第四章 存储器层次结构

- 存储器概述
- 主存储器
- 外部存储器
- 高速缓冲存储器(cache)

层次结构存储系统

- 学习目标
  - 理解CPU执行指令过程中为何要访存
  - 理解访存操作的大致过程及涉及到的部件
  - 了解层次化存储器系统的由来及构成
  - 了解CPU与主存储器之间的连接及读写操作
  - 掌握Cache机制并理解其对程序性能的影响
  - 理解程序局部性的重要性并能开发局部性好的程序

层次结构存储系统

- 第一讲：存储器概述
- 第二讲：主存储器
  - 基本存储元件、DRAM芯片、 SRAM芯片技术
  - 内存条及其与CPU的连接
  - 存储器芯片的扩展、主存控制器
- 第三讲：外部存储器
  - 磁盘存储器、固态硬盘
- 第四讲：高速缓冲存储器（cache）
  - Cache的基本工作原理
  - 映射方式、替换算法、写策略
  - Cache的设计
  - Cache和程序性能

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）

```
0000000000400540 <multstore>:
  400540: push   %rbx              # Save %rbx
  400541: mov    %rdx,%rbx         # Save dest
  400544: call   400550 <mult2>    # mult2(x,y)
  400549: mov    %rax,(%rbx)       # Save at dest
  40054c: pop    %rbx              # Restore %rbx
  40054d: ret                      # Return
```



```
long mutl2(long, long);
void multistore(long x, long y,
                long *dest) {
  long t = mult2(x, y);
  *dest = t;
}
// x:rdi, y:rsi, *dest:rdx
```

- 对于multstore函数的执行，何时需要访存？
  - 每条指令都需要从主存单元取到CPU执行　取指
  - push指令需要把寄存器内容压入栈中　取数
  - call指令需要将返回地址入栈　存数
  - mov指令需要从寄存器中取数送到主存　存数
  - pop指令需要从栈中取内容送入寄存器　取数
  - ret指令需要从栈中取出返回地址　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

存储器分类

依据不同的特性有多种分类方法

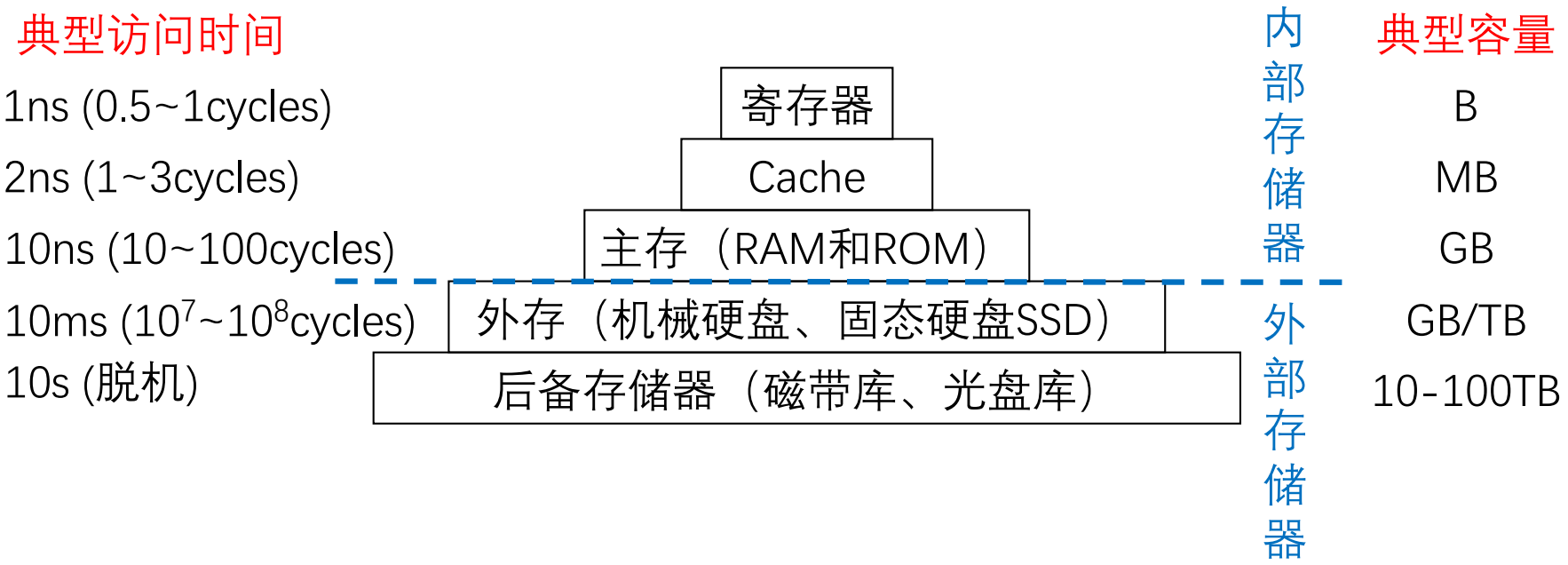
- 按工作性质/存取方式分类
  - 随机存取存储器 Random Access Memory (RAM)
    - 访问任一存储单元所需时间与单元的物理位置无关。
    - 例如：内存（易失性：SRAM/DRAM，非易失性：ROM）
  - 顺序存取存储器 Sequential Access Memory (SAM)
    - 数据按顺序从存储载体的始端读出或写入，因而存取时间的长短与信息所在位置有关。
    - 例如：磁带
  - 直接存取存储器 Direct Access Memory (DAM)
    - 直接定位到读写数据块，在读写数据块时按顺序进行。
    - 例如：磁盘
  - 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

依据不同的特性有多种分类方法

- 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~几百b）
  - 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（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（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
    - 用磁盘、SSD等实现，容量大而速度慢

存储器的层次结构



各类存储的时间和容量随硬件发展变化，但数量级相对关系相似。

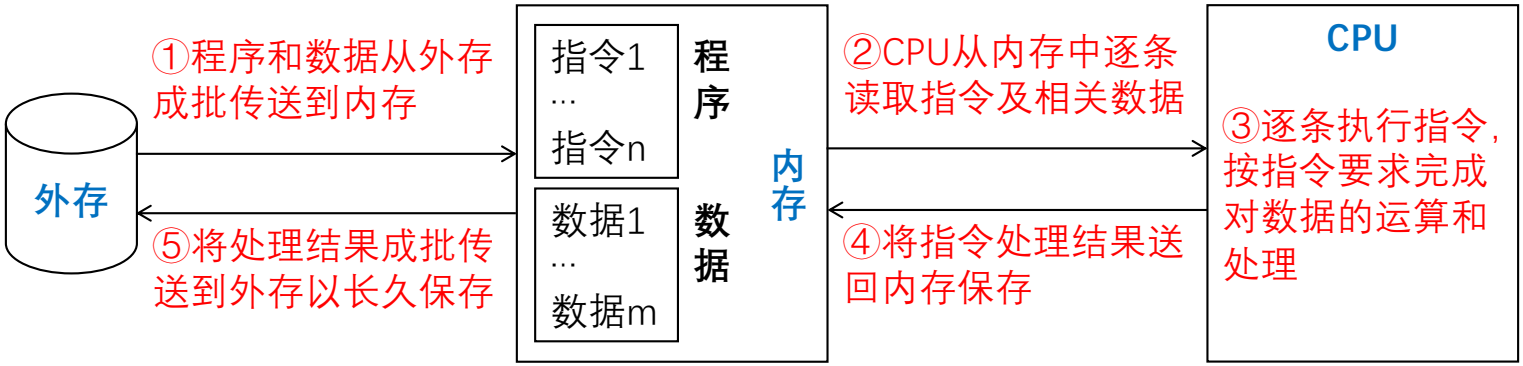
内存与外存的关系及比较

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（volatile）存储器，用于临时存放正在运行的程序和数据



时间、存储容量（或带宽）的单位

前缀	缩写	含义	数值
mill	m	one thousandth	10^{-3}
micro	u	one millionth	10^{-6}
nano	n	one billionth	10^{-9}
pico	p	one trillionth	10^{-12}
femto	f	one quadrillionth	10^{-15}
atta	a	one quintillionth	10^{-18}
kilo	K (k)	thousand	10^{3} or 2^{10}
mega	M	million	10^{6} or 2^{20}
giga	G	billion	10^{9} or 2^{30}
tera	T	trillion	10^{12} or 2^{40}
peta	P	quadrillion	10^{15} or 2^{50}
exa	E	quintillion	10^{18} or 2^{60}

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）

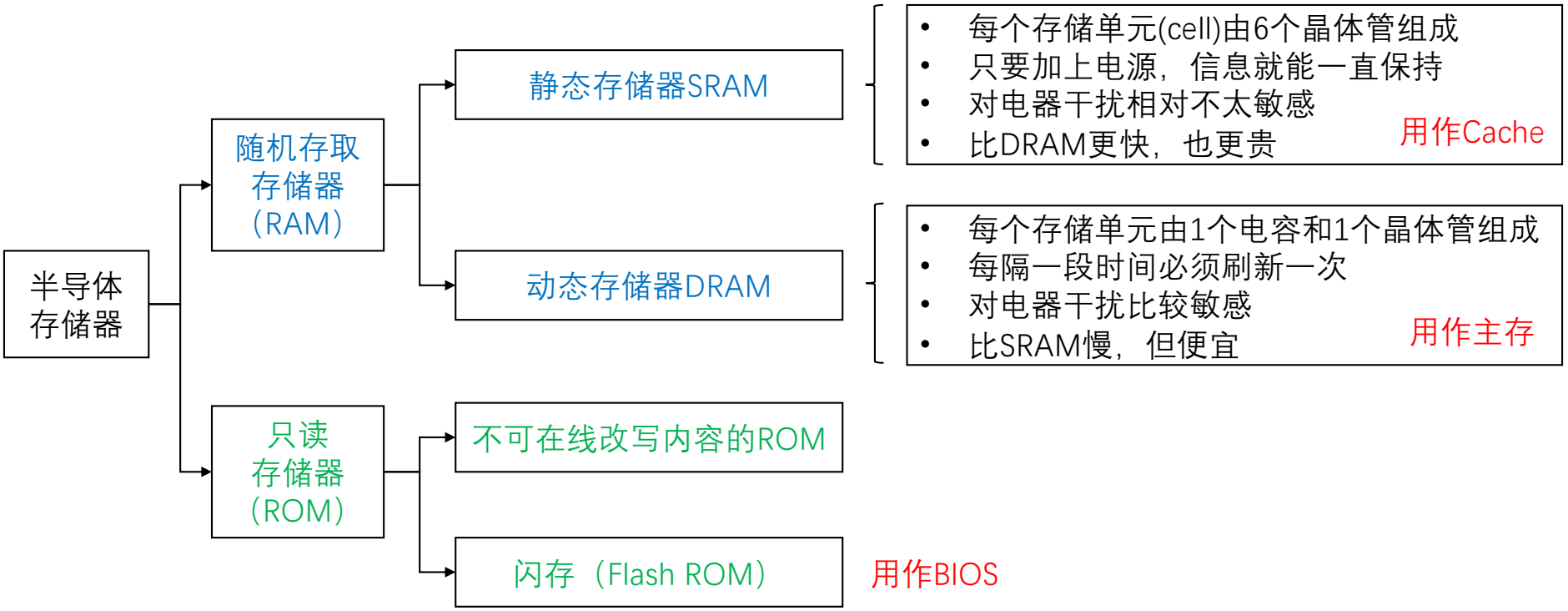
- 存储容量
  - 所包含的存储单元的总数（单位：MB或GB）
- 存取时间T_A
  - 从CPU送出内存单元的地址码开始，到主存读出数据并送到CPU（或者是把CPU数据写入主存）所需要的时间（单位：ns），分读取时间和写入时间
- 存储周期T_{MC}
  - 连读两次访问存储器所需的最小时间间隔，它应等于存取时间加上下一次存取开始前所要求的附加时间，因此，T_{MC}比T_A大（因为存储器由于读出放大器、驱动电路等都有一段稳定恢复时间，所以读出后不能立即进行下一次访问。类比货车的运货时间和发车周期是两个不同的概念。）

层次结构存储系统

- 第一讲：存储器概述
- 第二讲：主存储器
  - 基本存储元件、DRAM芯片、 SRAM芯片技术
  - 内存条及其与CPU的连接
  - 存储器芯片的扩展、主存控制器
- 第三讲：外部存储器
  - 磁盘存储器、固态硬盘
- 第四讲：高速缓冲存储器（cache）
  - Cache的基本工作原理
  - 映射方式、替换算法、写策略
  - Cache的设计
  - Cache和程序性能

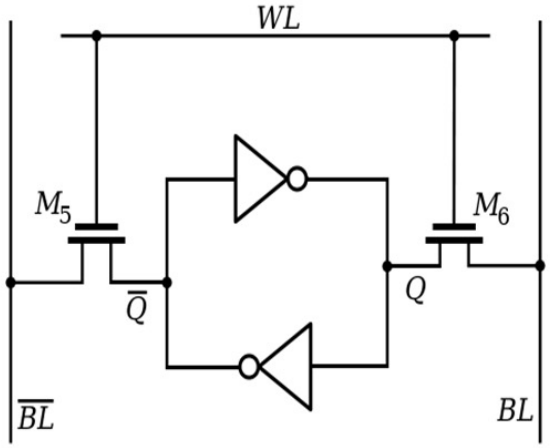
内部存储器的分类及应用

- 内部存储器由半导体存储器芯片组成，芯片有多种类型：



半导体RAM的组织

- 存储体 (Memory Bank)： 由记忆单元(位元)构成的存储阵列
- 记忆单元(Cell)的组织

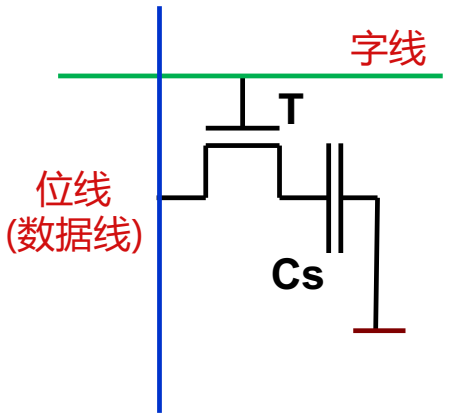


通电持续保持值

对电压扰动不敏感

SRAM

六管静态MOS管电路



漏电导致几十ms失去电荷（相对时钟周期ns足够长，周期性地读出、重写、刷新）

对干扰非常敏感

DRAM

单管动态MOS管电路

	每位晶体管数	相对访问时间	持续的?	敏感的?	相对花费	应用
SRAM	6	1×	是	否	1000×	高速缓存存储器
DRAM	1	10×	否	是	1×	主存，帧缓冲区

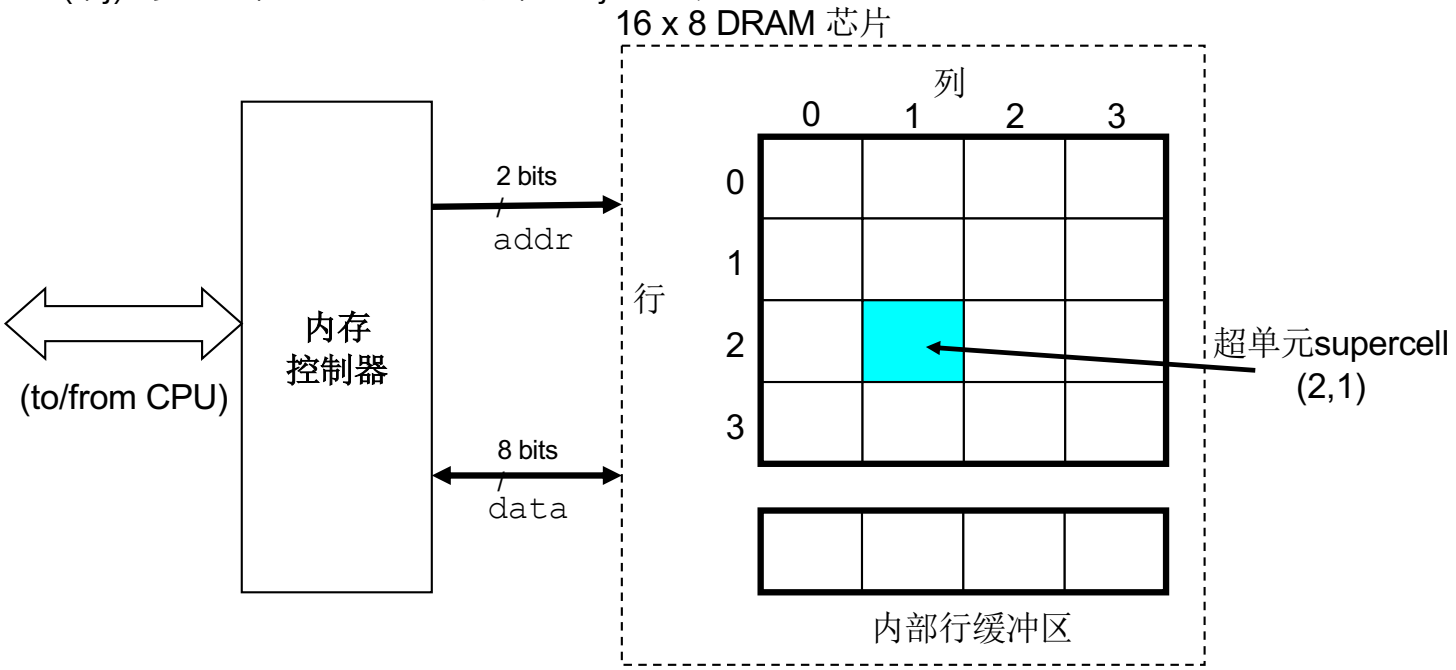
存储器芯片内部结构

- 地址译码方式
  - 一维（线性数组）：用于小容量的静态存储器
  - 二维（阵列）：用于容量较大的动态存储器

- $d \times w$ DRAM
  - DRAM 芯片中由d个存储单元（超单元supercell），每个超单元都由w个位组成
  - 一个$d \times w$的DRAM总共存储了dw位信息。
  - 存储矩阵/矩阵是存储单元的集合

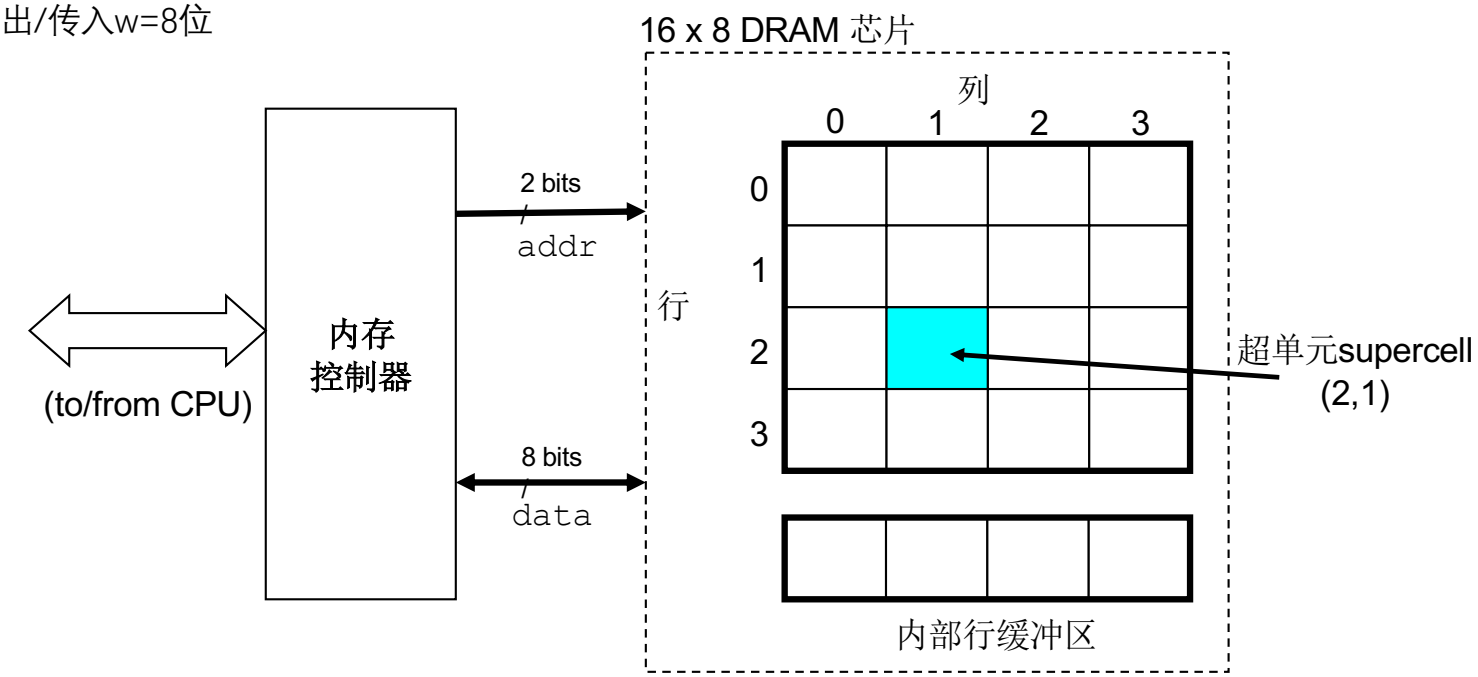
存储器芯片内部结构

- 例：16×8 DRAM芯片
  - 16个超单元，每个超单元8位
  - 二维，在X方向和Y方向进行译码（双译码）
  - 超单元被组织成一个r行c列的长方形阵列，rc=d
  - 每个超单元有形如(i, j)的地址，这里i表示行，而j表示列



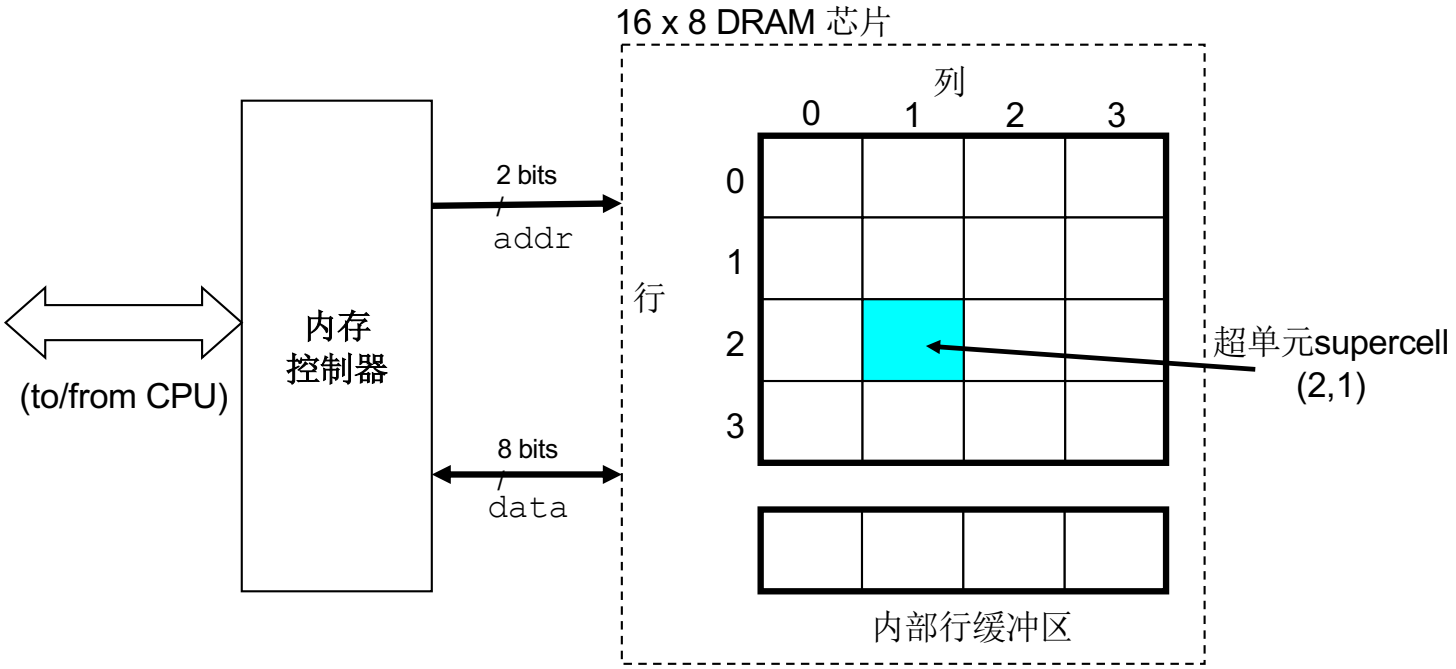
存储器芯片内部结构

- 例：16×8 DRAM芯片
  - 内存控制器，也称DRAM控制器或存储器控制器
    - 一侧连接系统总线，接收来自CPU的访存请求
      - 行访问选通脉冲（RAS=i=2）请求和列访问选通脉冲（CAS=j=1）请求复用地址引脚
    - 另一侧连接存储器总线，向主存芯片发送命令进行读写和刷新
      - 一次传出/传入w=8位



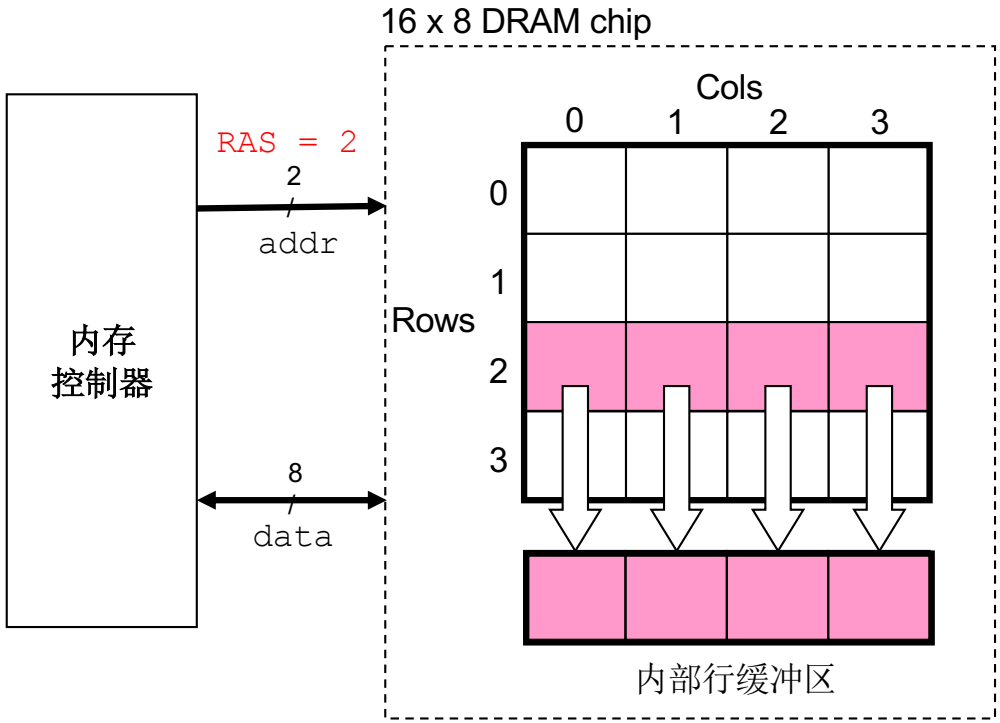
存储器芯片内部结构

- 例：16×8 DRAM芯片
  - 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，2根地址引脚
    - 每个超单元（supercell）有8位，一次传入传出8位，需8根数据引脚
    - 一个内部的行缓冲（row buffer），通常用SRAM元件实现



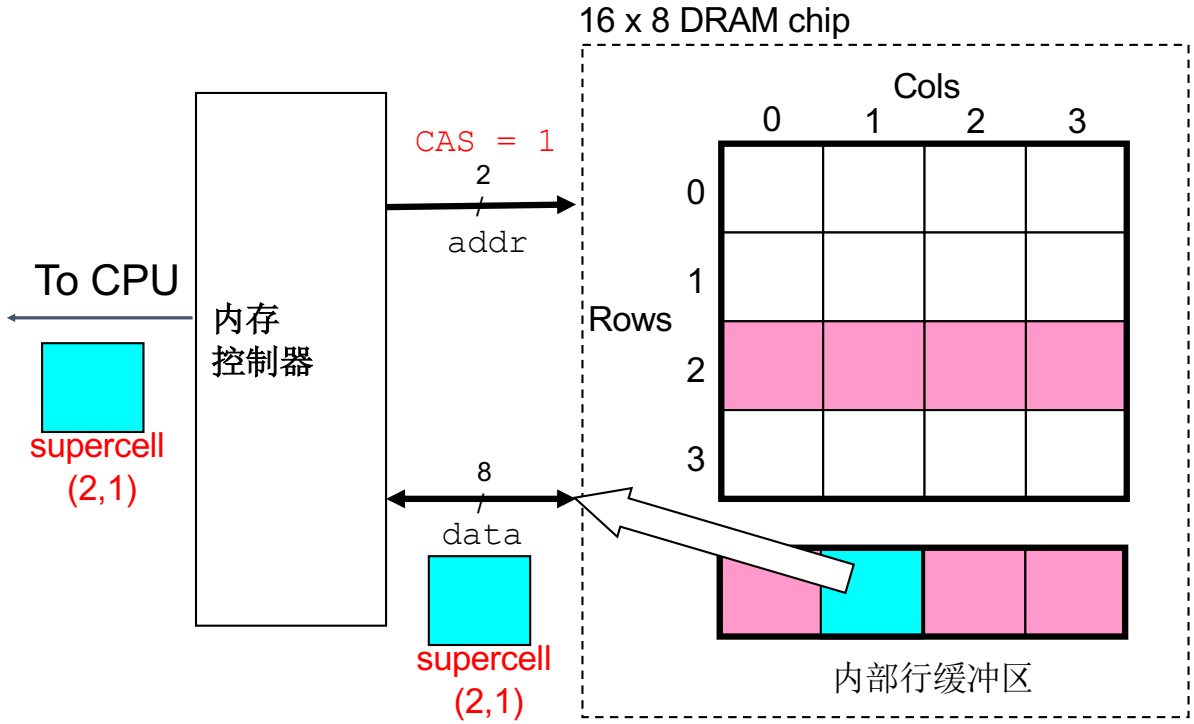
DRAM读写操作

- 读16×8 DRAM supercell (2,1)
  1. 内存控制器通过RAS请求，选择行2，行2整个一行数据被复制到内部行缓冲区



DRAM读写操作

- 读16×8 DRAM supercell (2,1)
  2. 内存控制器发送CAS请求，列地址为1，从内部行缓冲区复制出超单元(2, 1)的8位，发送到内存控制器，继而送到CPU



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$2^n = r \times c$。
  - 例如：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$\log_2 c$。
  - 例如：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（复用只改变引脚数）
- 为提高DRAM芯片的性价比，通常设置的r和c满足$r \leqslant c$且$|r-c|$最小。
  - 按行刷新，行数越少，刷新次数越少；行、列地址位数越接近，地址引脚越少
  - 例如：对于8K×8位DRAM芯片，其存储阵列设置为2^6行×2^7列，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$A_{12}A_{11}\cdots A_1A_0$中，行地址为$A_{12}A_{11}\cdots A_7$，列地址为$A_6\cdots A_1A_0$。

例：设r表示一个DRAM阵列中的行数，c表示列数，b_r表示行寻址所需的位数，b_c表示列寻址所需的位数。对于下面每个DRAM，确定2的幂数阵列维数，使得$\max(b_r, b_c)$最小。

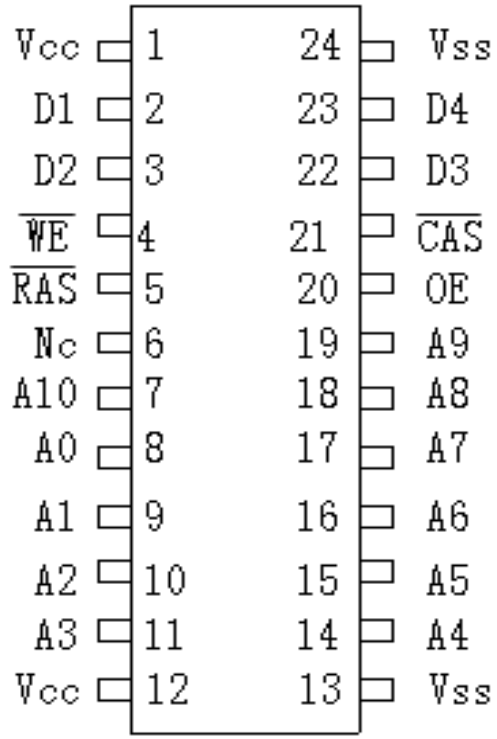
$d \times w$	r	c	b_r	b_c	$\max(b_r, b_c)$
16×4	4	4	2	2	2
128×8					
512×2					
256×32					

例：设r表示一个DRAM阵列中的行数，c表示列数，b_r表示行寻址所需的位数，b_c表示列寻址所需的位数。对于下面每个DRAM，确定2的幂数阵列维数，使得$\max(b_r, b_c)$最小。

$d \times w$	r	c	b_r	b_c	$\max(b_r, b_c)$
16×4	4	4	2	2	2
128×8	8	16	3	4	4
512×2	16	32	4	5	5
256×32	16	16	4	4	4

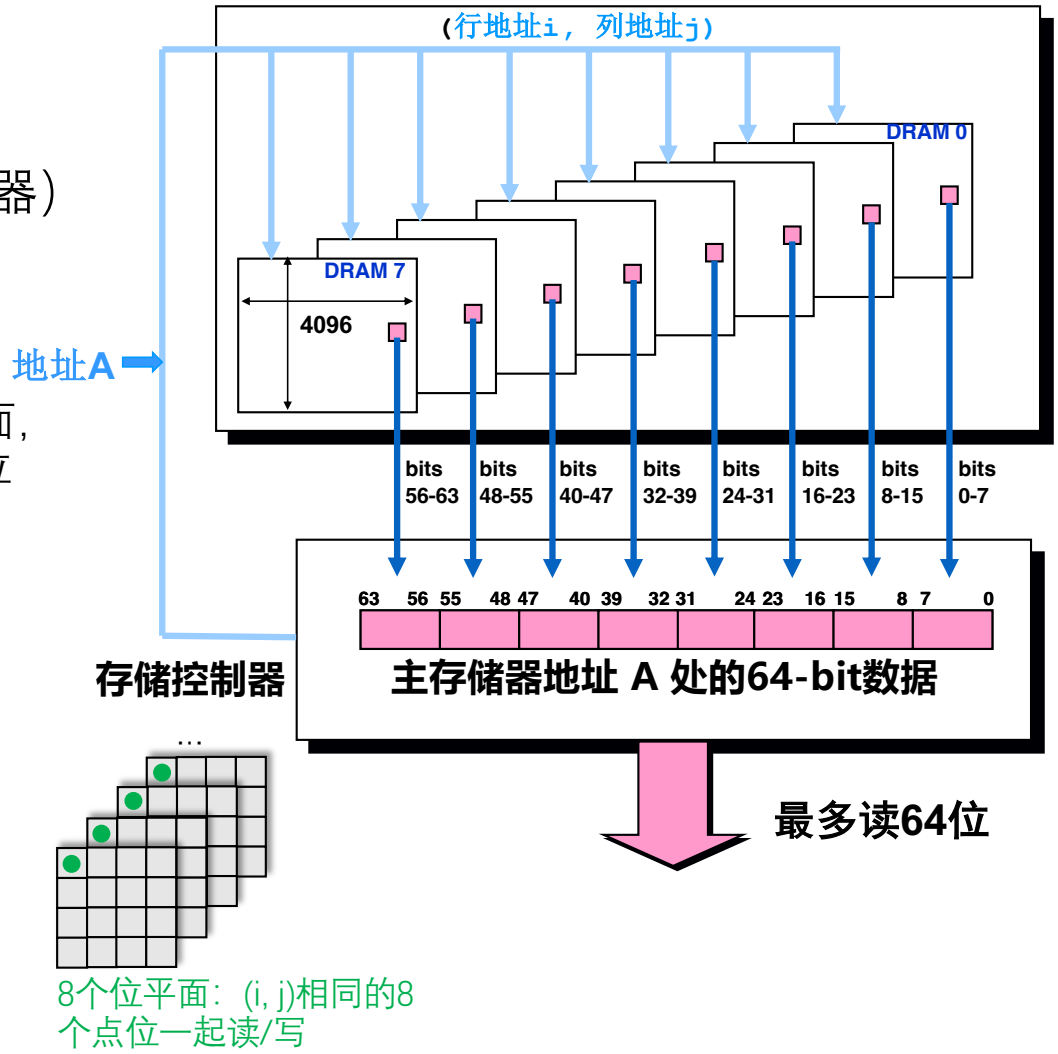
例：16M位（4Mx4） DRAM芯片

- 16M位 = 4Mx4b = 2048x2048x4 = 2^{11}x2^{11}x4
  - 地址引脚：11根线分时复用，由RAS和CAS提供控制时序
  - 4位数据位



例：128MB的DRAM存储器

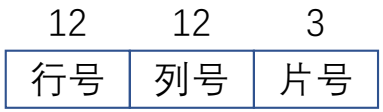
- DRAM芯片封装→内存模块（存储器）
  - 8片DRAM芯片封装
  - 每片16M*8位
  - 行列地址各12位（4096*4096）
  - 一个超单元存储8位1字节，8个位平面，8个(i,j)相同的超单元主存地址A处64位数据



例：128MB的DRAM存储器

- DRAM芯片封装→内存模块（存储器）
  - 8片DRAM芯片封装
  - 每片16M*8位
  - 行列地址各12位（4096*4096）
  - 一个超单元存储8位1字节，8个位平面，8个(i,j)相同的超单元主存地址A处64位数据

- 主存地址和片内地址的关系
  - 芯片内地址24位
    - 行12位，列12位
  - 主存地址A 24+3位
    - 3位用来选片，确定8字节中的位置
  - 假设第k行首地址为i（i=4096*8*k），则地址分布如下：



	DRAM0	DRAM1	…	DRAM7
第0列	i	i+1		i+7
第1列	i+8	i+9		i+15
…				
第4095列	i+8*4095	i+8*4095+1		i+8*4095+7

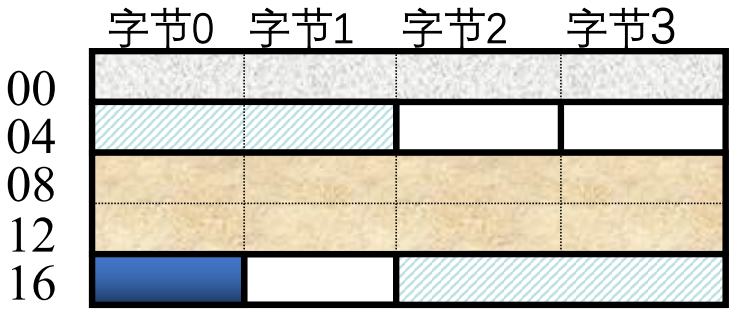
片内地址不连续
交叉编址，可同时读/写所有芯片

对齐（alignment）

例：int i, short k, double x, char c, short j　　假设主存由4片DRAM芯片封装

按边界对齐

- 按字节编址，起始地址是字节大小的整数倍

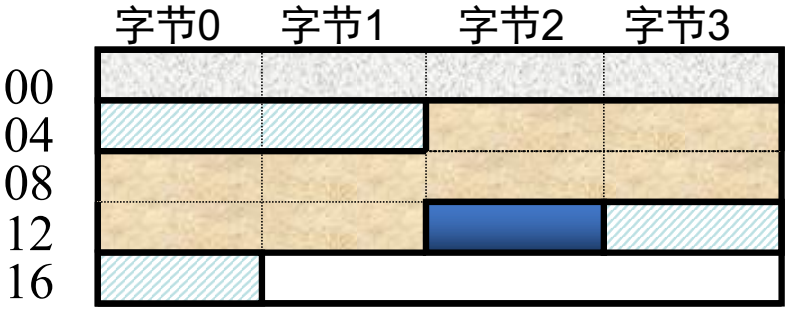


&i=0; &k=4; &x=8; &c=16; &j=18

x：2次访存，j：访存1次

边界不对齐

- 节约空间，增加访存次数



&i=0; &k=4; &x=6; &c=14; &j=15

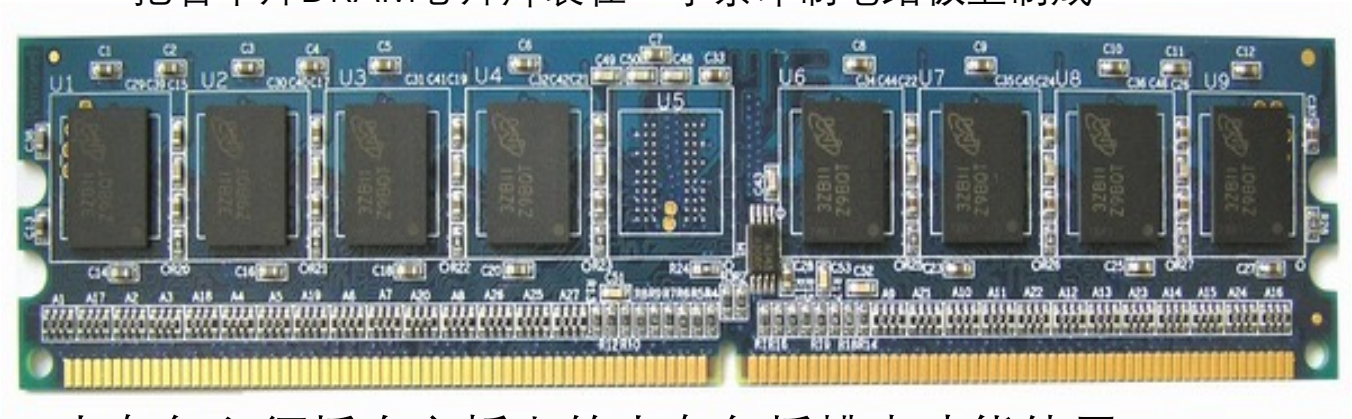
x：3次访存，j：访存2次

增强DRAM

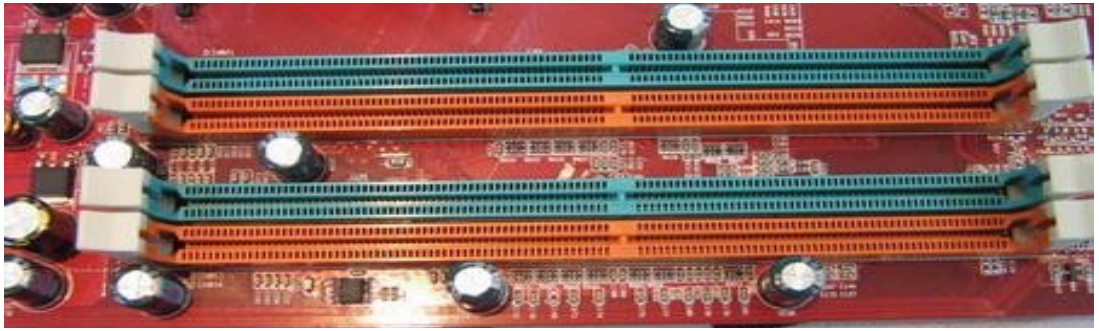
- Basic DRAM cell has not changed since its invention in 1966.
  - Commercialized by Intel in 1970.
- DRAM cores with better interface logic and faster I/O :
  - 同步DRAM (SDRAM)
    - Uses a conventional clock signal instead of asynchronous control
    - Allows reuse of the row addresses
  - 双倍数据速率同步DRAM (DDR SDRAM)
    - Double edge clocking sends two bits per cycle per pin
    - Different types distinguished by size of small prefetch buffer:
      - DDR (2 bits), DDR2 (4 bits), DDR3 (8 bits)
    - By 2010, standard for most server and desktop systems
      - Intel Core i7 supports only DDR3 SDRAM

PC机主存的物理结构

- 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

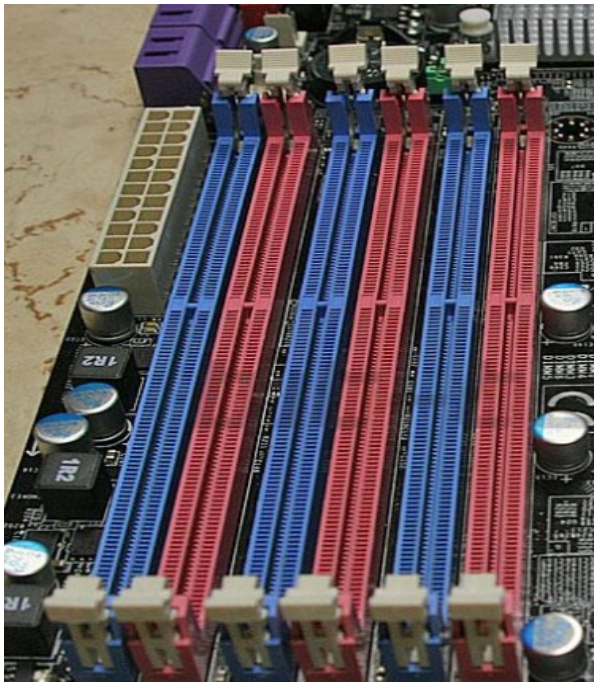


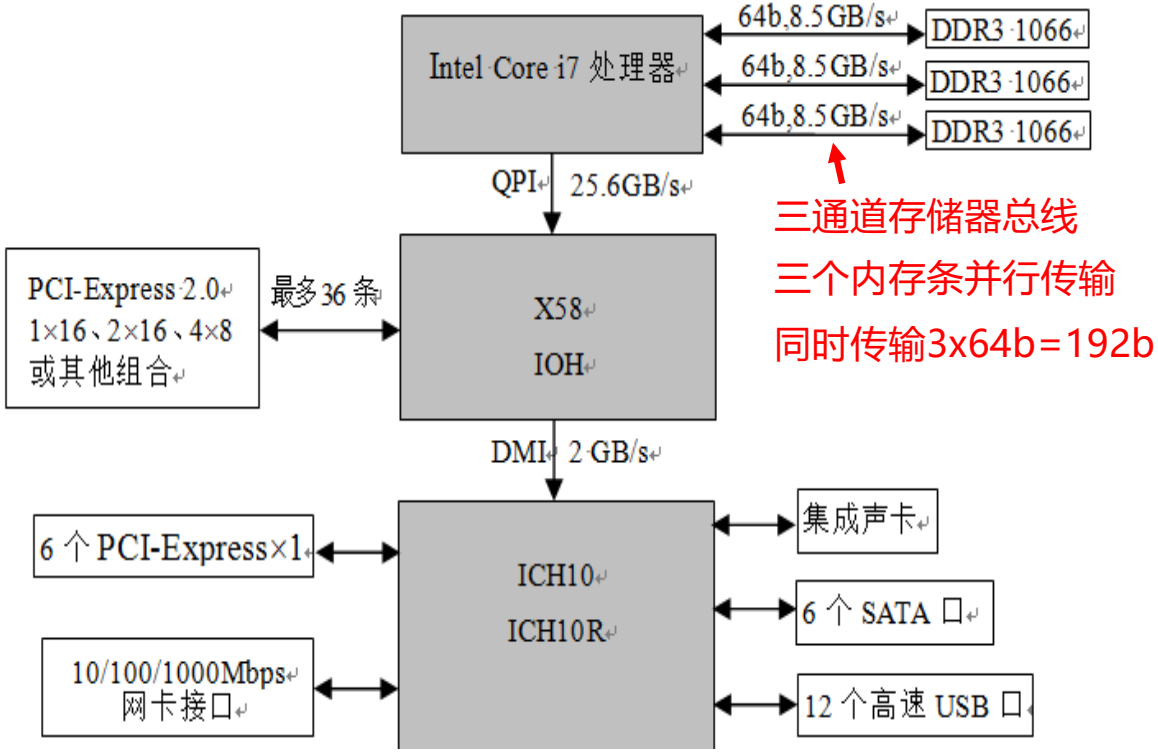
- 内存条必须插在主板上的内存条插槽中才能使用



计算机系统互连

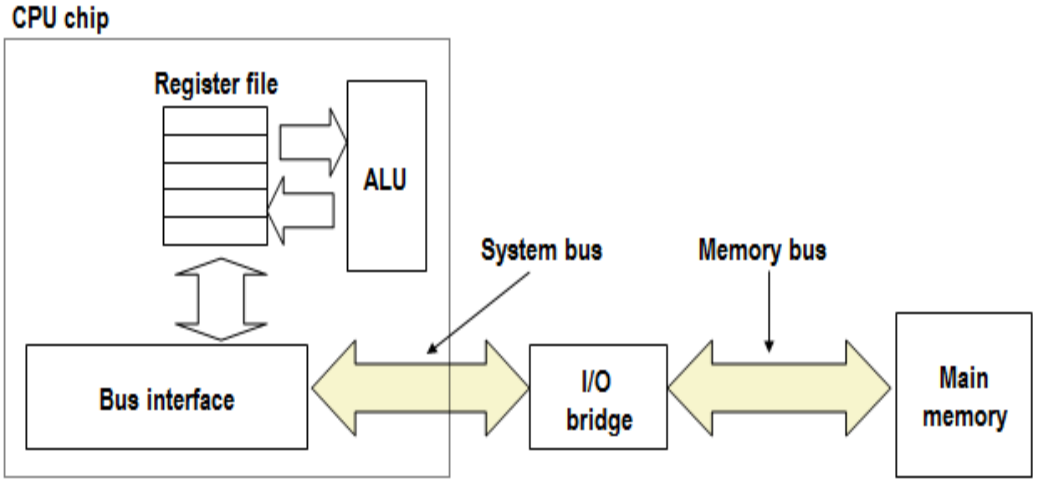
- 将同色的三个内存插槽插上内存条，系统便会自动识别进入三通道模式





主存模块的连接和读写

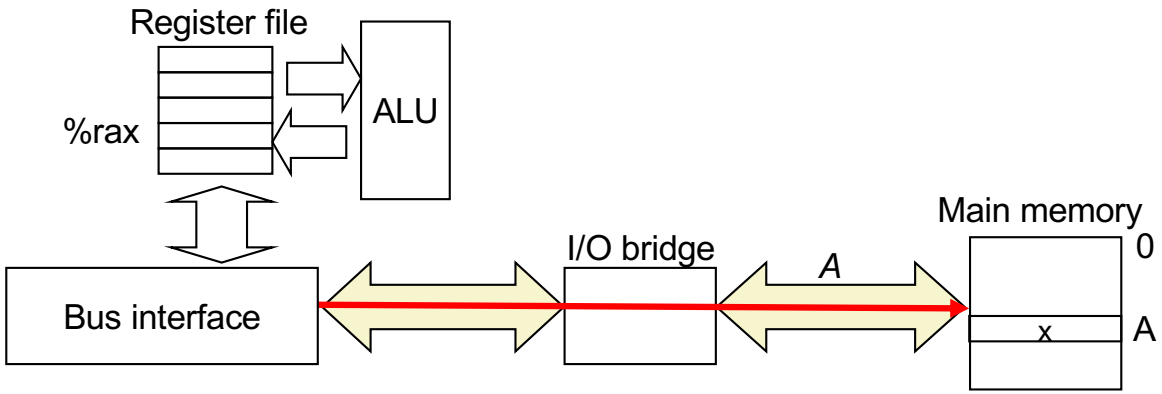
- 三类总线
  - 地址线
  - 数据线
  - 控制线
- I/O桥接器
  - 包括内存控制器
  - 将系统总线的信号翻译成内存总线的信号
  - 连接I/O总线（连接磁盘、图形卡等）



- 总线事务：在CPU和主存之间完成数据传送的一系列步骤
  - 读事务：从主存传数据到CPU
  - 写事务：从CPU传数据到主存

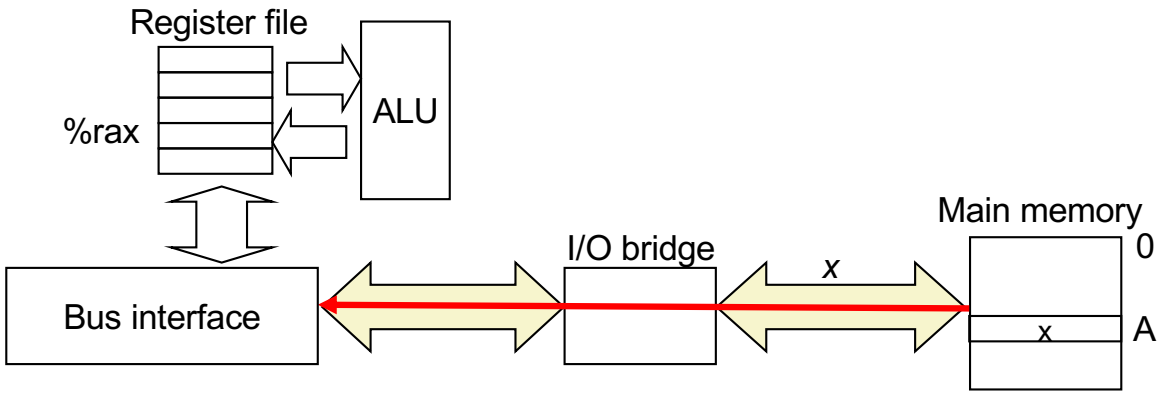
读事务

- movq A, %rax
  - 主存地址A的内容加载到寄存器%rax
- 总线接口发起读事务
  1. CPU将地址A放到系统总线上，I/O桥将信号传递到内存总线



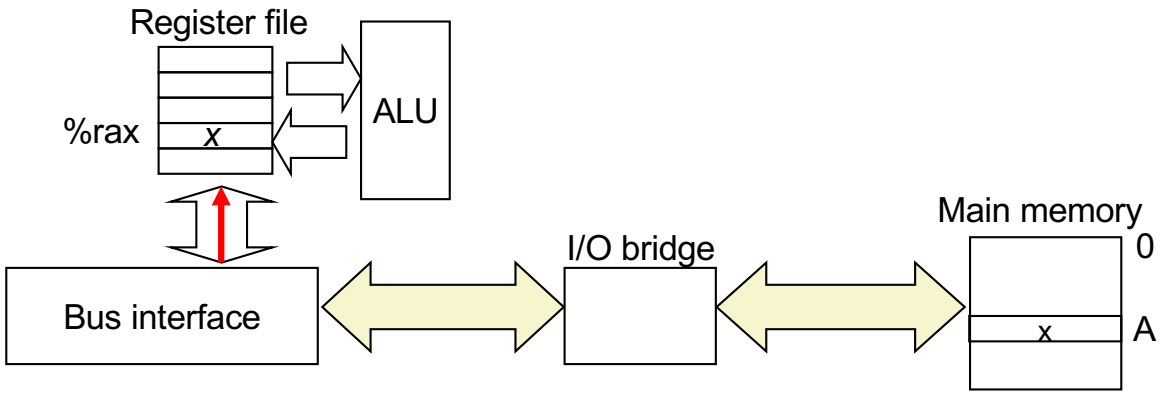
读事务

- movq A, %rax
  - 主存地址A的内容加载到寄存器%rax
- 总线接口发起读事务
  2. 主存从内存总线读地址A，从DRAM取数据字，将数据x写到内存总线；I/O桥翻译信号沿系统总线传递



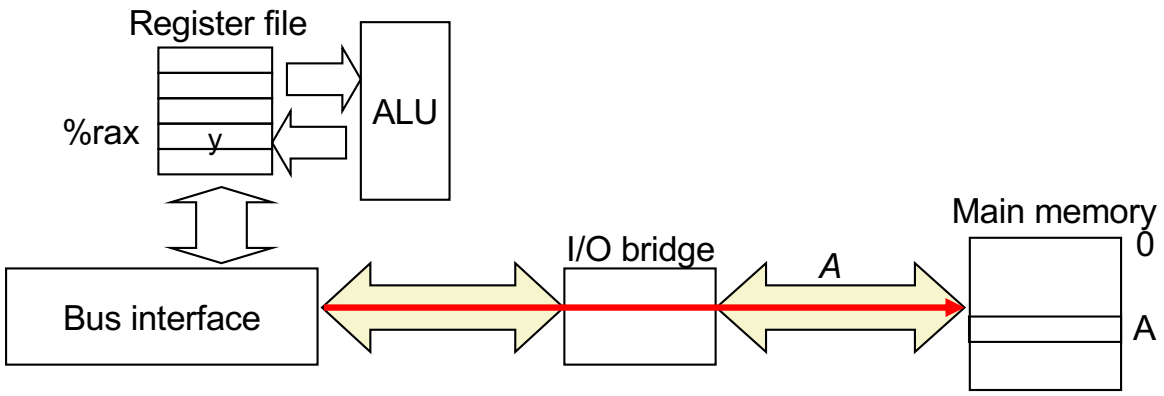
读事务

- movq A, %rax
  - 主存地址A的内容加载到寄存器%rax
- 总线接口发起读事务
  3. CPU从系统总线读数据，复制到寄存器%rax



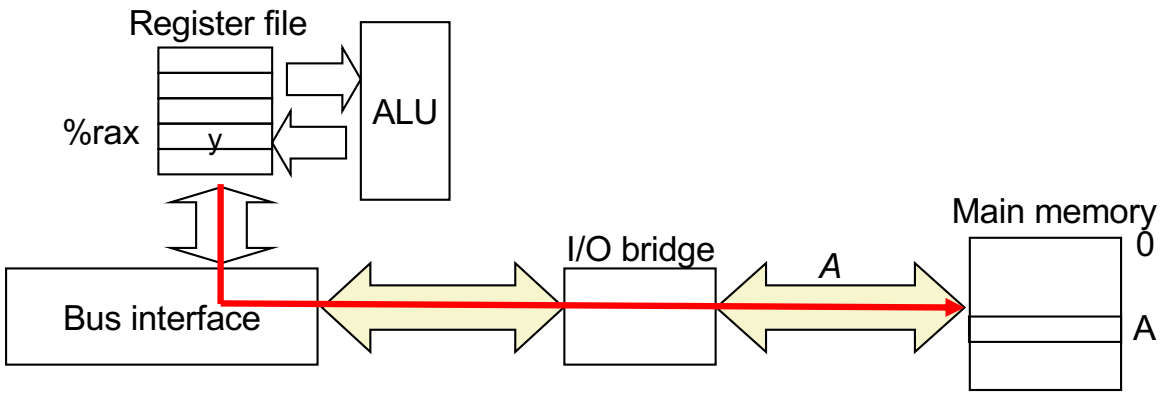
写事务

- movq %rax, A
  - 寄存器%rax的内容写到主存地址A
- CPU发起写事务
  1. CPU将地址A放到内存总线，主存读出地址A，等待数据字



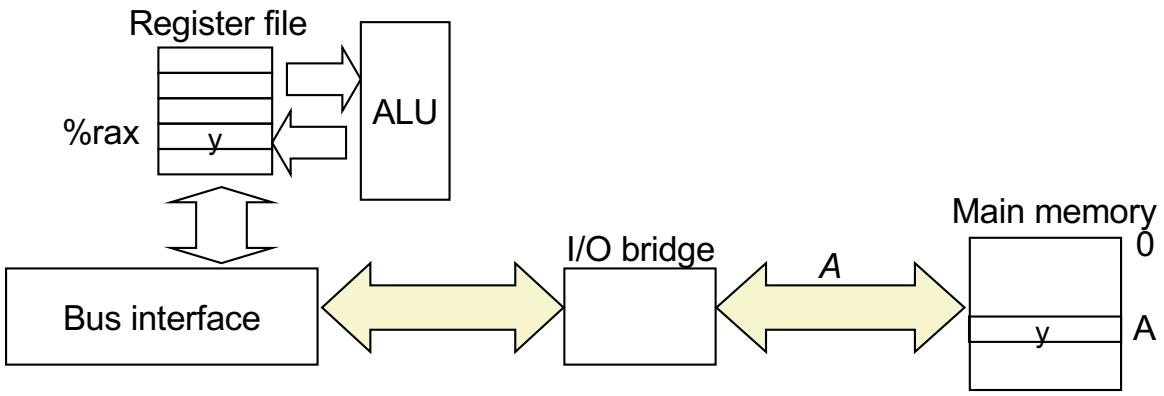
写事务

- movq %rax, A
  - 寄存器%rax的内容写到主存地址A
- CPU发起写事务
  2. CPU将数据字y放到总线上



写事务

- movq %rax, A
  - 寄存器%rax的内容写到主存地址A
- CPU发起写事务
  3. 主存从总线读出数据字y，写到地址A

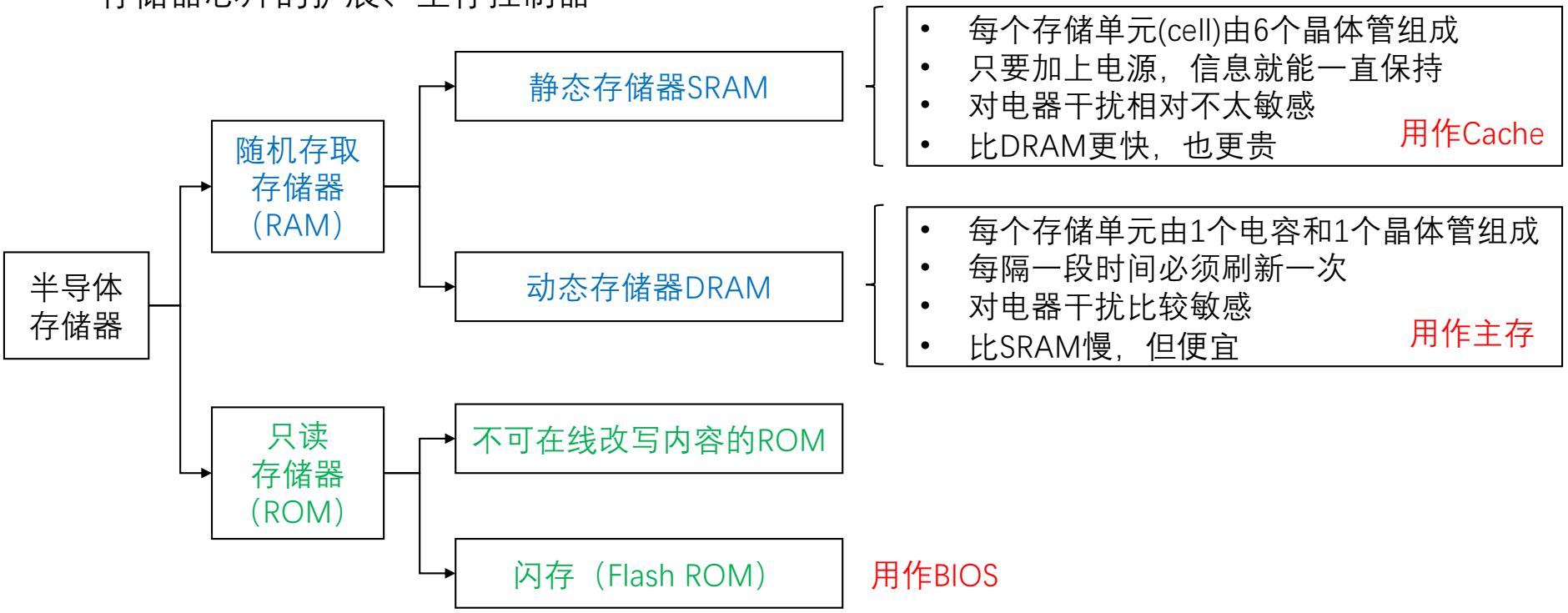


非易失性存储器

- 非易失性存储器（nonvolatile memory）即使是在断电后，仍然保存着它们的信息
  - 其中的程序被称为固件（firmware），通电后运行
    - BIOS
    - 图形卡/磁盘驱动
  - Read-only memory (ROM) 只读存储器
    - programmed during production
  - Programmable ROM (PROM) 可编程ROM
    - can be programmed once
  - Eraseable PROM (EPROM) 可擦写可编程ROM
    - 需要特殊物理设别，能够被擦除和重编程的次数的数量级可以达到千次
  - Electrically eraseable PROM (EEPROM) 电子可擦除PROM
    - 无需物理设备，能够被编程的次数的数量级可以达到十万次
  - 闪存（Flash Memory）
    - 基于EEPROM

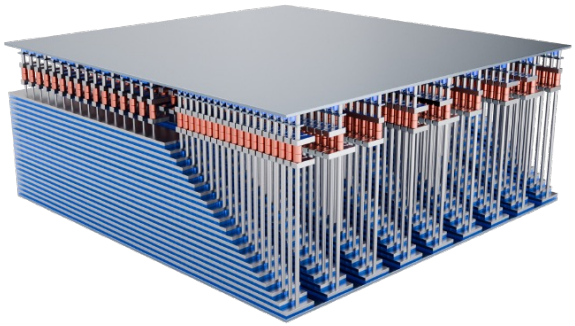
主存储器

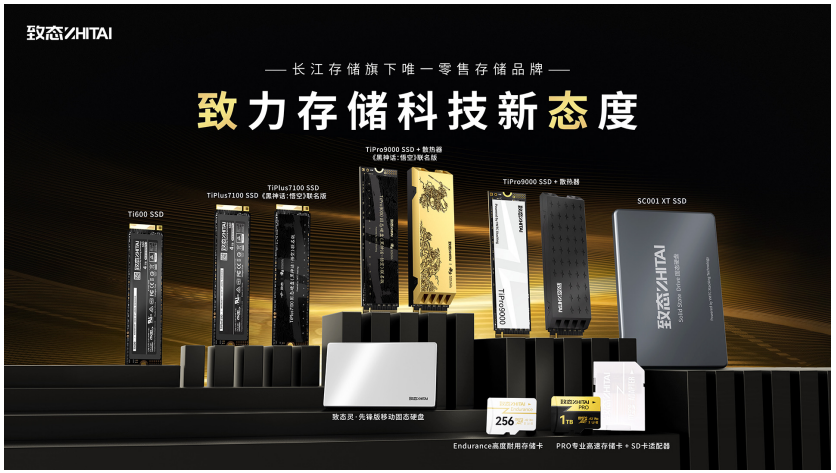
- 基本存储元件、DRAM芯片、 SDRAM芯片技术
- 内存条及其与CPU的连接
- 存储器芯片的扩展、主存控制器



长江存储

- 国产的长江存储已经能够量产全球最高堆叠层数（232层）的闪存芯片，具有自主知识产权，打破了海力士、美光、三星的垄断。





单选题 1分

设置

某容量为256MB的存储器由若干4M×8位的DRAM芯片构成，该DRAM芯片的地址引脚和数据引脚总数是（　　）。

A 19

B 22

C 30

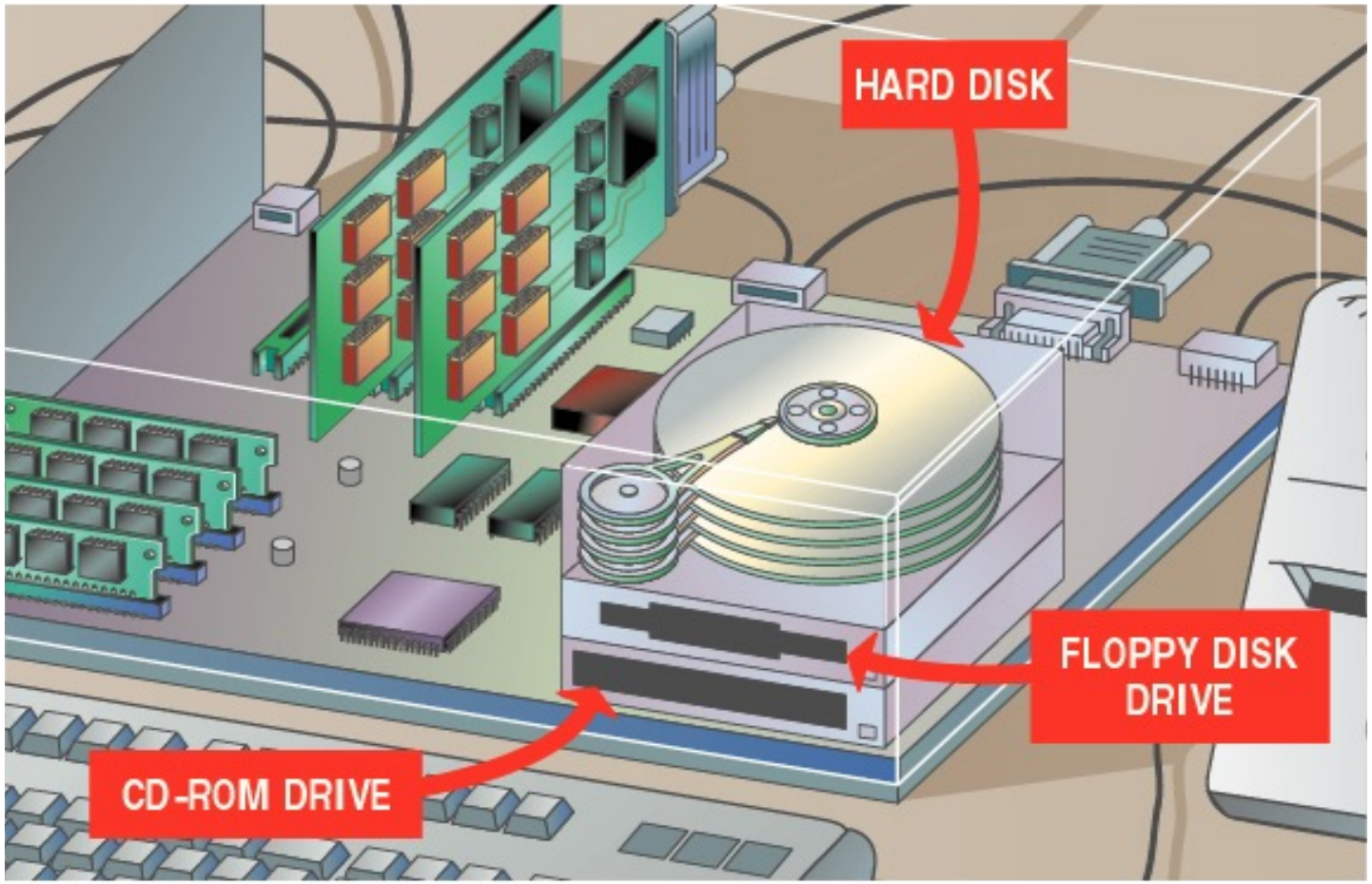
D 36

提交

层次结构存储系统

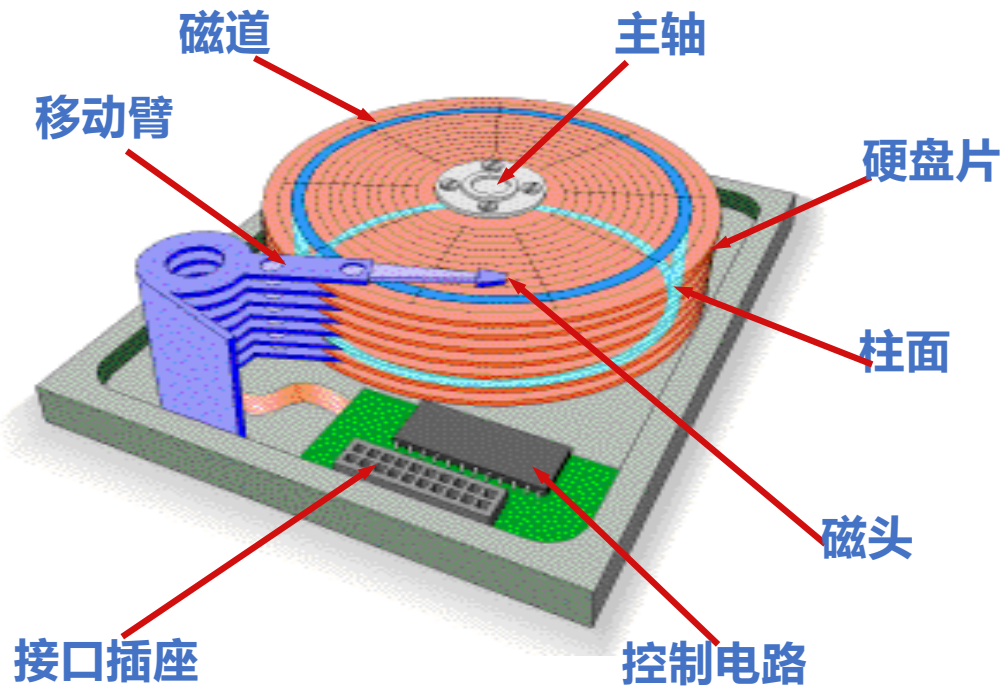
- 第一讲：存储器概述
- 第二讲：主存储器
  - 基本存储元件、DRAM芯片、 SRAM芯片技术
  - 内存条及其与CPU的连接
  - 存储器芯片的扩展、主存控制器
- 第三讲：外部存储器
  - 磁盘存储器、固态硬盘
- 第四讲：高速缓冲存储器（cache）
  - Cache的基本工作原理
  - 映射方式、替换算法、写策略
  - Cache的设计
  - Cache和程序性能

PC机中的外存储器



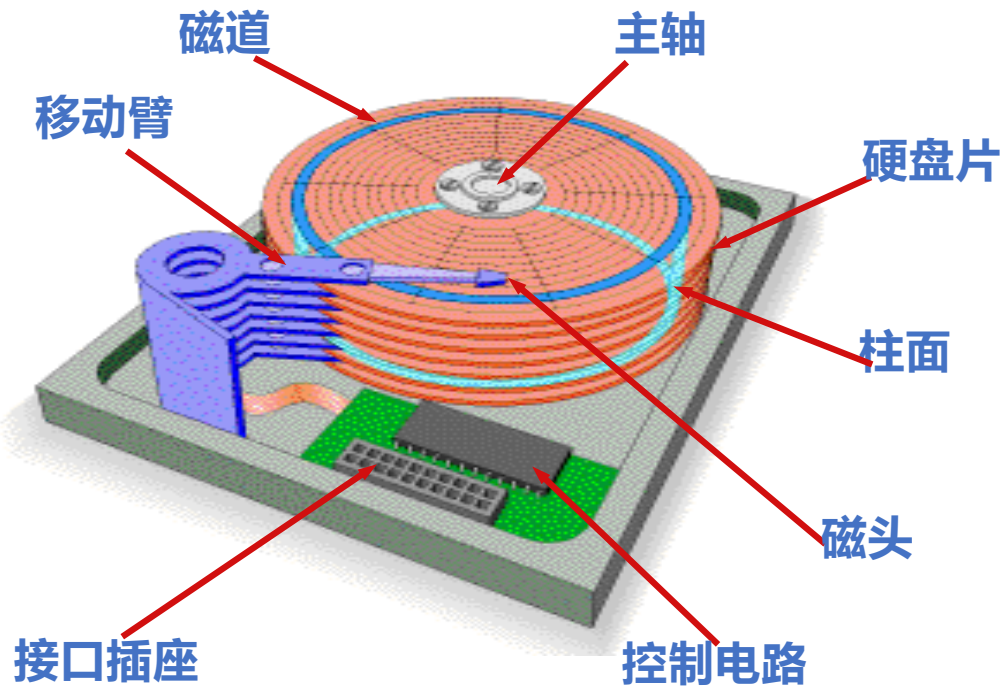
磁盘构造

- 磁盘（磁盘驱动器）是由一个或多个盘片构成的。每个盘片有两面，表面覆盖着磁性记录材料。

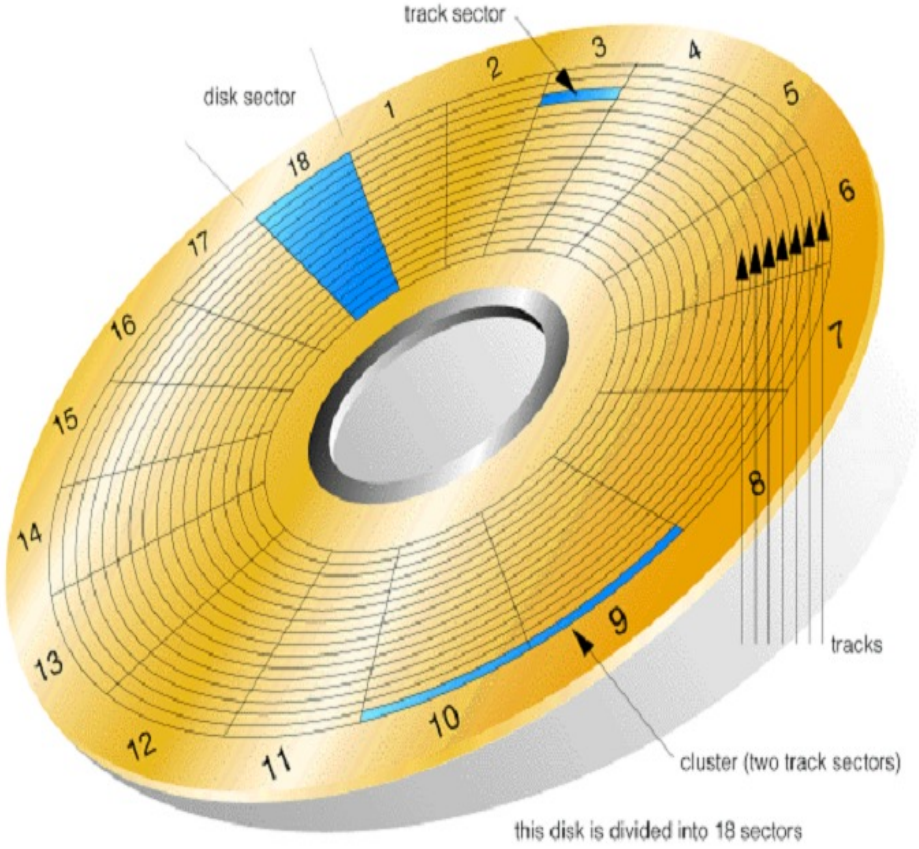


磁盘构造

- 盘片中央有一个可以旋转的主轴（spindle），它使得盘片以固定的旋转速率旋转，通常是5400~15000转每分钟(Revolution Per Minute , RPM) 。



磁盘构造



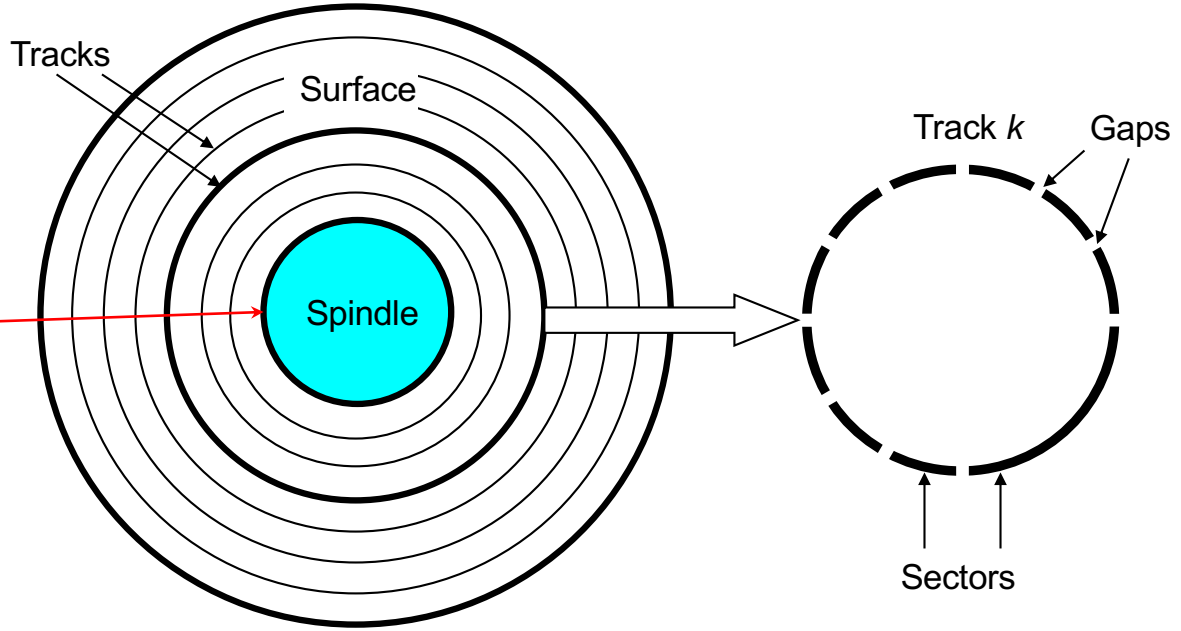
每个磁道被划分为若干段（扇区），每个扇区的存储容量相等。每个扇区都有一个编号 。

磁盘表面被分为许多同心圆，每个同心圆称为一个磁道。每个磁道都有一个编号，最外面的是0磁道。

磁盘构造

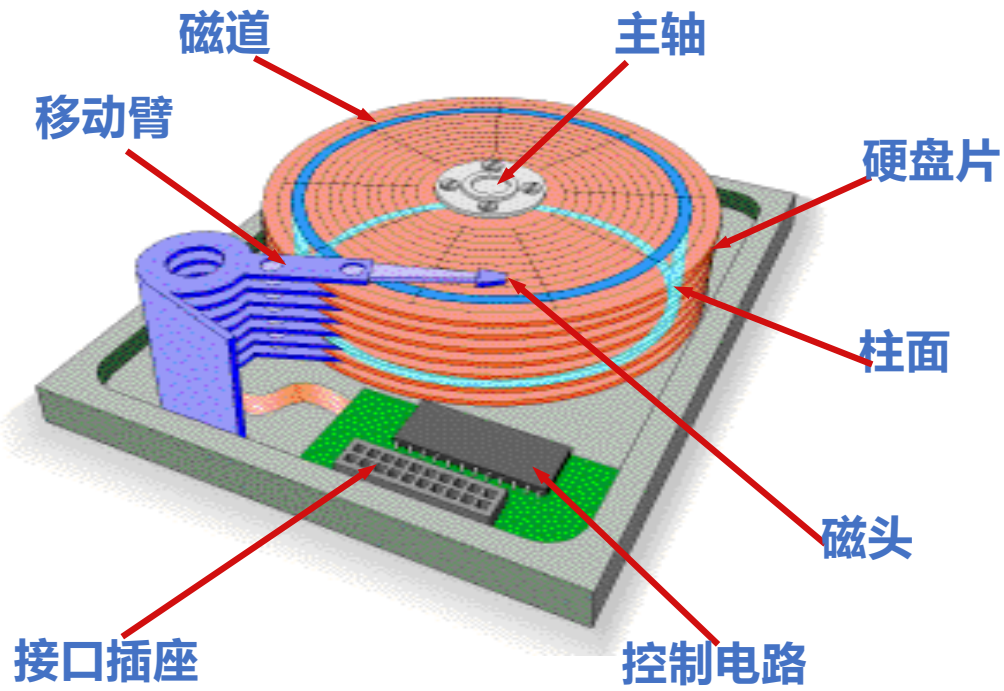
- 以前扇区的大小一般是512字节。现在多采用4096字节，通常称为4K扇区。
- 扇区之间由一些间隙（gap）分隔开，这些间隙不存储数据位，用作格式化标记。

每条磁道的位数由最内层磁道的周长决定



磁盘构造

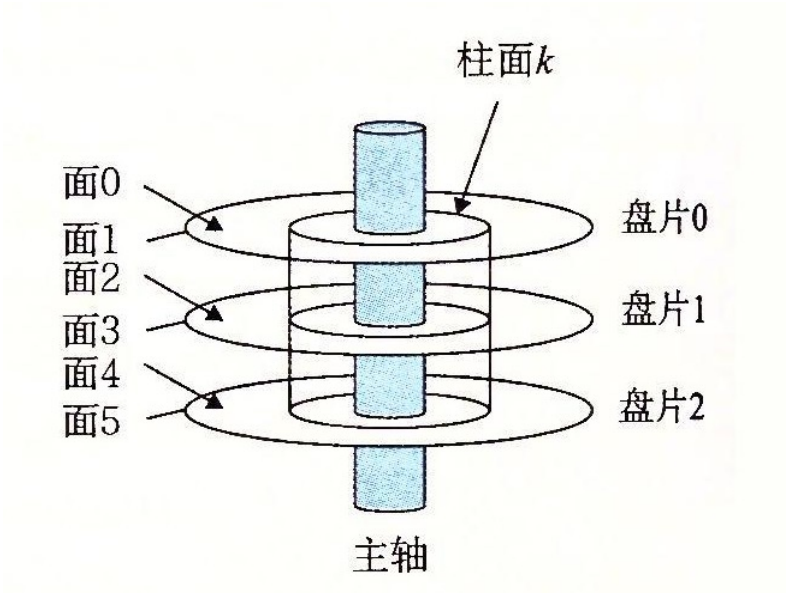
- 柱面是所有盘片表面上到主轴中心的距离相等的磁道的集合。



磁盘构造

- 柱面号就是磁道号，盘面号就是磁头号
- 每片有两个面，每面一个磁头
  - 例：假定有3片，1000个同心圆，每个圆分2000个扇区。
  - 则：磁盘空间划分为1000个柱面，每柱面6个磁头，每个磁道2000个扇区
- 磁盘数据地址

柱面号	盘面号	扇区号



磁盘容量

- 一个磁盘上可以记录的最大位数称为它的最大容量，或者简称为容量。

$$\text{磁盘容量} = \frac{\text{字节数}}{\text{扇区}} \times \frac{\text{平均扇区数}}{\text{磁道}} \times \frac{\text{磁道数}}{\text{表面}} \times \frac{\text{表面数}}{\text{盘片}} \times \frac{\text{盘片数}}{\text{磁盘}}$$

- 例:
  - 512 字节数/扇区
  - 300 平均扇区数/磁道
  - 20,000 磁道数/表面
  - 2 表面数/盘片
  - 5 盘片数/磁盘

  - 磁盘容量 = 512 x 300 x 20000 x 2 x 5
    = 30,720,000,000
    = 30.72 GB

磁盘容量

十进制前缀			IEC 定义的二进制前缀			值差
单词	前缀	值	单词	前缀	值	（%）
kilobyte	KB/kB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}	15%
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}	18%
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}	21%

磁盘操作

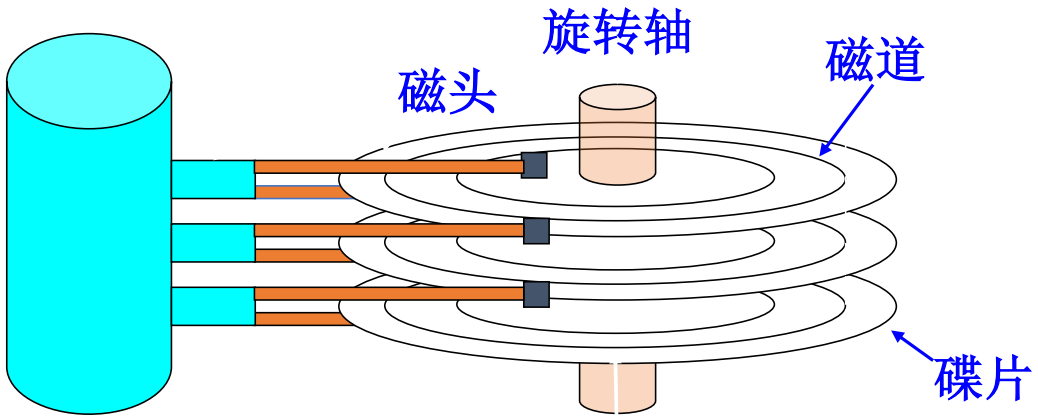
- 硬盘的操作流程如下：

  所有磁头同步寻道（由柱面号控制）

  → 选择磁头（由盘面号控制）

  → 被选中磁头等待扇区到达磁头下方（由扇区号控制）

  → 读写该扇区中数据



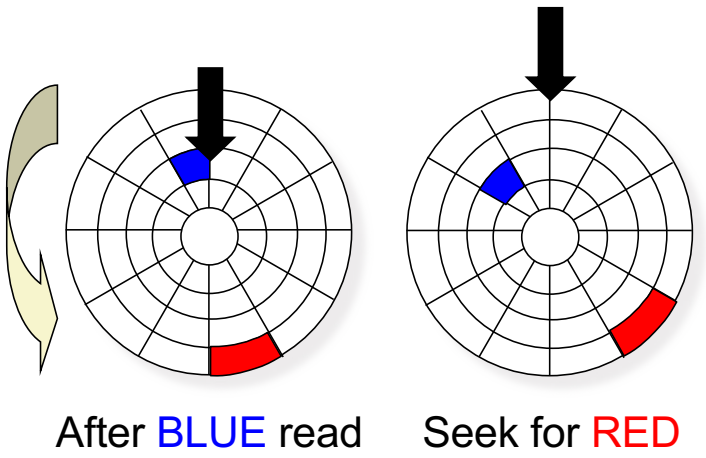
磁盘操作

所有磁头同步寻道（由柱面号控制）

→ 选择磁头（由盘面号控制）

→ 被选中磁头等待扇区到达磁头下方（由扇区号控制）

→ 读写该扇区中数据



寻道

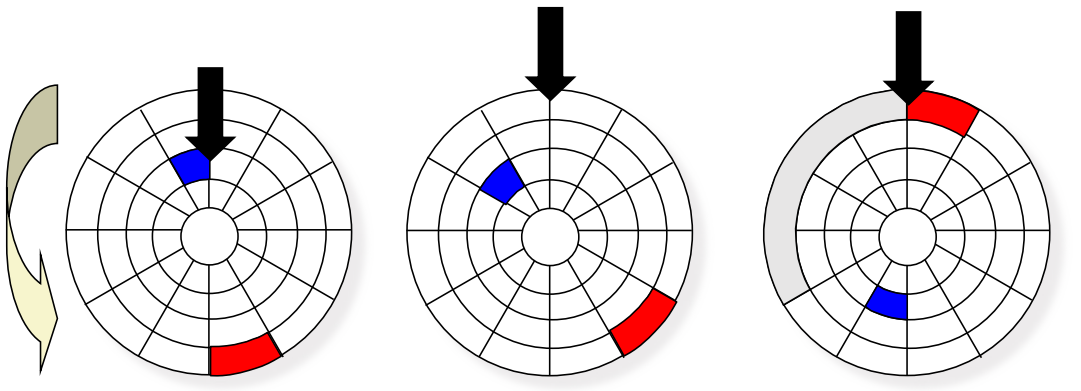
磁盘操作

所有磁头同步寻道（由柱面号控制）

→ 选择磁头（由盘面号控制）

→ 被选中磁头等待扇区到达磁头下方（由扇区号控制）

→ 读写该扇区中数据



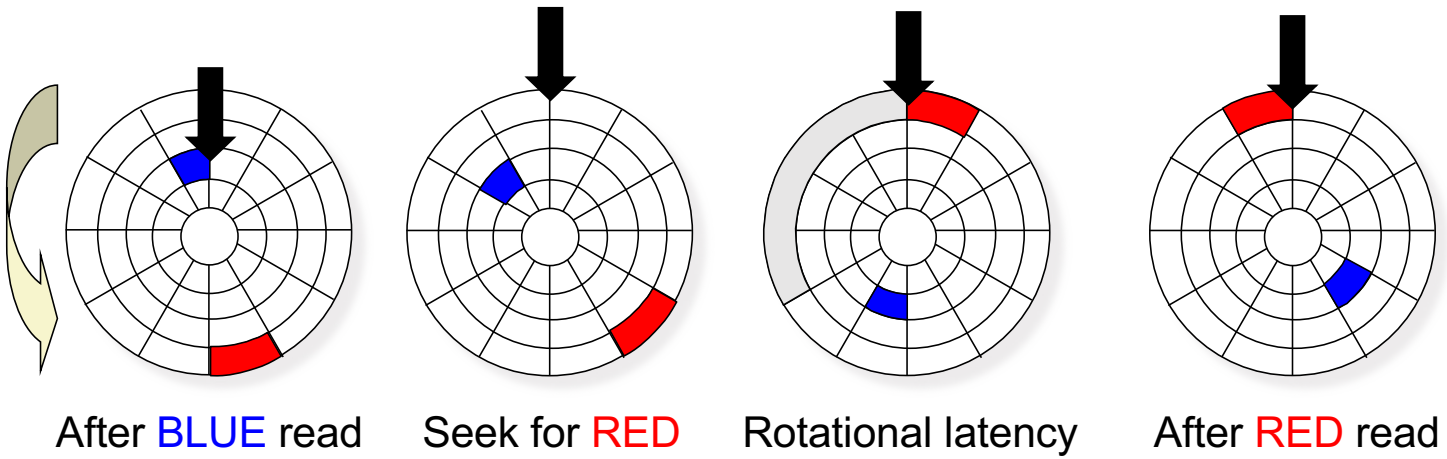
磁盘操作

所有磁头同步寻道（由柱面号控制）

→ 选择磁头（由盘面号控制）

→ 被选中磁头等待扇区到达磁头下方（由扇区号控制）

→ 读写该扇区中数据



单选题 1分

设置

如果某数据长度超过一个磁道的容量，应将它记录在同一个存储面上，还是记录在同一个柱面上?

A 记录在同一个存储面上

B 记录在同一个柱面上

提交

扇区访问时间组成

- 磁盘信息以扇区为单位进行读写
  - 扇区访问时间

    T = 平均寻道时间 + 平均旋转等待时间 + 数据传送时间
    - 平均寻道时间：磁头寻找到指定磁道所需平均时间
    - 平均旋转等待时间：指定扇区旋转到磁头下方所需平均时间

      =旋转半周所需要的时间=$0.5\times\frac{1}{\mathrm{RPM}}\times 60$ s
    - 数据传送时间：一个扇区的传送时间
      1. =每个扇区字节数/传输速率
      2. =旋转一个扇区所需要的时间=$\frac{1}{\text{平均扇区数/磁道}}\times\frac{1}{\mathrm{RPM}}\times 60$ s

磁盘访问时间计算

- 假定每个扇区512字节，磁盘转速为5400RPM，平均寻道时间为12ms，数据传输率为4MB/s，则磁盘扇区访问时间为多少？

解：扇区访问时间 = 平均寻道时间 + 平均旋转等待时间 + 数据传送时间

$$= 12 + 0.5\times\frac{1}{5400}\times 60\times 1000 + \frac{512}{4\times 10^6}\times 1000 \text{ (ms)}$$

$$= 12 + 5.56 + 0.13 = 17.69 \text{ ms}$$

方法1：数据传送时间=每个扇区字节数/传输速率

- 每个磁道有多少扇区？

解：数据传送时间=每个扇区字节数/传输速率= $\frac{1}{\text{平均扇区数/磁道}}\times\frac{1}{\text{RPM}}\times 60$ s

$$= \frac{512}{4\times 10^6} \text{ s} = \frac{1}{\text{平均扇区数/磁道}}\times\frac{1}{5400}\times 60 \text{ s}$$

平均扇区数/磁道 ≈ 87

访问时间比较

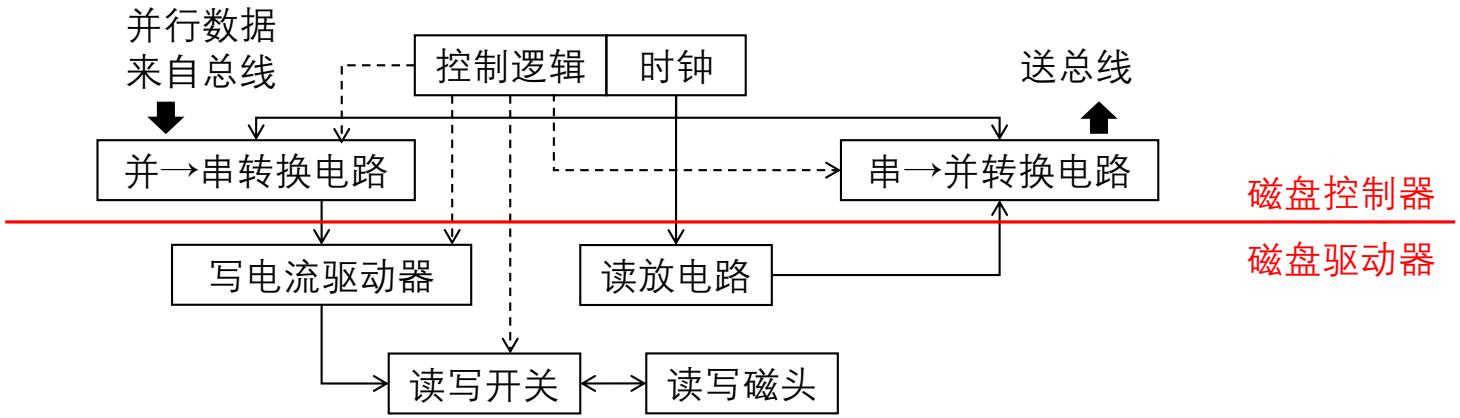
- 一个64位字，如果存储在SRAM中，访问时间约4ns；如果存储在DRAM中，访问时间约60ns。
- 现在要读512字节，SRAM访问耗时约256ns
- DRAM访问耗时约3840ns，约4us
- 磁盘访问耗时约10ms。

- 相同大小数据，磁盘访问时间，大约是
  - 访问DRAM的2500倍
  - 访问SRAM的40000倍
- 一般SRAM、DRAM、磁盘一次访问的数据量也是逐渐增大的

磁盘存储器的组成

- 磁记录介质：用来保存信息
- 磁盘驱动器：包括读写电路、读/写转换开关、磁头与磁头定位伺服系统等
- 磁盘控制器：包括控制逻辑、时序电路等
- 还包括数据缓存器、控制状态寄存器等

- 磁盘存储器的简化逻辑结构



逻辑磁盘块

- 向操作系统隐藏作为外部存储的磁盘的构造复杂性
- 现代磁盘将磁盘的构造呈现为一个B个扇区大小的逻辑块的序列，编号为0, 1, …, B-1。
- 磁盘控制器：维护逻辑块号和实际(物理)磁盘扇区之间的映射关系。
- 操作系统发起I/O操作，如读一个扇区的数据到主存
  1. 操作系统发送命令到磁盘控制器，让它读某个逻辑块号
  2. 控制器将逻辑块号翻译成一个地址三元组(柱面，盘面，扇区)
  3. （寻道）将读/写头移动到适当的柱面
  4. （旋转延迟）等待扇区移动到读/写头下
  5. （读写扇区）将读/写头感知到的位放到控制器上的一个小缓冲区中
  6. 将数据复制到主存中

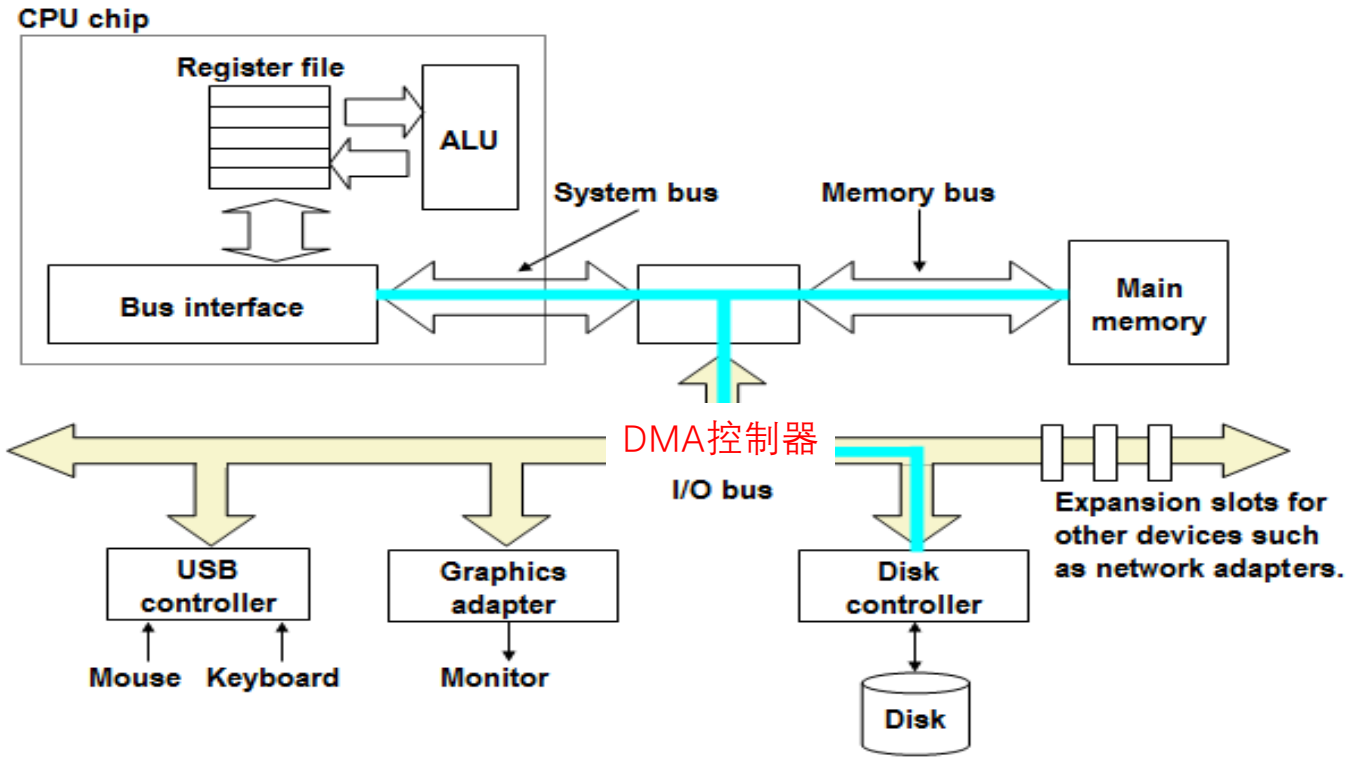
磁盘数据地址：

柱面号	盘面号	扇区号

磁盘存储器的连接

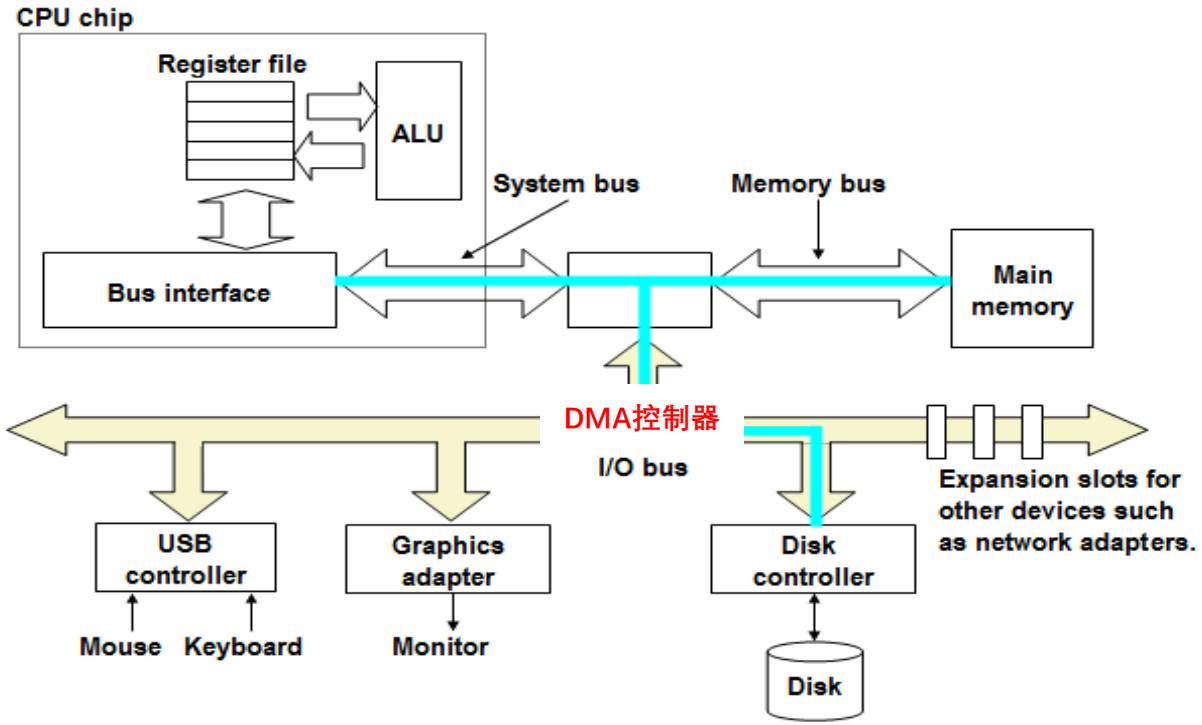
- 磁盘控制器连接在I/O总线上，I/O总线与其他总线（系统总线、存储器总线）之间用桥接器连接

- 磁盘的最小读写单位是扇区
  - 磁盘按成批数据交换方式进行读写
  - 采用直接存储器存取（DMA，Direct Memory Access）方式进行数据输入输出
  - 专门的DMA接口硬件来控制外设与主存间直接数据交换
    - 数据不通过CPU



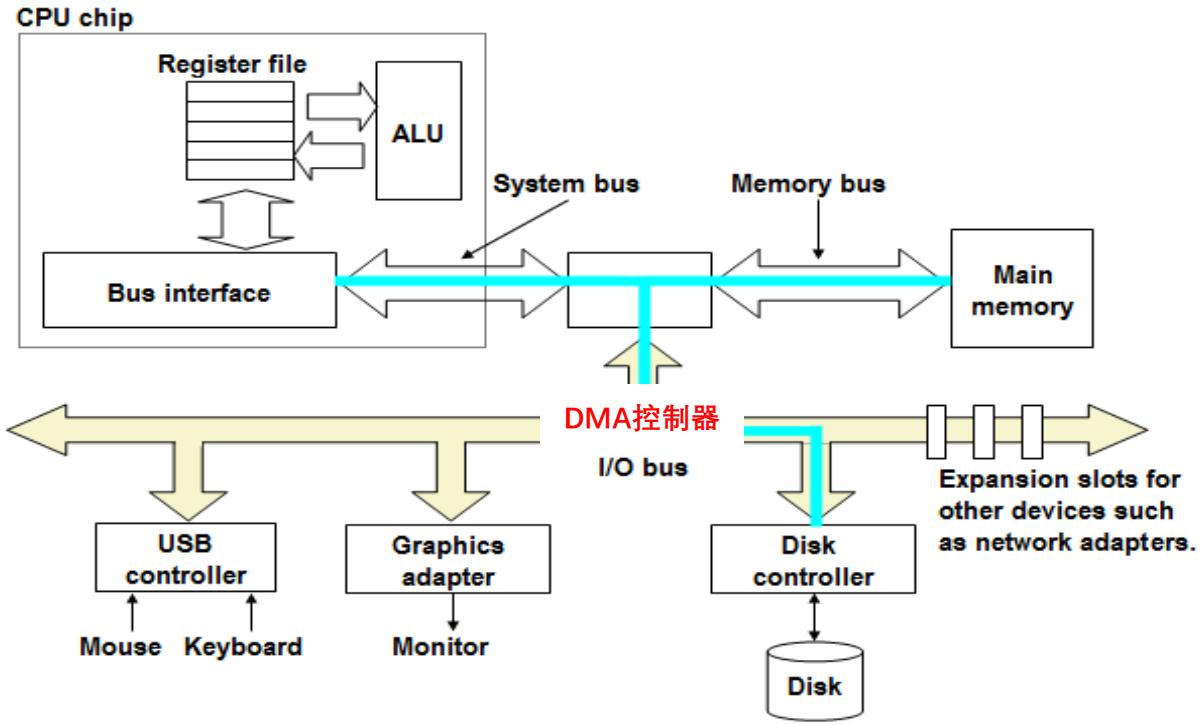
磁盘访问过程

- 第一步：CPU对DMA控制器初始化
  - 读/写命令、磁盘逻辑块号、主存起始地址、数据块大小
  - 然后启动磁盘驱动器工作



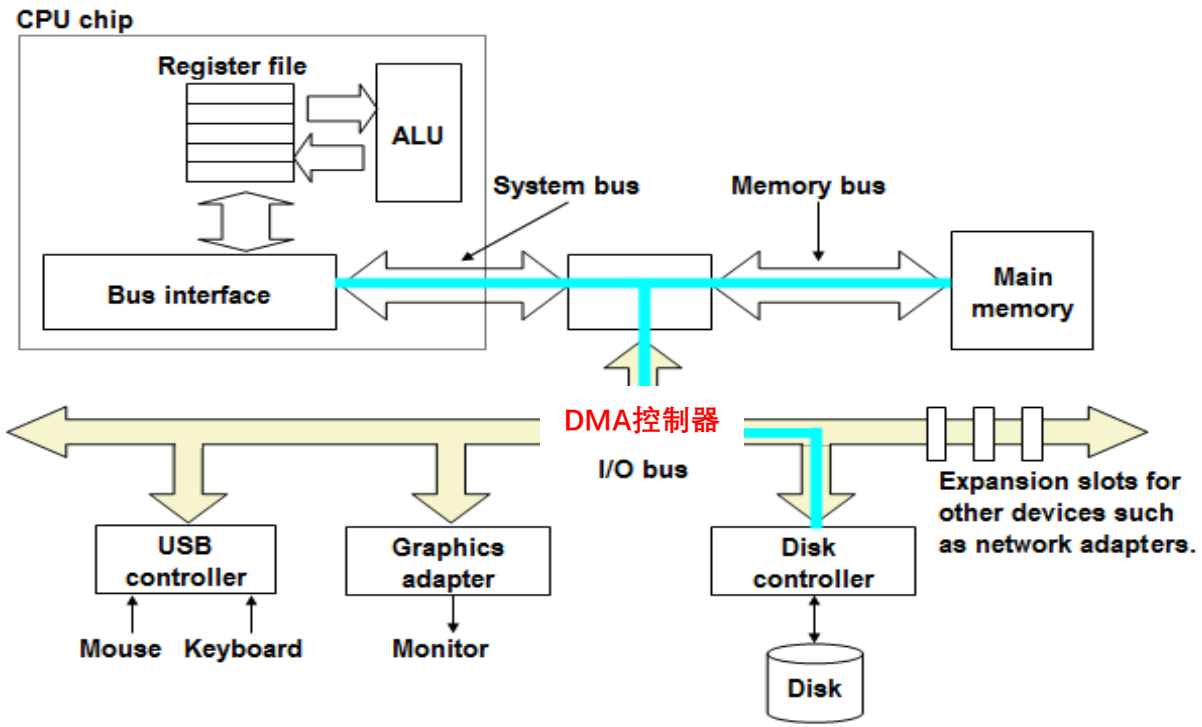
磁盘访问过程

- 第二步
  - 磁盘控制器读写相应的扇区，并按DMA方式把数据送主存
  - 此时CPU无需等待，可以执行其他指令



磁盘访问过程

- 第三步
  - 当DMA传送结束，DMA控制器发出DMA结束的中断请求，要求CPU进行相应的后处理



- 例：设某磁盘有两个记录面，每面磁道数1650，每磁道容量48538字节，转速为2400r/min，则：1）数据传输率是多少？2）设寻道时间是10～40ms，在一个磁道上写8000Byte数据，平均需要多少时间？

解：

1）（扇区）数据传送时间=每个扇区字节数/传输速率= $\frac{1}{\text{平均扇区数/磁道}} \times \frac{1}{\text{RPM}} \times 60\ \text{s}$

数据传输率=每磁道扇区数×每扇区字节数(=每磁道容量)×转速

=48538×2400/60=1941520 B/s。

2）平均寻道时间=(10+40)/2=25ms

平均旋转等待时间=磁盘旋转半周所需时间=$0.5 \times \frac{1}{2400} \times 60$ s = 0.0125 s=12.5 ms

数据传输时间=传输数据量(=扇区数×每扇区字节数)/数据传输率

= 8000/1941520 s = 4.12 ms

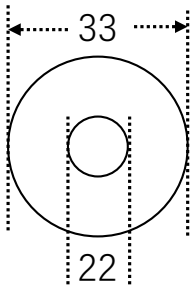
写入8000字节所需时间：

T=平均寻道时间+平均等待时间+数据传输时间

=25+12.5+4.12=41.62ms

- 例：磁盘组有6片磁盘，每片有两个记录面，最上最下两个面不用。存储区域内径22cm，外径33cm，道密度为40道/cm，内层位密度400位/cm，转速6000转/分。问：(1) 共有多少柱面？ (2) 盘组总存储容量是多少？(3) 数据传输率多少？

解：



(1) 有效存储区域 = (33-22)/2=5.5 cm
因为道密度=40道/cm，所以40×5.5=220道，即220个柱面
(2) 内层磁道周长为2πR = 2×3.14×11=69.08 cm
每道信息量= 400位/cm×69.08cm = 27632位 = 3454 B
每面信息量= 3454B×220 = 759880 B
盘组总容量=759880B×(6×2-2) = 7598800 B
(3) 磁盘数据传输率 D = rN，N为每条磁道容量，N=3454 B，r为磁盘转速，r=6000转/60秒=100转/秒。则 D = rN = 100×3454B=345400 B/s。

- 例：磁盘组有6片磁盘，每片有两个记录面，最上最下两个面不用。每面有220条磁道，每道信息量3454字节，问：采用定长数据块记录格式，即直接寻址的最小单位是一个记录块（一个扇区），若扇区大小为512字节，则寻址命令如何表示磁盘地址？

解：

该磁盘共有10个记录面，因此盘面号（即磁头号）需要4位；每个记录面有220条磁道，因此柱面号需要8位；若扇区大小为512字节，由每道信息量为3454字节可知，每道约有3454/512≈6.75个扇区，因此扇区号需要3位。可得到磁盘地址：

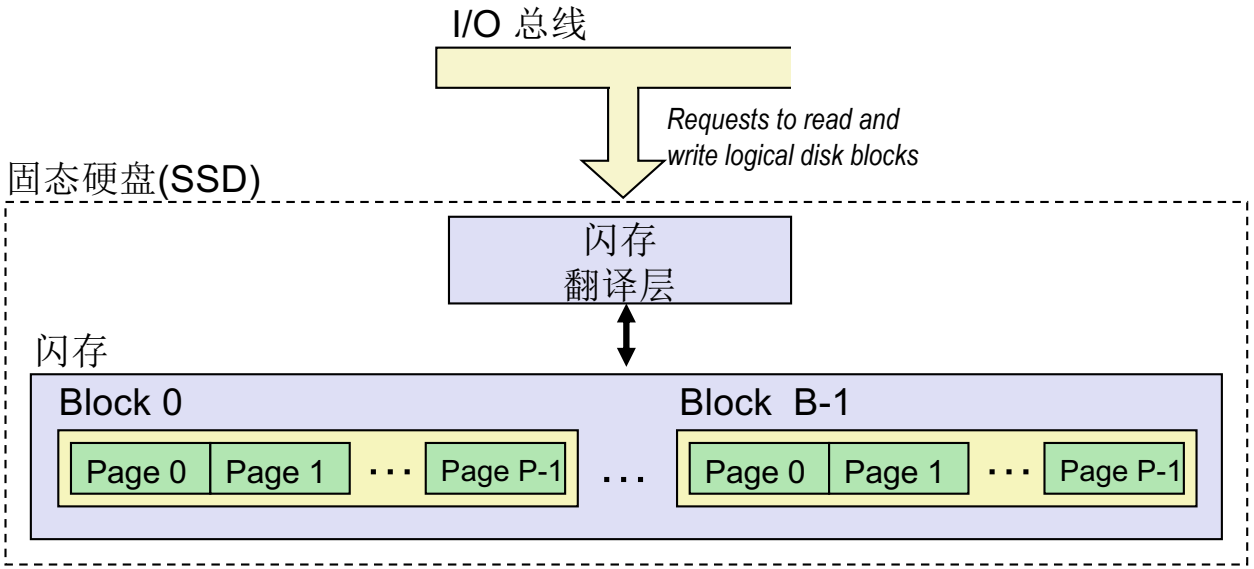
柱面号	盘面号	扇区号
8位	4位	3位

固态硬盘

- 固态硬盘（Solid State Disk，简称SSD）也被称为电子硬盘。
  - 它并不是一种磁表面存储器，而是一种使用Nand闪存组成的外部存储系统，与U盘并没有本质差别，只是容量更大，存取性能更好。
  - 电信号的控制使得固态硬盘的内部传输速率远远高于常规硬盘。
  - 随机读写时延比硬盘低两个数量级（随机读约几十微秒，随机写约几百微秒）。写操作比读操作慢。顺序读比顺序写大致快一倍，随机读比随机写大致快10倍。
  - 其接口规范和定义、功能及使用方法与传统硬盘完全相同，在产品外形和尺寸上也与普通硬盘一致。目前接口标准上使用USB、SATA和IDE，因此SSD是通过标准磁盘接口与I/O总线互连的。
  - 闪存的擦写次数有限，所以频繁擦写会降低其写入使用寿命。
    - 区块寿命的数量级大约是100000次重复写之后

固态硬盘

- 一个闪存芯片由若干个区块组成，每个区块由若干页组成。
  - 通常，页大小为512B~4KB，每个区块由32~128个页组成，因而区块大小16KB~512KB，数据可以按页为单位进行读写。
- 在SSD中有一个闪存翻译层，它将来自CPU的逻辑磁盘块读写请求翻译成对底层SSD物理设备的读写控制信号。因此，这个闪存翻译层相当于磁盘控制器。



提高机械硬盘的访问速度

教材，家庭作业6.22

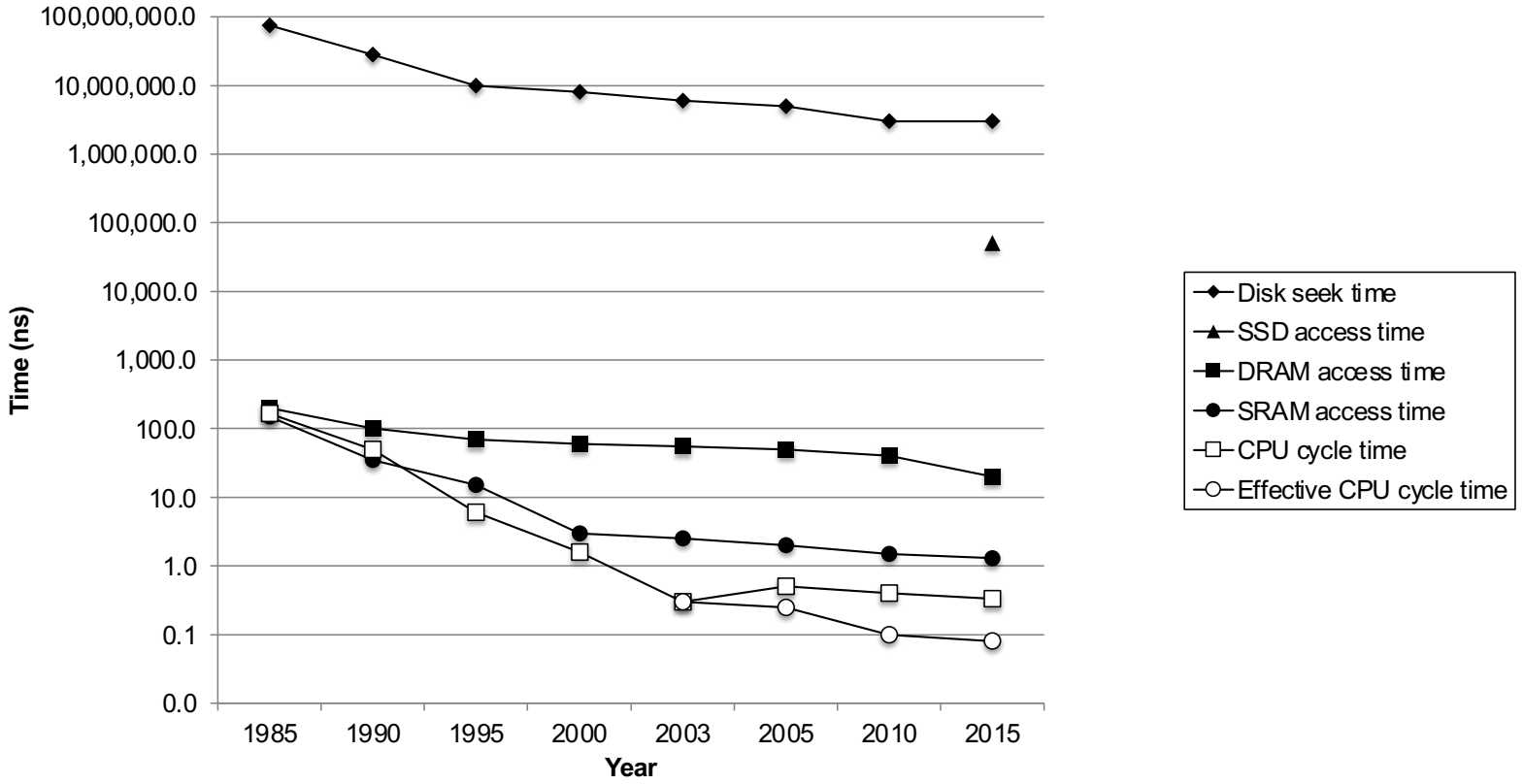
假设要求你设计一个每条磁道位数固定的旋转磁盘，知道每条磁道的位数是由最里层磁道的周长决定的，可以假设它就是中间那个圆洞的周长。因此，如果你把磁盘中间的洞做得大一点，每条磁道的位数就会增大，但是总的磁道数会减少。如果用r来表示盘面的半径，x*r表示圆洞的半径，那么x取什么值能使这个磁盘的容量最大？

层次结构存储系统

- 第一讲：存储器概述
- 第二讲：主存储器
  - 基本存储元件、DRAM芯片、 SRAM芯片技术
  - 内存条及其与CPU的连接
  - 存储器芯片的扩展、主存控制器
- 第三讲：外部存储器
  - 磁盘存储器、固态硬盘
- 第四讲：高速缓冲存储器（cache）
  - Cache的基本工作原理
  - 映射方式、替换算法、写策略
  - Cache的设计
  - Cache和程序性能

内存-外存-CPU速度差距

内存（DRAM）和外存（Disk/SSD）的性能滞后于CPU的性能



不存在的理想存储器

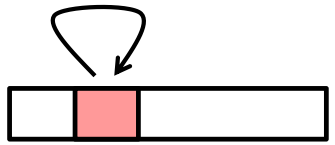
	容量	延迟	价格
寄存器	<1KB	1ns	$$$$
SRAM	1MB	2ns	$$$
DRAM	1GB	10ns	$$
Hard disk*	1000GB	10ms	$
不存在的理想存储器	100GB	1ns	$
*非易失性			

- 单独用某一种存储器，都不能满足我们的需求！
- 采用分层存储结构来构建计算机的存储体系！

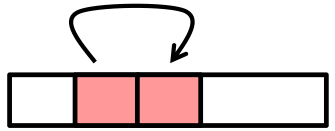
访存局部性

- 编写良好的程序应当具有良好的局部性
  - 更倾向于引用的数据项，临近于最近引用过的数据项，或者就是最近引用过的数据项本身

  - 时间局部性
    - 被引用过一次的内存位置很可能在不远的将来再次被多次引用
    - 循环、堆栈



  - 空间局部性
    - 如果一个内存位置被引用一次，程序很有可能在不远的将来引用其附近的内存位置
    - 指令顺序执行、数组



程序的局部性原理 例1

源程序

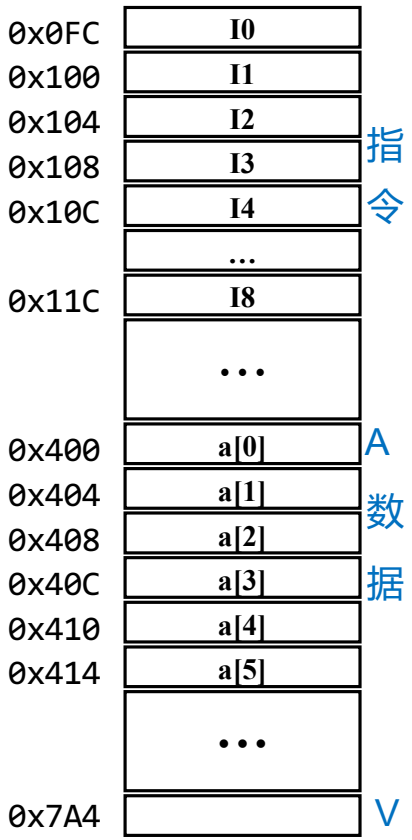
```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
*v = sum;
```

- 每条指令4字节
- 每个数组元素4字节
- 指令和数组元素在内存中分别连续存放
- sum, ap, i, t 均为通用寄存器
- A, V为内存地址

汇编程序伪码

```
I0:        sum <- 0
I1:        ap <- A          A是数组a的起始地址
I2:        i <- 0
I3:        if (i >= n) goto done
I4: loop:  t <- (ap)        数组元素a[i]的值
I5:        sum <- sum+t     累计在sum中
I6:        ap <- ap+4       计算下个数组元素地址
I7:        i <- i+1
I8:        if (i < n) goto loop
I9: done:  V <- sum         累计结果保存至地址V
```

主存布局



程序的局部性原理 例1

源程序

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

- 每条指令4字节
- 每个数组元素4字节
- 指令和数组元素在内存中分别连续存放
- sum, ap, i, t 均为通用寄存器
- A, V为内存地址

汇编程序伪码

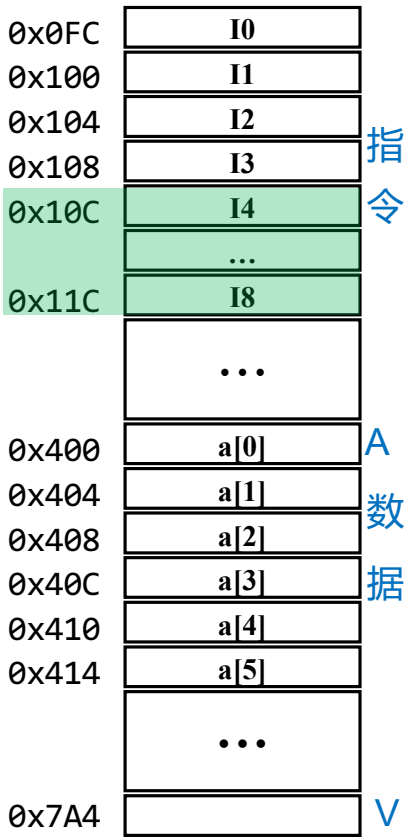
```
I0:        sum <- 0
I1:        ap <- A          A是数组a的起始地址
I2:        i <- 0
I3:        if (i >= n) goto done
I4: loop:  t <- (ap)        数组元素a[i]的值
I5:        sum <- sum+t     累计在sum中
I6:        ap <- ap+4       计算下个数组元素地址
I7:        i <- i+1
I8:        if (i < n) goto loop
I9: done:  V <- sum         累计结果保存至地址V
```

循环n次

若n足够大，则在一段时间内一直在局部区域内执行指令，故循环内指令的时间局部性好

指令顺序执行，空间局部性好

主存布局



程序的局部性原理 例1

源程序

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

- 每条指令4字节
- 每个数组元素4字节
- 指令和数组元素在内存中分别连续存放
- sum, ap, i, t 均为通用寄存器
- A, V为内存地址

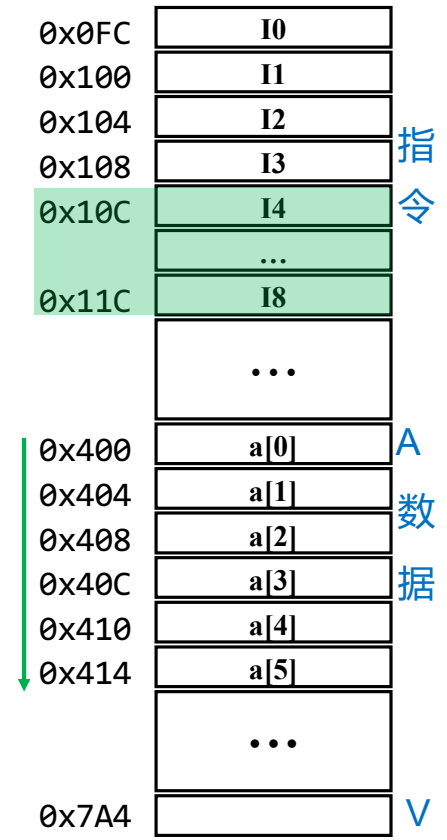
汇编程序伪码

```
I0:        sum <- 0
I1:        ap <- A          A是数组a的起始地址
I2:        i <- 0
I3:        if (i >= n) goto done
I4: loop:  t <- (ap)        数组元素a[i]的值
I5:        sum <- sum+t     累计在sum中
I6:        ap <- ap+4       计算下个数组元素地址
I7:        i <- i+1
I8:        if (i < n) goto loop
I9: done:  V <- sum         累计结果保存至地址V
```

数组元素按顺序存放，按顺序访问，故数据的空间局部性好

每个数组元素都只被访问1次，故数据没有时间局部性

主存布局



程序的局部性原理 例2

- 考虑功能相同的程序段A和B，对于数组a、变量sum和for循环体指令而言，A和B谁的时间局部性好，谁的空间布局性好？

程序段A：

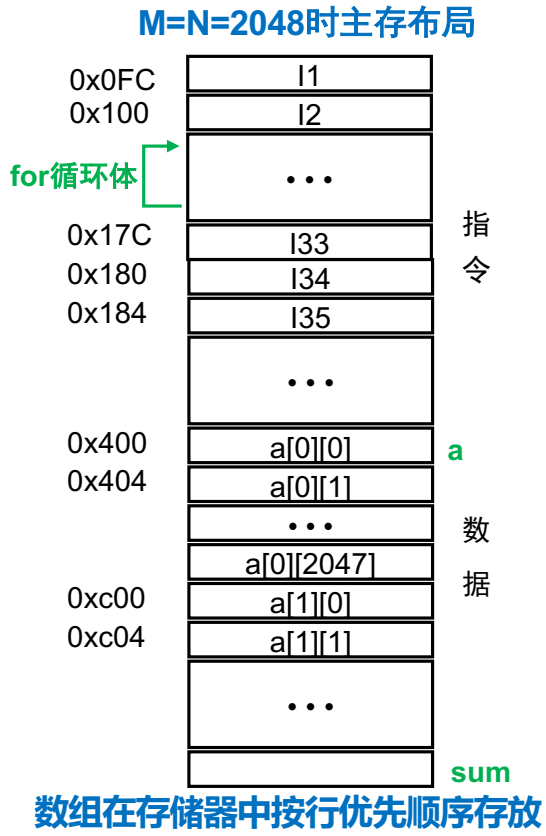
```
int sumarrayrows(int a[M][N]) {
  int i, j, sum=0;
  for (i=0; i<M, i++)
    for (j=0; j<N, j++)
      sum+=a[i][j];
  return sum;
}
```

程序段B：

```
int sumarraycols(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;
}
```

A

- 数组a
  - 访问顺序为a[0][0], a[0][1], …, a[0][2047], a[1][0], …，与存放顺序一致，空间局部性好
  - 每个a[i][j]访问1次，无时间局部性
- 变量sum
  - 单变量，无空间局部性
  - 每次循环访问一次，时间局部性好
- for循环体指令
  - 指令顺序存放，空间局部性好
  - 重复执行M*N次，时间局部性好



程序的局部性原理 例2

- 考虑功能相同的程序段A和B，对于数组a、变量sum和for循环体指令而言，A和B谁的时间局部性好，谁的空间布局性好？

程序段A：

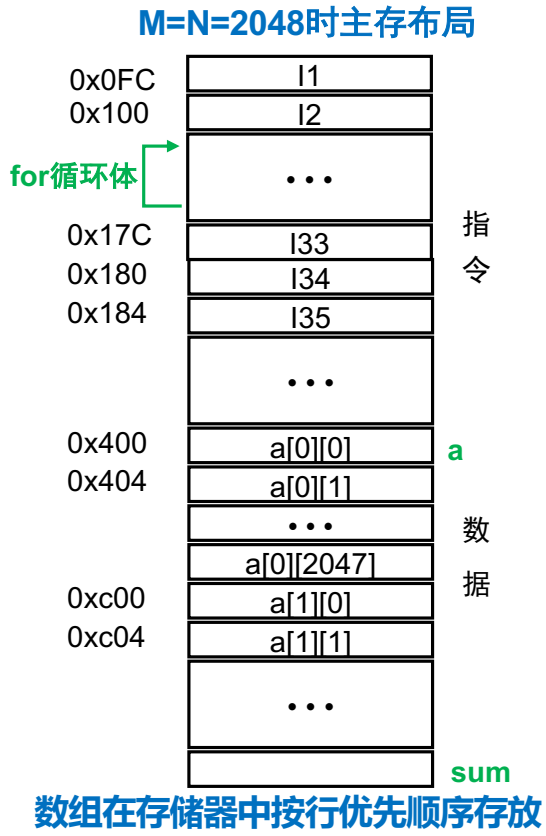
```
int sumarrayrows(int a[M][N]) {
  int i, j, sum=0;
  for (i=0; i<M, i++)
    for (j=0; j<N, j++)
      sum+=a[i][j];
  return sum;
}
```

程序段B：

```
int sumarraycols(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;
}
```

B

- 数组a
  - 访问顺序为a[0][0], a[1][0], …, a[2047][0], a[0][1], …，每次跳过2048个单元访存，空间局部性差（假设缓存小于8KB）
  - 每个a[i][j]访问1次，无时间局部性
- 变量sum
  - 单变量，无空间局部性
  - 每次循环访问一次，时间局部性好
- for循环体指令
  - 指令顺序存放，空间局部性好
  - 重复执行M*N次，时间局部性好



程序的局部性原理 例2

- 考虑功能相同的程序段A和B，对于数组a、变量sum和for循环体指令而言，A和B谁的时间局部性好，谁的空间布局性好？

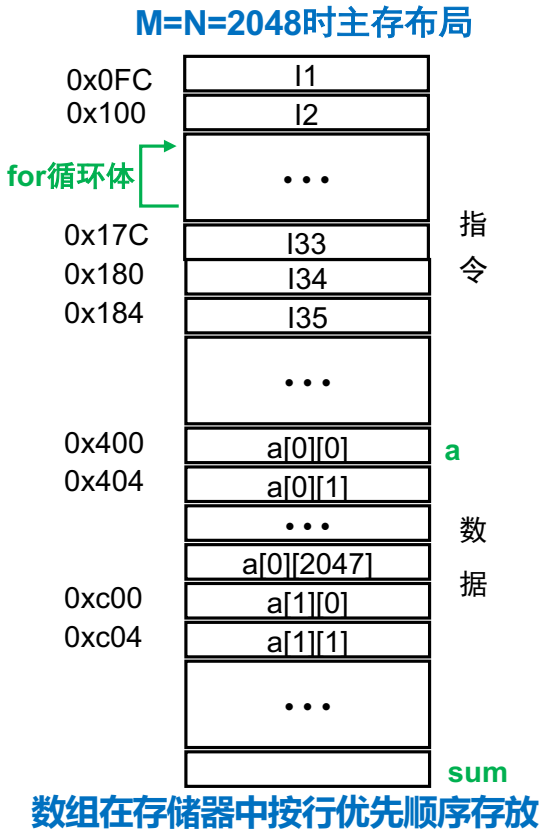
程序段A：

```
int sumarrayrows(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;
}
```

程序段B：

```
int sumarraycols(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;
}
```

- 数组a
  - A：空间局部性好
  - B：空间局部性差
- 实际运行结果（2GHz Intel Pentium 4）
  - A：59,393,288 时钟周期
  - B：1,277,877,876 时钟周期
- A比B快超过20倍!



存储器层次结构

- 存储体系：把各种不同存储容量、不同存取速度、不同价格的存储器，组成层次结构，并通过管理软件和辅助硬件将不同性能的存储器组合成有机的整体，称为计算机的存储层次或存储体系。
  - 特点

    1. 包含性

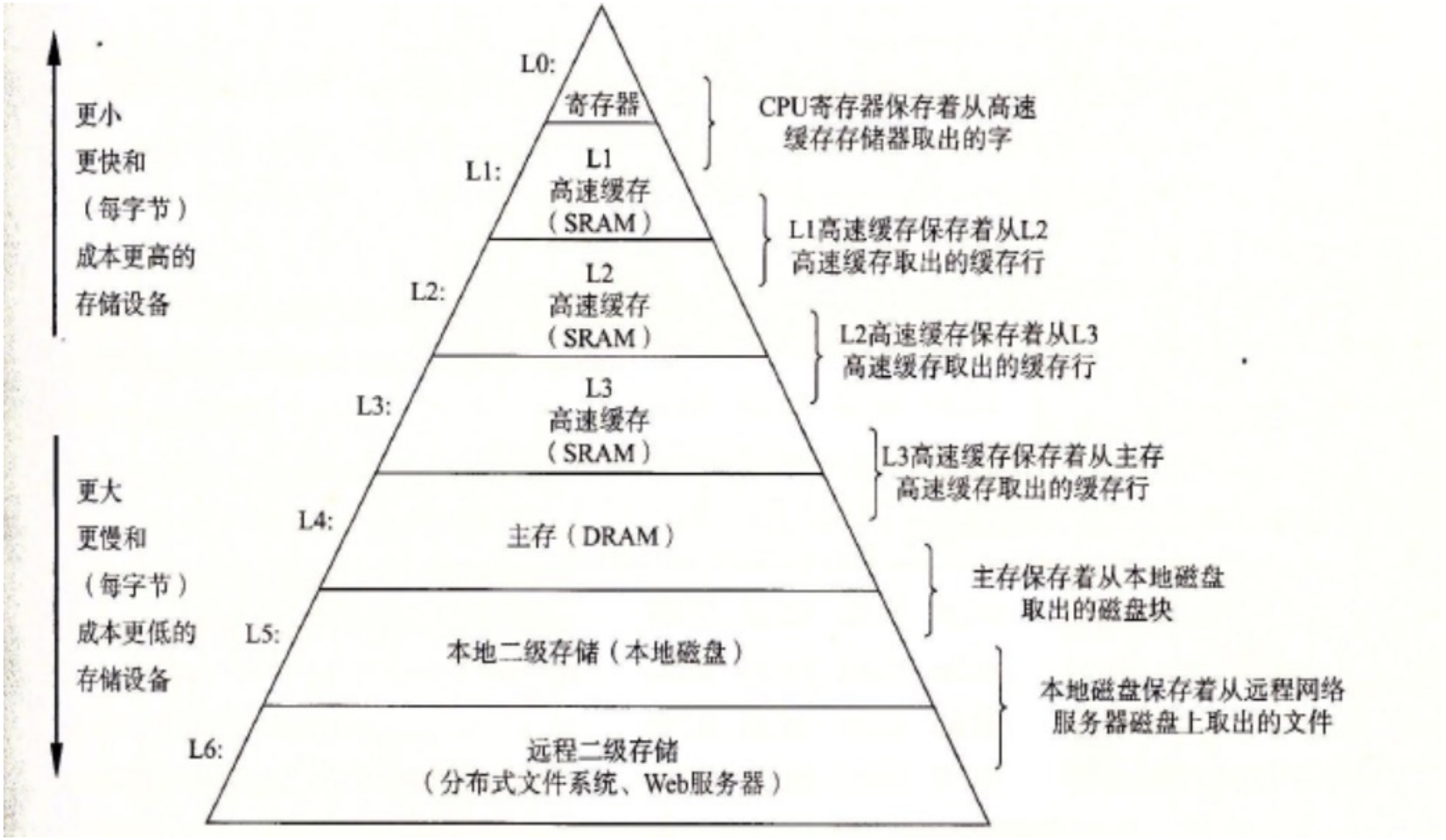
    $M_0 \subset M_1 \subset M_2 \subset \ldots\ldots \subset M_n$

    - 所有信息项最初存放在最外层M_n，在处理过程中，它的子集被复制到M_{n-1}，同样，M_{n-1}的子集被复制到M_{n-2}，……
    - 如果在M_i中找到一个信息字，那么同一个字的复制品在所有的高层M_{i+1}，M_{i+2}，……，M_n中都一定可以找到

    2. 相邻层之间的数据传送单位（典型字节值）

    - CPU ↔ 高速缓存：字（4/8字节）
    - 高速缓存 ↔ 主存储器：块（32字节）
    - 主存 ↔ 磁盘：页（4K字节）

存储器层次结构


更小
更快和
（每字节）
成本更高的
存储设备
更大
更慢和
（每字节）
成本更低的
存储设备
L0:
寄存器
CPU寄存器保存着从高速缓存存储器取出的字
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高速缓存取出的缓存行
L4:
主存（DRAM）
主存保存着从本地磁盘取出的磁盘块
L5:
本地二级存储（本地磁盘）
本地磁盘保存着从远程网络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）

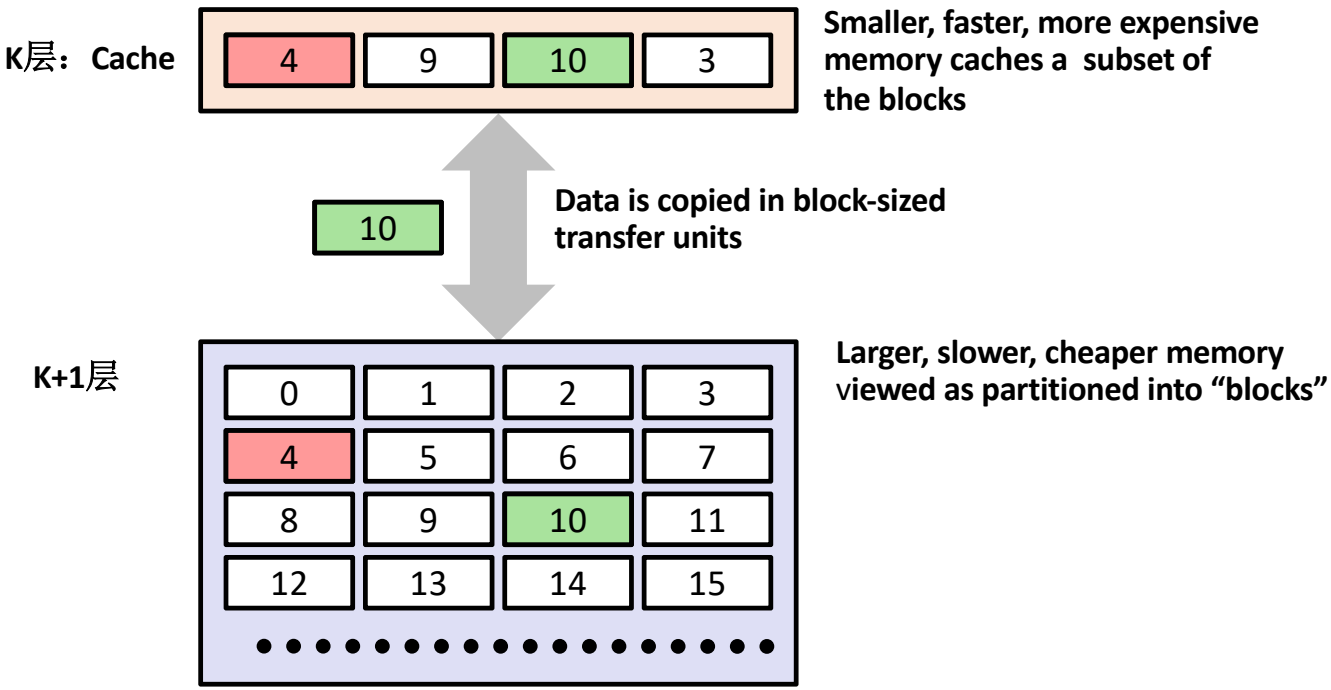
存储器层次结构（memory hierarchy）

- 基于缓存的存储器层次结构
  - 中心思想：每一层存储设备都可以视为较低一层的缓存。即对于每个k，位于k层的更快更小的存储设备作为位于k+1 层的更大更慢的存储设备的缓存。
    - 本地磁盘作为通过网络从远程磁盘取出的文件(例如Web页面)的缓存
    - 主存作为本地磁盘上数据的缓存
    - 高速缓存：主存的缓存
    - CPU寄存器组：高速缓存Cache的缓存
  - 有效性原因
    - 较慢的存储设备比较快的存储设备便宜
    - 程序倾向于展示局部性
      - 时间局部性：同一数据可能被多次使用，如果第一次访问不在上层存储中，则从下层复制传送，则后面的访问会更快
      - 空间局部性：把多个数据对象组织在一个块中，复制传送到上层存储，增加可能的访问速度
  - 类比：工厂的多级仓库

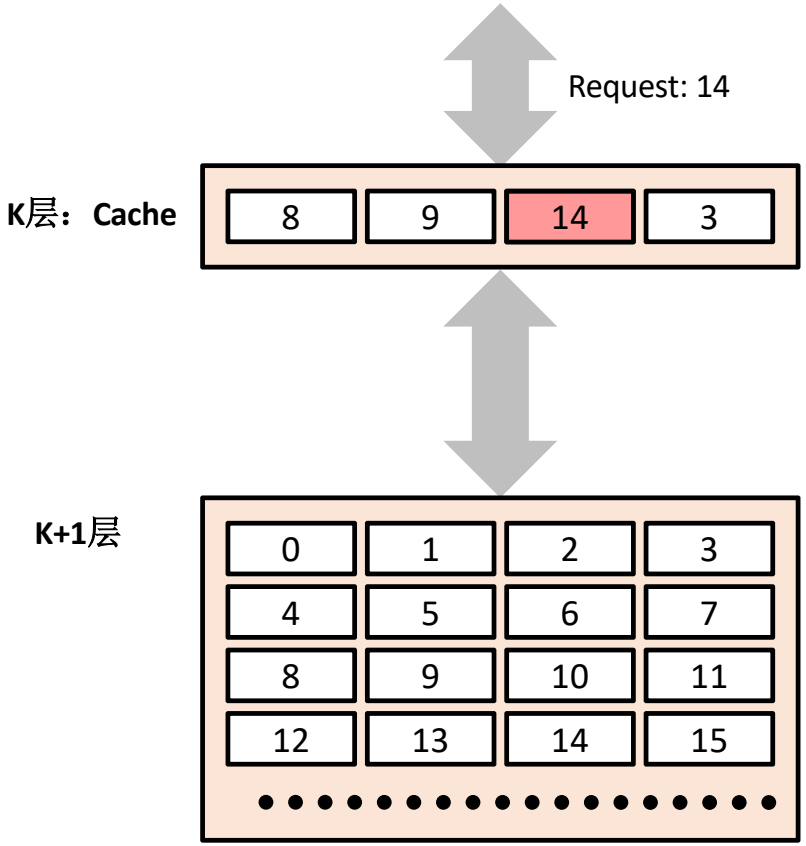
更靠近CPU　　更远离CPU

通常为固定大小，下层数据按临近关系分成若干块，上层缓存着下层的部分块

存储器层次结构（memory hierarchy）



存储器层次结构（memory hierarchy）

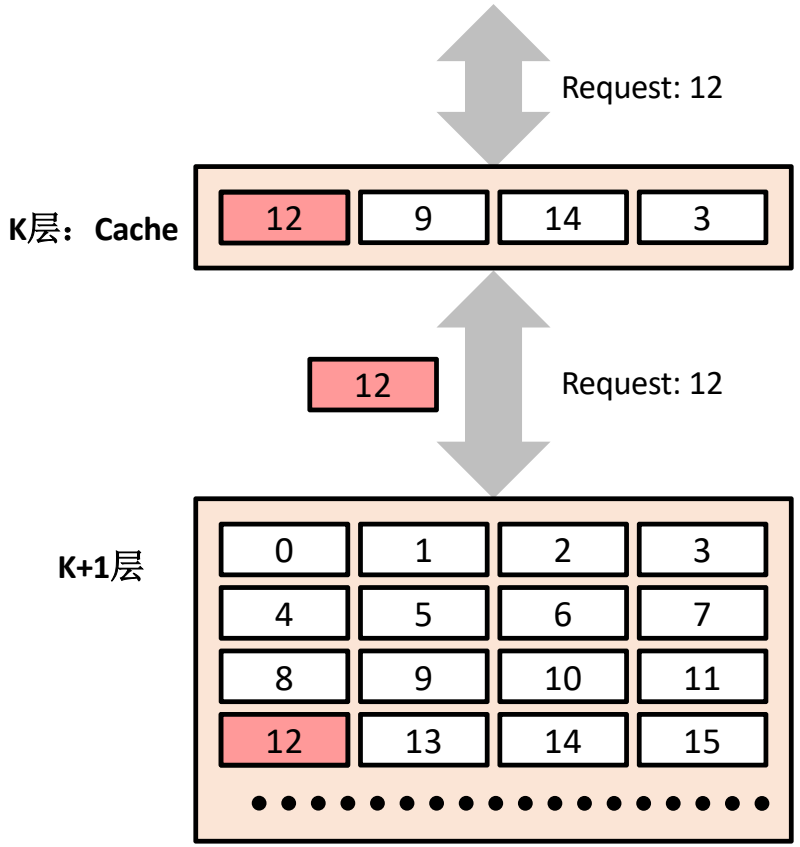


Data in block b is needed

Block b is in cache:

Hit命中!

存储器层次结构（memory hierarchy）



Data in block b is needed

Block b is not in cache:

Miss不命中! 强制性/冷不命中 冷缓存导致

冲突不命中 放置策略导致

Block b is fetched from *memory*

Block b is stored in cache

•Placement policy:（放置策略）
determines where b goes

•Replacement policy:（替换策略）
determines which block
gets evicted (victim)

存储体系的性能

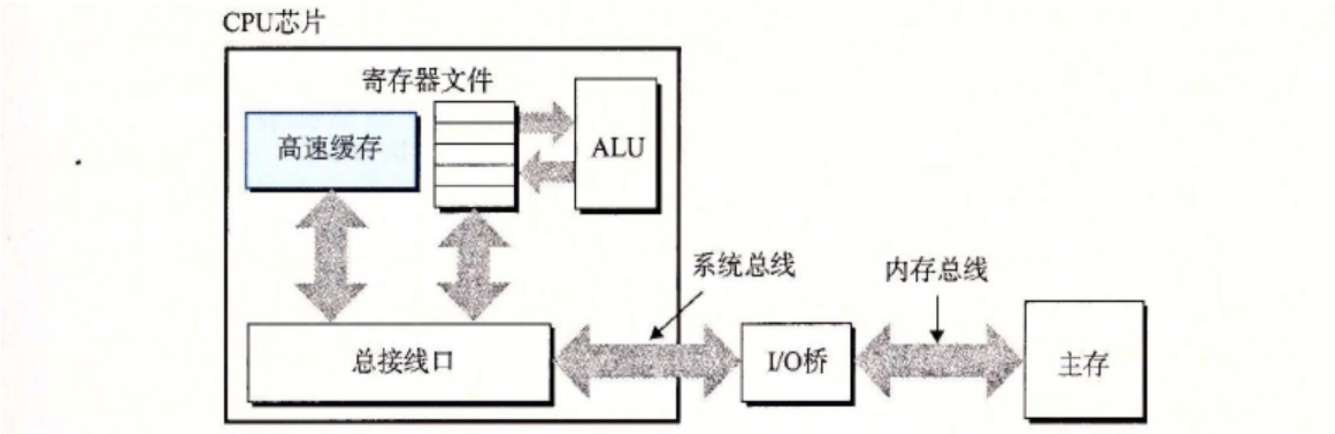
- 缓存命中：当需要M_{i+1}中的一个数据对象时，先在M_i中寻找，找到称为缓存命中，反之称为缓存失效/不命中
- 缓存命中率：在M_i层的命中率h_i是数据对象可以在M_i中找到的概率

- 存储体系性能：由层次结构的有效存取时间T_{eff}决定，依赖于相继层次的命中率和访问频率（f_i，每当发生失效，就要付出代价去访问更低层）

$$T_{eff} = \sum_{i=1}^{n} f_i t_i = h_1 t_1 + (1-h_1) h_2 t_2 + \cdots$$
$$+ (1-h_1)(1-h_2)\cdots(1-h_{n-1}) h_n t_n$$

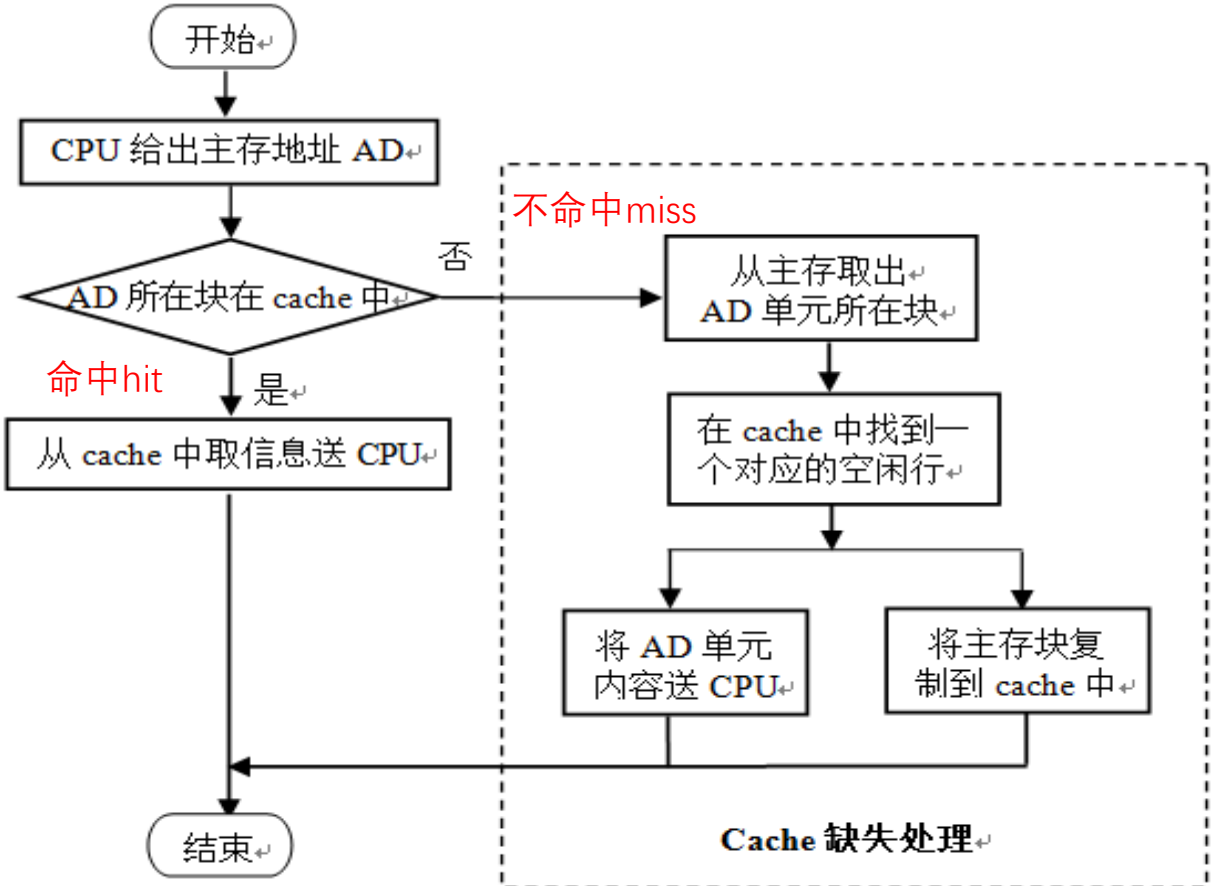
- 存储体系设计目标：使T_{eff}接近t1，使成本接近M_n

高速缓存存储器的典型总线结构



- Cache是一种小容量高速缓冲存储器，由SRAM组成。
- Cache直接制作在CPU芯片内，速度几乎与CPU/寄存器一样快。
- 程序运行时，CPU使用的一部分数据/指令会预先成批拷贝在Cache中，Cache的内容是主存储器中部分内容的副本。
- 当CPU需要从内存读(写)数据或指令时，先检查Cache，若有，就直接从Cache中读取，而不用访问主存储器。

Cache操作过程


开始
CPU 给出主存地址 AD
AD 所在块在 cache 中
否
命中hit
是
从 cache 中取信息送 CPU
结束
不命中miss
从主存取出
AD 单元所在块
在 cache 中找到一
个对应的空闲行
将 AD 单元
内容送 CPU
将主存块复
制到 cache 中
Cache 缺失处理

Cache实现

- Cache对程序员（编译器）是否透明?
  - 是透明的，程序员（编译器）在编写高级语言程序（生成低级语言程序）时无需了解Cache是否存在或如何设置
  - 了解Cache有助于编写（编译）高效的程序
  - Cache的思想也被应用于编程语言设计与实现（尤其是有自动垃圾收集机制的语言）

Cache实现

- 实现Cache机制需要解决哪些问题？
  1. 如何分块？
  2. 主存块与Cache之间如何映射？（放置策略）
  3. Cache已满时，怎么办？（替换策略）
  4. 写数据时怎样保证Cache和主存的一致性？
  5. 如何根据主存地址访问到Cache中的数据？

  …

Cache映射（Mapping）

- 主存被分成若干大小相同的块，称为主存块（Block）
- Cache也被分成相同大小的块，称为Cache行（line）或槽（Slot），每个Cache槽放一个主存块

- 放置策略
  - 直接（Direct）：每个主存块映射到Cache的固定行
  - 全相联（Full Associate）：每个主存块映射到Cache的任一行
  - 组相联（Set Associate）：每个主存块映射到Cache固定组中的任一行

直接映射Cache

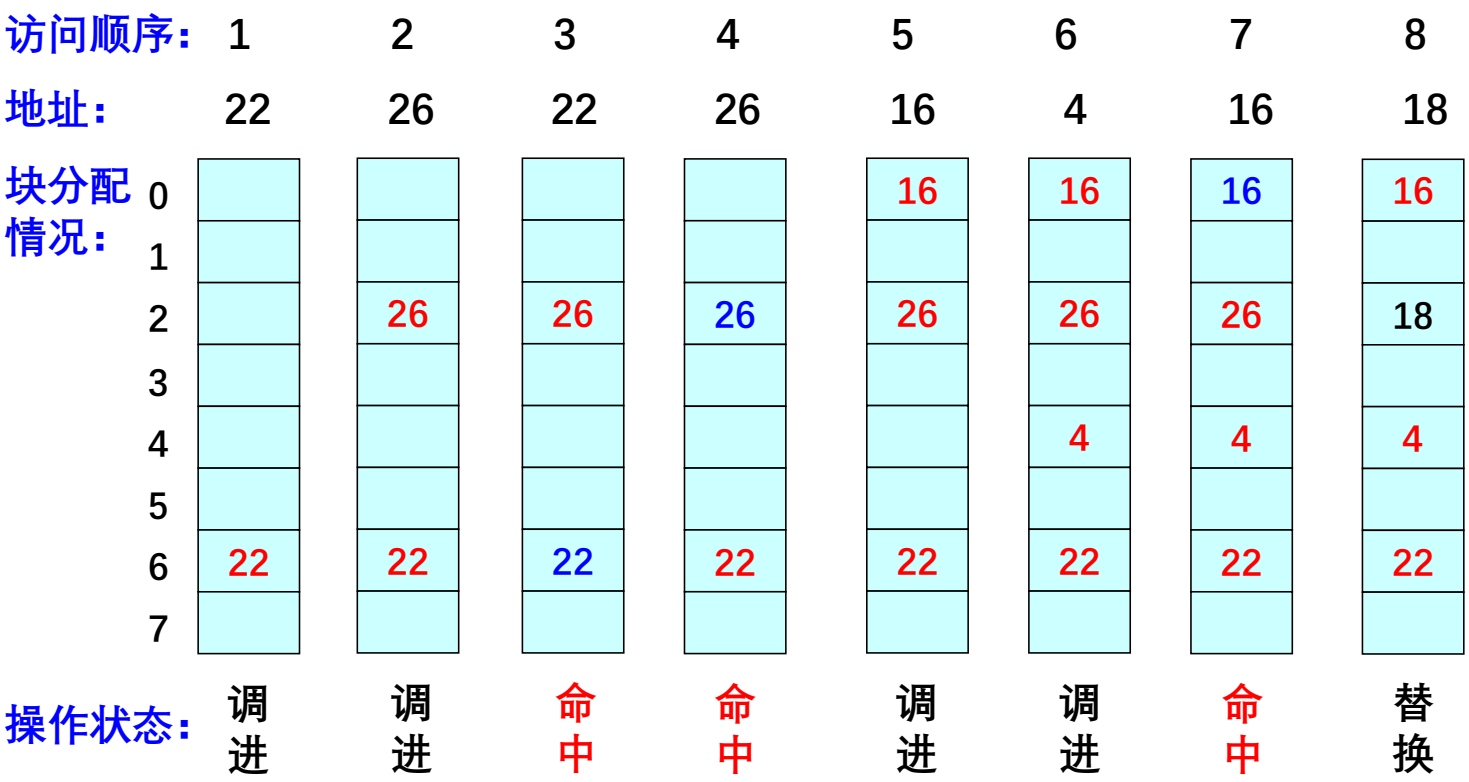
- 主存每一块只能映射到固定的Cache行（槽）
- 也称模映射(Module Mapping)
- 映射关系为：

  Cache行号 = 主存块号　mod　Cache行数
  - 4 = 100 mod 16 （假定Cache共有16行）

  即主存第100块映射到Cache的第4行中

- 特点
  - 容易实现
  - 无需考虑替换问题，和放置同策略
  - 不够灵活，Cache存储空间得不到充分利用，命中率低
    - 例如：需将主存第0块与第16块同时复制到Cache中时，由于它们都只能复制到Cache第0行，即使Cache其它行空闲，也有一个主存块不能写入Cache，导致频繁的Cache装入

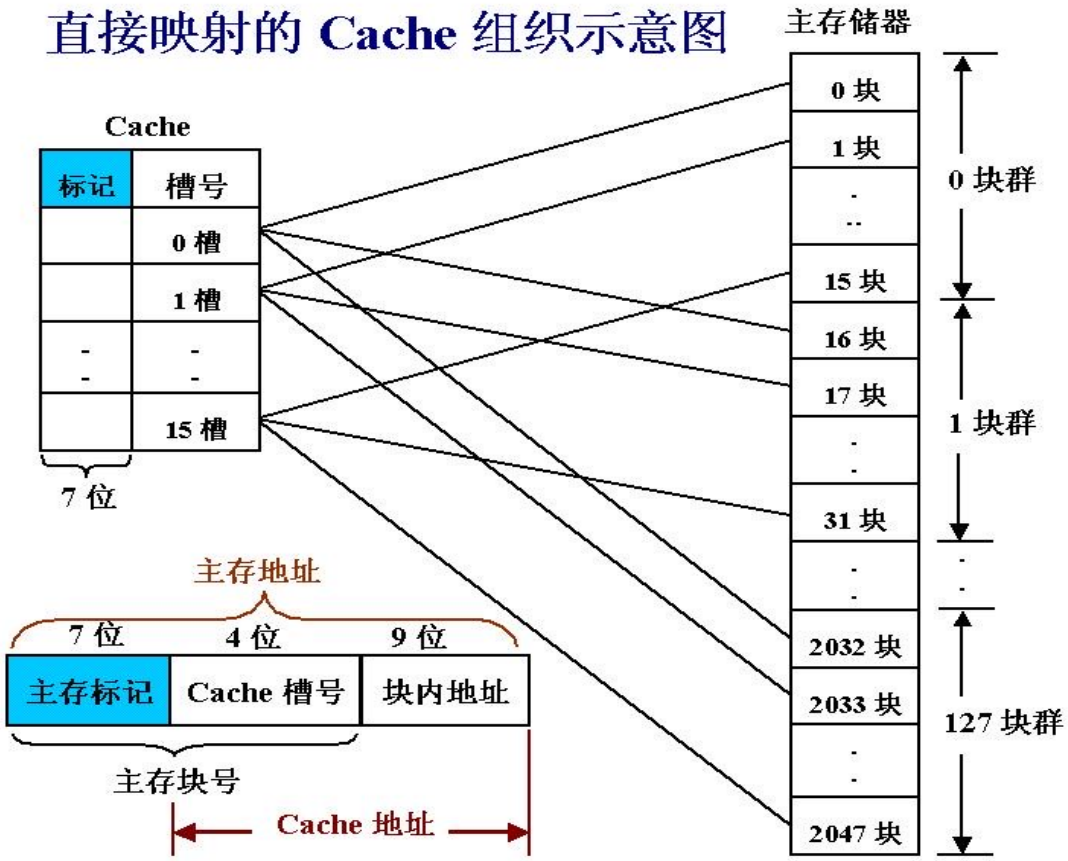
- 例：设一个cache中有8个块，访问主存进行读操作的块号序列为22、26、22、26、16、4、16、18，求直接映射下每次访问后cache中的内容。



直接映射Cache

- 主存　　9位
  - 1024KB=512B*2048块
- Cache
  - 8KB=512B*16行

- 2048/16=128块群　　7位
  - 主存标记=块群号
- 槽号=块号 mod 16　4位
- 块内地址：数据对象在内存块中的偏移

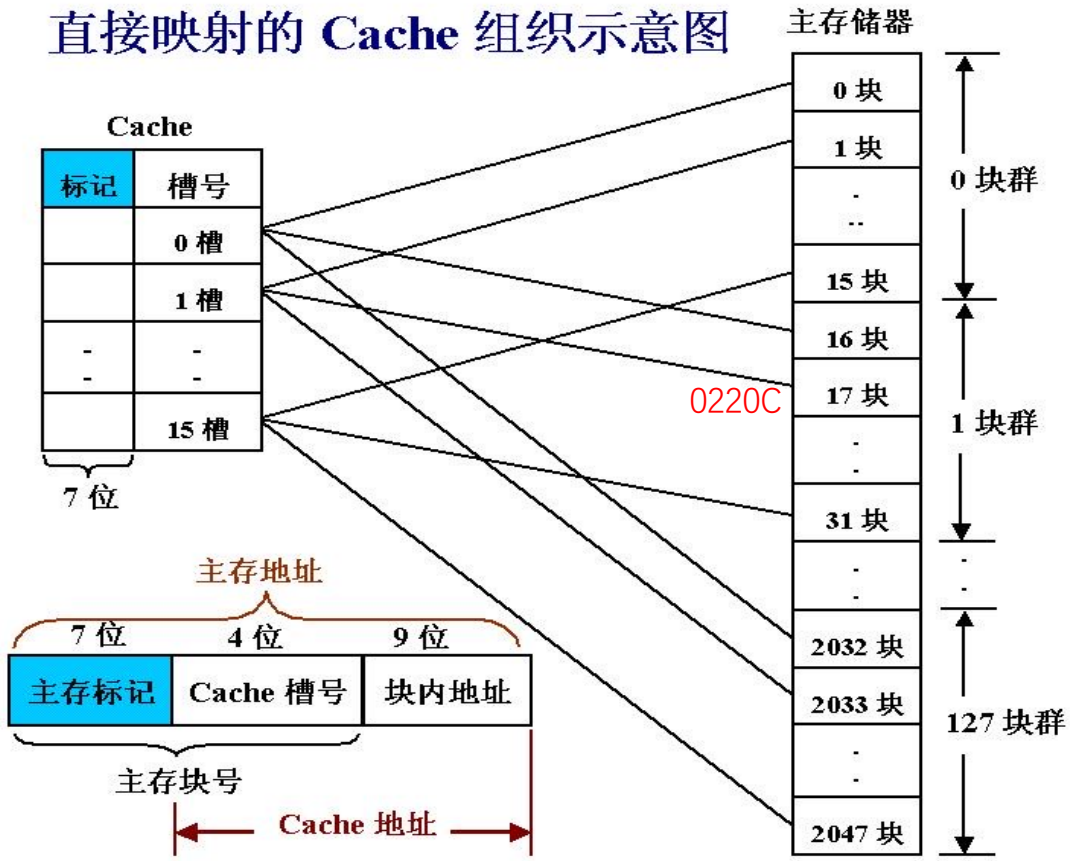


直接映射Cache

- 主存　　9位
  - 1024KB=512B*2048块
- Cache
  - 8KB=512B*16行

- 2048/16=128块群　　7位
  - 主存标记=块群号
- 槽号=块号 mod 16　　4位
- 块内地址：数据对象在内存块中的偏移

- 如何访问主存0220C单元
  - 0220C=0000 0010 0010 0000 1100
    块群号=1　槽号=1　块内地址=12
    主存块号=17



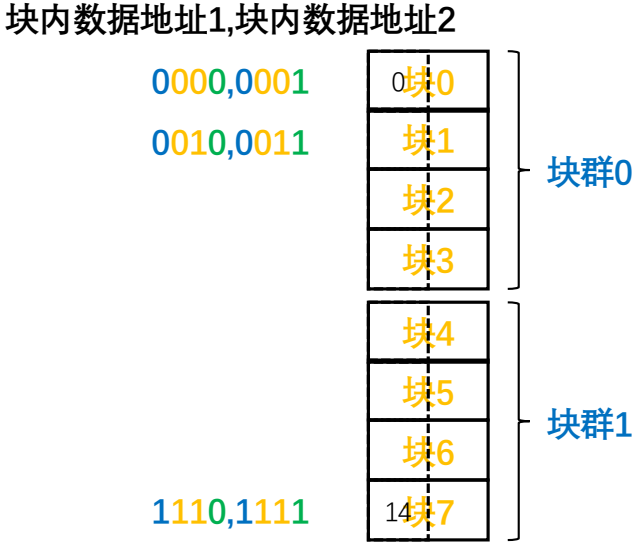
直接映射Cache

- Cache行=有效位（Valid Bit，V）+标记位（Tag，块群号）+高速缓存块（Data）
  - V为1表示信息有效，为0表示信息无效
  - 开机或复位时，使所有行的有效位V=0
  - 某行被替换后使其V=1
  - 某行装入新块时使其V=1
  - 通过使V=0来冲刷cache（例如：进程切换时，DMA传送时）
  - 通常为操作系统设置“cache冲刷”指令，因此cache对操作系统不是透明的

直接映射Cache

- 例：高速缓存有4行，每个块2个字节 地址是4位的

地址（十进制）	地址位			块号（十进制）
	标记位（t=1）	索引位（s=2）	偏移位（b=1）	
0	0	00	0	0
1	0	00	1	0
2	0	01	0	1
3	0	01	1	1
4	0	10	0	2
5	0	10	1	2
6	0	11	0	3
7	0	11	1	3
8	1	00	0	4
9	1	00	1	4
10	1	01	0	5
11	1	01	1	5
12	1	10	0	6
13	1	10	1	6
14	1	11	0	7
15	1	11	1	7



直接映射Cache

- 例：高速缓存有4行，每个块2个字节　地址是4位的

读地址0(0000)的字

冷不命中
调进

行	有效位	标记位	偏	移
00	1	0	m(0)	m(1)
01	0			
10	0			
11	0			

读地址1(0001)的字

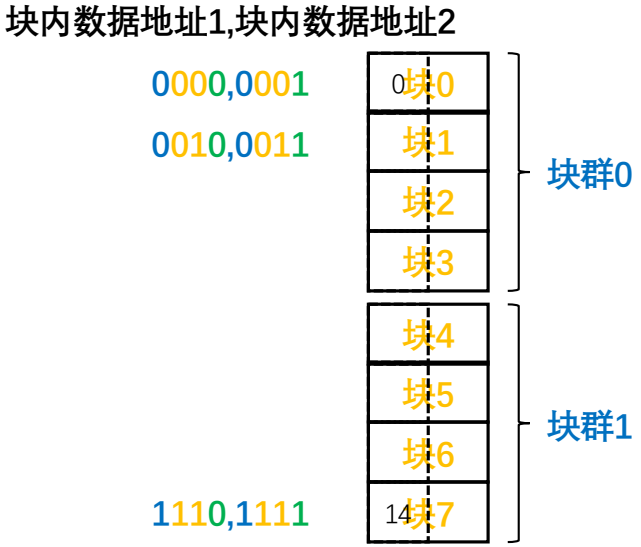
命中

行	有效位	标记位	偏	移
00	1	0	m(0)	m(1)
01	0			
10	0			
11	0			

读地址13(1101)的字

冷不命中
调进

行	有效位	标记位	偏	移
00	1	0	m(0)	m(1)
01	0			
10	1	1	m(12)	m(13)
11	0			



直接映射Cache

- 例：高速缓存有4行，每个块2个字节　地址是4位的

读地址8(1000)的字

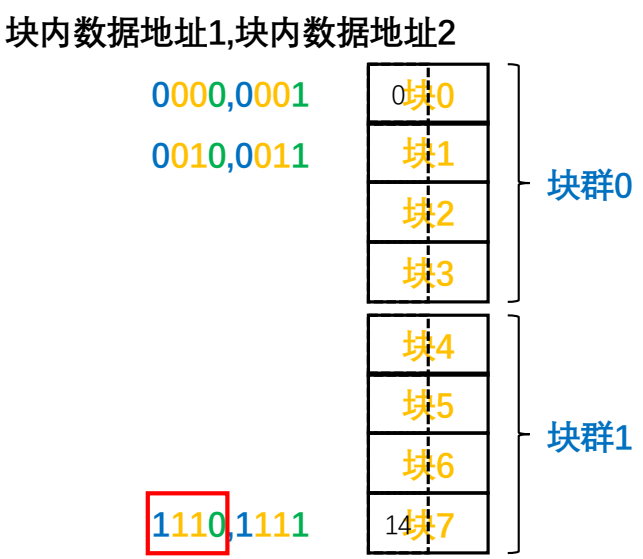
冲突不命中
替换

行	有效位	标记位	偏	移
00	1	1	m(8)	m(9)
01	0			
10	1	1	m(12)	m(13)
11	0			

读地址0(0000)的字

冲突不命中
替换

行	有效位	标记位	偏	移
00	1	0	m(0)	m(1)
01	0			
10	1	1	m(12)	m(13)
11	0			



主存地址4位：1110

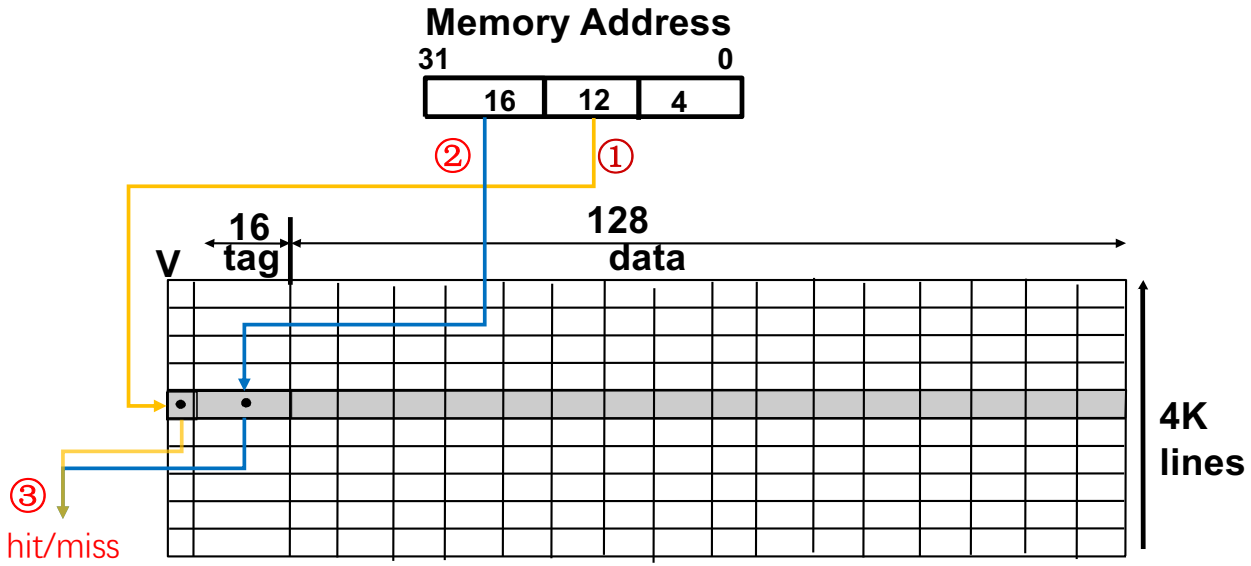
主存块号3位：111（块是传送单位，判断是否命中）

cache地址3位：110 （不同块群号同一取模块号，不能同时存在）

按字节编址，每个地址内的数据字8位

块内偏移地址需要4位　　64KB/16B=4K行，12位

- 例：主存和Cache之间直接映射，块大小为16B。Cache的数据区容量为64KB，主存地址为32位，按字节编址。问：1. 说明主存地址如何划分，2. 访存过程，3. Cache总容量。



Cache容量= 4Kx(1b+16b)+64KB=580Kb=72.5KB
数据占64KB/72.5KB=88.3%

- 例：Cache为64行，块大小为16B，主存地址（字节编址，十进制索引）1200对应的Cache行号为？

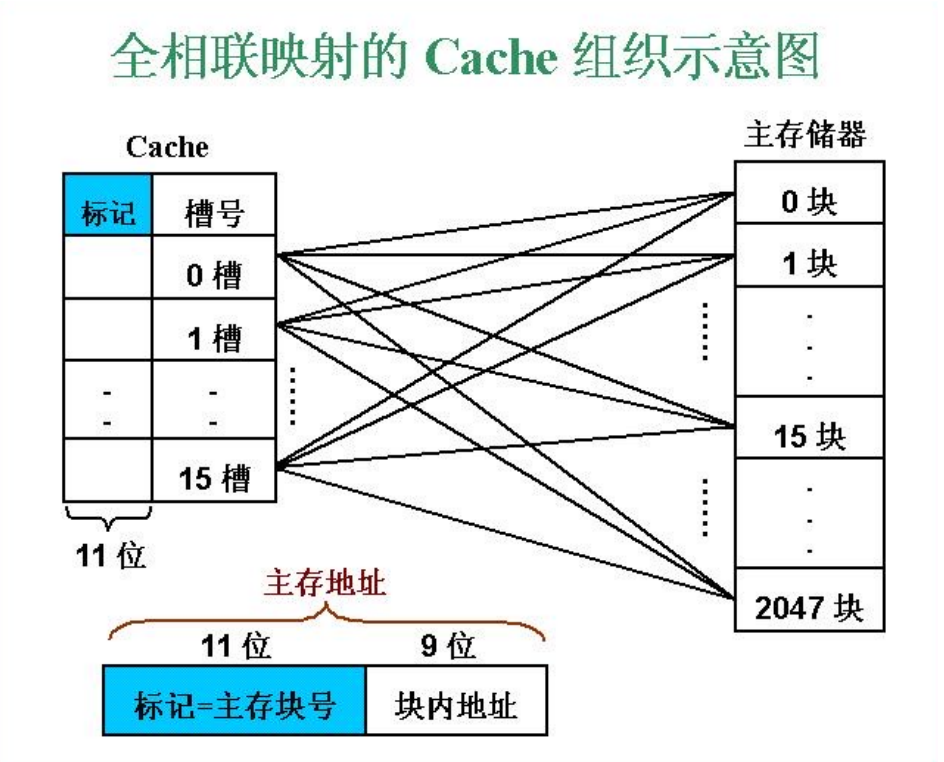
方法一：
1200 / 16 = 75，即放满75个块
75 mod 64 = 11（商1），即 块群号=1，Cache行号（块群内块号）=11

方法二：
块大小16B，块内地址偏移4位
Cache行数64，Cache行号6位
1200 = 100 1011 0000，行号=001011=11

全相联映射Cache

- 主存　　9位　　11位
  - 1024KB=512B*2048块
- Cache
  - 8KB=512B*16行

- 每个主存块可以装入任一cache行
  - Cache标记（tag）指出行对应的块
  - 主存标记即主存块号（不分块群）
  - 地址无槽号，存取时比较标记

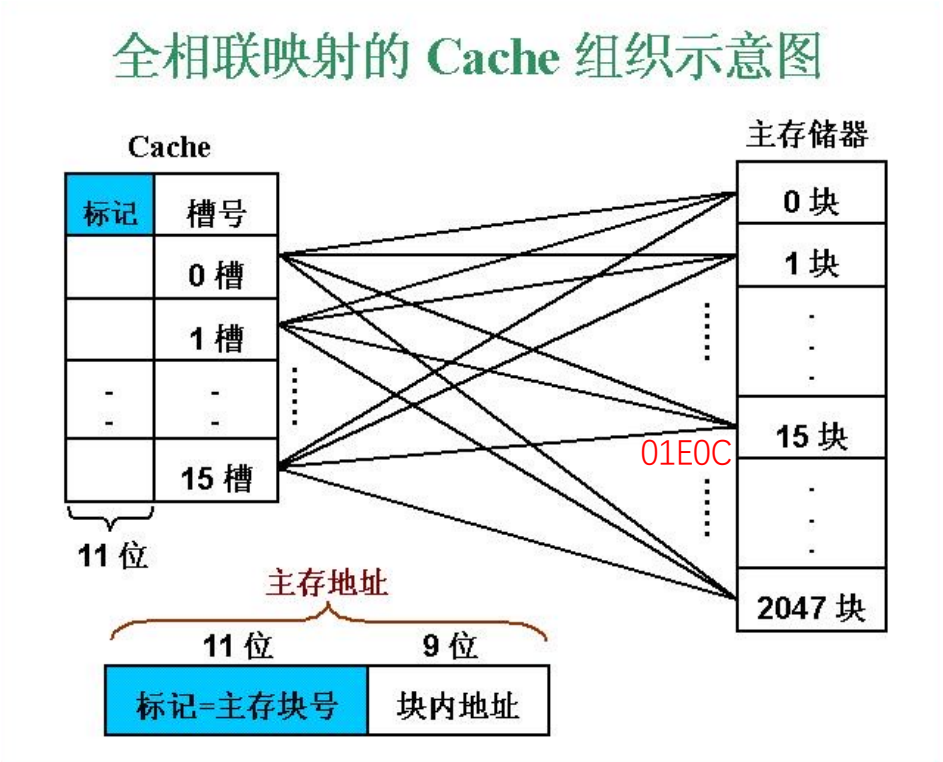


全相联映射Cache

- 主存　　9位　　11位
  - 1024KB=512B*2048块
- Cache
  - 8KB=512B*16行

- 每个主存块可以装入任一cache行
  - Cache标记（tag）指出行对应的块
  - 主存标记即主存块号（不分块群）
  - 地址无槽号，存取时比较标记

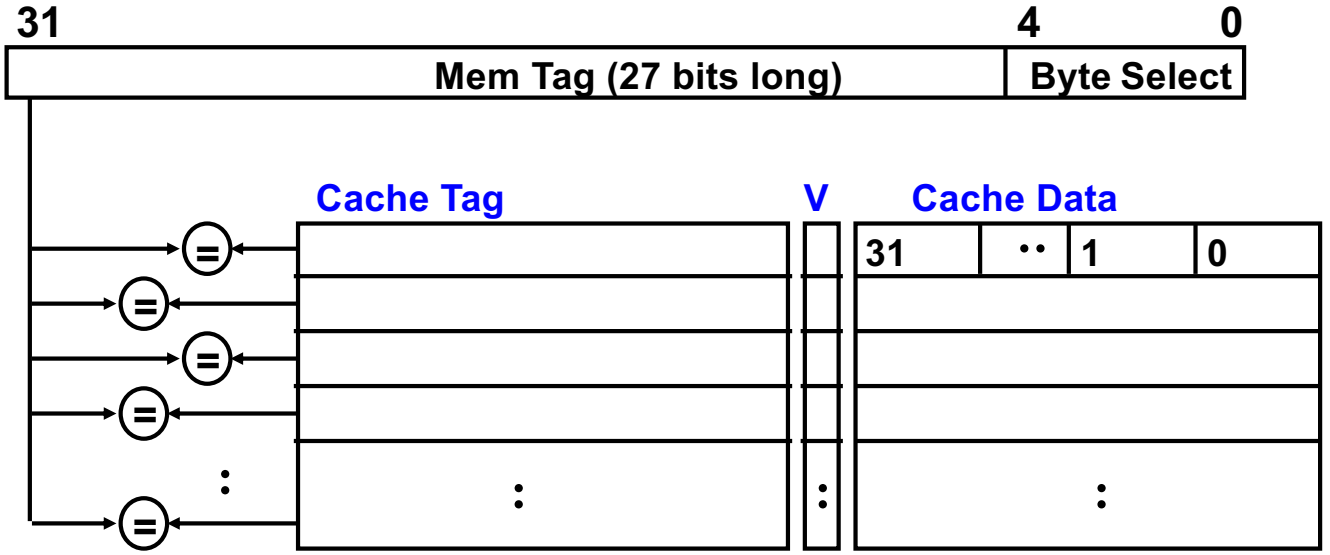
- 如何访问主存01E0C单元
  - 01E0C=0000 0001 1110 0000 1100
    块号=15　　块内地址=12



全相联映射Cache

- Cache行=有效位（V）+标记位（tag）+高速缓存块（Data）

- 优点：没有冲突不命中
  - 不会出现Cache行有空闲但不命中
- 缺点：并行比较，需要cache行数相同的比较器（最多只有1个相同）
  - 昂贵，只适用于小而快的cache（如虚拟内存的快表TLB）

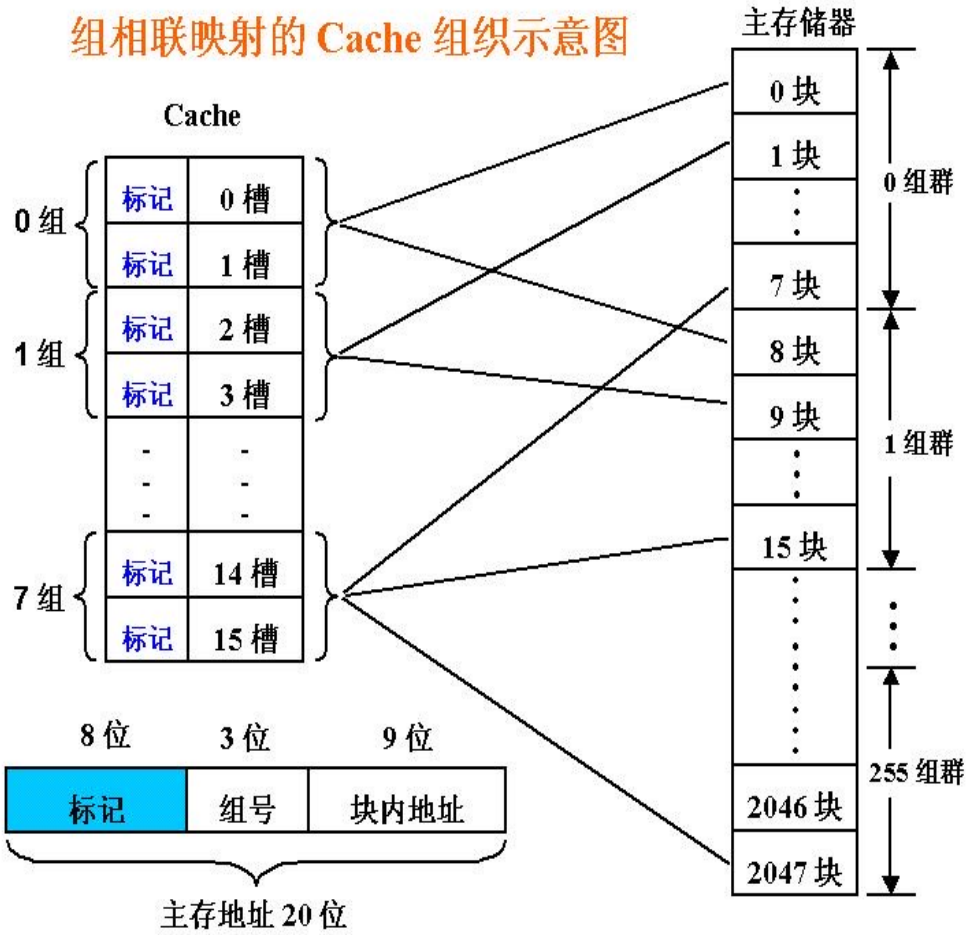


组相联映射Cache

- Cache行分组，主存块映射到固定组中任一行
  - 组间模映射，组内全相联
    - Cache组号 = 主存块号 mod Cache组数
      - E.g., Cache=8KB=8组*2行/组*512B/行
        - 4 = 100 mod 8，主存第100块映射到Cache第4组的任意行中
  - 结合直接映射和全相联映射的优点
    - 当Cache组数为1时，变成全相联映射
    - 当每组只有一个Cache行时，变成直接映射
  - 每组2或4行较为常见（称为2路或4路组相联）
  - 在大容量的L2/L3 Cache中会出现4路以上组相联

组相联映射Cache

- 主存　　9位
  - 1024KB=512B*2048块
- Cache　　3位
  - 8KB=512B*16行=512B*8组*2行/组
- 2048/8=256组群　8位
  - 标记=组群号
- 同一Cache组不同行一定来自不同内存组群，取出Cache组后比较标记确定哪一组群

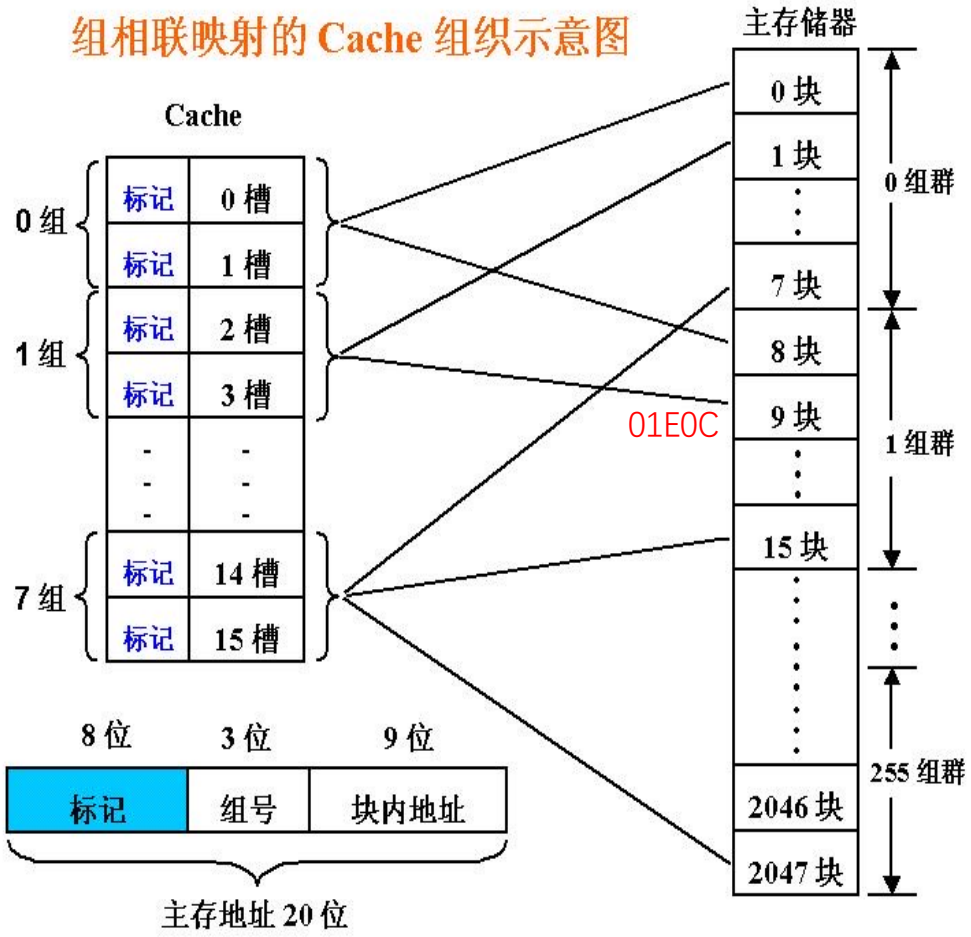


组相联映射Cache

- 主存　　9位
  - 1024KB=512B*2048块
- Cache　　3位
  - 8KB=512B*16行=512B*8组*2行/组

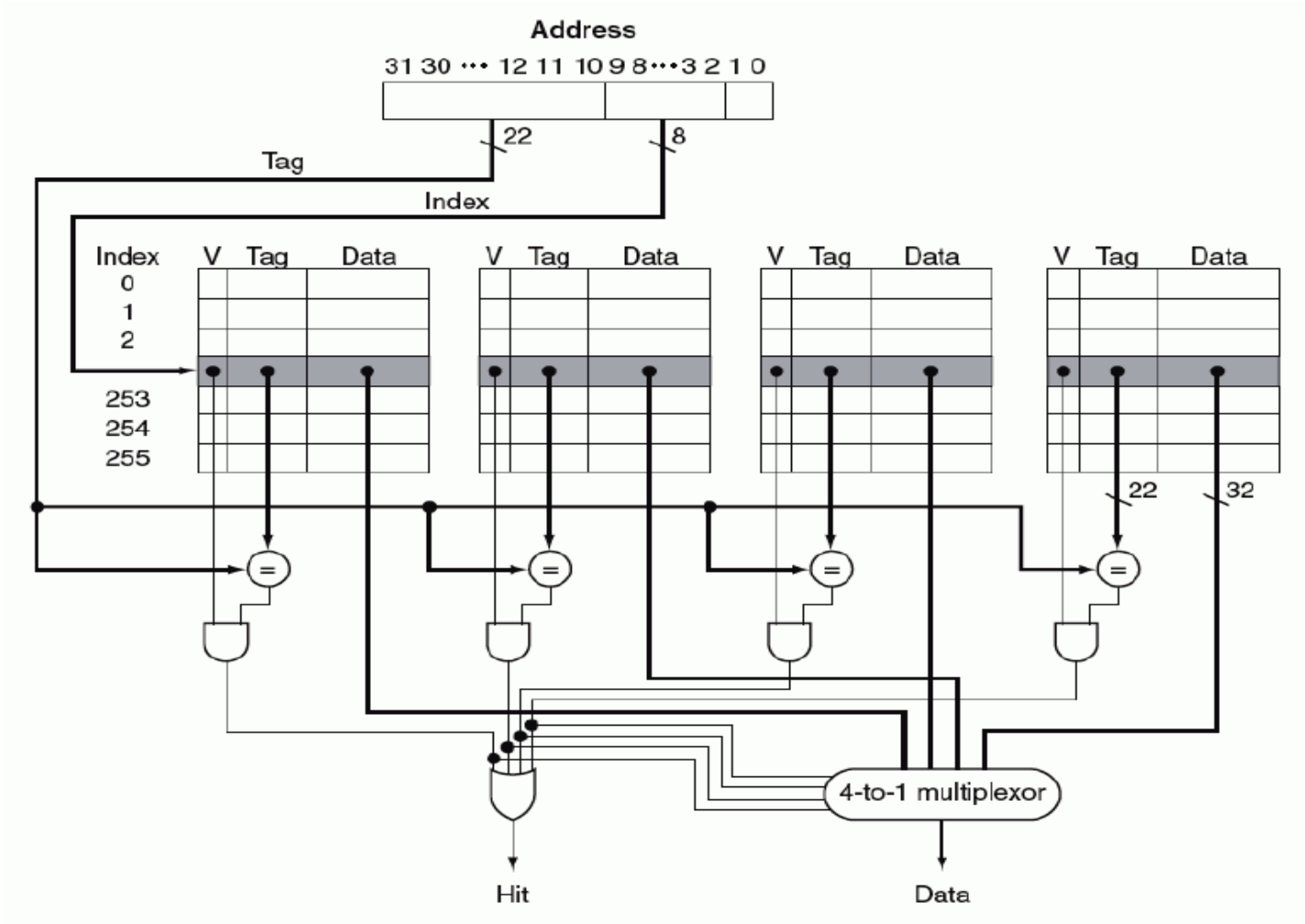
- 2048/8=256组群　8位
  - 标记=组群号
- 同一Cache组不同行一定来自不同内存组群，取出Cache组后比较标记确定哪一组群

- 如何访问主存01E0C单元
  - 01E0C=0000 0001 0010 0000 1100
    组群号=1　组号=1　块内地址=12
    主存块号=9



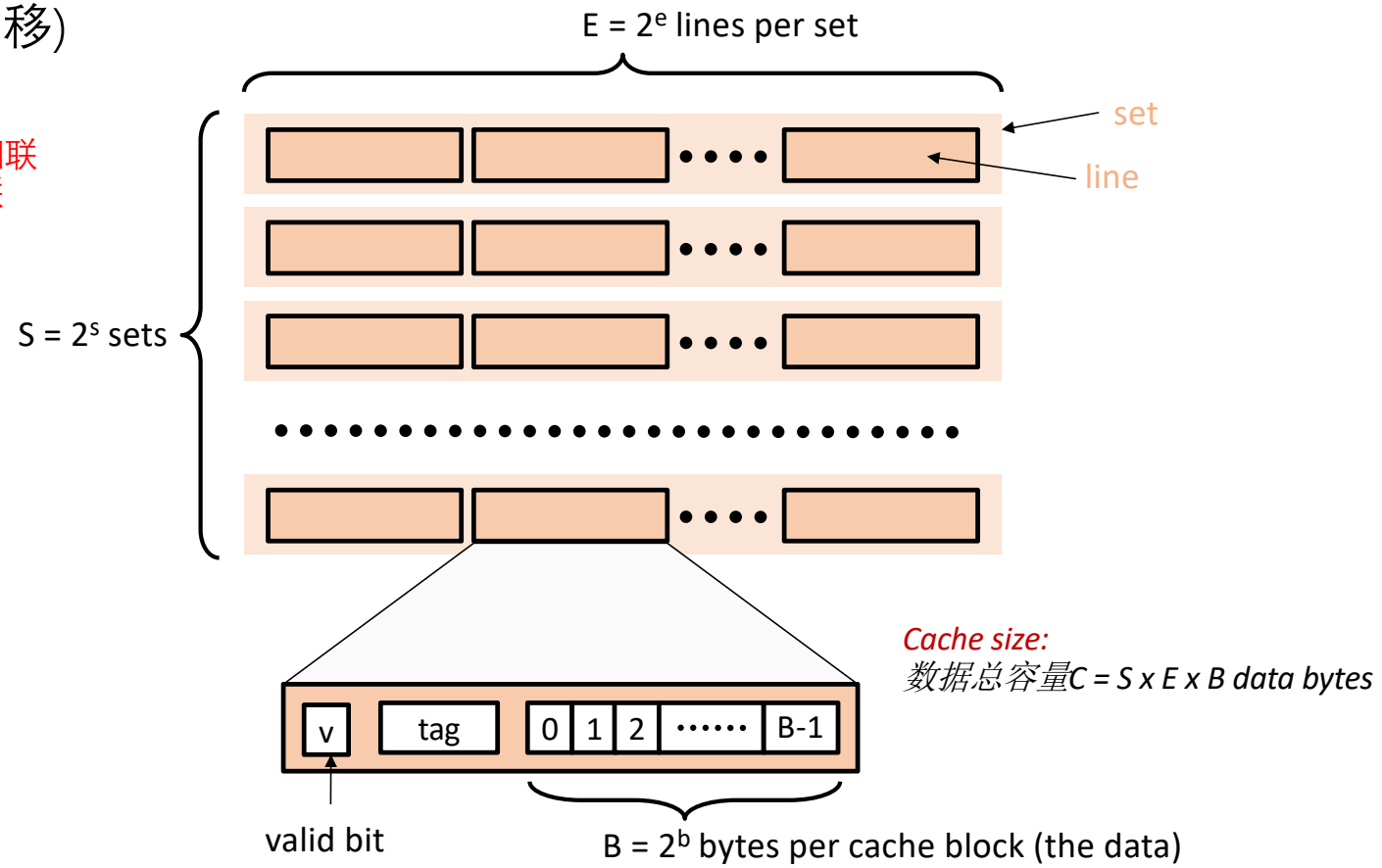
组相联映射Cache

- 标记22位，内存2^{22}组群
- 组号8位，cache 2^8组
- 块内地址2位，内存块2^2B=32b
- 每组4行（4个与门实现比较器，最多只有一个输出为1）



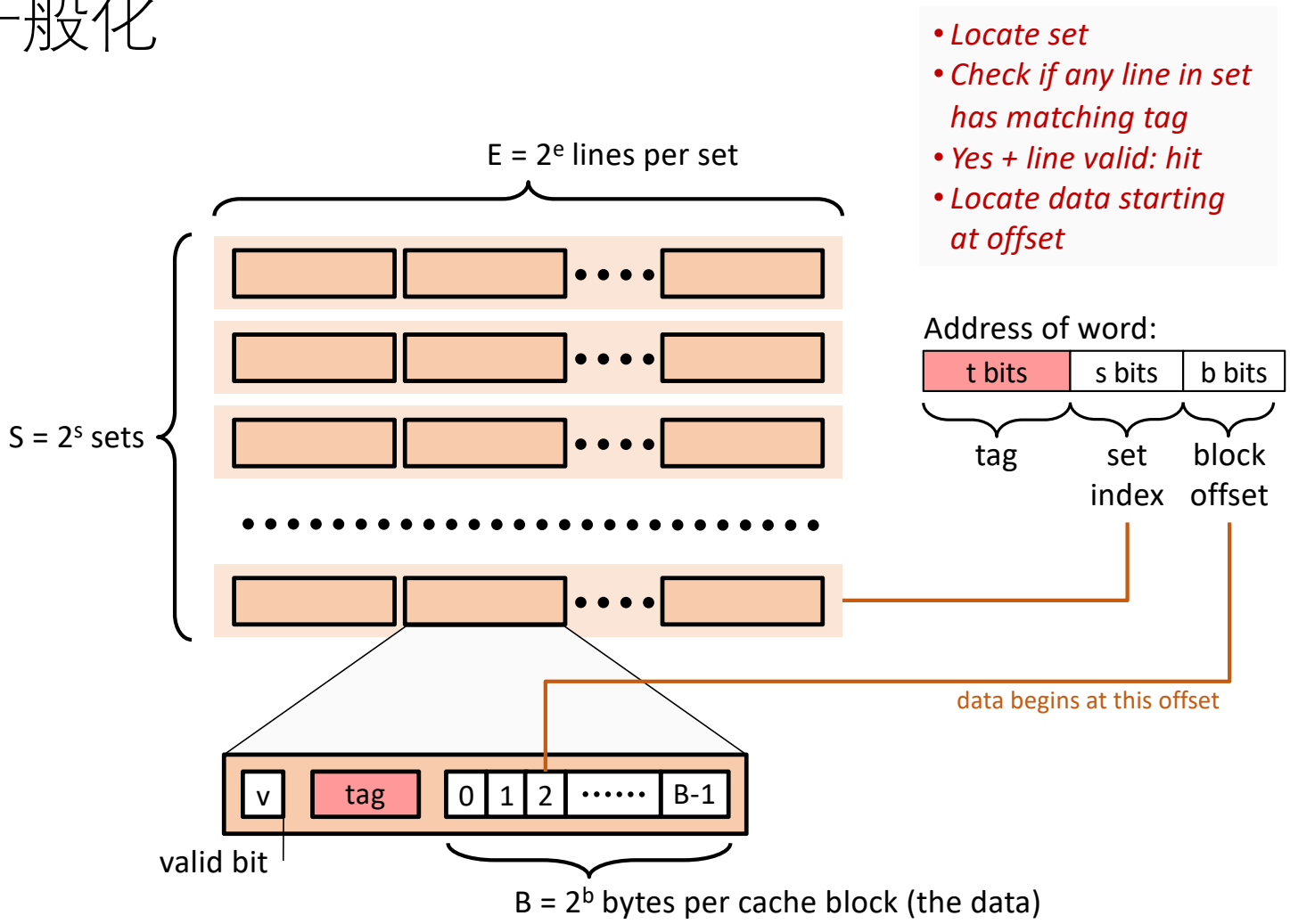
Cache组织的一般化

- (S,E,B)=(组,行,偏移)
  - E=1：直接映射
  - E=C/B（S=1）：全相联
  - 1<E<C/B：E路组相联



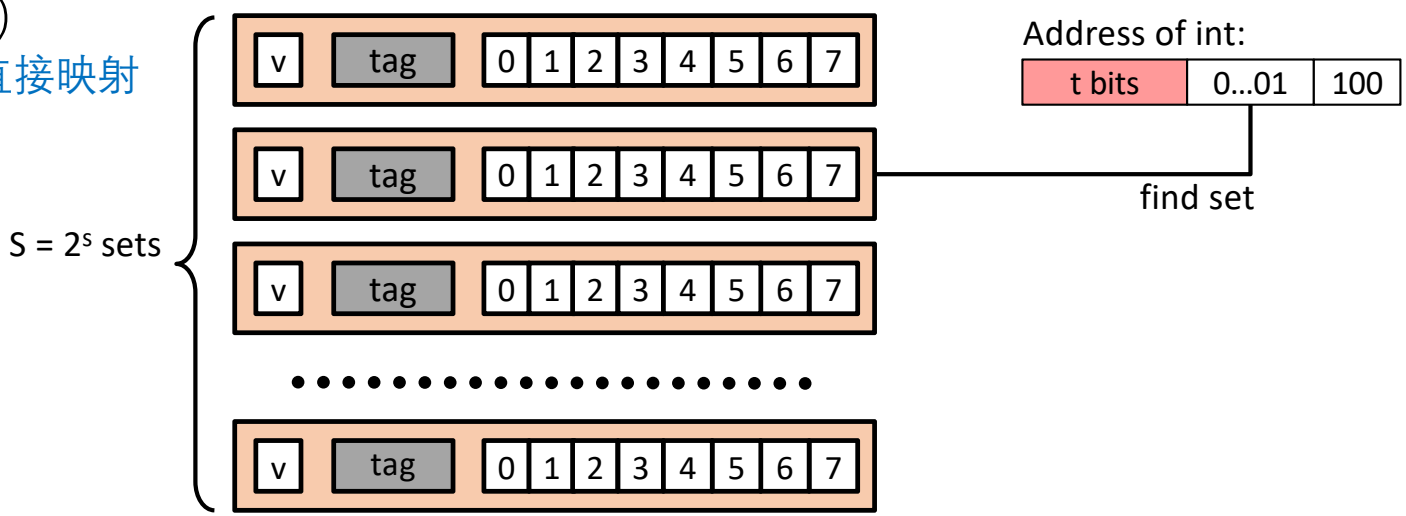
Cache组织的一般化

- Cache读



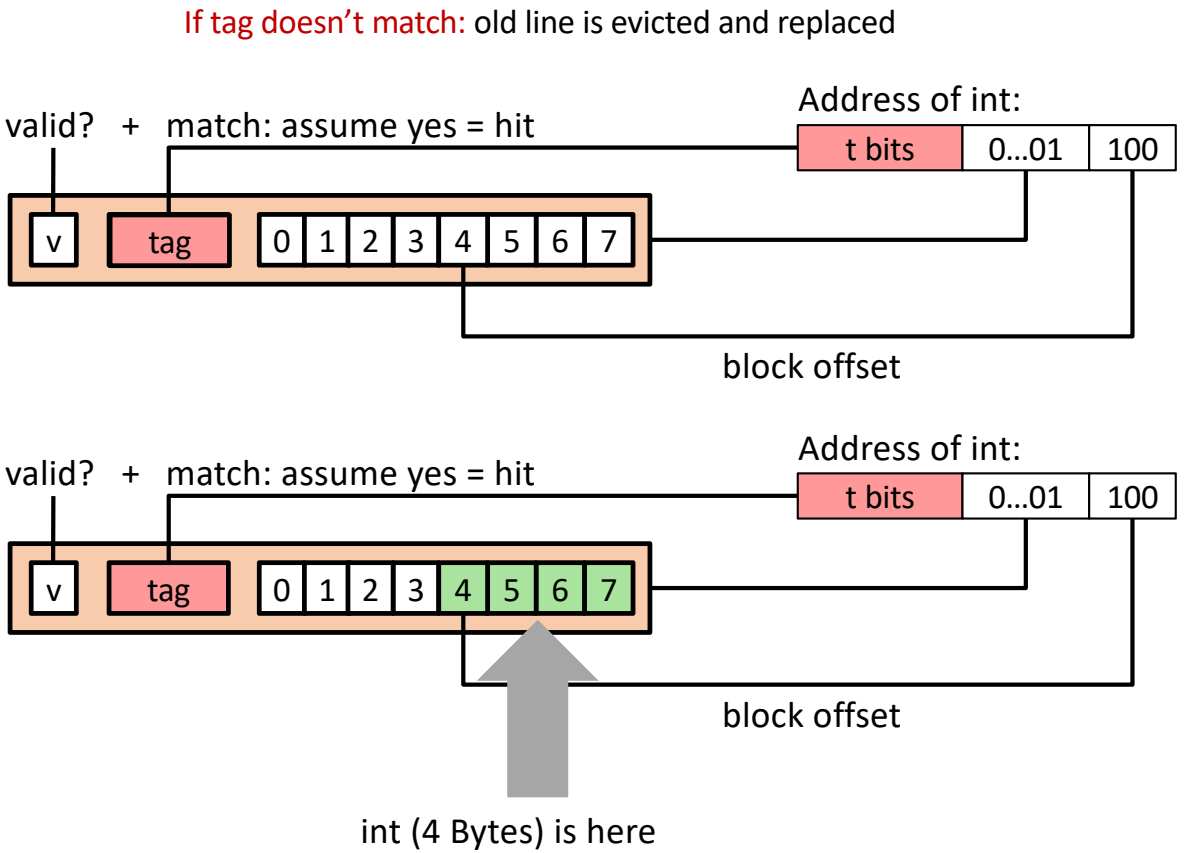
Cache组织的一般化

- (S,E,B)=(组,行,偏移)
  - E=1，每组1行，直接映射
  - 假设B=8
- 读int变量



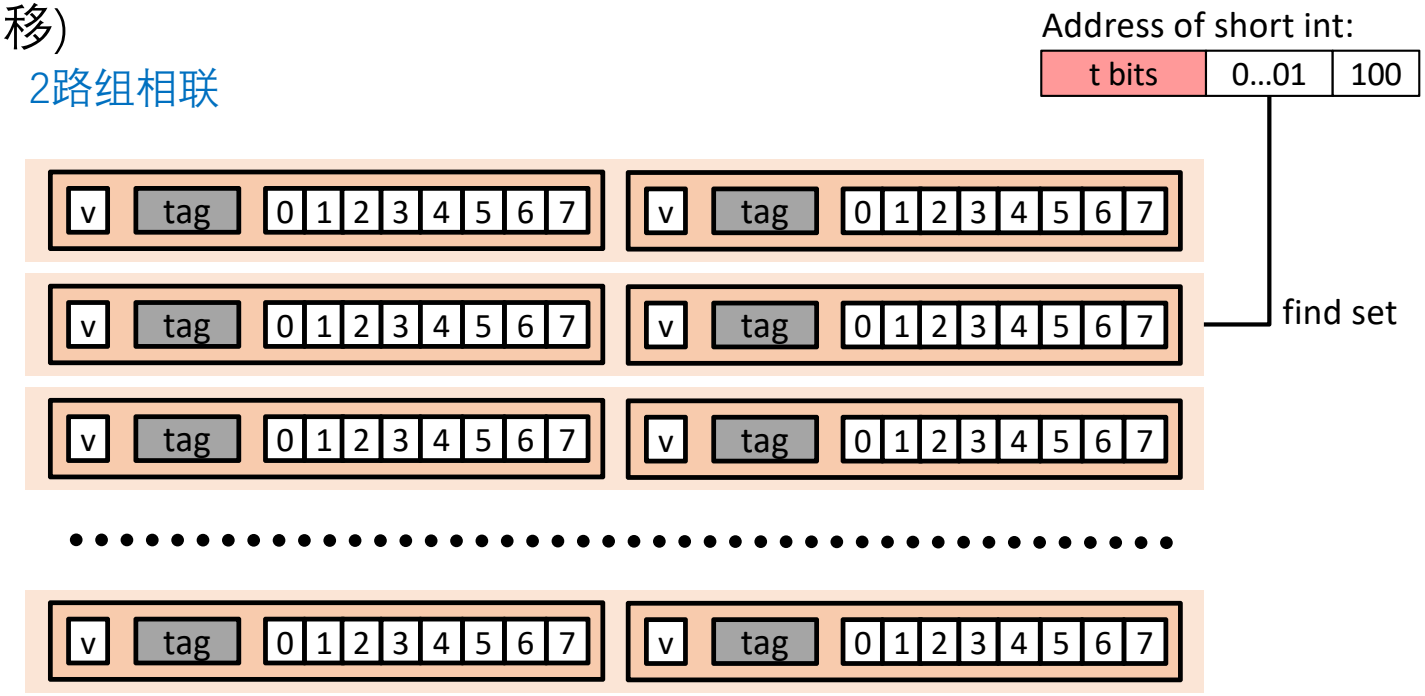
Cache组织的一般化

- (S,E,B)=(组,行,偏移)
  - E=1，每组1行，直接映射
  - 假设B=8
- 读int变量



Cache组织的一般化

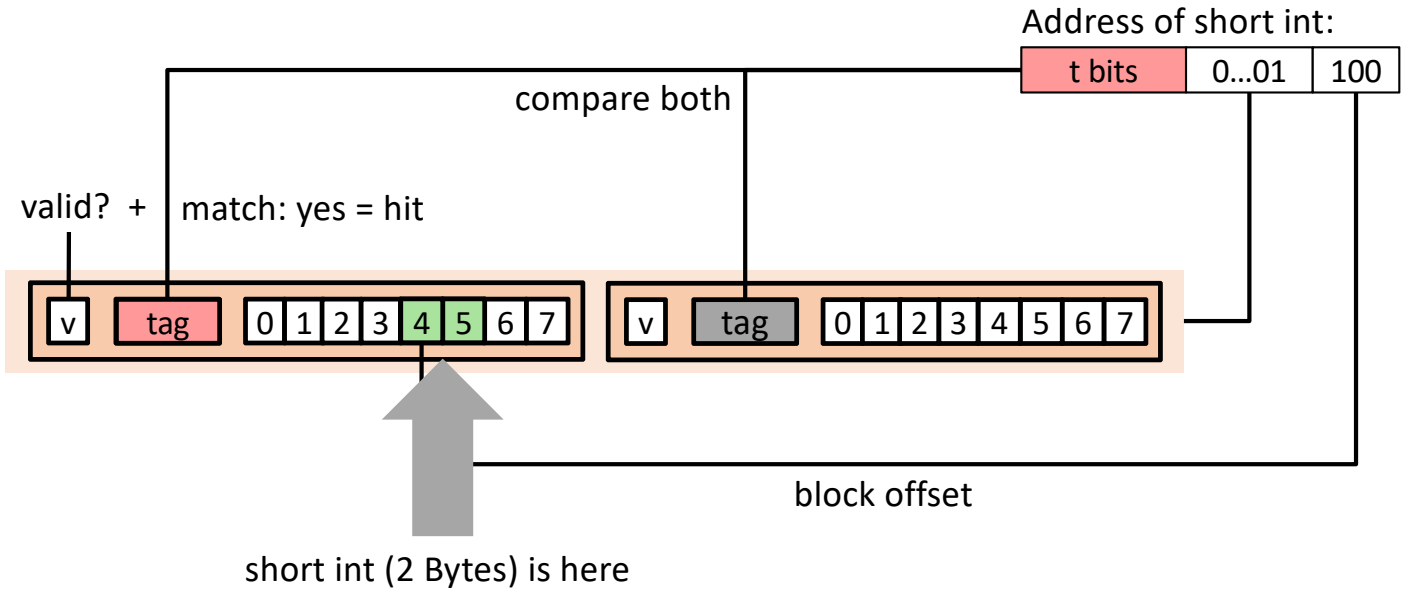
- (S,E,B)=(组,行,偏移)
  - E=2，每组2行，2路组相联
  - 假设B=8
- 读short变量



Cache组织的一般化

- (S,E,B)=(组,行,偏移)
  - E=2，每组2行，2路组相联
  - 假设B=8
- 读short变量

No match:
•One line in set is selected for eviction and replacement
•Replacement policies: random, least recently used (LRU), …



Cache组织的一般化

- (S,E,B)=(组,行,偏移)
- S=2组
- E=2行/组
- B=2字节/内存块

- 内存地址：

t=2	s=1	b=1
xx	x	x

  - 内存块数=组群数*Cache组数（组群内内存块数）
    $=2^t*S=8$
  - 地址数（字节编址）=内存块数*内存块字节大小
    =8*2=16（4位编码）

	v	Tag	Block
Set 0	1	00	M[0-1]
	1	10	M[8-9]
Set 1	1	01	M[6-7]
	0		

地址访问序列（每次读1B）		Cache命中情况
0	0000	miss
1	0001	hit
7	0111	miss
8	1000	miss
0	0000	hit

教材，练习题6.9

下表给出了几个不同的高速缓存的参数。确定每个高速缓存的高速缓存组数(S)、标记位数(t)、组索引位数(s)以及块偏移位数(b)。

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

C：cache数据总量，C=B*E*S

B：内存块大小（字节），b：块内地址位数，$B=2^b$

E：每组行数

S：组数，s：组索引位数，$S=2^s$

m：主存位数，2^m：主存地址数，m=(t,s,b)，t：标记位位数

- **直接映射，E=1，m=(主存块群号,cache行号,块内地址)**
- **全相联，E=C/B，S=1，m=(主存块号,∅,块内地址)**
- **组相联，1<E<C/B，m=(主存组群号,cache组号,块内地址)**

三种cache映射方式的比较

- 关联度：一个主存块 在某种映射方式下 能映射到的cache行数
  - 直接映射，关联度=1
  - 全相联：关联度=cache行数
  - 组相联：关联度=每个cache组的行数
  - 关联度=E（每组行数）
- cache大小和主存块大小一定时
  - 关联度越低，命中率越低
  - 关联度越低，判断是否命中的电路开销越小，电路延迟越小
  - 关联度越低，每个cache行中标记所占额外空间越小
- 假设主存地址32位，按字节编址，主存块大小为16B，cache有2^{12}行，则
  - 关联度=1，直接映射：每组1行，共4K组，标记占32-4-12=16位，标记总位数=4K*16=64K
  - 关联度=2，2路组相联：每组2行，共2K组，标记占32-4-11=17位，标记总位数=4K*17=68K
  - 关联度=4，4路组相联：每组4行，共1K组，标记占32-4-10=18位，标记总位数=4K*18=72K
  - 关联度=4K，全相联：每组4K行，共1组，标记占32-4-0=28位，标记总位数=4K*28=112K

例：一个组相联cache由64个行组成，每组4行。主存包含4K个块，每块128字，内存是字寻址。请表示内存地址的格式。

解：

- 128=2^7，块内地址位数b=7
- E=4，S=64/4=16，组索引位数s=4
- m=(t,s,b)，其中(t,s)确定一个主存块，共4k主存块需12位，标记（组群号）位数t=12-4=8
  或：4K主存块，每16个划分一个组群，同一组群中每一个块映射到不同的cache组，组群数=4K/16=2^8

标记t	组号s	块偏移
8位	4位	7位

- 主存地址19位，即主存寻址单元数2^{19}，每个单元1字（1字多少字节取决于机器字长/CPU数据通路的宽度，x86-64/ARMv8即64位8字节）

单选题 1分

设置

某计算机的Cache共有16行，采用2路组相联映射方式。每个主存块大小为32字节，按字节编址。主存129号单元所在主存块应装入到的Cache组号是（　　）。

A 0

B 2

C 4

D 6

提交

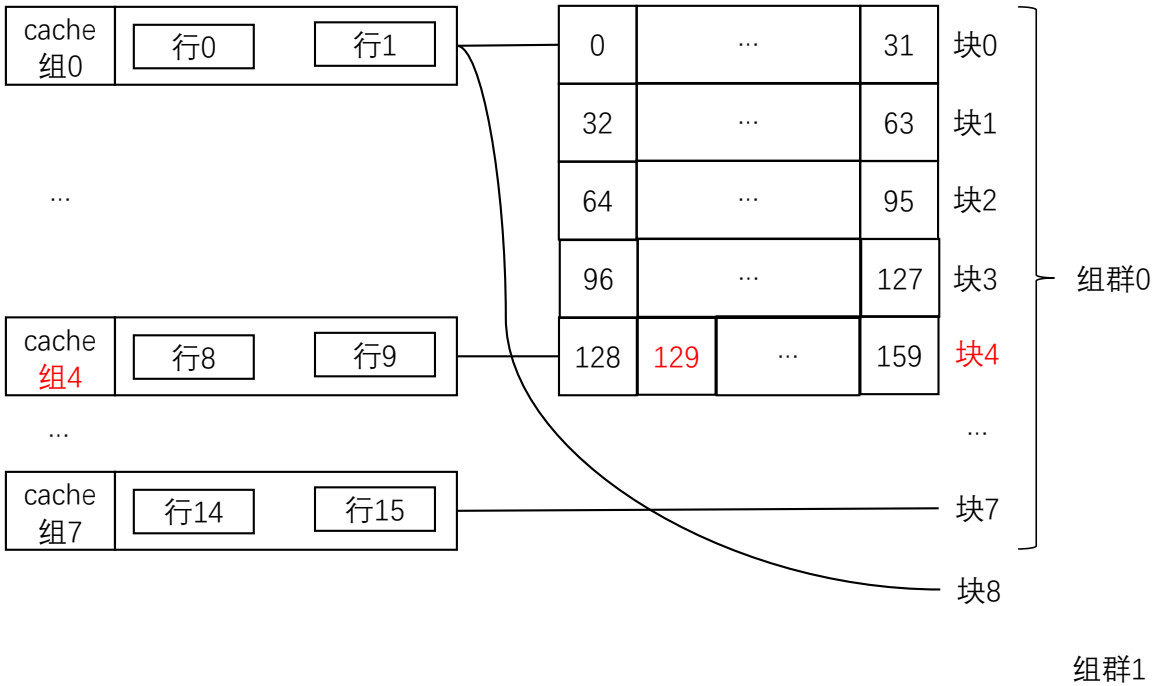
例：某计算机的Cache共有16行，采用2路组相联映射方式。每个主存块大小为32字节，按字节编址。主存129号单元所在主存块应装入到的Cache组号是（　　）。

解：

E=2行/组，S=16/2=8组

129/32=4…1，129号主存单元是内存块4中的单元1

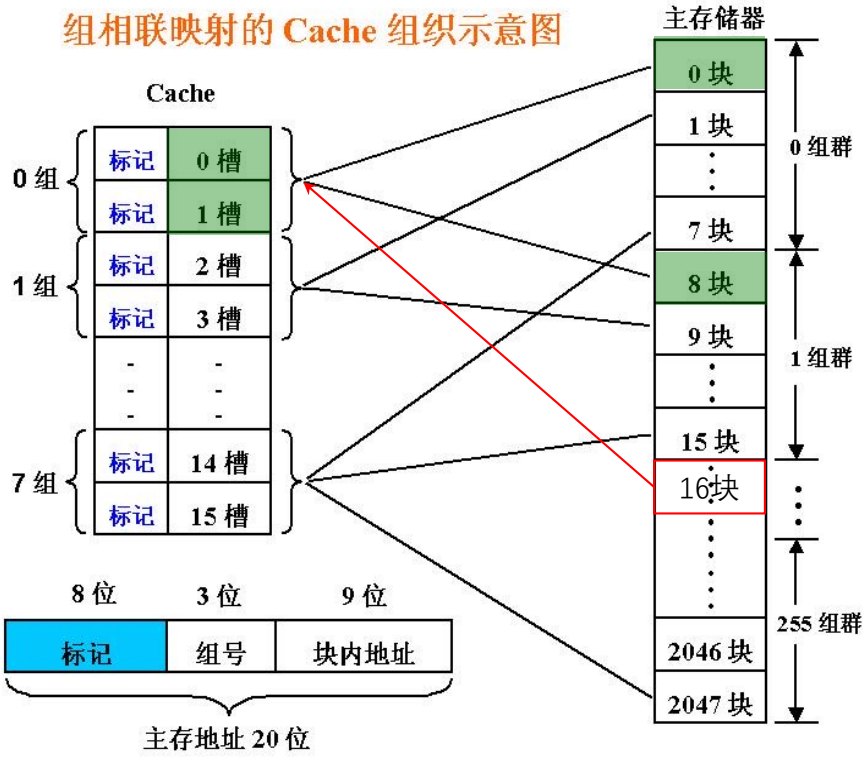
4 mod 8 = 4



替换算法

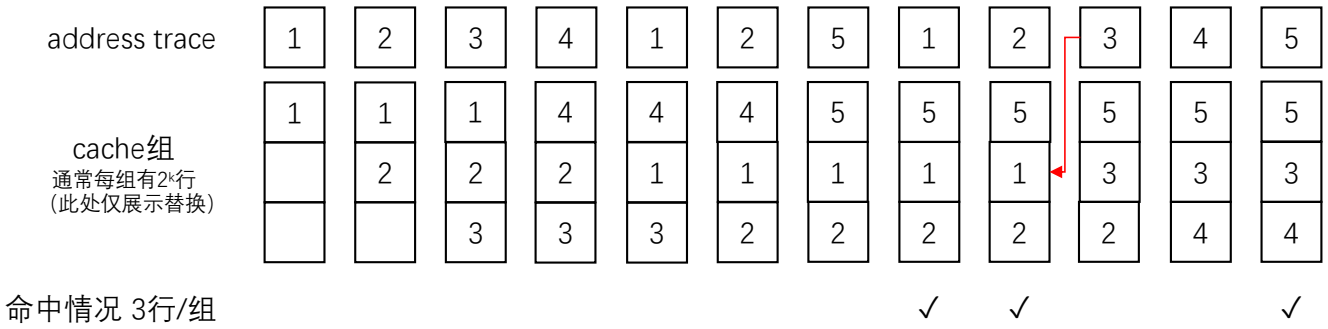
问题：2路组相联映射，E=2，S=8，假定第0组的两行分别被主存第0和8块占满，此时若需调入主存第16块，根据映射关系，它只能放到Cache第0组，因此，第0组中必须调出一块，那么调出哪一块呢？

- 何时替换
  - 直接映射：E=1，无需考虑替换谁
  - 组相联、全相联：需要考虑替换算法
- 替换过程
  - 主存取新块
  - 根据映射关系选择空cache行
  - 没有空行，替换
- 替换算法
  - 先进先出FIFO（first-in-first-out）
  - 最近最少用LRU（least-recently used）
  - 最不经常用LFU（least-frequently used）
  - 随机替换算法（Random）
  - …



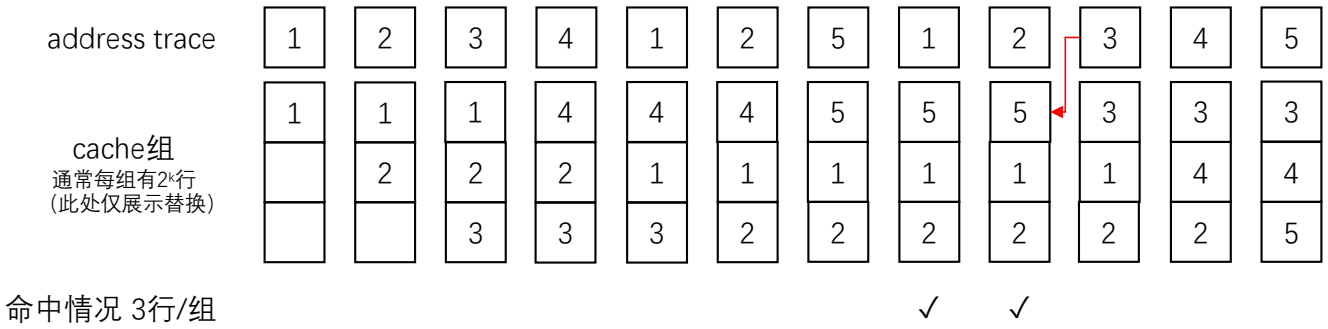
替换算法-FIFO

- FIFO：淘汰最早进入组的那一行
- 假定内存块{1,2,3,4,5}映射到同一cache组，对于以下地址访问序列，考察替换和命中情况。



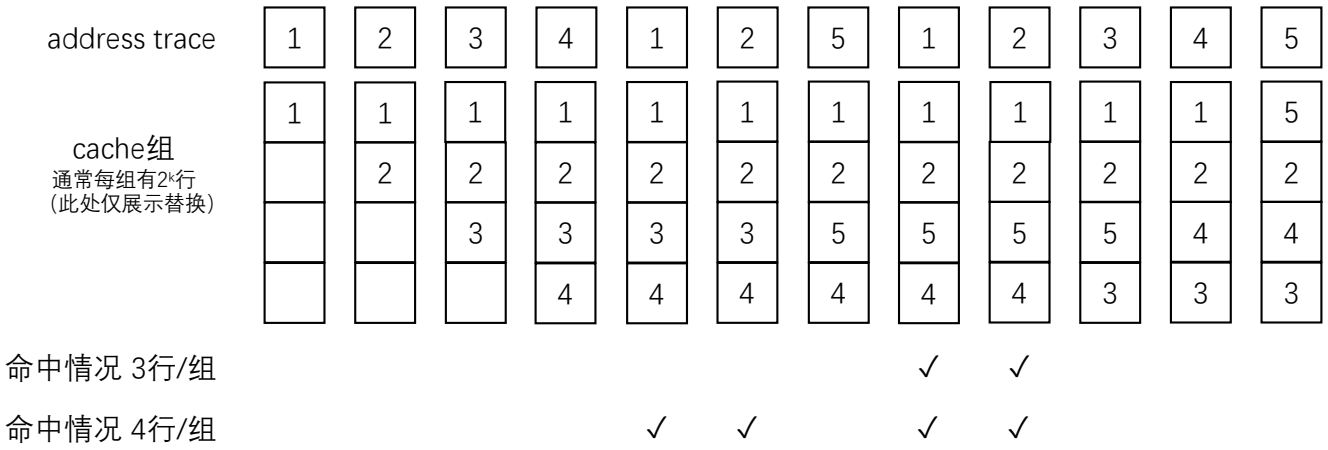
替换算法-LRU

- LRU：淘汰最后一次访问时间最久远的那一行
- 假定内存块{1,2,3,4,5}映射到同一cache组，对于以下地址访问序列，考察替换和命中情况。



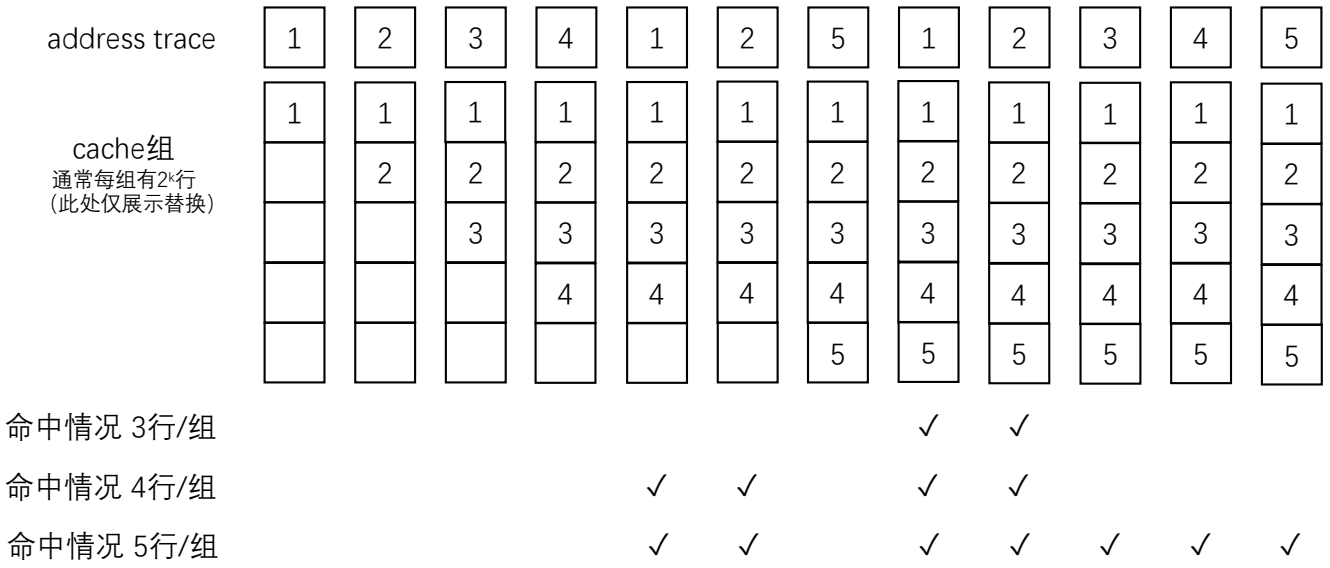
替换算法-LRU

- LRU：淘汰最后一次访问时间最久远的那一行
- 假定内存块{1,2,3,4,5}映射到同一cache组，对于以下地址访问序列，考察替换和命中情况。



替换算法-LRU

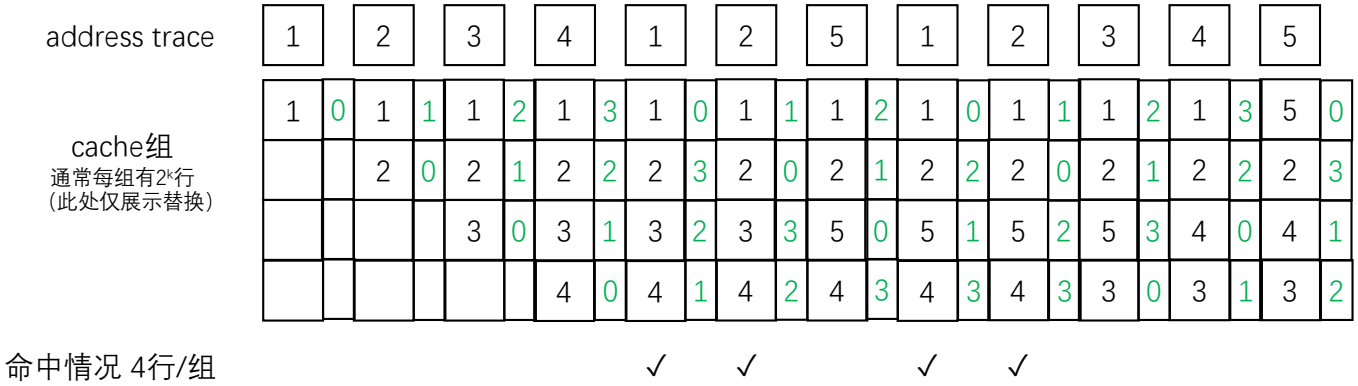
- LRU：淘汰最后一次访问时间最久远的那一行
- 假定内存块{1,2,3,4,5}映射到同一cache组，对于以下地址访问序列，考察替换和命中情况。



- 基于栈实现
- 命中率随cache组增大而提高（关联度越高，命中率越高）
- cache行不移动
- 每行有一个计数器（LRU位）记录使用情况

替换算法-LRU

- LRU：淘汰最后一次访问时间最久远的那一行
  - 计数器（LRU位）变化规则
    - 每组4行时，计数器有2位。计数值越小则说明越被常用。
    - 命中时，比被访问行计数低的计数器加1，被访问行的计数器置0，其余不变。
    - 未命中且该组未满时，新行计数器置为0，其余全加1。
    - 未命中且该组已满时，计数值为3的那一行中的主存块被淘汰，新行计数器置为0，其余加1。



- 例：假定计算机系统主存空间大小为32K字，按字编址，每字16位，且有一个数据区为4K字的4路组相联Cache，主存块大小为64字。假定Cache开始为空，处理器顺序地从存储单元0、1、…、4351中取数，一共重复10次。设Cache比主存快10倍。采用LRU算法。试分析Cache的结构和主存地址的划分。说明采用Cache后速度提高了多少？

解：

- 主存块数=32K / 64=512块
- Cache：E=4行/组，B=64字，总行数=4K / 64=64行，S=64/4=16

  结构为：16组 * 4行/组 * 64字/行
- 组群数=512 / 16=32，标记位数t=5，s=4，b=6

标识位	组号	块内地址
5位	4位	6位

- 例：假定计算机系统主存空间大小为32K字，按字编址，每字16位，且有一个数据区为4K字的4路组相联Cache，主存块大小为64字。假定Cache开始为空，处理器顺序地从存储单元0、1、…、4351中取数，一共重复10次。设Cache比主存快10倍。采用LRU算法。试分析Cache的结构和主存地址的划分。说明采用Cache后速度提高了多少？

解：

- 4351/64=67…63，即依次访问主存块0-67的所有字，重复10次

	第0行	第1行	第2行	第3行
第0组	0 64 48	16 0 64	32 16	48 32
第1组	1 65 49	17 1 65	33 17	49 33
第2组	2 66 50	18 2 66	34 18	50 34
第3组	3 67 51	19 3 67	35 19	51 35
第4组	4	20	36	52
…				
第15组	15	31	47	63

命中率p=(43520-68-20*9)/43520=99.43%
速度提高=10/(1*p+10*(1-p))=9.5倍

第1轮：68个内存块都只有首字miss，后续63字都hit；

第2轮：20个内存块只有首字miss，后续63字都hit；48个内存块所有字hit

第3-10轮：同第2轮

局部性！

写策略

- Cache读 字w
  - Cache命中，返回cache中w的副本给CPU
  - Cache不命中，去低级存储中找含w的块，块进入cache（可能发生替换），返回w给CPU
- Cache写
  - 如何保持cache和主存中数据的一致性
  - 写操作的两种情况
    - 写命中（Write Hit）：要写的单元已经在Cache中
    - 写不命中（Write Miss）：要写的单元不在Cache中

读比写容易，
指令cache比数据cache容易设计

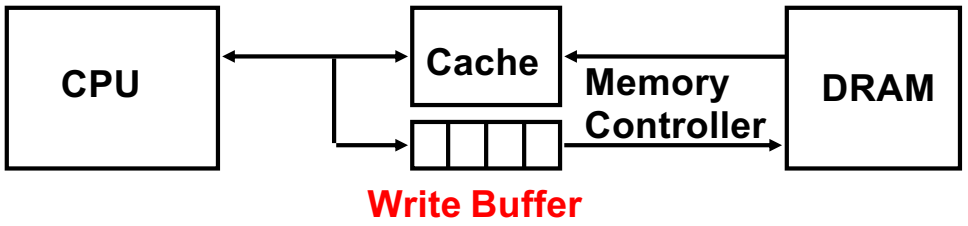
写策略

- 写命中的两种处理方式
  - Write Through (直写)　　简单，但每次写都引起总线流量
    - 更新字w在cache中的副本
    - 立即将w所在的块写回主存
  - Write Back (写回)　　由于局部性，总线流量显著降低，但复杂
    - 更新字w在cache中的副本
    - cache行的修改位（dirty bit）置1
    - 该cache行被替换时将其写回内存
- 写不命中的两种处理方式
  - Write Allocate (写分配)　　利用空间局部性，一旦写不命中就要先传送一个块
    - 先加载w所在块到cache
    - 后执行写命中（通常是写回方式）
  - Not Write Allocate (非写分配)
    - 直接写主存单元，不把主存块装入到Cache

写策略

- Write Through (直写)
  - 写缓冲
    - CPU同时写数据到Cache和Write Buffer
    - 内存控制器将缓冲内容写主存
    - Write buffer是一个FIFO队列

简单，但每次写都引起总线流量



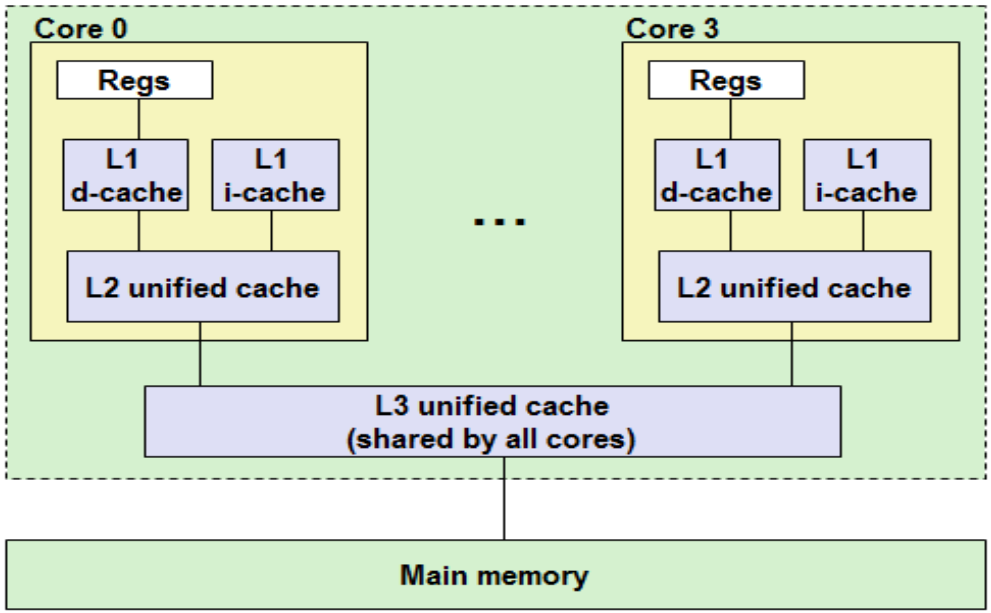
- 假定主存地址为32位，按字节编址，主存和Cache之间采用直接映射方式，主存块大小为4个字，每字32位，采用回写(Write Back)方式，则能存放4K字数据的Cache的总容量的位数至少是（　　）。

解：

- B=4*32b=4*4B=2^4B，　b=4
- C=4K*32b=B*E*S，E=1，S=2^{10}，s=10
- 内存地址=标记位（块群号）+cache行号+块内地址，32=(t,s,b)=(18,10,4)
- cache行=有效位+标记位+内存块数据，(V,t,B)=(1,18,4*32)=(1,18,128)=147
- 写回方式，每行需要1位修改位
- 总容量=(147+1)*2^{10} b=148K b

Intel Core i7某型号处理器的cache结构

- L1，i-cache、d-cache
  - 各32KB、8路、4个时钟周期
- L2 cache
  - 各256KB、8路、11个时钟周期
- L3 cache
  - 所有核共享
  - 8MB、16路、30~40个时钟周期
- 所有cache块大小都是64B



Cache与程序性能

例：某32位机器主存地址空间大小为256MB，按字节编址。指令cache和数据cache均有8行，采用直接映射，主存块为64B。假定编译时i, j, sum均分配在寄存器中，int数组a按行优先方式存放，其首址为320。

（1）不考虑用于一致性的控制位，数据cache的总容量为多少？

（2）考虑用于一致性的控制位，采用写回法，数据cache的总容量为多少？

（3）a[0][31]和a[1][1]各自所在主存块对应的cache行号分别是多少？

（4）程序A和B的数据访问命中率各是多少？哪个程序的执行时间更短？

程序 A：

```
int a[256][256];
……
int sum_array1 ( )
{
        int i, j, sum = 0;
        for ( i = 0; i < 256; i++)
                for (j = 0; j < 256; j++)
                        sum += a[i] [j];
        return sum;
}
```

程序 B：

```
int a[256][256];
……
int sum_array2 ( )
{
        int i, j, sum = 0;
        for ( j = 0; j < 256; j++)
                for ( i = 0; i < 256; i++)
                        sum += a[i] [j];
        return sum;
 }
```

Cache与程序性能

解:

（1）256MB / 64B = 2^{22}主存块，2^{22} / 8 = 2^{19}块群　　256M需要28位编码，19+3+6=28<32

无控制位，数据cache容量=(1+19+64*8)*8=4256b

（2）含控制位，数据cache容量=(1+19+64*8+1)*8=4264b

（3）&a[0][0]=320，&a[0][31]=320+31*4=444，444/64=6…60，a[0][31]在主存块6，

6 mod 8 = 6，在数据cache行6

&a[1][1]=320+1*256*4+1*4=1348，1348/64=21…4，a[1][1]在主存块21

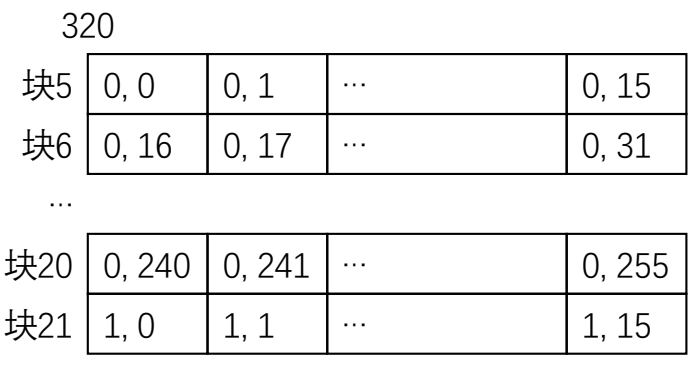
21 mod 8 = 5，在数据cache行5

Cache与程序性能

解：

（4）320/64=5…0，64B/4B=16

数组a在内存中的布局如右图所示



对于程序A，访问顺序同布局顺序，每块都是只有第一个数组元素miss，

命中率=15/16=93.75%

对于程序B，访问顺序为a[0][0], a[1][0], a[2][0], …，cache不命中装入顺序为：块5不命中 装入行5，块21不命中 装入行5 替换块5，…，即每个数组元素都miss

命中率=0

Cache参数及性能影响

- 不命中率（miss rate）
  - 在一个程序执行或程序的一部分执行期间，内存引用不命中的比率
  - 不命中数量/引用数量
- 命中率（hit rate）
  - 命中的内存引用比率
  - 1-不命中率
- 命中时间（hit time）
  - 从cache传送一个字到CPU所需的时间，包括组选择、行确认和字选择的时间
  - 对于L1高速缓存，命中时间的数量级一般为几个时钟周期
- 不命中处罚（miss penalty，缺失损失）
  - 由于不命中所需要的额外的时间
  - L1不命中需要从L2得到服务的处罚，通常是数10个周期
  - 从L3得到服务的处罚约50个周期
  - 从主存得到的服务的处罚约200个周期

平均访问时间

- 假设有一个一级缓存的存储层次结构，即 CPU ↔ Cache ↔ DRAM
- 平均访问时间 $t_{avg} = \alpha t_c + (1-\alpha)(t_c + t_m) = t_c + (1-\alpha)t_m$
  - α：缓存命中率
  - t_c：缓存命中时间
  - t_m：主存访问时间，即缓存不命中处罚
  - $t_c + t_m$：缓存不命中时访问时间
- 要降低平均访问时间，就要提高缓存命中率

例：对于SRAM访问时间4ns、DRAM访问时间40ns的硬件，希望其上一个程序的平均访问时间控制在5ns以内，则需要达到多少的缓存命中率？

解：4+(1-α)*40<=5，α>=97.5%

- 例：假定处理器时钟周期为2ns，某程序有3000条指令组成，每条指令执行一次，其中4条指令在取指令时发生cache缺失，其余指令都在cache中命中。在执行指令过程中，该程序需要1000次主存数据访问，其中6次发生cache缺失。问：（1）执行该程序的Cache命中率是多少？（2）若cache命中时间为1个时钟周期，缺失损失为10个时钟周期，则CPU在cache-主存层次的平均访问时间为多少？

解：

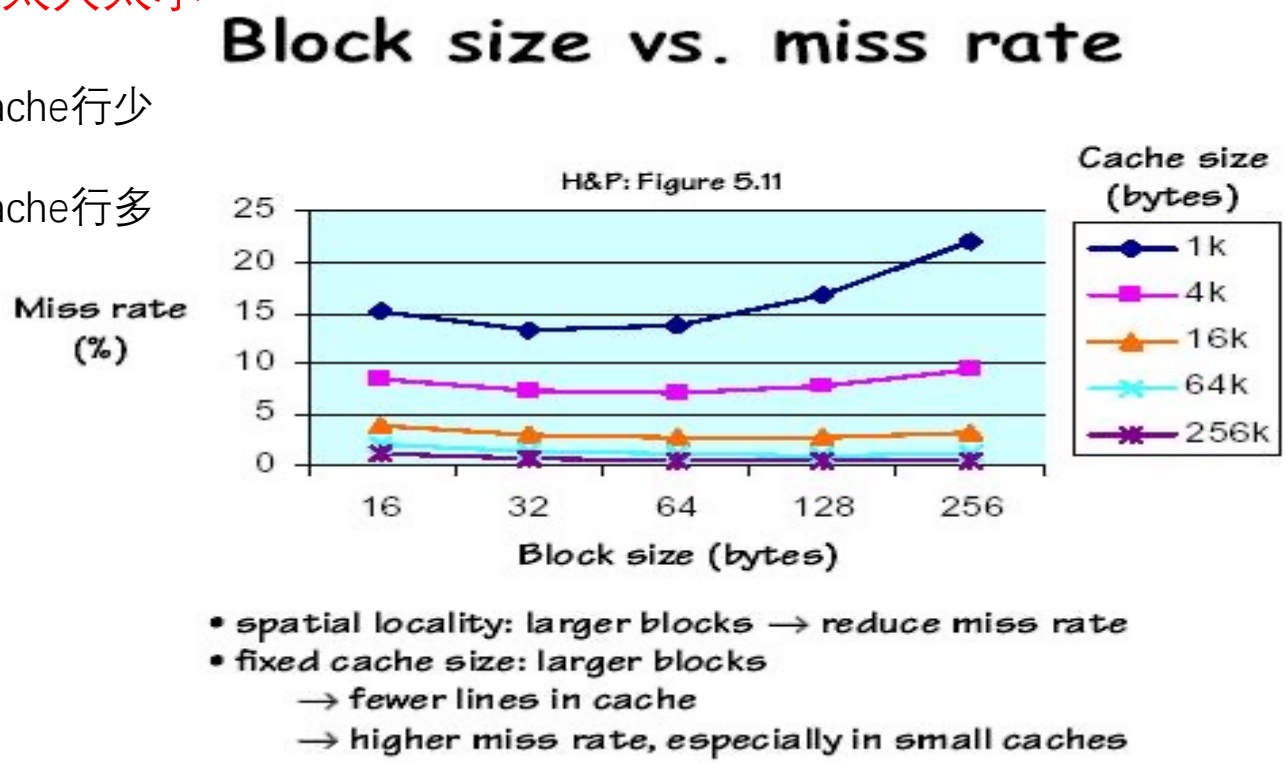
（1）指令和数据共访问3000+1000=4000次，其中不命中4+6=10次

命中率=(4000-10)/4000=99.75%

（2）平均访问时间=2+(1-0.9975)*20=2.05ns

Cache大小、Block大小 与 缺失率

- Cache越大，缺失率越低，成本也越高
- Cache大小一定，Block太大太小都会提高缺失率
  - Block大，局部性好，cache行少（对小cache影响更大）
  - Block小，局部性弱，cache行多



华为某公开难题

欢迎感兴趣的同学和我联系（大创/毕设/科研/竞赛，不局限于该题目，hmz@njupt.edu.cn，请附完整成绩单和简单自我介绍）

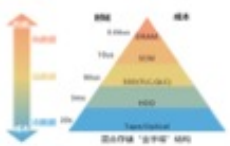
难题5：磁带介质数据排布算法

出题组织：理论研究部　　接口专家：

HUAWEI

问题背景

为满足温冷数据存储场景对于容量、性能、成本的综合需求，常将少量高速介质（如SSD）作为缓存与其它大容量廉价存储介质（HDD、磁带、光盘等）搭配，实现成本和性能的平衡。磁带作为常用的冷数据存储介质，是一种典型的线性介质，在介质表面以蛇形记录的方式进行读写（以Lto 9磁带为例，单盘容量18TB，横向磁道数~8000条，纵向长度~1km，单向移动速率~5m/s，加减速过程~2s，在顺序读写场景下最大带宽~400MB/s）。磁带表面积约为HDD ~100x，密度约~0.01x，空间距离制约是导致磁带读写时延高的核心原因。



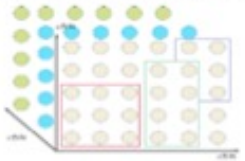




技术挑战

当用户数据存放在磁带中时，由于用户数据往往存在多维特征，以R^n表示(如时间、空间坐标、类别等)，用户读取数据时往往只使用其中关联的一部分，在多维空间中表现为提取一个矩形块。简单的数据排布方式（如按某一维度）会使得距离较近的数据分布在空间距离较远的带体上，单块数据的读取也会造成频繁的启停掉头，导致用户请求时延变长。

希望结合用户数据使用习惯，给出一种**多维数据到磁带介质的映射方法（数据排布算法）**，缩短数据空间分布距离，减少读取时的启停掉头频率，提升系统整体时延性能。



三维数据示例，用户按照某类矩形块读取数据

问题描述

- **建模**：对于线性介质，一种简单的建模方法是将一次读请求的时延分解成三部分：首字节寻址时延，掉头时延，读取位移时延。
- **约束条件**：
  - **用户习惯**：用户IO随机性（数据块大小、位置等）比较大，因此考虑均匀随机读写意义下的平均访问时延。
  - **额外条件**：可使用DRAM和SSD作为缓存，容量配比参考Lto 9磁带，单盘18TB磁带，约300MB DRAM，约540GB SSD。

模型

用户请求$B_k=\{I_i\}_k, I_j$—对应的文件编号，

变量定义：X_i, s_i—文件i地址，大小；v——速度，z_{ij}—文件i是否在文件j之后的指示变量，为1表示出现倒带

目标：极小化$T_1+T_2+T_3$，

表述

首字节寻址时延$T_1=\mathbb{E}_{X_0}\frac{X_{i_1}-X_0}{v}+Kz_{i_k i_1}$，掉头时延$T_2=K\sum_i z_{i_i i_{i-1}}$，读取位移时延$T_3=\sum_i\frac{|\Delta(X_{i_i}+s_{i_i})|}{v}$

当前结果

- 当前在某场景二维数据集上，采用磁带文件系统LTFS默认排布方式，按照某一维度优先排布，0.1M-20M数据块下，纯磁带介质全随机读时延166s；
- 在二维数据仿真结果中，不同维度优先排布方式下，行优先和列优先掉头次数差距可以达到3倍，对掉头做优化能够降低大约20%的掉头次数。

技术诉求

- 1. 步骤：设计模型与算法-离线仿真验证-在华为提供的测试环境与数据中5分钟内能跑出次优解且同时达成条件2和3的指标要求。
- 2. 以最短平均读时延为目标，求出磁带介质上最优或者次优的数据排布。
- 3. 0.1M-20M数据块下平均读时延降低20%，时延<130s或给出时延理论极限。

参考文献：

[1] Data Placement Techniques for Serpentine Tapes. HICSS.

[2] Magnetic Tape Storage Technology. IBM.