《计组II》

第七章——存储系统(3)

李瑞　副教授

蒋志平　讲师

计算机科学与技术学院

Cache/内存一致性问题

Cache/内存一致性问题

- 读数据是否存在一致性问题?
- 写数据如何保持Cache/Memory一致?
- 两种策略:
  - 写回法(Write Back)
  - 全写法(Write Through)

Cache/内存一致性问题

- **写回法（Write Back）**
  - 当CPU写Cache命中时，只将数据写入Cache，不立即写入主存。只有当被修改的Cache块被替换时才写回主存中。
  - 当CPU写Cache未命中时，则写修改是将相应主存块调入Cache之后，在Cache中进行。对主存的修改仍留待该块替换出去时进行。
  - 用修改位(Dirty Bit)，追踪是否被CPU修改过。

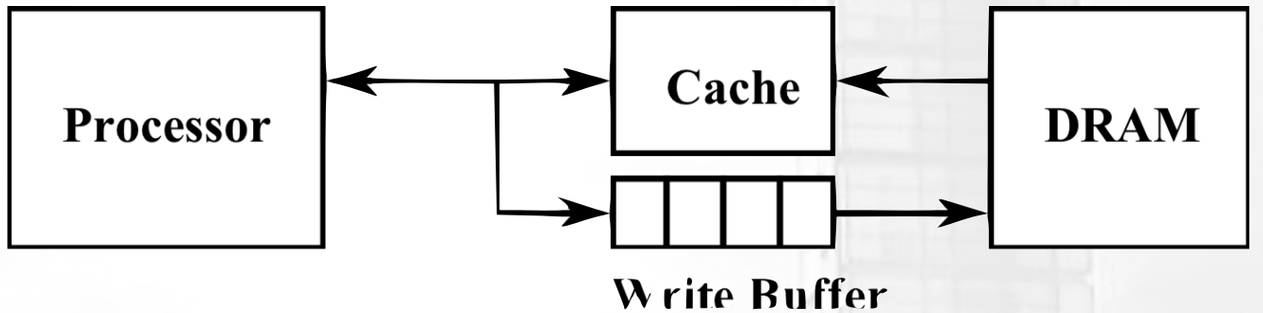
Cache/内存一致性问题

- **全写法（Write Through）**
  - 当Cache命中时，在将数据写入修改Cache的同时写入修改主存，较好地保证了主存与Cache内容的一致性。
  - 当Cache未命中，直接向主存进行写入。这时可以采用两种方式进行处理：

- What a simple solution ?!

Cache/内存一致性问题

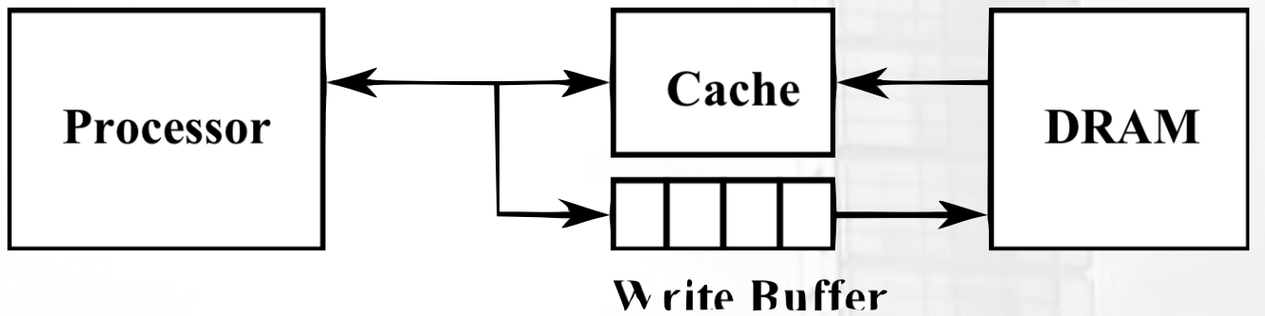
- **全写法（Write Through)**
  - Cache 和内存速度不一致如何解决?
    - 速度不够Buffer凑!



- 但貌似还有问题….
  - 谁把Buffer数据搬到内存?
    - 加个控制器让内存主动取回来…

Cache/内存一致性问题

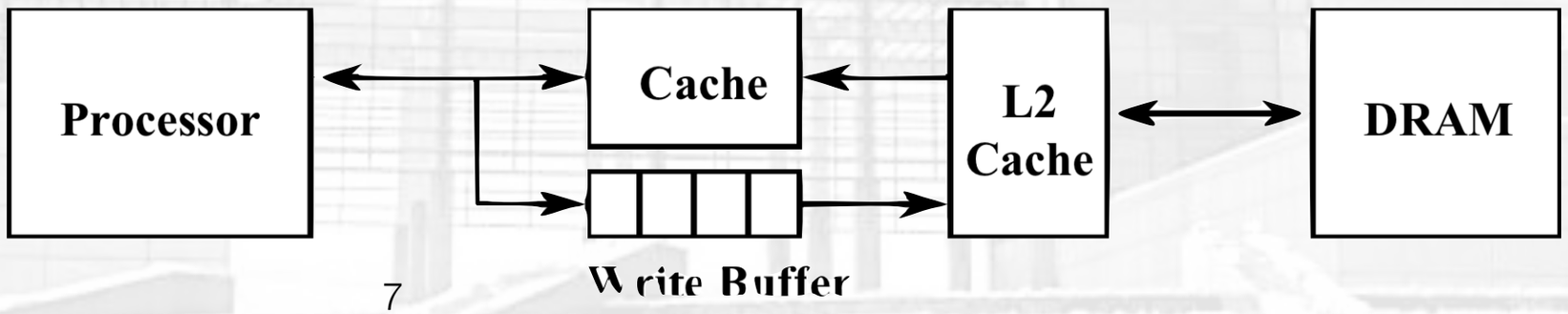
- 但貌似还有问题….
  - 谁把Buffer数据搬到内存?
    - 加个控制器让内存主动取回来…



- 但貌似还有！问题….
  - Buffer会不会爆掉?
    - 加2级Cache!

竞猜：L2 Cache的写策略是?

答：回写法(Write Back)!



7.3.3 主存与Cache内容的一致性问题

采用Cache的访存操作

Cache类型	操作	访存操作	访存时间
写直达Cache	读命中	只读Cache	T_C
	写命中	写Cache，同时写内存	T_C(隐藏T_M)
	读不命中	调入Cache块，再读Cache	$T_B + T_C$
	写不命中	只写内存	T_M
写回Cache	读命中	只读Cache	T_C
	写命中	只写Cache	T_C
	读不命中	调入Cache块，再读Cache	$T_B + T_C$
	写不命中	调入Cache块，再写Cache	$T_B + T_C$

Cache性能分析

7.3.4 Cache性能分析

1. 加速比

■ Cache-主存系统的平均访问周期T：

$$T = H \times TC + (1 - H) \times (TB + TC)$$

- Cache的访问周期为TC，主存的访问周期为TM，数据块调入Cache的块传输时间为TB，Cache的命中率为H。

■ Cache-主存系统的加速比SP：**SP＝TM/T　　<—(Old duration / new)**

7.3.4 Cache性能分析

2. 成本

■ **C = (C1×S1 + C2×S2)/(S1 + S2)**

主存价格　主存容量　Cache容量　Cache价格

S1>>S2

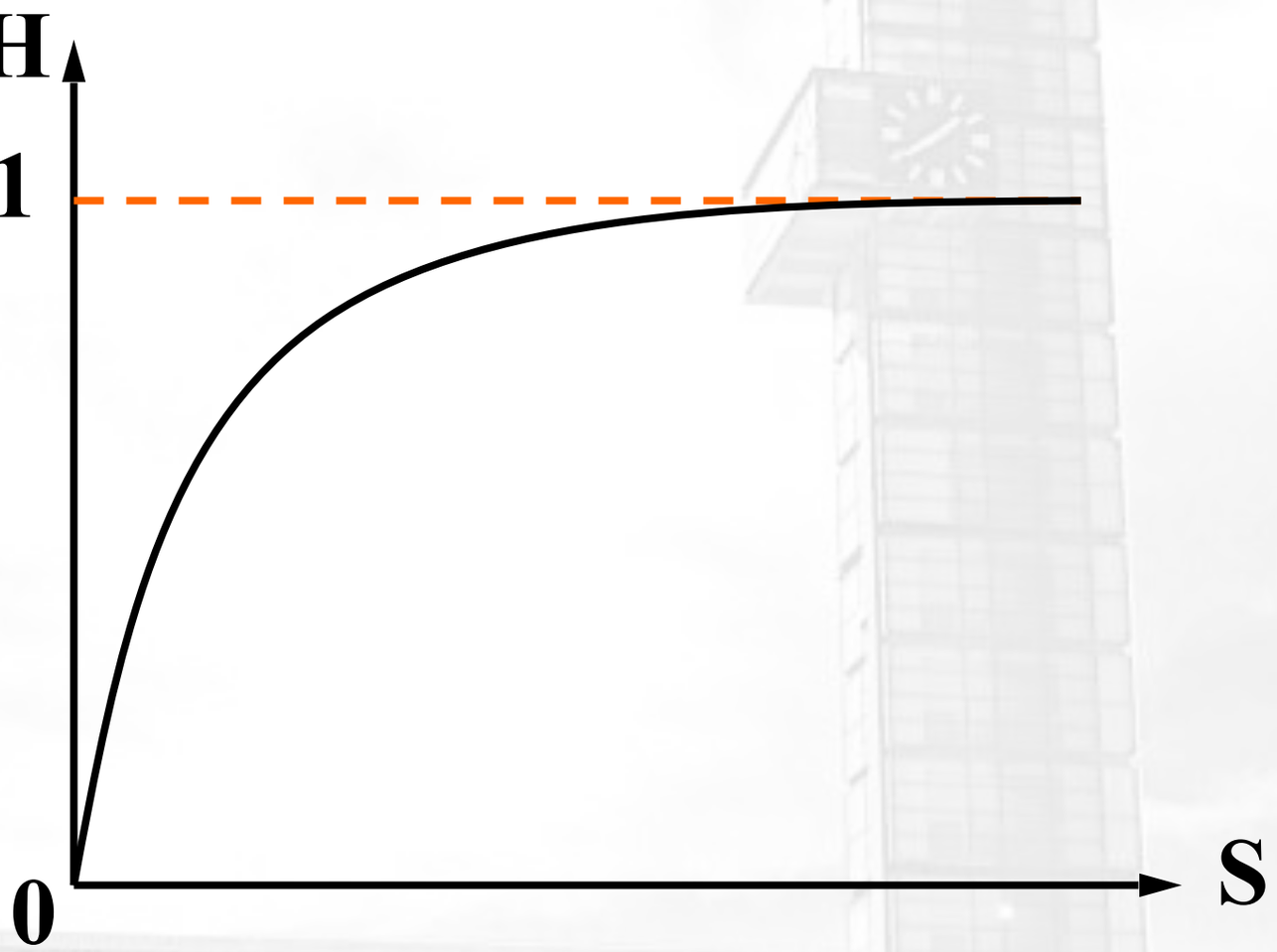
■ 尽管**Cache**的价格比主存高，但是当其容量很小时，存储器的平均价格**C**接近于主存的价格。

■ 由于设置了**Cache**使**CPU**的访存速度接近**Cache**的速度，而使存储器的成本接近于主存的价格。

7.3.4 Cache性能分析

3. 命中率与Cache容量的关系

$$H = 1 - S^{-0.5}$$

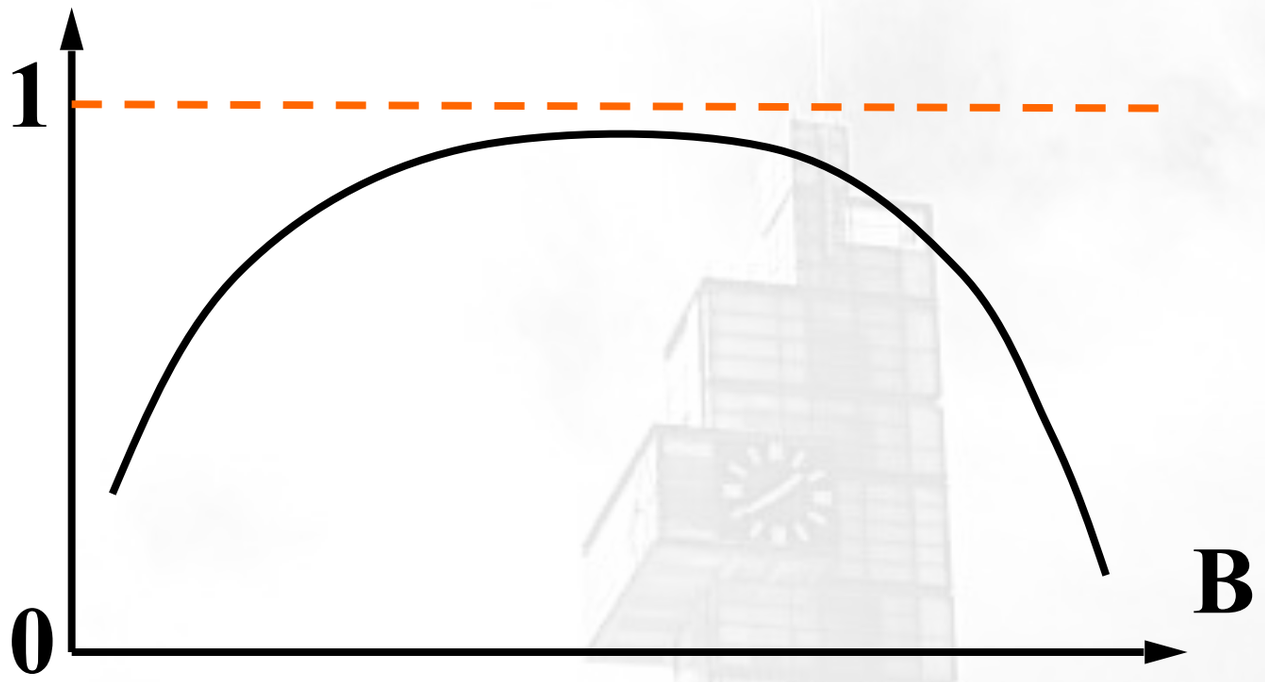


Cache容量S与命中率H的关系

7.3.4 Cache性能分析

4. 命中率与块大小的关系

在组相联组Cache容量一定时，块大小B与命中率的关系?



块大小B与命中率H的关系

假设A_t和A_{t+1}是相邻两次访问主存的地址，

$d = |A_t - A_{t+1}|$。

- 如果$d < B$，随着B的增大，A_t和A_{t+1}在同块的可能性增加，即H随着B的增大而提高。
- 如果$d > B$，A_t和A_{t+1}一定不在同一块内。随着B的增大，Cache块数减少，块替换将更加频繁。H随着B的增大而降低。

7.3.4 Cache性能分析

5. 多级Cache的命中率

- 两级**Cache**的总未命中率(总失效率):

  总失效率 = $(\text{失效率})_{\text{第一级}} \times (\text{失效率})_{\text{第二级}} \ldots\ldots \times (\text{失效率})_{\text{第N级}}$

- 【例】**10000**次访存，第一级**Cache**失效**400**次，第二级**Cache**失效**4**次。
  - $(\text{失效率})_{\text{第一级}} = 400/10000 = 4\%$
  - $(\text{失效率})_{\text{第二级}} = 4/400 = 1\%$
  - 利用两级**Cache**后总的失效率 = **0.04**%。

7.3.4 Cache性能分析

5. 多级Cache的命中率

【例】访问内存需50ns, L1 1ns 10%失效率，L2 5ns 1%失效率，L3 10ns 0.2%失效率，求L1， L1+L2，L1+L2+L3构架下的平均访问时间。

L1 ： $T = 1\ ns + (0.1 \times 50\ ns) = 6\ ns$

L1+2 ： $T = 1\ ns + (0.1 \times [5\ ns + (0.01 \times 50\ ns)]) = 1.55\ ns$

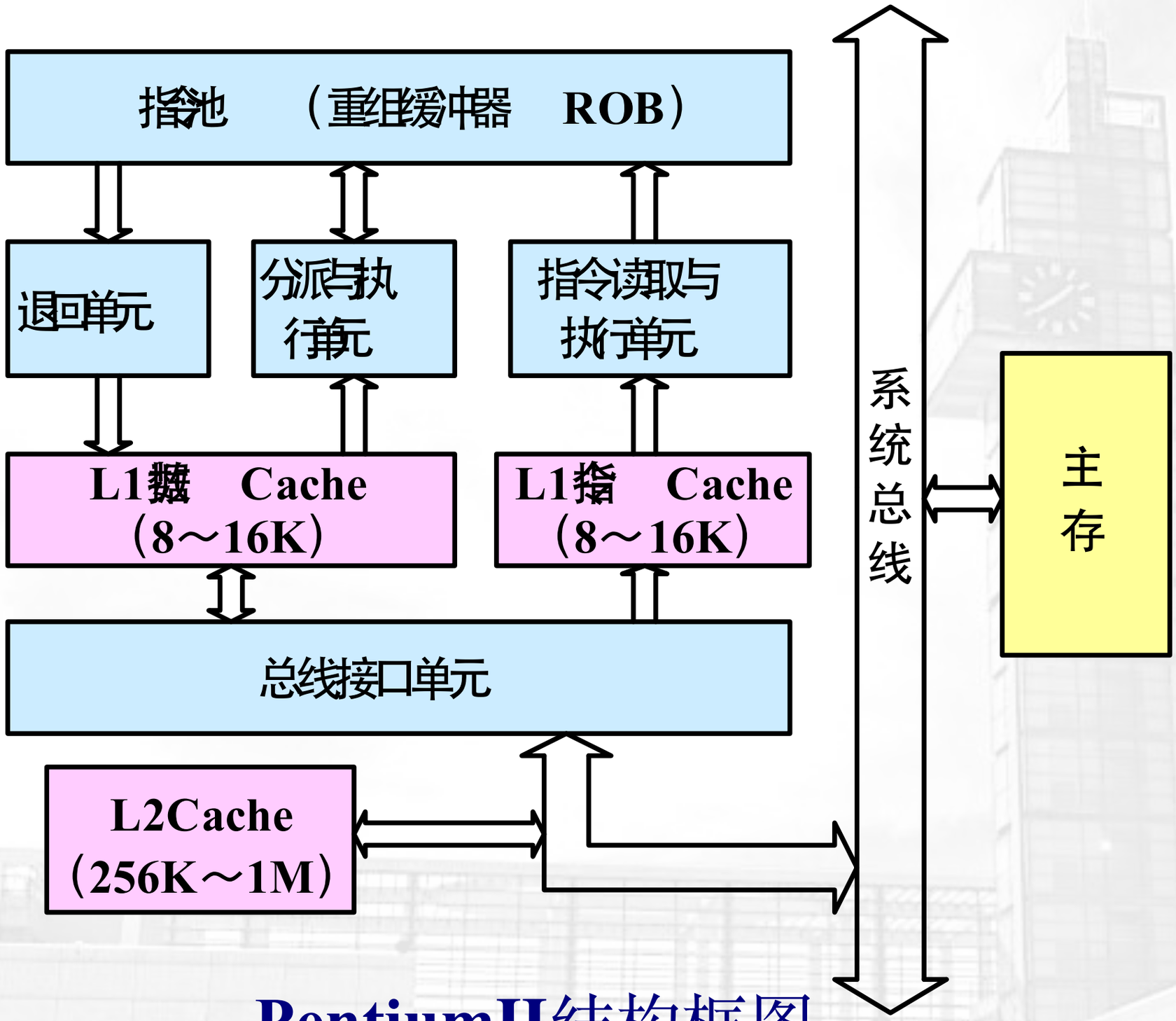
L1+2+3 : $T = 1\ ns + (0.1 \times [5\ ns + (0.01 \times [10\ ns + (0.002 \times 50\ ns)])]) = 1.5001\ ns$

7.3.5 Pentium的Cache

Pentium II 处理器

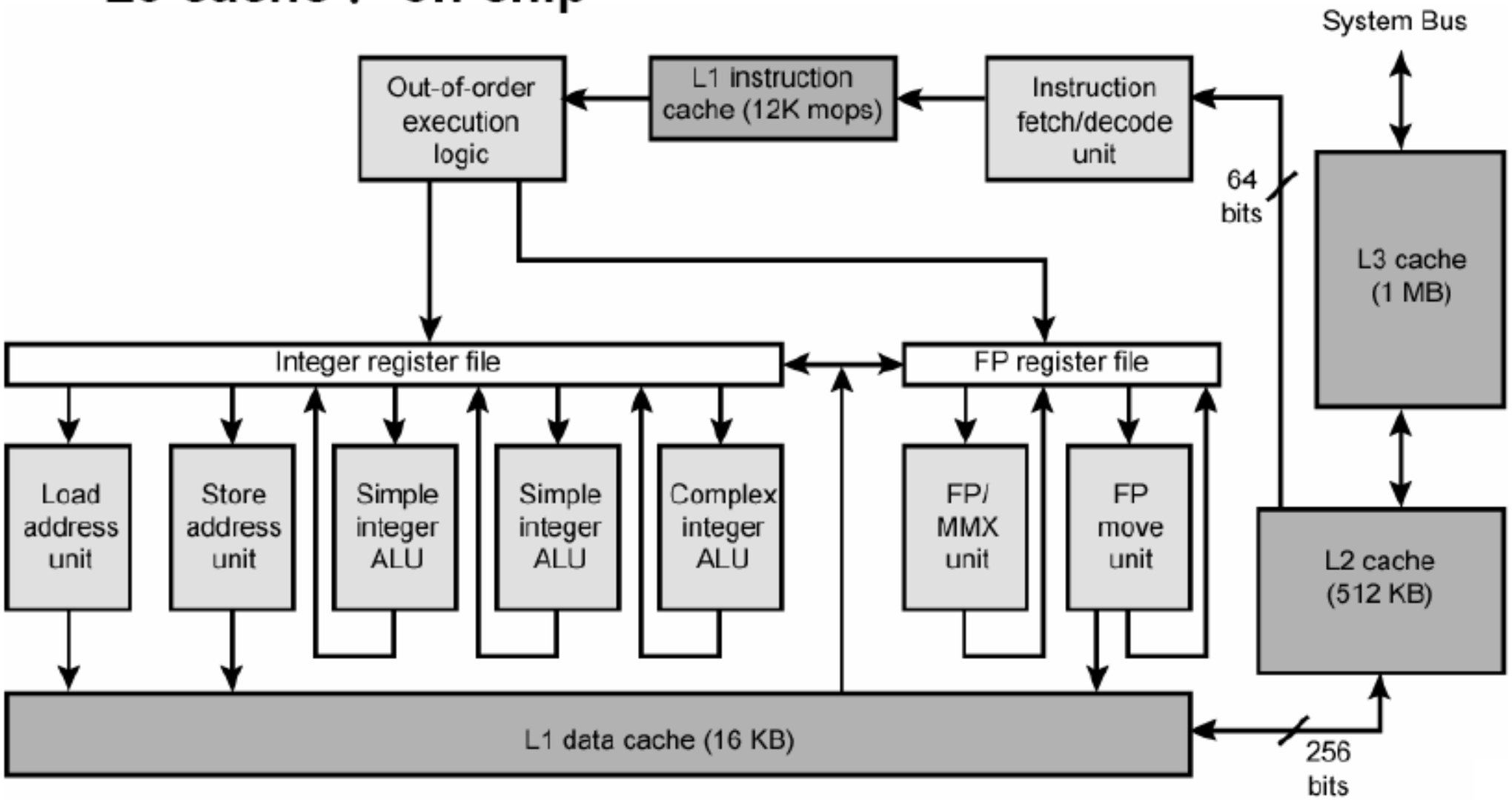
- 两级**Cache**
  - **L1**：**8～16KB**，**CPU**内部集成
    - 数据 **Cache**：可随机读/写，双端口存储器
    - 指令 **Cache**：只读
  - **L2**：容量大，在**CPU**外部
- 地址映射：组相联。
  - 对于**8K**的**L1 Cache**：
    - **128**组，每组**2**块(行)，每块(行)**32B**。
- 替换算法：**LRU**

7.3.5 Pentium的Cache



PentiumII结构框图

- **Pentium 4 Cache**
  - L1 caches: 64 byte lines，4 way set associative
  - L2 cache: 128 byte lines，8 way set associative
  - L3 cache : on chip



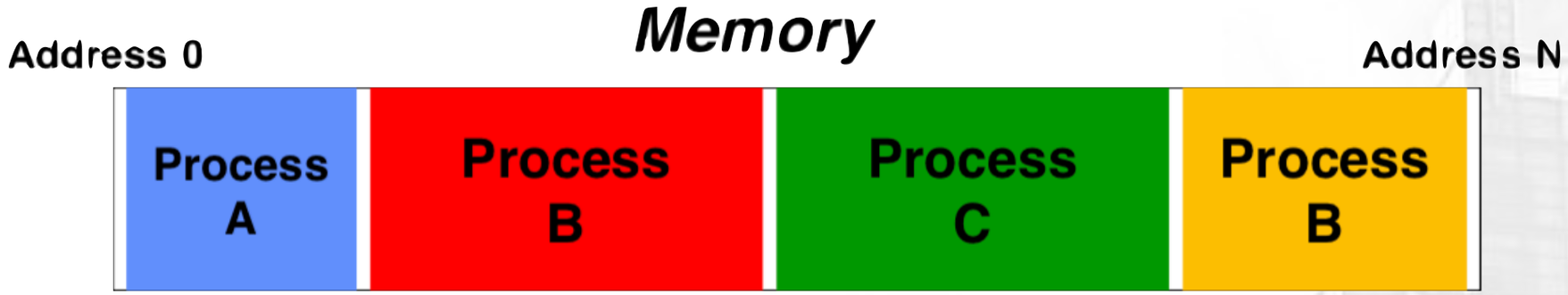
The Latest …

- **Intel Broadwell Arch. (2014)**
  - L1 64KB per core, L2 256KB per core, L3 2-8MB shared
  - L4 128MB eDRAM victim cache
- **Intel Kaby Lake Arch. (2016)**
  - L4 128MB also shared with GPU memory
  - Up to 40MB L3 on Xeon models
- **Qualcomm Snapdragon 845 (2017)**
  - L1 128KB, L2 2MB
- **Apple A12 (2018)**
  - L1 256KB, L2 8MB

虚拟存储器

Virtual Memory, VM

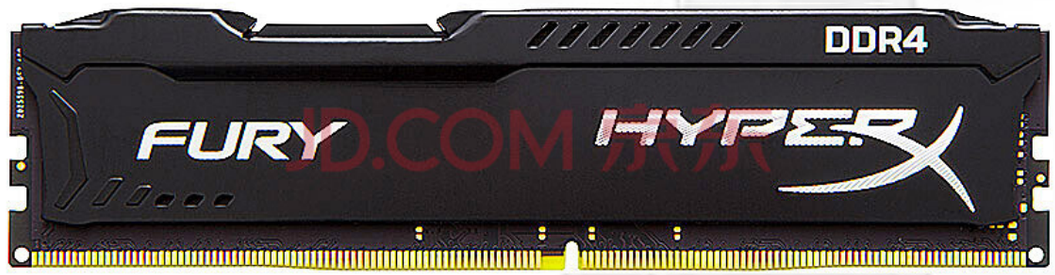
Q1:进程在内存中如何共存?



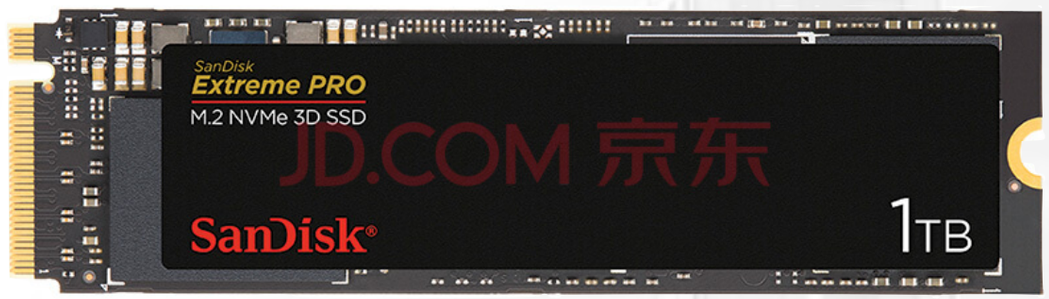
- 如果进程可以访问其它进程的内存，会怎样?
- 如果进程在运行时需要更多内存，怎么分配?
- 如果进程结束，内存如何分配再利用?

Q2: 内存真的够用么?

- **Kingston Fury DDR4 2400 16GB**
  - **RMB 1199**
  - **75RMB/GB**



- **SanDisk 1TB SSD**
  - **RMB 3499**
  - **3.4RMB/GB**



- **Price (at jd.com) on 2018.9.30**

VM的主要动机

- **提供进程内存的隔离**
  - VM机制防止跨进程访问内存(安全问题)
- **为每个进程提供一个“虚拟的巨大内存”任意访问**
  - 每个进程的VM可以远比实际内存大，PAE， 4TB on Linux
- **为每个进程提供一个“虚拟的连续内存空间”**
  - Coder/Compiler都不需要操心内存不连续问题

VM的次要动机

- 支持内存容量的动态调节
  - 机器运行时增加内存
- 支持超过物理容量的任务
  - 所有进程的VM空间之和 > 物理内存容量

VM提供了弹性的、安全的内存管理

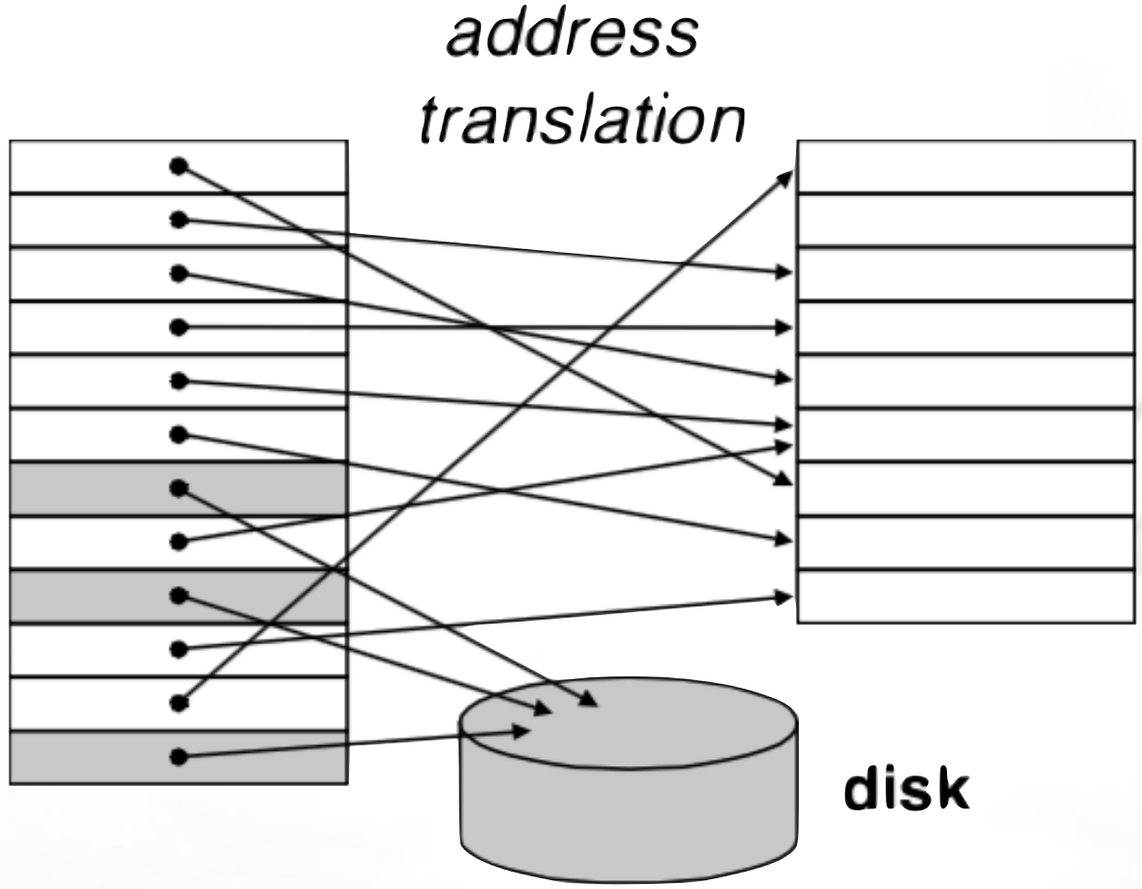
VM：两种地址空间

- 虚拟地址空间
  - 每个进程独占的“巨大 + 连续”的虚拟内存上的地址
- 物理地址空间
  - 实际的物理内存上的地址
  - 被操作系统内存+硬件完全屏蔽

VM的原理

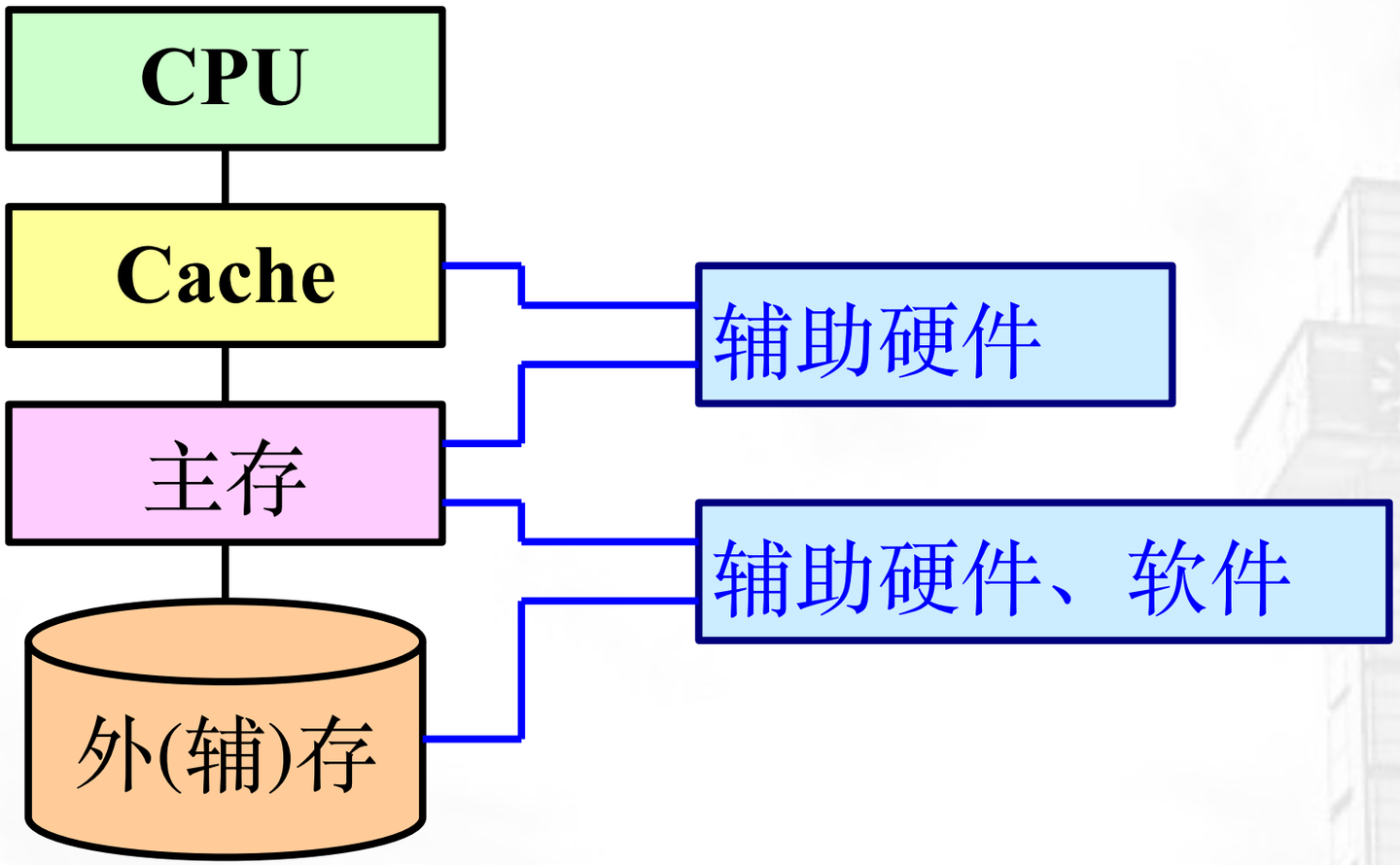
- 1. 把内存（物理&虚拟）分为同样大小的块或段
- 2. 每个进程在各自的互相隔离的VM中占用一些内存块或段
- 3. 通过“地址变换表”Address Translation Table，将VM中的块或段映射到物理内存
- 虚拟地址<—>物理地址变换由硬件完成~~

VM的原理



- 某种角度来说，内存成了外存的Cache
  - 实现“进程连续并且更大”的内存
  - 程序在内存中的实际位置无意味
  - 进程隔离、进程共享、内存动态管理等优势

7.4虚拟存储器——基本概念



Then, How ?

- 构造
  - 主存储器＋联机工作的外部存储器＋辅助硬件＋系统软件

7.4 虚拟存储器

- 因地址映象和变换方法不同，

  有三种虚拟存储器：

  - **段式虚拟存储器**
  - **页式虚拟存储器**
  - **段页式虚拟存储器**

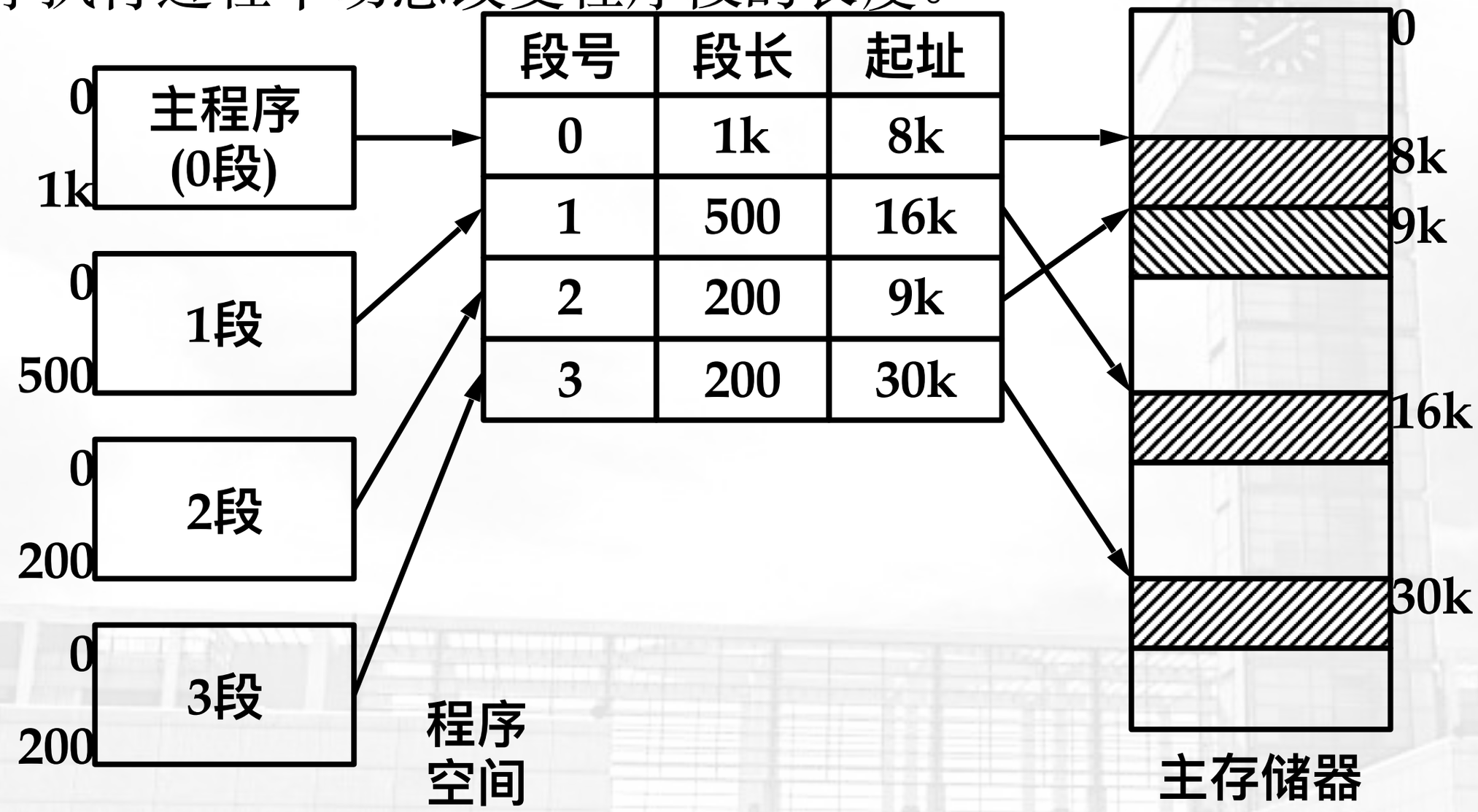
- 段式存储管理方式：

  将程序按逻辑意义分成段，按段进行调入、调出和管理。

段式虚拟存储器

- 地址映象方法：

每个程序段都从0地址开始编址，长度可长可短，可以在程序执行过程中动态改变程序段的长度。



段式虚拟存储器

- 地址变换方法：

① 由用户号找到基址寄存器；

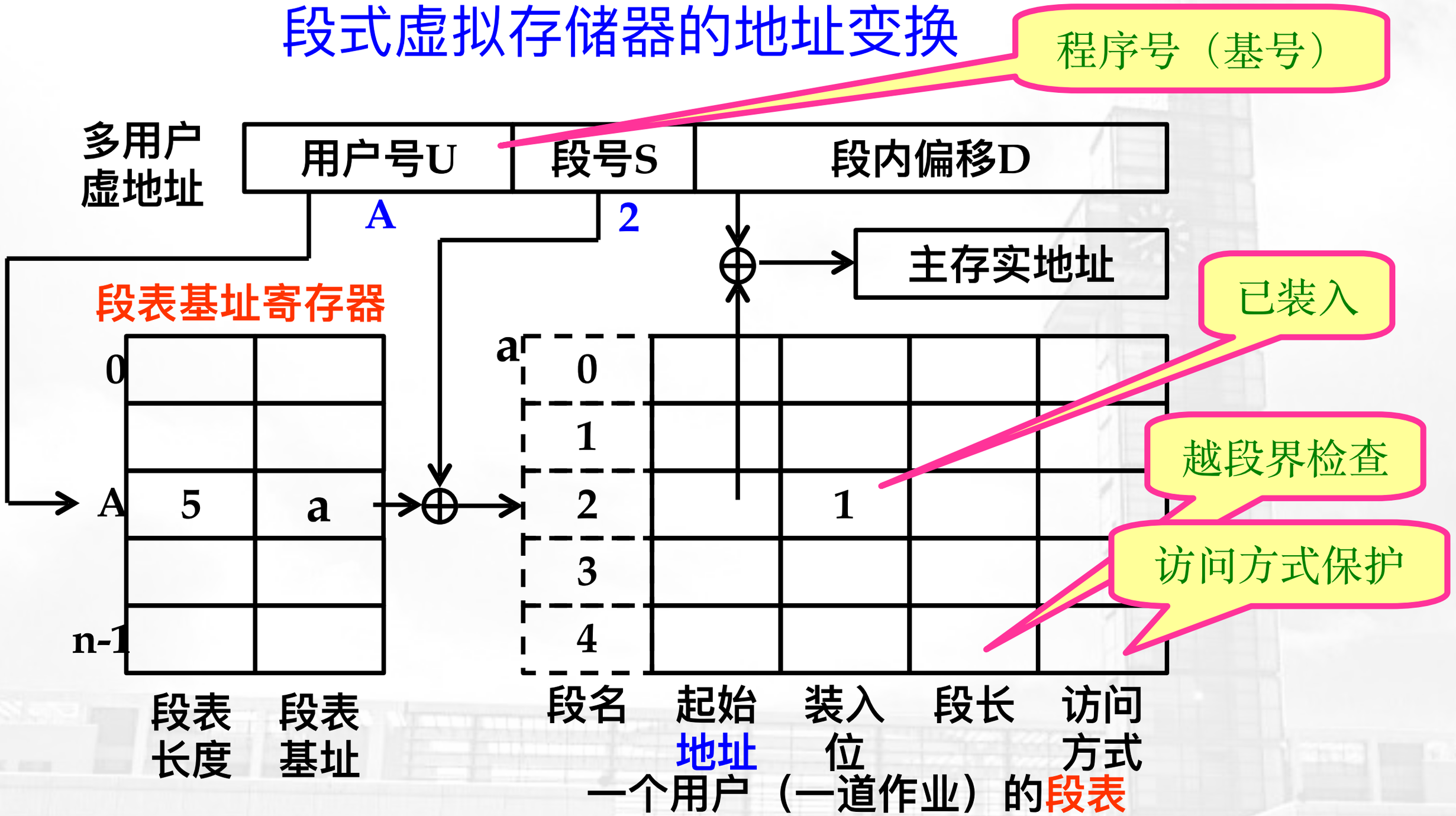
② 从基址寄存器中读出段表起始地址；

③ 把段表起始地址与虚地址中段号S相加得到段表中S段的地址；

④ 把S段的物理内存起始地址与段内偏移D相加就能得到主存实地址。

段式虚拟存储器

段式虚拟存储器的地址变换



段式虚拟存储器

- 主要优点：

① 程序的模块化性能好。

② 便于多道程序共享主存中的某些段。

③ 程序的动态链接和调度比较容易。

④ 便于按逻辑意义实现存储器的访问方式保护。

- 主要缺点：

① 地址变换所花费的时间长，两次加法。

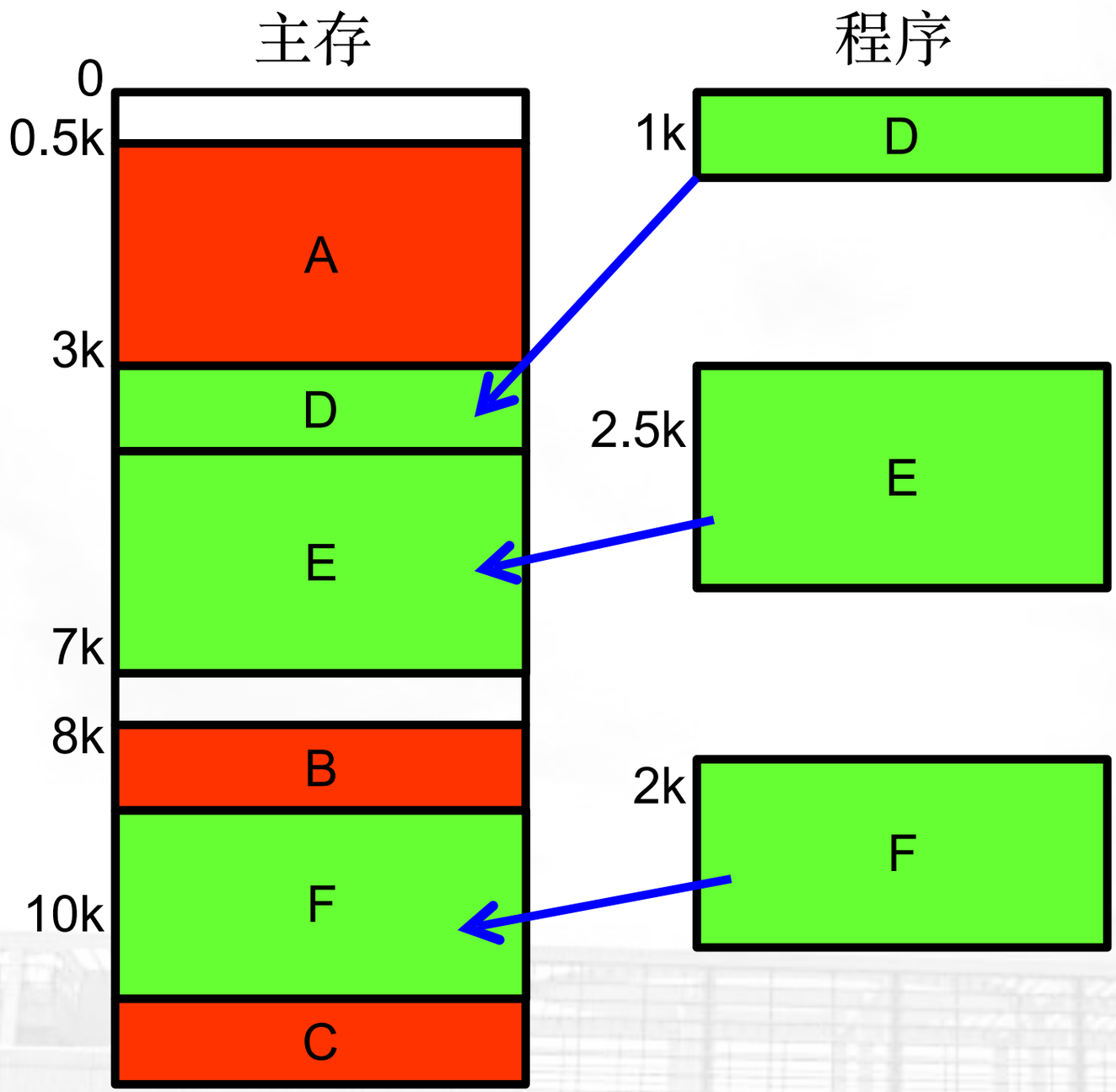
② 段映象表庞大，地址、段长字段太长。

③ 主存储器的利用率往往比较低——存储管理复杂；段间“零头”。

④ 对辅存（磁盘存储器）的管理比较困难。

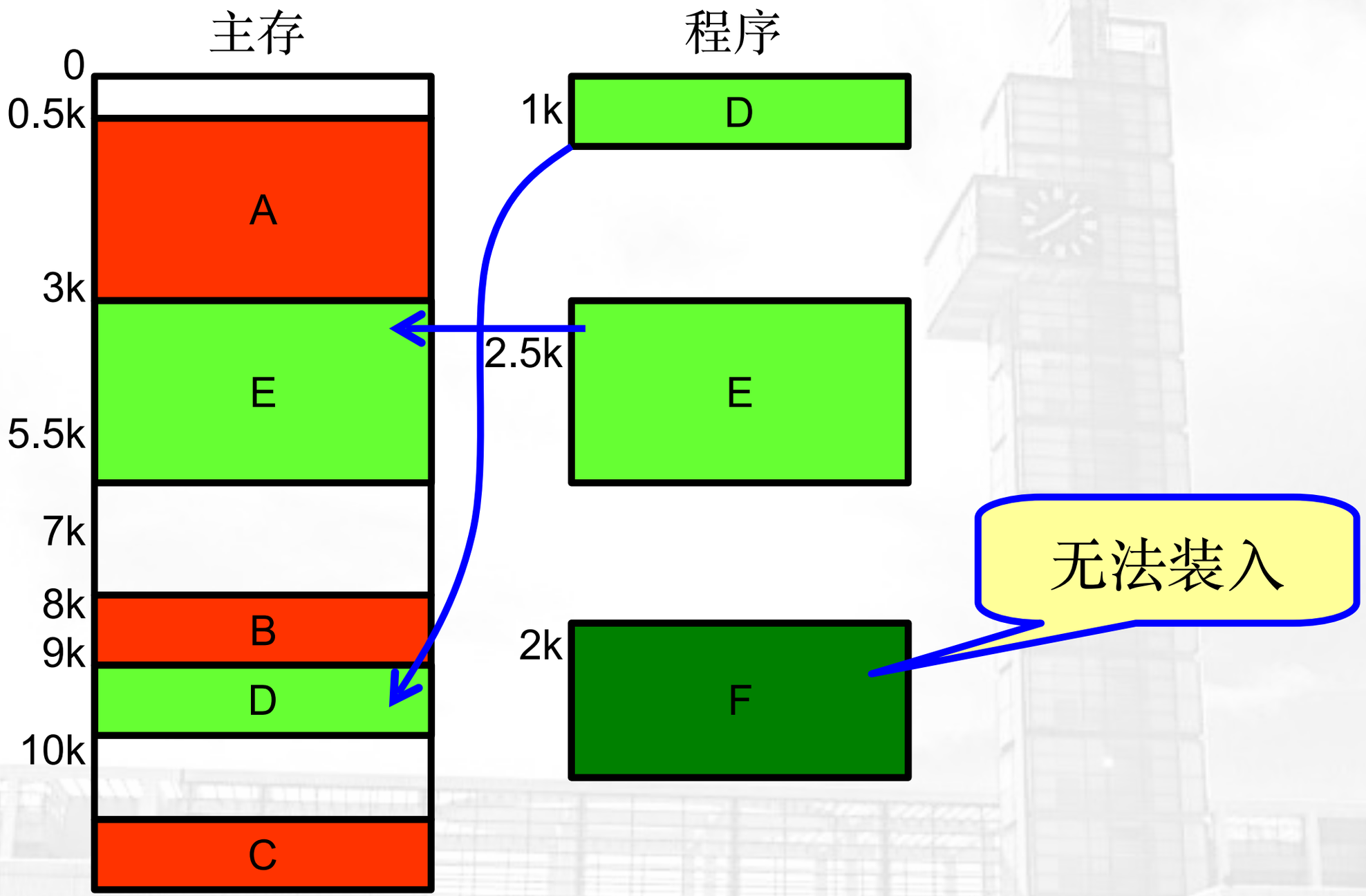
段式虚拟存储器

首先分配法



段式虚拟存储器

最佳分配法



页式虚拟存储器

- 页式存储管理方式：

  将物理空间和VM空间都等分成相同大小的页面(Page)，按页顺序编号，让程序的起点必须处在主存中某一个页面位置的起点上。

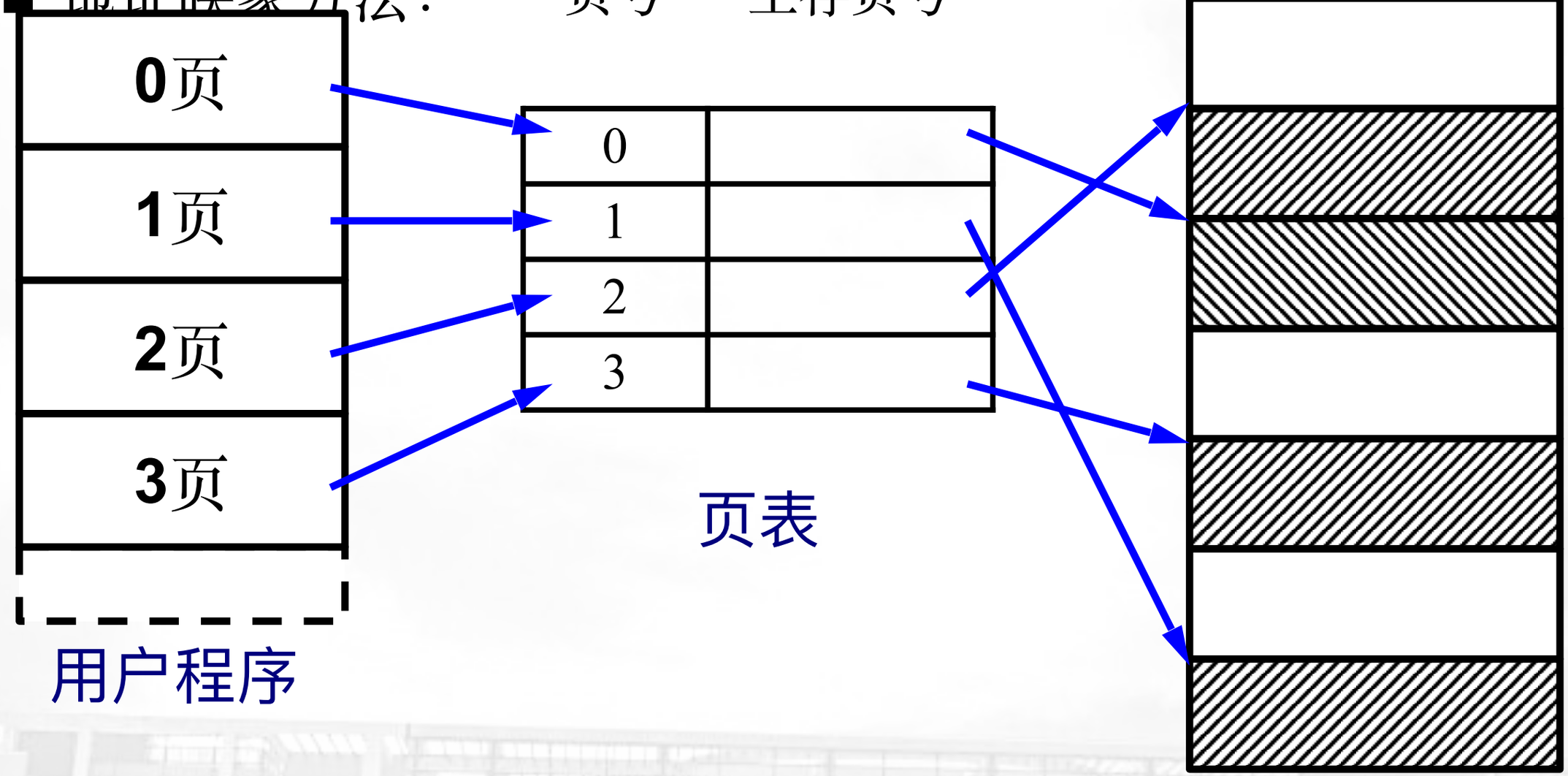
  任一主存单元的地址由页号和页内位移两个字段组成。

页式虚拟存储器

虚页/逻辑页

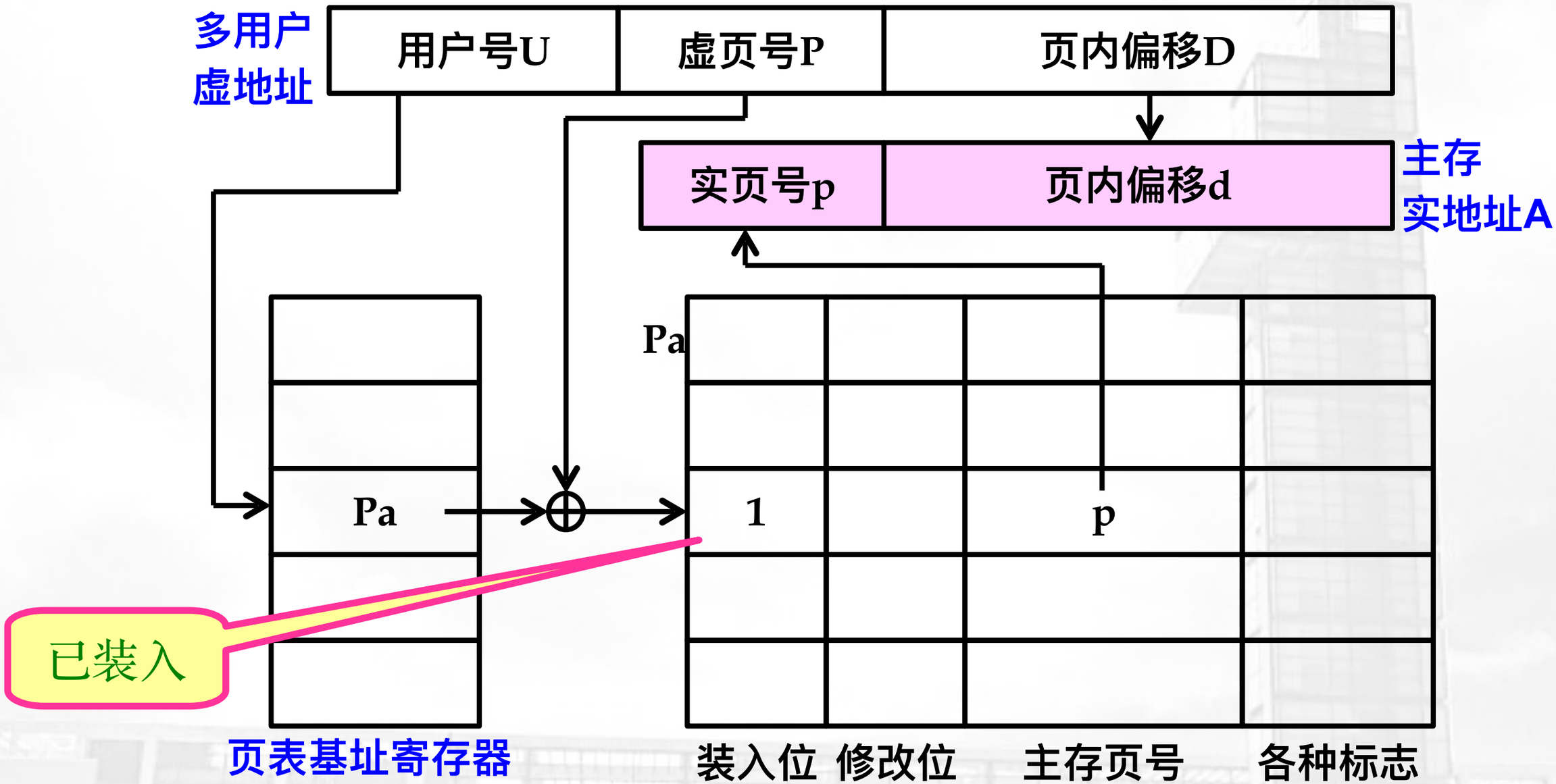
实页/物理页

地址映象方法：　　页号　　主存页号



页式虚拟存储器

- 地址变换方法：



7.4 虚拟存储器　　三、页式虚拟存储器

■主要优点：

① 主存储器的利用率比较高。

② 页表相对比较简单，使用硬件少。

③ 地址变换的速度比较快。

④ 对磁盘的管理比较容易。

■ 主要缺点：

① 程序的模块化性能不好

② 页表很长，需要占用很大的存储空间

例如：虚拟存储空间**4GB**，页大小**1KB**，则页表的容量为**4M**存储字。如果每个页表存储字占用**4**个字节，则页表的存储容量为**16MB**。

段页式虚拟存储器

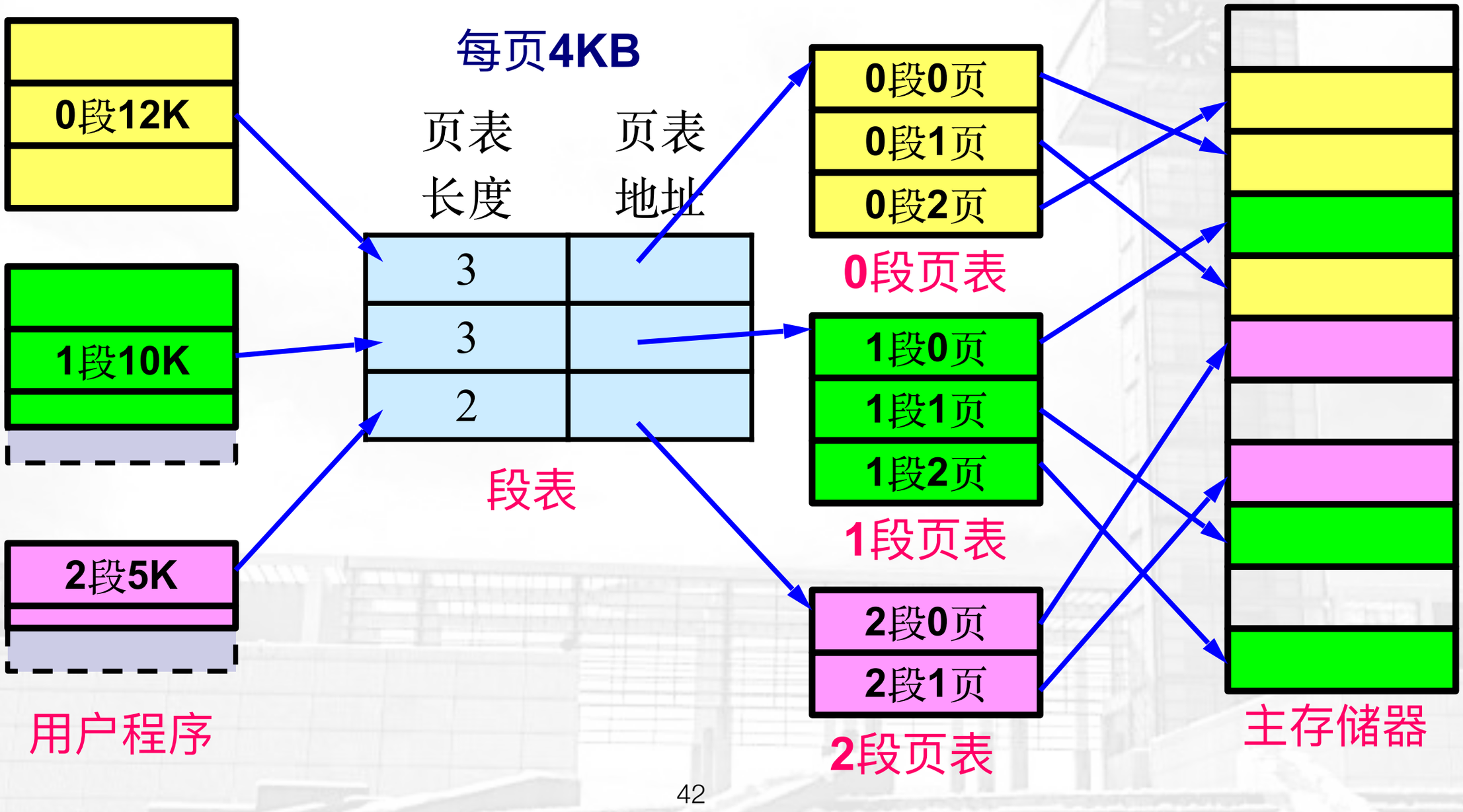
- 段页式存储管理方式：

  将程序按逻辑意义先分成段，再让各段和实主存都机械等分成相同大小的页面。每道程序通过一个段表和相应的一组页表来进行程序在主存空间中的定位。

段页式虚拟存储器

■ 地址映象方法：

每个程序段在段表中占一行， 在段表中给出页表长度和页表的起始地址，页表中给出每一页在主存储器中的实页号。



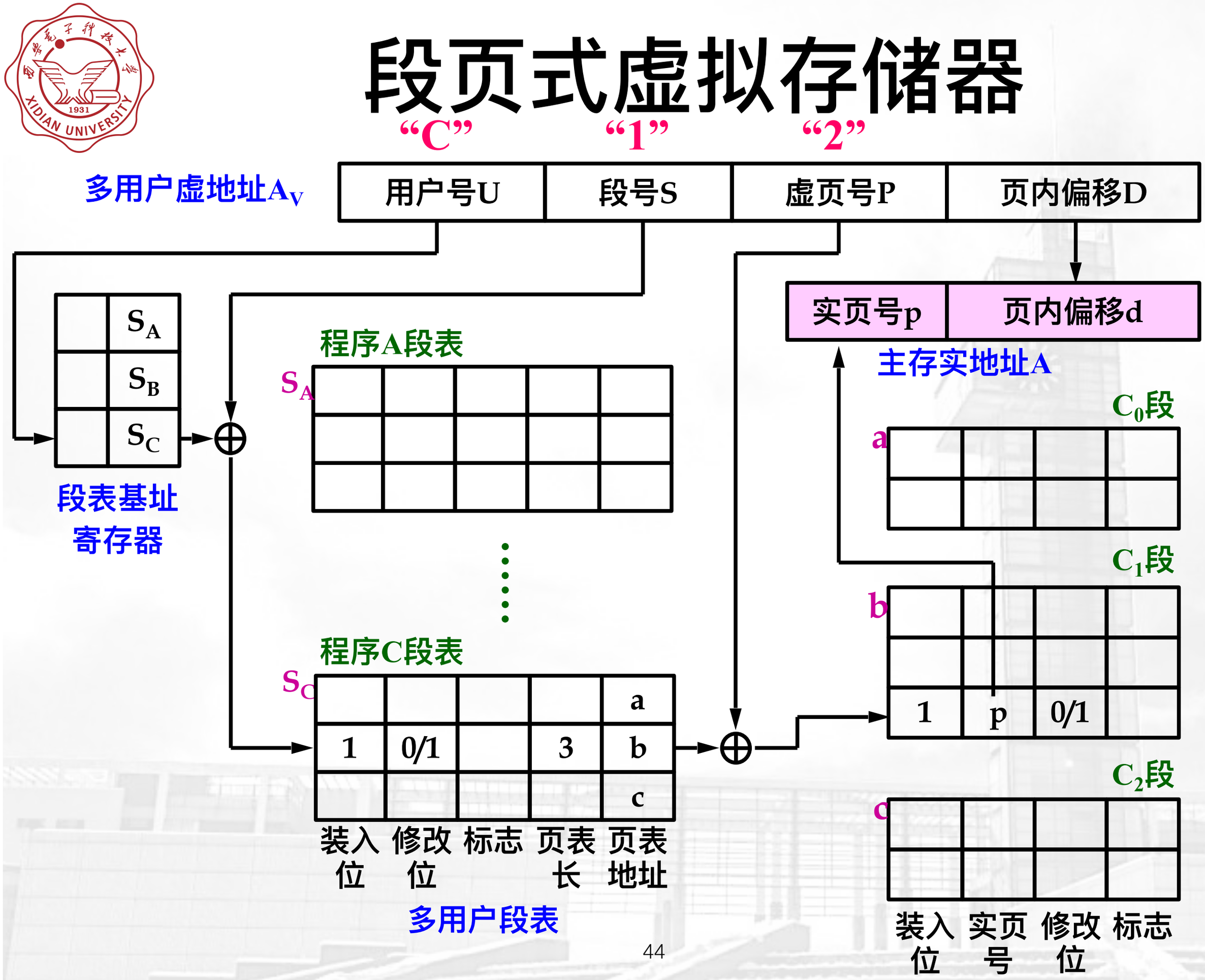
■ 地址变换方法：

① 先查段表，得到页表起始地址和页表长度；

② 再查页表找到要访问的主存实页号；

③ 把实页号**p**与页内偏移**d**拼接得到主存实地址。

段页式虚拟存储器


“C”
“1”
“2”
多用户虚地址A_V
用户号U
段号S
虚页号P
页内偏移D
实页号p
页内偏移d
主存实地址A
S_A
S_B
S_C
段表基址寄存器
程序A段表
S_A
C_0段
a
C_1段
b
C_2段
c
多用户页表
程序C段表
S_C
1
0/1
3
a
b
c
装入位
修改位
标志
页表长
页表地址
多用户段表
1
p
0/1
装入位
实页号
修改位
标志

VM页替换

- 物理内存不足时，将暂时不用的物理页替换到外存里可以维持系统运行， 虽然慢
- 常规策略
  - LRU，最近最少使用
    - 需要为每一页记录时间戳
  - FIFO，先进先出

VM页替换

- 如果所需页面不在内存...

1. CPU发现缺页
2. CPU发出Page Fault中断
3. 操作系统对Page Fault中断的处理程序启动
4. PF中断处理程序从外存调数据进内存（文件系统等...)
5. 操作系统更新页表（有效位=1）
6. 操作系统让CPU继续之前工作

VM页替换

- 虚拟存储器地址变换带来的速度问题：访问主存储器的速度要降低几倍 —— 不符合存储体系的要求。
  原因：

  ① 查段表、页表。

  - 如果段表、页表都在主存中，则包括访问主存本身这一次在内，主存的访问速度要降低2～3倍。

  - 如果段表、页表不在主存中，访问辅存，速度更低。

  ② 当页表或段表容量超过一个页面大小时，它们可能被映象到主存的不连续页面。

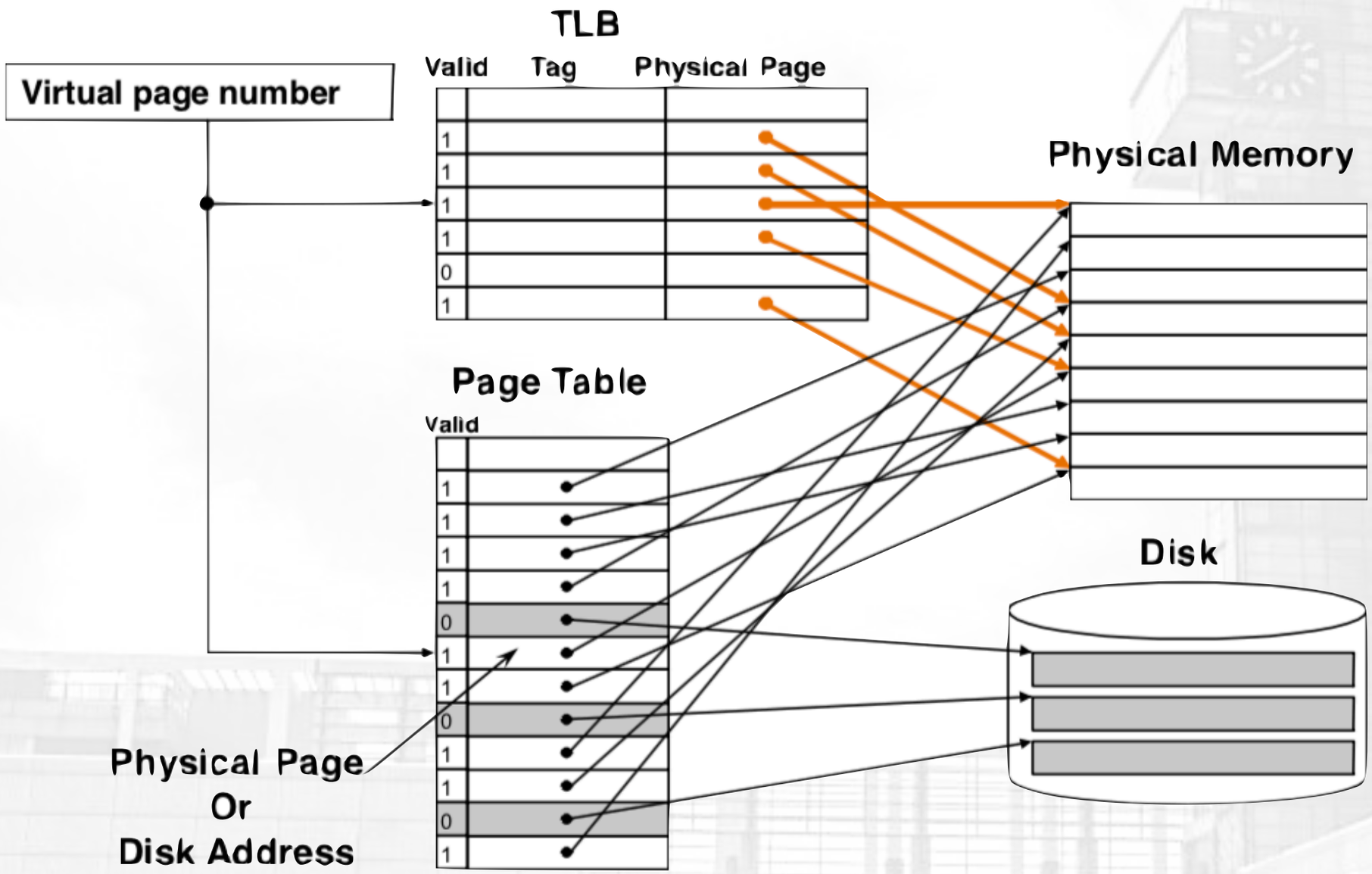
TLB (Translation Look-Aside Buffer)

- 页表的反复查询有明显的性能代价
  - 速度太慢，buffer上！

- TLB (Translation Look-Aside Buffer)
  - CPU内部专用的页表缓冲区！
  - 存储等级高于L1 Caches！

TLB (Translation Look-Aside Buffer)

- TLB，是**CPU内部对页表的专用缓冲区**



TLB (Translation Look-Aside Buffer)

- TLB, Cache, VM的访问路径