(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 548 194**

(51) Int. CI.:

G06F 12/02 (2006.01)
G06F 12/08 (2006.01)
G06F 12/10 (2006.01)
G06F 17/30 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **25.06.2010 E 10731653 (1)**

(97) Fecha y número de publicación de la concesión europea: **16.09.2015 EP 2433227**

(54) Título: **Indexación escalable en una memoria de acceso no uniforme**

(30) Prioridad:

26.06.2009 US 269633 P

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
14.10.2015

(73) Titular/es:

SIMPLIVITY CORPORATION (100.0%)
8 Technology Drive
Westborough, MA 01581-1756, US

(72) Inventor/es:

BOWDEN, PAUL y
BEAVERSON, ARTHUR J.

(74) Agente/Representante:

UNGRÍA LÓPEZ, Javier

DESCRIPCIÓN

Indexación escalable en una memoria de acceso no uniforme

Campo de la invención

La presente invención se refiere a métodos y aparatos para la construcción de un índice que escala a gran número de registros y proporciona una alta tasa de transacciones.

Antecedentes

Algunos sistemas de archivo modernos usan objetos para almacenar datos de archivo y otras estructuras internas del sistema de archivo ("metadatos"). Un archivo se descompone en muchos objetos pequeños, tal vez de sólo 4 KB (2^12 bytes). Para un sistema de archivo que abarca 64 TB (2^46 bytes), por ejemplo, esto da lugar a más de 2^(46-12) = 2^34, o aproximadamente 16 miles de millones de objetos cuya pista seguir.

En este contexto un objeto es una secuencia de datos binarios y tiene un nombre de objeto, a menudo una GUID (ID globalmente única), o un hash criptográfico del contenido, aunque otras convenciones de denominación son posibles a condición de que cada objeto único tenga un nombre único. Los nombres de objeto suelen ser cadenas binarias de longitud fija destinadas al uso por programas, en contraposición a personas. Los tamaños de objeto son arbitrarios, pero en la práctica son normalmente potencias de 2 y son del rango de 512 bytes (2^9) hasta 1MB (2^20). En este contexto los objetos no deberán confundirse con los objetos usados en lenguajes de programación tal como Java y C++.

El sistema de archivo necesita un índice (a veces denominado un diccionario o catálogo) de todos los objetos. Cada registro en el índice puede contener el nombre de objeto, la longitud, la posición y otra información variada. El índice puede tener como su clave primaria el nombre de objeto, la posición del objeto, o posiblemente ambos. Un registro es del orden de unas pocas decenas de bytes, siendo 32 bytes un ejemplo.

Las operaciones en este índice incluyen añadir una entrada, consultar una entrada, hacer modificaciones en la entrada, y borrar una entrada. Todas éstas son operaciones normales realizadas en cualquier índice.

Dado que estos sistemas de archivo operan con objetos, para que el sistema de archivo logre niveles de rendimiento aceptables, una solución de indexación tiene dos retos que no se cumplen fácilmente:

1) El número de entradas en el índice puede ser muy grande. En el ejemplo enumerado anteriormente, si cada entrada de índice es 32 (2^5) bytes, entonces el índice ocupa 2^(5+34) = 2^39, o 512 GB de memoria. Esto no encaja en el costo efectivamente de las tecnologías de memoria actuales.

2) Las operaciones contra el índice son grandes. Un sistema de almacenamiento comercialmente viable puede tener que funcionar a, por ejemplo, 256 MB/s (2^28 bytes/segundo). En tamaños de objeto de 4 KB, eso es 2^28-12) = 2^16, o 64 mil operaciones por segundo. Dado que los sistemas de archivo generan y referencian normalmente otros datos (objetos) internamente, la tasa operativa del índice puede exceder fácilmente de 100 mil operaciones/segundo. Como punto de comparación, un disco del estado actual de la técnica puede efectuar a lo sumo 400 operaciones por segundo.

Lograr los niveles necesarios de rendimiento y capacidad no es posible usando tecnología de memoria DRAM, o tecnología de disco, solas. La memoria DRAM es suficientemente rápida, pero no suficientemente densa. Los discos tienen la densidad, pero no el rendimiento. Escalar uno (la memoria DRAM o los discos) para lograr las características deseadas es demasiado caro.

Los nombres de objeto son a menudo uniformes tanto en su distribución como en sus configuraciones de acceso, de modo que los esquemas de cache normales, que dependen de localidad espacial y temporal, tienen un efecto limitado. Así, el problema de la indexación es difícil tanto en el tamaño como en las tasas de operación.

Un documento titulado: "An efficient Hash Index Structure for Solid State Disks", del que son autores Hongehan Roh y Sanghyun Park, propone una estructura de índice hash eficiente para SSDs que resuelve los inconvenientes SSDs y mejora su rendimiento. Una memoria intermedia que está estrechamente acoplada con el índice hash acumula peticiones de actualización de índice y se aplica una política de borrado de base log al índice.

Resumen de la invención

Un método de adaptar un acceso uniforme y su proceso de indexación con una memoria flash NAND de acceso no uniforme se reivindica en la reivindicación independiente 1. Un sistema informático se reivindica en la reivindicación independiente 19.

En una realización, el método incluye:

leer una o más entradas de datos de registro secuenciales de la memoria a la cache;

designar como libres las posiciones físicas en memoria de la que se leyó la una o más entradas.

En una realización, el método incluye:

presentar una pluralidad de posiciones de compartimento físico secuenciales en la memoria como un bloque libre leyendo cualesquiera entradas válidas en el bloque a la cache y designar como libres las posiciones físicas en memoria de la que se leyeron tales entradas.

En una realización:

el proceso de indexación genera peticiones de acceso aleatorio al índice en base a claves de índice uniformemente distribuidas y únicas.

En una realización:

las claves incluyen compendios de hash criptográfico.

En una realización:

el proceso de indexación incluye un proceso de hashing de desplazamiento.

En una realización:

el hashing de desplazamiento incluye un proceso de hashing de Cuckoo.

En una realización:

la memoria está limitada por uno o varios del tiempo de acceso a escritura aleatorio, tiempo de acceso a lectura-modificación-escritura aleatorio, escritura secuencial, restricciones de alineación, tiempo de borrado, límites y desgaste de bloque de borrado.

En una realización:

un tamaño del compartimento físico incluye un tamaño de escritura mínimo de la memoria.

En una realización:

el tamaño del compartimento físico incluye una página o página parcial.

En una realización:

la memoria tiene un bloque de borrado incluyendo una pluralidad de páginas.

En una realización el método incluye:

mantener una tabla de compartimentos válidos para hacer el seguimiento de qué posiciones de compartimento en la memoria son válidas.

En una realización:

un compartimento en memoria incluye un conjunto de una o más entradas de datos de registro y un autoíndice a la tabla de traducción de compartimentos.

En una realización:

las entradas de datos de registro en el compartimento no están ordenadas.

En una realización:

la entrada de datos de registro incluye campos para una clave, un recuento de referencias y una dirección de bloque físico.

En una realización:

la clave incluye un compendio de datos de hash criptográfico;

el campo de dirección de bloque físico contiene un indicador a la dirección de bloque físico de los datos almacenados en un dispositivo de almacenamiento.

En una realización:

las posiciones de compartimento lógicas son generadas por una pluralidad de funciones hash.

En una realización:

la memoria incluye una pluralidad de bloques de borrado, cada bloque de borrado incluye una pluralidad de páginas, y cada página incluye una pluralidad de compartimentos.

Según otra realización de la invención, se facilita un producto de programa de ordenador incluyendo un medio de código de programación que, cuando es ejecutado por un procesador, realiza los pasos del método anterior.

En una realización el método incluye:

designar como leídas solamente en cache las entradas de datos de registro escritas secuencialmente en la memoria.

En una realización:

la tabla de traducción de compartimentos se almacena en memoria persistente.

En una realización, el método incluye:

rastrear el número de compartimentos libres en un bloque de borrado e implementar un proceso para generar un bloque de borrado libre cuando se cumple un umbral de compartimentos libres.

En una realización:

el proceso de indexación realiza operaciones de indexación en base a peticiones de que los registros sean insertados, borrados, consultados y/o modificados.

En una realización:

el proceso de indexación presenta operaciones de compartimento lógico para leer y escribir a compartimentos físicos que almacenan los registros del índice.

En una realización:

las operaciones de compartimento físico incluyen lecturas aleatorias y escrituras secuenciales.

En una realización:

las operaciones de compartimento físico incluyen además órdenes de recorte.

En una realización:

la memoria incluye una capa de dispositivo físico caracterizada por acceso de lectura y escritura no uniforme e inmutabilidad con respecto al tamaño, la alineación y la temporización.

Breve descripción de los dibujos

La invención se entenderá más plenamente por referencia a la descripción detallada, en unión con las figuras siguientes:

La figura 1 es un diagrama esquemático de bloques que ilustra varias operaciones de indexación realizadas según una realización de la presente invención.

Las figuras 2A a 2D ilustran varias realizaciones de estructuras de datos que pueden ser usadas en la presente invención.

La figura 3 es un diagrama esquemático de bloques que ilustra una operación de consulta según una realización de la invención.

La figura 4 es un diagrama esquemático de bloques que ilustra una operación de inserción según una realización de la invención.

La figura 5 es un diagrama esquemático de bloques de una operación de borrado según una realización de la invención.

La figura 6 es un diagrama esquemático de bloques de una operación de actualización según una realización de la invención.

Las figuras 7A y 7B son diagramas de bloques esquemáticos que ilustran un proceso de lectura aleatoria para generar bloques de borrado libres según una realización de la invención.

Las figuras 8A y 8B son diagramas de bloques esquemáticos que ilustran otro método de generar bloques de borrado libres según un proceso de rebusca.

La figura 9 es un diagrama esquemático de bloques que ilustra una vista o pila de seis capas para ilustrar una implementación de la presente invención.

La figura 10 es un diagrama esquemático de una entrada de registro usada en una realización de la invención.

Las figuras 11A- 11E ilustran esquemáticamente una implementación de hashing de Cuckoo según una realización de la invención.

La figura 12 es una ilustración esquemática de múltiples compartimentos, conteniendo cada compartimento múltiples registros según una realización de la invención.

La figura 13 es un diagrama esquemático del contenido de un compartimento según una realización de la invención.

La figura 14 es un diagrama esquemático de bloques que ilustra un ejemplo de un chip flash físico que tiene múltiples dados, bloques de borrado, páginas, y compartimentos según una realización de la invención.

Y las figuras 15A-15B ilustran algunos componentes de una capa de gestión de dispositivo según una realización de la invención.

Descripción detallada

A. Visión general

Según una o más realizaciones de la invención, se usan tecnología de memoria y algoritmos especializados para crear índices que tienen simultáneamente grandes números de registros y requisitos de transacción. Una realización utiliza un algoritmo de indexación de hashing de desplazamiento, por ejemplo hashing de Cuckoo. La invención permite el uso de tecnologías de memoria de acceso no uniforme tal como dispositivos de memoria flash, cambio de fase y disco de estado sólido (SSD).

En varias realizaciones de la invención, se facilitan nuevas estructuras de datos y métodos para asegurar que un algoritmo de indexación funcione de una forma que sea natural (eficiente) para el algoritmo, mientras que el dispositivo de memoria ve configuraciones de I/O (entrada/salida) que son eficientes para el dispositivo de memoria.

Se crea una estructura de datos, una tabla de indirección, que mapea compartimentos lógicos vistos por el algoritmo de indexación a compartimentos físicos en el dispositivo de memoria. Este mapeado es tal que se mejora la operación de escritura en dispositivos de memoria de acceso no uniforme.

Se usa otra estructura de datos, una cache asociativa, para recoger compartimentos y escribirlos secuencialmente en el dispositivo de memoria, como parte de las políticas de evicción y reescritura de cache.

Se usan métodos para poblar la cache con compartimentos (de registros) que son requeridos por el algoritmo de indexación. Se puede leer compartimentos adicionales del dispositivo de memoria a la cache durante una lectura de demanda, o por un proceso de rebusca.

El uso de la cache, en unión con la tabla de indirección, permite grandes escrituras secuenciales en el dispositivo de memoria.

Aunque la tecnología flash tiene la capacidad fundamental de lograr la necesaria capacidad y tasas IO para el problema de indexación, las características de acceso flash no son uniformes. Esta no uniformidad es suficientemente significativa para que los algoritmos de indexación normales funcionen pobremente, si funcionan, con un dispositivo de memoria flash.

La memoria flash de acceso no uniforme que se usa en la presente invención es una memoria de lectura solamente programable borrable eléctricamente (EEPROM) que debe ser leída, escrita y borrada en tamaños de bloques grandes de cientos a miles de bits, es decir, no acceso aleatorio a nivel de byte. Físicamente, flash es una forma de memoria no volátil que guarda información en una serie de celdas de memoria hechas de transistores de puerta flotante. Hay dos tipos de dispositivos de memoria flash, flash NAND y flash NOR. La flash NAND proporciona una densidad más alta y gran capacidad a costo más bajo, con velocidades más rápidas de borrado, escritura secuencial y lectura secuencial, que la flash NOR. En el sentido en que se usa en esta solicitud y en la presente invención, memoria “flash” pretende cubrir memoria flash NAND y no memoria NOR. NAND incluye tanto dispositivos de celda de nivel único (SLC), donde cada celda guarda solamente un bit de información, como dispositivos de celdas multinivel (MLC) más nuevos, que pueden almacenar más que un bit por celda. Aunque flash NAND proporciona tiempos de acceso rápidos, no es tan rápida como la memoria DRAM volátil usada como memoria principal en PCs. Un dispositivo de memoria flash puede incluir o no un sistema de archivo flash. Los sistemas de archivo flash se usan normalmente con memorias flash embebidas que no tienen un controlador incorporado para realizar nivelación de desgaste y corrección de errores.

Un chip NAND flash normal puede almacenar varios GB de contenido. A diferencia de la memoria instalada en un ordenador, a la memoria del chip flash se debe acceder en ciertos tamaños y con ciertos límites. Además, una vez que se ha escrito una sección de memoria, hay que realizar una operación de borrado antes de que dichas posiciones de memoria puedan ser escritas de nuevo. Además, las posiciones se desgastan, de modo que asegurar que todas las posiciones tengan un número similar de escrituras complica más el uso. Los tiempos de lectura, los tiempos de escritura y los tiempos de borrado pueden variar de forma significativa (de microsegundos a milisegundos). Así, la temporización, la nivelación de desgaste y las restricciones de alineación hacen difícil el uso práctico de flash en el mejor de los casos.

Un dispositivo de memoria flash puede contener uno o más dados (pastillas de silicio). A cada dado, en su mayor parte, se puede acceder independientemente.

Un dado está compuesto de miles de bloques de borrado. Un bloque de borrado tiene un tamaño normal de 128-512 KB. Cuando hay que borrar datos, deben ser borrados en límites de bloques de borrado.

Otra limitación de flash NAND es que los datos solamente pueden escribirse secuencialmente. Además, el tiempo de configuración para escritura es largo, aproximadamente 10X que el de una lectura.

Se leen datos sobre granularidad de página. Una página puede ser del rango de 1 KB a 4 KB dependiendo del chip flash concreto. Con cada página están asociados unos pocos bytes que pueden ser usados para suma de verificación de código de corrección de error (ECC).

Se escriben datos sobre granularidad de página. Una vez escrita, la página no se puede escribir de nuevo hasta que su bloque de borrado (conteniendo la página) sea borrado. Un bloque de borrado puede contener desde varias docenas a más de 100 páginas.

Una excepción a la granularidad de página de lectura y escritura anterior son las escrituras subpágina, o programación de página parcial. Dependiendo de la tecnología, las páginas pueden ser escritas parcialmente hasta 4 veces antes de que sea necesario un borrado.

Dado que las páginas en un bloque flash NAND pueden ser escritas secuencialmente y solamente una vez entre operaciones de borrado de bloque, las escrituras posteriores requieren una escritura en una página diferente, situada normalmente en un bloque flash diferente. La cuestión de los borrados de bloque se maneja creando un grupo de bloques flash escribibles, una función del sistema de archivo flash.

Borrar un bloque de borrado es la operación más cara desde el punto de vista del tiempo, puesto que puede durar varios milisegundos. En dispositivos que se usan intensamente (con tráfico), la velocidad a la que se puede generar bloques de borrado (es decir, con qué rapidez puede haber disponibles bloques de borrado libres) es a menudo un factor de limitación en el diseño flash.

Muchos SSD (discos de estado sólido) usan tecnología flash. Los microprogramas en el SSD manejan dichos problemas de acceso en una capa llamada la capa de traducción en flash (FTL). Sin embargo, al hacerlo, los microprogramas hacen suposiciones acerca de la forma en que se usará el SSD (por ejemplo, en su mayor parte lecturas, en su mayor parte escrituras, tamaño y alineación de lecturas y escrituras), y como resultado de estas suposiciones, las características de rendimiento del SSD son a menudo subóptimas para algoritmos de indexación.

Muchos algoritmos de indexación que se encuentran en la literatura y en la práctica se basan en un modelo de acceso a memoria uniforme, es decir, toda la memoria es igualmente accesible en el tiempo tanto para lecturas como escrituras, y no hay restricciones de primer orden en el tamaño de acceso o la alineación.

Si se considera una solución de indexación, operaciones tales como inserción, borrado, consulta y modificación requieren normalmente cantidades de tiempo mayores y variadas, y escrituras y lecturas de bloques, normalmente pequeños bloques (4 KB más o menos), menos tiempo. Los bloques parecen ser aleatorios, es decir, cualquier bloque puede ser leído, y cualquier otro bloque puede ser escrito. Con algunos algoritmos, hay 10 perfiles de lectura-modificación-escritura aleatorios, es decir, se lee un bloque aleatorio, y luego se escribe de nuevo en la misma posición con datos ligeramente modificados.

Esta IO aleatoria que un algoritmo de indexación necesita para operar eficientemente, no es lo que flash pretende proporcionar. Aunque flash puede manejar también lecturas aleatorias, las escrituras aleatorias son difíciles, como las lecturas-modificaciones-escrituras. La razón de esto es que no se puede sobreescribir algo que ya se haya escrito, hay que borrarlo primero. Para complicar más la situación, el borrado tarda un tiempo, y debe tener lugar en grandes límites (normalmente 64 KB).

Cuando se borra un bloque de borrado, los datos válidos de dicho bloque tienen que ser movidos a otro lugar. Si el algoritmo escribe bloques de 4 KB aleatorios a través del dispositivo flash, una implementación ingenua daría lugar a que se borrasen bloques en todo el tiempo. Dado que los tiempos de borrado son lentos, el rendimiento sufriría significativamente.

Según la invención, para que las escrituras en la flash puedan ser secuenciales, pero conservando el acceso aleatorio lógico que espera el algoritmo de indexación, se crea una tabla de traducción o indirección. Esta tabla mapea los compartimentos lógicos (de registros) que necesite el algoritmo de indexación a compartimentos físicos (por ejemplo, páginas) del dispositivo flash.

Dado que el algoritmo de indexación lee en compartimentos (por ejemplo, páginas de datos de flash), con el fin de modificar el contenido de compartimento (operaciones de inserción, actualización o borrado), los compartimentos son movidos a una cache. Los compartimentos correspondientes en el dispositivo flash pueden ser marcados ahora como no válidos (libres). En el caso de un SSD, esto puede tomar la forma de una orden TRIM.

Según otras realizaciones de la invención, se facilitan métodos para generar bloques de borrado libres. En cualquier tiempo dado, un bloque de borrado puede tener una combinación de datos válidos y no válidos. Para liberar un bloque de borrado, todos los datos válidos deben ser movidos de dicho bloque. Hay dos mecanismos que pueden ser usados para llevarlo a cabo. Uno es usar las lecturas aleatorias generadas por el algoritmo de indexación para leer más (de lo que requiere el algoritmo de indexación) con el fin de liberar un bloque de borrado. Dado que el algoritmo de indexación tiende a generar lecturas aleatorias, con el tiempo todos los bloques de borrado son leídos eventualmente y recogidos en páginas vacías. Por ejemplo, si el bloque de borrado conteniendo la lectura tiene algunas páginas libres, y algunas páginas válidas, entonces el algoritmo puede elegir leer en todo el bloque de borrado y poner todas las páginas válidas en la cache. Esto tiene el efecto de liberar dicho bloque de borrado para un borrado subsiguiente y posterior escritura.

Alternativamente, por ejemplo, si dicho proceso de lectura aleatoria no es suficientemente rápido, se puede usar un proceso de rebusca separado (por ejemplo, hilo) para leer bloques de borrado, y poner las páginas válidas en la cache para coalescencia con otro bloque de borrado.

Cuando la cache se llena, hay que quitar entradas. Se recoge un conjunto de entradas de cache que serán escritas secuencialmente en un conjunto contiguo de páginas parciales (si el dispositivo flash permite escrituras de página parcial), múltiples páginas, y/o uno o más bloques de borrado. Cuando las entradas de cache se escriben en el dispositivo flash, la tabla de indirección es actualizada, de modo que el algoritmo de indexación todavía ve las entradas en una dirección lógica fija.

B. Operaciones de indexación

Ahora se describirán varias realizaciones de la invención utilizando las figuras acompañantes 1-6 para ilustrar varias operaciones de indexación realizadas según la presente invención. Las figuras 7-8 ilustran dos métodos de generar bloques de borrado libres para la utilización eficiente del medio de almacenamiento (por ejemplo, memoria flash). Estas realizaciones pretenden ser ilustrativas y no limitativas.

La figura 1 es una vista general de varias operaciones de indexación que utilizan una tabla de traducción de compartimentos 17 y cache 23 según una realización de la invención. En la parte superior de la figura 1 se representan tres operaciones de índice 12-14 como entradas alternativas a una función de consulta 15 y una función de traducción 16. Una primera operación de índice 12 es “consultar clave” para devolver datos satélites de (una entrada de registro) para la clave. Una segunda operación de índice 13 es “actualizar datos satélites para clave” para actualizar (modificar) la entrada de registro para la clave. Una tercera operación de índice 14 es “insertar clave

nueva" para insertar una nueva entrada de registro. Otra operación de índice, borrar, no se representa en la figura 1, pero se describe más adelante con respecto a la figura 5.

Las tres operaciones de índice realizan primero una función de consulta 15, donde se usa alguna función de la clave f(key) para generar un índice, aquí un identificador de compartimento lógico que soporta (por ejemplo, acelera) una consulta de tabla hash. El identificador de compartimento (índice) es introducido a una función de traducción 16 donde alguna función del identificador de compartimento lógico f(index) genera una posición de compartimento físico en la memoria flash. La función de traducción es implementada por una tabla de traducción de compartimentos 17, que es un mapa del identificador de compartimento lógico (proporcionado por el algoritmo de indexación) a una posición de memoria flash deseada (posición de compartimento físico en flash). Un diccionario (índice) almacenado en memoria flash 26 puede incluir registros que mapean una clave de consulta (por ejemplo, nombre de objeto) a datos satélites (por ejemplo, indicador de posición al objeto almacenado en disco).

A continuación, dependiendo de cuál de las tres operaciones de indexación se esté realizando (consulta, actualización o inserción), se lleva a cabo uno o varios pasos representados en la mitad inferior de la figura 1.

Para una operación de consulta 18, se lee 30 la entrada de compartimento identificada por la función de traducción del compartimento deseado 22 en memoria flash, dejando a un lado la cache (por ejemplo, si el compartimento deseado está almacenado en cache, se puede leer de la cache 23 más bien que de la memoria flash 26).

Para una operación de actualización 19, la entrada de compartimento identificada por la función de traducción (la entrada de compartimento original) se lee 30 de un compartimento deseado 22 en el bloque de borrado 21a de la memoria flash (o cache), el compartimento es actualizado y movido 32 a cache, y en una escritura posterior 24 una pluralidad de entradas de compartimento de cache son leídas secuencialmente a un conjunto contiguo de páginas parciales, múltiples páginas y/o bloques de borrado (por ejemplo un nuevo bloque de borrado 21b) en memoria flash. El proceso actualiza 33 el estado de todos los compartimentos movidos en flash a datos no válidos (por ejemplo, libres o disponibles para una operación de recorte).

Para una operación de inserción 20, se lee de nuevo un compartimento deseado de flash y se mueve 34 una entrada de compartimento modificada a cache, de nuevo para una escritura secuencial posterior 24 a una posición nueva en la memoria flash.

La figura 1 representa esquemáticamente una cache 23 para recoger una pluralidad de entradas de compartimento, antes de realizar una escritura secuencial 24 de la colección de entradas de compartimento de cache a compartimentos de memoria flash contiguos. En una realización, se usa una operación de rebusca 25 para crear bloques de borrado libres; el proceso incluye almacenar cualesquiera compartimentos válidos (del bloque de borrado) en cache durante el proceso de rebusca y reasignar el bloque de borrado flash como libre.

Siguiendo una explicación de las nuevas estructuras de datos ilustradas en la figura 2, las operaciones de indexación referenciadas en la figura 1 se describirán más específicamente con respecto a los diagramas de flujo de las figuras 3-6.

C. Estructuras de datos

La figura 2 ilustra varias realizaciones de estructuras de datos útiles en la presente invención. Se entiende que tales estructuras de datos son ilustrativas y no limitativas.

La figura 2a ilustra una realización de una tabla de traducción de compartimentos (BTT) 300 para traducir un índice de compartimento lógico (generado por el algoritmo de indexación) a una dirección de compartimento flash físico. Se representa una entrada de tabla BTT que tiene tres campos: válido 301; dirección de compartimento flash físico 302; y estado de compartimento extendido 303. La granularidad de dirección de compartimento es el tamaño de escritura mínimo del dispositivo flash, a saber, una escritura de página parcial (por ejemplo, para SLC NAND) o una escritura de página (por ejemplo, para MLC NAND). BTT es mapeado de 1:1 de entradas de compartimento lógico a físico. La tabla permite la reorganización de las asignaciones de compartimento flash para mayor rendimiento aleatorio (lecturas aleatorias y escrituras aleatorias por el algoritmo de indexación). Se puede añadir información de estado adicional a la BTT en el tercer campo para permitir la aceleración del algoritmo.

La figura 2b representa una realización de una tabla de compartimentos válidos (BVT) 305. Esta tabla rastrea qué compartimentos físicos en flash son válidos con el fin de gestionar la rebusca de compartimentos a bloques para recorte. Por ejemplo, un campo 306 etiquetado válido puede ser una serie de bits compacta (1 bit/compartimento). El tamaño de la BVT es el número total de entradas de compartimento flash, del que solamente un subconjunto es usada por la BTT.

La figura 2c ilustra una realización de compartimento flash 309 que tiene múltiples registros 310, 311, 312 … incluidos en el compartimento, junto con un indicador BTT inverso 313 (un autoíndice a la tabla de traducción de compartimentos 17). Así, cada compartimento contiene un conjunto de uno o más registros y un indicador inverso

para actualizar la BTT cuando compartimentos flash (por ejemplo, páginas) son insertados, movidos o borrados. Cada elemento del compartimento (registro o indicador) puede tener contenido redundante añadido, tal como bits ECC adicionales, para mejorar la fiabilidad individual de las estructuras de datos y aumentar de forma significativa la vida útil de los dispositivos de almacenamiento. Por ejemplo, un campo de número de secuencia opcional puede ser añadido al compartimento flash 309 para realizar verificación de consistencia de datos durante eventos de fallo de potencia; también se puede facilitar otros señalizadores de optimización.

Dado que el tamaño de registro es pequeño con relación al tamaño de compartimento, esto proporciona una oportunidad (opcional) de implementar información adicional de recuperación de errores en base a registros individuales. Esta característica opcional mejoraría la fiabilidad general de la solución incrementando el número de errores y fallos de bits que pueden ser corregidos y así aumentaría la duración operativa efectiva de la tecnología de almacenamiento subyacente.

La figura 2d representa un ejemplo de un dispositivo flash NAND SLC 315 conteniendo múltiples bloques de borrado 316 (1 a M). Cada bloque de borrado incluye múltiples páginas 317 (1 a N). En este ejemplo, cada página es 4 KB y cada página incluye múltiples compartimentos 318 (1 a B), siendo cada compartimento de 1 KB. En este ejemplo, el dispositivo soporta escrituras de página parcial.

Un compartimento representa un tamaño de escritura mínimo del dispositivo flash. Normalmente, un compartimento sería una página. Si están permitidas las escrituras de página parcial, entonces se puede proporcionar uno o más compartimentos por página flash, tal como un dispositivo SLC NAND de cuatro páginas parciales que soporte cuatro compartimentos por página.

Se facilitan múltiples páginas flash por bloque de borrado. Hay múltiples bloques de borrado por dispositivos flash, y cada bloque es borrado individualmente.

El subsistema flash normal consta de múltiples dispositivos flash. Los dispositivos flash NAND son escritos secuencialmente una vez por página (o página parcial) dentro de un bloque dado entre operaciones de borrado, con múltiples bloques disponibles para escritura y lectura simultáneas.

D. Diagramas de flujo de proceso

La figura 3 ilustra una realización de un proceso de operación de consulta para verificar la presencia de una clave y devolver datos satélites asociados. En el paso uno 41, se introduce una clave de consulta a una función de consulta. En el paso dos 42, la función de consulta f(key) genera un identificador de compartimento lógico que soporta (por ejemplo, acelera) una consulta de tabla hash. El identificador de compartimento lógico es introducido a una función de traducción, que en el paso tres 43 es mapeada a una posición de memoria flash (compartimento físico), mediante la tabla de traducción de compartimentos (BTT) 17. En el paso cuatro 44, el compartimento deseado en memoria flash es leído 45a de la memoria flash, a no ser que el compartimento esté almacenado en cache, en cuyo caso puede ser leído 45b de la cache 23. En el paso seis 46, los datos satélites (registro) para la clave son devueltos al algoritmo de indexación.

La figura 4 representa una realización de un proceso de operación de inserción. Un primer paso 71 introduce una clave a la función de consulta. En el paso dos 72, la función de consulta f(key) genera un índice, aquí un identificador de compartimento lógico. En el paso tres 73, el identificador de compartimento es introducido a una función de traducción que mapea el identificador de compartimento a una posición de compartimento físico de memoria flash donde deberá producirse la inserción, utilizando la tabla de traducción de compartimentos (BTT) 17. En el paso cuatro 74, el proceso de inserción recibe la posición de compartimento deseado de la función de traducción. En el paso cinco 75, el proceso de inserción lee el compartimento deseado 22 de un bloque de borrado 21a de la memoria flash 75a, o de la cache 75b. En el paso seis 76, el proceso de inserción inserta la entrada de registro al compartimento deseado y escribe el compartimento modificado en la cache. En el paso siete 77, múltiples entradas de compartimento (incluyendo el compartimento deseado modificado) son leídas de la cache 73 por el proceso de inserción. En el paso ocho 78, el proceso de inserción escribe el compartimento deseado modificado y otros compartimentos leídos de la cache a nuevas posiciones (páginas en el bloque de borrado 21b) en flash 26. En el paso nueve 79, el proceso de inserción actualiza la tabla de traducción de compartimentos 17 con las nuevas posiciones para todos los compartimentos movidos de la cache a la flash 79a, y también actualiza las entradas de compartimento válidas en BVT 79b para todos los compartimentos movidos. En el paso diez 80, el proceso de inserción marca las entradas de cache movidas leídas solamente (disponibles). En el paso once 81, el proceso de inserción marca los compartimentos flash originales (ahora movidos a un nuevo bloque de borrado) como libres.

La figura 5 ilustra una realización de un proceso de operación de borrado. En un primer paso 91, se proporciona una clave a una función de consulta. En el paso dos 92, la función de consulta f(key) genera un índice, aquí un identificador de compartimento lógico. En el paso tres 93, el identificador de compartimento es proporcionado a la función de traducción, que utiliza la tabla de traducción de compartimentos 17 para mapear el identificador de compartimento a una posición de compartimento físico de memoria flash. En el paso cuatro 94, el proceso de borrado recibe la posición de memoria flash. En el paso cinco 95, el compartimento deseado es leído de la flash. En

el paso seis 96, el proceso borra la entrada de registro original en el compartimento y escribe el compartimento modificado (con la entrada borrada) en la cache 23. En el paso siete 97, un grupo (colección) de compartimentos es leído de la cache. En el paso ocho 98, el compartimento deseado actualizado y otros compartimentos leídos de la cache 23 son escritos secuencialmente en un conjunto contiguo de páginas libres en la flash. En el paso nueve 99, el proceso de borrado actualiza la tabla de traducción de compartimentos con las nuevas posiciones en flash para todos los compartimentos movidos 99a, y actualiza su estado válido en la BVT 99b. En el paso diez 100, el proceso de borrado marca las entradas de cache como leídas solamente. En el paso once 101, el proceso de borrado marca los compartimentos flash originales ahora movidos a una posición nueva en flash como libres.

La figura 6 ilustra una realización de un proceso de operación de actualización para modificar un registro en un índice almacenado en memoria flash. En un primer paso 51, se proporciona una clave como entrada a una función de consulta. En el paso dos 52, la función de consulta f(key) genera un índice, aquí un identificador de compartimento lógico. El identificador de compartimento es introducido a una función de traducción. En el paso tres 53, la función de traducción mapea el identificador de compartimento a un compartimento físico en memoria flash donde deberá tener lugar la actualización, utilizando la tabla de traducción de compartimentos 17. En el paso cinco 55, el compartimento deseado es leído de la flash 55a o de la cache 55b. En el paso seis 56, después de actualizar la entrada, el compartimento actualizado se escribe en la cache 23. En el paso siete 57, un grupo de compartimentos es leído de la cache 23 y en el paso ocho 58, se escribe secuencialmente de la cache a una posición nueva en la memoria flash 26. En el paso nueve 59, el proceso de actualización actualiza la tabla de traducción de compartimentos 17 con las nuevas posiciones para todos los compartimentos movidos 59a, y actualiza su estado válido en la BVT 59b. En el paso diez 60, el proceso de actualización marca las entradas movidas como leídas solamente en la cache 23 (y así disponibles para ser sobreescritas). Finalmente, en el paso once 61, el proceso de actualización marca los compartimentos flash originales, ahora movidos a una posición nueva, como libres (disponibles).

La figura 7A ilustra una realización de un proceso para generar bloques de borrado libres, donde una lectura de demanda (generada por una operación de indexación anterior tal como una consulta, inserción o modificación) lee compartimentos adicionales en el mismo bloque de borrado (como el compartimento deseado). En la figura 7A, el proceso se ilustra con una petición de actualización. En el paso uno 111, se proporciona una clave a una función de consulta. En el paso dos 112, la función de consulta f(key) genera un índice, aquí un identificador de compartimento lógico. En el paso tres 113, el identificador de compartimento es mapeado a una posición de compartimento deseado físico en flash. En el paso cuatro 114, el proceso de actualización y rebusca recibe la posición de memoria flash deseada. En el paso cinco 115, el proceso identifica todos los compartimentos válidos en el mismo bloque de borrado que el compartimento deseado. En el paso seis, 116, el proceso de actualización lee el compartimento deseado y todos los compartimentos válidos identificados del bloque flash conteniendo el compartimento deseado. En el paso siete 117, el proceso actualiza la entrada de registro en el compartimento deseado y escribe todos los compartimentos válidos del bloque flash en la cache 23. En el paso ocho 118, el proceso de actualización lee un grupo de bloques de cache. En el paso nueve 119, el proceso de actualización escribe el compartimento deseado actualizado y otros compartimentos leídos de la cache 23 en la flash 26. En el paso diez 120, el proceso de actualización actualiza la tabla de traducción de compartimentos 17 con las nuevas posiciones para todos los compartimentos movidos (escritos de la cache al nuevo bloque de borrado 21b en flash) 120a, y actualiza las entradas de compartimento en la BVT 120b. En el paso once 121, el proceso de actualización marcas las entradas de cache ahora trasnochadas como leídas solamente. En el paso doce 122, el proceso de actualización marca el bloque flash original (todos los compartimentos en el bloque deseado) como libre.

La figura 7B ilustra una realización particular del proceso de lectura aleatoria recién descrito para generar bloques de borrado libres.

En esta realización, un algoritmo de indexación de hashing de desplazamiento 125 genera compartimentos lógicos 126. El tamaño de compartimento lógico visto por el algoritmo de indexación está unido al tamaño de bloque de borrado flash con el fin de hacer compatible el algoritmo de indexación y la memoria flash. Estos compartimentos serán leídos aleatoriamente como resultado de escrituras y actualizaciones de índice.

Una tabla de traducción de compartimento (indirección) 127 traduce un índice de compartimento lógico a una posición de compartimento físico de dispositivo flash. Esta tabla de indirección permite que el algoritmo de indexación opere aleatoriamente, para lecturas, escrituras y actualizaciones, y que todavía realice grandes escrituras secuenciales al nivel de dispositivo flash. Preferiblemente, la tabla de indirección se almacena en memoria persistente, pero puede ser reconstruida si es necesario si se almacena en memoria volátil.

La salida de la tabla de indirección, a saber la posición de compartimento físico de dispositivo, se facilita como entrada a una cache de compartimento completamente asociativa 128. En esta realización, si el contenido de un bloque de borrado vacío fifo 129 está debajo de una marca de agua alta Q, entonces se lee todo el bloque de borrado (conteniendo el compartimento de 4 KB deseado).

Los bloques de borrado alojan compartimentos lógicos, siendo una configuración normal un bloque de borrado que contiene 16 compartimentos lógicos de 4 KB. El dispositivo físico está configurado para una carga, por ejemplo, de

90%, lo que significa que 90% de los compartimentos está en uso. Se usa puesta en cache y victimización (evicción) para empaquetar (concentrar) compartimentos lógicos en la memoria flash de modo que la mayor parte del 10% de los compartimentos restante se concentre en bloques de borrado libres.

La victimización de cache (proceso de evicción) toma 16 compartimentos, recogidos en cache, y escribe los 16 compartimentos de cache en un bloque de borrado libre 130. Dado que los bloques de borrado son tocados aleatoriamente por las operaciones de lectura aleatorias, las operaciones de lectura pueden ser usadas para generar bloques de borrado libres. El uso de una función de hash criptográfico para generar los identificadores de compartimento lógicos aumentará la naturaleza aleatoria de las operaciones de lectura y así mejorará la generación de lecturas aleatorias de bloques de borrado libres.

Las figuras 8A y 8B ilustran un proceso de rebusca alternativo para generar bloques de borrado libres. Este proceso de rebusca no es una parte de ninguna operación de indexación. Más bien, se implementa como parte de una capa de gestión de dispositivo de nivel inferior. En este proceso, un grupo (algunos o todos) de compartimentos físicos en un bloque de borrado flash son leídos directamente de la flash, y la tabla de compartimentos válidos 27 se usa para determinar qué compartimentos del bloque de borrado son válidos.

Como se ilustra en la figura 8A, en el paso uno 220, un proceso de rebusca 25 lee un bloque de borrado completo 21a. En el paso dos 222, el proceso de rebusca usa la tabla de compartimentos válidos 27 para identificar todos los compartimentos de los leídos que son válidos. En el paso tres 224, para cada compartimento válido, el identificador de compartimento lógico es extraído del compartimento. En el paso cuatro 226, los compartimentos válidos son almacenados en la cache 23, cada uno indexado por su identificador de compartimento lógico.

La figura 8B representa un ejemplo donde, en el paso uno, el proceso de rebusca 25 lee compartimentos [94, 97] inclusive. En el paso dos, el proceso determina en 95 y 96 qué compartimentos son válidos. Los compartimentos válidos se representan en la tabla de compartimentos válidos designados con un “1”, y los compartimentos no válidos con un “0”. En el paso tres, los identificadores de compartimento lógicos para los compartimentos 95 y 96, a saber las etiquetas 23 y 49 respectivamente, son extraídos de los compartimentos. En el paso cuatro, las dos etiquetas y sus respectivos compartimentos 95 y 96 son insertados en la cache usando sus respectivas etiquetas 23, 49 como el índice.

E. Visión a nivel de pila e implementación

Otro ejemplo más específico de la invención se describirá ahora con respecto a las figuras 9-16.

La figura 9 representa una vista o pila de seis capas 200 para ilustrar una implementación de la presente invención en la que una capa de adaptación flash 207 adapta una vista de perfil de uso IO deseada por un algoritmo de indexación 203, que es una vista muy diferente de la deseada por el dispositivo físico de memoria flash 211. En el nivel superior 201 se facilita un diccionario (índice) de registros, para el que se requieren algunas operaciones de indexación 204 (consulta, borrado, inserción y modificación de un registro). Una capa de algoritmo de indexación 203 implementa el diccionario con uno o más algoritmos de indexación, por ejemplo, un algoritmo de hashing de desplazamiento de Cuckoo. El algoritmo de indexación tiene una visión de cómo las claves al índice serán almacenadas por una capa de persistencia de índice 205. La visión de indexación es una visión lógica, que especifica posiciones de dirección lógicas. La visión asume además que habrá acceso uniforme al índice con respecto al tamaño, la alineación y la temporización, y que el índice se almacena en un almacenamiento mutable (estable).

La capa de persistencia de índice 205 presentará operaciones de compartimento lógico 206 para leer y escribir, a compartimentos físicos que almacenan los registros del índice. Estas operaciones de compartimento lógico 206 son presentadas a una capa de adaptación flash 207, que, como se ha descrito previamente, traduce los compartimentos lógicos (del proceso de indexación) a posiciones de compartimento físico en el dispositivo de almacenamiento flash. La capa de adaptación flash adapta así la visión y el perfil de uso IO deseado por el algoritmo de indexación anterior, a la visión muy diferente deseada por el dispositivo físico de almacenamiento (memoria flash 211) debajo. Aquí las operaciones de compartimento físico 208 incluyen lecturas aleatorias y escrituras agregadas (bloque secuencial), que constituyen un modelo no uniforme de acceso a compartimento. Las operaciones de compartimento físico en este ejemplo pueden incluir además órdenes de recorte.

Las operaciones de compartimento físico son implementadas por una capa de gestión de dispositivo 209 que rastrea y coordina los recursos en el dispositivo flash físico. Estas operaciones de dispositivo físico 210 incluyen aquí lecturas aleatorias, grandes escrituras secuenciales, y órdenes de recorte.

La capa de dispositivo físico 211 se caracteriza por su lectura y escritura no uniformes e inmutabilidad con respecto al tamaño, la alineación y la temporización. Ejemplos de tales dispositivos físicos incluyen flash en raw, cambio de fase, un SSD, y/o flash con un sistema de archivo flash residente en el dispositivo.

La presente invención permite mejoras opcionales adicionales debajo de la capa de gestión de dispositivo tales

como:

* El modelo de recorte de compartimento (recorte de página fino) y compartimentos de seguimiento dentro de una página permiten una mejor gestión de bloque de borrado si se incorporan directamente a un sistema de archivo flash de un SSD o dispositivo de almacenamiento equivalente.

* El mapeado de compartimentos sobre páginas flash es una abstracción. Los compartimentos se podrían mapear a páginas parciales para SLC NAND para aumentar la duración de dichos dispositivos minimizando la cantidad de datos escritos en la flash por cada cambio. Los compartimentos también se pueden mapear sobre múltiples páginas flash si esto es beneficioso para el rendimiento general del sistema.

La figura 10 representa un ejemplo de un registro de índice. El registro 140 es de 32 bytes en total, incluyendo un primer campo de 20 bytes 141 para almacenar una huella dactilar (clave). Una huella dactilar es preferiblemente un compendio de hash criptográfico del contenido de datos, por ejemplo, un algoritmo hash SHA-1. Para facilitar la ilustración, más bien que teclear la huella dactilar en dígitos hex como “AB92345E203 …”, una huella dactilar individual se designará en las figuras 11-14 con una sola letra mayúscula como P, Q, R, S, T. Estas letras mayúsculas también actuarán como un proxy para todo el registro, de nuevo para simplificación a efectos de ilustración. Los campos del registro también incluyen un campo de recuento de referencias de dos bytes 142, un campo de dirección de bloque físico de cinco bytes 143, un campo de señalizadores de un byte 144, y un campo variado de cuatro bytes 145. El campo PBA 143 contiene un indicador a la dirección de bloque físico de los datos almacenados en disco, para la huella dactilar designada 141. El recuento de referencias rastrea el número de referencias a los datos almacenados en disco.

Según una realización de la invención, la huella dactilar 141 procedente del registro de índice se usa como una clave de entrada a la función de consulta f(key) descrita previamente (figura 1). En este ejemplo, la función f(key) incluye un conjunto de cuatro funciones hash H_0, H_1, H_2 y H_3. Por lo general, se puede usar cualquier conjunto de dos o más funciones hash. La función hash H_x mapea la huella dactilar a un rango [0, N-1] inclusive, donde N es el tamaño de la tabla hash. Dado que en este ejemplo las huellas dactilares propiamente dichas son hashes, se puede extraer BitFields para generar la familia siguiente de cuatro valores hash:

$H_0(x) = x<0:31> \bmod N$

$H_1(x) = x<032:63> \bmod N$

$H_2(x) = x<064:95> \bmod N$

$H_3(x) = x<096:127> \bmod N$

La anchura de BitField extraída es mayor o igual a $\log_2 (N)$. Se puede usar cualquier combinación de bits disjuntos, con sujeción a la restricción $\log_2 (N)$. Como se ilustra en la figura 10, solamente la huella dactilar en el primer campo 141 es sometida a hash, para formar la clave. El contenido restante (campos 142-145) del registro 140 incluye un valor o carga.

La figura 11 ilustra un ejemplo de un algoritmo de indexación de hashing de desplazamiento conocido como hashing de Cuckoo. Para facilitar la ilustración, se usan dos funciones solamente. La figura 11A representa una rejilla 2x3 en la que la huella dactilar P genera valores hash 2 y 5 a partir de las funciones $H_0(x)$ y $H_1(x)$, respectivamente, mientras que la huella dactilar Q genera valores hash 1 y 3 a partir de estas mismas funciones. El algoritmo de hashing de Cuckoo seleccionará de entre los dos valores hash alternativos para poner P y Q en una de las siete ranuras etiquetadas 0-6 (figura 11 B). P puede ir en una de dos posiciones, 2 o 5, y Q puede ir en una de dos posiciones, 1 o 3. El algoritmo pone Q en la ranura vacía más baja 1 y P en la ranura 2, como se representa en la figura 11C. Aunque en este ejemplo el depósito de registro se denomina una ranura que contiene un registro, se deberá entender que la invención no se limita a ello; los algoritmos de indexación también ven un compartimento, que contiene múltiples registros, como un depósito. Aquí se usa una sola ranura de registro para simplificar la explicación.

Ahora, se facilita otra huella dactilar R que genera valores hash de 1 y 2 a partir de las mismas funciones hash (véase la tabla en la figura 11 D). El algoritmo de hashing pondrá R en la posición izquierda, a saber la ranura 1, desplazando la entrada actual Q (figura 11E). Q será movido ahora a la otra posición opcional especificada por $H_1(Q)$, a saber la posición 3. El algoritmo seguirá desplazando registros hasta que cada registro aterrice en una ranura vacía.

En este ejemplo, para llevar a cabo la operación de “insertar R”, el algoritmo de indexación genera las peticiones de lectura y escritura siguientes:

Lectura 1 (obtiene Q)

Lectura 2 (obtiene P)

Escritura 1 (escribir R)

Lectura 3 (comprobación de validez)

Escritura 3 (Q)

Las dos primeras lecturas se usan para validar que R ya no está presente en el índice. La comprobación de validez (lectura 3) determina si la ranura número 3 está vacía; si es así, entonces se puede escribir Q en la ranura 3 y el algoritmo se hace como si no se reescribiese ninguna entrada en la ranura 3. Si la ranura 3 no estuviese vacía, entonces la entrada actual en la ranura 3 tendría que ser movida a otra ranura. El contenido de la ranura 3 es conocido si se tiene un mapa de bits; de otro modo, hay que leer la entrada en la ranura 3 para determinar su estado. Cada entrada contiene un bit válido que indica si dicha entrada es válida. Válido significa que está en uso (y el ocupante actual de la posición tiene que ser desplazado). No válido significa que la posición está vacía, y el registro procesado se puede escribir allí. El contenido de los bits válidos también puede ser almacenado en un mapa de bits separado, a costa de algo de memoria.

El algoritmo de hashing de Cuckoo es recursivo, porque sigue escribiendo sobre entradas, desplazando el contenido previo, hasta que aterriza en una entrada vacía. En la práctica, este proceso raras veces excede de un desplazamiento.

El algoritmo de indexación tiene operaciones de registro tanto de compartimento como individuales. El algoritmo de indexación se ha descrito anteriormente (en la figura 11) como poner un registro en un depósito (ranura), pero el algoritmo de indexación entiende que los registros también pueden estar agregados a compartimentos, es decir, compartimentos que contienen múltiples registros. Así, el ejemplo anterior no es limitador y tiene la finalidad de ilustrar las operaciones de registro en general.

Como se ha descrito previamente, dado que la lectura y la escritura de registros individuales no es eficiente para la memoria flash, los registros individuales son agregados a compartimentos. La figura 12 ilustra cuatro compartimentos, conteniendo cada uno dos o más registros, es decir, el compartimento B_0 con las posiciones de registro 0 y 1, B_1 con las posiciones de registro 2 y 3, B_2 con las posiciones de registro 4 y 5, y B_3 con las posiciones de registro 6 y x. El tamaño de compartimento es una función de (y preferiblemente es igual a) el tamaño de escritura mínimo dictado por el dispositivo flash, es decir, escritura de página completa o escritura de página parcial. Un tamaño de compartimento normal puede ser 4 KB. No se requiere ordenación específica de registros dentro del compartimento: en todo el compartimento se busca un registro válido durante la operación de consulta, de modo que el registro pueda ser insertado en cualquier punto dentro del compartimento. Al desplazamiento, según el algoritmo de hashing de Cuckoo, una entrada en el compartimento puede ser desplazada aleatoriamiente. El algoritmo de indexación escribe así compartimentos lógicos en lo que parece ser posiciones aleatorias, una cada vez, que eventualmente son agregadas por la capa de adaptación flash a escrituras contiguas (secuenciales) físicamente más grandes en el dispositivo flash.

La figura 13 ilustra un ejemplo de una entrada de compartimento 160. Un tamaño de compartimento de 4 KB se basa en el tamaño de escritura mínimo de dispositivo subyacente, aquí una página de 4 KB. El compartimento de 4 KB incluye un primer campo de 4 bytes 161 que especifica el número de registros en la entrada de compartimento. Un campo de etiqueta de 4 bytes 162 especifica el identificador de compartimento lógico. Este identificador (etiqueta) es una dirección lógica, no una física. La tabla de traducción mapea la dirección de compartimento de algoritmo (ABA) a una dirección de compartimento flash FBA. La cache opera como una cache virtual (en terminología CPU), con cada línea (entrada) de cache identificada por una etiqueta, una ABA en este caso. Cuando el algoritmo pide registros, todo lo que sabe al pasar a través de la cache es que la ABA pedida está en cache; dónde está mapeada (a la FBA) está en el extremo inferior de la cache (por ejemplo, véase el indicador inverso 313 a la BTT, en la figura 2C). El compartimento incluye un campo 163 para contener una pluralidad de registros R_0, R_1, R_2 …, teniendo cada registro un tamaño de 32 bytes. En este ejemplo, un compartimento de 4 KB contendrá: (4096 – 4 - 4)/32 registros, es decir, aproximadamente 127 registros por compartimento.

La figura 14 es un diagrama esquemático de un dispositivo de memoria flash 164 que ilustra los tamaños relativos de un compartimento, página y bloque de borrado en una realización. El dispositivo flash físico es un chip (paquete) 165 que tiene un tamaño de 2GB. En el chip, hay dos dados (pastillas de silicio) 166a, 167b. En cada dado puede haber 2^14 bloques de borrado, siendo normalmente cada bloque de borrado 167 de 64 KB. Una página 168 es el tamaño mínimo que se puede escribir, aquí 4 KB, y determina el tamaño del compartimento 169, también 4 KB, usado más arriba en la pila (véase la figura 9).

La figura 15 ilustra la selección de componentes según una realización de una capa de gestión de dispositivo (209 en la figura 9) para seguimiento y coordinación de los recursos en el dispositivo flash físico. La figura 15A representa (en la parte superior) una pluralidad de páginas (compartimentos) 170, seguida de un mapa de asignación de página 171 que indica qué páginas son válidas (1 es válido, 0 es no válido). Debajo de esto hay un mapa de recorte

pendiente 172, de las páginas a recortar en el futuro, pero todavía no se ha hecho. La asignación de página y los mapas de recorte pendientes pueden ser usados en varias realizaciones de la invención como se ha descrito previamente, para determinar si un compartimento contiene datos válidos (véase la tabla de compartimentos válidos 27 ilustrada en la figura 1).

La figura 15B ilustra un ejemplo de una tabla de descriptores de bloque de borrado 175, indexada por el índice de bloque de borrado. Cada entrada de descriptor de bloque de borrado 176 incluye una pluralidad de campos, incluyendo el número borrado 177, el número de escrituras parciales 178, el número de lecturas parciales 179, el número de lecturas completas 180, el número de escrituras completas 181, y el número de errores 182. Esta información puede ser usada al generar bloques de borrado libres como se ha descrito previamente en varias realizaciones de la invención.

F. Otras realizaciones

La presente invención puede ser usada para implementar un índice para un sistema de archivo, como el descrito en US número de serie ___, en tramitación y del mismo cesionario, titulada Sistema de archivo, de A. J. Beaverson y P. Bowden, presentada en la misma fecha que la presente solicitud y que reivindica prioridad por la US Provisional número 61/269.633 presentada el 26 de Junio de 2009. Aquí se reivindica prioridad por ambas solicitudes y sus descripciones completas se incorporan aquí por referencia en su totalidad.

Las realizaciones de la invención se pueden implementar en circuitería electrónica digital, o en hardware de ordenador, microprogramas, software o en sus combinaciones. Las realizaciones de la invención se pueden implementar como un producto de programa de ordenador, es decir, un programa de ordenador realizado de forma tangible en un medio legible por ordenador, por ejemplo, en un dispositivo de almacenamiento legible por máquina, para ejecución por, o para controlar la operación de, aparatos de procesado de datos, por ejemplo, un procesador programable, un ordenador, o múltiples ordenadores. Un programa de ordenador puede estar escrito en cualquier forma de lenguaje de programación, incluyendo lenguajes compilados o interpretados, y se puede desplegar en cualquier forma, incluyendo como un programa autónomo o como un módulo, componente, subrutina u otra unidad adecuada para uso en un entorno informático. Un programa de ordenador puede ser desplegado para ejecución en un ordenador o en múltiples ordenadores en un lugar o distribuido a través de múltiples lugares e interconectados por una red de comunicaciones.

Los pasos del método de las realizaciones de la invención pueden ser realizados por uno o más procesadores programables que ejecuten un programa de ordenador para realizar funciones de la invención operando en datos de entrada y generando salida. Los pasos del método también pueden ser realizados por, y el aparato de la invención puede ser implementado como, circuitería lógica de tipo especial, por ejemplo, una FPGA (matriz de puertas programable in situ) o un ASIC (circuito integrado específico de aplicación).

Los procesadores adecuados para la ejecución de un programa de ordenador incluyen, a modo de ejemplo, microprocesadores tanto de tipo general como especial, y uno o más procesadores de cualquier tipo de ordenador digital. Por lo general, un procesador recibirá instrucciones y datos de una memoria de lectura solamente o una memoria de acceso aleatorio o ambas. Los elementos esenciales de un ordenador son un procesador para ejecutar instrucciones y uno o más dispositivos de memoria para almacenar instrucciones y datos. Por lo general, un ordenador también incluirá, o estará acoplado operativamente para recibir datos o transferir datos de/a, o ambos, uno o más dispositivos de almacenamiento masivo para almacenar datos, por ejemplo, magnéticos, discos magnetoópticos o discos ópticos. Los soportes de información adecuados para realizar instrucciones y datos de programa de ordenador incluyen todas las formas de memoria no volátil, incluyendo a modo de ejemplo dispositivos de memoria de semiconductores, por ejemplo, EPROM, EEPROM, y dispositivos de memoria flash; discos magnéticos, por ejemplo, discos duros internos o discos extraíble; discos magnetoópticos; y discos CD ROM y DVD-ROM. El procesador y la memoria puede ser complementados por, o incorporados en, circuitería lógica de tipo especial.

Se ha de entender que se pretende que la descripción anterior ilustre y no limite el alcance de la invención.

REIVINDICACIONES

1. Un método de adaptar un proceso de indexación de acceso uniforme con una memoria flash NAND de acceso no uniforme (26, 235), incluyendo el método:

a) almacenar un diccionario de registros de índice en la memoria de acceso no uniforme (26), incluyendo los registros de índice claves de índice;

b) mantener una tabla de traducción de compartimentos (17, 300) para mapear identificadores de compartimento lógicos a posiciones de compartimento físico de la memoria (26) incluyendo generar un identificador de compartimento lógico por el hashing de desplazamiento de una clave de índice (141) e incluyendo la tabla un mapeado del identificador de compartimento lógico a una posición de compartimento físico (22, 309) de la memoria (26) donde se almacena el registro de índice asociado (310, 311, 312);

c) recoger en cache (23) una pluralidad de compartimentos, donde cada compartimento incluye un conjunto de registros de índice que tiene el mismo identificador de compartimento lógico antes de escribir la colección de compartimentos de la cache en posiciones de compartimento físico contiguas de la memoria (26) como una escritura secuencial; y

d) actualizar la tabla de traducción de compartimentos (17) con las posiciones de compartimento físico para los compartimentos de la colección escrita desde la cache a la memoria (26).

2. El método de la reivindicación 1, incluyendo:

leer uno o más registros de índice secuenciales desde la memoria (26, 23) a la cache;

designar como libres las posiciones físicas en memoria (26) de la que se lee el uno o varios registros de índice.

3. El método de la reivindicación 1, incluyendo:

presentar una pluralidad de posiciones de compartimento físico secuenciales (22, 309) de la memoria como un bloque libre leyendo cualesquiera registros de índice válidos en el bloque a la cache y designar como libres las posiciones físicas de la memoria de la que se leyeron tales registros de índice.

4. El método de la reivindicación 1, donde:

generar una pluralidad de identificadores de compartimento lógicos para la clave de índice, donde la función hashing de desplazamiento selecciona de entre la pluralidad de identificadores de compartimento lógicos generados.

5. El método de la reivindicación 1, donde:

la memoria está limitada por uno o varios de tiempo de acceso a escritura aleatorio, tiempo de acceso de lectura-modificación-escritura aleatorio, escritura secuencial, restricciones de alineación, tiempo de borrado, límites y desgaste de bloque de borrado.

6. El método de la reivindicación 1, donde:

el tamaño de compartimento es una función del tamaño de escritura mínimo de la memoria tal como una página o página parcial.

7. El método de la reivindicación 6, donde:

la memoria tiene un bloque de borrado incluyendo una pluralidad de páginas.

8. El método de la reivindicación 1, incluyendo:

mantener una tabla de compartimentos válidos (271) para rastrear qué posiciones de compartimento físico de la memoria son válidas.

9. El método de la reivindicación 1, donde:

cada posición de compartimento físico (23) de la memoria incluye con el conjunto de registros de índice un autoíndice en la tabla de traducción de compartimentos.

10. El método de cualquier reivindicación precedente, donde:

los registros de índice del compartimento no están ordenados.

11. El método de la reivindicación 1, donde:

la tabla de traducción de compartimentos está almacenada en una memoria persistente.

12. El método de la reivindicación 1, incluyendo:

rastrear el número de posiciones de compartimento físico libres en un bloque de borrado e implementar un proceso para generar un bloque de borrado libre cuando se encuentra un umbral de posiciones de compartimento libres.

13. El método de la reivindicación 1, donde:

el proceso de indexación realiza operaciones de indexación en base a peticiones de que los registros de índice sean insertados, borrados, consultados y/o modificados.

14. El método de la reivindicación 1, donde:

el proceso de indexación presenta operaciones de compartimento lógico para leer y escribir a posiciones de compartimento físico que almacenan los registros de índice.

15. El método de la reivindicación 1, donde:

la memoria flash NAND (26) incluye una pluralidad de bloques de borrado, incluyendo cada bloque de borrado una pluralidad de páginas, e incluyendo cada página una pluralidad de compartimentos.

16. El método de la reivindicación 15, incluyendo:

generar bloques de borrado libres leyendo compartimentos adicionales en la cache en respuesta a operaciones de lectura aleatorias.

17. El método de la reivindicación 15, incluyendo:

realizar un proceso de rebusca (114) para generar bloques de borrado libres leyendo bloques de borrado en la cache.

18. Un producto de programa de ordenador incluyendo un medio de código de programa que, cuando es ejecutado por un procesador, realiza los pasos de método de la reivindicación 1.

19. Un sistema informático incluyendo:

una memoria flash NAND de acceso no uniforme (26) conteniendo un diccionario de registros de índice almacenados en posiciones de compartimento físico de la memoria;

una tabla de traducción de compartimentos (17) que mapea un identificador de compartimento lógico, generado por hashing de desplazamiento de una clave de índice del diccionario, a una posición de compartimento físico de la memoria (26) donde está almacenado un registro de índice (310, 311, 312) asociado con la clave de índice;

una cache (23) para recoger compartimentos, incluyendo cada compartimento un conjunto de registros de índice que tiene el mismo identificador de compartimento lógico a escribir a la memoria;

un medio para acceder a (18) posiciones de compartimento físico de la memoria (26) mapeadas a identificadores de compartimento lógicos suministrados a la tabla de traducción de compartimentos por un proceso de indexación de acceso uniforme;

un medio para escribir (24) secuencialmente una colección de los compartimentos de la cache en posiciones de compartimento físico contiguas de la memoria (26); y

un medio para actualizar (19) la tabla de traducción de compartimentos (17) con las posiciones de compartimento físico para los compartimentos de la colección.

20. El sistema de la reivindicación 19, donde:

la memoria (26) que guarda el índice incluye una capa de dispositivo físico **caracterizado** por acceso de lectura y escritura no uniforme e inmutabilidad con respecto al tamaño, la alineación y la temporización.

21. El sistema de la reivindicación 19, donde:

la memoria flash NAND (26) que guarda el índice incluye una pluralidad de bloques de borrado, cada bloque de borrado incluye una pluralidad de páginas, y cada página incluye una pluralidad de compartimentos.

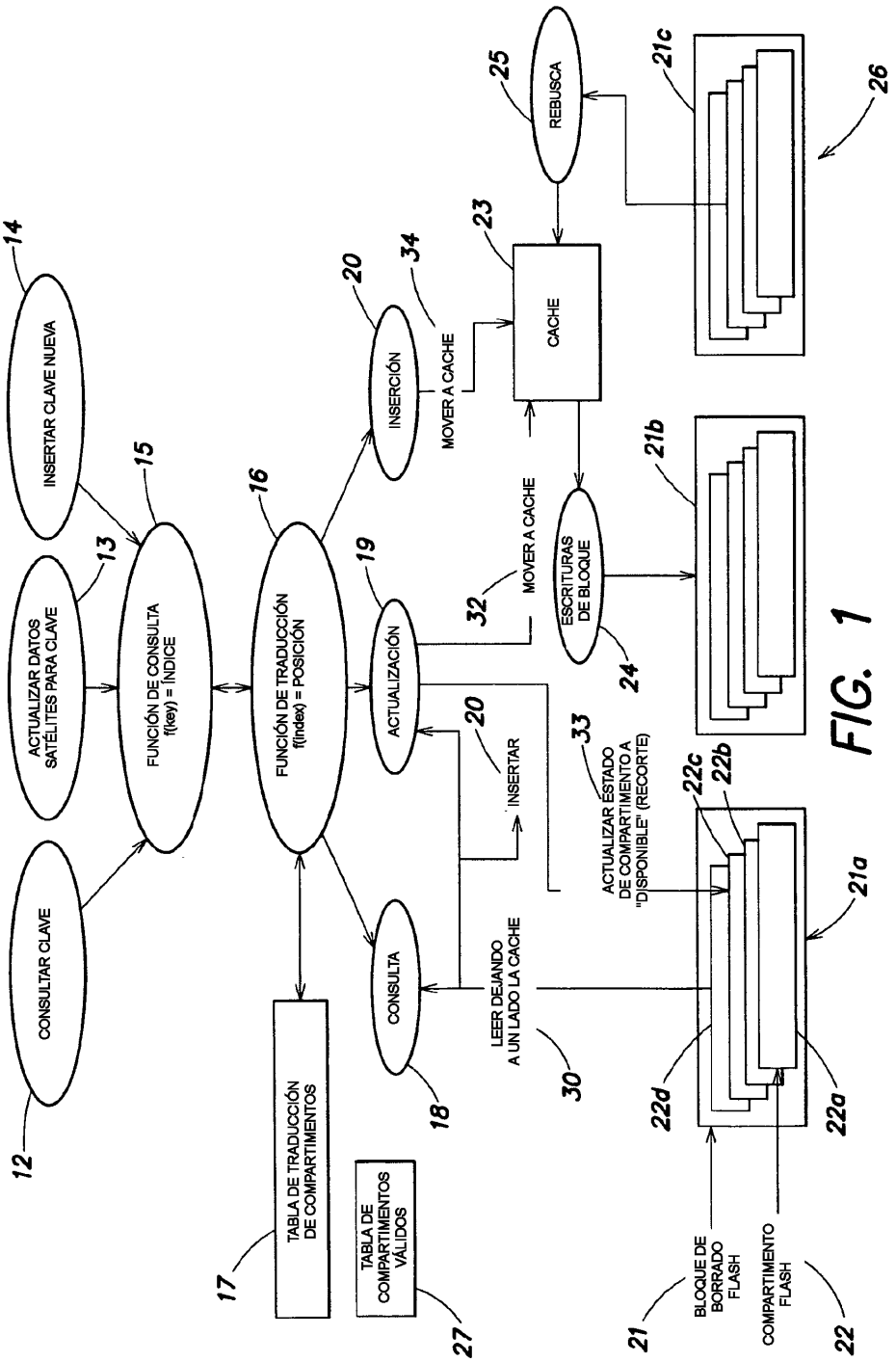
INSERTAR CLAVE NUEVA
ACTUALIZAR DATOS SATÉLITES PARA CLAVE
CONSULTAR CLAVE
FUNCIÓN DE CONSULTA f(key) = ÍNDICE
FUNCIÓN DE TRADUCCIÓN f(índex) = POSICIÓN
INSERCIÓN
MOVER A CACHE
ACTUALIZACIÓN
CONSULTA
INSERTAR
LEER DEJANDO A UN LADO LA CACHE
ACTUALIZAR ESTADO DE COMPARTIMENTO A "DISPONIBLE" (RECORTE)
CACHE
REBUSCA
ESCRITURAS DE BLOQUE
MOVER A CACHE
TABLA DE TRADUCCIÓN DE COMPARTIMENTOS
TABLA DE COMPARTIMENTOS VÁLIDOS
BLOQUE DE BORRADO FLASH
COMPARTIMENTO FLASH
12
13
14
15
16
17
18
19
20
21
21a
21b
21c
22
22a
22b
22c
22d
23
24
25
26
27
30
32
33
34

FIG. 1

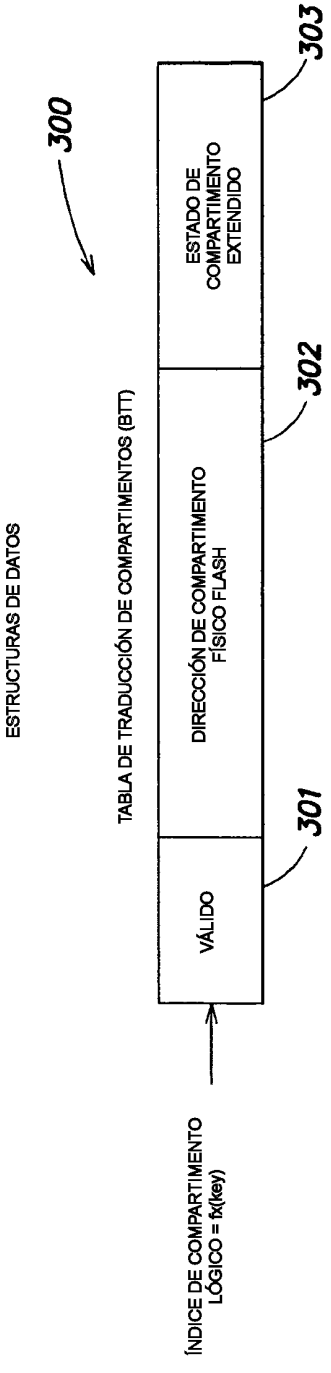
300
ESTRUCTURAS DE DATOS
TABLA DE TRADUCCIÓN DE COMPARTIMENTOS (BTT)
303
ESTADO DE COMPARTIMENTO EXTENDIDO
302
DIRECCIÓN DE COMPARTIMENTO FÍSICO FLASH
301
VÁLIDO
ÍNDICE DE COMPARTIMENTO LÓGICO = fx(key)

FIG. 2A

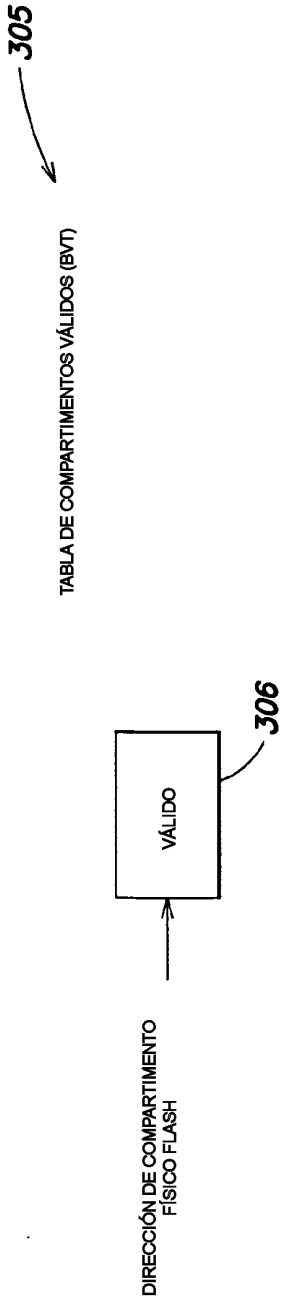
305
TABLA DE COMPARTIMENTOS VÁLIDOS (BVT)
306
VÁLIDO
DIRECCIÓN DE COMPARTIMENTO FÍSICO FLASH

FIG. 2B

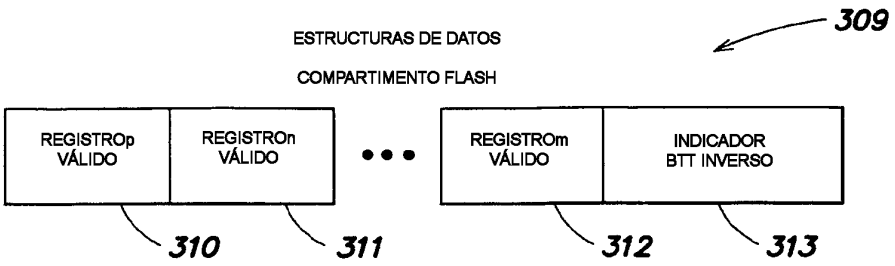
309
ESTRUCTURAS DE DATOS
COMPARTIMENTO FLASH
REGISTROp VÁLIDO
REGISTROn VÁLIDO
REGISTROm VÁLIDO
INDICADOR BTT INVERSO
310
311
312
313

FIG. 2C

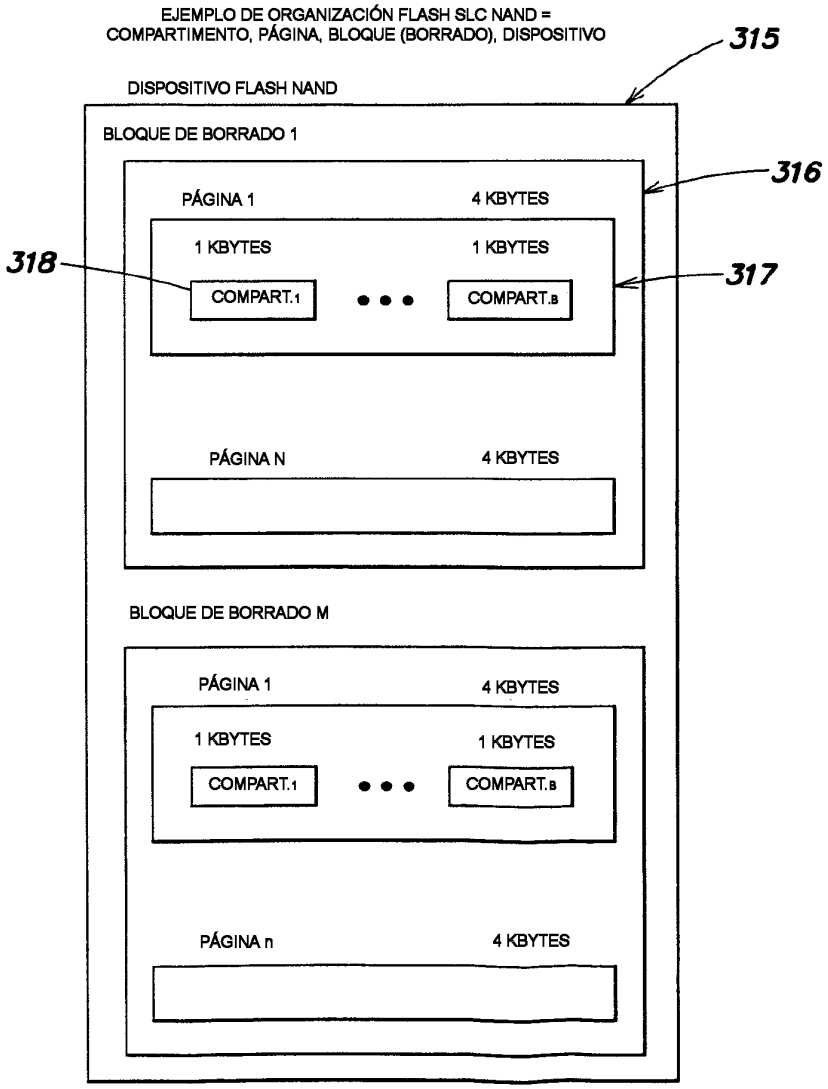
EJEMPLO DE ORGANIZACIÓN FLASH SLC NAND = COMPARTIMENTO, PÁGINA, BLOQUE (BORRADO), DISPOSITIVO
315
DISPOSITIVO FLASH NAND
BLOQUE DE BORRADO 1
316
PÁGINA 1
4 KBYTES
1 KBYTES
1 KBYTES
318
317
COMPART.1
COMPART.B
PÁGINA N
4 KBYTES
BLOQUE DE BORRADO M
PÁGINA 1
4 KBYTES
1 KBYTES
1 KBYTES
COMPART.1
COMPART.B
PÁGINA n
4 KBYTES

FIG. 2D

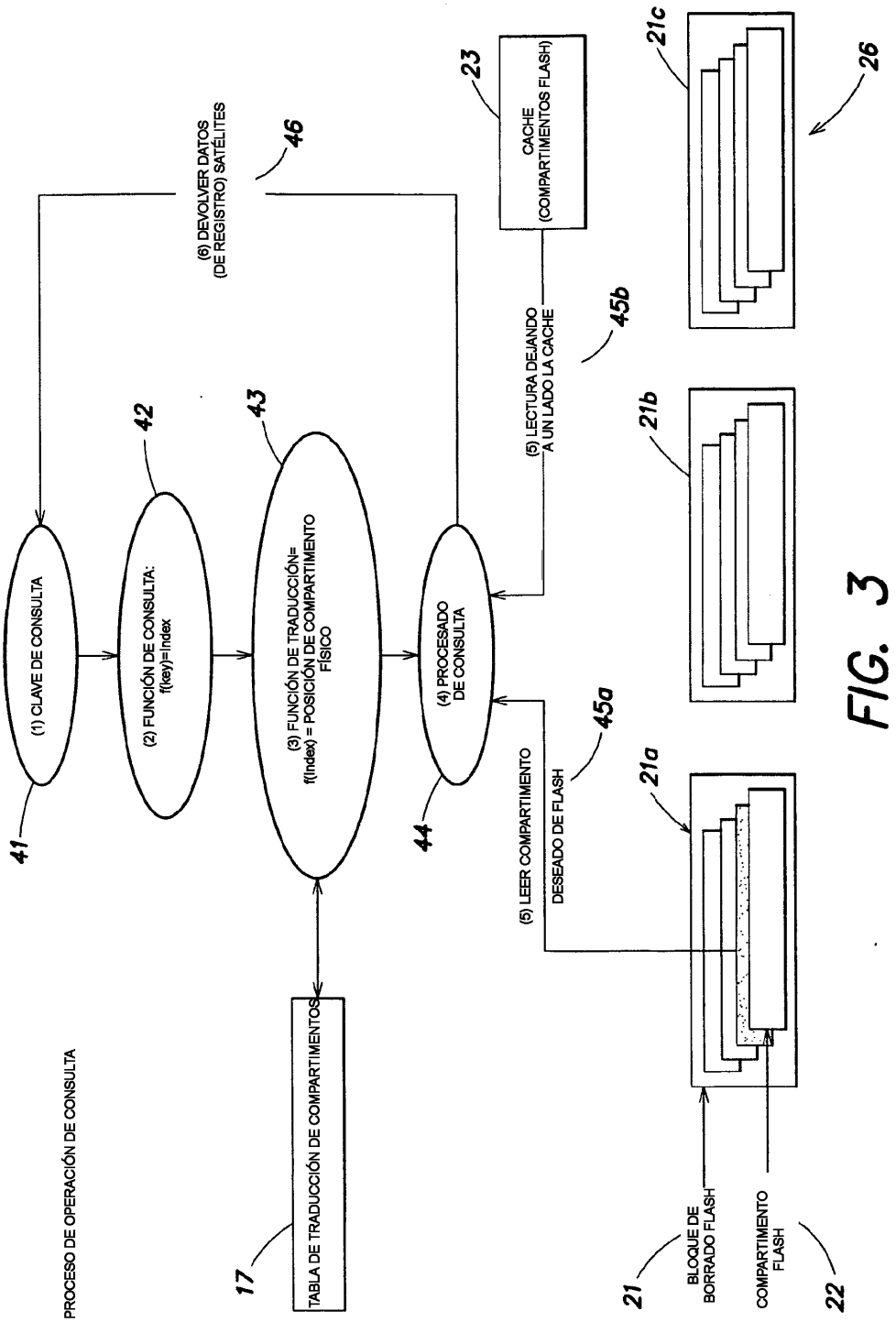
PROCESO DE OPERACIÓN DE CONSULTA
(1) CLAVE DE CONSULTA
(2) FUNCIÓN DE CONSULTA: f(key)=Index
(3) FUNCIÓN DE TRADUCCIÓN= f(Index) = POSICIÓN DE COMPARTIMENTO FÍSICO
(4) PROCESADO DE CONSULTA
(5) LECTURA DEJANDO A UN LADO LA CACHE
(5) LEER COMPARTIMENTO DESEADO DE FLASH
(6) DEVOLVER DATOS (DE REGISTRO) SATÉLITES
TABLA DE TRADUCCIÓN DE COMPARTIMENTOS
CACHE (COMPARTIMENTOS FLASH)
BLOQUE DE BORRADO FLASH
COMPARTIMENTO FLASH
41
42
43
44
45a
45b
46
17
21
21a
21b
21c
22
23
26

FIG. 3

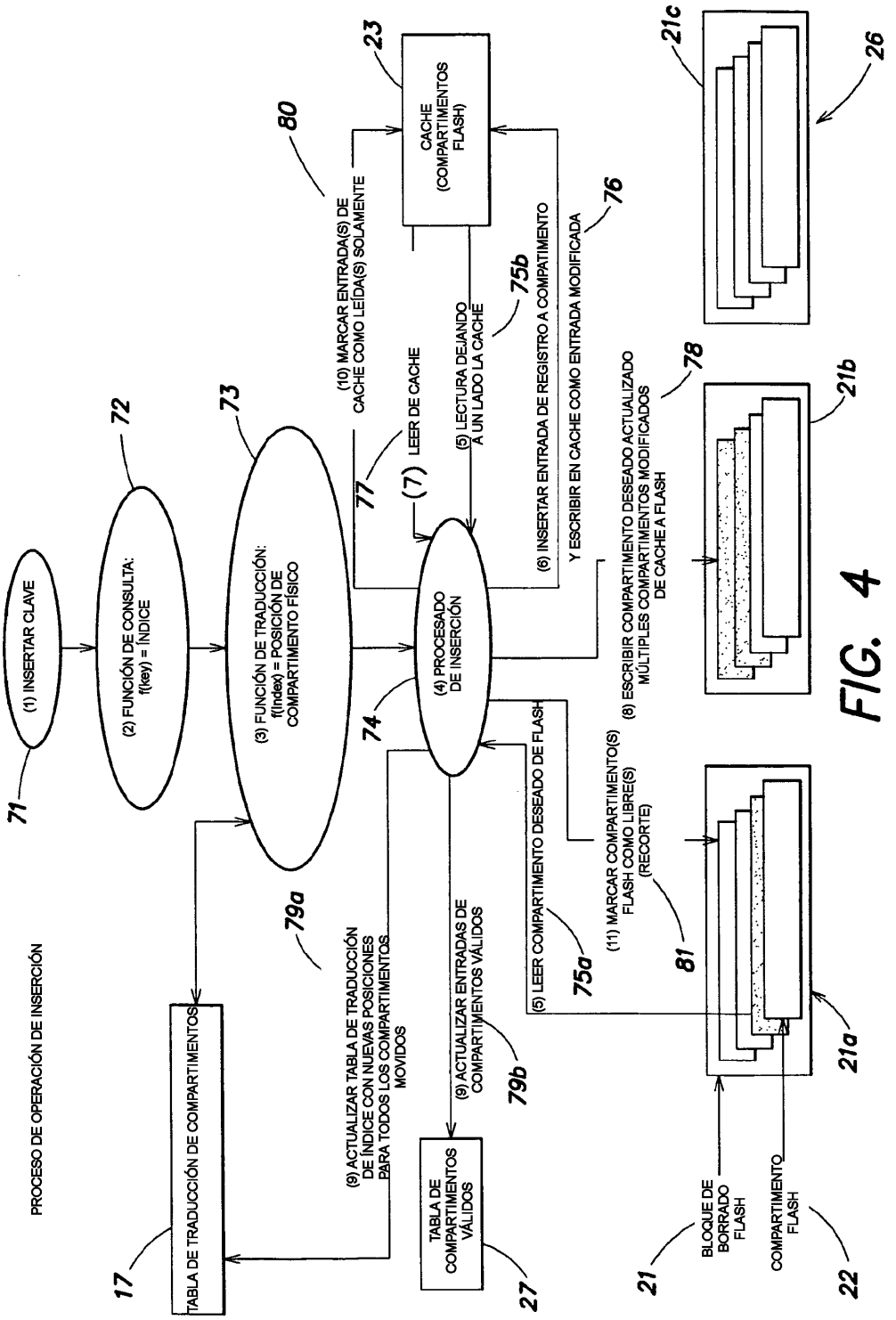
PROCESO DE OPERACIÓN DE INSERCIÓN
(1) INSERTAR CLAVE
71
(2) FUNCIÓN DE CONSULTA: f(key) = ÍNDICE
72
(3) FUNCIÓN DE TRADUCCIÓN: f(Index) = POSICIÓN DE COMPARTIMIENTO FÍSICO
73
(4) PROCESADO DE INSERCIÓN
74
(5) LECTURA DEJANDO A UN LADO LA CACHÉ
75b
(5) LEER COMPARTIMIENTO DESEADO DE FLASH
75a
(6) INSERTAR ENTRADA DE REGISTRO A COMPATIMIENTO Y ESCRIBIR EN CACHE COMO ENTRADA MODIFICADA
76
(7) LEER DE CACHE
77
(8) ESCRIBIR COMPARTIMIENTO DESEADO ACTUALIZADO MÚLTIPLES COMPARTIMIENTOS MODIFICADOS DE CACHE A FLASH
78
(9) ACTUALIZAR TABLA DE TRADUCCIÓN DE ÍNDICE CON NUEVAS POSICIONES PARA TODOS LOS COMPARTIMIENTOS MOVIDOS
79a
(9) ACTUALIZAR ENTRADAS DE COMPARTIMIENTOS VÁLIDOS
79b
(10) MARCAR ENTRADA(S) DE CACHE COMO LEÍDA(S) SOLAMENTE
80
(11) MARCAR COMPARTIMENTO(S) FLASH COMO LIBRE(S) (RECORTE)
81
CACHE (COMPARTIMENTOS FLASH)
23
TABLA DE TRADUCCIÓN DE COMPARTIMENTOS
17
TABLA DE COMPARTIMENTOS VÁLIDOS
27
BLOQUE DE BORRADO FLASH
21
COMPARTIMENTO FLASH
22
21a
21b
21c
26

FIG. 4

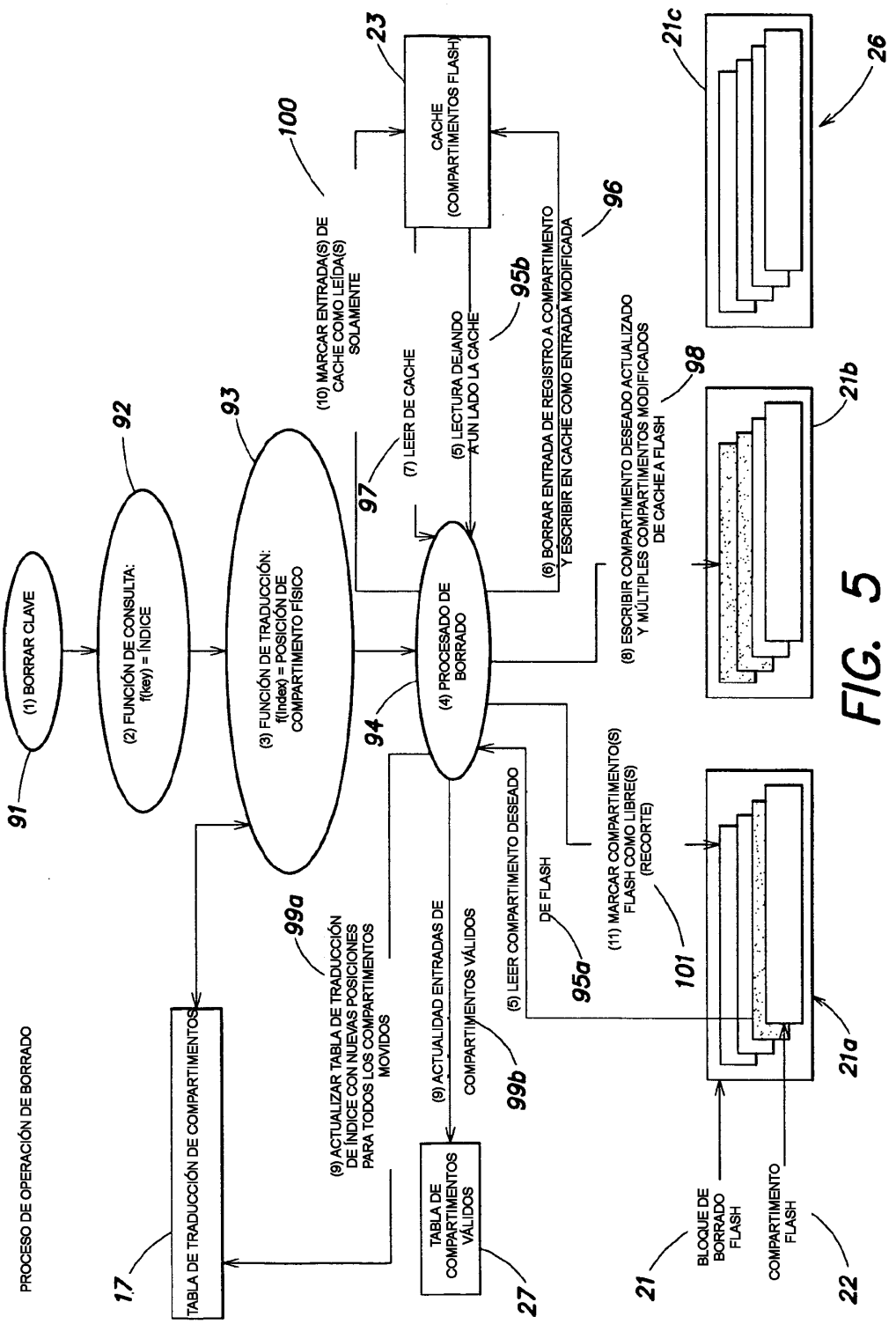
PROCESO DE OPERACIÓN DE BORRADO
(1) BORRAR CLAVE
91
(2) FUNCIÓN DE CONSULTA: f(key) = ÍNDICE
92
(3) FUNCIÓN DE TRADUCCIÓN: f(Index) = POSICIÓN DE COMPARTIMENTO FÍSICO
93
100
(10) MARCAR ENTRADA(S) DE CACHE COMO LEÍDA(S) SOLAMENTE
97
(7) LEER DE CACHE
(5) LECTURA DEJANDO A UN LADO LA CACHE
95b
CACHE (COMPARTIMENTOS FLASH)
23
(6) BORRAR ENTRADA DE REGISTRO A COMPARTIMENTO Y ESCRIBIR EN CACHE COMO ENTRADA MODIFICADA
96
(4) PROCESADO DE BORRADO
94
(8) ESCRIBIR COMPARTIMENTO DESEADO ACTUALIZADO Y MÚLTIPLES COMPARTIMENTOS MODIFICADOS DE CACHE A FLASH
98
(11) MARCAR COMPARTIMENTO(S) FLASH COMO LIBRE(S) (RECORTE)
101
(5) LEER COMPARTIMENTO DESEADO DE FLASH
95a
(9) ACTUALIZAR TABLA DE TRADUCCIÓN DE ÍNDICE CON NUEVAS POSICIONES PARA TODOS LOS COMPARTIMENTOS MOVIDOS
99a
(9) ACTUALIDAD ENTRADAS DE COMPARTIMENTOS VÁLIDOS
99b
TABLA DE TRADUCCIÓN DE COMPARTIMENTOS
17
TABLA DE COMPARTIMENTOS VÁLIDOS
27
BLOQUE DE BORRADO FLASH
21
COMPARTIMENTO FLASH
22
21a
21b
21c
26

FIG. 5

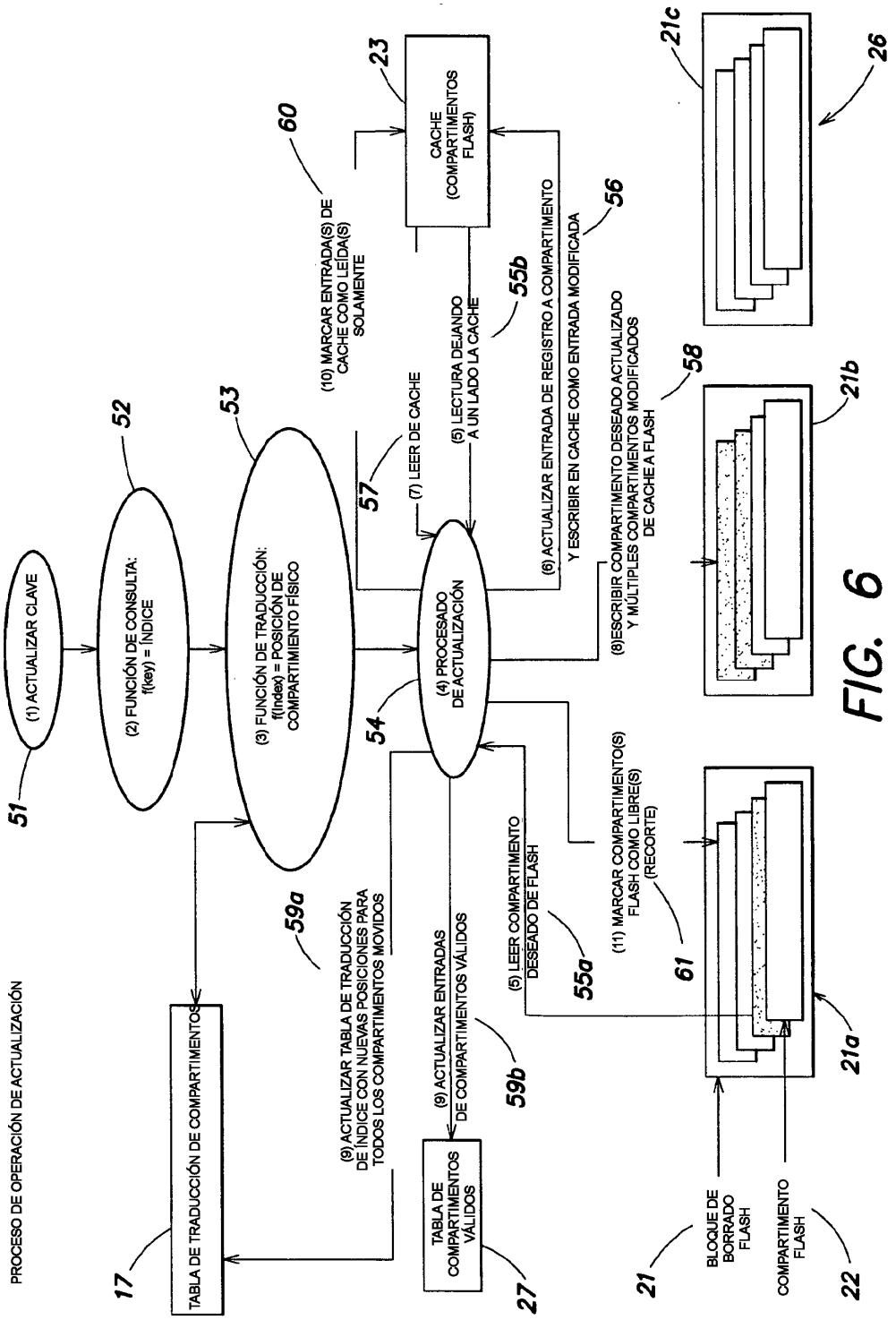
PROCESO DE OPERACIÓN DE ACTUALIZACIÓN
(1) ACTUALIZAR CLAVE
51
(2) FUNCIÓN DE CONSULTA: f(key) = ÍNDICE
52
(3) FUNCIÓN DE TRADUCCIÓN: f(índex) = POSICIÓN DE COMPARTIMIENTO FÍSICO
53
(4) PROCESADO DE ACTUALIZACIÓN
54
(5) LECTURA DEJANDO A UN LADO LA CACHE
55b
(5) LEER COMPARTIMENTO DESEADO DE FLASH
55a
(6) ACTUALIZAR ENTRADA DE REGISTRO A COMPARTIMENTO Y ESCRIBIR EN CACHE COMO ENTRADA MODIFICADA
56
(7) LEER DE CACHE
57
(8)ESCRIBIR COMPARTIMENTO DESEADO ACTUALIZADO Y MÚLTIPLES COMPARTIMENTOS MODIFICADOS DE CACHE A FLASH
58
(9) ACTUALIZAR TABLA DE TRADUCCIÓN DE ÍNDICE CON NUEVAS POSICIONES PARA TODOS LOS COMPARTIMENTOS MOVIDOS
59a
(9) ACTUALIZAR ENTRADAS DE COMPARTIMENTOS VÁLIDOS
59b
(10) MARCAR ENTRADA(S) DE CACHE COMO LEÍDA(S) SOLAMENTE
60
(11) MARCAR COMPARTIMENTO(S) FLASH COMO LIBRE(S) (RECORTE)
61
CACHE (COMPARTIMENTOS FLASH)
23
TABLA DE TRADUCCIÓN DE COMPARTIMENTOS
17
TABLA DE COMPARTIMENTOS VÁLIDOS
27
BLOQUE DE BORRADO FLASH
21
COMPARTIMENTO FLASH
22
21a
21b
21c
26

FIG. 6

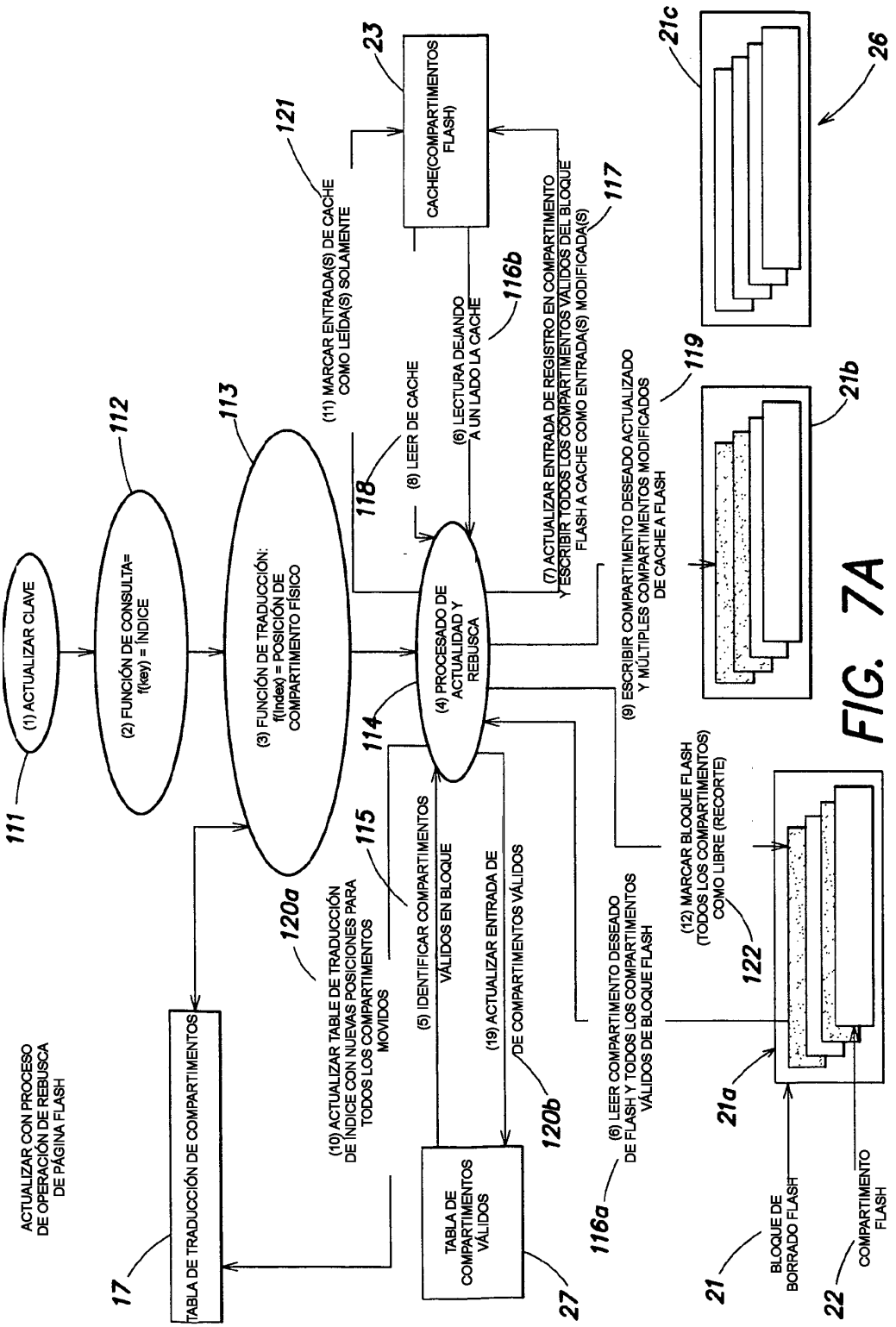
ACTUALIZAR CON PROCESO DE OPERACIÓN DE REBUSCA DE PÁGINA FLASH
(1) ACTUALIZAR CLAVE
111
(2) FUNCIÓN DE CONSULTA= f(key) = ÍNDICE
112
(3) FUNCIÓN DE TRADUCCIÓN: f(Index) = POSICIÓN DE COMPARTIMENTO FÍSICO
113
(4) PROCESADO DE ACTUALIDAD Y REBUSCA
114
(5) IDENTIFICAR COMPARTIMENTOS VÁLIDOS EN BLOQUE
115
(6) LECTURA DEJANDO A UN LADO LA CACHE
116b
(6) LEER COMPARTIMENTO DESEADO DE FLASH Y TODOS LOS COMPARTIMENTOS VÁLIDOS DE BLOQUE FLASH
116a
(7) ACTUALIZAR ENTRADA DE REGISTRO EN COMPARTIMENTO Y ESCRIBIR TODOS LOS COMPARTIMENTOS VÁLIDOS DEL BLOQUE FLASH A CACHE COMO ENTRADA(S) MODIFICADA(S)
117
(8) LEER DE CACHE
118
(9) ESCRIBIR COMPARTIMENTO DESEADO ACTUALIZADO Y MÚLTIPLES COMPARTIMENTOS MODIFICADOS DE CACHE A FLASH
119
(10) ACTUALIZAR TABLE DE TRADUCCIÓN DE ÍNDICE CON NUEVAS POSICIONES PARA TODOS LOS COMPARTIMENTOS MOVIDOS
120a
(19) ACTUALIZAR ENTRADA DE DE COMPARTIMENTOS VÁLIDOS
120b
(11) MARCAR ENTRADA(S) DE CACHE COMO LEÍDA(S) SOLAMENTE
121
(12) MARCAR BLOQUE FLASH (TODOS LOS COMPARTIMENTOS) COMO LIBRE (RECORTE)
122
CACHE(COMPARTIMENTOS FLASH)
23
TABLA DE TRADUCCIÓN DE COMPARTIMENTOS
17
TABLA DE COMPARTIMENTOS VÁLIDOS
27
BLOQUE DE BORRADO FLASH
21
COMPARTIMENTO FLASH
22
21a
21b
21c
26

FIG. 7A

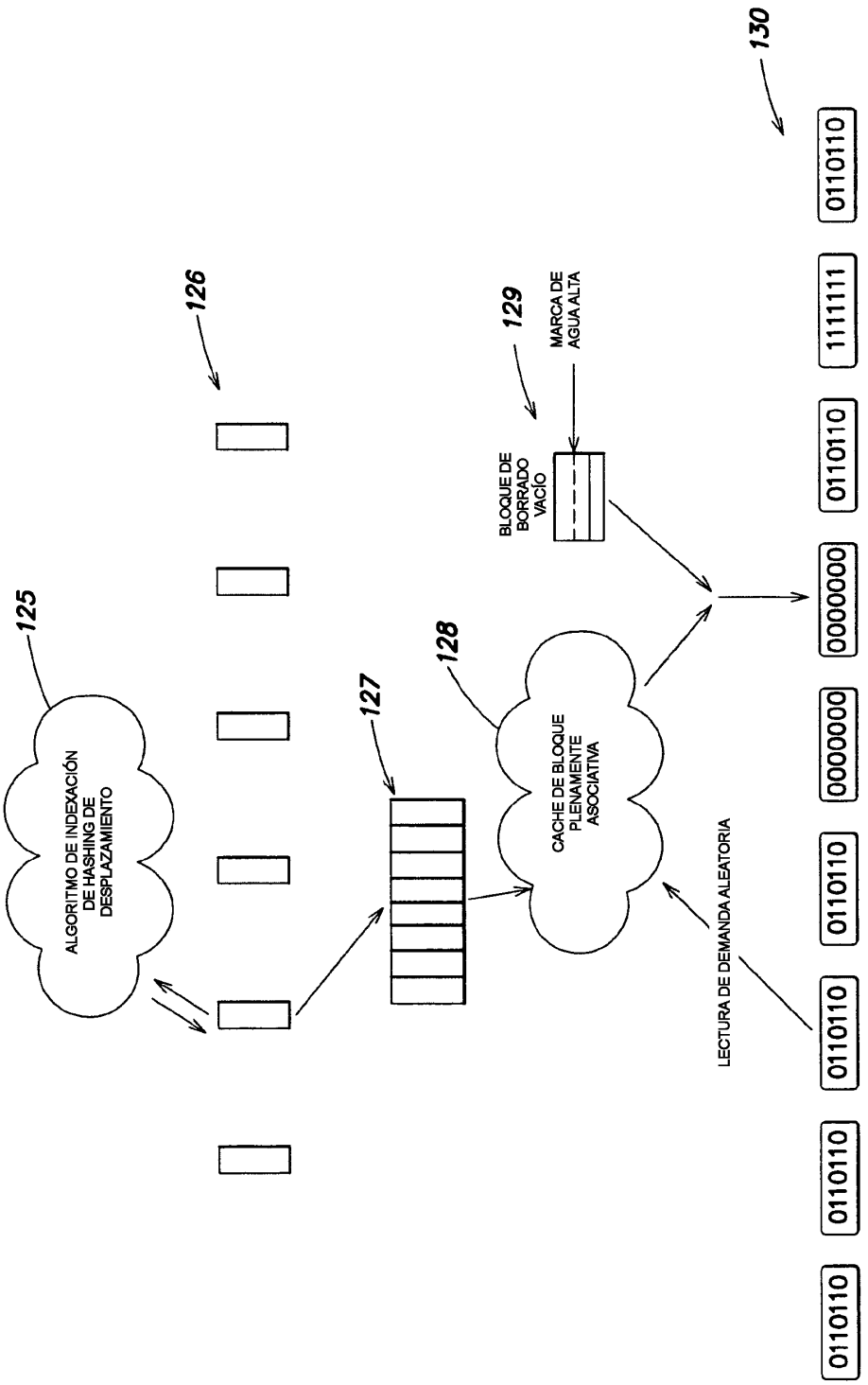
130
126
129
MARCA DE AGUA ALTA
BLOQUE DE BORRADO VACÍO
125
ALGORITMO DE INDEXACIÓN DE HASHING DE DESPLAZAMIENTO
127
128
CACHE DE BLOQUE PLENAMENTE ASOCIATIVA
LECTURA DE DEMANDA ALEATORIA
0110110
0110110
0110110
0110110
0000000
0000000
0110110
1111111
0110110

FIG. 7B

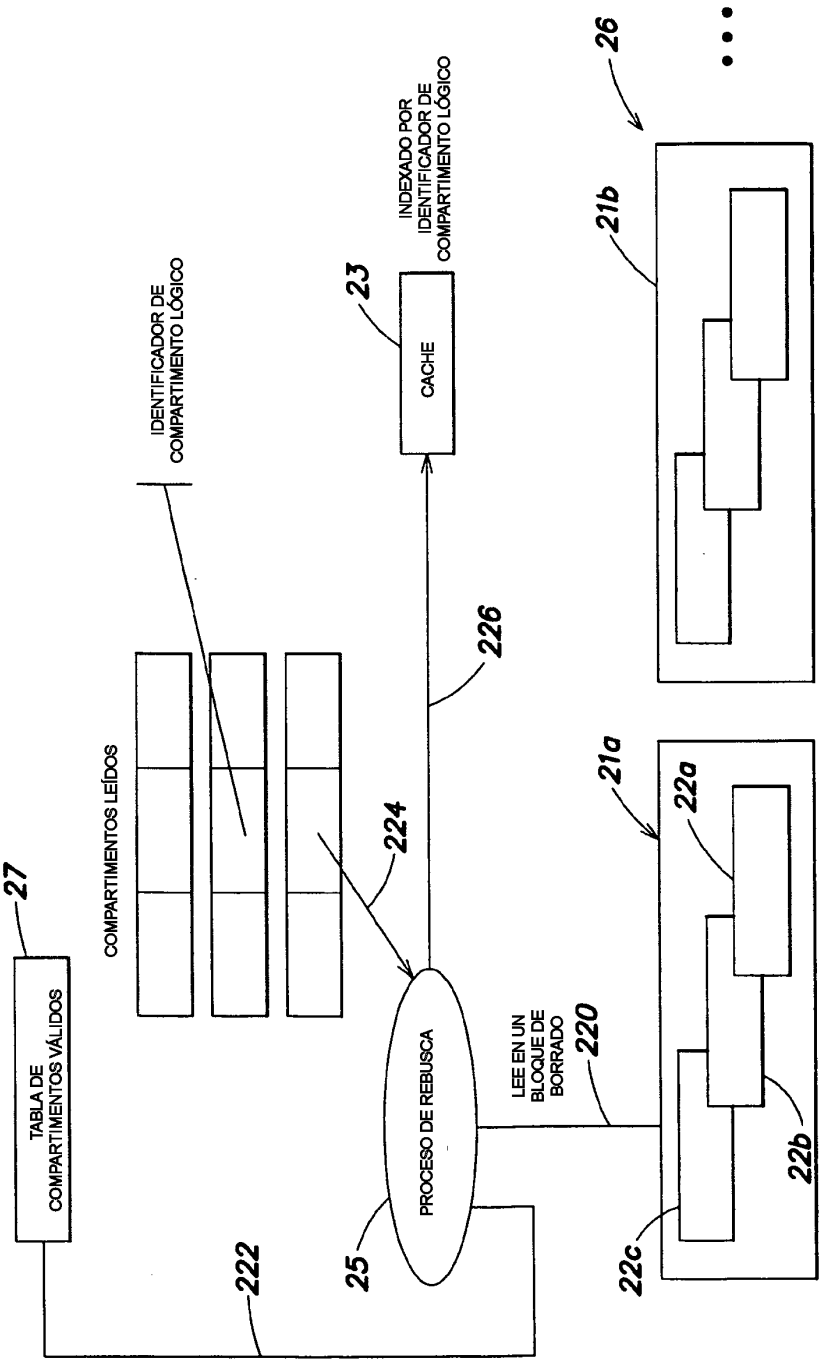
27
TABLA DE COMPARTIMENTOS VÁLIDOS
COMPARTIMENTOS LEÍDOS
IDENTIFICADOR DE COMPARTIMENTO LÓGICO
INDEXADO POR IDENTIFICADOR DE COMPARTIMENTO LÓGICO
23
CACHE
226
224
25
PROCESO DE REBUSCA
LEE EN UN BLOQUE DE BORRADO
220
222
21a
22a
22b
22c
21b
26

FIG. 8A

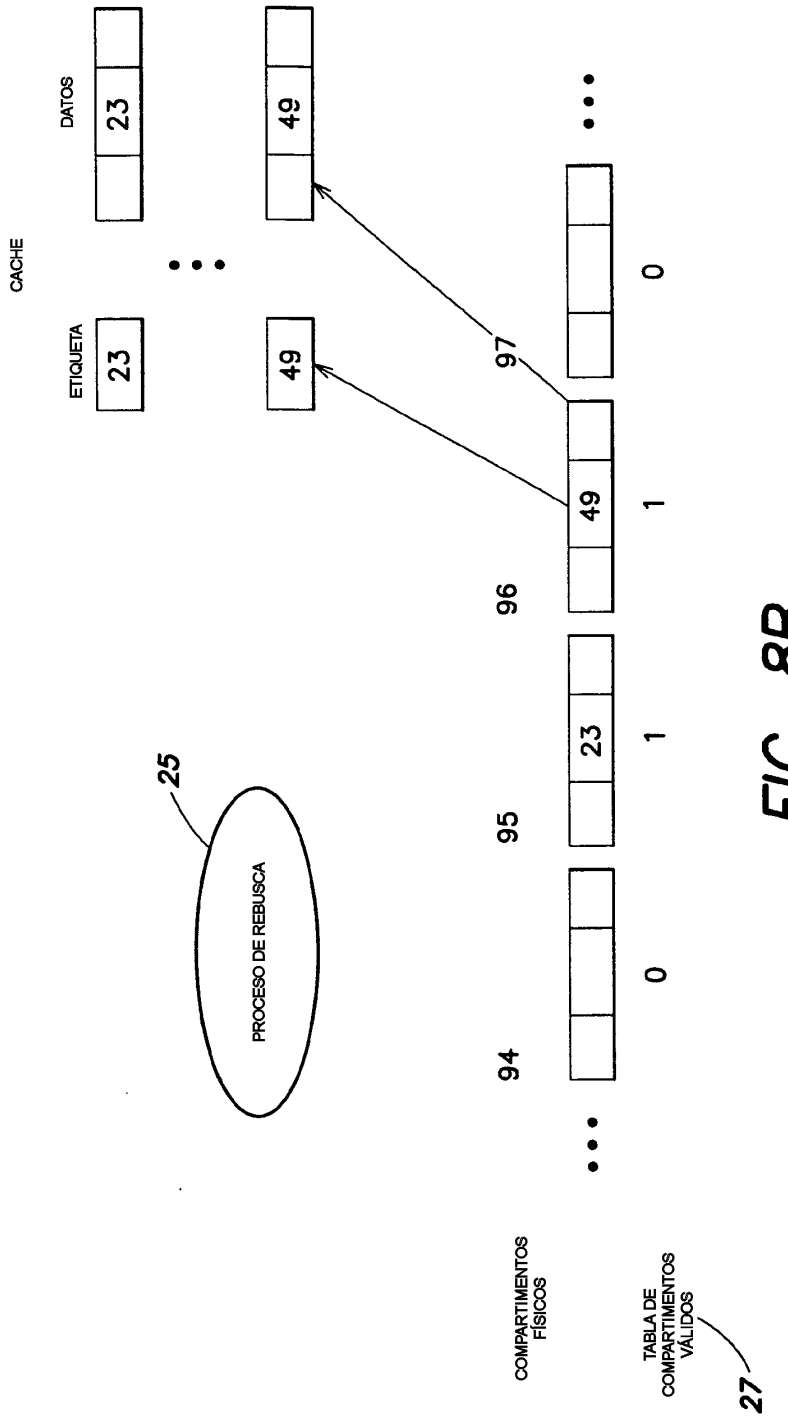
CACHE
DATOS
ETIQUETA
23
49
23
49
97
96
95
94
1
1
0
0
25
PROCESO DE REBUSCA
COMPARTIMENTOS FÍSICOS
TABLA DE COMPARTIMENTOS VÁLIDOS
27

FIG. 8B

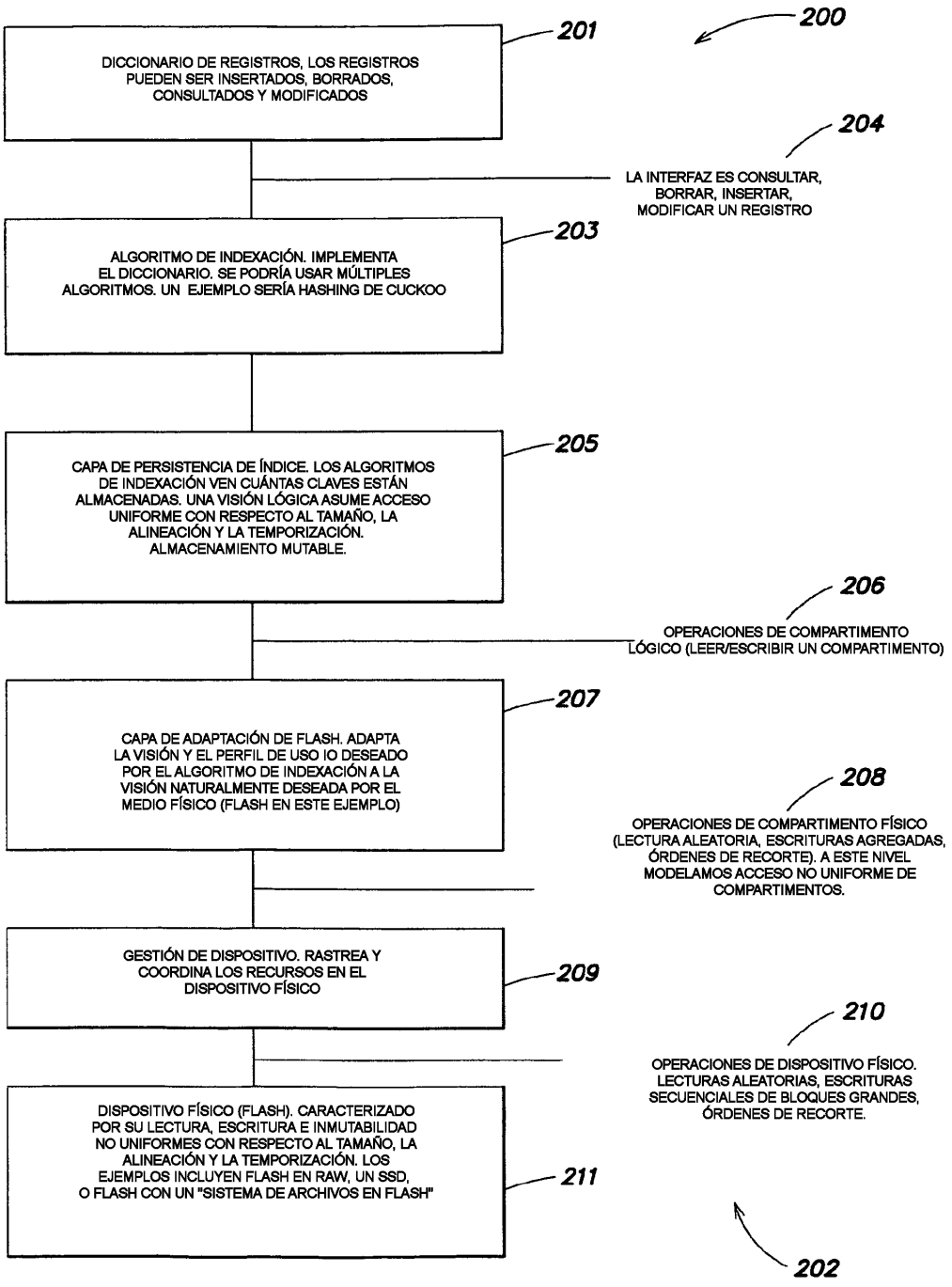
200
201
DICCIONARIO DE REGISTROS, LOS REGISTROS PUEDEN SER INSERTADOS, BORRADOS, CONSULTADOS Y MODIFICADOS
204
LA INTERFAZ ES CONSULTAR, BORRAR, INSERTAR, MODIFICAR UN REGISTRO
203
ALGORITMO DE INDEXACIÓN. IMPLEMENTA EL DICCIONARIO. SE PODRÍA USAR MÚLTIPLES ALGORITMOS. UN EJEMPLO SERÍA HASHING DE CUCKOO
205
CAPA DE PERSISTENCIA DE ÍNDICE. LOS ALGORITMOS DE INDEXACIÓN VEN CUÁNTAS CLAVES ESTÁN ALMACENADAS. UNA VISIÓN LÓGICA ASUME ACCESO UNIFORME CON RESPECTO AL TAMAÑO, LA ALINEACIÓN Y LA TEMPORIZACIÓN. ALMACENAMIENTO MUTABLE.
206
OPERACIONES DE COMPARTIMENTO LÓGICO (LEER/ESCRIBIR UN COMPARTIMENTO)
207
CAPA DE ADAPTACIÓN DE FLASH. ADAPTA LA VISIÓN Y EL PERFIL DE USO IO DESEADO POR EL ALGORITMO DE INDEXACIÓN A LA VISIÓN NATURALMENTE DESEADA POR EL MEDIO FÍSICO (FLASH EN ESTE EJEMPLO)
208
OPERACIONES DE COMPARTIMENTO FÍSICO (LECTURA ALEATORIA, ESCRITURAS AGREGADAS, ÓRDENES DE RECORTE). A ESTE NIVEL MODELAMOS ACCESO NO UNIFORME DE COMPARTIMENTOS.
209
GESTIÓN DE DISPOSITIVO. RASTREA Y COORDINA LOS RECURSOS EN EL DISPOSITIVO FÍSICO
210
OPERACIONES DE DISPOSITIVO FÍSICO. LECTURAS ALEATORIAS, ESCRITURAS SECUENCIALES DE BLOQUES GRANDES, ÓRDENES DE RECORTE.
211
DISPOSITIVO FÍSICO (FLASH). CARACTERIZADO POR SU LECTURA, ESCRITURA E INMUTABILIDAD NO UNIFORMES CON RESPECTO AL TAMAÑO, LA ALINEACIÓN Y LA TEMPORIZACIÓN. LOS EJEMPLOS INCLUYEN FLASH EN RAW, UN SSD, O FLASH CON UN "SISTEMA DE ARCHIVOS EN FLASH"
202

FIG. 9

HASHING DE DESPLAZAMIENTO (HASHING DE CUCKOO)

	$H_0(X)$	$H_1(X)$
P	2	5
Q	1	3

FIG. 11A

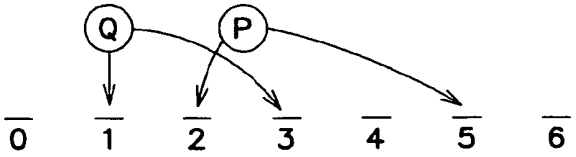

FIG. 11B

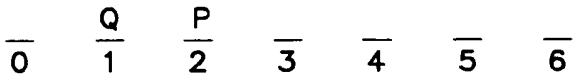

FIG. 11C

	$H_0(R)$	$H_1(R)$
R	1	2

FIG. 11D

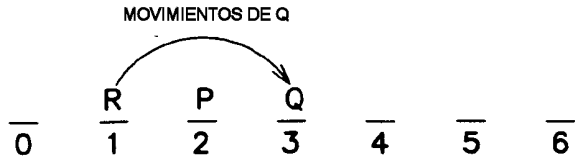

FIG. 11E

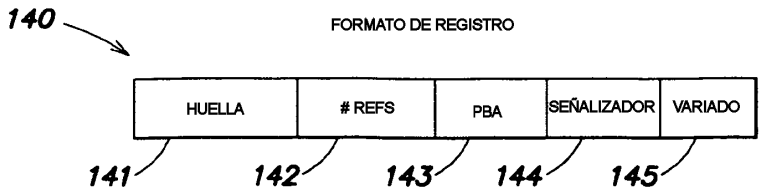

FIG. 10

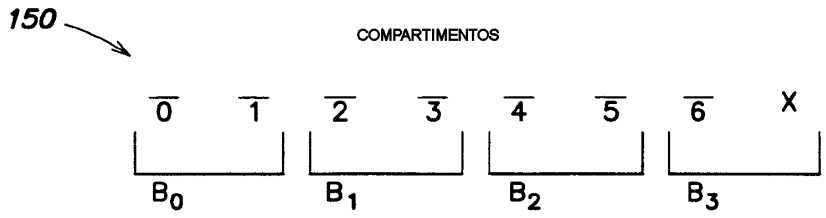

FIG. 12

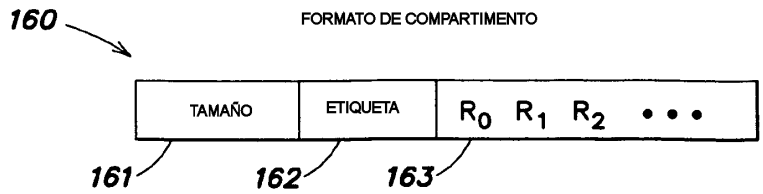

FIG. 13

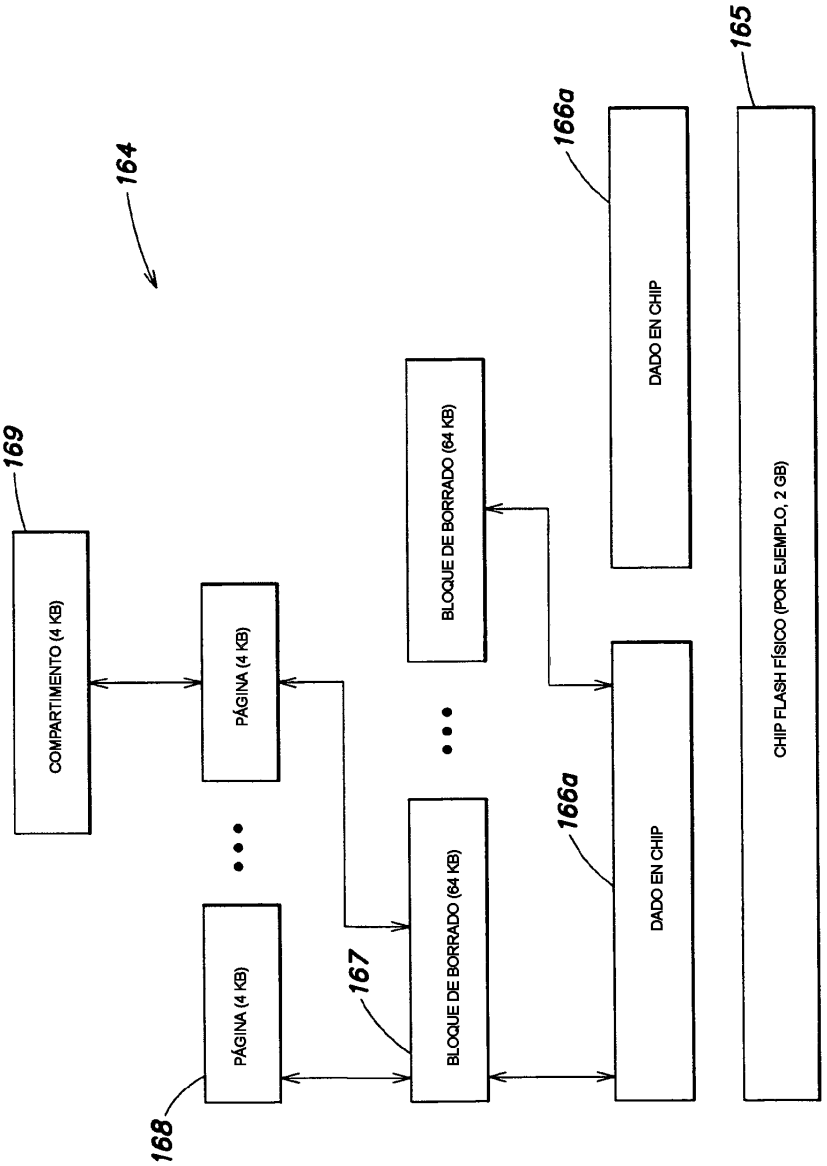
164
165
166a
166a
169
167
168
COMPARTIMENTO (4 KB)
PÁGINA (4 KB)
PÁGINA (4 KB)
BLOQUE DE BORRADO (64 KB)
BLOQUE DE BORRADO (64 KB)
DADO EN CHIP
DADO EN CHIP
CHIP FLASH FÍSICO (POR EJEMPLO, 2 GB)

FIG. 14

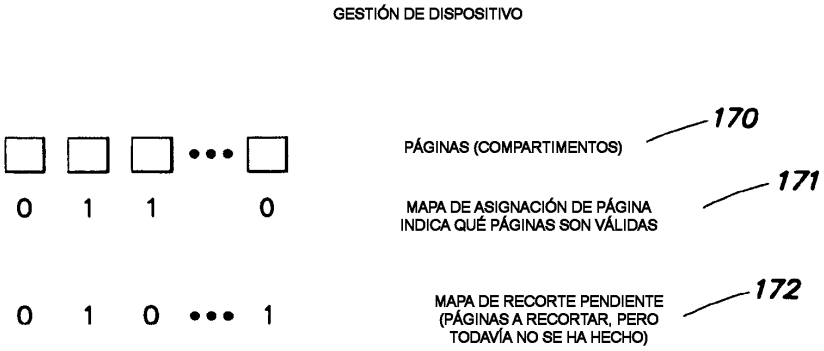

FIG. 15A

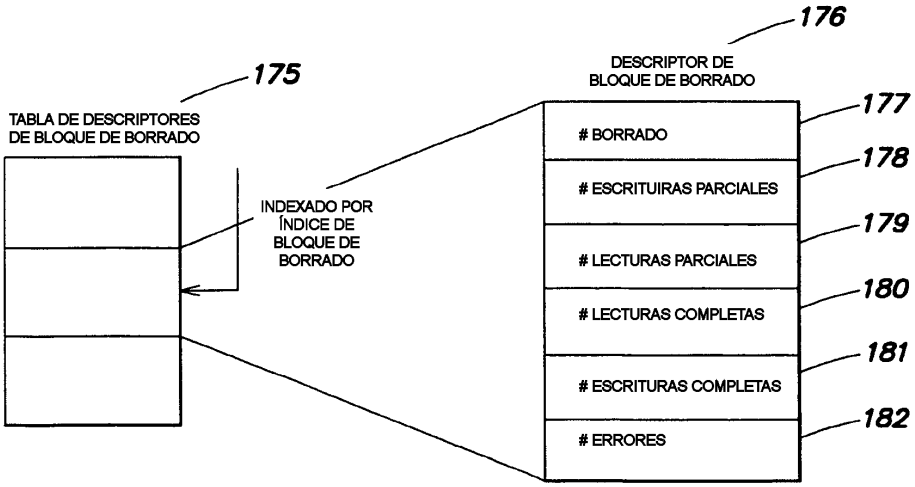

FIG. 15B