[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 12/02 (2006.01)

[12] 发明专利申请公布说明书

[21] 申请号 200480042129.3

[43] 公开日 2007 年 2 月 28 日

[11] 公开号 CN 1922585A

[22] 申请日 2004.12.21
[21] 申请号 200480042129.3
[30] 优先权
[32] 2003.12.30 [33] US [31] 10/750,155
[32] 2004. 8.13 [33] US [31] 10/917,867
[86] 国际申请 PCT/US2004/043762 2004.12.21
[87] 国际公布 WO2005/066793 英 2005.7.21
[85] 进入国家阶段日期 2006.8.25
[71] 申请人 桑迪士克股份有限公司
地址 美国加利福尼亚州
[72] 发明人 艾伦·威尔士·辛克莱
瑟吉·阿纳托利耶维奇·戈罗别茨
艾伦·戴维·贝内特
彼得·约翰·史密斯
[74] 专利代理机构 北京律盟知识产权代理有限责任公司
代理人 刘国伟
权利要求书 4 页 说明书 66 页 附图 63 页

[54] 发明名称

具有非循序更新区块管理的非易失性存储器及方法

[57] 摘要

在具有支持具有非循序逻辑单位的更新区块的区块管理系统的一非易失性存储器中，将一非循序更新区块中的逻辑单位的一索引在 RAM 中缓冲并定期存储到非易失性存储器中。 在一实施例中，将所述索引存储在专用于存储索引的一区块中。 在另一实施例中，将所述索引存储在更新区块本身中。在又一实施例中，将索引存储在每一逻辑单位的标头中。 在另一方面中，在上一索引更新之后但在下一索引更新之前写入的所述逻辑单位会将其索引信息存储在每一逻辑单位的标头中。 以此方式，在电源中断后，不必在一初始化期间执行扫描，即可确定最近写入的逻辑单位的位置。 在又一方面中，将一区块管理成部分循序且部分非循序，指向一个以上逻辑子群组。

1. 一种在一组织成区块的非易失性存储器中存储及更新数据的方法，每一区块已分割成可一起擦除的存储器单位，每一存储器单位用于存储一数据逻辑单位，所述方法包含：

将数据组织成复数个逻辑群组，每一逻辑群组为逻辑单位的一群组；

接收封装在逻辑单位中的主机数据；

根据一第一次序，在一第一区块中存储一逻辑群组的逻辑单位的一第一版本；及

根据与所述第一次序不同的一第二次序，在一包括所述逻辑单位的后续版本的第二区块中存储逻辑单位的最新版本；及

响应于一预定义的触发事件，在一第三区块中存储自一最新触发事件后存储于所述第二区块中的逻辑单位的一目录。

2. 如权利要求 1 所述的方法，其进一步包含：

提供每一逻辑单位的一标头部分；及

对于自一最新的预定义的触发事件后存储于所述第二区块中的中间逻辑单位，在每一所述中间逻辑单位的所述标头部分中存储所述中间逻辑单位的一目录。

3. 如权利要求 1 所述的方法，其中所述非易失性存储器具有浮栅存储器单元。

4. 如权利要求 1 所述的方法，其中所述非易失性存储器为快闪 EEPROM。

5. 如权利要求 1 所述的方法，其中所述非易失性存储器为 NROM。

6. 如权利要求 1 所述的方法，其中所述非易失性存储器在一存储卡中。

7. 如权利要求 1 到 6 中任一权利要求所述的方法，其中所述非易失性存储器具有各自存储一位数据的存储器单元。

8. 如权利要求 1 到 6 中任一权利要求所述的方法，其中所述非易失性存储器具有各自存储一位以上数据的存储器单元。

9. 一种在一组织成区块的非易失性存储器中存储及更新数据的方法，每一区块已分割成可一起擦除的存储器单位，每一存储器单位用于存储一数据逻辑单位，所述方法包含：

将数据组织成复数个逻辑群组，每一逻辑群组为逻辑单位的一群组；

接收封装在逻辑单位中的主机数据；

根据一第一次序，在一第一区块中存储逻辑单位的一第一版本；及

根据与所述第一次序不同的一第二次序，在一包括所述逻辑单位的后续版本的第

二区块中存储逻辑单位的最新版本；及

响应于一预定义的触发事件，在所述第二区块的一可用位置中存储自一最新触发事件后存储于所述第二区块中的逻辑单位的一目录。

10. 如权利要求 9 所述的方法，其进一步包含：

提供用于每一逻辑单位的一标头部分；及

对于自一最新的预定义的触发事件后存储于所述第二区块中的中间逻辑单位，在每一所述中间逻辑单位的所述标头部分中存储所述中间逻辑单位的一目录。

11. 如权利要求 9 所述的方法，其中所述非易失性存储器具有浮栅存储器单元。

12. 如权利要求 9 所述的方法，其中所述非易失性存储器为快闪 EEPROM。

13. 如权利要求 9 所述的方法，其中所述非易失性存储器为 NROM。

14. 如权利要求 9 所述的方法，其中所述非易失性存储器在一存储卡中。

15. 如权利要求 9 到 14 中任一权利要求所述的方法，其中所述非易失性存储器具有各自存储一位数据的存储器单元。

16. 如权利要求 9 到 14 中任一权利要求所述的方法，其中所述非易失性存储器具有各自存储一位以上数据的存储器单元。

17. 一种在一组织成区块的非易失性存储器中存储及更新数据的方法，每一区块已分割成可一起擦除的存储器单位，每一存储器单位用于存储一数据逻辑单位，所述方法包含：

在一第一区块中存储逻辑单位；

响应于一预定义的触发事件，在所述非易失性存储器的一部分中存储至此在所述第一区块中的逻辑单位的一目录；

提供每一逻辑单位的一标头部分；及

对于自一最新的预定义的触发事件后存储于所述区块中的中间逻辑单位，在每一所述中间逻辑单位的所述标头部分中存储所述中间逻辑单位的一目录。

18. 如权利要求 17 所述的方法，其中所述非易失性存储器具有浮栅存储器单元。

19. 如权利要求 17 所述的方法，其中所述非易失性存储器为快闪 EEPROM。

20. 如权利要求 17 所述的方法，其中所述非易失性存储器为 NROM。

21. 如权利要求 17 所述的方法，其中所述非易失性存储器在一存储卡中。

22. 如权利要求 17 到 21 中任一权利要求所述的方法，其中所述非易失性存储器具有各自存储一位数据的存储器单元。

23. 如权利要求 17 到 21 中任一权利要求所述的方法，其中所述非易失性存储器具有各

自存储一位以上数据的存储器单元。

24. 一种在一组织成区块的非易失性存储器中存储及更新数据的方法，每一区块已分割成可一起擦除的存储器单位，每一存储器单位用于存储一数据逻辑单位，所述方法包含：

将数据组织成复数个逻辑群组，每一逻辑群组为逻辑单位的一群组；

接收封装在逻辑单位中的主机数据；

根据一第一次序，在一第一区块中存储逻辑单位的一第一版本；及

根据与所述第一次序不同的一第二次序，在一包括所述逻辑单位的后续版本的第二区块中存储逻辑单位的最新版本；

提供每一逻辑单位的一标头部分；及

在存储于所述第二区块中的每一逻辑单位的所述标头部分中存储目录数据，逻辑单位的所述目录数据存储在所述第二区块中。

25. 如权利要求 24 所述的方法，其中所述非易失性存储器具有浮栅存储器单元。

26. 如权利要求 24 所述的方法，其中所述非易失性存储器为快闪 EEPROM。

27. 如权利要求 24 所述的方法，其中所述非易失性存储器为 NROM。

28. 如权利要求 24 所述的方法，其中所述非易失性存储器在一存储卡中。

29. 如权利要求 24 到 28 中任一权利要求所述的方法，其中所述非易失性存储器具有各自存储一位数据的存储器单元。

30. 如权利要求 24 到 28 中任一权利要求所述的方法，其中所述非易失性存储器具有各自存储一位以上数据的存储器单元。

31. 一种在一组织成区块的非易失性存储器中存储及更新数据的方法，每一区块已分割成可一起擦除的存储器单位，每一存储器单位用于存储一数据逻辑单位，所述方法包含：

将数据组织成复数个逻辑群组，每一逻辑群组为逻辑单位的一群组；

接收封装在逻辑单位中的主机数据；

根据一第一次序，在一第一区块中存储一逻辑群组的逻辑单位的一第一版本；及

根据一第二次序，在一包括所述逻辑单位的后续版本的第二区块中存储逻辑单位的最新版本，其中已存储的所述逻辑单位具有与所述第一次序相同次序的一第一子群组及与所述第一次序不同次序的一第二子群组；及

在所述非易失性存储器的一部分中维持逻辑单位的所述第二子群组的一目录。

32. 如权利要求 31 所述的方法，其中所述非易失性存储器具有浮栅存储器单元。

33. 如权利要求 31 所述的方法，其中所述非易失性存储器为快闪 EEPROM。

34. 如权利要求 31 所述的方法，其中所述非易失性存储器为 NROM。

35. 如权利要求 31 所述的方法，其中所述非易失性存储器在一存储卡中。

36. 如权利要求 31 到 35 中任一权利要求所述的方法，其中所述非易失性存储器具有各自存储一位数据的存储器单元。

37. 如权利要求 31 到 35 中任一权利要求所述的方法，其中所述非易失性存储器具有各自存储一位以上数据的存储器单元。

具有非循序更新区块管理的非易失性存储器及方法

技术领域

本发明一般涉及非易失性半导体存储器，尤其涉及具有能有效处理区块内的更新数据的存储器区块管理系统的非易失性半导体存储器。

背景技术

能够进行电荷的非易失性存储的固态存储器，尤其是封装成小形状因数卡的EEPROM和快闪EEPROM的形式的固态存储器，近来已成为多种移动和手提装置中所选择的存储器，特别是信息设备和消费型电子产品。与同样是固态存储器的RAM（随机存取存储器）不同，快闪存储器为非易失性的，且即使在关闭电源之后仍能保留其存储的数据。并且，与ROM（只读存储器）不同，快闪存储器类似于磁盘存储装置都是可重写的。尽管成本较高，但快闪存储器日益用于大容量存储应用中。基于旋转磁性媒体的常规大容量存储器（例如，硬磁盘驱动器和软盘），并不适合移动和手提环境。这是因为磁盘驱动器倾向于体积庞大，容易出现机械故障且具有高等待时间和高功率需求。这些不合需要的属性使得基于磁盘的存储器无法在大多数的移动和便携式应用中使用。另一方面，嵌入式和抽取式卡形式的快闪存储器均理想地适用于移动和手持环境，因为其尺寸小、功率消耗低、高速和高可靠性特征。

快闪EEPROM与EEPROM（电可擦除且可编程只读存储器）的类似之处在于，快闪EEPROM是可被擦除且可将新的数据写入或“编程”到其存储器单元中的非易失性存储器。两者均利用场效晶体管结构的位于半导体衬底中通道区上方且在源极与漏极区之间的浮动（未连接）导电栅极。接着在浮栅上方提供控制栅极。浮栅上保留的电荷量控制晶体管的阈值电压特性。也就是说，对于浮栅上给定电平的电荷，存在必须在“开启”晶体管之前向控制栅极施加的相应电压（阈值），以允许在其源极与漏极区之间导电。尤其，例如快闪EEPROM的快闪存储器允许同时擦除存储器单元的整个区块。

浮栅可保持某一电荷范围，且因此可编程为阈值电压窗口内的任何阈值电压电平。阈值电压窗口的尺寸受限于装置的最小和最大阈值电平，而装置的最小和最大阈值电平对应于可编程到浮栅上的电荷范围。阈值窗口一般根据存储器装置的特性、操作条件和历史记录而定。原则上，在此窗口内，每一截然不同的、可解析的阈值电压电平范围均可用来指定单元的明确的存储器状态。

通常通过两种机制的一种，将作为存储器单元的晶体管编程为“已编程”状态。在“热电子注入”中，施加于漏极的高电压加速跨越在衬底通道区上的电子。同时，施加于控制栅极的高电压吸引热电子通过薄栅极电介质到浮栅上。在“隧穿注入”中，相对于衬底而向控制栅极施加高电压。依此方式，将电子从衬底吸引到介入的浮栅。虽然习惯使用术语“编程”描述通过将电子注入到存储器单位的初始擦除的电荷存储单元来写入到存储器以改变存储器状态，但此术语现已与例如“写入”或“记录”的较为常见的术语互换使用。

可通过许多机制来擦除存储器装置。对于EEPROM而言，通过相对于控制栅极而向衬底施加高电压以便诱导浮栅的电子隧穿经过薄氧化物到衬底通道区（即，Fowler-Nordheim隧穿），使存储器单元为电可擦除的。通常，EEPROM可逐字节地擦除。对于快闪EEPROM而言，可以一次电擦除全部或每次电擦除一个或一个以上最小可擦除区块的方式擦除存储器，其中最小可擦除区块可能由一个或一个以上扇区组成，且每一扇区可存储512字节或更多数据。

存储器装置通常包含可安装在卡上的一个或一个以上存储器芯片。每一存储器芯片包含外围电路（例如，解码器和擦除、写入与读取电路）支持的存储器单元阵列。较为精密的存储器装置还附有执行智能型且较高级存储器操作和接口连接的控制器。

如今正使用的是许多市售成功的非易失性固态存储器装置。这些存储器装置可为快闪EEPROM，或可使用其它类型的非易失性存储器单元。美国专利第5,070,032号、第5,095,344号、第5,315,541号、第5,343,063号和第5,661,053号、第5,313,421号和第6,222,762号中给出了快闪存储器和系统及其制造方法的实例。明确地说，美国专利第5,570,315号、第5,903,495号、第6,046,935号中描述了具有NAND串结构的快闪存储器装置。并且，也可利用具有用于存储电荷的介电层的存储器单元来制造非易失性存储器装置。使用介电层来取代前面所描述的导电浮栅元件。Eitan等人的2000年11月的IEEE Electron Device Letters，第21卷，第11号，第543-545页中的“NROM: A Novel Localized Trapping, 2-Bit Nonvolatile Memory Cell”中已描述这些利用介电存储元件的存储器装置。ONO介电层延伸穿过源极与漏极扩散之间的通道。一个数据位的电荷局限在邻近于漏极的介电层中，且另一个数据位的电荷局限在邻近于源极的介电层中。例如，美国专利第5,768,192号和第6,011,725号揭示具有夹在两个二氧化硅层之间的捕集电介质的非易失性存储器单元。通过分离地读取电介质内的空间上分离的电荷存储区域的二进制状态来实施多状态数据存储。

为了改进读取和编程性能，并行地读取或编程阵列中多个电荷存储元件或存储器晶

体管。因此，一起读取或编程存储器元件的“页”。在现有的存储器结构中，行通常含有若干交叉的页或其可构成一页。页的所有存储器元件将被一起读取或编程。

在快闪存储器系统中，擦除操作可能需要多达长于读取和编程操作的数目级。因此，希望具有大尺寸的擦除区块。以此方式，将擦除时间分摊在大量的存储器单元上。

依照快闪存储器的性质，数据必须写入到已擦除的存储器位置。如果要更新主机的某一逻辑地址的数据，一种方式是将更新数据重写在相同的物理存储器位置。也就是说，逻辑到物理地址映射不变。然而，这将意味着将必须首先擦除含有所述物理位置的整个擦除区块，且接着用已更新的数据进行重写。尤其在要更新的数据只占据擦除区块的小部分时，此更新方法效率较低，因为其需要擦除并重写整个擦除区块。此也将导致存储器区块的比较频繁的擦除再循环，这鉴于此类型存储器装置的有限耐久性并不理想。

管理快闪存储器系统的另一问题关于系统控制和目录数据。在各种存储器操作的过程期间产生并存取数据。因此，其有效处理和迅速存取将直接影响性能。由于快闪存储器用来存储且为非易失性的，所以希望在快闪存储器中维持此类型的数据。然而，因为控制器与快闪存储器之间存在介入文件管理系统，所以无法直接存取数据。并且，系统控制和目录数据倾向于成为活跃的且成为片段，此并不有益于具有大尺寸区块擦除的系统中的存储。常规地，在控制器RAM中设定此类型的数据，藉此允许由控制器直接存取。在为存储器装置上电之后，初始化过程使得能够扫描快闪存储器以便编辑将要放在控制器RAM中的必要的系统控制和目录信息。此过程耗费时间且需要控制器RAM容量，对于持续增加的快闪存储器容量更是如此。

US 6,567,307揭示一种在大擦除区块中处理扇区更新的方法，其包括在多个充当高速暂存存储器（scratch pad）的擦除区块中记录更新数据，和最后在各种区块中合并有效扇区，并在按逻辑循序次序对其进行重新排列之后重写扇区。以此方式，在每一最微小的更新时，不必擦除和重写区块。

WO 03/027828和WO 00/49488均揭示一种处理大擦除区块中更新的存储器系统，所述处理包括将逻辑扇区地址分割成区。保留逻辑地址范围的较小区用于活跃系统控制数据，所述较小区与用于用户数据的另一区分离。以此方式，对系统控制数据的在其自身区中的操控将不与另一区中关联的用户数据互相作用。更新处于逻辑扇区层级，且写入指针指向将要写入的区块中的相应物理扇区。映射信息缓冲在RAM中，且最后被存储在主存储器的扇区配置表中。逻辑扇区的最新版本将使现有区块中的所有先前版本过时，现有区块因此变得部分过时。执行垃圾收集以将部分过时区块保持为可接受的数目。

现有技术系统倾向于使更新数据分布在许多区块上，或更新数据可使许多现有的区

块部分过时。结果通常是部分过时区块必须进行大量垃圾收集，此效率较低且促使存储器提早老化。并且，与非循序更新相比，没有处理循序更新的系统且有效的方式。

因此，普遍需要高容量且高性能的非易失性存储器。明确地说，需要一种能够在大区块中执行存储器操作而没有上述问题的高容量非易失性存储器。

发明内容

一种非易失性存储器系统是以照物理存储器位置的物理群组来组织。每一物理群组（元区块）可作为单位而擦除且可用来存储一逻辑群组的数据。存储器管理系统允许通过配置专用于记录逻辑群组的更新数据的元区块来更新一逻辑群组的数据。更新元区块以接收次序来记录更新数据，且对于记录以原始存储的正确逻辑次序（循序）与否（混乱）并不存在限制。最后关闭更新元区块以进行进一步记录。将发生若干过程的一个过程，但最终将以取代原始元区块的以正确次序完全填满的元区块结束。在混乱的情况下，以有益于频繁更新的方式将目录数据维持在非易失性存储器中。系统支持正同时更新的多个逻辑群组。

本发明的一个特征允许以逐个逻辑群组的方式来更新数据。因此，在更新逻辑群组时，逻辑单位的分布（以及由于更新而过时的存储器单位的散布）在范围上受到限制。当逻辑群组正常地包含在物理区块内时尤为如此。

在逻辑群组的更新期间，通常需要配置一个或两个区块来缓冲已更新的逻辑单位。因此，只需要在相对较少数目的区块上执行垃圾收集。通过合并或压缩可执行混乱区块的垃圾收集。

更新过程的经济性在更新区块的一般处理中会越加明显，从而与循序区块相比，无需为混乱（非循序）更新配置额外的区块。所有的更新区块均被配置为循序的更新区块，且任何更新区块均可改变为混乱的更新区块。当然，更新区块从循序到混乱的改变是任意的。

系统资源的有效使用允许同时更新多个逻辑群组。这可进一步增加效率并减少额外开销。

分布在多个存储器平面上的存储器的对准

根据本发明的另一方面，对于组织成可擦除区块并由多个存储器平面构成（从而可并行地读取逻辑单位或将逻辑单位编程到多个平面中）的存储器阵列，当将更新存储在给定存储器平面中的第一区块的原始逻辑单位时，规定将已更新的逻辑单位保持在与原始相同的平面中。这可通过将已更新的逻辑单位记录到仍在相同平面中的第二区块的下

一可用位置而完成。优选地，将逻辑单位存储在平面中具有与它的其它版本相同的偏移位置，使得给定逻辑单位的所有版本由相同组的感测电路提供服务。

在优选实施例中，相应地用逻辑单位的当前版本来填补从上一个编程的存储器单位到下一可用的平面对准存储器单位之间的任何介入的间隙。通过用逻辑上跟随着上一个编程的逻辑单位的逻辑单位的当前版本以及用逻辑上超前于存储在下一可用的平面对准存储器单位中的逻辑单位的逻辑单位的当前版本来填充间隙，从而完成填补。

以此方式，将逻辑单位的所有版本维持在相同平面中具有与原始相同的偏移，从而在垃圾收集操作中，不需要从不同平面检索逻辑单位的最新版本而降低性能。在优选实施例中，用最新的版本来更新或填补平面上的每一存储器单位。因此，可并行地读出来自每一平面的逻辑单位，此将按照逻辑循序次序而无需进一步重新排列。

此方案通过允许在平面上重新排列逻辑群组的逻辑单位的最新版本并避免需要从不同的存储器平面搜集最新版本，来缩短合并混乱区块的时间。这是有益的，因为主机接口的性能规格限定存储器系统完成扇区写入操作的最大等待时间。

分阶段程序错误处理

根据本发明的另一方面，在具有区块管理系统的存储器中，在时间紧急（time-critical）存储器操作期间，通过继续中断区块（breakout block）中的编程操作来处理区块中的程序失败。稍后，在较不紧急的时间，将中断前记录在失败区块中的数据传送到也可能是中断区块的另一区块。接着，可丢弃失败区块。以此方式，当遇到缺陷区块时，可在不损失数据且不会因必须立刻传送缺陷区块中的存储的数据而超过指定的时间限制的前提下进行处理。此错误处理对于垃圾收集操作尤其关键，从而在紧急时间期间不需要对崭新的区块重复整个操作。随后，在适宜的时间，可通过重新定位到另一区块来挽救来自缺陷区块的数据。

程序失败处理在合并操作期间尤其重要。正常的合并操作将驻存在原始区块和更新区块中的逻辑群组的所有逻辑单位的当前版本合并到合并区块中。在合并操作期间，如果在合并区块中发生程序失败，那么将提供另一个区块充当中断合并区块以接收剩余逻辑单位的合并。以此方式，不必将逻辑单位复制一次以上，且仍可在为正常合并操作指定的时段内完成例外处理的操作。在适宜的时间，可通过将群组的所有未处理完成的逻辑单位合并到中断区块中来完成合并操作。适宜的时间将是在当前主机写入操作以外的有时间执行合并的一些其它时段期间。一个此适宜的时间是在存在更新但没有关联的合并操作的另一主机写入期间。

实质上，可将程序失败处理的合并视为以多个阶段实施。在第一阶段中，在发生程

序失败之后，将逻辑单位合并到一个以上区块中以便避免将每一逻辑单位合并一次以上。在适宜的时间完成最后阶段，其中优选地通过以循序次序将所有逻辑单位收集到中断合并区块中，将逻辑群组合并到一个区块中。

非循序更新区块索引

根据本发明的另一方面，在具有支持具有非循序逻辑单位的更新区块的区块管理系统的非易失性存储器中，非循序更新区块中逻辑单位的索引被缓冲在RAM中，并周期性地存储到非易失性存储器中。在一个实施例中，索引存储在专用于存储索引的区块中。在另一实施例中，索引存储在更新区块本身中。在又一实施例中，索引存储在每一逻辑单位的标题中。在另一方面，在上一个索引更新之后但在下一个索引更新之前写入的逻辑单位的索引信息存储在每一逻辑单位的标题中。以此方式，在电源中断之后，不必在初始化期间执行扫描即可确定最近写入的逻辑单位的位置。在又一方面，将区块管理成部分循序和部分非循序，指向一个以上逻辑子群组。

控制数据完整性&管理

根据本发明的另一方面，例如控制数据的一些或全部的关键数据如果维持在复制项（duplicate）中，那么保证其额外等级的可靠性。以一方式执行复制，使得对于使用两次编程技术来连续地编程相同组的存储器单元的多位的多状态存储器系统而言，第二次中的任何编程错误将不破坏第一次建立的数据。复制也有助于检测写入中止、检测误测（misdetection）（即，两个副本均具有良好的ECC但数据不同），并增加额外等级的可靠性。已考虑数据复制的若干技术。

在一个实施例中，在稍早的一次编程中已编程给定数据的两个副本之后，后续的一次编程避免编程存储两个副本中的至少一个的存储器单元。以此方式，在后续的一次编程在完成之前中止并破坏稍早的一次的数据的事件中，两个副本中至少一个不会受到影响。

在另一实施例中，给定数据的两个副本存储在两个不同的区块中，其中两个副本中至多一个的存储器单元在后续的一次编程中被编程。

在又一实施例中，在一次编程中已存储给定数据的两个副本之后，将不再对存储两个副本的存储器单元组执行进一步编程。可通过存储器单元组的最终一次编程中编程两个副本来达成此目的。

在又一实施例中，可在二进制编程模式中将给定数据的两个副本编程到多状态存储器中，从而将不会对已编程的存储器单元进行进一步编程。

在又一实施例中，对于使用两次编程技术来连续地编程相同组的存储器单元的多位

的多状态存储器系统而言，使用容错码来编码多个存储器状态，从而稍早的一次编程所建立的数据不会受到后续的一次编程中的错误的影响。

根据本发明的另一方面，在具有区块管理系统的非易失性存储器中，实施存储器区块的“控制垃圾收集”或占先式重新定位以避免大量的更新区块均恰巧同时需要重新定位的情况。例如，在更新用于控制区块管理系统操作的控制数据时可发生此情况。控制数据类型的层级可以不同程度的更新频率而存在，从而导致其关联的更新区块需要以不同速率进行垃圾收集或重新定位。将存在一个以上控制数据类型的垃圾收集操作同时发生的某些时间。在极端的情况中，所有控制数据类型的更新区块的重新定位阶段可排队，从而导致所有的更新区块均需要同时重新定位。

从以下对本发明的优选实施例的描述中将了解本发明的额外特征和优点，应结合附图来阅读所述描述。

附图说明

图1示意地说明适于实施本发明的存储器系统的主要硬件组件。

图2说明根据本发明的优选实施例组织成扇区（或元区块）的物理群组并由控制器的存储器管理器管理的存储器。

图3A（i）到图3A（iii）示意地说明根据本发明的优选实施例逻辑群组与元区块之间的映射。

图3B示意地说明逻辑群组与元区块之间的映射。

图4说明元区块与物理存储器中的结构的对准。

图5A说明通过链接不同平面的最小擦除单位而构成的元区块。

图5B说明从每一平面选择最小擦除单位（MEU）以链接到元区块中的一个实施例。

图5C说明其中从每一平面选择一个以上MEU以链接到元区块中的另一实施例。

图6是如控制器和快闪存储器中实施的元区块管理系统的示意方框图。

图7A说明以循序次序写入到循序的更新区块的逻辑群组中的扇区的实例。

图7B说明以混乱次序写入到混乱的更新区块的逻辑群组中的扇区的实例。

图8说明由于具有逻辑地址不连续性的两个分离的主机写入操作而以循序次序写入到循序的更新区块的逻辑群组中的扇区的实例。

图9是说明根据本发明的一般实施例更新区块管理器更新一逻辑群组的数据的过程的流程图。

图10说明根据本发明的优选实施例更新区块管理器更新一逻辑群组的数据的过程

的流程图。

图11A是更加详细地说明关闭图10所示的混乱更新区块的合并过程的流程图。

图11B是更加详细地说明关闭图10所示的混乱更新区块的压缩过程的流程图。

图12A说明逻辑群组的所有可能的状态，和在各种操作下其间的可能的转换。

图12B是罗列逻辑群组的可能的状态的表。

图13A说明元区块的所有可能的状态，和在各种操作下其间的可能的转换。元区块是对应于逻辑群组的物理群组。

图13B是罗列元区块的可能的状态的表。

图14（A）到图14（J）是展示对逻辑群组的状态以及对物理元区块的各种操作的效果的状态图。

图15说明用于追踪开启和关闭的更新区块及用于配置的已擦除区块的配置区块列表（ABL）结构的优选实施例。

图16A说明混乱区块索引（CBI）扇区的数据字段。

图16B说明记录在专用元区块中的混乱区块索引（CBI）扇区的实例。

图16C是说明存取经历混乱更新的给定逻辑群组的逻辑扇区的数据的流程图。

图16D是说明根据已将逻辑群组分割成子群组的替代实施例，存取经历混乱更新的给定逻辑群组的逻辑扇区的数据的流程图。

图16E说明对于将每一逻辑群组分割成多个子群组的实施例中，混乱区块索引（CBI）扇区及其功能的实例。

图17A说明群组地址表（GAT）扇区的数据字段。

图17B说明记录在GAT区块中的群组地址表（GAT）扇区的实例。

图18是说明使用并再循环已擦除区块的控制和目录信息的分布和流程的示意方框图。

图19是展示逻辑到物理地址转换的过程的流程图。

图20说明在存储器管理的操作过程中，对控制数据结构执行的操作的层级。

图21说明由多个存储器平面构成的存储器阵列。

图22A说明根据本发明的一般实施方案具有平面对准的更新的方法的流程图。

图22B说明图22A所示的流程图中存储更新的步骤的优选实施例。

图23A说明不管平面对准如何以循序次序写入到循序的更新区块的逻辑单位的实例。

图23B说明不管平面对准如何以非循序次序写入到混乱更新区块的逻辑单位的实

例。

图24A说明根据本发明的优选实施例具有平面对准和填补的图23A的循序更新实例。

图24B说明根据本发明的一个优选实施例具有平面对准但不进行填补的图23B的混乱更新实例。

图24C说明根据本发明的另一优选实施例具有平面对准和填补的图23B的混乱更新实例。

图25说明每一页含有用于存储两个逻辑单位（例如，两个逻辑扇区）的两个存储器单位的示范性存储器组织。

图26A与图21的存储器结构类似，只是每一页含有两个扇区而不是一个扇区。

图26B说明图26A所示的具有以示意线性方式布局的存储器单位的元区块。

图27说明在不填补将要从一个位置复制到另一位置的逻辑单位的情况下更新区块中的平面对准的替代方案。

图28说明合并操作期间缺陷区块中发生程序失败时对另一区块重复合并操作的方案。

图29示意地说明具有允许足够时间以完成写入（更新）操作以及合并操作的定时或写入等待时间的主机写入操作。

图30说明根据本发明的一般方案程序失败处理的流程图。

图31A说明程序失败处理的一个具体实施例，其中第三（最后的重新定位）区块不同于第二（中断）区块。

图31B说明程序失败处理的另一实施例，其中第三（最后的重新定位）区块与第二（中断）区块相同。

图32A说明导致合并操作的初始更新操作的流程图。

图32B说明根据本发明的优选实施例多阶段合并操作的流程图。

图33说明多阶段合并操作的第一和最后阶段的示范性定时。

图34A说明中断合并区块不用作更新区块而更大程度上用作合并操作已中断的合并区块的情况。

图34B说明始于图34A的多阶段合并的第三和最后阶段。

图35A说明将中断合并区块维持为接收主机写入的更新区块而不是合并区块的情况。

图35B说明第二情况的始于图35A的多阶段合并的第三和最后阶段。

图36A说明应用于主机写入触发更新区块的关闭且更新区块为循序时的情况的分阶段程序错误处理方法。

图36B说明在更新区块的更新的情况中应用于（局部区块系统）的分阶段程序错误处理方法。

图36C说明处理垃圾收集操作的分阶段程序错误，或不支持映射到元区块的逻辑群组的存储器区块管理系统中的清除。

图37说明在相同逻辑群组的每N个扇区写入之后将CBI扇区写入到关联的混乱索引扇区区块的程序安排的实例。

图38A说明直到在预定数目的写入之后在其中记录CBI扇区时的更新区块。

图38B说明图38A的在索引扇区之后在其中进一步记录数据页1、2和4的更新区块。

图38C说明图38B的写入有触发索引扇区的下一次记录的另一逻辑扇区的更新区块。

图39A说明存储在混乱更新区块中的每一数据扇区标题中的中间写入的中间索引。

图39B说明在写入的每一扇区的标题中存储中间写入的中间索引的实例。

图40说明存储在混乱更新区块的每一数据扇区标题中的混乱索引字段中的信息。

图41A说明当每一存储器单元存储两个位的数据时，4状态存储器阵列的阈值电压分布。

图41B说明使用葛莱码的现有的2次编程方案。

图42说明通过将每一扇区保存在复制项中来保护关键数据的方式。例如，将扇区A、B、C、和D保存在复制的副本中。如果一个扇区副本中存在数据破坏，那么可改为读取另一扇区副本。

图43说明将复制的扇区通常保存在多状态存储器中的非稳固性。

图44A说明将关键数据的交错的复制的副本保存到多状态存储器中的一个实施例。

图44B说明将关键数据的复制的副本只保存到多状态存储器的逻辑上页的另一实施例。

图44C说明以多状态存储器的二进制模式保存关键数据的复制的副本的又一实施例。

图45说明同时将关键数据的复制的副本保存到两个不同的元区块的又一实施例。

图46A在说明4状态存储器阵列的阈值电压分布方面与图41A类似，且展示为图46B的参考。

图46B说明通过使用容错码同时保存关键数据的复制的副本的又一实施例。

图47是展示数据的两个副本的可能的状态和数据有效性的表。

图 48 说明存储控制数据的存储器区块的占先式重新定位的流程图。

具体实施方式

图1示意地说明适于实施本发明的存储器统的主要硬件组件。存储器统20通常通过主机接口以主机10进行操作。存储器系统通常为存储卡或嵌入式存储器系统的形式。存储器系统20包括由控制器100控制操作的存储器200。存储器200包含分布在一个或一个以上集成电路芯片上的一个或一个以上阵列的非易失性存储器单元。控制器100包括：接口110、处理器120、任选的协处理器121、ROM 122（只读存储器）、RAM 130（随机存取存储器）、和可任选地编程的非易失性存储器124。接口110具有一个将控制器连接到主机的组件和另一个连接到存储器200的组件。存储在非易失性ROM 122和/或任选的非易失性存储器124中的韧件为处理器120提供代码以实施控制器100的功能。处理器120或任选的协处理器121可处理错误校正码。在替代实施例中，由状态机（未图示）来实施控制器100。在又一实施例中，控制器100在主机内实施。

逻辑和物理区块结构

图2说明根据本发明的优选实施例的存储器，其被组织成扇区（或元区块）的物理群组并由控制器的存储器管理器进行管理。将存储器200组织成元区块，其中每一元区块是一组可一起擦除的物理扇区S_0、...、S_{N-1}。

主机10在文件系统或操作系统下执行应用程序时存取存储器200。通常，主机以逻辑扇区为单位来定址数据，其中，例如，每一扇区可含有512字节的数据。并且，主机通常以逻辑群集为单位来读取或写入到存储器系统，每一逻辑群集由一个或一个以上逻辑扇区组成。在一些主机系统中，可能存在任选的主机侧（host-side）存储器管理器以在主机处执行较低级的存储器管理。在大多数情况下，在读取或写入操作期间，主机10本质上向存储器系统20发布命令以读取或写入含有一串具有连续地址的数据逻辑扇区的程序段。

存储器侧存储器管理器实施在存储器系统20的控制器100中以便管理快闪存储器200的元区块中主机逻辑扇区的数据的存储和检索。在优选的实施例中，存储器管理器含有用于管理元区块的擦除、读取和写入操作的许多软体模块。存储器管理器还维持与其在快闪存储器200和控制器RAM 130中的操作关联的系统控制和目录数据。

图3A（i）到图3A（iii）示意地说明根据本发明的优选实施例逻辑群组与元区块之间的映射。物理存储器的元区块具有用于存储逻辑群组的N个逻辑扇区的数据的N个物理扇区。图3A（i）展示来自逻辑群组LG_i的数据，其中逻辑扇区呈现连续的逻辑次序0、

1、...、N-1。图3A（ii）展示正以相同的逻辑次序存储在元区块中的相同数据。当以此方式存储时，将元区块视为“循序的”。一般来说，元区块可具有以不同次序存储的数据，在此情况下，将元区块视为“非循序的”或“混乱的”。

逻辑群组的最低地址与其映射到的元区块的最低地址之间可能存在偏移。在此情况下，逻辑扇区地址在元区块内以环形的方式从逻辑群组的底部卷绕回顶部。例如，在图3A（iii）中，元区块在其始于逻辑扇区k的数据的第一位置中进行存储。在到达最后的逻辑扇区N-1时，元区块卷绕到扇区0，且最终在其最后的物理扇区中存储与逻辑扇区k-1关联的数据。在优选的实施例中，使用页标记来识别任何偏移，例如，识别存储在元区块的第一物理扇区中的数据的起始逻辑扇区地址。当两个区块仅由于页标记而不同时，将认为两个区块以类似的次序存储其逻辑扇区。

图3B示意地说明逻辑群组与元区块之间的映射。除了数据正被更新的少量逻辑群组外，每一逻辑群组均映射到唯一的元区块。逻辑群组在已更新之后可映射到不同的元区块。将映射信息维持在一组逻辑到物理目录中，稍后将更详细地进行描述。

也考虑其它类型的逻辑群组到元区块映射。例如，由Alan Sinclair在与本申请案同一天申请的标题为“Adaptive Metablocks”的共同待决并共同拥有的美国专利申请案中揭示了具有可变尺寸的元区块。所述共同待决的申请案的全部揭示内容以引用的方式并入本文中。

本发明的一个特征为，系统以单一逻辑分割操作，和存储器系统的整个逻辑地址范围中的逻辑扇区群组均以相同的方式进行处理。例如，可将含有系统数据的扇区和含有用户数据的扇区分布在逻辑地址空间中的任何地方。

与现有技术系统不同，并不为了将可能含有高频率和小尺寸更新的数据的扇区局限在逻辑地址空间中而将系统扇区（即，与文件配置表、目录或子目录有关的扇区）特别地进行分割或分区。实际上，更新扇区的逻辑群组的本方案将有效地处理典型的系统扇区的存取样式以及典型的文件数据的存取样式。

图4说明元区块与物理存储器中的结构的对准。快闪存储器包含可作为单位而一起擦除的存储器单元的区块。此擦除区块是快闪存储器的最小擦除单位或存储器的最小可擦除单位（MEU）。最小擦除单位是存储器的硬件设计参数，但在支持多个MEU擦除的一些存储器系统中，可能配置包含一个以上MEU的“超级MEU”。对于快闪EEPROM，MEU可包含一个扇区，但优选地包含多个扇区。在所示的实例中，其具有M个扇区。在优选的实施例中，每一扇区可存储512字节的数据且具有用户数据部分和用于存储系统或额外开销数据的标题部分。如果元区块由P个MEU构成，且每一MEU含有M个扇区，

那么每一元区块将具有N = P*M个扇区。

在系统层级上，元区块代表存储器位置的群组，例如，可一起擦除的扇区。将快闪存储器的物理地址空间视为一组元区块，元区块是最小的擦除单位。本说明书中，术语“元区块”和“区块”用来同义地定义在系统层级上媒体管理的最小擦除单位，且术语“最小擦除单位”或MEU用来表示快闪存储器的最小擦除单位。

链接最小擦除单位（MEU）以形成元区块

为了使编程速度和擦除速度最大化，尽可能利用并行方式：对将要并行编程的处于多个MEU中的多页信息和将要并行擦除的多个MEU进行排列。

在快闪存储器中，页是可在单一操作中一起编程的存储器单元的编组。页可包含一个或一个以上扇区。并且，可将存储器阵列分割成一个以上平面，其中一次只可编程或擦除平面内的一个MEU。最后，可将平面分布在一个或一个以上存储器芯片中。

在快闪存储器中，MEU可包含一个或一个以上页。可将快闪存储器芯片内的MEU组织在平面中。由于可同时编程或擦除每一平面的一个MEU，所以通过从每一平面选择一个MEU来形成多个MEU元区块是有利的（参看下文图5B）。

图5A说明通过链接不同平面的最小擦除单位而构成的元区块。每一元区块（例如，MB0、MB1、...）由来自存储器系统的不同平面的MEU构成，其中不同平面可分布在一个或一个以上芯片中。图2所示的元区块链接管理器170管理每一元区块的MEU的链接。除非MEU中的一者失败，否则在初始格式化过程期间配置每一元区块，并在系统的整个寿命期间保留其组成MEU。

图5B说明从每一平面选择一个最小擦除单位（MEU）以便链接成元区块的一个实施例。

图5C说明从每一平面选择一个以上MEU以便链接成元区块的另一实施例。在另一实施例中，可从每一平面选择一个以上MEU以形成超级MEU。例如，超级MEU可由两个MEU形成。在此情况下，可能采取一次以上的过程来进行读取或写入操作。

由Carlos Gonzales等人在与本申请案同一天申请的标题为“Adaptive Deterministic Grouping of Blocks into Multi-Block Structures”的共同待决并共同拥有的美国专利申请案中也揭示了将MEU链接和再链接成元区块。所述共同待决的申请案的全部揭示内容以引用的方式并入本文中。

元区块管理

图6是如控制器和快闪存储器中实施的元区块管理系统的示意方框图。元区块管理系统包含实施在控制器100中的各种功能模块，并将各种控制数据（包括目录数据）维

持在依照阶层而分布在快闪存储器200和控制器RAM 130中的表和列表中。实施在控制器100中的功能模块包括：接口模块110、逻辑到物理地址转换模块140、更新区块管理器模块150、擦除区块管理器模块160和元区块链接管理器170。

接口110允许元区块管理系统与主机系统进行接口连接。逻辑到物理地址转换模块140将来自主机的逻辑地址映射到物理存储器位置。更新区块管理器模块150管理存储器中给定的数据逻辑群组的数据更新操作。已擦除区块管理器160管理元区块的擦除操作及其用于存储新信息的配置。元区块链接管理器170管理扇区的最小可擦除区块的子群组的链接以构成给定的元区块。这些模块的具体实施方式将在其个别段落中给出。

在操作期间，元区块管理系统产生控制数据（例如，地址、控制与状态信息）并结合控制数据来工作。由于许多控制数据倾向于是经常变化的小型数据，所以不可容易地有效地存储并维持在具有大区块结构的快闪存储器中。为了在非易失性快闪存储器中存储较静态的控制数据，同时在控制器RAM中定位较少数目的较有变化的控制数据，以便进行更有效的更新和存取，使用阶层式与分散式方案。在电源关机或故障的事件中，此方案允许通过扫描非易失性存储器中的一小组的控制数据来在快速地重建易失性控制器RAM中的控制数据。这是可能的，因为本发明限制与给定的逻辑群组的数据的可能的活动关联的区块数目。以此方式限制扫描。另外，将需要持久性的一些控制数据存储在可逐扇区更新的非易失性元区块中，每一更新导致记录取代先前扇区的新扇区。为控制数据使用扇区索引方案以便在元区块中追踪逐扇区的更新。

非易失性快闪存储器200存储相对较静态的大量控制数据。此包括：群组地址表（GAT）210、混乱区块索引（CBI）220、已擦除区块列表（EBL）230和MAP 240。GAT 210追踪扇区的逻辑群组与其相应元区块之间的映射。除非经历更新，否则映射不会改变。CBI 220追踪更新期间逻辑上非循序的扇区的映射。EBL 230追踪已擦除的元区块的集区（pool）。MAP 240是展示快闪存储器中所有元区块的擦除状态的位图。

易失性控制器RAM 130存储经常变化并被存取的一小部分控制数据。此包括配置区块列表（ABL）134和已清除区块列表（CBL）136。ABL 134追踪用于记录更新数据的元区块配置，而CBL 136追踪已解除配置并擦除的元区块。在优选实施例中，RAM 130充当存储在快闪存储器200中的控制数据的高速缓冲存储器。

更新区块管理器

更新区块管理器150（如图2所示）处理逻辑群组的更新。根据本发明的一个方面，向经历更新的扇区的每一逻辑群组配置用于记录更新数据的专用更新元区块。在优选实施例中，将逻辑群组的一个或一个以上扇区的任何程序段记录在更新区块中。可管理更

新区块以接收循序次序或非循序（也称为“混乱”）次序的已更新数据。混乱更新区块允许在逻辑群组内以任何次序更新扇区数据，并任意重复个别扇区。明确地说，循序的更新区块可变成混乱更新区块而不需要重新定位任何数据扇区。对于混乱数据更新不需要预定的区块配置；任何逻辑地址的非循序写入自动地适应。因此，与现有技术系统不同，关于逻辑群组的各种更新程序段是以逻辑循序还是非循序次序并不存在特殊处理。一般更新区块将仅仅用来以主机请求的次序记录各种程序段。例如，即使主机系统数据或系统控制数据倾向于以混乱方式更新，但不需要以与具有主机用户数据的区域不同的方式来处理逻辑地址空间的对应于主机系统数据的区域。

优选地将扇区的完整逻辑群组的数据以逻辑上循序次序存储在单一元区块中。以此方式，预定义已存储的逻辑扇区的索引。当元区块以预定义的次序存储给定逻辑群组的所有扇区时，其可称为“完整的”。对于更新区块，当其最终以逻辑上循序次序用更新数据填满时，接着更新区块将变成容易地取代原始元区块的已更新的完整元区块。另一方面，如果更新区块以逻辑上与完整区块不同的次序被更新数据填满，那么更新区块为非循序或混乱更新区块且必须进一步处理次序紊乱的程序段从而最终以与完整区块的次序相同的次序存储逻辑群组的更新数据。在优选的情况下，单一元区块中为逻辑上循序次序。进一步的处理涉及将更新区块中的已更新扇区和原始区块中的未变化的扇区合并到又一更新元区块中。接着，合并的更新区块将为逻辑上循序次序并可用来取代原始区块。在某一预定条件下，合并过程之前进行一个或一个以上压缩过程。压缩过程仅仅将混乱更新区块的扇区重新记录到取代的混乱更新区块中，同时除去已被相同逻辑扇区的后续更新致使过时的任何复制的逻辑扇区。

更新方案允许多达预定义的最大值的多个更新线程同时运行。每一线程是使用其专用更新元区块经历更新的逻辑群组。

循序数据更新

当首先更新属于逻辑群组的数据时，元区块经配置并专用作逻辑群组的更新数据的更新区块。当从主机接收写入逻辑群组（现有的元区块已完整地存储其所有扇区）的一个或一个以上扇区的程序段的命令时，配置更新区块。对于第一主机写入操作，将第一程序段的数据记录在更新区块上。由于每一主机写入是具有连续逻辑地址的一个或一个以上扇区的程序段，所以遵循第一更新在性质上始终循序。在后续的主机写入中，以从主机接收的次序将相同逻辑群组内的更新程序段记录在更新区块中。区块继续接受管理作为循序的更新区块，同时在关联的逻辑群组内由主机更新的扇区保持逻辑上循序。在此逻辑群组中更新的所有扇区写入到此循序的更新区块直到区块关闭或转换为混乱更

新区块为止。

图7A说明由于两个分离的主机写入操作而以循序次序写入到循序的更新区块的逻辑群组中的扇区的实例，同时逻辑群组的原始区块中的相应扇区变得过时。在主机写入操作#1中，更新逻辑扇区LS5-LS8中的数据。已更新为LS5'-LS8'的数据记录在新配置的专用更新区块中。

为了方便，将逻辑群组中将要更新的第一扇区记录在始于第一物理扇区位置的专用更新区块中。一般而言，将要更新的第一逻辑扇区不一定是群组的逻辑第一扇区，且因此逻辑群组的起始与更新区块的起始之间可能存在偏移。此偏移称为页标记，如先前结合图3A所描述。以逻辑上循序次序更新后续的扇区。当写入逻辑群组的最后扇区时，群组地址卷绕且写入序列从群组的第一扇区继续。

在主机写入操作#2中，更新逻辑扇区LS9-LS12中数据的程序段。更新为LS9'-LS12'的数据记录在直接跟随着写入结束处的位置中的专用更新区块中。可看到两个主机写入使得更新数据已以逻辑上循序次序记录在更新区块中，即LS5'-LS12'。更新区块可视为循序的更新区块，因为其已以逻辑上循序次序被填充。记录在更新区块中的更新数据使原始区块中的相应数据过时。

混乱数据更新

当关联的逻辑群组内由主机更新的任何扇区为逻辑上非循序时，为现有的循序的更新区块起始混乱更新区块管理。混乱更新区块是数据更新区块的形式，其中关联的逻辑群组内的逻辑扇区可以任何次序进行更新并重复任意次。其可在主机写入的扇区是逻辑上非循序时通过从循序的更新区块转换成正被更新的逻辑群组内的先前已写入的扇区而得以建立。随后在此逻辑群组中更新的所有扇区写入在混乱更新区块中的下一可用扇区位置，而无论其在群组内的逻辑扇区地址如何。

图7B说明由于五个分离的主机写入操作而正以混乱次序写入到混乱更新区块的逻辑群组中的扇区的实例，同时逻辑群组的原始区块中被取代的扇区和混乱更新区块中被复制的扇区变得过时。在主机写入操作#1中，更新存储在原始元区块中的给定逻辑群组的逻辑扇区LS10-LS11。已更新的逻辑扇区LS10'-LS11'存储在新配置的更新区块中。此时，更新区块为循序的更新区块。在主机写入操作#2中，将逻辑扇区LS5-LS6更新为LS5'-LS6'并记录在紧紧跟随着上一个写入的位置中的更新区块中。此可使更新区块从循序的转换为混乱的更新区块。在主机写入操作#3中，再次更新逻辑扇区LS10并将其记录在更新区块的下一位置作为LS10"。此时，更新区块中的LS10"取代先前记录中的LS10'，而LS10'进而取代原始区块中的LS10。在主机写入操作#4中，再次更新逻辑扇区LS10中

的数据并将其记录在更新区块的下一位置作为LS10'''。因此，LS10'''现为逻辑扇区LS10的最后且唯一有效的数据。在主机写入操作#5中，更新逻辑扇区LS30中的数据并将其记录在更新区块中作为LS30'。因此，此实例说明可以任何次序并以任意重复，将逻辑群组内的逻辑扇区写入在混乱更新区块中。

强制循序更新

图8说明由于具有逻辑地址不连续性的两个分离的主机写入操作而正以循序次序写入到循序更新区块的逻辑群组中的扇区的实例。在主机写入#1中，将逻辑扇区LS5-LS8中的更新数据记录在专用更新区块中作为LS5'-LS8'。在主机写入#2中，将逻辑扇区LS14-LS16中的更新数据记录在跟随着上一个写入的更新区块中作为LS14'-LS16'。然而，LS8与LS14之间存在地址跳跃，且主机写入#2通常将致使更新区块非循序。由于地址跳跃并不很多，所以一个选择是在执行主机写入#2之前通过将来自原始区块的介入的扇区的数据复制到更新区块而首先执行填补操作（#2A）。以此方式，保持更新区块的循序性质。

图9是说明根据本发明的一般实施例更新区块管理器更新逻辑群组的数据的过程的流程图。更新过程包含以下步骤：

步骤260：将存储器组织成区块，将每一区块分割成可一起擦除的存储器单位，每一存储器单位用于存储一逻辑单位的数据。

步骤262：将数据组织成逻辑群组，将每一逻辑群组分割成逻辑单位。

步骤264：在标准情况下，根据第一指定次序，优选地以逻辑上循序次序，将逻辑群组的所有逻辑单位存储在原始区块的存储器单位中。以此方式，得知存取区块中个别逻辑单位的索引。

步骤270：对于给定逻辑群组（例如，LG_x）的数据，请求更新LG_x内的逻辑单位。（逻辑单位更新作为实例而给出。一般而言，更新将是LG_x内一个或一个以上连续逻辑单位的程序段。）

步骤272：所请求的更新逻辑单位将存储在专用于记录LG_x的更新的第二区块中。记录次序根据第二次序，通常是请求更新的次序。本发明的一个特征允许将更新区块初始地设定成通用于以逻辑上循序或混乱次序记录数据。因此根据第二次序，第二区块可为循序的区块或混乱的区块。

步骤274：当过程按回路返回到步骤270时，第二区块继续记录所请求的逻辑单位。在关闭的预定条件实现时，将关闭第二区块以不接收进一步的更新。在此情况下，过程继续进行到步骤276。

步骤276：确定已关闭的第二区块是否以与原始区块类似的次序记录其更新逻辑单位。当两个区块记录逻辑单位仅相差一页标记时，认为两个区块具有类似次序，如结合图3A所描述。如果两个区块具有类似次序，那么过程继续进行到步骤280，否则，需要在步骤290中执行稍许垃圾收集。

步骤280：由于第二区块具有与第一区块相同的次序，所以其可用来取代原始的第一区块。接着，更新过程结束于步骤299。

步骤290：从第二区块（更新区块）和第一区块（原始区块）中搜集给定逻辑群组的每一逻辑单位的最新版本。接着以与第一区块类似的次序将给定逻辑群组的已合并的逻辑单位写入到第三区块。

步骤292：由于第三区块（已合并的区块）具有与第一区块类似的次序，所以其可用来取代原始的第一区块。接着，更新过程结束于步骤299。

步骤299：当停闭过程建立完整的更新区块时，其变成给定逻辑群组的新标准区块。逻辑群组的更新线程将终止。

图10是说明根据本发明的优选实施例，更新区块管理器更新逻辑群组的数据的过程的流程图。更新过程包含以下步骤：

步骤310：对于给定逻辑群组（例如，LG_x）的数据，请求更新LG_x内的逻辑扇区。（扇区更新作为实例而给出。一般而言，更新将是LG_x内的一个或一个以上连续逻辑扇区的程序段。）

步骤312：如果专用于LG_x的更新区块尚未存在，那么继续进行到步骤410以起始逻辑群组的新的更新线程。此将通过配置专用于记录逻辑群组的更新数据的更新区块而实现。如果已存在开启的更新区块，那么继续进行到步骤314以开始将更新扇区记录到更新区块上。

步骤314：如果当前更新区块已混乱（即，非循序），那么直接继续进行到步骤510，以便将所请求的更新扇区记录到混乱更新区块上。如果当前更新区块为循序的，那么继续进行到步骤316以便处理循序的更新区块。

步骤316：本发明的一个特征允许于将更新区块初始地设定成通用于以逻辑上循序或混乱次序记录数据。然而，由于逻辑群组最终将其数据以逻辑上循序次序存储在元区块中，所以希望尽可能将更新区块保持为循序的。接着，当关闭更新区块以进行进一步更新时，将需要较少的处理，因为将不需要进行垃圾收集。

因此，确定所请求的更新是否将遵循更新区块的当前循序次序。如果更新循序进行，那么继续进行到步骤510以执行循序更新，且更新区块将维持循序。另一方面，如果更

新不循序进行（混乱更新），那么其将在未采取其它行动的情况下将循序更新区块转换成混乱更新区块。

在一个实施例中，不会进行更多的行动来挽救此情况，且过程直接进行到步骤370，其中允许更新将更新区块变成混乱更新区块。

任选的强制循序过程

在另一实施例中，任选地执行强制循序过程步骤320以顾及到待决的混乱更新而尽可能保存循序更新区块。存在两种情况，其两者均需要复制原始区块的遗失扇区以维持记录在更新区块上的逻辑扇区的循序次序。第一种情况是更新建立短的地址跳跃。第二种情况是提早结束更新区块以将其保持为循序。强制循序过程步骤320包含以下子步骤：

步骤330：如果更新建立的逻辑地址跳跃不大于预定的量C_B，那么过程继续进行到步骤350的强制循序更新过程，否则过程继续进行到步骤340以考虑其是否适合进行强制循序停闭。

步骤340：如果未填充的物理扇区数目超过预定的设计参数C_C（其典型值为更新区块尺寸的一半），那么更新区块相对地未被使用且将不提早关闭。过程继续进行到步骤370，且更新区块将变得混乱。另一方面，如果实质上填充了更新区块，那么认为其已被充分利用且因此进入步骤360以进行强制循序停闭。

步骤350：只要地址跳跃不超过预定的量C_B，强制循序更新允许当前循序更新区块维持循序。实质上，复制更新区块的关联的原始区块的扇区以填充地址跳跃跨越的间隙。因此，在继续进行到步骤510之前，将用介入的地址中的数据填补循序更新区块以便循序记录当前的更新。

步骤360：如果当前循序更新区块实质上已被填充而不是由待决的混乱更新转换为混乱更新区块，那么强制循序停闭允许当前循序的更新区块结束。将混乱或非循序更新定义为具有未被上述地址跳跃例外涵盖的前向地址转换、后向地址转换或地址重复的更新。为了防止循序更新区块被混乱更新转换，通过复制更新区块的关联的原始部分过时区块的扇区来填充更新区块的未写入的扇区位置。接着原始区块完全过时且被擦除。现在，当前更新区块具有完整组的逻辑扇区，且接着结束而作为取代原始元区块的完整的元区块。接着，过程继续进行到步骤430以在其位置中配置新的更新区块，从而接受在步骤310中首先请求的待决的扇区更新的记录。

转换成混乱更新区块

步骤370：当待决的更新不以循序次序且为任选的时，如果无法满足强制循序条件，那么在过程继续进行到步骤510时，通过允许在更新区块上记录具有非循序地址的待决

更新扇区允许将循序更新区块转换为混乱更新区块。如果存在最大数目的混乱更新区块，那么在允许转换继续进行之前，必须关闭最久未存取的混乱更新区块；因此防止超过混乱区块的最大数目。最久未存取的混乱更新区块的识别与步骤420中所描述的一般情况相同，但仅限于混乱更新区块。此时可通过如步骤550中所描述的合并来实现关闭混乱更新区块。

配置受到系统限制的新的更新区块

步骤410：将擦除元区块配置为更新区块的过程始于确定是否超过预定的系统限制。由于资源有限，存储器管理系统通常允许同时存在更新区块的预定最大数目C_A。此限制为循序更新区块和混乱更新区块的总数，且为设计参数。在优选实施例中，此限制为（例如）最大8个更新区块。并且，由于对系统资源的较高需求，对可同时开启的混乱更新区块的最大数目也可能存在相应的预定限制（例如，4）。

因此，在已配置C_A个更新区块时，接着只有在关闭现有的已配置的请求中的一者之后才可满足下一配置请求。过程继续进行到步骤420。当开启的更新区块的数目小于C_A时，过程直接进行到步骤430。

步骤420：在超过更新区块的最大数目C_A的事件中，关闭最久未存取的更新区块并执行垃圾收集。将最久未存取的更新区块识别为与最久未存取的逻辑区块关联的更新区块。为了确定最久未存取的区块，存取包括逻辑扇区的写入和选择性读取。以存取的次序维持开启的更新区块的列表；初始化时，不假设存取次序。在更新区块为循序时，更新区块的关闭遵循结合步骤360和步骤530而描述的类似过程，且在更新区块为混乱时，遵循结合步骤540而描述的类似过程。此关闭可为在步骤430中配置新的更新区块创造空间。

步骤430：通过配置新的元区块作为专用于给定逻辑群组LG_X的更新区块来满足配置请求。接着，过程继续进行到步骤510。

将更新数据记录到更新区块上

步骤510：将所请求的更新扇区记录到更新区块的下一可用物理位置上。接着，过程继续进行到步骤520以确定更新区块是否准备就绪即可停闭。

更新区块停闭

步骤520：如果更新区块仍具有接受额外更新的空间，那么继续进行到步骤570。否则继续进行到步骤522以停闭更新区块。在当前所请求的写入尝试写入多于区块的空间能够容纳的逻辑扇区时，存在填充更新区块的两个可能的实施方案。在第一实施方案中，将写入请求分成两个部分，第一部分写入直到区块的最后物理扇区。接着关闭区块且写

入的第二部分将视为下一请求的写入。在另一实施方案中，抑制所请求的写入，同时区块填补它的其余扇区并接着关闭。所请求的写入将视为下一请求的写入。

步骤522：如果更新区块为循序的，那么继续进行到步骤530以进行循序关闭。如果更新区块为混乱的，那么继续进行到步骤540以进行混乱关闭。

循序更新区块停闭

步骤530：由于更新区块为循序的且已完全填充，所以存储在其中的逻辑群组为完整的。元区块为完整的并取代原始的元区块。此时，原始区块完全过时且可被擦除。接着，过程继续进行到步骤570，其中给定逻辑群组的更新线程结束。

混乱更新区块停闭

步骤540：由于更新区块被非循序填充且可能含有一些逻辑扇区的多个更新，所以执行垃圾收集以挽救其中的有效数据。混乱更新区块将为已压缩或已合并。在步骤542中将确定执行哪一过程。

步骤542：为了执行压缩或合并将视更新区块的退化而定。如果逻辑扇区已更新多次，那么其逻辑地址高度退化。将有相同逻辑扇区的多个版本记录在更新区块上，且只有最后记录的版本为逻辑扇区的有效版本。在含有多个版本的逻辑扇区的更新区块中，截然不同的逻辑扇区的数目将大大少于逻辑群组的数目。

在优选实施例中，当更新区块中截然不同的逻辑扇区的数目超过预定的设计参数C_D（其典型值为逻辑群组尺寸的一半）时，停闭过程将在步骤550中执行合并，否则过程将继续进行到步骤560进行压缩。

步骤550：如果将要合并混乱更新区块，那么将以含有已合并数据的新的标准元区块取代原始区块和更新区块。合并之后，更新线程将结束于步骤570。

步骤560：如果将要压缩混乱更新区块，那么其将由载有已压缩数据的新的更新区块取代。压缩之后，已压缩的更新区块的处理将结束于步骤570。或者，可将压缩延迟直到再次对更新区块进行写入为止，因此排除压缩之后进行没有介入的更新的合并的可能性。接着，当步骤502中出现在LG_X中进行更新的下一请求时，将在给定逻辑区块的进一步更新中使用新的更新区块。

步骤570：当停闭过程建立完整的更新区块时，所述区块变成给定逻辑群组的新标准区块。逻辑群组的更新线程将终止。在停闭过程建立取代现有更新区块的新的更新区块时，将使用新的更新区块来记录为给定逻辑群组请求的下一更新。在更新区块未结束时，过程将在步骤310中出现在LG_X中进行更新的下一请求时继续。

从上述过程可知，在关闭混乱更新区块时，进一步处理记录在其上的更新数据。明

确地说，通过压缩到另一个混乱区块的过程，或通过和其关联的原始区块一起合并以形成新的标准循序区块的过程，对其有效数据进行垃圾收集。

图11A为更详细地说明关闭图10所示的混乱更新区块的合并过程的流程图。混乱更新区块合并是在更新区块正结束时（例如，当更新区块已填满其写入的最后物理扇区位置时）执行的两个可能过程的一者。当区块中写入的截然不同的逻辑扇区数目超过预定的设计参数C_D时，选择进行合并。图10所示的合并过程步骤550包含以下子步骤：

步骤551：在混乱更新区块正关闭时，将配置取代其的新的元区块。

步骤552：在混乱更新区块及其关联的原始区块中搜集每一逻辑扇区的最新版本并忽略所有的过时扇区。

步骤554：将所搜集的有效扇区以逻辑上循序次序记录在新的元区块上以形成完整的区块，即，以循序次序记录的逻辑群组的所有逻辑扇区的区块。

步骤556：以新的完整区块取代原始区块。

步骤558：擦除已停闭的更新区块和原始区块。

图11B为更详细地说明关闭图10所示的混乱更新区块的压缩过程的流程图。当区块中写入的截然不同的逻辑扇区数目低于预定的设计参数C_D时，选择进行压缩。图10所示的压缩过程步骤560包含以下子步骤：

步骤561：在混乱更新区块正被压缩时，将配置取代其的新的元区块。

步骤562：在将要压缩的现有的混乱更新区块中搜集每一逻辑扇区的最新版本。

步骤564：将所搜集的扇区记录在新的更新区块上以形成具有已压缩扇区的新的更新区块。

步骤566：以具有已压缩扇区的新的更新区块取代现有的更新区块。

步骤568：擦除已停闭的更新区块。

逻辑和元区块状态

图12A说明逻辑群组的所有可能状态，和在各种操作下其间的可能的转换。

图12B是罗列逻辑群组的可能状态的表。逻辑群组状态定义如下：

1. 完整：可能已使用页标记卷绕方式以逻辑上循序次序将逻辑群组中的所有逻辑扇区写入在单一元区块中。

2. 未写入：逻辑群组中未曾写入逻辑扇区。逻辑群组在群组地址表中标示为未写入且不具有已配置的元区块。响应于此群组内每一扇区的主机读取而返回预定的数据样式。

3. 循序更新：可能已使用页标记以逻辑上循序次序将逻辑群组内的一些扇区写入在

元区块中，因此其取代群组的任何先前完整状态的相应的逻辑扇区。

4. 混乱更新：可能已使用页标记以逻辑上非循序次序将逻辑群组内的一些扇区写入在元区块中，因此其取代群组的任何先前完整状态的相应的逻辑扇区。群组内的扇区可被写入一次以上，最新版本取代所有先前版本。

图13A说明元区块的所有可能状态，和在各种操作下其间的可能的转换。

图13B是罗列元区块的可能状态的表。元区块状态定义如下：

1. 已擦除：元区块中的所有扇区已被擦除。

2. 循序更新：可能已使用页标记部分地写入元区块，其扇区为逻辑上循序次序。所有扇区均属于相同的逻辑群组。

3. 混乱更新：已部分或完全写入了元区块，其扇区为逻辑上非循序次序。任何扇区均可被写入一次以上。所有扇区均属于相同的逻辑群组。

4. 完整：可能已使用页标记以逻辑上循序次序完全写入了元区块。

5. 原始：元区块先前为完整的但至少一个扇区已因主机数据更新而过时。

图14（A）到图14（J）是展示对逻辑群组的状态以及物理元区块的各种操作的效果的状态图。

图14（A）展示对应于第一写入操作的逻辑群组和元区块转换的状态图。主机以逻辑上循序次序将先前未写入的逻辑群组的一个或一个以上扇区写入到新配置的已擦除元区块。逻辑群组和元区块进入循序更新状态。

图14（B）展示对应于第一完整操作的逻辑群组和元区块转换的状态图。先前未写入的循序更新逻辑群组因主机循序写入所有扇区而变成完整的。如果卡通过以预定的数据样式填充其余未写入的扇区来填满群组，那么也可发生转换。元区块变得完整。

图14（C）展示对应于第一混乱操作的逻辑群组和元区块转换的状态图。先前未写入的循序更新逻辑群组在主机已非循序地写入至少一个扇区时变得混乱。

图14（D）展示对应于第一压缩操作的逻辑群组和元区块转换的状态图。从旧区块将先前未写入的混乱更新逻辑群组内的所有有效扇区复制到新的混乱元区块，接着将其擦除。

图14（E）展示对应于第一合并操作的逻辑群组和元区块转换的状态图。从旧的混乱区块移动先前未写入的混乱更新逻辑群组内的所有有效扇区，以便以逻辑上循序次序填充新配置的已擦除区块。以预定的数据样式填充主机未写入的扇区。接着擦除旧的混乱区块。

图14（F）展示对应于循序写入操作的逻辑群组和元区块转换的状态图。主机以逻

辑上循序次序将完整逻辑群组的一个或一个以上扇区写入到新配置的已擦除元区块。逻辑群组和元区块进入循序更新状态。先前完整的元区块变成原始元区块。

图14（G）展示对应于循序填充操作的逻辑群组和元区块转换的状态图。循序更新逻辑群组在主机循序写入其所有扇区时变得完整。此也可发生在用来自原始区块的有效扇区填充循序更新逻辑群组以使其完整的垃圾收集期间，在此之后擦除原始区块。

图14（H）展示对应于非循序写入操作的逻辑群组和元区块转换的状态图。循序更新逻辑群组在主机非循序地写入至少一个扇区时变得混乱。非循序扇区写入可促使更新区块或相应的原始区块中的有效扇区变得过时。

图14（I）展示对应于压缩操作的逻辑群组和元区块转换的状态图。从旧区块将混乱更新逻辑群组内的所有有效扇区复制到新的混乱元区块，接着将其擦除。原始区块不受影响。

图14（J）展示对应于合并操作的逻辑群组和元区块转换的状态图。从旧的混乱区块复制混乱更新逻辑群组内的所有有效扇区，以便以逻辑上循序次序填充新配置的已擦除区块。接着擦除旧的混乱区块和原始区块。

更新区块追踪和管理

图15说明用于追踪已开启和已关闭的更新区块及已擦除区块以进行配置的配置区块列表（ABL）的结构的优选实施例。配置区块列表（ABL）610保存在控制器RAM 130中以允许管理已擦除区块、已配置的更新区块、关联的区块和控制结构的配置，并使得能够进行正确的逻辑到物理地址转换。在优选实施例中，ABL包括已擦除区块的列表、开启的更新区块列表614、和关闭的更新区块列表616。

开启的更新区块列表614是ABL中具有开启更新区块的属性的区块条目组。开启的更新区块列表具有当前开启的每一数据更新区块的一个条目。每一条目保存以下信息。LG是当前更新元区块专用的逻辑群组地址。循序/混乱是表示用循序或混乱更新数据填充更新区块的状态。MB是更新区块的元区块地址。页标记是记录在更新区块的第一物理位置处的起始逻辑扇区。写入的扇区号码表示当前写入到更新区块上的扇区号码。MB_0是关联的原始区块的元区块地址。Page Tag_0是关联的原始区块的页标记。

关闭的更新区块列表616是配置区块列表（ABL）的子集。其为ABL中具有关闭的更新区块的属性的区块条目组。关闭的更新区块列表具有已关闭的每一数据更新区块的一个条目，但其条目在逻辑到主物理目录中尚未更新。每一条目保存以下信息。LG是当前更新区块专用的逻辑群组地址。MB是更新区块的元区块地址。页标记是记录在更新区块的第一物理位置处的起始逻辑扇区。MB_0是关联的原始区块的元区块地址。

混乱区块索引

循序更新区块具有以逻辑上循序次序存储的数据，因此可容易地定位区块中的任何逻辑扇区。混乱更新区块具有其以紊乱的次序存储的逻辑扇区且也可存储逻辑扇区的多个更新世代。必须维持额外信息以追踪每一有效逻辑扇区定位在混乱更新区块中的位置。

在优选实施例中，混乱区块索引数据结构允许追踪并快速存取混乱区块中的所有有效扇区。混乱区块索引独立地管理小区域的逻辑地址空间，并有效地处理系统数据和用户数据的热区（hot region）。索引数据结构实质上允许在快闪存储器中维持具有不常更新需求的索引信息，从而性能不会明显受到影响。另一方面，将混乱区块中最近写入扇区的列表保存在控制器RAM中的混乱扇区列表中。并且，将来自快闪存储器的索引信息的高速缓冲存储器保存在控制器RAM中，以便减少用于地址转换的快闪扇区存取的数目。每一混乱区块的索引存储在快闪存储器中的混乱区块索引（CBI）扇区中。

图16A说明混乱区块索引（CBI）扇区的数据字段。混乱区块索引扇区（CBI扇区）在逻辑群组中含有每一扇区的映射到混乱更新区块的索引，从而界定逻辑群组的每一扇区在混乱更新区块或其关联的原始区块内的位置。CBI扇区包括：用于追踪混乱区块内有效扇区的混乱区块索引字段、用于追踪混乱区块的地址参数的混乱区块信息字段、和用于追踪存储CBI扇区的元区块（CBI区块）内的有效CBI扇区的扇区索引字段。

图16B说明记录在专用元区块中的混乱区块索引（CBI）扇区的实例。专用的元区块将称为CBI区块620。在更新CBI扇区时，将其写入在CBI区块620中下一可用的物理扇区位置。因此，CBI扇区的多个副本可存在于CBI区块中，其中只有最后写入的副本为有效的。例如，已用作为有效版本的最新版本将逻辑群组LG_1的CBI扇区更新三次。区块中最后写入的CBI扇区中的一组索引识别CBI区块中每一有效扇区的位置。在此实例中，区块中最后写入的CBI扇区是LG_{136}的CBI扇区，且其索引组是取代所有先前索引组的有效索引组。当CBI区块最终变成用CBI扇区完全填充时，通过将所有有效扇区重写到新的区块位置，在控制写入操作期间压缩区块。接着擦除整个区块。

CBI扇区内的混乱区块索引字段含有逻辑群组或自群组内的每一逻辑扇区的映射到混乱更新区块的索引条目。每一索引条目表示相应的逻辑扇区的有效数据所在的混乱更新区块内的偏移。保留的索引值表示混乱更新区块中不存在逻辑扇区的有效数据，且关联的原始区块中的相应扇区为有效的。一些混乱区块索引字段条目的高速缓冲存储器保存在控制器RAM中。

CBI扇区内的混乱区块信息字段含有存在于系统中的每一混乱更新区块的一个条

目，从而记录区块的地址参数信息。此字段中的信息只在CBI区块中的最后写入的扇区中有效。此信息也出现在RAM中的数据结构中。

每一混乱更新区块的条目包括三个地址参数。第一地址参数是与混乱更新区块关联的逻辑群组（或逻辑群组号码）的逻辑地址。第二地址参数是混乱更新区块的元区块地址。第三地址参数是写入在混乱更新区块中的最后的扇区的物理地址偏移。偏移信息设定初始化期间混乱更新区块的扫描起点以在RAM中重建数据结构。

扇区索引字段含有CBI区块中的每一有效CBI扇区的条目。其界定CBI区块内有关每一许可的混乱更新区块的最近写入的CBI扇区所在的偏移。索引中偏移的保留值表示许可的混乱更新区块并不存在。

图16C为说明存取正在进行混乱更新的给定逻辑群组的逻辑扇区的数据的流程图。在更新过程期间，将更新数据记录在混乱更新区块中，同时未改变的数据保留在与逻辑群组关联的原始元区块中。在混乱更新下存取逻辑群组的逻辑扇区的过程如下：

步骤650：开始定位给定逻辑群组的给定逻辑扇区。

步骤652：在CBI区块中定位最后写入的CBI扇区。

步骤654：通过查找最后写入的CBI扇区的混乱区块信息字段来定位与给定逻辑群组关联的混乱更新区块或原始区块。此步骤可在步骤662之前的任何时间执行。

步骤658：如果最后写入的CBI扇区指向给定的逻辑群组，那么定位CBI扇区。继续进行到步骤662。否则，继续进行到步骤660。

步骤660：通过查找最后写入的CBI扇区的扇区索引字段来定位给定逻辑群组的CBI扇区。

步骤662：通过查找已定位的CBI扇区的混乱区块索引字段来定位混乱区块或原始区块中的给定逻辑扇区。

图16D是说明根据已将逻辑群组分割成子群组的替代实施例，存取正进行混乱更新的给定逻辑群组的逻辑扇区的数据的流程图。CBI扇区的有限容量仅可追踪预定最大数目的逻辑扇区。当逻辑群组具有多于单一CBI扇区可处理的逻辑扇区时，将逻辑群组分割成多个子群组，向每一子群组配置CBI扇区。在一个实例中，每一CBI扇区的容量足以追踪由256个扇区组成的逻辑群组和多达8个混乱更新区块。如果逻辑群组的尺寸超过256个扇区，那么存在用于逻辑群组内每一256扇区子群组的分离的CBI扇区。CBI扇区可存在而用于逻辑群组内多达8个子群组，从而支持尺寸为多达2048个扇区的逻辑群组。

在优选实施例中，使用间接索引方案来促进索引的管理。扇区索引的每一条目具有直接和间接字段。

直接扇区索引可界定CBI区块内关于特定混乱更新区块的所有可能CBI扇区所在的偏移。此字段中的信息仅在关于所述特定混乱更新区块的最后写入的CBI扇区中有效。索引中偏移的保留值表示CBI扇区并不存在，因为关于混乱更新区块的相应的逻辑子群组也不存在，或由于配置了更新区块而尚未更新。

间接扇区索引界定CBI区块内的关于每一许可的混乱更新区块的最近写入的CBI扇区所在的偏移。索引中偏移的保留值表示许可的混乱更新区块并不存在。

图16D展示在混乱更新下存取逻辑群组的逻辑扇区的过程，如下：

步骤670：将每一逻辑群组分割成多个子群组并向每一子群组配置CBI扇区。

步骤680：开始定位给定逻辑群组的给定子群组的给定逻辑扇区。

步骤682：在CBI区块中定位最后写入的CBI扇区。

步骤684：通过查找最后写入的CBI扇区的混乱区块信息字段来定位与给定子群组关联的混乱更新区块或原始区块。此步骤可在步骤696之前的任何时间执行。

步骤686：如果最后写入的CBI扇区指向给定的逻辑群组，那么继续进行到步骤691。否则，继续进行到步骤690。

步骤690：通过查找最后写入的CBI扇区的间接扇区索引字段来定位给定逻辑群组的多个CBI扇区中最后写入的CBI扇区。

步骤691：已定位与给定逻辑群组的子群组的一者关联的至少一CBI扇区。继续。

步骤692：如果所定位的CBI扇区指向给定子群组，那么定位给定子群组的CBI扇区。继续进行到步骤696。否则，继续进行到步骤694。

步骤694：通过查找当前定位的CBI扇区的直接扇区索引字段来定位给定子群组的CBI扇区。

步骤696：通过查找给定子群组的CBI扇区的混乱区块索引字段来定位混乱区块或原始区块中的给定逻辑扇区。

图16E说明对于将每一逻辑群组分割成多个子群组的实施例，混乱区块索引（CBI）扇区及其功能的实例。逻辑群组700最初将其完整的数据存储在原始元区块702中。接着，逻辑群组配合配置专用的混乱更新区块704进行更新。在本实例中，将逻辑群组700分割成子群组，这些子群组A、B、C、D每一者具有256个扇区。

为了在子群组B中定位第i个扇区，首先定位CBI区块620中最后写入的CBI扇区。最后写入的CBI扇区的混乱区块信息字段提供定位给定逻辑群组的混乱更新区块704的地址。同时，其提供写入在混乱区块中的最后扇区的位置。此信息用于扫描和重建索引的事件中。

如果最后写入的CBI扇区结果是给定逻辑群组的四个CBI扇区的一者，那么将进一步确定其是否正是含有第i个逻辑扇区的给定子群组B的CBI扇区。如果是，那么CBI扇区的混乱区块索引将指向用于存储第i个逻辑扇区的数据的元区块位置。扇区位置可能在混乱更新区块704中或在原始区块702中。

如果最后写入的CBI扇区结果是给定逻辑群组的四个CBI扇区的一者但不恰好属于子群组B，那么查找其直接扇区索引以定位子群组B的CBI扇区。一旦定位了此确切的CBI扇区，查找混乱区块索引以在混乱更新区块704和原始区块702中定位第i个逻辑扇区。

如果最后写入的CBI扇区结果不是给定逻辑群组的四个CBI扇区的任一者，那么查找其间接扇区索引以定位四个CBI扇区中的一者。在图16E所示的实例中，定位了子群组C的CBI扇区。接着，查找子群组C的此CBI扇区的直接扇区索引以定位子群组B的确切的CBI扇区。此实例展示在查找混乱区块索引时，发现第i个逻辑扇区未改变，且将在原始区块中定位其有效数据。

在给定逻辑群组的子群组C中定位第j个逻辑扇区时进行类似的考虑。此实例展示最后写入的CBI扇区结果不是给定逻辑群组的四个CBI扇区的任何一者。其间接扇区索引指向给定群组的四个CBI扇区的一者。所指向的四个中的最后写入结果也正是子群组C的CBI扇区。在查找其混乱区块索引时，发现第j个逻辑扇区定位在混乱更新区块704中的指定位置。

控制器RAM中存在系统中每一混乱更新区块的混乱扇区列表。每一列表均含有自从相关CBI扇区在快闪存储器中最后被更新之后，写入在混乱更新区块中的扇区的记录。特定混乱更新区块的逻辑扇区地址（其可保存在混乱扇区列表中）的数目是典型值为8到16的设计参数。将列表的最佳尺寸确定为其对混乱数据写入操作的额外开销的作用与初始化期间的扇区扫描时间之间的权衡。

在系统初始化期间，为了识别自从关联的CBI扇区的一者的先前更新之后写入的有效扇区，必须扫描每一混乱更新区块。在控制器RAM中，构建每一混乱更新区块的混乱扇区列表。只需要在最后写入的CBI扇区中，从每一区块的混乱区块信息字段中界定的最后扇区地址开始扫描每一区块。

在配置混乱更新区块时，写入CBI扇区以对应于所有的已更新逻辑子群组。将混乱更新区块的逻辑和物理地址会写入在扇区中可用的混乱区块信息字段中，其中零条目（null entry）在混乱区块索引字段中。在控制器RAM中开启混乱扇区列表。

在关闭混乱更新区块时，以从扇区中混乱区块信息字段去除的区块的逻辑和物理地址写入CBI扇区。RAM中相应的混乱扇区列表变成未使用的。

修改控制器RAM中的相应的混乱扇区列表以包括写入到混乱更新区块的扇区的记录。当控制器RAM中的混乱扇区列表没有可用空间用于对混乱更新区块进行进一步扇区写入的记录时，为关于列表中扇区的逻辑子群组写入已更新的CBI扇区，且清除列表。

当CBI区块620变满时，将有效的CBI扇区复制到已配置的已擦除区块，且擦除先前的CBI区块。

地址表

图2所示的逻辑到物理地址转换模块140负责在快闪存储器中将主机的逻辑地址关联到相应的物理地址。逻辑群组与物理群组（元区块）之间的映射存储在分布在非易失性快闪存储器200和易失性但较敏捷的RAM 130（参看图1）中的一组表和列表中。地址表维持在存储器系统中含有每一逻辑群组的元区块地址的快闪存储器中。另外，最近写入的扇区的逻辑到物理地址记录暂时保存在RAM中。在系统上电之后进行初始化时，可从快闪存储器中的区块列表和数据扇区标题中重新构建这些易失性记录。因此，快闪存储器中的地址表只需要偶而更新，从而导致控制数据的额外开销写入操作的低百分比。

逻辑群组的地址记录的层级包括RAM中的开启的更新区块列表、关闭的更新区块列表和维持在快闪存储器中的群组地址表（GAT）。

开启的更新区块列表是控制器RAM中当前开启用于写入已更新主机扇区数据的数据更新区块的列表。一区块的条目在区块关闭时移动到关闭的更新区块列表。关闭的更新区块列表是控制器RAM中已关闭的数据更新区块的列表。列表中条目的子集在控制写入操作期间移动到群组地址表中的一扇区。

群组地址表（GAT）是存储器系统中主机数据的所有逻辑群组的元区块地址的列表。GAT含有根据逻辑地址循序排序的每一逻辑群组的一个条目。GAT中的第n个条目含有具有地址n的逻辑群组的元区块地址。在优选实施例中，其为快闪存储器中的表，包含具有界定存储器统中每一逻辑群组的元区块地址的条目的一组扇区（称为GAT扇区）。GAT扇区定位在快闪存储器中一个或一个以上专用的控制区块（称为GAT区块）中。

图17A说明群组地址表（GAT）扇区的数据字段。GAT扇区可（例如）具有足够的容量以含有一组128个连续逻辑群组的GAT条目。每一GAT扇区包括两个成分，即：用于范围内每一逻辑群组的元区块地址的一组GAT条目，和GAT扇区索引。第一成分含有用于定位与逻辑地址关联的元区块的信息。第二成分含有用于定位GAT区块内所有有效GAT扇区的信息。每一GAT条目具有三个字段，即：元区块号码、如先前结合图3A（iii）所定义的页标记、和表示元区块是否已重新链接的标志。GAT扇区索引列出GAT区块中有效GAT扇区的位置。此索引处于每一GAT扇区中但被GAT区块中下一写入的GAT扇区

的版本所取代。因此只有最后写入的GAT扇区中的版本为有效的。

图17B说明正记录在一个或一个以上GAT区块中的群组地址表（GAT）扇区的实例。GAT区块是专用于记录GAT扇区的元区块。在更新GAT扇区时，将其写入在GAT区块720中下一个可用的物理扇区位置。因此，GAT扇区的多个副本可存在于GAT区块中，其中只有最后写入的副本为有效的。例如，GAT扇区255（含有逻辑群组LG_{3968} - LG_{4098}的指针）至少已用作为有效版本的最新版本更新了两次。区块中最后写入的GAT扇区中的一组索引识别GAT区块中每一有效扇区的位置。在此实例中，区块中最后写入的GAT扇区是GAT扇区236，且其索引组是取代所有先前索引组的有效索引组。当GAT区块最后变成用GAT扇区完全填充时，通过将所有有效扇区重写到新的区块位置，在控制写入操作期间压缩区块。接着擦除整个区块。

如先前所描述，GAT区块在逻辑地址空间的区域中含有逻辑上连续组的群组的条目。GAT区块内的GAT扇区每一者含有128个连续逻辑群组的逻辑到物理映射信息。在GAT区块所跨越的地址范围内，存储所有逻辑群组的条目所需的GAT扇区数目仅占区块中总扇区位置的一部分。因此，通过将GAT扇区写入在区块中下一个可用扇区位置，可更新GAT扇区。GAT区块中所有有效GAT扇区及其位置的索引维持在最近写入的GAT扇区中的索引字段中。GAT区块中由有效GAT扇区占用的总扇区的一部分为系统设计参数，其通常为25%。然而，每一GAT区块中最多有64个有效GAT扇区。在大逻辑容量的系统中，可能必须在一个以上的GAT区块中存储GAT扇区。在此情况下，每一GAT区块与固定范围的逻辑群组关联。

可将GAT更新执行为控制写入操作的一部分，此操作在ABL用尽配置的区块时被触发（参看图18）。其与ABL填充和CBL清空操作同时执行。在GAT更新操作期间，一个GAT扇区具有用来自关闭的更新区块列表中的相应条目的信息更新的条目。当更新GAT条目时，从关闭的更新区块列表（CUBL）去除任何相应的条目。例如，基于关闭的更新区块列表中的第一条目来选择将要更新的GAT扇区。将已更新扇区写入到GAT区块中的下一可用扇区位置。

当没有扇区位置可供已更新的GAT扇区使用时，在控制写入操作期间进行GAT重写操作。配置新的GAT区块，且从整个GAT区块以循序次序复制GAT索引所界定的有效GAT扇区。接着擦除整个GAT区块。

GAT高速缓冲存储器是控制器RAM 130中，GAT扇区中的128个条目的再分部分（subdivision）中的条目的副本。GAT高速缓冲存储器条目的数目是系统设计参数，典型值为32。每次从GAT扇区读取条目时，建立相关扇区再分部分的GAT高速缓冲存储器。

维持多个GAT高速缓冲存储器。其数目是典型值为4的设计参数。GAT高速缓冲存储器根据最久未使用以不同扇区再分部分的条目进行重写。

已擦除的元区块管理

图2所示的擦除区块管理器160使用用于维持目录和系统控制信息的一组列表来管理擦除区块。这些列表分布在控制器RAM 130和快闪存储器200中。当必须配置已擦除的元区块以存储用户数据或存储系统控制数据结构时，选择保存在控制器RAM中的配置区块列表（ABL）中的下一可用元区块号码（参看图15）。类似地，在退出元区块之后将其擦除时，将其号码添加到也保存在控制器RAM中的已清除区块列表（CBL）。相对较静态的目录和系统控制数据存储在快闪存储器中。这些包括罗列快闪存储器中所有元区块的已擦除状态的已擦除区块列表和位图（MAP）。已擦除区块列表和MAP存储在个别扇区中并记录到称为MAP区块的专用元区块。这些分布在控制器RAM和快闪存储器中的列表提供已擦除区块记录的层级以有效地管理已擦除元区块的使用。

图18是说明使用和再循环已擦除区块的控制和目录信息的分布和流程的示意方框图。控制和目录数据维持在列表中，所述列表保存在控制器RAM 130中或保存在驻存在快闪存储器200中的MAP区块750中。

在优选实施例中，控制器RAM 130保存配置区块列表（ABL）610和已清除区块列表（CBL）740。如先前结合图15所描述，配置区块列表（ABL）追踪最近已配置哪些元区块以存储用户数据或存储系统控制数据结构。在需要配置新的已擦除元区块时，在配置区块列表（ABL）中选择下一可用的元区块号码。类似地，使用已清除区块列表（CBL）来追踪已解除配置并擦除的更新元区块。在控制器RAM 130（参看图1）中保存ABL和CBL以便在追踪相对活跃的更新区块时进行快速存取和简易操控。

配置区块列表（ABL）追踪将成为更新区块的已擦除元区块的集区和已擦除元区块的配置。因此，这些元区块中的每一者可由指定其是否为ABL待决配置中的已擦除区块、开启的更新区块、或关闭的更新区块的属性来描述。图18展示ABL含有已擦除的ABL列表612、开启的更新区块列表614、和关闭的更新区块列表616。另外，与开启的更新区块列表614关联的是关联的原始区块列表615。类似地，与关闭的更新区块列表关联的是关联的已擦除原始区块列表617。如先前图15所示，这些关联的列表分别是开启的更新区块列表614和关闭的更新区块列表616的子集。已擦除的ABL区块列表612、开启的更新区块列表614、和关闭的更新区块列表616全部为配置区块列表（ABL）610的子集，每一者中的条目分别具有相应的属性。

MAP区块750是专用于存储快闪存储器200中的擦除管理记录的元区块。MAP区块存

储MAP区块扇区的时间序列，其中每一MAP扇区为擦除区块管理（EBM）扇区760或MAP扇区780。因为已擦除区块配置用尽且在退出元区块时再循环，所以关联的控制和目录数据优选地包含在可在MAP区块中更新的逻辑扇区中，将更新数据的每一例项（instance）记录在新的区块扇区中。EBM扇区760和MAP扇区780的多个副本可存在于MAP区块750中，其中只有最新版本为有效的。有效的MAP扇区的位置的索引包含在EMB区块的字段中。有效的EMB扇区总是在控制写入操作期间最后被写入在MAP区块中。当MAP区块750已满时，通过将所有有效扇区重写到新的区块位置而在控制写入操作期间将其压缩。接着擦除整个区块。

每一EBM扇区760含有已擦除的区块列表（EBL）770，此为已擦除区块的总体的子集地址的列表。已擦除的区块列表（EBL）770充当含有已擦除的元区块号码的缓冲器，从所述缓冲器中周期性地取用元区块号码以重新填充ABL，并周期性地将元区块号码添加到所述缓冲器中以重新清空CBL。EBL 770为可用的区块缓冲器（ABB）772、已擦除的区块缓冲器（EBB）774和已清除的区块缓冲器（CBB）776充当缓冲器。

可用的区块缓冲器（ABB）772含有紧接着先前ABL填充操作的ABL 610中条目的副本。其实际上是恰好在ABL填充操作之后的ABL的备份副本。

已擦除的区块缓冲器（EBB）774含有先前已从MAP扇区780或从CBB列表776传送的已擦除的区块地址（描述如下），且所述地址可用于在ABL填充操作期间传送到ABL 610。

已清除的区块缓冲器（CBB）776含有在CBL清空操作期间已从CBL 740传送并随后将传送到MAP扇区780或传送到EBB列表774的已擦除区块的地址。

MAP扇区780的每一者含有称为MAP的位图结构。MAP为快闪存储器中的每一元区块使用一个位，所述位用来表示每一区块的擦除状态。对应于EBM扇区中的ABL、CBL或已擦除的区块列表中罗列的区块地址的位在MAP中不设定为已擦除状态。

区块配置算法永远不使用在MAP、已擦除的区块列表、ABL或CBL内的不含有有效数据结构且不指定为已擦除区块的任何区块，且因此所述区块不可被存取用来存储主机或控制数据结构。此提供从可存取的快闪存储器地址空间排除具有缺陷位置的区块的简单机制。

图18所示的层级允许有效地管理已擦除区块记录，并提供存储在控制器的RAM中述区块地址列表的完全安全性。以非频繁的方式在这些区块地址列表与一个或一个以上MAP扇区780之间交换已擦除的区块条目。在电源关闭之后的系统初始化期间，可通过存储在快闪存储器中的扇区中的已擦除区块列表和地址变换表中的信息和对快闪存储

器中少量被引用的数据区块进行的有限的扫描，来重建这些列表。

为了更新已擦除元区块记录的层级而采用的算法以一次序来配置使用已擦除区块：将来自MAP区块750的按照地址次序的区块突发（burst）与来自CBL 740的区块地址突发交叉，其反映主机更新区块的次序。对于大多数元区块尺寸和系统存储器容量而言，单一MAP扇区可为系统中的所有元区块提供位图。在此情况下，已擦除的区块总是以与记录在此MAP扇区中相同的地址次序而被配置使用。

擦除区块管理操作

如上述，ABL 610是具有可经配置使用的已擦除元区块和最近已配置为数据更新区块的元区块的地址条目的列表。ABL中的区块地址的实际数目在作为系统设计变数的最大与最小限制之间。在制造期间，格式化的ABL条目的数目是卡类型和容量的函数。另外，由于可用的已擦除区块的数目因寿命期间区块的故障而缩减，所以ABL中条目的数目可能缩减接近系统寿命的终点。例如，在填充操作之后，ABL中的条目可指定可用于以下目的的区块。每一区块具有一个条目的部分写入数据更新区块的条目不超过对于同时开启的更新区块的最大限度的系统限制。对于用于配置为数据更新区块的已擦除区块，条目在一个到二十个之间。对于用于配置为控制区块的已擦除区块，条目为四个。

ABL填充操作

由于ABL 610因配置而变得耗尽，所以需要对其进行重新填充。填充ABL的操作发生在控制写入操作期间。此在以下情况时被触发：必须配置区块，但ABL含有的已擦除区块条目不足以用于配置为数据更新区块或用于一些其它控制数据更新区块。在控制写入期间，ABL填充操作与GAT更新操作同时进行。

在ABL填充操作期间发生以下动作。

1. 保留具有当前数据更新区块的属性的ABL条目。

2. 保留具有关闭的数据更新区块的属性的ABL条目，除非区块的条目正在同时进行的GAT更新操作中被写入，此情况下从ABL中去除所述条目。

3. 保留用于未配置的擦除区块的ABL条目。

4. 压缩ABL以便去除因去除条目而产生的间隙，从而维持条目的次序。

5. 通过附加来自EBB列表的下一可用条目来完全填充ABL。

6. 用ABL中的当前条目重写ABB列表。

CBL清空操作

CBL是控制器RAM中已擦除区块地址的列表，对已擦除区块条目的数目的限制与ABL相同。清空CBL的操作发生在控制写入操作期间。因此，其与ABL填充/GAT更新操

作或CBI区块写入操作同时进行。在CBL清空操作中，从CBL 740去除条目并将其写入到CBB列表776。

MAP交换操作

当EBB列表774清空时，MAP扇区780中的擦除区块信息与EBM扇区760之间的MAP交换操作可周期性地发生在控制写入操作期间。如果系统中的所有已擦除元区块均记录在EBM扇区760中，那么不存在MAP扇区780且不执行MAP交换。在MAP交换操作期间，将用于向给EBB 774馈送已擦除区块的MAP扇区视为源MAP扇区782。相反地，将从CBB 776接收已擦除区块的MAP扇区视为目的地MAP扇区784。如果只存在一个MAP扇区，那么如下文所界定其可充当源MAP扇区和目的地MAP扇区两者。

在MAP交换期间执行以下动作。

1. 以递增指针为基础来选择源MAP扇区。

2. 以不在源MAP扇区中的第一CBB条目中的区块地址为基础来选择目的地MAP扇区。

3. 如CBB中相关条目所界定来更新目的地MAP扇区，且从CBB中去除所述条目。

4. 除非不存在分离的源MAP扇区，否则将已更新的目的地MAP扇区写入在MAP区块中。

5. 如CBB中相关条目所界定来更新源MAP扇区，且从CBB中去除所述条目。

6. 将CBB中剩余的条目附加到EBB。

7. 用从源MAP扇区界定的已擦除扇区地址尽可能地填充EBB。

8. 将已更新的源MAP扇区写入在MAP区块中。

9. 将已更新的EBM扇区写入在MAP区块中。

列表管理

图18展示各种列表之间的控制和目录信息的分布和流程。为了方便，在列表元素之间移动条目或改变条目属性的操作，在图18中识别为[A]到[O]，如下。

[A] 在将已擦除区块配置为主机数据的更新区块时，将其在ABL中的条目属性从已擦除的ABL区块改变为开启的更新区块。

[B]在将已擦除的区块配置为控制区块时，去除其在ABL中的条目。

[C]在建立具有开启的更新区块属性的ABL条目时，将关联的原始区块字段添加到条目以记录正被更新的逻辑群组的原始元区块地址。从GAT获得此信息。

[D] 当关闭更新区块时，其在ABL中的条目属性从开启的更新区块改变为关闭的更新区块。

[E]当关闭更新区块时，擦除其关联的原始区块，且将其在ABL中的条目的关联的原始区块字段的属性改变为已擦除的原始区块。

[F]在ABL填充操作期间，地址在相同控制写入操作期间在GAT中更新的任何关闭的更新区块将其条目从ABL中去除。

[G] 在ABL填充操作期间，在从ABL去除关闭的更新区块的条目时，将其关联的已擦除原始区块的条目移动到CBL。

[H] 在擦除控制区块时，将用于其的条目添加到CBL。

[I] 在ABL填充操作期间，从EBB列表将已擦除区块条目移动到ABL，且赋予已擦除区块条目已擦除ABL区块的属性。

[J] 在ABL填充操作期间修改所有相关的ABL条目之后，ABL中的区块地址取代ABB列表中的区块地址。

[K] 与控制写入期间的ABL填充操作同时进行，将CBL中已擦除区块的条目移动到CBB列表。

[L]在MAP交换操作期间，从CBB列表将所有相关条目移动到MAP目的地扇区。

[M] 在MAP交换操作期间，从CBB列表将所有相关条目移动到MAP源扇区。

[N] 在MAP交换操作期间的[L]和[M]之后，从CBB列表将所有其余条目移动到EBB列表。

[O] 在MAP交换操作期间的[N]之后，如果可能，那么从MAP源扇区移动除了在[M]中移动的条目以外的条目，以便填充EBB列表。

逻辑到物理地址转换

为了在快闪存储器中定位逻辑扇区的物理位置，图2所示的逻辑到物理地址转换模块140执行逻辑到物理地址转换。除了最近已更新的逻辑群组外，可使用驻存在控制器RAM 130中的快闪存储器200或GAT高速缓冲存储器中的群组地址表（GAT）来执行大多数的转换。最近已更新的逻辑群组的地址转换将需要查找主要驻存在控制器RAM 130中的更新区块的地址列表。因此，逻辑扇区地址的逻辑到物理地址转换的过程视与定位有扇区的逻辑群组关联的区块的类型而定。区块的类型为：完整区块、循序数据更新区块、混乱数据更新区块、关闭的数据更新区块。

图19是展示逻辑到物理地址转换过程的流程图。实质上，通过首先使用逻辑扇区地址查找各种更新目录（例如，开启的更新区块列表和关闭的更新区块列表），定位相应的元区块和物理扇区。如果关联的元区块不是更新过程的一部分，那么由GAT提供目录信息。逻辑到物理地址转换包括以下步骤：

步骤800：给定逻辑扇区地址。

步骤810：查找控制器RAM中开启的更新区块列表614中的给定逻辑地址（参看图15和图18）。如果查找失败，那么继续进行到步骤820，否则继续进行到步骤830。

步骤820：在关闭的更新区块列表616中查找给定的逻辑地址。如果查找失败，给定的逻辑地址不是任何更新过程的一部分；继续进行到步骤870，以进行GAT地址转换。否则继续进行到步骤860以进行关闭的更新区块地址转换。

步骤830：如果含有给定逻辑地址的更新区块为循序的，那么继续进行到步骤840以进行循序更新区块地址转换。否则继续进行到步骤850以进行混乱更新区块地址转换。

步骤840：使用循序更新区块地址转换来获得元区块地址。继续进行到步骤880。

步骤850：使用混乱更新区块地址转换来获得元区块地址。继续进行到步骤880。

步骤860：使用关闭的更新区块地址转换来获得元区块地址。继续进行到步骤880。

步骤870：使用群组地址表（GAT）转换来获得元区块地址。继续进行到步骤880。

步骤880：将元区块地址转换为物理地址。转换方法视元区块是否已重新链接而定。

步骤890：获得物理扇区地址。

更详细地描述各种地址转换处理，如下：

循序更新区块地址转换（步骤840）

可直接从开启的更新区块列表614中的信息完成与循序更新区块关联的逻辑群组中目标逻辑扇区地址的地址转换（图15和图18），如下。

1. 从列表中的“页标记”和“写入的扇区号码”字段确定目标逻辑扇区是否已定位在更新区块或其关联的原始区块中。

2. 从列表中读取适合目标逻辑扇区的元区块地址。

3. 从合适的“页标记”字段确定元区块内的扇区地址。

混乱更新区块地址转换（步骤850）

与混乱更新区块关联的逻辑群组中目标逻辑扇区地址的地址转换序列如下。

1. 如果从RAM中的混乱扇区列表确定扇区是最近写入的扇区，那么可直接从其在此列表中的位置来完成地址转换。

2. 在CBI区块中最近写入的扇区在其混乱区块数据字段内含有与目标逻辑扇区地址相关的混乱更新区块的物理地址。其在间接扇区索引字段内也含有关于此混乱更新区块的最后写入的CBI扇区的CBI区块内的偏移（参看图16A到图16E）。

3. 这些字段中的信息均被存储在RAM中，从而排除在后续的地址转换期间读取扇区的需要。

4. 读取步骤3处由间接扇区索引字段识别的CBI扇区。

5. 将最近存取的混乱更新子群的直接扇区索引字段存储在RAM中，而从而排除在步骤4处执行读取以重复存取相同的混乱更新区块的需要。

6. 在步骤4或步骤5处读取的直接扇区索引字段进而识别关于含有目标逻辑扇区地址的逻辑子群组的CBI扇区。

7. 从步骤6中识别的CBI扇区读取目标逻辑扇区地址的混乱区块索引条目。

8. 最近读取的混乱区块索引字段可存储在控制器RAM中，从而排除执行步骤4和步骤7处的读取以重复存取相同的逻辑子群组的需要。

9. 混乱区块索引条目界定目标逻辑扇区在混乱更新区块或关联的原始区块中的位置。如果目标逻辑扇区的有效副本处于原始区块中，那么通过使用原始元区块和页标记信息来对其进行定位。

关闭的更新区块地址转换（步骤860）

可直接从关闭的区块更新列表中的信息来完成与关闭的更新区块关联的逻辑群组中目标逻辑扇区地址的地址转换（参看图18），如下。

1. 从列表中读取配置到目标逻辑群组的元区块地址。

2. 从列表中的“页标记”字段确定元区块内的扇区地址。

GAT地址转换（步骤870）

如果逻辑群组不受到开启或关闭的区块更新列表的参考，那么其在GAT中的条目为有效的。由GAT参考的逻辑群组中的目标逻辑扇区地址的地址转换序列如下。

1. 评估RAM中可用的GAT高速缓冲存储器的范围，以便确定目标逻辑群组的条目是否包含在GAT高速缓冲存储器中。

2. 如果在步骤1中发现目标逻辑群组，那么GAT高速缓冲存储器含有全部群组地址信息，包括元区块地址和页标记两者，从而允许转换目标逻辑扇区地址。

3. 如果目标地址不在GAT高速缓冲存储器中，那么必须读取目标GAT区块的GAT索引以识别关于目标逻辑群组地址的GAT扇区的位置。

4. 最后存取的GAT区块的GAT索引保存在控制器RAM中，且不需要从快闪存储器读取扇区即可进行存取。

5. 将每一GAT区块的元区块地址和写入在每一GAT区块中的扇区数目的列表保存在控制器RAM中。如果步骤4处所需要的GAT索引不可用，那么可立即从快闪存储器中进行读取。

6. 从由步骤4或步骤6处获得的GAT索引界定的GAT区块中的扇区位置读取关于目

标逻辑群组地址的GAT扇区。用含有目标条目的扇区的再分部分来更新GAT高速缓冲存储器。

7. 从目标GAT条目内的元区块地址和“页标记”字段获得目标扇区地址。

元区块到物理地址转换（步骤880）

如果与元区块地址关联的标志表示元区块已被重新链接，那么从BLM区块读取相关的LT扇区以确定目标扇区地址的擦除区块地址。否则，直接从元区块地址确定擦除区块地址。

控制数据管理

图20说明在存储器管理的操作过程中，对控制数据结构执行的操作层级。数据更新管理操作对驻存在RAM中的各种列表发挥作用。控制写入操作对快闪存储器中各种控制数据扇区和专用区块发挥作用，且还与RAM中的列表交换数据。

在RAM中对ABL、CBL和混乱扇区列表执行数据更新管理操作。当将已擦除的区块配置为更新区块或控制区块时，或当关闭更新区块时，更新ABL。当擦除控制区块时，或当将关闭的更新区块的条目写入到GAT时，更新CBL。当将扇区写入到混乱更新区块时，更新更新混乱扇区列表。

控制写入操作促使将来自RAM中的控制数据结构的信息写入到快闪存储器中的控制数据结构，必要时随之更新快闪存储器和RAM中其它支持的控制数据结构。当ABL不含有将被配置为更新区块的已擦除区块的其它条目时，或当重写CBI区块时，触发控制写入操作。

在优选实施例中，在每一控制写入操作期间执行ABL填充操作、CBL清空操作和EBM扇区更新操作。当含有EBM扇区的MAP区块变满时，将有效的EBM和MAP扇区复制到已配置的已擦除区块，且接着擦除先前的MAP区块。

在每一控制写入操作期间，写入一个GAT扇区，且相应地修改关闭的更新区块列表。当GAT区块变满时，执行GAT重写操作。

如上所述，在某些混乱扇区写入操作之后，写入CBI扇区。当CBI区块变满时，将有效的CBI扇区复制到已配置的已擦除区块，且接着擦除先前的CBI区块。

如上所述，当EBM扇区中的EBB列表中不存在其它已擦除的区块条目时执行MAP交换操作。

每当重写MAP区块时，在专用的MAPA区块中写入记录MAP区块的当前地址的MAP地址（MAPA）扇区。当MAPA区块变满时，将有效的MAPA扇区复制到已配置的已擦除区块，且接着擦除先前的MAPA区块。

每当重写MAPA区块时，将启动扇区写入当前的启动区块中。当启动区块变满时，将有效的启动扇区从启动区块的当前版本复制到备份版本，接着所述备份版本变成当前版本。先前的当前版本被擦除并变成备份版本，且将有效的启动扇区写入回其中。

分布在多个存储器平面上的存储器的对准

如先前结合图4和图5A到图5C所描述，为了增加性能，并行地操作多个存储器平面。基本上，每一平面具有其自身的读出放大器组作为读取和程序电路的一部分，以便并行地服务跨越平面的存储器单元的相应页。在组合多个平面时，可并行地操作多个页，从而使性能更为提高。

根据本发明的另一方面，对于组织成可擦除区块并由多个存储器平面构成（因而可并行地读取逻辑单位或将逻辑单位并行地编程到多个平面中）的存储器阵列，当将要更新存储在给定存储器平面中的第一区块的原始逻辑单位时，提供所需以将已更新的逻辑单位保持在与原始相同的平面中。此通过将已更新的逻辑单位记录到仍在相同平面中的第二区块的下一可用位置而完成。优选地，将逻辑单位存储在平面中与其它版本相同的偏移位置，从而给定逻辑单位的所有版本由相同组的感测电路服务。

在优选实施例中，因此用逻辑单位的当前版本来填补上一已编程存储器单位与下一可用平面对准存储器单位之间的任何介入的间隙。通过用逻辑上跟随着上一已编程逻辑单位的逻辑单位的当前版本，且用逻辑上在存储在下一可用的平面对准存储器单位中的逻辑单位前面的逻辑单位的当前版本来填充间隙，而完成填补。

以此方式，将逻辑单位的所有版本维持在具有与原始相同偏移的相同平面中，从而在垃圾收集操作中，不需要从不同平面检索逻辑单位的最新版本而降低性能。在优选实施例中，用最新的版本来更新或填补每一平面上的每一存储器单位。因此，在多个平面上的并行操作中，逻辑单位将按照逻辑循序次序而无需进一步重新排列。

此方案通过允许平面上重新排列逻辑群组的逻辑单位的最新版本，且避免需要搜集不同存储器平面的最新版本，而缩短合并混乱区块的时间。此是有益的，因为主机接口的性能规格限定由存储器系统完成扇区写入操作的最大等待时间。

图21说明由多个存储器平面构成的存储器阵列。存储器平面可来自相同的存储器芯片或多个存储器芯片。每一平面910具有其自身的读取和程序电路912以并行地服务存储器单元的页914。在不损失一般性的情况下，在所示的实例中，存储器阵列具有四个并行操作的平面。

一般而言，逻辑单位是主机系统存取的最小单位。通常逻辑单位是尺寸为512字节的扇区。页是平面中并行读取或编程的最大单位。通常逻辑页含有一个或一个以上逻辑

单位。因此，在组合多个平面时，可将并行读取或编程的最大总数单位视为存储器单元的元页，其中元页由多个平面的每一者的页构成。例如，如MP_0的元页具有四页，来自平面P0、P1、P2和P3的每一者一页，其并行地存储逻辑页LP_0、LP_1、LP_2、LP_3。因此，与仅在一个平面中操作相比，存储器的读取和写入性能增加四倍。

将存储器阵列进一步组织成元区块，例如MB_0、...、MB_j，其中每一元区块内的所有存储器单元可作为单位而一起擦除。例如MB_0的元区块由多个存储器位置构成，用于存储数据的逻辑页914，例如LP_0-LP_{N-1}。元区块中的逻辑页根据其被填充在元区块中的次序，以预定的序列分布在四个平面P0、P1、P2和P3上。例如，在以逻辑上循序次序填步逻辑页时，以第一平面中的第一页、第二平面中的第二页等循环次序访问平面。在到达最后的平面之后，填充以循环的方式返回以便在下一元页中从第一平面再次开始。以此方式，可在所有平面均被并行操作时并行地存取连续系列的逻辑页。

一般而言，如果存在并行地操作的W个平面，且以逻辑上循序次序填充元区块，那么元区块中第k个逻辑页将驻存在平面x中，其中x = k MOD W。例如，在四个平面的情况下，W = 4，在以逻辑循序次序填充区块时，第5个逻辑页LP5将驻存在由5 MOD 4给定的平面中，即平面1，如图21中可以看到。

每一存储器平面中的存储器操作由一组读取/写入电路912来执行。进出读取/写入电路的每一者的数据在控制器920的控制下通过数据总线930进行传送。控制器920中的缓冲器922通过数据总线930来帮助缓冲数据的传送。明确地说，在第一平面中的操作需要存取第二平面中的数据时，需要两个步骤的过程。控制器首先从第二平面读出数据，接着通过数据总线和缓冲器将数据传送到第一平面。事实上，在大多数存储器结构中，在两个不同的位线之间传送数据也需要通过数据总线920交换数据。

至少，此涉及在一个平面中从一组读取/写入电路传送出去，并进入另一平面中的另一组读取/写入电路。在平面来自不同芯片的情况下，将需要在芯片之间进行传送。本发明提供存储器区块管理的结构和方案，从而避免一个平面从另一平面存取数据，以便使性能最大化。

如图21所示，元页由多个逻辑页（每一平面中一个逻辑页）形成。每一逻辑页可由一个或一个以上逻辑单位组成。当数据正以逐逻辑单位的方式记录到跨越平面的区块中时，每一逻辑单位将落在四个存储器平面的一者中。

在更新逻辑单位时发生平面对准的问题。在当前的实例中，为了说明的目的，将逻辑单位视为512字节的逻辑扇区，且逻辑页也为逻辑单位宽。由于快闪存储器不允许在不首先擦除整个区块的情况下重写区块的一部分，所以不将逻辑页的更新写入在现有的

位置上，而是将其记录在区块的未使用的位置中。接着将逻辑单位的先前版本（一个或多个）视为过时的。在许多更新之后，区块可能含有由于已更新而因此变得过时的许多逻辑单位。因此可称此区块“变脏”，且垃圾收集操作将忽略脏逻辑单位而收集每一个别逻辑单位的最新版本并以逻辑上循序次序将其重新记录在一个或一个以上新的区块中。接着擦除并再循环脏区块。

当将已更新的逻辑单位记录在区块中的下一未使用的位置中时，其通常不被记录在与其先前版本相同的存储器平面中。当要进行垃圾收集操作（例如合并或压缩）时，逻辑单位的最新版本将记录在与原始相同的平面中以维持原始次序。然而，如果必须从另一平面检索最新版本，那么性能将降低。

因此，根据本发明的另一方面，当将要更新存储在给定平面中的第一区块的原始逻辑单位时，提供所需以保持已更新的逻辑单位处于与原始相同的平面中。此通过将已更新的逻辑单位记录到仍在相同平面中的第二区块的下一可用位置而完成。在优选实施例中，用与原始区块中的原始逻辑单位相同的相对位置中的逻辑单位的当前版本来填补（即，通过复制来填充）上一已编程存储器单位与下一可用的平面对准存储器单位之间的介入的间隙。

图22A是说明根据本发明的一般实施方案，具有平面对准的更新的方法的流程图。

步骤950：在组织成区块的非易失性存储器中，将每一区块分割成可一起擦除的存储器单位，每一存储器单位用于存储逻辑单位的数据。

步骤952：用多个存储器平面构成存储器，每一平面具有用于并行地服务存储器页的一组感测电路，所述存储器页含有一个或一个以上存储器单位。

步骤954：根据第一次序将逻辑单位的第一版本存储在第一区块的存储器单位中，将每一第一版本逻辑单位存储在存储器平面的一者中。

步骤956：根据不同于第一次序的第二次序将逻辑单位的后续版本存储在第二区块中，将每一后续版本存储在与第一版本相同的平面中的下一可用存储器单位中，从而可通过相同组的感测电路从相同平面存取逻辑单位的所有版本。

图22B说明在图22A所示的流程图中存储更新的步骤的优选实施例。

步骤956'包括步骤957、步骤958和步骤959。

步骤957：将每一区块分割成元页，每一元页由每一平面的页构成。此步骤可在存储步骤的任一者之前执行。

步骤958：根据与第一次序不同的第二次序将逻辑单位的后续版本存储到第二区块，将每一后续版本存储在在元页中具有与第一版本相同偏移的下一可用存储器单位中。

步骤959：与存储逻辑单位的后续版本同时进行，通过根据第一次序复制逻辑单位的当前版本，以逐元页的方式填补在所述下一可用存储器单位之前的任何未使用的存储器单位。

图23A说明不论平面对准如何而以循序次序写入到循序更新区块的逻辑单位的实例。所述实例展示每一逻辑页为逻辑扇区的尺寸，例如LS0、LS1...。在四平面的实例中，可将每一区块（例如MB_0）视为分割成元页MP_0、MP_1、...，其中每一元页（例如MP_0）含有分别来自平面P0、P1、P2和P3的四个扇区（例如LS0、LS1、LS2和LS3）。因此，以循环次序以逐扇区的逻辑单位将区块填充在平面P0、P1、P2和P3中。

在主机写入操作#1中，正更新逻辑扇区LS5-LS8中的数据。将作为LS5'-LS8'的已更新数据记录在始于第一可用位置的新配置的更新区块中。

在主机写入操作#2中，正更新逻辑扇区LS9-LS12中数据的程序段。将作为LS9'-LS12'的已更新数据记录在紧接着最后写入结束处的位置中的更新区块中。可以看到，两次主机写入使得已以逻辑上循序次序将更新数据记录在更新区块中，即LS5'-LS12'。更新区块可视为循序更新区块，因为其已以逻辑上循序次序被填充。记录在更新区块中的更新数据使原始区块中的相应的数据过时。

然而，根据下一个可用位置但不论平面对准如何而将更新逻辑扇区记录在更新区块中。例如，扇区LS5原始地记录在平面P1中，但已更新的LS5'现记录在P0中。类似地，其它更新扇区全部未对准。

图23B说明不论平面对准如何而以非循序次序写入到混乱更新区块的逻辑单位的实例。

在主机写入操作#1中，更新存储在原始元区块中的给定逻辑群组的逻辑扇区LS10-LS11。将已更新的逻辑扇区LS10'-LS11'存储在新配置的更新区块中。此时，更新区块为循序的更新区块。在主机写入操作#2中，将逻辑扇区LS5-LS6更新为LS5'-LS6'并将其记录在紧接着上一写入的位置中的更新区块中。此将循序的更新区块转换为混乱的更新区块。在主机写入操作#3中，再次更新逻辑扇区LS10'并将其作为LS10"而记录在更新区块的下一位置中。此时，更新区块中的LS10"取代先前记录中的LS10'，而LS10'进而取代原始区块中的LS10。在主机写入操作#4中，再次更新逻辑扇区LS10"中的数据并将其作为LS10"'而记录在更新区块的下一位置中。因此，LS10"'现为逻辑扇区LS10的最后且唯一有效的版本。LS10的所有先前版本现均过时。在主机写入操作#5中，更新逻辑扇区LS30中的数据并将其作为LS30'而记录在更新区块中。在此实例中，可以任何次序并以任何重复将逻辑群组内的逻辑单位写入在混乱更新区块中。

同样，根据下一可用位置但不论平面对准如何而将更新逻辑扇区记录在更新区块中。例如，扇区LS10原始地记录在平面P2（即，MP_2、第三平面）中，但已更新的LS10'现记录在P0（即，MP_0'，第一平面）中。类似地，在主机写入#3中，再次将逻辑扇区LS10'更新为LS10"并放置在结果也在平面P0（MP_1'中的第一平面）中的下一可用位置中。因此，可以看到，一般而言，将更新扇区记录到区块的下一可用位置中可导致已更新扇区被存储在与其先前版本不同的平面中。

具有通过填补而填充的介入间隙的平面对准的循序更新区块

图24A说明根据本发明的优选实施例，具有平面对准和填补的图23A的循序更新实例。

在主机写入操作#1中，将更新为LS5'-LS8'的数据记录在始于第一可用平面对准位置的新配置的更新区块中。在此情况下，LS5原始地在P1中，P1是元页的第二平面。因此，将在更新区块的第一可用元页MP_0的相应平面中编程LS5'-LS7'。同时，用原始区块的元页中在LS5之前的逻辑扇区LS4的当前版本填补MP_0'中未使用的第一平面的间隙。接着将原始LS4视为过时的数据。接着将剩余的LS8'记录在下一元页MP_1'的第一个平面中并进行平面对准。

在主机写入操作#2中，将更新为LS9'-LS12'的数据记录在下一可用平面对准位置中的更新区块中。因此，将LS9'记录在下一可用平面对准存储器单位中，其为MP_1'的第二平面。在此情况下，不会产生间隙，且不必要进行填补。将更新区块视为循序更新区块，因为其已以逻辑上循序次序被填充。此外，其因每一更新逻辑单位与其原始的一样处于相同平面中而进行了平面对准。

具有介入间隙的平面对准的混乱更新区块

图24B说明根据本发明的一个优选实施例，具有平面对准和不进行填补的图23B的混乱更新实例。

在主机写入操作#1中，将已更新的逻辑扇区LS10'-LS11'存储在新配置的更新区块中。不是将其存储在下一可用的存储器单位中，而是将其存储在下一可用的平面对准存储器单位中。由于LS10'和LS11'原始地分别存储在平面P2和P3（原始区块的MP_2的第三和第四平面）中，所以下一可用的平面对准存储器单位将处于更新区块的MP_0'的第三和第四平面中。此时，更新区块为非循序的，以未填充、未填充、LS10'和LS11'的次序填充元页MP_0'的页。

在主机写入操作#2中，将逻辑扇区LS5-LS6更新为LS5'-LS6'并将其记录在下一可用的平面对准的位置中。因此，第二（P1）和第三（P2）平面或原始区块的MP_1的存储器

单位中的LS5'和LS6'将被编程到更新区块中的下一可用的元页MP_1'的相应平面中。此在MP_1'中留下位于前面的未使用的第一平面。

在主机写入操作#3中，再次更新逻辑扇区LS10'并将其作为LS10"而记录在更新区块的下一平面对准位置中。因此，其将被写入到下一可用的第三平面（将在MP_2'中）。此留下MP_1'的最后平面和MP_2'的前两个平面的位于前面的间隙。此将使MP_0'中的LS10'过时。

在主机写入操作#4中，再次更新逻辑扇区LS10"中的数据并将其作为LS10"'而记录在更新区块中的元页MP_2'的下一可用第三平面中。因此，LS10"'现为逻辑扇区LS10的最后且唯一有效的版本。此留下由MP_2'中的最后平面和MP_3'中的前两个平面组成的间隙。

在主机写入操作#5中，更新逻辑扇区LS30中的数据并将其作为LS30'而记录在更新区块中。由于原始的LS30驻存在元页的P2或第三平面中，所以其将被写入到更新区块中的下一可用的第三平面。在此情况下，其将为MP_4'的第三平面。从MP_3'的最后平面到MP_4'的前两个平面产生间隙。因此，此实例说明可以平面对准的方式，以任何次序并以任何重复，将逻辑群组内的逻辑扇区写入在混乱更新区块中。在后续的垃圾收集操作中，将便利地由相同组的感测电路来服务给定逻辑扇区的所有版本，尤其是最新版本。

具有通过填补而填充的介入间隙的平面对准的混乱更新区块

图24C说明根据本发明的另一优选实施例，具有平面对准和填补的图23B的混乱更新实例。

此操作与图24B所示的操作类似，但首先通过填补来填充介入间隙。在主机写入操作#1中，首先用驻存在原始区块中的LS8和LS9的当前版本来填补由元页MP_0'的第一和第二未使用的平面产生的间隙。此致使原始区块中的LS8和LS9过时。此时，更新区块是循序的更新区块，其中以LS8、LS9、LS10'和LS11'的次序填充元页MP_0'。

在主机写入操作#2中，MP_1'中位于前面的未使用的第一平面产生间隙，且首先用LS4来填补间隙。此将致使原始区块中的LS4过时。与之前一样，第二写入将循序更新区块转换为混乱更新区块。

在主机写入操作#3中，MP_1'中未使用的最后平面和MP_2'的前两个平面产生间隙。首先用跟随着上一已编程的LS6'的LS7来填补MP_1'的最后平面，且用在LS10之前的逻辑单位（即LS8和LS9）来填补MP_2'的前两个平面。此将使MP_0'中的LS10'和原始区块中的LS7-LS9过时。

在主机写入操作#4中，产生由MP_2'中的最后平面和MP_3'中的前两个平面组成的间隙。由作为元页MP_2'中跟随着最后写入的LS10"的逻辑单位的当前版本的LS11'填补MP_2'

的最后平面。与元页MP_3'中在LS10"'之前的逻辑单位一样，分别由LS8和LS9填补MP_3'的前两个平面。

在主机写入操作#5中，将相应地填补（即，分别用LS11'、LS28和LS29）从MP_3'的最后平面到MP_4'前两个平面的间隙。因此，此实例说明可以平面对准的方式，以任何次序并以任何重复，将逻辑群组内的逻辑扇区写入在混乱更新区块中。

在优选实施例中，元页含有来自个别平面的循环系列的页。由于可并行地读取或编程元页，所以便利地以元页的粒度实施每一主机更新。如果存在任何填补，那么以逐元页的方式与已更新的逻辑单位一起记录填补。

在图24A和图24C的实例所说明的实施例中，在每一主机写入期间，对在将要编程更新的平面对准存储器单位之前的未使用的存储器单位执行填补。在下一主机写入之前，延后跟随着上一已编程存储器单位的任何未使用的存储器单位的动作。一般而言，在每一元页的边界内填补任何位于前面的未使用的存储器单位。也就是说，如果位于前面的间隙跨越两个元页，那么以适合于每一元页的逻辑上循序次序对每一元页执行填补，但不论边界上的连续性如何。在合并区块的事件中，如果最后写入的元页是部分写入的话，那么将通过填补对其进行完全填充。

在另一实施例中，任何部分填充的元页在移动到下一元页之前被完全填补。

存储器单位粒度

视个别存储器结构所支持的适应性而定，读取或编程的单位可能存在变化。个别平面的独立特性允许独立地读取和编程元页中个别平面的每一页。上文给出的实例具有成为每一平面中的页的最大编程单位。在元页内，可能进行小于所有页的局部元页编程。例如，可能编程元页的前三页，且接着随后编程第四页。

并且，在平面层级，物理页可含有一个或一个以上存储器单位。如果每一存储器单位可存储数据扇区，那么物理页可存储一个或一个以上扇区。一些存储器结构支持局部页编程，其中可通过抑制页内选定的存储器单位的编程，在多次编程中，在不同的时间个别地编程选定的逻辑单位。

在存储器平面内用于逻辑群组的混乱更新的逻辑单位对准

在区块存储器管理系统中，以逻辑上循序次序将逻辑单位的逻辑群组存储在原始区块中。在更新逻辑群组时，将逻辑单位的后续版本存储在更新区块中。如果将逻辑单位混乱地（即，非循序地）存储在更新区块中，那么最终执行垃圾收集以收集原始区块和更新区块中逻辑单位的最新版本，并循序将其合并到新的原始区块中。如果将给定逻辑单位的已更新版本全部存储在与其原始区块中的原始版本对准的更新区块中，使得相同

组的感测电路可存取所有版本，那么垃圾收集操作将更为有效。

根据本发明的另一方面，在上述区块存储器管理系统中，在将存储器组织成一系列存储器页时（其中存储器单位的每一页由一组感测电路并行地服务），如果给定逻辑单位的所有版本在其被存储的页中全部具有相同的偏移位置，那么所有版本均对准。

图25说明每一页含有用于存储两个逻辑单位（例如，两个逻辑扇区）的两个存储器单位的示范性存储器组织。在原始区块中，由于以逻辑上循序次序存储逻辑扇区，所以将逻辑扇区LS0和LS1存储在页P0中，将逻辑扇区LS2和LS3存储在页P1中，并将逻辑扇区LS4和LS5存储在页P3中，等等。将看到，在此两个扇区的页中，从左边起第一扇区的页偏移为“0”，且第二扇区的页偏移为“1”。

在更新循序存储在原始区块中的逻辑扇区的逻辑群组时，将已更新的逻辑扇区记录在更新区块中。例如，逻辑扇区LS2驻存在原始区块中具有偏移“0”的页P0中。如果在第一写入中，如果将LS2更新为LS2'，那么其将存储在具有相同页偏移“0”的更新区块中的第一可用位置。此将处于页P0'的第一存储器单位中。如果在第二写入中，将LS5更新为LS5'，那么其将存储在具有相同页偏移“1”的更新区块中的第一可用位置。此将处于具有页P1'的偏移“1”的第二存储器单位中。然而，在存储LS5'之前，将通过在其中复制至少在每一页内将维持逻辑循序次序的逻辑扇区的最新版本，首先填补具有P0'中的偏移“1”和P1'中的偏移“0”的未使用的存储器单位。在此情况下，LS3将被复制到P0'中偏移“1”位置且LS4复制到P1'中偏移“0”位置。如果在第三写入中，再次将LS2'更新为LS2"，那么其将存储在P2'的偏移“0”中。如果在第四写入中，分别将LS22和LS23更新为LS22'和LS23'，那么其分别将存储在P3'的偏移“0”和“1”中。然而，在此之前，用LS3填补具有P2'中的偏移“1”的未使用的存储器单位。

上述更新序列假设可能在页内编程个别扇区。对于不支持局部页编程的一些存储器结构，必须将页内的所有扇区一起编程。在此情况下，在第一写入中，将LS2'和LS3一起编程到P0'中。在第二写入中，将LS4和LS5'一起编程到P1'中。在第三写入中，将LS2"和LS3一起编程到P2'中，等等。

元页内的平面对准

或者，编程的单位可具有元页的粒度。如果对混乱更新区块进行写入的粒度变成元页，那么结合图16A和图16B所述的CBI区块中的条目可与元页有关，而不是与扇区有关。增加的粒度减少必须为混乱更新区块记录的条目的数目，并允许直接消除索引且每一元区块使用单一CBI扇区。

图26A与图21的存储器结构类似，只是每一页含有两个扇区而不是一个。因此，可

以看到，元页MP_0的页的每一者现能够存储两个逻辑单位的数据。如果每一逻辑单位是扇区，那么将逻辑扇区循序存储在平面P0中LS0和LS1和平面P1中LS2和LS3等的MP_0中。

图26B说明图26A所示的具有以示意线性样式布局的存储器单位的元区块。与图21的单一扇区页相比，逻辑扇区以循环的方式存储在每一页中具有两个扇区的四个页中。

一般而言，如果存在W个并行操作的平面且每一页存在K个存储器单位，且以逻辑上循序次序填充元区块，那么元区块中第k个逻辑页将驻存在平面x中，其中x = k' MOD W，其中k' = INT（k/K）。例如，在四个平面的情况下，W = 4，且每一页2个扇区，K = 2，那么对于k = 5（表示第五个逻辑扇区LS5），其将驻存在由2 MOD 4给定的平面中（平面2），如图24A所示。一般而言，相同原理适用于实施上述的平面对准。

上文给出的实例用于多平面结构中页与平面的对准。在具有多个扇区的页的情况下，也维持页内的扇区对准将较为有利。以此方式，可便利地对于相同逻辑扇区的不同版本使用相同组的感测电路。有效地执行例如扇区的重新定位和“读取-修改-写入”的操作。在对准页内的扇区次序时，可使用与将页与平面对准相同的技术。并且视实施例而定，可能填补或可能不填补任何介入间隙。

不进行填补的逻辑单位或平面对准

图27说明在不填补将要从一个位置复制到另一位置的逻辑单位的情况下，在更新区块中进行平面对准的替代方案。可将与更新区块相交的四个平面的部分视为收集从主机接收的平面对准的已更新逻辑单位的四个缓冲器。在不在合适缓冲器的下一可用存储器单位中进行填补的情况下，编程从主机接收的每一逻辑单位。根据从主机接收的逻辑单位地址的序列，可能会有不同数目的逻辑单位被编程在每一平面中。

混乱更新区块MB'_1可含有逻辑元页的所有逻辑单位的已更新版本，例如对于MP'_0。其还可含有少于元页的所有逻辑单位，例如对于MP'_1。在MP'_1的情况下，可从相应的原始区块MB_0获得遗失的逻辑单位LS_4。

此替代方案在存储器结构支持并行读取每一平面的任意逻辑页时尤其有效。以此方式，即使个别逻辑页不是来自相同行，也可在单一并行读取操作中读取元页的所有逻辑页。

分阶段程序错误处理

当区块中存在程序失败时，那么通常将要存储到区块的所有数据移动到另一区块并将失败的区块标示为不良。视遇到失败的操作的定时规格而定，可能没有足够的时间来另外将存储的数据移动到另一区块。最坏的情况是在正常垃圾收集操作期间的程序失败，其中需要另一类似的垃圾收集操作将所有数据重新定位到另一区块。在此情况下，

可能违反给定的主机/存储器装置的规定的写入等待时间限制，因为所述限制通常经设计而容纳一次（而非两次）垃圾收集操作。

图28说明在合并操作期间缺陷区块中发生程序失败时在另一区块上重复合并操作的方案。在此实例中，区块1是以逻辑上循序次序存储逻辑群组的完整逻辑单位的原始区块。为了说明的目的，原始区块含有扇区A、B、C和D，每一者存储逻辑单位的子群组。当主机更新群组的特定逻辑单位时，将逻辑单位的较新版本记录在更新区块中，即区块2。如先前结合更新区块所描述，视主机而定，更新可以循序或非循序（混乱）次序来记录逻辑单位。最后，因为更新区块已满或一些其它原因而关闭更新区块以接收进一步的更新。当更新区块（区块2）关闭时，将驻存在更新区块或原始区块（区块1）上的逻辑单位的当前版本合并到新的区块（区块3）上，以便形成逻辑群组的新的原始区块。此实例展示更新区块在扇区B和D中含有逻辑单位的较新版本。为了方便，将扇区B和D示意地说明在区块2中未必是其被记录的位置，而是对准其在区块1中的原始位置。

在合并操作中，以循序次序将原始地驻存在区块1中的逻辑群组的所有逻辑单位的当前版本记录在合并区块（区块3）中。因此，首先从区块1将扇区A的逻辑单位复制到区块3，接着从区块2将扇区B复制到区块3。在此实例中，在从区块1将扇区C的逻辑单位复制到区块3时，区块3的缺陷导致程序失败。

一种处理程序失败的方式是在全新的区块（区块4）上重新启动合并程序。因此，将扇区A、B、C、D复制到区块4上，且接着丢弃缺陷区块3。然而，此将意味着前后执行两个合并操作，此可导致复制多达两个充满逻辑单位的区块。

存储器装置具有完成特定操作的特定时间容限。例如，在主机对存储器装置进行写入时，预计写入操作在指定的时间内完成，称为“写入等待时间”。当存储器装置（例如，存储卡）正忙于写入主机的数据时，向主机发送信号“占用”状态。如果“占用”状态持续的时间超过写入等待时间时段，那么主机将使写入操作暂停，并登记写入操作的例外或错误。

图29示意说明具有允许足够时间完成写入（更新）操作以及合并操作的定时或写入等待时间的主机写入操作。主机写入操作具有写入等待时间T_w，其将提供足够的时间以供完成将主机数据写入到更新区块的更新操作972（图29（A））。如先前在区块管理系统中所描述，对更新区块的主机写入可触发合并操作。因此，定时除了更新操作972之外也允许合并操作974（图29（B））。然而，必须响应于失败的合并操作而再一次重新启动合并操作，这可能花费太多时间并超过指定的写入等待时间。

根据本发明的另一方面，在具有区块管理系统的存储器中，通过继续中断区块中的

编程操作来处理时间紧急的存储器操作期间区块中的程序失败。稍后，在较不紧急的时间，将中断之前记录在失败区块中的数据传送到也可能是中断区块的另一区块。接着可丢弃失败的区块。以此方式，在遇到缺陷区块时，可在不损失数据且不会因必须立刻传送缺陷区块中的存储的数据而超过指定的时间限制的前提下进行处理。此错误处理对于垃圾收集操作尤其关键，从而在紧急时间期间不需要对崭新的区块重复整个操作。随后，在适宜的时间，可通过重新定位到另一区块来挽救来自缺陷区块的数据

图30说明根据本发明一般方案的程序失败处理的流程图。

步骤1002：将非易失性存储器组织成区块，将每一区块分割成可一起擦除的存储器单位，每一存储器单位用于存储逻辑单位的数据。

程序失败处理（第一阶段）

步骤1012：在第一区块中存储逻辑单位的数据的序列。

步骤1014：响应于存储许多逻辑单位之后第一区块处的存储失败，在充当第一区块的中断区块的第二区块中存储后续的逻辑单位。

程序失败处理（最后阶段）

步骤1020：响应于预定义的事件，将存储在第一区块中的逻辑单位传送到第三区块，其中第三区块与第二区块可能相同或可能不同。

步骤1022：丢弃第一区块。

图31A说明程序失败处理的一个实施例，其中第三（最后的重新定位）区块与第二（中断）区块不同。在阶段I期间，在第一区块上记录逻辑单位的序列。如果逻辑单位来自主机写入，那么可将第一区块视为更新区块。如果逻辑单位来自压缩操作的合并，那么可将第一区块视为重新定位区块。如果在某点处在区块1中遇到程序失败，那么提供充当中断区块的第二区块。将未能记录在区块1中的逻辑单位和任何后续逻辑单位改为记录在中断区块上。以此方式，不需要额外的时间来取代失败的区块1和驻存在其上的数据。

在中间阶段II中，可在区块1与区块2之间取得序列的所有已记录的逻辑单位。

在最后阶段III中，通过将逻辑单位重新定位到可充当重新定位区块的区块3，取代失败的区块1和驻存在其上的数据。因此，挽救了失败区块中的数据，接着可丢弃失败的区块。对最后阶段进行定时，使得其不与任何同时的存储器操作的定时冲突。

在此实施例中，重新定位区块3与中断区块2截然不同。此在中间阶段期间已以额外的逻辑单位记录中断区块时较为方便。因此，中断区块已变成更新区块，且可能不适于将缺陷区块1的逻辑单位重新定位到其中。

图31B说明程序失败处理的另一实施例，其中第三（最后的重新定位）区块与第二（中断）区块相同。阶段I和II与图31A所示的第一实施例类似。然而，在阶段III中，将缺陷区块1的逻辑单位重新定位到中断区块2。此在尚未以先前写入操作的原始序列以外的额外逻辑单位记录中断区块2时较为方便。以此方式，存储所讨论的逻辑单位需要最少的区块。

合并期间的程序失败处理的实施例

程序失败处理在合并操作期间尤其重要。正常的合并操作将驻存在原始区块和更新区块中的逻辑群组的所有逻辑单位的当前版本合并到合并区块中。在合并操作期间，如果合并区块中发生程序失败，那么将提供充当中断合并区块的另一区块来接收其余逻辑单位的合并。以此方式，不必复制逻辑单位一次以上，且仍可在为正常合并操作指定的时段内完成例外处理的操作。在适宜的时间，通过将群组的所有未完成的逻辑单位合并到中断区块中，可完成合并操作。适宜的时间将是在当前主机写入操作以外的有时间执行合并的某一其它时段期间。一个此适宜的时间是在存在更新但没有关联的合并操作的另一主机写入期间。

实质上，可将程序失败处理的合并视为以多个阶段实施。在第一阶段中，在发生程序失败之后，将逻辑单位合并到一个以上区块中以便避免将每一逻辑单位合并一次以上。在适宜的时间完成最后阶段，其中优选地通过以循序次序将所有逻辑单位收集到中断合并区块中，将逻辑群组合并到一个区块中。

图32A说明引起合并操作的初始更新操作的流程图。

步骤1102：将非易失性存储器组织成区块，将每一区块分割成可一起擦除的存储器单位，每一存储器单位用于存储逻辑单位的数据。

步骤1104：将数据组织成复数个逻辑群组，每一逻辑群组为可存储在区块中的逻辑单位的群组。

步骤1112：接收封装在逻辑单位中的主机数据。

步骤1114：通过根据第一次序在第一区块中存储逻辑群组的逻辑单位的第一版本，建立逻辑群组的原始区块。

步骤1116：通过根据第二次序在第二区块中存储包括逻辑群组的逻辑单位的后续版本，建立逻辑群组的更新区块。

步骤1119：在如前文段落所描述的某一预定义的事件时，执行垃圾收集以便在各种区块中收集逻辑单位的当前版本，并将其重新记录到新的区块中。

图32B说明根据本发明的优选实施例多阶段合并操作的流程图。

合并失败处理（阶段I）

错误处理的合并，阶段I操作1120包含步骤1122和步骤1124。

步骤1122：通过以与第一次序类似的次序在第三区块中存储逻辑群组的逻辑单位的当前版本，建立逻辑群组的合并区块。

步骤1124：响应于合并区块的存储失败，通过以与第一次序类似的次序在第四区块中存储第三区块所没有的逻辑群组的逻辑单位，提供中断合并区块。

由于已将区块1和区块2中的数据传送到区块3和区块4，所以可擦除区块1和区块2以释放空间。在优选实施例中，可立即将区块2释放到EBL（已擦除的区块列表，参看图18）并重新使用。如果区块1为关闭的更新区块且存在相应的GAT条目指向的另一区块，那么才可释放区块1。

实质上，区块3变成逻辑群组的原始区块，且区块4变成区块3的取代循序更新区块。

在完成阶段I合并之后，存储器装置通过释放占用信号而向主机发送信号。

中间操作（阶段II）

阶段II（中间操作1130）可在阶段III合并操作1140之前发生。如步骤1132、1134和1136中任一者所给出，可能存在许多可能的情况。

步骤1132：或者在逻辑群组的写入操作中，对作为更新区块的第四区块（中断合并区块）进行写入。

如果主机对所讨论的逻辑群组进行写入，那么区块4（其为中断合并区块且其至此已承担取代循序更新区块的角色）将用作正常更新区块。视主机写入而定，其可维持循序或变成混乱状态。作为更新区块，其将在某点触发关闭另一混乱区块，如先前优选实施例所描述。

如果主机对另一逻辑群组进行写入，那么直接进行到阶段III操作。

步骤1134：或者在读取操作中，读取其中第三区块作为逻辑群组的原始区块且第四区块作为更新区块的存储器。

在此情况下，将从作为逻辑群组的原始区块的区块3读取扇区A和B的逻辑单位，且将从作为群组的更新区块的区块4读取扇区C和D的逻辑单位。由于只可从区块3读取扇区A和B，所以将无法存取编程失败的页，且无法存取其后未写入的部分。虽然尚未更新快闪存储器中的GAT条目，且GAT条目仍指向作为原始区块的区块1，但将不从中读取数据，且此区块本身已在先前被擦除。

另一可能性是主机读取逻辑群组中逻辑单位。在此情况下，将从作为逻辑群组原始区块的区块3读取扇区A和B的逻辑单位，且将从作为群组的循序更新区块的区块4读取扇

区C和D的逻辑单位。

步骤1136：或者在上电初始化中，通过扫描第一到第四区块中的任一者的内容来重新识别第一到第四区块中的任一者。

中间阶段的另一可能性是关闭存储器装置的电源，并接着重新启动。如上文所描述，在上电初始化期间，扫描配置区块列表中的区块（将要使用的擦除集区的区块，参看图15和图18）以识别已成为逻辑群组的特殊状态原始区块（区块3）和关联的循序更新区块（区块4）的缺陷合并区块。中断区块（区块4）的第一逻辑单位中的标志将指示关联的区块为已遭遇程序错误的原始区块（区块3）。通过查阅区块目录（GAT），接着可定位区块3。

在一个实施例中，将标志编程到中断合并区块（区块4）中的第一逻辑单位中。此帮助指示逻辑群组的特殊状态：即，其已被合并到两个区块（即，区块3和区块4）中。

使用标志来识别具有缺陷区块的逻辑群组的替代方法是，利用不像原始区块应为已满的特性（除非错误发生在最后页中，且最后页没有ECC错误）检测在扫描期间为缺陷的区块。并且，视实施方案而定，可能存在关于存储在快闪存储器中的控制数据结构中的失败群组/区块的信息记录，而不只是写入到中断合并区块（区块4）的第一扇区的标题区域中的标志。

合并完成（阶段III）

步骤1142：响应于预定义的事件，且对于从阶段I之后未进一步记录第四区块的第一情况，以与第一次序类似的次序，在其中存储逻辑群组的所有未完成的逻辑单位的当前版本；且对于从阶段I之后已进一步记录第四区块的第二情况，将第三和第四区块合并到第五区块中。

步骤1144：此后，对于第一情况，操作存储器时，以已合并的第四区块作为逻辑群组的原始区块；且对于第二情况，操作存储器时，以第五区块作为逻辑群组的原始区块。

只要存在不会违反任何指定时间限制的机会，可执行阶段III中的最后合并。优选的情况是，在存在对另一逻辑群组的不附带有合并操作的更新操作时，“捎带（piggy-back）”在下一主机写入时隙上。如果对于另一逻辑群组的主机写入触发本身的垃圾收集，那么将使阶段III合并延后。

图33说明多阶段合并操作的第一和最后阶段的示范性定时。主机写入等待时间是具有时段T_w的每一主机写入时隙的宽度。主机写入1是简单的更新，且逻辑群组LG_1中第一组逻辑单位的当前版本记录在关联的更新区块上。

在主机写入2，在逻辑群组LG_1上发生更新，致使更新区块关闭（如，已满）。将提

供新的更新区块以便记录其余的更新。提供新的更新区块可触发垃圾收集，垃圾收集引起对于LG_4的合并操作，以便再循环区块以供重新使用。以循序次序将LG_4群组的当前逻辑单位记录在合并区块上。合并操作继续进行直到在合并区块中遭遇缺陷为止。然后调用阶段I合并，其中合并操作在中断合并区块上继续。同时，LG_4的最后合并（阶段III）等待下一机会。

在主机写入3，也发生逻辑群组LG_2的逻辑单位的写入以触发LG_2的合并。此意味着已经完全利用时隙。

在主机写入4，操作只是将LG_2的一些逻辑单位记录到其更新区块。时隙中剩余的时间提供执行LG_4的最后合并的机会。

不将中断合并区块转换为更新区块的实施例

图34A和图34B分别说明图28和图31的实例适用的多阶段合并的阶段I和阶段III操作的第一情况。

图34A说明中断合并区块不用作更新区块而是用作合并操作已中断的合并区块的情况。明确地说，图34A表示图33所示的主机写入#2，其中主机写入属于逻辑群组LG_1的逻辑单位的更新，且在此期间，此操作也触发与另一逻辑群组LG_4关联的区块的合并。

原始区块（区块1）和更新区块（区块2）的形成方式与图28的实例相同。类似地，在合并操作期间，已知合并区块（区块3）在合并扇区C的逻辑单位时具有缺陷。然而，不像图28所示的重新合并方案，本多阶段方案在充当中断合并区块的新提供的区块（区块4）上继续合并操作。因此，在阶段I合并操作中，已在合并区块（区块3）中合并扇区A和B中的逻辑单位。当合并区块中发生程序失败时，循序将扇区C和D中其余的逻辑单位复制到中断合并区块（区块4）。

如果主机原始地在第一逻辑群组中写入更新而触发与第二逻辑群组关联的区块的合并操作，那么将第一逻辑群组的更新记录到第一逻辑群组的更新区块（通常为新的更新区块）中。此时，中断合并区块（区块4）不用来记录合并操作以外的任何更新数据并仍然是尚待完成的中断合并区块。

由于区块1和区块2中的数据现完全包含在另一区块（区块3和区块4）中，所以可将其擦除以便再循环。地址表（GAT）经更新以指向作为逻辑群组的原始区块的区块3。更新区块的目录信息（在ACL中，参看图15和图18）也经更新以指向已成为逻辑群组（例如，LG_4）的循序更新区块的区块4。

结果，已合并的逻辑群组不局限在一个区块中，而是分布在缺陷合并区块（区块3）和中断合并区块（区块4）上。此方案的重要特征是，群组中的逻辑单位只在此阶段期

间合并一次，以将合并散布在一个以上区块为代价。以此方式，可在正常指定的时间内完成合并操作。

图34B说明始于图34A的多阶段合并的第三和最后阶段。如结合图33所描述，在第一阶段之后的适宜时间（例如，在不触发随附合并操作的后续主机写入期间）执行阶段III合并。明确地说，图34B表示发生图33所示的主机写入#4的时隙。在此时段期间，主机写入更新属于逻辑群组LG_2的逻辑单位而不会触发另一额外的合并操作。因此，可便利地利用时隙中剩余的时间来进行阶段III操作，以便完成逻辑群组LG_4的合并。

此操作将尚不处于中断区块中的LG_4的所有未完成的逻辑单位合并到中断区块中。在此实例中，这意味着将从区块3以逻辑上循序次序将扇区A和B复制到中断区块（区块4）。由于区块中逻辑单位的绕回方案和使用页标记（参看图3A），即使实例展示在区块4中，在扇区C和D之后记录扇区A和B，但也将已记录的序列视为等同于A、B、C、D的循序次序。视实施方案而定，优选地从区块3获得将要复制的未完成的逻辑单位的当前版本，因为其已为已合并的形式，但也可从尚未被擦除的区块1和区块2中收集。

在中断区块（区块4）上完成最后合并之后，其将被指定为逻辑群组的原始区块，且将相应地更新合适的目录（例如，GAT，参看图17A）。类似地，将失败的物理区块（区块3）标示为不良并将其排除。其它区块（区块1和区块2）将被擦除并再循环。同时，将LG_2的更新记录在与LG_2关联的更新区块中。

将中断合并区块变成更新区块的实施例

图35A和图35B分别说明图28和图33的实例适用的多阶段合并的阶段I和阶段III操作的第二情况。

图35A说明将中断合并区块维持为接收主机写入的更新区块而不是合并区块的情况。此适用于（例如）更新逻辑群组LG_4的主机写入，且在此过程中，也触发相同逻辑群组中的合并。

与图34A的情况一样，继续将区块1和区块2合并到区块3上，直到在处理扇区C时遇到程序失败为止。接着在中断合并区块（区块4）上继续合并。在中断区块（区块4）中已合并未完成的逻辑单位（例如，在扇区C和D中）之后，并在阶段III中等待完成其中的逻辑群组合并，而是将中断区块维持为更新区块。此情况尤其适于主机写入更新逻辑群组并触发相同逻辑群组的合并的情况。在此实例中，此使得能够将逻辑群组LG_4的主机更新的记录记录在中断合并区块（区块4）中，而不是记录到新的更新区块。更新区块（先前为中断合并区块（区块4））可视其中所记录的主机数据而定为循序的或变得混乱。在所示的实例中，区块4已变得混乱，因为扇区C中逻辑单位的后续较新版本使得区

块4中的先前版本过时。

在中间阶段期间，区块3将被视为LG_4的原始区块，且区块4将为关联的更新区块。

图35B说明始于第二情况的图35A的多阶段合并的第三和最后阶段。如结合图33所描述，在第一阶段之后的适宜时间（例如，在不触发随附合并操作的后续主机写入期间）执行阶段III合并。在此时段期间，主机写入更新属于逻辑群组的逻辑单位而不会触发合并操作。因此，可便利地利用时隙中剩余的时间来进行阶段III操作，以便完成逻辑群组LG_4的合并。

接着从区块3和区块4将逻辑群组LG_4的垃圾收集到新的合并区块（区块5）。接着将区块3标示为不良，将区块4再循环，且新的合并区块（区块5）将变成逻辑群组LG_4的新的原始区块。其它区块（区块1和区块2）也会被擦除并再循环。

分阶段程序失败处理的其它实施例

图31A、图31B、图34A、图34B、图35A和图35B中所描述的实例适用于优选的区块管理系统，其中每一物理区块（元区块）仅存储属于相同逻辑群组的逻辑单位。本发明同样适用于不存在逻辑群组到物理区块对准的其它区块管理系统，例如WO 03/027828和WO 00/49488中所揭示的区块管理系统。图36A、图36B和图36C中说明在这些其它系统中实施分阶段程序失败处理方法的一些实例。

图36A说明如适用于主机写入触发更新区块的关闭且更新区块为循序的情况的分阶段程序错误处理方法。此情况中的关闭通过将原始区块2的其余的有效数据（B和C）复制到循序更新区块3而完成。在数据部分C编程开始时发生程序失败的情况下，将把部分C编程到保留的区块4。接着可将新的主机数据写入到新的更新区块5（未图示）。此方法的阶段II和III与混乱区块关闭的情况相同。

图36B说明如可适合用于更新区块的更新的情况中（局部区块系统）的分阶段程序错误处理方法。在此情况下，将逻辑群组存储在原始区块1和其它更新区块中。合并操作包括从原始区块1和其它更新区块2将数据复制到更新区块的一者（根据一些规则选定，图中的区块3）。与已描述的主要情况的区别在于，区块3已被部分地写入。

图36C说明处理垃圾收集操作的分阶段程序错误，或不支持映射到元区块的逻辑群组的存储器区块管理系统中的清除。WO 03/027828 A1中描述了此存储器区块管理（循环存储）系统。循环存储系统的明显的特征是不为单一逻辑群组配置区块。支持元区块中控制数据的多个逻辑编组。垃圾收集涉及从部分过时的区块将可能没有任何关系（随机逻辑区块地址）的有效数据扇区取到可能已具有某些数据的重新定位区块。如果重新定位区块在操作期间变满，那么将开启另一区块。

非循序更新区块索引

在上文关于混乱区块索引并结合图16A到图16E的段落中，CBI扇区可用来存储追踪随机地存储在混乱或非循序更新区块中的逻辑扇区的位置的索引。

根据本发明的另一方面，在具有支持具有非循序逻辑单位的更新区块的区块管理系统的非易失性存储器中，将缓冲在RAM中的更新区块中的逻辑单位的索引周期性地存储在非易失性存储器中。在一个实施例中，将索引存储在专用于存储索引的区块中。在另一实施例中，将索引存储在更新区块中。在又一实施例中，将索引存储在每一逻辑单位的标题中。在另一方面，在上一索引更新之后但在下一索引更新之前写入的逻辑单位将其索引信息存储在每一逻辑单位的标题中。以此方式，在电源中断之后，不必在初始化期间执行扫描，即可确定最近写入的逻辑单位的位置。在又一方面，将区块管理成部分循序且部分非循序，指向一个以上逻辑子群组。

在预定触发事件之后存储在CBI区块中的CBI扇区中的索引指针

根据结合图16A到图16E所描述的方案，将混乱区块中最近写入的扇区的列表保存在控制器RAM中。只在与给定的混乱区块关联的逻辑群组中的预定数目的写入之后，才将含有最新索引信息的CBI扇区写入到快闪存储器（CBI区块620）。以此方式，缩减对CBI区块进行的更新的数目。

在逻辑群组的CBI扇区的下一更新之前，将逻辑群组的最近写入的扇区的列表保存在控制器RAM中。此列表在存储器装置遭遇电源关闭时遗失，但可在电源启动之后的初始化中通过扫描已更新区块而得以重建。

图37说明在相同逻辑群组的每N个扇区写入之后将CBI扇区写入到关联的混乱索引扇区区块的程序安排的实例。此实例展示进行同时更新的两个逻辑群组LG_3和LG_{11}。最初，将LG_3的逻辑扇区以循序次序存储在原始区块中。以主机指定的次序将群组中逻辑扇区的更新记录在关联的更新区块上。此实例展示混乱更新序列。同时，也以与其更新区块类似的方式更新逻辑群组LG_{11}。在每一逻辑扇区写入之后，将其在更新区块中的位置保存在控制器RAM中。在每一预定的触发事件之后，以混乱索引扇区的形式将更新区块中逻辑扇区的当前索引写入到非易失性混乱索引扇区区块。例如，在每N个写入之后发生预定的触发事件，其中N可能为3。

虽然给出的实例关于作为扇区的数据逻辑单位，但所属领域的技术人员将了解，逻辑单位可为一些其它集合体，例如含有扇区或扇区群组的页。并且，循序区块中的第一页不一定是逻辑页0，因为绕回的页标记可能在适当位置。

在预定触发事件之后存储在混乱更新区块中的CBI扇区中的索引指针

在另一实施例中，在其中每N个写入之后，将索引指针存储在混乱更新区块本身中的专用CBI扇区中。此方案与先前描述的也将索引存储在CBI扇区中的实施例类似。区别在于，在先前的实施例中，将CBI扇区记录在CBI扇区区块中，而不是更新区块本身中。

此方法基于将所有混乱区块索引信息保存在混乱更新区块本身中。图37A、图37B和图37C分别说明同样以三个不同阶段存储CBI扇区的更新区块的状态。

图38A说明直到在预定数目的写入之后在其中记录CBI扇区时的更新区块。在此实例中，在主机已循序写入逻辑扇区0-3之后，其接着发布再次写入逻辑扇区1的另一版本的命令，因此破坏数据写入的连续序列。接着实施CBI扇区中载送的混乱区块索引，将更新区块转换为混乱更新区块。如上文所描述，CBI是含有混乱区块的所有逻辑扇区的索引的索引。例如，第0个条目表示第0个逻辑扇区的更新区块的偏移，且类似地，第n个条目表示第n个逻辑扇区的偏移。将CBI扇区写入到更新区块中的下一可用位置。为了避免频繁的快闪存取，在每N个数据扇区写入之后写入CBI扇区。在此实例中，N为4。如果此时损失电源，那么最后写入的扇区将为CBI扇区，且此区块将被视为混乱更新区块。

图38B说明图38A的更新区块在索引扇区之后进一步在其中记录逻辑扇区1、2和4。逻辑扇区1和2的较新版本取代先前记录在更新区块中的较旧版本。此时在电源周期的情况下，必须首先找到最后写入的扇区，且接着必须扫描多达N个扇区以便找到最后写入的索引扇区和最近写入的数据扇区。

图38C说明图38B的更新区块写入另一逻辑扇区以触发索引扇区的下一记录。另外N个（N=4）扇区写入之后的相同更新区块记录CBI扇区的另一当前版本。

此方案的优点是不需要分离的CBI区块。同时，不必担心物理快闪扇区的额外开销数据区域是否足够大而容纳混乱更新区块中有效扇区的索引所需要的数目的条目。接着，混乱更新区块含有所有的信息，且地址转换不需要外部数据。此使算法比较简单，缩减关于CBI区块压缩的控制更新的数目且级联（cascade）控制更新较短。（参看上文关于CBI区块管理的段落）。

关于存储在混乱更新区块中的数据扇区标题中的最近写入的扇区的信息

根据本发明的另一方面，在每N个写入之后，将记录在区块中的逻辑单位的索引存储在非易失性存储器中，且将关于中间写入的逻辑单位的当前信息存储在写入的每一逻辑单位的额外部分中。以此方式，在电源重新启动之后，不必扫描区块，即可从区块中最后写入的逻辑单位的额外部分快速获得关于从上一索引更新之后写入的逻辑单位的信息。

图39A说明存储在混乱更新区块中每一数据扇区的标题中的中间写入的中间索引。

图39B说明在写入的每一扇区的标题中存储中间写入的中间索引的实例。在此实例中，在已写入四个扇区LS_0-LS_3之后，写入CBI索引作为区块中的下一扇区。其后，将逻辑扇区LS'_1、LS'_2和LS_4写入到区块。每次，标题将存储从上一CBI索引之后写入的逻辑单位的中间索引。因此，LS'_2中的标题将具有给出上一CBI索引的偏移（即，位置）以及LS'_1的偏移的索引。类似地，LS_4中的标题将具有给出上一CBI索引的偏移以及LS'_1和LS'_2的偏移的索引。

最后写入的数据扇区始终含有关于多达N个最后写入的页的信息（即，一直到最后写入的CBI扇区）。只要电源重新启动，上一CBI索引提供在CBI索引扇区之前写入的逻辑单位的索引信息，且在最后写入的数据扇区的标题中找到写入的后续逻辑单位的索引信息。此优点在于：在初始化时不必为其后写入的扇区扫描区块以确定其位置。

在数据扇区的标题中存储中间索引信息的方案同样适用于无论CBI索引扇区存储在更新区块本身中还是存储在分离的CBI扇区区块中，如前文段落中所描述。

存储在混乱更新区块中的数据扇区标题中的索引指针

在另一实施例中，将整个CBI索引存储在混乱更新区块中每一数据扇区的额外部分中。

图40说明存储在混乱更新区块中的每一数据扇区标题中的混乱索引字段中的信息。

扇区标题的信息容量有限，且因此可将由任何单一扇区提供的索引范围设计为层级索引方案的一部分。例如，存储器的特定平面内的扇区可仅向平面内的扇区提供索引。并且，可将逻辑地址的范围划分成许多子范围以允许采用间接索引方案。例如，如果将具有64个逻辑地址的扇区存储在平面中，那么每一扇区可具有用于扇区偏移值的3个字段，每一字段能够存储4个偏移值。第一字段定义逻辑偏移范围0-15、15-31、32-47和48-63内最后写入的扇区的物理偏移。第二字段定义在其相关范围内每一4个扇区的4个子范围的物理偏移值。第三字段定义在其相关子范围内4个扇区的物理偏移值。因此，通过读取多达3个扇区的间接偏移值，可确定混乱更新区块内逻辑扇区的物理偏移。

此方案的优点是也不需要分离的CBI区块或CBI扇区。然而，此可能只有当物理快闪扇区的额外开销数据区域足够大而容纳混乱更新区块中有效扇区的索引所需要的数目的条目时才适用。

混乱更新区块的逻辑群组内有限的逻辑范围

在逻辑群组内，缩减可非循序地写入的扇区的逻辑范围。此技术的主要优点为：由于只需要读取一个多扇区页（在多芯片的情况下，可并行读取页）来获得目的地页的所有数据（假设源与目的地对准，如果未对准，那么可能需要另一读取），使得可更快速

地完成循序写入的数据的复制，所以范围以外的扇区在原始区块中保持循序写入，且垃圾收集操作可在更短的时间内完成。并且，可使用芯片上的副本特征，将循序数据从源复制到目的地，而不需要在控制器之间来回传送数据。如果源数据已分散（如混乱区块中所发生），那么可能需要每一扇区读取多达一个页以便收集将要写入到目的地的所有扇区。

在一个实施例中，不是完全地将逻辑范围限制为某数目的扇区，而是通过限制CBI的数目来完成（只限制大群组/元区块的混乱范围较合理，此需要多个混乱区块索引来覆盖整个逻辑群组的范围）。例如，如果元区块/群组具有2048个扇区，那么其将需要多达8个CBI扇区，每一CBI扇区覆盖256个扇区的一个子群组的连续逻辑范围。如果将CBI的数目限制为4，那么混乱区块可用来写入多达4个子群组的扇区（其中任何一个）。因此，允许逻辑群组具有多达4个部分或完全混乱的子群组，且最少有4个子群组将保持完全循序。如果混乱区块具有与其关联的4个有效CBI扇区，且主机写入在这些CBI扇区范围以外（混乱子群组）的扇区，那么应合并并关闭混乱逻辑群组。但这极不可能发生，因为在实际应用中，主机在2048个扇区的范围（逻辑群组）内不需要多于4个的256个扇区的混乱范围（子群组）。因此，在正常情况下，垃圾收集时间也不会受到影响，但限制规则的防范形成垃圾收集太长（可能触发主机的逾时）的极端情况。

部分循序混乱更新区块的索引

当循序更新区块在将区块转换为混乱管理模式之前已被部分地写入时，逻辑群组的已循序更新的扇区的全部或部分可继续被视为已循序更新，且可将混乱更新管理仅应用于逻辑群组的地址范围的子集。

控制数据完整性&管理

存储在存储器装置中的数据可能因为电源中断或特定存储器位置变得有缺陷被破坏。如果遭遇存储器区块缺陷，那么将数据重新定位到不同的区块并将缺陷区块丢弃。如果错误不扩大，那么可通过与数据一起保存的错误校正码（ECC）在运行中进行校正。然而，存在ECC无法校正已破坏数据的情况。例如，当错误位的数目超过ECC的容量时。此对于例如与存储器区块管理系统关联的控制数据的关键数据是无法接受的。

控制数据的实例为目录信息和与存储器区块管理系统关联的区块配置信息，例如结合图20所描述。如上文所描述，将控制数据维持在高速RAM和较慢的非易失性存储器区块中。将任何经常变更的控制数据维持在具有周期性控制写入的RAM中以便更新存储在非易失性元区块中的同等的信息。以此方式，将控制数据存储在非易失性但较慢的快闪存储器中而不需要经常存取。例如图20所示的GAT、CBI、MAP和MAPA的控制数据结

构的层级维持在快闪存储器中。因此，控制写入操作促使RAM中控制数据结构的信息更新快闪存储器中同等的控制数据结构。

关键数据复制

根据本发明的另一方面，例如控制数据的一些或全部的关键数据如果维持在复制项中，那么保证额外等级的可靠性。以一方式执行复制，使得对于使用两次编程技术来连续地编程相同组的存储器单元的多位的多状态存储器系统而言，第二次中的任何编程错误将不破坏第一次建立的数据。复制也有助于检测写入中止、检测误测（即，两个副本均具有良好的ECC但数据不同），并增加额外等级的可靠性。已考虑数据复制的若干技术。

在一个实施例中，在稍早的一次编程中已编程给定数据的两个副本之后，后续的一次编程避免编程存储两个副本中的至少一个的存储器单元。以此方式，在后续的一次编程在完成之前中止并破坏稍早的一次的数据的事件中，两个副本中至少一个不会受到影响。

在另一实施例中，给定数据的两个副本存储在两个不同的区块中，其中两个副本中至多一个的存储器单元在后续的一次编程中被编程。

在又一实施例中，在一次编程中已存储给定数据的两个副本之后，将不再对存储两个副本的存储器单元组执行进一步编程。可通过存储器单元组的最终一次编程中编程两个副本来达成此目的。

在又一实施例中，可在二进制编程模式中将给定数据的两个副本编程到多状态存储器中，从而将不会对已编程的存储器单元进行进一步编程。

在又一实施例中，对于使用两次编程技术来连续地编程相同组的存储器单元的多位的多状态存储器系统而言，使用容错码来编码多个存储器状态，从而稍早的一次编程所建立的数据不会受到后续的一次编程中的错误的影响。

在每一存储器单元存储一位以上数据的多状态存储器中引发数据复制的复杂性。例如，4状态存储器可由两个位表示。一种现有的技术是使用2次编程来编程此存储器。第一位（下页位）由第一次编程进行编程。随后，在第二次编程中编程相同单元以表示所要的第二位（上页位）。为了不改变第二次编程中第一位的值，使第一位的存储器状态表示依赖于第二位的值。因此，在第二位的编程期间，如果因电源中断或其它原因而发生错误并造成不正确的存储器状态，那么也会破坏第一位的值。

图41A说明当每一存储器单元存储两个位的数据时，4状态存储器阵列的阈值电压分布。四个分布表示四个存储器状态“U”、“X”、“Y”和“Z”的总体。在编程存储器单

元之前，首先将其擦除到其“U“或“未写入”状态。随着存储器单元逐渐被编程，累进地达到存储器状态“X”、“Y”和“Z”。

图41B说明使用葛莱码的现有的2次编程方案。四个状态可由两个位表示，下页位和上页位，例如（上页位,下页位）。对于将要并行编程的单元的页，实际上存在两个逻辑页：逻辑下页和逻辑上页。第一次编程只编程逻辑下页。通过合适的编码，对于单元的相同页的后续第二次编程将编程逻辑上页而不用重设逻辑下页，。一般使用的代码是葛莱码，其中只有一个位在转换到邻近的状态时发生改变。因此，此代码的优点为：因为只涉及一个位，所以对于错误校正的要求较少。

使用葛莱码的一般方案是假设“1”表示“不编程”条件。因此，已擦除的存储器状态“U”可表示为：（上页位，下页位）=（1,1）。在编程逻辑下页的第一次编程中，存储数据“0”的任何单元的逻辑状态将因此从（x,1）转换为（x,0），其中“x”表示上位的“任意（don't care)”值。然而，由于上位尚未被编程，所以为了一致，也可将“x”标示为“1”。通过将单元编程为存储器状态“X”来表示（1,0）逻辑状态。也就是说，在第二次编程之前，下位值“0”可由存储器状态“X”表示。

执行第二次编程来存储逻辑上页的位。只有需要值为“0”的上页位的那些单元将被编程。在第一次编程之后，页中的单元处于逻辑状态（1,1）或（1,0)。为了保存第二次编程中下页的值，必须区分下位值“0”或“1”。对于从（1,0）到（0,0）的转换，将所讨论的存储器单元编程为存储器状态“Y”。对于从（1,1）到（0,1）的转换，将所讨论的存储器单元编程为存储器状态“Z”。以此方式，在读取期间，可通过确定在单元中编程的存储器状态来解码下页位和上页位。

然而，葛莱码2次编程方案在第二次编程错误时可能产生问题。例如，在下位为“1”时将上页位编程为“0”，将造成（1,1）转换成（0,1)。此需要将存储器单元累进地从“U”进行编程通过“X”和“Y”而到达“Z”。如果在完成编程之前发生电源中断，那么存储器单元可能在转换存储器状态的一者中（比如，“X”）结束。在读取存储器单元时，“X”将被解码为逻辑状态（1,0)。此对于上位和下位造成不正确的结果，因为其本应该为（0,1)。类似地，如果编程在达到“Y”时受到中断，那么其将对应于（0,0)。虽然上位现为正确的，但下位仍有误。

因此，可以看出上页编程的问题可破坏已处于下页中的数据。尤其当第二次编程涉及在中间存储器状态上通过时，程序中止可能使编程在所述存储器状态中结束，从而造成解码不正确的下页位。

图42说明通过在复制项中保存每一扇区来保护关键数据的方式。例如，将扇区A、B、

C和D保存在复制副本中。如过一个扇区副本中存在数据破坏，那么可改为读取另一扇区副本。

图43说明通常将复制的扇区保存在多状态存储器中的非稳固性。如上文所描述，在示范性4状态存储器中，多状态页实际上包括分别在两次编程中编程的逻辑下页和逻辑上页。在所示的实例中，页为四个扇区宽。因此，扇区A及其复制项将同时编程在逻辑下页中，且类似地，对于扇区B及其复制项也是如此。接着在逻辑上页中的后续第二次编程中，将同时编程扇区C, C，且对于扇区D, D也是如此。如果在编程扇区C, C的中间发生程序中止，那么将破坏下页中的扇区A, A。除非在上页编程之前首先读取并缓冲了下页扇区，否则下页扇区如果受到破坏可能无法修复。因此，同时保存关键数据的两个副本（例如，扇区A, A）无法防止其由于其上页中对扇区C, C的后续有疑问的保存而受到破坏。

图44A说明将关键数据的交错的复制副本保存到多状态存储器的一个实施例。基本上，以与图43相同的方式保存下页，即，扇区A, A和扇区B, B。然而，在上页编程中，扇区C和D与其复制项交叉成C, D, C, D。如果支持局部页编程，那么可同时编程扇区C的两个副本，且对于扇区D的两个副本也是如此。如果（比如）两个扇区C的程序遭到中止，那么只可在扇区A的一个副本和扇区B的一个副本上破坏下页。其它副本维持不受影响。因此，如果在第一次编程中存储的关键数据存在两个副本，那么其应该不会受到同时进行的后续第二次编程影响。

图44B说明只将关键数据的复制副本保存到多状态存储器的逻辑方页的另一实施例。在此情况下，不使用下页中的数据。关键数据及其复制项（例如，扇区A, A和扇区B, B）只保存到逻辑上页。以此方式，如果存在程序中止，那么可将关键数据重写在另一逻辑上页中，而对下页数据的任何破坏将无关紧要。此办法基本上使用每一多状态页的存储容量的一半。

图44C说明以多状态存储器的二进制模式保存关键数据的复制副本的又一实施例。在此情况下，以二进制模式编程每一存储器单元，其中仅将其阈值范围划分成两个区域。因此仅存在一次编程，且在发生程序中止时可在不同位置中重新启动编程。此办法也使用每一多状态页的存储容量的一半。美国专利第6,456,528 B1号中描述了以二进制模式操作多状态存储器，所述专利的全部揭示内容以引用的方式并入本文中。

图45说明同时将关键数据的复制副本保存到两个不同元区块的又一实施例。如果区块的一者变成不可用，那么可从另一区块读取数据。例如，关键数据包含在扇区A、B、C、D和E、F、G、H和I、J、K、L中。每一扇区保存在复制项中。两个副本将同时写入

到两个不同的区块，区块0和区块1。如果将一个副本写入到逻辑下页，那么另一副本将写入到逻辑上页。以此方式，将始终存在编程到逻辑上页的副本。如果发生程序中止，那么可将其重新编程到另一逻辑上页。同时，如果下页遭受破坏，那么在另一区块中始终存在另一上页副本。

图46B说明通过使用容错码来同时保存关键数据的复制副本的又一实施例。图46A在说明4状态存储器阵列的阈值电压分布方面与图41A类似，且展示为图46B的参考。容错码实质上避免在任何中间状态中转换的任何上页编程。因此，在第一次下页编程中，逻辑状态（1, 1）转换为（1, 0），如表示为将已擦除的存储器状态“U” 编程为“Y”。在将上页位编程为“0”的第二次编程中，如果下页位为“1”，那么逻辑状态（1, 1）转换为（0, 1），如表示为将已擦除的存储器状态“U”编程为“X”。如果下页位为“0”，那么逻辑状态（1, 0）转换为（0, 0），如表示为将存储器状态“Y”编程为“Z”。由于上页编程仅涉及编程为下一邻近的存储器状态，因此程序中止不改变下页位。

串列写入

关键数据的复制副本优选地如上所述被同时写入。避免同时破坏两个副本的另一方式是循序写入副本。此方法较缓慢，但副本本身指示在控制器检查两个副本时其编程是否成功。

图47是展示两个数据副本的可能状态和数据有效性的表。

如果第一和第二副本没有ECC错误，那么认为数据的编程已完全成功。可从任一副本获得有效数据。

如果第一副本没有ECC错误但第二副本具有ECC错误，那么可意味着编程在第二副本编程的中间受到中断。第一副本含有有效数据。即使错误为可校正的，第二副本数据也不可信赖。

如果第一副本没有ECC错误且第二副本已清空（擦除），那么可意味着编程在第一副本编程结束之后但在第二副本开始之前受到中断。第一副本含有有效数据。

如果第一副本有ECC错误且第二副本已清空（擦除），那么可意味着编程在第一副本编程的中间受到中断。即使错误为可校正的，第一副本可能仍含有无效数据。

为了读取维持在复制项中的数据，以下技术为优选的，因为其利用复制副本的存在。读取并比较两个副本。在此情况下，图47所示的两个副本的状态可用来确保不存在错误误测。

在另一实施例中，控制器只读取一个副本，为了顾及速度和简单性，优选地在两个副本之间交替进行副本读取。例如，在控制器读取控制数据时，其读取（比如）副本1，

下一控制读取（任何控制读取）应来自副本2，接着再次为副本1等。以此方式，将可定期读取两个副本并检查两个副本的完整性（ECC检查）。此减少无法在因恶化的数据保留所造成的时间错误中进行检测的风险：。例如，如果通常只读取副本1，那么副本2可能逐渐恶化达到错误再也无法被ECC挽救的程度，且无法再使用第二副本。

占先式数据重新定位

如结合图20所述，区块管理系统在其操作期间在快闪存储器中维持一组控制数据。此组控制数据存储在与主机数据相同的元区块中。因此，控制数据本身将受到区块管理，且将受到更新的影响，并因此受到垃圾收集操作的影响。

也已描述了控制数据的层级，其中较低层级中的控制数据比较高层级中的控制数据更新得更为频繁。例如，假设每一控制区块具有N个将要写入的控制扇区，那么通常发生控制更新和控制区块重新定位的以下序列。再次参看图20，每N个CBI更新填满CBI区块并触发CBI重新定位（重写）和MAP更新。如果混乱区块被关闭，那么也触发GAT更新。每一GAT更新触发MAP更新。每N个GAT更新填满区块并触发GAT区块重新定位。此外，当MAP区块变满时，也触发MAP区块重新定位和MAPA区块（如果存在的话，否则BOOT区块直接指向MAP）更新。此外，当MAPA区块变满时，也触发MAPA区块重新定位、BOOT区块更新和MAP更新。此外，在BOOT区块变满时，触发另一BOOT区块的活跃BOOT区块重新定位。

由于通过顶部处的BOOT控制数据，随后是MAPA、MAP，且接着GAT来形成层级，因此，在每N^3个GAT更新中，将存在“级联控制更新”，其中将重新定位所有的GAT、MAP、MAPA和BOOT区块。在此情况下，在因主机写入导致的混乱或循序更新区块关闭而引起GAT更新时，也存在垃圾收集操作（即，重新定位或重写）。在混乱更新区块垃圾收集的情况下，将更新CBI，而这也会触发CBI区块重新定位。因此，在此极端的情况中，需要同时收集大量元区块的垃圾。

可以看到，层级的每一控制数据区块在得以填补和重新定位方面具有其自身的周期性。如果每一控制数据区块运作正常，那么将存在发生以下情形的时候：大量区块的阶段将排队并触发同时涉及所有那些区块的大量重新定位或垃圾收集。许多控制区块的重新定位将花费较长时间，且应加以避免，因为部分主机不容许因大量控制操作而造成的长时间延迟。

根据本发明的另一方面，在具有区块管理系统的非易失性存储器中，实施存储器区块的“控制垃圾收集”或占先式重新定位以避免发生大量的更新区块均恰巧同时需要进行重新定位的情况。例如，在更新用于控制区块管理系统的操作的控制数据时可发生此

情况。控制数据类型的层级可以不同程度的更新频率而存在，从而导致其关联的更新区块需要以不同速率进行垃圾收集或重新定位。将存在一个以上控制数据类型的垃圾收集操作同时发生的某些时间。在极端的情况中，所有控制数据类型的更新区块的重新定位阶段可排队，从而导致所有的更新区块均需要同时重新定位。

本发明避免了这种不合需要的情况，其中当前的存储器操作无论何时均可适应自发性的垃圾收集操作，更新区块的占先式重新定位预先在完全填充区块之前发生。明确地说，将优先权提供给具有最慢速率的处于最高层级的数据类型的区块。以此方式，在重新定位最慢速率区块之后，将不需要进行相对较长时间的另一垃圾收集。并且，处于较高层级的较慢速率区块不具有许多将要触发的重新定位的级联。可将本发明方法视为：为了避免所讨论的各种区块的阶段对准而将某种抖动（dithering）引入到事物的整体混合中。因此，只要存在机会，将占先地重新定位具有不被完全填充的微小容限的缓慢填充区块。

在处于较低层级的控制数据因级联效应而比处于较高阶层的控制数据变化得更快的控制数据层级系统中，将优先权提供给处于较高阶层的控制数据的区块。执行自发性占先式重新定位的机会的一个实例发生在以下情况时：主机写入本身不触发重新定位，因此可利用其等待时间时段中的任何剩余时间来进行占先式重新定位操作。一般而言，务必需要重新定位的区块前的容限是在区块全满之前预定数目的未写入存储器单位。所考虑的容限足以促成在完全填充的区块前但不过早而导致资源浪费的重新定位。在优选实施例中，预定数目的未写入存储器单位在一与六个存储器单位之间。

图48说明存储控制数据的存储器区块的占先式重新定位的流程图。

步骤1202：将非易失性存储器组织成区块，每一区块分割成可一起擦除的存储器单位。

步骤1204：维持不同类型的数据。

步骤1206：对不同类型的数据配置等级（ranking）。

步骤1208：将所述不同类型数据的更新存储在复数个区块中，使得每一区块实质上存储相同类型的数据。

步骤1210：响应于具有少于预定数目的清空存储器单位且具有所述复数个区块中最高等级的数据类型的区块，将所述区块的数据的当前更新重新定位到另一区块。如果未受到中断，进行步骤1208。

用于实施图20所示控制数据的占先式重新定位的示范性算法如下：

如果（（不存在因用户数据而造成的垃圾收集）或（MAP留有6个或更少的未写入扇

区）或（GAT留有3个或更少的未写入扇区）

那么

如果（BOOT留有1个未写入扇区）

那么重新定位BOOT（即，重新定位到区块）

否则

如果（MAPA留有1个未写入扇区）

那么重新定位MAPA并更新MAP

否则

如果（MAP留有1个未写入扇区）

那么重新定位MAP

否则

如果（上一已更新或最大的GAT留有1个未写入扇区）

那么重新定位GAT

否则

如果（CBI留有1个未写入扇区）

那么重新定位CBI

否则

否则

退出

因此，占先式重新定位通常在不发生用户数据垃圾收集时完成。在最糟的情况下，当每一主机写入触发用户数据垃圾收集，但存在足够的时间进行一个区块的自发性重新定位时，一次可执行一个控制区块的占先式重新定位。

由于用户数据垃圾收集操作和控制更新可能与物理错误同时发生，所以较佳地通过事先（比如在区块仍具有2个或更多未写入存储器单位（例如，扇区）时）先行进行占先式重新定位或受控的垃圾收集来获得较大的安全容限。

虽然已参照特定实施例描述了本发明的各种方面，但应了解，本发明受所附权利要求书的完整范围的保护。

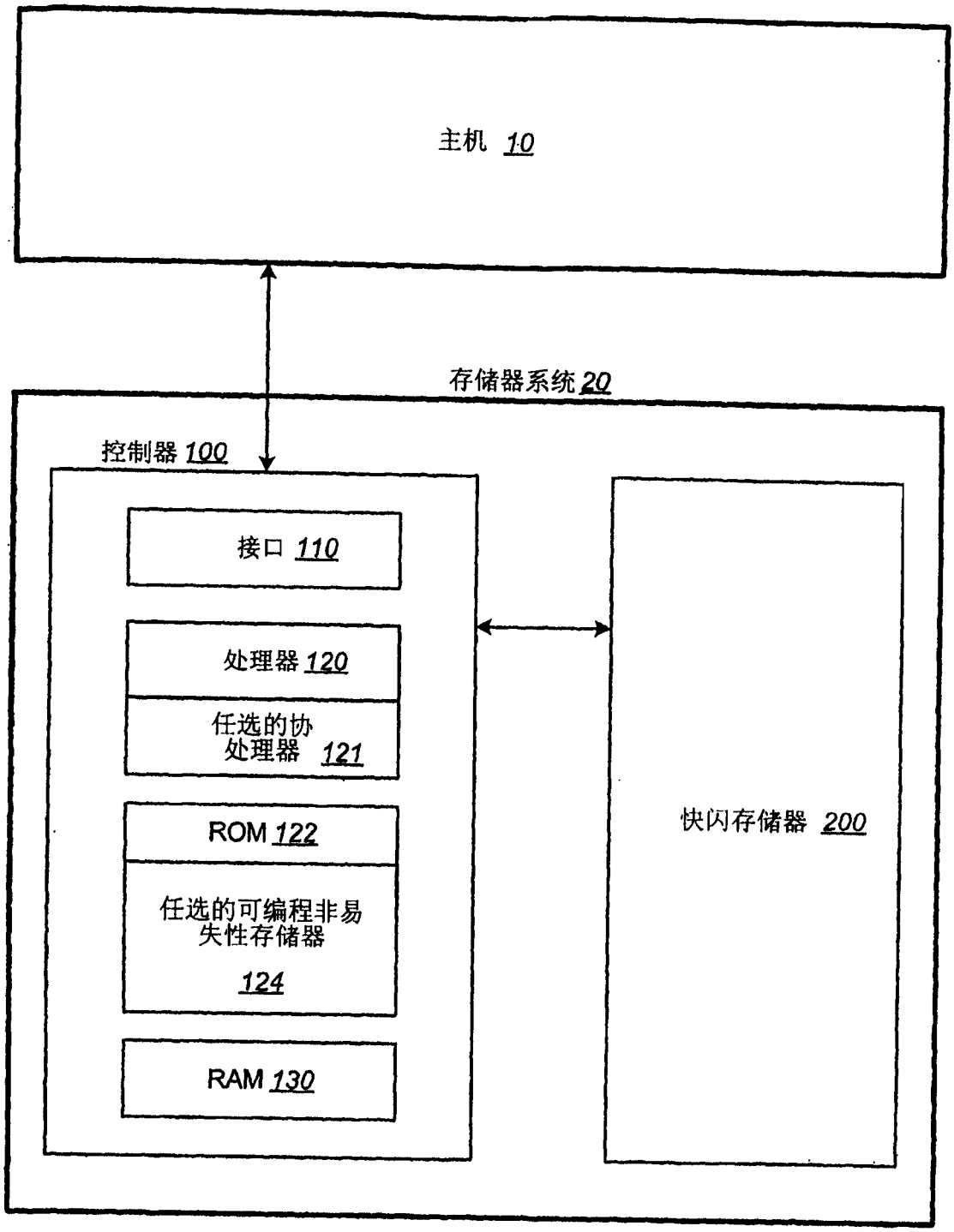

主机 10
存储器系统 20
控制器 100
接口 110
处理器 120
任选的协处理器 121
ROM 122
任选的可编程非易失性存储器 124
RAM 130
快闪存储器 200

图 1

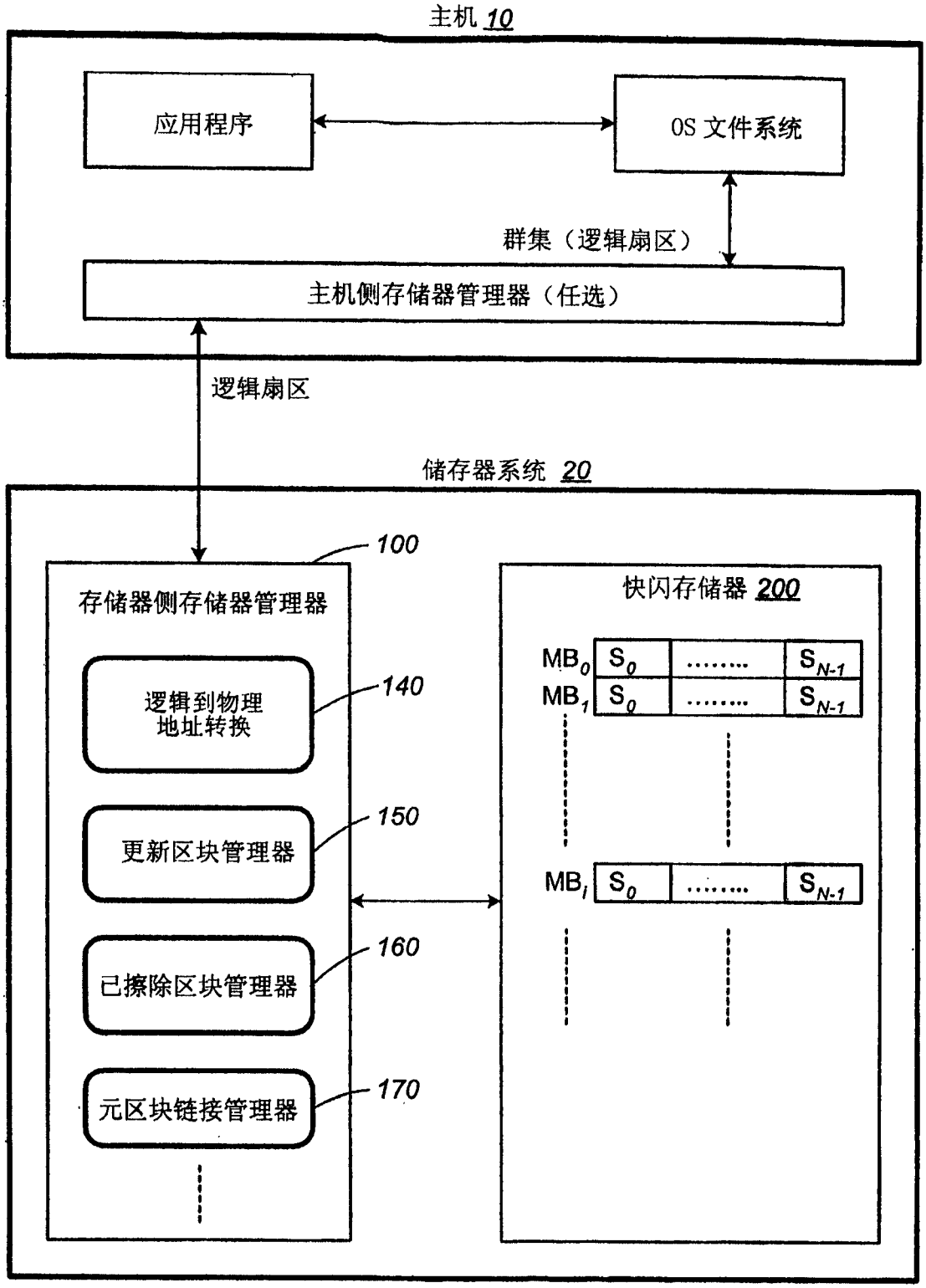

主机 10
应用程序
OS 文件系统
群集（逻辑扇区）
主机侧存储器管理器（任选）
逻辑扇区
储存器系统 20
100
存储器侧存储器管理器
逻辑到物理
地址转换
140
更新区块管理器
150
已擦除区块管理器
160
元区块链接管理器
170
快闪存储器 200
MB0 S0 ..... SN-1
MB1 S0 ..... SN-1
MBI S0 ..... SN-1

图 2

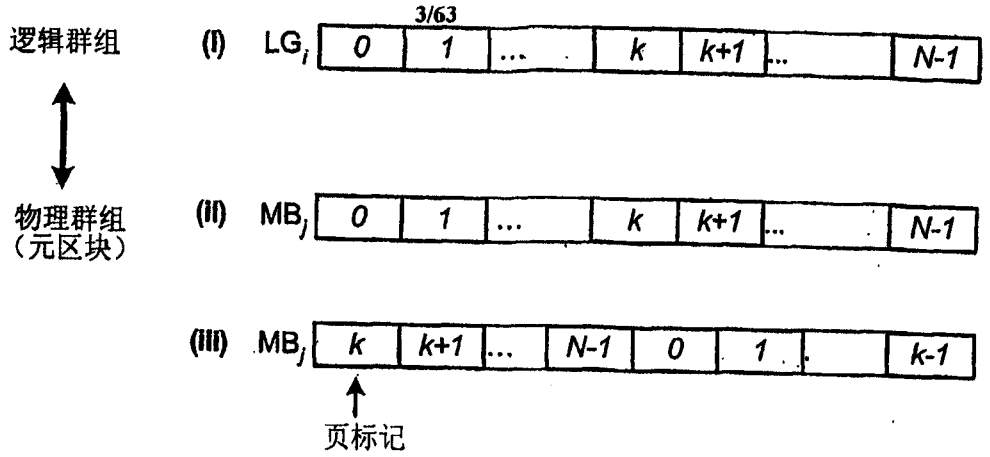

3/63
逻辑群组
(I) LG$_i$
0
1
...
k
k+1
..
N-1
物理群组
(元区块)
(ii) MB$_j$
0
1
...
k
k+1
..
N-1
(iii) MB$_j$
k
k+1
...
N-1
0
1
k-1
页标记

图 3A

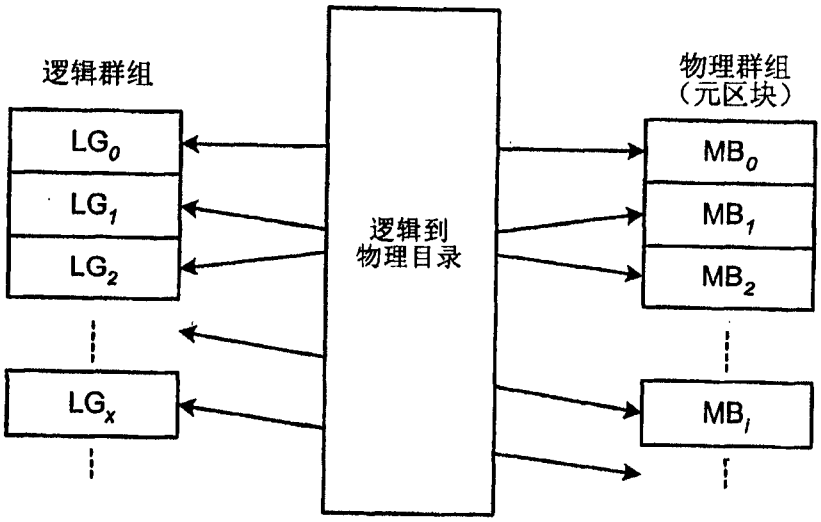

逻辑群组
LG$_0$
LG$_1$
LG$_2$
LG$_x$
逻辑到
物理目录
物理群组
(元区块)
MB$_0$
MB$_1$
MB$_2$
MB$_i$

图 3B

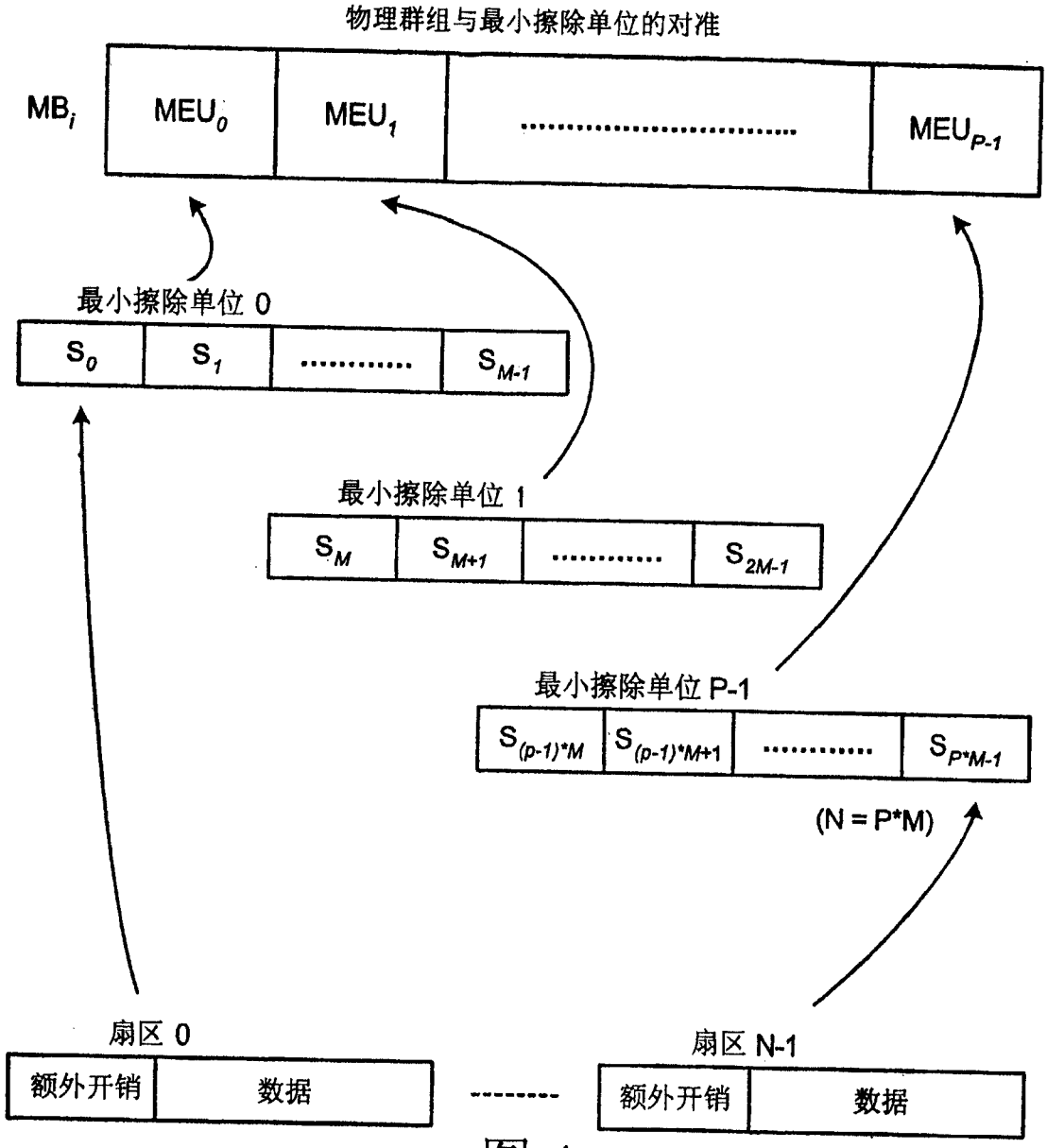

物理群组与最小擦除单位的对准
MB_i
MEU_0
MEU_1
MEU_P-1
最小擦除单位 0
S_0
S_1
S_M-1
最小擦除单位 1
S_M
S_M+1
S_2M-1
最小擦除单位 P-1
S_(p-1)*M
S_(p-1)*M+1
S_P*M-1
(N = P*M)
扇区 0
额外开销
数据
扇区 N-1
额外开销
数据

图 4

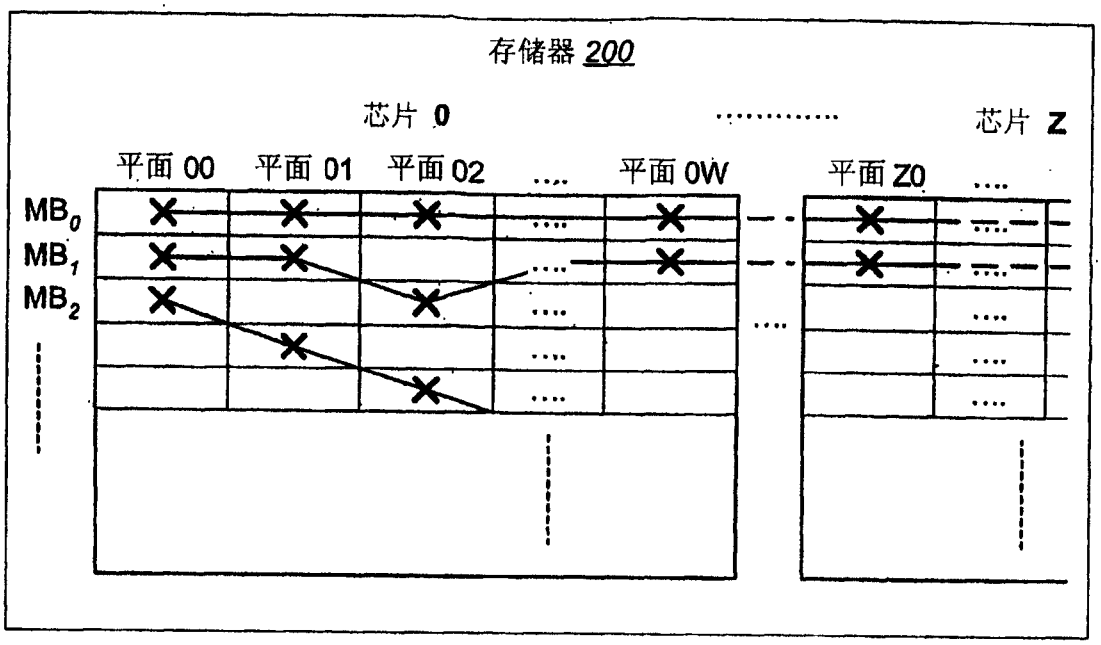


图 5A

图 5B

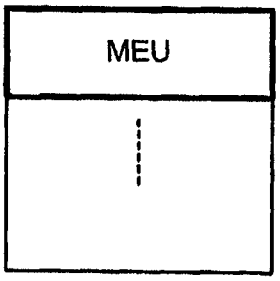


图 5C

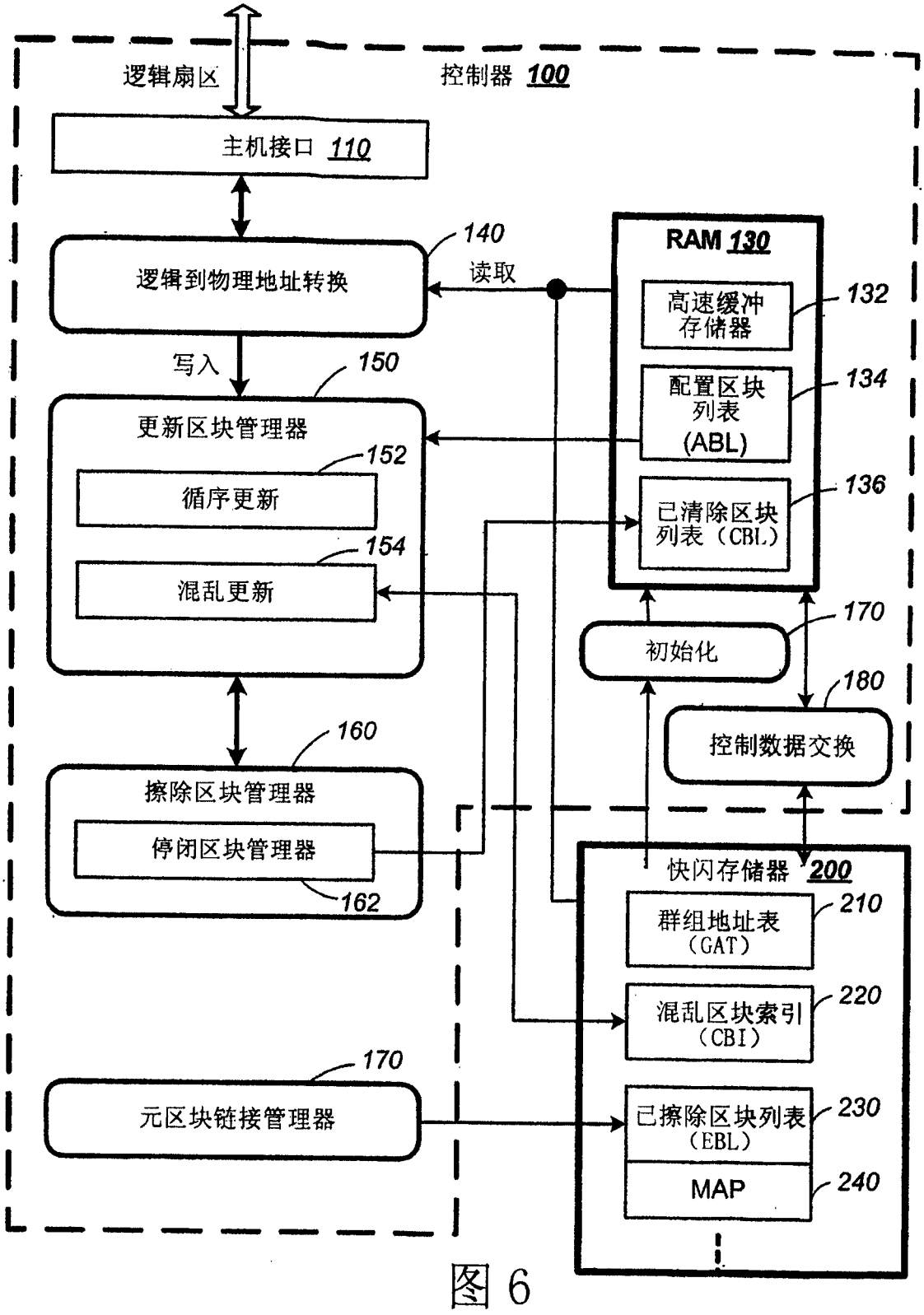
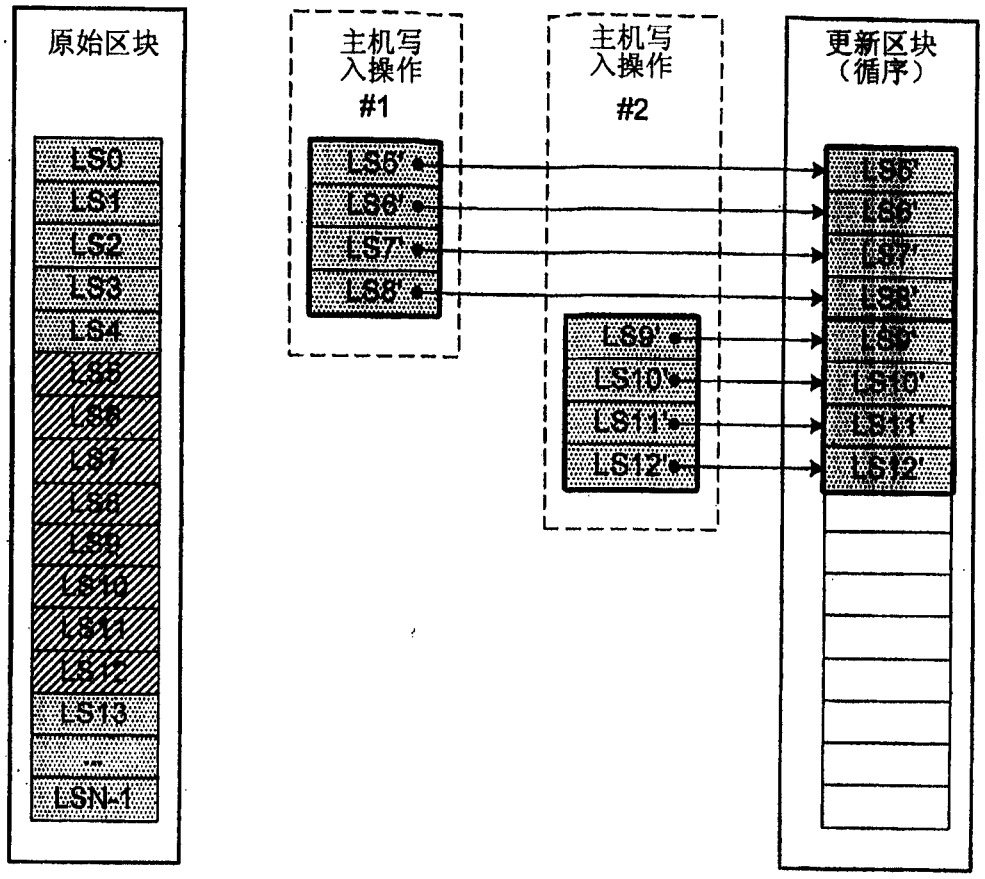
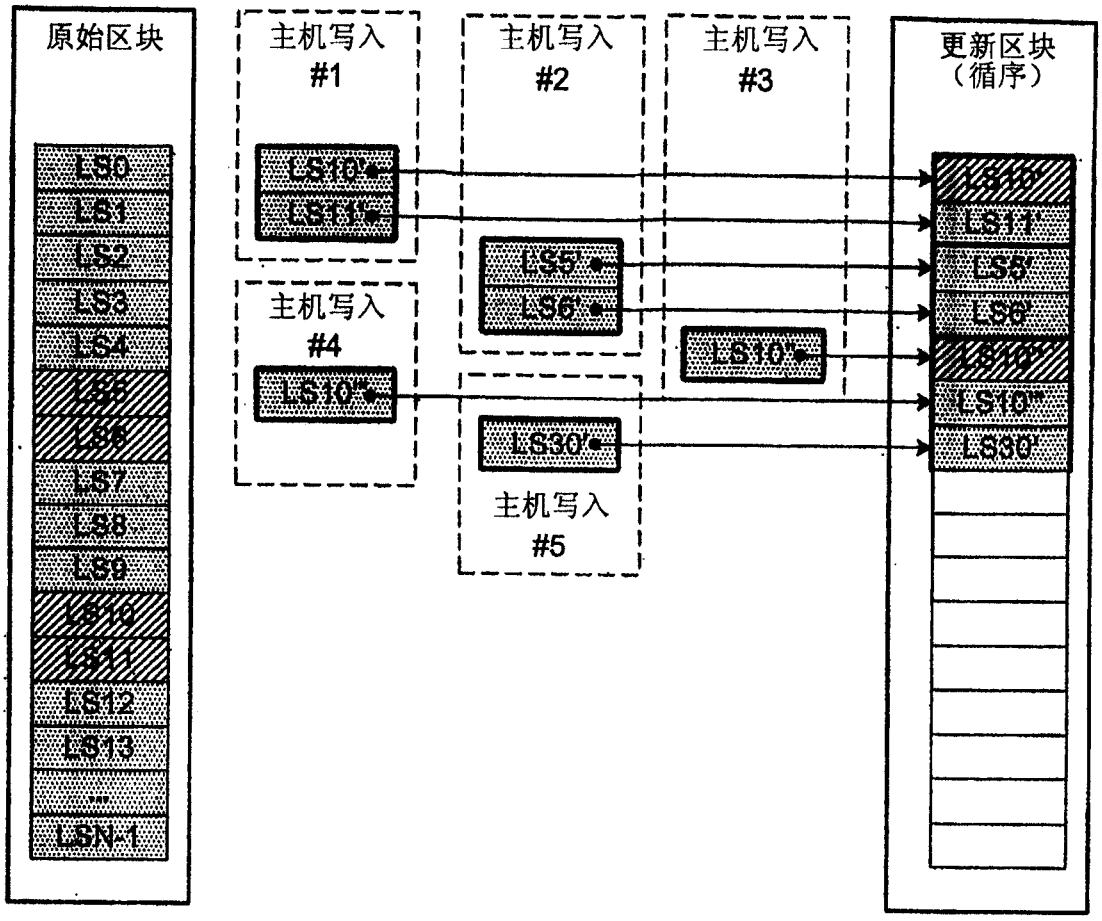

逻辑扇区
控制器 100
主机接口 110
逻辑到物理地址转换
140
读取
写入
150
更新区块管理器
152
循序更新
154
混乱更新
160
擦除区块管理器
停闭区块管理器
162
170
元区块链接管理器
RAM 130
高速缓冲存储器
132
配置区块列表 (ABL)
134
已清除区块列表（CBL）
136
初始化
170
180
控制数据交换
快闪存储器 200
群组地址表（GAT）
210
混乱区块索引（CBI）
220
已擦除区块列表（EBL）
230
MAP
240

图 6


原始区块
LS0
LS1
LS2
LS3
LS4
LS5
LS6
LS7
LS8
LS9
LS10
LS11
LS12
LS13
...
LSN-1
主机写入操作 #1
LS6'
LS6'
LS7'
LS8'
主机写入操作 #2
LS9'
LS10'
LS11'
LS12'
更新区块（循序）
LS6'
LS6'
LS7'
LS8'
LS9'
LS10'
LS11'
LS12'
图例：
已擦除
有效

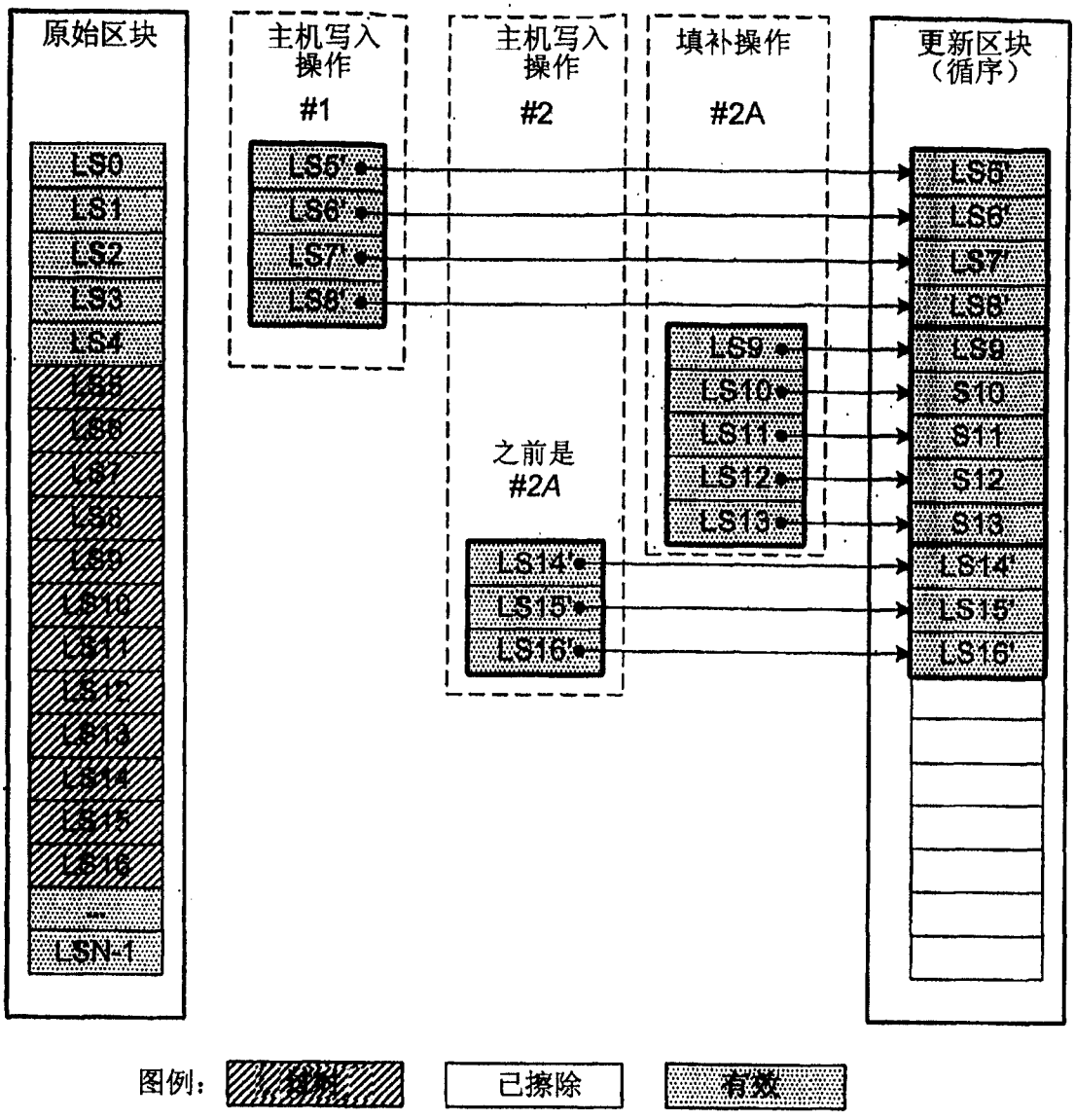
循序更新实例

图 7A



混乱更新（非循序）实例

图 7B


原始区块
主机写入操作 #1
主机写入操作 #2
填补操作 #2A
更新区块（循序）
LS0
LS1
LS2
LS3
LS4
LS5
LS6
LS7
LS8
LS9
LS10
LS11
LS12
LS13
LS14
LS15
LS16
...
LSN-1
LS5'
LS6'
LS7'
LS8'
LS9'
LS10'
LS11'
LS12'
LS13'
之前是 #2A
LS14'
LS15'
LS16'
图例：
已擦除
有效

强制循序更新实例

图 8

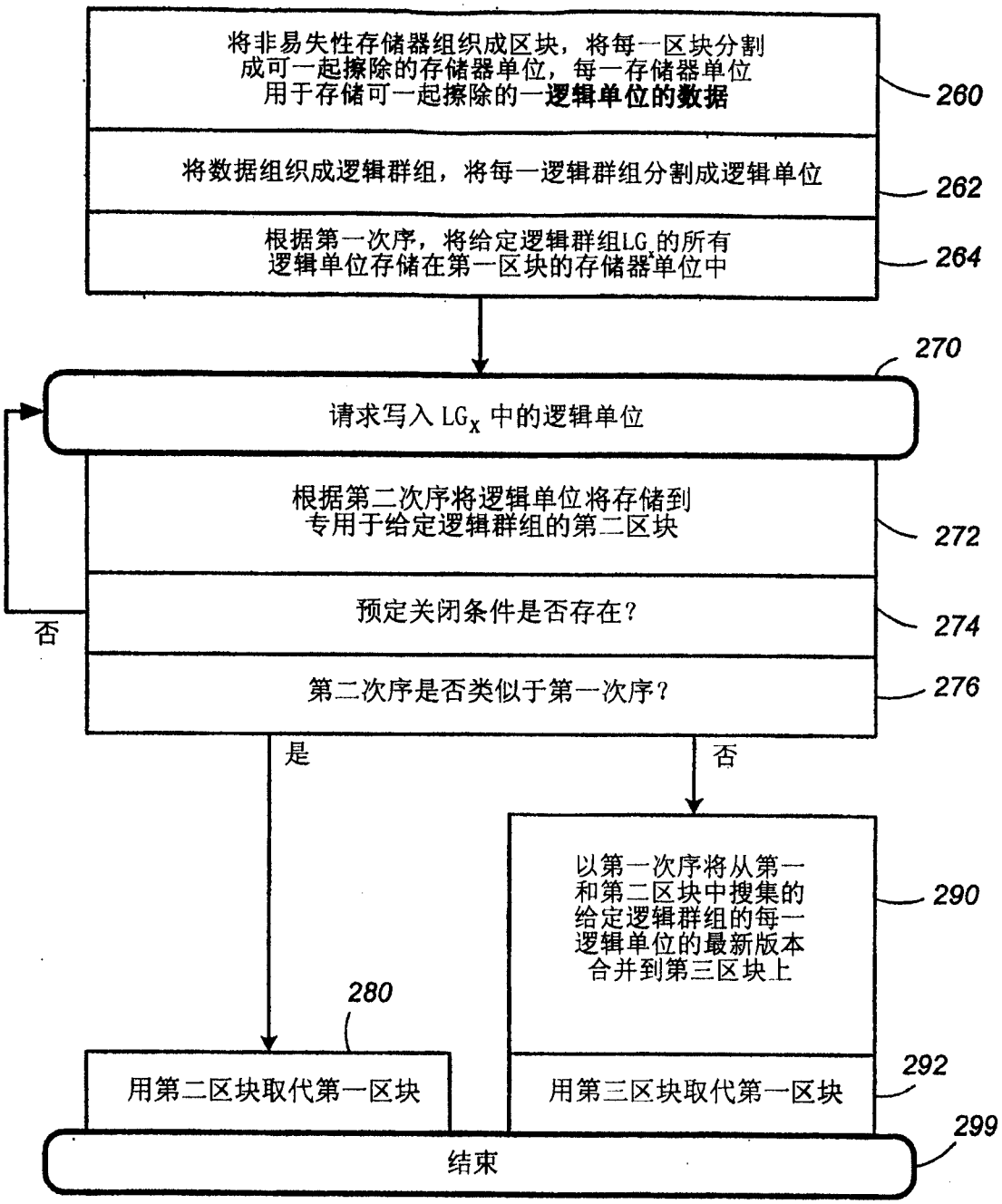

将非易失性存储器组织成区块，将每一区块分割成可一起擦除的存储器单位，每一存储器单位用于存储可一起擦除的一逻辑单位的数据
260
将数据组织成逻辑群组，将每一逻辑群组分割成逻辑单位
262
根据第一次序，将给定逻辑群组LG的所有逻辑单位存储在第一区块的存储器单位中
264
270
请求写入 LG_X 中的逻辑单位
根据第二次序将逻辑单位将存储到专用于给定逻辑群组的第二区块
272
预定关闭条件是否存在？
274
否
第二次序是否类似于第一次序？
276
是
否
以第一次序将从第一和第二区块中搜集的给定逻辑群组的每一逻辑单位的最新版本合并到第三区块上
290
280
用第二区块取代第一区块
用第三区块取代第一区块
292
结束
299

图 9

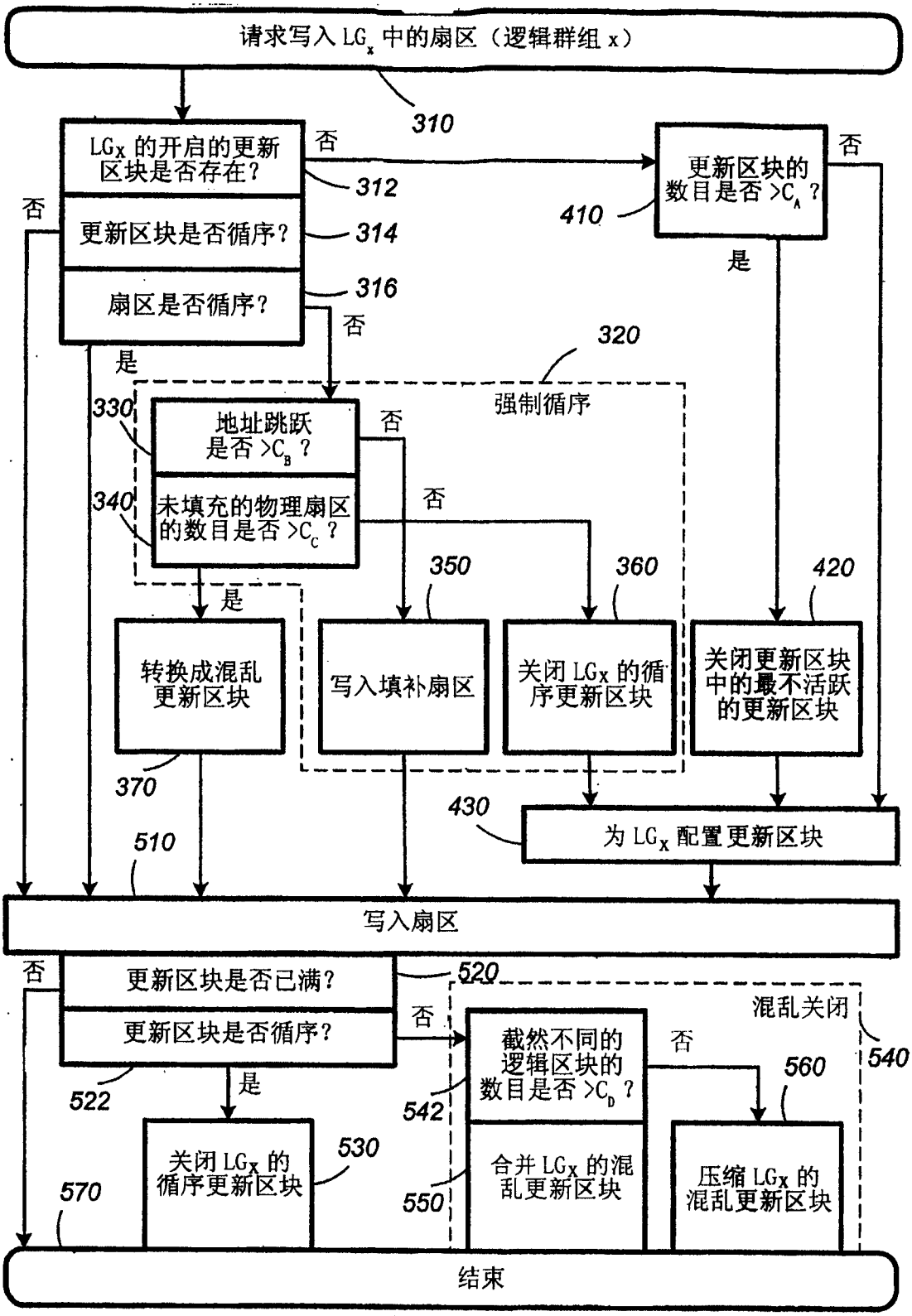

请求写入 LG$_x$ 中的扇区（逻辑群组 x）
310
LG$_X$ 的开启的更新区块是否存在？
312
更新区块是否循序？
314
扇区是否循序？
316
否
是
更新区块的数目是否 >C$_A$ ？
410
是
否
320
强制循序
330
地址跳跃是否 >C$_B$ ？
340
未填充的物理扇区的数目是否 >C$_C$ ？
否
否
是
350
360
420
转换成混乱更新区块
写入填补扇区
关闭 LG$_X$ 的循序更新区块
关闭更新区块中的最不活跃的更新区块
370
430
为 LG$_X$ 配置更新区块
510
写入扇区
否
更新区块是否已满？
更新区块是否循序？
520
混乱关闭
540
否
截然不同的逻辑区块的数目是否 >C$_D$ ？
560
522
是
542
关闭 LG$_X$ 的循序更新区块
530
570
合并 LG$_X$ 的混乱更新区块
550
压缩 LG$_X$ 的混乱更新区块
结束

图 10

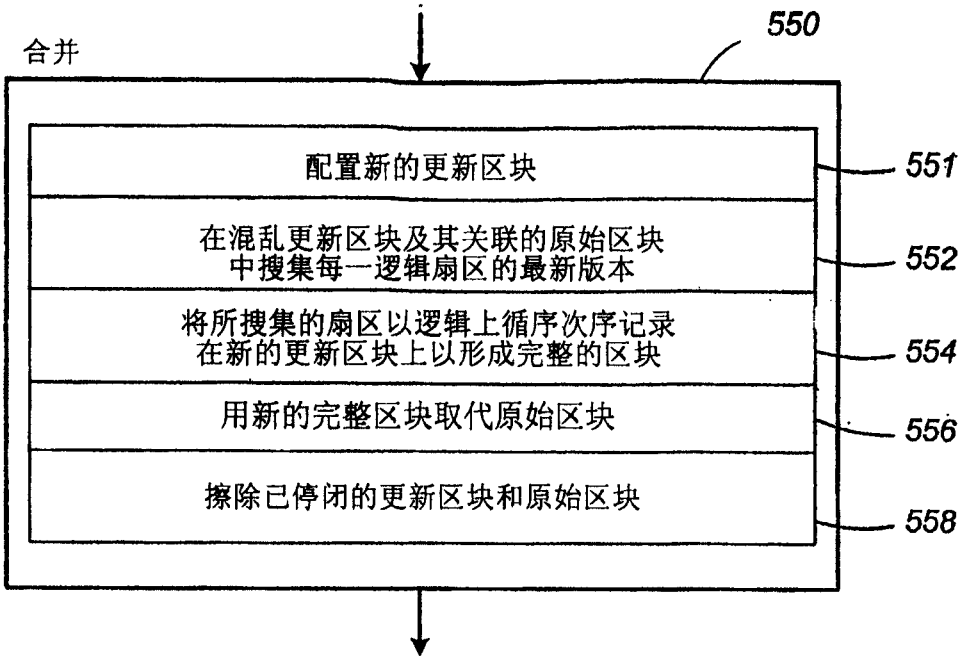

合并
550
配置新的更新区块
551
在混乱更新区块及其关联的原始区块中搜集每一逻辑扇区的最新版本
552
将所搜集的扇区以逻辑上循序次序记录在新的更新区块上以形成完整的区块
554
用新的完整区块取代原始区块
556
擦除已停闭的更新区块和原始区块
558

图 11A

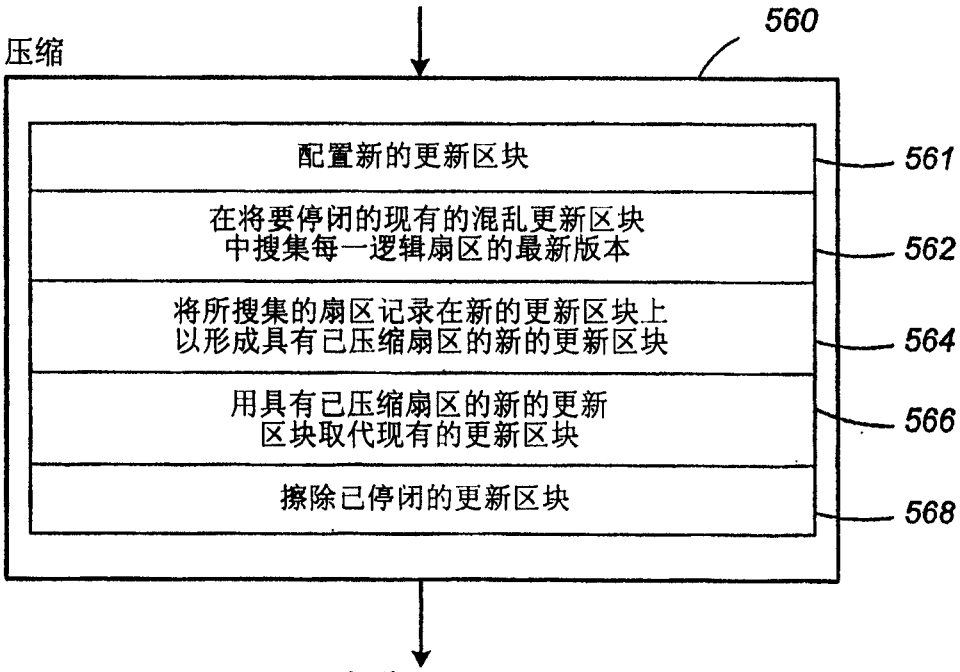

压缩
560
配置新的更新区块
561
在将要停闭的现有的混乱更新区块中搜集每一逻辑扇区的最新版本
562
将所搜集的扇区记录在新的更新区块上以形成具有已压缩扇区的新的更新区块
564
用具有已压缩扇区的新的更新区块取代现有的更新区块
566
擦除已停闭的更新区块
568

图 11B

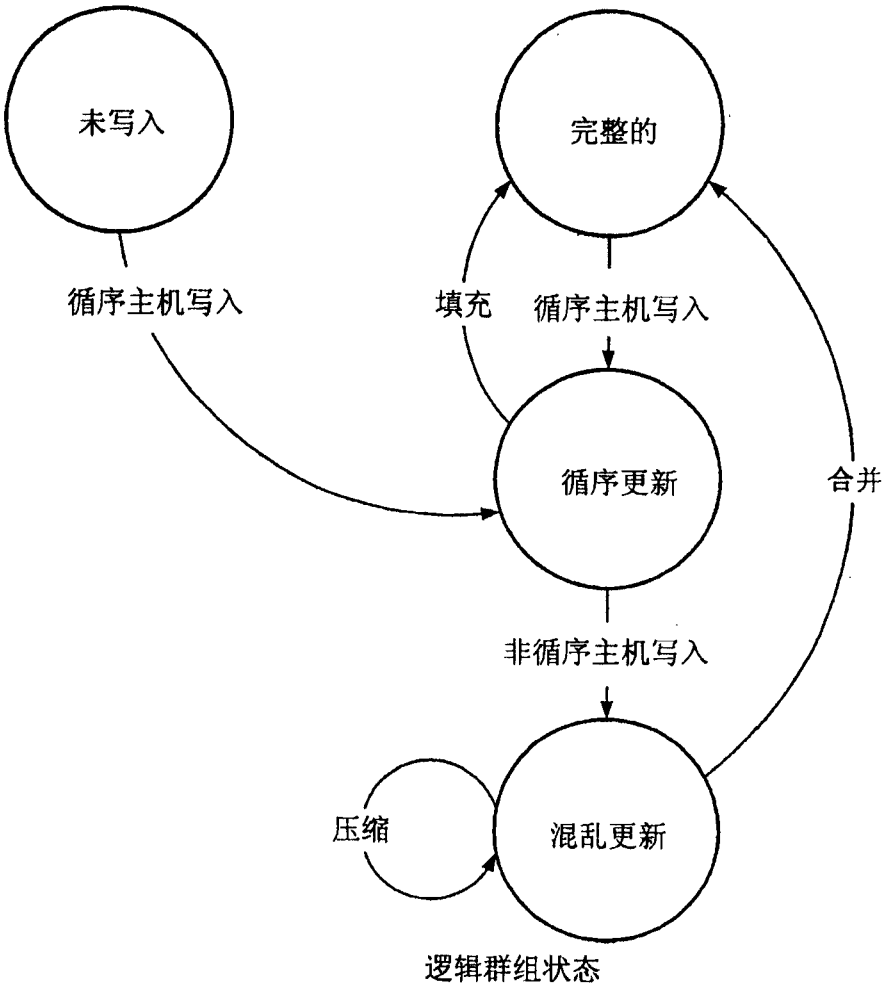


图 12A

逻辑群组状态	描述
完整的	以逻辑上循序次序在单一元区块中完成逻辑群组
未写入	逻辑群组的任何部分均未被写入
循序更新	将逻辑群组的更新程序段写入到更新区块而 不改变其现有的循序属性
混乱更新	以逻辑上非循序次序将逻辑群组的更新程序段写入到更新区块

图 12B

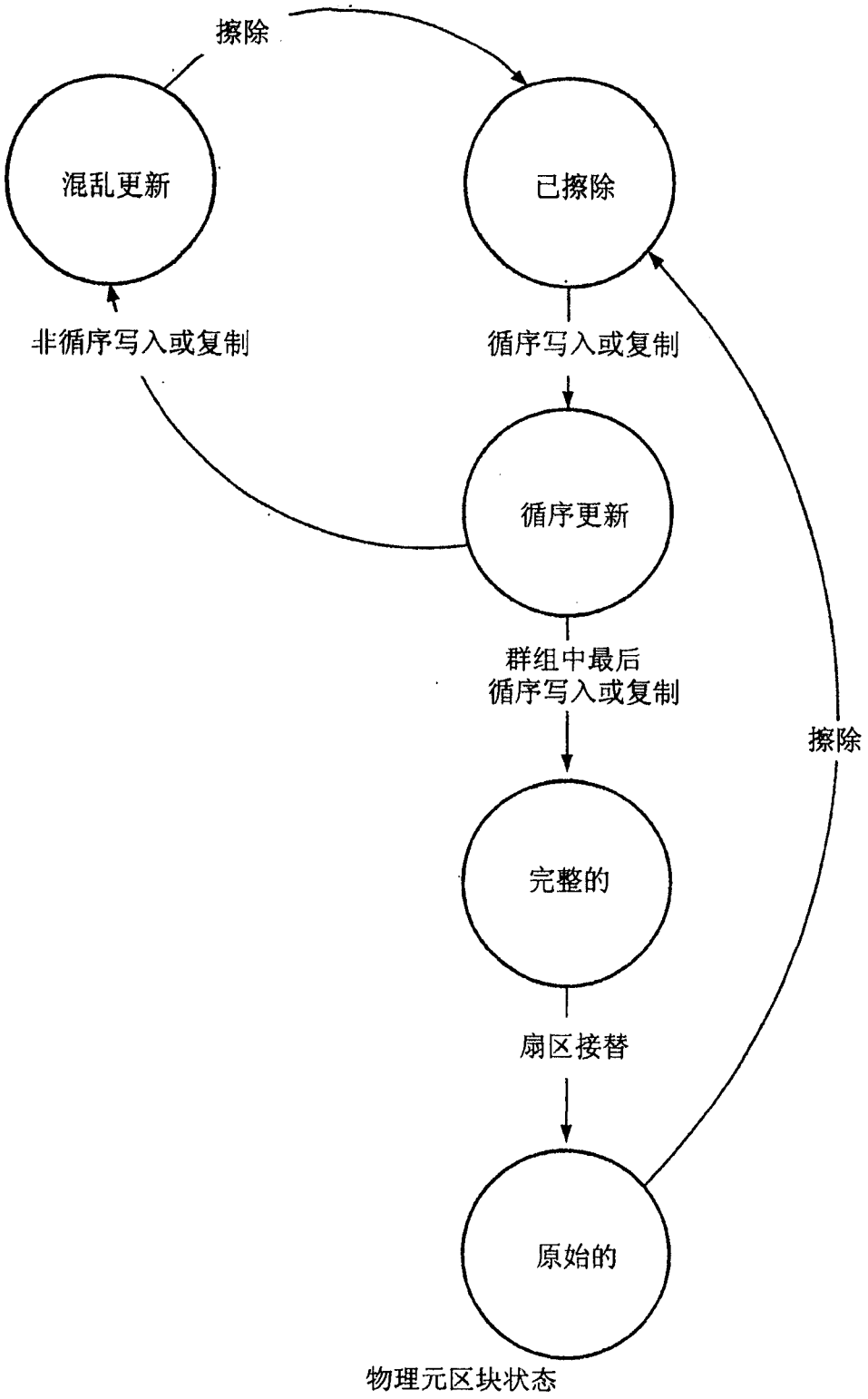

擦除
混乱更新
已擦除
非循序写入或复制
循序写入或复制
循序更新
群组中最后
循序写入或复制
擦除
完整的
扇区接替
原始的
物理元区块状态

图 13A

元区块状态	描述
已擦除	擦除了元区块中的所有扇区
循序更新	可能使用页标记部分地写入元区块，其中扇区为逻辑上循序次序。所有扇区均属于相同的逻辑群组
非循序（混乱）更新	部分或完全写入元区块，其中扇区为逻辑上非循序次序。任何扇区均可被写入一次以上。所有扇区均属于相同的逻辑群组。
完整的	可能使用页标记以逻辑上循序次序完全写入了元区块。
原始的	元区块先前为完整的但至少一个扇区已因主机数据更新而过时

图 13B

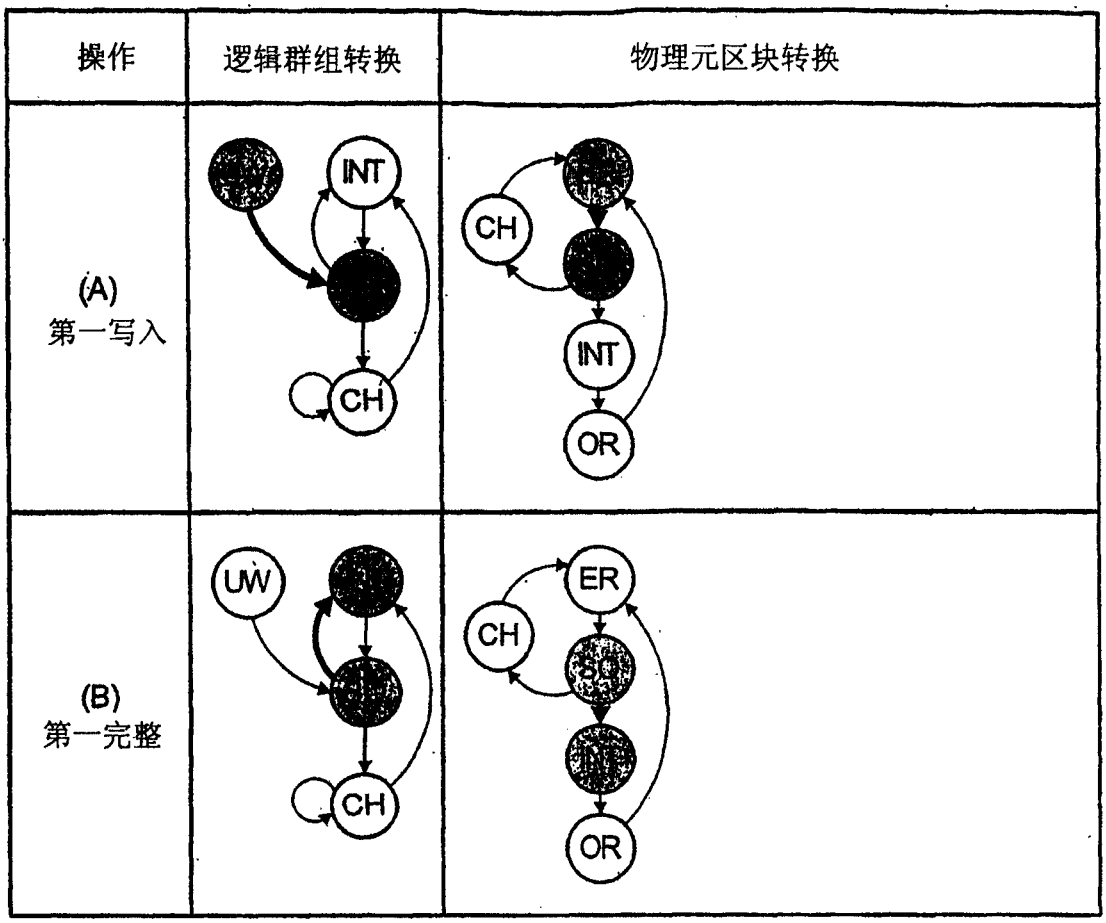


图 14（A）到图 14（B）

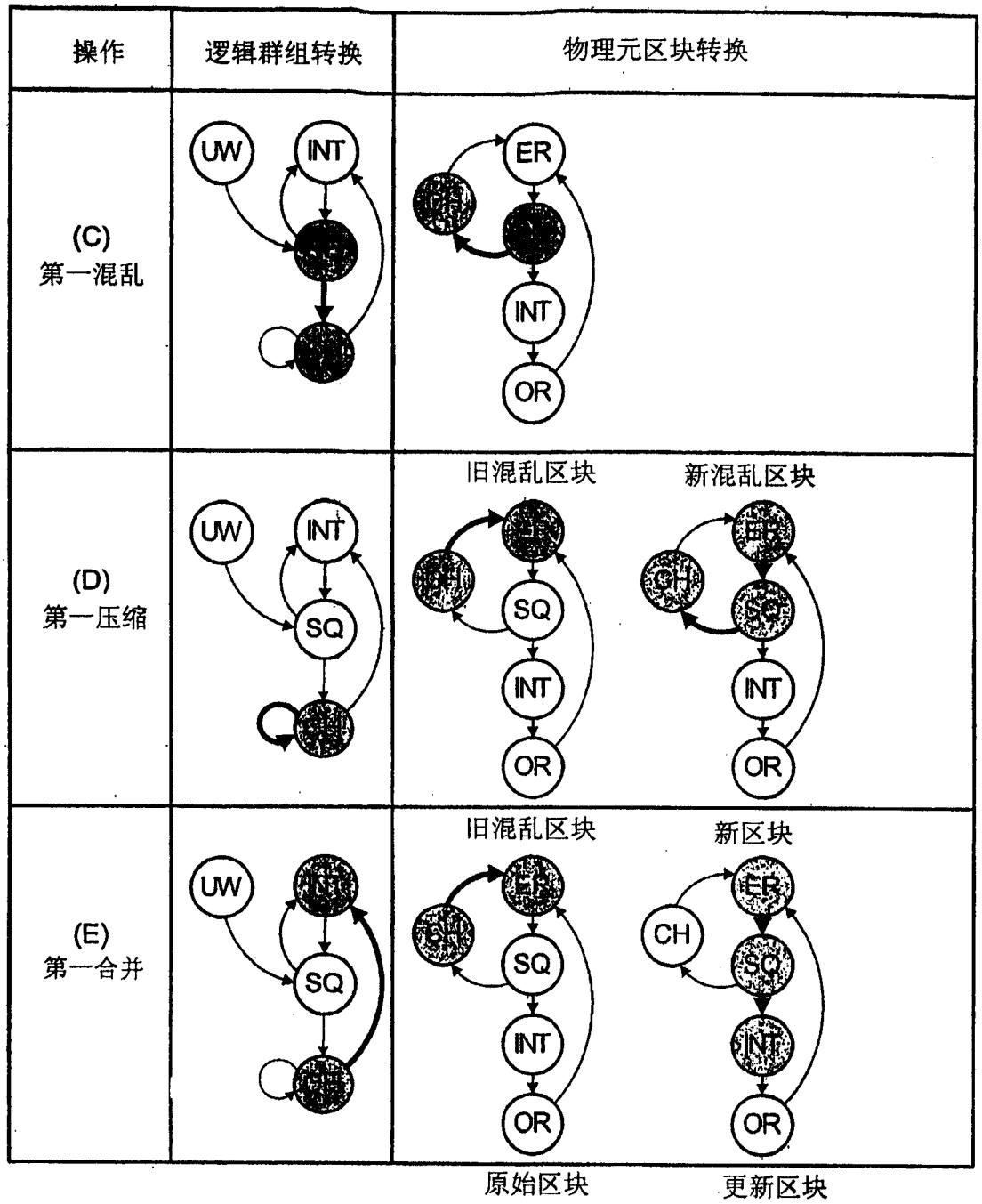

操作
逻辑群组转换
物理元区块转换
(C)
第一混乱
UW
INT
ER
CH
INT
OR
(D)
第一压缩
UW
INT
SQ
旧混乱区块
新混乱区块
ER
CH
SQ
INT
OR
(E)
第一合并
UW
INT
SQ
旧混乱区块
新区块
CH
原始区块
更新区块

图 14（C）到图 14（E）

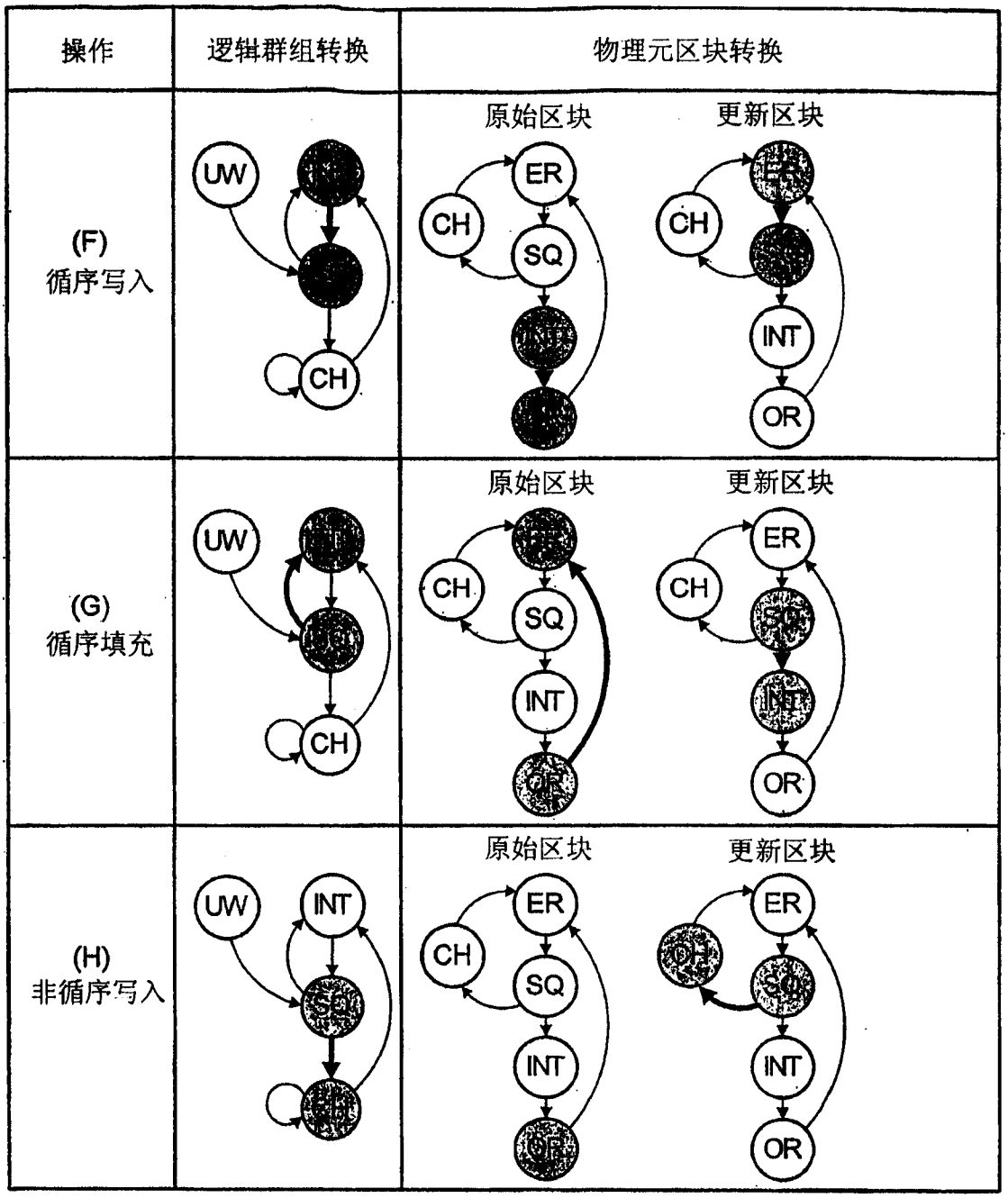

操作
逻辑群组转换
物理元区块转换
原始区块
更新区块
(F)
循序写入
UW
CH
ER
SQ
INT
OR
(G)
循序填充
(H)
非循序写入

图 14（F）到图 14（H）

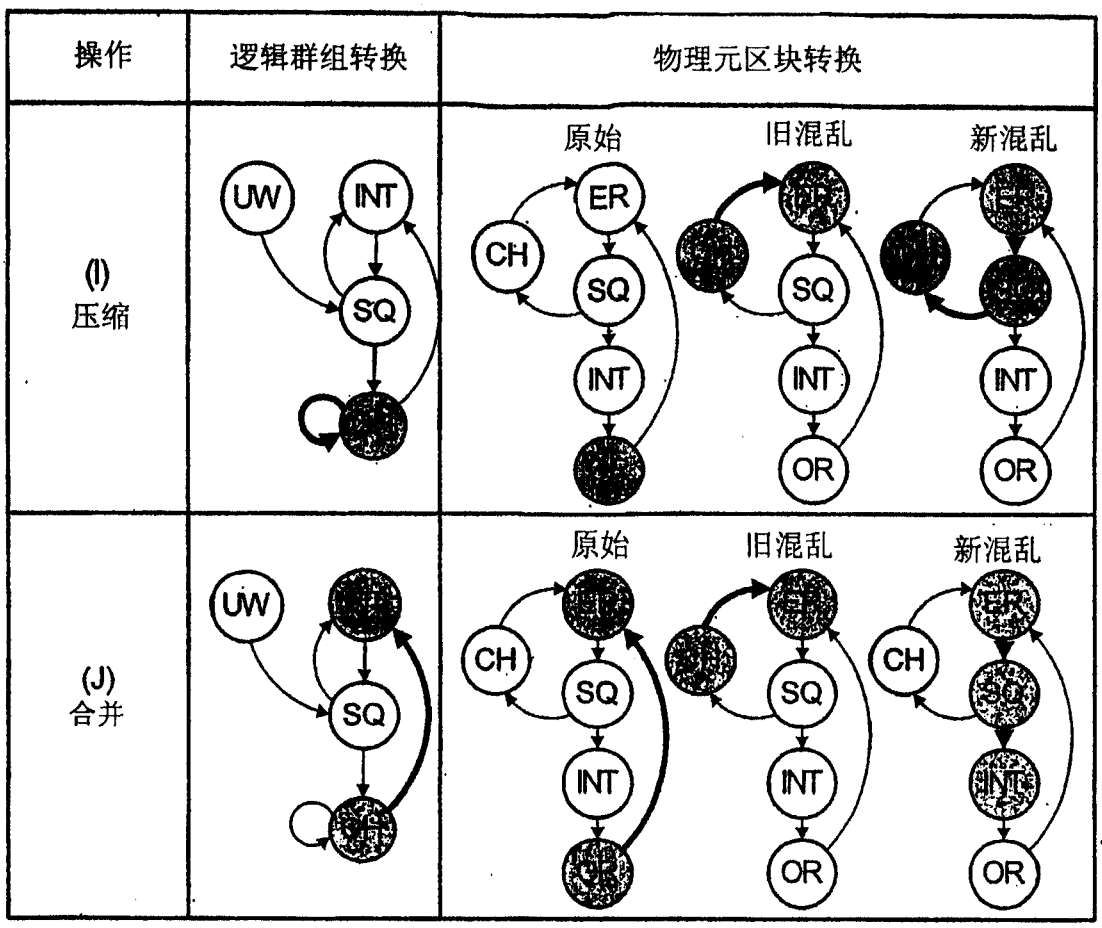


图 14（I）到图 14（J）

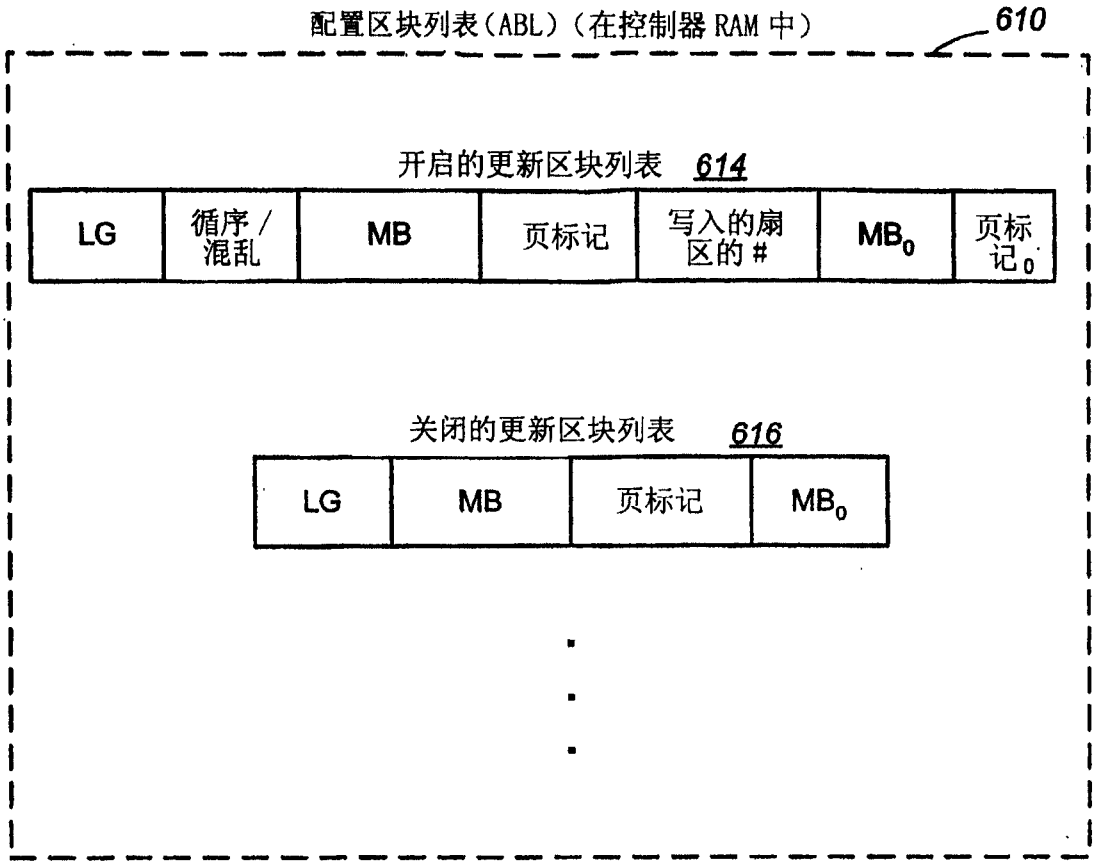

配置区块列表(ABL)(在控制器 RAM 中)
610
开启的更新区块列表 614
LG
循序 / 混乱
MB
页标记
写入的扇区的 #
MB0
页标记0
关闭的更新区块列表 616
LG
MB
页标记
MB0

图 15

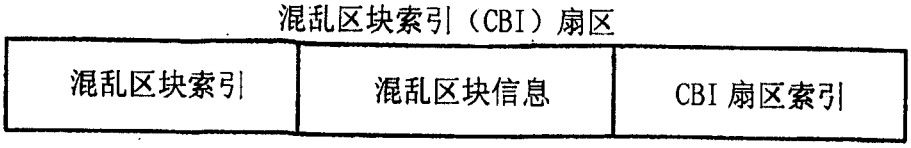

混乱区块索引（CBI）扇区
混乱区块索引
混乱区块信息
CBI 扇区索引

图 16A

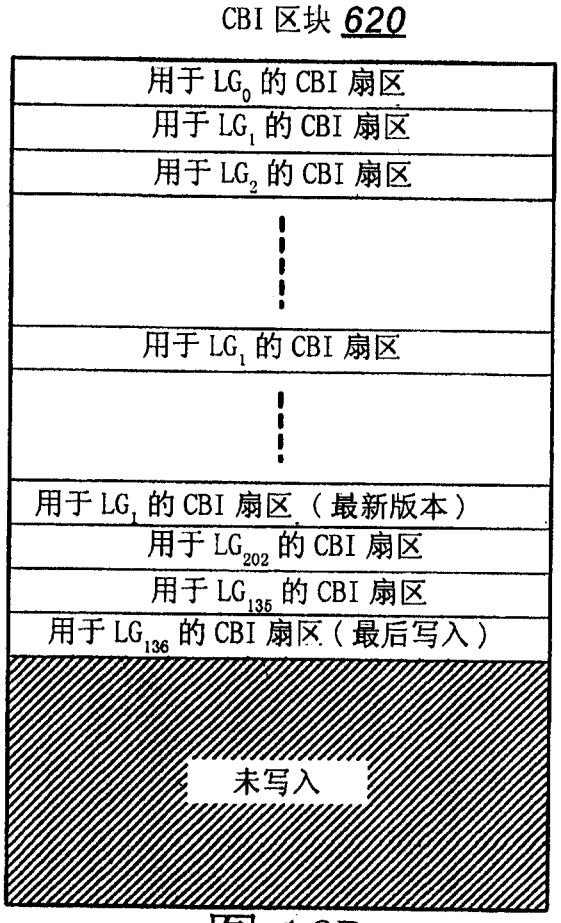

CBI 区块 620
用于 LG_0 的 CBI 扇区
用于 LG_1 的 CBI 扇区
用于 LG_2 的 CBI 扇区
用于 LG_1 的 CBI 扇区
用于 LG_1 的 CBI 扇区（最新版本）
用于 LG_{202} 的 CBI 扇区
用于 LG_{135} 的 CBI 扇区
用于 LG_{136} 的 CBI 扇区（最后写入）
未写入

图 16B

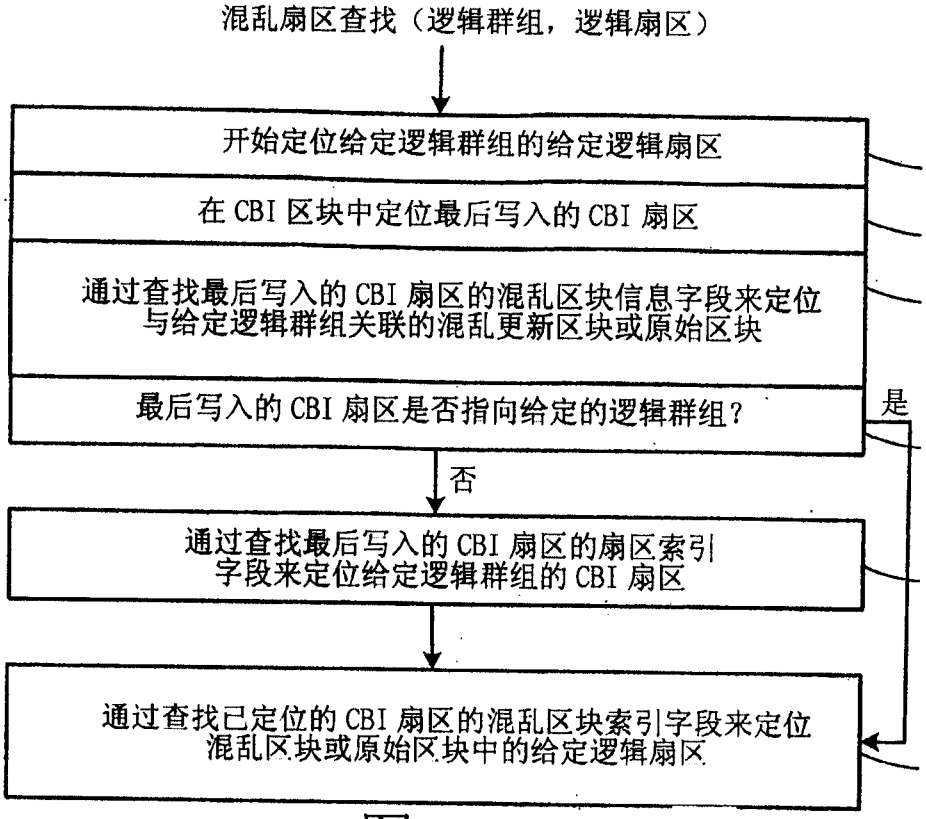

混乱扇区查找（逻辑群组，逻辑扇区）
开始定位给定逻辑群组的给定逻辑扇区
在 CBI 区块中定位最后写入的 CBI 扇区
通过查找最后写入的 CBI 扇区的混乱区块信息字段来定位与给定逻辑群组关联的混乱更新区块或原始区块
最后写入的 CBI 扇区是否指向给定的逻辑群组？
是
否
通过查找最后写入的 CBI 扇区的扇区索引字段来定位给定逻辑群组的 CBI 扇区
通过查找已定位的 CBI 扇区的混乱区块索引字段来定位混乱区块或原始区块中的给定逻辑扇区

图 16C

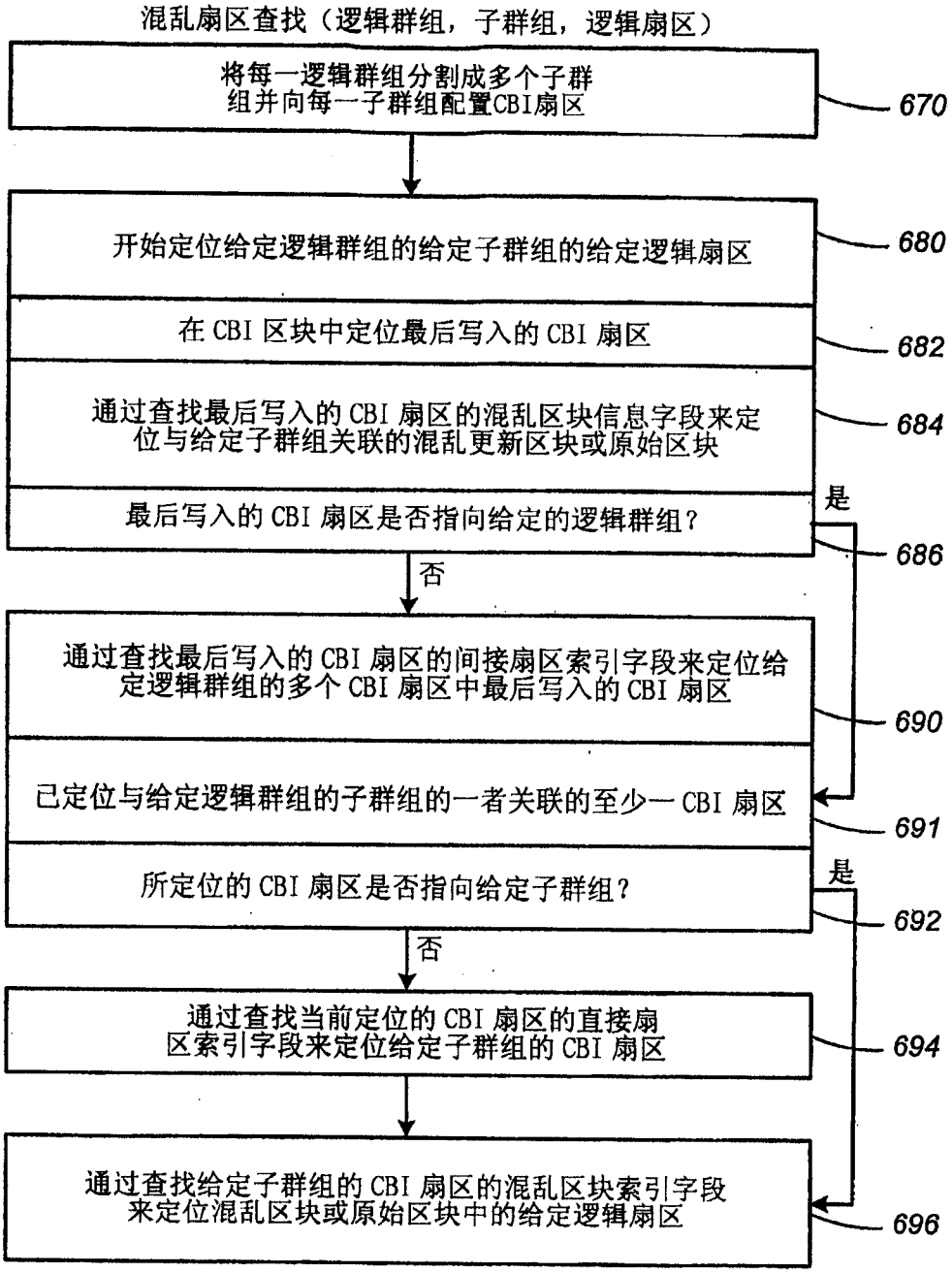

混乱扇区查找（逻辑群组，子群组，逻辑扇区）
将每一逻辑群组分割成多个子群组并向每一子群组配置CBI扇区
670
开始定位给定逻辑群组的给定子群组的给定逻辑扇区
680
在 CBI 区块中定位最后写入的 CBI 扇区
682
通过查找最后写入的 CBI 扇区的混乱区块信息字段来定位与给定子群组关联的混乱更新区块或原始区块
684
最后写入的 CBI 扇区是否指向给定的逻辑群组？
是
686
否
通过查找最后写入的 CBI 扇区的间接扇区索引字段来定位给定逻辑群组的多个 CBI 扇区中最后写入的 CBI 扇区
690
已定位与给定逻辑群组的子群组的一者关联的至少一 CBI 扇区
691
所定位的 CBI 扇区是否指向给定子群组？
是
692
否
通过查找当前定位的 CBI 扇区的直接扇区索引字段来定位给定子群组的 CBI 扇区
694
通过查找给定子群组的 CBI 扇区的混乱区块索引字段来定位混乱区块或原始区块中的给定逻辑扇区
696

图 16D

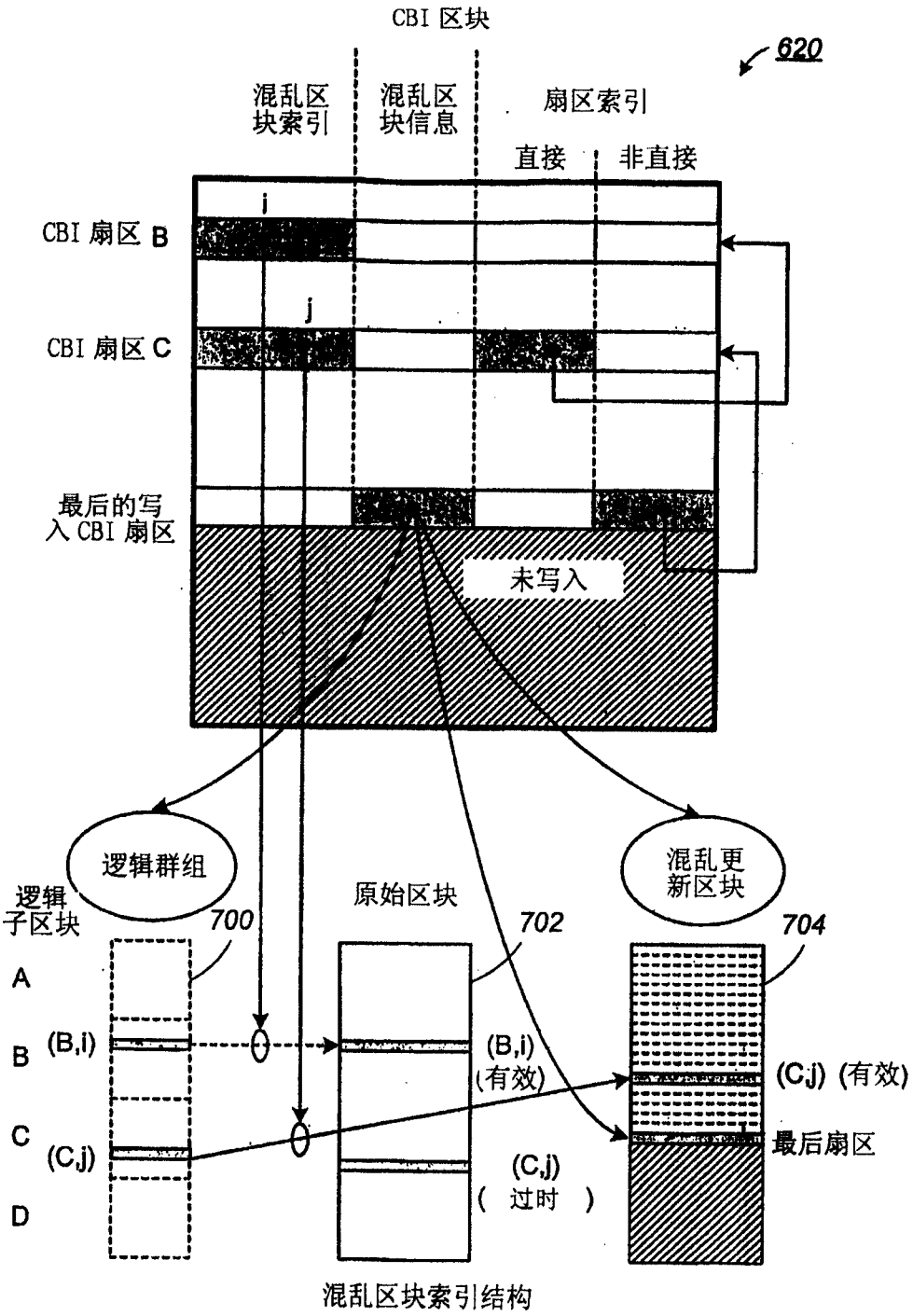

CBI 区块
620
混乱区块索引
混乱区块信息
扇区索引
直接
非直接
CBI 扇区 B
CBI 扇区 C
最后的写入 CBI 扇区
未写入
逻辑群组
逻辑子区块
700
原始区块
702
混乱更新区块
704
A
B
C
D
(B,i)
(C,j)
(B,i) (有效)
(C,j) (过时)
(C,j) (有效)
最后扇区
混乱区块索引结构

图 16E

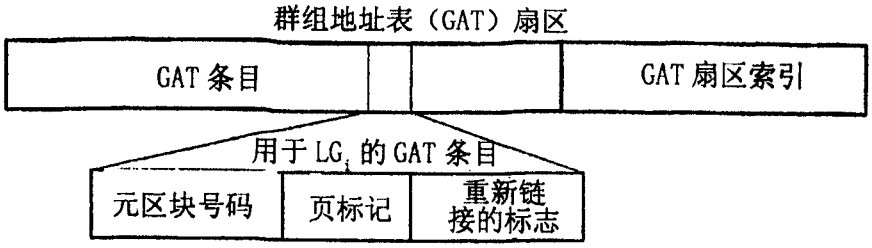

群组地址表（GAT）扇区
GAT 条目
GAT 扇区索引
用于 LG_i 的 GAT 条目
元区块号码
页标记
重新链接的标志

图 17A

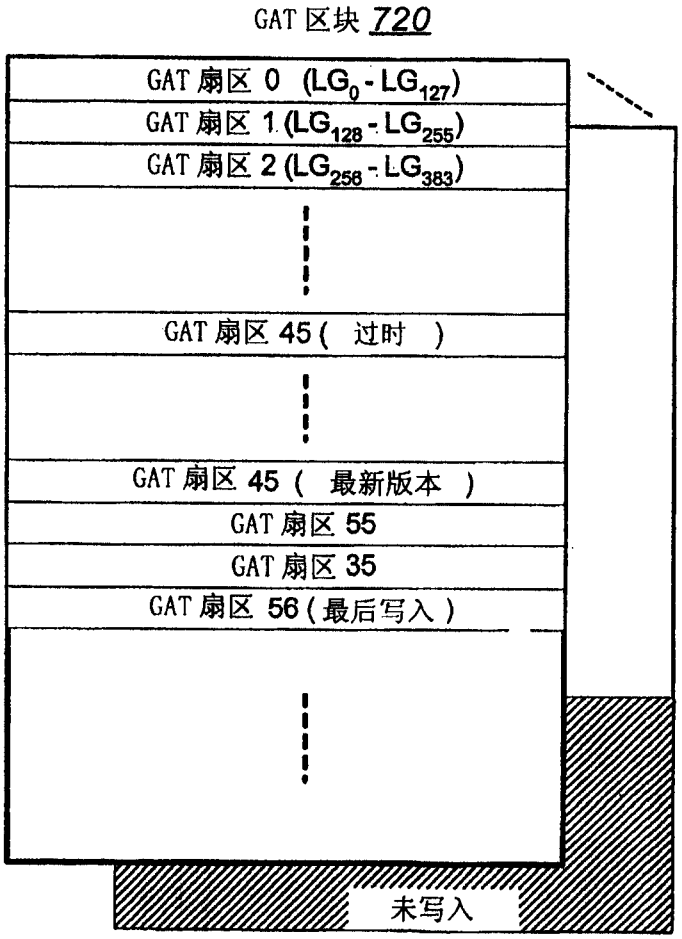

GAT 区块 720
GAT 扇区 0 (LG_0 - LG_{127})
GAT 扇区 1 (LG_{128} - LG_{255})
GAT 扇区 2 (LG_{256} - LG_{383})
GAT 扇区 45 (过时)
GAT 扇区 45 (最新版本)
GAT 扇区 55
GAT 扇区 35
GAT 扇区 56 (最后写入)
未写入

图 17B

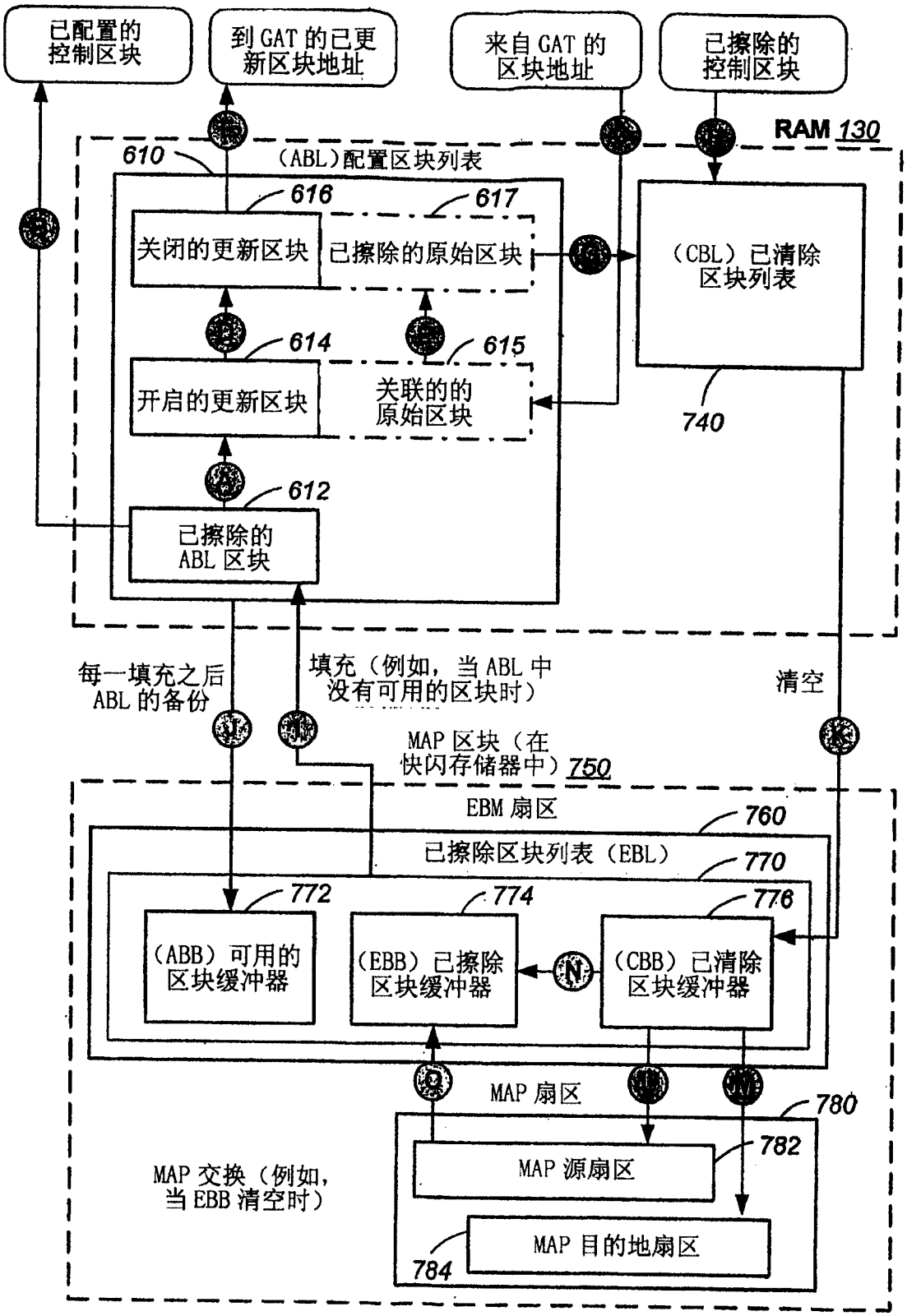
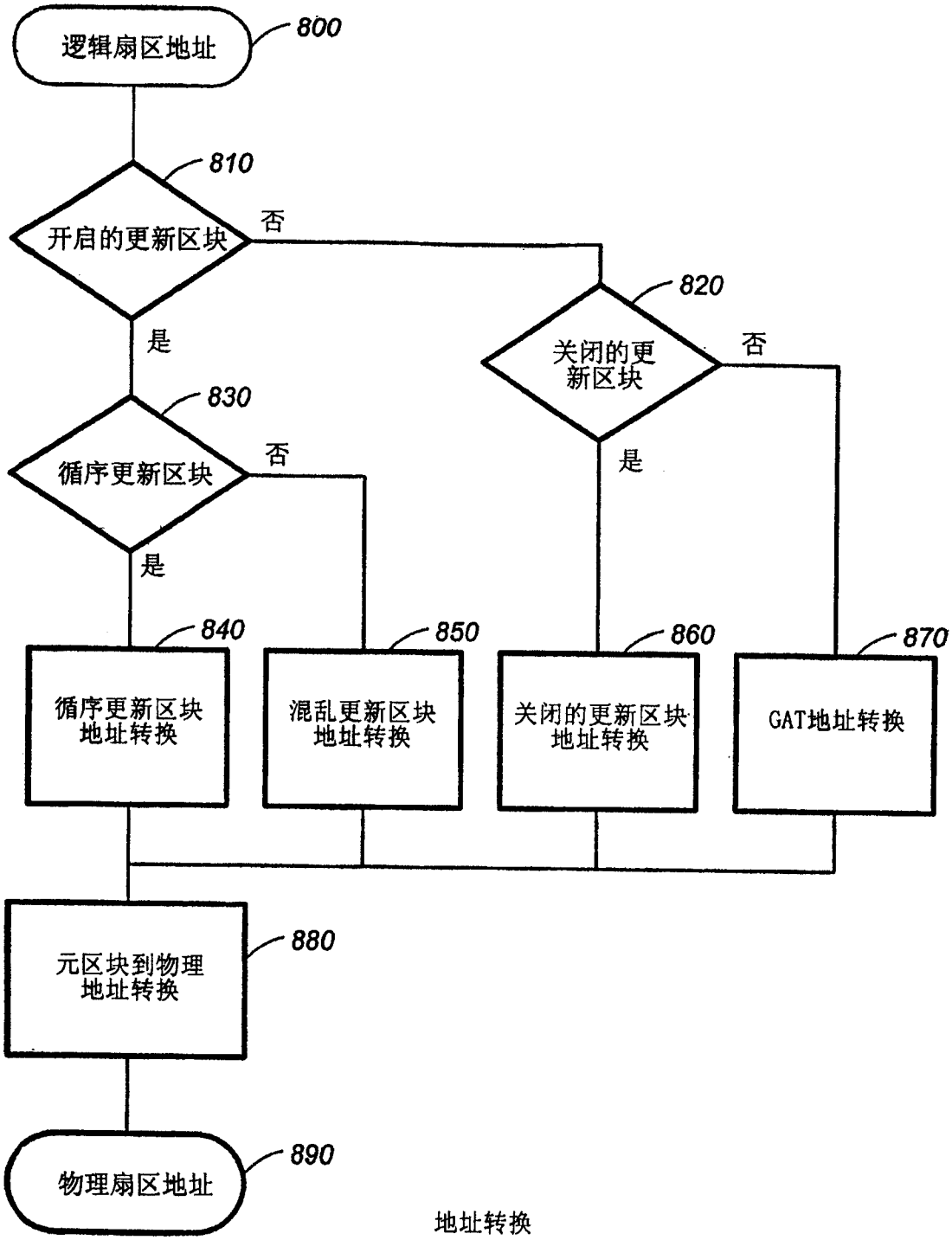
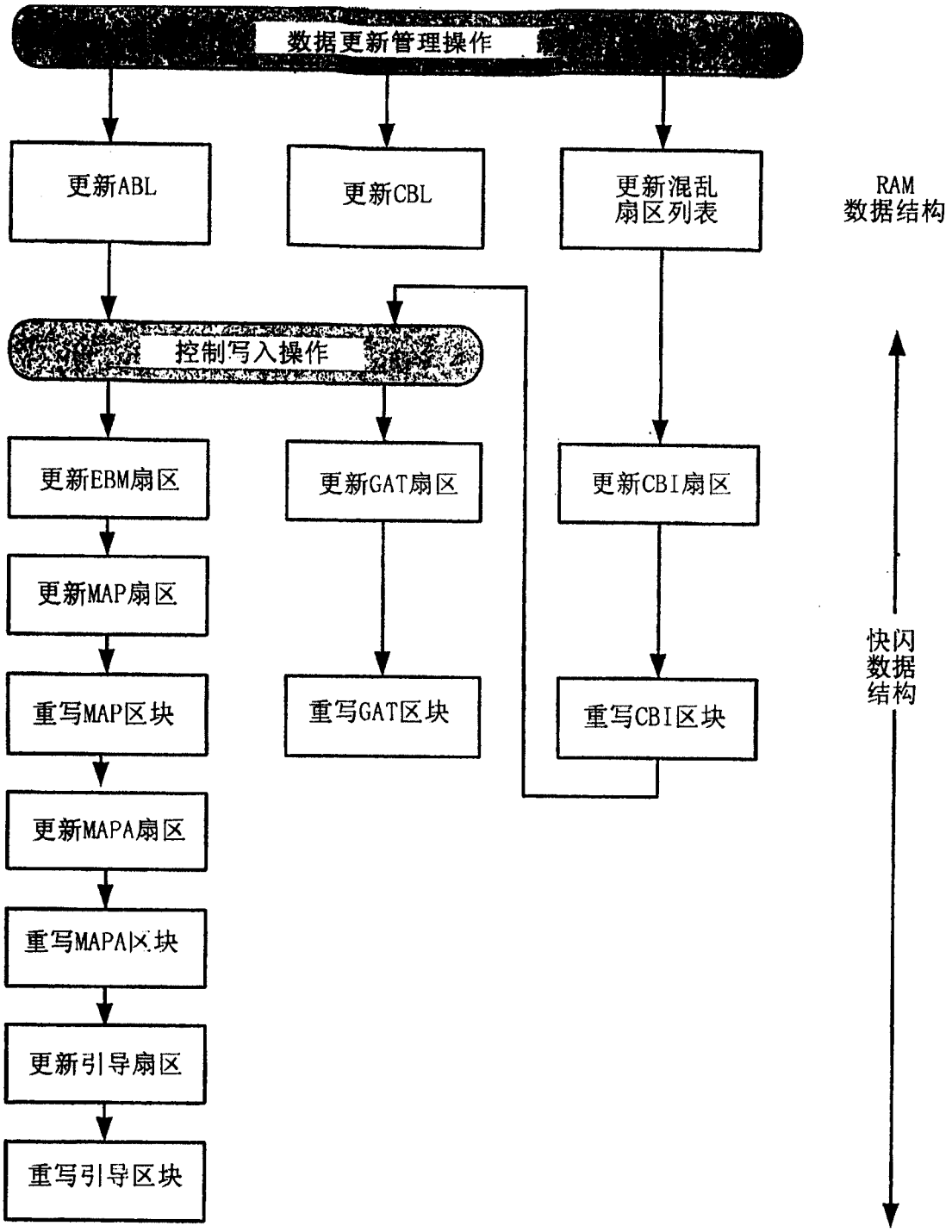

已配置的控制区块
到 GAT 的已更新区块地址
来自 GAT 的区块地址
已擦除的控制区块
RAM 130
610
(ABL)配置区块列表
616
617
关闭的更新区块
已擦除的原始区块
(CBL) 已清除区块列表
614
615
开启的更新区块
关联的的原始区块
740
612
已擦除的 ABL 区块
每一填充之后 ABL 的备份
填充（例如，当 ABL 中没有可用的区块时）
清空
MAP 区块（在快闪存储器中）750
EBM 扇区
760
已擦除区块列表 (EBL)
770
772
774
776
(ABB) 可用的区块缓冲器
(EBB) 已擦除区块缓冲器
(CBB) 已清除区块缓冲器
MAP 扇区
780
782
MAP 源扇区
MAP 交换（例如，当 EBB 清空时）
MAP 目的地扇区
784

图 18


逻辑扇区地址
800
开启的更新区块
810
否
是
关闭的更新区块
820
否
是
循序更新区块
830
否
是
循序更新区块地址转换
840
混乱更新区块地址转换
850
关闭的更新区块地址转换
860
GAT地址转换
870
元区块到物理地址转换
880
物理扇区地址
890

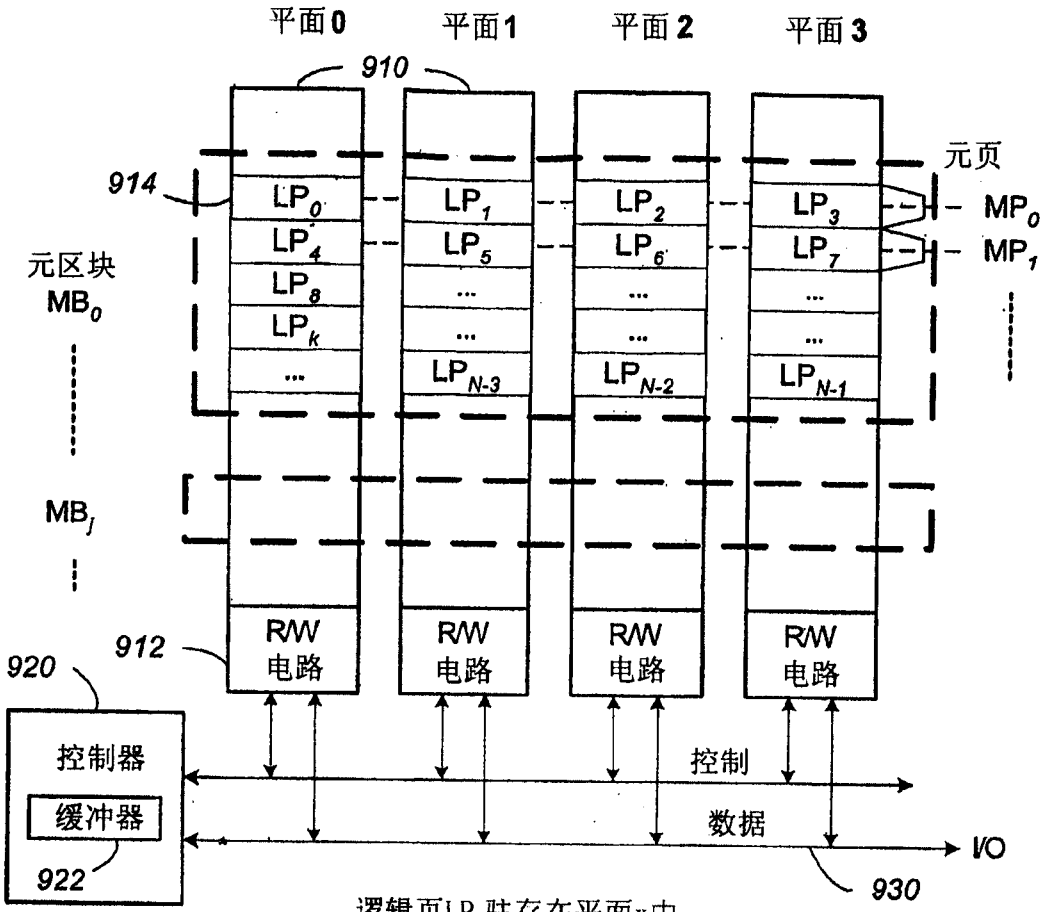
地址转换

图 19



对于控制数据结构的操作

图20



逻辑页LP_k驻存在平面x中，
其中x= k MOD（k，W），
W（例如，=4）是平面的总数目。

图21

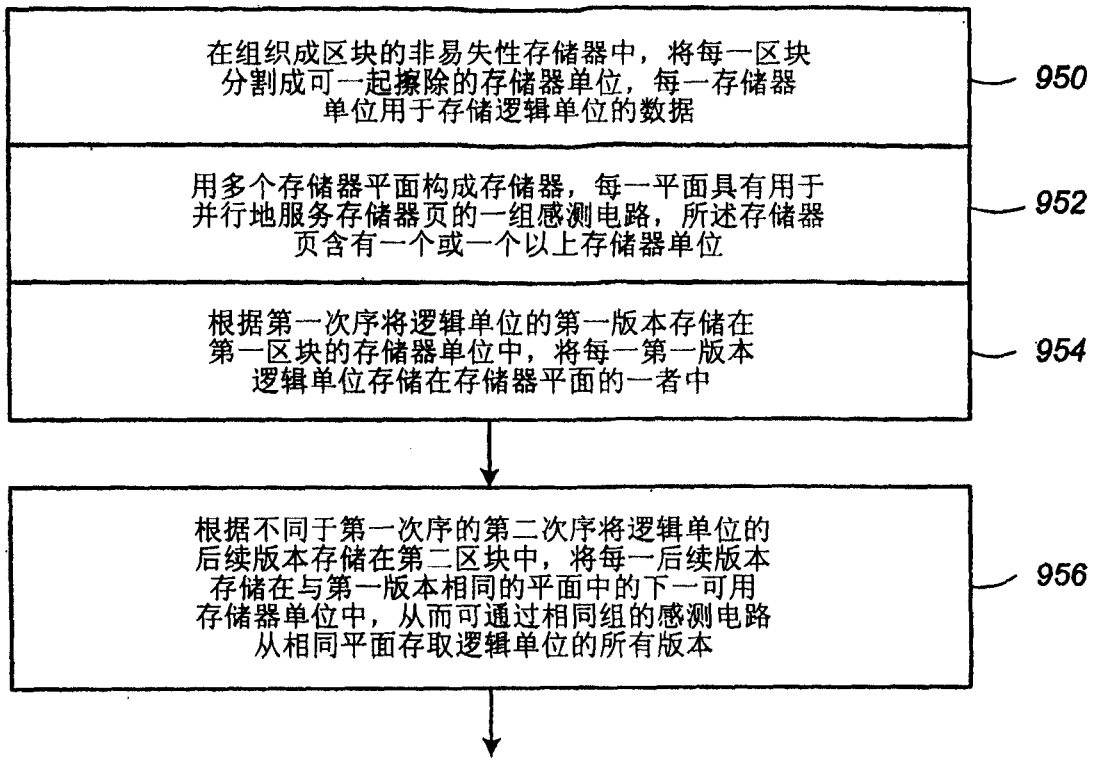

在组织成区块的非易失性存储器中，将每一区块分割成可一起擦除的存储器单位，每一存储器单位用于存储逻辑单位的数据
950
用多个存储器平面构成存储器，每一平面具有用于并行地服务存储器页的一组感测电路，所述存储器页含有一个或一个以上存储器单位
952
根据第一次序将逻辑单位的第一版本存储在第一区块的存储器单位中，将每一第一版本逻辑单位存储在存储器平面的一者中
954
根据不同于第一次序的第二次序将逻辑单位的后续版本存储在第二区块中，将每一后续版本存储在与第一版本相同的平面中的下一可用存储器单位中，从而可通过相同组的感测电路从相同平面存取逻辑单位的所有版本
956

图22A

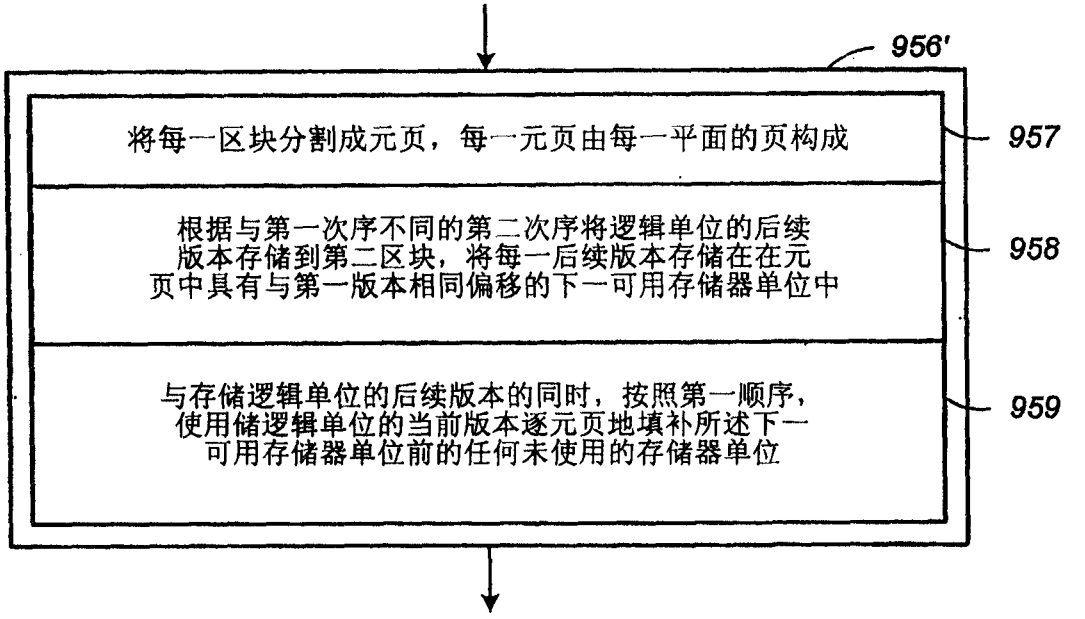

956'
将每一区块分割成元页，每一元页由每一平面的页构成
957
根据与第一次序不同的第二次序将逻辑单位的后续版本存储到第二区块，将每一后续版本存储在在元页中具有与第一版本相同偏移的下一可用存储器单位中
958
与存储逻辑单位的后续版本的同时，按照第一顺序，使用储逻辑单位的当前版本逐元页地填补所述下一可用存储器单位前的任何未使用的存储器单位
959

图22B

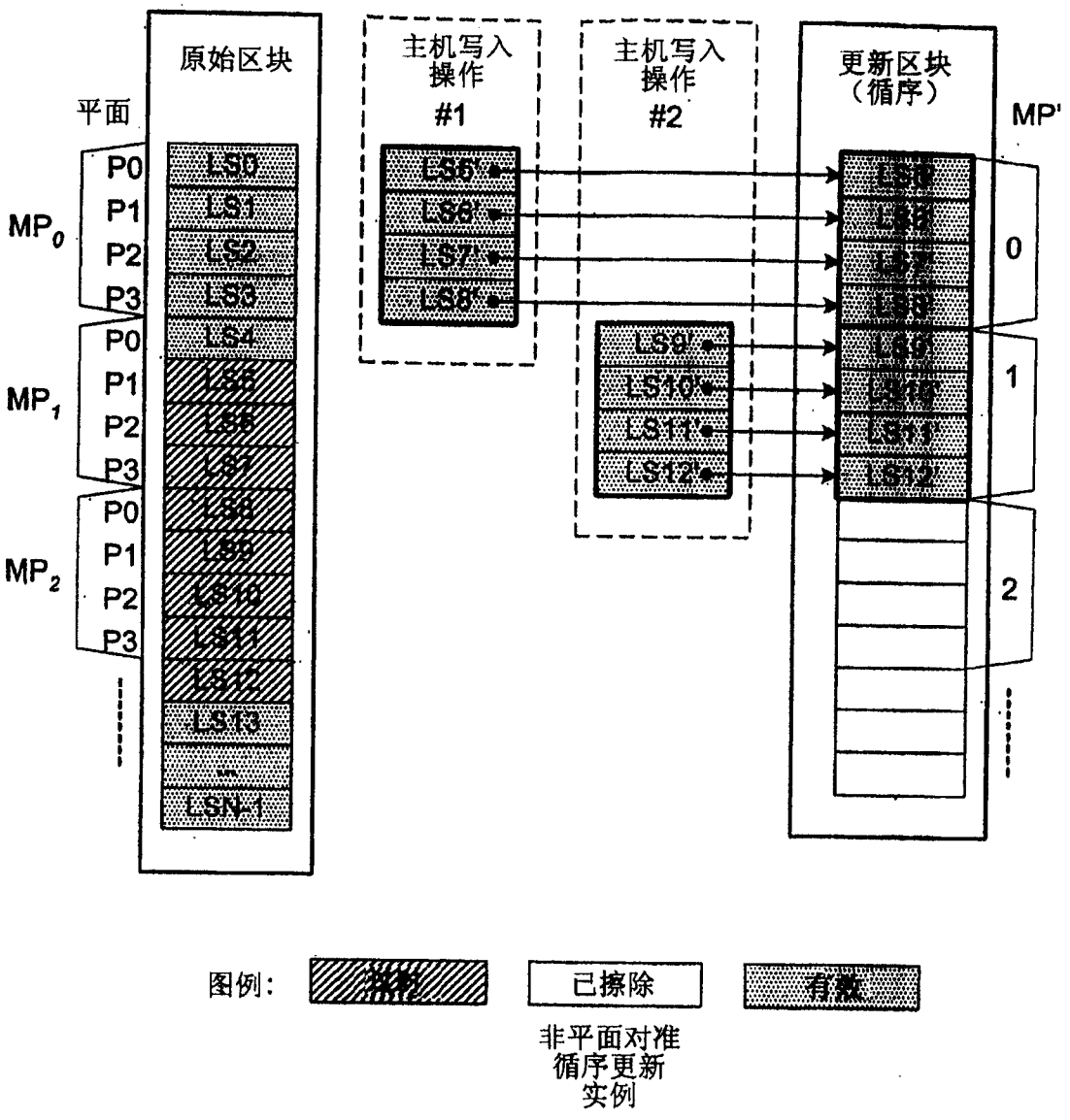

原始区块
平面
主机写入操作 #1
主机写入操作 #2
更新区块（循序）
MP'
MP0
MP1
MP2
P0
P1
P2
P3
LS0
LS1
LS2
LS3
LS4
LS5
LS6
LS7
LS8
LS9
LS10
LS11
LS12
LS13
LSN-1
LS6'
LS6''
LS7'
LS8'
LS9'
LS10'
LS11'
LS12'
0
1
2
图例：
已擦除
有效
非平面对准循序更新实例

图23A

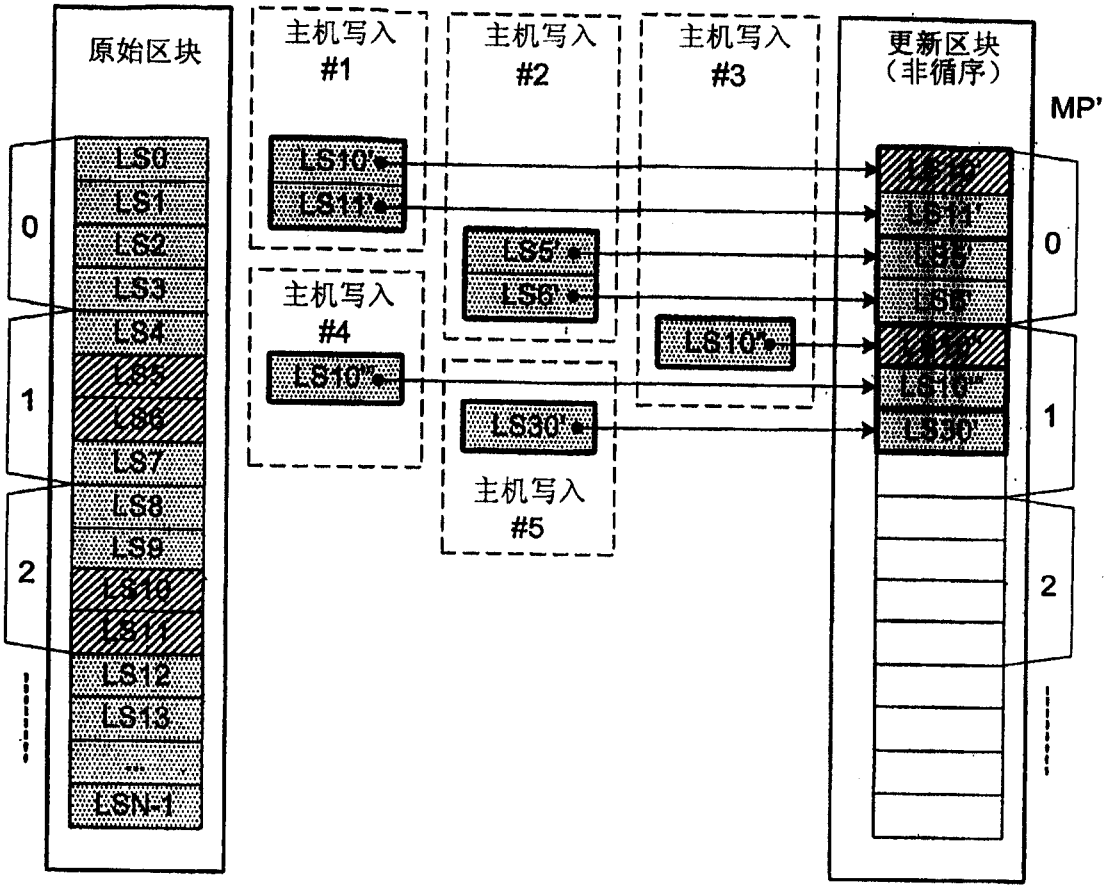

原始区块
LS0
LS1
LS2
LS3
LS4
LS5
LS6
LS7
LS8
LS9
LS10
LS11
LS12
LS13
LSN-1
0
1
2
主机写入
#1
LS10'
LS11'
主机写入
#2
LS5'
LS6'
主机写入
#3
LS10"
主机写入
#4
LS10"
主机写入
#5
LS30'
更新区块
（非循序）
MP'
0
1
2
LS30
图例：
已擦除
有效
非平面对准
混乱更新
（非循序）
实例

图23B

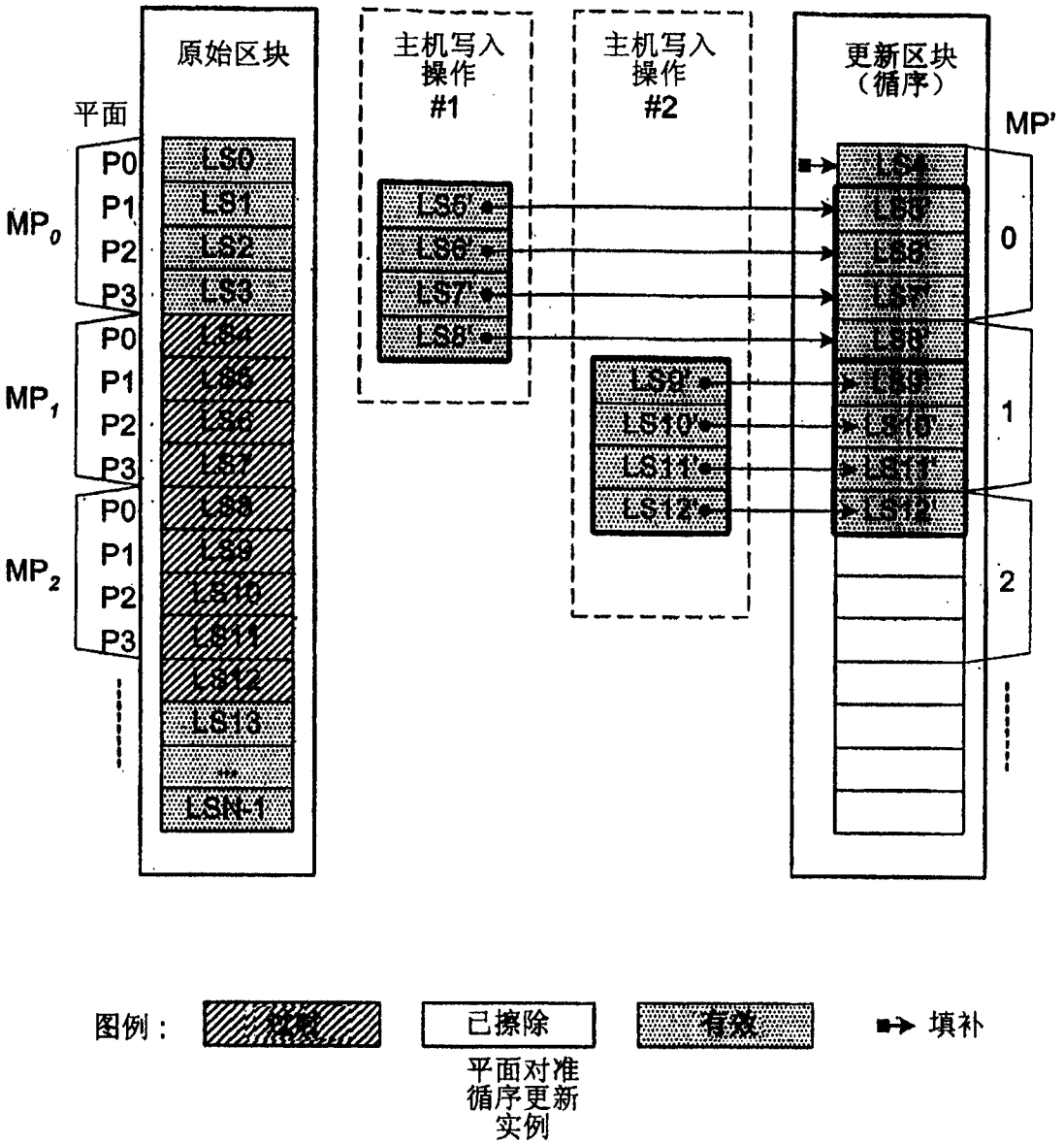

原始区块
主机写入操作 #1
主机写入操作 #2
更新区块（循序）
平面
MP0
MP1
MP2
MP'
P0
P1
P2
P3
LS0
LS1
LS2
LS3
LS4
LS5
LS6
LS7
LS8
LS9
LS10
LS11
LS12
LS13
...
LSN-1
LS6'
LS7'
LS8'
LS9'
LS10'
LS11'
LS12'
0
1
2
图例：
已擦除
有效
填补
平面对准循序更新实例

图24A

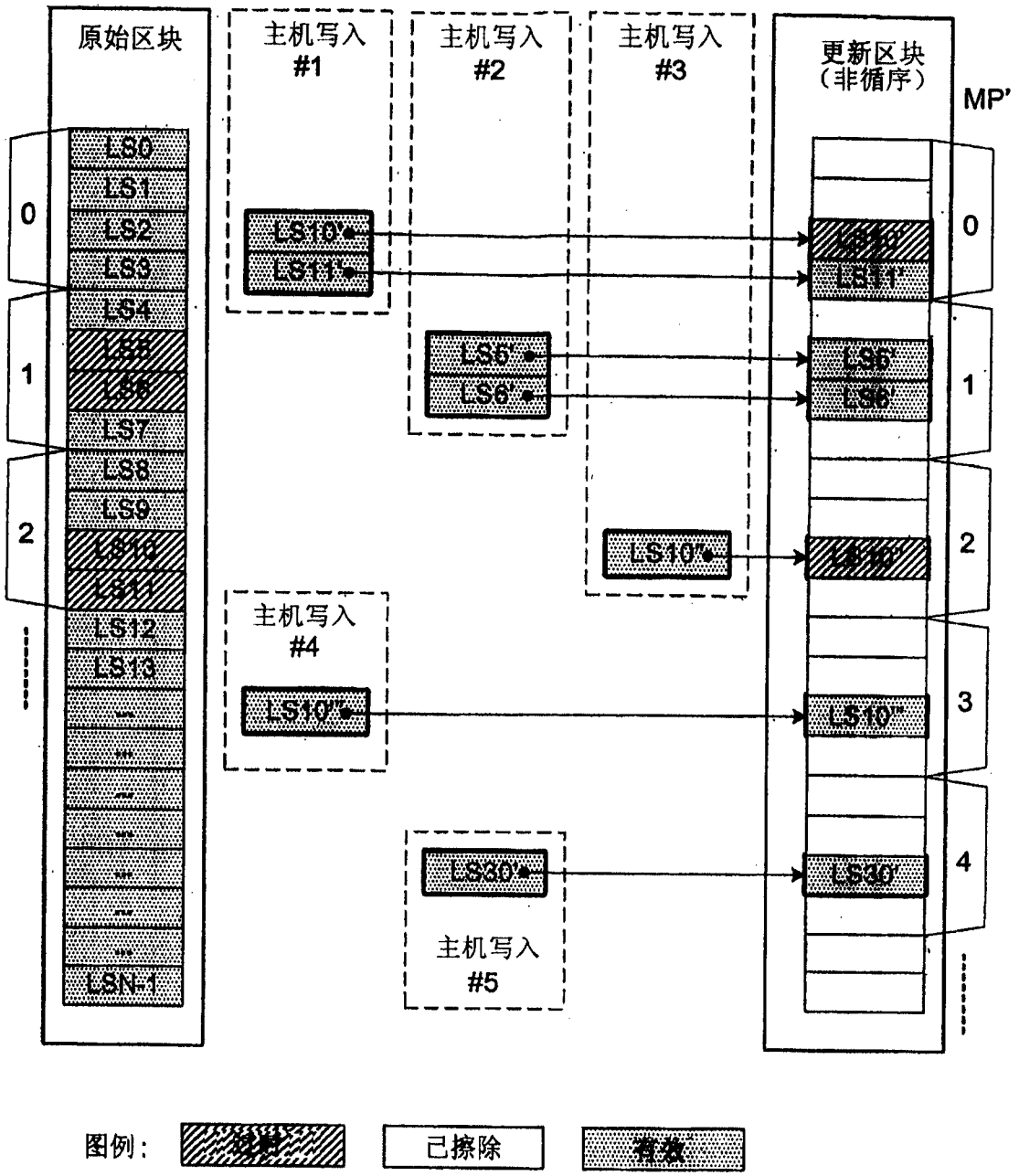
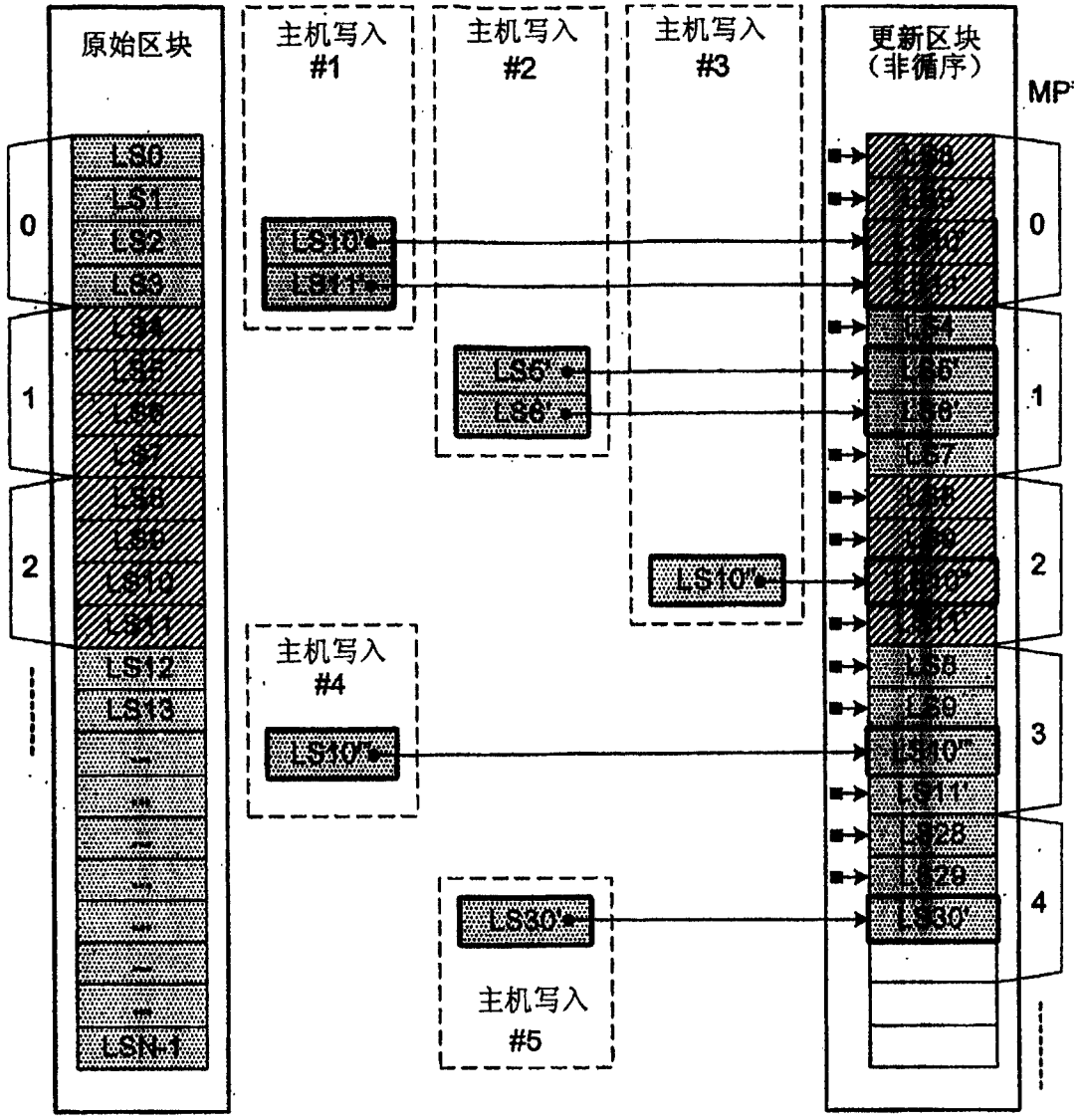
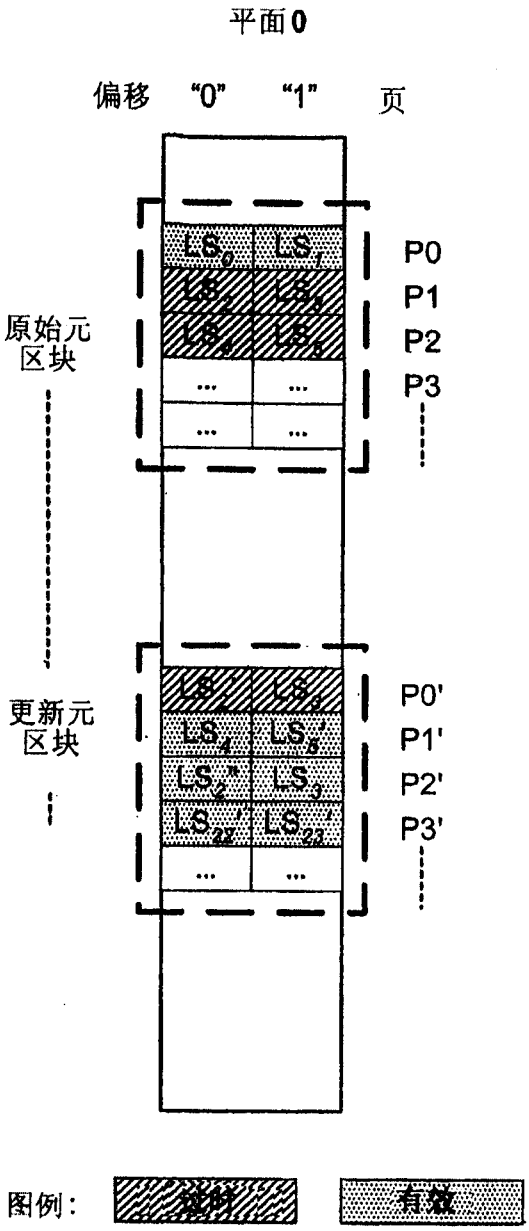

原始区块
主机写入 #1
主机写入 #2
主机写入 #3
更新区块（非循序）
MP'
0
1
2
LS0
LS1
LS2
LS3
LS4
LS5
LS6
LS7
LS8
LS9
LS10
LS11
LS12
LS13
LSN-1
LS10'
LS11'
LS6'
LS6'
LS10'
主机写入 #4
LS10''
主机写入 #5
LS30'
LS10'
LS11'
LS6'
LS6'
LS10''
LS30'
3
4
图例：
已擦除
有效
平面对准
混乱更新
（非循序）
未填补实例

图24B



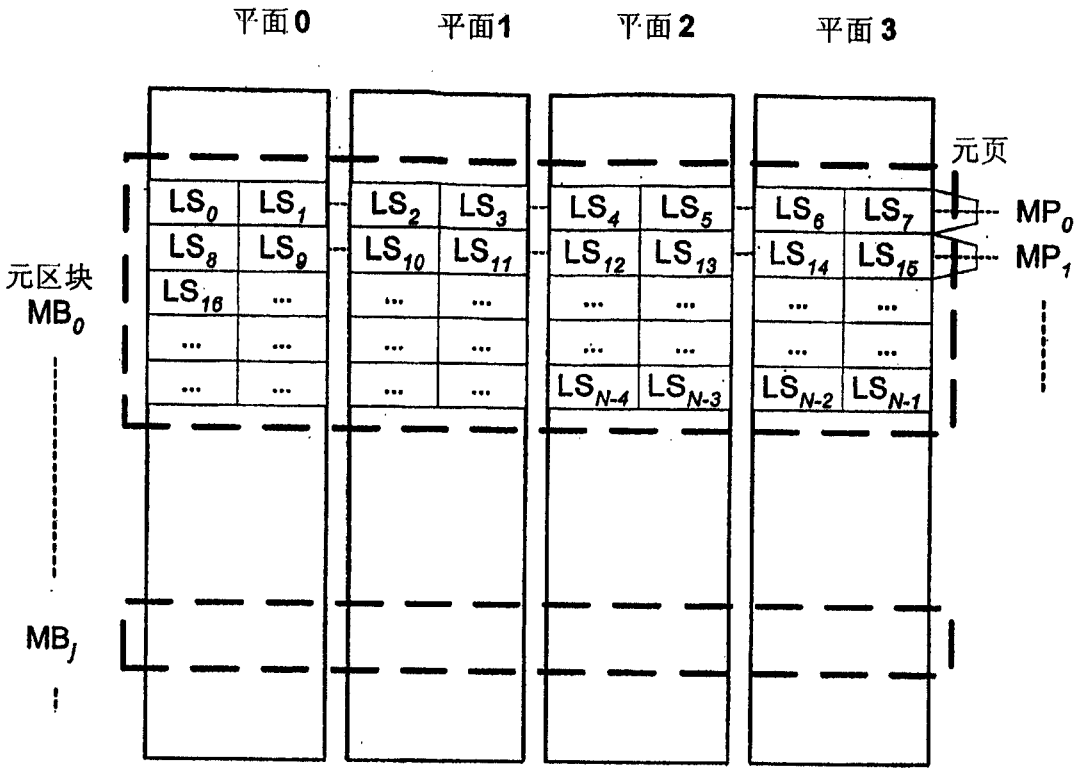
平面对准
混乱更新
（非循序）
填补实例

图24C



页内的扇区对准实例

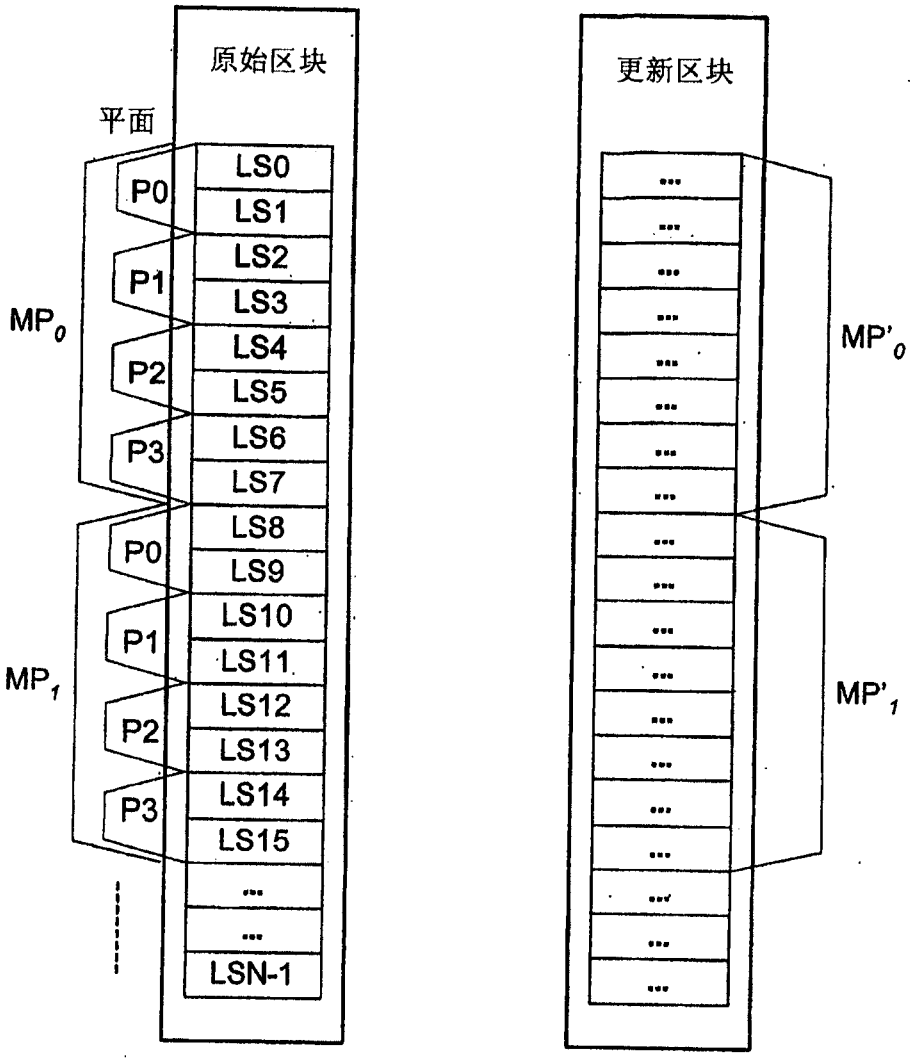
图25



LSk驻存在平面x中
其中x=k' MOD (W),
k' =INT (k/K),
K (例如, =2) 是平面的页中扇区的总数,
W (例如, =4) 是平面的总数目。

多扇区页平面对准

图26A



多扇区页平面对准

图26B

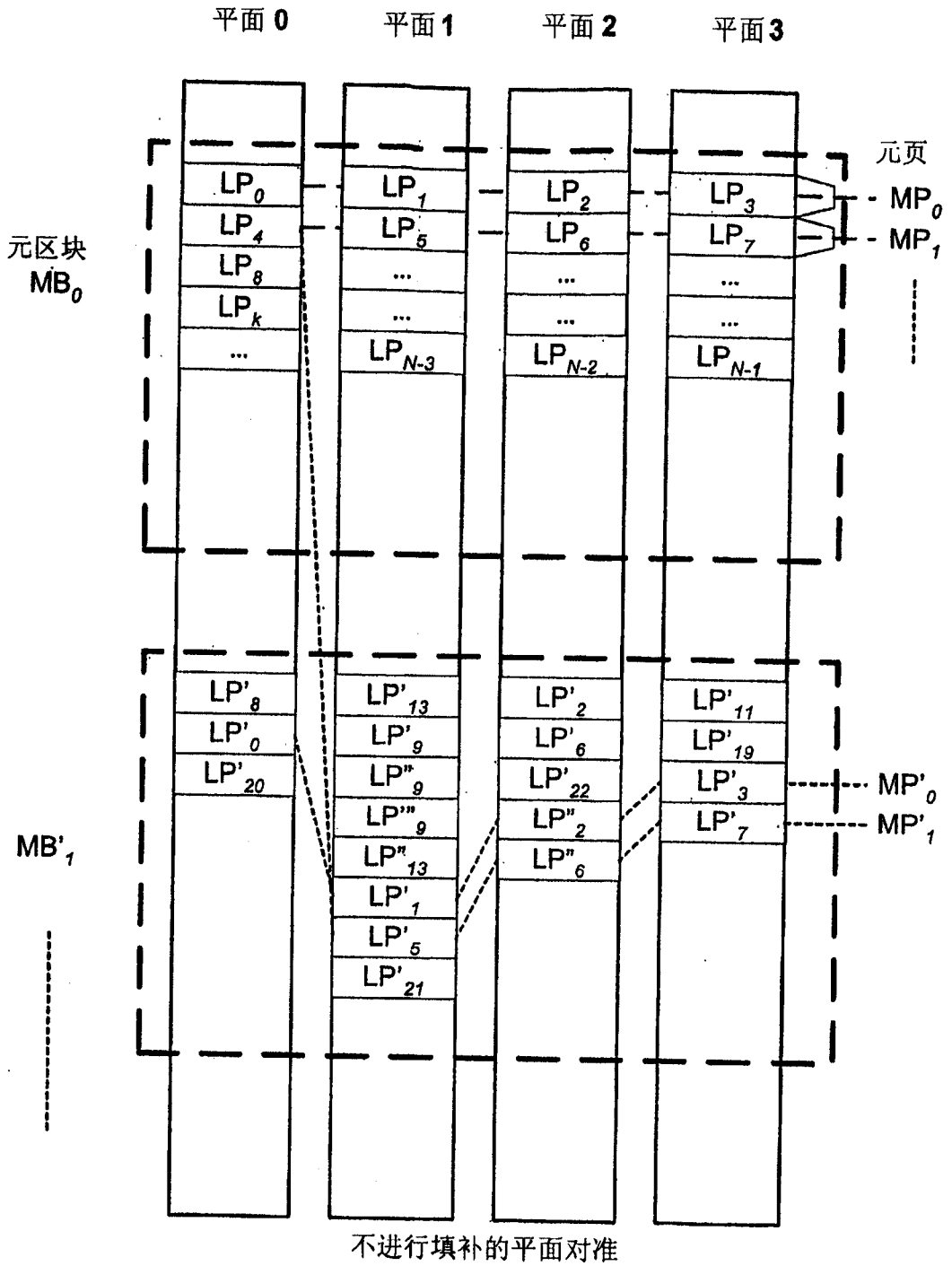
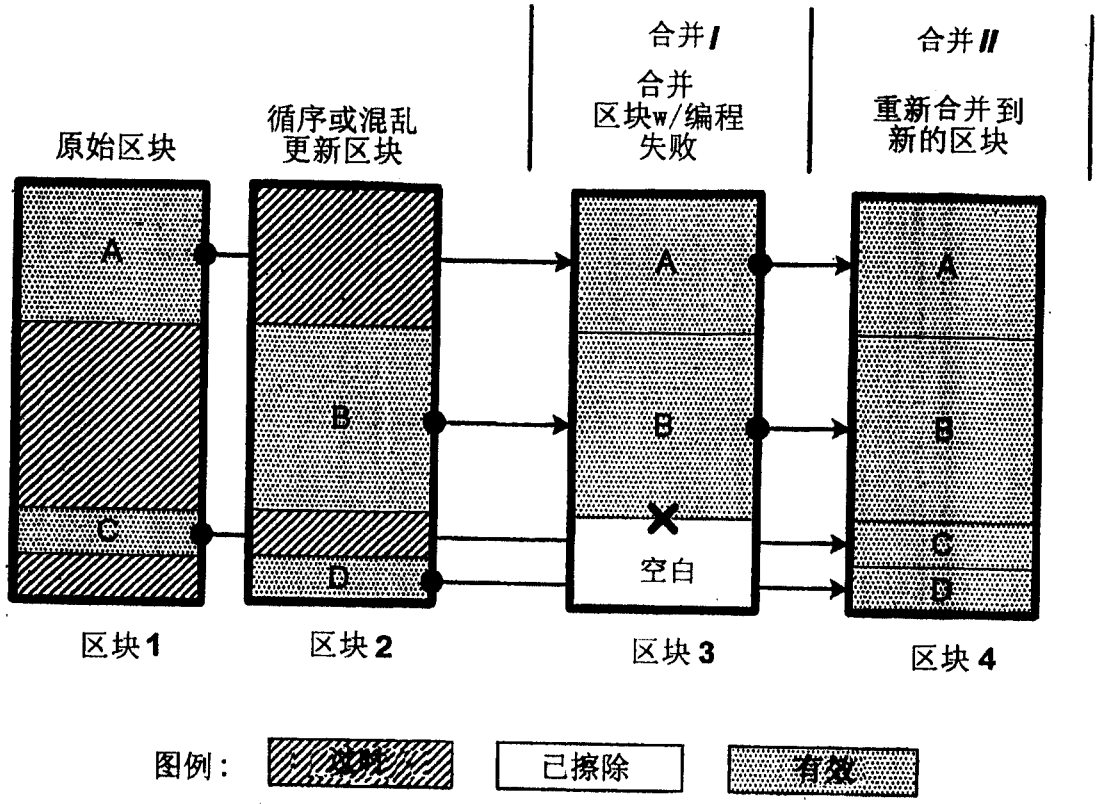
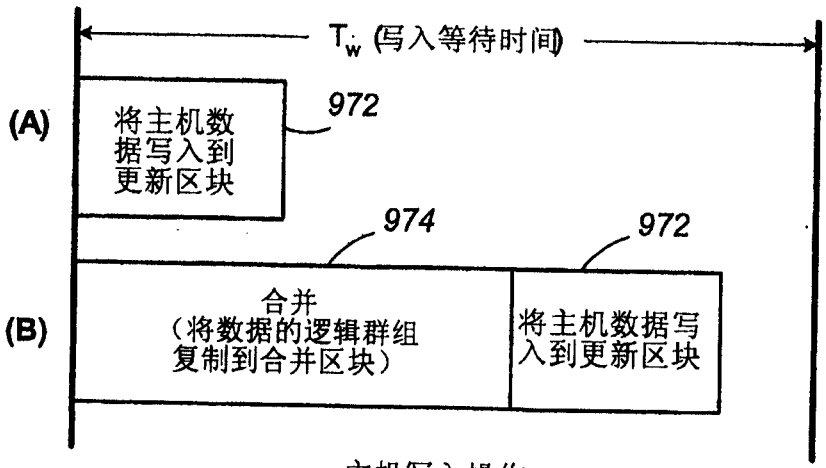

平面 0
平面 1
平面 2
平面 3
元页
元区块
MB0
LP0
LP4
LP8
LPk
LP1
LP5
LP2
LP6
LP3
LP7
MP0
MP1
LPN-3
LPN-2
LPN-1
MB'1
LP'8
LP'0
LP'20
LP'13
LP'9
LP"9
LP'"9
LP"13
LP'1
LP'5
LP'21
LP'2
LP'6
LP'22
LP"2
LP"6
LP'11
LP'19
LP'3
LP'7
MP'0
MP'1
不进行填补的平面对准

图27



通过重新合并来处理合并失败

图28



主机写入操作

图29

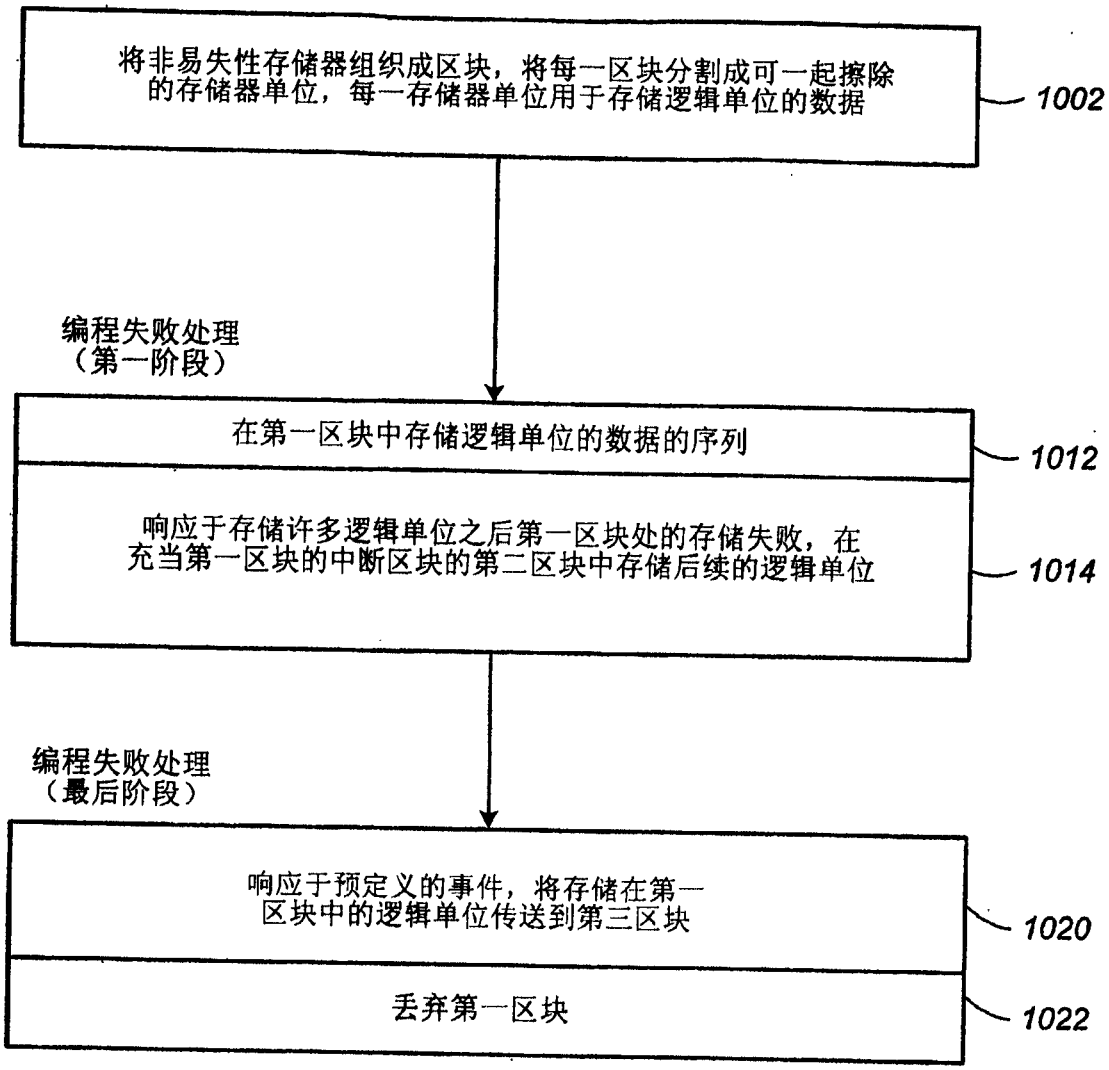
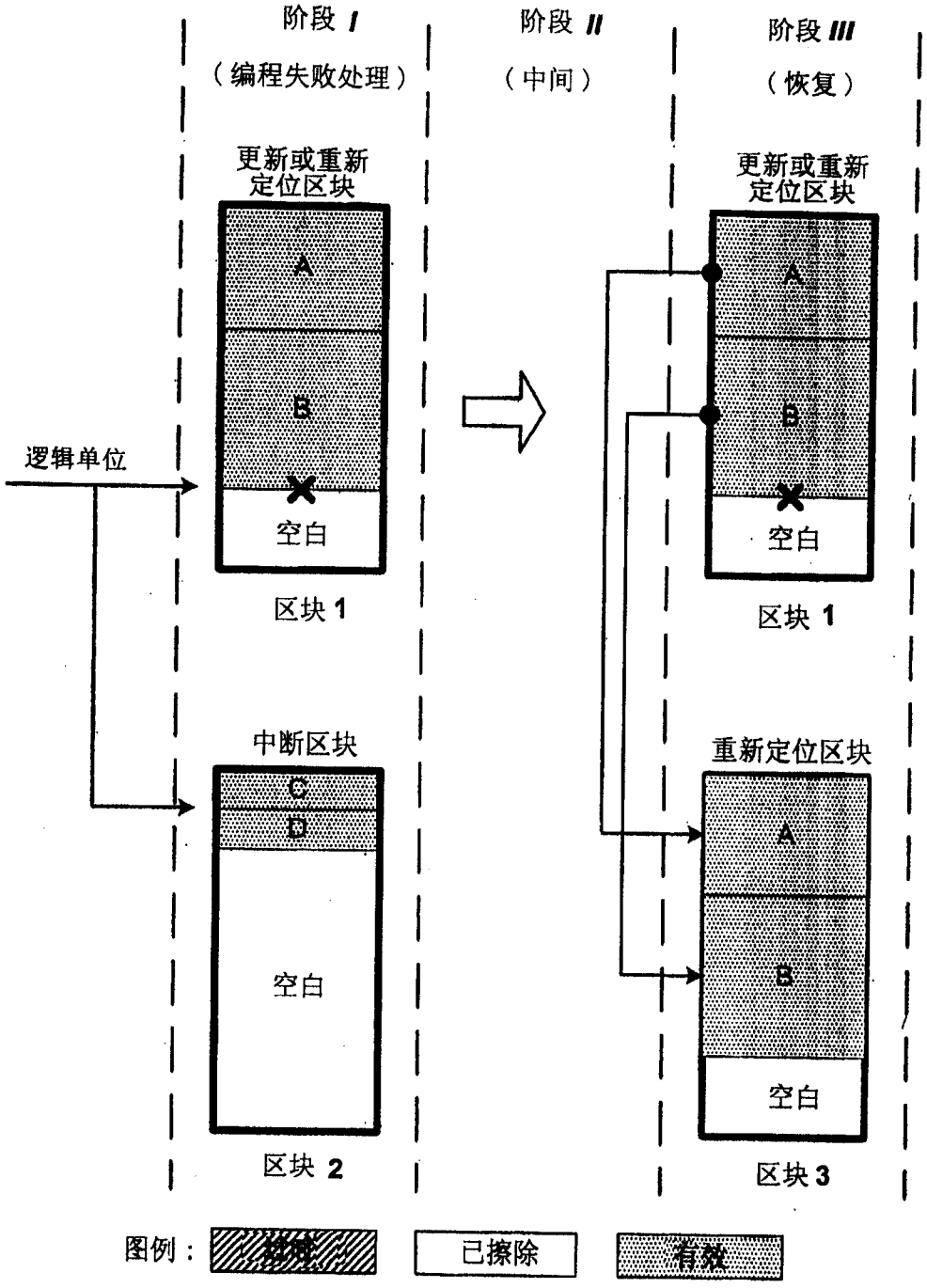
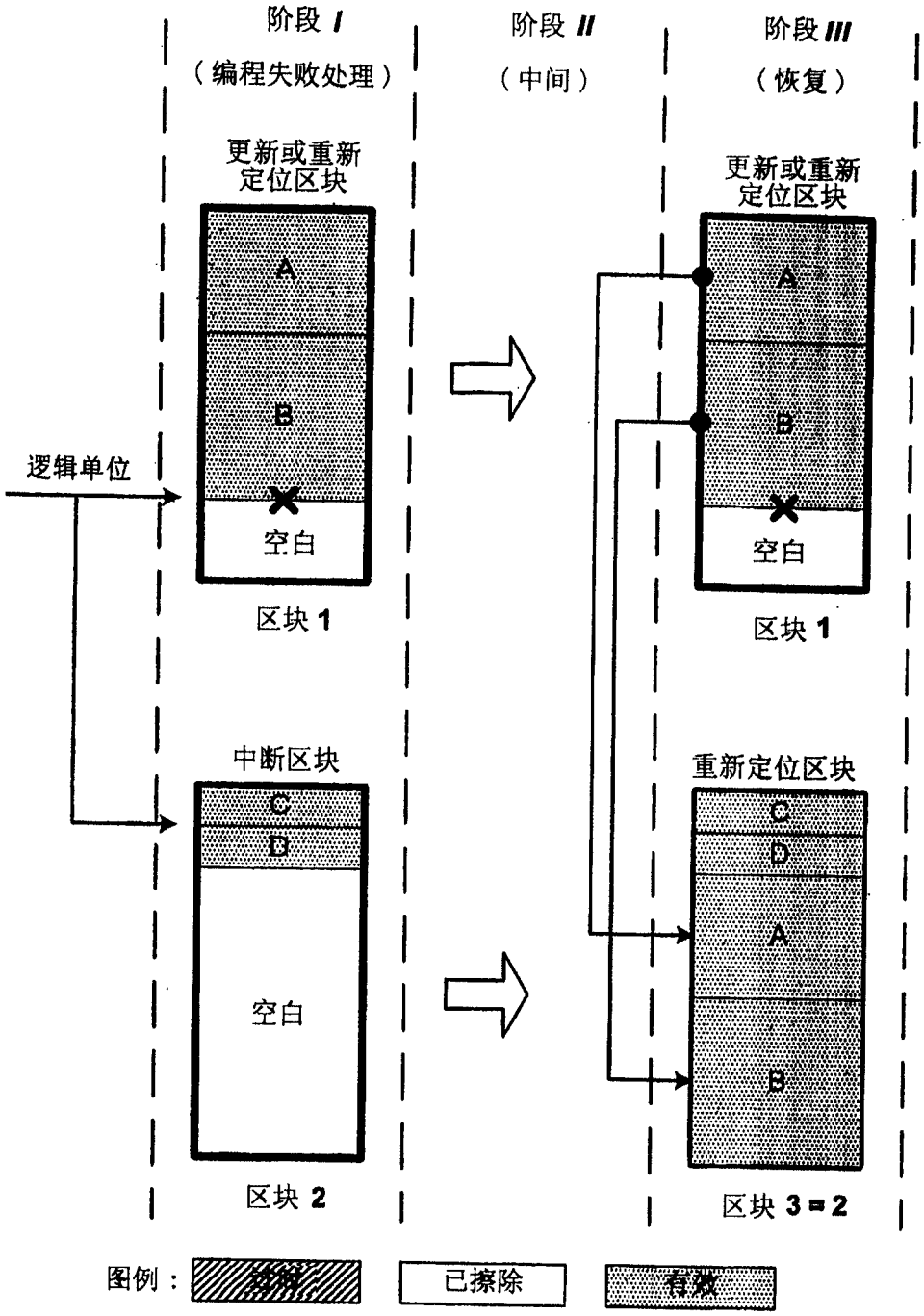

将非易失性存储器组织成区块，将每一区块分割成可一起擦除的存储器单位，每一存储器单位用于存储逻辑单位的数据
1002
编程失败处理（第一阶段）
在第一区块中存储逻辑单位的数据的序列
1012
响应于存储许多逻辑单位之后第一区块处的存储失败，在充当第一区块的中断区块的第二区块中存储后续的逻辑单位
1014
编程失败处理（最后阶段）
响应于预定义的事件，将存储在第一区块中的逻辑单位传送到第三区块
1020
丢弃第一区块
1022

图30



编程失败处理
（情况 1：区块 3= 新的）

图 31A



编程失败处理
（情况 2：区块 3= 区块 2）

图 31B

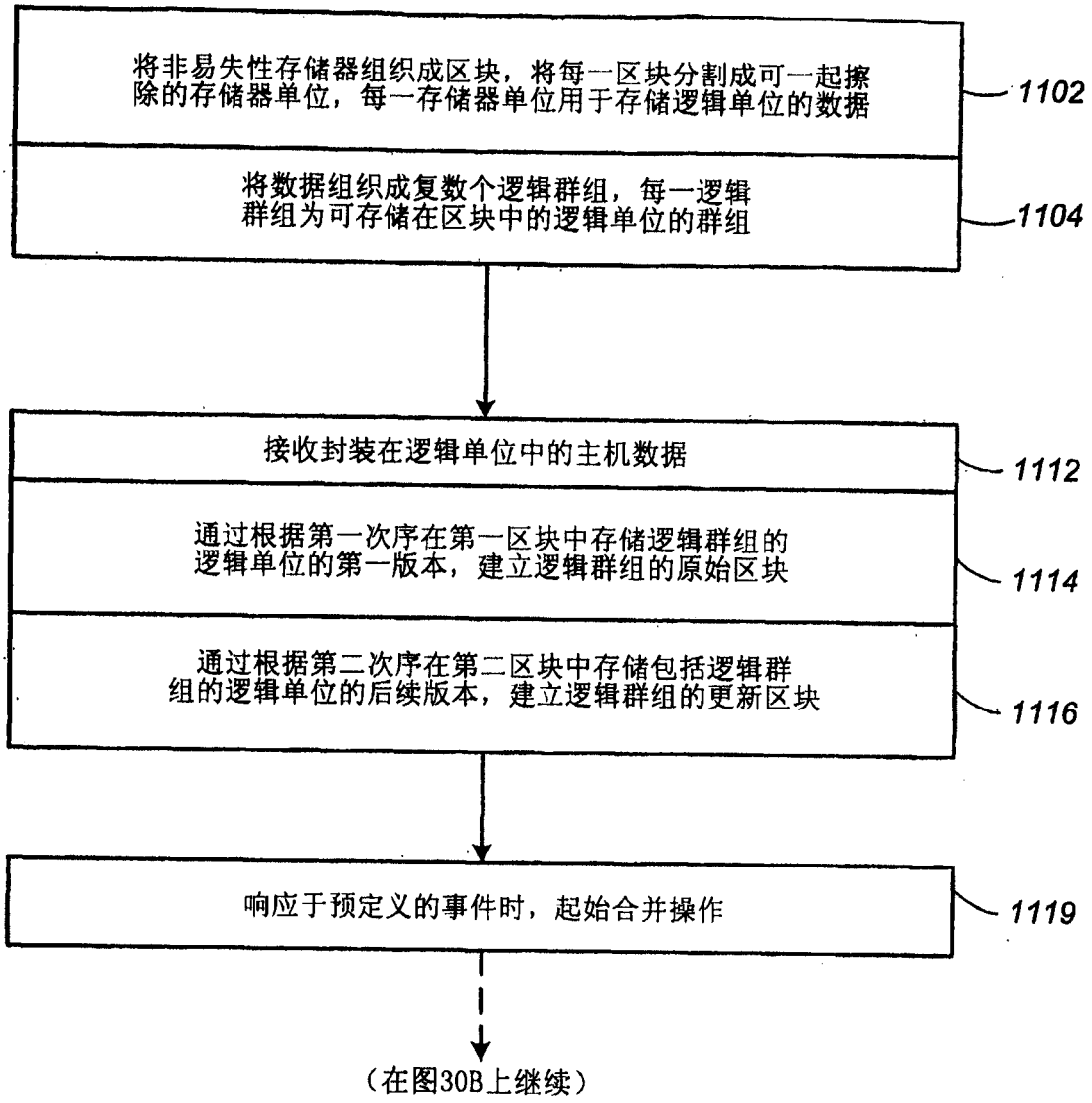

将非易失性存储器组织成区块，将每一区块分割成可一起擦除的存储器单位，每一存储器单位用于存储逻辑单位的数据
1102
将数据组织成复数个逻辑群组，每一逻辑群组为可存储在区块中的逻辑单位的群组
1104
接收封装在逻辑单位中的主机数据
1112
通过根据第一次序在第一区块中存储逻辑群组的逻辑单位的第一版本，建立逻辑群组的原始区块
1114
通过根据第二次序在第二区块中存储包括逻辑群组的逻辑单位的后续版本，建立逻辑群组的更新区块
1116
响应于预定义的事件时，起始合并操作
1119
（在图30B上继续）

图32A

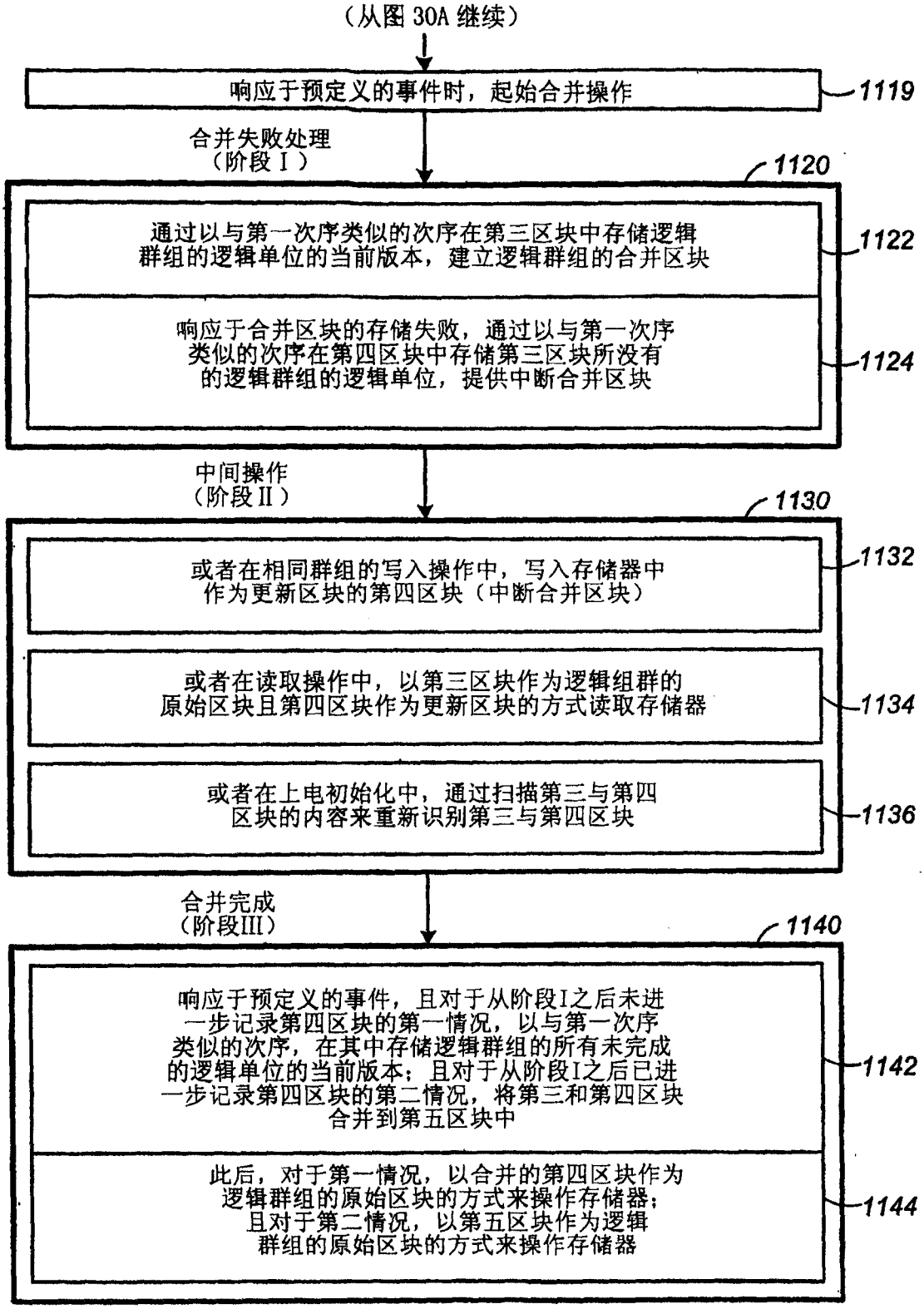

（从图 30A 继续）
响应于预定义的事件时，起始合并操作
1119
合并失败处理
（阶段 I）
1120
通过以与第一次序类似的次序在第三区块中存储逻辑群组的逻辑单位的当前版本，建立逻辑群组的合并区块
1122
响应于合并区块的存储失败，通过以与第一次序类似的次序在第四区块中存储第三区块所没有的逻辑群组的逻辑单位，提供中断合并区块
1124
中间操作
（阶段 II）
1130
或者在相同群组的写入操作中，写入存储器中作为更新区块的第四区块（中断合并区块）
1132
或者在读取操作中，以第三区块作为逻辑组群的原始区块且第四区块作为更新区块的方式读取存储器
1134
或者在上电初始化中，通过扫描第三与第四区块的内容来重新识别第三与第四区块
1136
合并完成
（阶段III）
1140
响应于预定义的事件，且对于从阶段I之后未进一步记录第四区块的第一情况，以与第一次序类似的次序，在其中存储逻辑群组的所有未完成的逻辑单位的当前版本；且对于从阶段I之后已进一步记录第四区块的第二情况，将第三和第四区块合并到第五区块中
1142
此后，对于第一情况，以合并的第四区块作为逻辑群组的原始区块的方式来操作存储器；且对于第二情况，以第五区块作为逻辑群组的原始区块的方式来操作存储器
1144

图 32B

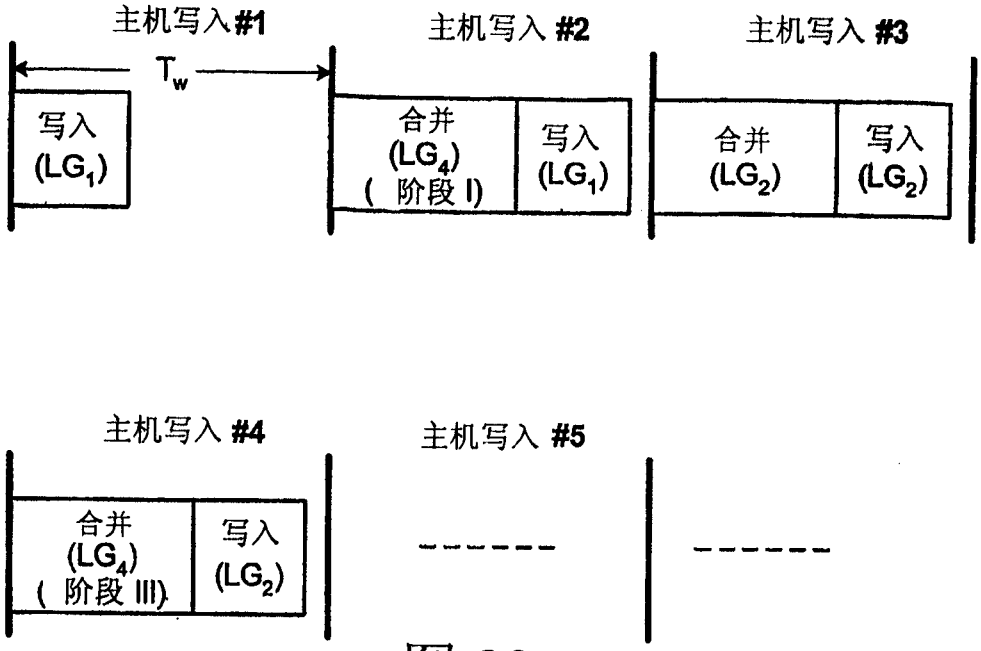
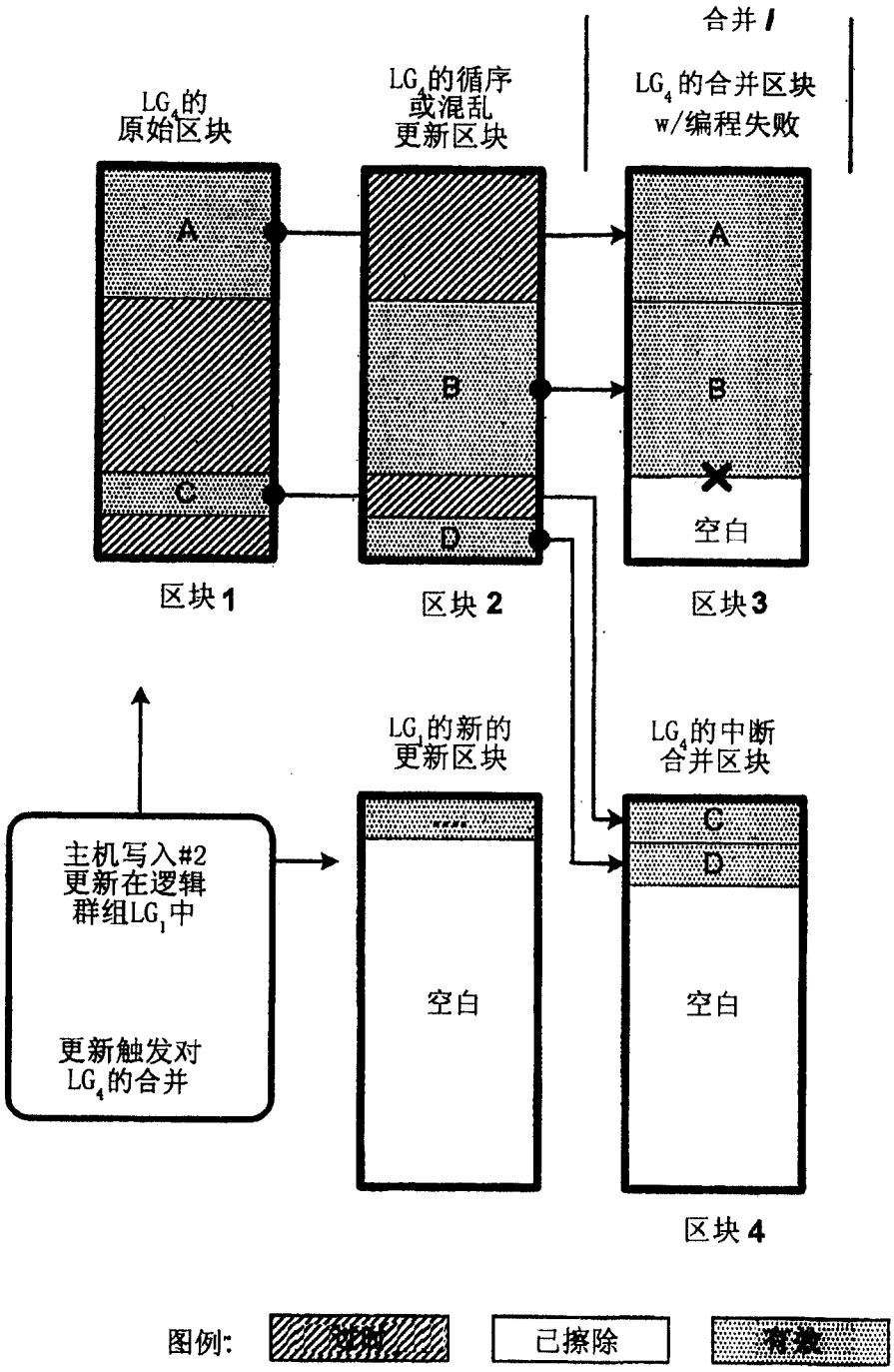
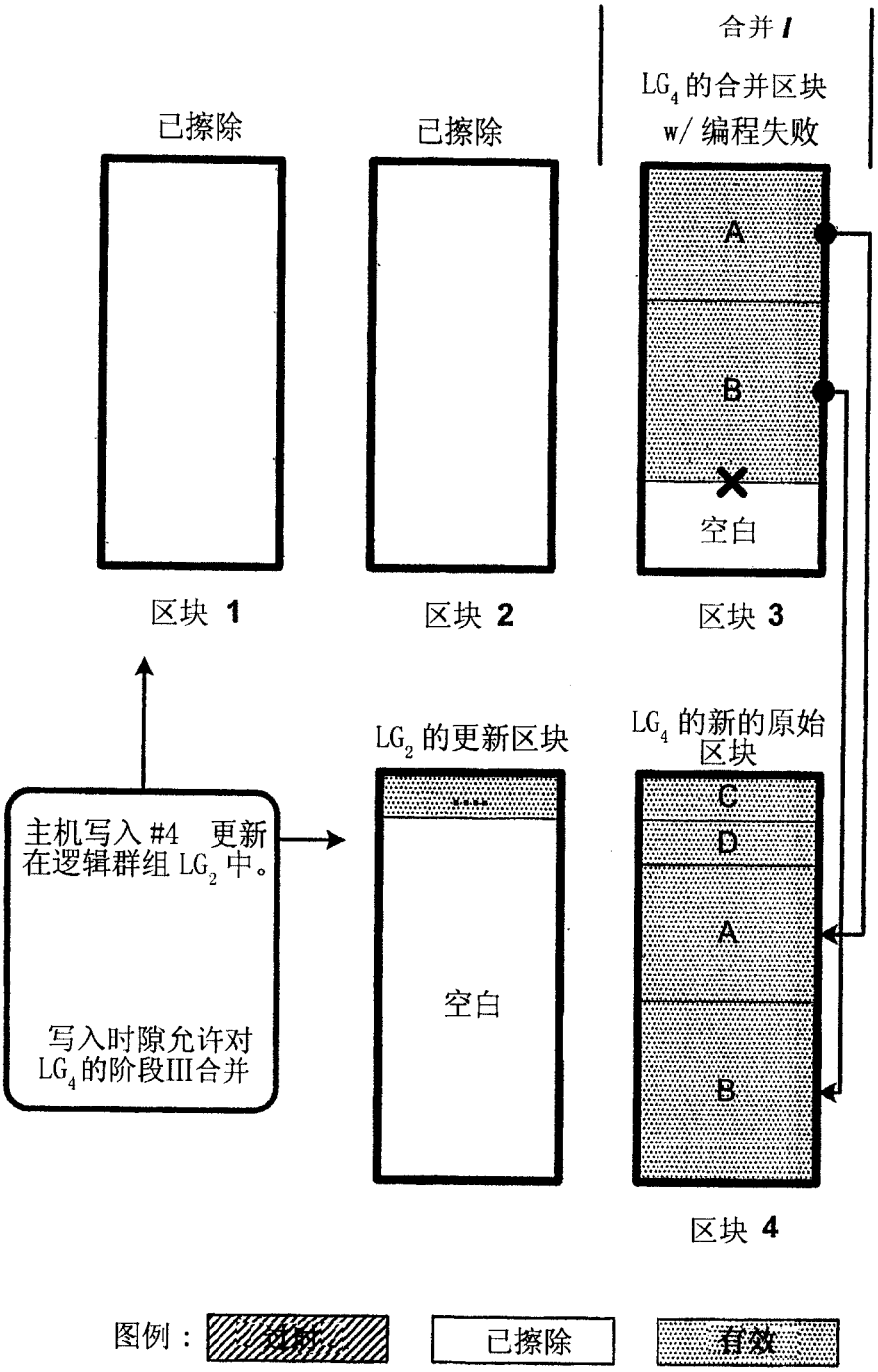

主机写入#1
T_w
写入
(LG_1)
主机写入 #2
合并
(LG_4)
(阶段 I)
写入
(LG_1)
主机写入 #3
合并
(LG_2)
写入
(LG_2)
主机写入 #4
合并
(LG_4)
(阶段 III)
写入
(LG_2)
主机写入 #5

图 33



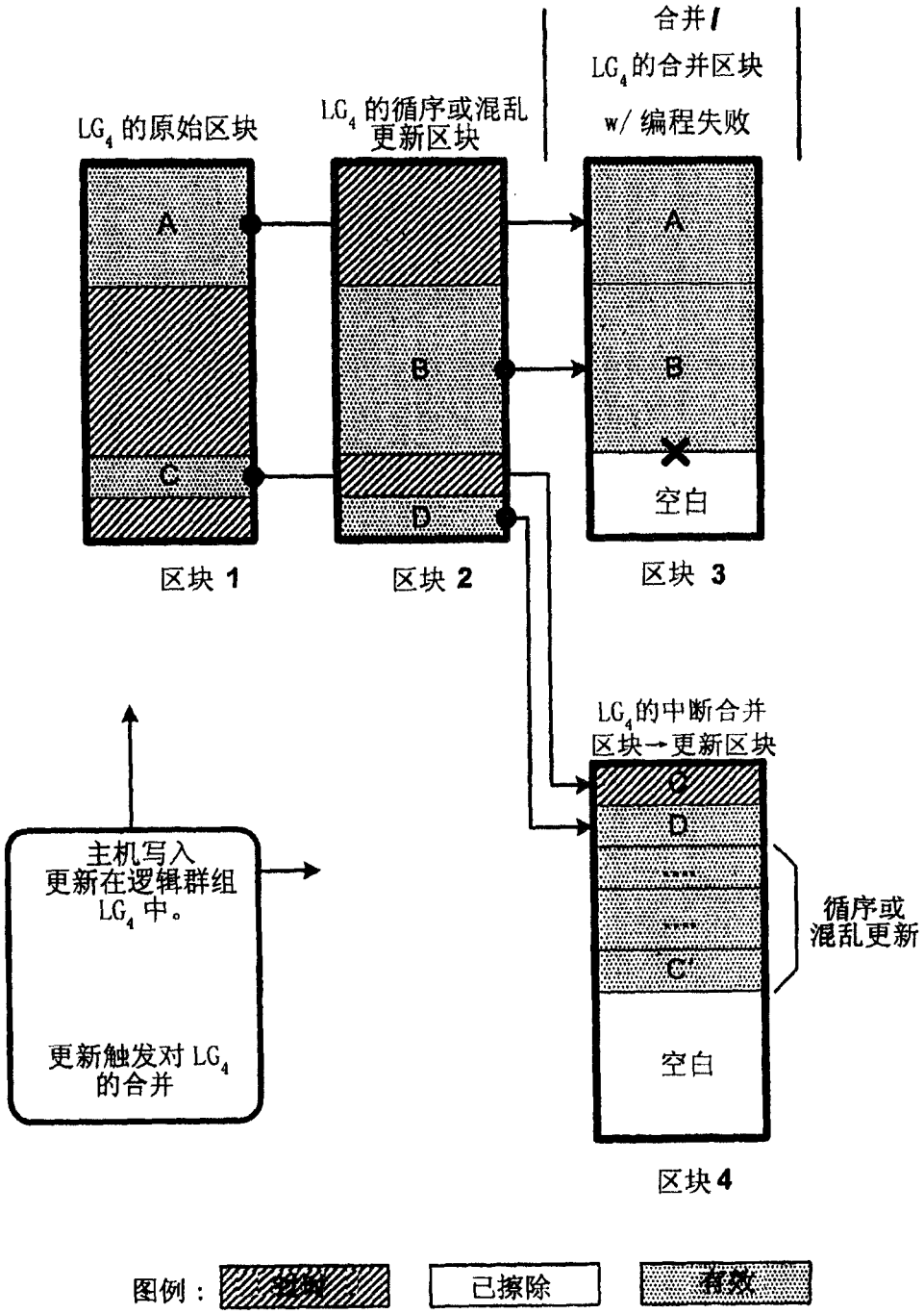
合并失败处理（阶段 I）
（情况1：中断区块上没有额外的更新）

图34A



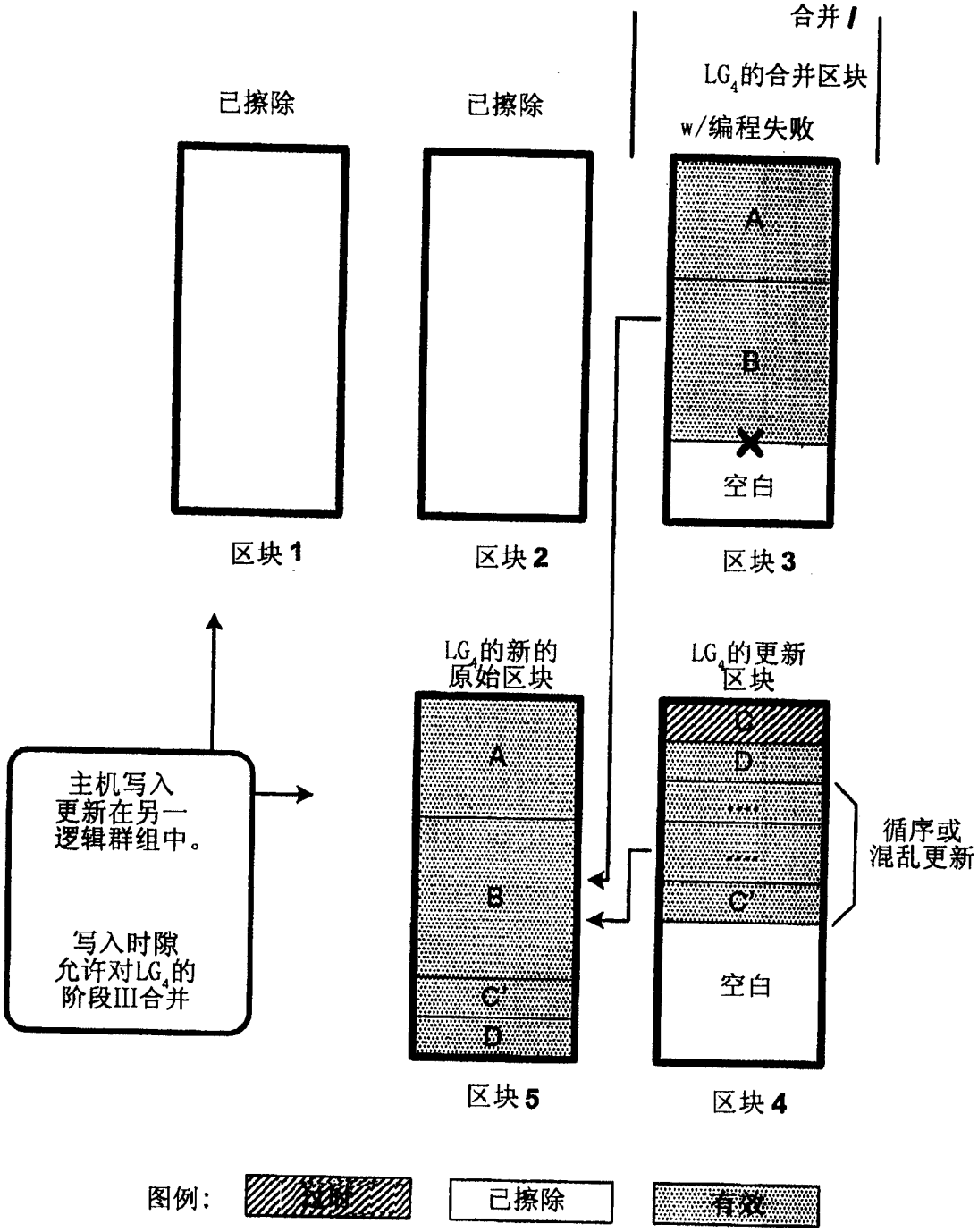
合并失败处理（阶段III）
（情况 1：中断区块上没有额外的更新）

图 34B



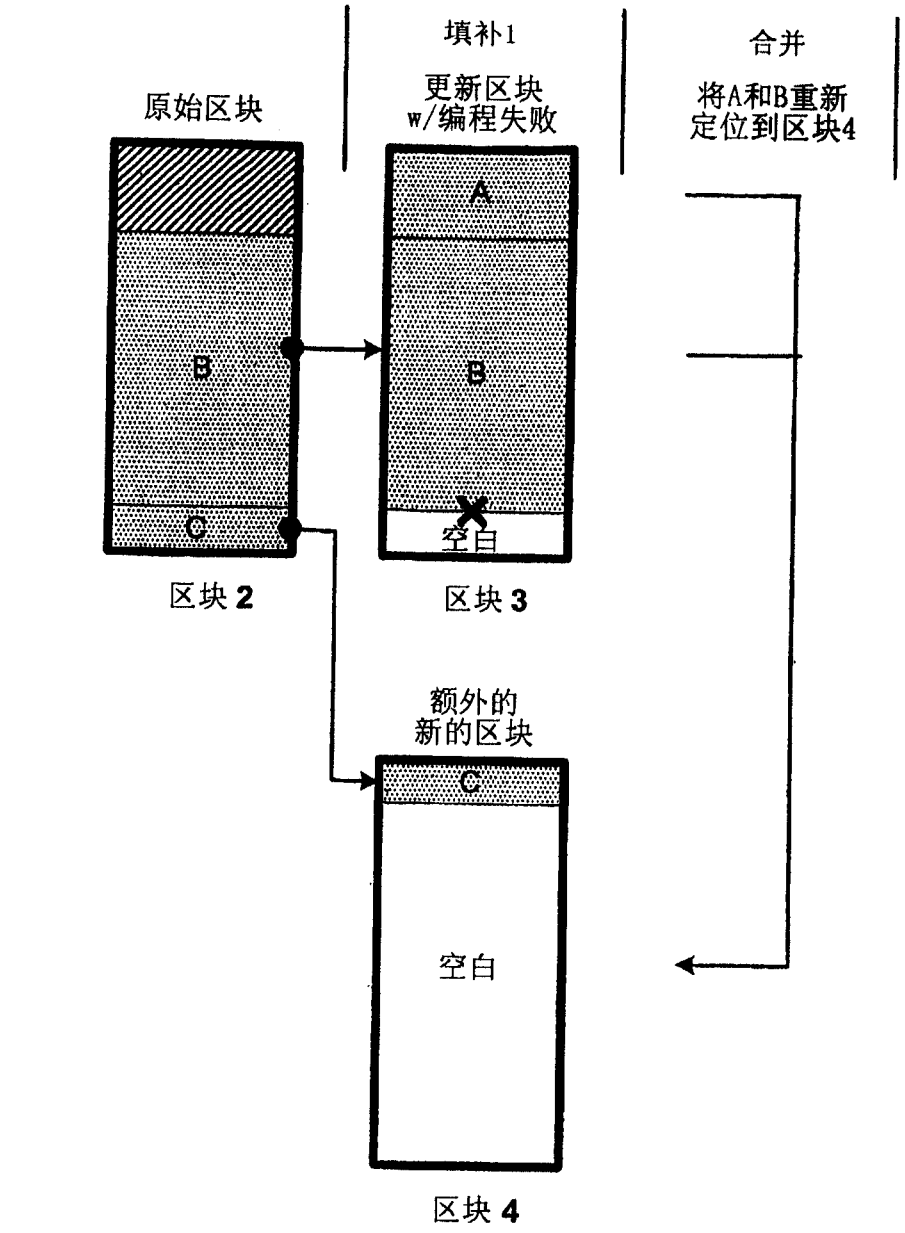
合并失败处理（阶段 I）
（情况 2：中断区块上有额外的更新）

图 35A



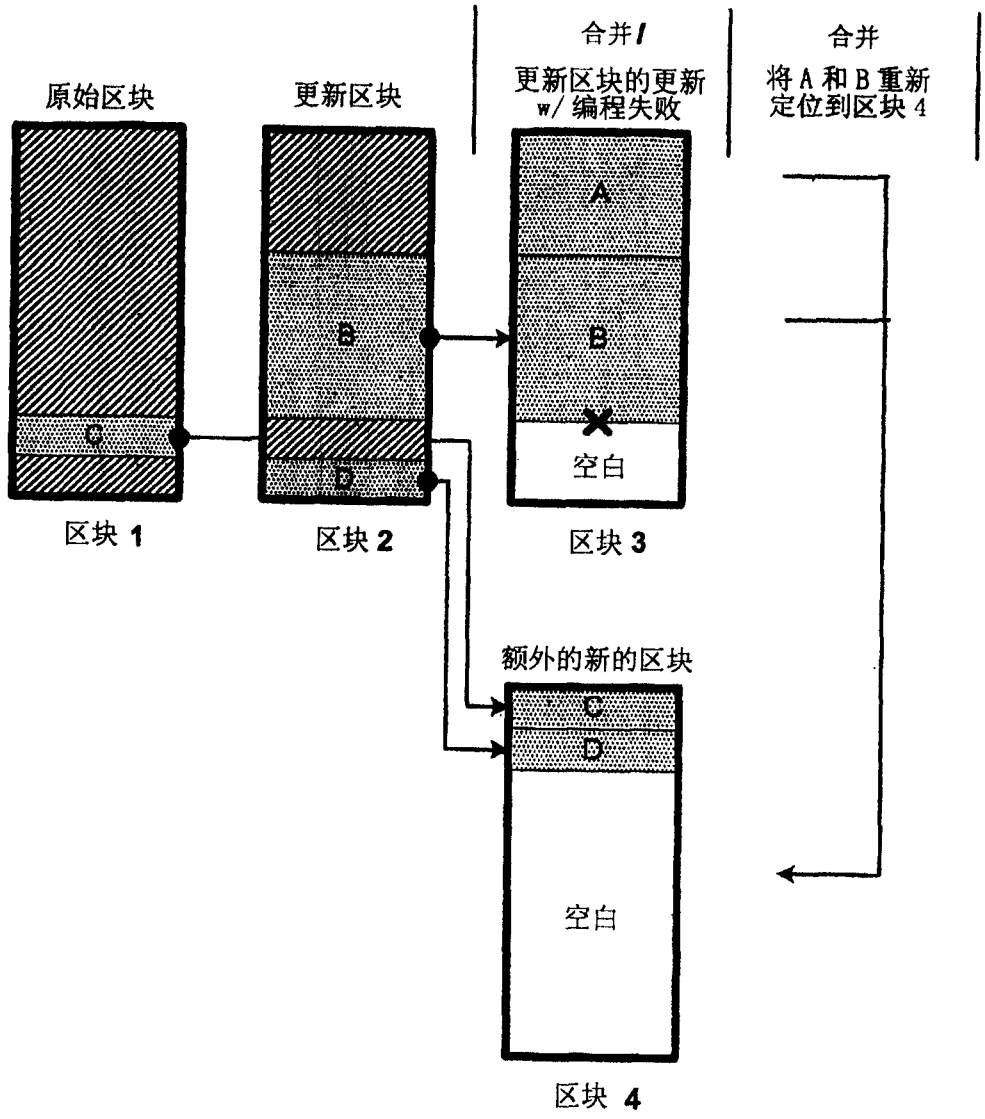
合并失败处理（阶段III）
（情况2：中断区块上有额外的更新）

图35B



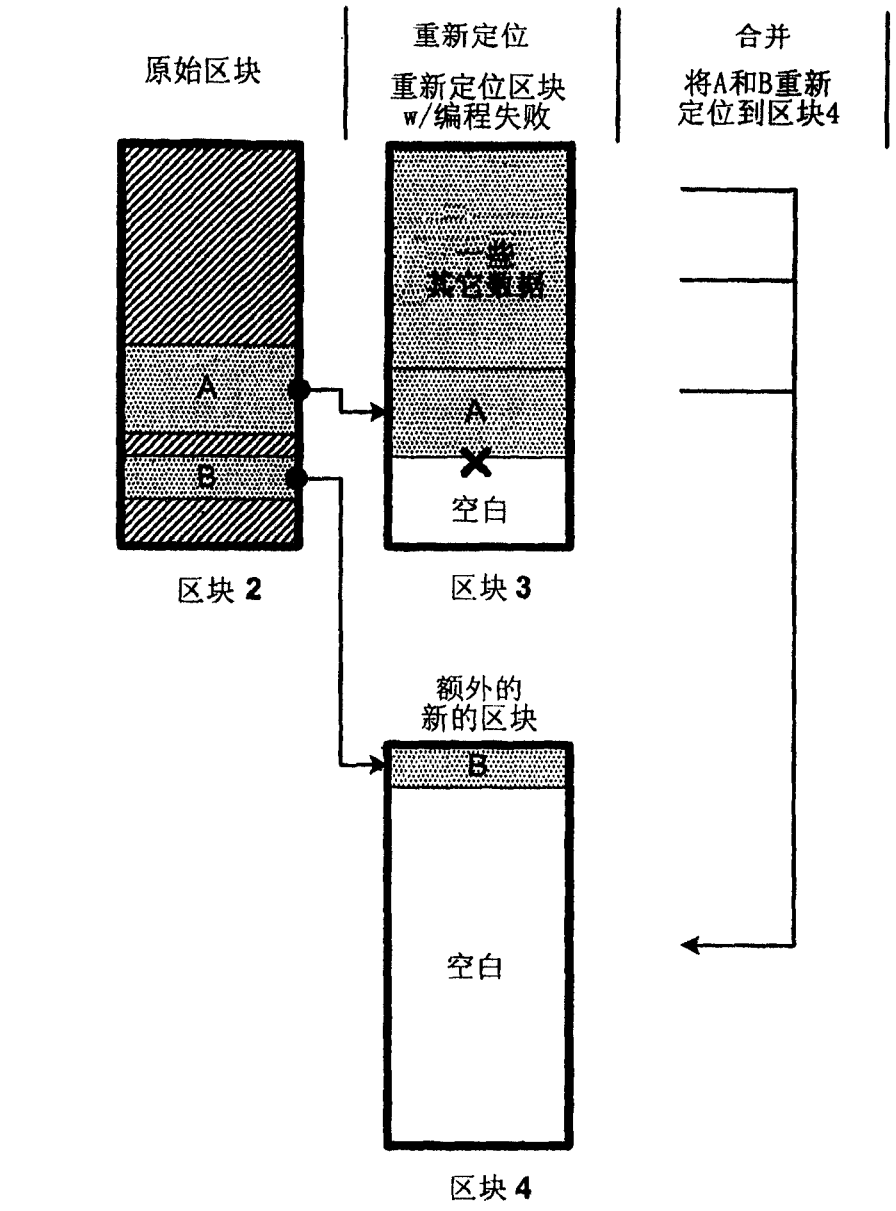
循序填补失败处理（阶段 I 和III）

图36A



更新区块的更新的合并的
失败处理（阶段Ⅰ和Ⅲ）

图 36B



重新定位失败处理
（不进行逻辑编组）
（阶段 I 和III）

图36C

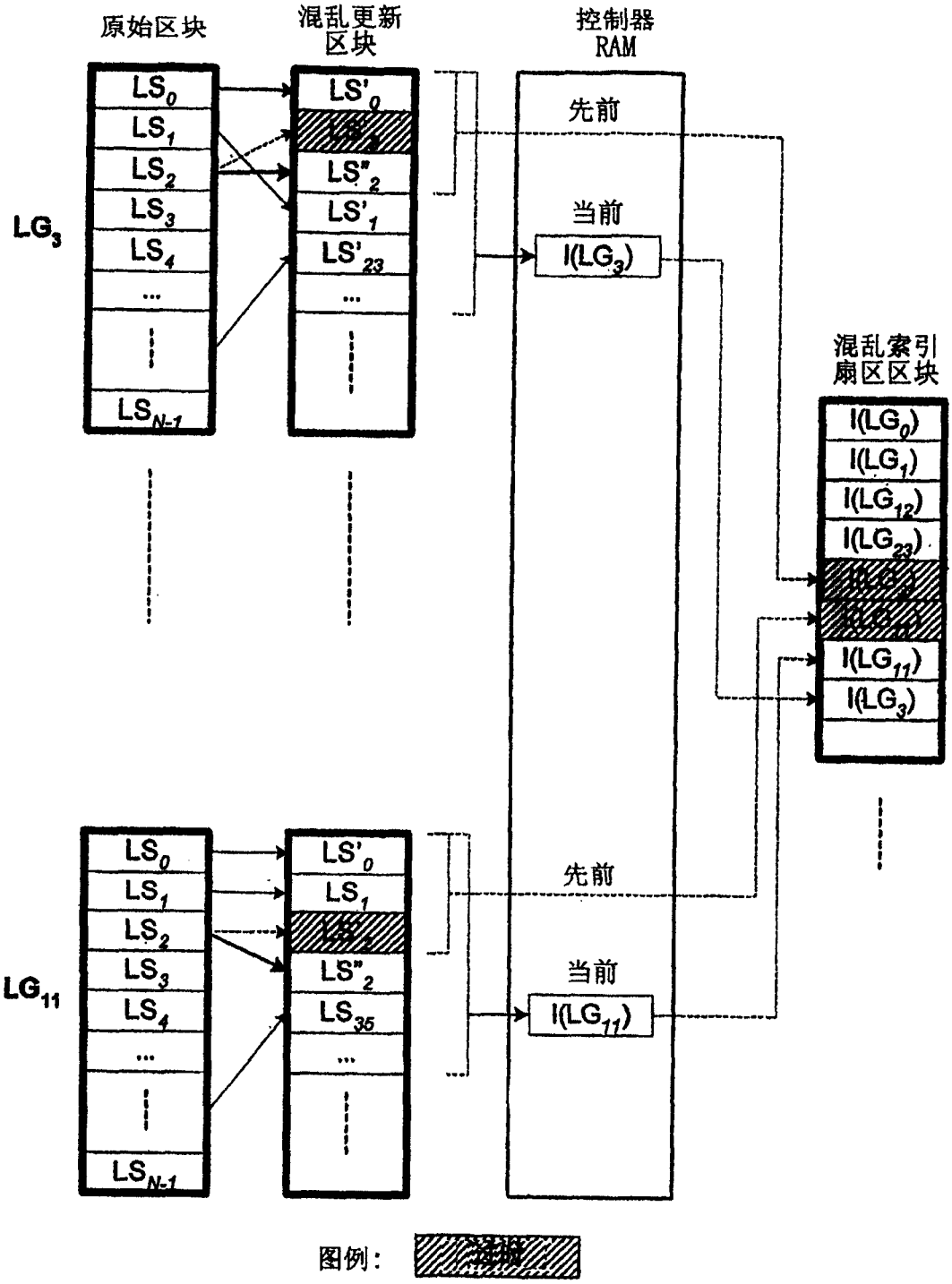

原始区块
混乱更新区块
控制器RAM
LG3
LS0
LS1
LS2
LS3
LS4
LSN-1
LS'0
LS'2
LS"2
LS'1
LS'23
先前
当前
I(LG3)
混乱索引扇区区块
I(LG0)
I(LG1)
I(LG12)
I(LG23)
I(LG11)
I(LG3)
LG11
LS35
I(LG11)
图例:

图37

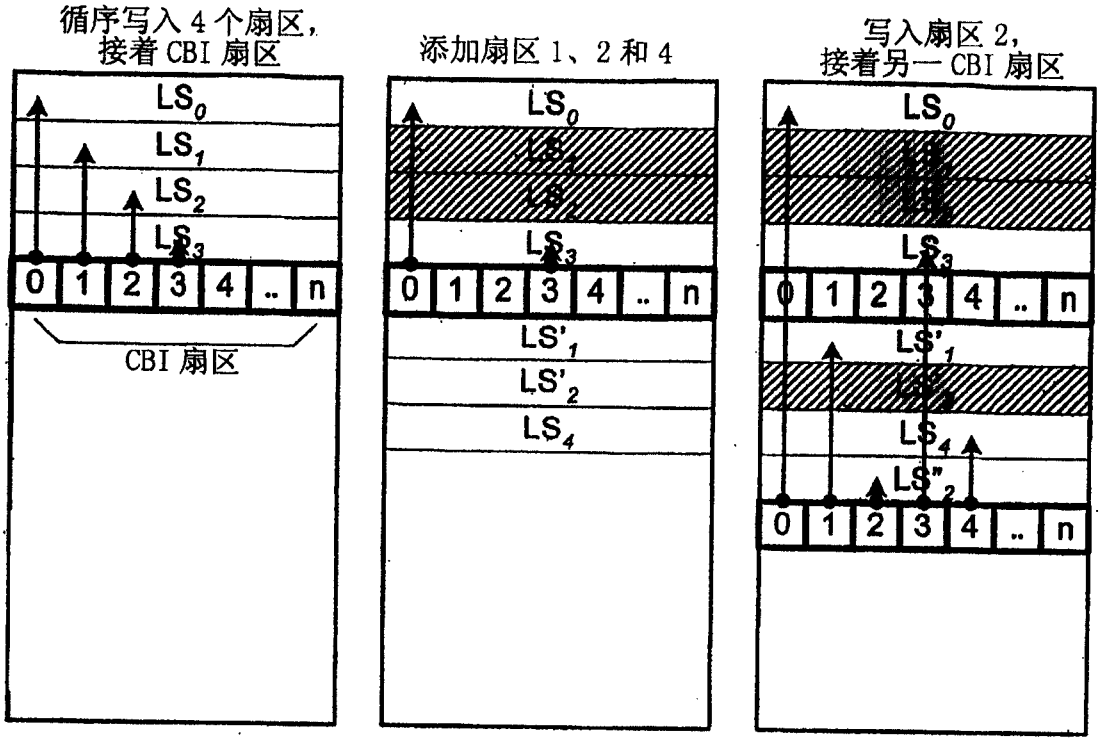


图 38A 图 38B 图 38C

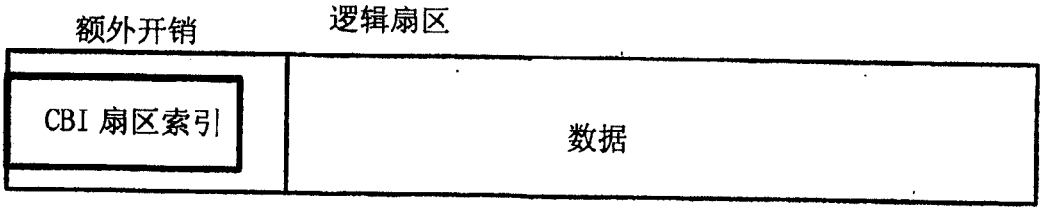


图 40

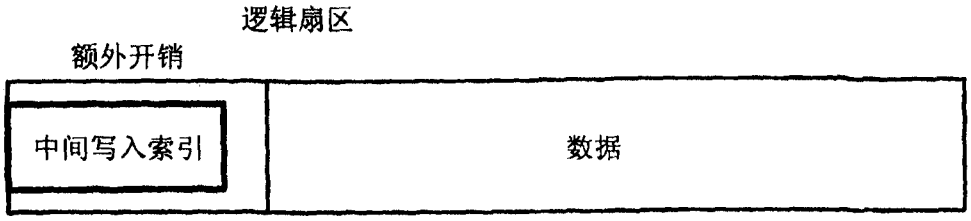

逻辑扇区
额外开销
中间写入索引
数据

图39A

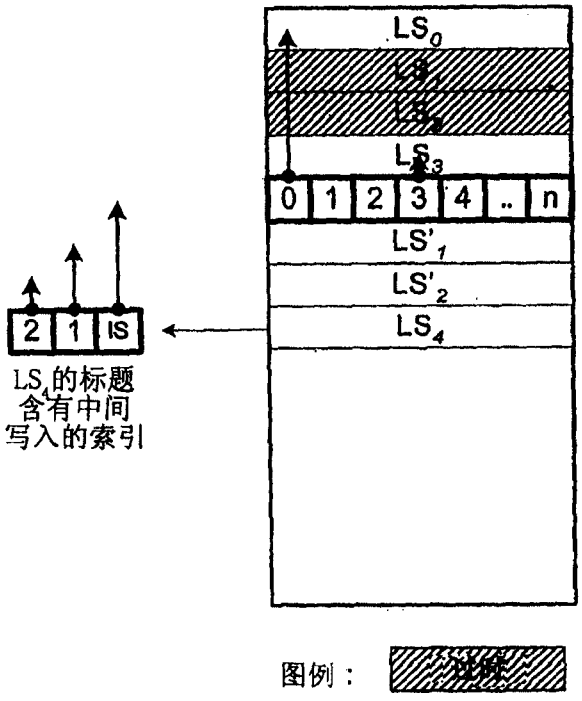
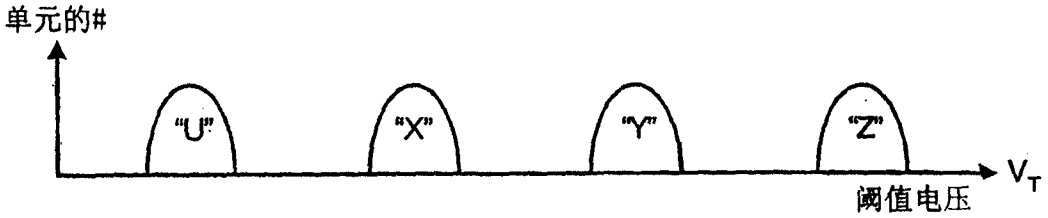
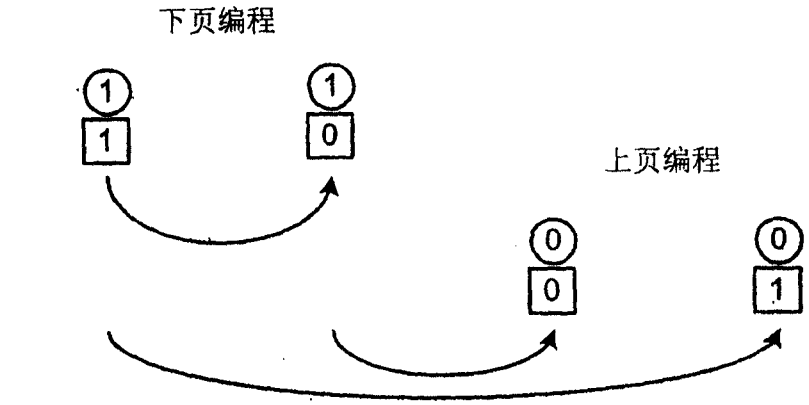

LS0
LS1
LS2
LS3
0 1 2 3 4 .. n
LS'1
LS'2
LS4
2 1 IS
LS4的标题
含有中间
写入的索引
图例：
说明

图39B



多状态存储器

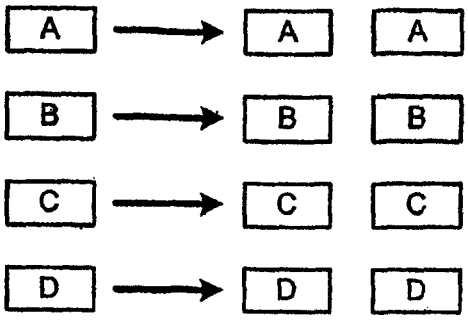
图41A



图例： 逻辑上页 逻辑下页

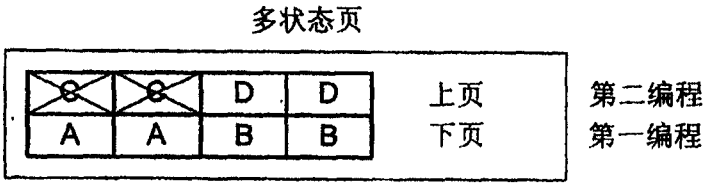
以葛莱码进行2次编程

图41B



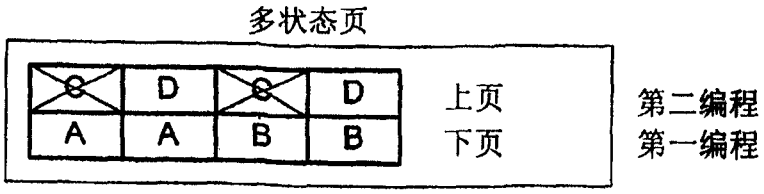
要复制的扇区

图42



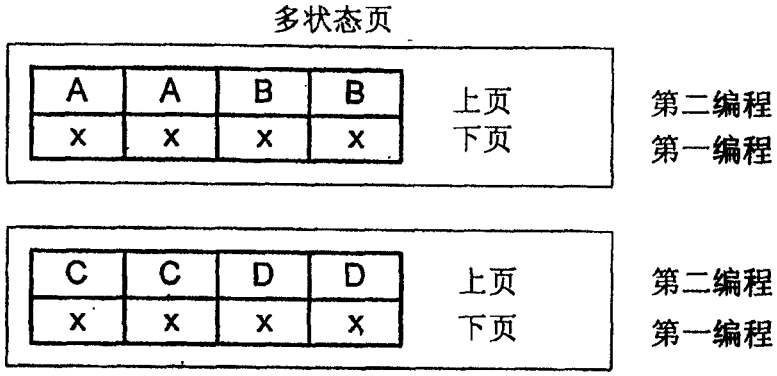
页中的复制扇区的非稳固排列

图43



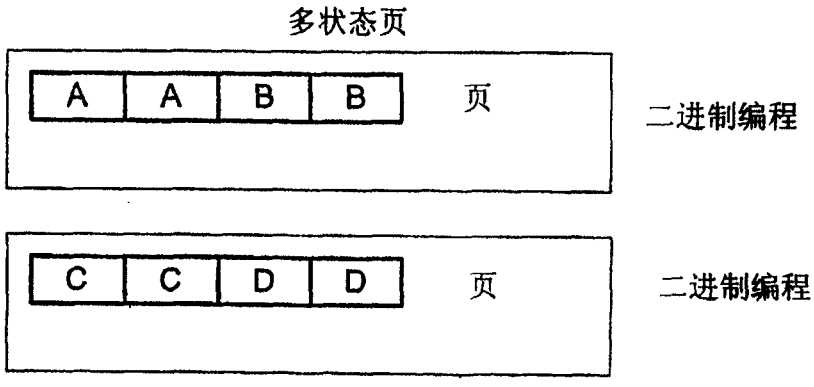
页中的复制扇区的稳固排列

图44A



仅在上页中的复制扇区

图44B



二进制编程模式中的复制扇区

图44C

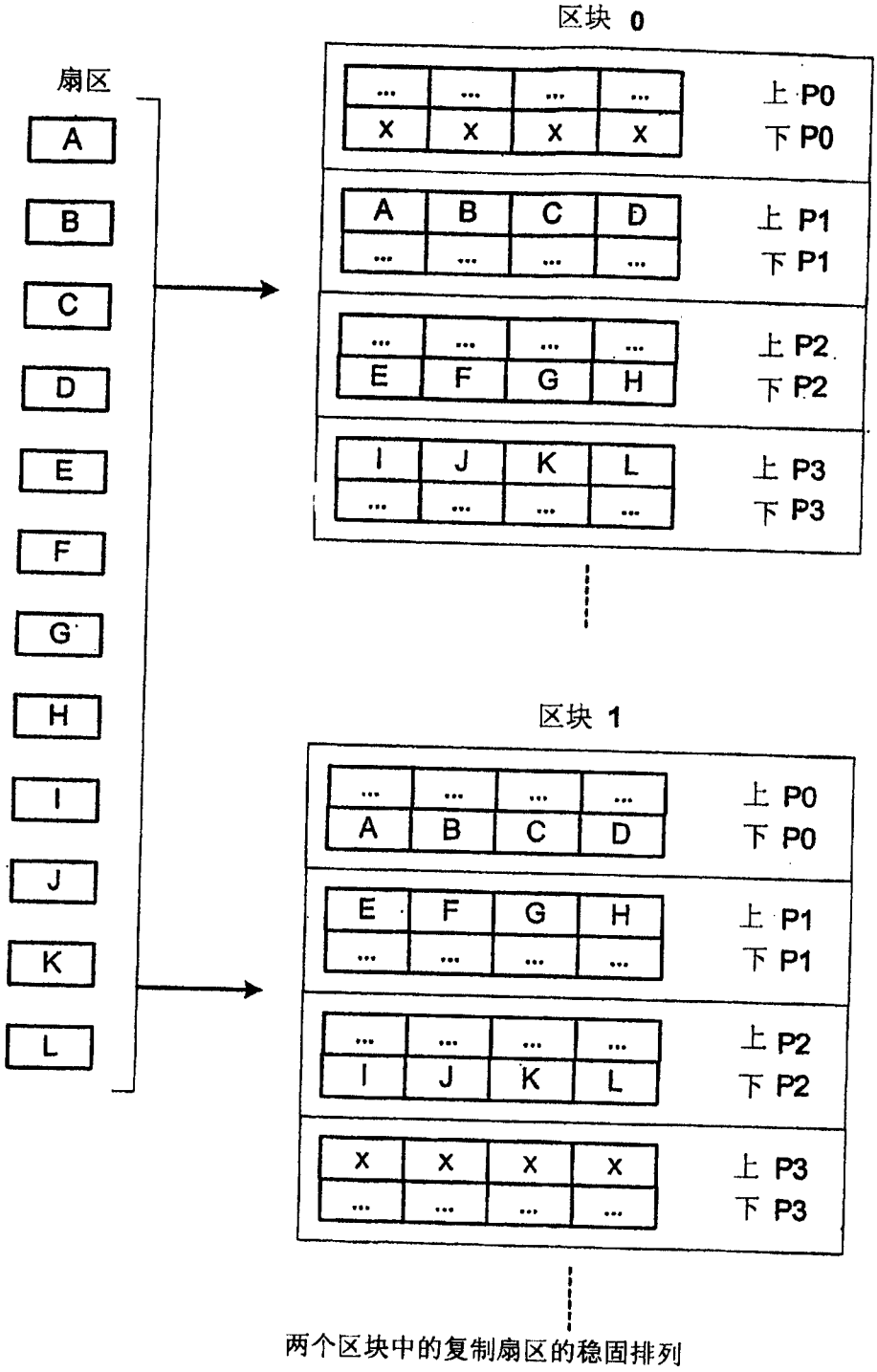
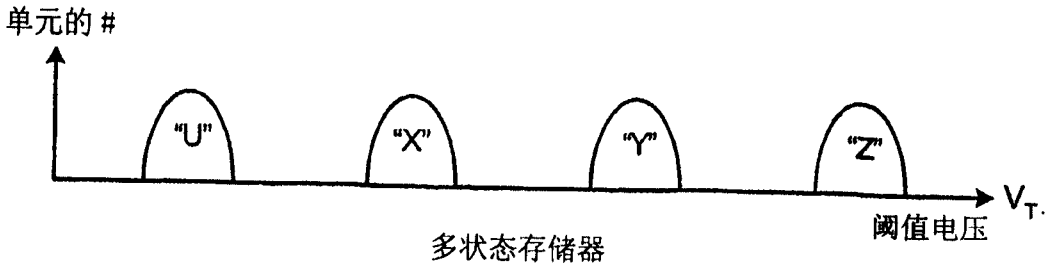
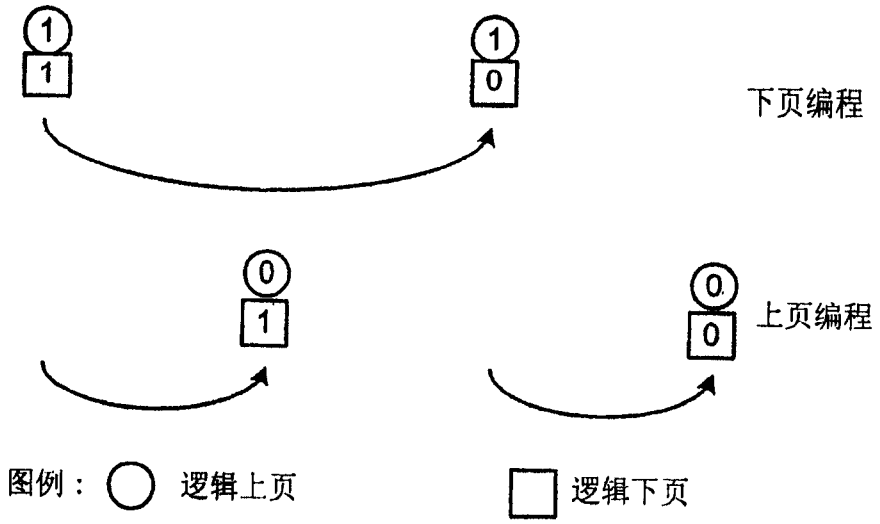

区块 0
扇区
A
B
C
D
E
F
G
H
I
J
K
L
... ...
X X X X
上 P0
下 P0
A B C D
... ...
上 P1
下 P1
... ...
E F G H
上 P2
下 P2
I J K L
... ...
上 P3
下 P3
区块 1
... ...
A B C D
上 P0
下 P0
E F G H
... ...
上 P1
下 P1
... ...
I J K L
上 P2
下 P2
X X X X
... ...
上 P3
下 P3
两个区块中的复制扇区的稳固排列

图 45



多状态存储器

图 46A



以容错码进行 2 次编程

图 46B

状态	副本 1	副本 2	有效副本
1	EEC 良好	EEC 良好	副本 1 & 2
2	EEC 良好	EEC 不良	副本 1
3	EEC 良好	空白	副本 1
4	EEC 不良	空白	没有

图 47

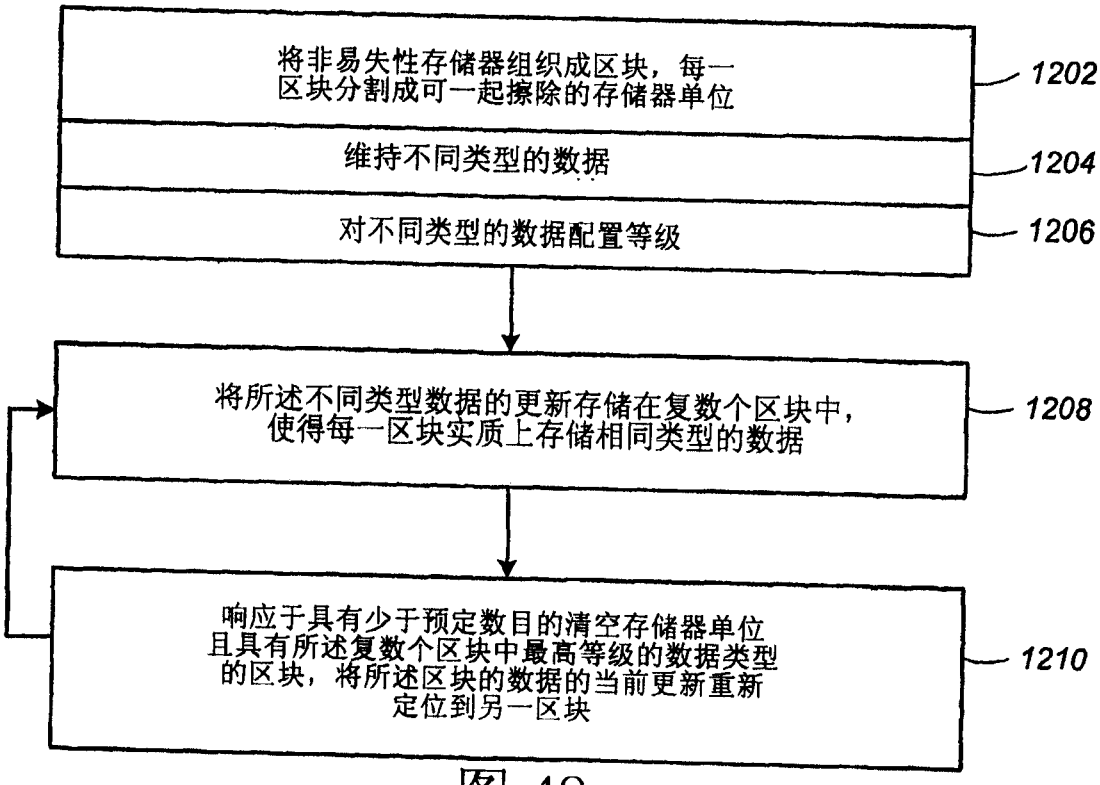


图 48