发生了彻底的变化。如果现在的地址空间是2^{64}字节，页面大小为4KB，我们需要一个有2^{52}个表项的页表。如果每一个表项8个字节，那么整个页表就会超过3000万GB（30PB）。仅仅为页表耗费3000万GB不是个好主意（现在不是，可能以后几年也不是）。因而，具有64位分页虚拟地址空间的系统需要一个不同的解决方案。

解决方案之一就是使用倒排页表（inverted page table）。在这种设计中，在实际内存中每一个页框有一个表项，而不是每一个虚拟页面有一个表项。例如，对于64位虚拟地址，4KB的页，1GB的RAM，一个倒排页表仅需要262 144个页表项。表项记录哪一个（进程，虚拟页面）对定位于该页框。

虽然倒排页表节省了大量的空间（至少当虚拟地址空间比物理内存大得多的时候是这样的），但它也有严重的不足：从虚拟地址到物理地址的转换会变得很困难。当进程n访问虚拟页面p时，硬件不再能通过把p当作指向页表的一个索引来查找物理页框。取而代之的是，它必须搜索整个倒排页表来查找某一个表项(n, p)。此外，该搜索必须对每一个内存访问操作都要执行一次，而不仅仅是在发生缺页中断时执行。每一次内存访问操作都要查找一个256K的表是不会让你的机器运行得很快的。

走出这种两难局面的办法是使用TLB。如果TLB能够记录所有频繁使用的页面，地址转换就可能变得像通常的页表一样快。但是，当发生TLB失效时，需要用软件搜索整个倒排页表。一个可行的实现该搜索的方法是建立一张散列表，用虚拟地址来散列。当前所有在内存中的具有相同散列值的虚拟页面被链接在一起，如图3-14所示。如果散列表中的槽数与机器中物理页面数一样多，那么散列表的冲突链的平均长度将会是1个表项，这将会大大提高映射速度。一旦页框号被找到，新的（虚拟页号，物理页框号）对就会被装载到TLB中。

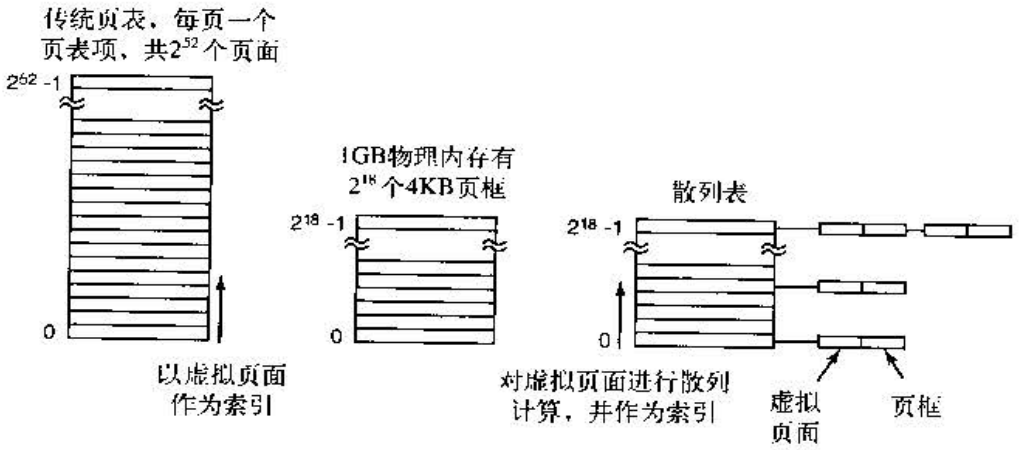

图3-14 传统页表与倒排页表的对比

倒排页表在64位机器中很常见，因为在64位机器中即使使用了大页面，页表项的数量还是很庞大的。例如，对于4MB页面和64位虚拟地址，需要2^{42}个页表项。处理大虚存的其他方法可参见Talluri等人的论文（1995）。

3.4 页面置换算法

当发生缺页中断时，操作系统必须在内存中选择一个页面将其换出内存，以便为即将调入的页面腾出空间。如果要换出的页面在内存驻留期间已经被修改过，就必须把它写回磁盘以更新该页面在磁盘上的副本；如果该页面没有被修改过（如一个包含程序正文的页面），那么它在磁盘上的副本已经是最新的，不需要回写。直接用调入的页面覆盖掉被淘汰的页面就可以了。

当发生缺页中断时，虽然可以随机地选择一个页面来置换，但是如果每次都选择不常使用的页面会提升系统的性能。如果一个被频繁使用的页面被置换出内存，很可能它在很短时间内又要被调入内存，这会带来不必要的开销。人们已经从理论和实践两个方面对页面置换算法进行了深入的研究。下面我们将介绍几个最重要的算法。

有必要指出，“页面置换”问题在计算机设计的其他领域中也会同样发生。例如，多数计算机把最近使用过的32字节或64字节的存储块保存在一个或多个高速缓存中。当这些高速缓存存满之后就必须选