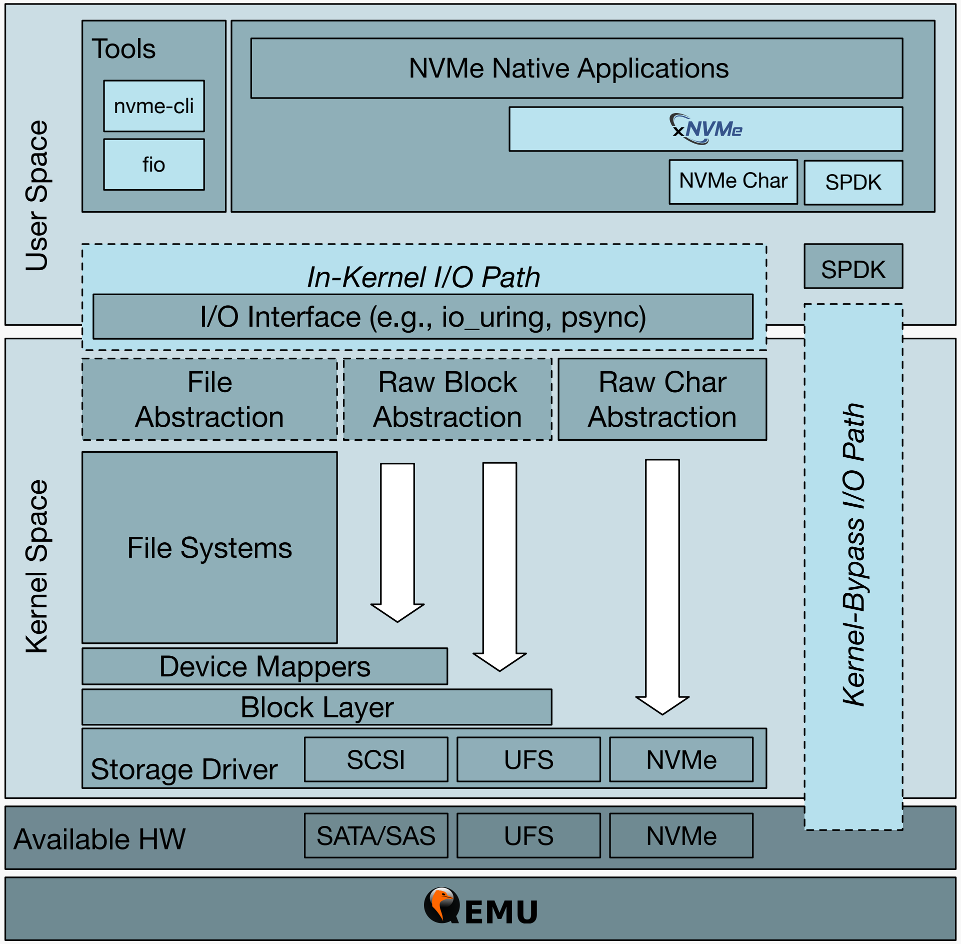
Upstream Kernel

Feature	Version
Generic char interface: initial support	5.13
Generic char interface: any command set	6.0
Io_uring command, Big SQE, Big CQE	5.19
Uring-passthrough for NVMe	5.19
Efficiency: Fixed-Buffer, Completion polling	6.1
Fine-granular access	6.2

Tooling

- **NVMe Cli can use /dev/ngXnY to issue any command**
- **Fio**
  - New 'io_uring_cmd' ioengine
    - FDP support
    - DIF/DIX support
  - t/io_uring support
- **Liburing**
  - Big SQE/CQE awareness
  - Uring-passthrough tests on /dev/ngXnY

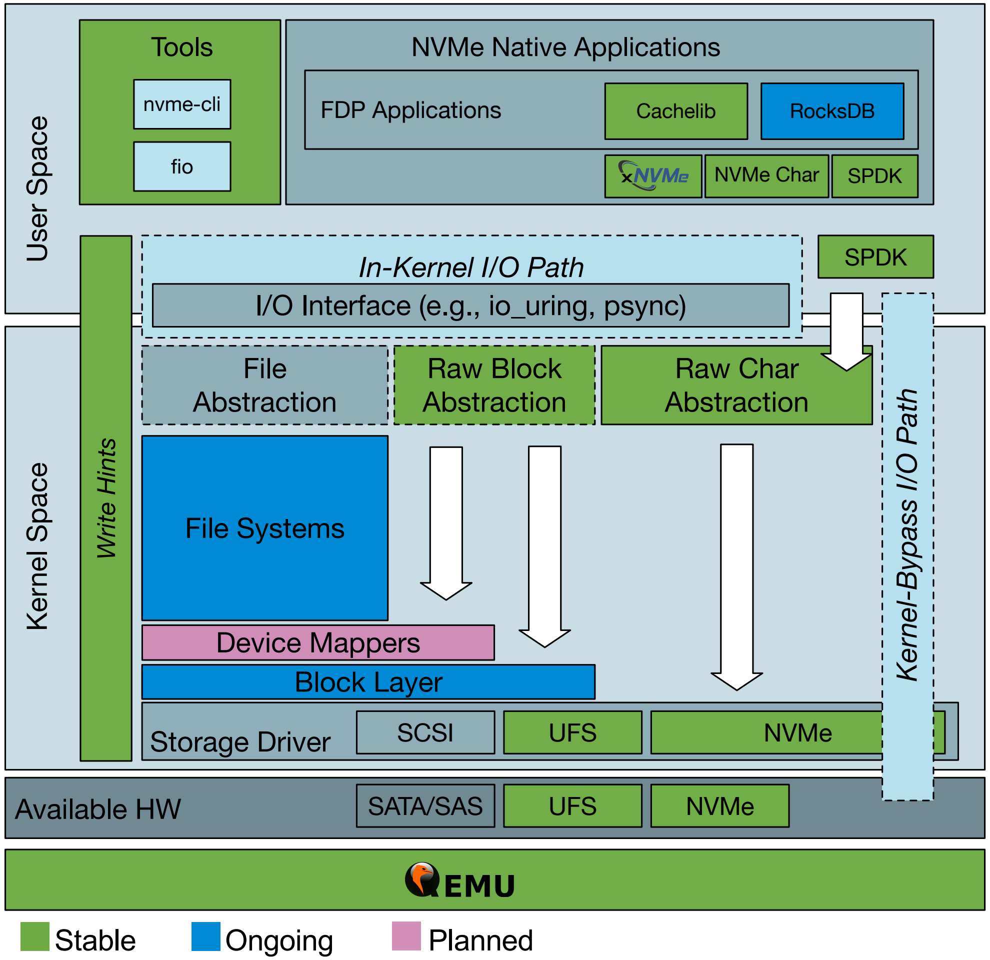
Architecture



FDP: Linux Ecosystem

Confidential

- **FDP already deployed and being using I/O Passthru interface**
  - Upstream integration in Cachelib
  - Storage backed released for RocksDB
- **SPDK Support**
- **Upstream tooling support**
  - fio, nvme-cli, xNVMe, QEMU
- **Ongoing work for block layer support**
  - Enabling hints for block layer
  - Enabling hints for file systems
    - Internal for the FS (e.g., metadata, b-tree)
    - Applications use hints being passed to user-space
  - Trying to re-use existing infrastructure as much as possible (details in next slide)



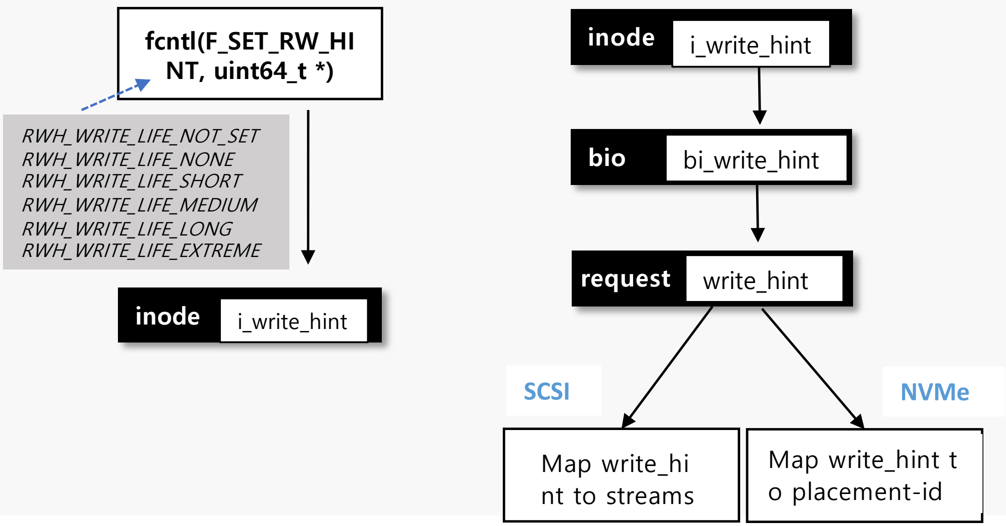
FDP: Ongoing Block Support

Confidential

Use existing write hints (v1-v3)

- **User Interface**
  - Set using fcntl F_SET_RW_HINT
  - Query using fcntl F_GET_RW_HINT
  - The interface supports one type of hint (data lifetime) with 6 possible values
- **Kernel**
  - Stores the hint value in i_write_hint field of file's inode
  - During IO, the hint is propagated down (both direct & buffered I/O)
  - https://lore.kernel.org/linux-nvme/20240702102619.164170-1-joshi.k@samsung.com/

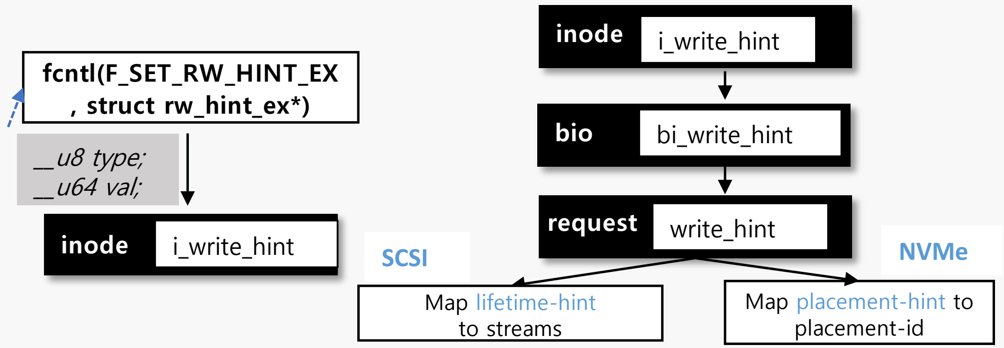
Set / Query write-hints   Dispatch write-hints



Use placement hints (>=V3)

- **User Interface**
  - Set using new fcntl F_SET_RW_HINT_EX
  - Query using new fcntl F_GET_RW_HINT_EX
  - The interface allows passing multiple types of hints
  - TYPE_RW_LIFETIME_HINT with 6 possible values
  - TYPE_RW_PLACEMENT_HINT with 128 possible values
- **Kernel**
  - Stores hint type and value in i_write_hint field of file's inode
  - One bit (MSB) is used to indicate the hint type
  - The inode retains either lifetime hint or placement hint (user decides)
  - During I/O, the hint is propagated down (both direct & buffered I/O)
  - https://lore.kernel.org/linux-nvme/20240910150200.6589-1-joshi.k@samsung.com/

Set / Query write-hints   Dispatch write-hints



Take Aways

Confidential

- **Data placement has been an unsolved problem in storage for at least the past 10 years**
  - NAND & Rotational Media
  - Several technologies have emerged
  - Moving from full host-based placement to host/device collaboration
- **End-to-end solutions in an open-ecosystem is key for success**
  - Need to target different use-cases
  - Need to support different vendors
  - Need to avoid fragmentation
- **We have stabilized in 2 models for 2 different media and use-cases**
  - Zoned Storage
    - HDDs (ZBC for SMR): Stabilized through time
    - UFS (Zoned UFS): Single File System (F2FS) and controlled environment (Android Mobile Devices)
  - FDP
    - NVMe SSDs: Flexibility & backwards compatibility for different hyperscale & enterprise use-cases

THE NEXT CREATION STARTS HERE

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