B. 最糟糕情况随机写(303MB/s)：$(10^9 \times 128) \times (1/303) \times (1/(86\,400 \times 365)) \approx 13$ 年。

C. 平均情况(20GB/天)：$(10^9 \times 128) \times (1/20\,000) \times (1/365) \approx 17.535$ 年。

所以即使 SSD 连续工作，也能持续至少 8 年时间，这大于大多数计算机的预期寿命。

6.6　在 2005 年到 2015 年的 10 年间，旋转磁盘的单位价格下降了大约 166 倍，这意味着价格大约每 18 个月下降 2 倍。假设这个趋势一直持续，1PB 的存储设备，在 2015 年花费 30 000 美元，在 7 次这种 2 倍的下降之后会降到 500 美元以下。因为这种下降每 18 个月发生一次，我们可以预期在大约 2025 年，可以用 500 美元买到 1PB 的存储设备。

6.7　为了创建一个步长为 1 的引用模式，必须改变循环的次序，使得最右边的索引变化得最快：

```
1   int sumarray3d(int a[N][N][N])
2   {
3       int i, j, k, sum = 0;
4
5       for (k = 0; k < N; k++) {
6           for (i = 0; i < N; i++) {
7               for (j = 0; j < N; j++) {
8                   sum += a[k][i][j];
9               }
10          }
11      }
12      return sum;
13  }
```

这是一个很重要的思想。要保证你理解了为什么这种循环次序改变就能得到一个步长为 1 的访问模式。

6.8　解决这个问题的关键在于想象出数组是如何在内存中排列的，然后分析引用模式。函数 clear1 以步长为 1 的引用模式访问数组，因此明显地具有最好的空间局部性。函数 clear2 依次扫描 N 个结构中的每一个，这是好的，但是在每个结构中，它以步长不为 1 的模式跳到下列相对于结构起始位置的偏移处：0、12、4、16、8、20。所以 clear2 的空间局部性比 clear1 的要差。函数 clear3 不仅在每个结构中跳来跳去，而且还从结构跳到结构，所以 clear3 的空间局部性比 clear2 和 clear1 都要差。

6.9　这个解答是对图 6-26 中各种高速缓存参数定义的直接应用。不那么令人兴奋，但是在能真正理解高速缓存如何工作之前，你需要理解高速缓存的结构是如何导致这样划分地址位的。

高速缓存	m	C	B	E	S	t	s	b
1.	32	1024	4	1	256	22	8	2
2.	32	1024	8	4	32	24	5	3
3.	32	1024	32	32	1	27	0	5

6.10　填充消除了冲突不命中。因此，四分之三的引用是命中的。

6.11　有时候，理解为什么某种思想是不好的，能够帮助你理解为什么另一种是好的。这里，我们看到的坏的想法是用高位来索引高速缓存，而不是用中间的位。

A. 用高位做索引，每个连续的数组片(chunk)由 2^t 个块组成，这里 t 是标记位数。因此，数组头 2^t 个连续的块都会映射到组 0，接下来的 2^t 个块会映射到组 1，依此类推。

B. 对于直接映射高速缓存(S，E，B，m)=(512，1，32，32)，高速缓存容量是 512 个 32 字节的块，每个高速缓存行中有 $t=18$ 个标记位。因此，数组中头 2^{18} 个块会映射到组 0，接下来 2^{18} 个块会映射到组 1。因为我们的数组只由 $(4096 \times 4)/32 = 512$ 个块组成，所以数组中所有的块都被映射到组 0。因此，在任何时刻，高速缓存至多只能保存一个数组块，即使数组足够小，能够完全放到高速缓存中。很明显，用高位做索引不能充分利用高速缓存。

6.12　两个低位是块偏移(CO)，然后是 3 位的组索引(CI)，剩下的位作为标记(CT)。