Introduction

NVM Express® (NVMe®) released the ratified technical proposal TP4146a Flexible Data Placement (FDP)[1] that defines a new method for placing logical blocks into the non-volatile storage in an effort to reduce Write Amplification Factor (WAF) by the SSD. This paper provides:

- A definition for WAF and why it is important to minimize.
- An overview of the FDP architecture and how it is different from a conventional NAND-based NVMe SSD.
- Recommendations for how host software should use FDP to minimize WAF.
- An overview of the host software stack support for FDP.
- A comparison of FDP to other data placement methods defined by NVM Express.

NVM Express 2.0 Family of specifications has defined two other mechanisms for data placement: Streams and Zoned Namespace (ZNS). Throughout this paper, differences between the FDP and the other mechanisms are identified.

Streams provides independent paths (i.e., streams) to writing logical blocks to a namespace in different NAND. The host performs data placement (i.e., which Stream to write) by specifying the stream identifier in the directive of each write command.

Zoned Namespaces breaks up each namespace into a set of equally sized zones. Each zone is associated to physical NAND and corresponds to a sequential set of logical blocks. The host performs data placement by the logical blocks specified in a write command which means that the logical block address specifies the logical blocks being written and which zone is being written. A zone is required to be written sequentially.

Write Amplification Factor

The write amplification factor (WAF) is a measure of the amount of write amplification that occurs on an SSD, because of the way underlying NAND media works. The simple WAF equation when just considering logical blocks written to NAND is:

WAF = (Number of logical blocks written to NAND by the SSD) /
(Number of logical blocks written to NAND by host)

If the SSD only writes the host logical blocks once, then WAF is 1. If the SSD writes the initial host logical blocks more than once, then WAF is greater than 1.

NAND is internally organized in pages, and data can be written to it only on a page by page basis. Whereas, NAND can be only be erased at block level. Typically, a block consists of several hundreds of pages and are required to be erased before being written.

Due to these constraints, when a host writes data to an SSD, the SSD controller may need to read, erase, and then write back the data multiple times in order to complete future write operations. Ideally, the WAF should be as close to 1 as possible. However, it depends on the

[1] https://nvmexpress.org/specification/nvm-express-base-specification/se

nature of the workload and the amount of stored data on SSD. A large WAF negatively impacts SSD performance and its usable life. Therefore, the SSD controller needs to be designed with careful considerations to handle WAF.

How Conventional NVMe SSDs Work

FDP was developed as a method to reduce garbage collection by the SSD[2].

Garbage collection is the process by which the SSD makes previously fully written NAND blocks available for future host writes by moving any host-written logical blocks from those NAND blocks to other NAND blocks (i.e., re-writing host written logical blocks and increasing WAF). This occurs in a conventional SSD today because host software has no control of where logical blocks are placed in the physical NAND of the SSD and cannot assist in managing this process.

Figure 1 illustrates a high-level view of a typical SSD. NAND blocks from each NAND die are grouped into what is called a superblock. For example, NAND Block 0 from each NAND die across all of the NAND channels can form a single superblock. Prior to selecting a particular superblock for writing by the host, all of the NAND blocks within the superblock are erased. Then, as the host writes logical blocks to the SSD, those logical blocks are buffered by the controller until enough logical block data is accumulated that aligns to the write size of a NAND block which is less than the size of a NAND block. There are dedicated resources within the controller to manage and buffer these written logical blocks to a superblock which are referred as a host write resource. At that point, the SSD writes a set of the buffered logical blocks to the NAND block (logically appending the logical blocks to the superblock). The process of buffering the logical blocks and then writing the logical blocks to NAND blocks in that superblock continues until the NAND blocks within the superblock are written to capacity. If the logical

[2] https://semiconductor.samsung.com/resources/white-paper/Samsung_SSD_White_Paper.pdf

blocks written are associated with different namespaces, then the superblock consists of host written logical blocks from different namespaces.

The SSD manages the writes to the NAND blocks within the superblock in a manner that maximizes writing to NAND dies in parallel (i.e., maximizing write performance) while maintaining the expected advertised power.

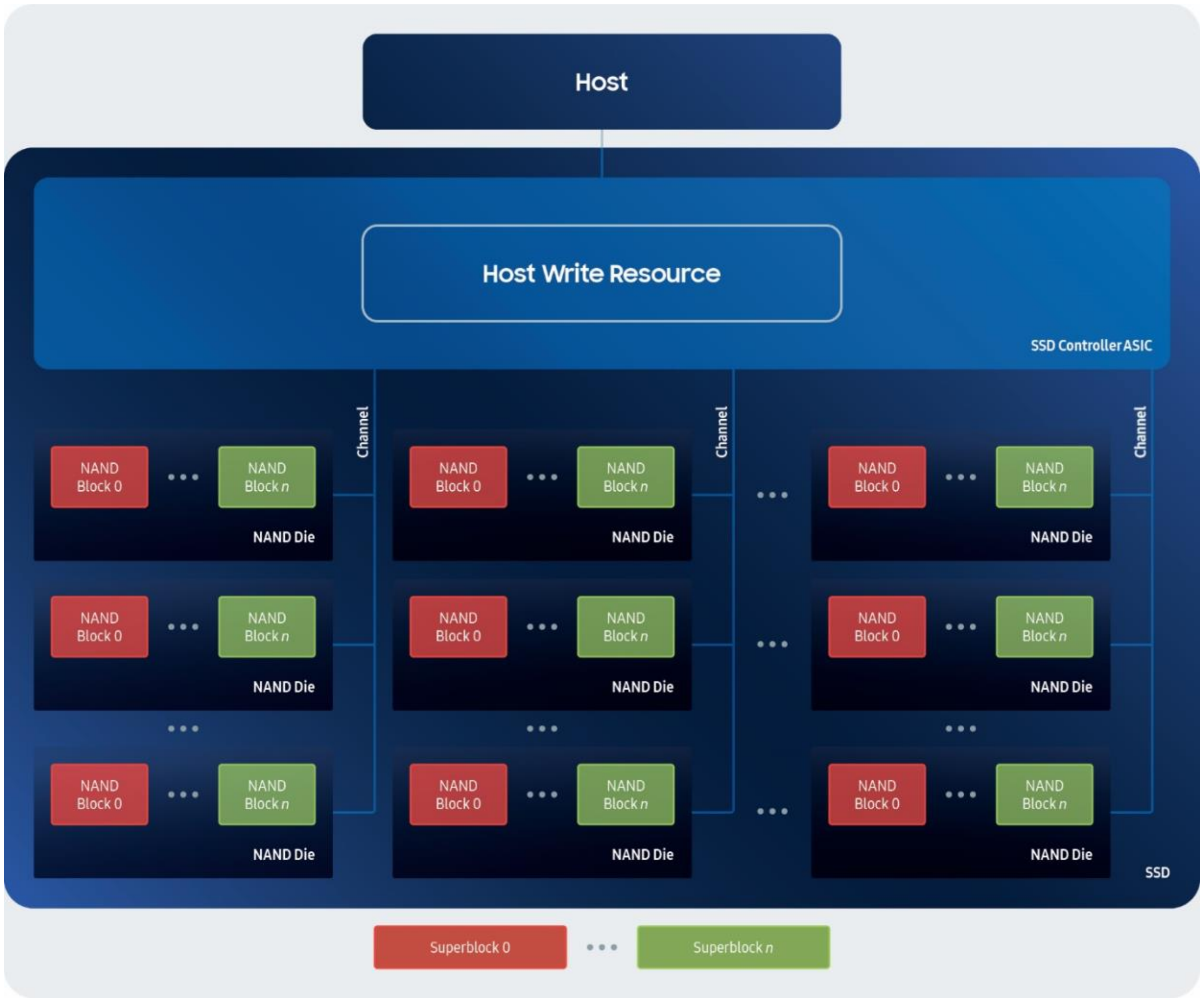


Figure 1. Typical Conventional NVMe SSD

Logically, the writing of logical blocks to a superblock can be represented as appending the logical blocks to the previously written logical blocks as shown in Figure 2. The write commands specify the namespace and the logical block address within that namespace that are written. Therefore, superblocks contain a mixing of logical blocks from different namespaces and the logical block address may be non-sequential.

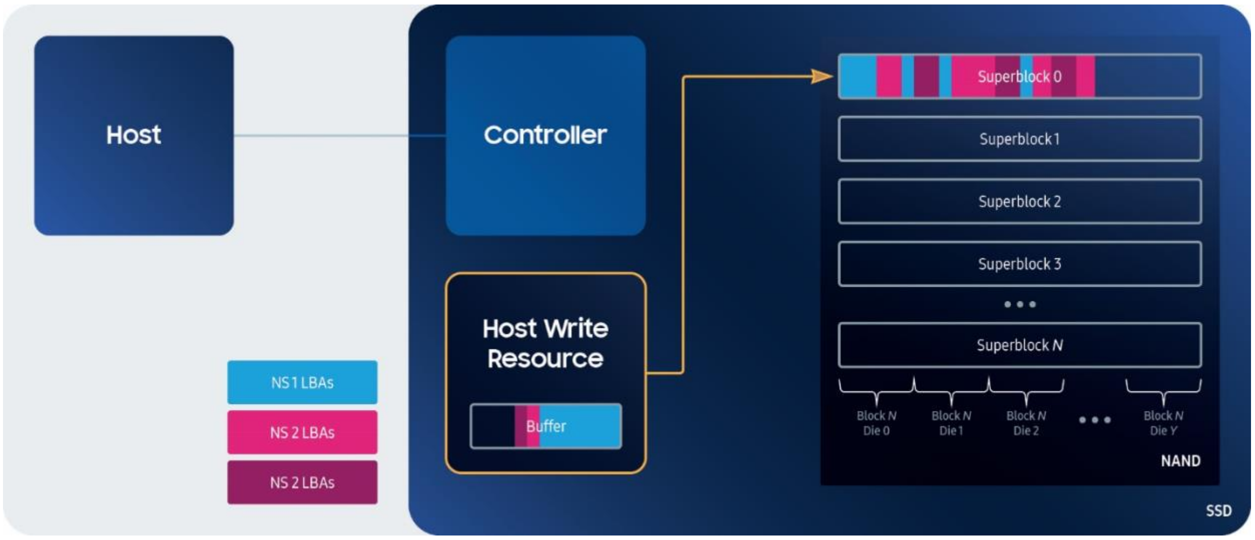


Figure 2. Logical view of a Typical Conventional NVMe SSD using Superblocks

Once a superblock is written to capacity, the SSD selects a different superblock to continue writing host logical blocks. Prior to selecting that superblock, the SSD needs to make sure that the superblock is erased. But if that selected superblock still contains valid previously written logical blocks, then those valid logical blocks are written to another superblock by the SSD prior to erasing the NAND blocks in the selected superblock (i.e., garbage collected) as illustrated in Figure 3. But what is a valid logical block?

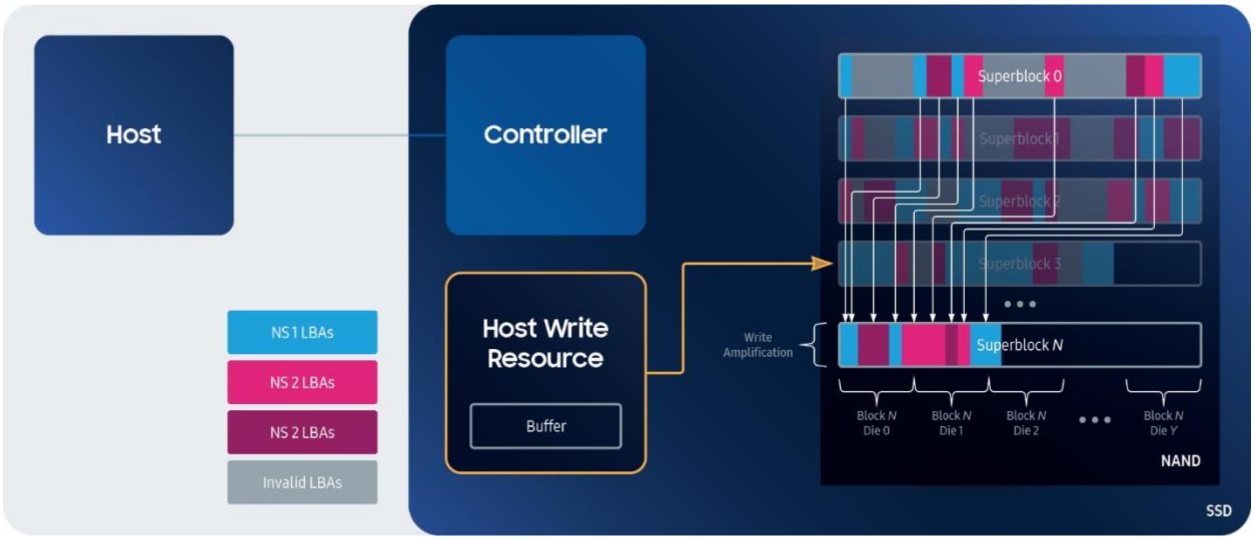


Figure 3. Host Write Resource switching to Logical view of a Typical Conventional NVMe SSD using Superblocks

The SSD keeps track of each logical block written to NAND. If the host writes a logical block to a namespace, then that logical block is now the valid logical block, and the previous written instances of that logical block in the NAND is invalid as that logical block has been overwritten with newer data. These invalid logical blocks may still exist in NAND blocks, but are not required to be moved as part of the garbage collection.

NVM Express has mechanisms for the host to deallocate a valid logical block. The Dataset Management command is such a mechanism. When a valid logical block is deallocated, then the SSD internally marks that logical block as an invalid logical block and therefore that logical block is not required to be moved as part of the garbage collection.

Since the host has no knowledge of superblocks, the host cannot determine which valid logical blocks to either rewrite or deallocate to avoid the garbage collection.

The theory of data placement is that the host can manage the logical blocks written to specific superblocks so that a host can guarantee that logical blocks written to a superblock are invalid before the SSD erases the NAND blocks in that superblock for future writes (i.e., removing all SSD garbage collection). This provides the host the ability to control the WAF in the SSD even a WAF of 1.

Flexible Data Placement (FDP)

The NVM Express ratified technical proposal TP4146a Flexible Data Placement provides a new data placement mechanism that allows to host to control which logical blocks are written into a set of NAND blocks managed by the SSD called a Reclaim Unit. In addition, the host is able to write to more than one Reclaim Unit at a time allowing the host to isolate the data written to Reclaim Unit per application or even within an application to separate data written that has different life cycles (referred to as hot and cold data).

The following is an overview of the FDP architecture as shown in Figure 4 that has the following components:

• **Reclaim Unit (RU):**	A set of NAND blocks that a host may write logical blocks.
• **Reclaim Group (RG):**	A collection of Reclaim Units.
• **Reclaim Unit Handle (RUH):**	A resource within the SSD to manage and buffer the logical blocks to write to a Reclaim Unit (i.e., a host write resource). A namespace is allowed to access to one or more RUHs. If a namespace has access to more than one RUH, the host is allowed to write to multiple RUs at the same time.
• **Placement Handle:**	An index into the list of Reclaim Unit Handles accessible by namespace that is defined at namespace creation. Host writes to that namespace can only access the Reclaim Unit Handles in that list.
• **Endurance Group:**	A collection of NAND blocks that endurance is managed as a single unit with the intention that the NAND blocks wear out at the same time (i.e., SSD life). In a typical SSD, there exists a single Endurance Group.
• **FDP Configuration**	A host selectable configuration that may be enabled that defines the Reclaim Unit size, number of Reclaim Groups, the number of Reclaim Unit Handles, and other information not addressed by this paper.

- **Data Placement Directive** A directive in a write command that uses the Directive Specific (DSPEC) field in that command to identify the Reclaim Unit Handle and Reclaim Group. The host is requesting that logical blocks for a write command be place in the Reclaim Unit referenced by the tuple of information. If a write command does not specify this directive, then the SSD uses Reclaim Unit Handle associate with Placement Handle 0h and selects the Reclaim Group to place the logical blocks.

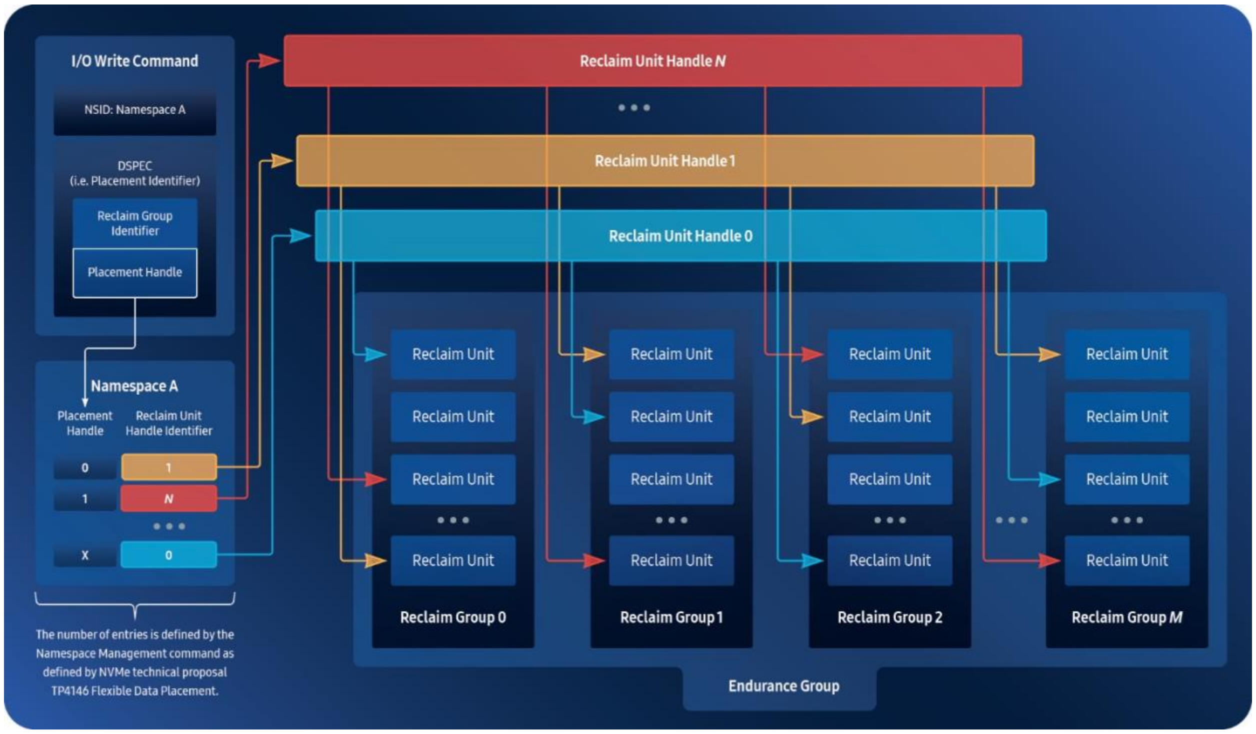


Figure 4. Multiple Reclaim Group Flexible Data Placement Architecture

Data placement by the host for a write command is illustrated in Figure 4. At namespace creation, the host specifies a list of Reclaim Unit Handles accessible to by that namespace. Each write command to that namespace that specifies the Data Placement Directive uses the Directive Specific (DSPEC) field to specify the Reclaim Unit to place the logical blocks being written. The Placement Identifier specifies a Reclaim Group and an index into the list of Reclaim Unit Handles accessible to by that namespace (i.e., Placement Handle). The Reclaim Unit that is being referenced by the Reclaim Unit Handle in the Reclaim Group is where the host is requesting the logical blocks to be written.

A Reclaim Unit can be thought of as a superblock but the host is provided the size of the Reclaim Unit and can manage which logical blocks are written to a Reclaim Unit which is not available in a conventional SSD. A Reclaim Unit is similar to a zone in Zoned Namespaces (ZNS) with the exception that there is no state machine associated to the Reclaim Unit and the host may:

- Write logical blocks randomly into the Reclaim Unit; and

- If different namespaces share access to the same Reclaim Unit Handle, then logical blocks from those namespaces may be written to the same Reclaim Unit.

If the SSD is managing NAND blocks using superblocks, then there is a single Reclaim Group as shown in Figure 5. In this case, the SSD is managing the actual writing of logical blocks across the NAND dies and will maximize parallelism. If the SSD is managing NAND blocks per NAND die, then there is a Reclaim Group for each die giving the host the ability to write the a Reclaim Unit within a specific NAND die. In this case, host software is managing the writing of logical blocks to NAND dies and is responsible for managing the writing of data across all of the NAND dies.

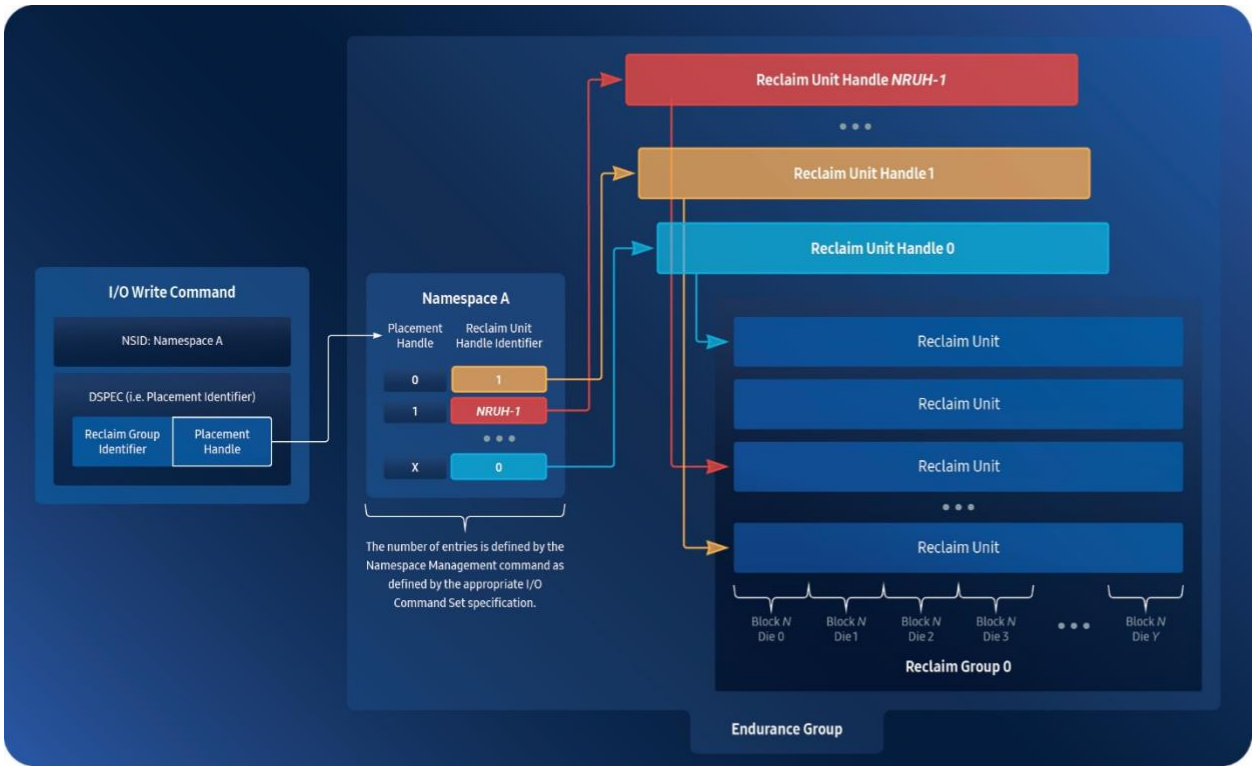


Figure 5. Single Reclaim Group Flexible Data Placement Architecture

The ability for namespaces to share access to the same Reclaim Unit Handle is important for FDP to claim backwards compatibility. Backwards compatibility is the ability to plug an SSD that has FDP enabled into a server that is not aware of FDP and that server is able to write logical blocks without getting an error. A ZNS SSD is not backwards compatible as the host is required to have ZNS knowledge to be able to write the SDD.

Designing in backward compatibility into the FDP architecture is crucial in giving a host the option on planning how to modify the host software to take full advantage of FDP and fully manage the garbage collection of the SSD. For example, FDP can be enabled on an SSD and inserted into a server that does not understand FDP. Host writes to namespaces causes the SSD to select the Reclaim Unit Handle and Reclaim Group to place the logical blocks to a Reclaim Unit. Since FDP is enabled, the host software can be modified at any time the host wants to change their software to support FDP.

The set of Reclaim Unit Handles is analogous to an SSD supporting multiple streams where write commands may specify the Stream Directive to indicate the write resources to use for writing the logical blocks. FDP provides the host the ability to know if writes are aligned to the Reclaim Unit by issuing an I/O Management Receive command to determine how many logical blocks are available to be written to the Reclaim Units currently being referenced by the Reclaim Unit Handles accessible to a namespace. The NVMe Streams capability has no mechanism for the host to determine the current alignment. ZNS has the number of active zones which is analogous to the set of Reclaim Unit Handles.

To obtain a WAF of 1 by the SSD using FDP, host software has to overwrite or deallocate all of the logical blocks written to a Reclaim Unit before the SSD is required to erase the NAND blocks in that Reclaim Unit for future writes.

If the host is rewriting logical blocks prior to the SSD requiring to perform garbage collection on the previous written data for those logical blocks, then by the time the SSD would perform garbage collection, all of the previously written data is invalid and the SSD does not have to move the logical blocks; therefore, no host action is required. Otherwise, the host has to track each logical block written to each Reclaim Group. For those logical blocks that have not been rewritten or deallocated, the host is required copy those logical blocks to another Reclaim Unit to avoid garbage collection by the SSD. In this case, the SSD WAF is 1 for that Reclaim Unit, but the system incurs an increase in WAF as the host was required to move the logical blocks instead of the SSD as part of garbage collection.

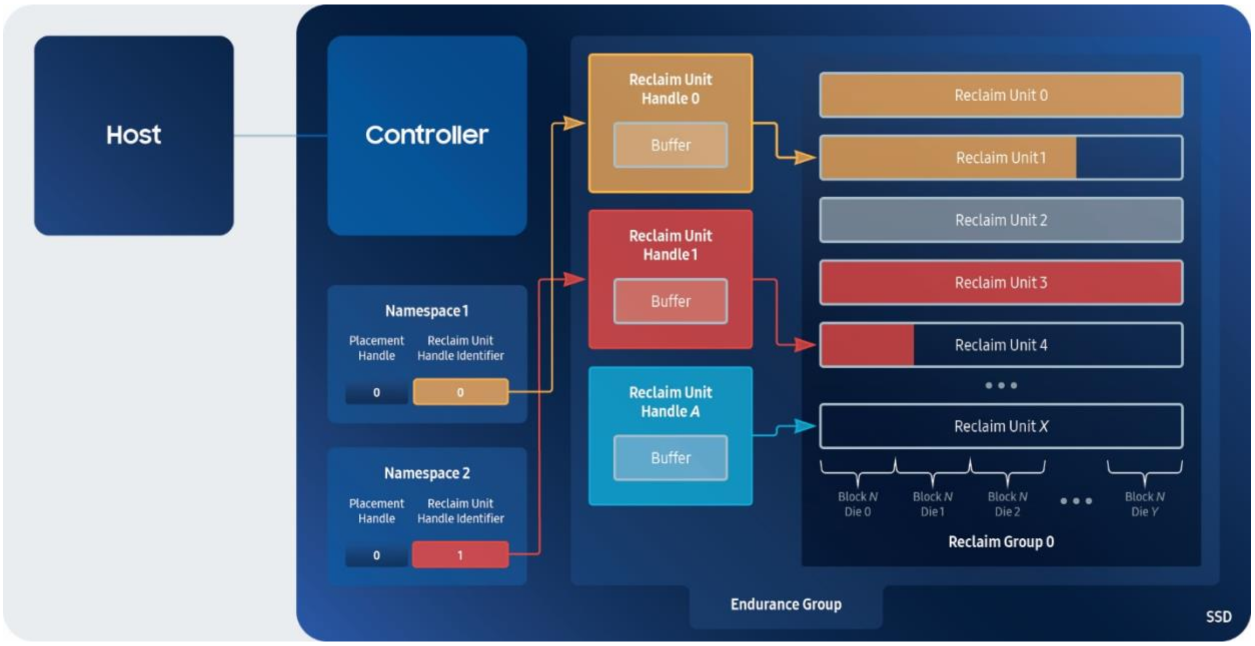


Figure 6. FDP Reducing WAF

Figure is an illustration of two namespaces having access to two different Reclaim Unit Handles which means that when the host initially writes logical blocks to those namespaces, the logical blocks for the different namespaces are isolated in different Reclaim Units. In this example:

- Reclaim Unit 0 was previously written to capacity with logical blocks for Namespace 1.

- Reclaim Unit 1 is the current Reclaim Unit being written by Reclaim Unit Handle 0 and future writes to Namespace 1 are written to this Reclaim Unit.
- Reclaim Unit 2 was previously written to capacity with logical blocks for Namespace 2 but the host has either rewritten the logical blocks in another Reclaim Unit or has deallocated the logical blocks.
- Reclaim Unit 3 was previously written to capacity with logical blocks for Namespace 2.
- Reclaim Unit 4 is the current Reclaim Unit being written by Reclaim Unit Handle 1 and future writes to Namespace 2 are written to this Reclaim Unit.

Because the host causes all logical blocks in Reclaim Unit 2 to be invalid, then when the SSD requires to use the NAND associated with Reclaim Unit 2 for future writes, the SSD has no valid logical blocks to copy to another Reclaim Unit meaning there is no WAF increase as opposed to Figure 3. Figure also illustrates the separation of namespace logical blocks making it easier for the host to track the LBAs to Reclaim Units.

Logical blocks with similar data life (i.e., called temperature) should be written to the same Reclaim Unit so that all of the logical blocks should become obsolete (i.e., invalid) at around the same time due to being re-written or deallocated.

Refer to Samsung's OCP Global Summit 2022 FDP presentation [3] if additional understanding of FDP is needed.

Investing in obtaining the best SSD WAF

From a systems perspective, designing a NVMe-base data placement solution to reduce WAF presents a trade-off, as there are several alternatives in the standard. Specifically, the comparison between Zoned Namespaces (ZNS) and FDP becomes particularly interesting.

ZNS is a strict interface that requires the host software stack fully support ZNS. To use an SSD supporting ZNS, that host is required to port all affected host software to use the strict ZNS interface. A ZNS SSD is not able to be used in a server that does not support ZNS (i.e., it is not backwards compatible to a conventional SSD.

In our experience, ZNS is well-suited for solutions built from the ground up that can guarantee sequential writes, fixed size objects, write error handling, and a robust I/O completion path to handle anonymous writes using the Append Command[4]. This is because this approach removes the indirection layer present in traditional Flash Translation Layers (FTLs), and at the same time allows zones to be mapped to system representations (e.g., objects). In this system, ZNS allows to fully leverage the TCO advantages of the technology. However, systems where any of the above is difficult to guarantee, our recommendation is to be wary of the amount of work required in the host software stack. This is especially relevant when these solutions introduce indirection layers to bypass any of the requirements imposed by ZNS. Examples include (i) host-side FTLs

[3] OCP Global Summit FDP Presentation recorded on YouTube, https://www.youtube.com/watch?v=ZEISXHcNmSk&pp=ygUXZmxleGlibGUgZGF0YSBwbGFjZW1lbnQ%3D

[4] Dayan, Niv, et al. "Modelling and Managing SSD Write-Amplification." *ArXiv.org*, 1 Apr. 2015, arxiv.org/abs/1504.00229. Accessed 18 Oct. 2023.

to translate a small amount of random writes into sequential writes, (ii) indirections to support in-place metadata updates, or (iii) sub-zone mappings to enable variable object sizes. In our experience with real-life deployments, the WAF introduced by these software layers surpasses the benefits of the ZNS interface.

FDP is a more flexible interface that enables the host software stack to introduce incremental benefits. This gives system designers an explicit trade-off between the desired WAF and the investment in the engineering effort.

In our experience, FDP is well-suited for retrofitting existing applications that follow software data placement policies, which can then be communicated to the device in exchange for a WAF benefit. These types of applications are already deployed, they are not always sequential, deal with objects of different sizes, and rely in robust, existing I/O paths. Changes to these applications are difficult as they support existing infrastructure. Here, FDP will in most cases not provide a perfect alignment leading to WAF 1. Instead, FDP will make it possible to leverage host data separation to improve end-to-end WAF incrementally. System designers will be able to evaluate the engineering effort required to improve WAF and take this as yet another metric to compute their engineering return on investment (ROI).

At Samsung, we have successfully implemented FDP support for Cachelib and RocksDB and have been able to quantify WAF benefits with minimal application changes. At the time of this writing, this support is currently being upstreamed. Refer to Figure 7 and Figure for measured WAF with Cachelib at varying SSD utilizations where the SSDs were pre-conditioned prior to the measurement.

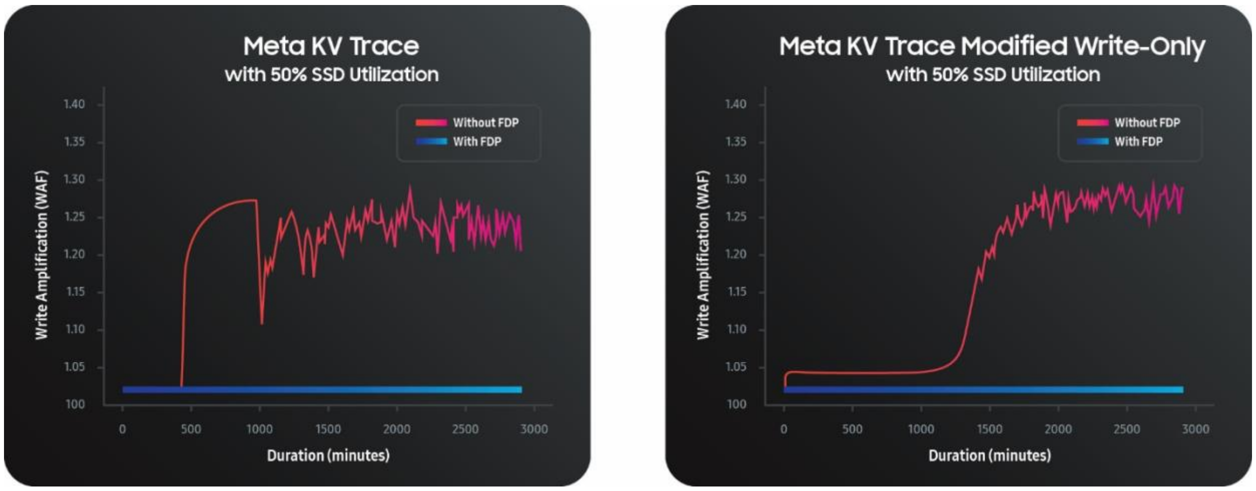


Figure 7. Cachelib results with 50% SSD utilizations

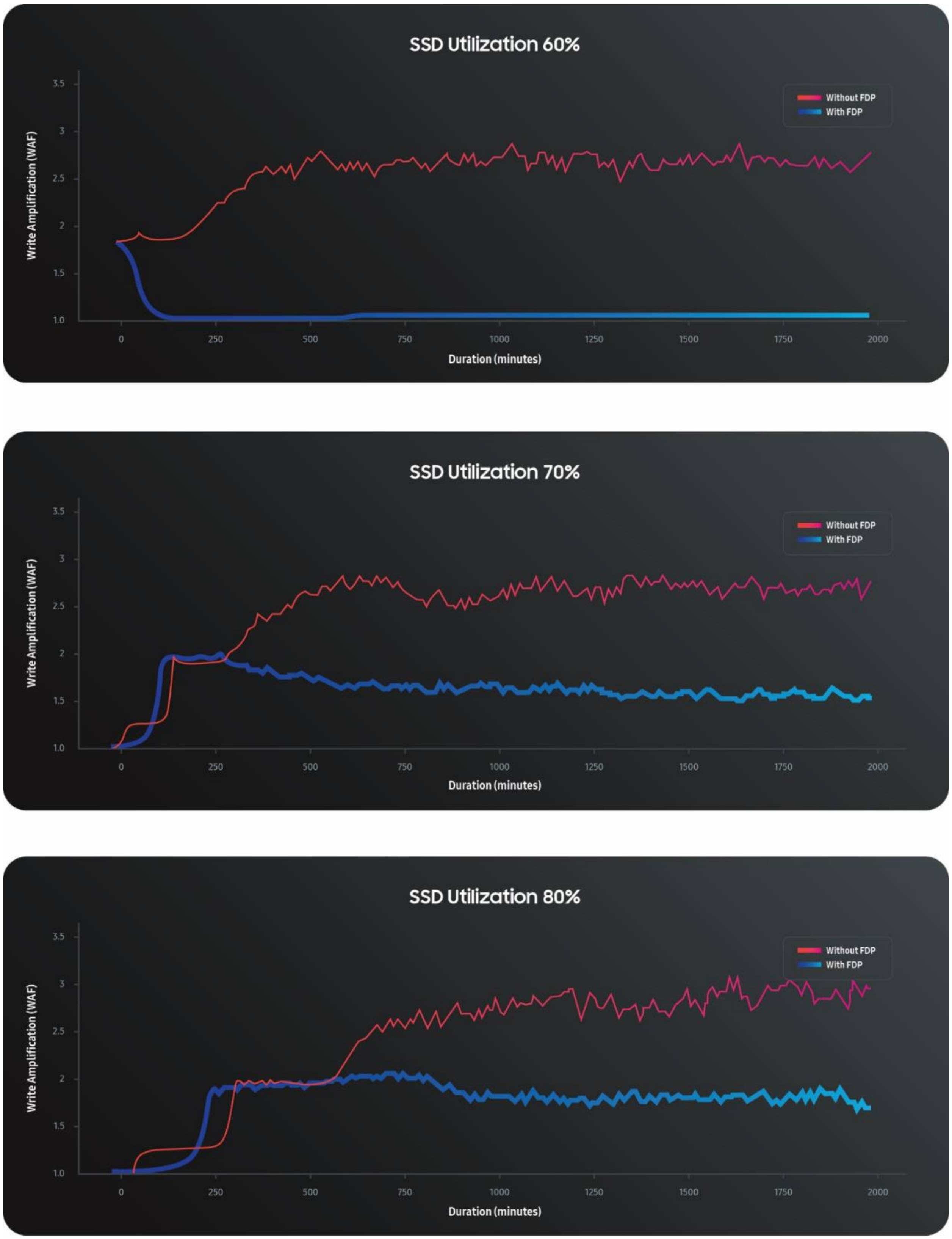


Figure 8. Cachelib results with various SSD utilizations

Computing Return on Investment (ROI) with FDP

Reducing the WAF of the SSD improves the endurance and performance of the SSD. Storage systems commonly do not fully utilize the SSD Capacity to achieve acceptable levels of endurance or performance. Not fully using the SSD capacity increases the over-provisioning (OP) thus lowering WAF. This is an expensive proposition since the cost of the SSD capacity is not utilized for storage. The relationship between OP and WAF depends on the workload; however, the WAF of random write workloads is well understood and can be approximated using well-known mathematical methods as illustrated in Figure 9. The technique used to derive the graph is a Lambert equation method for a random write workload.   Refer to reference papers[5] for further detail. Contact Samsung for specific product WAF information.

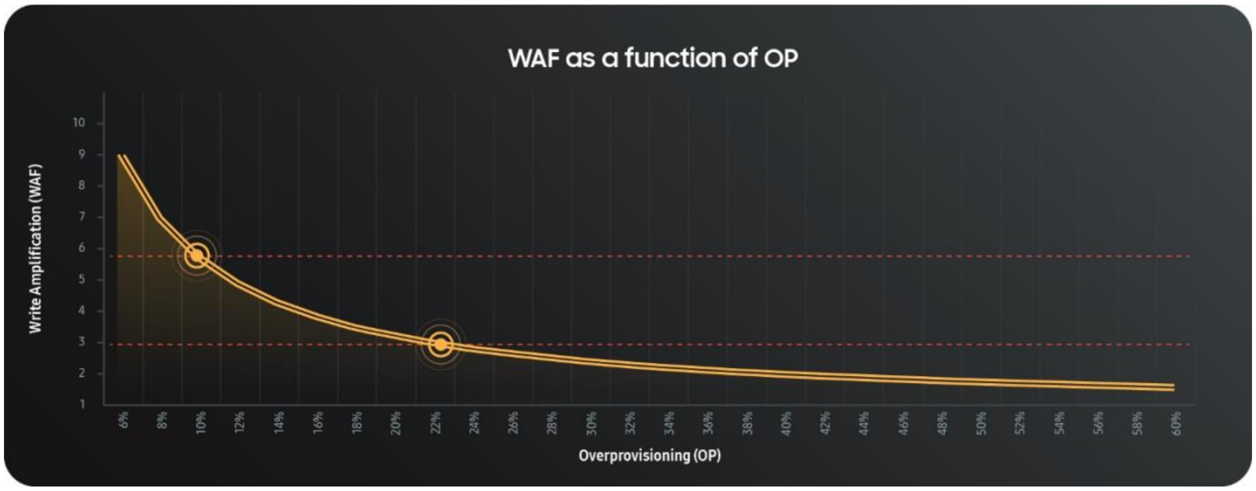


Figure 9. WAF as a function of OP

As an example, if a random workload is experiencing a WAF of 5.7 at an OP of 10% but endurance is required to be increased by 2X, WAF must be reduced by half. OP of approximately 23% is required to achieve WAF of 2.85. The additional 13% OP is lost SSD logical capacity.

$$\text{OP} = \text{SSD physical capacity/logical capacity} - 1$$

Another reason a system may increase OP is to improve the performance of random writes. A WAF > 1 requires SSD garbage collection. Garbage collection involves reading and writing valid logical blocks from a NAND block prior to erasing that NAND block for future writes. This internal work reduces the performance of the SSD. The reduction in random write performance can be approximated with a simple equation that takes into account the extra internal garbage collections read and write traffic.

Random write performance at WAF of N = RWP_N

$$RWP_N = RWP_1/(1 + (WAF-1)*1.1)$$

[5] Peter Desnoyers, Analytic Modeling of SSD Write Performance.

Furthermore, random mixed read/write workloads can be modeled under the assumption that available SSD resources, such as NAND die, are equally available to be utilized by both reads and writes. There is no WAF impact to 100% read workloads. The performance impact of WAF increases as the percent of writes increases.

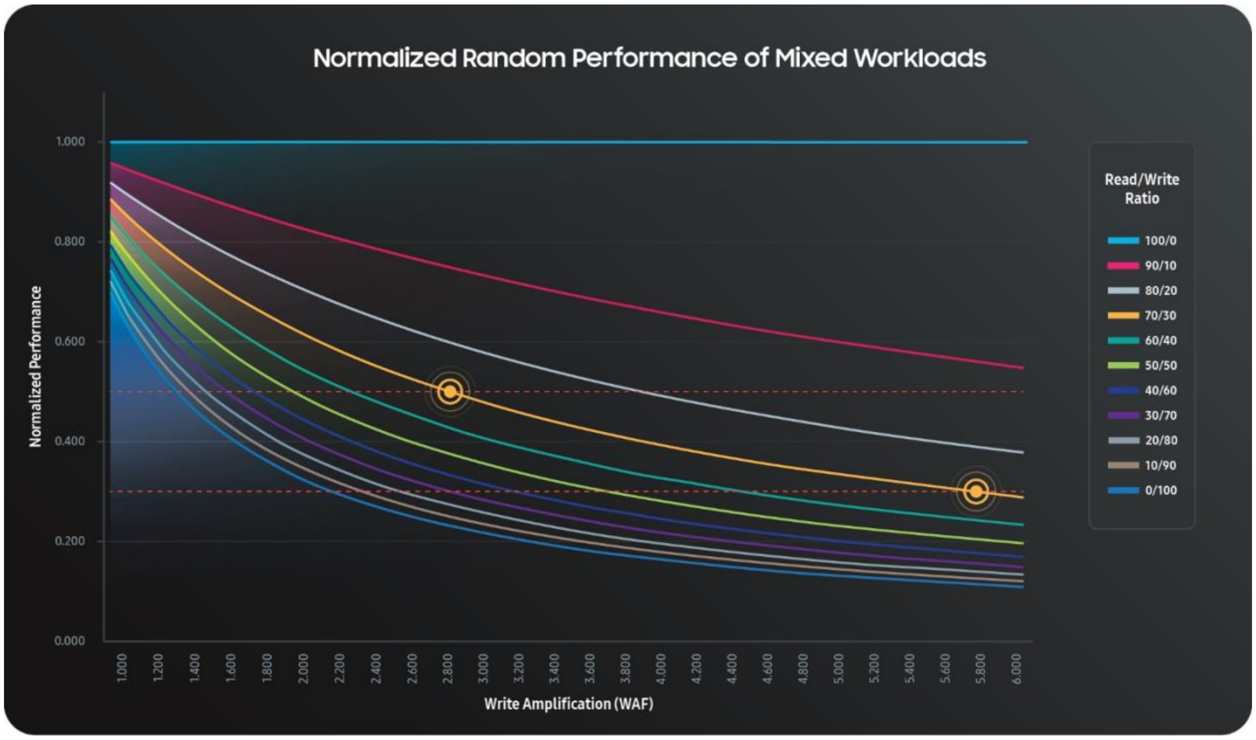


Figure 10. Normalized Random Performance

Repeating the example above and using Figure 10, if a workload is encountering a WAF of 5.7 and assuming a read/write ratio of 70/30, its normalized performance is 0.3. Reducing WAF to 2.85 increases normalized performance to 0.5. An approximate 67% increase in performance but at the same expense as above; OP increased from 10% to 23%.

FDP is an alternate way to improve the endurance and performance of a SSD. How much WAF can be lowered by converting a system to FDP is a function of the host workload; therefore each workload may have a different solution when modifying the software to support FDP. Workloads that (partially) sequentially write or can be converted to sequential writing will generally benfit the most. As one example use case, CacheLib was utilized to assess the improvements to WAF. “CacheLib is a C++ library for accessing and managing cache data. It is a thread-safe API that enables developers to build and customize scalable, concurrent caches. It is targeted at applications that dedicate gigabytes of memory to cache information[6]”.

CacheLib issues both metadata and data writes. Example CacheLib workloads were converted to support FDP by using a dedicated Reclaim Unit Handle for (small) metadata and a different

[6] “CacheLib.” *Cachelib.org*, cachelib.org/. Accessed 18 Oct. 2023.

dedicated Reclaim Unit Handle for (big) data. This conversion required minimal effort for FDP since CacheLib inherently sequentially writes a large portion of its data. Other systems may need to alter the nature of how data is written and trimmed/unmapped to align to FDP. For this particular example, no other host software changes were required. Small data is effectively written randomly while big data is written sequentially. The sequentially written data is overwritten thus encounters a WAF of 1. The device OP is applied to the random data by the SSD since it is not being used for the sequential data. Since the capacity of the random data written is relatively small, a few percent of SSD capacity, the device OP is able to achieve a WAF of close to 1 even for the random data. Converting to FDP dramatically improved the observed SSD WAF. Note that the system still encounters non-device based WAF for its internal log-structured write methods. FDP did not impact the non-device based WAF.

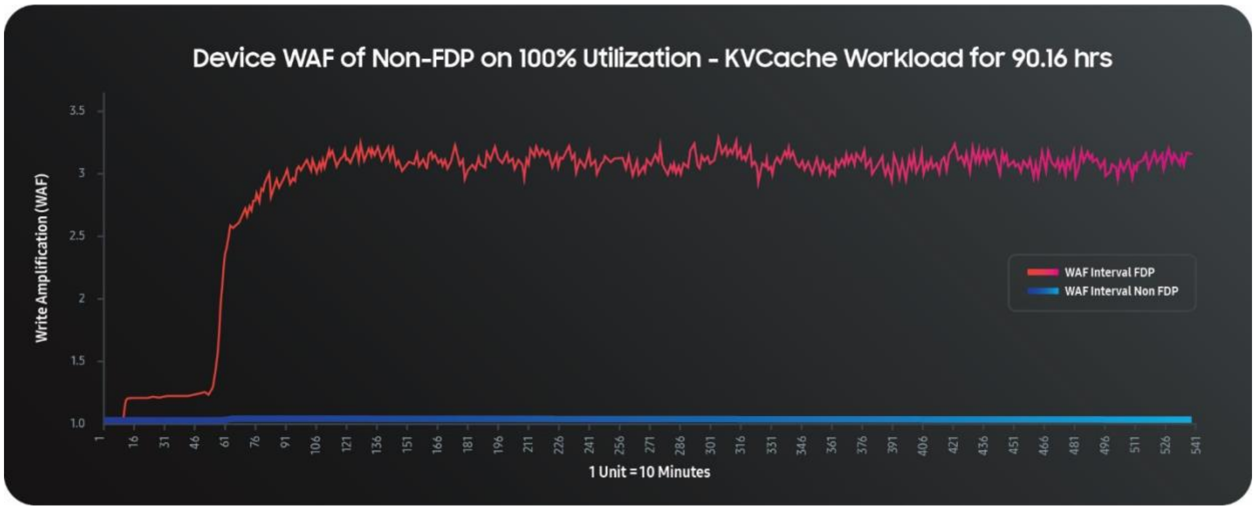


Figure 11. CacheLib WAF

Figure 11 illustrates the observed WAF improvement with FDP on CacheLib. SSD WAF dramatically improved from approximately 3 to 1. The endurance improvement directly scales with the improvement in WAF. Performance improvements are a function of the read/write ratio. This workload is dominated by approximately 90% reads. Using the translation provided by Figure 10 normalized performance increased from 0.74 to 0.96. If this increased performance is consumed then the lifetime of the SSD is impacted by the increased write activity. FDP conversion enabled increased performance of (0.96/0.74 = 1.3) and increased life of 3/1.3 = 2.3. Note: increased life is based on DWPD rather than warranty period. Note that the performance improvement estimates only include the impact of WAF. Vendor specific FDP implementation design decisions may impact performance.

Other workloads and systems are bound to exhibit different gains and value propositions. CacheLib analysis of FDP indicated very significant value. Analysis of another log-structured workload has also reaffirmed that FDP gains are not isolated to CacheLib.

FDP and the Host Software Stack

FDP is supported in several parts of the Linux ecosystem as shown in Figure 12.

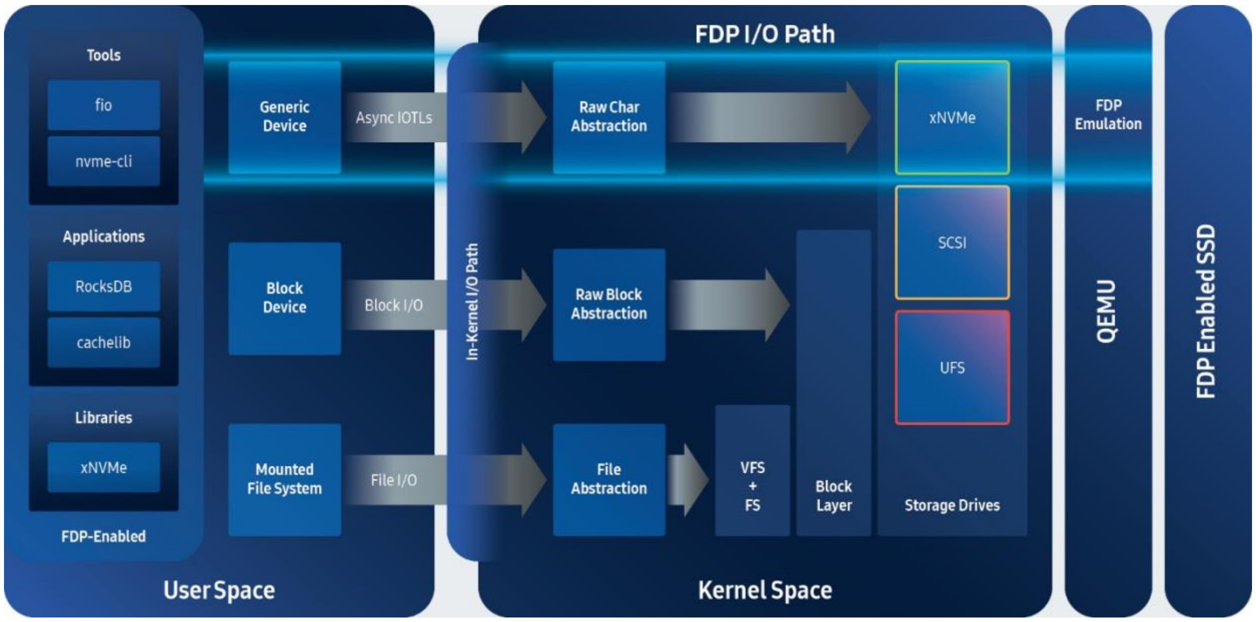


Figure 12. FDP Support in the Linux Ecosystem

In the Linux kernel, FDP leverages the I/O Passthru interface [7]- an asynchronous path based on io_uring to the NVMe generic character device that allows user-space applications to directly access the Kernel NVMe driver, bypassing the block layer. This means that applications can leverage an end-to-end architecture similar to SPDK [8], but using the in-kernel NVMe driver instead. This simplifies the host stack as it removes the need to deploy and manage user-space drivers, delegating this responsibility to the operating system. I/O Passthru is fully supported in the Linux kernel since version 6.2.

In user-space, both xNVMe[8] and SPDK[9] provide full support for FDP. xNVMe enables FDP in different storage paths by leveraging the SPDK NVMe driver and the I/O Passthru. The ability to connect FDP-enabled SSDs with standard storage interfaces makes adoption easy for application developers and storage infrastructure architects. FDP is also present in widely used tools such as fio and nvme-cli, which enable respectively testing and management.

From an application point of view, FDP is supported in Cachelib. Here, FDP allows to separate Cachelib's two caches: BigHash – the cache for small items, and BlockCache – the cache for big

[7] "SDC2021: Enabling Asynchronous I/O Passthru in NVMe-Native Applications." *Www.youtube.com*, www.youtube.com/watch?v=mtiQlPZsw4w. Accessed 18 Oct. 2023.
[8] "XNVMe: Cross-Platform Libraries and Tools for NVMe Devices." *GitHub*, 29 Aug. 2023, github.com/OpenMPDK/xNVMe.
[9] "Storage Performance Development Kit." *GitHub*, 29 Aug. 2023, github.com/spdk/spdk.

items. By separating these two caches, Cachelib can achieve better WAF[10]. The changes to Cachelib are minimal and are contained within Cachelib itself – no changes are needed to the applications on top of Cachelib.

From an emulation perspective, FDP is fully supported as of QEMU[11] version 8.0. Using QEMU, operating system and application developers are free to develop in advance of FDP-enabled SSDs becoming widely available.

Comparing FDP to Streams and ZNS

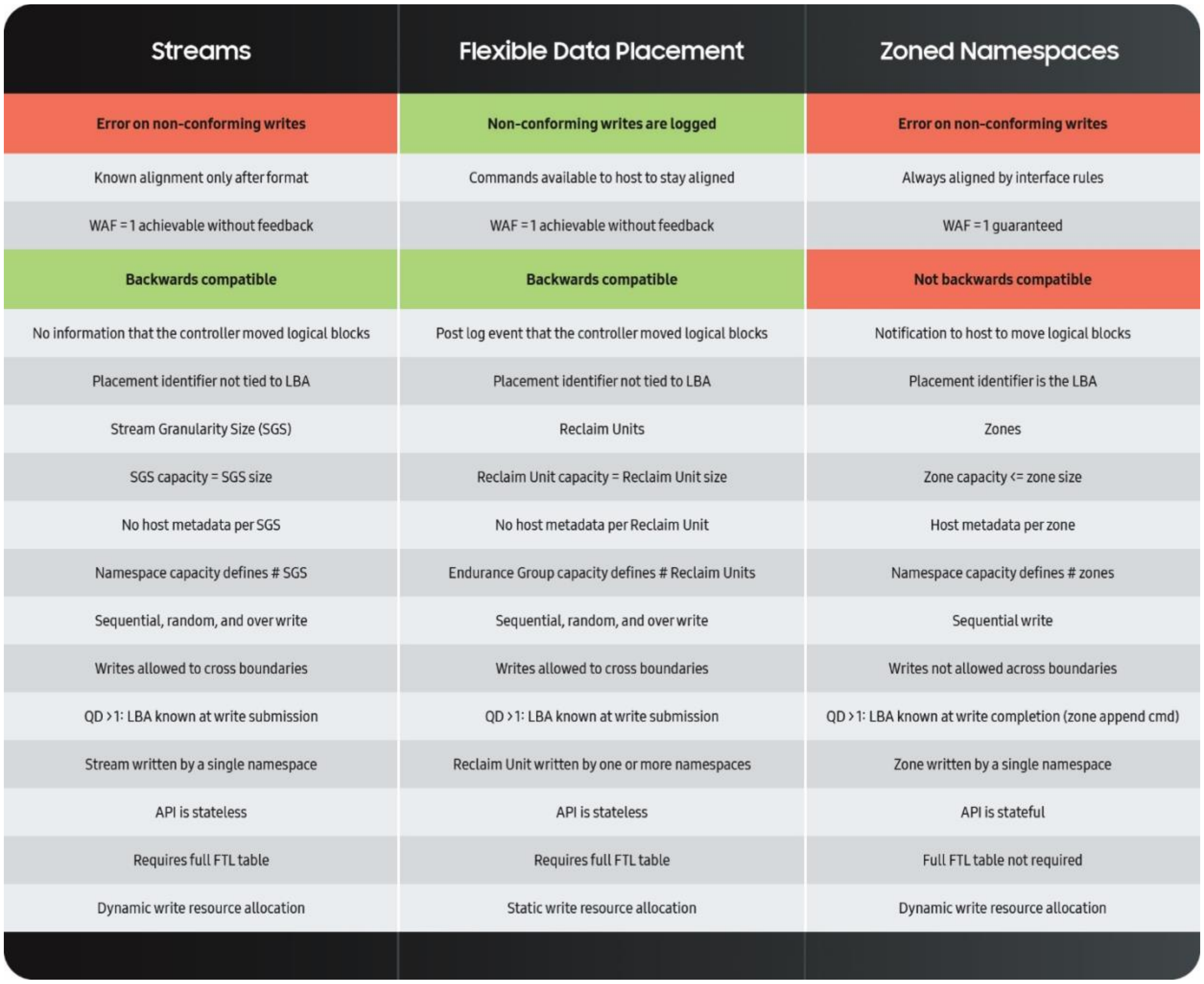
Table 1 provides a side-to-side comparison that shows that FDP is the most flexible for allowing host software to achieve the best WAF based on investment on adapting host software. The following paragraphs provide details of the rows highlighted in orange.

FPD was architected to complete a host issued write command that does not conform to the defined protocol and to provide feedback to the host that such a write did not conform to the protocol. For example, if a host issues a write command that the identified Reclaim Unit Handle and Reclaim Group in the Data Placement Directive is invalid, then the write is completed by the SSD and is placed in a Reclaim Unit accessible to the namespace selected by the controller and the event is logged in an FDP event, if enabled by the host. If the host issues a write to an invalid Stream the command is aborted. If a host issues a write command to a ZNS namespace that crosses a zone boundary, then the write command is aborted.

Both Streams and FDP enabled SSD can be removed from a server and plugged into an older server and would just work. The older server is able to create namespaces and write to namespaces without any errors. That is not true with ZNS. A ZNS SSD can only operate in a server that understands ZNS.

[10] "Overview: A Random Walk down the Cache Library | CacheLib." *Cachelib.org*, cachelib.org/docs/Cache_Library_Architecture_Guide/overview_a_random_walk/. Accessed 18 Oct. 2023.
[11] https://gitlab.com/qemu-project/qemu

Table 1 Comparison between Streams, FDP, and ZNS



Streams	Flexible Data Placement	Zoned Namespaces
Error on non-conforming writes	**Non-conforming writes are logged**	**Error on non-conforming writes**
Known alignment only after format	Commands available to host to stay aligned	Always aligned by interface rules
WAF = 1 achievable without feedback	WAF = 1 achievable without feedback	WAF = 1 guaranteed
Backwards compatible	**Backwards compatible**	**Not backwards compatible**
No information that the controller moved logical blocks	Post log event that the controller moved logical blocks	Notification to host to move logical blocks
Placement identifier not tied to LBA	Placement identifier not tied to LBA	Placement identifier is the LBA
Stream Granularity Size (SGS)	Reclaim Units	Zones
SGS capacity = SGS size	Reclaim Unit capacity = Reclaim Unit size	Zone capacity <= zone size
No host metadata per SGS	No host metadata per Reclaim Unit	Host metadata per zone
Namespace capacity defines # SGS	Endurance Group capacity defines # Reclaim Units	Namespace capacity defines # zones
Sequential, random, and over write	Sequential, random, and over write	Sequential write
Writes allowed to cross boundaries	Writes allowed to cross boundaries	Writes not allowed across boundaries
QD > 1: LBA known at write submission	QD > 1: LBA known at write submission	QD > 1: LBA known at write completion (zone append cmd)
Stream written by a single namespace	Reclaim Unit written by one or more namespaces	Zone written by a single namespace
API is stateless	API is stateless	API is stateful
Requires full FTL table	Requires full FTL table	Full FTL table not required
Dynamic write resource allocation	Static write resource allocation	Dynamic write resource allocation

In Conclusion

This paper discusses FDP architecture and presents a compelling case on how FDP can help to improve the performance and endurance of an SSD. An example of CacheLib is demonstrated to show how FDP can help to achieve WAF of a SSD closer to 1.

All measured data shown in this paper is collected from Samsung's next-generation PM9D3 datacenter SSD. Samsung is collaborating with our customers and partners to support FDP in future products. For more information contact Samsung support.

This paper also shows Samsung's commitment to the development of FDP and continuing the advancement of host stack in supporting FDP.

Disclaimer: The performance analysis shown in this paper only considers the impact of WAF and does not include any specific FDP implementation choices.