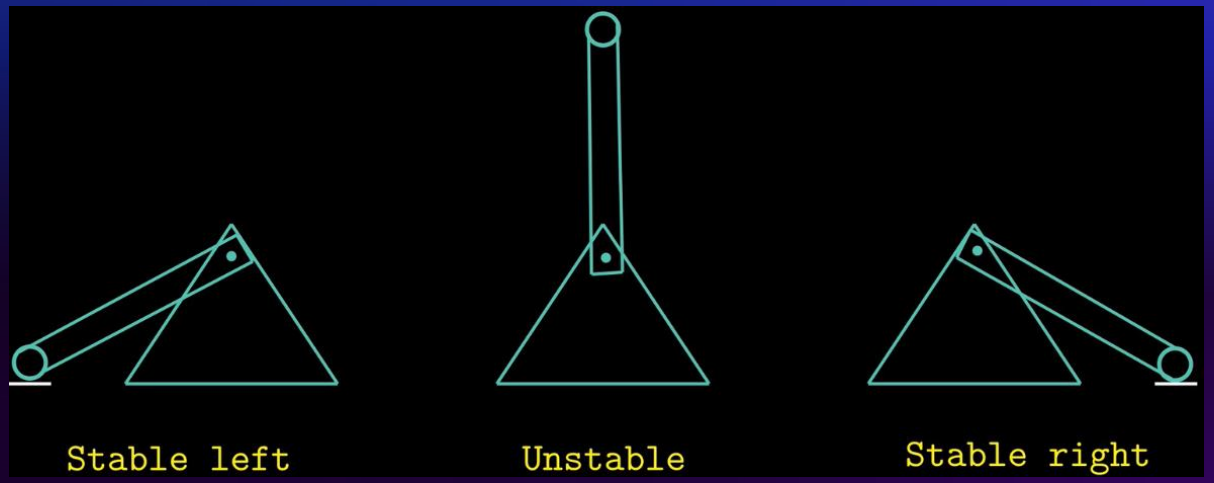
存儲器層次結構

6.1-6.3

目录

Random Access Memory（随机访问储存器）

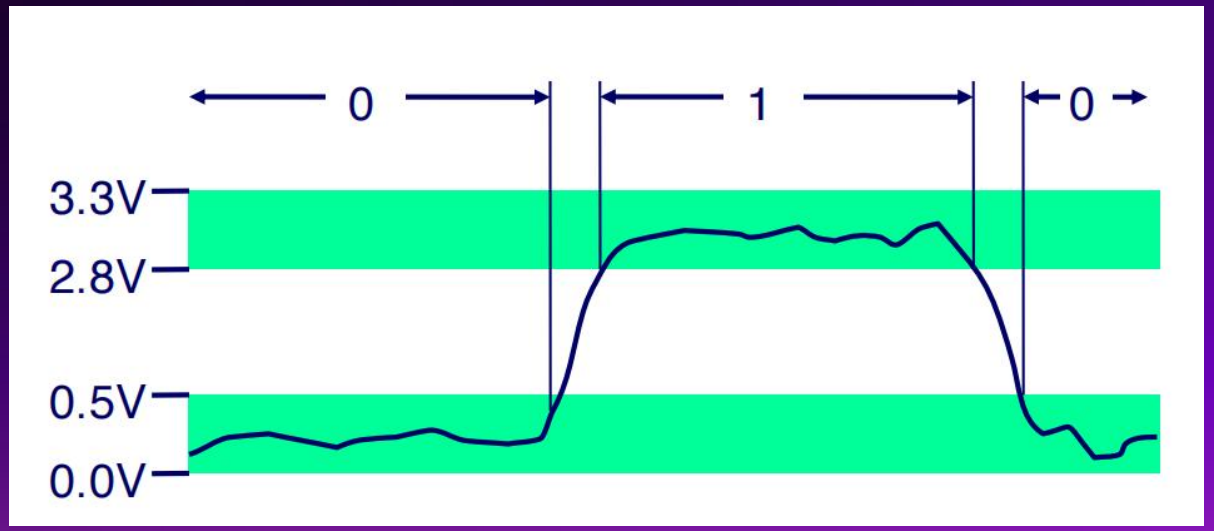
- Static RAM（静态RAM，SRAM）
- Dynamic RAM（动态RAM，DRAM）
  - SRAM将每个位存储在一个双稳态的存储器单元内。每个单元需要6个晶体管。
  - 何为双稳态？
  ↓这个是动力学上的定义



基本定义 播报 编辑

在势能方面，一个双稳态系统有两个局部势能极小值，它们之间有一个局部极大值。一个双稳态的机械设备的例子是灯的开关，开关要么“开”要么“关”，但不会在二者之间停留。

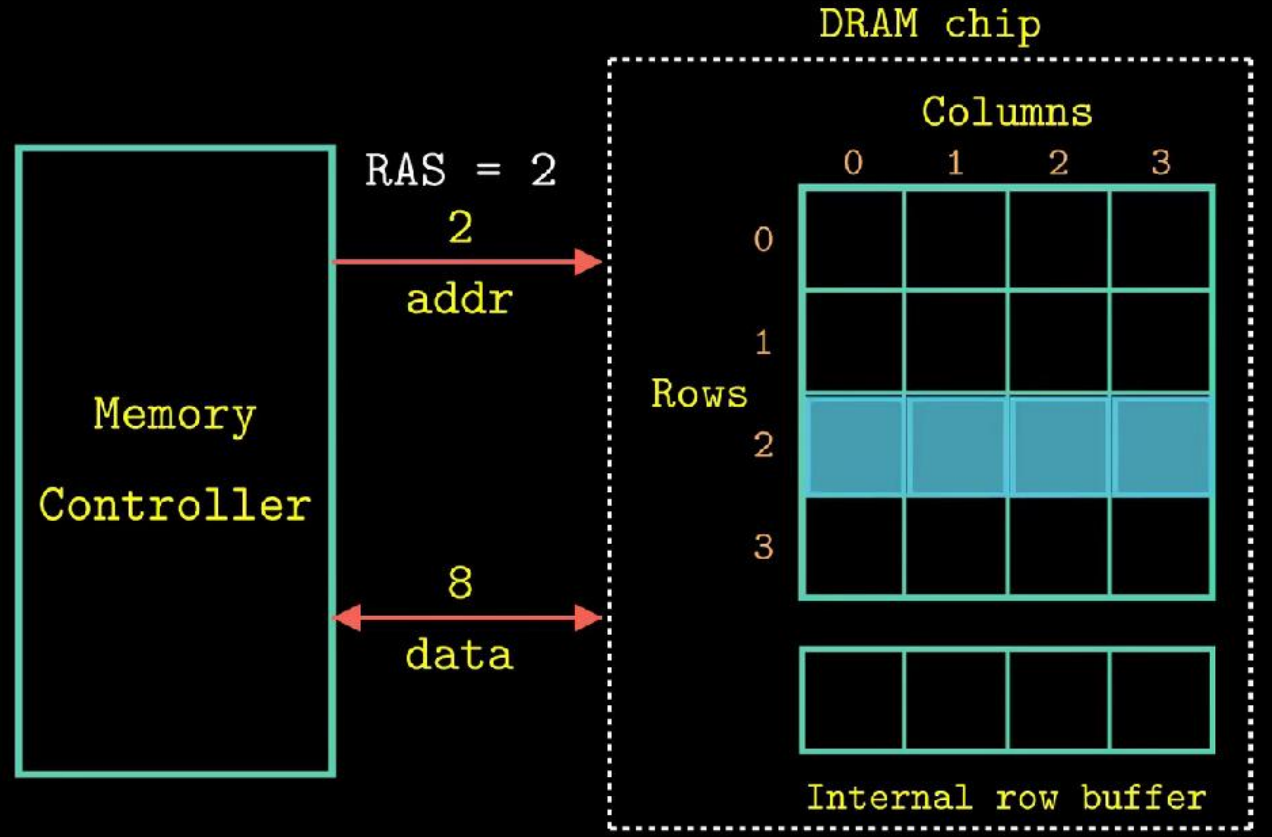
- 可以套用在我们这里的理解上

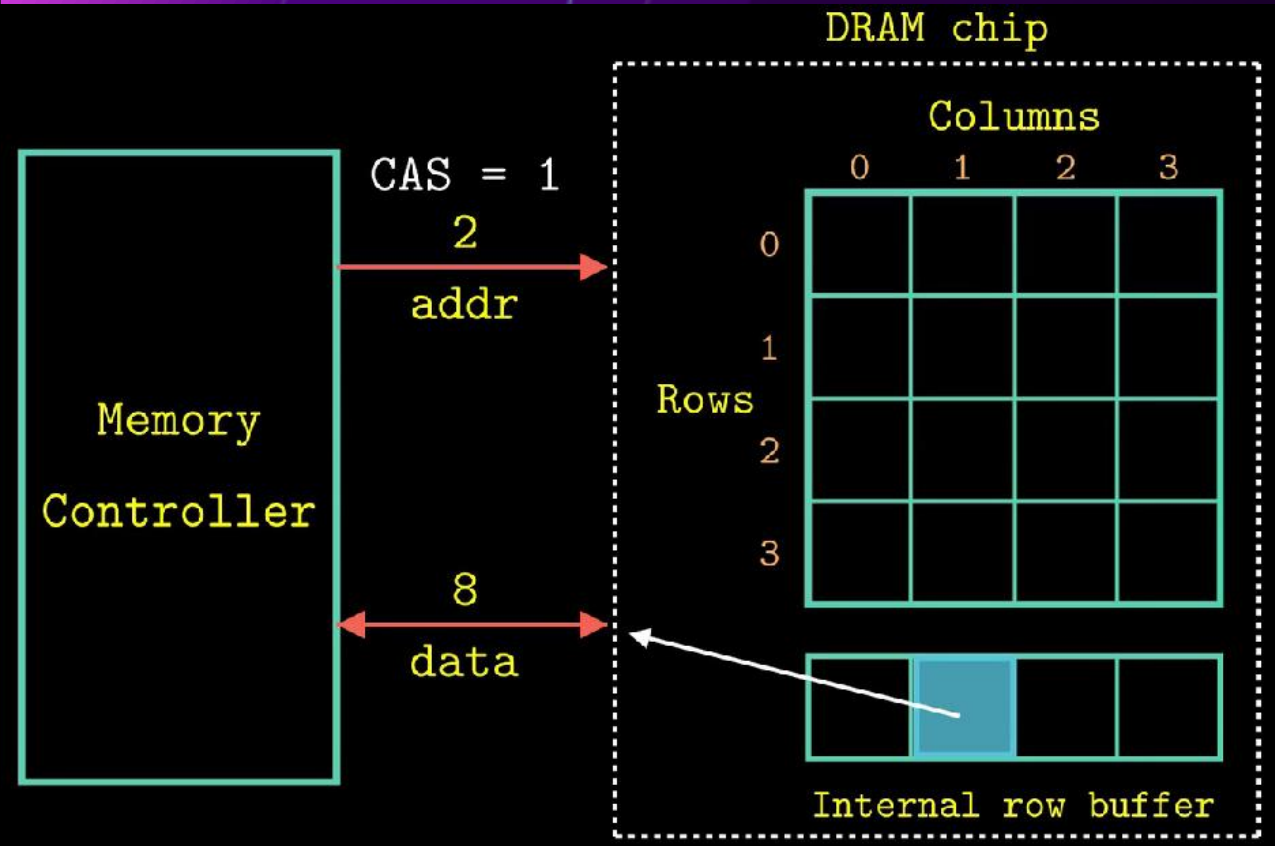


DRAM 和 SRAM 的不同

- DRAM的每个位储存为对一个电容的充电（也就是每个位使用1个晶体管）
- 不像SRAM具有双稳态特质，DRAM对干扰非常敏感
- 可能会漏电
- 访问时间更长，大概是SRAM的10倍
- 但是也便宜得多，单位储存量的价格只有SRAM的1/10
- 所以DRAM会用在内存，SRAM会用在高速缓存

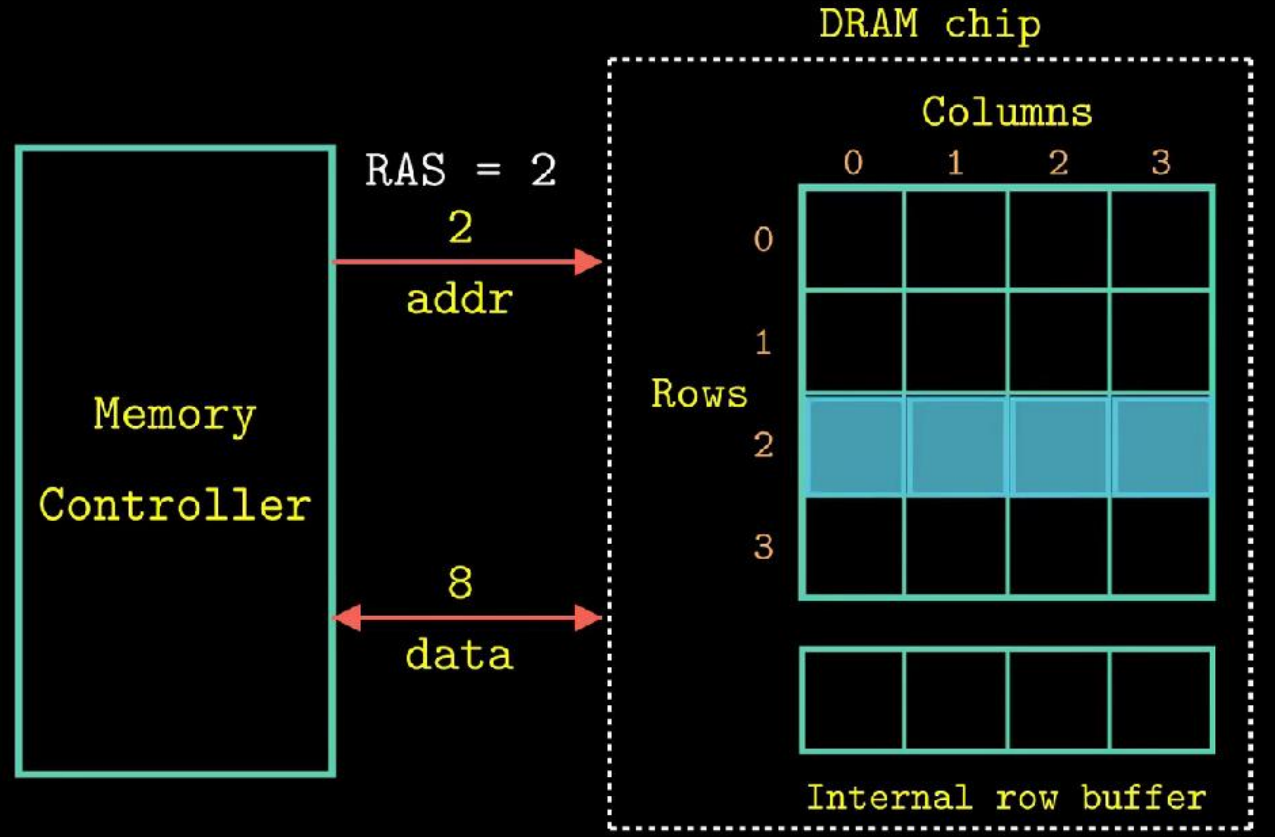
- 一个DRAM芯片被分为d个超单元
- 在这个例子里d=16，分为4行4列
- 读取数据需要内存控制器来操作
- 就好比数据是书，内存是图书馆，内存控制器是图书管理员
- DRAM芯片和内存控制器间有两种引脚，一种是地址引脚，一种是数据引脚

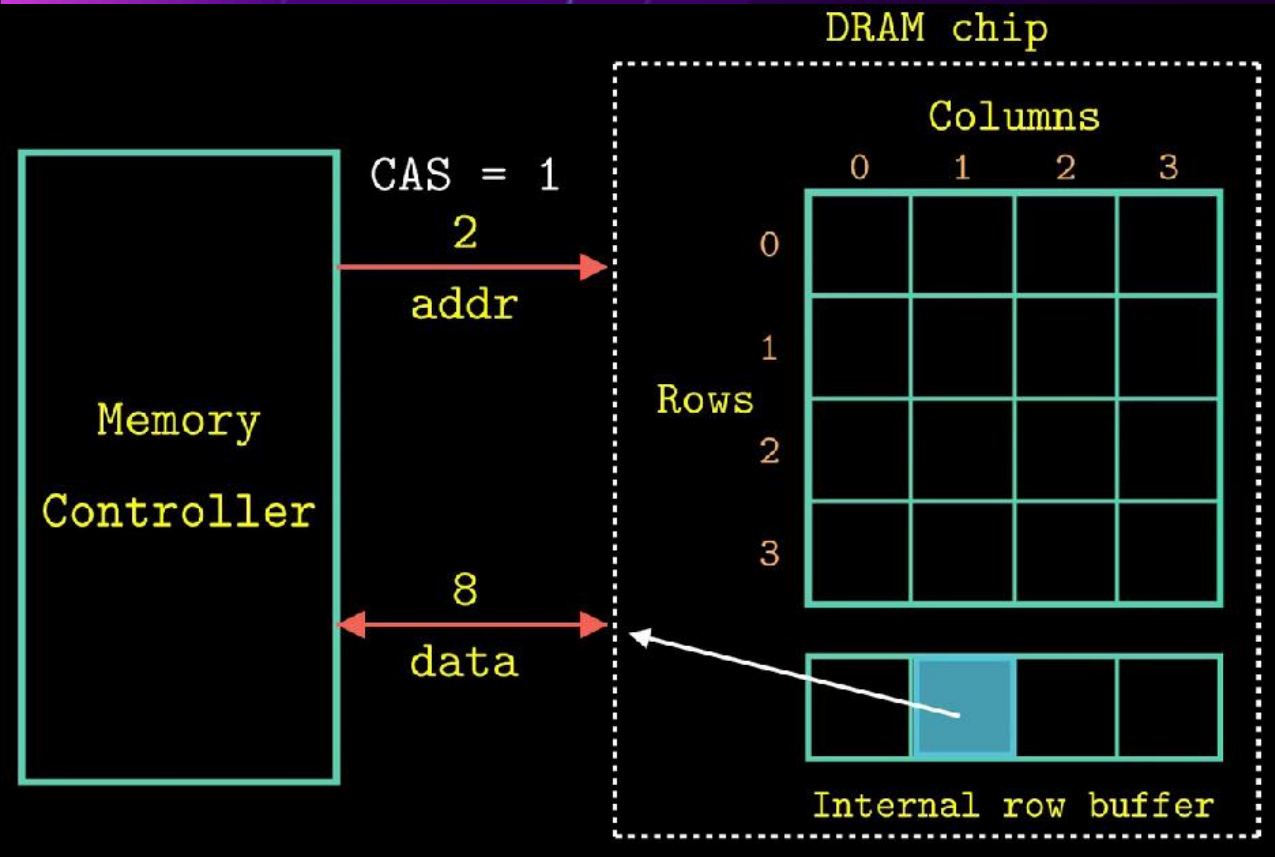




- 读取数据时，先通过地址引脚发送行数2
- DRAM芯片把整个第2行放在行缓冲区
- 再通过地址引脚发送列数1
- 从缓冲区读取第1行的超单元，通过数据引脚发送回内存控制器
- 一个超单元的容量一般是8位
- 为什么设计成二维阵列？

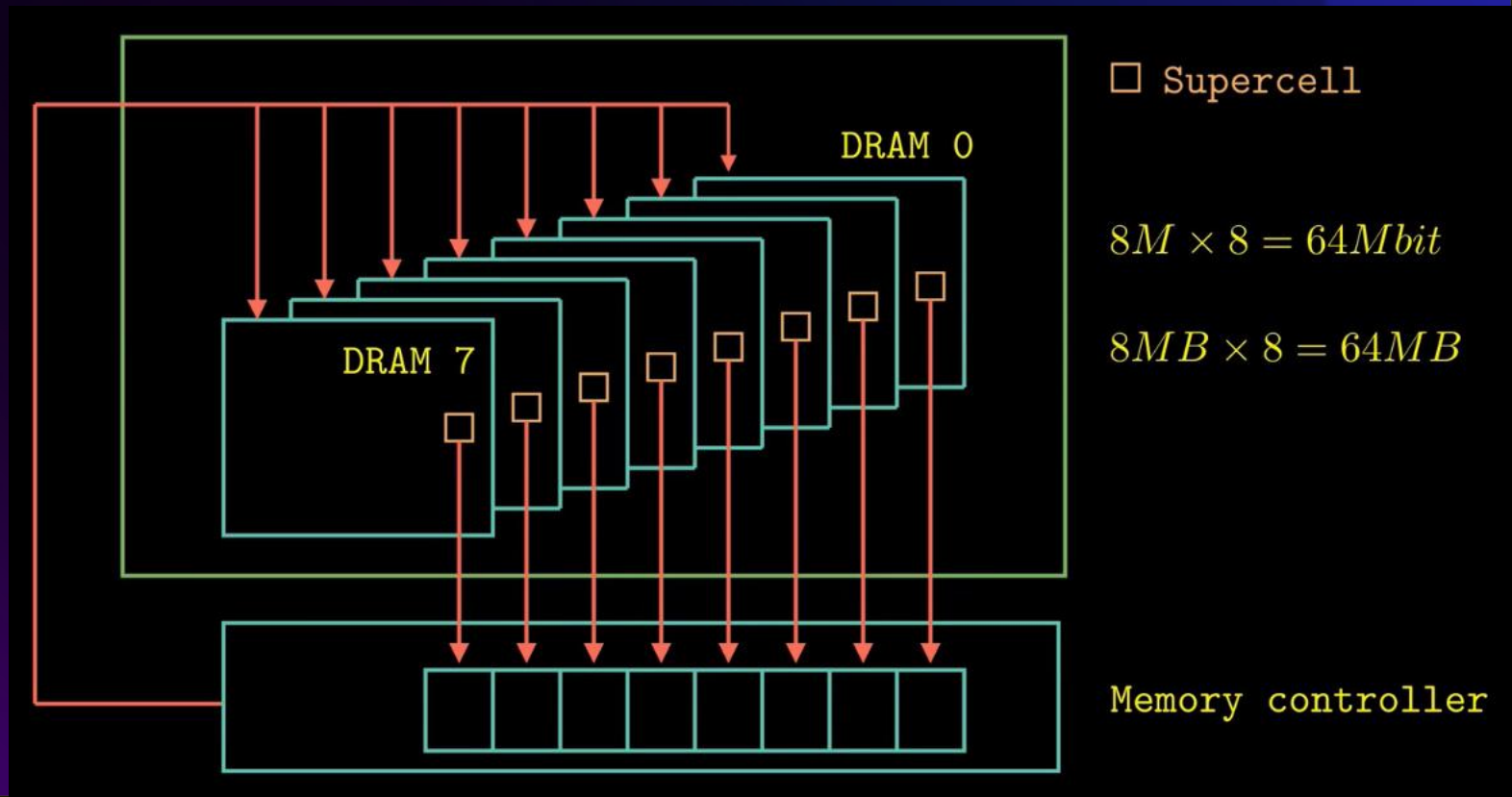
•如果设计成一维线性数组，以这里的16个超单元为例，由于1个引脚每次只能传输1个0或1作为数据，2^4=16，所以需要一次性使用四个引脚来定位超单元
•设计成4*4的阵列，由于2^2=4，可以只使用两个引脚，先定位行，再定位列，减少了引脚的数量，可以节约成本





•当然也有一些缺点，因为需要分两步发送数据，会增加访问时间
•但整体上弊大于利

- DRAM芯片一般会封装成内存模块，比如这里由DRAM0到DRAM7共8个DRAM芯片组成
- 如果要读取一个8位（64bit）的数据，我们会从8个超单元上读取，通常这8个超单元平均会分布在8块芯片上的相同位置
- 一般而言，DRAM0储存最低的8位，以此类推，DRAM7储存最高的8位
- 内存控制器会把地址转换为超单元地址（i，j），发送到内存模块，内存模块把这个地址发给DRAM芯片，每个芯片读取对应地址的数据，发送给内存模块，内存模块将其整合成64bit的数据，返回给内存控制器
- 现代的DRAM越来越多了



磁盘（DISK）

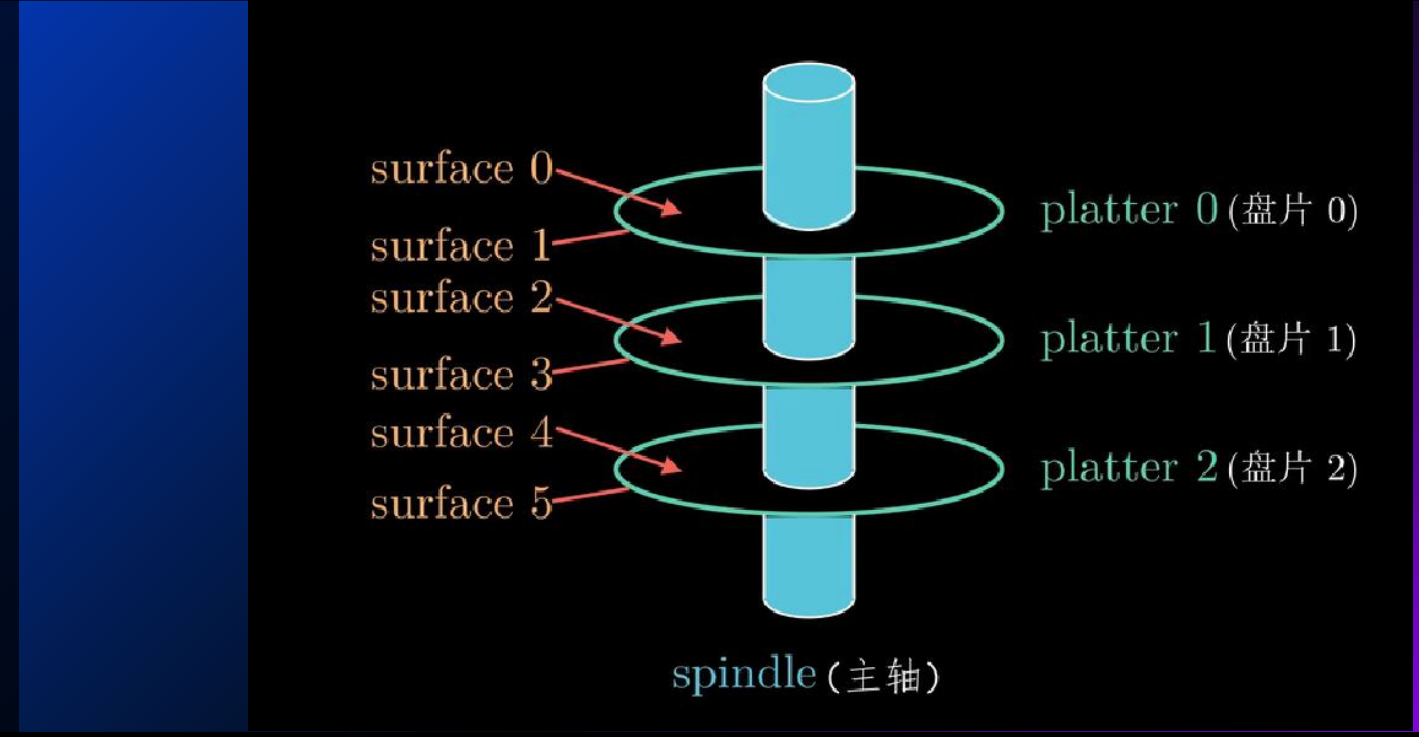
· 一般分为机械磁盘和固态磁盘，机械硬盘也称旋转磁盘

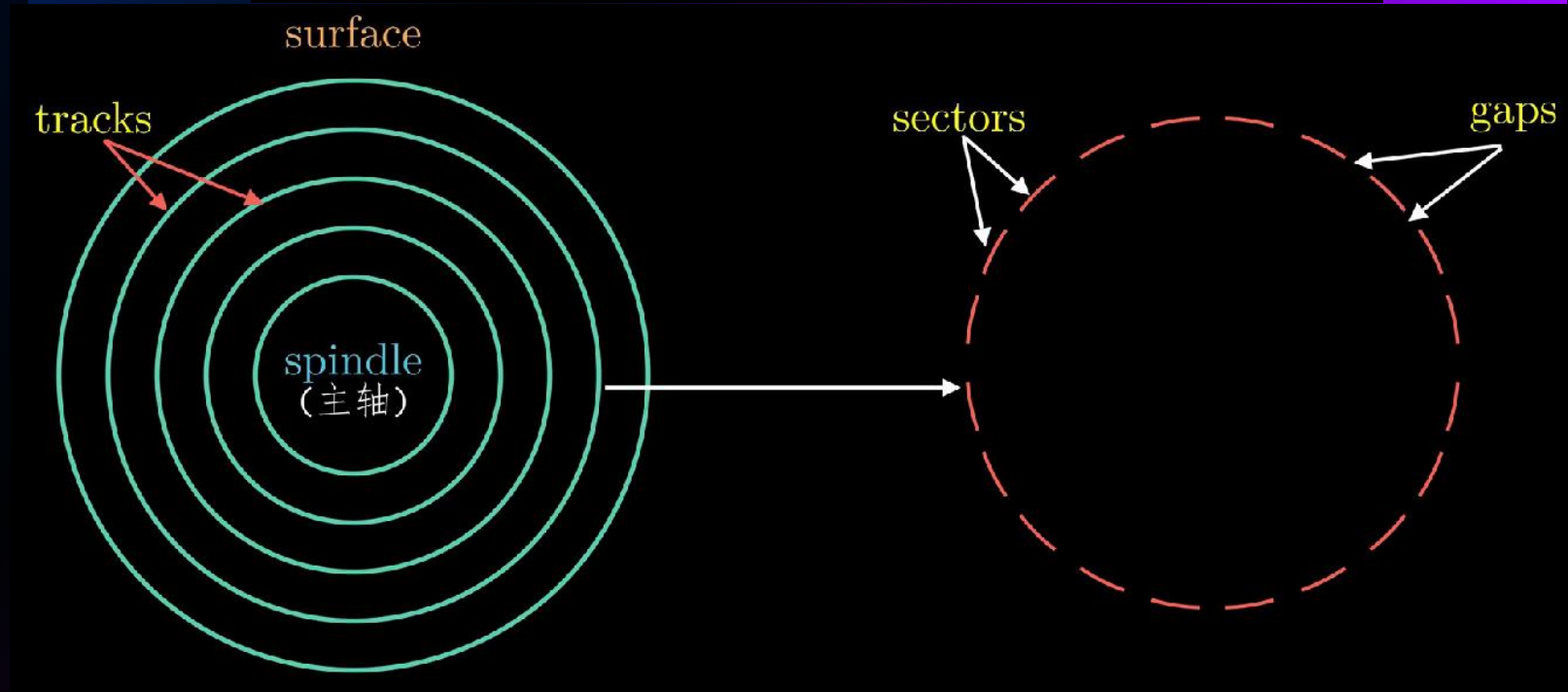
· 机械磁盘由一个主轴，多个盘片组成，每个盘片有两面

· 每个面上有多个同心圆磁道

· 每个磁道分为多个扇区

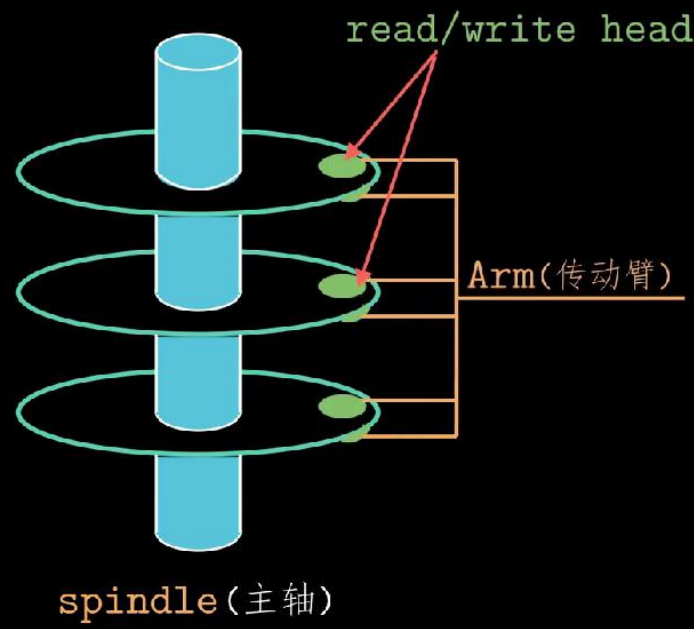
· 扇区中间有不储存信息的间隙





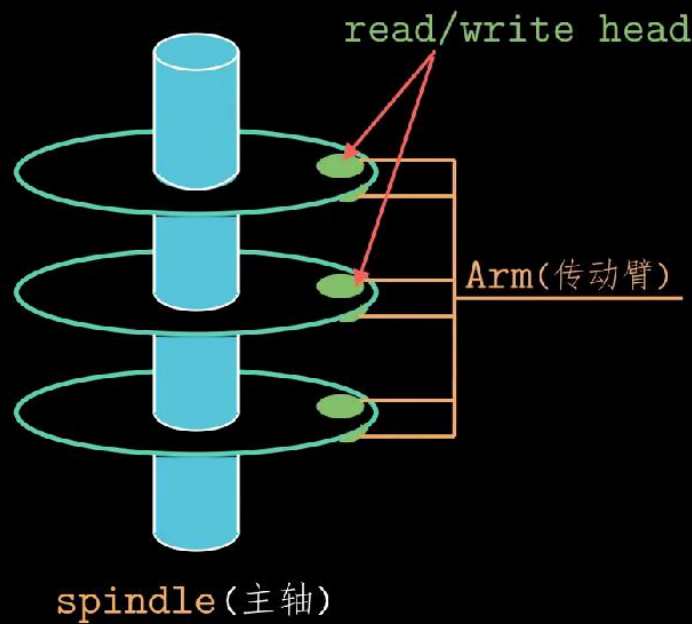
如何计算磁盘读取速度

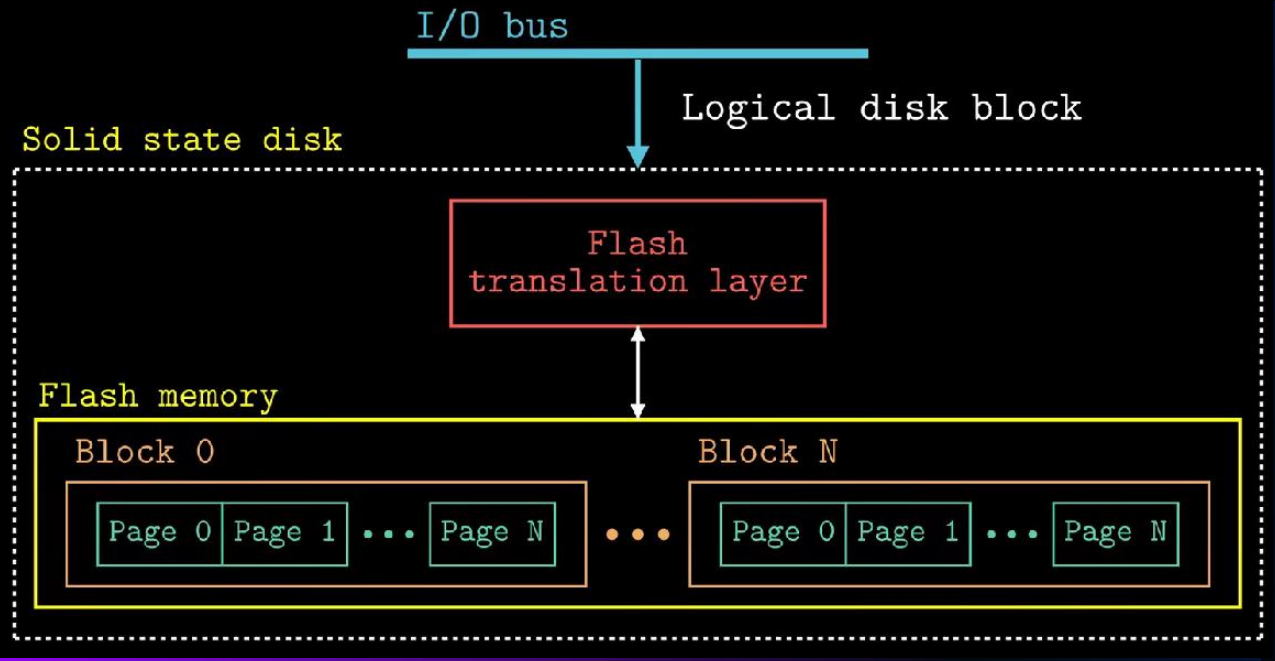
- 对扇区的访问时间主要由以下三个部分组成
- 寻道时间，即通过移动传动臂，把读写头定位到所需磁道上，这个时间通常是多次测试取平均值得到
- 旋转时间，即将读写头定位到所需扇区的开头。由于不知道需要的扇区与读写头的相对位置关系，最坏的情况是读写头和扇区刚刚错过，这时需要转一整圈才能到目的扇区。用时为1/RPM（Revolution Per Minute,每分钟转速，一般要转换为s或ms为单位）。平均情况下需要旋转半圈，用时是上面的一半。
- 传送时间，即读取一个（或多个）扇区需要的时间，一般而言我们忽略扫过扇区间隙需要的时间，用时是1*需要读取的扇区数/(RPM*每条磁道上的总扇区数)
- ·总时间是以上三个部分相加



如何计算磁盘读取速度

- 如果只读取一个扇区，由于一条磁道上的扇区数以百计，传送时间相较于旋转时间很短，主要时间由前两个部分组成
- 如果读取多个扇区，有多种情况。最好的情况是所有需要读取的扇区是连续的，如此只需一次寻道和一次旋转。
- 最差的情况下是所有的扇区都是随机分布的，如此每读取一个扇区就需要一次寻道和一次旋转。
- 在读取多个扇区的情况下，“最坏情况”一般并不考虑谈论旋转时间时的“最坏情况”，我们默认旋转时间用平均值。这里说的“最坏”是指储存的扇区在磁盘上的分布情况。
- · 所以如果可以的话，我们希望每次需要读取的数据，会储存在比较相邻的扇区上，这需要用到“局部性”。





固态硬盘（SSD）

· 固态硬盘使用闪存芯片来取代传动臂和盘片的工作方式

· 闪存由多个块组成，每个块又由多个页组成，数据以页为单位读写

· SSD有一个特点，如果想要写某一页，需要这一页所属的整个块都被擦除（一般是指全部设置为1），不过擦除后这个块内的所有页可以直接再写一次

· 这决定了读SSD比写SSD要快

· 由于擦和写SSD会造成磨损，我们一般会希望磨损能比较平均地分布在各个块上，这样可以让整体的寿命更长

· 闪存翻译层中有一个平均磨损逻辑来解决这个问题

· 此外，我们也会希望每次写的页在同一个或者较少的块内，以减少擦的次数

· 这又与局部性有关

局部性

- 一个编写良好的计算机程序通常具有良好的局部性，它包括两个部分，时间局部性和空间局部性。
- 时间局部性，意思是我们倾向于引用最近引用过的数据项。换句话说，如果我们需要在运行一个程序的过程中多次引用某一个数据，我们最好让这几次引用在时间上临近。
- 空间局部性，意思是我们倾向于引用在存储上临近于“其它最近引用过的数据项”的数据项，空间局部性的作用，在磁盘读取的时候已经解释过。

局部性

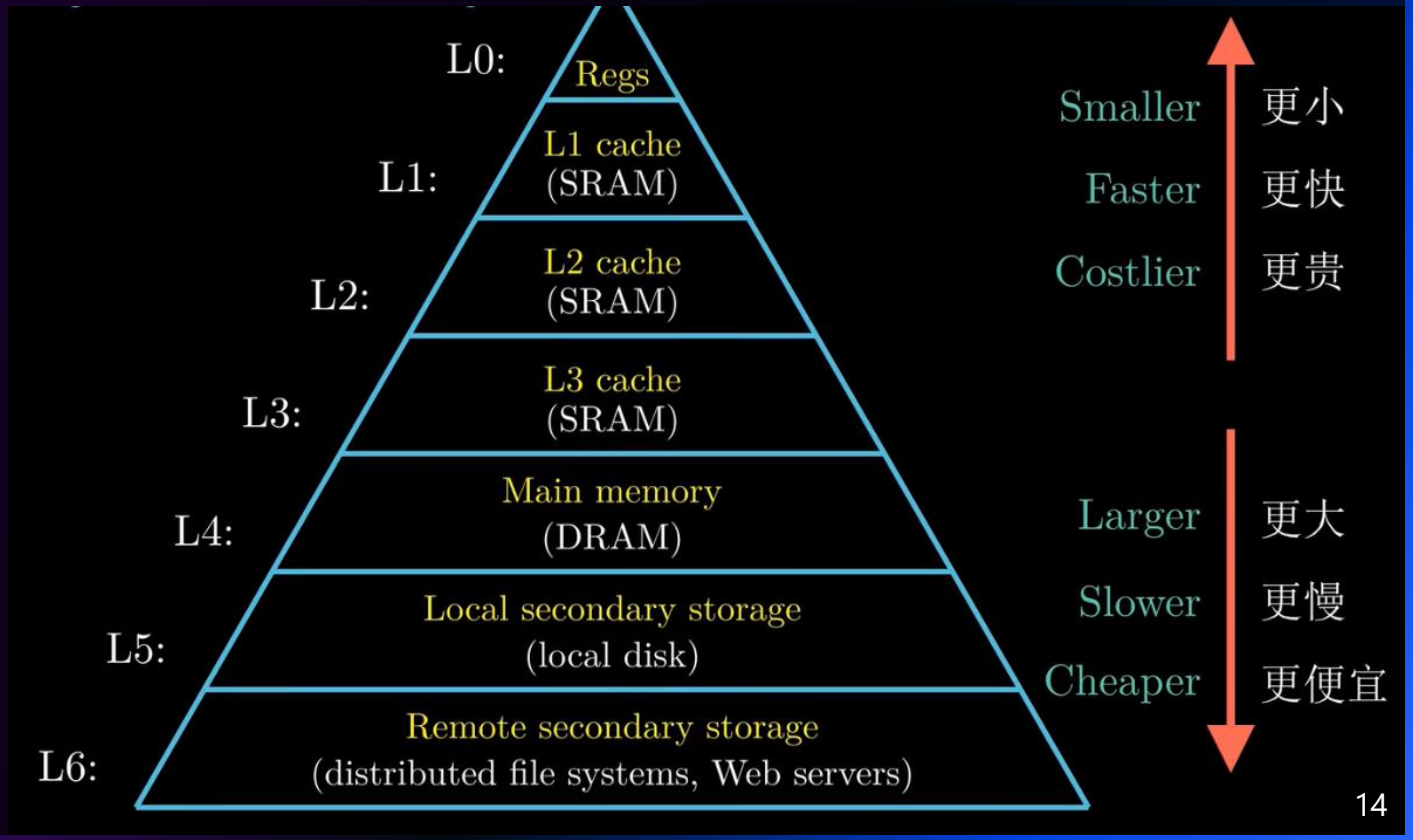
- 举一个例子，比如右边的这两个程序，它们唯一的区别是读取数组元素时，是一行一行地读还是一列一列地读
- 但数组在储存时，默认是按照行顺序来存放的。也就是说，同一行的数据会存储在相邻的空间里。
- 所以上面这个程序一行一行地读，就比下面这个程序一列一列地读要快。
- 这是空间局部性的例子，接下来讲述为何需要时间局部性。这首先要理解存储器层次结构。

```
1   int sumarrayrows(int a[M][N])
2   {
3       int i, j, sum = 0;
4
5       for (i = 0; i < M; i++)
6           for (j = 0; j < N; j++)
7               sum += a[i][j];
8       return sum;
9   }
```

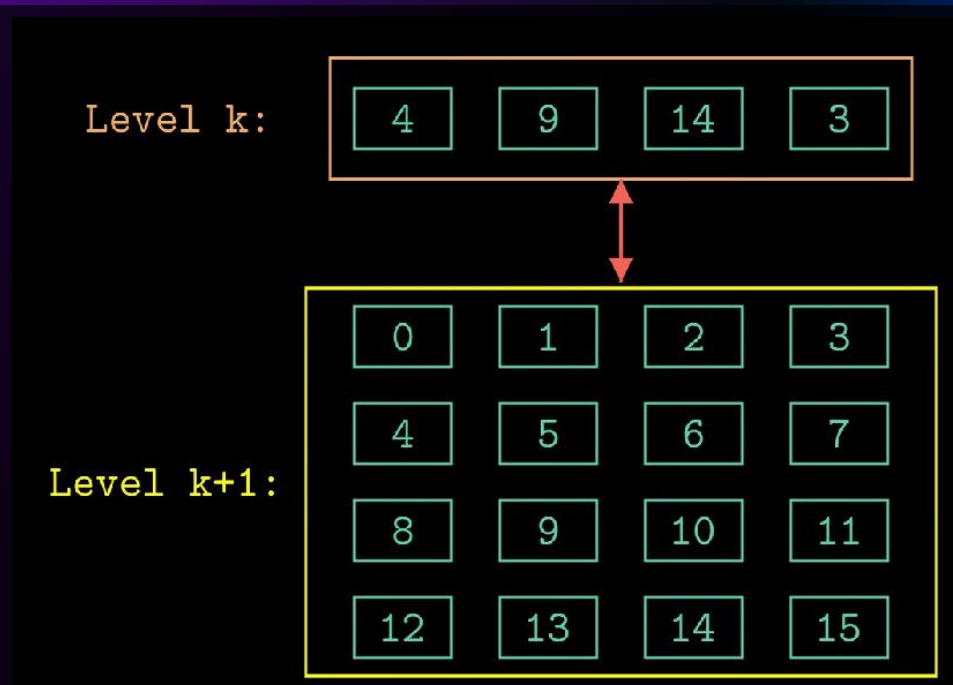
```
1   int sumarraycols(int a[M][N])
2   {
3       int i, j, sum = 0;
4
5       for (j = 0; j < N; j++)
6           for (i = 0; i < M; i++)
7               sum += a[i][j];
8       return sum;
9   }
```

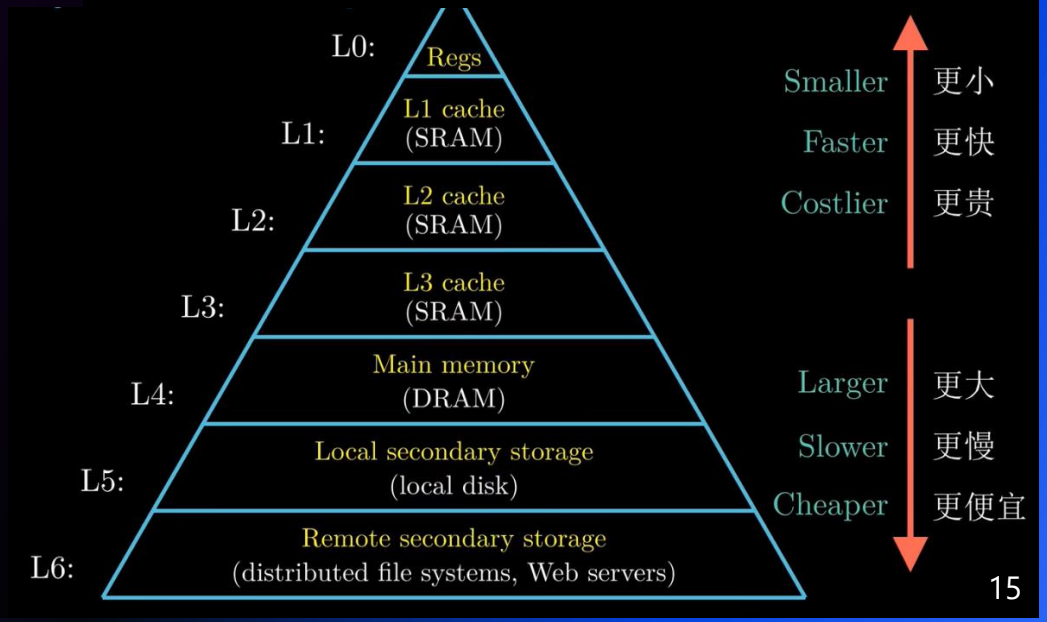
- 在这张图中，不同的存储器构成了一个类似于金字塔的结构，越靠上的存储器容量越小，也更贵；越靠下的存储器容量越大，并且单位储量更便宜。
- 但是越靠上的存储器，读取也越快。比如寄存器的读取只需要1个时钟周期；L1至L3的读取需要几个时钟周期；基于DRAM的主存读取需要几十几百个时钟周期。

- 时钟周期是什么？
- 时钟周期定义为时钟频率的倒数，是计算机中最基本的、最小的时间单位。在一个时钟周期内，CPU仅完成一个最基本的动作，比如一次二进制加法。
- （你可以通过按CTRL+SHIFT+ESC打开任务管理器，找到“性能”这一栏，里面会有以GHz或者MHz为单位的“速度”这一栏，其倒数就是一个时钟周期的时间）
- 我们的电脑时钟频率一般在1GHz上下，对应的时钟周期大概是1ns，有个概念。



- 以右图为例，我们把每一层存储器中的数据都分为块，如果相邻层的块大小一致，那么下一层的块数量就更多。数据以块为单位传输。
- 每一层中会存有下一层中的一部分数据，称为缓存。
- 当我们需要k+1层的某个数据d时，会先看k层中有没有d的缓存，如果有，从k层中读取d就行。这称为缓存命中。
- 如果没有，我们就需要从k+1层中取出含d的块，来替换k层中原来缓存的一个块。这称为缓存不命中。
- 如果我们需要多次用到一个数据d，当我们后一次读取数据d时，我们会希望它尽可能已经在k层中缓存好。所以我们会希望它离上一次引用时间短一些（因为间隔时间越长，原先k层中含有d的块就越可能被替换掉）
- 这解释了为什么需要时间局部性





THANK YOU

2024.10.30