

对于随机访问储存器(RAM)

简单有俩个分类:  
1~静态储存器 SRAM 这里的静态是指SRAM不需要不断刷新来保护数据，只在数据更新时刷新一次，这个在断电时其储存的数据会丢失，且速度较之另外一个储存器更快，通常被用作高速缓存，集成在芯片中

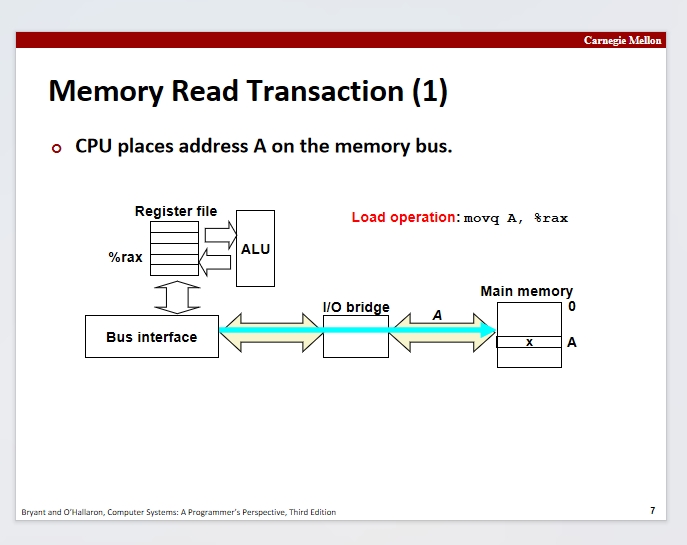
2~动态储存器 DRAM 一直在刷新以保证数据的准确性这个在断电时其储存的数据会丢失，且速度更慢，通常用作低速缓存。与主存在一起

现代RAM的一种:  
**闪存**~现代只读存储器的一种形式，

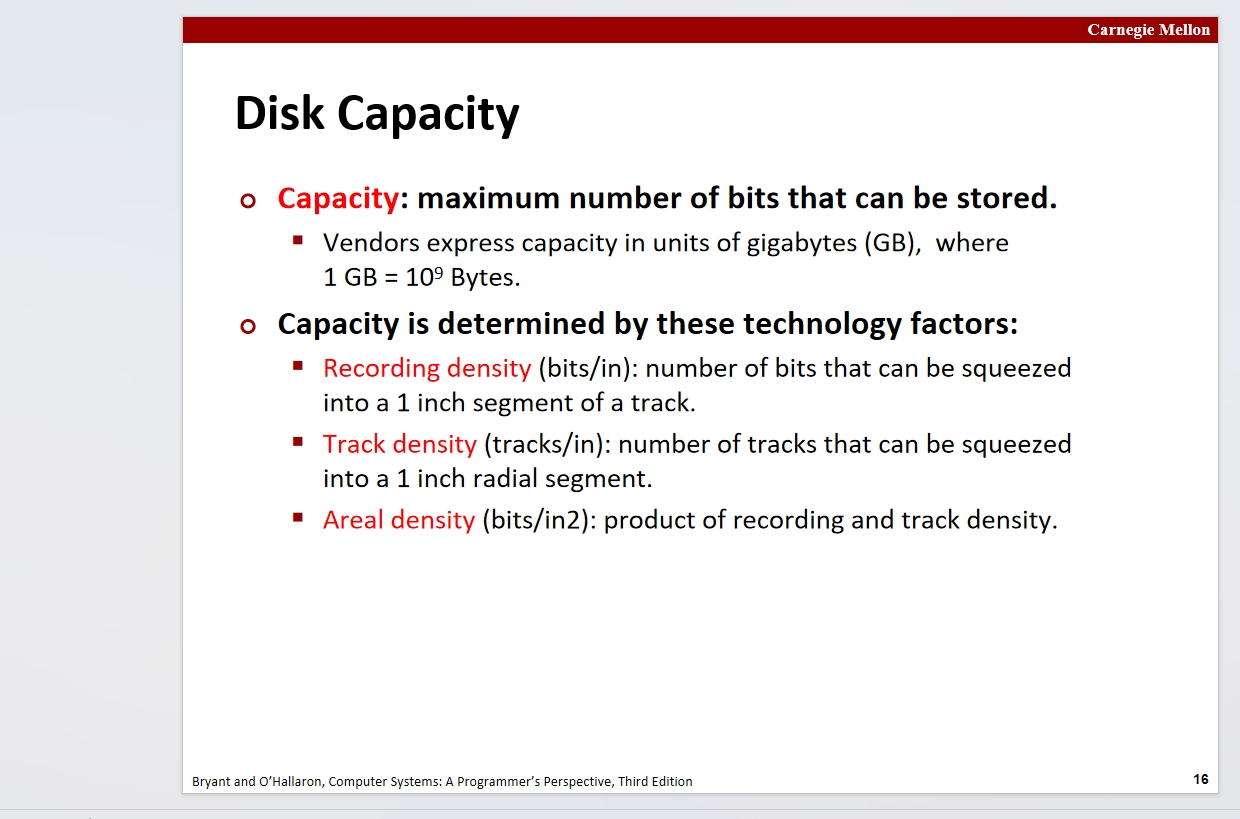
特性：可写性：传统的ROM通常不可修改，但闪存可写，可以多次读写，用户可在 闪存上储存和删除数据

速度：闪存的读写速度比传统的ROM更快

耐用性：闪存的写入次数有限，当写入等操作有差不多10w次时会差不多 失效，现代技术通过穿戴平衡和错误校正等方法来延长其使用寿 命。



这里简单介绍下数据在底层上的传递，就比如这里的mov指令，当cpu接到该指令的时候，会将A的地址写到总线上，该地址经主线到达主存，主存读取这个地址，然后等待数据（这里是目标的A值）到达系统主线，请注意，这里指令是将一个A值放到寄存器%rax中，当主存接收到数据之后，会将数据写入到之前传进来的地址处，至此，就完成了mov指令对于一个数据的移动



简单介绍下磁盘的物理层次，首先，先将磁盘划分为一个个同心圆环，这一个个圆环就是磁道，接着，将这些同心圆环再依次细分为多个面积相同的扇区，由于要划分出区域，相邻片区间的空隙是需要考虑的

这里简单复习下计组方面的东西（磁表面储存器的主要技术指标）：  
1~**记录密度** 决定单独一个扇区或者说单位长度内能储存多少比特(二进制信息量)，一个磁道可以储存对应的0/1信息

2~**储存容量** 外存能储存的二进制信息总数量

3.4.5....

接下来分析下磁道，扇区

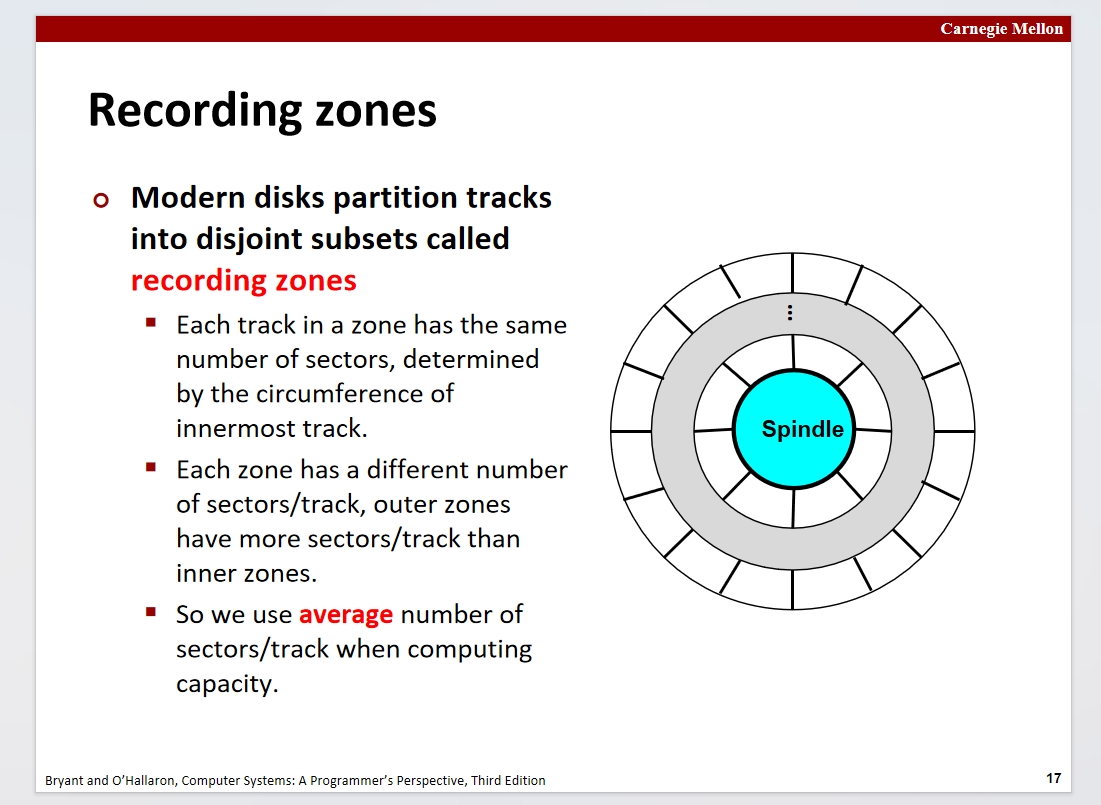
对于一个磁盘，其上的所有扇区面积都是相同的，且每个磁道上的扇区数量通常也是一致的。而又由于磁道在磁盘上的物理性质，这就导致要保证不同位置上的磁道上的扇区要保证面积一致，最好需要通过调整一些来实现，有一种固定扇区的技术，对于每个磁道上的扇区数量，都是一致的

但是，还有技术会通过改变每个磁道间的扇区数量来实现而不是上面那个改变扇区直径来实现，如下图。在这种技术中，每个磁道中的扇区数并不是一个常数，相反，他们通常与他们所处磁道的同心圆半径成正比，磁道距离圆心越远，该磁道能划分出的扇区越多

这种技术就是变扇区技术，可以更好地利用磁盘的表面空间，尤其是在外圈可以容纳更多数据的情况下。这种设计可以提高数据存储的效率和读写速度。

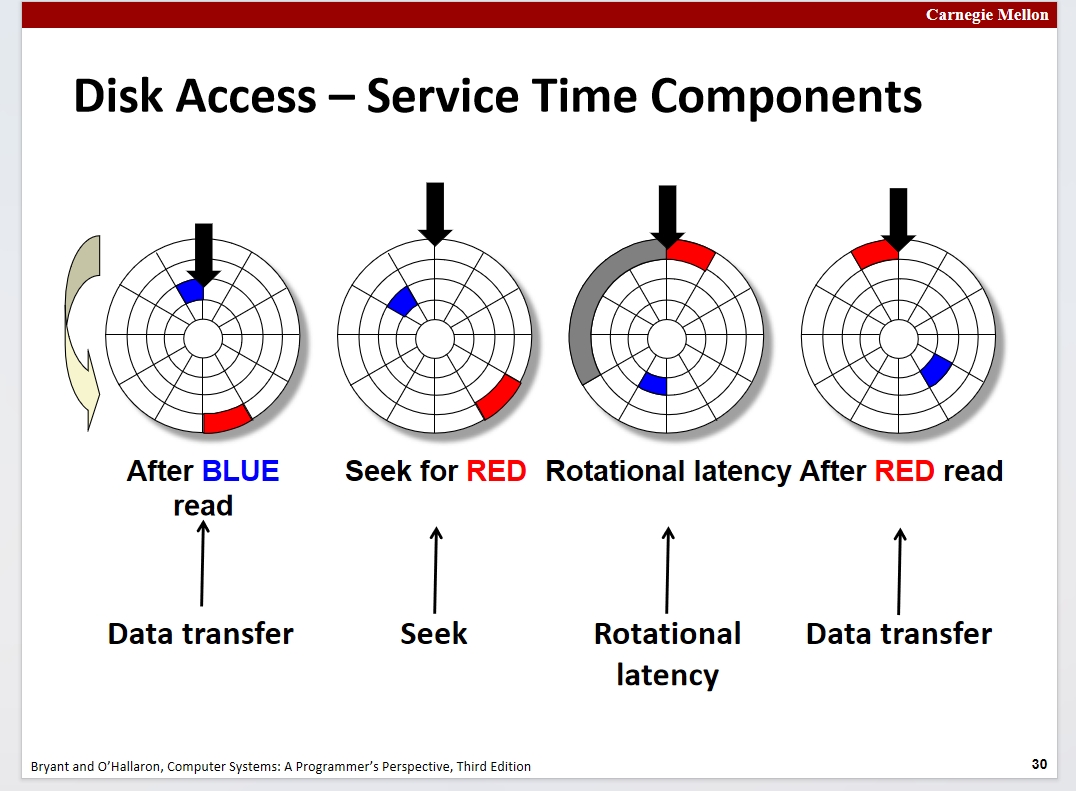
变扇区技术相对于扇区固定的情况，其最主要的优势就是可以在外层磁道划分出更多的扇区，能够更有效的利用磁盘表面的储存空间等等

无论对于哪种技术，每个磁盘上的磁道直径都是一样的，也就是说每个磁道，如果划分数量可以是动态的，就可以更好的利用每一个磁道上的面积，不然如果磁道数量固定且由于外层的面积越来越大，越外层的磁道浪费的空间越大



因此，对于磁盘容量的计算其实是相对来说比较简单的，由小到大来计算的话，就是

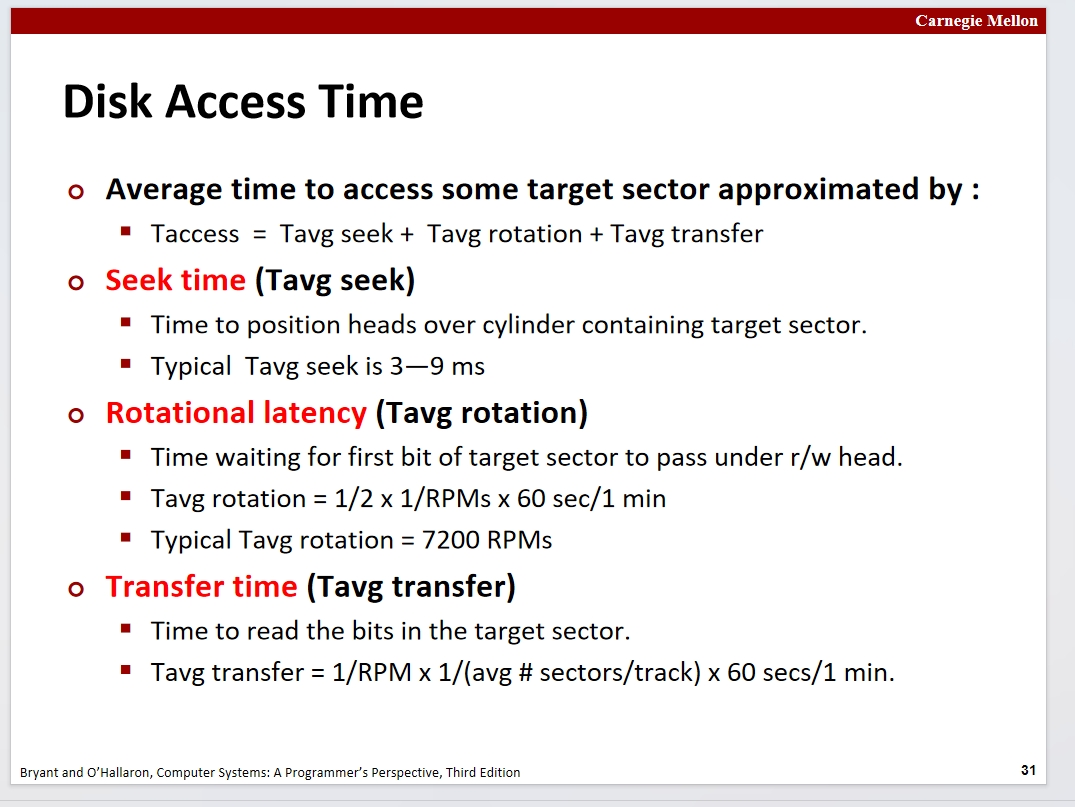
每个扇区的字节数\*一个磁道上的平均扇区数\*一个磁盘面上的磁道数\*一个盘片的盘面数\*一个磁盘上的盘片数



接下来分析下第三个性质 **平均寻址时间**

对于一次读取数据的过程，就比如这里，我们需要先读取一个内层磁道蓝色区域的数据，再读取外层磁道红色区域的数据，首先，假设一开始时磁头就位于这个蓝色区域所在的磁道了，这时由于磁盘的旋转，磁头会相对的经过我们要读取的蓝色区域的数据，再经过一些操作将这些数据传给总线。接着要读取外层磁道上的数据，此时控制器会先将磁头移动到该红色区域所对应的磁道上，一般都是伸缩臂，磁头一般不会发生水平的移动，接着等待磁盘的旋转使得磁盘上的红色区域的信息被磁头所读取

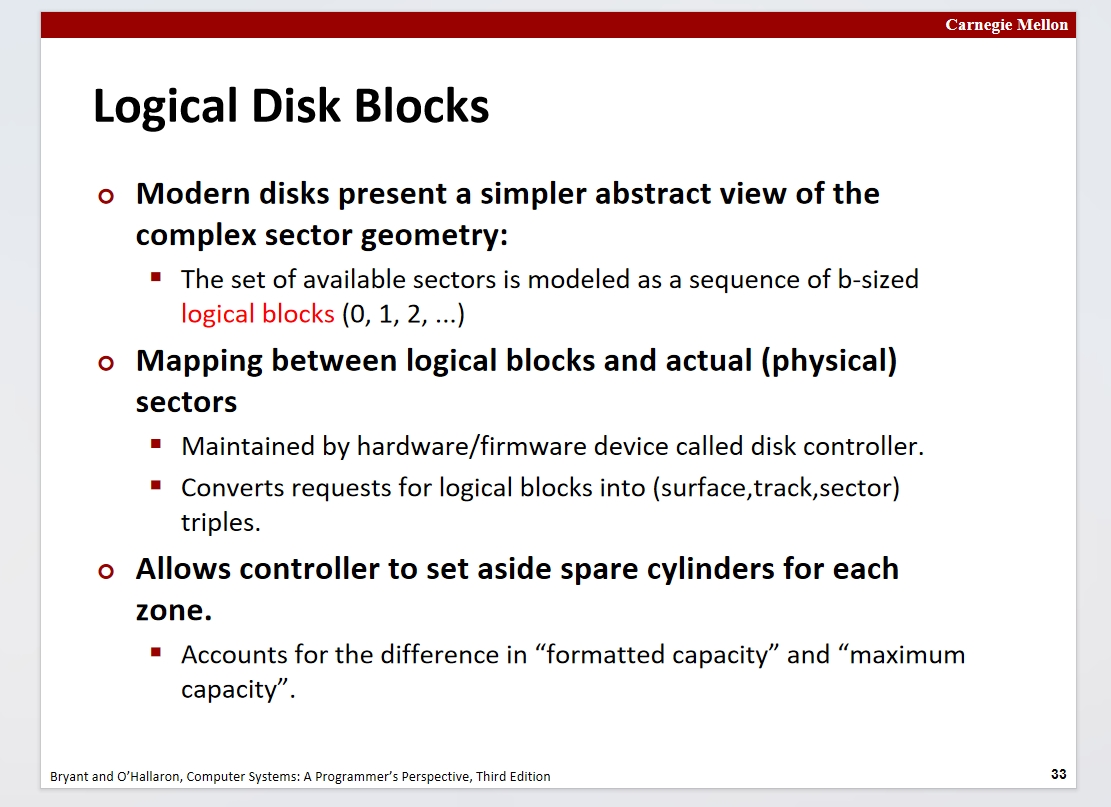
至于怎么确定的读取数据的位置，我不到啊，不过好像是通过一系列的映射和控制来实现的



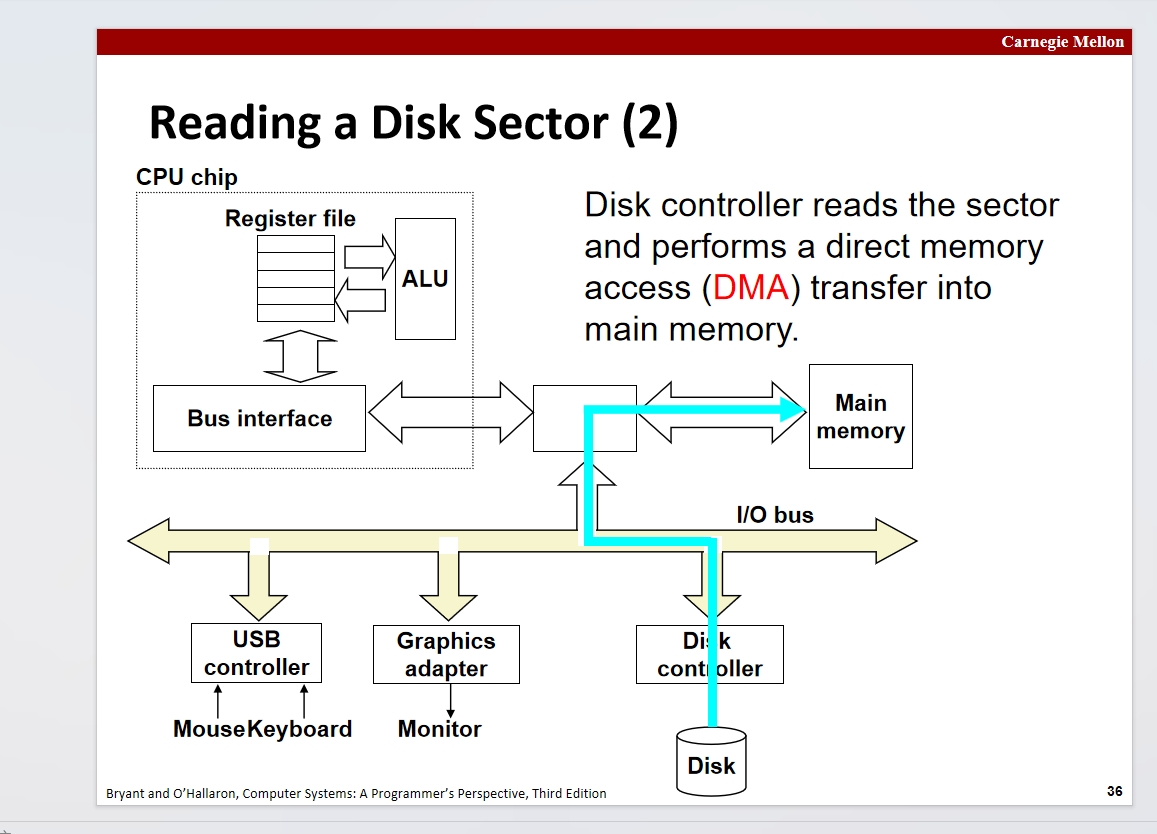
由于这个寻找过程中涉及到了物理层面的移动，所以这个过程是相当耗时的，就比如磁头的移动，一般都得花费3-5ms，这是机器限制

接着还有每一次在磁道上寻找片区的耗费时间(**旋转延迟**)，这个也涉及到了物理层面的移动，所以这也是耗时的,这个延迟一般在几ms到10+ms间，这在一定程度上取决于磁盘的转速

还有一个因素就是数据的传输时间，读取后需要将数据通过一系列方式读出，这也是耗时的点，这个的耗时一般相对较少

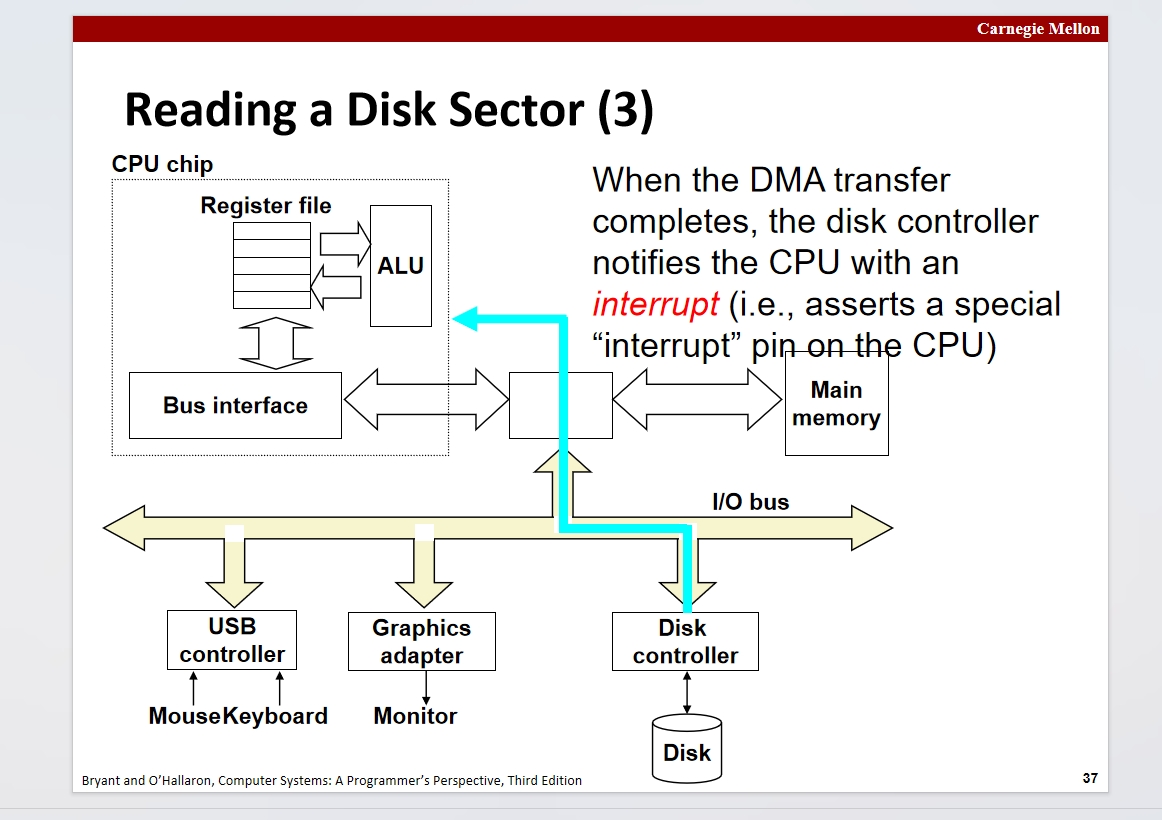


在现代计算机中，对于磁盘的处理，一般是将磁盘分为一些逻辑块来进行处理，将磁盘上的分区来划分为一个个块来进行映射，这些块是从0开始编号的，而一个块中可能包含有多个扇区，而这些处理磁盘控制器已经抽象好了，CPU只需要关心取用哪些块的数据，至于物理层面上的，可以通过映射来找到



接下来继续分析下底层的

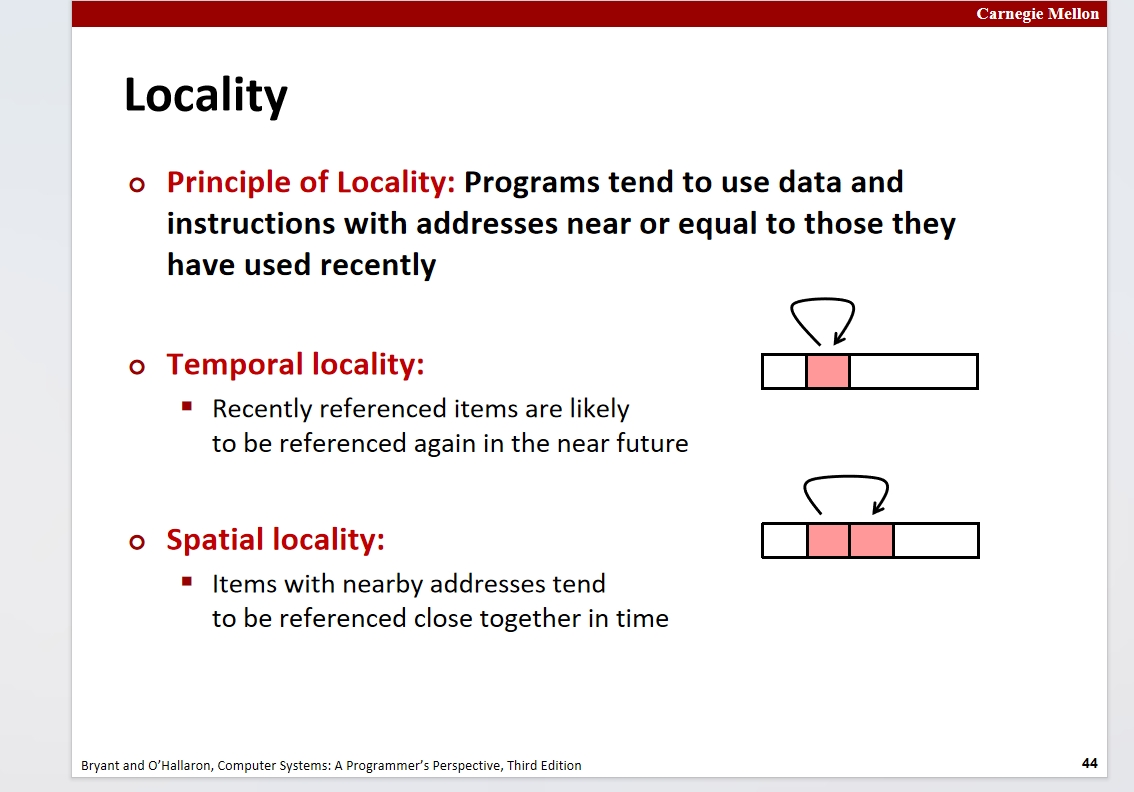
首先，CPU将三要素（指令，逻辑块号，内存地址）通过总线传给disk controller磁盘控制器，磁盘控制器通过映射找到对应数据在磁盘上的位置并读出，然后磁盘控制器会将这些数据通过I/O总线和一些其他部件将要读取的数据送到主存中去，注意，这个过程中没有经过CPU，正如上图中的蓝色路径，也就是说在这个过程中，CPU对这个操作是未知的，而要想让CPU确认操作是否已经完成



而当上面那个往主存里写入信息的操作结束后，磁盘控制器会通过一种**中断**的机制来通知CPU，这个机制应用了CPU的一个引脚，当写入完成后，控制器会修改CPU一个引脚的值，将其从0修改为1，以此来实现通知CPU写入操作已经完成，该扇区已经被复制，接着CPU可以通过这个中断引脚跳到之前的程序中继续执行

这种**中断**机制的好处在于

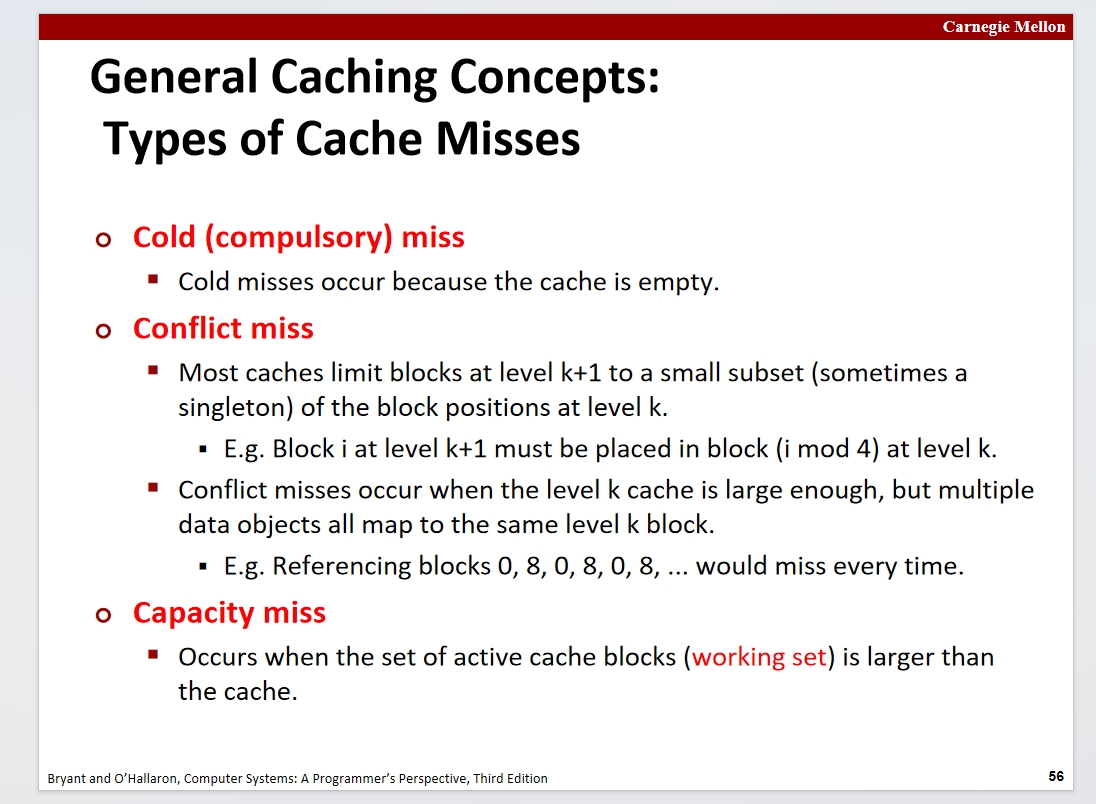
对于计算机来说，从磁盘中读取数据太慢了，在这段时间内，CPU甚至于能执行上百万条程序，如果让CPU停下来等待数据读取完毕，这会是一个严重的性能损失，因此，这里在要读取数据时直接将读取数据所需要的三要素直接全部打包发送给磁盘控制器，接下来，在接收到操作完成之前，CPU不会再关注这个操作，其会转而去执行其他的程序，使得充分利用CPU的算力资源，接下来当中断引脚状态发生改变时，CPU停下当前执行的程序并转而去执行这个引脚所对应的程序块，这姑且可以看做是对中断机制的一种理解，这样，就一定程度上减小了由于读取数据的低效性而带来的性能损失



接下来讨论下**局部性**(Locality)

分为空间局部性和时间局部性，简单来说:  
 时间局部性指的是最近访问的储存器位置很可能在不久的将来成为再次被引用的属性

空间局部性指的是你访问了一个位置，那么在不久的将来你很可能继续访问这个位置之后的数据



接下来分析下缓存不命中的情况：**冷不命中 容量不命中 冲突未命中**

冷不命中很好理解，这种情况是指缓存中没有数据导致的缓存不命中，而这种情况显然是无法避免的，当程序逐渐运行后，各级缓存会被逐渐填满，此时冷不命中会逐渐消失，这是就进入到了另一种情况，容量不命中，这种不命中其实也是不可避免的，毕竟缓存的容量是有限的，你不能奢望你每次访问的数据都已经在缓存中了

以此我们可以来引入一个**工作集**的概念，**工作集**是指在一定时间内一个程序需要访问的所有数据集合，包含了程序使用的变量，指令等。就比如一个计算元素数量为19的数组的总和的for循环，我们可以将这个for循环视为一个工作集，而一般，我们的缓存区一般不能够容纳19个数据这么大的数据量，也就是说，对于一定范围内的数组元素，我们可以通过缓存来访问，但由于缓存大小有限，所以对于整个工作集的工作过程中，缓存的容量不命中是不可避免的

因此，但工作集的大小大于缓存时，缓存容量不命中是经常且不可避免发生的

还有一种miss叫**冲突未命中，**这种未命中是取决于缓存底层的实现的，由于一些原因，缓存的实现应该相对简单以保证缓存的速度，比如说，计组中说缓存一般可以通过映射来实现与主存块间的映射，而需要注意的是，缓存的容量是远远小于主存的，缓存与缓存间的映射先不考虑，而由于缓存容量远远小于主存，而一般映射都是一个取模算法来实现的，这就意味着多个主存块是很可能映射到一个缓存块中的，这在计组中也有说明，而这种覆盖可能就导致了我们将会使用的数据被覆盖，导致冲突未命中

之后还是得看计组课再开一个仓啊