# 计算机组成与结构
# —存储系统4

**计算机科学与技术学院**


# 虚拟存储器
# Virtual Memory, VM

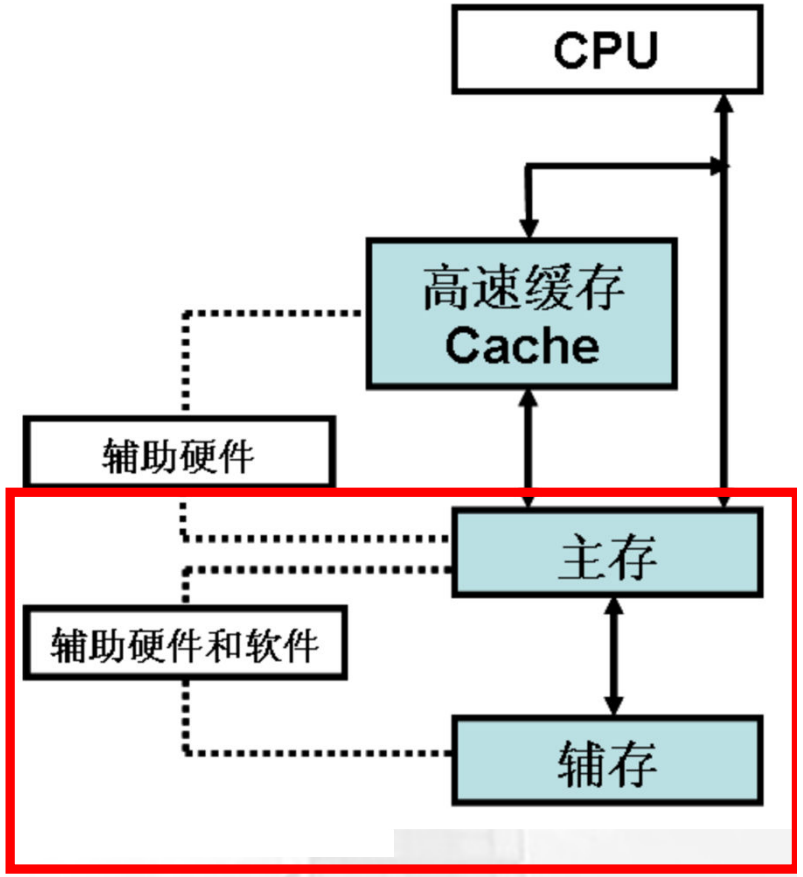
# 存储系统

## 存储系统的多层次结构

cache-主存-辅存三级存储体系：三级存储层次

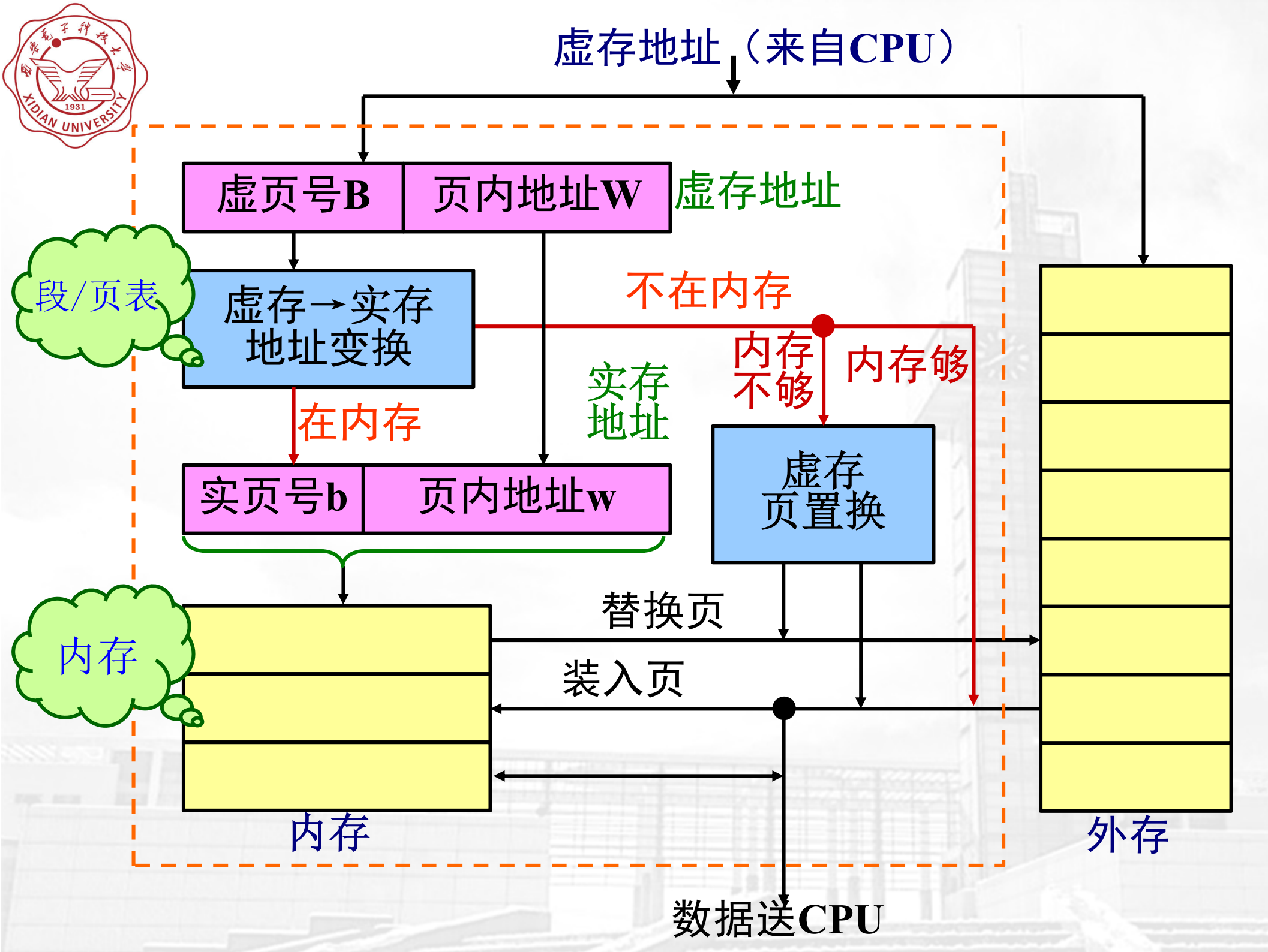
主存-辅存存储层次：　大容量、低成本；软硬件结合完成

cache-主存存储层次：　高速度、低成本；纯硬件完成




解决存储系统三个指标：容量、速度、价格/位的矛盾

**虚拟存储**


虚存地址（来自CPU）
虚页号B
页内地址W
虚存地址
段/页表
虚存→实存
地址变换
不在内存
内存
不够
内存够
实存
地址
在内存
虚存
页置换
实页号b
页内地址w
替换页
内存
装入页
内存
外存
数据送CPU

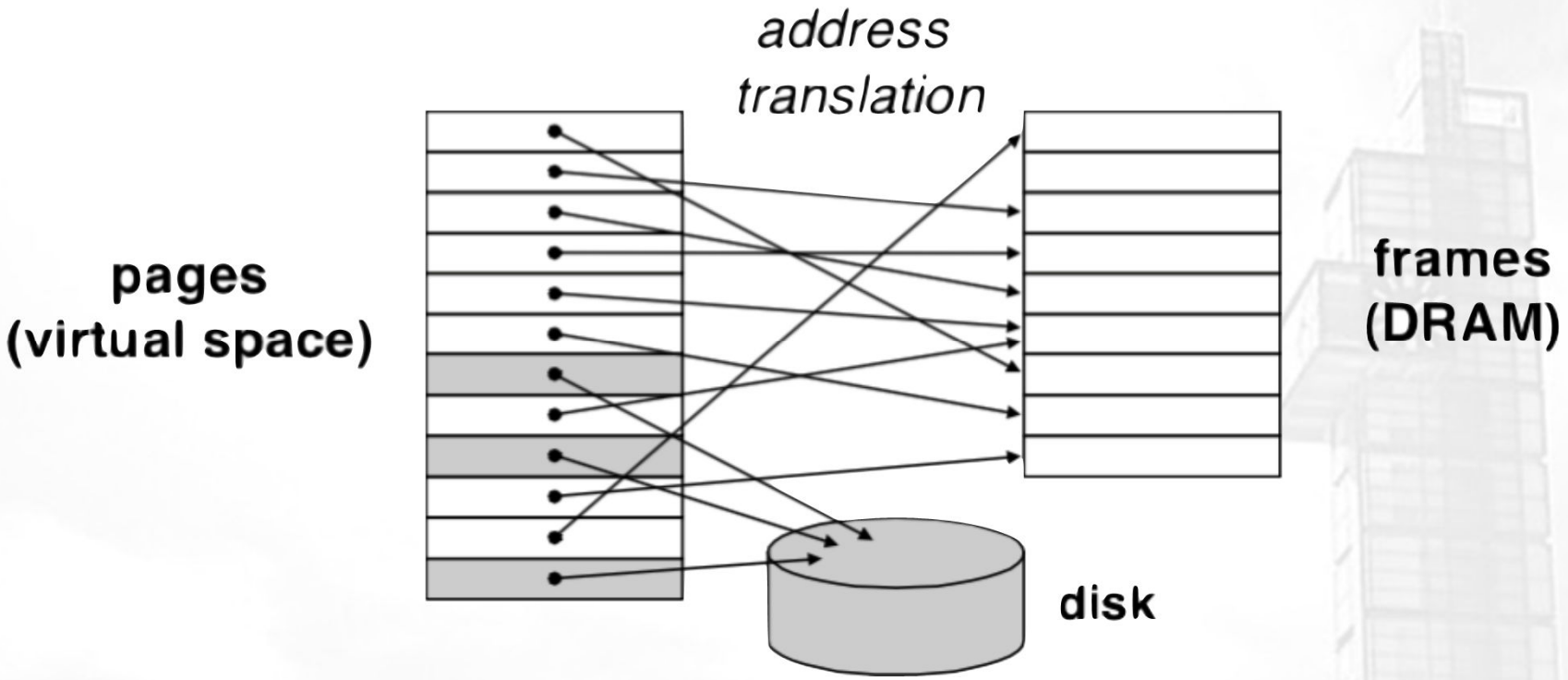
# VM：两种地址空间

- **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+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

- 三种地址映象和变换方法
  - **段式虚拟存储器**
  - **页式虚拟存储器**
  - **段页式虚拟存储器**

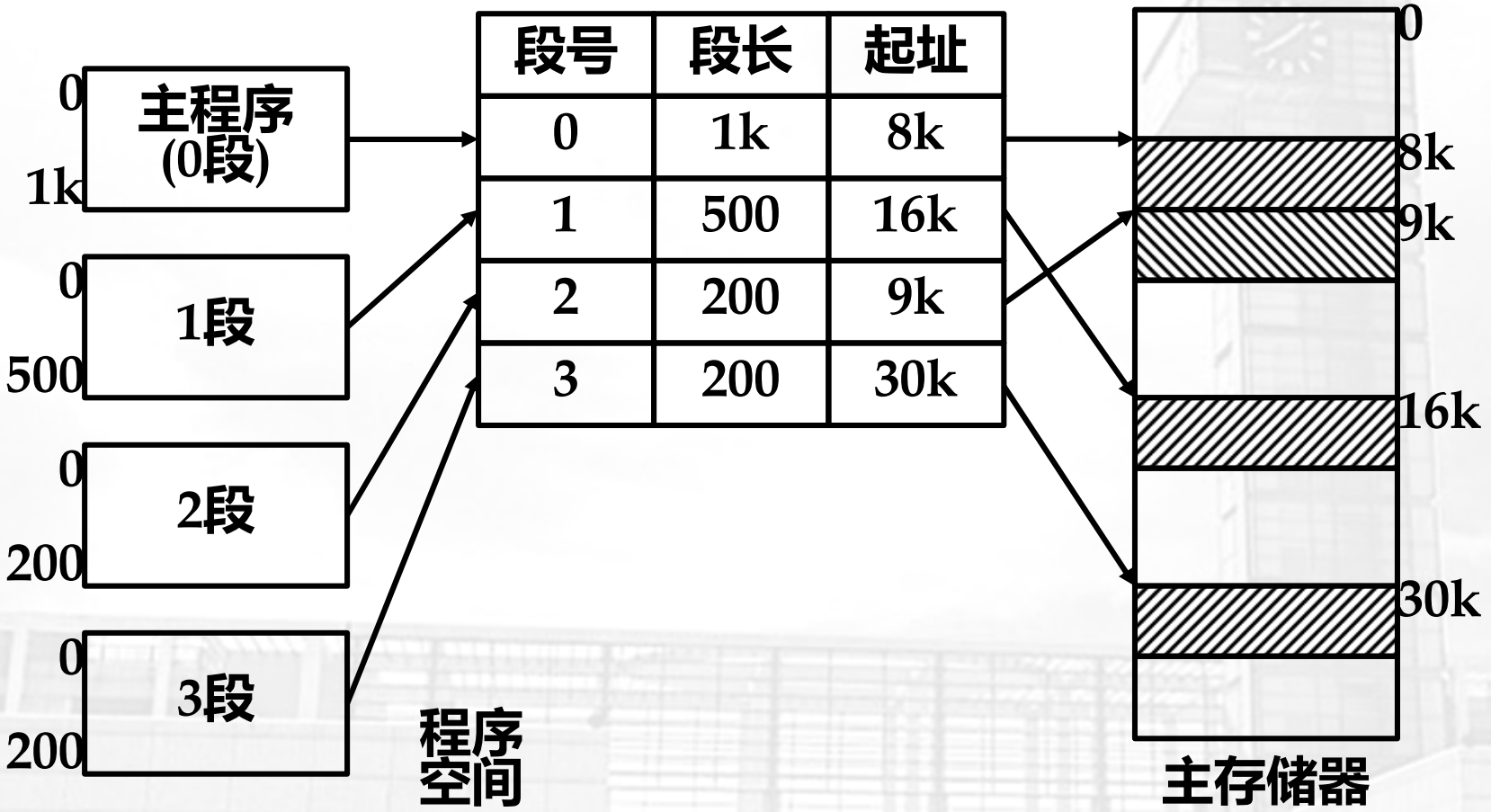
## 段式存储管理方式：

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

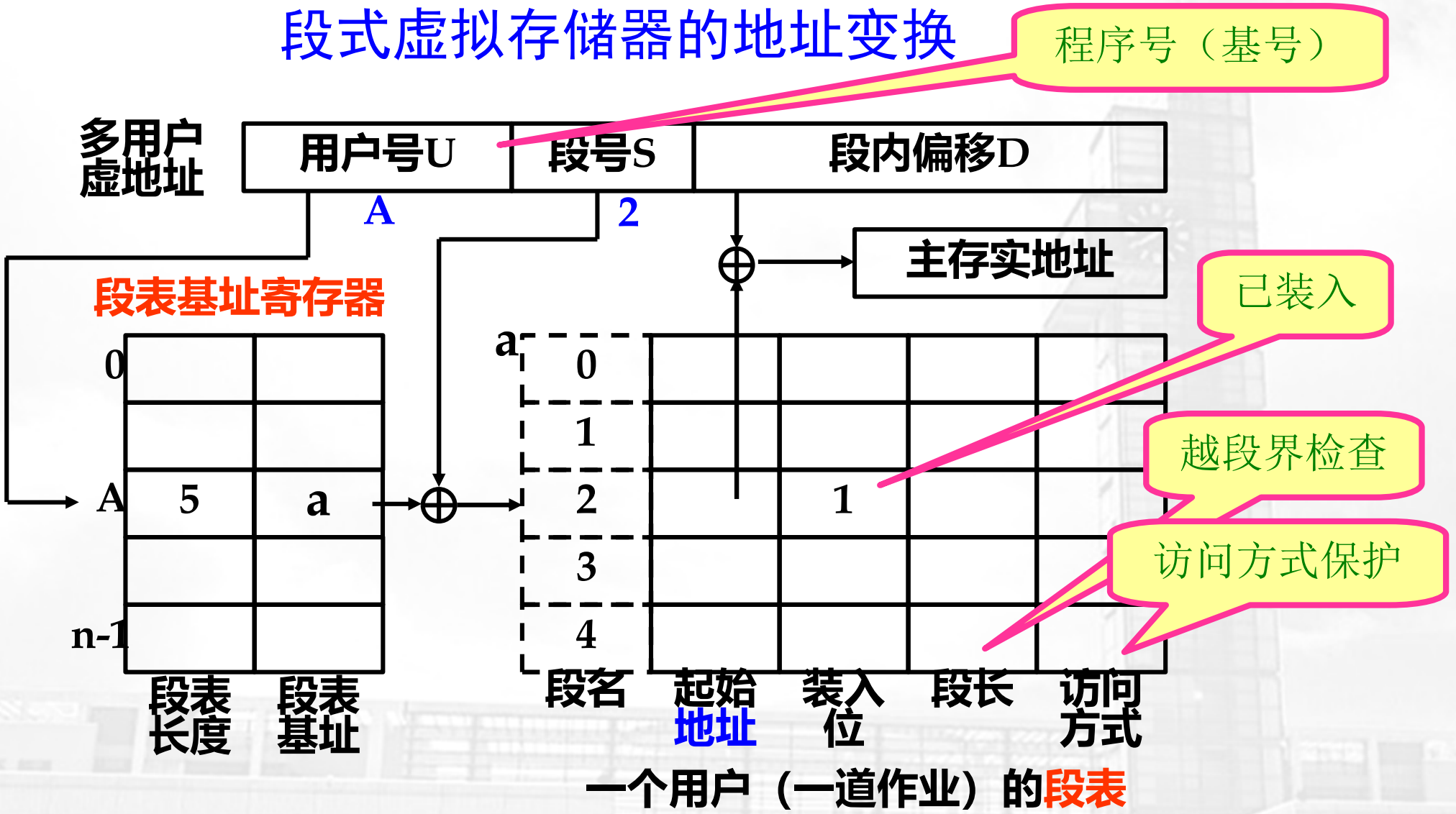
■ 地址映象方法：

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

## 段式虚拟存储器的地址变换



# 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

## 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。

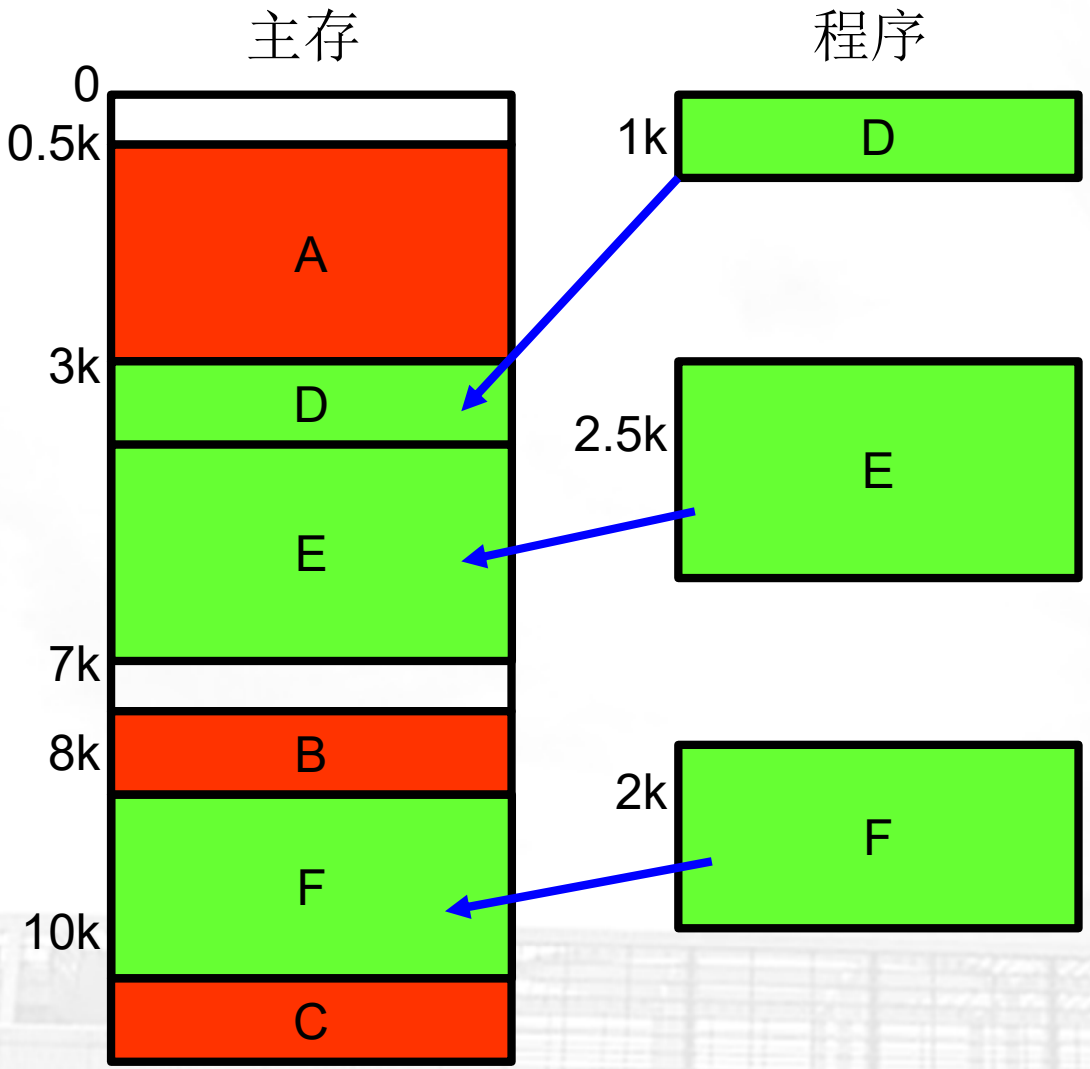
② 段映象表庞大，地址、段长字段太长。

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。

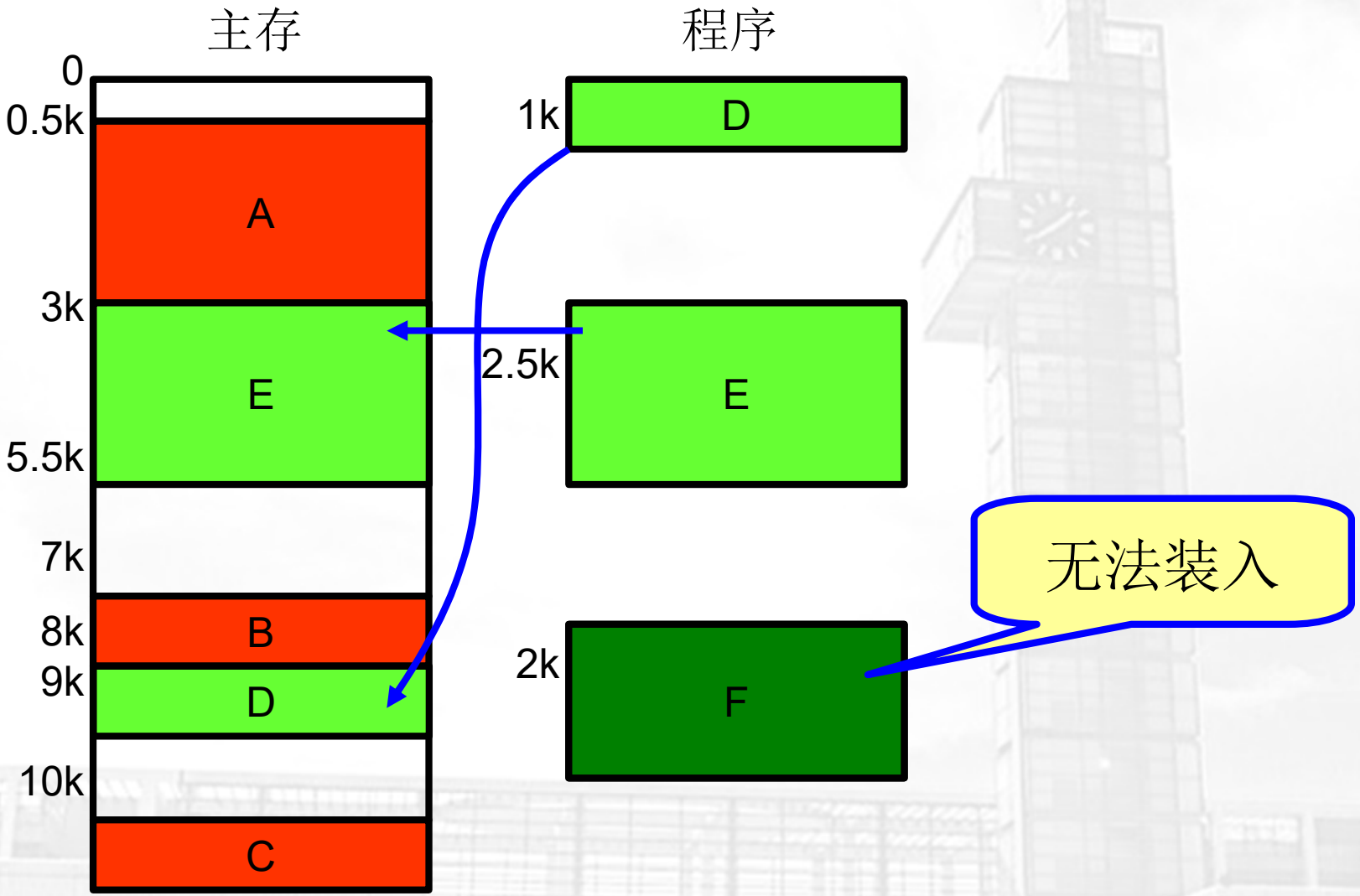
# 段式虚拟存储器

首先分配法



# 段式虚拟存储器

最佳分配法



# 页式虚拟存储器

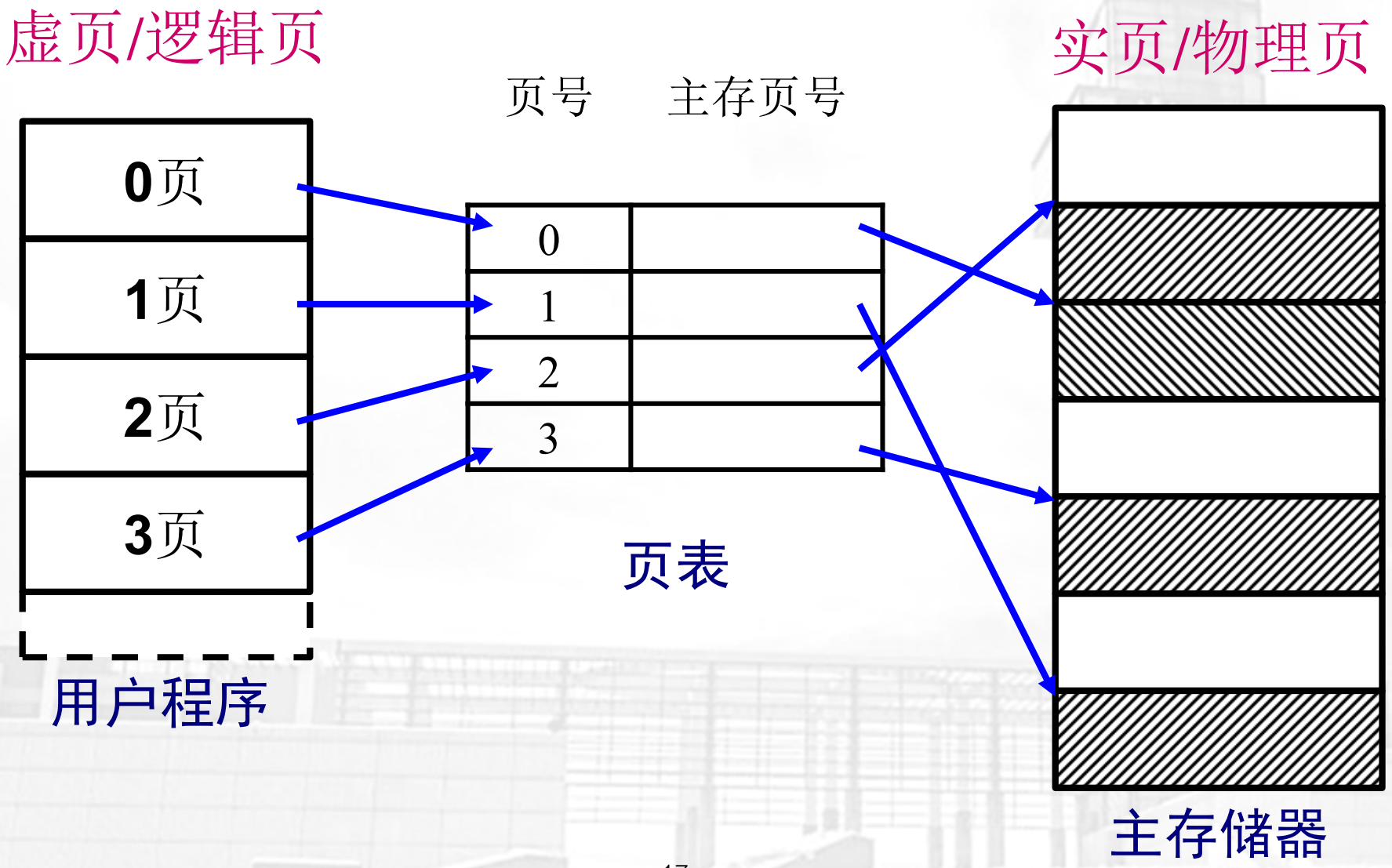
■ 页式存储管理方式：

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。

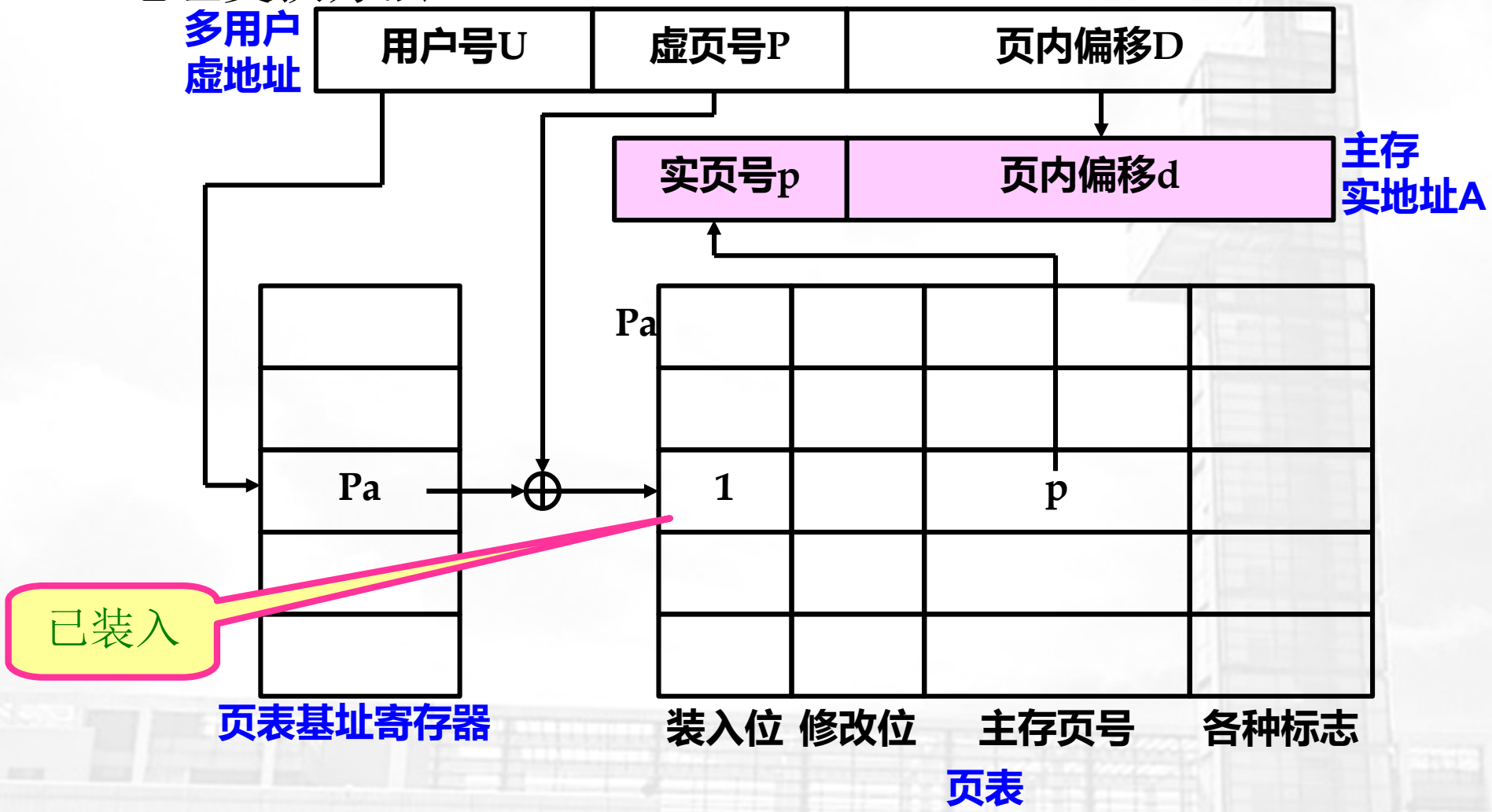
# 页式虚拟存储器

■ 地址映象方法：



# 页式虚拟存储器

■ 地址变换方法：



# 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

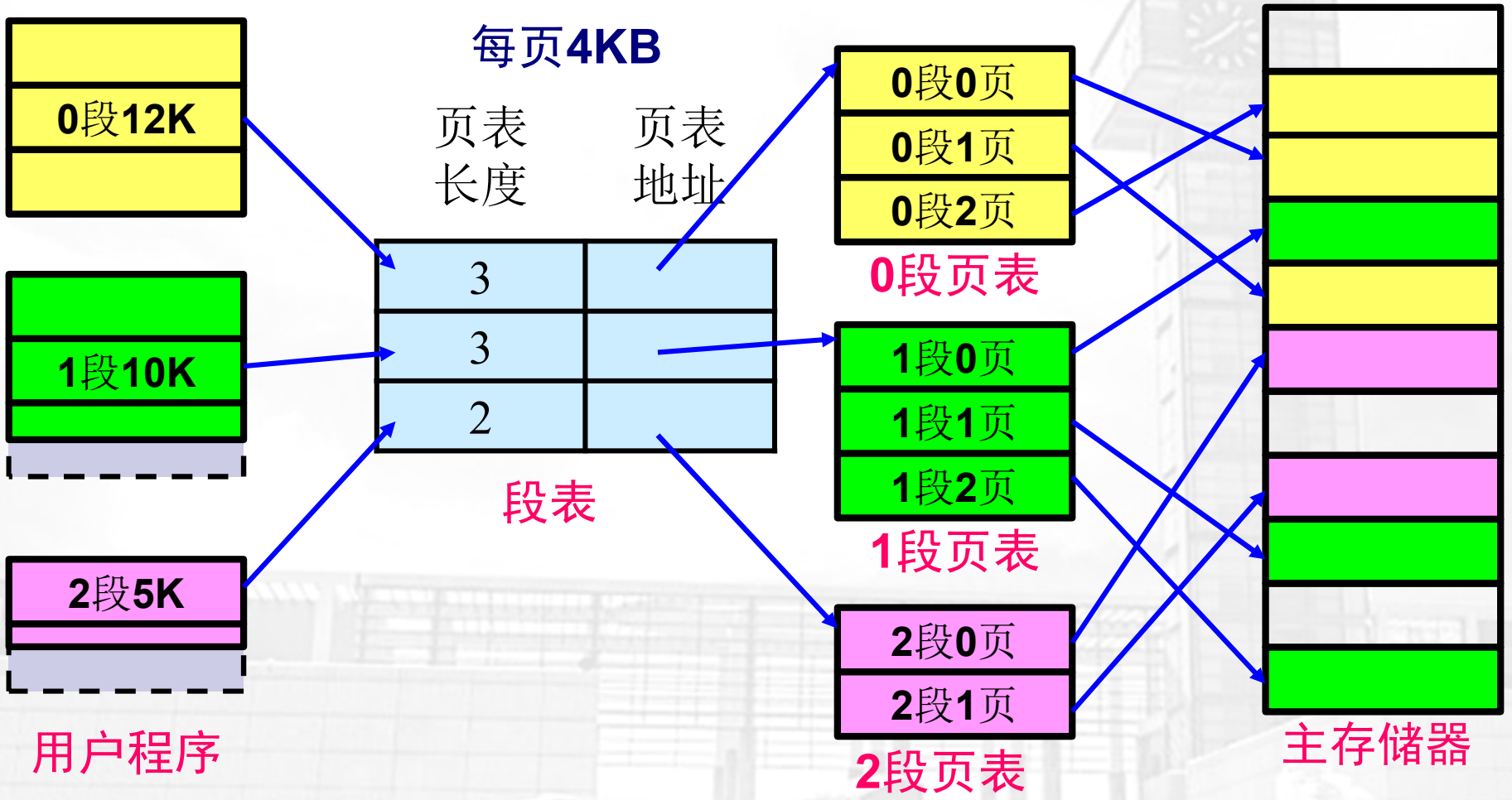
**段页式**存储管理方式：

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。

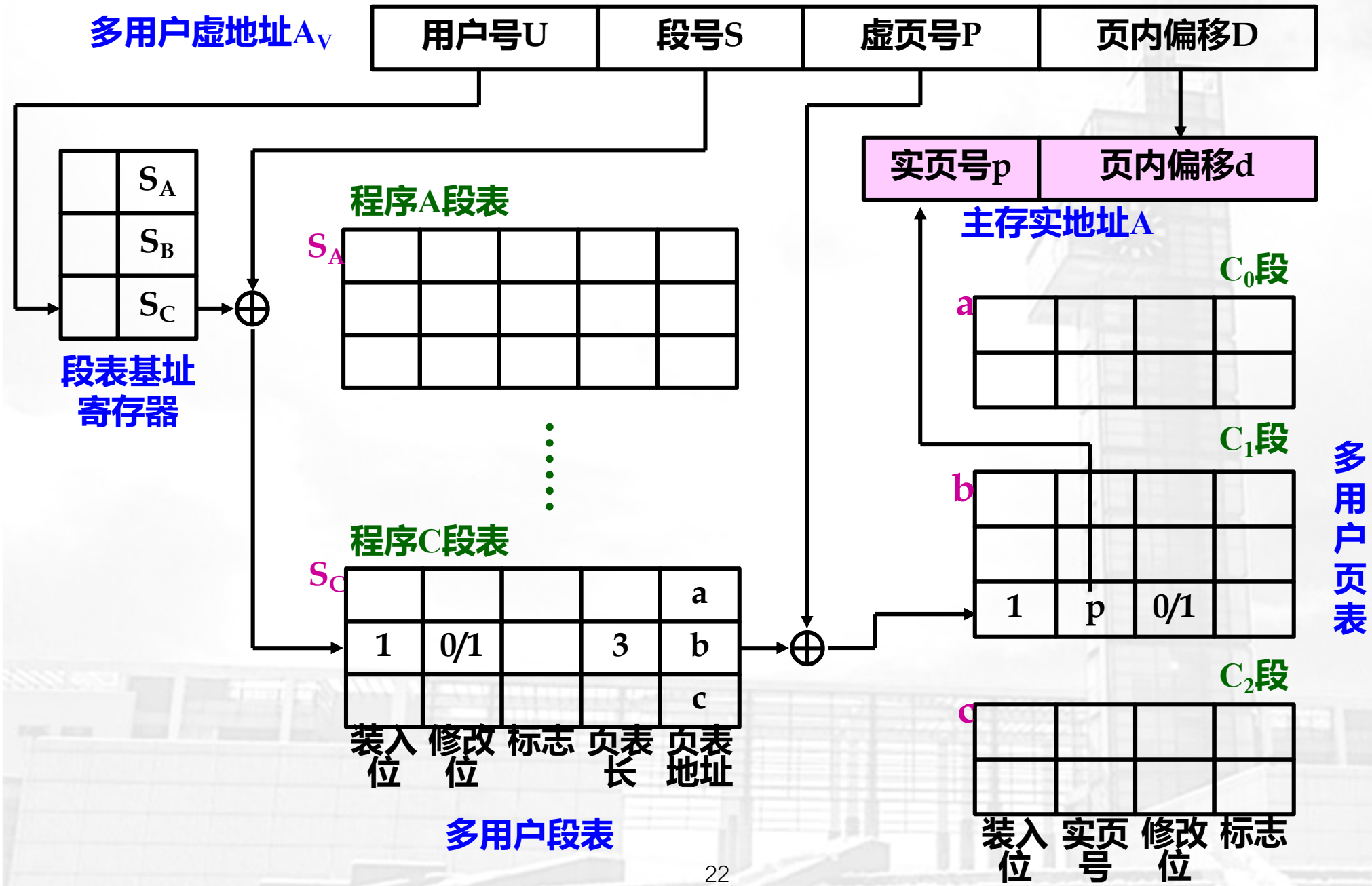
# 段页式虚拟存储器

**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。



# 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
$S_A$
$S_B$
$S_C$
段表基址寄存器
程序A段表
$S_A$
程序C段表
$S_C$
a
1
0/1
3
b
c
装入位
修改位
标志
页表长
页表地址
多用户段表
实页号p
页内偏移d
主存实地址A
$C_0$段
a
$C_1$段
b
1
p
0/1
$C_2$段
c
装入位
实页号
修改位
标志
多用户页表

# 段页式虚拟存储器

## 地址变换方法：

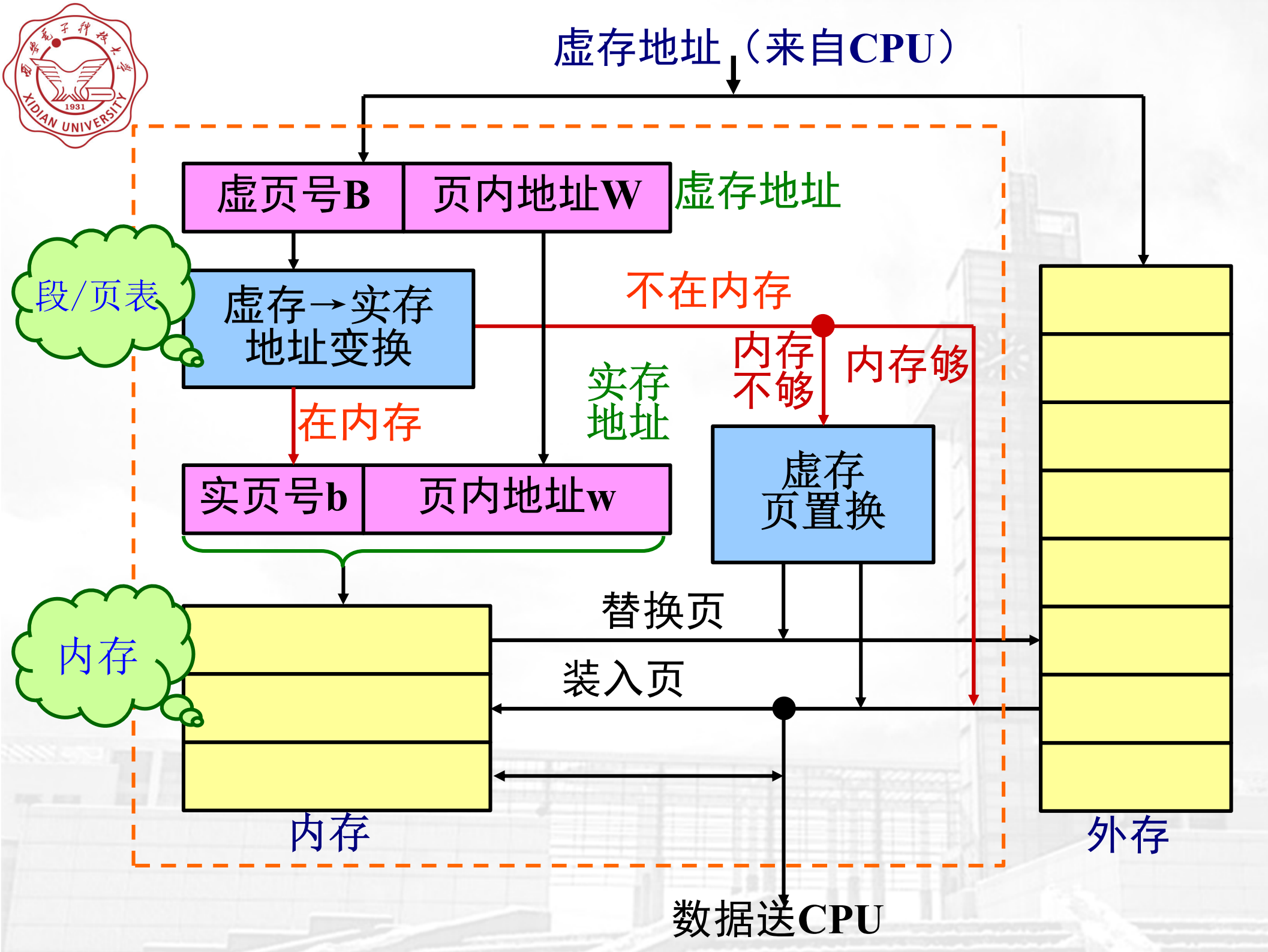
① 先查段表，得到页表起始地址和页表长度；

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。


# Linux的内存管理

- 严格来说，是段页式（兼容x86-386构架)
- 但是，所有的页都在段0
  - 因此，Linux实际是页式内存管理


虚存地址（来自CPU）
虚页号B
页内地址W
虚存地址
段/页表
虚存→实存
地址变换
不在内存
内存
不够
内存够
实存
地址
在内存
虚存
页置换
实页号b
页内地址w
替换页
内存
装入页
内存
外存
数据送CPU

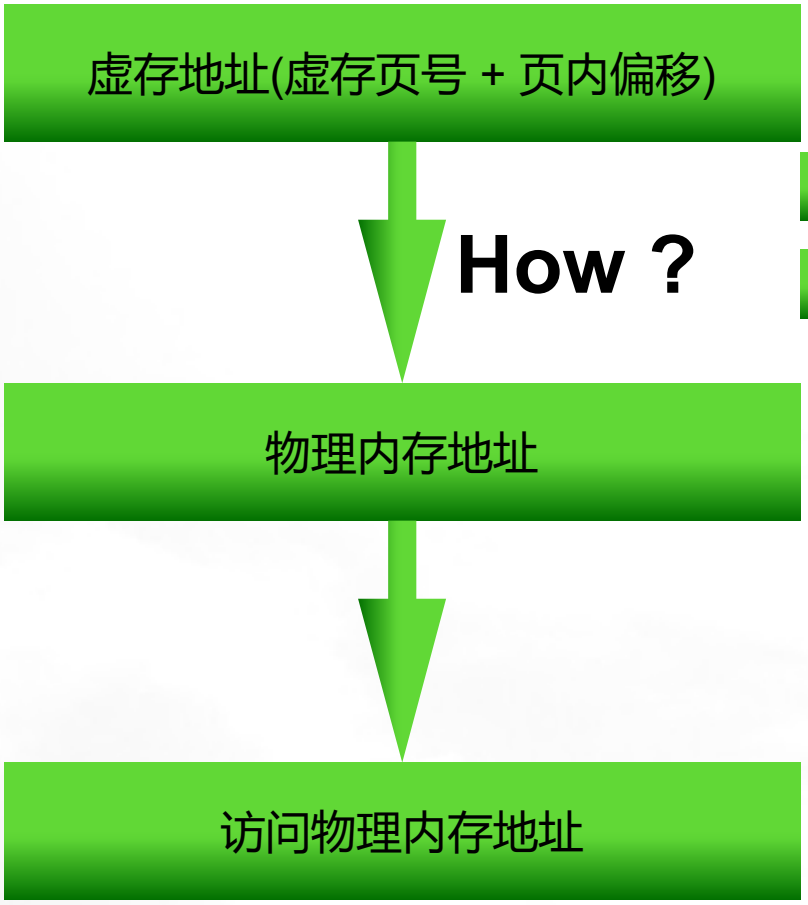
# VM访存流程

- CPU给出虚拟地址
  - 查段/页表
    - 如果在内存，访问
    - 如果所需页面不在内存…
      - CPU发出Page Fault(PF)中断
      - 启动操作系统注册的FP中断处理程序
      - 中断处理程序从外存调数据进内存（文件系统等…）
      - 操作系统更新页表（有效位=1）
      - 操作系统让切换回原进程

# 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
- 虽然慢，但可以维持系统运行
- 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模型下的内存访问




1. 找到当前进程的页表/段页表
2. 查表 + 计算

**不难推测几点：**

1. **地址转换(or 计算)成为CPU极频繁的工作；**
2. **页表被极频繁的访问；**
3. **so, 大量页表内容进驻cache；**
4. **cache容量有限，进程指令/数据的cache被挤占；**
5. **更可能的情况是，页表体积常远大于cache，使页表本身都无法快速访问；**
6. **系统性能严重下降**
7. **极端情况：页表都无法完全放在内存中，可能被置换到外存**

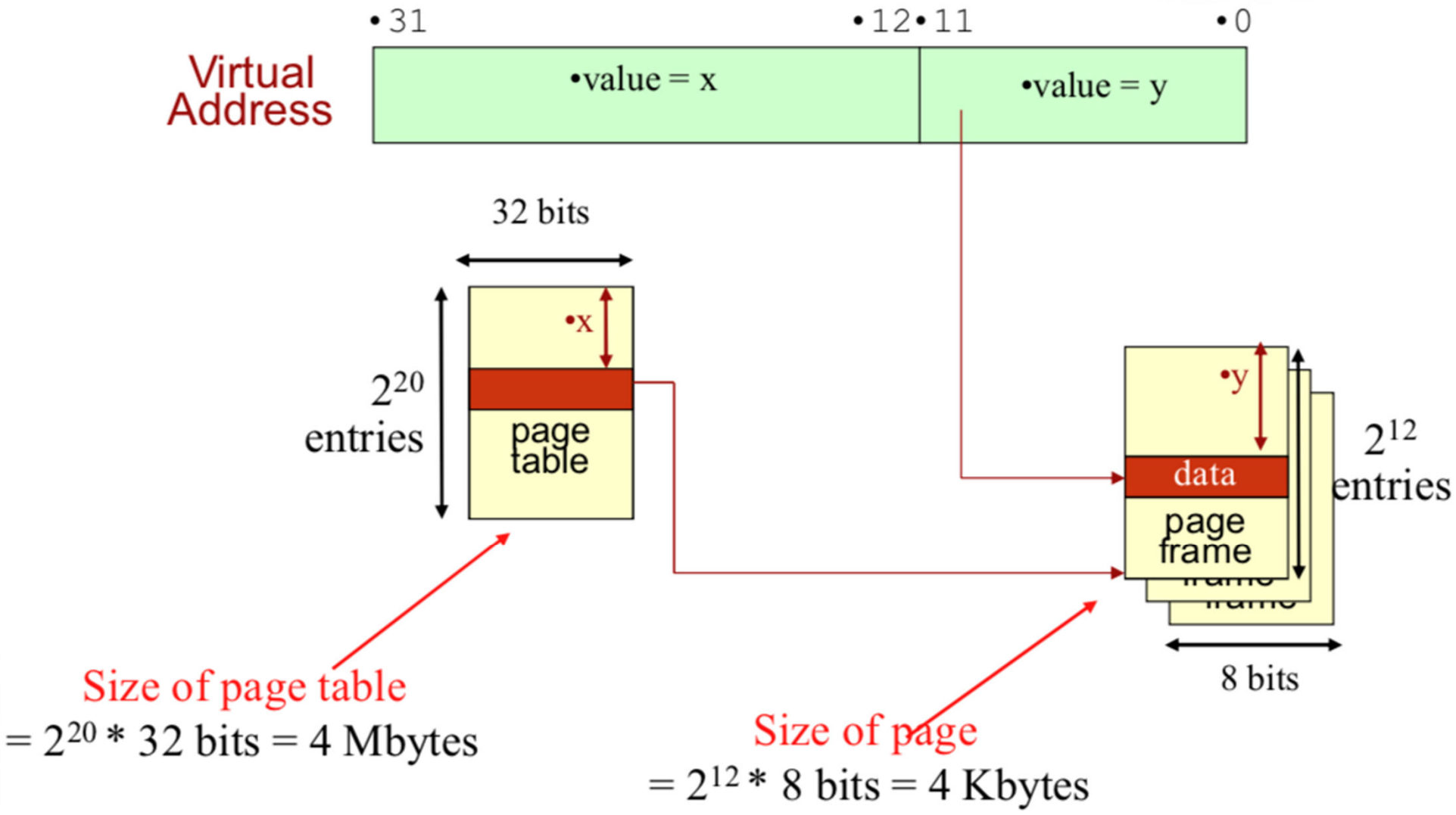
***So How to accelerate ?***

# VM模型下的内存访问

- **优化一：用专用的内存管理单元(MMU)硬件处理VM地址变换**
  - **MMU是专用硬件**
  - **页表被存储在特殊内存位置PTBR**

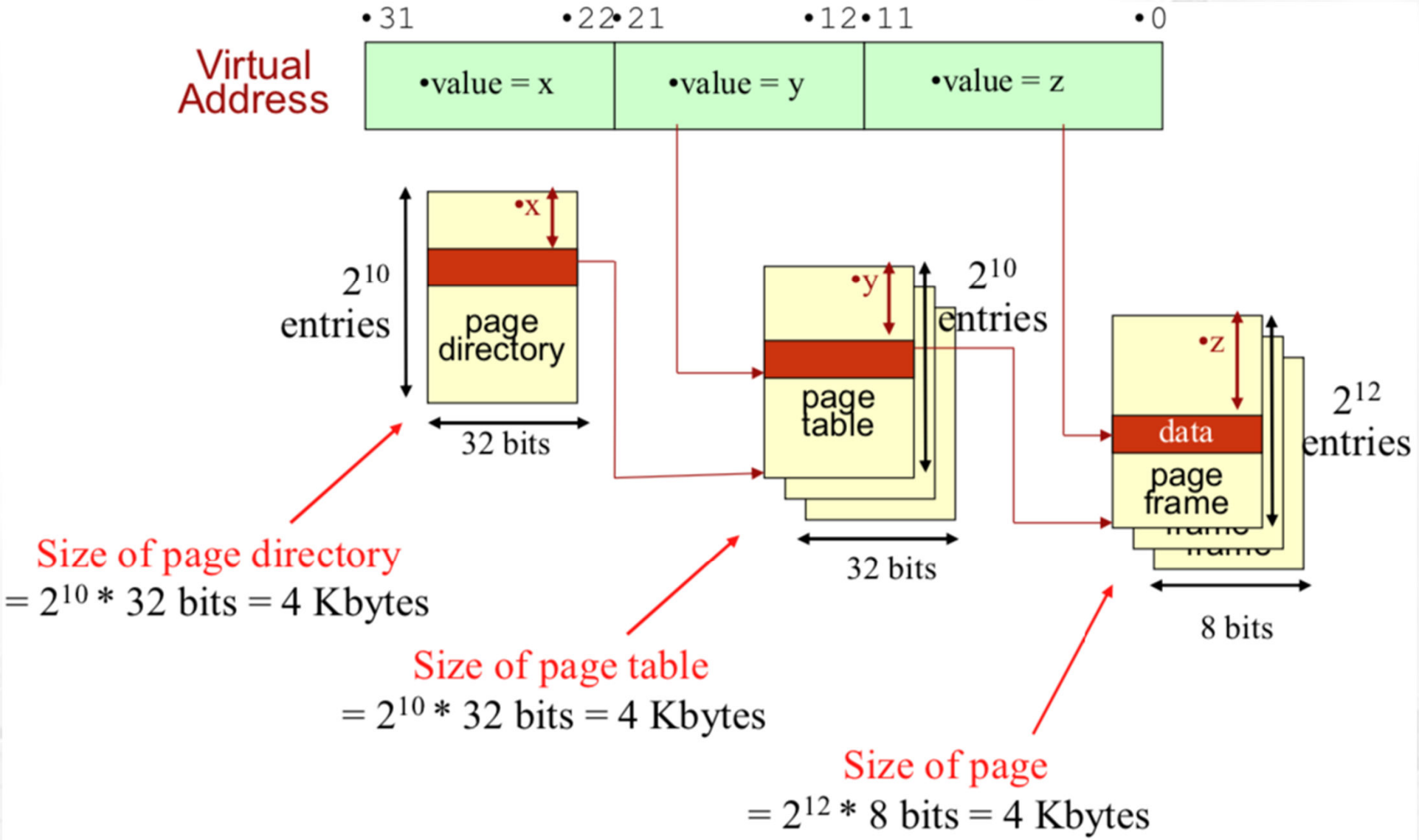
# VM模型下的内存访问

- **优化二：单极页表太大，怎么办?**



# VM模型下的内存访问

- **优化二：两级页表！**

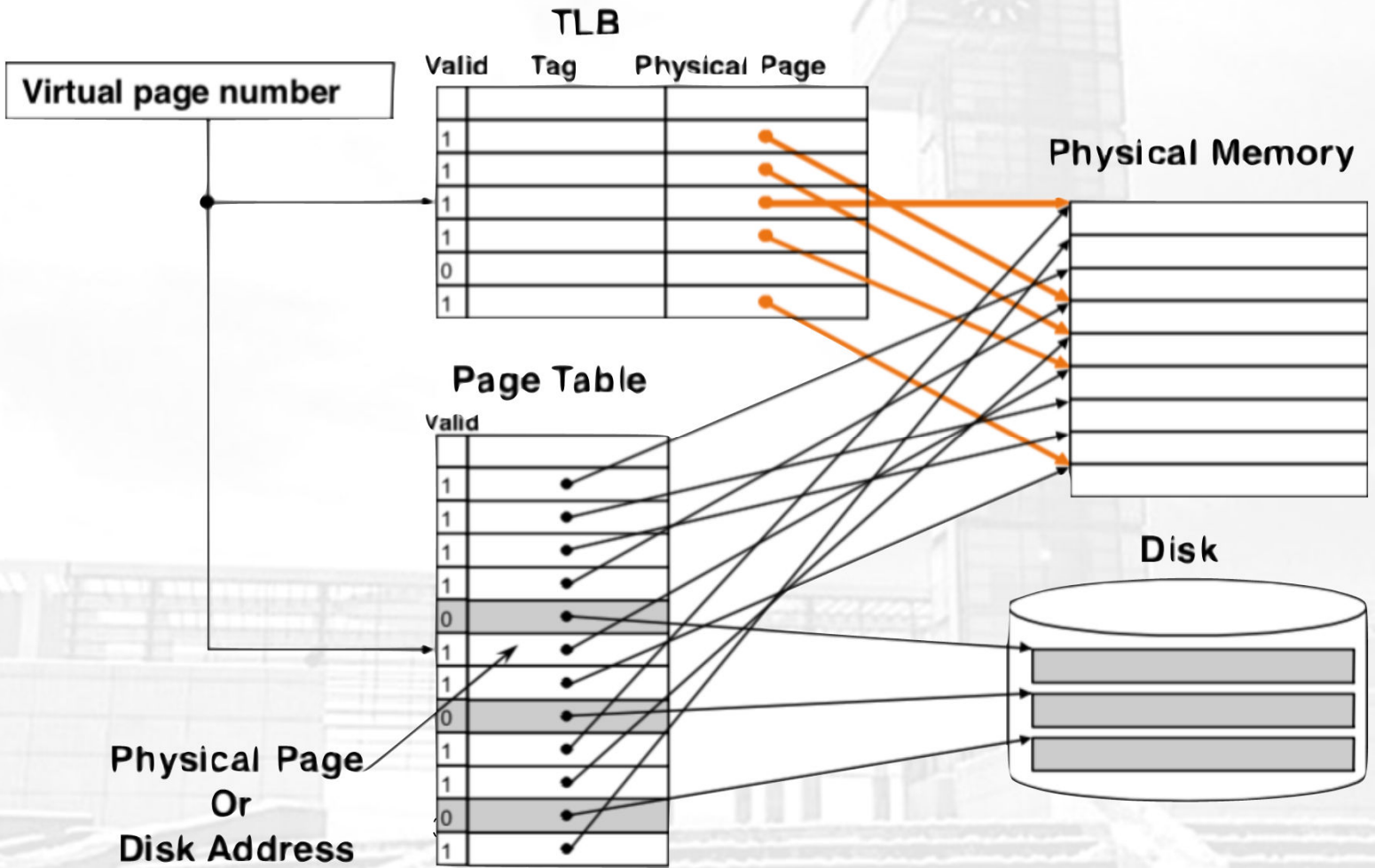



# VM模型下的内存访问

- **优化三：页表在内存访问慢，在Cache挤程序空间，怎么办?**

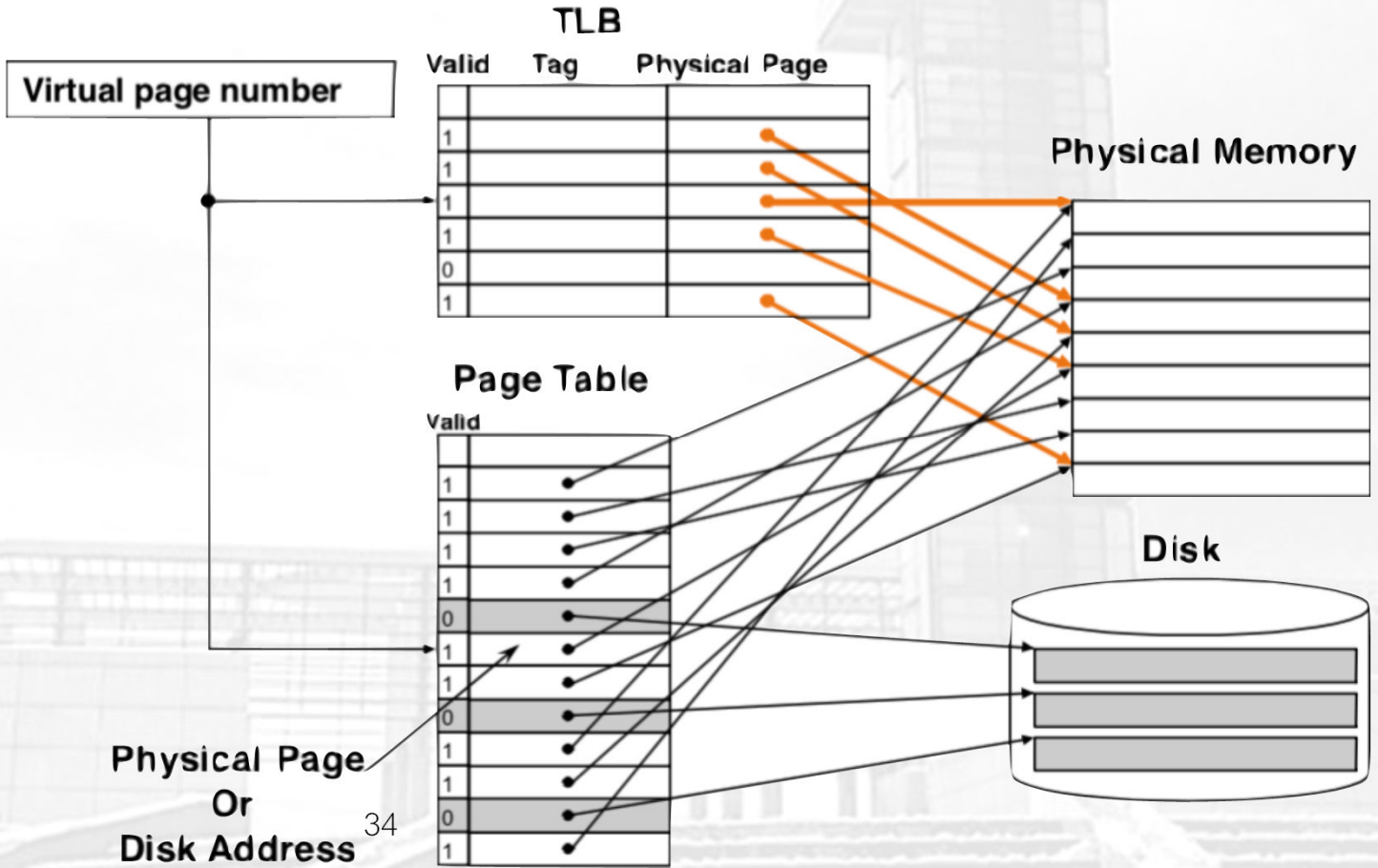
# VM模型下的内存访问

- 优化三：给页表加一个专用的Cache——TLB
  - Translation Look-aside Buffers
  - 比L1 Cache的级别更高！
    - 在CPU和Cache之间
  - 同样有hit rate/ miss rate的指标



# VM模型下的内存访问

- 优化三：给页表加一个专用的Cache——TLB
  - 因为TLB是个Cache…
    - TLB的映射方式：1/2/4/8路组相联
    - TLB的替换策略：随机、FIFO、LRU
    - TLB也搞成多级TLBs….
    - TLB又优化为I-TLB, D-TLB…



# VM模型下的内存访问

- **最终的内存访问流程~**