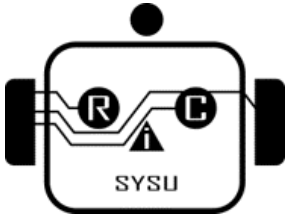


计算机组成原理

第五章：存储器层次结构

中山大学计算机学院

陈刚

2022年秋季

上讲回顾

回顾内容

- 5.3 高速缓冲存储器
  - Cache的一致性问题
  - Cache的替换策略
  - 多级Cache

回顾——Cache的一致性问题

如何保持Cache一致性呢？

写命中(Write Hit)
要写的单元已经在Cache中

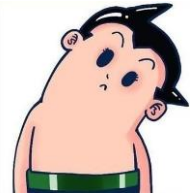
- **Write Through**
  (写通过、写直达、直写)
- **Write Back**
  (一次性写、写回、回写)

写不命中(Write Miss)
要写的单元不在Cache中

- **Allocate-on-miss (写分配)**
- **No-allocate-on-write (写不分配)**

回顾——Cache替换算法

什么时候需要进行Cache替换?



直接映射（Direct Mapped）Cache

- 映射唯一，无条件用新信息替换老信息

N路组相联（N-way Set Associative）Cache

- 每个主存数据有N个Cache行可选择，需考虑替换哪一行

全相联（Fully Associative）Cache

- 每个主存数据可存放到Cache任意行中，需考虑替换哪一行

常用替换算法：

先进先出FIFO、最近最少用LRU、最不经常使用LFU、随机替换

本节概要

重点内容

- 5.4 虚拟存储器

基本要求

- 理解虚拟存储器的基本概念
- 掌握逻辑地址—物理地址的转换、页表和缺页处理
- 了解替换策略、快表和存储保护

虚拟存储器的基本概念

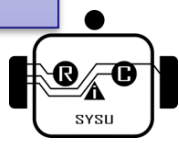
虚拟存储器的基本概念

存储管理

- 早期采用单道程序，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

  无需进行存储管理，即使有也是很简单的。
- 现在采用多道程序，系统的存储器中包含：
  - 操作系统
  - 若干个用户程序

在多道程序系统中，存储器中“用户程序”区须进一步划分以适应多个进程交替执行。划分任务由OS动态执行，称为存储器管理(memory management)



虚拟存储器的基本概念

同时运行更多进程，内存需求增大，怎么办？

- **虚拟存储器(Virtual Memory)**
  - 一种将内外存统一管理的存储管理机制就是虚拟存储管理。将用户程序存储空间分页，但不是把所有页面一起调到主存，而是采用“按需调页(Demand Paging)”，在外存和主存之间以固定页面进行调度。
  - 虚拟存储器引入了“虚拟地址”的概念。其概念是1961年由英国曼彻斯特大学的Kilbrn等提出

构造虚拟存储系统的主要动机

1、允许多个程序有效而安全地共享存储器，消除内存因小而有限的存储器容量给程序设计造成的障碍

- 可以更有效的共享处理器和主存
- 虚拟存储器实现程序地址空间到物理空间的转换，加强了各个程序地址空间之间的保护

2、允许单用户程序大小超过主存储器的容量

- 虚拟存储器自动管理主存和辅存组成的两级层次结构

虚拟存储器的基本概念

虚拟地址的出现可追朔到上世纪六十年代的 Atlas 计算机。当时 Atlas 计算机是一个庞然大物，只有96KB的内部存储器和576KB的磁鼓作为外部存储器。

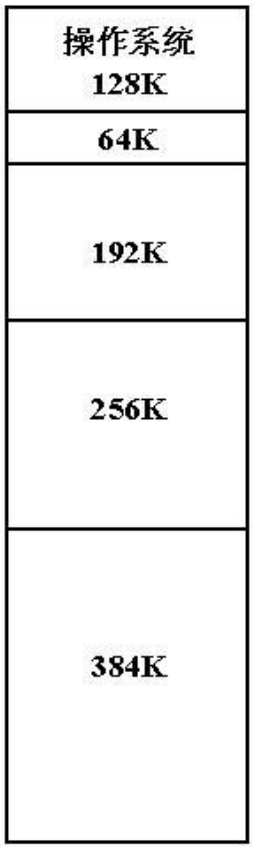
程序员被迫管理物理内存与磁鼓之间的数据交换，尽可能地利用外部存储器多换入换出数据，以扩展物理内存空间。这些数据交换工作严重地挫伤了程序员的编程热情。在那样的背景下， Atlas计算机提出了Virtual Memory，同时引入分页机制。

虚拟存储器的基本概念

简单分区

- 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
- 当一个进程调入主存时，分配给它一个能容纳的最小分区

例:对于需196K的进程可分配256K分区



固定分区举例

简单分区方式的缺点：

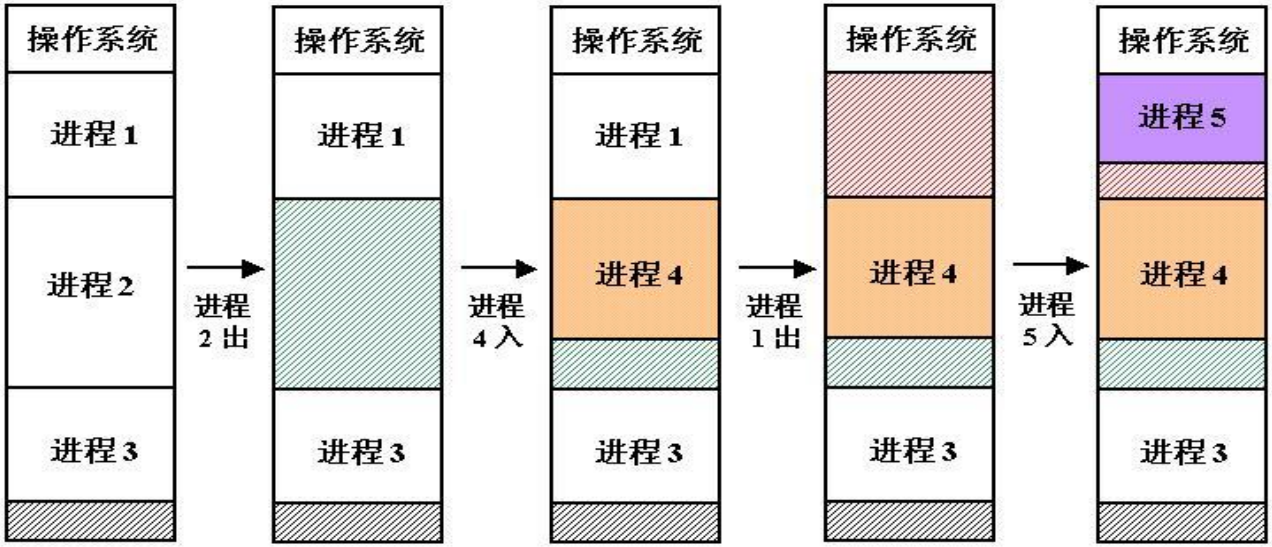
固定长度的分区，可能会浪费主存空间。多数情况下，进程对分区大小的需求不可能和提供的分区大小一样

可以采用有效的可变长分区的方式！

虚拟存储器的基本概念

可变长分区（Partitioning）

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



分区的效果

虚拟存储器的基本概念

分页（Paging）

基本思想

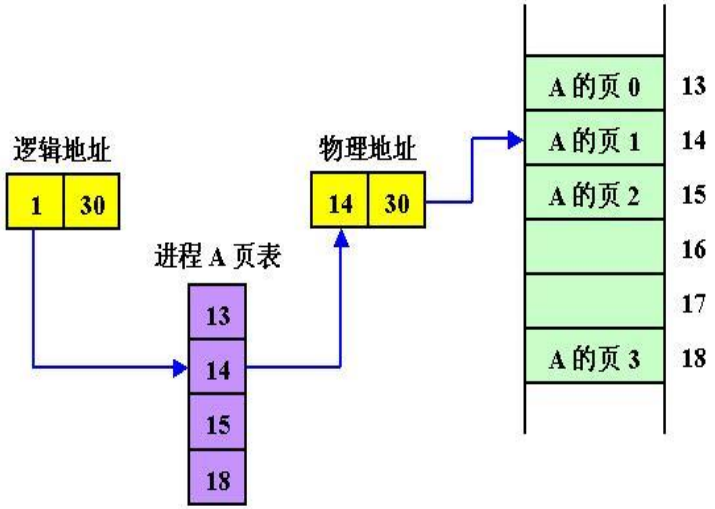
- 内存分成固定长且较小的存储块，每个进程也划分成固定长的程序块
- 程序块(页/page)可装到内存可用的存储块(页框/page frame)中
- 无需用连续页框来存放一个进程
- 操作系统为每个进程生成一个页表
- 通过页表(page table)实现逻辑地址向物理地址转换(Address Mapping)

页表：页和页框一一对应的关系表

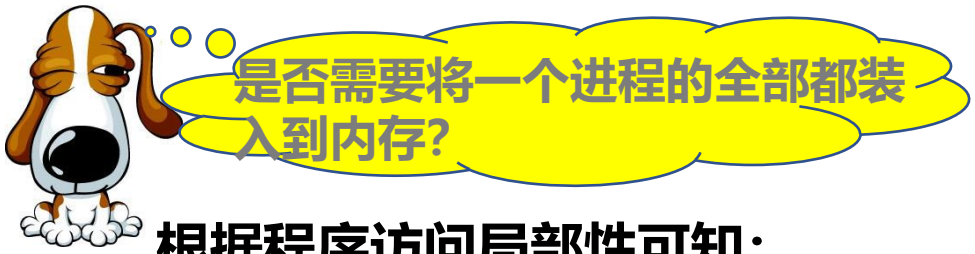
虚拟存储器的基本概念

分页（Paging）

- 逻辑地址(Logical Address)
  - 程序中的指令所用的地址
- 物理地址(physical Address)
  - 存放指令或数据的实际内存地址



逻辑和物理地址



根据程序访问局部性可知：

可以仅把当前活跃页面调入主存，其余留在磁盘上！

浪费的空间最多是最后一页的一部分！

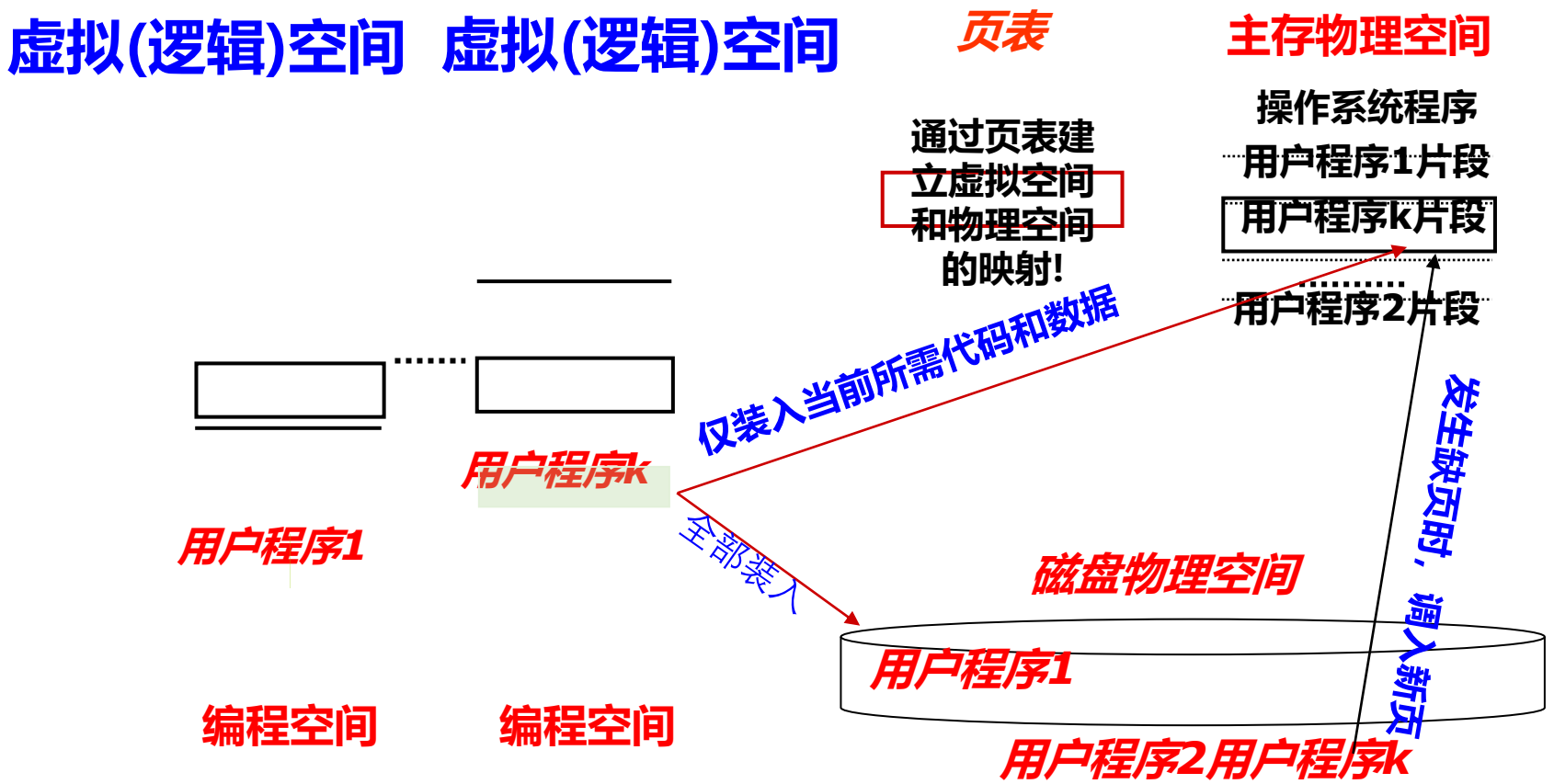
采用 “按需调页 / Demand Paging”方式对主存进行分配！

虚拟存储器的基本概念

◆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(亦称虚拟地址或虚地址)转化为物理地址(亦称主存地址或实地址)
- 在发生指令或数据访问失效时，由操作系统进行主存和磁盘之间的信息交换

虚拟存储器的基本概念



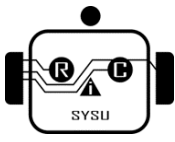
◆ 程序执行时，将部分(“热”页)映射到主存，其余映射到磁盘

虚拟存储器组织方式

虚拟存储器的组织方式

实现虚拟存储器管理，需考虑：

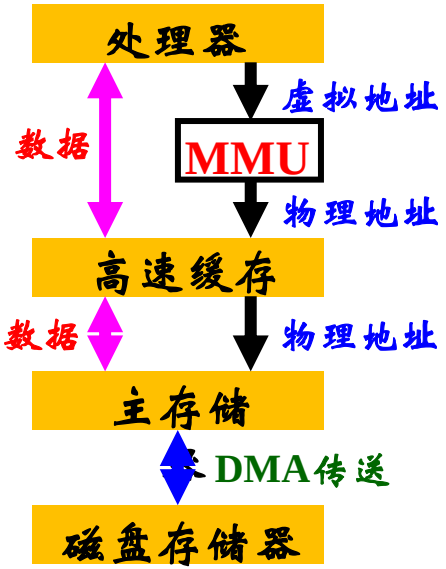
- 块大小(在虚拟存储器中“块”被称为“页 / Page”)应多大?
- 主存与辅存的空间如何分区管理?
- 程序块 / 存储块之间如何映像?
- 逻辑地址和物理地址如何转换，转换速度如何提高?
- 主存与辅存间如何进行内容切换(与Cache所用策略相似)?
- 页表如何实现，页表项中要记录哪些信息?
- 如何加快访问页表的速度?
- 如果要找的内容不在主存，怎么办?
- 如何保护进程各自的存储区不被其他进程访问?



虚拟存储器的组织方式

- 虚拟存储器机制由硬件与操作系统共同协作实现，涉及到OS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

- 三种虚拟存储器实现方式：
  页式、段式、段页式



虚拟存储器的组织方式

页式虚拟存储器

- 虚存地址空间和主存地址空间按统一大小分成若干页，虚存称为虚页，主存称为实页
- 程序按页从磁盘调入主存(某一虚页调入某一实页)
- 虚存的地址格式(逻辑地址格式)

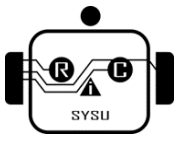
  虚页号 + 页内地址
- 主存的地址格式(物理地址格式)

  实页号 + 页内地址

虚拟存储器的组织方式

页式虚拟存储器

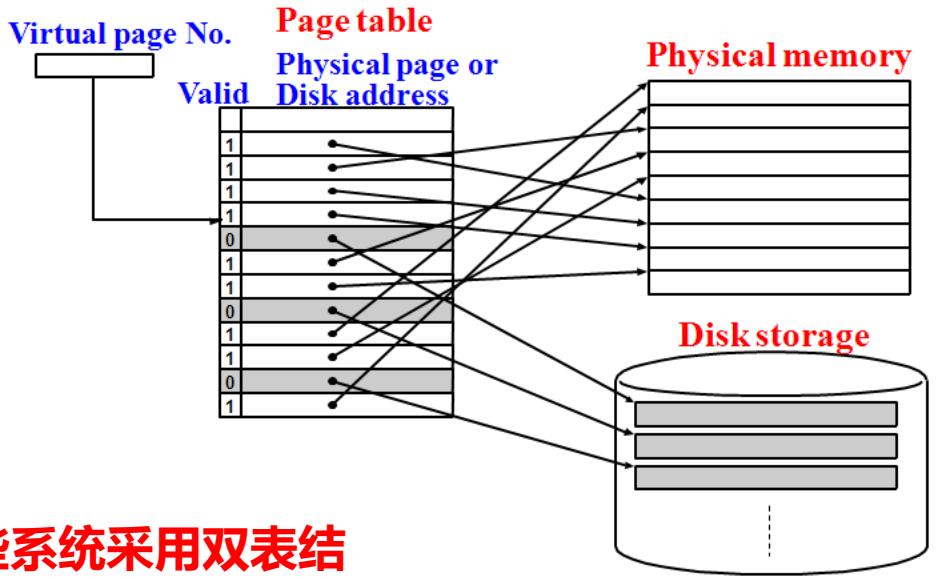
- 虚拟地址的引入分离了程序员看到的逻辑地址和处理器使用的物理地址， 通过建立一个映射关系表来存放虚拟地址与物理地址之间的映射关系，该映射关系表被称为页表(Page Table)
- 页表：反映虚页与实页的映射关系，用来实现虚实地址的转换
  - 建立在主存中，操作系统为每道程序建立一个页表
  - 设置页表基址寄存器：保存页表在主存中的起始地址



虚拟存储器的组织方式

页式虚拟存储器

- 指令给出虚拟地址
- 每个页表项记录对应的虚页情况
- Valid为0说明“miss”(称为page fault / 缺页)
- CPU执行指令时，先将逻辑地址转换为物理地址
- 地址转换由MMU实现



有些系统采用双表结构：主存页地址和磁盘页地址分开

◆缺页的代价是多少?

读磁盘(数百万个时钟周期)

页大小比Cache中Block大得多！

采用全相联映射！ Why?

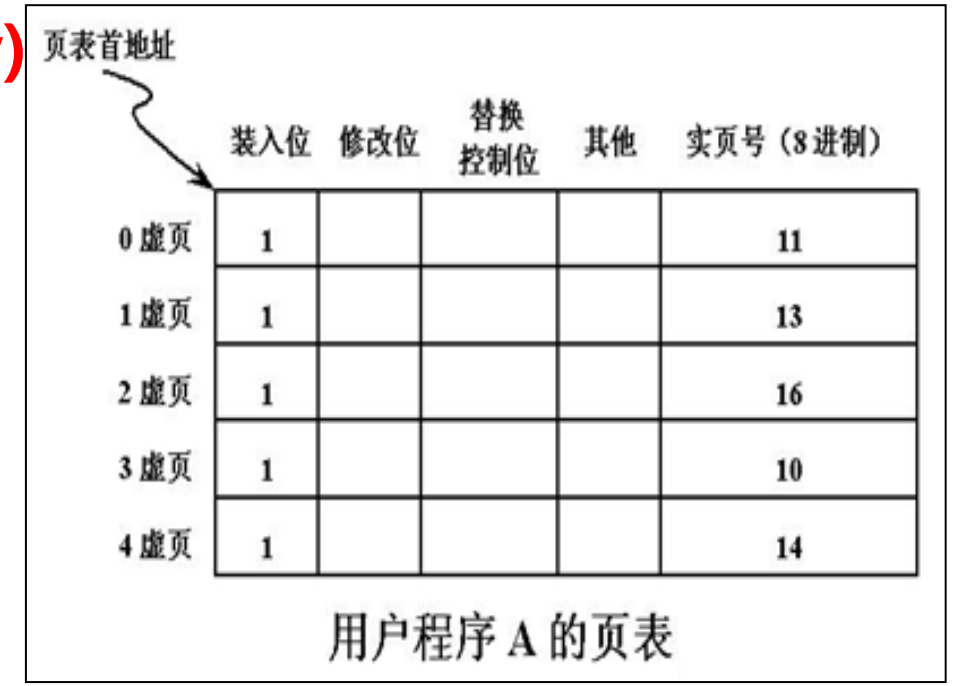
通过软件来处理“缺页”！Why?

采用Write Back写策略！ Why?

虚拟存储器的组织方式

页表结构

- 每个进程有一个页表
- 页表中有装入位、修改(Dirty)位、替换控制位、访问权限位、禁止缓存位、实页号
- 在页表中，一个页号与其对应的物理块号称之为一个页表项
- 页表项数由进程大小决定
- 页表在主存的首地址记录在页表基址寄存器中



用户程序 A 的页表

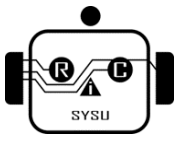
虚拟存储器的组织方式

例：若某页式虚拟存储器，有32位虚拟地址，页大小为4KB，每个页表项占4B，则操作系统为进程分配的页表最大为多少？

解：　页表的项数最大为：$2^{32}/2^{12}=2^{20}$

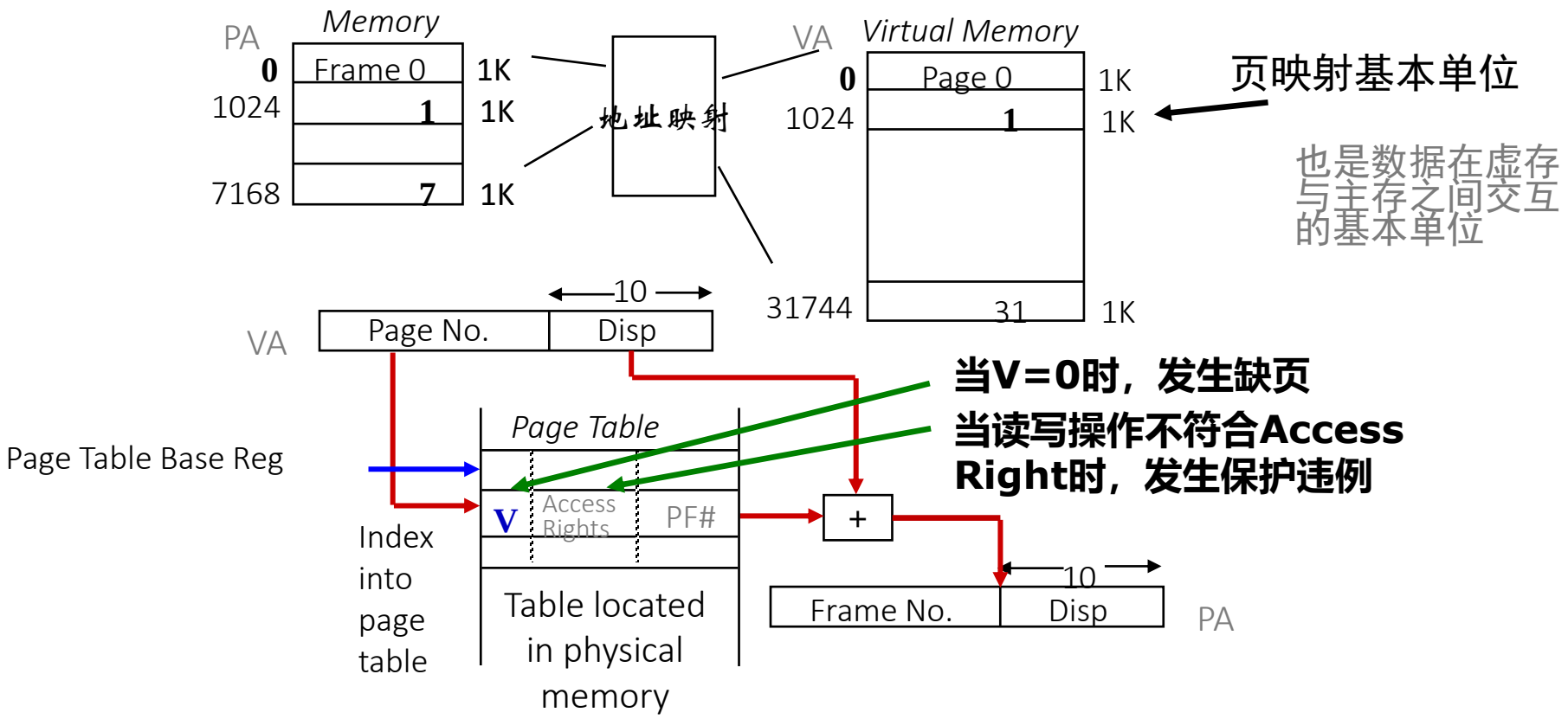
页表大小=2^{20}页表项×2^{2}B/页表项=4MB

操作系统为每个进程分配的页表为4MB 。



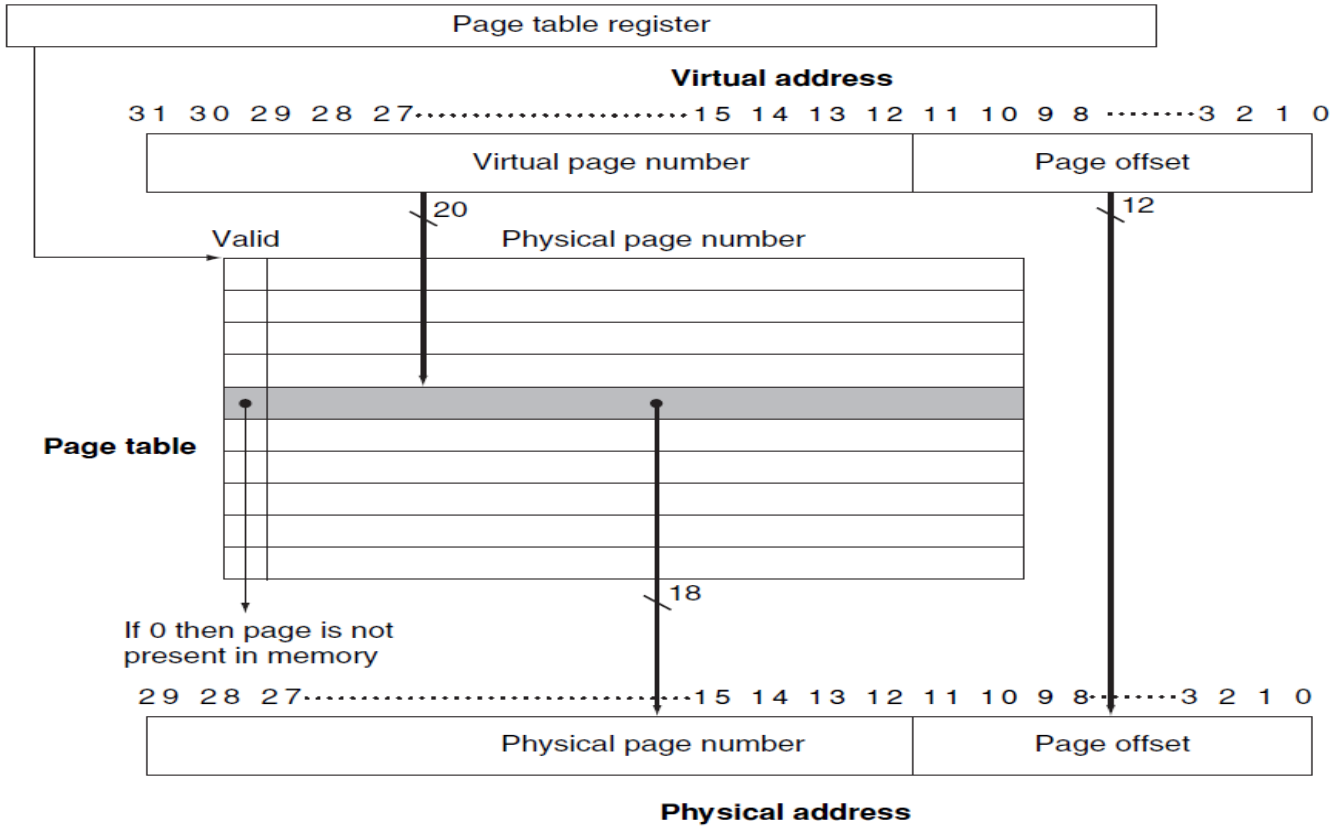
虚拟存储器的组织方式

逻辑地址转换为物理地址的逻辑过程



虚拟存储器的组织方式

逻辑地址转换为物理地址的硬件实现过程

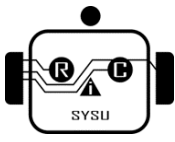


虚拟存储器的组织方式

虚拟存储器访问中可能出现的异常情况

1. 缺页(page fault)

- 产生条件：当Valid(有效位 / 装入位)为 0 时
- 相应处理：从磁盘中读信息到内存，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，页面淘汰时，根据“dirty”位确定是否要写磁盘
- 异常处理结束后：缺页发生时，当前指令的执行被阻塞，当前进程挂起；缺页处理结束后，回到原指令继续执行



虚拟存储器的组织方式

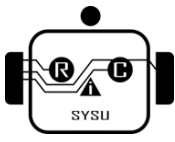
虚拟存储器访问中可能出现的异常情况

2. 保护违例(protection_violation_fault)

- **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”信息
- **异常处理结束后：** 当前指令执行被阻塞，当前进程终止

Access Rights (存取权限)可能的取值有哪些?

R = Read-only、R/W = read/write、X = execute only



快表(TLB)

快表（TLB）

问题1：一次内存引用要访问几次内存?

- 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，一次访问所需的数据（或指令），简单的虚拟存储器速度太慢

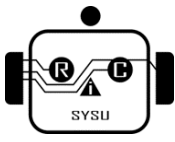
提高访问性能的关键在于依靠页表的访问局部性

- 解决办法：使用Cache来存储页表项，称为TLB，它包含了最近使用的那些页表项

  转换后备缓冲器TLB(Translation Lookaside Buffer)

问题2：引入TLB的目的是什么?

减少到内存查页表的次数！（加速虚实地址转换）



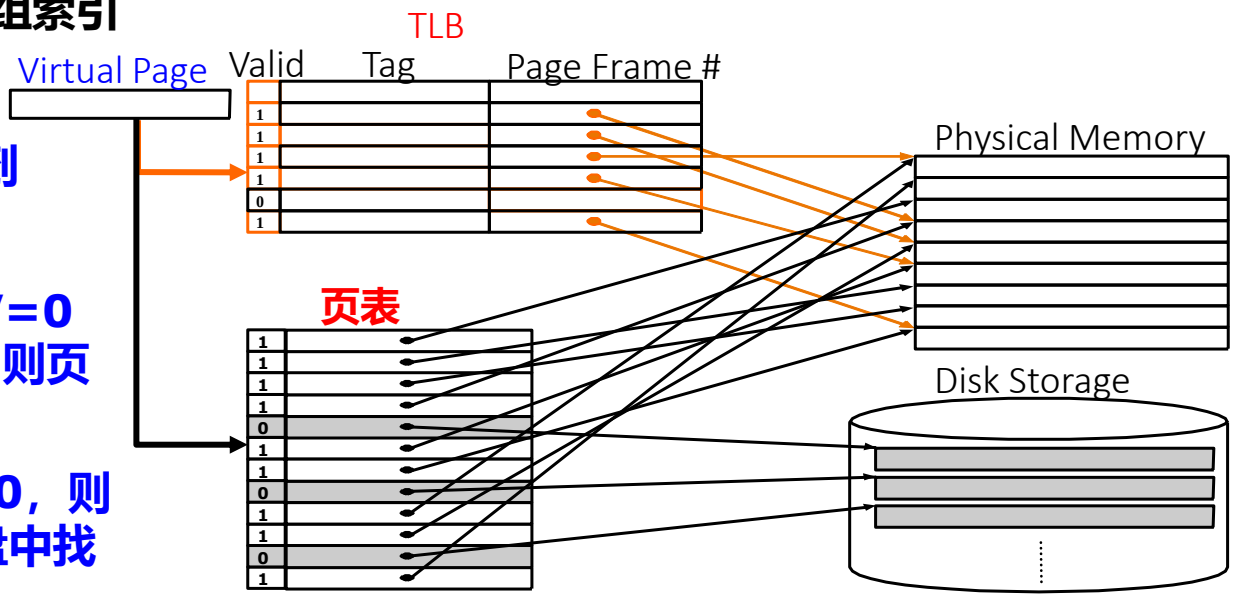
快表（TLB）

◆TLB组织

- TLB全相联时没有Index，只有Tag，虚页号需与每个Tag比
- 组相联时，虚页号高位为Tag，低位为Index—组索引

Translation Lookaside Buffer or *TLB(快表)*

Virtual Address (Tag+Index)	Physical Address	Dirty	Ref	Valid	Access	
虚页号:Tag+Index	对应物理页框号					



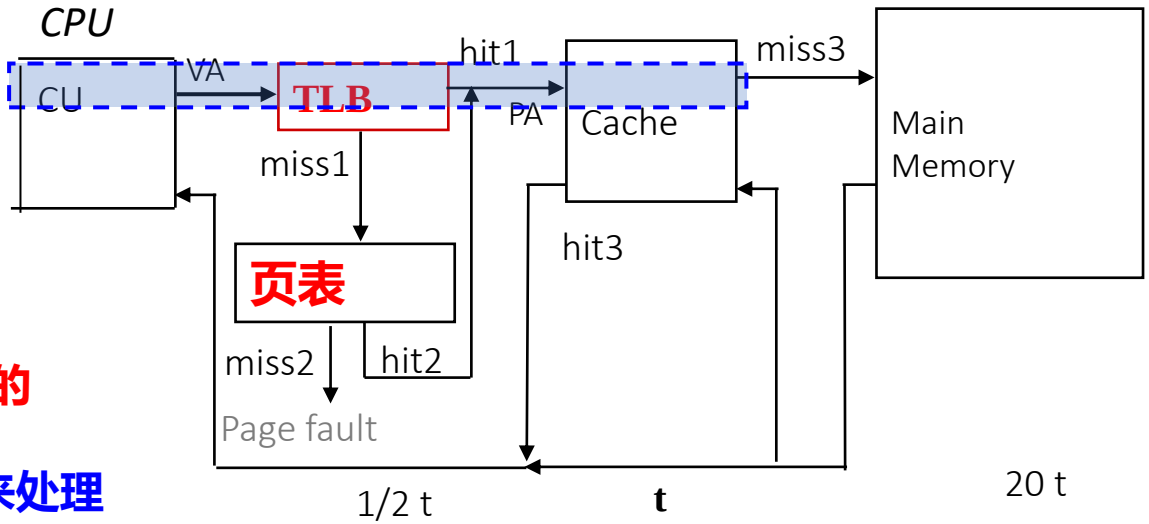
先由虚页号到TLB中找

若TLB中的V=0或Tag≠VA,则页表中找

若页表中V=0，则缺页，到磁盘中找

快表（TLB）

Miss1:
TLB中没有VA
Miss2:
页面不在主存
Miss3:
PA 在主存中，不在Cache中



miss1>>miss2, Why?

因为TLB中的项比主存中的页数少得多！

可以用硬件也可以用软件来处理TLB失靶(miss1)

Miss1处理：查内存页表，若hit2，则将页表项装入TLB中，并进行地址转换；否则“缺页”，引起相应的异常处理

多用全相联：命中率高，小、成本不高
采用随机替换策略：降低替换算法开销
采用回写策略：减少访问内存的次数

TLB的一些典型指标：

TLB大小：16~512项
块大小：1~2项(每个表项4-8B)
命中时间：0.5~1个时钟周期
失靶损失：10~100个时钟周期
命中率：90~99%

快表（TLB）

- 三种不同缺失：TLB缺失、Cache缺失、缺页
- 三种缺失的组合情况的可能性分析

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,信息不在主存,TLB中一定无该页表项
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

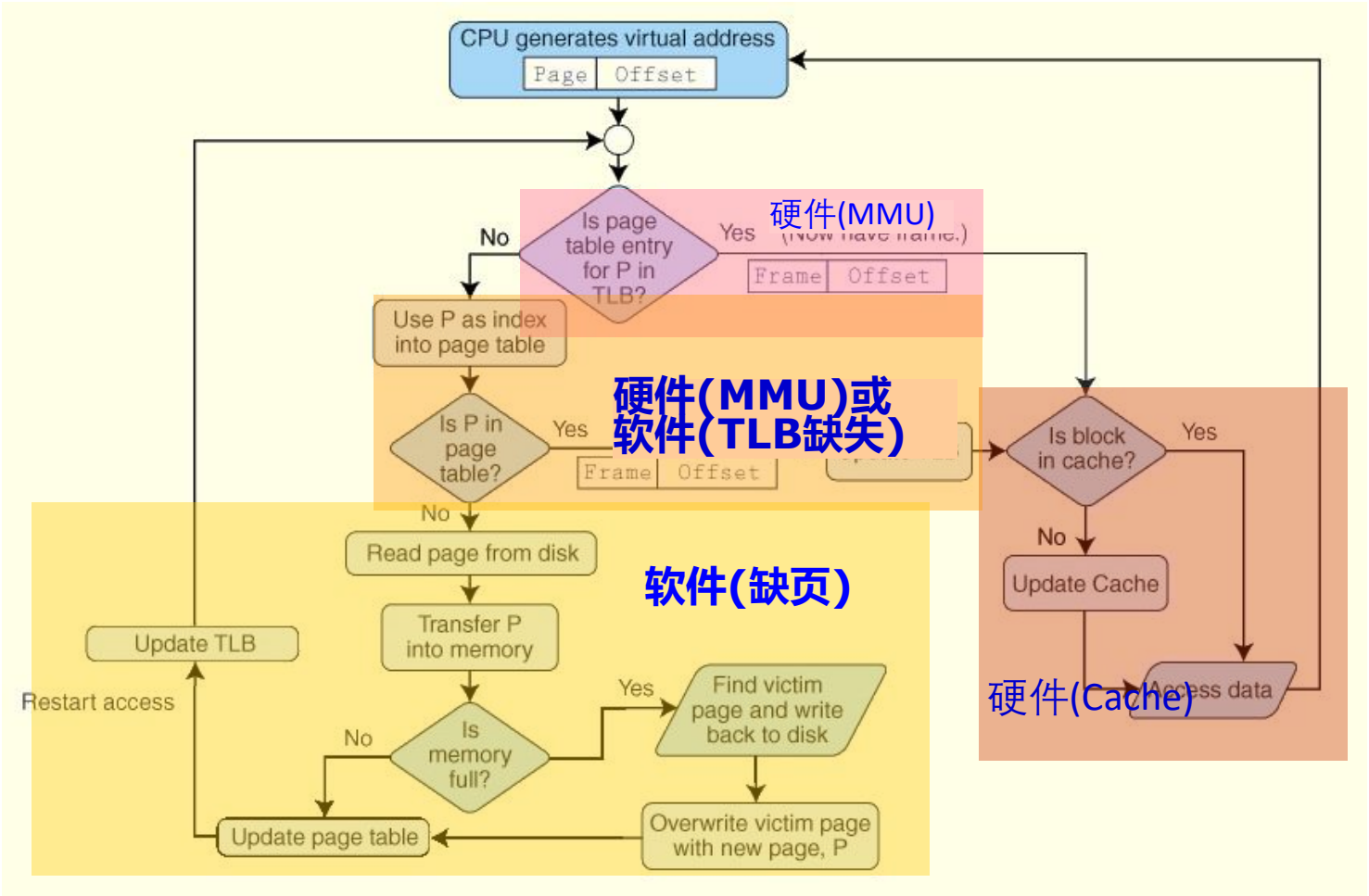
最好的情况应是:hit、hit、hit，此时访问主存几次? 不需要访问主存!

上表，最好的情况是: hit、hit、miss和miss、hit、hit 只需访问主存1次

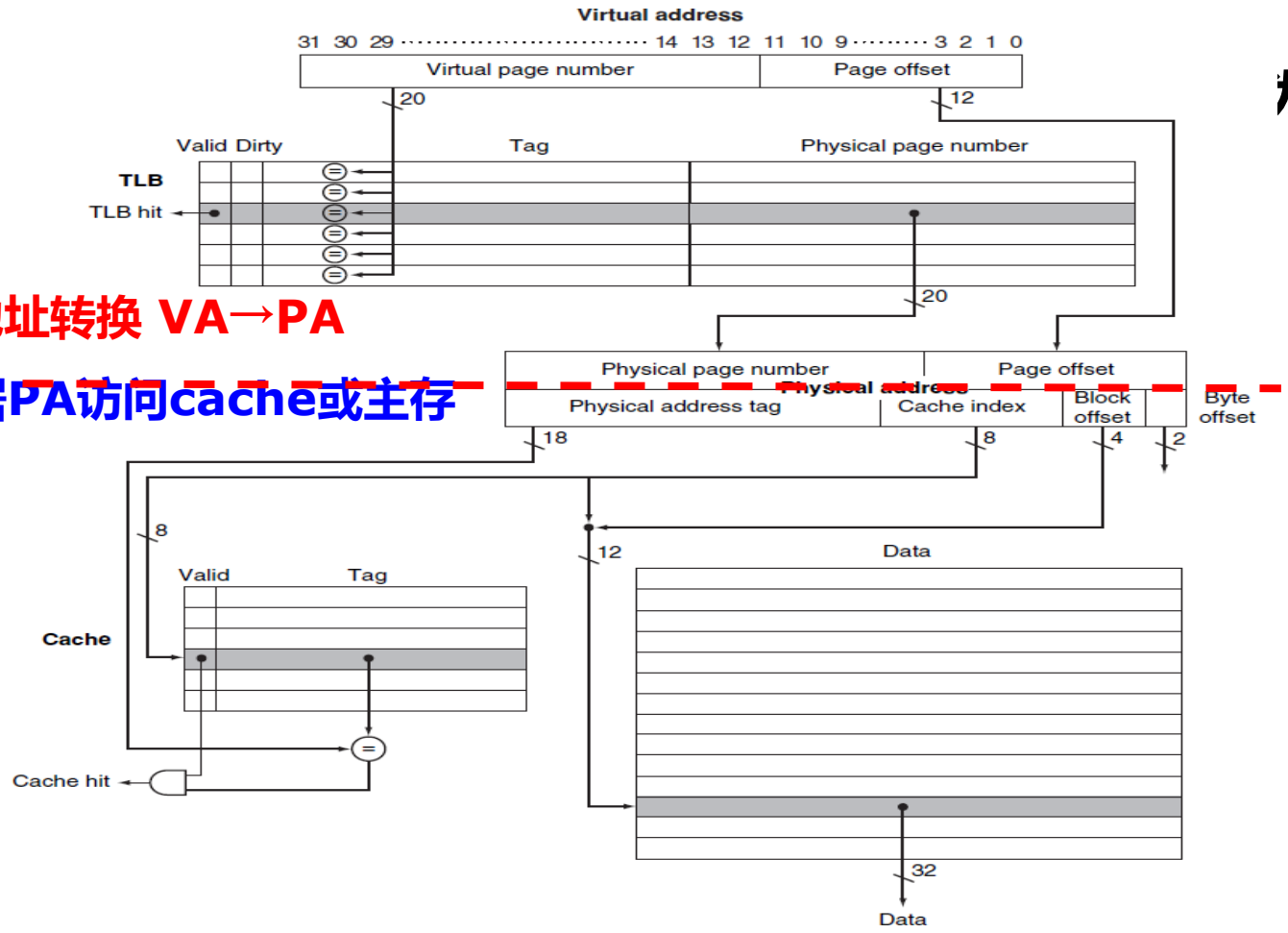
最坏的情况是: miss、miss、miss 需访问磁盘、并访存至少2次

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

VM、TLB和Cache组成的层次结构处理



快表（TLB）



地址转换 VA→PA

根据PA访问cache或主存

为多少?

快表（TLB）

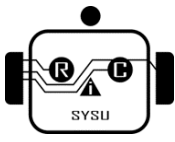
TLB的设计问题

- 多进程的频繁切换为 TLB制造了不小麻烦。在现代处理器系统中，每个进程都使用各自独立的虚拟地址空间，进程切换意味着虚拟地址空间的切换，也意味着 TLB 的刷新。进程频繁切换导致TLB需要频繁预热，这个开销是难以接受的。
- 多线程处理器的引入再次增加了TLB设计的负担。在多线程处理器中存在多个逻辑CPU，几个逻辑CPU共享同一条流水线，而使用着不同的虚拟地址空间，需要用 TLB进行虚实地址转换。多数情况下，这些逻辑CPU共享 TLB，对进程切换带来的 TLB刷新操作是Zero-Tolerance。需要将Logical Processor ID也加入到TLB Entry中。

快表（TLB）

TLB的设计问题

- TLB的Entry 结构日趋稳定，却迎来了更严峻的挑战。因为主存膨胀速度越发不可控制，程序对主存容量需求越来越高。主存容量迅速增长，使得TLB 的Coverage Rate (TLB所能管理的存储器空间与主存容量的比值)在逐年降低，直接导致TLB Miss Rate 的不断增大。



快表（TLB）

页式虚拟存储器的特点

- **优点**：实现简单，开销少。因为只有进程的最后一个零头（内部碎片）不能利用，故浪费很小
- **缺点**：由于页不是逻辑上独立的实体，可能会出现如“一条指令跨页”等情况（CISC），使处理、管理、保护和共享都不方便

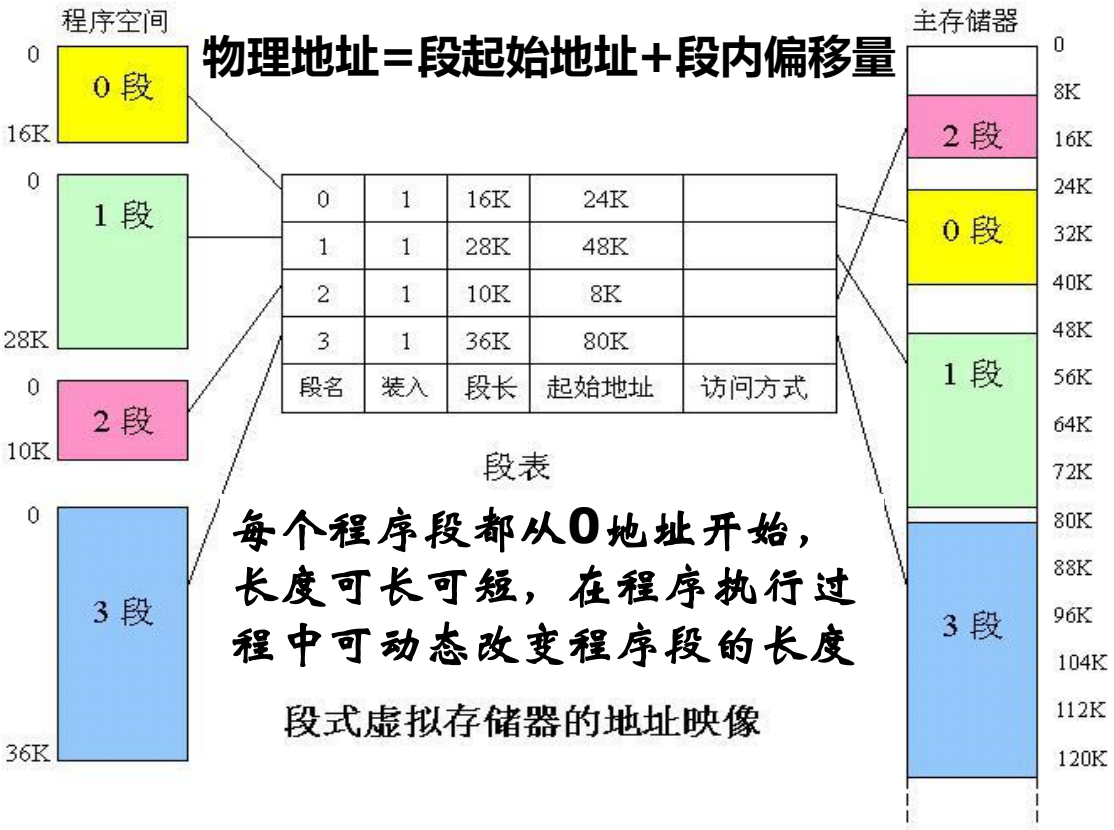
快表（TLB）

段式虚拟存储器的实现

- 程序员或OS将程序模块或数据模块分配给不同主存段，一个大程序由多个代码段和多个数据段构成，按照程序的逻辑结构划分成多个相对独立的部分。(如过程、子程序、数据表、阵列等)
- 段通常带有段名或基地址，便于编写程序、编译器优化和操作系统调度管理
- 段可作为独立逻辑单位被其他程序调用，以形成段间连接，产生规模更大的程序
- 分段系统将主存空间按实际程序中的段来划分，每个段在主存中的位置记录在段表中，并附以“段长”项说明
- 段表本身也是主存中的一个可再定位段
- 段本身是程序的逻辑结构所决定的一些独立部分，所以分段对程序员是不透明的(而分页对程序员却是透明的)

5.4.3 快表（TLB）

段式虚拟存储器的地址映象

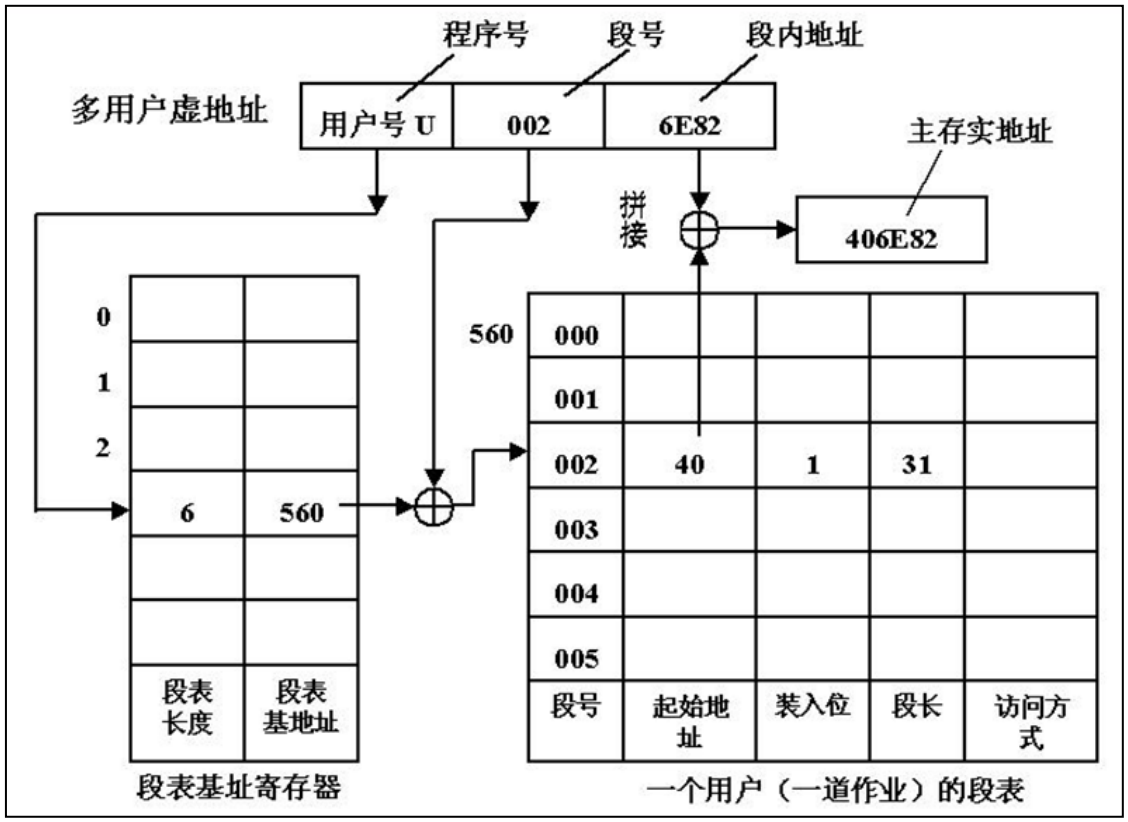


Faults（异常情况）：

- **缺段(段不存在)：装入位= 0**
- **地址出界： 偏移量超出最大段长**
- **保护违例：访问操作不符合访问方式指定类型**

5.4.3 快表（TLB）

段式虚拟存储器的地址变换



快表（TLB）

段式虚拟存储器的特点

- **优点：** 段的分界与程序的自然分界对应，故段具有逻辑独立性，易于编译、管理、修改和保护，便于多道程序共享；某些类型的段(堆栈、队列)具有动态可变长度，允许自由调度以有效利用主存空间
- **缺点：** 段长各不相同，起、终点不定，变化很大，给主存分配带来麻烦，且易在段间留下许多空余的零碎空间不好利用，造成浪费(如：长段调出后，调进的短段会造成碎片)

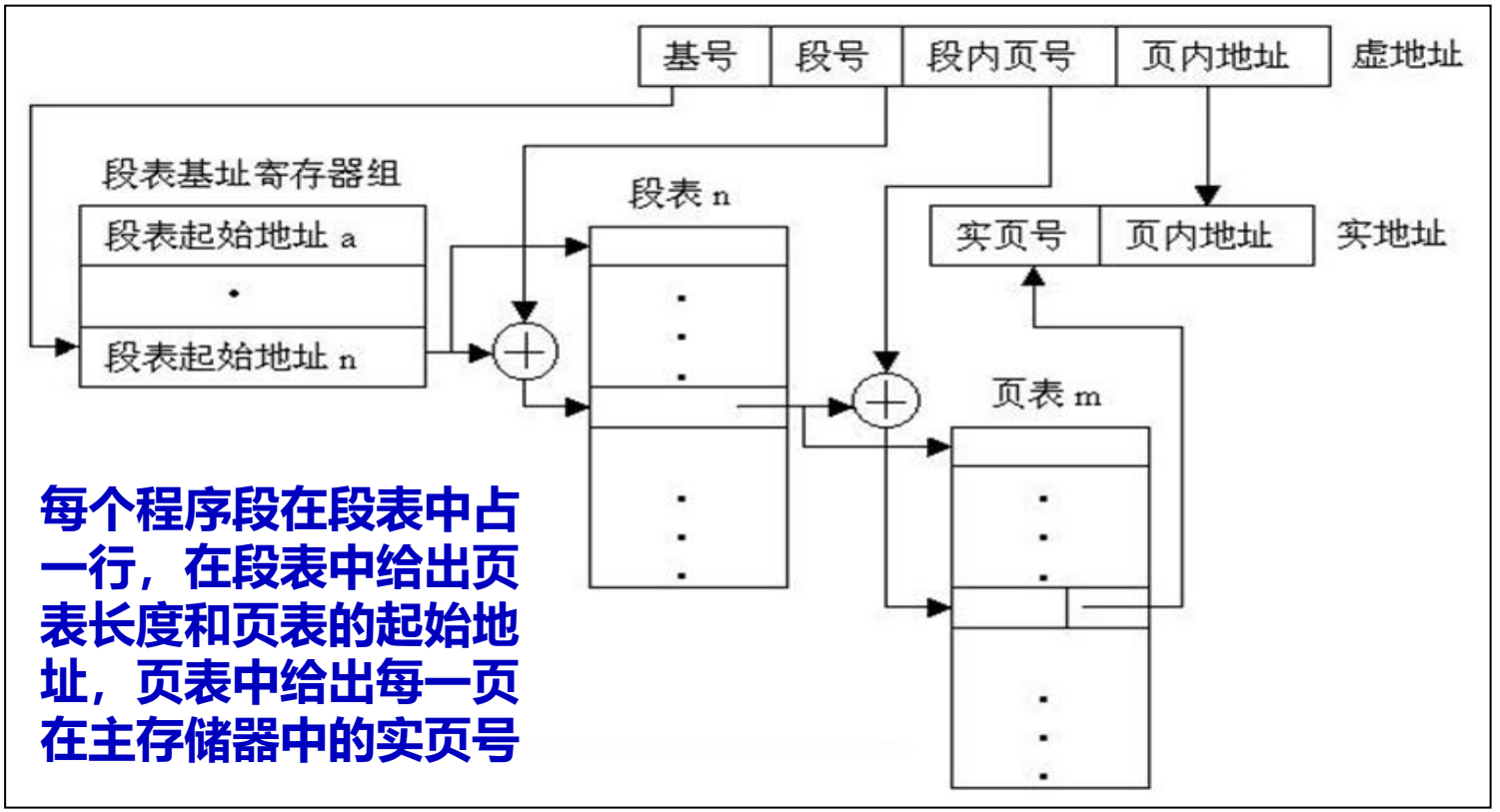
快表（TLB）

段页式虚拟存储器基本思想

- 段式和页式有机结合。程序按模块分段，段内再分页，进入主存仍以页为基本单位
- 逻辑地址由段地址、页地址和页内偏移量三个字段构成
- 用段表和页表(每段一个)进行两级定位管理
- 根据段地址到段表中查阅与该段相应的页表指针，再转向页表，然后根据页地址从页表中查到该页在主存中的页框地址，由此访问到页内某数据

快表（TLB）

段页式虚拟存储器的地址变换



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

快表（TLB）

1. 某计算机系统有一个TLB和一个L1 Data Cache。该系统按字节编址，虚拟地址16位，物理地址12位，页大小为128B，TLB采用四路组相联方式，共16个页表项， L1 Data Cache采用直接映射方式，块大小为4B，共16行。系统运行到某一时刻时，TLB、页表和L1 Data Cache中部分内容如图示。请问:

(1)虚拟地址中哪几位表示虚拟页号？哪几位表示页内偏移？虚拟页号中哪几位表示TLB标记？哪几位表示TLB索引?

(2)物理地址中哪几位表示物理页号？哪几位表示页内偏移?

(3)主存物理地址是如何划分标记字段、行索引字段和块地址字段的?

(4)CPU从地址067AH中取出的值是多少？说明CPU读取地址067AH中内容的过程。

快表（TLB）

虚拟地址067AH

=0000011 00 1111010B

(b) 部分页表
(开始16项)

虚页号	页框号	有效位
00	08	1
01	03	1
02	14	1
03	02	1
04		0
05	16	1
06		0
07	07	1
08	13	1
09	17	1
0A	09	1
0B		0
0C	19	1
0D		0
0E	11	1
0F	0D	1

组号	标记	页框号	有效位	标记	页框号	有效位	标记	页框号	有效位	标记	页框号	有效位
0	03		0	09	0D	1	00		0	07	02	1
1	03	2D	1	02		0	04		0	0A		0
2	02		0	08		0	06		0	03		0
3	07		0	63	0D	1	0A	34	1	72		0

(a) TLB：四路组相联，四组，16个页表项

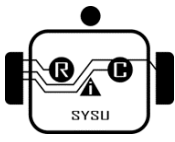
物理地址0CFAH =11001 1111010B

快表（TLB）

(C) L1 Data Cache:
直接映射方式，块大小为4B，共16行

物理地址 =11001 1111010B

行索引	标记	有效位	字节3	字节2	字节1	字节0
00	19	0	12	56	C9	AC
01	15	1				
02	1B	0	03	45	12	CD
03	36	0				
04	32	1	23	34	C2	2A
05	0D	1	46	67	23	3D
06		0				
07	16	1	12	54	65	DC
08	24	1	23	62	12	3A
09	2D	0				
0A	2D	1	43	62	23	C3
0B		0				
0C	12	1	76	83	21	35
0D	16	1	A3	F4	23	11
0E	33	1	2D	4A	45	55
0F	14	0				



小结

- 虚拟存储器是磁盘和主存之间的缓存管理机制，而不是一种物理存储器
- 引入虚拟存储器，使程序员可以在一个极大的存储空间编写程序，无需知道运行程序的物理存储器究竟有多大
- 虚拟存储器采用“按需调页”技术，把一部分程序调到主存，一部分存放在磁盘上
- 交换的块(称为页)比Cache-Memory层次的块要大得多
- 采用全相联映射，通过页表实现逻辑地址和物理地址转换，由硬件(MMU)实现
- 缺页处理由OS完成(cache miss处理由硬件实现)
- 采用Write Back写策略
- 页表中记录装入位、访问权限、使用情况、修改位、磁盘地址或页框号
- 经常使用的页表项放到一个特殊的Cache中，称之为快表TLB
- 虚拟存储器有三种管理模式：页式、段式、段页式

联系方式

- Acknowledgements:
- This slides contains materials from following lectures:
  - Computer Architecture (ETH，NUDT，USTC，SYSU)

- Research Area：
  - 计算机视觉与机器人应用计算加速，
  - 人工智能和深度学习芯片及智能计算机

- Contact：
  - 中山大学计算机学院
  - 管理学院D101（图书馆右侧）
  - 机器人与智能计算实验室
  - cheng83@mail.sysu.edu.cn