计算机组成原理
(2014级)

计算机组成原理课程组

（刘旭东、肖利民、牛建伟、栾钟治）

Tel： 82316285

Mail：liuxd@buaa.edu.cn

liuxd@act.buaa.edu.cn

第八讲：外部存储与虚拟存储

一．外部存储设备

1. 磁表面存储器
2. 光盘存储器

二．虚拟存储系统

1. 概述
2. 页式虚拟存储系统

磁表面存储原理

❖磁表面存储器

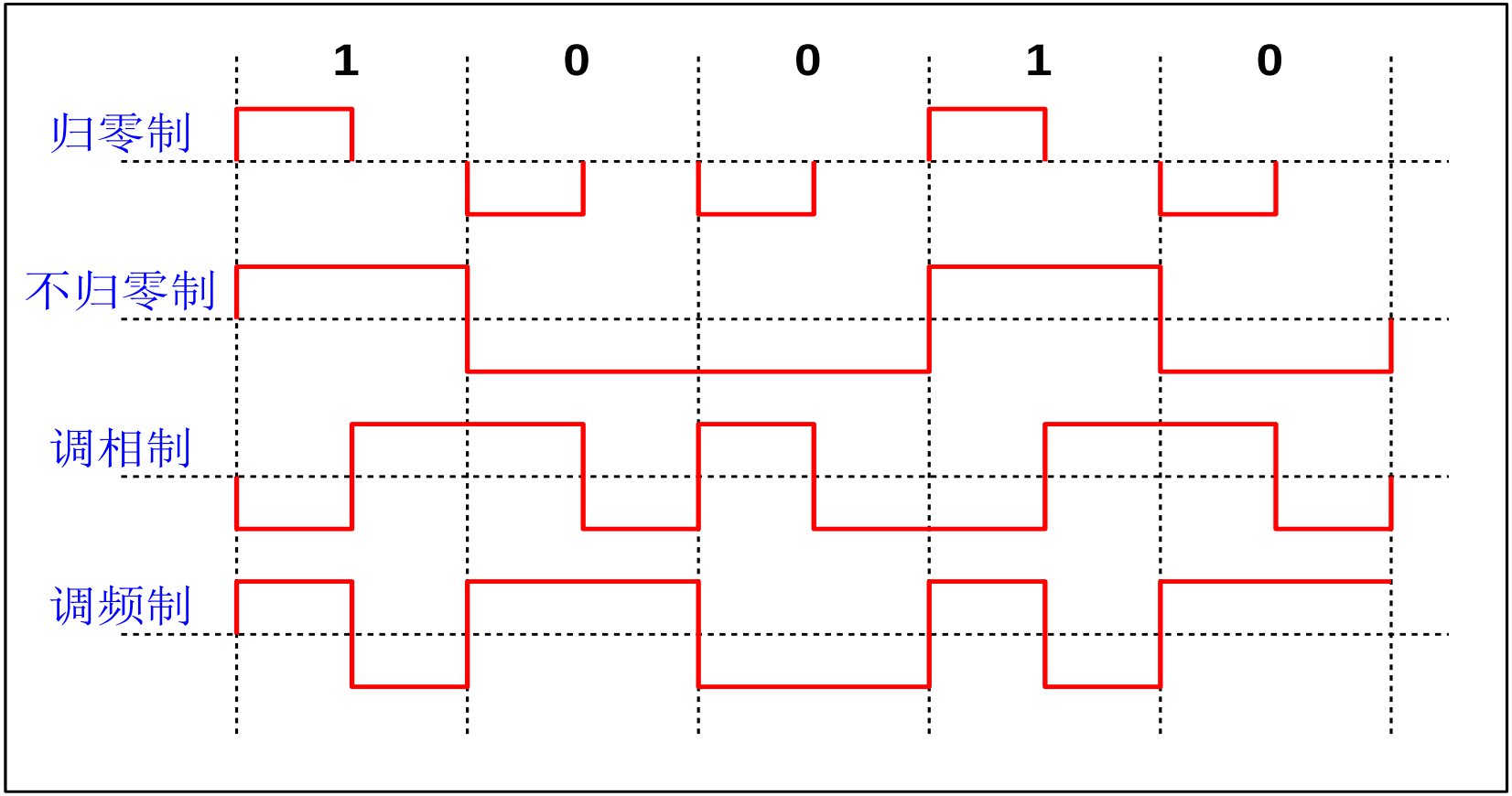
- ➢磁头：体积小，重量轻；
- ➢软盘采用接触方式，硬盘采用浮动方式（浮动磁头，薄膜磁头）
- ➢磁记录材料：极细的γ－Fe_2O_3颗粒，涂在（或喷射）在盘面上，形成细密、均匀、光滑的磁膜。
- ➢片基（载体）：塑料（软盘），金属（硬盘）

❖记录原理

- ➢通过磁头与介质的相对运动完成读写操作。
- ➢写入：根据写入代码确定写入驱动电流的方向，使磁表面被磁化的极性方向不同，以区别“0”和“1”；
- ➢读出：磁头相对磁化单元做切割磁力线运动，磁化单元的极性决定了感应电势的方向，以此区别“0”和“1”。

磁记录编码方式

- 磁记录编码方式实际上是写入电流的变化方式



磁记录编码方式

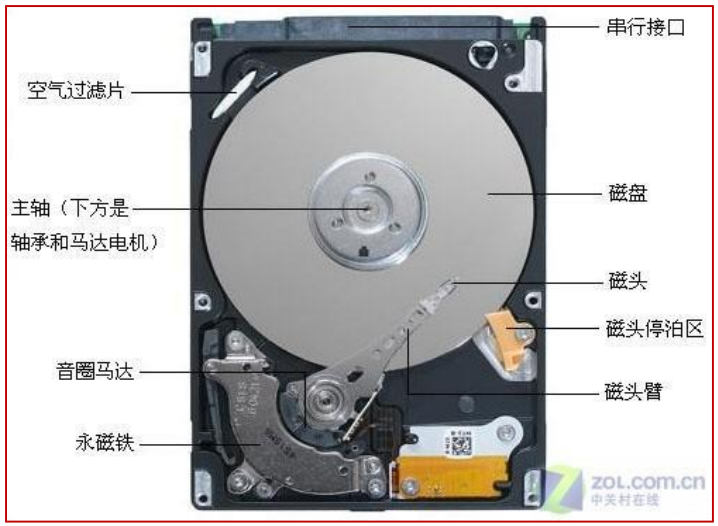
❖评价记录方式的主要指标

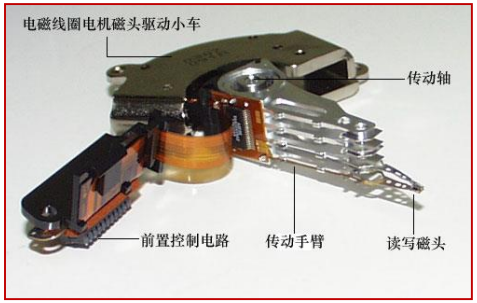
- 可靠性：归零制低，调相制高；
- 编码效率：用记录一位信息的最大磁化翻转次数表示；FM与PM为2，NRZ为1；
- 自同步能力：能否直接从读出的信号中提取同步信号；NRZ没有自同步能力，PM，FM等都具备自同步能力；

硬磁盘基本结构

❖结构

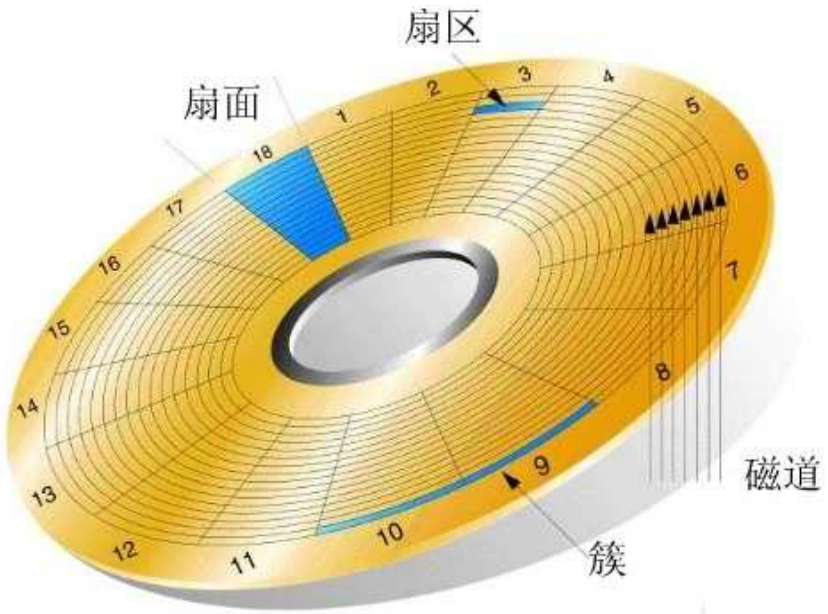
- ➢全密封：浮动磁头组件、磁头驱动机构、盘片和主轴组件和前置控制电路等密封在一起。
- ➢磁头：非接触式浮动磁头，盘面分启停区和数据区。不工作时，磁头停留在启停区；工作时，磁盘高速旋转带动气流使磁头漂浮在磁表面上方，头盘间隙仅有**0.1**微米**~0.3**微米；
- ➢读写电路：安装在磁头臂接近磁头的地方，以减少干扰；
- ➢旋转速度：**3600RPM，7200RPM，10000RPM，15KPRM**；一般等角速度旋转。





硬磁盘基本结构

- 磁盘存储结构
  - 盘面：一个磁盘包含若干盘片，每盘片分上下两个盘面，每个盘面由一磁头负责读写
  - 磁道：磁盘表面的同心圆环
  - 扇区：每个磁道包含若干扇区，扇区是磁盘数据读写的最小单位，扇区容量一般为512 ~ 4096 字节（默认为512字节，默认每个磁道包含的扇区数是相同）
  - 柱面（cylinder）：不同盘面相同半径磁道构成的圆柱



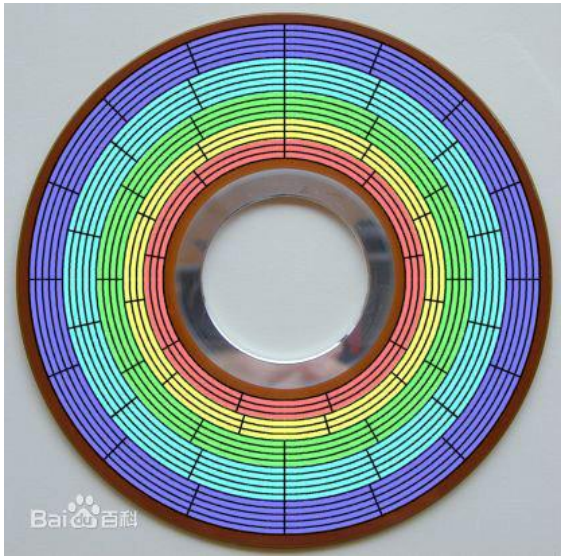
磁盘上的磁道、扇区和簇

- 扇区的地址表示：

扇区地址：	Cylinder #	Head #	Sector #

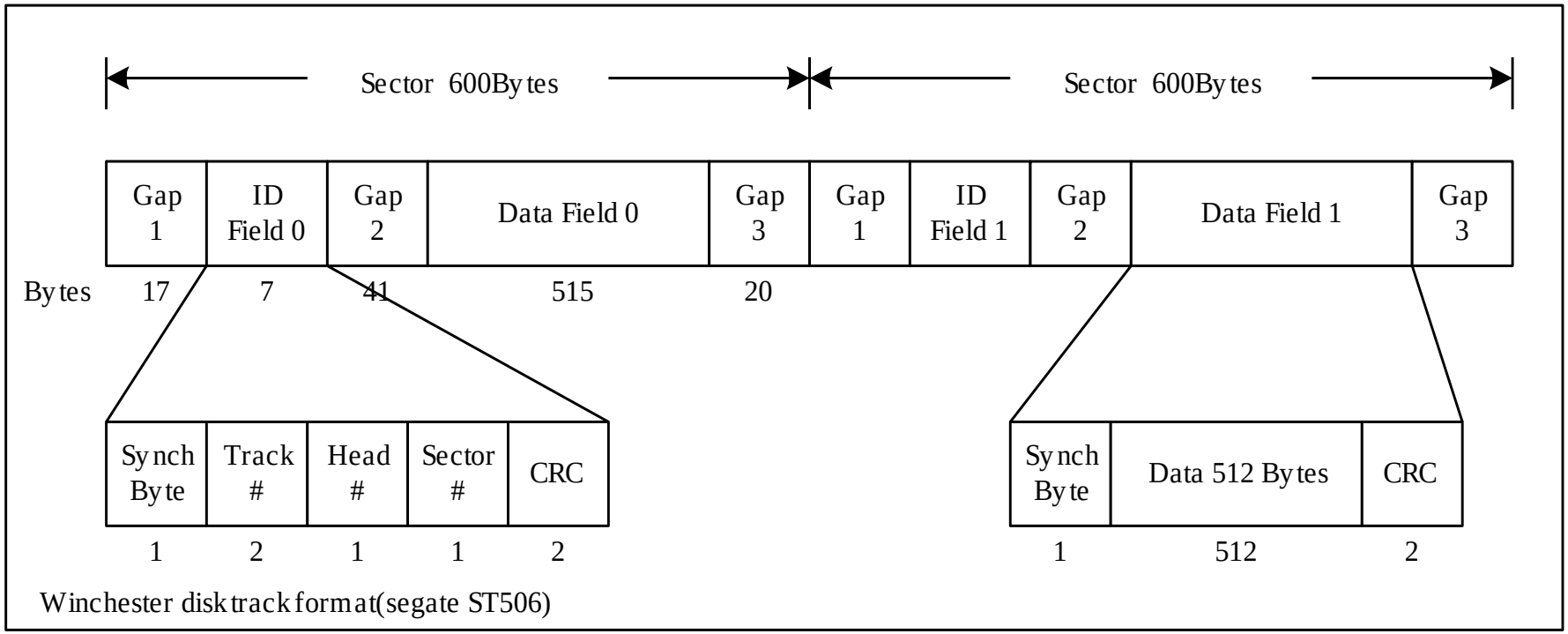
硬磁盘基本结构

- **补充：**现在的硬盘都使用ZBR（ZonedBitRecording分区域记录）技术，盘片表面从里向外划分为数个区域，不同区域的磁道扇区数目不同，同一区域内各磁道扇区数相同，盘片外圈区域磁道长扇区数目较多,内圈区域磁道短扇区数目较少，大体实现了等密度，从而获得了更多的存储空间。大多数产品划分了16个区域，最外圈的每磁道扇区数正好是最内圈的一倍（373～746正是啦）。这样的话，当磁盘主轴马达按一定角速度（每秒N转）旋转的时候，越往外，线速度越大，单位时间内读取的扇区数就越多，传输率就越高。（是不是也可以稍微理解通常把系统盘数据放在磁盘最外圈的原因了）



硬磁盘基本结构

❖ 扇区数据格式示例（**Segate ST506 磁盘扇区格式**）



磁盘的性能参数

❖性能指标

- 记录密度
  - 道密度：磁盘沿半径方向单位长度的磁道数；
  - 位密度：单位长度磁道记录二进制的位数。
- 存储容量：磁头数×磁道（柱面）数×每道扇区数×每扇区字节数
- 寻道时间 T_S：磁头从当前位置定位到目标磁道所需时间（用平均值表示）；
- 寻区时间 T_w：磁头定位到目标磁道后，等待目标扇区旋转到磁头下所需的时间（用平均值表示）；
- 访问时间（也称寻址时间）T_A：$T_A=T_S+T_w$
- 数据传输率 D_r：单位时间内传输的数据位数（b/s）

软磁盘

❖软盘（Floppy Disk）

- ➢尺寸：5.25 inch, 3.5 inch
- ➢容量：360KB，1.2MB, 720KB，1.44MB

硬磁盘的类型

- **IDE硬盘**
  - **IDE（Integrated Drive Electronics）：80年代出现，主要为 IBM PC 兼容机所用的低价磁盘，由BIOS处理磁盘的读写等操作。**
- **SCSI硬盘**
  - **SCSI(Small Computer System Interface)：接口与IDE不同，具有更高的数据传输率。**
  - **SCSI接口上所有设备（不一定是磁盘）可以同时操作，这是与IDE和最大的不同之处。**

Name	Data bits	Bus Mhz	MB/Sec
SCSI-1	8	5	5
SCSI-2	8	5	5
Fast SCSI-2	8	10	10
Fast & Wide SCSI-2	16	10	20
Ultra SCSI	16(32)	20	40
Ultra2 Wide SCSI	16		80
Ultra-160m/Ultra-320m	16		160/320

硬磁盘的类型

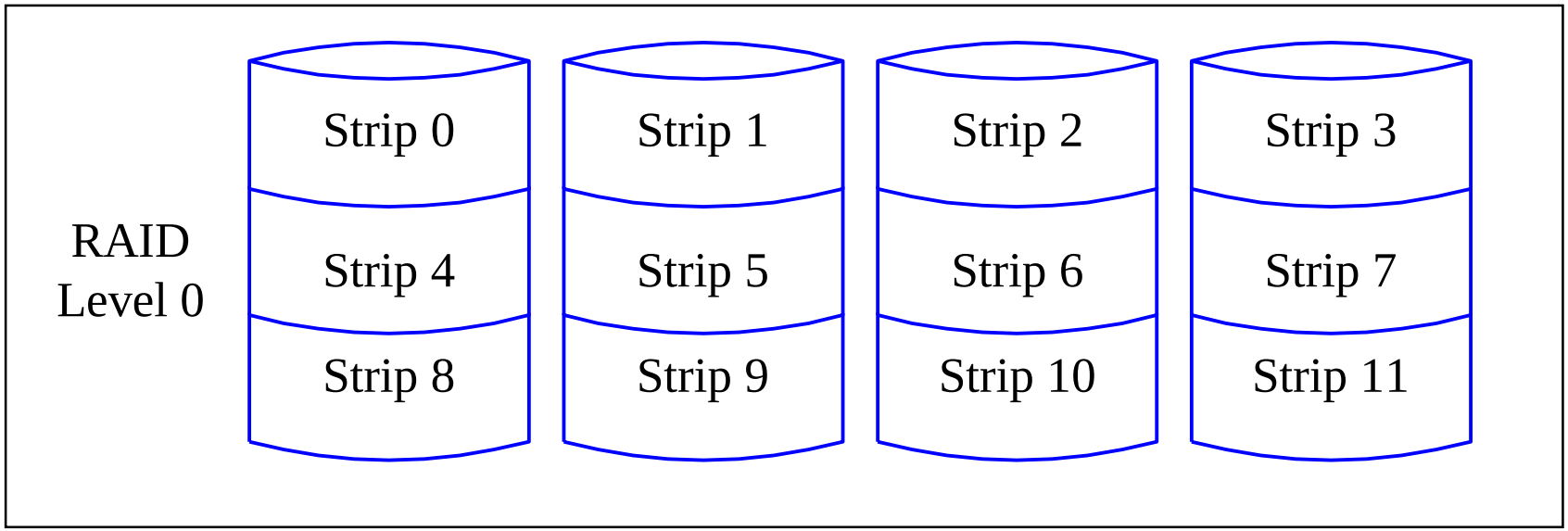
❖RAID

- Reduntant Array of Independent Disks（独立冗余磁盘阵列）
- RAID由多个物理构成，但被操作系统当成一个逻辑磁盘，数据分布在不同的物理磁盘上，冗余磁盘用于保存数据校验信息，校验信息保证在出现磁盘损坏时能够有效的恢复数据；
- RAID特点
  - 通过把多个磁盘组织在一起作为一个逻辑卷提供磁盘跨越功能
  - 通过把数据分成多个数据块（Block）并行写入/读出多个磁盘以提高访问磁盘的速度
  - 通过镜像或校验操作提供容错能力
- RAID包括六种不同模式：RAID 0, RAID1, RAID2，RAID3,RAID4和RAID 5。

硬磁盘的类型

❖RAID 0：无差错控制的带区组

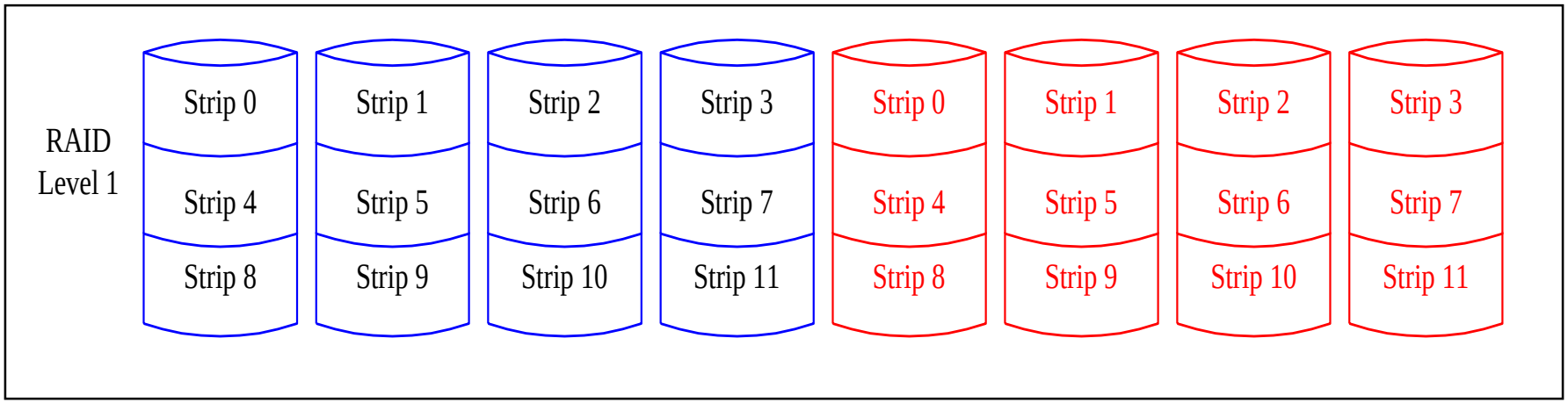
- 实际上不应属于RAID家族成员，完全没有冗余；
- 数据条带（Strip）化分布在不同的物理磁盘上。Strip可以是物理磁盘上的一块存储区（扇区或其他单位）。
- 磁盘组中每一个磁盘同一位置的磁盘区构成一个逻辑上的带区，所以一个带区分布在多个磁盘上。
- 单个I/O 操作访问的数据分布在一个带区上时，可实现I/O操作的并行处理，改善数据传输性能。



硬磁盘的类型

❖RAID 1：镜像结构

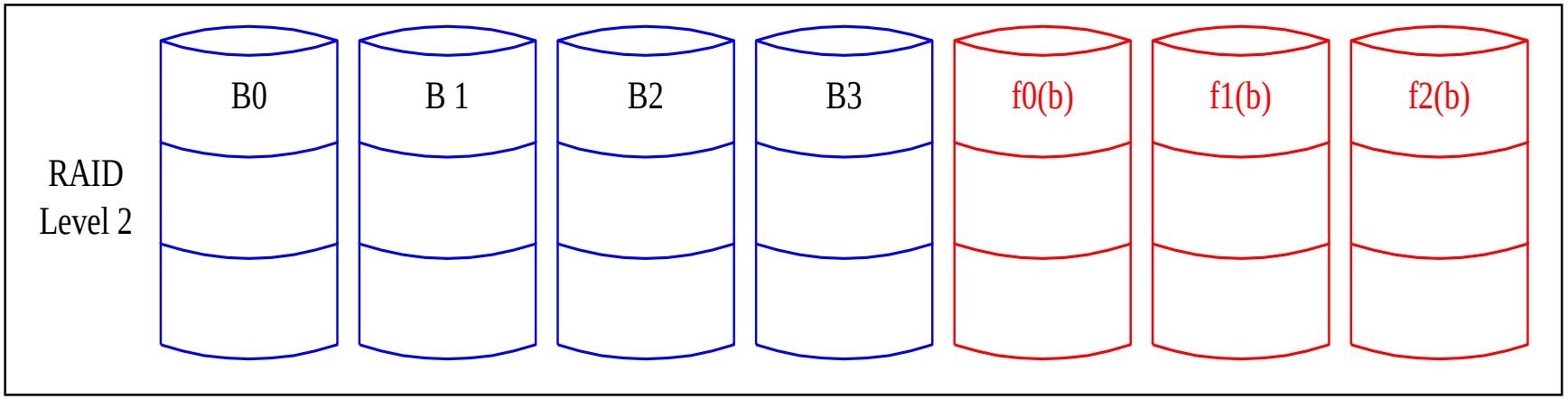
- 简单镜像磁盘冗余方案，成本太高；
- 与RAID 0类似，用户数据和系统数据条带（Strip）化分布在不同的物理磁盘上（包括镜像磁盘）。
- 读操作同时在两组磁盘中进行，数据从访问时间小的磁盘组中获得，所以，读操作性能得到改善。
- 写操作同时在两组磁盘中进行，写操作的访问时间以速度慢的为准，所以，写操作性能指标不高。
- 出现磁盘损坏时，数据恢复简单。



硬磁盘的类型

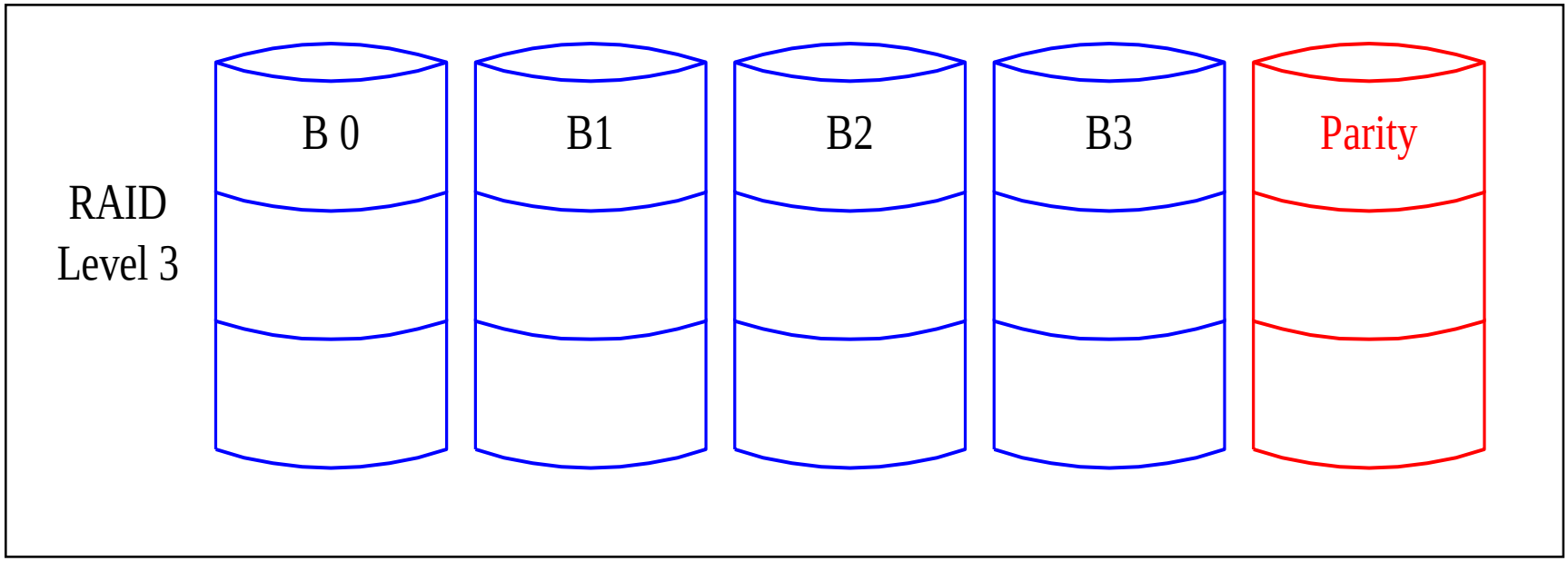
❖RAID 2：带海明校验

- ➢采用完整的并行访问技术，所有磁盘在任何时刻都并行地响应I/O 请求；磁盘组中物理磁盘处于完全同步状态，以保证任何时刻，所有磁盘的磁头都处于相同位置。
- ➢数据按较小的条带（一个字或一个字节）分布在不同的磁盘上。
- ➢根据磁盘数据计算错误校验码（比如海明码），校验码按位分布在冗余磁盘对应位置上。
- ➢数据传输率高；访问效率高；
- ➢成本比较高（比RAID1稍低）



硬磁盘的类型

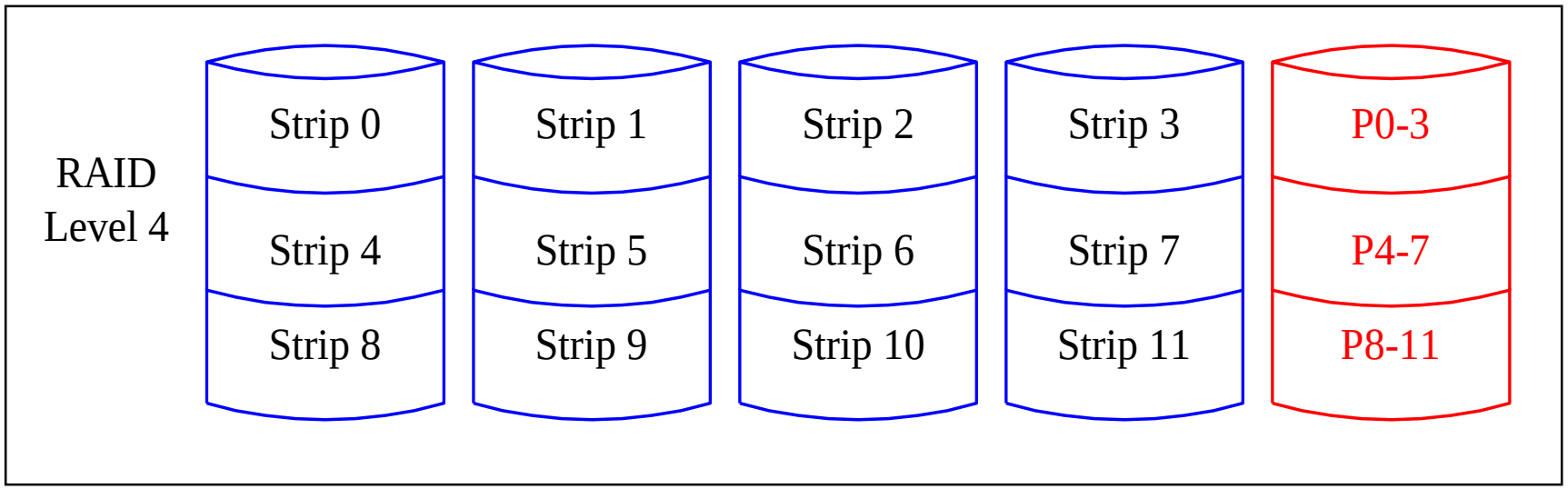
- RAID 3：带奇偶校验码的并行传送
  - 与RAID2一样，采用并行访问技术；
  - 数据按较小的条带（一个字或一个字节）分布在不同的磁盘上。
  - 校验码是简单的奇偶校验码（1位），保存在独立的冗余磁盘对应位置上。
  - 一个磁盘损坏，可以方便地实现数据恢复；
  - 数据传输率高；访问效率高；



硬磁盘的类型

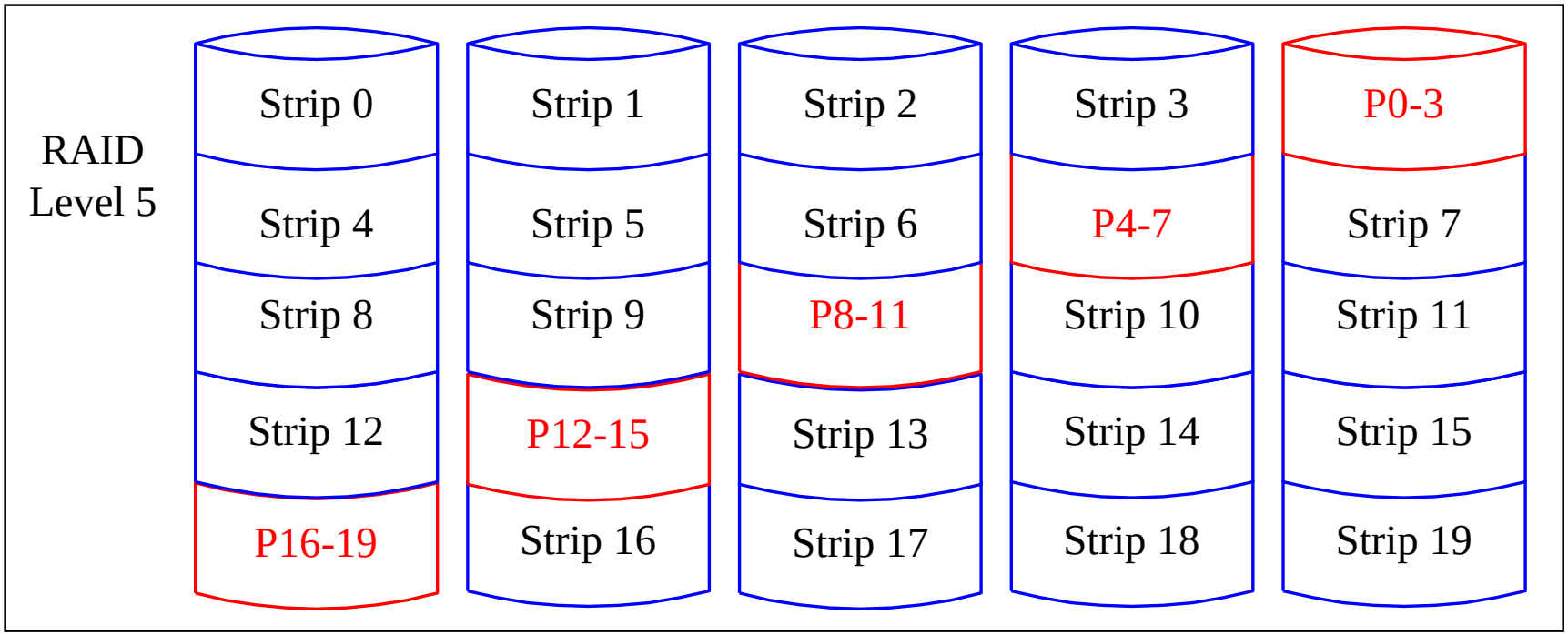
❖RAID 4：带奇偶校验码的独立磁盘结构

- ➢采用独立访问技术，每个磁盘独立工作，分散的I/O请求将得到很好的并行处理
- ➢数据按较大的条带分布在不同的磁盘上。
- ➢校验码是奇偶校验码，保存在独立的冗余磁盘对应位置上。
- ➢一个磁盘损坏，可以方便地实现数据恢复；
- ➢写操作效率较低，需要计算奇偶校验位，磁盘组中一个磁盘写操作，均需要读取原检验信息，重新计算校验信息，再写校验信息。



硬磁盘的类型

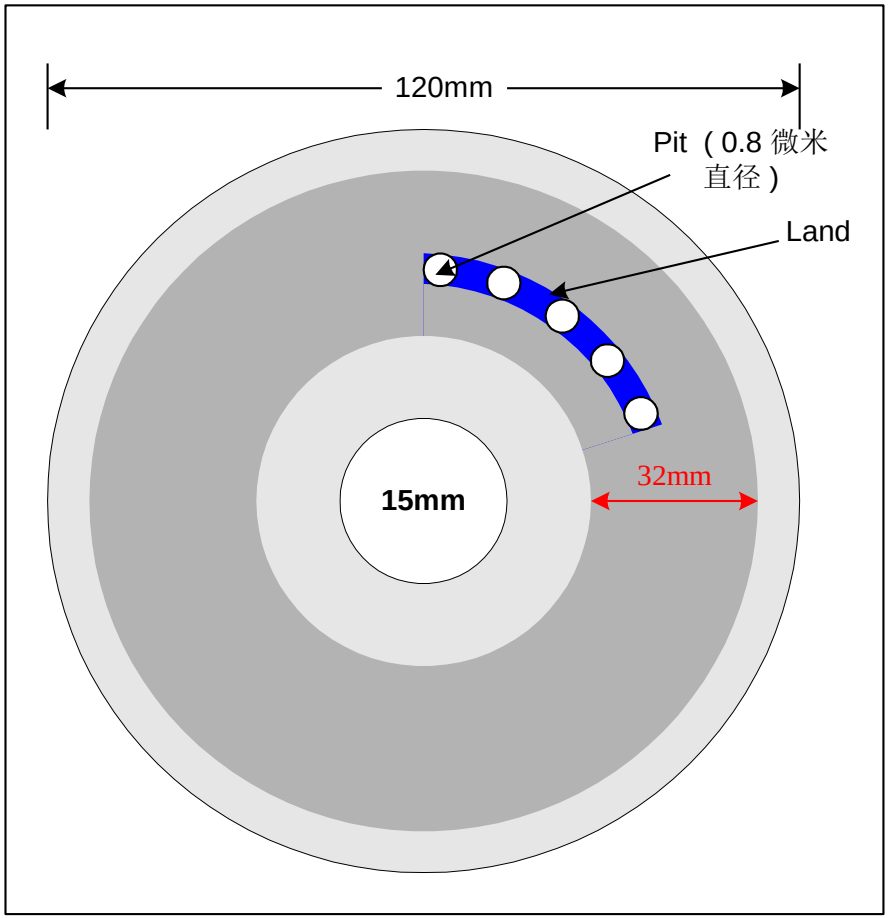
- **RAID 5：分布式奇偶校验的独立磁盘结构**
  - 与**RAID 4**的差别仅在于校验信息的保存位置；数据校验码作为条带的一部分保存在磁盘组不同的磁盘中



光盘存储器

CD-ROM

- 规格：直径120mm，厚度1.2mm，中心孔径 15mm
- 结构：树脂片基，铝反射层，保护膜，印刷层
- 数据记录区：32mm宽的环形记录带。
  - 等线速度方式：一个螺旋环环绕22188次（600环/mm，总长度约 5.6km长）
  - 等角速度方式

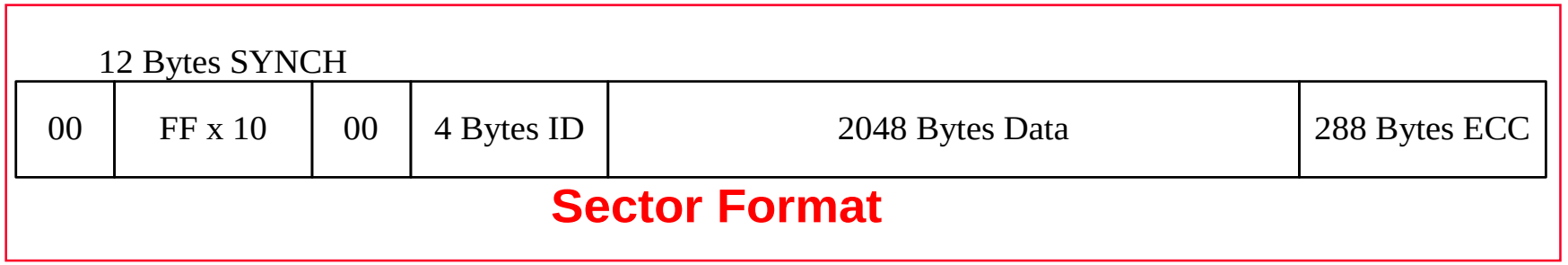


数据记录

- 凹点（Pit）表示 0
- Land 表示 1

光盘存储器

❖CD-ROM的数据格式

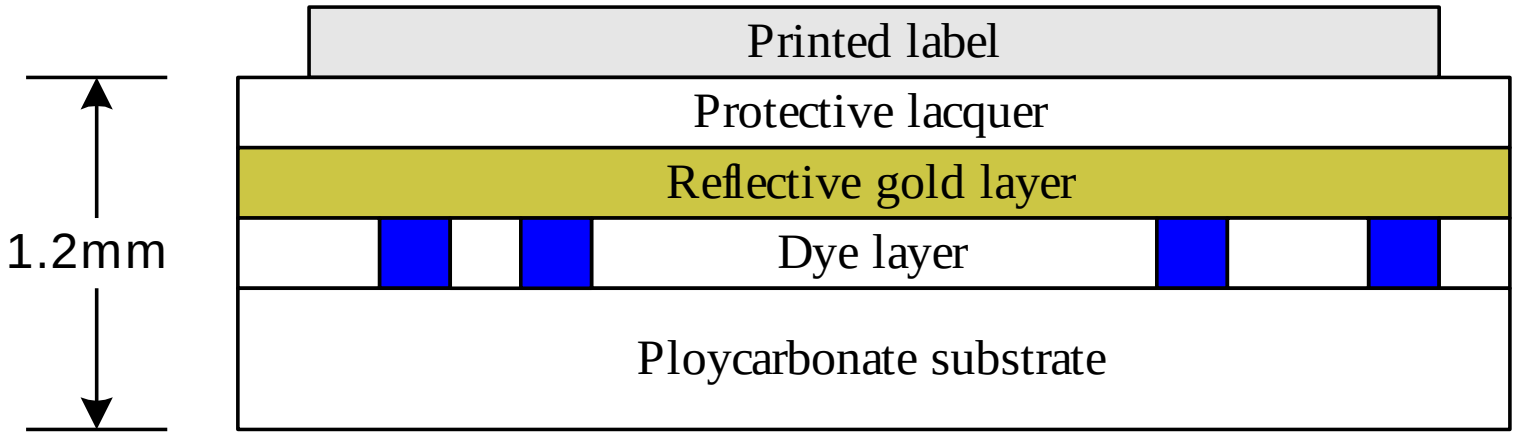


- **Symbol：14位，8位数据，6位海明校验位(看成一个Bytes)；**
- **Frame：42个连续Symbol（588bits），其中192位（24字节）存储数据，其余396位用于错误纠正与控制；**
- **Sector：98个frame构成一个Sector（总计2352Bytes）。**
- **总容量：650MB**
- **等线速度旋转时：单速：120cm/s（最内圈530RPM，（最外圈200RPM），75 Sectors/Sec（150KB/S）。**
- **制作过程：母板压模**
- **读机制：0.78微米波长红外激光，根据反射光的强度判断是0还是1；**

光盘存储器

❖CD-R（Recordables）

- 在片基（树脂）与反射层（金）中增加了一层染料层作为数据记录层，初始状态下，染料层透明，在写入状态时，高能量（8-16mw）使照射处的染料变色，变成不透明点，不可再恢复成透明状态。读出状态下(0.5mw)，根据透明不透明判断是0还是1。



光盘存储器

❖CD-RW（Rewritables）

- 与CD-R的差别是采用合金层代替染料层。一般采用银、铟、锑、碲合金。该合金具有两种稳定状态：透明状态（晶体结构）和不透明状态（无序结构），初始时为晶体结构。
- CD-RW工作时采用三种不同功率的激光：
  - 大功率（写）：合金熔化，由晶体结构变为无序结构；
  - 中等功率（擦除）：合金熔化，由无序结构变为晶体结构；
  - 小功率（读）

光盘存储器

❖DVD（Digital Video Disk）

与CD-ROM的差别：

- Pit直径更小（0.4微米）；
- 环绕密度更高（0.74微米，CDROM是1.6微米）；
- 0.65微米波长红色激光（CDROM是0.78微米的红外激光）；
- 容量：单面单层4.7GB，单面双层8.5GB，双面单层9.4GB，双面双层17GB。
- 数据传输率：单速DVD 1.4M Bytes/Sec。

第八讲：外部存储与虚拟存储

一．外部存储设备

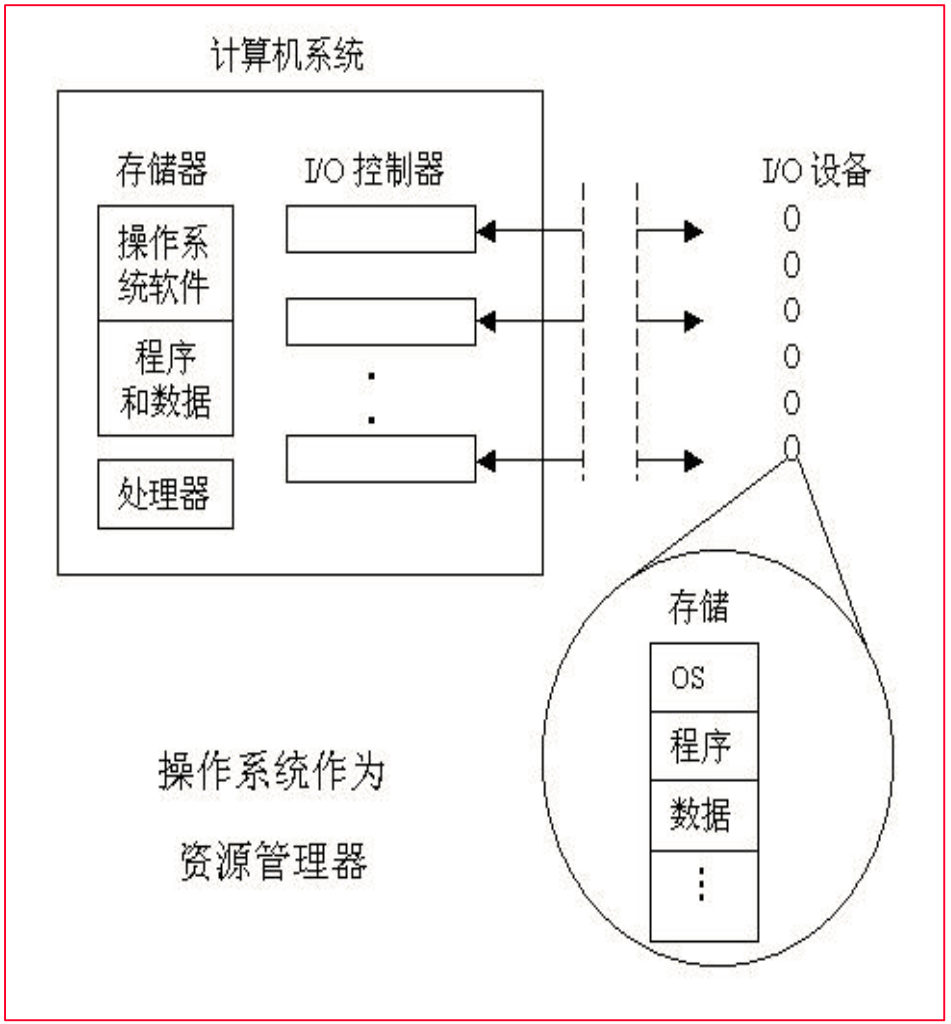
1. 磁表面存储器
2. 光盘存储器

二．虚拟存储系统

1. 概述
2. 页式虚拟存储系统

概述

- 存储器管理：操作系统的功能
  - 操作系统：合理地管理、调度计算机的硬件资源。存储器作为一种空间资源也由OS来管理；
  - CPU执行的程序：总是在操作系统和用户程序之间切换。主存中同时要存储OS和用户程序。磁盘中也要存储OS和用户程序；
  - CPU中的存储器管理部件MMU协助OS完成存储器访问。



OS为“进程”分配存储器资源，所以，先了解一下进程的概念。

概述

❖一个典型程序的转换处理过程

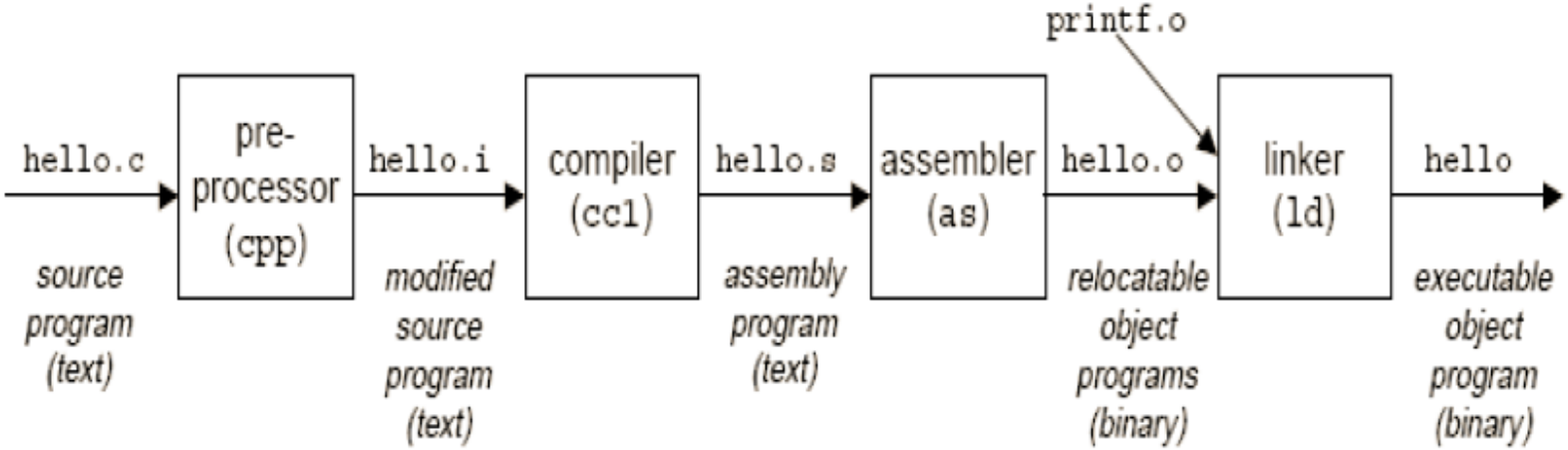
经典的“ hello.c ”C-源程序

```
1 #include <stdio.h>
2
3 int main()
4 {
5 printf("hello, world\n");
6 }
```

程序的功能是：
输出“hello,world”

hello.c的ASCII文本表示

```
# i n c l u d e <sp> < s t d i o .
35 105 110 99 108 117 100 101 32 60 115 116 100 105 111 46
h > \n \n i n t <sp> m a i n ( ) \n {
104 62 10 10 105 110 116 32 109 97 105 110 40 41 10 123
\n <sp> <sp> <sp> <sp> p r i n t f ( " h e l
10 32 32 32 32 112 114 105 110 116 102 40 34 104 101 108
l o , <sp> w o r l d \ n " ) ; \n }
108 111 44 32 119 111 114 108 100 92 110 34 41 59 10 125
```



概述

❖Hello的执行过程

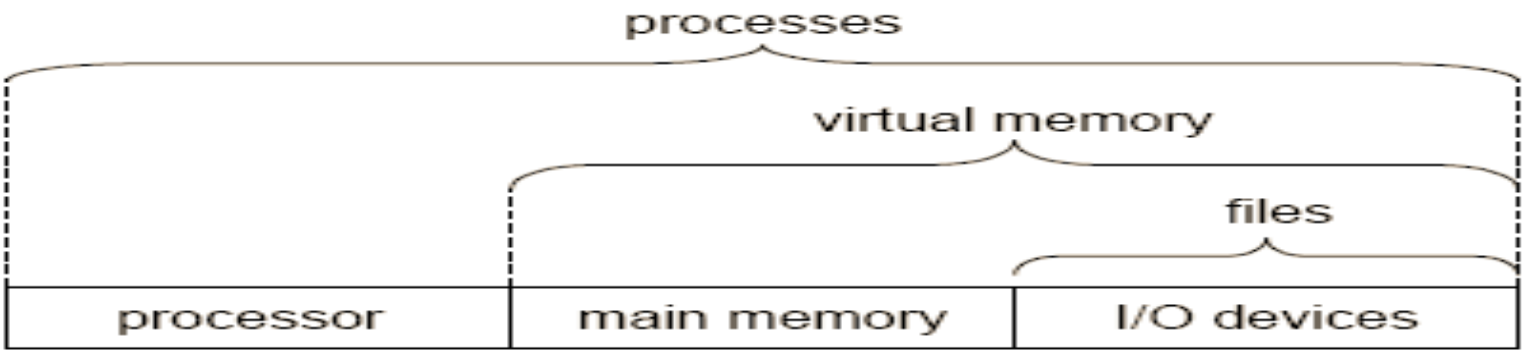
- ➢Unix系统Shell命令行输入：hello，回车
- ➢Shell程序调用驻留在内存的“加载器”程序，由加载器从磁盘上找到特定的hello目标文件，将其指令代码和数据（“hello, world\n”）从磁盘读到主存；
- ➢处理器从hello程序的指令代码开始执行；
- ➢Hello程序将“hello, world\n”串中的字节从主存读出，送到显示器输出。

```
unix> ./hello [Enter]
hello, world
unix>
```

问题：**hello**程序何时被装入？谁来装入？被谁启动？每次是否被装到相同的地方？**Hello**程序是否知道还有其他程序在同时运行？是否直接访问硬件资源？

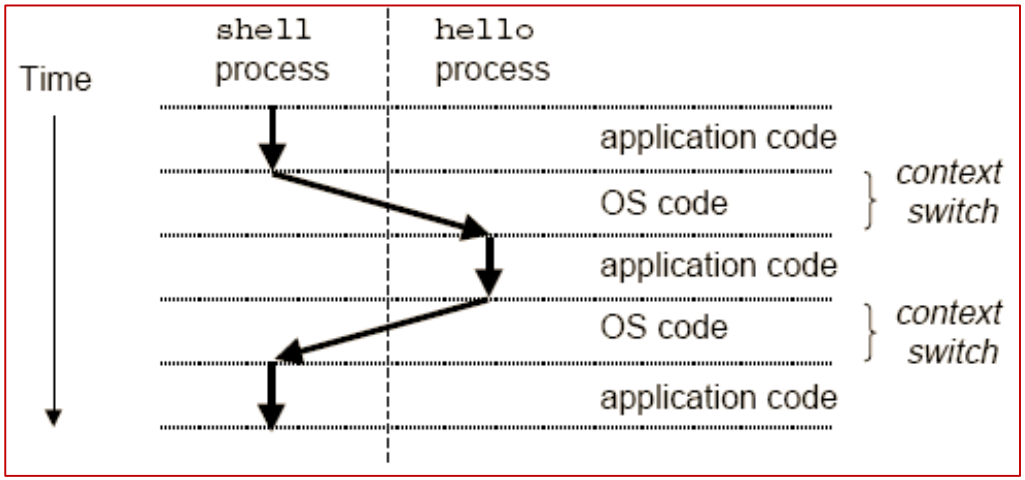
概述

- 操作系统在程序执行过程中的作用
  - Hello执行过程中，其本身没有直接访问键盘、显示器、磁盘和主存储器这些硬件资源，而是依靠操作系统提供的服务来间接访问。
  - 操作系统的两个主要的作用：
    - 硬件资源管理
    - 为用户使用系统提供一个操作接口
  - 操作系统通过三个基本的抽象概念（进程、虚拟存储器、文件）实现硬件资源管理
    - 文件（files）是对I/O设备的抽象表示
    - 虚拟存储器（Virtual Memory）是对主存和磁盘I/O的抽象表示
    - 进程（processes）是处理器、主存和I/O设备的一种抽象表示，实际上是对运行程序的抽象表示



概述

- 进程：操作系统对运行程序的抽象
  - 一个系统上可以同时运行多个进程，而每个进程都认为自己独占系统，实际上，操作系统让处理器交替执行不多进程中的指令；
  - 操作系统的交替执行机制称为“上下文切换（context switching）”
  - 进程的上下文：指进程运行所需的所有状态信息，例如：PC、寄存器堆的当前值、主存的内容、段/页表等
    - 系统中有一套状态单元存放当前运行进程的上下文
  - 上下文切换过程：（任何时刻，系统中只有一个进程正在运行）
    - 把正在运行的进程换下，换一个新进程到处理器执行，上下文切换时，必须保存换下进程的上下文，恢复换上进程的上下文



开始，Shell进程等待命令行输入
输入“Hello”后Shell进行系统调用
OS保存shell上下文，创建并换入hello进程
Hello进程中止时，进行系统调用
OS恢复shell进程上下文，并换入shell进程

由于在一个进程的整个生命周期中，有不同进程在处理器交替运行，所以运行时间很难准确、重复测量。

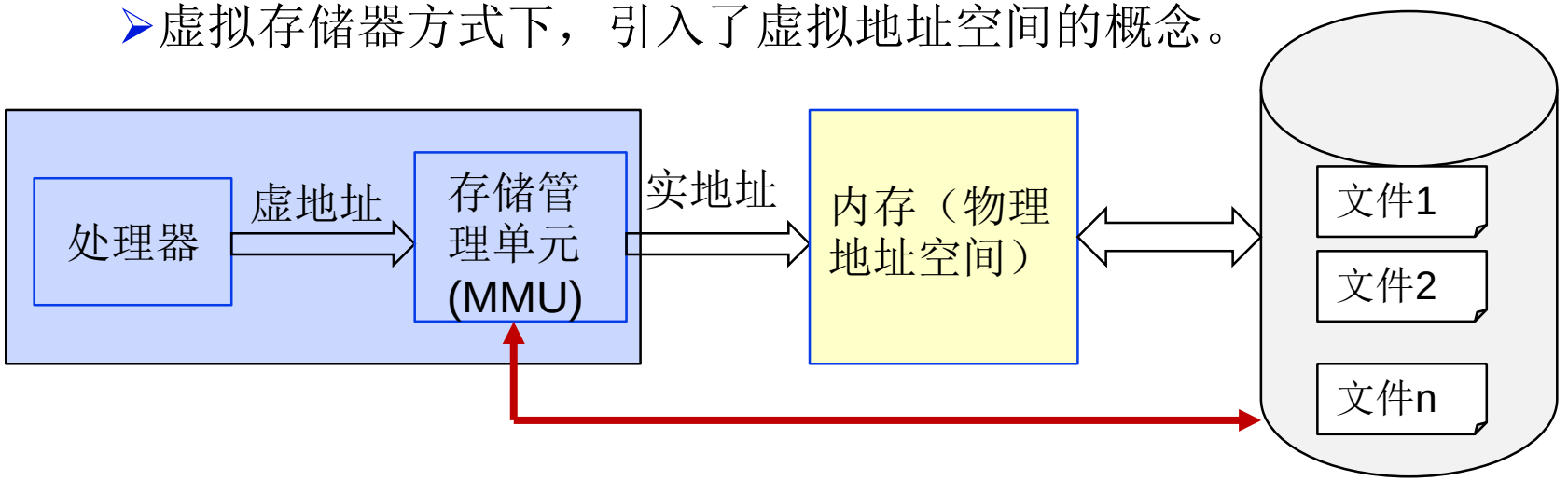
概述

- 虚拟存储技术的动机
  - （早期）采用单道程序运行，系统的主存中包含：
    - 操作系统（常驻监控程序）
    - 正在执行的一个用户程序

    所以无需进行存储管理，即使有也很简单
  - 采用多道程序运行，主存中包括操作系统和若干用户程序（进程）
    - 多道程序（进程）如何有效安全的共享内存？
      - 同时运行的程序对内存需求之和超过计算机实际内存容量；
      - 局部性原理：某个时间点，只需要一部分活跃的程序
    - 如何消除小的主存容量对编程的限制？
      - 编写程序时，不知道程序运行时将和哪些程序共享内存；编程者总是希望把每个程序编译在它自己的地址空间中。
      - 单用户程序大小超过主存容量

概述

- 解决之道
  - 内存管理采用交换机制（硬件和操作系统实现），进程保存在辅存中，进程执行时，只将其活跃部分调入内存（局部性原理）。此时主存可以视为辅存的“高速缓存”。
  - 这样一种把主存当做辅助存储器的高速缓存技术，称为虚拟存储技术，也称为虚拟存储器（virtual memory）。
  - 虚拟存储器能从逻辑上为用户提供一个比物理存贮容量大得多、可寻址的“主存储器”。
  - 虚拟存储器的容量与物理主存大小无关，而受限于计算机的地址结构。
  - 虚拟存储器方式下，引入了虚拟地址空间的概念。



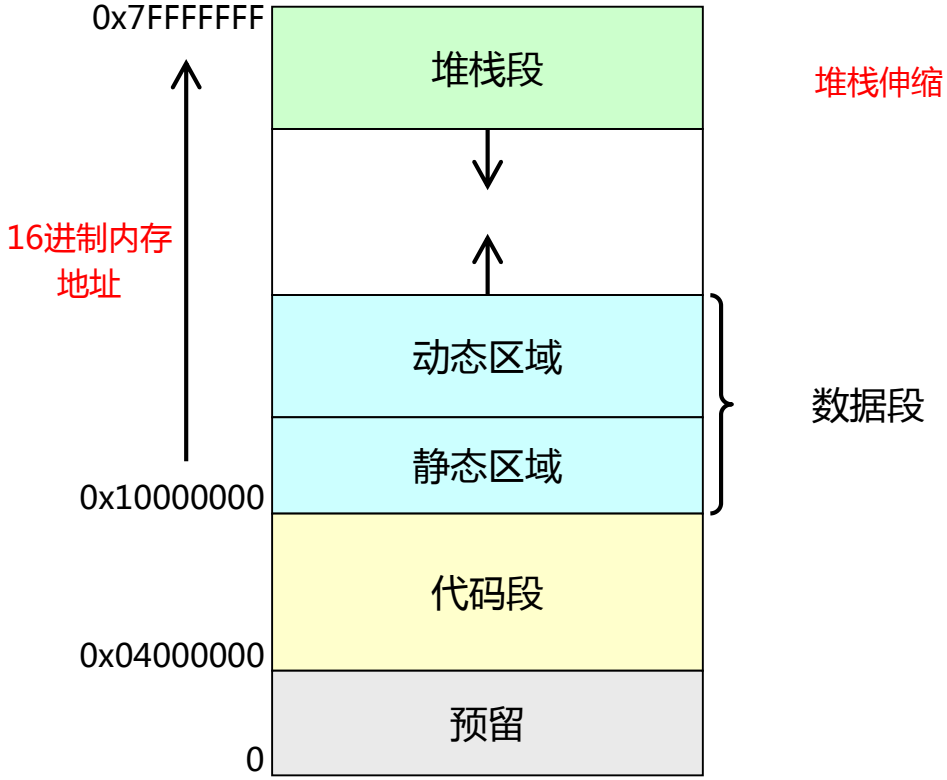
MIPS程序在内存中的存放

- Text: 程序代码段
- Static data: 全局变量

  例如, C语言中的静态变量, 常数数组和串

  $gp 寄存器初始地址±偏移量寻址本段内存
- Dynamic data: 堆

  例如, C中的malloc, Java中的new
- Stack: 栈，自动存储区



概述

虚存空间与物理空间

- 用户程序空间：用户程序给出的指令和数据地址不是真正的主存实际地址，称为虚地址或逻辑地址，其对应的存储空间称为虚存空间或逻辑地址空间。
- 物理内存空间：真正的主存实际地址称为实地址或物理地址，其对应的存储空间称为物理空间或主存空间。

虚拟存储器要解决的问题

- 虚存空间与物理空间之间的数据交换：交换哪些数据？每次交换多少？
- 虚地址与实地址的转换问题：虚地址格式、实地址格式、怎么判断当前访问的虚地址对应的数据是不是在物理空间中，如何把虚地址转换为实地址？如何加速这种判断和转换？
- 缺失处理和替换策略：访问的内容不在物理空间中怎么处理？

概述

❖虚拟存储管理模式：简单分区模式

- 主存分配：
  - 操作系统：固定
  - 用户区：分区
- 简单分区方案：使用长度不等的固定长分区(fixed-size partition)。
- 当一个进程调入主存时，分配给它一个能容纳它的最小的分区。

例如，对于需196K的进程可分配256K的分区。

- 简单分区方式的缺点：
  - 因为是固定长度的分区，故可能会浪费主存空间。多数情况下，进程对分区大小的需求不可能和提供的分区大小一样。

问题：如何生成物理地址？

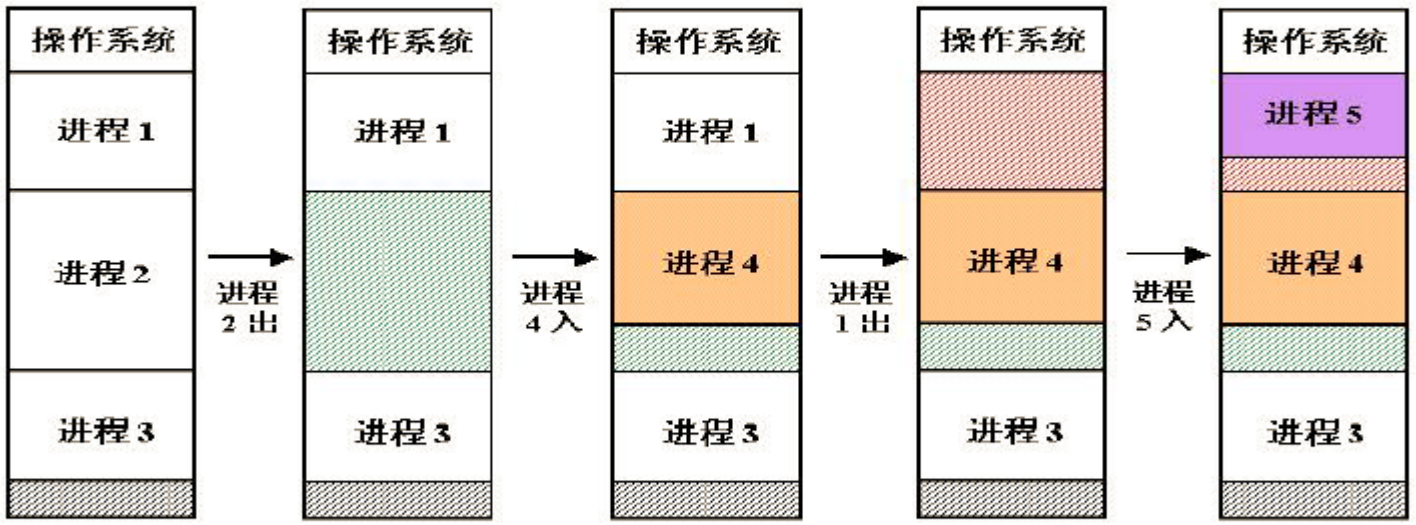
固定分区举例

可以采用更有效的可变长分区的方式！

概述

❖虚拟存储管理模式：可变分区模式

- 分配的分区大小与进程所需大小一样。
- 特点：开始较好，但到最后在存储器中会有许多小空块出现。时间越长，存储器中的碎片就会越来越多，因而存储器的利用率下降。



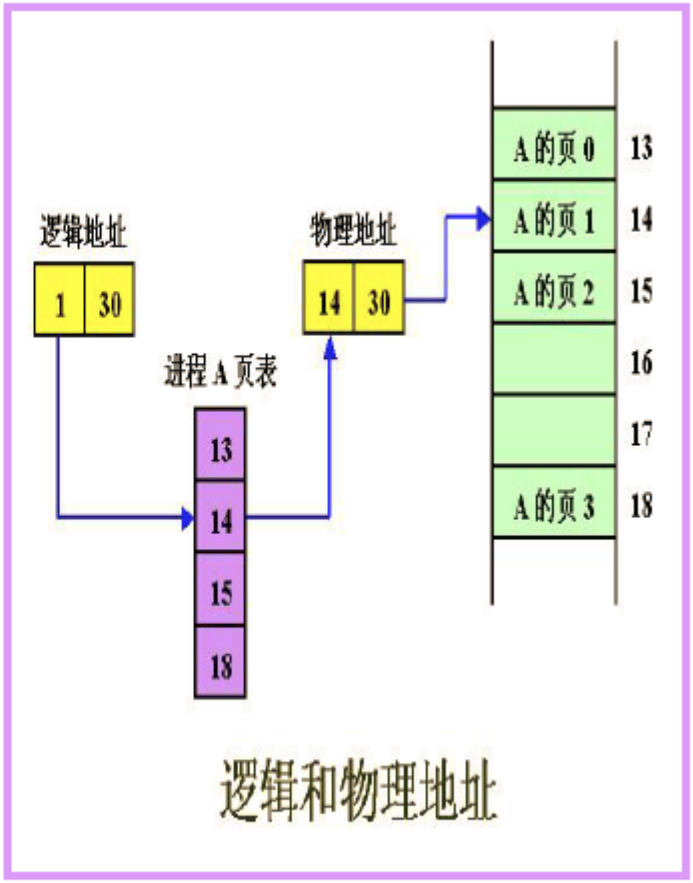
分区的效果

更有效的方式是分页！

概述

❖虚拟存储管理模式：分页模式

- 基本思想：
  - 把内存分成固定长且比较小的存储块，每个进程也被划分成固定长的程序块
  - 程序块（页/page）可装到存储器可用的存储块(页框/page frame)中
  - 无需用连续页框来存放一个进程
  - 操作系统为每个进程生成一个页表
  - 通过页表(page table)实现逻辑地址向物理地址转换（Address Mapping ）
- 逻辑地址（Logical Address）：
  - 程序中的指令所用的地址
- 物理地址（physical或Memory Address）：
  - 存放指令或数据的实际内存地址



逻辑和物理地址

问题：是否需要将一个进程的全部都装入到内存？

根据程序访问的局部性可知：可以仅把当前活跃的页面调入主存，其余留在磁盘上！

浪费的空间最多是最后一页的部分！

概述

❖虚拟存储系统的特征

- ➢程序员在比实际主存空间大得多的逻辑地址空间中编写程序
- ➢程序执行时，把当前需要的程序段和相应的数据块调入主存，其他暂不用的部分存放在磁盘上
- ➢指令执行时，通过硬件（MMU）将逻辑地址（也称虚拟地址或虚地址）转化为物理地址（也称主存地址或实地址）
- ➢在发生程序或数据访问失效时，由操作系统进行主存和磁盘之间的信息交换。

虚拟存储器机制由硬件与操作系统共同协作实现，涉及到操作系统中的许多概念，如进程、进程的上下文切换、存储器分配、虚拟地址空间、缺页处理等。

概述

虚拟存储器小知识

1. 虚拟存储器源出于英国ATLAS计算机的一级存储器概念。这种系统的主存为16千字的磁芯存储器，但中央处理器可用20位逻辑地址对主存寻址。
2. 1970年，美国RCA公司研究成功虚拟存储器系统。
3. IBM公司于1972年在IBM370系统上全面采用了虚拟存储技术。
4. 目前虚拟存储器已成为计算机系统中非常重要的部分。

❖页式虚拟存储器

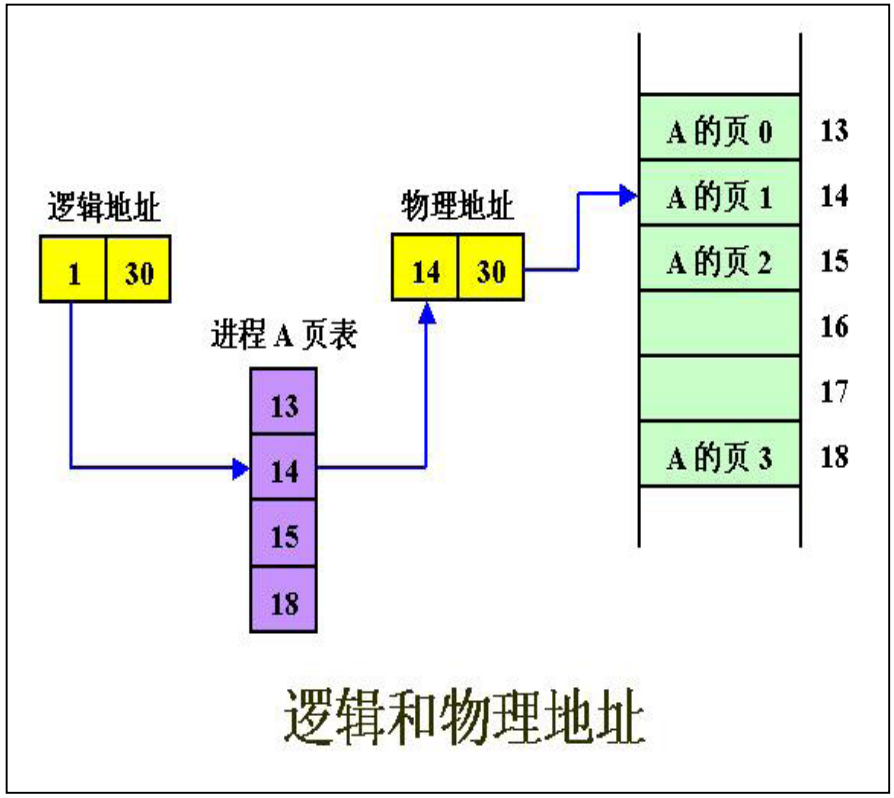
- 主存分成固定长度且比较小的存储块，称为实页（物理的页）；
- 进程也划分成相同长度的程序块，称为虚页（虚拟的页）；
- 主存按页顺序编号，每个独立编址的程序（进程）空间有自己的页号顺序，通过调度辅存中程序各页可离散装入主存不同实页位置。
- CPU执行指令时，首先需将逻辑地址转换为主存的物理地址，地址转换由CPU中的MMU实现。
- 页式调度：按页交换
  - 优点：页内零头小，页表对程序员来说是透明的，地址变换快，调入操作简单；
  - 缺点：各页不是程序的独立模块，不便于实现程序和数据保护。

页式虚拟存储器

❖页式虚拟存储器

- 虚存（逻辑地址空间）和主存（物理地址空间）按固定大小分成若干页，虚存页称为虚页，主存页称为实页。辅存中程序按页调入内存；
- 页表：记录虚页与实页的映射关系，实现虚实地址的转换，页表建立在内存中，操作系统为每道程序建立一个页表。页表用虚页号作为索引，页表项包括虚页对应的实页号和有效位。
- 页表寄存器：保存页表在内存中的首地址。
- 虚地址格式：
- 实地址格式：

虚页号	页内地址
实页号	页内地址



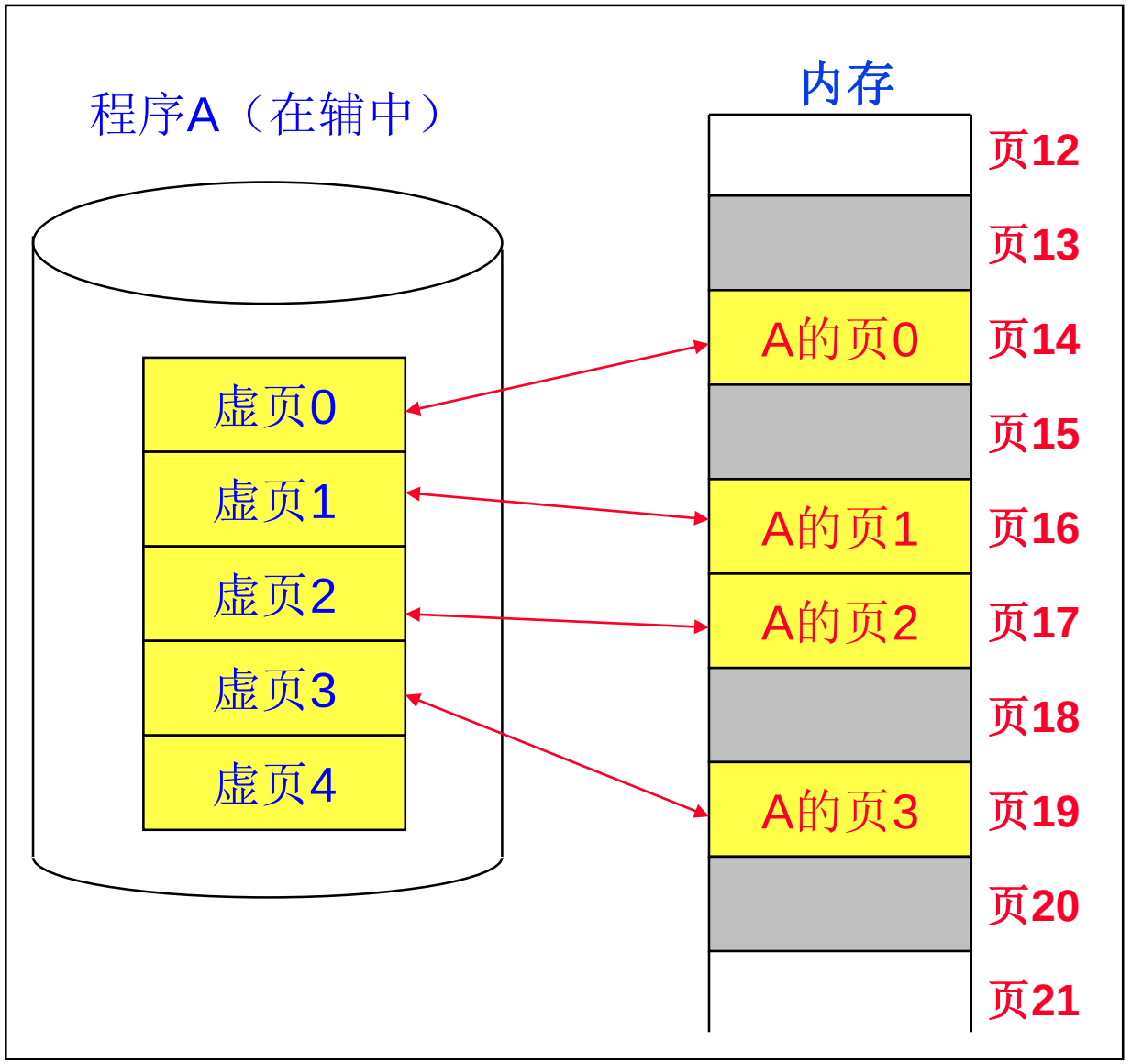
逻辑和物理地址

页式虚拟存储器

页表（Page Table）

虚页	实页号	有效位	修改位
0	14	1	1
1	16	1	0
2	17	1	1
3	19	1	1
4	XX	0	1
	XX	0	0
	XX	0	0

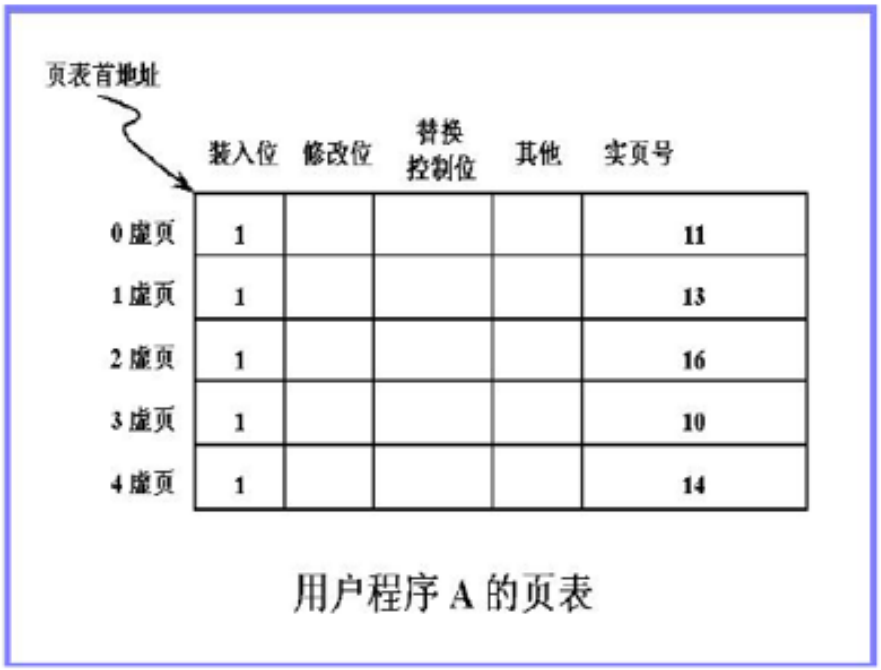
程序A的页表



页式虚拟存储器

❖页表

- ➢每个进程有一个页表，页表项数取决于虚拟地址的结构。
- ➢页表项包括：实页号、装入位、修改位等
- ➢页表在内存中的首地址记录在页表基址寄存器中。



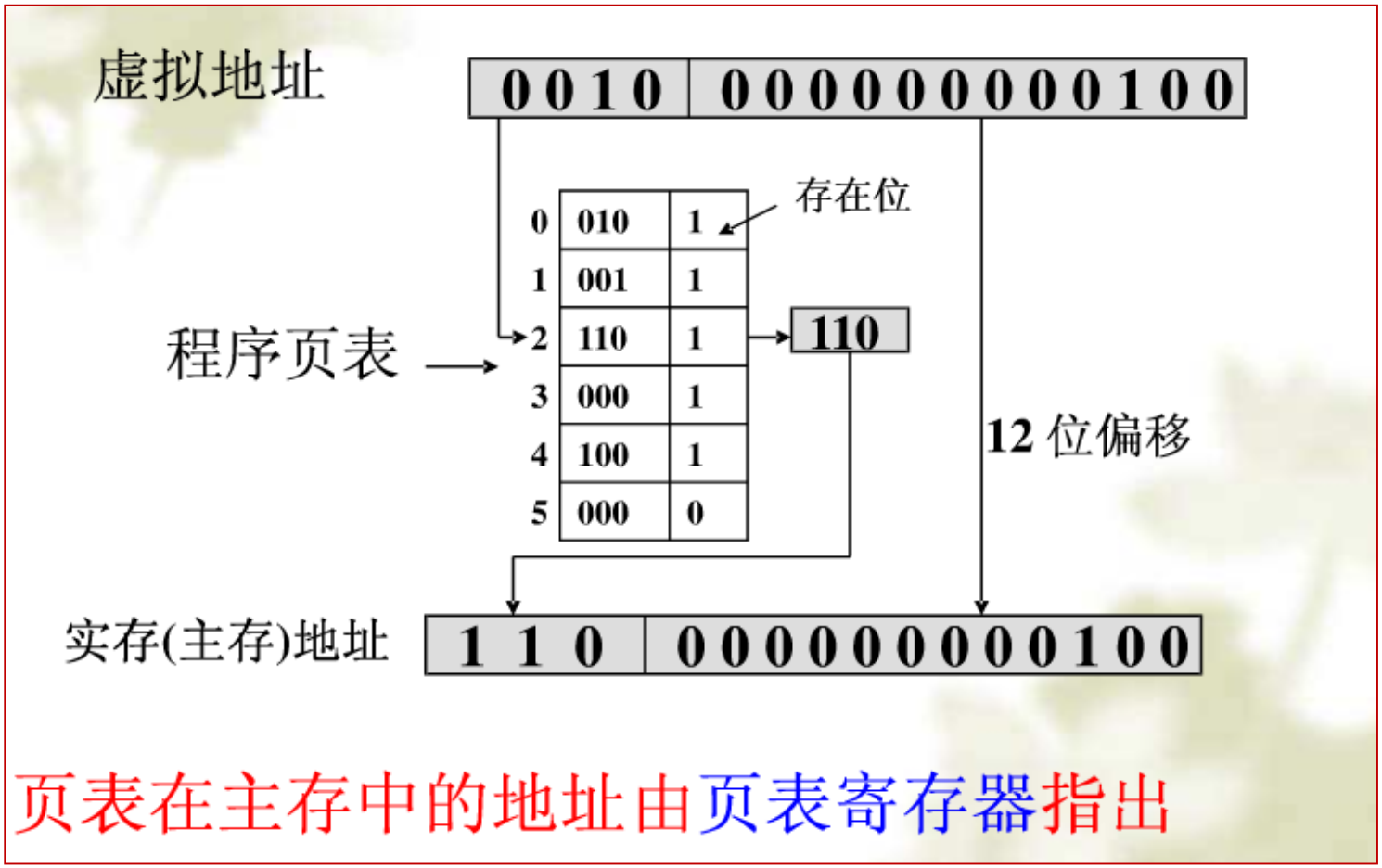
用户程序 A 的页表

❖页表空间问题

- ➢页表可能很大。如VAX系统中，用户程序（进程）虚拟存储空间最大可达2GB，按512B分页，有2^{22}页，页表可以包括2^{22}个页表项。显然，这么大的页表需要占用很大的内存空间。
- ➢多进程运行，对个页表同时都在内存。多个同时运行的进程的页表空间超过内存可分配空间的可能性是存在的。
- ➢采用多级页表机制：将页表分页，当前使用的页的页表项所在页在内存，其余在外存，页表也采用按页交换机制。

页式虚拟存储器

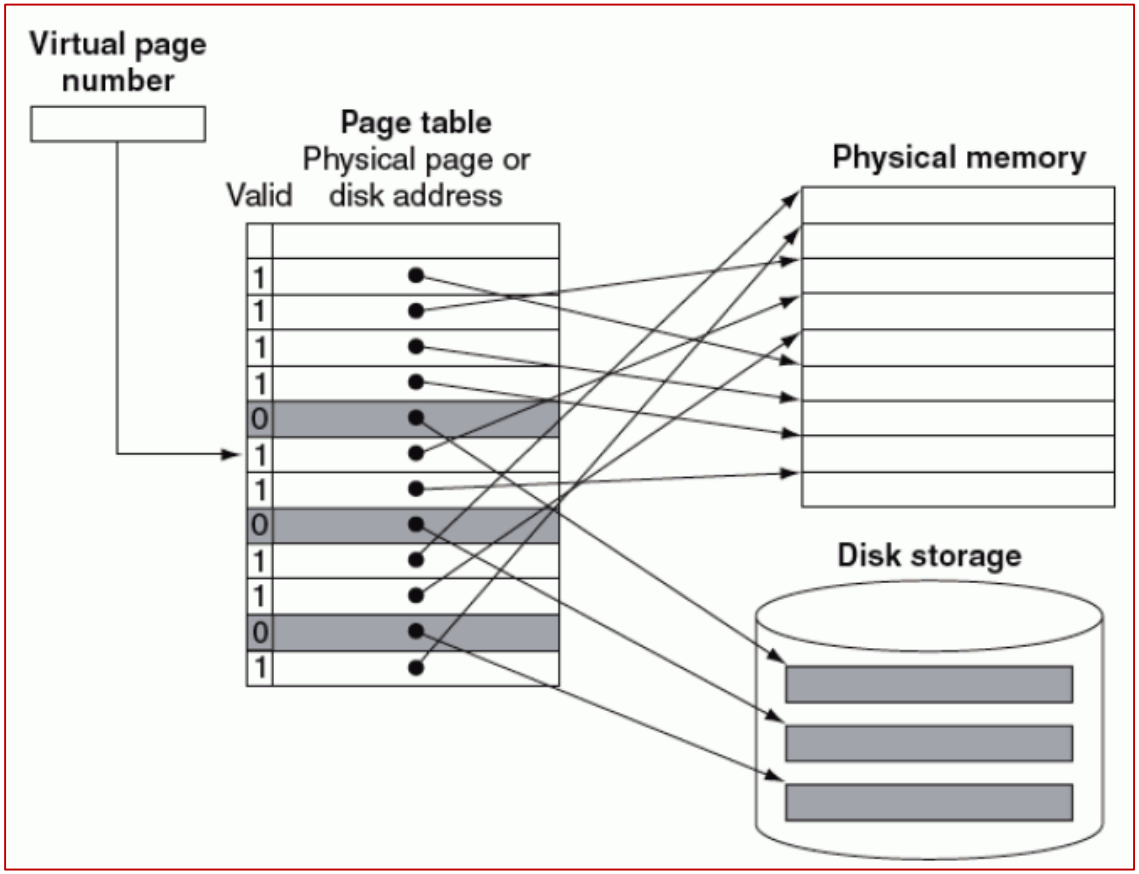
- 虚拟地址到物理地址的转换



页式虚拟存储器

❖一种页表机制

- 当虚页在内存中时（装入位为1时），页表提供虚页到实页的映射（实页号）；
- 当虚页不在内存时（装入位为0时），页表提供保存该虚页的磁盘地址，以便进行磁盘读取；



有些系统采用双表结构，页表只提供虚页到实页的映射，由外页表实现虚页到磁盘地址的映射。

页式虚拟存储器

❖ 举例

某计算机虚拟地址32位，物理内存128MB，页大小4KB。

（1）程序虚拟空间最多可有多少页？

（2）页表项共有多少位？

（3）每个页表占多少内存空间？

❖ 解答

虚地址32位：虚页号（20位）+ 页内偏移（12位）

实地址27为：实页号（15位）+ 页内偏移（12位）

每个程序虚拟空间最多可有：2^{20}个虚页

每个页表项：1位（有效位）+ 15位（实页号）=16位

每个页表所占空间：$2^{20} \times 16 = 16Mb = 2MB$

多道程序运行时，所有程序的页表都在内存中，页表占用内存空间不可忽视，极端情况下，页表有可能消耗所有内存空间。（采用多级页表解决）

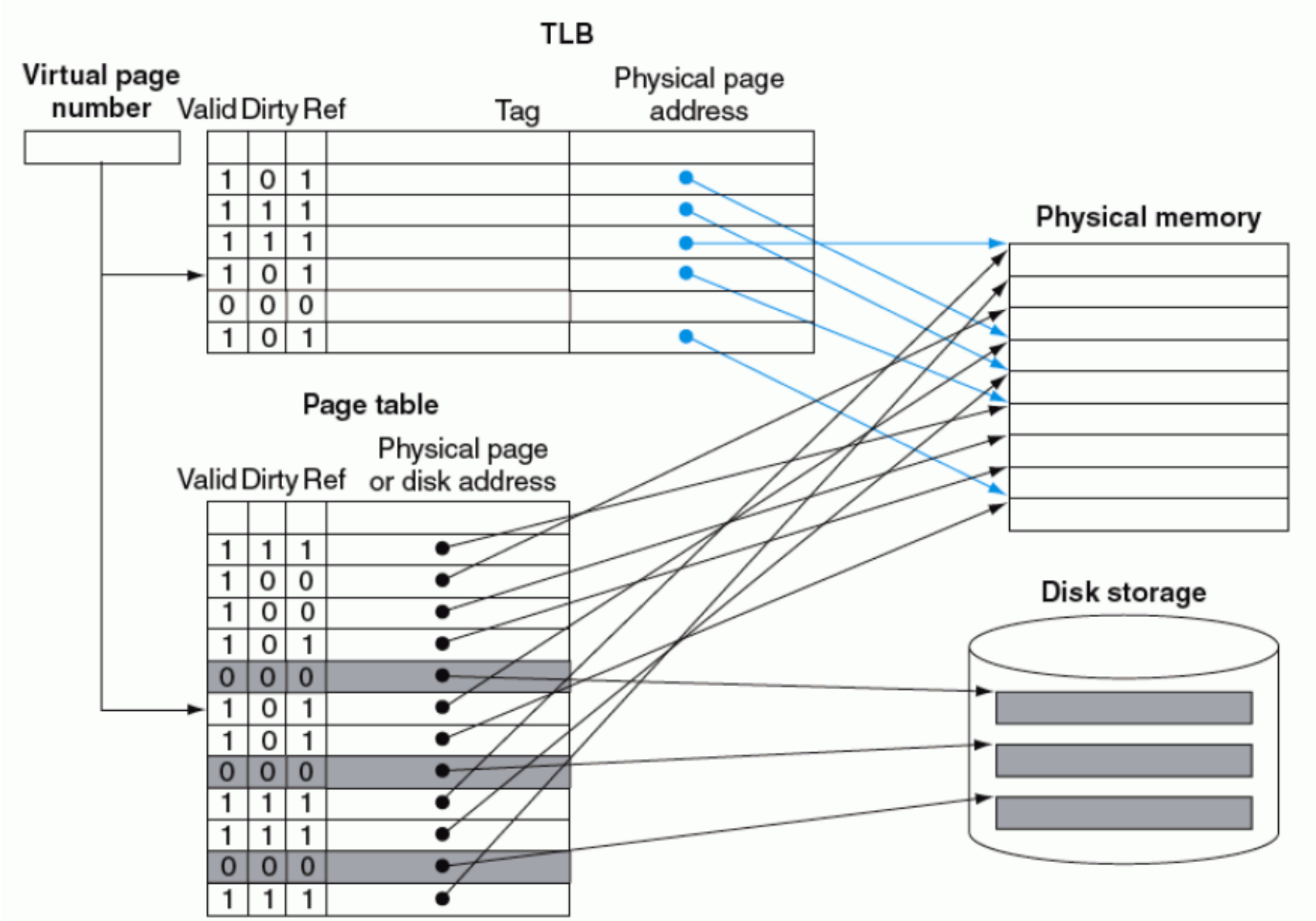
页式虚拟存储器

- **加快地址转换，采用快表TLB（Translation Lookaside Buffer，转换后备缓冲器）**
  - 问题：每次虚拟存储器的访问带来两次存储器访问，一次访问页表，一次访问所需数据（或指令），简单的虚拟存储器速度慢。
  - 解决办法：使用Cache存储部分活跃的页表项，称为TLB（快表），它包含了最近使用的那些页表项。
  - TLB内容（全相联模式）：标记（虚页号）、数据块（实页号）、有效位、修改位。
  - TLB一般采用全相联

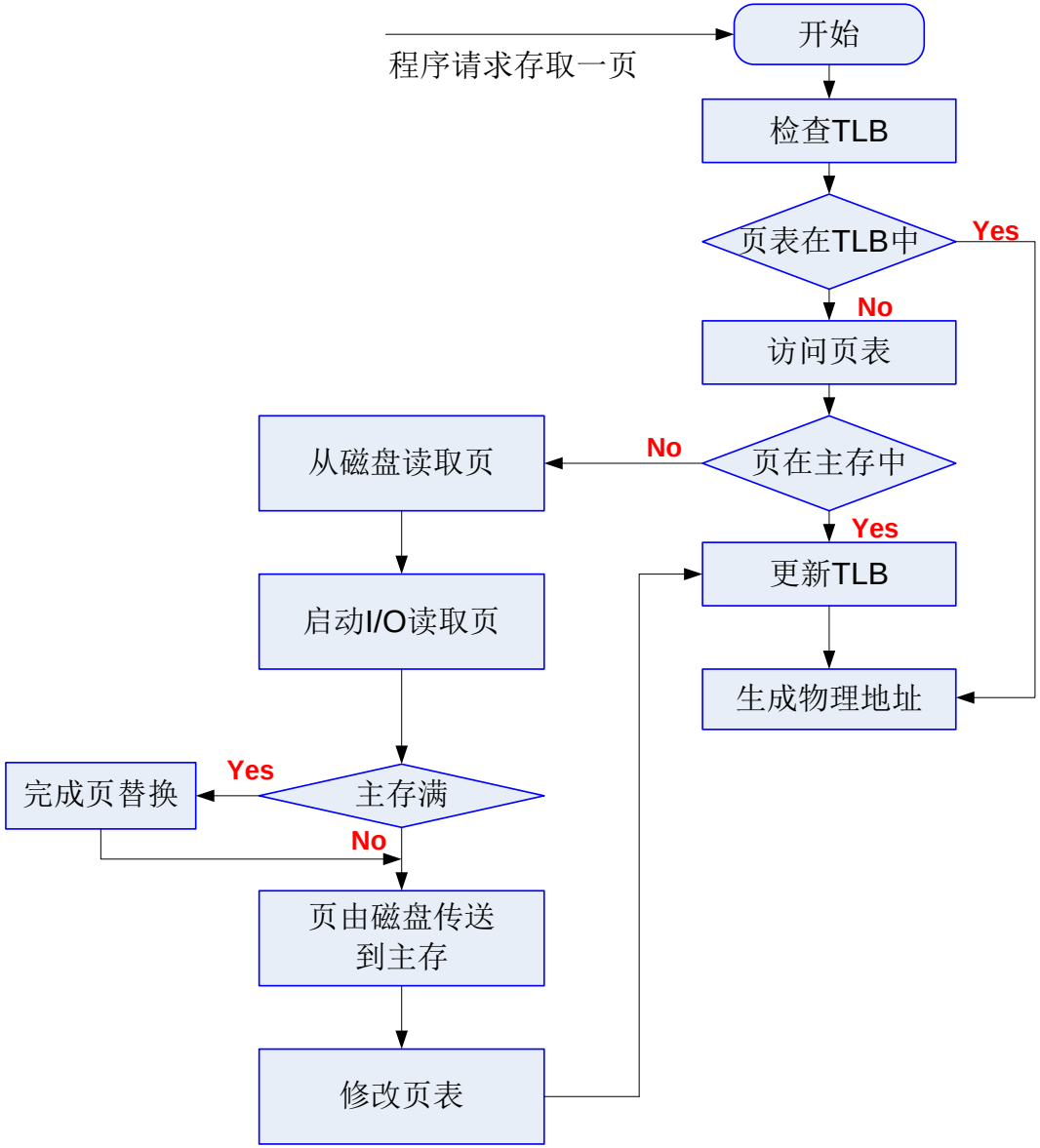
快表（TLB）

有效位	修改位	标记（tag）	数据
		虚页号	实页号
		虚页号	实页号
		虚页号	实页号
		虚页号	实页号

页式虚拟存储器

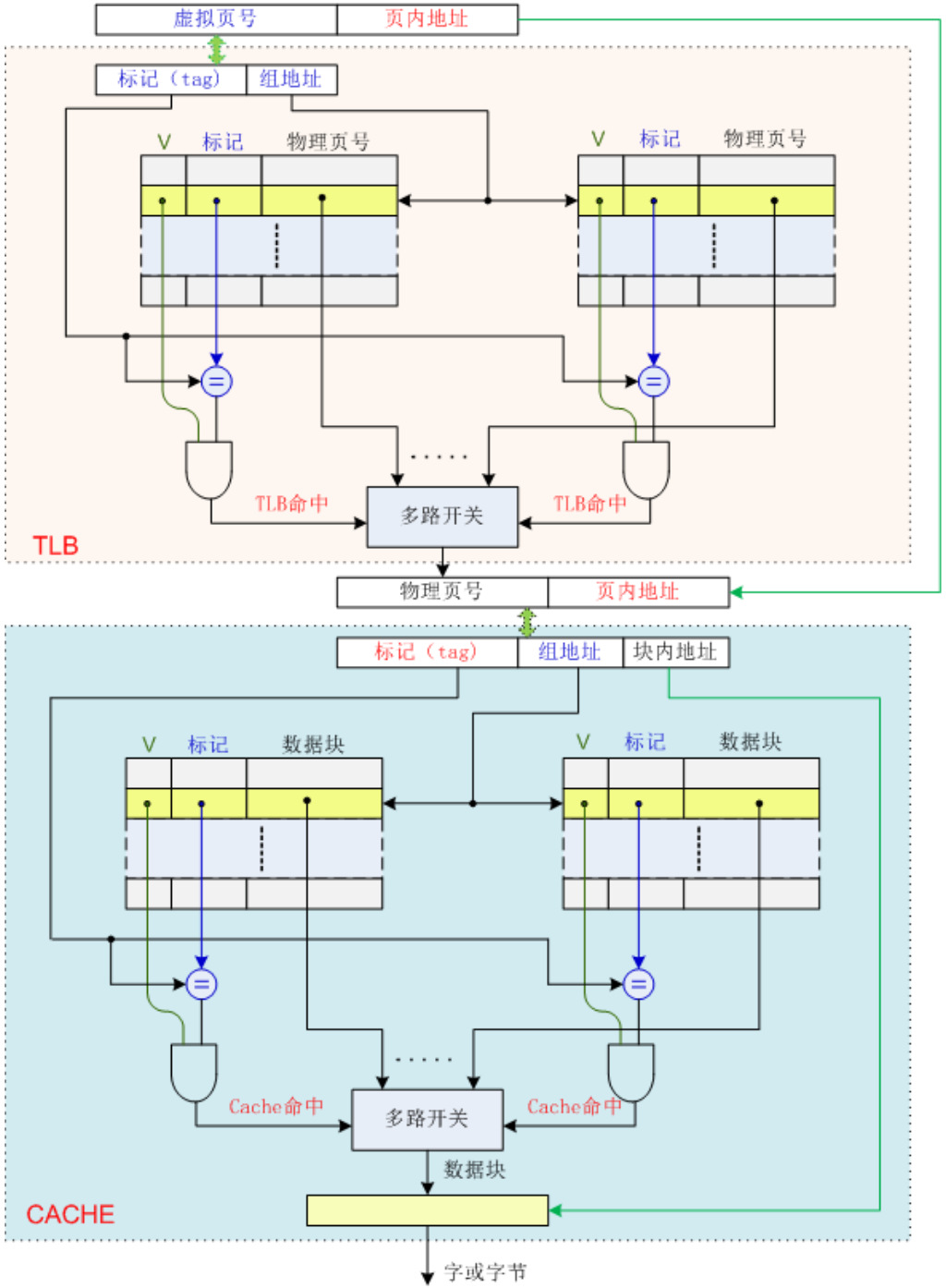

TLB
Virtual page number
Valid Dirty Ref
Tag
Physical page address
1 0 1
1 1 1
1 1 1
1 0 1
0 0 0
1 0 1
Physical memory
Page table
Physical page or disk address
Valid Dirty Ref
1 1 1
1 0 0
1 0 0
1 0 1
0 0 0
1 0 1
1 0 1
0 0 0
1 1 1
1 1 1
0 0 0
1 1 1
Disk storage

页式虚拟存储器

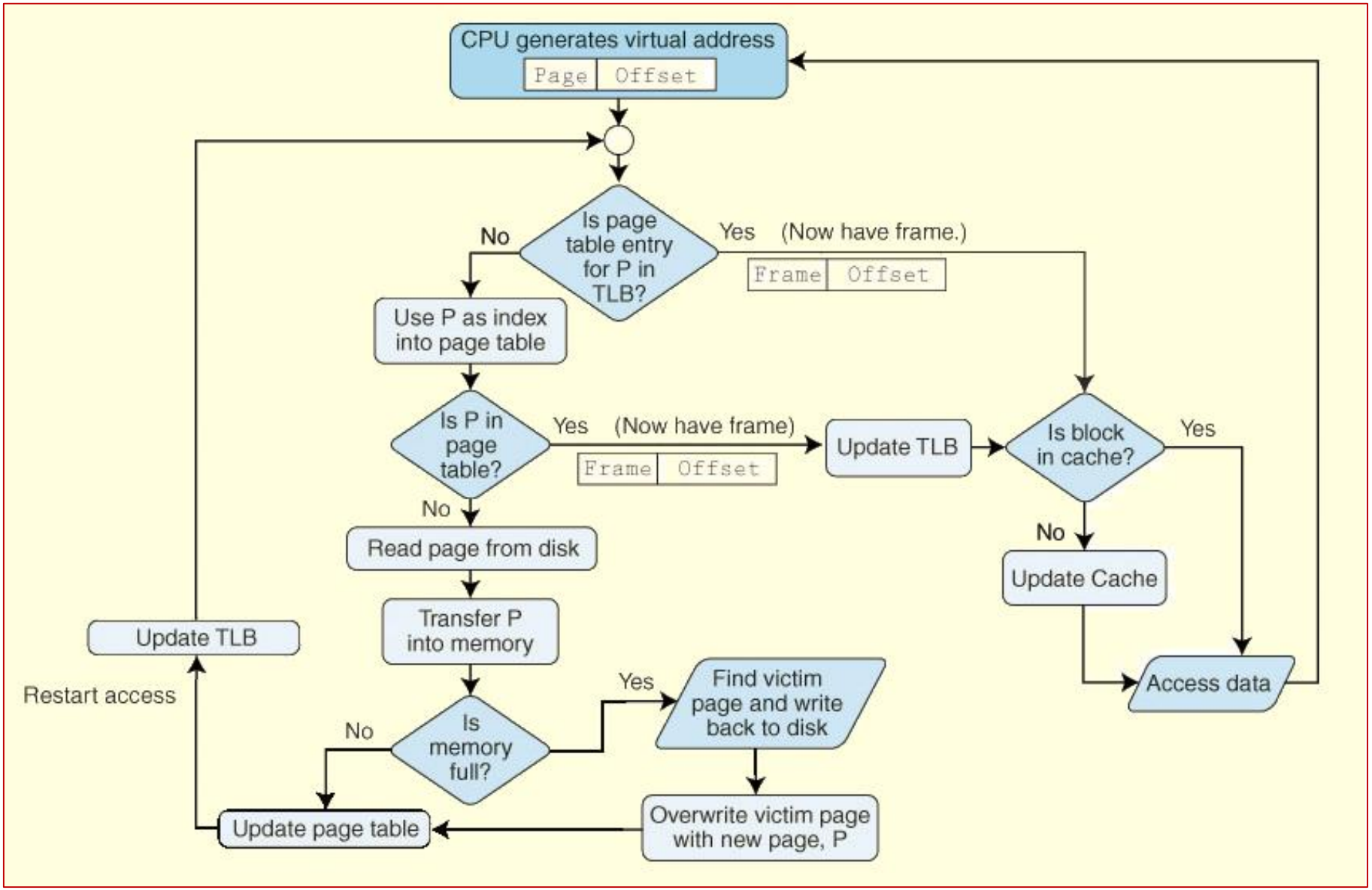

开始
程序请求存取一页
检查TLB
页表在TLB中
Yes
No
访问页表
从磁盘读取页
No
页在主存中
Yes
更新TLB
启动I/O读取页
生成物理地址
完成页替换
Yes
主存满
No
页由磁盘传送
到主存
修改页表

页式虚拟存储器

- CPU通过TLB和Cache访问的全过程
  - TLB和Cache都采用组相联
  - 以虚拟页号为依据访问TLB获取物理页号
  - 以物理地址为依据访问Cache获取最终数据



页式虚拟存储器


CPU generates virtual address
Page
Offset
Is page table entry for P in TLB?
No
Yes
(Now have frame.)
Frame
Offset
Use P as index into page table
Is P in page table?
Yes
(Now have frame)
Frame
Offset
Update TLB
Is block in cache?
Yes
No
Update Cache
Access data
No
Read page from disk
Transfer P into memory
Update TLB
Restart access
Yes
Find victim page and write back to disk
No
Is memory full?
Update page table
Overwrite victim page with new page, P

页式虚拟存储器

- TLB，页表，Cache三种缺失的可能性

TLB	Page table	Cache	Possible? If so, under what circumstance?
hit	hit	miss	可能，TLB命中则页表一定命中，但实际上不会查页表
miss	hit	hit	可能，TLB缺失但页表可能命中，信息在主存，就可能在Cache
miss	hit	miss	可能，TLB缺失但页表可能命中，信息在主存，但可能不在Cache
miss	miss	miss	可能，TLB缺失页表可能缺失，信息不在主存，一定也不在Cache
hit	miss	miss	不可能，页表缺失，说明信息不在主存，TLB中一定没有该页表项
hit	miss	hit	同上
miss	miss	hit	不可能，页表缺失，说明信息不在主存，Cache中一定也没有该信息

最好的情况应该是hit、hit、hit，此时，访问主存几次？ 不需要访问主存！

以上组合中，最好的情况是什么？hit、hit、miss和miss、hit、hit 只需访问主存1次

以上组合中，最坏的情况是什么？ miss、miss、miss 需访问磁盘、并访存至少2次

介于最坏和最好之间的是什么？ miss、hit、miss 不需访问磁盘、但访存至少2次

举例

假定页式虚拟存储系统按字节编址，逻辑地址36位，页大小16KB，物理地址32位，页表中包括有效位和修改位各1位、使用位和存期方式位各2位，且所有虚拟页都在使用中。请问：

（1）每个进程的页表大小为多少？

（2）如果所使用的快表（TLB）总表项数为256项，且采用2路组相联Cache实现，则快表大小至少为多少？

❖ **解答（1）**

页面大小：16KB=2^{14}，页内偏移14位

虚地址36位：虚页号=36-14=22位

实地址32为：实页号=32-14=18位

每个进程最多可有：2^{22}个虚页

每个页表项：1 +1+2 +2+ 18=24位

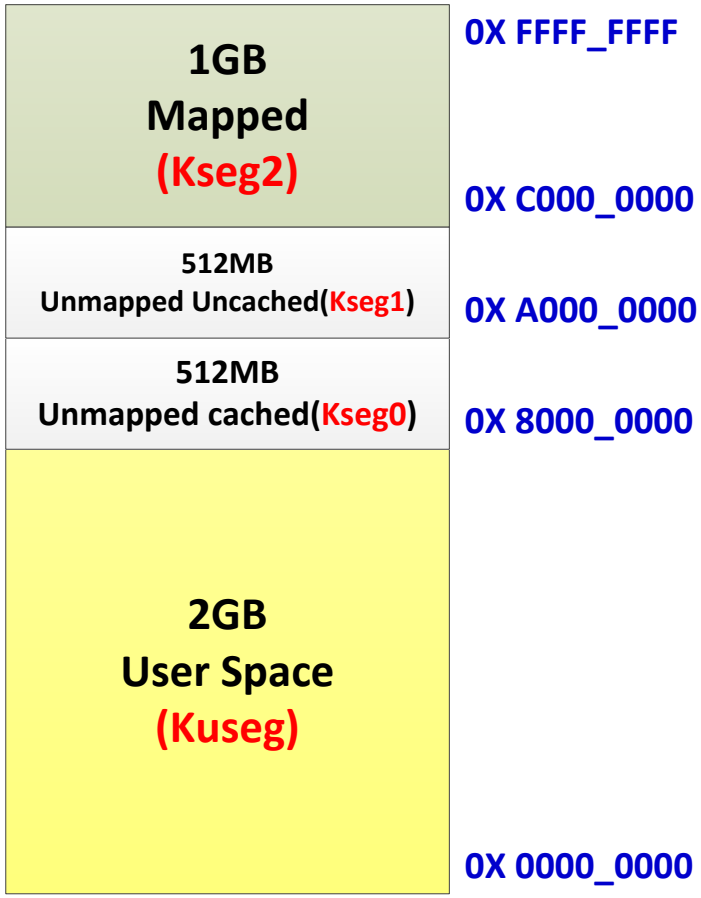
每个页表所占空间：$2^{22}\times 24$=12MB

（2）

- TLB：256个表项，2路组相联，所以共有128组
- 22位虚页号：7位组地址，15位Tag
- TLB每个表项：15+24=39位
- TLB容量：39×256=9984位=1248字节

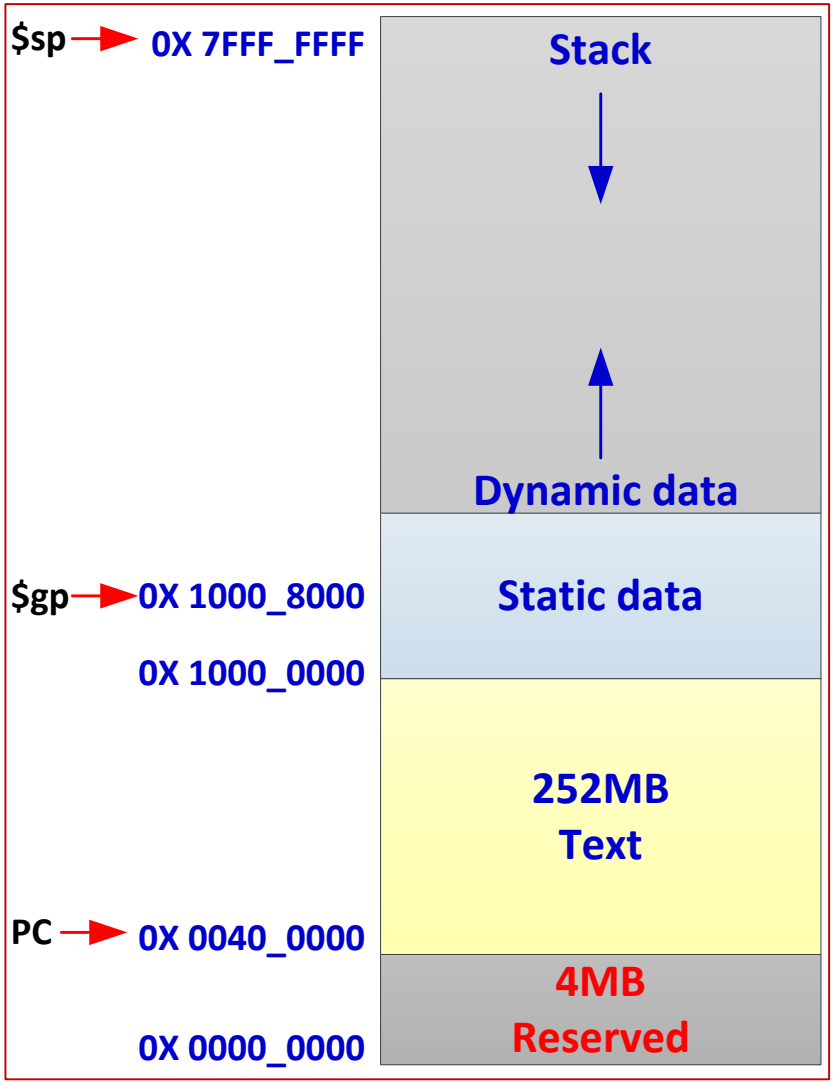
MIPS的存储空间管理

- MIPS CPU运行模式：用户态和核心态
  - Kuseg：0000000 - 7FFF FFFF (2G)，用户模式下可用，即MIPS规范约定用户空间为2G在带有MMU的机器里，这些地址都将由MMU转换。
  - KSeg0：80000000 - 9FFFFFFF(512M)，通过将最高位清零映射到物理地址低端512M(00000000 - 1FFF FFFF)空间。这种映射简单，无需MMU转换（Unmapped）。这段地址的存取都会通过高速缓存(cached)，对于有MMU 的系统，操作系统内核会存放在该区域。
  - KSeg1：A0000000 - BFFFFFFF(512M)，将最高3位清零映射到物理地址低端512M(00000000 - 1FFFFFFF)空间，无需MMU转换（Unmapped），kseg1不使用缓存（Uncached）。kseg1是系统重启时能正常工作的内存映射地址空间，重新启动时的入口向量是BFC00000，这个向量对应的物理地址是1FC00000。因此你可以使用这段地址空间来访问你的初始化程序的ROM。
  - KSeg2：C0000000 - FFFFFFFF (1G)，只能在核心态下使用，并且要经过MMU转换，一般情况下你不需要使用这段地址空间。



MIPS的存储空间管理

- MIPS按如下约定为程序分配空间
  - 堆栈在高地址区，从高到低增长。
  - 过程调用时，生成当前“栈帧”，返回后退回当前栈帧
  - 程序的动态数据（如：C中的malloc申请区域、链表）在堆(heap)中从低向高进行存放和释放（free时）栈区位于堆栈高端，堆区位于堆栈低端，静态数据区上方。
  - 全局指针$gp固定设为0x10008000，其16位偏移量的访问范围为0x10000000 到0x1000FFFF
  - 静态数据区从固定的0x10000000处开始存放
  - 程序代码从固定的0x00400000处开始存放，故PC的初始值为0x00400000。



MIPS每个进程的虚拟（逻辑）地址空间