

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

The Accused Instrumentalities consist of Meta's data center servers, including Meta owned data centers, Meta operated data centers, and data centers operated on behalf of Meta, that support Flexible Data Placement (FDP) and/or NVMe 2.1 and associated hardware, software and firmware. For example, this includes but is not limited to, making data centers with Samsung's PM9D3 solid-state drives ("SSDs") and using data centers with Samsung's PM9D3 SSDs.¹ The Accused Instrumentalities infringe claims 1, 9, and 14 ("the Asserted Claims") of U.S. Patent No. 10,776,023 ("the '023 patent") in at least the following exemplary ways.

Meta infringes the Asserted Claims under 35 U.S.C. § 271(a) by making and/or using the Accused Instrumentalities in the United States. Meta makes the Accused Instrumentalities when it combines a storage device, such as an SSD, that supports FDP and/or NVMe 2.1 with a Meta server. Once the storage device is installed, the resulting system infringes as it contains each element of the asserted claims. Further, once the storage device is installed and the server is powered on for Meta's usage, the resulting usage infringes as it contains each element of the asserted claims.

Meta's CacheLib, a software running on Meta data center servers, provides an example of Meta benefitting and/or taking advantage of the Accused Instrumentality's FDP functionality. To the extent it is argued that the asserted claims require the FDP functionality to be invoked and/or used, CacheLib independently demonstrates such infringement. CacheLib causes Meta's servers to invoke FDP as described in the contentions below.

These infringement contentions are provided based on information obtained to date and may not be exhaustive. Gaea's investigation of Meta's infringement is ongoing. Gaea reserves the right to seek to supplement and/or amend these disclosures to identify additional accused instrumentalities and/or to further identify where each element of each asserted claim is found in each accused instrumentality, including on the basis of discovery obtained from Meta and from third parties during the course of this litigation.

Discovery is ongoing, including identification of the specific storage devices used by Meta and the storage devices' support for various identified functionality elements of FDP and/or NVMe 2.1.

¹ <https://semiconductor.samsung.com/news-events/tech-blog/nvme-fdp-a-promising-new-ssd-data-placement-approach/>

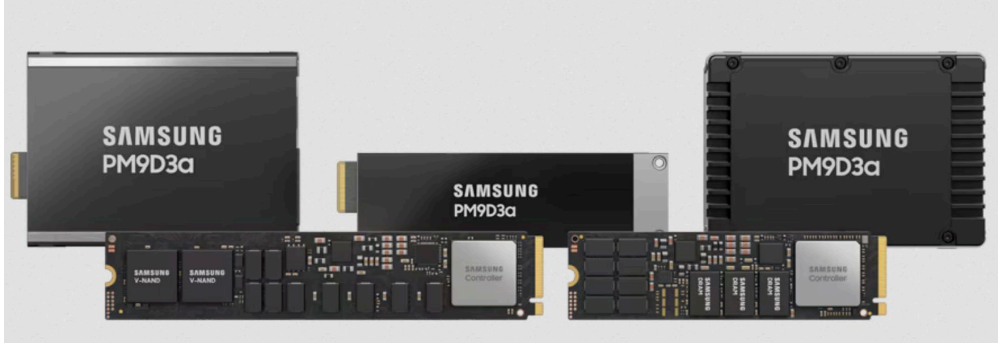
Exhibit A: Meta’s Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

’023 Patent Claim 1	’023 Patent Accused Instrumentalities
1[P] An apparatus comprising:	The Meta Accused Instrumentalities are an apparatus. For example, the Accused Instrumentalities include data center servers that include SSDs which together are devices or systems. For example, public documentation indicates that Meta’s Accused Instrumentalities include Samsung SSDs: “At Samsung, we enabled FDP on Meta (Facebook)’s CacheLib deployments using Samsung PM9D3 SSD. This work has been merged with the mainstream CacheLib code.”² To the extent Meta’s Accused Instrumentalities include other storage devices that support FDP and/or NVMe 2.1, the Accused Instrumentalities that include these other storage devices are further examples of the Meta Accused Instrumentalities consisting of an apparatus. Discovery into the specific storage devices, including but not limited to SSDs, used by Meta is ongoing.
1[A] a storage media;	The Meta Accused Instrumentalities include a storage media. For example, within each SSD there is at least one solid-state non-volatile flash memory integrated circuit for data storage. For example, public documentation indicates that Meta’s data centers include Samsung SSDs, including but not limited to the Samsung PM9D3 SSD, which contains NAND and/or V-NAND which is a storage media³.

² <https://semiconductor.samsung.com/news-events/tech-blog/nvme-fdp-a-promising-new-ssd-data-placement-approach/>

³ <https://image.semiconductor.samsung.com/image/samsung/p6/semiconductor/products/pf-layout/spec-ssd-dcssd-pm9d3a.png>
(showing “Samsung V-NAND”)

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

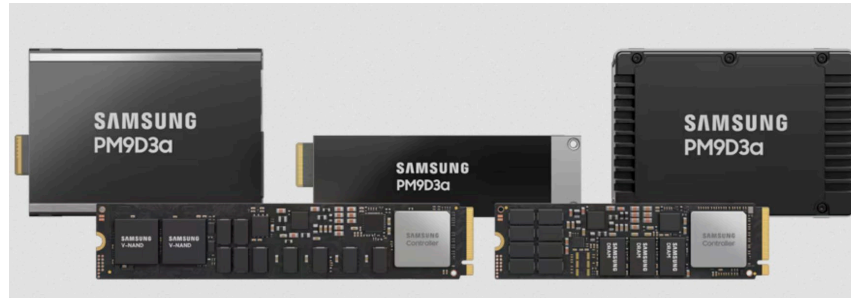
	 As another example, Samsung's PM9D3a is described as using "... Samsung's eight-generation three bit V-NAND technology ..."⁴ To the extent Meta's Accused Instrumentalities utilize other storage devices that support FDP and/or NVMe 2.1, the Accused Instrumentalities that include these other storage devices are further examples of the Meta Accused Instrumentalities that includes a storage media. Discovery into the specific storage devices, including but not limited to SSDs, used by Meta is ongoing, including the specific storage media used by the SSD.
1[B] a device controller operatively coupled with the storage media	The Meta Accused Instrumentalities include a device controller operatively coupled with the storage media and are configured such that it meets the subsequent claim limitations below. For example, within each SSD there is device controller operatively coupled with the storage media. As an example, Samsung SSDs such as the Samsung PM93D3a include a device controller as shown in the Samsung image below labeled "SAMSUNG CONTROLLER"⁵:

⁴ <https://semiconductor.samsung.com/news-events/tech-blog/samsung-pm9d3a-solid-state-drive/>

⁵ <https://image.semiconductor.samsung.com/image/samsung/p6/semiconductor/products/pf-layout/spec-ssd-dcssd-pm9d3a.png>

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

configured to at least:



As another example, Samsung documentation⁶ describes a device controller architecture that consists of Software and Hardware components as shown below:

(showing “Samsung V-NAND”)

⁶ Samsung White Paper “Power Loss Protection in SSDs” at pg 3.

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

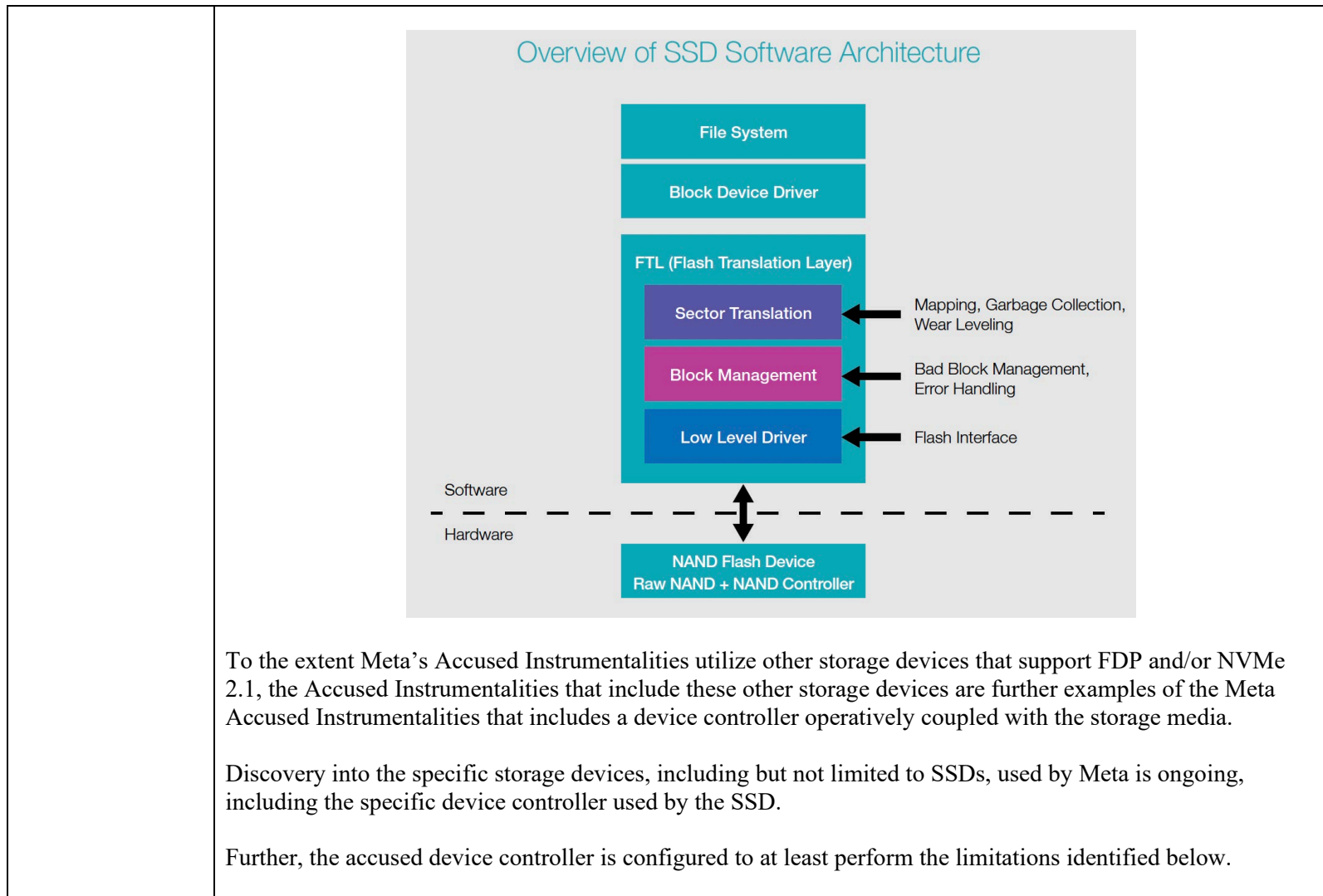


Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

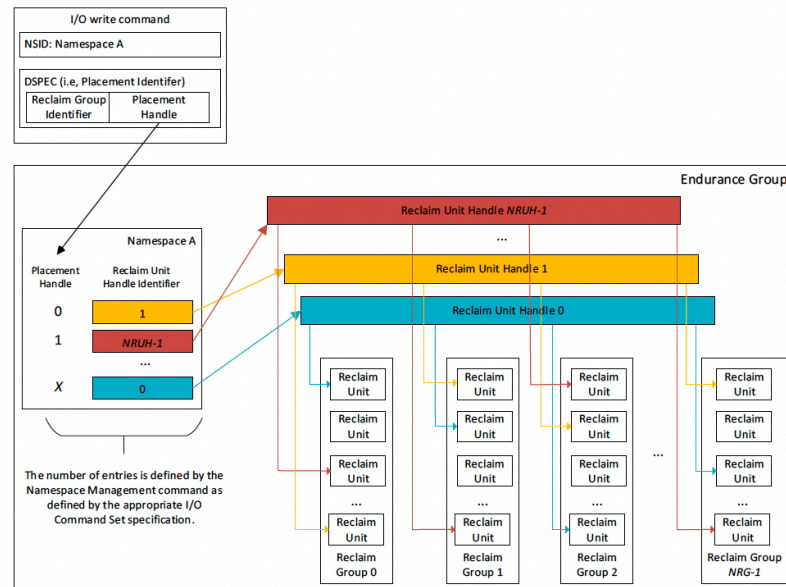
1[B1] receive, from a host, at least one storage device policy, wherein the at least one storage device policy indicates at least a recording method for writing content to the storage media;	The Meta Accused Instrumentalities include a device controller operatively coupled with the storage media and configured to receive from a host, at least one storage device policy, wherein the at least one storage device policy indicates at least a recording method for writing content to the storage media. For example, the NVMe 2.1 specification includes the Flexible Data Placement (FDP) functionality⁷: 8.1.10 Flexible Data Placement 8.1.10.1 Flexible Data Placement Overview The Flexible Data Placement (FDP) capability is an optional capability which allows the host to reduce write amplification by aligning user data usage to physical media. The controller supports log pages which indicate the status of FDP, statistics about the FDP operation, and information the host uses to detect and correct usage patterns that increase write amplification. The scope of the Flexible Data Placement capability is an Endurance Group. The host enables or disables Flexible Data Placement by issuing a Set Features command specifying the Flexible Data Placement feature (refer to section 5.1.25.1.20) and the FDP configuration (refer to section 5.1.12.1.28) to be applied to an Endurance Group. The host is required to delete all namespaces associated with the specified Endurance Group before modifying the value of the Flexible Data Placement feature (e.g., transitioning from disabled to enabled). If a Set Features command specifies: • the Flexible Data Placement feature; • an Endurance Group in which one or more namespaces exist; and • a different value from the current value of that feature for that Endurance Group, then the controller shall abort the command with a status code of Command Sequence Error. If a Set Features command is successful and changes the Feature value, then: • all events in the FDP Events log page (refer to section 5.1.12.1.31) are cleared; and • all fields in the FDP Statistics log page (refer to section 5.1.12.1.30) are cleared to 0h. Refer to section 3.2.4 for the logical view of the non-volatile storage capacity for the Endurance Group with Flexible Data Placement enabled. The Namespace Management command (refer to section 8.1.15) is used to create a namespace within the Endurance Group. All namespaces in an Endurance Group with Flexible Data Placement enabled shall be associated with the NVM Command Set (refer to Figure 311) (i.e., the Command Set Identifier (CSI) field in the Namespace Management command shall be cleared to 00h). The capacity for the user data stored by a write command to a namespace is allocated from the Reclaim Unit referenced by the Reclaim Unit Handle and Reclaim Group specified by that write command. The user data for a namespace is allowed to be in any stored Reclaim Unit within any Reclaim Group within the Endurance Group (refer to Figure 15). The Namespace Management command specifies a Placement Handle List (refer to the Namespace Management command in the appropriate I/O Command Set specification). Each entry in that list specifies the Reclaim Unit Handle associated with the Placement Handle for that entry. The Placement Handles are numbered from 0h to the number of entries in the list minus 1. Figure 616 shows an example where Namespace A associates Reclaim Unit Handle 1 with Placement Handle 0. A specific Reclaim Unit Handle
---	--

⁷ See, e.g., NVMe 2.1 Base Specification at § 8.1.10.

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	is not allowed to be associated with more than one Placement Handle per namespace. There are no other requirements on which Reclaim Unit Handle is associated with which Placement Handle. Namespaces are allowed to utilize the same (i.e., share) Reclaim Unit Handle but may have restrictions defined in the applicable I/O Command Set specification. The host is required to enable the Data Placement Directive (refer to section 8.1.8.4) to submit write commands that specify a Placement Identifier (refer to Figure 282 and Figure 283) in the DSPEC field. That Placement Identifier uniquely specifies the Reclaim Unit in which to place the user data (refer to Figure 616). The controller determines the Reclaim Unit by using the Reclaim Unit Handle associated with the Placement Handle that was specified by the host when the namespace was created. For example, if the write command in Figure 616 specifies a Reclaim Group Identifier value of 2h and a Placement Handle value of 1h, then the host is targeting the user data for that write command to be placed into the Reclaim Unit associated with Reclaim Group 2h and Reclaim Unit Handle <i>NRUH-1</i>. A Placement Identifier is invalid if: • the Reclaim Group Identifier value is greater than or equal to the number of Reclaim Groups supported by the enabled FDP configuration (refer to the NRG field in Figure 280); or • the Placement Handle is greater than or equal to the number of Placement Handles supported by the Namespace Management command of the applicable I/O Command Set specification. If the Placement Identifier in the DSPEC field of a write command is invalid, then the controller shall: • select the Reclaim Group and Reclaim Unit Handle accessible by the specified namespace to determine the placement of the user data for that write command; and • generate an Invalid Placement Identifier FDP Event if that event is enabled for the selected Reclaim Unit Handle. To receive the FDP event, the host should enable the Invalid Placement Identifier FDP Event on all Reclaim Unit Handles. Any write command to a namespace that exists in an Endurance Group with Flexible Data Placement enabled that does not specify the Data Placement Directive uses the Placement Handle value 0h and the controller selects the Reclaim Group for that command. For example, if the write command does not specify the Data Placement Directive, then, as illustrated in Figure 616, Placement Handle value of 0h is associated with Reclaim Unit Handle 1h and the controller may place the user data into the Reclaim Unit associated any Reclaim Group using Reclaim Unit Handle 1h.
--	--

Figure 616: Flexible Data Placement Model



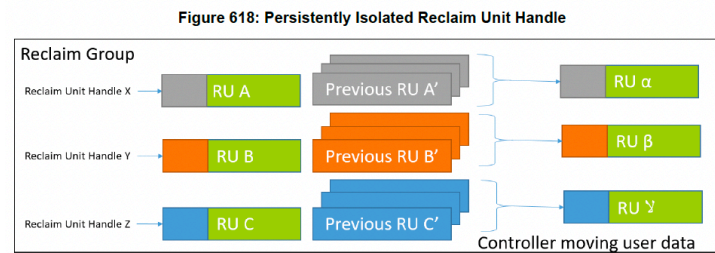
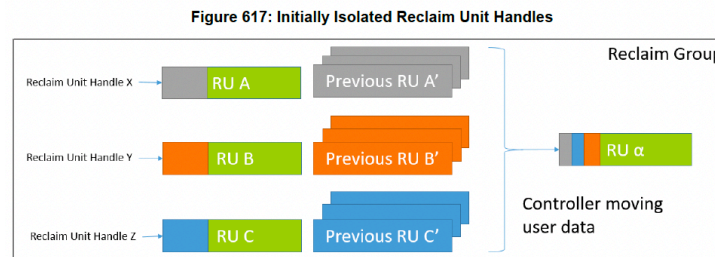
A write command that places user data into a Reclaim Unit causes reduction of the remaining capacity of that Reclaim Unit. If that Reclaim Unit is written to capacity, then the controller modifies the Reclaim Unit Handle referencing that Reclaim Unit to reference a different Reclaim Unit within the same Reclaim Group.

A controller may modify a Reclaim Unit Handle to reference a different Reclaim Unit as part of performing a sanitize operation (refer to section 8.1.24).

A Reclaim Unit shall only be referenced by one Reclaim Unit Handle. Therefore, a write command that utilizes a Reclaim Unit Handle results in the user data from that write command being initially isolated from the user data from any other write command that utilizes a different Reclaim Unit Handle. Each Reclaim Unit Handle has a type as defined in Figure 281 that identifies the isolation requirements of the user data moved by the controller to a different Reclaim Unit due to vendor specific controller operations (e.g., garbage collection).

A controller is allowed to move user data that was written by the host that utilized a Reclaim Unit Handle with an Initially Isolated type (Figure 281) to a different Reclaim Unit within the same Reclaim Group. In addition, the controller is allowed to move user data written by the host that utilized different Reclaim Unit Handles with an Initially Isolated type to that different Reclaim Unit within the same Reclaim Group as shown in Figure 617.

A controller is allowed to move user data that was written by the host that utilized a Reclaim Unit Handle with a Persistently Isolated type (Figure 281) to a different Reclaim Unit within the same Reclaim Group. However, the controller shall only move user data written by the host that utilized the same Reclaim Unit Handles with a Persistently Isolated type to the different Reclaim Unit within the same Reclaim Group as shown in Figure 618.



The result of a Namespace Management command that creates a namespace within the Endurance Group is that at least one Placement Handle is defined for that namespace (refer to the Namespace Management section in applicable I/O Command Set specification).

For any namespace created in an Endurance Group that has Flexible Data Placement enabled, the host may issue an I/O Management Receive command to obtain a list of Placement Handles and the associated Reclaim Unit Handles that are used by the Data Placement Directive in write commands.

If the Flexible Data Placement capability is supported, then the controller shall support the following:

- the Feature Identifiers Supported and Effects log page (refer to section 5.1.12.1.18);
- the I/O Management Receive command (refer to section 7.3) and the Reclaim Unit Handle Status Management Operation in that command (refer to Figure 561);
- the I/O Management Send command (refer to section 7.4) and the Placement Identifier Update Management Operation in that command (refer to Figure 565);
- the Flexible Data Placement feature (refer to section 5.1.25.1.20);
- the FDP Configurations log page (refer to section 5.1.12.1.28);
- the Reclaim Unit Handle Usage log page (refer to section 5.1.12.1.29);
- the FDP Statistics log page (refer to section 5.1.12.1.30);
- the Flexible Data Placement Events feature (refer to section 5.1.25.1.21);
- the FDP Events log page (refer to section 5.1.12.1.31); and
- the Data Placement Directive (refer to section 8.1.8.4).

8.1.10.2 Enabling Flexible Data Placement (Informative)

The host prepares an Endurance Group for operation in Flexible Data Placement using the following process:

Exhibit A: Meta’s Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	1) Validate that the Flexible Data Placement capability is supported by issuing an Identify command to access the Identify Controller data structure and checking that the Flexible Data Placement Support (FDPS) bit is set to '1'. 2) Delete any existing namespaces that exist in the Endurance Group where Flexible Data Placement is to be enabled. 3) Issue a Get Log Page command specifying the FDP Configurations log page (refer to section 5.1.12.1.28). Parse the FDP Configurations List in the returned log page to determine the desired FDP configuration. 4) Enable Flexible Data Placement utilizing that desired FDP configuration by issuing a Set Features command specifying: a. the Flexible Data Placement feature; b. the Endurance Group Identifier field set to the ENDGID of the Endurance Group in which Flexible Data Placement is to be enabled; c. the FDPE bit set to '1' (i.e., enabling Flexible Data Placement); and d. the Flexible Data Placement Configuration Index field set to the index of the desired FDP configuration from the FDP Configurations List in the FDP Configurations log page. 5) Issue Get Features commands for the FDP Events feature (refer to section 5.1.25.1.21) to acquire the list of supported FDP events and the enabled state of each supported FDP event. 6) Issue Set Features commands for the FDP Events feature (refer to section 5.1.25.1.21) to specify if FDP events are required for Reclaim Unit Handles. 7) Issue an Identify command specifying the Identify Namespace data structure (i.e., CNS 00h as defined in Figure 273) to identify which LBA formats are supported. 8) For each namespace to be created in the Endurance Group, issue a Namespace Management command specifying: a. the Select field set to the Create operation (refer to the NVM Express Base Specification); b. the User Data Format; and c. the Placement Handle List used to define the Reclaim Unit Handle associated with each Placement Handle of the namespace (refer to the Namespace Management section of the applicable I/O Command Set specification). 9) For each namespace created in an Endurance Group in which the Data Placement Directive is to be utilized by the host by write commands specifying that namespace: a. Issue a Directive Send command specifying: i. the Directive Operation field set to Enable Directive (i.e., 01h); ii. the NSID of that namespace; iii. the DTYPE field in Command Dword 11 set to Identify (i.e., 00h); iv. the DTYPE field in Command Dword 12 set to Data Placement (i.e., 02h); and v. the Enable Directive bit set to '1' (i.e., to enable the Data Placement Directive). b. Issue an Identify command with CNS value 08h (i.e., the I/O Command Set Independent Identify Namespace data structure) to determine if a volatile write cache is present for the namespace. Following a Controller Level Reset, the host performs the following actions before resuming use of Flexible Data Placement in an Endurance Group: 1) Determine the FDP configuration: a. Issue a Get Feature command specifying the Flexible Data Placement feature (i.e., 1Dh) and the ENDGID for the Endurance Group to determine if Flexible Data Placement is enabled and the Flexible Data Placement Configuration Index. b. Issue a Get Log Page command specifying the FDP Configurations log page (i.e., Log Page Identifier 20h). 2) Issue Get Features commands for the FDP Events feature (refer to section 5.1.25.1.21) to acquire
--	---

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

- 3) Issue Set Features commands for the FDP Events feature (refer to section 5.1.25.1.21) to specify if supported FDP events are required for Reclaim Unit Handles.
- 4) For each namespace in the Endurance Group that the Data Placement Directive is to be utilized by the host by write commands specifying that namespace:
 - a. Issue an I/O Management Receive command to determine the Placement Handles for that namespace; and
 - b. Issue an Identify command with CNS value 08h (i.e., the I/O Command Set Independent Identify Namespace data structure) to determine if a volatile write cache is present for the namespace.

8.1.10.3 Write Commands using Flexible Data Placement (Informative)

Hosts writing user data to a namespace created in an Endurance Group that has Flexible Data Placement enabled have two mechanisms for specifying the placement of the user data with the Endurance Group to the controller.

1. Each namespace has a default Placement Handle. If any write command:
 - a. has the DTYPE field being ignored by the controller (refer to section 8.1.8.4); or
 - b. has the DTYPE field not being ignored by the controller and specifies a DTYPE field value cleared to 00h,

then the Placement Handle value of 0h for the specified namespace is combined with a Reclaim Group selected by the controller to create the Placement Identifier (refer to Figure 282 and Figure 283) utilized for that write command.

2. If any write command has the DTYPE field not being ignored by the controller and specifies a DTYPE field set to Data Placement (i.e., 02h) and specifies a namespace in which the Data Placement Directive is enabled, then the Placement Identifier specified by the DSPEC field specifies the Reclaim Group Identifier and Placement Handle used for the write command.

To reduce write amplification, hosts track the user data written to an entire Reclaim Unit and deallocate that user data by issuing one or more Dataset Management commands specifying the AD bit set to '1' and one or more Range fields specifying the LBA ranges that were written to that Reclaim Unit.

In addition, the NVMe specification includes the write command⁸:

3.3.6 Write command

The Write command writes data and metadata, if applicable, to the I/O controller for the logical blocks indicated. The host may also specify protection information to include as part of the operation.

The command uses Command Dword 2, Command Dword 3, Command Dword 10, Command Dword 11, Command Dword 12, Command Dword 13, Command Dword 14, and Command Dword 15 fields. If the command uses PRPs for the data transfer, then the Metadata Pointer, PRP Entry 1, and PRP Entry 2 fields are used. If the command uses SGLs for the data transfer, then the Metadata SGL Segment Pointer and SGL Entry 1 fields are used.

⁸ See, e.g., NVM Command Set Specification at § 3.3.6.

Exhibit A: Meta’s Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

Figure 72: Write - Command Dword 13 if CETYPE is non-zero

Bits	Description
31:16	Directive Specific (DSPEC): Specifies the Directive Specific value associated with the Directive Type field (refer to the Key Per I/O section in the NVM Express Base Specification).
15:00	Command Extension Value (CEV): The definition of this field is dependent on the value of the CETYPE field. Refer to the Key Per I/O section in the NVM Express Base Specification.

In addition, the NVMe base specification includes the “Directive Send command”⁹:

5.1.7 Directive Send command

The Directive Send command transfers a data buffer that is dependent on the Directive Type to the controller. Refer to section 8.1.8.

The Directive Send command uses the Data Pointer, Command Dword 10, and Command Dword 11 fields. Command Dword 12 and Command Dword 13 may be used based on the Directive Type field and the Directive Operation field. All other command specific fields are reserved.

Figure 178: Directive Send – Data Pointer

Bits	Description
127:00	Data Pointer (DPTR): This field specifies the start of the data buffer. Refer to Figure 92 for the definition of this field.

Figure 179: Directive Send – Command Dword 10

Bits	Description
31:00	Number of Dwords (NUMD): This field specifies the number of dwords to transfer. This is a 0’s based value.

Figure 180: Directive Send – Command Dword 11

Bits	Description
31:16	Directive Specific (DSPEC): The interpretation of this field is Directive Type dependent. Refer to section 8.1.8.
15:08	Directive Type (DTYPE): This field specifies the Directive Type. Refer to Figure 597 for the list of Directive Types.
07:00	Directive Operation (DOPER): This field specifies the Directive Operation to perform. The interpretation of this field is Directive Type dependent. Refer to section 8.1.8.

In addition, the NVMe base specification includes the “Data Placement Directive”¹⁰:

⁹ See, e.g., NVM Base Specification at § 5.1.7.

¹⁰ See, e.g., NVM Base Specification at § 8.1.8.4.

Exhibit A: Meta’s Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	8.1.8.4 Data Placement (Directive Type 02h, Optional) The Data Placement Directive enables the host to specify to the controller the Reclaim Unit (refer section 8.1.10) to place the user data in I/O commands specified by the appropriate I/O Command Set specification. The Data Placement Directive has no Directive Operations defined. Any Directive Receive command or Directive Send command that specifies a Data Placement Directive Type shall be aborted by the controller with a status code of Invalid Field in Command. If the specified namespace is not contained in an Endurance Group with Flexible Data Placement enabled, then the controller shall abort the Directive Send command with a status code of FDP Disabled.
--	---

Exhibit A: Meta’s Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	In addition, the NVMe base specification includes “Namespace Management”¹¹: 8.1.15 Namespace Management The Namespace Management capability consists of the Namespace Management command (refer to section 5.1.21) and the Namespace Attachment command (refer to section 5.1.20). The Namespace Management command is used to create a namespace or delete a namespace. The Namespace Attachment command is used to attach and detach controllers from a namespace. The Namespace Management capability is intended for use during manufacturing or by a system administrator. In addition to the Namespace Management capability, there are other events and capabilities that are able to affect the number of namespaces that exist within an NVM subsystem (e.g., the Capacity Management capability as described in section 8.1.4). If the Namespace Management capability is supported, then the controller: a) shall support the Namespace Management command and the Namespace Attachment command; b) shall set the NMS bit to ‘1’ in the OACS field (refer to Figure 312); c) shall support the Attached Namespace Attribute Changed asynchronous event (refer to Figure 151 and section 5.1.25.1.5); d) should support the Allocated Namespace Attribute Changed asynchronous event (refer to Figure 151 and section 5.1.25.1.5); and e) may support Namespace Granularity (refer to the NVM Command Set Specification). A controller may support the Namespace Attachment command without supporting the Namespace Management command. Such a controller: a) shall not set the Namespace Management Supported (NMS) bit to ‘1’ in the OACS field (refer to Figure 312); and b) should support the Namespace Attribute Change asynchronous event. If a namespace is detached from a controller, then the NSID that referred to that namespace becomes an inactive NSID (refer to section 3.2.1.4) on that controller. If a namespace is deleted from the NVM subsystem, then the NSID that referred to that namespace becomes an unallocated NSID (refer to section 3.2.1.3) in the NVM subsystem. Previously submitted but uncompleted or subsequently submitted commands to the affected NSID are handled by the controller as if they were issued to an inactive NSID (refer to Figure 92). The size of a namespace that contains formatted storage is based on the size requested in a create operation, the format of the namespace, and any characteristics (e.g., endurance). The controller determines the NVM capacity allocated for that namespace. Namespaces may be created with different usage characteristics (e.g., endurance) that utilize differing amounts of NVM capacity. Namespace characteristics and the mapping of these characteristics to NVM capacity usage are outside the scope of this specification. Reporting of capacity information for the NVM subsystem, Domain, Endurance Group, and NVM Set are described in section 3.8. For each namespace that contains formatted storage, the NVM Set and the Endurance Group that contain the namespace are reported in the Identify Namespace data structure. The NVM Set to be used for a namespace that contains formatted storage is based on the value in the NVM
--	--

¹¹ See, e.g., NVMe Base Specification at § 8.1.15.

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	Set Identifier field in a create operation. If the NVM Set Identifier field is cleared to 0h in a create operation for a namespace that contains formatted storage, then the controller shall choose the NVM Set from which to allocate capacity to create that namespace. If the NVM Set Identifier field and the Endurance Group Identifier field are both cleared to 0h in a create operation for a namespace that contains formatted storage, then the controller shall choose the Endurance Group and the NVM Set from which to allocate capacity to create that namespace. If the NVM Set Identifier field is cleared to 0h and the Endurance Group Identifier field is set to a non-zero value in a create operation for a namespace that contains formatted storage, then the controller shall choose the NVM Set in the specified Endurance Group from which to allocate capacity to create that namespace. If the NVM Set Identifier field is set to a non-zero value and the Endurance Group Identifier field is cleared to 0h in a create operation for a namespace that contains formatted storage, then the controller shall abort the command with a status code of Invalid Field in Command. If the NVM Set Identifier field and the Endurance Group Identifier field are both set to non-zero values in a create operation for a namespace that contains formatted storage and the specified NVM Set exists in the specified Endurance Group, then the controller shall allocate capacity for that created namespace from the specified NVM Set. If the NVM Set Identifier field and the Endurance Group Identifier field are both set to non-zero values in a create operation for a namespace that contains formatted storage and the specified NVM Set does not exist in the specified Endurance Group, then the controller shall abort the command with a status code of Invalid Field in Command. For each namespace that contains formatted storage, the NVM capacity used for that namespace is reported in the Identify Namespace data structure (refer to the applicable I/O Command Set specification). The controller may allocate NVM capacity in units such that the requested size for a namespace that contains formatted storage may be rounded up to the next unit boundary. The units in which NVM capacity is allocated are reported in the Namespace Granularity List (refer to the NVM Command Set Specification), if supported. For example, when using the NVM Command Set, if host software requests a namespace of 32 logical blocks with a logical block size of 4 KiB for a total size of 128 KiB and the allocation unit for the implementation is 1 MiB, then the NVM capacity consumed may be rounded up to 1 MiB. The NVM capacity fields may not correspond to the logical block size multiplied by the total number of logical blocks. The method of allocating ANA Group identifiers is outside the scope of this specification. If the ANA Group Identifier (refer to Figure 319 and the Identify Namespace data structure in the NVM Command Set Specification) is cleared to 0h, then the controller shall determine the ANAGRPID that is assigned to that namespace. As described above, the NVMe specification describes receiving, from a host, at least one storage device policy, wherein the at least one storage device policy indicates at least a recording method for writing content to the storage media. For example, § 8.1.10 describes the FDP functionality. Further, § 8.1.10.1 describes “[t]he host enables or disables Flexible Data Placement by issuing a Set Features command specifying the Flexible Data Placement feature (refer to section 5.1.25.1.20) and the FDP configuration (refer to section 5.1.12.1.28) to be applied to an Endurance Group.” Next § 8.1.10.1 describes “[t]he Namespace Management command (refer to section 8.1.15) is used to create a namespace within the Endurance Group” further explaining that “[t]he
--	---

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	Namespace Management command specifies a Placement Handle List” Further, § 8.1.10.3 describes the FDP write command “specifying the placement of the user data” using the Placement Handle, Reclaim group, and Placement Identifier. Additionally, § 3.3.6 describes the Write Command whereby, for example, the Placement Identifier is used as a parameter (e.g., DSPEC). Thus, the Accused Instrumentalities are “configured to” receive at least one storage policy (e.g., the Namespace Management command including a specified capacity (§ 8.1.15), enable the Data Placement Directive by sending the Directive Send command (§ 5.1.7, § 8.1.8.4), and a Placement Identifier embedded in the DSPEC field of each Write Command (§ 3.3.6)), wherein the at least one storage device policy indicates at least a recording method for writing content to the storage media as required by the claim. To the extent the FDP commands and associated data identified above is found not to literally satisfy the claimed “storage device policy,” the FDP commands and associated data described above satisfies the limitation under the doctrine of equivalents because it performs substantially the same function of indicating and/or selecting a recording method for writing content, in substantially the same way by providing host-supplied FDP information used by the Accused Instrumentalities for writing content in an indicated FDP region, to achieve substantially the same result of causing the content to be written according to the host provided recording method. Any differences between the claimed “storage device policy” and the FDP commands and associated data described in the NVMe Sections described above are insubstantial because the FDP commands and associated data provides the same operative instruction, selection, and/or indication used by the Accused Instrumentalities to determine the recording method for writing the content at the direction of the host. As shown below, Meta’s CacheLib benefits and/or takes advantage of the Accused Instrumentalities’ FDP functionality by sending at least one storage device policy, wherein the at least one storage device policy indicates at least a recording method for writing content to the storage media. To the extent it is argued that the asserted claims require the FDP functionality to be invoked and/or used, the description below also demonstrates infringement. For example, the file FDP enabled Cache.md¹² explains:
--	--

¹²

https://github.com/facebook/CacheLib/blob/v2026.06.01.00/website/docs/Cache_Library_User_Guides/FDP_enabled_Cache.md?plain=1

	### NVMe® Flexible Data Placement (NVMe® FDP) NVM Express®(NVMe®) released the ratified technical proposal TP4146a Flexible Data Placement (FDP) that defines a new method for placing host data (logical blocks) into the SSD in an effort to reduce SSD [Write Amplification Factor (WAF)](https://nvmexpress.org/nvmeflexible-data-placement-fdp-blog/). This provides the host a mechanism to control which of its data is placed into different sets of physical locations of the SSD (NAND blocks) called Reclaim Units. The host is able to write into multiple such Reclaim Units at a time allowing the host to isolate its data that has different lifetime. For more information on NVMe® FDP and various use cases refer this document: [Introduction to FDP](https://download.semiconductor.samsung.com/resources/white-paper/FDP_Whitepaper_102423_Final_10130020003525.pdf). ### How does CacheLib use FDP? CacheLib's BigHash and BlockCache produce distinct IO patterns on the SSD. BigHash generates a random write pattern while BlockCache generates a sequential write pattern. In a conventional SSD, these writes get mixed up in the physical NAND media. This intermixing can lead to a higher SSD Write Amplification Factor (WAF). To combat this in production environments CacheLib uses upto [50% of the SSD as host over-provisioning](https://www.usenix.org/system/files/osdi20-berg.pdf). The Flexible Data Placement (FDP) support within CacheLib aims to segregate the BigHash and BlockCache data of CacheLib within the SSDs. This reduces the device WAF even when configured with 0% host over-provisioning and improves device endurance. FDP support within Navy is optional. ### How to enable FDP in the SSD? <pre>```bash #Delete any pre-exisitng namespaces nvme delete-ns /dev/nvme0 -n 1</pre>
--	---

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	<pre>#Disable FDP nvme set-feature /dev/nvme0 -f 0x1D -c 0 -s nvme get-feature /dev/nvme0 -f 0x1D -H #Enable FDP nvme set-feature /dev/nvme0 -f 0x1D -c 1 -s #Verify whether FDP has been enabled/disabled nvme get-feature /dev/nvme0 -f 0x1D -H # Get capacity of drive and use it for further calculations nvme id-ctrl /dev/nvme0 grep nvmcap sed "s/,//g" awk '{print \$3/4096}' # Create namespace. use the capacity values from the above command in --nsze. For e.g. capacity is 459076086 nvme create-ns /dev/nvme0 -b 4096 --nsze=459076086 --ncap=459076086 -p 0,1,2,3 -n 4 #Attach namespace. e.g. NS id = 1, controller id = 0x7 nvme attach-ns /dev/nvme0 --namespace-id=1 --controllers=0x7 # Deallocate nvme dsm /dev/nvme0n1 -n 1 -b 459076086 ``` ### How to enable FDP in CacheLib? To enable Flexible Data Placement (FDP) support in `Device` layer of Navy, use the following configuration: <pre>```cpp navyConfig.setEnableFDP(enableFDP); ```</pre></pre>
--	---

Exhibit A: Meta’s Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	* When set to `true`, FDP is enabled and the BigHash and BlockCache device writes get segregated within the SSD. Apart from enabling FDP explicitly, the steps to setup and run a CacheLib instance remain unchanged. * To enable FDP in the CacheBench config file add the following line to the cache_config parameters: <pre>```cpp "navyEnableIoUring": true, "navyQDepth": 1, "deviceEnableFDP" : true ```</pre> As another example, the following exemplary code shows Meta’s CacheLib’s benefiting and/or taking advantage of FDP functionality and, to the extent necessary, that this limitation is met: 653 nvmConfig.navyConfig.setEnableFDP(config_.deviceEnableFDP); 13
--	--

¹³ Cache.h

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

```
45  ▾  ### How to enable FDP in the SSD?
46
47  ```bash
48  #Delete any pre-exisitng namespaces
49  nvme delete-ns /dev/nvme0 -n 1
50
51  #Disable FDP
52  nvme set-feature /dev/nvme0 -f 0x1D -c 0 -s nvme get-feature /dev/nvme0 -f 0x1D -H
53
54  #Enable FDP
55  nvme set-feature /dev/nvme0 -f 0x1D -c 1 -s
56
57  #Verify whether FDP has been enabled/disabled
58  nvme get-feature /dev/nvme0 -f 0x1D -H
59
60  # Get capacity of drive and use it for further calculations
61  nvme id-ctrl /dev/nvme0 | grep nvmcap | sed "s/,//g" | awk '{print $3/4096}'
62
63  # Create namespace. use the capacity values from the above command in --nsze. For e.g. capacity is 459076086
64  nvme create-ns /dev/nvme0 -b 4096 --nsze=459076086 --ncap=459076086 -p 0,1,2,3 -n 4
65
66  #Attach namespace. e.g. NS id = 1, controller id = 0x7
67  nvme attach-ns /dev/nvme0 --namespace-id=1 --controllers=0x7
68
69  # Deallocate
70  nvme dsm /dev/nvme0n1 -n 1 -b 459076086
71  ```
```

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

```
72
73  ✓ ### How to enable FDP in CacheLib?
74  To enable Flexible Data Placement (FDP) support in `Device` layer of Navy, use the following configuration:
75
76  ```cpp
77      navyConfig.setEnableFDP(enableFDP);
78  ```
79  * When set to `true`, FDP is enabled and the BigHash and BlockCache device writes get segregated within the SSD.
80
81  Apart from enabling FDP explicitly, the steps to setup and run a CacheLib instance remain unchanged.
82
83  * To enable FDP in the CacheBench config file add the following line to the cache_config parameters:
84
85  ```cpp
86      "navyEnableIoUring": true,
87      "navyQDepth": 1,
88      "deviceEnableFDP" : true
89  ```
14
299      // Device info vector for FDP support
300      const std::vector<std::shared_ptr<FdpNvme>> fdpNvmeVec_{};
```

¹⁴ FDP_enabled_Cache.md

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

```
957  ✓ std::unique_ptr<folly::AsyncBaseOp> AsyncIoContext::prepNvmeIo(IOPp& op) {
958      #ifndef CACHELIB_IOURING_DISABLE
959          std::unique_ptr<folly::IoUringOp> iouringCmdOp;
960          IOReq& req = op.parent_;
961
962          auto& options = static_cast<folly::IoUring*>(asyncBase_.get())->getOptions();
963          iouringCmdOp = std::make_unique<folly::IoUringOp>(<
964              folly::AsyncBaseOp::NotificationCallback(), options);
965
966          iouringCmdOp->initBase();
967          struct io_uring_sqe& sqe = iouringCmdOp->getSqe();
968          if (req.opType_ == OpType::READ) {
969              fdpNvmeVec_[kDefaultFdpIdx]->prepReadUringCmdSqe(sqe, op.data_, op.size_,
970                  op.offset_);
971          } else {
972              fdpNvmeVec_[kDefaultFdpIdx]->prepWriteUringCmdSqe(
973                  sqe, op.data_, op.size_, op.offset_, op.placeHandle_.value_or(-1));
974          }
975          io_uring_sqe_set_data(&sqe, iouringCmdOp.get());
976          return std::move(iouringCmdOp);
977      #else
978          return nullptr;
979      #endif
980  }
```

15

Additional exemplary evidence showing Meta's benefiting and/or taking advantage of FDP functionality, and to the extent necessary, that this limitation is found in:

- Cache.h¹⁶
- Device.cpp¹⁷
- Device.h¹⁸

¹⁵ Device.cpp

¹⁶ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/cachebench/cache/Cache.h>

¹⁷ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/Device.cpp>

¹⁸ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/Device.h>

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	- FdpNvme.cpp¹⁹ - FdpNvme.h²⁰ - NavyConfig.h²¹ Discovery into the specific storage devices, including but not limited to SSDs, used by Meta is ongoing, including the implementation of FDP and/or NVMe 2.1 by the SSD as identified above.				
1[B2] receive a storage request, from the host, to store content;	The Meta Accused Instrumentalities include a device controller operatively coupled with the storage media and configured to receive a storage request, from the host, to store content. For example, the NVMe command specification includes functionality for write commands indicating that the SSD is configured to receive a storage request from the host, to store content²²: 3.3.6 Write command The Write command writes data and metadata, if applicable, to the I/O controller for the logical blocks indicated. The host may also specify protection information to include as part of the operation. The command uses Command Dword 2, Command Dword 3, Command Dword 10, Command Dword 11, Command Dword 12, Command Dword 13, Command Dword 14, and Command Dword 15 fields. If the command uses PRPs for the data transfer, then the Metadata Pointer, PRP Entry 1, and PRP Entry 2 fields are used. If the command uses SGLs for the data transfer, then the Metadata SGL Segment Pointer and SGL Entry 1 fields are used. Figure 66: Write – Metadata Pointer <table><tr><th>Bits</th><th>Description</th></tr><tr><td>63:00</td><td>Metadata Pointer (MPTR): This field contains the Metadata Pointer, if applicable. Refer to the Common Command Format figure in the NVM Express Base Specification for the definition of this field.</td></tr></table>	Bits	Description	63:00	Metadata Pointer (MPTR): This field contains the Metadata Pointer, if applicable. Refer to the Common Command Format figure in the NVM Express Base Specification for the definition of this field.
Bits	Description				
63:00	Metadata Pointer (MPTR): This field contains the Metadata Pointer, if applicable. Refer to the Common Command Format figure in the NVM Express Base Specification for the definition of this field.				

¹⁹ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/FdpNvme.cpp>

²⁰ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/FdpNvme.h>

²¹ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/allocator/nvmcache/NavyConfig.h>

²² See, e.g., NVM Command Set Specification § 3.3.6

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

Figure 68: Write – Command Dword 2 and Dword 3

Bits	Description
63:48	Reserved
47:00	Logical Block Tags Upper (LBTU): This field and Command Dword 14 specify the variable sized Logical Block Storage Tag (LBST) and Initial Logical Block Reference Tag (ILBRT) fields, which are defined in section 5.3.1.4.1. If the namespace is not formatted to use end-to-end protection information, then this field is ignored by the controller.

Figure 69: Write – Command Dword 10 and Command Dword 11

Bits	Description
63:00	Starting LBA (SLBA): This field indicates the 64-bit address of the first logical block to be written as part of the operation. Command Dword 10 contains bits 31:00; Command Dword 11 contains bits 63:32.

Figure 70: Write – Command Dword 12

Bits	Description
31	Limited Retry (LR): If this bit is set to '1', then the controller should apply limited retry efforts. If this bit is cleared to '0', then the controller should apply all available error recovery means to write the data to the NVM.
30	Force Unit Access (FUA): If this bit is set to '1', then for data and metadata, if any, associated with logical blocks specified by the Write command, the controller shall write that data and metadata, if any, to non-volatile medium before indicating command completion. There is no implied ordering with other commands. If this bit is cleared to '0', then this bit has no effect.
29:26	Protection Information (PRINFO): Specifies the protection information action and check field, as defined in Figure 11.
25	Reserved
24	Storage Tag Check (STC): This bit specifies the Storage Tag field shall be checked as part of end-to-end data protection processing as defined in Figure 12.
23:20	Directive Type (DTYPE): Specifies the Directive Type associated with the Directive Specific field (refer to the Directives section in the NVM Express Base Specification).
19:16	Command Extension Type (CETYPE): Specifies the Command Extension Type that applies to the command (refer to the Key Per I/O section in the NVM Express Base Specification).
15:00	Number of Logical Blocks (NLB): This field indicates the number of logical blocks to be written. This is a 0's based value.

The definition of Command Dword 13 is based on the CETYPE value. If the CETYPE value is cleared to 0h, then Command Dword 13 is defined in Figure 71. If the CETYPE value is non-zero, then Command Dword 13 is defined in Figure 72.

Figure 71: Write – Command Dword 13 if CETYPE is cleared to 0h

Bits	Description
31:16	Directive Specific (DSPEC): Specifies the Directive Specific value associated with the Directive Type field (refer to the Directives section in the NVM Express Base Specification).

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

Figure 71: Write – Command Dword 13 if CETYPE is cleared to 0h

Bits	Description																	
15:08	Reserved																	
07:00	Dataset Management (DSM): This field indicates attributes for the LBA(s) being written.																	
	<table><tr><th>Bits</th><th>Definition</th></tr><tr><td>07</td><td>Incompressible (INCPRS): If this bit is set to '1', then data is not compressible for the logical blocks indicated. If this bit is cleared to '0', then no information on compression is provided.</td></tr><tr><td>06</td><td>Sequential Request (SEQREQ): If this bit is set to '1', then this command is part of a sequential write that includes multiple Write commands. If this bit is cleared to '0', then no information on sequential access is provided.</td></tr></table>	Bits	Definition	07	Incompressible (INCPRS): If this bit is set to '1', then data is not compressible for the logical blocks indicated. If this bit is cleared to '0', then no information on compression is provided.	06	Sequential Request (SEQREQ): If this bit is set to '1', then this command is part of a sequential write that includes multiple Write commands. If this bit is cleared to '0', then no information on sequential access is provided.											
	Bits	Definition																
	07	Incompressible (INCPRS): If this bit is set to '1', then data is not compressible for the logical blocks indicated. If this bit is cleared to '0', then no information on compression is provided.																
	06	Sequential Request (SEQREQ): If this bit is set to '1', then this command is part of a sequential write that includes multiple Write commands. If this bit is cleared to '0', then no information on sequential access is provided.																
	Access Latency (AL): This field specifies the expected access latency.																	
	<table><tr><th>Value</th><th>Definition</th></tr><tr><td>00b</td><td>None. No latency information provided.</td></tr><tr><td>01b</td><td>Idle. Longer latency acceptable.</td></tr><tr><td>10b</td><td>Normal. Typical latency.</td></tr><tr><td>11b</td><td>Low. Smallest possible latency.</td></tr></table>	Value	Definition	00b	None. No latency information provided.	01b	Idle. Longer latency acceptable.	10b	Normal. Typical latency.	11b	Low. Smallest possible latency.							
	Value	Definition																
	00b	None. No latency information provided.																
	01b	Idle. Longer latency acceptable.																
10b	Normal. Typical latency.																	
11b	Low. Smallest possible latency.																	
Access Frequency (AF): This field specifies the expected access frequency.																		
<table><tr><th>Value</th><th>Definition</th></tr><tr><td>0h</td><td>No frequency information provided.</td></tr><tr><td>1h</td><td>Typical number of reads and writes expected for this LBA range.</td></tr><tr><td>2h</td><td>Infrequent writes and infrequent reads to the LBA range indicated.</td></tr><tr><td>3h</td><td>Infrequent writes and frequent reads to the LBA range indicated.</td></tr><tr><td>4h</td><td>Frequent writes and infrequent reads to the LBA range indicated.</td></tr><tr><td>5h</td><td>Frequent writes and frequent reads to the LBA range indicated.</td></tr><tr><td>6h</td><td>One time write. E.g., command is due to virus scan, backup, file copy, or archive.</td></tr><tr><td>7h to Fh</td><td>Reserved</td></tr></table>	Value	Definition	0h	No frequency information provided.	1h	Typical number of reads and writes expected for this LBA range.	2h	Infrequent writes and infrequent reads to the LBA range indicated.	3h	Infrequent writes and frequent reads to the LBA range indicated.	4h	Frequent writes and infrequent reads to the LBA range indicated.	5h	Frequent writes and frequent reads to the LBA range indicated.	6h	One time write. E.g., command is due to virus scan, backup, file copy, or archive.	7h to Fh	Reserved
Value	Definition																	
0h	No frequency information provided.																	
1h	Typical number of reads and writes expected for this LBA range.																	
2h	Infrequent writes and infrequent reads to the LBA range indicated.																	
3h	Infrequent writes and frequent reads to the LBA range indicated.																	
4h	Frequent writes and infrequent reads to the LBA range indicated.																	
5h	Frequent writes and frequent reads to the LBA range indicated.																	
6h	One time write. E.g., command is due to virus scan, backup, file copy, or archive.																	
7h to Fh	Reserved																	

Figure 72: Write - Command Dword 13 if CETYPE is non-zero

Bits	Description
31:16	Directive Specific (DSPEC): Specifies the Directive Specific value associated with the Directive Type field (refer to the Key Per I/O section in the NVM Express Base Specification).
15:00	Command Extension Value (CEV): The definition of this field is dependent on the value of the CETYPE field. Refer to the Key Per I/O section in the NVM Express Base Specification.

Figure 73: Write – Command Dword 14

Bits	Description
31:00	Logical Block Tags Lower (LBTL): This field and bits 47:00 of Command Dword 2 and Dword 3 specify the variable sized Logical Block Storage Tag (LBST) and Initial Logical Block Reference Tag (ILBRT) fields, which are defined in section 5.3.1.4.1. If the namespace is not formatted to use end-to-end protection information, then this field is ignored by the controller.

Figure 74: Write – Command Dword 15

Bits	Description
31:16	Logical Block Application Tag Mask (LBATM): This field specifies the Application Tag Mask value. If the namespace is not formatted to use end-to-end protection information, then this field is ignored by the controller. Refer to section 5.3.
15:00	Logical Block Application Tag (LBAT): This field specifies the Application Tag value. If the namespace is not formatted to use end-to-end protection information, then this field is ignored by the controller. Refer to section 5.3.

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	As another example, the NVMe 2.1 specification includes protocols for FDP that includes receiving a storage request from the host, to store content²³: 8.1.10.3 Write Commands using Flexible Data Placement (Informative) Hosts writing user data to a namespace created in an Endurance Group that has Flexible Data Placement enabled have two mechanisms for specifying the placement of the user data with the Endurance Group to the controller. 1. Each namespace has a default Placement Handle. If any write command: a. has the DTYPE field being ignored by the controller (refer to section 8.1.8.4); or b. has the DTYPE field not being ignored by the controller and specifies a DTYPE field value cleared to 00h, then the Placement Handle value of 0h for the specified namespace is combined with a Reclaim Group selected by the controller to create the Placement Identifier (refer to Figure 282 and Figure 283) utilized for that write command. 2. If any write command has the DTYPE field not being ignored by the controller and specifies a DTYPE field set to Data Placement (i.e., 02h) and specifies a namespace in which the Data Placement Directive is enabled, then the Placement Identifier specified by the DSPEC field specifies the Reclaim Group Identifier and Placement Handle used for the write command. To reduce write amplification, hosts track the user data written to an entire Reclaim Unit and deallocate that user data by issuing one or more Dataset Management commands specifying the AD bit set to '1' and one or more Range fields specifying the LBA ranges that were written to that Reclaim Unit. As shown above, the Accused Instrumentalities receive from the host a storage request to store content through NVMe Write commands, for example, including the Write commands described in NVMe Section 8.1.10.3, "Write Commands using Flexible Data Placement," and/or the Write command described in NVMe Section 3.3.6, which request that the Accused Instrumentalities write and store content, for example, as indicated in the DPTR field at the Logical Block Address indicated in the SLBA field. As shown below, Meta's CacheLib benefits and/or takes advantage of the Accused Instrumentalities' FDP functionality by sending a storage request to store content. To the extent it is argued that the asserted claims require the FDP functionality to be invoked and/or used, the exemplary code below also demonstrates infringement.
--	--

²³ See, e.g., NVMe 2.1 Standard at § 8.1.10.3.

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

For example, the following exemplary code confirms Meta CacheLib's benefiting and/or taking advantage of FDP, and to the extent necessary, that this limitation is met:

```
299 // Device info vector for FDP support
300 const std::vector<std::shared_ptr<FdpNvme>> fdpNvmeVec_{};

957  std::unique_ptr<folly::AsyncBaseOp> AsyncIoContext::prepNvmeIo(IOOp& op) {
958  #ifndef CACHELIB_IOURING_DISABLE
959      std::unique_ptr<folly::IoUringOp> iouringCmdOp;
960      IOReq& req = op.parent_;
961
962      auto& options = static_cast<folly::IoUring*>(asyncBase_.get())->getOptions();
963      iouringCmdOp = std::make_unique<folly::IoUringOp>(
964          folly::AsyncBaseOp::NotificationCallback(), options);
965
966      iouringCmdOp->initBase();
967      struct io_uring_sqe& sqe = iouringCmdOp->getSqe();
968      if (req.opType_ == OpType::READ) {
969          fdpNvmeVec_[kDefaultFdpIdx]->prepReadUringCmdSqe(sqe, op.data_, op.size_,
970                                                         op.offset_);
971      } else {
972          fdpNvmeVec_[kDefaultFdpIdx]->prepWriteUringCmdSqe(
973              sqe, op.data_, op.size_, op.offset_, op.placeHandle_.value_or(-1));
974      }
975      io_uring_sqe_set_data(&sqe, iouringCmdOp.get());
976      return std::move(iouringCmdOp);
977  #else
978      return nullptr;
979  #endif
980  }
```

24

²⁴ Device.cpp

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

```
136  void FdpNvme::prepFdpUringCmdSqe(struct io_uring_sqe& sqe,
137                                  void* buf,
138                                  size_t size,
139                                  off_t start,
140                                  uint8_t opcode,
141                                  uint8_t dtype,
142                                  uint16_t dspec) {
143      uint32_t maxTfrSize = nvmeData_.getMaxTfrSize();
144      if ((maxTfrSize != 0) && (size > maxTfrSize)) {
145          throw std::invalid_argument("Exceeds max Transfer size");
146      }
147      // Clear the SQE entry to avoid some arbitrary flags being set.
148      memset(&sqe, 0, sizeof(struct io_uring_sqe));
149
150      sqe.fd = file_.fd();
151      sqe.opcode = IORING_OP_URING_CMD;
152      sqe.cmd_op = NVME_URING_CMD_IO;
153
154      struct nvme_uring_cmd* cmd = (struct nvme_uring_cmd*)&sqe.cmd;
155      if (cmd == nullptr) {
156          throw std::invalid_argument("Uring cmd is NULL!");
157      }
158      memset(cmd, 0, sizeof(struct nvme_uring_cmd));
159      cmd->opcode = opcode;
160
161      // start LBA of the IO = Req_start (offset in partition) + Partition_start
162      uint64_t sLba = (start >> nvmeData_.lbaShift()) + nvmeData_.partStartLba();
```

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

```
162     uint64_t sLba = (start >> nvmeData_.lbaShift()) + nvmeData_.partStartLba();
163     uint32_t nLb = (size >> nvmeData_.lbaShift()) - 1; // nLb is 0 based
164
165     /* cdw10 and cdw11 represent starting lba */
166     cmd->cdw10 = sLba & 0xffffffff;
167     cmd->cdw11 = sLba >> 32;
168     /* cdw12 represent number of lba's for read/write */
169     cmd->cdw12 = (dtype & 0xFF) << 20 | nLb;
170     cmd->cdw13 = (dspec << 16);
171     cmd->addr = (uint64_t)buf;
172     cmd->data_len = size;
173
174     cmd->nsid = nvmeData_.nsId();
175 }
```

```
185  void FdpNvme::prepWriteUringCmdSqe(
186     struct io_uring_sqe& sqe, void* buf, size_t size, off_t start, int handle)
187     static constexpr uint8_t kPlacementMode = 2;
188     uint16_t pid;
189
190     if (handle == -1) {
191         pid = getFdpPID(kDefaultPIDIdx); // Use the default stream
192     } else if (handle >= 0 && handle <= maxPIDIdx_) {
193         pid = getFdpPID(static_cast<uint16_t>(handle));
194     } else {
195         throw std::invalid_argument("Invalid placement identifier");
196     }
197
198     prepFdpUringCmdSqe(sqe, buf, size, start, nvme_cmd_write, kPlacementMode,
199                       pid);
200 }
```

25

²⁵ Device.cpp

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	Additional exemplary evidence showing Meta's CacheLib benefitting and/or taking advantage of FDP, and to the extent necessary, that this limitation is met is found in: - Device.cpp²⁶ - Device.h²⁷ - FdpNvme.cpp²⁸ - RegionManager.cpp²⁹ - BigHash.h³⁰ - BigHash.cpp³¹ - BlockCache.h³² - BlockCache.cpp³³ To the extent the above is found not to literally satisfy the claim, it satisfies the limitation under the doctrine of equivalents because the commands perform substantially the same function of receiving a storage request to store content from the host, in substantially the same way by providing a host issued write command to the Accused Instrumentalities, to achieve substantially the same result of causing the Accused Instrumentalities to write and store the content. Any differences are insubstantial because the commands provide the same functionality of receiving a request for storing content from the host. Discovery into the specific storage devices, including but not limited to SSDs, used by Meta is ongoing, including the implementation of FDP and/or NVMe 2.1 and/or NVM Write Command by the SSD as identified above.
1[B3] store the content on the storage media by	The Meta Accused Instrumentalities include a device controller operatively coupled with the storage media and configured to store the content on the storage media by utilizing the recording method indicated by the at least one storage device policy.

²⁶ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/Device.cpp>

²⁷ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/Device.h>

²⁸ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/FdpNvme.cpp>

²⁹ https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/RegionManager.cpp

³⁰ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/bighash/BigHash.h>

³¹ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/bighash/BigHash.cpp>

³² https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/BlockCache.h

³³ https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/BlockCache.cpp

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

utilizing the recording method indicated by the at least one storage device policy;	As an example, the NVMe 2.1 specification includes protocols for FDP that includes storing the content on the storage media by utilizing the recording method indicated by the at least one storage device policy³⁴: 8.1.10.3 Write Commands using Flexible Data Placement (Informative) Hosts writing user data to a namespace created in an Endurance Group that has Flexible Data Placement enabled have two mechanisms for specifying the placement of the user data with the Endurance Group to the controller. 1. Each namespace has a default Placement Handle. If any write command: a. has the DTYPE field being ignored by the controller (refer to section 8.1.8.4); or b. has the DTYPE field not being ignored by the controller and specifies a DTYPE field value cleared to 00h, then the Placement Handle value of 0h for the specified namespace is combined with a Reclaim Group selected by the controller to create the Placement Identifier (refer to Figure 282 and Figure 283) utilized for that write command. 2. If any write command has the DTYPE field not being ignored by the controller and specifies a DTYPE field set to Data Placement (i.e., 02h) and specifies a namespace in which the Data Placement Directive is enabled, then the Placement Identifier specified by the DSPEC field specifies the Reclaim Group Identifier and Placement Handle used for the write command. To reduce write amplification, hosts track the user data written to an entire Reclaim Unit and deallocate that user data by issuing one or more Dataset Management commands specifying the AD bit set to '1' and one or more Range fields specifying the LBA ranges that were written to that Reclaim Unit. 3.3.6 Write command The Write command writes data and metadata, if applicable, to the I/O controller for the logical blocks indicated. The host may also specify protection information to include as part of the operation. The command uses Command Dword 2, Command Dword 3, Command Dword 10, Command Dword 11, Command Dword 12, Command Dword 13, Command Dword 14, and Command Dword 15 fields. If the command uses PRPs for the data transfer, then the Metadata Pointer, PRP Entry 1, and PRP Entry 2 fields are used. If the command uses SGLs for the data transfer, then the Metadata SGL Segment Pointer and SGL Entry 1 fields are used.
---	--

³⁴ See, e.g., NVMe 2.1 Base Specification at §§ 8.1.10.1, 8.1.10.3; NVMe Command Set Specification at § 3.3.6.

Figure 71: Write – Command Dword 13 if CETYPE is cleared to 0h

Bits	Description
31:16	Directive Specific (DSPEC): Specifies the Directive Specific value associated with the Directive Type field (refer to the Directives section in the NVM Express Base Specification).
15:08	Reserved

As stated above for limitation 1[B1], Meta’s CacheLib benefits and/or takes advantage of the Accused Instrumentalities’ FDP functionality by sending at least one storage device policy. Further, Meta’s CacheLib benefits and/or takes advantage of the Accused Instrumentalities’ FDP functionality by, for example, providing the FDP placement information. To the extent it is argued that the asserted claims require the FDP functionality to be invoked and/or used, the description below also demonstrates infringement.

The following exemplary code shows Meta’s CacheLib benefitting and/or taking advantage of FDP functionality and, to the extent necessary, that this limitation is met:

```
299      // Device info vector for FDP support
300      const std::vector<std::shared_ptr<FdpNvme>> fdpNvmeVec_{};
```

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

```
957  ✓ std::unique_ptr<folly::AsyncBaseOp> AsyncIoContext::prepNvmeIo(IOOp& op) {
958      #ifndef CACHELIB_IOURING_DISABLE
959          std::unique_ptr<folly::IoUringOp> iouringCmdOp;
960          IOReq& req = op.parent_;
961
962          auto& options = static_cast<folly::IoUring*>(asyncBase_.get())->getOptions();
963          iouringCmdOp = std::make_unique<folly::IoUringOp>(
964              folly::AsyncBaseOp::NotificationCallback(), options);
965
966          iouringCmdOp->initBase();
967          struct io_uring_sqe& sqe = iouringCmdOp->getSqe();
968          if (req.opType_ == OpType::READ) {
969              fdpNvmeVec_[kDefaultFdpIdx]->prepReadUringCmdSqe(sqe, op.data_, op.size_,
970                                                                  op.offset_);
971          } else {
972              fdpNvmeVec_[kDefaultFdpIdx]->prepWriteUringCmdSqe(
973                  sqe, op.data_, op.size_, op.offset_, op.placeHandle_.value_or(-1));
974          }
975          io_uring_sqe_set_data(&sqe, iouringCmdOp.get());
976          return std::move(iouringCmdOp);
977      #else
978          return nullptr;
979      #endif
980  }
```

35

³⁵ Device.cpp

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

```
136  void FdpNvme::prepFdpUringCmdSqe(struct io_uring_sqe& sqe,
137                                  void* buf,
138                                  size_t size,
139                                  off_t start,
140                                  uint8_t opcode,
141                                  uint8_t dtype,
142                                  uint16_t dspec) {
143      uint32_t maxTfrSize = nvmeData_.getMaxTfrSize();
144      if ((maxTfrSize != 0) && (size > maxTfrSize)) {
145          throw std::invalid_argument("Exceeds max Transfer size");
146      }
147      // Clear the SQE entry to avoid some arbitrary flags being set.
148      memset(&sqe, 0, sizeof(struct io_uring_sqe));
149
150      sqe.fd = file_.fd();
151      sqe.opcode = IORING_OP_URING_CMD;
152      sqe.cmd_op = NVME_URING_CMD_IO;
153
154      struct nvme_uring_cmd* cmd = (struct nvme_uring_cmd*)&sqe.cmd;
155      if (cmd == nullptr) {
156          throw std::invalid_argument("Uring cmd is NULL!");
157      }
158      memset(cmd, 0, sizeof(struct nvme_uring_cmd));
159      cmd->opcode = opcode;
160
161      // start LBA of the IO = Req_start (offset in partition) + Partition_start
162      uint64_t sLba = (start >> nvmeData_.lbaShift()) + nvmeData_.partStartLba();
```

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

```
162     uint64_t sLba = (start >> nvmeData_.lbaShift()) + nvmeData_.partStartLba();
163     uint32_t nLb = (size >> nvmeData_.lbaShift()) - 1; // nLb is 0 based
164
165     /* cdw10 and cdw11 represent starting lba */
166     cmd->cdw10 = sLba & 0xffffffff;
167     cmd->cdw11 = sLba >> 32;
168     /* cdw12 represent number of lba's for read/write */
169     cmd->cdw12 = (dtype & 0xFF) << 20 | nLb;
170     cmd->cdw13 = (dspec << 16);
171     cmd->addr = (uint64_t)buf;
172     cmd->data_len = size;
173
174     cmd->nsid = nvmeData_.nsId();
175 }
```

```
185  void FdpNvme::prepWriteUringCmdSqe(
186     struct io_uring_sqe& sqe, void* buf, size_t size, off_t start, int handle)
187     static constexpr uint8_t kPlacementMode = 2;
188     uint16_t pid;
189
190     if (handle == -1) {
191         pid = getFdpPID(kDefaultPIDIdx); // Use the default stream
192     } else if (handle >= 0 && handle <= maxPIDIdx_) {
193         pid = getFdpPID(static_cast<uint16_t>(handle));
194     } else {
195         throw std::invalid_argument("Invalid placement identifier");
196     }
197
198     prepFdpUringCmdSqe(sqe, buf, size, start, nvme_cmd_write, kPlacementMode,
199                       pid);
200 }
```

36

³⁶ FdpNvme.cpp

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	Additional exemplary evidence showing Meta CacheLib benefitting and/or taking advantage of FDP and, to the extent necessary, that this limitation is met is found in: - Device.cpp³⁷ - Device.h³⁸ - FdpNvme.cpp³⁹ - RegionManager.cpp⁴⁰ - BigHash.h⁴¹ - BigHash.cpp⁴² - BlockCache.h⁴³ - BlockCache.cpp⁴⁴ As shown above, the Accused Instrumentalities store content by utilizing the recording method indicated by the at least one storage device policy, for example, including the command described in NVMe Section §§ 8.1.10.3 and 3.3.6, which requests that the Accused Instrumentalities write and store content, for example, indicated in the DPTR field at the LBA indicated in the SLBA field using the FDP storage policy indicated by the DSPEC field. Discovery into the specific storage devices, including but not limited to SSDs, used by Meta is ongoing, including the implementation of FDP and/or NVMe 2.1 and/or NVM Write Command by the SSD as identified above.
1[B4] record storage information for the content on a	The Meta Accused Instrumentalities include a device controller operatively coupled with the storage media and configured to record storage information for the content on a memory of the device controller, comprising at least a content identifier for the content.

³⁷ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/Device.cpp>

³⁸ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/Device.h>

³⁹ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/FdpNvme.cpp>

⁴⁰ https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/RegionManager.cpp

⁴¹ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/bighash/BigHash.h>

⁴² <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/bighash/BigHash.cpp>

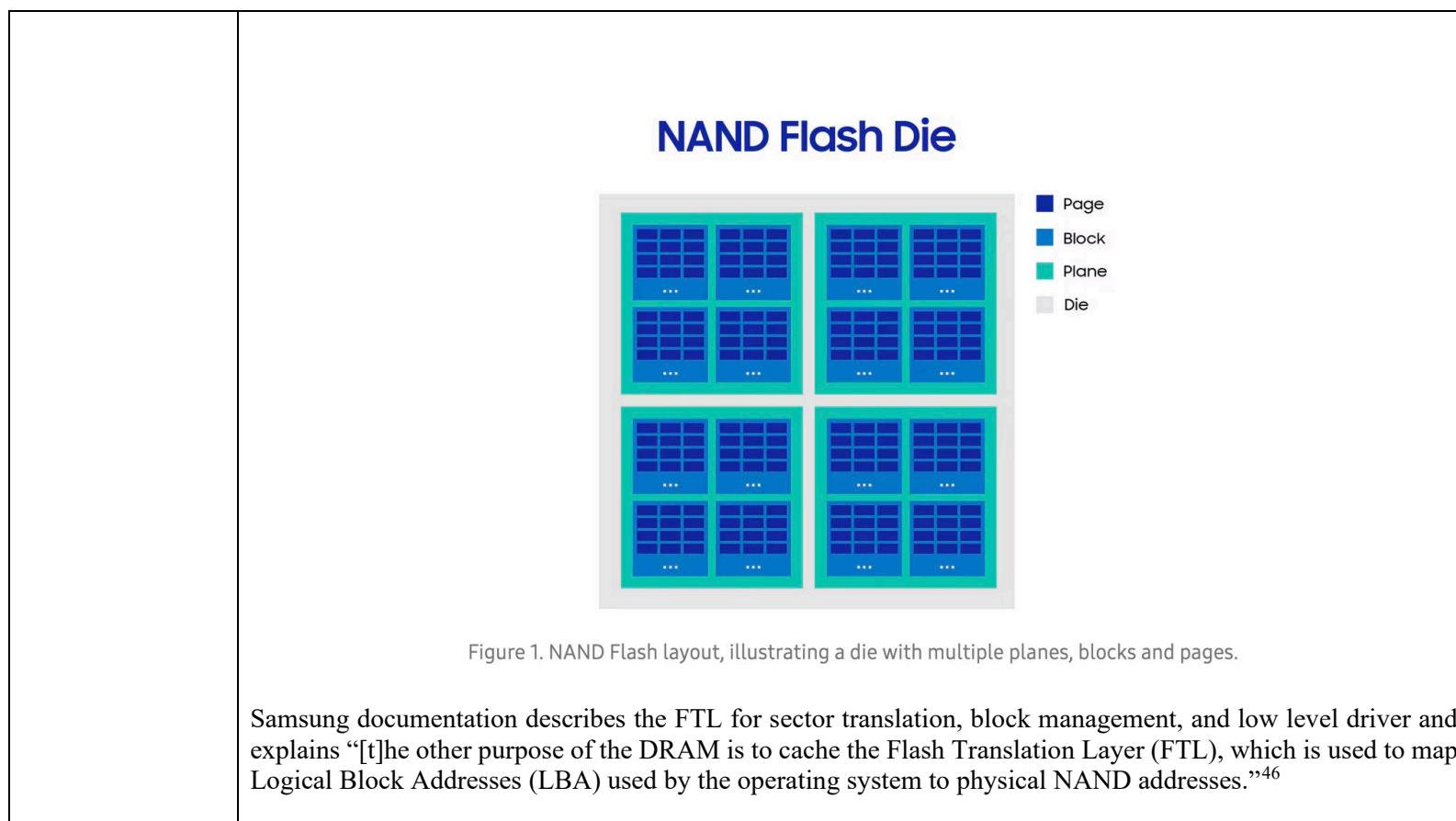
⁴³ https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/BlockCache.h

⁴⁴ https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/BlockCache.cpp

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

memory of the device controller, comprising at least a content identifier for the content;	For example, Samsung public documentation indicates that the Samsung SSDs include a Flash Translation Layer (FTL) that is used to record storage information for the content on a memory of the device controller comprising at least a content identifier (for example, the Logical Block Address, or the identifier indicated in the SLBA field) for the content⁴⁵: writing data to the same physical address internally. The Flash Translation Layer (FTL) in SSDs take care of this management using multiple steps: • Write the new data to an available free page in the open SSD block • Invalidate the old page, which had the previously written data • Update the FTL metadata information to point to the new page. The metadata here refers to a mapping that translates the logical addresses used by the host to physical addresses in the NAND media (L2P Mapping).
--	---

⁴⁵ <https://semiconductor.samsung.com/news-events/tech-blog/a-brief-history-of-data-placement-technologies/>



⁴⁶ https://image-us.samsung.com/SamsungUS/b2b/resource/2016/07/08/b2b_resource_WHP-SSD-POWERLOSSPROTECTION-r1-JUL16J.pdf

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

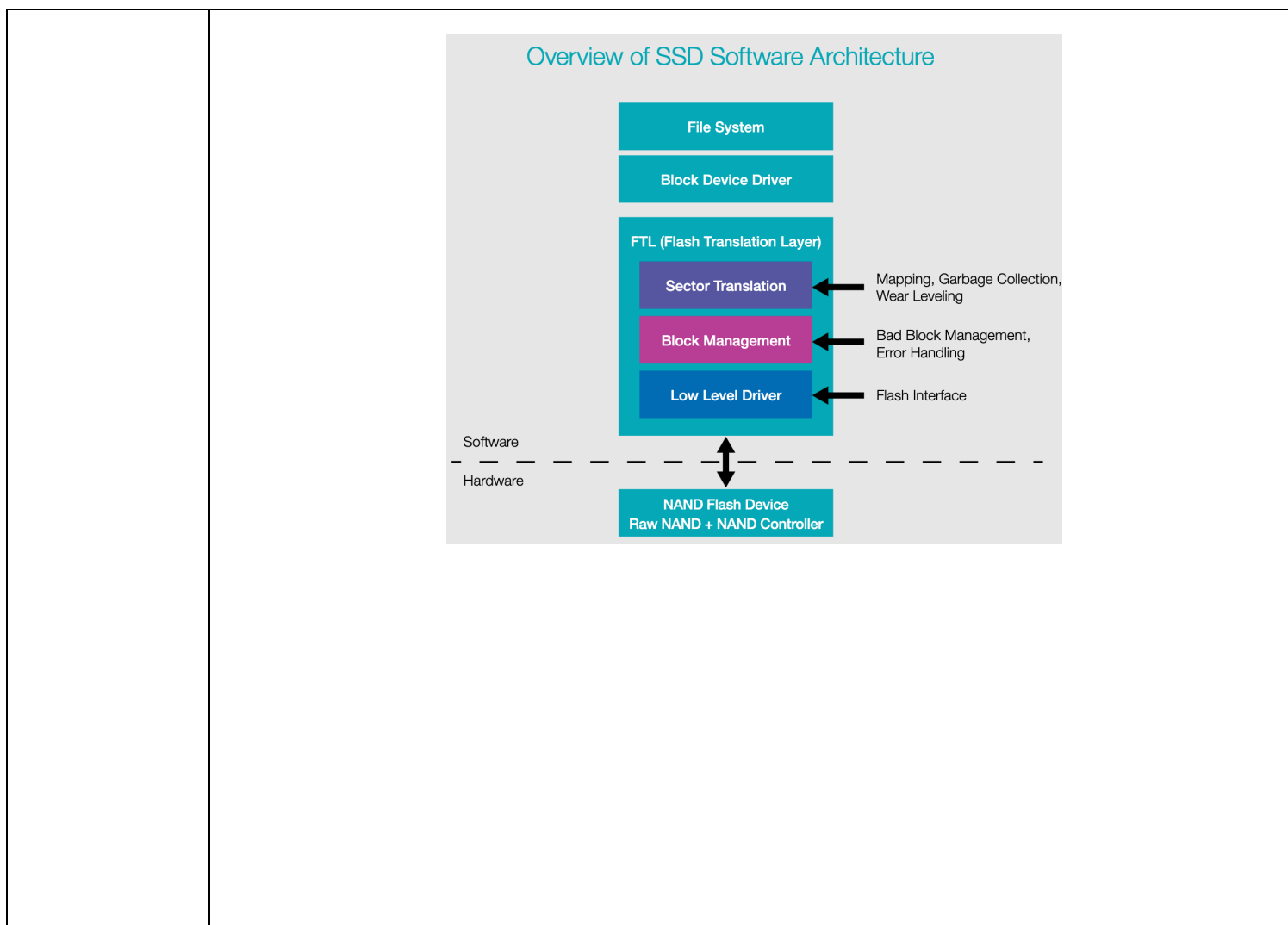


Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	FLASH TRANSLATION LAYER: KEEPING DATA ORGANIZED The other purpose of the DRAM is to cache the Flash Translation Layer (FTL), which is used to map Logical Block Addresses (LBA) used by the operating system to physical NAND addresses. The FTL is constantly changing as new write IOs from the host are written to the drive and the drive performs background tasks such as garbage collection and wear leveling, thus using DRAM for FTL caching is necessary because DRAM has substantially lower latency than NAND, its endurance is practically unlimited and it's bit-addressible. Without the DRAM cache, SSDs would not perform as well as they do, and there would also be endurance concerns as FTL changes consist of very small writes and NAND can only be programmed at the page-level (typically 8KB or 16KB in modern NAND). Because of the volatile nature of DRAM (i.e. all data is lost in an event of power loss), SSDs need to incorporate power loss protection (PLP) techniques to prevent against data loss and corruption. During a normal shut down or sleep, an SSD is given enough time by the operating system to write all contents of DRAM to NAND, but in case of an unexpected power loss, the data in the DRAM may be lost. Since DRAM hosts the most current version of the FTL, an unexpected power loss may corrupt all data in the SSD if proper power loss protections are not in place. Unexpected power losses may be due to power outages, but also due to power supply glitches that don't impact the rest of the system. Hence an uninterruptible power supply (UPS) or any other form of backup power is not enough to prevent against unexpected power losses, and SSDs need internal mechanisms to guarantee data integrity. Discovery into the specific SSDs used by Meta as well as the specific mechanism invoked by Meta Accused Instrumentalities is ongoing including but not limited to, (1) the FTL mechanism used by the SSDs and (2) memory associated with storing the FTL. As described above, for example, the FTL includes a data structure, for example a table, that stores mapping information and would reside in a memory (e.g., DRAM, NVRAM, registers/controllers, etc.) accessible by the device controller. As another example, the NVMe specification⁴⁷ describes utilizing the LBA field as part of the write command and thus, the SSD would support the LBA conversion to the physical address in an associated data structure stored in memory:
--	---

⁴⁷ See, e.g., NVMe Command Set Specification Revision 1.2 at Fig. 69

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

Figure 69: Write – Command Dword 10 and Command Dword 11

Bits	Description
63:00	Starting LBA (SLBA): This field indicates the 64-bit address of the first logical block to be written as part of the operation. Command Dword 10 contains bits 31:00; Command Dword 11 contains bits 63:32.

As shown below, Meta's CacheLib benefits and/or takes advantage of the Accused Instrumentalities' FDP functionality by providing, for example, the Logical Block Address (LBA). To the extent it is argued that the asserted claims require the FDP functionality to be invoked and/or used, the description below also demonstrates infringement.

For example, the following exemplary code confirms that the limitation is met:

```
299 // Device info vector for FDP support
300 const std::vector<std::shared_ptr<FdpNvme>> fdpNvmeVec_{};
```

48

⁴⁸ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/Device.cpp>

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

```
957  ✓ std::unique_ptr<folly::AsyncBaseOp> AsyncIoContext::prepNvmeIo(IOOp& op) {
958      #ifndef CACHELIB_IOURING_DISABLE
959          std::unique_ptr<folly::IoUringOp> iouringCmdOp;
960          IOReq& req = op.parent_;
961
962          auto& options = static_cast<folly::IoUring*>(asyncBase_.get())->getOptions();
963          iouringCmdOp = std::make_unique<folly::IoUringOp>(
964              folly::AsyncBaseOp::NotificationCallback(), options);
965
966          iouringCmdOp->initBase();
967          struct io_uring_sqe& sqe = iouringCmdOp->getSqe();
968          if (req.opType_ == OpType::READ) {
969              fdpNvmeVec_[kDefaultFdpIdx]->prepReadUringCmdSqe(sqe, op.data_, op.size_,
970                                                                  op.offset_);
971          } else {
972              fdpNvmeVec_[kDefaultFdpIdx]->prepWriteUringCmdSqe(
973                  sqe, op.data_, op.size_, op.offset_, op.placeHandle_.value_or(-1));
974          }
975          io_uring_sqe_set_data(&sqe, iouringCmdOp.get());
976          return std::move(iouringCmdOp);
977      #else
978          return nullptr;
979      #endif
980  }
```

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

```
136  void FdpNvme::prepFdpUringCmdSqe(struct io_uring_sqe& sqe,
137                                  void* buf,
138                                  size_t size,
139                                  off_t start,
140                                  uint8_t opcode,
141                                  uint8_t dtype,
142                                  uint16_t dspec) {
143      uint32_t maxTfrSize = nvmeData_.getMaxTfrSize();
144      if ((maxTfrSize != 0) && (size > maxTfrSize)) {
145          throw std::invalid_argument("Exceeds max Transfer size");
146      }
147      // Clear the SQE entry to avoid some arbitrary flags being set.
148      memset(&sqe, 0, sizeof(struct io_uring_sqe));
149
150      sqe.fd = file_.fd();
151      sqe.opcode = IORING_OP_URING_CMD;
152      sqe.cmd_op = NVME_URING_CMD_IO;
153
154      struct nvme_uring_cmd* cmd = (struct nvme_uring_cmd*)&sqe.cmd;
155      if (cmd == nullptr) {
156          throw std::invalid_argument("Uring cmd is NULL!");
157      }
158      memset(cmd, 0, sizeof(struct nvme_uring_cmd));
159      cmd->opcode = opcode;
160
161      // start LBA of the IO = Req_start (offset in partition) + Partition_start
162      uint64_t sLba = (start >> nvmeData_.lbaShift()) + nvmeData_.partStartLba();
```

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

```
162     uint64_t sLba = (start >> nvmeData_.lbaShift()) + nvmeData_.partStartLba();
163     uint32_t nLb = (size >> nvmeData_.lbaShift()) - 1; // nLb is 0 based
164
165     /* cdw10 and cdw11 represent starting lba */
166     cmd->cdw10 = sLba & 0xffffffff;
167     cmd->cdw11 = sLba >> 32;
168     /* cdw12 represent number of lba's for read/write */
169     cmd->cdw12 = (dtype & 0xFF) << 20 | nLb;
170     cmd->cdw13 = (dspec << 16);
171     cmd->addr = (uint64_t)buf;
172     cmd->data_len = size;
173
174     cmd->nsid = nvmeData_.nsId();
175 }
```

```
185  void FdpNvme::prepWriteUringCmdSqe(
186     struct io_uring_sqe& sqe, void* buf, size_t size, off_t start, int handle)
187     static constexpr uint8_t kPlacementMode = 2;
188     uint16_t pid;
189
190     if (handle == -1) {
191         pid = getFdpPID(kDefaultPIDIdx); // Use the default stream
192     } else if (handle >= 0 && handle <= maxPIDIdx_) {
193         pid = getFdpPID(static_cast<uint16_t>(handle));
194     } else {
195         throw std::invalid_argument("Invalid placement identifier");
196     }
197
198     prepFdpUringCmdSqe(sqe, buf, size, start, nvme_cmd_write, kPlacementMode,
199                       pid);
200 }
```

49

⁴⁹ FdpNvme.cpp

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	Additional exemplary evidence showing Meta's CacheLib benefiting and/or taking advantage of FDP functionality, and to the extent necessary, that this limitation is met is found in the files bellowing showing for example, the Logical Block Address (LBA) from CacheLib: - FdpNvme.cpp⁵⁰ - Device.cpp⁵¹ - Device.h⁵² - RegionManager.cpp⁵³ - BigHash.h⁵⁴ - BigHash.cpp⁵⁵ - BlockCache.h⁵⁶ - BlockCache.cpp⁵⁷ Discovery into the specific storage devices, including but not limited to SSDs, used by Meta is ongoing, including the implementation of FDP and/or NVMe 2.1 and/or NVM Write Command by the SSD as identified above.
1[B5] subsequent to storing the content, receive a content request for the content, comprising at least the content	The Meta Accused Instrumentalities include a device controller operatively coupled with the storage media and configured to subsequent to storing the content, receiving a content request for the content, comprising at least the content identifier for the content. As described above for limitation 1[B4] (incorporated by reference here), the Meta Accused Instrumentalities include a FTL that stores translation between the logical and physical layer. Further, subsequent to storing the content, the SSD receives a content request for the content, comprising at least the content identifier for the content.

⁵⁰ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/FdpNvme.cpp>

⁵¹ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/Device.cpp>

⁵² <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/Device.h>

⁵³ https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/RegionManager.cpp

⁵⁴ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/bighash/BigHash.h>

⁵⁵ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/bighash/BigHash.cpp>

⁵⁶ https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/BlockCache.h

⁵⁷ https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/BlockCache.cpp

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

identifier for the content;	As explained above for previous limitations, the Accused Instrumentalities include a Flash Translation Layer that translate the host logical address into the actual physical NAND location where the data is stored. Discovery into the specific mechanism invoked by Meta Accused Instrumentalities is ongoing including but not limited to, the FTL mechanism used by the SSDs for data retrieval. As another example, the NVMe specification⁵⁸ includes protocols for reading the content stored in the SSD utilizing, for example, the SLBA field (including the LBA) and/or DPTR fields: 3.3.4 Read command The Read command reads data and metadata, if applicable, from the I/O controller for the LBAs indicated. The command may specify protection information to be checked as part of the read operation. The command uses Command Dword 2, Command Dword 3, Command Dword 10, Command Dword 11, Command Dword 12, Command Dword 13, Command Dword 14, and Command Dword 15 fields. If the command uses PRPs for the data transfer, then the Metadata Pointer, PRP Entry 1, and PRP Entry 2 fields are used. If the command uses SGLs for the data transfer, then the Metadata SGL Segment Pointer and SGL Entry 1 fields are used. Figure 50: Read – Data Pointer <table><tr><th>Bits</th><th>Description</th></tr><tr><td>127:00</td><td>Data Pointer (DPTR): This field specifies where data is transferred to. Refer to the Common Command Format figure in the NVM Express Base Specification for the definition of this field.</td></tr></table> Figure 52: Read – Command Dword 10 and Command Dword 11 <table><tr><th>Bits</th><th>Description</th></tr><tr><td>63:00</td><td>Starting LBA (SLBA): This field indicates the 64-bit address of the first logical block to be read as part of the operation. Command Dword 10 contains bits 31:00; Command Dword 11 contains bits 63: 32.</td></tr></table> Thus, the Accused Meta Instrumentalities include SSDs (including but not limited to the Samsung previous identified) that are configured such that subsequent to storing the content, receive a content request for the content, comprising at least the content identifier for the content.	Bits	Description	127:00	Data Pointer (DPTR): This field specifies where data is transferred to. Refer to the Common Command Format figure in the NVM Express Base Specification for the definition of this field.	Bits	Description	63:00	Starting LBA (SLBA): This field indicates the 64-bit address of the first logical block to be read as part of the operation. Command Dword 10 contains bits 31:00; Command Dword 11 contains bits 63: 32.
Bits	Description								
127:00	Data Pointer (DPTR): This field specifies where data is transferred to. Refer to the Common Command Format figure in the NVM Express Base Specification for the definition of this field.								
Bits	Description								
63:00	Starting LBA (SLBA): This field indicates the 64-bit address of the first logical block to be read as part of the operation. Command Dword 10 contains bits 31:00; Command Dword 11 contains bits 63: 32.								

⁵⁸ See, e.g. NVMe NVM Command Set Specification Revision 1.2 at § 3.3.4

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

As shown below, Meta CacheLib benefits and/or takes advantage of the Accused Instrumentalities' FDP functionality by, further providing, for example, the Logical Block Address (LBA) as part of retrieving content. To the extent it is argued that the asserted claims require the FDP functionality to be invoked and/or, the description below also demonstrates infringement.

The following exemplary code shows Meta's CacheLib benefitting and/or taking advantage of FDP functionality and, to the extent necessary, that this limitation is met:

```
299      // Device info vector for FDP support
300      const std::vector<std::shared_ptr<FdpNvme>> fdpNvmeVec_{};
```

```
966      iouringCmdOp->initBase();
967      struct io_uring_sqe& sqe = iouringCmdOp->getSqe();
968      if (req.opType_ == OpType::READ) {
969          fdpNvmeVec_[kDefaultFdpIdx]->prepReadUringCmdSqe(sqe, op.data_, op.size_,
970                                                         op.offset_);
971      } else {
972          fdpNvmeVec_[kDefaultFdpIdx]->prepWriteUringCmdSqe(
973              sqe, op.data_, op.size_, op.offset_, op.placeHandle_.value_or(-1));
974      }
975      io_uring_sqe_set_data(&sqe, iouringCmdOp.get());
976      return std::move(iouringCmdOp);
```

59

⁵⁹ Device.cpp

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

```
136  void FdpNvme::prepFdpUringCmdSqe(struct io_uring_sqe& sqe,  
137                                  void* buf,  
138                                  size_t size,  
139                                  off_t start,  
140                                  uint8_t opcode,  
141                                  uint8_t dtype,  
142                                  uint16_t dspec) {  
143      uint32_t maxTfrSize = nvmeData_.getMaxTfrSize();  
144      if ((maxTfrSize != 0) && (size > maxTfrSize)) {  
145          throw std::invalid_argument("Exceeds max Transfer size");  
146      }  
147      // Clear the SQE entry to avoid some arbitrary flags being set.  
148      memset(&sqe, 0, sizeof(struct io_uring_sqe));  
149  
150      sqe.fd = file_.fd();  
151      sqe.opcode = IORING_OP_URING_CMD;  
152      sqe.cmd_op = NVME_URING_CMD_IO;  
153  
154      struct nvme_uring_cmd* cmd = (struct nvme_uring_cmd*)&sqe.cmd;  
155      if (cmd == nullptr) {  
156          throw std::invalid_argument("Uring cmd is NULL!");  
157      }  
158      memset(cmd, 0, sizeof(struct nvme_uring_cmd));  
159      cmd->opcode = opcode;
```

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

```

160
161     // start LBA of the IO = Req_start (offset in partition) + Partition_start
162     uint64_t sLba = (start >> nvmeData_.lbaShift()) + nvmeData_.partStartLba();
163     uint32_t nLb = (size >> nvmeData_.lbaShift()) - 1; // nLb is 0 based
164
165     /* cdw10 and cdw11 represent starting lba */
166     cmd->cdw10 = sLba & 0xffffffff;
167     cmd->cdw11 = sLba >> 32;
168     /* cdw12 represent number of lba's for read/write */
169     cmd->cdw12 = (dtype & 0xFF) << 20 | nLb;
170     cmd->cdw13 = (dspec << 16);
171     cmd->addr = (uint64_t)buf;
172     cmd->data_len = size;
173
174     cmd->nsid = nvmeData_.nsId();
175 }

```

```

177  void FdpNvme::prepReadUringCmdSqe(struct io_uring_sqe& sqe,
178                                   void* buf,
179                                   size_t size,
180                                   off_t start) {
181     // Placement Handle is not used for read.
182     prepFdpUringCmdSqe(sqe, buf, size, start, nvme_cmd_read, 0, 0);
183 }

```

60

Additional exemplary evidence showing Meta's CacheLib benefitting and/or taking advantage of FDP functionality, and to the extent necessary, that this limitation is met:

- FdpNvme.cpp⁶¹
- Device.cpp⁶²

⁶⁰ FdpNvme.cpp

⁶¹ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/FdpNvme.cpp>

⁶² <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/Device.cpp>

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	- Device.h⁶³ - RegionManager.cpp⁶⁴ - BigHash.h⁶⁵ - BigHash.cpp⁶⁶ - BlockCache.h⁶⁷ - BlockCache.cpp⁶⁸ Discovery into the specific storage devices, including but not limited to SSDs, used by Meta is ongoing, including the implementation of FDP and/or NVMe 2.1 and/or NVM Read Command by the SSD as identified above.
1[B6] in response to the content request, retrieve from the memory of the device controller, the storage information for the content;	The Meta Accused Instrumentalities include a device controller operatively coupled with the storage media and configured to in response to the content request, retrieve from the memory of the device controller, the storage information for the content. As described above for limitation 1[B4] (incorporated by reference here), the Meta Accused Instrumentalities include a FTL that stores translation between the logical and physical layer. Further, in response to the content request, retrieve from the memory of the device controller, the storage information for the content. As explained above for previous limitations, the Accused Instrumentalities include a Flash Translation Layer that translate the host logical address into the actual physical NAND location where the content is stored and transfers the retrieved data to the location indicated by the SLBA and/or DPTR fields. Discovery into the specific storage devices, including but not limited to SSDs, used by Meta and the mechanism invoked by Meta Accused Instrumentalities is ongoing including but not limited to, the FTL mechanism used by the SSDs for data retrieval.

⁶³ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/Device.h>

⁶⁴ https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/RegionManager.cpp

⁶⁵ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/bighash/BigHash.h>

⁶⁶ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/bighash/BigHash.cpp>

⁶⁷ https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/BlockCache.h

⁶⁸ https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/BlockCache.cpp

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

1[B7] recover the content from the storage media, at least in part according to the storage information; and	The Meta Accused Instrumentalities include a device controller operatively coupled with the storage media and configured to recover the content from the storage media at least in part according to the storage information. As described above for limitation 1[B4] (incorporated by reference here), the Meta Accused Instrumentalities include a FTL that stores translation between the logical and physical layer. Further, the SSDs recover the content from the storage media, at least in part according to the storage information. As explained above for previous limitations, the Accused Instrumentalities include a Flash Translation Layer that translate the host logical address into the actual physical NAND location where the content is stored and transfers the retrieved data to the location indicated by the SLBA and/or DPTR fields. Discovery into the specific storage devices, including but not limited to SSDs, used by Meta and the specific mechanism invoked by Meta Accused Instrumentalities is ongoing including but not limited to the FTL mechanism used by the SSDs for data retrieval. In addition to the above, this limitation is also shown by Meta's CacheLib successfully storing and retrieving data from FDP supported SSDs.
1[B8] respond to the content request at least by returning the content to the host.	The Meta Accused Instrumentalities include a device controller operatively coupled with the storage media and configured to respond to the content request at least by returning the content to the host. As described above for limitation 1[B4] (incorporated by reference here), the Meta Accused Instrumentalities include a FTL that stores translation between the logical and physical layer. Further, the SSDs respond to the content request at least by returning the content to the host. As explained above for previous limitations, the Accused Instrumentalities include a Flash Translation Layer that translate the host logical address into the actual physical NAND location where the content is stored and transfers the retrieved data to the location indicated by the SLBA and/or DPTR fields. Discovery into the specific storage devices, including but not limited to SSDs, used by Meta and the specific mechanism invoked by Meta Accused Instrumentalities is ongoing including but not limited to the FTL mechanism used by the SSDs for data retrieval.

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

	In addition to the above, this limitation is also shown by Meta's CacheLib successfully storing and retrieving data from FDP supported SSDs. As shown below, Meta's CacheLib benefits and/or takes advantage of the FDP functionality and further provides, retrieving content from the SSD. To the extent it is argued that the asserted claims require the FDP functionality to be invoked and/or used, the description below also demonstrates infringement. For example, Meta's CacheLib's successfully storing and retrieving data using the Accused Instrumentalities' FDP confirms that Meta benefits and/or takes advantage of the Accused Instrumentalities' FDP functionality. This also shows, to the extent necessary, that infringement occurs. Additional exemplary evidence showing Meta's CacheLib benefiting and/or and taking advantage of FDP, and to the extent necessary, that this limitation is found in: - FdpNvme.cpp⁶⁹ - Device.cpp⁷⁰ - Device.h⁷¹ - RegionManager.cpp⁷² - BigHash.h⁷³ - BigHash.cpp⁷⁴ - BlockCache.h⁷⁵ - BlockCache.cpp⁷⁶
--	--

⁶⁹ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/FdpNvme.cpp>

⁷⁰ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/Device.cpp>

⁷¹ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/common/Device.h>

⁷² https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/RegionManager.cpp

⁷³ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/bighash/BigHash.h>

⁷⁴ <https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/bighash/BigHash.cpp>

⁷⁵ https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/BlockCache.h

⁷⁶ https://github.com/facebook/CacheLib/blob/v2026.06.01.00/cachelib/navy/block_cache/BlockCache.cpp

Exhibit A: Meta’s Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

’023 Patent Claim 9	’023 Patent Accused Instrumentalities
9[P] An apparatus comprising:	This limitation is met by the Meta Accused Instrumentalities for the same reasons described above for limitation 1[P].
9[A] a storage media;	This limitation is met by the Meta Accused Instrumentalities for the same reasons described above for limitation 1[A].
9[B] a device controller operatively coupled with the storage media configured to at least:	This limitation is met by the Meta Accused Instrumentalities for the same reasons described above for limitation 1[B].
9[B1] receive, from a host, at least one storage device policy, wherein the at least one storage device policy indicates at least a recording method for writing content to the storage media;	This limitation is met by the Meta Accused Instrumentalities for the same reasons described above for limitation 1[B1].

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

9[B2] store content on the storage media by utilizing the recording method indicated by the at least one storage device policy;	This limitation is met by the Meta Accused Instrumentalities for the same reasons described above for limitation 1[B3] pursuant to a storage request as described above for limitation 1[B2].
9[B3] record storage information for the content on a memory of the device controller, comprising at least a content identifier for the content;	This limitation is met by the Meta Accused Instrumentalities for the same reasons described above for limitation 1[B4].
9[B4] subsequent to storing the content, receive a content request for the content, comprising at least the content identifier for the content;	This limitation is met by the Meta Accused Instrumentalities for the same reasons described above for limitation 1[B5].

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

9[B5] in response to the content request, retrieve from the memory of the device controller the storage information for the content;	This limitation is met by the Meta Accused Instrumentalities for the same reasons described above for limitation 1[B6].
9[B6] recover the content from the storage media, according to the storage information and according to the at least one storage device policy; and	This limitation is met by the Meta Accused Instrumentalities for the same reasons described above for limitation 1[B7]. Further, the recovery is according to the storage information and according to the at least one storage device policy as described above for limitations 9[B1] and 9[B3].
9[B7] respond to the content request at least by returning the content to the host.	This limitation is met by the Meta Accused Instrumentalities for the same reasons described above for limitation 1[B8].

Exhibit A: Meta's Infringement of U.S. Patent No. 10,776,023 – 7/29/2026

'023 Patent Claim 14	'023 Patent Accused Instrumentalities
14[A] The apparatus of claim 9, wherein the storage media comprises a solid-state memory device.	The Meta Accused Instrumentalities are SSDs whereby the storage media comprises a solid-state memory device, including for example, non-volatile flash memory. For example, Samsung's PM9D3a is described as using "... Samsung's eight-generation three bit V-NAND technology ..." Discovery into the specific SSDs used by Meta is ongoing. To the extent Meta uses other SSDs, those SSDs comprise a solid state memory device, including NAND.