类似地，第 k 层的存储器被划分成较少的块的集合，每个块的大小与 $k+1$ 层的块的大小一样。在任何时刻，第 k 层的缓存包含第 $k+1$ 层块的一个子集的副本。例如，在图 6-22 中，第 k 层的缓存有 4 个块的空间，当前包含块 4、9、14 和 3 的副本。

数据总是以块大小为*传送单元*(transfer unit)在第 k 层和第 $k+1$ 层之间来回复制的。虽然在层次结构中任何一对相邻的层次之间块大小是固定的，但是其他的层次对之间可以有不同的块大小。例如，在图 6-21 中，L1 和 L0 之间的传送通常使用的是 1 个字大小的块。L2 和 L1 之间(以及 L3 和 L2 之间、L4 和 L3 之间)的传送通常使用的是几十个字节的块。而 L5 和 L4 之间的传送用的是大小为几百或几千字节的块。一般而言，层次结构中较低层(离 CPU 较远)的设备的访问时间较长，因此为了补偿这些较长的访问时间，倾向于使用较大的块。

1. 缓存命中

当程序需要第 $k+1$ 层的某个数据对象 d 时，它首先在当前存储在第 k 层的一个块中查找 d。如果 d 刚好缓存在第 k 层中，那么就是我们所说的*缓存命中*(cache hit)。该程序直接从第 k 层读取 d，根据存储器层次结构的性质，这要比从第 $k+1$ 层读取 d 更快。例如，一个有良好时间局部性的程序可以从块 14 中读出一个数据对象，得到一个对第 k 层的缓存命中。

2. 缓存不命中

另一方面，如果第 k 层中没有缓存数据对象 d，那么就是我们所说的*缓存不命中*(cache miss)。当发生缓存不命中时，第 k 层的缓存从第 $k+1$ 层缓存中取出包含 d 的那个块，如果第 k 层的缓存已经满了，可能就会覆盖现存的一个块。

覆盖一个现存的块的过程称为*替换*(replacing)或*驱逐*(evicting)这个块。被驱逐的这个块有时也称为*牺牲块*(victim block)。决定该替换哪个块是由缓存的*替换策略*(replacement policy)来控制的。例如，一个具有*随机替换策略*的缓存会随机选择一个牺牲块。一个具有最近最少被使用(LRU)替换策略的缓存会选择那个最后被访问的时间距现在最远的块。

在第 k 层缓存从第 $k+1$ 层取出那个块之后，程序就能像前面一样从第 k 层读出 d 了。例如，在图 6-22 中，在第 k 层中读块 12 中的一个数据对象，会导致一个缓存不命中，因为块 12 当前不在第 k 层缓存中。一旦把块 12 从第 $k+1$ 层复制到第 k 层之后，它就会保持在那里，等待稍后的访问。

3. 缓存不命中的种类

区分不同种类的缓存不命中有时候是很有帮助的。如果第 k 层的缓存是空的，那么对任何数据对象的访问都会不命中。一个空的缓存有时被称为*冷缓存*(cold cache)，此类不命中称为*强制性不命中*(compulsory miss)或*冷不命中*(cold miss)。冷不命中很重要，因为它们通常是短暂的事件，不会在反复访问存储器使得缓存*暖身*(warmed up)之后的稳定状态中出现。

只要发生了不命中，第 k 层的缓存就必须执行某个*放置策略*(placement policy)，确定把它从第 $k+1$ 层中取出的块放在哪里。最灵活的替换策略是允许来自第 $k+1$ 层的任何块放在第 k 层的任何块中。对于存储器层次结构中高层的缓存(靠近 CPU)，它们是用硬件来实现的，而且速度是最优的，这个策略实现起来通常很昂贵，因为随机地放置块，定位起来代价很高。