第六章 存储器层次结构

第一部分

存储器层级结构与局部性

教师：史先俊
计算机科学与技术学院
哈尔滨工业大学

- 存储技术及其趋势
- 局部性
- 存储器层次结构中的高速缓存

回顾：程序及指令的执行过程

在内存存放的指令实际上是机器代码（0/1序列）

```
08048394 <add>:
1  8048394:  55          push   %ebp
2  8048395:  89  e5       mov    %esp, %ebp
3  8048397:  8b  45 0c    mov    0xc(%ebp), %eax
4  804839a:  03  45 08    add    0x8(%ebp), %eax
5  804839d:  5d          pop    %ebp
6  804839e:  c3          ret
```

栈是主存中的一个区域！

- 对于add函数的执行，以下情况下都需要访存：
  - ✓每条指令都需从主存单元取到CPU执行 取指
  - ✓PUSH指令需把寄存器内容压入栈中 存数 POP指令则相反 取数
  - ✓第3条mov指令需要从主存中取数后送到寄存器 取数
  - ✓第4条add指令需要从主存取操作数到ALU中进行运算 取数
  - ✓ret指令需要从栈中取出返回地址，以能正确回到调用程序执行 取数

访存是指令执行过程中一个非常重要的环节！ 取指、取数、存数

基本术语

记忆单元 （存储基元 / 存储元 / 位元） （Cell）

具有两种稳态的能够表示二进制数码0和1的物理器件

存储单元 / 编址单位（Addressing Unit）

具有相同地址的位构成一个存储单元，也称为一个编址单位

存储体/ 存储矩阵 / 存储阵列（Bank）

所有存储单元构成一个存储阵列

编址方式（Addressing Mode）

字节编址、按字编址

存储器地址寄存器（Memory Address Register - MAR）

用于存放主存单元地址的寄存器

存储器数据寄存器（ Memory Data Register-MDR (或MBR) ）

用于存放主存单元中的数据的寄存器

存储器分类

依据不同的特性有多种分类方法

（1）按工作性质/存取方式分类

随机存取存储器 Random Access Memory (RAM)

每个单元读写时间一样，且与各单元所在位置无关。如：内存。

（注：原意主要强调地址译码时间相同。现在的DRAM芯片采用行缓存，因而可能因为位置不同而使访问时间有所差别。）

顺序存取存储器 Sequential Access Memory (SAM)

数据按顺序从存储载体的始端读出或写入，因而存取时间的长短与信息所在位置有关。例如：磁带。

直接存取存储器 Direct Access Memory(DAM)

直接定位到读写数据块，在读写数据块时按顺序进行。如磁盘。

相联存储器 Associate Memory（AM）

Content Addressed Memory (CAM)

按内容检索到存储位置进行读写。例如：快表。

存储器分类

(2) 按存储介质分类

半导体存储器：双极型，静态MOS型，动态MOS型
磁表面存储器：磁盘（Disk）、磁带 （Tape）
光存储器：CD，CD-ROM，DVD

(3) 按信息的可更改性分类

读写存储器（Read / Write Memory)：可读可写
只读存储器（Read Only Memory)：只能读不能写

(4) 按断电后信息的可保存性分类

非易失（不挥发）性存储器(Nonvolatile Memory)
信息可一直保留， 不需电源维持。
(如：ROM、磁表面存储器、光存储器等)
易失（挥发）性存储器(Volatile Memory)
电源关闭时信息自动丢失。 (如：RAM、Cache等)

存储器分类

(5) 按功能/容量/速度/所在位置分类

寄存器(Register)

封装在CPU内，用于存放当前正在执行的指令和使用的数据

用触发器实现，速度快，容量小（几~几十个）

高速缓存(Cache)

位于CPU内部或附近，用来存放当前要执行的局部程序段和数据

用SRAM实现，速度可与CPU匹配，容量小（几MB）

内存储器MM（主存储器Main (Primary) Memory）

位于CPU之外，用来存放已被启动的程序及所用的数据

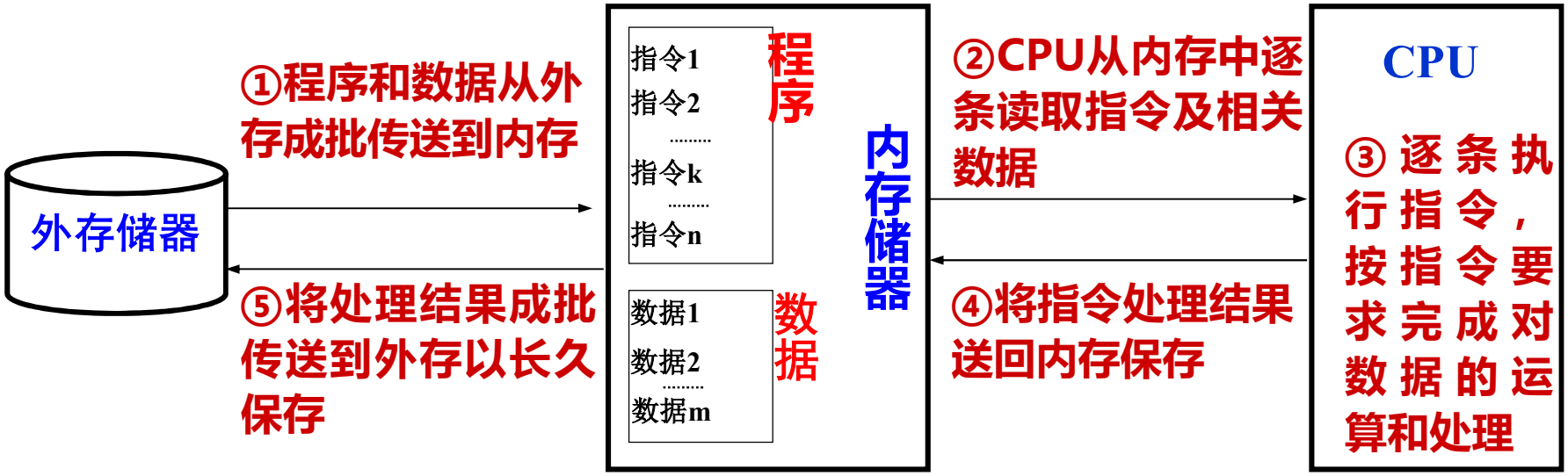
用DRAM实现，速度较快，容量较大（几GB）

外存储器AM (辅助存储器Auxiliary / Secondary Storage)

位于主机之外，用来存放暂不运行的程序、数据或存档文件

用磁表面或光存储器实现，容量大而速度慢

内存与外存的关系及比较



- ✓ **外存储器（简称外存或辅存）**
  - **存取速度慢**
  - **成本低、容量很大**
  - **不与CPU直接连接，先传送到内存，然后才能被CPU使用。**
  - **属于非易失性存储器，用于长久存放系统中几乎所有的信息**

- ✓ **内存储器（简称内存或主存）**
  - 存取速度快
  - 成本高、容量相对较小
  - 直接与CPU连接，CPU对内存中可直接进行读、写操作
  - 属于易失性存储器(volatile)，用于临时存放正在运行的程序和数据

主存的结构

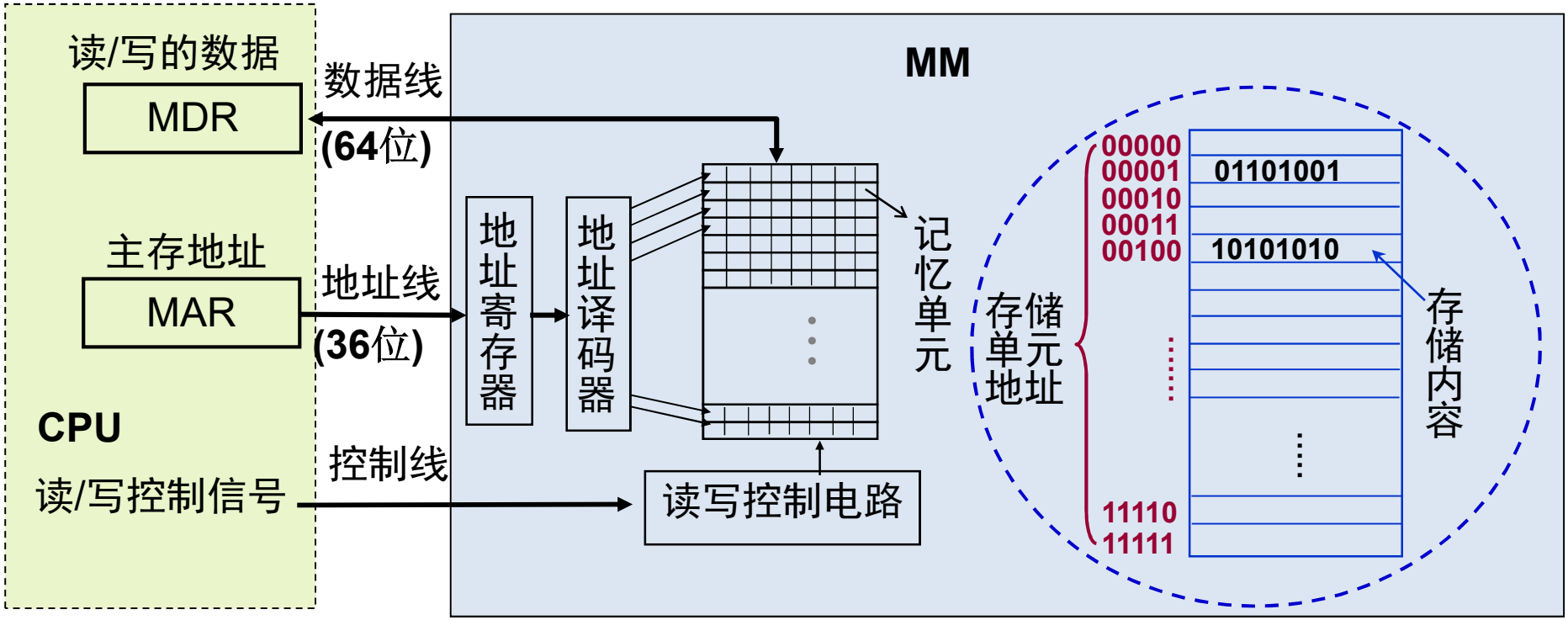
问题：主存中存放的是什么信息？CPU何时会访问主存?

指令及其数据！CPU执行指令时需要取指令、取数据、存数据！

问题：地址译码器的输入是什么？输出是什么？可寻址范围多少?

输入是地址，输出是地址驱动信号（只有一根地址驱动线被选中）。

可寻址范围为0~2^{36}-1，即主存地址空间为64GB（按字节编址时）。



主存地址空间大小不等于主存容量（实际安装的主存大小）！

若是字节编址，则每次最多可读/写8个单元，给出的是首(最小)地址.

主存的主要性能指标

按字节连续编址，每个存储单元为1个字节（8个二进位）

性能指标：

存储容量：所包含的存储单元的总数（单位：MB或GB）

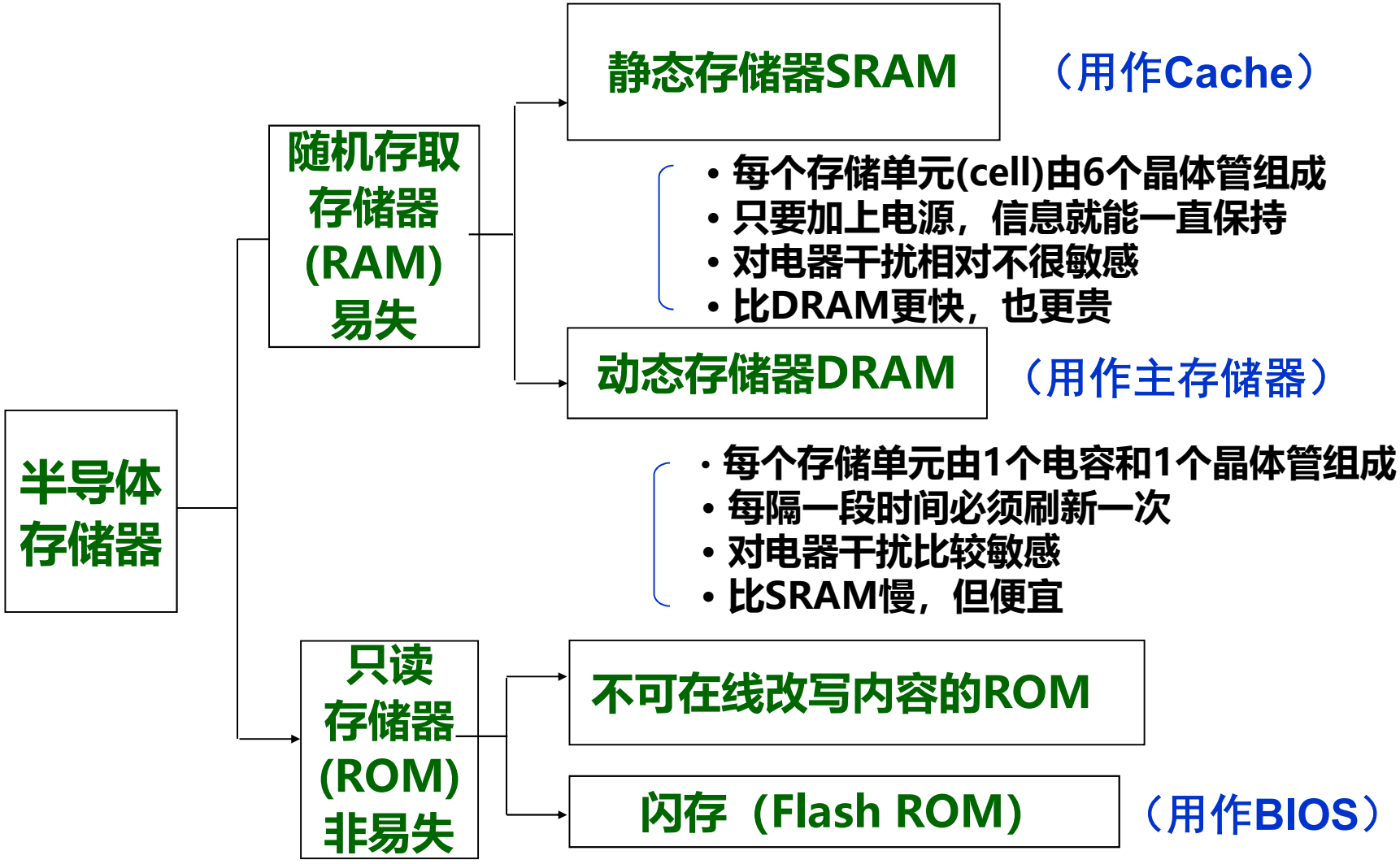
存取时间T_A：从CPU送出内存单元的地址码开始，到主存读出数据并送到CPU（或者是把CPU数据写入主存）所需要的时间（单位：ns，1 ns = 10^{-9} s），分读取时间和写入时间

存储周期T_{MC}：连读两次访问存储器所需的最小时间间隔，它应等于存取时间加上下一次存取开始前所要求的附加时间，因此，T_{MC}比T_A大（因为存储器由于读出放大器、驱动电路等都有一段稳定恢复时间，所以读出后不能立即进行下一次访问。）

（就像一趟火车运行时间和发车周期是两个不同概念一样。）

内存储器的分类及应用

内存由半导体存储器芯片组成，芯片有多种类型：



存储技术趋势

SRAM

度量标准	1985	1990	1995	2000	2005	2010	2015	*2015:1985*
美元/MB	2,900	320	256	100	75	60	*25*	*116*
访问时间 (ns)	150	35	15	3	2	1.5	*1.3*	*115*

DRAM

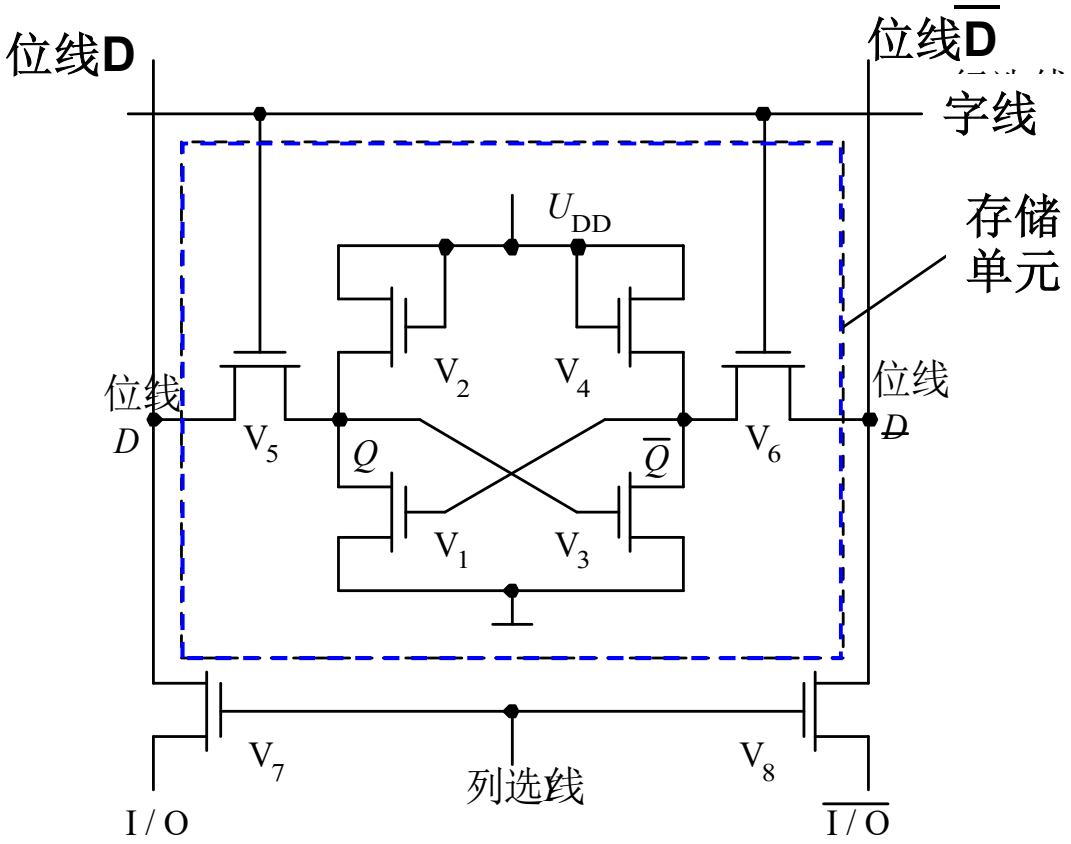
度量标准	1985	1990	1995	2000	2005	2010	2015	*2015:1985*
美元/MB	880	100	30	1	0.1	0.06	0.02	*44,000*
访问时间 (ns)	200	100	70	60	50	40	20	*10*
典型的大小 (MB)	0.256	4	16	64	2,000	8,000	16.000	*62,500*

磁盘

度量标准	1985	1990	1995	2000	2005	2010	2015	*2015:1985*
美元/GB	100,000	8,000	300	10	5	0.3	0.03	*3,333,333*
访问时间 (ms)	75	28	10	8	*5*	*3*	*3*	*25*
典型的大小 (GB)	0.01	0.16	1	20	160	1,500	3,000	*300,000*

六管静态MOS管电路（不作要求）

6管静态NMOS记忆单元



SRAM中数据保存在一对正负反馈门电路中，只要供电，数据就一直保持，不是破坏性读出，也无需重写，即无需刷新！

信息存储原理： 看作带时钟的RS触发器

写入时：
- 置字线为1
- 位线上是被写入的二进位信息0或1
- 存储单元(触发器)按位线的状态设置成0或1

读出时：
- 置字线为1
- 置2个位线为高电平
- 存储单元状态不同，位线的输出不同

保持时：
- 字线为0（低电平）

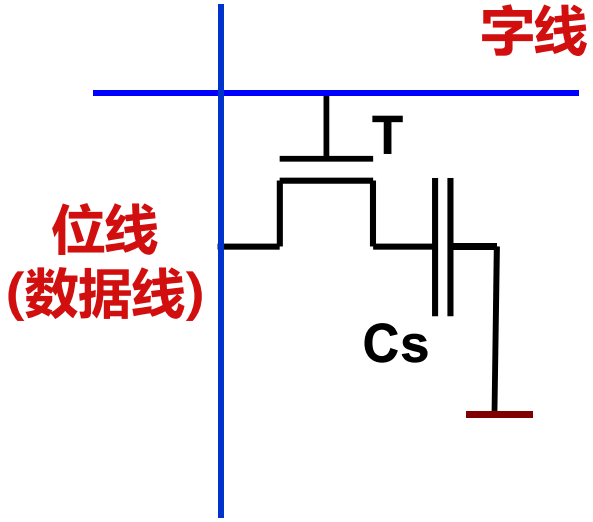
动态单管记忆单元电路（不作要求）

读写原理：字线上加高电平，使T管导通。

写“0”时，数据线加低电平，使C_S上电荷对数据线放电；

写“1”时，数据线加高电平，使数据线对C_S充电；

读出时，数据线上有一读出电压。它与C_S上电荷量成正比。



优点：电路元件少，功耗小，集成度高，用于构建主存储器

缺点：速度慢、是破坏性读出（需读后再生）、需定时刷新

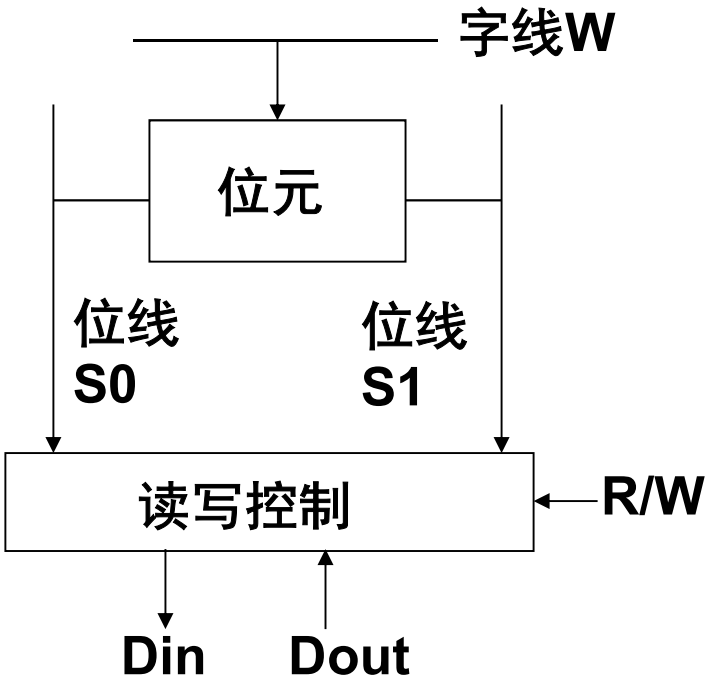
刷新：DRAM的一个重要特点是，数据以电荷的形式保存在电容中，电容的放电使得电荷通常只能维持几十个毫秒左右，相当于1M个时钟周期左右，因此要定期进行刷新（读出后重新写回），按行进行（所有芯片中的同一行一起进行），刷新操作所需时间通常只占1%~2%左右。

半导体RAM的组织

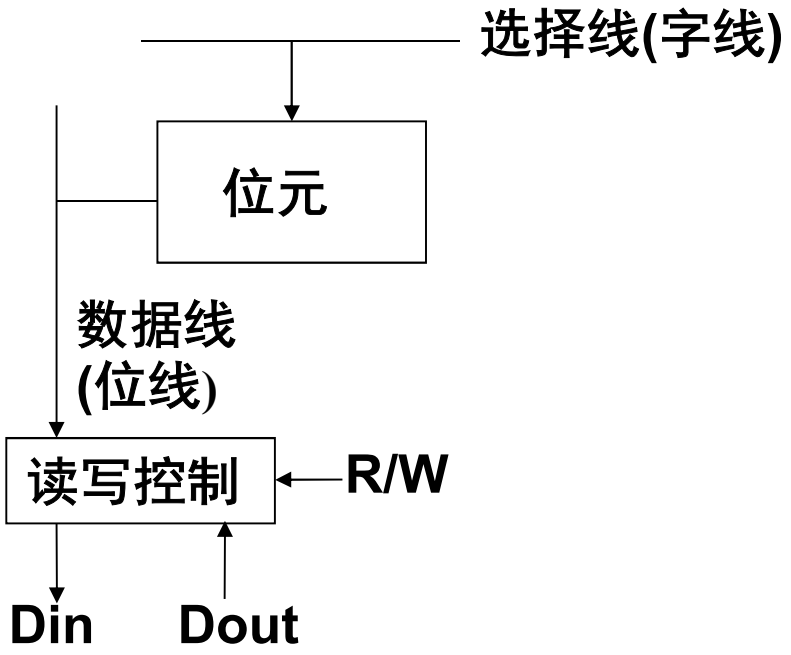
记忆单元(Cell)→存储器芯片(Chip)→ 内存条（存储器模块）

存储体(Memory Bank)： 由记忆单元(位元)构成的存储阵列

记忆单元的组织：

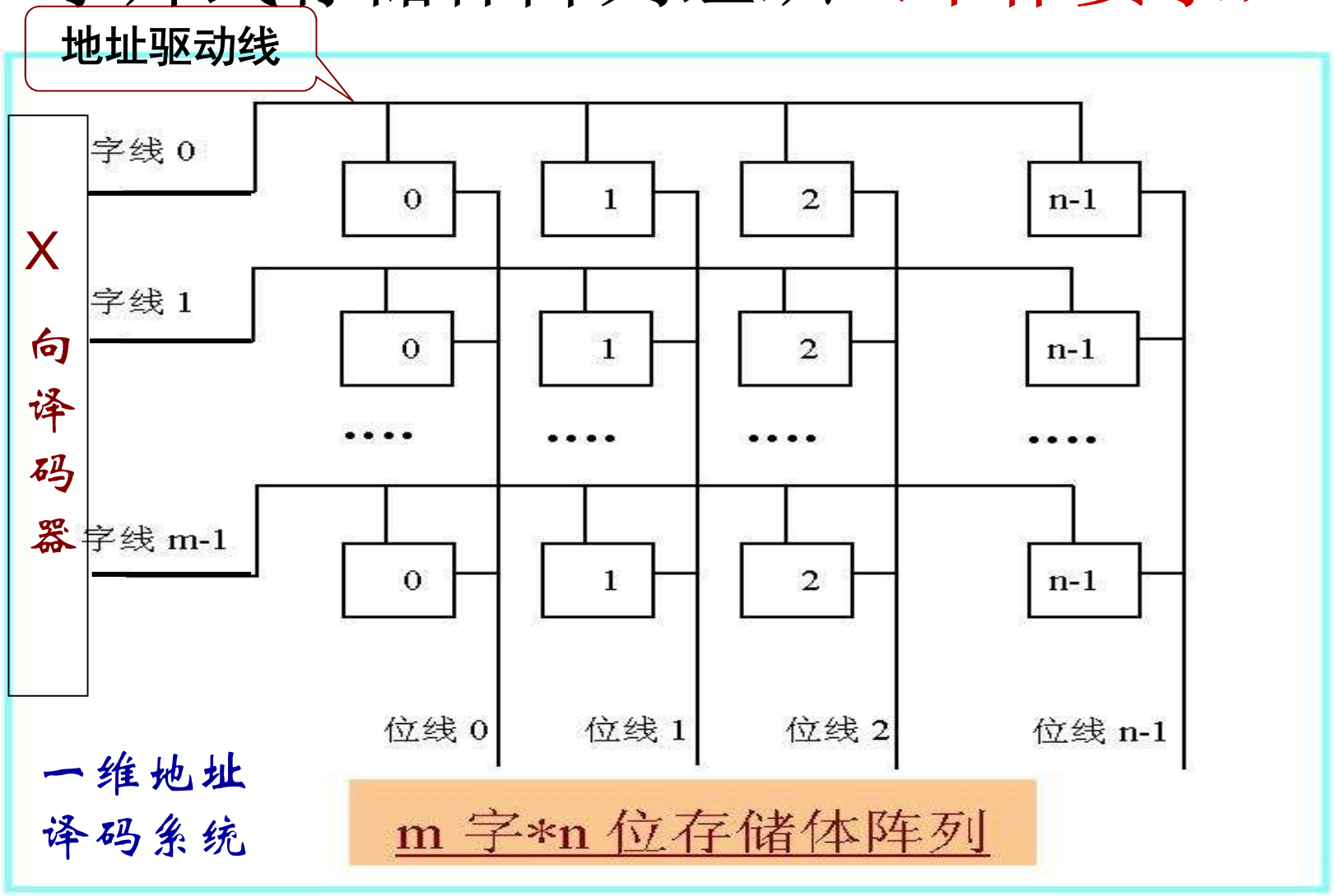


SRAM



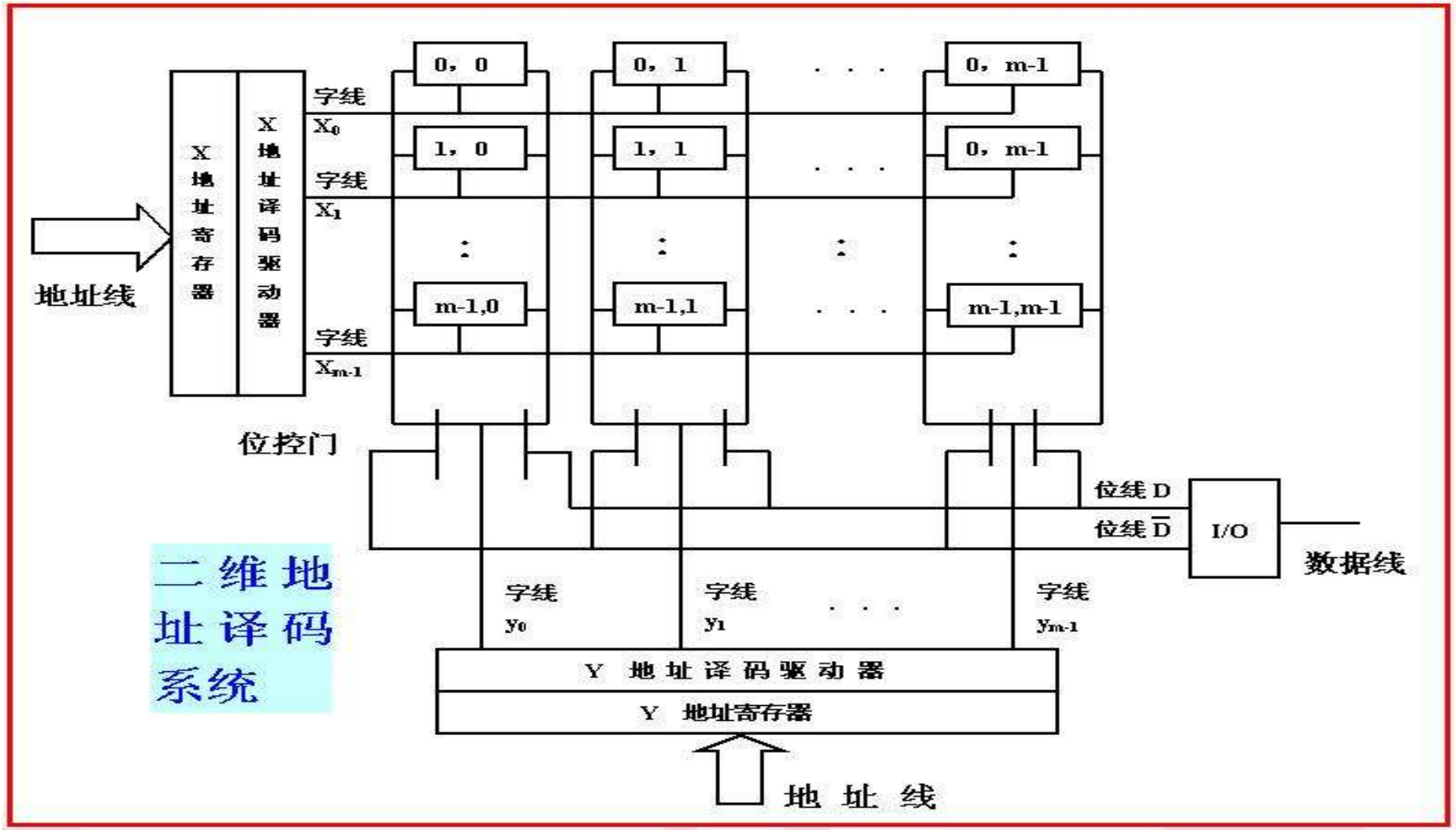
DRAM

字片式存储体阵列组织（不作要求）



一般SRAM为字片式芯片，只在x向上译码，同时读出字线上所有位！

位片式存储体阵列组织（不作要求）



位片式在字方向和位方向扩充，需要有片选信号
DRAM芯片都是位片式

举例：典型的16M位DRAM（4Mx4）

16M位 = 4Mbx4 = 2048x2048x4 = 2^{11}x2^{11}x4

(1) 地址线：11根线分时复用，由RAS和CAS提供控制时序。

(2) 需4个位平面，对相同行、列交叉点的4位一起读/写

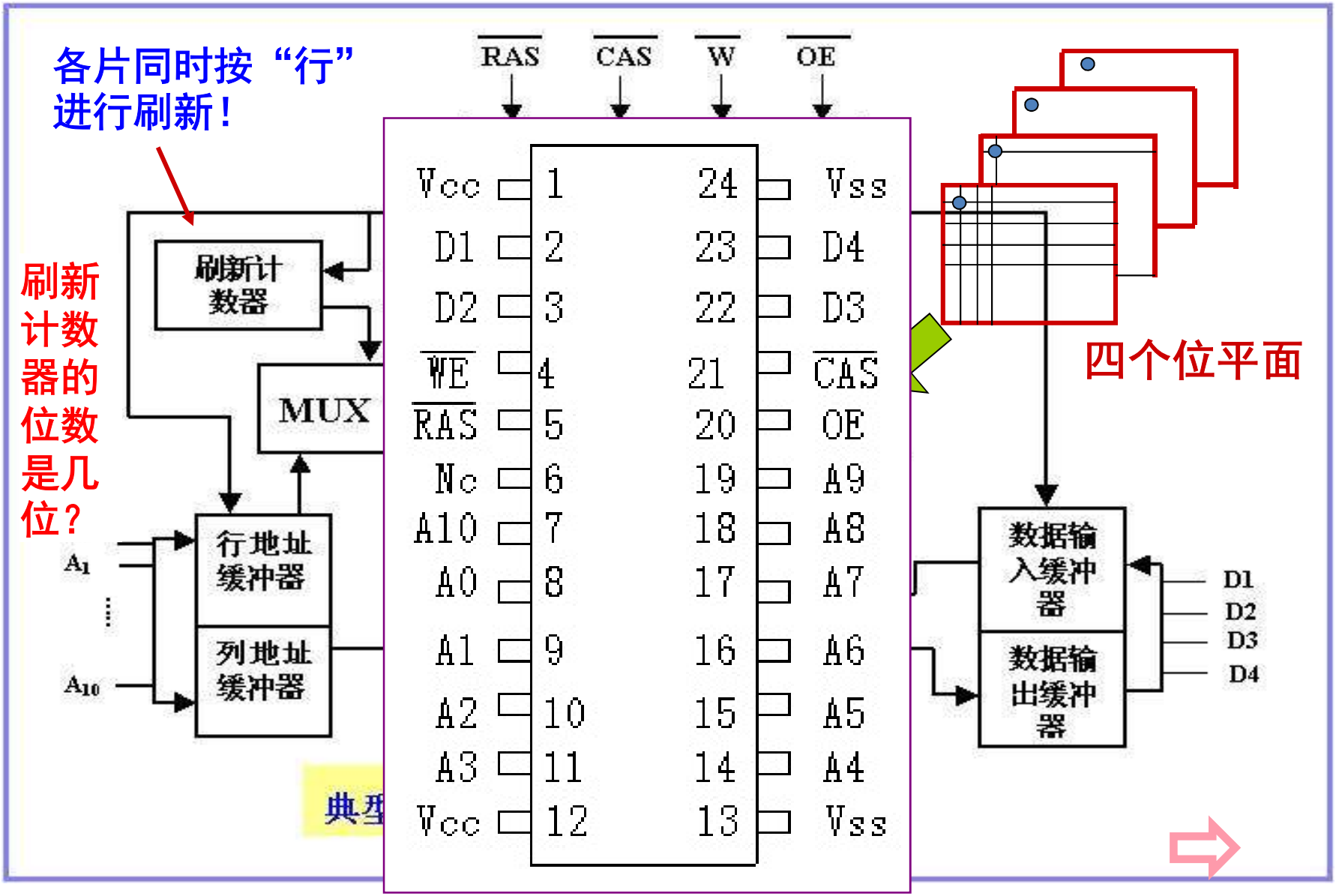
(3) 内部结构框图

问题：

为什么每出现新一代DRAM芯片，容量至少提高到4倍?

行地址和列地址分时复用，每出现新一代DRAM芯片，至少要增加一根地址线。每加一根地址线，则行地址和列地址各增加一位，所以行数和列数各增加一倍。因而容量至少提高到4倍。

举例：典型的16M位


各片同时按“行”进行刷新！
刷新计数器的位数是几位？
RAS
CAS
W
OE
刷新计数器
MUX
行地址缓冲器
列地址缓冲器
A1
A10
Vcc 1
D1 2
D2 3
WE 4
RAS 5
Nc 6
A10 7
A0 8
A1 9
A2 10
A3 11
Vcc 12
24 Vss
23 D4
22 D3
21 CAS
20 OE
19 A9
18 A8
17 A7
16 A6
15 A5
14 A4
13 Vss
四个位平面
数据输入缓冲器
数据输出缓冲器
D1
D2
D3
D4
典型

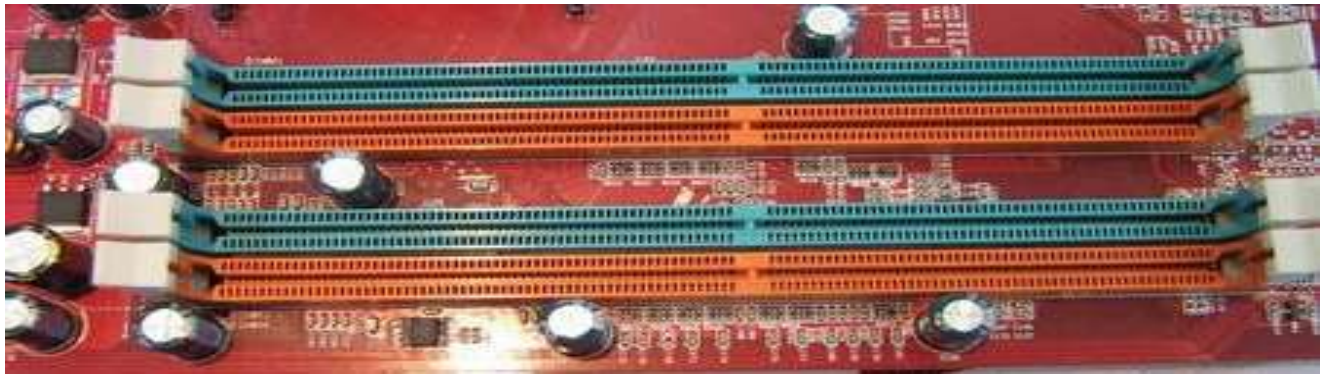
PC机主存储器的物理结构

由若干内存条组成

内存条的组成：

把若干片DRAM芯片焊装在一小条印制电路板上制成

内存条必须插在主板上的内存条插槽中才能使用

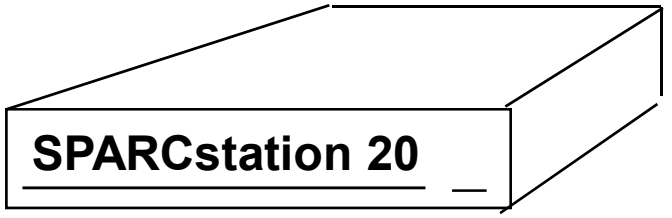


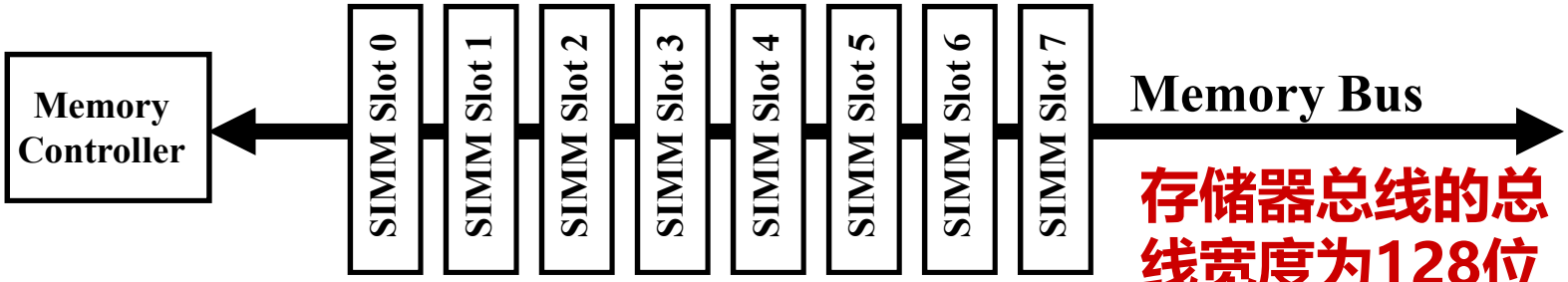
目前流行的是DDR2、DDR3内存条：

- **采用双列直插式，其触点分布在内存条的两面**
- **DDR条有184个引脚，DDR2有240个引脚**
- **PC机主板中一般都配备有2个或4个DIMM插槽**

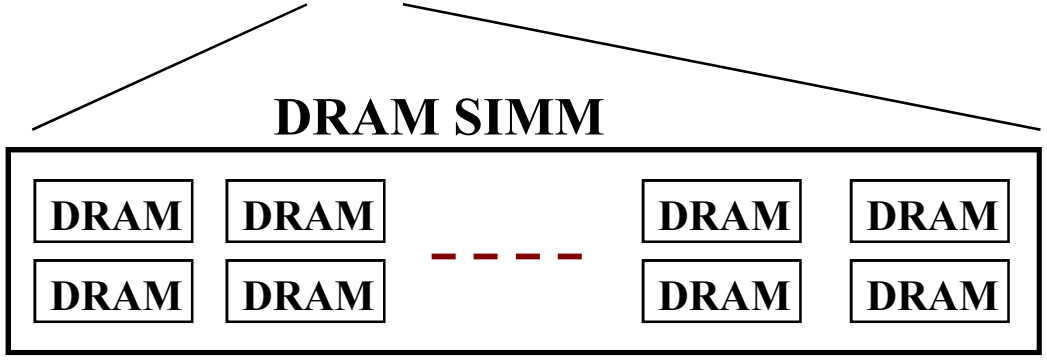
举例： SPARCstation 20’s Memory Module

总线宽度是指总线中数据线的条数





存储器总线的总线宽度为128位



每个内存条最多能同时读出128位数据

每次访存操作总是在某一个内存条内进行！

举例： SPARCstation 20’s内存条(模块)

one memory module （内存条）

Smallest: 4 MB = 16x 2Mb DRAM chips, 8 KB of Page SRAM

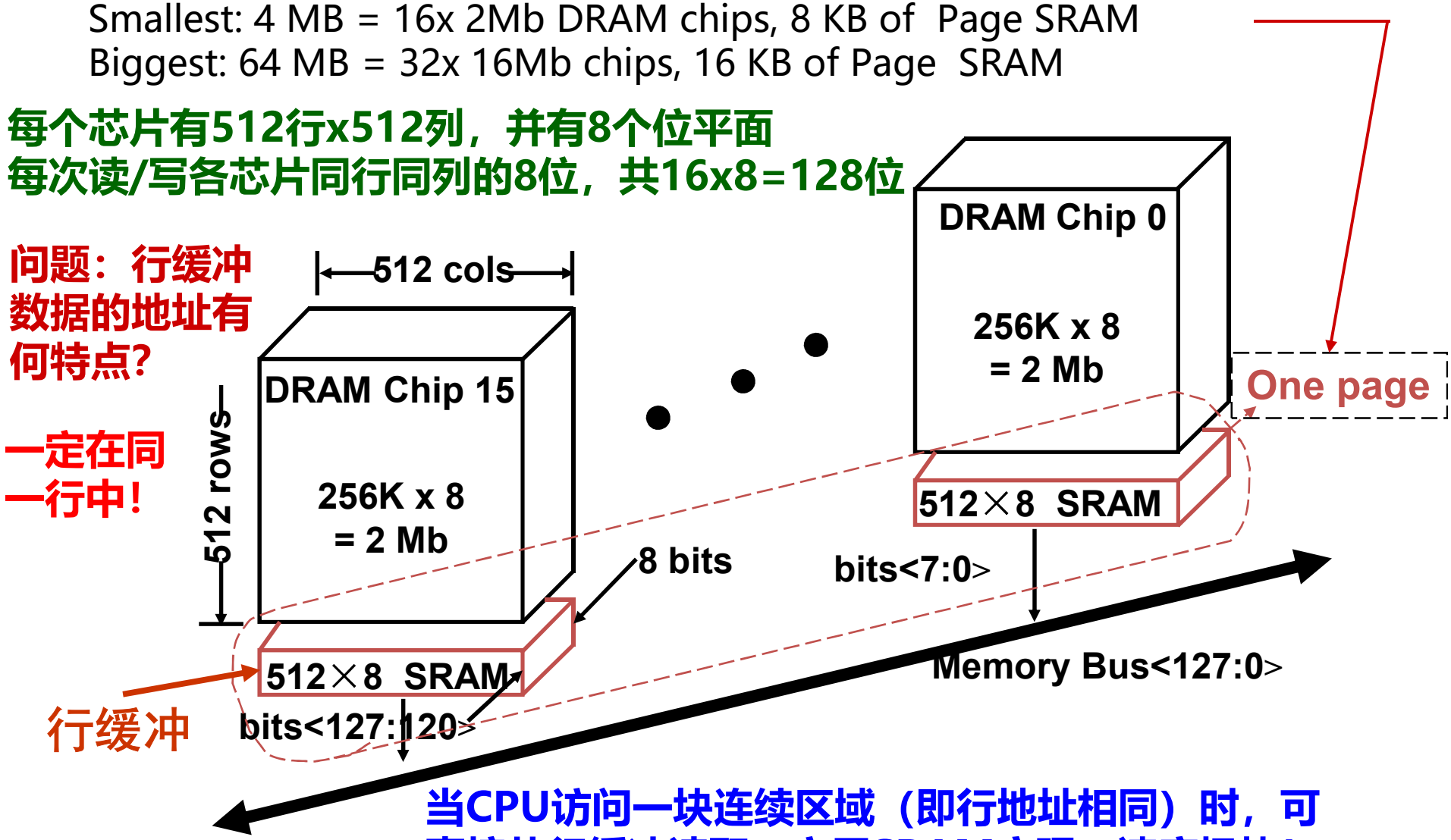
Biggest: 64 MB = 32x 16Mb chips, 16 KB of Page SRAM

每个芯片有512行x512列，并有8个位平面

每次读/写各芯片同行同列的8位，共16x8=128位

问题：行缓冲数据的地址有何特点?

一定在同一行中!



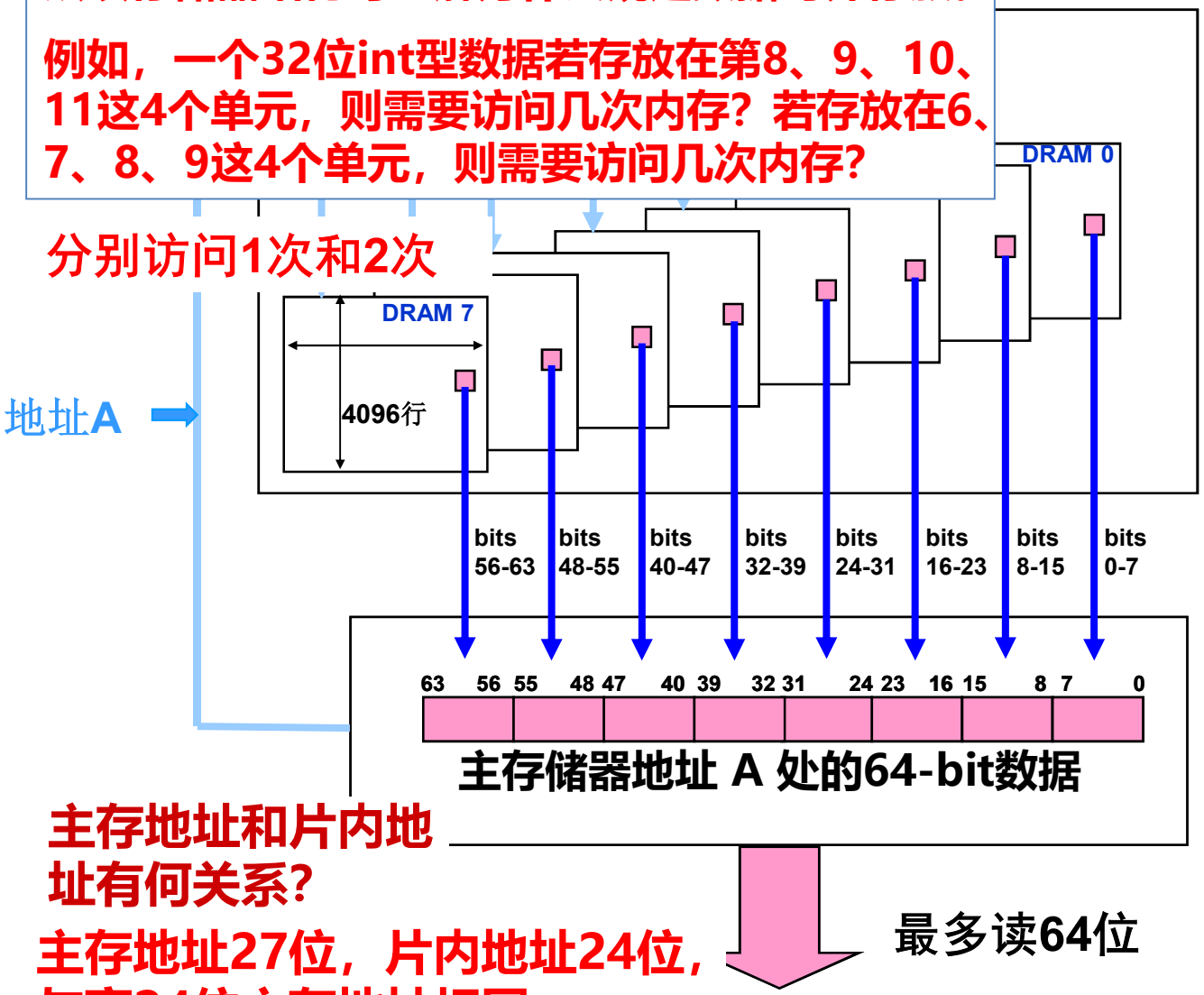
当CPU访问一块连续区域（即行地址相同）时，可直接从行缓冲读取，它用SRAM实现，速度极快!

举例： 128MB的DRAM存储器

从该存储器结构可理解为什么规定数据对齐存放。

例如，一个32位int型数据若存放在第8、9、10、11这4个单元，则需要访问几次内存？若存放在6、7、8、9这4个单元，则需要访问几次内存?

分别访问1次和2次



- 由8片DRAM芯片构成
- 每片 16Mx8 bits
- 行地址、列地址各12位
- 每行共4096列(8位/列)
- 选中某一行并读出之后再由列地址选择其中的一列(8个二进位) 送出

芯片内地址是否连续?

不连续，交叉编址，可同时读写所有芯片。

- 行、列地址为(i,j)的8个单元

主存地址和片内地址有何关系?

主存地址27位，片内地址24位，与高24位主存地址相同。

主存低3位地址的作用是什么?

确定8个字节中的哪个，即用来选片。

DRAM芯片的规格

若一个$2^n \times b$位DRAM芯片的存储阵列是r行×c列，则该芯片容量为$2^n \times b$位且$2^n = r \times c$。如：16K×8位DRAM，则r=c=128。

芯片内的地址位数为n，其中行地址位数为$\log_2 r$，列地址位数为$\log_2 c$。

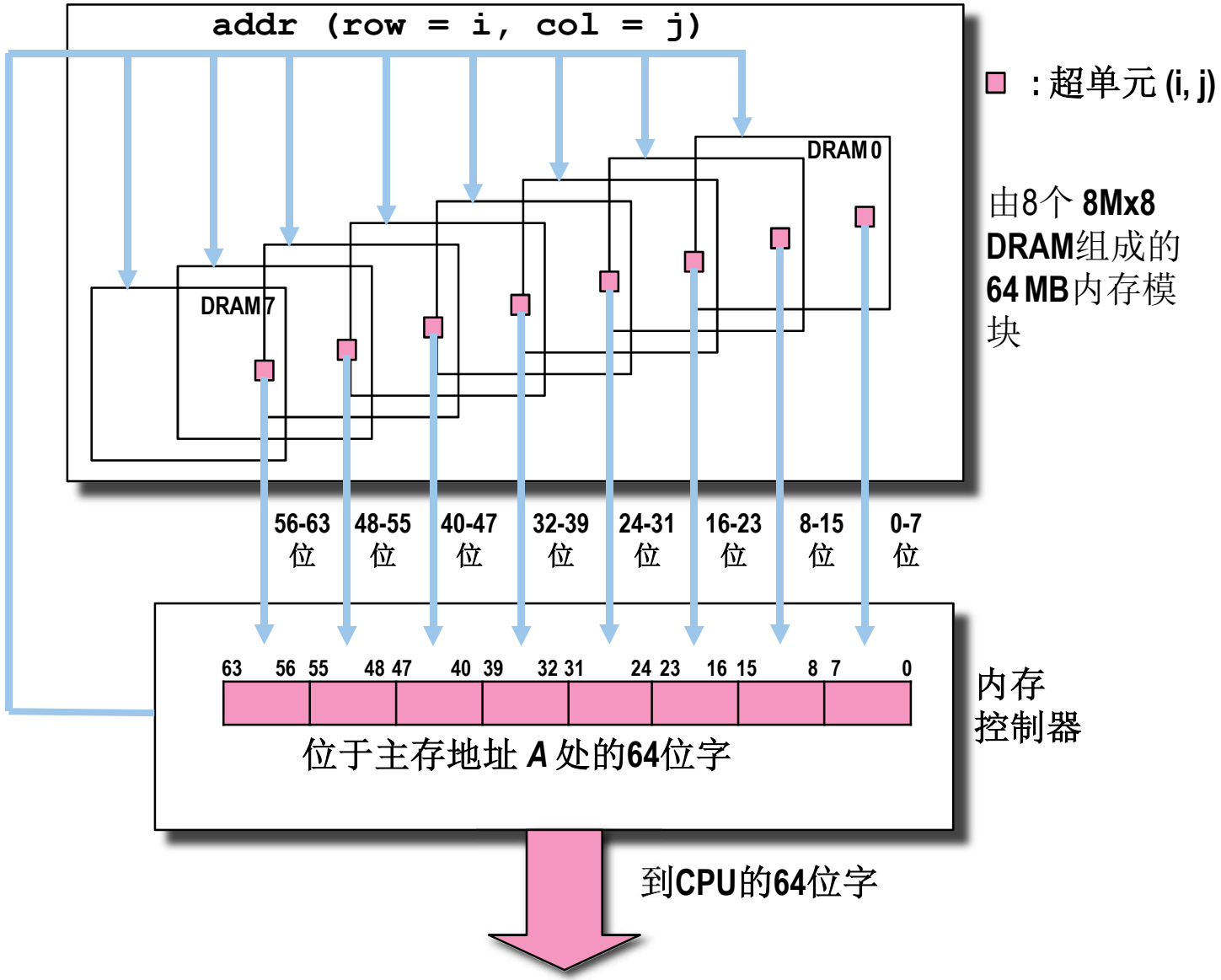
如：16K×8位DRAM，则n=14，行、列地址各占7位。

n位地址中高位部分为行地址，低位部分为列地址

为提高DRAM芯片的性价比，通常设置的r和c满足r≤c且|r-c|最小。

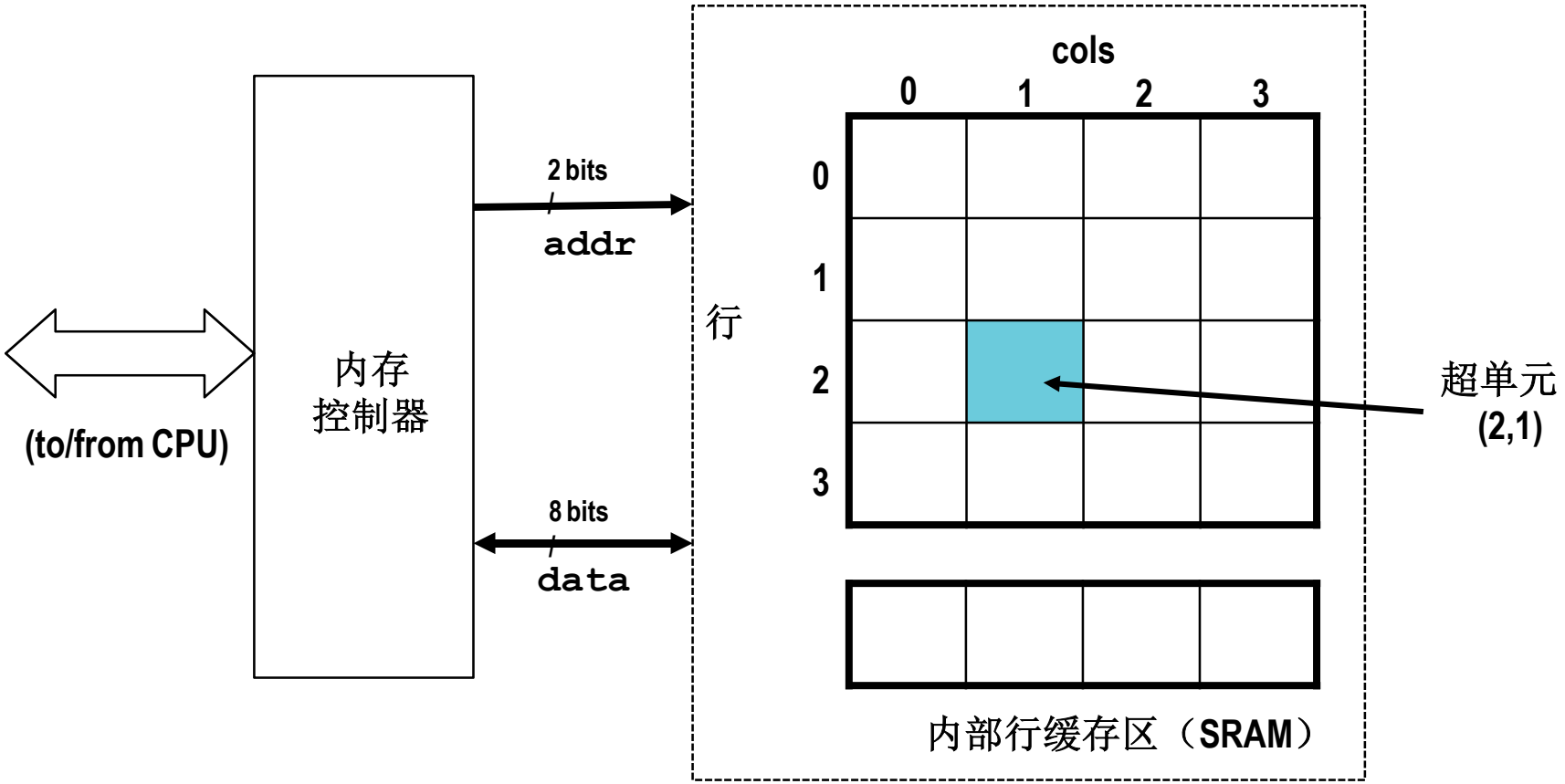
例如，对于8K×8位DRAM芯片，其存储阵列设置为2^6行×2^7列，因此行地址和列地址的位数分别为6位和7位，13位芯片内地址A_{12} $A_{11}...A_1A_0$中，行地址为A_{12} $A_{11}...A_7$，列地址为$A_6...A_1A_0$。因按行刷新，为尽量减少刷新次数，故行数越少越好，但是，为了减少地址引脚，应尽量使行、列地址位数一致

内存模块


addr (row = i, col = j)
DRAM 0
DRAM 7
56-63 位
48-55 位
40-47 位
32-39 位
24-31 位
16-23 位
8-15 位
0-7 位
63 56 55 48 47 40 39 32 31 24 23 16 15 8 7 0
位于主存地址 A 处的64位字
到CPU的64位字
: 超单元 (i, j)
由8个 8Mx8 DRAM组成的 64 MB内存模块
内存控制器

主存模块的连接和读写操作

DRAM芯片内部结构示意图

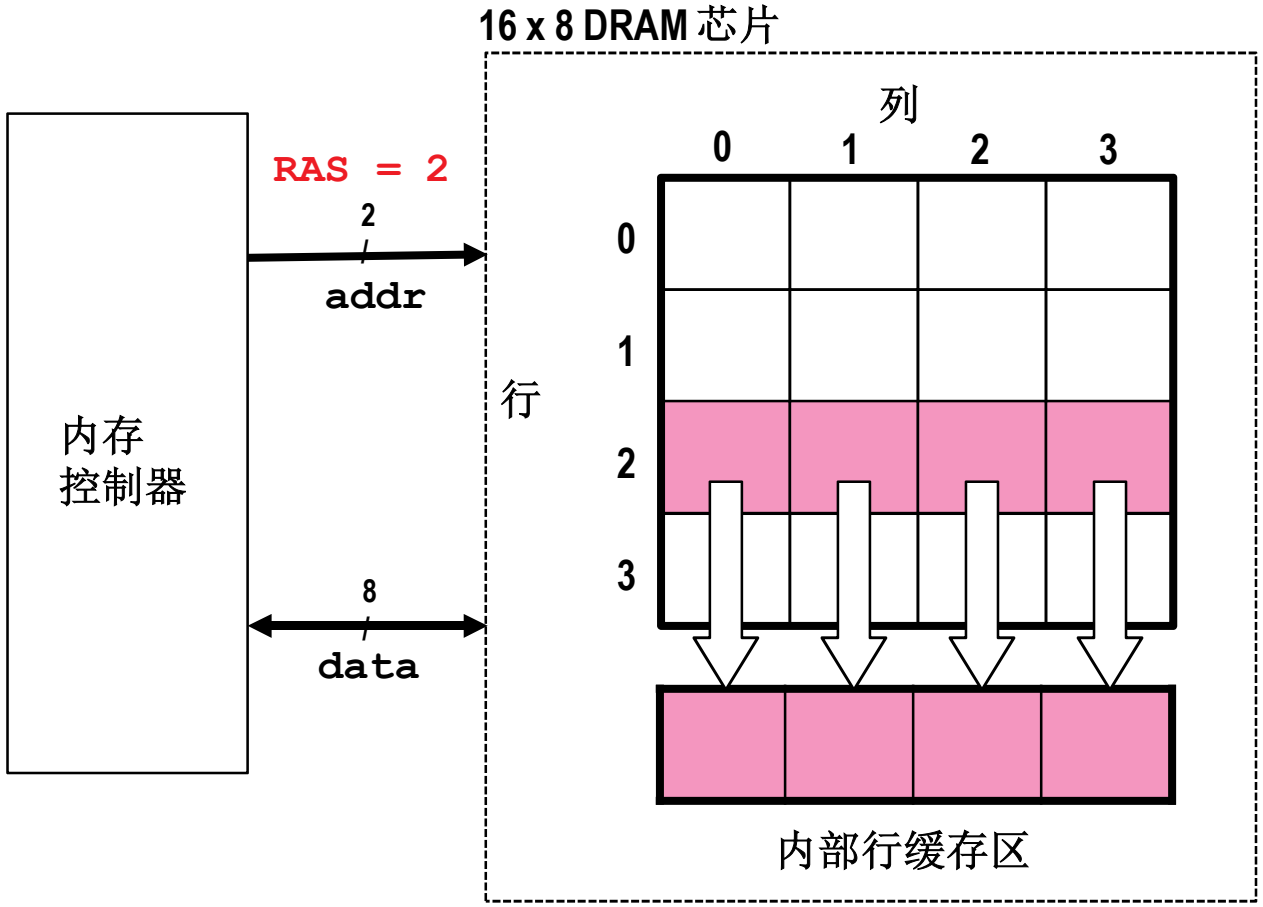


图中芯片容量为16×8位，存储阵列为4行×4列，地址引脚采用复用方式，因而仅需2根地址引脚，每个超元（supercell）有8位，需8根数据引脚，有一个内部的行缓冲（row buffer），通常用SRAM元件实现。

读 DRAM 超单元 (2,1)

Step 1(a): 行访问选通脉冲(**RAS**) 选中行 **2** 。 行列地址复用

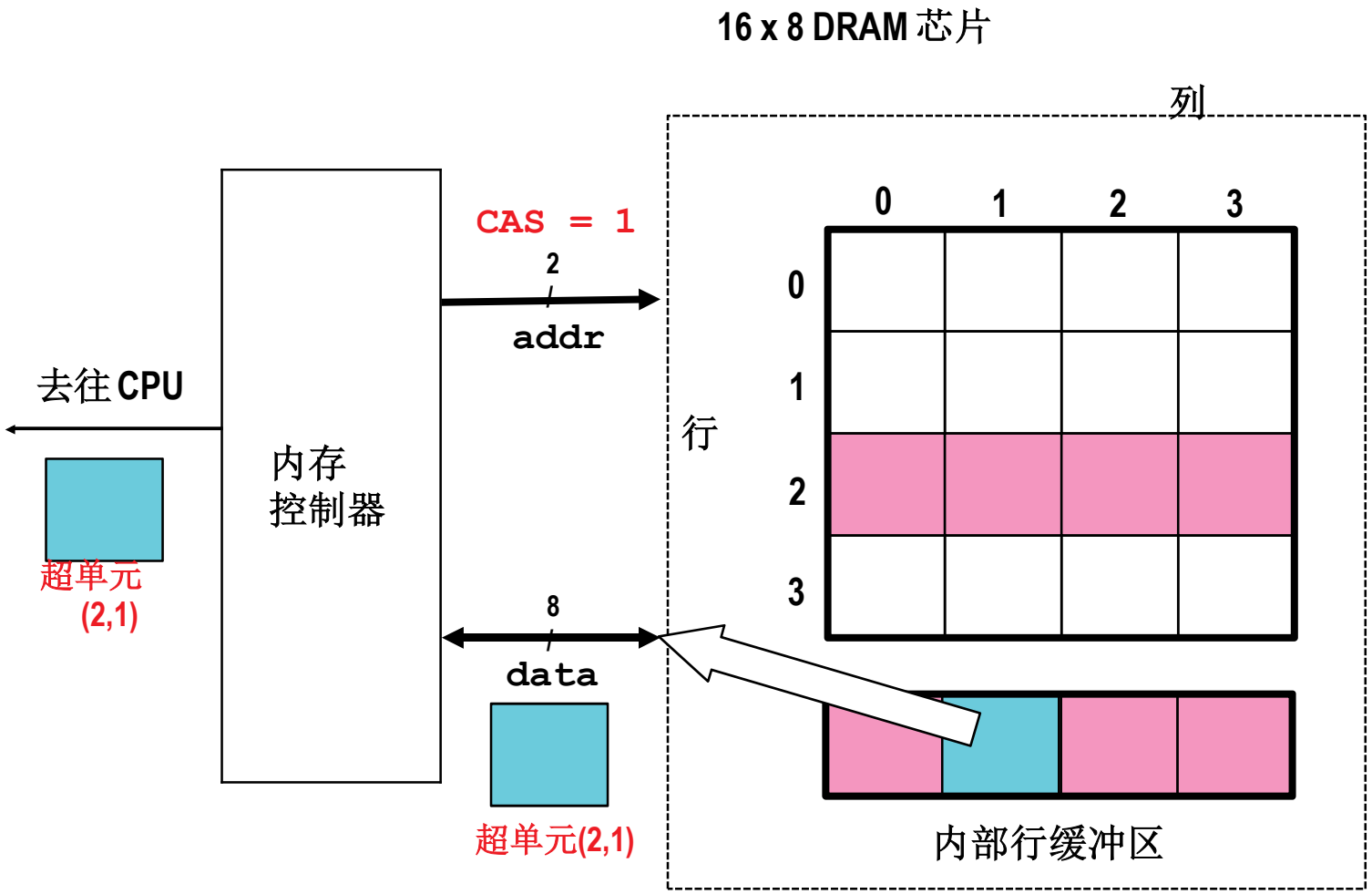
Step 1(b): 行 **2** 的整个内容复制到内部行缓存区。



读 DRAM 超单元 (2,1)

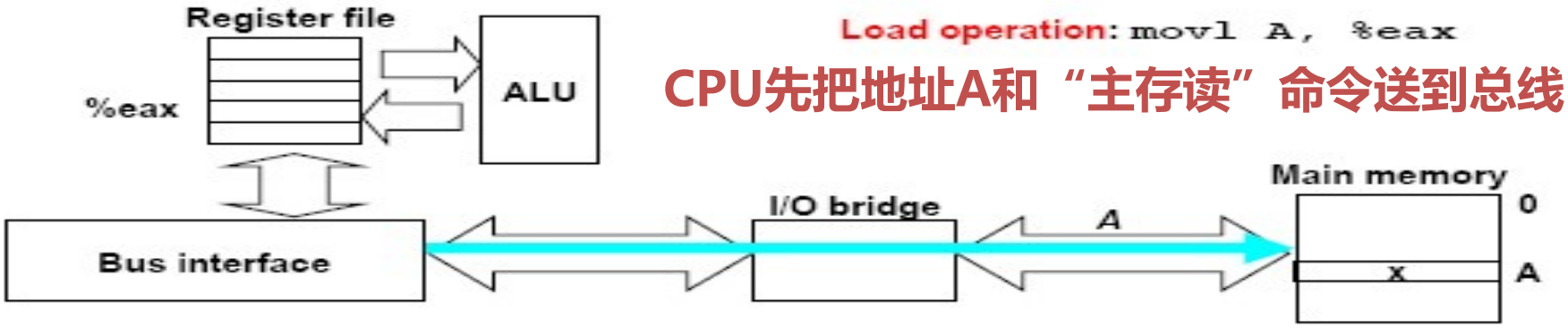
Step 2(a): 列访问选通脉冲 (**CAS**) 选中列 **1**。行列地址复用

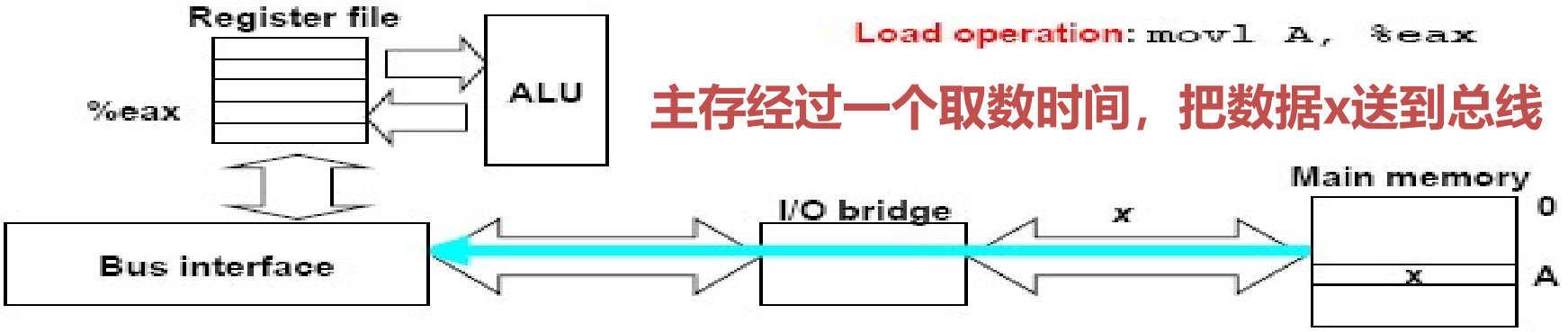
Step 2(b): 超单元 **(2,1)** 从内部行缓存区复制到**data**线上，最终发送给**CPU**。

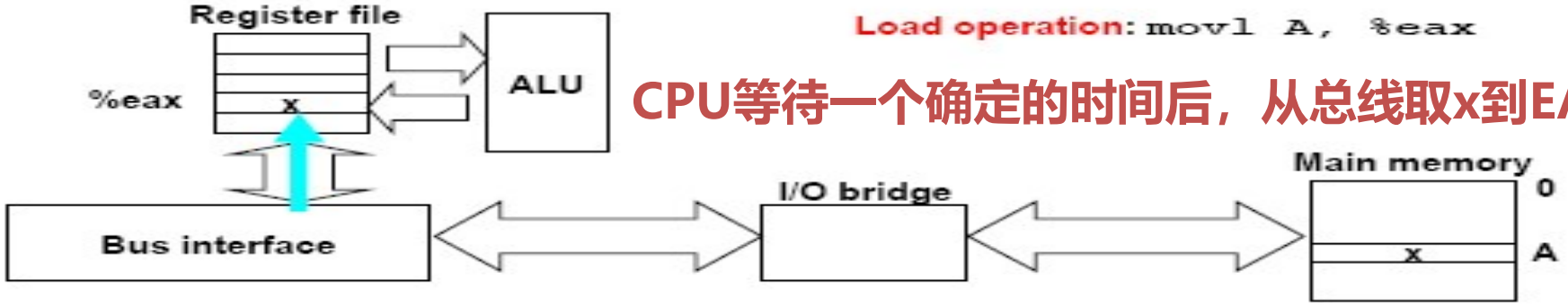


指令“movl 8(%ebp), %eax”操作过程

由8(%ebp)得到地址A的过程较复杂，涉及MMU、TLB、页表等重要概念！







SRAM vs DRAM一览

	每位晶体管数	访问时间	持续的？	敏感的?	花销	应用
SRAM	4 或 6	1X	否	可能	1000x	高速缓存
DRAM	1	10X	是	是	1X	主存, 帧缓冲区

增强的DRAMs

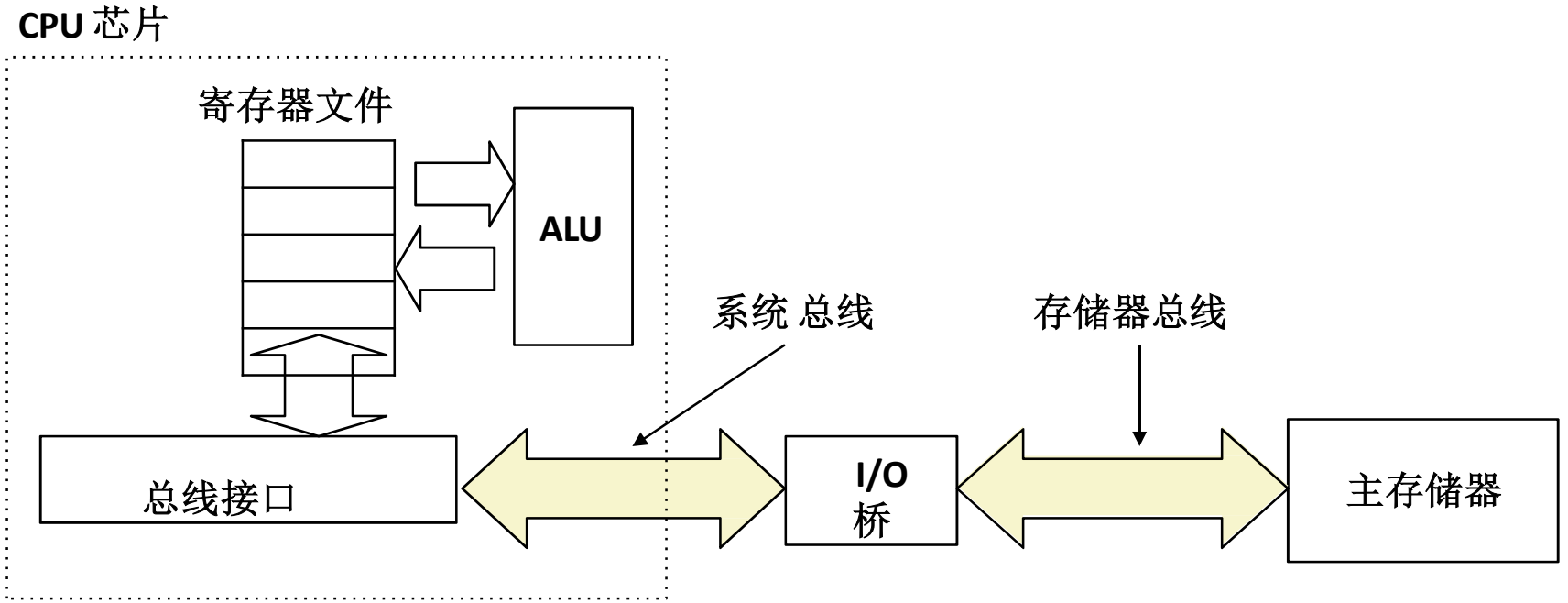
- 自1966年DRAM问世以来，其基本单元就没有变化。
  - Intel 于1970年将其推向市场
- DRAM 集成了更好的接口逻辑与更快的I/O传输接口：
  - 同步 DRAM (SDRAM)
    - 使用常见的时钟信号取代异步控制信号
    - 允许行地址复用(比如, RAS, CAS, CAS, CAS)
  - 双倍数据速率同步DRAM (DDR SDRAM)
    - 每个时钟周期每个引脚使用两个时钟沿传送两比特控制信号
    - 以预取缓冲区的大小来划分不同类型:
      - DDR (2 bits), DDR2 (4 bits), DDR3 (8 bits)
    - 到2010年，多数服务器和桌面系统均支持该标准
    - Intel Core i7 支持DDR3 和 DDR4 SDRAM

非易失性存储器

- **DRAM 和 SRAM 是易失性存储器**
  - 断电数据丢失
- **非易失性存储器断电后，依然保持数据**
  - 只读存储器(ROM): 生产时写入程序，只能写一次
  - 可编程 ROM (PROM): 可以重新编程一次
  - 可擦除 PROM (EPROM): 可用紫外线整块擦除
  - 电可擦除PROM (EEPROM): 可用电子信号整块擦除
  - 闪存: 基于EEPROM, 以块为单位进行擦除
    - 100,000 次擦除后即磨损坏
- **非易失性存储器的应用**
  - 存储固件程序的ROM(BIOS,磁盘控制器，网卡,图形加速器, 安全子系统,...)
  - 固态硬盘(U盘, 智能手机, mp3播放器, 平板电脑, 笔记本电脑...)
  - 磁盘高速缓存

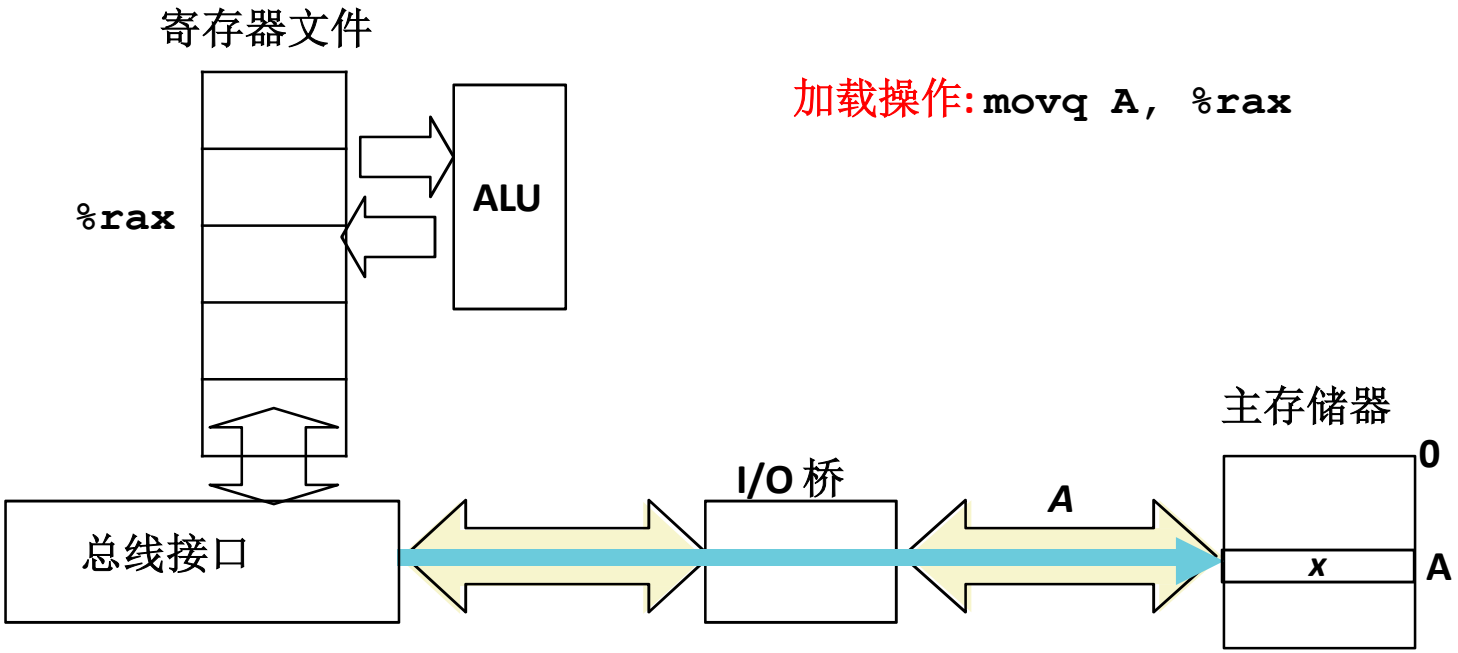
连接CPU和存储器的典型总线结构

- 一条总线(bus)是由多条并排的电线组成的一束线，其传输地址、数据和控制信号
- 多个设备共享多条总线



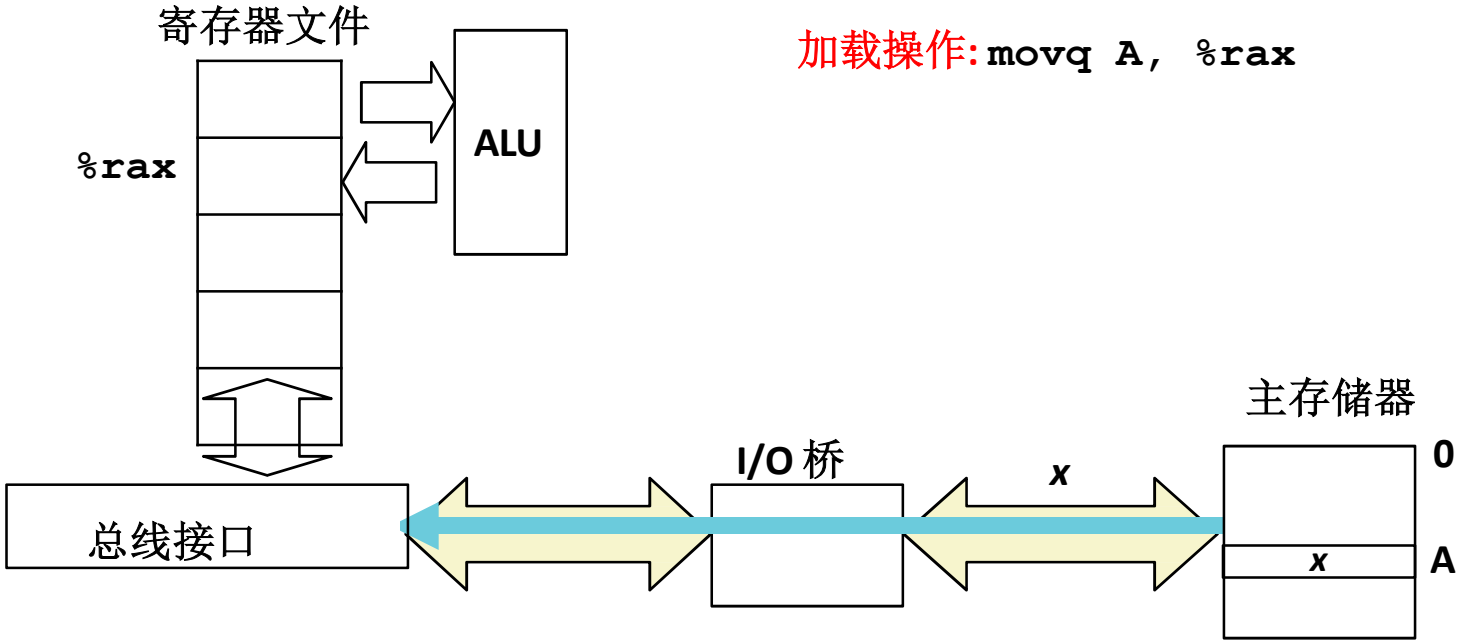
存储器读事务(1)

- CPU将地址A放到总线上。



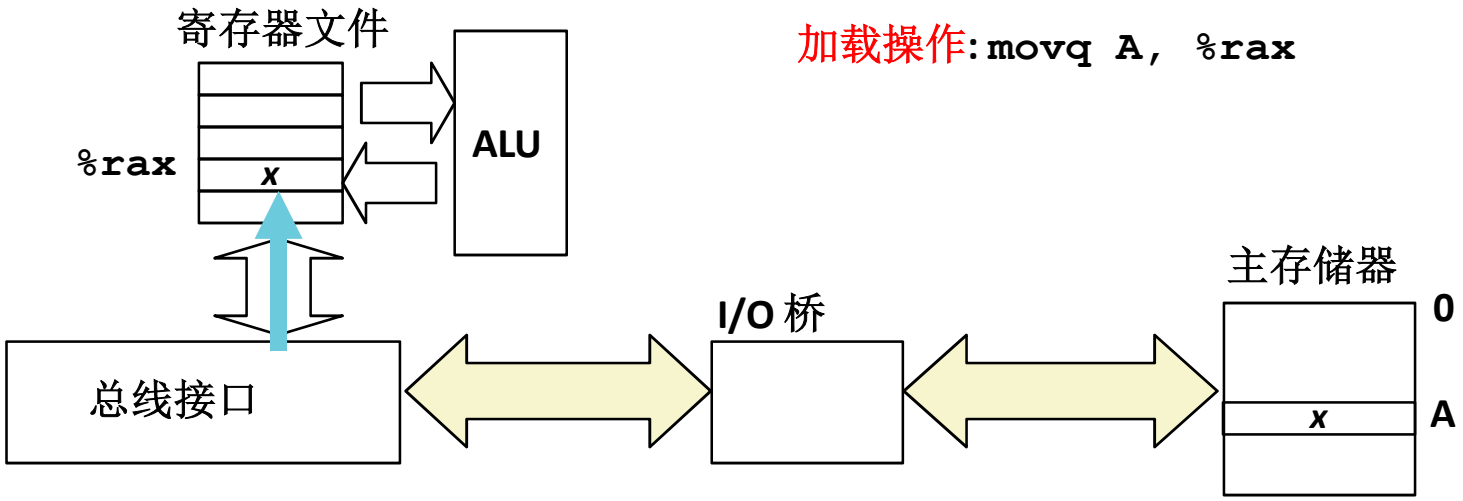
存储器读事务(2)

- 主存储器从总线上读地址A，取出字x, 然后将x放到总线上。



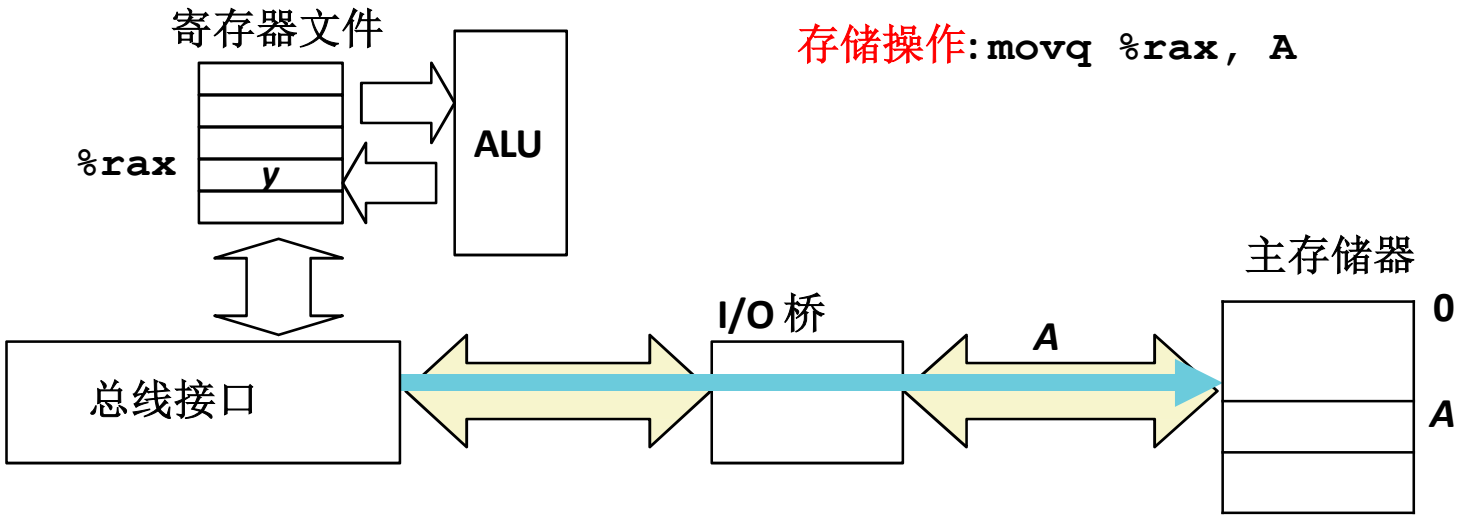
存储器读事务(3)

- CPU 从总线上读入字x, 并将其放入寄存器%rax



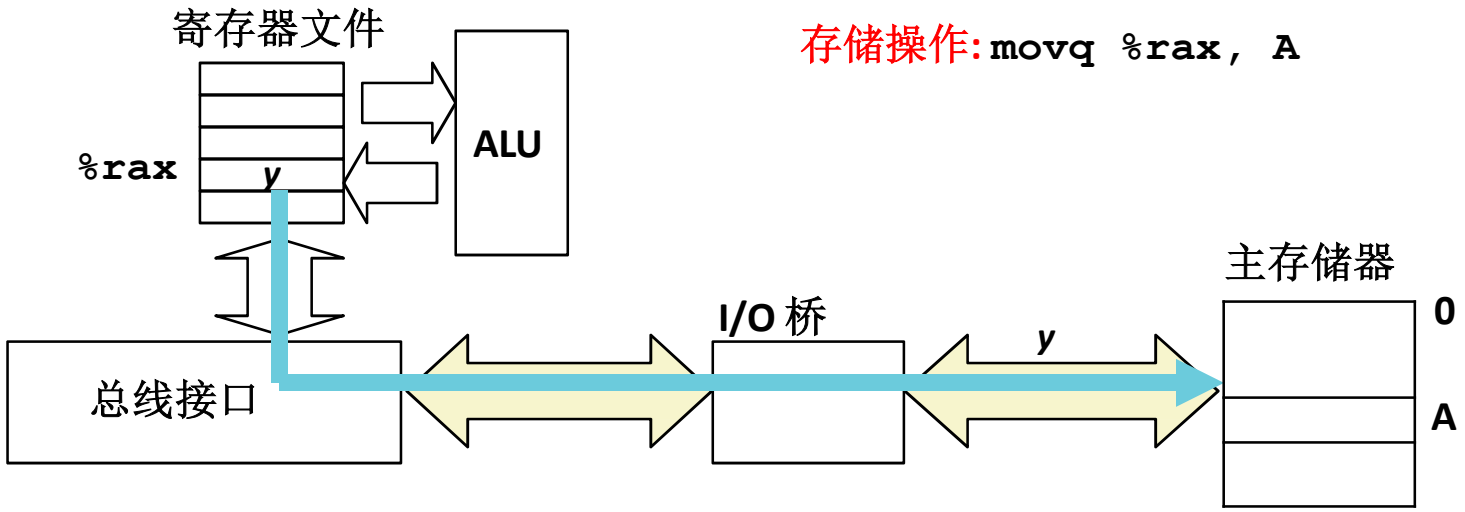
存储器写事务(1)

- CPU 将地址A放到总线上，主存储器读地址A并等待数据到来。



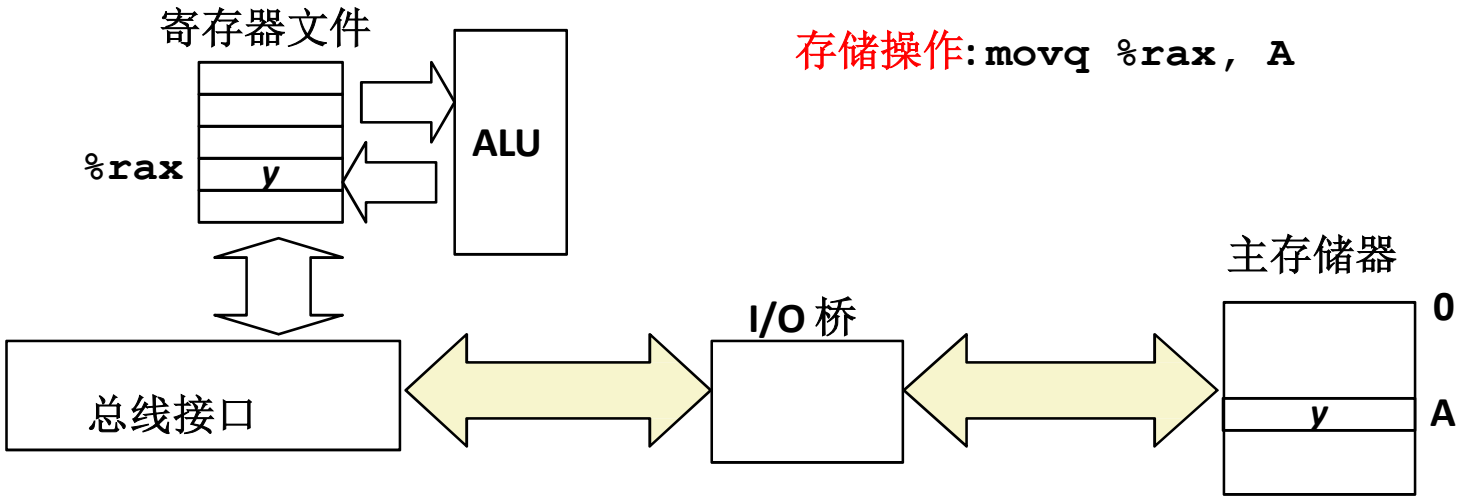
存储器写事务(2)

- CPU 将字y放到总线上。

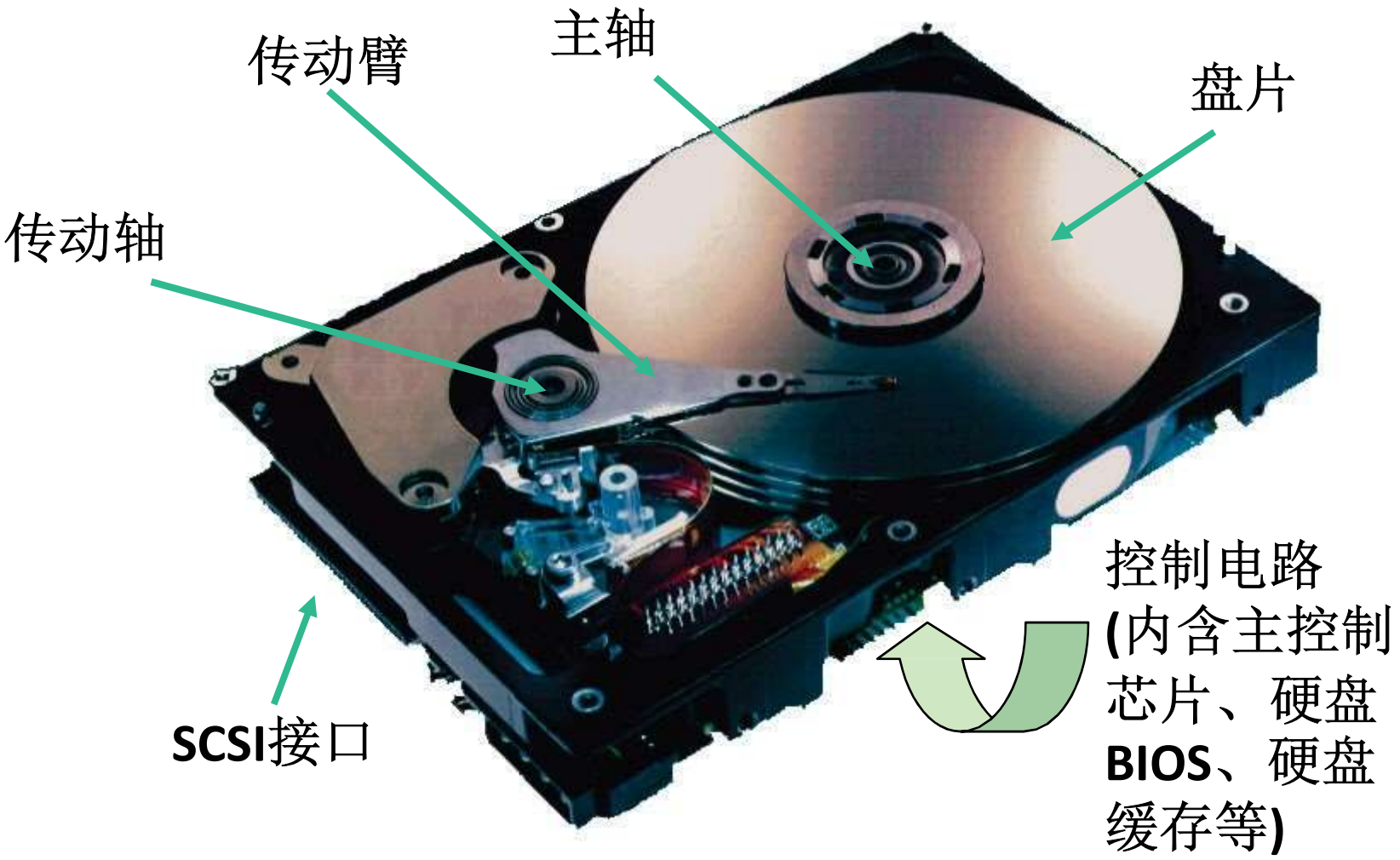


存储器写事务(3)

- 主存储器从总线上读入字**y**，并将其存入地址**A**中。

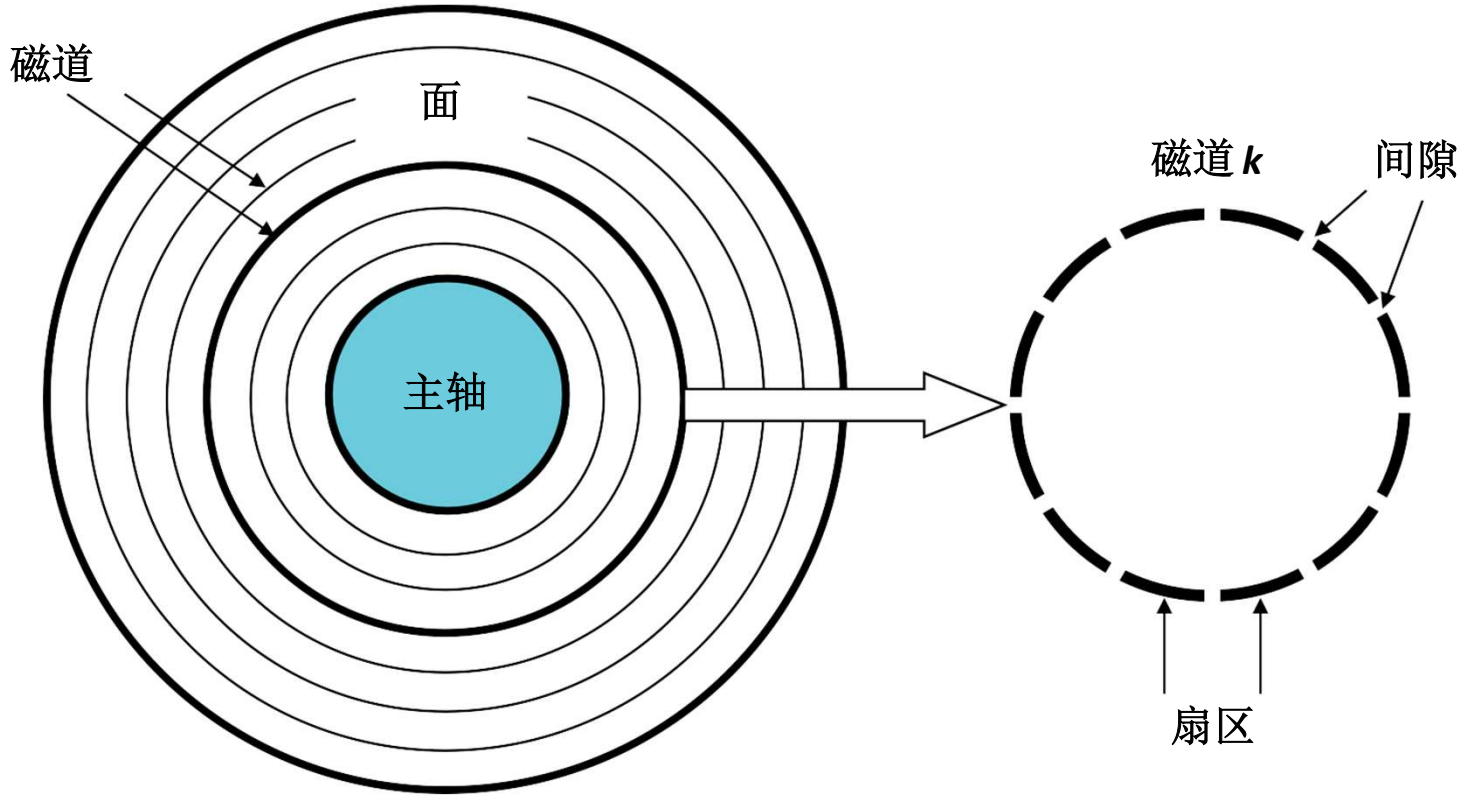


磁盘内部结构



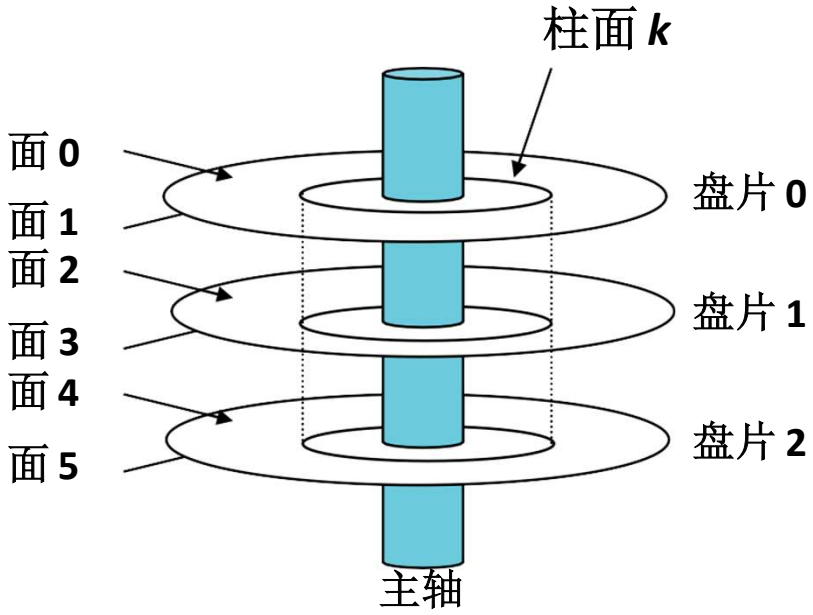
磁盘结构

- 磁盘由盘片(platter)构成，每个盘片包含两面(surface)。
- 每面由一组称为磁道(track)的同心圆组成。
- 每个磁道划分为一组扇区(sector)，扇区之间由间隙(gap)隔开。



磁盘结构(多个盘片)

- 同一半径上的所有磁道组成一个柱面。

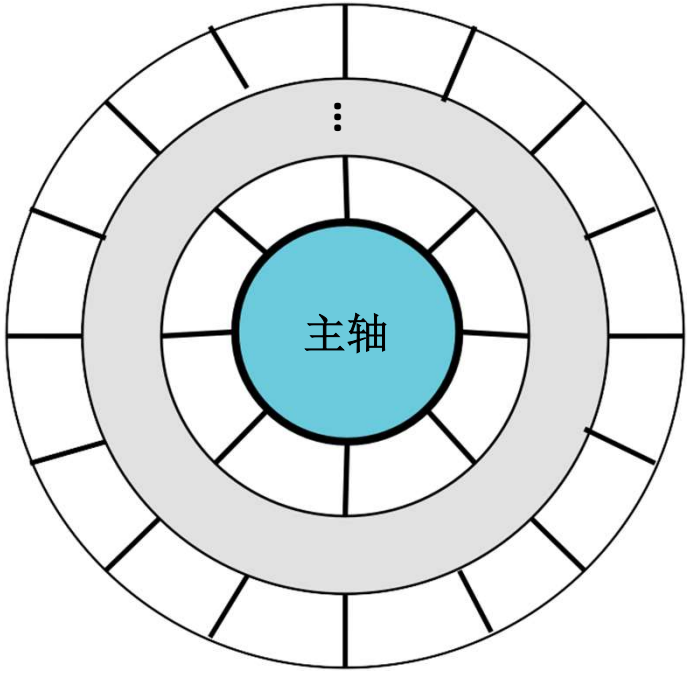


磁盘容量

- **容量(Capacity): 磁盘上可以存储的最大位(bits)数。**
  - 制造商以千兆字节(GB)为单位来表达磁盘容量，这里 1 GB = 10^9 字节。
- **磁盘容量由以下技术因素决定:**
  - 记录密度(Recording density) (位/英寸): 磁道一英寸的段中可放入的位数。
  - 磁道密度(Track density) (道/英寸): 从盘片中心出发半径上一英寸的段内可以有的磁道数。
  - 面密度(Areal density) (位/平方英寸): 记录密度与磁道密度的乘积。

分区记录

- 现代磁盘将所有磁道划分为若干分组(recording zone)，组内各磁道相邻
  - 区域内各磁道的扇区数目相同，扇区数取决于区域内最内侧磁道的圆周长。
  - 各区域的每磁道扇区数都不同，外圈区域的每磁道扇区数比内圈区域多.
  - 所以我们使用每磁道平均扇区数来计算磁盘容量。



计算磁盘容量

容量 = (字节数/扇区) x (平均扇区数/磁道) x (磁道数/面)
x (面数/盘片) x (盘片数/磁盘)

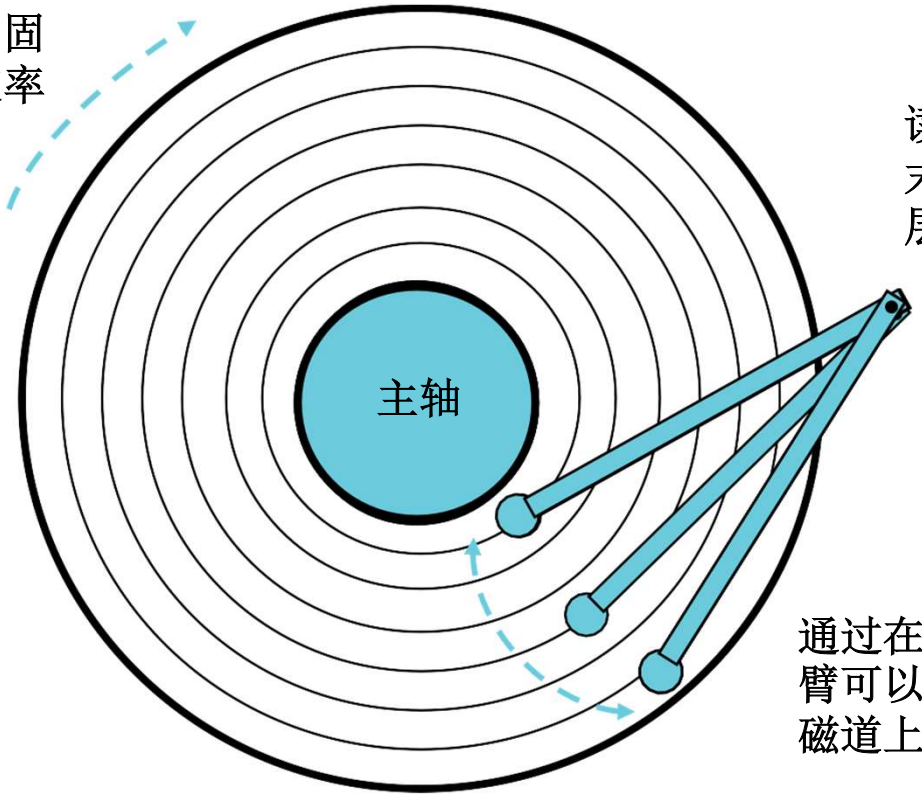
例:

- 512 字节/扇区
- 300 扇区/磁道 (平均值)
- 20,000 磁道/面
- 2 面/盘片
- 5 盘片/磁道

容量 = 512 x 300 x 20,000 x 2 x 5
= 30,720,000,000
= 30.72 GB

磁盘操作 (单盘片)

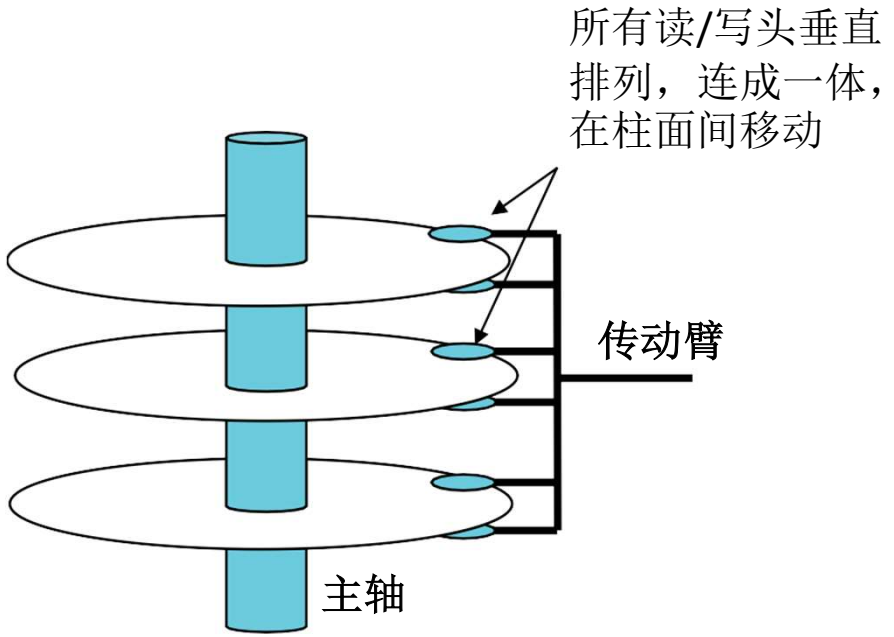
磁盘表面以固定的旋转速率旋转



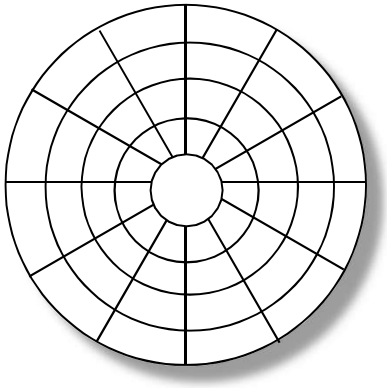
读/写头连到传动臂的末端，在磁盘表面上一层薄薄的气垫上飞翔

通过在半径方向上移动，传动臂可以将读/写头定位在任何磁道上

磁盘操作(多盘片)



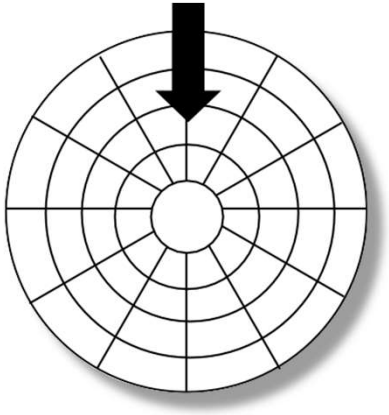
磁盘结构 – 单盘片俯视图



面由若干磁道构成

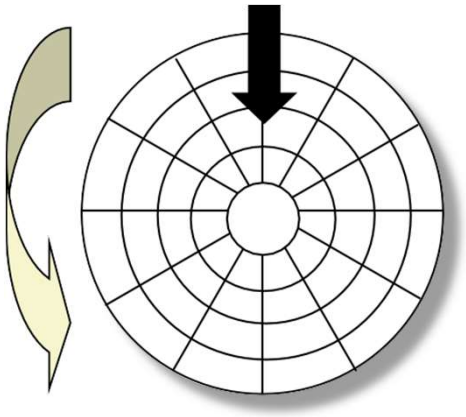
磁道被划分为若干扇区

磁盘访问



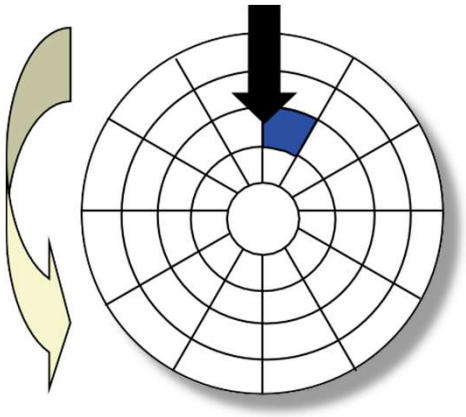
读/写头在磁道上方

磁盘访问



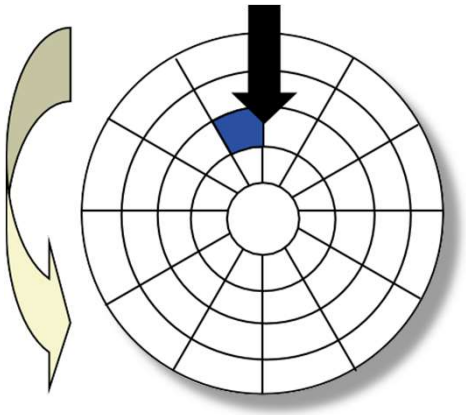
盘面逆时针旋转

磁盘访问 – 读



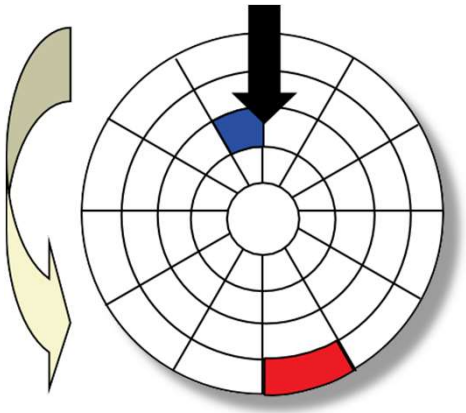
读取蓝色扇区

磁盘访问 – 读



读完蓝色扇区

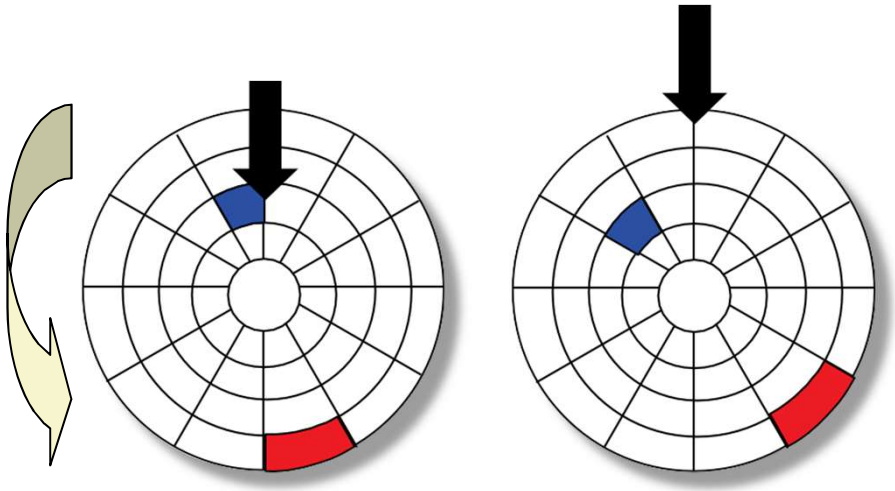
磁盘访问 – 读



读完蓝色扇区

请求读取红色扇区

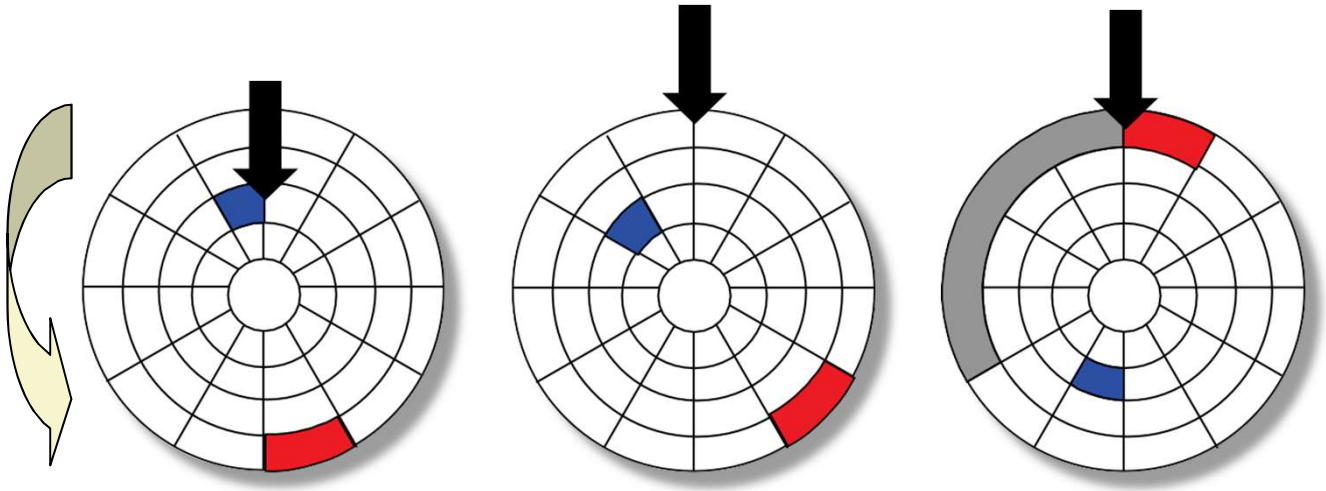
磁盘访问 – 寻道



读完蓝色扇区　　寻找红色扇区

寻找红色扇区所在磁道

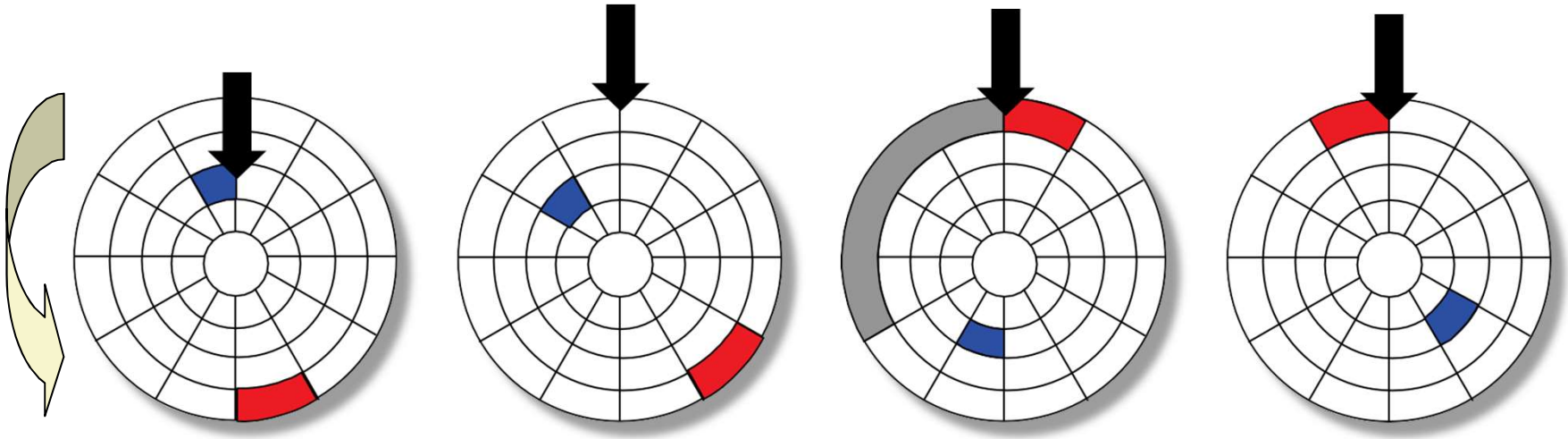
磁盘访问– 旋转延迟



读完蓝色扇区　　寻找红色扇区　　旋转延迟

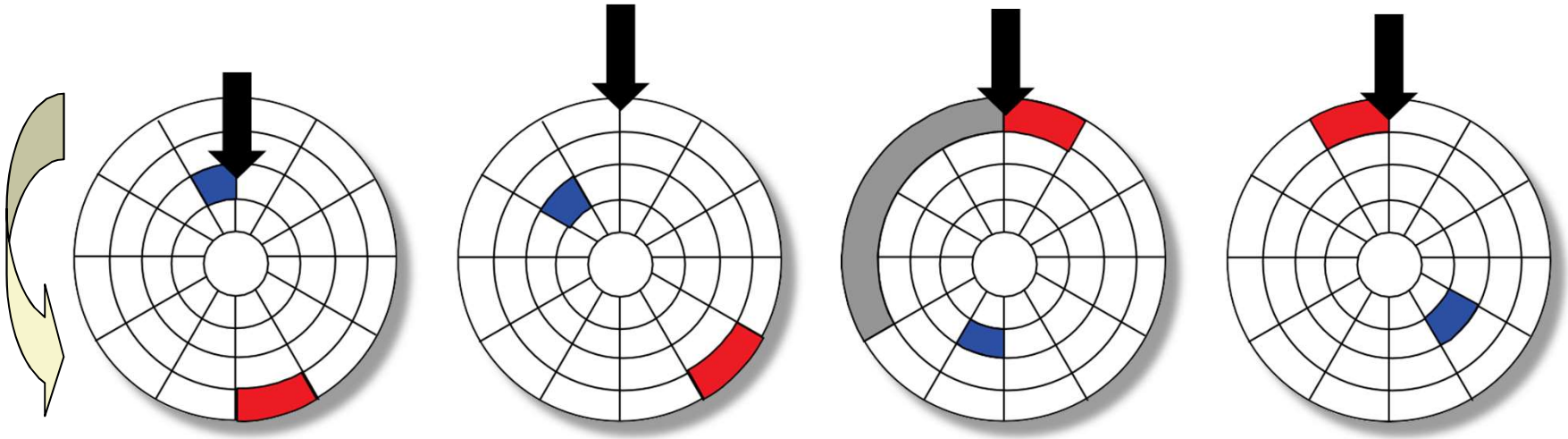
旋转盘面，使读/写头在红色扇区上方

磁盘访问– 读



读取红色扇区

磁盘访问– 访问时间构成



磁盘访问时间

- 访问目标扇区的平均时间大致为:
  - 访问时间 = 寻道时间 + 平均旋转延迟 + 数据传输时间
- **寻道时间(Seek time)**
  - 读/写头移动到目标柱面所用时间
  - 通常寻道时间为：3—9 ms
- **旋转延迟(Rotational latency)**
  - 旋转盘面使读/写头到达目标扇区上方所用时间
  - 平均旋转延迟= 1/2 x 1/RPMs x 60 sec/1 min （RPM：转/分钟）
  - 通常 平均旋转延迟 = 7,200 RPMs
- **数据传输时间(Transfer time)**
  - 读目标扇区所用时间
  - 数据传输时间 = 1/RPM x 1/(平均扇区数/磁道) x 60 secs/1 min

磁盘访问时间 - 举例

- 给定条件:
  - 旋转频率 = 7,200 转/分钟
  - 平均寻道时间 = 9 ms.
  - 平均扇区数/磁道 = 400.
- 计算:
  - 平均旋转延迟= ?
  - 数据传输时间= ?
  - 访问时间= ?

磁盘访问时间

- 访问目标扇区的平均时间大致为:
  - 访问时间 = 寻道时间 + 平均旋转延迟 + 数据传输时间
- 寻道时间(Seek time)
  - 读/写头移动到目标柱面所用时间
  - 通常寻道时间为：3—9 ms
- 旋转延迟(Rotational latency)
  - 旋转盘面使读/写头到达目标扇区上方所用时间
  - 平均旋转延迟= 1/2 x 1/RPMs x 60 sec/1 min （RPM：转/分钟）
  - 通常 平均旋转延迟 = 7,200 RPMs
- 数据传输时间(Transfer time)
  - 读目标扇区所用时间
  - 数据传输时间 = 1/RPM x 1/(平均扇区数/磁道) x 60 secs/1 min

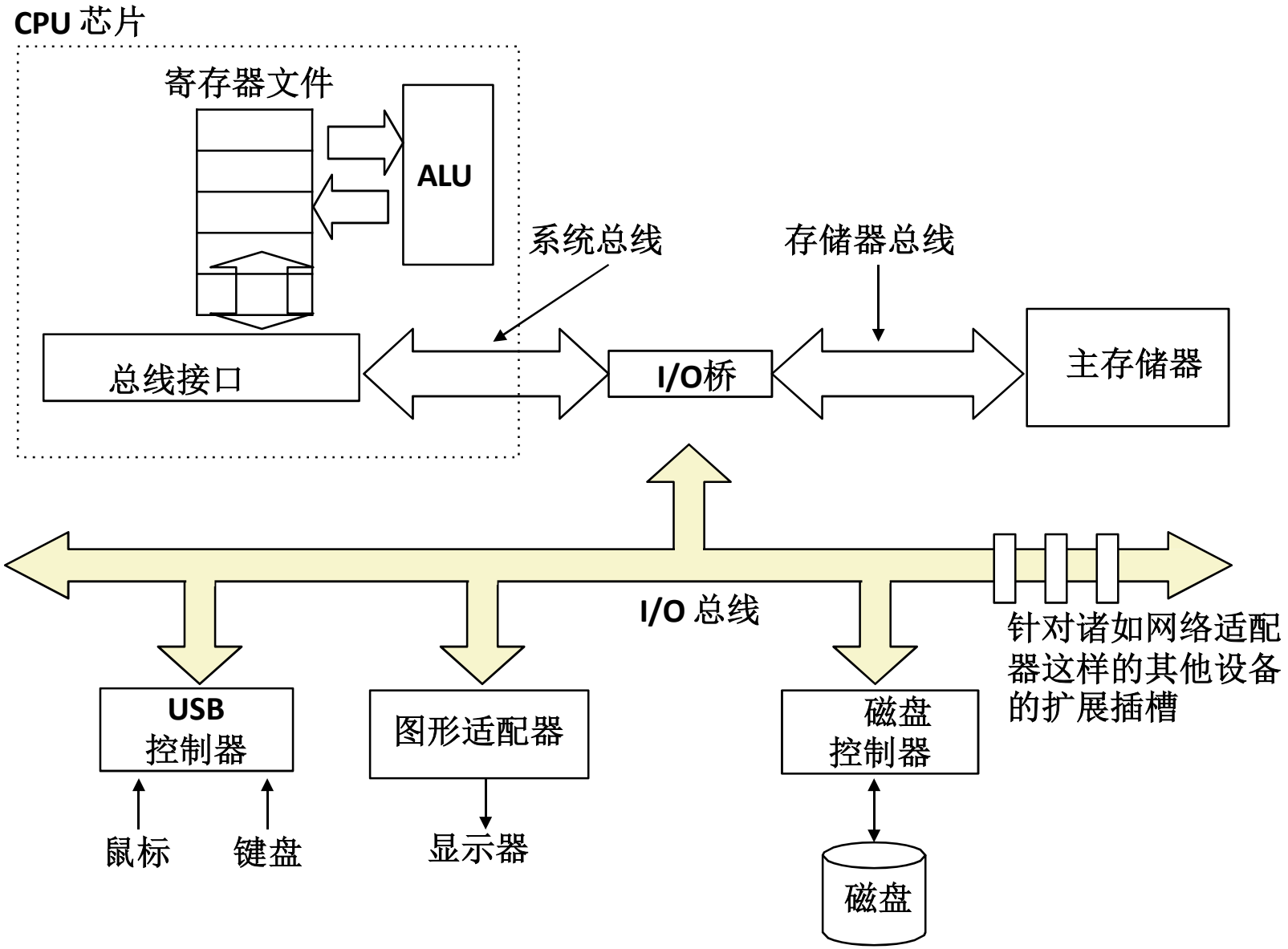
磁盘访问时间 - 举例

- 给定条件:
  - 旋转频率 = 7,200 转/分钟
  - 平均寻道时间 = 9 ms.
  - 平均扇区数/磁道 = 400.
- 计算结果:
  - 平均旋转延迟 = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms.
  - 数据传输时间 = 60/7200 RPM x 1/400 扇区数/磁道 x 1000 ms/sec = 0.02 ms
  - 访问时间 = 9 ms + 4 ms + 0.02 ms
- 重点:
  - 访问时间主要由寻道时间和旋转延迟时间组成。
  - 访问扇区首位花费时间较长，其他位较快。
  - ***SRAM 访问时间大约为4 ns/双字, DRAM 大约为 60ns/双字。***
    - ***磁盘比SRAM慢大约40,000倍, 比DRAM慢大约2,500倍。***

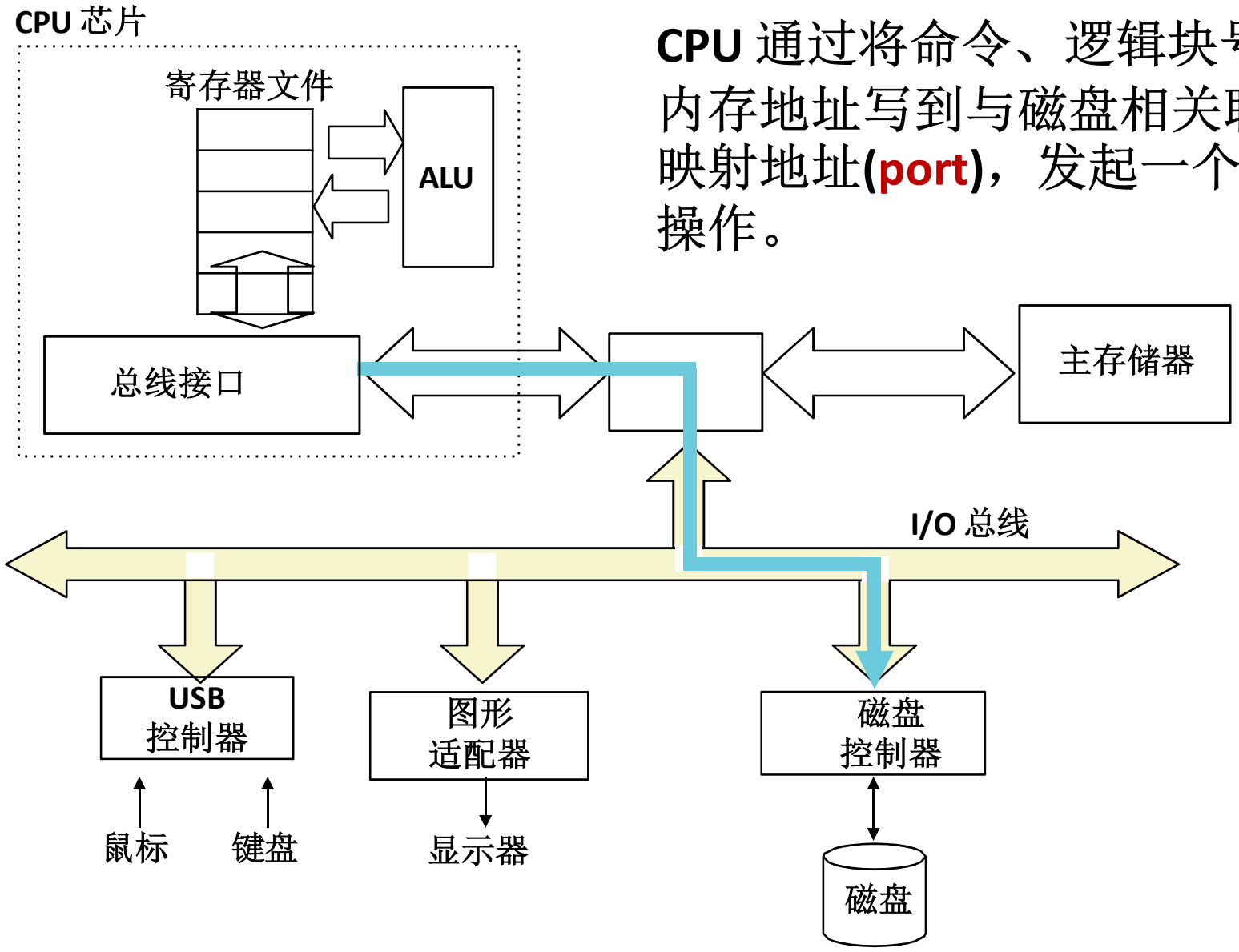
逻辑磁盘块

- 现代磁盘以简单的抽象视图来表示复杂的磁盘构造:
  - 磁盘被抽象成b个扇区大小的逻辑块(logical block)序列 (编号为0, 1, 2, ...)
- 逻辑块与物理扇区之间的映射关系
  - 由磁盘控制器维护
  - 磁盘控制器将逻辑块号转换为一个三元组(盘面,磁道,扇区)
- 允许磁盘控制器为每个分区预留一组柱面作为备份
  - 区分“格式化容量”与“最大容量”

I/O 总线

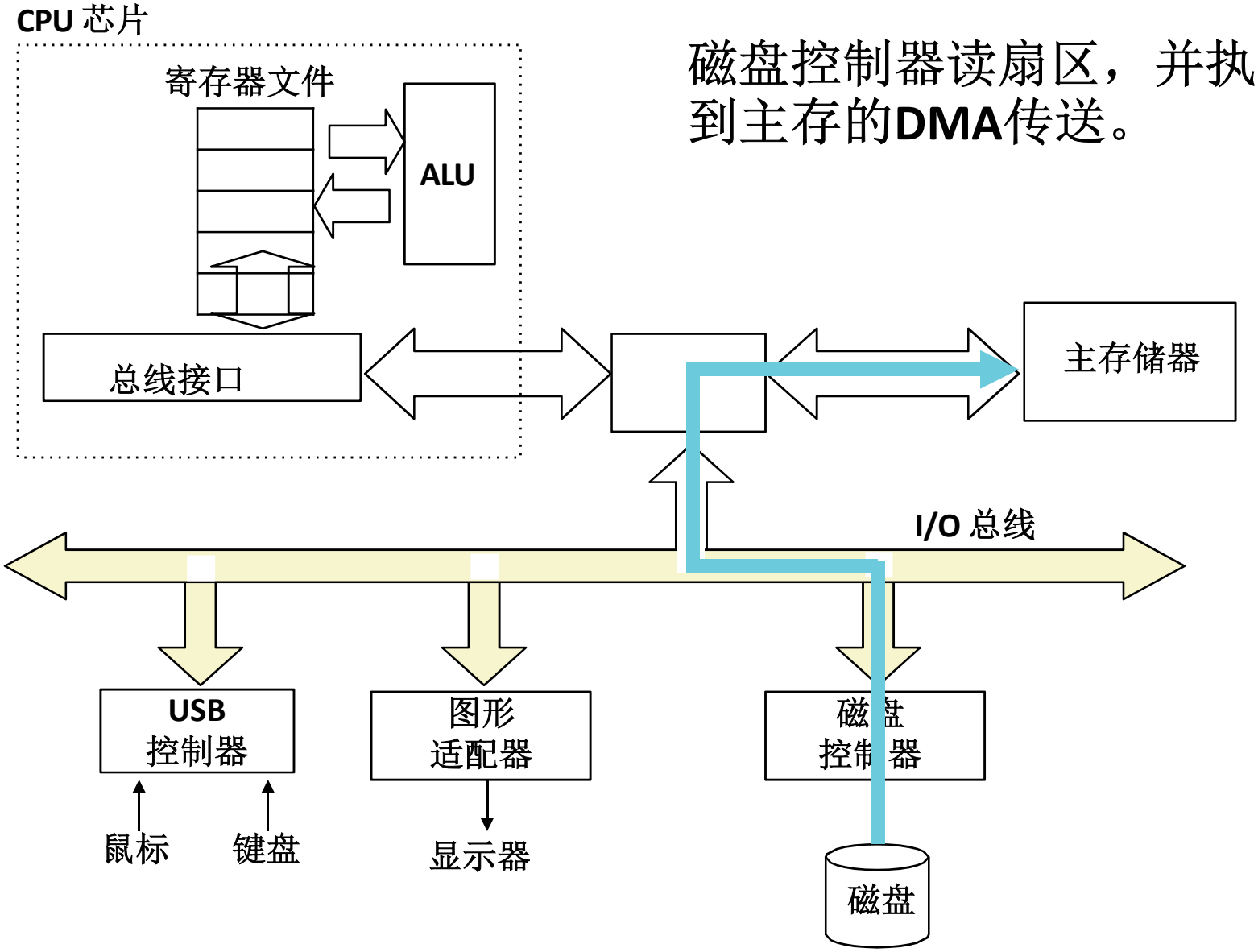

CPU 芯片
寄存器文件
ALU
系统总线
存储器总线
总线接口
I/O桥
主存储器
I/O 总线
针对诸如网络适配器这样的其他设备的扩展插槽
USB
控制器
图形适配器
磁盘
控制器
鼠标
键盘
显示器
磁盘

读磁盘扇区 (1)



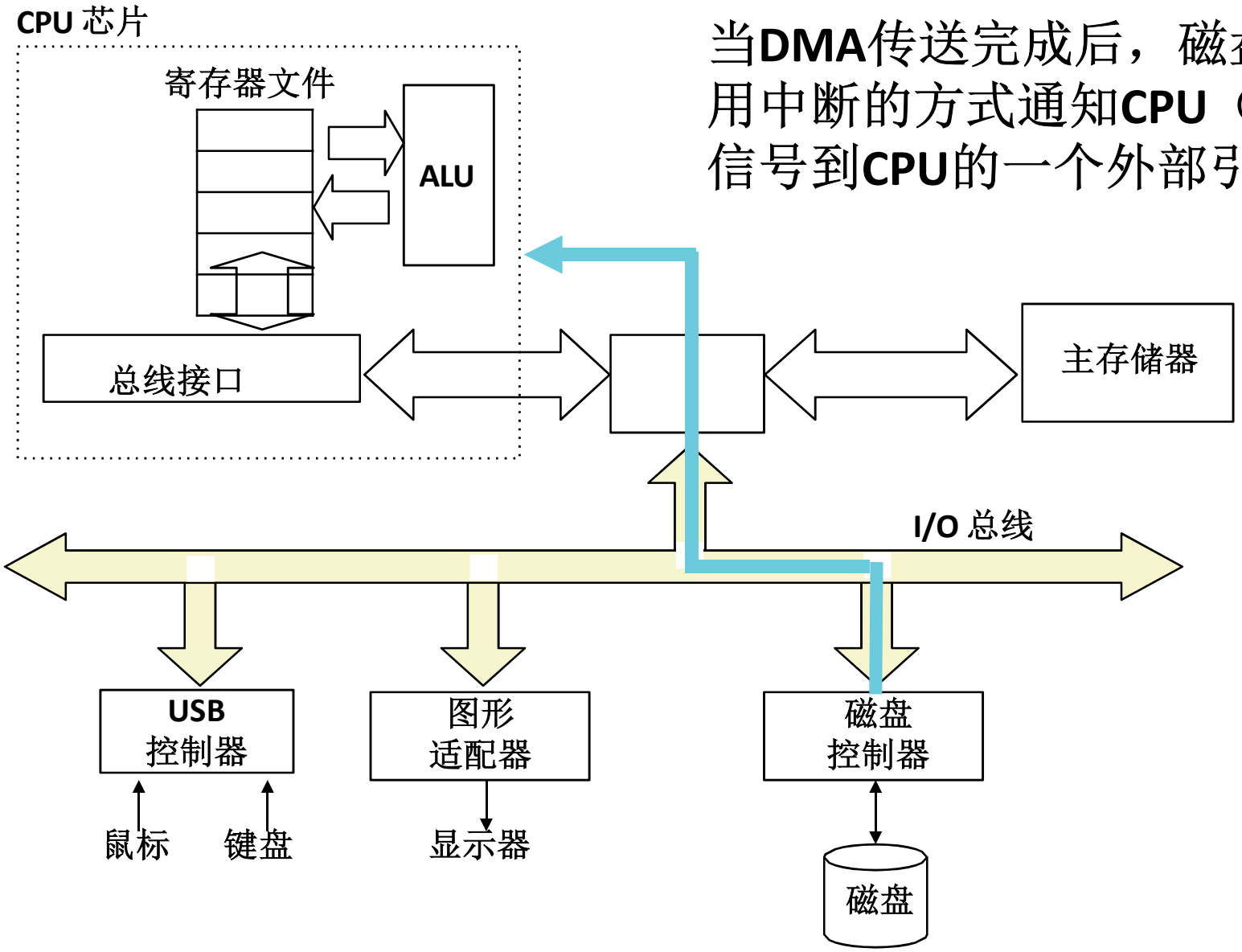
CPU 通过将命令、逻辑块号和目的内存地址写到与磁盘相关联的内存映射地址(**port**)，发起一个磁盘读操作。

读磁盘扇区 (2)



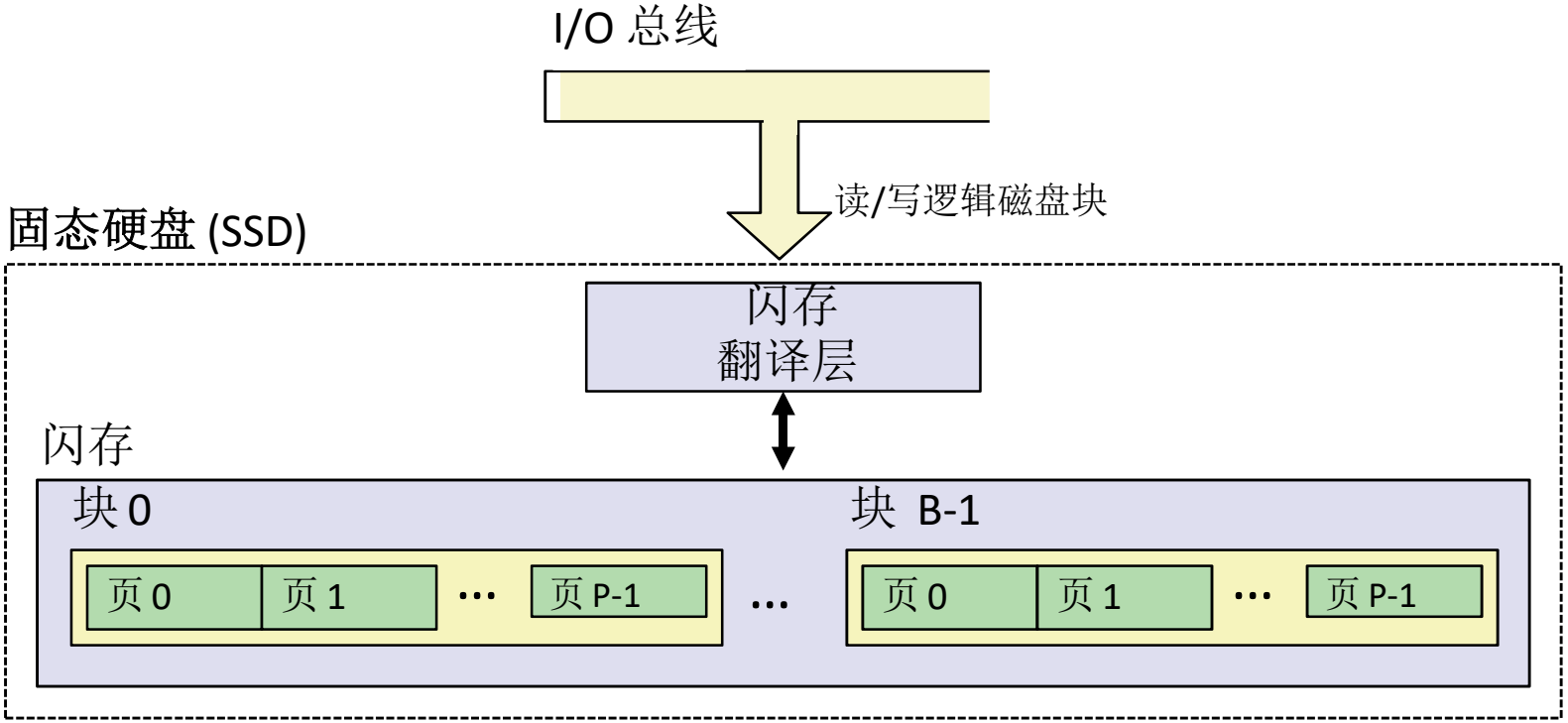
磁盘控制器读扇区，并执行到主存的DMA传送。

读磁盘扇区(3)



当DMA传送完成后，磁盘控制器用中断的方式通知CPU（即发送信号到CPU的一个外部引脚上）。

固态硬盘 (SSDs)



- 页大小： **512B ~ 4KB,** 块大小： **32 ~ 128**页
- 数据以页为单位进行读写
- 只有某页所属块整个被擦除后，才能写该页
- 大约 **100,000** 次重复写之后，块就会磨损坏。

SSD 性能特性

顺序读吞吐量	**550 MB/s**	顺序写吞吐量	**470 MB/s**
随机读吞吐量	**365 MB/s**	随机写吞吐量	**303 MB/s**
平均顺序读访问时间	**50 us**	平均顺序写访问时间	**60 us**

- 顺序访问比随机访问快
  - 典型存储器层次结构问题
- 随机写较慢
  - 擦除块需要较长的时间(~1ms)
  - 修改一页需要将块中所有页复制到新的块中
  - 早期SSD 读/写速度之间的差距更大

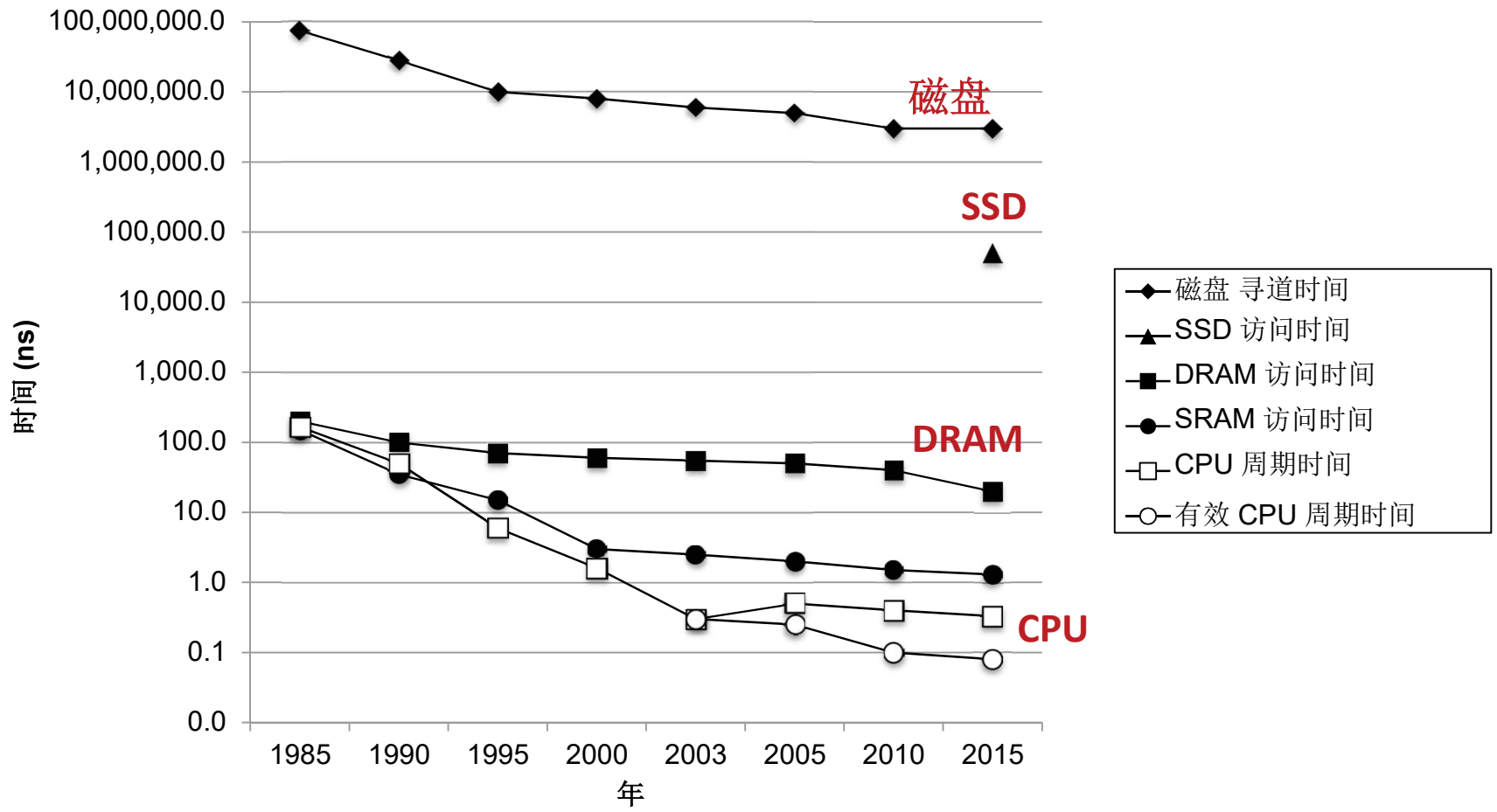
资料来源: **Intel SSD 730** 产品详细说明书

SSD vs 机械磁盘

- 优点
  - 没有移动部件 → 更快、能耗更低、更结实
- 缺点
  - 会磨损
    - 闪存翻译层中的平均磨损逻辑试图通过将擦除平均分布在所有块上来最大化每个块的寿命
    - 比如， Intel SSD 730 保证能经得起 128 PB (128 x 10^{15} 字节)的写
  - 2015年，SSD每字节比机械磁盘贵大约30倍
- 应用
  - MP3播放器、智能手机、笔记本电脑
  - 开始在台式机和服务器中应用

CPU-储存器 之间的差距

DRAM、磁盘和CPU速度之间的差距



用局部性原理(locality)来解决

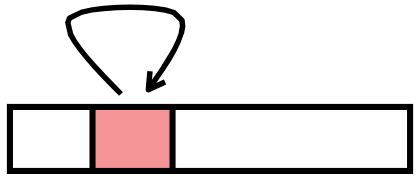
解决CPU-存储器之间速度差距的关键是程序中特有的局部性特点。

- 存储技术与趋势
- 局部性
- 存储器层次结构中的高速缓存

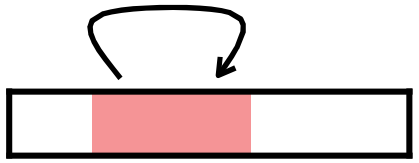
局部性

- **局部性原理(Principle of Locality):** 程序倾向于使用最近一段时间，距离其较近地址的指令和数据。

- **时间局部性(Temporal locality):**
  - 当前被访问的信息近期很可能还会被再次访问



- **空间局部性(Spatial locality):**
  - 在最近的将来将用到的信息很可能与现在正在使用的信息在空间地址上是临近的



局部性举例

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

- 对数据的引用
  - 顺序访问数组元素
    (步长为1的引用模式)
  - 变量**sum**在每次循环迭代中被引用一次
- 对指令的引用
  - 顺序读取指令
  - 重复循环执行for循环体

空间局部性

时间局部性

空间局部性

时间局部性

对局部性的定性评价

- 声明: 能够查看程序代码并对程序局部性有定性的认识，是专业程序员的一项关键技能。

- 问题: 关于数组 a，函数**sum_array_rows**具有良好的局部性吗？

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

局部性举例

- 问题:关于数组a，函数sum_array_cols具有良好的局部性吗？

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

局部性举例

- 问题: 你能改变下面函数中循环的顺序，使得它以步长为 1 的引用模式扫描三维数组 a (从而函数具有良好的局部性)?

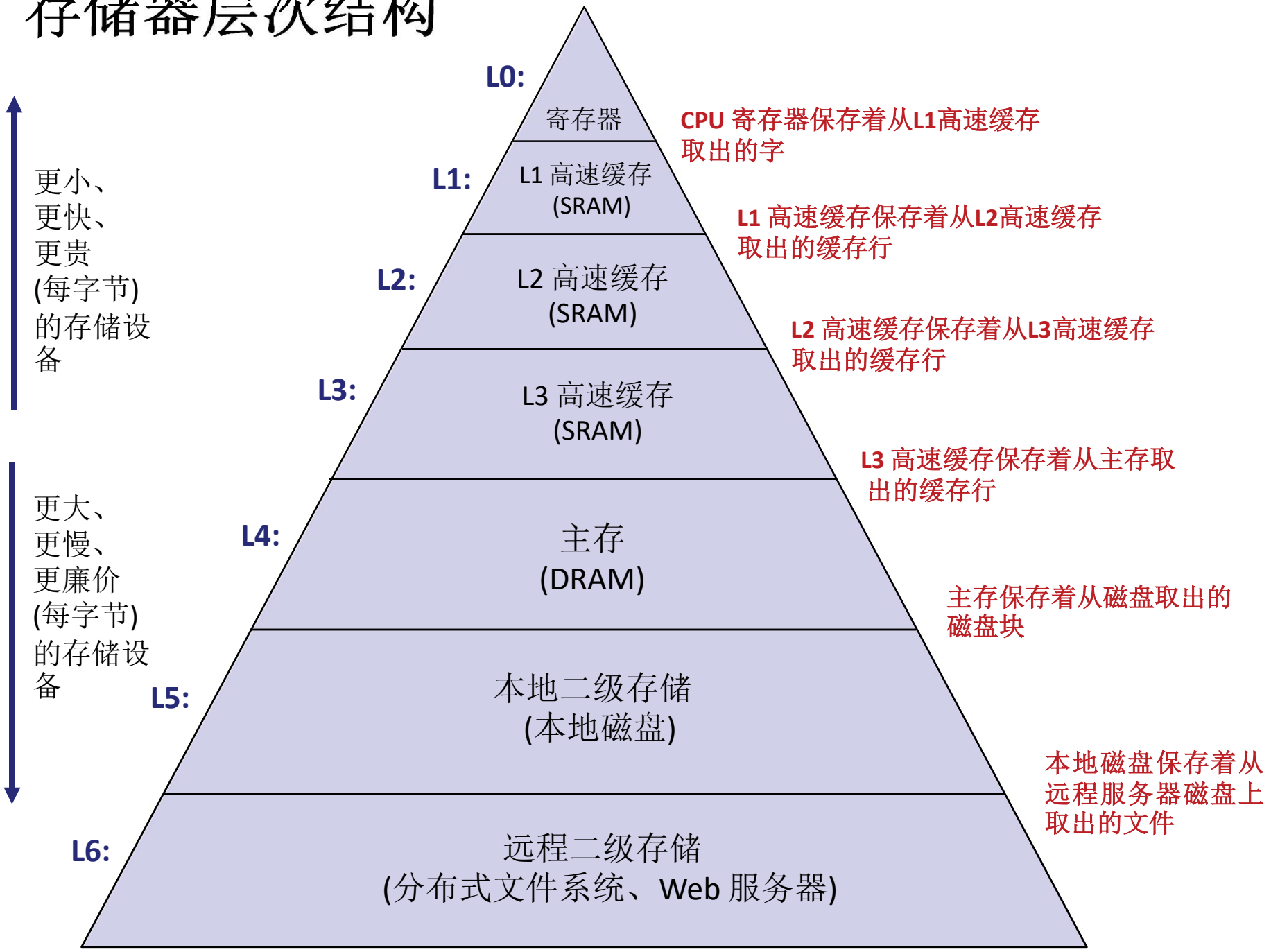
```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < M; k++)
                sum += a[k][i][j];
    return sum;
}
```

存储器层次结构

- 软硬件的基本稳定特性:
  - 高速存储器技术成本高, 容量小，且耗电大，易发热
  - CPU与存储器之间的速度差距越来越大
  - 编写良好的程序往往表现出良好的局部性
- 这些基本特性相互补充
- 以上特性给出一条组织存储器系统的途径 — 存储器层次结构(memory hierarchy).

存储器层次结构


L0:
寄存器
CPU 寄存器保存着从L1高速缓存取出的字
L1:
L1 高速缓存
(SRAM)
L1 高速缓存保存着从L2高速缓存取出的缓存行
L2:
L2 高速缓存
(SRAM)
L2 高速缓存保存着从L3高速缓存取出的缓存行
L3:
L3 高速缓存
(SRAM)
L3 高速缓存保存着从主存取出的缓存行
L4:
主存
(DRAM)
主存保存着从磁盘取出的磁盘块
L5:
本地二级存储
(本地磁盘)
本地磁盘保存着从远程服务器磁盘上取出的文件
L6:
远程二级存储
(分布式文件系统、Web 服务器)
更小、更快、更贵(每字节)的存储设备
更大、更慢、更廉价(每字节)的存储设备

存储器层次结构中的缓存

缓存类型	缓存什么	被缓存在何处	延迟(周期数)	由谁管理
寄存器	4-8 字节字	CPU 核心	0	编译器
TLB	地址译码	片上 TLB	0	硬件 MMU
L1 高速缓存	64字节块	片上 L1	4	硬件
L2 高速缓存	64字节块	片上 L2	10	硬件
虚拟内存	4KB 页	主存	100	硬件 + OS
缓冲区缓存	部分文件	主存	100	OS
磁盘缓存	磁盘扇区	磁盘控制器	100,000	磁盘固件
网络缓冲区缓存	部分文件	本地磁盘	10,000,000	NFS 客户
浏览器缓存	Web页	本地磁盘	10,000,000	Web浏览器
Web缓存	Web 页	远程服务器磁盘	1,000,000,000	Web 代理服务器

总结

- **CPU**、主存、大容量存储设备之间的速度差距持续扩大
- 编写良好的程序表现出良好的局部性
- 利用局部性特点，基于高速缓存的存储器层次结构有利于缩小速度差距