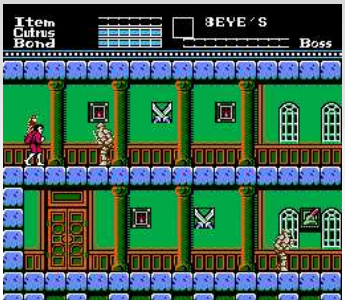
6.1 局部性原理

- 虚拟存储的需求背景
- 局部性原理

增长迅速的存储需求

电脑游戏

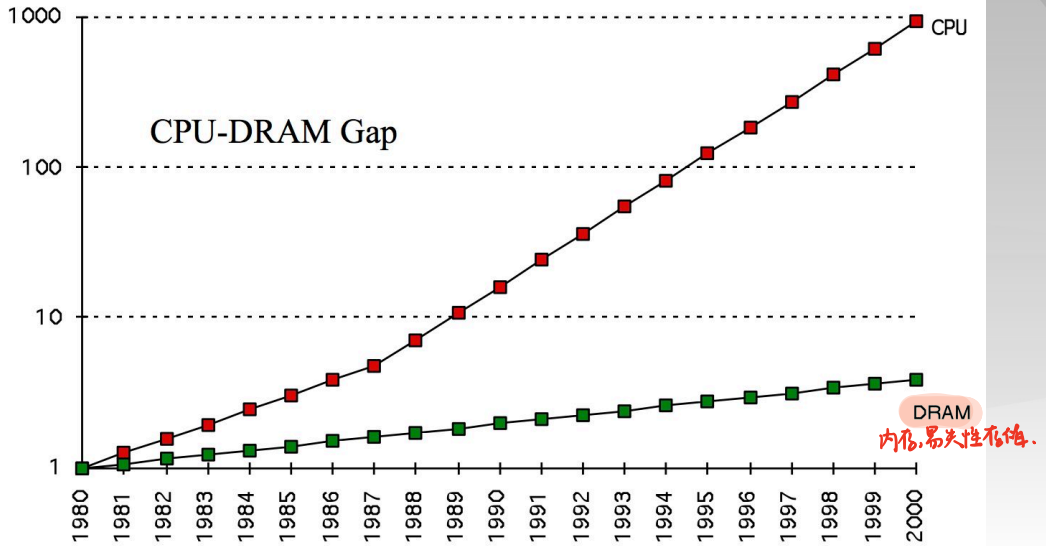
一代	二代	三代	四代	五代	六代	七代	八代
437K	883K	1.9M	6M	6.3M	59M	100M	138M



程序规模的增长速度远远大于存储器容量的增长速度

CPU与内存的速度差异

- Processor vs Memory Performance



1980: no cache in microprocessor;

1995 2-level cache

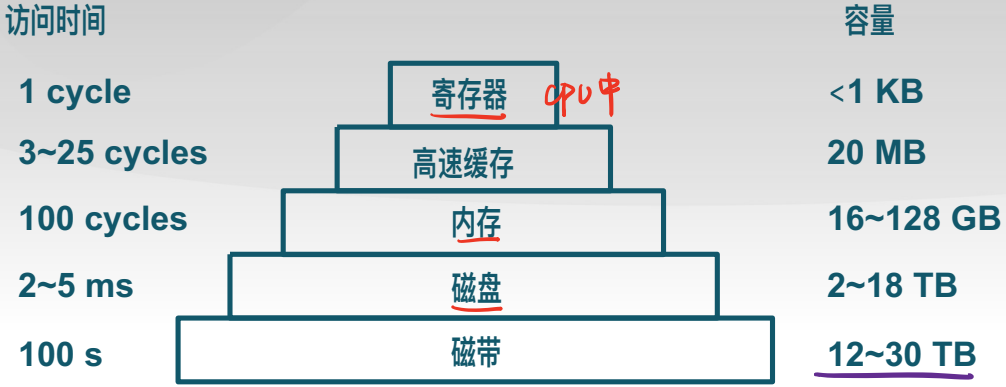
存储层次结构

- 理想中的存储器

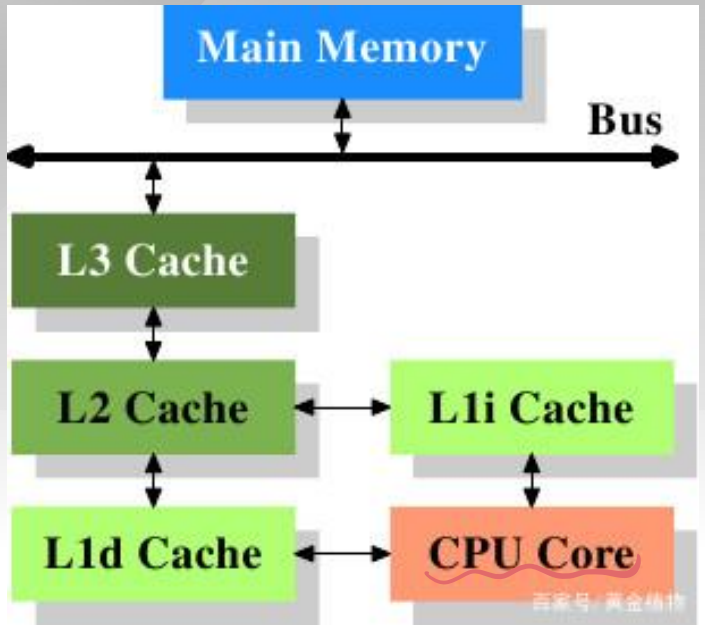
  容量更大、速度更快、价格更便宜的非易失性存储器

- 实际中的存储器

存储器层次结构

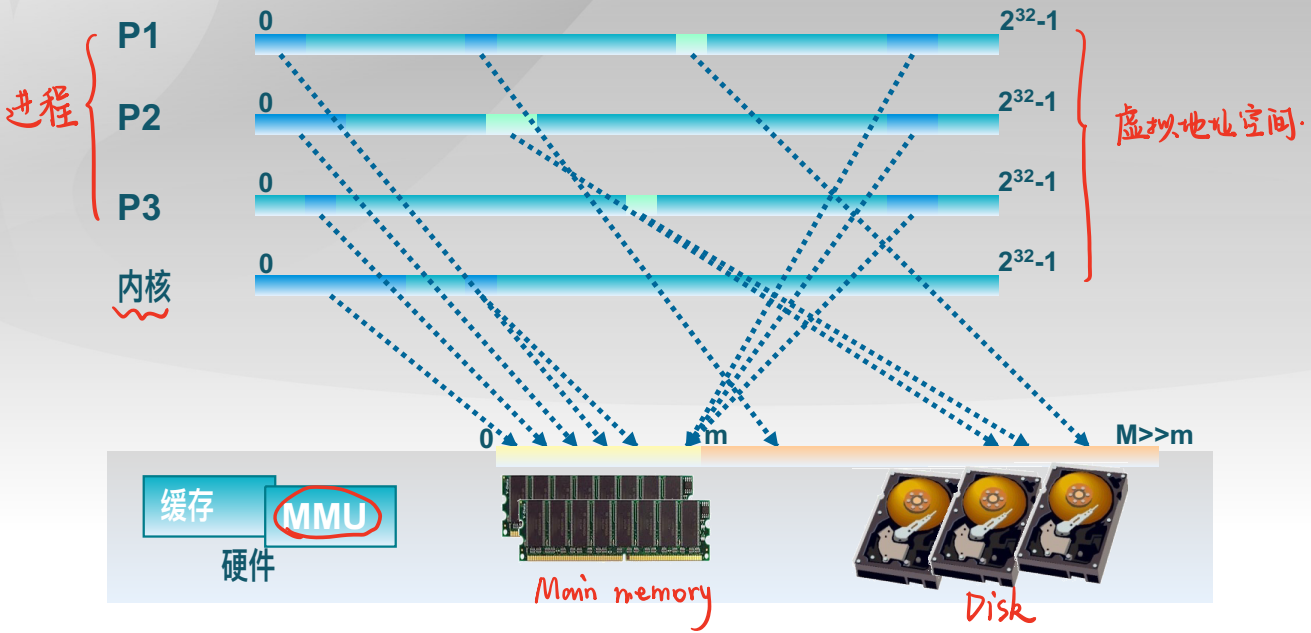


CPU-缓存-主内存


Main Memory
Bus
L3 Cache
L2 Cache
L1i Cache
L1d Cache
CPU Core

操作系统的存储抽象

- 操作系统对存储的抽象：地址空间



虚拟存储需求

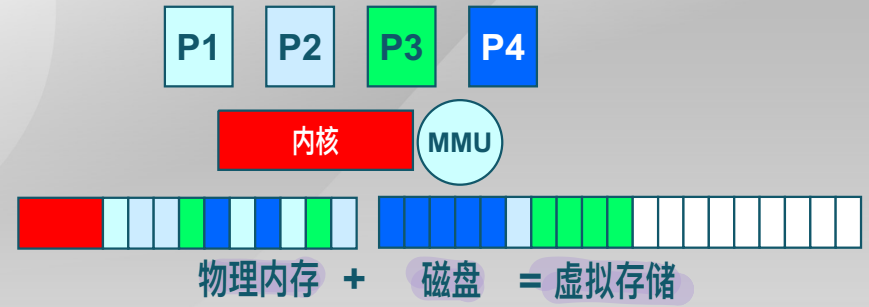
- 计算机系统时常出现内存空间不够用

  - 覆盖（overlay）：在较小的内存中运行较大的程序
    应用程序手动把需要的指令和数据保存在内存中（进程之内）
  - 交换（swapping）
    操作系统自动把暂时不能执行的程序保存到外存中（进程之间）
  - 虚拟存储
    在有限容量的内存中，以页为单位自动装入更多更大的程序

6.1 局部性原理

- 虚拟存储的需求背景
- 局部性原理

虚拟存储技术的目标



- 只把部分程序放到内存中，从而运行比物理内存大的程序
  - 由操作系统自动完成，无需程序员的干涉
- 实现进程在内存与外存之间的交换，从而获得更多的空闲内存空间
  - 在内存和外存之间只交换进程的部分内容

局部性原理（principle of locality）

- 程序在执行过程中的一个较短时期，所执行的指令地址和指令的操作数地址，分别局限于一定区域
  - 时间局部性
    - 一条指令的一次执行和下次执行，一个数据的一次访问和下次访问都集中在一个较短时期内
  - 空间局部性
    - 当前指令和邻近的几条指令，当前访问的数据和邻近的几个数据都集中在一个较小区域内
  - 分支局部性
    - 一条跳转指令的两次执行，很可能跳到相同的内存位置
- 局部性原理的意义
  - 从理论上来说，虚拟存储技术是能够实现的，而且可取得满意的效果

不同程序编写方法的局部性特征

$2^2 \times 2^{10} = 2^{12}$

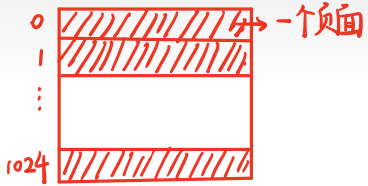
例子：页面大小为4K，分配给每个进程的物理页面数为1。在一个进程中，定义了如下的二维数组int A[1024][1024]，该数组按行存放在内存，每一行放在一个页面中

程序编写方法1：

```
for (j = 0; j < 1024; j++)
for (i = 0; i < 1024; i++)
    A[i][j] = 0;
```

程序编写方法2：

```
for (i=0; i<1024; i++)
for (j=0; j<1024; j++)
    A[i][j] = 0;
```



不同程序编写方法的局部性特征

0	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$ ……	$a_{0,1023}$
1	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$ ……	$a_{1,1023}$
	……			
	……			
1023	$a_{1023,0}$	$a_{1023,1}$ ……		$a_{1023,1023}$

访问页面的序列为：

解法1：

0, 1, 2, ……1023, 0, 1, ……, 共1024组

共发生了1024×1024次缺页中断

解法2：

0, 0, …… 1, 1, ……, 2, 2, ……, 3, 3, ……

共发生了1024次缺页中断

6.2 覆盖和交换

- 覆盖
- 交换

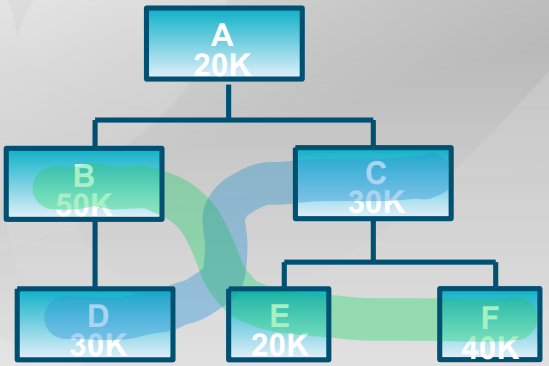
覆盖(Overlay)技术：进程中

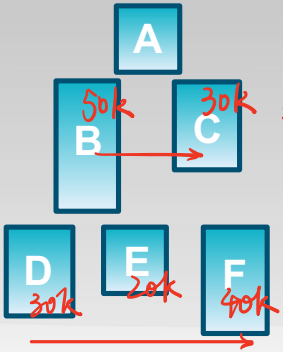
- 目标
  - 在较小的可用内存中运行较大的程序
- 方法

  依据程序逻辑结构，将程序划分为若干功能相对独立的模块；将不会同时执行的模块共享同一块内存区域
  - 必要部分（常用功能）的代码和数据常驻内存
  - 可选部分（不常用功能）放在其他程序模块中,只在需要用到时装入内存
  - 不存在调用关系的模块可相互覆盖，共用同一块内存区域

覆盖技术示例

程序调用结构：190K





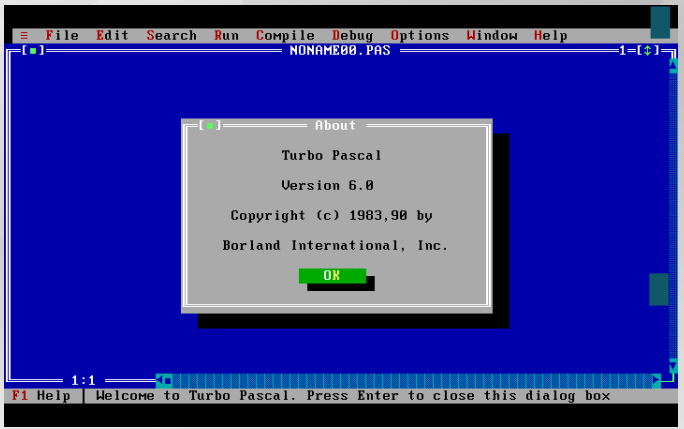
内存总共：110K

内存更小了！

另一种调用方法：(100K)

- A占一个分区：20K
- B、E和F共用一个分区：50K
- C和D共用一个分区：30K

覆盖技术的不足



Turbo Pascal的Overlay系统单元
支持程序员控制的覆盖技术

增加编程困难

- 需程序员划分功能模块，手动！
  并确定模块间的覆盖关系
- 增加了编程的复杂度；

增加执行时间

- 从外存装入覆盖模块
- 时间换空间

6.2 覆盖和交换

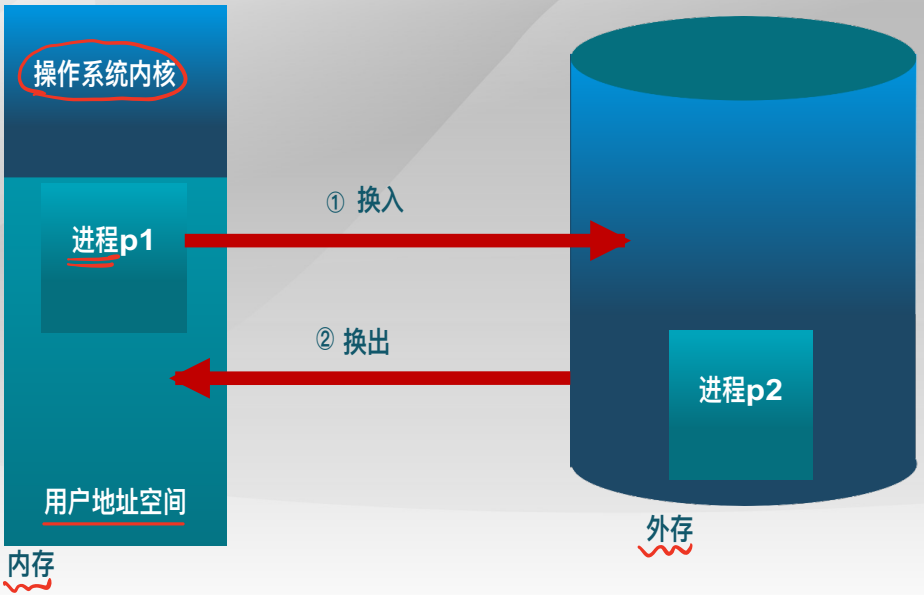
- 覆盖
- 交换

交换(对换, Swap)技术

一个进程在内存空间是够的,但多个进程不够⇒交换;
一个进程就已不够⇒覆盖

- 目标
  - 增加正在运行或需要运行的程序的内存
- 实现方法
  - 可将暂时不能运行的程序放到外存
  - 换入换出的基本单位
    - 整个进程的地址空间
  - 换出（swap out）
    - 把一个进程的整个地址空间保存到外存
  - 换入（swap in）
    - 将外存中某进程的地址空间读入到内存

交换技术



（本图摘自Silberschatz, Galvin and Gagne：“Operating System Concepts”）

交换技术面临的问题

- 交换时机：何时需要发生交换?
  - 只当内存空间不够或有不够的可能时换出
- 交换区大小(外存)
  - 存放所有用户进程的所有内存映像的拷贝 把所有暂停的进程都放外存.
- 程序换入时的重定位：换出后再换入时要放 在原处吗?
  - 采用动态地址映射的方法

覆盖与交换的比较

- 覆盖
  - 只能发生在没有调用关系的模块间 (进程内部)
  - 程序员须给出模块间的逻辑覆盖结构 ⇒ 麻烦!
  - 发生在运行程序(进程)的内部模块间
- 交换
  - 以进程为单位
  - 不需要模块间的逻辑覆盖结构 → OS完成
  - 发生在内存进程间

虚拟页式存储中的外存管理

- 在何处保存未被映射的页?
  - 应能方便地找到在外存中的页面内容
  - 对换空间（磁盘或者文件）
    - 采用特殊格式存储未被映射的页面
- 虚拟页式存储中的外存选择
  - 代码段：可执行二进制文件
  - 动态加载的共享库程序段：动态调用的库文件
  - 其它段：交换空间

虚拟页式存储管理的性能

- 有效存储访问时间（effective memory access time EAT）
  - EAT = 访存时间 * (1-p)
    + 缺页异常处理时间 * 缺页率p
  - 例子
    - 访存时间: 10 ns
    - 磁盘访问时间: 5 ms
    - 缺页率p
    - 页修改概率q
    - EAT = 10(1–p) + 5,000,000p(1+q)

对于被修改过的页，还需要在置换它的时候写入外存，共访问两次外存

10：访存时间