[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 17/30 (2006.01)
G06F 17/40 (2006.01)

[12] 发明专利申请公布说明书

[21] 申请号 200680050056.1

[43] 公开日 2009 年 1 月 21 日

[11] 公开号 CN 101351799A

[22] 申请日 2006.11.17
[21] 申请号 200680050056.1
[30] 优先权
[32] 2005.12.30 [33] US [31] 11/275,434
[86] 国际申请 PCT/US2006/044735 2006.11.17
[87] 国际公布 WO2007/078444 英 2007.7.12
[85] 进入国家阶段日期 2008.6.30
[71] 申请人 微软公司
地址 美国华盛顿州
[72] 发明人 J·R·豪威尔 J·R·道瑟

[74] 专利代理机构 上海专利商标事务所有限公司
代理人 张政权

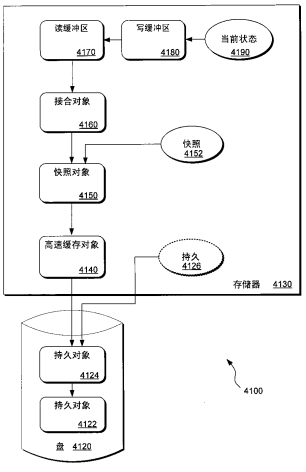
权利要求书 4 页 说明书 32 页 附图 25 页

[54] 发明名称

用增量分页器来管理状态

[57] 摘要

一种增量分页器用原子的、隔离的事务来维护数据库。 当一事务试图对该数据库作出改变时，该增量分页器将改变储存在写缓冲区中，并在介入事务没有字面上或实质上改变该事务所依赖的数据库状态时应用该改变。 该增量分页器通过将写缓冲区与数据库的当前状态接合来形成表示数据库状态的新数据结构来应用改变以提交事务。 该增量分页器根据该增量分页器遵守的快照来接合写缓冲区以维持效率，以便保留数据库的所选状态。 该增量分页器通过将所选数据移至持久存储来使得数据库的所选部分变得持久。 该增量分页器还提供在持久存储和当前事务之间的高速缓存对象以提高对数据的访问效率。



1．一种用于管理数据库的计算机实现的方法，包括：

将所述数据库作为一系列部分映射（100、310）来维护，其中所述部分映射中的每一个包括地址到值的至少一个分配；

识别一所选部分映射的状态为对追加到先前添加到所述系列的前导部分映射（110）的部分映射（310）的编译；

建立指向所选部分映射（310）的指针（120）；以及

限制对所选部分映射(110、310)的改变，使得对为其建立了所述指针(120)的所选部分映射的状态没有改变。

2．如权利要求 1 所述的方法，其特征在于，还包括确定在所选映射内所述地址到值的分配，包括：

从由所述指针（120）指示的状态开始，并从由所述指针（120）指示的状态向着所述系列的相对一端前进；以及

访问所选部分映射，直到满足以下条件之一：

找到所述地址到值的第一分配；以及

到达所述系列的末尾。

3．如权利要求 1 所述的方法，其特征在于，所述所选部分映射包括最近被追加到所述部分映射系列的部分映射（310），并且所述指向所选部分映射的指针包括当前状态指针（120）。

4．如权利要求 3 所述的方法，其特征在于，还包括：

为一事务试图在所述数据库中改变的至少一个分配创建一新的部分映射（310）；以及

当启动所述事务时，将所述新的部分映射（310）指向由所述当前状态指针（120）指示的状态。

5．如权利要求 4 所述的方法，其特征在于，还包括通过在满足以下条件之一时将所述当前状态指针（120）指向所述新的部分映射，在所述事务的执行一完成时就提交所述事务：

当启动所述事务时所述当前状态指针（120）继续指向由所述当前状态指

针（120）指示的状态；以及

在启动所述事务时由所述当前指针（120）指示的状态与由所述当前状态指针（120）指示的新状态之间没有介入事务改变了所述事务访问的前一分配。

6．如权利要求 5 所述的方法，其特征在于，还包括在未提交所述事务时，通过丢弃所述新的部分映射来中止所述事务。

7．如权利要求 1 所述的方法，其特征在于，对所选部分映射建立所述指针以保留所选部分映射的快照（3210）。

8．如权利要求 1 所述的方法，其特征在于，还包括：

标识所述系列中没有对其建立指针的第一部分映射（3110）；

标识在将所述第一部分映射（3110）追加到所述系列之后追加到所述系列的第二部分映射；

通过以下动作之一将包括在所述第二部分映射（3110）中的每一分配插入到所述第一分配中：

向所述第一分配（3110）添加包括在所述第二部分映射中的、先前未包括在所述第一部分映射中的任何分配；以及

用具有包括在所述第二部分映射中的分配的分配来盖写同时包括在所述第一部分映射（3110）和所述第二部分映射两者中的任何分配；以及

从所述系列中移除所述第二部分映射。

9．如权利要求 1 所述的方法，其特征在于，还包括将来自低速存储的数据高速缓存在高速存储中，包括：

将第一部分映射（3910）储存在所述高速存储（3960）中；

将第二部分映射（3930）储存在所述低速存储（3950）中；以及

对所述系列中的部分映射排序，使得所述第一部分映射（3910）的第一状态通过参考所述第二部分映射（3930）来定义。

10．如权利要求 9 所述的方法，其特征在于，所述高速存储（3960）包括存储器，而所述低速存储（3950）包括盘存储。

11．如权利要求 9 所述的方法，其特征在于，还包括以下动作的至少之一：

将至少一个分配从所述第二部分映射复制到所述第一部分映射（4008）；以及

将所述至少一个分配从所述第一部分映射复制到所述第二部分映射（4012）。

12. 如权利要求 9 所述的方法，其特征在于，还包括在向所述第一部分映射写入新分配时将所述新分配写入所述第二部分映射（3808）。

13. 一种用于管理针对数据库的事务的计算机实现的方法，包括：

标识在启动事务时所述数据库的状态（1204）；

将所述事务试图写入所述数据库的数据储存在一缓冲区中（1214）；

维护指向所述数据库的当前状态的当前状态指针（120），使得所述当前状态指针指向所述当前状态中最近插入的对象；

在试图提交所述事务时（2102），寻找所述事务试图写入的数据与当前状态不一致的指示（2104）；

在没有找到所述事务试图写入的数据与所述当前状态不一致的指示时，通过将所述缓冲区指向所述当前状态并将所述当前状态指针指向作为所述最近插入的对象的缓冲区（2106）来提交所述事务。

14. 如权利要求 13 所述的方法，其特征在于，还包括从所述数据库中寻找所需数据，包括：

标识所述当前状态指针（120）指示的当前状态；以及

连续地访问所述当前状态中的每一对象，直到首次找到储存了所需数据的对象或发现所述当前状态没有储存所需数据。

15. 如权利要求 13 所述的方法，其特征在于，所述事务试图写入的数据与所述当前状态不一致的指示包括以下的至少一个：

确定所述当前状态不同于当启动所述事务时所述数据库的状态（2904）；以及

确定所述当前状态不同于当启动所述事务时所述数据库的状态并且所述当前状态指示一介入事务执行了以下动作的至少一个：

将数据写入所述事务读取的地址；以及

改变了所述事务读取的地址处的数据（2906）。

16. 如权利要求 13 所述的方法，其特征在于，还包括将储存在所述当前状态中的多个对象中的数据接合成一接合对象（3510）。

17. 如权利要求 16 所述的方法，其特征在于，还包括通过防止所述数据库的一所选状态与在所选状态之后添加到所述当前状态的一个或多个对象接合来保留所述所选状态。

18. 一种用于管理针对数据库的事务的计算机实现的方法，包括：

维护指向所述数据库的当前状态的当前状态指针（120）；

标识在启动事务时存在的所述数据库的当前状态（1206）；

为所述事务创建一写缓冲区（1206）；

将所述事务试图写入所述数据库的数据储存在所述写缓冲区中（1214）；

将所述写缓冲区指向所述当前状态（1206）；以及

将一事务指针指向所述写缓冲区以表示在所述事务被提交给所述数据库时将被应用于所述数据库的改变的部分映射（1208）。

19. 如权利要求 18 所述的方法，其特征在于，当所述事务试图将另外的数据写入所述数据库时，还包括：

创建一附加写缓冲区（1212）；

将所述另外的数据储存在所述附加写缓冲区中（1212）；

将所述附加写缓冲区指向所述写缓冲区（1214）；以及

将所述事务指针指向所述附加写缓冲区（1216）。

20. 如权利要求 18 所述的方法，其特征在于，还包括当满足以下条件之一时将所述事务提交给所述数据库：

所述当前状态保持与在启动所述事务时存在的状态相同（2904）；以及

所述当前状态已改为包括一介入事务，除非所述事务已读取了所述介入事务执行了以下动作中的至少一个的值：

写；以及

改变（2906）。

用增量分页器来管理状态

背景

执行数据库事务时的一个关键问题是数据库完整性。为保持数据库完整，重要的是确保事务遵守诸如原子性和隔离等要求。原子性要求命令要么执行一事务的全部任务，要么不执行任何任务，因而不完整的事务不会被应用于数据库。隔离要求规定事务与其它事务分开处理，因而没有事务访问或干涉另一事务的中间状态。遵守原子性和隔离要求防止事务彼此破坏并破坏作为整体的数据库。

在联网系统中，许多用户和许多应用程序可能并发地试图对同一数据执行事务。即使在单机系统中，多个应用程序也可能用彼此竞争的每一应用程序的多个处理线程来并发地试图对同一数据执行事务。由于难以预见，更不用说避免在事务中可能向数据库应用冲突或不一致的改变的所有情形，因此维持数据库完整性是一项重大的挑战。

保持数据库完整性的一种方式是对当前事务使用的数据库的一部分加上锁。对数据库的一部分加上锁防止其它事务在当前事务完成之前读取或盖写该数据。防止其它事务从数据库的该部分读取确保其它事务将不会基于当前事务可能改变的值而得出结果。因此，防止其它事务盖写数据库的该部分确保其它事务不会影响当前事务。

然而，尽管对数据库的一部分加上锁有可能有助于保持数据完整性，但是它也会导致补偿成本。对多个事务试图访问的数据库的一部分加上锁可造成事务的大量积累。部分或全部等待的事务可能不会对数据库的当前状态作出任何改变，或可能不会对当前事务所使用的特定值进行读取、写入或改变。然而，对数据库的该部分加上锁将保持数据库的完整性，即使它使得对数据库的访问变慢。

一种形式的锁是租用。正如在公寓、建筑物、汽车和设备租用的情况中一样，租用是一段有限时间内的独占授权。由此，当一事务被授予对所选数据的

租用时，向该事务提供在一段有限时间内对所选数据的独占访问。通过将对该数据的独占访问限于一段时间，如果该事务在事务完成时无法释放数据，或者万一该事务在其上运行的系统崩溃，则数据将被解锁，使得其它事务不必无限制地等待来访问该数据。常规上，等待所选数据的每一事务将必须等待直到先前对该数据排队的所有事务完成其对该数据的使用，或直到分配给每一事务的租用期满。

随着等待访问租用数据的事务数量的增加，可能会有显著的延迟。另外，控制数据租用的系统在尝试访问数据时可造成瓶颈。作为对数据访问的单个控制点控制租用的系统在试图响应租用请求时可导致事务延迟。此外，由于系统可能正在接收对同一所选数据的多个请求，因此处理这多个且可能重复的请求将浪费计算周期，从而导致对寻求同一数据或任何其它数据的事务的进一步延迟。

概述

一种增量分页器用原子的、隔离的事务来维护数据库。当一事务试图对该数据库作出改变时，该增量分页器将改变储存在写缓冲区中，并在介入的事务没有在字面上或实质上改变该事务所依赖的数据库的状态时应用改变。该增量分页器通过将写缓冲区与数据库的当前状态联合以形成表示该数据库的状态的新数据结构来应用改变以提交事务。该增量分页器根据该增量分页器所遵守的快照来接合写缓冲区以维持效率，以便保留数据库的所选状态。该增量分页器通过将所选数据移至持久存储来使得数据库的所选部分持久。该增量分页器还提供持久存储和当前事务之间的高速缓存对象以促进对数据的高效访问。

提供本概述以便用简化的形式介绍将在以下详细描述中进一步描述的一些概念。本概述并不旨在确定所要求保护的主题的关键特征或必要特征，也不旨在用于帮助确定所要求保护的主题的范围。

附图简述

该详细描述将参考附图来描述。在附图中，三位数参考标号的最左边一位或四位数参考标号的最左边两位标识了该参考标号首次出现的附图。在不同附

图中使用相同的参考标号来指示相似或相同的项目。

图 1 示出了数据库的初始、当前状态。

图 2 示出了针对数据库的第一事务。

图 3-6 示出了使用缓冲区来执行并提交给数据库的事务。

图 7-11 示出了使用累积写缓冲区来执行的事务。

图 12 示出了使用写缓冲区执行事务的一种模式。

图 13 示出了使用单写缓冲区执行的事务。

图 14 示出了使用单写缓冲区来执行事务的一种模式。

图 15 示出了在将第一事务作为介入事务执行的同时针对数据库执行的第二事务。

图 16-20 示出了在唤醒介入事务时对当前事务的提交和中止。

图 21 示出了在唤醒介入事务时确定何时提交或中止一事务的一种模式。

图 22-27 示出了使用读和写缓冲区执行并提交给数据库的潜在竞争的事务。

图 28 示出了使用读和写缓冲区执行事务的一种模式。

图 29 示出了使用读和写缓冲区在唤醒介入事务时确定何时提交或中止一事务的一种模式。

图 30-31 示出了缓冲区的接合。

图 32-34 示出了使用快照对象来保留数据库的所选状态不被接合到后续状态中。

图 35 示出了在保留一个或多个所选状态的同时接合缓冲区的一种模式。

图 36-27 示出了变为持久的缓冲区。

图 38 示出了使对象变得持久的一种模式。

图 39 示出了高速缓存对象的使用。

图 40 示出了创建并维护高速缓存对象的一种模式。

图 41 示出了先前描述的对象集合。

图 42 示出了适用于执行先前所描述的数据库事务和操作的示例性操作环境。

详细描述

概览

术语“增量分页器”描述了用于在维持数据库的完整性并保留数据库的所选状态的同时处理对数据库的改变的方法和系统的各实施例。通过遵守识别的对象之间的指针的不变性，该增量分页器保留了数据库状态的完整性，同时还维持了效率。

在一种模式中，该增量分页器通过使用当前状态指针来跟踪数据库的当前状态来管理事务，其中该当前状态包括储存在该数据库中的所有数据的完整映射。在该完整映射中，对于为该数据库定义的每一地址或地址表示，将有一个数据值或空值。部分映射可包括该增量分页器用于储存一事务试图应用于在启动该事务时存在的数据库的当前状态的改变的一个或多个对象。如果该事务被提交给数据库，则该部分映射被追加到该数据库的原始当前状态以形成新的当前状态。当通过新应用的部分映射来访问该新的当前状态时，该部分映射中对地址的值分配取代了原始状态中的分配，且因此有效地盖写或改变了数据库的原始状态中的数据值。

为阐明并区分本描述中使用的术语，数据库的映射指的是对数据值的地址值分配。数据库的完整映射指的是数据库中的所有地址值对储存在这些地址处的数据值（或空值）的一组完整的分配。部分映射指的是一事务对数据库作出或试图作出的一组改变或盖写，包括该事务试图写入数据的地址、以及该事务试图写入到该地址的数据值。当一事务被提交给数据库时，一部分映射可被添加到先前的完整映射，得到该数据库的经更新的当前映射。由此，数据库的当前完整映射可包括一系列部分映射，其中后来添加的部分映射可盖写或改变包括在先前的部分映射中的地址分配。

数据库的状态指的是储存在存储器、盘或大容量存储中的对象集合，在该对象集合中储存了数据库的映射。状态中的对象包括增量分页器为事务创建的用于储存该事务试图写入数据库的改变的缓冲区，以及其中可收集缓冲区的其它对象。缓冲区、其它对象以及这些对象如何连接、接合和以其它方式操纵将在以下详细描述。数据库的当前状态包括呈现出数据库的当前的、完整的映射的那些对象。如以下解释的，增量分页器还可维护其它状态，诸如当启动一事

务时该数据库的状态。增量分页器跟踪该状态以确定该事务是否应当被提交给数据库。增量分页器还维护所选状态以保留这些状态以及这些状态所表示的数据库的映射以供稍后使用。

当每一事务启动时，增量分页器允许该事务基于在启动该事务时存在的数据库的状态来执行。该事务然后创建诸如写缓冲区等至少一个对象，并将该写缓冲区指向在启动该事务时存在的数据库的状态。增量分页器可使用一系列缓冲区，其中每一缓冲区储存该事务试图应用于数据库的单个改变，其中每一缓冲区指向保留该数据库的状态的前一缓冲区。或者，增量分页器可将一事务试图作出的所有改变储存在一累积写缓冲区中，该累积写缓冲区指向当启动该事务时的数据库的当前状态。指向启动事务时的状态的缓冲区或其它对象表示该事务试图应用于数据库的改变的部分映射。

如果增量分页器没有发现关于一个或多个介入事务阻止其应用当前事务的指示，则增量分页器通过将增量指针用于跟踪数据库的当前状态的当前状态指针改为指向为事务创建的最后一个缓冲区，或唯一缓冲区来提交事务。将当前状态指针改为指向缓冲区创建了该数据库的新的、经更新的状态，这可能改变数据库的完整映射中的数据值。

增量分页器可使用读缓冲区和写缓冲区来处理事务。读缓冲区跟踪该事务所访问的数据。当当前事务完成时，增量分页器使用读缓冲区来确定是否要提交该事务。如果读缓冲区显示当前事务读取了被介入事务盖写或改变的数据，则增量分页器可中止该事务。

在增量分页器提交了对数据库的多个改变之后，增量分页器可为了效率起见接合数据库的当前状态中的缓冲区或其它对象。增量分页器可将一个部分映射中的多个对象与另一部分映射中的对象接合。两个部分映射中较新的一个中的地址分配将被添加到或替换包括在较旧的部分映射中所包括的分配。接合可得到一个或多个接合的对象，包括存储器对象或持久对象，或者增量分页器可通过将储存在缓冲区中的改变应用于数据库的原始状态来接合缓冲区。增量分页器提供快照指针以允许保留所选状态。与快照指针相关联的任何状态将不与表示对由该快照指针保留的状态所做的改变的缓冲区接合。

增量分页器允许将数据库和对数据库所做的改变变为持久的。增量分页器

可将用于维护数据库的当前状态的一个或多个对象复制到持久的、非易失性存储中。增量分页器还提供了高速缓存对象以提供对储存在大容量介质中的数据库的当前状态的一部分，包括该增量分页器移至非易失性存储以使这些部分变得持久的当前状态的部分的更快速访问。增量分页器可使用高速缓存对象来储存数据库状态中的空数据范围以便于范围查询。

使用写缓冲区来处理事务

图 1 示出了数据库的当前状态 100。当前状态 100 包括数据库的示例性原始状态 110。数据库可驻留在易失性存储器或非易失性存储中。数据库分页器使用当前状态指针 120 来指向仅包括图 1 中的原始状态 110 的当前状态。

原始状态 110 是维护当前各自储存了一空值的五个记录，即记录 0 130、记录 1 140、记录 2 150、记录 3 160 和记录 4 170 的对象。注意空值无需实际上存储在原始状态中。原始状态 110 还可包括事务可向其应用改变的任何初始状态，包括空和非空值。

图 2 示出了包括多个指令 210-270 的一个示例性事务，即事务 1 200。在某些指令，诸如指令 1 200“x=Read(3)”试图从数据库中读取数据。其它指令，诸如指令 3 240“Write(2, x)”试图向数据库应用将导致将一值分配或重新分配给数据库中的地址的改变。

图 3 示出了增量分页器针对数据库启动诸如事务 1 200 等事务的一种模式。事务 1 200 以事务开始指令 trans_start 210 开始。在启动事务 1 210 之后，当前状态指针 120 继续指向在启动该事务时数据库的状态，即原始状态 110。增量分页器创建第一写缓冲区 310，其中增量分页器将储存事务 1 200 试图应用的改变。增量分页器将写缓冲区 310 指向启动事务 2 200 时的状态，该状态由当前状态指针 120 指示。增量分页器还创建事务指针 320，并将事务指针 320 指向第一写缓冲区 300，而第一写缓冲区 300 进而指向当前状态。

图 4-6 示出了增量分页器如何处理事务 1 200。参考图 2，指令 1 220“x=Read(3)”从地址 3 中读取 x 的当前值，即空值。指令 2 230“x=x+1”将 x 的值递增 1，这在这一情况下将 x 的值改为 1。指令 3 240“Write(2, x)”指示计算系统将 x 的值写入地址 2。

增量分页器不将值 1 写入原始状态 110 中的地址 2。相反，如图 4 所示，增量分页器创建第二写缓冲区 410。第二写缓冲区 410 指向第一写缓冲区 310，并且增量分页器将事务指针 320 切换为指向第二写缓冲区 410 以维护数据库的部分映射，该部分映射指示事务 2 200 将如何修改启动事务时的数据库的状态。增量分页器将把值 1 写入地址 2 的改变储存在第二写缓冲区 410 中。

图 5 示出了增量分页器对事务 1 200 试图对数据库作出的第二改变的响应。指令 4 250“Write(4, "DOG")”指示计算系统将串“DOG”储存在地址 4 中。再一次，增量分页器并非写入对原始状态 110 的改变，而是创建指向第二写缓冲区 410 的第三写缓冲区 510。增量分页器还将事务指针 320 改为指向第三写缓冲区 510 以保留数据库的状态（因为事务 1 200 将修改该状态），并将串“DOG”储存在第三写缓冲区 510 中。由此，从事务指针 320 中读取的对于事务 21200 的数据库的部分映射包括地址 2 储存值 1 以及地址 4 储存串“DOG”的修改。然而，事务既没有改变原始状态 110，也没有改变当前状态指针 120 所指示的当前状态。

图 6 示出了增量分页器如何将事务 1 200 提交给数据库。由于事务指针 320 通过写缓冲区 510、410 和 310，或指向与当前状态指针 120 所指向的相同的状态，因此增量分页器确定在事务 1 200 被提交给数据库之前没有介入事务改变了数据库的当前状态。由此，增量分页器通过将当前状态指针 120 改为指向为事务 1 200 创建的部分映射来将事务 1 200 提交给数据库，其中该部分映射包括写缓冲区 310、410 和 510 修改的原始状态 110。由此，增量分页器将当前状态指针 120 改为指向第三写缓冲区 510，即事务 1 200 添加的最后一个写缓冲区。增量分页器然后删除事务指针 320。当前状态指针 120 现在指向数据库的经更新的状态，该经更新的状态包括由储存在写缓冲区 310、410 和 510 中的改变更新的原始状态 110。

使用图 6 的当前状态，当一事务访问数据库时，增量分页器使用在启动该事务时存在的数据库的状态，这由当前状态指针 120 来标识。使用当前状态，如果一事务试图标识缓冲区 510 中地址 4 的内容，则它将找到串“DOG”。另一方面，如果该事务寻找内容地址 2，则它不会在缓冲区 510 中找到。由此，事务将前进到缓冲区 410，在那里事务将找到值 1。在图 6 的示例中，如果事

务寻找另一值，则事务将访问所有缓冲区，直到发现原始状态 110 仅包括对于其它地址的空值。由此，增量分页器通过向数据库的当前状态添加缓冲区或其它对象来改变数据。

增量分页器的一个恒定性是它遵守其所创建的每一指针的不变性。该不变性恒定性维持了每一对象所依赖的，并且因此所指向的状态永不改变，只要有一指针指向该对象。例如，第一写缓冲区 310 指向数据库的原始状态 110。不论事务 1 200 应用于数据库的后续改变如何，第一写缓冲区 310 都始终指向击仅包括原始状态 110 的数据库状态。类似地，第二写缓冲区 410 指向第一写缓冲区 310。第二写缓冲区 410 还指向数据库的不变状态，包括原始状态 110 和写缓冲区 310，这不会修改原始状态 110。第三写缓冲区 510 指向第二写缓冲区 410，由此指向其自己的数据库的局部映射，包括如由第二写缓冲区 410 修改的原始状态 110。这些指针的不变性具有以下描述的优点。

累积写缓冲区和重写单写缓冲区

在刚才描述的模式中，增量分页器创建一新的写缓冲区以启动事务并储存该事务试图应用于数据库的每一改变。或者，增量分页器可采用诸如累积缓冲区等不同类型的对象，或者增量分页器可盖写现有缓冲区以储存一个或多个改变。

图 7 示出了增量分页器使用累积写缓冲区来处理事务。如图 3 中的示例一样，增量分页器通过创建指向当启动事务时存在的当前状态（如由当前状态指针 120 所指示的）的第一累积缓冲区 710 来启动事务 1 200。增量分页器再次创建事务指针 320 并将其指向最近创建的写缓冲区以维护将被事务 1 200 修改的数据库的部分映射。

图 8 示出了增量分页器通过添加指向第一累积缓冲区 710 的第二累积缓冲区 810 来继续处理事务。增量分页器将事务指针 320 指向最近创建的写缓冲区，即第二累积缓冲区 810。增量分页器将事务 1 200 试图应用于数据库的改变储存在第二累积缓冲区 810 中，并将地址 2 的值设为 1。第二累积缓冲区 810 不不表现为累积的，因为事务 1 200 迄今仅尝试了一次改变。

图 9 示出了增量分页器通过添加指向第二累积缓冲区 810 的第三累积缓冲

区 910 来继续处理事务 1 200。增量分页器将事务指针 320 指向最近创建的写缓冲区，即第三写缓冲区 910。增量分页器现在储存事务 1 200 试图应用于数据库的改变，并将地址 2 的值设为 1，并且还储存地址 4 处的串“DOG”。

图 10 和 11 示出了增量分页器使用累积写缓冲区的一种模式的优点。图 10 示出了当事务 1 200 尝试的所有改变都被储存在第三累积缓冲区 910 中时，事务 1 200 将应用于数据库的所有改变的部分映射被包括在单个对象，即累积写缓冲区 910 中。该部分映射即使在如图 10 的示例中那样省略累积缓冲区 710 和 810 时也将被保留。

根据不变性恒定性，没有事务可以干涉或访问一事务的中间状态，因此，不需要保留写缓冲区 710 和 810 来维持事务指针 320 所指示的状态的完整性。结果，在其所储存的改变也被储存在最近创建的累积缓冲区 910 中之后，不需要维护累积缓冲区 710 和 810。

由此，如图 11 所示，当增量分页器创建并写入每一新的累积缓冲区时，增量分页器可丢弃之前的缓冲区。第一阶段 1100 示出在第一改变之后的事务 1 200 的当前状态已被写入第二累积缓冲区 810 中，其中地址 2 的值被设为 1。由于第二累积缓冲区 810 包括储存在第一写缓冲区 710 中的任何改变，因此在第二阶段 1110，增量分页器可以将第二累积缓冲区 810 指向第一累积缓冲区 710 所指向的状态，并释放第一累积缓冲区 710 及其指针而不改变由事务指针 320 所指示的状态。事务指针 320 所指向的状态更短，因为它不通过仅包括储存在第二累积缓冲区 810 中的相同数据的一个子集的缓冲区。在第三阶段 1120 中，增量分页器可以释放用于储存第一累积缓冲区 710 的存储器，从而节省了存储器用于其它用途。如果增量分页器应用了如图 11 所示的类似阶段来储存事务 1 200 的下一改变，则结果将与图 10 所示的相同。

图 12 示出了增量分页器用于处理事务的一种模式。流程图 1200 在框 1202 接收事务开始。框 1204 创建将指向该事务试图应用于数据库的任何改变的事务指针。框 1206 创建第一写缓冲区，并将该缓冲区指向接收到该事务时存在的当前状态。框 1208 将事务指针指向第一写缓冲区，第一写缓冲区进而指向启动该事务时的当前状态。

框 1210 确定事务是否试图向数据库应用改变而非例如仅仅从数据库读取

数据。如果是，则框 1212 创建一附加写缓冲区并将其指向先前创建的缓冲区。该附加写缓冲区可以是单写缓冲区或累积写缓冲区。如果使用了累积写缓冲区，则在一种模式中，框 1212 将该附加累积缓冲区指向前一缓冲区所指向的状态，并如参考图 11 所述地释放前一缓冲区。

框 1214 将事务试图应用于数据库的改变储存在该附加写缓冲区中。框 1216 将事务指针指向该附加写缓冲区。框 1218 确定该事务是否试图向数据库应用任何另外的改变。如果是，则流程图 1200 循环到框 1212，并且框 1212 创建一附加缓冲区。另一方面，如果框 1218 确定该事务没有试图应用另外的改变，则框 1220 将如在下一节中所述地提交改变。框 1224 等待另一事务的接收。

另一方面，如果框 1210 确定该事务没有试图向数据库应用任何改变，则框 1222 释放第一写缓冲区和事务指针，因为它们不再需要。框 1224 再次等待另一事务的接收。

增量分页器还可如图 13 所示使用单个可重写写缓冲区来代替使用一系列累积缓冲区。在第一阶段 1300，增量分页器接收事务并创建单写缓冲区 1310，并将其指向当前状态。增量分页器然后创建切换事务指针 1320 并将其指向单写缓冲区 1310。切换指针 1320 如下所述保持了增量分页器中的指针的不变性。以此方式盖写或重写缓冲区通过使用单个累积缓冲区来储存事务试图应用的改变而非使用多个缓冲区来储存每一改变保留了增量分页器所识别的不变性恒定性。

在使用单写缓冲区 1310 的第二阶段 1330，增量分页器将切换指针 1320 指离单写缓冲区 1310，并用事务所指示的改变或另外的改变来更新单写缓冲区。如上所述，增量分页器保留所有指针的不变性。由此，在技术上，如果切换指针在它向缓冲区 1310 写入改变时继续指向单写缓冲区 1310，则将违反该不变性恒定性：切换指针指向的状态将看到变化。由此，在一种模式中，增量分页器使用它能指离单写缓冲区 1310 的切换指针 1320。

在第三阶段 1340，一旦增量分页器更新了单写缓冲区，则增量分页器将切换指针 1320 指回单写缓冲区 1310 以维持事务试图应用于数据库的改变的部分映射。在第四阶段 1350，增量分页器再次将切换指针 1320 指离单写缓冲区

1310，并更新单写缓冲区以添加事务指示的另一改变。在第五阶段 1360，一旦增量分页器向单写缓冲区添加了另外的改变，该增量分页器再一次将切换指针 1320 指回单写缓冲区 1310 以维持部分映射。

注意，该增量分页器不需要字面上使用实际被指离缓冲区 1310 的切换指针 1320。为示出逻辑上保持了不变性恒定性，该增量分页器被描述为使用由切换指针 1320 指示的可重写缓冲区。或者，如果重写原子地发生使得在重写发生时系统中没有一部分解除对指针的引用，则可以改为使用普通的事务指针 320，以便仍有效地保持恒定性：如果没有人观察到其值，则一指针不是指针。因此，在这一受限情况下，增量分页器使用原地重写，因为它保持了不变性恒定性。

图 14 示出了图 13 中所描绘的增量分页器用于通过使用单写缓冲区来处理事务的模式。流程图 1400 在框 1402 接收事务时开始。框 1404 创建将指示事务试图应用于数据库的改变的部分映射的切换指针。框 1406 创建单写缓冲区并将该缓冲区指向接收事务时的当前状态。框 1408 将切换指针指向单写缓冲区以指示该事务试图应用于数据库的部分映射。

框 1410 确定该事务是否试图向数据库应用改变。如果是，则框 1412 将切换指针指离单写缓冲区。框 1414 将改变储存到单写缓冲区中。如图 13 所示，该单写缓冲区是累积的，因此，可向该单写缓冲区添加多个改变。框 1416 将切换指针指回单写缓冲区。

框 1418 确定该事务是否试图向数据库应用任何另外的改变。如果是，则流程图 1400 循环到框 1412，并且框 1412 创建一附加缓冲区。另一方面，如果框 1418 确定该事务没有试图应用另外的改变，则框 1420 将试图如在下一节中所描述的那样提交改变。框 1214 等待另一事务的接收。

另一方面，如果框 1410 确定该事务没有试图向数据库应用任何改变，则框 1422 释放单写缓冲区和切换指针，因为它们不再需要。框 1424 等待另一事务的接收。

提交事务

增量分页器支持用于确定何时提交事务的不同模式。增量分页器允许多个

事务并发地访问启动每一事务时存在的数据库状态。然而，当多个事务并发地运行时，第一个被提交给数据库的事务可改变其它并发执行的事务所依赖的数据库状态。基于过时的或被取代的数据库状态的事务将导致在被提交给数据库时的无效结果。由此，增量分页器仅在介入事务没有干涉当前事务所依赖的数据库状态时才提交当前事务。

图 15 示出了增量分页器与事务 1 200 同时地处理的第二事务，即事务 2（Tx2）1500，这导致可能冲突的事务。面对可能冲突的事务，增量分页器确定要将哪一事务提交给数据库并中止哪一事务。事务 2 1500 调用多个指令 1510-1570。诸如指令 1 1520“x=Read(0)”等某些指令试图从数据库中读取数据，而诸如指令 4 1550“Write(3, "CAT")”等某些指令试图向数据库应用改变。

图 16 示出了其中增量分页器接收并并发地处理两个事务 1 200 和事务 2 1500 的情形。这两个事务中的任一个都可在另一个之前、在任一事务被提交给数据库之前启动。这两个事务在当前状态指针 120 继续指向原始状态 110（此时是当前状态）时启动。

增量分页器如上所述地通过创建指向由当前状态指针 120 指向的原始状态 110 的第一写缓冲区 310 来启动事务 1 200，并创建指向对该事务最近创建的写缓冲区的事务指针 320。增量分页器创建储存事务 1 200 试图对数据库作出的改变的附加写缓冲区 410 和 510。

对于事务 2 1500，增量分页器创建指向原始状态 110 的第一写缓冲区 1610 以及指示对该事务最近创建的写缓冲区的事务指针 1620。类似地，增量分页器为事务 2 1500 创建附加写缓冲区。在图 16 中，指令 1 1520“x=Read(0)”和指令 2 1530“y=Read(1)”都从原始状态 110 读取空值。由此，条件指令 3 1540“If y=x then”为真，并且指令 4 1550“Write(3, "CAT")”执行。增量分页器创建指向第一写缓冲区 1610 的附加写缓冲区 1630，将指令 4 1550 指示的改变储存在附加写缓冲区 1630 中，并将事务指针 1620 指向附加写缓冲区 1630。事务指针 320 和事务指针 1620 都指向包括每一事务将应用于数据库的改变的部分映射。

事务 1 200 和事务 2 1500 可能对数据库作出冲突的改变，或者诸如指令 3 1540 等条件指令可能依赖于另一事务可能会改变的数据。以下描述增量分页器

以不同方式解决这些潜在冲突的模式。

当介入事务改变当前状态时中止事务

图 17 和 18 示出了增量分页器用于处理潜在的冲突事务的一种模式：通过在增量分页器提交了改变数据库状态的介入事务之后中止一事务。在图 17 中，增量分页器已如上对于图 6 所描述的那样将事务 1 200 提交给数据库：事务指针 320 被移除，并且当前状态指针 120 被切换为指向为事务创建的最后一个写缓冲区 510。在将事务 1 200 提交给数据库之后，增量分页器确定事务 2 1500 已完成执行并试图将其提交给数据库。

在确定是否将事务 2 1500 提交给数据库时，增量分页器确定诸如事务 1 200 等介入事务是否改变了在当前事务，即事务 2 1500 启动时存在的数据库状态。为作出这一判定，增量分页器通过确定第一写缓冲区 1610 指向什么状态来比较事务 2 1500 启动时的数据库状态。增量分页器然后将其与当前状态指针 120 所指向的数据库的当前状态进行比较。换言之，增量分页器将当前状态指针 120 与事务指针 1620 和插入的写缓冲区 1610 和 1630 进行比较以确定它们是否都指向同一对象。在将事务 1 200 提交给数据库之后，增量分页器确定当前状态指针 120 指向写缓冲区 510，而事务 2 1500 通过其第一写缓冲区 1610 指向原始状态 110。

由于事务 2 1500 指向除当前状态指针 120 指示的当前状态之外的其它状态，因此增量分页器中止事务 2 1500。图 18 示出了增量分页器通过删除写缓冲区 1610 和 1630 以及事务指针 1620 来释放不再成为数据库的当前状态的一部分的存储器而中止事务 2 1500。

中止对于当前状态的任何改变的事务是避免提交冲突事务的一种抗风险方法。如在图 16-18 的示例中那样，事务 1 200 不改变当执行事务 2 1500 时存在的数据库状态中的任何值。然而，有可能介入事务 1 200 已经改变了事务 2 1500 所依赖的数据。事务 1 200 改变了数据库的当前状态，且因此可能已改变了事务 2 1500 所依赖的数据。

尽管事务 2 1500 被中止，但该事务可以如图 19 和 20 所示地重新启动。在图 19 中，增量分页器在提交事务 1 200 之后启动事务 2 1500。由此，当事务

2 1500 被启动时，增量分页器创建了指向在重新启动事务 2 1500 时存在的状态的第一写缓冲区 1910。该状态由当前状态指针 120 来指示，该指针指向由事务 1 200 添加到状态的写缓冲区 510。增量分页器然后创建最初指向写缓冲区 1910 的事务指针 1920。增量分页器添加增量分页器向其储存事务 2 1500 试图应用于数据库的改变的附加写缓冲区 1930，然后增量分页器将事务指针 1920 改为指向附加写缓冲区 1930 以指示表示事务 2 1500 试图应用于数据库的改变的部分映射。

此时，当增量分页器确定事务 2 1500 完成并且试图将事务提交给数据库时，增量分页器发现没有介入事务将数据库的当前状态从事务 2 1500 所依赖的状态改变。事务指针 1920 通过写缓冲区 1930 和 1910 指向写缓冲区 510。当前状态指针仍指向写缓冲区 510，因此，事务指针 1920 仍指向当前状态。

图 20 示出了增量分页器将事务 2 1500 提交给数据库。增量分页器将当前状态指针 120 改为指向事务 2 1500 添加的最后一个缓冲区，即写缓冲区 1930。增量分页器然后删除事务指针 1920。当前状态指针现在所指向的当前状态包括被储存在写缓冲区 410 和 510 中的事务 1 200 所应用的改变以及储存在写缓冲区 1930 中的事务 2 1500 所应用的改变所修改的原始状态 110。

图 21 示出了增量分页器用于确定是提交还是中止事务的该第一模式。流程图 2100 在框 2102 处开始。框 2102 确定一完成的事务何时试图提交给数据库，并且流程图 2100 循环到框 2102 直到一事务试图提交给数据库。当框 2102 检测到一完成的事务试图提交时，框 2104 确定当前状态是否与如由用于当前事务的事务指针所指示的当启动事务时存在的状态相同。换言之，框 2104 确定为事务创建的第一写缓冲区是否指向当前状态指针所指向的同一状态。如果是，则框 2106 将当前状态指针指向为该事务创建的最后一个缓冲区，这是事务指针也指向的缓冲区，从而完成作为提交事务的结果的对数据库的更新。框 2108 删除事务指针，从而释放相关联的存储器。流程图 2100 循环到框 2102 以检测下一完成的事务试图提交给数据库。

另一方面，如果框 2104 确定当前状态不与当启动事务时存在的状态相同，则框 2104 确定当前状态已改变。框 2110 释放事务的写缓冲区。框 2112 删除事务指针，完成事务的中止。如上所述，中止的事务可以被重新启动，并且在

事务完成并试图提交时，增量分页器将确定该事务是否应被提交给数据库。

防止事务被不必要地中止的读缓冲区

代替增量分页器只要当前事务指向不匹配当前状态的状态就中止当前事务，增量分页器可以改为在介入事务写入了当前事务已读取的数据时中止当前事务。只要当前状态已改变就中止当前事务确保不会将冲突的改变应用于数据库。然而，当可能存在同时执行的许多事务，并且这些事务中的某一些可能涉及冗长或复杂的计算时，中止事务浪费了计算资源。由此，如果介入事务没有访问或改变当前事务读取或依赖的任何数据，增量分页器就不必中止该当前事务。

一种模式的增量分页器为每一事务创建一读缓冲区。读缓冲区跟踪事务访问的任何数据。当完成的事务视图提交给数据库，但发现一介入事务已经改变了该数据库的状态时，增量分页器可以将读缓冲区与该数据库的当前状态进行比较以确定是否有必要中止当前事务。在另一种模式中，增量分页器在当前状态已改变并且介入事务已改变或盖写了当前事务读取的数据时中止当前事务。在此模式中，增量分页器可以在介入事务向当前事务读取的地址写入数据时中止事务，或者增量分页器可以将当前事务读取的实际数据与介入事务写入的数据进行比较以确定该数据值是否实际改变。

图22-24示出了使用读缓冲区并且在介入事务写入了当前事务读取的数据时中止事务的一种模式的增量分页器。图 22 示出了针对数据库的当前状态执行的两个事务，即事务 1 2200 和事务 2 2250。当前状态指针 120 指向原始状态 110，由此指示当前状态或数据库的当前状态。对于事务 1 2200，增量分页器创建读缓冲区 2210、写缓冲区 2220 和事务指针 2230。如上所述，增量分页器可以对每一事务创建一个或多个写缓冲区。如果读缓冲区的大小有限制或者出于其它原因，增量分页器还可创建多于一个读缓冲区。增量分页器将读缓冲区 2210 指向当前状态，将写缓冲区 2220 指向读缓冲区 2210，并将事务指针 2230 指向写缓冲区 2220。相应地，对于事务 2 2250，增量分页器创建读缓冲区 2260、写缓冲区 2270 和事务指针 2280。增量分页器将读缓冲区 2260 指向当前状态，将写缓冲区 2270 指向读缓冲区 2260，并将事务指针 2280 指向写缓冲区 2270。

用于事务 1 2200 的读缓冲区 2210 指示事务 1 2200 已经读取了地址 3 处的数据。用于事务 2 2250 的读缓冲区 2260 指示事务 2 2250 也已经读取了地址 3 处的数据。然而，即使每一事务读取了相同的数据，也没有一个事务试图改变该地址处的数据。由此，增量分页器并没有被迫中止任一事务。

与空的写缓冲区一样，读缓冲区不改变数据库的状态。因此，增量分页器的各实施例可采用不出于数据库的当前状态的读缓冲区，就如同它曾经是其中储存了数据的对象一样。相反，增量分页器可以被储存在该增量分页器可创建以储存事务试图应用的改变的部分映射之外。读缓冲区可以被储存在一单独的位置中，并且事务指针可以维护指示该读缓冲区的位置的一单独指针。

如果增量分页器仅在介入事务改变了当前事务读取的地址处的数据时才中止当前事务，则读缓冲区应当储存所读取的地址和值，如以下参考图 25-27 所描述的。另一方面，如果增量分页器将在介入事务盖写了当前事务曾读取的数据时中止该事务，则即使介入事务写了相同的值，读缓冲区也只需包括该事务所读取的一个或多个地址，如在图 22-24 中所示。

图 23 示出了在增量分页器向数据库提交了事务 1 2200 之后数据库的状态。增量分页器将当前状态指针 120 改为指向为事务 1 2200 创建的最后一个写缓冲区 2220，并删除事务指针 2230。一旦事务 2 2250 完成，增量分页器就确定它是否应提交事务 2 2250。再一次，由于数据库的当前状态在增量分页器提交事务 2 2250 之前改变，因此在如先前参考图 15-21 所描述的确定是否提交事务的模式中，增量分页器将中止事务 2 2250。相反，当前模式作出更实质性的评估。

在此模式中，即使数据库的当前状态已改变，增量分页器也不中止当前事务，除非介入事务将数据写入了当前事务所访问的地址。增量分页器比较事务 2 2250 的读缓冲区 2260 以确定介入事务，即事务 1 2200 是否写入了事务 2 2250 从中读取数据的地址，由此改变了处理或事务 2 2250 所依赖的状态。事务 2 2250 根据读缓冲区 2260 仅从地址 3 读取数据。然而，检查了当前状态指针 120 指示的当前状态，事务 1 2200 根据写缓冲区 2220 仅将数据写入地址 4。由此，增量分页器将事务 2 2250 提交给数据库。

图 24 示出了增量分页器将事务 2 2250 提交给数据库的当前状态。增量分

页器将事务 2 2250 的读缓冲区 2260 指向当前状态指针所指示的对象，即事务 1 2200 的写缓冲区 2220。增量分页器然后将当前状态指针 120 改为指向事务 2 2250 的写缓冲区 2270，从而将事务 2 2250 提交给数据库。

图25-27示出了其中增量分页器仅在介入事务不仅写入了介入事务写入数据的地址，而且介入事务还实际改变了当前事务读取的数据值时才中止当前事务的较不抗风险的模式。图 25 示出了针对数据库的当前状态执行的两个事务，即事务 1 2500 和事务 2 2550。当前状态由当前状态指针 120 指示，该指针指向数据库的原始状态 110。对于事务 1 2500，增量分页器创建读缓冲区 2510、写缓冲区 2420 和事务指针 2530。增量分页器将读缓冲区 2510 指向那时的当前状态，将写缓冲区 2520 指向读缓冲区 2510，并将事务指针 2230 指向写缓冲区 2520。相应地，对于事务 2 2550，增量分页器创建读缓冲区 2560、写缓冲区 2570 和事务指针 2580。增量分页器将读缓冲区 2560 指向那时的当前状态，将写缓冲区 2570 指向读缓冲区 2560，并将事务指针 2580 指向写缓冲区 2570。

与图 22-24 的示例形成对比，用于事务 1 2500 的读缓冲区 2510 指示事务 1 2500 读取了地址 3 的值，并且从地址 3 读取的值为空。用于事务 2 2550 的读缓冲区 2560 指示事务 2 2500 也读取了地址 3 的值，并且也发现从地址 3 读取的值为空。用于事务 2 2550 的写缓冲区 2570 储存对地址 3 的值的改变，从而将储存的值改为串“CAT”。

图 26 示出了在增量分页器向数据库提交了事务 2 2550 之后数据库的当前状态。因此，增量分页器将当前状态指针 120 改为指向储存事务 2 2550 应用于数据库的任何改变的写缓冲区 2570，并丢弃事务指针 2580。在此模式中，事务 1 250 一旦完成，增量分页器确定事务 1 2500 是否读取事务 2 2550 改变的数据。

增量分页器将事务 1 2510 的读缓冲区 2510 与当前状态指针 120 指示的数据库的当前状态进行比较。根据读缓冲区 2510，事务 1 读取了地址 3 并发现值为空。增量分页器还跟随当前状态指针 120 指示的数据库的当前状态，并发现事务 2 2550 将地址 3 处储存的值改为“CAT”。由此，事务 1 2500 试图应用于数据库的任何改变可以基于现在过时的数据。

因此，增量分页器如图 27 所示地中止事务 1 2500。增量分页器释放或删

除读缓冲区 2510、写缓冲区 2520 和事务指针 2530，并且增量分页器保持当前状态指针 120 指向储存事务 2 2550 应用于数据库的改变的写缓冲区 2570。如有需要，事务 1 2200 随后可被如上所述地重新启动。

注意，即使事务 2 2550 向事务 1 2500 读取的地址写入了新值，事务 2 2250 写入的数据也可能没有改变事务 1 2500 的结果。然而，为确定因事务 2 2550 写入的数据而导致的事务 1 2500 将改变的结果，事务 1 2500 必须被重新执行。这一模式的增量分页器不中止事务 1 2500 仅仅是因为当前状态已改变，但是提供了对它是否应当中止缺少时间来重新运行事务 1 2550 的当前事务的某一实质性分析。

图 28 示出了增量分页器用于通过创建读缓冲区和单写缓冲区来处理事务以便于确定增量分页器是否应提交事务的更实质性模式的一种模式。增量分页器不限于结合使用读缓冲区来使用单写缓冲区，并且单写缓冲区的选择仅是一种可能的替换。

流程图 2800 在框 2802 接收事务时开始。框 2804 创建将指示该事务试图应用于数据库的改变的部分映射的切换指针。框 2806 创建一读缓冲区并将其指向启动事务时存在的状态。框 2808 创建单写缓冲区并将该单写缓冲区指向读缓冲区。框 2810 将切换指针指向单写缓冲区以指示该事务试图应用的改变的部分状态。

框 2812 将事务读取的任何数据储存在读缓冲区中。如上所述，在其中增量分页器可在介入事务读取了与当前事务相同的数据的时候中止事务的模式中，读缓冲区只需储存所读取的数据的地址。另一方面，如果增量分页器基于介入事务是否写入了由当前事务读取的数据来确定是否提交该事务，则读缓冲区应储存地址和从该地址读取的数据两者。

框 2814 确定事务是否试图向数据库应用改变。如果是，则框 2816 将切换指针指离单写缓冲区。框 2818 将改变储存到单写缓冲区中。框 2820 将切换指针指回单写缓冲区。

框 2822 确定事务是完成还是继续执行。如果事务未完成，则流程图 2800 循环到框 2812 以将事务读取的任何数据储存在读缓冲区中。另一方面，如果框 2822 确定事务完成，则框 2824 将试图提交事务作出的改变。框 2826 等待

另一事务的接收。

图 29 示出了其中增量分页器使用读缓冲区来确定是提交还是中止事务的一种模式。流程图 2900 在框 2902 处开始。框 2902 确定完成的事务何时试图向数据库提交，并且流程图 2900 循环到框 2902，直到一事务试图提交到数据库。当框 2902 检测到试图提交的已完成事务时，框 2904 确定当前状态自从该事务被启动以来是否已改变。

如果当前状态未改变，则框 2912 将当前状态指针指向为该事务创建的单写缓冲区，或多个写缓冲区中的最后一个，这是事务指针也指向的缓冲区。框 2914 删除事务指针，从而完成数据库的更新。如果数据库的当前状态未改变，则增量分页器可以在不检查读缓冲区或搜索对数据库的应用的情况下提交事务。如果当前状态未改变，则没有介入事务将改变当前事务所依赖的数据。流程图 2900 循环到框 2902 以检测试图提交到数据库的下一已完成事务。

另一方面，如果框 2904 确定当前自从事务被启动以来已改变，则框 2906 确定读缓冲区中列出的数据是否在当前状态中改变。再一次，这一模式的增量分页器在当前状态因增量分页器提交介入事务而已改变时仅检查读缓冲区。如果读缓冲区中列出的数据在当前状态中尚未改变，则增量分页器将提交该事务并且流程图 2900 前进到框 2912。

另一方面，如果框 2906 在当前状态中发现数据已被写入读缓冲区中的地址，或者如果该地址处的数据的值不同于储存在读缓冲区中的值，则增量分页器中止该事务。框 2908 释放用于该事务的读和写缓冲区。框 2910 然后释放或删除用于当前事务的事务指针，从而完成事务的中止。流程图 2900 然后循环到框 2902 以等待试图提交的下一事务。

接合事务并保留状态

增量分页器的写缓冲区提供了许多好处。举一个例子，当事务不能被提交并且必须被中止时，增量分页器忽略或删除它为该事务创建的写缓冲区，并且增量分页器不需要重写或撤消否则可能被应用于数据库的错误的或冲突的改变。然而，在增量分页器向当前状态添加许多写缓冲区时，对当前状态中的所有数据的访问可能变得低效。即使在图 20 的相对简单的示例中，事务也有可

能必须通过写缓冲区 1930、1910、510、410、310 以及原始状态 110 读回，以确定事务试图读取的数据的值。一种模式的增量分页器接合写缓冲区以确保对数据库的当前状态的高效访问。

图 30 示出了增量分页器如何接合图 20 的写缓冲区的一种模式。在图 30 中，增量分页器通过向数据库的原始状态 110 应用储存在写缓冲区中的改变来接合写缓冲区。在第一阶段 3000，增量分页器识别指示写缓冲区将被串接到的指针的串接指针 3010。串接指针 3010 的好处将在以下进一步描述。增量分页器然后应用储存在指向原始状态 110 的第一写缓冲区中的改变。如图 30 所示，第一写缓冲区是空的写缓冲区 310。增量分页器通过从当前状态中省略空的写缓冲区来接合诸如写缓冲区 310 等空缓冲区。

注意，空的写缓冲区 310 不改变标识事务试图应用于数据库的改变的部分映射。由此，一旦增量分页器创建了储存事务试图应用的改变的写缓冲区，就可立即丢弃空的写缓冲区，或者增量分页器可能甚至不创建空的写缓冲区 310。例如，一旦写缓冲区 510 被添加以改变地址 4 处的值，则增量分页器的一个实施例可能将写缓冲区 510 指向启动事务时存在的状态，并丢弃空的写缓冲区 310。

在第二阶段 3020，增量分页器将储存在下一写缓冲区 410 中的改变应用于原始状态 110，从而用经更新的状态 3030 来替换原始状态。储存在写缓冲区 410 中的、将地址 2 处的值设为 1 的改变被应用于地址 2 3040 以创建经更新的后备存储 3030。

最终结果 3050 示出数据库的新状态 3060。在新状态 3060 中，应用储存在写缓冲区 510 中的改变以将储存在地址 4 3070 处的值改为串“DOG”，并且应用储存在写缓冲区 1930 中的改变以将储存在地址 3 3080 处的值改为串“CAT”。增量分页器现在切换了当前状态指针 120 以指向新状态 3060。增量分页器然后省略接合指针 3010。

关于增量分页器的缓冲区接合有四点要注意。首先，即使在接合过程期间，每一指针也继续不变地指向数据库的同一状态。例如，在第二阶段 3020，当增量分页器将写缓冲区 410 中的改变应用于原始状态 110 以创建经更新的状态 3030 时，由其它指针指示的状态保持相同。当前状态指针 120、写缓冲区 510、

1910 和 1930 以及接合指针 3010 都仍指向呈现所有相同值的数据库状态。

第二，如果启动了一个或多个新事务并将其提交以改变数据库的当前状态，则新事务将继续指向数据库的不变状态。在启动新事务时，增量分页器创建指向由当前状态指针 120 指示的当前状态。如果增量分页器提交该事务，则增量分页器切换当前状态指针 120 以指向为每一后续事务创建的最后一个写缓冲区。由此，即使在接合写缓冲区时，数据分页器也保留了新事务的状态的不变性。

第三，接合指针 3010 指示接合过程将停止的点。当增量分页器将附加写缓冲区提交给当前状态时，增量分页器将不接合附加缓冲区，直到增量分页器参与了进一步的接合。接合可以无限制地继续，但是将计算资源专用于不断地接合可能仅仅是几个新的写缓冲区并不是对计算资源的一种高效使用。

第四，如图 31 所示，增量分页器可以接合缓冲区而不盖写数据库的原始状态 110。例如，原始状态 110 可以被储存在盘存储中，而其访问可能是低效的，或者可能希望维持原始状态 110 而没有后续事务应用的改变。因此，增量分页器可将写缓冲区接合到一个或多个中间对象。

在图 31 中，在将写缓冲区接合到中间对象的第一阶段 3100，增量分页器在要接合的第一个写缓冲区，即写缓冲区 310 和原始状态 110 之间创建一中间接合对象 3110。插入接合对象 3110 不改变数据库内的任何当前状态的不变性：当前状态指针 120 和写缓冲区 310、410、510、1910 和 1930 仍指向数据库的同一状态。接合对象 3110 以及原始状态 110 可以在任何期望的存储中维护，包括高速缓冲存储器、存储器、盘或其它形式的存储。

最终结果 3120 示出增量分页器已将储存在写缓冲区 410、510 和 1930 中的所有改变接合到接合对象 3110。由此，储存在写缓冲区 410 中的将地址 2 处的值改为 1 的改变并非被应用于原始状态 110，而是被储存在地址 2 3130 处的接合对象 3110 中。类似地，增量分页器将来自写缓冲区 510 和 1930 的改变分别储存在地址 3 3140 和地址 4 3150 处的接合对象 3110 中。增量分页器将当前状态指针 120 改为指向接合对象 3110，接合对象 3110 进而指向原始状态 110。由此，数据库的状态从不改变，并且当前状态指针 120 继续指向在增量分页器将写缓冲区接合到接合对象 3110 之前存在的同一数据库状态。

有利的是，试图读取数据的新事务现在只需检查两个数据存储：接合对象3110和原始状态110。如果后者是需要的，则增量分页器可以将接合对象3110的内容接合到原始状态以创建经更新的后备存储。或者，增量分页器可以维持接合对象，并执行写缓冲区到同一接合对象3110的后续接合。再或者，增量分页器可以如以下对于快照进一步描述地创建附加的接合对象。

如上所述，增量分页器维持指针的不变性。在数据库内的接合对象中，表示在添加每一对象时存在的中间状态的缓冲区之间的指针被消除，只要保留了当前状态。然而，如果除来自后续缓冲区的指针之外的其它指针指向一缓冲区，则增量分页器识别的不变性确保该指针指示的状态必须经受得住缓冲区接合。利用增量分页器的指针不变性，增量分页器将遵守指向所选状态的快照指针，并且即使在增量分页器接合快照指针之前或之后的缓冲区时也保留该状态。

图32示出了增量分页器将随快照指针3210一起保留的数据库的当前状态。当前状态指针120指向当前状态，其中原始状态110被写缓冲区3220、3230和3240修改。第一写缓冲区3220将串“PEAR”写入地址0。第二写缓冲区3230将值1写入地址2。第三缓冲区3240将串“DOG”写入地址4。

当增量分页器被指示保留当前状态时，增量分页器创建指向当前状态的快照指针3210。增量分页器插入快照指针3210以保留例如当自动保留常规备份的程序或用户请求请求诸如当前状态等状态的快照时的状态。由于快照指针3210指向所选状态，因此由于增量分页器所遵守的指针不变性，增量分页器将维持该指针和相应的状态。

图33示出了在附加事务改变了当前状态之后数据库的较新状态。第四写缓冲区3310将串“PEACH”写入地址0。第五写缓冲区3320将串“CAT”写入地址4。增量分页器将当前状态指针120切换为指向第五写缓冲区3320。在图33中，第五写缓冲区3320用串“CAT”盖写写缓冲区3240储存在地址4处的串“DOG”。然而，快照指针3310仍指向表示图32的状态的写缓冲区3240。

图34示出了增量分页器所允许的接合。假设其中增量分页器不盖写原始状态110的情况，则增量分页器通过收集在快照指针3210和第一接合对象3410中的原始对象的后备存储之间写缓冲区3220、3230和3240应用的改变来接合

数据库的当前状态中的对象。增量分页器然后收集第二接合对象 3320 中写缓冲区 3310 和 3320 应用的改变。当前状态指针 120 指向第二接合对象 3420。第二接合对象 3420 指向第一接合对象 3410，后者进而指向原始状态 110。

增量分页器使用单独的接合对象 3410 和 3420 来保留快照对象 3210 的指针的不变性。只要快照对象 3210 存在，增量分页器就将维持图 32 的所选状态。由此，例如，在图 33 和 34 的当前状态中，增量分页器用写缓冲区 3320 储存在地址 4 处的串“CAT”来替换写缓冲区 3240 储存在地址 4 处的串“DOG”。然而，在快照指针 3210 保留的所选状态中，串“DOG”保持被储存在第一接合对象 3410 中的地址 4 处。

关于快照指针 3210 有五点要注意。首先，增量分页器可通过添加快照指针来保留任何所选时刻的数据库状态。其次，增量分页器可包括多个快照指针来保留多个状态。第三，如上对于接合所描述的，增量分页器可以继续接收事务并向数据库提交改变，而同时由于指针的不变性，仍保留一个或多个较早的状态。第四，当快照指针 3210 保留的状态不再需要时，移除快照指针 3210，并且后续接合将释放保留的状态。

第五，快照指针允许增量分页器在单个数据存储中维护多个不同的状态。当保存常规数据库的备份映像以维护当前状态时，每一映像被分开储存，因为后续映像会盖写先前的映像。多个状态可消耗大量的存储，并且备份映像通常被移交给较慢的存储设备以便节省较快的设备中的存储空间。然而，由于增量分页器在较早的状态上构建后续状态，因此快照指针在单个数据存储内保留较早的状态。

图 35 示出了增量分页器接合缓冲区或诸如先前创建的、现在将被进一步接合的接合对象等其它对象的一种模式。流程图 3500 以框 3502 开始。框 3502 启动对数据库当前状态中的对象的接合。一程序可不时地启动接合，或者用户可启动接合。框 3504 标识接合起始点，这可包括如图 34 的示例中的原始状态、最后接合对象、或增量分页器将从中接合添加到数据库的缓冲区的另一点。框 3506 前进到下一未接合对象。在图 33 的示例中，从原始状态 110 开始，框 3504 标识用于接合作为下一未接合对象，即写缓冲区 3220 的起始点。

框 3508 确定是否有除了来自下一个创建的缓冲区之外的指向缓冲区的任

何指针。当状态或数据库未改变时，增量分页器可以在接合一系列缓冲区或其它对象时消除指针，但是增量分页器不消除其它指针，诸如来自快照指针 3210 的指针。如果框 3508 确定没有除了来自下一缓冲区之外的其它指针，则框 3510 在接合对象中将该缓冲区与下一个创建的缓冲区相组合。框 3512 确定是否有可能要接合的另外的对象。另外的对象可包括下一个创建的缓冲区，或者如果数据库中的对象是先前接合的，则可能有要进一步接合的接合对象。如果有可能要接合的另外的对象，则流程图 3500 循环到框 3506 以便前进到下一未接合对象。然而，框 3512 可以确定没有其它对象要接合。例如，如果增量分页器已经完整地接合了数据库，则增量分页器将到达未接合的当前状态指针 120。当框 3512 确定没有要接合的另外的对象时，框 3514 完成接合。完成接合可能例如发送或记录确认操作完成的消息。

另一方面，框 3508 可确定有来自除状态中的下一对象之外的指针。例如，如图 32-34 所示，快照指针 3210 可以指向一对象以保留该对象所表示的状态。当框 3508 确定有另一指针时，流程图 3500 前进到框 3512 以确定是否有可能要接合的另外的对象。

使状态变得持久

增量分页器允许使数据库的所选部分变得持久。增量分页器将数据库的所选部分提交给非易失性存储以保留它们。

图 36 示出了数据库的当前状态，包括部分地储存在盘 3610，即持久介质，且部分地储存在存储器 3620，即易失性介质中的对象。当前状态指针 120 指向当前状态，该状态包括储存在盘 3610 上且增量分页器已向其提交了写缓冲区 3630 和 3640 中的改变的原始状态 110。

在一种模式中，增量分页器允许程序或用户通过插入持久指针 3650 来使数据库的所选部分变得持久。持久指针 3650 指向包括要变得持久的对象的数据库状态。在图 36 中，持久指针 3650 指向包括第一写缓冲区 3630 的状态。

如图 37 所示，增量分页器将包括在持久指针 3250 所指示的状态中的任何对象移至诸如盘存储 3710 等持久可存储介质。在本示例中，第一写缓冲区 3630 被移至盘存储 3710。增量分页器允许数据库状态驻留在任何数量的存储介质

上。由此，增量分页器向其写入第一写缓冲区 3730 的盘存储 3710 可以是与原始状态 110 所在的盘存储 3610 相同的存储设备，或者它可以是单独的存储设备。不包括在持久指针 3650 所指向的状态中的第二写缓冲区 3640 留在存储器 3620 中。

一旦增量分页器将持久指针 3650 所指示的数据库的当前状态的一部分储存到持久存储，则增量分页器可以释放持久指针。增量分页器可以生成确认数据库的所选部分已变得持久的消息。

图 38 示出了增量分页器使缓冲区变得持久的一种模式。流程图 3500 以框 3802 开始。框 3802 接收持久性请求。框 3804 定位持久性指针以确定要变得持久的数据库的部分。框 3806 标识持久性指针和要变得持久的数据库中的最后一个持久存储部分之间的所有对象。框 3808 将持久性指针和最后一个持久存储之间的所有对象复制到持久存储对象。框 3810 将指向要变得持久的当前状态的部分的任何指针改为指向持久存储对象。在图 37 的示例中，增量分页器将写缓冲区 3640 的指针改为指向持久对象。或者，例如，如果有指向持久指针 3650 所指向的状态的其它指针，诸如当前状态指针 120 或快照指针 3210，则这些指针也指向由框 3808 创建的持久对象。

一旦持久性指针 3650 所指向的状态中的对象被复制到持久对象并且改变了指向该状态的指针，框 3812 就释放持久性指针所指向的存储器中的对象以释放存储器用于其它用途。框 3814 删除该持久性指针。框 316 通过例如向作出持久性请求的用户发送消息或向系统日志添加消息来确认持久性请求的完成。

高速缓存数据

增量分页器还提供了对数据的高速缓存以改善性能。增量分页器允许诸如用参考图 7-9 描述的一系列累积缓冲区来对数据进行二重存储。一状态内的数据的二重存储不会改变该状态。由此，增量分页器可以将储存在诸如盘存储或其它大容量存储等低速存储中的数据高速缓存在诸如存储器等高速存储中，而不更改状态。

图 39 示出了包括高速缓存对象 3910 的数据库的当前状态。高速缓存对象

指针3910指向要高速缓存的对象，诸如储存在盘存储3950中的持久对象3930。增量分页器将未储存在持久对象 3930（此情况中为读缓冲区 3962）中的第一对象指向高速缓存对象 3910。

在当前状态中，如果一事务查找地址 0 的值，则该事务将在写缓冲区 3968 中找到地址 0 的当前值，该写缓冲区将串“CAT”写入当前状态。该事务不需要查看比写缓冲区 3968 远的地方，并且该事务将以存储器检索速度接收到地址 0 的值。另一方面，如果一事务寻找储存在地址 2 处的值，则增量分页器将贯穿当前状态来寻找该值直到找到它。如果没有高速缓存对象 3910，则增量分页器将去往盘存储 3950 中的持久对象 3930。访问来自盘存储 3950 的对象与访问存储器 3960 相比较慢。在存储器 3930 中创建高速缓存对象 3910 可提高访问效率。

在图 39 的示例中，持久对象 3930 储存一地址范围，诸如储存值 1 的地址 2 3932、储存串“CAT”的地址 3 3934、以及储存串“DOG”的地址 4 3936 处的值。持久对象 3930 还储存其中没有储存任何数据的一范围的空值 3940，直到储存串“ROCK”的地址 99 3938。

增量分页器在存储器 3960 中创建高速缓存对象 3910。增量分页器将高速缓存对象 3910 指向要高速缓存的对象，在此情况中是持久对象 3930。增量分页器然后将第一非持久对象，即读缓冲区 3962 指向高速缓存对象 3910。

一旦增量分页器创建了高速缓存对象并改变指针以在当前状态中插入高速缓存对象，增量分页器就填充该高速缓存。在一种模式中，当增量分页器从持久对象 3930 检索数据时，增量分页器将同一数据储存在高速缓存对象 3910 中。并且，在花费了时间来访问持久对象之后，增量分页器可以从持久对象 3930 检索一数据块并将其储存在高速缓存对象 3910 中。例如，在填充高速缓存对象 3910 之前，增量分页器为其创建了读缓冲区 3962 的事务寻找来自地址 4 的数据，该数据被储存在持久对象 3930 中地址 4 3936 处。在访问持久对象 3930 的同时，增量分页器还可检索包括来自地址 2 3932、地址 3 3934 和其它地址一直到地址 99 3938 的数据的块的数据，并将该数据储存在高速缓存对象 3910 中这些地址处。

在增量分页器将从持久对象 3930 中检索到的数据储存在高速缓存对象

3910 中之后，对复制到高速缓存对象 3910 的数据的后续请求将以快得多的存储器速度来执行。由此，如果为其创建了读缓冲区 3962 的事务接着请求来自地址 99 的数据，则增量分页器可以从高速缓存对象 3910 中地址 99 3918 处检索该数据。类似地，如果为其创建了读缓冲区 3964 的事务请求来自地址 3 的数据，则增量分页器可以从高速缓存对象中地址 3 3914 处检索该数据而无需等待增量分页器从盘存储 3950 检索数据。

关于增量分页器的高速缓存对象 3910 有三个特征要注意。首先，高速缓存对象 3910 的插入不改变增量分页器如何创建和使用数据状态。例如，即使高速缓存对象 3910 中的地址 4 3916 储存了串“DOG”，但写缓冲区 3966 仍可通过将串“BIRD”写入写缓冲区 3966 中的地址 4 来改变数据库的状态。在后续的事务中，当前状态指针 120 所指向的当前状态将发现串“BIRD”被储存在地址 4 处，而不论高速缓存对象 3910 在地址 4 3916 处储存了什么。

其次，如果使用高速缓存对象 3910 的状态变得持久，则增量分页器不需要重新启动并重新填充高速缓存对象 3910。相反，当数据被复制到持久存储时，当被复制到盘的状态中的数据不同于储存在高速缓存对象 2910 中的数据时，增量分页器可以更新高速缓存对象 3910 中的数据。或者，增量分页器可以使高速缓存对象 3910 中当与被复制到持久存储的状态相比时不再为当前的条目无效。这两种技术都修改了旧状态的高速缓存以使其变为经更新的状态的有效高速缓存。通过这样做，增量分页器无限制地重复使用了高速缓存对象。如果高速缓存不能跨更新而保留，则当更新变得持久时，重构高速缓存时将损失效率。

第三增量分页器可以使用高速缓存对象 3910 来便于范围查询。高速缓存对象可以通过向后备存储发出范围查询来高效地发现空值顺串，并且可在增量分页器维护的高速缓存对象或另一对象的一部分中紧凑地表示这一顺串。例如，空值顺串的表示可以被包括在高速缓存对象的一字段中。在储存在存储器中的对象中维护该表示将提高访问空值串的表示的速度以便于范围查询。例如，如果一事务希望找到在“BIRD”、“CAT”或“DOG”之后的下一宠物串，则当在持久对象 3930 中在“DOG”之后有一长串空数据 3940 时，高速缓存对象 3910 可包括诸如用于地址 5 3920 的第一空地址字段中的条目，指示该

顺串中的最后一个地址储存空值，或者第一个地址储存非空值以支持范围查询。

图 40 示出了增量分页器维护高速缓存对象的一种模式。流程图 4000 以框 4002 开始。框 4002 在存储器或其它快速访问存储中创建高速缓存对象。框 4004 将该高速缓存对象指向被高速缓存的对象。框 4006 标识指向被高速缓存的对象的对象，并将该对象指向高速缓存对象。框 4008 将事务读取的地址和数据从被高速缓存的对象储存到高速缓存对象中。如上所述，当增量分页器从被高速缓存的对象中检索数据时，增量分页器较佳地将从与所读取的地址相邻的地址块中检索数据，以便可能转移对后续盘访问操作的需求。

框 4010 确定指向该高速缓存对象的状态是否要变为持久。如果否，则流程图 4000 循环到框 4008 以继续将事务读取的地址和数据储存到高速缓存对象。然而，当框 4010 确定指向高速缓存的状态要变为持久时，框 4012 用变为持久的对象中的改变来更新高速缓存条目。或者，增量分页器可以使过时的高速缓存条目无效而非更新它们。一旦框 4012 更新了高速缓存条目，框 4014 就将未变得持久的第一个对象指向高速缓存对象。

增量分页器使用的对象

作为概述，图 41 示出了增量分页器用于方便先前描述的示例性操作的对象。使用新的参考标号来引用本概述中所表示的通用对象。图 41 示出了一个示例性情形 4100，包括储存在持久盘存储 4120 和存储器 4130 中的一范围的这些对象。

增量分页器可以使用盘存储 4120 中的一个或多个持久对象 4122 和 4124。这些对象可以储存如上所述变得持久的原始状态和任何其它状态。例如，持久指针 4126 被呈现为虚线轮廓，因为它在它所指向的、出现在持久对象 4124 中的状态变得持久之后被移除。持久对象可被储存在一个或多个设备上。

增量分页器可以包括储存在存储器 4130 中的多个对象。增量分页器 4140 可以包括插入在高速缓存的持久对象 4122 和 4124 与后续对象之间的高速缓存对象 4140。包括快照指针 4152 保留的状态的快照对象 4150 指向高速缓存对象 4140。快照对象 4150 包括所选状态，包括接合对象或一系列未接合对象。如

上所述，即使包括在保留状态中或稍后被添加到保留状态的对象被接合，快照指针 4152 也可以保留状态。

由于在接合对象 4160 处生成了最后一个接合操作，因此诸如读缓冲区 4170 和写缓冲区 4180 等多个缓冲区被添加到当前状态。当前状态指针 4190 指示当前状态。

用于实现示例性实施例的操作环境

图 42 示出了用于实现增量分页器的示例性操作环境 4200。操作环境 4200 仅为合适的操作环境的一个示例，并非对以上描述的增量分页器的示例性实施例或其它实施例的使用范围或功能提出任何局限。也不应将操作环境 4200 解释为对示例性操作环境 4200 中示出的任一组件或其组合具有任何依赖或需求。

实现增量分页器的过程可在诸如程序模块等在操作环境 4200 中执行的计算机可执行指令的一般上下文中描述。一般而言，程序模块包括执行特定的任务或实现特定的抽象数据类型的例程、程序、对象、组件、数据结构等等。此外，本领域的技术人员可以理解，实现增量分页器的过程可以使用各种计算系统配置来实施，包括手持式设备、多处理器系统、基于微处理器的或可编程消费电子产品、小型机、大型计算机等等。实现增量分页器的过程也可以在其中任务由通过通信网络链接的远程处理设备来执行的分布式计算环境中实践。在分布式计算环境中，程序模块可以位于包括存储器存储设备的本地和远程计算机存储介质中。

参考图 42，用于实现增量分页器的过程的示例性操作环境 4200 包括计算机 4210，它包括处理单元 4220、系统存储器 4230 以及将包括系统存储器 4230 的各类系统组件耦合至处理单元 4220 的系统总线 4221。

计算机 4210 通常包括各种计算机可读介质。作为示例而非局限，计算机可读介质包括计算机存储介质和通信介质。计算机存储介质的示例包括但不限于，随机存取存储器（RAM）；只读存储器（ROM）；电可擦除可编程只读存储器（EEPROM）；闪存或其它存储器技术；CD ROM、数字多功能盘（DVD）或其它光学或全息盘存储；磁带盒、磁带、磁盘存储或其它磁存储设备；或可以用来储存所期望的信息并可由计算机 4210 访问的任一其它介质。系统存储

器4230包括易失性和/或非易失性存储器形式的计算机存储介质，如ROM 4231和RAM 4232。基本输入/输出系统4233（BIOS）包括（如在启动时）帮助在计算机4210内的元件之间传输信息的基本例程，它通常储存在ROM 4231中。RAM 4232通常包含处理单元4220立即可访问和/或当前正在操作的数据和/或程序模块。作为示例而非局限，图42示出了操作系统4234、应用程序4235、其它程序模块4236和程序数据4237。

计算机4210也可包括其它可移动/不可移动、易失性/非易失性计算机存储介质。仅作示例，图42示出了对不可移动、非易失性磁介质进行读写的硬盘驱动器4241，对可移动、非易失性磁盘4252进行读写的磁盘驱动器4251，以及对可移动、非易失性光盘4256，如CD ROM或其它光介质进行读写的光盘驱动器4255。可以在示例性操作环境中使用的其它可移动/不可移动、易失性/非易失性计算机存储介质包括但不限于，磁带盒、闪存单元、数字多功能盘、数字录像带、固态RAM、固态ROM等等。硬盘驱动器4241通常通过不可移动存储器接口，如接口4240连接到系统总线4221。磁盘驱动器4251和光盘驱动器4255通常通过可移动存储器接口，如接口4250连接到系统总线4221。

上文讨论并在图42示出的驱动器及其关联的计算机存储介质为计算机4210提供了计算机可读指令、数据结构、程序模块和其它数据的存储。例如，示出硬盘驱动器4241储存操作系统4244、应用程序4245、其它程序模块4246和程序数据4247。注意，这些组件可以与操作系统4234、应用程序4235、其它程序模块4236和程序数据4237相同，也可以与它们不同。通常，储存在RAM中的操作系统、应用程序等是从硬盘驱动器4241读取的相应系统、程序或数据的部分，这些部分的大小和范围可取决于所需功能而变化。此处对操作系统4244、应用程序4245、其它程序模块4246和程序数据4247给予不同的标号以说明至少它们是不同的副本。用户可以通过输入设备，如键盘4262；定位设备4261（通常指鼠标、跟踪球或触摸垫）；无线输入接收组件4263；或诸如遥控器等无线源向计算机4210输入命令和信息。其它输入设备（未示出）可包括话筒、操纵杆、游戏垫、圆盘式卫星天线、扫描仪等等。这些和其它输入设备通常通过耦合至系统总线4221的用户输入接口4260连接至处理单元4220，但是也可以通过其它接口和总线结构连接，如并行端口、游戏端口、IEEE

4294 端口或通用串行总线（USB）4298 或红外（IR）总线 4299。如上所述，输入/输出功能可以经由通信网络以分布式方式来促进。

显示设备 4291 也通过接口，如视频接口 4290 连接至系统总线 4221。显示设备 4291 可以是显示计算机 4210 的输出的任何设备，不限于监视器、LCD 屏幕、TFT 屏幕、平板显示器、常规电视机或屏幕投影仪。除显示设备 4291 之外，计算机也可包括其它外围输出设备，如扬声器 4297 和打印机 4296，它们通过输出外围接口 4295 连接。

计算机 4210 可以使用到一个或多个远程计算机，如远程计算机 4280 的逻辑连接在网络化环境中操作。远程计算机 4280 可以是个人计算机，并通常包括许多或所有以上相对于计算机 4210 所描述的元件，尽管在图 42 中仅示出了存储器存储设备 4281。图 42 描述的逻辑连接包括局域网（LAN）4271 和广域网（WAN）4273，但也可包括其它网络，诸如到城域网（MAN）、内联网或因特网的连接。

当在 LAN 网络环境中使用时，计算机 4210 通过网络接口或适配器 4270 连接至 LAN 4271。当在 WAN 网络环境中使用时，计算机 4210 通常包括调制解调器 4272 或用于通过 WAN 4273，如因特网建立通信的其它装置。调制解调器 4272 可以是内置或外置的，它通过网络接口 4270 或其它适当的机制连接至系统总线 4221。调制解调器 4272 可以是电缆调制解调器、DSL 调制解调器或其它宽带设备。在网络化环境中，相对于计算机 4210 所描述的程序模块或其部分可储存在远程存储器存储设备中。作为示例而非局限，图 42 示出远程应用程序 4285 驻留在存储器设备 4281 上。可以理解，示出的网络连接是示例性的，也可以使用在计算机之间建立通信链路的其它手段。

尽管未示出计算机 4210 的许多其它内部组件，但是本领域的普通技术人员将认识到，这些组件和互连是公知的。例如，在计算机 4210 内包括诸如电视调谐卡和网络接口卡等各种扩展卡是常规的。因此，关于计算机 4210 的内部构造的附加细节不需要在描述实现增量分页器的过程的示例性实施例时公开。

当计算机 4210 被开启或重启时，储存在 ROM 4231 中的 BIOS 4233 指示处理单元 4220 将操作系统或其必要的部分从硬盘驱动器 4241 加载到 RAM

4232 中。一旦操作系统中被指定为操作系统 4244 的所复制部分被加载到 RAM 4232 中，处理单元 4220 就执行该操作系统代码，并使得与操作系统 4232 的用户界面相关联的可视元素被显示在显示设备 4291 上。通常，当应用程序 4245 被用户打开时，从硬盘驱动器 4241 中读取程序代码和相关数据，并且将必要的部分复制到 RAM 4232 中，所复制的部分此处由参考标号 4235 来表示。

结论

尽管以对结构特征和/或方法动作专用的语言描述了示例性实施例，但是可以理解，所附权利要求书不一定限于以上所描述的具体特征或动作。相反，这些具体特征和动作是作为示例性实施例来公开的。

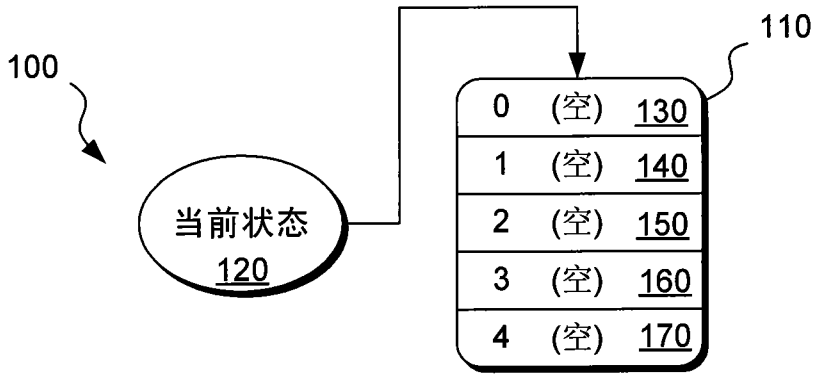

100
当前状态
120
110
0 (空) 130
1 (空) 140
2 (空) 150
3 (空) 160
4 (空) 170

图 1

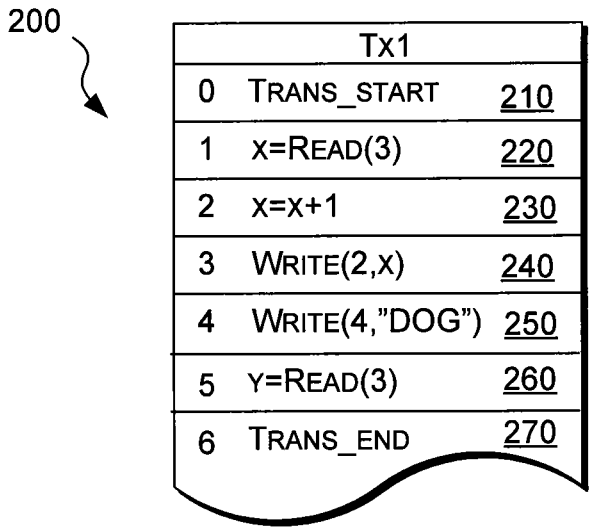

200
Tx1
0 TRANS_START 210
1 x=READ(3) 220
2 x=x+1 230
3 WRITE(2,x) 240
4 WRITE(4,"DOG") 250
5 y=READ(3) 260
6 TRANS_END 270

图 2

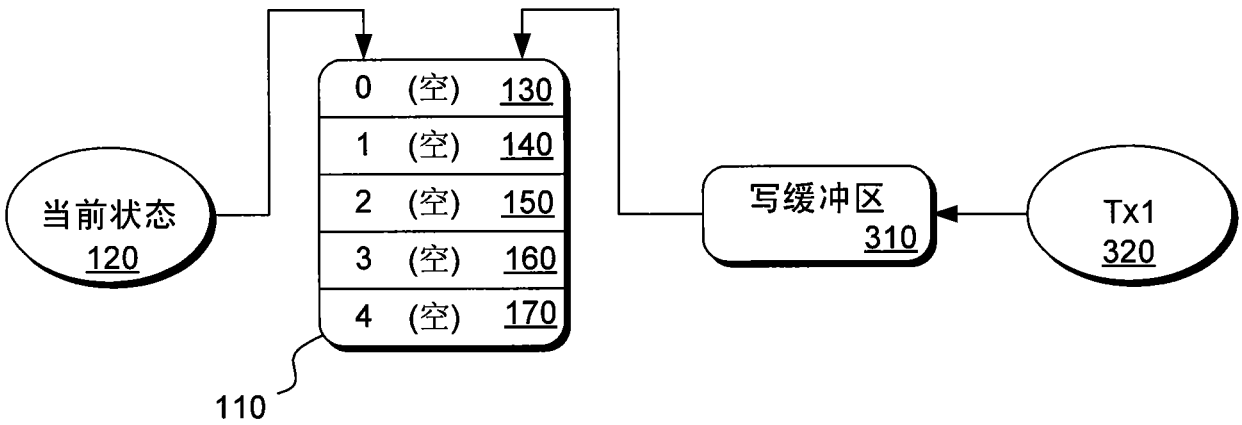

当前状态
120
0 (空) 130
1 (空) 140
2 (空) 150
3 (空) 160
4 (空) 170
110
写缓冲区
310
Tx1
320

图 3

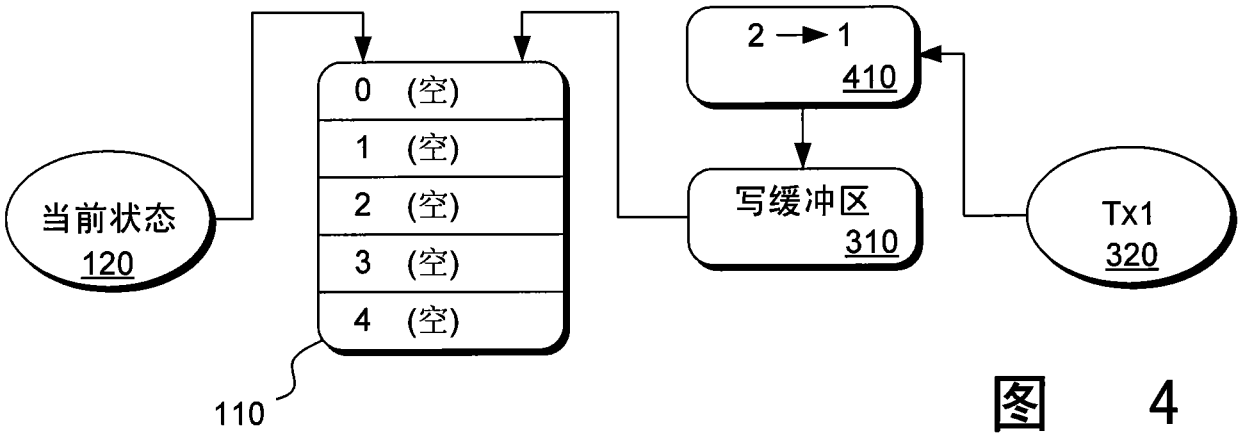

2 → 1
410
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
当前状态
120
写缓冲区
310
Tx1
320
110

图 4

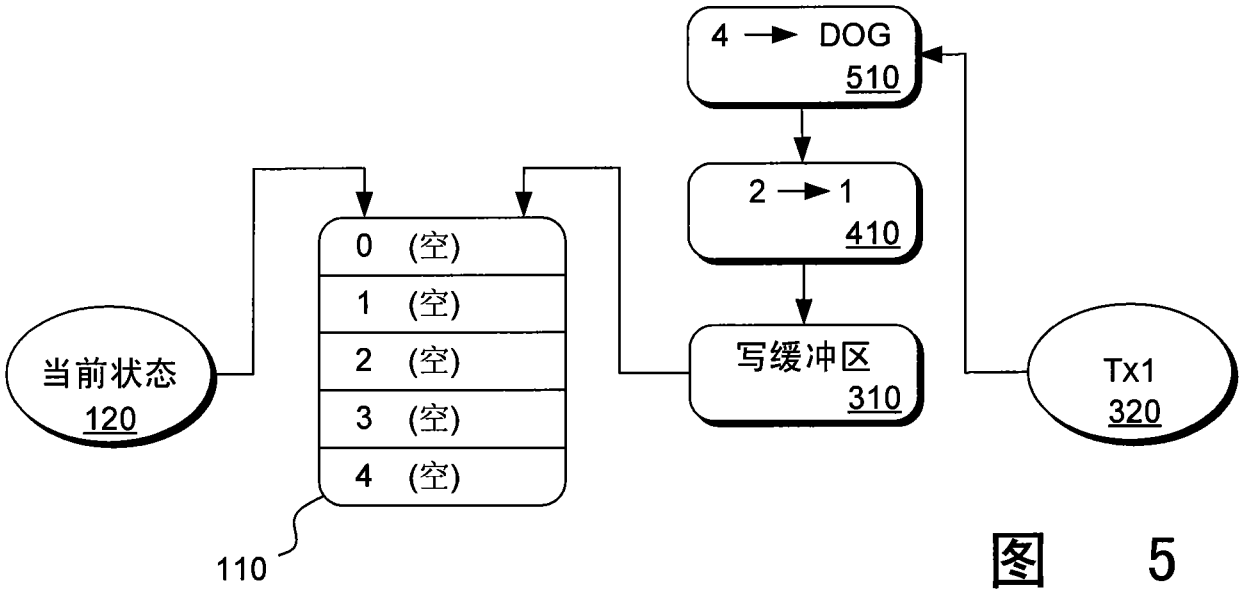

4 → DOG
510
2 → 1
410
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
当前状态
120
写缓冲区
310
Tx1
320
110

图 5

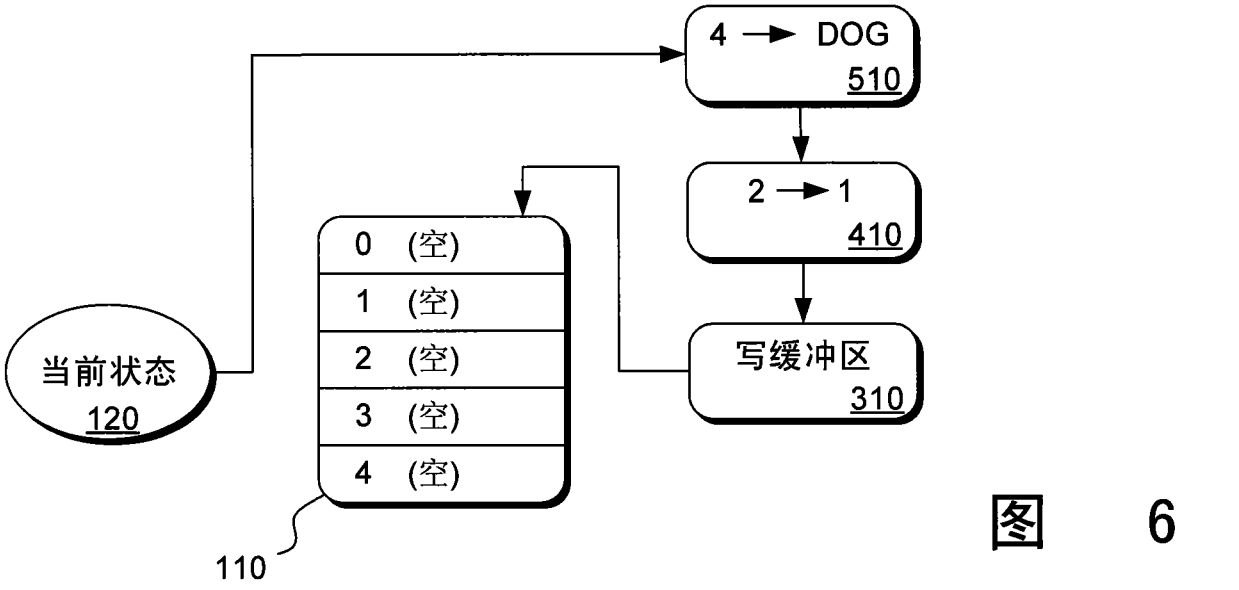

4 → DOG
510
2 → 1
410
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
当前状态
120
写缓冲区
310
110

图 6

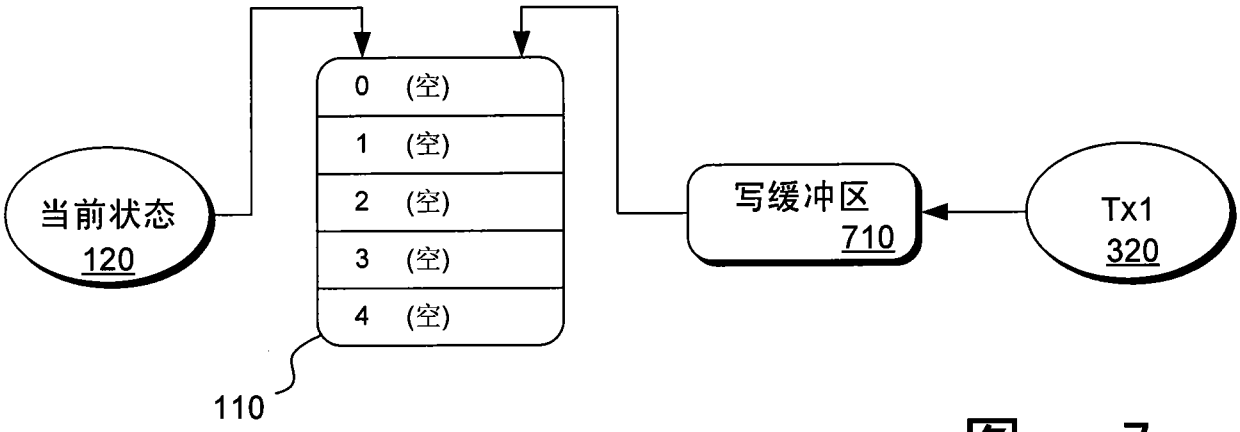

0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
当前状态
120
写缓冲区
710
Tx1
320
110

图 7

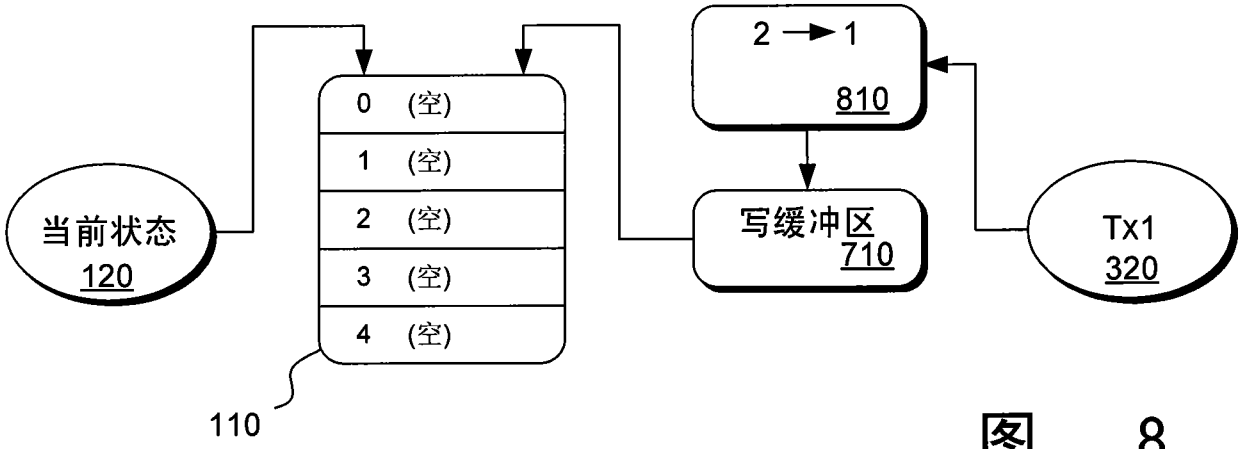

2 → 1
810
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
当前状态
120
写缓冲区
710
Tx1
320
110

图 8

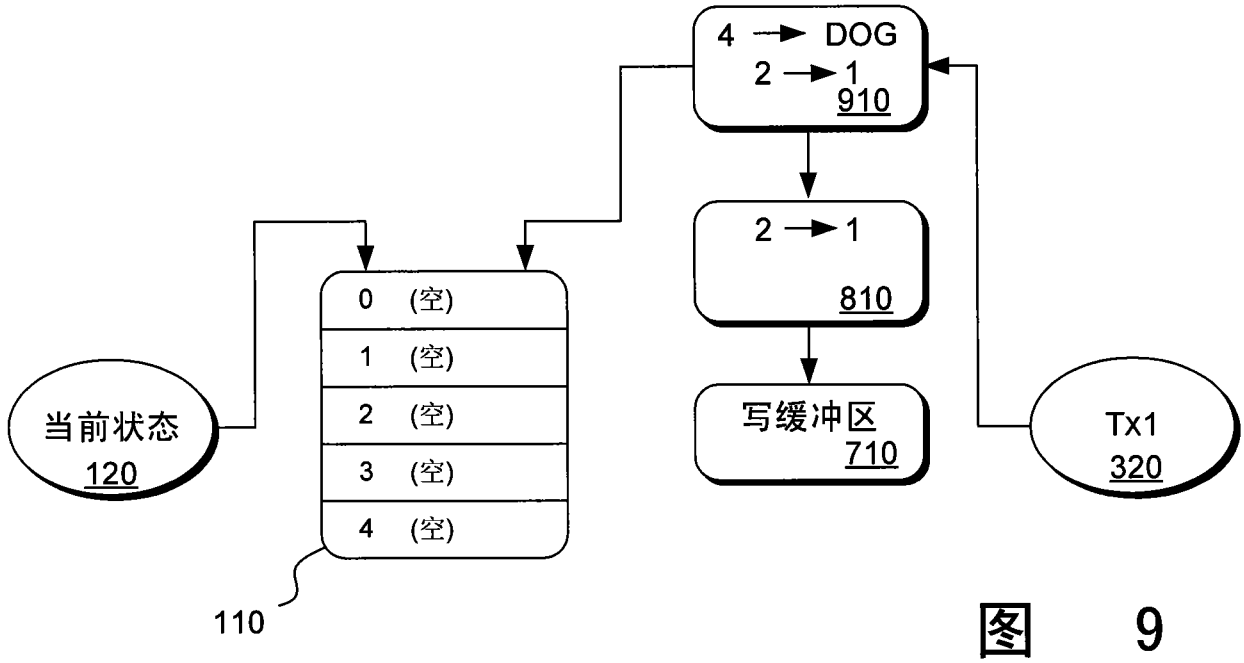

4 → DOG
2 → 1
910
2 → 1
810
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
当前状态
120
写缓冲区
710
Tx1
320
110

图 9

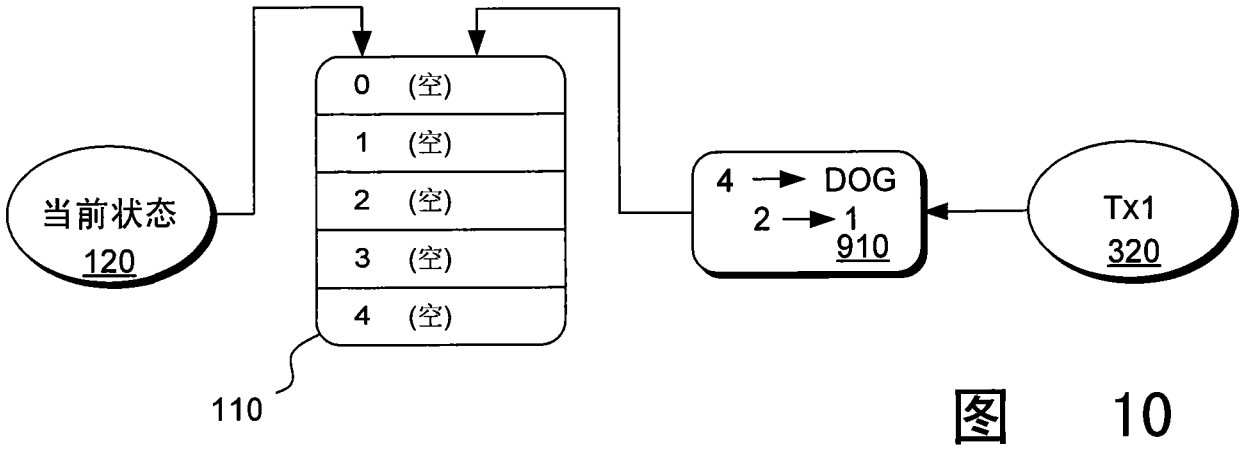
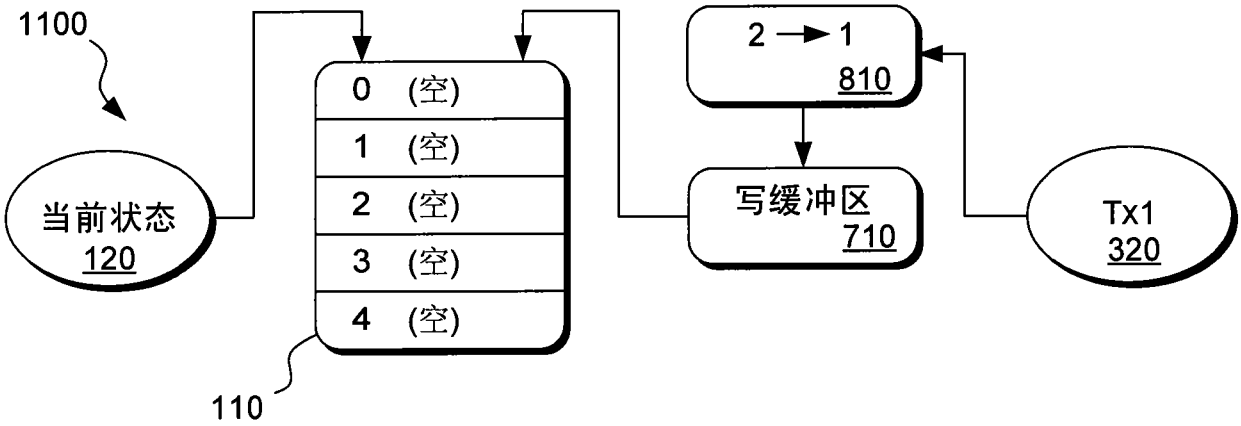
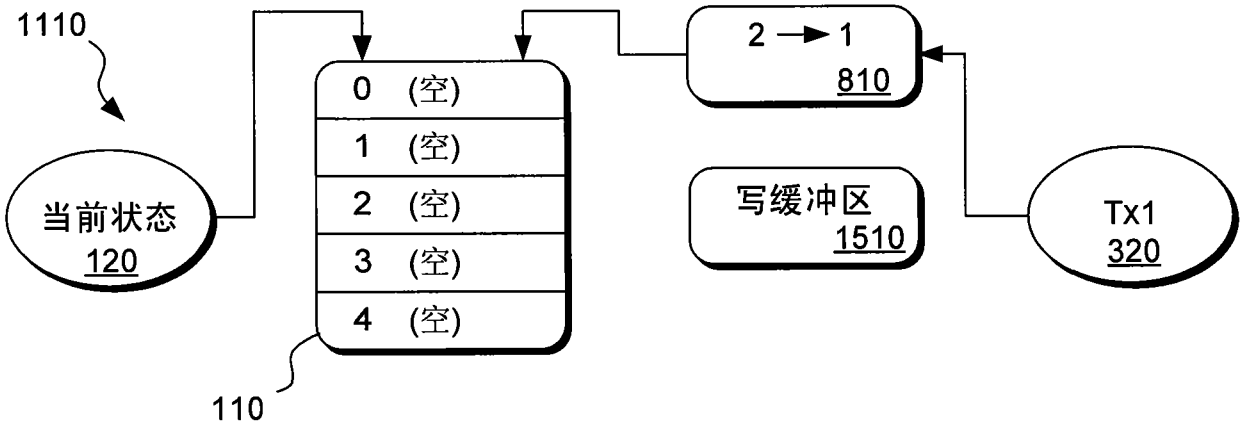

0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110
当前状态
120
4 → DOG
2 → 1
910
Tx1
320

图 10


1100
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110
当前状态
120
2 → 1
810
写缓冲区
710
Tx1
320


1110
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110
当前状态
120
2 → 1
810
写缓冲区
1510
Tx1
320

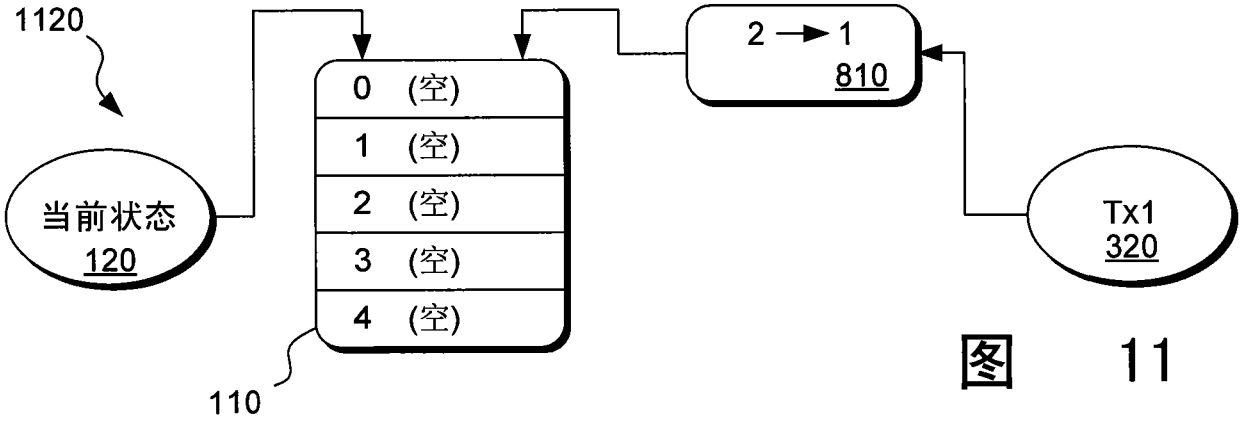

1120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110
当前状态
120
2 → 1
810
Tx1
320

图 11

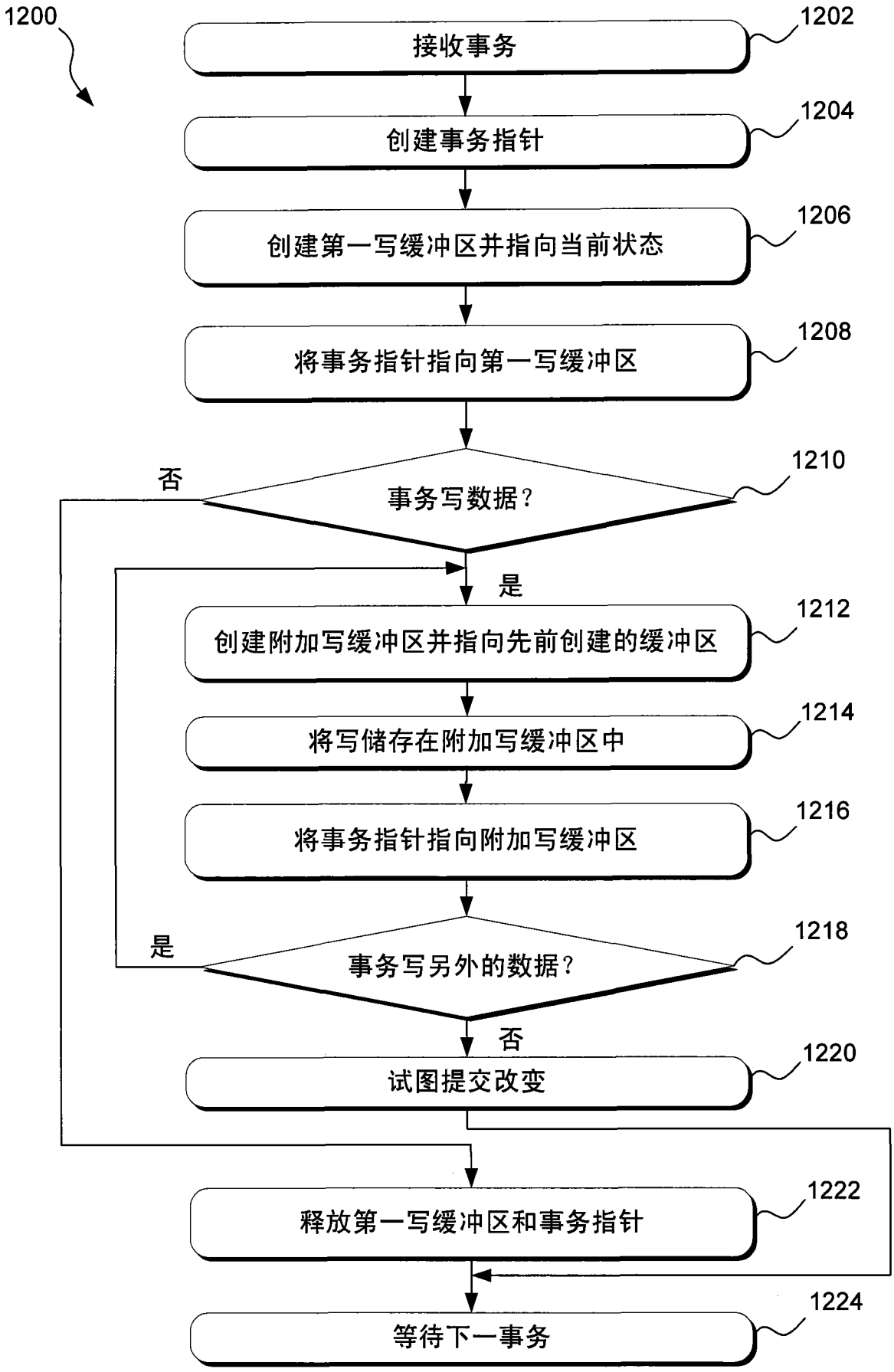

1200
接收事务
1202
创建事务指针
1204
创建第一写缓冲区并指向当前状态
1206
将事务指针指向第一写缓冲区
1208
事务写数据？
1210
否
是
创建附加写缓冲区并指向先前创建的缓冲区
1212
将写储存在附加写缓冲区中
1214
将事务指针指向附加写缓冲区
1216
事务写另外的数据？
1218
是
否
试图提交改变
1220
释放第一写缓冲区和事务指针
1222
等待下一事务
1224

图 12

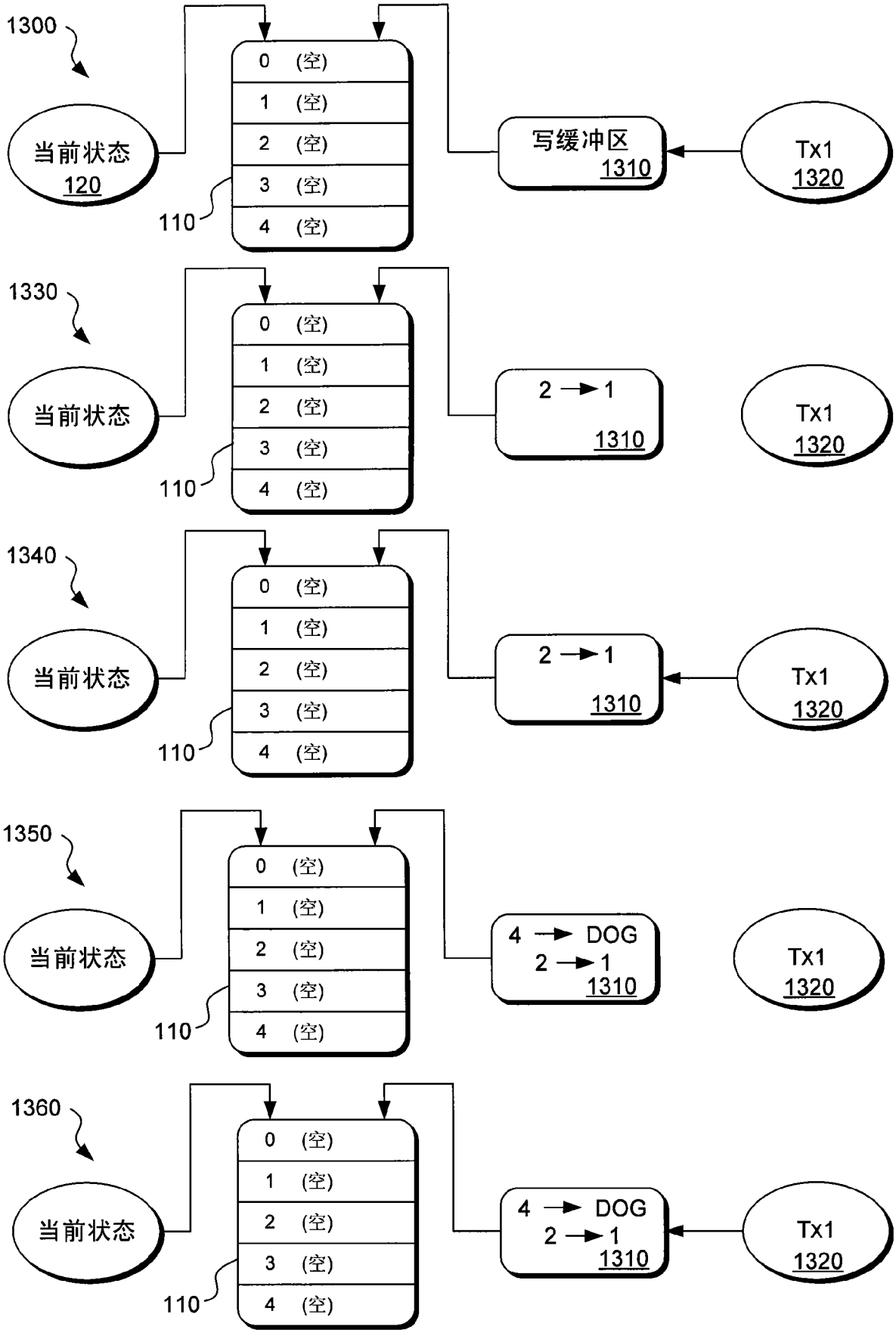

1300
当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110
写缓冲区
1310
Tx1
1320
1330
当前状态
110
2 → 1
1310
Tx1
1320
1340
当前状态
110
2 → 1
1310
Tx1
1320
1350
当前状态
110
4 → DOG
2 → 1
1310
Tx1
1320
1360
当前状态
110
4 → DOG
2 → 1
1310
Tx1
1320

图 13

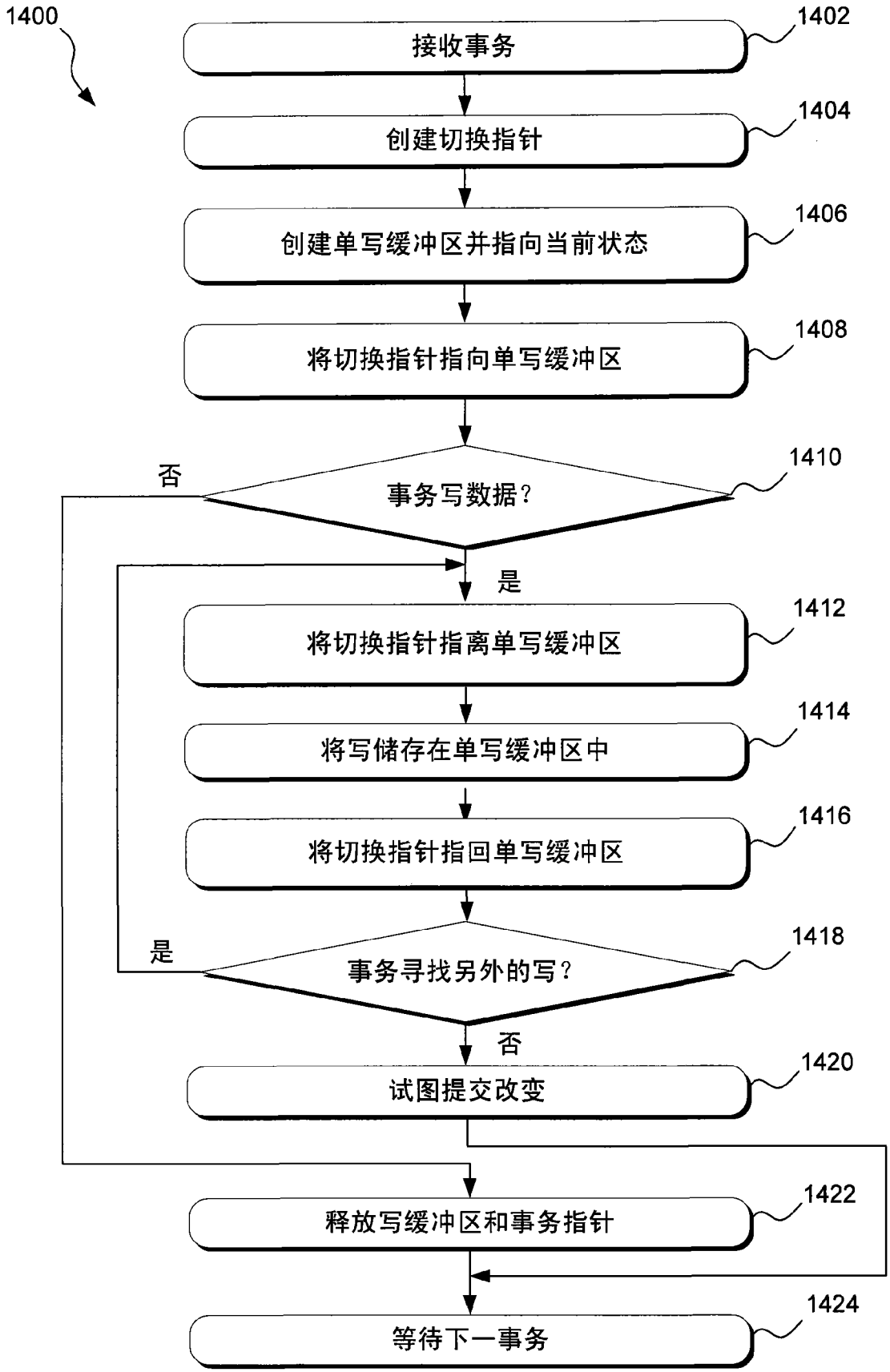

1400
接收事务
1402
创建切换指针
1404
创建单写缓冲区并指向当前状态
1406
将切换指针指向单写缓冲区
1408
事务写数据？
1410
否
是
将切换指针指离单写缓冲区
1412
将写储存在单写缓冲区中
1414
将切换指针指回单写缓冲区
1416
事务寻找另外的写？
1418
是
否
试图提交改变
1420
释放写缓冲区和事务指针
1422
等待下一事务
1424

图 14

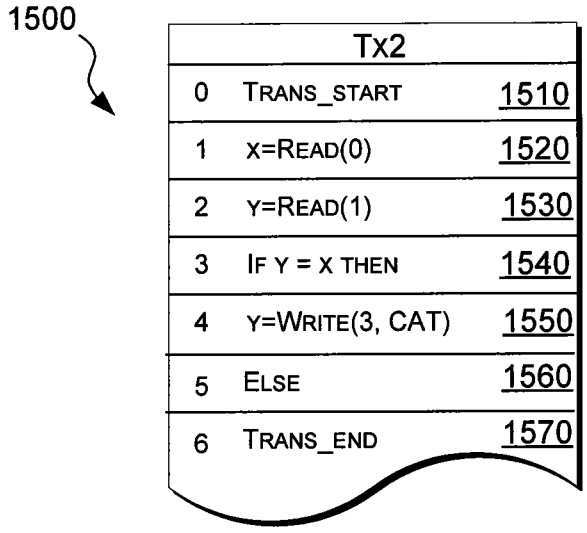

1500
Tx2
0 TRANS_START 1510
1 X=READ(0) 1520
2 Y=READ(1) 1530
3 IF Y = X THEN 1540
4 Y=WRITE(3, CAT) 1550
5 ELSE 1560
6 TRANS_END 1570

图 15

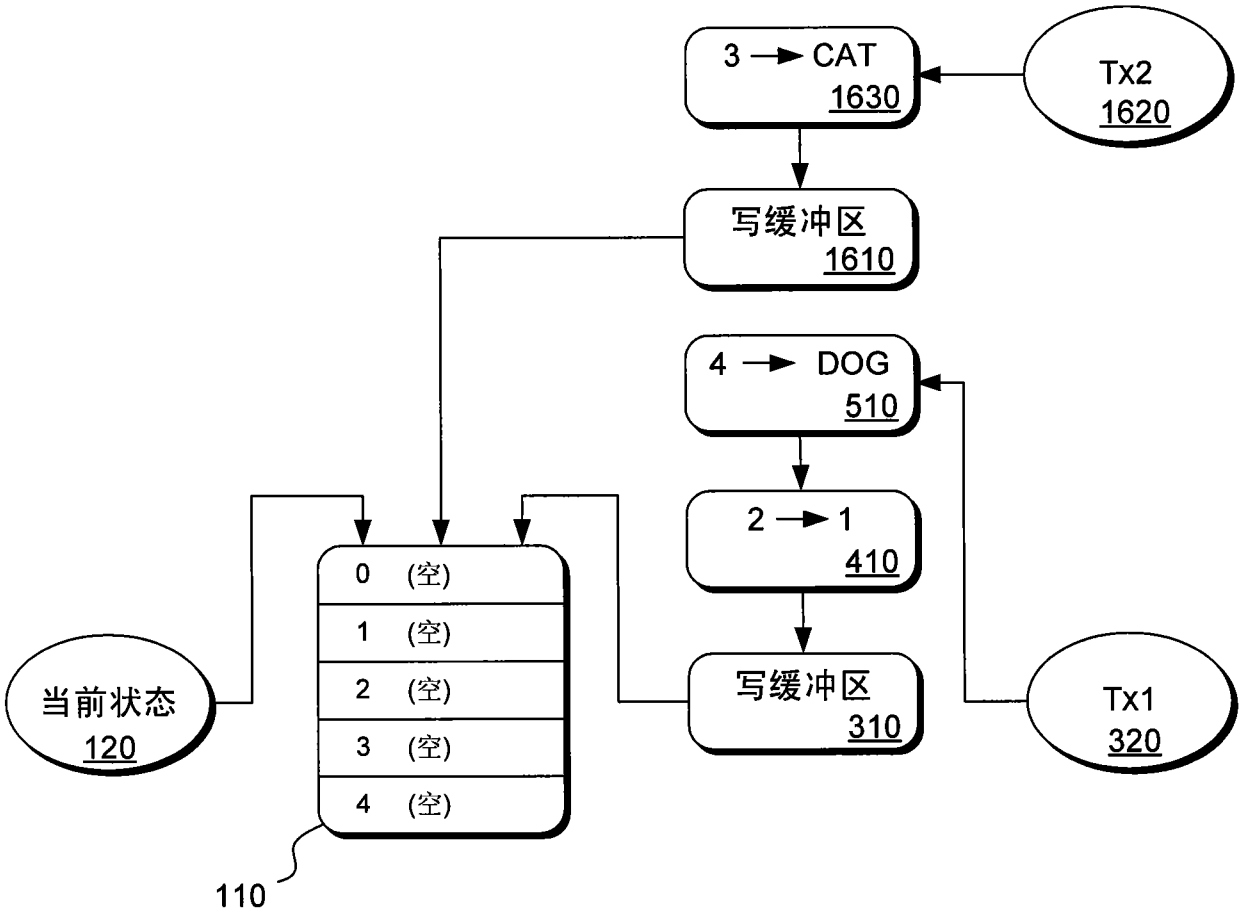

3 → CAT
1630
Tx2
1620
写缓冲区
1610
4 → DOG
510
2 → 1
410
写缓冲区
310
Tx1
320
当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110

图 16

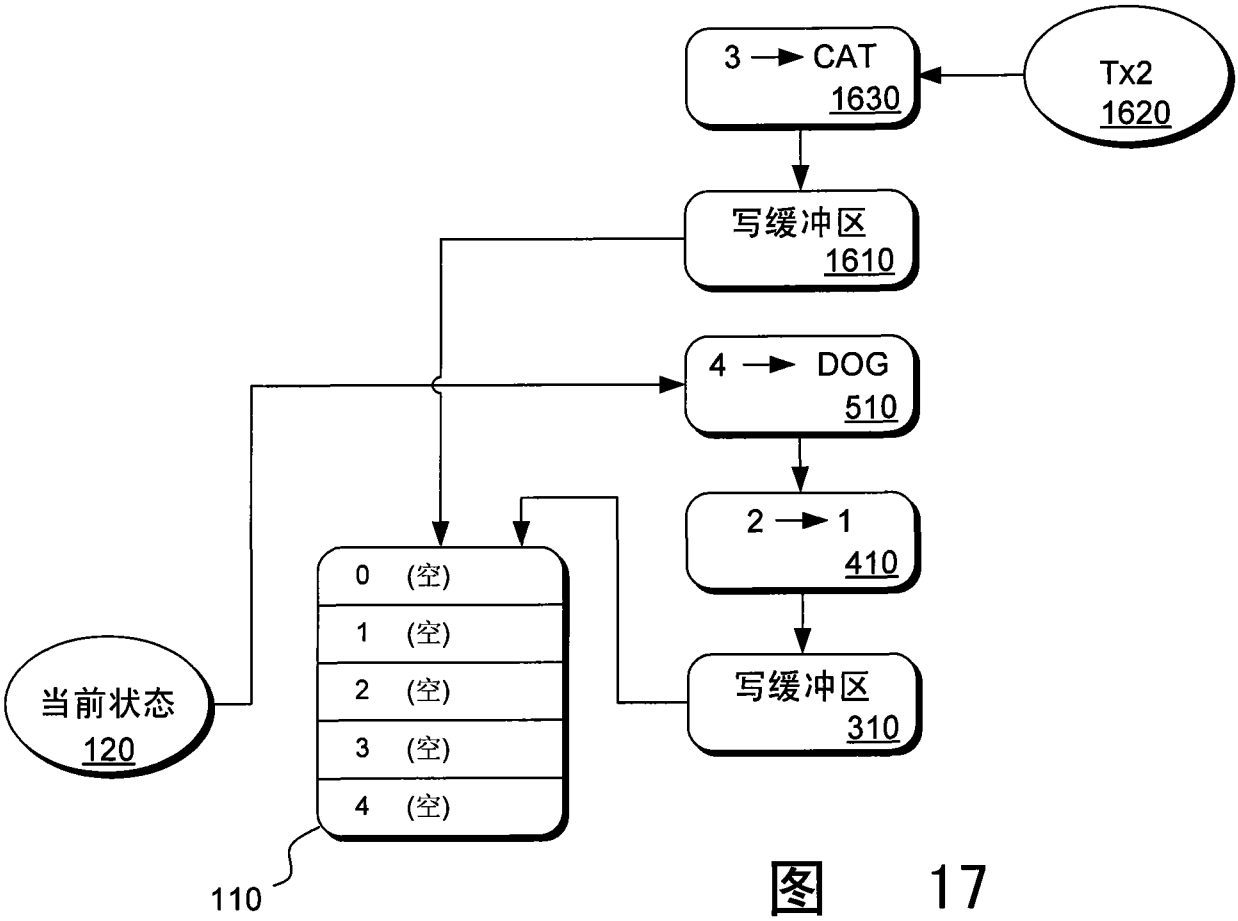

3 → CAT
1630
Tx2
1620
写缓冲区
1610
4 → DOG
510
2 → 1
410
写缓冲区
310
当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110

图 17

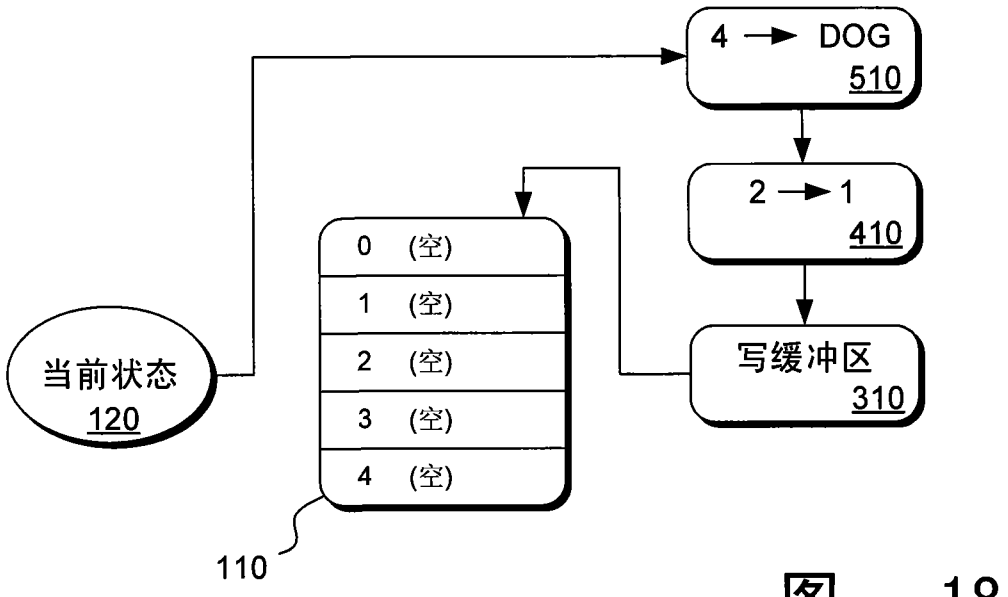

4 → DOG
510
2 → 1
410
写缓冲区
310
当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110

图 18

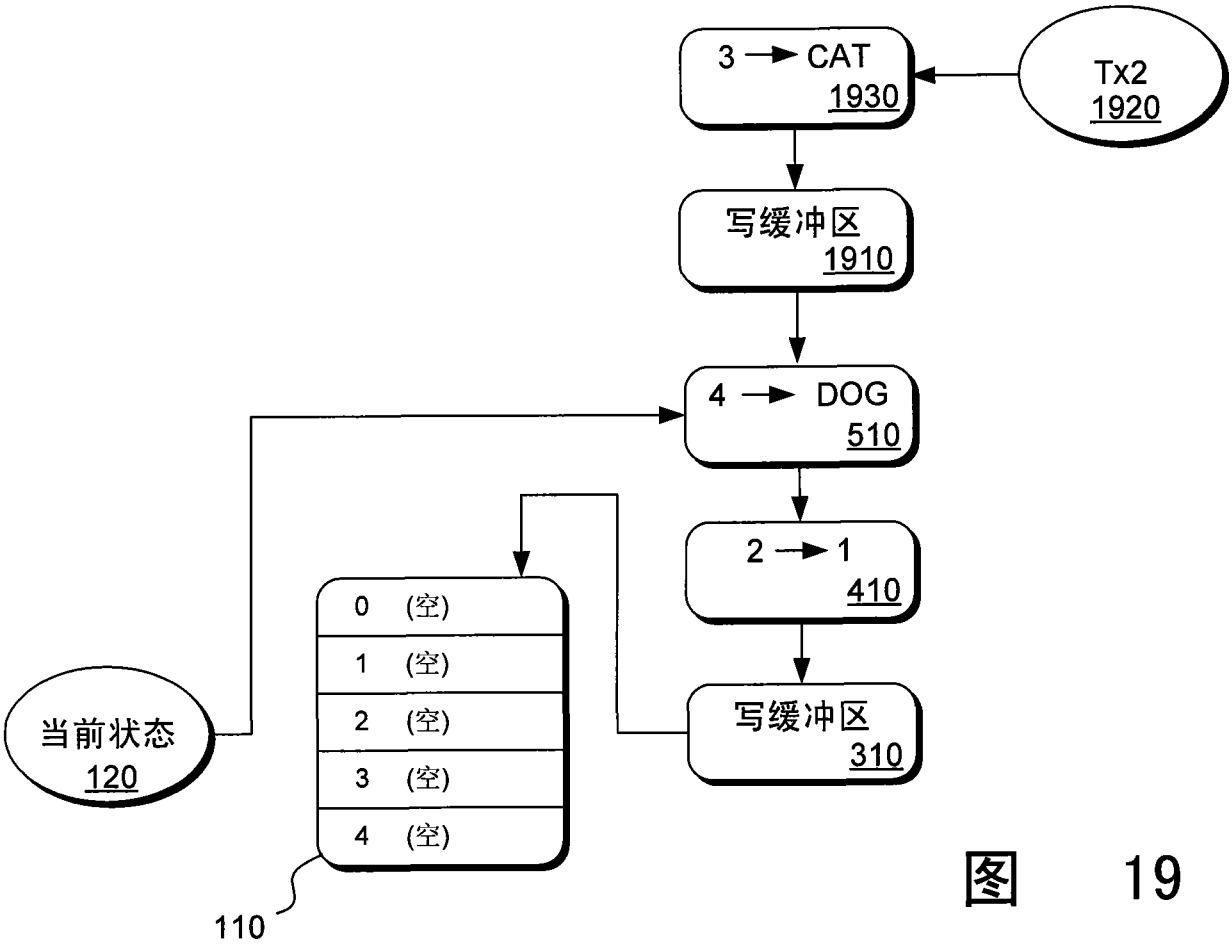

3 → CAT
1930
Tx2
1920
写缓冲区
1910
4 → DOG
510
2 → 1
410
写缓冲区
310
当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110

图 19

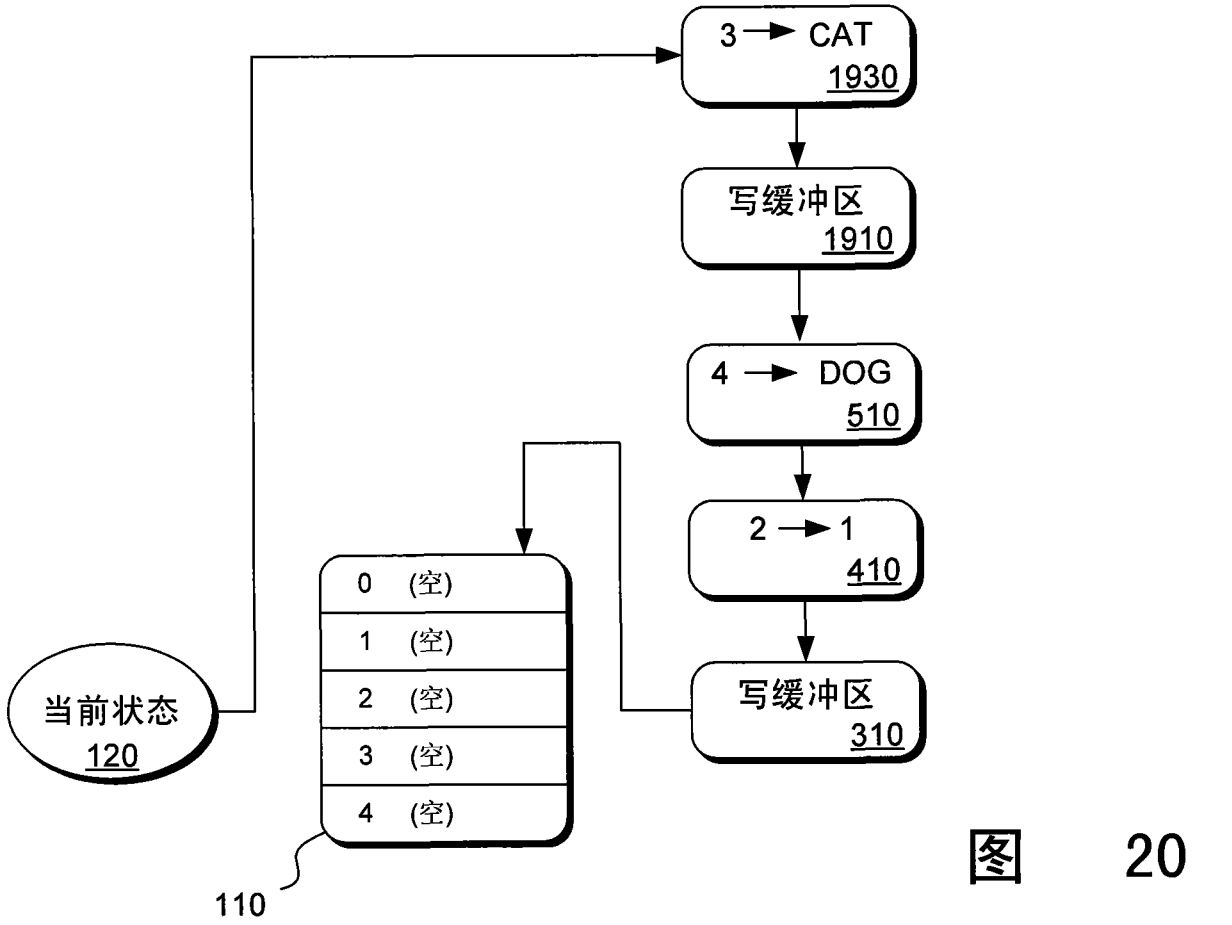

3 → CAT
1930
写缓冲区
1910
4 → DOG
510
2 → 1
410
写缓冲区
310
当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110

图 20

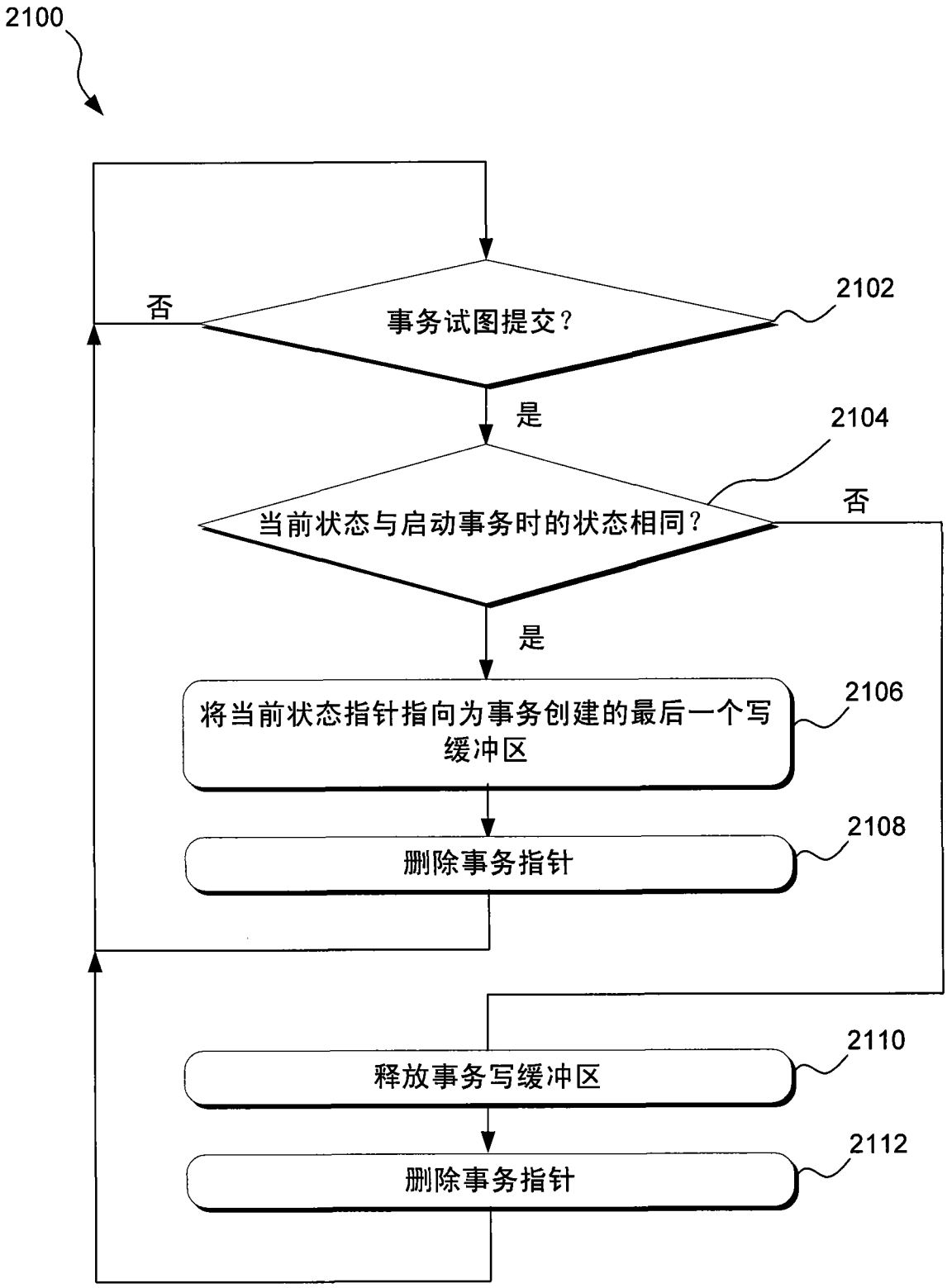

2100
事务试图提交？
否
2102
是
当前状态与启动事务时的状态相同？
2104
否
是
将当前状态指针指向为事务创建的最后一个写缓冲区
2106
删除事务指针
2108
释放事务写缓冲区
2110
删除事务指针
2112

图 21

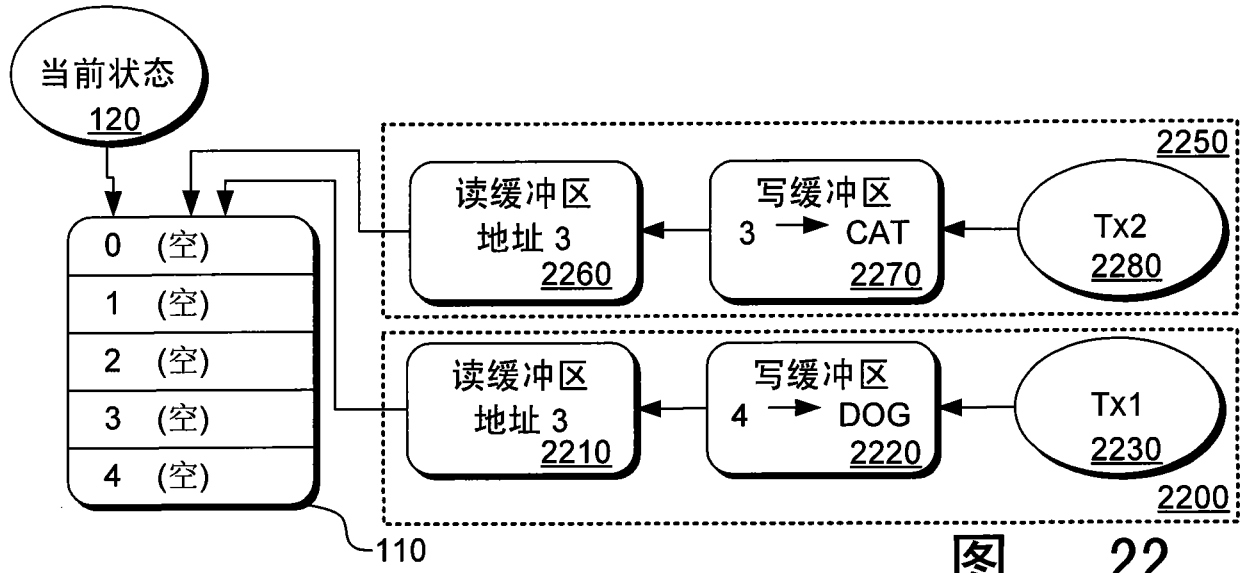

当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110
读缓冲区
地址 3
2260
写缓冲区
3 → CAT
2270
Tx2
2280
2250
读缓冲区
地址 3
2210
写缓冲区
4 → DOG
2220
Tx1
2230
2200

图 22

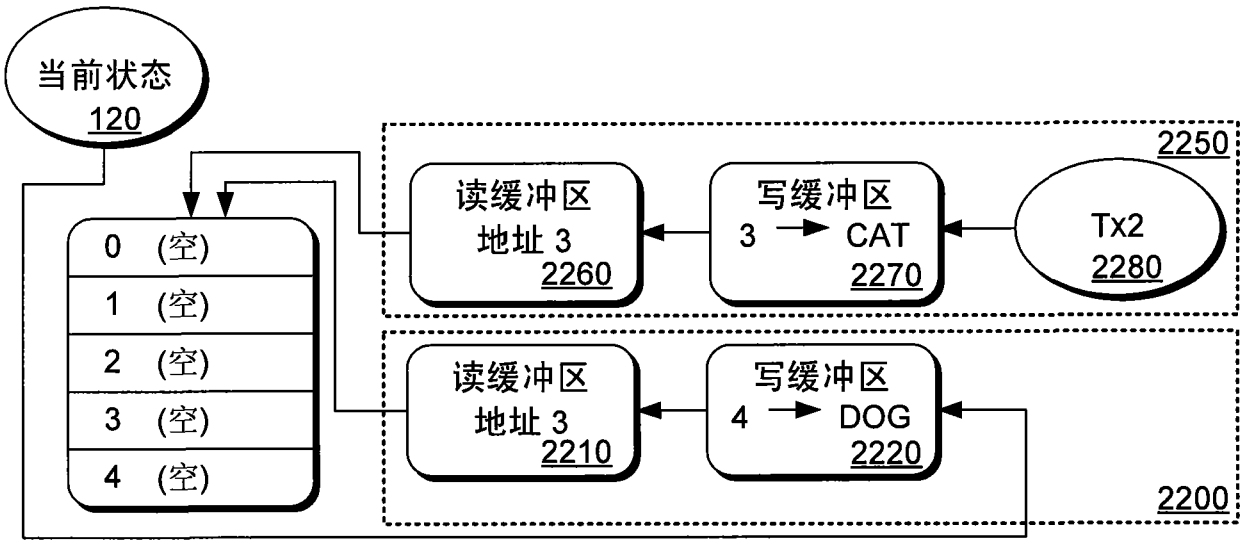

当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
读缓冲区
地址 3
2260
写缓冲区
3 → CAT
2270
Tx2
2280
2250
读缓冲区
地址 3
2210
写缓冲区
4 → DOG
2220
2200

图 23

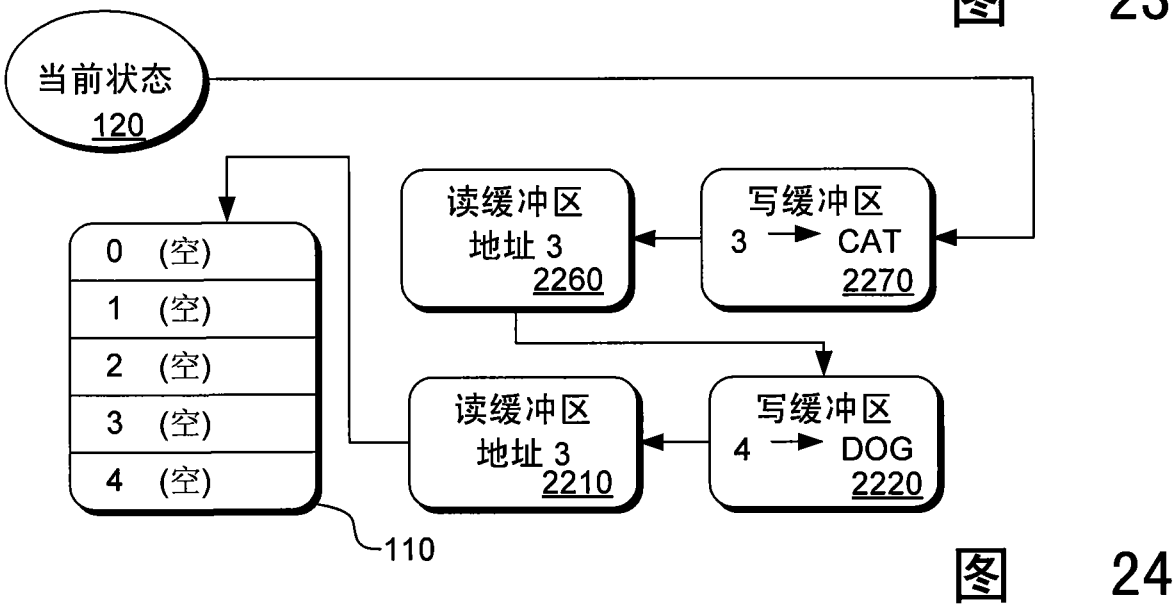

当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110
读缓冲区
地址 3
2260
写缓冲区
3 → CAT
2270
读缓冲区
地址 3
2210
写缓冲区
4 → DOG
2220

图 24

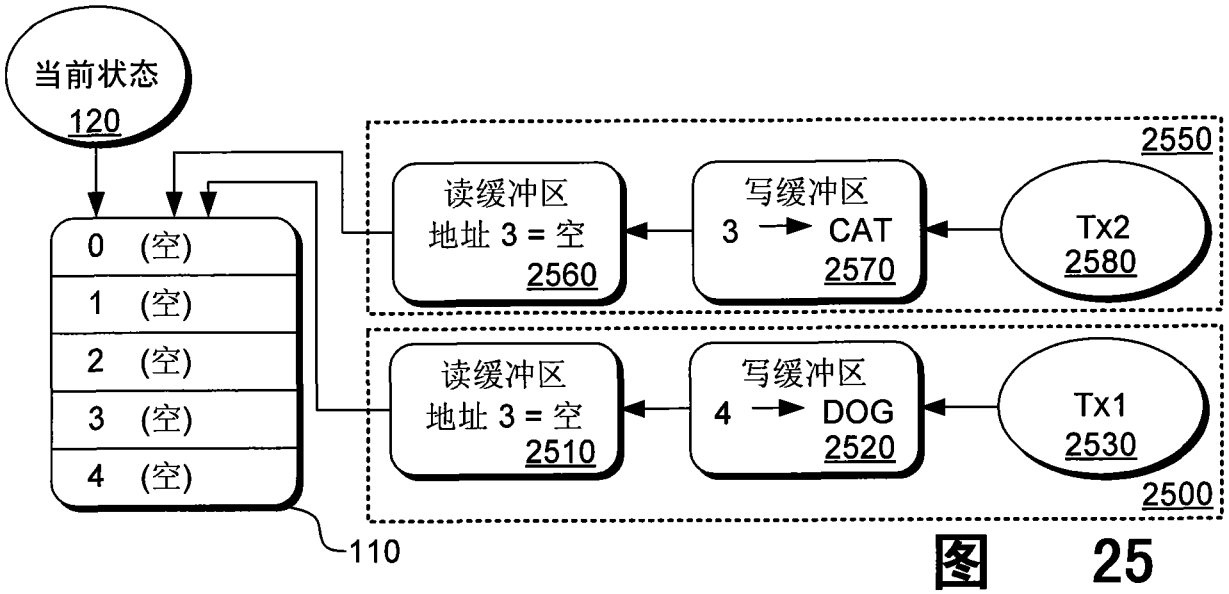

当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110
2550
读缓冲区
地址 3 = 空
2560
写缓冲区
3 → CAT
2570
Tx2
2580
读缓冲区
地址 3 = 空
2510
写缓冲区
4 → DOG
2520
Tx1
2530
2500

图 25

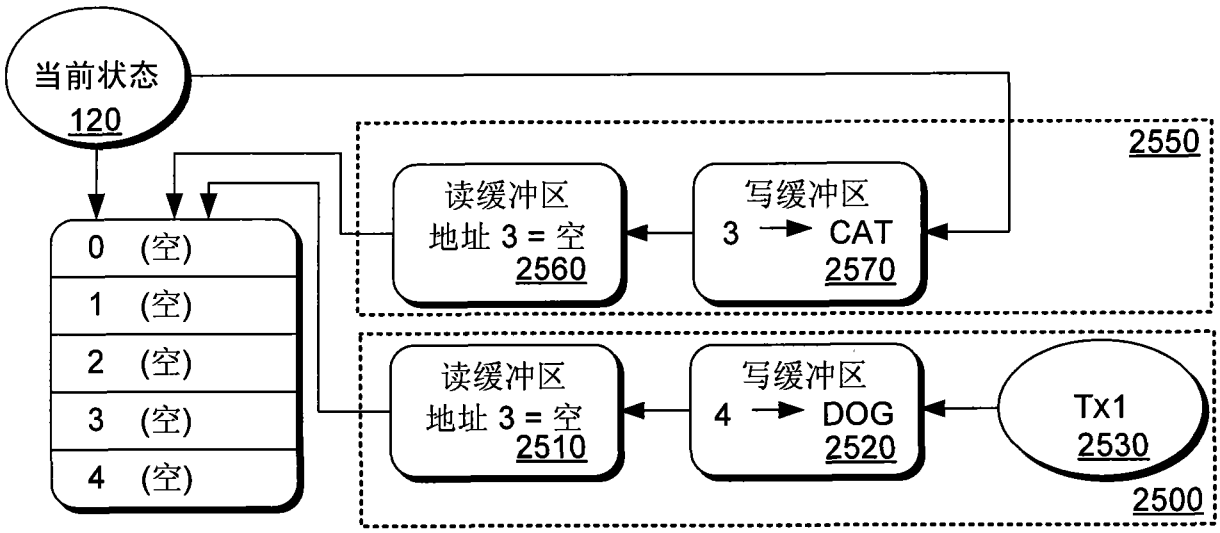

当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
2550
读缓冲区
地址 3 = 空
2560
写缓冲区
3 → CAT
2570
读缓冲区
地址 3 = 空
2510
写缓冲区
4 → DOG
2520
Tx1
2530
2500

图 26

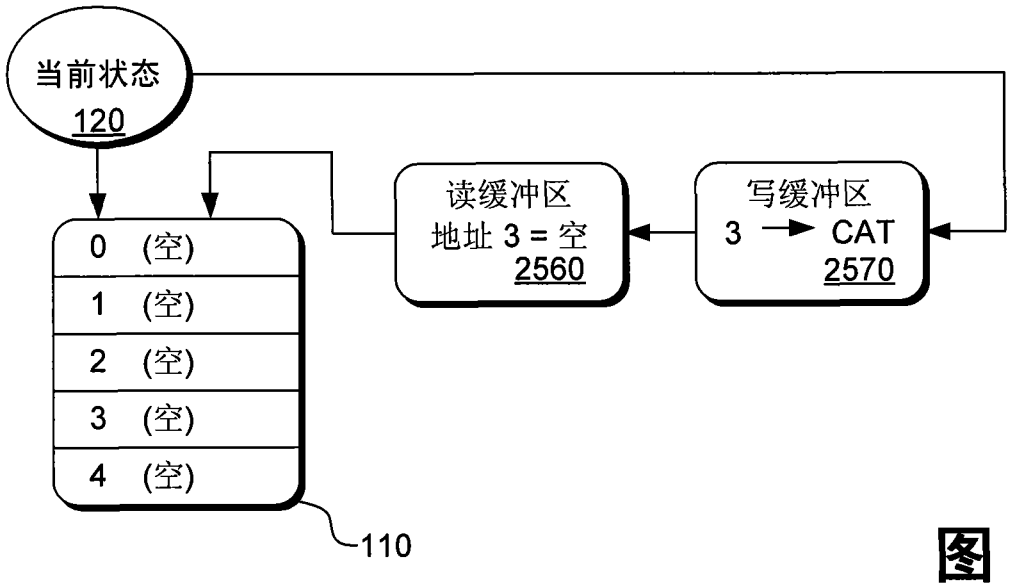

当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110
读缓冲区
地址 3 = 空
2560
写缓冲区
3 → CAT
2570

图 27

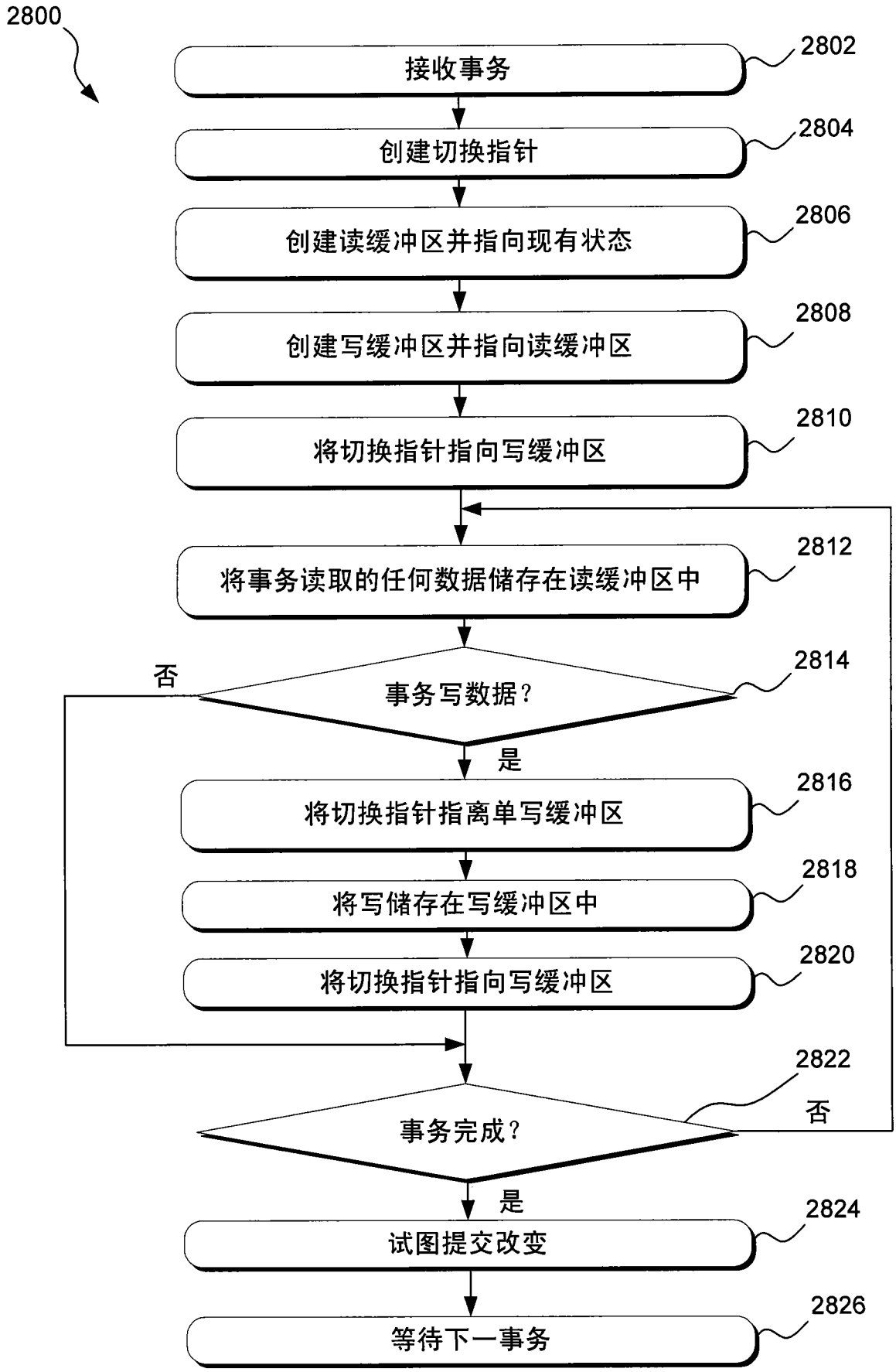

2800
接收事务
2802
创建切换指针
2804
创建读缓冲区并指向现有状态
2806
创建写缓冲区并指向读缓冲区
2808
将切换指针指向写缓冲区
2810
将事务读取的任何数据储存在读缓冲区中
2812
事务写数据？
2814
否
是
将切换指针指离单写缓冲区
2816
将写储存在写缓冲区中
2818
将切换指针指向写缓冲区
2820
事务完成？
2822
否
是
试图提交改变
2824
等待下一事务
2826

图 28

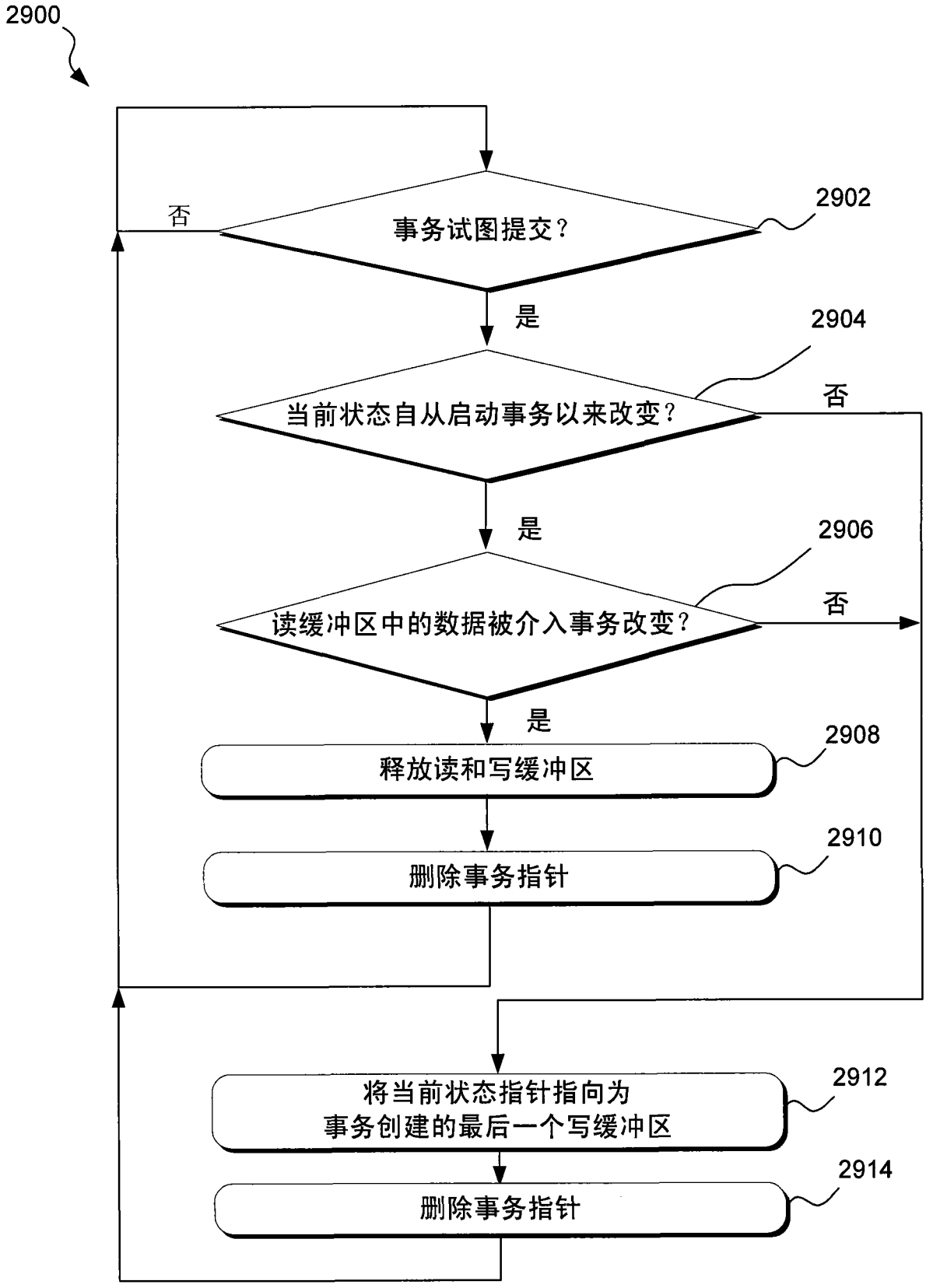
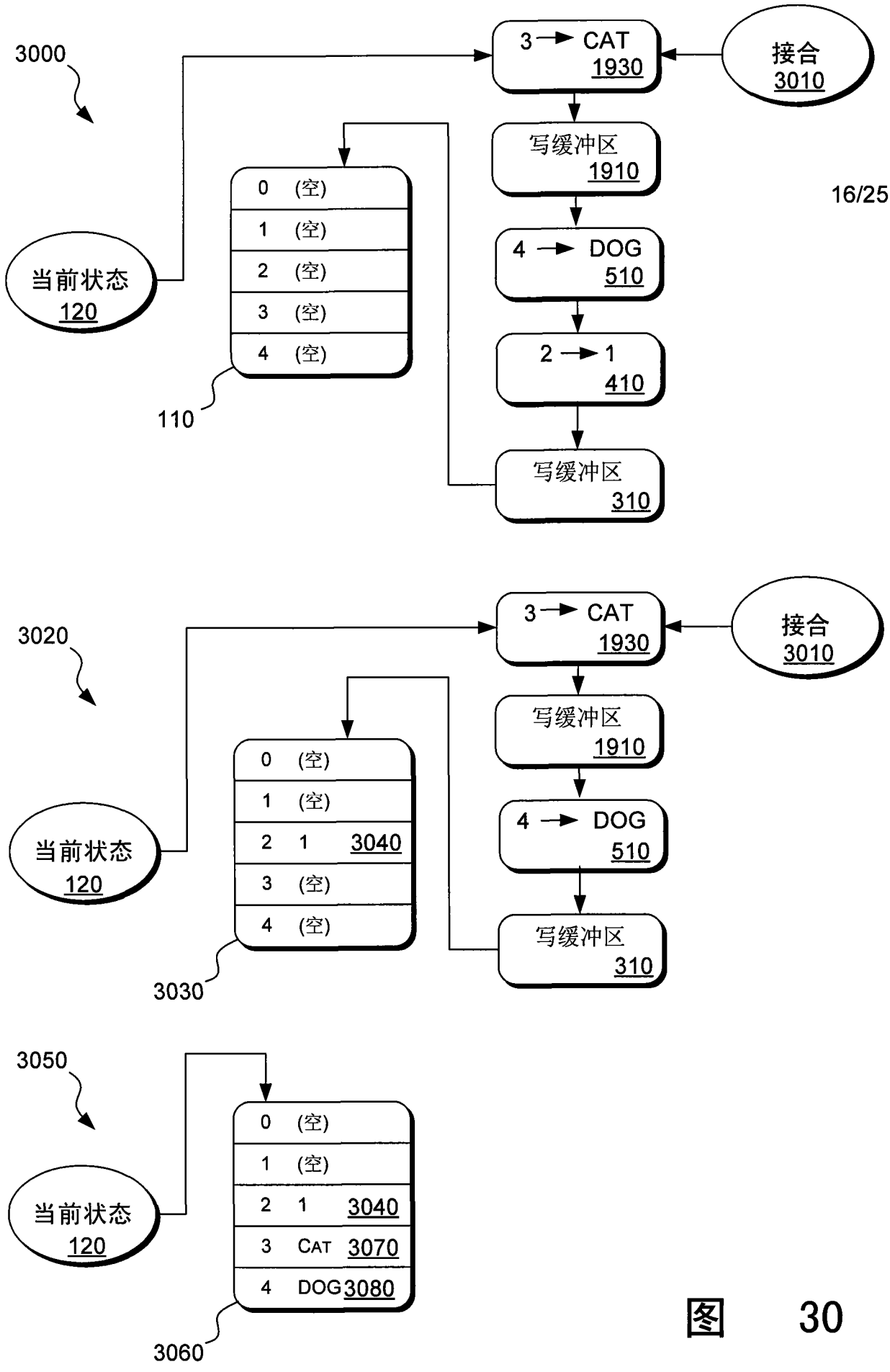

2900
事务试图提交？
否
2902
是
当前状态自从启动事务以来改变？
2904
否
是
读缓冲区中的数据被介入事务改变？
2906
否
是
释放读和写缓冲区
2908
删除事务指针
2910
将当前状态指针指向为
事务创建的最后一个写缓冲区
2912
删除事务指针
2914

图 29


3000
3 → CAT
1930
接合
3010
写缓冲区
1910
4 → DOG
510
2 → 1
410
写缓冲区
310
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110
当前状态
120
3020
3 → CAT
1930
接合
3010
写缓冲区
1910
4 → DOG
510
写缓冲区
310
0 (空)
1 (空)
2 1 3040
3 (空)
4 (空)
3030
当前状态
120
3050
0 (空)
1 (空)
2 1 3040
3 CAT 3070
4 DOG 3080
3060
当前状态
120

16/25

图 30

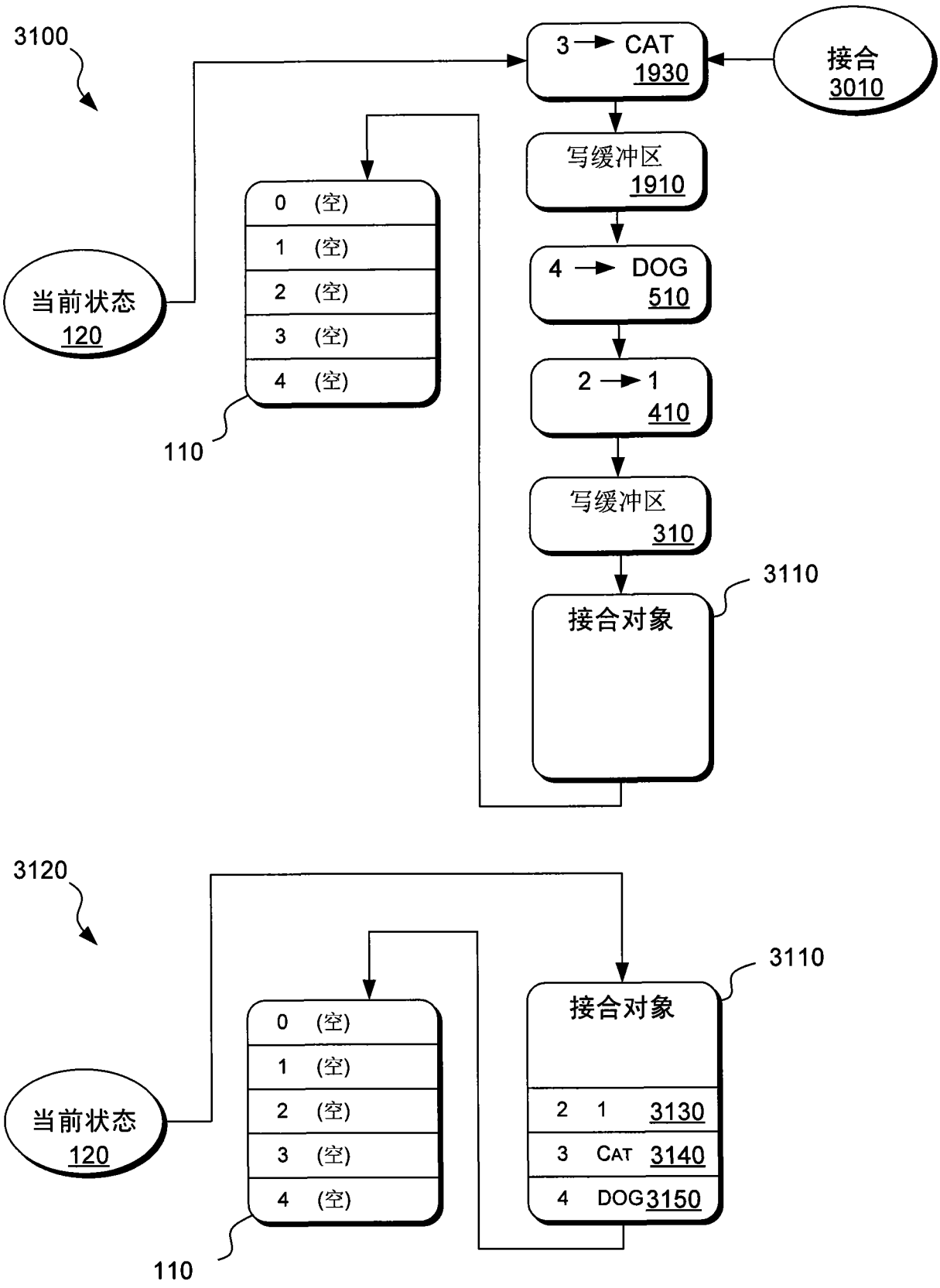

3100
3 → CAT
1930
接合
3010
写缓冲区
1910
4 → DOG
510
2 → 1
410
写缓冲区
310
3110
接合对象
当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110
3120
3110
接合对象
2 1 3130
3 CAT 3140
4 DOG 3150
当前状态
120
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110

图 31

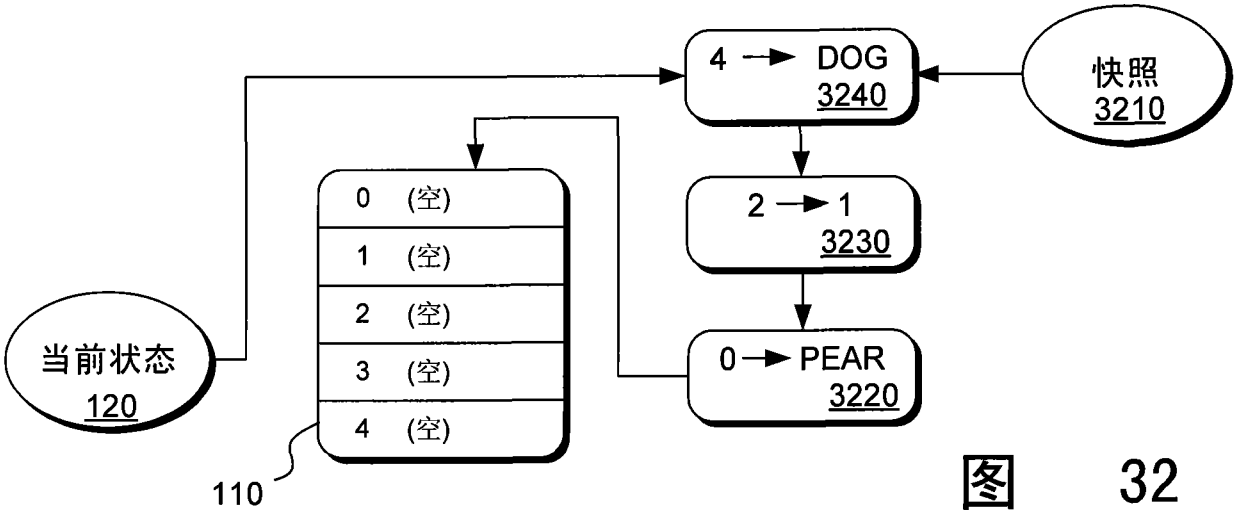

4 → DOG
3240
快照
3210
2 → 1
3230
0 → PEAR
3220
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
当前状态
120
110

图 32

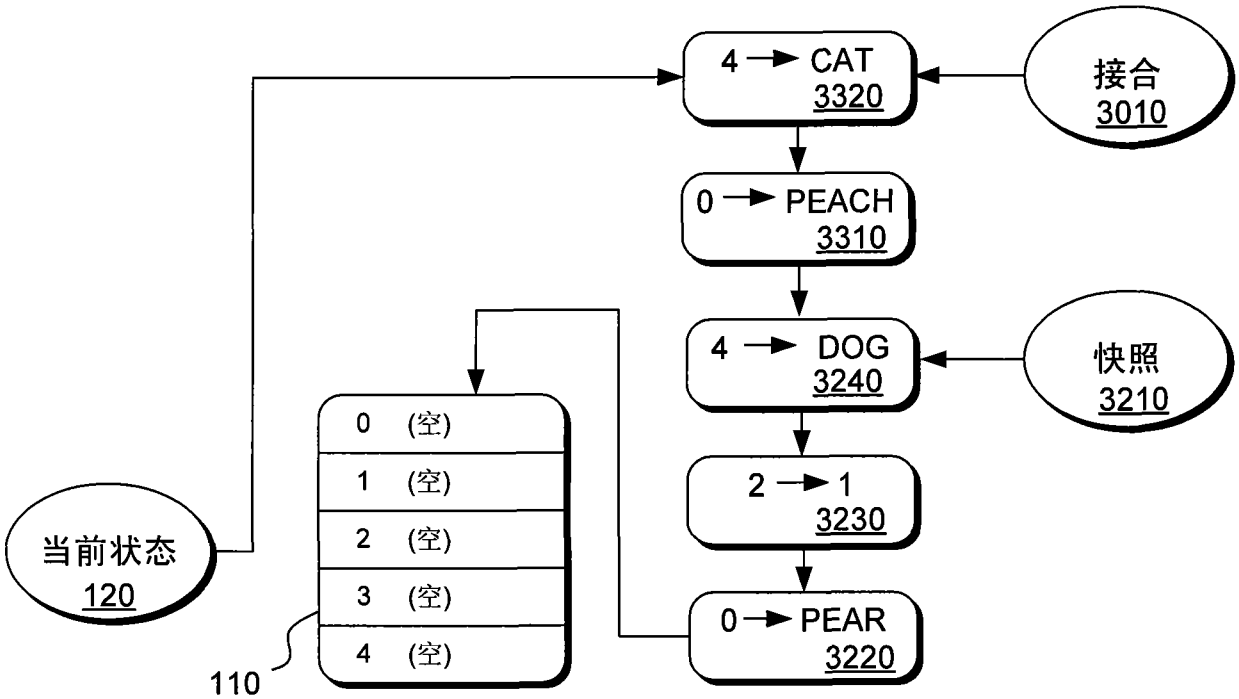

4 → CAT
3320
接合
3010
0 → PEACH
3310
4 → DOG
3240
快照
3210
2 → 1
3230
0 → PEAR
3220
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
当前状态
120
110

图 33

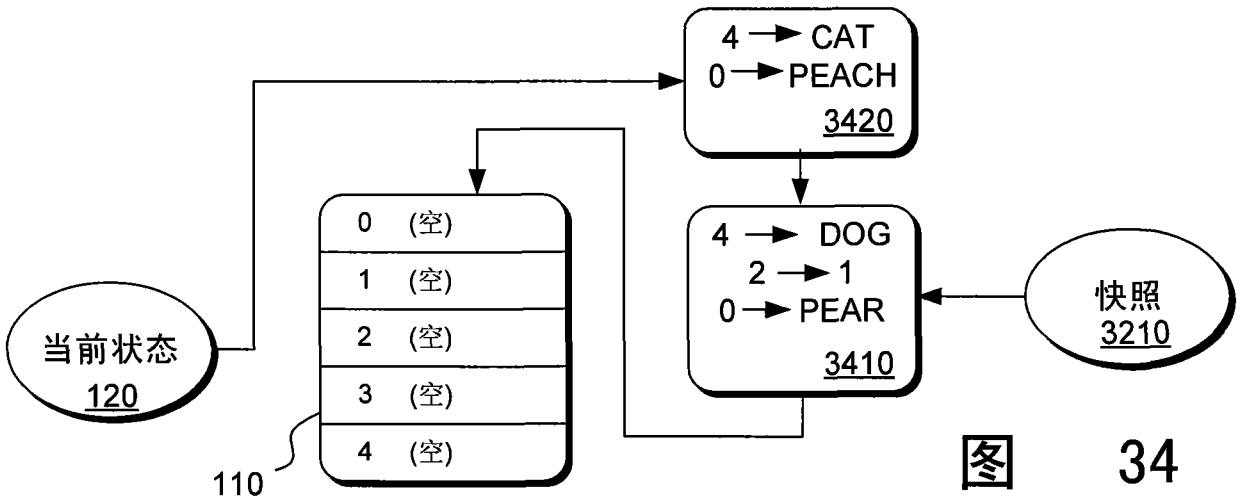

4 → CAT
0 → PEACH
3420
4 → DOG
2 → 1
0 → PEAR
3410
快照
3210
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
当前状态
120
110

图 34

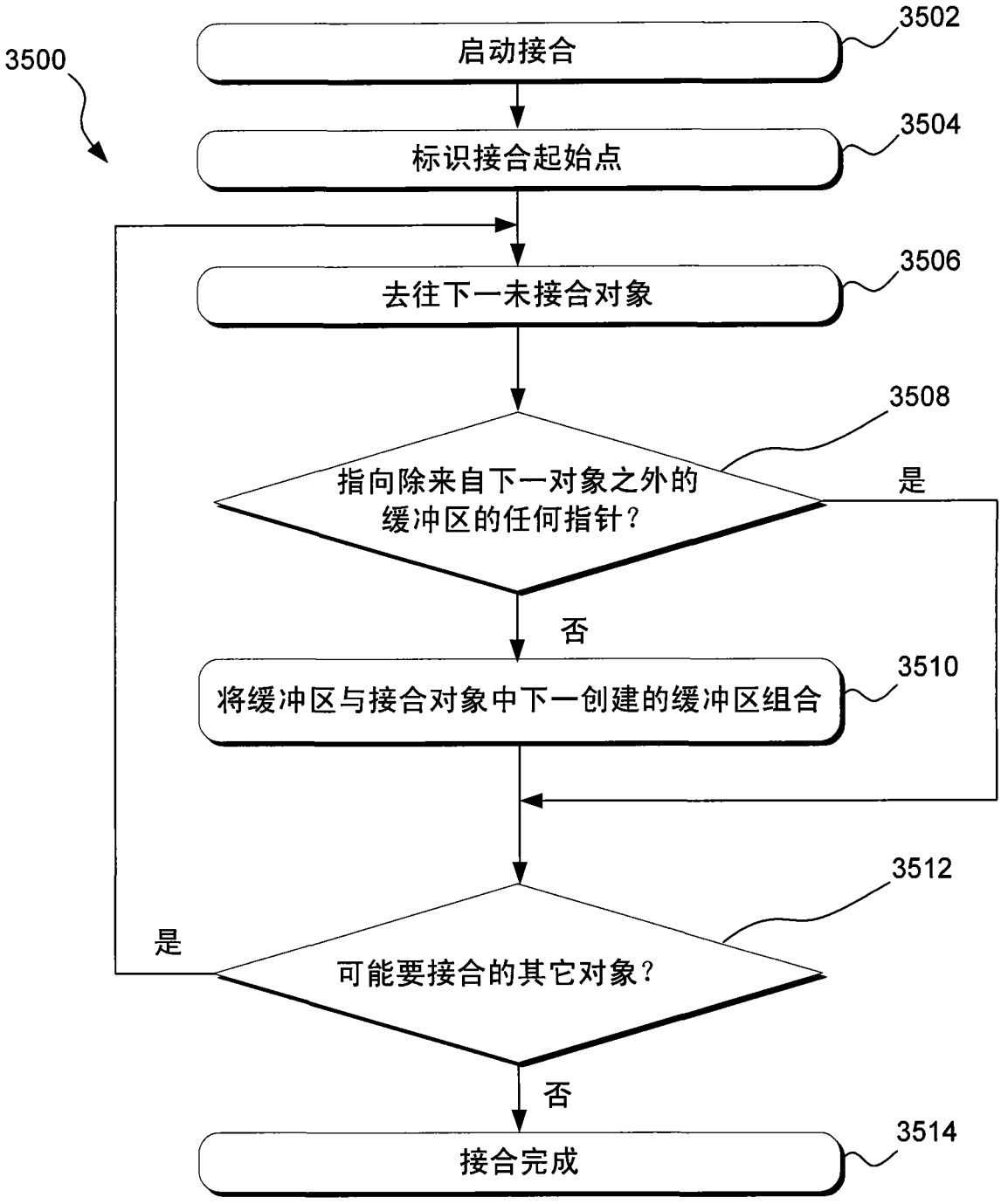

3500
启动接合
3502
标识接合起始点
3504
去往下一未接合对象
3506
指向除来自下一对象之外的缓冲区的任何指针？
3508
是
否
将缓冲区与接合对象中下一创建的缓冲区组合
3510
可能要接合的其它对象？
3512
是
否
接合完成
3514

图 35

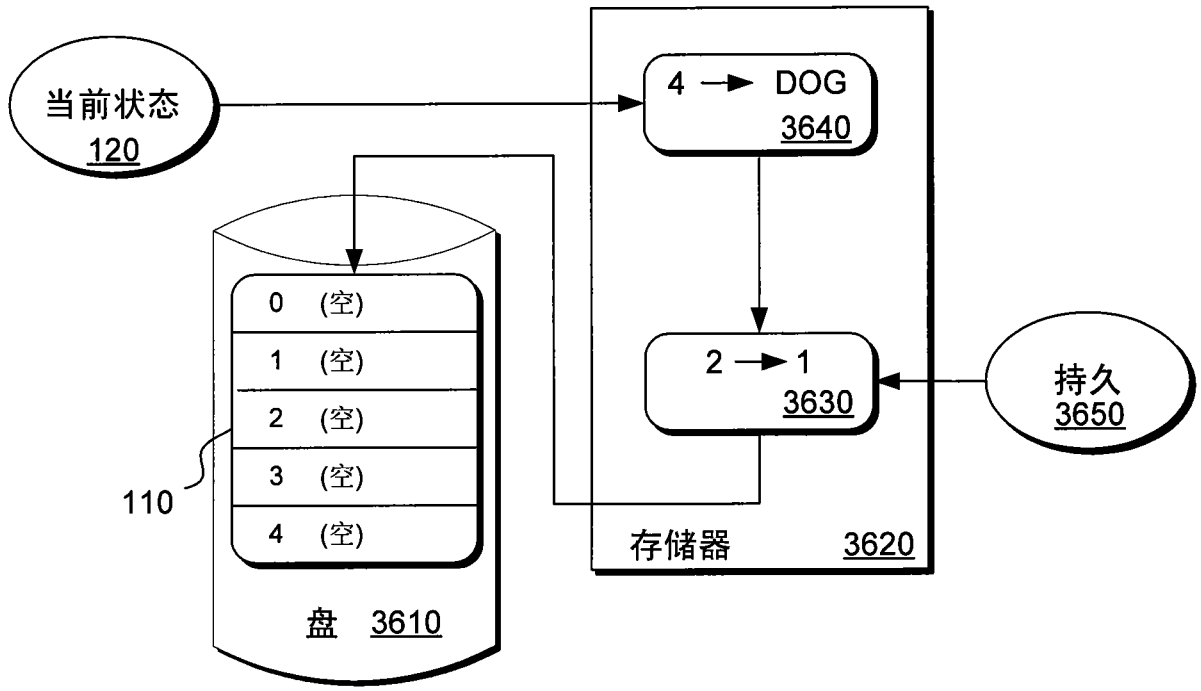

当前状态
120
4 → DOG
3640
2 → 1
3630
持久
3650
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110
存储器 3620
盘 3610

图 36

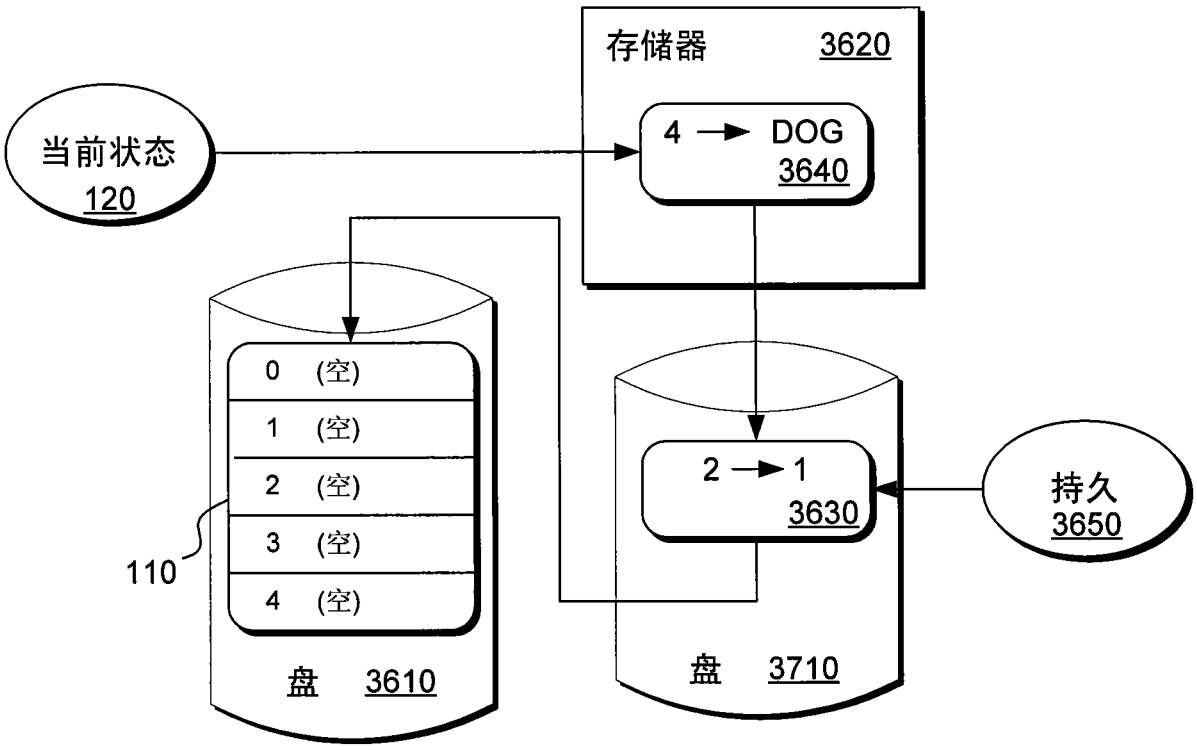

存储器 3620
当前状态
120
4 → DOG
3640
2 → 1
3630
持久
3650
0 (空)
1 (空)
2 (空)
3 (空)
4 (空)
110
盘 3610
盘 3710

图 37

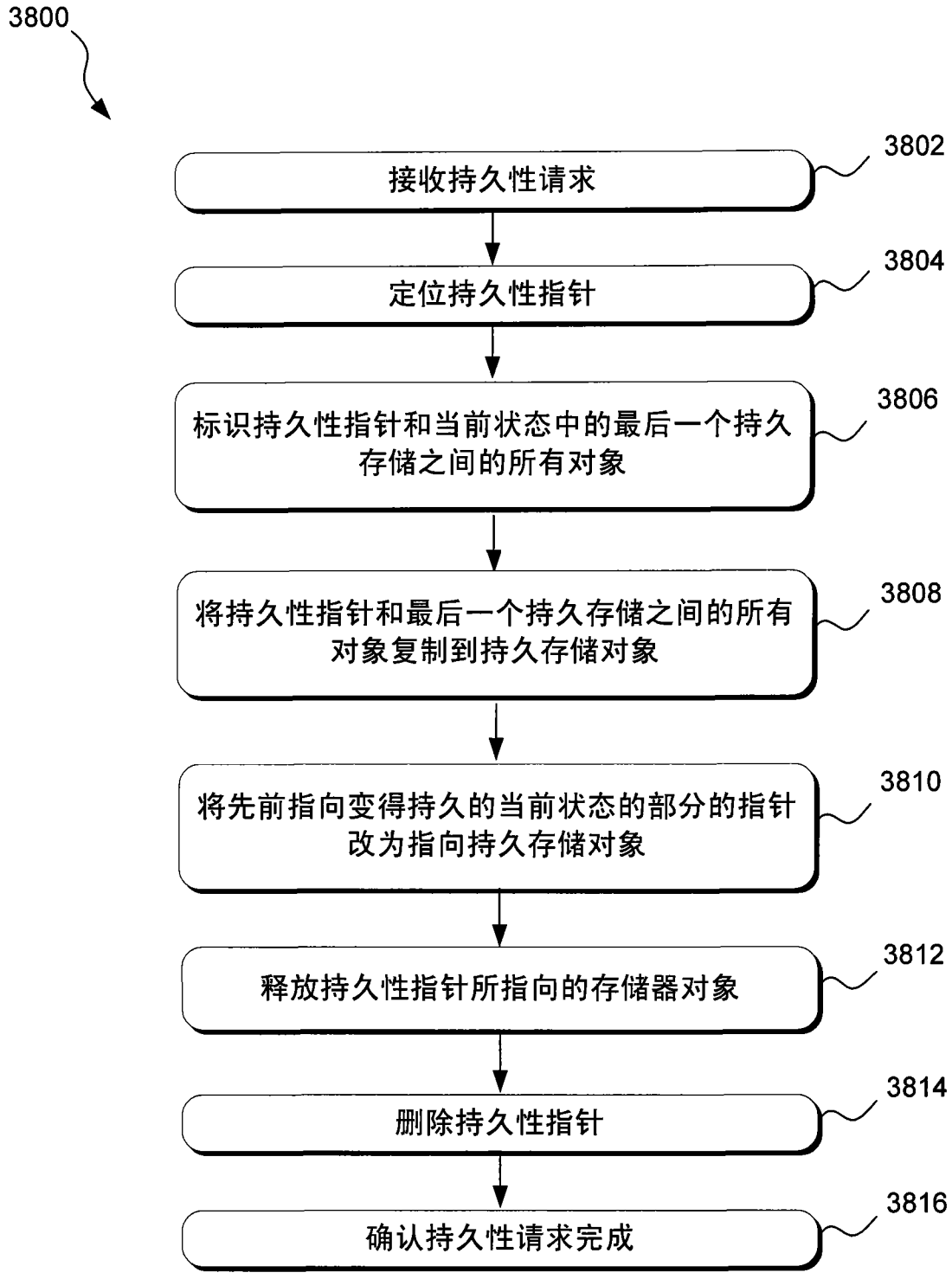

3800
接收持久性请求
3802
定位持久性指针
3804
标识持久性指针和当前状态中的最后一个持久存储之间的所有对象
3806
将持久性指针和最后一个持久存储之间的所有对象复制到持久存储对象
3808
将先前指向变得持久的当前状态的部分的指针改为指向持久存储对象
3810
释放持久性指针所指向的存储器对象
3812
删除持久性指针
3814
确认持久性请求完成
3816

图 38

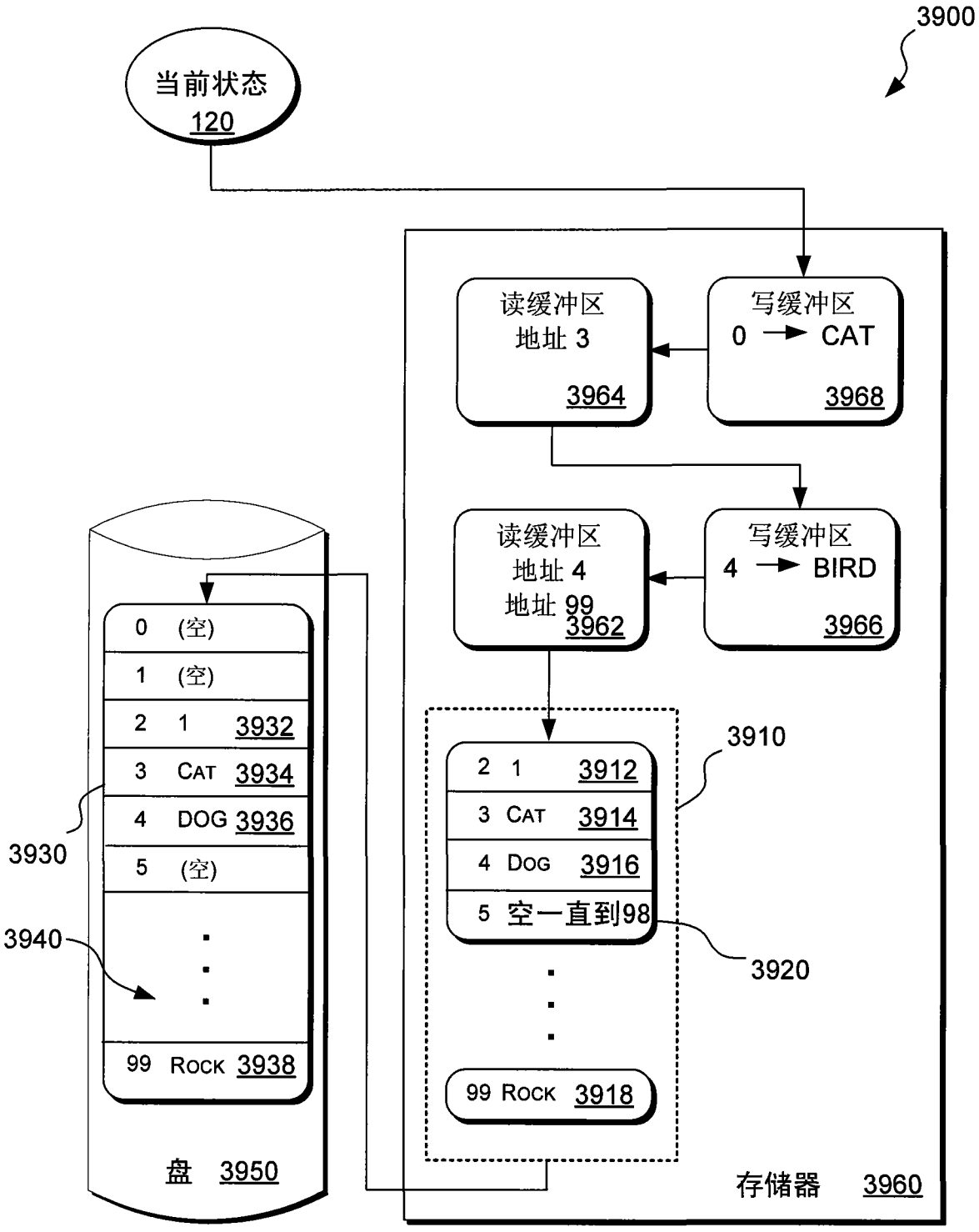

3900
当前状态
120
读缓冲区
地址 3
3964
写缓冲区
0 → CAT
3968
读缓冲区
地址 4
地址 99
3962
写缓冲区
4 → BIRD
3966
0 (空)
1 (空)
2 1 3932
3 CAT 3934
4 DOG 3936
5 (空)
99 ROCK 3938
3930
3940
盘 3950
2 1 3912
3 CAT 3914
4 DOG 3916
5 空一直到98
99 ROCK 3918
3910
3920
存储器 3960

图 39

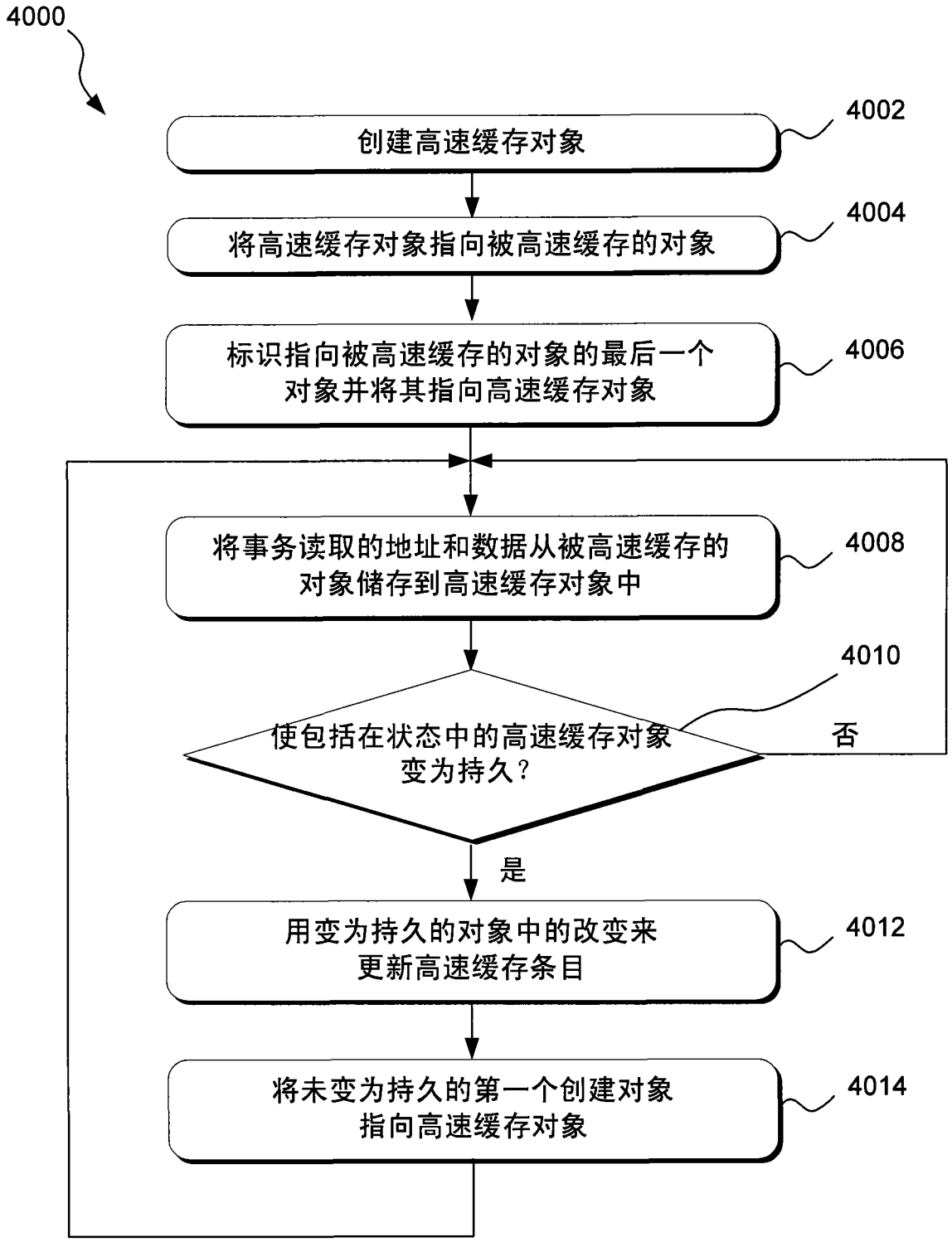

4000
创建高速缓存对象
4002
将高速缓存对象指向被高速缓存的对象
4004
标识指向被高速缓存的对象的最后一个对象并将其指向高速缓存对象
4006
将事务读取的地址和数据从被高速缓存的对象储存到高速缓存对象中
4008
使包括在状态中的高速缓存对象变为持久？
4010
否
是
用变为持久的对象中的改变来更新高速缓存条目
4012
将未变为持久的第一个创建对象指向高速缓存对象
4014

图 40

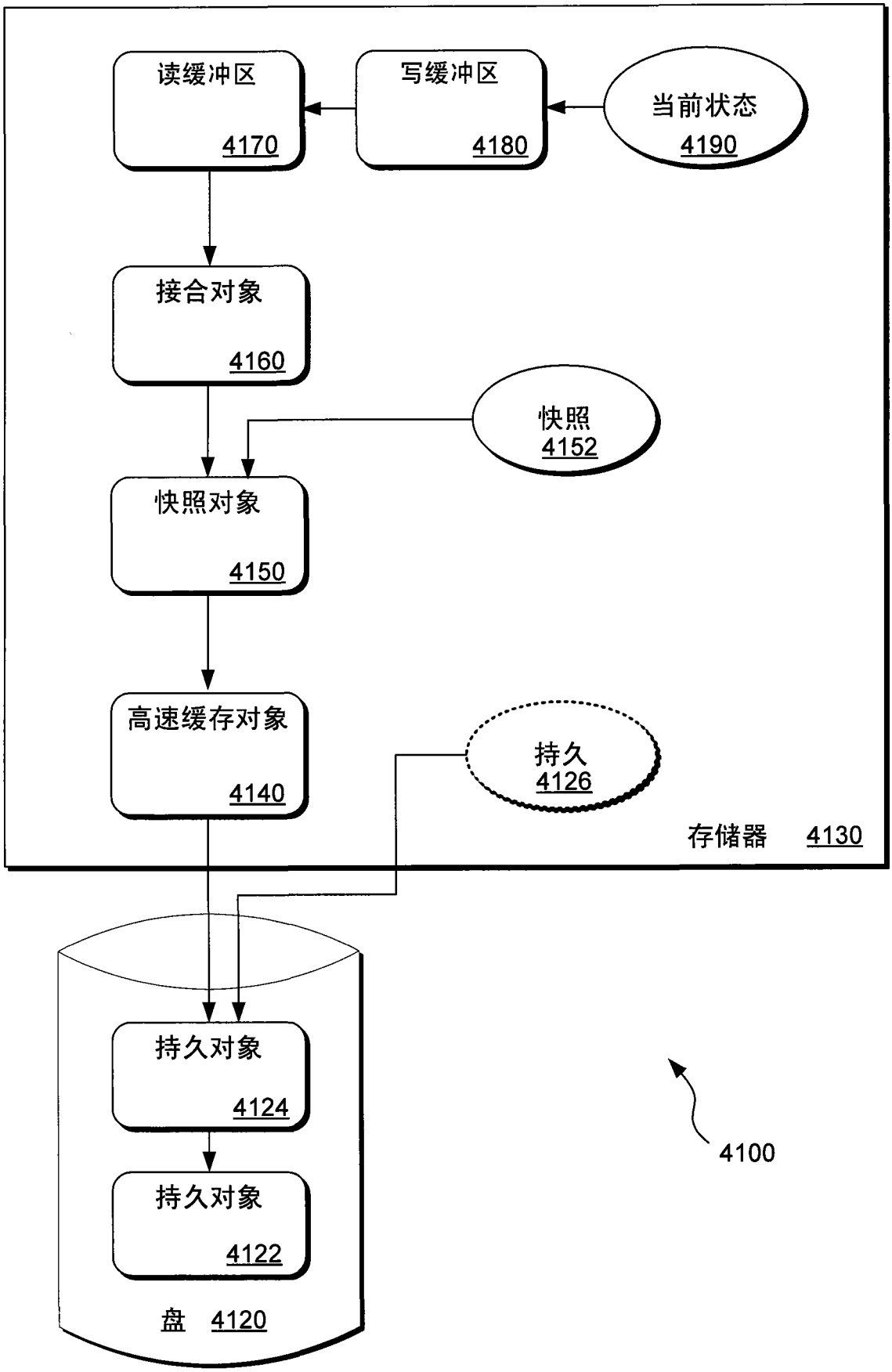

读缓冲区
4170
写缓冲区
4180
当前状态
4190
接合对象
4160
快照
4152
快照对象
4150
高速缓存对象
4140
持久
4126
存储器 4130
持久对象
4124
持久对象
4122
盘 4120
4100

图 41

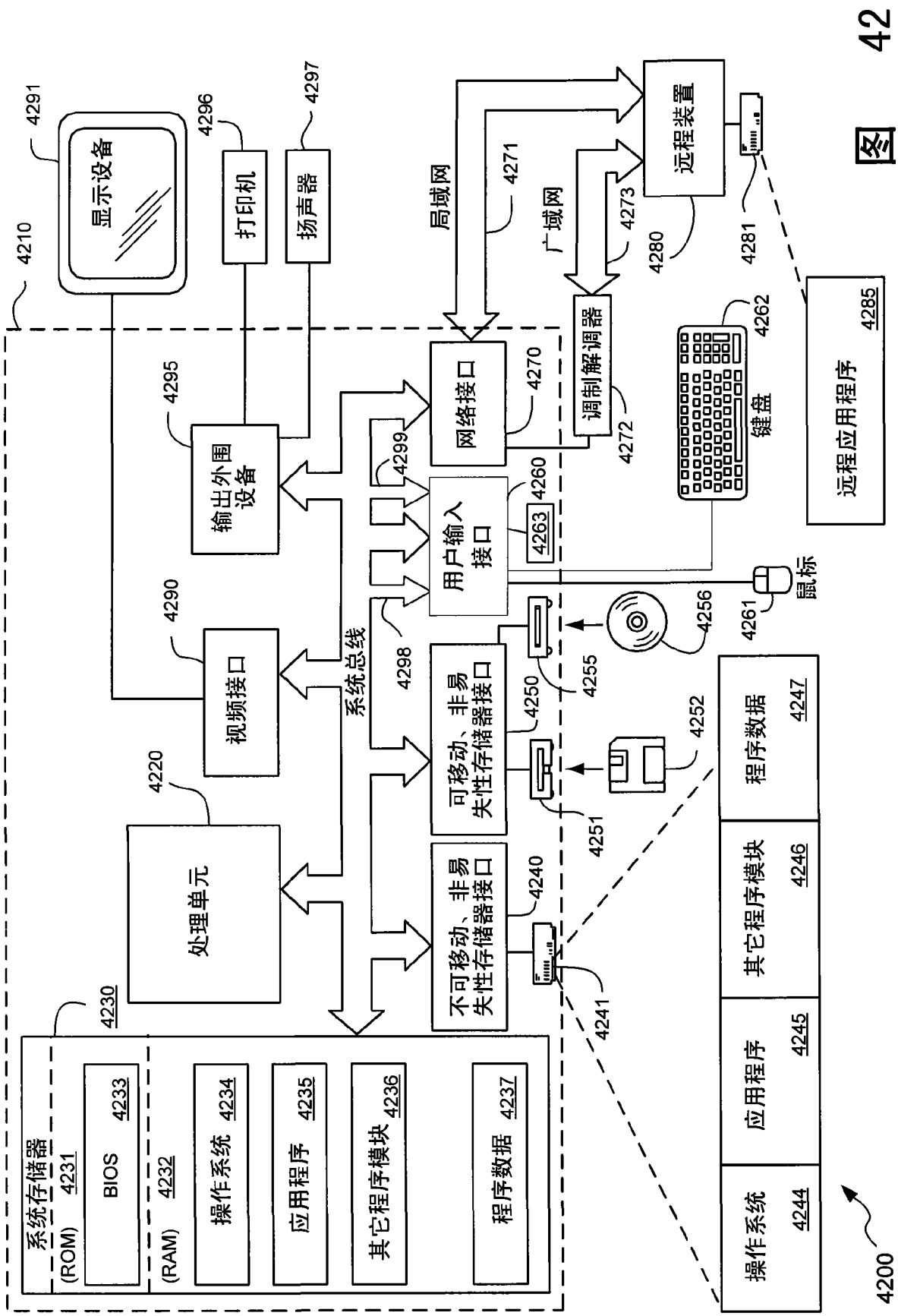

4200
操作系统 4244
应用程序 4245
其它程序模块 4246
程序数据 4247
4241
不可移动、非易失性存储器接口 4240
可移动、非易失性存储器接口 4250
4251
4252
4255
4256
处理单元 4220
视频接口 4290
系统总线 4298
用户输入接口 4260
4263
网络接口 4270
调制解调器 4272
4273
远程装置 4280
4281
远程应用程序 4285
键盘 4262
鼠标 4261
输出外围设备 4295
4299
局域网 4271
广域网
打印机 4296
扬声器 4297
显示设备 4291
4210
系统存储器 4230
(ROM) 4231
BIOS 4233
(RAM) 4232
操作系统 4234
应用程序 4235
其它程序模块 4236
程序数据 4237

图 42