因此，硬件缓存通常使用的是更严格的放置策略，这个策略将第 $k+1$ 层的某个块限制放置在第 k 层块的一个小的子集中(有时只是一个块)。例如，在图 6-22 中，我们可以确定第 $k+1$ 层的块 i 必须放置在第 k 层的块(i mod 4)中。例如，第 $k+1$ 层的块 0、4、8 和 12 会映射到第 k 层的块 0；块 1、5、9 和 13 会映射到块 1；依此类推。注意，图 6-22 中的示例缓存使用的就是这个策略。

这种限制性的放置策略会引起一种不命中，称为冲突不命中(conflict miss)，在这种情况中，缓存足够大，能够保存被引用的数据对象，但是因为这些对象会映射到同一个缓存块，缓存会一直不命中。例如，在图 6-22 中，如果程序请求块 0，然后块 8，然后块 0，然后块 8，依此类推，在第 k 层的缓存中，对这两个块的每次引用都会不命中，即使这个缓存总共可以容纳 4 个块。

程序通常是按照一系列阶段(如循环)来运行的，每个阶段访问缓存块的某个相对稳定不变的集合。例如，一个嵌套的循环可能会反复地访问同一个数组的元素。这个块的集合称为这个阶段的工作集(working set)。当工作集的大小超过缓存的大小时，缓存会经历容量不命中(capacity miss)。换句话说就是，缓存太小了，不能处理这个工作集。

4. 缓存管理

正如我们提到过的，存储器层次结构的本质是，每一层存储设备都是较低一层的缓存。在每一层上，某种形式的逻辑必须管理缓存。这里，我们的意思是指某个东西要将缓存划分成块，在不同的层之间传送块，判定是命中还是不命中，并处理它们。管理缓存的逻辑可以是硬件、软件，或是两者的结合。

例如，编译器管理寄存器文件，缓存层次结构的最高层。它决定当发生不命中时何时发射加载，以及确定哪个寄存器来存放数据。L1、L2 和 L3 层的缓存完全是由内置在缓存中的硬件逻辑来管理的。在一个有虚拟内存的系统中，DRAM 主存作为存储在磁盘上的数据块的缓存，是由操作系统软件和 CPU 上的地址翻译硬件共同管理的。对于一个具有像 AFS 这样的分布式文件系统的机器来说，本地磁盘作为缓存，它是由运行在本地机器上的 AFS 客户端进程管理的。在大多数时候，缓存都是自动运行的，不需要程序采取特殊的或显式的行动。

6.3.2　存储器层次结构概念小结

概括来说，基于缓存的存储器层次结构行之有效，是因为较慢的存储设备比较快的存储设备更便宜，还因为程序倾向于展示局部性：

- *利用时间局部性*：由于时间局部性，同一数据对象可能会被多次使用。一旦一个数据对象在第一次不命中时被复制到缓存中，我们就会期望后面对该目标有一系列的访问命中。因为缓存比低一层的存储设备更快，对后面的命中的服务会比最开始的不命中快很多。
- *利用空间局部性*：块通常包含有多个数据对象。由于空间局部性，我们会期望后面对该块中其他对象的访问能够补偿不命中后复制该块的花费。

现代系统中到处都使用了缓存。正如从图 6-23 中能够看到的那样，CPU 芯片、操作系统、分布式文件系统中和万维网上都使用了缓存。各种各样硬件和软件的组合构成和管理着缓存。注意，图 6-23 中有大量我们还未涉及的术语和缩写。在此我们包括这些术语和缩写是为了说明缓存是多么的普遍。