计算机组织与系统结构

输入/输出系统与总线

I/O System and Buses

（第十五讲）

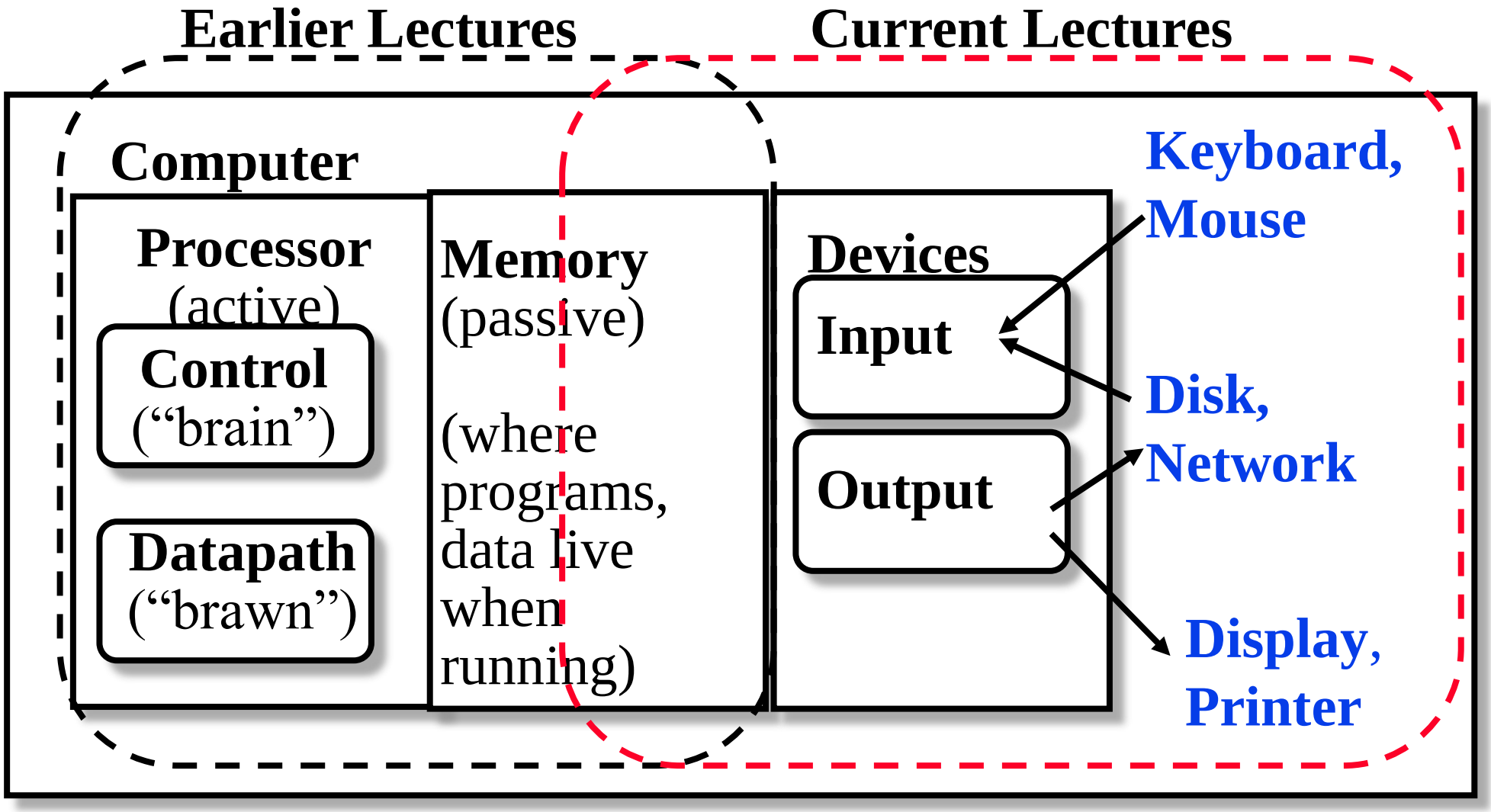
程 旭

2021.1.7

本讲提纲

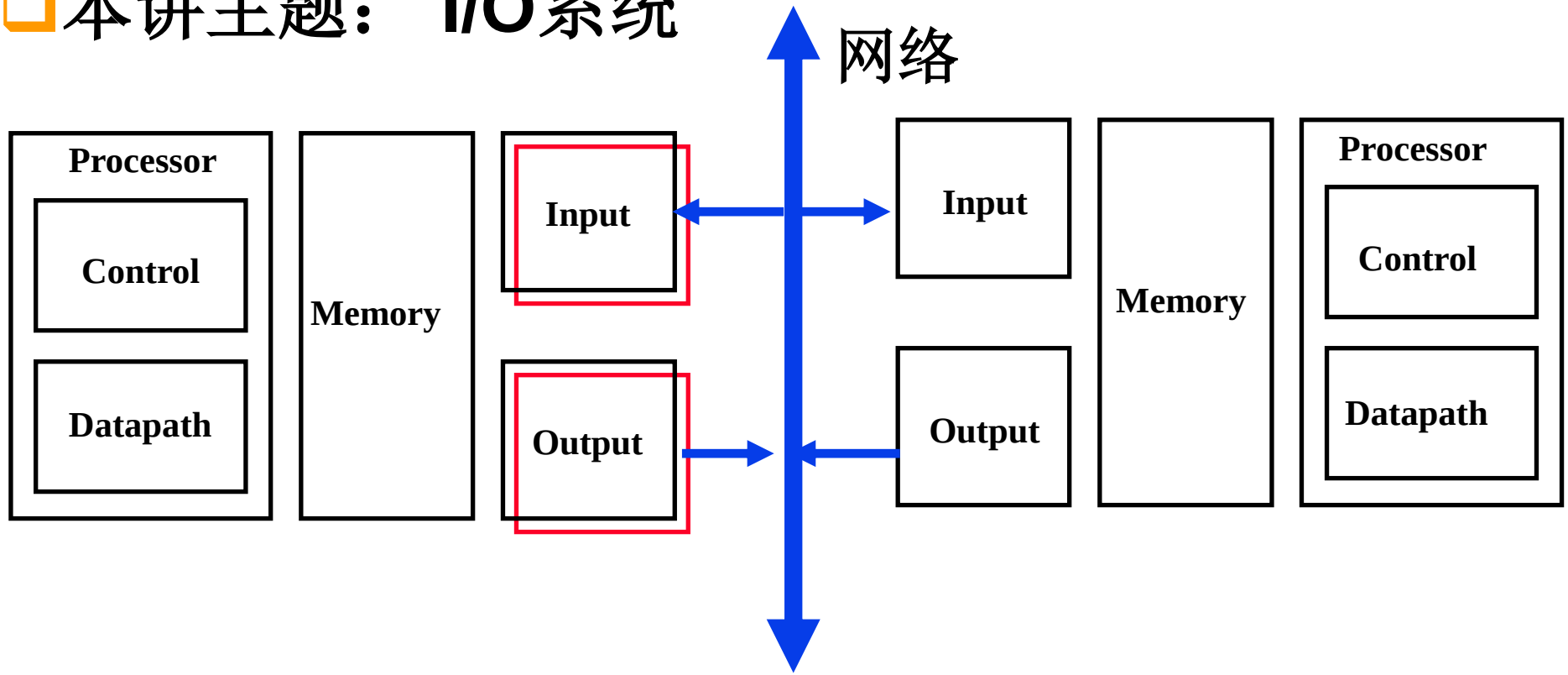
- I/O性能与测度
- I/O设备的特性
- 总线

5 components of any Computer



本讲内容的地位?

- 本讲主题： I/O系统



Motivation for Input/Output

- I/O is how humans interact with computers
- I/O gives computers long-term memory.
- I/O lets computers do amazing things:

MIT Media Lab
"Sixth Sense"
http://youtu.be/ZfV4R4x2SK0

- Computer without I/O like a car w/no wheels; great technology, but gets you nowhere

I/O Device Examples and Speeds

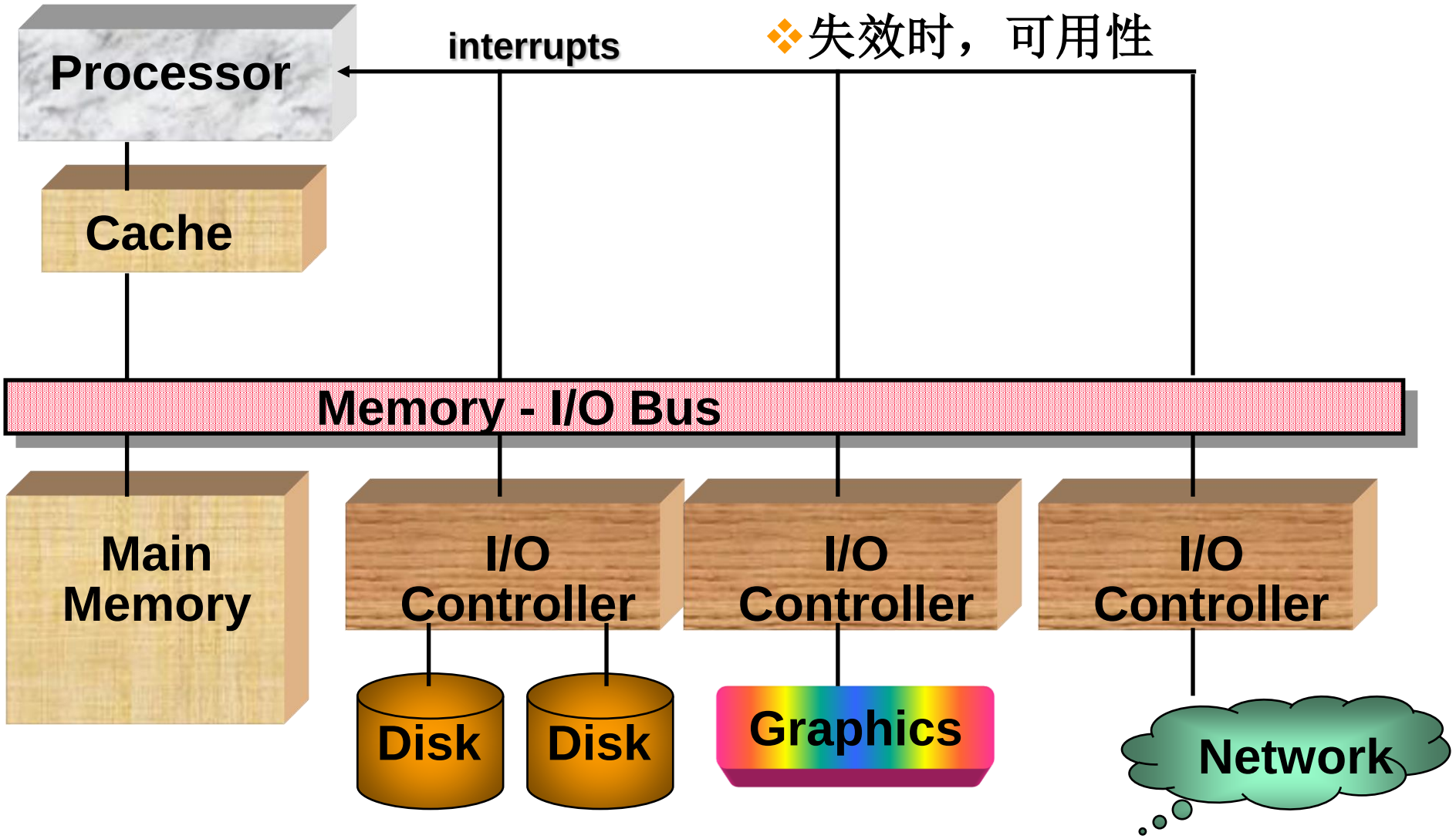
- I/O Speed: bytes transferred per second (from mouse to Gigabit LAN: 7 orders of magnitude!)

Device	Behavior	Partner	Data Rate(KBytes/s)
Keyboard	Input	Human	0.01
Mouse	Input	Human	0.02
Voice output	Output	Human	5.00
Floppy disk	Storage	Machine	50.00
Laser Printer	Output	Human	100.00
Magnetic Disk	Storage	Machine	10,000.00
Wireless Network	I or O	Machine	10,000.00
Graphics Display	Output	Human	30,000.00
Wired LAN Network	I or O	Machine	125,000.00

When discussing transfer rates, use 10^x

I/O 系统设计的论题

- 性能
- 可扩展性（Expandability）
- 失效时，可用性

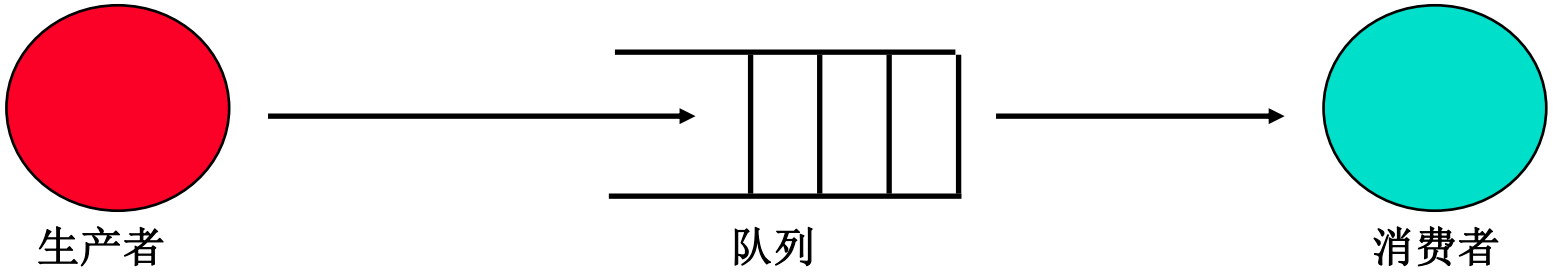


I/O 系统性能

- I/O 系统的性能与系统的许多部分有关 (受制于最弱的环节）：
  - CPU
  - 存储系统
    - 内部和外部cache
    - 主存
  - 底层互联（总线）
  - I/O控制器
  - I/O设备
  - I/O软件的速度 (操作系统)
  - 软件使用I/O设备的效率

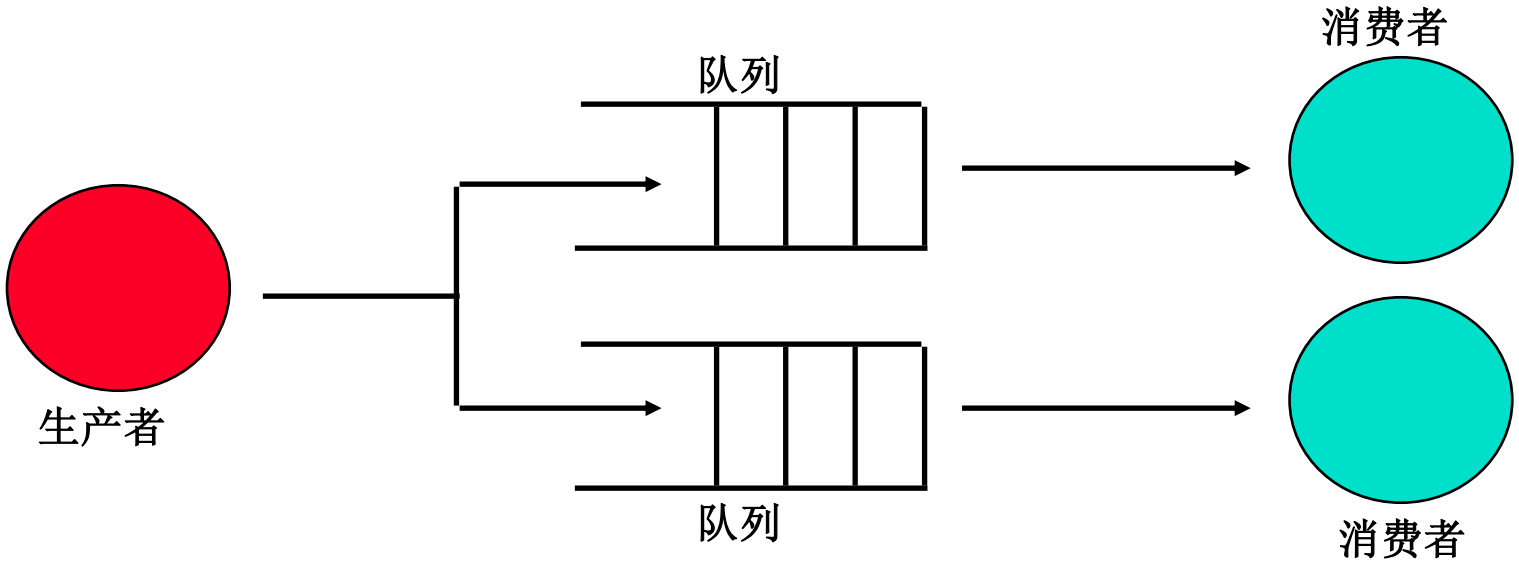
- 两种通用的性能指标：
  - 吞吐率：I/O带宽
  - 响应时间：时延

简化的生产者-消费者模型



- 吞吐率：
  - 消费者在单位之间内完成的任务数目
  - 为了达到最高可能的吞吐率：
    - 消费者从不停顿
    - 队列从不为空
- 响应时间：
  - 从 某一任务进入队列开始
  - 到 该任务被消费者完成为止
  - 为了最小化响应时间：
    - 队列应该为空
    - 服务者应该空闲

增大吞吐率



- 通常，可以采用以下方法改进吞吐率：
  - 在瓶颈问题上，增加硬件
  - 减少负载相关时延
- 相对而言，响应时间难以减少：
  - 最终受制于光速！ (但目前，距离光速的限制还很远！)

存贮技术的驱动力

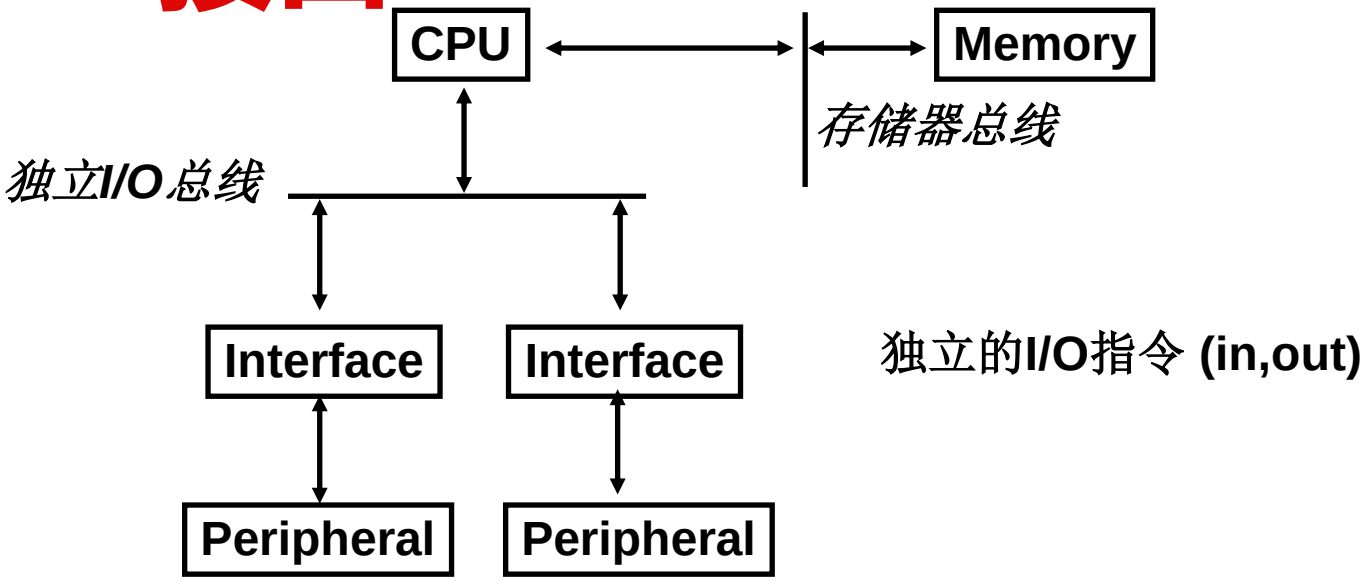
- 主流计算模式的驱动
  - 五十年代: 批处理 到 在线处理 的转变
  - 九十年代: 集中处理 到 普及计算 的转变
    - 计算机无处不在：电话、电子书籍、汽车、摄像机
    - 全球性光纤网络及无线网络

- 存贮工业的成效：
  - 嵌入式存贮
    - 更小、更便宜、更可靠、更低功耗
  - 数据使用
    - 高容量、层次式管理存储系统

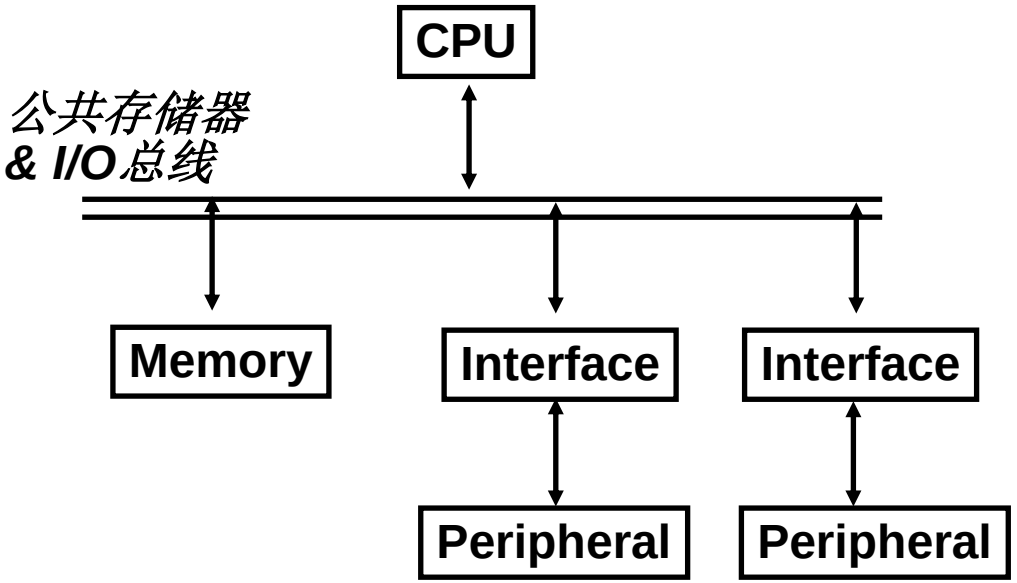
处理器接口

- 处理器接口
  - 中断
  - 存储器映射I/O
- I/O控制结构
  - 轮询（Polling）
  - 中断（Interrupts）
  - 直接存储器访问（DMA）
  - I/O控制器
  - I/O处理器
- 容量、访问时间、带宽
- 互联
  - 总线

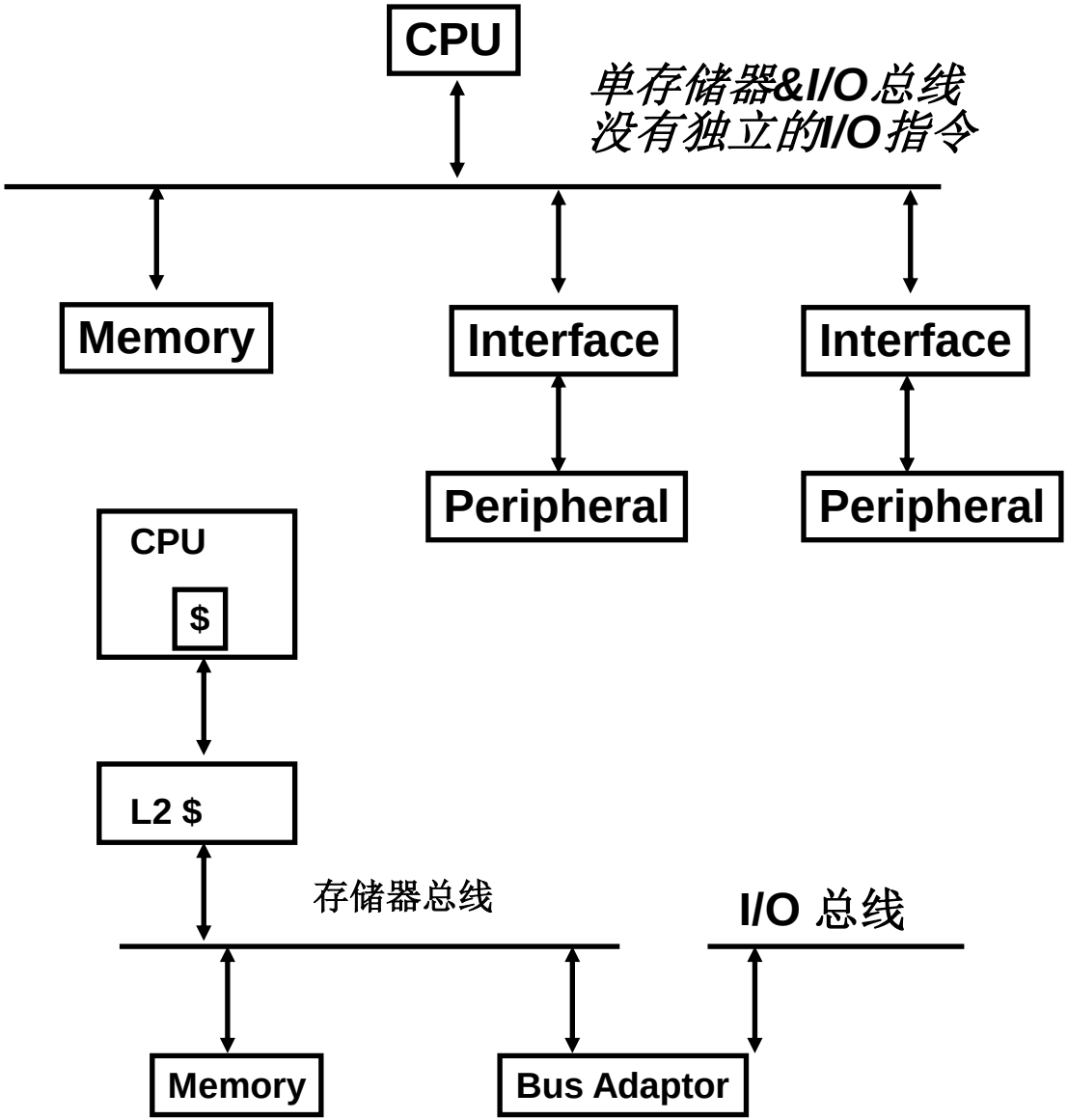
I/O 接口

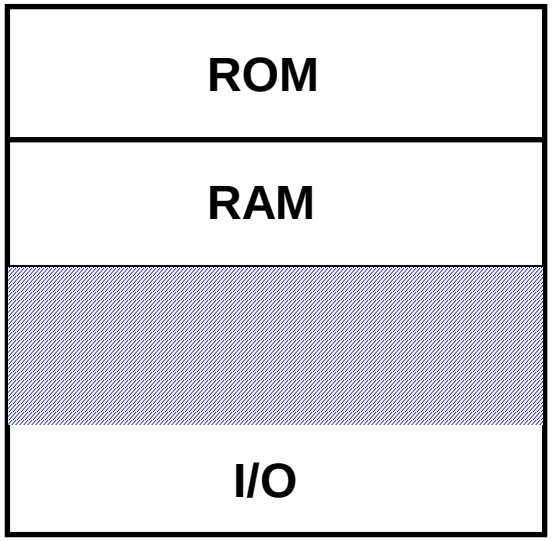


独立的I/O指令 (in,out)

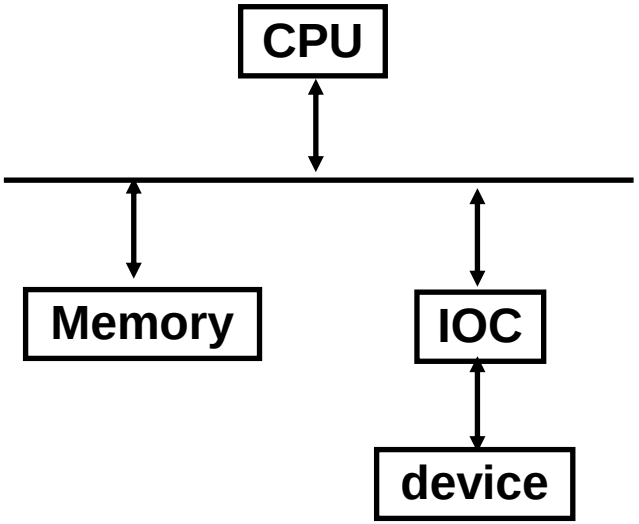


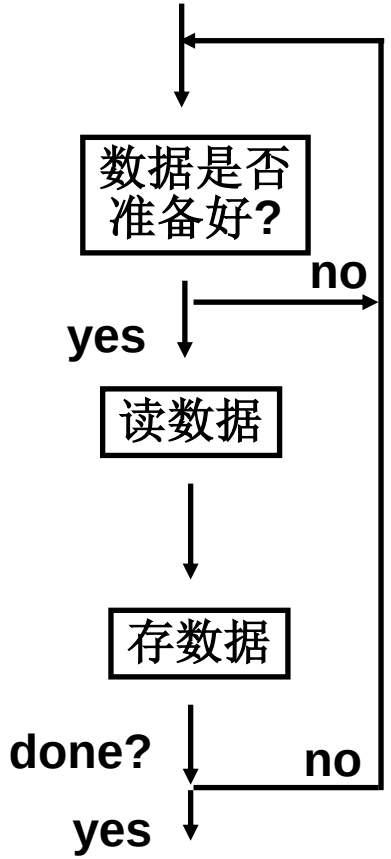
存储器映射 I/O


CPU
单存储器&I/O总线
没有独立的I/O指令
Memory
Interface
Interface
Peripheral
Peripheral
CPU
$
L2 $
存储器总线
I/O 总线
Memory
Bus Adaptor


ROM
RAM
I/O

可编程I/O (轮询)





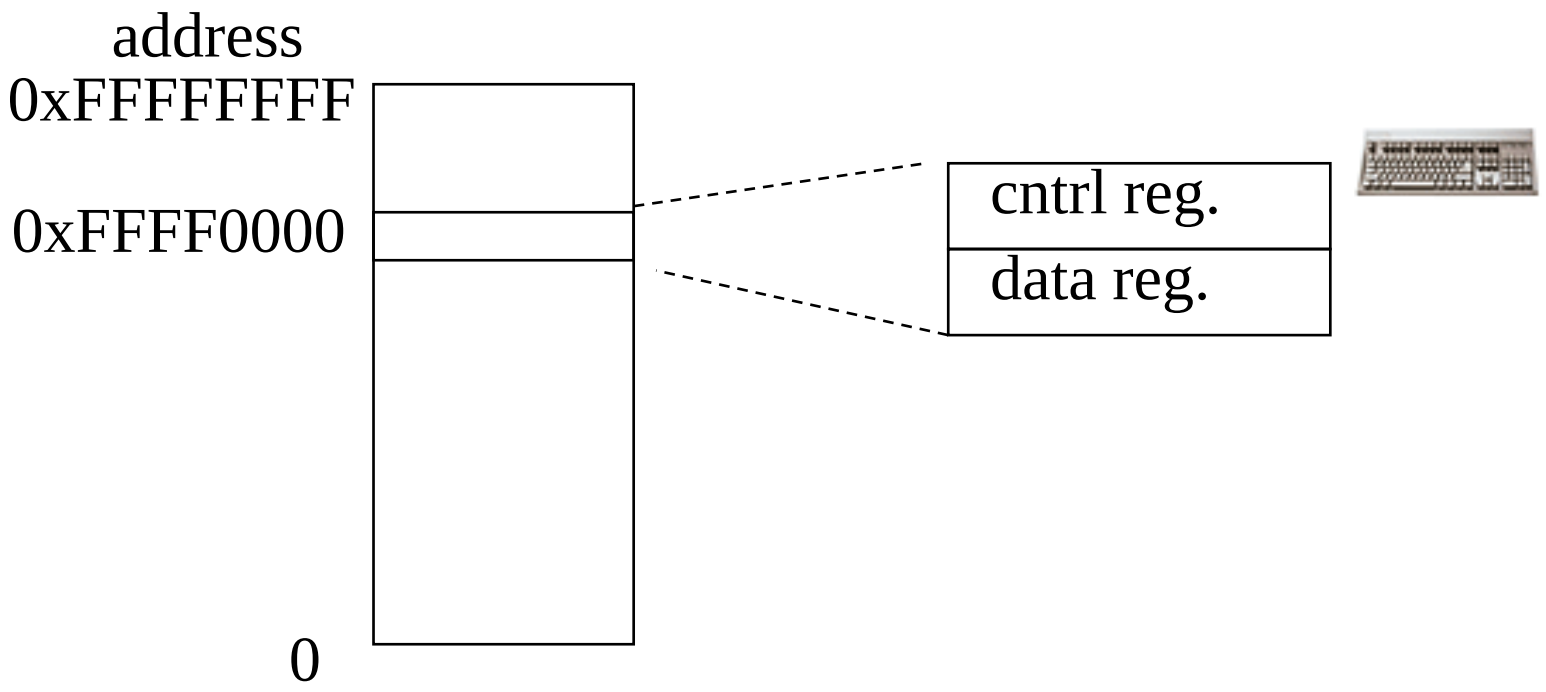
若忙就循环等待
不能很有效地利用
除非设备非常快!

但需要不断检测 I/O

I/O工作可以分散到
计算代码之中

Memory Mapped I/O

- Certain addresses are not regular memory
- Instead, they correspond to registers in I/O devices



Processor-I/O Speed Mismatch

- 1GHz microprocessor can execute 1 billion load or store instructions per second, or 4,000,000 KB/s data rate
  - I/O devices data rates range from 0.01 KB/s to 125,000 KB/s
- Input: device may not be ready to send data as fast as the processor loads it
  - Also, might be waiting for human to act
- Output: device not be ready to accept data as fast as processor stores it
- What to do?

Processor Checks Status before Acting

- Path to a device generally has 2 registers:
  - Control Register, says it's OK to read/write (I/O ready) [think of a flagman on a road]
  - Data Register, contains data
- Processor reads from Control Register in loop, waiting for device to set Ready bit in Control reg (0 1) to say its OK
- Processor then loads from (input) or writes to (output) data register
  - Load from or Store into Data Register resets Ready bit (1 0) of Control Register
- This is called "Polling"

I/O Example (polling)

- Input: Read from keyboard into `$v0`

```
                lui   $t0, 0xffff #ffff0000
Waitloop:       lw    $t1, 0($t0) #control
                andi  $t1,$t1,0x1
                beq   $t1,$zero, Waitloop
                lw    $v0, 4($t0) #data
```

- Output: Write to display from `$a0`

```
                lui   $t0, 0xffff #ffff0000
Waitloop:       lw    $t1, 8($t0) #control
                andi  $t1,$t1,0x1
                beq   $t1,$zero, Waitloop
                sw    $a0, 12($t0) #data
```

"Ready" bit is from processor's point of view!

Cost of Polling a Mouse?

- Assume for a processor with a 1GHz clock it takes 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning). Determine % of processor time for polling
- Mouse: polled 30 times/sec so as not to miss user movement
- Mouse Polling [clocks/sec]

  = 30 [polls/s] * 400 [clocks/poll] = 12K [clocks/s]
- % Processor for polling:

  $12*10^3$ [clocks/s] / $1*10^9$ [clocks/s] = 0.0012%

  Polling mouse little impact on processor

% Processor time to poll hard disk

- **Hard disk: transfers data in 16-Byte chunks and can transfer at 16 MB/second. No transfer can be missed. (we'll come up with a better way to do this)**

❑ **Frequency of Polling Disk**

= 16 [MB/s] / 16 [B/poll] = 1M [polls/s]

❑ **Disk Polling, Clocks/sec**
= 1M [polls/s] * 400 [clocks/poll]
= 400M [clocks/s]

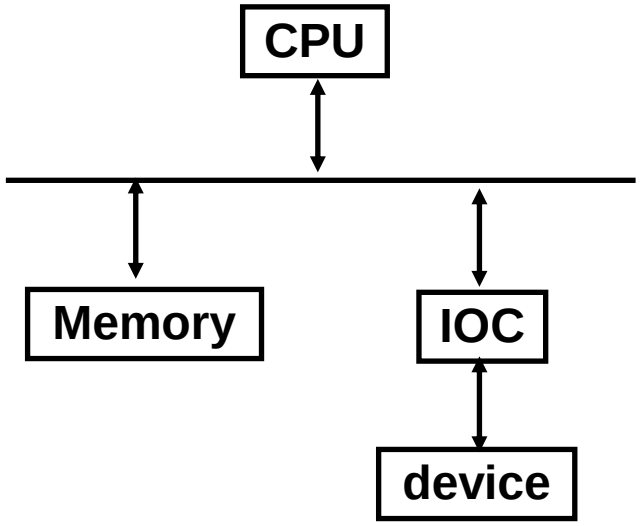
❑ **% Processor for polling:**

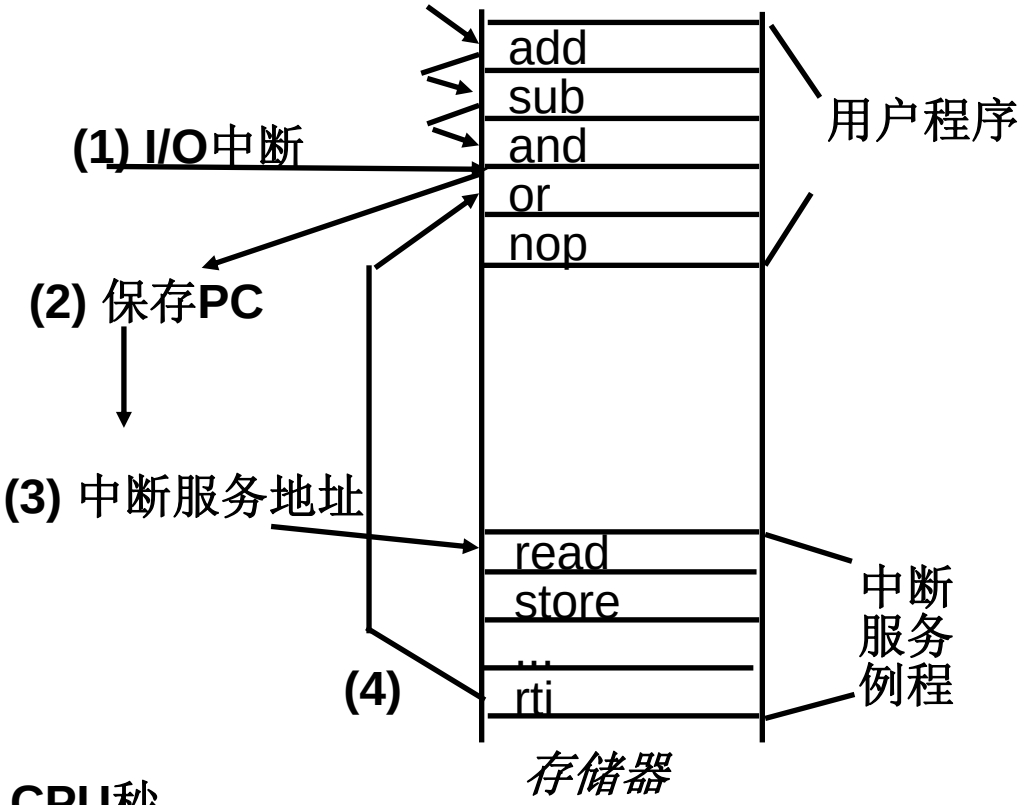
$400*10^6$ [clocks/s] / $1*10^9$ [clocks/s] = 40%

Unacceptable

(Polling is only part of the problem – main problem is that accessing in small chunks is inefficient)

中断驱动数据传输





用户程序仅在实际传输中才暂停

以每1ms一次的速率传输1000次：
1000次中断（每2微秒一次中断）
1000次中断服务（每次98微秒） = 0.1 CPU秒

设备传输率 = 10 MBytes/sec => 0 .1 x 10^{-6} sec/byte => 0.1 μsec/byte
=> 1000 bytes = 100 μsec
1000次传输 x 100 μsec = 100 ms = 0.1 CPU seconds

离设备传输率还有很大空间！中断开销的1/2

Benefit of Interrupt-Driven I/O

- Find the % of processor consumed if the hard disk is only active 5% of the time. Assuming 500 clock cycle overhead for each transfer, including interrupt:
  - Disk Interrupts/s = 5% * 16 [MB/s] / 16 [B/interrupt]
    = 50,000 [interrupts/s]
  - Disk Interrupts [clocks/s]
    = 50,000 [interrupts/s] * 500 [clocks/interrupt]
    = 25,000,000 [clocks/s]
  - % Processor for during transfer:
    $2.5*10^7$ [clocks/s] / $1*10^9$ [clocks/s] = 2.5% Busy
- DMA (Direct Memory Access) even better – only one interrupt for an entire page!

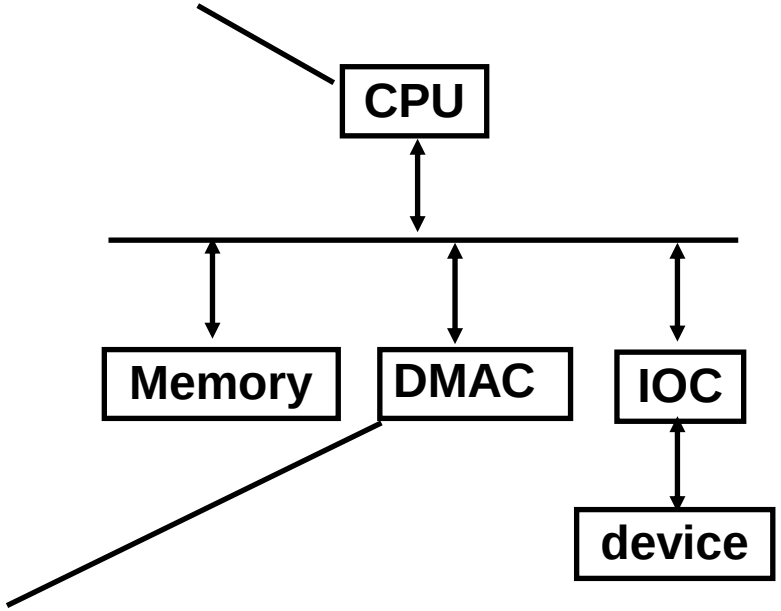
直接存储器访问

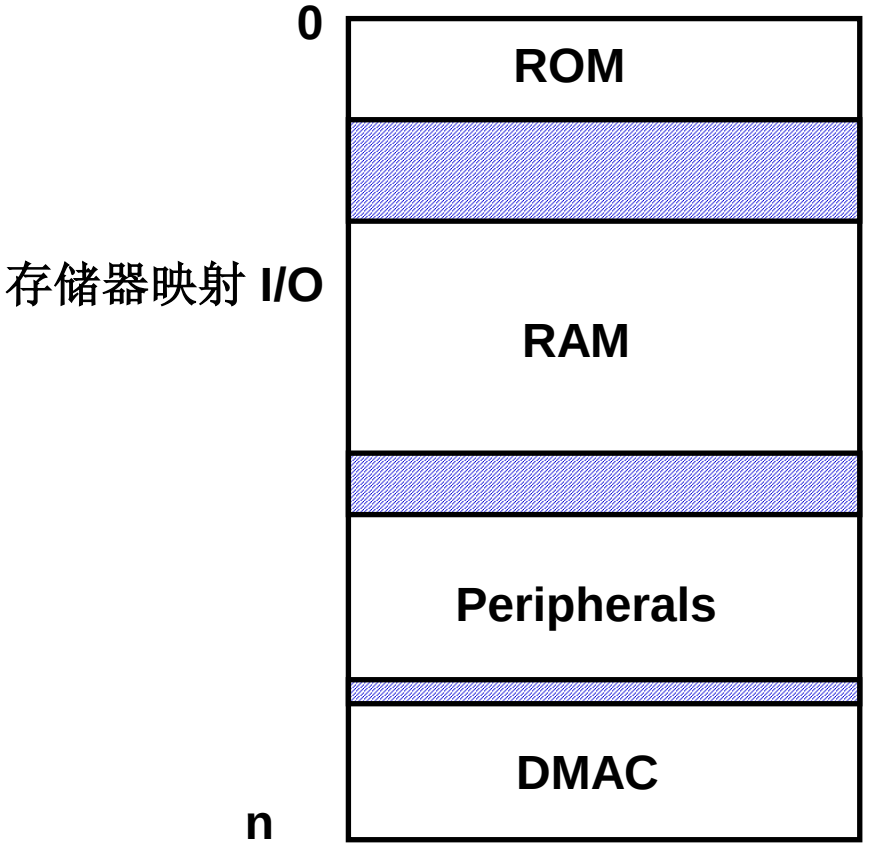
以每毫秒一次的速率完成1000次传输的时间：

1 DMA建立时间　50 μsec
1 次中断　2 μsec
1 次中断服务　48 μsec

0.0001 秒的CPU时间

CPU向DMAC发送开始地址、方向；
然后，发射“开始”命令。





DMAC　向外设控制器提供握手信号，
向存储器提供存储地址和握手信号

Operation of a DMA Transfer

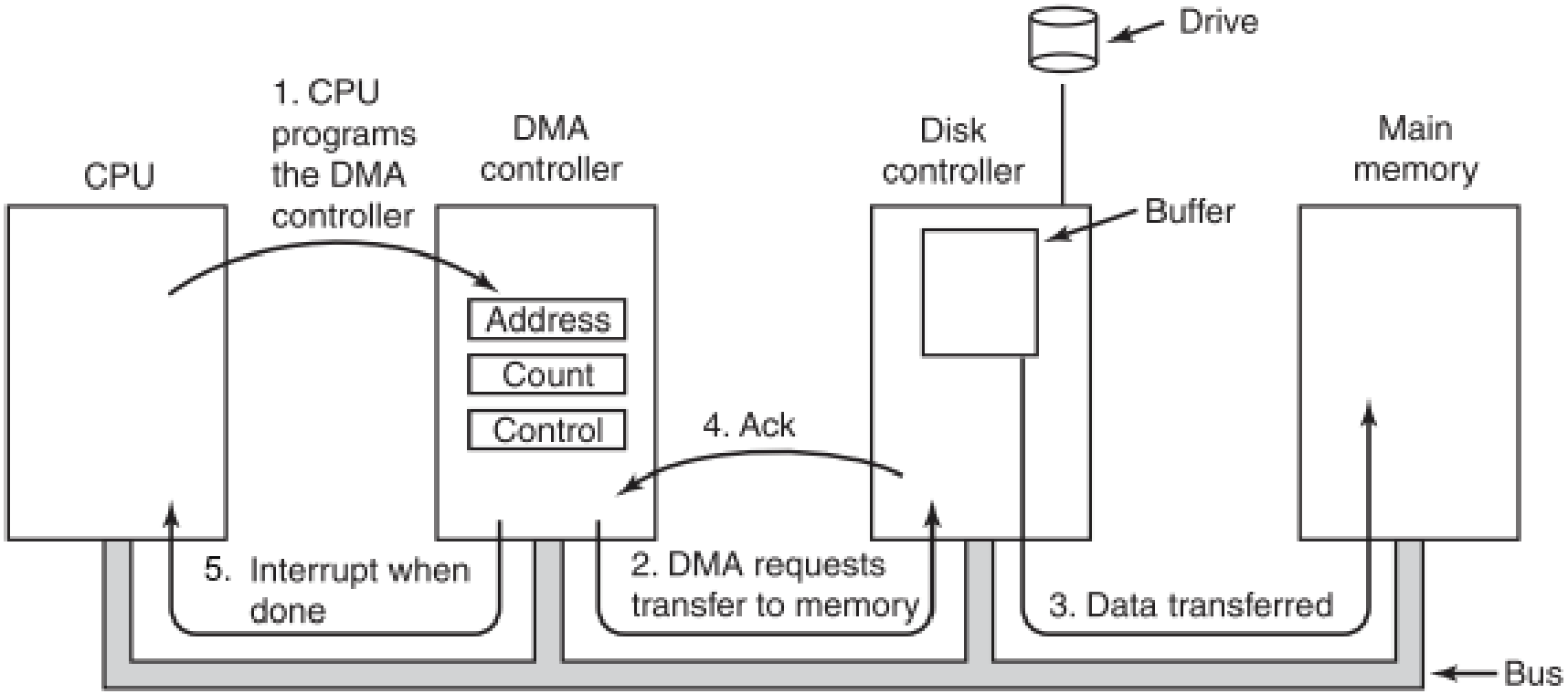
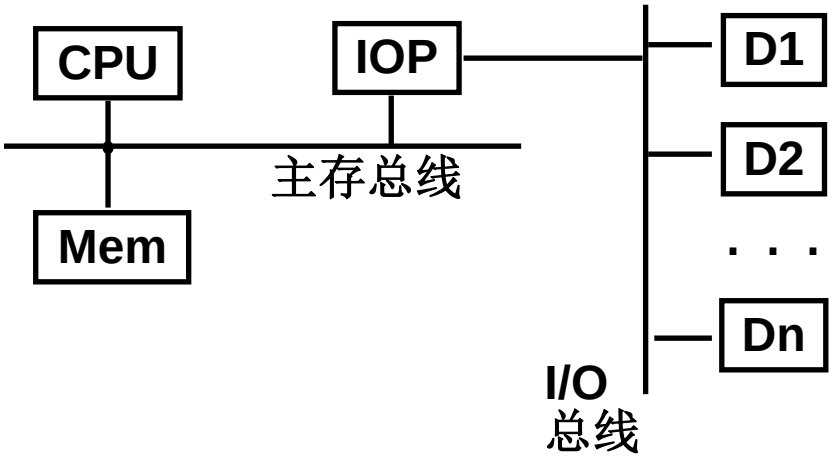
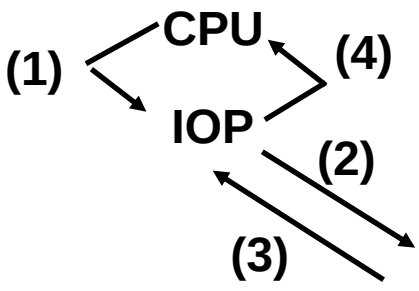


Figure 5-4. Operation of a DMA transfer.

[From Section 5.1.4 Direct Memory Access in *Modern Operating Systems* by Andrew S. Tanenbaum, Herbert Bos, 2014]

输入/输出处理器

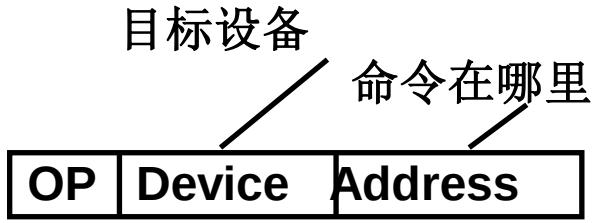




向 IOP发射指令

完成后中断

存储器



在存储器中查找命令

做什么

数据放在哪里

多少

特殊请求

设备 与 存储器之间的数据传送由IOP直接控制

IOP 偷取 存储器周期

与处理器体系结构的关系

- I/O 指令已经基本消失了
- 中断向量 被 跳转表（jump tables）替代
  PC <- M [中断向量地址 + 中断号]
  PC <- 中断向量地址 + 中断号
- 中断：
  - 中断处理程序保存寄存器，并且重新使能更高优先级的中断的请求
  - 中断类型的数量不断减少；中断处理程序必须询问中断控制器

与处理器体系结构的关系（续）

- 提高处理器性能增设的cache对I/O提出新的问题
  - 冲洗CACHE非常费时，而I/O可能污染cache
  - 可以从共享存储多处理器的“监听(snooping)”策略借鉴解决方案
- 虚拟存储器 对 DMA 提出新问题
- 一些Load/store结构可能要求原子性操作
  - 装入锁定（load locked）、条件存储（store conditional）
- 处理器难以进行 上下文切换（context switch）

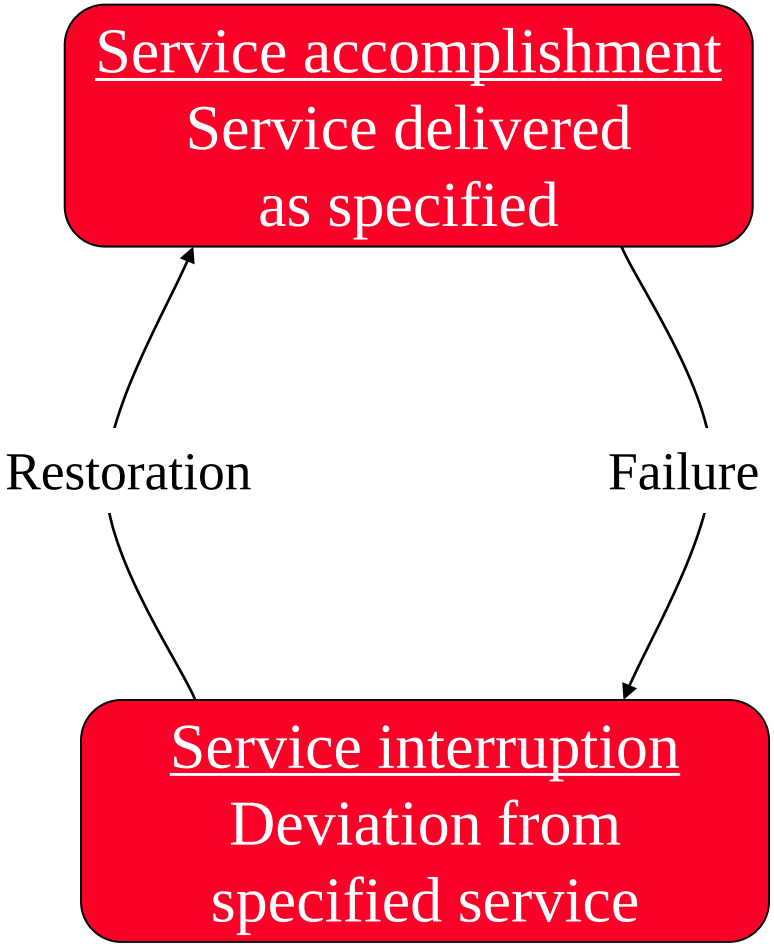
I/O设备的类型和特征

- 行为：一个 I/O设备如何工作?
  - 输入设备：只读
  - 输出设备：只写，不能读
  - 存贮设备：可以重读，通常也可重写

- 合作对象：
  - 在I/O设备的另一方要么是人，要么是机器
  - 要么是输入设备传入数据，要么是输出设备读取数据

- 数据传输率
  - 数据可以传送的最大速率
    - 在I/O设备和主存之间
    - 或者 在I/O设备与CPU之间

Dependability



- **Fault: failure of a component**
  - **May or may not lead to system failure**

Dependability Measures

- Reliability: Mean Time To Failure (MTTF)
- Service interruption: Mean Time To Repair (MTTR)
- Mean time between failures (MTBF)
  - MTBF = MTTF + MTTR
- Availability = MTTF / (MTTF + MTTR)
- Improving Availability
  - Increase MTTF: More reliable hardware/software + Fault Tolerance
  - Reduce MTTR: improved tools and processes for diagnosis and repair

Availability Measures

- Availability = MTTF / (MTTF + MTTR) as %
  - MTTF, MTBF usually measured in hours

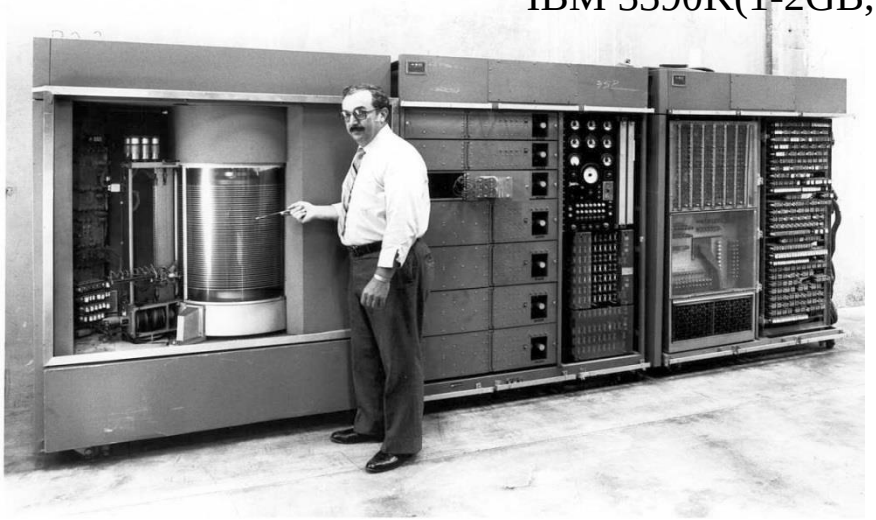
- Since hope rarely down, shorthand is "number of 9s of availability per year"
  - 1 nine: 90% => 36 days of repair/year
  - 2 nines: 99% => 3.6 days of repair/year
  - 3 nines: 99.9% => 526 minutes of repair/year
  - 4 nines: 99.99% => 53 minutes of repair/year
  - 5 nines: 99.999% => 5 minutes of repair/year

可靠性 与 可用性
(Reliability and Availability)

- 常被混淆的两个概念
  - 可靠性：是否有部件失效？
  - 可用性：是否用户还可以正确使用系统？
- 可用性可以通过增加硬件来改进：
  - 例如： 存储器中增加ECC
- 可靠性只能通过下列方法改进：
  - 改善使用环境状态
  - 建造更加可靠的元器件和部件
  - 减少系统使用的元器件和部件数
    - 可以通过使用低成本、低可靠性的部件来改进可用性

Evolution of the Disk Drive

IBM 3390K(1-2GB, $50,000/GB), 1986

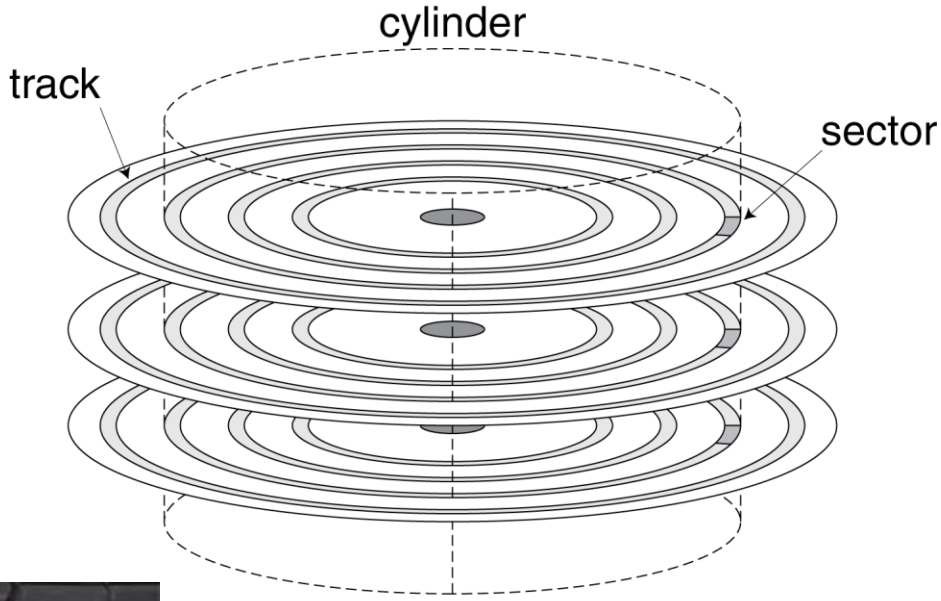


IBM RAMAC 305(5Mb,$3200/Month), 1956

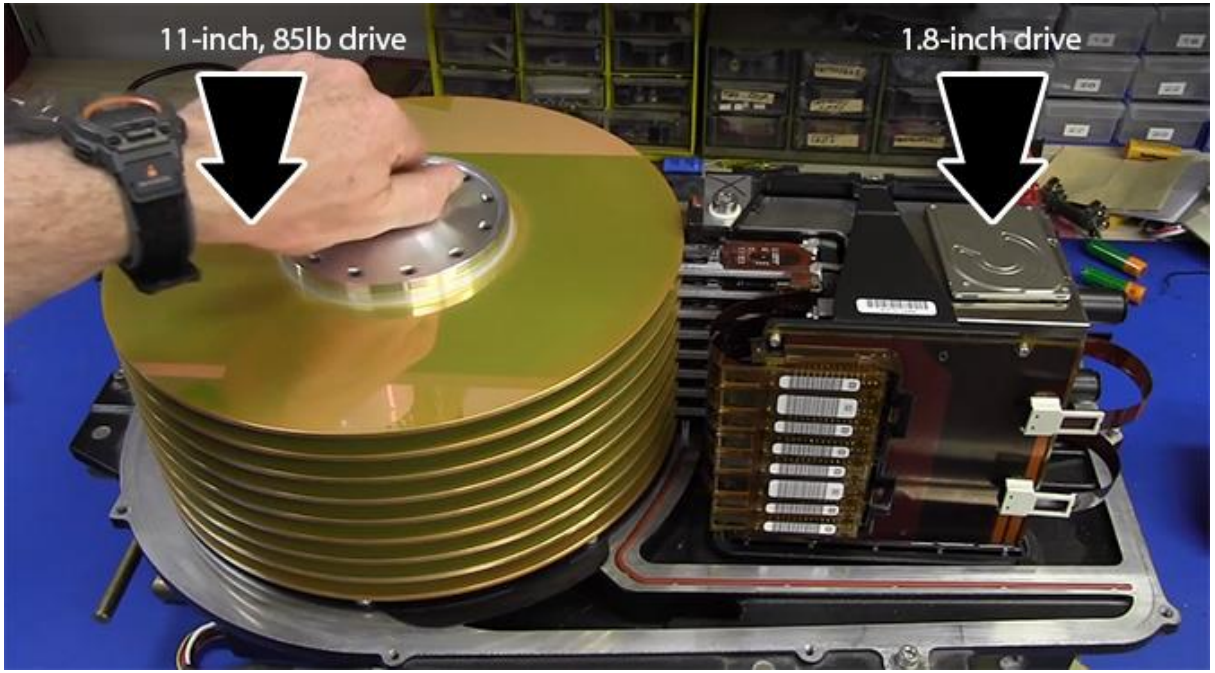
Apple SCSI, 1986

Disk Storage

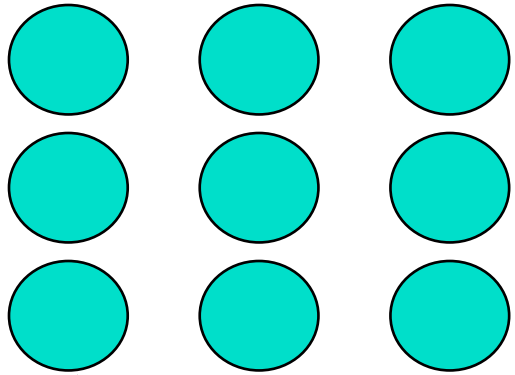
- Nonvolatile, rotating magnetic storage



IBM 3390K



磁盘阵列



- 一种磁盘存储的新组织
  - 大量容量小、价廉的磁盘构成的阵列
  - 通过使用很多磁盘驱动器来提高潜在吞吐率:
    - 数据分布在多个磁盘上
    - 对不同磁盘进行多次访问
- 可靠性比单个磁盘更低
  - 但是可以通过增加冗余磁盘改进可用性，可以利用信息重建丢失信息
  - **MTTR:** 平均修复时间，小时级别
  - **MTTF:** 平均无故障时间，三年至五年

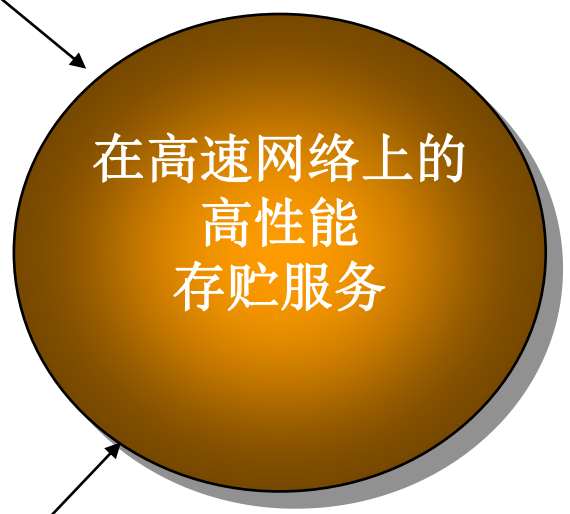
网络附属存贮

磁盘大小逐步缩小

14" ⇒10" ⇒ 8" ⇒ 5.25" ⇒ 3.5" ⇒ 2.5" ⇒ 1.8" ⇒ 1.3" ⇒. . .

基于磁盘阵列的高带宽磁盘系统

网络提供了更好的物理和逻辑接口：独立的CPU 和存贮系统！



网络文件服务

支持远程文件访问的操作系统 结构

3 Mb/s ⇒ 10Mb/s ⇒ 50 Mb/s ⇒ 100 Mb/s ⇒ 1 Gb/s ⇒ *10 Gb/s*

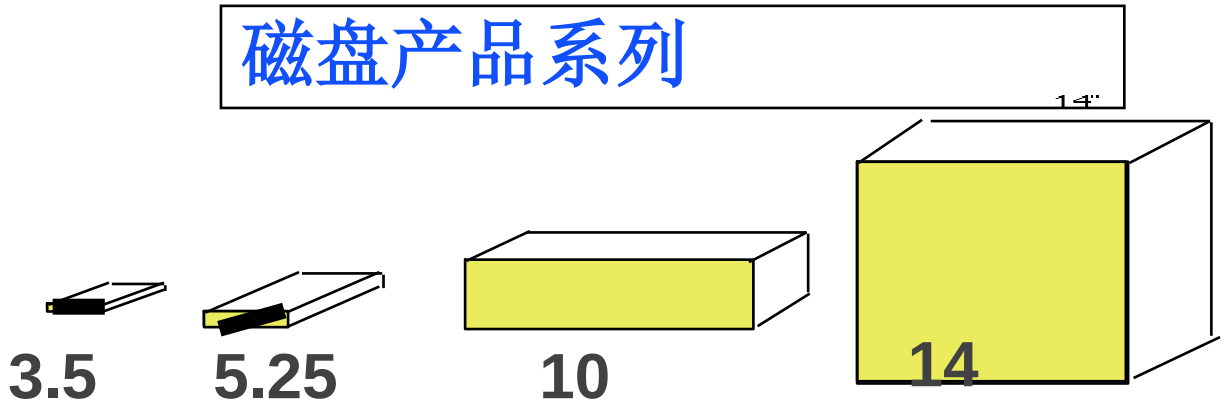
网络的持续高带宽传输能力

网络带宽逐步增加

磁盘阵列的制造上优势

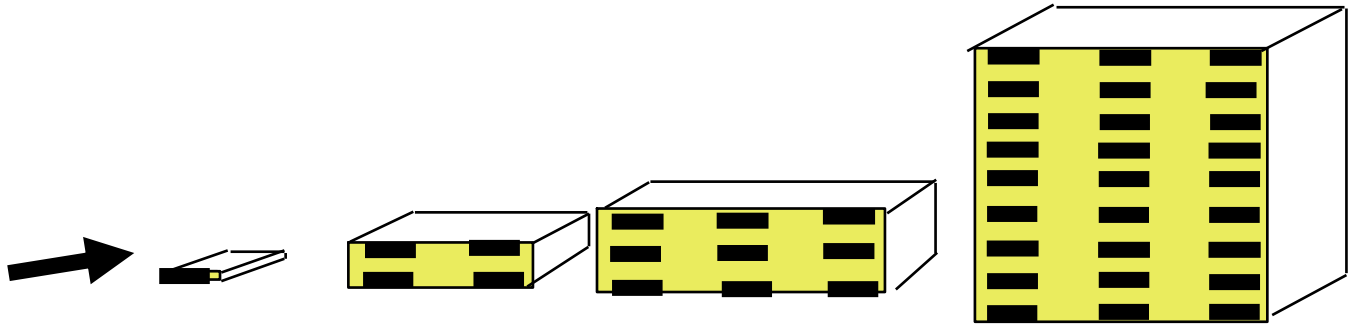
常规：

4种磁盘设计



磁盘阵列：

1种磁盘设计



阵列的可靠性

- N个磁盘的可靠性 = 1个磁盘的可靠性 ÷ N

  50,000小时 ÷ 70 磁盘 = 700 小时

  磁盘系统的平均无故障时间：从 6 年 跌至 1个月！

- 没有冗余的阵列 在使用中 太不可靠！

可与访问过程 并行 进行重构的热备份：可以达到很高的媒体可用性

Array Reliability

- **Reliability - whether or not a component has failed**
  - **measured as Mean Time To Failure (MTTF)**
- **Reliability of N disks**
  = Reliability of 1 Disk ÷ N
  (assuming failures independent)
  - **50,000 Hours ÷ 70 disks = 700 hour**
- **Disk system MTTF:**
  Drops from 6 years to 1 month!
- **Disk arrays too unreliable to be useful!**

Error Detection/Correction Codes

- Memory systems generate errors (accidentally flipped-bits)
  - DRAMs store very little charge per bit
  - "Soft" errors occur occasionally when cells are struck by alpha particles or other environmental upsets
  - "Hard" errors can occur when chips permanently fail
  - Problem gets worse as memories get denser and larger
- Memories protected against failures with EDC/ECC
- Extra bits are added to each data-word
  - Used to detect and/or correct faults in the memory system
  - Each data word value mapped to unique *code word*
  - A fault changes valid code word to invalid one, which can be detected

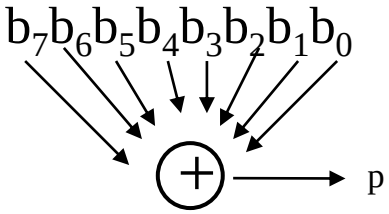
Block Code Principles

- Hamming distance = difference in # of bits
- p = 011011, q = 001111, Ham. distance (p,q) = 2
- p = 011011,
  q = 110001,
  distance (p,q) = ?
- Can think of extra bits as creating a code with the data
- What if minimum distance between members of code is 2 and get a 1-bit error?

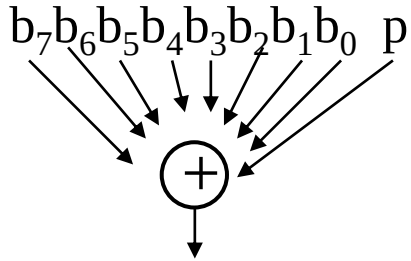
Richard Hamming, 1915-98
Turing Award Winner

Parity: Simple Error-Detection Coding

- **Each data value, before it is written to memory is "tagged" with an extra bit to force the stored word to have *even parity*:**



- **Each word, as it is read from memory is "checked" by finding its parity (including the parity bit).**



- Minimum Hamming distance of parity code is 2
- A non-zero parity check indicates an error occurred:
  - 2 errors (on different bits) are not detected
  - Nor any even number of errors, just odd numbers of errors are detected

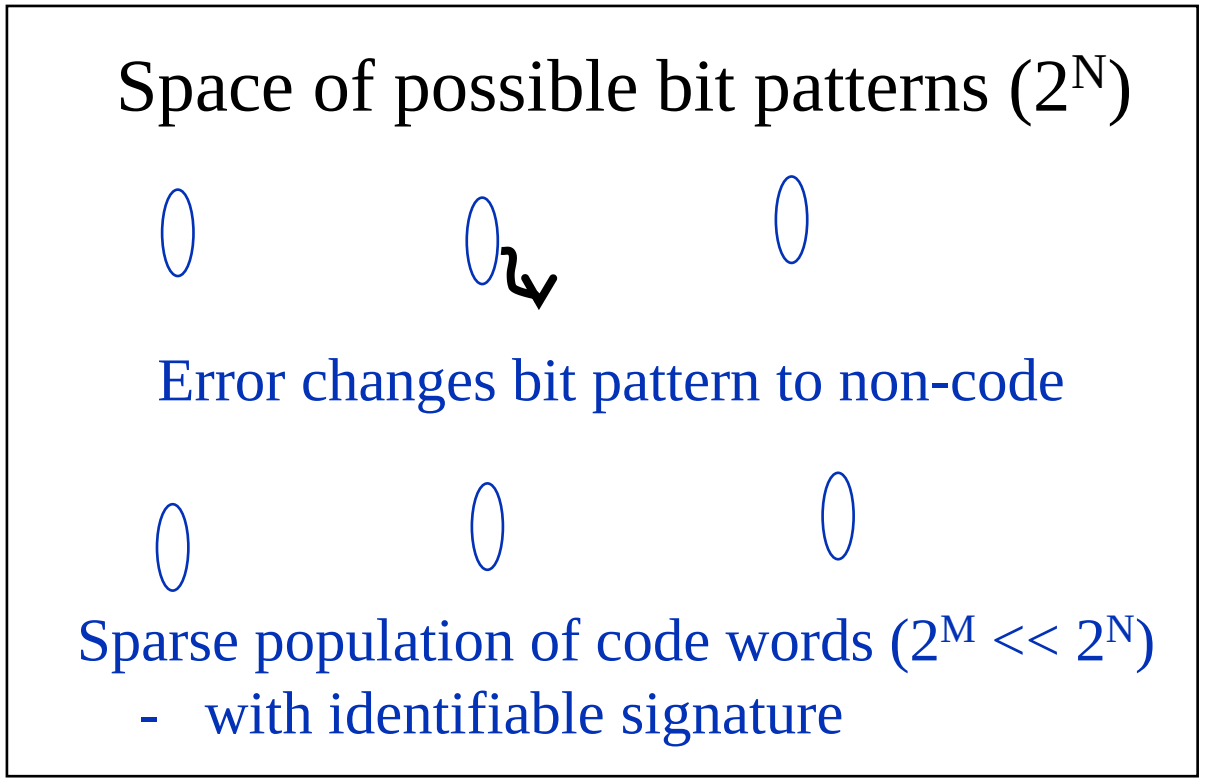
Parity Example

- Data 0101 0101
- 4 ones, even parity now
- Write to memory:
  0101 0101 0
  to keep parity even
- Data 0101 0111
- 5 ones, odd parity now
- Write to memory:
  0101 0111 1
  to make parity even
- Read from memory
  0101 0101 0
- 4 ones => even parity, so no error
- Read from memory
  1101 0101 0
- 5 ones => odd parity, so error
- What if error in parity bit?

Suppose Want to Correct One Error?

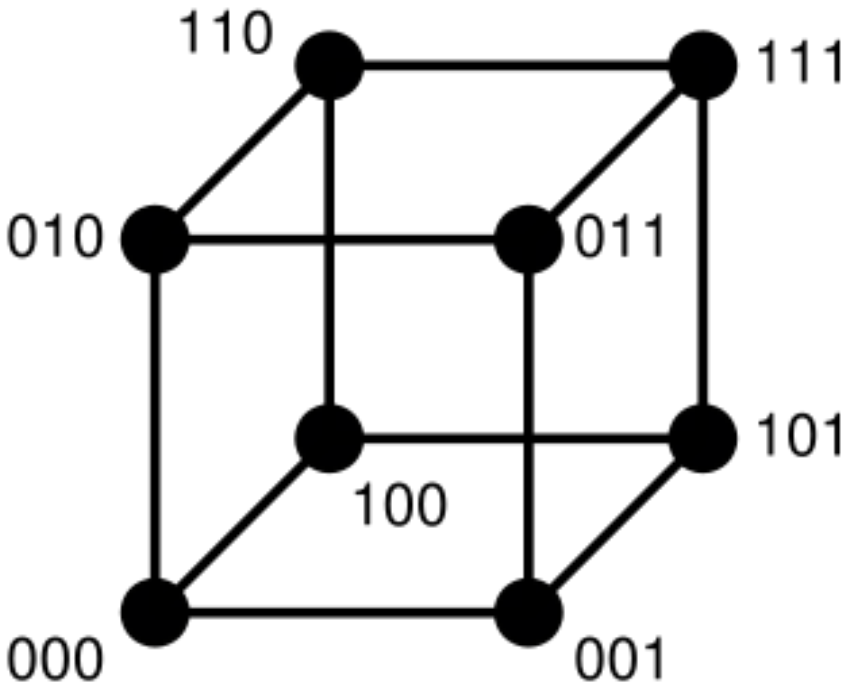
- Hamming came up with simple to understand mapping to allow Error Correction at minimum distance of three
  - Single error correction, double error detection
- Called “Hamming ECC”
  - Worked weekends on relay computer with unreliable card reader, frustrated with manual restarting
  - Got interested in error correction; published 1950
  - R. W. Hamming, “Error Detecting and Correcting Codes,” *The Bell System Technical Journal*, Vol. XXVI, No 2 (April 1950) pp 147-160.

Detecting/Correcting Code Conce

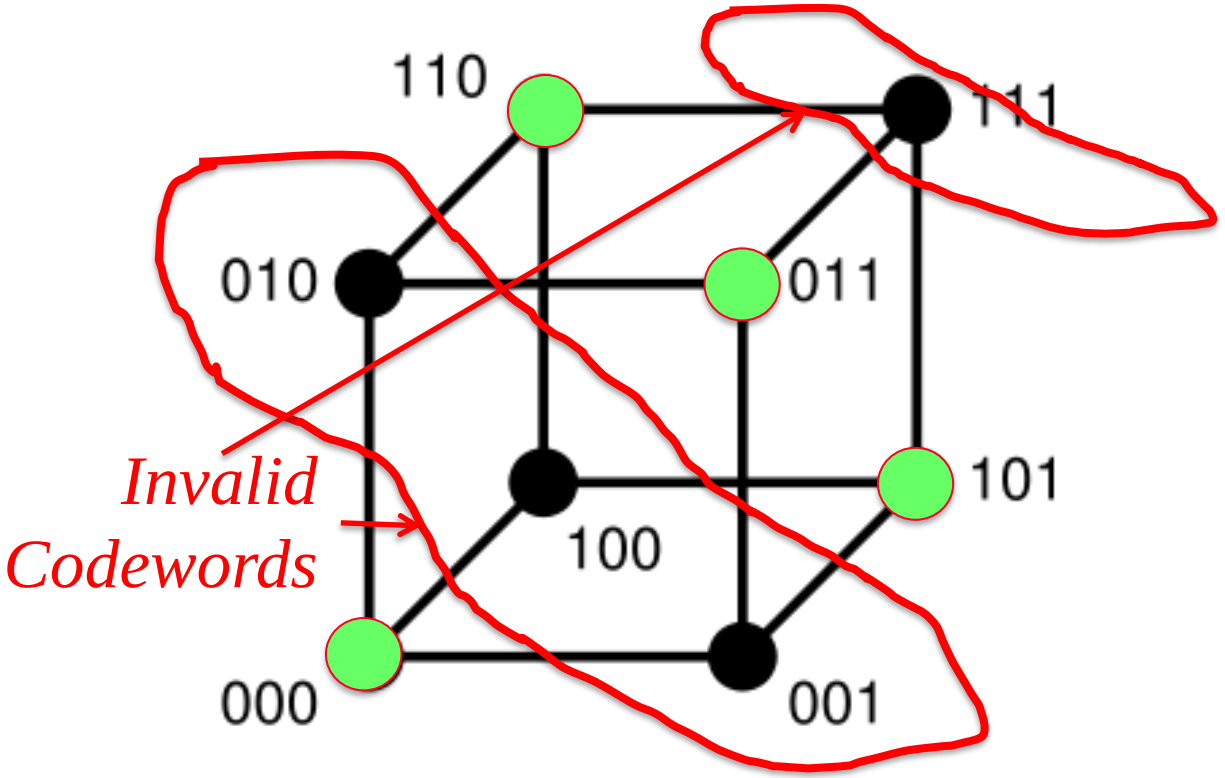


- **Detection**: bit pattern fails codeword check
- **Correction**: map to nearest valid code word

Hamming Distance: Eight Code Wo



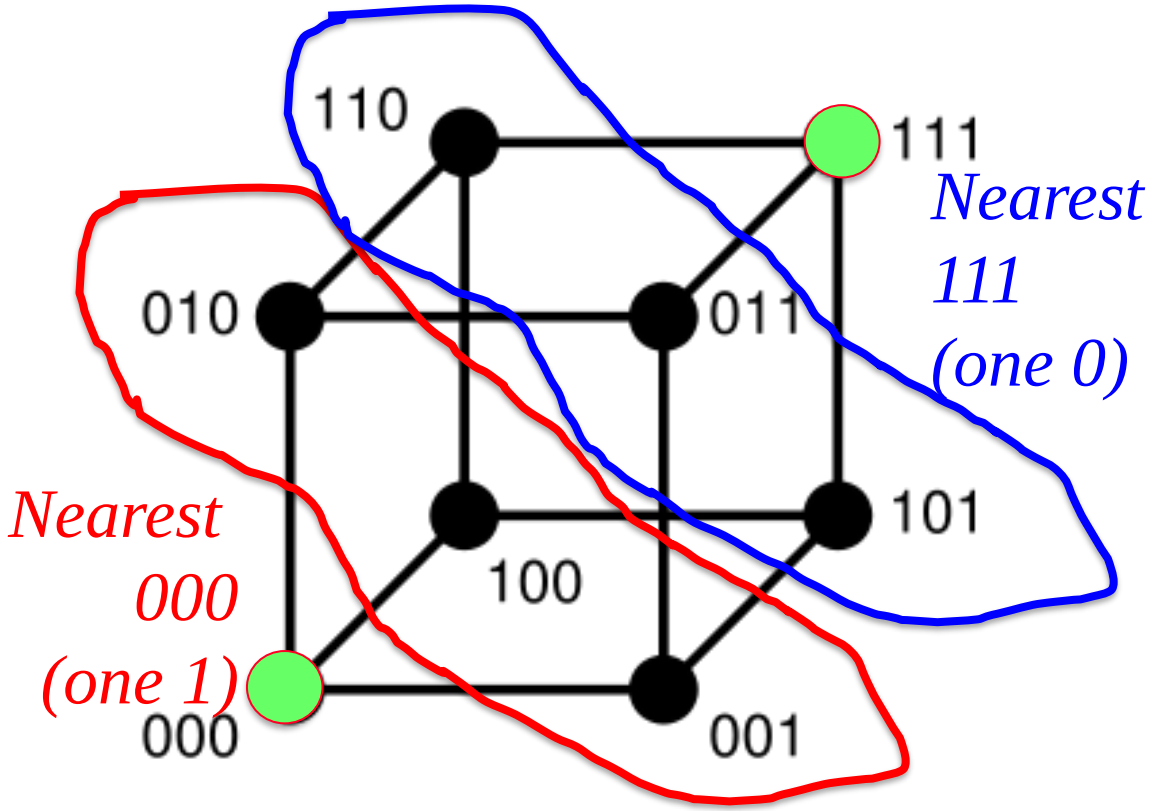
Hamming Distance 2: Detection
Detect Single Bit Errors



- No 1 bit error goes to another valid codeword
- ½ codewords are valid

Hamming Distance 3: Correction

Correct Single Bit Errors, Detect Double Bit Errors



- No 2 bit error goes to another valid codeword; 1 bit error near
- 1/4 codewords are valid

Graphic of Hamming Code

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11
Parity bit coverage	p1	X		X		X		X		X		X		X		X
	p2		X	X			X	X			X	X			X	X
	p4				X	X	X	X					X	X	X	X
	p8								X	X	X	X	X	X	X	X

- http://en.wikipedia.org/wiki/Hamming_code

Hamming ECC

Set parity bits to create even parity for each group

- **A byte of data: 10011010**
- **Create the coded word, leaving spaces for the parity bits:**
- **_ _ 1 _ 0 0 1 _ 1 0 1 0**

  1 2 3 4 5 6 7 8 9 a b c – bit position

- **Calculate the parity bits**

Hamming ECC

- Position 1 checks bits 1,3,5,7,9,11:
  ? _ 1 _ 0 0 1 _ 1 0 1 0. set position 1 to a _:

- Position 2 checks bits 2,3,6,7,10,11:
  0 ? 1 _ 0 0 1 _ 1 0 1 0. set position 2 to a _:

- Position 4 checks bits 4,5,6,7,12:
  0 1 1 ? 0 0 1 _ 1 0 1 0. set position 4 to a _:

- Position 8 checks bits 8,9,10,11,12:
  0 1 1 1 0 0 1 ? 1 0 1 0. set position 8 to a _:

Hamming ECC

- Final code word: 011100101010
- Data word: 1 001 1010

Hamming ECC Error Check

- Suppose receive 011100101110

0 1 1 1 0 0 1 0 1 1 1 0

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11
Parity bit coverage	p1	X		X		X		X		X		X		X		X
	p2		X	X			X	X			X	X			X	X
	p4				X	X	X	X					X	X	X	X
	p8								X	X	X	X	X	X	X	X

Hamming ECC Error Check

- Suppose receive

  01110010 1 1 1 0

Hamming ECC Error Check

- Suppose receive

```
01110010 1110
0 1 0 1 1 1   √
 11   01   11   x-Parity 2 in error
   1001     0  √
        01110   x-Parity 8 in error
```

- *Implies position 8+2=10 is in error*

```
01110010 1110
```

Hamming ECC Error Correct

- Flip the incorrect bit …

  011100101010

Hamming ECC Error Correct

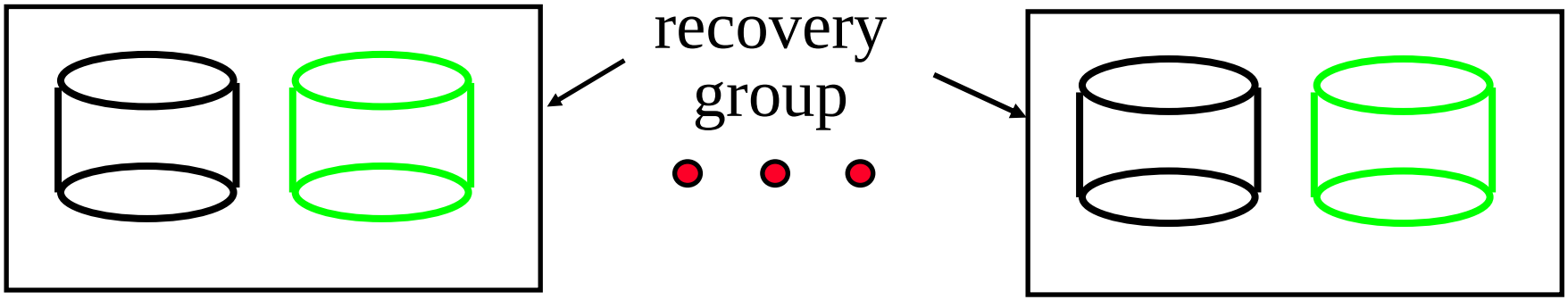
- Suppose receive

```
01110010 10
0 1 0 1 1 1   √
 11   01   01  √
  1001      0  √
       01010   √
```

RAID: Redundant Arrays of (Inexpensive) Disks

- Files are "striped" across multiple disks
- Redundancy yields high data availability
  - Availability: service still provided to user, even if some components failed
- Disks will still fail
- Contents reconstructed from data redundantly stored in the array
  - Capacity penalty to store redundant info
  - Bandwidth penalty to update redundant info

Redundant Arrays of Inexpensive Disks
RAID 1: Disk Mirroring/Shadowing



- Each disk is fully duplicated onto its "mirror"
  Very high availability can be achieved
- Bandwidth sacrifice on write:
  Logical write = two physical writes
  Reads may be optimized
- Most expensive solution: 100% capacity overhead

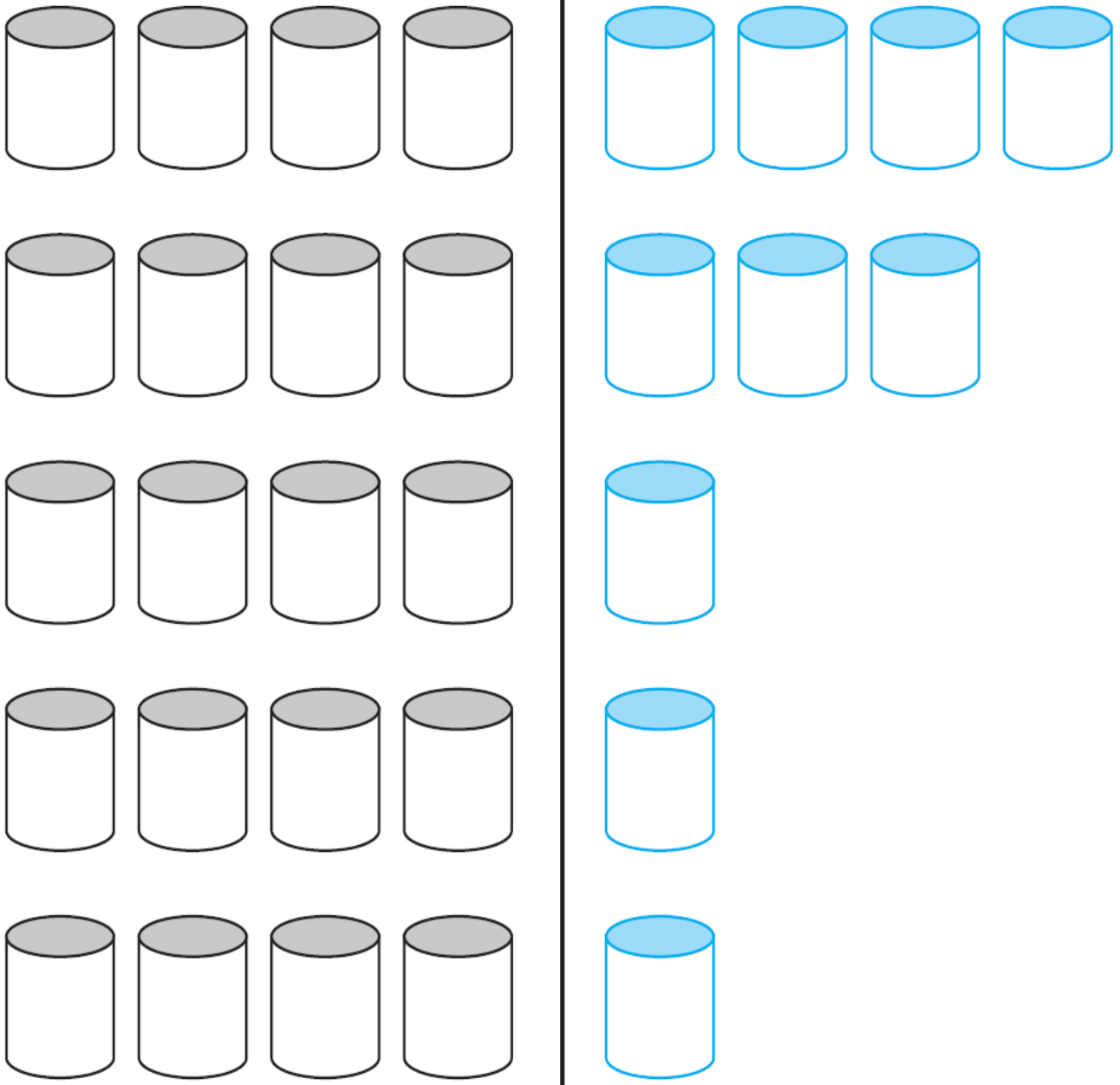
RAID 1
(Mirroring)
EMC, HP(Tandem), IBM

RAID 2
(Error detection and
correction code) Unused

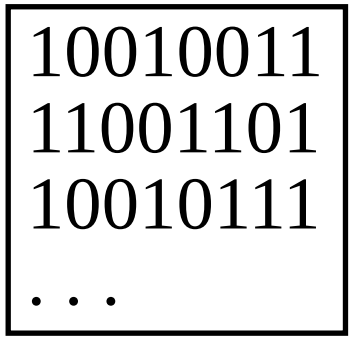
RAID 3
(Bit-interleaved parity)
Storage concepts

RAID 4
(Block-interleaving parity)
Network appliance

RAID 5
(Distributed block-
interleaved parity)
Widely used



Redundant Array of Inexpensive Disks RAID 3: Parity Disk

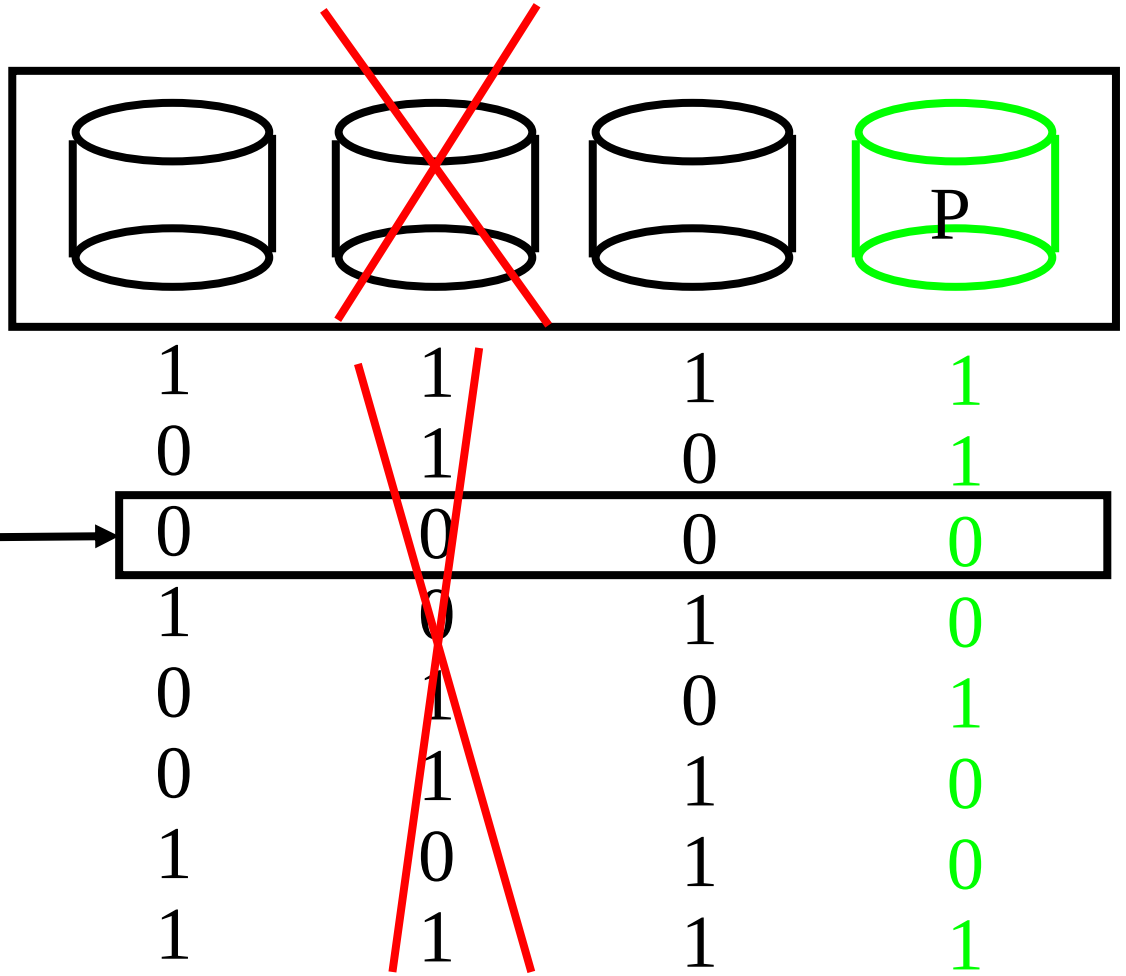


logical record

Striped physical records

P contains sum of other disks per stripe mod 2 ("parity")

If disk fails, subtract P from sum of other disks to find missing information



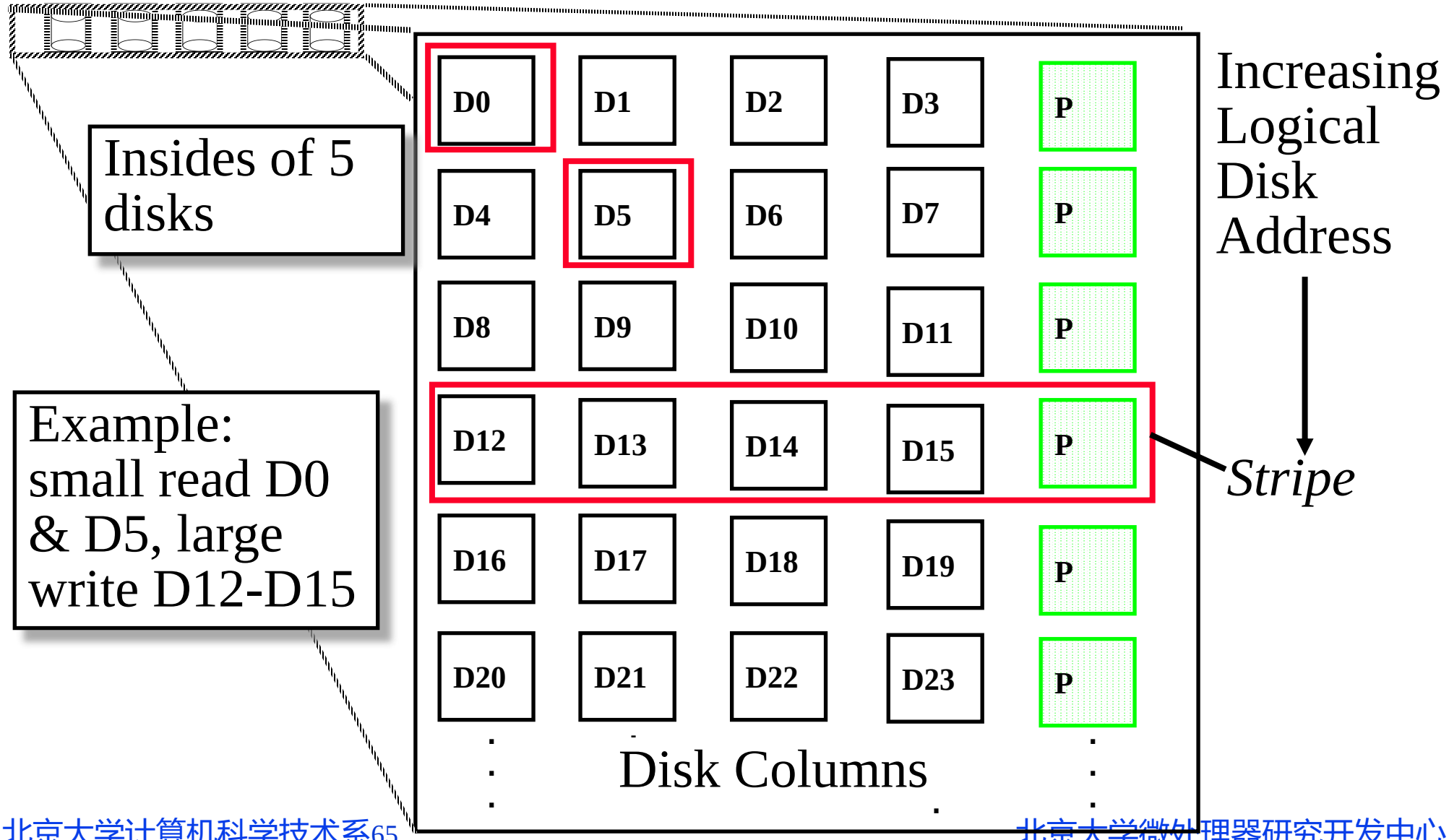
RAID 3

- Sum computed across recovery group to protect against hard disk failures, stored in P disk
- Logically, a single high capacity, high transfer rate disk: good for large transfers
- Wider arrays reduce capacity costs, but decreases availability
- 33% capacity cost for parity if 3 data disks and 1 parity disk

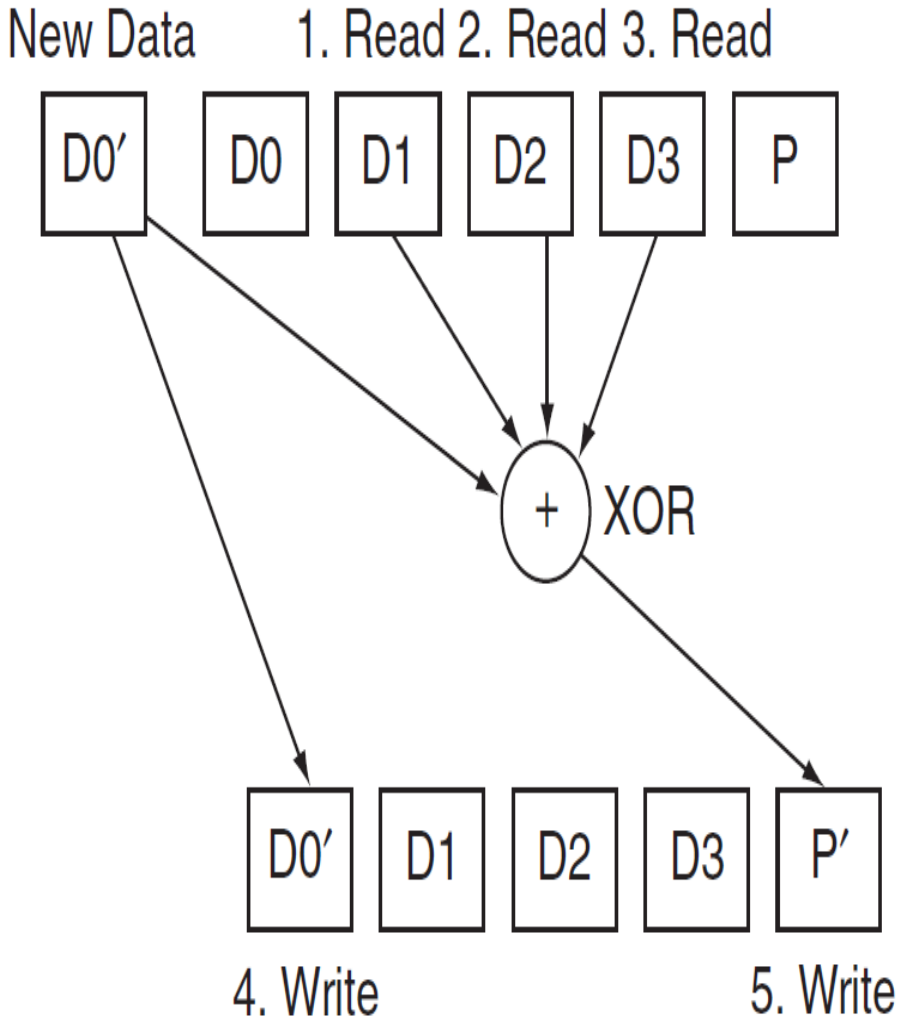
Inspiration for RAID 4

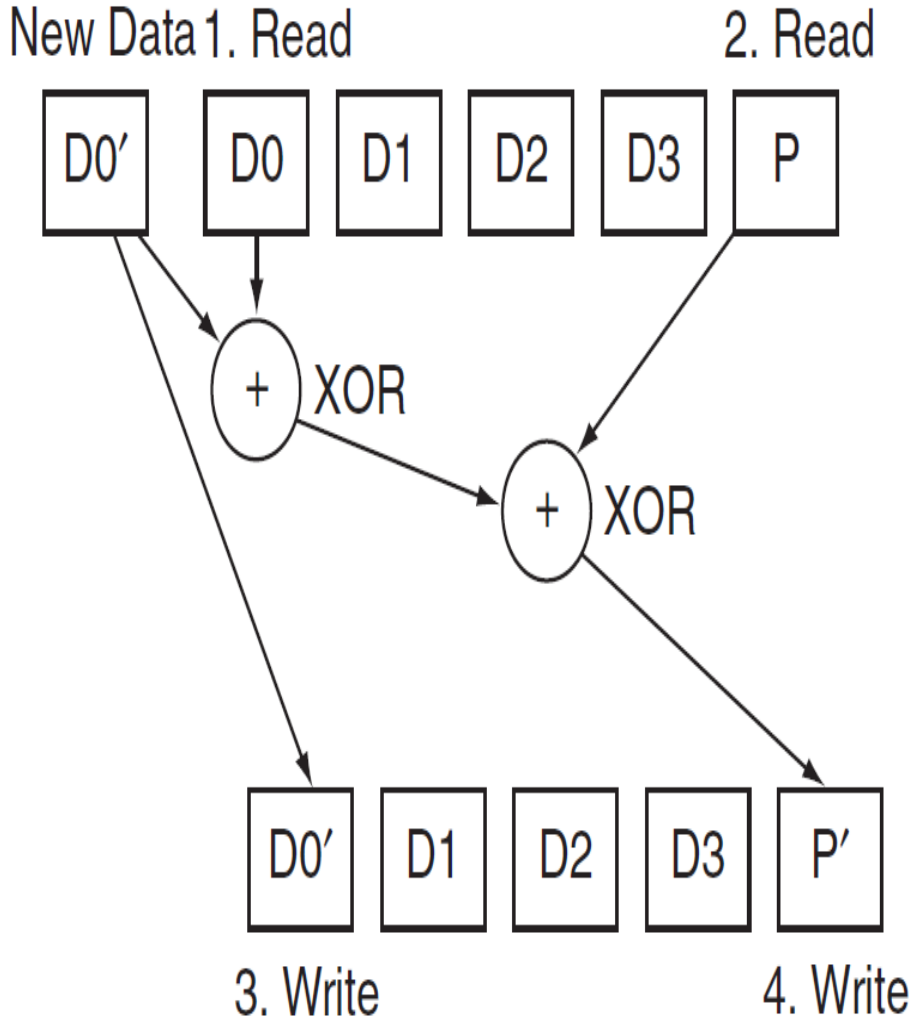
- RAID 3 relies on parity disk to discover errors on Read
- But every sector has an error detection field (Reed Solomon Code)
- To catch errors on read, rely on error detection field vs. the parity disk
- Allows independent reads to different disks simultaneously

Redundant Arrays of Inexpensive Disks RAID High I/O Rate Parity



Small write update on RAID 4

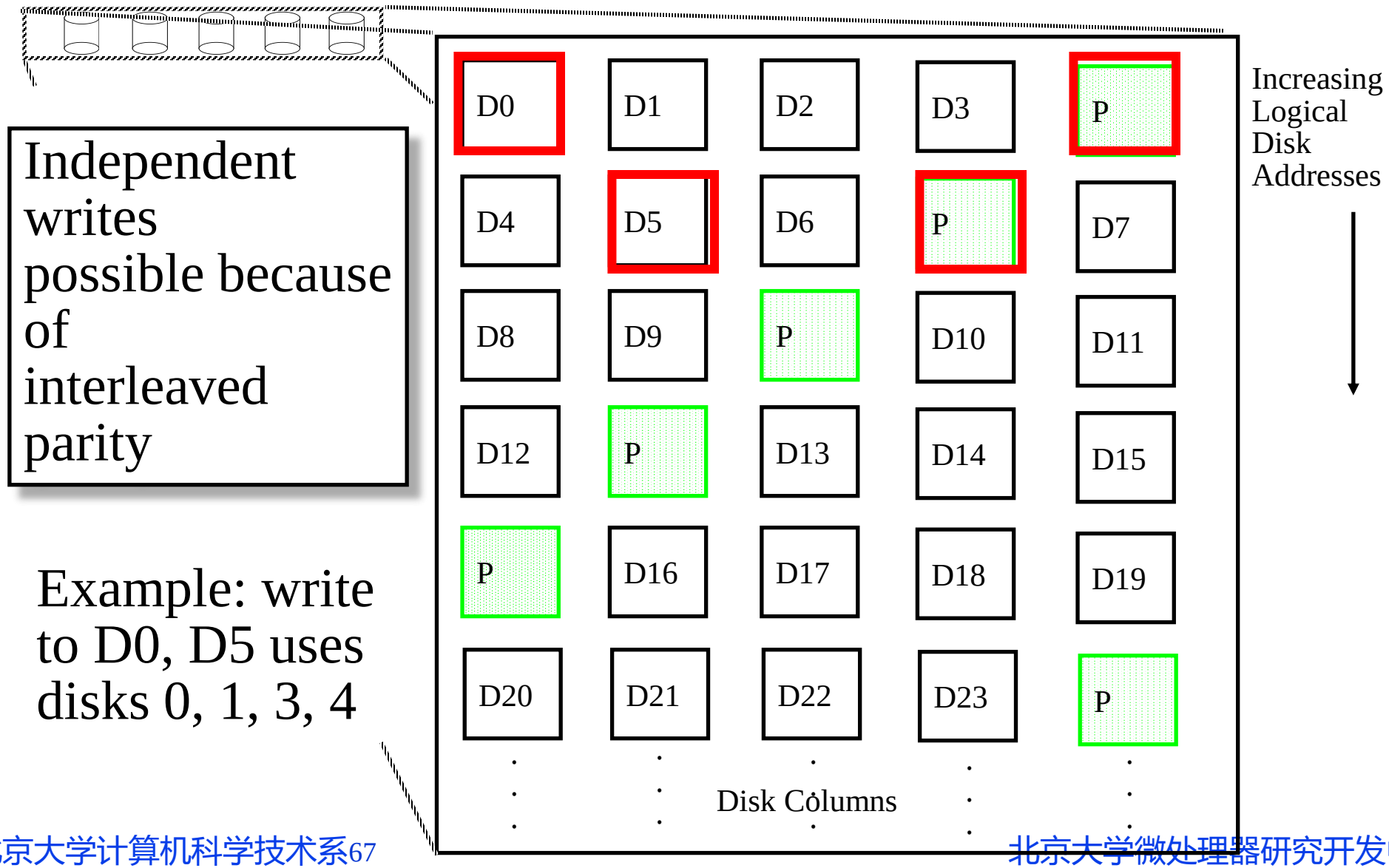




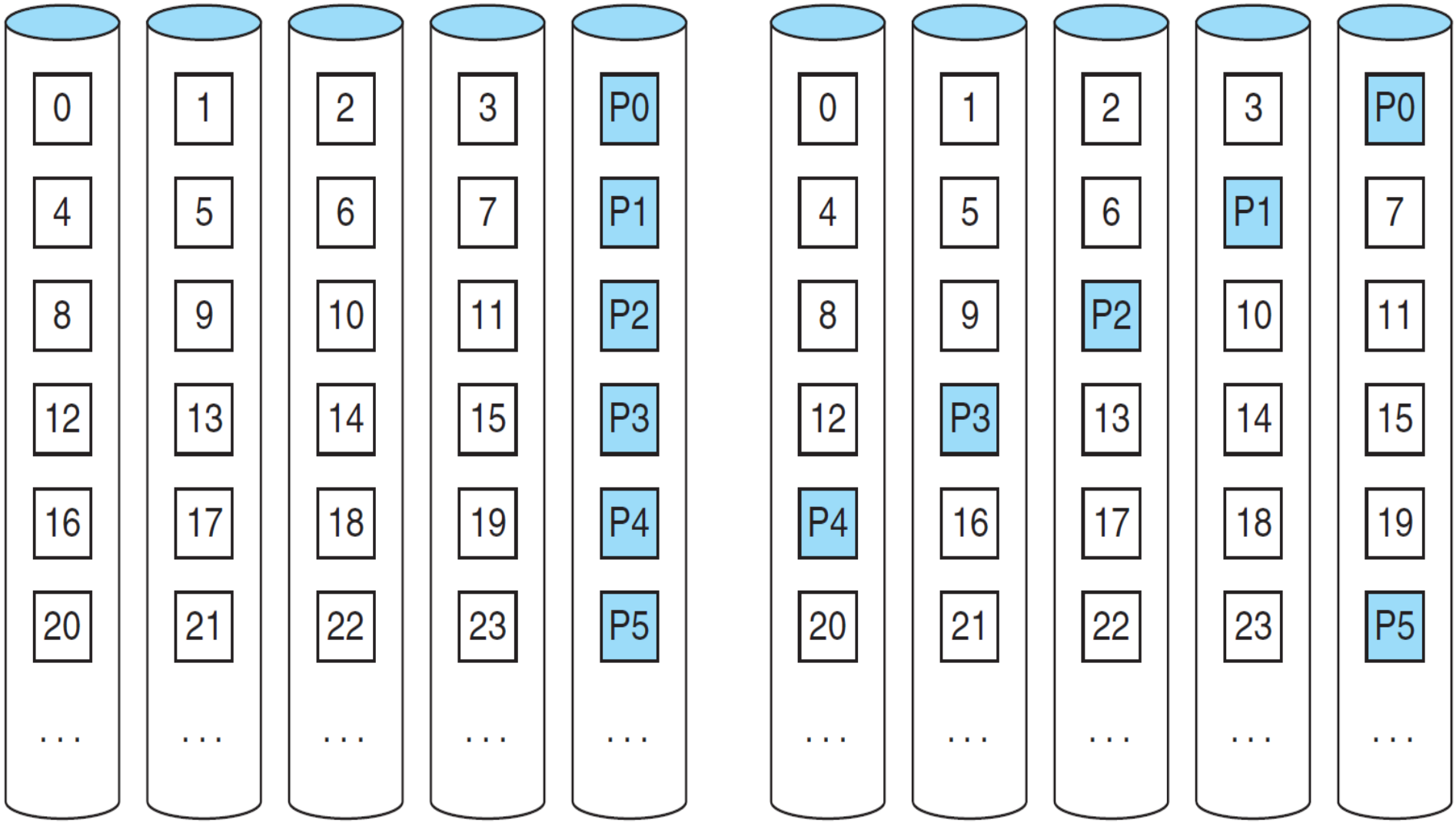
RAID 5: High I/O Rate Interleaved Parity

Independent writes possible because of interleaved parity

Example: write to D0, D5 uses disks 0, 1, 3, 4



RAID 4 vs RAID 5



使用DRAM的二级存储

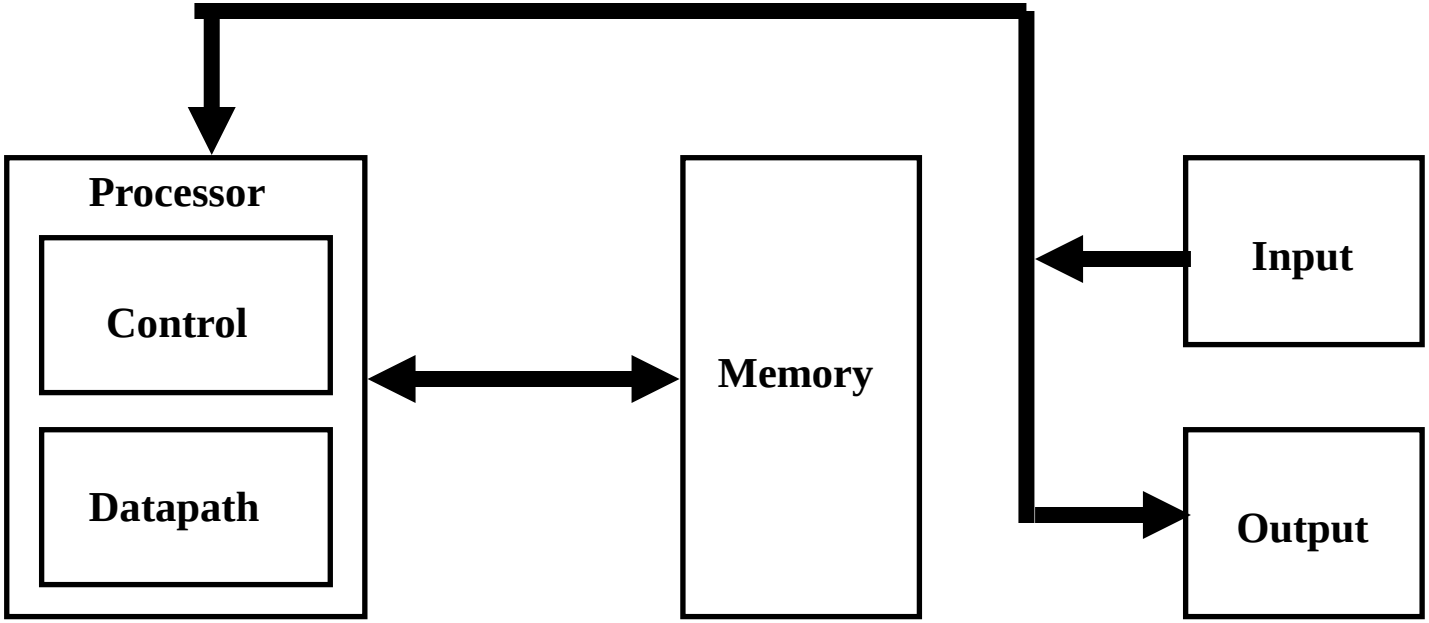
- 可以按两种方式使用DRAM作为第二级存储：
  - 固态盘（Solid state disk）
  - 扩展存储
- 固态盘：
  - 象磁盘一样进行操作，但是
    - 更快
    - 成本更高
- 扩展存储：
  - 允许数据块从主存移进和移出的较大存储器

I/O系统小结

- 磁盘 I/O基准程序：
  - 超级计算机应用程序：主要关心数据传输率
  - 事务处理：主要关心 I/O速率
  - 文件系统：主要关心文件访问

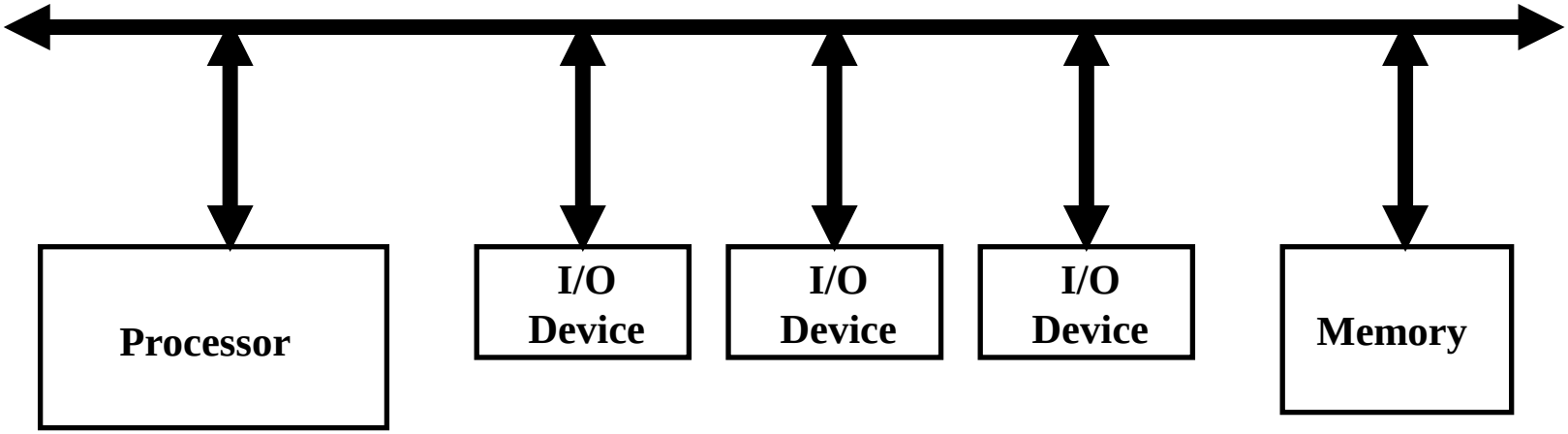
- 磁盘访问时间包括以下三部分：
  - 寻道时间：广告数值为 12 -- 20ms，现实情况可能更低
  - 旋转时延： 7200RPM：4.2ms；5400 RPM： 5.6 ms
  - 传输时间：每秒 2 -- 4 MB

总线：将I/O与处理器和存储系统连接起来



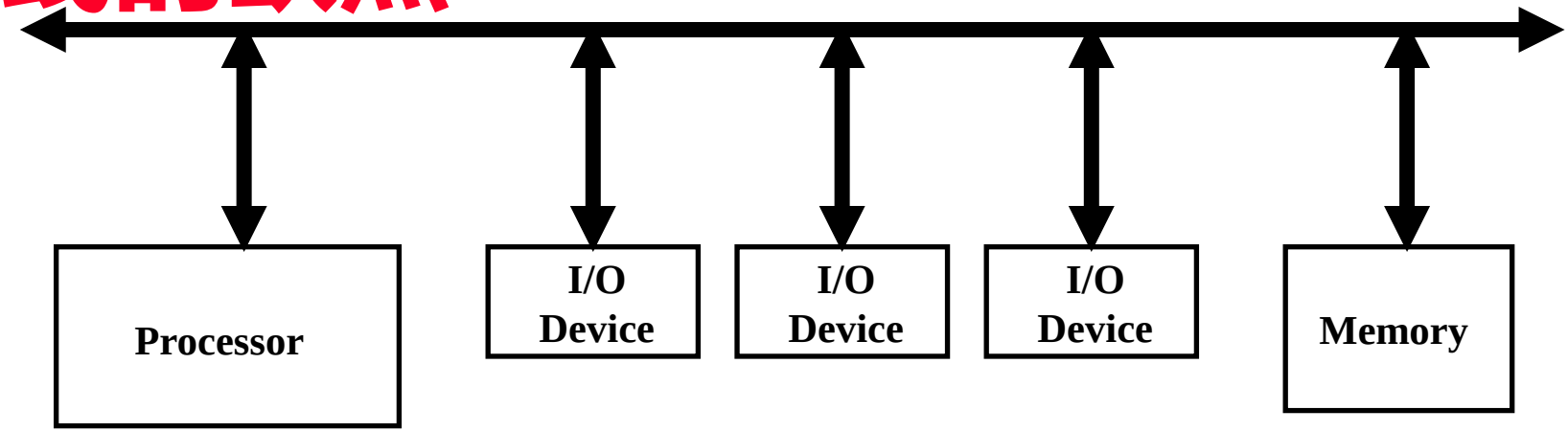
- 总线是一组共享的通信链路
- 它使用一组线路将多个子系统连接起来

总线的优点



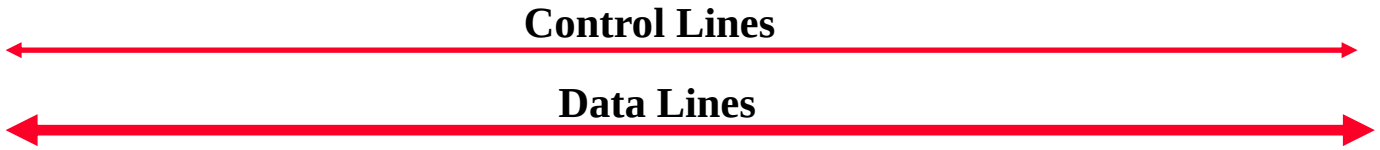
- 多功能性：
  - 易于增加新设备
  - 外设可在多个使用相同总线标准的计算机系统之间移动
- 低成本：
  - 可以以多种方式共享使用单一一组线路

总线的缺点



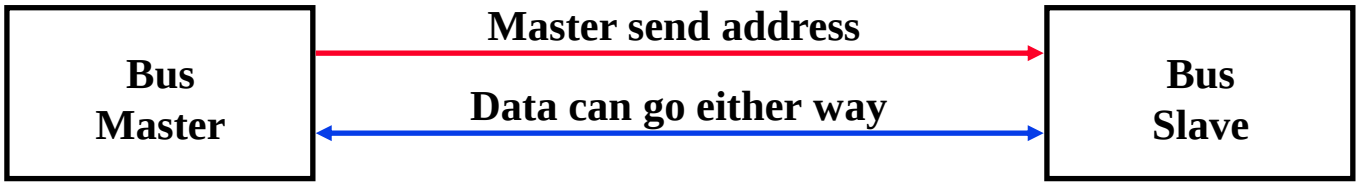
- 可能导致通信瓶颈
  - 总线的带宽制约了最大 I/O吞吐率
- 最高总线速度主要受制于：
  - 总线的长度
  - 总线上设备的数目
  - 需要支持多种设备的范围，特别是这些设备具有：
    - 时延差异很大
    - 数据传输率差异很大

总线的典型组织



- 控制线：
  - 信号请求和应答
  - 指示数据线上的信息类型
- 数据线在源设备和目的设备之间传递信息：
  - 数据 和 地址
  - 复杂命令
- 一个总线事务 包括 两部分：
  - 发送地址
  - 接收 或 发送 数据

主设备 与 从设备

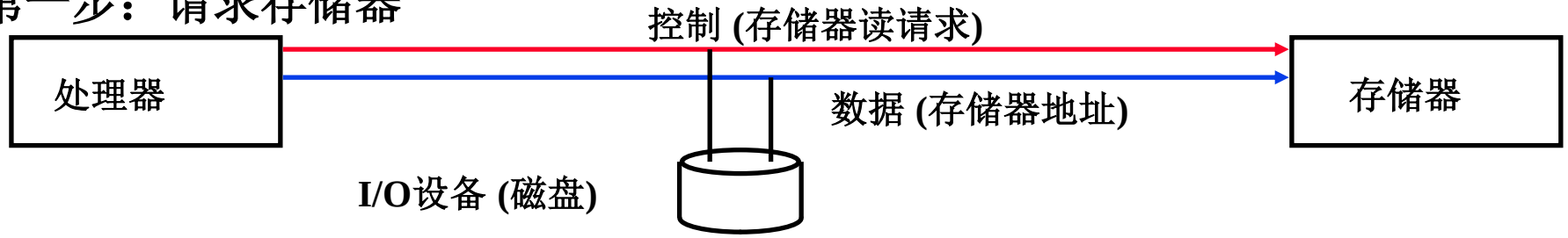


- 一次总线事务 包括 两部分：
  - 发送地址
  - 接收 或 发送 数据
- 主设备（Master）是
  - 通过发送地址 来启始 总线事务 的设备
- 从设备 通过 下列过程对下述地址产生反应：
  - 如果主设备请求数据，就发送数据给主设备
  - 如果主设备要发送数据，就接收来自主设备的数据

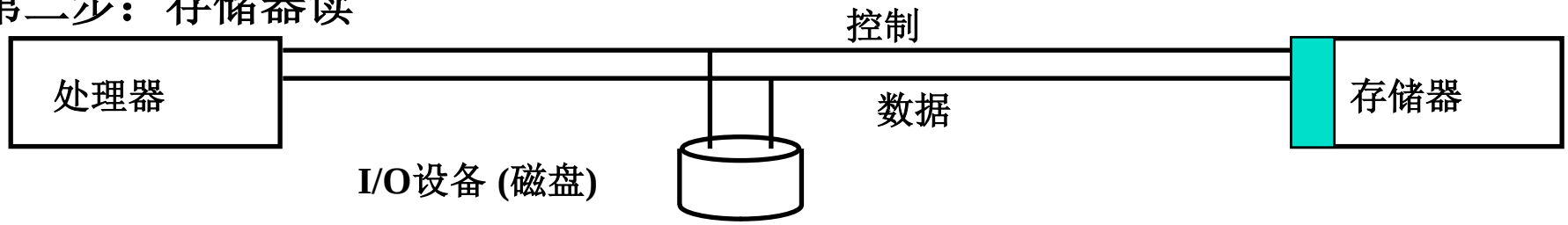
输出操作

- 这里输出是指处理器发送数据到I/O设备

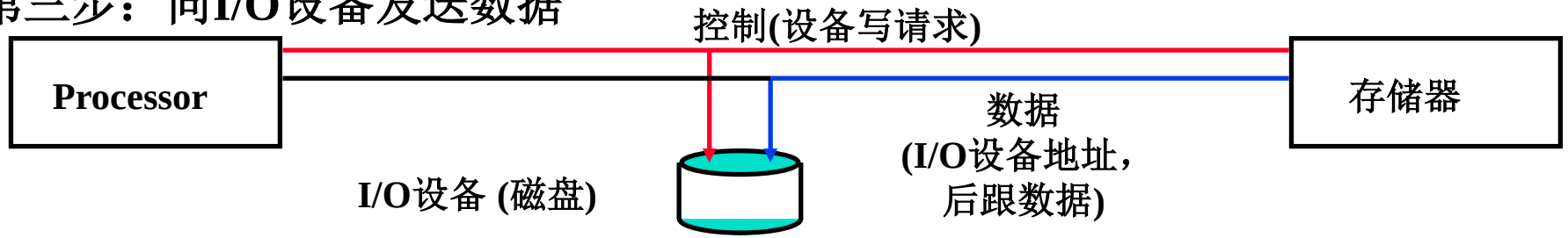
第一步：请求存储器



第二步：存储器读

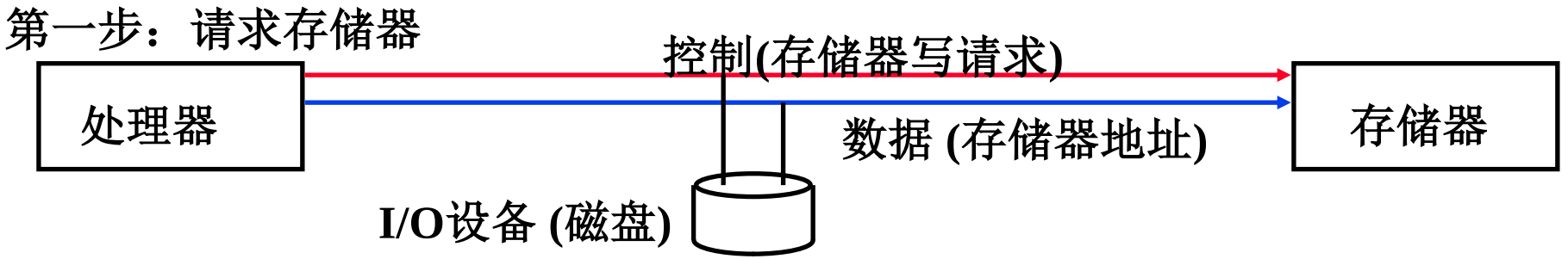


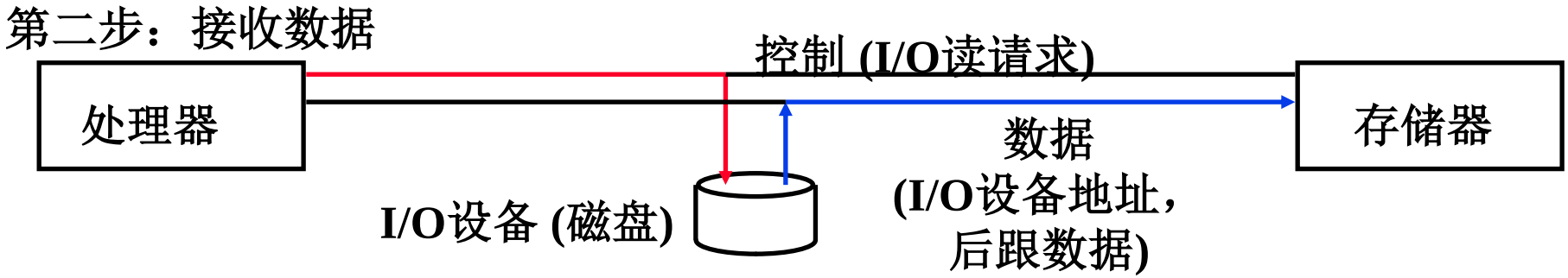
第三步：向I/O设备发送数据



输入操作

- 这里输入是指处理器从I/O设备接收数据





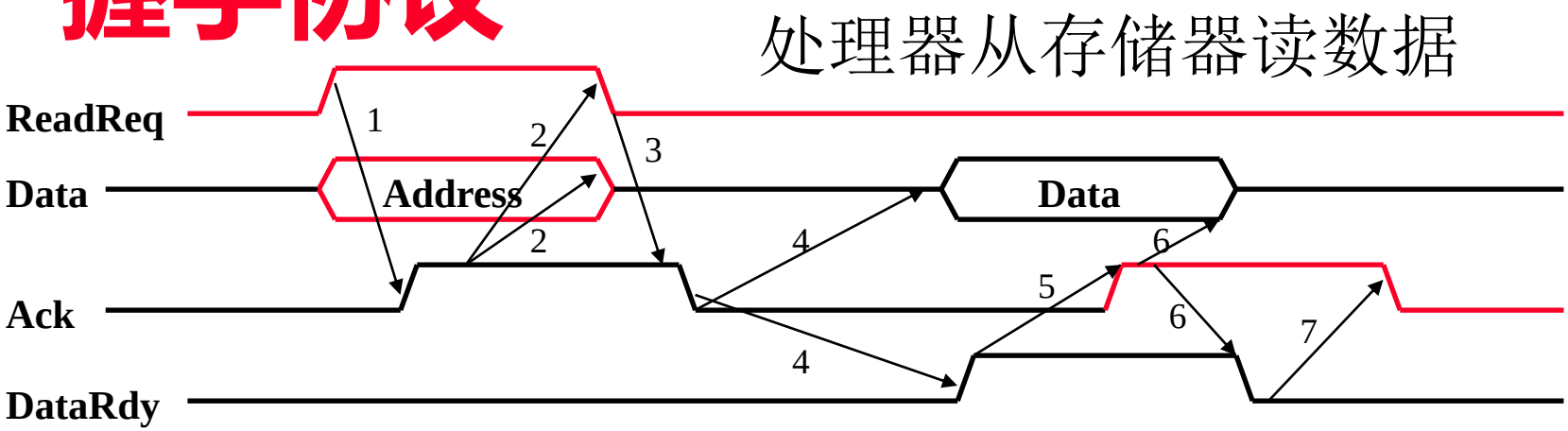
总线类型

- 处理器-存储器总线 (面向设计)
  - 距离短、速度快
  - 仅需要与存储系统匹配
    - 可达最大存储器到处理器带宽
  - 直接与处理器相连
- I/O总线(工业标准)
  - 通常，更长也更慢
  - 需要与广泛的 I/O设备匹配
  - 与处理器-存储器总线或底板总线相连
- 底板总线 (工业标准)
  - 底板（Backplane）：底盘（chassis）内的互联结构
  - 允许处理器、存储器和I/O设备共存
  - 成本优势：所有部件共享一条总线

同步 和 异步 总线

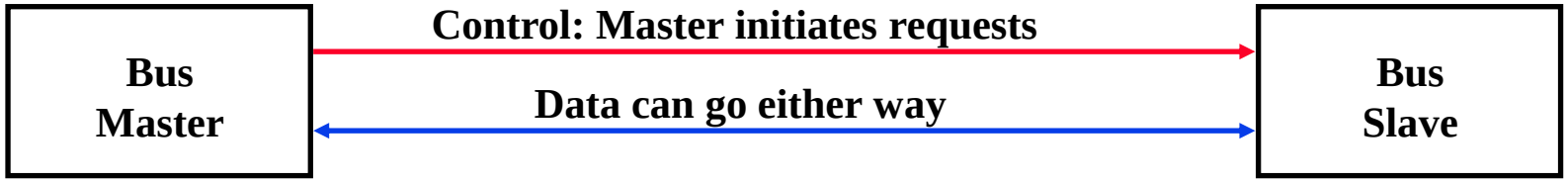
- 同步总线：
  - 在控制线中包括一个时钟线
  - 采用与时钟有关的一组固定通信协议
  - 优点：使用的逻辑非常少，可以以非常高的速度运行
  - 缺点：
    - 总线上的每个设备都必须以相同的时钟频率运行
    - 为了避免时钟扭斜，如果这种总线的速度很快，那么它就不能太长
- 异步总线：
  - 它没有时钟驱动
  - 可以适用于很广泛的设备
  - 它很容易增长，而不需考虑时钟扭斜问题
  - 需要握手协议I

握手协议



- 三条控制线
  - **ReadReq**：指示对存储器有一次读请求
    - 同时，地址被放置在数据线上
  - **DataRdy**：指示现在在数据线上的数据已经准备好
    - 同时，数据被放置在数据线上
  - **Ack**：向另一方应答 **ReadReq** 或者 **DataRdy**

获得对总线的访问权

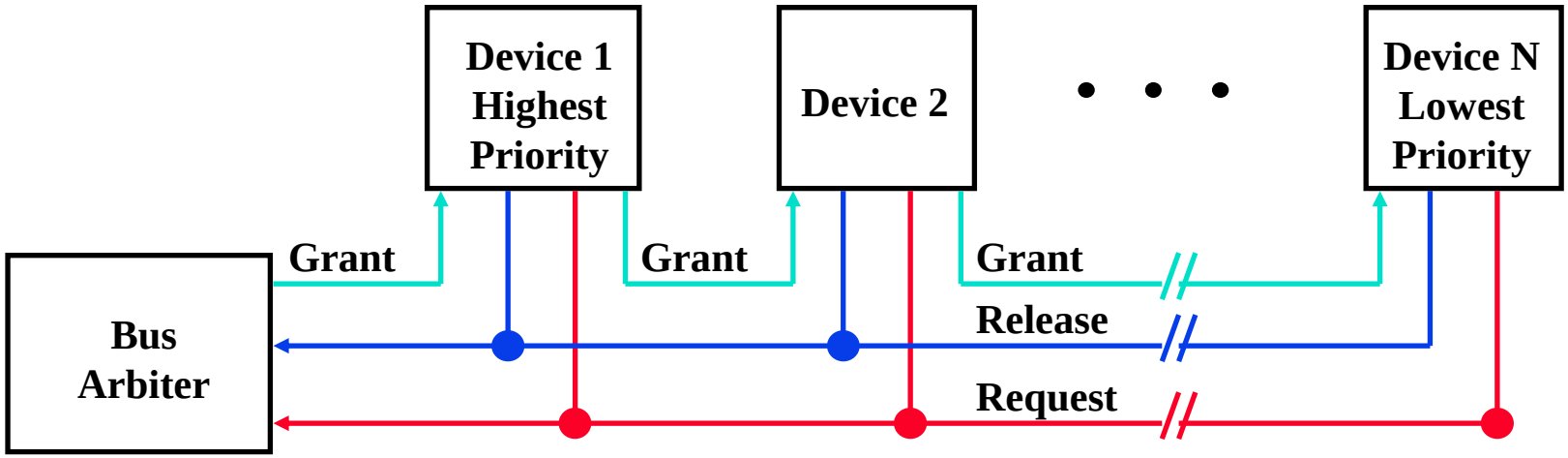


- 在总线设计中，最重要的一个论题：
  - 需要使用某总线的设备是如何获得并保留对该总线的使用权的？
- 通过主-从安排，可以避免混乱：
  - 只有总线主设备才能够控制对总线的访问：
    该主设备发起并控制所有的总线请求
  - 从设备对读和写请求发出响应
- 最简单的一个系统：
  - 处理器是唯一的总线主设备
  - 所有的总线请求都必须受到处理器的控制
  - 主要缺陷：在每次总线事务中都必须有处理器参与

多个潜在总线主设备：需要仲裁

- 总线仲裁策略：
  - 希望使用总线的总线主设备发出总线请求
  - 该总线主设备在获得请求许可之前不能使用该总线
  - 该总线主设备在使用总线结束之后必须向仲裁器发出信号
- 总线仲裁策略通常需要平衡以下两个要素：
  - 总线优先权：最高优先权的设备必须被最先服务
  - 公平：即使是最低优先权的设备也不能永远得不到总线服务
- 总线仲裁策略大致可分为如下四类：
  - 基于自测的分布式仲裁：每个希望使用总线的设备都将代表自身的标志码放置在总线上；
  - 基于冲突监测的分布式仲裁：以太网使用该策略；
  - 菊花链（Daisy chain）仲裁
  - 集中、并行仲裁

菊花链总线仲裁策略



- 优点：简单
- 缺点：
  - 不能保证公平：低优先级的设备可能永远得不到服务
  - 使用菊花链准用信号也会限制总线的速度

操作系统的职责

- 操作系统可以被看为以下两部分的接口：
  - I/O硬件 与 请求 I/O的程序
- I/O系统的三大特征：
  - I/O系统被使用该处理器的多个程序共享
  - I/O系统通常使用中断（外部产生的意外事件） 来通信有关 I/O操作的信息
    - 中断必须由操作系统处理，中断将导致系统进入管态（supervisor mode）
  - 对 I/O设备的低级控制非常复杂：
    - 管理一组并发事件
    - 正确控制设备的需求必须非常详细

操作系统的需求

- 对共享I/O资源提供保护
  - 保证用户程序只能访问该用户有权访问的那部分I/O设备
- 提供访问设备的一种抽象：
  - 提供处理低级设备操作的例程
- 处理I/O设备产生的中断
- 提供公平访问共享I/O资源的策略
  - 所有用户程序必须平等访问I/O资源
- 对访问进行调度以求增大系统的吞吐率

OS和I/O系统通信需求

- 操作系统必须能够防止：
  - 用户程序与 I/O设备之间的通信
- 如果用户程序能够直接对I/O进行操作，那么
  - 就不能提供对共享I/O资源的保护了。
- 需要三种类型的通信：
  - OS必须能够给I/O设备提供命令
  - 当 I/O 设备完成操作或遇到错误时，它必须能够通报OS。
  - 数据必须在存储器和I/O设备之间传输

给 I/O设备发送命令

- 可以使用两种方式对I/O设备寻址：
  - 专用 I/O指令
  - 存储器映像 I/O

- 专用 I/O指令指明：
  - 设备号 和 命令字
    - 设备号：处理器通过一组连线（这组连线是I/O总线的一部分）与设备进行通信
    - 命令字：它通常在总线的数据线上发送

- 存储器映像 I/O：
  - 地址空间的一部分分配给I/O设备
  - 对这些地址进行读和写就被解释为对给I/O设备的命令
  - 防止用户程序直接发送I/O 操作：
    - I/O地址空间受到地址变换机制的保护

I/O 设备通报 OS

- 当发生下列情况时。OS需要知道：
  - I/O设备完成了某一操作
  - I/O操作遇到了错误

- 可以通过两种不同的方式实现：
  - 轮寻（Polling）：
    - I/O设备将信息放置在状态寄存器中；
    - OS定期检测状态寄存器
  - I/O中断：
    - 每当 I/O 设备需要处理器关注时，它就中断处理器继续进行正在处理的工作。

I/O中断

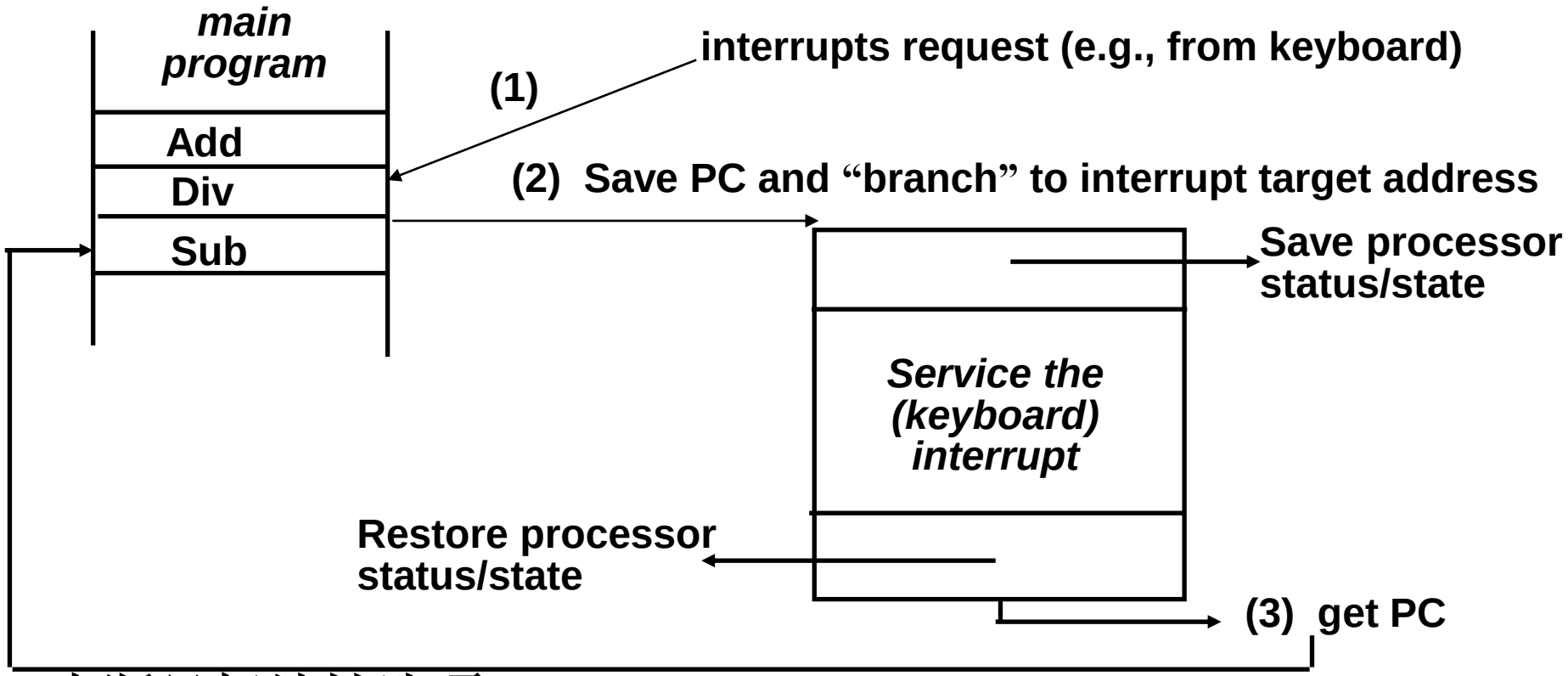
- I/O中断就像一般的意外事件，只是：
  - I/O中断是异步的
  - 需要进一步传送信息
- 相对于指令执行而言， I/O中断是异步的：
  - I/O中断并不与任何指令相关联
  - I/O中断并不防止任何指令继续执行
    - 我们可以在我们自己认为合适的时候处理这种中断
- I/O中断比一般意外事件更加复杂：
  - 需要传达产生中断的设备的身份信息
  - 中断请求具有不同的紧急性
    - 需要对中断请求优先排队

程序中断/意外事件硬件

- 硬件中断服务：
  - 保存 PC (或在流水机器中多个 PC值)
  - 抑制正在被处理的中断
  - 转移到中断服务例程
  - 选项：
    - 保存状态、保存寄存器、保存中断信息
    - 改变状态、改变操作模式、获取中断信息

- I/O中断的一个有利特点：
  - 异步性：不与指定指令相关联
  - 可以在流水线的最方便的地方来处理它！

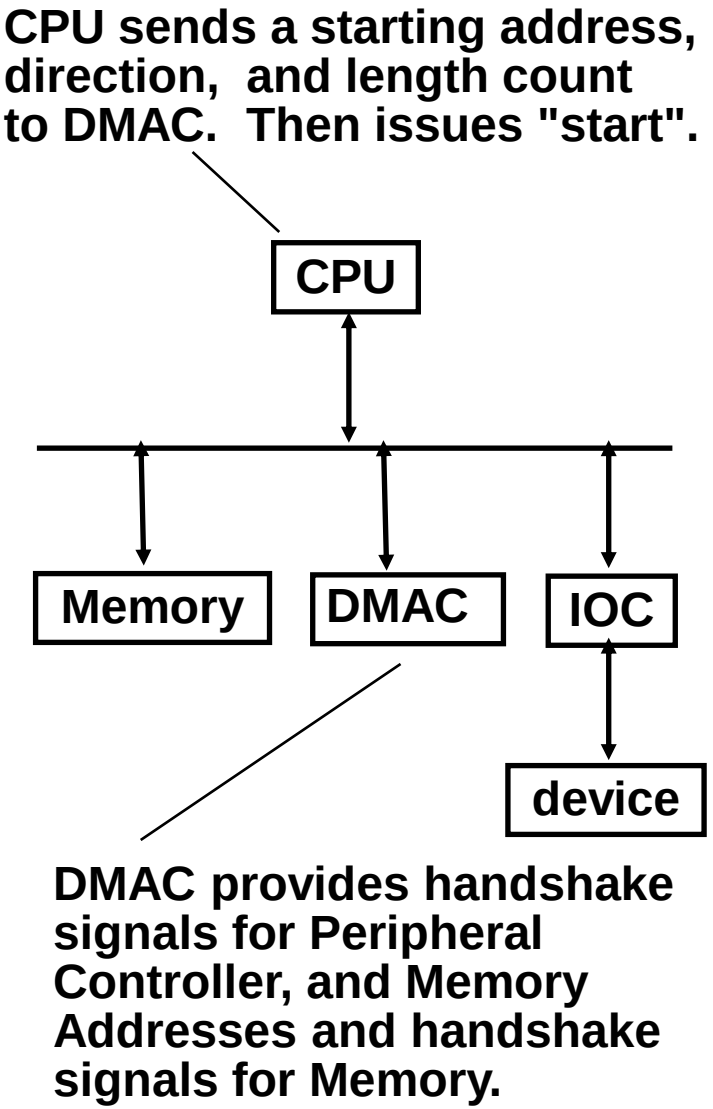
从程序员来看



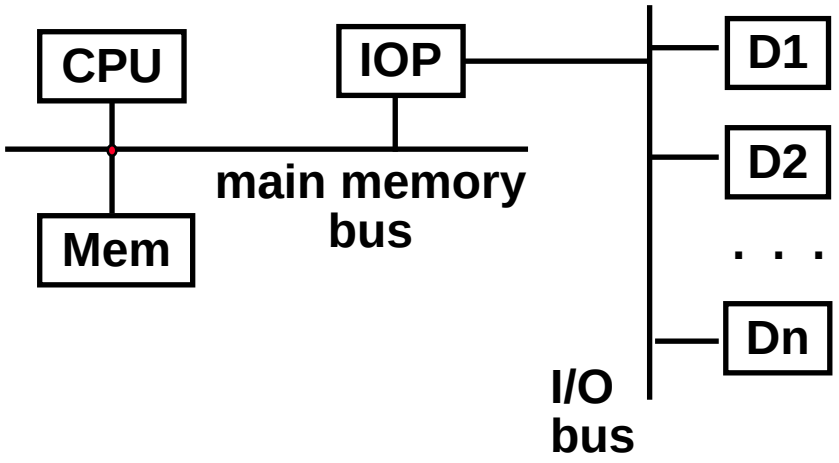
- 中断目标地址选项：
  - 通用：转移到所有中断的公共地址，然后软件对原因进行译码并指出下一步做什么
  - 专用：根据中断类型和/或级别自动转移到不同的地址—向量化中断

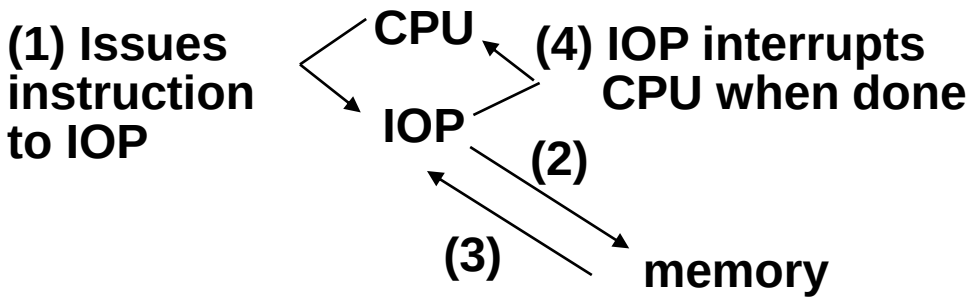
把CPU从I/O处理中解放出来：DMA

- Direct Memory Access (DMA):
  - External to the CPU
  - Act as a maser on the bus
  - Transfer blocks of data to or from memory without CPU intervention



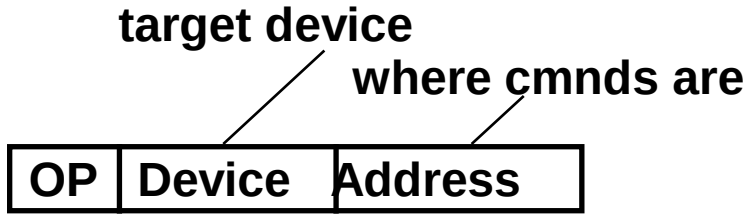
把CPU从I/O处理中解放出来：IOP





Device to/from memory transfers are controlled by the IOP directly.

IOP steals memory cycles.



IOP looks in memory for commands