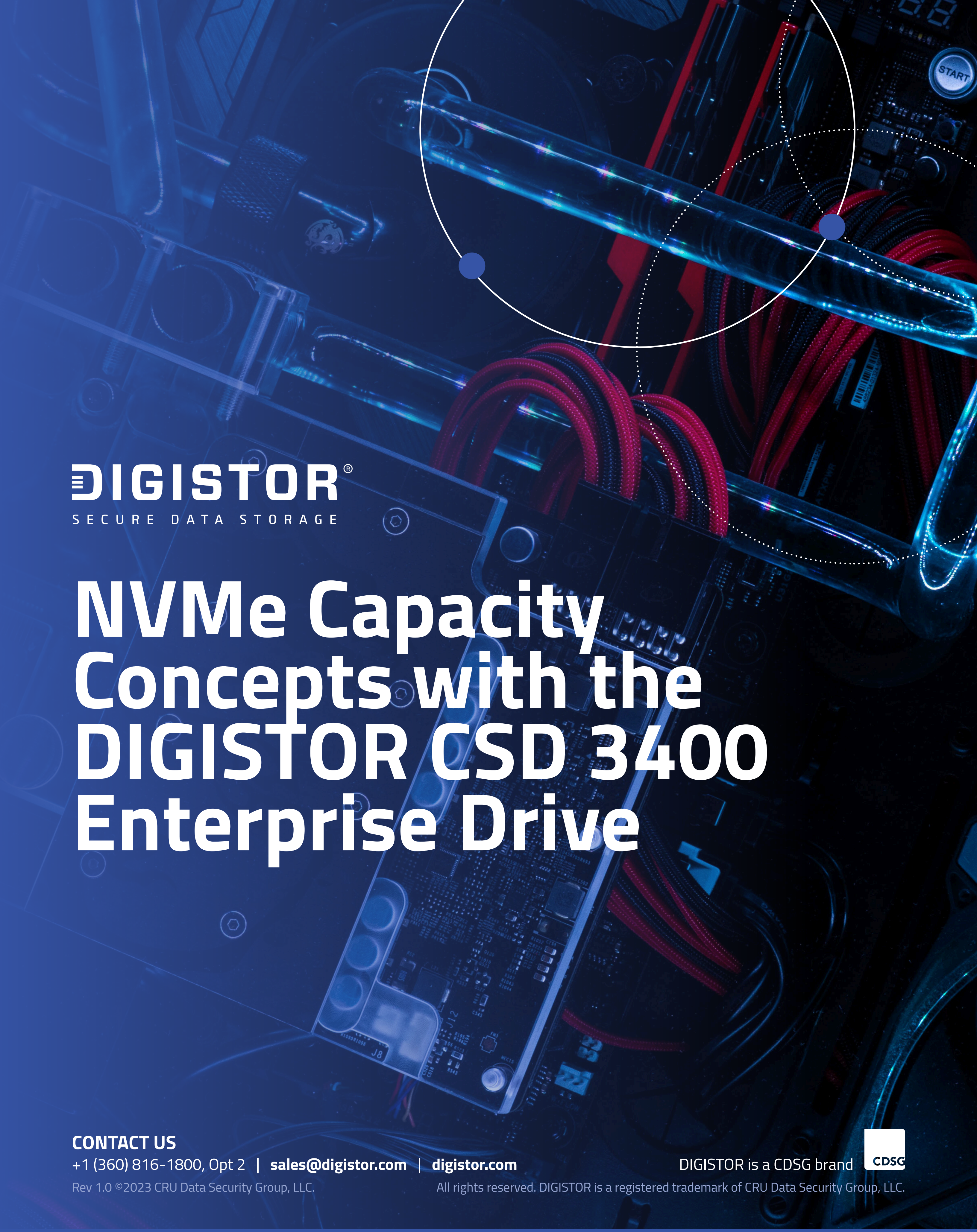


DIGISTOR®

SECURE DATA STORAGE

NVMe Capacity Concepts with the DIGISTOR CSD 3400 Enterprise Drive

CONTACT US

+1 (360) 816-1800, Opt 2 | **sales@digistor.com** | **digistor.com**

DIGISTOR is a CDSG brand

ABOUT THIS DOCUMENT

This document provides a comprehensive explanation of industry-standard capacity concepts and the capacity reporting mechanisms provided by the NVMe specification. While applicable to any operating system, the examples included in this document are based on Linux using the industry standard NVMe management CLI (nvme-cli), available on GitHub at https://github.com/linux-nvme/nvme-cli and the `blockdev` command line tool (https://man7.org/linux/man-pages/man8/blockdev.8.html).

DOCUMENT VERSION HISTORY

Version	Date (MM/DD/YYYY)	Change Description
1.0	11/13/2023	The initial release of this document.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. PHYSICAL AND USER CAPACITIES

A typical NAND Flash-based NVMe SSD comprises an NVMe IO controller, and the collection of NAND die connected to it. Each die contributes a share of the total storage capacity available to the IO controller. It is the fundamental responsibility of the IO controller to make this collection of independent die appear as a monolithic pool of data storage to the host.

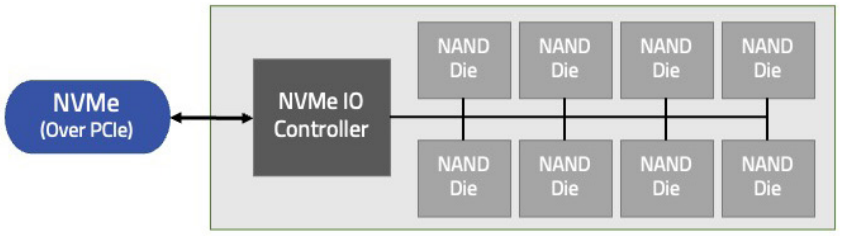
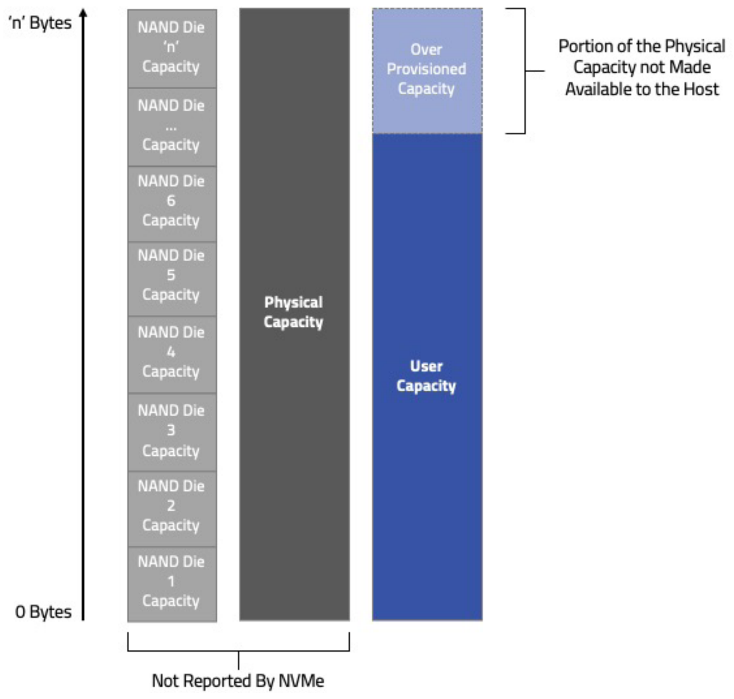


Figure 1: Basic Composition of an NVMe SSD

The quantity of NAND die (and the way they are connected to the controller) significantly impacts the SSD's performance characteristics. In general, higher die counts lead to better performance characteristics. However, as NAND die density continues to increase, the quantity of die needed to provide the same amount of data storage decreases. Consequently, the performance characteristics of a NAND Flash-based NVMe SSD at any fixed capacity point tend to decline over time. That is, capacity and performance are intertwined.



The total capacity of all NAND die comprising the SSD is called physical capacity. The physical capacity of an SSD is not reported to the host by any standard NVMe mechanisms. The SSD vendor decides how much physical capacity it makes available to the host, which becomes the user or usable capacity. The rest is held back as "over-provisioned" capacity. Over-provisioned capacity is the difference between the physical capacity and the user capacity. The quantity of over-provisioned capacity directly determines garbage collection's maximum intensity and, thus, the SSD's endurance and sustained write performance. A vendor may choose to increase the physical capacity or reduce the user capacity to improve the endurance and sustained write characteristics of an SSD. This is the difference between so-called "read intensive" and "write intensive" SSDs.

Figure 2: Physical and User Capacities

1.1. ADVERTISED CAPACITY AND BYTE PREFIX ISSUES

The advertised capacity of an SSD is the total capacity claimed by the SSD vendor and is an expression of the user capacity. The advertised capacity value is found on the SSD label. This value is always a base 10 (power of ten) in the storage industry, meaning that a kilobyte is 1000 bytes (10^3), a megabyte is 1,000,000 (10^6) bytes and is often confused with base 2 (power of two) values where a kilobyte is 1024 bytes (2^{10}), a megabyte is 1,048,576 (220), etc. Many software tools report in base 2 units such that the total capacity of an SSD reported by software is often *lower* than the advertised capacity (or what's printed on the label). In other words, whether a kilobyte means 1000 or 1024 bytes is often ambiguous.

The International Electrotechnical Commission (IEC), in collaboration with the Institute of Electrical and Electronic Engineers (IEEE) and the International Organization for Standardization (ISO), introduced new units to disambiguate base 10 and base 2 units. Under IEC standards, a kilobyte (KB) is 1000 bytes, and a kibibyte (KiB) is 1024 bytes. IEC units have become common, but adoption is far from universal, and the possibility of misinterpreting the meaning of a kilobyte remains.

SI Unit	SI Bytes	IEC Unit	IEC Bytes	SI-to-IEC Delta
Kilobyte	10^3	Kibibyte	2^{10}	2.4%
Megabyte	10^6	Mebibyte	2^{20}	4.9%
Gigabyte	10^9	Gibibyte	2^{30}	7.4%
Terabyte	10^{12}	Tebibyte	2^{40}	10.0%
Petabyte	10^{15}	Pebibyte	2^{50}	12.6%
Exabyte	10^{18}	Exbibyte	2^{60}	15.3%

Table 1 – SI vs. IEC Unit Prefixes and Values

1.2. USER-CAPACITY REPORTING IN NVME

In an NVMe SSD, the total user capacity of the SSD is expressed in bytes. This avoids the ambiguity of the unit base or the details of the internal SSD geometry. This value can be found in the Identify Controller Data Structure under the TNVMCAP (Total NVM Capacity) field.

```
# Query total capacity on a 3.84TB SSD
$ nvme id-ctrl /dev/nvme0 | grep nvmcap
tnvmcap: 3840755982336 # Total usable bytes
unvmcap : 0
```

The UNVMCAP (Unallocated NVM Capacity) indicates how many bytes of capacity is currently not reachable by the host. While the TNVMCAP value is fixed, the UNVMCAP can range from 0 to TNVMCAP. Typically, UNVMCAP is zero to maximize the capacity of the SSD.

2. BLOCKS AND SECTORS

Although the user capacity of an NVMe SSD is reported in bytes, NVMe SSDs provide block storage. Block storage devices do not provide an address for each byte of capacity but rather group bytes into larger units called sectors that are addressed using a logical block address (LBA). A sector is the minimum unit of access to the disk, and each sector has a corresponding LBA. The SSD guarantees that for a given LBA, it will always return the data last written to that address; however, the underlying physical location of the data can change due to rewrites or garbage collection.

2.1. LOGICAL VS. PHYSICAL SECTORS

The logical sector size is the host's minimum access granularity. It may differ from the SSD's internal physical sector size, as the physical sector size is the IO controller's minimum access granularity to the NAND media. The logical sector size can be smaller or equal to the physical-sector size. There is no advantage to using a logical-sector size larger than the physical-sector size; thus, the product does not support these configurations.

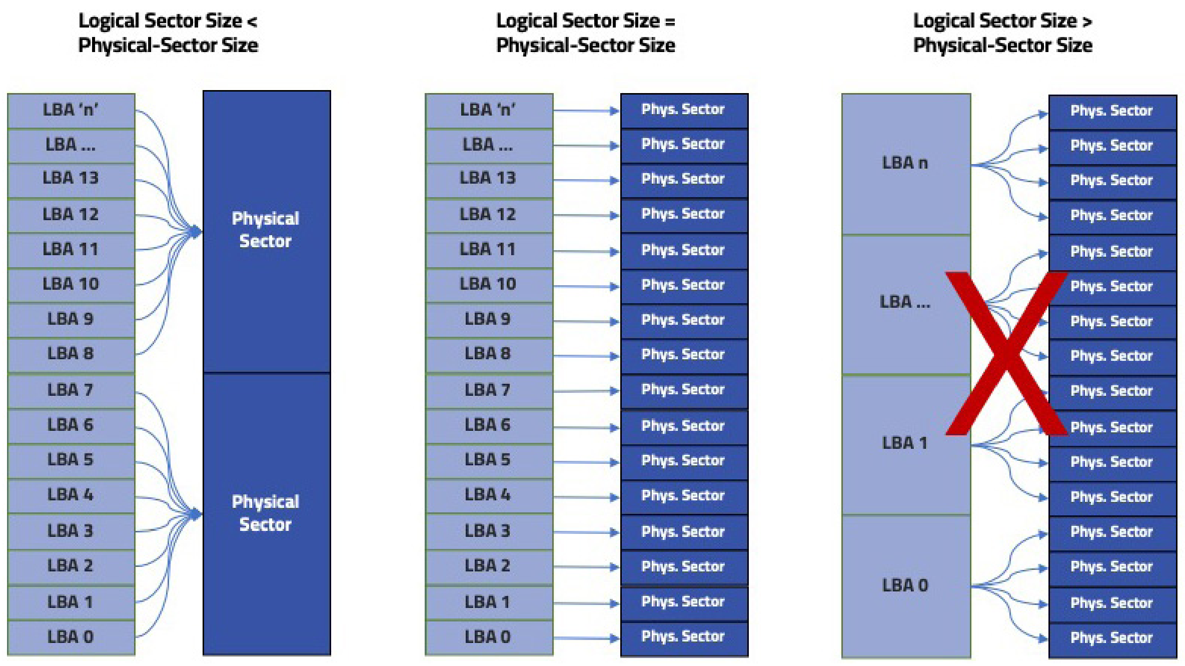

Logical Sector Size < Physical-Sector Size
Logical Sector Size = Physical-Sector Size
Logical Sector Size > Physical-Sector Size
LBA 'n'
LBA ...
LBA 13
LBA 12
LBA 11
LBA 10
LBA 9
LBA 8
LBA 7
LBA 6
LBA 5
LBA 4
LBA 3
LBA 2
LBA 1
LBA 0
Physical Sector
Phys. Sector
LBA n
X

Figure 3: Logical to Physical-Sector Relationships

2.2. QUERYING LOGICAL AND PHYSICAL SECTOR SIZES

It is mandatory for NVMe devices to report the logical-sector size but optional to report details about the internal-sector size configuration. The blockdev command can be used to inspect the logical and physical-sector sizes of a block device.

```
# Query the logical sector size in bytes
$ sudo blockdev --getss /dev/nvme0n1 512

# Query the physical sector size in bytes
$ sudo blockdev --getpbsz /dev/nvme0n1
4096
```

For the device in the case above, the logical-sector size is 512 bytes, and the physical sector size is 4 KiB. We can use the Identify Namespace data structure to obtain these sizes from the NVMe device. As an NVMe device may support more than one logical-sector size, a list of supported LBA formats (LBAF) is provided along with a field – Formatted LBA Size (FLBAS) - that indicates the index of the format in use.

```
# Query the FLBAS index
$ sudo nvme id-ns /dev/nvme0n1 | grep flbas
flbas : 0

# Show the list of supported LBAFs
$ sudo nvme id-ns /dev/nvme0n1 | grep lbaf
nlbaf : 1 # Zeroes-based value, meaning 2 are supported
lbaf 0 : ms:0 lbads:9 rp:0 (in use)
lbaf 1 : ms:0  lbads:12 rp:0
```

The LBA data size (LBADS) is reported as '9' (index 0) in the above case. This is an exponent value with base 2 (i.e., 9 means 29 or 512-bytes).

Determining the physical-sector size is more complex and can be inferred from several different values in the Identify namespace data structure. The most straightforward method is to query the preferred IO (NPWA) alignment.

```
# Query the NPWA
$ sudo nvme id-ns /dev/nvme0n1 | grep npwa
npwa: 7 # Zeroes-based value
```

In the above, a value of '7' is a zeroes-based value counted in units of the logical-sector size (i.e., 8 * 512 = 4 KiB). Note: the Linux NVMe driver derives the physical-sector size value from other Identify Namespace Data Structure fields.

2.3. BLOCK SIZES AND ACCESS ALIGNMENT

An IO command contains a single LBA and a count of contiguous logical sectors in the LBA space. For example, if the host wants to write 8 LBA's worth of data starting at LBA 0, then the LBA in the IO command will be set to 0 with the length set to 8, and LBAs 0 through 7 will be written to the disk. The number of logical sectors in an IO command is called the block size. The block size is always a multiple of the logical sector size.

When the logical sector size is smaller than the physical sector size, there is always the potential for misalignment. Misalignment occurs when a host block requests only a portion of a physical sector. In the case of a read, this results in wasted read data and, thus, degraded performance. In the case of a write, the entire contents of a physical sector must be rewritten (even though...only a portion of the data is changed) which forces read-modify-write behavior. An entire physical sector must be read, a portion of the sector data is then updated, and finally, the entire physical sector is written to disk. This not only impacts performance but also increases write amplification.

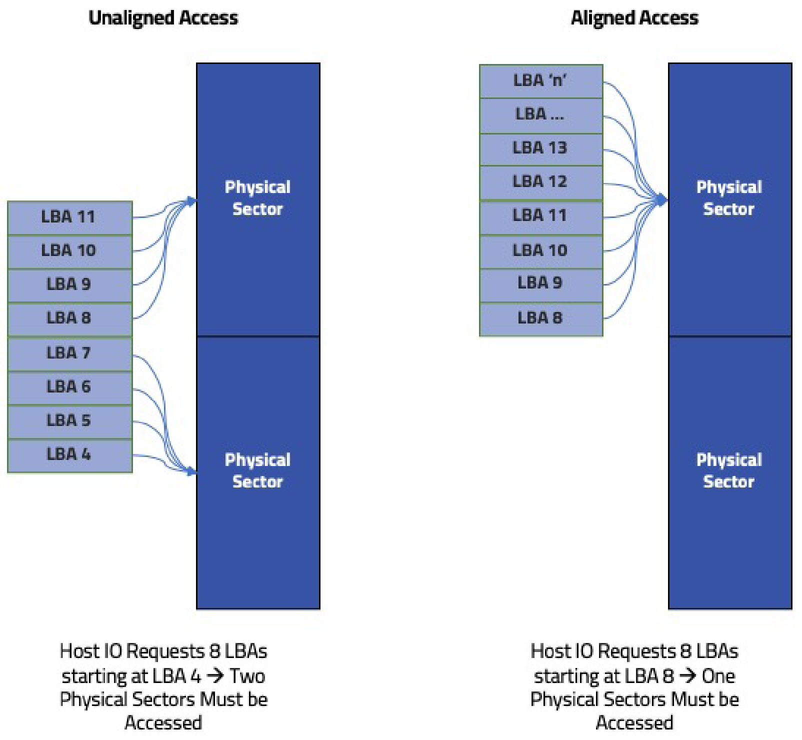


Figure 4: Access Alignment

With some care, the host can avoid misaligned writes. When the physical sector size matches the logical-sector size, there is no chance for misalignment.

2.4. COMMON SECTOR SIZES

Historically, SSDs contained 512 bytes per sector. As the drives grew larger, it became commonplace to see 4096-byte (4KiB) sectors, sometimes referred to as “advanced format” HDDs. From the outset, the internal-sector size of SSDs has most commonly been 4KiB. It’s a common misconception that this results from the NAND page size or some other attribute of NAND Flash. Indeed, early NAND Flash SSDs commonly used NAND devices with a 4KiB page size; however, this was a mere coincidence, and the typical page size of NAND has increased over time. 4KiB sectors are commonly used in SSDs to reduce the size of the logical to physical (L2P) map. The logical-to-physical map translates an LBA to a physical location with the NAND Flash. Using a 4KiB sector-size yields an eight-fold decrease in map size compared to 512-byte sectors. If the map is stored in DRAM, this represents considerable cost savings. As SSD capacities continue to grow, sector sizes larger than 4KiB will become more commonplace to keep the size of the logical to physical map manageable.

Some SSDs support additional “protection information” for a logical sector-level checksum. For example, a 512-byte sector can be augmented with 8 additional checksum bytes. Thus, the sector size becomes 520 bytes, sometimes expressed as 512+8, to distinguish the separation of data bytes from protection information. Similarly, a 4 KiB logical sector may have 8 or even 64 bytes of additional bytes for protection information resulting in 4104-byte and 4160-byte sector sizes (or 4096+8 and 4096+64, respectively).

3. INDUSTRY STANDARDS FOR CAPACITY VALUES

The storage industry has established standards to ensure that the number of logical sectors available to the host for a given advertised capacity is the same across all vendors. For example, a 7.68 TB SSD from Vender A provides the same number of logical sectors as a 7.68 TB SSD from Vendor B.

The International Disk Drive and Equipment Materials Association (IDMEA) provides the most widely followed standard for drive capacity reporting in the LBA1-03 specification. The SFF-8447 specification is a newer standard advanced by the Storage and Networking Industry Association (SNIA) that extends the LBA1-03 specification. Both standards are publicly available:

- LBA1-03: http://www.idema.org/wp-content/downloads/2169.pdf
- SFF-8447: https://www.snia.org/node/4706

The specifications provide formulas that take the advertised capacity and logical sector size as inputs and output the appropriate sector count. The following table summarizes the exact byte counts (reflected in TNVMCAP) demanded by each standard for commonly advertised capacities.

Advertised Capacity in TB	**LBA1-03 TNVMCAP**	**SFF-8447 TNVMCAP**
1.60	1600321314816	
1.92	1920383410176	
2.00	2000398934016	
3.20	3200631791616	
3.84	3840755982336	
4.00	4000787030016	
6.40	6401252745216	
7.68	7681501126656	
8.00	8001563222016	
12.80	12802494652416	12800076283904
15.36	15362991415296	15360950534144
16.00	16003115606016	16000900661248

Advertised Capacity in TB	LBA1-03 TNVMCAP	SFF-8447 TNVMCAP
25.60	25604978466816	25600152567808
30.72	30725971992576	30720827326464
32.00	32006220374016	32000727580672
51.20	51209946095616	51200305135616
61.44	61451933147136	61440580911104
64.00	64012429910016	64000381419520

Table 2 – Industry Standard TNVMCAP Values

Consistently, both standards provide slightly more capacity than a pure base-10 conversion of the advertised capacity.

4. NVME NAMESPACES

In the simplest case, the total user capacity is addressed using a single, continuous LBA range. That is, all the user capacity is available in a single namespace. This is the default shipping configuration for virtually all NVMe SSDs. NVMe optionally allows an SSD to contain more than one namespace. For example, the user capacity can be divided into two-equal size namespacies, each addressing half of the total user capacity. In this case, both namespaces have independent LBA ranges starting from 0. Operating systems treat each namespace as a block device (or logical drive). Therefore, it is possible to use the namespaces in independent fashion (like drive partitions).

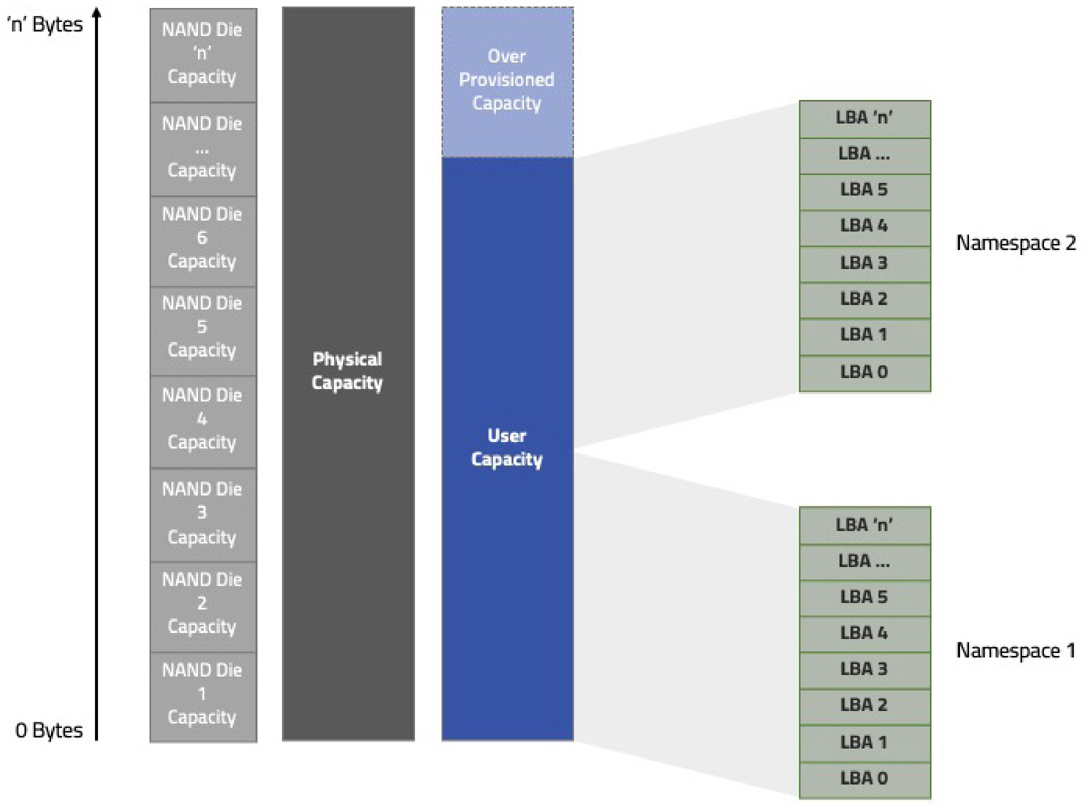


Figure 5: A Multi-Namespace Configuration

With the potential existence of more than one namespace, all NVMe IO commands specify the LBA and sector count and the namespace identifier (NSID).

4.1. NAMESPACE MANAGEMENT

The optional NVMe namespace-management feature set is used to modify the namespace configuration of the SSD. Only the single-default namespace is supported if an SSD does not support Namespace Management. With Namespace Management, an existing namespace can be "detached" from the controller (removed from host access), deleted, or re-attached. If a namespace is deleted, its previously allocated capacity will be reflected in the UNVMCAP field of the Identify Controller data structure. This indicates that capacity is available to create a new namespace (or it can be left aside as additional over-provisioned capacity).

Namespace creation involves specifying several critical parameters of the namespace:

1. The logical sector size of the namespace (FLBAS) is not arbitrary and is selected from a list of supported options provided by the IO controller. All namespaces may require the same logical sector size configuration depending on the IO controller.
2. The **Namespace Size** (NSZE) is the number of LBAs available to the host.
3. The **Namespace Capacity** (NCAP) is the actual capacity provided to support the Namespace Size. The namespace is considered thin provisioned when the Namespace Capacity is less than the Namespace Size. Thin provisioning is also an optional NVMe feature.

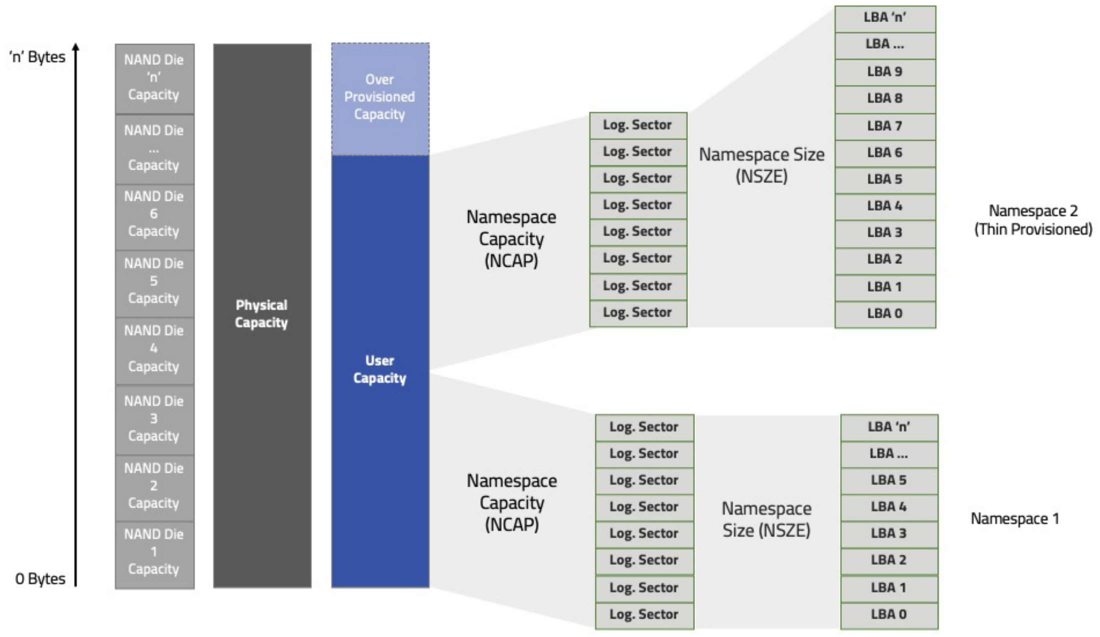


Figure 6: Thin Provisioned and Non-Thin Provisioned Namespaces

When the NCAP is less than the NSZE, an additional parameter called **Namespace Usage (NUSE)** is used to reflect the current capacity utilization of the namespace. Monitoring the Namespace Usage (NUSE), a parameter is required to ensure that the Namespace Capacity (NCAP) is not exceeded.

4.2. QUERYING NAMESPACE CAPACITY PARAMETERS

The three namespace capacity parameters – NSZE, NCAP, and NUSE – are reported in the Identify Namespace Data Structure.

```
# Query the NSZE
$ sudo nvme id-ns /dev/nvme0n1 | grep nsze
ncap      : 0x1bf1f72b0 # count is in logical sectors

# Query the NCAP
$ sudo nvme id-ns /dev/nvme0n1 | grep ncap
ncap      : 0x1bf1f72b0 # count is in logical sectors

# Query the NUSE
$ sudo nvme id-ns /dev/nvme0n1 | grep nuse
nuse      : 0 # count is in logical sectors
```

4.3. QUERYING FOR NAMESPACE-MANAGEMENT AND THIN-PROVISIONING SUPPORT

Support for namespace management is indicated in the Identify-Controller data structure in the Optional Admin Command Support (OACS) field.

```
# Query for Namespace Management Support
$ sudo nvme id-ctrl /dev/nvme0 -H | grep "NS Management"
[3:3]: 0x1 NS Management and Attachment Supported
```

The Identify-Namespace data structure in the Namespace Feature (NSFEAT) field indicates support for thin provisioning. Since namespace management is supported, a namespace identifier of all ones will indicate the capabilities shared between all namespaces.

```
# Query for Thin Provisioning Support
$ sudo nvme id-ns /dev/nvme0 -n 0xffffffff -H | grep Thin
[0:0] : 0x1 Thin Provisioning Supported
```

The specific field name is referred to as **THINP** in the NVMe specification. Lastly, the upper limit on the number of namespaces supported by the IO Controller is indicated in the **identify-controller data structure's Number of Namespaces (NN) field.**

```
# Query the Maximum Number of Namespaces
$ sudo nvme id-ctrl /dev/nvme0 | grep nn
nn        8
```

In the above case, 8 namespaces can be supported concurrently.

4.4. ATTACHING AND DETACHING NVME NAMESPACES

Because a single namespace, including the total-user capacity, is the default shipping configuration for all SSDs, there is typically no space left (as indicated in UNVMCAP) to create more namespaces without deleting the default namespace. Deleting and creating namespaces are both two-step processes in NVMe. Deleting a namespace must first be "detached" from the IO controller. It may then be deleted.

Similarly, after a namespace is created, it must be "attached" to the IO controller. This is because a namespace can be attached to multiple IO controllers (for multi-pathing). The basic flow of namespace creation is shown in Figure 11.

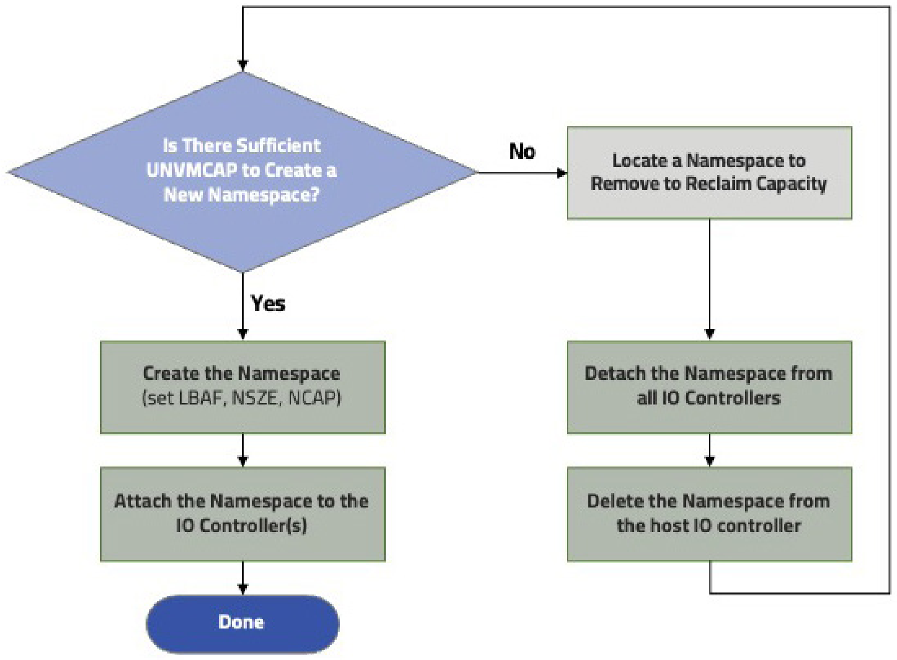

Is There Sufficient UNVMCAP to Create a New Namespace?
No
Locate a Namespace to Remove to Reclaim Capacity
Yes
Create the Namespace (set LBAF, NSZE, NCAP)
Detach the Namespace from all IO Controllers
Attach the Namespace to the IO Controller(s)
Delete the Namespace from the host IO controller
Done

Figure 7: Namespace Modification Flow

4.5. RULES FOR NAMESPACE CREATION

When creating a namespace, the logical sector size (through FLBAS), Namespace Size (NSZE), and Namespace Capacity (NCAP) are provided by the host. The value for FLBAS must be in the list of supported LBA format (LBAF) options. That is, arbitrary sector sizes are not supported. The Namespace Size (NSZE) and Namespace Capacity (NCAP)-parameters should be allocated according to the preferred allocation granularity of the IO controller. These can be found in the Namespace Granularity List-data structure.

```
# Query the preferred namespace granularity
$ sudo nvme id-ns-granularity /dev/nvme0n1
Identify Namespace Granularity List:
ATTR: Namespace Granularity Attributes: 0x0
NUMD: Number of Descriptors: 0
Entry {0}
NSG: Namespace Size Granularity :0x1000
NCG: Namespace Capacity Granularity :0x1000
```

The namespace size granularity applies to the NSZE parameter, while the namespace capacity granularity value applies to the NCAP parameter. In the above case, both are 4 KiB. Intuitively, and it makes sense that the granularity of a namespace should be a multiple of the physical sector size.

Namespace IDs are assigned at the time of namespace creation. It is a best practice to avoid creating "holes" in namespace IDs. For example, if there are three namespaces with namespace IDs 1, 2, and 3. Deleting namespace 2 without deleting 3, creating a new namespace may result in a mismatch between the namespace ID and the number assigned in the device filename (e.g., /dev/nvme0n(n)). Removing all namespaces, then re-creating namespaces in the desired sequence is preferred.

4.6. NAMESPACE CONFIGURATION EXAMPLE

In the following example, an SSD with a single namespace is reconfigured with two namespaces with equal-capacity parameters.

```
# Query the controller-capacity values
$ sudo nvme id-ctrl /dev/nvme0 | grep nvmcap
tnvmcap : 3840755982336
unvmcap  : 0

# Query the number of existing namespaces
$ sudo nvme list-ns /dev/nvme0
[ 0]:0x1
```

There is one namespace and no UNVMCAP, indicating that the single namespace is using the total capacity of the SSD.

```
# Query the namespace-capacity values

# LBA format
sudo nvme id-ns /dev/nvme0n1 | grep lbaf
nlbaf : 1
lbaf 0 : ms:0 lbads:9 rp:0 (in use)
lbaf 1 : ms:0 lbads:12 rp:0

# Namespace Size
$ sudo nvme id-ns /dev/nvme0n1 | grep nsze
nsze      : 0x1bf1f72b0

# Namespace Capacity
$ sudo nvme id-ns /dev/nvme0n1 | grep ncap
ncap      : 0x1bf1f72b0

# Namespace Use
$ sudo nvme id-ns /dev/nvme0n1 | grep nuse
nuse      : 0
```

The single namespace uses 512-byte sectors with NSZE set equal to NCAP.

Next, the namespace is detached and deleted (note that this is destructive to data):

```
# Detach the namespace
$ sudo nvme detach-ns /dev/nvme0 -n 1
warning: empty controller-id list will result in no actual change in the namespace attachment detach-ns: Success, nsid:1

# Delete the namespace
$ sudo nvme delete-ns /dev/nvme0 -n 1
delete-ns: Success, deleted nsid:1
```

All the capacity in the SSD will now reflect as unallocated:

```
# Query the controller-capacity values
$ sudo nvme id-ctrl /dev/nvme0 | grep nvmcap
tnvmcap : 3840755982336
unvmcap  : 3840755982336
```

When creating new namespaces, the preferred granularity values of the controller should be respected.

```
# Query the preferred namespace granularity
$ sudo nvme id-ns-granularity /dev/nvme0n1
Identify Namespace Granularity List:
ATTR: Namespace Granularity Attributes: 0x0
NUMD: Number of Descriptors: 0
Entry {0}
NSG: Namespace Size Granularity :0x1000
NCG: Namespace Capacity Granularity :0x1000
```

Dividing the TNVMCAP in half yields 1,920,377,991,168 bytes; this value must have a granularity equal to 4 KiB per the preferred granularity of the controller. Since 1,920,377,991,168 / 4096 = 468,842,283 with no remainder, the granularity requirement is met.

```
# Create namespace 1
$ sudo nvme create-ns /dev/nvme0 --flbas=0 --nsze=3750738264 --ncap=3750738264 create-ns: Success, created nsid:1

# Create namespace 2
$ sudo nvme create-ns /dev/nvme0 --flbas=0 --nsze=3750738264 --ncap=3750738264 create-ns: Success, created nsid:2
```

The namespaces must be attached and rescanned to become visible to the host.

```
# Attach namespace 1
$ sudo nvme attach-ns /dev/nvme0 -n 1
warning: empty controller-id list will result in no actual change in the namespace attachment attach-ns: Success, nsid:1

# Attach namespace 2
$ sudo nvme attach-ns /dev/nvme0 -n 2
warning: empty controller-id list will result in no actual change in the namespace attachment attach-ns: Success, nsid:2create-
ns: Success, created nsid:2

# Rescan the namespaces
sudo nvme ns-rescan /dev/nvme0
```

Now, two equal size logical drives are ready for use.

```
# List block devices
$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
nvme3n1 259:6 0 1.7T 0 disk
nvme3n2 259:10 0 1.7T 0 disk
```

5. THIN PROVISIONING

The Namespace Size (NSZE) parameter determines the size of the namespace that is addressable by the host. At the same time, the Namespace Capacity (NCAP) describes the maximum quantity of data that can be stored in the namespace at any given time. If these are set to be equal, then there is no chance for the host to "overfill" the allocated capacity; however, this can result in stranded capacity. Stranded capacity can occur under two general scenarios. First, the expected disk utilization is low when multiple namespaces are created (and consequently multiple logical disks). In this case, fully provisioning each namespace will allocate capacity the host will never use. Second, the IO controller reduces data through compression or deduplication, reducing capacity usage. In this case, the host could potentially store more data than the NCAP indicates.

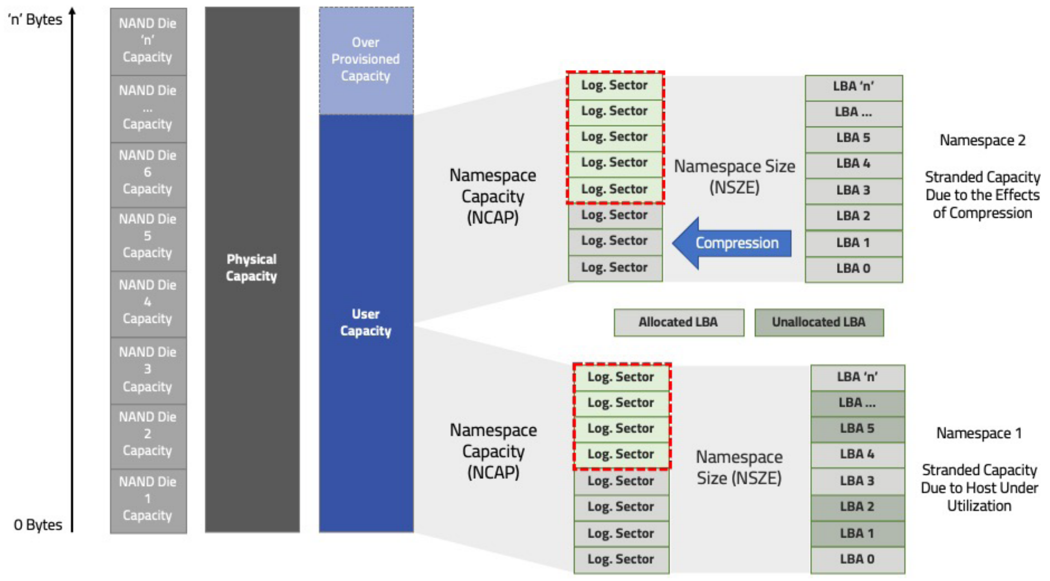


Figure 8: Stranded Capacity Outlined in Red

In either case, the stranded capacity problem is resolved by increasing the addressable size of the namespace without allocating all the underlying capacity needed to store a fully-populated namespace.

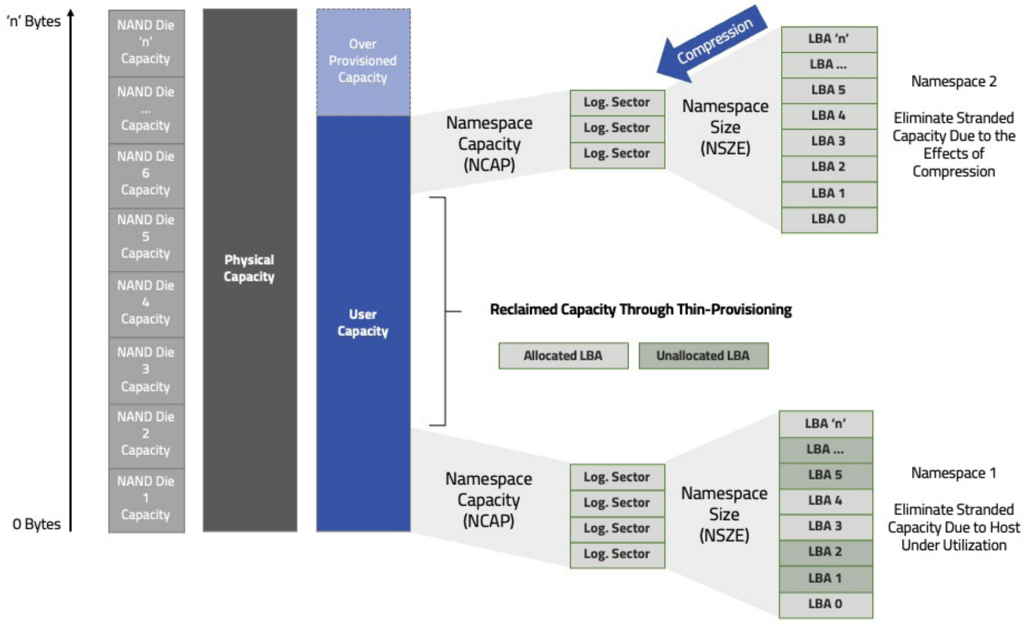


Figure 9: Using Thin Provisioning to Avoid Stranded Capacity

5.1. MONITORING NAMESPACE USAGE

When a namespace is thin provisioned, capacity can be used more efficiently; however, because the capacity visible to the host is larger than the underlying capacity, there exists the possibility of over flowing the namespace. The NVMe specification provides the Namespace Usage (NUSE) parameter in the Identify Namespace Data Structure to measure the actual Namespace Capacity utilization at any given time. Without thin provisioning, the host must monitor only the logical utilization of disks (e.g., file system utilization). With thin provisioning, hosts must also monitor the physical utilization through the Namespace Usage parameter.

5.2. DEALLOCATION AND TRIM

The Namespace Utilization increases when a logical sector is written (or allocated). Sectors can only be deallocated when trimmed (via the Dataset Management command) or optionally through a Write Zeroes command. An NVM Format operation with secure erase will deallocate all sectors in a namespace. Using thin provisioning requires a deallocation (or trim) strategy. This is commonly achieved at the filesystem level by mounting with the discard option enabled (enabling trim); however, for applications that use raw device access, the application must directly support trim operations.

6. NAMESPACE CREATION WITH EXTENDED CAPACITY

When the sum of the Namespace Sizes (NSZE) for all namespaces created does not exceed the TNVMCAP value, the values for the Namespace Size (NSZE) and Namespace Capacity (NCAP) are subject only to the minimum granularity requirements. When the sum of all Namespace Sizes (NSZE) exceeds the TNVMCAP value, the SSD operates in extended capacity mode, and the sum of Namespace Sizes (NSZE) across all namespaces must calculate to an integer gigabyte value following the IDEMA LBA1-03 specification.

The relationship between capacity and sector counts defined by IDEMA LBA1-03 are as follows:

For 512-byte logical sectors:
NSZE = (97,696,368) + (1,953,504 * (Desired Capacity in GB - 50))

For 4096-byte logical sectors:
NSZE = (12,212,046) + (244,188 * (Desired Capacity in GB - 50))

This rule applies only to the Namespace Size (NSZE). The Namespace Capacity (NCAP) values remain subject only to the granularity requirements.

When only a single-extended-capacity namespace is desired, a valid NSZE value can be obtained simply by calculating the appropriate value based on the sector size. With multiple namespaces in extended capacity, care must be taken to ensure that any namespace created that puts the device into extended capacity mode satisfies the added IDEMA LBA1-03 condition based on the sum of all Namespace Sizes (NSZE). The nature of the IDEMA LBA1-03 formulas complicates this. The sum of two or more valid NSZE values derived from the desired capacity in GB will not then sum to a total NSZE that calculates to a valid capacity value (i.e., an integer number of gigabytes).

6.1.OVER-FILL WITH EXTENDED CAPACITY

As the Namespace Usage (NUSE) exceeds the Namespace Capacity (NCAP), the SSD will begin to apply a progressively stronger write throttle. This "soft-landing" ensures that any necessary writes to recover from the over-fill condition can still be performed. For example, re-mounting a filesystem on an over-filled namespace. Nonetheless, proper capacity management techniques are strongly recommended to avoid over-fill conditions before they occur. This fundamentally begins with monitoring the Namespace Usage (NUSE) field. The specific framework is infrastructure dependent.

7. SUMMARY OF NVME CAPACITY CONCEPTS

The advertised capacity is the total capacity reported by an SSD vendor. The relationship of the advertised capacity to the number of available bytes on the SSD is governed by industry standards such that any SSD with the same advertised capacity will provide the same number of bytes for host storage. The advertised-usable capacity is reported as a base 10 value. Still, many software programs report capacity in base 2 values, which is potentially confusing. The base must be considered whenever a capacity value is reported using prefixes (e.g., KB or MB). IEC units (e.g., KIB vs. KB or MIB vs. MB) are one way to avoid confusion but are not universally adopted.

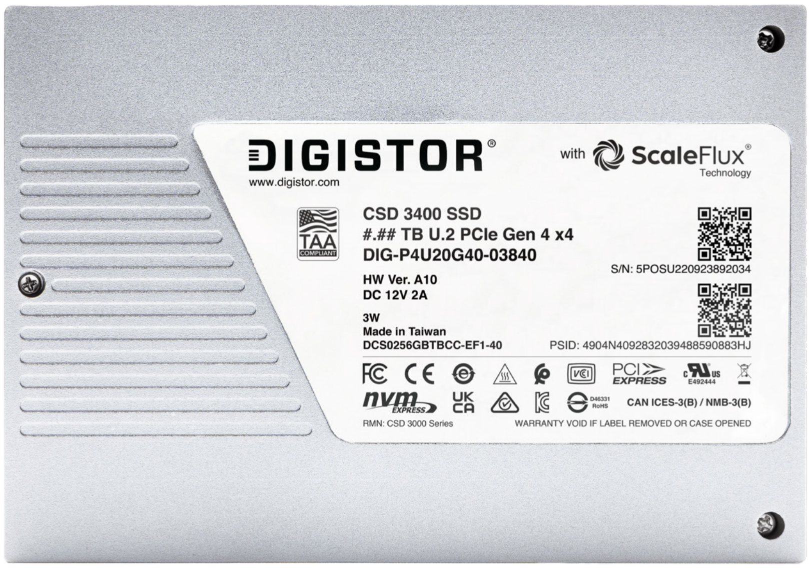


Figure 10: Advertised Capacity on the SSD Label

The physical capacity of a NAND-based NVMe SSD is the total quantity of NAND capacity available to the IO controller of the SSD. There is no standard way of querying this value in the NVMe specification. Two SSDs with the same advertised capacity may contain different amounts of physical capacity. The physical capacity is generally reported using base 2 prefixes if a vendor discloses.

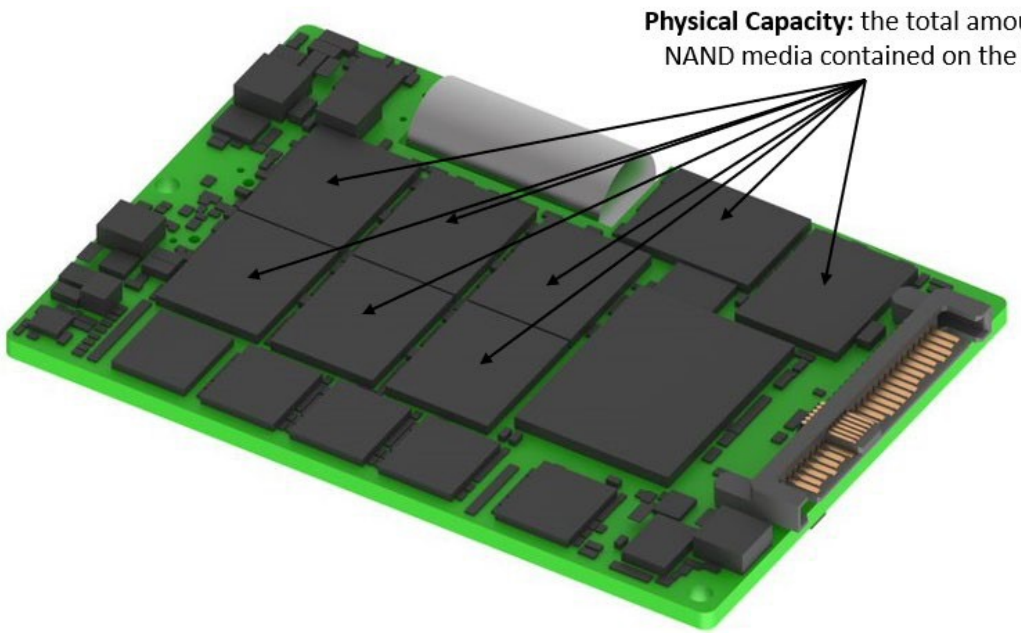

Physical Capacity: the total amount of NAND media contained on the SSD

Figure 11: Physical Capacity

The user capacity is the total capacity the SSD reports to the host. In NVMe, this is the TNVMCAP value in the Identity Controller data structure. This value is reported in bytes. The difference between the physical and user capacity is the built-in over-provisioned capacity of the SSD. SSDs with higher built-in over-provisioning will have higher sustained write performance and endurance than those with lower over- provisioning.

An NVMe SSD's capacity is exposed to the host in independent namespaces. A namespace is a linear range of logical block addresses (LBAs) that the OS treats as an independent-block device. The namespace format indicates the logical-sector size and is found in the Identify Namespace data structure (FLBAS) or reported by blockdev with the getss option. The logical sector size is the minimum unit of access to the namespace. Block size refers to the number of logical sectors accessed by the host in an IO command. If an SSD supports multiple logical sector sizes, a namespace can be reformatted to use a different logical-sector size with the NVM Format command.

The internal physical-sector size may be larger than the logical-sector size. When this is the case, best performance and endurance require that the host-block size is aligned to units of the internal-physical-sector size. The physical-sector size can be inferred from the Identify Namespace data structure values but is most easily obtained through blockdev with the --getpbsz option. The NVMe driver interprets the namespace parameters to populate the appropriate physical-sector-size value in this case.

If an SSD does not support the namespace management feature set, all the user capacity will be encapsulated in a single namespace. If the Namespace Management feature is supported, multiple namespaces can be created. The Identify Controller data structure (NN field) reports the total number of simultaneous namespaces supported. When namespaces are deleted, the freed capacity is reported in the UNVMCAP field of the Identify Controller data structure. The UNVMCAP field determines how much capacity is available to create new namespaces.

The three critical parameters specified when creating a namespace are the logical sector size (through LBAF -- the LBA format selection), the Namespace Size (NSZE), and Namespace Capacity (NCAP). The namespace size (NSZE) sets the number of LBAs exposed to the host. The namespace capacity (NCAP) determines how much storage is allocated to the namespace. To set NSZE greater than NCAP, the SSD must support thin provisioned namespaces. If the SSD does not support thin provisioning, then NSZE must equal NCAP.

Thin provisioning can improve storage efficiency by eliminating stranded capacity. If thin provisioning is deployed, the host must do the following:

1. Monitor the Namespace Usage (NUSE) parameter for all thin provisioned namespaces to ensure it does not exceed the Namespace Capacity (NCAP).
2. Employ a trim (deallocation) strategy to ensure that LBAs that are no longer needed can be freed by the SSD and lower the Namespace Usage (NUSE).