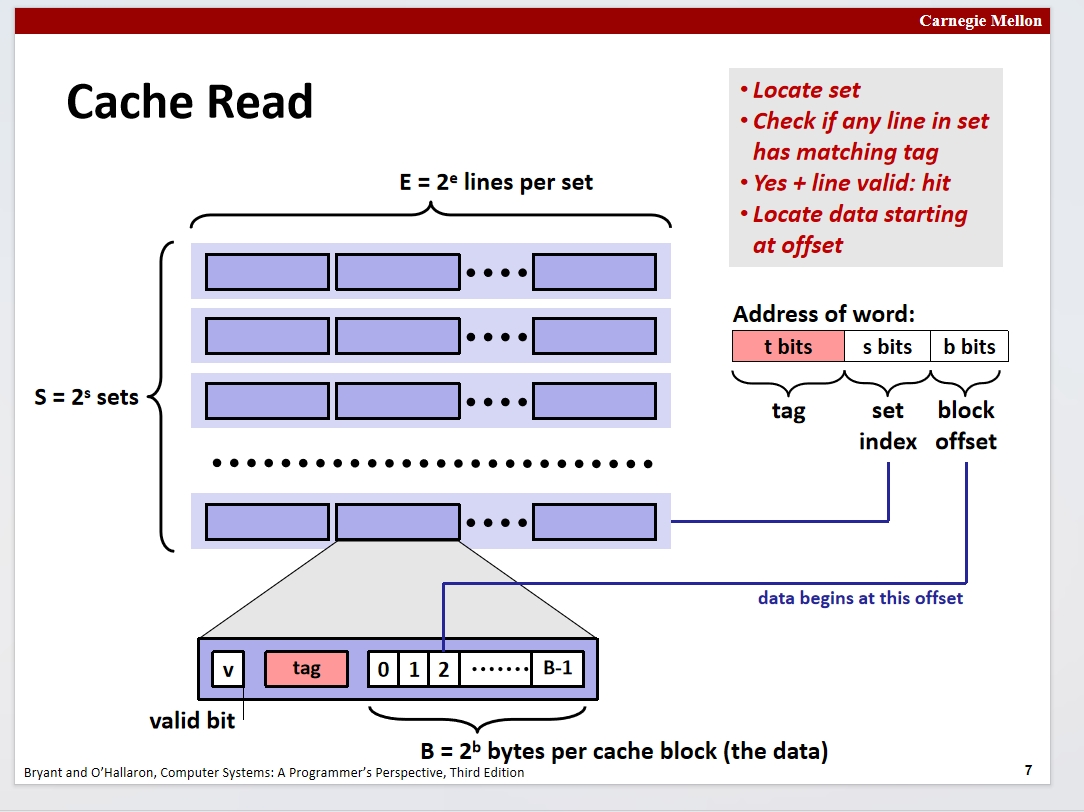


接下来来了解下载计组中提到过的缓存器的组织，首先，整个缓存被分为多个set，如上图，一行一个set，每一行即每一个set中又分为多个块，对于每个块，又细分为三个部分

分析下单个块的信息，一般，一个块中有三部分的信息，首先是一个有效位来表示该数据是否有效，接下来是一个映射位，用来表示这个快对应的具体的主存中的对应位置，接下来就是数据位，这个就是该缓存中实际储存数据的区域。具体部分等之后计组库再进行补充吧



接下来分析下cpu取缓存中的情况，首先，CPU会将一段地址发送给缓存并要求缓存返回这段地址所对应的信息

这段字地址分为三部分，如上图中的Address of word

有t位标签位Tag，s位标签位set Index，b位标签位block offset

缓存对于此地址的使用：首先，缓存先接收s位的标签位，来判断该目标地址所在组，若该地址在现有的缓存中。接下来开始逐个检查该set中每一个块的tag来判断是否存在一个与目标地址相符合的元素，接下来还会检查块中元素的valid bit来检查该块是否有效，一般1为有效，0则为无效，如果标签匹配并且有效位也为1，那么这个快就是一个有效的块，这是可以视作是一个**命中**的情况

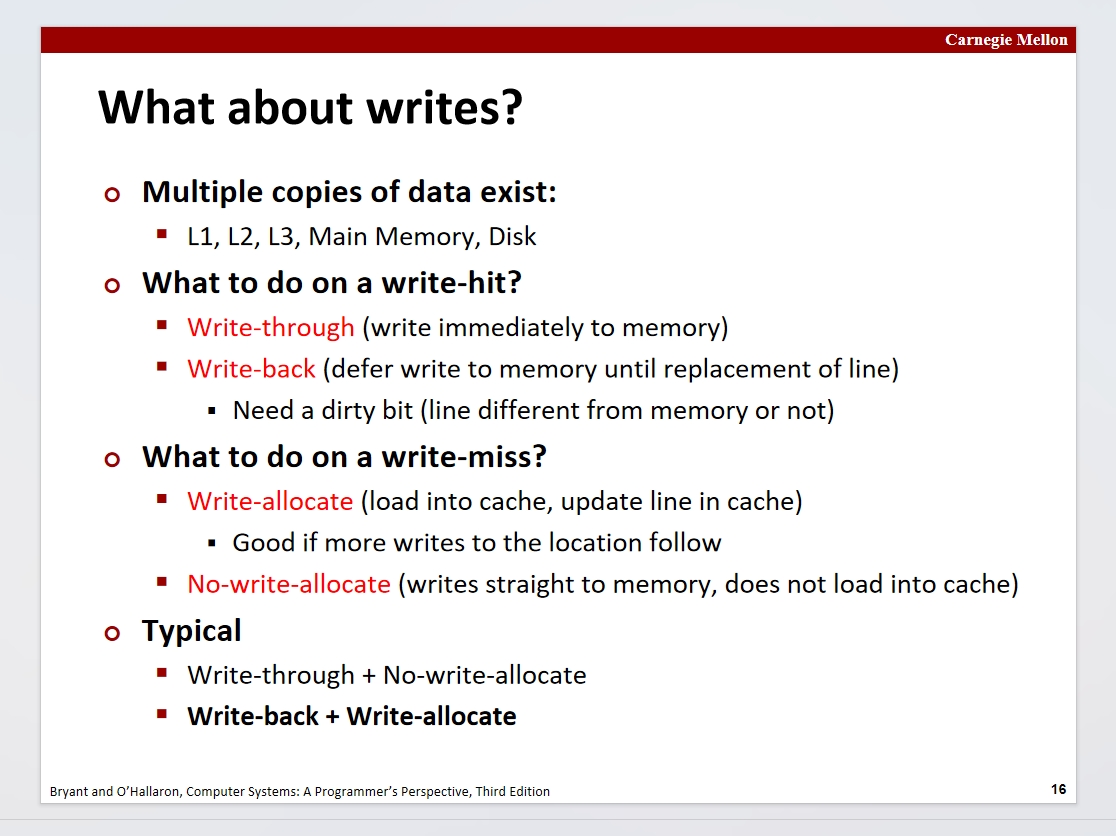
一但命中了一个缓存块，接下来缓存会通过b位的偏移块来进行数据位置的确定

这里再分析下读取多个字节的情况(硬控我十几分钟)

对于多字节的读取，CPU发送给缓存的地址还是跟单字节的读取的架构一样，数据域还是一个相对于块的数据域的偏移量，至于怎么区分几个字节的读取，这个取决于操作系统和CPU的指令集架构等硬件的配合，就比如movq等指令，指令的后缀指示了要操作的数的大小，操作系统会隐式的将要读取的数据设置为这个后缀所对应的字节数，而这个是在CPU发送给缓存的地址中所没有体现的，但他是确实存在的，通过操作系统和硬件等的协作，可以实现不同字节数的数据的读取

也就是说，这个多少个字节数的读取，其实是一个高层次的抽象，由编译器已经给我们实现了，不需要我们去管理这个细节

接下来就进行缓存与主存间的分析



当我们对缓存中的数据进行了替换时，(当我们CPU对数据进行了修改的时候，它会先修改缓存中对应的数据)，但是对于主存中的数据修改，这分为俩种情况:**写回**和**不写回**

其实也很简单，对于**不写回**的情况，就是始终保证主存和缓存中的数据保持一致，当我们对缓存中的数据进行修改的时候，系统会自动的对主存中的数据进行同步，但是这样存在一定的效率损失，因为对于主存的访问以及修改相对来说太慢了，但我们需要频繁的对数据进行修改时，这种写回的操作会很耗费时间。另一种方式是**写回**，这种方式即是跟写回的方式相反，不会一直保持主存和缓存中的数据一致，相反，它缓存中的数据只有在特定时候才会被写回到主存中去。在这种方式下，缓存需要一些额外的位来记录当前缓存中的数据是否被CPU所修改，当被修改时，缓存中对应位的数据会被记录位已修改的状态，但是并不会直接写回到主存中，只有当程序结束时，系统才会去识别这些标志位，并将那些被修改了的位写回到缓存中去，这就是**写回**的情况

还有就是写分配的情况，当程序要对数据进行写并且该数据并不在缓存中时有俩种情况，一种就是通过算法将要写的数据写入到缓存中，然后在写入到主存中，这样使得之后一段时间内如果要使用该数据能直接从缓存中去读取，另一种就是不写如缓存，直接写入到主存中，至于使用哪种方法，这个已经封装抽象好了

接下来进入到一个重点，编写一个缓存友好的代码

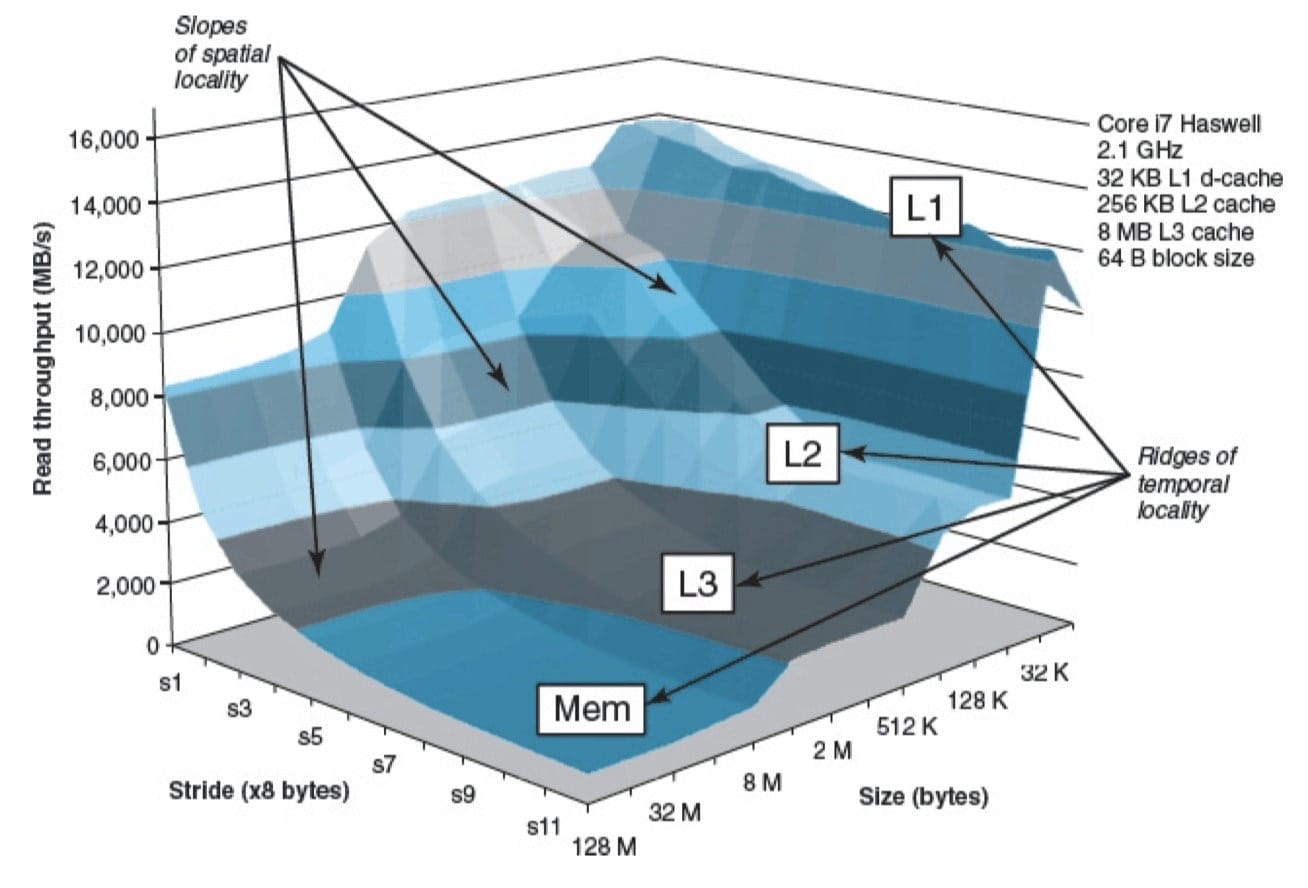
为什么要编写缓存友好的代码，第一步简单来讲，就是说这能极大的优化效率。已知，从缓存中读取数据是比从主存中读取数据快的多的，那么如果我们能编写出尽可能提高缓存利用率的代码，就能极大的优化程序的效率，就比如一个for循环，如果我们只是用到了几个局部变量，那么这个的效率是相当高的。相反，如果我们使用了一系列没有逻辑的变量，那么这个需要缓存的不断改变，则会显著的降低效率

就比如在一个函数中使用全局变量，由于程序不知道这个全局变量会发生什么，所以程序无法将这个全局变量放到寄存器中。相反，重复的使用堆栈中的变量是比较友好的，因为这些可以被放于寄存器中，这个速度能有很大的提升。又如，逐元素的访问数组的元素是友好的，由于块的存在，每次缓存能够命中的元素能与接下来可能访问到的元素有良好的匹配

就比如64位大小的块，每次能储存俩个int型的数据，这如果对一个int型数组进行逐元素访问，这个缓存命中率是可观的，这是步长为1的情况。而在在这种块大小的情况下，如果步长为2，那么这个效率会劣化到50%

**接下来进入内存山的概念**

首先



首先，明确下这张图中的几个坐标的属性，其中stride属性是一次访问的步长，即一次跳几个元素。Size属性可以理解为每次传递时要读取的元素总量，某种意义上说，这个反映了在一次内存访问中涉及到的数据量或数据结构的大小。Stride属性是指在访问数组或其他数据结构时访问相邻元素之间的距离。这是因为在缓存中，相邻元素一般会被加载到一个块中，当这个stride较小时，能够更好的利用到缓存，提高缓存的命中率。

在内存山图中可以看到，对于一个系统，它的stride越小，size越小，其的内存传输效率越高，这是因为在这种情况下可以很好的符合缓存的局部性原理，小步长是得缓存的块的储存多个字节的特性能更好的命中，size越小能使得缓存能够容纳对应的数据等，使得不必去频繁地访问主存去寻找数据。因此可以看到，在stride位于这里的最小，size位于这里的最小是，传输速率达到了最大值

接下来说一下内存山中的几个性质：

1~山脊线 一般来说，各个缓存级别都有一条山脊线，其中最高的一条应该是为于最上层的山脊线，这个山脊线是在一级缓存的，且这条山脊线代表的效率应该是最高的。一般来说，山脊线理论上应该是水平的，但是由于一些时间函数等的会导致绘制出来的山脊线不是很水平