存储器

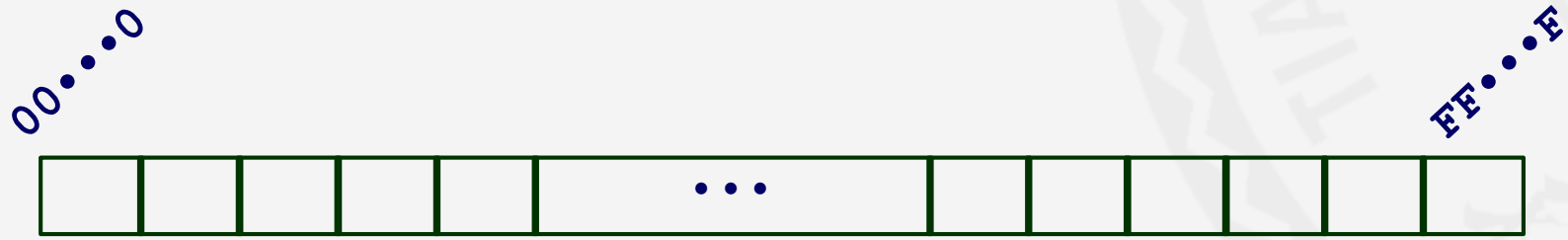
Memory

本章内容

Topic

- 存储技术
  Storage technologies
  - 随机访问存储器
    Random-Access Memory
  - 磁盘存储
    Disk Storage
  - 固态硬盘
    Solid State Disks
  - 存储技术趋势
    Storage Technology Treads
- 局部性原理
  Locality of reference
- 存储器层次结构
  The memory hierarchy

面向字节的存储器组织
Byte-Oriented Memory Organization



- 程序根据地址引用数据
  Programs refer to data by address
  - 概念上，可将其想象为一个非常大的字节数组
    Conceptually, envision it as a very large array of bytes
    - 事实上不是，但可以这么认为
      In reality, it's not, but can think of it that way
  - 地址就像数组的下标
    An address is like an index into that array
    - 用指针来存储地址
      a pointer variable stores an address

回顾：几种简单的存储器寻址模式
Rivisited: Simple Memory Addressing Modes

- 间接寻址 (R) Mem[Reg[R]]
  Normal (R) Mem[Reg[R]]
  - 寄存器 **R** 指向了存储器的地址
    Register **R** specifies memory address
  - 和C语言中的指针作用相同
    Pointer dereferencing in C

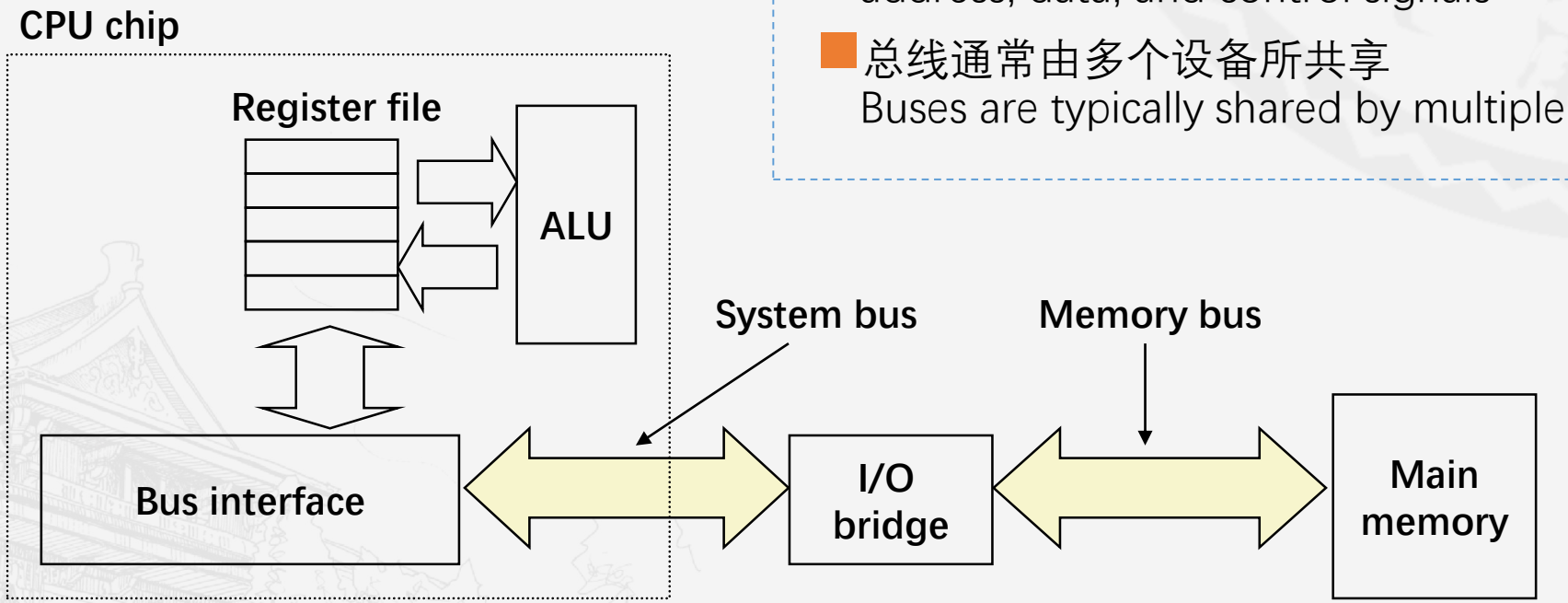
```
movq (%rcx),%rax
```

- 基地址+偏移量寻址 D(R) Mem[Reg[R]+D]
  Displacement D(R) Mem[Reg[R]+D]
  - 寄存器 **R** 指定了存储器区域的开始位置
    Register **R** specifies start of memory region
  - 常数 **D** 是偏移量
    Constant displacement D specifies offset

```
movq 8(%rbp),%rdx
```

使用传统总线结构连接CPU和内存
Traditional Bus Structure Connecting CPU and Memory

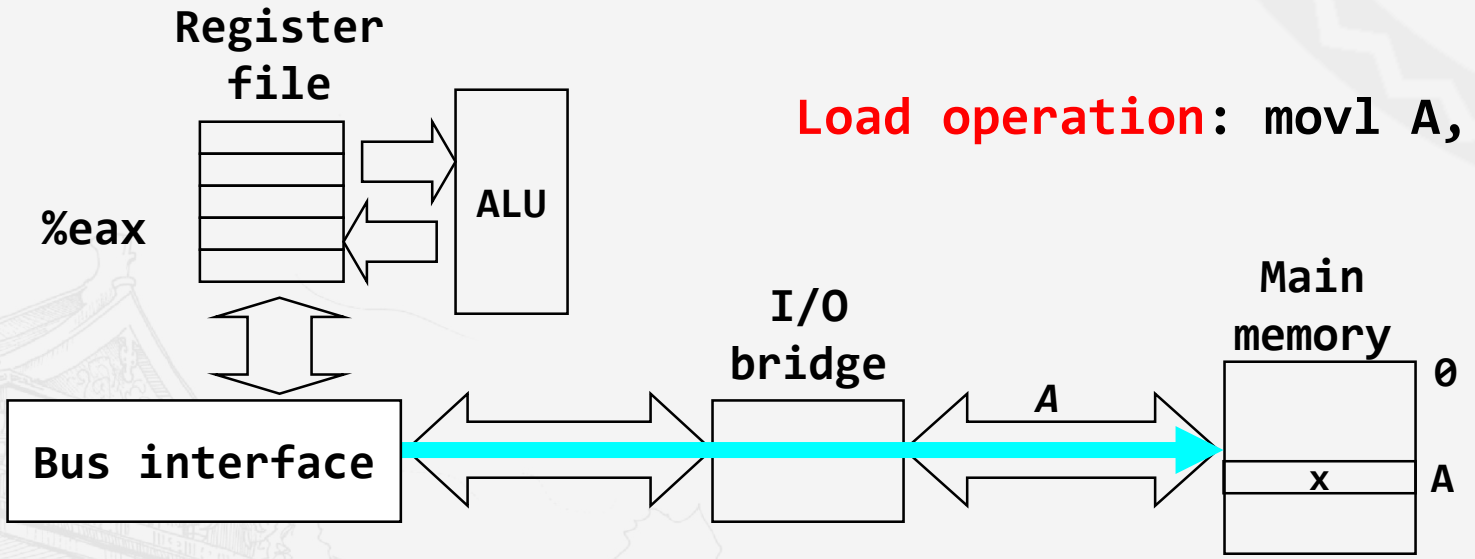
- 总线：一组用于传输地址、数据和控制信号的并行线路
  Bus: a collection of parallel wires that carry address, data, and control signals
- 总线通常由多个设备所共享
  Buses are typically shared by multiple devices



内存读 （1）
Memory Read Transaction (1)

CPU将地址A发送到总线上

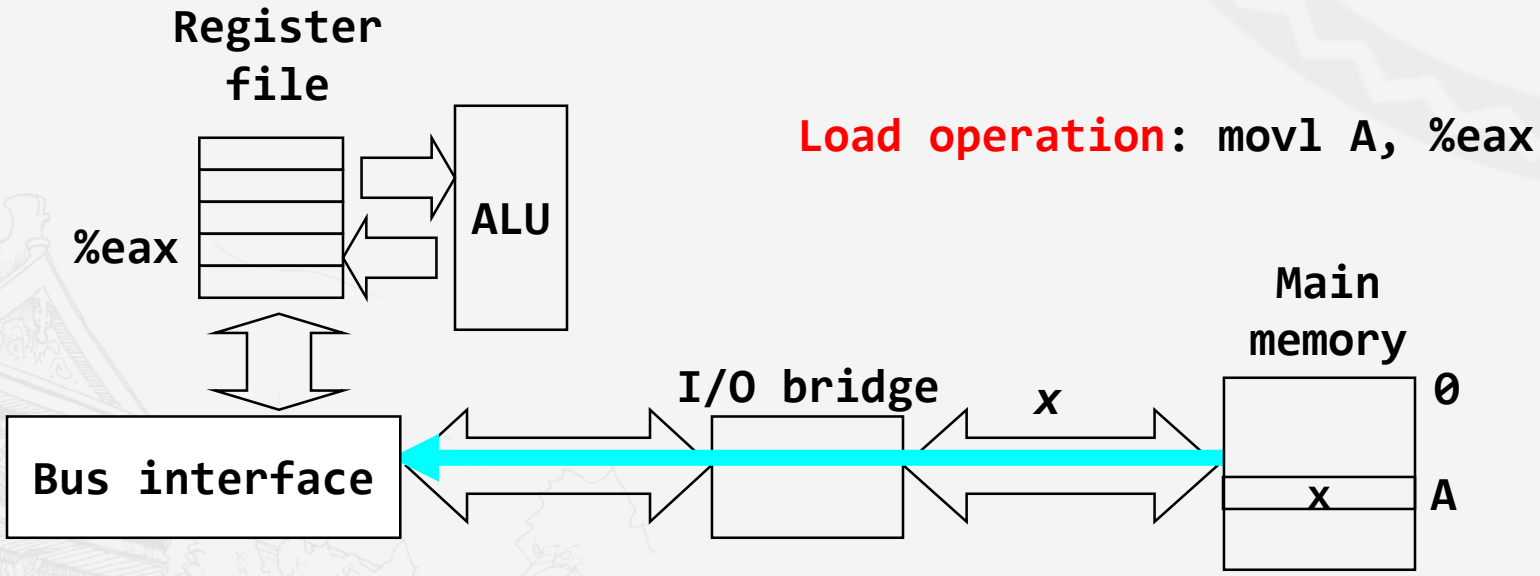
CPU places address A on the bus



内存读 （2）
Memory Read Transaction (2)

主存从总线上接收到地址A，检索到数据x（字），然后发送到总线上

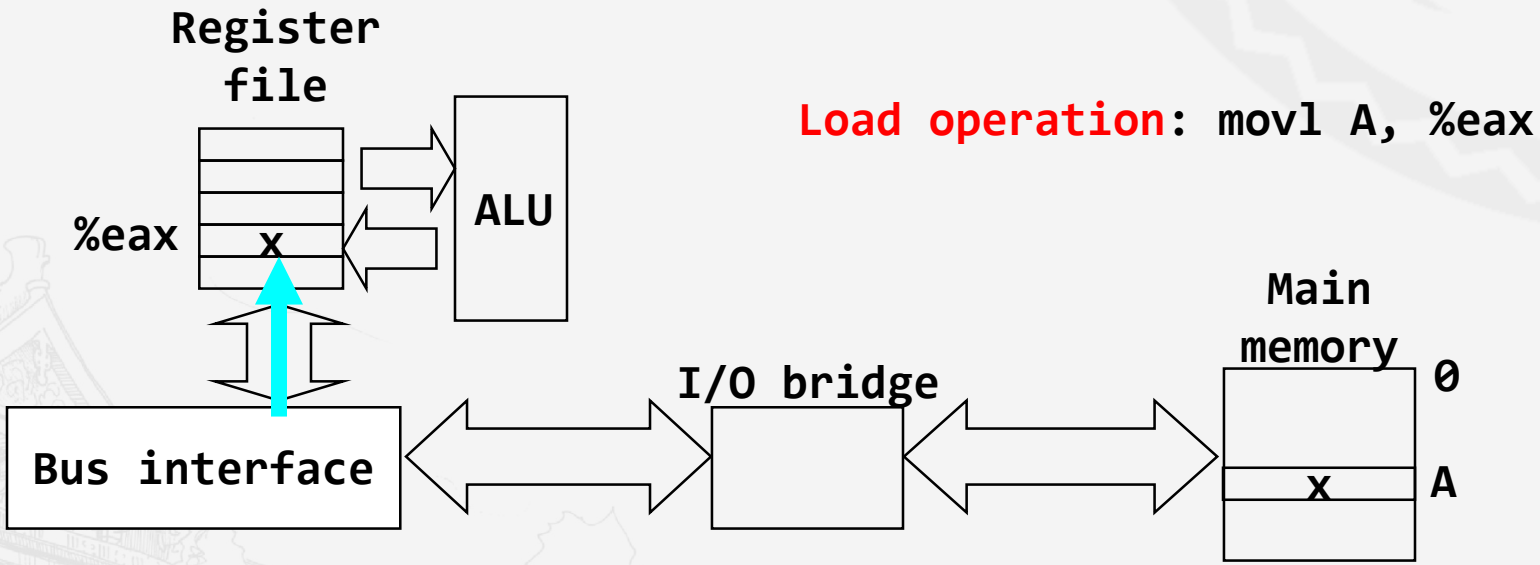
Main memory reads A from the bus, retrieves word x (data), and places it on the bus



内存读 （3）
Memory Read Transaction (3)

CPU从总线上接收数据x（字）， 然后写入到%eax寄存器中

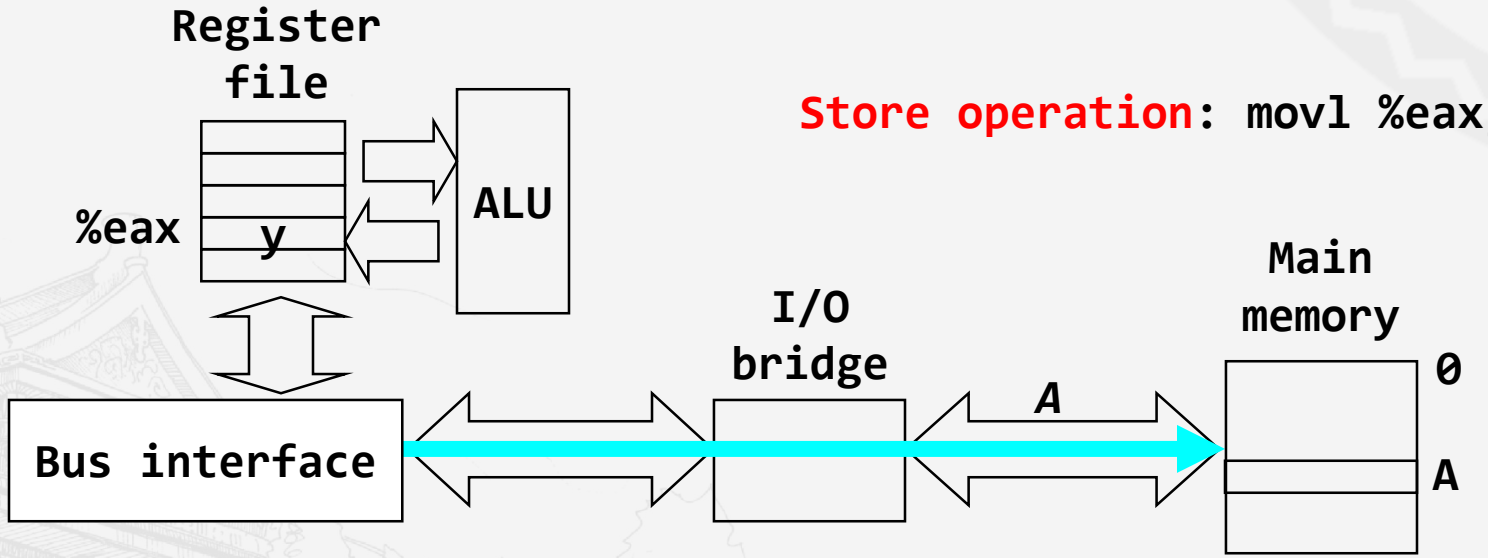
CPU read word x from the bus and copies it into register %eax



内存写 （1）
Memory Write Transaction (1)

CPU将地址A发送到总线上，主存收到地址并等待相应数据的到达

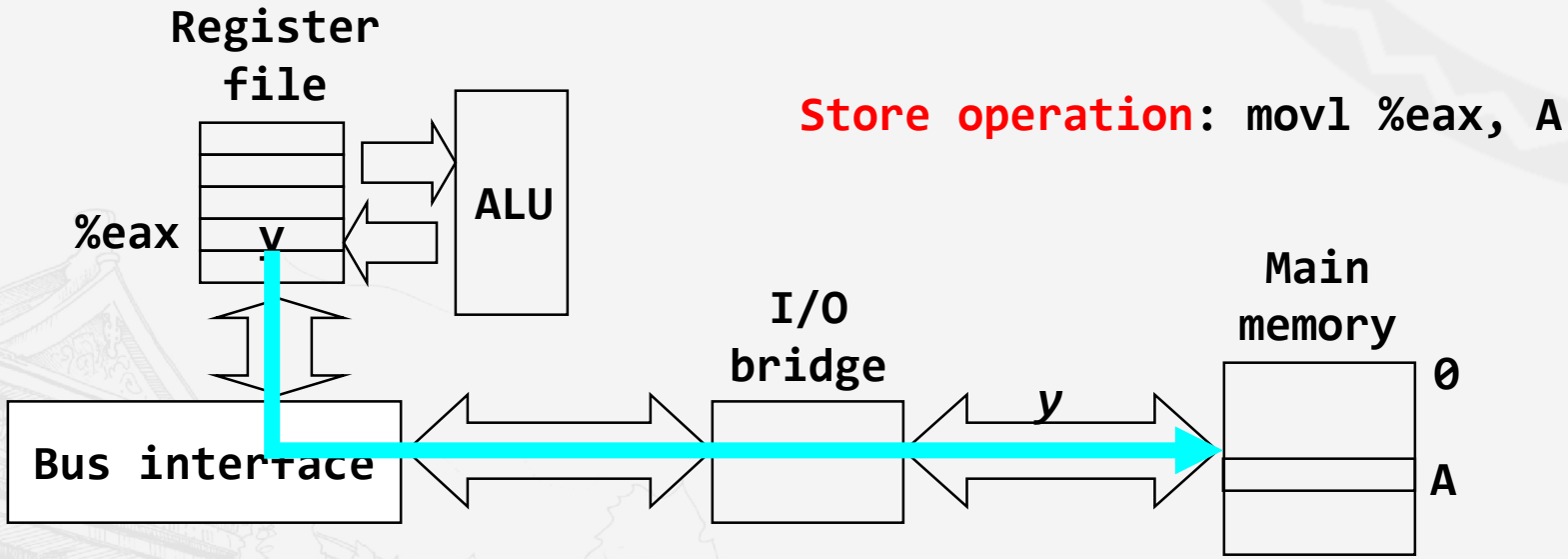
CPU places address A on bus. Main memory reads it and waits for the corresponding data word to arrive.



内存写 （2）
Memory Write Transaction (2)

CPU通过总线发送数据y （字）

CPU places data word y on the bus.

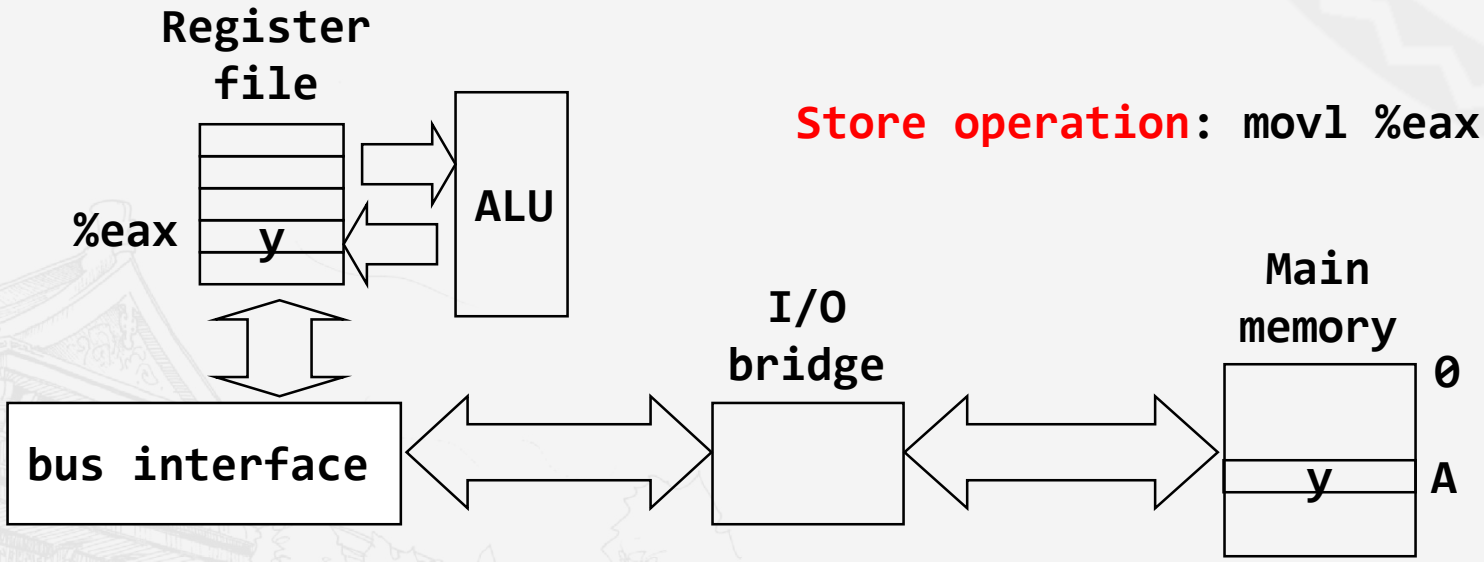


Store operation: movl %eax, A

内存写 （3）
Memory Write Transaction (3)

主存从总线上接收到数据y（字）， 然后存储到地址A

Main memory reads data word y from the bus and stores it at address A

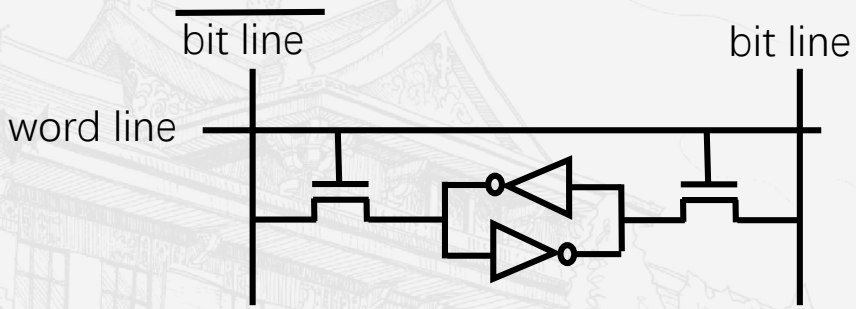


存储技术

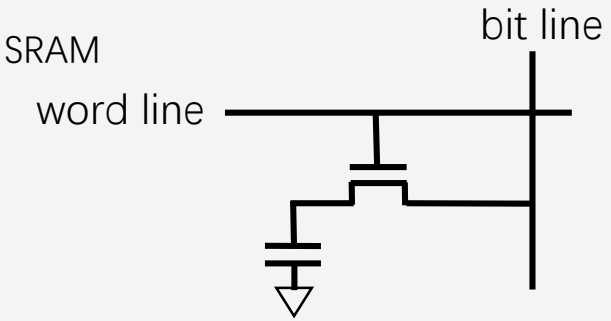
Storage technologies

随机访问存储器
Random-Access Memory (RAM)

- 静态随机访问存储器
  Static RAM (SRAM)
  - 每个单元用四个或六个晶体管构成的电路存储一个比特
    Each cell stores a bit with a four or six-transistor circuit
  - 只要保持供电，内容就会一直保持
    Retains value indefinitely, as long as it is kept powered
  - 对电噪声(EMI)、辐射等相对不敏感（抗干扰能力强）
    Relatively insensitive to electrical noise (EMI), radiation, etc
  - 比DRAM更快也更贵
    Faster and more expensive than DRAM



- 动态随机访问存储器
  Dynamic RAM (DRAM)
  - 每个单元用一个电容器存储一个比特
    Each cell stores bit with a capacitor
  - 还需要一个晶体管用于存取
    One transistor is used for access
  - 数据必须每10-100毫秒刷新一次
    Value must be refreshed every 10-100 ms
  - 对干扰(电噪声，辐射，…)比SRAM更敏感
    More sensitive to disturbances (EMI, radiation,…) than SRAM
  - 比SRAM更慢也更便宜
    Slower and cheaper than SRAM



存储技术

Storage technologies

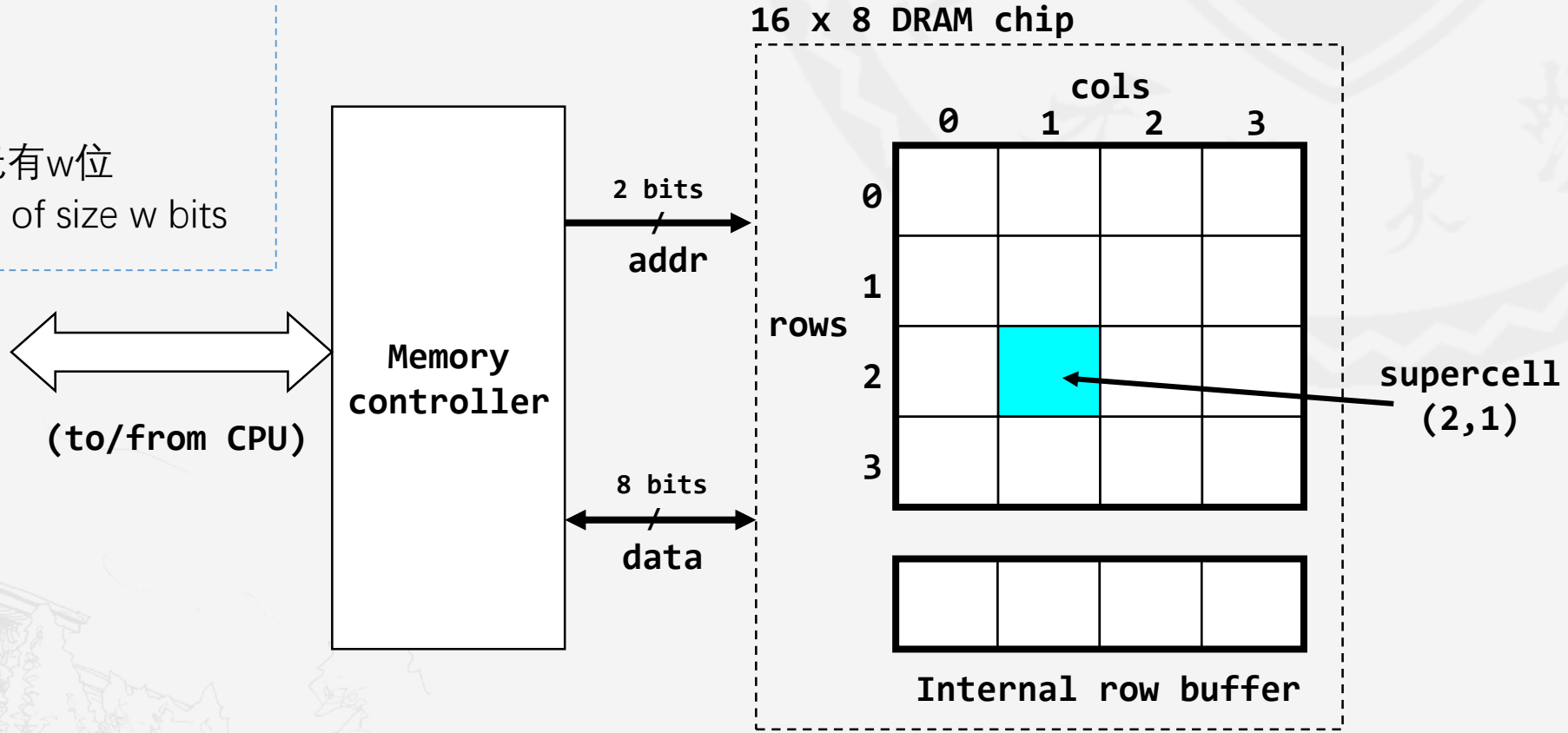
动态随机访问存储器
Dynamic RAM (DRAM)

- 主要特征
  Key features
  - 传统上，DRAM通常被封装成一个芯片
    DRAM is traditionally packaged as a chip
  - 基本存储单位通常是一个单元（每个单元一位）
    Basic storage unit is normally a cell (one bit per cell)
  - 大多数计算机上的主存都是有多个DRAM芯片构成的
    Multiple DRAM chips form main memory in most computers

- 技术特点
  Technical characteristics
  - 以两个维度（行和列）组织
    Organized in two dimensions (rows and columns)
  - 在DRAM芯片内访问：先选择行，然后选择列
    To access (within a DRAM chip): select row then select column
  - 结果：第二次访问同一行比不同的行更快
    Consequence: 2nd access to the same row faster than different row

常规的DRAM组织
Conventional DRAM Organization

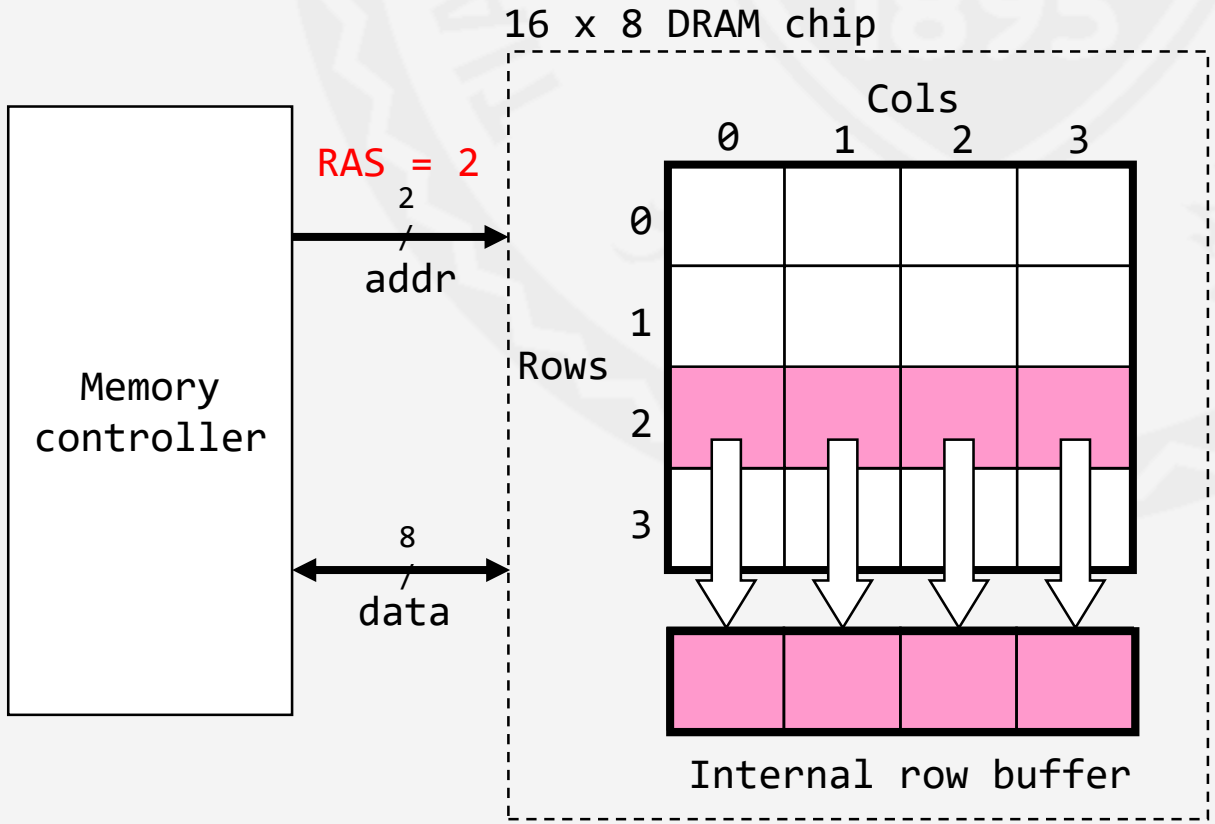
- d x w DRAM:
  - 总容量：d x w位
    Total bits: d x w
  - d 个超单元，每个超单元有w位
    Organized as d supercells of size w bits



读DRAM超单元(2,1)
Reading DRAM Supercell (2,1)

第一步（a）：行访问选通信号选择第2行
Step 1(a): Row access strobe (RAS) selects row 2

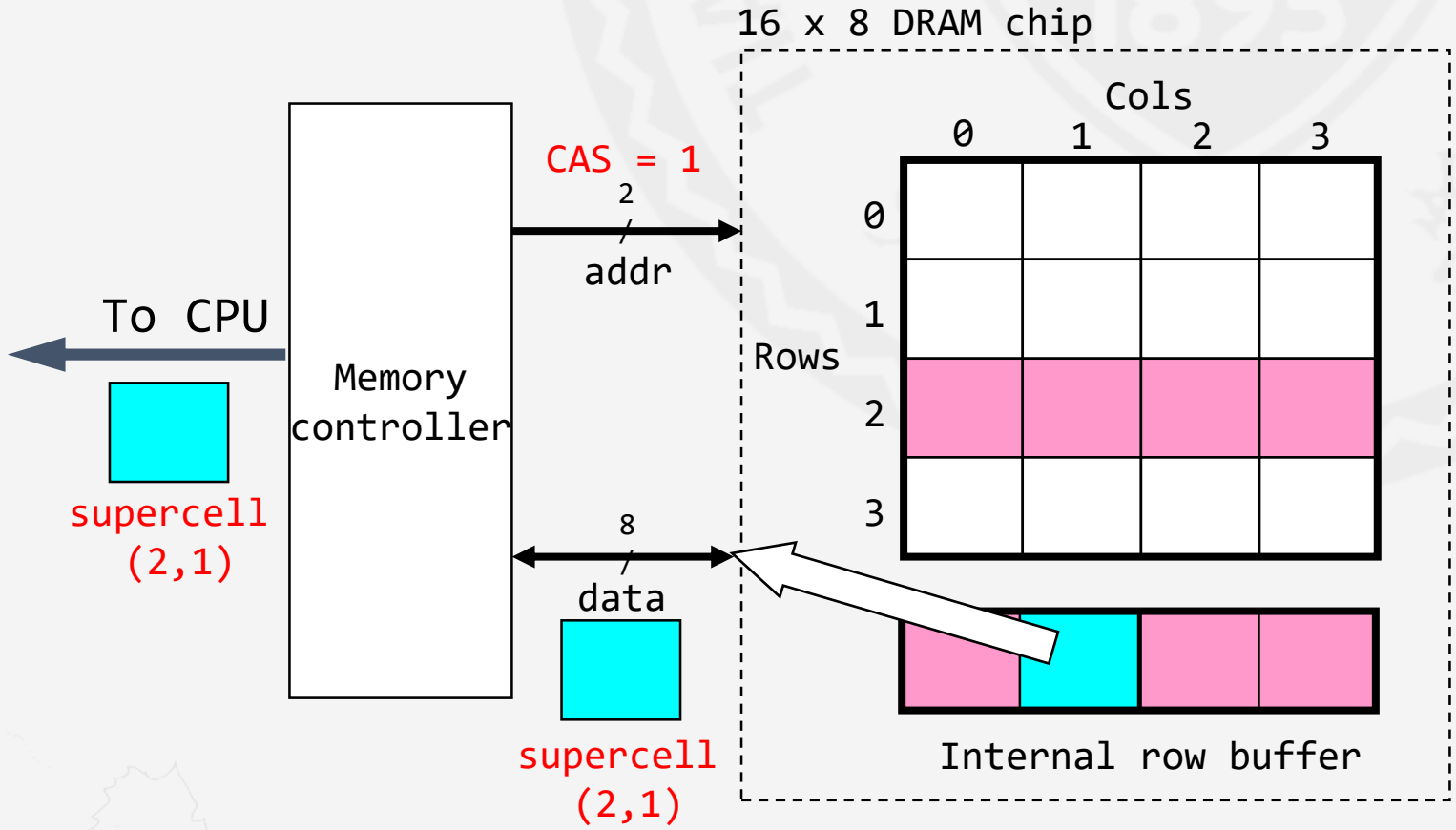
第一步（b）：行2复制到DRAM阵列的行缓冲区中
Step 1(b): Row 2 copied from DRAM array to row buffer



读DRAM超单元(2,1)
Reading DRAM Supercell (2,1)

第二步（a）：列访问选通信号选择第1列
Step 2(a): Column access strobe (CAS) selects column 1

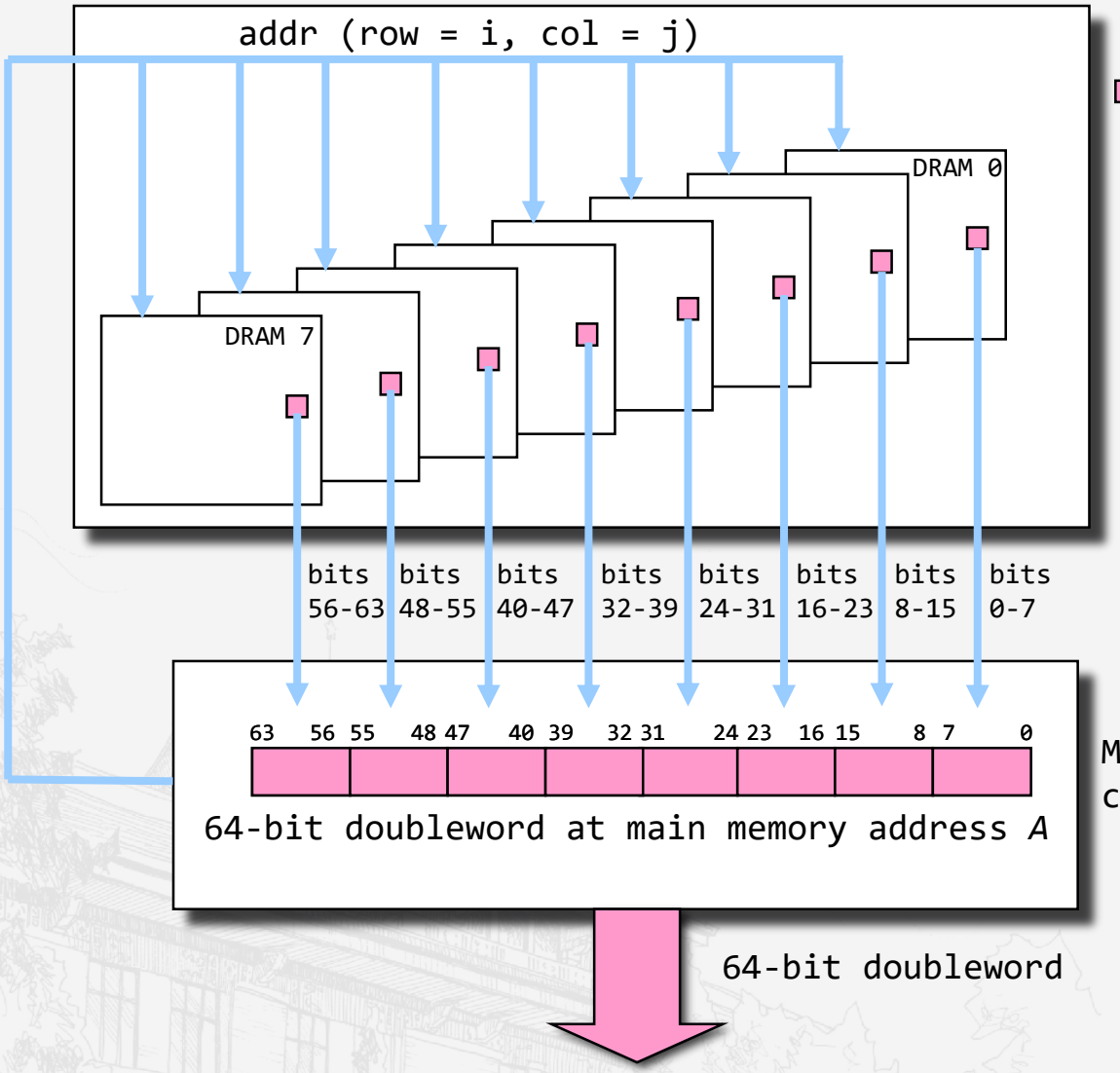
第二步（b）：超单元（2,1）从缓冲区复制到数据线，并最终返回CPU
Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU



存储技术

Storage technologies

内存条
Memory Modules



■ : supercell (i,j)

64MB内存条由8个 8Mx8 的DRAM芯片构成
64 MB memory module consisting of eight 8Mx8 DRAMs

增强型DRAM
Enhanced DRAMs

- 基本DRAM单元自1966年发明以来没有改变
  Basic DRAM cell has not changed since its invention in 1966
  - 1970年被英特尔商用
    Commercialized by Intel in 1970
- 更强大接口和更快I/O的DRAM
  DRAM cores with better interface logic and faster I/O :
  - **同步DRAM**
    Synchronous DRAM (SDRAM)
    - 使用常规时钟信号代替异步控制
      Uses a conventional clock signal instead of asynchronous control
    - 允许行地址复用（例如，RAS，CAS，CAS，CAS）
      Allows reuse of the row addresses (e.g., RAS, CAS, CAS, CAS)
  - **双数据速率同步DRAM**
    Double data-rate synchronous DRAM (DDR SDRAM)
    - 采用双沿时钟触发，每周期每引脚发送两比特
      Double edge clocking sends two bits per cycle per pin
    - 根据预取小缓冲区的尺寸区分的不同类型:
      - DDR (2 bits), DDR2 (4 bits), DDR3 (8 bits)
    - 目前是大多数服务器和桌面系统的标准
      At present, standard for most server and desktop systems
    - Intel 酷睿 i9 处理器目前支持DDR5 SDRAM
      Intel Core i9 supports DDR5 SDRAM

SRAM和DRAM的比较
Comparison between SRAM and DRAM

	每比特晶体管 Trans. per bit	访问时间 Access time	需要刷新? Needs refresh?	需要EDC? Needs EDC?	成本 Cost	应用 Applications
SRAM	4 or 6	1X	No	Maybe	100x	高速缓存 Cache memories
DRAM	1	10X	Yes	Yes	1X	主存 Main memories 显存 Frame buffers

EDC: Error Detection Code
错误检测码（校验码）

存储技术

Storage technologies

非易失存储器
Nonvolatile Memories

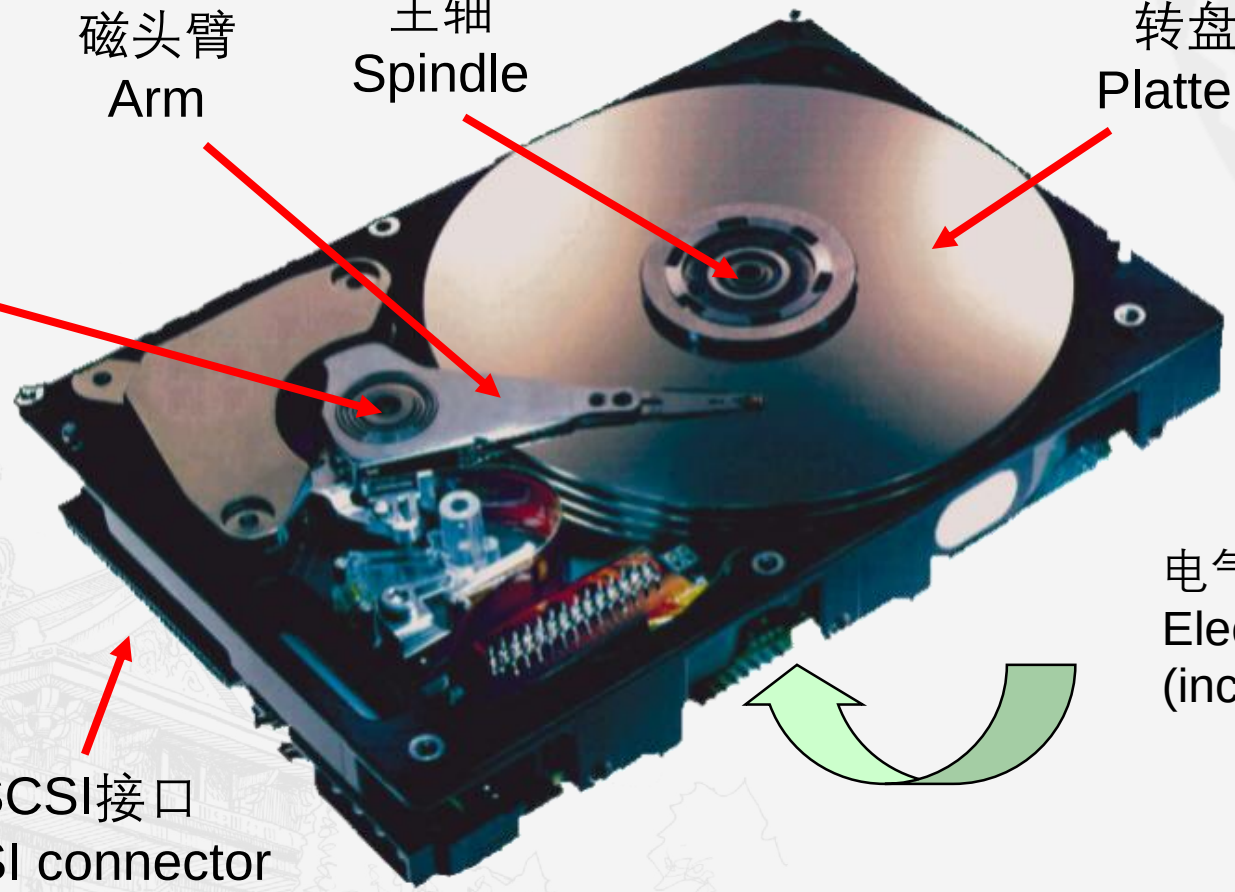
- DRAM和SRAM都是易失存储器
  DRAM and SRAM are volatile memories
  - 断电时，丢失信息
    Lose information if powered off
- 最常见的非易失存储器是硬盘
  Most common nonvolatile storage is the hard disk
  - 旋转的盘片（像DVD一样），大容量，但非常慢
    Rotating platters (like DVDs)… plentiful capacity, but very slow
- 非易失存储器可以在断电时保持信息
  Nonvolatile memories retain value even if powered off
  - ROM、PROM、EPROM、EEPROM、Flash Memory
- 非易失存储器的应用
  Uses for Nonvolatile Memories
  - 存储程序固件(BIOS、磁盘控制器、网卡、图形加速卡、安全子系统等)
    Firmware programs stored in a ROM (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,…)
  - 固态硬盘（作为机械硬盘的替代，应用于智能手机、平板电脑、笔记本电脑等电子产品中）
    Solid state disks (replace rotating disks in thumb drives, smart phones, tablets, laptops,…)

本章内容

Topic

- 存储技术
  Storage technologies
  - 随机访问存储器
    Random-Access Memory
  - 磁盘存储
    Disk Storage
  - 固态硬盘
    Solid State Disks
  - 存储技术趋势
    Storage Technology Treads
- 局部性原理
  Locality of reference
- 存储器层次结构
  The memory hierarchy

磁盘里面有什么?
What's Inside A Disk Drive?



电气结构 （包含处理器和存储器！）
Electronics
(including a processor and memory!)

图片来自希捷
Image courtesy of Seagate Technology

磁盘构造
Disk Geometry

- 磁盘由多个转盘组成，每个盘有两个面

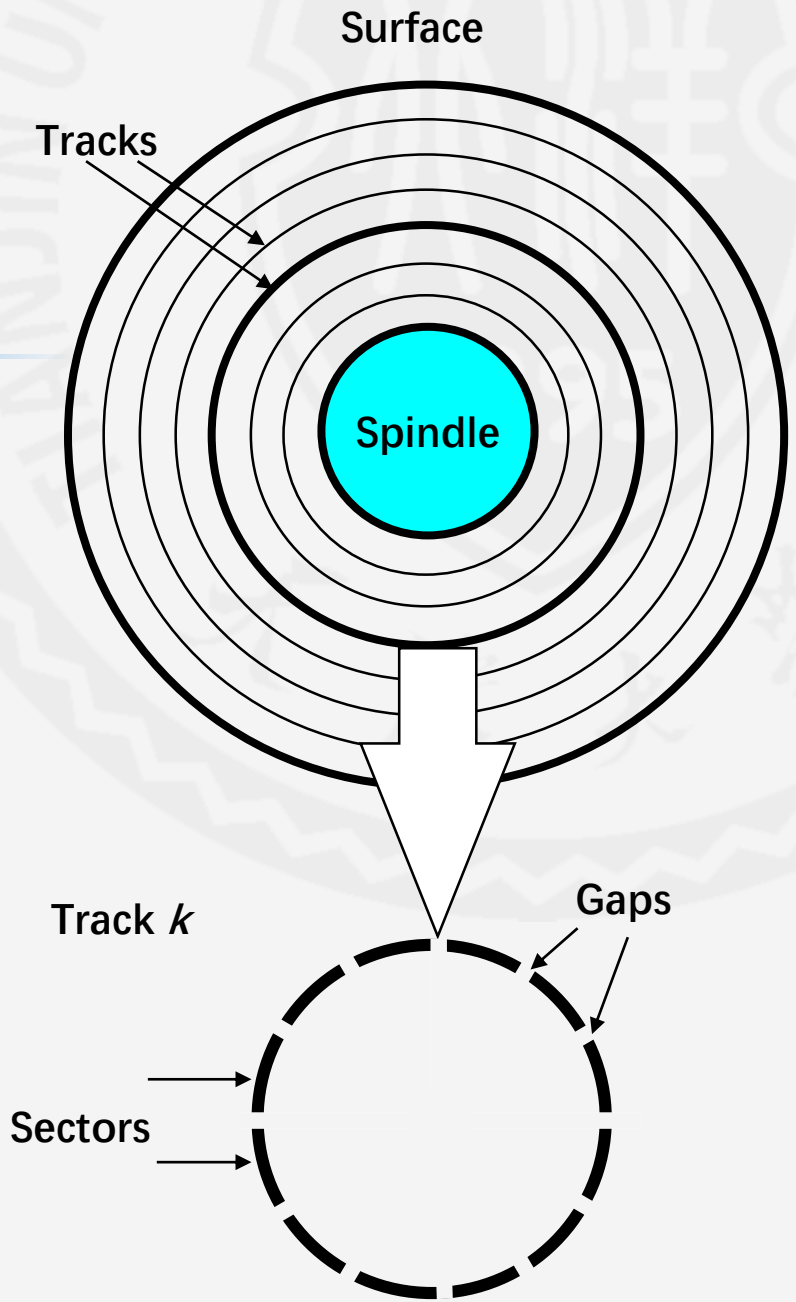
  Disks consist of platters, each with two surfaces

- 每个面由多个同心圆，称为磁道

  Each surface consists of concentric rings called tracks

- 每个磁道由多个扇区组成，扇区之间通过间隙分隔开

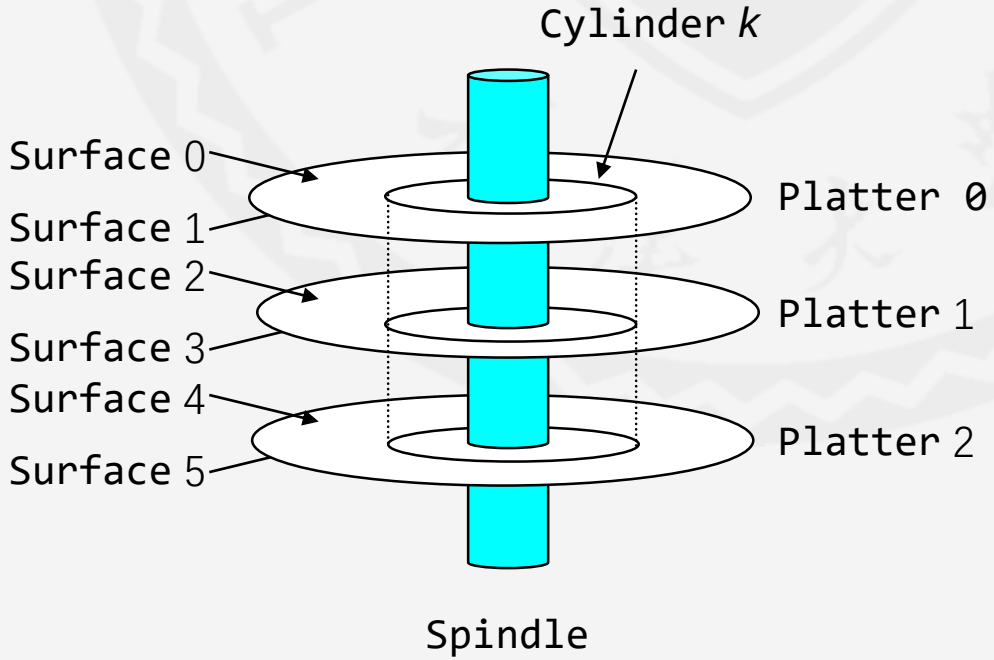
  Each track consists of sectors separated by gaps



磁盘构造（多盘片视角）
Disk Geometry(Muliple-Platter View)

- 柱面：所有盘片的面上到主轴中心距离相等的磁道的集合

  A cylinder is the collection of tracks on all the surfaces that are equidistant from the center of the spindle.



磁盘的容量
Disk Capacity

- 容量：能被存储的最大位数
  Capacity: maximum number of bits that can be stored
  - 厂商一版使用 GB 作为容量单位（非标准单位）
    Vendors express capacity in units of gigabytes (GB)

1 GB = 10^9 Bytes

- 决定容量的因素：
  Capacity is determined by these technology factors:
  - 记录密度（位/英寸）：每英寸的磁道上可以放入的位数
    Recording density (bits/in): number of bits that can be squeezed into a 1 inch segment of a track.
  - 磁道密度（磁道/英寸）：每英寸半径内的磁道数
    Track density (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment
  - 面密度（位/平方英寸）：记录密度与磁道密度的乘积
    Areal density (bits/in2): product of recording and track density

多区记录
Multiple Zone Recording

- 现代磁盘将磁道划分为不相交的子集，称为记录区

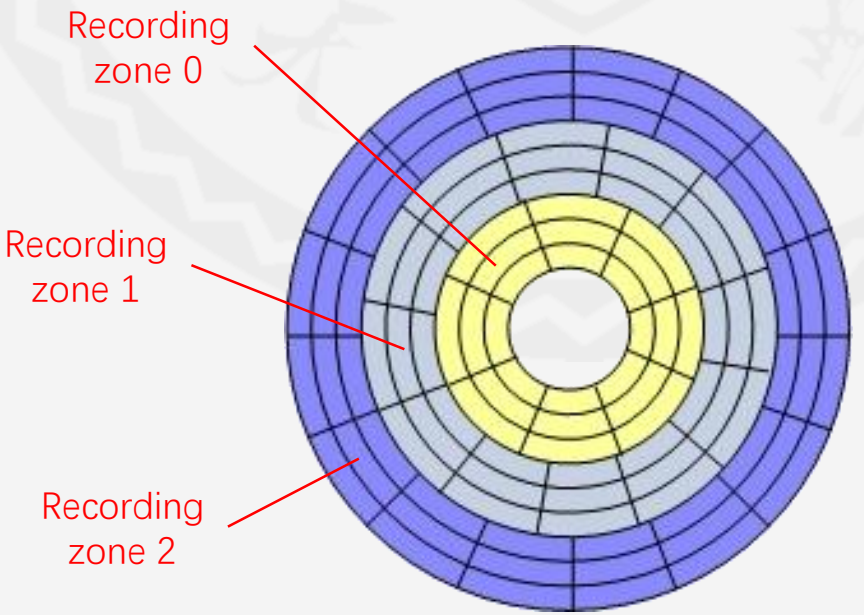
  Modern disks partition tracks into disjoint subsets called recording zones

  - 区域中的每个磁道具有相同数量的扇区，由最内侧的磁道的周长决定

    Each track in a zone has the same number of sectors, determined by the circumference of innermost track

  - 记录区之间的扇区磁道比是不同的

    Each zone has a different number of sectors/trac



计算磁盘容量
Computing Disk Capacity

Capacity = (# bytes/sector) x (avg. # sectors/track) x (# tracks/surface) x (# surfaces/platter) x (# platters/disk)

Example:

512 bytes/sector
300 sectors/track (on average)
20,000 tracks/surface
2 surfaces/platter
5 platters/disk

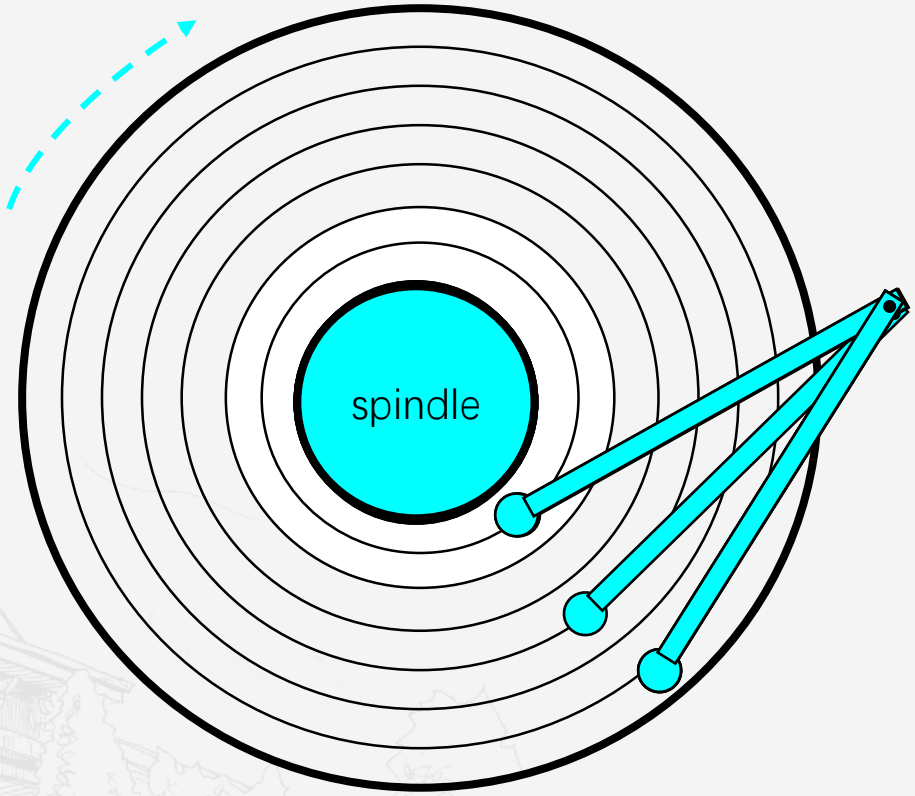
Capacity = 512 x 300 x 20000 x 2 x 5

= 30,720,000,000

= 30.72 GB

磁盘操作（单盘片视角）
Disk Operation (Single-Platter View)

磁盘的盘片以固定的转速旋转
The disk surface spins at a fixed rotational rate

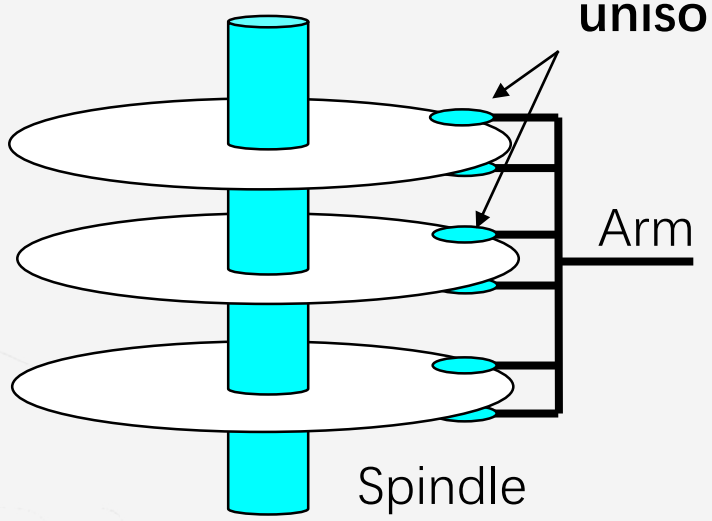


读/写头连接到臂的末端，并在薄的空气垫上掠过磁盘表面
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

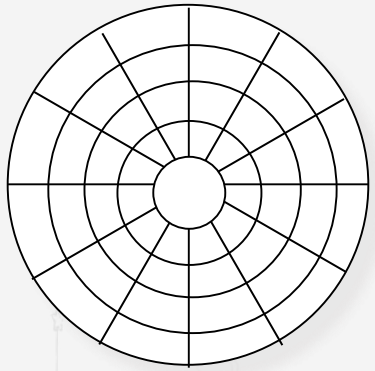
沿着径向移动，臂可以将读/写头定位在任何磁道上
By moving radially, the arm can position the read/write head over any track.

磁盘操作（多盘片视角）
Disk Operation (Multi-Platter View)

读写头步调一致的从一个扇区移动到另一个扇区
Read/write heads move in unison from cylinder to cylinder



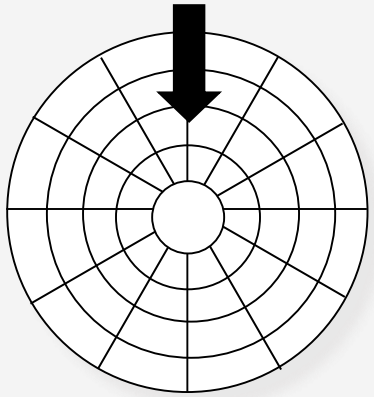
磁盘结构（单盘片俯视）
Disk Structure (top view of single platter)



面由多个磁道组成
Surface organized into tracks

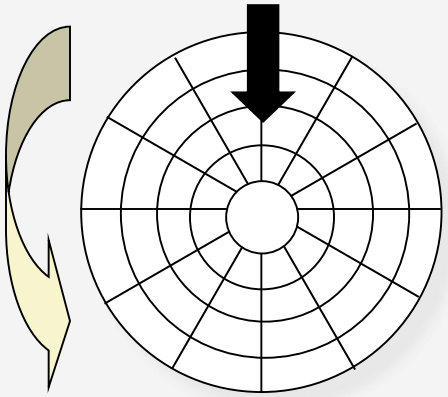
磁道被划分为多个扇区
Tracks divided into sectors

磁盘访问
Disk Access



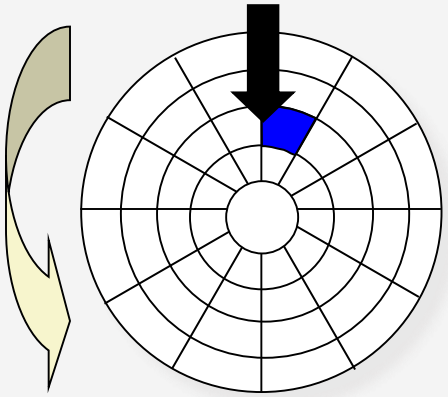
读写头位于磁道上方
Head in position above a track

磁盘访问
Disk Access



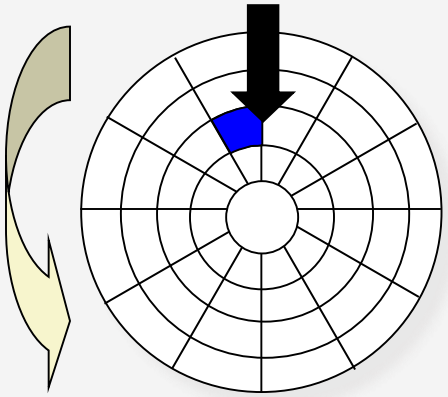
逆时针旋转
Rotation is counter-clockwise

磁盘访问——读
Disk Access - Read



即将读蓝色扇区
About to read blue sector

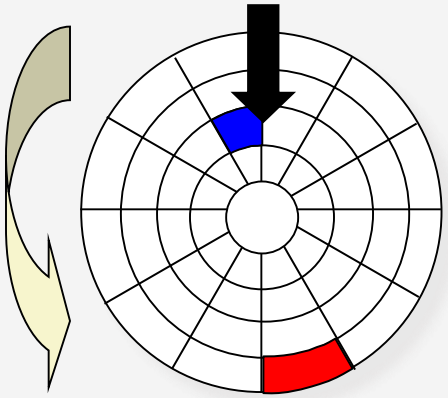
磁盘访问——读
Disk Access - Read



After BLUE read

蓝色扇区完成度操作
After reading blue sector

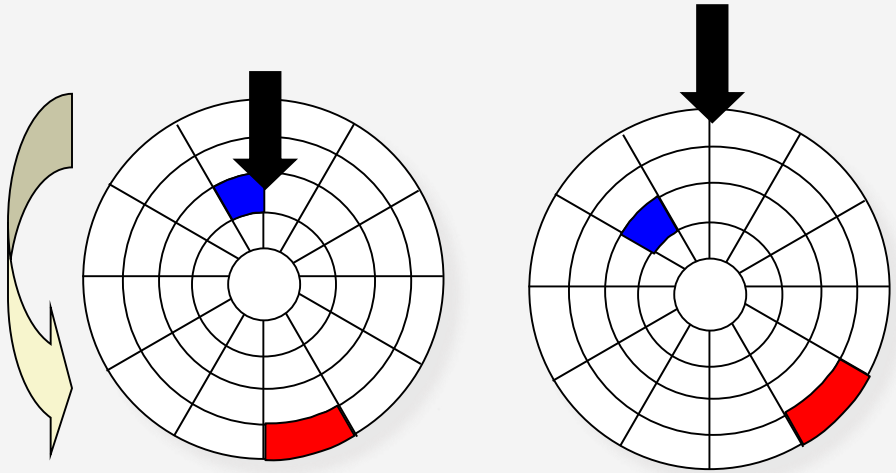
磁盘访问——读
Disk Access - Read



After BLUE read

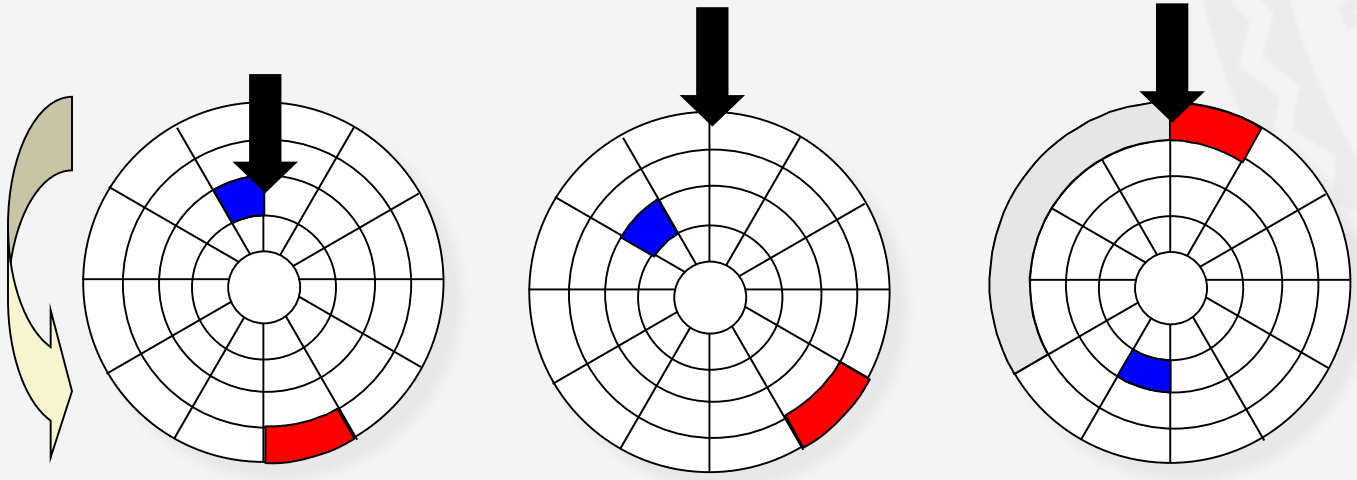
下一个需要读红色扇区
Red request scheduled next

磁盘访问——寻道
Disk Access - Seek



寻找红色扇区所在磁道
Seek to red's track

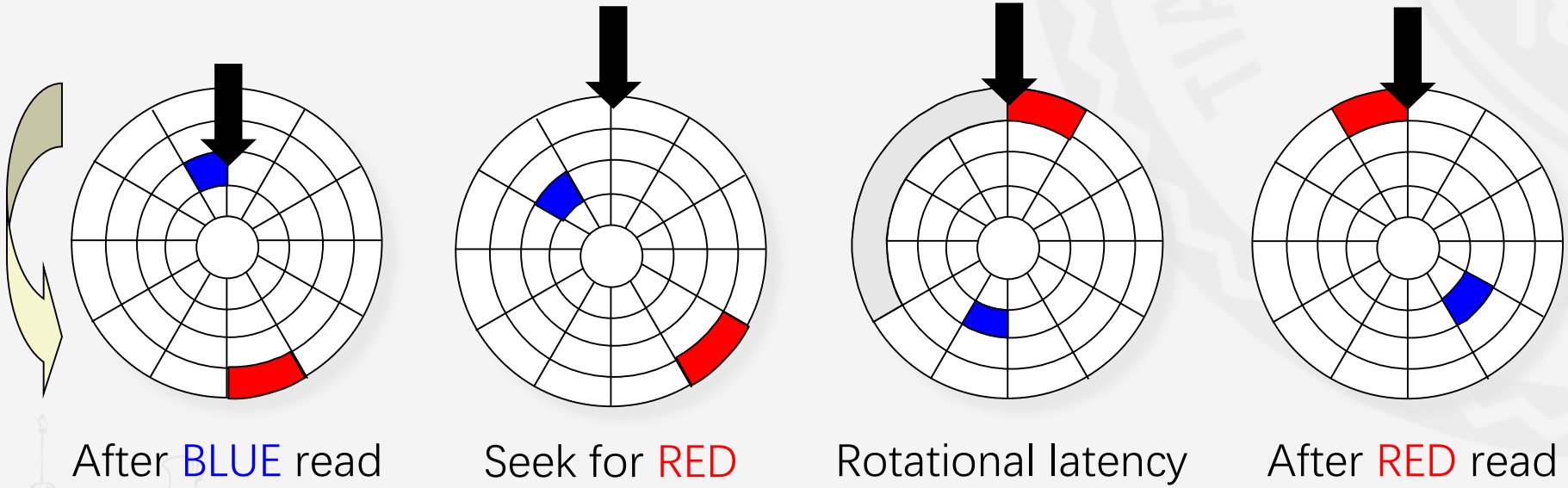
磁盘访问——旋转延迟
Disk Access - Rotational Latency



等待红色扇区旋转到读写头

Wait for red sector to rotate around

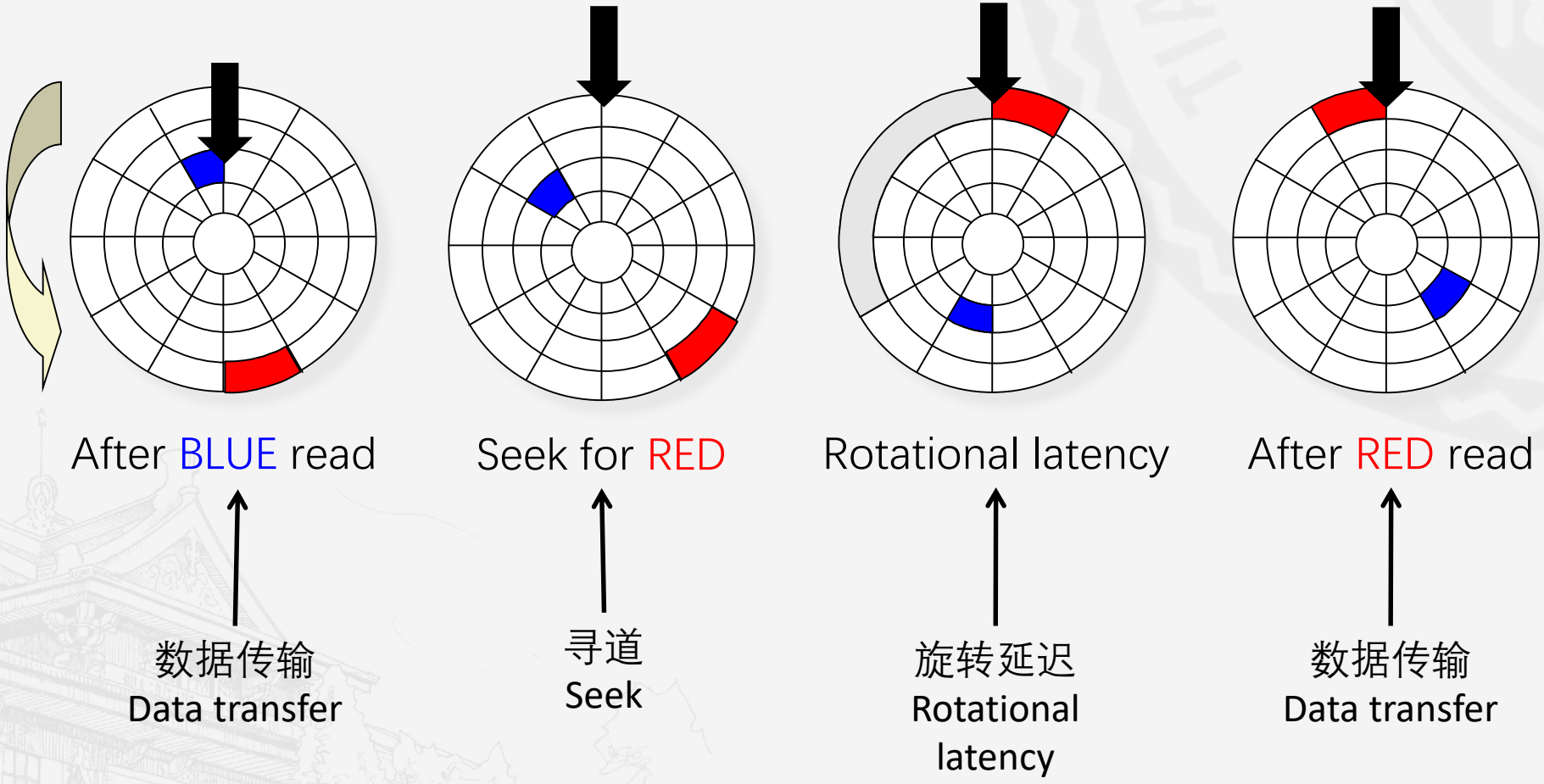
磁盘访问——读
Disk Access - Read



After BLUE read　Seek for RED　Rotational latency　After RED read

完成红色扇区的读取
Complete read of red

磁盘访问——操作时间的构成
Disk Access – Service Time Components



存储技术

Storage technologies

磁盘访问时间
Disk Access Time

RPM：**R**evolution **P**er **M**inutes
每分钟转数

- 访问某目标扇区的平均时间近似表示为：
  Average time to access some target sector approximated by :
  - $T_{access} = T_{avg\ seek} + T_{avg\ rotation} + T_{avg\ transfer}$
- 寻道时间 $T_{avg\ seek}$
  Seek time
  - 将读写头移动到目标扇区所在柱面的时间
    Time to position heads over cylinder containing target sector
  - 典型值：3 - 9ms
    Typical $T_{avg\ seek}$ = 3 - 9 ms
- 旋转延迟 $T_{avg\ rotation}$
  Rotational latency
  - 等待目标扇区第一个比特通过读写头所花费的时间
    Time waiting for first bit of target sector to pass under r/w head
  - $T_{avg\ rotation}$ = 1/2 x 1/RPMs x 60 sec/1 min
  - 典型每分钟转速 = 7200
    Typical Revolution Per Minutes = 7200

- 传输时间 $T_{avg\ transfer}$
  Transfer time
  - 读取扇区内所有数据的时间
    Time to read the bits in the target sector
  - 等待目标扇区第一个比特通过读写头所花费的时间
    $T_{avg\ transfer}$ = 1/RPM x 1/(avg # sectors/track) x 60 secs/1 min

举例：磁盘访问时间
Disk Access Time Example

- 已知：
  Given:
  - 转速 = 7200 RPM
    Rotational rate = 7,200 RPM
  - 平均寻道时间 = 9 ms
    Average seek time = 9 ms
  - 平均每个扇区的磁道数 = 400
    Avg # sectors/track = 400
- 可得：
  Derived:
  - $T_{avg\ rotation}$ = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms.
  - $T_{avg\ transfer}$ = 60/7200 RPM x 1/400 secs/track x 1000 ms/sec = 0.02 ms
  - T_{access} = 9 ms + 4 ms + 0.02 ms

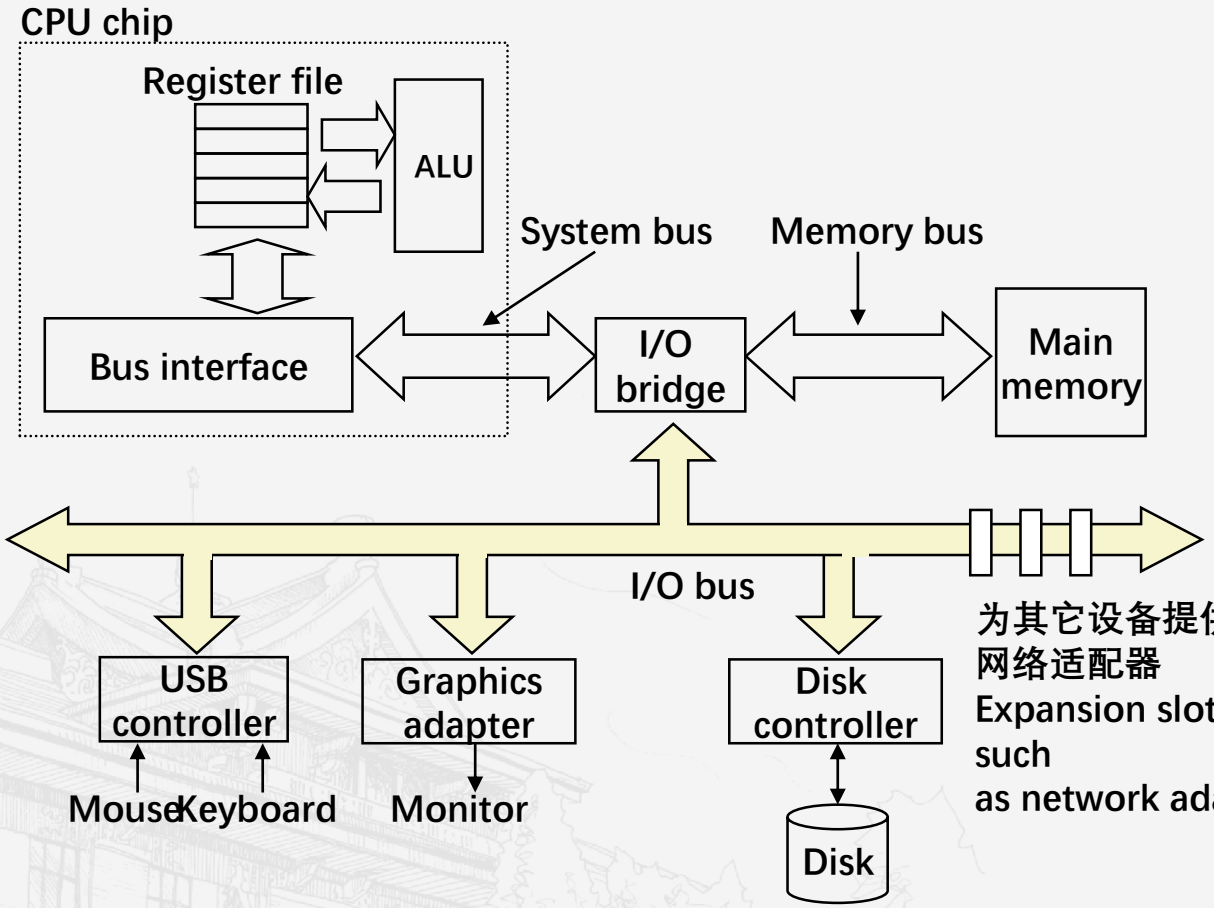
- 要点
  Important points:
  - 访问时间主要由寻道时间和旋转延迟决定
    Access time dominated by seek time and rotational latency
  - 定位到第一个扇区的第一个比特的时间开销是最大的，剩余时间几乎可以忽略不计
    First bit in a sector is the most expensive, the rest are free.
  - SRAM的访问时间是4ns/双字
    DRAM是60ns/双字
    SRAM access time is about 4 ns/doubleword
    DRAM about 60 ns
  - 磁盘比SRAM慢四万倍左右，比DRAM慢2500倍
    Disk is about 40,000 times slower than SRAM, 2,500 times slower then DRAM

逻辑磁盘块
Logical Disk Blocks

- 现代磁盘将复杂扇区构造呈现为一个简单的视图：
  Modern disks present a simpler abstract view of the complex sector geometry:
  - 可用扇区集合被建模成大小为B的逻辑块序列(0，1，2，...)
    The set of available sectors is modeled as a sequence of b-sized logical blocks (0, 1, 2, ...)
- 建立逻辑块与物理扇区之间的映射
  Mapping between logical blocks and actual (physical) sectors
  - 由磁盘控制器的硬件/固件维护这种映射关系
    Maintained by hardware/firmware device called disk controller
  - 将对逻辑块的请求转换为三元组（盘面、磁道、扇区）
    Converts requests for logical blocks into (surface, track, sector) triples
- 允许控制器为每个区域留出备用柱面
  Allows controller to set aside spare cylinders for each zone
  - 这也导致了“格式化容量”和“最大容量”之间存在差异
    Accounts for the difference in “formatted capacity” and “maximum capacity”.

计算机科学领域的任何问题都可以通过增加一个间接的中间层来解决。
All problems in computer science can be solved by another level of indirection.
——Butler Lampson, 1972

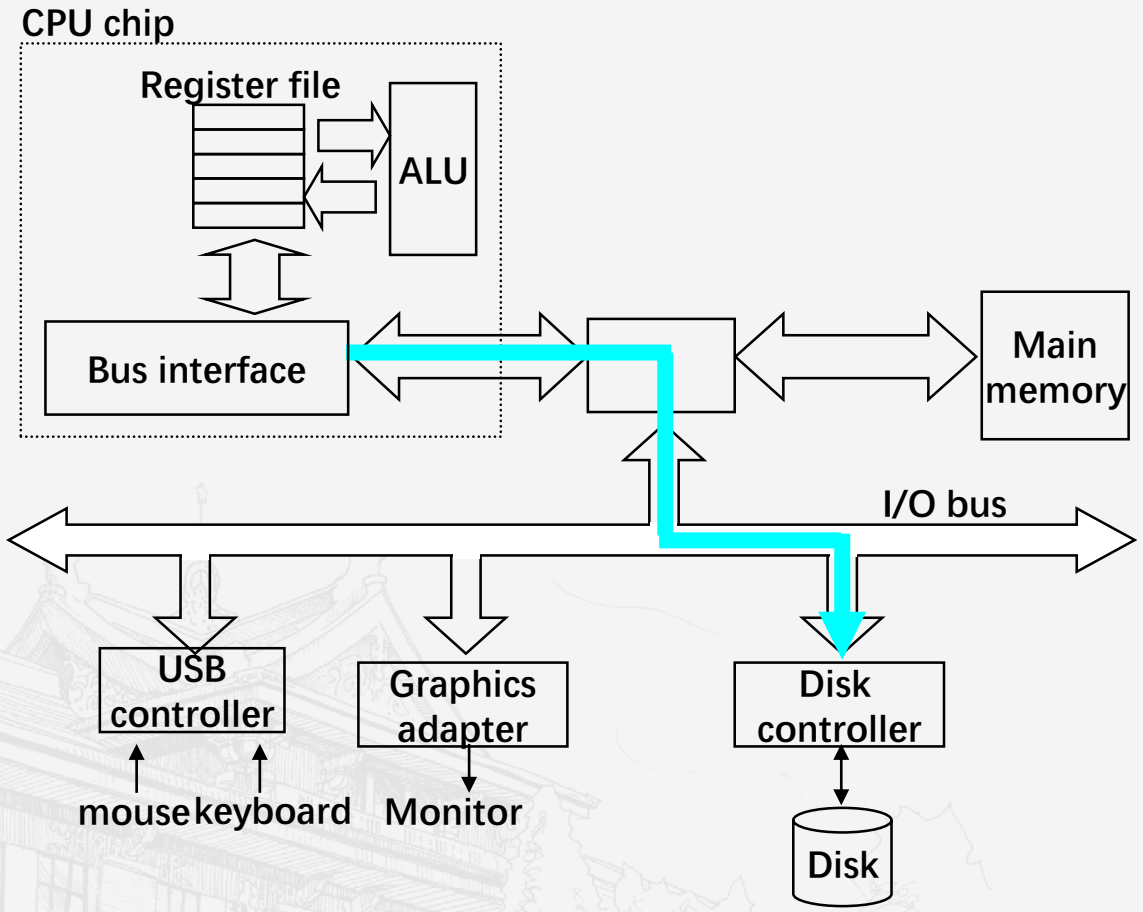
I/O总线
Logical Disk Blocks



I/O桥：用于实现高速总线与低速总线的协同工作
I/O Bridge: used to realize the cooperative work of high-speed bus and low-speed bus

为其它设备提供的扩展插槽，例如：网络适配器
Expansion slots for other devices such as network adapters.

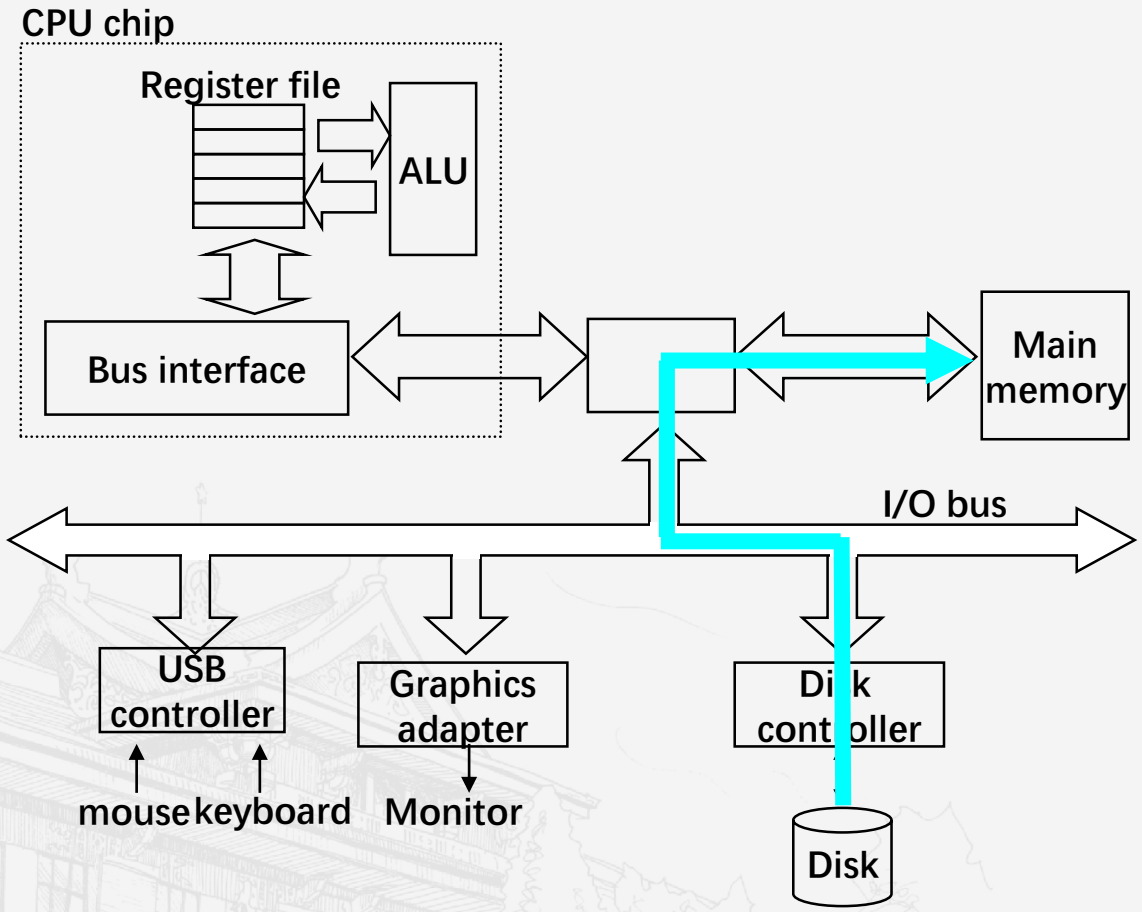
读一个磁盘扇区（1）
Reading a Disk Sector (1)



CPU通过向与磁盘控制器相关的**端口**（地址）写入命令、逻辑块号和目标存储器地址来开启磁盘读取。

I/O CPU initiates a disk read by writing a command, logical block number, and destination memory address to a **port** (address) associated with disk controller.

读一个磁盘扇区（2）
Reading a Disk Sector (2)



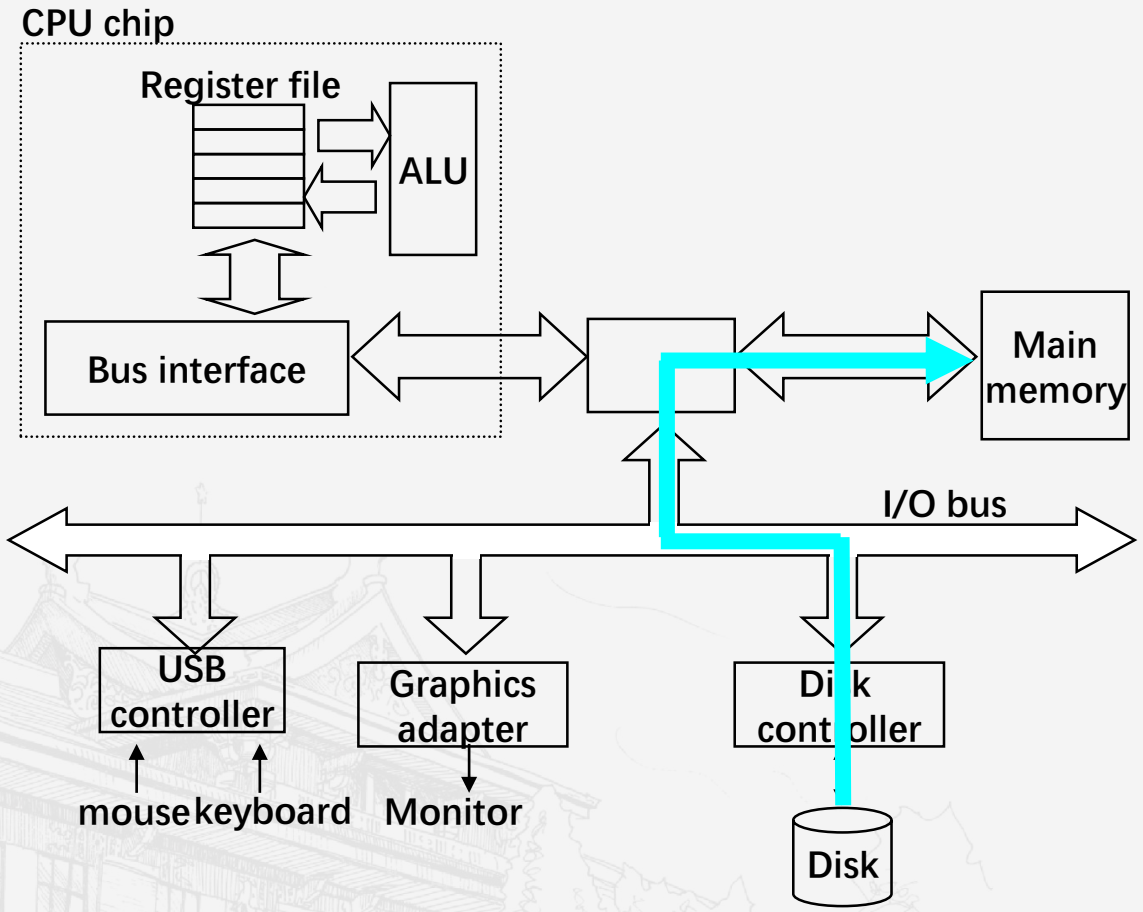
磁盘控制器读取扇区信息，并启动DMA将磁盘数据向主存的传输。

Disk controller reads the sector and performs a DMA transfer into main memory.

DMA：Direct Memory Access

直接内存访问技术（绕开CPU）

读一个磁盘扇区（3）
Reading a Disk Sector (3)



当DMA传输完成时，磁盘控制器用一个中断通知CPU（即在CPU上触发一个特殊的中断引脚）

When the DMA transfer completes, the disk controller notifies the CPU with an interrupt (i.e., asserts a special “interrupt” pin on the CPU)

中断：见“异常”一章

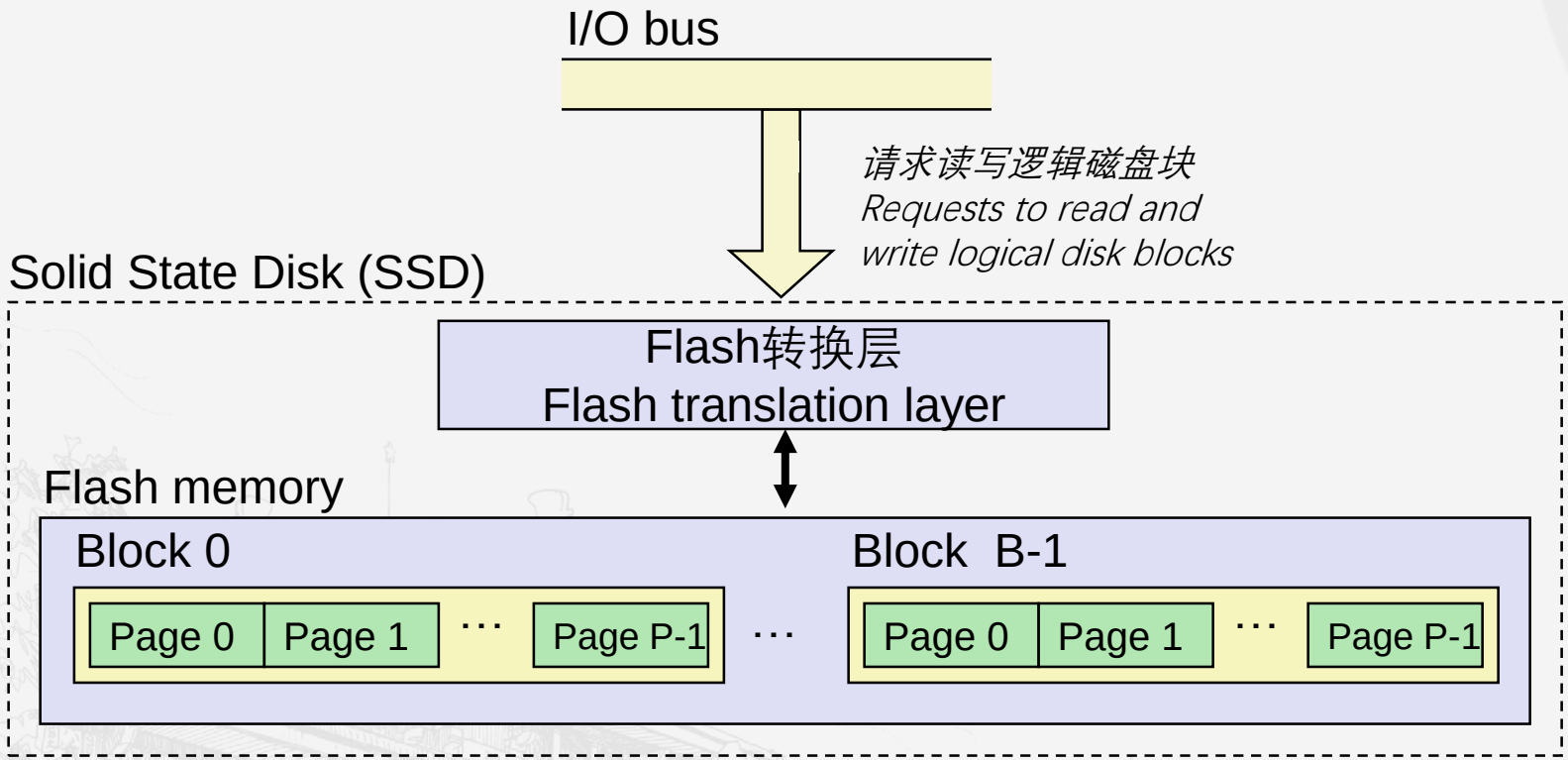
Interrupt：see Chapter “Exceptions”

本章内容

Topic

- 存储技术
  Storage technologies
  - 随机访问存储器
    Random-Access Memory
  - 磁盘存储
    Disk Storage
  - 固态硬盘
    Solid State Disks
  - 存储技术趋势
    Storage Technology Treads
- 局部性原理
  Locality of reference
- 存储器层次结构
  The memory hierarchy

固态硬盘（SSD）
Solid State Disks (SSDs)



- 页：512B – 4KB
  Pages: 512B to 4KB
- 块：32 – 128 个页
  Blocks: 32 to 128 pages
- 数据读写以页为单位
  Data read/written in units of pages
- 页仅能才擦除后被写入
  Page can be written only after its block has been erased
- 通常擦除操作以块为单位
  Commonly, Data erased in units of blocks
- 块的寿命大约是100,000次写入
  A block wears out after 100,000 repeated writes

固态硬盘性能特点
SSD Performance Characteristics

- 为什么随机写慢?
  Why are random writes so slow?
  - 擦除一个块的时间很慢（将近1ms）
    Erasing a block is slow (around 1 ms)
  - 写入页将触发当前块中所有有效页的一次复制
    Write to a page triggers a copy of all useful pages in the block
    - 找到一个用过的块（新块）并擦除它
      Find an used block (new block) and erase it
    - 将页面写入新块
      Write the page into the new block
    - 块将其他页面从旧块复制到新块
      Copy other pages from old block to the new block

顺序读吞吐量		顺序写吞吐量	
Sequential read tput	250 MB/s	Sequential write tput	170 MB/s
随机读吞吐量		随机写吞吐量	
Random read tput	140 MB/s	Random write tput	14 MB/s
随机读访问时间		随机写访问时间	
Rand read access	30 us	Random write access	300 us

固态硬盘与磁盘间的选择
SSD Tradeoffs vs. Rotating Disks

- 优点（固态硬盘）
  Advantages
  - 没有机械结构 → 更快，更低能耗，更结实
    No moving parts → faster, less power, more rugged
- 缺点
  Disadvantages
  - 寿命问题
    Have the potential to wear out
    - 可以通过Flash转换层（FTL）的负载均衡来缓解
      Mitigated by “wear leveling logic” in flash translation layer an used block (new block) and erase it
    - 例如：Intel X25 保证 1PB（10^{15} byte）数据的随机读写寿命
      E.g. Intel X25 guarantees 1 petabyte (1015 bytes) of random writes before they wear out
  - 在2010年，大约每字节的成本是（磁盘）的100倍
    In 2010, about 100 times more expensive per byte

- 应用
  Applications
  - MP3播放器，智能手机，笔记本电脑，桌面，甚至服务器上也开始应用
    MP3 players, smart phones, laptops, desktops and servers

本章内容

Topic

- 存储技术
  Storage technologies
  - 随机访问存储器
    Random-Access Memory
  - 磁盘存储
    Disk Storage
  - 固态硬盘
    Solid State Disks
  - 存储技术趋势
    Storage Technology Treads
- 局部性原理
  Locality of reference
- 存储器层次结构
  The memory hierarchy

存储技术趋势
Storage Trends

SRAM

Metric	1980	1985	1990	1995	2000	2005	2010	*2010:1980*
$/MB	19,200	2,900	320	256	100	75	60	*320*
access (ns)	300	150	35	15	3	2	1.5	*200*

DRAM

Metric	1980	1985	1990	1995	2000	2005	2010	*2010:1980*
$/MB	8,000	880	100	30	1	0.1	0.06	*130,000*
access (ns)	375	200	100	70	60	50	40	*9*
typical size (MB)	0.064	0.256	4	16	64	2,000	8,000	*125,000*

Disk

Metric	1980	1985	1990	1995	2000	2005	2010	*2010:1980*
$/MB	500	100	8	0.30	0.01	0.005	0.0003	*1,600,000*
access (ms)	87	75	28	10	8	*4*	*3*	*29*
typical size (MB)	1	10	160	1,000	20,000	160,000	1,500,00	*1,500,000*

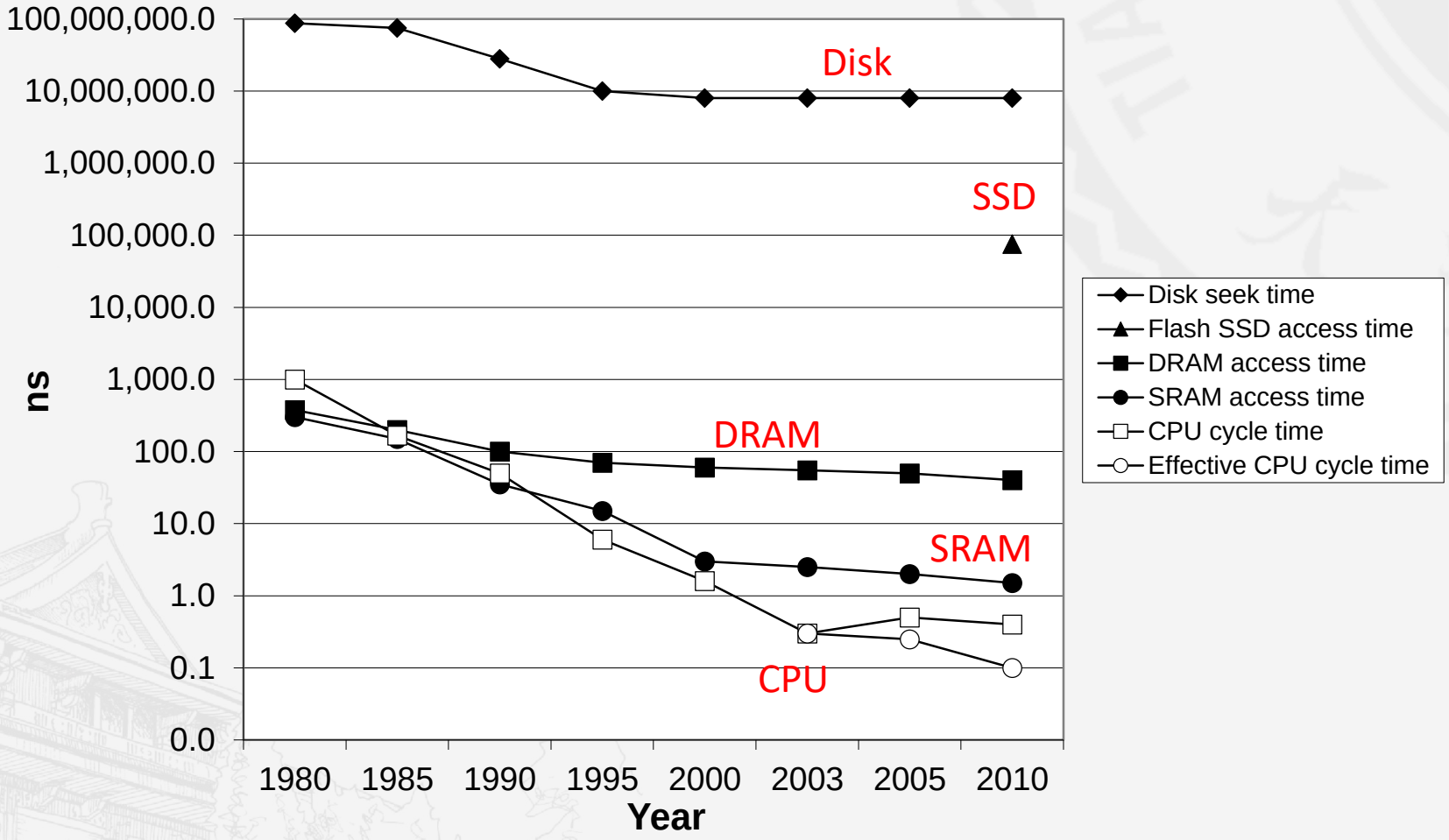
CPU时钟速率
CPU Clock Rates

计算机发展史上的拐点，当设计师撞上“功耗墙”
Inflection point in computer history when designers hit the “Power Wall”

	1980	1990	1995	2000	2003	2005	2010	*2010:1980*
CPU	8080	386	Pentium	P-III	P-4	Core 2	Core i7	---
Clock rate (MHz)	1	20	150	600	3300	2000	2500	2500
Cycle time (ns)	1000	50	6	1.6	0.3	0.50	0.4	2500
Cores	1	1	1	1	1	2	4	4
Effective cycle time (ns)	1000	50	6	1.6	0.3	0.25	0.1	10,000

CPU与内存之间的性能“鸿沟”
The CPU-Memory Gap

DRAM，磁盘和CPU间性能差距越来越大
The gap widens between DRAM, disk, and CPU speeds.



本章内容

Topic

- 存储技术

  Storage technologies
- 局部性原理

  Locality of reference
- 存储器层次结构

  The memory hierarchy

局部性原理

Locality of reference

问题的提出：内存的访问速度慢
Issue: Memory access is slow

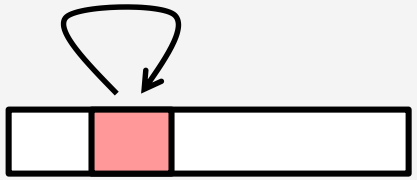
- 与CPU指令周期相比，DRAM的访问速度相当慢
  DRAM access is much slower than CPU cycle time
  - DRAM芯片的访问时间大约是30-50ns
    A DRAM chip has access times of 30-50ns
    - 将数据从主存传输至寄存器的时间更是至少要3倍于此
      Transferring from main memory into register can take 3X or more longer than that
  - CPU指令周期是亚纳秒级的，每次内存访问的时间大约是100多个CPU指令周期
    With sub-nanosecond cycles times, 100s of cycles per memory access
    - 而且这种差距还有扩大的趋势
      the gap grows over time
- 结果：内存的访问效率对计算机系统的性能有重要的影响
  Consequence: memory access efficiency crucial to performance
  - 程序中近乎1/3的指令时内存数据的加载和存储
    approximately 1/3 of instructions are loads or stores
  - 这个问题需要硬件设计者和程序员共同来解决
    both hardware and programmer have to work at it

局部性原理

Locality of reference

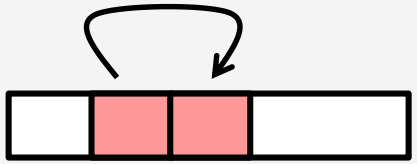
- 局部性原理：程序倾向于使用最近访问过的数据和指令，或是与之临近的数据和指令。
  Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently

- 时间局部性
  Temporal locality



  - 最近使用过的指令/数据很可能在不久的将来还会再次被使用
    Recently referenced items are likely to be referenced again in the near future

- 空间局部性
  Spatial locality



  - 临近地址的指令/数据在短期内可能会使用
    Items with nearby addresses tend to be referenced close together in time

局部性原理

Locality of reference

举例：程序的局部性特征
Locality Example

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

- 数据使用
  Data references
  - 连续使用数组元素（步长为1）
    Reference array elements in succession (stride-1 reference pattern)
    空间局部性 Spatial locality
  - 每一次循环都会使用sum 变量
    Reference variable sum each iteration
    时间局部性 Temporal locality
- 指令使用
  Instruction references
  - 按照顺序使用指令
    Reference instructions in sequence
    空间局部性 Spatial locality
  - 通过循环反复使用指令
    Cycle through loop repeatedly
    时间局部性 Temporal locality

局部性原理

Locality of reference

局部性特征对程序性能影响的定性估计
Qualitative Estimates of Locality

- 注意：通过阅读代码对其局部性特征有一个定性的认知，对专业程序员而言是一种重要的能力
  Claim: Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer
- 问题：这些函数相对于数组a是否具有良好的局部性特征?
  Question: Does this function have good locality with respect to array a?

```
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;
}
```

```
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;
}
```

局部性原理

Locality of reference

举例：局部性特征
Locality Example

- 你能通过重新排列循环，使三维数组a实现在内存中步长为1的访问吗？（更好的空间局部性）

  Question: Can you permute the loops so that the function scans the 3-d array a with a stride-1 reference pattern (and thus has good spatial locality)?

```
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;
}
```

本章内容

Topic

- 存储技术

  Storage technologies
- 局部性原理

  Locality of reference
- 存储器层次结构

  The memory hierarchy

存储器层次结构

The memory hierarchy

- 硬件和软件都有一些基本且持久的特性：
  Some fundamental and enduring properties of hardware and software:
  - 快速存储技术单位字节的成本更高，容量更小，并且需要更多的功率（发热）
    Fast storage technologies cost more per byte, have less capacity, and require more power (heat!)
  - CPU与主存的性能差距拉大
    The gap between CPU and main memory speed is widening
  - 优秀的程序往往表现出良好的局部性特征
    Well-written programs tend to exhibit good locality

- 以上这些性质其实可以完美契合，相辅相成
  These fundamental properties complement each other beautifully

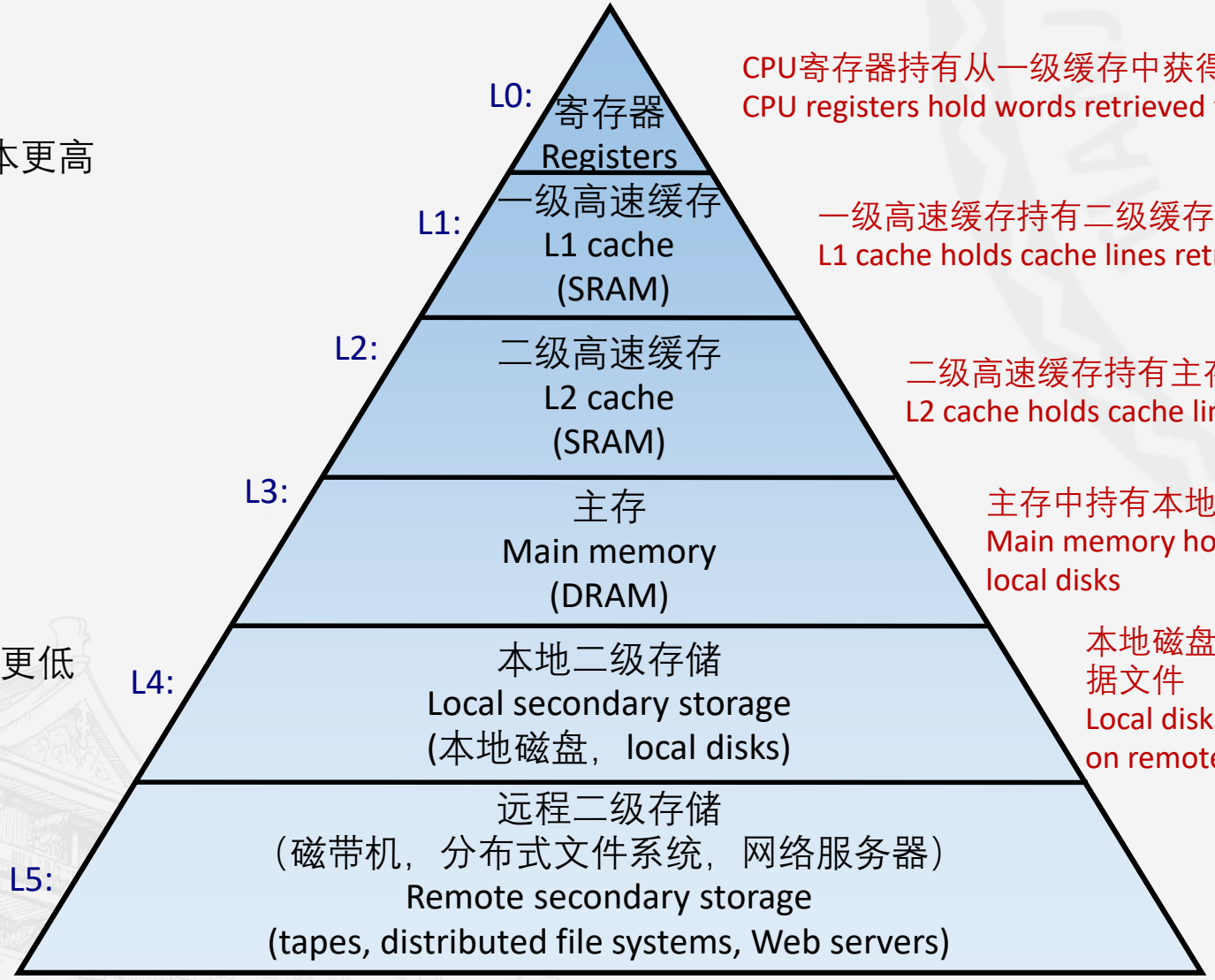
- 建构一种用与组织存储器与存储系统的方法，称之为存储器层次结构
  An approach for organizing memory and storage systems are suggested known as a memory hierarchy

存储器层次结构

The memory hierarchy

更小，
更快，
单位成本更高
Smaller,
faster,
costlier
per byte

更大，
更慢
单位成本更低
Larger,
slower,
cheaper
per byte



CPU寄存器持有从一级缓存中获得的数据
CPU registers hold words retrieved from L1 cache

一级高速缓存持有二级缓存中的数据块
L1 cache holds cache lines retrieved from L2 cache

二级高速缓存持有主存中的数据块
L2 cache holds cache lines retrieved from main memory

主存中持有本地磁盘上的数据块
Main memory holds disk blocks retrieved from local disks

本地磁盘中持有远程网络服务器上的数据文件
Local disks hold files retrieved from disks on remote network servers

存储器层次结构

The memory hierarchy

缓存
Cache

- 缓存：更小、更快速存储设备来暂时存储较大、较慢存储设备中数据子集

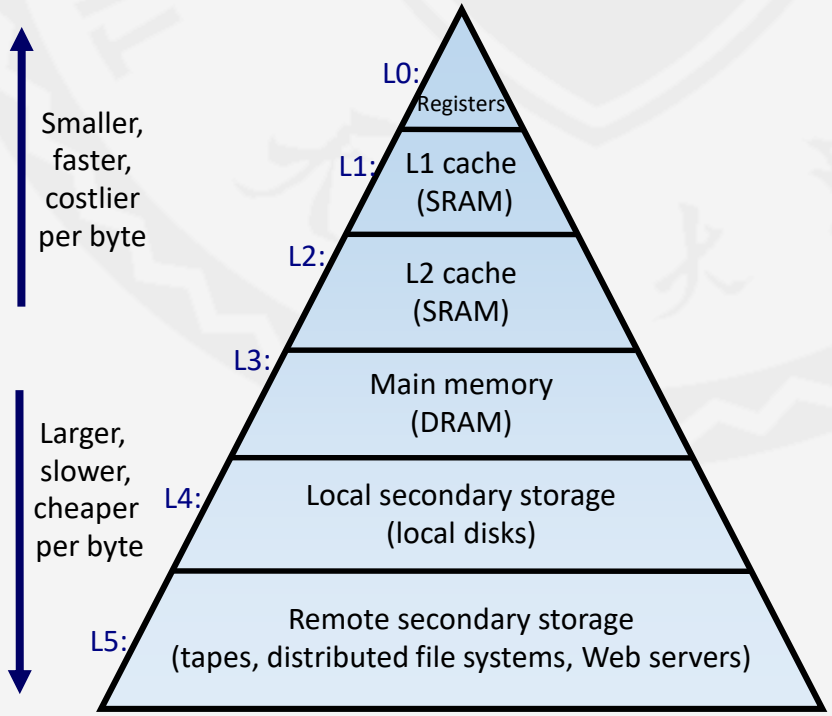
  Cache: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device

- 存储器层次结构的基本思想：

  Fundamental idea of a memory hierarchy

  - 对于更小、更快的第k层，缓存较大、较慢的第k+1层中的数据

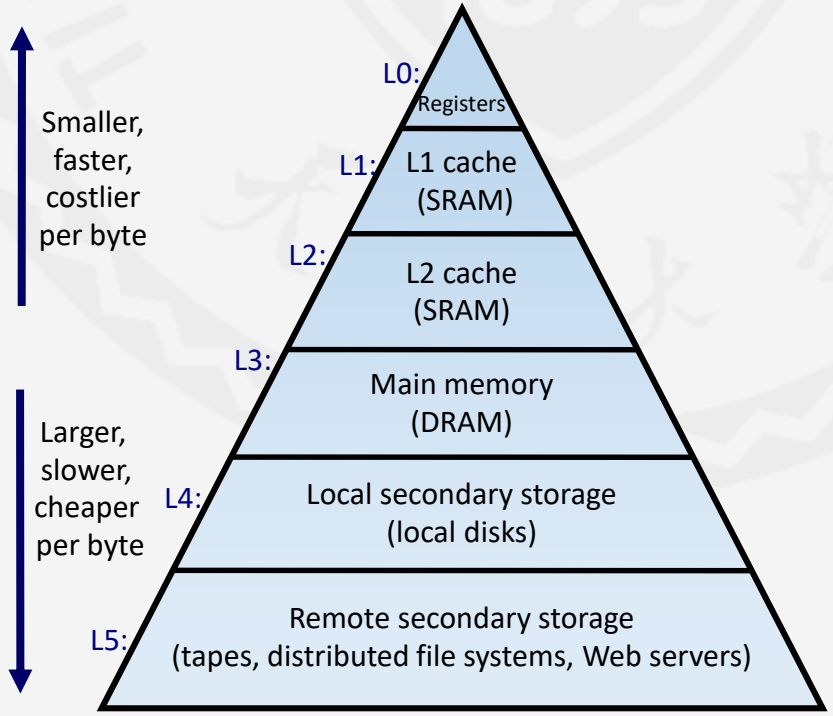
    For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1



存储器层次结构

The memory hierarchy

- 为什么存储器层次结构这种设计可以有效工作?
  Why do memory hierarchies work?
  - 程序的局部性原理：程序访问第k层的数据的概率要比第k+1层更大
    Because of locality, programs tend to access the data at level k more often than they access the data at level k+1
  - 因此，第k+1层的存储可以更慢，更大，每比特数据更便宜
    Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit
- **重要思想**：存储器层次结构构建了一个巨大的存储池，其成本接近于最底部的廉价存储，其速度接近于最顶部的快速存储
  Big Idea: The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.

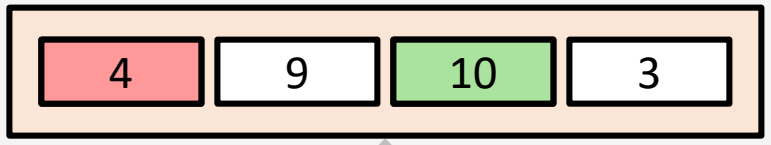


存储器层次结构

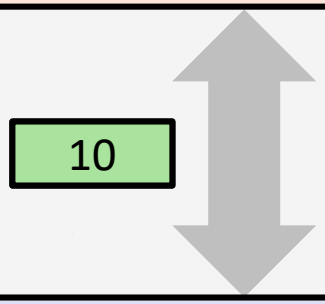
The memory hierarchy

缓存的基本概念
General Cache Concepts

高速缓存
Cache

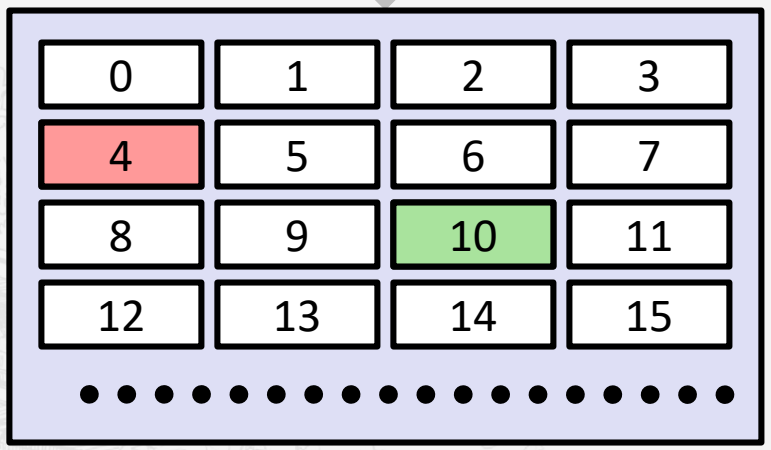


更小，更快，成本更高，缓存内存数据块的子集
Smaller, faster, more expensive
memory caches a subset of the blocks



数据复制时，以“块”作为基本传输单元
Data is copied in block-sized transfer units

主存
Memory

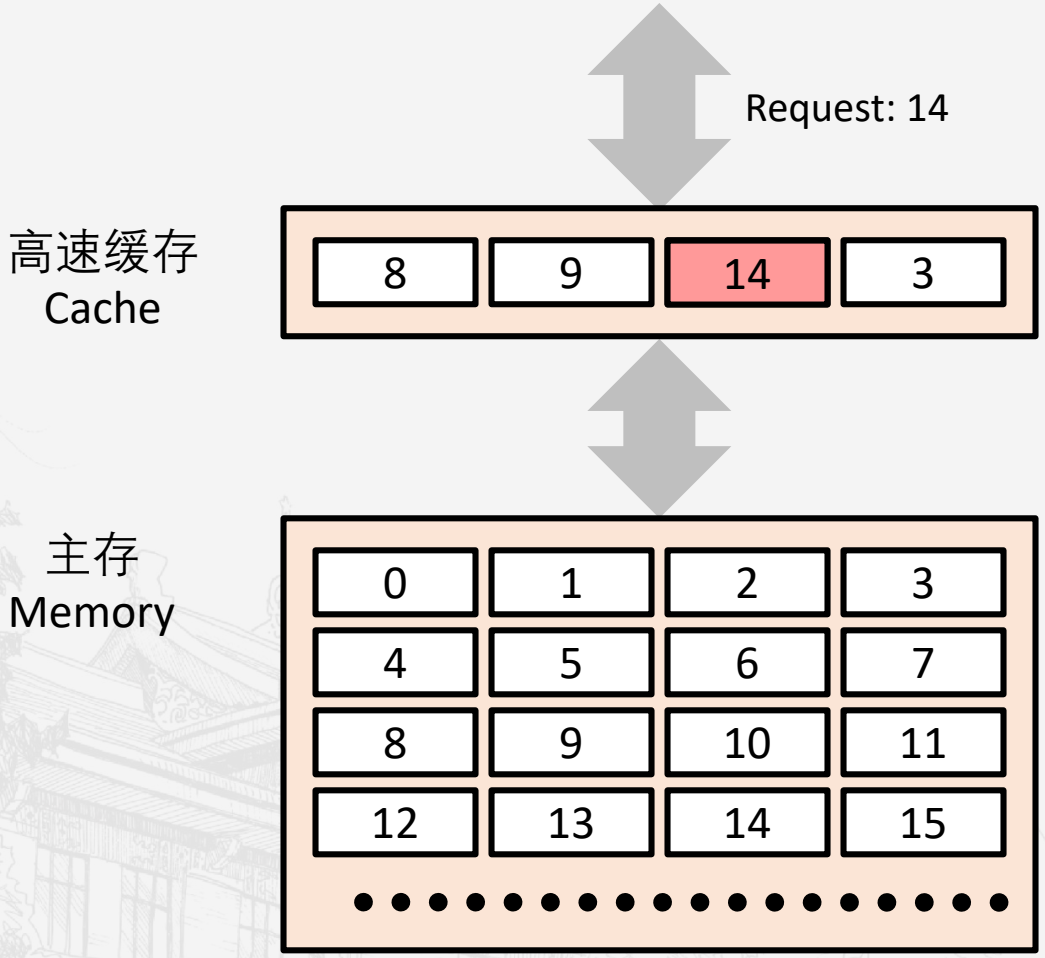


更大，更慢，成本更低的内存，在逻辑上内存可以被划分为多个数据块
Larger, slower, cheaper memory
viewed as partitioned into “blocks”

存储器层次结构

The memory hierarchy

缓存的基本概念：命中
General Cache Concepts：Hit



请求的数据位于块14中

Data in block #14 is needed

块14在高速缓存中：命中!

Block #14 is in cache: Hit!

存储器层次结构

The memory hierarchy

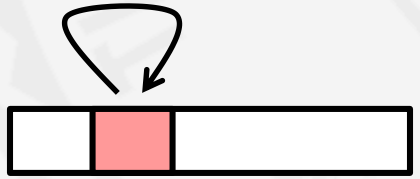
局部性特征如何导致缓存命中
How locality induces cache hits

- 时间局部性

  Temporal locality

  - 如果第二次到第N次都访问相同位置，将会导致命中

    2nd through Nth accesses to same location will be hits

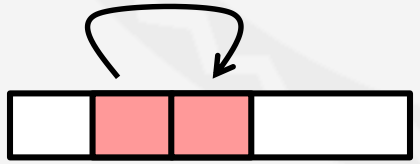


- 空间局部性

  Spatial locality

  - 缓存块中包含多个数据，第二次到第N次数据访问可能会命中在第一次访问数据所处的缓存快上

    Cache blocks contains multiple words, so 2nd to Nth word accesses can be hits on cache block loaded for 1st word



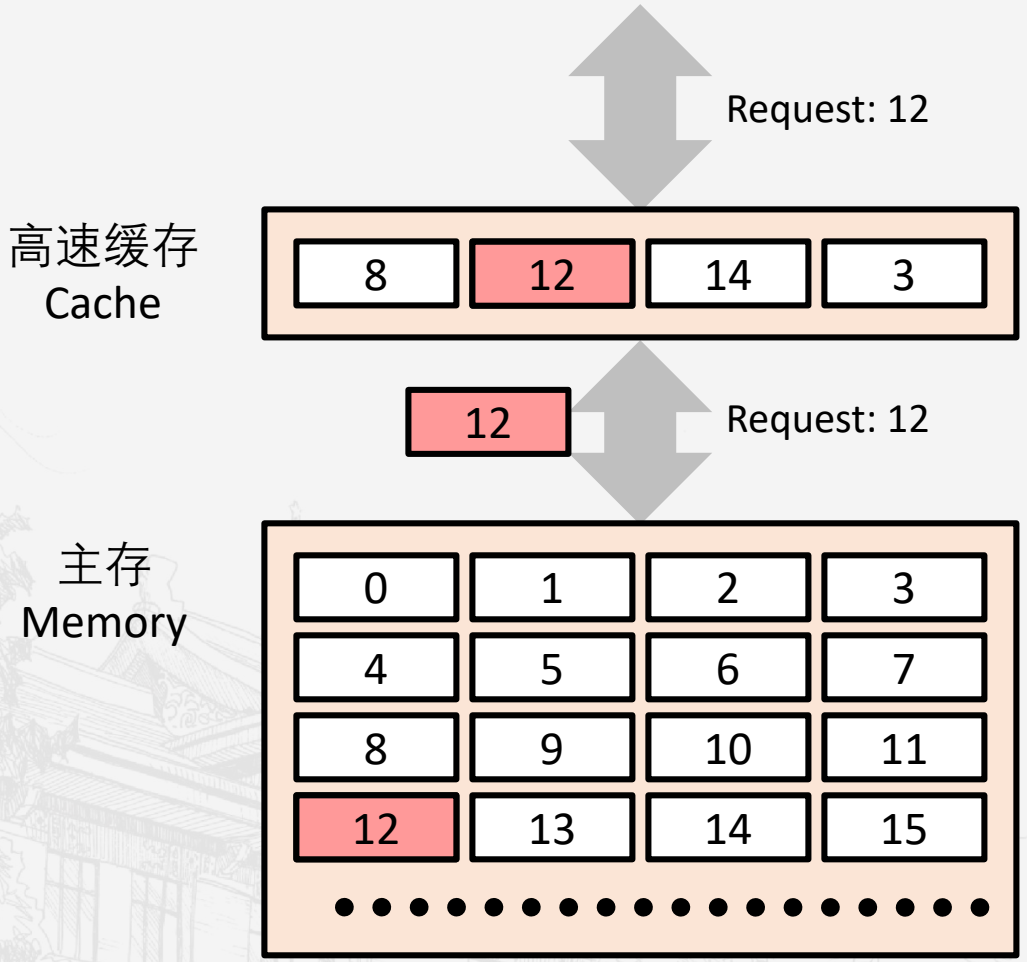
- DRAM的行缓冲器也是一个缓存的案例

  Row buffer in DRAM is another example

存储器层次结构

The memory hierarchy

缓存的基本概念：未命中（缺失）
General Cache Concepts：Miss



请求的数据位于块12中
Data in block #12 is needed

块12未在高速缓存中： 未命中!
Block #12 is not in cache: Miss!

从内存中取出块12
Block #12 is fetched from memory

块12存储在高速缓存中
Block #12 is stored in cache

- 块放置策略
  Placement policy:

决定了块#12存储在缓存中的那个位置
determines where #12 goes

- 块替换策略
  Replacement policy:

决定了高速缓存中那个块要被换出（牺牲）
determines which block gets evicted (victim)

各种类型的缓存未命中
Types of Cache Misses

- 冷（强制）未命中
  Cold (compulsory) miss
  - 对块的第一次访问必然会错过
    The first access to a block has to be a miss
  - 大多数冷未命中发生在（程序、系统）启动时，因为缓存是空的
    Most cold misses occur at the beginning, because the cache is empty
- 容量未命中
  Capacity miss
  - 当活跃的缓存块集合（工作集）大于缓存容量时发生
    Occurs when the set of active cache blocks (working set) is larger than the cache

各种类型的缓存未命中
Types of Cache Misses

- 冲突未命中
  Conflict miss
  - 大多数缓存机制都是将k+1层的块的位置映射在k层指定的一个子集中（有时甚至固定的某个位置）
    Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k
    - 例如：级别k+1的块i必须放在级别k的块(i mod 4)中
      E.g., Block i at level k+1 must be placed in block (i mod 4) at level k
  - 即使k层缓存足够大，但当k+1层的多个数据对象都映射到相同k级块时，也会发生冲突未命中
    Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block
    - 例如：按下列次序引用快0、8、0、8、0、8 ……，每次都会出现未命中
      E.g., Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time

存储器层次结构

The memory hierarchy

存储器层次结构中的各种缓存案例
Examples of Caching in the Hierarchy

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 bytes words	CPU core	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware
L1 cache	64-bytes block	On-Chip L1	1	Hardware
L2 cache	64-bytes block	On/Off-Chip L2	10	Hardware
Virtual Memory	4-KB page	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Disk firmware
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

总结
Summary

- CPU、内存和大规模存储设备之间的速度差距持续增大
  The speed gap between CPU, memory and mass storage continues to widen

- 优秀的程序展现出良好的局部性特征
  Well-written programs exhibit a property called locality

- 基于缓存机制的存储器层次结构，可以利用程序的局部性原理，降低存算间的性能差距
  Memory hierarchies based on caching close the gap by exploiting locality