US 20120239853A1

(19) **United States**

(12) **Patent Application Publication**

**Moshayedi**

(10) **Pub. No.: US 2012/0239853 A1**

(43) **Pub. Date: Sep. 20, 2012**

(54) **SOLID STATE DEVICE WITH ALLOCATED FLASH CACHE**

(75) Inventor: **Mark Moshayedi**, Newport Coast, CA (US)

(73) Assignee: **STEC, INC.**, Santa Ana, CA (US)

(21) Appl. No.: **12/492,110**

(22) Filed: **Jun. 25, 2009**

**Related U.S. Application Data**

(60) Provisional application No. 61/075,709, filed on Jun. 25, 2008.

**Publication Classification**

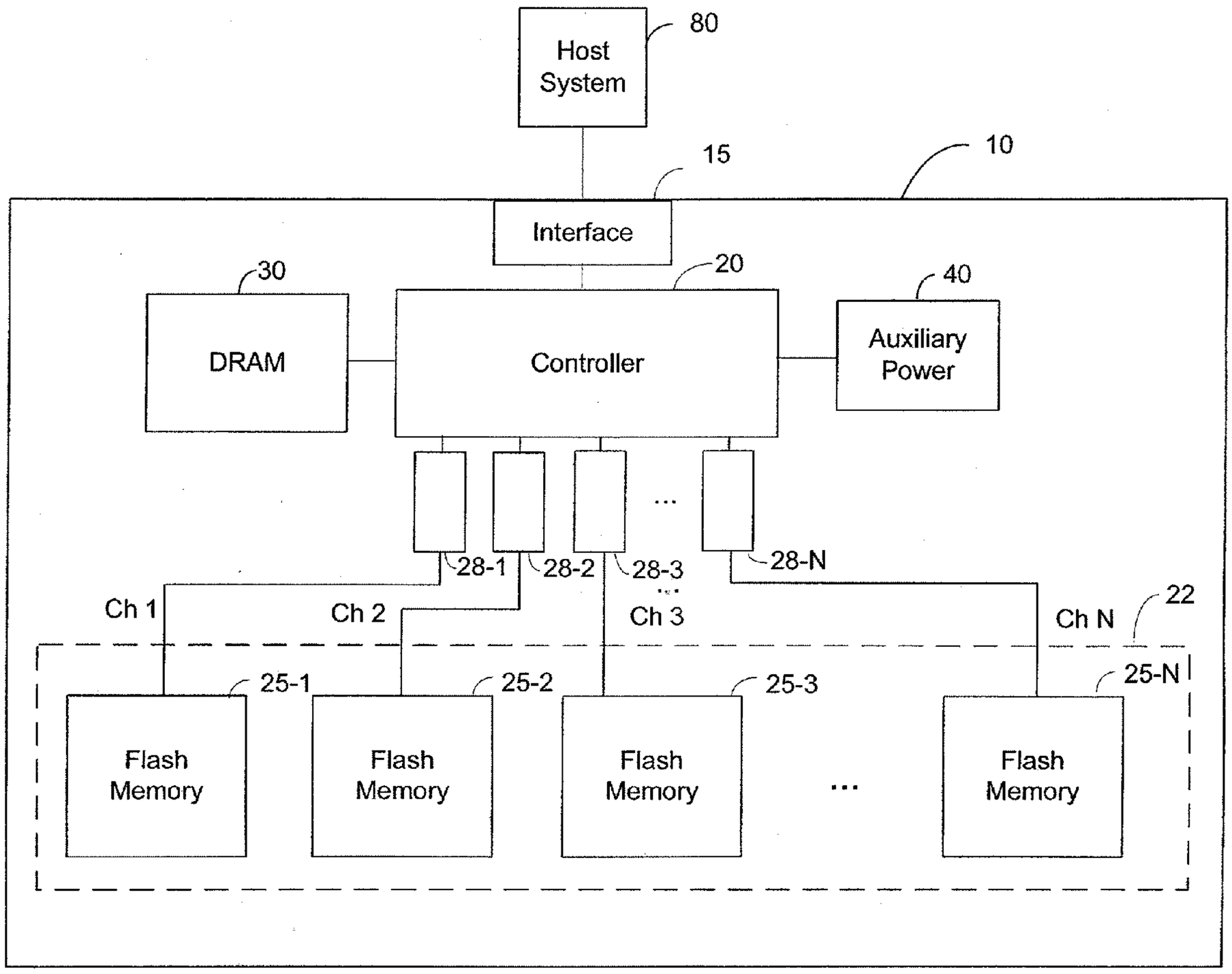
(51) **Int. Cl.**

| | |
|---|---|
| ***G06F 12/00*** | (2006.01) |
| ***G06F 12/08*** | (2006.01) |
| ***G06F 12/02*** | (2006.01) |

(52) **U.S. Cl.** ..... **711/103**; 711/118; 711/E12.001; 711/E12.008; 711/E12.017

(57) **ABSTRACT**

A flash storage device, and methods for a flash storage device, having improved write performance are provided. Data is received from a host system, the data comprising a data segment, the data segment is temporarily stored in a data buffer of the random access memory, the data segment is assigned to a logical block address, and the data segment is written to an allocated cache portion of the flash memory. Subsequently, the data segment is written from the allocated cache portion of the flash memory to a main storage portion of the flash memory.



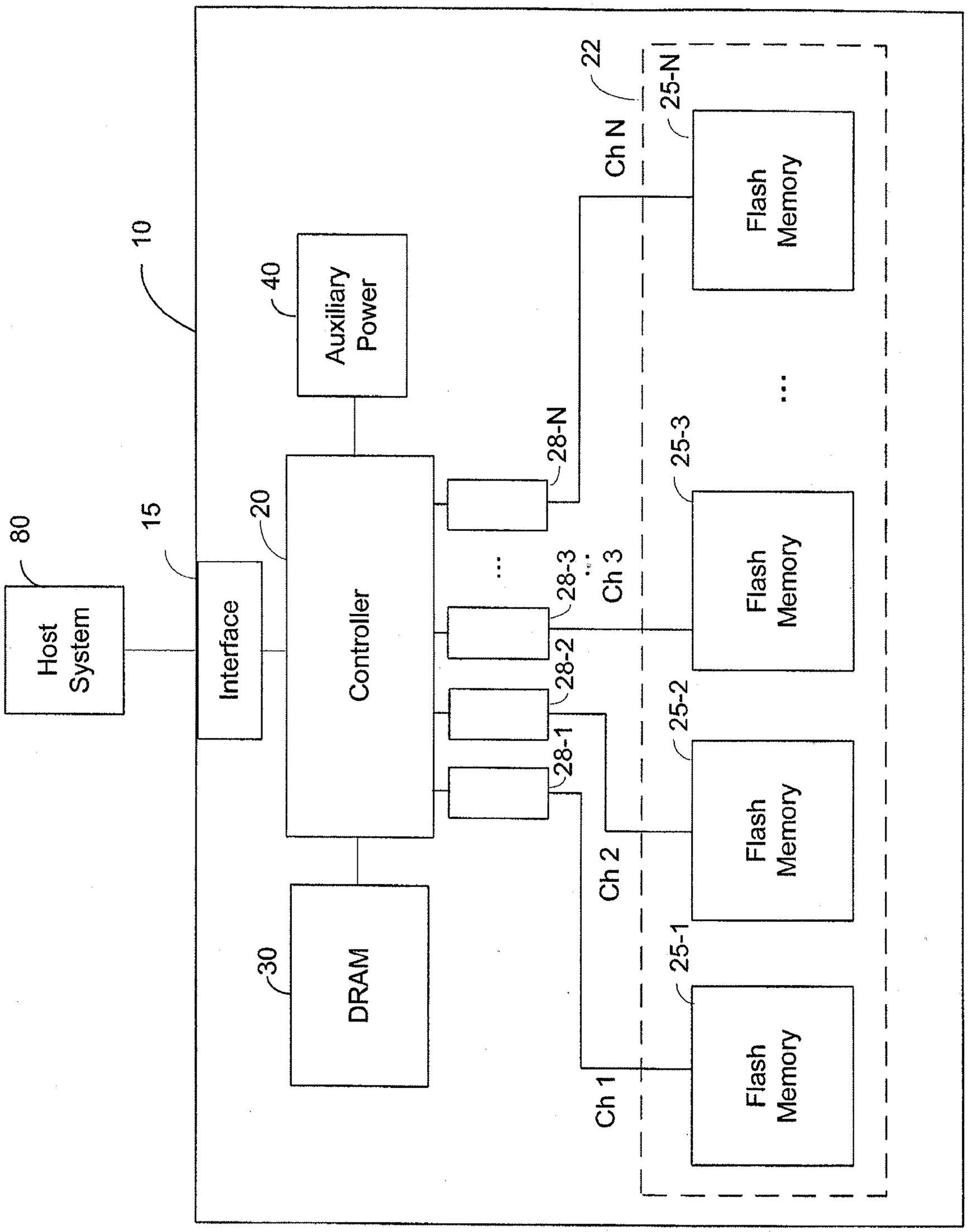

10
40
Auxiliary Power
80
Host System
15
Interface
20
Controller
30
DRAM
22
Ch N
25-N
Flash Memory
28-N
28-3
Ch 3
25-3
28-2
Ch 2
25-2
28-1
Ch 1
25-1

FIG. 1

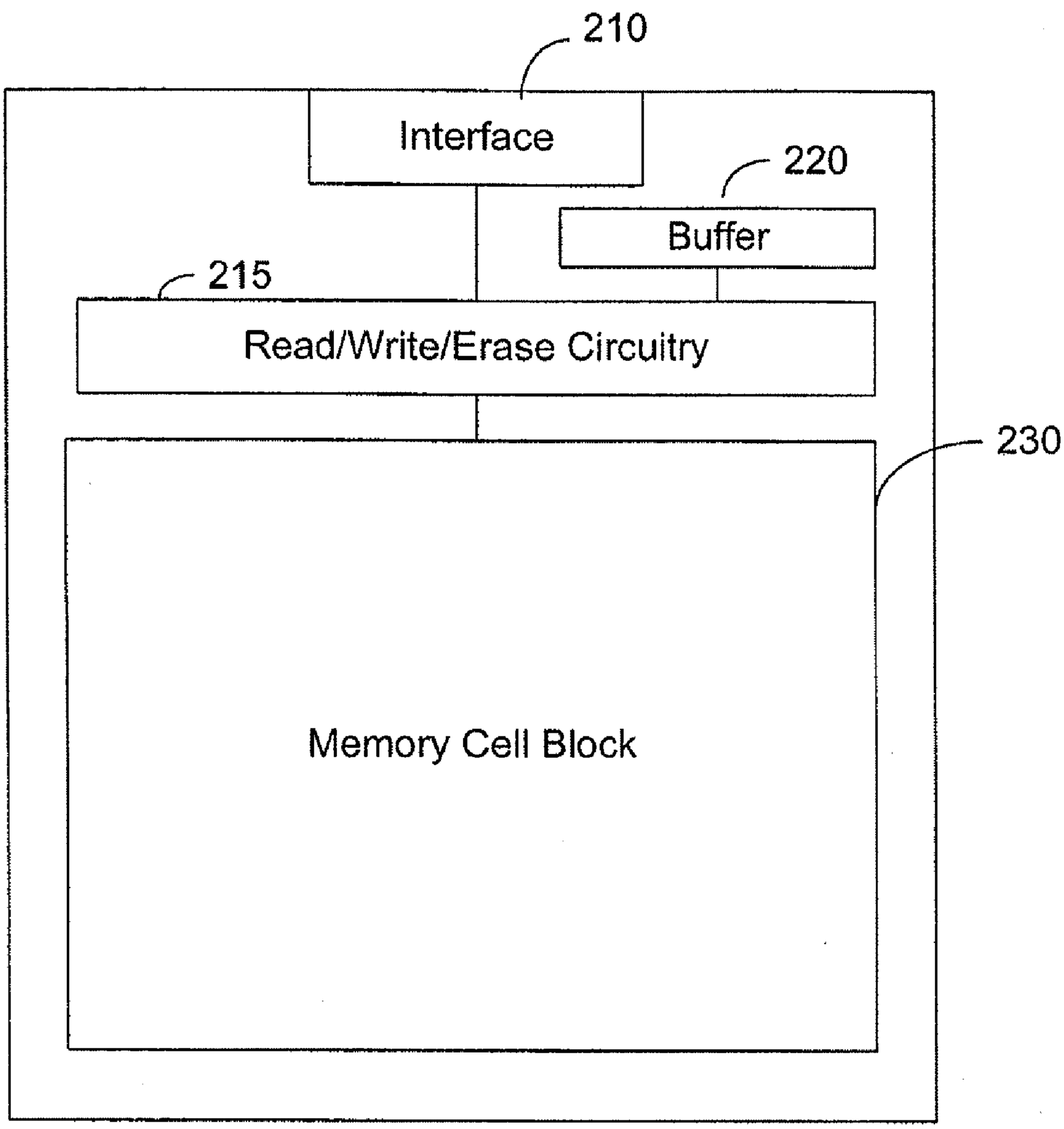

25
210
Interface
220
Buffer
215
Read/Write/Erase Circuitry
230
Memory Cell Block

FIG. 2

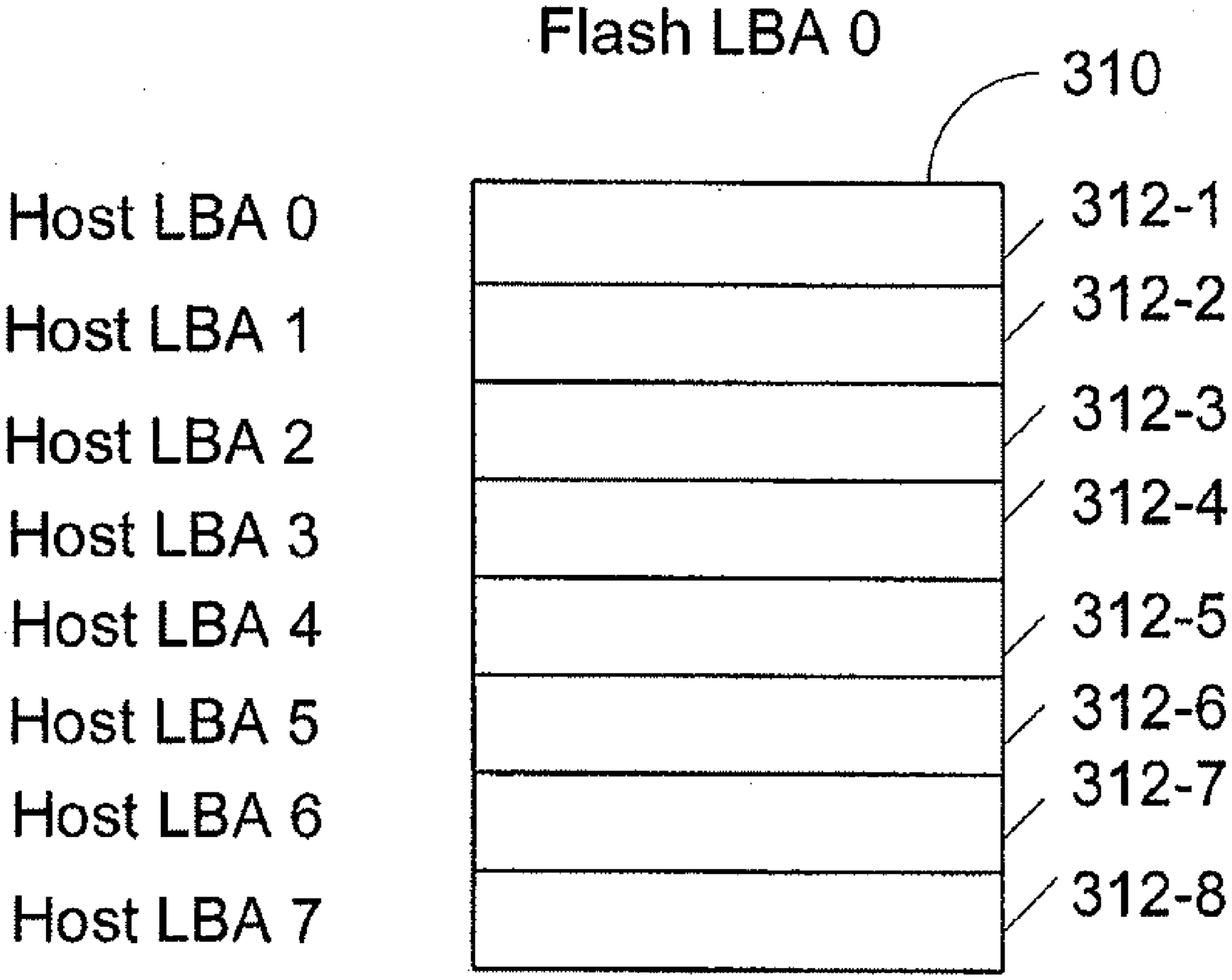

Flash LBA 0
310
Host LBA 0
Host LBA 1
Host LBA 2
Host LBA 3
Host LBA 4
Host LBA 5
Host LBA 6
Host LBA 7
312-1
312-2
312-3
312-4
312-5
312-6
312-7
312-8

FIG. 3A

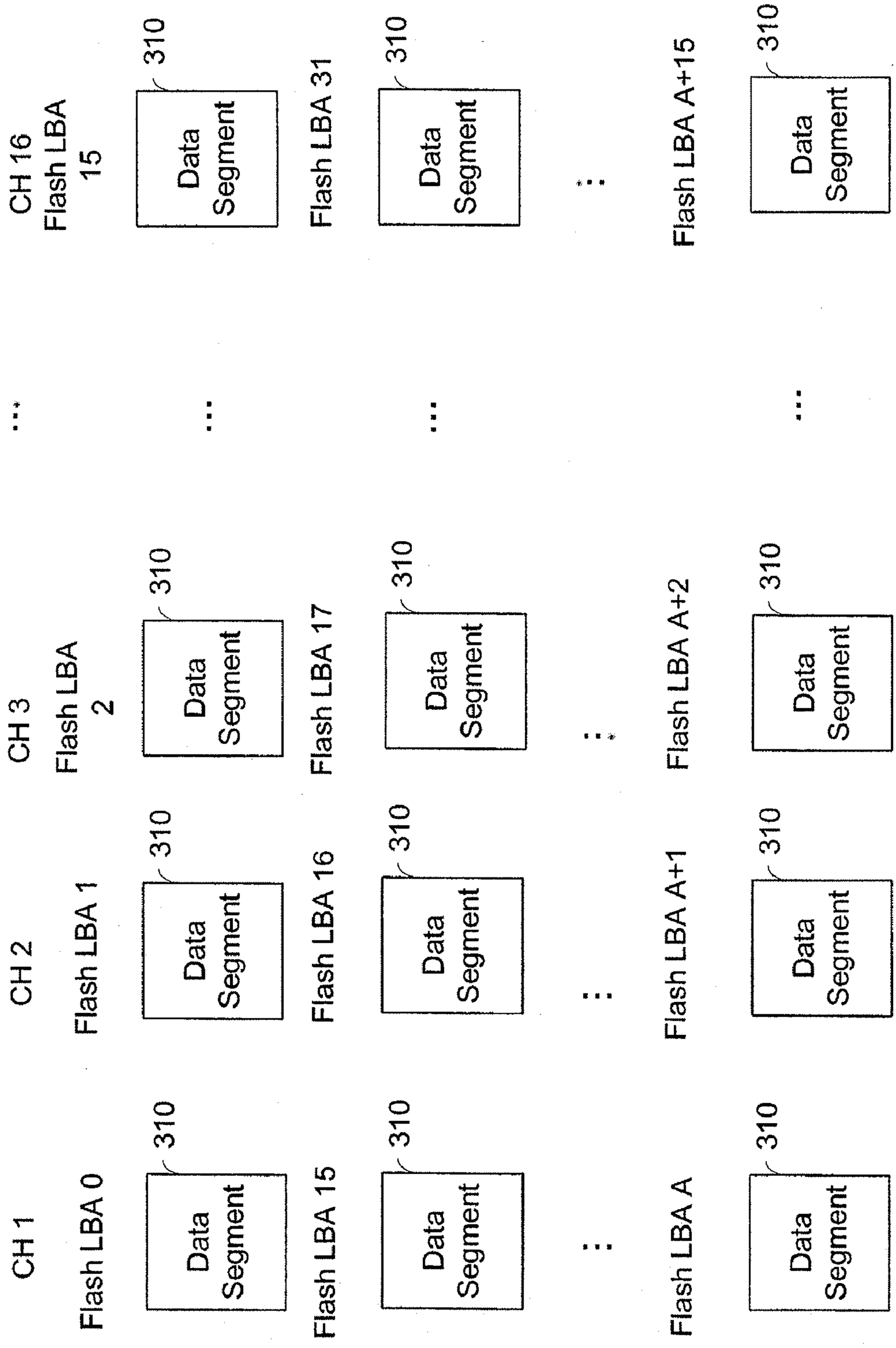

CH 1
CH 2
CH 3
CH 16
Flash LBA 0
Flash LBA 1
Flash LBA 2
Flash LBA 15
Flash LBA 16
Flash LBA 17
Flash LBA A
Flash LBA A+1
Flash LBA A+2
Flash LBA 31
Flash LBA A+15
310
Data Segment

FIG. 3B

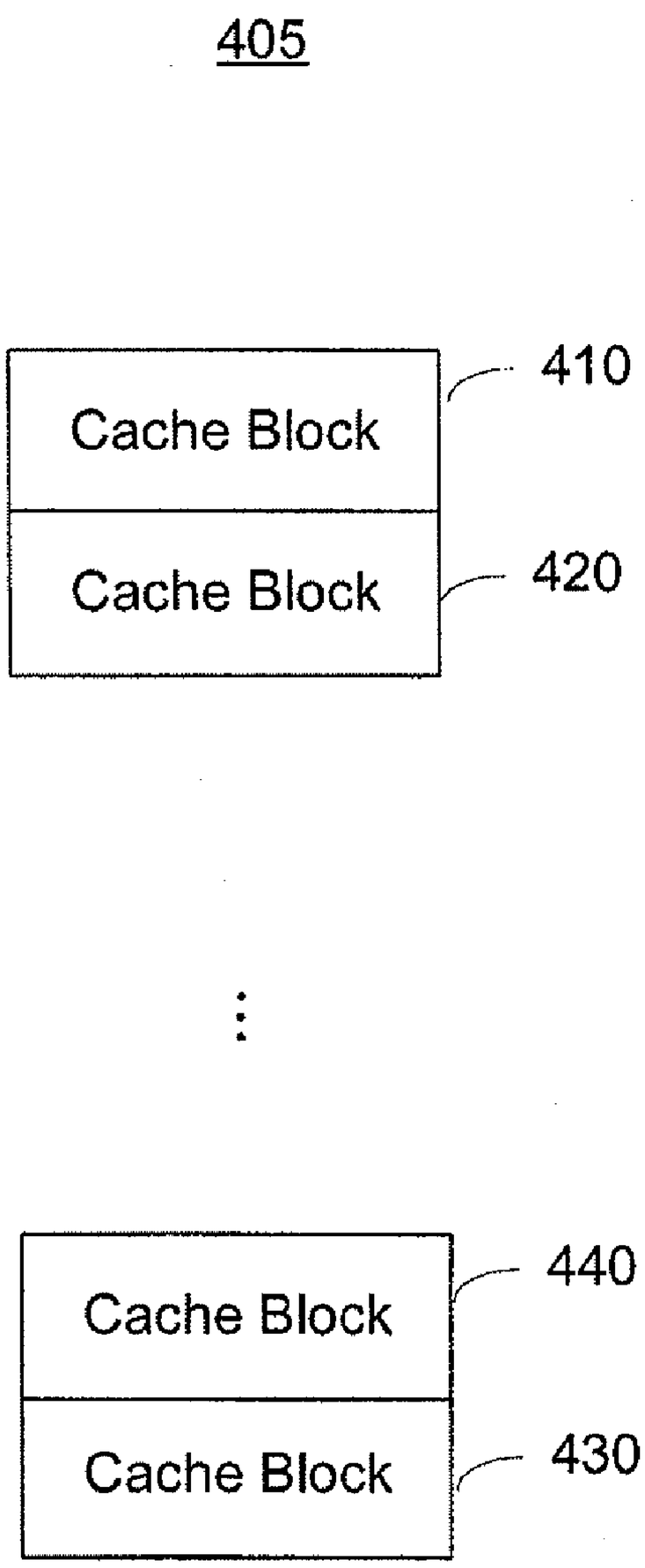

405
Cache Block
410
Cache Block
420
Cache Block
440
Cache Block
430

FIG. 4A

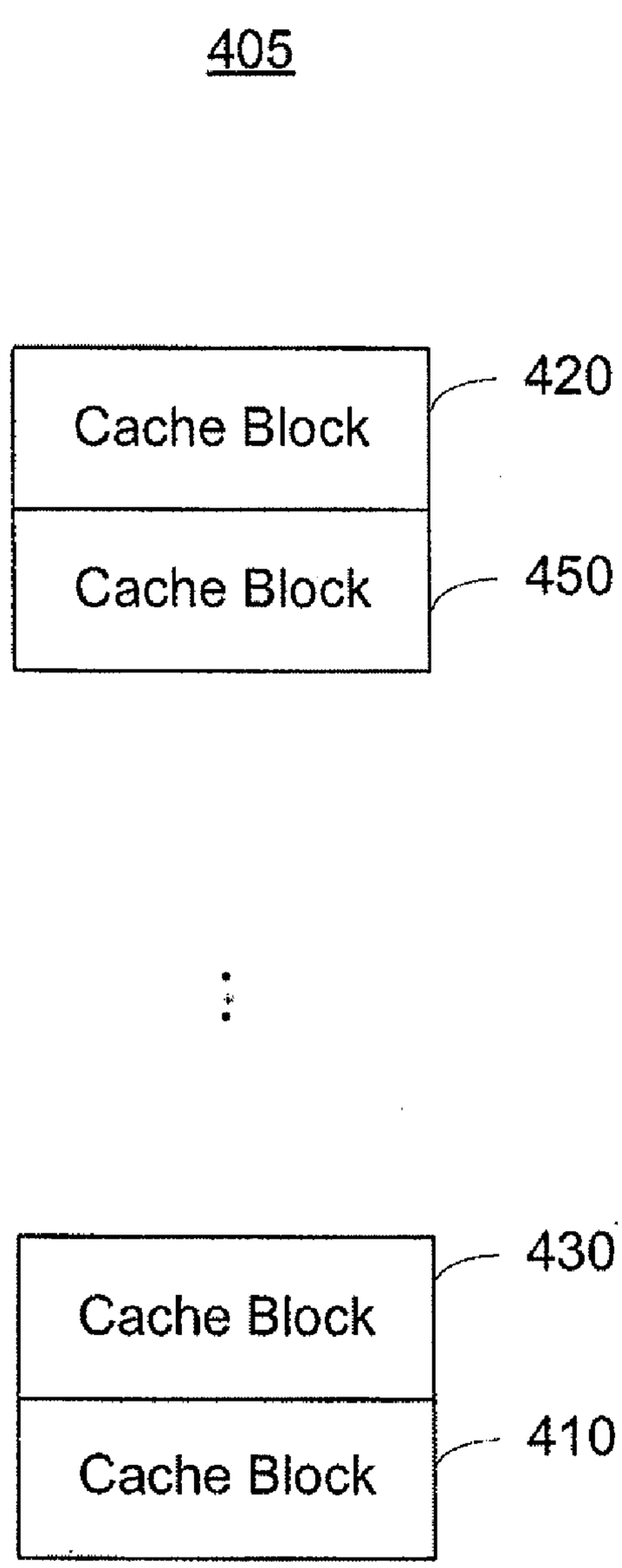

405
Cache Block
420
Cache Block
450
Cache Block
430
Cache Block
410

FIG. 4B

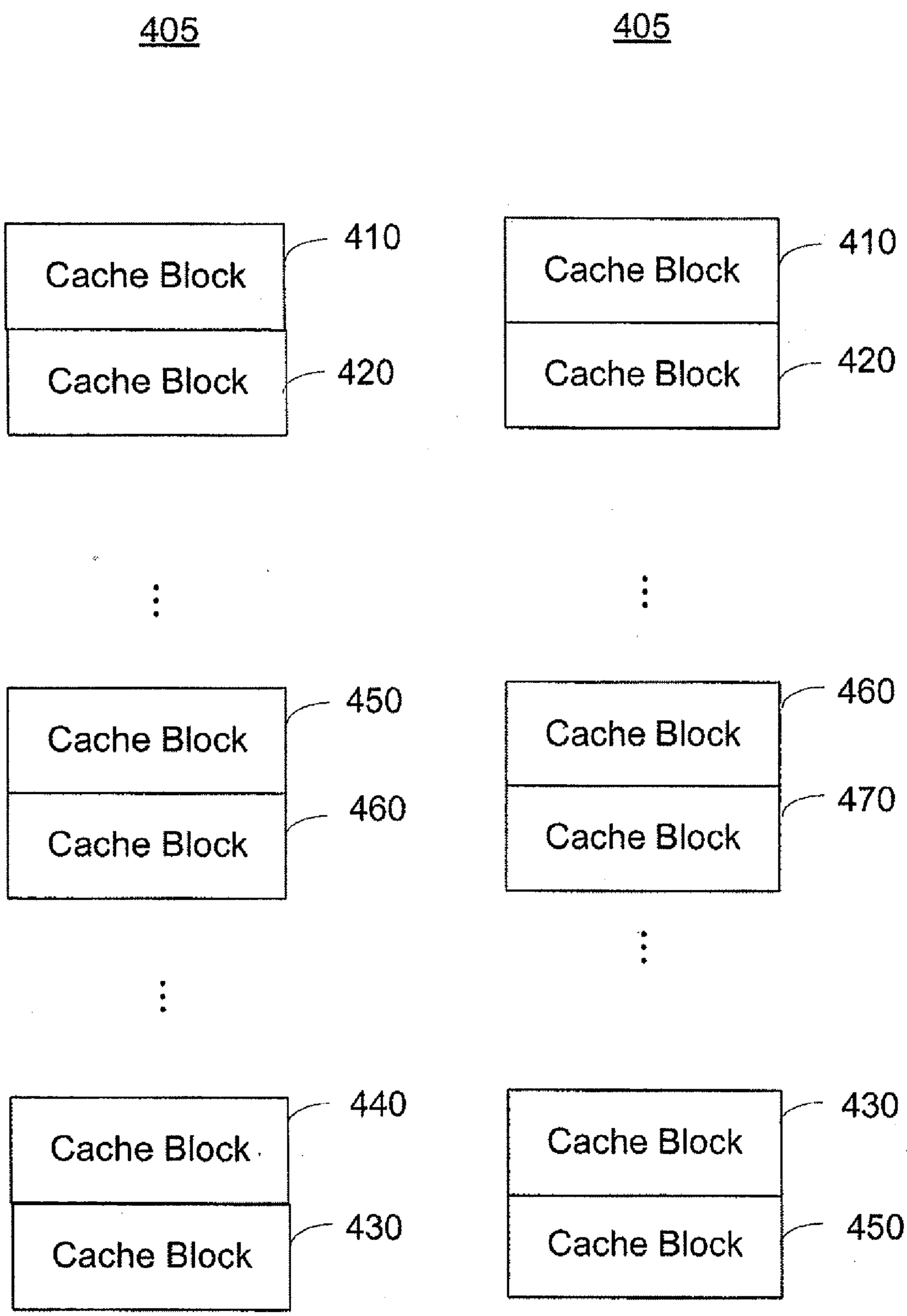

405
Cache Block
410
Cache Block
420
Cache Block
450
Cache Block
460
Cache Block
440
Cache Block
430

FIG. 5A


405
Cache Block
410
Cache Block
420
Cache Block
460
Cache Block
470
Cache Block
430
Cache Block
450

FIG. 5B

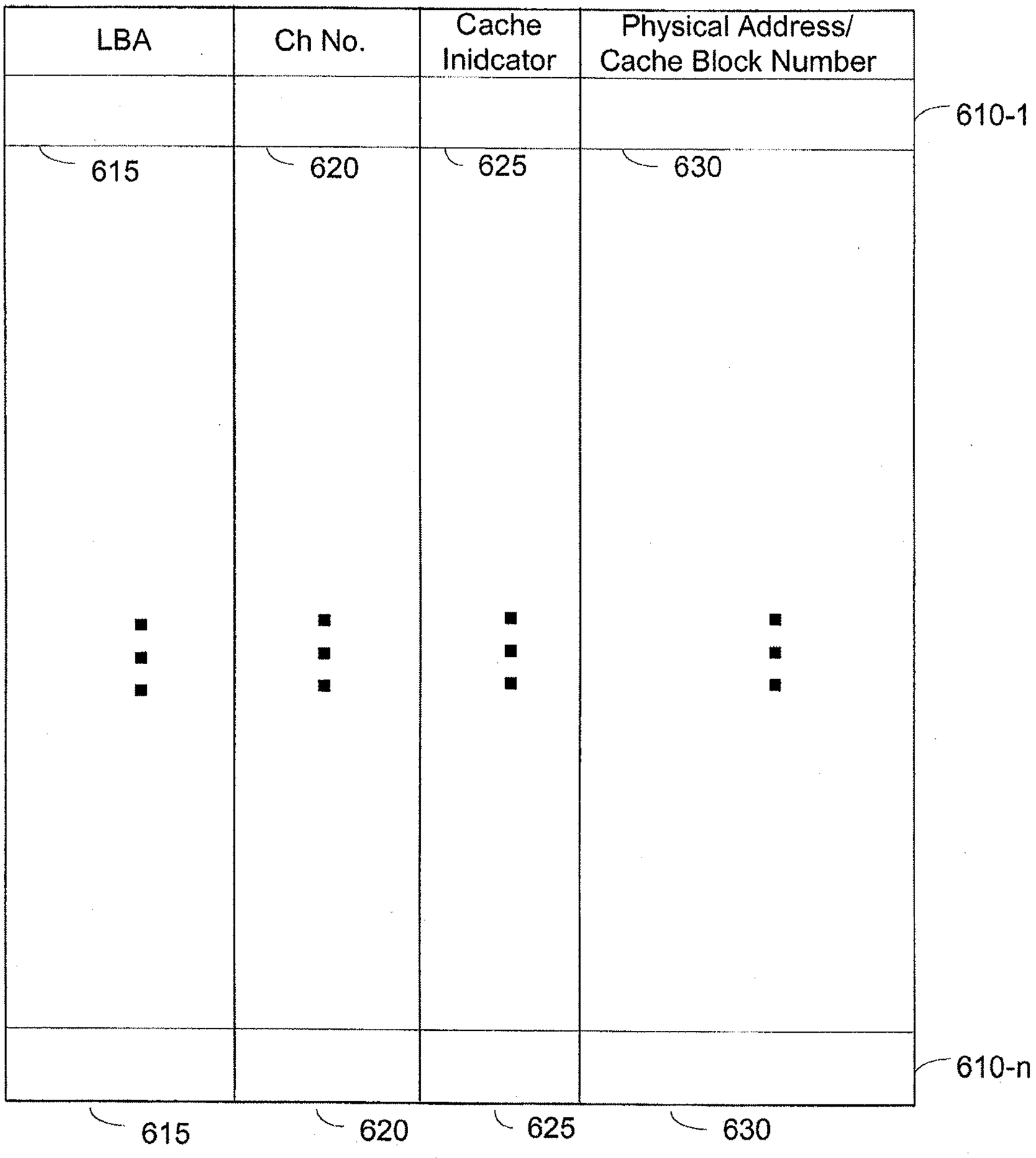

605
LBA
Ch No.
Cache Inidcator
Physical Address/ Cache Block Number
610-1
615
620
625
630
610-n
615
620
625
630

FIG. 6

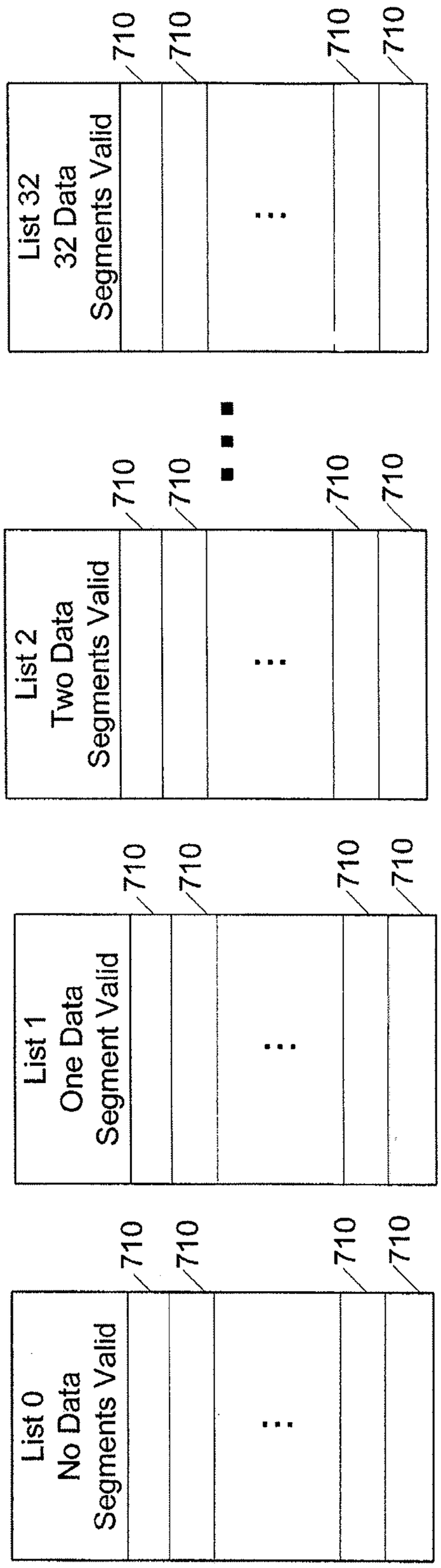

List 32
32 Data
Segments Valid
List 2
Two Data
Segments Valid
List 1
One Data
Segment Valid
List 0
No Data
Segments Valid
710

FIG. 7

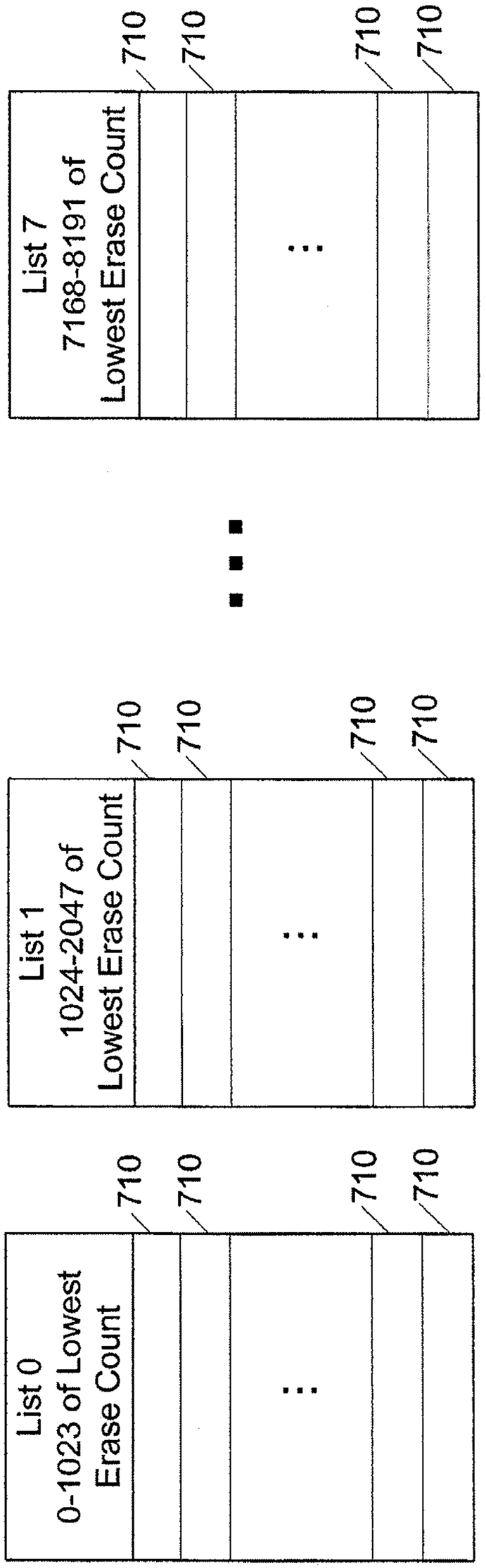

List 7
7168-8191 of
Lowest Erase Count
List 1
1024-2047 of
Lowest Erase Count
List 0
0-1023 of Lowest
Erase Count
710

FIG. 8

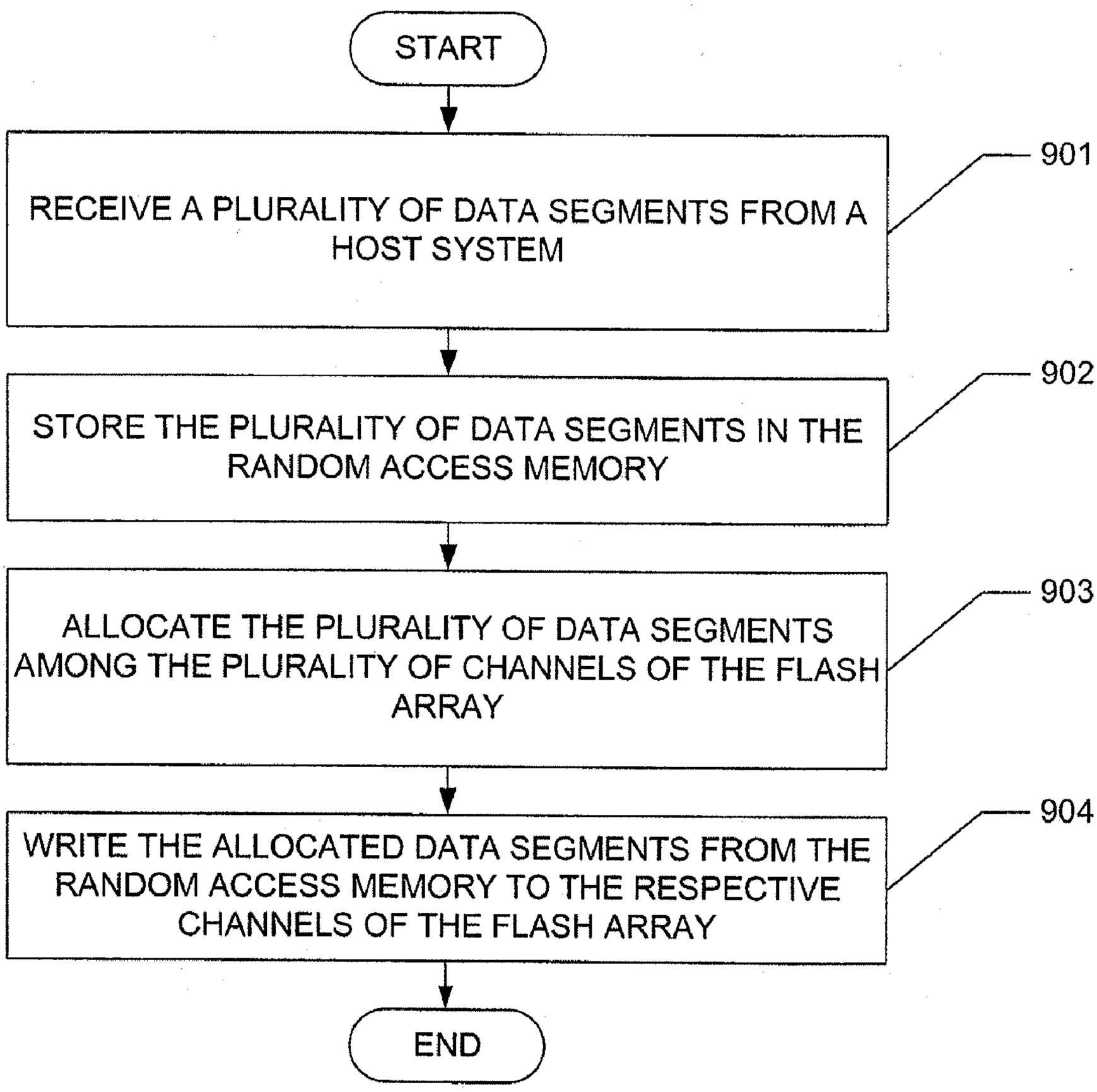

START
RECEIVE A PLURALITY OF DATA SEGMENTS FROM A HOST SYSTEM
901
STORE THE PLURALITY OF DATA SEGMENTS IN THE RANDOM ACCESS MEMORY
902
ALLOCATE THE PLURALITY OF DATA SEGMENTS AMONG THE PLURALITY OF CHANNELS OF THE FLASH ARRAY
903
WRITE THE ALLOCATED DATA SEGMENTS FROM THE RANDOM ACCESS MEMORY TO THE RESPECTIVE CHANNELS OF THE FLASH ARRAY
904
END

Fig. 9A

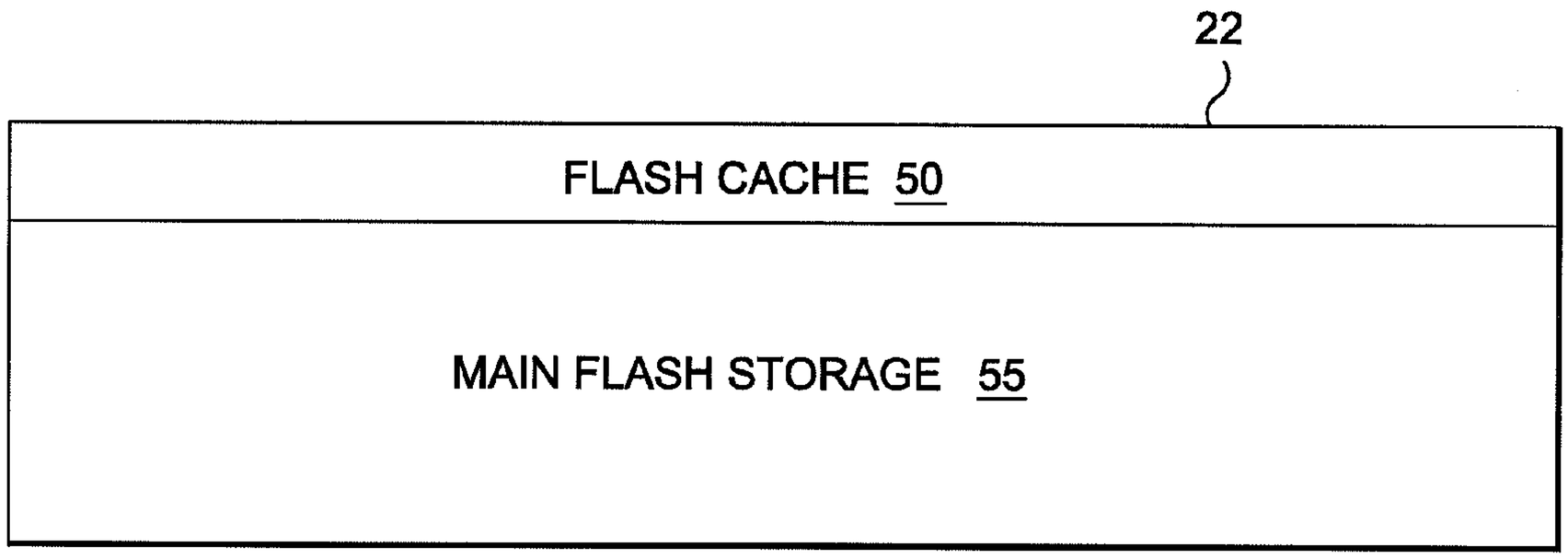



FIG. 9B

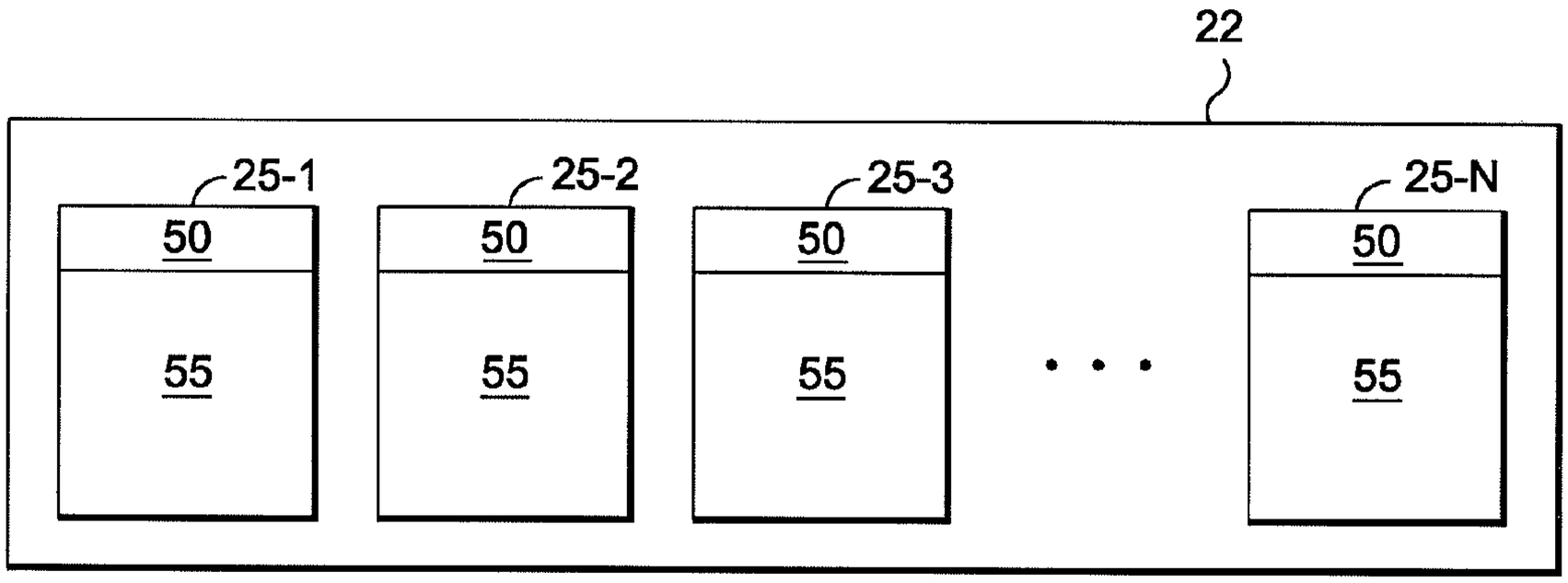



FIG. 9C

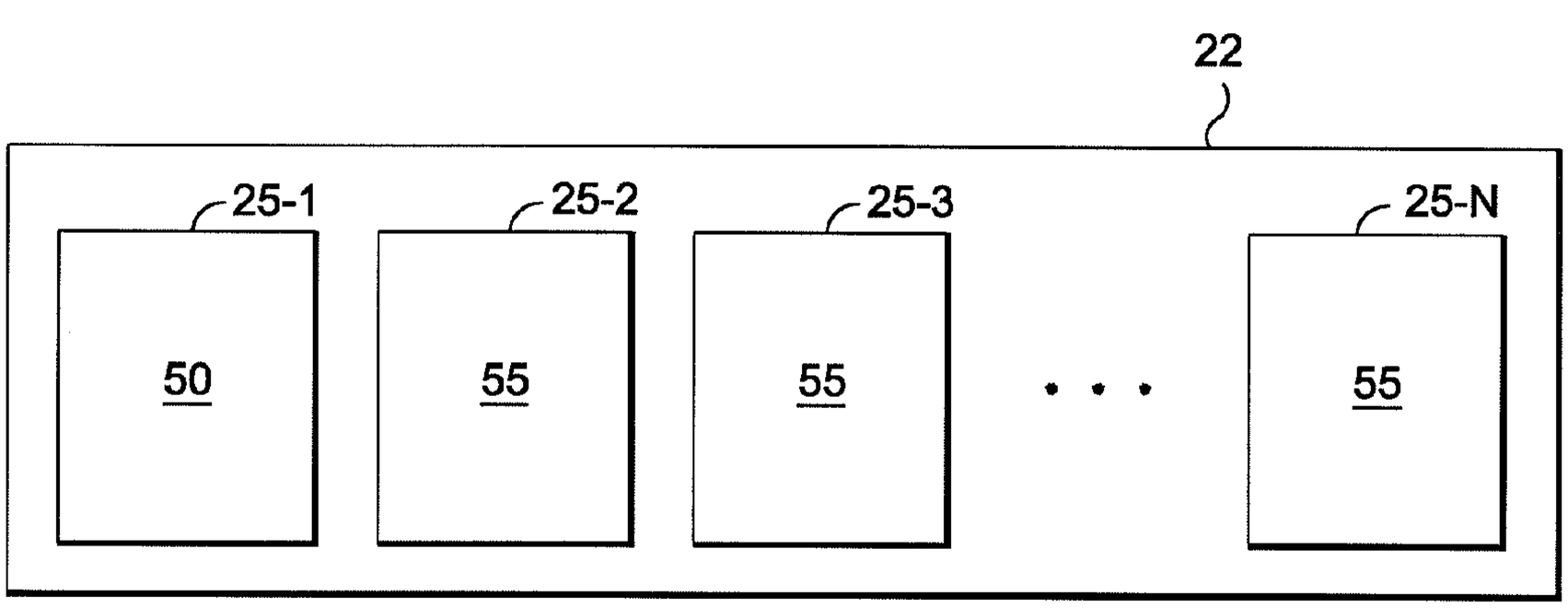



FIG. 9D

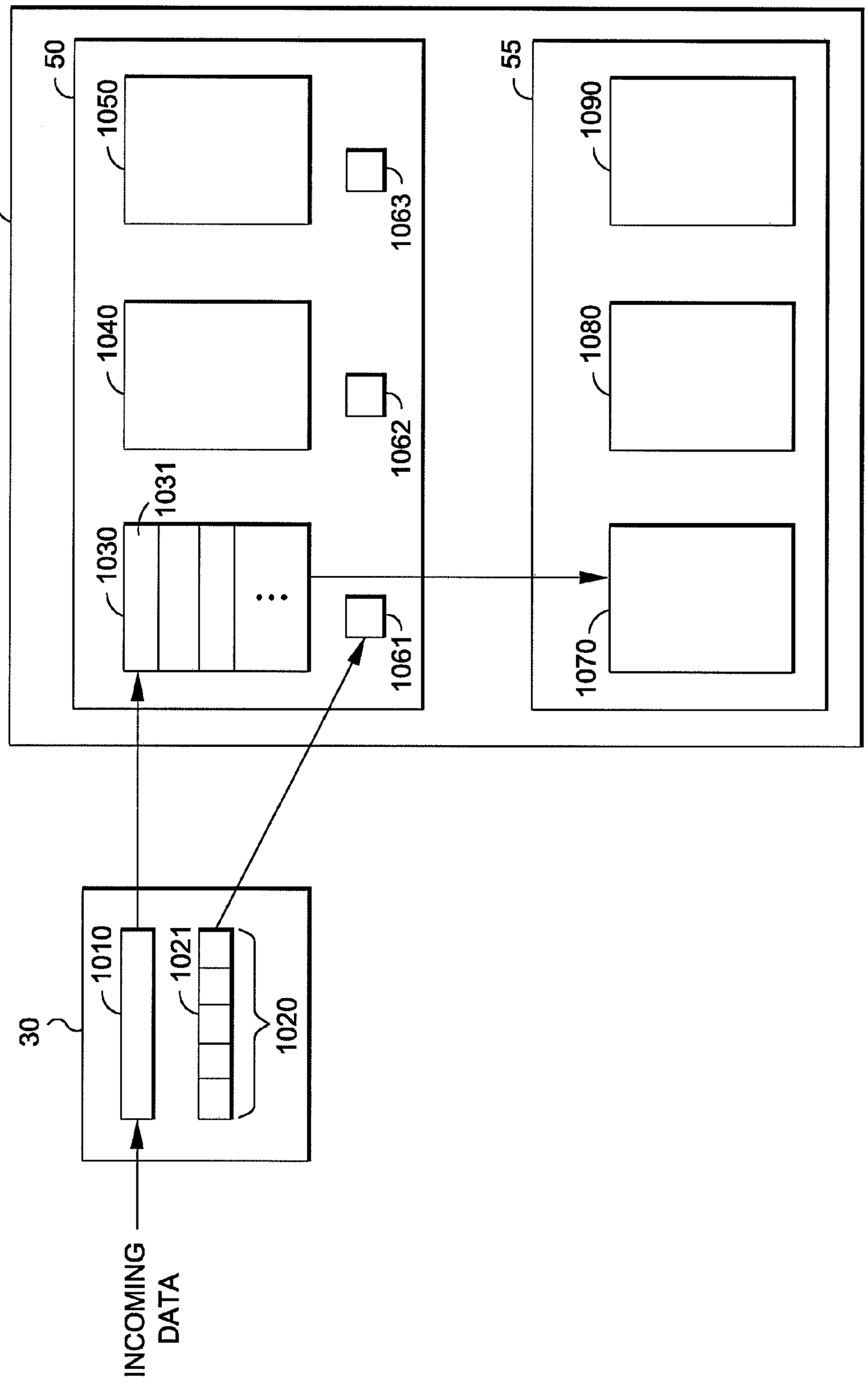

25-1 to 25-N
50
1050
1063
1040
1062
1031
1030
1061
55
1090
1080
1070
30
1010
1021
1020
INCOMING DATA

FIG. 10

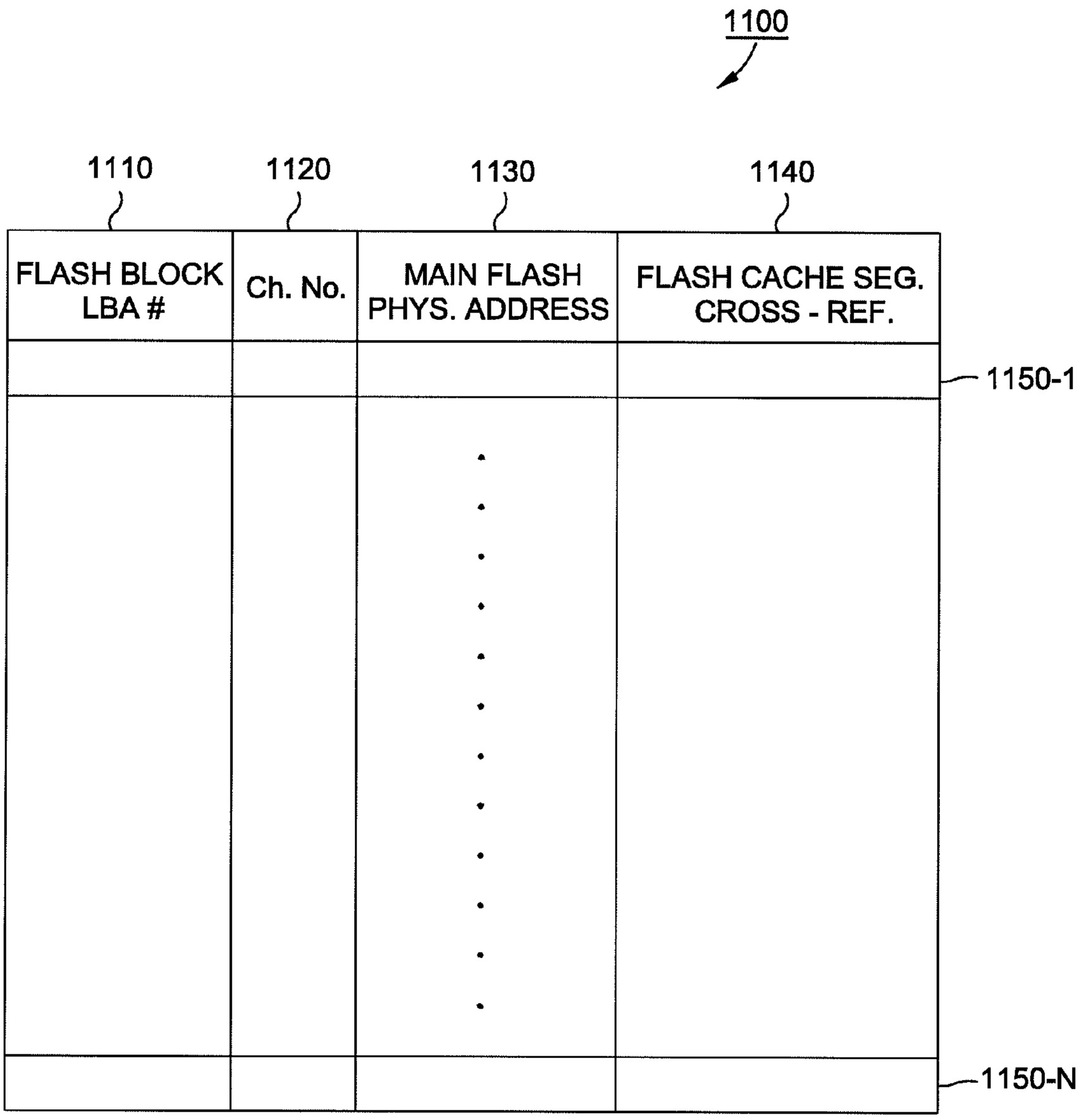

1100
1110
1120
1130
1140
FLASH BLOCK LBA #
Ch. No.
MAIN FLASH PHYS. ADDRESS
FLASH CACHE SEG. CROSS - REF.
1150-1
1150-N

FIG. 11A

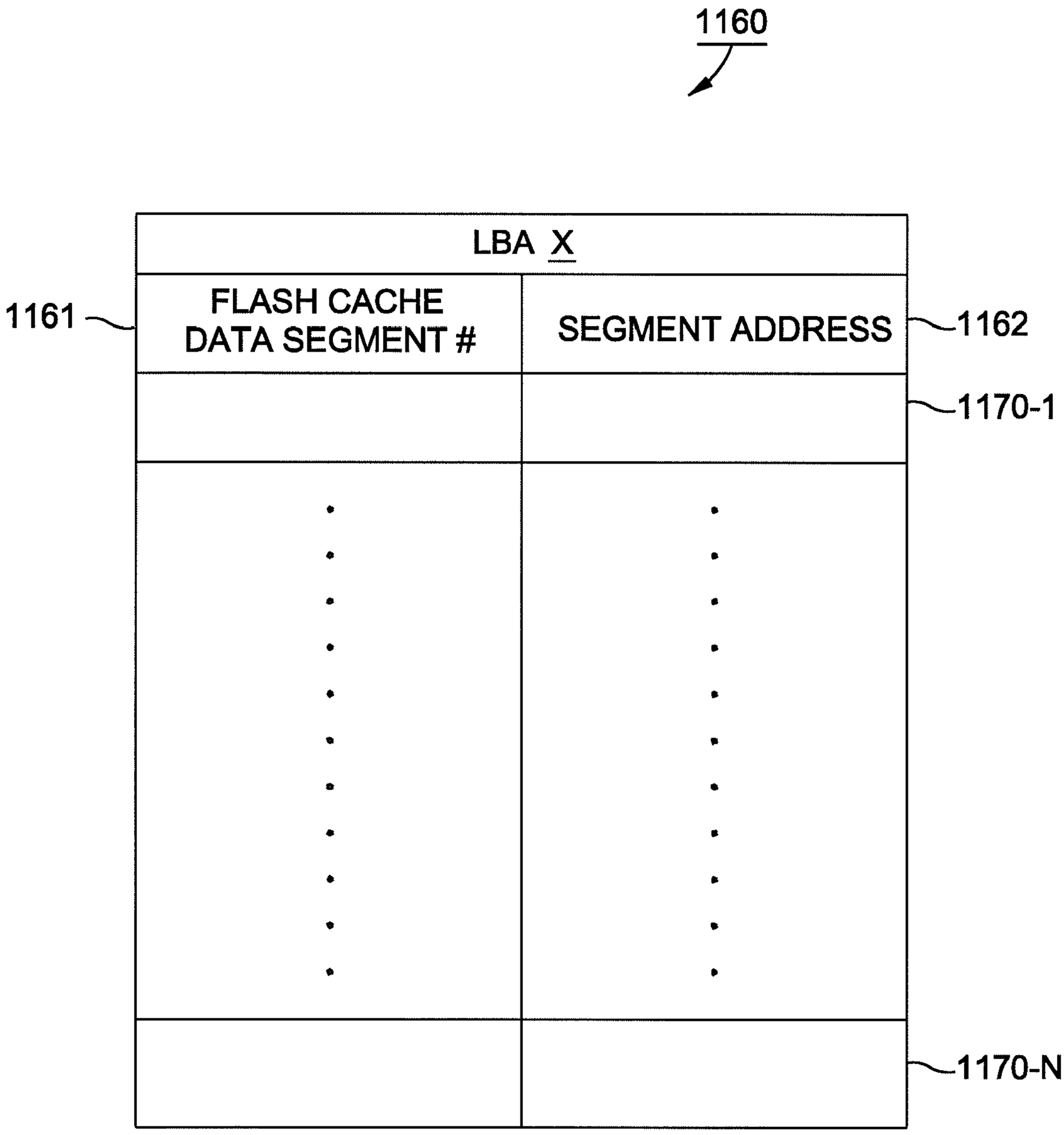

1160
LBA X
1161
FLASH CACHE DATA SEGMENT #
SEGMENT ADDRESS
1162
1170-1
1170-N

FIG. 11B

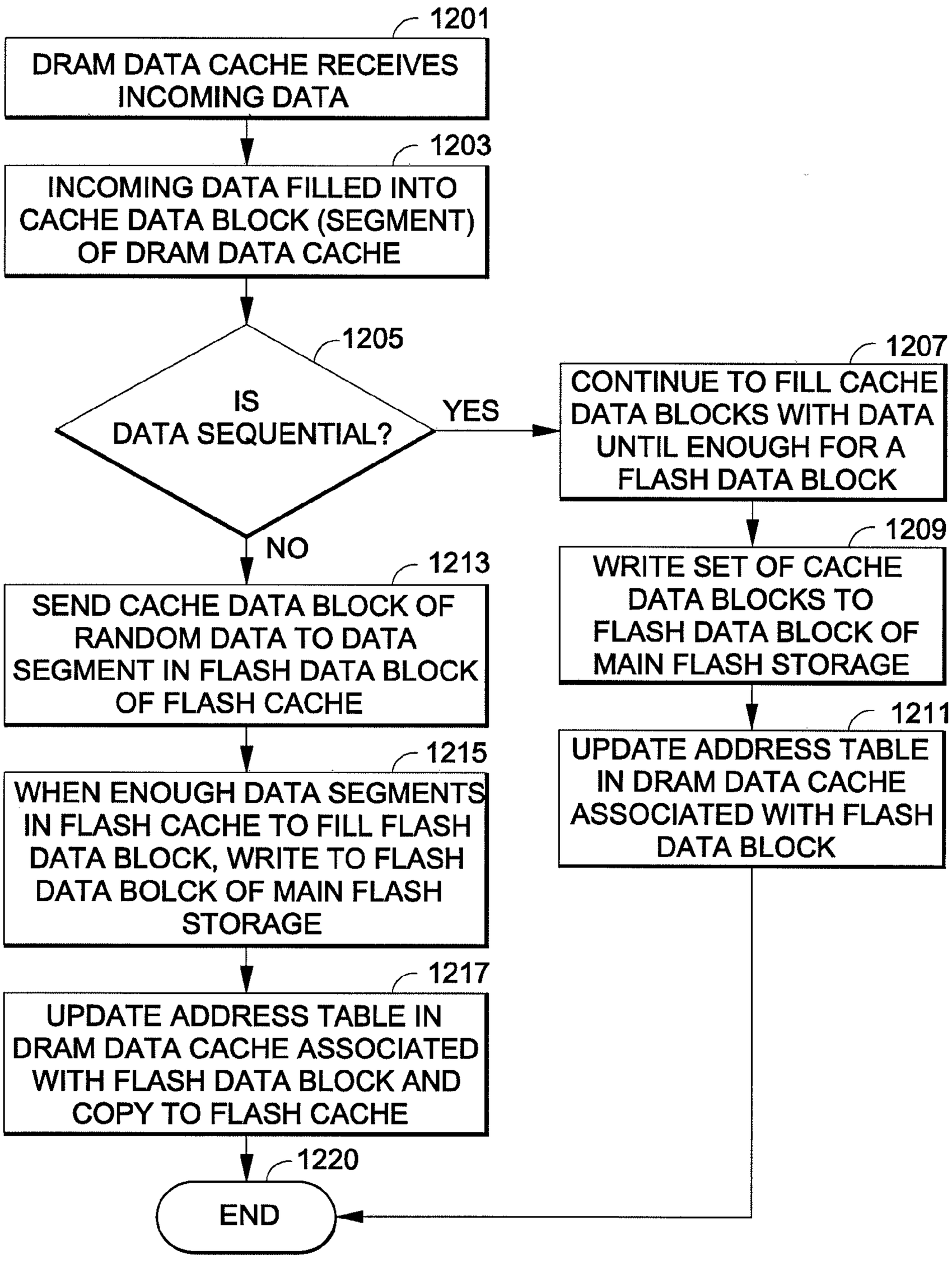

1201
DRAM DATA CACHE RECEIVES INCOMING DATA
1203
INCOMING DATA FILLED INTO CACHE DATA BLOCK (SEGMENT) OF DRAM DATA CACHE
1205
IS DATA SEQUENTIAL?
YES
NO
1207
CONTINUE TO FILL CACHE DATA BLOCKS WITH DATA UNTIL ENOUGH FOR A FLASH DATA BLOCK
1209
WRITE SET OF CACHE DATA BLOCKS TO FLASH DATA BLOCK OF MAIN FLASH STORAGE
1211
UPDATE ADDRESS TABLE IN DRAM DATA CACHE ASSOCIATED WITH FLASH DATA BLOCK
1213
SEND CACHE DATA BLOCK OF RANDOM DATA TO DATA SEGMENT IN FLASH DATA BLOCK OF FLASH CACHE
1215
WHEN ENOUGH DATA SEGMENTS IN FLASH CACHE TO FILL FLASH DATA BLOCK, WRITE TO FLASH DATA BOLCK OF MAIN FLASH STORAGE
1217
UPDATE ADDRESS TABLE IN DRAM DATA CACHE ASSOCIATED WITH FLASH DATA BLOCK AND COPY TO FLASH CACHE
1220
END

FIG. 12

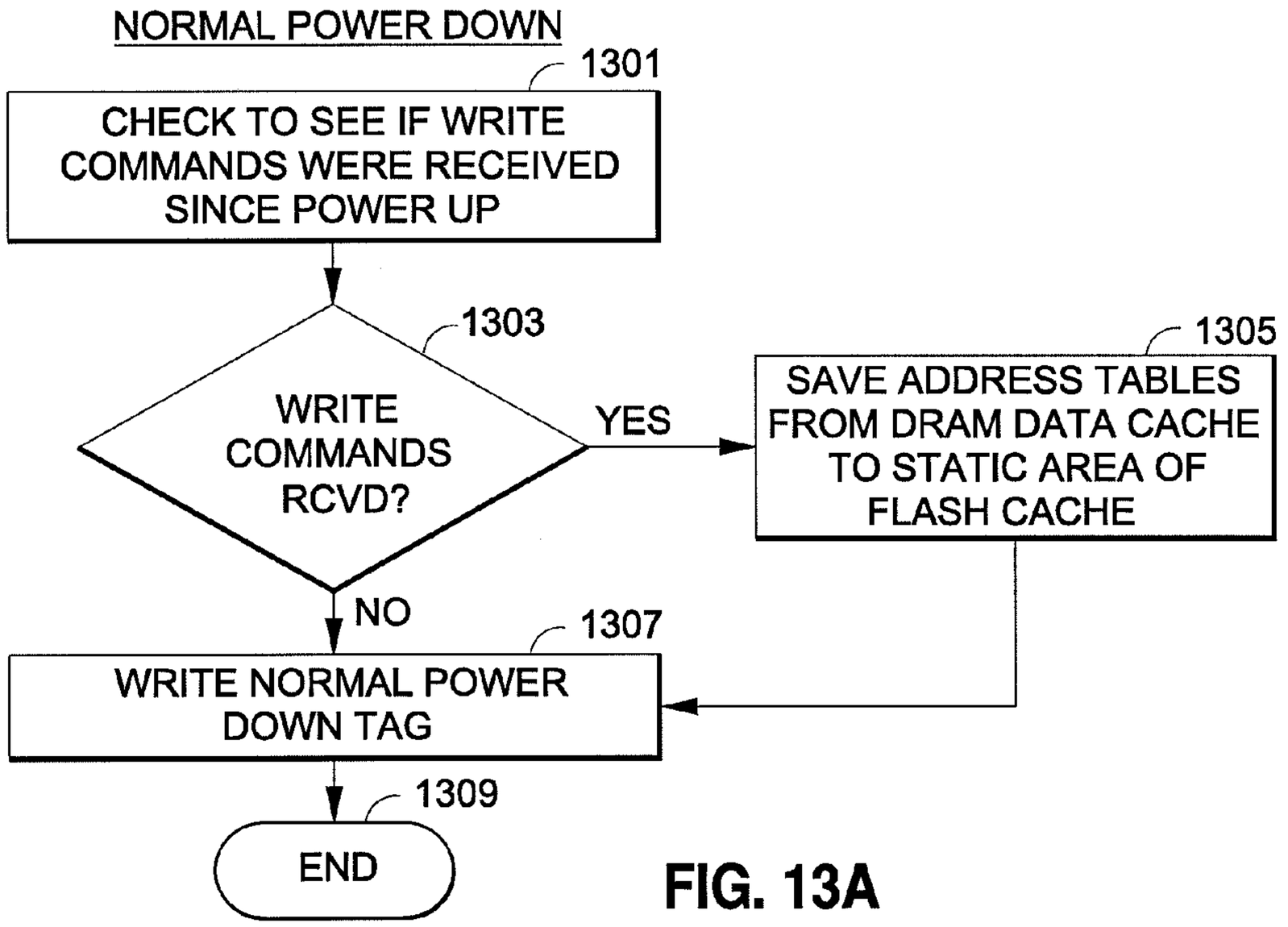



FIG. 13A

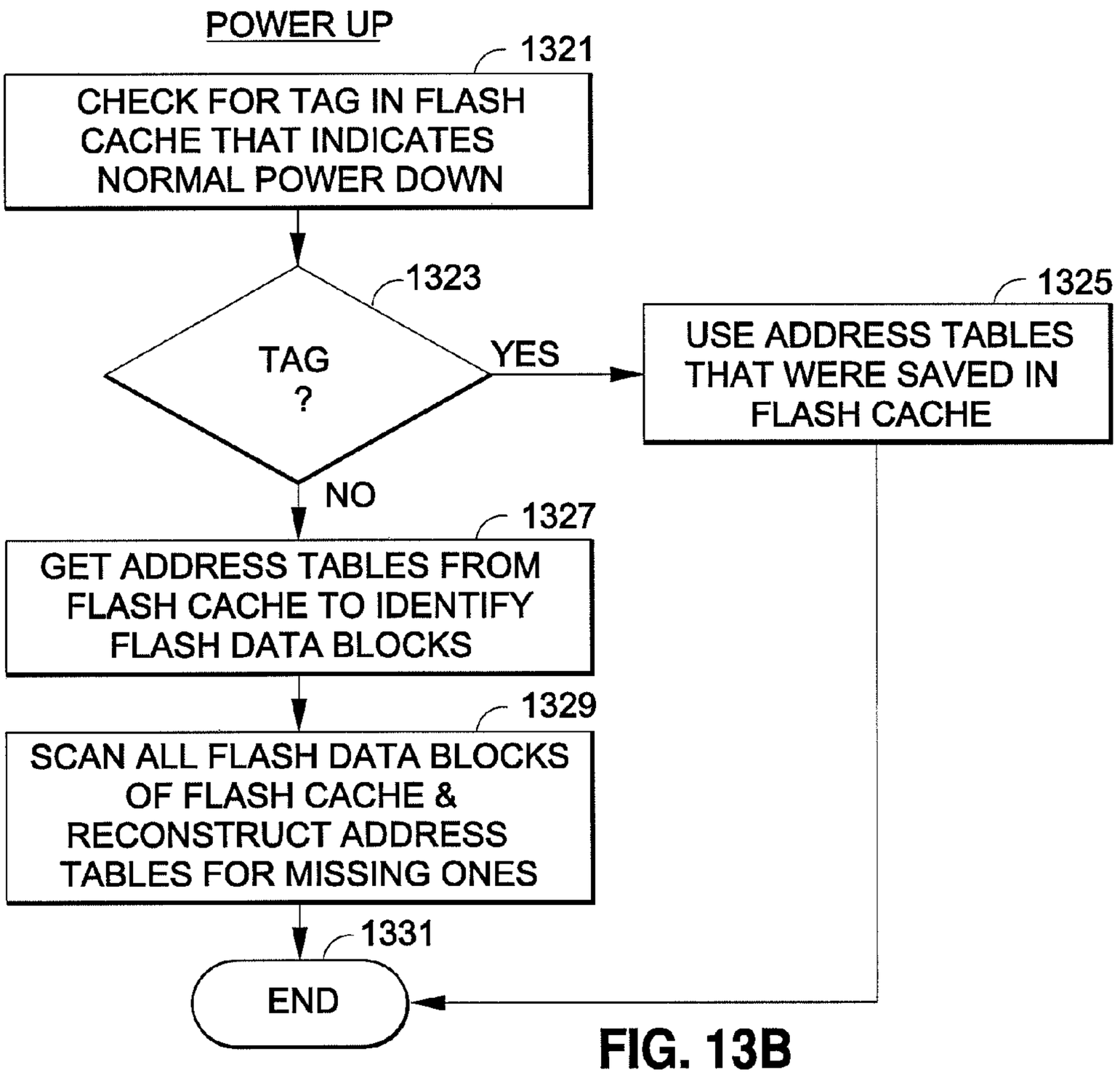



FIG. 13B

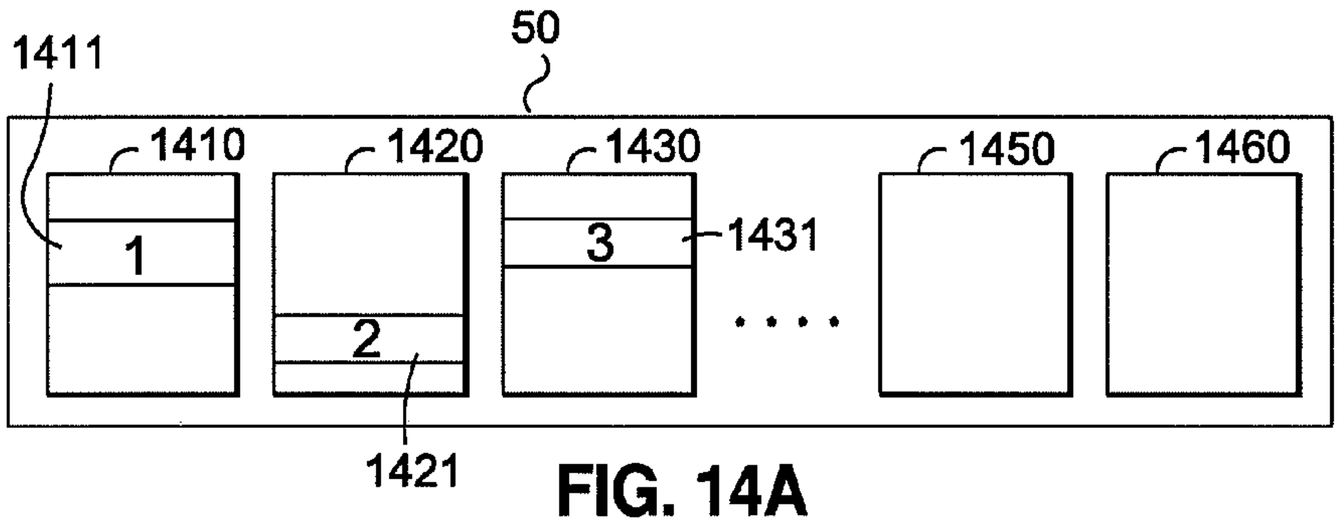



FIG. 14A

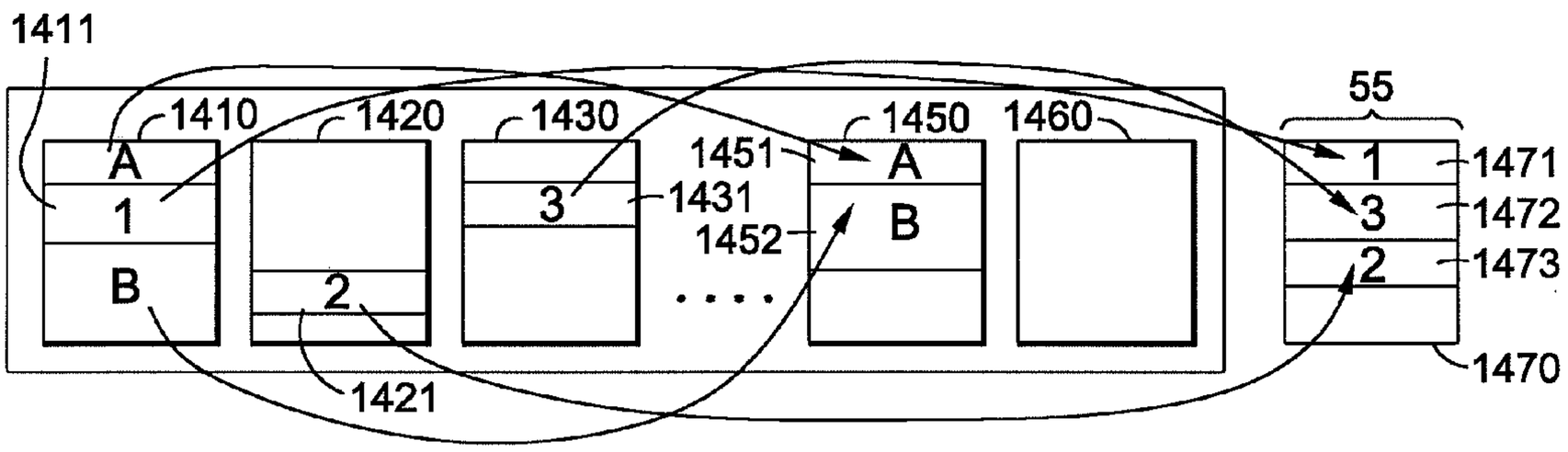



FIG. 14B

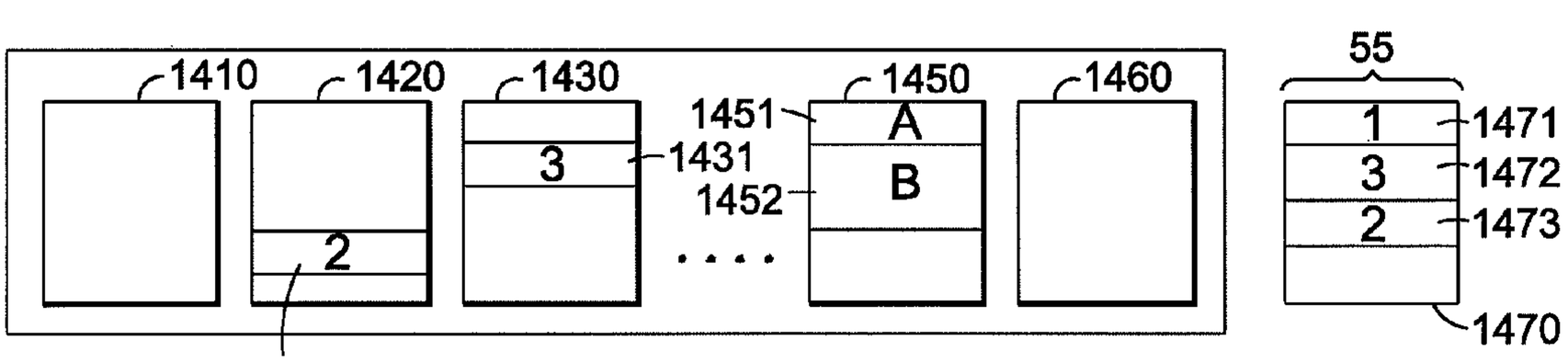



FIG. 14C

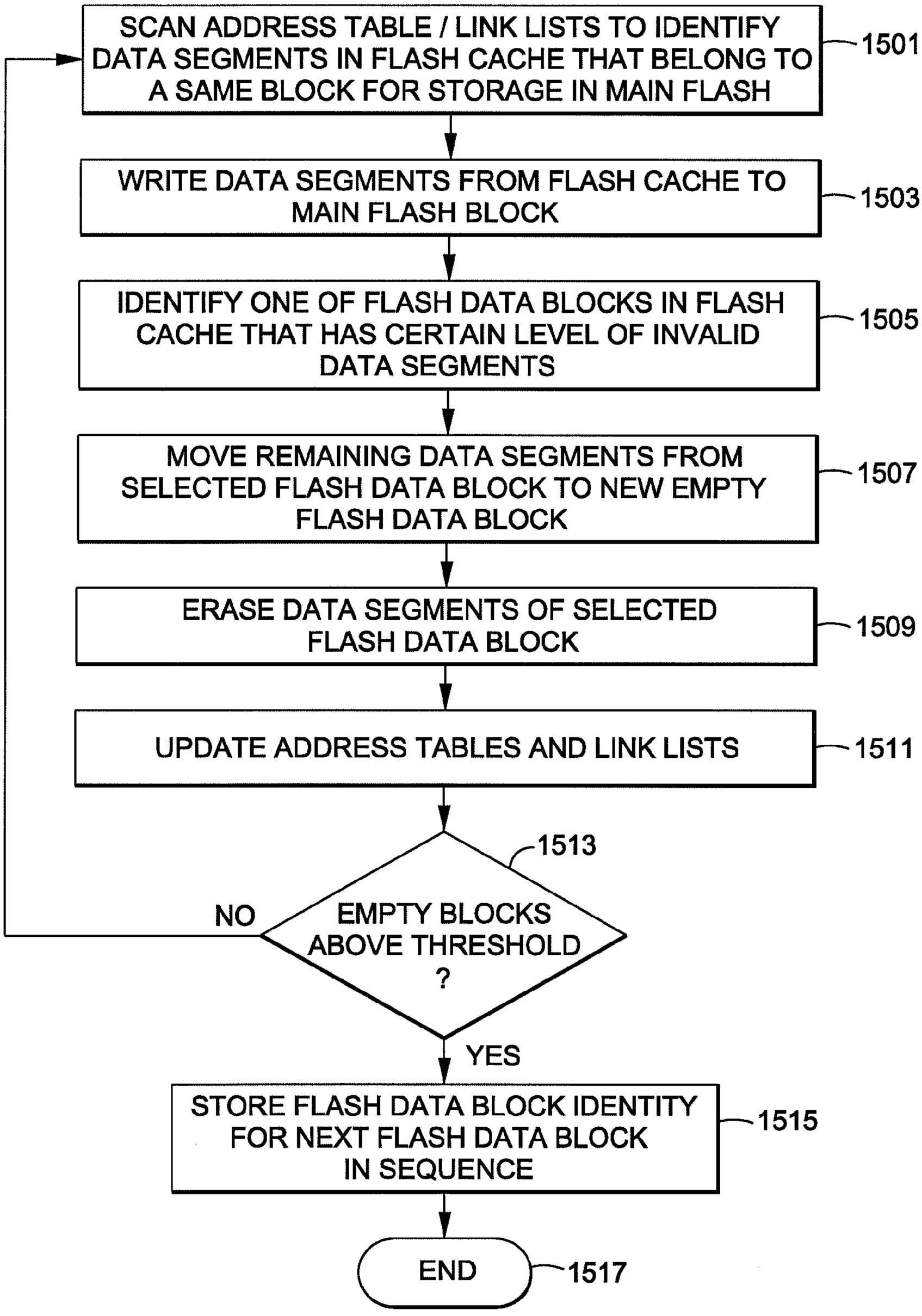

SCAN ADDRESS TABLE / LINK LISTS TO IDENTIFY DATA SEGMENTS IN FLASH CACHE THAT BELONG TO A SAME BLOCK FOR STORAGE IN MAIN FLASH
1501
WRITE DATA SEGMENTS FROM FLASH CACHE TO MAIN FLASH BLOCK
1503
IDENTIFY ONE OF FLASH DATA BLOCKS IN FLASH CACHE THAT HAS CERTAIN LEVEL OF INVALID DATA SEGMENTS
1505
MOVE REMAINING DATA SEGMENTS FROM SELECTED FLASH DATA BLOCK TO NEW EMPTY FLASH DATA BLOCK
1507
ERASE DATA SEGMENTS OF SELECTED FLASH DATA BLOCK
1509
UPDATE ADDRESS TABLES AND LINK LISTS
1511
1513
EMPTY BLOCKS ABOVE THRESHOLD ?
NO
YES
STORE FLASH DATA BLOCK IDENTITY FOR NEXT FLASH DATA BLOCK IN SEQUENCE
1515
END
1517

FIG. 15

# SOLID STATE DEVICE WITH ALLOCATED FLASH CACHE

## RELATED APPLICATION

**[0001]** The present application claims the benefit of priority under 35. U.S.C. §119 from U.S. Provisional Patent Application Ser. No. 61/075,709, entitled "SOLID STATE DEVICE," filed on Jun. 25, 2008, which is hereby incorporated by reference in its entirety for all purposes.

## FIELD OF THE INVENTION

**[0002]** The present invention generally relates to storage devices and, in particular, relates to data access in a flash storage device.

## BACKGROUND OF THE INVENTION

**[0003]** Flash memory is an improved form of Electrically-Erasable Programmable Read-Only Memory (EEPROM). Traditional EEPROM devices are only capable of erasing or writing one memory location at a time. In contrast, flash memory allows multiple memory locations to be erased or written in one programming operation. Flash memory can thus operate at higher effective speeds than traditional EEPROM.

**[0004]** Flash memory enjoys a number of advantages over other storage devices. It generally offers faster read access times and better shock resistance than a hard disk drive (HDD). Unlike dynamic random access memory (DRAM), flash memory is non-volatile, meaning that data stored in a flash storage device is not lost when power to the device is removed. For this reason, a flash memory device is frequently referred to as a flash storage device, to differentiate it from volatile forms of memory. These advantages, and others, may explain the increasing popularity of flash memory for storage applications in devices such as memory cards, USB flash drives, mobile phones, digital cameras, mass storage devices, MP3 players and the like.

**[0005]** Current flash storage devices suffer from a number of limitations. Although flash memory can be read or written at the physical page level, it can only be erased or rewritten at the block level. Beginning with a pre-erased block, data can be written to any physical page within that block. However, once data has been written to a physical page, the contents of that physical page cannot be changed until the entire block containing that physical page is erased. In other words, while flash memory can support random-access read and write operations, it can not support random-access rewrite or erase operations.

**[0006]** Generally, a flash storage device is comprised of large physical blocks that are optimized for large block sequential data transfer. Consequently, there is considerable overhead in the block carry-over and garbage collection which adversely impact write performance. As the density of a flash storage device increases, the number of blocks is increased, resulting in even more overhead and lower performance for write operations.

**[0007]** Accordingly, there is a need for improved memory controllers and memory management methods to improve the write performance of flash storage devices.

## SUMMARY OF THE INVENTION

**[0008]** A flash storage device, and methods for a flash storage device, having improved write performance are provided. Data is received from a host system, the data comprising a data segment, the data segment is temporarily stored in a data buffer of the random access memory, the data segment is assigned to a logical block address, and the data segment is written to an allocated cache portion of the flash memory. Subsequently, the data segment is written from the allocated cache portion of the flash memory to a main storage portion of the flash memory.

**[0009]** According to another aspect of the invention, data is received from a host system, the data comprising a data segment, the data segment is temporarily stored in a data buffer of the random access memory, the data segment is assigned to a logical block address, and the data segment is written to an allocated cache portion of the flash memory. Subsequently, the data segment is written from the allocated cache portion of the flash memory to a main storage portion of the flash memory. An address table is maintained in the random access memory, wherein the address table comprises a plurality of logical block addresses cross-referenced to a plurality of physical addresses, and the address table is updated upon writing the data segment from the allocated cache portion to a location in the main storage portion of the flash memory by storing the physical address of the location as a cross-reference to the logical block address assigned to the data segment.

**[0010]** According to another aspect of the invention, data is received from a host system, the data comprising a data segment, the data segment is temporarily stored in a data buffer of the random access memory, the data segment is assigned to a logical block address, and the data segment is written to an allocated cache portion of the flash memory. Subsequently, the data segment is written from the allocated cache portion of the flash memory to a main storage portion of the flash memory. It is determined whether the received data is sequential data, and the writing of the data segment to an allocated cache portion of the flash memory and the writing of the data segment from the allocated cache portion of the flash memory to the main storage portion of the flash memory are only performed if the received data is determined to not be sequential. In the case that the received data is determined to be sequential, the data segment is written directly to the main storage portion of the flash memory.

**[0011]** Data is received from a host system, the data comprising a data segment, the data segment is temporarily stored in a data buffer of the random access memory, the data segment is assigned to a logical block address, and the data segment is written to an allocated cache portion of the flash memory. Subsequently, the data segment is written from the allocated cache portion of the flash memory to a main storage portion of the flash memory. A background cleanup operation is performed in which the allocated cache portion is scanned to identify a set of data segments that collectively contain a set of sequential data, and writing the set of data segments to the main storage portion of the flash memory, upon which each one of the set of data segments is identified as in invalid data segment.

**[0012]** It is to be understood that both the foregoing summary and the following detailed description are exemplary and explanatory and are intended to provide further explanation of the invention as claimed.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0013]** The accompanying drawings, which are included to provide further understanding of the invention and are incorporated in and constitute a part of this specification, illustrate

embodiments of the invention and together with the description serve to explain the principles of the invention. In the drawings:

**[0014]** FIG. **1** depicts a block diagram of a flash storage device according to one aspect of the disclosure.

**[0015]** FIG. **2** depicts a block diagram of a flash memory according to an aspect of the disclosure.

**[0016]** FIG. **3**A illustrates an example of a data segment comprising a plurality of data sectors according to an aspect of the disclosure.

**[0017]** FIG. **3**B illustrates a process for allocating data segment in a data cache among channels according to an aspect of the disclosure.

**[0018]** FIGS. **4**A and **4**B illustrate a linked list for cache blocks according to an aspect of the disclosure.

**[0019]** FIGS. **5**A and **5**B illustrate a linked list for cache blocks according to another aspect of the disclosure.

**[0020]** FIG. **6** illustrates an example of an address table according to an aspect of the disclosure.

**[0021]** FIG. **7** illustrates an example of linked lists for keeping track of valid data segments in data blocks according to an aspect of the disclosure.

**[0022]** FIG. **8** illustrates an example of linked lists for keeping track of erase counts of data blocks according to an aspect of the disclosure.

**[0023]** FIG. **9**A is a flow chart illustrating a method of transferring data in a flash storage device according to an aspect of the disclosure.

**[0024]** FIGS. **9**B, **9**C and **9**D depict block diagrams of a flash storage device with an allocated cache portion according to embodiments of the present invention.

**[0025]** FIG. **10** depicts a block diagram of a flash storage device in which data is buffered from DRAM to an allocated cache portion according to one embodiment of the present invention.

**[0026]** FIGS. **11**A and **11**B illustrate an example of an address table for a main flash storage portion and a cross-reference table for corresponding data in an allocated cache portion, respectively, according to an embodiment of the present invention.

**[0027]** FIG. **12** depicts a flowchart of an operation for an allocated cache portion according to one embodiment of the present invention.

**[0028]** FIGS. **13**A and **13**B depict flowcharts of a power-down sequence and a power-up sequence for maintaining address tables for an allocated cache portion according to an embodiment of the present invention.

**[0029]** FIGS. **14**A, **14**B and **14**C depicts block diagrams of a flash storage device having an allocated cache portion in which a background cleanup operation is performed according to one embodiment of the present invention.

**[0030]** FIG. **15** depicts a flowchart of a background cleanup operation of an allocated cache portion in a flash storage device according to one embodiment of the present invention.

## DETAILED DESCRIPTION OF THE INVENTION

**[0031]** In the following detailed description, numerous specific details are set forth to provide a full understanding of the present invention. It will be apparent, however, to one ordinarily skilled in the art that the present invention may be practiced without some of these specific details. In other instances, well-known structures and techniques have not been shown in detail to avoid unnecessarily obscuring the present invention.

**[0032]** FIG. **1** is a block diagram of a multiple-channel flash storage device **10** according to an aspect. The multiple-channel flash storage device **10** includes an interface **15**, a controller **20** and a flash array **22**. The interface **15** interfaces the flash storage device **10** to a host system **80**, and allows the flash storage device **10** to receive data (e.g., to be written into the flash array **22**) from the host system **80** and send data to the host system **80** (e.g., data read from the flash array **22**). The controller **20** controls operations of the flash storage device **10** and manages data flow between the host system **80** and the flash array **22**, as discussed further below.

**[0033]** The controller **20** may be implemented with a general purpose processor, a digital signal processor (DSP), an application specific integrated circuit (ASIC), a field programmable gate array (FPGA) or other programmable logic device, discrete gate or transistor logic, discrete hardware components, or any combination thereof designed to perform the functions described herein. The controller **20** may also include firmware (e.g., software code) that is stored in a machine-readable medium and executed by a processor to perform the functions described herein.

**[0034]** The flash array **22** comprises a plurality of flash memories **25-1** to **25**-N split among N channels. Each flash memory **25-1** to **25**-N may comprise one or more physical flash chips, which may be implemented using NAND flash, NOR flash, or other flash technology. The flash memories **25-1** to **25**-N may comprise different numbers of flash chips. For example, within the same flash storage device **10**, some of the flash memories **25-1** to **25**-N may comprise one flash chip while others may comprise more than one flash chip.

**[0035]** The flash storage device **10** further comprises a plurality of flash memory interfaces **28-1** and **28**-N. Each flash memory interface **28-1** to **28**-N interfaces the controller **20** to one of the flash memories **25-1** to **25**-N via the corresponding channel. Each of the N channels may be implemented using one or more physical I/O buses coupled between one of the flash memory interfaces **28-1** to **28**-N and the corresponding flash memory **25-1** to **25**-N. Each of the N channels allows the corresponding flash memory interface **28-1** to **28**-N to send read, write and/or erase commands to the corresponding flash memory **25-1** to **25**-N. As discussed further below, each flash memory interface **28-1** to **28**-N may include a register (e.g., First-In-First-Out (FIFO) register) that queues read, write and/or erase commands from the controller **20** for the corresponding flash memory **25-1** to **25**-N. Although the term "channel," as used above, referred to the bus coupled between a flash memory interface **28-1** and **28**-N and the corresponding flash memory **25-1** to **25**-N, the term "channel" may also refer to the corresponding flash memory **25-1** to **25**-N that is addressable through the bus.

**[0036]** The flash memories **25-1** to **25**-N may be logically divided into data blocks. A data block may also be referred to as a memory block. Each data block may be further divided into data segments. A data segment may also be referred to as a page. For example, each data block may be 128 kilobytes (K bytes) in size, and each data segment may be 4 kilobytes (K bytes) in size. In this example, each data block has 32 4 K byte data segments. The data blocks may have other sizes besides 128 K bytes, including, but not limited, to 256 K bytes or 512 K bytes. For example, a 256 K byte data block may have 64 4 K byte data segments. The data segments may also have other sizes besides 4 K bytes. For example, the data segments may have any size that is a multiple of 1 K bytes or 2 K bytes. A data block may span one or more physical flash chips.

**[0037]** The flash storage device **10** also comprises a Dynamic Random Access Memory (DRAM) **30**. Other types of random access memory and/or volatile memory may also be used. The DRAM **30** may be used to buffer data to be written into the flash array **22** and buffer data read from the flash array **22**. The data to be written may be incoming data from a host system **80** and/or data being rewritten from one portion of the flash array **22** to another portion of the flash array **22**. The flash storage device **10** also comprises an auxiliary power device **40** for providing backup power, which is described in further detail below.

**[0038]** FIG. **2** is a block diagram of an exemplary flash memory **25** according to an aspect. The flash memory **25** includes an interface **210**, read/write/erase circuitry **215**, a buffer **220**, and a memory cell block **230**. The interface **210** interfaces the flash memory **25** to the corresponding flash memory interface **28-1** to **28**-N via the corresponding channel, and allows the read/write/erase circuitry **215** to receive read, write and/or erase commands from the corresponding flash memory interface **28-1** to **28-1** and to send data read from the memory cell block **230** to the corresponding flash memory interface **28-1** to **28**-N in response to a read command. The memory cell block **230** comprises an array of flash memory cells, where each cell may store one bit or multiple bits. The flash memory cells may be implemented using NAND flash, NOR flash, or other flash technology.

**[0039]** The read/write/erase circuitry **215** may write data to the memory cell block **230** based on a write command from the corresponding flash memory interface **28-1** to **28**-N. The write command may include an address of where the corresponding data is stored in the data cache and a physical address of where the data is to be written in the flash memory **25**. The read/write/erase circuitry **215** may also read data from the memory cell block **230** based on a read command from the corresponding flash memory interface **28-1** to **28**-N and send the read data to the corresponding flash memory interface **28-1** to **28**-N. The corresponding flash memory interface **28-1** to **28**-N may then transfer the read data to the data cache. The read command may include a physical address of where the data is to be read from the flash memory **25**. The read/write/erase circuitry **215** may also erase one or more data blocks in the memory cell block **230** based on an erase command from the corresponding flash memory interface **28-1** to **28**-N. The erase command may include a physical address of the data block to be erased.

**[0040]** The buffer **220** may be used to temporarily store data to be written to the memory cell block **230**. For example, when the corresponding flash memory interface **28-1** to **28**-N sends a write command to the flash memory **25**, the flash memory interface **28-1** to **28**-N may also transfer the data (e.g., data segment) for the write command from the data cache to the flash memory **25**. Upon receiving the data for the write command, the read/write/erase circuitry **215** may temporarily store the received data in the buffer **220** and write the data from the buffer **220** to the memory cell block **230** at the physical address specified in the write command. When all of the data in the buffer **220** for the write command has been written to the memory cell block **20**, the read/write/erase circuitry **215** may send an indication to the corresponding flash memory interface **28-1** to **28**-N that the flash memory **25** has successfully completed the write operation for the write command and is ready for another command.

**[0041]** The buffer **220** may also be used to temporarily store data read from the memory cell block **230**. For example, when the read/write/erase circuitry **215** reads data from the memory cell block **230** in response to a read command, the read/write/erase circuitry may temporarily store the read data in the buffer **220**. When all of the data requested by the read command has been read from the memory cell block **230** and stored in the buffer **220**, the read/write/erase circuitry **215** may send all of the data from the buffer **220** to the corresponding flash memory interface **28-1** to **28**-N. The flash memory interface **28-1** to **28**-N may then transfer the read data to the data cache.

**[0042]** In one aspect, the host system **80** may send data to and receive data from the flash storage device **10** in data sectors. For example, each data sector may be 512 bytes in size with eight data sectors per 4 K byte data segment. The host system **80** may use different size data sectors.

**[0043]** In one aspect, the host system **80** may address data sectors stored in the flash storage device **10** using host Logical Block Addresses (LBAs). The host LBAs allow the host system **80** to address data sectors to be written to or read from the flash storage device **10** without having to know the physical locations of the data sectors in the flash storage device **10**. The host LBAs may be implemented using an addressing scheme where data sectors are located by an index, with the host LBA of a first data sector being host LBA **0**, the host LBA of a second data sector being host LBA **1**, and so on.

**[0044]** In one aspect, the flash storage device **10** may store eight data sectors (e.g., 512 bytes) from the host system **80** into one data segment (e.g., 4 K bytes). The number of data sectors per data segment may be different depending on the size of a data sector and the size of a data segment. In this aspect, the controller **20** may address data segments in the flash storage device **10** using flash LBAs, in which 8 host LBAs corresponds to one flash LBA identifying a data segment. In this disclosure, the term "flash LBA" refers to an LBA for logically addressing a data segment and the term "host LBA" refers to an LBA for logically addressing a data sector.

**[0045]** For the example in which there are eight data sectors per data segment, the host LBAs and flash LBAs may begin at zero so that host LBAs **0-7** correspond to flash LBA **0**, host LBAs **8-15** correspond to flash LBA **1**, and so on. As a result, a sequence of eight host LBAs identifying eight data sectors corresponds to a flash LBA identifying a data segment that includes the eight data sectors. In this example, the controller **20** may determine the flash LBA corresponding to a host LBA by dividing the host LBA by the number of data sectors in a data segment (e.g., 8) and using the integer quotient for the flash LBA. For example, the data sector identified by host LBA **14** is stored in the data segment identified by flash LBA **1**.

**[0046]** FIG. **3**A illustrates an example of a data segment **310** comprising eight data sectors **312-1** to **312-8**. Each data sector **312-1** to **312-8** may be stored at a know position within the data segment **310** relative to the other data sectors **312-1** to **312-8**. When the controller **20** receives a host LBA for a data sector, the controller **20** may determine the flash LBA for the corresponding data segment, as discussed above. The controller **20** may also determine the corresponding one of the data segments **312-1** to **312-8** within the data segment **310** based on the relative position of the received host LBA within the sequence of host LBAs corresponding to the flash LBA. For example, if the host LBA is third within the sequence of host LBAs corresponding to the flash LBA, then the third data segment **312-3** within the data segment identified by the flash

LBA corresponds to the host LBA. FIG. **3**A shows an example of a data segment **310** identified by flash LBA **0** and the corresponding host LBAs **1-7** for the data sectors **312-1** to **312-8**. The data segment **310** may include a sector index indicating which of the data sectors **312-1** to **312-8** in the data segment **310** are valid. For example, if the host system **80** sends a write command for only host LBAs **0-3**, then the data segment for flash LBA **0** would hold four valid data sectors.

**[0047]** In one aspect, the host system **80** may address data sectors to be written to and/or read from the flash storage device **10** using host LBAs. For example, when writing data to the flash storage device **10**, the host system **80** may send one or more data sectors with one or more corresponding host LBAs. Later, the host system **80** may randomly read any one of the data sectors from the flash storage device **10** using the corresponding host LBAs in a host read command. The controller **20** of the flash storage device **10** may write and/or read requested data sectors to and/or from the flash array **22** using the corresponding flash LBAs, which may be determined as discussed above.

**[0048]** In an aspect, the flash LBAs in the flash storage device **10** are split evenly among the N channels so that 1/N of the flash LBAs are assigned to the first channel, and so forth. Evenly splitting the flash LBAs among the channels increases the likelihood that, on a read corresponding to more than one flash LBA, not all of the data comes from one channel.

**[0049]** The flash LBAs may be evenly split among the N channels using any one of a number of methods. For example, the controller **20** may use an algorithm to determine which channel to assign each flash LBA. The algorithm may be a MOD function that divides a flash LBA by the number of channels N and outputs the remainder of the division, where the remainder represents the channel assigned to the flash LBA. For example, in a 16 channel device, flash LBA **12101** would be assigned to channel **5** (i.e., **12101** MOD **16**=**5**). This algorithm provides an even distribution of the flash LBAs among channels. The above algorithm provides just one example of evenly splitting the flash LBAs among the channels. Other algorithms may be used to evenly split flash LBAs among the channels. An example in which flash LBAs are evenly split among channels is discussed below.

**[0050]** An address table mapping all flash LBAs to physical addresses in the flash array **22** is kept in the DRAM **30**. The controller **20** in the flash storage device **10** stores data segments into physical addresses in the flash array **20** and maps each flash LBA to the physical address where the corresponding data segment is stored in the address table. When the host system **80** sends a host read command to the flash storage device **10** with one or more host LBAs, the controller **20** determines the corresponding LBAs, as discussed above. The controller **20** then maps the corresponding flash LBAs to physical addresses in the flash array **22** using the address table and reads the requested data from the physical addresses in the flash array **22**. A flash LBA may be 28bits in length or have another length depending on the amount of logically addressable memory in the flash array **22**. An example of an address table is given below.

**[0051]** When the controller **20** writes a data segment to a physical address in the flash array **22**, the controller **20** may store the corresponding flash LBA in the physical address with the data segment, along with other information discussed below. This allows the controller **20** to reconstruct the address table by reading the flash LBAs stored in the physical addresses of the flash array **22** in the event that the address table is lost.

**[0052]** The flash storage device **10** has power back-up to make sure that certain functions are performed in the event of a power loss. One of these functions may be to store the address table in non-volatile memory to preserve the table, e.g., by writing the address table from the DRAM **30** to flash array **22** when the power goes off (either expectedly or unexpectedly). This enables the flash storage device **10** to start up or initialize faster because the address table can be read from flash (which may only take two seconds) instead of having to read all of the data in flash memory to reconstruct the address table. In other words, the address table does not have to be reconstructed since the address table is stored in the flash array **22** by the back-up power.

**[0053]** In one aspect, the auxiliary power device **40** comprises super capacitors that charge when the flash storage device **10** receives power, and provides auxiliary power when the main power goes off by using energy stored in the super capacitors. This allows data (e.g., address table) to be written into flash array **22** when the power goes off unexpectedly. The controller **20** may detect a power failure using a voltage threshold detector that detects when the voltage of a power supply falls below a threshold voltage. Upon detecting a power failure, the controller **20** may save all critical data (e.g., address table) to the flash array **22** using the auxiliary power provided by the auxiliary power device **40**. Besides triggering this process upon detection of a power failure, the controller **20** may trigger this process on a sync cache command, flush cache command, power down immediate command, standby command, etc. The commands may come from the host system **80** and/or the controller **20**. The auxiliary power device **40** may comprise one or capacitors, one or more batteries, or other forms of energy storage devices.

**[0054]** Data received by the flash storage device **10** is stored in the DRAM **30** and regardless of whether the data is received sequentially or randomly, the data is written from the DRAM **30** to the flash array **22** in a sequential manner on a per channel basis, as discussed further below. This makes data reads from the flash array **22** random. Since flash memory typically has good sequential write performance and random read performance, this arrangement takes advantage of the sequential write performance and random read performance of the flash memories **25-1** to **25**-N.

**[0055]** In one aspect, data is buffered in a portion of the DRAM **30** referred to as a data cache. The data cache may occupy all remaining DRAM **30** after everything else has been allocated in the DRAM **30**, for example, the address table and the linked lists discussed below. Therefore, the size of the data cache varies depending on the device size and configuration. In one aspect, the flash storage device **10** does not start up unless there is a minimum amount of memory (e.g., at least 16 MB of DRAM) available for the data cache. The data cache may be split up into a plurality (e.g., thousands) of small cache blocks which are each the size of one data segment (e.g., 4K bytes).

**[0056]** All read and write data is transmitted between the host system **80** and the flash array **22** via the data cache in the DRAM **30**. When the host system **80** writes data, the data is written into the data cache. When the data cache receives enough data to fill a complete data block, the data is read from the data cache and written to the flash array **22**. When the host system **80** reads data, the data is read from the flash array **22**,

written into the data cache and then read from the data cache and transmitted to the host system **80** over the host interface. A central processing unit (CPU) of the host system **80** does not have to copy the data to/from the data cache or receive the data. The CPU may instruct hardware where to send the data or where to retrieve the data from and the hardware does the rest (e.g., via direct memory access (DMA)).

**[0057]** FIG. **3**B is a diagram illustrating a process for allocating data segments **310** in the data cache of the DRAM **30** among the different channels. In the example in FIG. **3**B, there are 16 channels, although it is to be understood that other numbers of channels may be used (e.g., 8 channels). For example, the number of channels may be any power of two, which allows easier implementation of the controller **20** using firmware. The data segments **310** may be data received from the host system **80** in one or more host write commands. In this example, the data segments **310** are given corresponding flash LBAs by the controller **20**. As discussed above, for the example in which there are eight data sectors per data segment, each flash LBA may correspond to eight host LBAs. For example, flash LBA **0** may correspond to host LBAs **0-7**, flash LBA **1** may correspond to host LBAs **8-15**, and so forth.

**[0058]** As shown in FIG. **3**B, the flash LBAs are split evenly among the 16 channels and striped across the 16 channels. In the example in FIG. **3**A, flash LBA **0** is assigned to channel **1**, flash LBA **1** is assigned to channel **2**, flash LBA **2** is assigned to channel **3**, and so on. As data is received from the host system **80** during a host write and stored in the data cache, the number of data segments **310** allocated to each channel increases. In the example in FIG. **3**B, the flash LBAs are sequential across the channels. However, this does not have to be the case. For example, the flash LBAs may comprise two or more random LBAs when the host system **80** writes data randomly to the flash storage device **10** (random host LBAs) instead of sequentially to the flash storage device **10** (sequential host LBAs). In either case, the controller **20** can evenly split the corresponding flash LBAs among the N channels, for example, using the algorithm described above or other method.

**[0059]** When the number of data segments **310** for a channel in the data cache reaches enough to fill a data block (e.g., 32 data segments), the controller **20** writes the data segments **310** from the data cache to the corresponding flash memory **25-1** to **25**-N via the corresponding channel. The controller **20** may write the data segments **310** for a channel to the corresponding flash memory **25-1** to **25**-N by queuing write commands for the data segments **310** in the register (e.g., FIFO register) of the corresponding flash memory interface **28-1** to **28**-N (e.g., one write command for each data segment **310**). The flash memory interface **28-1** to **28**-N may then send the queued write commands to the corresponding flash memory **25-1** to **25**-N via the corresponding channel. For example, the flash memory interface **28-1** to **28**-N may sequentially send the queued write commands to the corresponding flash memory **25-1** to **25**-N via the corresponding channel. Each time a write command is sent to the corresponding flash memory **25-1** to **25**-N, the flash memory interface **28-1** to **28**-N may transfer the data segment for the write command from the data cache to the corresponding flash memory **25-1** to **25**-N via the corresponding channel. In this example, the flash memory interface **28-1** to **28**-N may send one write command to the corresponding flash memory **25-1** and **25**-N and wait until it receives an indication from the flash memory **25-1** to **25**-N that the data segment has been successfully written before sending the next write command in the register to the channel.

**[0060]** The flash memory interfaces **28-1** to **28**-N may send write commands to their respective flash memories **25-1** to **25**-N substantially in parallel via the respective channels. In addition, during the time a flash memory interface **28-1** to **28**-N waits for the corresponding flash memory **25-1** to **25**-N to complete a write command after transferring data for the write command from the data cache to the corresponding flash memory **25-1** to **250**-N, other flash memory interfaces **28-1** to **28**-N may transfer data for other write commands from the data cache to corresponding flash memories **25-1** to **25**-N. The parallel write operations among the plurality of flash memories **25-1** to **25**-N results in faster write times and maximizes the write performance of the flash array **22**.

**[0061]** When the controller **20** receives a host read command from the host system **80** including host LBAs, the controller **20** determines the corresponding flash LBAs, as discussed above. The controller **20** then maps the flash LBAs to the corresponding physically addresses in the flash array **22** using the address table. The controller **20** then generates read commands corresponding to the physically addresses and queues the read commands in the registers of the respective flash memory interfaces **28-1** to **28**-N. Each flash memory interface **28-1** to **28**-N sends the corresponding read commands to the corresponding flash memory **25-1** to **25**-N via the corresponding channel and writes the resulting read data to the data cache. When all of the requested data is written to the data cache in the DRAM **30**, the controller **20** may transfer the requested data from the data cache to the host system **80** via the interface **15**. Splitting flash LBAs evenly among the channels helps ensure that, for a host read spanning several flash LBAs, the requested data is read from a plurality of the flash memories **25-1** to **25**-N.

**[0062]** In one aspect, the data cache in the DRAM **30** may be divided into cache blocks. Each cache block may correspond to a physical address in the DRAM **30** for storing one data segment (e.g., 4 K bytes).

**[0063]** In one aspect, the cache blocks are all held in a linked list. When a new cache block is required for a new data segment, the data segment in the cache block that is on the top of the linked list is removed from the data cache (assuming it is not active) and the cache block is allocated for the new data segment. The cache block with the new data segment is then placed at the end of the linked list and the new data segment is stored in a physical address of the data cache corresponding to the cache block. The new data segment may come from data in a host write command or data read from the flash array **22** in response to a host read command. If the data in the cache block is not accessed again, then the cache block will slowly move up the linked list as cache blocks are allocated for new data segments and moved to the end of the linked list.

**[0064]** FIGS. **4**A and **4**B illustrate an example of a linked list **405** according to an aspect. FIG. **4**A shows the linked list **405** in which cache block **410** is on the top of the linked list **405** and cache block **430** is on the end of the linked list **405**. In this example, when a new data segment needs to be stored in the data cache, the controller **20** removes the data segment corresponding to cache block **410** from the data cache, allocates cache block **410** to the new data segment and moves the cache block **410** to the end of the linked list **405**. FIG. **4**B shows the linked list **405** after cache block **410** has been moved to the end of the linked list **405**. As a result, the other

cache blocks in the linked list **405** have been moved up one position in the linked list **405**. For example, cache block **430** has been moved up one position from the end of the linked list **405**, as shown in FIG. **4**B.

**[0065]** In the above aspect, the controller **20** may continue to store a data segment in the data cache after the data segment has been written to the flash array **22**. The controller **20** removes the data segment from the data cache when the corresponding cache block reaches the top of the linked list and the controller **20** needs to allocate the cache block for a new data segment.

**[0066]** If a host read command is received from the host system **80** and the data for the read command is found in the data cache, then the flash array **22** does not need to be accessed. In this case, the controller **20** may retrieve the data from the data cache and the corresponding cache block is moved from the current position on the linked list to the end of the linked list. This way frequently accessed data will be retained in the data cache because a cache block that is being frequently accessed will be continually moved to the end of the linked list and not reach to the top of the linked list from where the data in the cache block will be removed from the data cache.

**[0067]** FIGS. **5**A and **5**B illustrate an example of the linked list **405** according to an aspect. FIG. **5**A shows the linked list **405** in which cache block **410** is on the top of the linked list **405**, cache block **430** is on the end of the linked list **405**, and cache block **450** is at a position somewhere between cache blocks **410** and **430**. In this example, when a host read command requests a data segment in the data cache corresponding to cache block **450**, the controller **20** retrieves the data segment from the data cache and moves cache block **450** to the end of the linked list **405**. FIG. **5**B shows the linked list **405** after cache block **450** has been moved to the end of the linked list **405**. As a result, the other cache blocks that where previously below cache block **450** in the linked **405** have been moved up one position. For example, cache block **430** has been moved up one position from the end of the linked list **405**, as shown in FIG. **5**B. If the data segment corresponding to cache block **450** is frequently read, then cache block **450** will continually move to the end of the linked list **405**.

**[0068]** In one aspect, a cache block structure is maintained for each cache block. The cache block structure includes a physical address of the cache block in the DRAM **30** where the corresponding data segment is stored and other information (e.g., the time that the data segment was written into the data cache). The cache block structure may also include a physical address in the flash memory **22** where the corresponding data segment is stored if the data segment has already been written to the flash memory **22**.

**[0069]** All data to be written to the flash array **22** is buffered in the DRAM **30** for a predetermined amount of time (e.g., 10 seconds) to allow the next sequential data (e.g., data sectors) in a data sequence to arrive and fill the data cache of the DRAM **30**. This allows the controller **20** to write a large amount of data from the DRAM **30** to the flash array **22** at one time and free up the controller **20** to perform other operations. If the next sequential data does not arrive in the predetermined amount of time (e.g., 10 seconds), then the data in the data cache of the DRAM **30** is written to the flash array **22** and may be combined with other data that needs to be written. A complete data block may not be written to a channel of the flash memory array **22** when power is turned off and there is not a complete data block in the DRAM **30** to write to the channel. In this case, the controller **20** may write an incomplete data block to the channel of the flash array **22**. Data in the DRAM **30** can also be written to the flash array **22** upon receiving a command to flush the data cache, clear the data cache, or clear the cache.

**[0070]** In one aspect, the controller **20** uses an address table to locate a data segment for a given flash LBA in either the flash array **22** or the data cache of the DRAM **30**. The address table may also be referred to as a virtual-to-physical (V2P) table since the address table maps a logical address to a physical address.

**[0071]** FIG. **6** shows an example of an address table **605** according to an aspect of the disclosure. Each of the entries **610-1** to **610**-n in the address table **605** corresponds to a flash LBA and may be arranged in order of the flash LBAs so that flash LBA **0** corresponds to the first entry, flash LBA **1** to the second entry, and so on. The entry **610-1** to **610**-*n* for each flash LBA **615** may include a channel number **620**, a cache indicator **625** indicating whether the corresponding data segment is in the data cache, and depending on whether the data segment is in the data cache, a physical address of the data in the flash array **22** or a cache block number **630**.

**[0072]** If the cache indicator **625** indicates that the data segment for a flash LBA **615** is not in the data cache, then the entry includes the physical address **630** of the data segment in the flash array **22**. In this case, the controller **20** uses the physical address **630** to locate the data segment in the flash array **22**. The physical address **630** can be in a hardware format so that the physical address can be passed directly to the hardware side of the flash storage device **10** to retrieve the data segment from the flash array **22**.

**[0073]** If the cache indicator **625** indicates that the data for a flash LBA **615** is in the data cache, then the entry for the flash LBA **615** includes a cache block number **630**. The cache block number **630** points to the cache block corresponding to the flash LBA. This allows the controller **20** to retrieve the cache structure of the corresponding cache block. The cache structure in turn provides the controller **20** with the physical address where the data is located in the DRAM **30** and the NAND device information if required plus other useful information (e.g., the time that the corresponding data segment was written into the data cache). The NAND device information may include a physical address of the corresponding data segment in the flash array **22**.

**[0074]** In one aspect, the controller **20** updates the address table **605** when data sectors corresponding to a data segment are received from the host system **80** in a host write command or read from the flash array **22** in response to a host read command. The controller **20** allocates a cache block from the top of the linked list for the corresponding data segment. The controller **20** then creates an entry in the address table **605** for the data segment including the flash LBA **615** of the data segment and a cache block number **630** of the cache block allocated to the data segment. The controller **20** also sets the cache indicator **625** to indicate that the data segment is located in the data cache. If the data segment is eventually written to the flash array **22**, then the controller **20** may include the physically address of the data segment in the flash array **22** in the corresponding cache block. When the data segment is removed from the data cache, then the controller **20** may update the entry for the data segment accordingly. The controller **20** may set the cache indicator **625** to indicate that the data segment is not stored in the data cache and include the

physical address of the data segment in the flash array **22**. The controller **20** may retrieve the physical address from the cache structure in the cache block.

**[0075]** In one aspect, each data block in the flash array **22** can hold random data segments instead of consecutive data segments. Thus, when random write data is received for a flash LBA (e.g., in a random host write command), it is not necessary for the controller **20** to re-write a complete data block for the corresponding data segment. For example, when the controller **20** receives write data (updated data) for a flash LBA corresponding to a data segment in a data block (old data block) in the flash array **22**, the controller **20** writes the write data (updated data) to another data block (new data block) in the flash array **22** instead of re-writing the old data block with the write data (updated data) for the flash LBA. In this case, the data segment corresponding to the flash LBA in the old data block is invalid since the write data (updated data) for the flash LBA is stored in the new data block. This reduces wear on the old data block because the old data block does not have to be erased and rewritten when one of its data segments is subsequently updated by the host system **80**. Thus, a data block can hold a varying number of valid data segments at a given time, depending on how many data segments in the data block are updated and written to other data blocks.

**[0076]** In one aspect, the controller **20** maintains a table for each data block in the flash array **22**. The table for each data block identifies the number of valid data segments in the data block and the flash LBAs corresponding to valid data segments held in the data block. In an aspect, the controller **20** uses this information to identify data blocks with the fewest valid data segments. After identifying the data blocks with the fewest valid data segments, the controller **20** copies the valid data segments of the identified data blocks to one or more other data blocks in the flash array **22**, and erases the identified data blocks to provide pre-erased data blocks for more write data. This allows the maximum number of data blocks to be freed up for write data while having to copy the least amount of data to other data blocks. As a result, fewer write operations are required to free up data blocks, providing more bandwidth for host read and write data.

**[0077]** In one aspect, each data block is kept in one of a plurality of linked lists depending on the number of valid data segments in the data block. All data blocks with only one valid data segment each are kept in one list, all data blocks with only two valid data segments are kept in another list, and so on. As discussed further below, the controller **20** performs dynamic wear leveling (to free up data blocks) using the lists of valid data segments to identify data blocks having only one valid data segment, data blocks having only two valid data segments, and so on. There are also separate lists for bad (e.g., defective) data blocks, empty data blocks, free data blocks and data blocks (e.g., data blocks being written to), etc. Each data blocks is also kept in one of a plurality of linked lists depending on the number of erase cycles performed on the data block. As discussed further below, the controller **20** performs static wear leveling using the linked lists of erase cycles.

**[0078]** In one aspect, the controller **20** places a time stamp on each data segment that is written into the flash array **22** allowing the controller **20** to know which data segments are newer than the old ones. The time stamps may be generated by incrementing a counter when a write operation is performed and using the current count value of the counter for the time stamp for data being written during the write operation. For example, the time stamp for a write may be 11000, then 11001 for the next write, and so on. Thus, the time stamps allow the controller **20** to determine the order in which data segments were written into the flash array **22** by comparing their time stamps. In this example, a data segment with a time stamp having a higher count value is newer than a data segment with a time stamp having a lower count value. The time stamp for a data segment may be stored with the data segment in the flash array **20** and/or stored in a table.

**[0079]** In an aspect, the controller **20** compares the time stamps of data segments in the background and not during regular read/write operations. When two data segments in different data blocks correspond to the same flash LBA, then the older data segment (as indicated by the corresponding time stamp) is invalid. Eventually, a data block with many invalid data segments is erased and one or more valid data segments that were in the data block are combined with other incoming data or valid data from other blocks that are being erased. The combined data is written to a new data block, freeing up the erased block for write operations. When the one or more valid data segments are written to the new data block, the address table is updated with the new physical addresses of the valid data segments. In an aspect, the above operations are performed in the background to make pre-erased blocks available for write operations. Since this is a background operation, the flash storage device **10** may need to allocate more flash memory than is logically addressable by the controller **20**. For example, a flash storage device **10** with 256 GB of memory may only have 146 GB of addressable memory (e.g., addressable to the host system **80**). This helps provide pre-erased blocks to write to most of the time.

**[0080]** Preferably, the time stamps are used if reconstruction of the address table is required. When there are two or more physical addresses for the same flash LBA, the time stamps of the corresponding data segments may be used to determine which data segment is the most recent (e.g., data segment with the time stamp having the highest count value is the most recent). In this example, the physical address with the more recent time stamp is used to reconstruct the address table entry for the flash LBA.

**[0081]** FIG. **7** illustrates linked lists that are used to determine which data block **710** to remove data from and erase to provide pre-erased blocks for high-speed writing according to one aspect. As an example, each data block **710** may be 128 K bytes in size and hold 32 4 K bytes data segments. Normally, a data block **710** will be full when it is first written to and will therefore have 32 valid data segments in this example. The linked lists may cover data blocks in the entire flash array **22**. Alternatively, the linked lists may be implemented for each channel of the flash array **22**.

**[0082]** The link lists are used to keep track of the number of valid data segments in each data block **710**. In one aspect, each data block **710** is in one of the linked lists of valid data segments. In the example above, there are 33 linked lists since each data block has zero to 32 valid data blocks. List **0** comprises data blocks **710** with no data segment holding valid data. List **1** comprises data blocks **710** with only one data segment holding valid data, list **2** comprises data blocks **710** with two data segments holding valid data, and so forth. List **32** comprises data blocks **710** with all 32 data segments holding valid data. The number of linked lists may be different from 32 depending on the sizes of the data segments and/or data blocks. For example, the number of linked lists may be lower if larger data segments are used (fewer data

segments in each data block **710**). As another example, the number of linked lists may be larger if larger data blocks **710** are used (more data segments per data block). The flash storage device **10** may include lists of pre-erased data blocks **710**, bad data blocks **710**, system blocks, etc. The data blocks **710** in the lists may be identified by addresses of or pointers to the data blocks **710** therein or other means.

**[0083]** By way of example, when a data block XX is initially written to, the data block XX will be full and therefore go to the list with all valid data segments (list **32** in the example illustrated in FIG. **7**). If one of the data segments in data block XX is subsequently written again, then the data segment will go into a different data block YY. For example, if the flash storage device **10** receives a host write command from the host system **80** that results in a write to flash LBA **12100** and a data segment for flash LBA **12100** is already stored in data block XX, then the controller **20** writes the data in the write command for flash LBA **12100** to data block YY. Thus, the data segment for flash LBA **12100** is rewritten to data block YY, invalidating the data segment for flash LBA **12100** in data block XX. As a result, the data block XX will have only 31 valid data segments, and the controller **20** will move the data block XX from list **32** to list **31** in the example illustrated in FIG. **7**. Also, the address table for flash LBA **12100** is updated with the physical address of the data segment in data block YY. Each time another data segment in data block XX becomes invalid, data block XX is moved down to the appropriate list.

**[0084]** Pre-erased blocks that are available for write operations are identified in a pre-erased list. When the number of pre-erased data blocks in the pre-erased list gets low (e.g., below a predetermined threshold), the controller **20** may need to erase some of the data blocks **710** to increase the number of pre-erased data blocks available for write operations. In one aspect, the controller **20** determines which data blocks **710** to erase by looking in the lists of valid data segments in ascending order. The controller **20** first looks in list **0** for data blocks **710** holding no valid data. Since data blocks in list **0** require no movement of data to erase, these data blocks **710** are erased and placed into the pre-erased list first. If list **0** is empty and/or additional pre-erased blocks are needed, then the controller looks in list **1** for data blocks **710** with only one valid data segment. Each data block **710** in list **1** only require the movement of one data segment to be erased. If list **1** is empty and/or additional pre-erased blocks are needed, then the controller looks in list **2**, and so forth. As data blocks **710** are erased, they are added to the end of the pre-erased list. When a pre-erased block is needed, it is taken from the top of the pre-erased list. This provides a form of dynamic wear leveling. The earlier pre-erased blocks are on the top of the list.

**[0085]** When multiple data blocks **710** with one valid data segment are erased to produce pre-erased blocks, the valid data segments of these erased data blocks **710** may be combined and rewritten into a new single data block. For example, if there are 32 data blocks **710** with one valid segment each, then the 32 valid data segments of these data blocks **710** may be written into a new block **710**, and all 32 data blocks **710** can be erased or moved to list **0**. The new block **710** will have all valid data segments. The physical addresses of the 32 valid segments are updated in the address table accordingly.

**[0086]** FIG. **8** illustrates linked lists that are used to provide static wear leveling according to one aspect. The lists keep track of the erase counts of the data blocks **710**. The controller **20** uses the lists to provide wear leveling by keeping the erase counts of the data blocks **710** within a certain range (e.g., 8192 erase counts) of each other. In addition to belonging to one of the lists of valid data segments discussed above, the data blocks **710** (with the exception of BAD or System data blocks) are in one of a plurality of static wear lists. The example in FIG. **8** has eight static wear lists. Each list comprises data blocks **710** that are within a certain range of the lowest erase count. The linked lists may cover data blocks in the entire flash array **22**. Alternatively, the linked lists may be implemented for each channel of the flash array **22**.

**[0087]** In the example illustrated in FIG. **8**, list **0** comprises data blocks **710** with erase counts that are within 1023 of the lowest count. List **1** comprises data blocks **710** with erase counts that are within a range of 1024-2047 of the lowest count, and so forth up to list **7**, which comprises data blocks **710** with erase counts that are within a range of 7168-8191 of the lowest count. Although the example in FIG. **8** has eight lists, any number of lists may be used with any count range.

**[0088]** Each time a data block **710** is erased, it is moved to the end of the list it is in. In practice, this means that each list tends to be in roughly an ascending numerical sequence. This may be especially true at the start of the list where data blocks with the lowest counts are located.

**[0089]** When a data block **710** reaches list **7** (highest list), the block is deemed as hot and in need of static wear leveling. The data block **710** in list **7** will have been pre-erased at this point (since the count of a data block is not bumped up until the data block is erased). Data is copied from the data block at the head of list **0** (which should hold static data because of its low erase count) to the pre-erased data block **710** with the high erase count in list **7**. The low count data block **710** from which the data was moved is erased and put into the free pool. When list **0** becomes empty, the lists are all moved down by one (which involves moving head and tail pointers) and the lowest count is bumped up by 1024. In this example, the data blocks **710** in list **1** are moved down to list **0**, the data blocks **710** in list **2** are moved down to list **1**, and so forth. After the bump up, the lowest count is 1024 higher and list **7** is emptied since the data blocks previously in list **7** have been moved down to list **6**.

**[0090]** The controller **20** may use both the lists of valid data segments and the lists of erase counts to determine which data segments need to be moved and which data blocks **710** need to be erased.

**[0091]** The lists described above may be stored in the DRAM **30** for fast access by the controller **20** during operation of the flash storage device **10**, and stored in the flash array **22** when the flash storage device **10** is turned off.

**[0092]** FIG. **9**A is a flow chart illustrating a method of transferring data in a flash storage device comprising a random access memory and a plurality of channels of a flash array according to an aspect of the disclosure. The method begins with step **901**, in which the flash storage device receives a plurality of data segments from a host system. Each data segment may comprise data sectors. In step **902**, the plurality of data segments are stored in the random access memory (e.g., DRAM **30**). In step **903**, the plurality of data segments are allocated among the plurality of channels of the flash array. In step **904**, the allocated data segments are written from the random access memory to the respective channels of the flash array.

**[0093]** In another aspect of the invention, a portion of the flash memory **25-1** to **25**-N may be allocated to be used as an allocated flash cache portion which can be used as a cache for

buffering data to and from the remaining portion of flash memory **25-1** to **25**-N. In this manner, the I/O performance of buffering data to flash memory **25-1** to **25**-N is increased in comparison to just using DRAM **30** as a data buffer in flash storage device **10**.

**[0094]** In this regard, FIG. **9**B depicts a flash storage device with an allocated flash cache portion according to one embodiment. As seen in FIG. **9**B, flash storage device **10** allocates a portion of flash array **22** as flash cache **50** and the remainder of flash array **22** is allocated as main flash storage **55**. As described above with regard to FIG. **1**, flash array **22** can comprise more than one flash chip, such as flash memory **25-1** to **25**-N, in which case the allocation of flash cache **50** and main flash storage **55** can be distributed among flash memory **25-1** to **25**-N as shown in FIG. **9**C. In FIG. **9**C, it can be seen that the same allocation of flash cache **50** and main flash storage **55** is provided in each of flash memory **25-1** to **25**-N. Of course, it should be appreciated that different allocations of flash cache **50** and main flash storage **55** can be provided in each of flash memory **25-1** to **25**-N. In an alterative aspect, the allocation of flash cache **50** and main flash storage **55** can be distributed among flash memory **25-1** to **25**-N such that all of flash cache **50** is provided in only certain ones of flash memory **25-1** to **25**-N and main flash storage **55** is provided in the remainder of flash memory **25-1** to **25**-N, as shown in FIG. **9**D. In FIG. **9**D, flash memory **25-1** is shown as being entirely utilized for flash cache **50** while the remainder flash memory **25-2** to **25**-N is used for main flash storage **55**. It should be appreciated that any combination and variation of allocation between flash cache **50** and main flash storage **55** across multiple flash memory **25-1** to **25**-N can be implemented according to the invention. In one aspect of the invention, the apportionment of flash memory **25-1** to **25**-N into flash cache **50** and main flash storage **55** can be programmable according to an algorithm or through user selection.

**[0095]** In one embodiment, the portion of flash memory **25-1** to **25**-N that is allocated to flash cache **50** is proportional to the total size of flash memory **25-1** to **25**-N. For example, the size of flash cache **50** could be 3 GB for flash memory **25-1** to **25**-N having a total size of 64 GB, and could be 6 GB for flash memory **25-1** to **25**-N having a total size of 128 GB.

**[0096]** In one aspect, flash cache **50** is partitioned in a similar manner to DRAM cache **30** described above. For example, flash cache **50** can be partitioned into flash data blocks each of which contain multiple data segments. For example, a flash data block of flash cache **50** can contain 128 data segments, each data segment being 4 k in size, but could be other sizes such as 2 k. To obtain increased performance of flash storage device **10** in conjunction with flash cache **50**, the main flash storage **55** can be partitioned into large flash data blocks also, such as logical blocks of 256 k, 512 k or 1M size. The benefit of these large blocks is discussed further below with regard to the efficient buffering of sequential data directly to the main flash storage **55**.

**[0097]** FIG. **10** depicts a block diagram of a flash storage device in which data is buffered from DRAM to an allocated cache portion according to one embodiment of the invention. In FIG. **10**, DRAM **30** is shown which acts as a buffer for incoming data from controller **20** (not shown), and flash memory **25-1** to **25**-N is also shown. DRAM **30** shows a cache data block **1010** of a total data cache of DRAM **30** and also includes address table array **1020**. Cache data block **1010** shown in FIG. **10** is of the same size as a data segment to facilitate transfer of data into flash memory **25-1** to **25**-N, but can in the alternative be another size. For example, the size of cache data block **1010** shown is 4 k, which is the size of a data segment.

**[0098]** Address table array **1020** is comprised of a series of separate address tables for correlating LBAs to physical addresses of flash memory **25-1** to **25**-N where the corresponding data for each LBA is stored. In one example, each address table of address table array **1020** corresponds to a set of cache data blocks (data segments) that comprise enough data to fill a flash data block and which is therefore transferred from the data cache of DRAM **30** into flash memory **25-1** to **25**-N (either in flash cache **50** or in main flash storage **55**). In this regard, flash cache **50** is shown in FIG. **10** to include flash data blocks **1030**, **1040** and **1050**, each of which has a corresponding flash cache address table, **1061**, **1062** and **1063**, respectively. Similarly, main flash storage **55** is shown in FIG. **10** to have flash data blocks **1070**, **1080** and **1090**. Of course, flash cache **50** and main flash storage **55** may have other numbers and sizes of flash data blocks.

**[0099]** As discussed above, host LBAs are mapped to LBAs by controller **20** and the LBAs are distributed among the channels of flash memory **25-1** to **25**-N according to an algorithm. On this note, also as discussed above in one aspect, each LBA is stored along with the corresponding data in flash memory **25-1** to **25**-N so that one or more address tables of address table array **1020** can be recreated if necessary by scanning the stored data segments in flash memory **25-1** to **25**-N.

**[0100]** Address table array **1020** shown in FIG. **10** is includes one or more address tables that correspond to physical addresses of data stored in main flash storage **55** and also includes one or more address tables that correspond to physical addresses of data stored in flash cache **50**. In the embodiment shown in FIG. **10**, incoming data is buffered through the data cache of DRAM **30** and sent to flash cache **50** which organizes the data into flash data blocks and then sends the data to a flash data block in main flash storage **55**. In an alternative, a flash data block of data can be sent directly from the data cache of DRAM **30** to main flash storage **55**, depending on the type of data as discussed further below. So, some address tables of address table array **1020** relate to data residing in flash cache **50** and some address tables relate to data that is already stored in main flash storage **55**. In FIG. **10**, address table **1021** of address table array **1020** is an address table that corresponds to data stored in flash cache **50**, and in particular to data stored in flash data block **1030** of flash cache **50**.

**[0101]** As seen in FIG. **10**, when enough incoming data is provided to fill a cache data block (segment) of DRAM **30**, such as cache data block **1010**, the cache data block is transferred into an active flash data block of flash cache **50**, such as flash data block **1030** shown in FIG. **10**. In this example, cache data block **1010** is shown to be filled and transferred into the first data segment **1031** of flash data block **1030**. When an entire flash data block is written from the data cache of DRAM **30** to flash cache **50**, the corresponding address table **1021** is updated to reference the physical addresses of where the data segments (cache data blocks) are stored in the flash data block in correlation with the LBAs that correspond to each data segment. Accordingly, the "virtual-to-physical" address table for data which is cached in a flash data block of flash cache **50** is also stored along with the corresponding flash data block in flash cache **50**. In the example shown in FIG. **10**, address table **1021** is copied to address table **1061** in

correspondence with flash data block **1030** of flash cache **50**. In this manner, a portion of flash memory **25**-**1** to **25**-N is used as a cache in order to build up a large block of data for efficient transfer into a flash data block of main flash storage **55**, such as flash data block **1070**. As will be discussed further below, this flash cache aspect is particularly useful with regard to the gathering and organization of random incoming data into flash data blocks for efficient write operations of flash storage device **10**.

**[0102]** FIG. **11**A depicts an example is shown of a main flash storage address table according to an aspect of the invention. In FIG. **11**A, main flash storage address table **1100** is shown in which each of record entries **1150**-**1** to **1150**-N of main flash storage address table **1100** corresponds to an LBA of a flash data block stored in main flash storage **55**. In this regard, the record entries **1150**-**1** to **1150**-N may be arranged in numerical order of the LBAs so that LBA **0** corresponds to the first entry **1150**-**1**, LBA **1** to the second entry, and so on, whereby the highest number LBA corresponds to the last entry **1150**-N. The main flash storage address table **1100** includes columns for entry in each record of a channel number **1120**, a main flash storage physical address **1130** and a flash cache segment cross-reference address **1140**. The channel number **1120** indicates which channel of flash memory **25**-**1** to **25**-N corresponds to the particular LBA of the record. The main flash storage physical address **1130** indicates the actual physical address location in main flash storage **55** of the data associated with the LBA.

**[0103]** Flash cache segment cross-reference address **1140** may contain an address if there are one or more data segments stored in flash cache **50** associated with the LBA data stored in main flash storage **55**. This address entry is used as both an indicator of whether or not there is such corresponding data in flash cache **50**, and also in the case that there is such corresponding data in flash cache **50** it provides an actual physical address of a cross-reference table that associates the data segments in flash cache **50** with the LBA of the particular record in main flash storage address table **1100**. If no address is entered in the record (such as **1150**-**1**) for flash cache segment cross-reference address **1140**, then this indicates that there is no data in flash cache **50** that corresponds to the LBA in the record. If there is an address entered in the record (such as **1150**-**1**) for flash cache segment cross-reference address **1140**, then this indicates that there is data in flash cache **50** that corresponds to the LBA in the record, and the address is used to access a separate lookup table that correlates the data segments stored in flash cache **50** that correspond to the LBA in the record. It should be appreciated that main flash storage address table **1100** may not necessarily be in a table format but may instead be a link list of record entries **1150**-**1** to **1150**-N.

**[0104]** In this regard, FIG. **11**B, shows an example of flash cache segment cross-reference table **1160**. As seen in FIG. **11**B, flash cache segment cross-reference table **1160** is dedicated to a specific LBA (in this example, LBA X) corresponding to a flash data block in flash main storage **55**. It can therefore be appreciated that there are similar tables for all flash data blocks in flash main storage that have corresponding data provided in flash cache **50**. In this manner, the address to the flash cache segment cross-reference table corresponding to a particular LBA can be obtained from main flash storage address table **1100** as described above, and the flash cache segment cross-reference table can then be searched to determine of there are any data segments in flash cache **50** that correspond to the LBA.

**[0105]** Flash cache segment cross-reference table **1160** of FIG. **11**B includes record entries **1170**-**1** to **1170**-N that correspond to each possible data segment to the particular LBA. In this example, there are 128 data segments in each flash data block corresponding to an LBA, but there are other possible configurations depending on the size of the flash data block and the data segment. In each of record entries **1170**-**1** to **1170**-N there are two columns, flash cache data segment number **1161** and flash cache data segment address **1162**. Flash cache data segment number **1161** includes the particular data segment number for a data segment stored in flash cache **50** that corresponds to the flash data block associated with the particular LBA. There are only entries in flash cache segment cross-reference table **1160** for those data segments that are stored on flash cache **50**. Accordingly, there may be only one of record entries **1170**-**1** to **1170**-N provided in flash cache segment cross-reference table **1160** if there is only one data segment stored in flash cache **50** corresponding to the LBA. Record entries **1170**-**1** to **1170**-N are preferably arranged in ascending numerical order in flash cache segment cross-reference table **1160**, but that is not necessary. Flash cache data segment address **1162** is the physical address for the data segment identified in flash cache data segment number **1161** of each existing one of record entries **1170**-**1** to **1170**-N. In this manner, a segment cross-reference table can be quickly accessed for each LBA of main storage data to determine if the LBA has corresponding data segments provided in flash cache **50**. It should be appreciated that flash cache segment cross-reference table **1160** may not necessarily be in a table format but may instead be a link list of record entries **1170**-**1** to **1170**-N. Accordingly, flash cache segment cross-reference table **1160** provides an efficient form of address table **1021** formed in DRAM **30** as shown in FIG. **10**, which is then copied to address table **1061** in flash cache **50** when the writing of a flash data block from DRAM **30** data cache to flash cache **50** is completed.

**[0106]** In this regard, address table **1061** can also be provided with link lists that can be used to manage and track data segments of the flash data blocks of flash cache **50** in a similar to that already described above with respect to FIGS. **4**A, **4**B, **5**A, **5**B, **7** and **8**. In particular, link lists of the flash cache blocks can be used in a similar to that described above with respect to FIGS. **4**A, **4**B, **5**A and **5**B in order to manage the ordered use of flash cache blocks. For example a flash cache block that is empty can be placed at the top of the link list for next active use, followed by flash cache blocks that are the oldest or the least accessed, and the most active accessed flash cache blocks would be maintained at the bottom of the link list.

**[0107]** Similarly, multiple link lists of the flash cache blocks can be used in a similar to that described above with respect to FIG. **7** in order to track valid and invalid data segments in the flash cache blocks to assist in cleanup operations of the flash cache blocks, such as a background cleanup operation of the flash cache blocks in flash cache **50**, as is described in more detail below. Lastly, multiple link lists of the flash cache blocks can be used in a similar to that described above with respect to FIG. **8** in order to erase count status of the flash cache blocks to assist in wear leveling management of the flash cache blocks in flash cache **50**, as is described in more detail below.

**[0108]** FIG. **12** depicts a flowchart describing an operation for an allocated cache portion of the flash memory **25-1** to **25**-N, such as flash cache **50**, according to an embodiment of the invention. In FIG. **12**, the operation begins with step **1201** in which incoming data is sent to DRAM **30** data cache, such as from controller **20** in response to a write command from host **80** (both shown in FIG. **1**). Next, in step **1203**, the incoming data is filled into a cache data block of the data cache in DRAM **30**. A determination is made in step **1205** whether the incoming data stored in the cache data block is sequential data or is random data. If the incoming data is sequential data, then the data cache of DRAM **30** continues to fill cache data blocks with the incoming sequential data in step **1207** until set of cache data blocks is completed that is enough to fill a flash data block for being written to main flash storage **55**. In step **1209**, the completed set of cache data blocks is written to a flash data block of main flash storage **55**. The address table in data cache of DRAM **30** that corresponds to the flash data block is then updated with the physical address of the flash data block in main flash storage **55**. This path of the operation ends at step **1220**.

**[0109]** If the incoming data is not sequential data but is random data, then the cache data block of the incoming random data is written from the data cache of DRAM **30** to a data segment of a flash data block of flash cache **55** in step **1213**. In one aspect, the cache data block in the data cache of DRAM **30** is made free for being re-written once it is successfully written to the data segment of a flash data block of flash cache **55**. In this manner, less capacity of DRAM is necessary for data caching of incoming data. Accordingly, if there is a power failure, the data is already stored in flash cache **50** and if there is initially a write failure during the write of the data from data cache in DRAM **30** to flash cache **50** then the data can be written to a different flash data block of flash cache **50**. Next, in step **1215**, once enough data segments are available to fill the flash data block of flash cache **50**, the flash data block is written to main flash storage **55**. Then, the address table in data cache of DRAM **30** that corresponds to the flash data block is updated with the physical address of the flash data block in main flash storage **55**, and the address table is copied to flash cache **50** to be stored in association with the flash data block in flash cache **50**. This path of the operation ends at step **1220**. In this manner, large blocks of incoming sequential data are efficiently written directly to main flash storage **55** without the need for interim buffering in flash cache **50**. On the other hand, incoming random data is first directed to interim buffering in flash cache **50** where it is collected in a large block size and then efficiently written to main flash storage **55**. For example, much of the randomly written incoming data from host **80** may actually be sequential data that is written randomly. The above described use of flash cache **50** allows the randomly written sequential data to be organized into proper large sequential data blocks for efficient writing to main flash storage **55**. For incoming data that is written randomly from host **80** and is truly random (is not associated with other incoming data), the random data is written to main flash storage **55** using a read-modify-write operation for the corresponding large flash data block where the random data is to be stored. The above operations using flash cache **50** can result in a reduced capacity need for cache data in DRAM **30** and can reduce the need to re-arrange and cleanup data after it has been written into flash main storage **55**, otherwise known as a reduction in write amplification.

**[0110]** FIGS. **13**A and **13**B depict flowcharts of a power-down sequence and a power-up sequence for maintaining address tables for the flash cache according to an embodiment of the invention. In FIG. **13**A, a normal power down sequence is shown in which flash device **10** checks in step **1301** to see if write commands have been received since the device was last powered up. In step **1303**, it is determined if write commands have been received since the last power up, and if so, the current versions of the address tables for the flash data blocks in flash cache **50** are saved from the data cache of DRAM **30** to a static area of flash cache **50** in step **1305**. If, in step **1303**, it is determined that write commands have not been received since the last power up, then the versions of the address tables already stored in flash cache **50** are still current and do not need to be updated. In either case, a tag is written to the static area of flash cache **50** to indicate that flash device **10** is being powered down under normal circumstances, and the process ends at step **1309**.

**[0111]** In FIG. **13**B, a power up sequence is shown in which flash device **10** first checks in step **1321** for the presence of a tag was written to the static area of flash cache **50** to indicate that flash device **10** was last powered down under normal circumstances. In step **1323**, it is determined if such a tag is present, and if so, the address tables that are already stored in flash cache **50** are valid for use, so this path of the process ends at step **1331**. If, in step **1323**, it is determined if such a tag is not present, then in step **1327** the last saved versions of the address tables are obtained from the static area of flash cache **50** and the active flash data blocks of flash cache **50** are identified according to the address tables. Next, the flash device **10** scans all active flash blocks of flash cache **50** and reconstructs new versions of the address tables based on the LBA and physical address information of the active flash blocks. The process then ends at step **1331**. In this manner, the above described processes can prevent the loss of address table data during normal power down sequences, and can provide for efficient reconstruction of the address tables upon a power-up sequence following an abnormal power down of flash device **10**.

**[0112]** As mentioned above, the use of flash cache **50** allows for randomly written data that is sequential in nature to be efficiently packaged together for a particular flash data block and then efficiently written to main flash storage **55**. In this regard, FIGS. **14**A, **14**B and **14**C are block diagrams of a background "cleanup" operation that is conducted during idle time of flash device **10** to gather randomly written data stored in flash cache **50** for a common flash data block and write that block to main flash storage **55**, thereby freeing up flash data blocks in flash cache **50** for subsequent use.

**[0113]** As seen in FIG. **14**A, flash cache **50** is shown, and is seen to contain flash data blocks **1410**, **1420**, **1430** and on to **1450** and **1460**. Flash data blocks **1410**, **1420** and **1430** contain data and flash blocks **1450** and **1460** are empty and ready for use. In this regard, a background cleanup operation is performed when flash device **10** is idle, such as when there is one second during which no host commands have been sent to flash device **10**. In the example of FIG. **14**A, flash data blocks **1410**, **1420** and **1430** contain data segments **1411**, **1421** and **1431**, respectively, which have been randomly written into different flash data blocks but which are sequential and therefore belong to a same data block. These data segments can be identified by, for example, utilizing the address table described above with respect to FIG. **11**A and the cross-reference table described above with respect to FIG. **11**B.

**[0114]** FIG. **14**B depicts the next step of the background cleanup operation example. In FIG. **14**B, the background cleanup operation writes the data segments **1411**, **1421** and **1431** from flash data blocks **1410**, **1420** and **1430**, respectively, to sequential data segments **1471**, **1472** and **1473** in flash data block **1470** of flash main storage **55**. Also, the data on either side of data segment **1411** in flash data block **1410** is moved to empty flash data block **1450**. In this manner, the sequential data in data segments **1411**, **1421** and **1431** is gathered and written in a block fashion to flash main storage **55** and the remaining data of flash data block **1410** is sent to a new block of flash cache **50**, thereby making flash data block **1410** of flash cache **50** ready for re-use.

**[0115]** In FIG. **14**C, flash data block **1410** of flash cache **50** is erased and is now made ready for re-use. Also, flash data segments **1421** and **1431** of flash data blocks **1420** and **1430** of flash cache **50** are marked as invalid now that their data has been moved to main flash storage **55**. From here, the operation can continue and all flash data blocks containing invalid data segments, or above a certain number of invalid data segments, can be identified and cleaned up in a similar fashion. In one aspect, certain parameters are used to control the amount of cleanup performed in the background cleanup operation. For example, a threshold percentage of all flash data blocks in flash cache **50** can be set, such as 50%, and when this percentage of flash data blocks in flash cache **50** have been cleaned and emptied, the background cleanup operation would be stopped until a subsequent idle time. In this example, the next instance of the background cleanup operation could begin at the next flash data block in sequence after the last flash data block that was cleaned up in the last instance of the background cleanup operation. Also, the background cleanup operation could be limited to operate on cleanup of only those flash data blocks in flash cache **50** that would result in at least a threshold number, such as 16, of data segments to be written to a block of main flash storage **55**. These thresholds and parameters can of course be changed to achieve more or less cleanup of flash data blocks in flash cache **50** with each instance of the background cleanup operation.

**[0116]** Accordingly, the example of the background cleanup operation described above can efficiently gather and write sequential data from flash cache **50** to flash main storage **55** and also free flash data blocks in flash cache **50** for subsequent reuse. As data is written to main flash storage, and data in flash data blocks of flash cache **50** is rendered invalid and flash data blocks of flash cache **50** are erased, the address tables corresponding to the relevant flash data blocks are updated to represent the new location of the data, and to represent the status of the flash data blocks in flash cache **50**. For example, as mentioned above, in one aspect of the invention link lists can be maintained to represent the empty "erased" flash data blocks of flash cache **50** which are ready for use, such as the link lists described above with respect to FIGS. **4**A, **4**B, **5**A and **5**B. These link lists can be stored in data cache of DRAM **30** along with the address table of the associated flash data block. Similarly, link lists can also be maintained to represent the flash data blocks of flash cache **50** that contain invalid data segments, such as the link lists described above with respect to FIG. **7**.

**[0117]** FIG. **15** is a flowchart that describes the steps of a background cleanup operation according to an embodiment of the invention. As seen in FIG. **15**, the process begins at step **1501** when flash device **10** is idle, such as when there is one second during which no host commands have been sent to flash device **10**. In step **1501**, data segments are identified in flash cache **50** which have been randomly written into different flash data blocks of flash cache **50** but which are sequential and therefore belong to a same data block. These data segments can be identified by, for example, utilizing the address table described above with respect to FIG. **11**A and the cross-reference table described above with respect to FIG. **11**B. Next, in step **1503**, the identified data segments from step **1501** are written to a same flash data block of main flash storage **55**. For example, the same flash data block of main flash storage **55** can be the flash data block corresponding to the LBA associated with the identified data segments. In one aspect of the invention, the identified data segments are only written to a same flash data block of main flash storage **55** in step **1503** if the number of identified data segments is equal to or greater than a predetermined threshold, such as 16, in order increase efficiency by reducing write amplification. Of course, this threshold can be modified (such as to 8, 4, 2 and 1) as it becomes more necessary to cleanup more flash data blocks. The identified data segments in the flash data blocks of flash cache **30** that were just written to main flash storage **55** are marked as invalid in the address table and/or link lists as described above.

**[0118]** In step **1505**, one of the flash data blocks in flash cache **50** is identified based on having a certain level of invalid data segments. This can be a flash data block of flash cache **50** from which the most number of data segments were just written to main flash storage **55**, or it can be another flash data block of flash cache **50** with a high amount of invalid data segments. As discussed above, link lists can be used to represent the flash data blocks of flash cache **50** that contain certain levels of invalid data segments, such as the link lists described above with respect to FIG. **7**. These link lists can be used for the background cleanup operation to make the determination in step **1505** of which next flash data block of flash cache **50** should be selected for cleanup.

**[0119]** In step **1507**, the remaining data segments (other than the invalid data segments) of the identified flash data block are written to an empty flash data block of flash cache **50**, thereby making the identified flash data block ready for erase and re-use. The data segments of identified flash data block are then erased in step **1509**. The address tables and/or link lists discussed above are updated in step **1511** to reflect the writing of data segments to main flash storage **55**, and to a new flash data block in flash cache **50** and to reflect the fact that the identified flash data block is now empty and ready for re-use. It is next determined in step **1513** if the background cleanup operation should continue by checking the link lists associated with flash cache **50** to see if the number of empty flash data blocks is above a certain percentage (such as 50%) of all flash data blocks in flash cache **50**. If the number of empty flash data blocks is not above the certain percentage, then the background cleanup operation continues again at step **1501**. If the number of empty flash data blocks is above the certain percentage, then the identity of the next flash data block in cleanup sequence (based on number of invalid blocks or other factors) is identified and stored in step **1515** in static data of flash cache **50**. In this manner, the next instance of the background cleanup operation can begin at the next identified flash data block. The process then ends at step **1517**. In this manner, the background cleanup operation efficiently gathers

and writes sequential data segments from flash cache **50** into flash main storage **55** and also frees flash data blocks in flash cache **50** for subsequent use.

**[0120]** The combination of moving sequential data to main flash storage and cleaning up flash data blocks with invalid data segments during the background cleanup operation improves the efficiency of the flash cache **50**. Other techniques and methods can also be employed during the background cleanup operation to improve efficiency and to maintain the operability of the flash data blocks of flash cache **50**, such as a wear leveling operation as described above with respect to FIG. **8**. In this manner, static wear leveling can also be achieve for the flash data blocks of flash cache **50** based on maintained relative erase counts for the flash data blocks in static wear link lists, whereby the erase counts of all flash data blocks of flash cache **50** are maintained with a predetermined threshold of each other, such as 512 or 1024 erase counts.

**[0121]** While the present invention has been particularly described with reference to the various figures and embodiments, it should be understood that these are for illustration purposes only and should not be taken as limiting the scope of the invention. There may be many other ways to implement the invention. Many changes and modifications may be made to the invention, by one having ordinary skill in the art, without departing from the spirit and scope of the invention.

**1**. A method of transferring data in a flash storage device comprising a random access memory and a flash memory, the method comprising:

receiving data from a host system, the data comprising a data segment;

temporarily storing the data segment in a data buffer of the random access memory;

assigning the data segment to a logical block address;

determining if the data segment stored in the data buffer comprises sequential data or random data; and

writing the data segment to an allocated cache portion of the flash memory if the data segment is determined to comprise random data.

**2**. The method of claim **1**, further comprising a second writing step of writing the data segment from the allocated cache portion of the flash memory to a main storage portion of the flash memory.

**3**. The method of claim **1**, wherein the flash memory comprises an array of plurality of flash memory units, and the allocated cache portion of the flash memory is evenly distributed across the plurality of flash memory units.

**4**. The method of claim **1**, wherein the flash memory comprises an array of plurality of flash memory units, and the allocated cache portion of the flash memory is not uniformly distributed across the plurality of flash memory units.

**5**. (canceled)

**6**. The method of claim **2**, wherein in the second writing step the data segment is written from the allocated cache portion to a location in the main storage portion that corresponds to the logical block address assigned to the data segment.

**7**. The method of claim **6**, wherein in the second writing step a plurality of other data segments having the same assigned logical block address as the data segment are written along with the data segment to the location in the main storage portion that corresponds to the assigned logical block address.

**8**. The method of claim **7**, wherein the second writing step is not conducted until a predetermined number of data segments having the same assigned logical block address have been written to the allocated cache portion of the flash memory.

**9**. The method of claim **8**, wherein the allocated cache portion of the flash memory and the main storage portion of the flash memory are each comprised of a plurality of flash data blocks each of which is sized to accommodate the predetermined number of data segments.

**10**. The method of claim **1**, wherein the allocated cache portion of the flash memory and the main storage portion of the flash memory are each comprised of a plurality of flash data blocks.

**11**. The method of claim **7**, wherein the data segment and the plurality of other data segments are sequential data segments.

**12**. The method of claim **2**, further comprising:

maintaining an address table in the random access memory, wherein the address table comprises a plurality of logical block addresses cross-referenced to a plurality of physical addresses; and

updating the address table upon writing the data segment from the allocated cache portion to a location in the main storage portion of the flash memory by storing the physical address of the location as a cross-reference to the logical block address assigned to the data segment.

**13**. The method of claim **12**, wherein the address table also includes a cross-reference between the plurality of logical block addresses and a plurality of allocated cache portion indicators, each indicator indicating that a data segment corresponding to the respective logical block address is stored in the allocated cache portion of the flash memory.

**14**. The method of claim **13**, wherein the each of the plurality of allocated cache portion indicators is a physical address pointing to a different logical block cache table stored in the random access memory, the logical block cache table comprising a list of all data segments stored in the allocated cache portion that correspond to the same logical block address which is cross-referenced to the allocated cache portion indicator.

**15**. The method of claim **14**, wherein the logical block cache table is a link list of all data segments stored in the allocated cache portion that correspond to the same logical block address which is cross-referenced to the allocated cache portion indicator.

**16**. The method of claim **14**, wherein the logical block cache table also includes a physical address of a location in the allocated cache portion where each data segment listed in the logical block cache table is stored.

**17**. The method of claim **1**, further comprising:

writing the data segment from the data buffer of the random access memory to a main storage portion of the flash memory without temporarily storing the data segment in the allocated cache portion of the flash memory if the data segment is determined to comprise sequential data.

**18**. The method of claim **17**, wherein a plurality of other data segments having the same assigned logical block address as the data segment are written along with the data segment to a location in the main storage portion of the flash memory that corresponds to the assigned logical block address.

**19**. The method of claim **12**, further comprising:

storing a copy of the address table in the flash memory prior to a power-down operation of the flash storage device; and

transferring the address table from the flash memory to the random access memory upon power-up operation of the flash storage device.

**20**. The method of claim **19**, wherein in the storing step, a normal power- down indicator tag is also stored in flash memory, and further comprising a reconstruction step, which upon power-up operation of the flash storage device checks the flash memory for the presence of a normal power-down indicator tag, wherein, in the case that the normal power-down indicator tag is not present, all data segments in flash memory are scanned to gather information related to all data segments present in flash memory and their corresponding logical block addresses, and the address table is reconstructed based on the gathered information.

**21**. The method of claim **2**, further comprising:

conducting a background cleanup operation in which the allocated cache portion is scanned to identify a set of data segments that collectively contain a set of sequential data, and writing the set of data segments to the main storage portion of the flash memory, upon which each one of the set of data segments is identified as in invalid data segment.

**22**. The method of claim **21**, wherein the flash memory is comprised of a plurality of flash data blocks, and wherein the background cleanup operation further includes selecting a flash data block which has a predetermined number of invalid data segments and then erasing the selected flash data block for subsequent use and identifying the selected flash data block as an erased flash data bock.

**23**. The method of claim **22**, wherein in the background cleanup operation all data segments in the selected flash data block which are not identified as an invalid data segment are moved to another flash data block in the allocated cache portion before the selected flash data block is erased.

**24**. The method of claim **22**, wherein the background cleanup operation continues until a predetermined percentage of all flash data blocks are identified as erased flash data bocks.

**25**. A flash storage device comprising:

a flash memory;

a random access memory; and

a controller configured to perform the steps of:

receiving data from a host system, the data comprising a data segment;

temporarily storing the data segment in a data buffer of the random access memory;

assigning the data segment to a logical block address;

determining if the data segment stored in the data buffer comprises sequential data or random data; and

writing the data segment to an allocated cache portion of the flash memory if the data segment is determined to comprise random data.

**26**. The flash storage device of claim **25**, the controller being further configured to perform a second writing step of writing the data segment from the allocated cache portion of the flash memory to a main storage portion of the flash memory.

**27**. The flash storage device of claim **25**, wherein the flash memory comprises an array of plurality of flash memory units, and the allocated cache portion of the flash memory is evenly distributed across the plurality of flash memory units.

**28**. The flash storage device of claim **25**, wherein the flash memory comprises an array of plurality of flash memory units, and the allocated cache portion of the flash memory is not uniformly distributed across the plurality of flash memory units.

**29**. (canceled)

**30**. The flash storage device of claim **26**, wherein in the second writing step the data segment is written from the allocated cache portion to a location in the main storage portion that corresponds to the logical block address assigned to the data segment.

**31**. The flash storage device of claim **30**, wherein in the second writing step a plurality of other data segments having the same assigned logical block address as the data segment are written along with the data segment to the location in the main storage portion that corresponds to the assigned logical block address.

**32**. The flash storage device of claim **31**, wherein the second writing step is not conducted until a predetermined number of data segments having the same assigned logical block address have been written to the allocated cache portion of the flash memory.

**33**. The flash storage device of claim **32**, wherein the allocated cache portion of the flash memory and the main storage portion of the flash memory are each comprised of a plurality of flash data blocks each of which is sized to accommodate the predetermined number of data segments.

**34**. The flash storage device of claim **25**, wherein the allocated cache portion of the flash memory and the main storage portion of the flash memory are each comprised of a plurality of flash data blocks.

**35**. The flash storage device of claim **31**, wherein the data segment and the plurality of other data segments are sequential data segments.

**36**. The flash storage device of claim **26**, the controller being further configured to perform the steps of :

maintaining an address table in the random access memory, wherein the address table comprises a plurality of logical block addresses cross-referenced to a plurality of physical addresses; and

updating the address table upon writing the data segment from the allocated cache portion to a location in the main storage portion of the flash memory by storing the physical address of the location as a cross-reference to the logical block address assigned to the data segment.

**37**. The flash storage device of claim **36**, wherein the address table also includes a cross-reference between the plurality of logical block addresses and a plurality of allocated cache portion indicators, each indicator indicating that a data segment corresponding to the respective logical block address is stored in the allocated cache portion of the flash memory.

**38**. The flash storage device of claim **37**, wherein the each of the plurality of allocated cache portion indicators is a physical address pointing to a different logical block cache table stored in the random access memory, the logical block cache table comprising a list of all data segments stored in the allocated cache portion that correspond to the same logical block address which is cross-referenced to the allocated cache portion indicator.

**39**. The flash storage device of claim **38**, wherein the logical block cache table is a link list of all data segments stored in the allocated cache portion that correspond to the same logical block address which is cross-referenced to the allocated cache portion indicator.

**40**. The flash storage device of claim **38**, wherein the logical block cache table also includes a physical address of a location in the allocated cache portion where each data segment listed in the logical block cache table is stored.

**41**. The flash storage device of claim **26**, the controller being further configured to perform the step of:

writing the data segment from the data buffer of the random access memory to a main storage portion of the flash memory without temporarily storing the data segment in the allocated cache portion of the flash memory if the data segment is determined to comprise sequential data.

**42**. The flash storage device of claim **41**, wherein a plurality of other data segments having the same assigned logical block address as the data segment are written along with the data segment to a location in the main storage portion of the flash memory that corresponds to the assigned logical block address.

**43**. The flash storage device of claim **36**, the controller being further configured to perform the steps of:

storing a copy of the address table in the flash memory prior to a power-down operation of the flash storage device; and

transferring the address table from the flash memory to the random access memory upon power-up operation of the flash storage device.

**44**. The flash storage device of claim **43**, wherein in the storing step, a normal power-down indicator tag is also stored in flash memory, and the controller being further configured to perform a reconstruction step, which upon power-up operation of the flash storage device checks the flash memory for the presence of a normal power-down indicator tag, wherein, in the case that the normal power-down indicator tag is not present, all data segments in flash memory are scanned to gather information related to all data segments present in flash memory and their corresponding logical block addresses, and the address table is reconstructed based on the gathered information.

**45**. The flash storage device of claim **26**, the controller being further configured to perform the step of:

conducting a background cleanup operation in which the allocated cache portion is scanned to identify a set of data segments that collectively contain a set of sequential data, and writing the set of data segments to the main storage portion of the flash memory, upon which each one of the set of data segments is identified as in invalid data segment.

**46**. The flash storage device of claim **45**, wherein the flash memory is comprised of a plurality of flash data blocks, and wherein the background cleanup operation further includes selecting a flash data block which has a predetermined number of invalid data segments and then erasing the selected flash data block for subsequent use and identifying the selected flash data block as an erased flash data bock.

**47**. The flash storage device of claim **46**, wherein in the background cleanup operation all data segments in the selected flash data block which are not identified as an invalid data segment are moved to another flash data block in the allocated cache portion before the selected flash data block is erased.

**48**. The flash storage device of claim **46**, wherein the background cleanup operation continues until a predetermined percentage of all flash data blocks are identified as erased flash data bocks.

**49**. A non-transitory machine readable medium carrying one or more sequences of instructions for managing memory operations in a flash storage device having a flash memory, a random access memory and a controller, wherein execution of the one or more sequences of instructions by one or more processors in the controller cause the one or more processors to perform the steps of:

receiving data from a host system, the data comprising a data segment;

temporarily storing the data segment in a data buffer of the random access memory;

assigning the data segment to a logical block address;

determining if the data segment stored in the data buffer comprises sequential data or random data; and

writing the data segment to an allocated cache portion of the flash memory if the data segment is determined to comprise random data.

* * * * *