

# 计算机组成原理

翁睿

哈尔滨工业大学

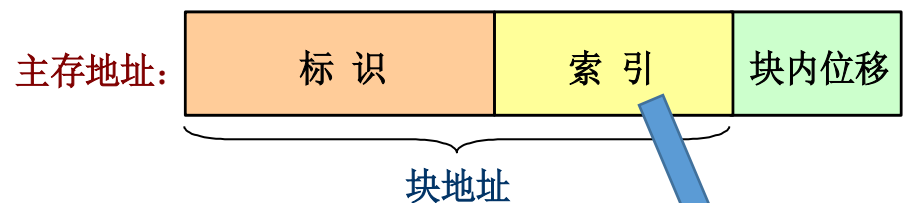
# 四、查找方法

4.3

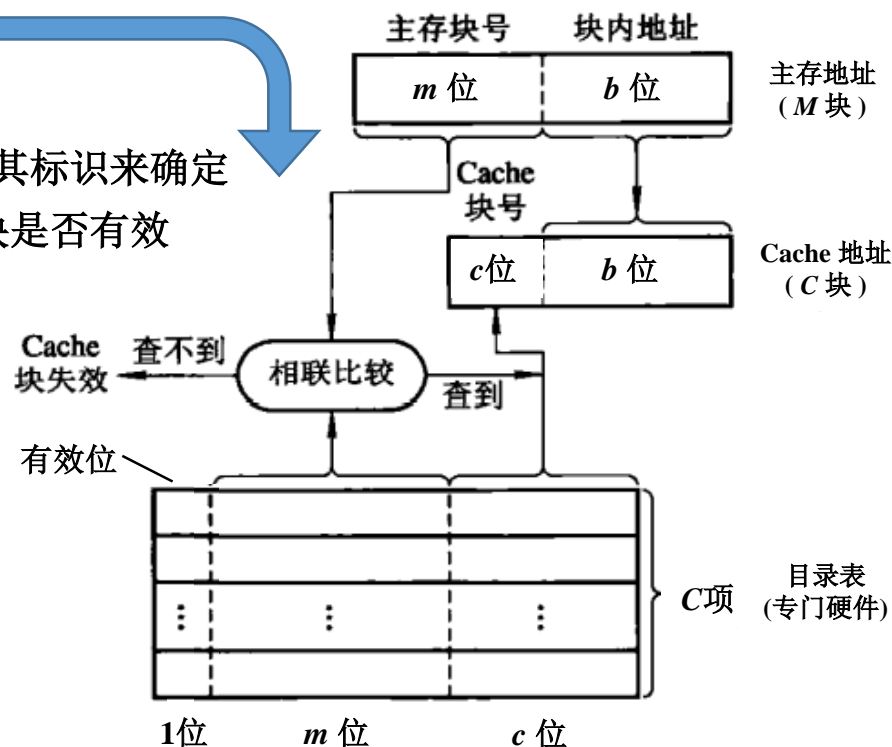
- 当CPU访问Cache时，如何确定Cache中是否有所要访问的块？
- 若有的话，如何确定其位置？
- 通过查找目录表来实现

- 目录表的结构

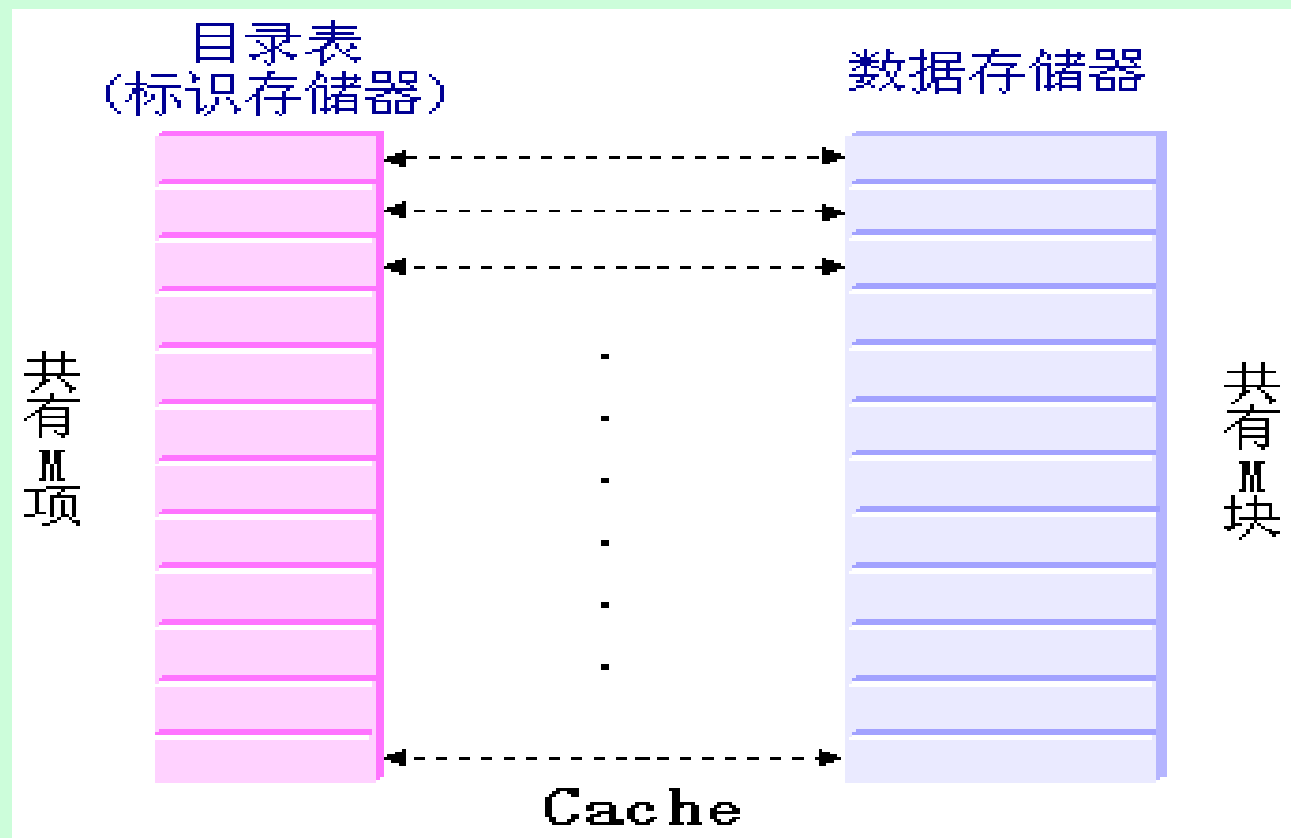
- 主存块的块地址的高位部分，称为标识
- 在候选位置内，每个主存块能唯一地由其标识来确定
- 每一项有一个有效位，指出Cache中的块是否有效



组相联: 只需查找候选位置  
所对应的目录表项



# Cache目录表的结构



目录表项:

有效位

标 识 tag

访存地址:

tag

index

## 二、地址映射

### 小结

成本高

不灵活

直接 某一主存块只能固定映射到某一缓存块

全相联 某一主存块能映射到任一缓存块

组相联 某一主存块能映射到某一缓存组中的任一块

## 三、替换算法

所要解决的问题：当新调入一块，而该块能够占用的Cache位置已被占满时，替换哪一块？

- 直接映象Cache中的替换很简单  
因为只有一个块，别无选择。
- 在组相联和全相联Cache中，则有多个块供选择。
  - 主要的替换算法有三种
    - 随机法  
优点：实现简单
    - 先进先出法FIFO
    - 最近最少使用法LRU

对于容量足够大的Cache，  
LRU和随机法的  
命中率差别不大。

## 四、写策略

## 4.3

### 4. 两种写策略

- ◆ **写直达法**：执行“写”操作时，不仅写入Cache，而且也写入下一级存储器。
- ◆ **写回法**：执行“写”操作时，只写入Cache。仅当Cache中相应的块被替换时，才写回主存。  
(设置“脏位”)

### 5. 两种写策略的比较

- ◆ **写回法的优点**：速度快，占用存储器频带低
- ◆ **写直达法的优点**：易于实现，一致性好

## 6. 写缓冲器

## 7. “写”操作时的调块

- ◆ **按写分配(写时取)**: 写失效时, 先把所写单元所在的块调入Cache, 再行写入。
- ◆ **不按写分配(绕写法)**: 写失效时, 直接写入下一级存储器而不调块。

## 8. 写策略与调块

写回法 —— 按写分配

写直达法 —— 不按写分配

### 3. 工作过程

- ① 处理器传送给Cache物理地址
- ② 由索引选择标识的过程
  - 根据索引从目录项中读出相应的标识和有效位
- ③ 从Cache中读出标识之后，用来同从CPU发来的块地址中标志域部分进行比较
  - 为了保证包含有效的信息，必须要设置有效位
- ④ 如果有一个标识匹配，且标志位有效，则此次命中
  - 通知CPU取走数据



- “写”访问命中

- 前三步一样，只有在确认标识匹配后才把数据写入

- 提高写入性能的方法：**设置写缓冲器**

(提高“写”访问的速度)

- 写缓冲器按字寻址，含有4个块，块大小为4个字。
  - 当要进行写入操作时，如果写缓冲器不满，那么就把数据和完整的地址写入缓冲器。对CPU而言，本次“写”访问已完成，CPU可以继续往下执行。由写缓冲器负责把该数据写入主存。
  - 在写入缓冲器时，要进行写合并检查。即检查本次写入数据的地址是否与缓冲器内某个有效块的地址匹配。如果匹配，就把新数据与该块合并。

# 发生读不命中与写不命中时的操作

## 4.3

- **读不命中：**向CPU发出一个暂停信号，通知它等待，并从下一级存储器中新调入一个数据块（32字节）
  - Cache与下一级存储的数据通路宽度为16B，传送一次需5个周期，因此，一次传送需要10个周期
- **写不命中：**将使数据“绕过”Cache，直接写入主存。
  - 写直达—不按写分配
- 因为是写直达，所以替换时不需要写回

## 六、改进Cache性能

4.3

平均访存时间 = 命中时间 + 失效率 × 失效开销

可以从三个方面改进Cache的性能：

### (1) 降低失效率

例如：增加块大小、提高相联度

### (2) 减少失效开销

例如：多级Cache、写缓存、请求字优先处理

### (3) 减少Cache命中时间

例如：容量小且结构简单的Cache

## 例1

- 某计算机字长32位，采用直接映射cache,主存容量4MB，cache数据存储体容量为4KB，字块长度为8个字。
1. 画出直接映射方式下主存地址划分情况。
  2. 设cache初始状态为空，若CPU顺序访问0-99号单元，并从中读出100个字，假设访问主存一次读一个字，并重复此顺序10次，请计算cache命中率。
  3. 如果cache存取时间是2ns，主存访问时间是20ns，平均访问时间是多少。
  4. cache-主存系统访问效率。

## 例1

- 某计算机字长32位，采用直接映射cache,主存容量4MB，cache数据存储体容量为4KB，字块长度为8个字。

1. 画出直接映射方式下主存地址划分情况。

解：主存字数： $4\text{MB}/4\text{B}=1\text{M}=2^{20}$ （20位地址）

Cache字数： $4\text{KB}/4\text{B}=1\text{K}=2^{10}$ （10位地址）

字块长度：8个字= $2^3$ （3位块内地址）

字块标记	块号	块内地址
10位	7位	3位

## 例1

字块标记	块号	块内地址
10位	7位	3位

2. 设cache初始状态为空，若CPU顺序访问0-99号单元，并从中读出100个字，假设访问主存一次读一个字，并重复此顺序10次，请计算cache命中率。

解： 0地址 → 0000 0000 00 00 0000 0 000

99地址 → 0000 0000 00 00 0110 0 011

地址范围覆盖块号0-12，共13个字块。

访问第一遍：发生13次调块操作，填充缓存，其余 $100-13=87$ 次均命中。

访问剩余9遍：全部命中，共 $100 \times 9 = 900$ 次。

例1

字块标记	块号	块内地址
10位	7位	3位

4.3

2. 设cache初始状态为空，若CPU顺序访问0-99号单元，并从中读出100个字，假设访问主存一次读一个字，并重复此顺序10次，请计算cache命中率。

解：

根据命中率的定义：

$$H=(87+900)/(100 \times 10) \times 100\%=98.7\%$$

例1

字块标记	块号	块内地址
10位	7位	3位

3. 如果cache的存取时间是2ns，主存访问时间是20ns，平均访问时间是多少

解：

根据平均访问时间的定义：

$$\begin{aligned}T_a &= H \times t_c + (1-H) \times t_m \\&= 98.7\% \times 2 + (1-98.7\%) \times 20 \\&= 2.234\text{ns}\end{aligned}$$



例1

字块标记	块号	块内地址
10位	7位	3位

4. cache-主存系统的访问效率

解:

根据Cache-主存系统访问效率的定义:

$$\begin{aligned} e &= t_c / T_a \\ &= 2\text{ns} / 2.234\text{ns} \\ &= 0.895 \end{aligned}$$

【2016 统考真题】有如下 C 语言程序段：

```
for(k=0; k<1000; k++)
```

```
    a[k] = a[k] + 32;
```

若数组 a 和变量 k 均为 int 型，int 型数据占 4B，数据 Cache 采用直接映射方式，数据区大小为 1KB、块大小为 16B，该程序段执行前 Cache 为空，则该程序段执行过程中访问数组 a 的 Cache 缺失率约为

A. 1.25%

B. 2.5%

C. 12.5%

D. 25%

解：

字块标记

块号

块内地址

若干位

6位

2位

1KB=64个块    1块=4字    共256个字

0-999: 00 00 00 00 00 ~ 11 11 10 01 11

读缺失：每圈64次/256次访存≈0.25

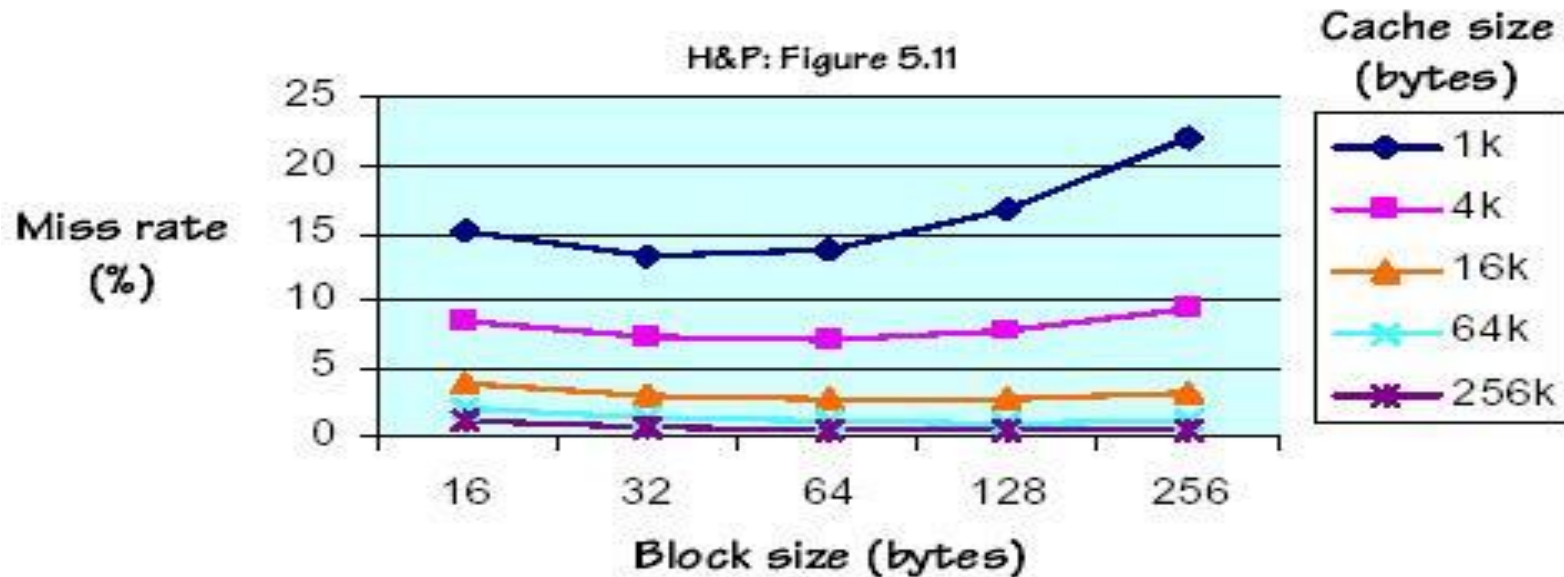
写缺失：每圈0次/256次访存=0

总 共：每圈64次/512次访存≈0.125

# Cache大小、Block大小和缺失率的关系

## 4.3

### Block size vs. miss rate



- spatial locality: larger blocks → reduce miss rate
- fixed cache size: larger blocks  
→ fewer lines in cache  
→ higher miss rate, especially in small caches

**Cache大小:** Cache越大, Miss率越低, 但成本越高!

**Block大小:** Block大小与Cache大小有关, 且不能太大, 也不能太小!

# Cache系统的设计

## 4.3

- 刚引入Cache时只有一个Cache。近年来多Cache系统成为主流
- 多Cache系统中，需考虑两个方面：

### [1] 单级/多级？

**外部(Off-chip)Cache:**不做在CPU内而是独立设置一个Cache

**片内(On-chip)Cache:** 将Cache和CPU作在一个芯片上

**单级Cache:** 只用一个片内Cache

**多级Cache:** 同时使用L1 Cache和L2 Cache，甚至有L3 Cache，

L1 Cache更靠近CPU，其速度比L2快，其容量比L2小

### [2] 联合/分立？

**分立:** 指数据和指令分开存放在各自的数据和指令Cache中

**一般L1 Cache都是分立Cache**

**L1 Cache的命中时间比命中率更重要！**

**联合:** 指数据和指令都放在一个Cache中

**一般L2 Cache都是联合Cache**

**L2 Cache的命中率比命中时间更重要！**

- 采用L2 Cache的系统，其缺失损失的计算如下：
  - 若L2 Cache包含所请求信息，则缺失损失为L2 Cache访问时间
  - 否则访问主存，并取到L1 Cache和L2 Cache（缺失损失更大）
- 例：某处理器在无cache缺失时CPI为1，时钟频率为5GHz。假定访问一次主存的时间（包括所有的缺失处理）为100ns，平均每条指令在L1 Cache中的缺失率为2%。若增加一个L2 Cache，其访问时间为5ns，而且容量足够大到使全局缺失率减为0.5%，问处理器执行指令的速度提高了多少？

解：如果只有一级Cache，则缺失只有一种。即L1缺失(需访问主存)，其缺失损失为： $100\text{ns} \times 5\text{GHz} = \underline{500\text{个时钟}}$ ， $\text{CPI} = 1 + 500 \times 2\% = 11.0$ 。

如果有二级Cache，则有两种缺失：

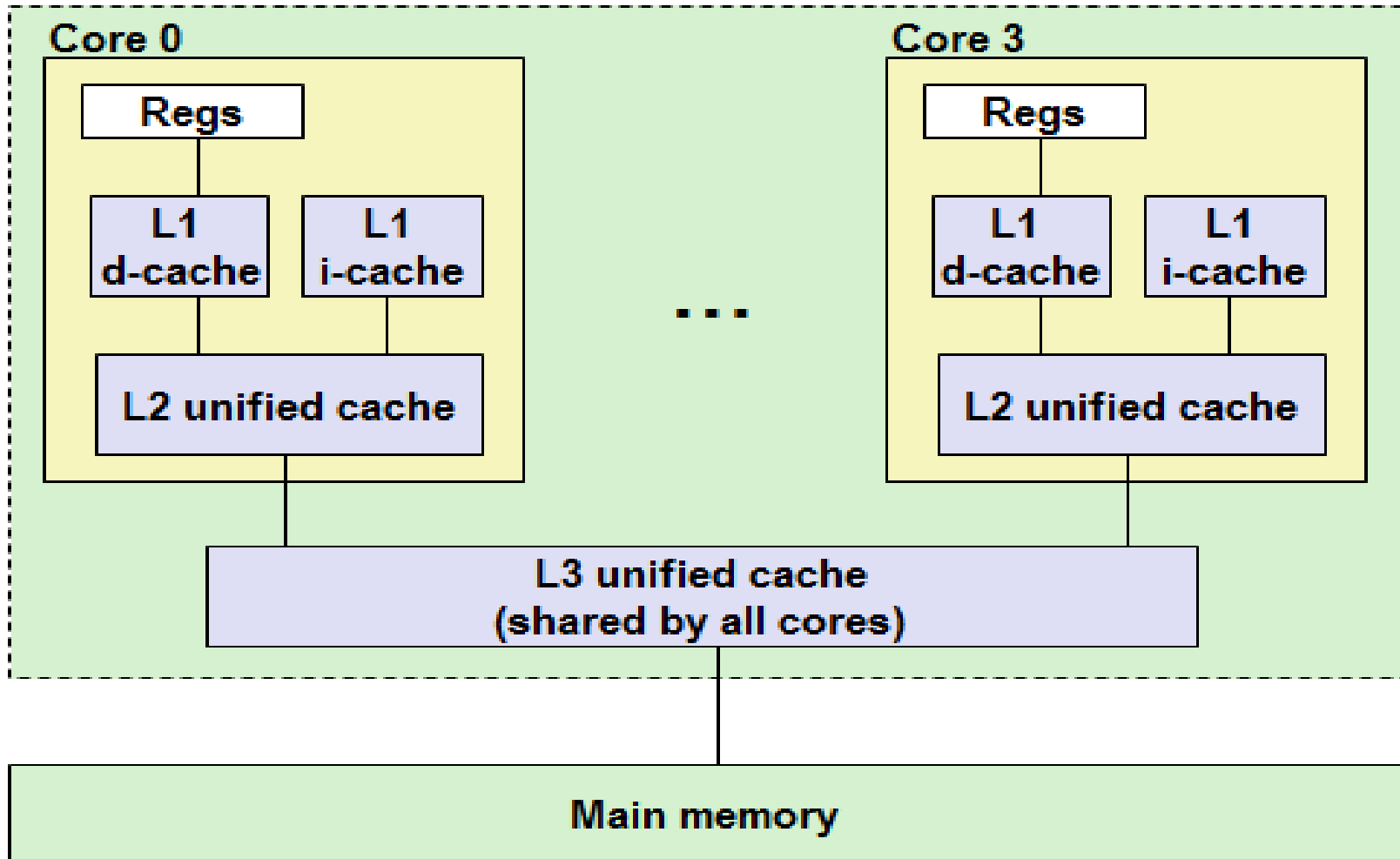
L1缺失(需访问L2 Cache)： $5\text{ns} \times 5\text{GHz} = \underline{25\text{个时钟}}$

L1和L2都缺失(需访问主存)：500个时钟

因此， $\text{CPI} = 1 + \underline{25} \times 2\% + \underline{500} \times 0.5\% = 4.0$

二者的性能比为 $11.0/4.0 = 2.8$ 倍！

# 实例：Intel Core i7处理器的cache结构 4.3




i-cache和d-cache都是32KB、8路、4个时钟周期；L2 cache：256KB、8路、11个时钟周期。所有核共享的L3 cache：8MB、16路、30~40个时钟周期。Core i7中所有cache的块大小都是64B

# 实例：Intel Core i7处理器的cache结构 4.3

hwinfo64检测图：QL3X = Intel i7-7820HK (ES)

CPU



CPU #0

核心

4 / 8

-

Intel Core -2400	14 nm
Stepping A0	TDP 45 W
代码名称 Kaby Lake-H	MCU 34
QDF QL3X (ES)	工程样品
平台 BGA1440	
Cache L1 4x32 + 4x32	L2 4x256
	L3 8M
	L4

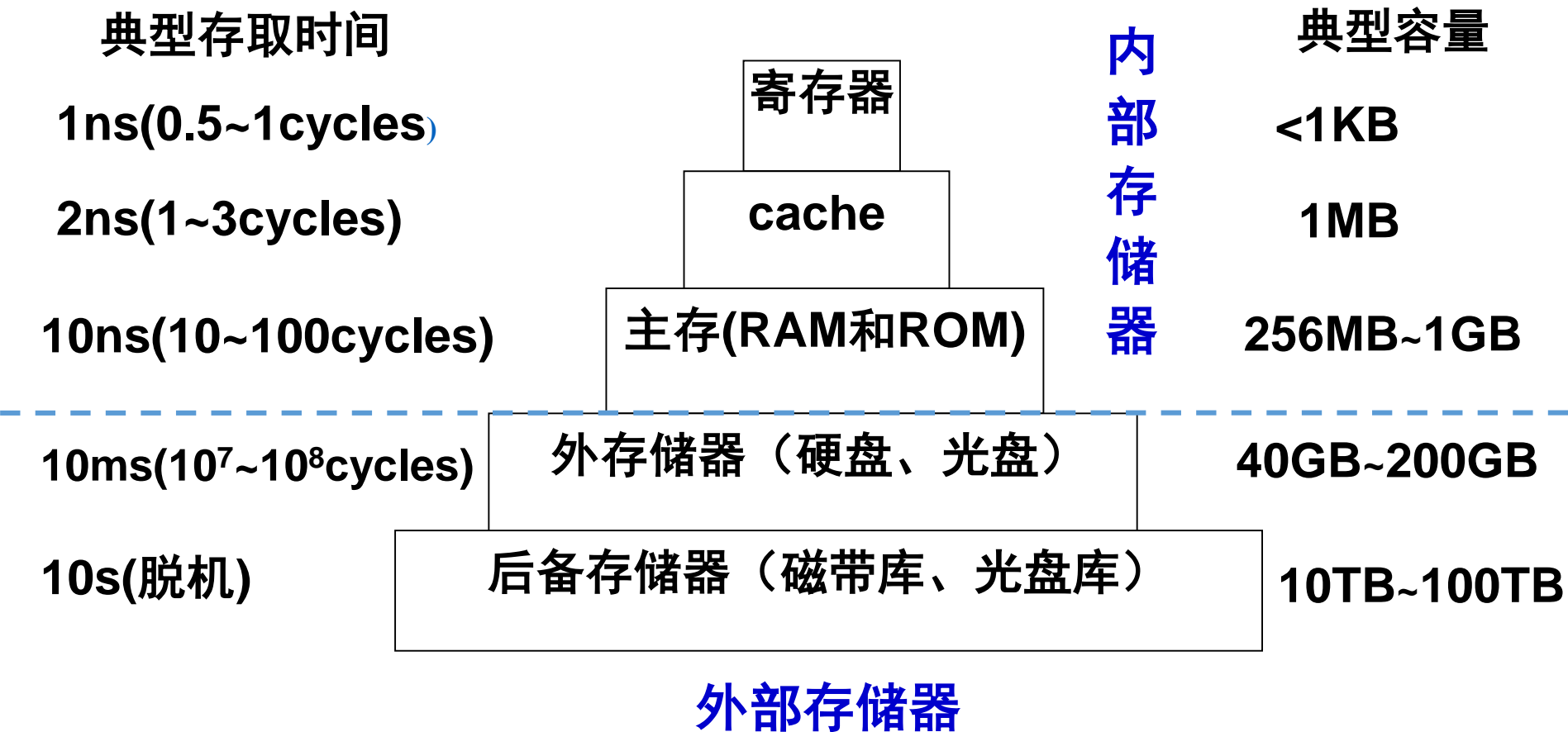
i-cache和d-cache都是32KB、8路、4个时钟周期；L2 cache：256KB、8路、11个时钟周期。所有核共享的L3 cache：8MB、16路、30~40个时钟周期。Core i7中所有cache的块大小都是64B

## 4.4 虚拟存储器

- 虚拟存储器概述
- 页式虚拟存储器及页表管理
- 虚拟存储器地址映射与变换
- 虚拟存储器举例



# 存储器的层次结构



列出的时间和容量会随时代变化，但数量级相对关系不变。

# 虚拟存储器

- 比尔·盖茨(1981) “无论对谁来说, 640K内存都足够了”
- 如何在有限的主存空间运行较多的较大的的用户程序?
  - 初衷: 采用虚存来扩大寻址空间
  - 现在: 主要用于存储保护, 程序共享
- 主存、辅存在OS和硬件的管理之下, 提供更大的存储空间
- 虚存空间是靠辅存(磁盘)来支持的
- 虚拟存储采用与cache类似原理, 提高速度, 降低成本
- 用户感觉到的是一个速度接近主存而容量极大的存储器

# 虚拟存储系统的基本概念

- 虚拟存储技术的引入用来解决一对矛盾：

有限的主存空间 ⇔ 运行较多的较大的用户程序

- 虚拟存储技术的实质

- 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(缺页)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**和**操作系统**共同协作实现

# 虚拟存储系统的基本概念

由于虚拟存储器机制由硬件和操作系统共同协作实现，其中涉及到操作系统中的许多概念，如：

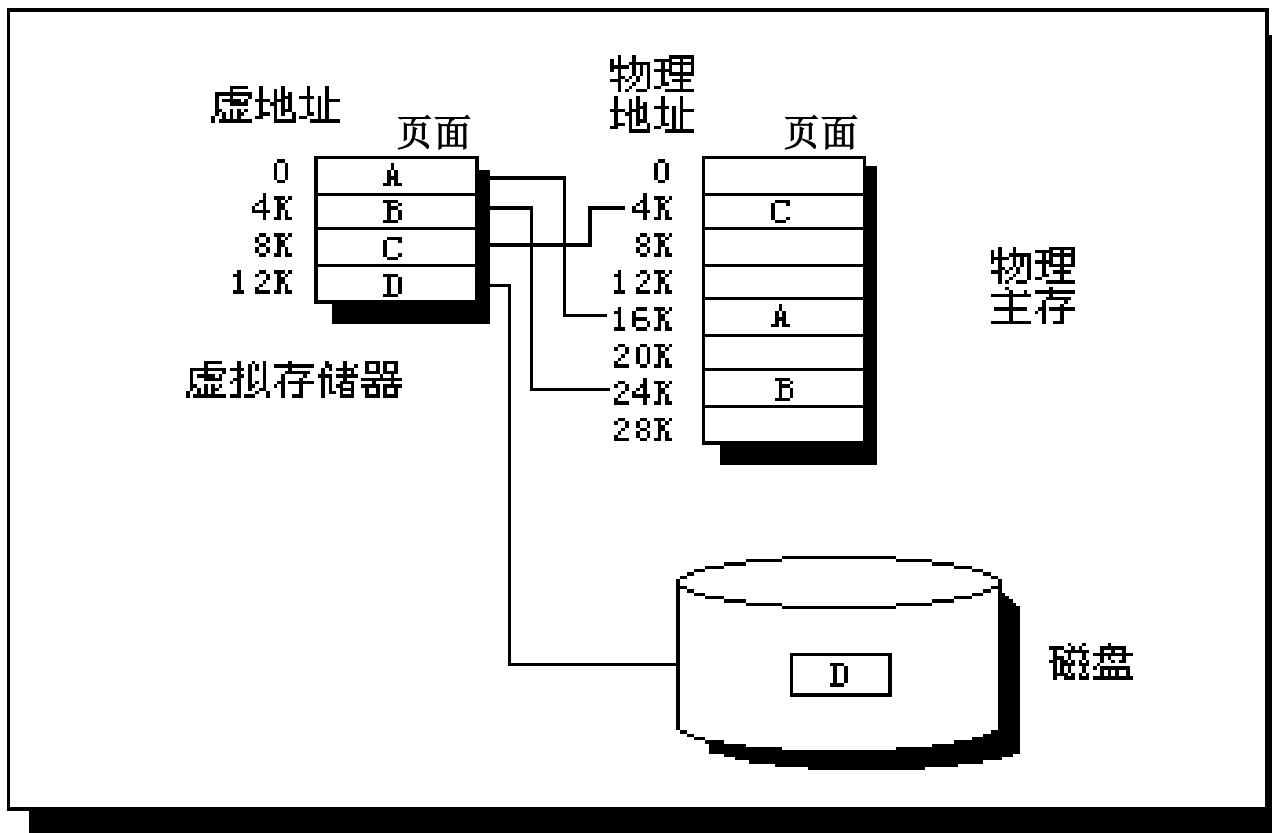
- 进程
- 进程的上下文切换
- 存储器分配
- 虚拟地址空间
- 缺页处理
- .....

# 虚拟存储器基本原理

## 4.4

### 虚拟存储器的特点

- ◆程序员可以利用巨大的逻辑空间，而不必做存储管理工作
- ◆多个进程可以共享主存空间
- ◆采用动态再定位，简化了程序的装入



# 虚拟存储器基本原理

## 4.4

### 虚存管理方式--页式管理

- 页式虚存把主存与外存均划分成等长的页面。常用页大小为**4KB~64KB**。
- 便于维护，便于生成页表，类似于**cache**的块表
- 不容易产生碎块，存储空间浪费小
- 页不是独立的实体，处理、保护和共享不方便

# 虚拟存储器基本原理

## 4.4

### 虚存管理方式--段式管理

- 段式虚存把空间划分为可变长的块，称为段，段的分界与程序的自然分界相对应，段最小长度为1个字节，最大因机器而异，常为 $2^{16}\text{B} \sim 2^{32}\text{B}$ 。
- 段的独立性—易于编译、管理、修改和维护，也便于多道程序共享
- 各段的长度不同，给主存的分配带来麻烦
- 段式管理容易产生碎块，浪费主存空间

# 虚拟存储器基本原理

## 4.4

### 虚存管理方式--段页式管理

- 程序按模块分段
- 段再分成长度固定的页。
- 程序调入调出按页面来进行
- 程序共享保护按段进行
- 兼备段式，页式管理的优点
- 在地址映像中需要多次查表



# “Cache—主存”与“主存—辅存”层次的区别

## 4.4

存储层次 比较项目	“Cache —主存” 层次	“主存—辅存” 层次
目 的	帮助主存提升速度	帮助主存扩展容量
存储管理实现	主要由专用硬件实现	主要由软件实现
访问速度的比值 (第一级和第二级)	几比一	几百比一
典型的块(页)大小	几十个字节	几百到几千个字节
CPU对第二级的 访问方式	可直接访问	均通过第一级
失效时CPU是否切换	不切换	切换到其他进程

参数	一级cache	虚拟存储器
块（页）大小	16~128字节	4096~65536字节
命中时间	1~3个时钟周期	50~150个时钟周期
失效开销	8~150个时钟周期	$10^6 \sim 10^7$ 个时钟周期
失效率	0.1%~10%	0.00001%~0.001%
地址映射关系	25~45位物理地址映射到14~20位cache地址	32~64位虚地址映射到25~45位物理地址

# 有关虚拟存储器的四个问题

## 4.4

### ◆映射规则

**全相联。**以降低失效率为主要目标。

### ◆查找算法

**页表，段表，TLB。**

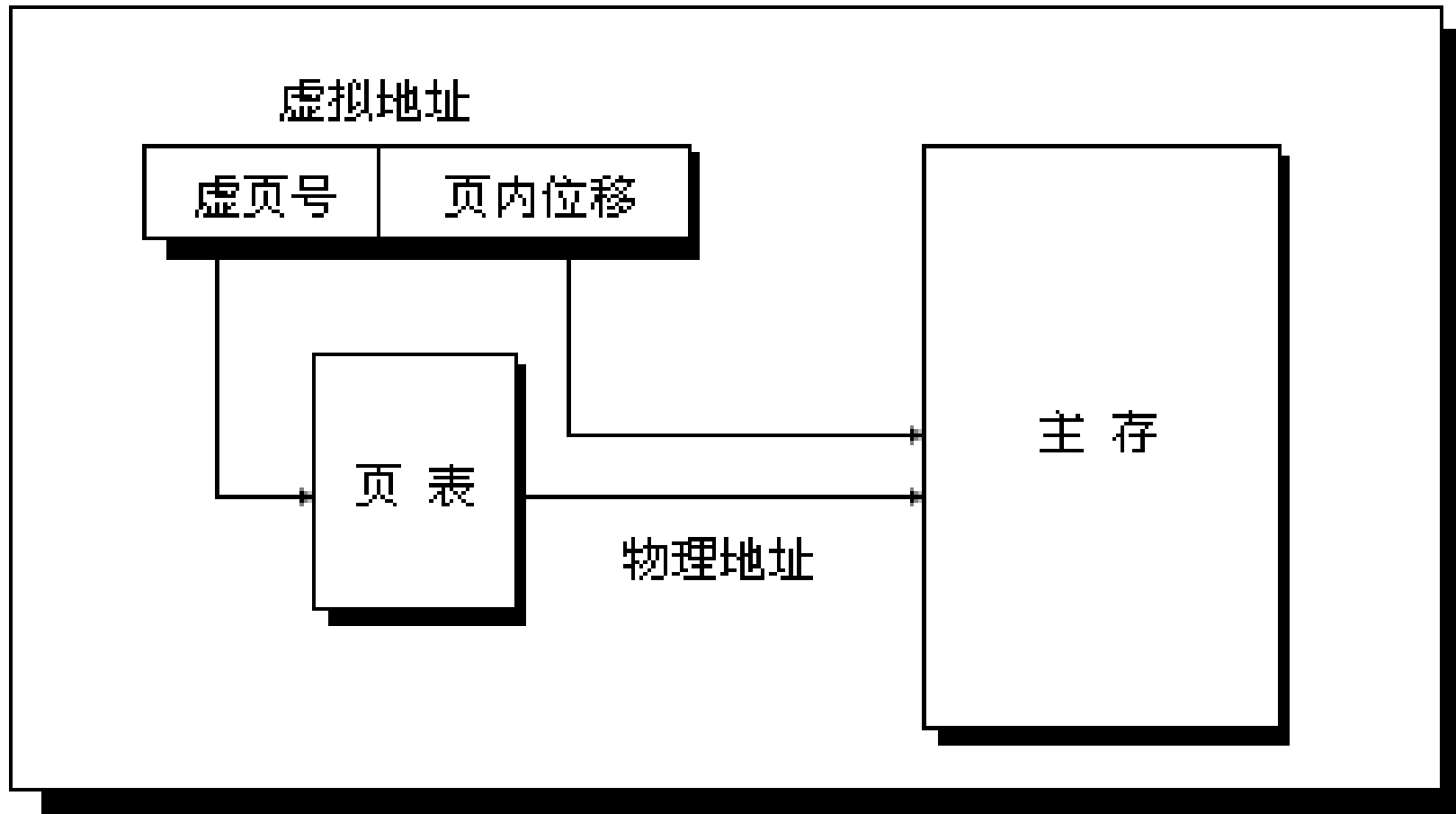
页表和段表：索引页号或段号的数据结构，含有所要查找的块的物理地址。

段式系统：段内位移 加 段的物理地址 就是最终的物理地址。

页式系统：只需简单地将页内位移拼接在相应页面的物理地址之后。

# 页表

4.4



# 有关虚拟存储器的四个问题

## 4.4

### ◆ 替换算法

LRU等。

尽可能减少页故障，OS为每个页面设置“使用位”。

### ◆ 写策略

写回法

## 4.4 虚拟存储器

- 虚拟存储器概述
- 页式虚拟存储器及页表管理
- 虚拟存储器地址映射与变换

# 分页（Paging）

## 4.4

- 基本思想：
  - 内存被分成固定长且比较小的存储块（页框、实页、物理页）
  - 每个进程也被划分成固定长的程序块（页、虚页、逻辑页）
  - 程序块可装到存储器中可用的存储块中
  - 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）

# 分页（Paging）

## 4.4

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）：
  - 程序中指令所用地址(进程所在地址空间)，也称为**虚拟地址（Virtual Address，简称VA）**
- 物理地址（Physical Address，简称PA）：
  - 存放指令或数据的实际内存地址，也称为**实地址、主存地址。**



# 分页 (Paging)

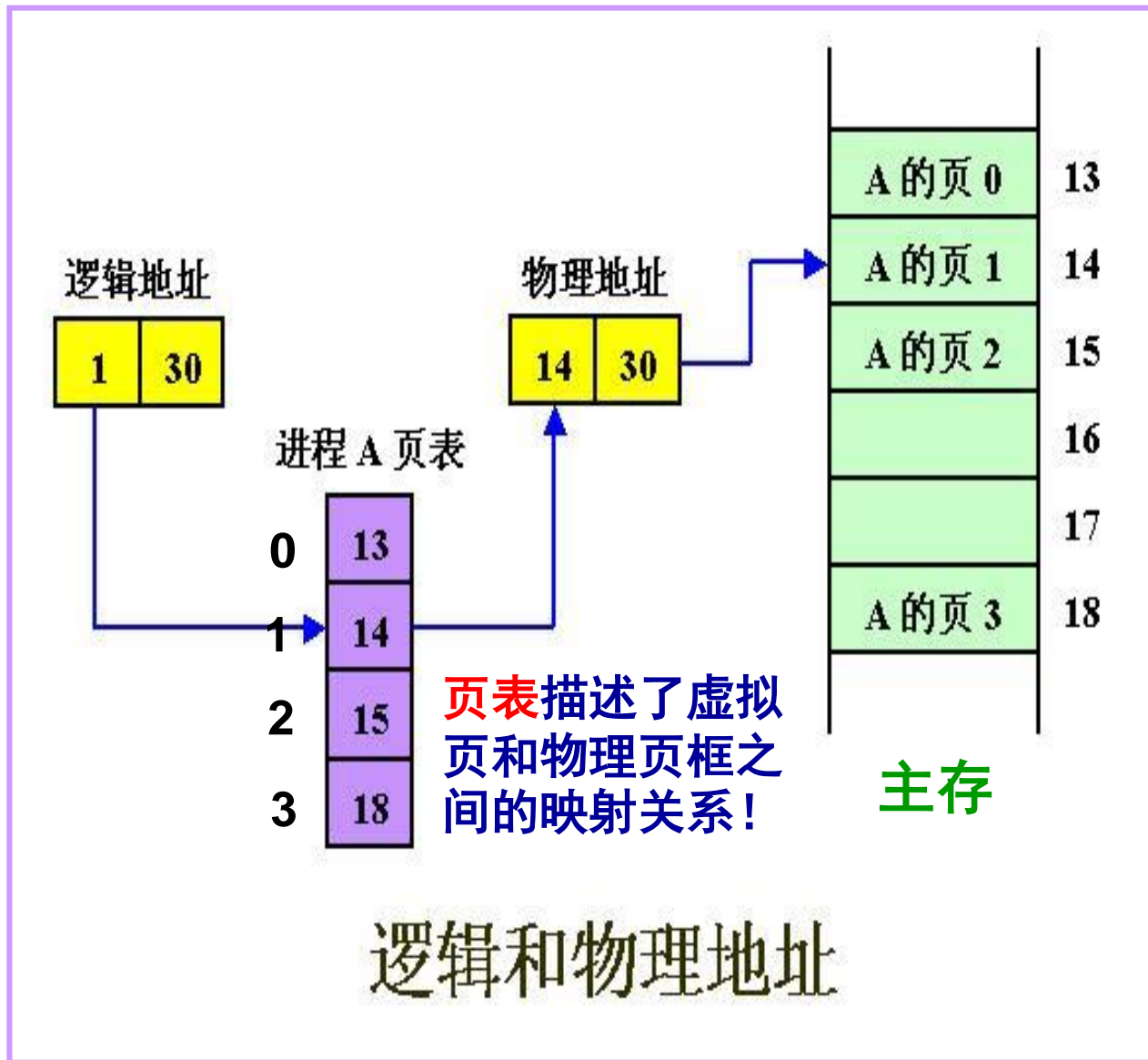
## 4.4

**问题：是否需要将一个进程的全部页面都装入内存？**

**根据程序访问局部性可知：可把当前活跃的页面调入主存，其余留在磁盘上！**

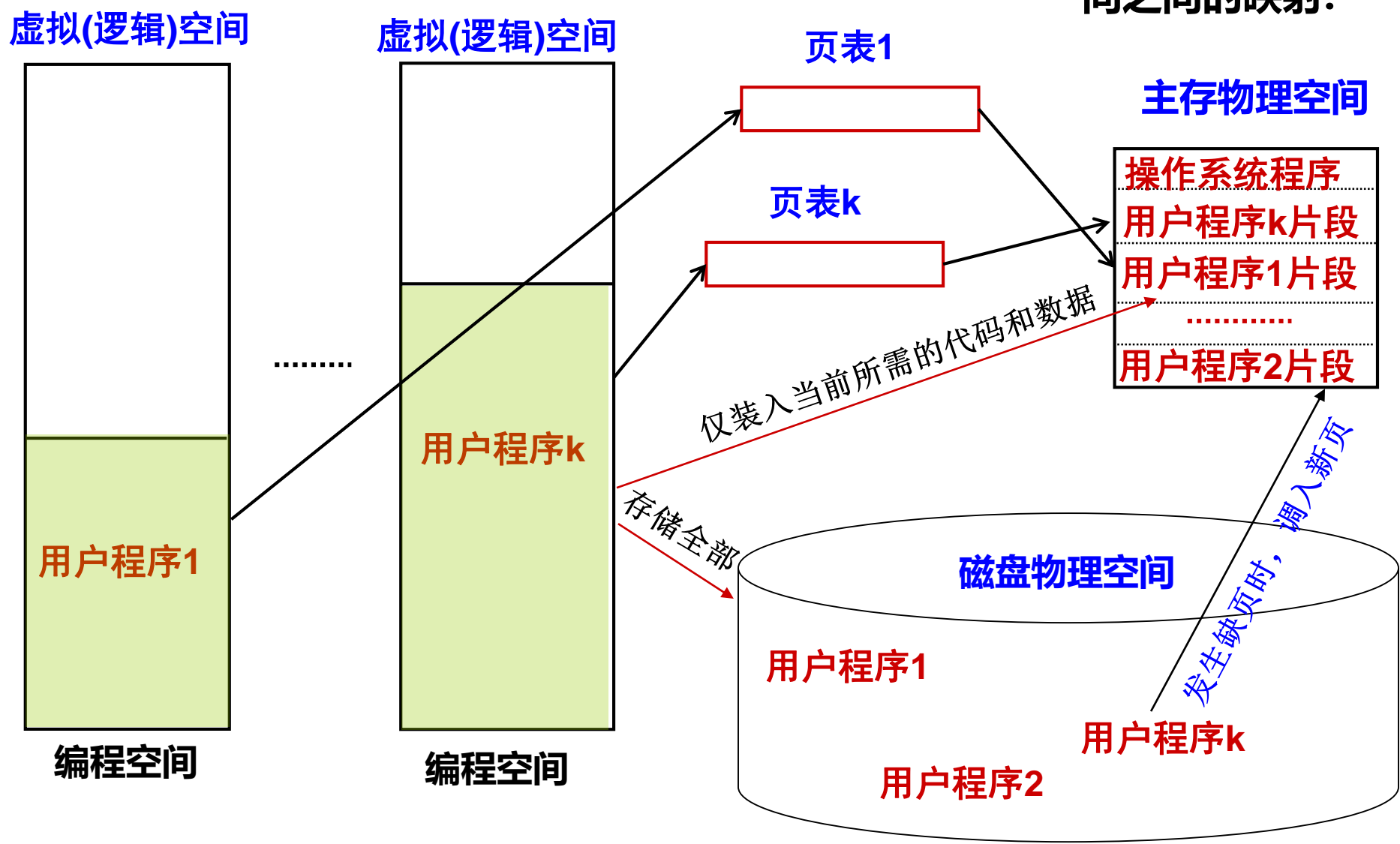
**采用“按需调页 Demand Paging”方式分配主存！这就是虚拟存储管理概念**

**优点：浪费的空间最多是最后一页中的一部分！**



# 页表--虚拟空间与物理空间的映射

通过页表建立虚拟空间和物理空间之间的映射!



# 虚拟地址空间

- Linux在X86上的虚拟地址空间

(其他Unix系统的设计类此)

- 内核空间 (Kernel)
- 用户栈 (User Stack)
- 共享库 (Shared Libraries)
- 堆 (heap)
- 可读写数据 (Read/Write Data)
- 只读数据 (Read-only Data)
- 代码 (Code)

**问题：程序加载时是否真正从磁盘调入信息到主存？**

**按需调页**

实际上不会从磁盘调入，只是将虚拟页和磁盘上的数据/代码建立对应关系，称为“映射”。

0xC000 0000

0x08048000

0

Kernel virtual memory

User stack  
(created at runtime)

Memory-mapped region  
for shared libraries

Run-time heap  
(created by malloc)

Read/write segment  
(.data, .bss)

Read-only segment  
(.init, .text, .rodata)

Unused

1GB

%esp  
(栈顶)

brk

可执行  
文件相  
关内容  
“复制”  
到代码  
段和数  
据段

128MB

# 程序和数据的存储器分配

(基于MIPS处理器)

## 4.4

问题：你知道一个程序在“编辑、编译、汇编、链接、装入”过程中的哪个环节确定了每条指令及其操作数的虚拟地址吗？

- 链接时确定虚拟地址；
- 装入时生成页表以建立虚拟地址与物理地址之间的映射！

请参考《深入理解计算机系统》，和有关Linux内核分析方面的资料。

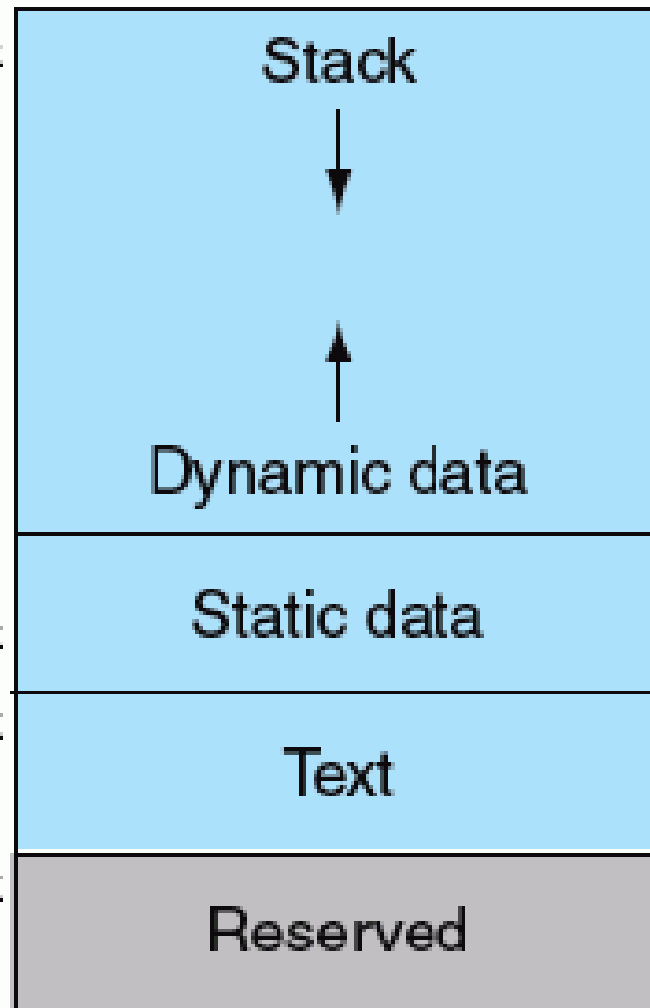
\$sp → 7fff ffff<sub>hex</sub>

\$gp → 1000 8000<sub>hex</sub>

1000 0000<sub>hex</sub>

pc → 0040 0000<sub>hex</sub>

0



每个用户程序都有相同的  
虚拟地址空间！

这就是每个进程的虚拟（逻辑）地址空间！

### ◆ 页表首地址记录在页表基址寄存器中

页表首地址



	装入位	修改位	替换控制位	其他	实页号 (8 进制)
0 虚页	1				11
1 虚页	1				13
2 虚页	1				16
3 虚页	1				10
4 虚页	1				14

- 每个进程有一个页表，其中有装入位、修改（Dirt）位、替换控制位、访问权限位、禁止缓存位、实页号。

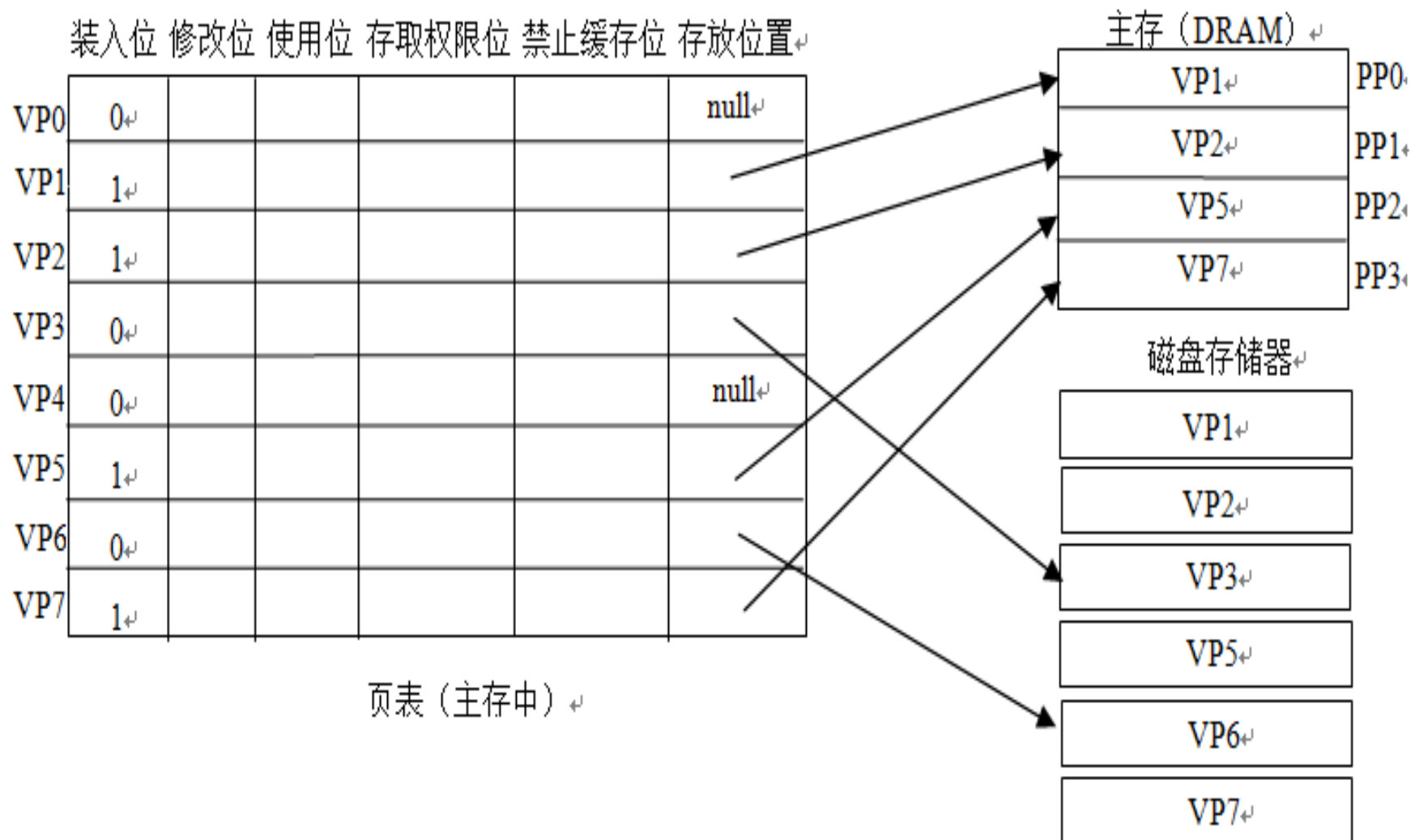
- 一个页表的项数由什么决定？

理论上由虚拟地址空间大小决定。

- 每个进程的页表大小一样吗？

各进程有相同虚拟空间，故理论上一样。实际大小看具体实现方式，如“空洞”页面如何处理等

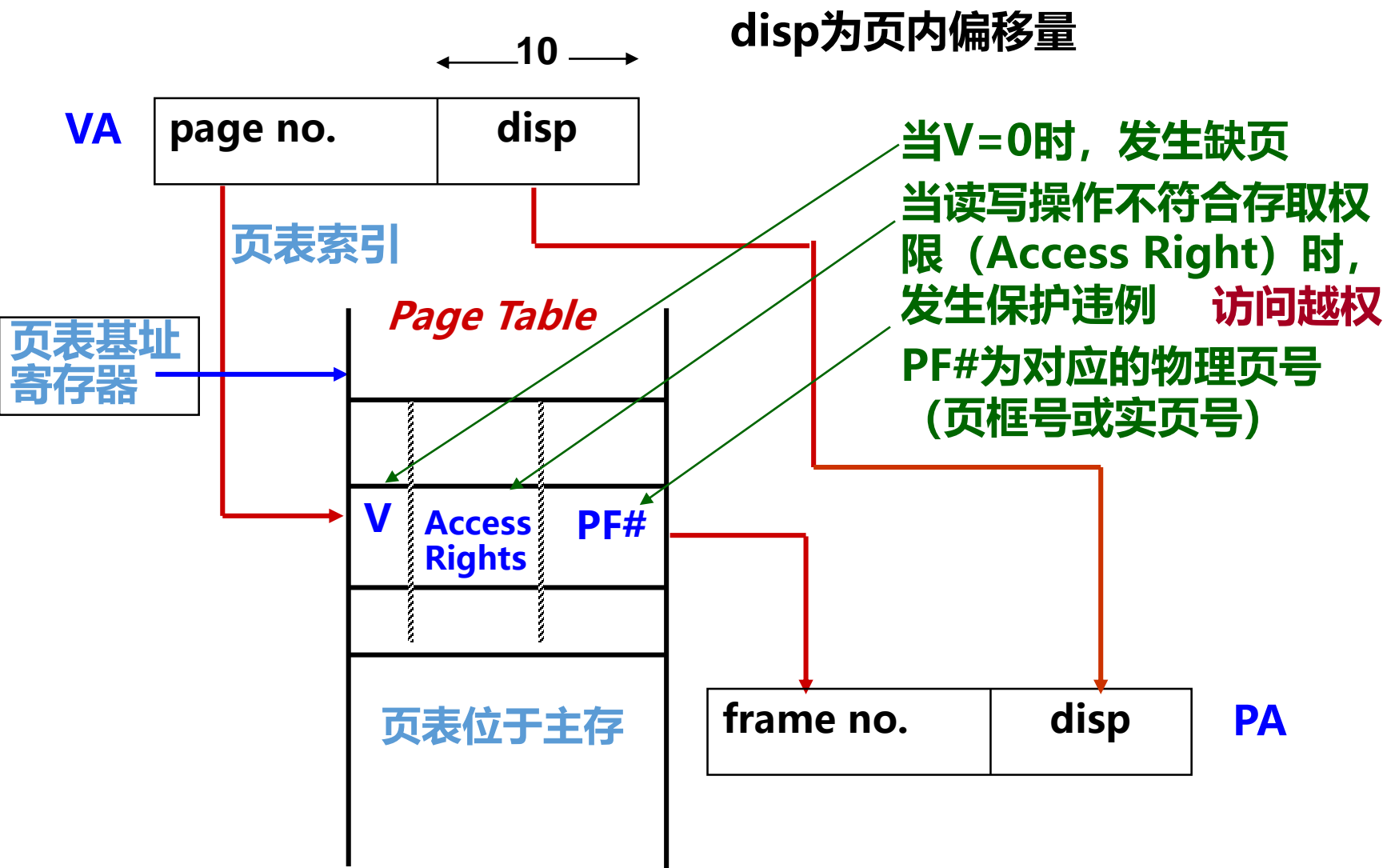
# 主存中的页表示例



- ◆ **未分配页**: 进程的虚拟地址空间中 “空洞”对应的页 (如VP0、VP4)
- ◆ **已分配的缓存页**: 有内容对应的已装入主存的页 (如VP1、VP2、VP5等)
- ◆ **已分配的未缓存页**: 有内容对应但未装入主存的页 (如VP3、VP6)

# 逻辑地址转换为物理地址的过程

4.4



# 信息访问中可能出现的异常情况

## 4.4

可能有两种异常情况：

### 1) 缺页 ( page fault)

**产生条件：**当Valid (有效位 / 装入位) 为 0 时

**相应处理：**从磁盘读到内存，若内存没有空间，则还要从内存选择一页替换到磁盘上，替换算法类似于Cache，采用回写法，淘汰时，根据“dirty”位确定是否要写磁盘

当前指令执行被阻塞，当前进程被挂起，处理结束回到原指令继续执行

### 2) 保护违例 ( protection\_violation\_fault ) 或访问违例

**产生条件：**当Access Rights (存取权限)与所指定的具体操作不相符时

**相应处理：**在屏幕上显示“内存保护错”或“访问违例”信息

当前指令的执行被阻塞，当前进程被终止



0x00007FF7E25836C9 指令引用了 0x0000000000000000 内存。该内存不能为 read。

**Access Rights (存取权限)可能的取值有哪些？**

R = Read only, R/W = read/write, X = execute only

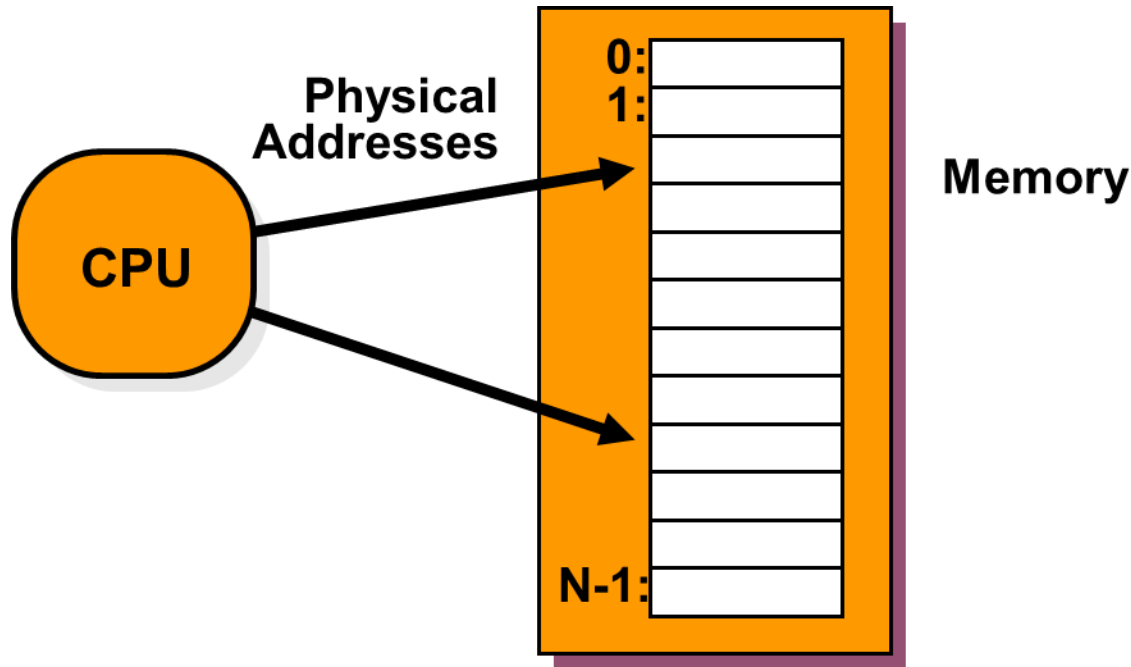


## 4.4 虚拟存储器

- 虚拟存储器概述
- 页式虚拟存储器及页表管理
- 虚拟存储器地址映射与变换

# 实地址计算机系统

- **CPU地址：物理内存地址**
  - 大多数Cray计算机，早期PC，大多数嵌入式系统

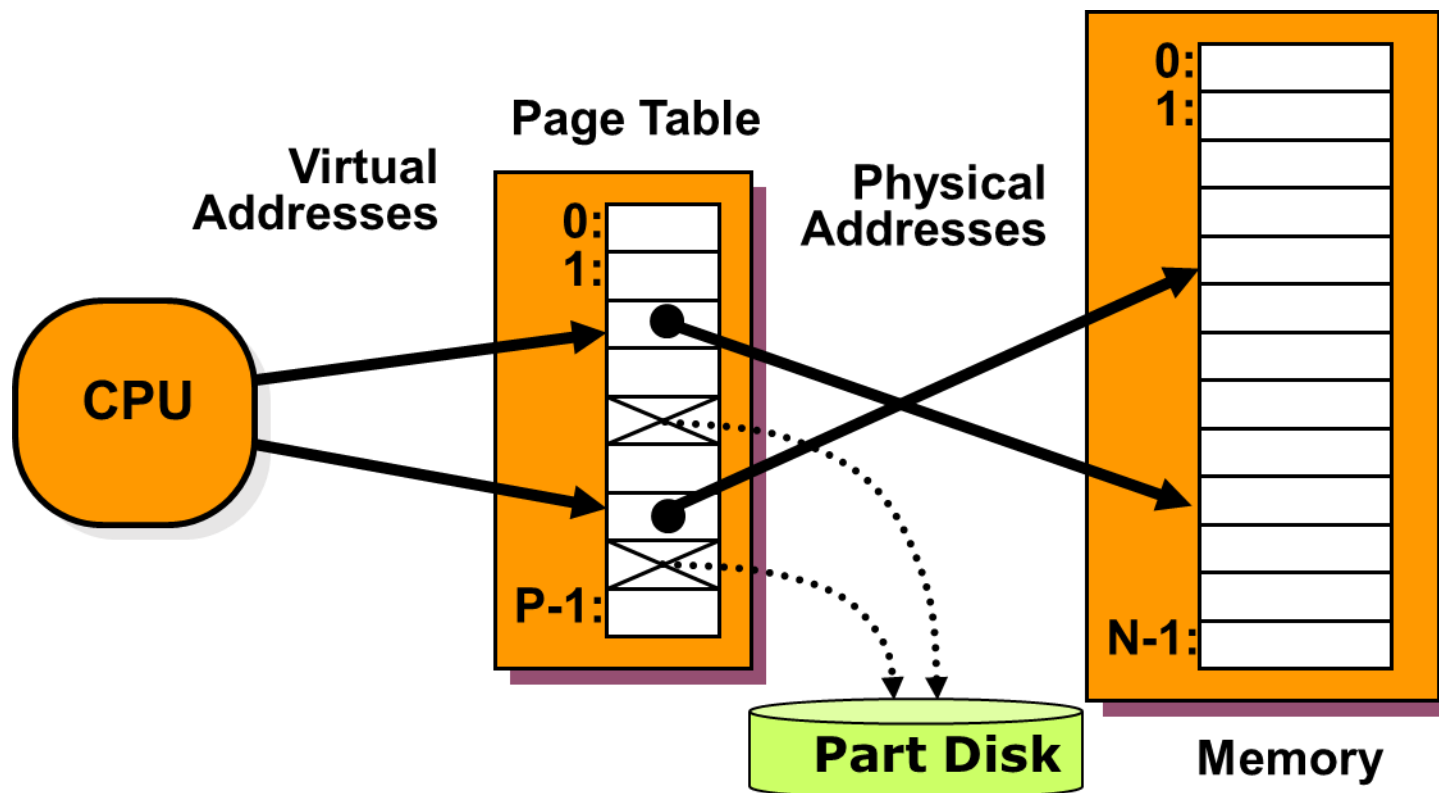


# 虚地址计算机系统

## 4.4

- **CPU地址：虚拟地址**

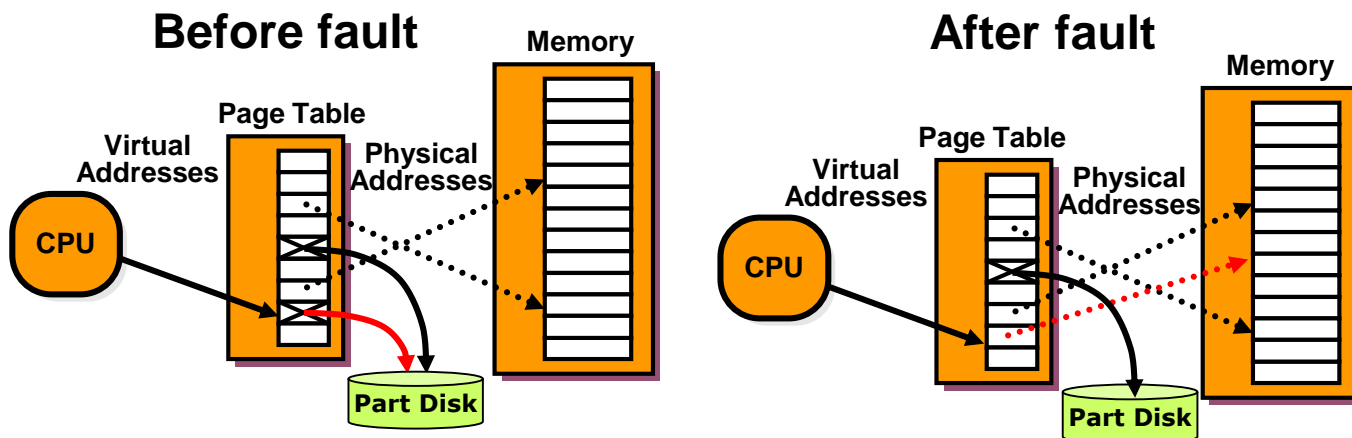
- 需要虚实变换，硬件通过OS维护的页表将虚拟地址转换为物理地址
- 工作站，服务器，现代PC



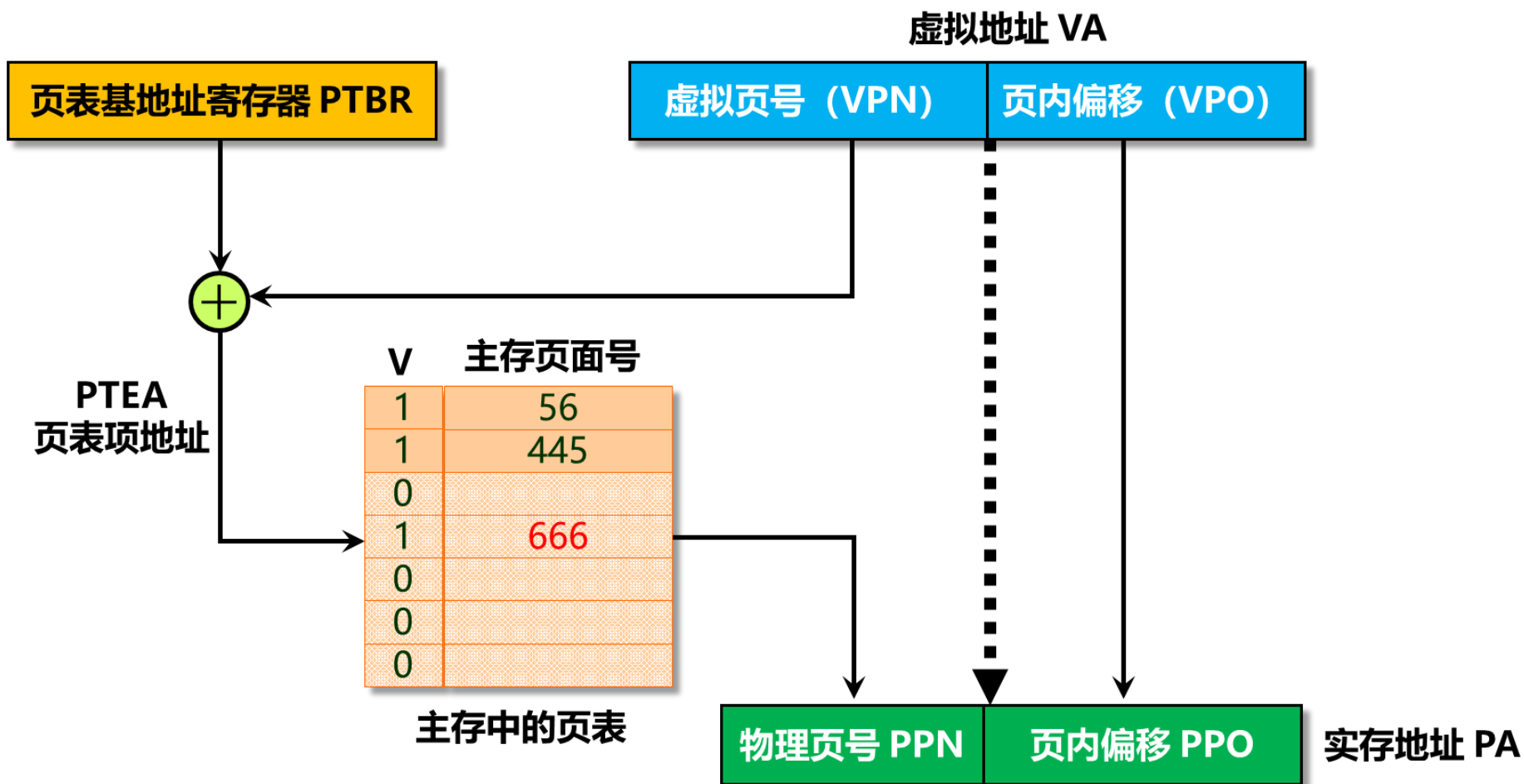
# 缺页 Page Faults

## 4.4

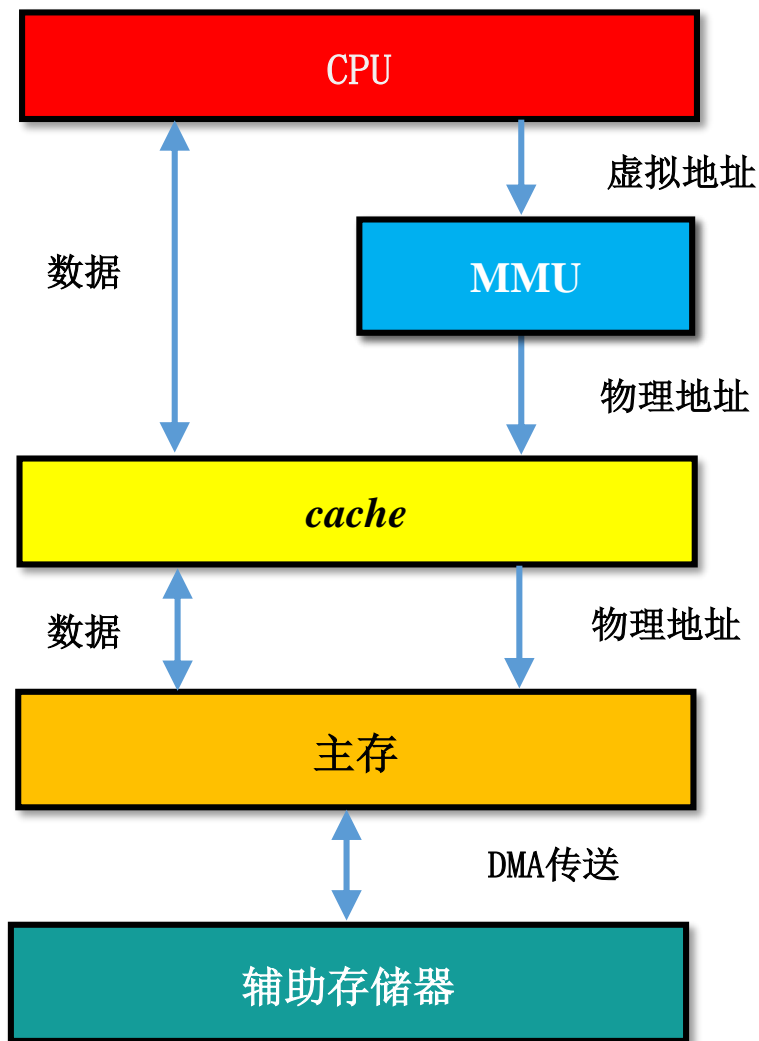
- 页表指示虚拟地址不在内存中
  - 操作系统负责将数据从磁盘迁移到内存中
  - 当前进程挂起
  - 操作系统负责所有的替换策略
  - 唤醒挂起进程



# 页式虚拟存储器结构

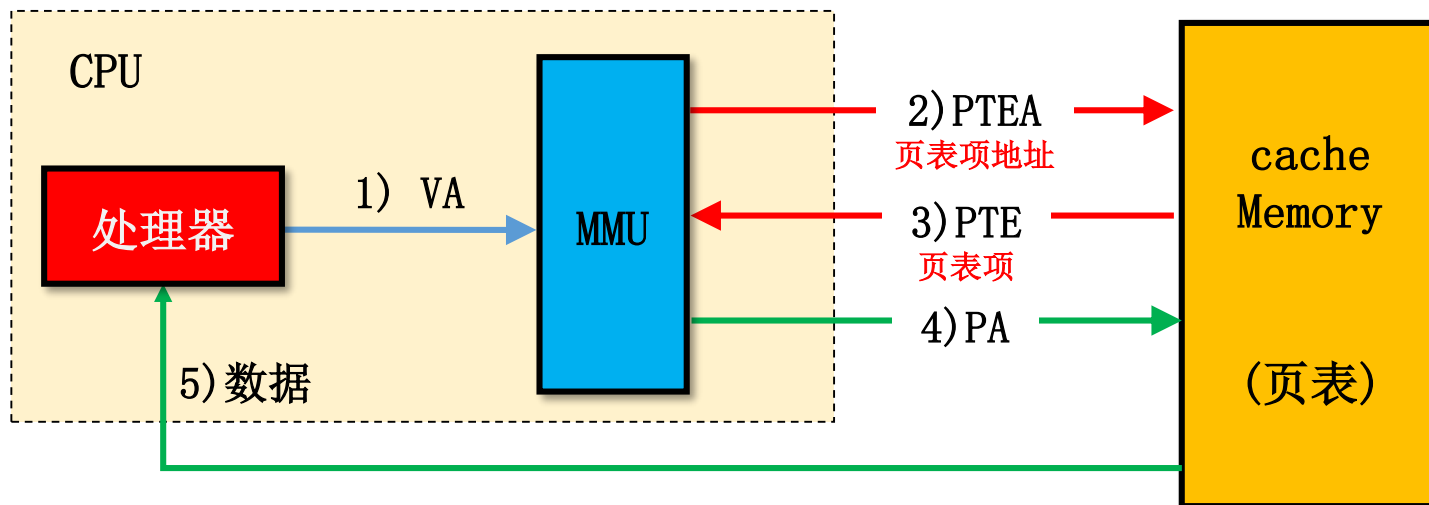


# 层次性结构

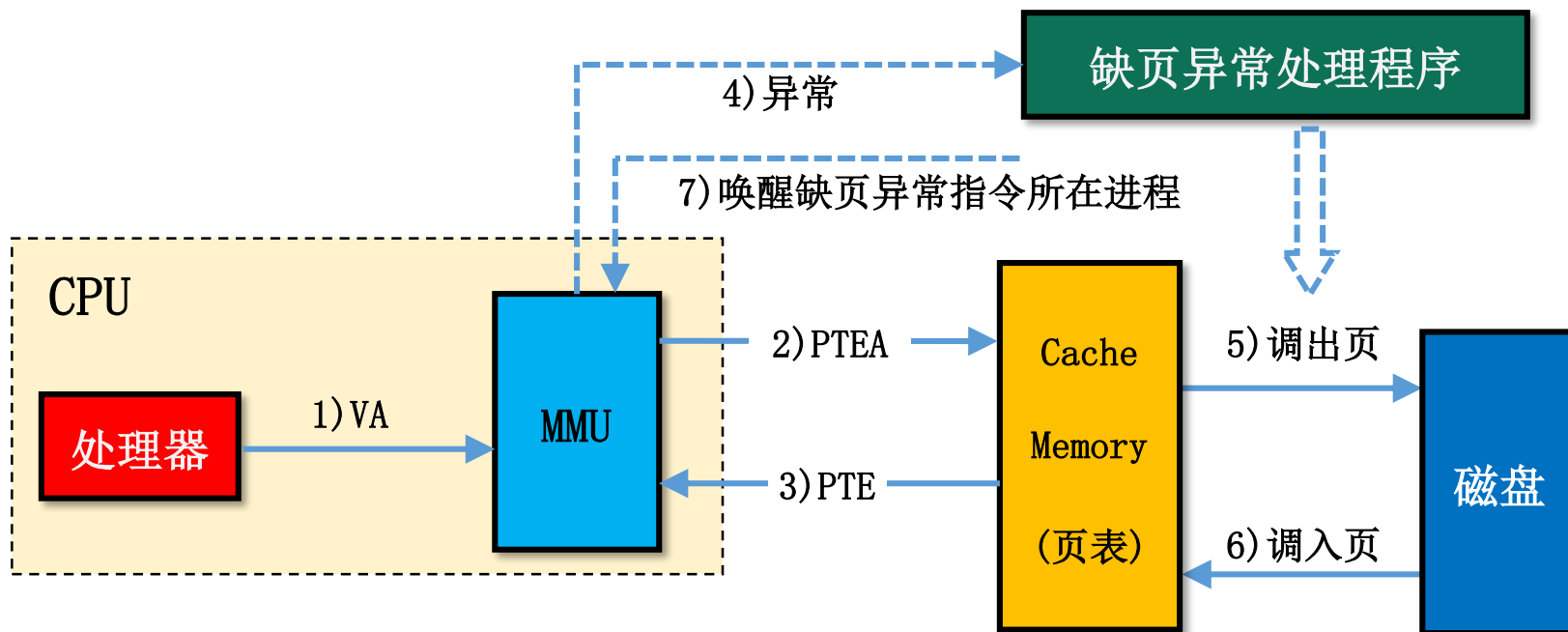


为什么cache用物理地址访问？

# 虚拟地址→物理地址（页命中）

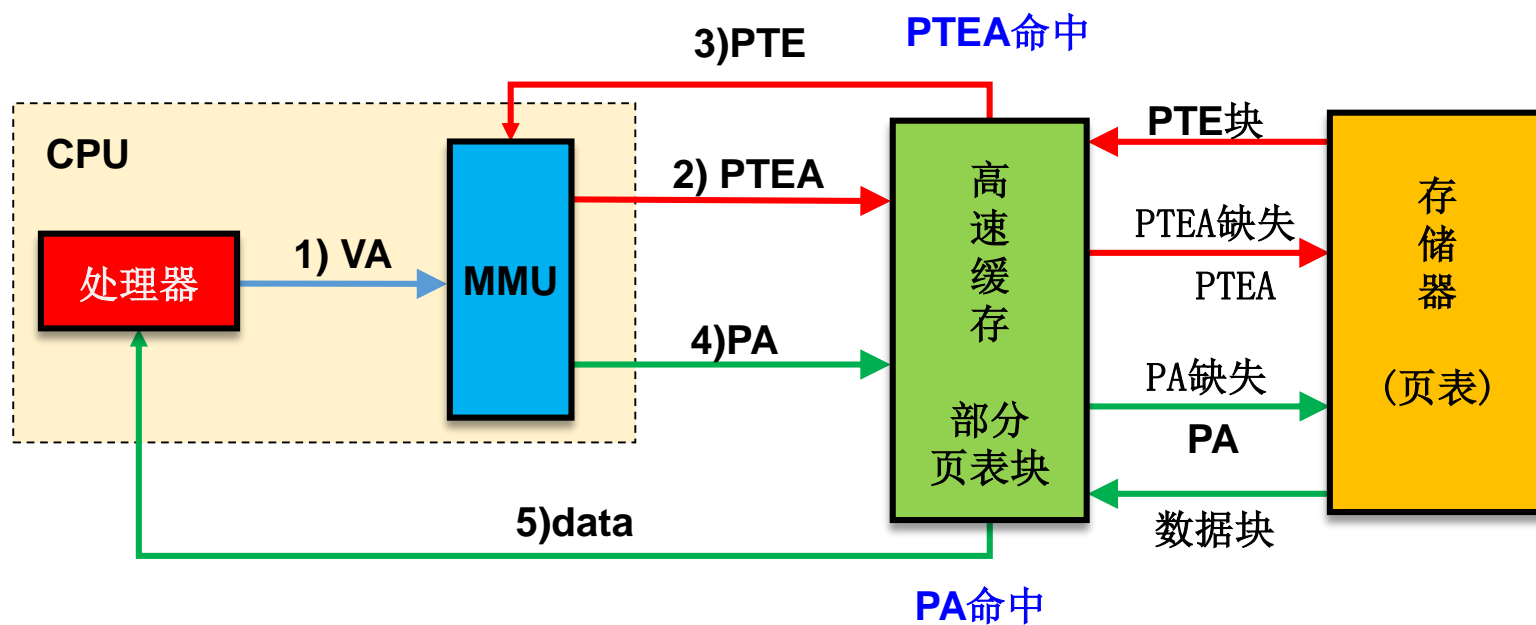


# 虚拟地址→物理地址（缺页）





# 带Cache的虚存-主存层次



# 快表提高地址转换速度

- 地址转换速度慢

- 访问页表，访问数据，需2次访存，速度慢
- 为缩小页表大小，OS普遍采用多级页表结构，速度更慢

- 加速方法：引入一个体积小的快表TLB

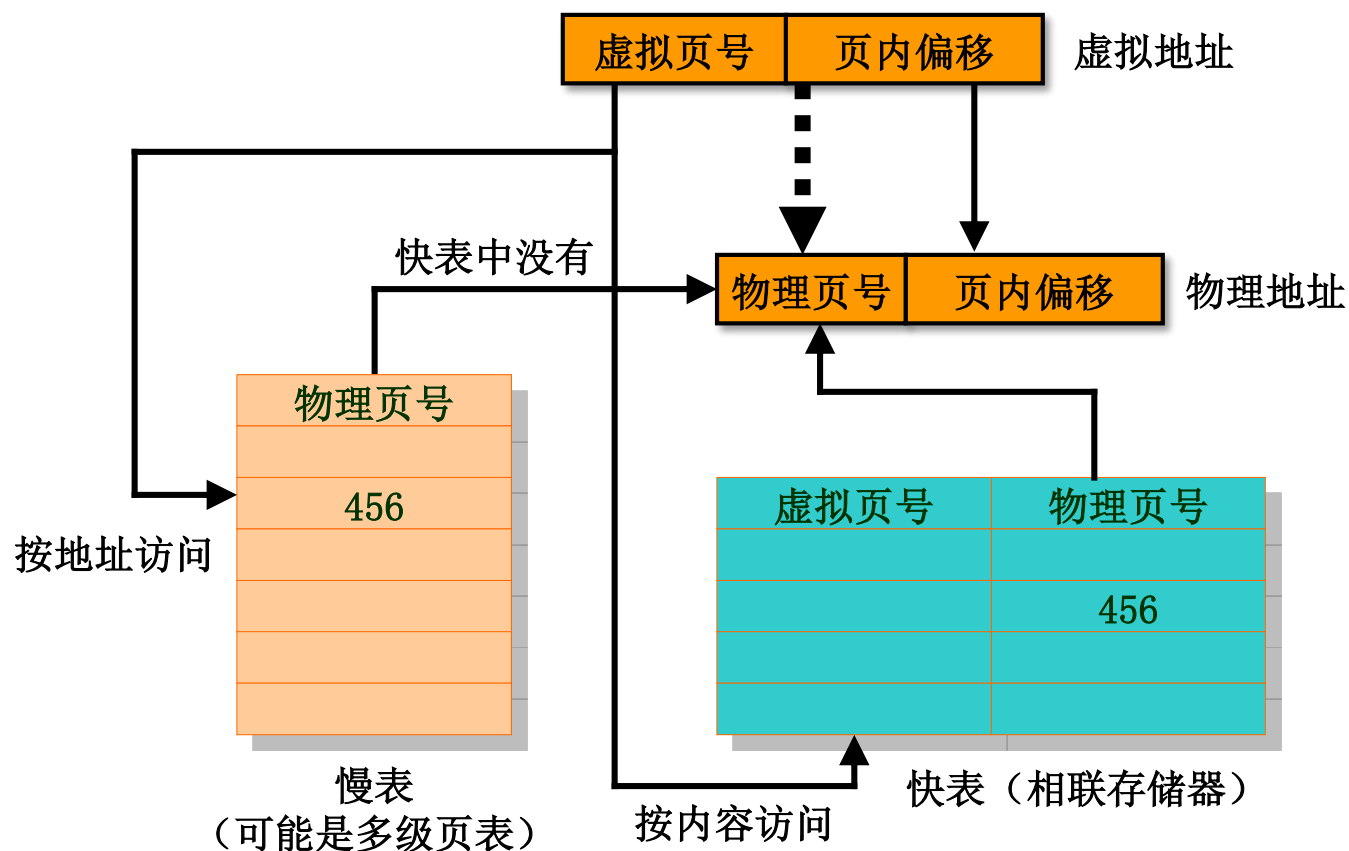
- 缓存页表中经常被访问的表项
- (Valid, VPN, PPN)

多路并发比较机制

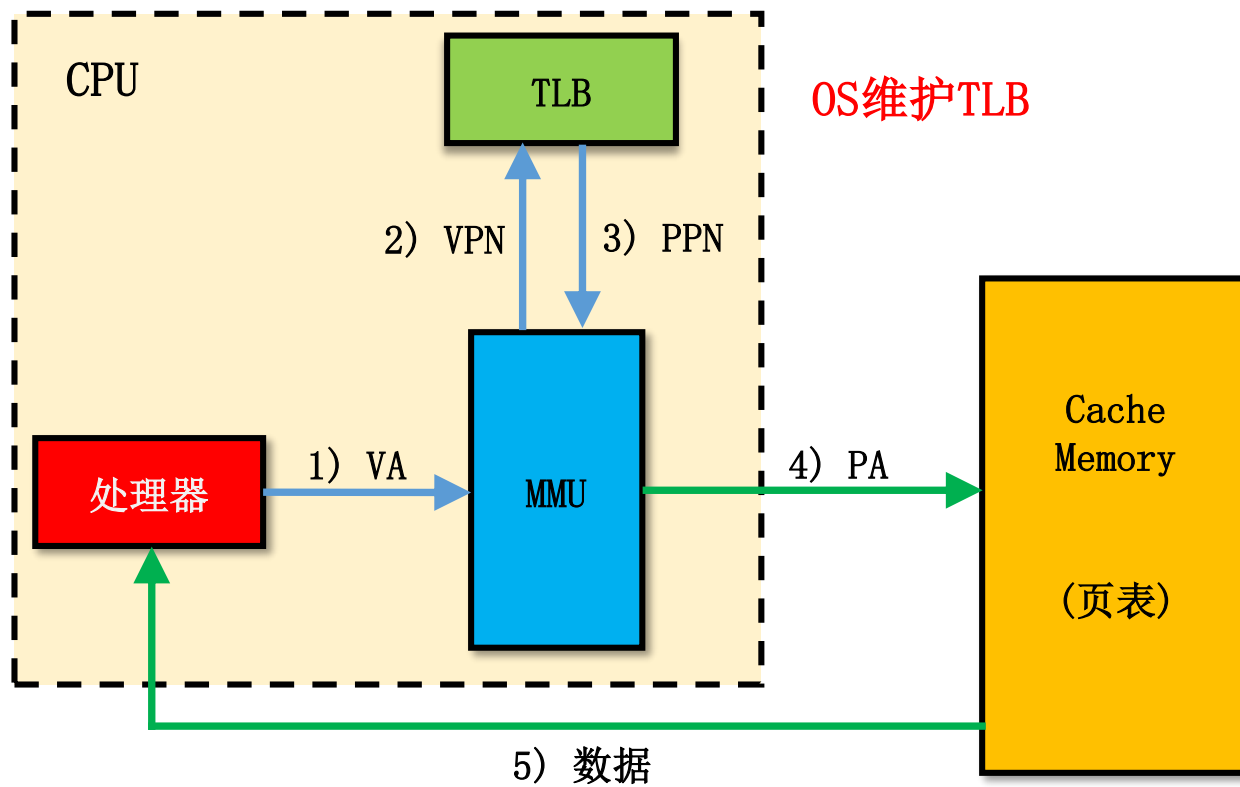
- 快表引入相联存储器机制，提高查找速度

- 采用随机替换算法

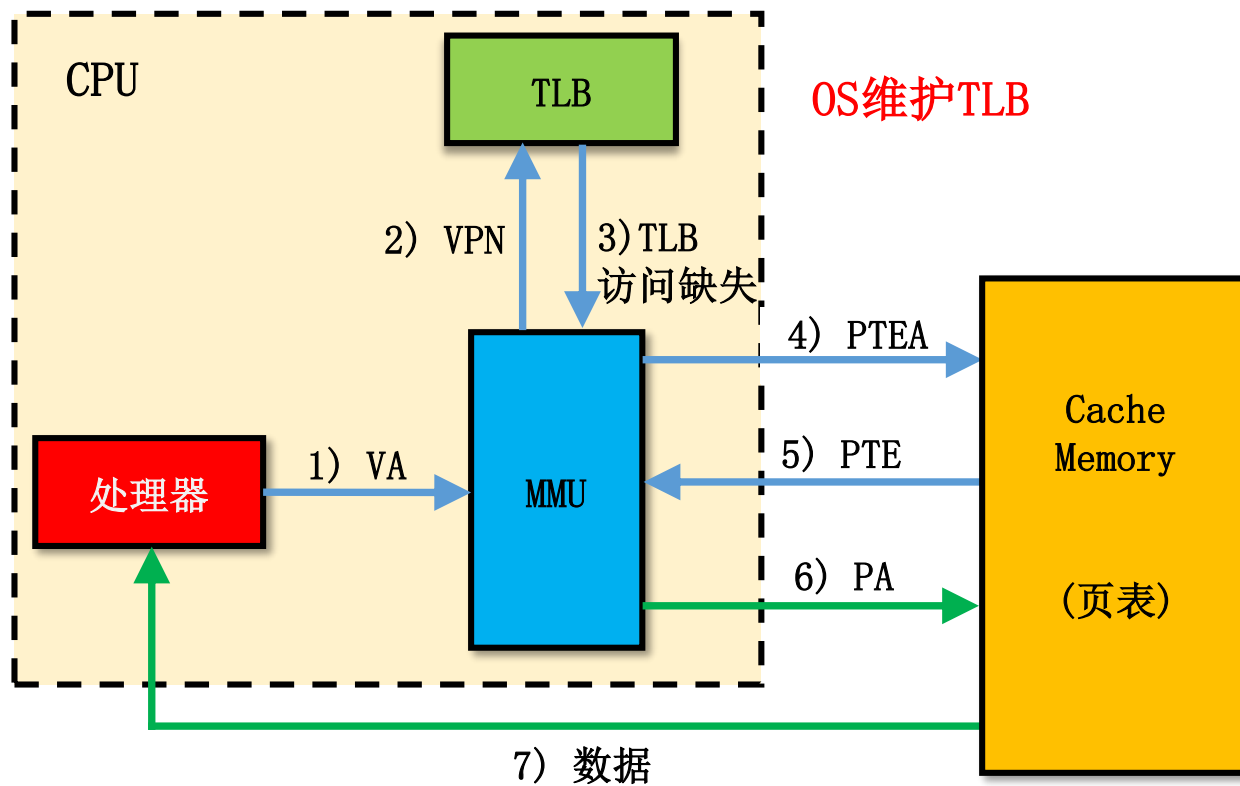
# 经快慢表实现内部地址转换



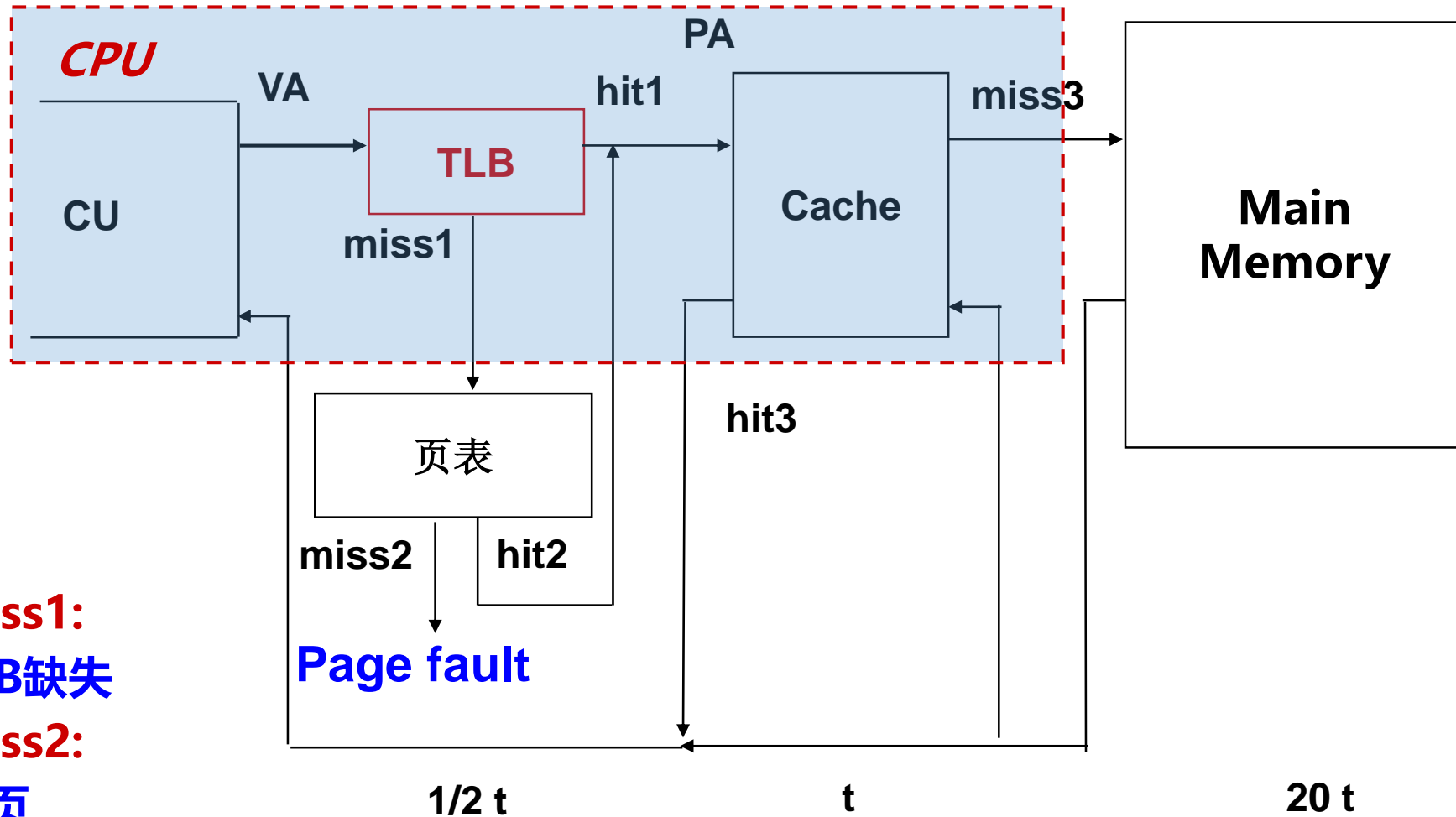
# TLB 命中



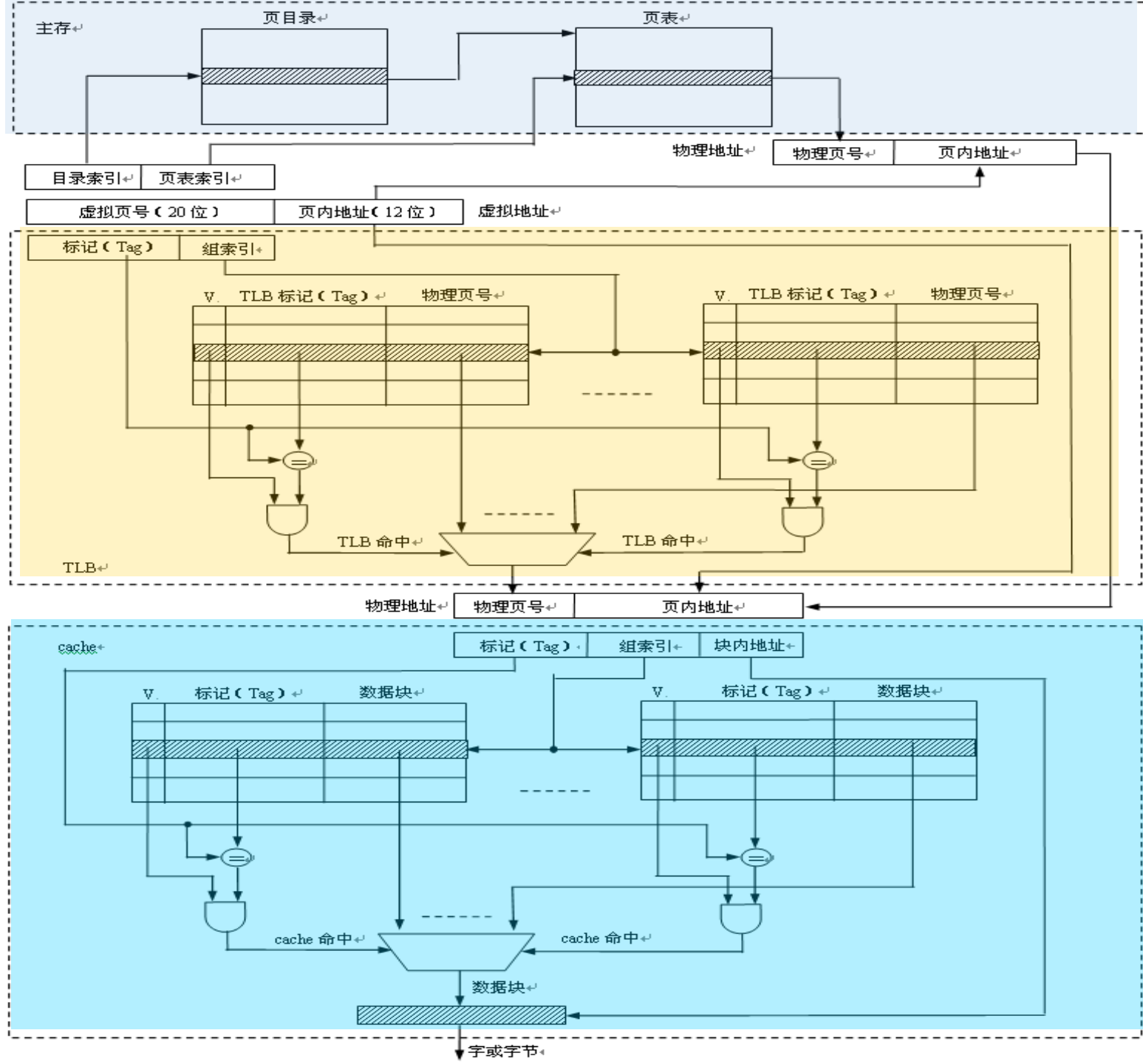
# TLB 缺失



# Translation Look-Aside Buffers



TLB冲刷指令和Cache冲刷指令  
都是操作系统使用的特权指令



虚拟地址

TLB

页表

物理地址

cache

主存

缺页处理

命中

缺失

主存

页目录

页表

物理地址

物理页号

页内地址

目录索引

页表索引

虚拟页号 (20 位)

页内地址 (12 位)

虚拟地址

标记 (Tag)

组索引

TLB 标记 (Tag)

物理页号

TLB 标记 (Tag)

物理页号

TLB 命中

TLB 命中

物理地址

物理页号

页内地址

cache

标记 (Tag)

组索引

块内地址

标记 (Tag)

数据块

标记 (Tag)

数据块

cache 命中

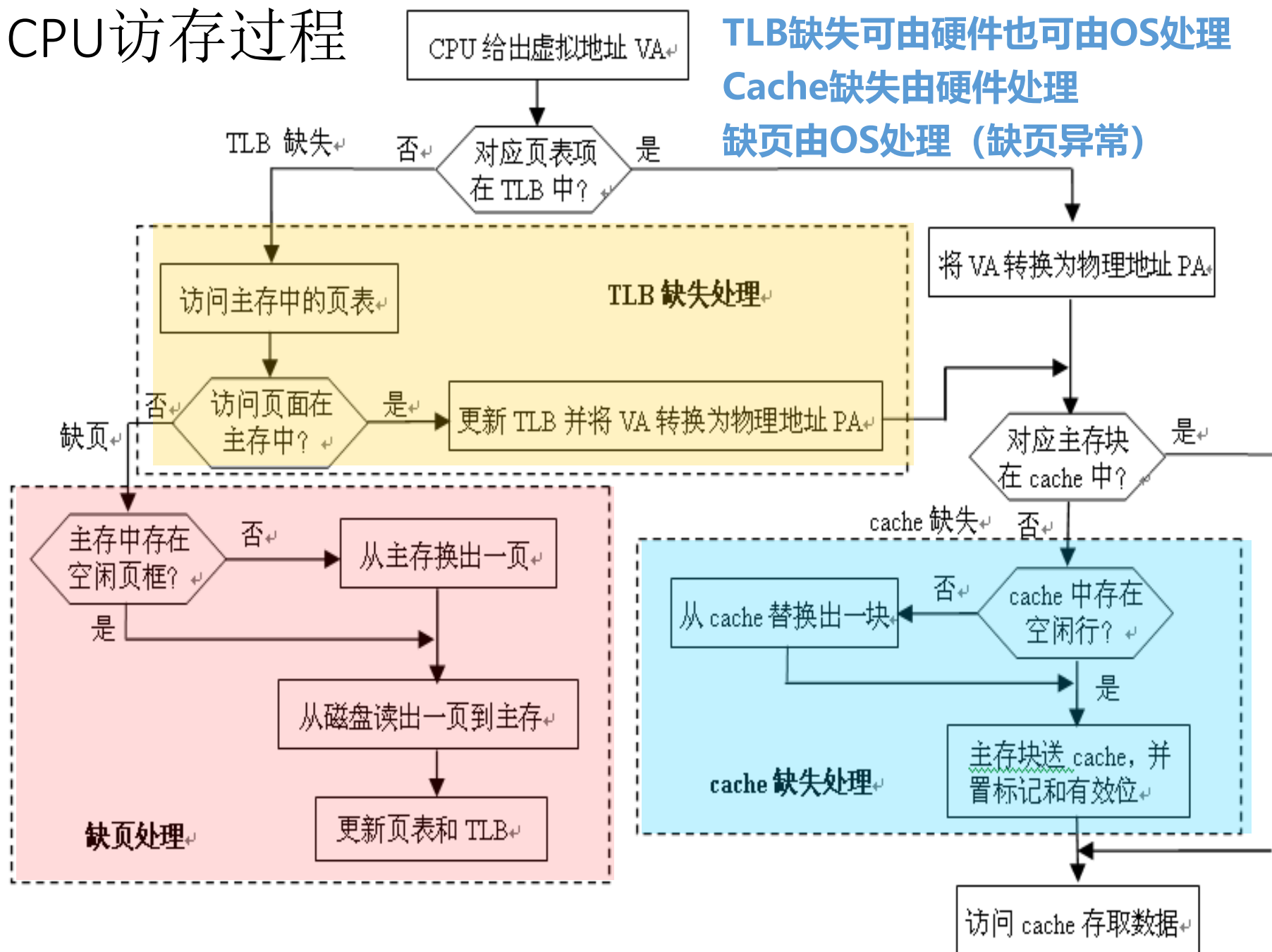
cache 命中

数据块

字或字节

# CPU访存过程

TLB缺失可由硬件也可由OS处理  
Cache缺失由硬件处理  
缺页由OS处理（缺页异常）





# 举例：三种不同缺失的组合

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缺失但页表命中，信息在主存，就可能在Cache
miss	hit	miss	可能，TLB缺失但页表命中，信息在主存，但可能不在Cache
miss	miss	miss	可能，TLB缺失页表缺失，信息不在主存，一定也不在Cache
hit	miss	miss	不可能，页表缺失，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，信息不在主存，Cache中一定也无该信息

最好的情况是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？ hit、hit、miss和miss、hit、hit 访存1次

以上组合中，最坏的情况是？ 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？ miss、hit、miss 不需访问磁盘、但访存至少2次